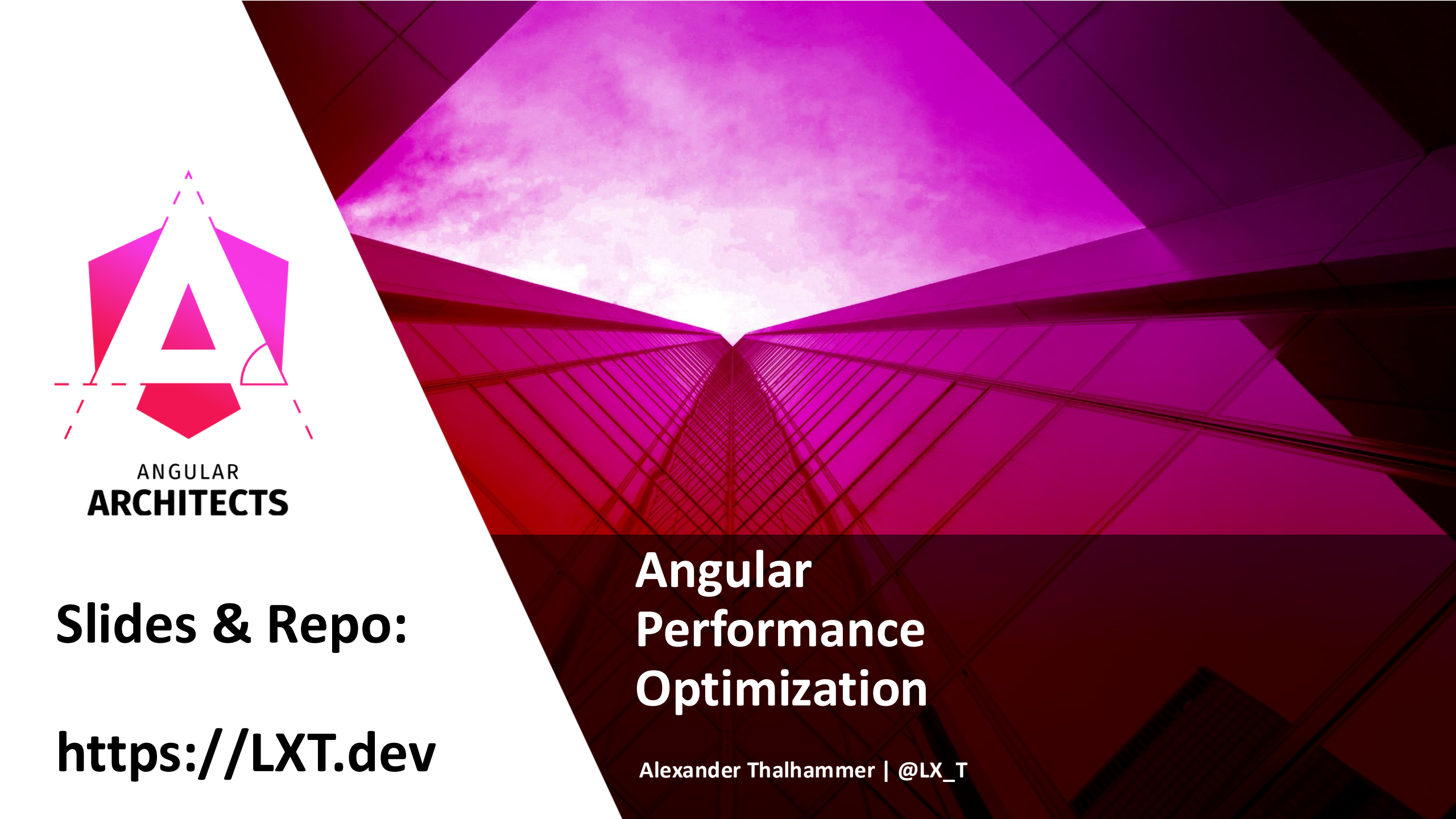


ANGULAR
ARCHITECTS

Angular Performance Optimization

Alexander Thalhammer | @LX_T

Slides & Repo:

https://LXT.dev

Agenda

Performance Tools

Runtime Perf & Change Det.

Initial Load Performance

Q & A or Lightning

Agenda Initial Load Performance

Lazy Loading Images & Build Opt.

Deferrable Views

Lazy Load. JS & Preloading

SSR & SSG

Performance Optimization

- now – 9:30 Performance Tools
- 9:30 – 10:30 Initial Load Lazy stuff
- 10:30 – 11:00  break
- 11:00 – 11:45 Initial Load modern shoot
- 11:45 – 12:15 Runtime Perf
- 12:15 – 12:30 Q&A else

Timetable

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE
SOFTWARE ARCHITECT

About me …

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE ARCHITECT

My office ☺

Hi, it's me → @LX_T

https://alex.thalhammer.name/

- **Alex Thalhammer** from Graz, Austria (since 1983)
  - Angular Software Tree GmbH (since 2019)
- WebDev for 22+ years (I‘ve come a long way baby)
- WordPress Dev (Web, PHP & jQuery, 2011 - 2017)
- **Angular Dev** (Web, TS, Rx since 2017 - NG 4.0.0 ☺)
- **Angular Evangelist, Coach** & **Consultant** (since 2020)
  - Member of Angular Architects

https://www.angulararchitects.io/



ANGULAR ARCHITECTS
INSIDE KNOWLEDGE
SOFTWARE ARCHITECT

Pollings

- OS
  - Mobile OS?
- Browser
- IntelliJ/WebStorm vs VS Code vs NeoVIM
- Which framework is the fastest?

Angular Performance Optimization

WTF?

Angular Performance Optimization

We distinguish between

- Initial load performance (classical web performance)
- Runtime performance (during usage, e.g. scrolling frame rate)

Starter Kit

- Clone from https://github.com/L-X-T/ng-days-perf-2024

Ready for Takeoff!

- What are your questions?

Img src: https://bit.ly/ng-tools-img

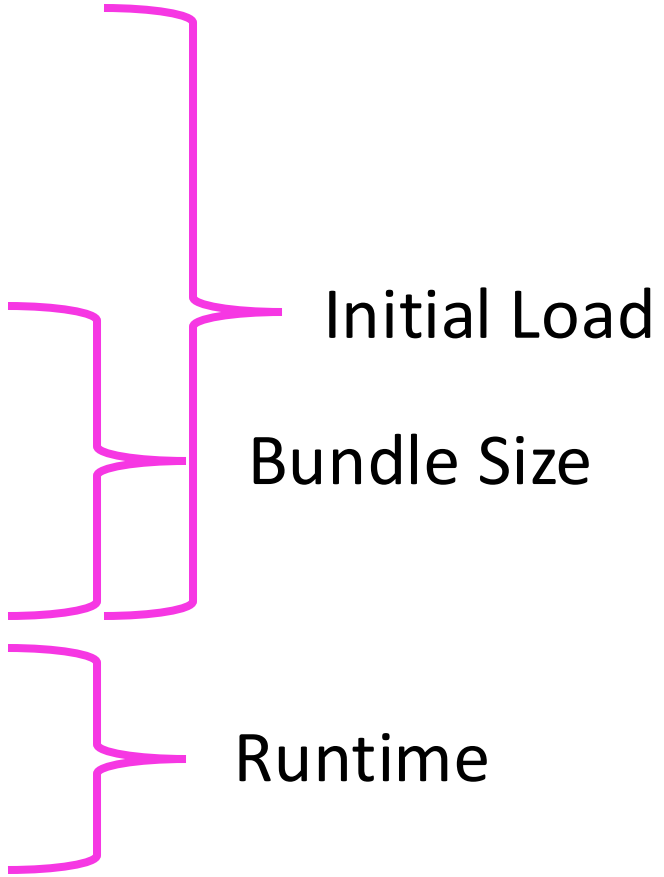
Audit Tools

- PageSpeed & Lighthouse
- Source Map Explorer
- Import Graph Visualizer
- Chrome DevTools
- Angular DevTools Profiler

Agenda

Audit Tools

- PageSpeed & Lighthouse
- Source Map Explorer
- Import Graph Visualizer
- Chrome DevTools
- Angular DevTools Profiler



Agenda

Web Audit Tools

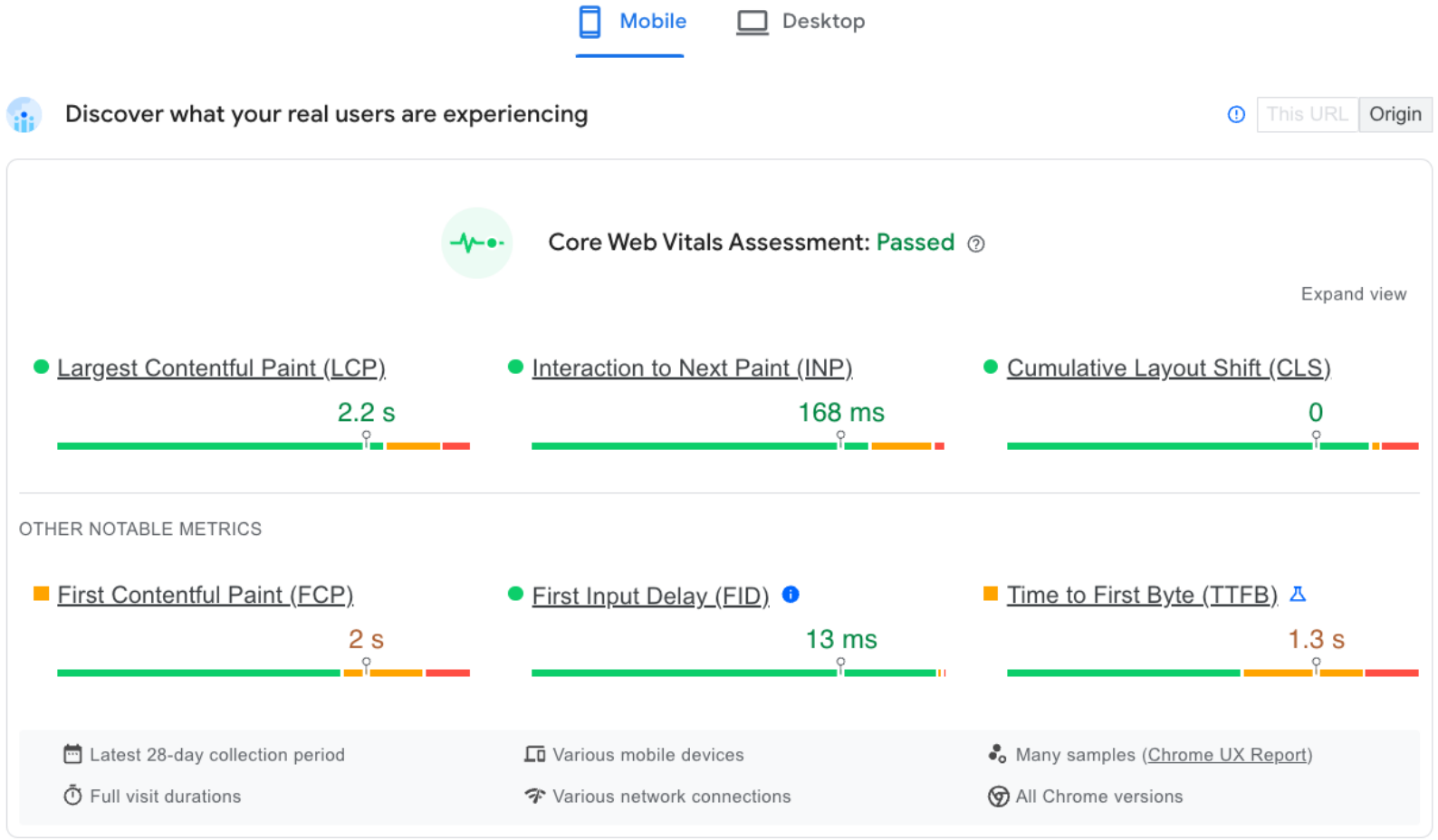
ANGULAR
ARCHITECTS

PageSpeed Insights vs Chrome Lighthouse

- PageSpeed Insights
- **Real** user data & **Lighthouse** lab results
- Go to pagespeed.web.dev & enter URL
- Test is being run on Google servers
- Performance, new since 2023
  - Accessibility
  - Best Practices
  - SEO

- Chrome Lighthouse extension
- Open URL in your browser
  (run in incognito mode, close other Apps)
- Open Lighthouse tab → Analyze
- Performance, other tests
  - Accessibility
  - Best Practices
  - SEO
  - **PWA**

PageSpeed Score – Real Users (Origin)



PageSpeed Score – Lab Data (URL)

