Caching In The Salesforce Platform

Caching is an essential part of any application. Moreover, caching is finding expanded relevance in modern, cloud-native, distributed application architectures because it helps end users experience better performance and allows processes to execute faster.

How does caching enhance end user experience?

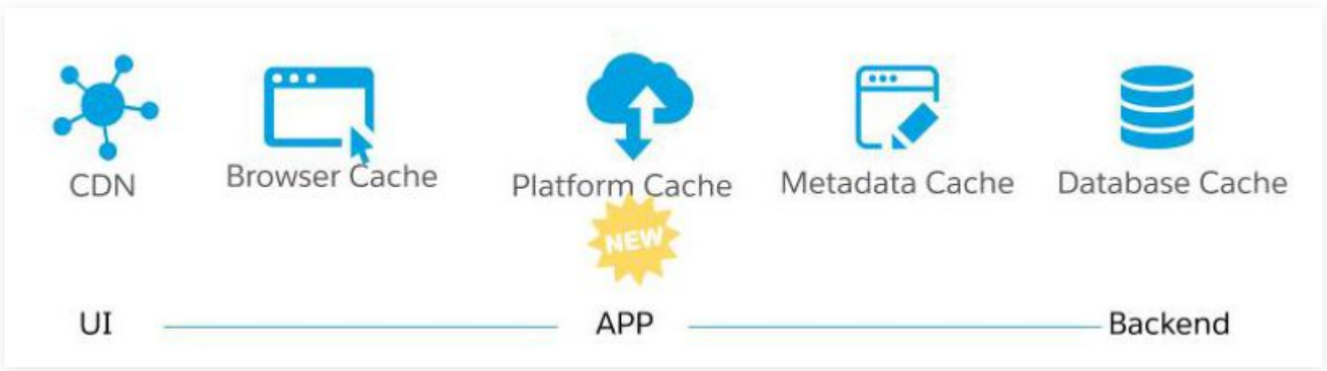
In a number of ways. Here are a few:

1. Caching reduces database consumption, frees up critical resources for other processes.
2. A cache stores data in-memory, this make retrieval very fast.
3. Cached is data closer to where the data is required i.e, processing or presentation layers this reduces the round trip time for transactions

To achieve this on the Salesforce platform, we can utilize a managed product called Platform Cache.
This blog post is for developers who want to create scalable and high performing applications on the Salesforce Platform using Platform Cache. We will review common use cases where caching is applicable as well as the related benefits.

Background

Caching has been around for a long time. As shown below, these days caching is present in all layers of an application stack and helps to speed up data access and processing in each layer.



A few examples of caching in an application stack:

1. Content delivery networks (CDN) cache static data closer to the user's location, significantly improving response times.
2. Browsers use cache called indexDB for applications like Lightning for fast page loads.
3. In the database, most query results are served from this buffer cache.

The application layer needs a cache too and Salesforce provides one.

Platform Cache

Imagine having lots of memory available to your code to store any data without having to worry about transaction boundary, heap size, or memory leak; maybe even using gigabytes of space. **Platform Cache** is the store that provides this**.**

Platform Cache is a **Redis-backed key value store** available for your Salesforce applications that can be accessed from Apex. This includes **triggers**, custom **web services** or Apex controllers for your **Lightning** components. It is accessed by using the ***APIs** *provided by Platform Cache.

Try it out

DE Orgs can request **10 MB** of Platform Cache, which is provisioned immediately. Enterprise and Unlimited licenses come with **30MB** by default and can get up to 3GB upon request.

What can be cached
Platform Cache can store data used by your Apex code for up to 48 hours. Data can be:

1. Records or sObject results of a SOQL query
2. Static data such as object metadata or user profiles, product catalogs
3. Apex lists of sObjects, strings or numbers
4. Other datastructures like Sets, Hashset etc.

