
expoQA®26
MADRID 26th, 27th & 28th May
expoqa.eu

From Polling to Events: The Future of Browser Automation

Diego Molina
Selenium Project Lead | Sr, Field Engineer at Sauce Labs

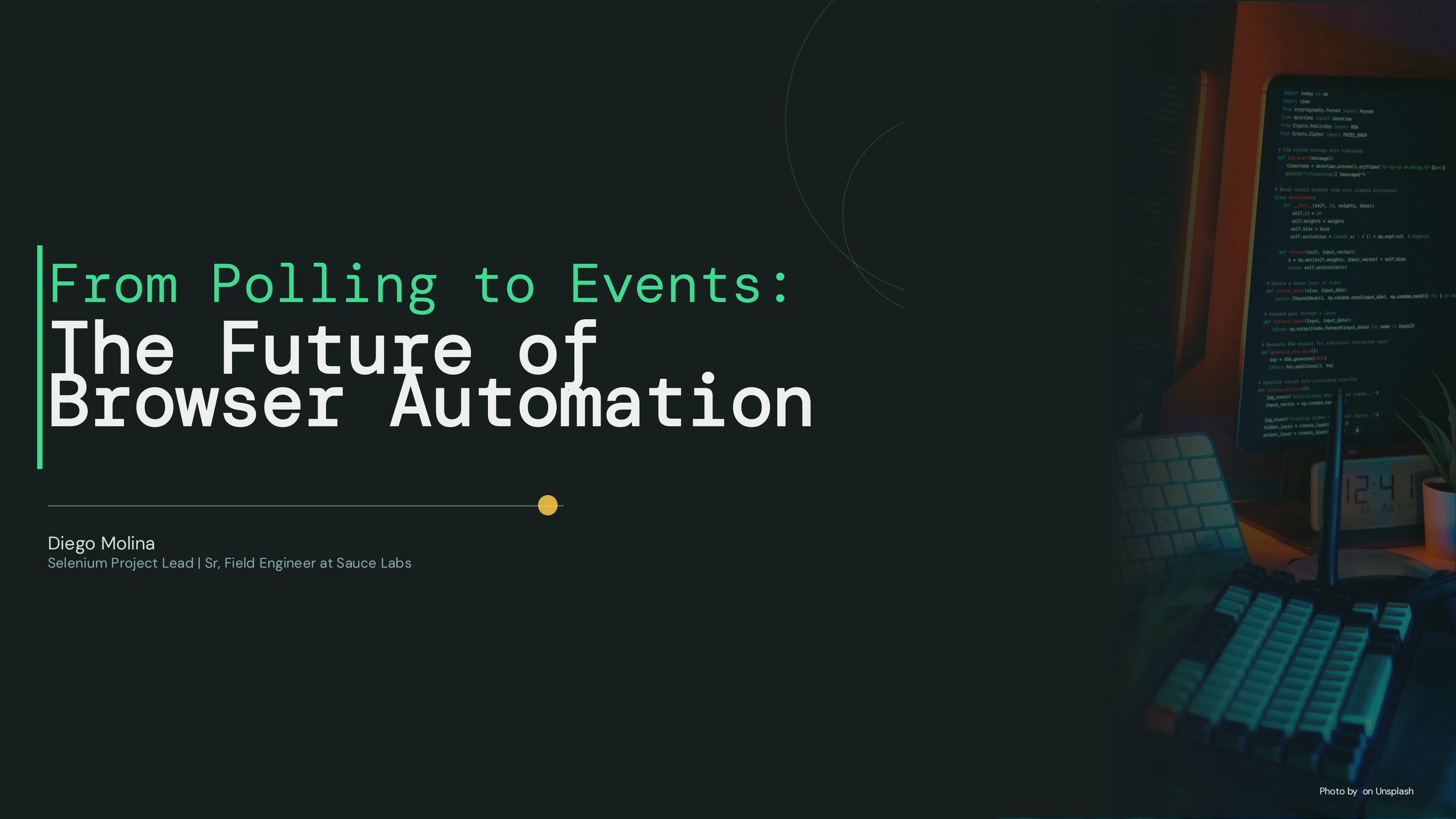


Photo by on Unsplash

About Me

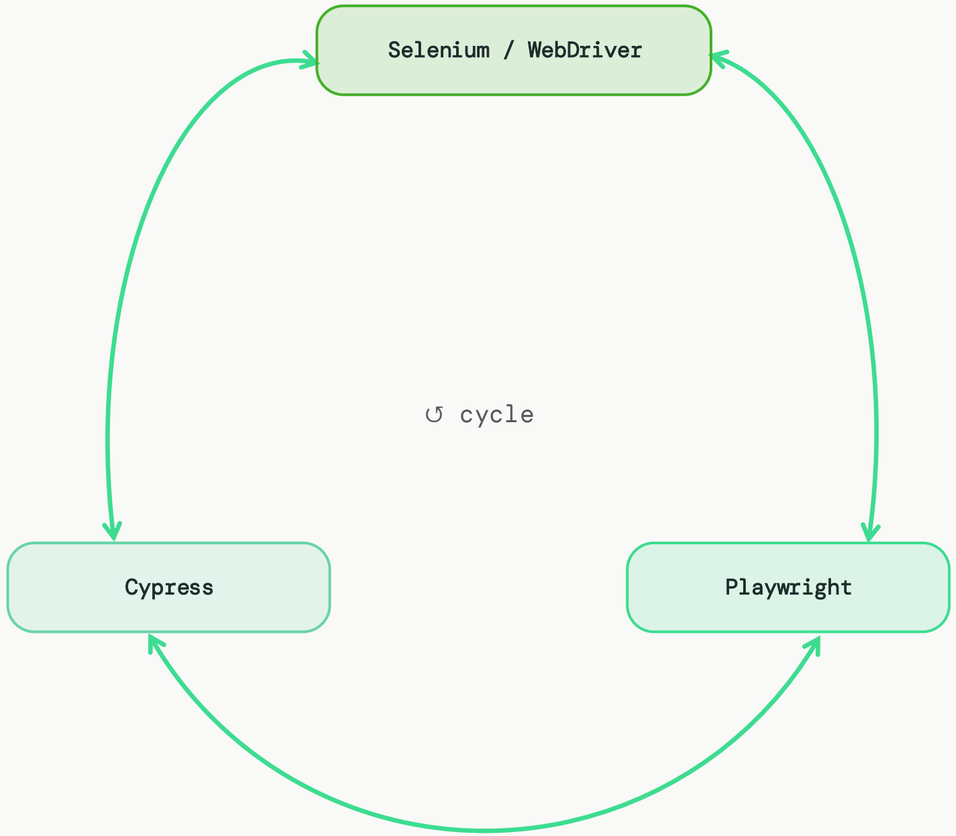
Selenium Project Lead
Senior Field Engineer at Sauce Labs

