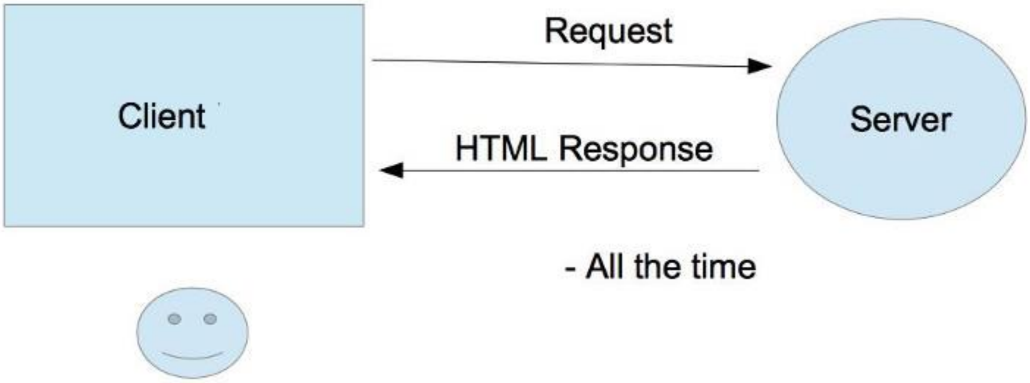
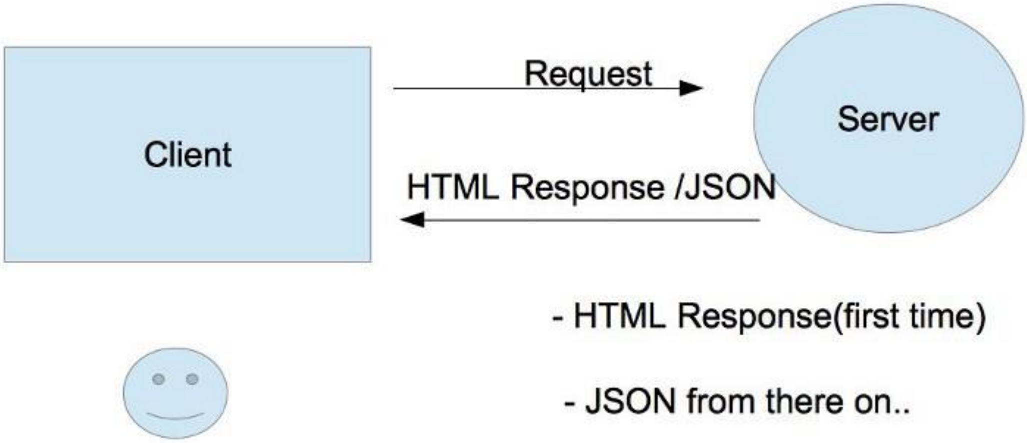
Old Way



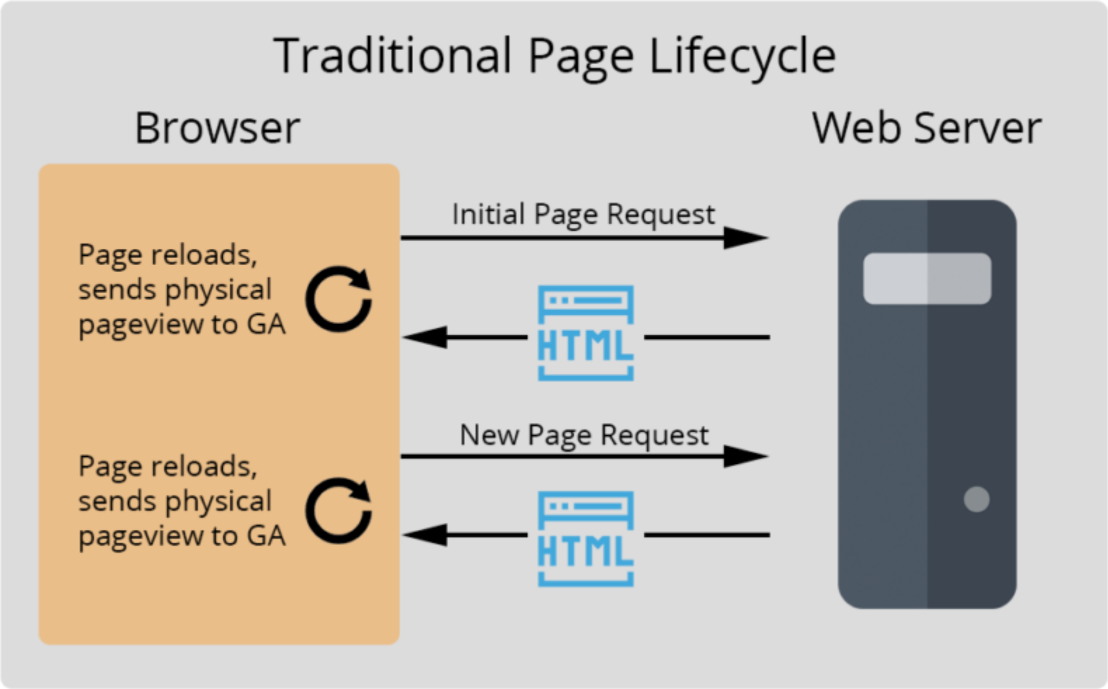
- All the time

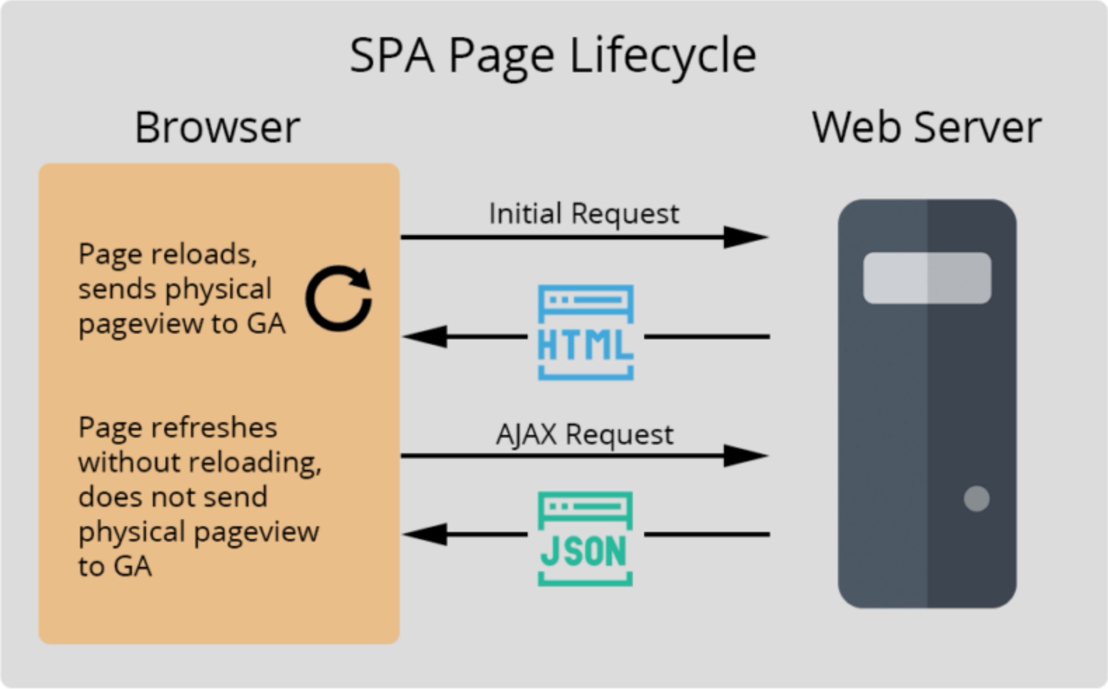
New Way

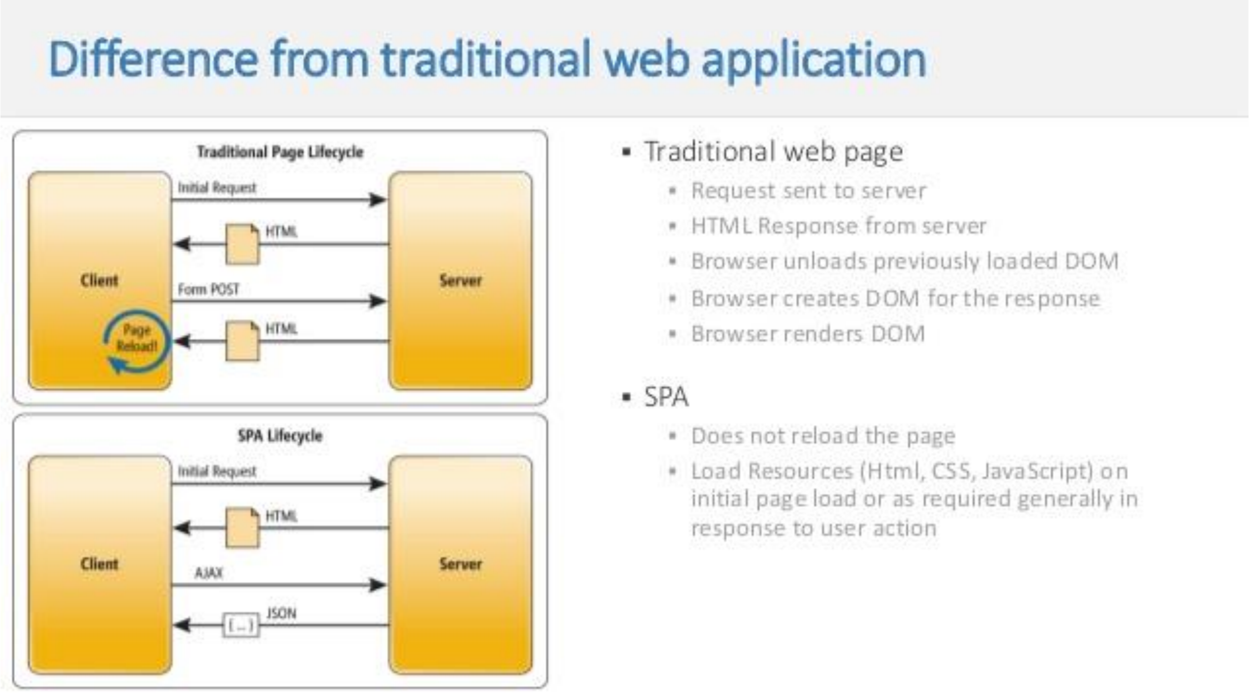


- HTML Response(first time)

- JSON from there on..


Traditional Page Lifecycle
Browser
Web Server
Initial Page Request
Page reloads, sends physical pageview to GA
HTML
New Page Request
Page reloads, sends physical pageview to GA
HTML


SPA Page Lifecycle
Browser
Web Server
Initial Request
Page reloads, sends physical pageview to GA
HTML
AJAX Request
Page refreshes without reloading, does not send physical pageview to GA
JSON

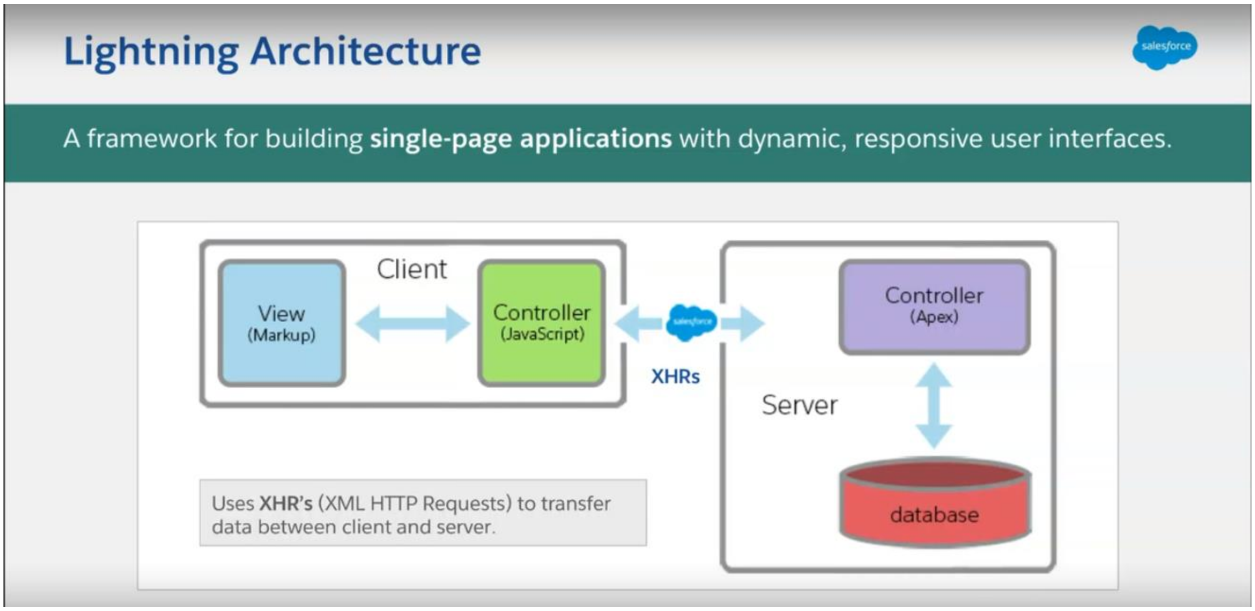


Difference from traditional web application

- Traditional web page
  - Request sent to server
  - HTML Response from server
  - Browser unloads previously loaded DOM
  - Browser creates DOM for the response
  - Browser renders DOM
- SPA
  - Does not reload the page
  - Load Resources (Html, CSS, JavaScript) on initial page load or as required generally in response to user action

Advantages of SPA

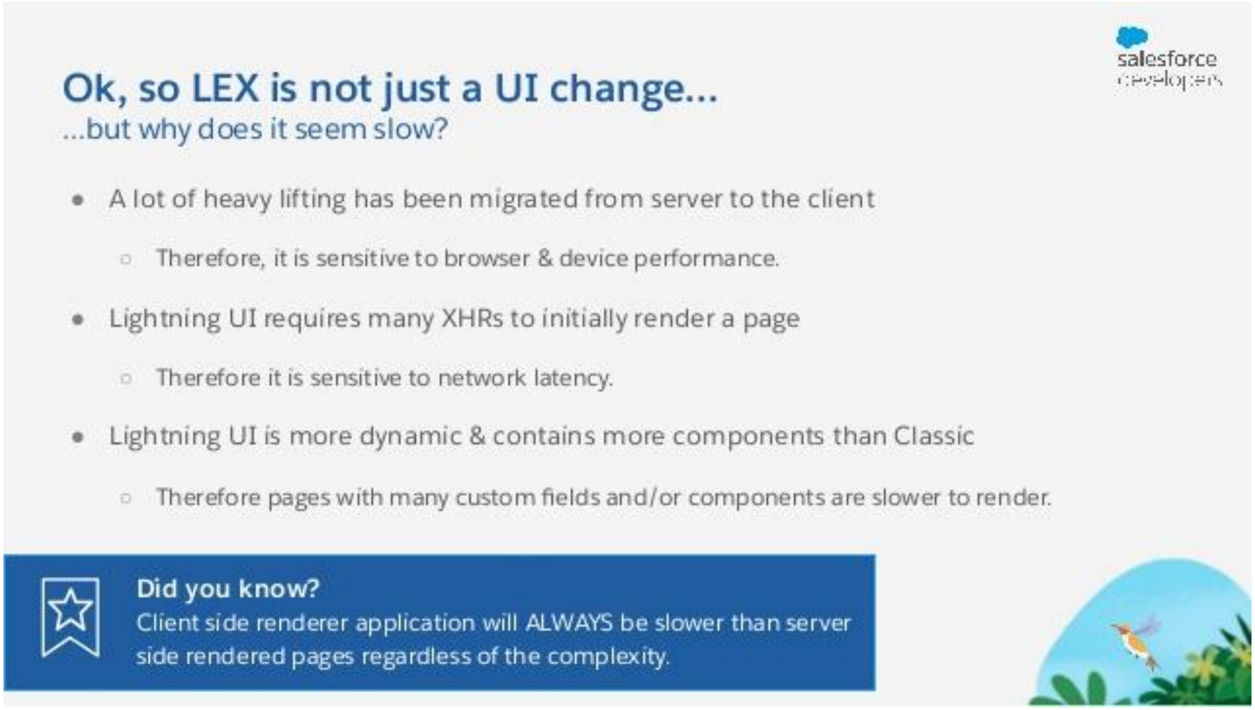
- Maintains application state.
- Help to bookmark and share.
- Keep the data and view in synchronization.
- Partial refresh
- Reduces static resource loading.
- Fast navigation between the application pages.



