Angular
Deliver web apps with confidence

Angular & signals ❤️

Modern Reactivity

iJS Munich

Nov. 14th 2024

Past Present Future

Reactivity

Defining Reactivity

Reactive Programming (RxJS)

- Event based
- Stream of Events
- Order & Time sensitive
- Each individual event that passes through the pipe is processed

Framework reactivity

Render the view when the model is updated

Matthieu Riegler

🇫🇷 / 🇩🇪 👨‍💻 Software Engineer

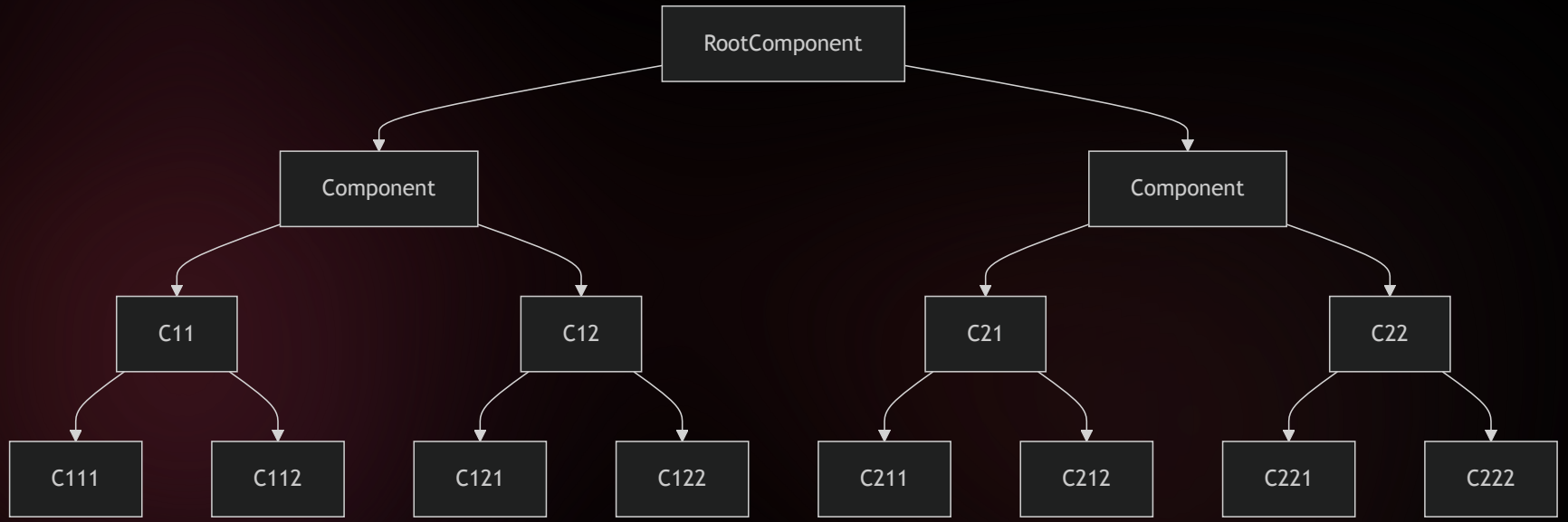
Angular Team / OSS

riegler.fr | Matthieu Riegler | JeanMeche | jeanmeche.com | jean__meche

How does
Angular *refresh views* ?

Change detection

Component Tree



What’s actually an update ?

```
@if(shouldDisplay()) {
    <my-details />
}

@for(item of list; track $index) {
   <my-item [item]="item" />
}

@switch(someValue) {
  @case a: ...
  @case b: ...
}
```

- Creation Mode
- Update Mode

```
function TestCmp_Template(rf, ctx) {
    if (rf & 1) {
      ɵɵtemplate(0, TestCmp_Conditional_0_Template, 1, 0
      ɵɵrepeaterCreate(
        1,
        TestCmp_For_2_Template,
        1,
        1,
        'my-item',
        0,
        ɵɵrepeaterTrackByIndex,
      );
    }
    if (rf & 2) {
      ɵɵconditional(ctx.shouldDisplay() ? 0 : -1);
      ɵɵadvance();
      ɵɵrepeater(ctx.list);
    }
}
```

Demo

Angular Template Compiler - based on 18.2.6

Template

```
<div>This is a static text node.<br> And this is a dynamic text node: {{test}}</div>
@if(isVisible) {
  <span>Some label</span>
}
```

Select a template

default

Prettify ✨

Share Example

The compiled template

```
1  ɵɵdefineComponent({
2    type: TestCmp,
3    selectors: [['test-cmp']],
```

jeanmeche.github.io/angular-compiler-output/

How does Angular *schedule* change detection ?