- Testing, browser & mobile automation
- Open source & conferences
- linkedin.com/in/diemol

THE INDUSTRY SITUATION

Teams keep **switching tools.**

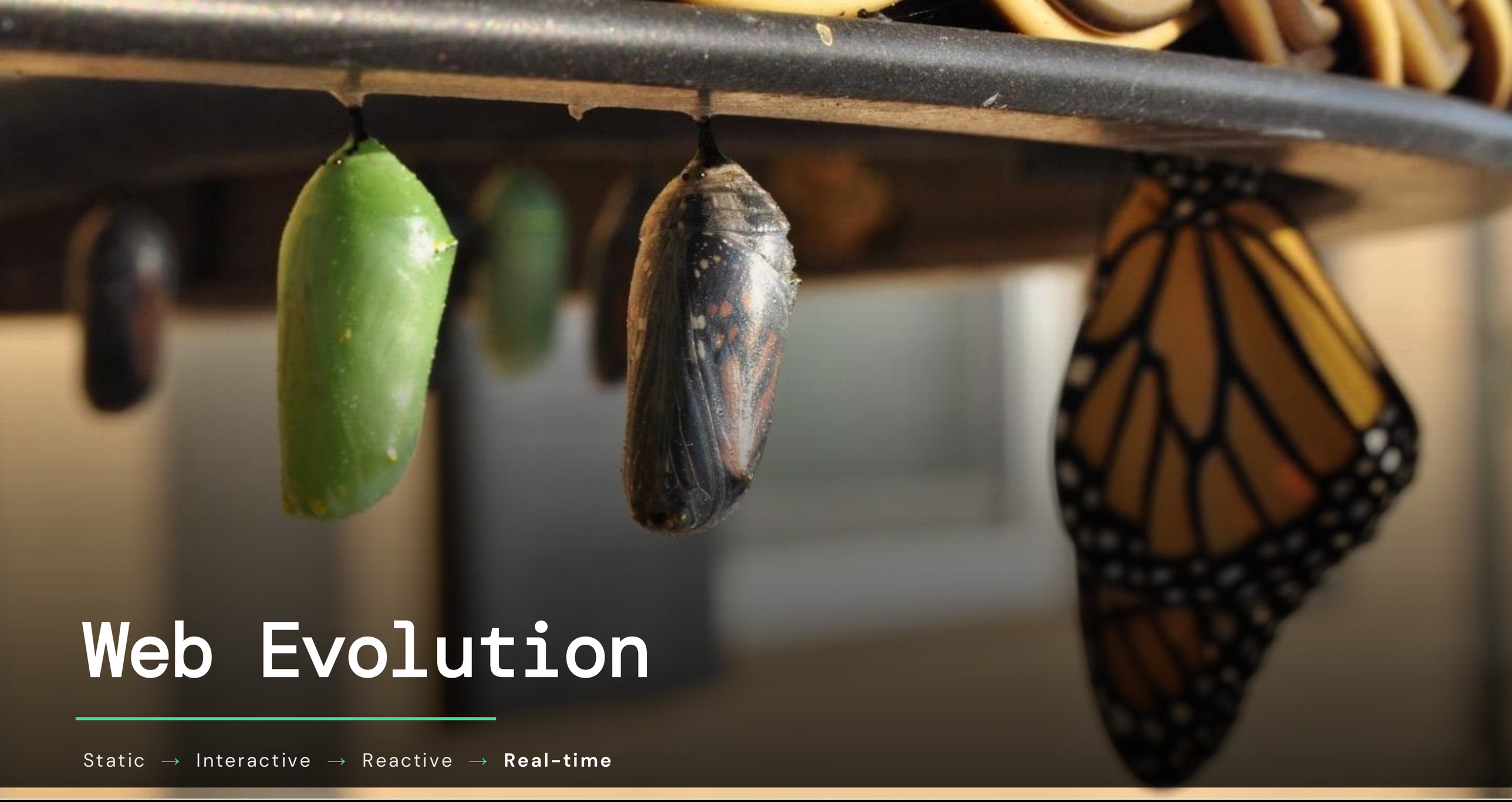
But they keep the **same approach.**



Agenda

1. The problem
2. Why polling breaks
3. Browser events
4. Selenium, Playwright & Cypress
5. Practical patterns
6. Conclusions

/The Web Changed/



Web Evolution

Static → Interactive → Reactive → **Real-time**

Web Evolution

01
Static Websites

HTML / CSS / JS

- → Server-rendered pages
- → jQuery DOM manipulation
- → Page-per-request navigation

