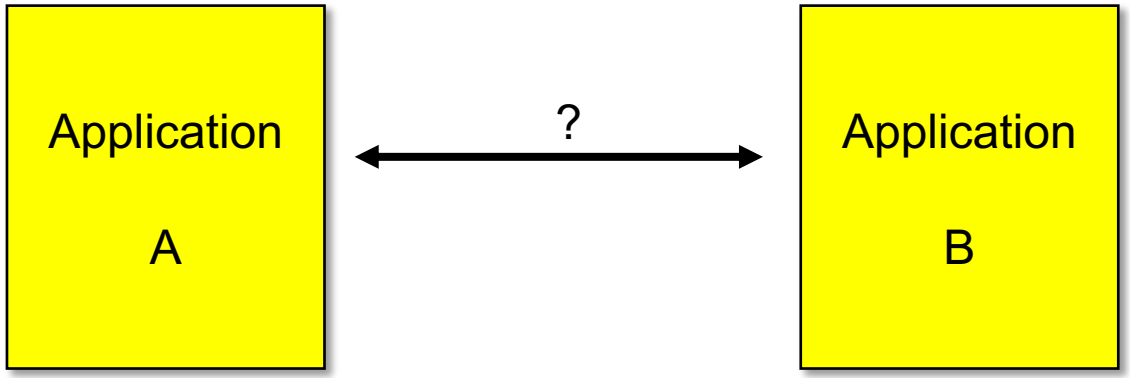
Service Integration Patterns

Enterprise Architectures for Big Data

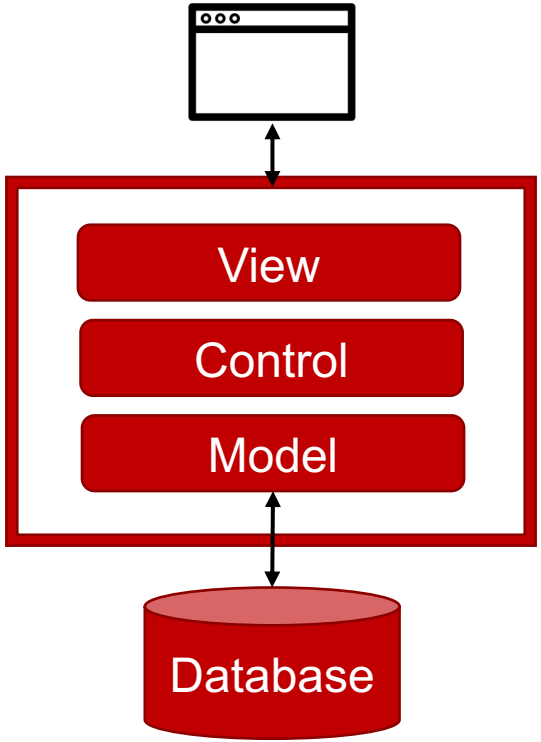
Problem

- How can I **integrate** multiple **distributed** applications or services so that they **work together** and can **exchange information**?