Diagnose performance issues

90	87	100	86
Performance	Accessibility	Best Practices	SEO

90

Performance

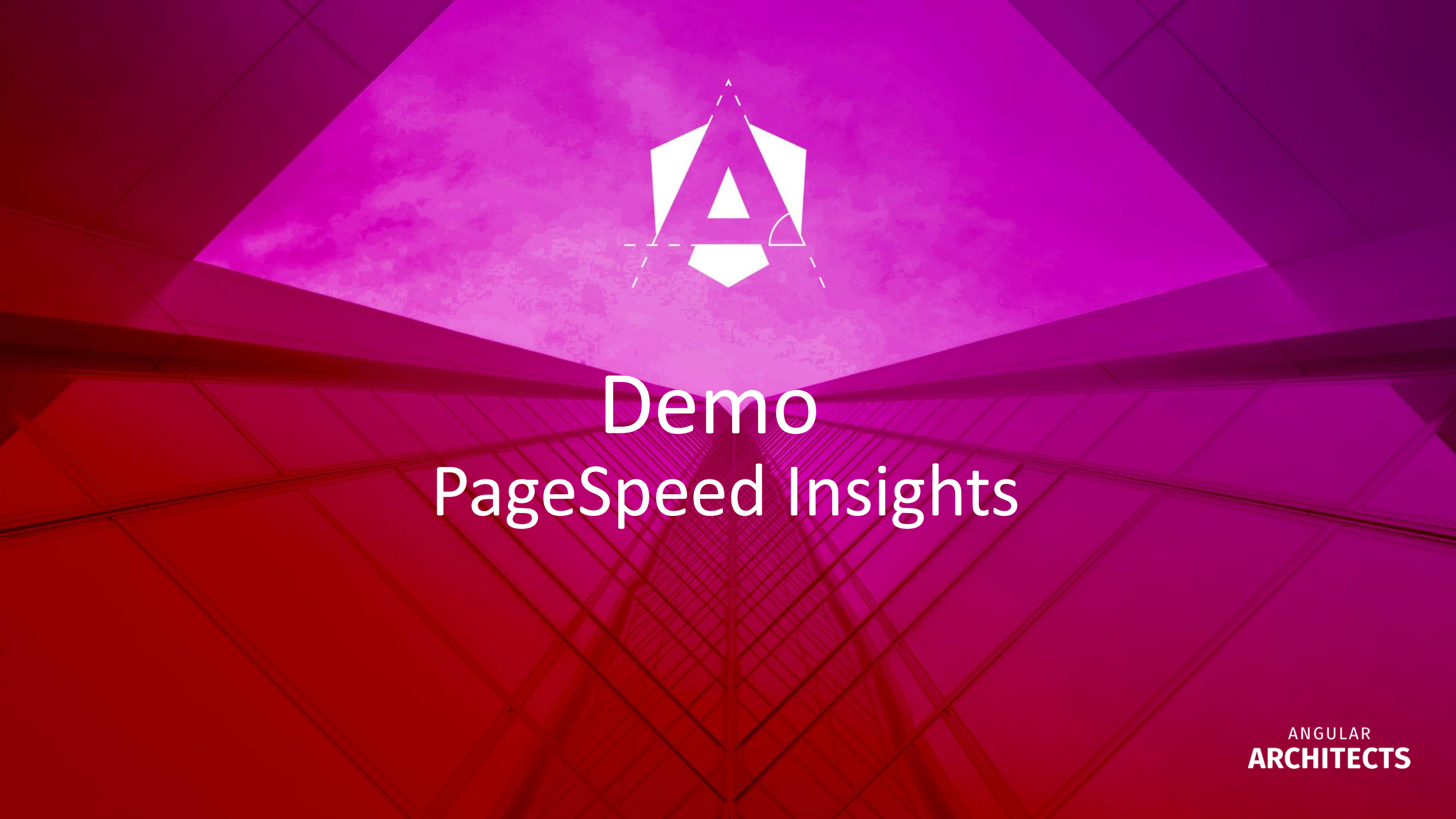
Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS Expand view

● First Contentful Paint 1.7 s	■ Largest Contentful Paint 3.0 s
■ Total Blocking Time 200 ms	● Cumulative Layout Shift 0.007
● Speed Index 2.9 s	

Captured at Apr 22, 2024, 8:22 PM GMT+2
Initial page load
Emulated Moto G Power with Lighthouse 11.5.0
Slow 4G throttling
Single page session
Using HeadlessChromium 122.0.6261.94 with lr



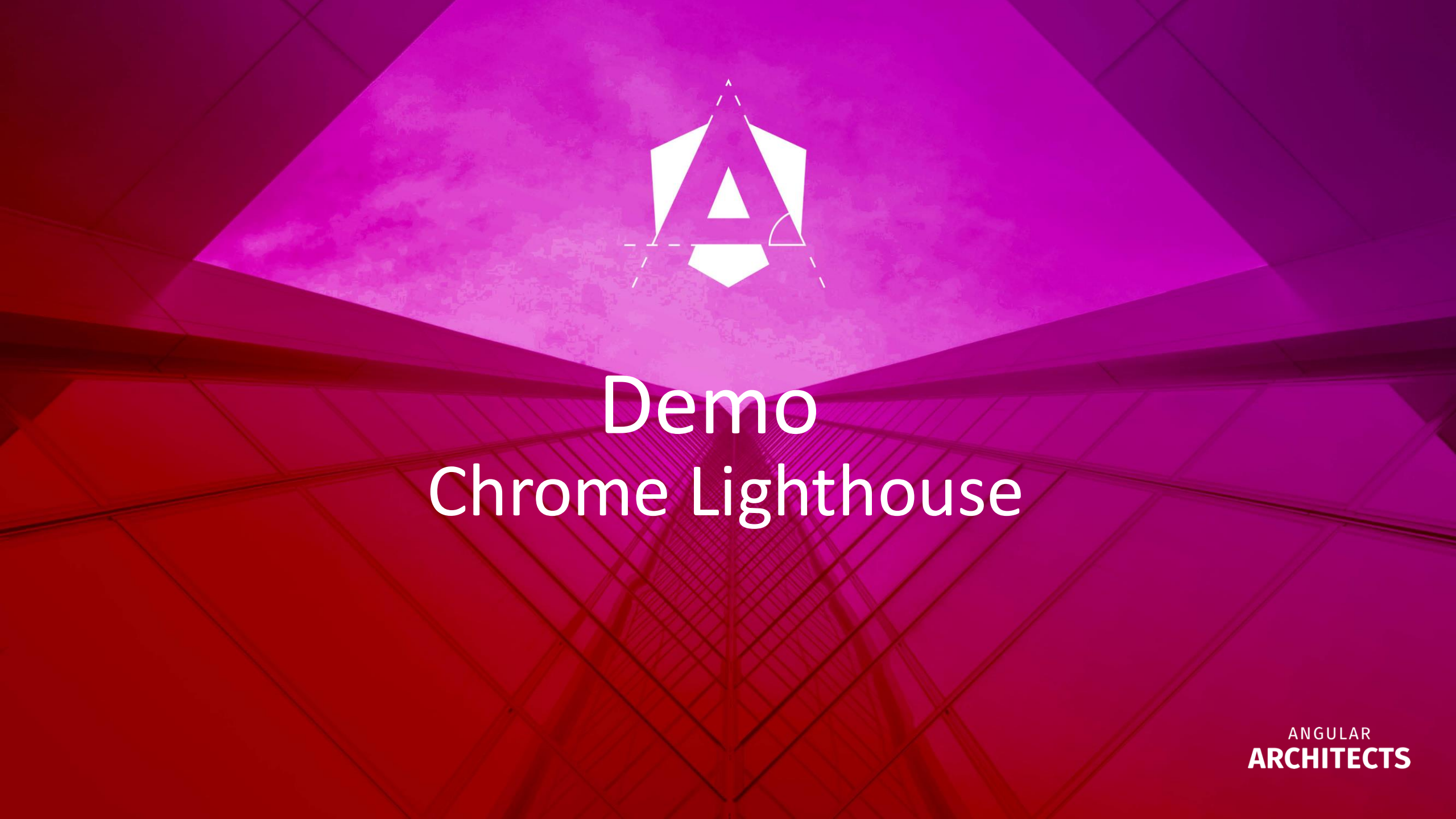
Demo
PageSpeed Insights

ANGULAR
ARCHITECTS

PageSpeed Insights & Chrome Lighthouse

What's being measured?

- Time to First Byte (TTFB)
- **First Contentful Paint (FCP)**
- **Speed Index** (originally by WebPageTest)
- **Largest Contentful Paint (LCP)**
- Time to Interactive (TTI)
- **Total Blocking Time (TBT)** → TTI - FCP
- **Cumulative Layout Shift (CLS)**
- First Input Delay (FID)
- Interaction to Next Paint (INP) → new!



