ARCHITECTURAL STYLES

part 2

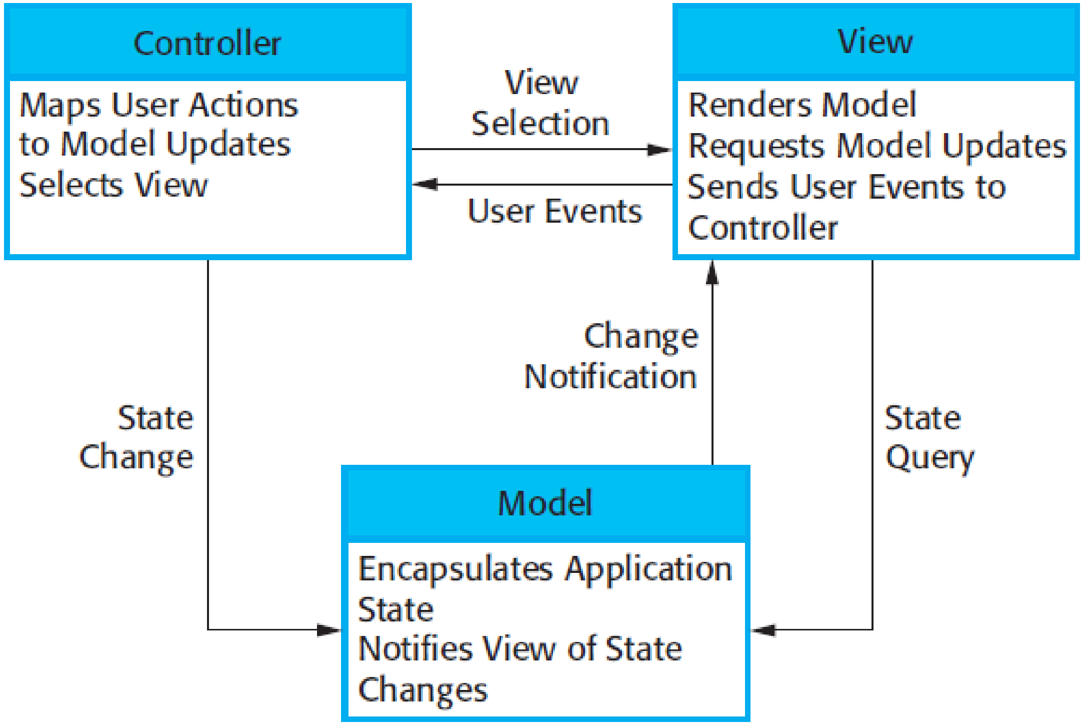
Reflection

- Architectural styles
  - Pipe-and-Filter
  - Client-server
  - Repository/Blackboard
  - Layered
- Today
  - Model-View-Controller
  - Implicit invocation/Message passing
  - Wrapper
  - Architectural styles in cloud

Model-view controller (MVC) style

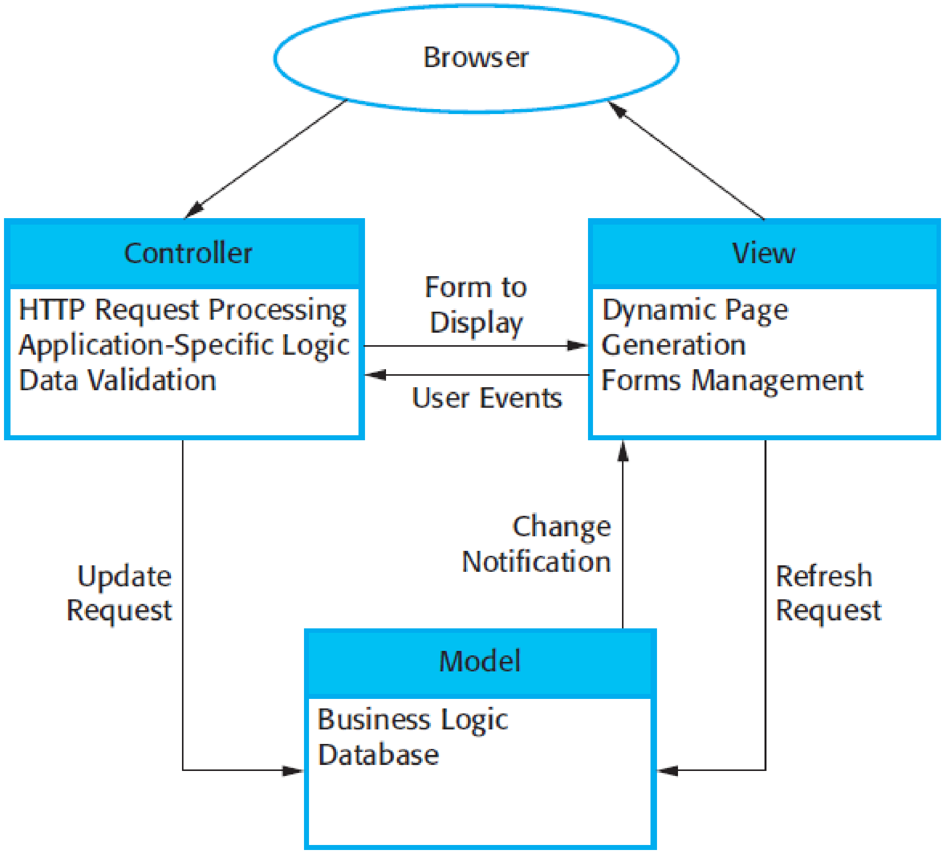
- Enables independence between data, presentation of data and user
- Model component represents knowledge. It manages the behavior and data of the application domain, sends information about its state (to the view), and responds to instructions to change state (usually from the controller)
- View has the duty to manage presentation of information to users
- Controller manages the interaction with the user (e.g. mouse clicks, key pressed, etc.) and informs the model or the view to take appropriate actions