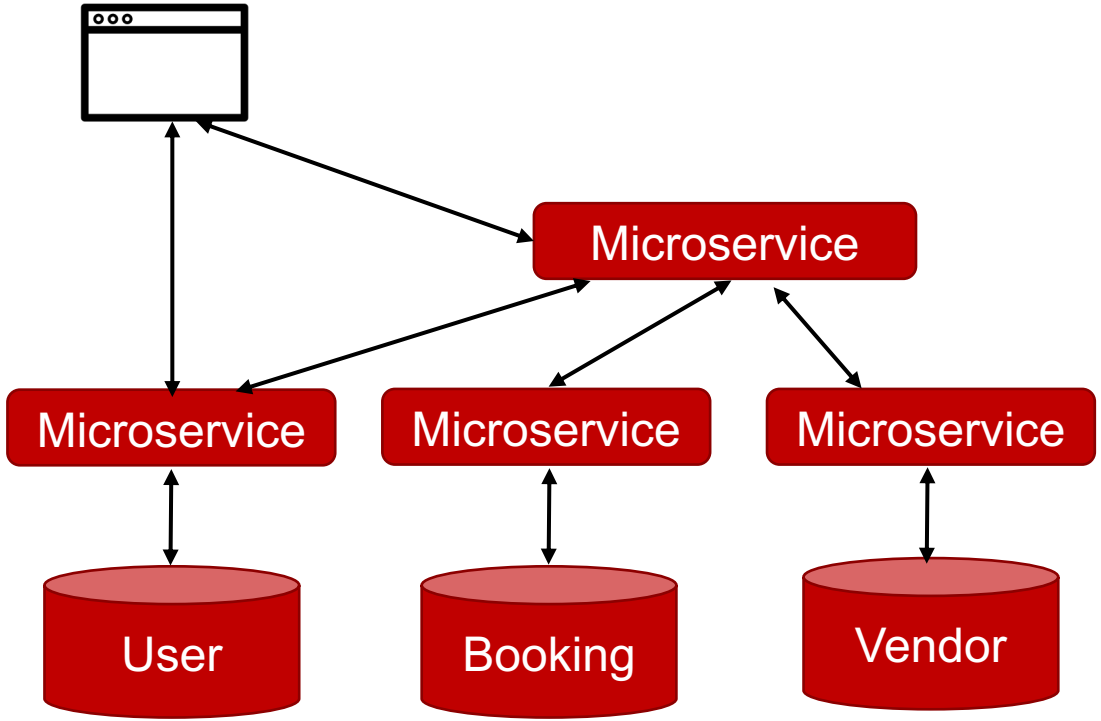


Monolithic vs. Microservice Architecture

Monolithic Architecture

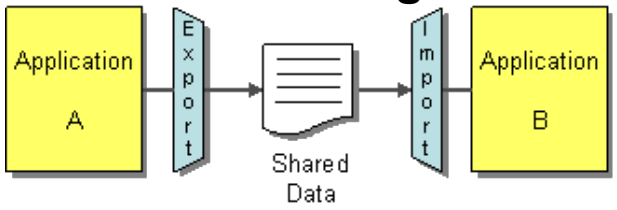


Microservice Architecture

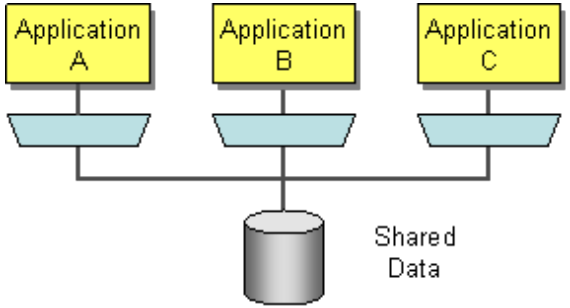


Integration Styles

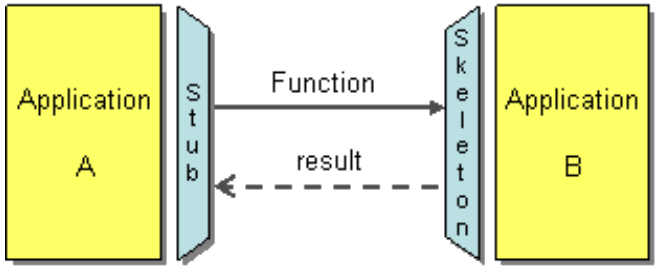
File Transfer Integration



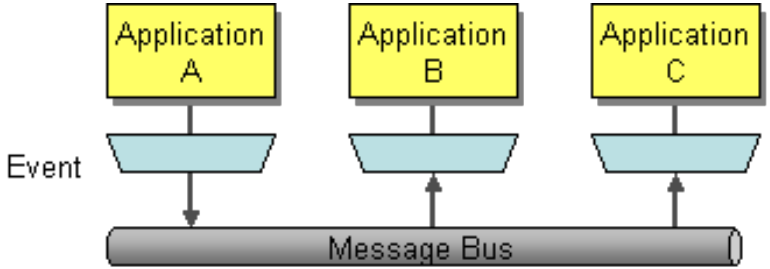
Shared Database

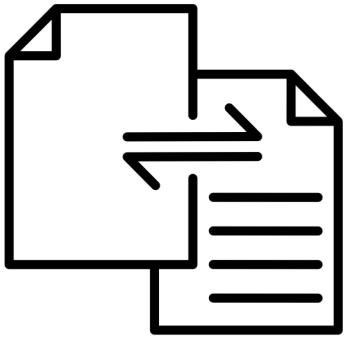


Remote Procedure Invocation



Message Queue

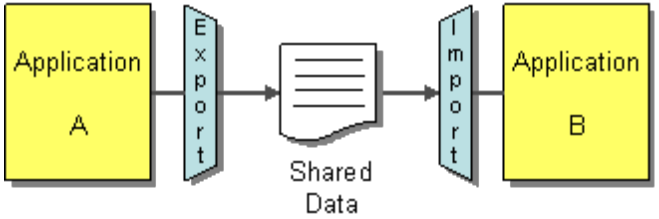




File Transfer Integration

File Transfer Integration

- **Problem**: How can I integrate multiple applications or services so that they work together and can exchange information?



- **Solution**: Have each application produce **files** containing information that other applications need to consume.
- Integrators take the responsibility of transforming files into different formats.
- Produce the files at regular intervals according to the nature of the business.

File Transfer Integration

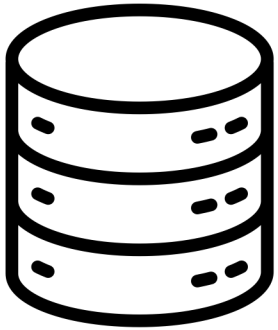
Pro

- Simplicity
- All programming languages have ways to read and write files
- Decoupling of applications
- Allow asynchronous communication
- Scalable for large data
- Scalable through Network File System (NFS), Hadoop (HDFS) or Cloud (AWS S3)
- Good for Batch processing

Con

- Needs common access to files/folders or cloud file storage
- Different file formats
- Agreement of filenames, folders, and formats
- No notification of other application
- Lacks timeliness (high latency)
- No concurrent write access

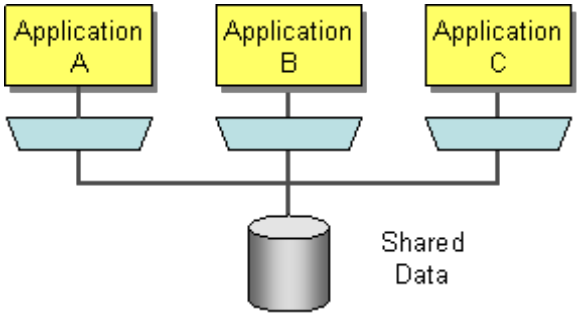
Source: Hohpe & Woolf (2004)



Shared Database

Shared Database

- **Problem**: How can I integrate multiple applications or services so that they work together and can exchange information?