Demo
Chrome Lighthouse

ANGULAR
ARCHITECTS

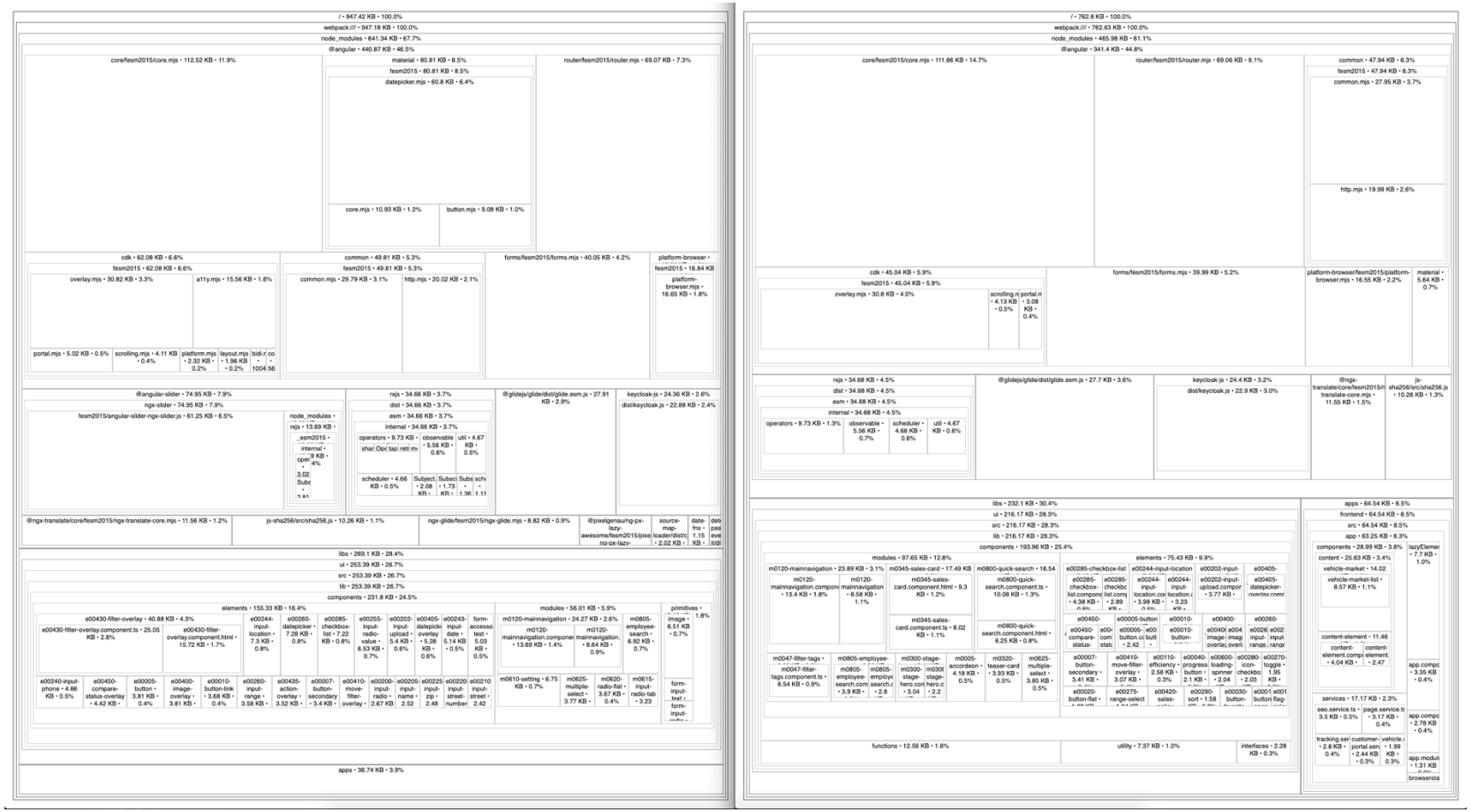
Build Analysis Tools

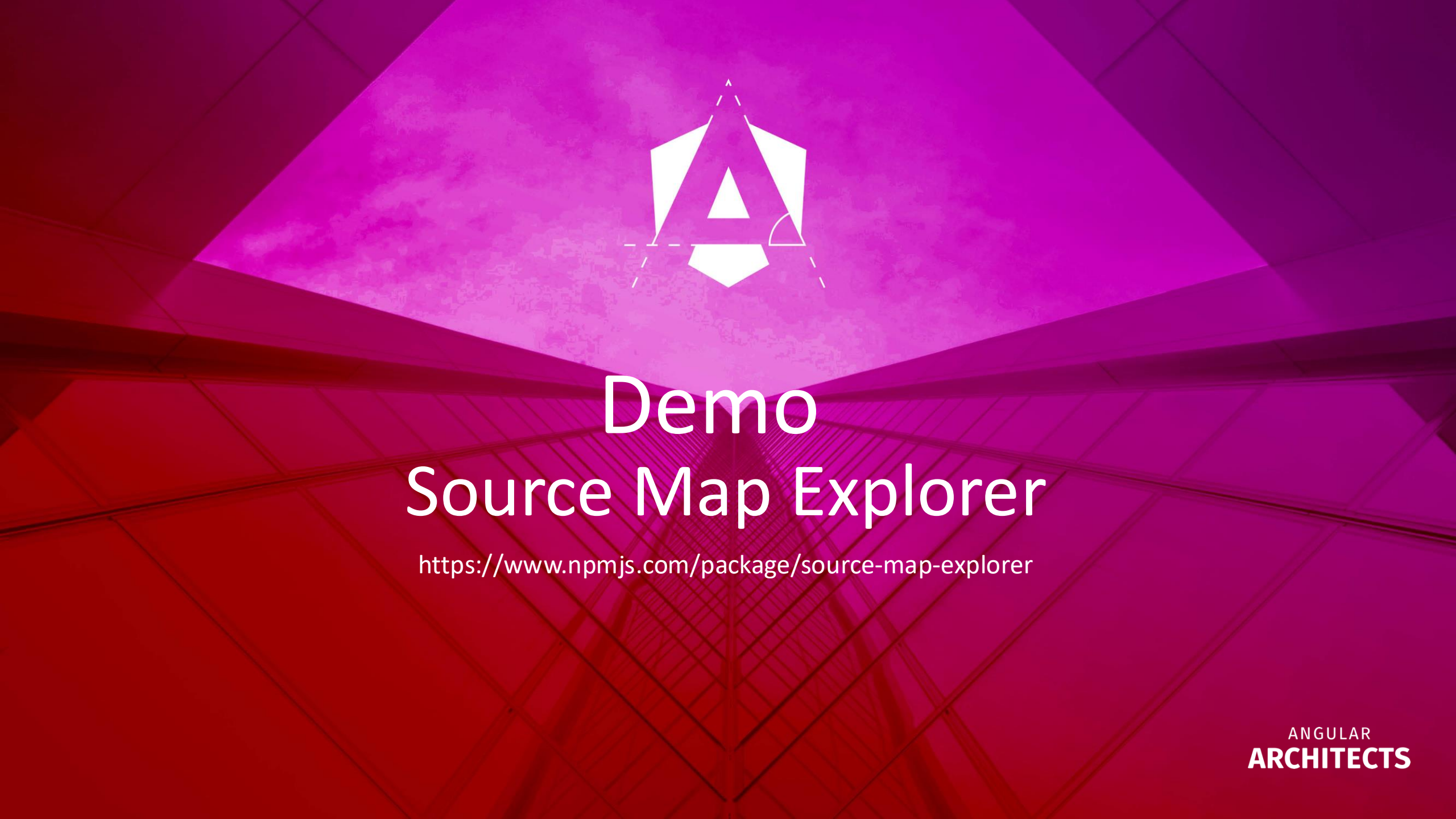
ANGULAR
ARCHITECTS

Source Map Explorer

- Needs generated source maps to work
  - Either set in build options (angular.json)
  - Or just use the build flag "--source-map"
- Analyzes a single js file or whole bundle / build
  - main bundle
  - or lazy loading bundles
- Determines where each byte in your code comes from
- Shows a black/white treemap visualization of build size

Source Map Explorer





Demo
Source Map Explorer

https://www.npmjs.com/package/source-map-explorer

ANGULAR
ARCHITECTS

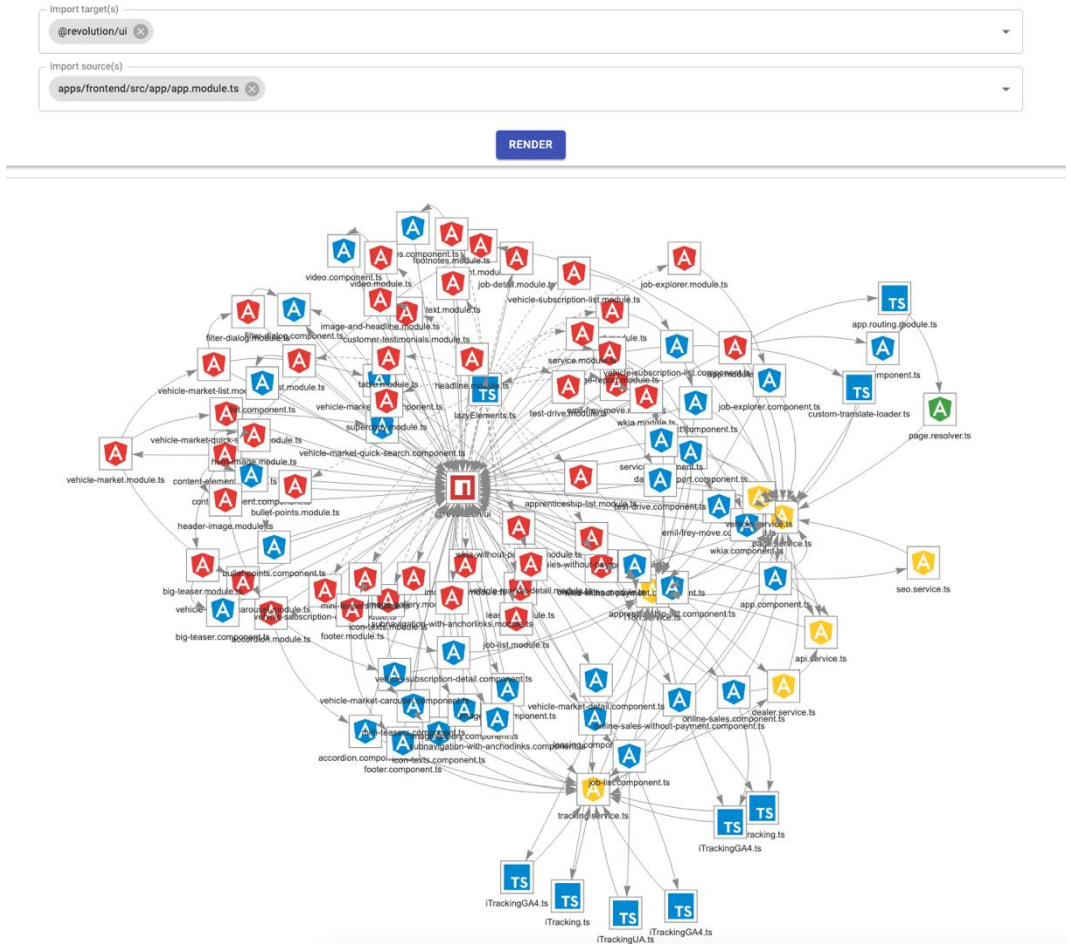
Import Graph Visualizer

– A development tool for filtering and visualizing import paths within a JavaScript/TypeScript application

– Allows filtering import paths by source & target modules

– Allows zooming in to a limited subsection of your app, which will likely be easier to analyze than the entire app

Import Graph Visualizer



Import Graph Visualizer – How To

– npx import-graph-visualizer --entry-points path/to/entry/module --ts-config path/to/tsconfig

– e.g. npx import-graph-visualizer --entry-points src/main.ts --ts-config tsconfig.app.json

Demo
Import Graph Visualizer

https://github.com/rx-angular/import-graph-visualizer

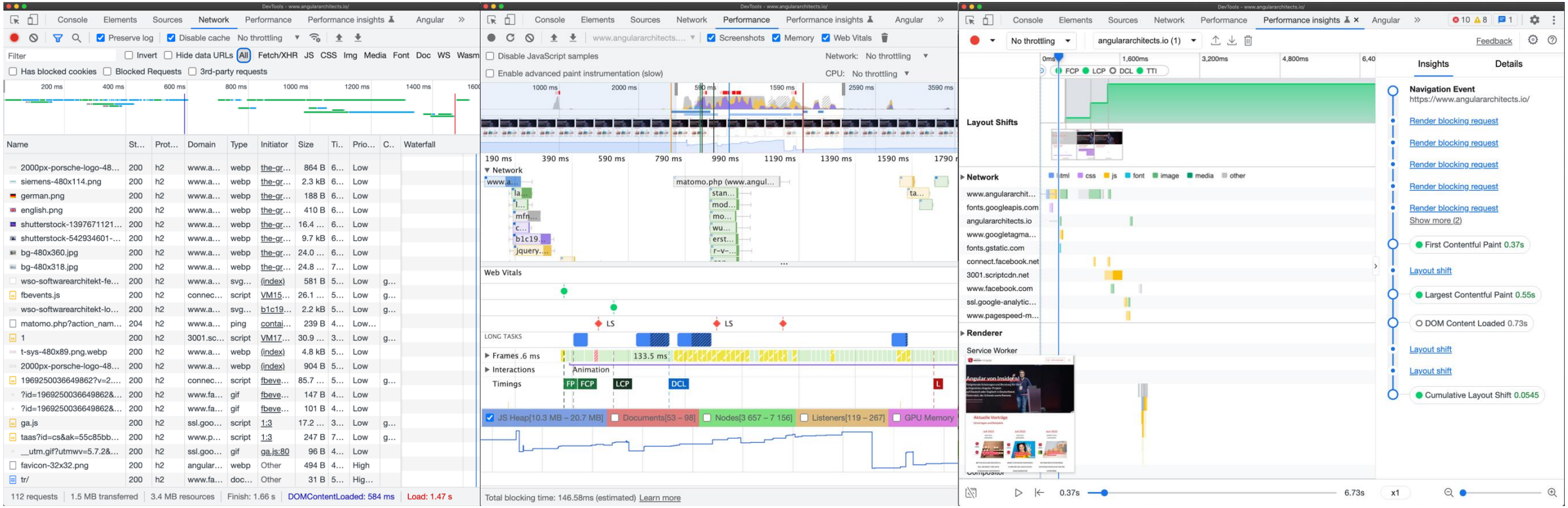
ANGULAR
ARCHITECTS

Google Chrome DevTools

- The Chrome DevTools are not only used for
  - Styling (Elements)
  - Debugging (Console)

- But also for Performance
  - Network
  - Performance
  - Memory
    - memory heap comparison
  - Performance Insights (Beta!)

Google Chrome DevTools



Demo
Chrome DevTools

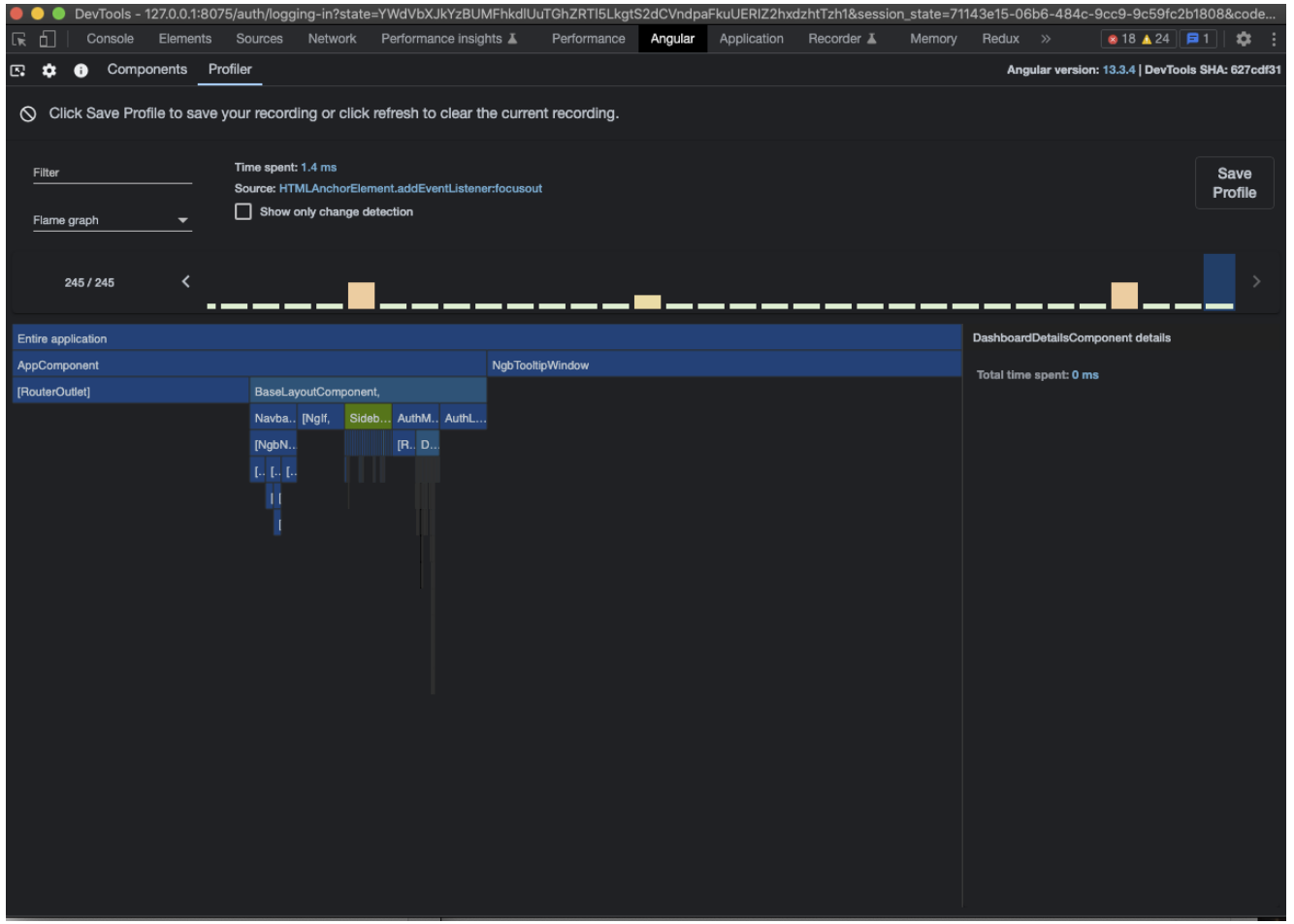
ANGULAR
ARCHITECTS

Angular DevTools Profiler

- Angular DevTools extension can be added to Chrome
  - https://chromewebstore.google.com/detail/angular-devtools/ienfalfjdbdpebioblfackkekamfmbnh
  - Features a **Component Tree** to inspect the components
  - **Profiler**
  - And the newly added **Injector Tree** (Dependency Injection)

- Profiler shows individual change detection (CD) cycles
  - What triggered CD
  - How much time it took executing CD