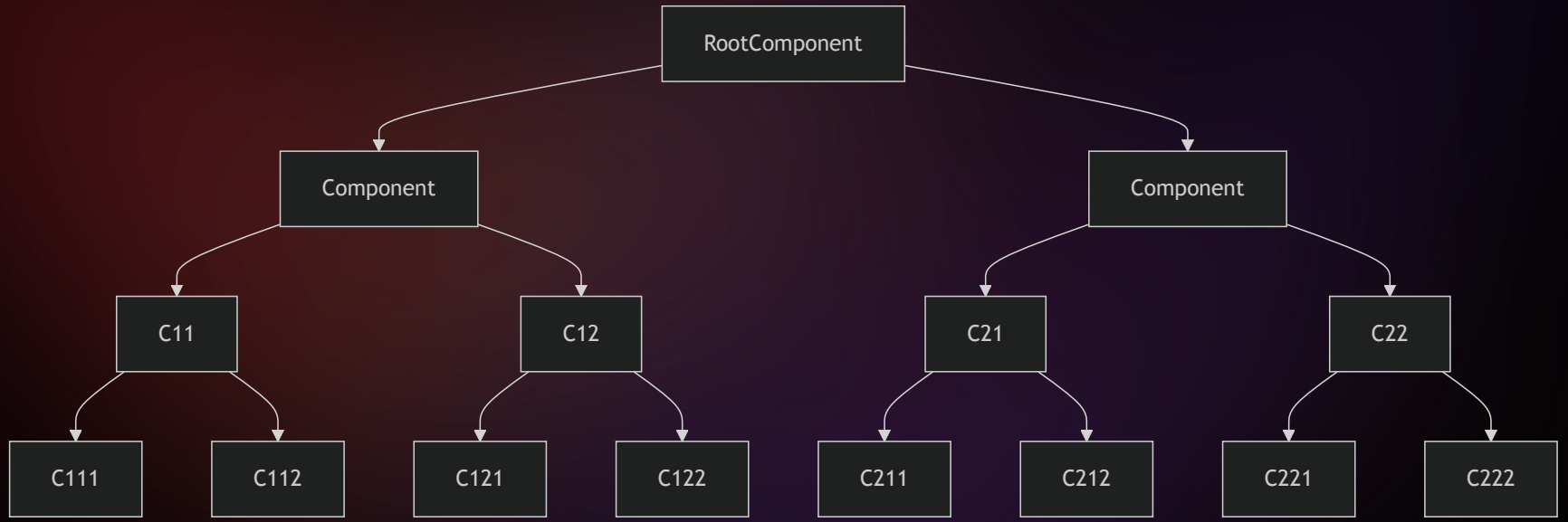
Zone.js

An execution context

- Patches asynchronous APIs
- Angular is notified when one of those APIs are invoked
- Angular *schedules* Change Detection

