The Integration Game

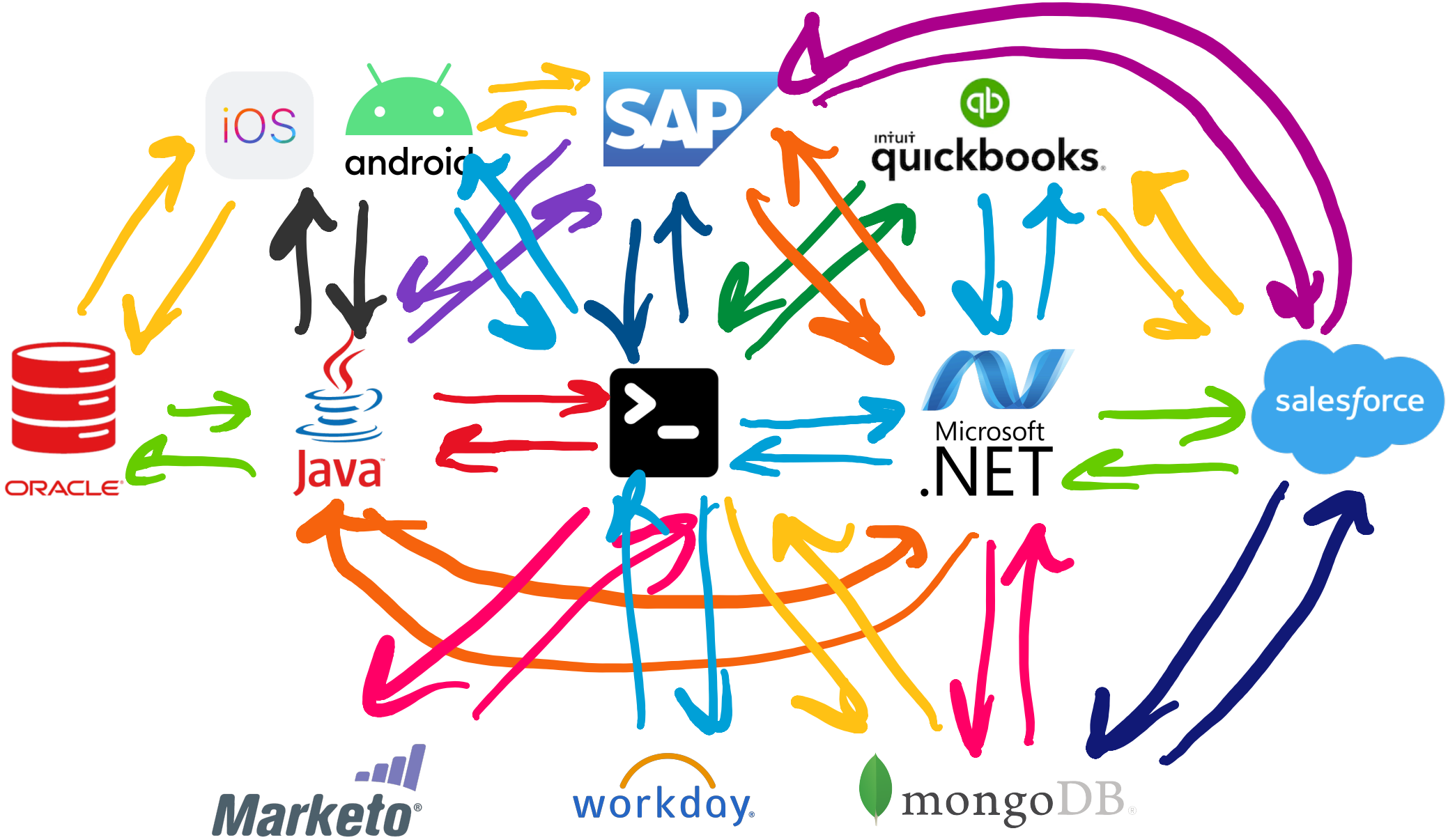
How to Pick a Winning Integration Pattern

Branden Bellanca

Salesforce Architect

@bsbellanca

/branden-bellanca


iOS
android
SAP
intuit quickbooks
ORACLE
Java
Microsoft .NET
salesforce
Marketo
workday
mongoDB


SAP
salesforce
workday

"We'll fix it in Phase 2"

Makeshift Integrations

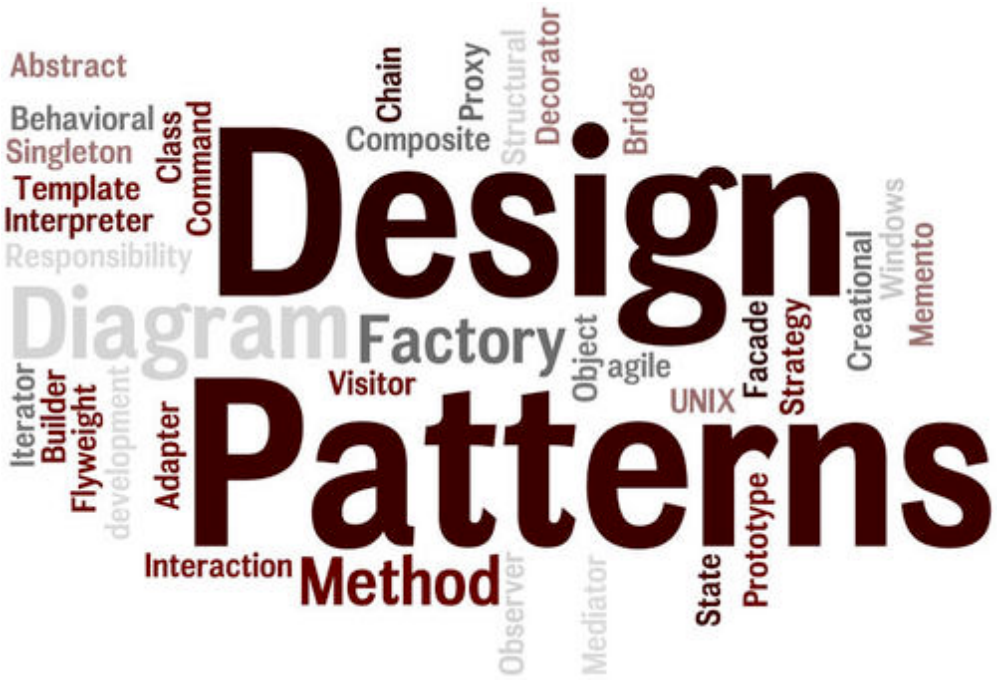
The "Great Value" Guide

@bsbellanca

INTEGRATION GAME

Design Patterns

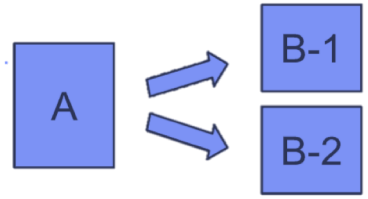
- Represent some of the best practices adapted by experienced developers
- Systematically names, motivates, and explains a general design that addresses a recurring design problem
- Describes the problem, the solution, when to apply the solution, and its consequences



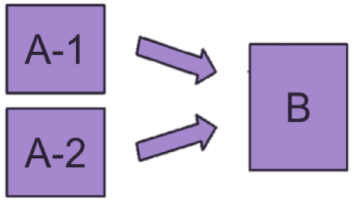
Integration Patterns

Migration

Broadcast



Aggregation

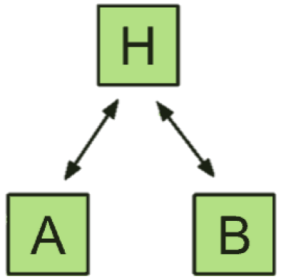


Bi-directional Sync

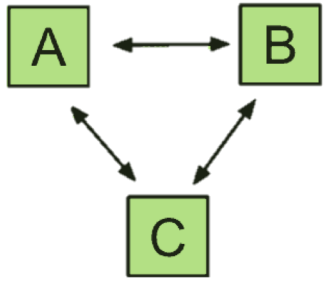
Correlation

INTEGRATION ARCHITECTURE

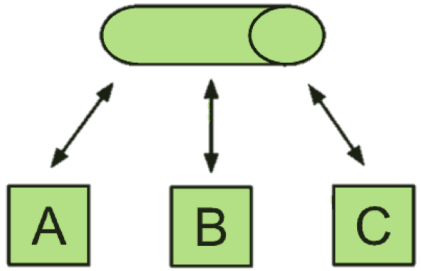
Point-to-Point



Hub and Spoke



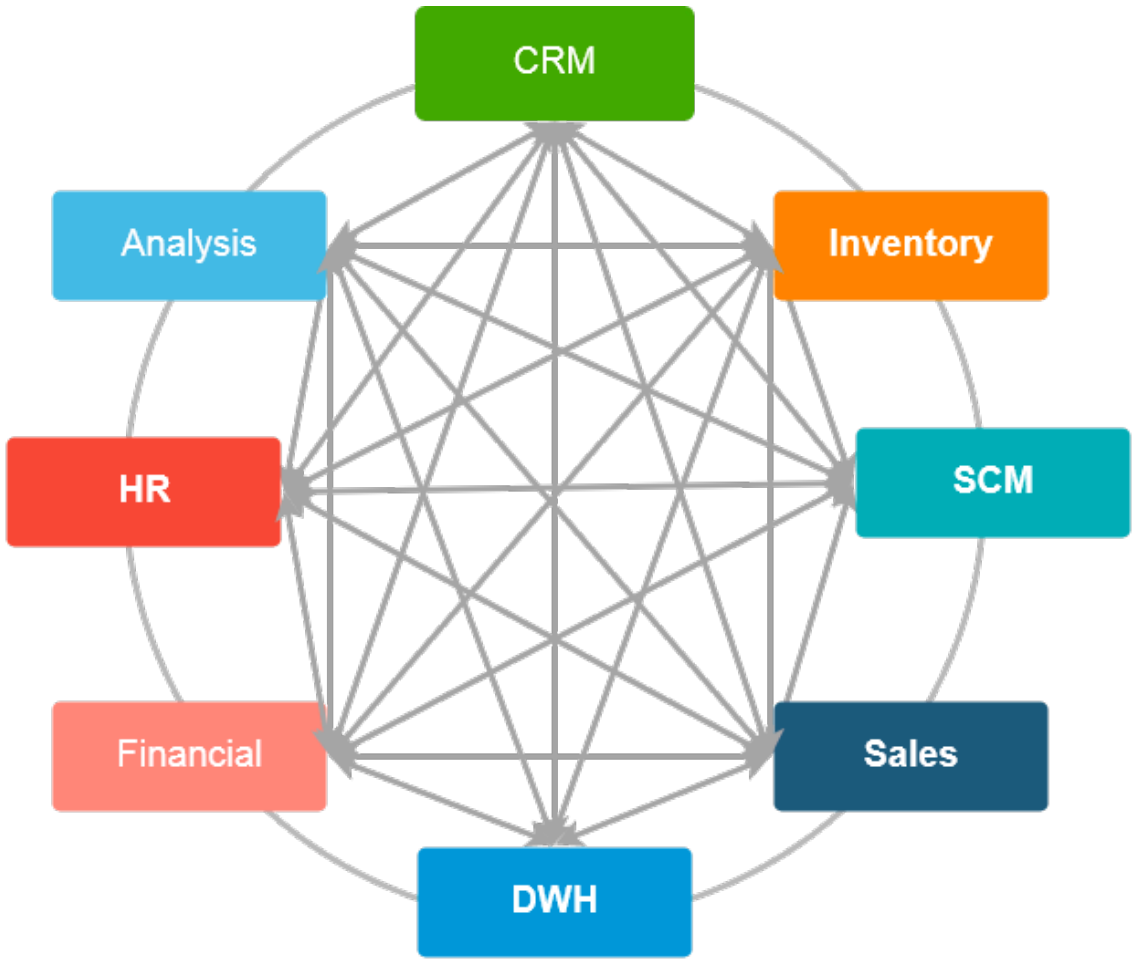
ESB



INTEGRATION ARCHITECTURE

Point-to-point

- Data flows directly from system to system
- Easy to implement
- Very hard to scale