Lightning Architecture

A framework for building **single-page applications** with dynamic, responsive user interfaces.

Uses **XHR's** (XML HTTP Requests) to transfer data between client and server.

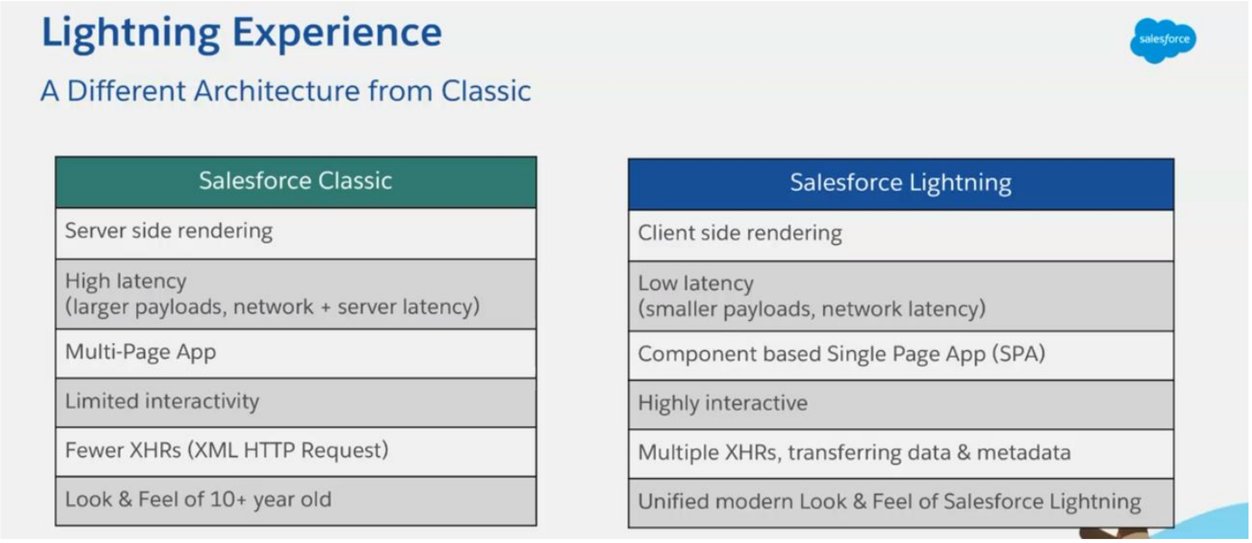


Ok, so LEX is not just a UI change...

...but why does it seem slow?

- A lot of heavy lifting has been migrated from server to the client
  - Therefore, it is sensitive to browser & device performance.
- Lightning UI requires many XHRs to initially render a page
  - Therefore it is sensitive to network latency.
- Lightning UI is more dynamic & contains more components than Classic
  - Therefore pages with many custom fields and/or components are slower to render.

Did you know?
Client side renderer application will ALWAYS be slower than server side rendered pages regardless of the complexity.



Lightning Experience

A Different Architecture from Classic

Salesforce Classic
Server side rendering
High latency (larger payloads, network + server latency)
Multi-Page App
Limited interactivity
Fewer XHRs (XML HTTP Request)
Look & Feel of 10+ year old

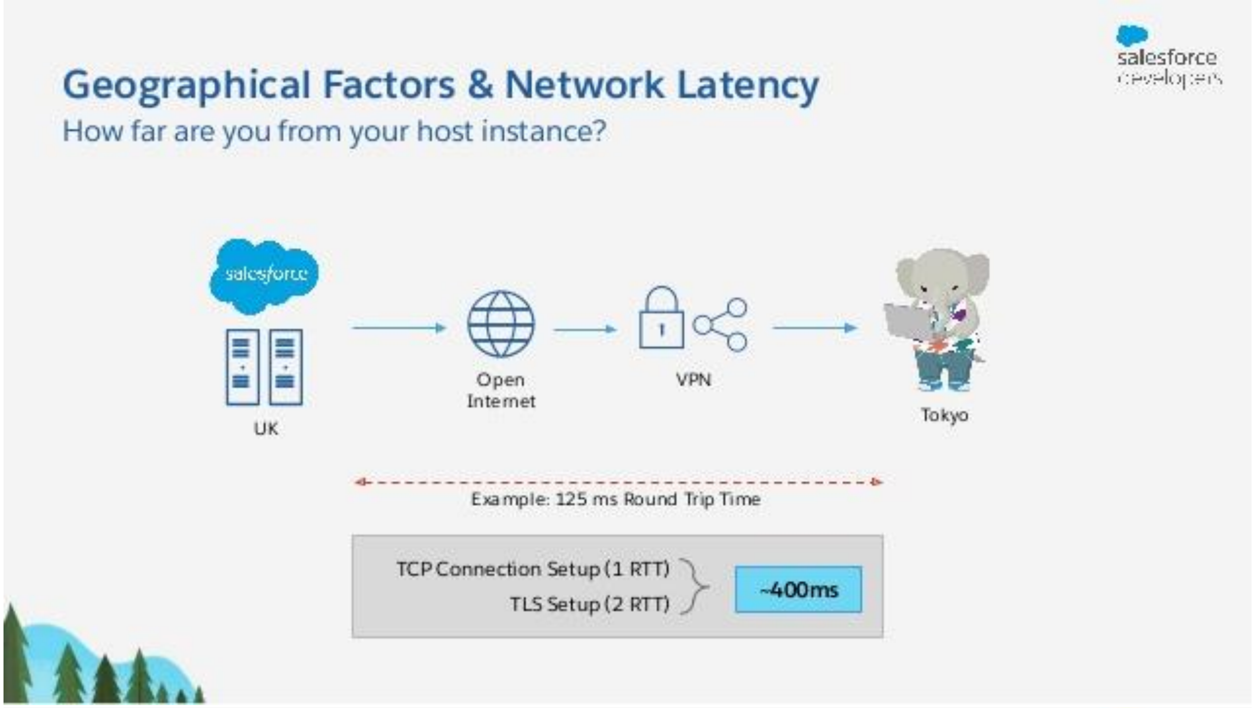
Salesforce Lightning
Client side rendering
Low latency (smaller payloads, network latency)
Component based Single Page App (SPA)
Highly interactive
Multiple XHRs, transferring data & metadata
Unified modern Look & Feel of Salesforce Lightning

How to measure EPT?

- Lightning Experience: Add an EPT counter to Lightning Experience.
  - Enable Debug mode
  - Add eptVisible=1 as a parameter to the Lightning URL.
- Lightning Usage App
  - Use app to view aggregated page and browser performance.
  - Build custom reports using Lightning Usage App objects.
- Event Monitoring: Use event types to monitor performance.
  - Lightning Page View, Lightning Interaction, Lightning Performance

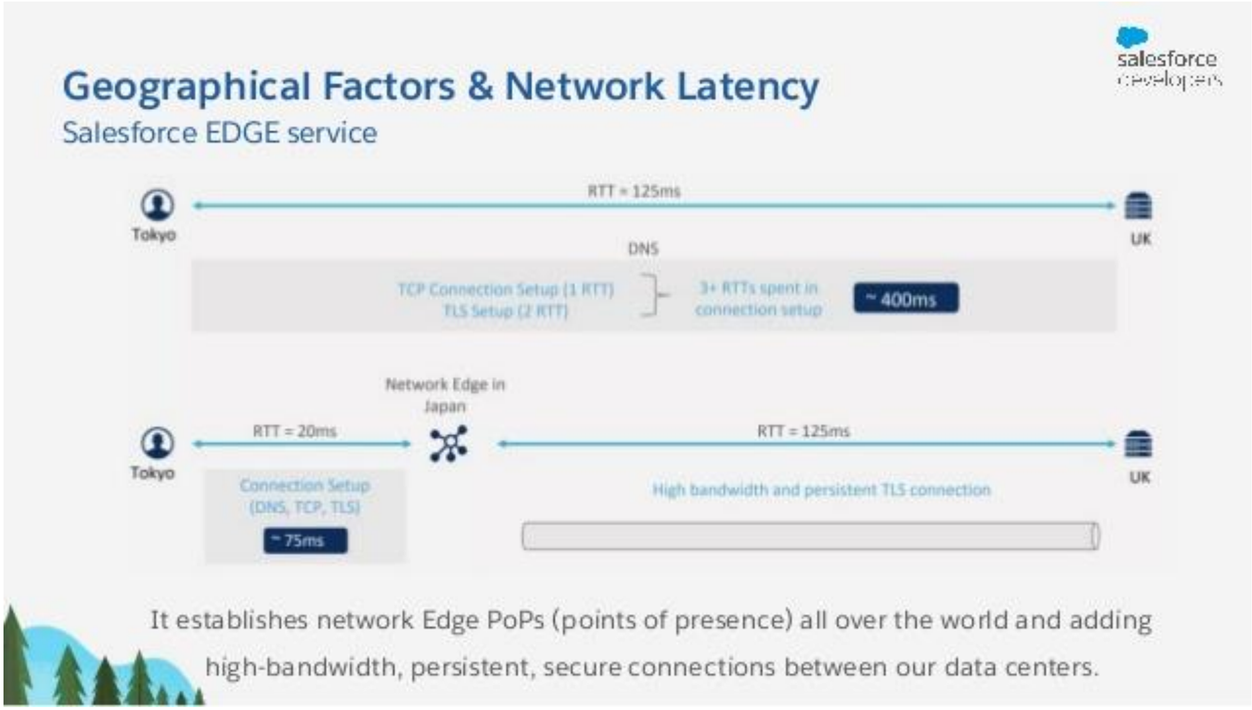
Factors affecting EPT

Geographical Factors and Network Latency	Device and client capabilities	Salesforce Org Configuration
Page Complexity	Component Architecture	Server Processing



Geographical Factors & Network Latency

How far are you from your host instance?

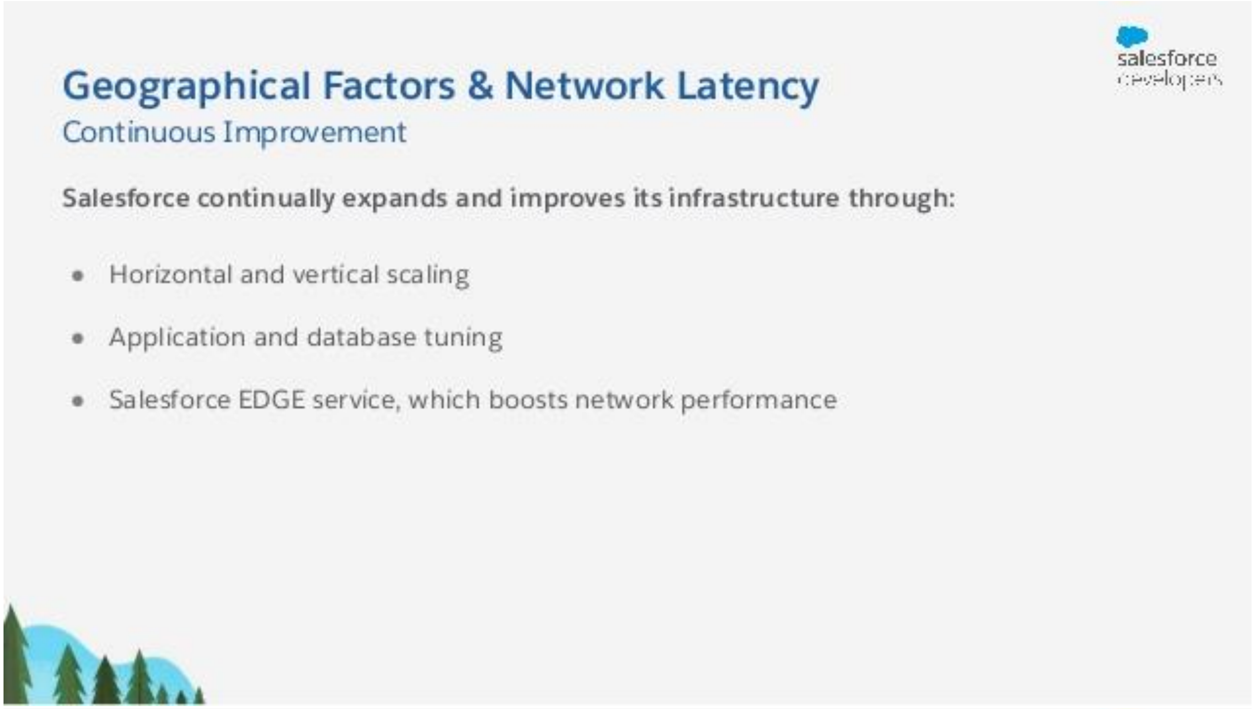


Geographical Factors & Network Latency

Salesforce EDGE service

Tokyo
RTT = 125ms
UK
DNS
TCP Connection Setup (1 RTT)
TLS Setup (2 RTT)
3+ RTTs spent in connection setup
~ 400ms
Network Edge in Japan
Tokyo
RTT = 20ms
RTT = 125ms
UK
Connection Setup (DNS, TCP, TLS)
~ 75ms
High bandwidth and persistent TLS connection

It establishes network Edge PoPs (points of presence) all over the world and adding high-bandwidth, persistent, secure connections between our data centers.

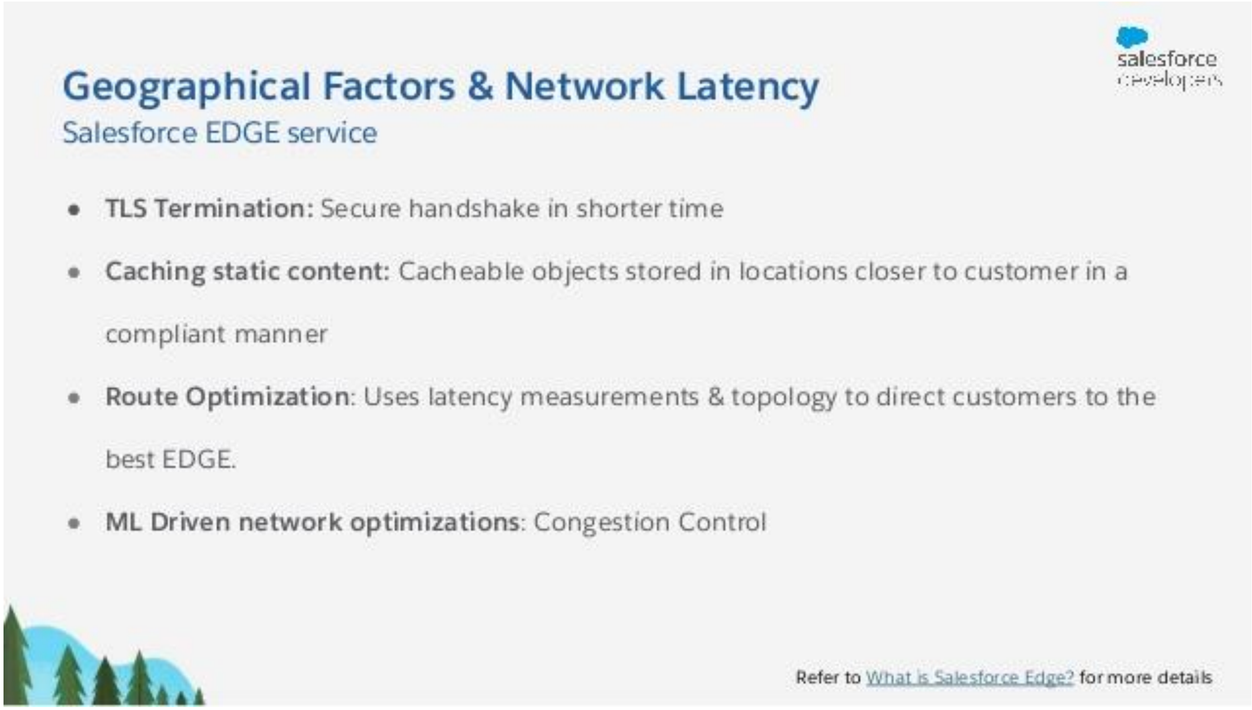


Geographical Factors & Network Latency

Continuous Improvement

Salesforce continually expands and improves its infrastructure through:

- Horizontal and vertical scaling
- Application and database tuning
- Salesforce EDGE service, which boosts network performance



Geographical Factors & Network Latency

Salesforce EDGE service

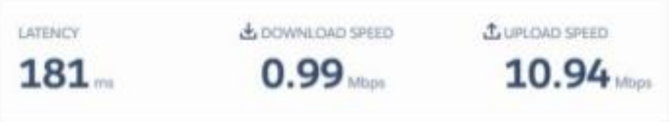
- **TLS Termination:** Secure handshake in shorter time
- **Caching static content:** Cacheable objects stored in locations closer to customer in a compliant manner
- **Route Optimization**: Uses latency measurements & topology to direct customers to the best EDGE.
- **ML Driven network optimizations**: Congestion Control

Refer to What is Salesforce Edge? for more details

Geographical Factors & Network Latency

Measure

- Ask your IT department to run a "ping" or "traceroute".
- Run the Salesforce Performance test - https://yourdomain.lightning.force.com/**speedtest.jsp**



LATENCY	DOWNLOAD SPEED	UPLOAD SPEED
181 ms	0.99 Mbps	10.94 Mbps

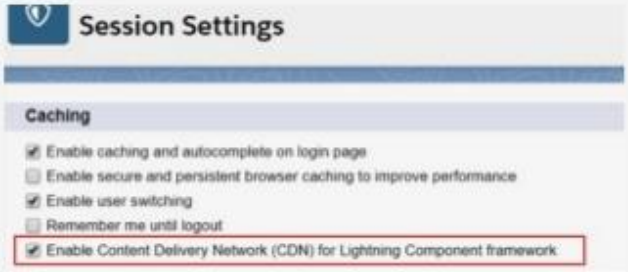
Recommended metrics

- Latency of **150 ms** or lower.
- Download speed of **3 Mbps** or higher.