Angular DevTools Profiler



Demo
Angular DevTools Profiler

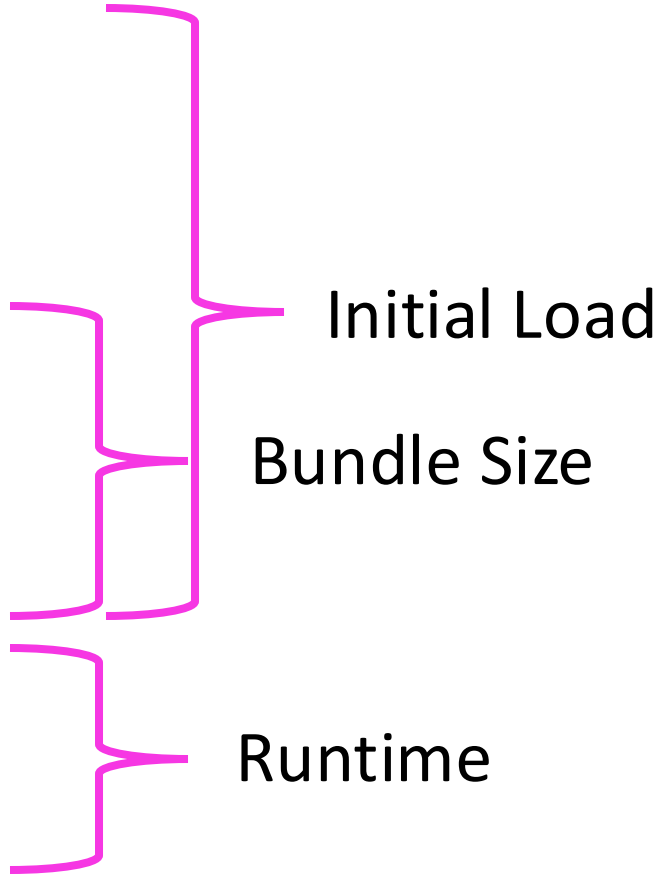
ANGULAR
ARCHITECTS

Audit Tools

- PageSpeed & Lighthouse
- WebPageTest.org

- Source Map Explorer
  - Webpack Bundle Analyzer
  - Vite Bundle Visualizer
- Import Graph Visualizer

- Chrome DevTools
- Angular DevTools Profiler



References

- Google Web Dev
  - https://pagespeed.web.dev
  - https://web.dev/metrics/
- Improving Load Performance - Chrome DevTools 101
  - https://www.youtube.com/watch?v=5fLW5Q5ODiE
- How to analyze your JavaScript bundles
  - https://www.youtube.com/watch?v=MxBCPc7bQvM

ANGULAR
ARCHITECTS

ANGULAR
ARCHITECTS

Initial Load Performance Assets & Build

Alexander Thalhammer | @LX_T

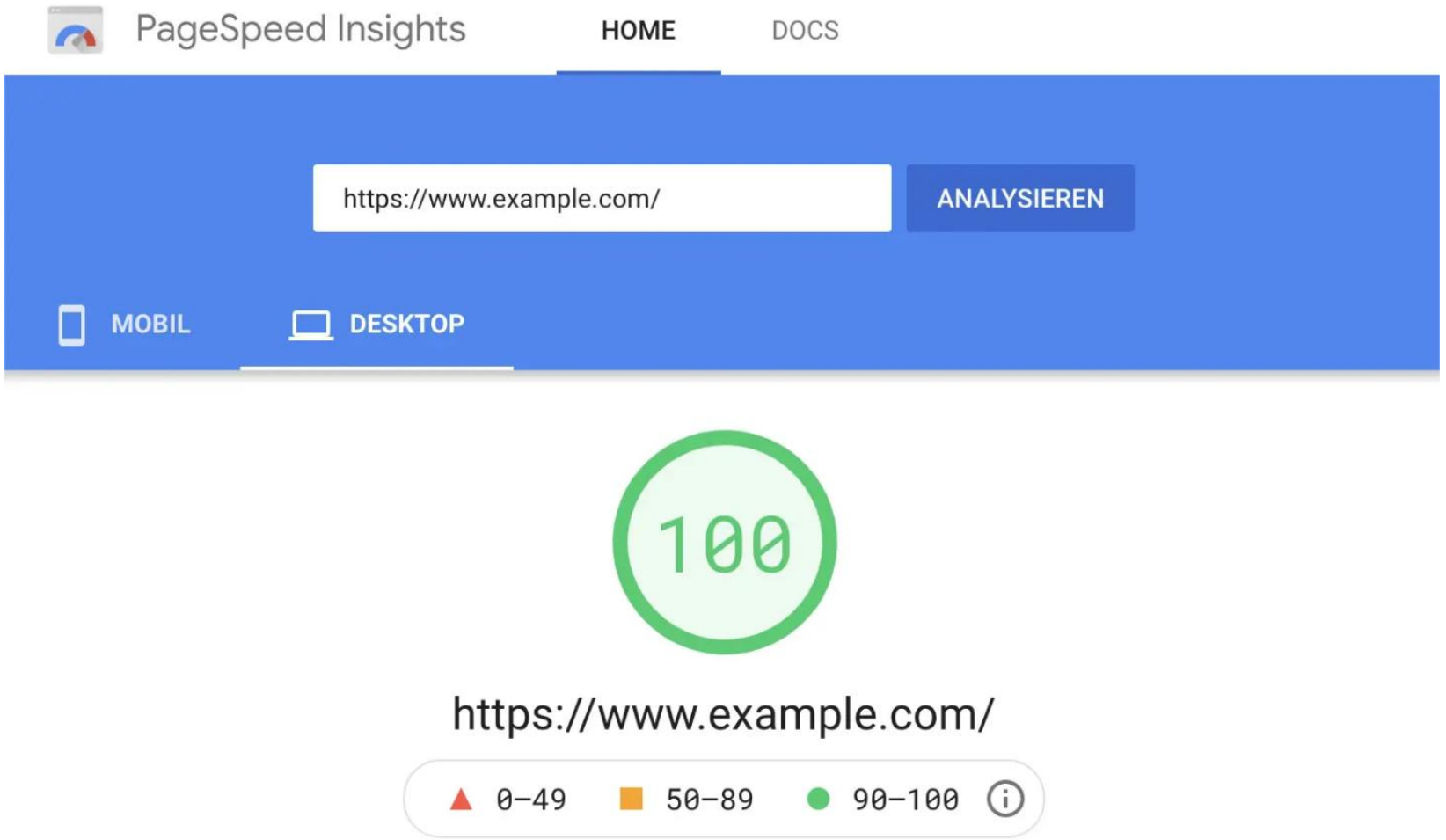
Outline - Initial Load Performance

- Assets & Build
- Lazy Loading
- Deferrable Views
- SSR & SSG

Outline - Assets & Build

- Use web performance best practices
- Use NgOptimizedImage (since NG 14.2.0)
- Use build optimization correctly
- Avoid large 3rd party libs / CSS frameworks

Web Performance Best Practices



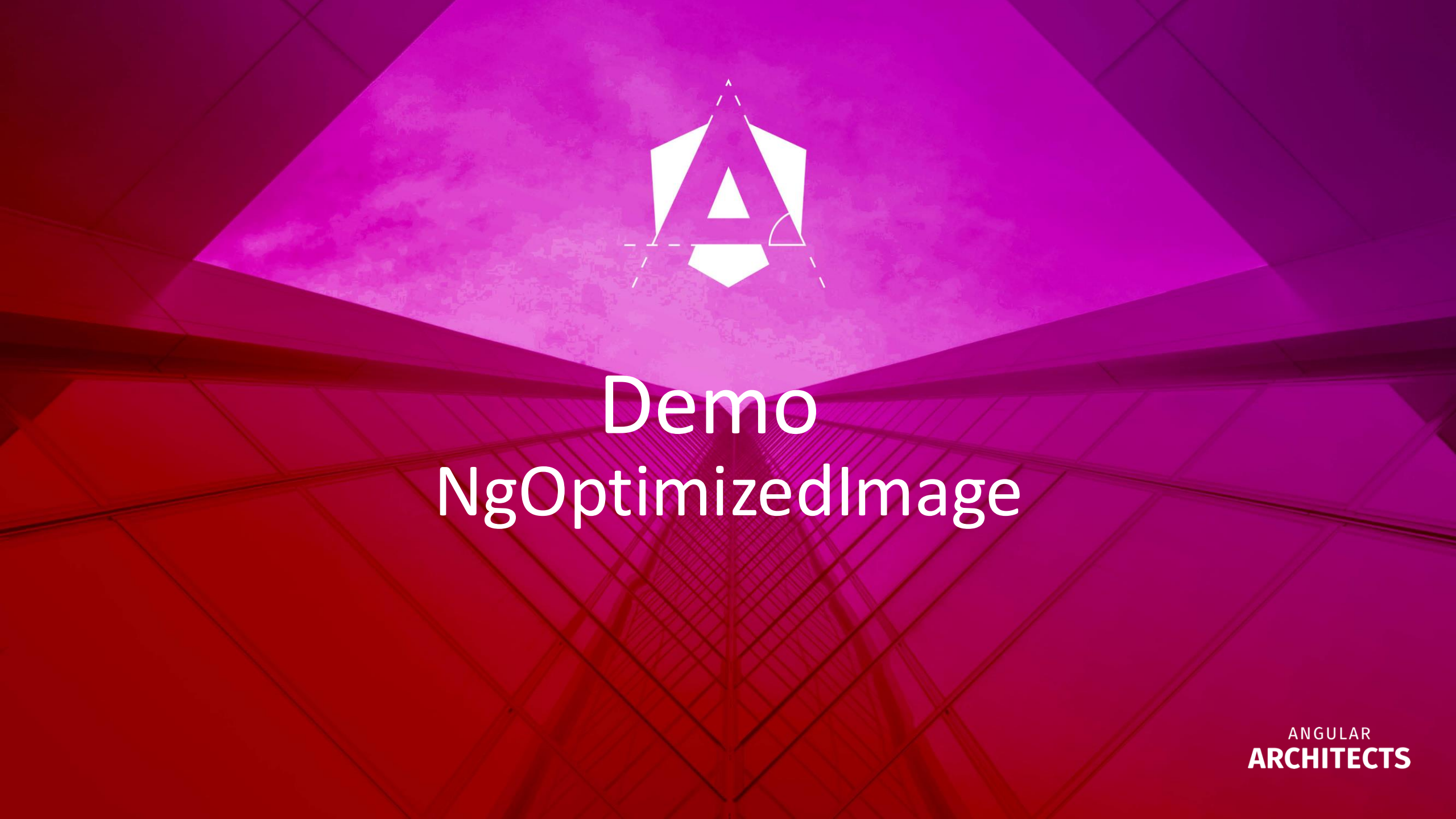
Use web performance best practices

Solutions:

- *Images not optimized → Use .webp, .avif or .svg*
- *Images not properly sized → Use srcsets*
- *Too large assets, too many assets → Clean up & lazy load assets*
- *Unused JS code or CSS styles → Clean up & lazy loading / deferring*
- *Slow server infrastructure → HTTP/3, CDN*
- *Caching not configured correctly → Configure it*
- *Compression not configured correctly → Brotli or Gzip*
- *…*

Use NgOptimizedImage (since NG 14.2.0)

- Problem: *Lighthouse or PageSpeed report image errors / warnings*
- Identify: Lighthouse & PageSpeed Insights / WebPageTest or DevTools
- Solution: Use NgOptimizedImage's ngSrc instead of src attribute
  - Takes care of intelligent lazy loading
  - Prioritization of critical images ("above-the-fold")
  - Also creates srcset & sizes attributes (for responsive sizes, since NG 15.0.0)
  - Also supports high-resolution devices ("Retina images")



