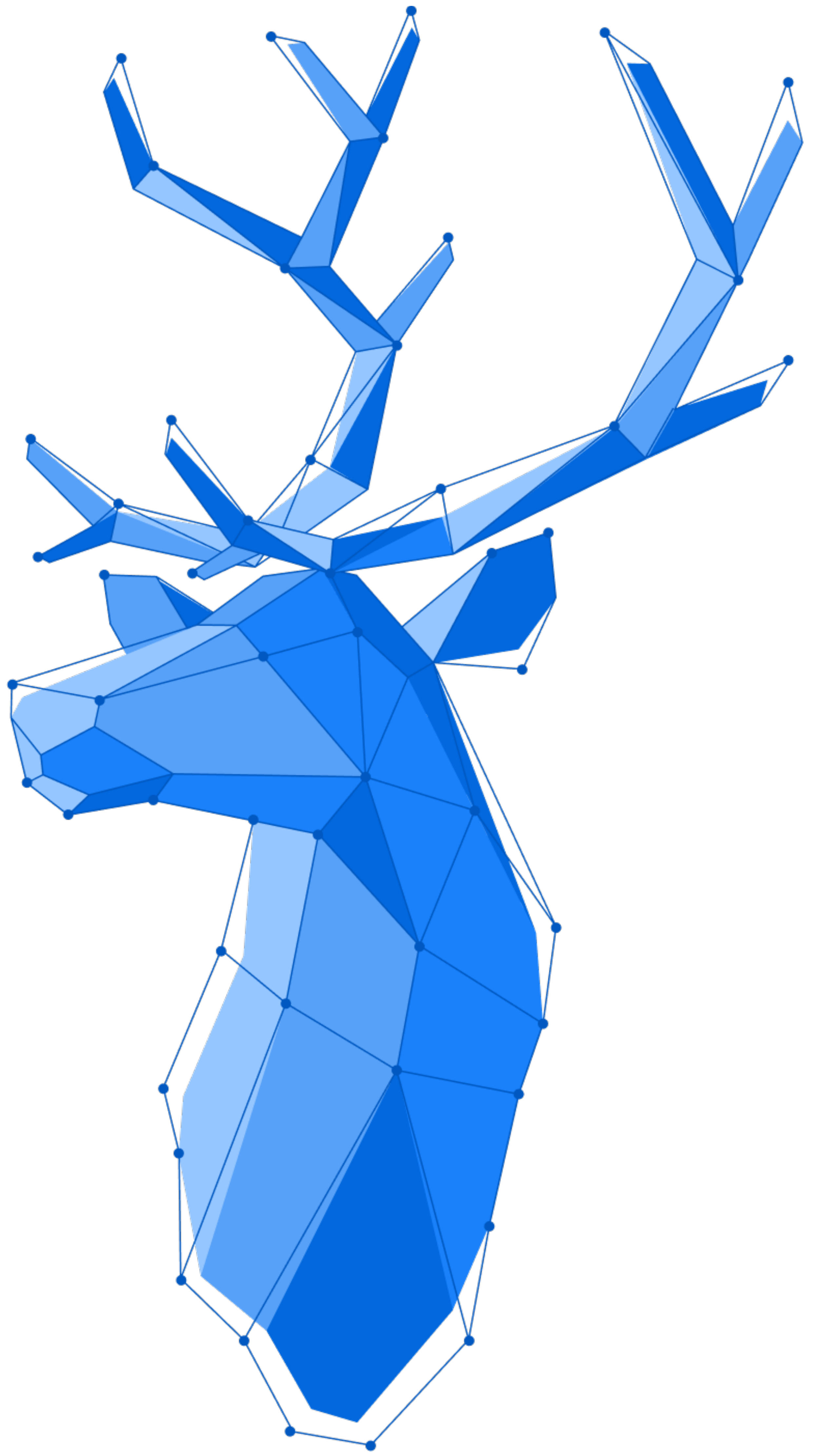
Using Full-Stack Signals for Real-Time Reactive UIs

Beyond REST

Leif Åstrand
VP of Research



Real-time use cases

Processing 50%

Lock

Notification

Homer

Hello chat!

