Angular Renaissance Slides

https://tinyurl.com/codeMashNgRen

The Angular Renaissance

Key New Features

Lance Finney

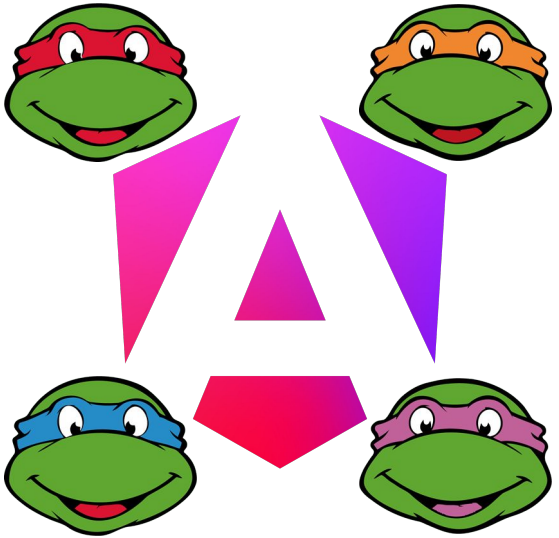
INFICON

CodeMash 2025

Brief History of Angular

AngularJS
2010

Angular (2+)
2016



Angular 17
Nov 8, 2023

Angular 19
Nov 19, 2024

Areas of focus

Developer experience	→ Make hard problems easy and impossible problems possible → Bring everyone along with the evolution of the framework
Simplification	→ Make Angular easy to use → Improve the learning journey and make it accessible
Openness	→ Listen to developers and focus on what is important → Develop the framework in the open and foster inclusive community
Brand identity	→ Keep the brand identity in sync with Angular's evolution → Evolve the framework with the evolution of web development

Minko Gechev at ngPoland 2024 https://myconf.dev/videos/minko-gechev-mark-thompson-keynote-session

Standalone Components

(and Pipes and Directives)

What changed?

Angular components were attached to NgModules

Components now are their own modules

Simpler mental model (and other advantages)

Converted Component

```
@Component({
  selector: 'tmnt-sewer',
  template: `
    <tmnt-welcome *ngIf="manholeOpen" />

    <button (click)="manholeOpen = !manholeOpen">
      {{ manholeOpen ? 'Close' : 'Open' }} Manhole
    </button>
  `
})
export class SewerComponent {
  manholeOpen = false;
}
```

```
@Component({
  selector: 'tmnt-sewer',
  standalone: true,
  imports: [NgIf, WelcomeComponent],
  template: `
    <tmnt-welcome *ngIf="manholeOpen" />

    <button (click)="manholeOpen = !manholeOpen">
      {{ manholeOpen ? 'Close' : 'Open' }} Manhole
    </button>
  `
})
export default class SewerComponent {
  manholeOpen = false;
}
```

Not needed after Angular 19

NgModule

```ts
const routes: Routes = [
  {
    path: '',
    component: SewerComponent,
    pathMatch: 'full'
  }
];

@NgModule({
  declarations: [SewerComponent, WelcomeComponent],
  imports: [CommonModule, RouterModule.forChild(routes)]
})
export class SewerModule {}
```

➡

Where did the routing go?

Bootstrapping changes

main.ts

```ts
void platformBrowserDynamic().bootstrapModule(AppModule);
```

app.module.ts

```ts
const routes: Routes = [
  { path: '', redirectTo: 'sewer', pathMatch: 'full' },
  {
    path: 'sewer',
    loadChildren: () =>
      import('./sewer/sewer.module').then(m => m.SewerModule)
  }
];

@NgModule({
  bootstrap: [AppComponent],
  declarations: [AppComponent],
  imports: [BrowserModule, RouterModule.forRoot(routes)]
})
export class AppModule {}
```

main.ts

```ts
void bootstrapApplication(AppComponent, appConfig);
```

app.config.ts

```ts
const routes: Routes = [
  { path: '', redirectTo: 'sewer', pathMatch: 'full' },
  {
    path: 'sewer',
    loadComponent: () => import('./sewer/sewer.component')
  }
];

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes)]
};
```