02
Interactive Apps

Angular / React / Vue

- → Component-based UIs
- → Client-side routing (SPA)
- → REST API consumption

03
Modern Reactive Web

Playwright / Cypress / WebDriver

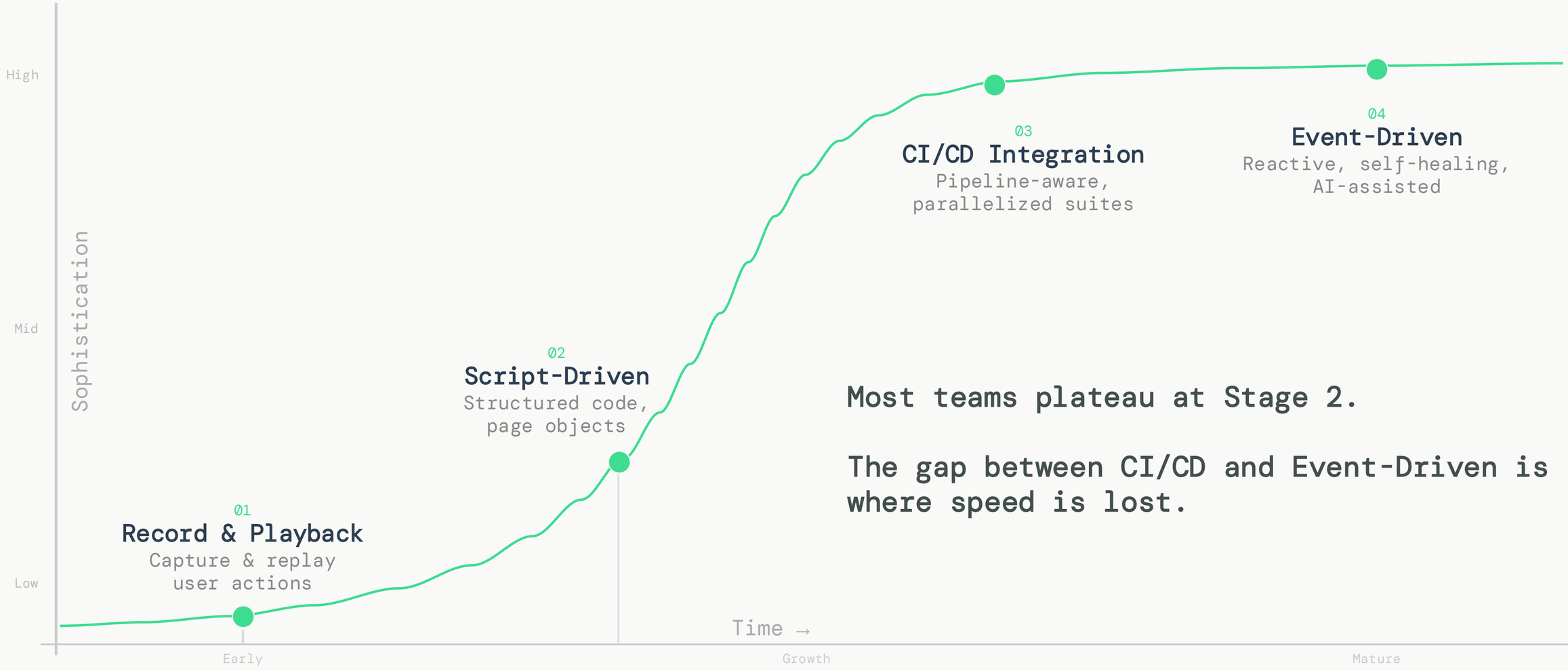
- → Event-driven architecture
- → Real-time state sync
- → Automated E2E testing

Testing Did Not Change Enough

The modern web has evolved. The automation approaches didn’t.

Most tests are still **reactive.**

The Automation Maturity Curve



Most teams plateau at Stage 2.

The gap between CI/CD and Event-Driven is where speed is lost.

The Polling Problem

Traditional Automation

sequential · reactive · procedural

```
wait()
|
click()
|
wait()
|
assert()
```

// reacts after the fact

Most tests still operate by checking and reacting after the fact.

Why Polling Fails Today

THEN

- Server-rendered pages
- Simple page lifecycle
- Predictable rendering
- Few async operations
- Limited background activity

→

NOW

- Dynamic rendering
- APIs & microservices
- Real-time DOM mutations
- Background requests
- Client-side state changes
- Event-driven UI behavior

CONSEQUENCES

brittle timing | missed states | flaky tests | difficult debugging

"The browser became asynchronous. Most tests did not."

Visibility Is Not Readiness

VISIBILITY

You can see it.

- Dashboards show red/green
- Logs capture events after they happen
- Metrics aggregate what has already occurred
- Alerts fire when thresholds are crossed

VS

READINESS

Can you act on it?

→ Tests respond to events in real time

→ The state is observed before it changes

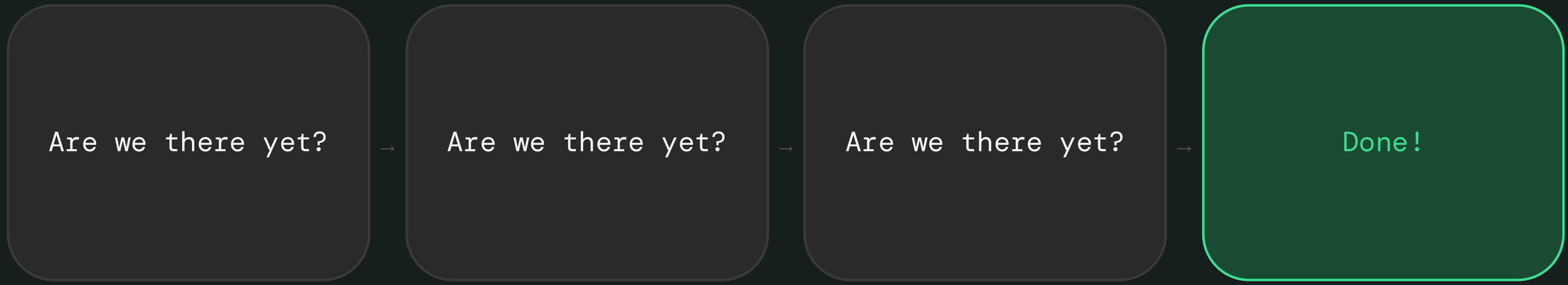
→ Assertions are predictive, not reactive

→ Failures surface intent, not timing

Readiness means being able to respond — before it's too late.

POLLING BEHAVIOUR

Checking after the fact.



↻ polls sent before state changes

page load: 65%...

test: waiting for 100%

Still loading... (poll #14)

Polling asks repeatedly: Did it happen yet?

Events notify immediately: it just happened.

