

# Enterprise Patterns

**Ching-Lin Yu**  
**Mozilla Taiwan**

# Introduction to Enterprise Systems

2

- Examples of enterprise systems
  - ▣ payroll, transaction processing, shipping tracking, accounting, foreign–trade exchange, etc.
- Characteristics
  - ▣ Emphasis on data
    - Persistent data
    - High volume of data
    - High complexity of data
    - Concurrent access of data
    - Lots of user interface screens to handle data

# Introduction to Enterprise Systems

3

- ▣ The need to **integrate** with other enterprise applications
- ▣ Different definitions and concepts with data between different departments/systems
- ▣ Complex business logic
  - may be political and illogical, but they are the rules of doing business

# Consider Beyond Functional Requirements

4

- Nonfunctional requirements have to be considered and addressed in designing enterprise systems:
  - ▣ Extensibility
  - ▣ Interoperability
  - ▣ Performance
  - ▣ Reliability
  - ▣ Security
  - ▣ Usability
  - ▣ etc.

# Architectural Considerations

5

- Layering
  - ▣ Divide and conquer a complicated system
  - ▣ Higher layers make use of lower layers
    - but not vice versa
  - ▣ Performance and scalability considerations
  - ▣ Principal layers:
    - Presentation: user interface
    - Domain: logic of the problem domain
    - Datasource: database, messaging systems or other remote systems

# Architectural Considerations

6

- Concurrency
  - ▣ often the most tricky aspect of the system
  - ▣ the system should act correctly on concurrent accesses
    - no deadlocks, corrupted data, lost updates, etc.
  - ▣ often provided in
    - database
    - hand-written concurrency control code

# Architectural Considerations

7

- Distribution strategies between layers
  - ▣ no distribution: for simple systems
  - ▣ inter-process communications (IPC)
    - remote method call style (Java RMI, .Net remoting, RPC in C, Facebook's thrift RPC)
    - serialize yourself (via TCP or HTTP)
    - message-oriented middleware
  - ▣ Decide what to transfer
    - more data/more frequent transfers lead to more performance degradation
    - should be minimized

# Enterprise/Cloud Computing Patterns

8

- Resource preparation
  - ▣ Singleton instance
  - ▣ Prototype images
- Architecture
  - ▣ n-Tier web pattern
  - ▣ adapter
  - ▣ facade
  - ▣ proxy & balancer
  - ▣ heartbeat
  - ▣ Map/Reduce

# Enterprise/Cloud Computing Patterns

9

- Behaviors
  - ▣ Queuing
  - ▣ Observer/Publish Subscribe
  - ▣ Command

# Resource Preparation

10

- Determines how your OS and base system play a role in your application
- Shows how to prepare your virtual images so that you can provide virtual instances on demand

# Singleton Instance

11

- What it is
  - ▣ The instance (real host, virtual machine, or software service) that is a **singleton**
  - ▣ There is only one instance in the environment
- When to use
  - ▣ To ensure only one copy of your software is running
  - ▣ Consistency is more important than reliability, scalability and performance
  - ▣ The load is low that the system can handle

# Singleton Instance

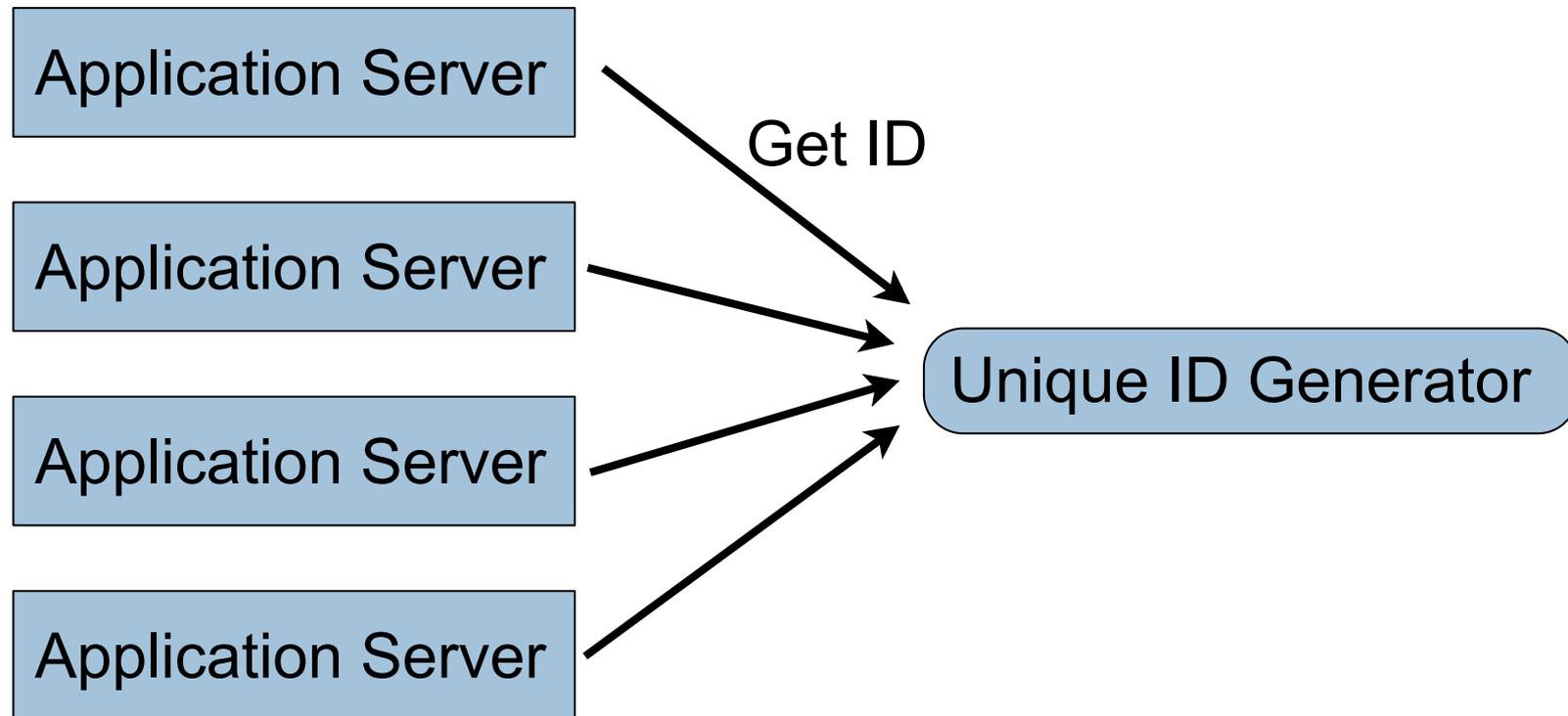
12

- Make preparations to ensure the system downtime is minimized
  - Backups
  - Standby instance

