Angular Performance

Your App at the Speed of Light

think tecture

Christian Liebel
@christianliebel
Consultant

Christian Liebel

Hello, it's me.

Cross-Platform & Serverless

Follow me:
@christianliebel

Blog:
christianliebel.com

Email:
christian.liebel
@thinktecture.com

Agenda

Runtime Performance

- Change Detection Basics
- Zone.js & NgZone
- Change Detection Strategies
- Change Detector
- Async Pipe

Load Time Performance

- Bundling
- Lazy Loading
- Preloading Strategies
- Server-Side Rendering
- Service Worker

First Rule

Don’t over-optimize.

Runtime Performance

In general: Reduce required computations during runtime (calculations, painting, layouting)

Not covered: CSS/JS tweaks, performance metrics, ...

Today: Angular-specific performance topics

Agenda

Runtime Performance

- Change Detection Basics
- Zone.js & NgZone
- Change Detection Strategies
- Change Detector
- Async Pipe

Load Time Performance

- Bundling
- Lazy Loading
- Preloading Strategies
- Server-Side Rendering
- Service Worker

Change Detection

Basics

```
// app.component.html
<h1>Hi {{ title }}!</h1>
```

```
// app.component.ts
@Component({ /* … */ })
export class AppComponent {
  title = 'Angular';
}
```

Change Detection

Basics

```
// app.component.html
<h1>Hi {{ title }}!</h1>

<button (click)="update()">
  Update
</button>
```


```
// app.component.ts
@Component({ /* … */ })
export class AppComponent {
  title = 'Angular';

  update() {
    this.title = 'Foo';
  }
}
```

Change Detection

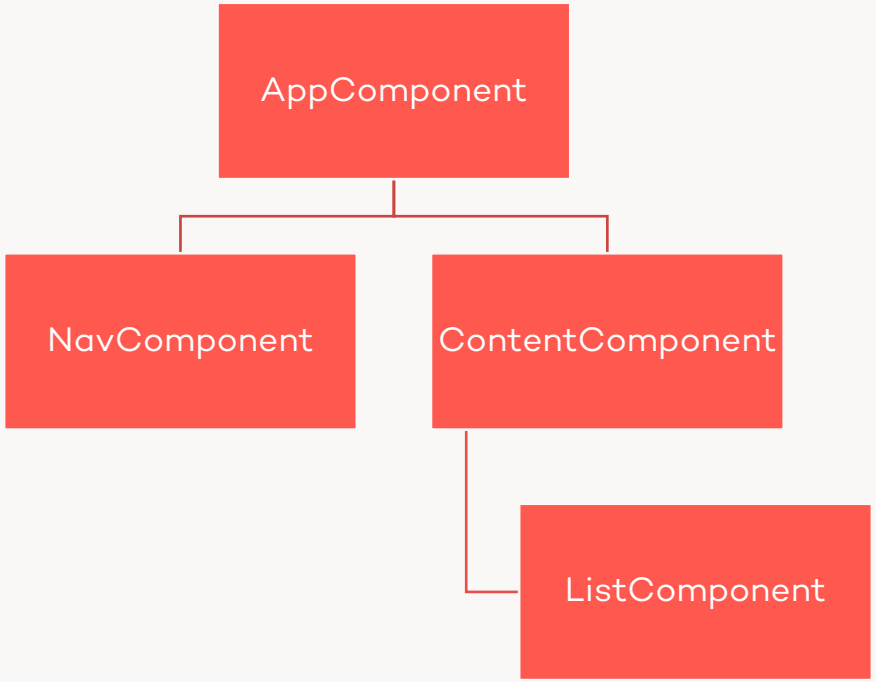
Basics

Change detection…

- is the magical part of Angular that makes data binding “just work”
- is a very handy feature that helps a lot, but it can also work against you
- is strongly related to Angular application performance

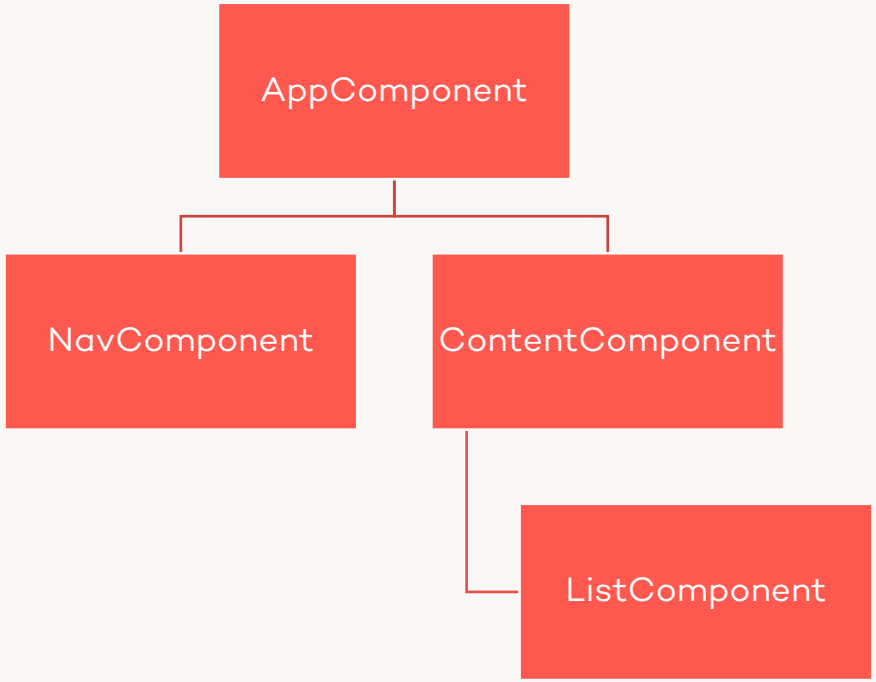
Change Detection

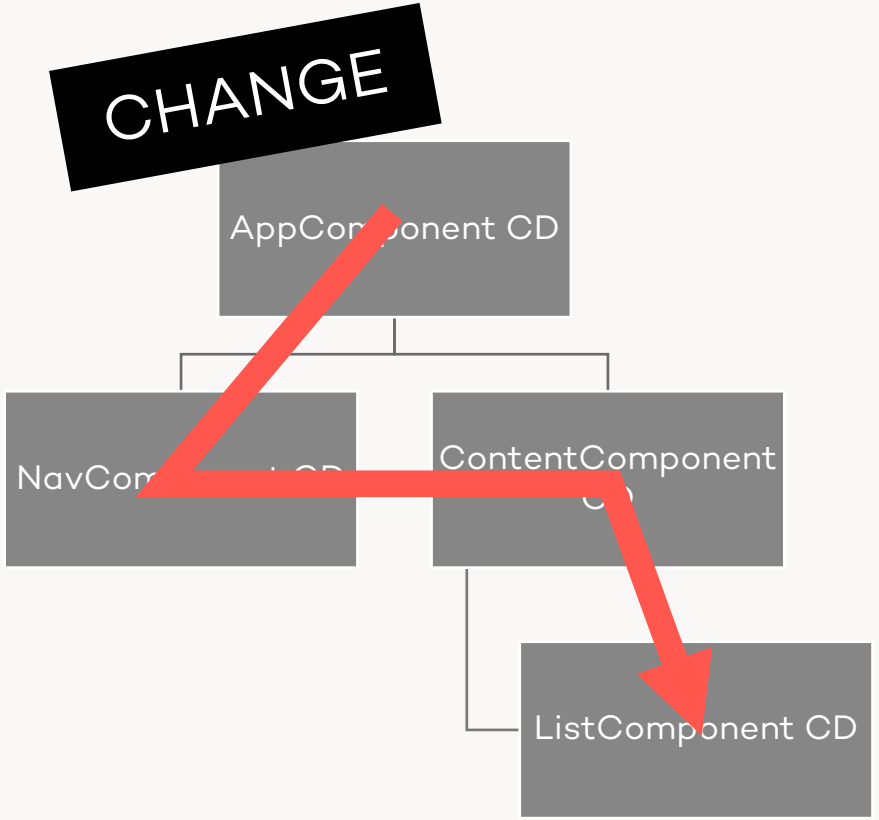
Component Tree



Change Detection

Change Detector Tree





Change Detection

Change Detector

`detectChanges()`

Called when an event has occured and bindings should be checked

```
this.title = 'Foo';
```

```
<h1>Hi {{ title }}!</h1>
```

Change Detection

Per default, each change in your application leads to...