Geographical Factors & Network Latency

Optimize

- Enable Content Delivery Network for Lightning Component Framework
  - Uses Akamai's CDN for static JavaScript and CSS.
  - Doesn't distribute your org's data or metadata in a CDN.
  - Serves requests for the following from 'static.<instance>.salesforce.com'
- Refactor components to reduce XHRs.



Session Settings

Caching

☑ Enable caching and autocomplete on login page

☐ Enable secure and persistent browser caching to improve performance

☑ Enable user switching

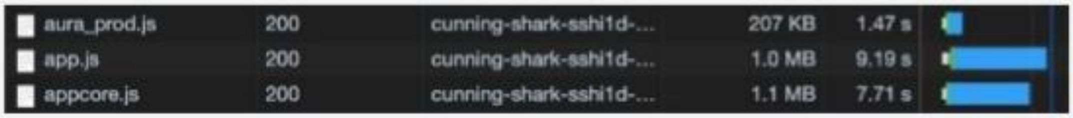
☐ Remember me until logout

☑ Enable Content Delivery Network (CDN) for Lightning Component framework

Geographical Factors & Network Latency

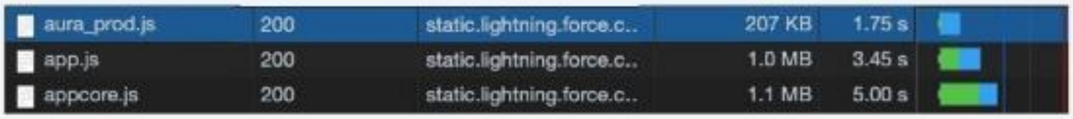
Optimize

Without CDN



aura_prod.js	200	cunning-shark-sshi1d-...	207 KB	1.47 s
app.js	200	cunning-shark-sshi1d-...	1.0 MB	9.19 s
appcore.js	200	cunning-shark-sshi1d-...	1.1 MB	7.71 s

With CDN



aura_prod.js	200	static.lightning.force.c..	207 KB	1.75 s
app.js	200	static.lightning.force.c..	1.0 MB	3.45 s
appcore.js	200	static.lightning.force.c..	1.1 MB	5.00 s

Geographical Factors & Network Latency

Which option is right for me?

Salesforce Edge

My Domain enabled orgs with global users

Lightning CDN

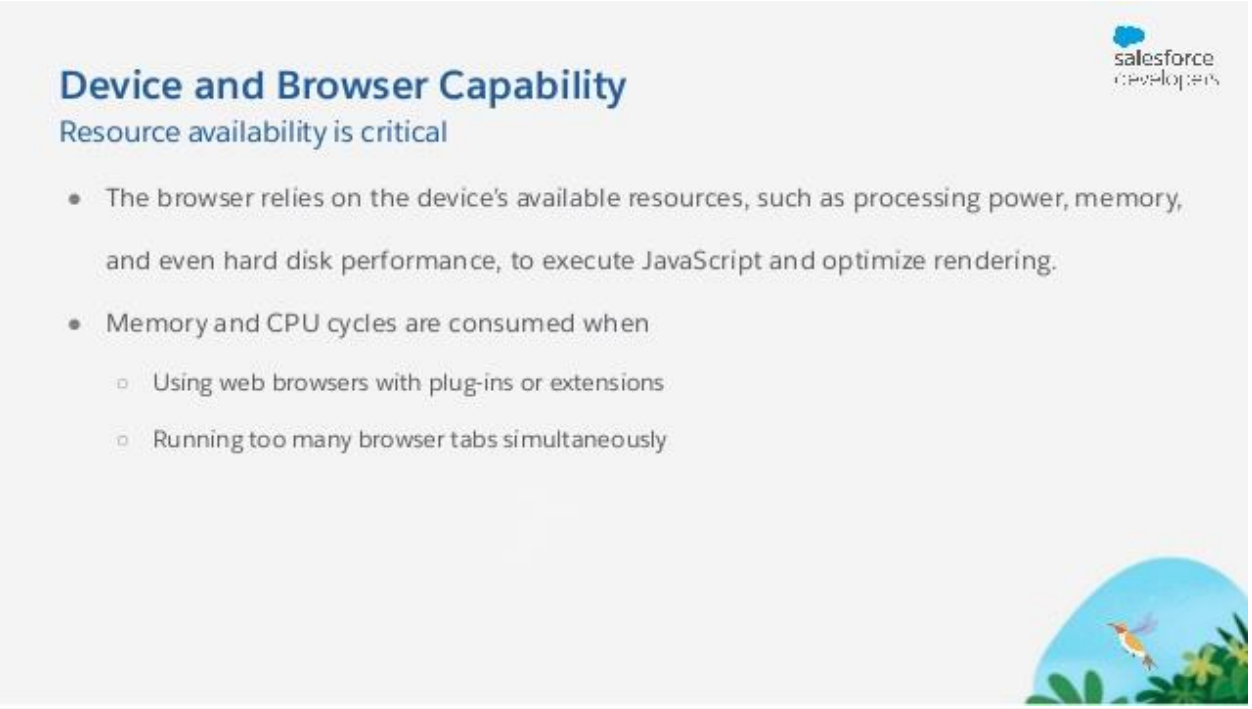
Static Lightning Components

Communities CDN

- For publicly cacheable resources
- With Salesforce CDN partner or any preferred CDN vendor

Express Connect

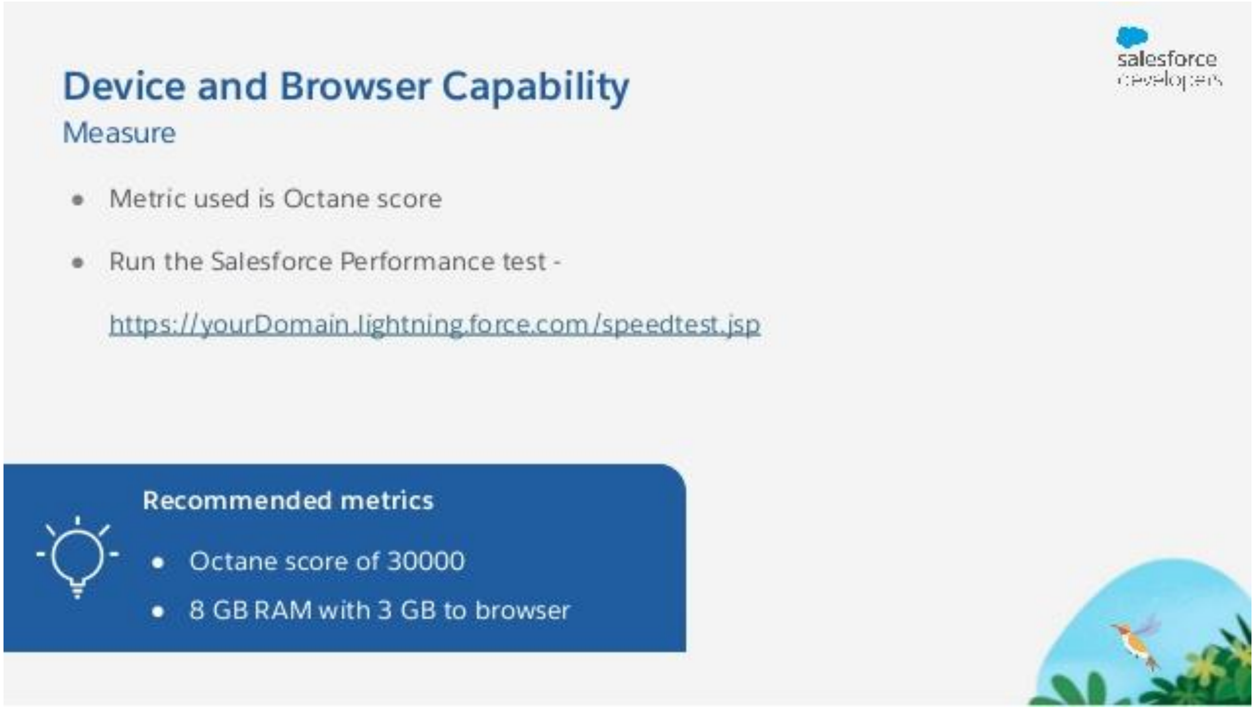
- Customers with limited network egress for security and compliance
- Private connection from your DC to the Salesforce infrastructure



Device and Browser Capability

Resource availability is critical

- The browser relies on the device's available resources, such as processing power, memory, and even hard disk performance, to execute JavaScript and optimize rendering.
- Memory and CPU cycles are consumed when
  - Using web browsers with plug-ins or extensions
  - Running too many browser tabs simultaneously



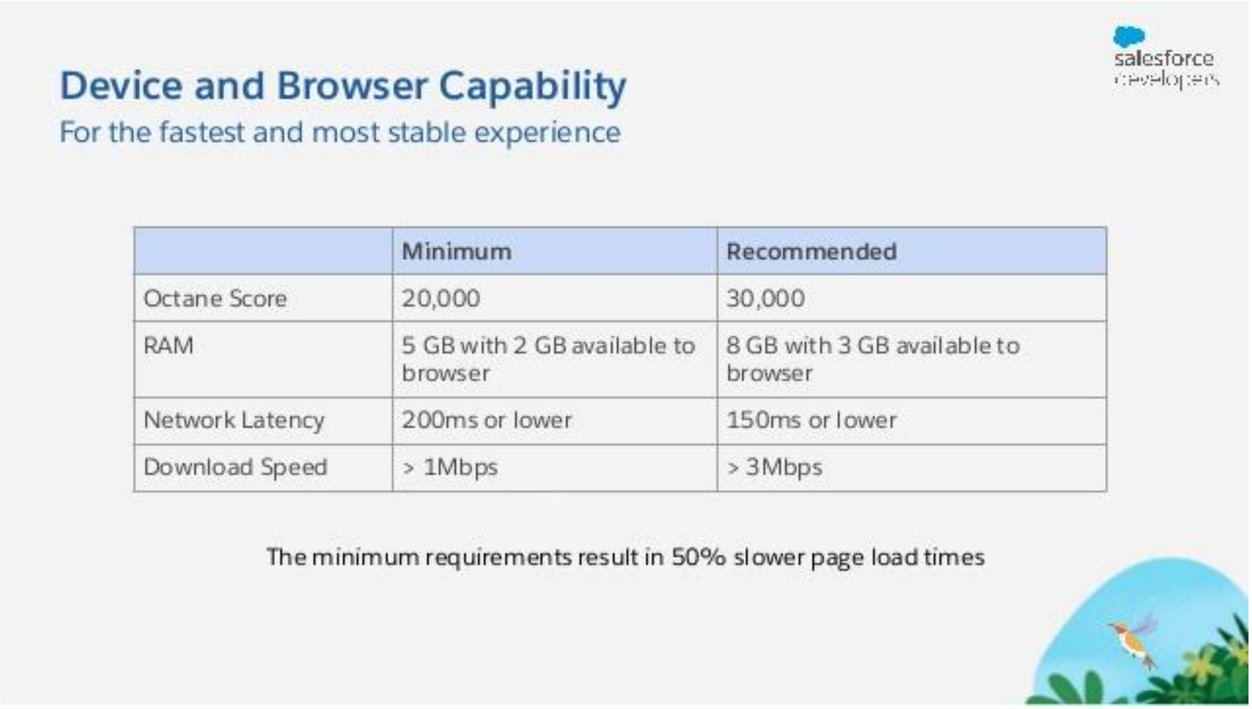
Device and Browser Capability

Measure

- Metric used is Octane score
- Run the Salesforce Performance test - https://yourDomain.lightning.force.com/speedtest.jsp

Recommended metrics

- Octane score of 30000
- 8 GB RAM with 3 GB to browser

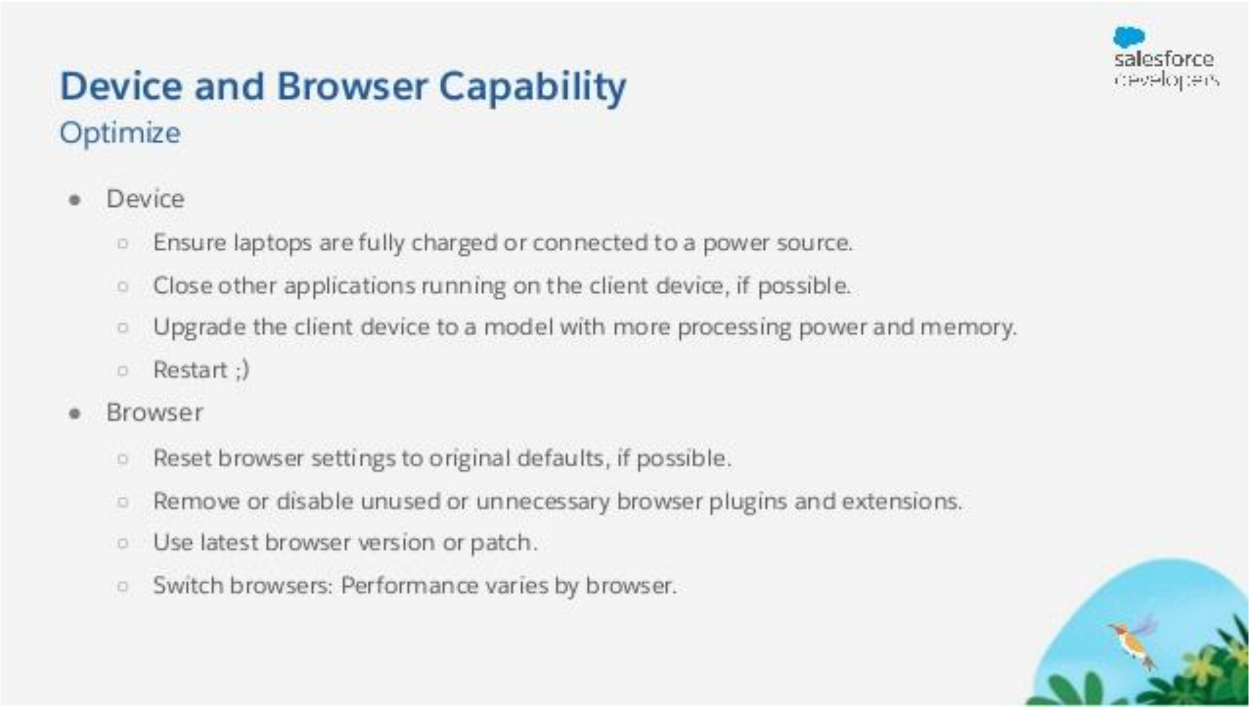


Device and Browser Capability

For the fastest and most stable experience

	Minimum	Recommended
Octane Score	20,000	30,000
RAM	5 GB with 2 GB available to browser	8 GB with 3 GB available to browser
Network Latency	200ms or lower	150ms or lower
Download Speed	> 1Mbps	> 3Mbps

The minimum requirements result in 50% slower page load times



Device and Browser Capability

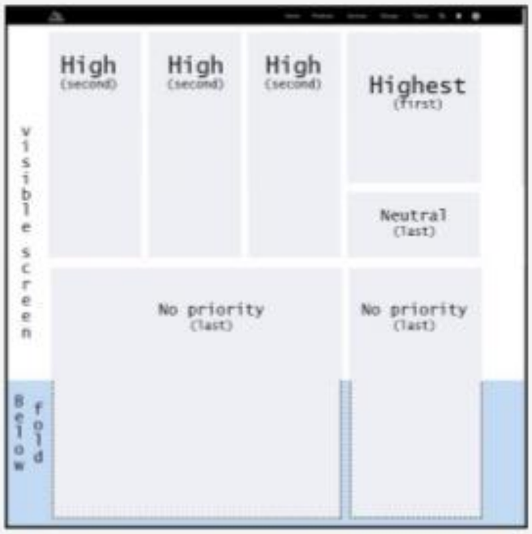
Optimize

- Device
  - Ensure laptops are fully charged or connected to a power source.
  - Close other applications running on the client device, if possible.
  - Upgrade the client device to a model with more processing power and memory.
  - Restart ;)
- Browser
  - Reset browser settings to original defaults, if possible.
  - Remove or disable unused or unnecessary browser plugins and extensions.
  - Use latest browser version or patch.
  - Switch browsers: Performance varies by browser.