# Singleton Instance

13

## □ Example



# Prototype Images

14

- What it is
  - ▣ A VM (virtual machine) image that serves as the **prototype**
  - ▣ When new VM instance is needed, it is cloned from the prototype image
  - ▣ The prototype image is built with software packages and configuration data common to the instances

# Prototype Images

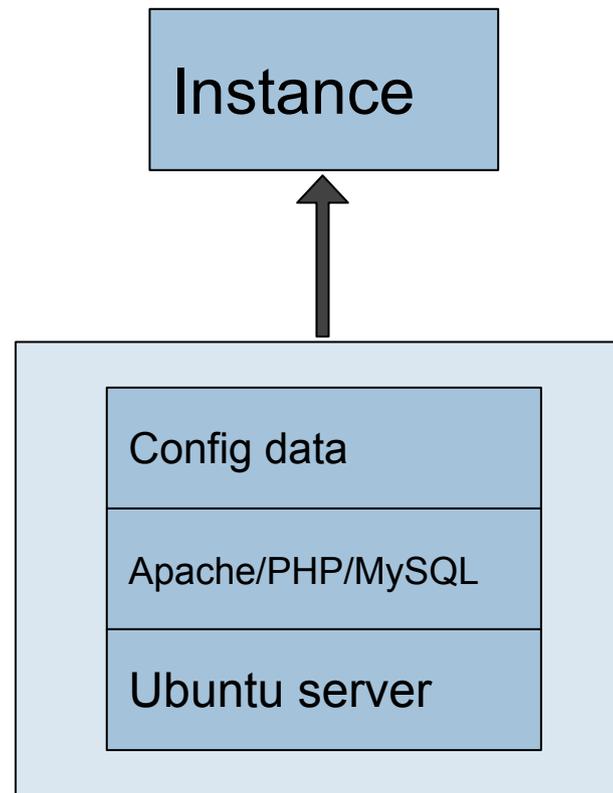
15

- When to use
  - ▣ The basic principle of prototype: when cloning is cheaper than building from scratch
  - ▣ You want to maintain a single copy of the image for multiple purposes
  - ▣ You want to apply updates when launching a new instance
  - ▣ You want to provide expandability to your system by creating new VM instances

# Prototype Images

16

## □ Example



# Architectures

17

- Determines how your application works with the rest of the world
- Determines how to spread and delegate the requests to your instances

# N-Tier Web Pattern

18

- What it is
  - ▣ An architecture that helps your application to scale vertically or horizontally
  - ▣ By dividing the application into different modular and swappable tiers (layers)
  - ▣ Typically has presentation, application, and database tiers
    - mapped to view, controller, and model in the MVC pattern

# N-Tier Web Pattern

19

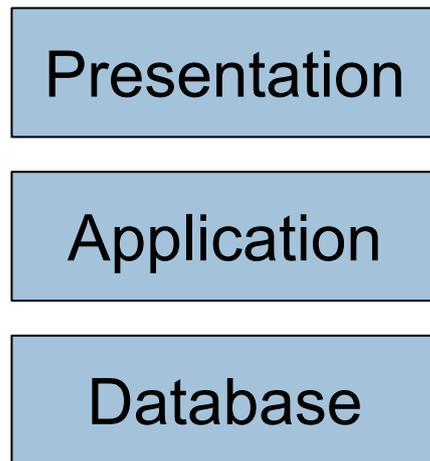
- When to use
  - ▣ When building a modular web application
  - ▣ When providing multiple interfaces to your application
  - ▣ When integrating multiple systems with each other