How does it work
Platform Cache provides simple APIs that can be called from Apex to add data to cache or retrieve data from cache.
Here's a simple example of caching a list of strings:

```
String[] a1 = ['asa','abc','cdc','nop'];
Cache.Org.put('key1', a1  ,300); //300 indicates the time to live (TTL) in seconds.
```

And to retrieve the same list:

```
String[] a1 = (LIST)Cache.Org.get('key1');
```

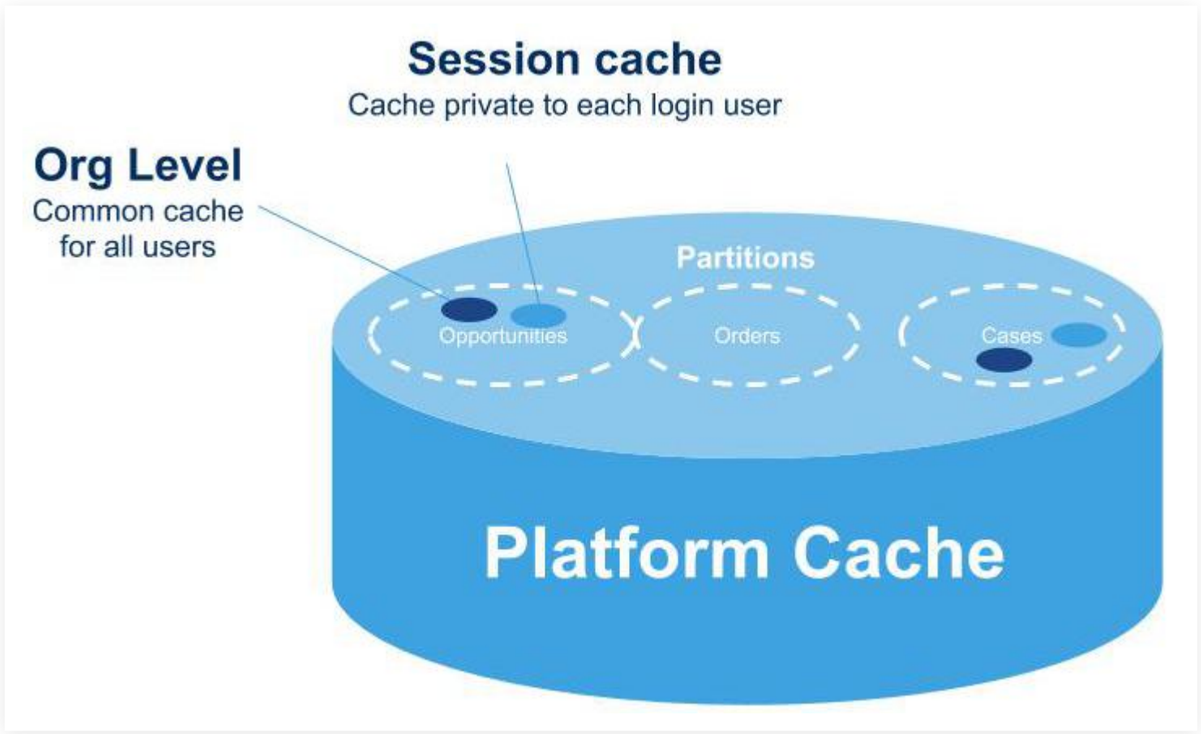
We'll look at more detailed examples in the use cases section below.

Features for designing your cache

Partitions & cache instances
Platform Cache can be partitioned and each partition can hold different data. For example, leads can go into the leads partition while opportunity triggers can go in the opportunities partition. Each partition can be further **divided** into **org cache** and **session cache instances.**

1. Session cache is unique to each user session, only the user session can write and read from the session cache
2. Org cache, where data is visible to all users and applications in the org. Therefore, any process (such as a trigger) can write to the org cache

Different use cases can benefit from these different cache types.



Use cases for caching

A Salesforce org is dynamic, with data flowing in, out and changing often. Salesforce internal analysis indicates that **reads** (querying) accounts for about **70%** of the activity in an org, while **30%** are **writes** or **updates**.
So caching data during the interval between changes is beneficial for performance and scalability of the org due to the high **volume** of reads during this interval.

In the following sections, we'll explore use cases where Platform Cache can improve performance and scalability in an Org. They follow the pattern of caching the high volume of reads even if only for short amounts of time. These use cases are based on real-world customer implementations of Platform Cache.