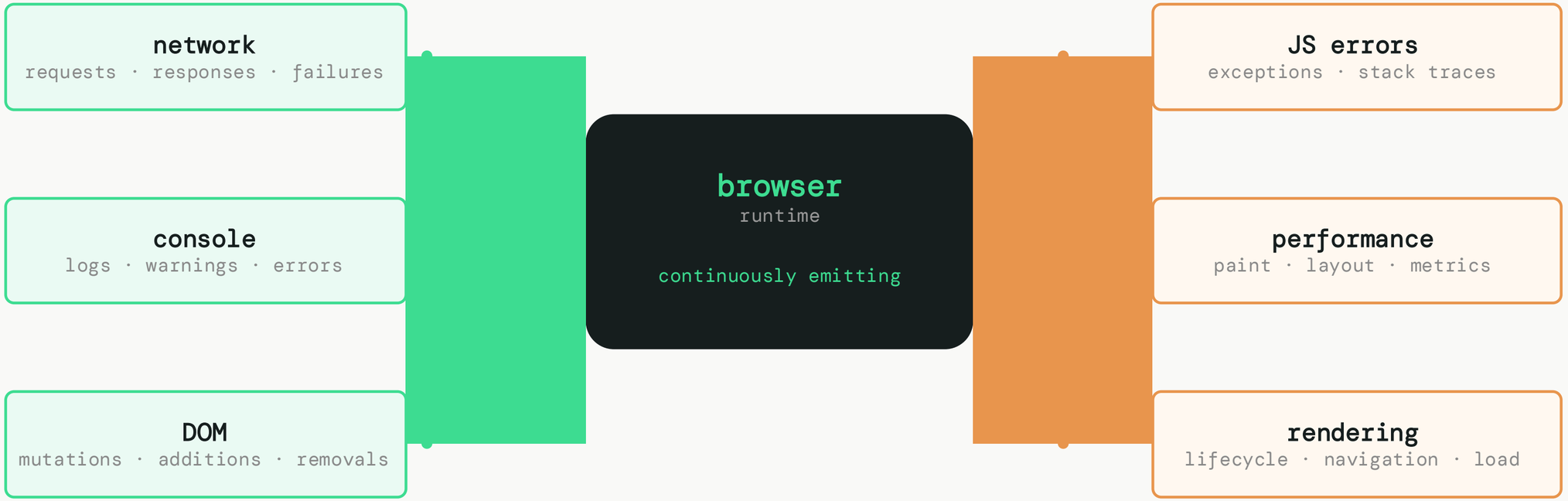
Stop checking.

Start listening.

[Browser Events]

Browsers Already Expose Events

The runtime already tells you what is happening.



Automation no longer needs to guess. The browser already knows.

POLLING

Did it happen?

→ check...

→ check...

→ check...

→ check...

still waiting...

Polling asks repeatedly.

EVENTS

It happened.

▶ event received

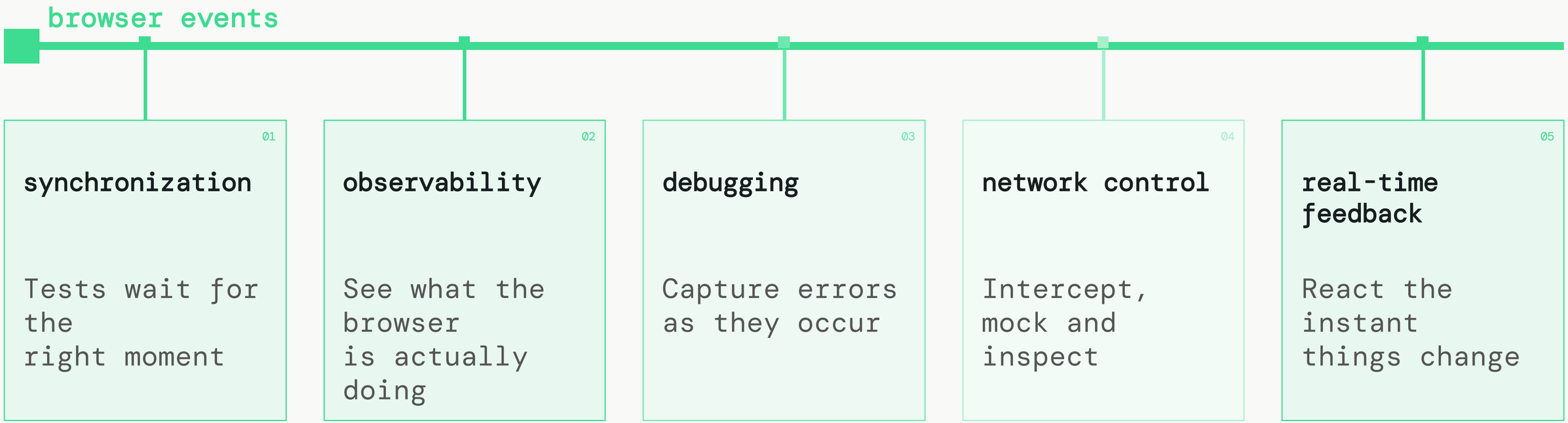
networkIdle → DOM ready → assert

listening.

Events notify immediately.

What Events Enable

Events turn the browser into an observable system.



Modern automation reacts instead of guessing.

Event-Based Mental Model

OLD

wait

↓

act

↓

assert

timing-dependent · sequential · rigid

vs

NEW

subscribe

react

observe

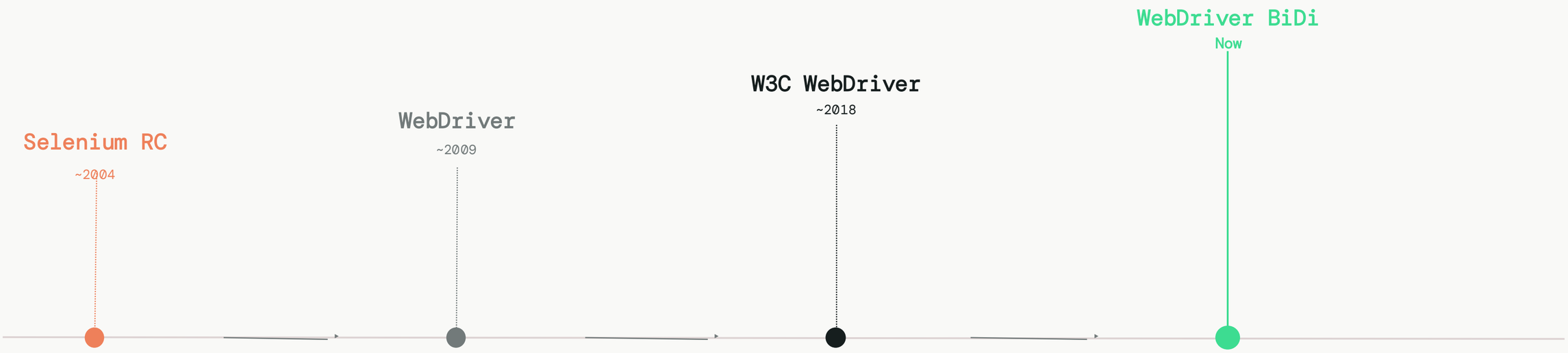
event-driven · responsive · real-time

We are shifting from controlling timing to reacting to signals.

