Understanding Core Concepts of Cypress

Marko Vajs
Software Development Engineer in Test

Module Overview

Learn about conditional testing and how to avoid common pitfalls

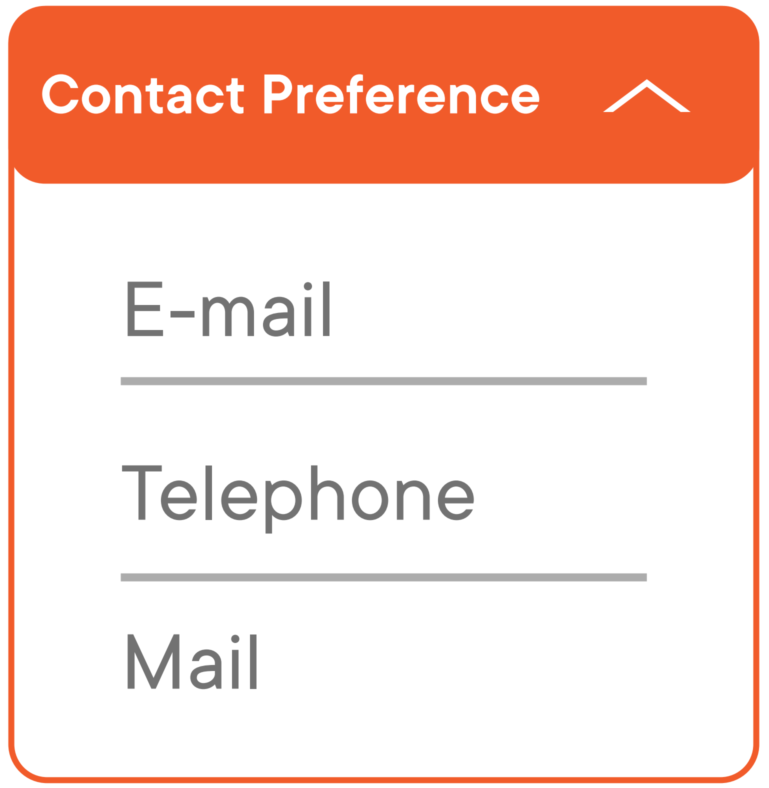
Explore the asynchronous nature of Cypress

Get to know Cypress's built-in retry-ability

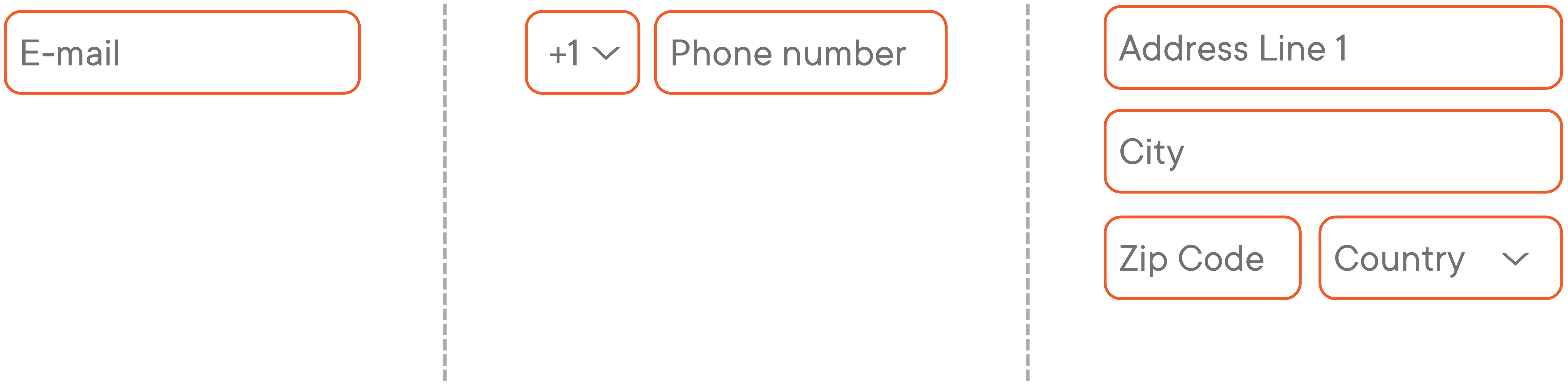
Conditional Testing

Conditional Testing

```
IF (<Condition>)
    THEN <Statements>
    ELSE <Statements>
ENDIF
```



```
if (contactPreference === "E-mail") {
   ...
} else if (contactPreference === "Telephone") {
   ...
} else if (contactPreference === "Mail") {
   ...
}
```