# N-Tier Web Pattern

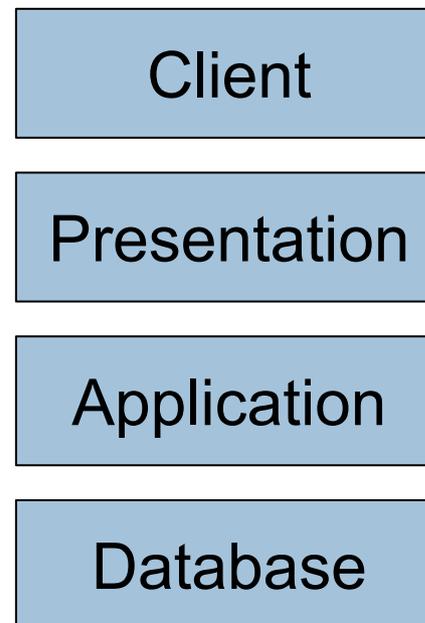
20

## □ Conceptual design

Typical 3 tiers



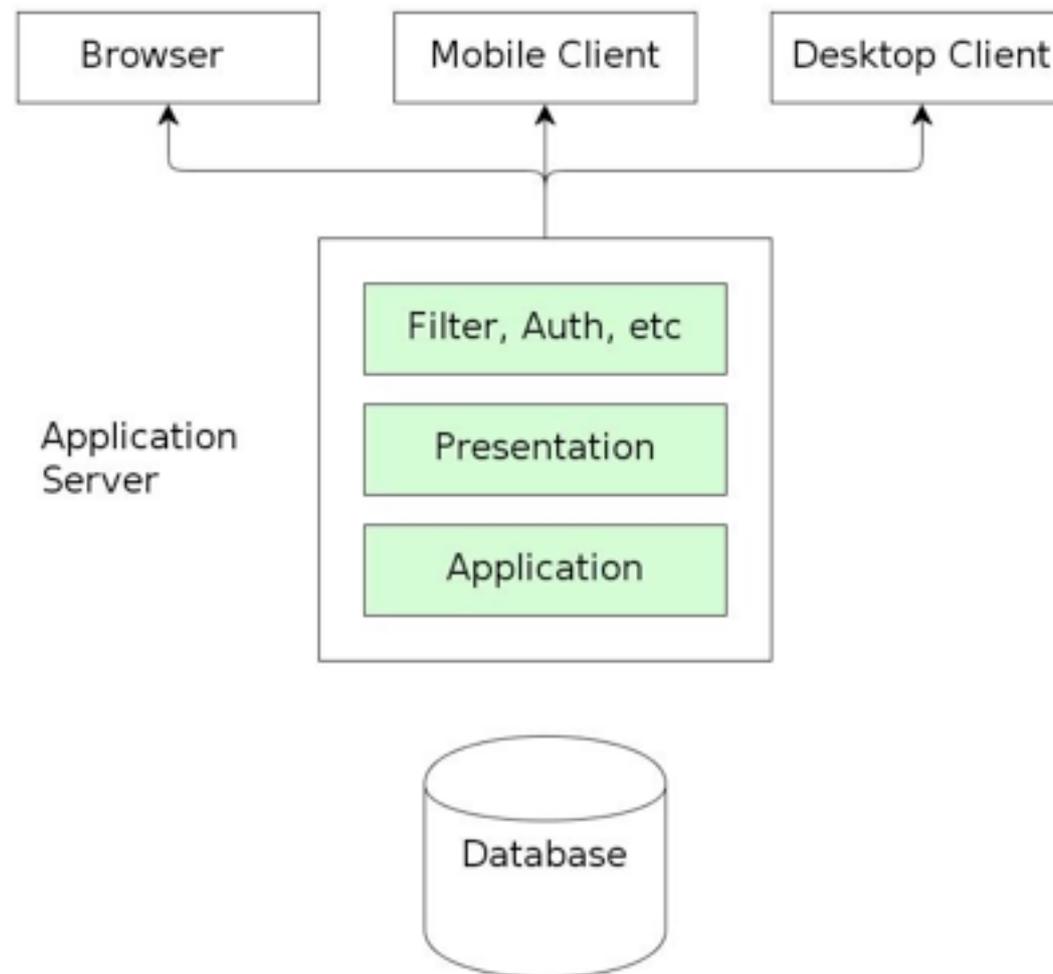
Adding rich client



# N-Tier Web Pattern

21

## □ Physical deployment



# Adapter

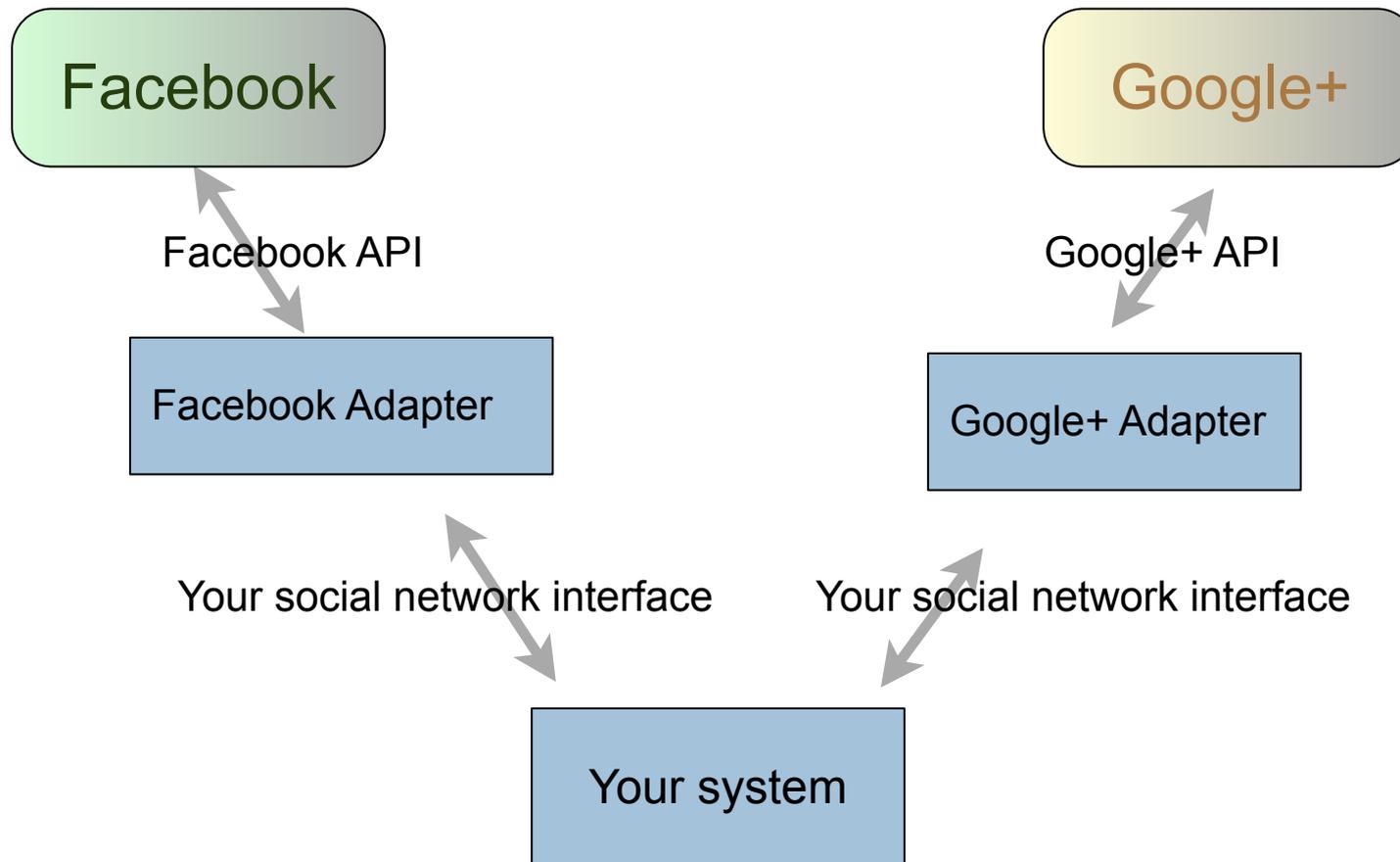
22

- What it is
  - ▣ The converter that converts another system's interface to what you expect
- When to use
  - ▣ When you need to interop with multiple systems
  - ▣ And you don't want to 'speak' each system's language
  - ▣ It's better to adapt the outside API to your system than to build your whole system using the outside API