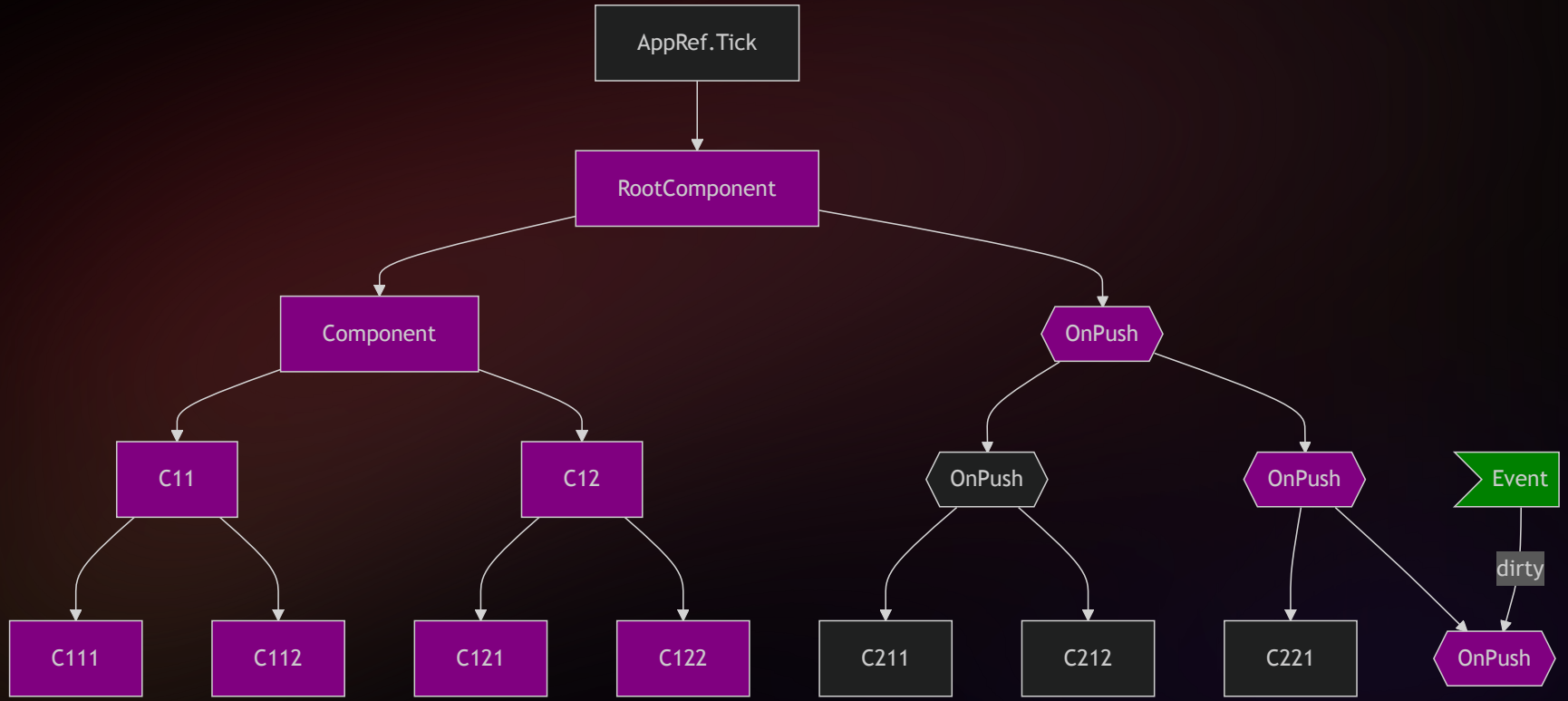
Change detection

Component Tree



Optimisations

`ChangeDetectionStrategy.OnPush`



Zoneless

What are you talking about ?

A question of when

Zone.js is Angular's scheduler

Downsides of Zone.js

- 30 kB, eagerly loaded
- Zones, async runtime context
- Messy stack traces
- No support for native async/await

How do we replace zone.js ?

A new scheduler

rAF-race

`setTimeout()` + `requestAnimationFrame()`

Scheduler triggers

- AsyncPipe (`markForCheck`)
- Template Event Listeners
- Signals

Enable zoneless

```
bootstrapApplication(AppComponent, {
  providers: [
    // Experimental provider in v18/v19
    provideExperimentalZonelessChangeDetection()
  ]
})
```

Modern reactivity

Signals 

Angular Signals

- Wrapper around a value (primitive or complex structure)
- Granularly tracks state changes

```
const counter = signal(0);

function reset() {
  counter.set(0);
}

function increment() {
  counter.update(value => value +1);
}
```

A Dependency tree

The signals implementation is defined in terms of two abstractions.

- Producers
- Consumers

Producers

Represent values which can deliver change notifications.

Consumers

Represent a reactive context which may depend on some number of producers.

Producers produce reactivity,

Consumers consume it

Consuming & deriving state

```
// A kind of producer
counter = signal(0);

// Another kind of producer, derived from the previous one
doubleCounter = computed(() => counter() * 2);
```

Note that the dependency are implicit.

```
doubleCounter = computed(() => {
  if(true) {
    counter() * 2
  } else {
    // never consumed
    someOtherSignal()
  }
});
```

Component templates

```
<section>
  value:  {{ counter() }}
  double value:  {{ doubleCounter() }}
</section>
```