- **Solution**: Integrate applications by having them store their data in a single **Shared Database**.

Shared Database

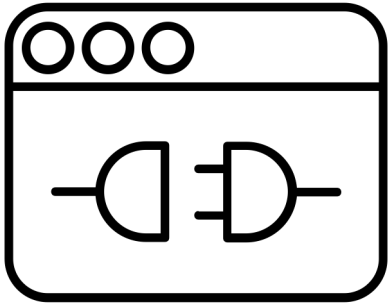
Pro

- Timely data
- Consistency
- Concurrent Read-Write Access (ACID Transactions)
- No file formats
- Integrated through
- Often powerful query language (like SQL)

Con

- Needs a unified schema
  - High Coupling
  - Applications might have their own schema
- Database might be a bottleneck
  - Especially for globally distributed systems
- No sharing of functionality (just data)
- No notification of other application
- Needs access to database (firewall, views, rights, …)
- Might allow complex SQL queries that will slow down the source system
- Tied to specific database technology

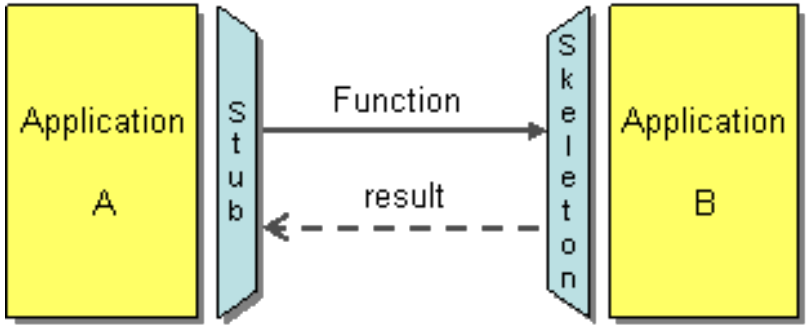
Source: Hohpe & Woolf (2004)



Remote Procedure Invocation
Web API

Remote Procedure Invocation
Web API

- **Problem**: How can I integrate multiple applications so that they work together and can exchange information?



- **Solution**: Develop each application as a large-scale object or component with encapsulated data.
- Provide an **interface** to allow other applications to interact with the running application.

Source: Hohpe & Woolf (2004)

REpresentational State Transfer (REST)

- Architectural style inspired by the Web
- **Resource**: things that the service itself knows about, like Customers
- Each Resource has a **Uniform Resource Locator (URL)**
- Different **Representations** of Resources
- How Resources are represented externally is completely decoupled from how they are stored internally
  - E.g. Internally relational DB, externally JSON or HTML
- Web (HTTP) for transport
- JSON (or XML) for representing/encoding

Hypertext Transfer Protocol (HTTP) Verbs

- Most common HTTP Methods:
  - GET Method Read (also used by Web Browsers)
  - POST Method Create (also used by Web Browsers)

- Only for APIs:
  - PUT Method Update
  - DELETE Method Delete

- CRUD = Create-Read-Update-Delete
- APIs can just use GET and POST HTTP Methods!

Hypertext Transfer Protocol (HTTP) Verbs

GET Method

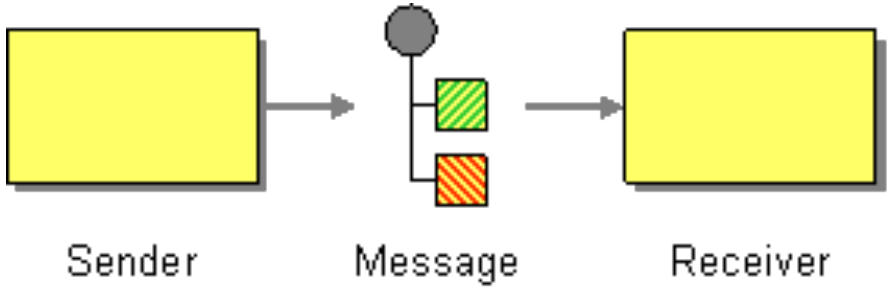
- Request data from a specified resource
- Query string (name/value pairs) in the URL of a GET request (in the address bar of the browser)
- Values: Only ASCII characters
- Length restrictions (maximum URL length is 2048 characters)
- Can be cached
- Can be bookmarked
- Reload is harmless (Idempotent)

POST Method

- Send data to create (update / delete) a resource
- When you submit a form in the browser
- Values: No restriction
- No length restrictions
- Never cached
- Cannot be bookmarked
- Reload: Data will be resubmitted

Messages

- **Subproblem**: How can two applications be connected by sending and receiving pieces of information?