- A single CD cycle
- From top to bottom (all components)
- Unidirectional (no cycles allowed)

DEMO

Change Detection

First findings

Reduce duration of a change detection cycle

- Reduce amount of bindings (e.g. grids: virtual scrolling via CDK)
- Avoid binding to (computationally intensive) getters or functions

Keep CD cycle < 16 ms!

Change Detection

Profiling

```
// main.ts
platformBrowserDynamic().bootstrapModule(AppModule).then(module =>
enableDebugTools(module.injector.get(ApplicationRef).components[0]));
```

Execute `ng.profiler.timeChangeDetection()` to measure the duration of a change detection run (500ms *or* 5 change detection cycles)

DEMO

Agenda

Runtime Performance

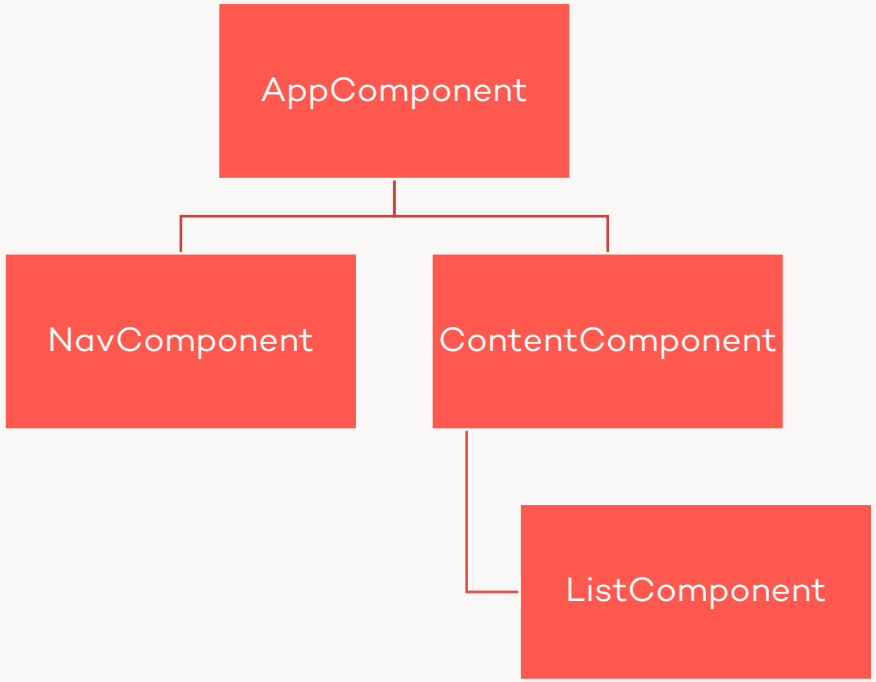
- Change Detection Basics
- Zone.js & NgZone
- Change Detection Strategies
- Change Detector
- Async Pipe

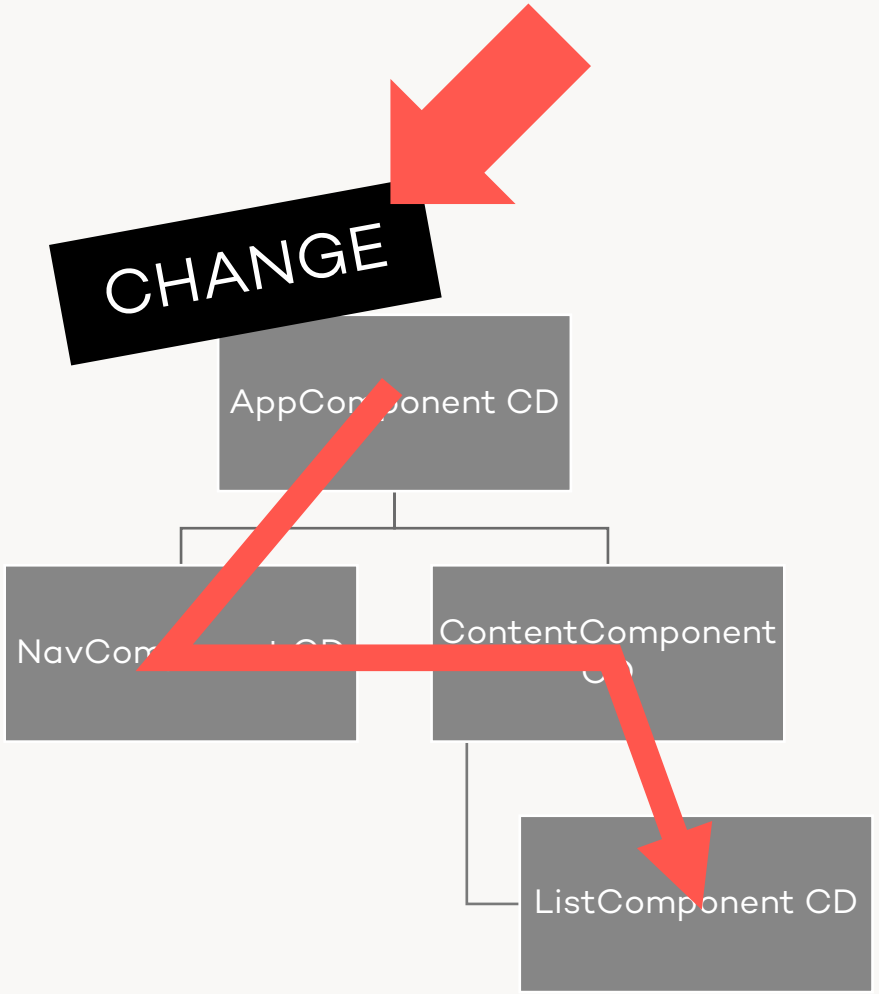
Load Time Performance

- Bundling
- Lazy Loading
- Preloading Strategies
- Server-Side Rendering
- Service Worker

Zone.js

How to detect a change?





Zone.js

A look at Angular's dependencies

```
"dependencies": {
    "@angular/common": "~7.2.0",

    "core-js": "^2.5.4",
    "rxjs": "~6.3.3",
    "zone.js": "~0.8.26"
  },
```

Zone.js

A Meta-Monkey Patch

setTimeout
setInterval
click
focus
mousemove
addEventListener
geolocation.getCurrentPosition
XMLHttpRequest
PromiseRejectionEvent
requestAnimationFrame

Zone.js

Execution Context

Debugging

Pending asynchronous tasks are known

Profiling

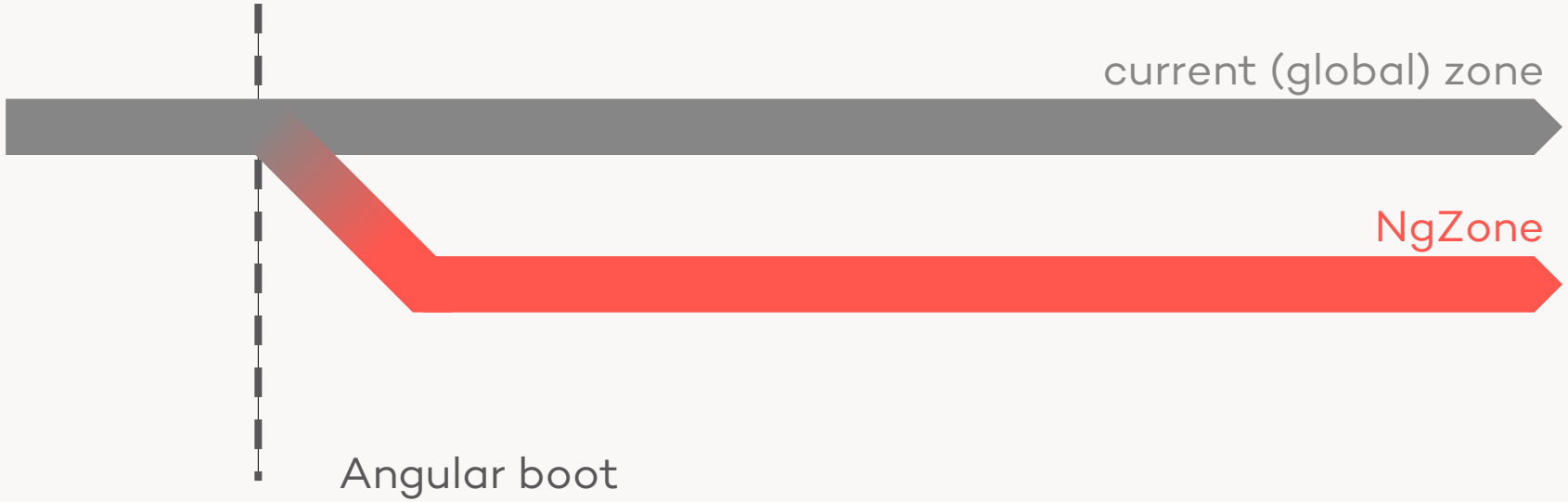
Measuring performance (Google Web Tracing Framework)

Mocking/Testing

Hooks *beforeTask*, ...