1. Caching data in Apex triggers

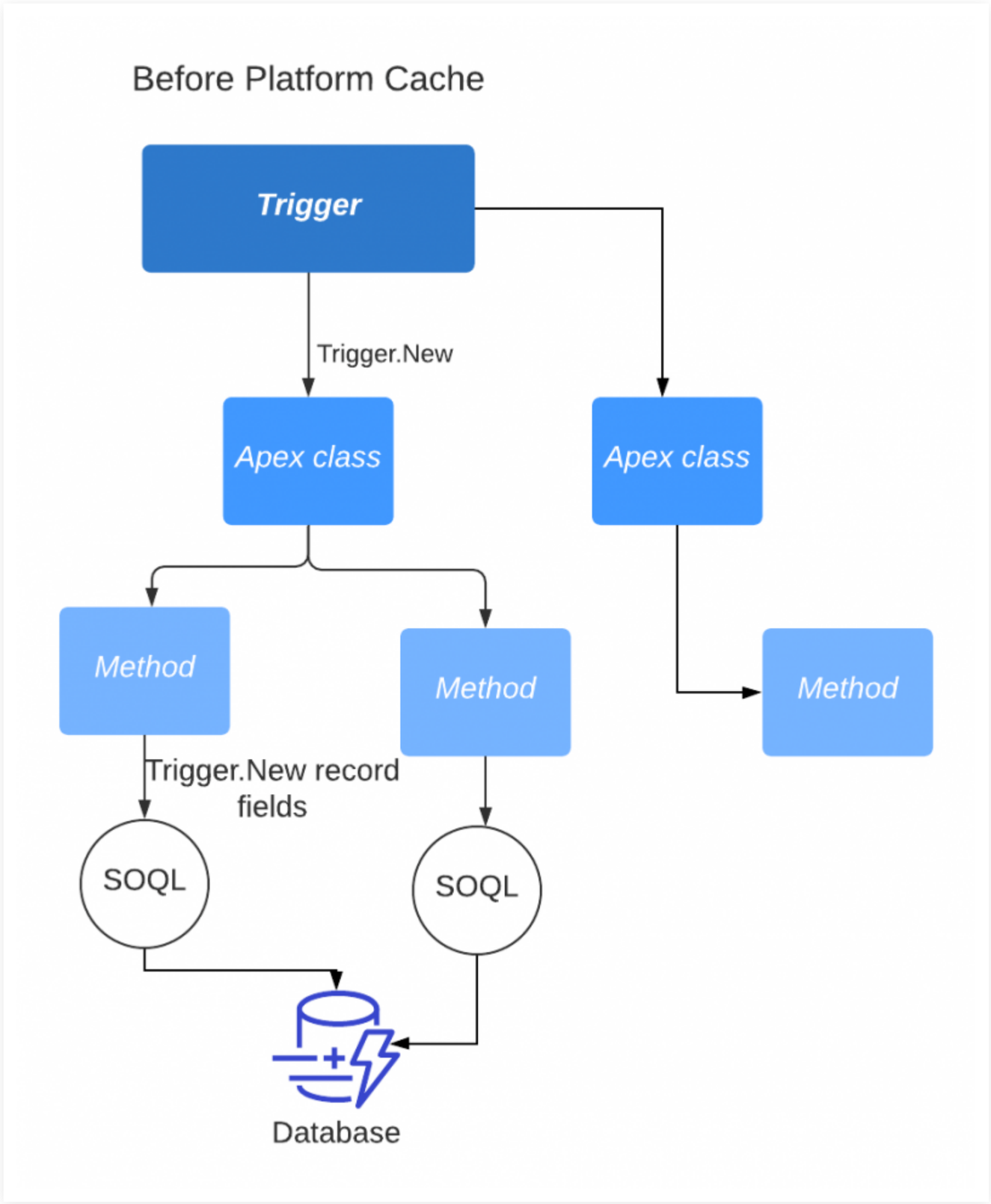
Trigger executions account for a huge volume of operations in the Salesforce Platform. Multiple versions of triggers execute for the same object, these triggers can be in different managed packages provided by ISV's or a standalone trigger.

Benefits

Caching data in Apex triggers helps reduce round trips to the database as the data in cache is closer to the trigger execution. Because the retrieval of data from cache takes less than 10ms, this improves performance of a transaction and greatly improves user experience.