- **Solution**: Package the information into a **Message**, a data record that the sender can transmit.

Example of JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
}
```

Example of XML

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
</person>
```

Remote Procedure Invocation - Web API

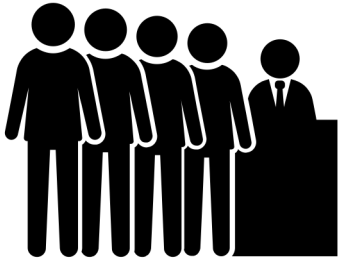
Pro

- Easy to use
- Ubiquitous
- Nearly all languages have a way to use Web APIs
- Sharing of functionality
- Encapsulation of data
- Timely response (However, compared to normal procedures: RPCs have network latency)

Con

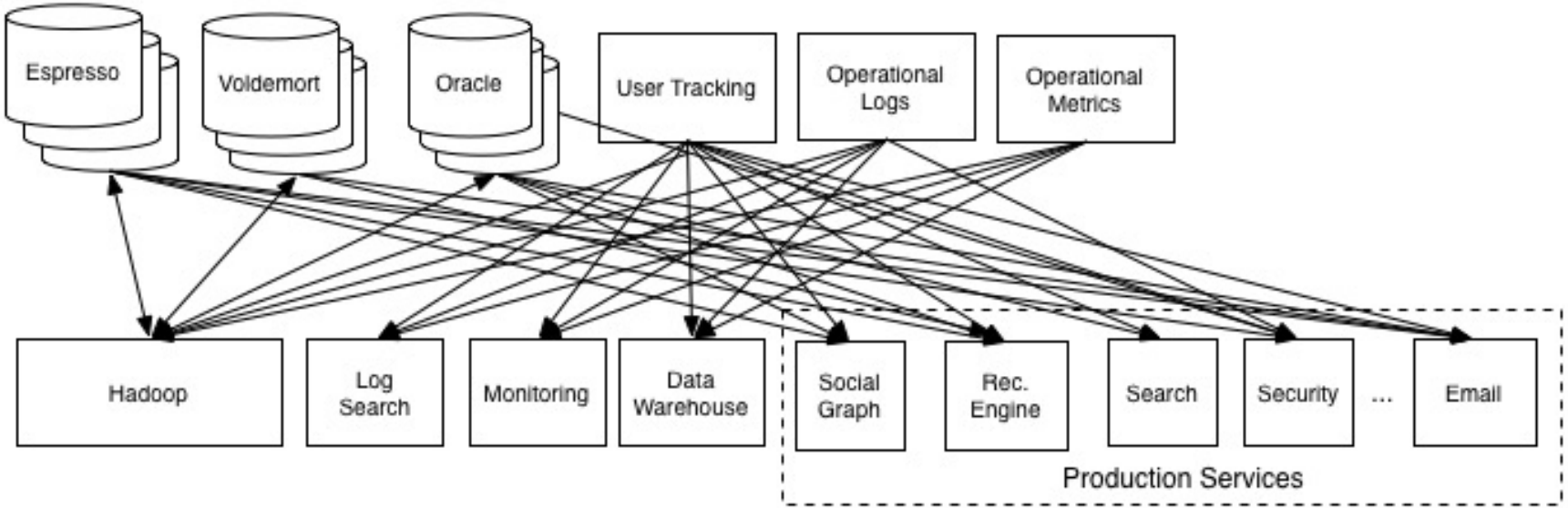
- Only request-response
- Only one-to-one connection
- Risk of tight coupling
- If response takes a long time, requesting application is blocked
- No asynchronous communication
- Receiver needs to understand message format
- Risk of changing message format

Source: Hohpe & Woolf (2004)