Zone.js

NgZone

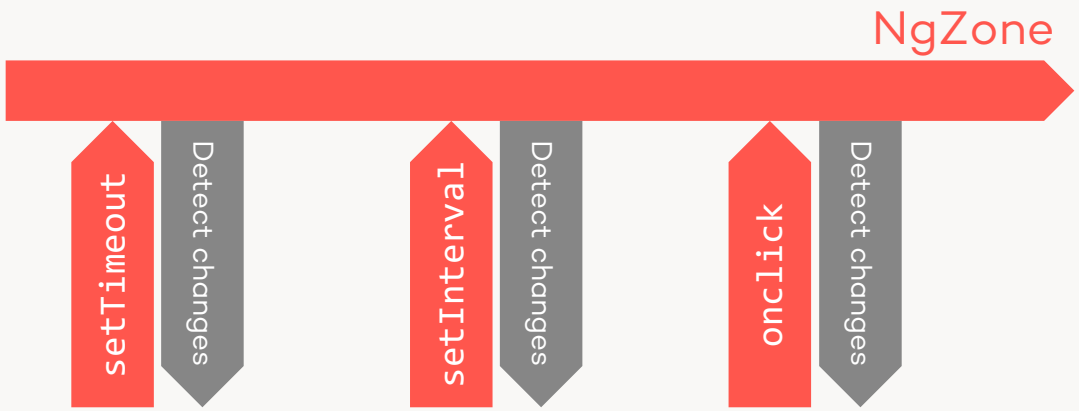


Zone.js

NgZone

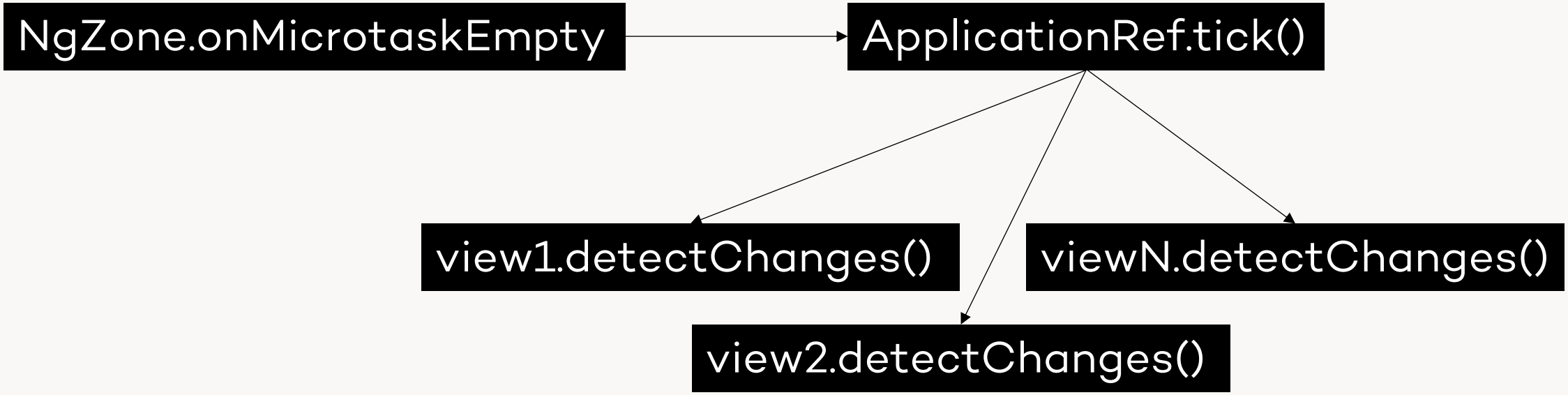
NgZone catches asynchronous operations from the Angular app

When no tasks are remaining for the current VM turn, the NgZone will trigger a change detection cycle (tick)



Zone.js

Change Detection Trigger



https://github.com/angular/angular/blob/master/packages/core/src/application_ref.ts

Zone.js

Common Pitfalls

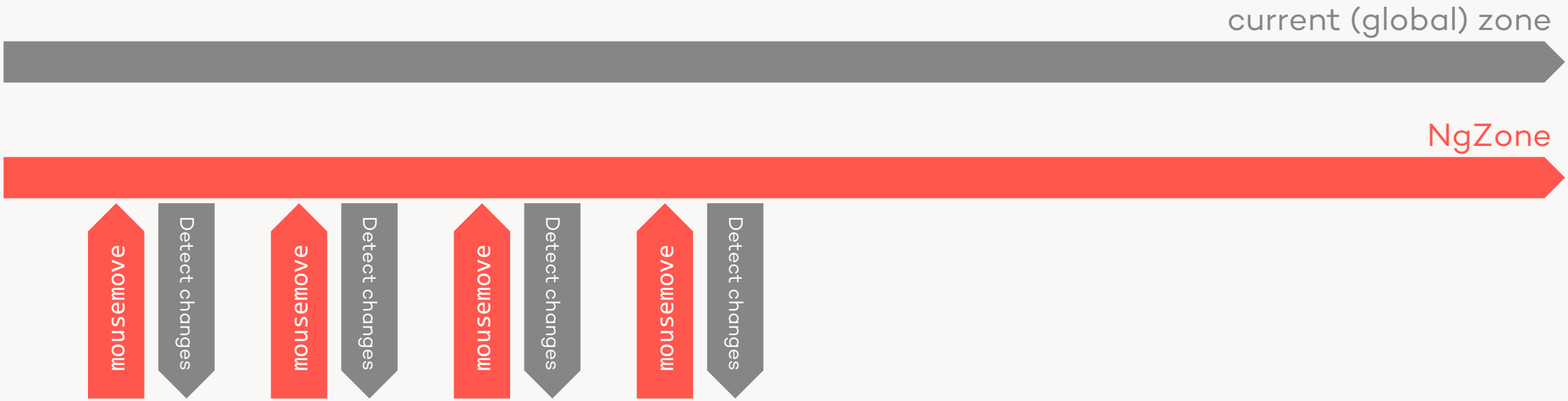
Long CD cycles in combination with high-frequency events

- mousemove
- scroll
- requestAnimationFrame
- setInterval with short intervals (clocks!)