# Adapter

23

## □ Example



# Facade

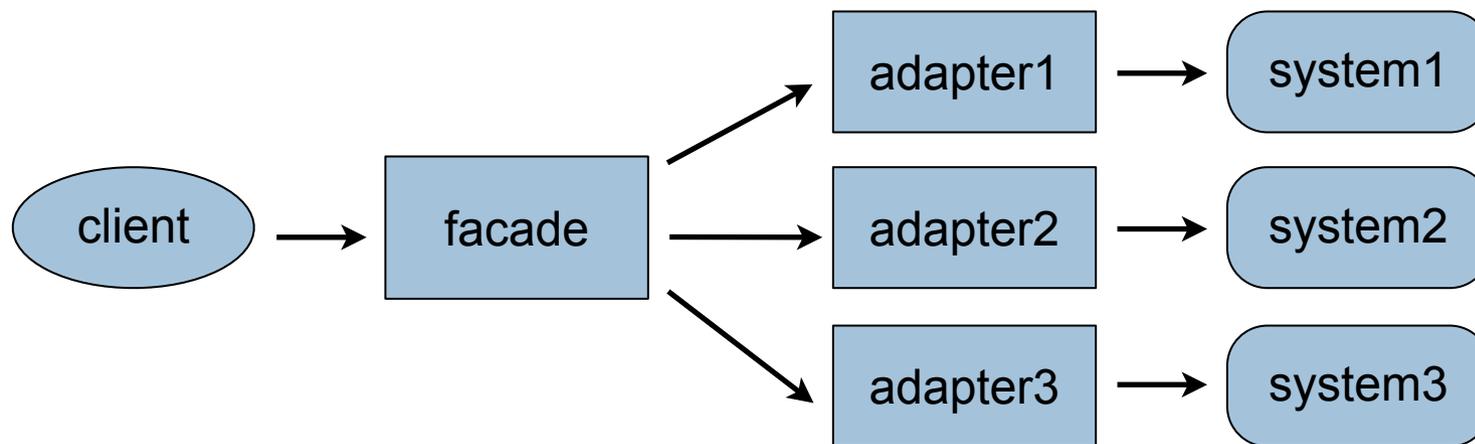
24

- What it is
  - ▣ A high level interface that unifies multiple interfaces
- When to use
  - ▣ When you want to provide a service that makes use of other services
  - ▣ For web-based applications, it's better to provide one-stop service than to redirect the user to different systems