Original vision fulfilled

Components were supposed to be standalone

The original compiler couldn’t do it

Project Ivy made it possible

Standalone Status

- Introduced in Angular 14/15
- Standalone by default for new components in Angular 17
- “standalone: true” implicit in Angular 19 (ng update fixes old components)

New Control Flow Syntax

https://angular.dev/guide/templates/control-flow

@if

```
imports: [NgIf, WelcomeComponent],
template: `
  <tmnt-welcome *ngIf="manholeOpen; else closed" />
  <ng-template #closed><p>Stay Out!</p></ng-template>

  <button (click)="manholeOpen = !manholeOpen">
    {{ manholeOpen ? 'Close' : 'Open' }} Manhole
  </button>
`
```

```
imports: [WelcomeComponent],
template: `
  @if (manholeOpen) {
    <tmnt-welcome />
  } @else {
    <p>Stay Out!</p>
  }

  <button (click)="manholeOpen = !manholeOpen">
    {{ manholeOpen ? 'Close' : 'Open' }} Manhole
  </button>
`
```

@for

```
imports: [NgIf, NgFor],
template: `
  <ul>
    <li *ngFor="let pizza of pizzas(); index as i">
      {{ i + 1 }}: {{ pizza.name }}
    </li>
    <li *ngIf="!pizzas().length">Bummer!</li>
  </ul>
`
```

➡

```
template: `
  <ul>
    @for (pizza of pizzas(); track pizza.id) {
      <li>{{ $index + 1 }}: {{ pizza.name }}</li>
    } @empty {
      <li>Bummer!</li>
    }
  </ul>
`
```

@switch

```
imports: [
  DonatelloComponent,
  LeonardoComponent,
  MichelangeloComponent,
  NgSwitch,
  NgSwitchCase,
  NgSwitchDefault,
  RaphaelComponent,
  SplinterComponent
],
template: `
  <ng-container [ngSwitch]="color">
    <tmnt-donatello *ngSwitchCase="'purple'" />
    <tmnt-leonardo *ngSwitchCase="'blue'" />
    <tmnt-michelangelo *ngSwitchCase="'orange'" />
    <tmnt-raphael *ngSwitchCase="'red'" />
    <tmnt-splinter *ngSwitchDefault />
  </ng-container>
`
```

```
imports: [
  DonatelloComponent,
  LeonardoComponent,
  MichelangeloComponent,
  RaphaelComponent,
  SplinterComponent
],
template: `
  @switch (color) {
    @case ('purple') {
      <tmnt-donatello />
    }
    @case ('blue') {
      <tmnt-leonardo />
    }
    @case ('orange') {
      <tmnt-michelangelo />
    }
    @case ('red') {
      <tmnt-raphael />
    }
    @default {
      <tmnt-splinter />
    }
  }
`
```

Advantages

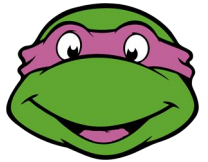
- Fewer imports
- Smaller bundles
- Better performance
- Better type safety
- Easier ergonomics

Control Flow Status

- Developer preview in Angular 17
- Stable in Angular 18

Deferrable Views

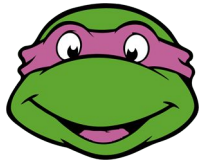
https://angular.dev/guide/templates/defer



Improve UX by lazy-loading views

- More fine-grained than existing route-based lazy-loading
- Works with standalone components

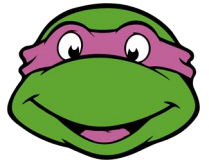
```
@defer (on idle) {
  <app-idle />
  <app-non-standalone />
} @placeholder (minimum 2s) {
  <p>This placeholder will go away after 2s</p>
} @loading (minimum 1s) {
  <p>Loading...</p>
} @error {
  <p>Uh oh</p>
}
```