MVC style

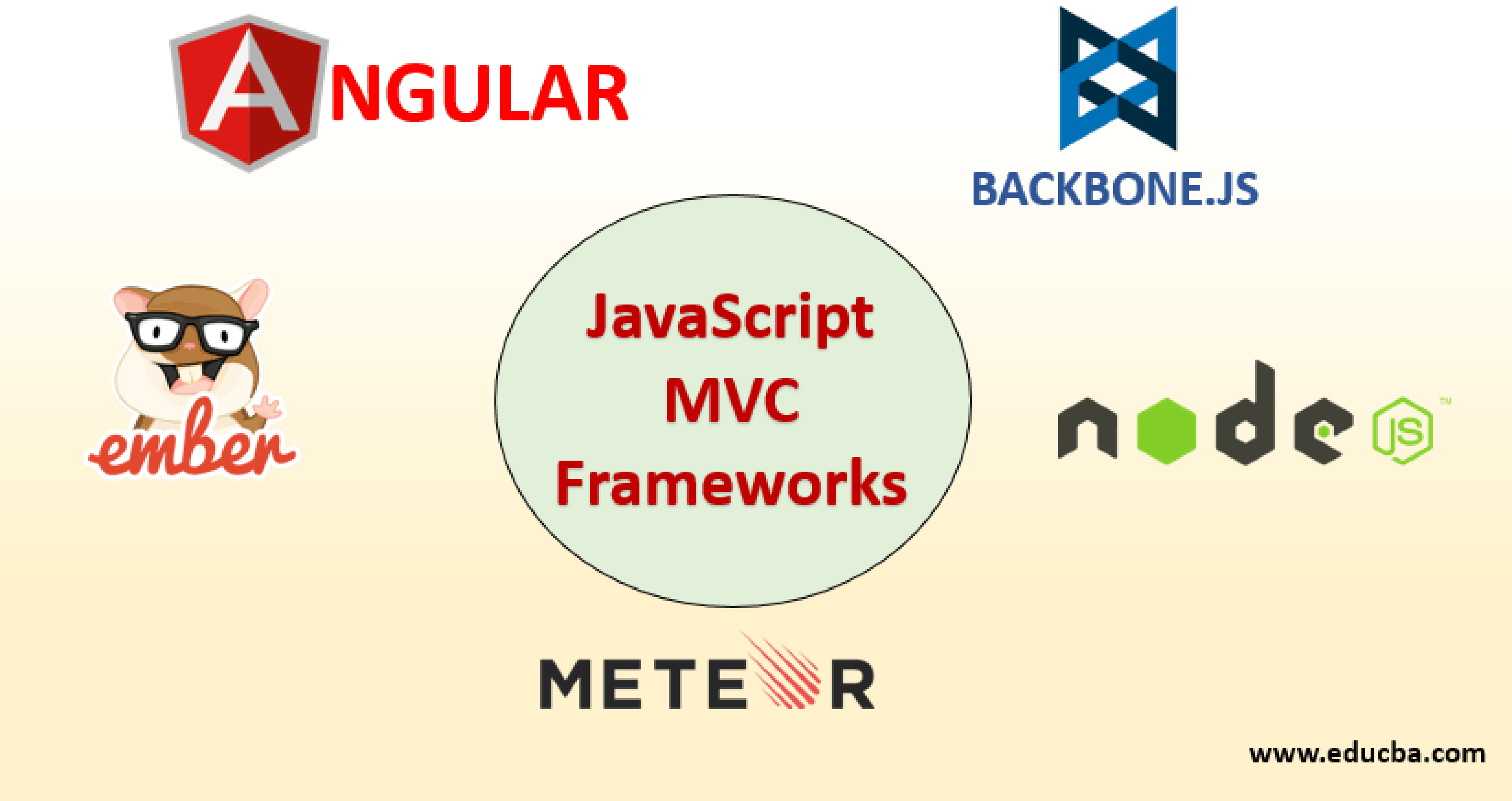


Software Engineering by Ian Sommerville, 9th edition (2010), Addison-Wesley Pub Co;

MVC - example



Software Engineering by Ian Sommerville, 9th edition (2010), Addison-Wesley Pub Co;


ANGULAR
BACKBONE.JS
ember
JavaScript MVC Frameworks
node JS
METEOR
www.educba.com

Advantages of MVC

- Great flexibility
  - Easy to maintain and implement future enhancements
  - Clear separation between presentation logic and business logic
  - Easier support for new types of users
- The view is separate and in most systems it undergoes a lot of changes

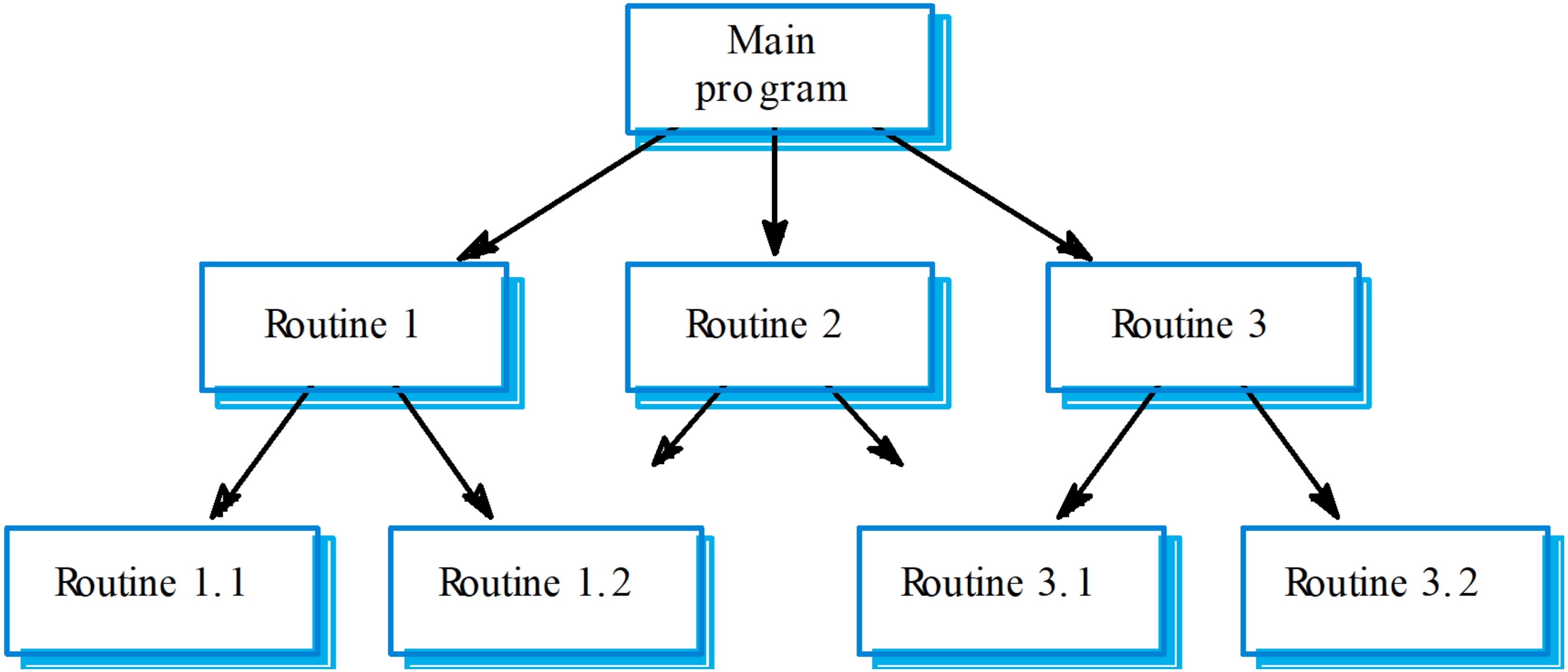
Disadvantages of MVC

- Even if data model is simple this style may introduce complexity and require a lot of additional code
  - Not suitable for small applications
- Performance issue when frequent updates in the model

Implicit invocation style

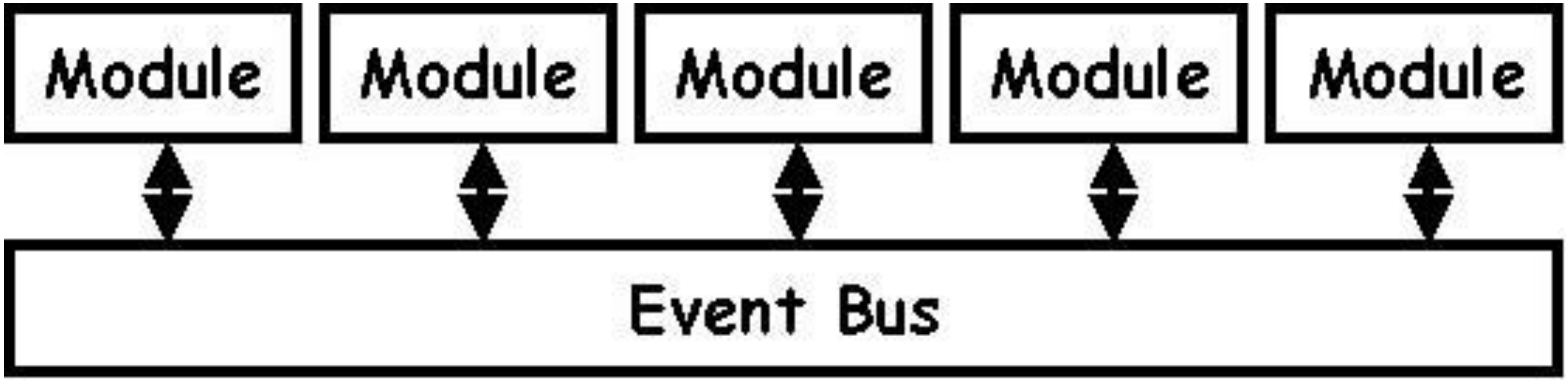
- Components within the system interact with each other by emission of events
- Events may contain not only control messages but also data
- Other names
  - Publish-subscribe
  - Event-based style
  - Message passing style

