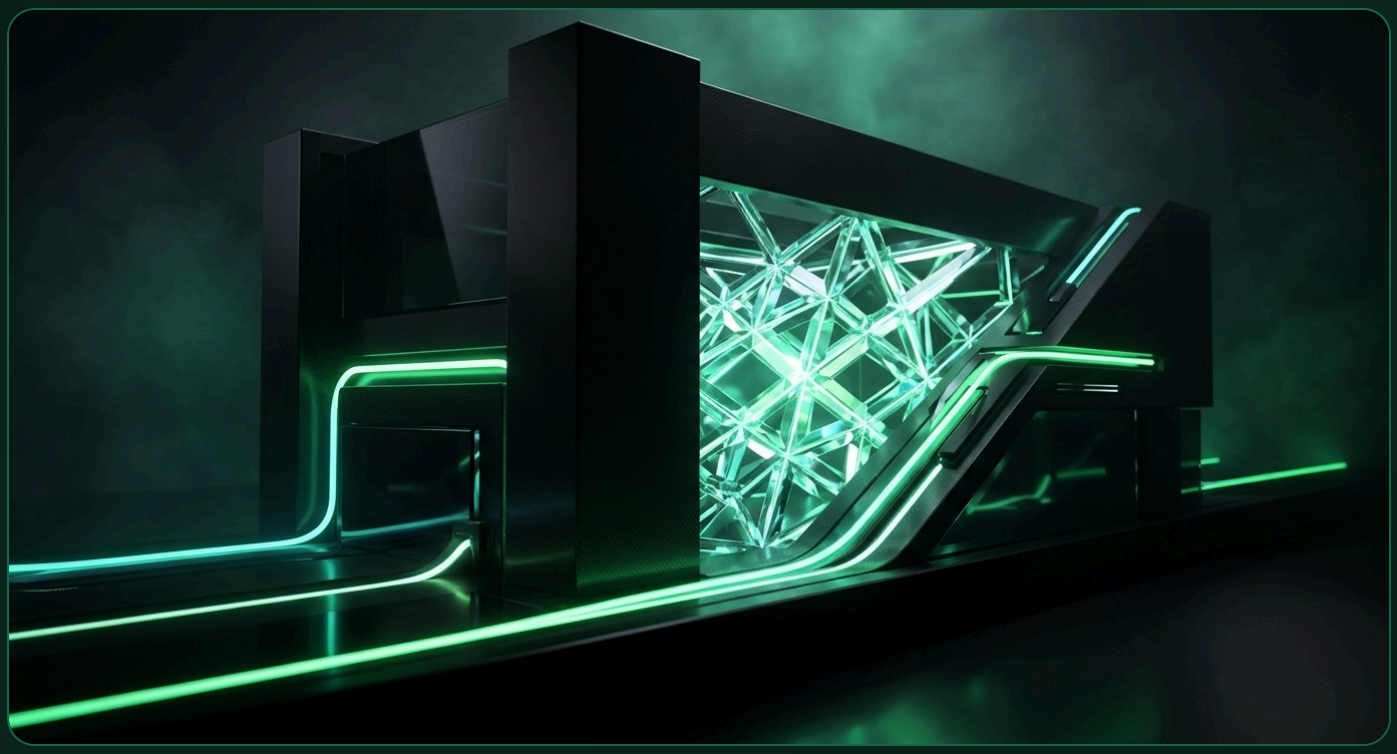
VERCEL & FULL-STACK AI APPLICATIONS • FIELD GUIDE

Next.js Cache Components & Partial Prefetching

The complete 12-page visual field guide to instant App Shells, deterministic `"use cache"` boundaries, `ensureStatic`, granular `<Suspense>` streaming, Partial Prefetching (`navigation()`/`prefetch()`), and `@next/playwright` `instant()` locks in Next.js 16.4+.

Next.js 16.4+ App Router | use cache & ensureStatic | Granular Suspense | next upgrade --agent | Partial Prefetching | @next/playwright instant()



Instant Static Shells

Prerender rich navigation chrome, headings, and cached data frames on millisecond-fast first paint while live request holes stream cleanly.

Zero-Guesswork Migration

Adopt route by route with first-party codemods, Turbopack `/_next/mcp` runtime checks, and top-down validation loops.