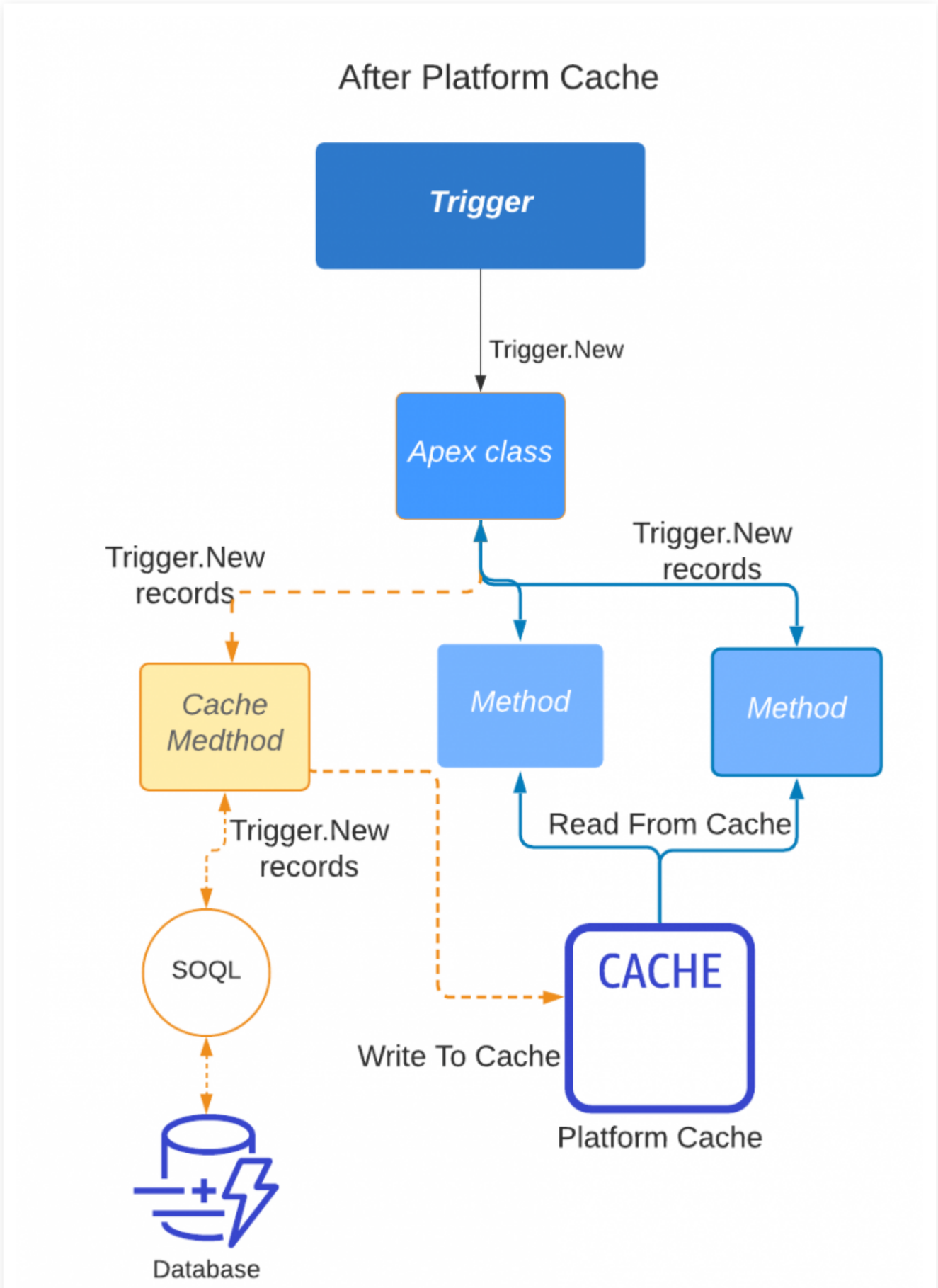
How it works
A trigger executes on a record or list of records that have changed (insert, update, delete, undelete). The changed records are stored in a variable called **Trigger.New** and input to the trigger code**.**
A common pattern is where the trigger calls a set of dependent Apex classes to execute complex logic. Triggers like *before update*, *after update*, *after insert* can query the trigger object multiple times for data in **relationship fields,**using IDs in Trigger.New as a filter to the query.

Regular flow of Apex triggers before Platform Cache:



An alternate option is to query a superset of fields from the trigger object and add it to the cache in an initial method.
All other methods will use the data from cache instead of doing individual database queries. This can be referred to as a cache aside (lazy loading) pattern. A cache utility class can be used which gets called from all trigger and dependent Apex classes for all cache related operations.

Flow of Apex triggers after Platform Cache:



Cache type to use

Use org cache for this use case as triggers cannot write to a session cache. Org cache is safe as the data is not expected to change during the trigger execution.