Explicit invocation

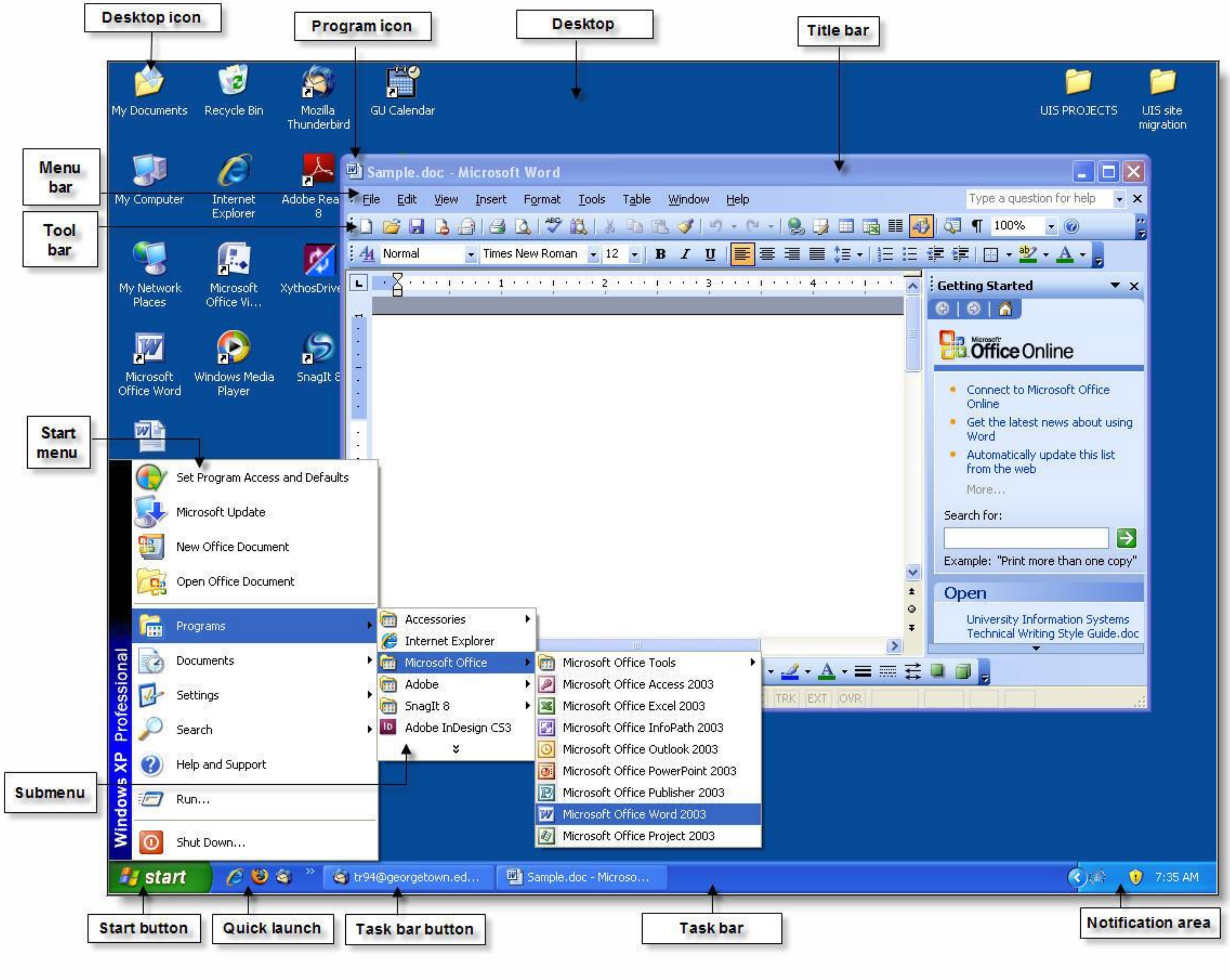


Implicit invocation style

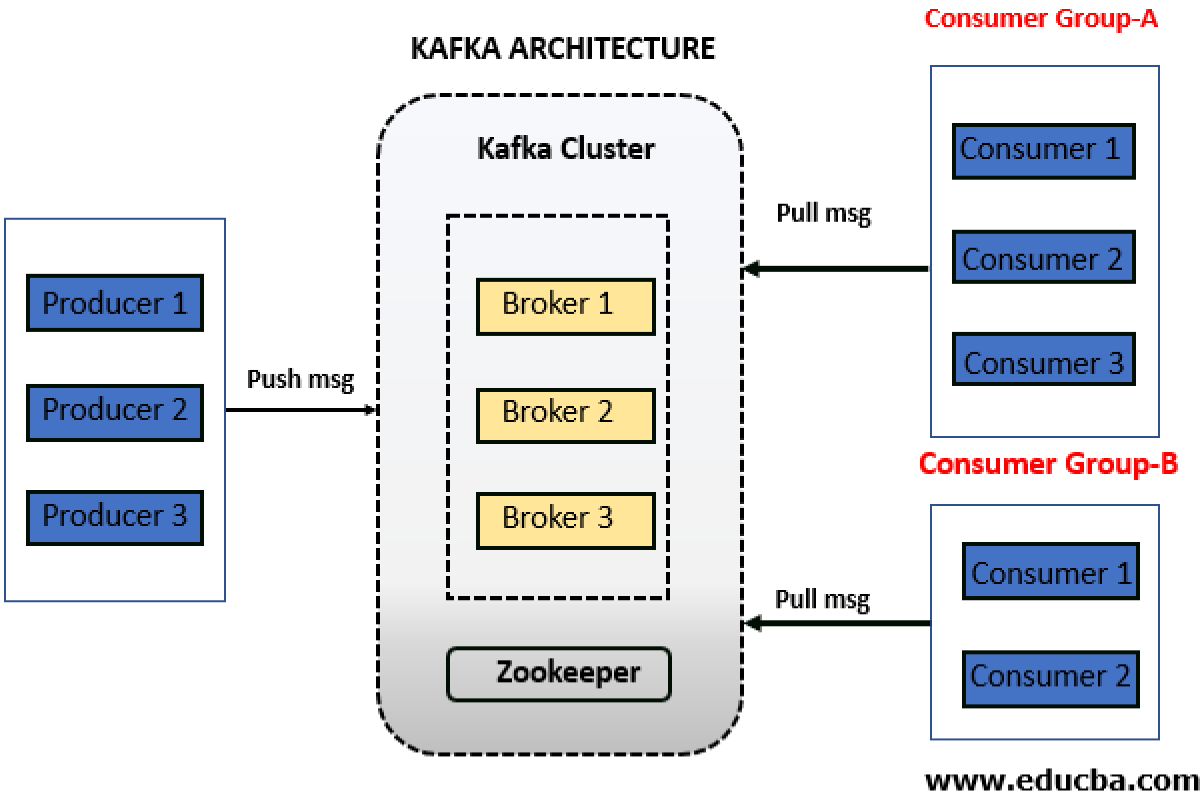
- Components of this style are running concurrently and communicate by receiving or emitting events
- Connector is an event bus
  - All component interact via the bus



Example of implicit invocation style



User interactions are passed to the application as events


KAFKA ARCHITECTURE
Consumer Group-A
Kafka Cluster
Consumer 1
Consumer 2
Consumer 3
Pull msg
Producer 1
Producer 2
Producer 3
Push msg
Broker 1
Broker 2
Broker 3
Consumer Group-B
Consumer 1
Consumer 2
Pull msg
Zookeeper
www.educba.com

Advantages of implicit invocation style

- Louse coupling
  - Components may be very heterogeneous
  - Components are easy to replace or reuse
- Big effectiveness for distributed systems - events are independent and can travel across the network
- Security – events are easily tracked and logged

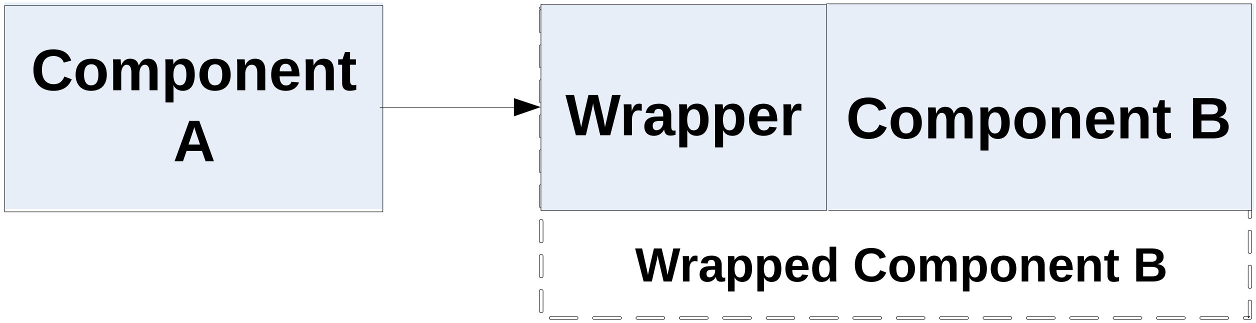
Disadvantages of implicit invocation style

- Vague structure of the system
  - Sequence of component executions is difficult to control
  - Hard debugging
- It is not sure if there exist a component to react to a given event
- Big amounts of data are difficult to be carried by events
- Reliability issue – malfunction of the event bus will bring the whole system down