Selenium/WebDriver, Playwright & Cypress

Selenium’s Evolution

From browser control to browser observability



Each stage solved the limitations of the previous generation.

What Is WebDriver BiDi

Bidirectional communication between test and browser.

TRADITIONAL WEBDRIVER

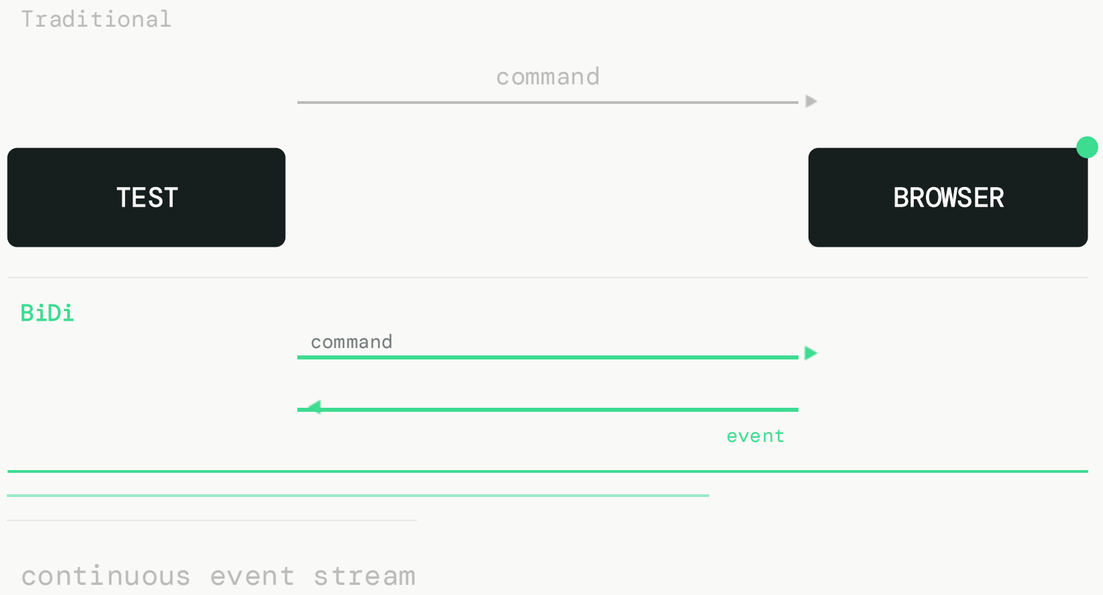
Test → Browser

- request / response
- sequential commands
- polling-based interaction

WEBDRIVER BIDI

Test ↔ Browser

- bidirectional communication
- browser events
- real-time notifications
- observability



WebDriver BiDi becomes a conversation, not just a set of commands.

Playwright’s Model

Built around direct access to browser capabilities and browser signals.

BROWSER COMMUNICATION

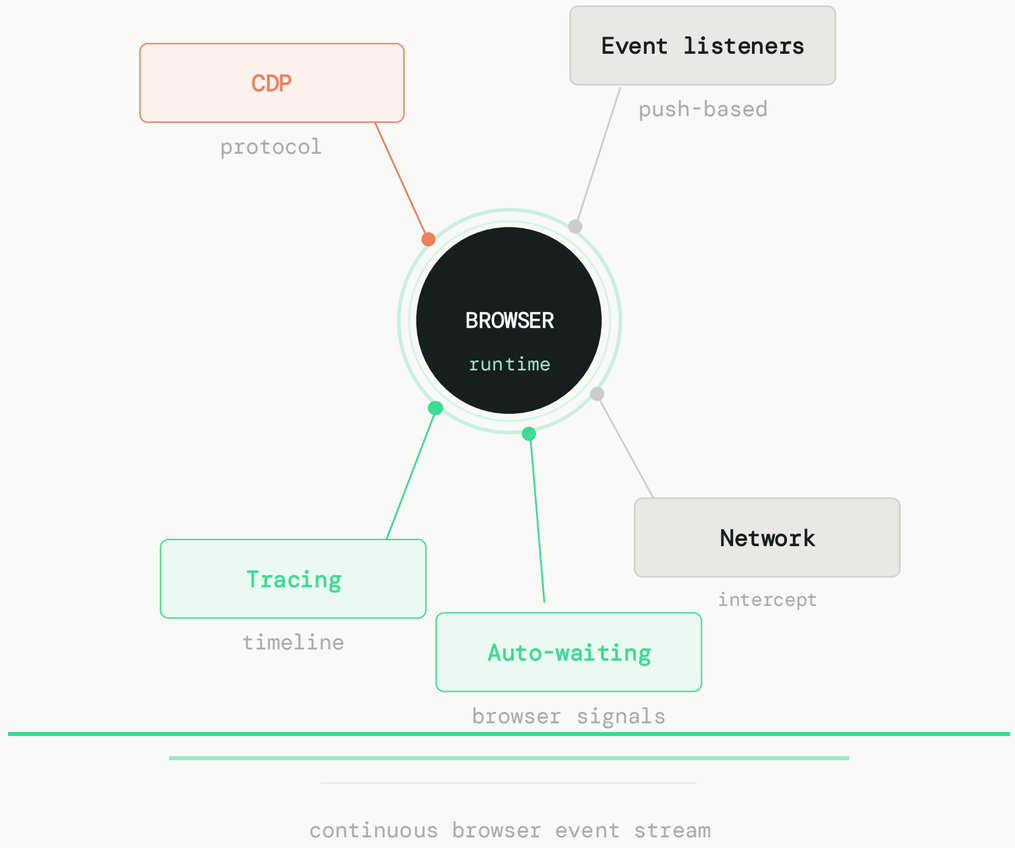
CDP + Event listeners

- **Direct protocol access** Chrome DevTools Protocol channel
- **Browser-native events** DOM, network, console, dialog
- **Push-based signals** no polling required

OBSERVABILITY & RELIABILITY

Tracing · Network · Auto-wait

- **Tracing** full browser activity timeline
- **Network interception** intercept, mock, or observe requests
- **Auto-waiting** reacts to browser state signals
- **HAR recording** captured network activity for replay



Playwright embraces browser observability by design.