DEMO

Zone.js

NgZone



Zone.js

NgZone Opt-Out

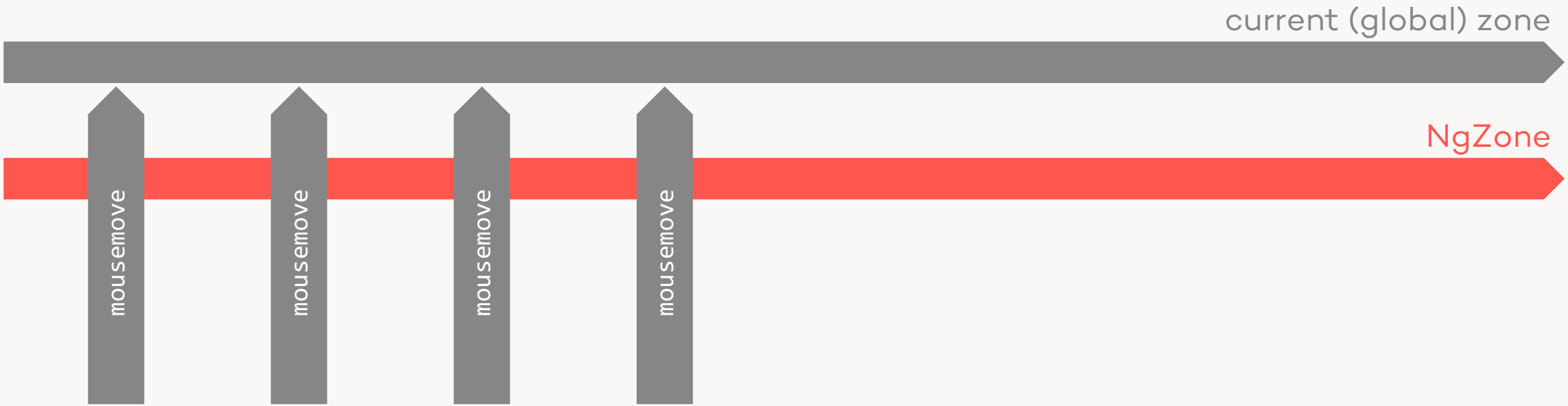
```
constructor (ngZone: NgZone) {
  ngZone.runOutsideAngular(() => {
    // runs outside Angular zone, for performance-critical code

    ngZone.run(() => {
      // runs inside Angular zone, for updating view afterwards
    });
  });
}
```

View and model can get out of sync!

Zone.js

NgZone



DEMO

Zone.js

Disable Patches (polyfills.ts)

```
(window as any).__Zone_disable_requestAnimationFrame = true;
// disable patch requestAnimationFrame

(window as any).__Zone_disable_on_property = true;
// disable patch onProperty such as onclick

(window as any).__zone_symbol__BLACK_LISTED_EVENTS = ['scroll',
'mousemove'];
// disable patch specified eventNames
```

View and model can get out of sync!

Zone.js

Disable Zone (= disable async change detection!)

```
platformBrowserDynamic().bootstrapModule(AppModule, {
  ngZone: 'noop'
});
```

```
constructor(applicationRef: ApplicationRef) {
  applicationRef.tick(); // trigger CD yourself
}
```

View and model can get out of sync!

Agenda

Runtime Performance

- Change Detection Basics
- Zone.js & NgZone
- **Change Detection Strategies**
- Change Detector
- Async Pipe

Load Time Performance

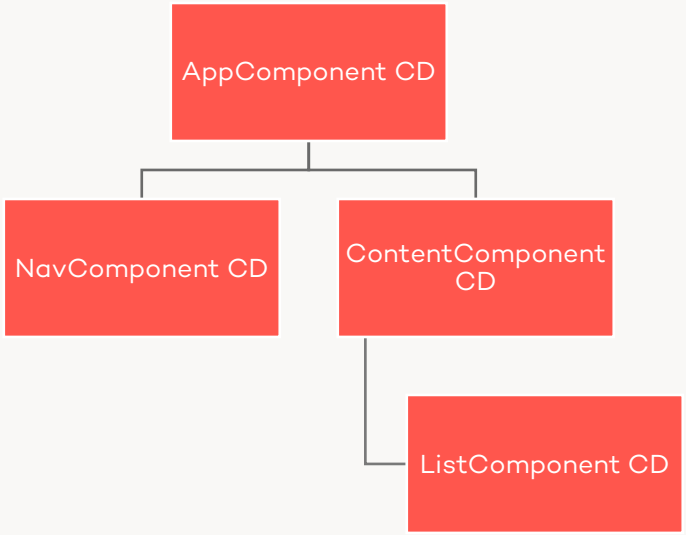
- Bundling
- Lazy Loading
- Preloading Strategies
- Server-Side Rendering
- Service Worker

Change Detection Strategies

Overview

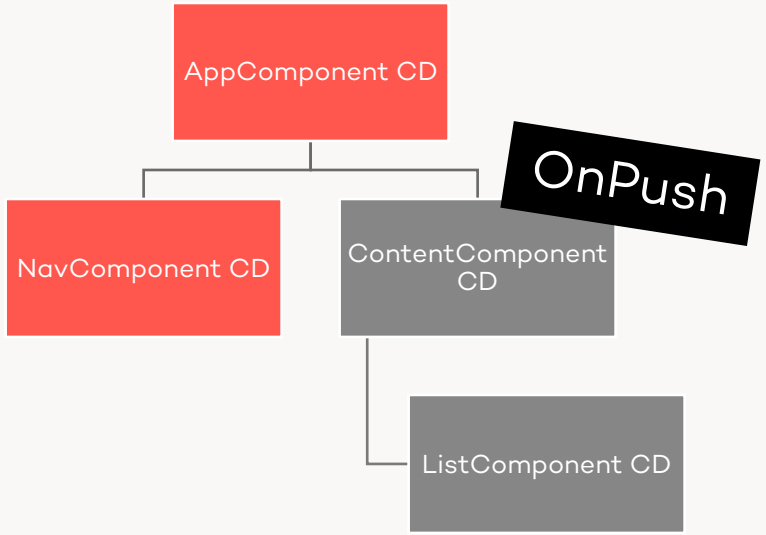
Default

Uses Zone.js for detecting changes and updates bindings



OnPush

Restricts change detection to changes of @Input parameters



Change Detection Strategies

OnPush

DEMO

```
<my-component [foo]="bar">
</my-component>

@Component({
  selector: 'my-component',
  template: '{{ foo }}',
  changeDetection:
    ChangeDetectionStrategy.OnPush
})
export class MyComponent {
  @Input()
  public foo: string;
}
```

Change detection only reacts to changes of @Input parameters

Angular compares the values passed to an @Input parameter (newValue === oldValue).

If you are passing objects, make sure to pass in new instances!

View and model can get out of sync!

Change Detector

OnPush & Detecting Changes

What to do if a component changes unrelated to an @Input parameter?

```
constructor(private dataService: DataService) {}

ngOnInit() {
  this.dataService.updates$
    .subscribe(newData => this.data = newData); // no update!
}
```

Change Detector

ChangeDetectorRef

```
constructor(cdRef: ChangeDetectorRef) {}
```

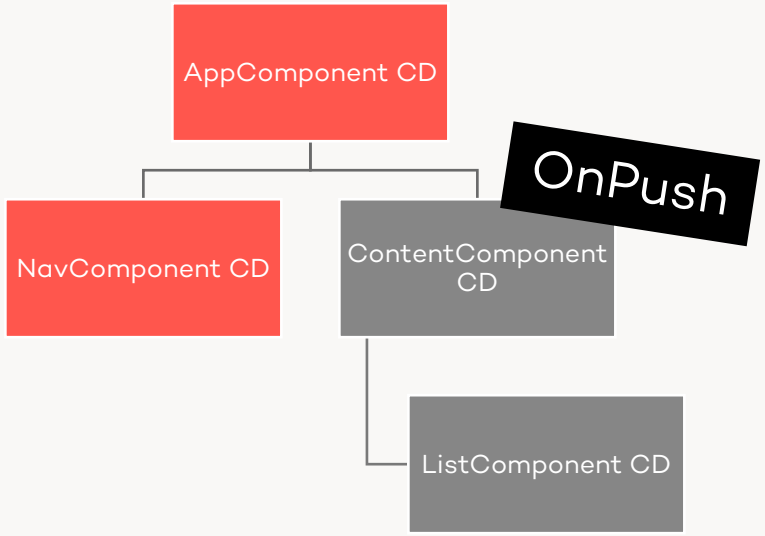
A reference to the ChangeDetector of your component