Demo
NgOptimizedImage

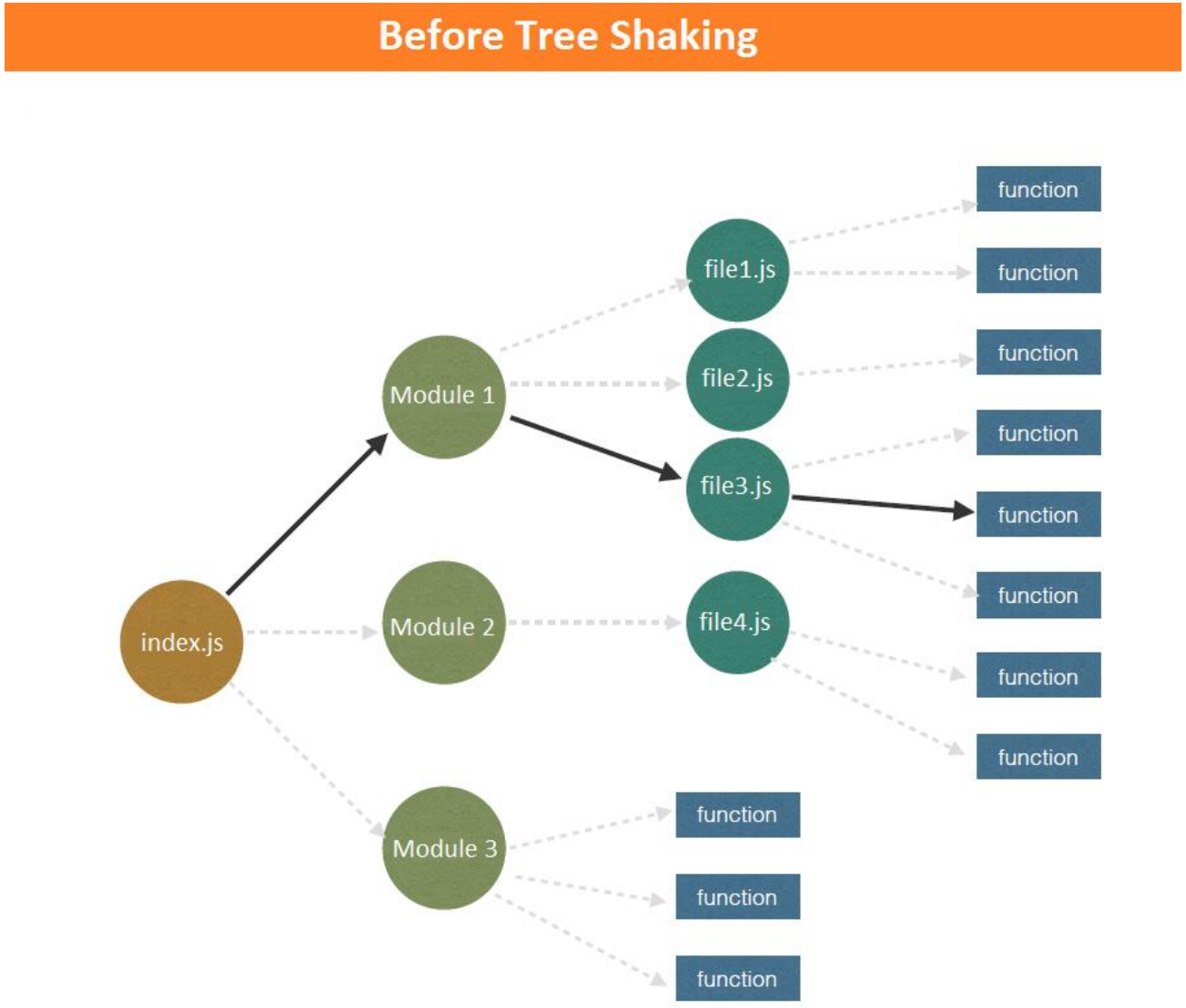
ANGULAR
ARCHITECTS

Build optimization

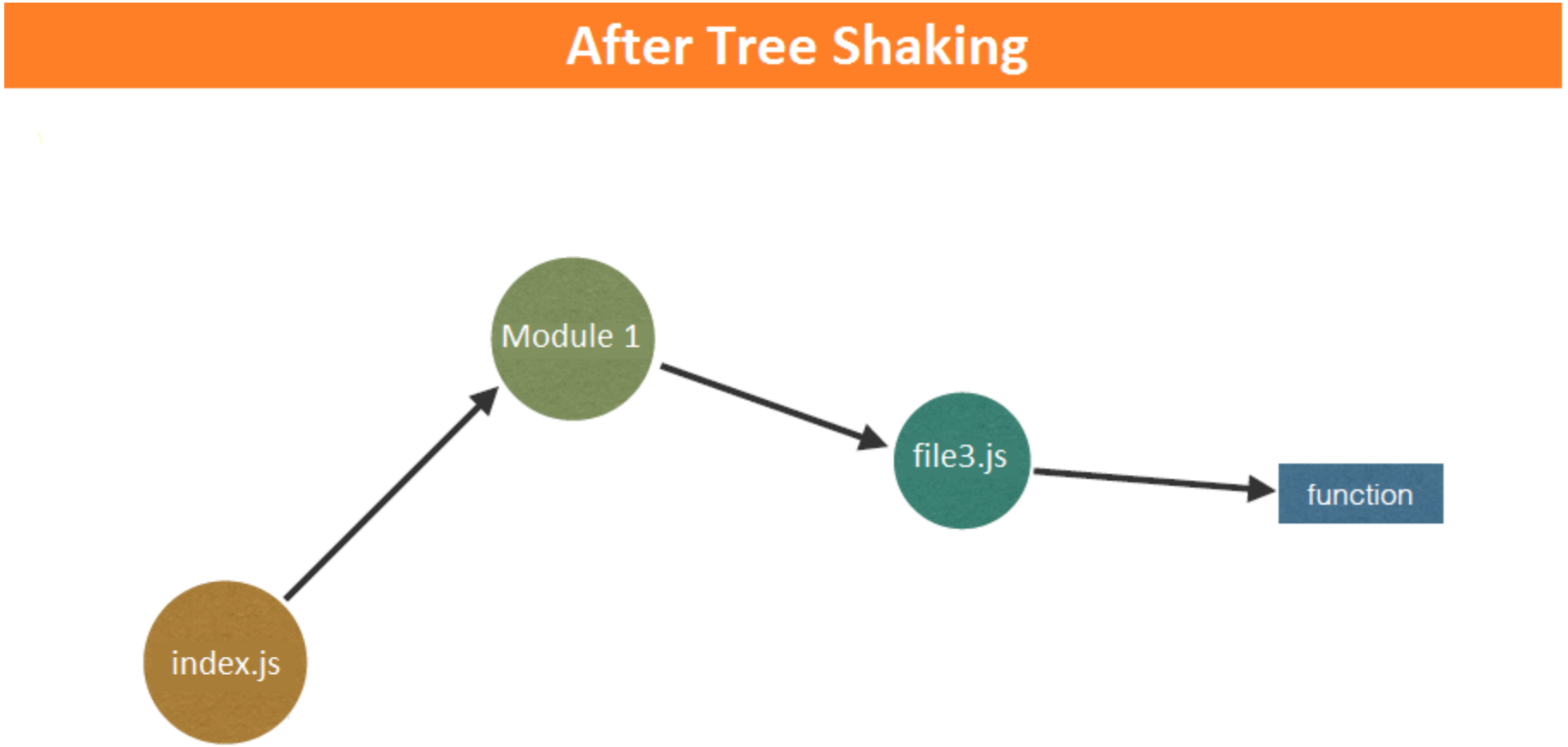
Advantages of Angular Ivy (since V9)

- Angular ViewEngine itself was not tree-shakable
- Default since NG 10, for libs default since NG 12
- AOT per default → You don't need to include the compiler!
- Ivy also does a lot of under the hood optimization
- Tools can easier analyse the code
  - Remove unneeded parts of frameworks
  - Called Tree Shaking
    - Also 3rd party and
    - Even our own libs

Tree Shaking



Tree Shaking



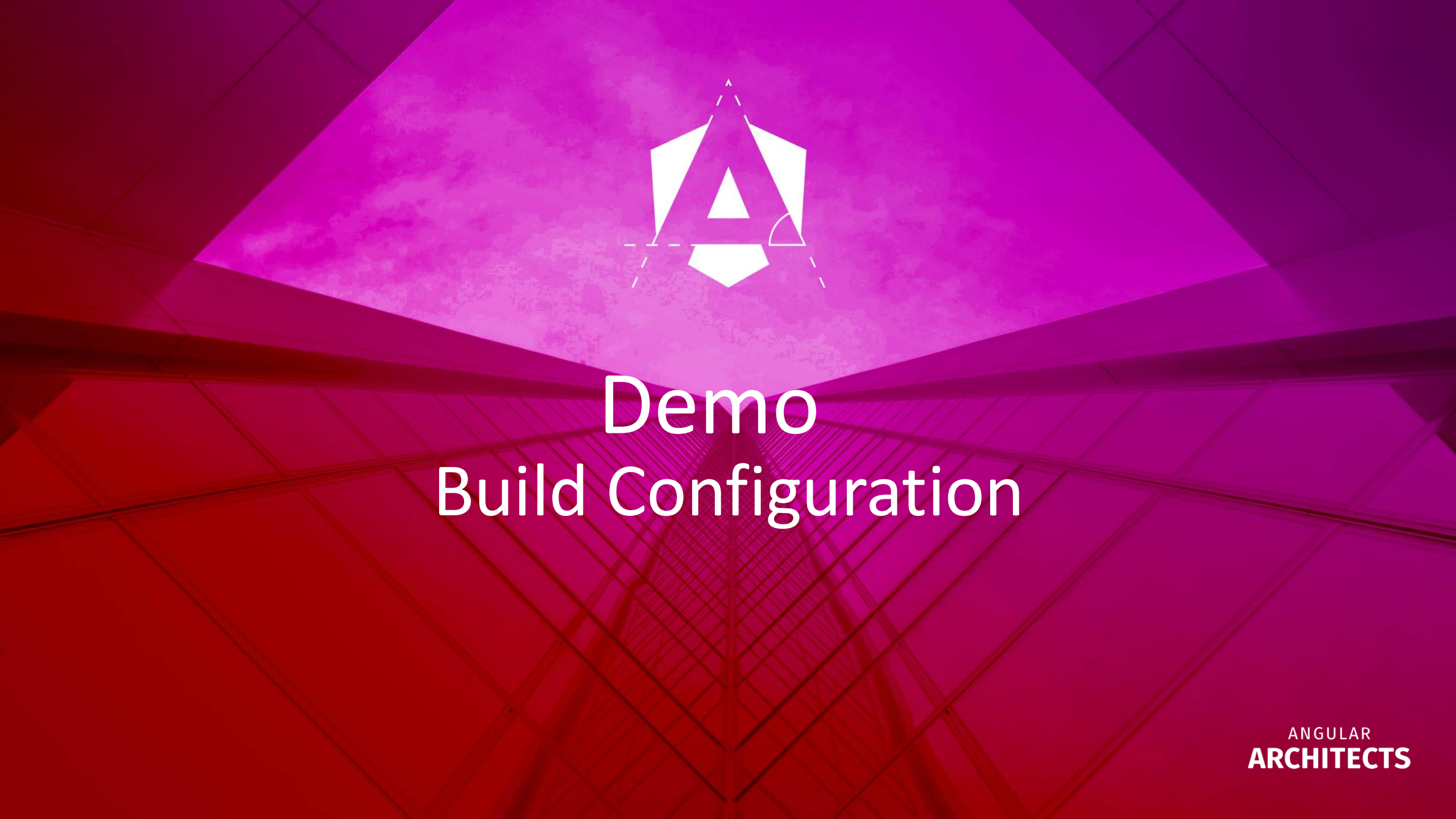
Use Build Optimization

Solution:

- Use production build
  - ng b(uild) (--c production)

- Set up angular.json (or project.json) correctly

```
"production": {
  "buildOptimizer": true,
  "optimization": true,
  "budgets": [
```



Demo
Build Configuration

ANGULAR
ARCHITECTS

Avoid large 3rd party libs / CSS frameworks

- Problem: *Importing large 3rd party libraries that are not treeshakable*
  - *moment*
  - *lodash*
  - *charts*
  - *…*
- Identify: Source Map Analyzer or Webpack Bundle Analyzer
- Solution: Remove or replace that lib / framework
  - *moment → luxon, day.js or date-fns*
  - *lodash → lodash-es*
  - *…*

Demo
Large Libs

ANGULAR
ARCHITECTS

Recap

- Use web performance best practices
- **Use NgOptimizedImage (since NG 14.2.0)**
- **Use build optimization correctly**
- Avoid large 3rd party libs / CSS frameworks

References

- Optimize the bundle size of an Angular application
  - https://www.youtube.com/watch?v=19T3O7XWJkA

- Angular Docs
  - NgOptimizedImage
  - NG Build

ANGULAR
ARCHITECTS

ANGULAR
ARCHITECTS

Initial Load Performance
Lazy Loading

Alexander Thalhammer | @LX_T

Outline - Initial Load Performance

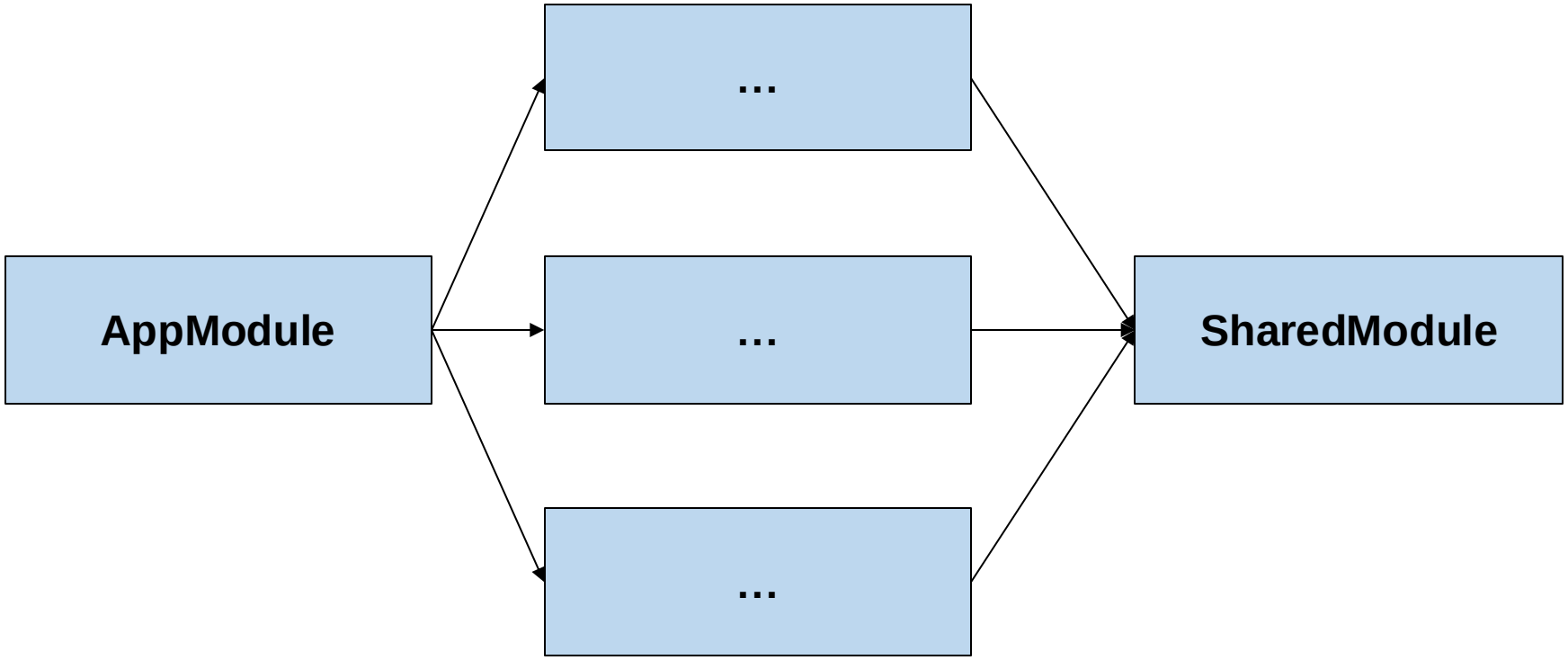
- Assets & Build
- Lazy Loading
- Deferrable Views
- SSR & SSG