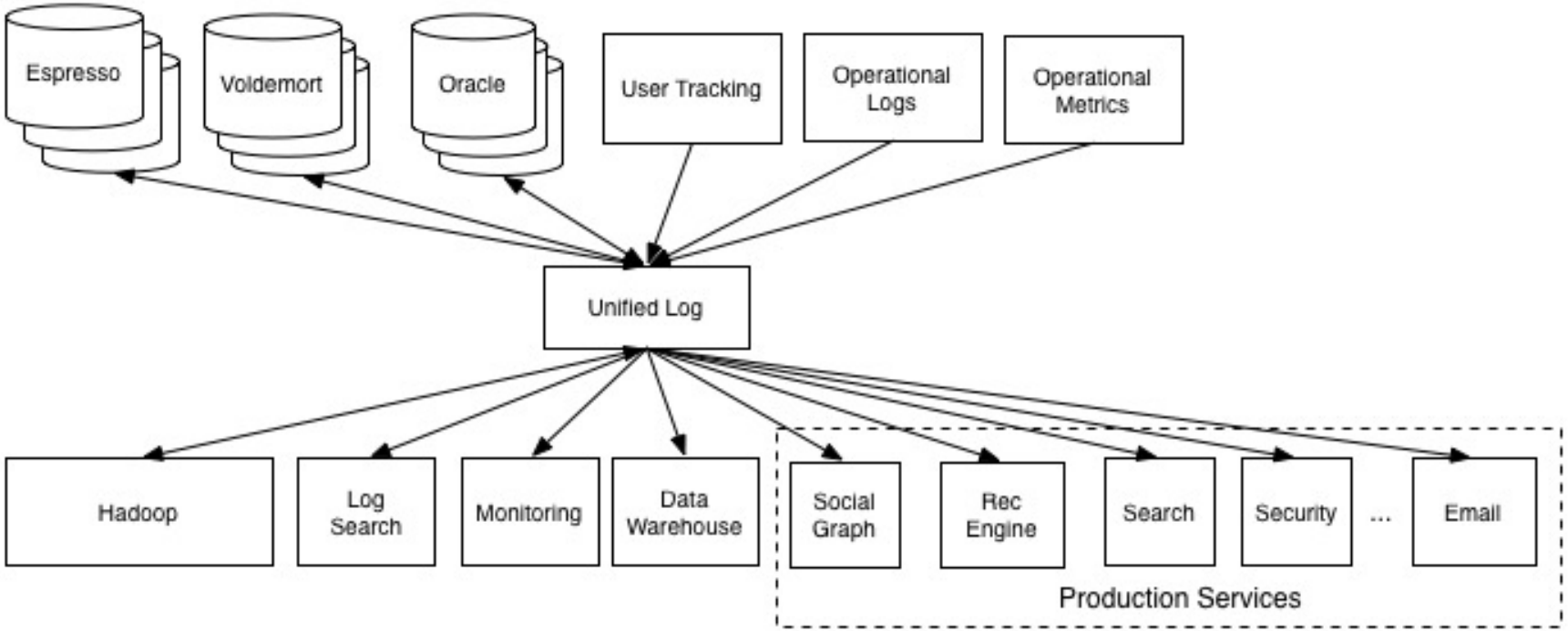


Message Queue

Tightly Coupled Systems

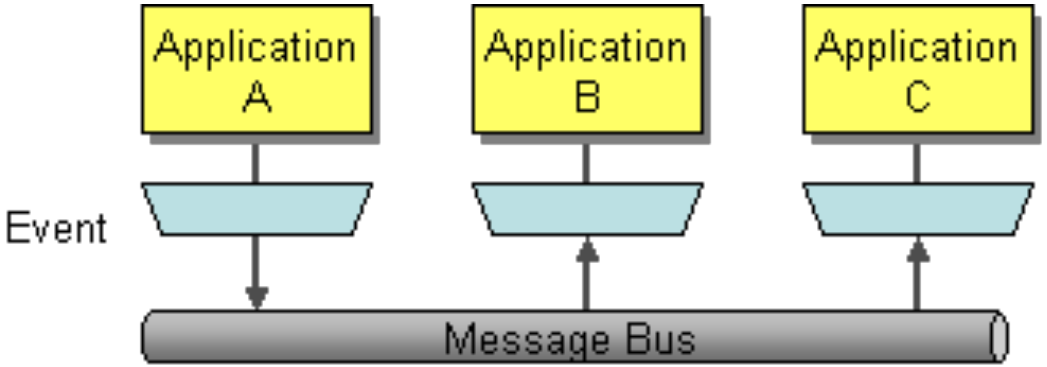


Decoupled Systems through a Message Queue



Messaging

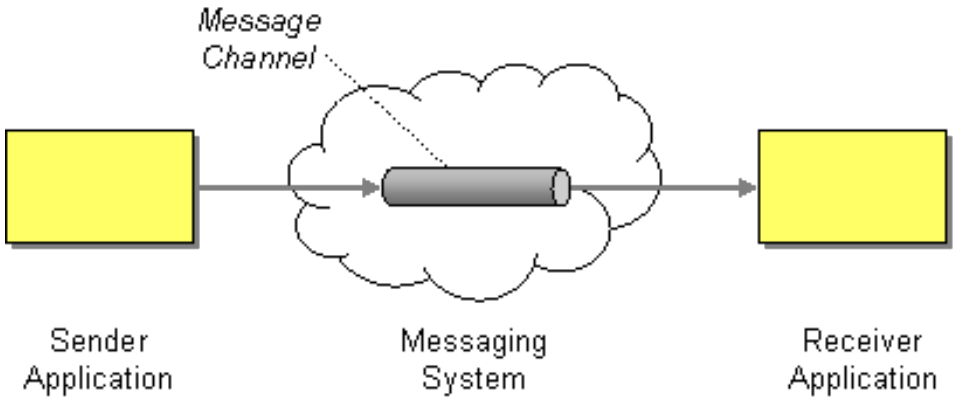
- **Problem**: How can I integrate multiple applications so that they work together and can exchange information?



- **Solution**: Use **Messaging** to transfer packets of data frequently, immediately, reliably, and **asynchronously**, using customizable formats.

