LIGHTNING DATA SERVICE

Definition

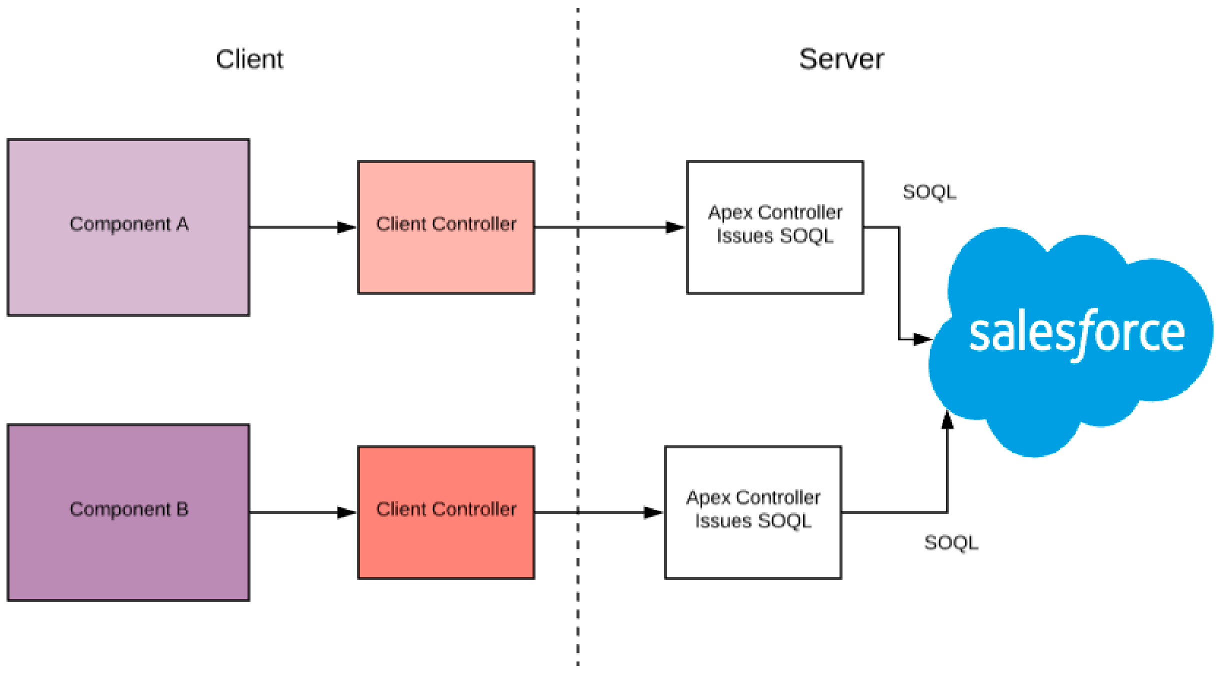
For applications in Lightning, LDS creates reads, updates or deletes (CRUD) records.

Earlier if we have to create, or, read, or update or delete a particular record, we would have had to call the server for help. A lot of time is lost and this whole operation involves page loads, callbacks, loading etc. However, with the LDS, you can avoid these. You will not be required to call the server for each operation being performed on the same data record.

Lightning Data Service allows to load, create, edit, or delete a record in our component without requiring Apex code as well as handles sharing rules and field-level security for us. Additionally, Lightning Data Service improves performance and user interface consistency.

If we have multiple custom components on a page with its own Apex controller, we can easily repeat SOQL queries.

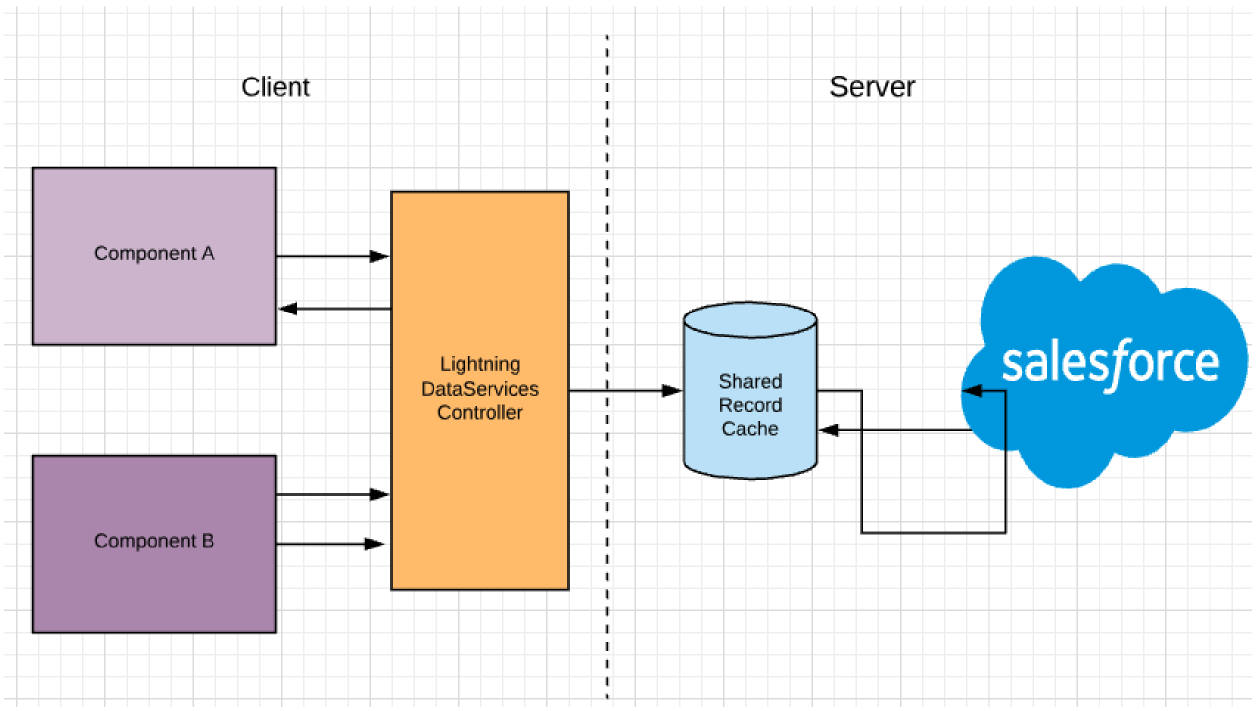
Let's take a look at a scenario wherein multiple components have their own SOQL query, and there is no data sharing between components, as illustrated in the following diagram:



Lightning Data Service provides the following advantages:

- Introduces the sharing of data via a common cache implementation.
- Avoids the need to write Apex, and also takes care of CRUD/FLS and DML security settings configured by the administrator.

The following diagram shows how the data flow is simplified with Lightning Data Service, with the ability to cache and share the data:

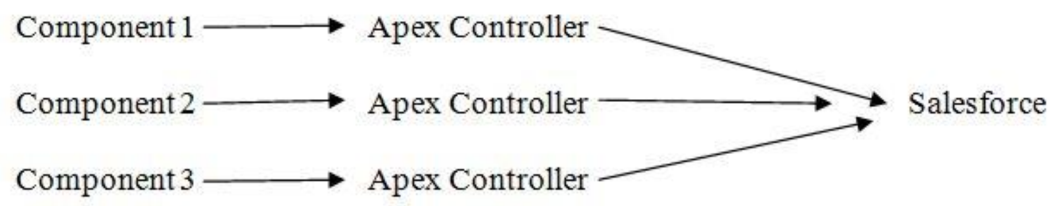


Advantages

1. Supports CRUD operations
2. No need to write Apex
3. No need to write Test classes
4. No need to fire SOQL queries explicitly
5. No need to handle CRUD/FLS permissions
6. Field level security and record sharing is inbuilt
    a. No need to handle sharing rules with, without or inherited sharing rules
    b. LDS features automatic caching and sharing of records that you can load in the LDS itself.
7. No need to handle frontend validations
8. No need to handle caching explicitly. Shared cache is used by all standard and custom components. With the LDS a consolidate request is made and stored in a highly efficient cache, the caching is shared across components on the users' browser, so server requests are not made on cache hits. LDS further optimizes the cache, by only storing the fields that have been requested
9. No need to worry about SOQL injections
10. Automatic record refresh from server. We can keep UI updated with latest data without refreshing the page This retains UI consistency and eliminates the need for a programmatic refresh.
11. Data consistency across components. No need to worry about the stale data in all the components in the record page
    a. Refresh the components with latest data in all components at a time
    b. Even if we have multiple components using the Lightning Data Service, they are all going to use a single request and deal with a cached response and in terms of performance this is huge.
    c. Lightning Data Service and the standard Salesforce UI share an integrated cache, so that if one is updated the others also reflect that change. This means that if our custom component makes a change to the record in question, the standard UI does not continue to display stale date.

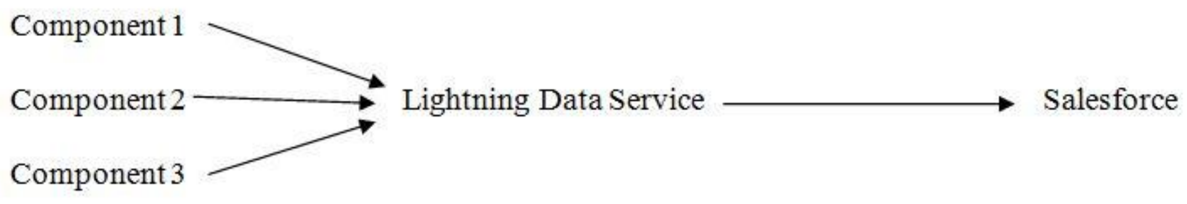
12. We can avoid JSON serialize/deserialize process to pass the data from UI to Server.
    a. It is very costly process
13. Data available offline. With the Lightning Data Service, you also get other built-in features like auto-notifications when there are record changes, as well as offline access for Salesforce1.
    If a user gets disconnected from the network, it allows him to work in offline with data. Data status gets updated once the network connection gets restored.
14. When a component updates the viewed record, all the other components using that record on the page are notified, and in most cases refreshed automatically. Additionally, it also supports the parent-child relationship between SObjects.
15. Performance Benefits:
    a. Data access using the LDS is very simple when compared to using a server-side Apex controller.
    b. Highly efficient local storage acts as the foundation for LDS. Furthermore, every component using the local storage can share it.
    c. High reduction of complexity in operations. This is because your component consolidates every bit of your data access code.
16. No need to call the server from each component. Automatic notification when record changes.
    Reduce server load, database load, and network transfer by fetching data once.

