1st: Need to Avoid Unnecessary Server Round-Trips

Try as much as possible away from unnecessary processing from the database and application server always results much better performance and ultimately a better user experience.
Three initiative we can take to achieve this

- ***Database operations***
- ***Filtering in Browser.***
- ***Caching.***

Database operations :

Numerous requests for data from the browser decrease the efficiency of performance. So it always good to execute one logical query, enhance with additional information as needed and return a list to the browser to display and handle any further actions or interactions.
On a note "Instead of multiple calls to the database for each child record, its best to combined them into one call."
For example format: One logical query that joins the child objects to retrieve collection of data.

> **SELECT Account.Name, Account.Type,(SELECT Contact.Name, Contact.Email FROM contacts),(SELECT Case.ClosedDate FROM cases),(SELECT <Obj>.<Field> FROM <Relationship_Name>) FROM Account**

May it happen Some Key fields that user frequently use in a page, in that scenario we need to retrieve the updated value of those fields each time from database. So issuing a server call again and again to retrieve the updated information from the database would be slow in performance perspective.

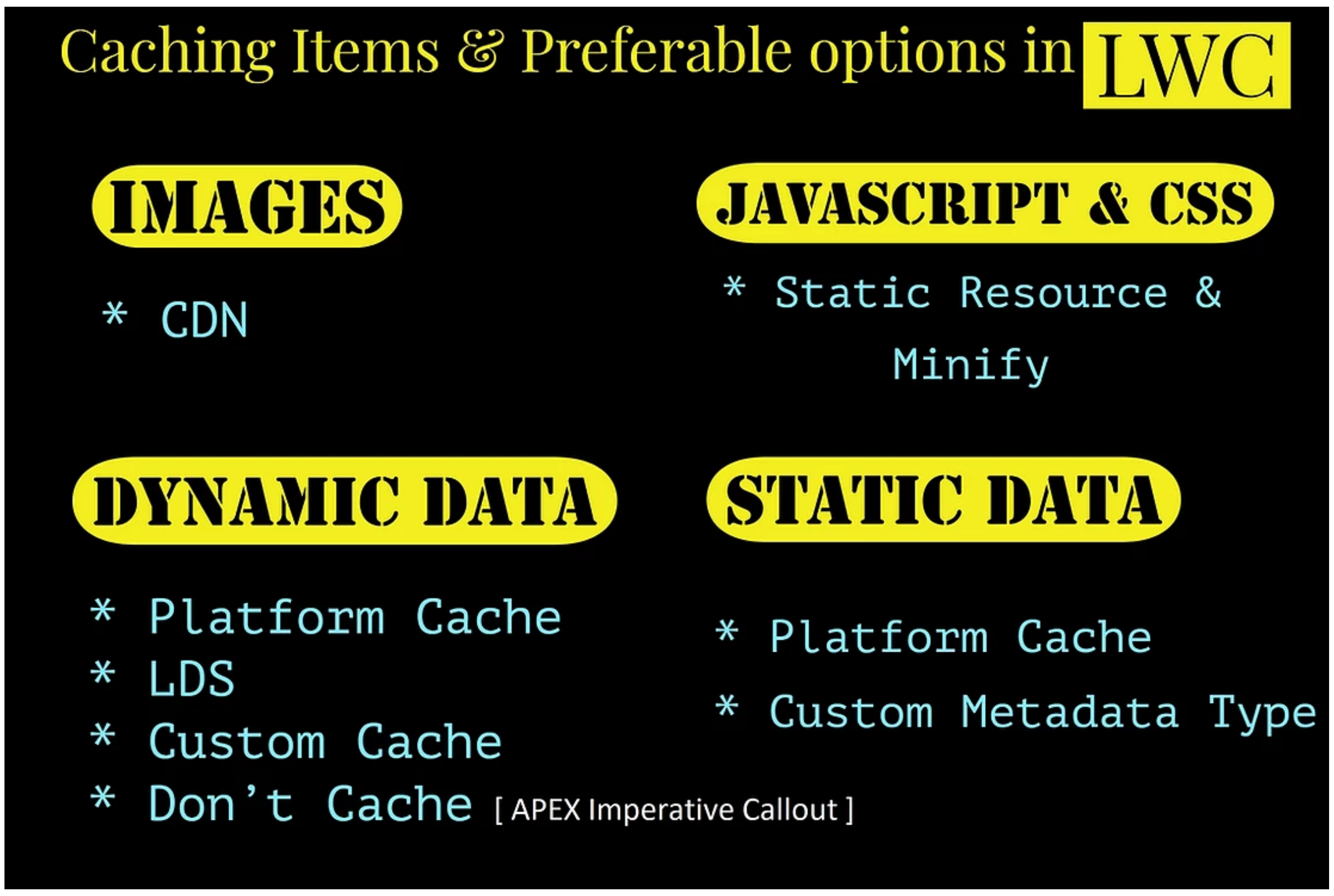
No Worries, LWC have the option of Lightning Data Service (LDS), service provides a cache shared amongst all the components that use the service, and it is smart enough to notify those components about changes of the records performed through the service so that the components synchronize. It works both for Lightning Web Components (LWC) and Aura component.