Features

- You can create your own triggers

- Six built-in triggers for loading and prefetching
  - on idle
  - on viewport
  - on interaction
  - on hover
  - on immediate
  - on timer



Deferrable Views Status

- Developer preview in Angular 17
- Stable in Angular 18
- Note: for both control flow and deferrable views, upgrade prettier to 3.1.0+

Signals

https://angular.dev/essentials/signals

A	B
Number of Guests	75
Cost per Guest	$56.12
Subtotal	$4,209.00
Employee Discount Percent	40.00%
Total Discount	$1,683.60
Tax Rate Percent	7.25%
Total Tax	$183.09
Total Cost	$2,708.49

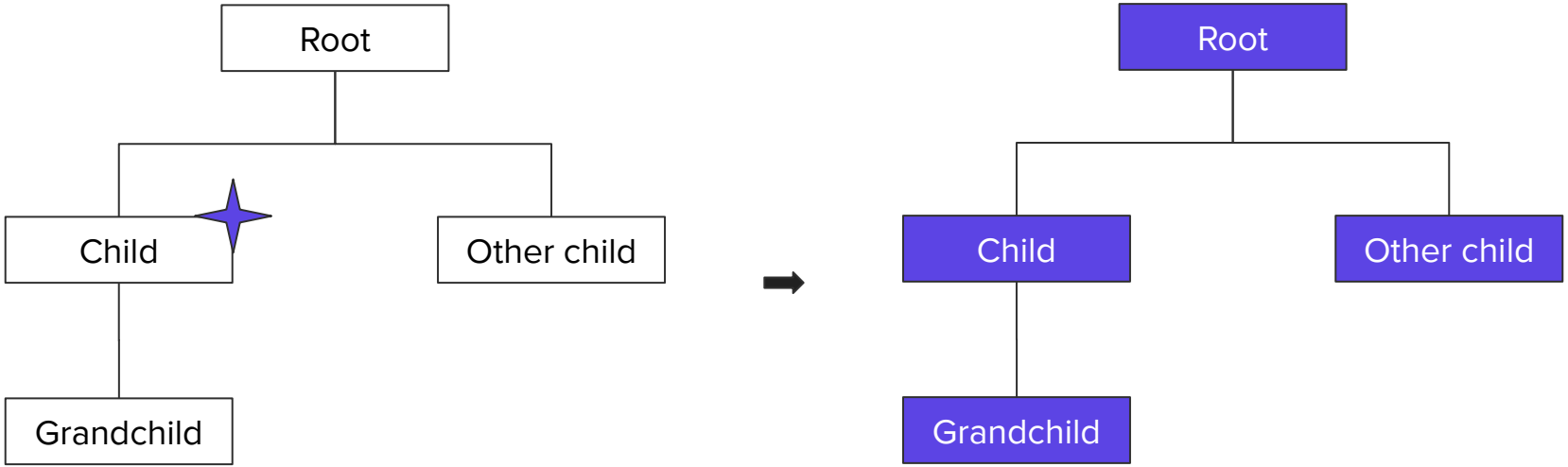
ilar to

vables

Main motivations

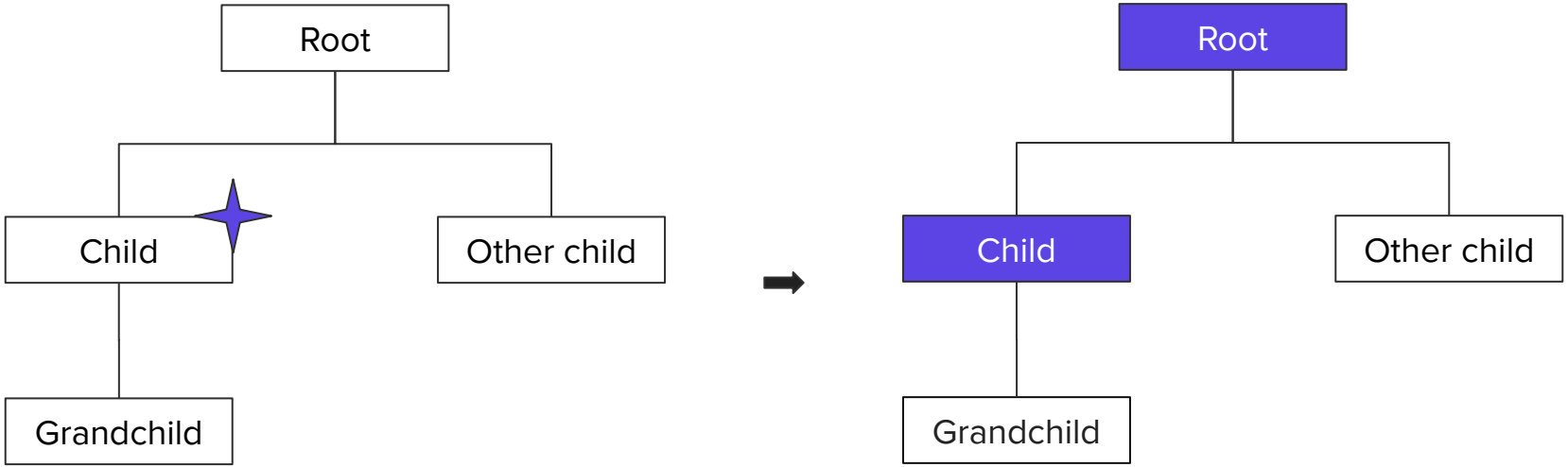
- Improve change detection
- Simpler declarative execution

Default change detection w/ Zones



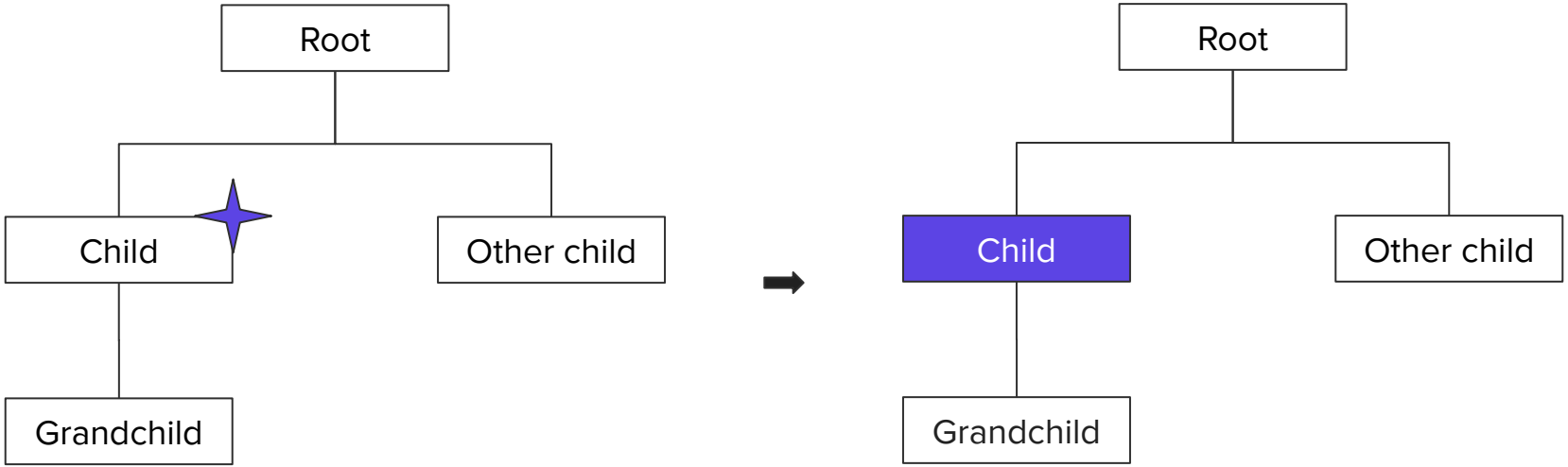
Demo from JetBrains showing change detection strategies: https://www.youtube.com/live/9SqXk7lw8AQ?si=zFUwSsPGoiCfsJEo&t=1889