Locked Production Proofs

Prevent silent regressions forever using `@next/playwright` `instant()` locks and Partial Prefetching URL-data sweeps.

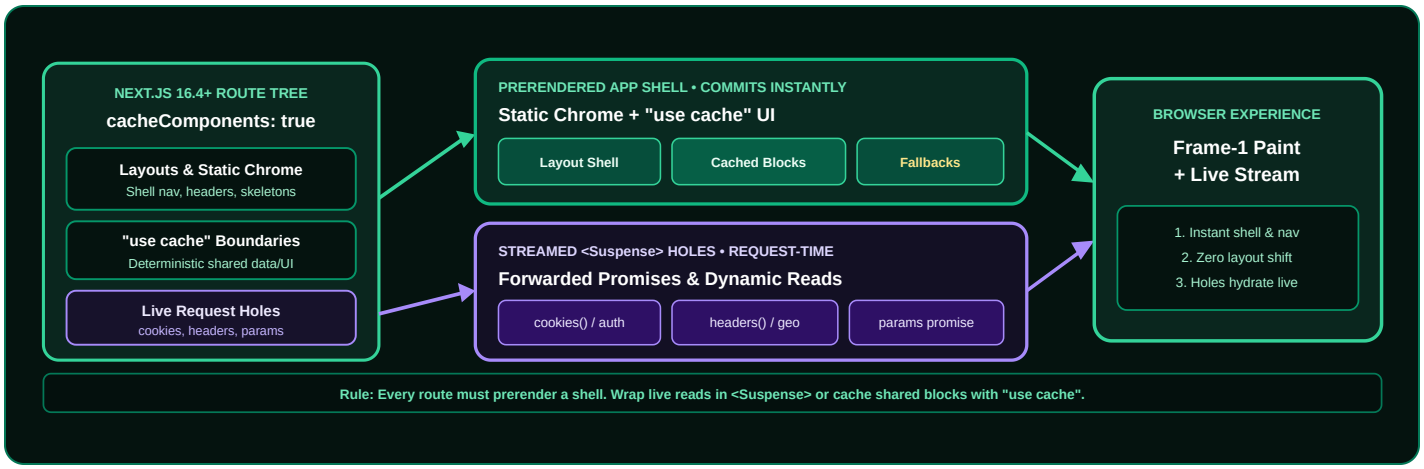
Instant Static Shell + Streamed Request Holes

Traditional SSR makes the user wait for the slowest database query before painting a single pixel. Turning on `cacheComponents: true` (now enabled by default in Next.js 16.4 `create-next-app` and slated to be default in Next.js 17) prerenders an instant **App Shell** and streams request-specific holes into `<Suspense>`.

Quick Navigation

1. Core Mental Model & Offline Docs p. 2
2. Prerequisites & Adoption Strategy Matrix p. 3
3. Annotated `next dev` + `/_next/mcp` Tour p. 4
4. Resolving the 3 Blocker Classes p. 5
5. Mastering `"use cache"` & Boundary Anatomy p. 6
6. Troubleshooting Clinic & Pre-Flight p. 12
7. Top-Down Loop & `next-dev-loop` p. 7
8. Partial Prefetching & URL-Data Sweeps p. 8
9. 5 Real-World Copy-Paste Recipes p. 9
10. Regression Proof with `instant()` p. 10
11. Cross-Tool Vercel Synergy Pipeline p. 11
Golden Rule: Push Reads Down, Keep Shells Rich All

How `cacheComponents: true` Splits Every Route



1 Offline Version-Matched Docs

Since Next.js 16.2, full App Router guides live directly inside `node_modules/next/dist/docs/` (e.g., `01-app/02-guides/migrating-to-cache-components.md`). Use `find node_modules/next/dist/docs -name '*.md'` to read exact API recipes offline.

2 Agent Upgrades & `AGENTS.md` Index

In Next.js 16.4+, run `npx next@canary upgrade --agent=latest` for agent-guided upgrades, or `npx @next/codemod@latest agents-md` to write a compact version-matched docs index into `AGENTS.md`.

Mental Model Takeaway: Prerender Everything You Can, Stream Only What Varies Per User

An empty `<Suspense>` wrapper around your entire page passes build validation but ships a blank spinner. Great Cache Components apps put headers, navigation, and shared catalog cards inside the instant shell, leaving only personalized or live request holes behind tight `<Suspense>` boundaries.