Here's a sample code in an Apex class called cache utility which can be called from a trigger to caches and retrieve opportunity records.

```
public with sharing class CacheUtility {
public static Boolean isOpptyCached = false;
public static void updateCache(List<Opportunity> opptyList) {
<i>//OpptyList is set of records in Trigger.new//</i>

List<Opportunity> cachedopplist = new List<Opportunity>();
cachedopplist = [SELECT ID,NAME,ACCOUNT.NAME.... from Opportunity
                                        WHERE ID in :newOppList]
if (!cachedopplist.isEmpty()) {
            for (Opportunity cdp: cachedopplist) {
                <i>//Add to cache, TTL set to 5 mins(300 seconds)//</i>
<i>                </i><i>//cdp.Id is the Opportunity ID and it's the key for the cache//</i>
                Cache.Org.put(cdp.Id, JSON.serialize(cdp),300);
                isOpptyCached = true;
            }
        }
        else {
     isOpptyCached = false;
      }
}
//Method to retrieve Opportunities from cache//
public static Opportunity getCacheOppty(String opptyId) {
        //opptyId is the key
        String opptyStr = (String)Cache.Org.get(opptyId);
        if (opptyStr!=null) {
            Opportunity opty = (Opportunity)JSON.deserialize(opptyStr,Opportunity.class); //Next version make Opportunity.class to <Object>.class to make this method and class gene
            return opty;
            }
            else {
                return null;
            }
    }
```

Here's a code snippet for using the cache utility class in a trigger & dependent Apex class.

```
List<Opportunity> opplist = new List<Opportunity>();
            if (!CacheUtility.isOpptyCached){
                for (String opty : OppIdSet) {
                    Opportunity oppty = CacheUtility.opptysMap.get(opty);
                    if (oppty != null) {
```

```
                        opplist.add(oppty);
                    }
                }
              else { //Fall back on the database//
                opplist = [SELECT Id, Nam FROM Opportunity
                                          WHERE Id IN :Trigger.New];
```

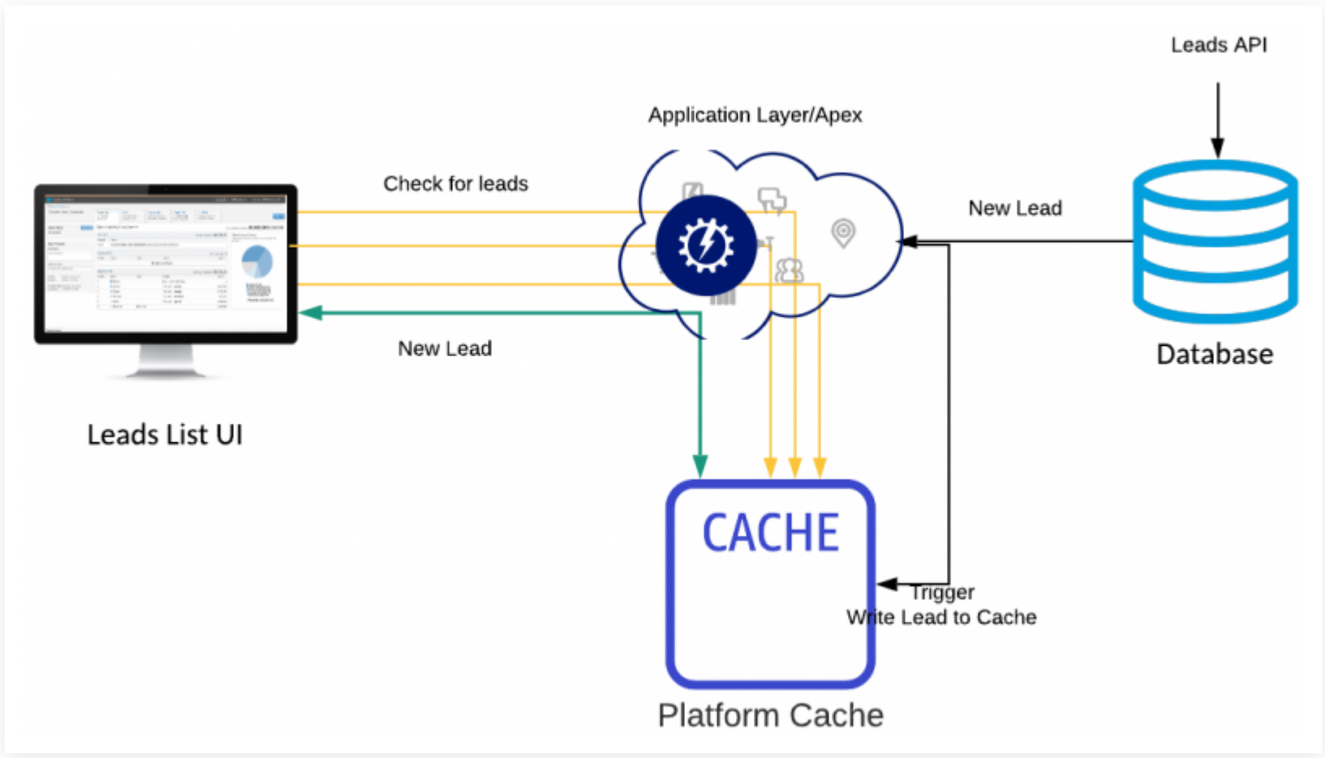
Note: We fall back to the original SOQL in case the cache does not have any data. This ensures there is no disruption to the transaction if the cache is empty.