Point-to-point Integration

INTEGRATION ARCHITECTURE

Hub and Spoke

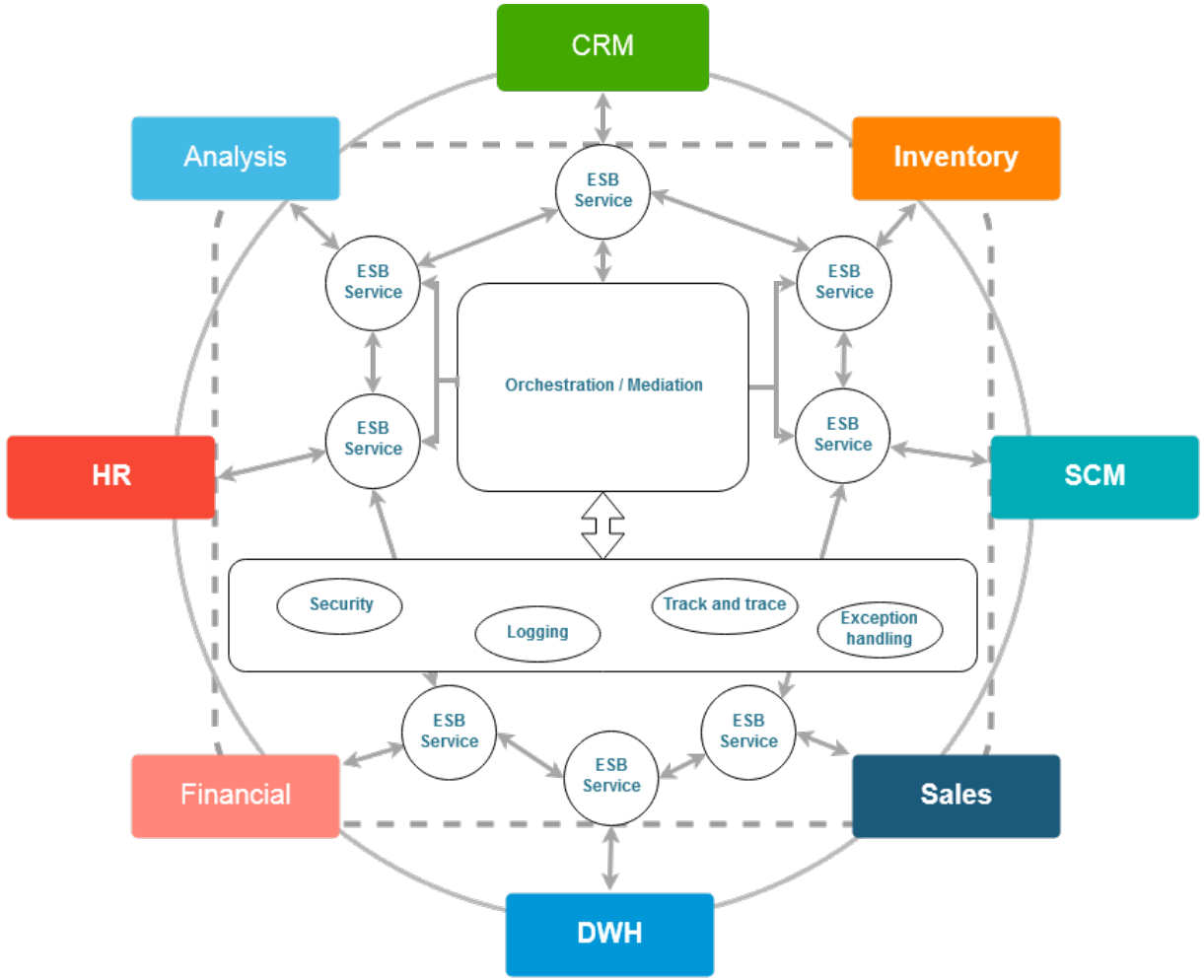
- Data flows through a central point
- Single point of failure
- Adds another place where development must take place and another runtime component

Hub-and-spoke Integration

INTEGRATION ARCHITECTURE

Enterprise Service Bus (ESB)

- Distributed services architecture
- All systems follow the same standards
- Any new system can plug into the bus
- Highly scalable



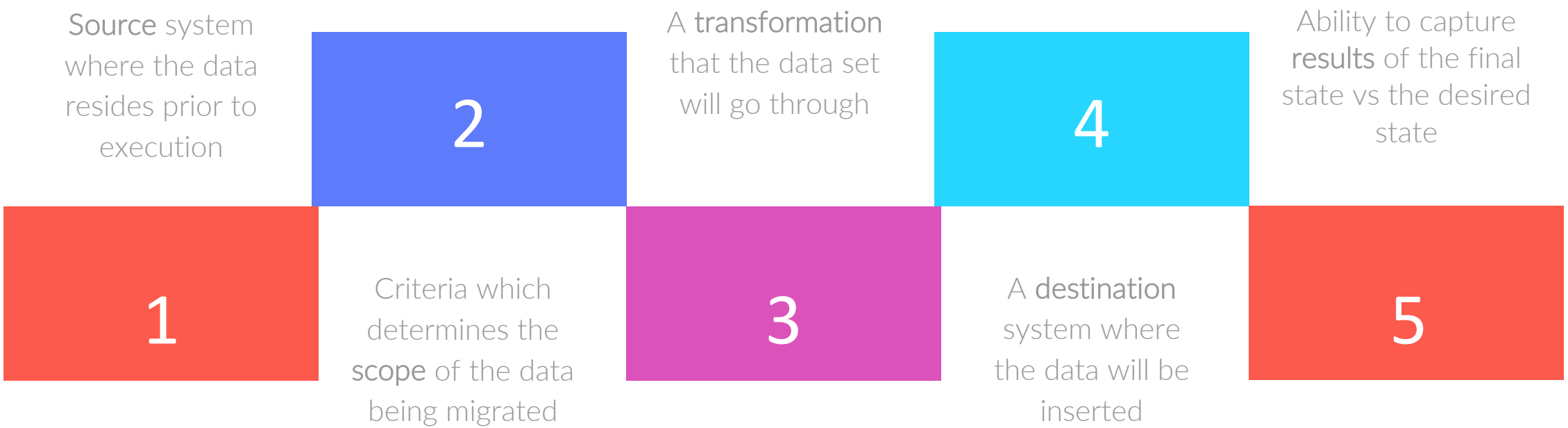
ESB Integration

Migration

The act of moving a specific set of data at a point in time from one system to the other

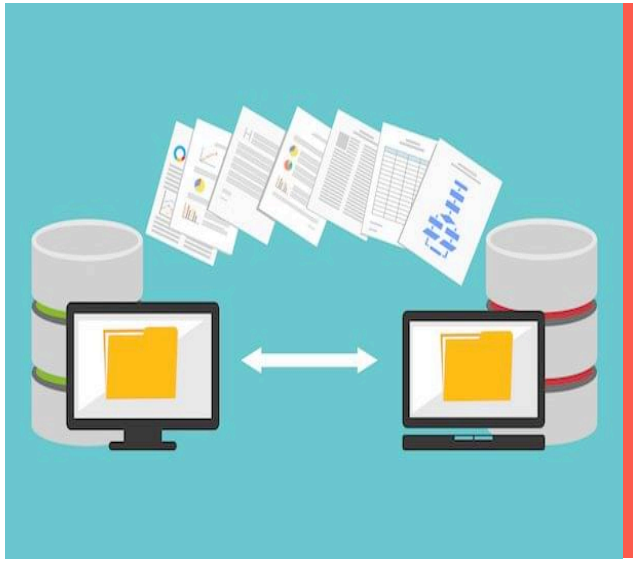
MIGRATION PATTERN

What is it?



MIGRATION PATTERN

Some Clarification



Data Migration

the act of moving data between systems

Application Migration

the act of moving functionality capabilities between systems

Business Data vs Business Capabilities

MIGRATION PATTERN

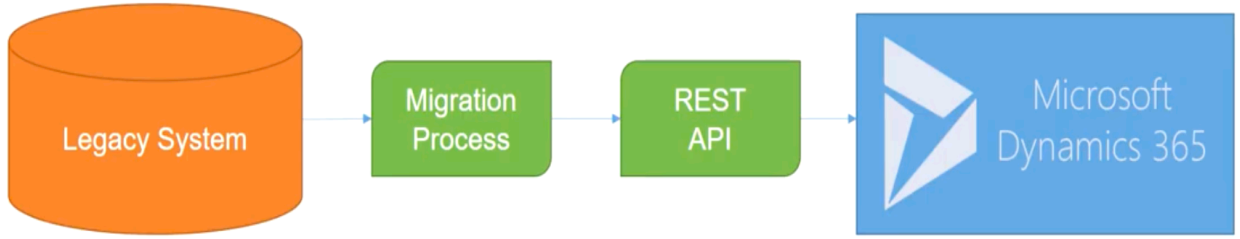
Is this really integration?

