Signals: Cara Baru Reactivity di Angular, Vue, dan Svelte 2026

Kalau Anda memperhatikan changelog framework favorit dalam dua tahun terakhir, ada satu kata yang muncul berulang: **signals**. Angular, Vue, Svelte, dan SolidJS kini konvergen ke model reactivity yang sama, meninggalkan pendekatan lama berbasis Zone.js atau virtual DOM diffing penuh. Artikel ini membahas apa itu signals, kenapa hampir semua framework mengadopsinya, dan bagaimana implementasinya berbeda di tiap ekosistem.

Apa itu Signals dan Kenapa Muncul Sekarang

Signal adalah kontainer reaktif yang menyimpan sebuah nilai dan secara otomatis memberi tahu apa pun yang "membaca" nilai itu ketika terjadi perubahan. Notifikasinya bersifat **sinkron, langsung, dan sangat granular** — hanya komputasi dan node UI yang benar-benar bergantung pada signal tersebut yang dijalankan ulang, bukan seluruh komponen [web:84]. Konsep ini pertama kali dipopulerkan oleh SolidJS, dan kini sedang dibahas di TC39 (badan standar JavaScript) sebagai kandidat native reactivity primitive untuk JavaScript itu sendiri, saat ini sudah mencapai Stage 1 dengan diskusi aktif menuju Stage 2 [web:88].

Alasan signals menyebar begitu cepat adalah karena ia memecahkan masalah fundamental reactivity lama: model berbasis re-render komponen penuh (React hooks) atau change detection global (Angular Zone.js) menghasilkan banyak komputasi yang sebenarnya tidak diperlukan. Signals membalik itu — hanya bagian yang benar-benar berubah yang di-update.

Signals di Angular — Implementasi & Contoh Kode

Angular memperkenalkan signals sejak versi 16 sebagai developer preview, dan kini di Angular 22 (Juni 2026), signals sudah menjadi arsitektur default: `OnPush` change detection jadi strategi default untuk komponen baru, dan mode zoneless (tanpa Zone.js) juga default untuk aplikasi baru [web:89]. Ekosistem Signals Angular kini punya sembilan API utama: `signal()`, `computed()`, `effect()`, `input()`, `output()`, `model()`, `viewChild()`, `linkedSignal()`, dan `resource()`/`httpResource()` [web:81].

```
// Signal dasar
const count = signal(0);
count.set(5);
count.update(v => v + 1);

// Computed signal (turunan otomatis)
const doubled = computed(() => count() * 2);

// httpResource() — mengganti pola subscribe manual
const user = httpResource(() => `/api/users/${id()}`);
```

`linkedSignal()`, ditambahkan diam-diam di Angular 19, berfungsi sebagai "computed yang bisa ditulis ulang" — berguna untuk state yang bergantung pada nilai lain tapi tetap perlu di-override manual [web:81]. Tim Angular melaporkan peningkatan performa change detection sebesar 40-60% untuk aplikasi besar yang bermigrasi dari Zone.js ke signals [web:84].

Signals di Vue — Reactivity API Terbaru

Vue sebenarnya sudah memakai konsep yang secara arsitektural identik dengan signals sejak Vue 3 lewat Composition API (`ref()`, `computed()`), hanya dengan nama berbeda dan berbasis Proxy untuk dependency tracking [web:84]. Yang baru di 2026 adalah Vue 3.6 yang memperkenalkan **Vapor Mode** — menghilangkan virtual DOM sepenuhnya untuk direct DOM update — dikombinasikan dengan **Alien Signals**, rewrite total `@vue/reactivity` yang memakai algoritma push-pull untuk mengurangi alokasi memori dan overhead komputasi [web:86]. Fase push menandai dependent computed sebagai "dirty" tanpa menjalankan komputasi apa pun, dan fase pull baru menghitung ulang saat nilai benar-benar dibaca dan memang dalam status dirty [web:86]. Hasilnya, Vue 3.6 dengan Vapor Mode bisa mencapai bundle size sekitar 16,8 KB gzip, jauh lebih kecil dari Vue standar.