    Basically, every web component needs server call to show data. Now if we are taking the traditional approach suppose we have 3 components in a page, and all are showing data about Account.



    Now we can see in a single page for the same data there is three apex call, and the database will be accessed three times. If data changes by component 1 it will reflect new data while others will have no information about this so they will keep showing the old date. Definitely we don't want this. So, after knowing all this issue we will be happy if we can get a middleman who can fetch data from Salesforce, cached it at his end, and send the data to all components. Also, if it can notify all components for any change in data.

    So here comes LDS for rescue.



Limitations

1. Cannot be used to perform queries using anything other than the recordID.
2. Not all object supports LDS Check for supported standard object
3. Cannot use for VF pages
4. Server-side support is not available
    a. If record gets updated from outside page, like using Dataloader or some other user, it will not be refreshed. Means, it supports client-side caching but not server side caching yet
    b. The Shared data storage of LDS provides notifications to all components that use a record whenever a component changes that record. It does not notify components if that record is changed on the server. For example, if the record is changed by batch class execution, records changed on the server are not updated locally until they are reloaded. In this case CDC, Platform events or Push Topic will help.
5. We can only perform primitive DML operations (create, read, edit and delete) on only one record. Bulk data is not supported
    a. We can perform only primitive DML operations (create, read, edit and delete) on a single record.
6. LDS only supports one operation in one transaction. To perform more operations in a single transaction, we have to use Apex code.
    a. To support multiple operations in a single transaction we need to use @AuraEnable methods.

A scenario when to use LDS

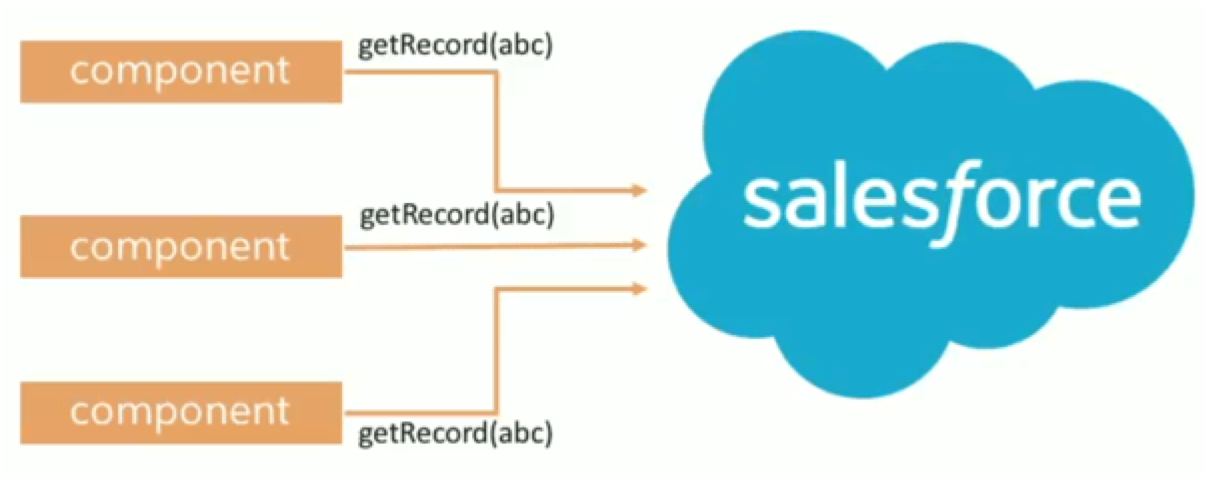
In the case of displaying and updating a particular data record in Lightning, you need to create two components. One component is for displaying the record data and the other is for updating it.
Since there are two components seeking to work on the same record data, both the components will call the server.

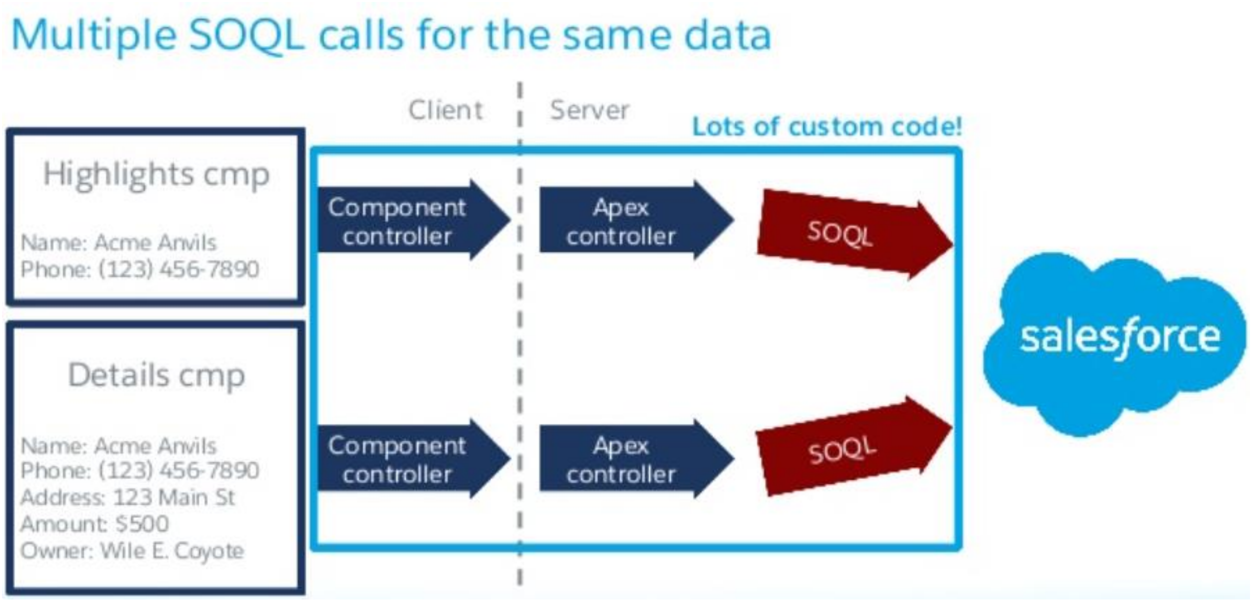
Due to this, there will be a **duplicate** call to the server for the same record. You can avoid this completely when using the LDS as the standard controller.

