



Rational[®]
the software development company

Phân tích thiết kế với UML

Bài 5: Kiến trúc hệ thống

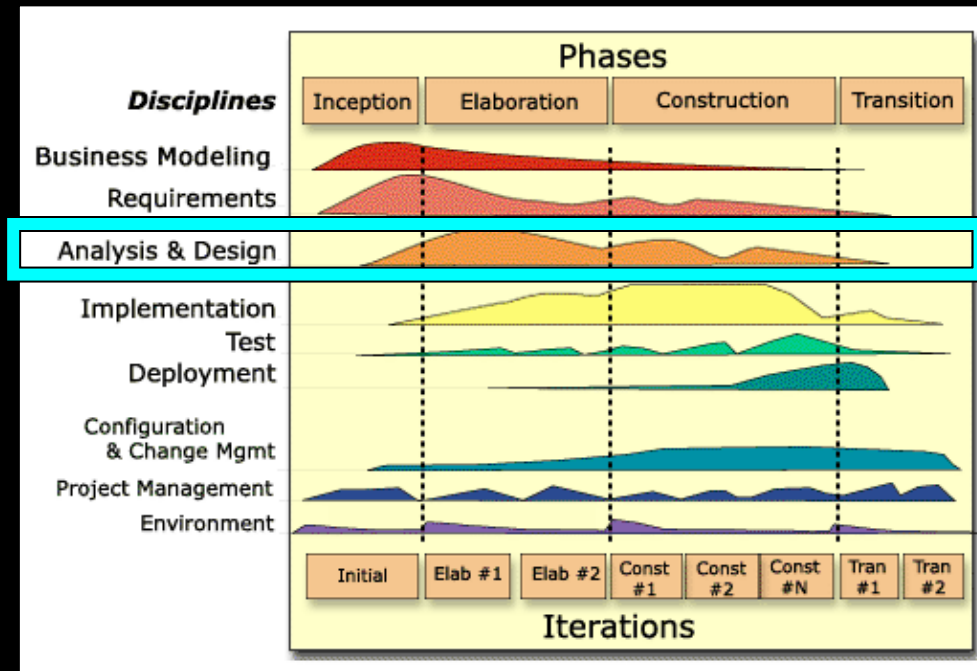
Nhắc lại kiến thức

- ♦ Mục đích của Phân tích và Thiết kế?
- ♦ Đầu vào và Đầu ra?
- ♦ Sự khác biệt giữa Phân tích và Thiết kế?
- ♦ Tiếp cận usecase-driven trong PT&TK
- ♦ Các góc nhìn kiến trúc – mô hình 4+1

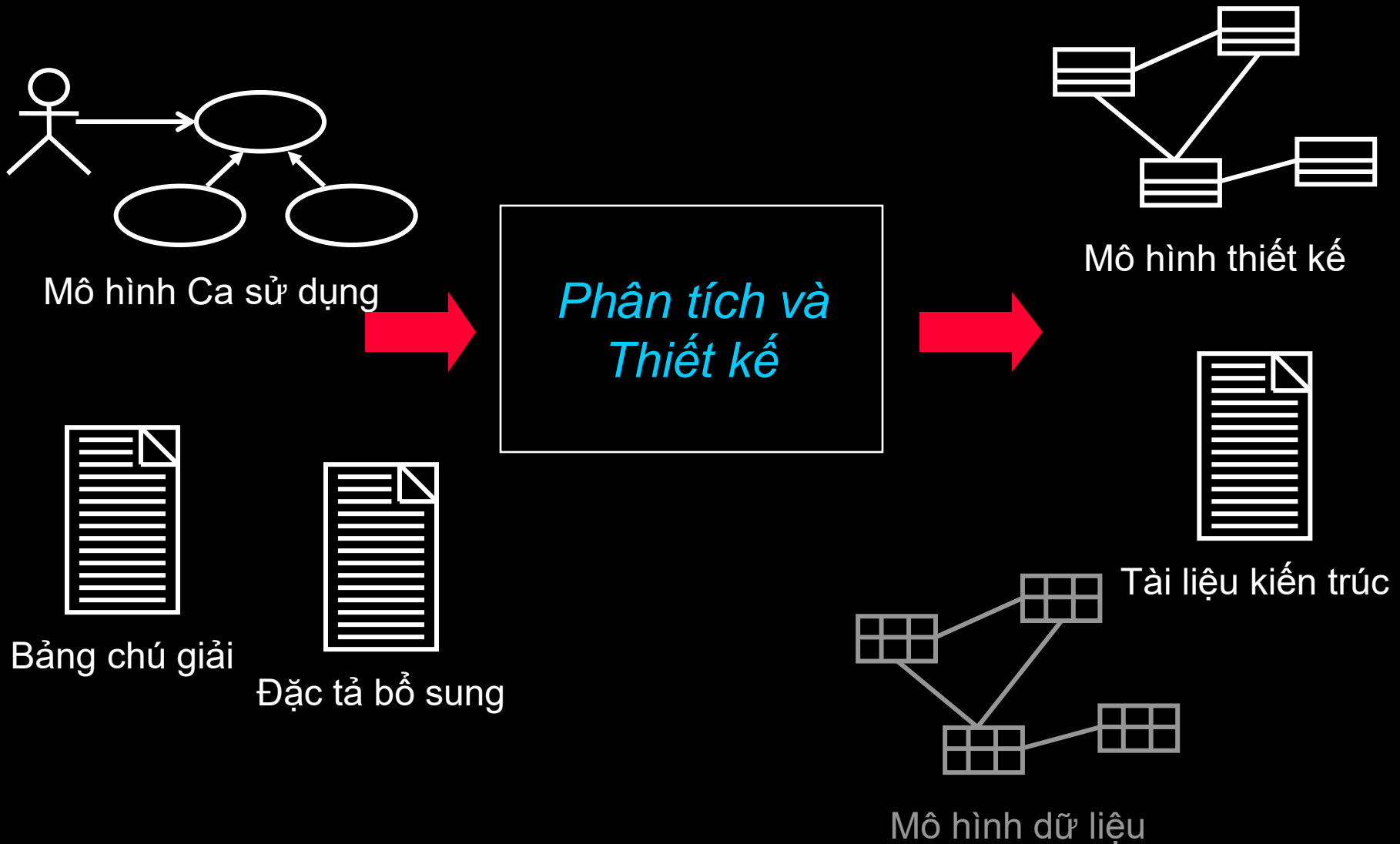
Mục đích của PT&TK

Mục đích của phân tích và thiết kế:

- Chuyển yêu cầu phần mềm thành một bản thiết kế.
- Đúc rút một bản kiến trúc tốt.
- Làm cho bản thiết kế thích ứng với môi trường triển khai và có hiệu năng tốt nhất.



Đầu vào và đầu ra của PT&TK



Phân biệt Phân tích với Thiết kế

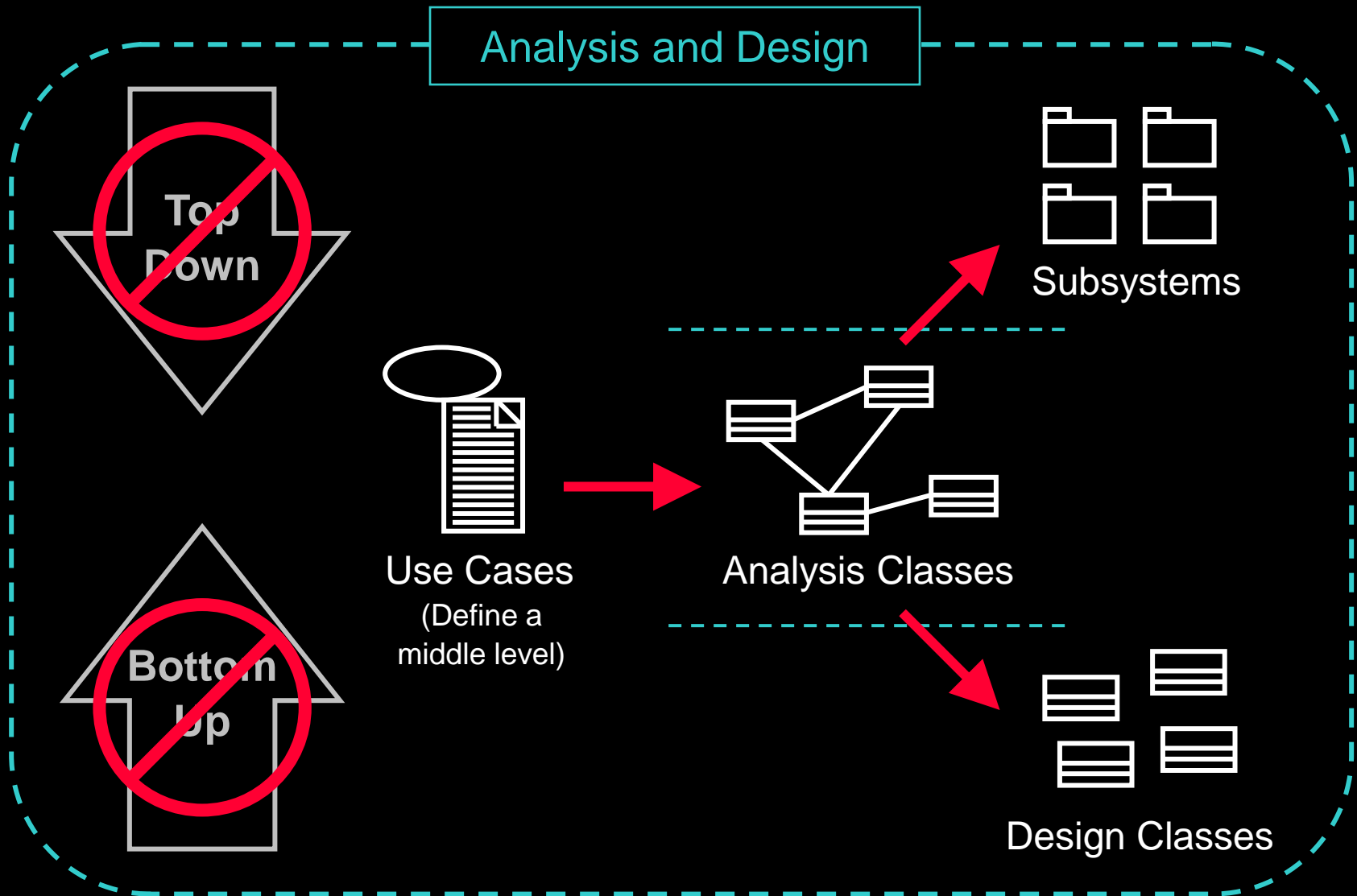
◆ Phân tích

- Tập trung vào việc tìm hiểu vấn đề
- Bản thiết kế lý tưởng
- Nằm bắt hành vi
- Nằm bắt cấu trúc hệ thống
- Đáp ứng các yêu cầu chức năng
- Mô hình cỡ nhỏ

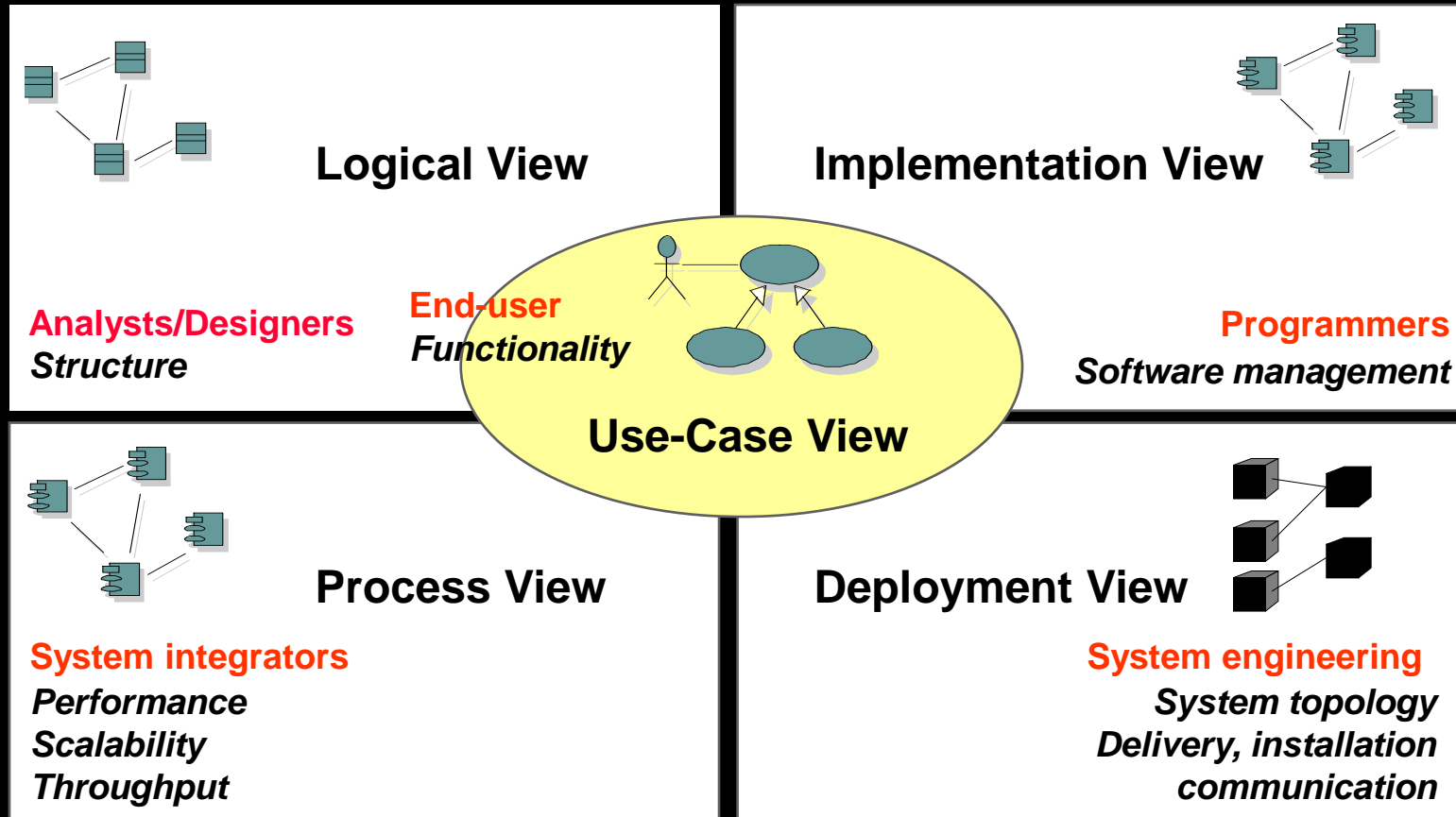
◆ Thiết kế

- Tập trung vào việc tìm hiểu giải pháp
- Xác định thao tác và thuộc tính
- Đảm bảo hiệu năng
- Xác định vòng đời đối tượng
- Đáp ứng các yêu cầu phi chức năng
- Mô hình cỡ lớn

Tiếp cận Dẫn dắt bởi ca sử dụng (usecase-driven)



Kiến trúc phần mềm: Mô hình “4+1 View”



Nội dung

- ◆ Kiến trúc là gì?
- ◆ Giải thích vai trò của kiến trúc trong vòng đời phát triển phần mềm.
- ◆ Giới thiệu khái niệm:
 - Mẫu kiến trúc – architectural pattern
 - Kỹ thuật – mechanism
- ◆ Các góc nhìn
 - Góc nhìn logic – logical view
 - Góc nhìn thực hiện – implementation view
 - Góc nhìn tiến trình – process view
 - Góc nhìn triển khai/bố trí – deployment

Kiến trúc phần mềm

- ♦ Kiến trúc phần mềm bao gồm một tập các quyết định về tổ chức của phần mềm.
 - Sự lựa chọn các yếu tố cấu trúc và giao diện của chúng để lắp ghép thành hệ thống
 - Sự cộng tác của các yếu tố cấu trúc
 - Sự lắp ghép, tích hợp các yếu tố cấu trúc và hành vi để tạo nên các hệ thống con
 - Phong cách kiến trúc

*Grady Booch, Philippe Kruchten, Rich Reitman, Kurt Bittner; Rational
(derived from Mary Shaw)*

Kiến trúc phần mềm

- ♦ Kiến trúc phần mềm cung cấp các góc nhìn khác nhau về một hệ thống phần mềm.
 - Góc nhìn logic:
 - Hệ thống được cấu thành từ đối tượng thuộc những lớp gì?
 - Sự cộng tác của các đối tượng này như thế nào?
 - Góc nhìn thực hiện
 - Các lớp thuộc hệ thống được xây dựng từ những thành phần mã nguồn gì?
 - Chúng được biên dịch/đóng gói thành các thành phần thực thi/thành phần bố trí gì?

Kiến trúc phần mềm

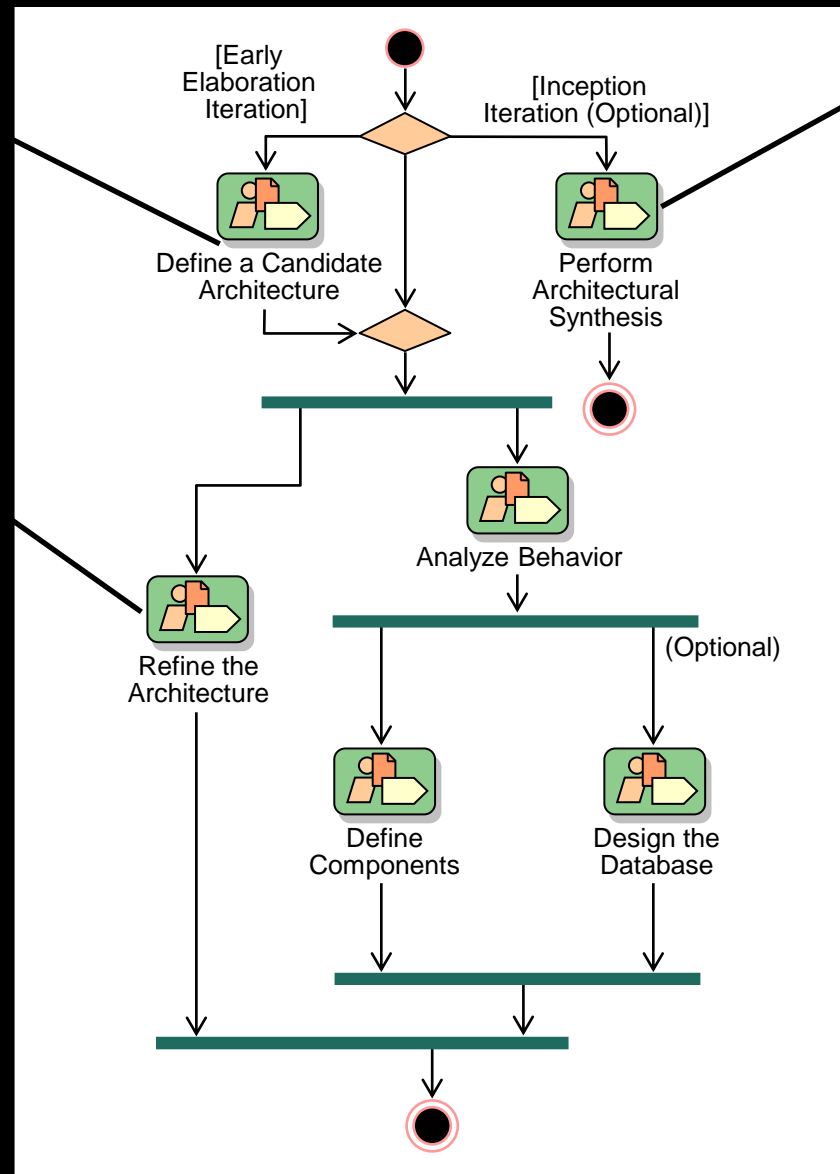
- ♦ Kiến trúc phần mềm cung cấp các góc nhìn khác nhau về một hệ thống phần mềm.
 - Góc nhìn tiến trình:
 - Hệ thống được thực hiện trên các luồng/tiến trình gì?
 - Đối tượng thuộc những lớp nào hoạt động trên từng luồng/tiến trình
 - Góc nhìn triển khai
 - Hệ thống phần mềm được triển khai trên hạ tầng phần cứng thể nào?
 - Các thành phần bố trí được đặt trên các nút tính toán như thế nào?