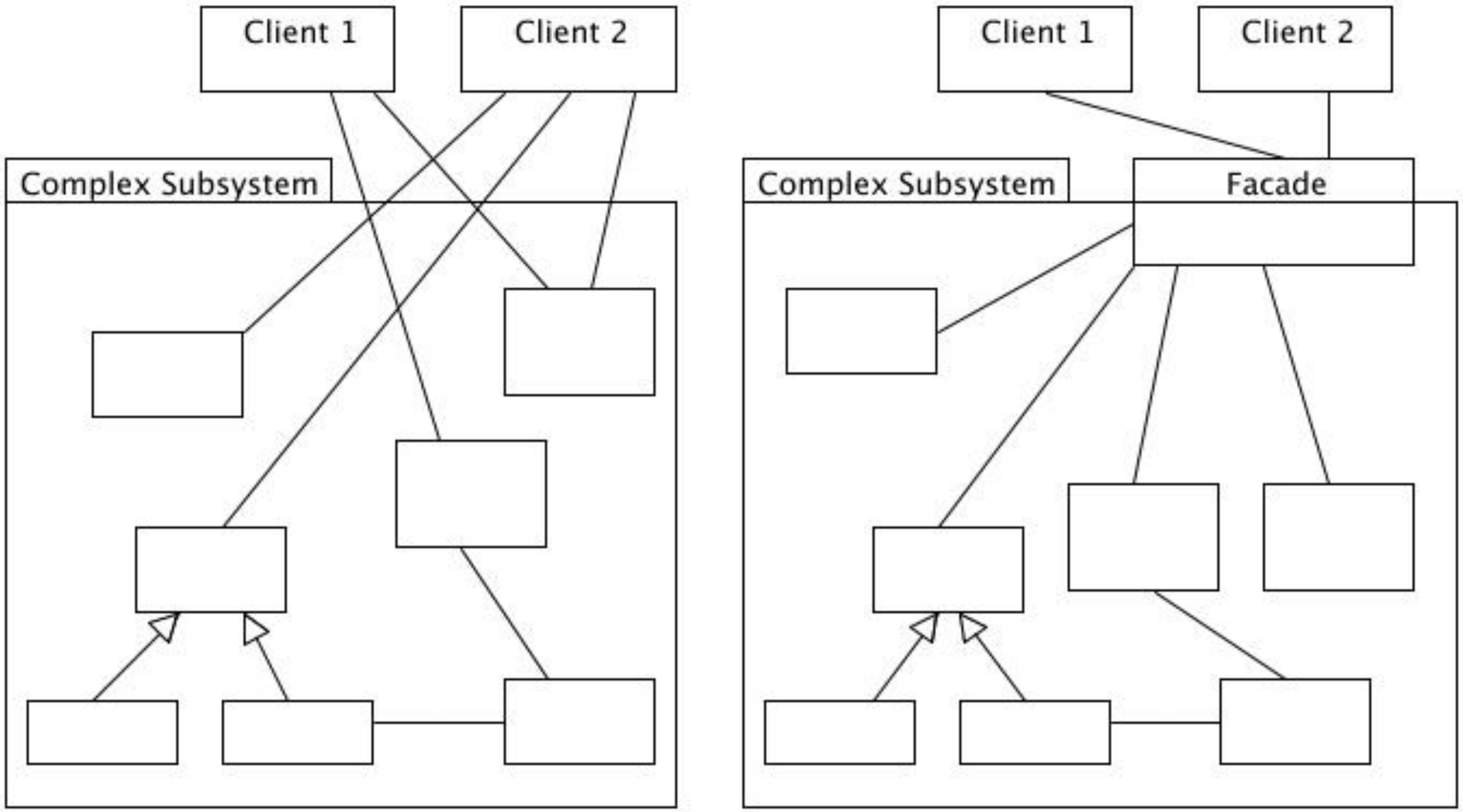
Wrapper style

- Introduce an additional component that acts as an intermediary between two interacting components
- Lets assume we have two components (client – A) and (server – B). Obviously A depends on B. This dependency should be minimized
- Different techniques exist, with respect to the nature of the dependency. They may have different names in literature, we call them here *wrapper* style

Wrapper

- Depending on its semantics may serve different purposes
  - Facade
  - Name server
  - Broker
  - 

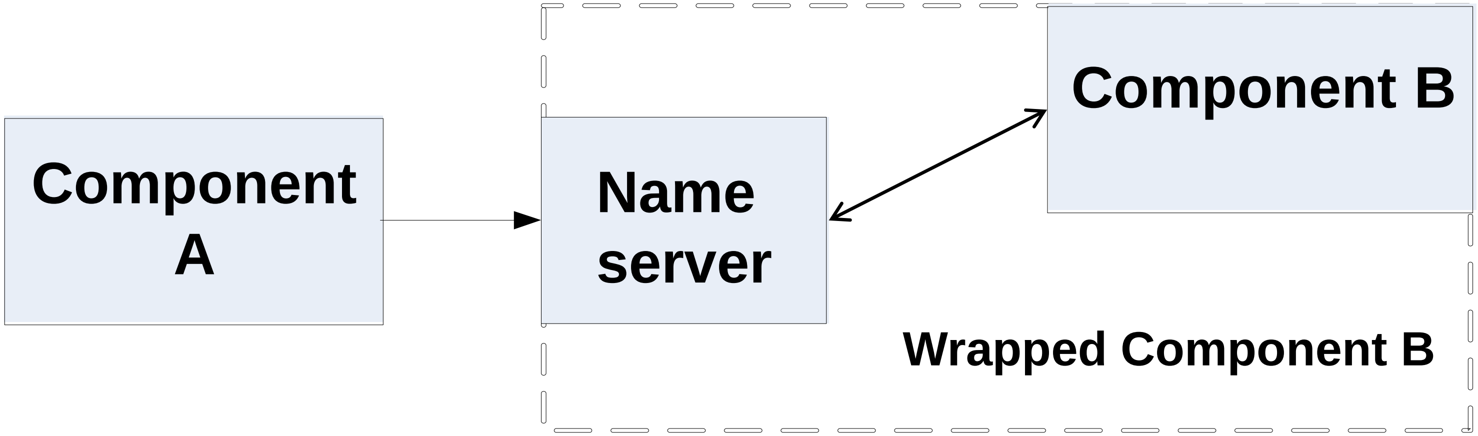
Facade



Source: http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/facade.html

Name server

- When the client (A) does not know about the location of the server (B), then the wrapper is called “name server”