Org Configuration

A few settings to remember

- Disable Debug Mode
- Enable Secure and Persistent Browser Caching
- Enable Progressive rendering on communities
- Limit # of Open tabs in Console Apps



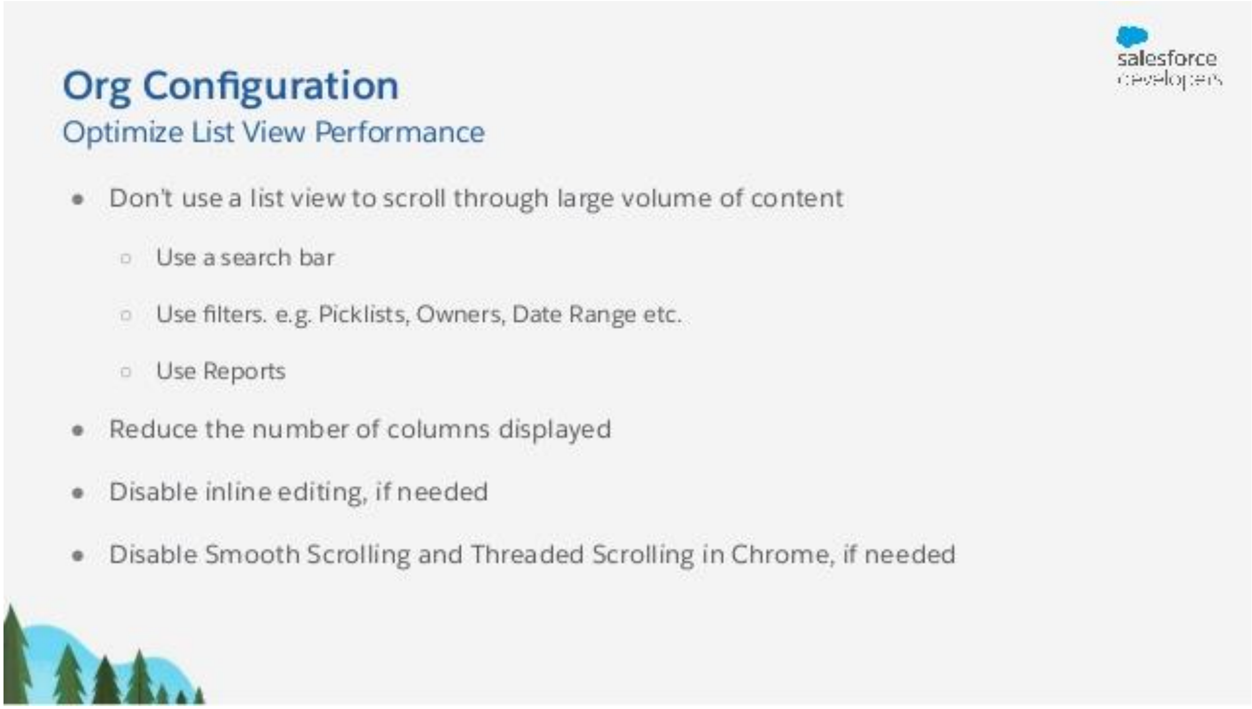
Org Configuration

List View Performance

- Scrolling performance degrades based on the number of rows in the page.

Number of rows and columns ∝ Number of HTML elements ∝ Time spent recalculating styles

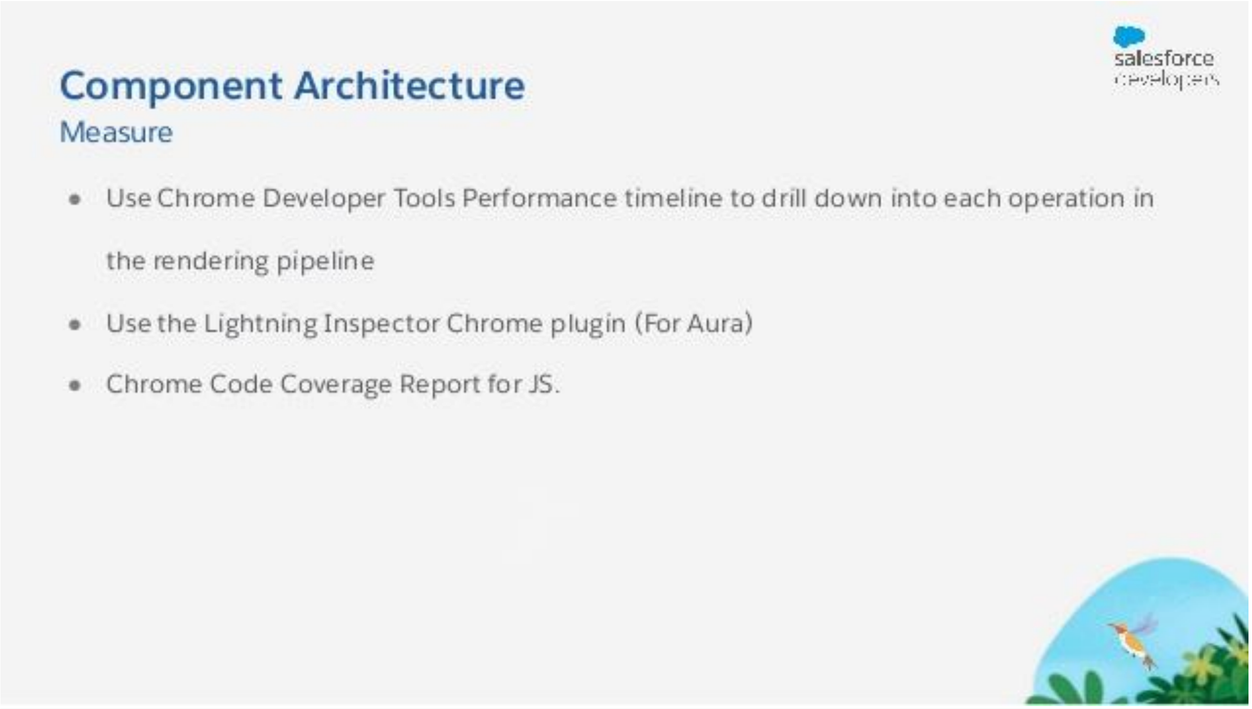
- How HTML elements are drawn and managed is dependent on the browser.



Org Configuration

Optimize List View Performance

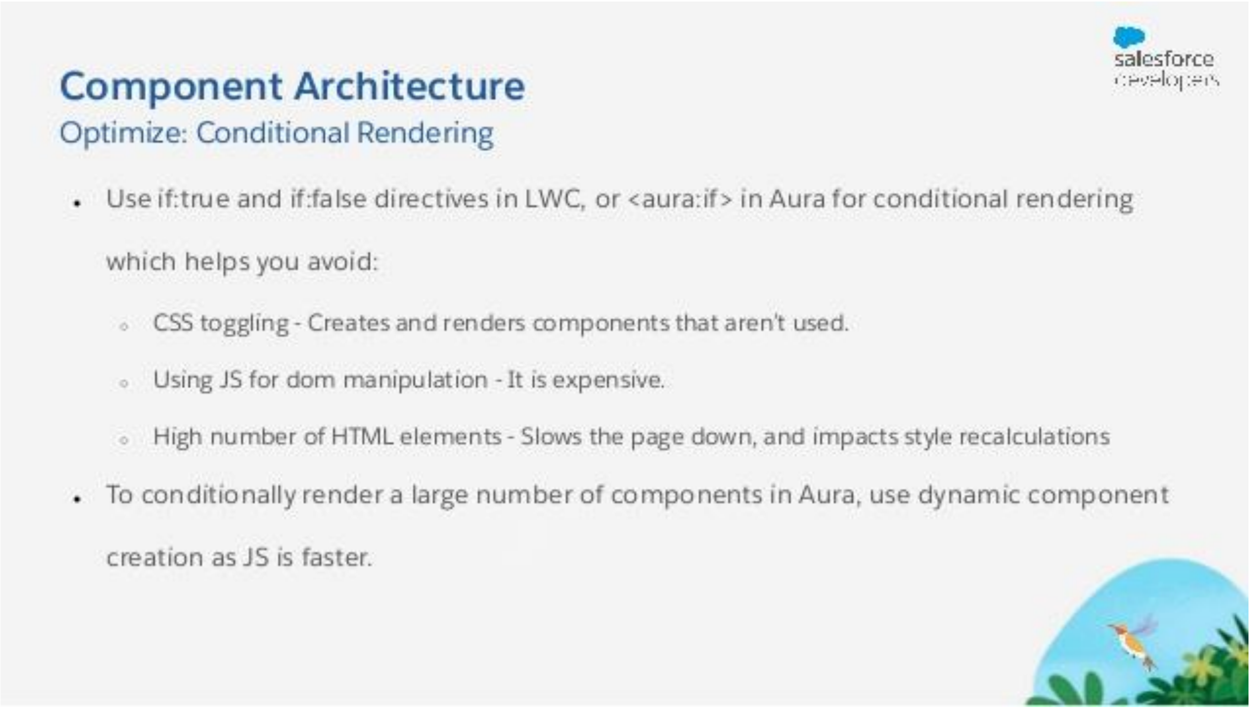
- Don't use a list view to scroll through large volume of content
  - Use a search bar
  - Use filters. e.g. Picklists, Owners, Date Range etc.
  - Use Reports
- Reduce the number of columns displayed
- Disable inline editing, if needed
- Disable Smooth Scrolling and Threaded Scrolling in Chrome, if needed



Component Architecture

Measure

- Use Chrome Developer Tools Performance timeline to drill down into each operation in the rendering pipeline
- Use the Lightning Inspector Chrome plugin (For Aura)
- Chrome Code Coverage Report for JS.



Component Architecture

Optimize: Conditional Rendering

- Use if:true and if:false directives in LWC, or <aura:if> in Aura for conditional rendering which helps you avoid:
  - CSS toggling - Creates and renders components that aren't used.
  - Using JS for dom manipulation - It is expensive.
  - High number of HTML elements - Slows the page down, and impacts style recalculations
- To conditionally render a large number of components in Aura, use dynamic component creation as JS is faster.

Component Architecture

Optimize: Fix Performance Warnings in Aura

- **Clean Unrendered Body**

  Occurs when you change the isTrue/if:true from true to false in the same rendering cycle. The unrendered body must be destroyed, which is avoidable work for the framework that slows down rendering time.

- **Multiple Items Set**

  Occurs when you set the items attribute of an <aura:iteration> or <template for:each> multiple times in the same rendering cycle.

Component Architecture

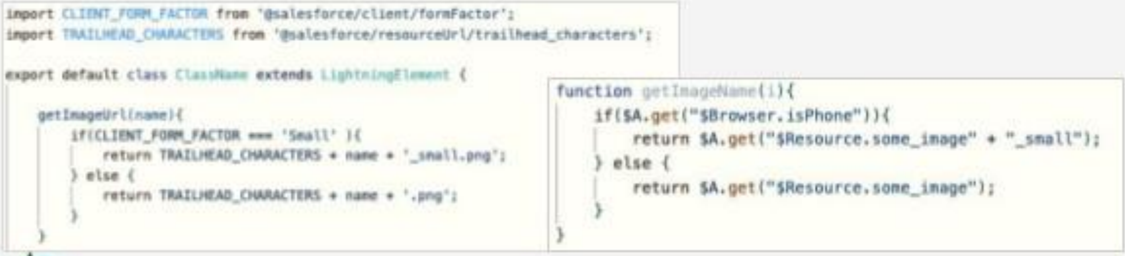
Optimize: A few tips

- Use caching features wherever possible
  - Lightning Data Service
  - Wire Decorators
  - Platform Cache
- Use Lazy Instantiation
  - Avoid creating components until the user clicks for them.
  - Use Pagination and Tabbed approach
  - Base Lightning Components like <lightning-tabset> and <lightning-tab> support lazy instantiation by default.
- Remove unused CSS, Avoid Complex selectors

Component Architecture

Optimize: A few tips

- Reduce Lightning Out dependencies
  - As a best practice only include top level components and expensive-to-create components.
  - Don't list the ones in lightning namespace
- Optimize images for mobile



```
import CLIENT_FORM_FACTOR from '@salesforce/client/formFactor';
import TRAILHEAD_CHARACTERS from '@salesforce/resourceUrl/trailhead_characters';

export default class ClassName extends LightningElement {

    getImageUrl(name){
        if(CLIENT_FORM_FACTOR === 'Small' ){
            return TRAILHEAD_CHARACTERS + name + '_small.png';
        } else {
            return TRAILHEAD_CHARACTERS + name + '.png';
        }
    }
```

```
function getImageName(i){
    if($A.get("$Browser.isPhone")){
        return $A.get("$Resource.some_image" + "_small");
    } else {
        return $A.get("$Resource.some_image");
    }
}
```

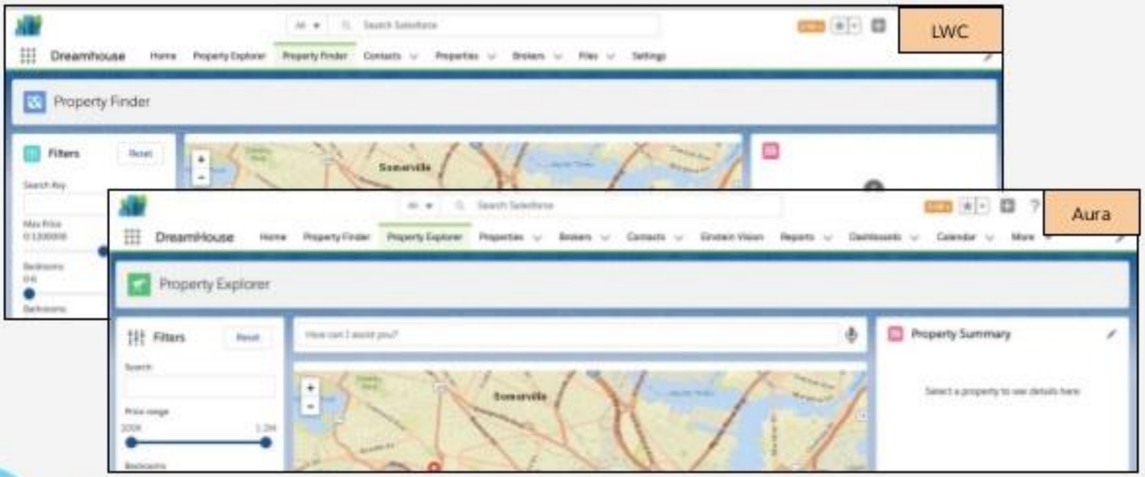
Component Architecture

Optimize: A few tips

- Use IFRAMEs sparingly
  - An iframe can block Lightning Component loading
  - Browser opens 6 connections for resources. iframes can hold many of them
- Minimize frequent navigation
  - Navigation makes components and pages reload
  - Include user-needed data as components
- Migrate component trees to LWC