In the context of one-off, one-way copy of data from **one legacy system to another newer system**... The argument is unconvincing

MIGRATION PATTERN

A special type of integration

- Migration is truly **moving** a set of data at a point in time (a snapshot) from one system to another
- This **move** may occur **multiple times**

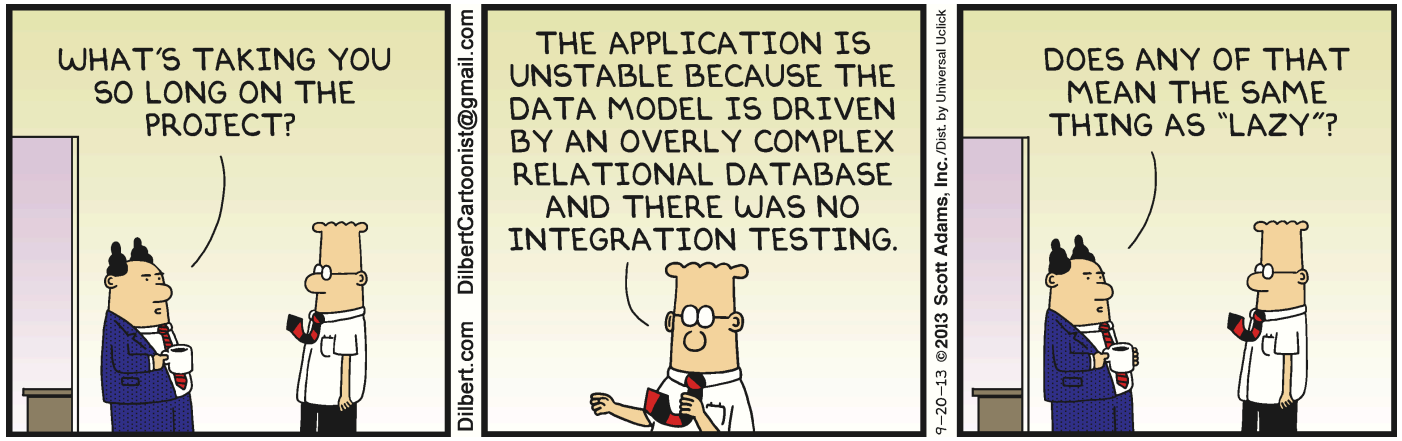


MIGRATION PATTERN

Why is it valuable?

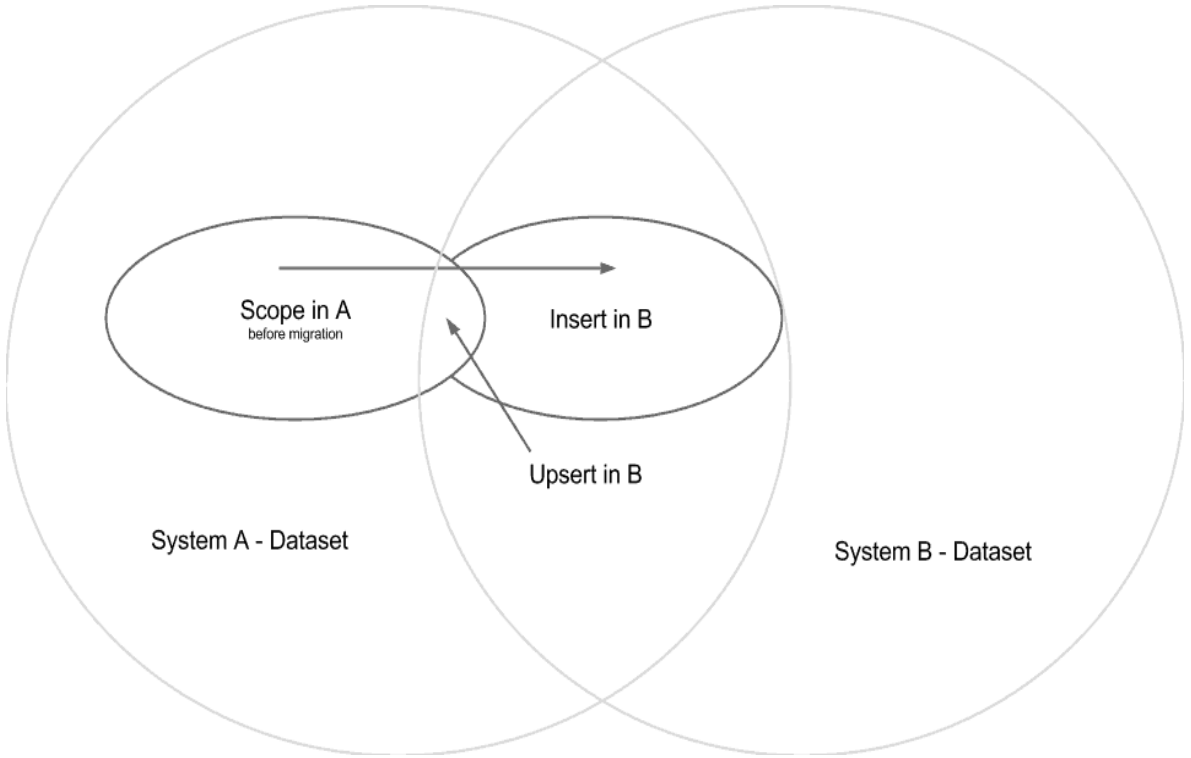
Keeps data agnostic from the tools that we use to create, view, and manage it

Without migration, we would lose any amassed data anytime that we changed systems



MIGRATION PATTERN

When is it useful?

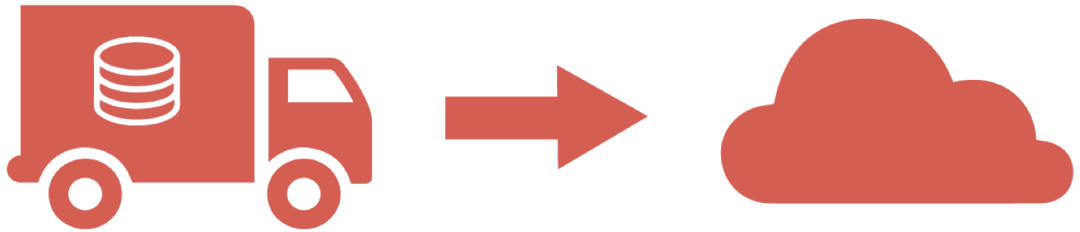


- **Moving** from one system to another
- **Moving** from one instance of a system to another
- **Backing up** a dataset
- **Adding** nodes to database clusters
- **Replacing** database hardware
- **Consolidating** systems

MIGRATION PATTERN

Consideration: Scoping the Dataset

- Migrations are a perfect time to clean up or at least move only the valuables to the new destination.
- It's tempting to just move everything, which tends to increase the scope of migrations in a futile effort to save potentially data with 0 value
- working with the data owner to reduce the scope of the data needing to be migrated can ensure only data that adds value is migrated



MIGRATION PATTERN

Consideration: Pull or Push?

Push Model

"once a specific event happens in the origin system, push all the data that matches a specific criteria to a processing app that will then insert it into the destination system"

Pull Model

"let the data collect in the origin system until I am ready to move it, once I am ready, pull all the data that matches my criteria and move it to the destination system."

MIGRATION PATTERN

Consideration: Triggering the Migration

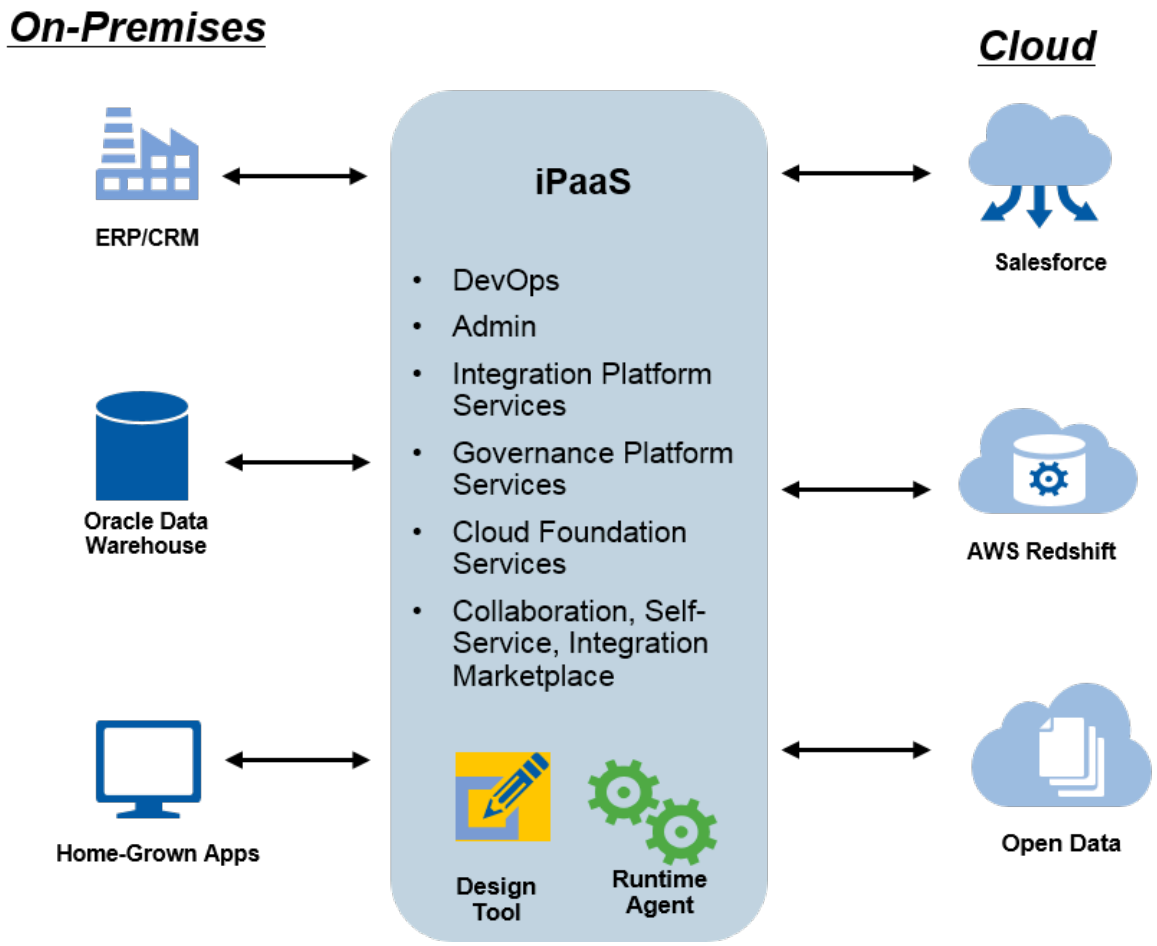
- Traditionally handled via scripts, custom code, or database management tools, that developers or IT operations will create
- IPaaS systems, such as Mulesoft or Dell Boomi, can capture migration scripts or apps as integration apps
- Having services for common migrations save your team a lot of time

MIGRATION PATTERN

What is an IPaaS?