Prerequisites, Flag Migration & Strategy Matrix

Next.js 16.4 (with **React 19.3** and Nov 2026 security patch `16.4.5+`) promotes Cache Components + Partial Prefetching to the default model for new apps. Upgrade first and strip legacy route-segment flags.

1 Upgrade to Next.js 16.4+ & Check App Tree

```
# Agent-guided upgrade (or npm install next@latest)
npx next@canary upgrade --agent=latest
npx next --version   # Target 16.4.5+ (Nov 2026)
```

Cache Components applies to `app/` (or `src/app/`). Set `experimental.agentUpgrade: 'security'` for auto upgrade nudges. Hybrid `pages/` routes keep working untouched.

! Remove Fatal & Legacy Config Keys

- **`experimental.dynamicIO` is fatal:** Renamed to top-level `cacheComponents`; leaving the old key aborts startup before any build runs.
- **Remove `experimental.useCache`:** Deprecated alias once `cacheComponents: true` is enabled.
- **Inventory route exports:** `cacheComponents: true` errors on any file still exporting `dynamic`, `revalidate`, or `fetchCache`—migrate those manually first (existing `fetch` & `unstable_cache` calls can stay).

```
// next.config.ts — clean Next.js 16.4+ baseline (default in create-next-app!)
import type { NextConfig } from 'next'

const nextConfig: NextConfig = {
  cacheComponents: true,
  partialPrefetching: true,   // Now part of the unified Cache Components model
}
export default nextConfig
```

Decision Matrix: Incremental Adoption vs. Direct Adoption

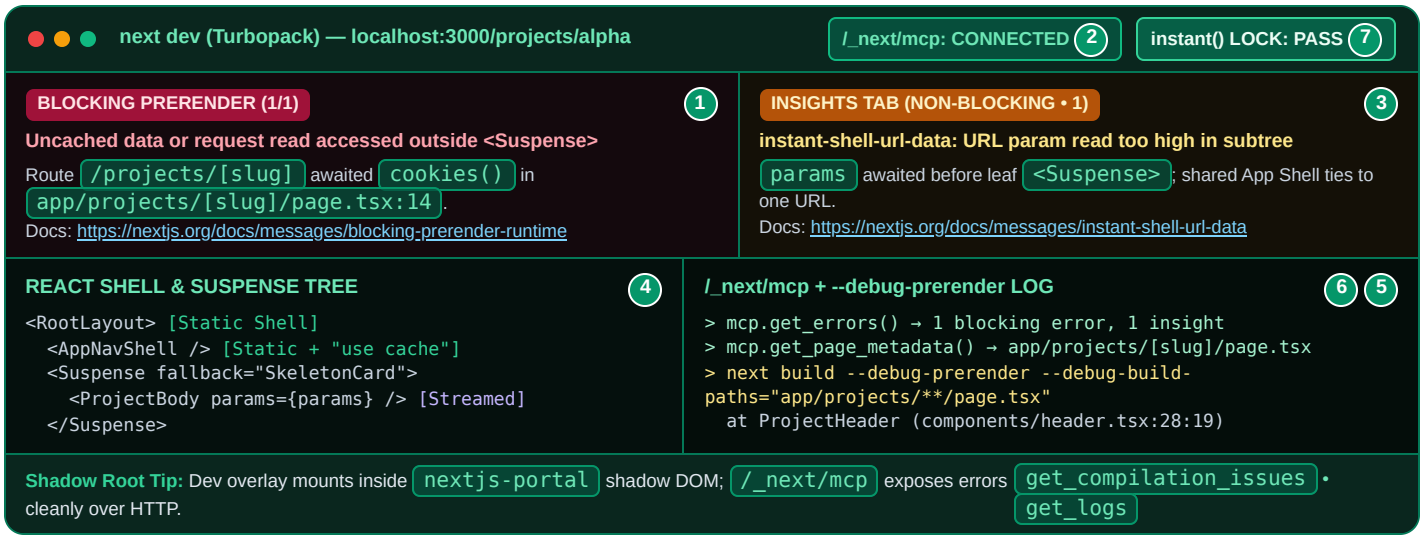
Dimension	Incremental Adoption (Recommended for Teams)	Direct Adoption (Small Apps / Single Branch)
Pre-Step Action	Run `npx @next/codemod@latest cache-components-instant-false ./app` (or `./src/app`) on a clean Git working tree, then enable `cacheComponents: true`.	Skip the codemod, set `cacheComponents: true` immediately, and use `next build` / `next dev` errors as your single-branch queue.
What Gets Modified	Codemod inserts `// TODO: Cache Components adoption` + `export const instant = false` into every Server `page`, `layout`, and `default` file.	No temporary opt-out markers added; every blocking route fails validation immediately until refactored in place.
PR & Review Shape	Ship a green baseline PR immediately (migrating previously `force-static` routes), then remove opt-outs feature-by-feature across small PRs.	All fixes sit together on one branch until every route in the app passes validation.
Hidden Gotchas	Codemod refuses dirty Git trees (use `--force` or commit lockfiles first) and cannot bypass Sync-IO calls (`new Date()`, `Math.random()`).	Large apps surface dozens of cascading layout errors at once; harder to bisect regressions during review.