Problem 1: Data Reuse across different component.

Let us consider a scenario where we have 3 components and each component is making an independent call to the server to perform the CRUD(Create, Read ,Update and Delete) operation on the same set of data.

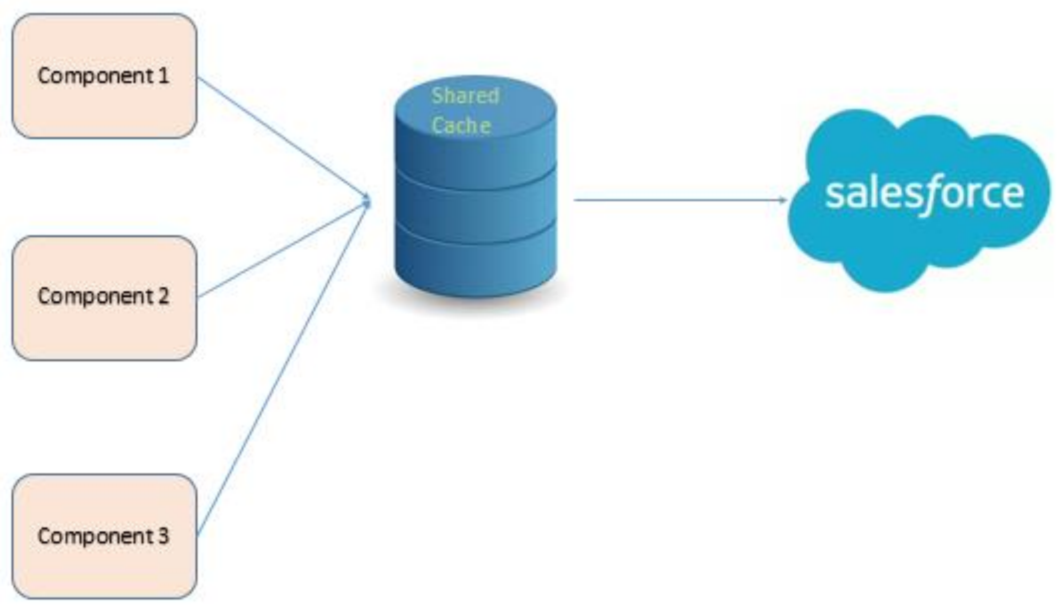
I.e., To achieve this, both Lightning components would invoke Apex method separately at the cost of performance, by issuing duplicate server calls. Below image depicts problem, where multiple Lightning component requests same content by making separate server calls.