Message Queue / Channel

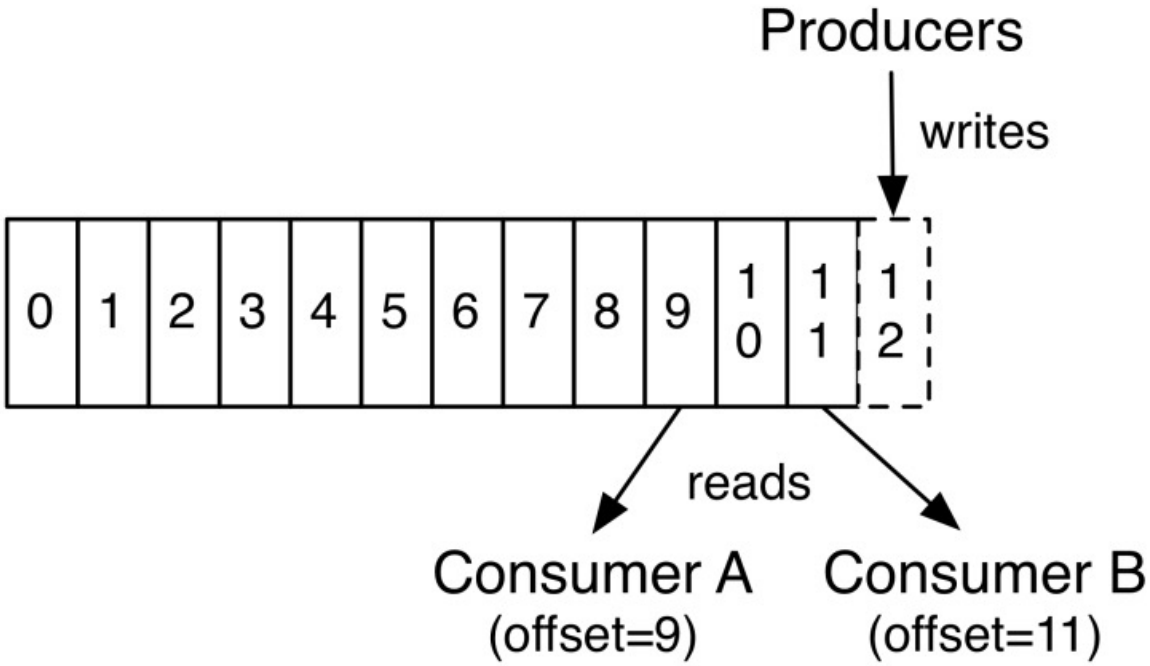
- **Subproblem**: How does one application communicate with another using messaging?



- **Solution**: Connect the applications using a **Message Queue**, where one application writes information to the channel and the other one reads that information from the channel.

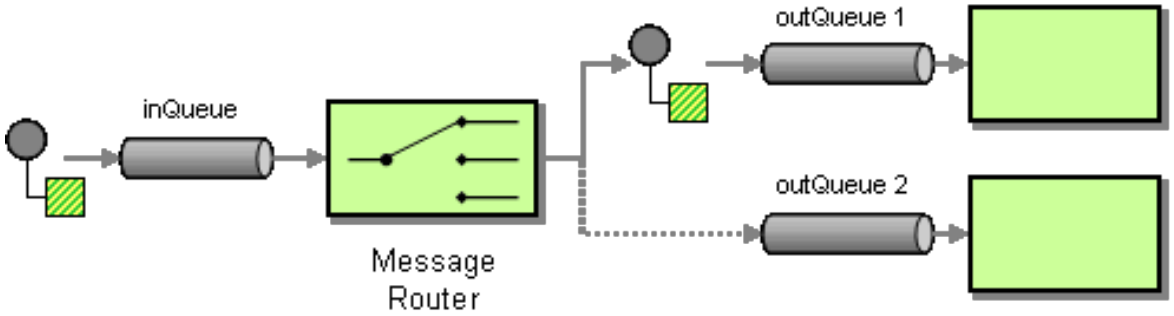
Source: Hohpe & Woolf (2004)

Log-structured Message Queue



Message Router

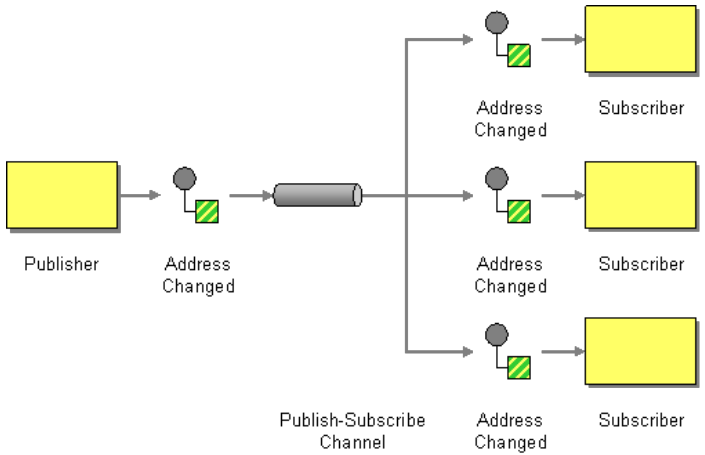
- **Subproblem**: How can you decouple individual processing steps so that messages can be passed to different filters depending on a set of conditions?



- **Solution**: Insert a special filter, a **Message Router**, which consumes a Message from one Message Channel and republishes it to a different Message Channel channel depending on a set of conditions.