Component Architecture

Optimize: Migrate to Lightning Web Components



Component Architecture

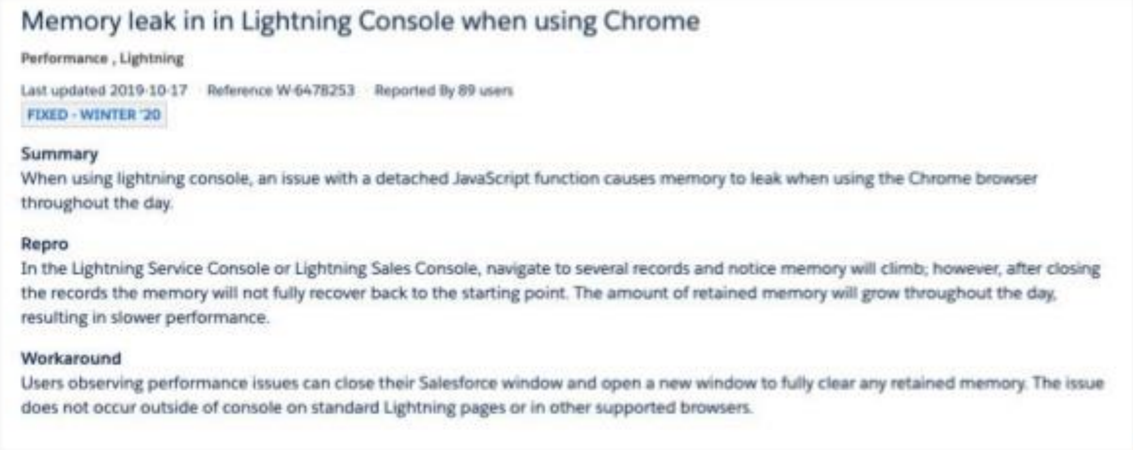
Optimize: Lightning Web Components Case Study

Page	Cache State	Aura (EPT, ms)	LWC (EPT, ms)	% Improvement
Property Finder	Cold	1080	813	-24.72%
Property Finder	Warm	428	157	-63.32%
Property Explorer	Cold	912	890	-2.41%
Property Explorer	Warm	289	197	-31.83%

Read more: Case Study: DreamHouse Gains Speed by Switching to LWC

Component Architecture

Optimize: Feel free to report issues



Memory leak in in Lightning Console when using Chrome

Performance , Lightning

Last updated 2019-10-17 · Reference W-6478253 · Reported By 89 users

FIXED - WINTER '20

Summary

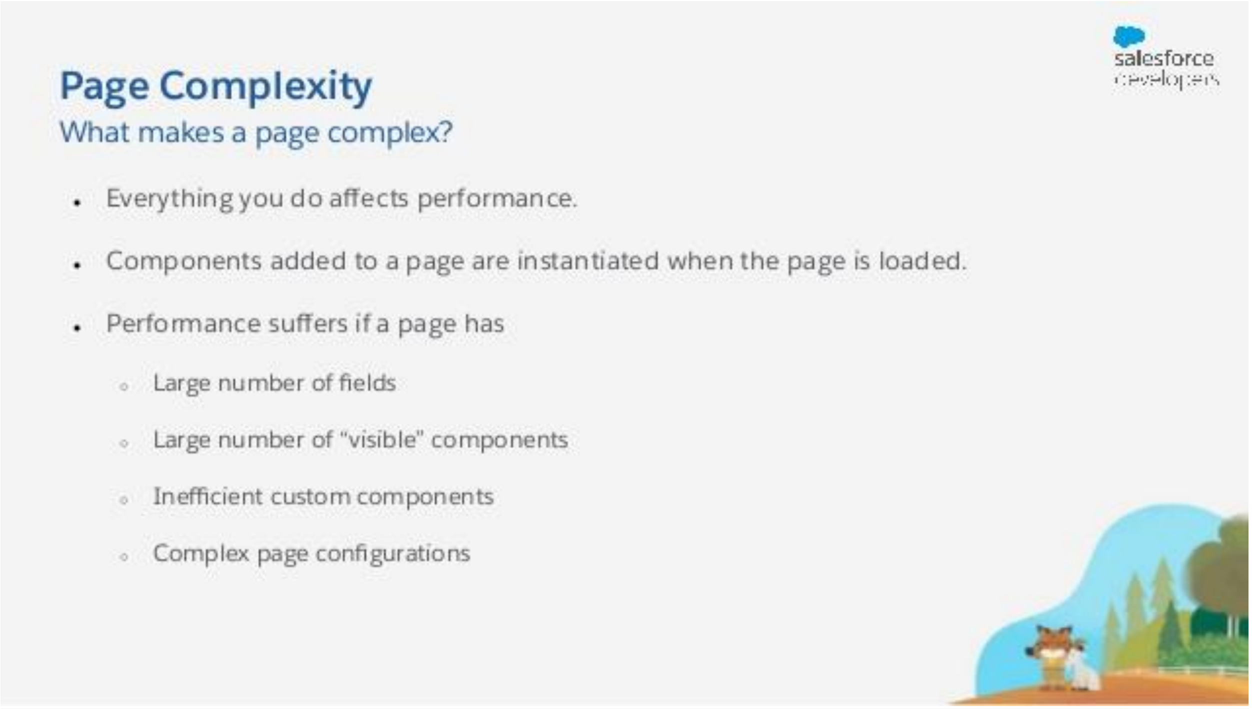
When using lightning console, an issue with a detached JavaScript function causes memory to leak when using the Chrome browser throughout the day.

Repro

In the Lightning Service Console or Lightning Sales Console, navigate to several records and notice memory will climb; however, after closing the records the memory will not fully recover back to the starting point. The amount of retained memory will grow throughout the day, resulting in slower performance.

Workaround

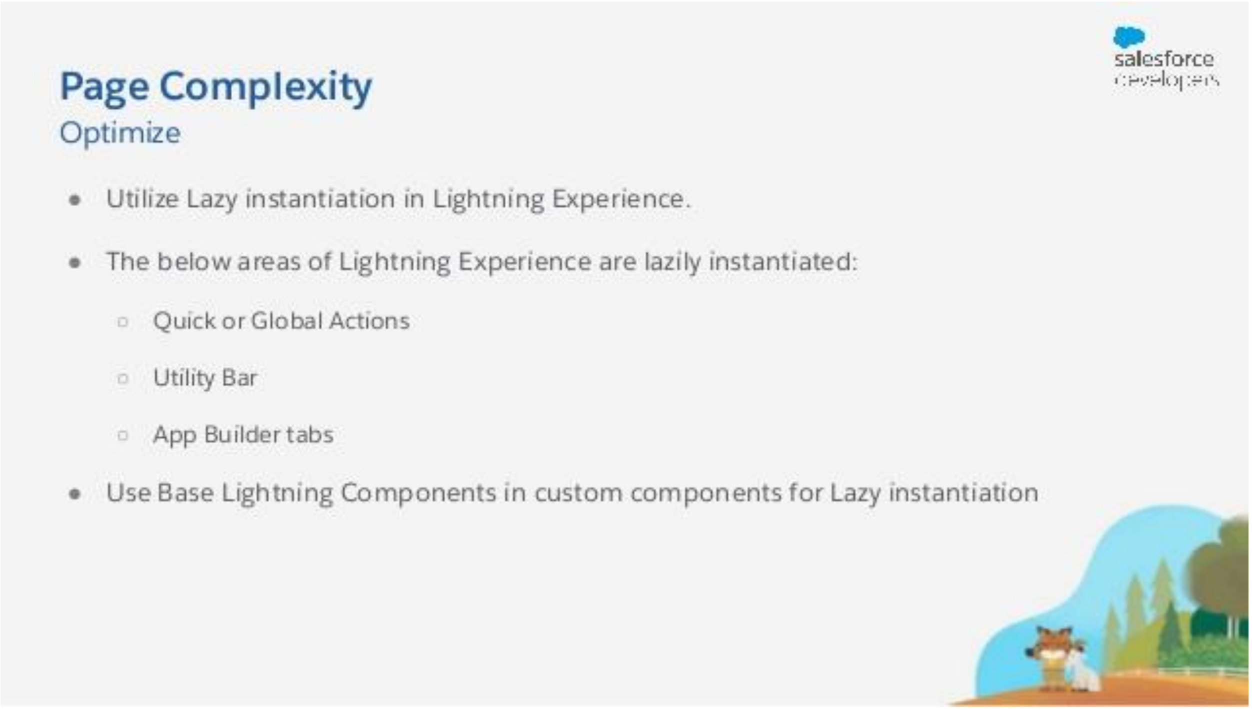
Users observing performance issues can close their Salesforce window and open a new window to fully clear any retained memory. The issue does not occur outside of console on standard Lightning pages or in other supported browsers.



Page Complexity

What makes a page complex?

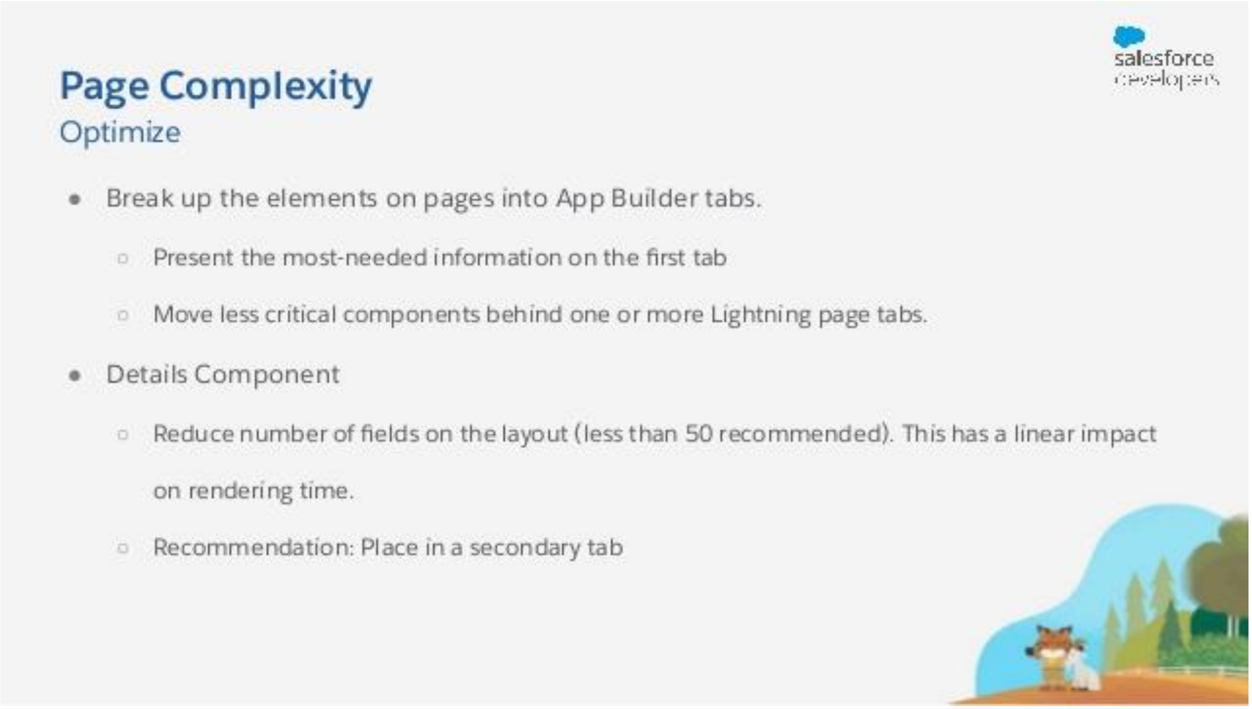
- Everything you do affects performance.
- Components added to a page are instantiated when the page is loaded.
- Performance suffers if a page has
  - Large number of fields
  - Large number of “visible” components
  - Inefficient custom components
  - Complex page configurations



Page Complexity

Optimize

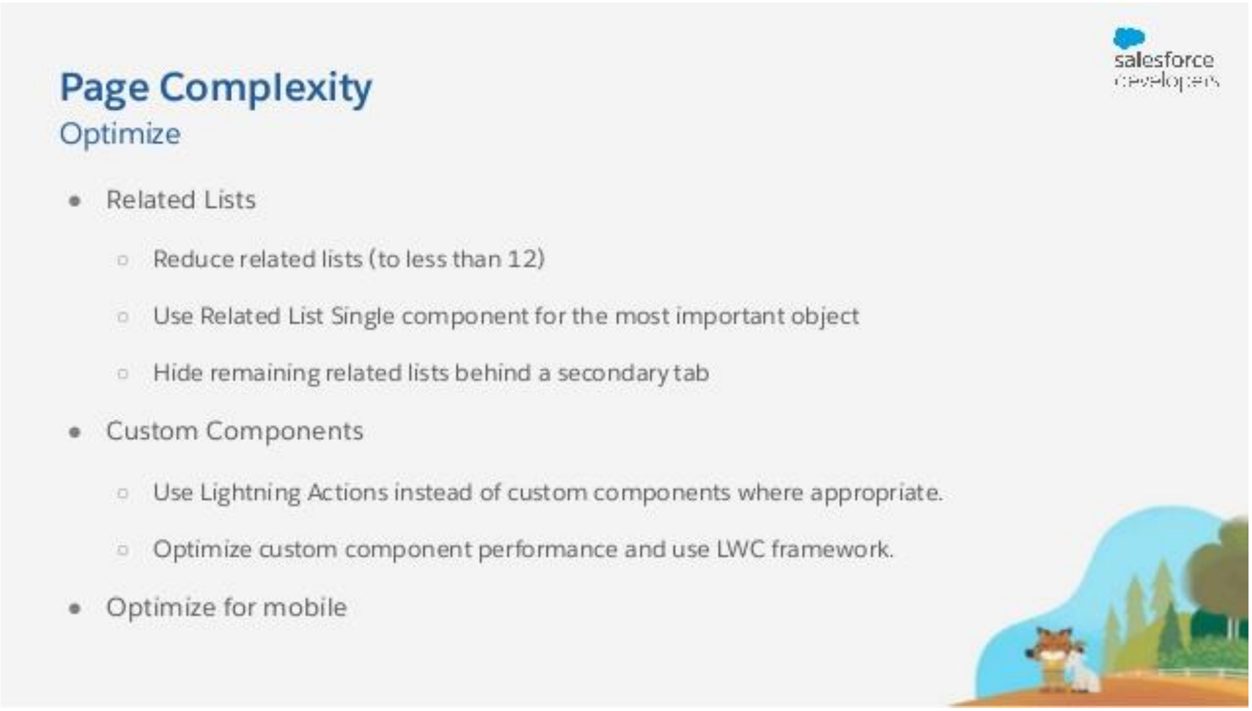
- Utilize Lazy instantiation in Lightning Experience.
- The below areas of Lightning Experience are lazily instantiated:
  - Quick or Global Actions
  - Utility Bar
  - App Builder tabs
- Use Base Lightning Components in custom components for Lazy instantiation