Kiến trúc trong vòng đời phát triển phần mềm

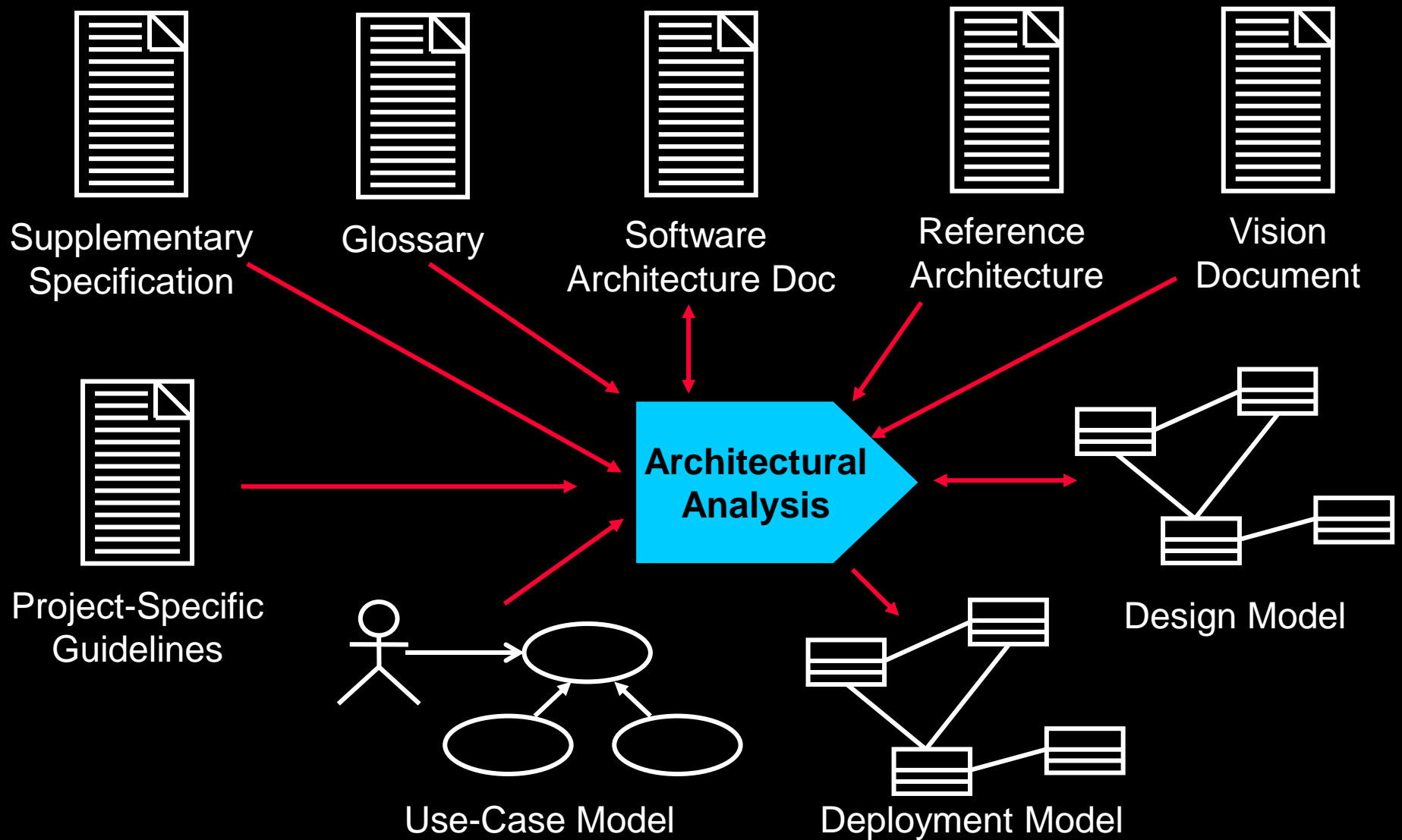
Trong bước lập đầu tiên của pha Elaboration, kiến trúc sư xác định một khung kiến trúc cho hệ thống (từ việc tổng hợp các mẫu kiến trúc sẵn có)
Bước này thường được gọi là: **Phân tích kiến trúc**

Trong trình phân tích thiết kế, khung kiến trúc sẽ được bổ sung, mở rộng với các yếu tố thiết kế để tạo thành bản thiết kế kiến trúc hoàn chỉnh

Trong pha đầu, kiến trúc sư thu thập, tổng hợp những mẫu thiết kế kiến trúc sẵn có



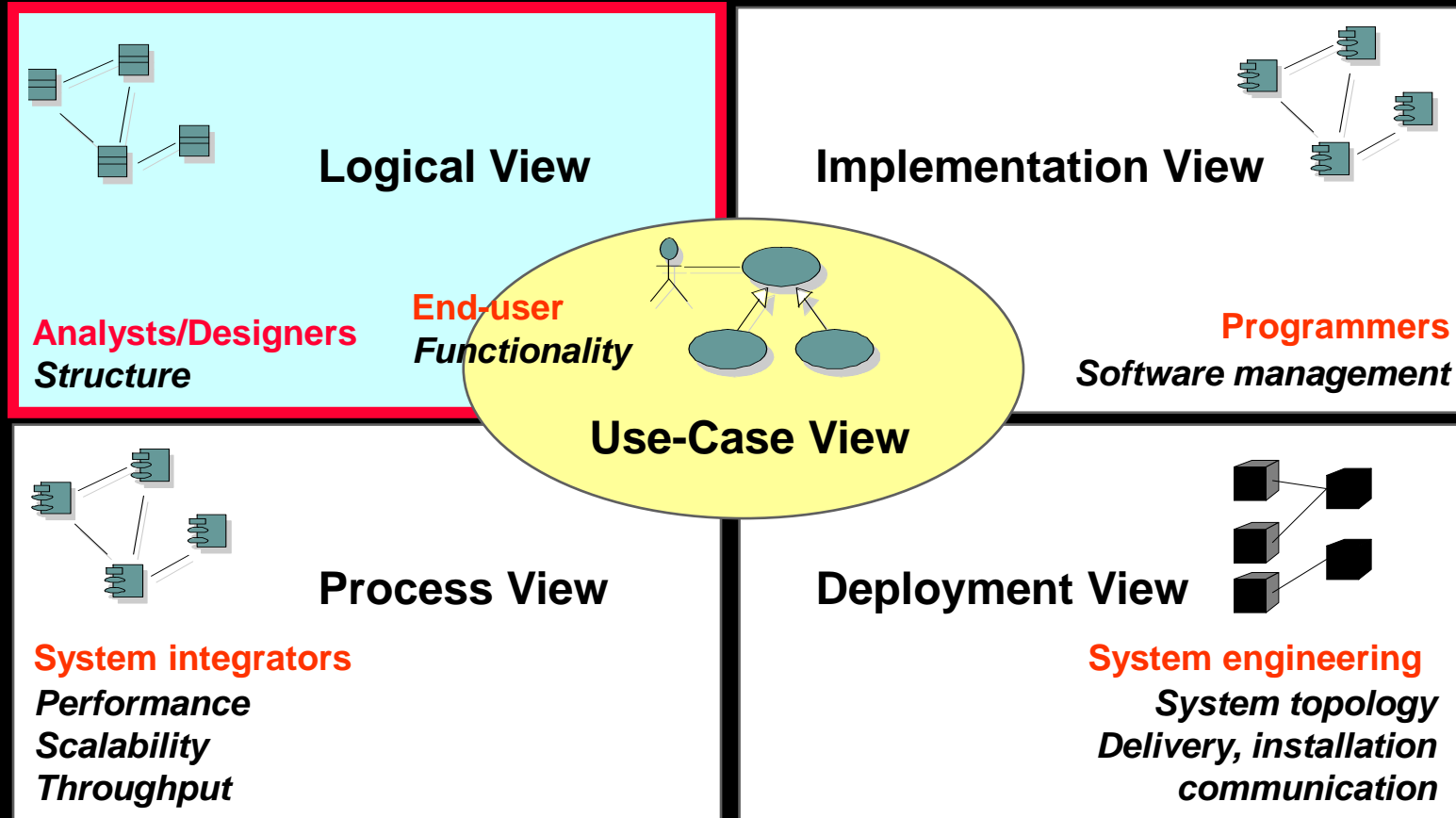
Phân tích kiến trúc



Tổ chức mức cao của hệ thống (góc nhìn logic)

- ★ ♦ Để có được góc nhìn toàn cảnh của hệ thống
 - Hệ thống được cấu thành từ các hệ thống con nào
 - Các hệ thống con có mối quan hệ với nhau như thế nào
- ♦ Tổ chức mức cao thường được trình bày dưới dạng biểu đồ gói dưới khung nhìn logic

“4+1 View” Model

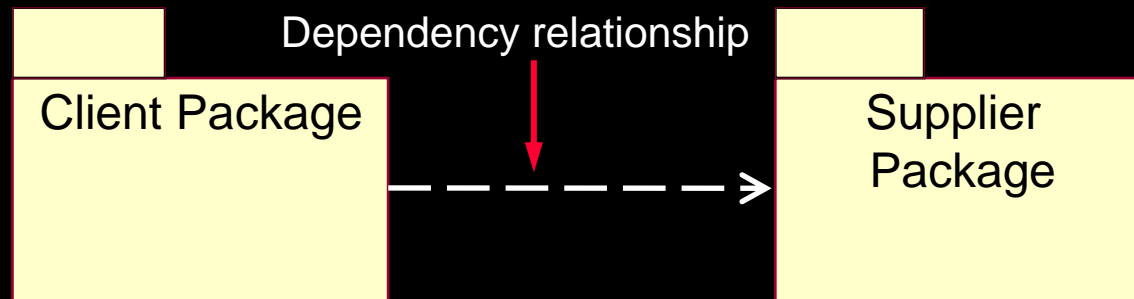


Ôn lại khái niệm gói

- ♦ Gói là kỹ thuật gom nhiều yếu tố liên quan lại thành nhóm.
- ♦ Gói có thể chứa các gói nhỏ hơn.



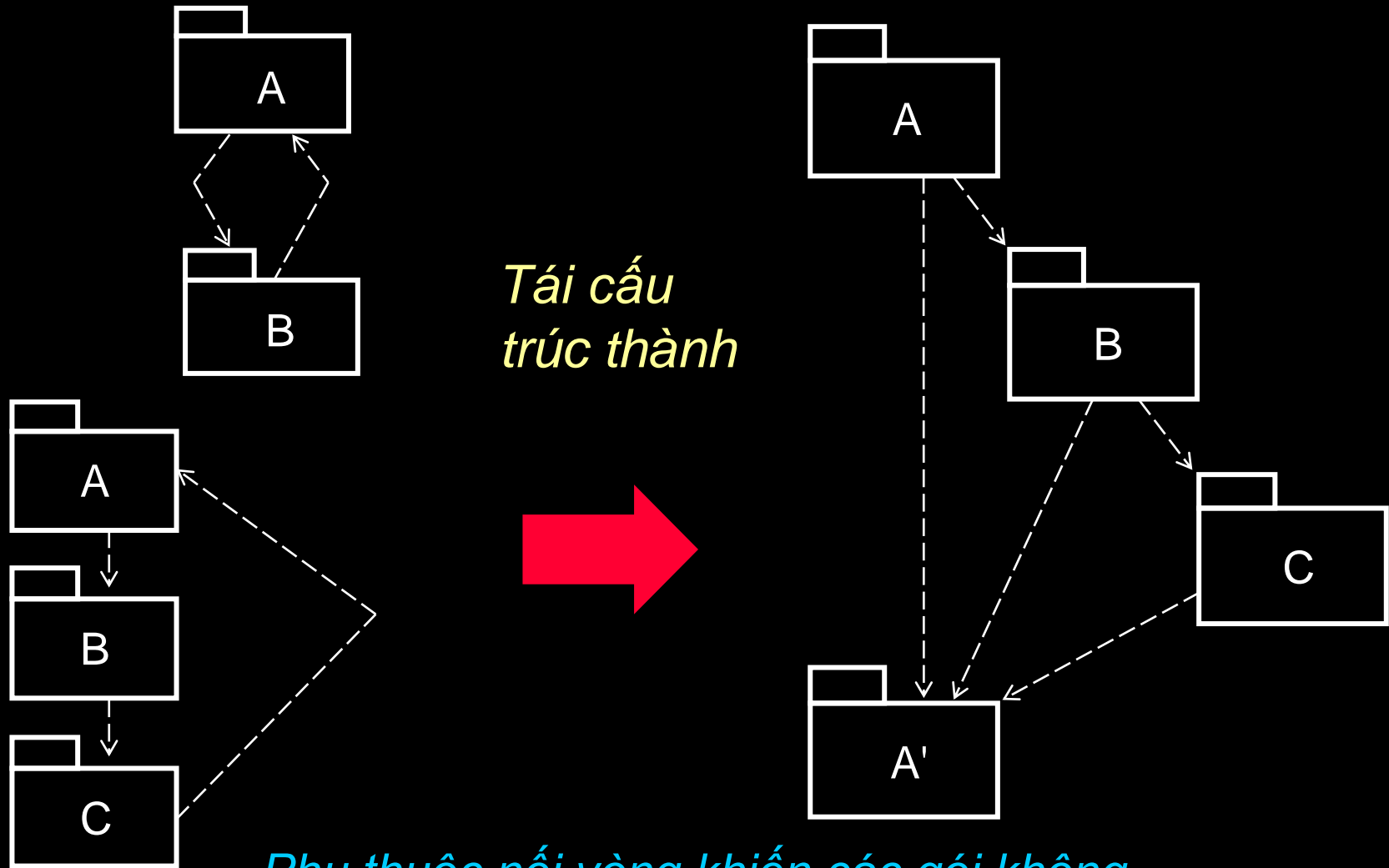
Quan hệ phụ thuộc giữa các gói



♦ Quan hệ phụ thuộc

- Sự thay đổi của gói Supplier có thể ảnh hưởng tới gói Client.
- Gói Client không thể tái sử dụng một cách độc lập bởi nó phụ thuộc vào gói Supplier.

Hạn chế phụ thuộc nối vòng giữa các gói



Phụ thuộc nối vòng khiến các gói không thể tái sử dụng một cách độc lập

Mẫu kiến trúc

- ♦ Tổ chức mức cao của các phần mềm thường có khung giống nhau
- ♦ Trong quá trình phân tích, kiến trúc sư thường lựa chọn khung kiến trúc cho hệ thống từ việc tổng hợp từ các kiến trúc sẵn có.
- ♦ Mẫu kiến trúc là một kỹ thuật khái quát hóa các kiến trúc có tổ chức tương tự nhau
 - Layers
 - Model-view-controller (M-V-C)
 - Pipes and filters
 - Blackboard

Khái niệm Mẫu và Khung

♦ Mẫu - Pattern

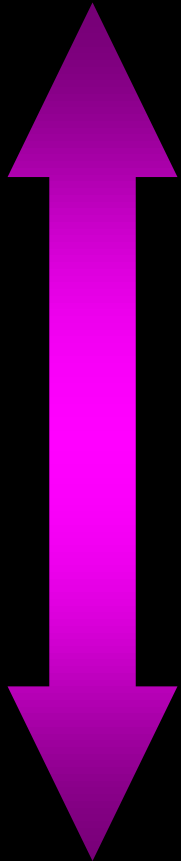
- Cung cấp một giải pháp chung cho một lớp vấn đề tương tự

♦ Khung - Framework

- Định nghĩa một nền tảng cho các giải pháp, từng giải pháp cụ thể có thể được phát triển, mở rộng từ nền tảng này

Mẫu kiến trúc phân tầng - Layering

**Specific
functionality**



**General
functionality**

Application Subsystems

Distinct application subsystems that make up an application — contains the value adding software developed by the organization.

Business-Specific

Business specific — contains a number of reusable subsystems specific to the type of business.

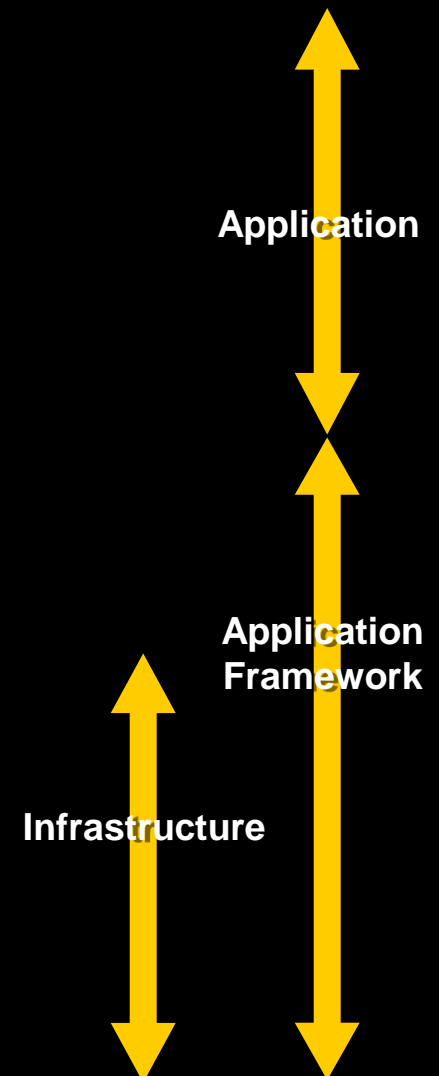
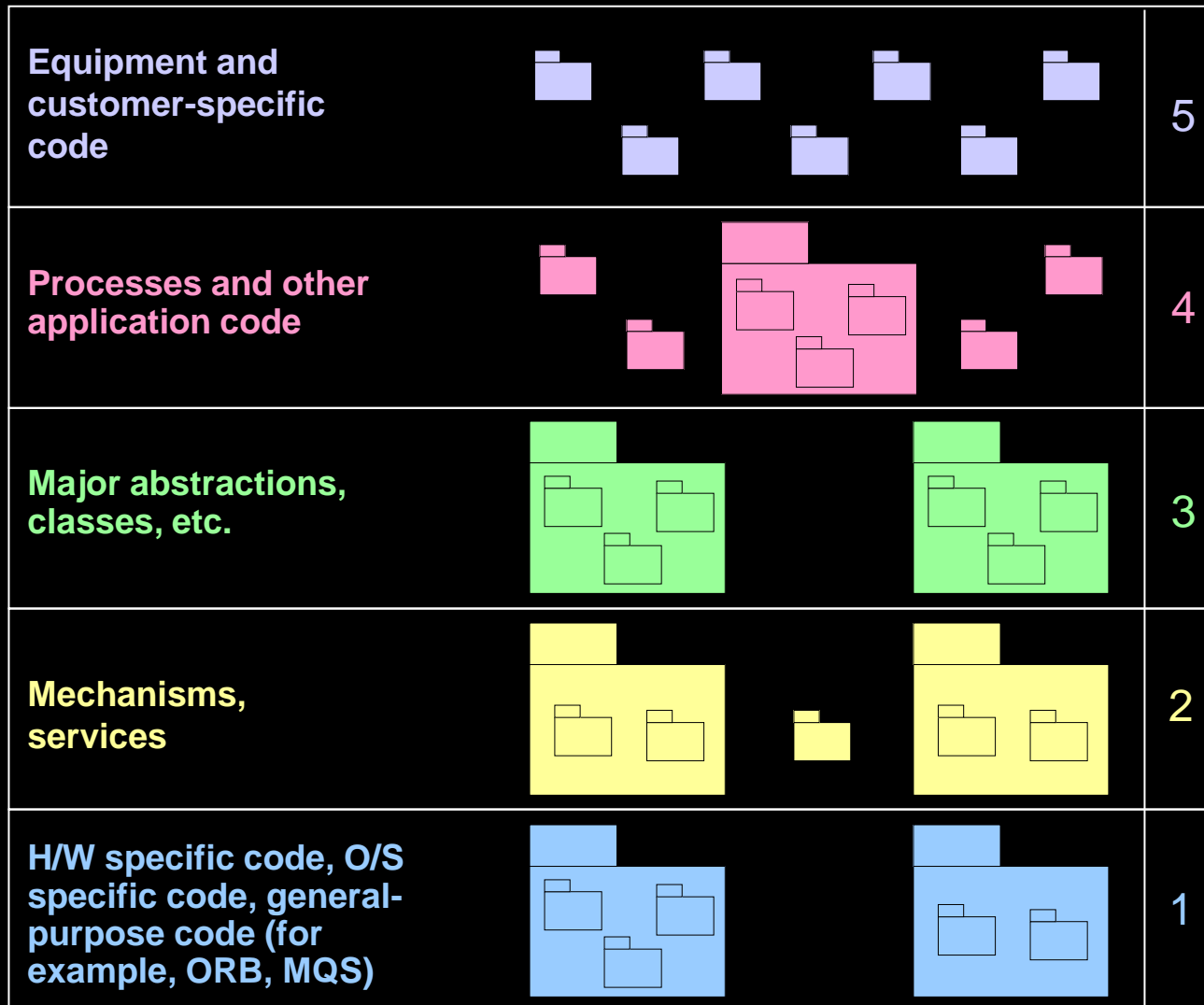
Middleware

Middleware — offers subsystems for utility classes and platform-independent services for distributed object computing in heterogeneous environments and so on.

System Software

System software — contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers, and so on.