How `export const instant = false` Resolves: Top-Down Build Shield vs. Client Navs

The codemod adds `export const instant = false` to *every* segment intentionally. During `next build`, the highest ancestor `instant = false` shields its entire subtree; during client-side navigations, Next.js still validates each descendant segment individually. Removing opt-outs top-down keeps every child safely deferred until you reach it.

Annotated Tour: `next dev` Overlay & `/_next/mcp`

Do not rely on HTTP 200 status codes or guesswork while migrating. Next.js 16.3+ pairs the browser dev overlay, amber Insights cards, and the live `/_next/mcp` server so you see every blocking read and React boundary in real time.



Interface Legend: The 7 Live Diagnostics Touchpoints

1. **Blocking-Prerender Fix Card:** Surfaces in red inside `next dev` with the exact offending line and canonical `https://nextjs.org/docs/messages/...` recipe link.
2. `/_next/mcp` **Turbopack Endpoint:** Built-in MCP server reporting routes, active segment files, compilation status, and server errors in one structured call.
3. **Amber Insights Tab:** Non-blocking warning tab for Partial Prefetching (`instant-shell-url-data`); disappears completely once every visited route is clean.
4. **Shell vs.** `<Suspense>` **Tree View:** Confirms via React DevTools that static chrome commits on frame 1 while only genuine dynamic leaves suspend.
5. `--debug-prerender` **Trace:** Disables early-exit on `next build` and prints full originating file-and-line traces across all blocking routes at once.
6. `get_errors` **&** `get_page_metadata`**:** Remember that prefetch insights surface in `get_errors` and dev logs—not the experimental `get_request_insights` recorder.
7. `@next/playwright` `instant()` **Lock Verdict:** Runs against a production `next build && next start` server to prove mathematically that the static shell renders immediately while deferred data is locked out.

Resolving the 3 Blocker Classes

Almost every prerender error during Cache Components adoption belongs to one of three predictable families. Fix the exact offending boundary rather than wrapping whole pages in a giant fallback.

1 Request-Time Reads

Culprits: `cookies()`, `headers()`, `await params`, `await searchParams` at page or layout top.

Fix: Move the read into a focused child component inside `<Suspense>`. Never `await params` at page top—forward the `params` or `searchParams` promise straight into the child and `await` it there.

2 Sync-IO Calls

Culprits: `new Date()`, `Date.now()`, `Math.random()`, `crypto.randomUUID()` during render.

Fix: These fail `next build` *even with* `instant = false`! If inside a shared layout, they break every child route. Run `--debug-prerender`, move into a `"use cache"` helper, or isolate behind `<Suspense>` / `connection()`.