Cypress’s Model

Runtime visibility and immediate feedback inside the browser.

BROWSER RUNTIME

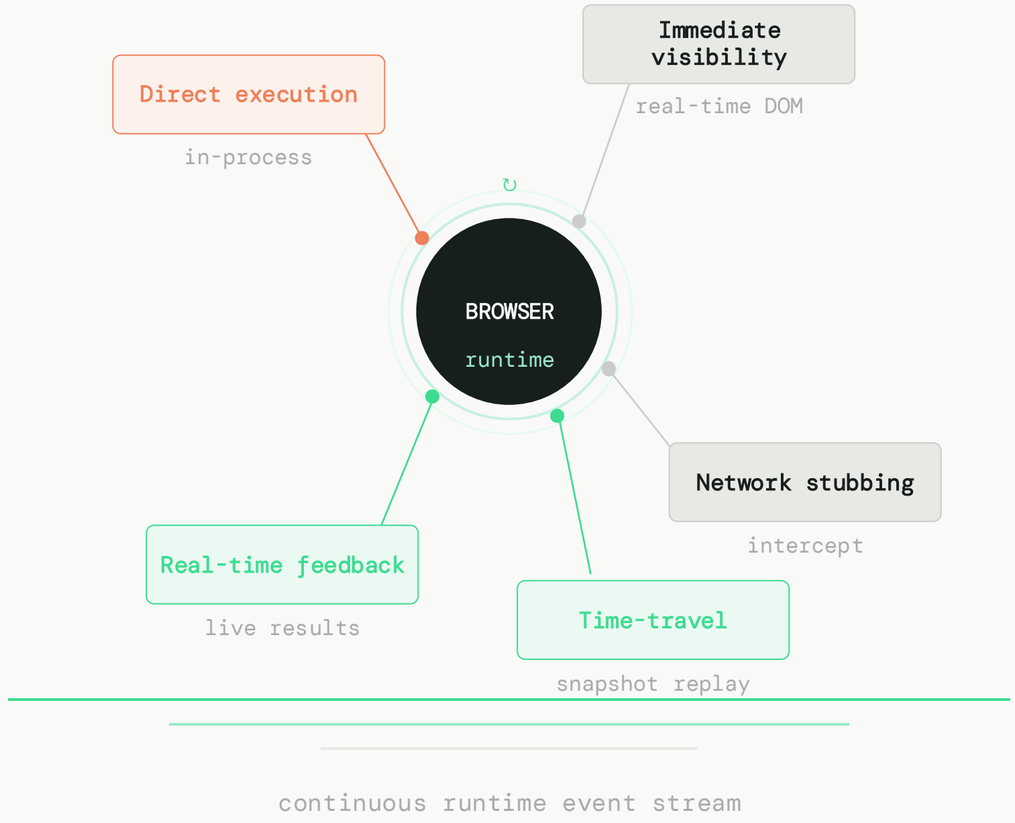
Direct execution inside browser

- **In-process execution** runs inside the browser context directly
- **Immediate visibility** DOM, network, and events in real time
- **No cross-origin gap** test and app share the same runtime

CONTROL, FEEDBACK & OBSERVABILITY

Stub · Inspect · Replay

- **Network stubbing** intercept and mock any request in-flight
- **Real-time feedback** live test results as the browser executes
- **Time-travel debugging** step through snapshots of browser state
- **Execution visibility** every command logged with DOM context



Cypress emphasizes immediate visibility into browser behavior.

Industry Convergence

Three tools. Independent paths. The same destination.

Selenium/WebDriver

BiDi Protocol

WebDriver BiDi

bidirectional protocol

Browser events

real-time notifications

Observability

network, console, DOM

Playwright

CDP + Events

CDP

direct browser protocol

Event listeners

push-based signals

Tracing

full browser timeline

Cypress

Runtime Visibility + CDP

Runtime visibility

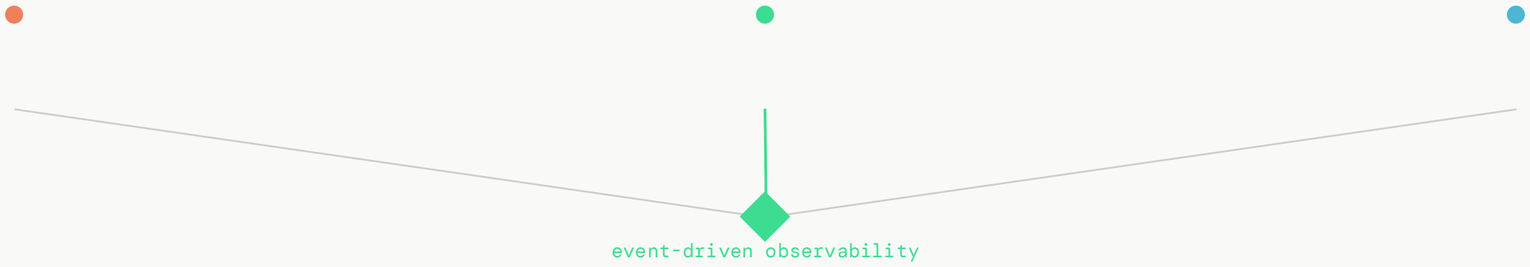
in-process execution

Real-time feedback

immediate loop

Network control

intercept & stub



Different implementations. Same direction.

The tooling differs more than the philosophy.

narrative shift

This Is NOT a Tool War

Selenium · Playwright · Cypress

is the wrong conversation.

The real shift is **reactive** vs **event-driven** automation.

Reactive vs. Event-Driven

The Real Conversation

{Common Patterns}

Common Patterns Revisited

the shift in perspective

Same problems.

Different approach.