# Facade

25

- Note: facade often makes use of adapters to interact with multiple systems

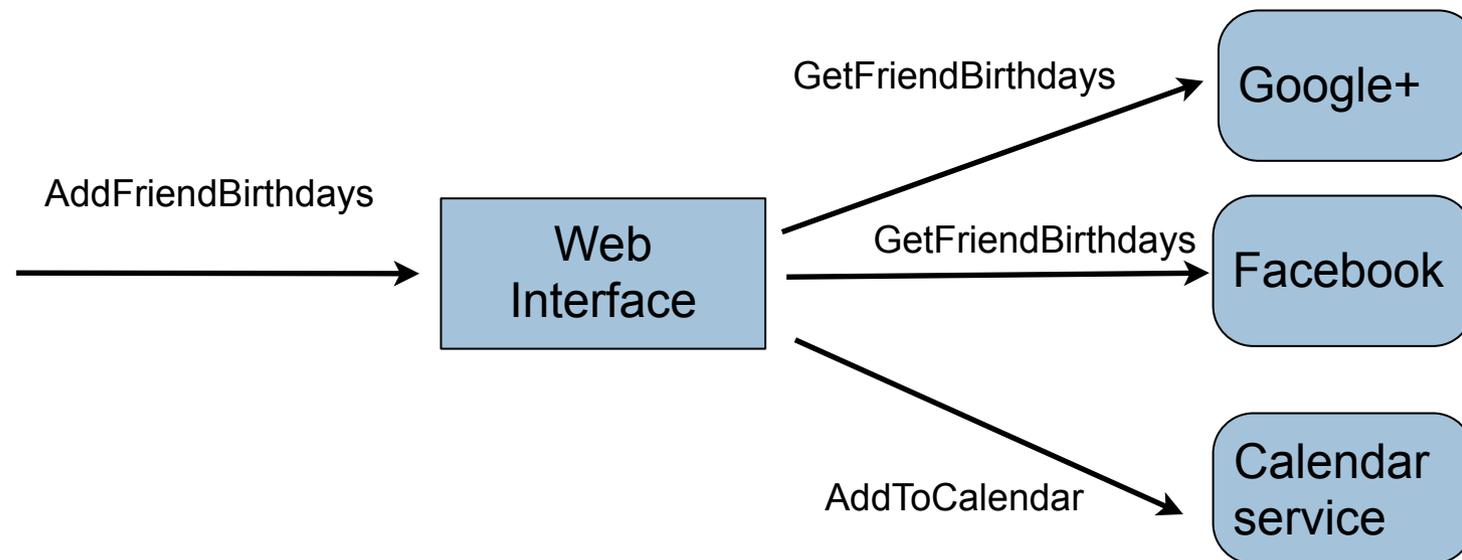


# Facade

26

## □ Example

- A new request: add friend birthdays to my calendar



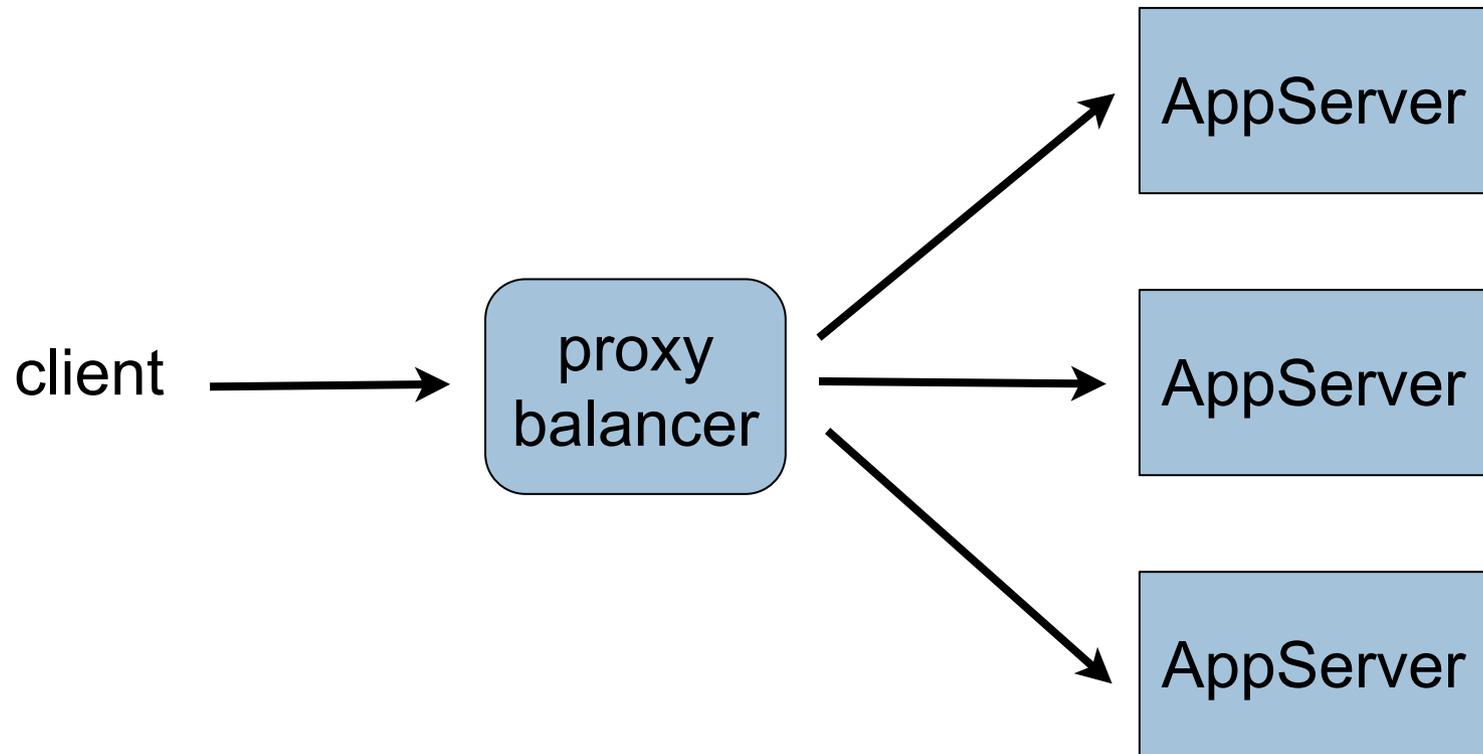
# Proxy & Balancer

27

- What it is
  - ▣ A host to transparently direct the requests to the application servers
  - ▣ To provide reliability and stability for your application
- When to use
  - ▣ You need to run multiple copies of the application for availability and performance
  - ▣ You need to scale your application seamlessly

# Proxy & Balancer

28



# Heartbeat

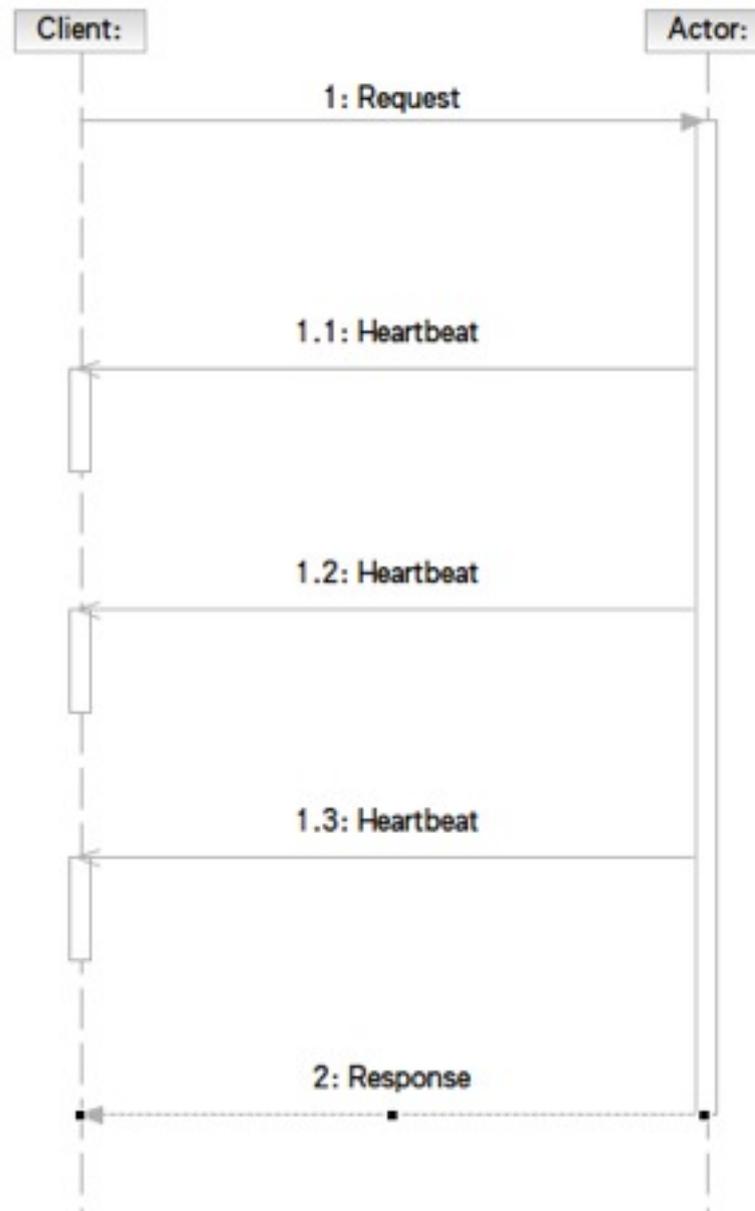
29

- What it is
  - ▣ A lightweight message the remote host sends back to the client periodically
  - ▣ Used to tell the client that the remote host is still alive
- When to use
  - ▣ You need to decide that the remote operation will not complete
  - ▣ The time to complete the operation is variable or not known a priori