Integration Platform as a Service

A cloud integration platform as a service, enabling connectivity to SaaS and cloud services and providing a secure method of accessing on-premises applications behind a firewall



Cost-efficiency

Reduces the need to generate code for customized integrations

Scalability

Can expand without having to worry about setting up another custom in-house integration

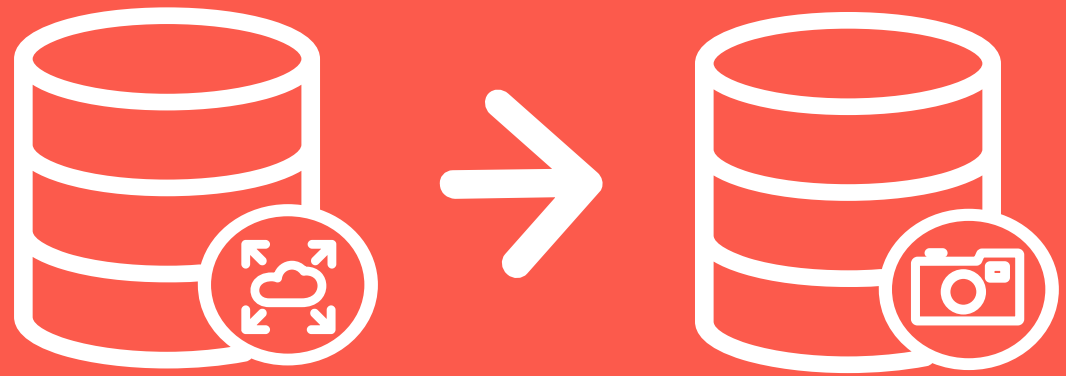
Ease of use

usually offers a bunch of prebuilt connectors that accelerate integration projects

MIGRATION PATTERN

Consideration: Snapshots

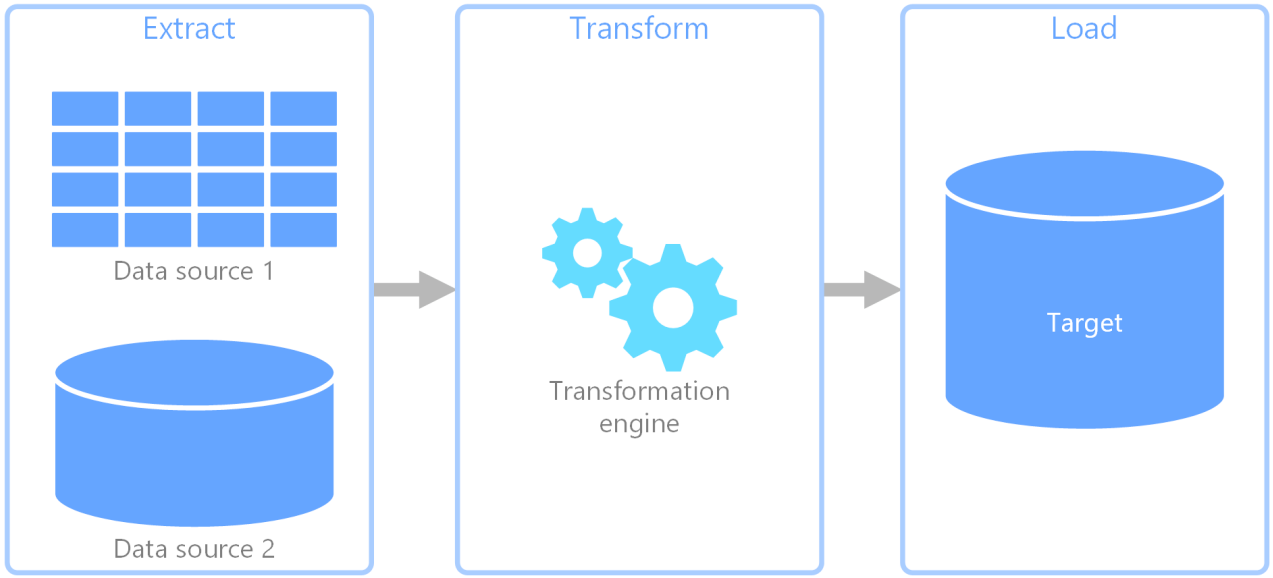
- Since migrations are theoretically executed at a point in time, any data which is created post execution will not be brought over
- To synchronize the ***delta*** you can use the broadcast pattern (which we'll see next)
- Keep in mind that data can still be created during the time that the migration is running unless you prevent it or create a separate table where new data can be stored



MIGRATION PATTERN

Consideration: Transforming the Dataset

Other than a few exceptions, like data backup/restore or migrating to new hardware, you usually need to modify the formats and structures of data during a migration



Extract
Process of reading data from a database

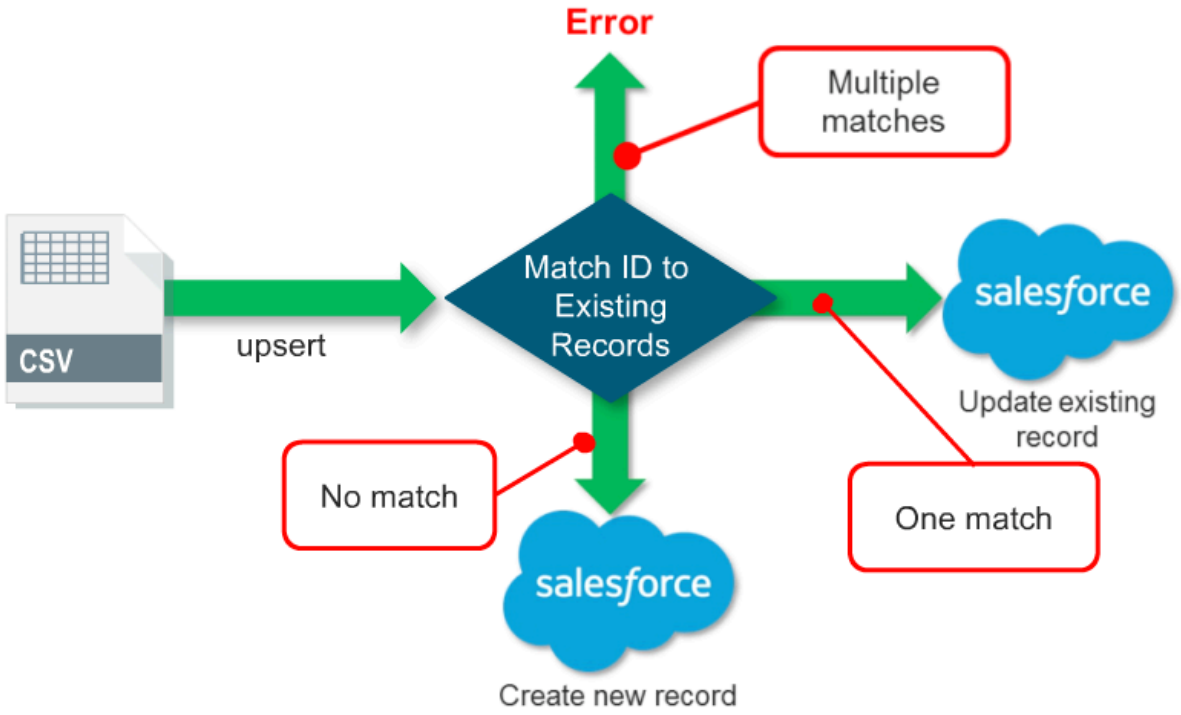
Transform
Process of converting extracted data from its previous form into the form it needs to be in so that it can be placed into another database

Load
Process of writing the data into the target database

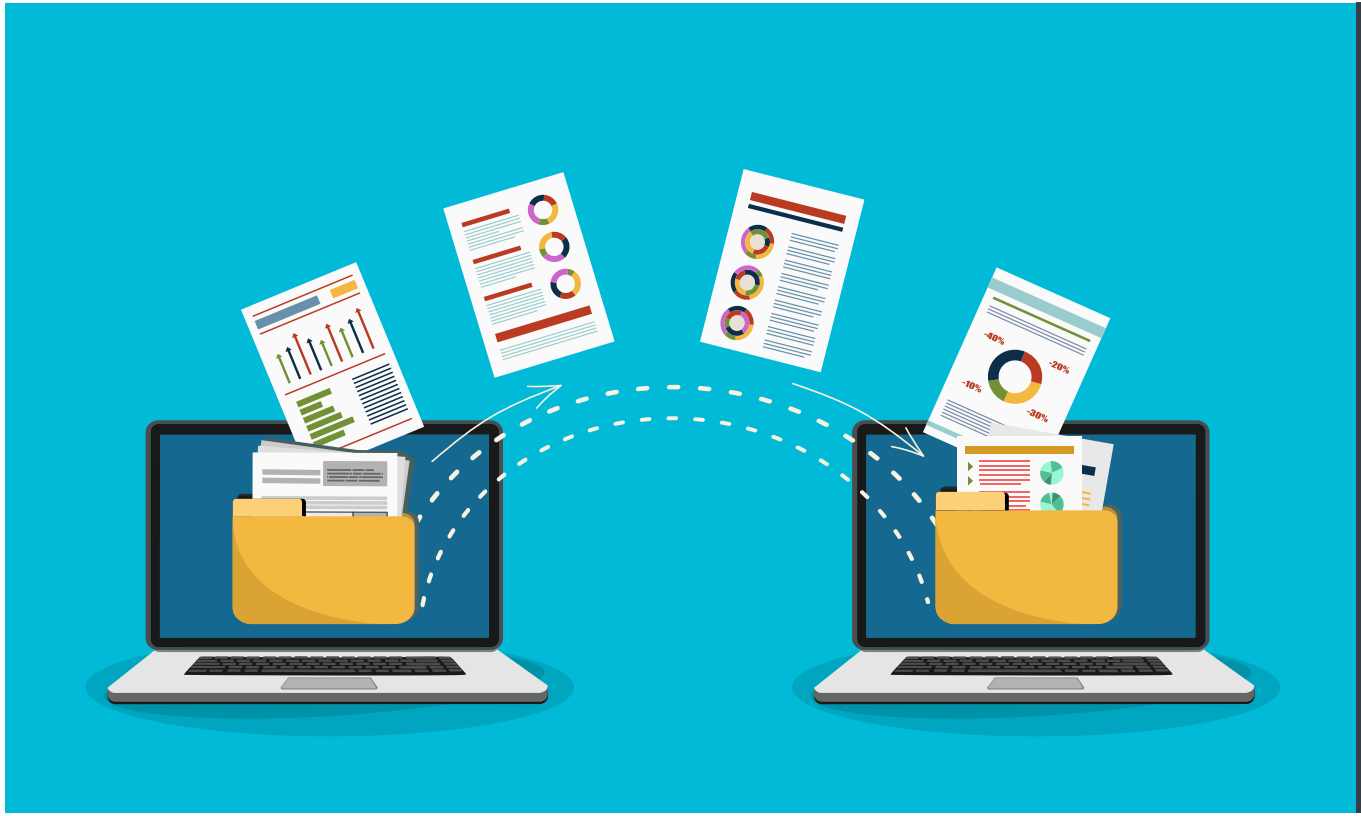
MIGRATION PATTERN

Consideration: Insert vs Update

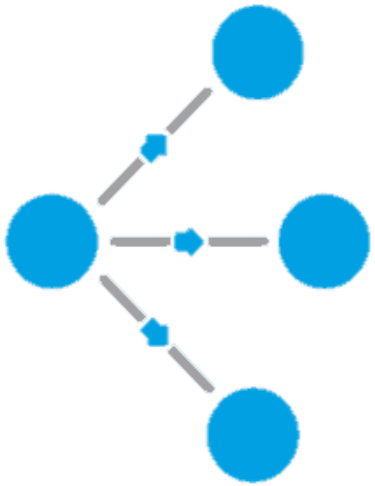
- When running a migration multiple times you may need to handle both creating and updating records
- Duplicate or incomplete entries can be extremely costly
- Intelligently picking the right fields to **upsert** on is especially important if the target system is the system of record
- Having a solution that checks for duplicates can make a huge difference, both proactively and reactively