OnPush change detection



Demo from JetBrains showing change detection strategies: https://www.youtube.com/live/9SqXk7Iw8AQ?si=zFUwSsPGoiCfsJEo&t=1889

Signal-based change detection



Warning: full implementation doesn't exist yet!

Demo from JetBrains showing change detection strategies: https://www.youtube.com/live/9SqXk7lw8AQ?si=zFUwSsPGoiCfsJEo&t=1889

- Zones add to initial bundle
- Zones run more than necessary
- Monkey patching is weird

Any other problems with Zones?

Alex Rickabaugh at ngPoland 2025:
https://myconf.dev/videos/alex-rickabaugh-pawel-kozlowski-keynote-session-exploring-control-flow-and-reactivity

Signals

```
  template: `
    @if (manholeOpen) {
      <tmnt-welcome />
    }

    <button (click)="manholeOpen = !manholeOpen">
      {{ manholeOpen ? 'Close' : 'Open' }} Manhole
    </button>
  `
})
export default class SewerComponent {
  manholeOpen = false;
}
```

➡

```
  template: `
    @if (manholeOpen()) {
      <tmnt-welcome />
    }

    <button (click)="manholeOpen.set(!manholeOpen())">
      {{ manholeOpen() ? 'Close' : 'Open' }} Manhole
    </button>
  `
})
export default class SewerComponent {
  manholeOpen: WritableSignal<boolean> = signal(false);
}
```

Signals (computed and update)

```
  template: `
    @if (manholeOpen()) {
      <tmnt-welcome />
    }

    <button (click)="manholeOpen.set(!manholeOpen())">
      {{ manholeOpen() ? 'Close' : 'Open' }} Manhole
    </button>
  `
})
export default class SewerComponent {
  manholeOpen: WritableSignal<boolean> = signal(false);
}
```