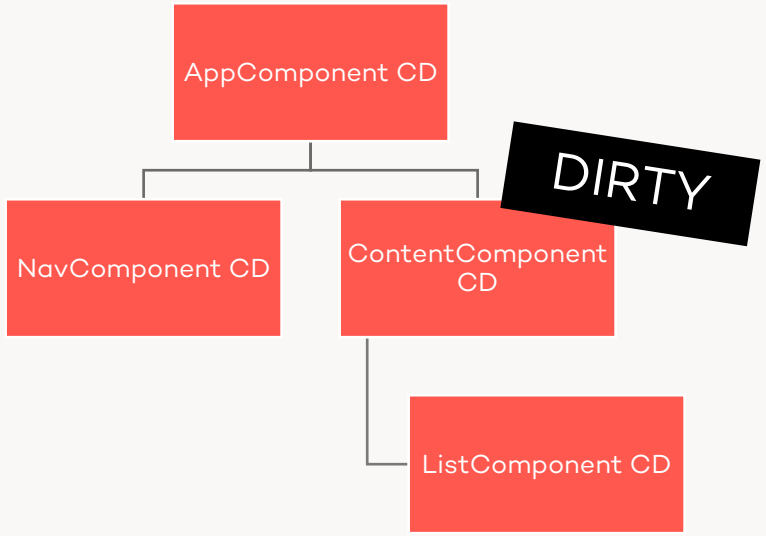
- `detectChanges()`
- `markForCheck()`
- `detach()`
- `checkNoChanges()`
- `reattach()`

Change Detector

markForCheck()

Explicitly marks a component as dirty/changed (when using OnPush)





Change Detector

markForCheck()

```
constructor(private dataService: DataService,
            private cdRef: ChangeDetectorRef) {}

ngOnInit() {
  this.dataService.updates$.subscribe(newData => {
    this.data = newData;
    this.cdRef.markForCheck();
  });
}
```

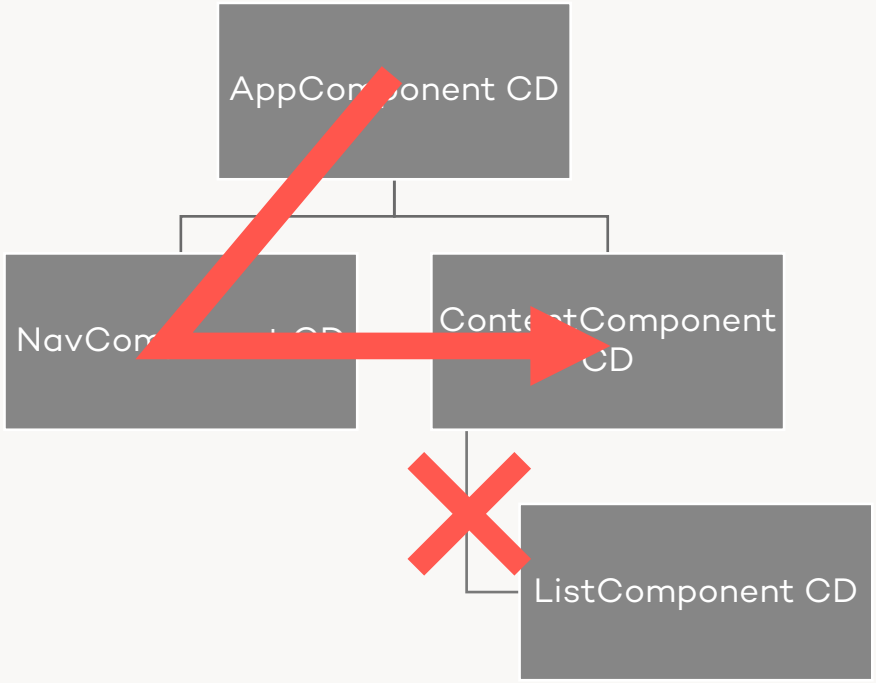
think tecture

Change Detector

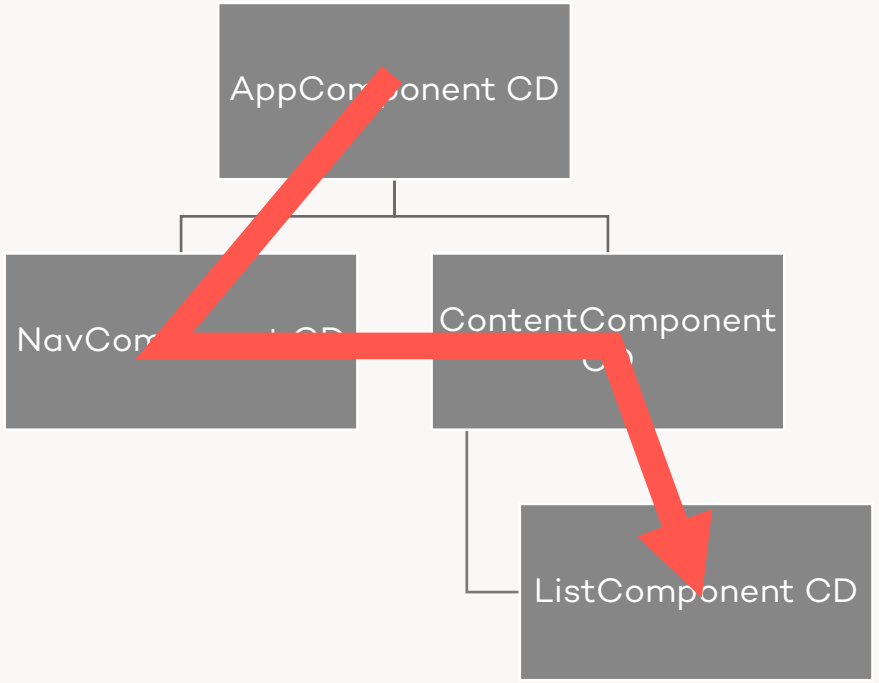
Detaching Components

View and model can get out of sync!

```
changeDetector.detach();
```




```
changeDetector.reattach();
```



Change Detector

Local Change Detection

```
constructor(cdRef: ChangeDetectorRef) {
  cdRef.detach(); // detaches this view from the CD tree
  // cdRef.detectChanges(); // detect this view & children
  // cdRef.reattach();
}
```

Change Detector

Findings

Reduce amount of change detection cycles

- Disable Zone.js (not a good idea in most cases)
- Opt-out of NgZone (for operations that should not affect bindings)
- Disable Zone.js patches (in case you can't opt-out, e.g. 3rd party libs)
- ChangeDetectionStrategy.OnPush (good default, but be careful)
- Local change detection via ChangeDetectorRef (for the few components that do not have to respond to changes from outside)

Agenda

Runtime Performance

- Change Detection Basics
- Zone.js & NgZone
- Change Detection Strategies
- Change Detector
- Async Pipe

Load Time Performance

- Bundling
- Lazy Loading
- Preloading Strategies
- Server-Side Rendering
- Service Worker

Async Pipe

Overview

Takes observables or promises

```
{{ data$ | async }}
```

Waits for the observable to emit/promise to resolve and then displays the value

Async Pipe

Advantages

For observables:

- Async Pipe subscribes for you
- Async Pipe takes care of unsubscribing from the observable
- Async Pipe calls `markForCheck` for each update – perfect match for OnPush!

https://github.com/angular/angular/blob/master/packages/common/src/pipes/async_pipe.ts

Async Pipe

Simplifying OnPush

```
// component.ts
data$: Observable<string>;
constructor(dataService: DataService) {
  this.data$ = this.dataService.updates$;
}

// component.html
{{ data$ | async }}
```

Load Time Performance

Not covered:

– HTTP/2, compression, …

Today:

– Reduce initial load (size & computation)
– Reduce perceived loading time
– Prevent downloading the same resource again

Agenda

Runtime Performance

- Change Detection Basics
- Zone.js & NgZone
- Change Detection Strategies
- Change Detector
- Async Pipe

Load Time Performance

- **Bundling**
- Lazy Loading
- Preloading Strategies
- Server-Side Rendering
- Service Worker