Summary



- Migration is the one time act of taking a specific data set from one system and moving it to another
- Creating an application for the purpose of migrations is valuable because you can make it a reusable service



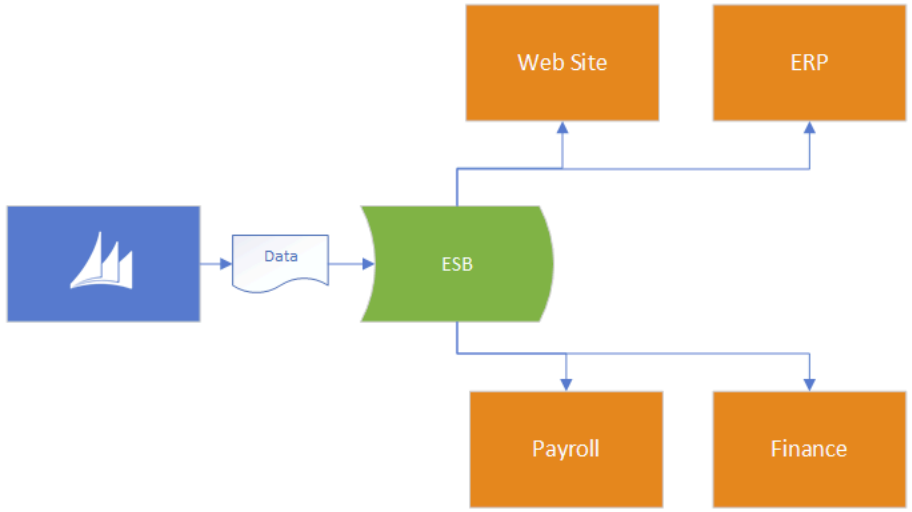
Broadcast

one way sync from one to many

BROADCAST PATTERN

What is It?

- Move of data from a single source system to many destinations
- Keeps data up to date between multiple systems, across time
- Like the migration pattern, only moves data in one direction, from the source to the destination



Migration vs. Broadcast

What separates using the broadcast pattern and the migration pattern set to automatically run every few seconds?

Migration

Full scope

large volumes

Parallel processing

graceful failure

Broadcast

Change capture

“as quickly as possible”

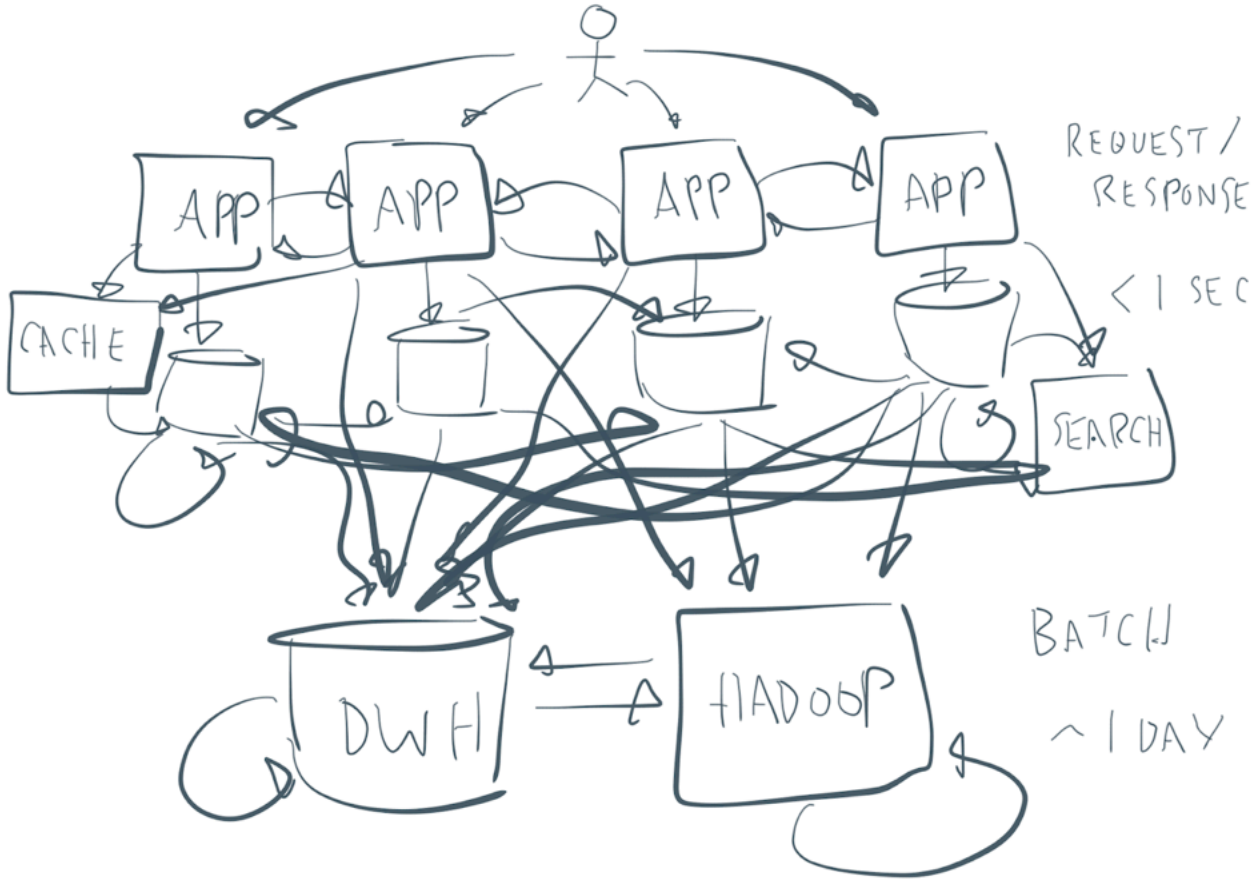
transactional

highly reliable and available

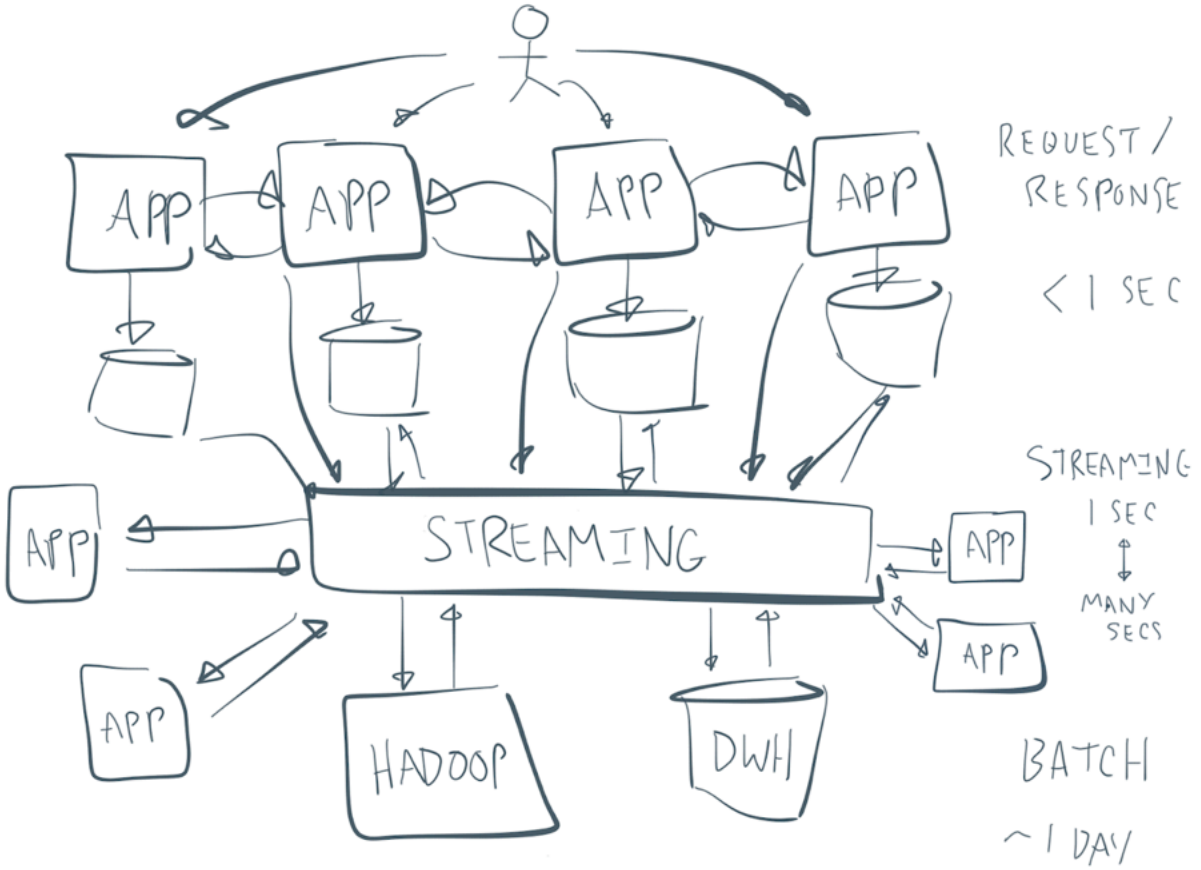
Transactional

If a data transfer (“transaction”) succeeds, data is committed (i.e. persisted) at destination

Note about EDA (Event Driven Architecture)



Note about EDA (Event Driven Architecture)

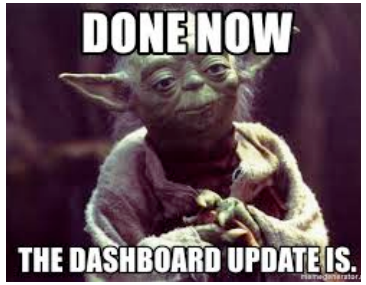


Simply put, the event is a significant change in state, which is triggered when a user takes an action.

BROADCAST PATTERN

Why is it valuable?