Mẫu kiến trúc phân tầng - Layered

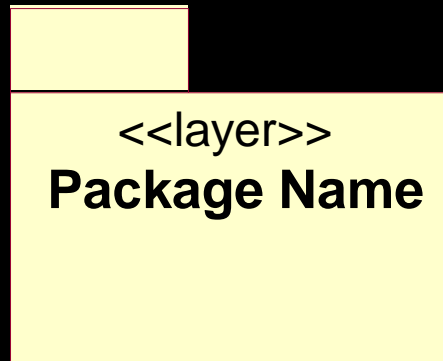


Mẫu kiến trúc phân tầng

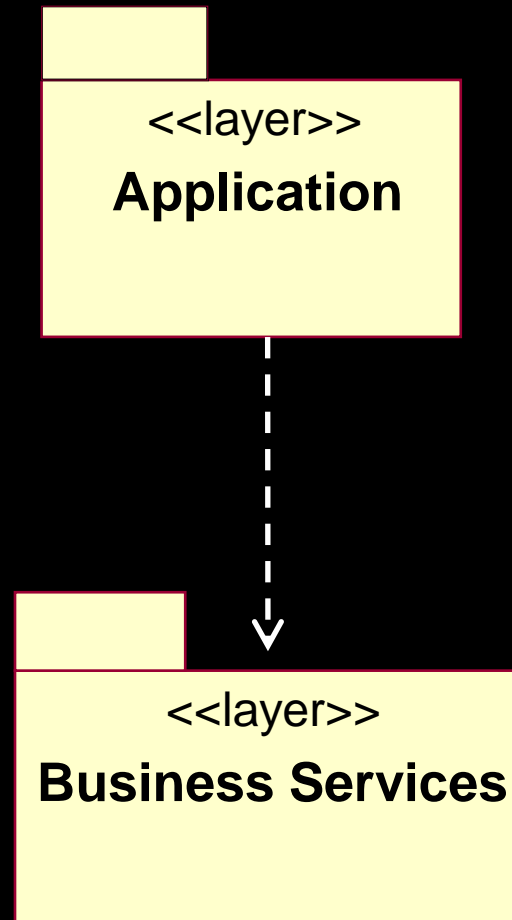
- ◆ Phân tách các mức độ trừu tượng
 - Các yếu tố có cùng mức độ trừu tượng được đặt chung trong một tầng
- ◆ Phân tách các mối quan tâm (separation of concerns)
 - Nhóm các yếu tố giống nhau lại
 - Phân tách các yếu tố khác nhau
- ◆ Resiliency
 - Ràng buộc lỏng
 - Đóng gói các thay đổi
 - Phân tách các yếu tố thuộc các nhóm khác nhau để có thể dễ thay đổi

Ký pháp UML

- ♦ Mỗi tầng có thể được biểu diễn bằng một gói với stereotype <<layer>>



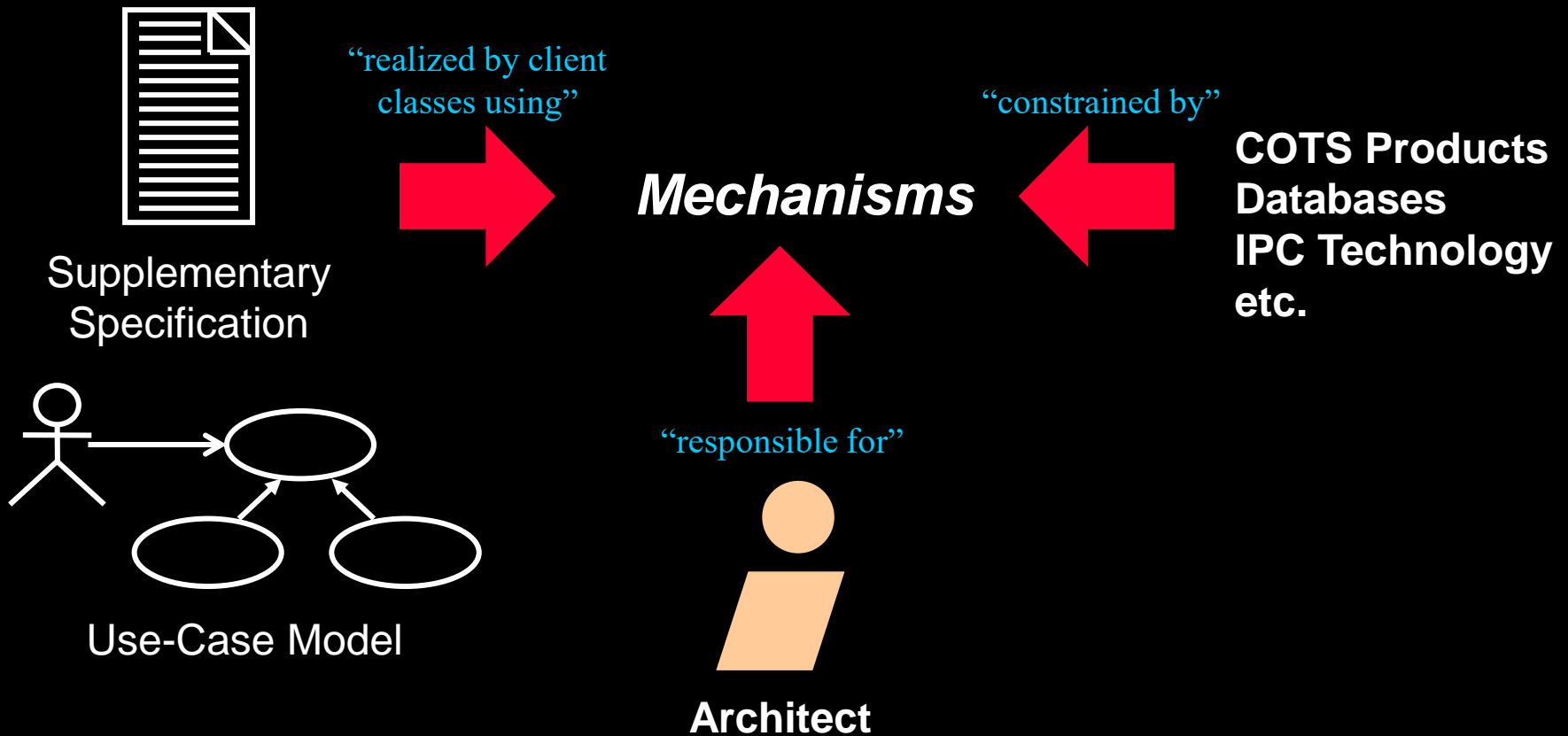
Ví dụ về một tổ chức phân tầng



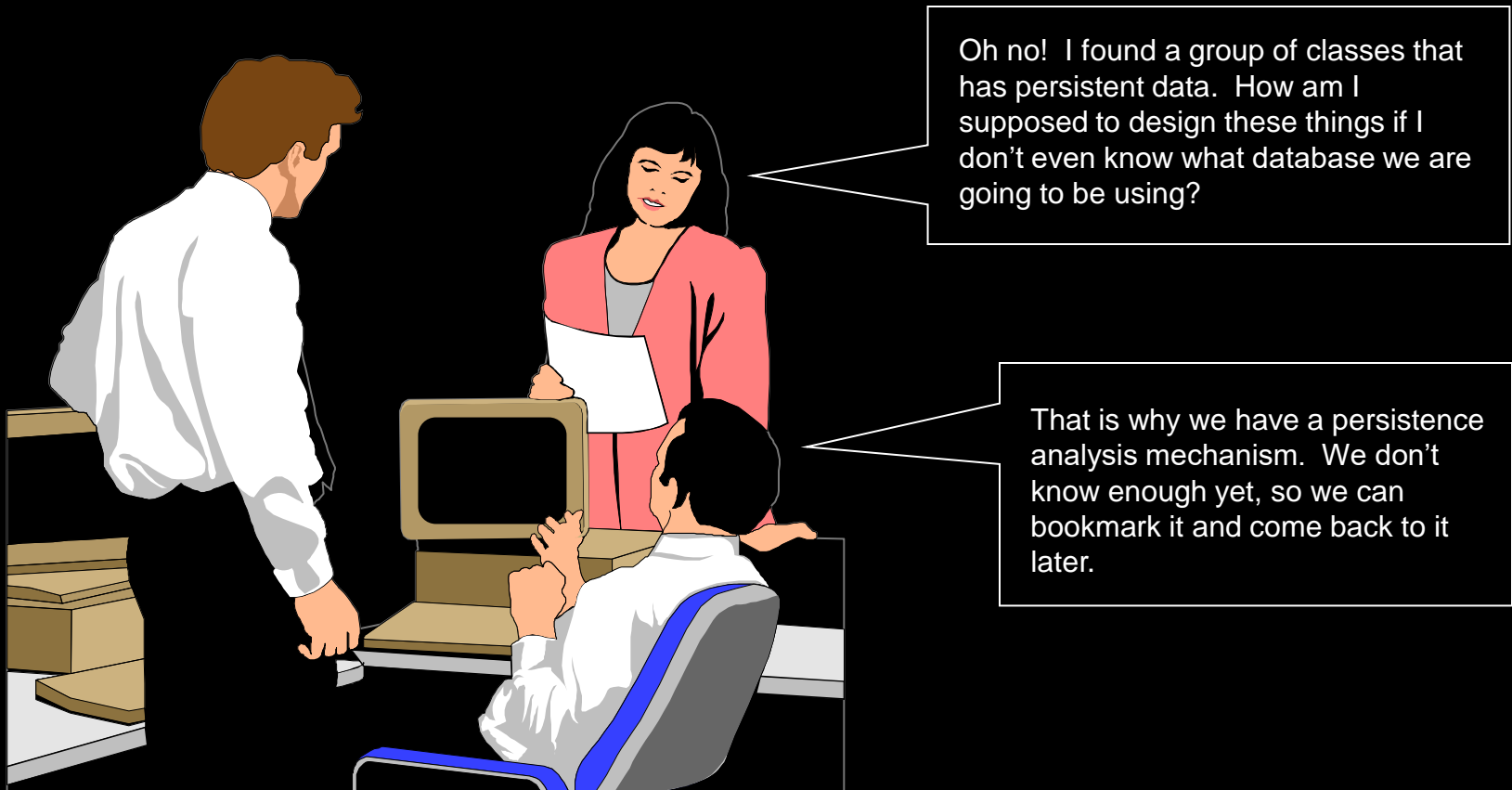
Kỹ thuật - Mechanisms

Yêu cầu chức năng

Môi trường thực hiện



Kỹ thuật phân tích



Analysis mechanisms are used during analysis to reduce the complexity of analysis and to improve its consistency by providing designers with a shorthand representation for complex behavior.

Ví dụ về các kỹ thuật phân tích

- ◆ Persistency
- ◆ Communication (IPC and RPC)
- ◆ Message routing
- ◆ Distribution
- ◆ Transaction management
- ◆ Process control and synchronization (resource contention)
- ◆ Information exchange, format conversion
- ◆ Security
- ◆ Error detection / handling / reporting
- ◆ Redundancy
- ◆ Legacy Interface

Đặc tả chi tiết các kỹ thuật phân tích

◆ Persistency mechanism

- Granularity
- Volume
- Duration
- Access mechanism
- Access frequency (creation/deletion, update, read)
- Reliability

◆ Inter-process Communication mechanism

- Latency
- Synchronicity
- Message size
- Protocol

Đặc tả chi tiết các kỹ thuật phân tích

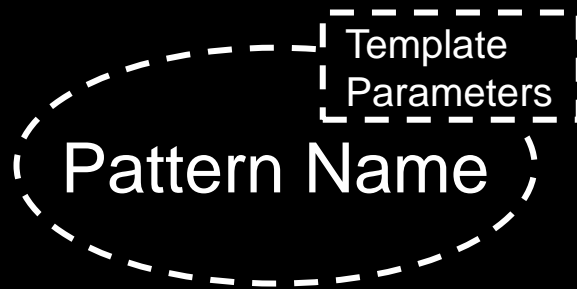
- ◆ Legacy interface mechanism
 - Latency
 - Duration
 - Access mechanism
 - Access frequency
- ◆ Security mechanism
 - Data granularity
 - User granularity
 - Security rules
 - Privilege types
- ◆ Others

Kỹ thuật thiết kế - design mechanism

- ♦ Kỹ thuật phân tích chỉ ra những kỹ thuật nào cần được áp dụng và những ràng buộc chi tiết cho các kỹ thuật này
 - Được thực hiện trong bước phân tích
- ♦ Kỹ thuật thiết kế mô tả rõ mô hình thiết kế cho kỹ thuật phân tích
 - Thường được mô tả dưới dạng mẫu thiết kế để có thể dễ dàng tái sử dụng

Mẫu thiết kế

- ♦ Mẫu thiết kế mô tả một giải pháp có tính lặp đi lặp lại cho một vấn đề thiết kế nào đó
- ♦ Erich Gamma et al. 1994. *Design Patterns—Elements of Reusable Object-Oriented Software*

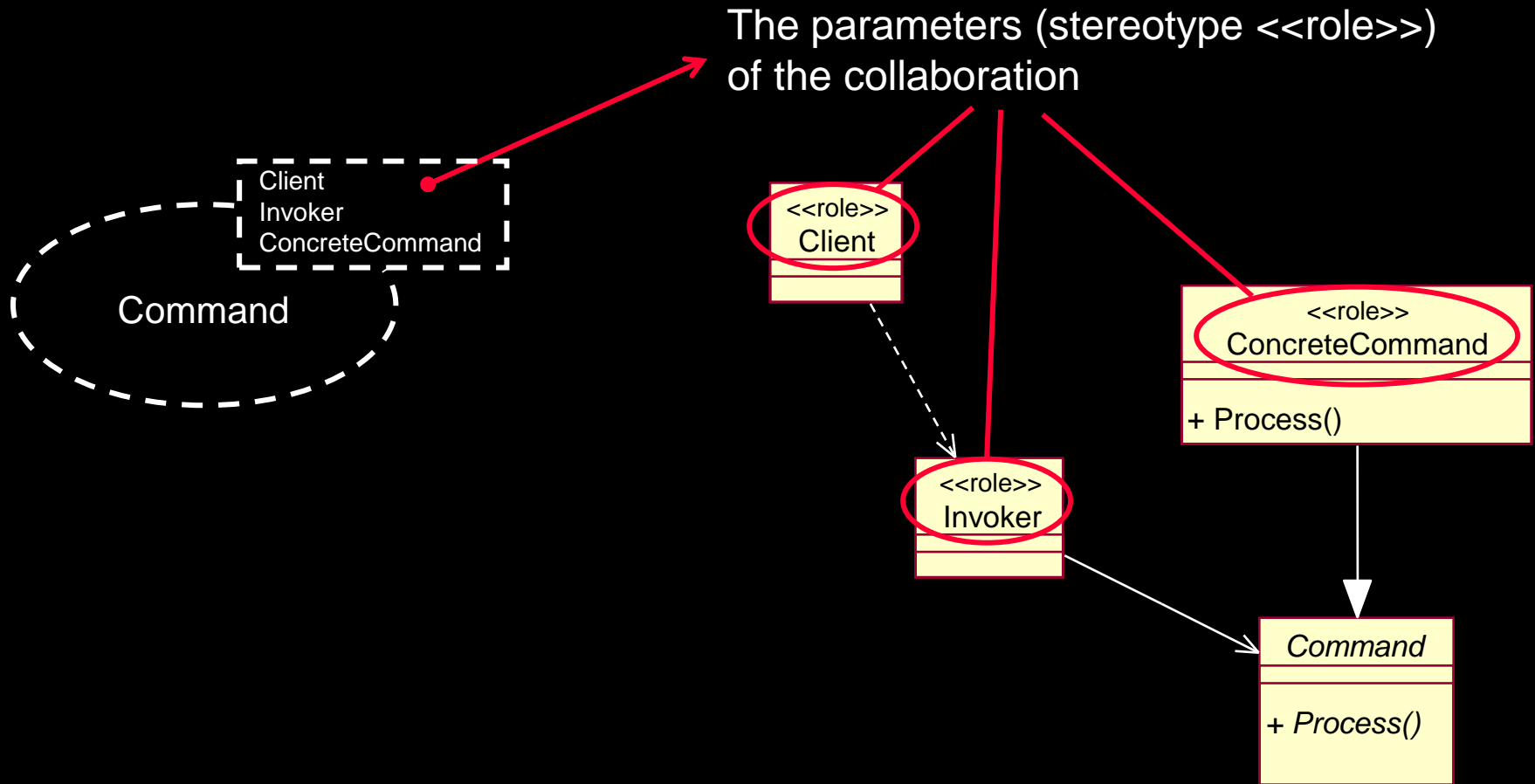


Ví dụ

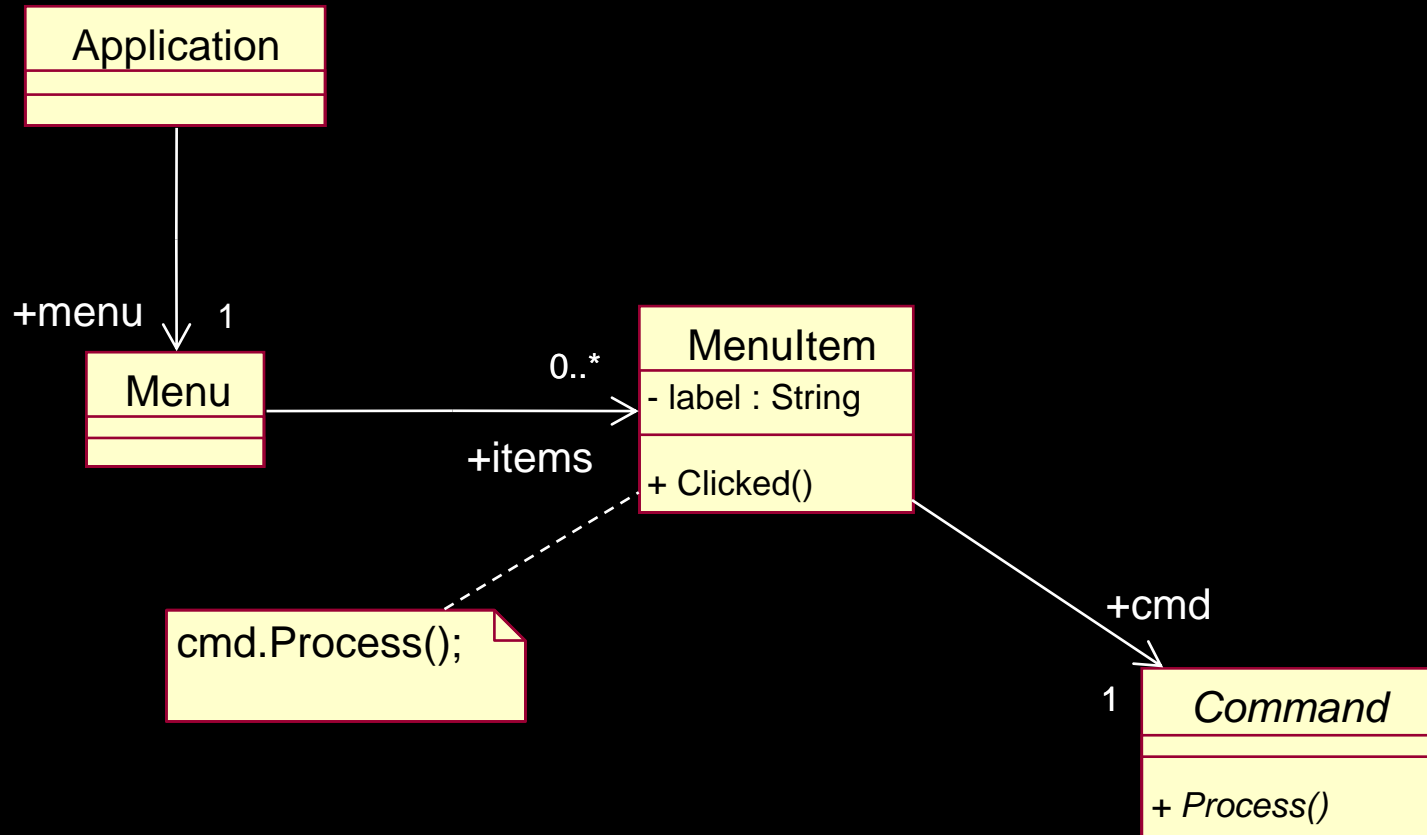
Mẫu thiết kế	Ví dụ
Command (behavioral pattern)	Đóng gói các yêu cầu thực thi trong đối tượng
Abstract factory (creational pattern)	Tạo các đối tượng đồ họa (buttons, scrollbars, windows, etc.) sao cho ứng dụng có thể dễ dàng thay đổi chúng
Proxy (structural pattern)	Đại diện phân phối các thông điệp từ một client, giúp client không cần quá quan tâm tới đích đến của thông điệp
Observer (behavioral pattern)	Khi trạng thái của một đối tượng thay đổi, các đối tượng phụ thuộc vào nó cần được thông báo.

Biểu diễn mẫu thiết kế với UML

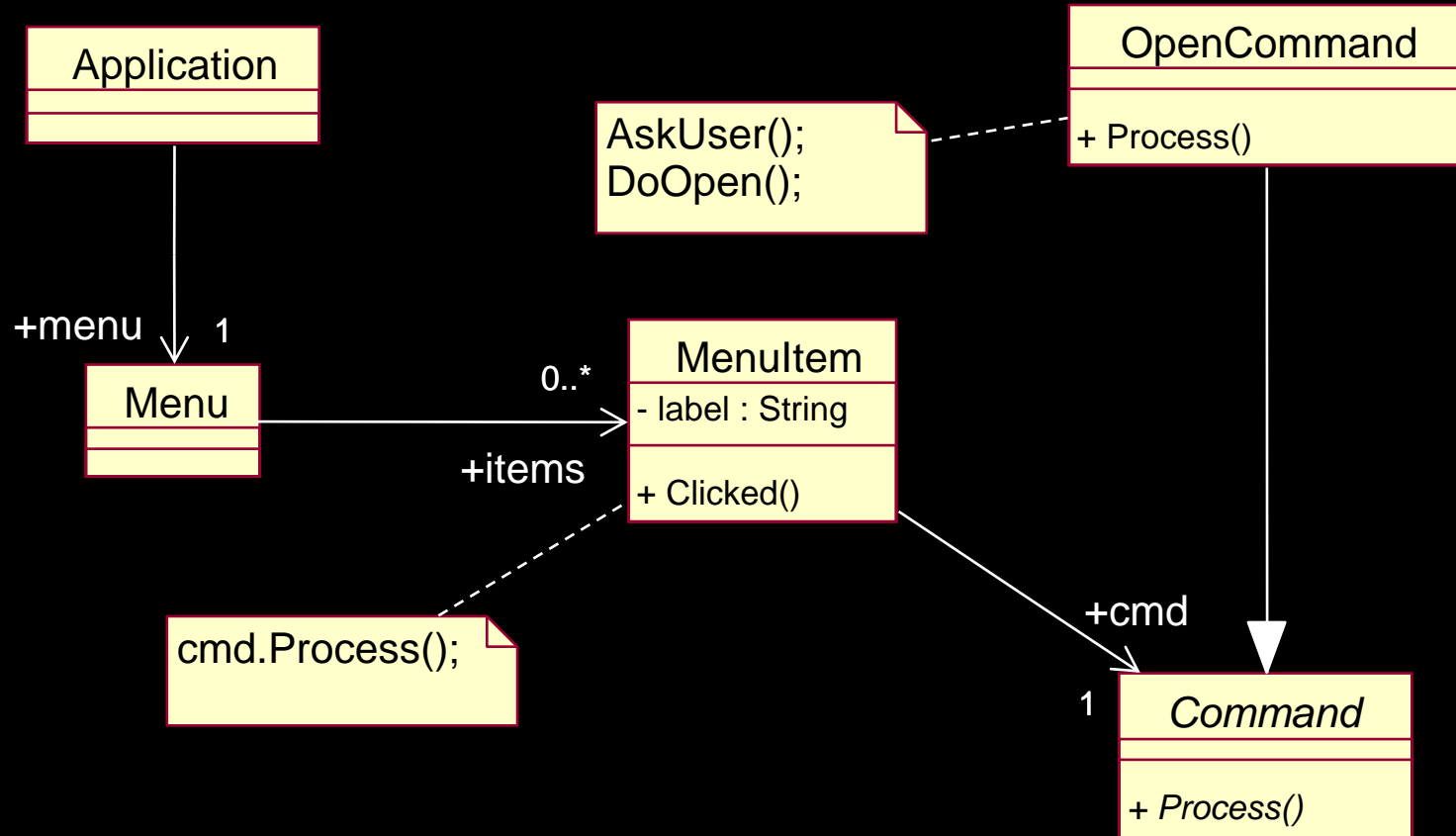
- ♦ A design pattern is a parameterized collaboration:



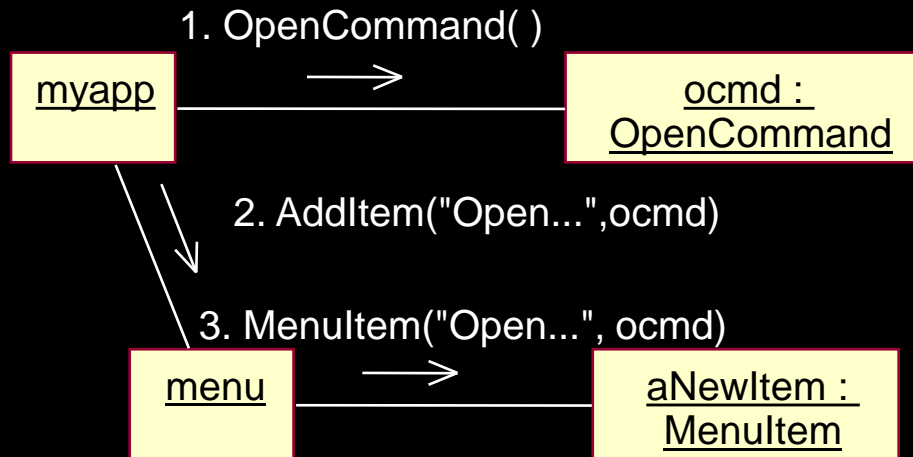
Command Pattern – cấu trúc



Command Pattern – cấu trúc

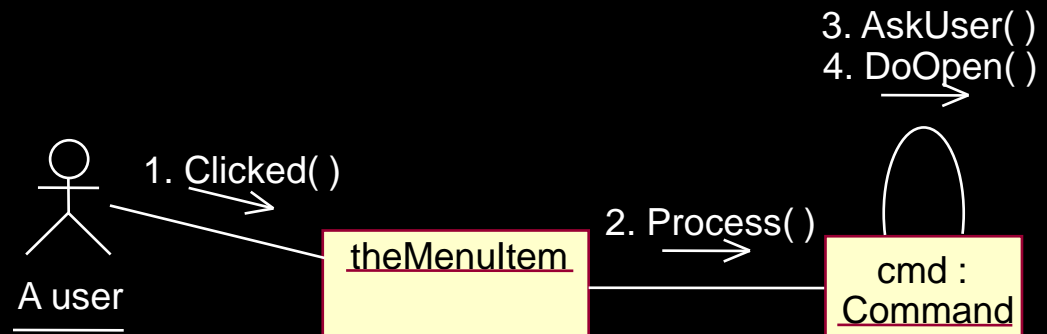


Command Pattern – hành vi

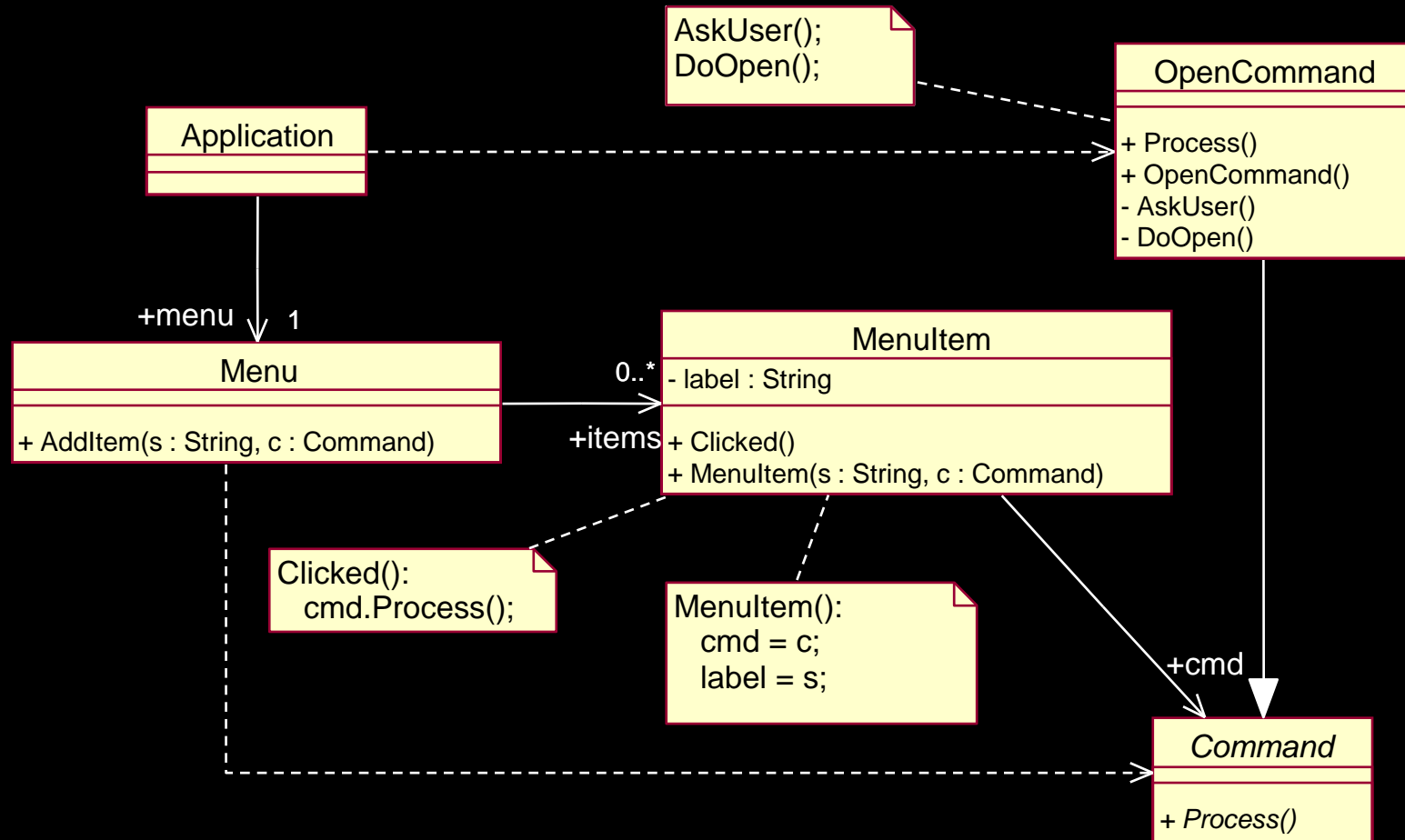


Initialization

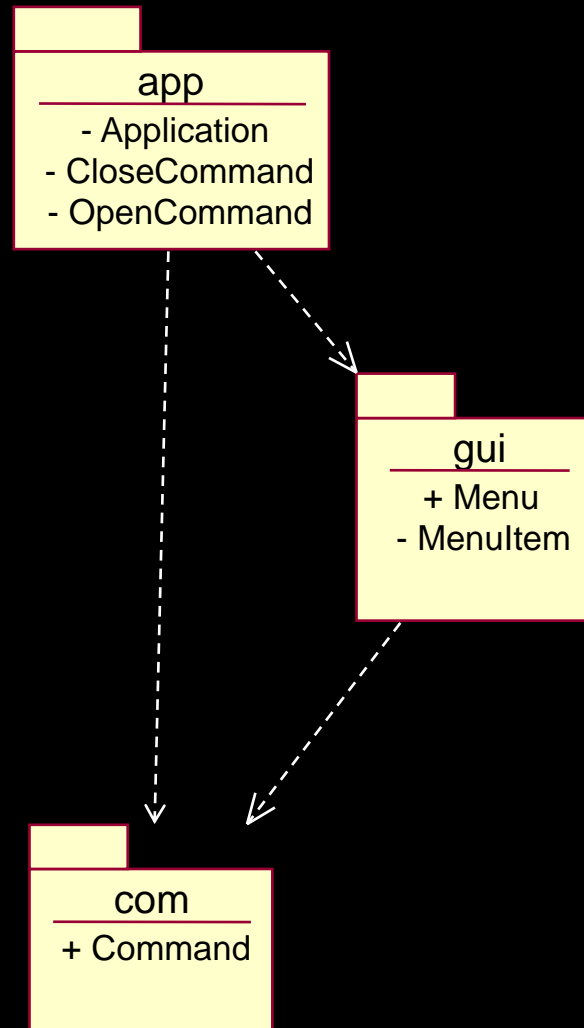
The user selects the *Open...* menu item



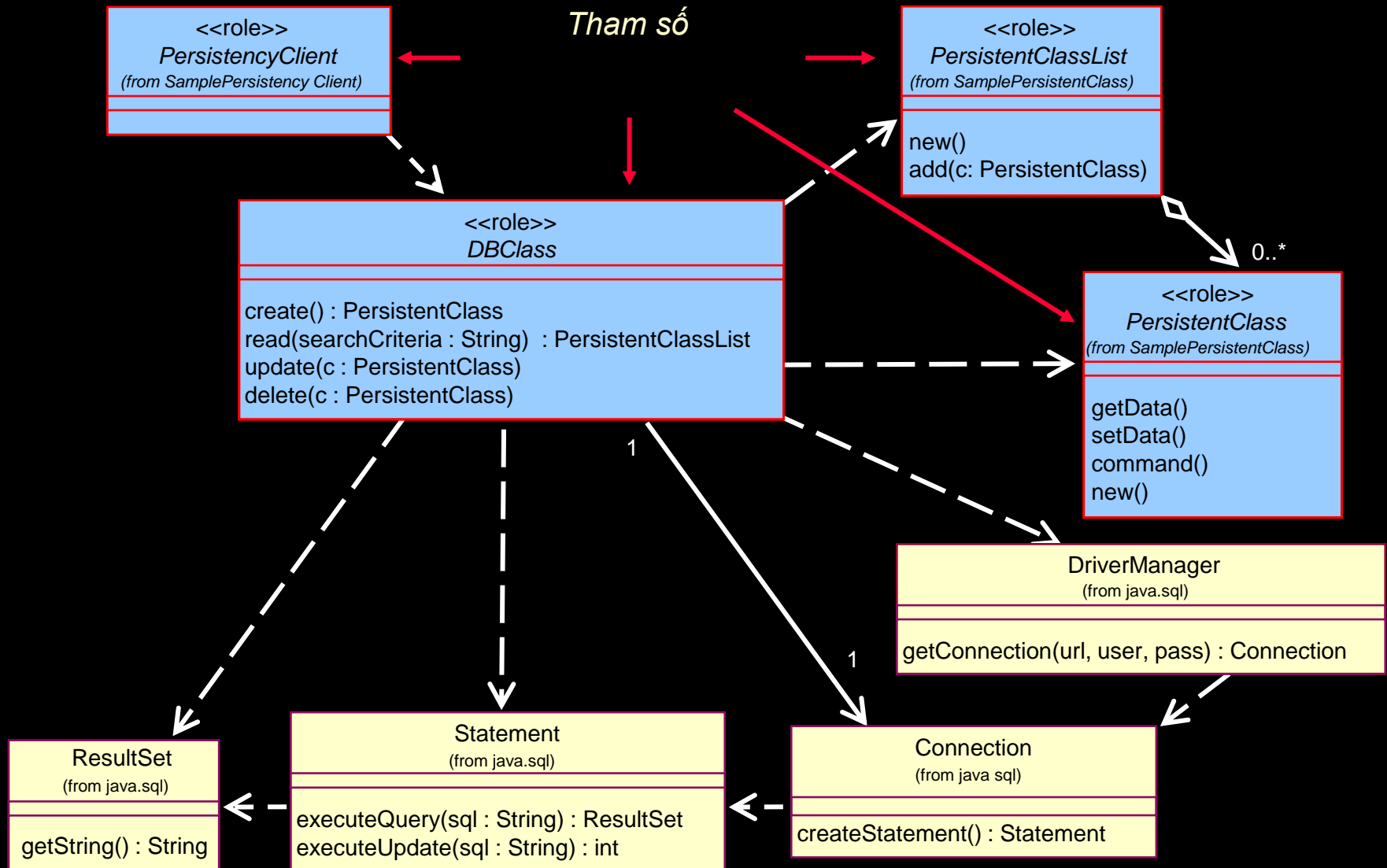
Command Pattern



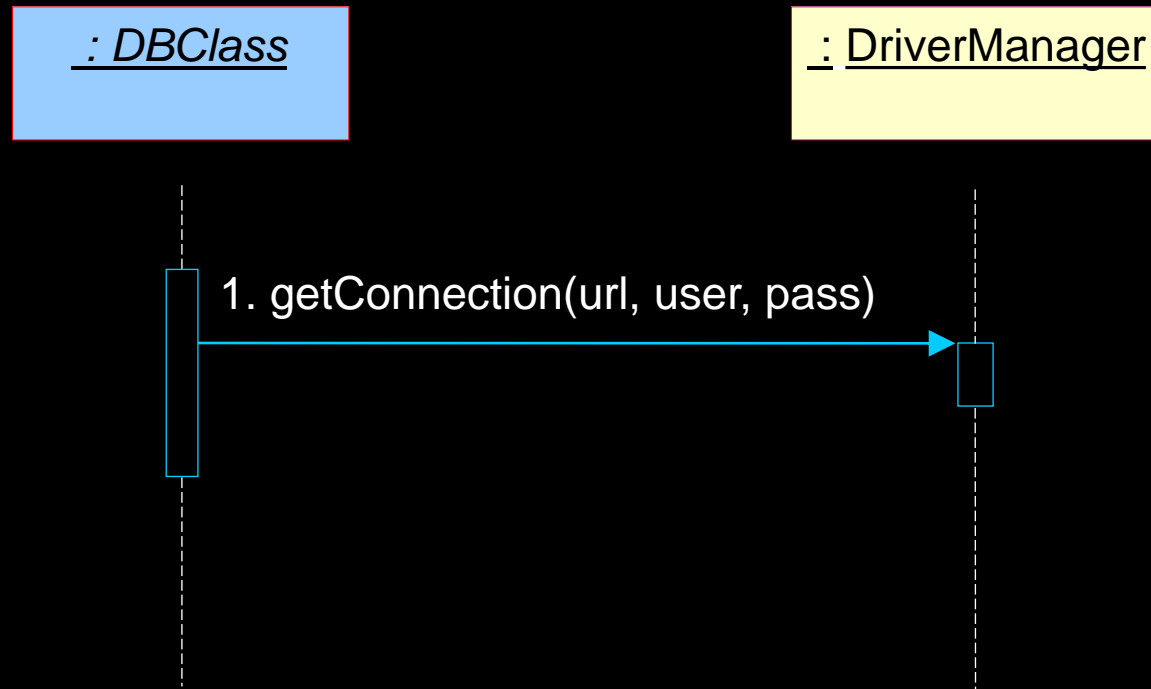
Command Pattern – cập nhật kiến trúc



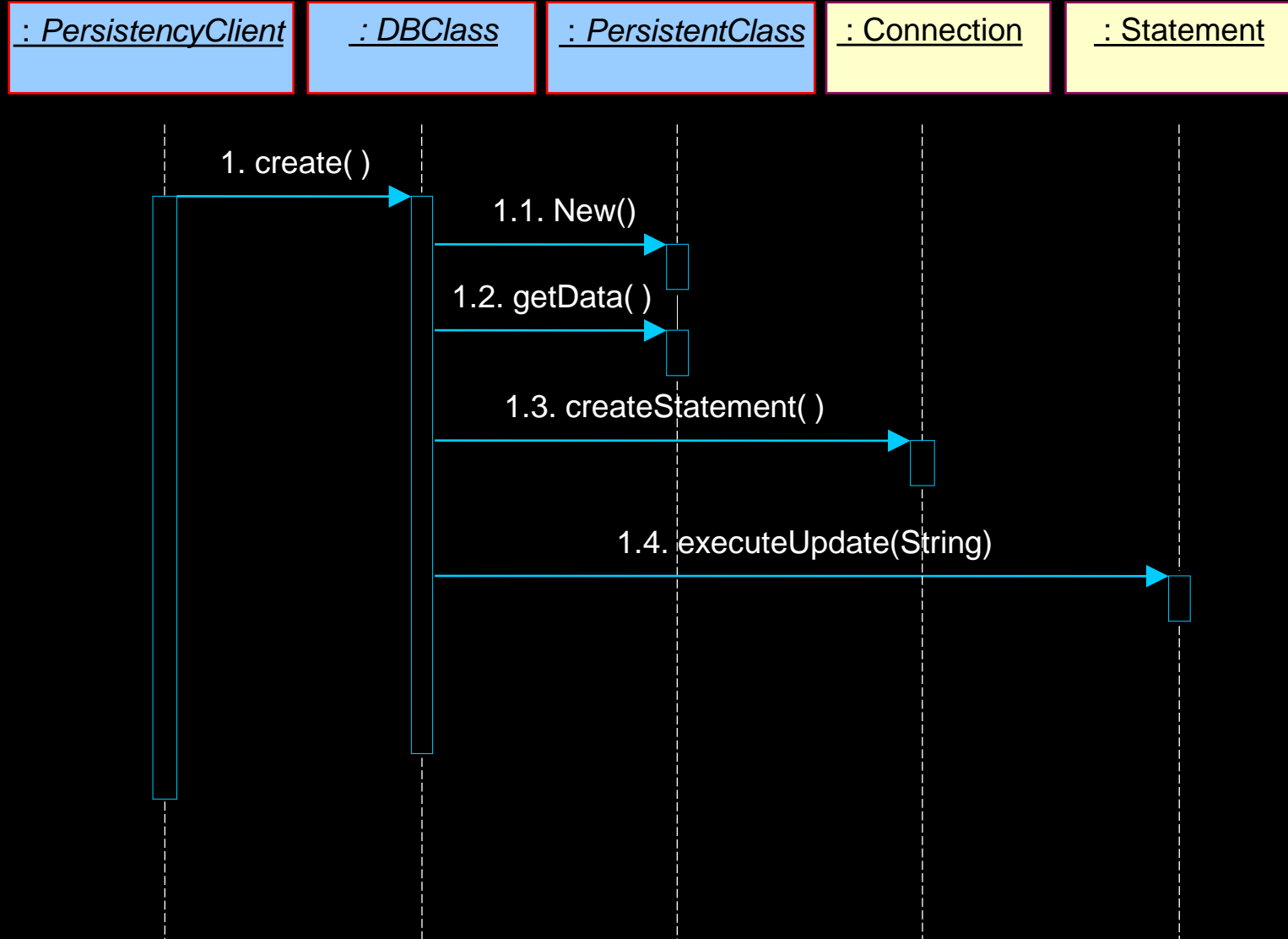
Ví dụ: Persistency: RDBMS: JDBC



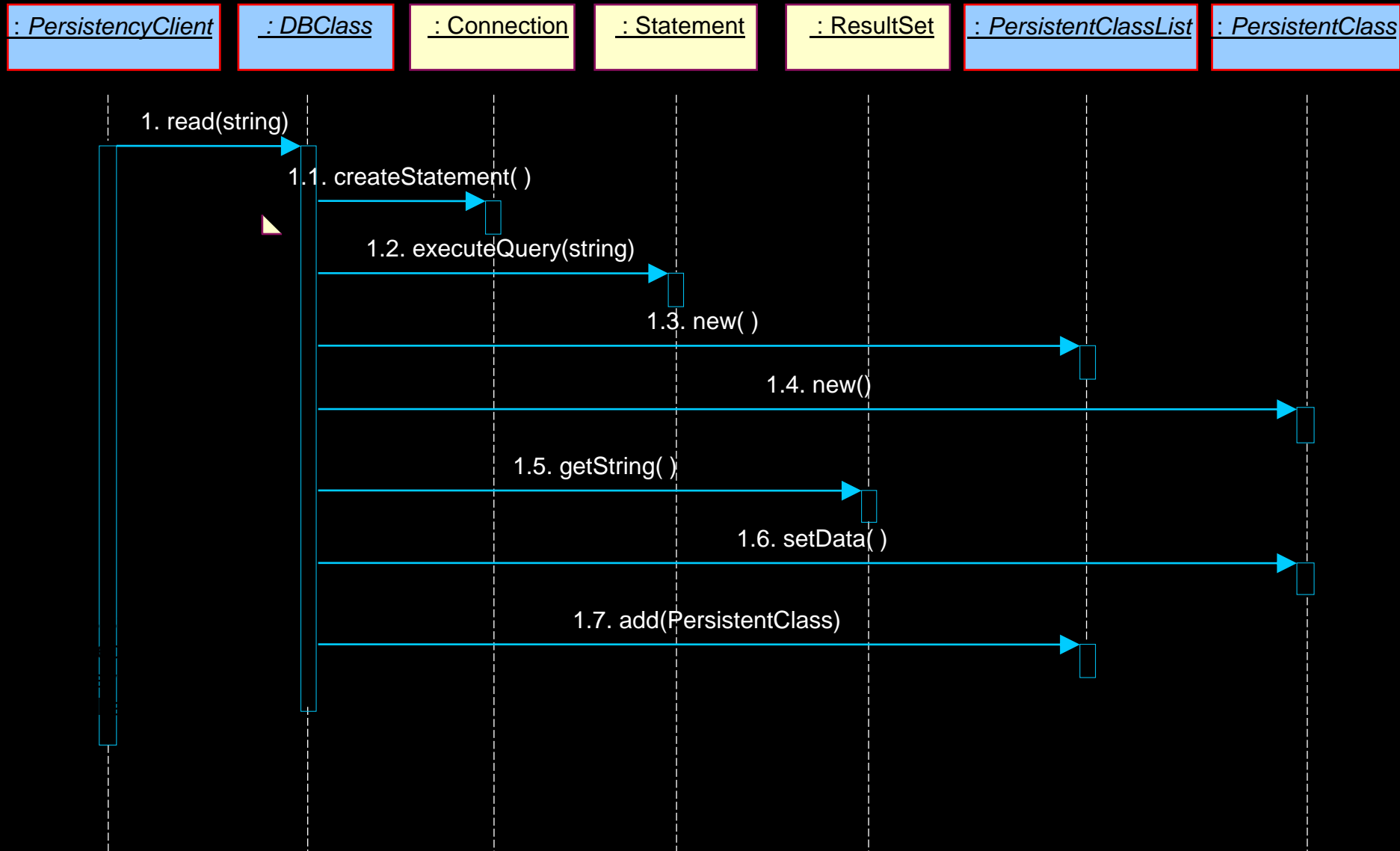
Ví dụ: Persistency: RDBMS: JDBC: Initialize