Filtering in Browser :

To avoid unnecessary server round-trip lets query for list of all data and then to get the specific require information from the list, each time we ensure filtering is done in client side code. In this way data retrieval through filter will be applied almost promptly.

Caching :

Lwc Component uses images, JavaScript libraries, CSS and database records. Where possible these need to be cached to reduce load and render times. Let's consider some Caching techniques that salesforce provide.


Caching Items & Preferable options in LWC

IMAGES

* CDN

JAVASCRIPT & CSS

* Static Resource & Minify

DYNAMIC DATA

* Platform Cache
* LDS
* Custom Cache
* Don't Cache [APEX Imperative Callout]

STATIC DATA

* Platform Cache
* Custom Metadata Type

Img1: Some Caching Technics in Salesforce.

2nd: Consider Browsers have limits

Avoid Large Complex Layouts
Avoid Layout Thrashing
Avoid forced synchronous layouts

Avoid Large Complex Layouts :

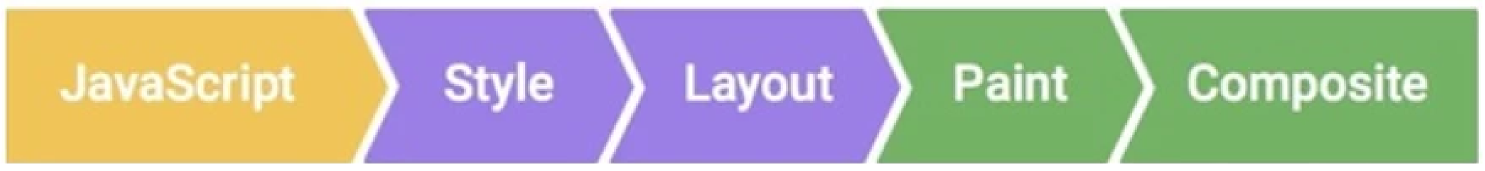
Reduce the number of elements as much as possible and make a compact UI. Combined records to display in layouts. Eliminate unnecessary elements & CSS from UI.

Avoid Layout Thrashing :

When any change in style then web browser has to reflow or repaint a web page many times before the page is 'loaded'. Its effects the performance of the page.

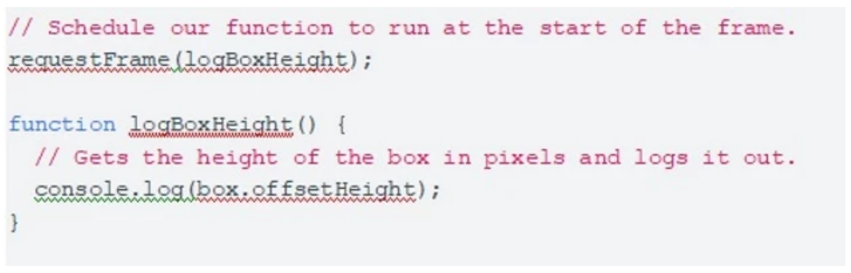
Avoid forced synchronous layouts :

In web JavaScript runs first, *then* style calculations, *then* layout. It is, however, possible to force a browser to perform layout earlier with JavaScript. This scenario is called forced synchronous layout.