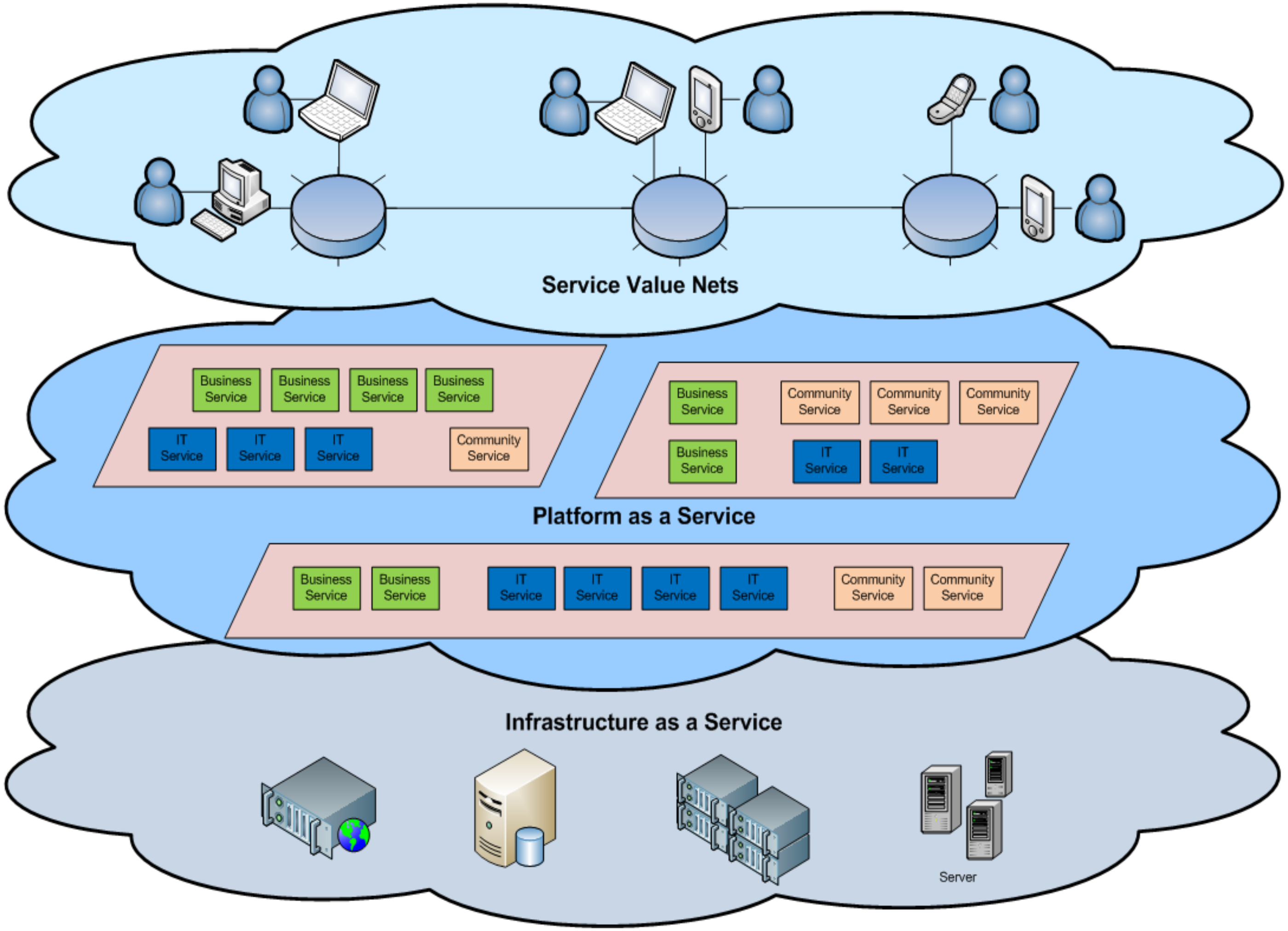
ARCHITECTURAL STYLES IN CLOUD

http://msdn.microsoft.com/en-us/library/dn568099.aspx

What is Cloud Computing?

- Cloud Computing is a general term used to describe a new class of network-based computing that takes place over the Internet,
  - A group of integrated and networked hardware, software and Internet infrastructure (called a platform).
  - Using the Internet for communication and transport provides hardware, software and networking services to clients
- Platforms hide the complexity and details of the underlying infrastructure from users and applications by providing very simple graphical interface or API (Applications Programming Interface).

Cloud Flavors?



SaaS
PaaS
IaaS

Architectural styles in cloud

- Circuit breaker
- Queue
- Caching
- Sharding

Circuit breaker

- Service fails in a distributed environment
- Classical solution is to implement a timeout for other services that call it
- However this may lead to needless resource consumption in cloud environment
  - Assume hundreds of users, waiting for a failed service and each one waiting for a timeout

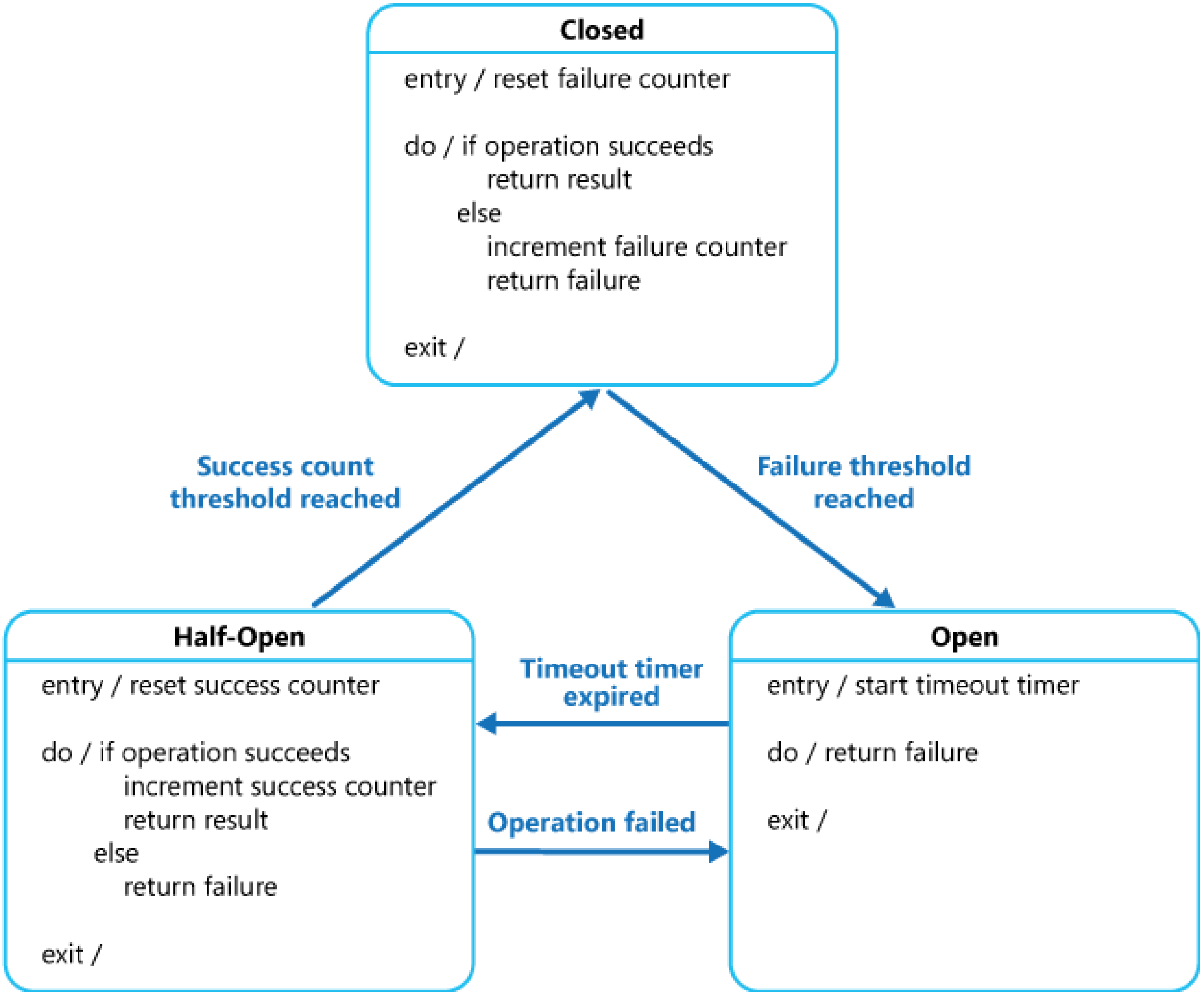
???

LIMBO SERVICES
WHAT GIVES, MAN? THIS LINE HASN'T MOVED IN HOURS!
CARTOONSTOCK .com
Search ID: jcon737
corl fu

Circuit breaker pattern

- It prevents an application from performing an operation that is unlikely to succeed
  - E.g. calling a service that is already known to be failed
- Acts as a proxy for operations that may fail
- Monitors the number of recent failures that have occurred, and then use this information to decide whether to allow the operation to proceed, or simply return an exception immediately

Circuit Breaker Pattern



Closed

```
entry / reset failure counter

do / if operation succeeds
         return result
      else
         increment failure counter
         return failure

exit /
```

Success count threshold reached

Failure threshold reached

Half-Open

```
entry / reset success counter

do / if operation succeeds
         increment success counter
         return result
      else
         return failure

exit /
```