Signals di Svelte 5 — Runes

Svelte 5 mengganti sistem reactivity implisit lamanya (`let count = 0` yang otomatis reaktif) dengan **Runes** — simbol eksplisit yang memberi tahu compiler dengan jelas apa yang reaktif [web:87]. Tiga rune utama: `$state()` untuk mendeklarasikan variabel reaktif, `$derived()` untuk nilai turunan

yang otomatis termemoisasi (mirip rumus Excel), dan `$effect()` untuk menjalankan efek samping saat state berubah [web:87].

```
let count = $state(0);
let doubled = $derived(count * 2);

$effect(() => {
  console.log(`Count sekarang: ${count}`);
});
```

Yang membedakan Svelte dari signal murni SolidJS adalah Runes bekerja di level compiler: perubahan state dipropagasi melalui dependency graph langsung ke node DOM yang bersangkutan, tanpa komponen itu sendiri dijalankan ulang sama sekali — granularitasnya setara dengan SolidJS tapi dengan API yang lebih ergonomis [web:66][web:82].

Signals vs Hooks: Perbandingan Konsep

Aspek	React Hooks	Signals (Angular/Vue/Svelte/Solid)
Trigger update	Re-eksekusi seluruh fungsi komponen	Propagasi langsung ke node yang bergantung
Dependency tracking	Manual (dependency array)	Otomatis (dilacak compiler/runtime)
Memoization	Eksplisit (`useMemo`, `useCallback`)	Implisit, otomatis
Efek samping	`useEffect` dengan cleanup manual	`$effect`/`effect()` dengan model serupa tapi lebih aman dari stale closure
Stale closure	Masalah umum yang sering dijumpai	Tidak mungkin terjadi karena tidak ada re-eksekusi fungsi
Granularitas update	Level komponen (lebih kasar)	Level nilai individual (paling halus)

Perbedaan paling praktis: pada React, mengubah satu state bisa memicu re-render seluruh komponen (sebelum dioptimasi manual), sedangkan pada model signals, hanya elemen DOM spesifik yang benar-benar bergantung pada nilai itu yang diperbarui [web:82][web:88].

Dampak Signals terhadap Performa Rendering

Karena signals menghilangkan kebutuhan virtual DOM diffing atau re-eksekusi komponen penuh, dampaknya langsung terasa pada dua metrik: waktu update UI dan penggunaan memori. Benchmark SolidJS (signal murni tanpa virtual DOM) secara konsisten menunjukkan performa 50-70% lebih cepat dibanding React pada operasi update data dalam jumlah besar [web:66]. Svelte 5 dengan Runes mencapai granularitas setara tapi dengan ekosistem yang lebih matang dan kurva belajar lebih landai [web:66]. Angular melaporkan peningkatan 40-60% pada change detection untuk aplikasi besar [web:84], sementara Vue 3.6 Vapor Mode dengan Alien Signals memangkas overhead memori lewat model push-pull yang menghindari komputasi ulang yang tidak perlu [web:86].

Pertanyaan yang Sering Diajukan

Apakah signals menggantikan Redux/Vuex?
Untuk banyak kasus, ya — signals cukup kuat untuk menangani state lokal dan bahkan state global sederhana tanpa boilerplate tambahan. Namun untuk pola state management kompleks dengan time-travel debugging atau middleware seperti yang ditawarkan Redux, banyak tim masih mengombinasikan keduanya, terutama di codebase besar yang sudah ada.

Apakah signals akan jadi standar native JavaScript?
Kemungkinan besar ya di masa depan, tapi belum sekarang. Proposal signals sudah di Stage 1 TC39 dengan dukungan dari berbagai vendor framework, tapi implementasinya masih eksperimental dan diskusi menuju Stage 2 masih berjalan [web:88].

Framework mana yang implementasi signals-nya paling matang?
Angular dan Svelte 5 saat ini punya implementasi paling stabil dan sudah production-ready sepenuhnya (Angular 22 bahkan menjadikannya default), sementara SolidJS tetap yang paling "murni" secara arsitektur karena signals adalah fondasi utamanya sejak awal, bukan tambahan di atas sistem lama [web:89][web:66].

Baca juga: Astro vs SolidJS vs Qwik untuk memahami bagaimana model reactivity ini berdampak pada framework performa tinggi, atau Panduan Framework Website Terbaik 2026 untuk gambaran arah tren framework secara keseluruhan.