Img 2 : Web UI execution

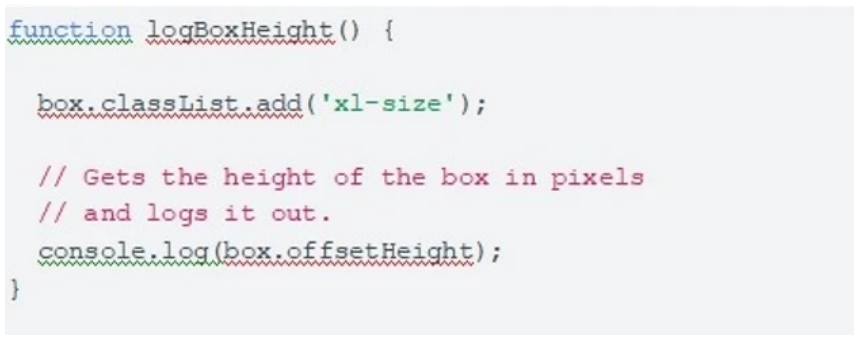
The first thing to keep in mind is that as the JavaScript runs all the old layout values from the previous frame are known and available for you to query. So if, for example, you want to write out the height of an element (let’s call it “box”) at the start of the frame you may write some code like this:



```
// Schedule our function to run at the start of the frame.
requestFrame(logBoxHeight);

function logBoxHeight() {
  // Gets the height of the box in pixels and logs it out.
  console.log(box.offsetHeight);
}
```

Things get problematic if you’ve changed the styles of the box *before* you ask for its height:

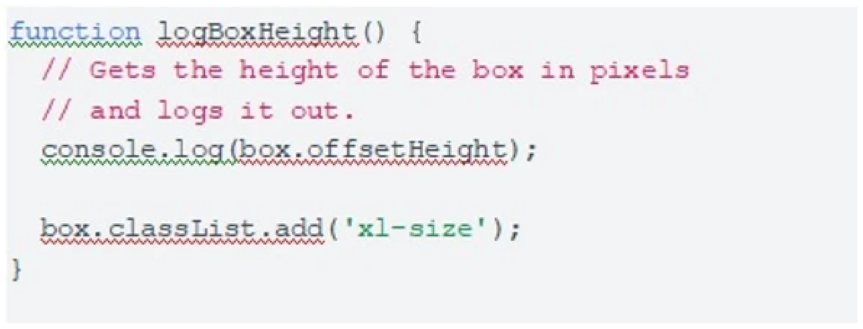


```
function logBoxHeight() {

  box.classList.add('xl-size');

  // Gets the height of the box in pixels
  // and logs it out.
  console.log(box.offsetHeight);
}
```

Now, in order to answer the height question, the browser must *first* apply the style change (because of adding the xl-size class), and *then* run layout. Only then will it be able to return the correct height. This is unnecessary and potentially expensive work. Because of this you should always batch your style reads and do them first (where the browser can use the previous frame’s layout values) and then do any writes:
Done correctly the above function would be:



```
function logBoxHeight() {
  // Gets the height of the box in pixels
  // and logs it out.
  console.log(box.offsetHeight);

  box.classList.add('xl-size');
}
```

For the most part you shouldn’t need to apply styles and then query values; using the last frame’s values should be sufficient. Running the style calculations and layout synchronously and earlier than the browser would like are potential bottlenecks, and not something you will typically want to do.

3rd: Salesforce has limits

As per Multitenant concept, resources allocated for the Salesforce Lightning Platform are shared between different customers. In order to make sure each customer is maintain their governor limits using their fair share of resources. Typically these are substantial in nature but you may hit these limits if you don’t research your approach before execute.

4th: Avoid blank screens

Never implement any blank screen for any feature. For example. To inform user that any component is loading, use styles suggested by LDS to indicate any component is loading.