Timeout timer expired

Operation failed

Open

```
entry / start timeout timer

do / return failure

exit /
```

Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

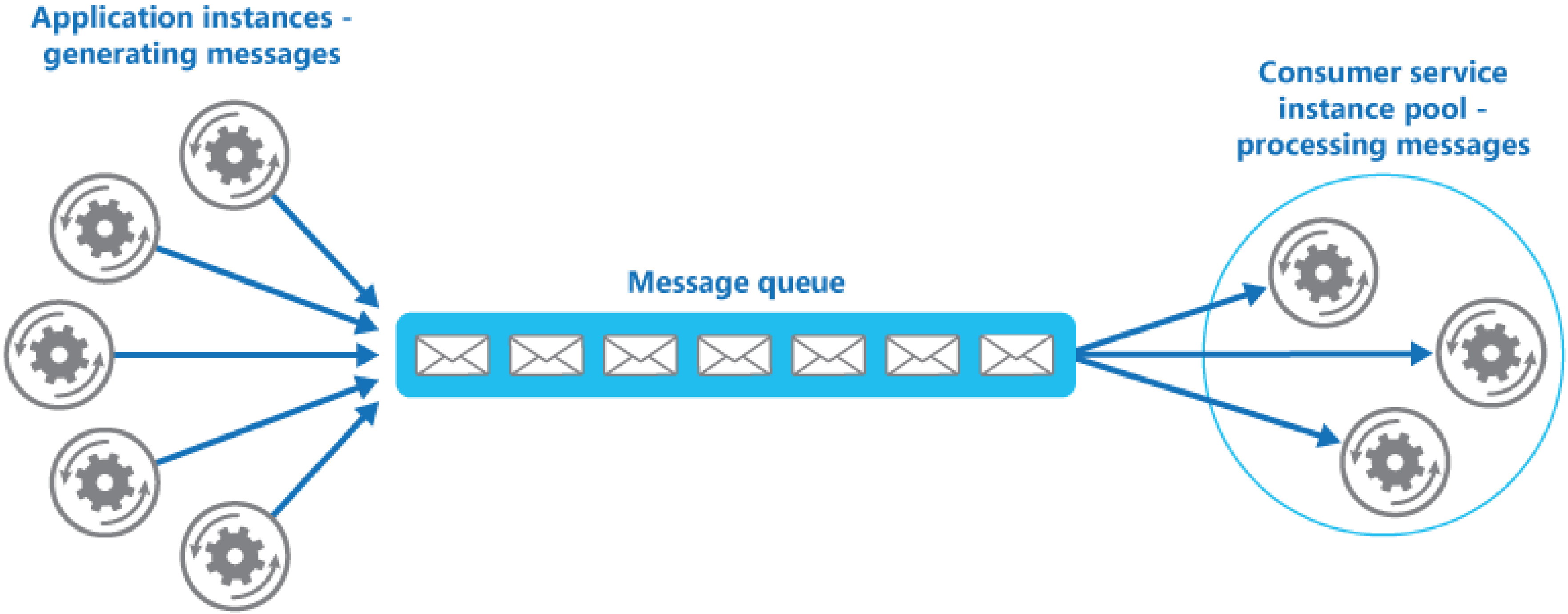
Circuit Breaker Pattern

- Helps to increase system dependability
  - A requesting application or large number of applications would not unnecessarily wait for timeout
    - When the service is known to be down
    - The network connection is temporarily down
    - The service is known to be very busy and not capable to respond

Queue

- Problem definition
  - In cloud you may have services that are flooded by a large number of concurrent requests by other services. In this case they may be overloaded or experience peak loads. You should find a solution to smooth heavy loads that may cause the service to fail or the calling task to time out.
- Different kinds of queues
  - Standard queue
  - Priority queue
  - Fixed length queue

Standard queue

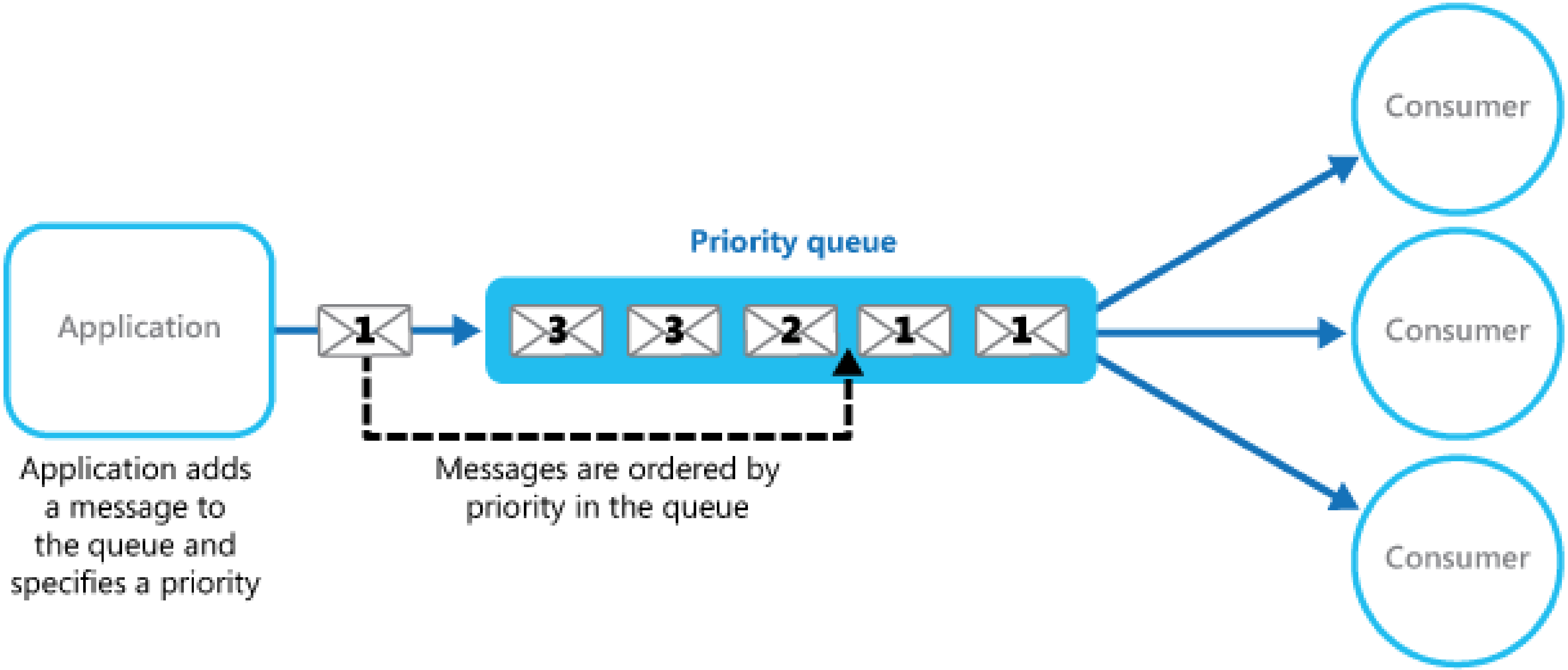

Application instances - generating messages
Message queue
Consumer service instance pool - processing messages

Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Standard queue

- The queue acts as a buffer, storing the messages until they are retrieved by the service.
- The service retrieves the messages from the queue and processes them
- Minimizes availability risks by a large number of concurrent requests.

Priority queue


Application
1
Priority queue
3
3
2
1
1
Consumer
Consumer
Consumer
Application adds a message to the queue and specifies a priority
Messages are ordered by priority in the queue

Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Priority queue

- Implements a policy to sort incoming request according to their priority
- Priority of request may be set by the sending applications or by the queue itself

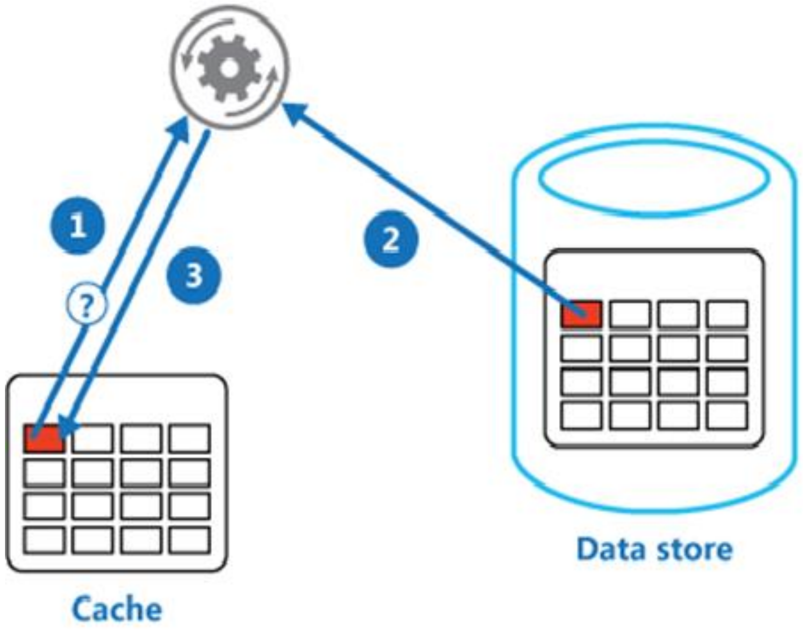
Fixed length queue

- We may design the queue in order to send an exception when a specific (extraordinary high for the service) amount of messages is reached
- In this case the requesting task will know that its request would not be processed and may take appropriate action, without waiting for the timeout

Cache

- Used to optimize repeated access to information held in a data store
- Will cached data be always completely consistent with the data in the data store?