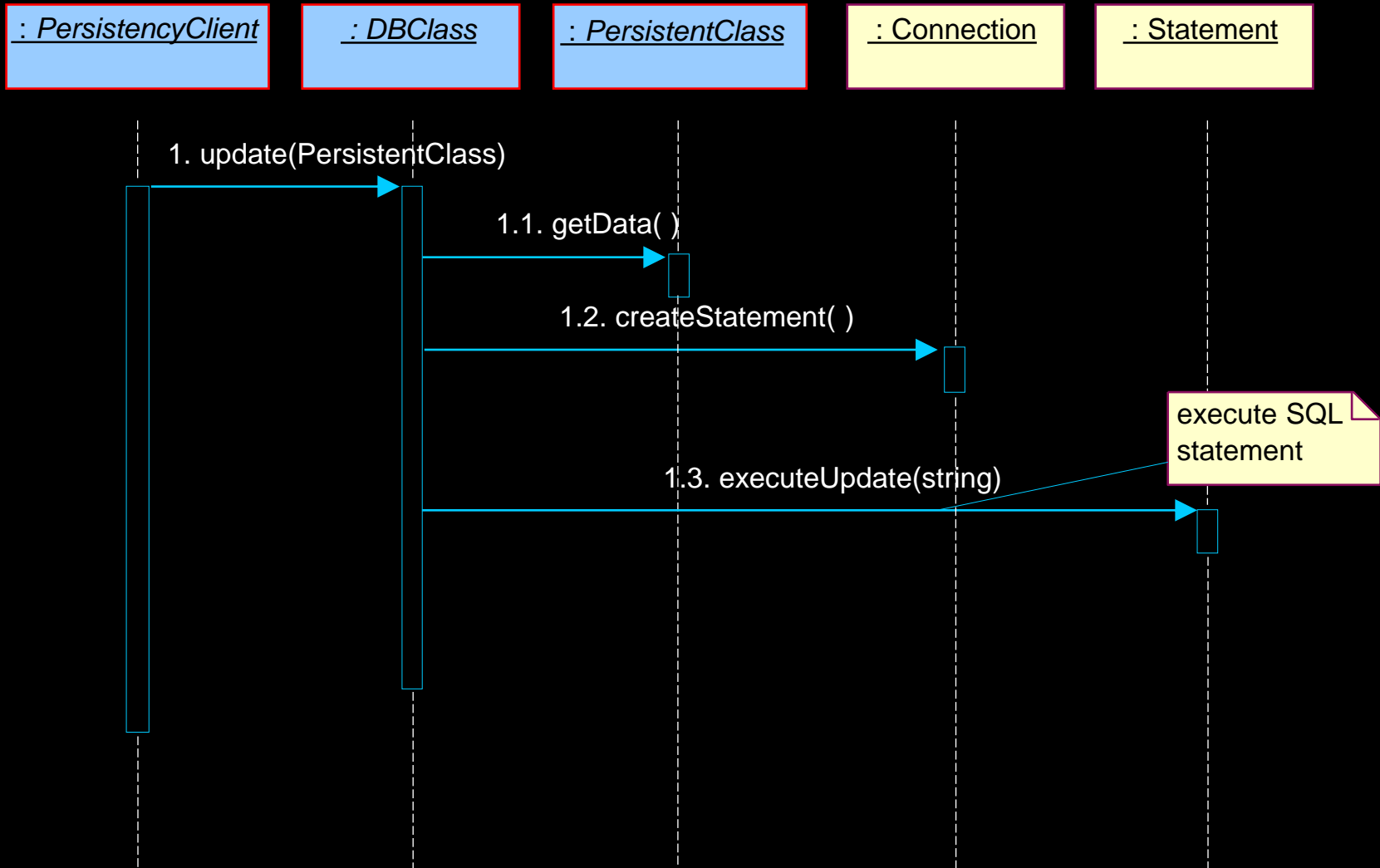
Ví dụ: Persistency: RDBMS: JDBC: Create



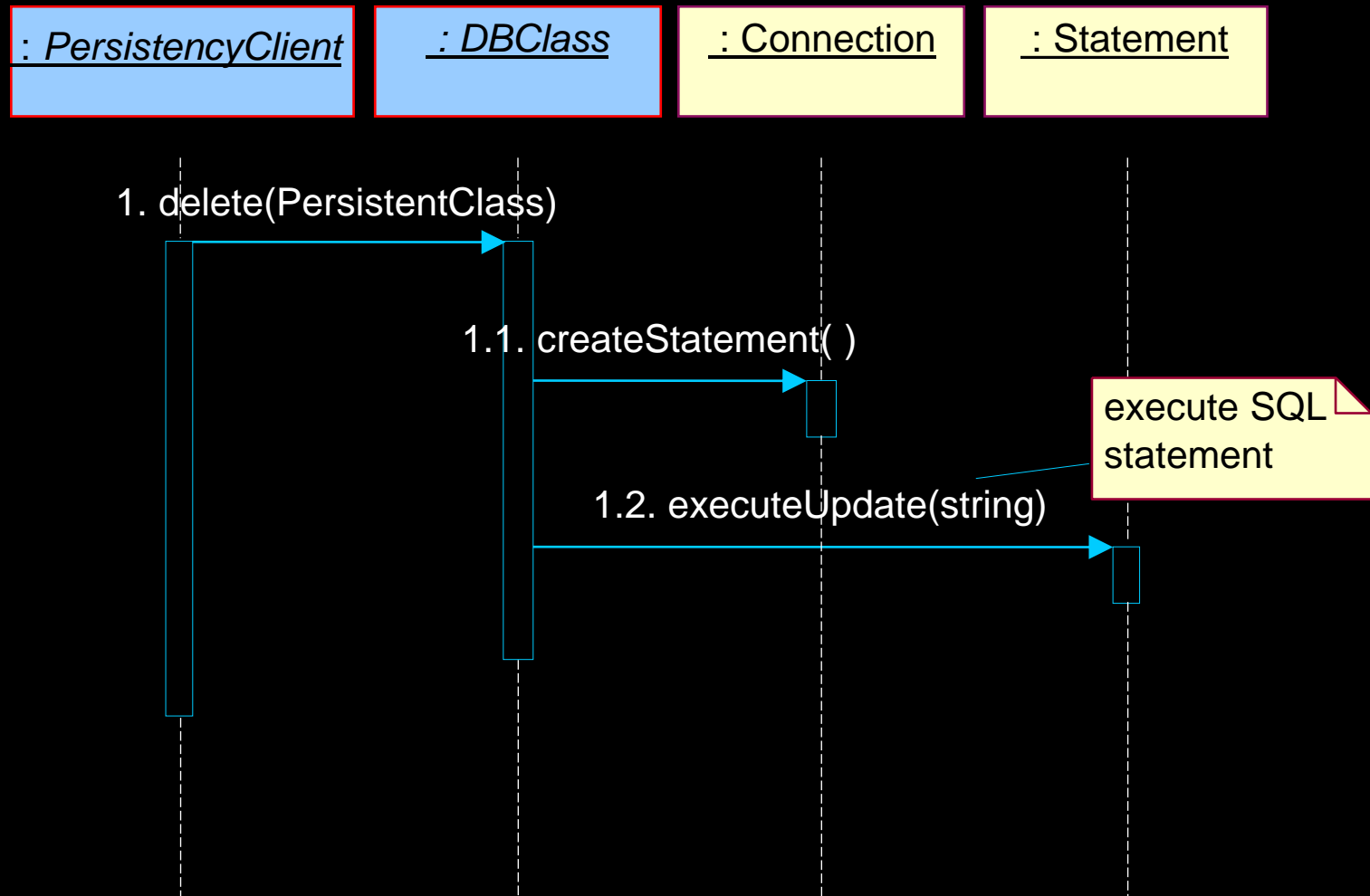
Ví dụ: Persistency: RDBMS: JDBC: Read



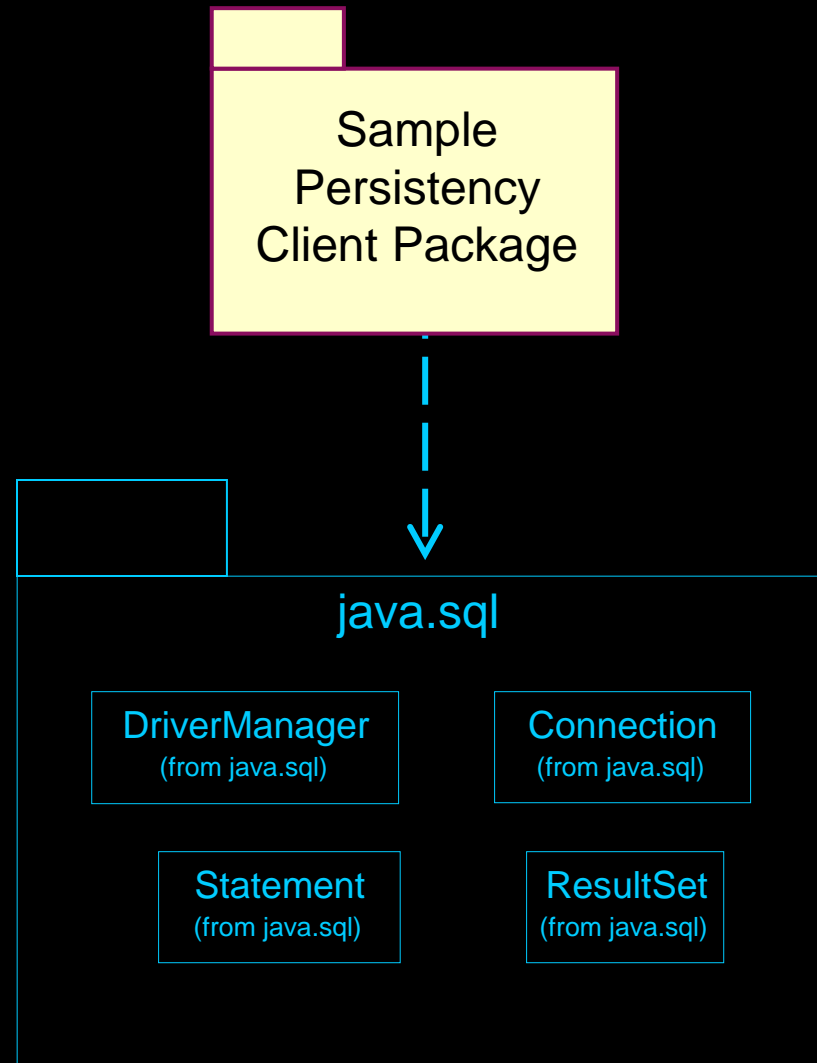
Ví dụ: Persistency: RDBMS: JDBC: Update



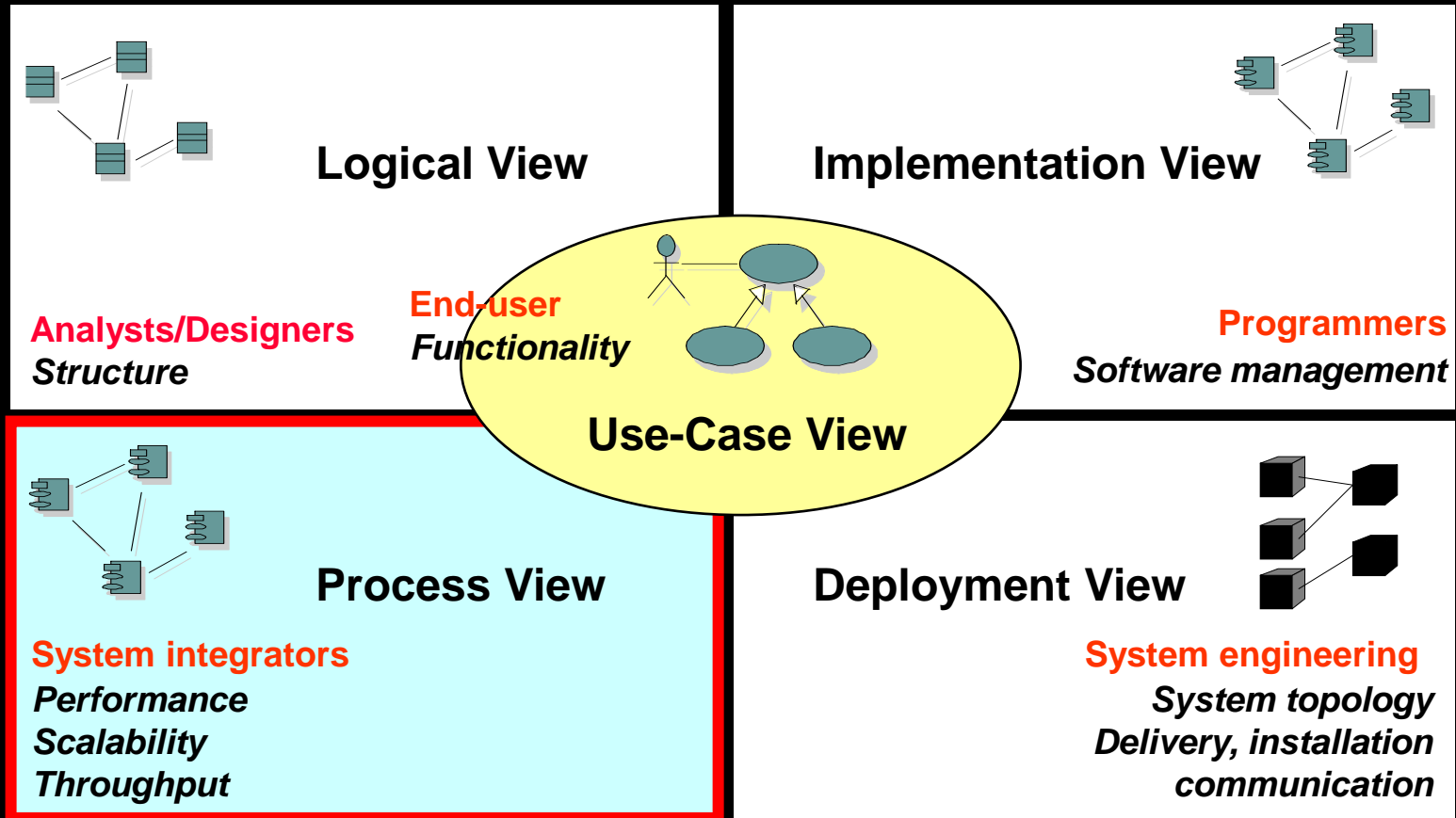
Ví dụ: Persistency: RDBMS: JDBC: Delete



Ví dụ: Cập nhật JDBC vào bản thiết kế kiến trúc



Kiến trúc thực thi: Khung nhìn tiến trình

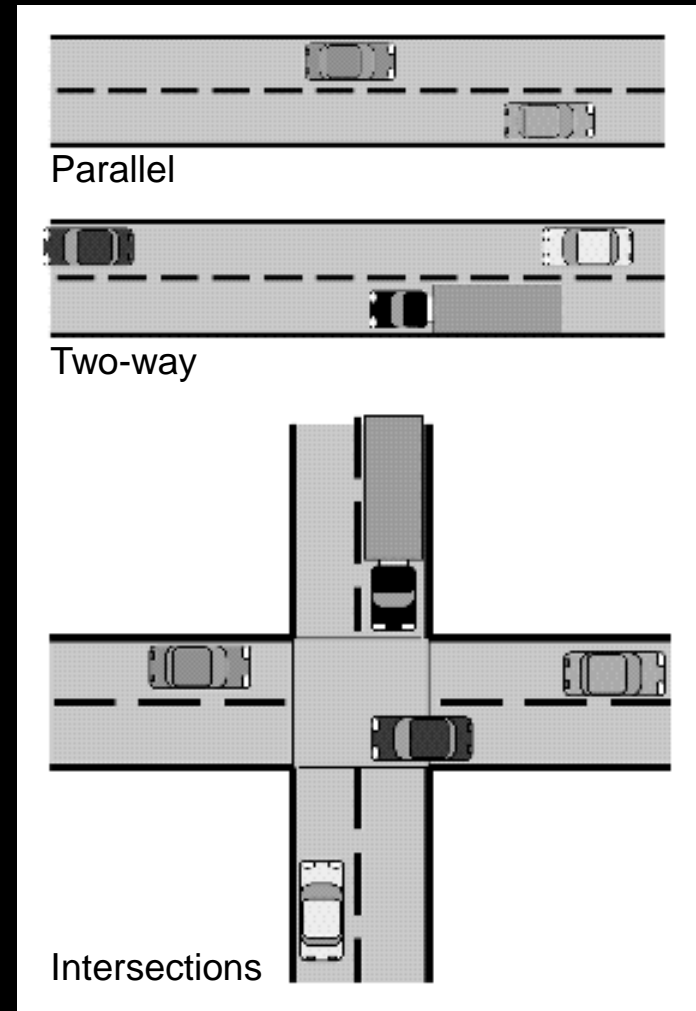


The Process View is an “architecturally significant” slice of the processes and threads of the Design Model

Tương tranh (concurrency)

♦ Ví dụ về tương tranh:

- Đường song song cần ít sự điều phối
- Đường hai chiều cần nhiều sự điều phối hơn
- Giao lộ cần điều phối cẩn thận



Tương tranh trong các hệ thống phần mềm

- ◆ Cần trả lời đồng thời cho nhiều yêu cầu từ bên ngoài
- ◆ Tăng hiệu năng



Các bước mô tả kiến trúc thực thi

- ◆ Phân tích nhu cầu thực hiện tương tranh
- ◆ Xác định các luồng/tiến trình
- ◆ Xác định vòng đời của các luồng/tiến trình
- ◆ Phân phối các yếu tố thiết kế lên các luồng/tiến trình

Nhu cầu tương tranh

♦ Nhu cầu tương tranh xuất phát từ các lý do:

- Mức độ phân tán của hệ thống.
- Tiếp cận hướng sự kiện của hệ thống.
- Mật độ tính toán của các giải thuật.
- Nhu cầu xử lý song song từ môi trường



Ví dụ:

- ♦ Trong hệ thống Course Registration System, nhu cầu tương tranh đến từ các yếu tố sau:
 - Nhiều người dùng đồng thời truy nhập hệ thống
 - Nếu một khóa học đã đầy thì những sinh viên đang đăng ký khóa học này cần phải được thông báo
 - Tốc độ truy nhập vào cơ sở dữ liệu bên ngoài (legacy course catalog database) không đáp ứng yêu cầu về hiệu năng

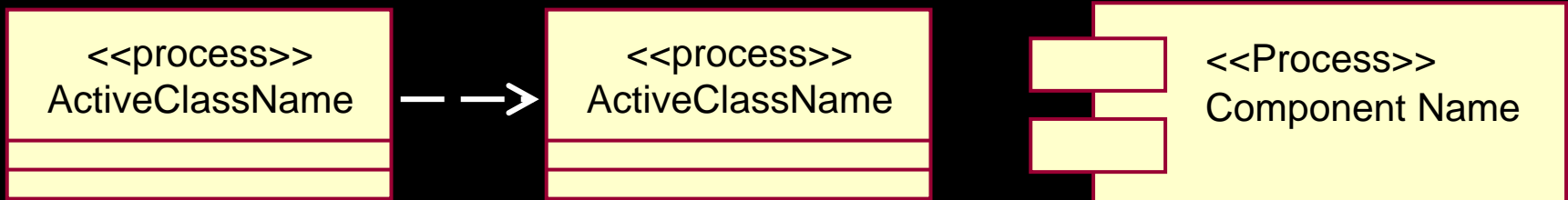
Xác định luồng/tiến trình

- ♦ Mọi luồng/tiến trình tương ứng với một luồng điều khiển độc lập
 - Tận dụng kiến trúc đa lõi/CPU/node
 - Tăng hiệu quả sử dụng CPU
 - Cải thiện thời gian đáp ứng dịch vụ
 - Cho phép các hoạt động ưu tiên
 - Tăng khả năng co giãn
 - Phân tách mối quan tâm
 - Cải thiện tính sẵn sàng
 - Hỗ trợ các phân hệ chính