➡

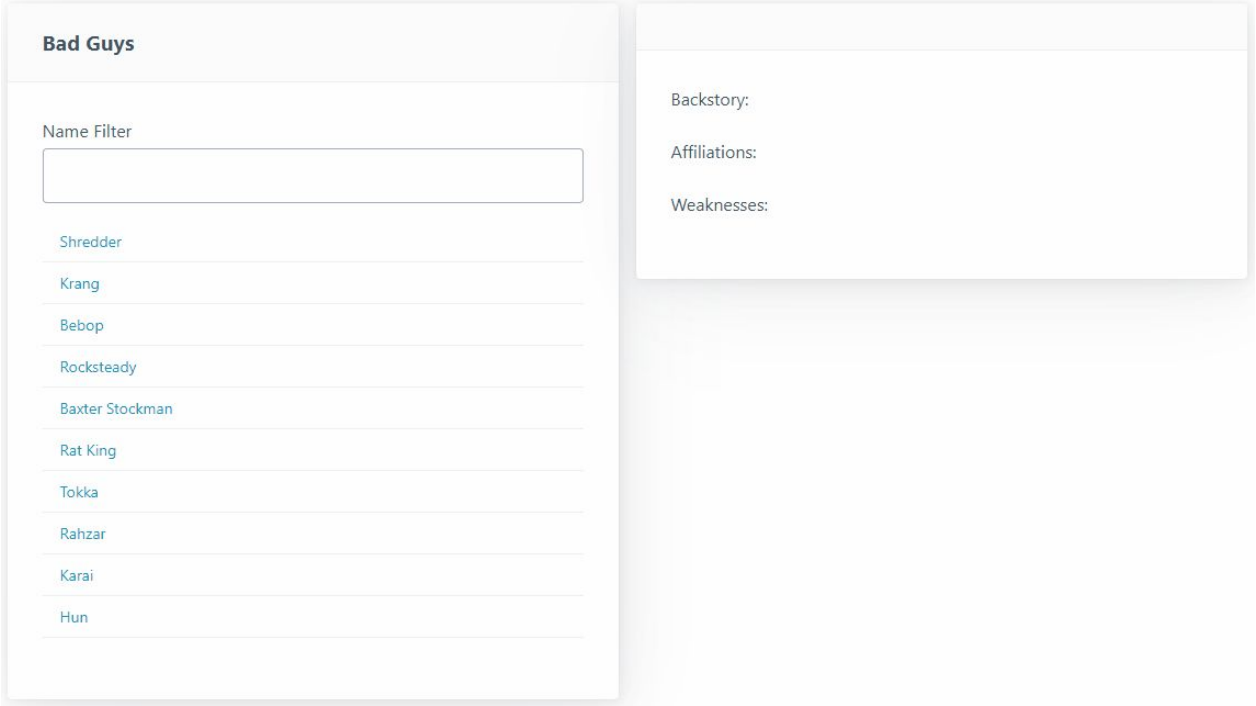
```
  template: `
    @if (manholeOpen()) {
      <tmnt-welcome />
    }

    <button (click)="toggleManhole()">
      {{ buttonText() }}
    </button>
  `
})
export default class SewerComponent {
  manholeOpen: WritableSignal<boolean> = signal(false);
  buttonText: Signal<string> = computed(
    () => `${this.manholeOpen() ? 'Close' : 'Open'} Manhole`
  );

  toggleManhole() {
    this.manholeOpen.update(open => !open);
  }
}
```

LinkedSignal - computed *and* writeable

```
manholeOpen: WritableSignal<boolean> = signal(false);
buttonText: WritableSignal<string> = linkedSignal(
  () => `${this.manholeOpen() ? 'Close' : 'Open'} Manhole`
);

changeLabel(label: string) {
  this.buttonText.set(label);
}
```

https://angular.dev/guide/signals/linked-signal

RxJS Interop



RxJS Interop (Template)

```
  <app-bad-guy-list-table-view
    [list]="(filteredList | async) ?? []"
    [selectedId]="selectedId | async"
    (selectId)="selectedId.next($event)"
  />
</article>

<app-bad-guy-detail-view
  [badGuy]="(selectedBadGuy | async) ?? undefined"
/>
```

➡

```
  <app-bad-guy-list-table-view
    [list]="filteredList()"
    [selectedId]="selectedId()"
    (selectId)="selectedId.set($event)"
  />
</article>

<app-bad-guy-detail-view
  [badGuy]="selectedBadGuy()"
/>
```

```
export class BadGuyListComp
  private loader = inject(Ba
  nameFilter = new FormContr
  selectedId = new Subject<n

  filteredList: Observable<B
    this.nameFilter.valueCha
      startWith(this.nameFi
      debounceTime(250),
      switchMap(searchText =
    );

  selectedBadGuy: Observable
    this.selectedId.pipe(swi
}
```

```
export class BadGuyListComponent {
  private loader = inject(BadGuyLoaderService);
  nameFilter = new FormControl('', { nonNullable: true });
  selectedId = signal<number | null>(null);

  filteredList: Signal<BadGuy[]> = toSignal(
    this.nameFilter.valueChanges.pipe(
      startWith(this.nameFilter.value),
      debounceTime(250),
      switchMap(searchText => this.loader.getList(searchText))
    ),
    { initialValue: [] }
  );

  selectedBadGuy: Signal<BadGuy | undefined> = toSignal(
    toObservable(this.selectedId).pipe(
      filter((id): id is number => id !== null),
      switchMap(id => this.loader.getDetails(id))
    )
  );
}
```