Handles any generic situation where a system needs to know information in near real time residing in another system



Use Case I

A real time reporting dashboard that receives updates from multiple systems in real time

Use Case II

A fulfilment processing system that immediately starts processing orders from your CRM, online e-shop, or internal tool

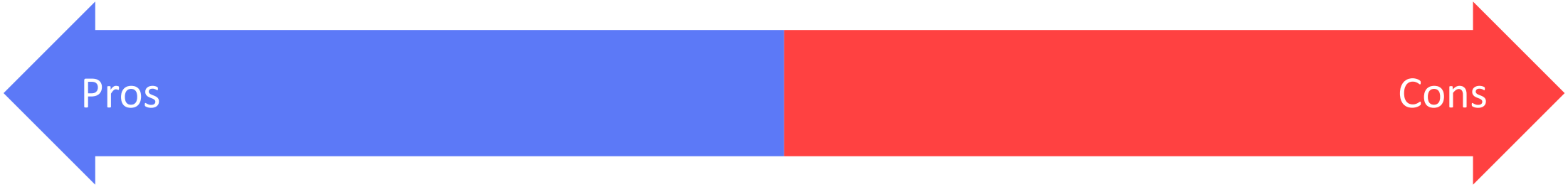
BROADCAST PATTERN

When is it Useful?

- A system needs to know as soon as an event in another system happens
- Data needs to flow from A to B automatically, without human involvement
- System A doesn't need to know what happens with the object in system B

Consideration: Input Flow

Option 1: Data Passed via Message Notification Payload

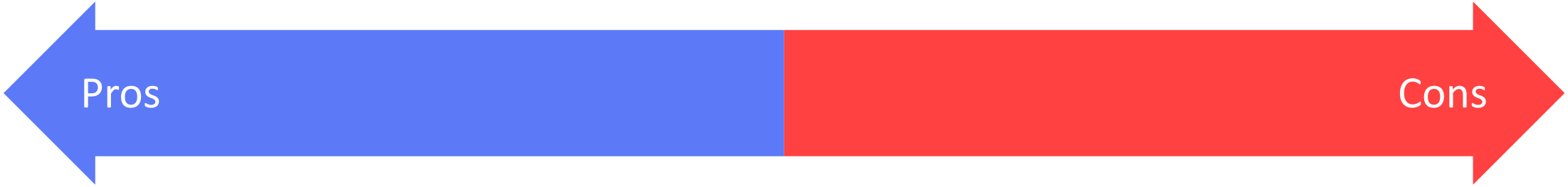


- minimizing the number of API calls

- code and/or configuration that sends notification needs to be developed differently for each source
- can become “chatty” when the source system has a lot of changes
- if the integration application is not running properly or is down you could lose data

Consideration: Input Flow

Option 2: Integration application pulls data from the source system



- pull will only run when the application is healthy
- watermark only updated once message is successfully processed

- still requires the development of the outbound message notifying the integration application that there is data waiting to be pulled
- can become “chatty” when the source system has a lot of changes

Consideration: Object Payload

- Moving field values as payloads is very expensive in terms of development cost and performance
- This means synchronizing whole objects even though only one field has changed, exposing a problem when you have multiple broadcasts feeding into the same system


```
{
  "id": "1",
  "type": "cars",
  "cars_array": [
    {
      "id": "1",
      "name": "Mercedes Benz",
      "desc": "Mercedes Benz is a global automobile marque and a
        division of the German company Daimler AG. The brand is
        known for luxury vehicles, buses, coaches, and lorries.
        The headquarters is in Stuttgart, Baden-Württemberg. The
        name first appeared in 1926 under Daimler-Benz."
    },
    {
      "id": "2",
      "name": "BMW",
      "desc": "BMW AG, originally an initialism for Bayerische
        Motoren Werke in German, or Bavarian Motor Works in
        English) is a German multinational company which
        currently produces luxury automobiles and motorcycles,
        and also produced aircraft engines until 1945."
    }
  ]
}
```

BROADCAST PATTERN

Consideration: Field Data

SCENARIO

CRM A and CRM B both broadcasting contacts into CRM C and you need to handle cases where both systems change the same or different fields on the same contact

Option 1

Merges the values of the two payloads, with the later one winning if the same field are affected

Option 2

Define a master in the integration application configuration at either the system, object or field level

BROADCAST PATTERN

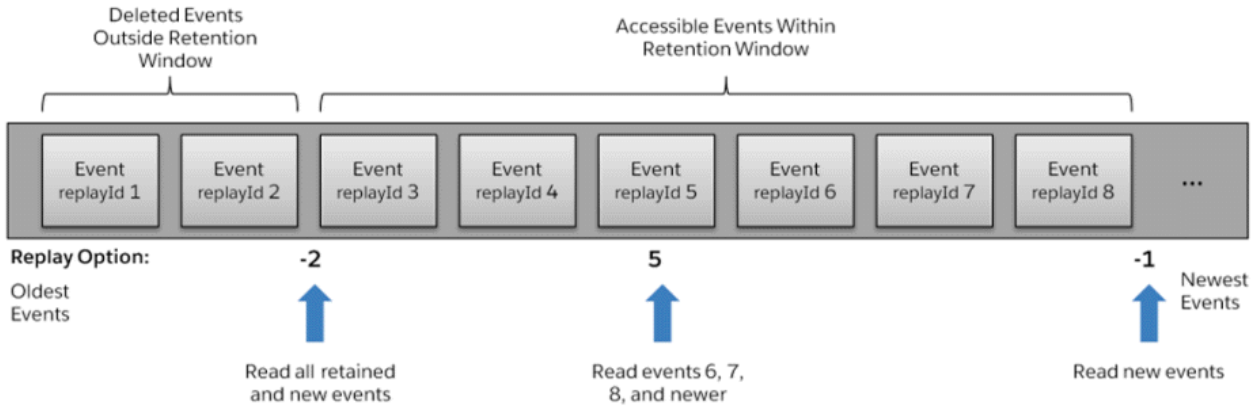
Exception Handling

- A simple exception management strategy can be to log all the issues to a file or send in an email
- Create a notification workflow based on the values in the payload so that different people are notified based on different errors for those exceptions that cannot be fixed by a prescriptive design time solution.

```
try {
  something();
}
catch(e) {
  window.location.href =
  "http://stackoverflow.com/search?q=[js]+" + e.message;
}
```

Consideration: Watermark (Replay)

- Keeps track of when the last poll was done such that we grab only the newest set of objects that have been modified in that time window
- In systems without watermarking, you may have to either process all the records, or produce a clever way to capture only those items to be processed
- In general, using a *last modified timestamp* as the replayid provides the cleanest way to pull only the latest content



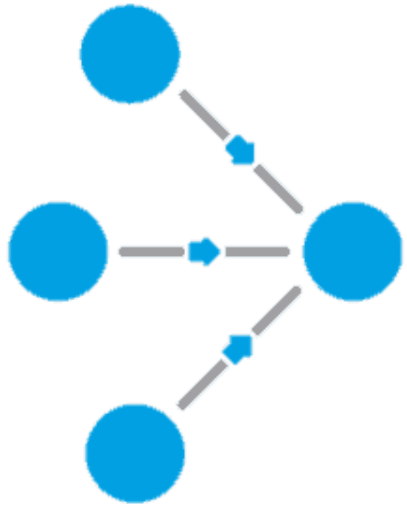
BROADCAST PATTERN

Summary

- Defined as the act of moving data from one to one or more systems in as near real-time fashion
- Valuable when you have data in one system and would like to have that data in other systems as soon as possible.

Aggregation

The act of taking or receiving data from multiple systems and inserting into one



SCENARIO

Your company has customer data in three different systems, and you need to generate a report which uses data from all three of those systems.

OPTION 1

a daily migration from each of those systems to a data repository and then query against that database.

- would have a whole other database to worry about and keep synchronized.
- As things change in the three other systems, would have to constantly ensure that the data repository is kept up to date
- Information would be a day old

OPTION 2

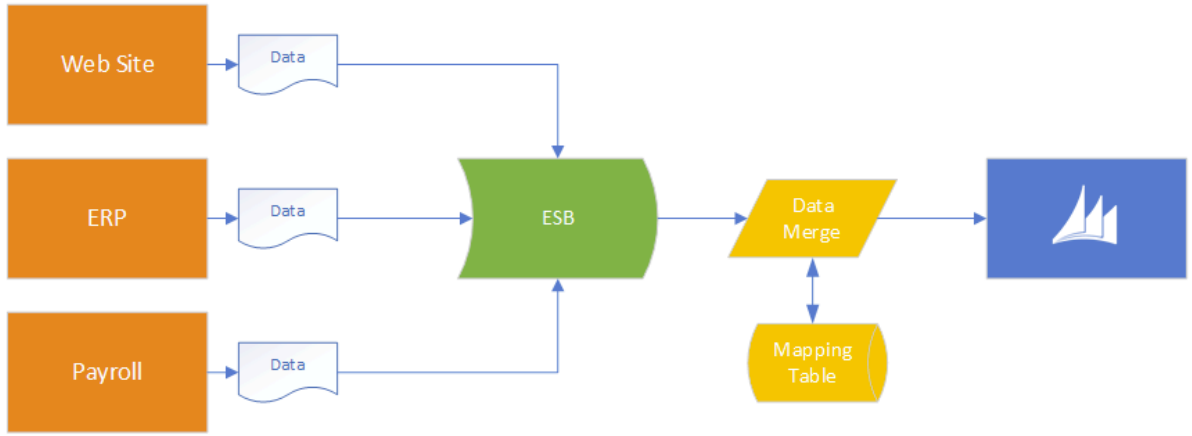
set up three broadcast applications so that the reporting database is always up to date

- would need to maintain a database whose only purpose is to store replicated data so that I can query it every so often.
- Not to mention the number of wasted API calls to ensure that the database is always up to x minutes from reality.

AGGREGATION PATTERN

What is It?

- The act of taking or receiving data from multiple systems and inserting into one
- On demand query multiple systems merge the data set, and do as you please with it
- From our scenario, you can build an integration app which queries the various systems, merges the data and then produces a report



AGGREGATION PATTERN

Why is it valuable?