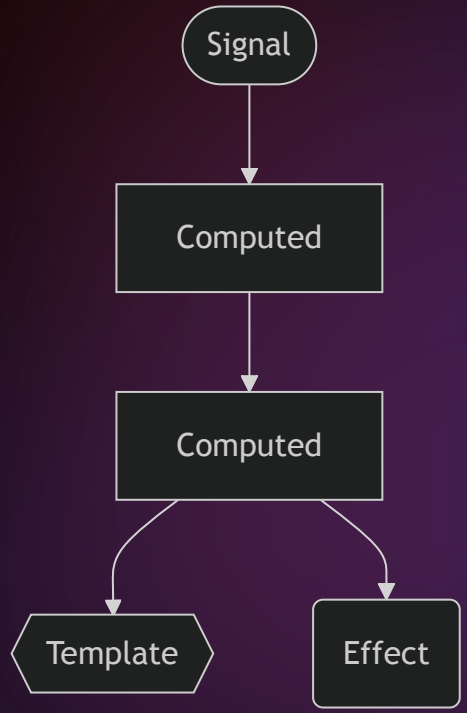
Linking the reactive word

```
effect(() => {
  const state = chartState();

  myChartLib.set(state);
})
```

`effect` is how you link reactive things to non-reactive things.

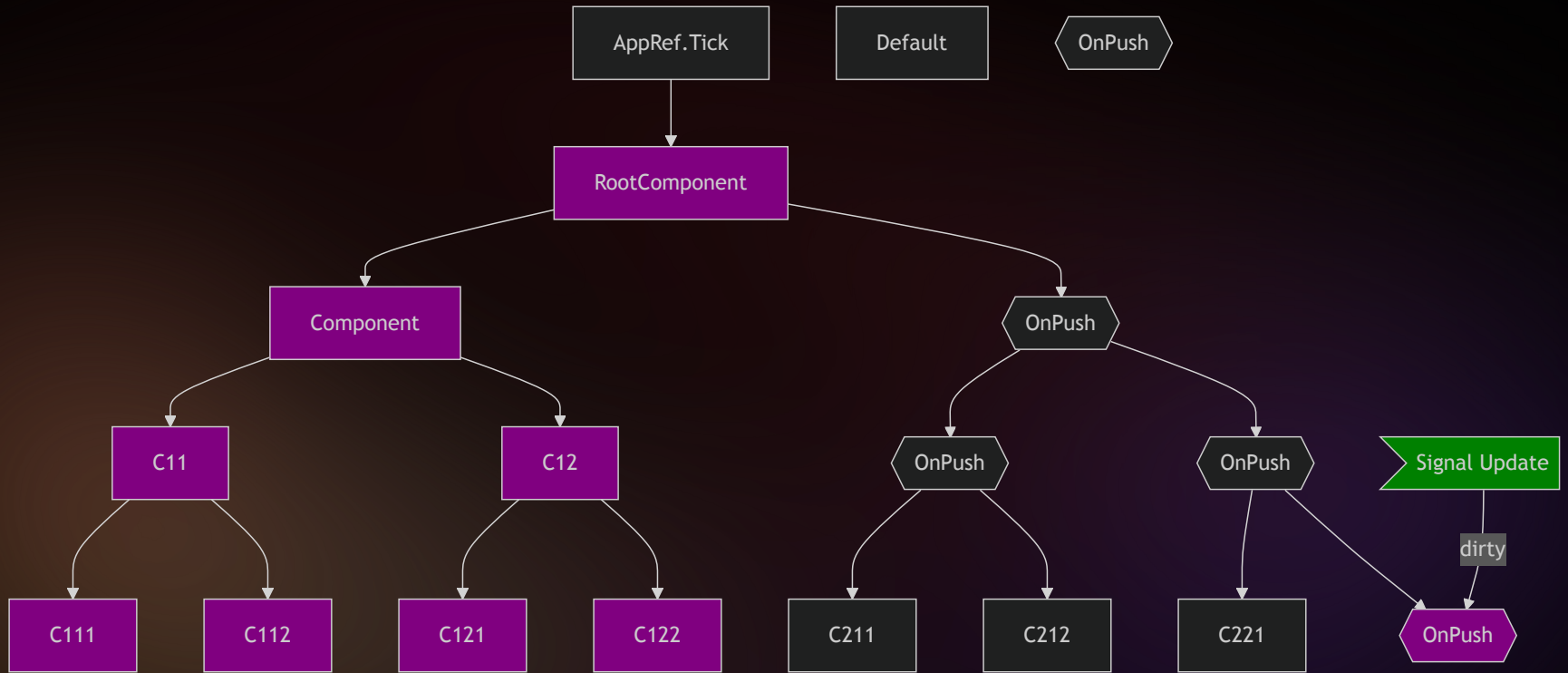
A reactive tree



Angular understands
the signal graph

Local Change Detection

Powered by signals



Demo

Trigger global actions

AppRef.tick() | setTimeout(...) | Click

Control dirty check coloring

Clear dirty check coloring automatically | Clear

Play with component input - current value: n/a

Trigger change | change input object (ref-change) | mutate input object (no ref-change) | via Observable (no ref-change) | Propagate in Angular zone