Signals can't do this!*

```
e);
able: true });

tList(searchText))

= toSignal(

ll),
(id))
```

Interop w/ rxResource

```
sortOrder = signal<'asc' | 'desc'>('asc');
usersResource = rxResource({
  request: ()  ({ sort: this.sortOrder() }),
  load   ( quest }) =>
    t .get<BadGuy[]>('/bad-guys', {
       quest.sort },

 boolean> = this.usersResource.isLoading;
 itableSignal<BadGuy[] | undefined> = this.usersResource.value;
rror: Signal<unknown> = this.usersResource.error;
status: Signal<ResourceStatus> = this.usersResource.status;
```

resource(...) for fetch/promise-based

Experimental in Angular 19!

https://angular.dev/guide/signals/resource/

Automatic Migration

$ ng generate @angular/core:signals

// or run individual, focused migrations

$ ng g @angular/core:signal-input-migration

$ ng g @angular/core:signal-queries-migration

$ ng g @angular/core:output-migration

effect()

```
ngOnInit() {
  const filterText: Observable<string | null> =
    this.getFilterText();

  filterText
    .pipe(takeUntilDestroyed(this.destroyRef))
    .subscribe(filterText => this.updateQueryParams(filterText));
}
```

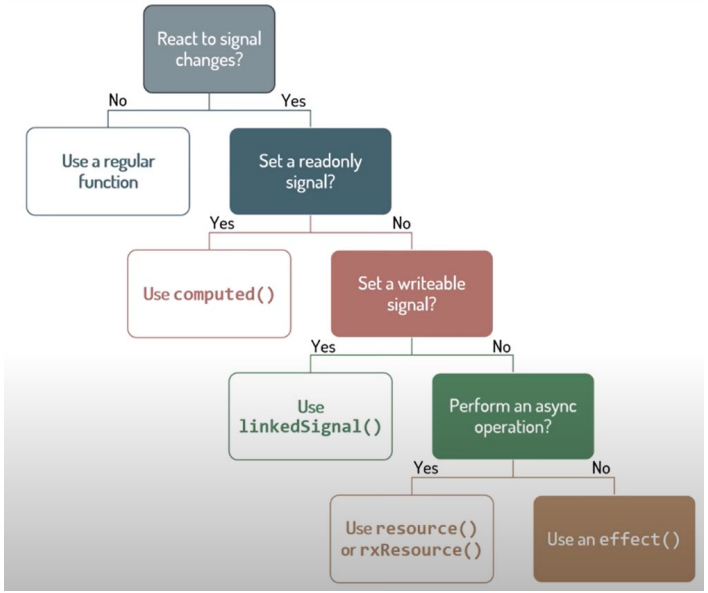
➡

```
ngOnInit() {
  const filterText: Signal<string | null> = this.getFilterText();

  effect(() => this.updateQueryParams(filterText()));
}
```

No subscription
cleanup needed*

https://angular.dev/guide/signals#effects

When to use these functions?



From Deborah Kurata: https://www.youtube.com/watch?v=XWz8pxQWD8c

Signals *and* RxJS

Ben Lesh
@BenLesh

Signals for state. Observables for events. It’s pretty simple. The opposite is silly/bad.

Inputs/Outputs/Queries

```ts
@ViewChild('el') divEl: ElementRef | undefined;
@ContentChildren(BadGuyComponent) badGuys:
  | QueryList<BadGuyComponent>
  | undefined;

@Input({ required: true }) list: BadGuy[] = [];
@Input() selectedId: number | null = null;
@Output() selectId = new EventEmitter<number>();
```