- Allows you to extract and process data from multiple systems in one application
- Data is up to date at the time that you need it, does not get replicated, and can be processed/merged to produce the dataset you want

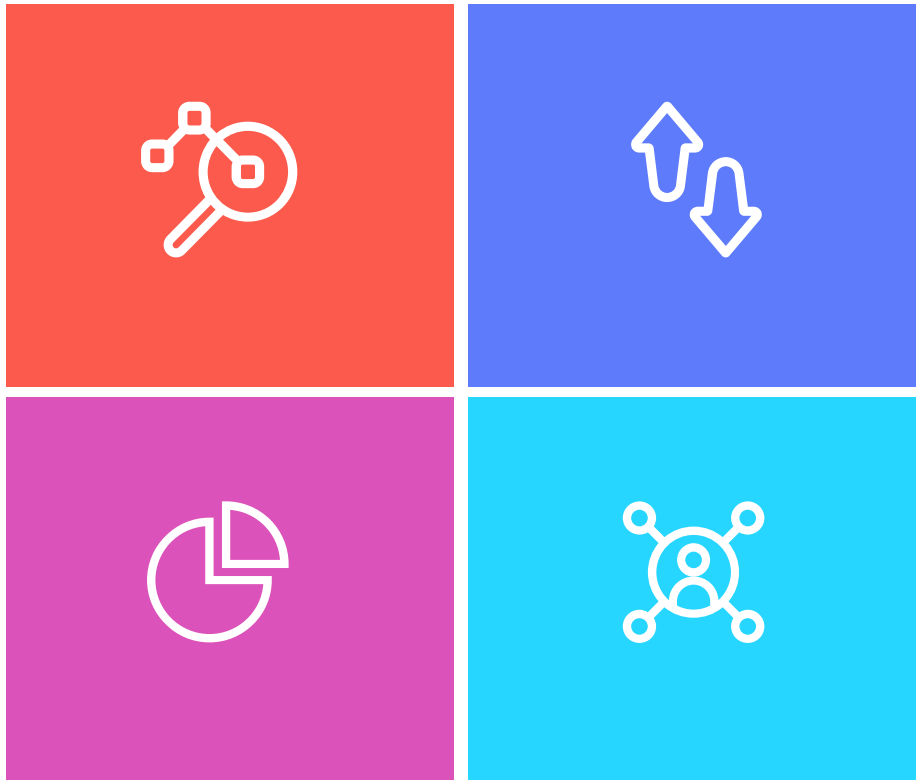
AGGREGATION PATTERN

When is it useful?

The aggregation pattern is valuable if you are creating:

Orchestration APIs

to “modernize” legacy systems



Compliance

auditing systems which need to have related data from multiple systems

Reports or Dashboards

which must pull data from multiple systems and create an experience with that data

Ease of Use

reduce the amount of learning that needs to take place across the various systems to ensure visibility into what is going on

Consideration: Collecting Data

Listener

create a listener which waits for messages from multiple systems and aggregates them in real time

Trigger

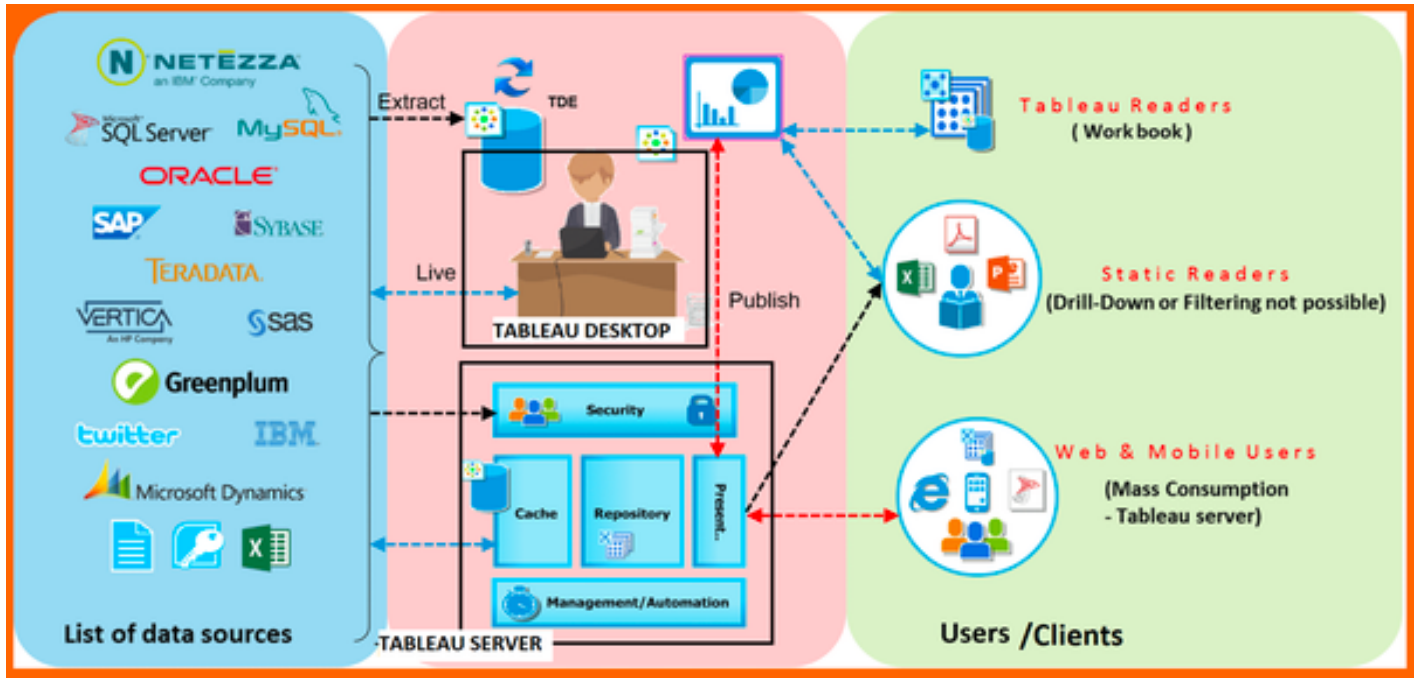
create an application that is triggered, processes, and returns the results

In general, reserve the aggregation pattern for collection of data on demand or for orchestration of data rather than synchronization

Consideration: Migration from Many to One vs Aggregation

- Although you can create a migration that collects data from many systems and then inserts them into one system, a migration is realistically the movement of existing data to the destination system where the data will reside.

- aggregation is meant to correlate, merge, and potentially transform the object and usually not storing the new data set into a system where it will continue to be used while deprecating the origin.



AGGREGATION PATTERN

Consideration: Source Data Scope

- Since we use different connectors for each system we are pulling data from you have the ability to write different queries to pull data from those systems

- This means that you can do things like get accounts that are older than 2 years in one system and account that are older that 1 year in the other

AGGREGATION PATTERN

Consideration: Merging the multiple datasets

- Once you have queried across the systems, you will most likely have stored the results in the local store for the duration of the app running. At this point, you need to choose how to merge the datasets or what to do with it.

Summary

- The aggregation pattern Is very useful when building APIs that collect and return data from multiple systems
- can be modified to merge and format the data as you would like and can be easily extended to insert the data into multiple systems

Bi-directional Synchronization

/baɪdɪˈrɛkʃənəl/

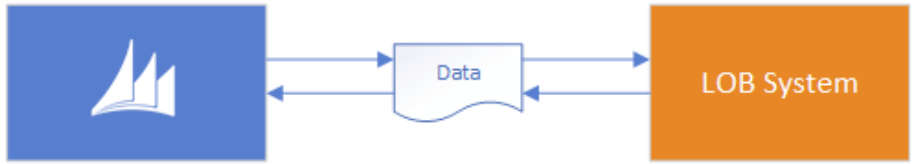
adjective
Moving or operating in two usually opposite directions

BI-DIRECTIONAL SYNC PATTERN

What is It?

- The act of unioning two datasets in two different systems to behave as one while respecting their need to exist as different datasets

- Different tools or different systems for accomplishing different functions on the same data set

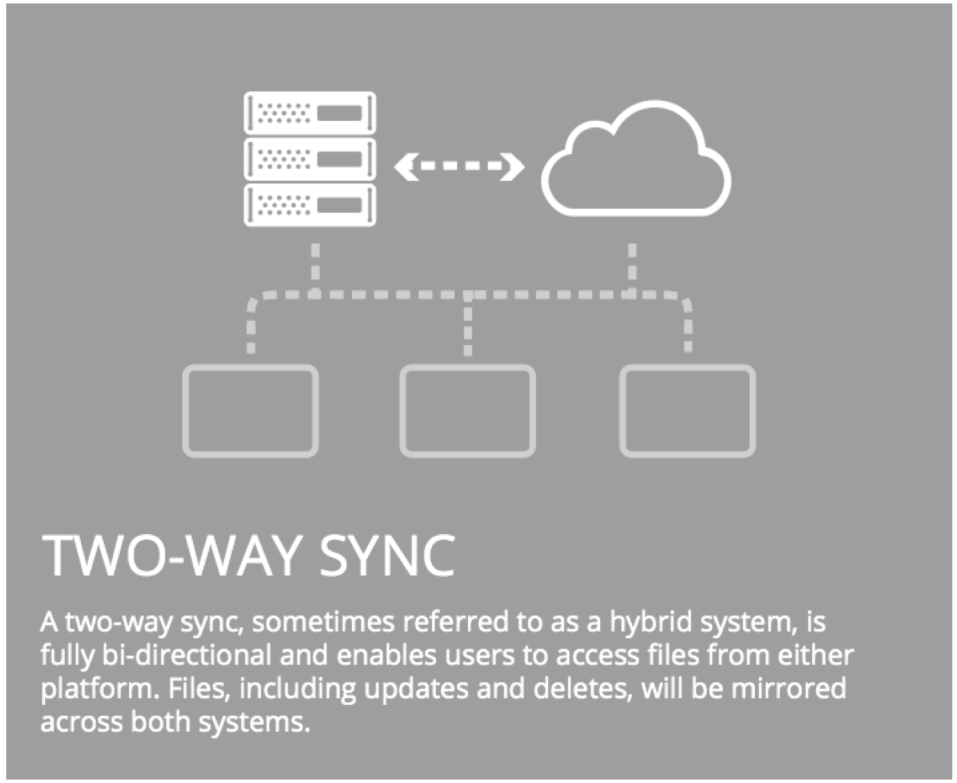


BI-DIRECTIONAL SYNC PATTERN

Why is it valuable?

- In a situation where you have two or more independent, isolated, representations of the same reality

- Bi-directional sync takes you from a suite of products that work well together but may not be the best at their own individual function, to a suite that you hand pick and integrate together using an integration platform.



BI-DIRECTIONAL SYNC PATTERN

Why is it valuable?

Single View of the Customer

Manually giving everyone access to all the systems that have a representation of the notion of a customer can hinder productivity, training, security, and cost

Bi-directional synchronization allows users to have a realtime view of the same customer within the perspective that they care about.

Your Data Expectations vs Reality

Consideration: Filtering and Object Mapping

SCENARIO

You have two Salesforce instances, one for the Ohio Sales team, and one for the rest of the US. If you implemented a bidirectional sync, instead of always moving all contacts from one to the other, you could set a filter like “From OH>US move all contacts that are in US but not OH” and “From US->OH move all contacts that are OH”.