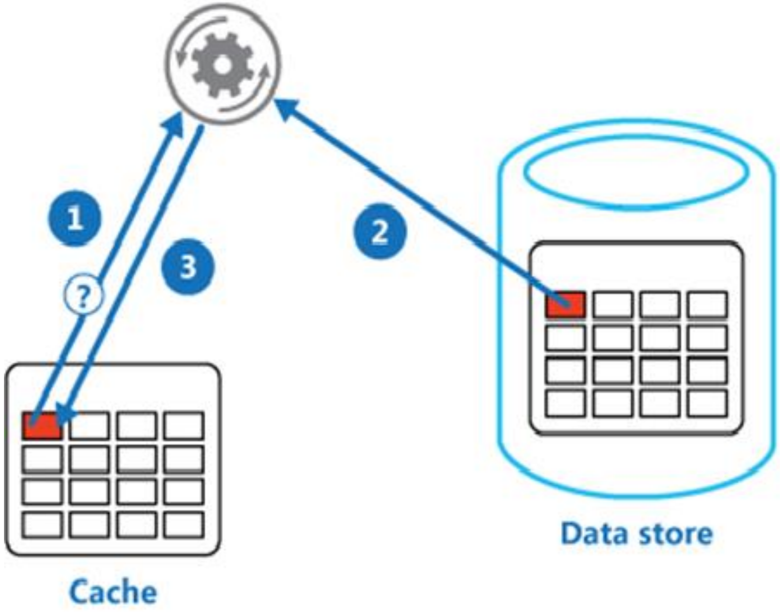
Cache



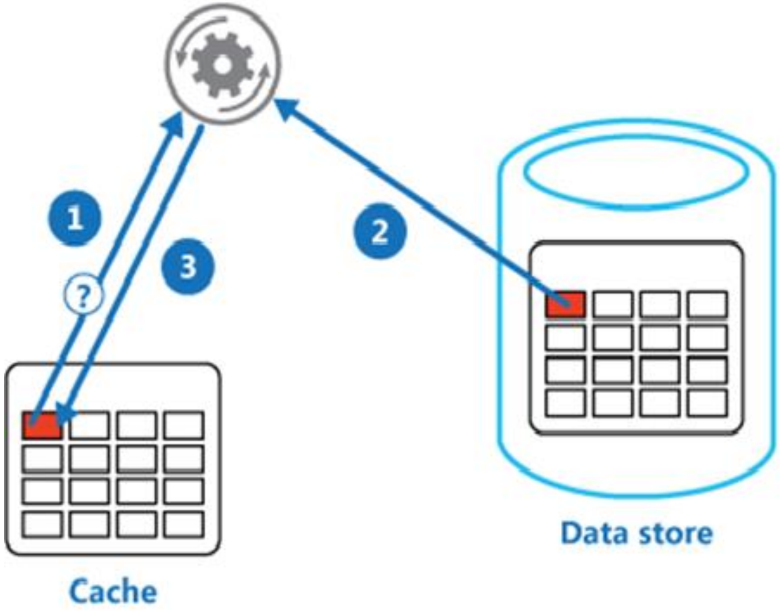
- Reading
  1. Look for item in the cache
  2. If it is not there, retrieve it from the cache
  3. Store a copy into the cache
- Writing
  1. Make the modification to the data store
  2. Invalidate the corresponding item in the cache.

Synchronization issue

Master



Slave



How is synchronization between two data stores managed

Sharding style

- The aim is to increase scalability when using big amounts of data
- The idea is to divide a data store into a set of horizontal partitions, called shards
- Motivation is to increase:
  - Storage space
  - Computing resources
  - Network bandwidth
  - Geographic disperse

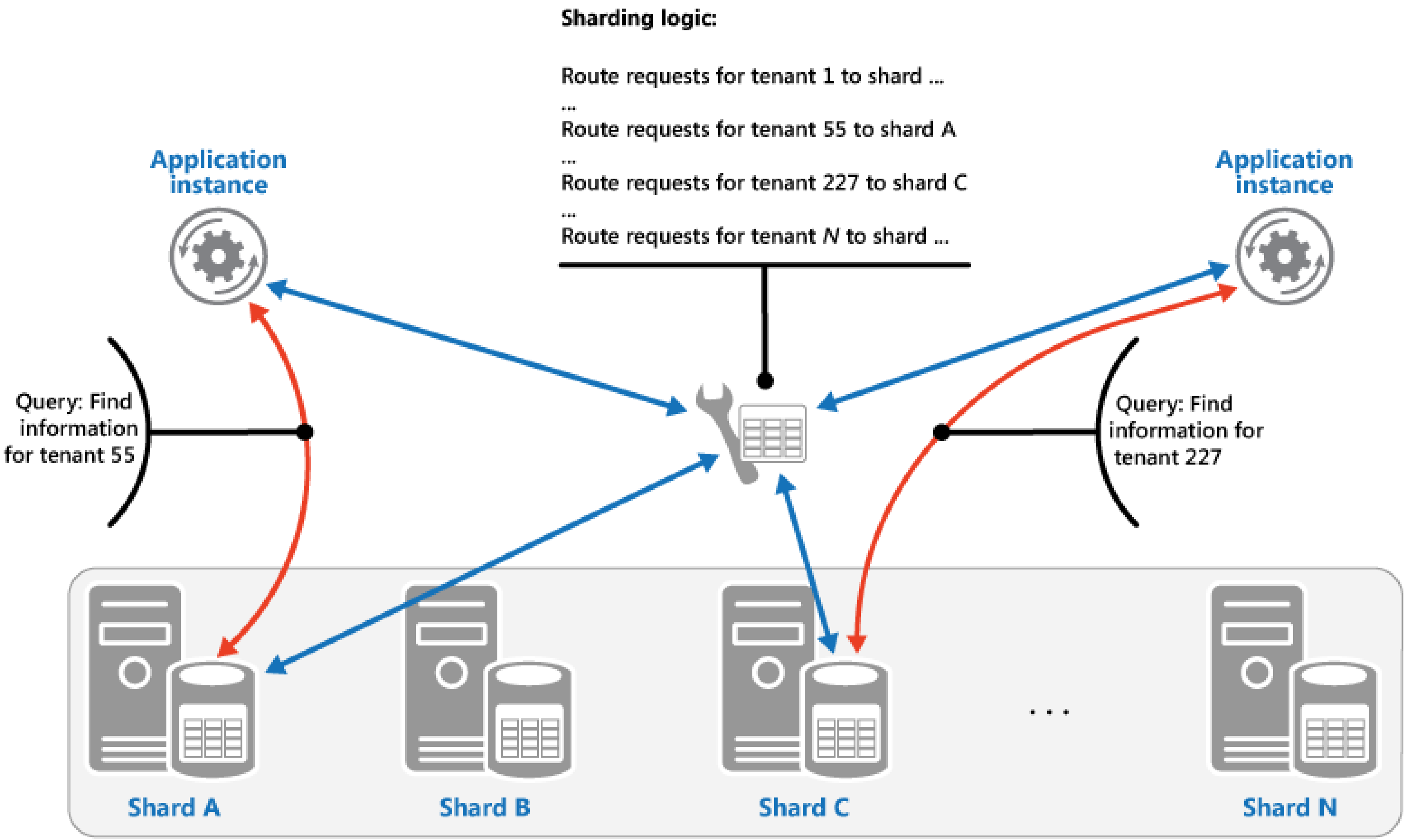
Sharding style

- Decision about how to divide data between shards is important
  - A shard typically contains items that fall within a specified range determined by one or more attributes of the data.
  - Data attributes form the *shard key* (aka *partition key*).
  - Shard key should be static. It should not be based on data that might change.
- Sharding physically organizes the data - when an application stores and retrieves data, the sharding logic directs the application to the appropriate shard
- Implementation issues about sharding logic
  - May be implemented as part of the data access code in the application
  - May be implemented by the data storage system if it transparently supports sharding.