Enter message

Editing together

Homer

If you remember only one thing from this talk

The secret to real-time updates is

Shared UI state

Poll

Learning about you

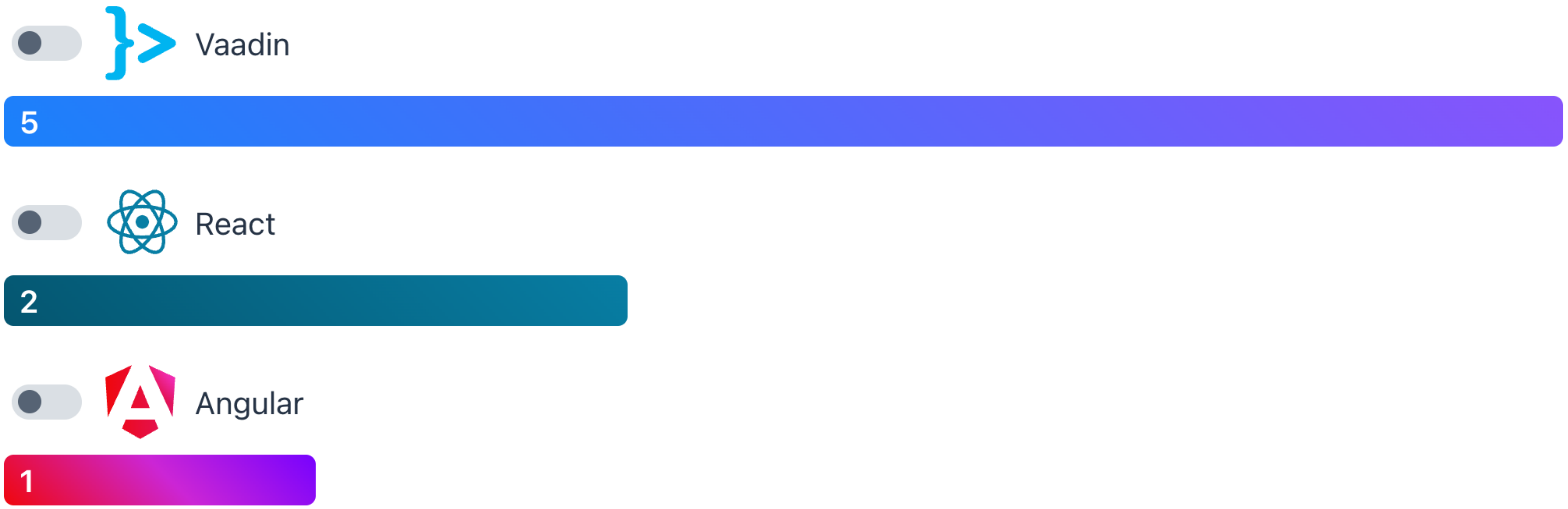
BUILDING A POLL IN 3 STEPS

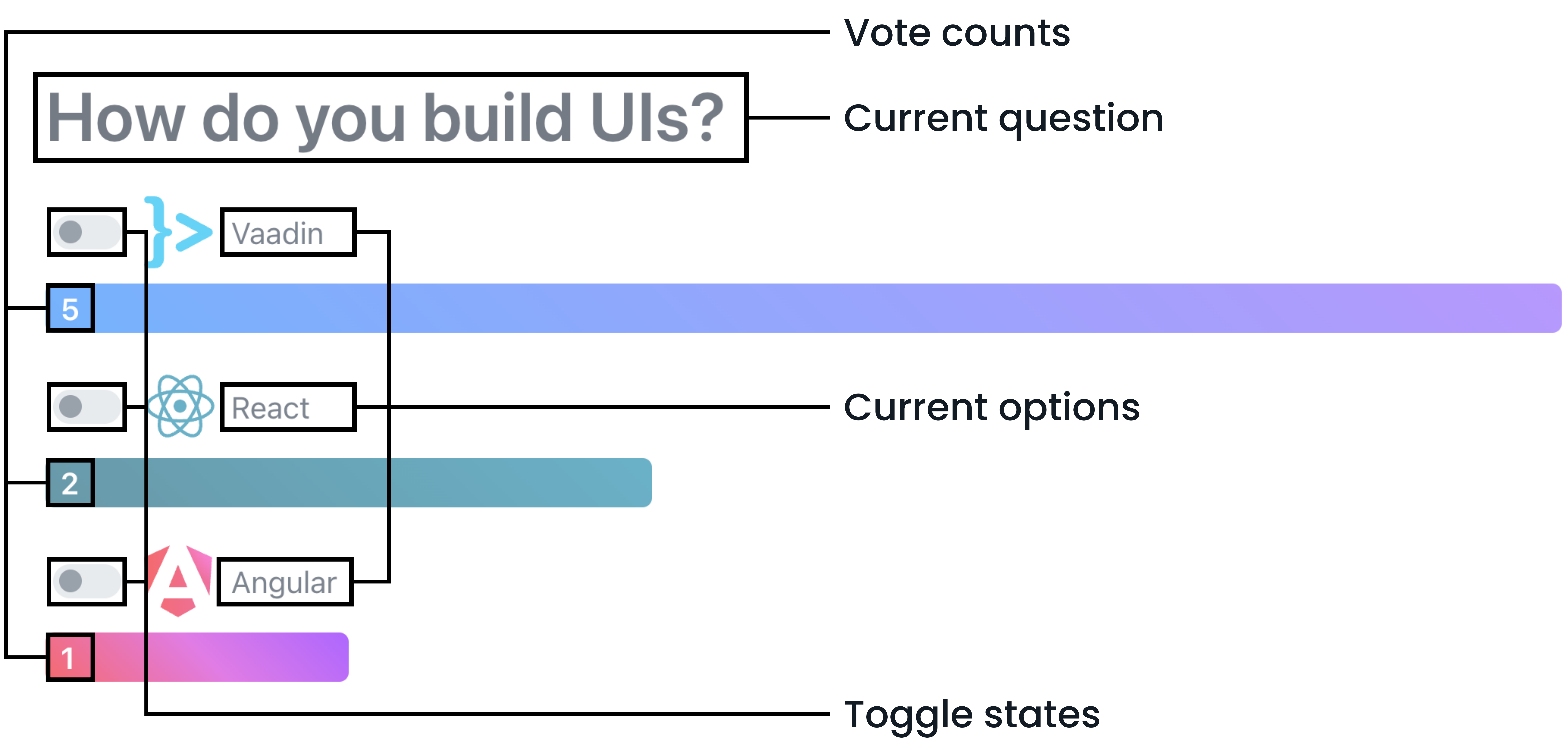
1. Basic implementation

2. Make it real-time

3. Do it yourself

How do you build UIs?




Vote counts
How do you build UIs?
Current question
Vaadin
5
React
Current options
2
Angular
1
Toggle states

vaadin}>

Demo: Add state management

Without considering real-time updates

vaadin}>

Signals: reactive value holders

```
const initialValue = 5;
const count = signal(initialValue);

const doubled = computed(() => count.value * 2);

effect(() => console.log(doubled.value));
// Logs 10

count.value++;
// Logs 12
```

Operations

Create	`signal(initial)`
Read	`x = signal.value`
Write	`signal.value = x`
Derive	`computed(() => {})`
Listen	`effect(() => {})`

`signal(initial)`	`signal(initial)`	`ref(initial)`
`x = signal()`	`x = signal.value`	`x = signal.value`
`signal.set(x)`	`signal.value = x`	`signal.value = x`
`computed(() => {})`	`computed(() => {})`	`computed(() => {})`
`effect(() => {})`	`effect(() => {})`	`watchEffect(() => {})`

Challenge: latency causes conflicts

Initial	`value = 0`			
Concurrent updates	`signal.value++`		`signal.value++`	
Expected	`value = 1`	then	`value = 2`	✗
Result	`value = 1`	then	`value = 1`	

Solution: atomic operations

Concurrent updates	`signal.increment()`		`signal.increment()`	✓
Result	`value = 1`	then	`value = 2`	

Demo: Make it real-time

In no time

vaadin}>

How do you do?

Option 1: Use Vaadin

Option 2.1: Build your own generic sync engine

Option 2.2: Implement use cases with a shared UI state mindset

Key concepts for a generic sync engine

- **Integrate** with regular Angular / Vue / React signals
- **Signal types** with atomic operations for numbers, lists, maps and trees
- **Event sourcing**: Latest value is the result of applying all operations
- **Optimistic updates** to show feedback before the server confirms
- **Transactions** to enable more complex operations
- **Cluster infrastructure** integration to distribute events between server nodes

Demo: Manually shared UI state

Use-case specific solution

QUESTIONS?

BUILDING A POLL IN 3 STEPS

1. Basic implementation

2. Make it real-time

3. Do it yourself

The secret to real-time updates is

Shared UI state

The end

Code at github.com/Legioth/poll