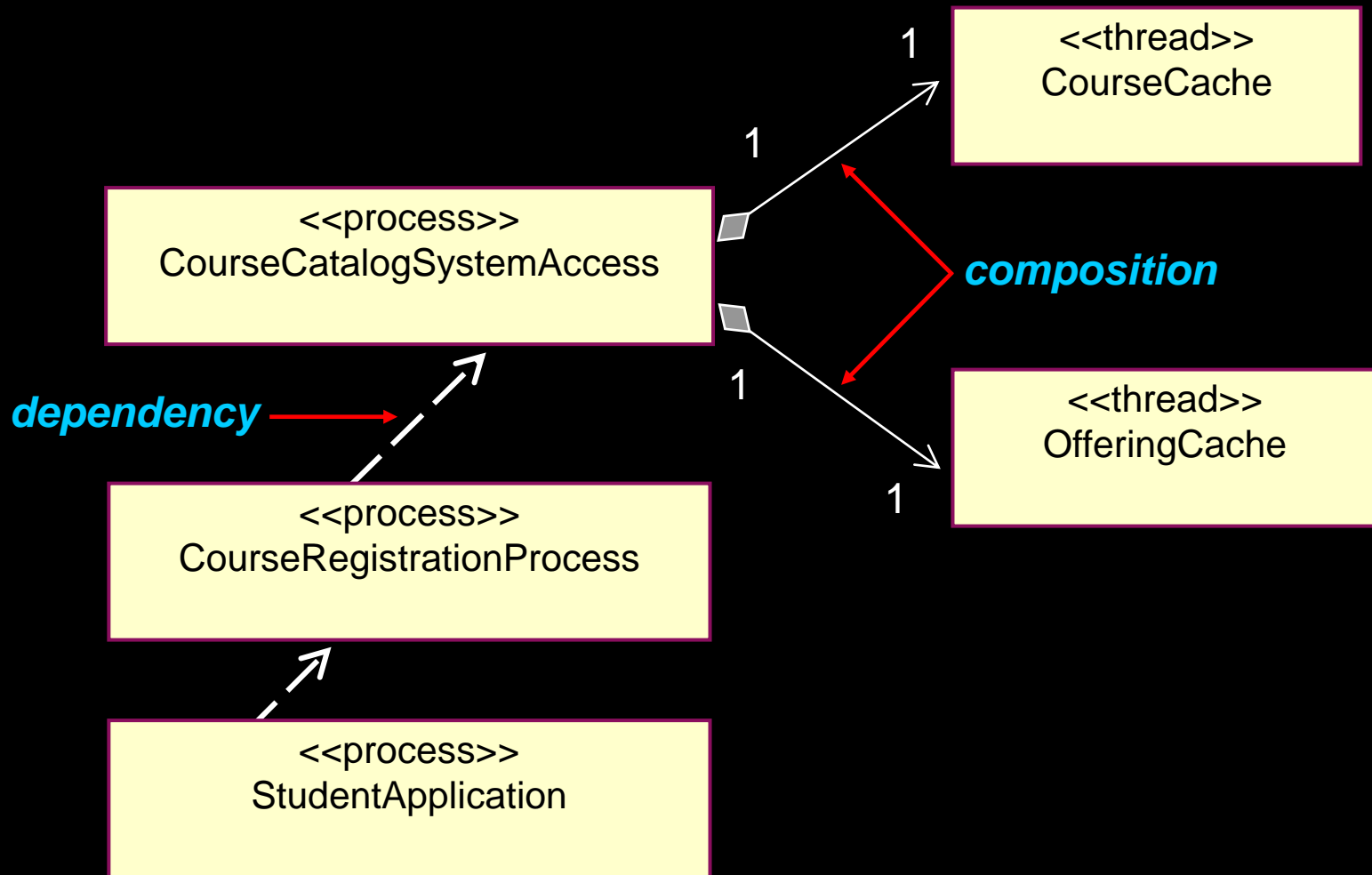


Mô hình hóa luồng/tiến trình

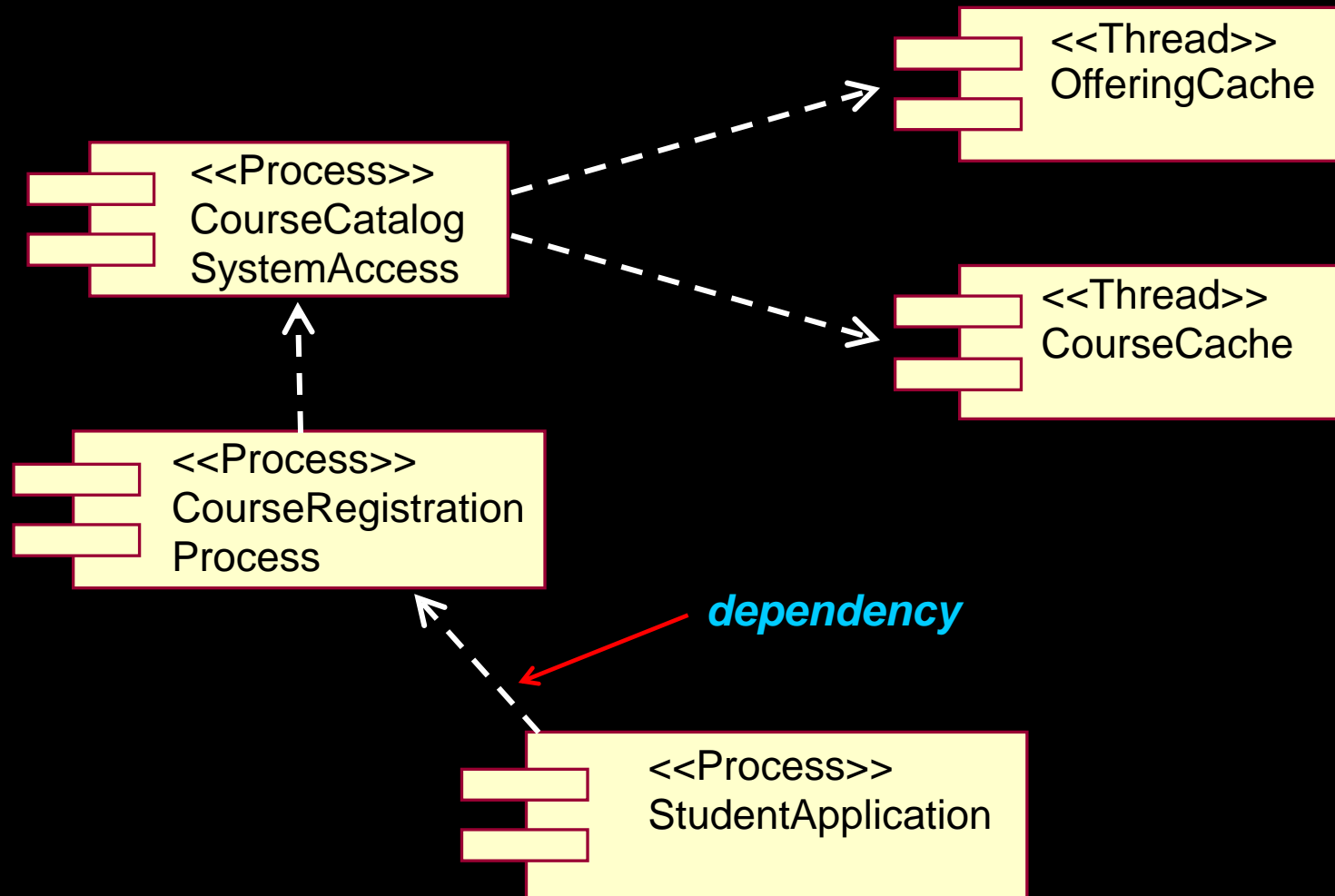
- ♦ Tiến trình có thể được mô hình hóa bằng
 - Active classes (Class Diagrams) và Objects (Interaction Diagrams)
 - Components (Component Diagrams)
- ♦ Stereotypes: <<process>> hoặc <<thread>>
- ♦ Mỗi quan hệ giữa các tiến trình có thể mô hình hóa bằng sự phụ thuộc



Ví dụ



Ví dụ



Vòng đời của các luồng/tiến trình

♦ Kiến trúc đơn tiến trình

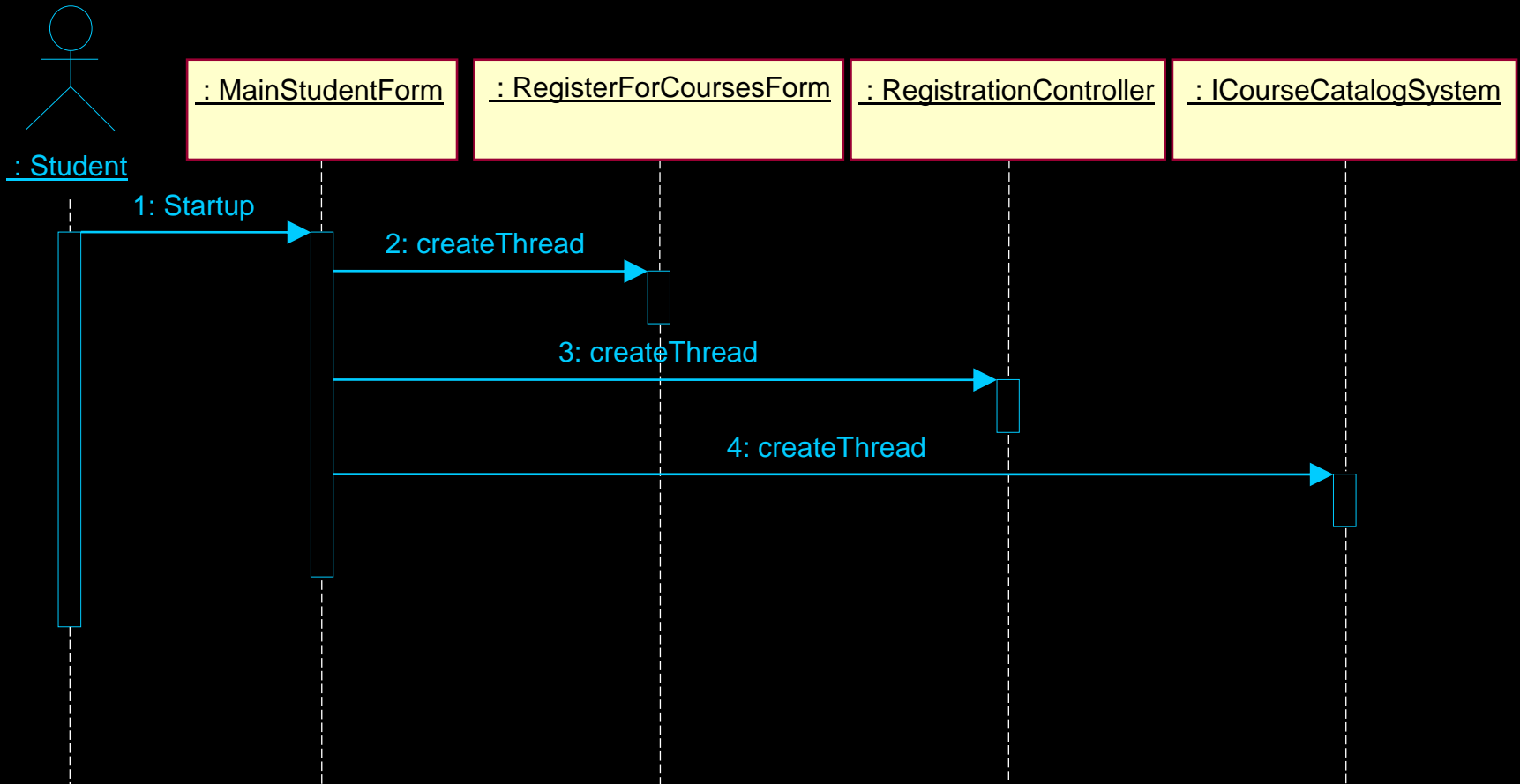
- Tiến trình được tạo ra khi ứng dụng khởi động và bị hủy đi khi ứng dụng kết thúc

♦ Kiến trúc đa tiến trình

- Các tiến trình thường được tạo ra từ một tiến trình gốc khi ứng dụng khởi động
- Các tiến trình có thể được hủy một cách độc lập

Note: The Course Registration System utilizes a multi-process architecture

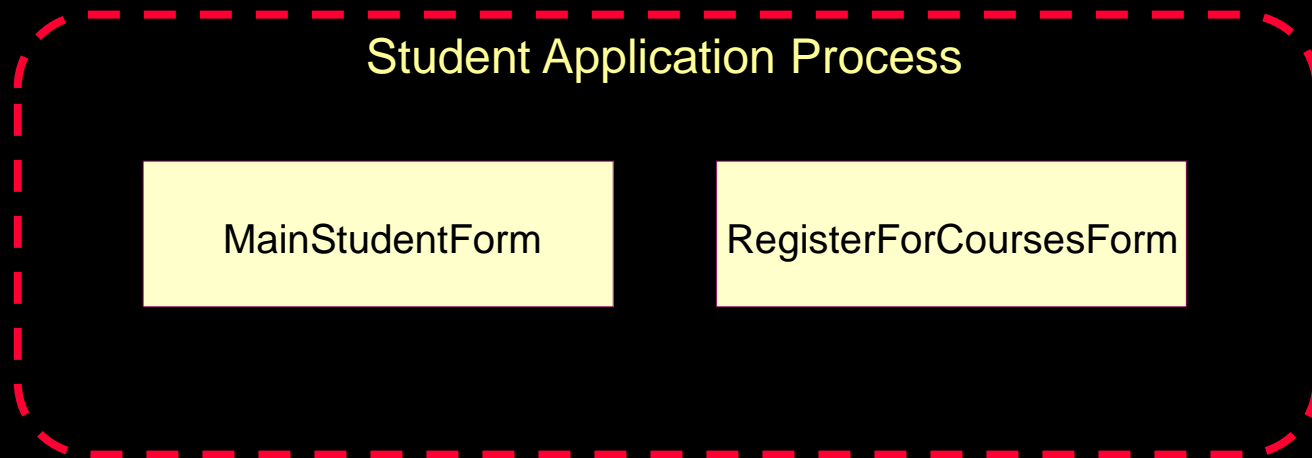
Ví dụ:



Creation of threads during application startup

Phân phối các yếu tố thiết kế lên các tiến trình

- ♦ Các thể hiện của một lớp cần được thực hiện trên ít nhất một luồng/tiến trình
 - Chúng có thể thực hiện trên nhiều luồng/tiến trình



Nguyên tắc phân phối

♦ Dựa trên:

- Nhu cầu về tương tranh và hiệu năng
- Nhu cầu phân tán hoặc thực thi song song song song
- Nhu cầu dự phòng và tính sẵn sàng

Chiến lược phân phối

Hai chiến lược phân phối (có thể áp dụng đồng thời)

◆ Inside-Out

- Nhóm các yếu tố thiết kế thường xuyên tương tác với nhau trong một luồng điều khiển
- Phân tách các yếu tố thiết kế không tương tác với nhau
- Lặp cho tới khi đạt được số lượng tối thiểu tiến trình phù hợp với yêu cầu phân tán và tài nguyên sẵn có

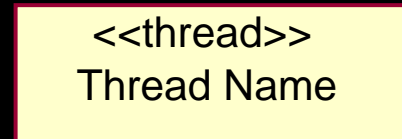
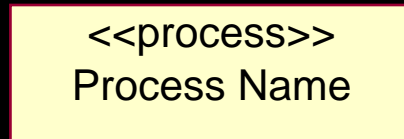
◆ Outside-In

- Xác định một luồng độc lập cho mỗi yêu cầu dịch vụ từ bên ngoài hoặc mỗi sự kiện bên ngoài cần đáp ứng
- Gom nhóm và giảm dần số lượng luồng điều khiển cho tới khi thỏa mãn yêu cầu phân tán và tài nguyên sẵn có

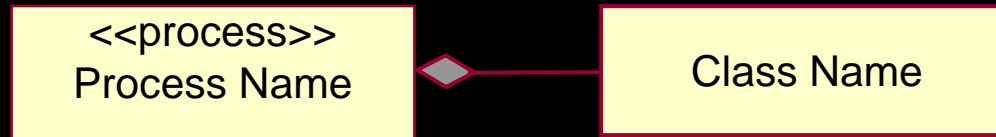
Mô hình hóa

♦ Biểu đồ lớp

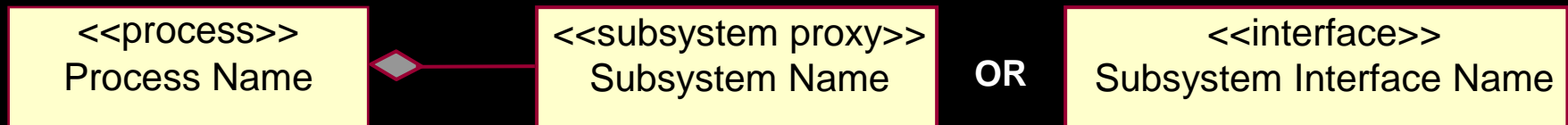
- Sử dụng active classes cho các tiến trình/luồng



- Sử dụng quan hệ Composition cho mục đích phân phối lớp

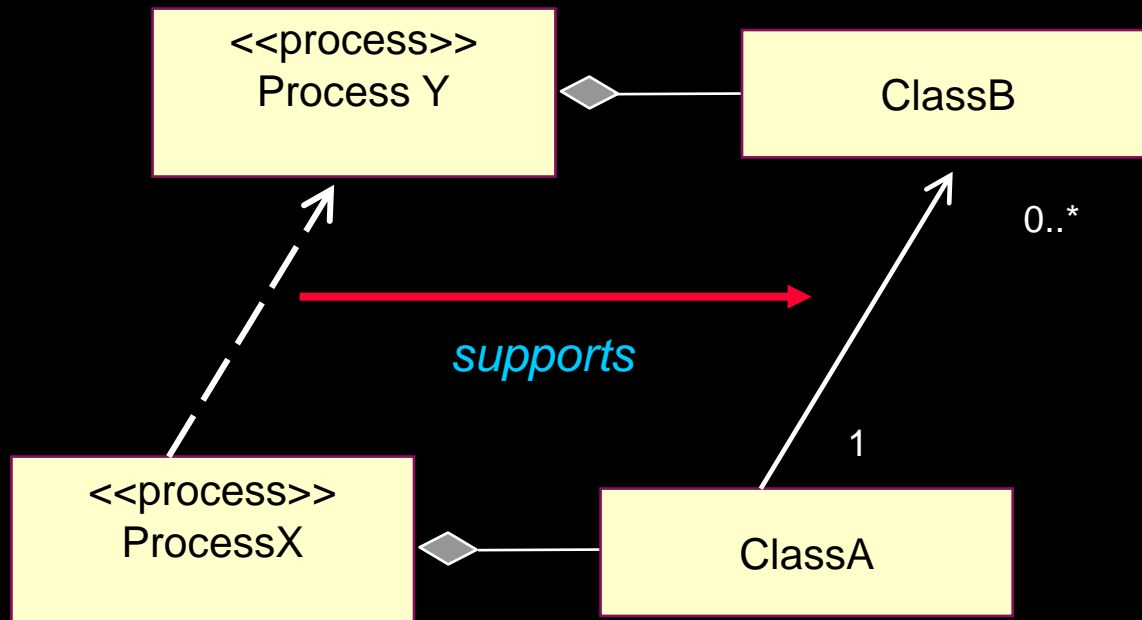


- Sử dụng quan hệ Composition cho mục đích phân phối hệ thống con

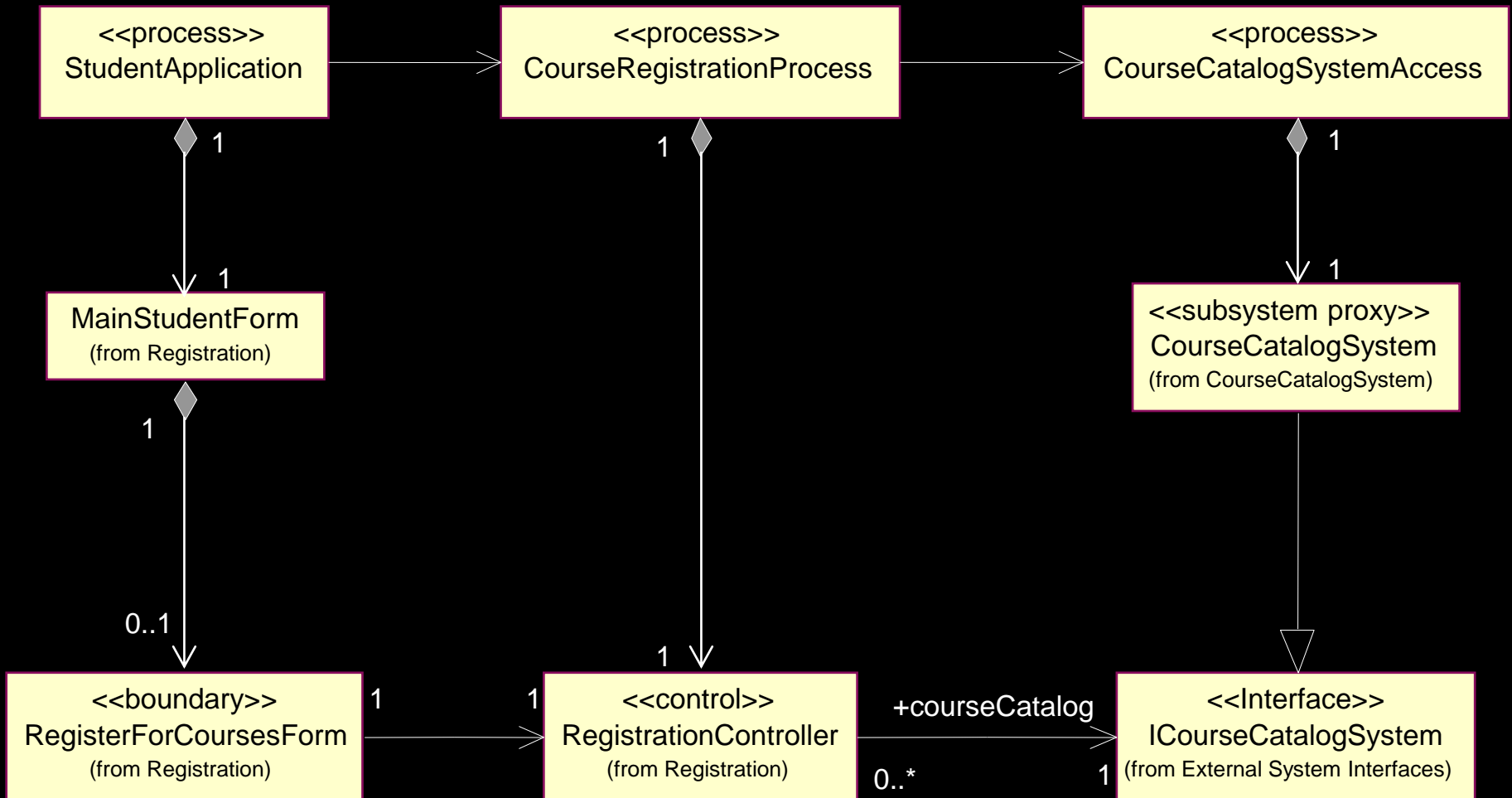


Quan hệ giữa các tiến trình

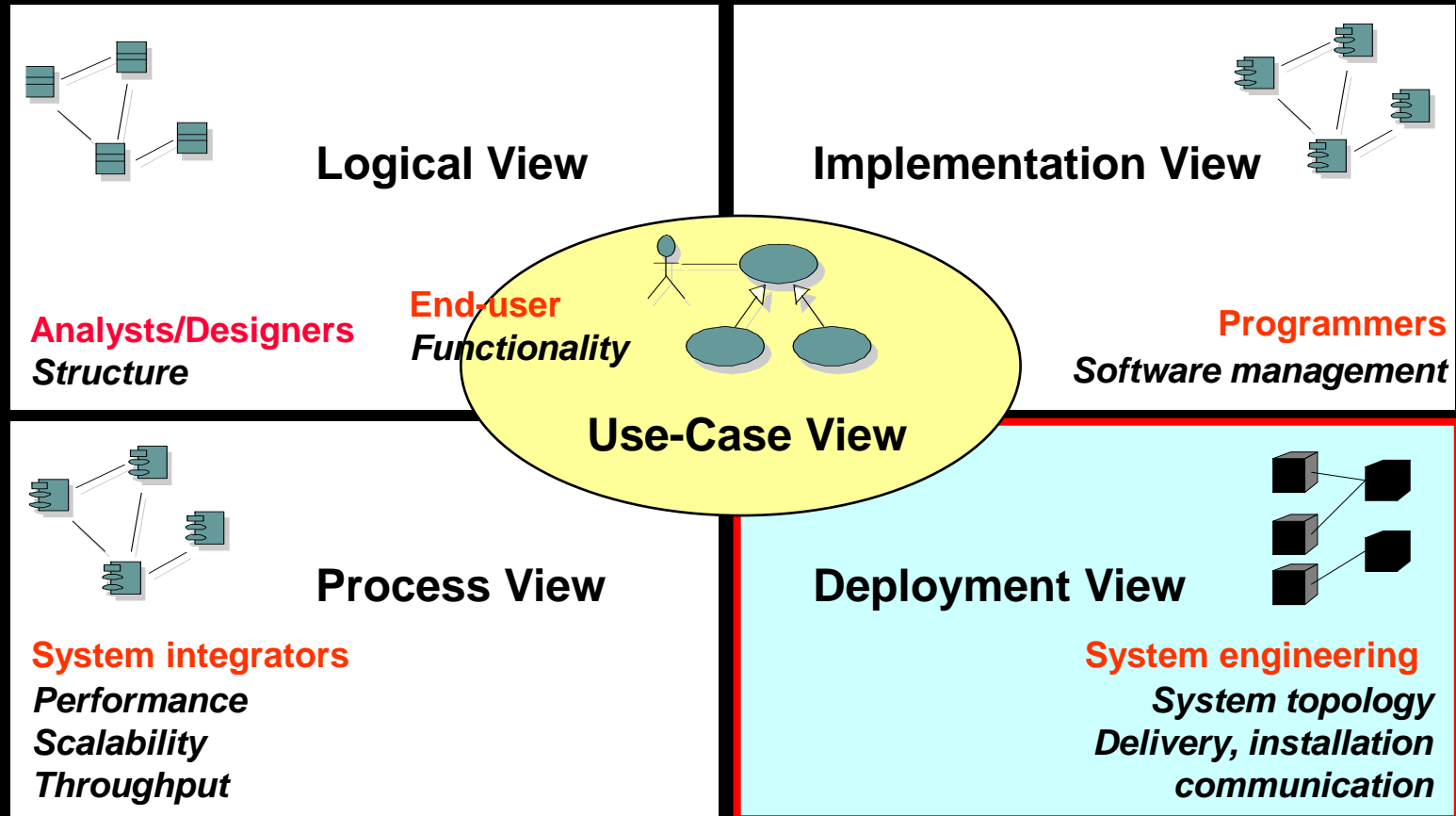
- ♦ Quan hệ giữa các tiến trình phải khớp với quan hệ giữa các yếu tố thiết kế