2. Polling for updated data

This is a common use case… here the application needs to be notified if a record has been updated or created. An example: Sales users want to get to the latest leads assigned to them as quickly as possible. The earlier the lead is contacted, the better the chances are it converting to a sale. It's a typical case of polling for new data. Notification systems like CometD are well suited, however such notification systems cannot be used due to limitations.

Benefits of caching
Polling for new data has a huge impact on the database and scalability of your org. Having the data in cache can shield the database from all those requests, freeing up database resources for other application needs.

How it works
A newly created lead can be copied to the cache using an update or insert trigger. The application showing the list of leads can check the Platform Cache frequently via Apex and retrieve the latest lead. If there is no lead, the call just returns null, as shown in the code below:




```
//In Trigger
Cache.Org.put('00501001001','006xjxjjxq1k2AAQ'); //Add lead to cache with user ID
                                                 //as the key
```

Cache type to use
Org cache can be used for this use case. A **trigger** can **copy** the Lead to the cache with the **userID** as the key. The list of assigned Leads can then be displayed for each sales user from the Org cache.

3. Multiple API calls for same data

In an eCommerce application, a shopping cart is maintained by an external application using Salesforce API. The API will retrieve Order and Order Items each time the cart is updated. This can be resource intensive as the cart items can be changed multiple times by the user.

Benefits of caching
When a user interacts with a shopping cart, they expect fast responses. By reducing the number of round trips to the database for retrieval and updates to the cart by using the cache as intermediate storage, the API calls are much faster. In use cases with large numbers of users updating their shopping carts, the cache helps to better scale the org.

How it works
Using Platform Cache, the initial Order and Order Items can be retrieved from the database and copied to the cache, subsequent calls can retrieve the cart and items from the cache, and updates will be written to the cache. Once the user saves the cart, the data in the cache can be written to the Order and OrderItems objects.

Cache type to use
Session cache is best suited for this use case, as data like the contents of a shopping cart are private to the user's session.

4. Caching metadata

There are types of data in Salesforce that don't change often. Examples of these data are Object metadata as well as user related data, such as profiles or role information. The user profiles and metadata are checked frequently due to the many transactions that happen, usually to see what permissions the user has.

Benefits

Metadata and profile information are suited for caching due to less frequent changes. This improves application performance for the end user as requests don't need to trek all the way up to the database for such small pieces of data.

How it works
If every request needs to check for user information, this type of data can be cached safely and with a longer TTL (Time to Live). The idea would be to get the metadata in an initial call using SOQL or Describe calls, then add the result to Platform cache; then for all subsequent calls, retrieve the data from cache. See below.

```
Profile profile = [Select Id, Name from Profile where Id = :profileId];
            if (profile!=null) {
                Cache.OrgPartition profilePartition = Cache.Org.getPartition('local.ProfileData');
                profilePartition.put(profile.Id,profile.Name,7200);
```

Similarly, object metadata can be cached, as shown here:

```
Map<String, Schema.SObjectType> GlobalMap = Schema.getGlobalDescribe();
        Schema.DescribeSObjectResult obj = GlobalMap.get(objectName).getDescribe();
        if (obj != null) {
            Cache.OrgPartition describePartition = Cache.Org.getPartition('local.DescribeData');
            describePartition.put(objectName,JSON.serialize(obj),7200);
```

Cache type to use

An org cache or a session cache can be used for this use case. If metadata is common to multiple users, then a generic key can be used and stored in org cache. Alternatively, a user session can retrieve metadata in an initial call and subsequent calls can refer to the data in session cache.

Takeaways

Here is what we can summarize

- Caching application data is highly recommended for performance and scalability as cache optimizes data access.
- Due to the nature of data in Salesforce, caching for short durations can benefit user experience and org scalability.
- Platform cache makes it easy to implement **caching in your applications**, as demonstrated above.