Outline - Lazy Loading & Deferrable Views

- Lazy Loading via modules / features
  - 2 most common pitfalls and their solutions
- Lazy Loading via standalone components
- Preloading
  - PreloadAllModules
  - Other strategies
- Lazy Loading without the router (a bit complicated)
- Lazy Loading below the fold (very, very complicated)
- Deferring (brand new in NG 17, very lean, replaces last 2)

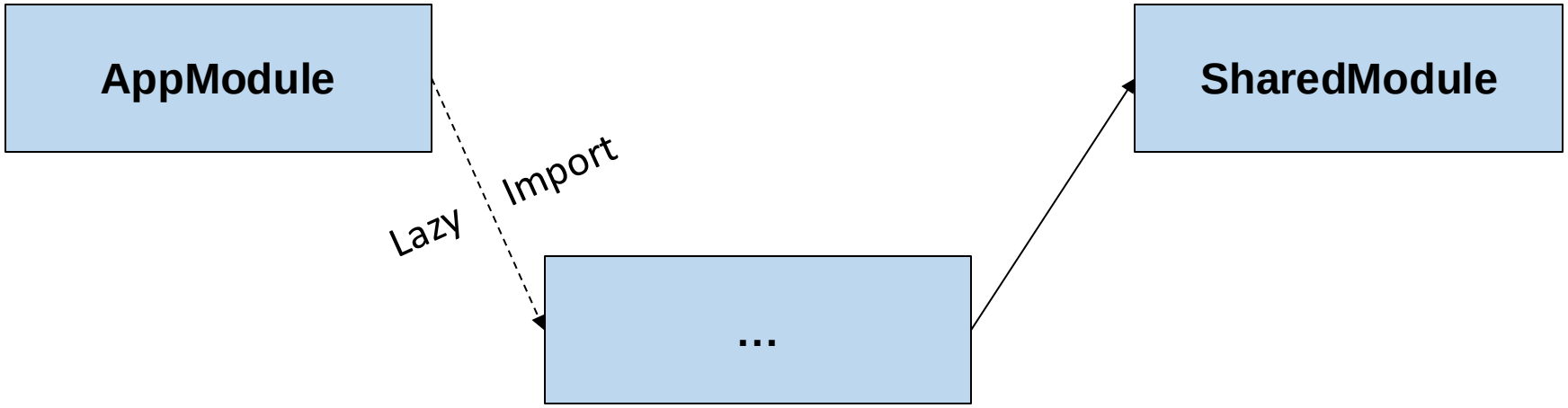
Lazy Loading (Angular 5)

Angular Module | Feature Structure



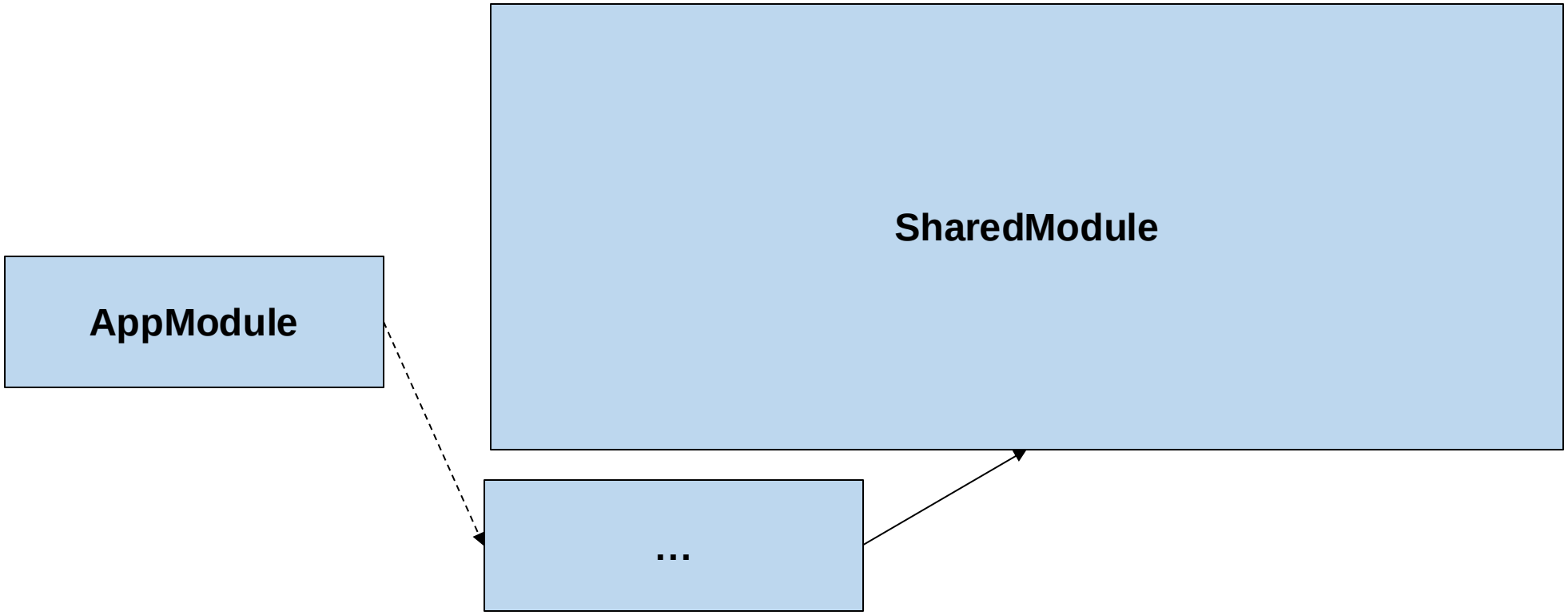
Root Module **Feature Modules** **Shared Module**

Angular Lazy Loading – Theory



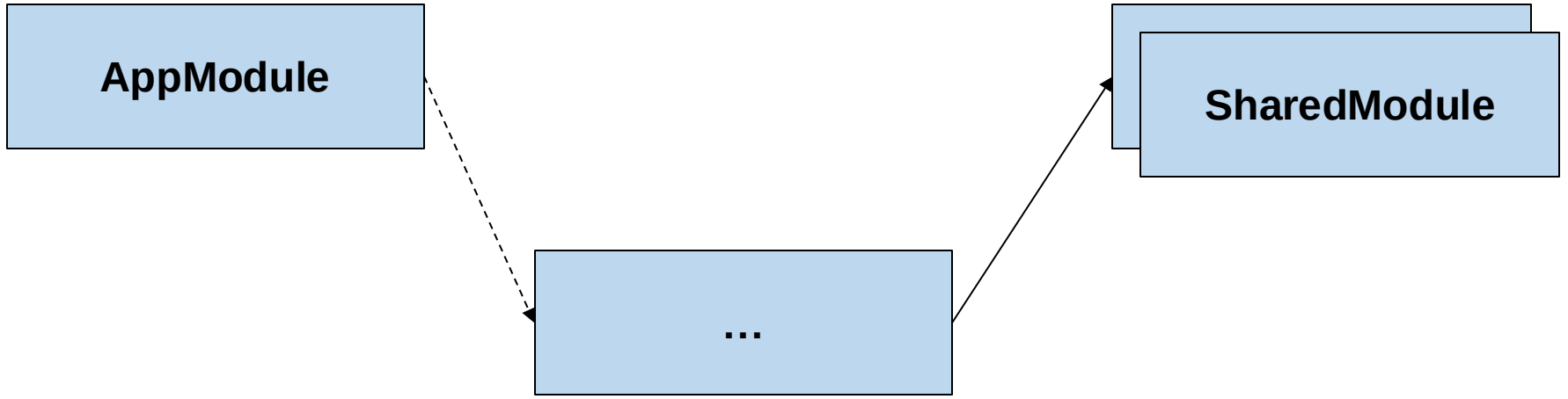
Root Module **Feature Modules** **Shared Module**

Angular Lazy Loading – Common Pitfall



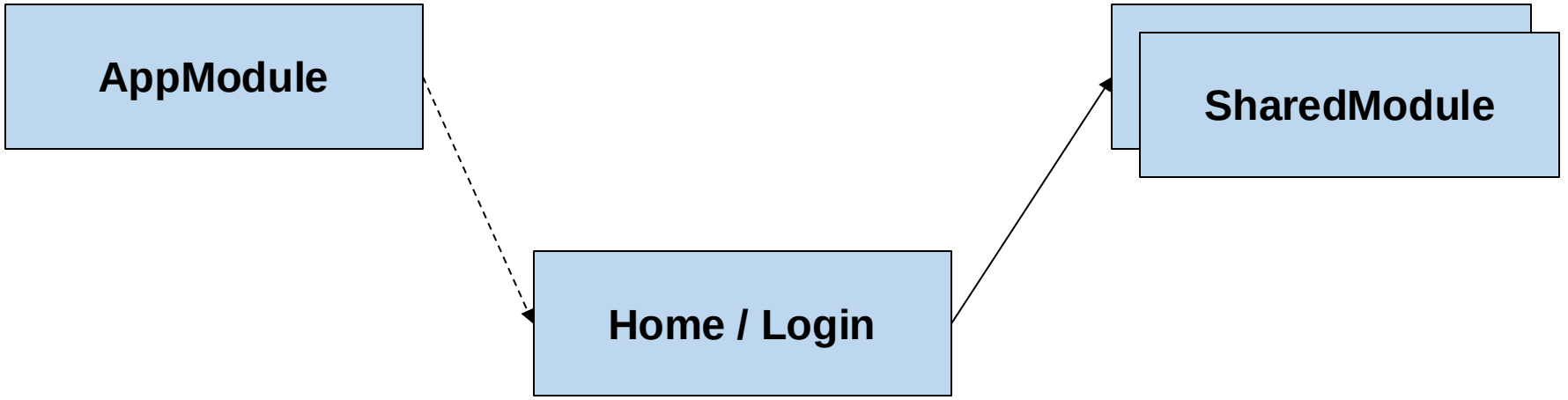
Root Module **Feature Modules** **Huge Shared Module**

Angular Lazy Loading - Solution



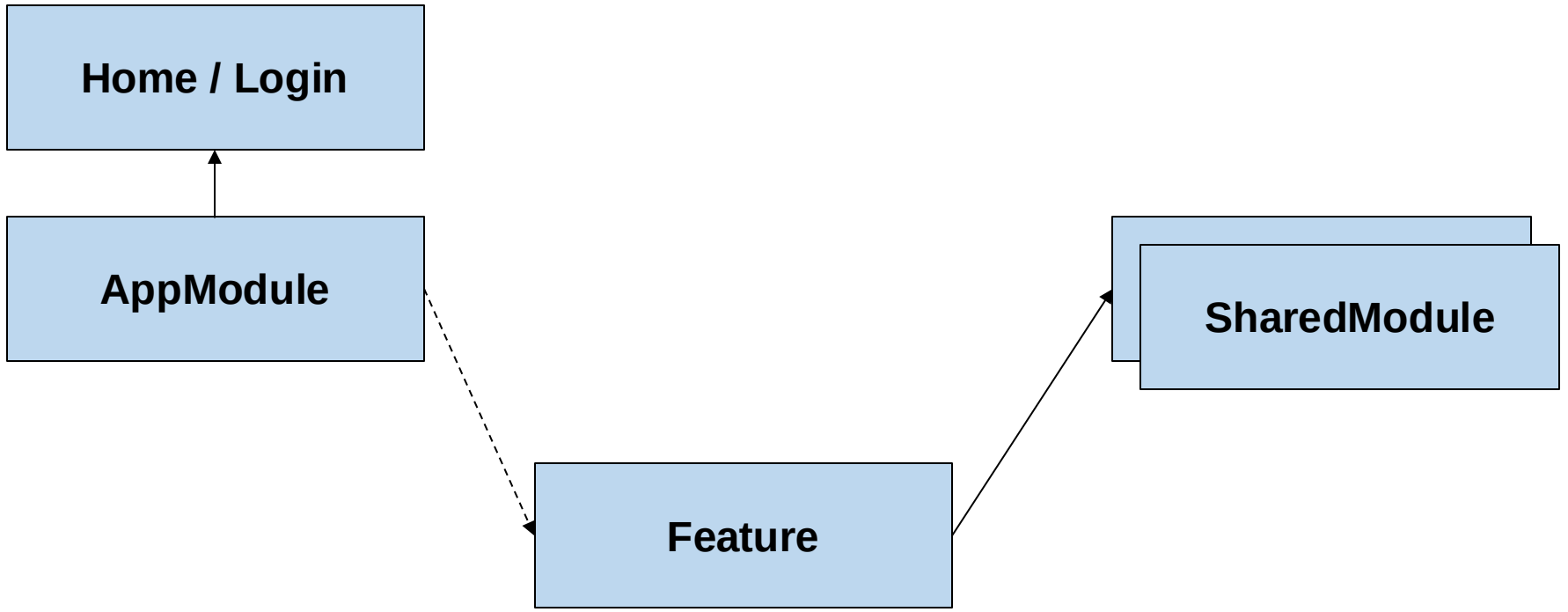
Root Module **Feature Modules** **Small Shared Modules**