Publish-Subscribe Channel

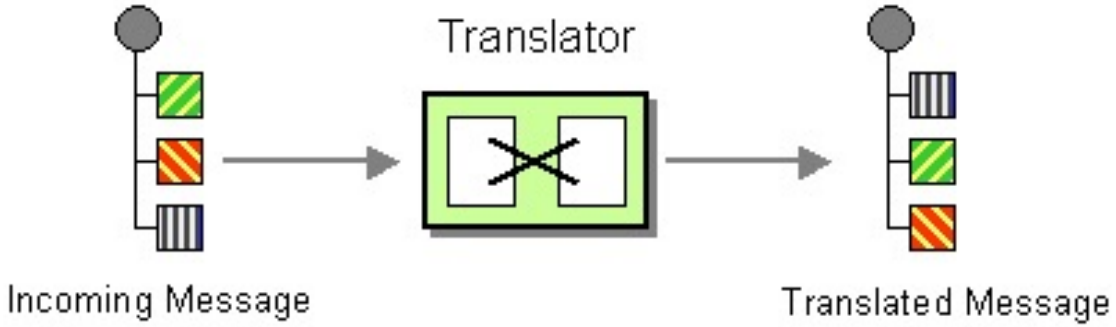
- **Subproblem**: How can the sender broadcast an event to all interested receivers?



- **Solution**: Send the event on a **Publish-Subscribe Channel**, which delivers a copy of a particular event to each receiver.

Message Translator

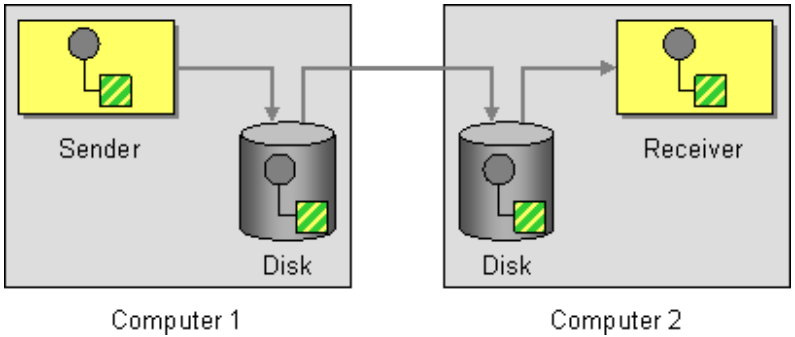
- **Subproblem**: How can systems using different data formats communicate with each other using messaging?



- **Solution**: Use a special filter, a **Message Translator**, between other filters or applications to translate one data format into another.

Guaranteed Delivery

- **Subproblem**: How can the sender make sure that a message will be delivered, even if the messaging system fails?



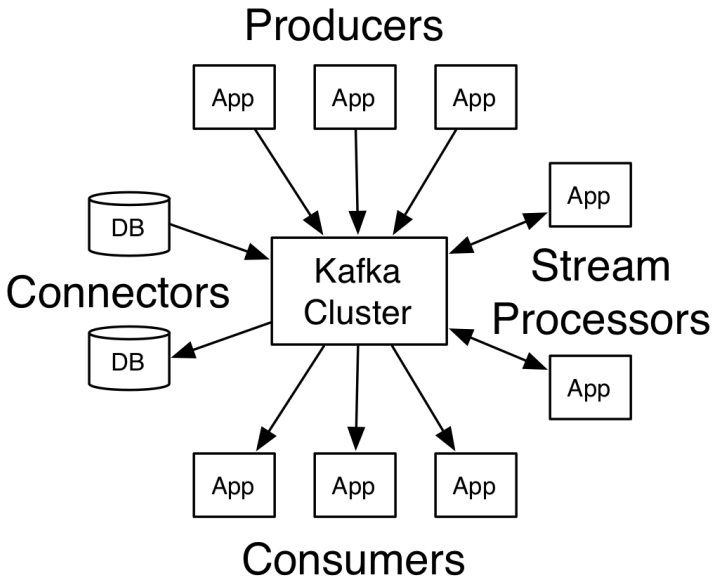
- **Solution**: Use **Guaranteed Delivery** to make messages persistent so that they are not lost even if the messaging system crashes.

Message Queues Systems

- Apache Kafka
- AWS Simple Queue Service (SQS)
- RabbitMQ
- TIBCO Enterprise Message Service
- IBM MQ
- Anypoint MQ (Mulesoft)
- Apache ActiveMQ
- Redis
- ...