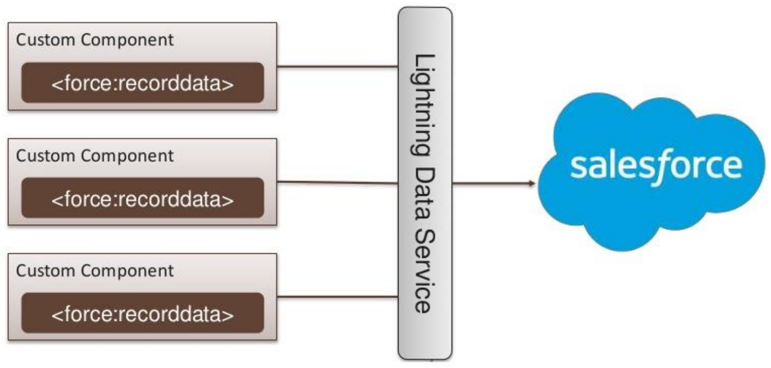


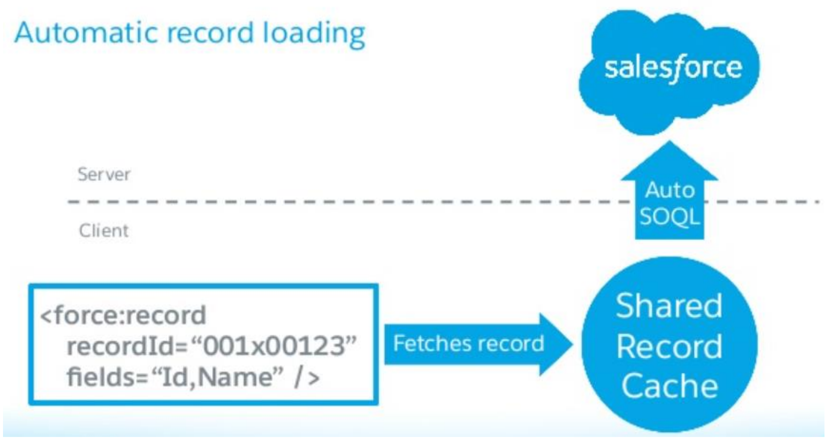
In such scenario, you would have Apex code being called separately for each component creating overhead for the system.