Angular Lazy Loading – Another Pitfall



Root Module **Feature Modules** **Small Shared Modules**

Angular Lazy Loading – Solution



Root Module **Feature Modules** **Small Shared Modules**

App Routes with Lazy Loading

```
export const appRoutes: Routes = [
    {
        path: 'home',
        component: HomeComponent
    },
    {
        path: 'flights_module',
        loadChildren: () => import('./flights/flights.module')
                              .then((m) => m.FlightsModule)
    },
    {
        path: 'flights_standalone',
        loadChildren: () => import('./flights/flights.routes')
    }
];
```

Routes for "lazy" Feature

```
export const flightsRoutes: Routes = [
    {
        path: 'flight-search',
        component: FlightSearchComponent,
        […]
    },
    […]
}

export default flightsRoutes; // for standalone
```

flights/ flight-search

Triggers Lazy Loading w/ loadChildren

Demo
Lazy Loading Module

ANGULAR
ARCHITECTS

Lazy Loading with standalone components

```
export const appRoutes: Routes = [
    {
        path: 'home',
        component: HomeComponent
    },
    {
        path: 'charts',
        loadComponent: () => import('./charts/charts.component')
                             .then((c) => c.ChartsComponent)
    }
];
```

Demo
Lazy Loading Standalone

ANGULAR
ARCHITECTS

What about services?

```
…
@Injectable({
  providedIn: 'root'
})
…
```

- When used by 1 lazy loaded module/comp exlusively it will be put into that chunk
- When used by 2 or more lazy loaded modules/comps it will be put into a common chunk
- When used by an eagerly loaded part it will be put into main bundle

Use Lazy Loading a lot

Problem:

- *Loading to much source (libs / components) at startup*
- *Resulting in a big main bundle (and vendor if used)*

Identify:

- Not using lazy loading throughout the App (source code)
- Webpack Bundle Analyzer or
- Source Map Explorer

Lazy Loading

- Lazy Loading means: Load it later, after startup
- Better initial load performance
- But: Delay during execution for loading on demand

Preloading

Preloading

- Once the initial load (the important one) is complete load the lazy loaded modules (before they are even used)

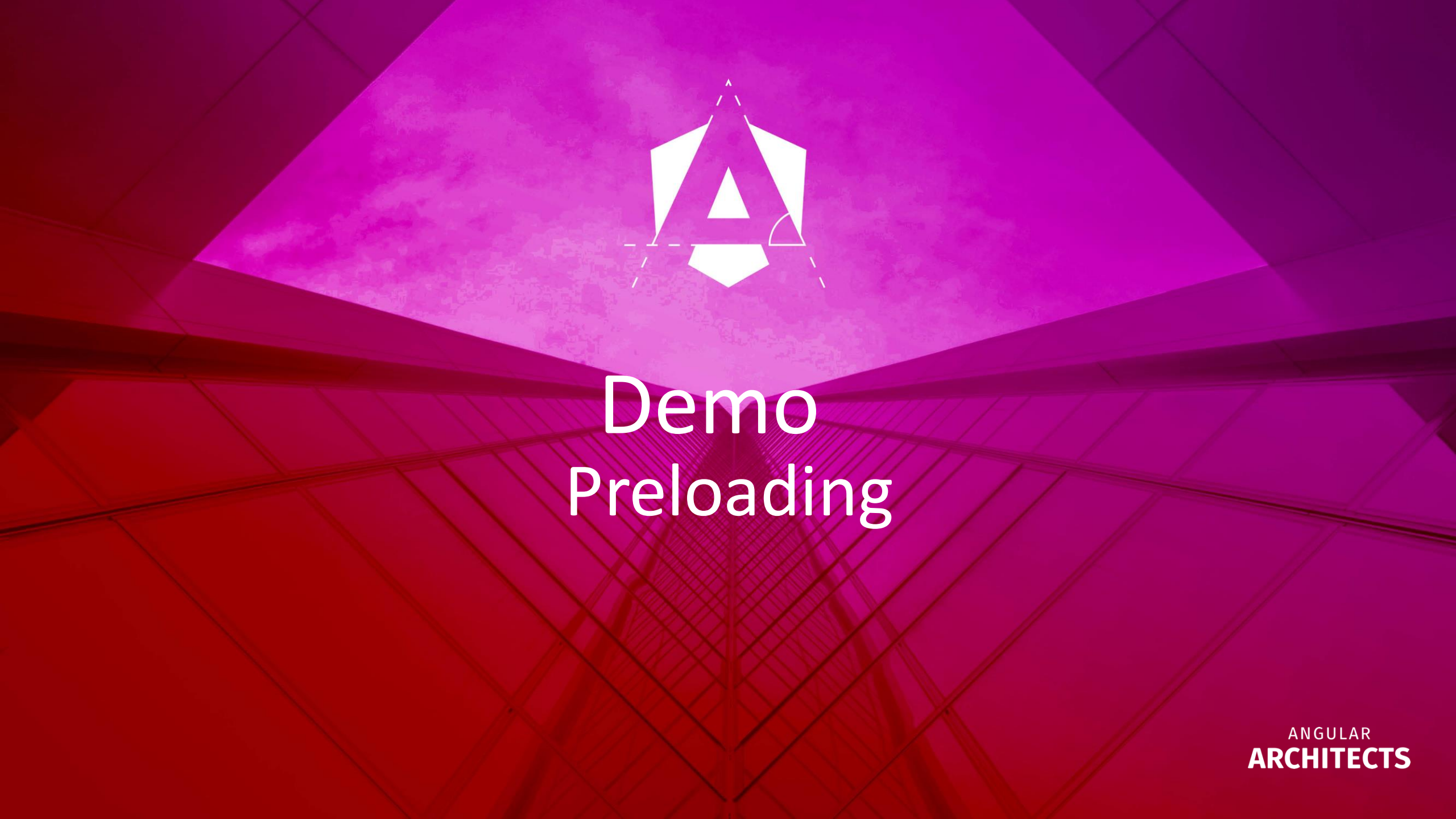
- When module is needed it is available immediately

Activate Preloading (in AppModule)

```
…
imports: [
    […]
    RouterModule.forRoot(
        appRoutes, { preloadingStrategy: PreloadAllModules }
    );
]
…
```

Activate Preloading (in app.config.ts, NG15)

```
…
providers: [
    […]
    provideRouter(
        appRoutes, withPreloading(PreloadAllModules),
    ),
]
…
```



Demo
Preloading

ANGULAR
ARCHITECTS

Intelligent Preloading with ngx-quicklink

```
…
imports: [
    […]
    QuicklinkModule,
    RouterModule.forRoot(
        appRoutes, { preloadingStrategy: QuicklinkStrategy }
    );
]
…
```

https://web.dev/route-preloading-in-angular/

https://www.npmjs.com/package/ngx-quicklink

Demo
Ngx Quicklink

ANGULAR
ARCHITECTS

Or CustomPreloadingStrategy

```
…
imports: [
    […]
    RouterModule.forRoot(
        appRoutes, { preloadingStrategy: CustomPreloadingStrategy }
    );
]
…
```

Use Lazy Loading a lot - but carefully ;-)

Solution:

- Implement lazy loading whereever you can
  - Use lazy loading with the router
    - Modules
    - Components (new since NG15!)
    - Maybe use a CustomPreloadingStrategy if App is very big
  - Use dynamic components

- Use Import Graph Visualizer to detect why things land in main bundle

- **But** don't lazyload the initial feature, because it will be delayed ;-)

- **And** don't destroy lazy loading by (eagerly) loading a huge shared module

Lazy Loading without the router

```
…
    @ViewChild('cnt', { read: ViewContainerRef }) vCR!: ViewContainerRef;
…
    async ngOnInit() {
        const esm = await import('./lazy/lazy.component');
        const lazyComponentRef = this.vCR.createComponent(esm.LazyComponent);
    }
…
```

```
…
    <ng-container #cnt></ng-container>
…
```

Demo
Dynamic Lazy Loading

ANGULAR
ARCHITECTS

Recap

- **Lazy Loading via modules / features**
  - Avoid 2 most common pitfalls and their solutions
- **Lazy Loading via standalone components**
- **Preloading**
  - PreloadAllModules
  - QuicklinkStrategy (ngx-quicklink)

References

- Angular Docs
  - Lazy-loading feature modules

ANGULAR
ARCHITECTS

ANGULAR
ARCHITECTS

Initial Load Performance Deferrable Views

Alexander Thalhammer | @LX_T

Outline - Initial Load Performance

- Assets & Build
- Lazy Loading
- Deferrable Views
- SSR & SSG

Deferrable Views (Angular 17)

- Problem: Lazy Loading without the router and especially Lazy Loading below the fold is rather complicated and inconvenient
- NG17 has the solution in the new template syntax control flow:
- It's called: **Deferrable Views**

Deferrable Views - syntax

```
…
@defer (on viewport) {
  <aa-lazy-component />
} @placeholder {
  <p>Component is loading on viewport.</p>
}
…
```

Deferrable Views - on

- on immediate (default)
- on viewport
- on hover
- on interaction
- on timer(4200ms)

Deferrable Views - when

- specifies an imperative condition as an expression that returns a bool
  - best used: boolean flag

- if the condition returns to false, the swap is not reverted
  - it is a one-time operation

```
…
@defer (when condition) {
  <aa-lazy-component />
}
…
```

Deferrable Views - prefetch

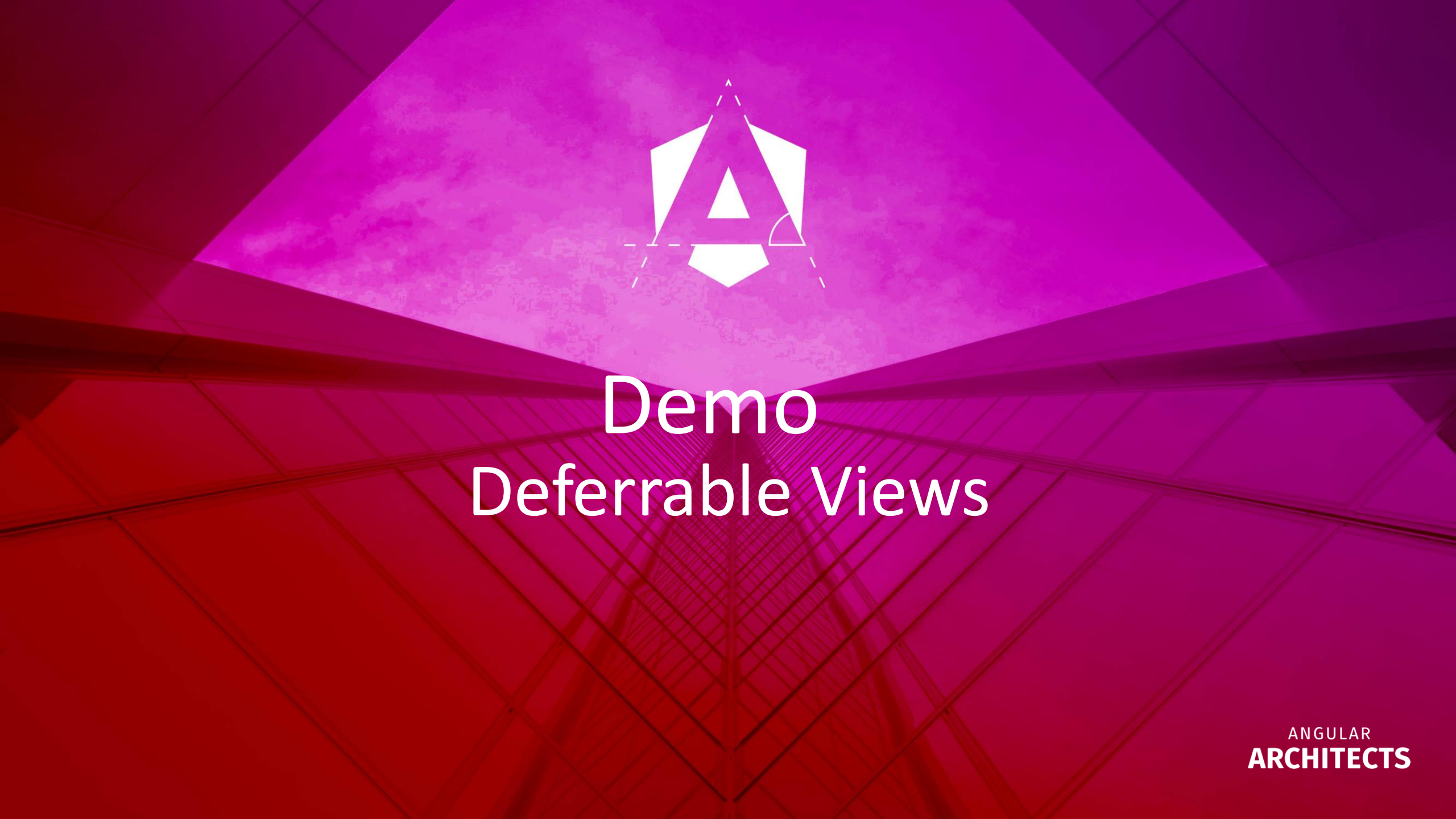
- allows to specify conditions **when prefetching of the dependencies should be triggered**

```
…
@defer (on viewport; prefetch on idle) {
 <aa-lazy-component />
}
 …
```

Deferrable Views - extras

- **@placeholder**
- **@loading**
- **@error**

```
…
@defer (on viewport; prefetch on idle) {
  <aa-lazy-component />
} @placeholder (minimum 500ms) {
  <img width="420" height="420" alt="lazy component placeholder"
src="placeholder.avif" />
} @loading (after 500ms; minimum 1s) {
  <img width="420" height="420" alt="lazy is loading spinner"
src="spinner.avif" />
} @error {
  <p>Why do I exist?</p>
}
…
```