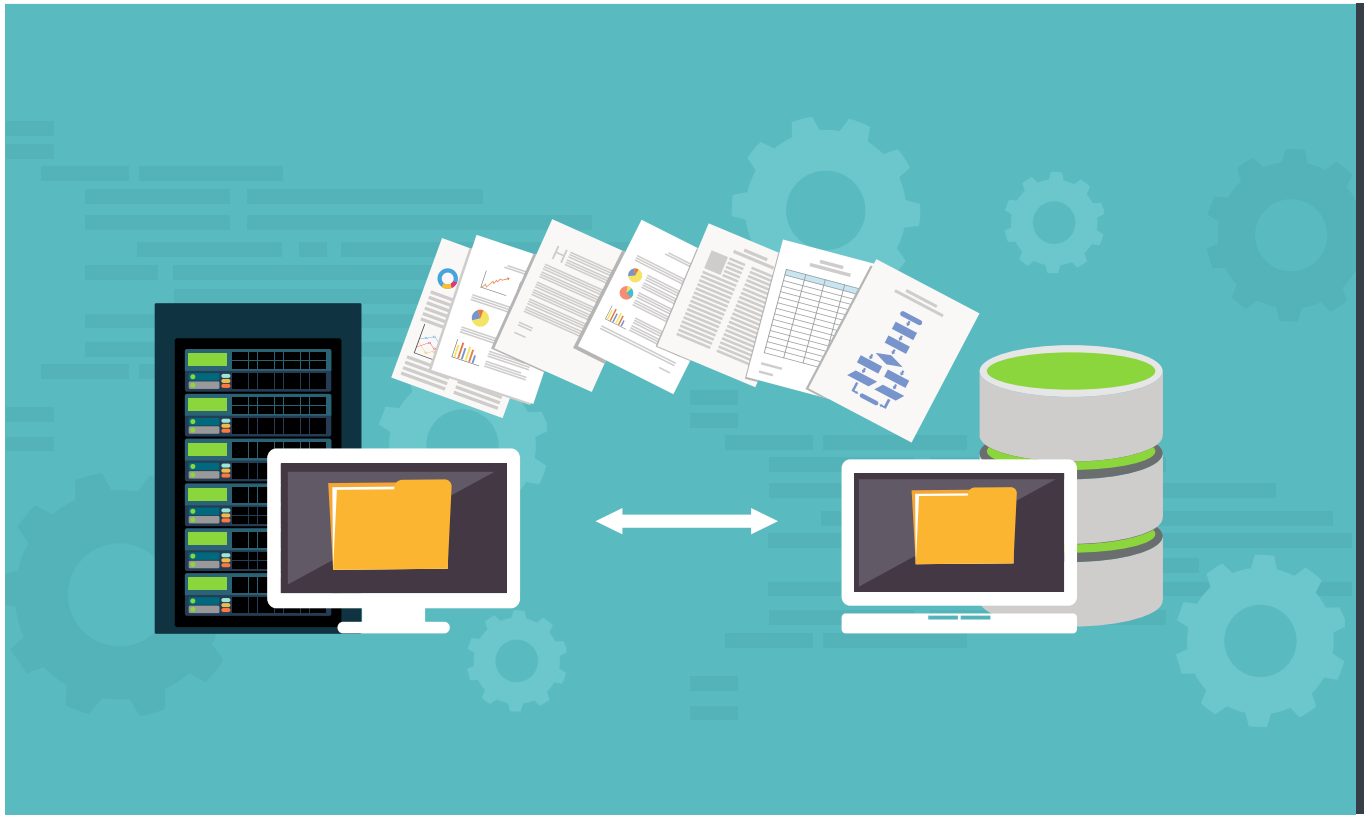
- Like a pair of broadcast application with the ability to scope and map the data in both directions
- With these kinds of filters you will make it so that you don’t synchronize contacts which make sense in your instance, but have no value in the other instance

BI-DIRECTIONAL SYNC PATTERN

Consideration: 3 or More Systems

- Need to change to a different architecture and start introducing the notion of master
- Consider a hub and spoke architecture:
  one system will be the system of record for information needing synchronized
- Alternatively. decouple the systems using queues and a pub-sub model where every system will broadcast its change to a queue and system can look for changes that they care about

Summary



- Effective when different systems need to accomplish different functions on the same data set
- Allows you to create suites of products that can work in harmony together

Correlation

/ˌkôrəˈlāSH(ə)n/

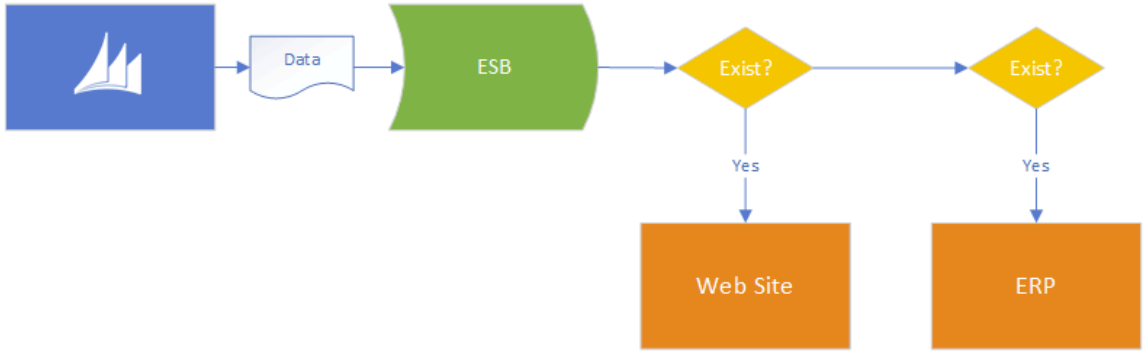
noun
1. a mutual relationship or connection between two or more things.

"research showed a clear **correlation** between recession and levels of property crime"

CORRELATION PATTERN

What is It?

- Identifies the intersection of two data sets and does a bi-directional synchronization only if record occurs in both systems naturally
- Unlike the bi-directional pattern, which synchronizes the union of the scoped dataset, correlation synchronizes the intersection
- Does not care where objects came from; agnostically synchronize as long as data is found in both systems



CORRELATION PATTERN

Why is it valuable?

- Useful in the case where you have two groups or systems that want to share data only if they both have a record representing the same item/person in reality

- Only bi-directionally synchronizes the objects on a “Need to know” basis rather than always moving the full scope of a dataset in both directions

“Hey, lets share data between the two hospital so if a patient uses either hospital, we will have a up to date record of what treatment they received at both hospitals.”

CORRELATION PATTERN

When is useful?

SCENARIO

You are a university, within a larger university system, and you are looking to generate reports across your students.

You don't want a bunch of students in those reports that never attended your university, but you want to include the units that those students completed at other universities in your university system.

Consideration: Bi-Directional Factor

What are key things to keep in mind when building applications using the correlation pattern?

Given that the correlation pattern is almost the same as the bi-directional sync, the same considerations apply.

The main difference is in the impact of how you define the Same.

Consideration: Definition of Same

Definition differs by industry and use case and the degree to which it is harmful will also differ by industry and use case

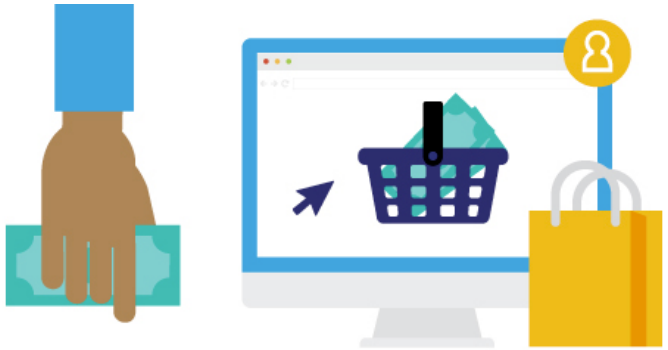
	Just Right	Too Strict	Too Loose
Bi-Directional Sync	Union of Data Sets	Duplicate Created	Wrong Records Merged
Correlation	Intersection of Data Sets	Not Synchronized	Wrong Records Merged

CORRELATION PATTERN

Consideration: Definition of Same

When keeping it real goes wrong

What would happen if if the criteria that evaluates if two records in two different systems are the same is incorrectly defined?



SCENARIO

An ecommerce site where the definition of “same customer” is defined by a composition of a customer’s first and last name

If we send recommendations via email, We would end up with an error where two people with the same name would receive the union of both sets recommendations

CORRELATION PATTERN

Consideration: Definition of Same

SCENARIO

Two babies named Kylo are born in the same hospital, with the last name Ren. Patient uniqueness in the hospital's systems is determined by

SSN + First Name + Last Name + City + Hospital

- It is paramount that you are very careful when picking the right ***definition of unique*** when dealing with high sensitivity information
- Likely they will share everything except SSN, so they likely are only apart in the hash by 1 wrong key stroke when filling in their info

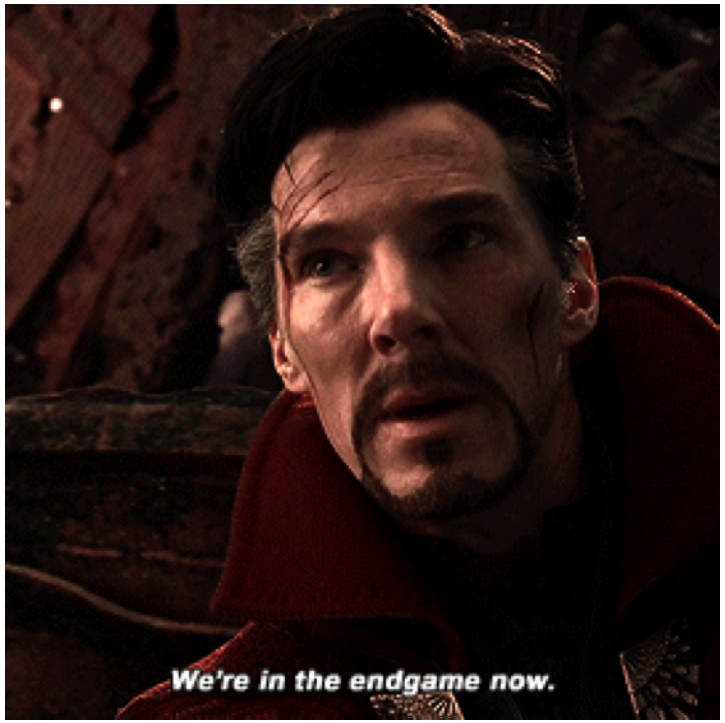
Summary

- The intersection of datasets as opposed to bi-directional sync which is the union of the datasets
- Allows you to keep two systems in sync at a lower "cost"

THE INTEGRATION GAME

Wrap-up

- Following these integration patterns will help the scalability and maintainability of your integrations
- Each of these patterns can be applied by themselves or composed
- Patterns have a time and place; “do what makes sense”



Thank You!