Sharding style

- Three basic strategies exist for implementation of sharding logic
  - Lookup strategy
  - Range strategy
  - Hash strategy

Lookup sharding strategy



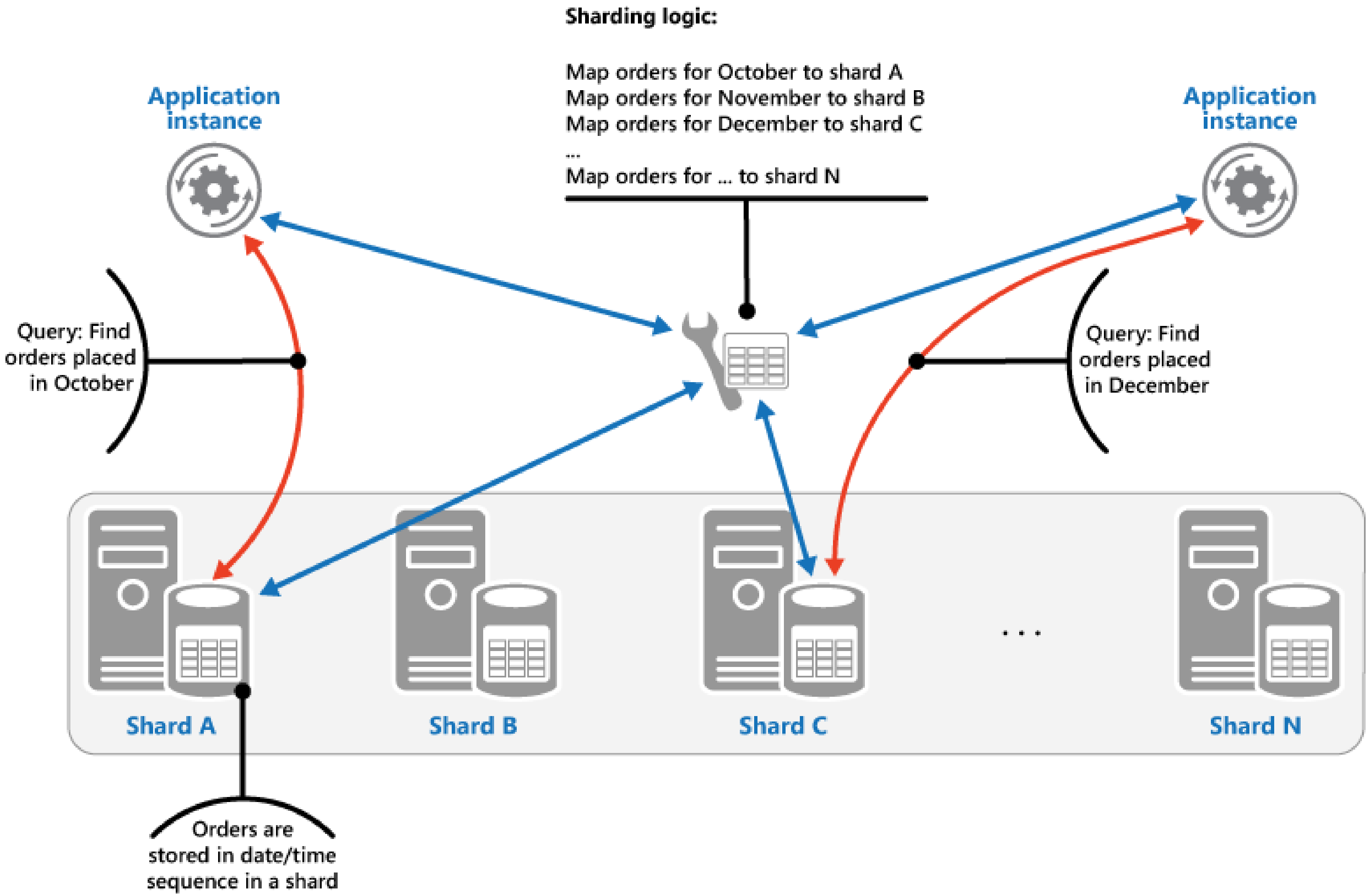
Sharding logic:

Route requests for tenant 1 to shard ...
...
Route requests for tenant 55 to shard A
...
Route requests for tenant 227 to shard C
...
Route requests for tenant *N* to shard ...

Application instance
Application instance
Query: Find information for tenant 55
Query: Find information for tenant 227
Shard A
Shard B
Shard C
. . .
Shard N

Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Range sharding strategy



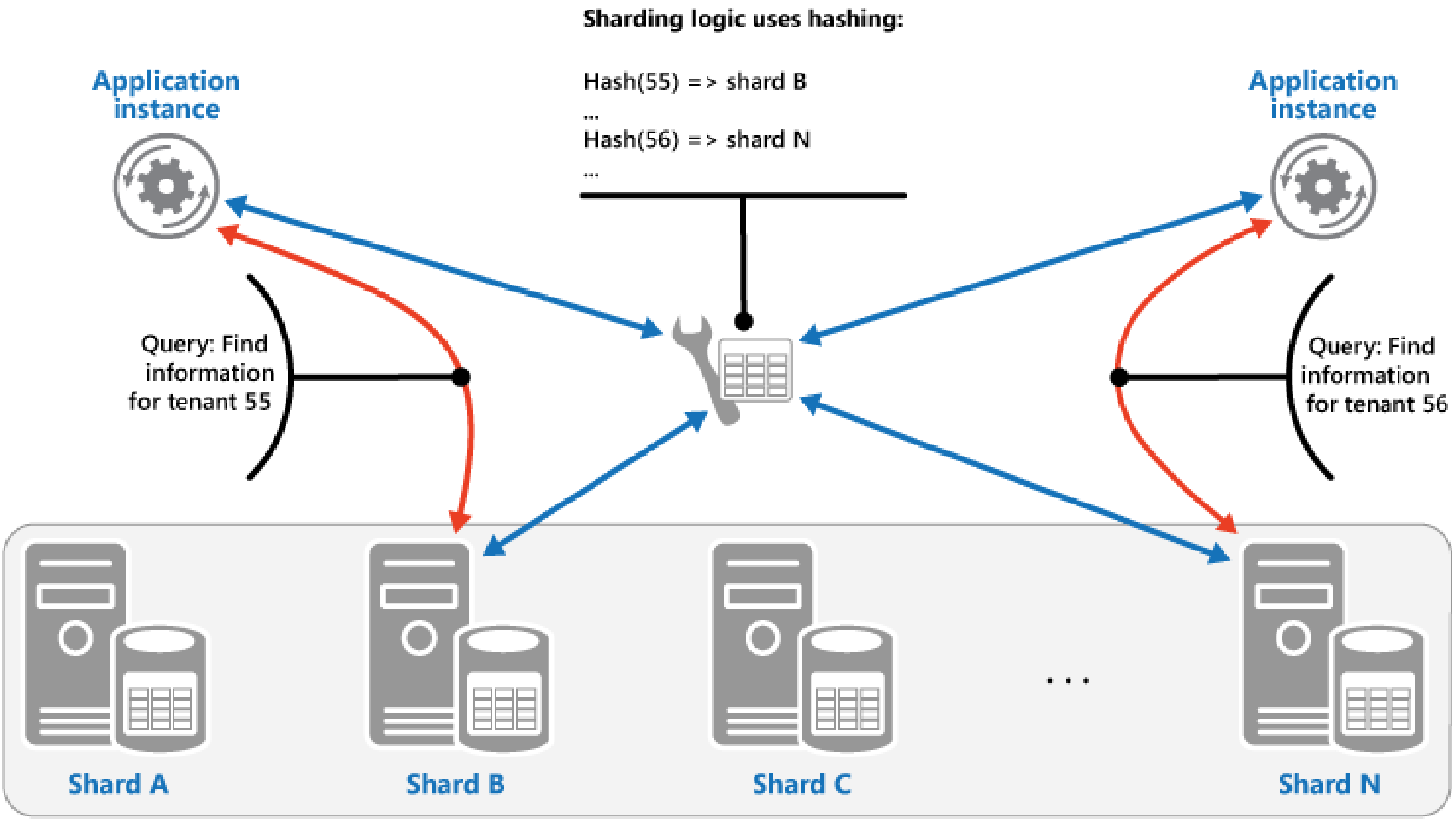
Sharding logic:

Map orders for October to shard A
Map orders for November to shard B
Map orders for December to shard C
...
Map orders for ... to shard N

Application instance
Application instance
Query: Find orders placed in October
Query: Find orders placed in December
Shard A
Shard B
Shard C
. . .
Shard N
Orders are stored in date/time sequence in a shard

Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Hash sharding strategy



Source: Cloud design patterns: http://msdn.microsoft.com/en-us/library/dn568099.aspx

Sharding advantages

- Better data management - abstraction of data physical location provides
  - Control over which shards contain which data
- Increased performance of the data storage

Sharding disadvantages

- Application performance issues
  - Overhead when determining the location of each data
  - More performance overhead when data that matches a single request is distribute among many shards
- Shards may contain misbalanced amount of data
- Sometimes it is extremly difficult to design a shard key that matches the requirements of every possible query against the data.