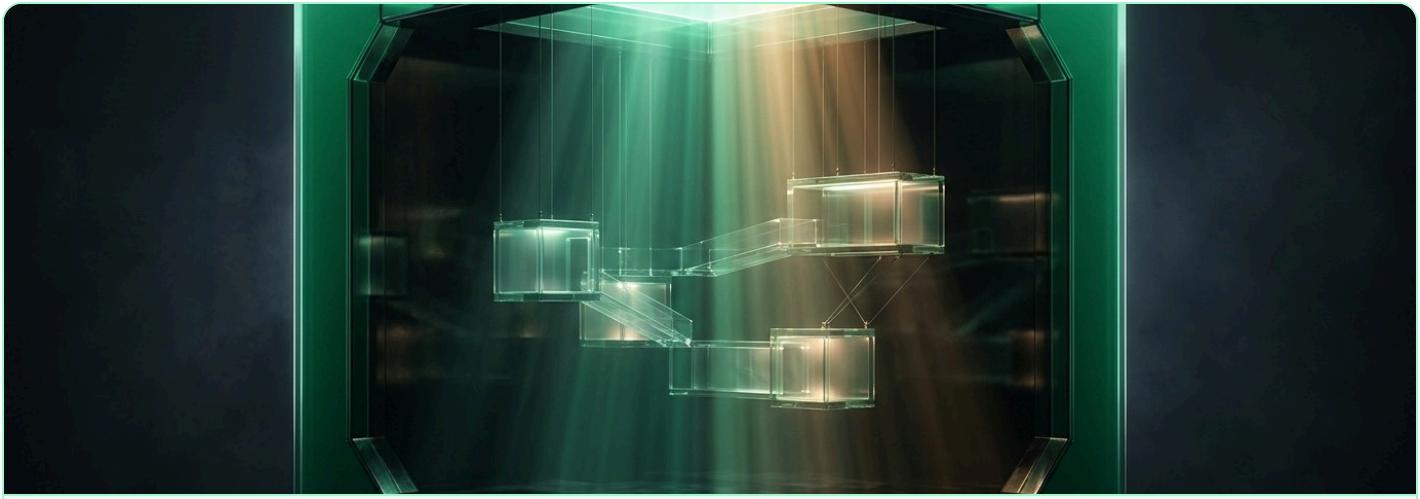
3 Client Nav Hooks

Culprit: `blocking-prerender-client-hook` from `usePathname()` or `useSearchParams()` in root nav.

Trap: Static pages pass (pathname is known at build) and hide the bug until you hit a dynamic route! Wrap the hook-reading client leaf in `<Suspense>` while keeping the outer nav bar static.

```
// WRONG: top-level await blocks the entire route shell
export default async function PostPage({
  params,
}: { params: Promise<{ slug: string }> }) {
  const { slug } = await params // Blocks before Suspense!
  return <Article slug={slug} />
}
```

```
// RIGHT: forward the promise into a Suspense child
export default function PostPage({
  params,
}: { params: Promise<{ slug: string }> }) {
  return (
    <ShellLayout>
      <Suspense fallback={<ArticleSkeleton />}>
        <ArticleBody params={params} />
      </Suspense>
    </ShellLayout>
  )
}
```



Solid Shell, Suspended Holes: Your outer layout and cached components form a rigid emerald glass frame that renders on frame 1, while live request-time reads (`cookies()`, `headers()`, forwarded `params`, and `usePathname()` leaves) stream cleanly into recessed `<Suspense>` bays.

Mastering `"use cache"`, `cacheLife` & Keys

When data is shared across visitors or changes on a predictable schedule, promote it into your static shell with `"use cache"` instead of streaming it on every request.

Data-Level vs. UI-Level Cache + Revalidation

```
import { cacheLife, cacheTag, updateTag } from 'next/cache'

export async function CatalogGrid({ category }: { category:
string }) {
  'use cache'
  cacheLife('hours')              // Built-in or custom
profile
  cacheTag('catalog', 'cat-' + category)

  const items = await db.products.findMany({ where: {
category } })
  return <ProductCards items={items} />
}

// Inside a Server Action after mutation:
export async function publishProduct(category: string) {
  'use server'
  await db.products.create(...)
  updateTag('cat-' + category)    // Immediate read-your-
own-writes
}
```

Cache Key Rules & Request-Data Guardrails

1. **Automatic Key Hashing:** Next.js hashes build ID, function location, serialized arguments, and closed-over values into the cache key.
2. **Never Call `cookies()` Inside `"use cache"`:** Read session or locale *outside* in a `<Suspense>` parent, then pass serializable primitives (e.g., `orgId`) in as arguments.
3. **Keep CLS Zero:** Match `<Suspense fallback>` dimensions (fixed heights, aspect boxes) to resolved content so streamed inserts never jump the page.