Revisiting real testing scenarios through an **event-driven lens.**

Network Interception

OLD

proxy tools

external HTTP proxy setup

backend dependency

server-side routing required

external setup

additional infra overhead

indirect control

disconnect from browser

indirect · external · infrastructure-heavy

NEW

listen

observe every request in real time

intercept

capture and inspect network calls

mock

replace responses with controlled data

browser-aware automation

direct · real-time · browser-native

supported by

Selenium | Playwright | Cypress

Modern frameworks can observe and control network traffic **directly.**

Demo

Authentication

OLD

UI login every test

repetitive setup

same credentials, same UI, every run

slow execution

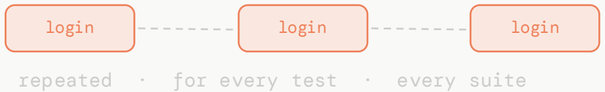
login forms add latency to every suite

authentication blocker

failures cascade through all tests

brittle by design

UI changes break auth for every test



NEW

tokens

persist auth state without UI

cookies

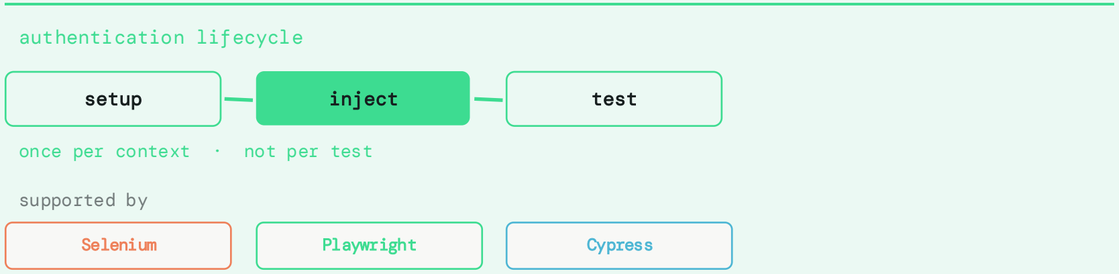
inject session directly into context

session injection

pre-authenticate before test starts

network hooks

intercept and handle auth requests



Authentication can become setup **instead of a bottleneck.**

Demo

Element Synchronization

OLD

sleep()

fixed delay — always wrong

retry()

polling until timeout

implicit wait()

global timeout applied to every element

timing guesswork

repeat until timeout · or just fail

NEW

DOM events

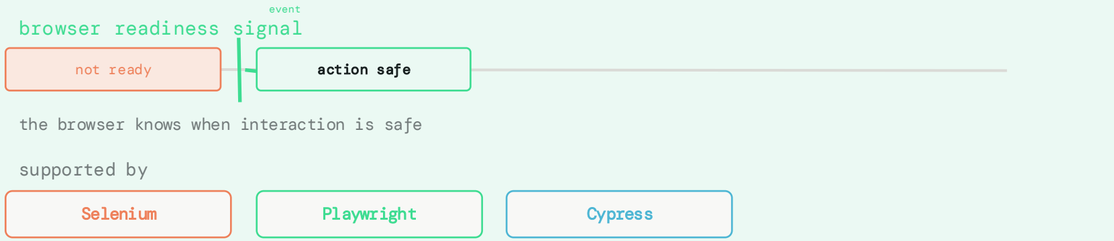
react when the element actually changes

network completion

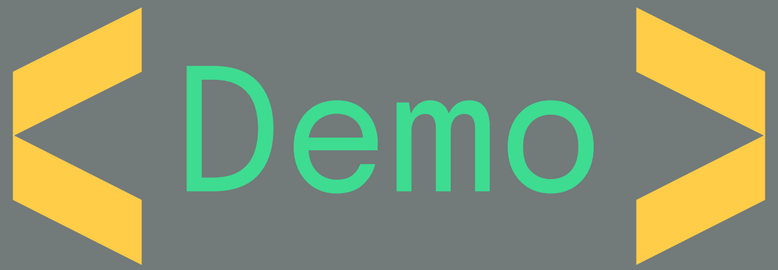
wait for requests the page depends on

actionability

visible, stable, and enabled — all three



Visibility alone does not guarantee readiness. **The browser already knows.**


Demo

Debugging

OLD

rerun and hope

no replay

cannot reproduce what actually happened

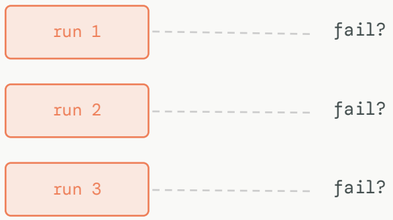
inference only

read the error, guess the cause

timing-dependent

failures may not recur on next run

the debugging loop



no visibility · only outcomes

NEW

console

live browser log stream

tracing

full action timeline with screenshots

HAR

complete network archive for replay

network events

every request and response observed

runtime metrics

CPU, memory, frame rate during test run

Modern debugging is **observability.**

Demo

The future of testing

Observability

Testing becomes:

diagnosis

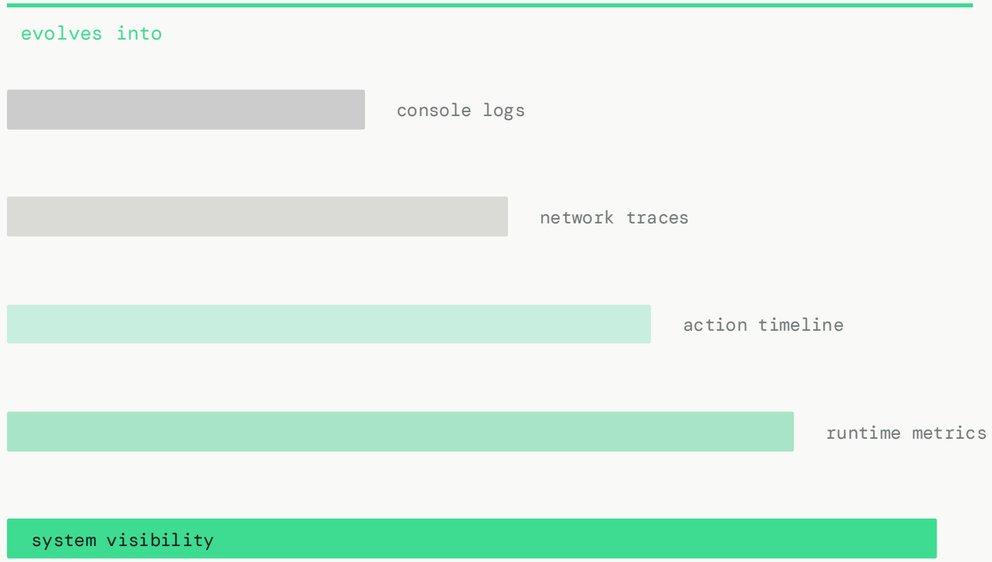
analytics

system visibility

beyond pass and fail · into runtime understanding

Not just

pass / fail.