Bundling

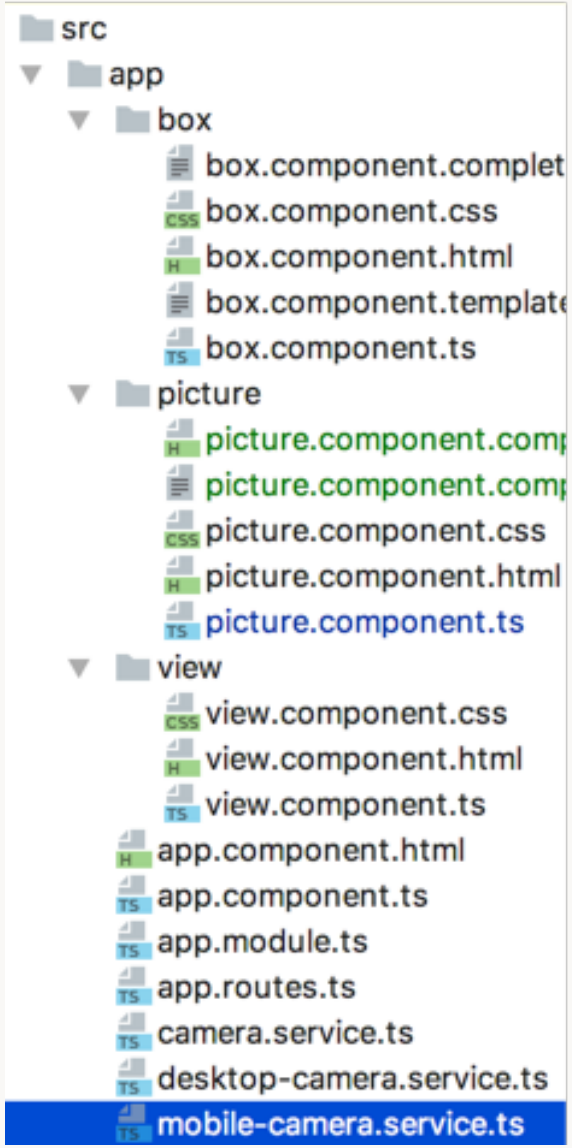
The Problem

Angular's development directory structure is hard to

- deploy
- serve
- cache
- …

Lots of files, lots of requests

Angular and its dependencies are large in size, apps use only a fragment



Bundling

The Problem

Just-in-Time compilation (JiT)

- Slow, client-side rendering
- Compiler is 1.2 MB large in size
- Template errors detected at runtime only
- Potentially dangerous (injection attacks)

Bundling

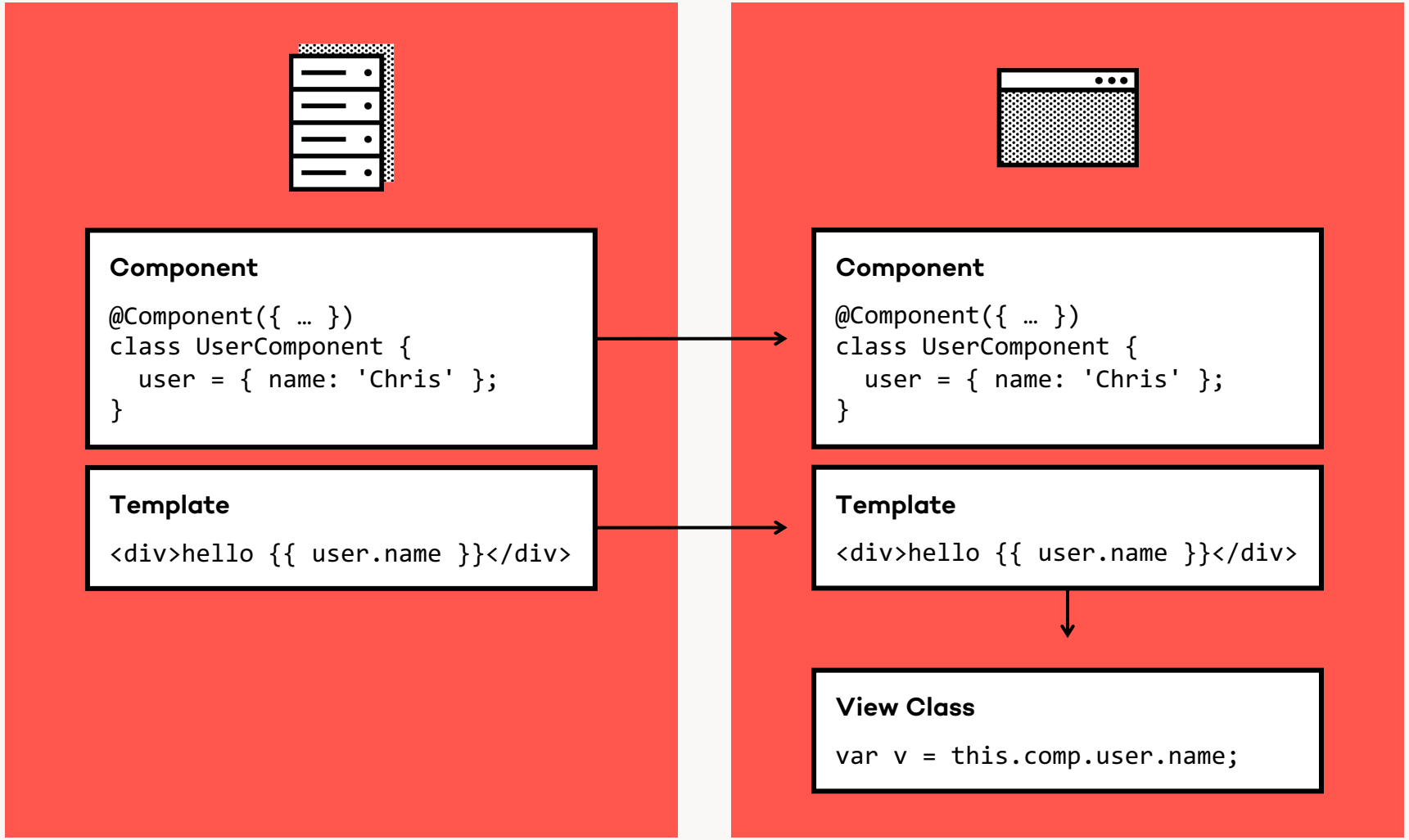
The Problem

Goal: Angular app

- with all components pre-compiled
- combined in a single (or few) file(s)
- without redundant/unused code
- uglified, compressed