Demo
Deferrable Views

ANGULAR
ARCHITECTS

References

- Angular Docs
  - Lazy-loading feature modules

- Angular Architects Blog
  - Deferrable Views (Blog post)

ANGULAR
ARCHITECTS

ANGULAR
ARCHITECTS

Initial Load Performance
SSR & SSG

Alexander Thalhammer | @LX_T

Outline - Initial Load Performance

- Assets & Build
- Lazy Loading
- Deferrable Views
- SSR & SSG

Outline - SSR & SSG

- Server-Side Rendering
- Hydration
- Prerendering
- Alternative: Use a URL cache

Server-side rendering

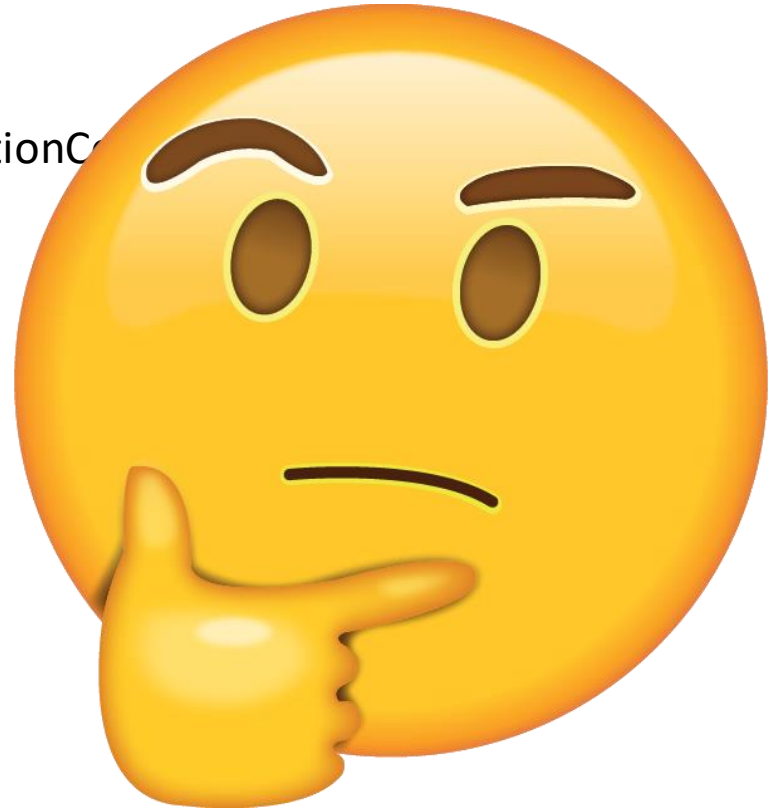
Server-side rendering (SSR)

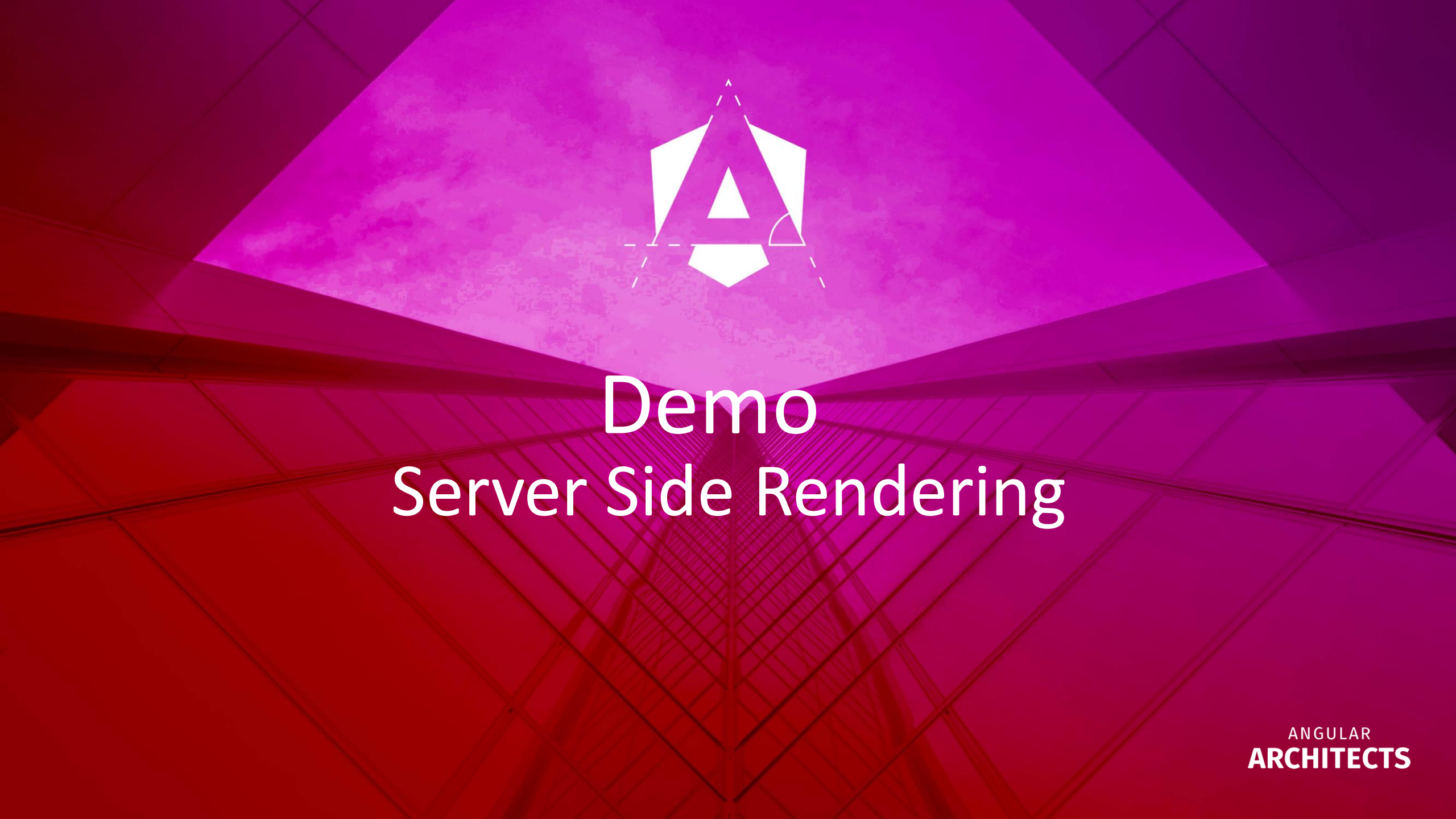
- Problem: *After download rendering on the client takes too much time*
  - *Search Engines may not be abled to index the App correctly*
- Identify: After .js files have been loaded js main thread takes too long
  - Search Engines don’t index correctly
- Solution: Use Angular Universal
  - Page is rendered on the server and then served to the client
  - **But** only useful for public pages (no user login)

Server-Side Rendering (Angular 16)

- New feature called *“non destructive hydration”*

```
export const appConfig: ApplicationC
  providers: [
    provideClientHydration(),
    [...]
  ],
};
```





Demo
Server Side Rendering

ANGULAR
ARCHITECTS

Prerender important routes (SSG)

- Problem: *Server response takes to long cos page has to be rendered*
- Identify : Long server response time when using Universal SSR
- Solution: Prerender the important pages on the server
  - Built-in Angular Universal since V11
  - Then serve them rendered to the user

Hybrid Rendering

- CSR Routes
  - Regular SPA (without SSR)
  - Server serves static files
- SSR Routes
  - Live content + Hydration
  - Server renders the routes
- Pre-rendered routes
  - Built time content + Hydration
  - Server serves built time rendered

Demo
Static Site Generation

ANGULAR
ARCHITECTS

Event Replay (Angular 18)

- Problem: *Clicking and interacting with app before hydration*
- Identify : Long server response time when using Universal SSR
- Solution: Event Replay

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideClientHydration(withEventReplay()),
    [...]
  ],
};
```

Hybrid Rendering (Angular 19)

- CSR Routes
  - Regular SPA (without SSR)
  - Server serves static files
- SSR Routes
  - Live content + Hydration
  - Server renders the routes
- Pre-rendered routes
  - Built time content + Hydration
  - Server serves built time rendered

```
{
  path: 'charts',
  component: ChartsComponent,
  component: RenderMode.Client,
},

{
  path: 'home',
  component: HomeComponent,
  renderMode: RenderMode.Server,
},

{
  path: post',
  component: PostComponent,
  component: RenderMode.Prerender,
},
```

Incremental Hydration (Angular 19)

- ergonomic API
  - known from @defer
- improve performance
  - initial load
  - other CWV
- starting in a few weeks

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideClientHydration(
      withEventReplay(),
      withPartialHydration()
    ),
    [...]
  ],
};
```

hydrate on

- hydrate on immediate (default)
- hydrate on viewport
- hydrate on hover
- hydrate on interaction
- hydrate on timer(4200ms)

hydrate when

- specifies an imperative condition as an expression that returns a bool
  - best used: boolean flag

- if the condition returns to false, the swap is not reverted
  - it is a one-time operation

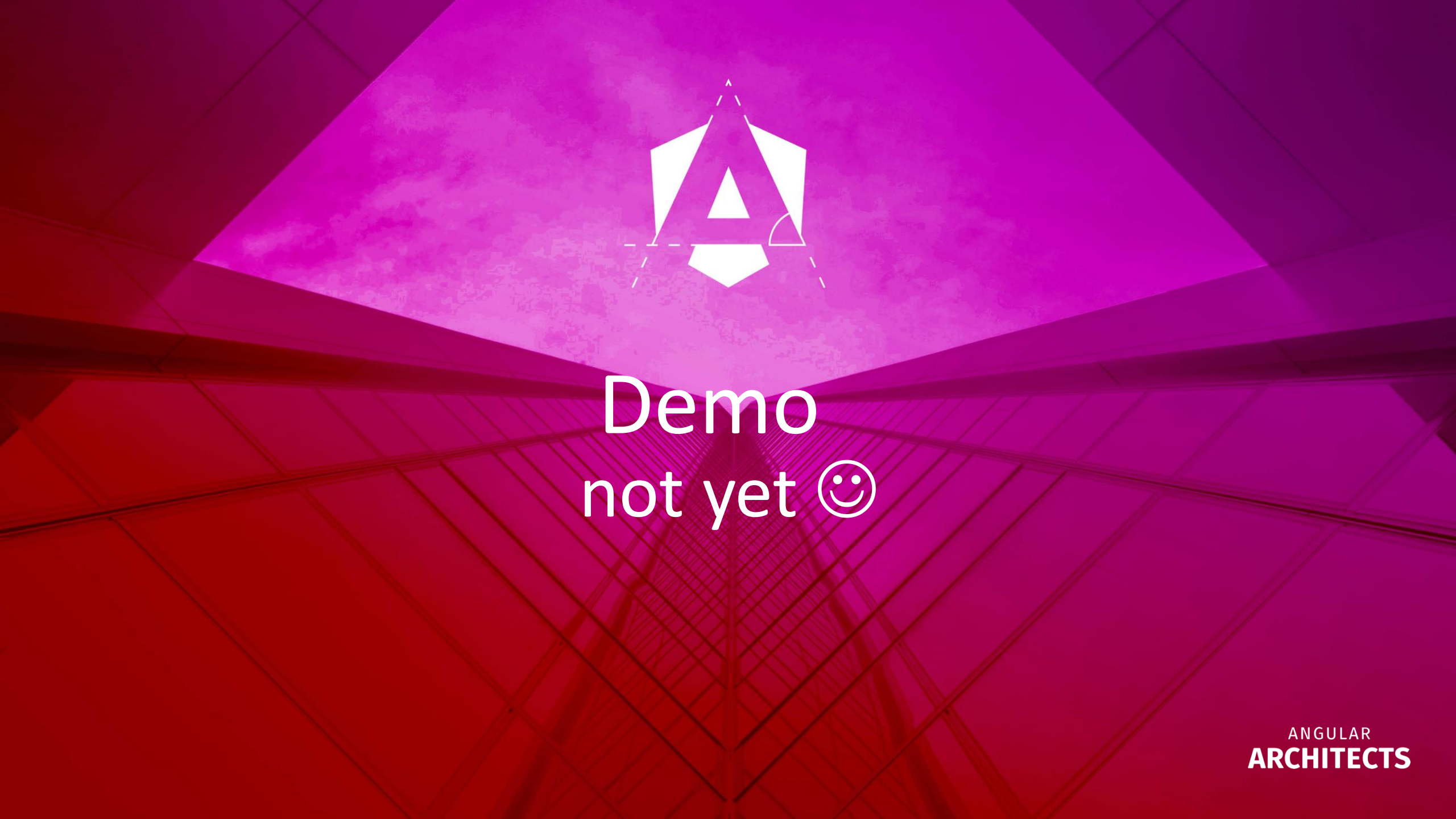
```
…
@defer (hydrate when condition) {
  <aa-lazy-component />
}
…
```

hydrate never

- component will be rendered but will not be hydrated

- means that even if the application is fully loaded on the client side, the defer block will remain static and not become interactive

```
…
@defer (hydrate never) {
  <aa-lazy-component />
}
…
```



Demo
not yet ☺

Recap

- Server-Side Rendering
- Prerendering / SSG
- Hydration
- Event Replay
- Hybrid Rendering
- Incremental Hydration

References

- Angular Architects Blog
  - Server-Side Rendering (Blog series)

- Angular Docs
  - Server-side rendering

ANGULAR
ARCHITECTS