Ví dụ:



Kiến trúc triển khai: The Deployment View



The Deployment View is an “architecturally significant” slice of the Deployment Model.

Bố trí các thành phần của hệ thống lên nhiều nút

- ◆ Giảm tải cho các bộ xử lý
- ◆ Nhu cầu sử dụng phần mềm
- ◆ Khả năng co giãn của hệ thống
- ◆ Lý do kinh tế



Các mẫu phân tán

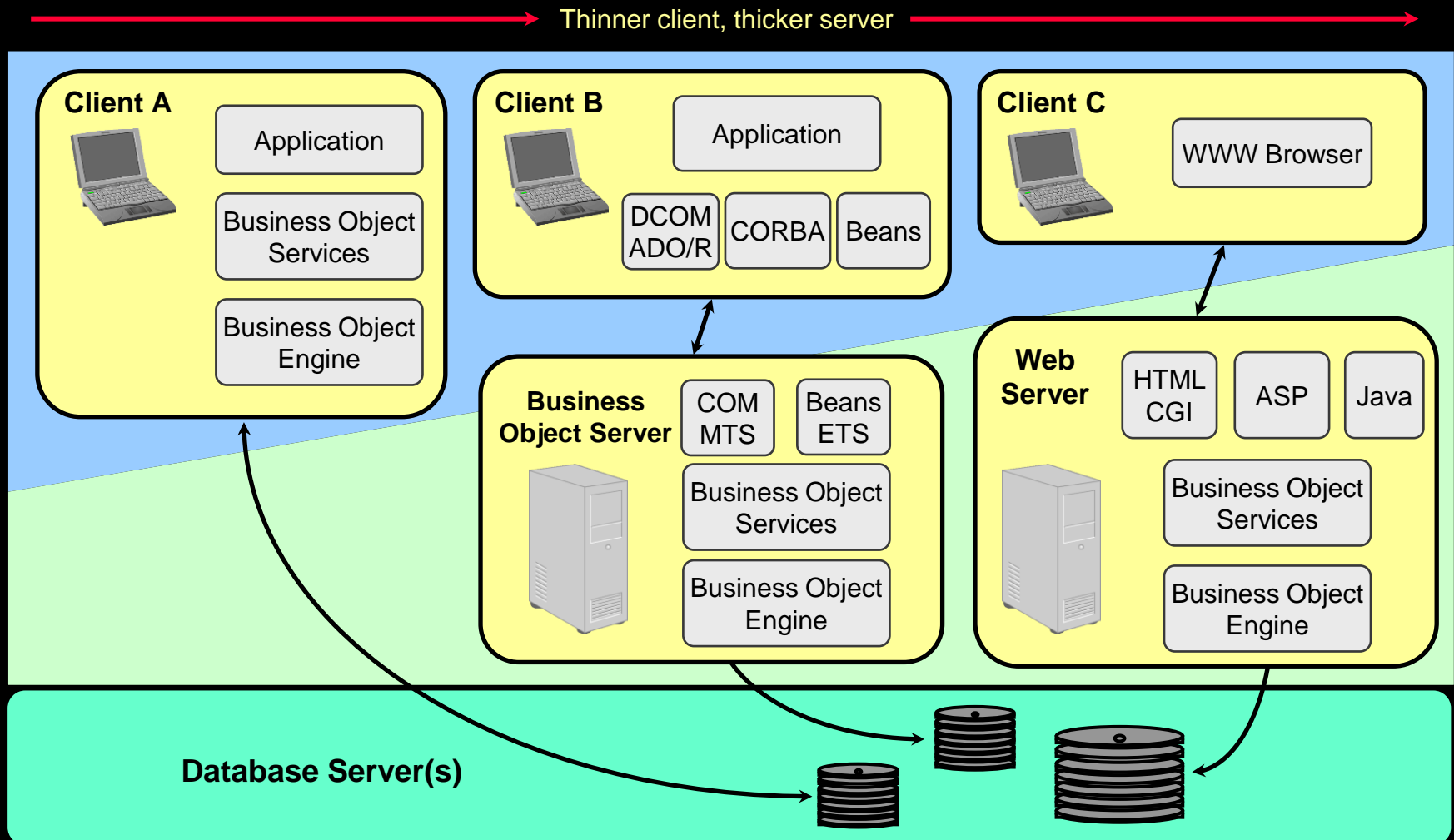
◆ Client/Server

- 3-tier
- Fat Client
- Fat Server
- Distributed Client/Server

◆ Peer-to-peer

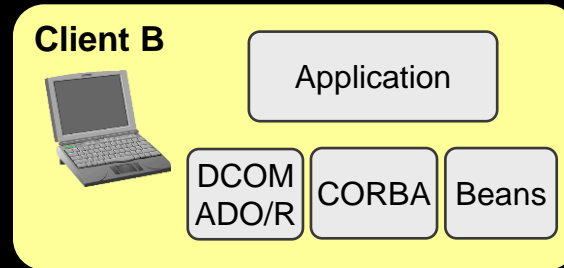


Client/Server Architectures

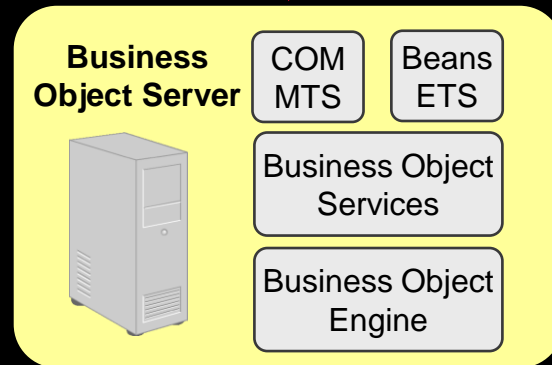


Client/Server: Three-Tier Architecture

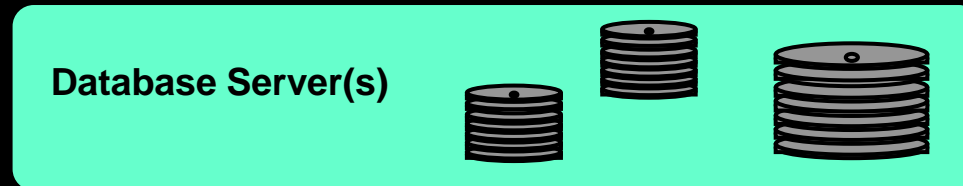
Application Services



Business Services



Data Services

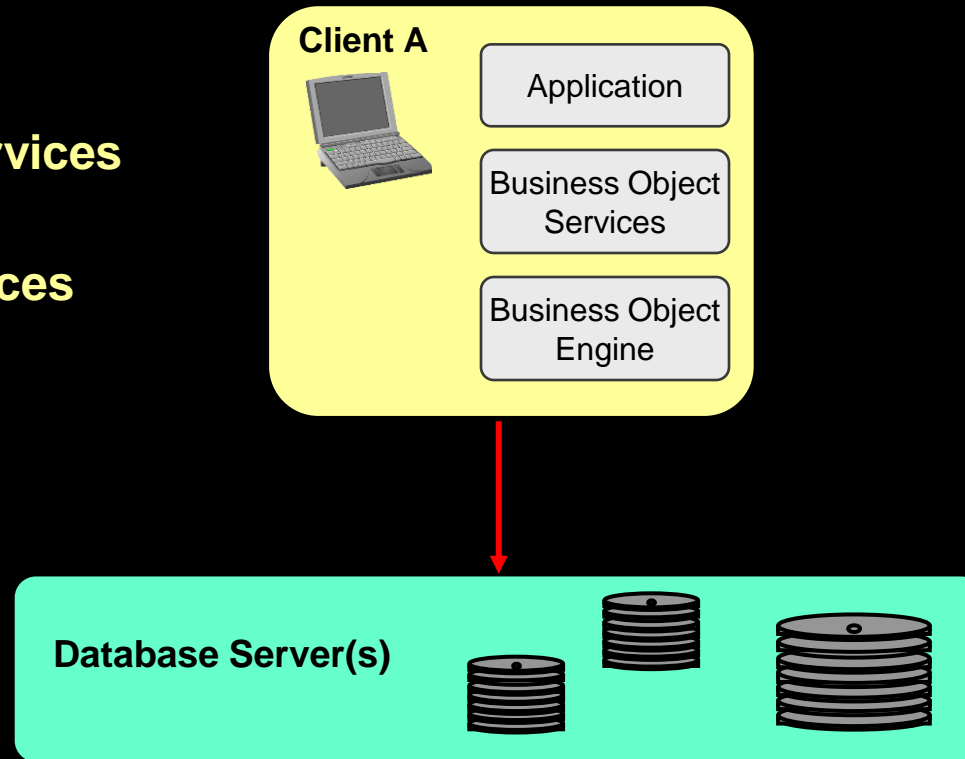


Client/Server: “Fat Client” Architecture

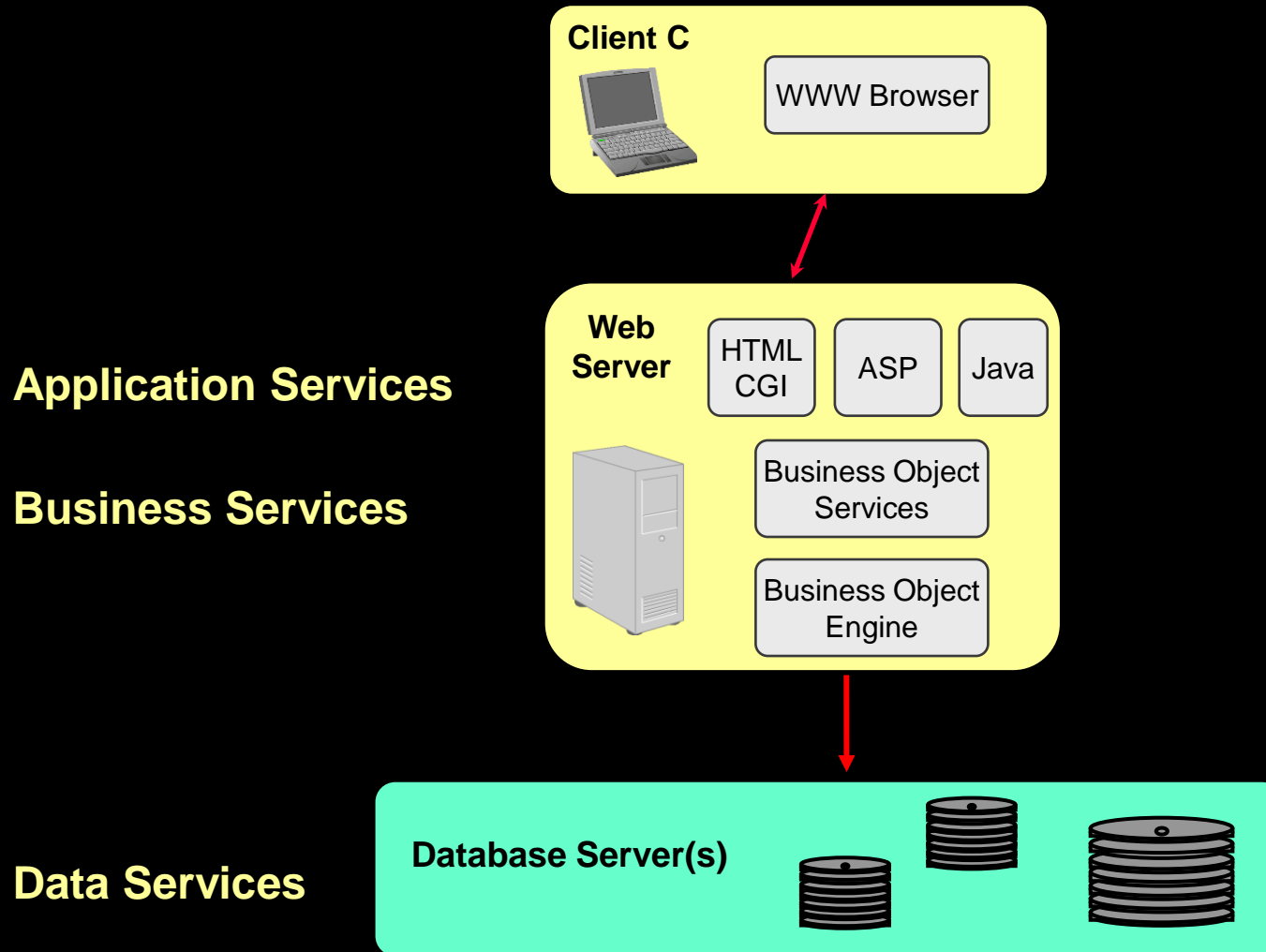
Application Services

Business Services

Data Services



Client/Server: Web Application Architecture

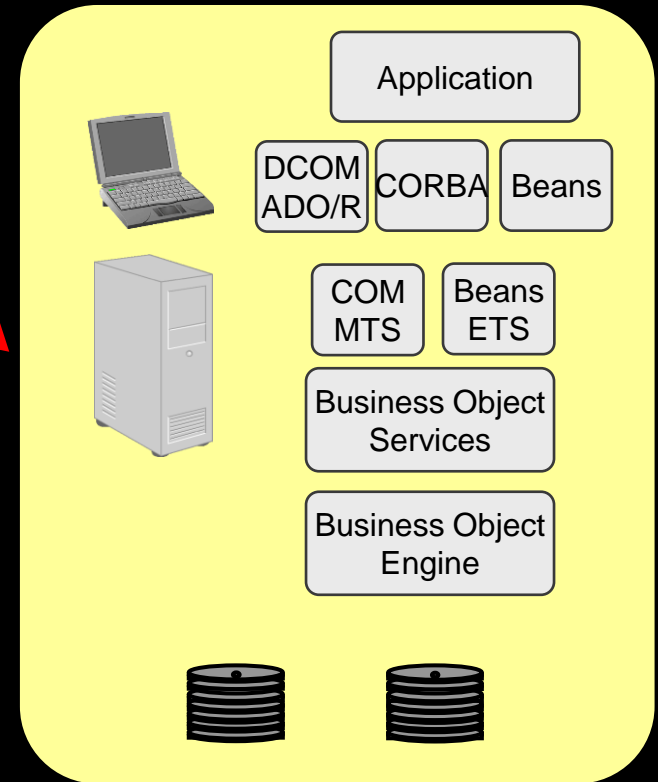
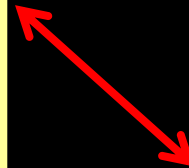
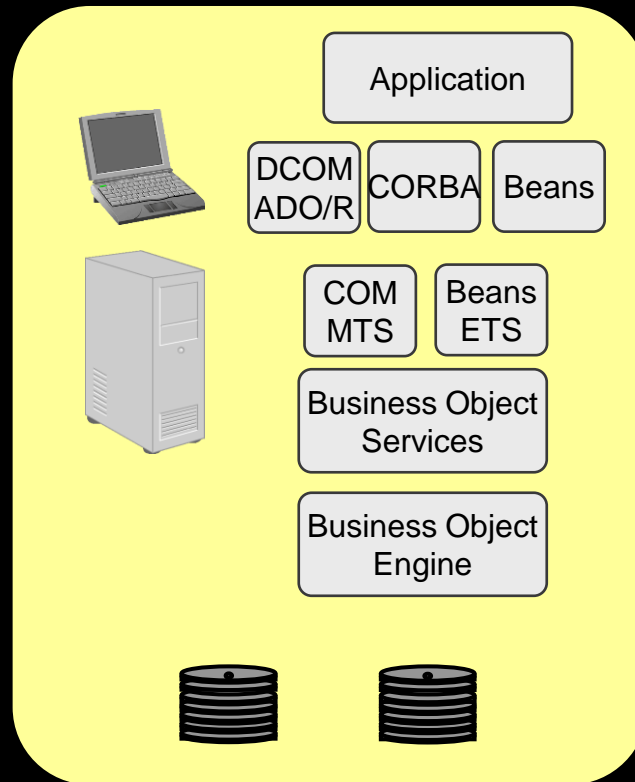


Peer-to-Peer Architecture

Application Services

Business Services

Data Services

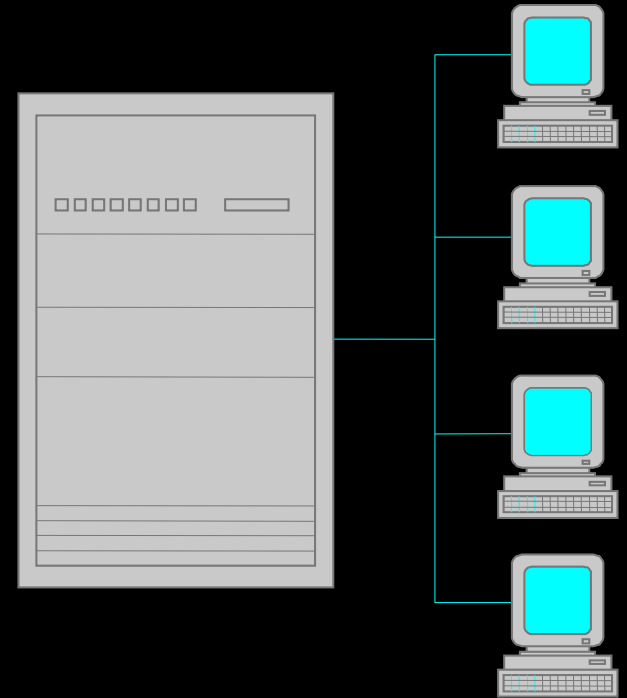


Các bước thực hiện

- ♦ Xác định cấu hình mạng tính toán
- ♦ Phân phối tiến trình lên các nút tính toán
- ♦ Định nghĩa các kỹ thuật phân tán

Cấu hình mạng

- ◆ End-user workstation nodes
- ◆ "Headless" processing server nodes
- ◆ Special configurations
 - Development
 - Test
- ◆ Specialized processors
- ◆ Network



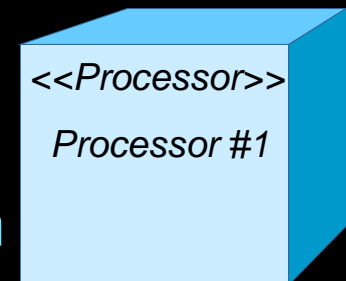
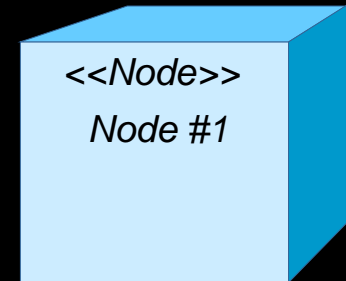
Mô hình hóa

♦ Node

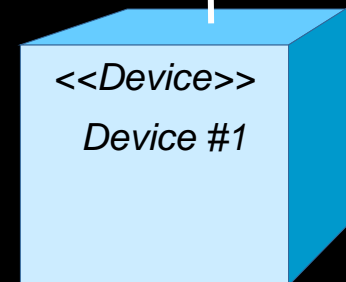
- Tài nguyên tính toán/xử lý
- Processor node
 - Thực thi phần mềm
- Device node
 - Thiết bị hỗ trợ
 - Thường được điều khiển bằng một tiến trình