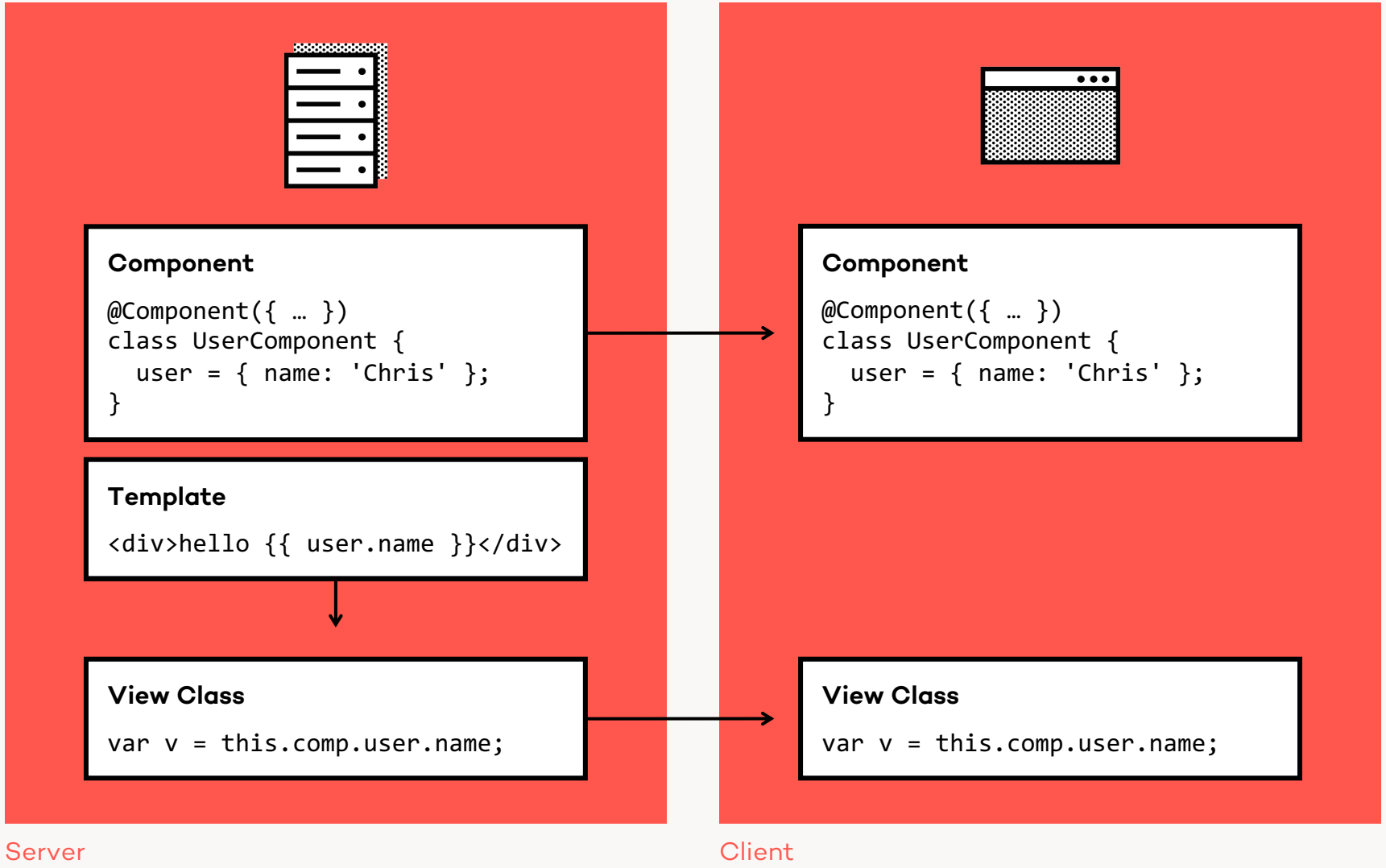
Bundling

JiT Compilation



Bundling

AoT Compilation



Bundling

ng build --prod

AoT

+

Tree Shaking: “walks the dependency graph, top to bottom, and shakes out unused code like dead needles in a Christmas tree.”

+

Build Optimizer: applies Angular optimizations to JavaScript code

Bundling

A Simple Demo App

Dev build: **4.2 MB** (without source maps)

AoT build: **2.8 MB** (without source maps)

AoT+TreeShake: **502K**

AoT+TreeShake+BuildOptimizer: 379K (**106K** gzipped)

Bundling

Differential Loading

Detect the platform and only deliver files required for this platform

First version introduced with Angular CLI 7.3.0

core.js Polyfills required for ES5 browsers are only delivered to ES5 browsers

Saves another 56+ K for modern browsers

Bundling

Differential Loading

```
// index.html
<!doctype html>
<html lang="en">
<script type="text/javascript" src="runtime.js"></script>
<script type="text/javascript" src="es2015-polyfills.js"
nomodule></script>
<script type="text/javascript" src="polyfills.js"></script>
<script type="text/javascript" src="styles.js"></script>
<script type="text/javascript" src="vendor.js"></script>
<script type="text/javascript" src="main.js"></script></body>
</html>
```

Bundling

Differential Loading

- Differential loading support comes to all files
- Angular CLI 8+ can produce ES5 + ES2015 bundles of your application
- ES2015 files (smaller footprint) will be delivered to modern browsers only

Agenda

Runtime Performance

- Change Detection Basics
- Zone.js & NgZone
- Change Detection Strategies
- Change Detector
- Async Pipe

Load Time Performance

- Bundling
- **Lazy Loading**
- Preloading Strategies
- Server-Side Rendering
- Service Worker

Lazy Loading

Overview

Angular router supports lazy loading components transparently

Lazy loaded components are not delivered to/loaded by the client on boot, but on purpose

Reduces load & perceived loading time

Lazy Loading

Overview

```
const ROUTES: Routes = [{
   path: 'admin',
   loadChildren: () => import('./lazy/lazy.module')
                              .then(m => m.LazyModule)
}];
```

Agenda

Runtime Performance

- Change Detection Basics
- Zone.js & NgZone
- Change Detection Strategies
- Change Detector
- Async Pipe

Load Time Performance

- Bundling
- Lazy Loading
- **Preloading Strategies**
- Server-Side Rendering
- Service Worker

Preloading Strategies

Configuring Lazy Loading

NoPreloading

- does not preload any route by default
- advantage: low size footprint
- disadvantage: takes some time after clicking a link to the lazy-loaded module, not offline capable

PreloadAllModules

- automatically preloads all modules after the application has launched (still better loading time!)
- advantage: lazy-loaded modules now load instant (also on the first click), offline capable
- disadvantage: higher footprint

Preloading Strategies

Configuring Lazy Loading

```
@NgModule({
  imports: [RouterModule.forRoot(routes, {
    preloadingStrategy: PreloadAllModules,
  })],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Preloading Strategies

Custom Strategy