Apache Kafka

- Massively scalable publish-subscribe message queue



APACHE kafka™
A distributed streaming platform

Message Queue

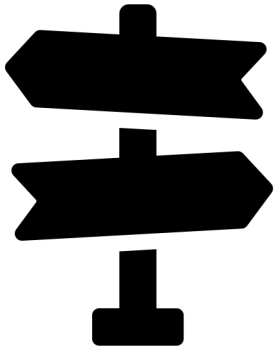
Pro

- Decoupling of services
- Allows asynchronous communication
- Broadcasting capability (1-to-n)
- Scalable
- Relatively timely
- Guaranteed only-once delivery
- Translation of different message standards
- Allows stream processing

Con

- Complicated
- High overhead
- Hard to debug and test
- Difficult integration beyond firewalls
- Might have still some data latency
- Not ideal for big batch jobs with few large files as results
- Different standards and products

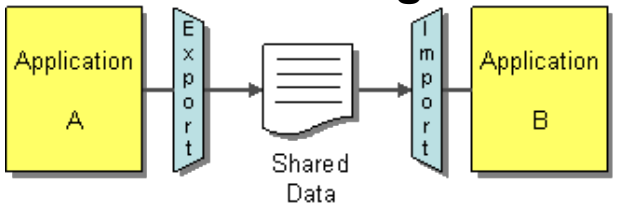
Source: Hohpe & Woolf (2004)



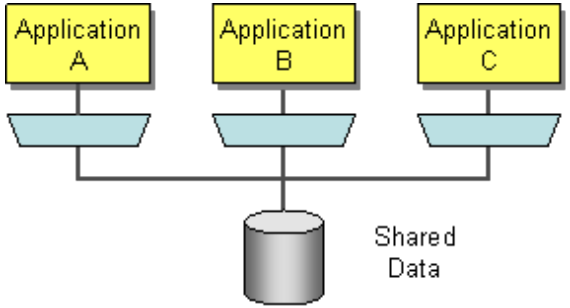
Decision Criteria for Integration Styles

Integration Styles

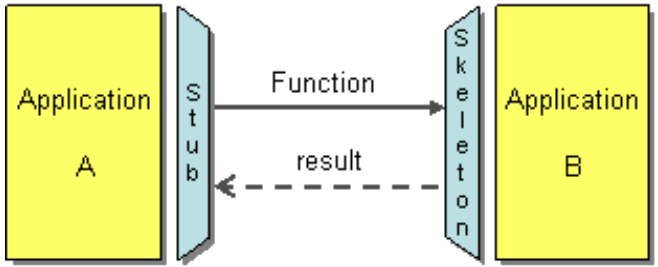
File Transfer Integration



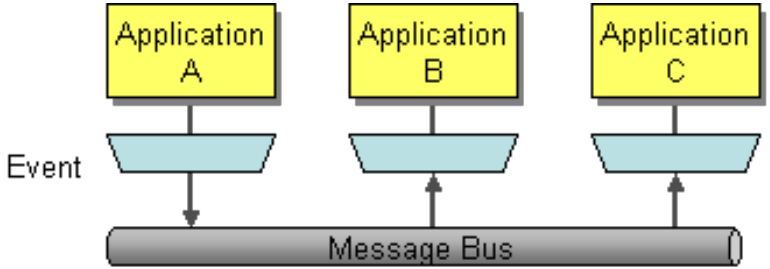
Shared Database



Remote Procedure Invocation



Message Queue



Decision Criteria for Integration Styles

1. Application coupling
2. Integration simplicity
3. Integration technology
4. Data format
5. Data timeliness
6. Data or functionality
7. Asynchronicity

Source: Hohpe & Woolf (2004)

Decision Criteria for Integration Styles

1. Application coupling
   - Even integrated applications should **minimize their dependencies** on each other so that each can evolve without causing problems for the others.
2. Integration simplicity
   - Developers should strive to **minimize changing the application** and minimize the amount of **integration code** needed.
3. Integration technology
   - Different integration techniques require varying amounts of **specialized software and hardware**.
   - These special tools can be expensive, can lead to vendor lock-in, and increase the burden on developers to understand how to use the tools to integrate applications.

Source: Hohpe & Woolf (2004)

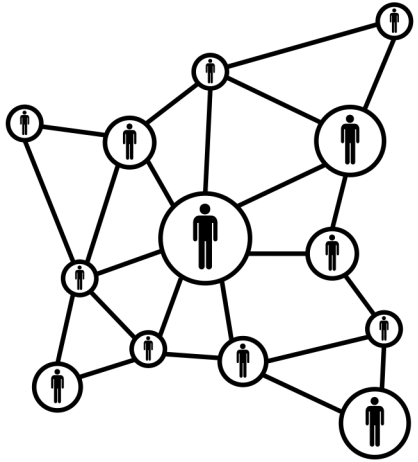
Decision Criteria for Integration Styles

4. Data format
   - Integrated applications must agree on the format of the data they exchange, or must have an intermediate translator to unify applications that insist on different data formats.
   - A related issue is **data format evolution and extensibility**—how the format can change over time and how that will affect the applications.
5. Data timeliness
   - Integration should **minimize the latency** between when one application decides to share some data and other applications have that data.
6. Functionality Sharing
   - Integrated applications may not want to simply share data, they may wish to **share functionality** such that each application can invoke the functionality in the others

Source: Hohpe & Woolf (2004)

Decision Criteria for Integration Styles

7. Asynchronicity
   - Procedure may not want to **wait for the subprocedure to execute**
   - It may want to invoke the subprocedure asynchronously, starting the subprocedure but then letting it **execute in the background**.



Conway's law

Conway's law

“Any organization that designs a system […] will produce a design whose structure is a copy of the **organization's communication structure**.”

Melvin E. Conway