Solution

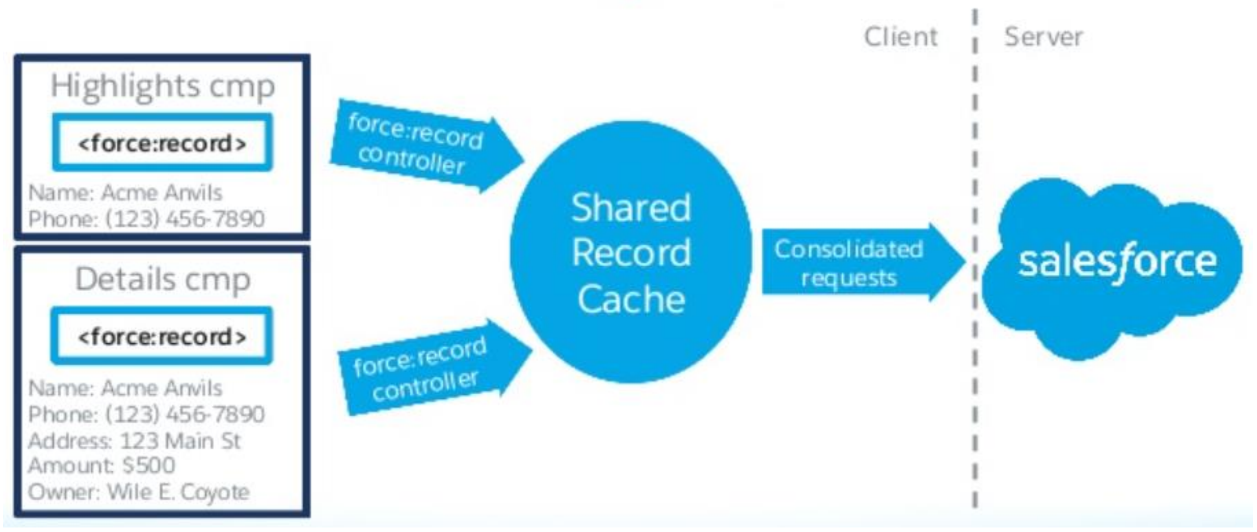


In Aura Component

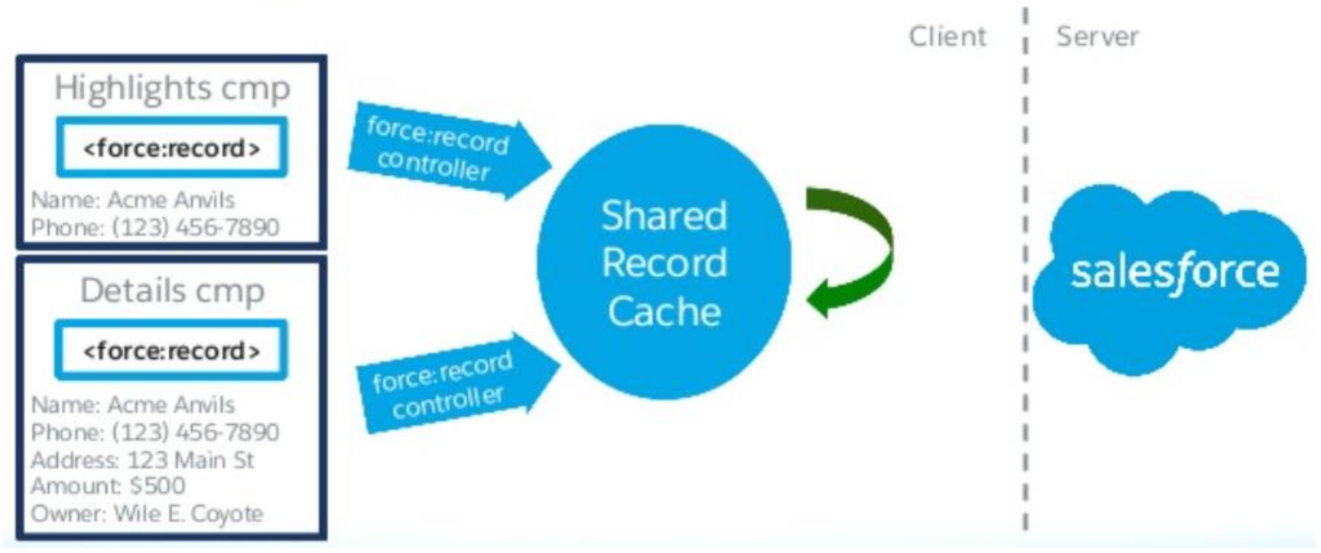




<force:record> handles loading records

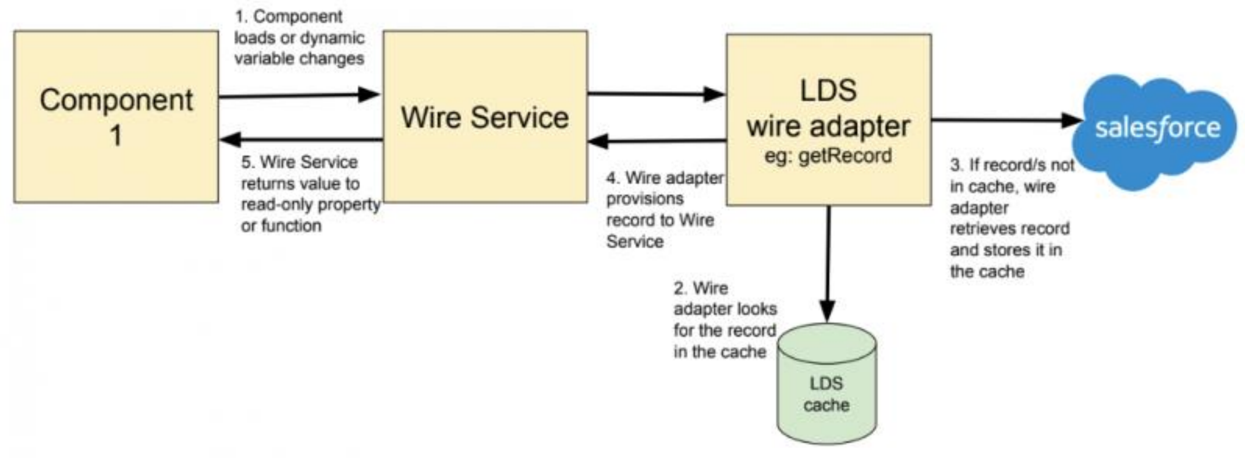


No server trip on cache hits!

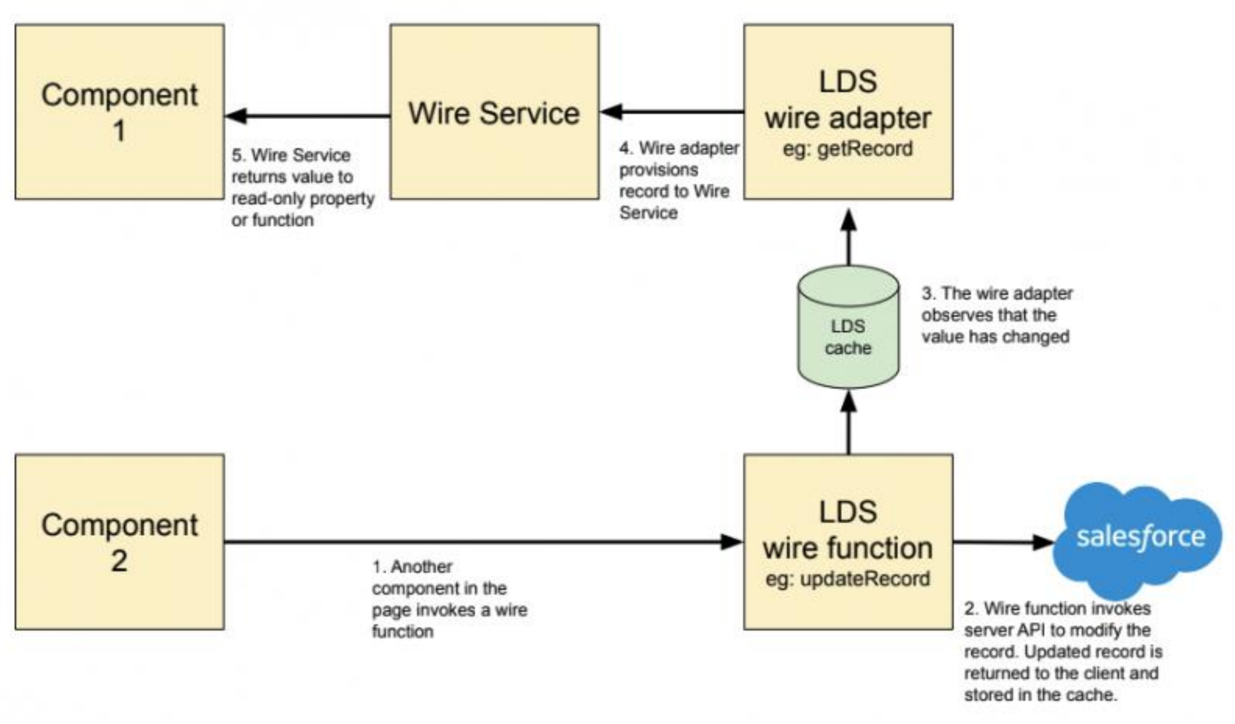


In LWC

Case-1: Single LWC Component


Component 1
1. Component loads or dynamic variable changes
Wire Service
LDS wire adapter eg: getRecord
salesforce
5. Wire Service returns value to read-only property or function
4. Wire adapter provisions record to Wire Service
3. If record/s not in cache, wire adapter retrieves record and stores it in the cache
2. Wire adapter looks for the record in the cache
LDS cache