Anatomy of a Clean Cache + Suspense Boundary Refactor

[1. Instant Static Chrome] [2. "use cache" Data/UI Slice] [3. cacheLife + cacheTag] [4. Tight <Suspense> + CLS Fallback] [5. Server Action updateTag]

Render header, sidebar, and page title synchronously outside any boundary so frame 1 paints rich UI. Extract shared catalog or team metadata into an async helper or Server Component marked "use cache" with serializable props. Declare cacheLife('hours') and granular cacheTag('team-' + teamId) so prerender includes the cached slice. Wrap only the personalized user banner or live billing meter in <Suspense fallback={<BannerSkeleton h="48px" />}> passing userId outside the cache. Call updateTag('team-' + teamId) in the mutating Server Action and test the round-trip under next start.

Critical Runtime Trap: Passing `next build` Does Not Prove `"use cache"` Is Safe

A hidden `cookies()`, `headers()`, or auth session helper called deep inside a `"use cache"` call tree often passes `next build` silently, then throws a runtime error when a real user hits `next start`. Always verify every newly cached subtree against a live production run (`next build && next start`) and test its mutation invalidation path.

Top-Down Loop & `next-dev-loop` Verification

Adopt one product feature at a time (e.g., `app/settings/profile/**`), walking strictly top-down from root layout to leaf pages while cross-checking `/_next/mcp` against a real browser.

1 Why Root Layouts Must Un-Opt First

Validation follows **first-explicit-config-wins** top-down resolution:

- **Ancestor shields children:** Removing `export const instant = false` on a leaf does nothing while its parent layout still exports `instant = false`.
- **Children shield ancestor:** Removing a layout's opt-out leaves `next build` green while child pages keep theirs—actual layout blockers only fire once descendant opt-outs come off!
- **Clean root first:** Un-opt `app/layout.tsx`, clear any Sync-IO or client-hook nav blockers, then descend feature by feature.

2 Scoped Build & Prerender Flags

By default `next build` exits on the first blocking route. Use these two diagnostic flags when isolating tricky errors:

```
# Report ALL blocking routes + full file:line traces
npx next build --debug-prerender

# Build ONLY specific file-path globs (NOT URL paths!)
npx next build --debug-prerender --debug-build-paths="app/admin/**/page.tsx"
```

Passing a URL like `/admin` matches zero files and silently builds nothing.

Two-View Verification with Turbopack `next-dev-loop`

```
# 1. Start Turbopack dev server (default in Next.js 16.3+; avoid --webpack) & bind worktree browser session
SESSION="$(agent-browser session id --scope worktree --prefix next-dev-loop)"
agent-browser --session "$SESSION" --restore --enable react-devtools open http://localhost:3000/settings/profile

# 2. Cross-check framework state over /_next/mcp (SSE responses: extract JSON from data: lines)
#    • get_compilation_issues  → confirms Turbopack compile is clean
#    • get_routes              → lists app/ route inventory
#    • get_page_metadata       → reveals exact segment files powering the visited route
#    • get_errors / get_logs   → catches runtime, hydration, and blocking-prerender errors
```

Check Non-Blank Paint

Verify in `agent-browser` that frame 1 renders real navigation, headings, and skeletons—not a blank screen.

Check Sibling Routes

Whenever you refactor a shared layout or sidebar component, re-check sibling routes under that layout.

Grow Shells Deliberately

Once green, apply `next-cache-components-optimizer` to lift shared blocks out of `<Suspense>` into `"use cache"`.

Per-Route Success Bar: Dev Overlay Clean + Browser Verified + Passing `next build`

Never delete `.next` while `next dev` is running (it breaks active incremental state). When a feature passes `next-dev-loop`, finish with `npx next build` as your final commit gate before moving to the next feature.

Partial Prefetching (`partialPrefetching: true`)

Legacy Next.js prefetched entire routes per URL. With `partialPrefetching: true`, default links reuse **one shared App Shell** across every dynamic parameter, while only opted-in `<Link prefetch={true}>` links pre-warm cached URL-specific data ahead of entry.