# Heartbeat

30

## □ The pattern



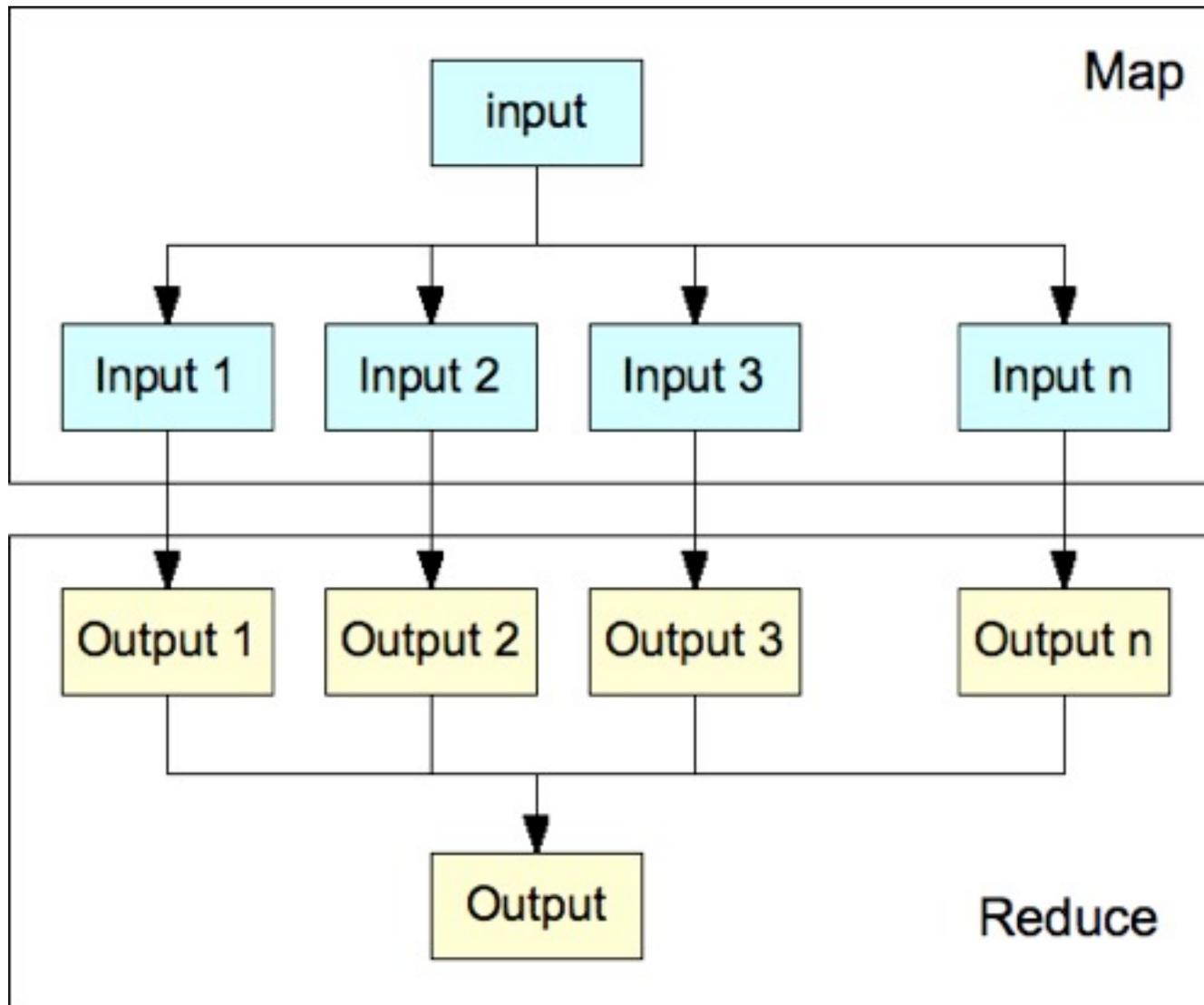
# Map/Reduce

31

- What it is
  - ▣ A divide-and-conquer pattern
    - Split a large task into small pieces that are manageable and distributable
    - Run the small pieces in parallel
    - Combine the processed pieces into the final result
- When to use
  - ▣ You have a large input set to process
  - ▣ The input is splittable
  - ▣ The data need to be processed quickly

# Map/Reduce

32



# Map/Reduce

33

- Implementations
  - Google's services
  - Apache Hadoop
  - Nokia Disco
  - Mapreduce.Net
  - Skynet

# Behaviors

34

- Determine object interactions in the system with exterior systems
- Determine how to execute actions on data
  - ▣ Synchronous or asynchronous
- Reduce coupling in the system

# Queuing

35

- What it is
  - ▣ The subsystems (colleagues) communicate with each other using the message queue system
  - ▣ A subsystem sends a message to the queue for requesting services
  - ▣ Another subsystem receives and removes the message to handle the request
  - ▣ Asynchronous request handling
  - ▣ Supports multicast requests
  - ▣ Is fault tolerant

# Queuing

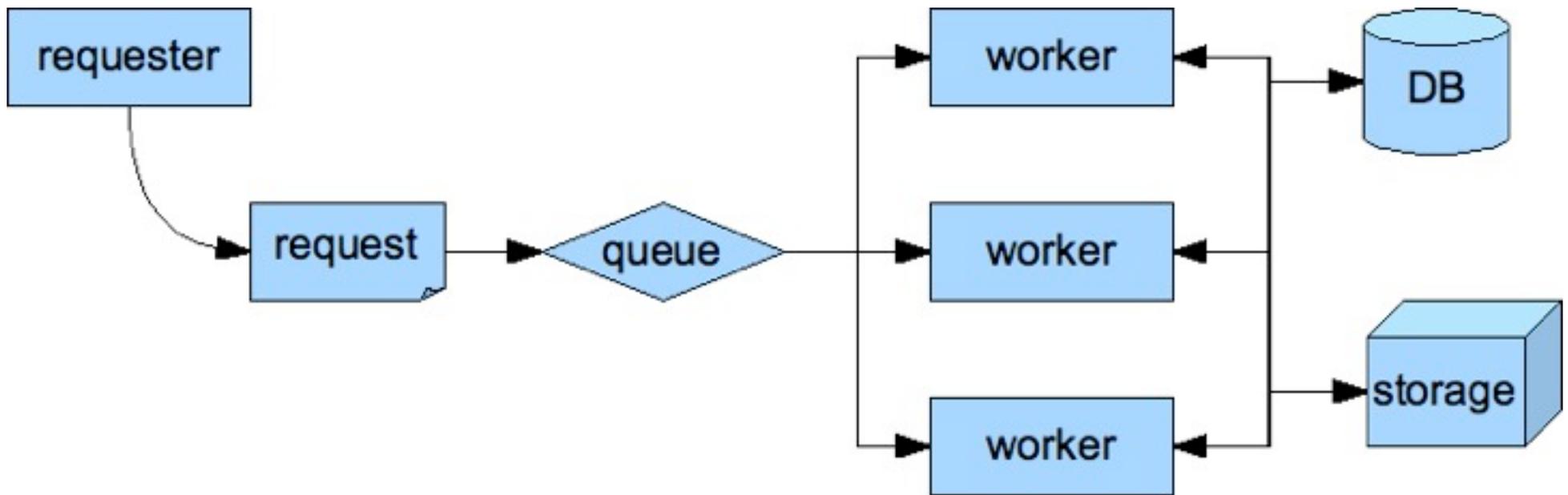
36

- When to use
  - ▣ You need fault tolerance for requests
  - ▣ You need to scale by adding workers at runtime
  - ▣ You need to decouple and hide the request sender and receiver