Page Complexity

Optimize

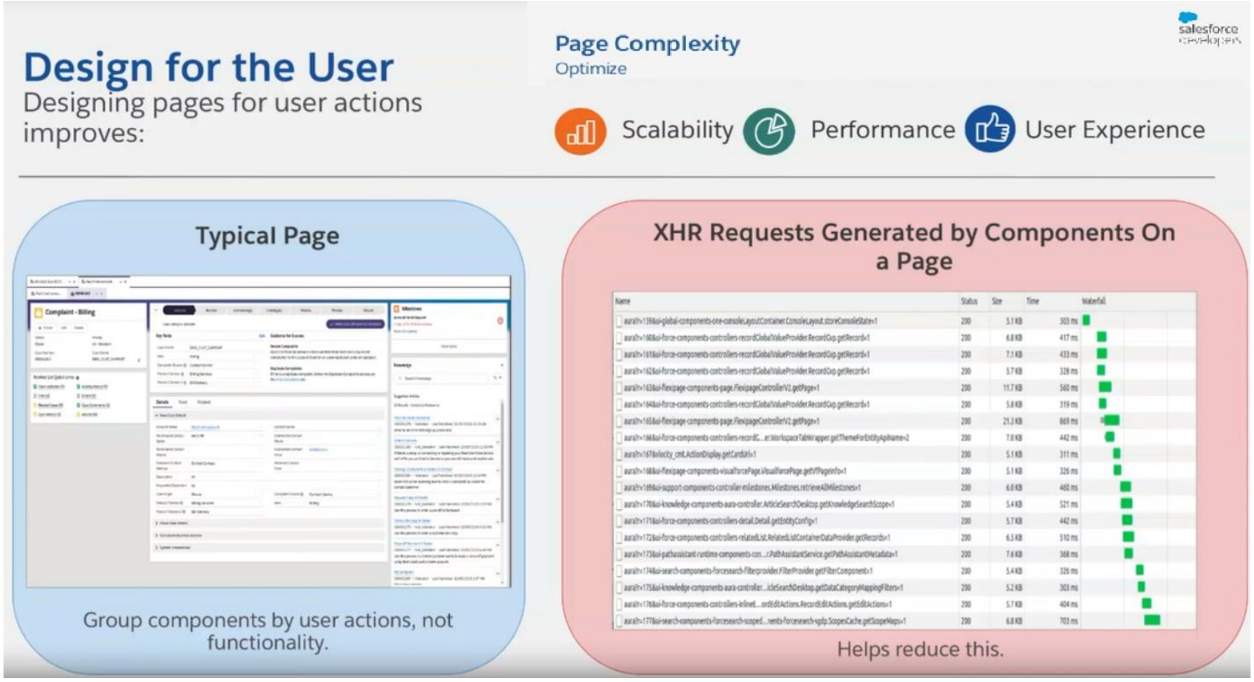
- Break up the elements on pages into App Builder tabs.
  - Present the most-needed information on the first tab
  - Move less critical components behind one or more Lightning page tabs.
- Details Component
  - Reduce number of fields on the layout (less than 50 recommended). This has a linear impact on rendering time.
  - Recommendation: Place in a secondary tab



Page Complexity

Optimize

- Related Lists
  - Reduce related lists (to less than 12)
  - Use Related List Single component for the most important object
  - Hide remaining related lists behind a secondary tab
- Custom Components
  - Use Lightning Actions instead of custom components where appropriate.
  - Optimize custom component performance and use LWC framework.
- Optimize for mobile



Design for the User

Designing pages for user actions improves:

Page Complexity

Optimize

Scalability · Performance · User Experience

Typical Page

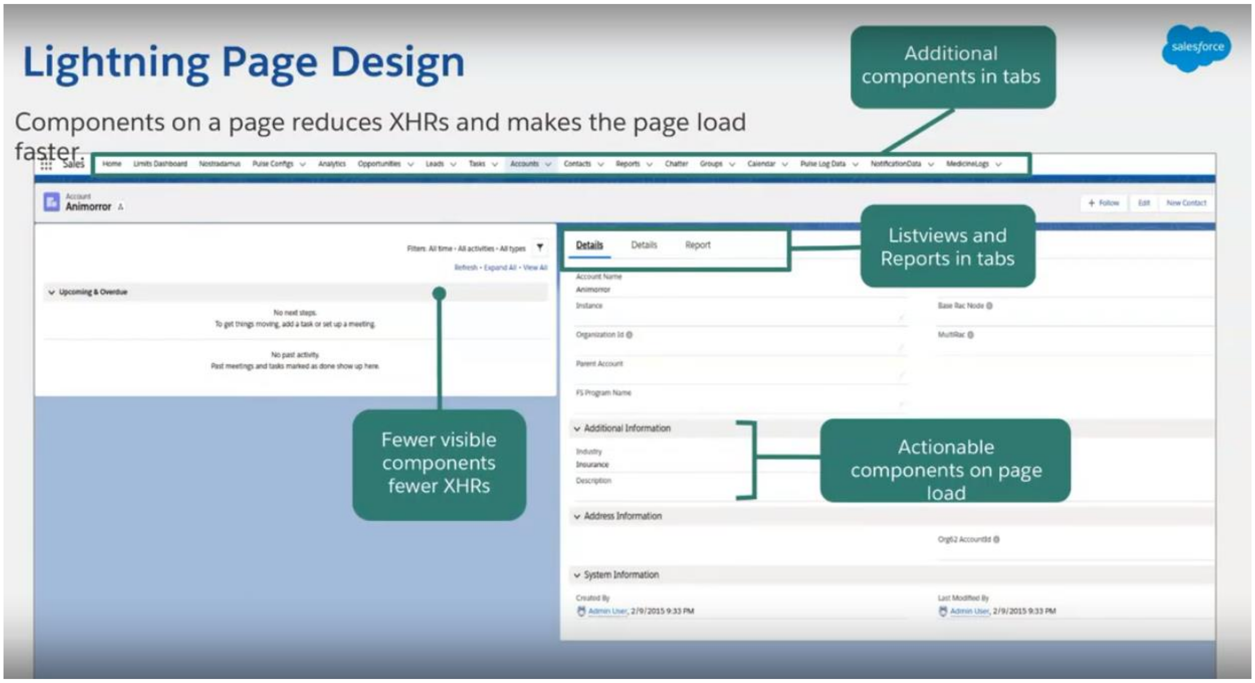
Group components by user actions, not functionality.

XHR Requests Generated by Components On a Page

Name	Status	Size	Time	Waterfall
aura?r=159&ui-global-components-one-consoleLayoutContainer.ConsoleLayout.storeConsoleState=1	200	5.1 KB	303 ms	🟩
aura?r=160&ui-force-components-controllers-recordGlobalValueProvider.RecordGvp.getRecord=1	200	6.8 KB	417 ms	🟩
aura?r=161&ui-force-components-controllers-recordGlobalValueProvider.RecordGvp.getRecord=1	200	7.1 KB	433 ms	🟩
aura?r=162&ui-force-components-controllers-recordGlobalValueProvider.RecordGvp.getRecord=1	200	5.7 KB	328 ms	🟩
aura?r=163&ui-flexipage-components-page.FlexipageControllerV2.getPage=1	200	11.7 KB	560 ms	🟩
aura?r=164&ui-force-components-controllers-recordGlobalValueProvider.RecordGvp.getRecord=1	200	5.8 KB	319 ms	🟩
aura?r=165&ui-flexipage-components-page.FlexipageControllerV2.getPage=1	200	21.3 KB	869 ms	🟩
aura?r=166&ui-force-components-controllers-recordG...er.WorkspaceTabWrapper.getThemeForEntityApiName=2	200	7.0 KB	442 ms	🟩
aura?r=167&vlocity_cmt.ActionDisplay.getCardUrl=1	200	5.1 KB	311 ms	🟩
aura?r=168&ui-flexipage-components-visualforcePage.VisualforcePage.getVfPageInfo=1	200	5.1 KB	326 ms	🟩
aura?r=169&ui-support-components-controller-milestones.Milestones.retrieveAllMilestones=1	200	6.0 KB	460 ms	🟩
aura?r=170&ui-knowledge-components-aura-controller.ArticleSearchDesktop.getKnowledgeSearchScope=1	200	5.4 KB	521 ms	🟩
aura?r=171&ui-force-components-controllers-detail.Detail.getEntityConfig=1	200	5.7 KB	442 ms	🟩
aura?r=172&ui-force-components-controllers-relatedList.RelatedListContainerDataProvider.getRecords=1	200	6.5 KB	510 ms	🟩
aura?r=173&ui-pathassistant-runtime-components-con...r.PathAssistantService.getPathAssistantMetadata=1	200	7.6 KB	368 ms	🟩
aura?r=174&ui-search-components-forcesearch-filterprovider.FilterProvider.getFilterComponent=1	200	5.4 KB	326 ms	🟩
aura?r=175&ui-knowledge-components-aura-controller...icleSearchDesktop.getDataCategoryMappingFilters=1	200	5.2 KB	303 ms	🟩
aura?r=176&ui-force-components-controllers-inlineE...ordEditActions.RecordEditActions.getEditActions=1	200	5.7 KB	404 ms	🟩
aura?r=177&ui-search-components-forcesearch-scoped...nents-forcesearch-sgdp.ScopesCache.getScopeMaps=1	200	6.8 KB	703 ms	🟩

Helps reduce this.

Tip-1



Lightning Page Design

Components on a page reduces XHRs and makes the page load faster.

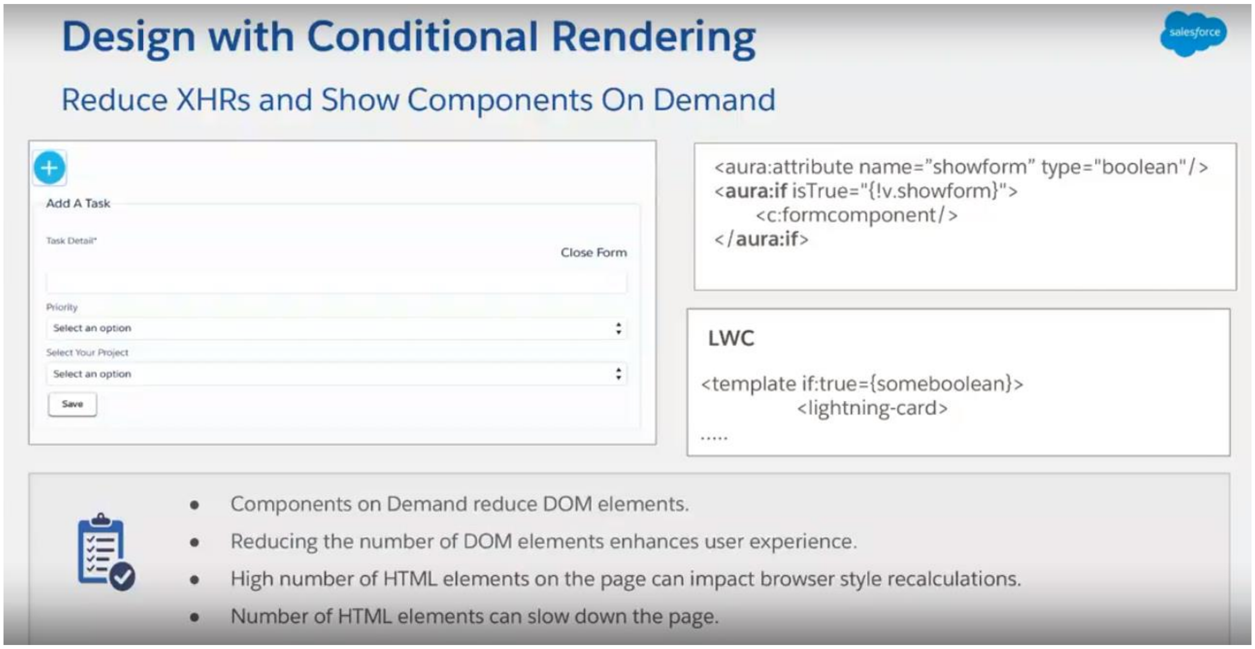
Additional components in tabs

Listviews and Reports in tabs

Fewer visible components fewer XHRs

Actionable components on page load

Tip-2



Design with Conditional Rendering

Reduce XHRs and Show Components On Demand

Add A Task

Task Detail*

Close Form

Priority

Select an option

Select Your Project

Select an option

Save

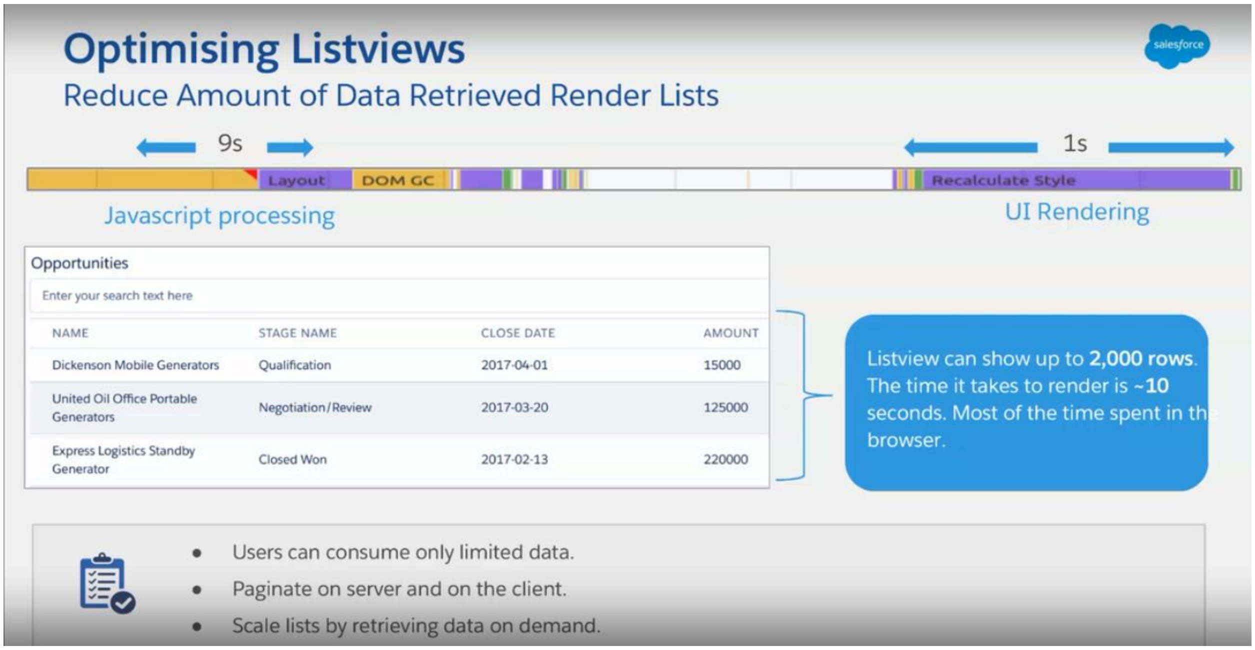
```
<aura:attribute name="showform" type="boolean"/>
<aura:if isTrue="{!v.showform}">
      <c:formcomponent/>
</aura:if>
```

LWC

```
<template if:true={someboolean}>
          <lightning-card>
.....
```

- Components on Demand reduce DOM elements.
- Reducing the number of DOM elements enhances user experience.
- High number of HTML elements on the page can impact browser style recalculations.
- Number of HTML elements can slow down the page.

Tip-3



Optimising Listviews

Reduce Amount of Data Retrieved Render Lists

Opportunities

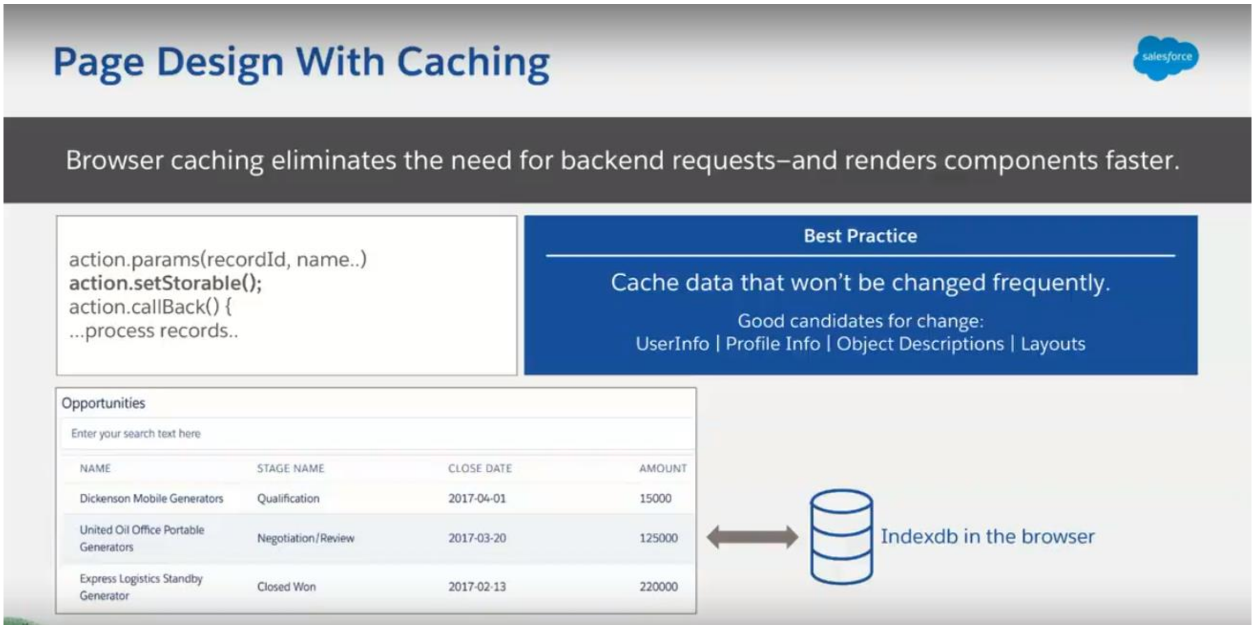
Enter your search text here

NAME	STAGE NAME	CLOSE DATE	AMOUNT
Dickenson Mobile Generators	Qualification	2017-04-01	15000
United Oil Office Portable Generators	Negotiation/Review	2017-03-20	125000
Express Logistics Standby Generator	Closed Won	2017-02-13	220000

Listview can show up to **2,000 rows**. The time it takes to render is ~**10** seconds. Most of the time spent in th browser.