Case-2: Multiple LWC Components (one component data reflect in another component)

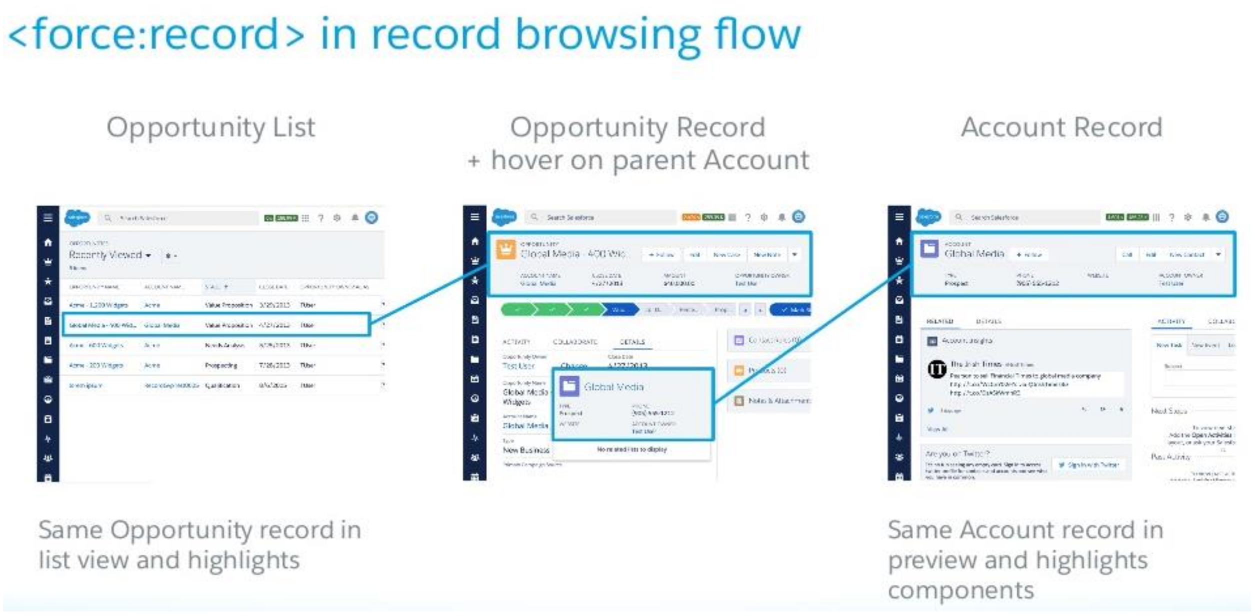

Component 1
Wire Service
LDS wire adapter eg: getRecord
5. Wire Service returns value to read-only property or function
4. Wire adapter provisions record to Wire Service
LDS cache
3. The wire adapter observes that the value has changed
Component 2
LDS wire function eg: updateRecord
salesforce
1. Another component in the page invokes a wire function
2. Wire function invokes server API to modify the record. Updated record is returned to the client and stored in the cache.

Problem 2: Data Inconsistency

It is possible that the field is updated but because of some issue it is not reflected everywhere.

Problem 3: UI inconsistency

If data gets changed because of any component, Other component needs to be informed with the help of Lightning Events. Again, it would cost performance, as every Lightning component may listen that event.



<force:record> in record browsing flow

Opportunity List

Opportunity Record
+ hover on parent Account

Account Record

Same Opportunity record in list view and highlights

Same Account record in preview and highlights components

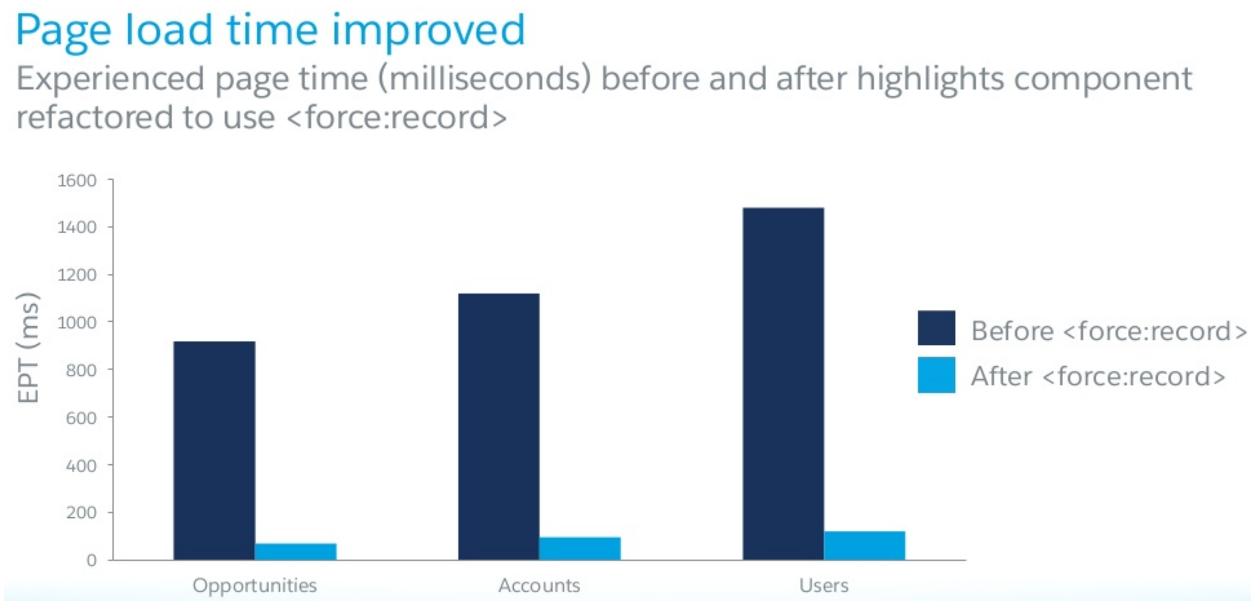
Problem 4 : Server-side code for basic data handling