# Queuing

37

## □ The pattern



# Observer/Publish-Subscribe

38

- What it is
  - ▣ The observer pattern implemented in a distributed environment
  - ▣ One-to-many messaging pattern
  - ▣ Publisher sends a message to the “topic”
  - ▣ The subscribers receive the message from the “topic”
  - ▣ Asynchronous in nature
  - ▣ Often provided by messaging middleware

# Observer/Publish-Subscribe

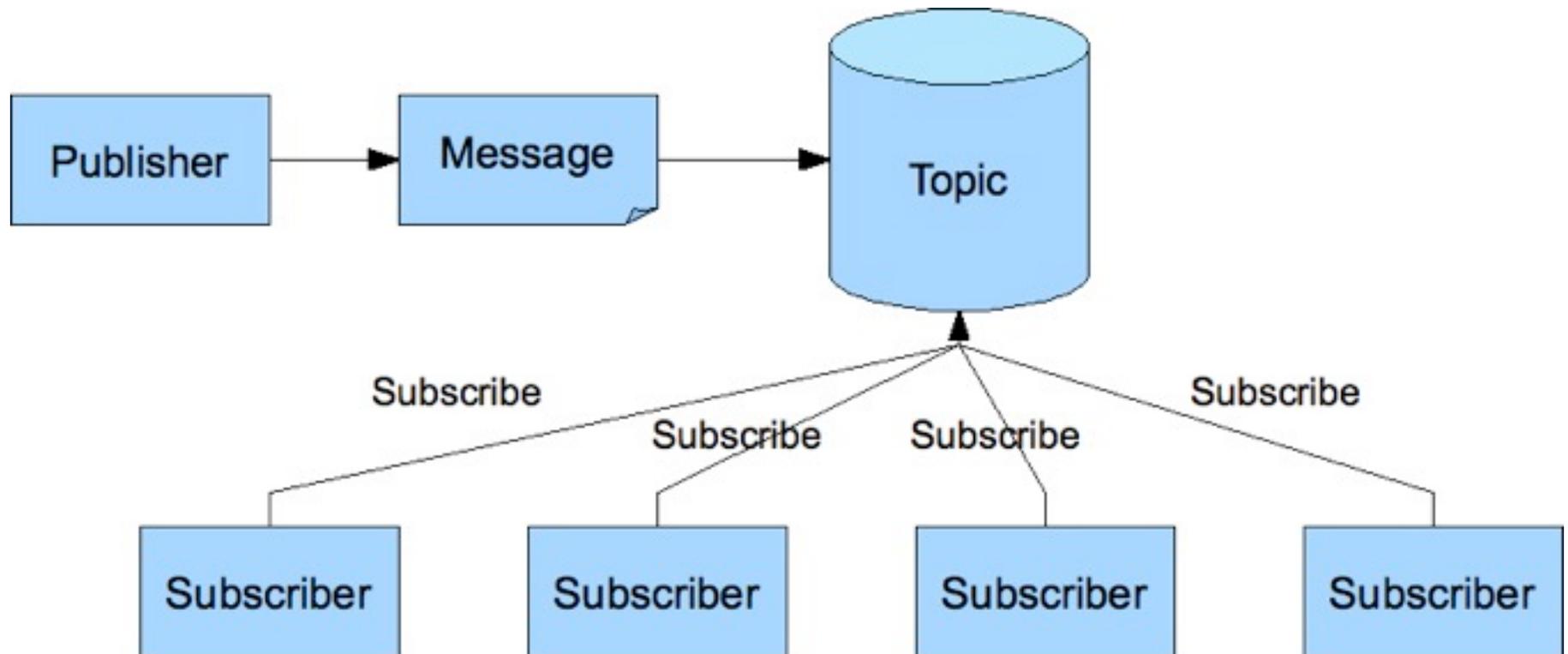
39

- When to use
  - ▣ You need a more scalable alternative to periodical polling of the remote service
  - ▣ Multicast messages must be delivered to the recipients
    - In a reliable manner
  - ▣ You need to decouple and hide the request sender and receiver