Default

ngOnChanges attached

input value:

object prop:

observable:

local signal: 0

Click

OnPush

jeanmeche.github.io/angular-change-detection/

Root effect / View Effect

Root effect

- top-level effects within an application
- independent from component updates
- runs on `ApplicatinRef.tick()`

```
@Injectable()
class MyStateService {
  constructo() {
    effect(() => { /* ... */ })
  }
}
```

Usecases

- Propagating state changes (when computeds are not an option).
- Synchronizing state with the backend or some local storage
- Rendering not tied to a component
- Logging/Debugging

View Effect

- Components depend on inputs
- `effect()` can affect component/child state

```
const show = computed(() => !!cond());
let view;
effect(() => {
  if (show()) {
    view = this.vcr.createEmbeddedView(this.childView);
  } else {
    view.destroy();
  }
});
```

Important to execute the effect ahead of the component CD.

afterRenderEffect() Experimental

Effect x DOM Access hook.

```
afterRenderEffect({
  earlyRead: () => ...,
  write: () => ...,
  mixedReadWrite: () => ..,
  read: () => ...,
});
```

Designed to access DOM after Angular rendered and updated it.

Error handling

`effect()` are part of CD, they throw on top.

```
it('should throw error...', () => {
  // create an effect that throws
  const appRef = TestBed.inject(ApplicationRef);
  effect(
    () => {
      throw new Error('fail!');
    },
    {injector: appRef.injector},
  );

  // explicitly run the CD and check for the thrown exception
  expect(() => appRef.tick()).toThrowError('fail!');
});
```

Effect test with `ErrorHandler`

```
it('should throw error...', () => {
  let lastError: any = null;
  class FakeErrorHandler extends ErrorHandler {
    override handleError(error: any): void {
      lastError = error;
    }
  }

  TestBed.configureTestingModule({
    providers: [{provide: ErrorHandler, useFactory: () => new FakeErrorHandler()}],
    // we make sure to prevent tick() from throwing.
    rethrowApplicationErrors: false,
  });

  // create an effect that throws
  const appRef = TestBed.inject(ApplicationRef);
  effect(
    () => { throw new Error('fail!'); },
    {injector: appRef.injector},
  );

  // explictly run the CD
  appRef.tick();
  expect(lastError.message).toBe('fail!');
});
```

Future of reactivity

linkedSignal Experimental

A writable, derived & locale state

```
import { Component, linkedSignal, signal } from '@angular/core';

@Component({
  template: `<p>First book in list: {{ firstBook() }}</p>`
})
export class BookListComponent {
  books = input<string[]>();

  firstBook = linkedSignal({
    source: this.books,
    computation: books => books[0]
  });

  // this also works (shorthand notation)
  // firstBook = linkedSignal(() => this.books()[0]);

  overrideFirstBook() {
    this.firstBook.set('jQuery');
  }
}
```

Resource Experimental

Asynchronous resource loading using Signals

```
bookResource = resource({
  request: () => selectedBookId()
  loader: (param) => bookService.getBook(param.request)
})
```

The loader is triggered every time the request dependencies change.

```
// reloading
bookResource.reload()

// set a local state
bookResource.set({ /** bookObject */})

bookResource.hasValue()
bookResource.isLoading() //
bookResource.asReadonly()
```

Forms Prototyping stage

Prototype 🧪

```
@Component({
  template: `
    <input [field]="form.field.firstName" />
    <input [field]="form.field.lastName" />
  `
})
class UserProfile {
  data = signal({firstName: 'Matt', lastName: 'Riegler'})

  // Form is derived from data
  form = autoForm(this.data)
}
```

Powered by signal, unified, flexible and interoperable

Router Brainstorming stage

Prototype 🧪

Router coordinates data fetching

Rendering becomes synchronous

Resource API ties them together

Signals for everyone

TC39 proposal to land Signals into the platform

github.com/tc39/proposal-signals

Thank you!

Slides on riegler.com