- Users can consume only limited data.
- Paginate on server and on the client.
- Scale lists by retrieving data on demand.

Tip-4



Page Design With Caching

Browser caching eliminates the need for backend requests–and renders components faster.

```
action.params(recordId, name..)
action.setStorable();
action.callBack() {
...process records..
```

Best Practice

Cache data that won't be changed frequently.

Good candidates for change:
UserInfo | Profile Info | Object Descriptions | Layouts

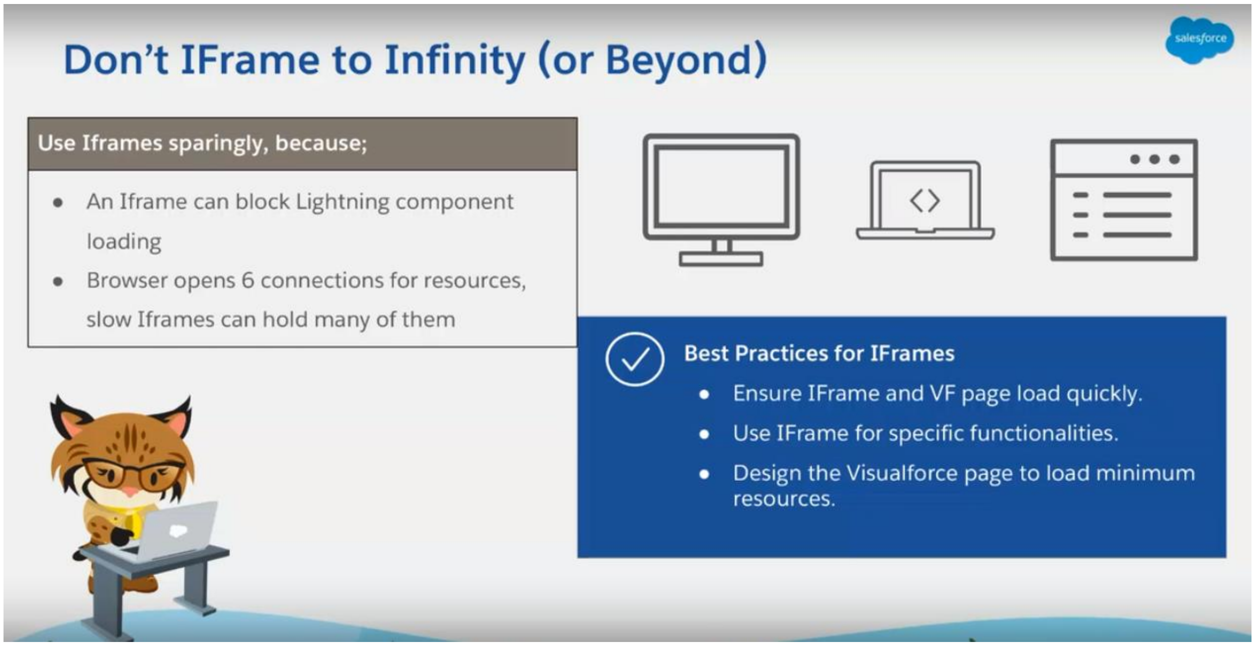
Opportunities

Enter your search text here

NAME	STAGE NAME	CLOSE DATE	AMOUNT
Dickenson Mobile Generators	Qualification	2017-04-01	15000
United Oil Office Portable Generators	Negotiation/Review	2017-03-20	125000
Express Logistics Standby Generator	Closed Won	2017-02-13	220000

Indexdb in the browser

Tip-5



Don't IFrame to Infinity (or Beyond)

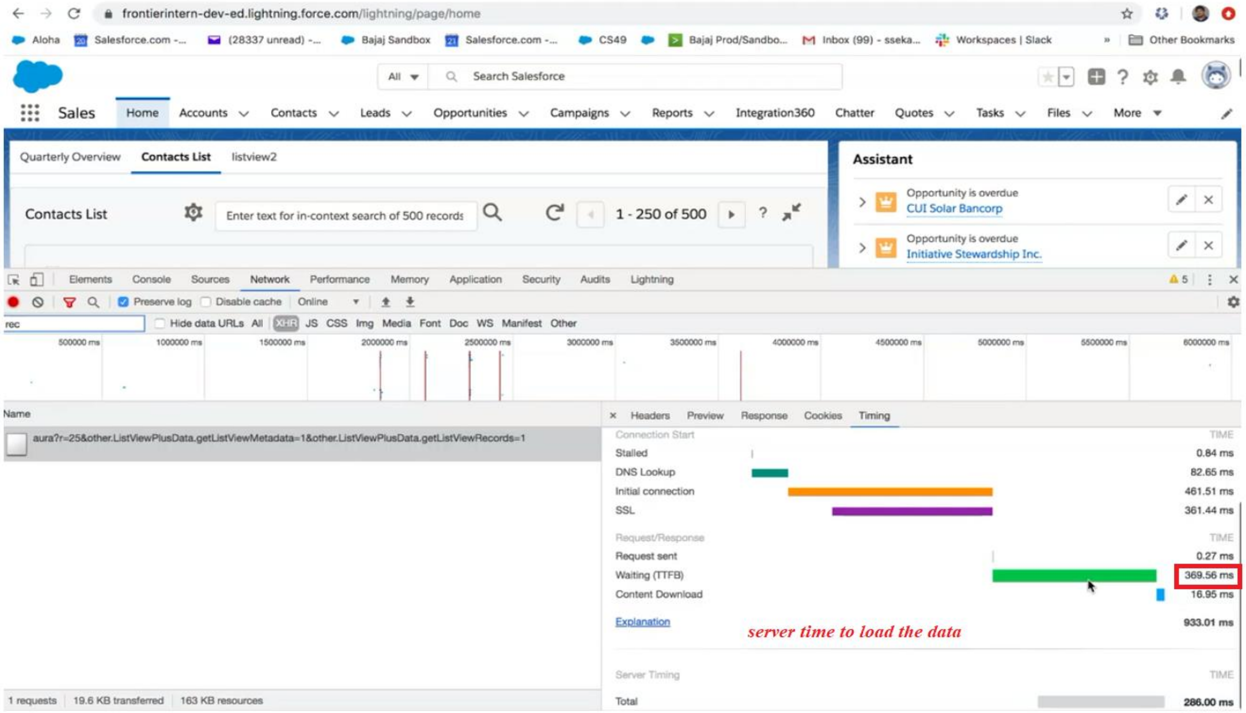
Use Iframes sparingly, because;

- An Iframe can block Lightning component loading
- Browser opens 6 connections for resources, slow Iframes can hold many of them

Best Practices for IFrames

- Ensure IFrame and VF page load quickly.
- Use IFrame for specific functionalities.
- Design the Visualforce page to load minimum resources.

Tip-6



Contacts List

Enter text for in-context search of 500 records

1 - 250 of 500

Assistant

Opportunity is overdue
CUI Solar Bancorp

Opportunity is overdue
Initiative Stewardship Inc.

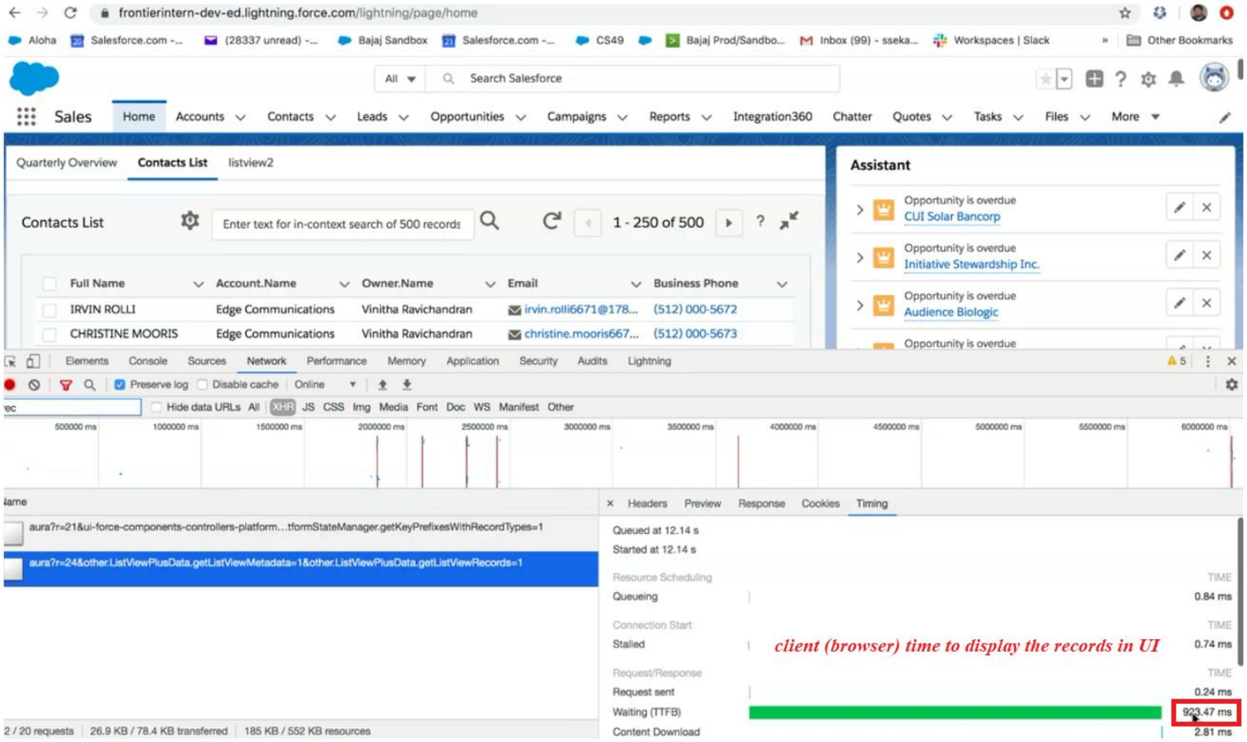
aura?r=25&other.ListViewPlusData.getListViewMetadata=1&other.ListViewPlusData.getListViewRecords=1

Connection Start	TIME
Stalled	0.84 ms
DNS Lookup	82.65 ms
Initial connection	461.51 ms
SSL	361.44 ms
Request/Response	**TIME**
Request sent	0.27 ms
Waiting (TTFB)	369.56 ms
Content Download	16.95 ms
Explanation	933.01 ms
Server Timing	**TIME**
Total	286.00 ms

server time to load the data

1 requests | 19.6 KB transferred | 163 KB resources

Tested multiple times by refreshing the page(below is the average time)



Contacts List

Enter text for in-context search of 500 records

1 - 250 of 500

Full Name	Account.Name	Owner.Name	Email	Business Phone
IRVIN ROLLI	Edge Communications	Vinitha Ravichandran	irvin.rolli6671@178...	(512) 000-5672
CHRISTINE MOORIS	Edge Communications	Vinitha Ravichandran	christine.mooris667...	(512) 000-5673

Assistant

Opportunity is overdue
CUI Solar Bancorp

Opportunity is overdue
Initiative Stewardship Inc.

Opportunity is overdue
Audience Biologic

aura?r=21&ui-force-components-controllers-platform...tformStateManager.getKeyPrefixesWithRecordTypes=1
aura?r=24&other.ListViewPlusData.getListViewMetadata=1&other.ListViewPlusData.getListViewRecords=1

Queued at 12.14 s
Started at 12.14 s

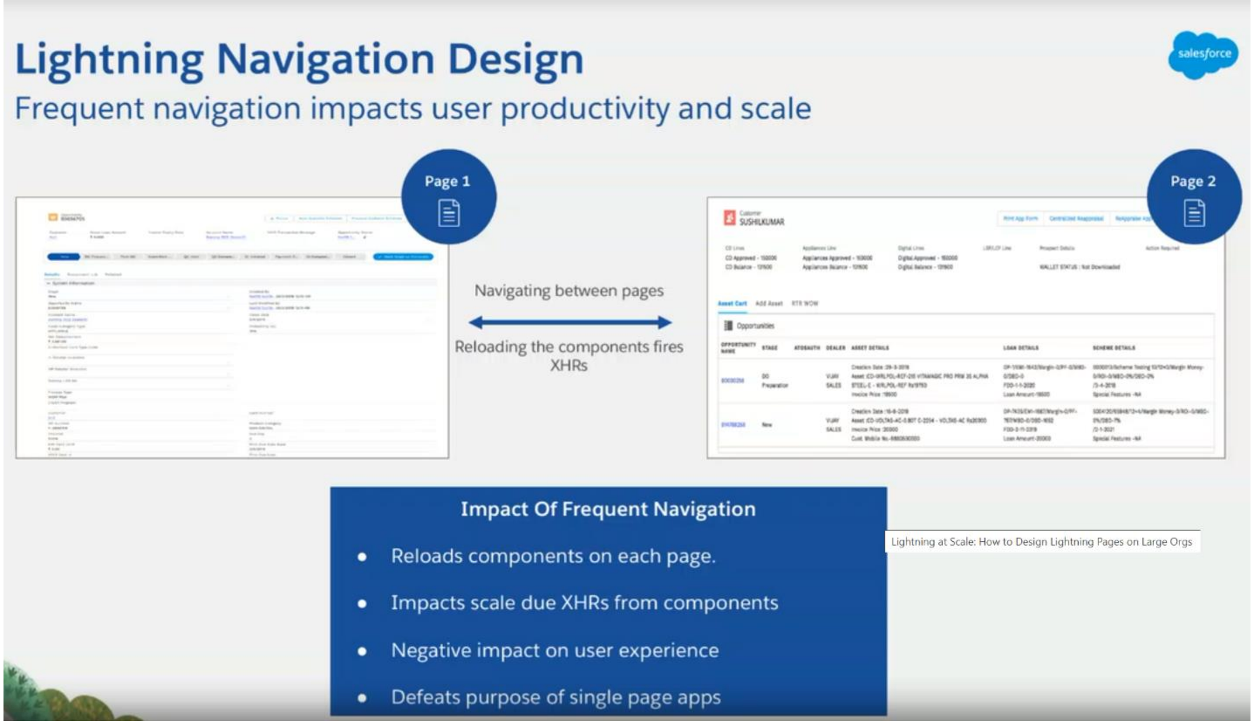
Resource Scheduling	TIME
Queueing	0.84 ms
Connection Start	**TIME**
Stalled	0.74 ms
Request/Response	**TIME**
Request sent	0.24 ms
Waiting (TTFB)	923.47 ms
Content Download	2.81 ms

client (browser) time to display the records in UI

2 / 20 requests | 26.9 KB / 78.4 KB transferred | 185 KB / 552 KB resources

So, Server Processing time is less than Client processing time (JavaScript takes more time to render the page)