Developer needs to write Apex code and handler many scenarios like SOQL, Field level security, writing test classes and many more.

Problem-5: Performance

1. Reduces the page loading time from seconds to milliseconds.

2. Reduces the app server load and database server load.



How it works in real time?

If you are using **multiple components** together and they work on the **same record**, the process of communication between them can be exhausting.

The communication process generally contains these steps:

1. Use Server-Side Apex call to fetch a dataset
2. Perform DML operations using Apex
3. Notify other components using Lightning Events
4. Refresh other components manually
5. Check for FLS (Field Level Security) and CRUD (Create Access Update and Delete) security using code
6. Write unit tests

Through **Lightning Data Service (LDS)**, Salesforce provides us an easy way to avoid these exhausting steps:

1. **No separate Server-Side Apex calls**
   LDS analyzes all the server-side requests of all components which are related to the same record and sends a single shared request which **improves the performance.**
2. **Perform DML operations without writing Apex code**
   It provides **CRUD** functionality

   a. **getNewRecord()** – Allows to **Read** record dataset
   b. **getNewRecord()** – **Creates** new record
   c. **saveRecord(callback_function)** – **Saves** record after updating it
   d. **deleteRecord(callback_function)** – **Deletes** record dataset

3. **Auto Notification**
   Changes in any component automatically get notified to other components which help to get rid of the need to use events to notify each other.

4. **Refresh Components and Cache automatically**
   reloadRecord() refreshes other components in container. After saving changes in a record, LDS automatically refreshes the cache too

5. **Standard Controller**
   Like Visualforce Standard Controller, LDS takes care of **CRUD operations** with **FLS (Field Level Security)** on record data.

6. **Cache to work offline**
   If the user gets disconnected and is working offline, it **automatically syncs** the data as soon as the connection is up.
7. **Consistent Data**
   If multiple components are working on the same data, LDS uses a **single instance** to make sure the data is consistent.

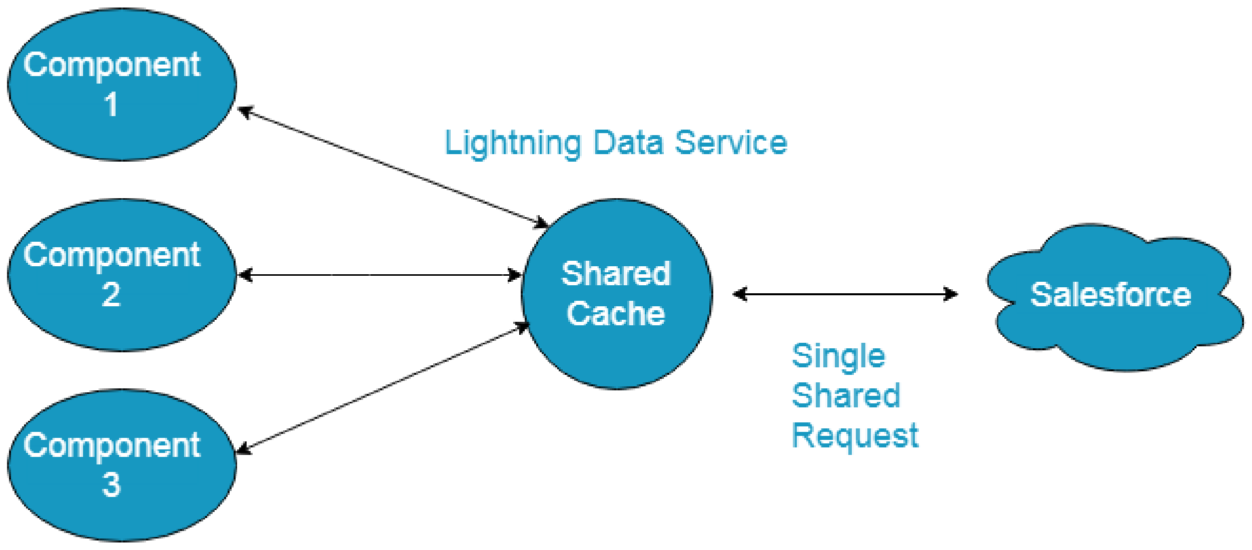


Fig. LDS removes Duplicate calls and uses Single Shared Request