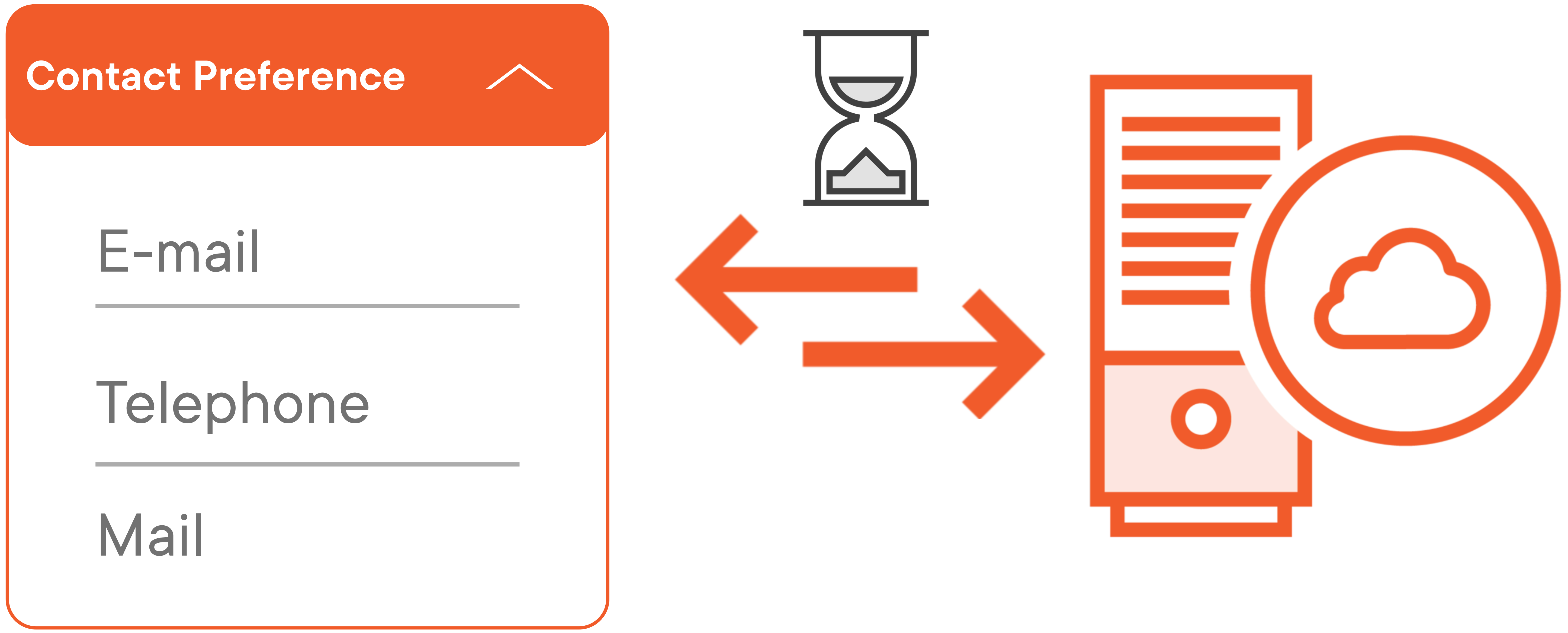
Is It Possible to Make Conditional Tests Consistently?

Conditional testing can only be used when the state of the application has stabilized.

No pending network requests

No timeouts and intervals

No async/await code


Contact Preference
E-mail
Telephone
Mail

In highly-dynamic applications, content updates are based on events that the user usually cannot control.

Strategies to Overcome Non-deterministic Behavior

- **Remove the need to do conditional testing**
- **Make application behave deterministically**
- **Check other sources of truth (like server or database)**
- **Embed data into other places you could read off**
- **Add data to the DOM that you can read off to know how to proceed**

Conditional testing is a test design consideration.

Mixing Synchronous and Asynchronous Code

Cypress is built to match the asynchronous nature of modern web applications.

```
cy.get('button.primary').click();
```

Cypress commands do not return their subjects. They yield them.

```
it('test', () => {
    let username = undefined;

    cy.visit('http://localhost:4200');
    cy.get('#username').then(($el) => {
        username = $el.text();
    });

   if (username) {
     cy.contains(username).click();
   } else {
     cy.contains('My Profile').click();
   }
});
```

```
it('test', () => {
    let username = undefined;

    cy.visit('http://localhost:4200');
    cy.get('#username').then(($el) => {
        username = $el.text();
       if (username) {
           cy.contains(username).click();
       } else {
           cy.contains('My Profile').click();
       }
    });
  });
```

Demo

Mixing synchronous and asynchronous code

Exploring Retry-ability

```
cy.get('.main-list li') // command
.should('have.length', 3) // assertion
```

The retry-ability allows tests to complete each command without hard-coding waits.

Cypress only retries commands that query the DOM.

`.get()` `.find()` `.contains()`

Commands that are not retried are the ones that could potentially change the state of the application (e.g., click).

Changing the Timeout

The default timeout can be changed on a command level or globally.

cypress.json

```json
{
    "defaultCommandTimeout": 0
}
```

adventure.spec.js

```javascript
cy.contains('More Details')
    .click({ timeout: 0 });
```

```
cy.get('ul li') // yields only one <li>
     .find('p') // retries multiple times on a single <li>
     .should('contain', 'Filtered by') // never succeeds
```

Demo

Exploring retry-ability

Module Summary

Module Summary

Cypress commands are asynchronous

Conditional testing requires a stable source of truth

Cypress retries some commands by default, so there is no need for hard-coding waits

Up Next:
Leveraging App Actions and Page Objects