Tip-7



Lightning Navigation Design

Frequent navigation impacts user productivity and scale

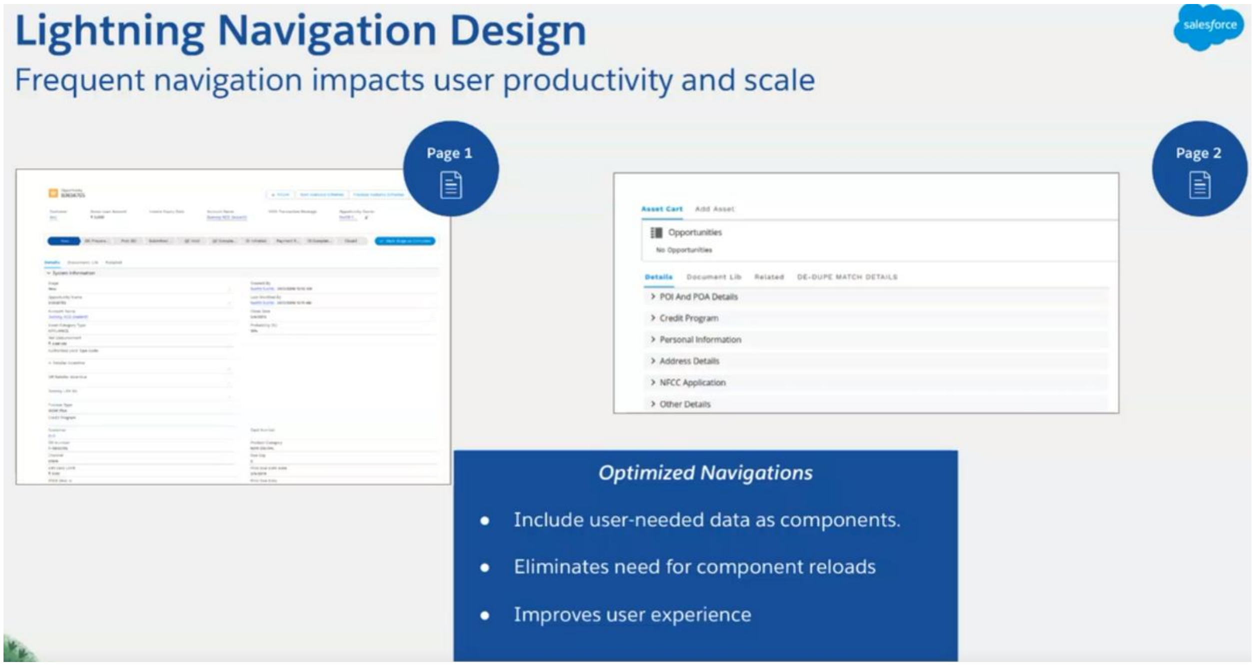
Navigating between pages

Reloading the components fires XHRs

Impact Of Frequent Navigation

- Reloads components on each page.
- Impacts scale due XHRs from components
- Negative impact on user experience
- Defeats purpose of single page apps

Lightning at Scale: How to Design Lightning Pages on Large Orgs

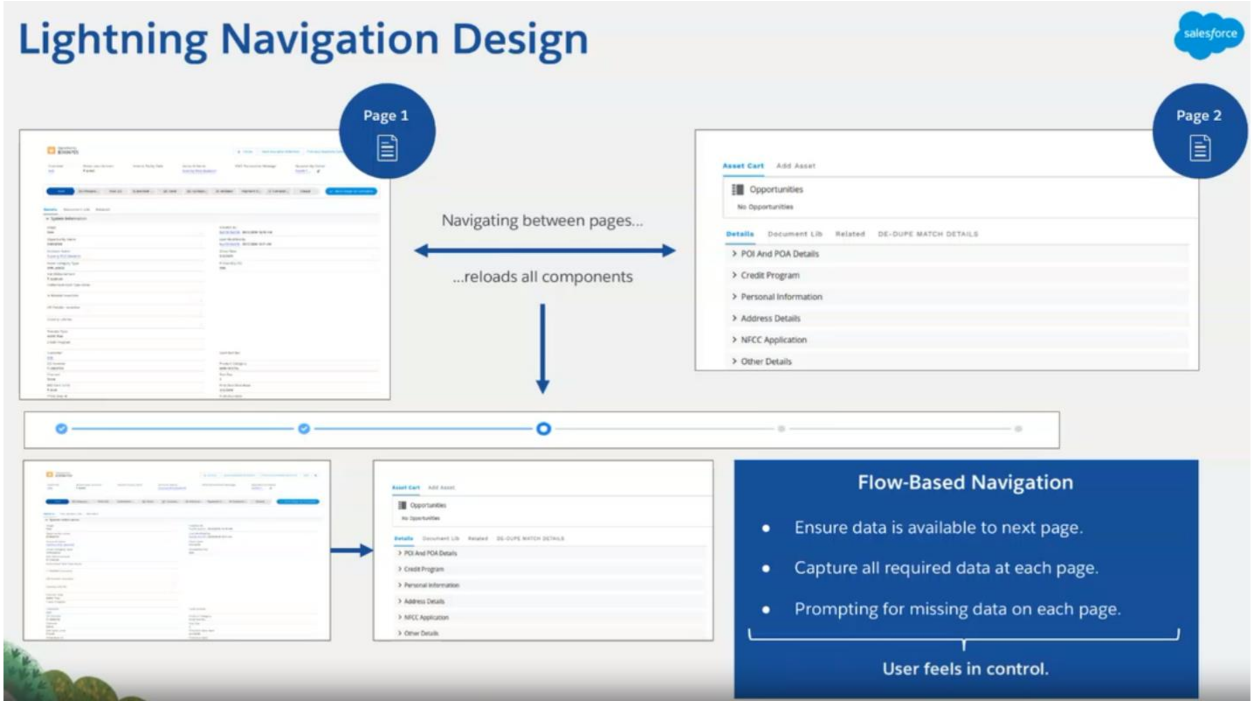


Lightning Navigation Design

Frequent navigation impacts user productivity and scale

Optimized Navigations

- Include user-needed data as components.
- Eliminates need for component reloads
- Improves user experience



Lightning Navigation Design

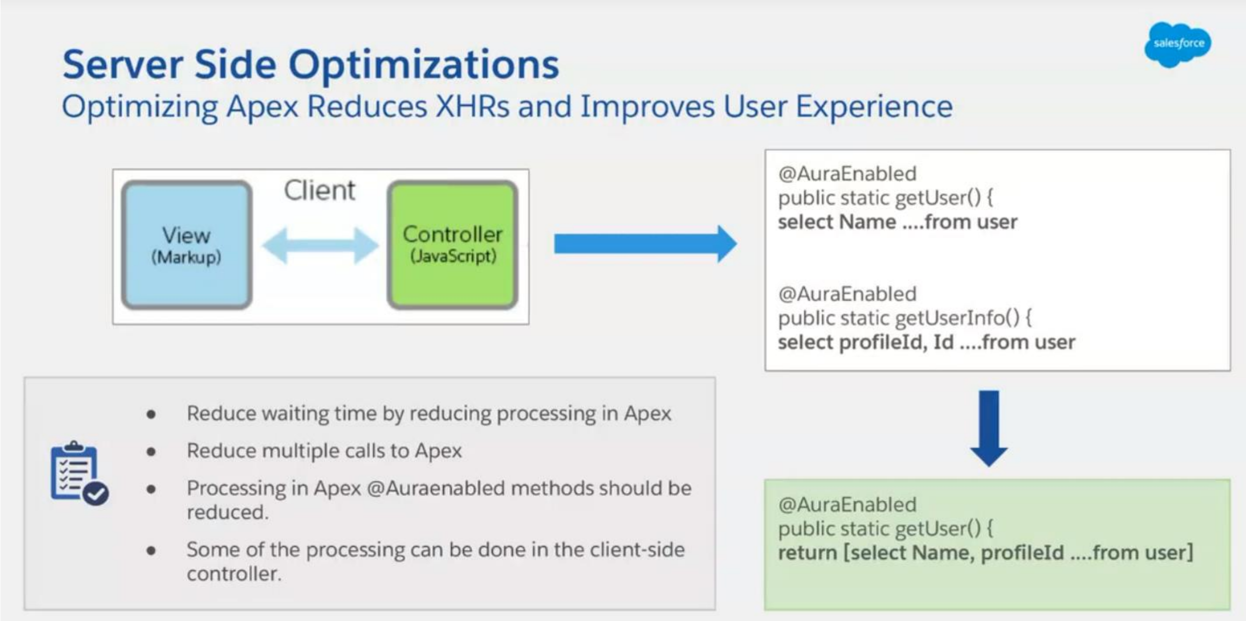
Navigating between pages...

...reloads all components

Flow-Based Navigation

- Ensure data is available to next page.
- Capture all required data at each page.
- Prompting for missing data on each page.

User feels in control.



Server Side Optimizations

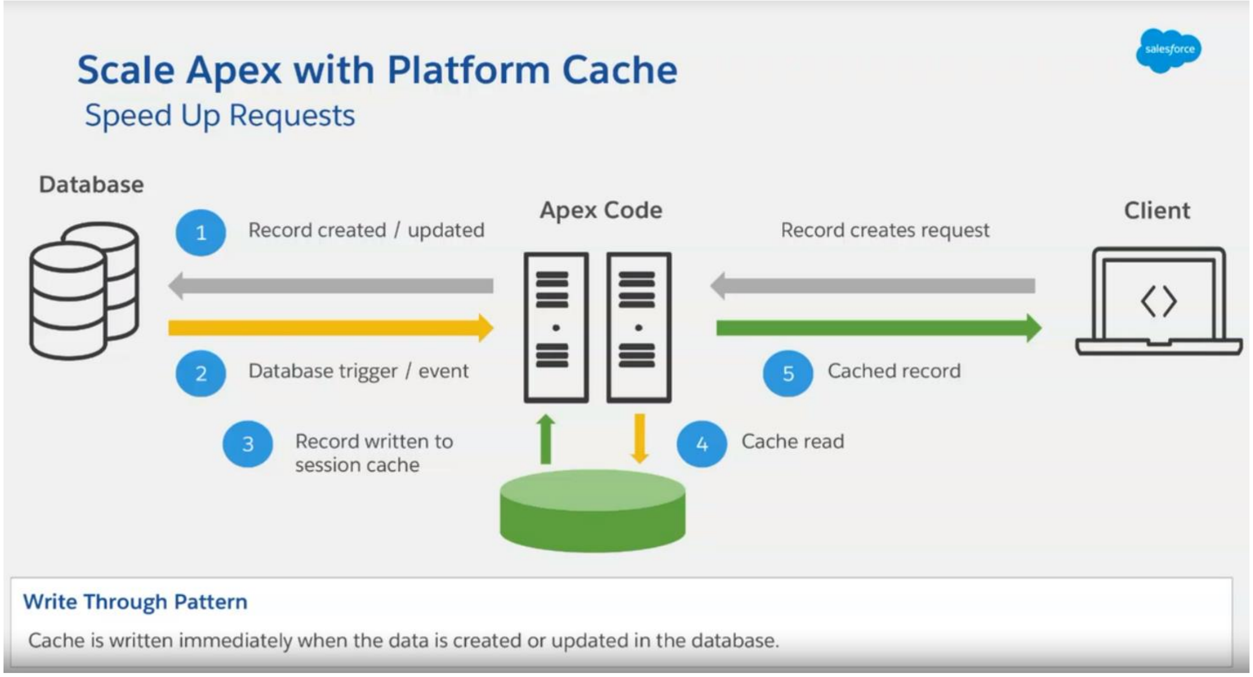
Optimizing Apex Reduces XHRs and Improves User Experience

```
@AuraEnabled
public static getUser() {
select Name ....from user

@AuraEnabled
public static getUserInfo() {
select profileId, Id ....from user
```

```
@AuraEnabled
public static getUser() {
return [select Name, profileId ....from user]
```

- Reduce waiting time by reducing processing in Apex
- Reduce multiple calls to Apex
- Processing in Apex @Auraenabled methods should be reduced.
- Some of the processing can be done in the client-side controller.



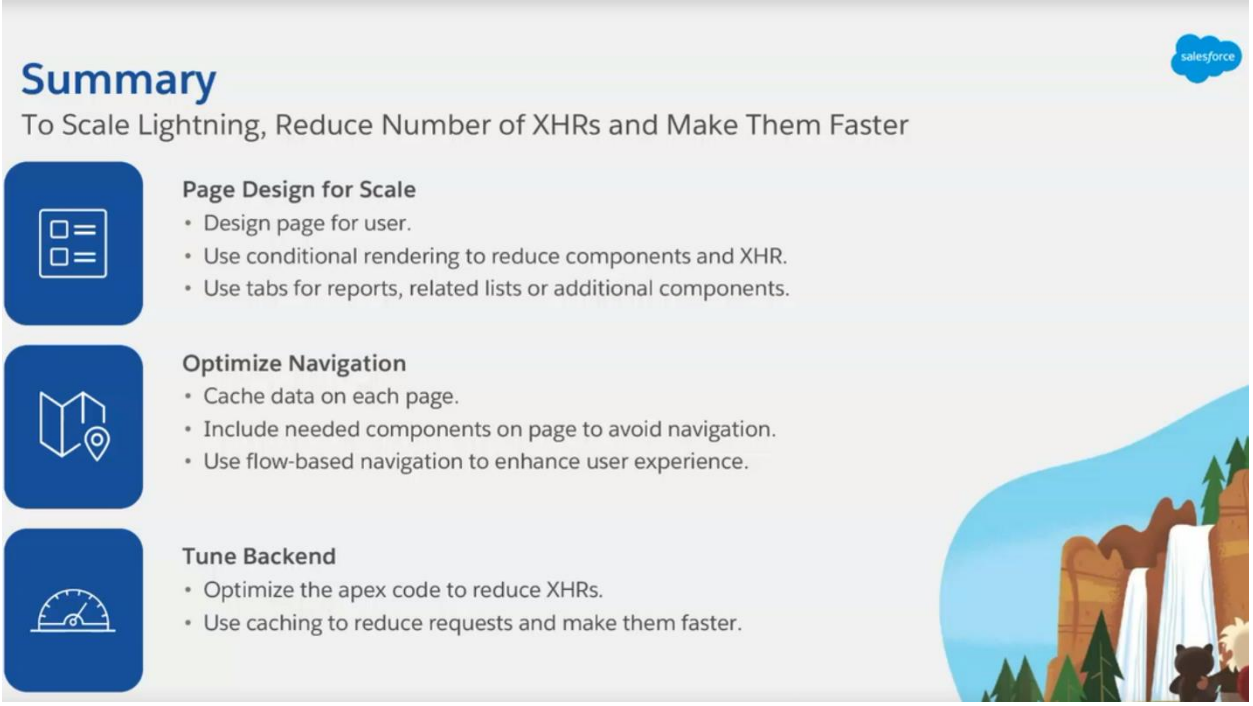
Scale Apex with Platform Cache

Speed Up Requests

Database
Apex Code
Client
1 Record created / updated
2 Database trigger / event
3 Record written to session cache
4 Cache read
Record creates request
5 Cached record

Write Through Pattern

Cache is written immediately when the data is created or updated in the database.



Summary

To Scale Lightning, Reduce Number of XHRs and Make Them Faster

Page Design for Scale

- Design page for user.
- Use conditional rendering to reduce components and XHR.
- Use tabs for reports, related lists or additional components.

Optimize Navigation

- Cache data on each page.
- Include needed components on page to avoid navigation.
- Use flow-based navigation to enhance user experience.

Tune Backend

- Optimize the apex code to reduce XHRs.
- Use caching to reduce requests and make them faster.

Few More Tips

- Reducing Fields: When setting up Lightning, users often have some 100-200 fields on the page, and this can lead to slow load speeds. Take some time to evaluate which fields are critical and reduce fields to what your clients will need.
- Related Lists: Active related lists can get bulky, so another way to speed things up is by minimizing the number of these on the page. Again, keep what you need, but get rid of the rest. Additionally, you might want to consider supplementing your related lists with related quick links. Related quick links allow users to access the information they need but does not require all the data to load onto the page.
- Disable unnecessary Plug-ins: Depending on the CPU power/memory resources, your plug-ins, and extensions use, you may have to disable some of them. Evaluate and test the plug-ins/extensions in your environment to troubleshoot which ones are causing slow speeds and then act accordingly.
- List Views, Reports, and Dashboards: To speed up the loading of list views, reports, and dashboards, avoid filtering by formula fields. Also, you should always try to filter by relevant date ranges so as not to query all records all the time (unless necessary). By filtering for date range, you will get faster results on filters for this year, Record Type, or any other picklist field. Also, when filtering on picklist or text fields, your results will query faster when you use “Equals” rather than “Contains.”
- Do not introduce too many locks especially while working on Batch Jobs.
- Try to avoid Junction objects.
- Stable Internet Connection: As a business user, a reliable internet connection is a must for your team. Again, Lightning relies on the browser and a reliable internet connection will be detrimental to the success of your Lightning integration. If necessary, switch to a different browser (for example, Salesforce notes that Chrome has performed more consistently than Internet Explorer). Just as well, ensure that the devices your team is on support the high CPU usage required to run Lightning. You can also do an Octane evaluation of your client’s devices to assess their browser processing capabilities.
- Restart Your Browser: It’s essential to maintain a consistent schedule for restarting your browser. Frequently, this helps limit the amount of devices/browser applications that are causing load speeds. By freeing up your resources, your operating system will work more efficiently and give more power to your Lightning Experience application.