Modern automation helps **explain systems,** not only validate them.

Conclusions

Key Takeaways

01

The web became event-driven

Modern browsers communicate state in real time — automation must listen

02

Testing must evolve too

Reactive, observable automation replaces timing-based execution

03

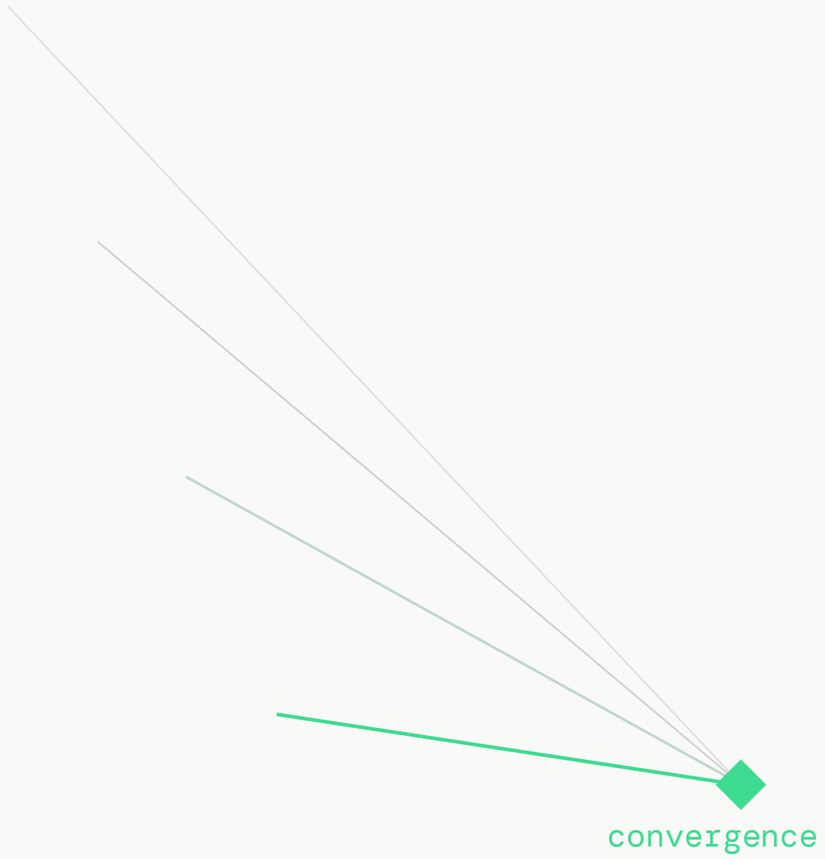
Tools are converging

Selenium, Playwright, and Cypress align on event-driven foundations

04

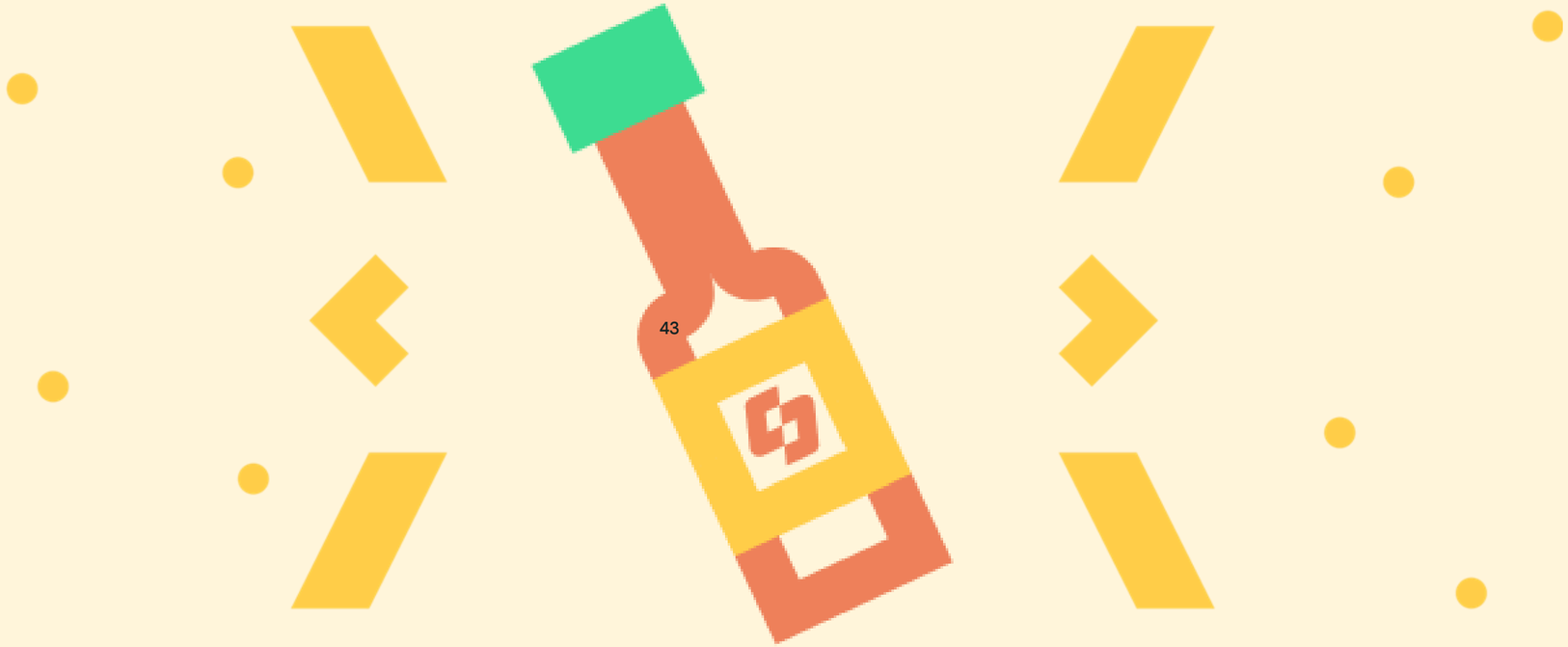
The mindset matters most

The biggest change is not the tooling — it is the mental model



The biggest change is not the tooling. **It is the mental model.**

Open Q&A



Thank you!

linkedin.com/in/diemol

expoQA®26

MADRID 26th, 27th & 28th May

Thank you for attending

expoqa.eu