♦ Connection

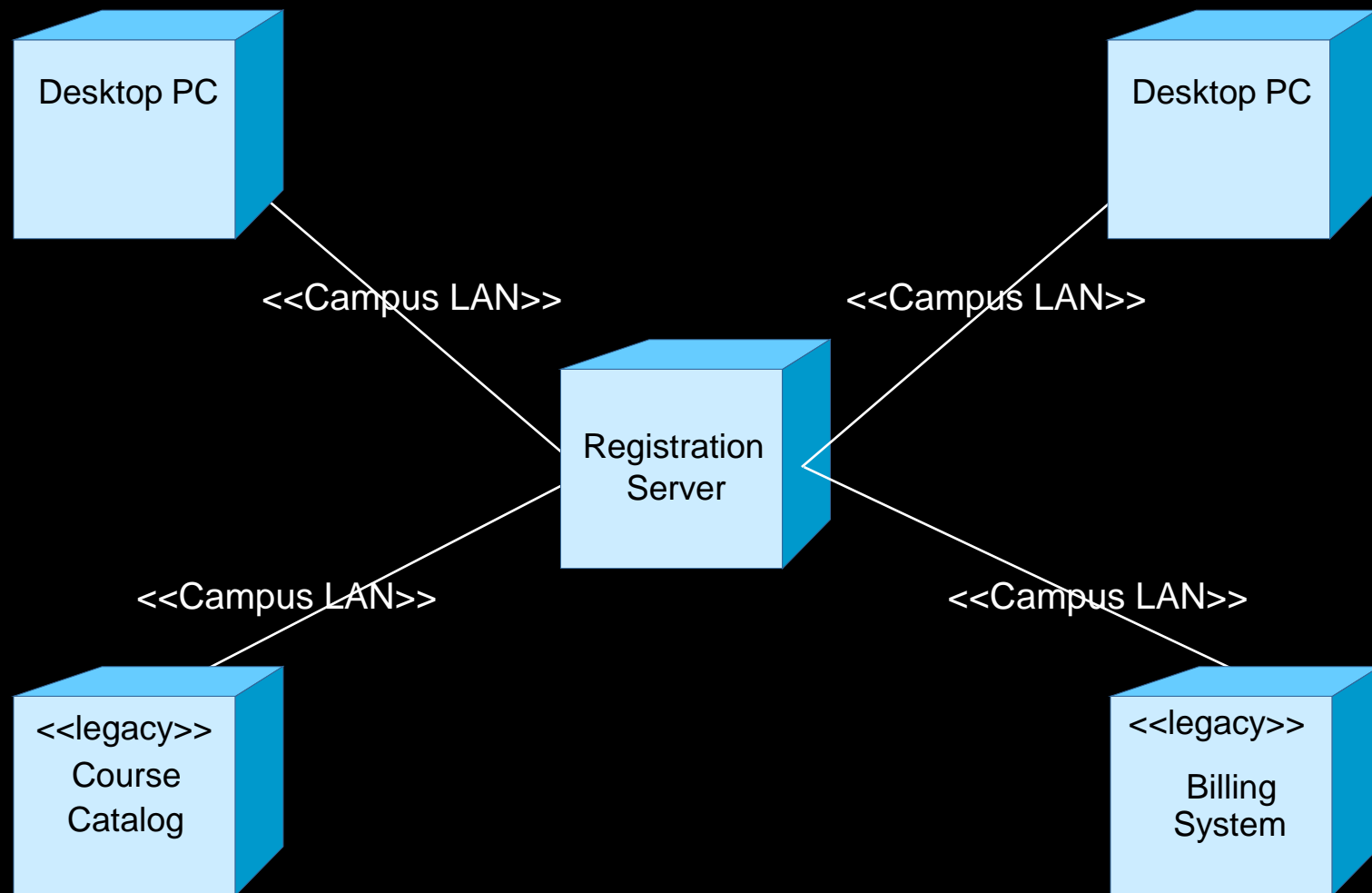
- Phương tiện giao tiếp giữa các node
- Mạng vật lý
- Giao thức phần mềm



Connection



Ví dụ

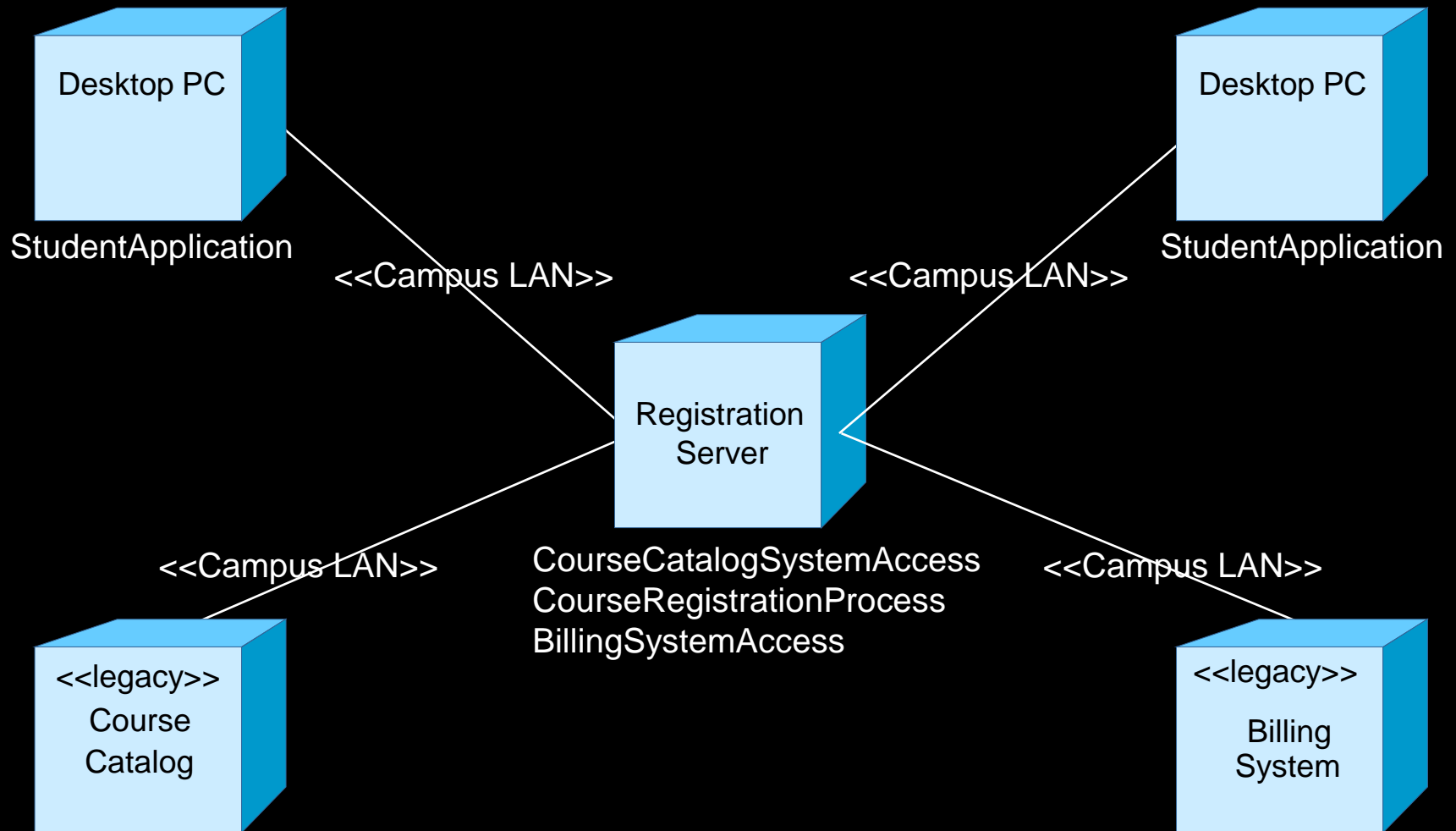


Phân phối tiến trình lên các nút tính toán

- ◆ Dựa theo mẫu
- ◆ Theo thời gian đáp ứng
- ◆ Theo nhu cầu tối ưu băng thông mạng
- ◆ Theo năng lực của các nút tính toán
- ◆ Communication medium bandwidth
- ◆ Theo hạ tầng hiện có

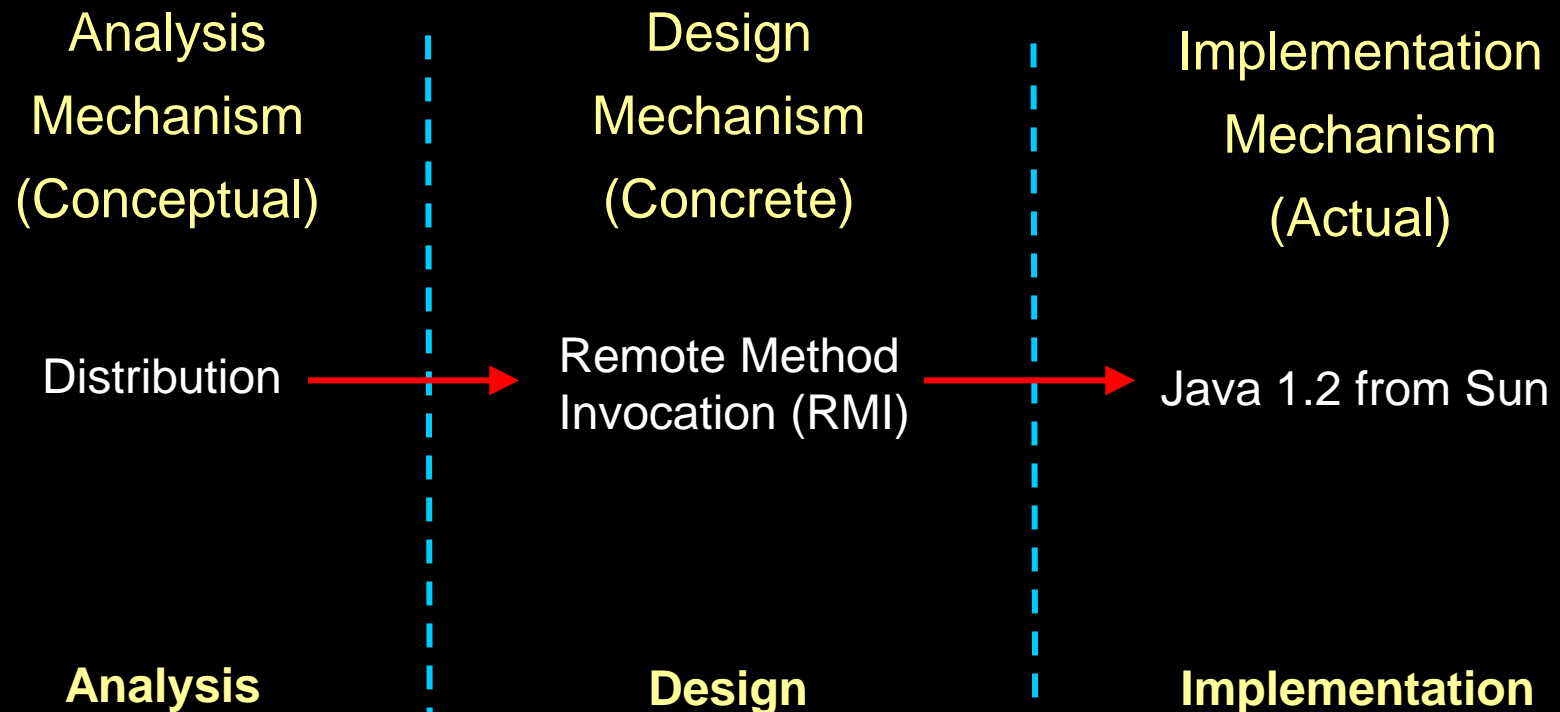


Ví dụ



Kỹ thuật phân tán

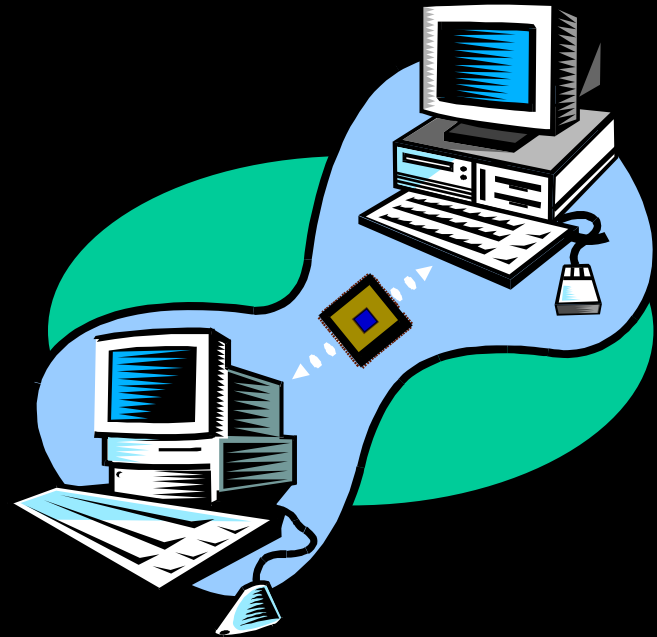
♦ Ví dụ: RMI



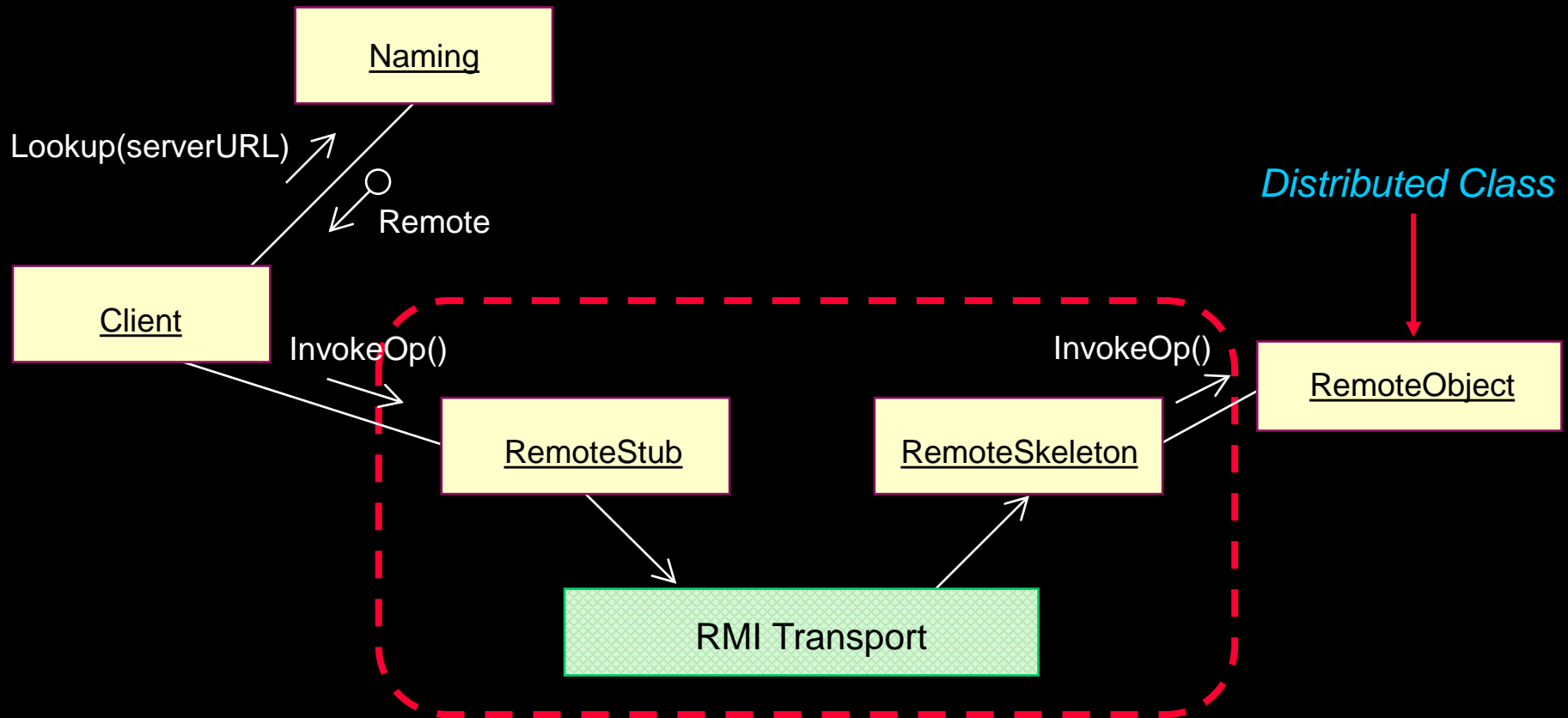
Kỹ thuật thiết kế: Phân tán: RMI

◆ Đặc tính

- Latency
- Synchronicity
- Message Size
- Protocol

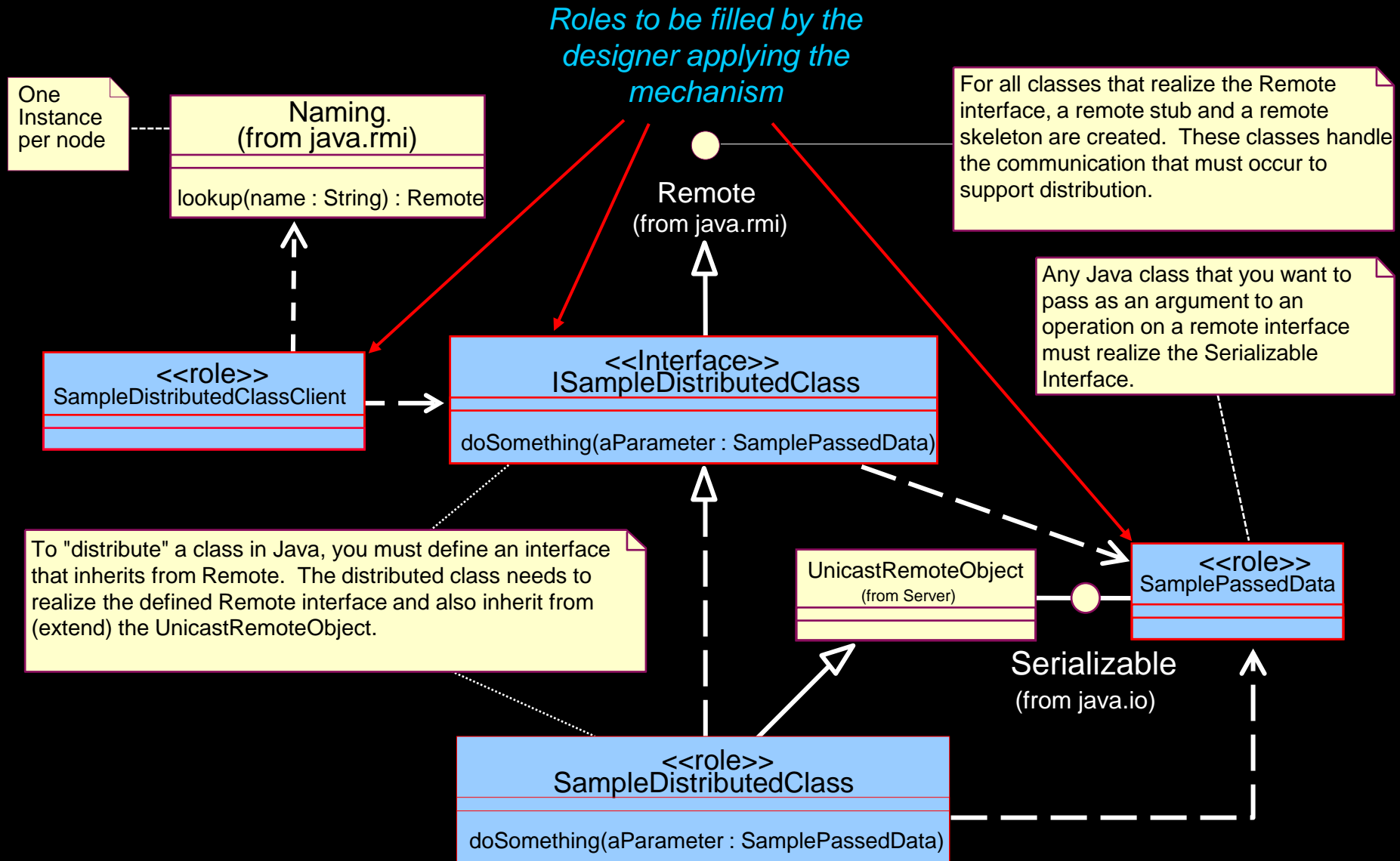


Remote Method Invocation (RMI)



Provided “for free” with RMI for each distributed class

RMI (cont.)



Các bước tích hợp RMI vào bản thiết kế

- ♦ Đóng gói RMI support classes (vd., Remote, Serializable interfaces, Naming Service)
 - *java.rmi và java.io package trong Middleware layer*
- ♦ Với mỗi lớp được phân tán:
 - *Controllers đặt trong Application layer*
 - *Phụ thuộc từ Application tới Middleware layer là cần thiết cho sự truy nhập java packages*
 - Định nghĩa giao diện cho lớp hiện thực hóa Remote
 - Định nghĩa lớp kế thừa của UnicastRemoteObject


(continued)

Các bước tích hợp RMI vào bản thiết kế

- ♦ Định nghĩa lớp cho các dữ liệu được chuyển giao phân tán bằng cách hiện thực hóa Serializable interface
 - *Core data types đặt trong Business Services layer*
 - *Phụ thuộc từ Business Services layer đến Middleware layer là cần thiết để truy nhập java.rmi*

(continued)

Incorporating RMI: Steps (cont.)

- ◆ Have distributed class clients lookup the remote objects using the Naming service
 - *Most Distributed Class Clients are forms*
 - *Forms are in the Application layer*
 - *Dependency from the Application layer to the Middleware layer is needed to access java.rmi*
 - Add relationship from Distributed Class Clients to Naming Service
 - ◆ Create and update interaction diagrams with distribution processing
- 
- Deferred

Ví dụ