```
preload(route: Route, fn: () => Observable<any>): Observable<any> {
  // decide based on route (or other external information)
  // call fn to preload the module
  // otherwise, return of(null)
}
```

https://github.com/angular/angular/blob/8.1.x/packages/router/src/router_preloader.ts#L41

Agenda

Runtime Performance

- Change Detection Basics
- Zone.js & NgZone
- Change Detection Strategies
- Change Detector
- Async Pipe

Load Time Performance

- Bundling
- Lazy Loading
- Preloading Strategies
- Server-Side Rendering
- Service Worker

Server-Side Rendering

Principle

Angular Universal

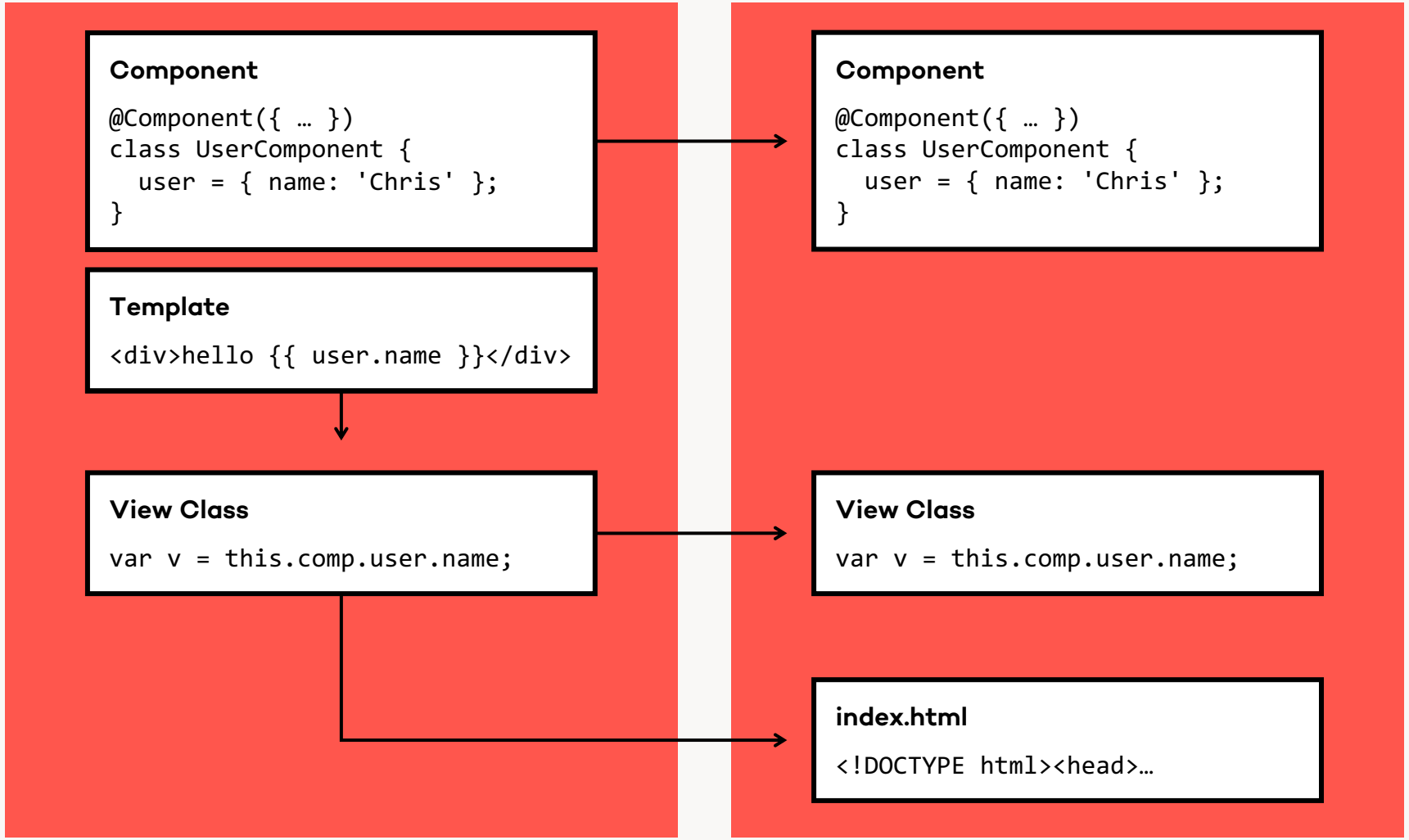
Pre-render the website using the same sources that are served

Once Angular kicks in, the view is replaced with the client-rendered one

Supports Node.js (Express) & ASP.NET Core

Server-Side Rendering

Principle



Server-Side Rendering

Purpose

Search Engine Optimization

Preview Links (Social Media)

Graceful Degradation

Reduce Perceived Loading Time/Quick First Contentful Paint (FCP)

Improve Performance for Mobile/Low-Powered Devices

Server-Side Rendering

The Web App Gap

Server-Side Rendering

Preboot.js

Filling the Web App Gap

Records interactions of the user on the server-rendered part

Replays the interaction once Angular kicks in on the client side

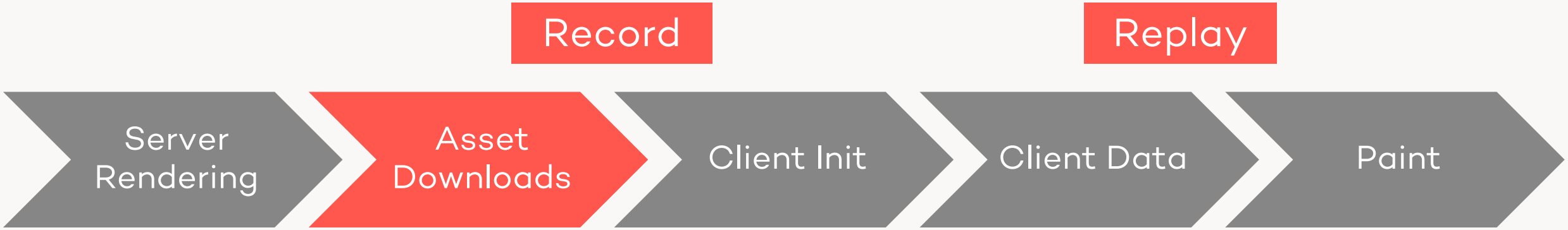
Provided by the Angular team

Open source

https://github.com/angular/preboot

Server-Side Rendering

Preboot.js & The Web App Gap



Agenda

Runtime Performance

- Change Detection Basics
- Zone.js & NgZone
- Change Detection Strategies
- Change Detector
- Async Pipe

Load Time Performance

- Bundling
- Lazy Loading
- Preloading Strategies
- Server-Side Rendering
- Service Worker

Service Worker

Idea: Never load the same resource twice

Download resources once and store them in a local cache

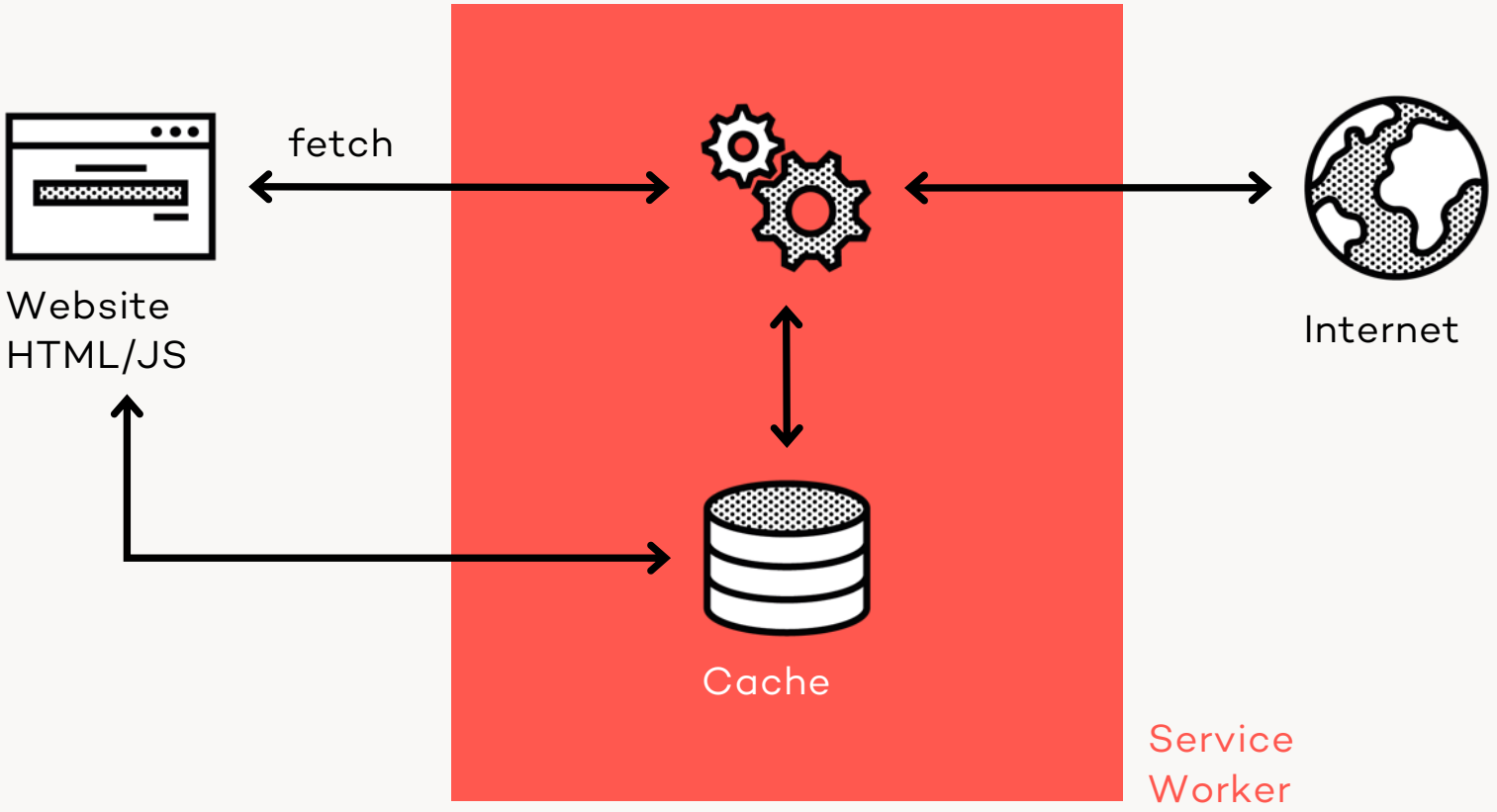
The next time the user wants to open the application, load the contents from there

Makes your application sources offline-capable

Significantly improves loading time

Service Worker

Architecture



Service Worker

@angular/service-worker

Service Worker implementation provided by the Angular team

Features

- Caching
- Offline Availability
- Push Notifications

Service Worker is generated by the CLI (prod builds only)

```
ng add @angular/pwa
```

Cheat Sheet

Runtime Performance

1. **Don’t over-optimize**
2. **Reduce duration of a change detection cycle**
   - Reduce amount of bindings
   - Avoid binding to (computationally intensive) getters or functions
3. **Reduce amount of change detection cycles**
   - Disable zone
   - NgZone
   - Zone.js patches
   - ChangeDetectorRef
   - ChangeDetectionStrategy

Thank you

for your kind attention!

think tecture

Christian Liebel

@christianliebel

christian.liebel@thinktecture.com