# Observer/Publish-Subscribe

40

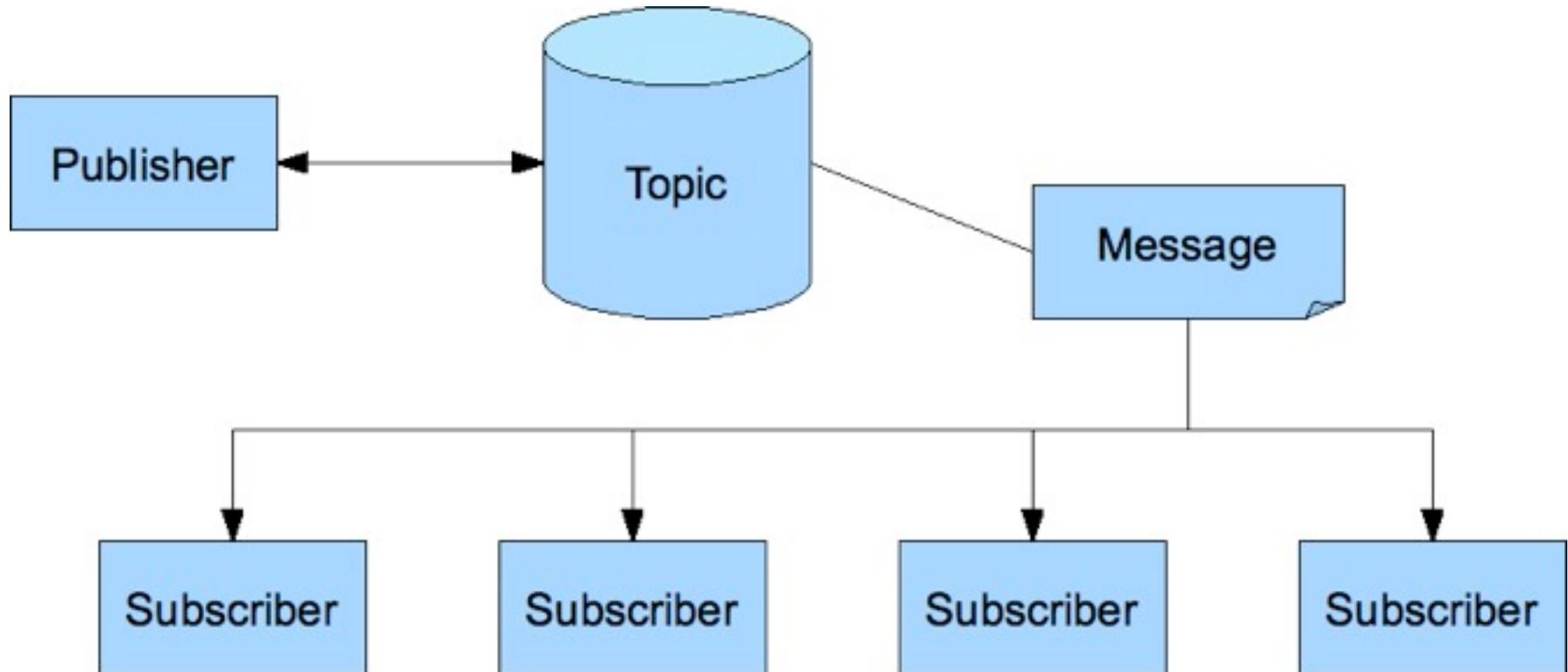
- Publisher sends the message



# Observer/Publish-Subscribe

41

- Each subscriber gets a copy of the message



# Command

42

- What it is
  - ▣ A request/action encapsulated as an object
  - ▣ The object can be passed to other hosts for execution
  - ▣ Supports queueing, logging, rollback operations (with the Memento pattern)
  - ▣ Often used with the queue pattern
    - To support high availability, scalability

# Command

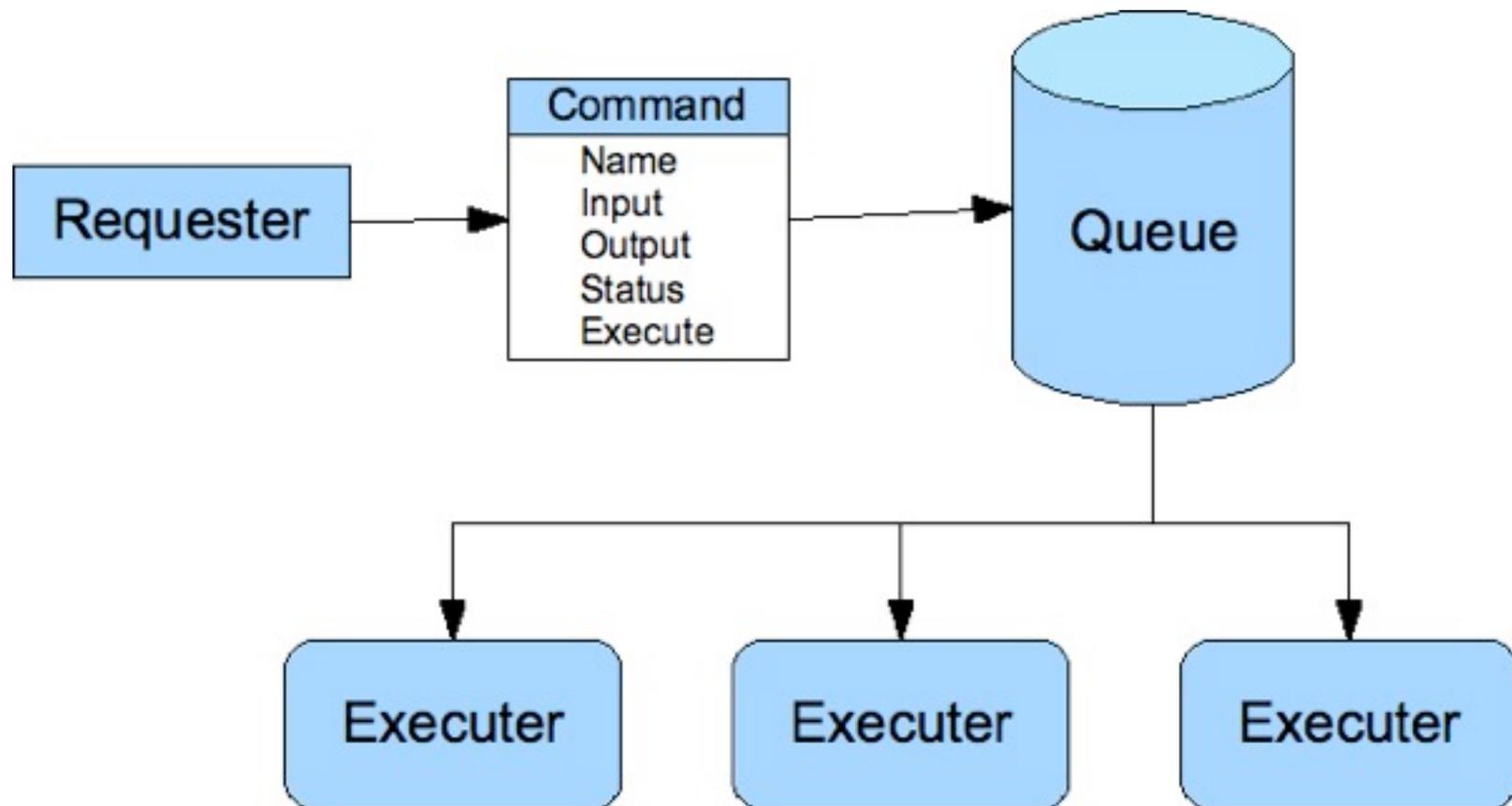
43

- When to use
  - ▣ You have multiple types of executable jobs to perform
  - ▣ You want to track statistics of the jobs
  - ▣ You want to specifically define how the job is executed

# Command

44

## □ The pattern



# Command

45

- Implementation
  - ▣ The capability of
    - serializing an object
    - transmitting the object over the network
    - optionally storing the object
    - deserializing the object on another host to process
  - ▣ Example
    - Java Hadoop cluster
      - Submit the action packaged as a jar file to the cluster

# Summary

46

- Characteristics and requirements of enterprise systems
- Architectural considerations
- Think beyond functional requirements
- GoF patterns applied in distributed/enterprise computing

# References

47

- Christopher M. Moyer, *Building Application in the Cloud: Concepts, Patterns and Projects*, Addison–Wesley, 2011
- Mark Grand, *Java Enterprise Design Patterns*, John Wiley & Sons, 2002
- Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison Wesley, 2002