➡

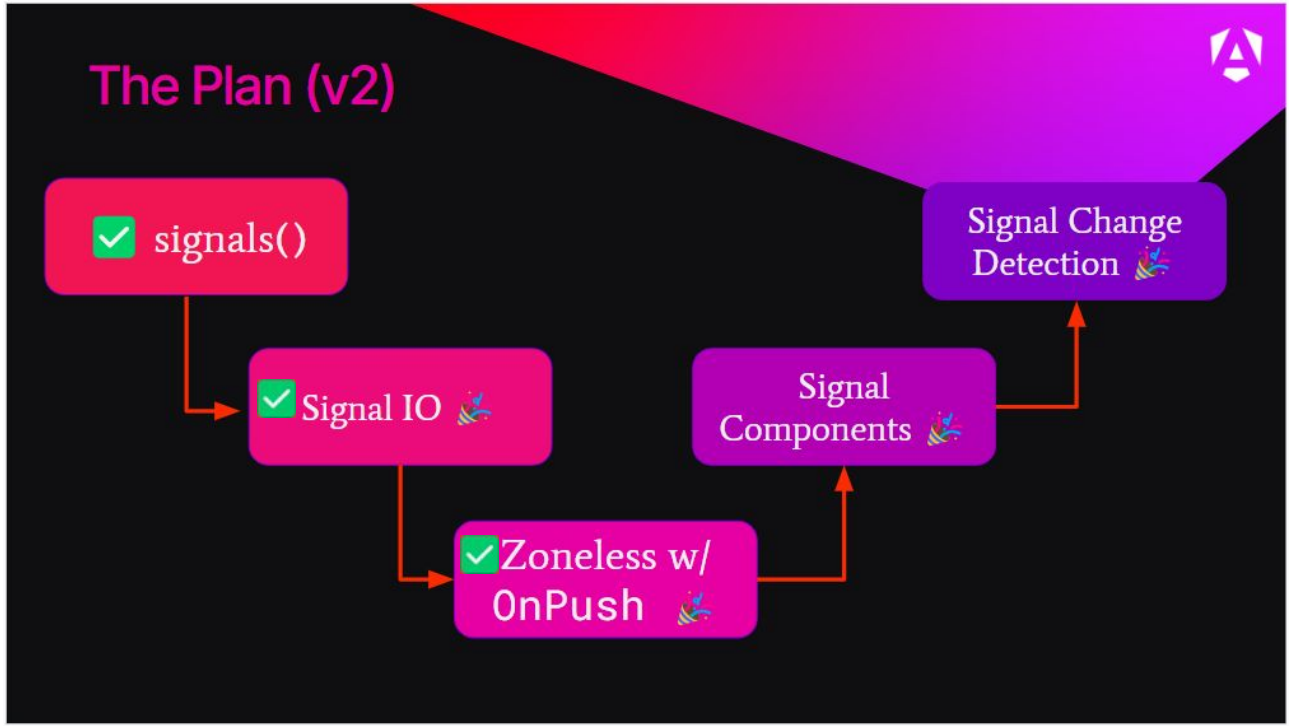
```ts
divEl = viewChild<ElementRef>('el');
badGuys = contentChildren(BadGuyComponent);

list = input.required<BadGuy[]>();
selectedId = input<number | null>();
selectId = output<number>();
```

https://angular.dev/guide/signals/inputs and https://angular.dev/guide/components/output-fn# and https://angular.dev/guide/signals/queries

Signals Status

- Developer preview in Angular 16
- Mostly stable in Angular 19 (not effect, rxResource/resource, or linkedSignal)
- The future?

Roadmap



From Alex Rickabaugh's talk at NgGlühwein 2023: https://t.co/M4vQ5EH4EJ

Many other improvements

- Faster builds with Vite and esbuild
- SSR (Server-Side Rendering)
- Incremental hydration
- New lifecycle hooks
- @let
- Migration schematics
- New documentation website: Angular.dev
- And much more:
  - https://blog.angular.io/introducing-angular-v17-4d7033312e4b
  - https://blog.angular.dev/angular-v18-is-now-available-e79d5ac0affe
  - https://blog.angular.dev/meet-angular-v19-7b29dfd05b84

Thanks for coming!

LMFinney LMFinney

@lmfinney.bsky.social

https://tinyurl.com/codeMashNgRen