1 5-Step Staged Rollout Sequence

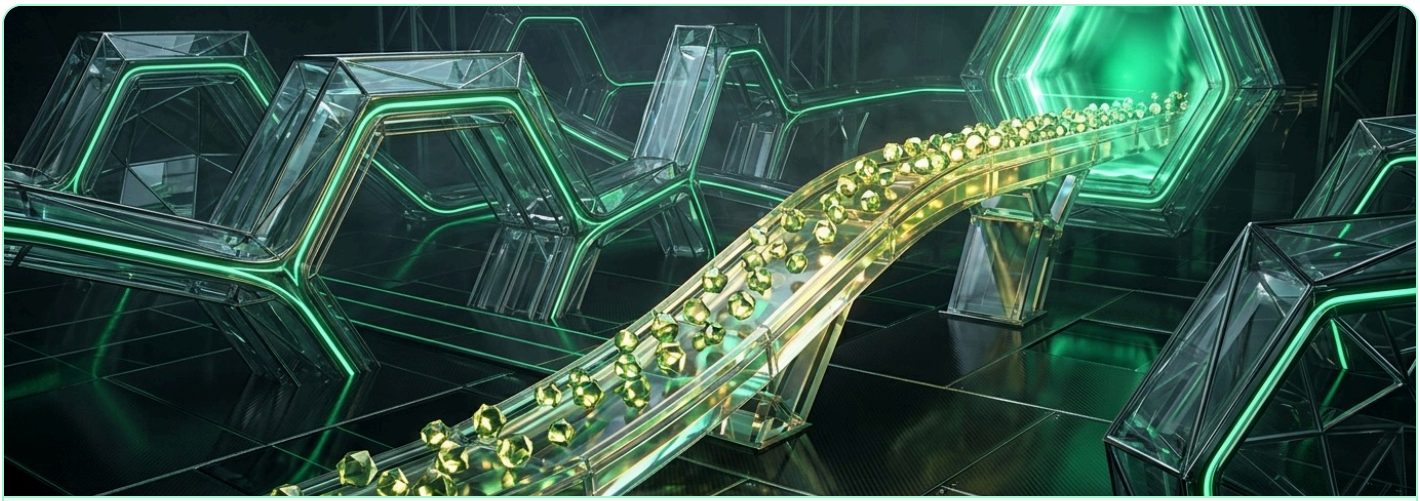
1. **Audit before enabling:** Search `app/` and shared components (`rg -n '\bprefetch\b|router\.prefetch'`) for effective `prefetch={true}` links and imperative `router.prefetch()` calls.
2. **Lock legacy baseline:** Write and pass `@next/playwright` `instant()` tests under `next start` with the flag still OFF.
3. **Stage destinations:** Add `export const prefetch = 'partial'` to audited routes until every locked test passes again.
4. **Flip flag + codemod:** Set `partialPrefetching: true` in `next.config.ts` and run `npx @next/codemod@canary remove-partial-prefetch ./app`.
5. **Sweep Insights:** Visit routes in `next dev` to clear any remaining `instant-shell-url-data` warnings.

```
// Next.js 16.4: lock static routes + defer heavy cached
data
export const ensureStatic = 'navigation' // or 'prefetch' |
'shell'

import { navigation, prefetch } from 'next/cache'
async function Thread({ id }: { id: string }) {
  await navigation() // exclude from prefetch; load on nav
  return <Messages rows={await getThread(id)} />
}
```

Catching `instant-shell-url-data` & Prefetch Bloat

Push `await params` into a leaf `<Suspense>` boundary so the shared shell stays URL-agnostic, and call `await navigation()` or `await prefetch()` (Next.js 16.4+) to defer heavy subtrees until click or explicit prefetch.



One Shared Hull, Selective Pre-Warming: Default `<Link>` navigations share a single cached emerald App Shell across infinite dynamic URLs, while explicit `<Link prefetch={true}>` paths pre-warm golden-green URL-specific data crystals before the click.

5 Real-World Copy-Paste Recipes

Drop these five production-tested patterns straight into Next.js 16.4+ App Router codebases. Every recipe respects non-blocking shell composition, `ensureStatic` guarantees, and clean mutation boundaries.

Recipe & Goal	Boundary & Config	Copy-Paste Snippet / Command
1. Incremental Route Adoption Starter Safe Team Rollout	Clean Git tree `cacheComponents: true` `instant = false` markers	`npx @next/codemod@latest cache-components-instant-false ./app` Set `cacheComponents: true` in `next.config.ts`, remove `experimental.dynamicIO`, migrate `force-static` routes first, then run `npx next build --debug-prerender`.
2. Promise-Forwarded Dynamic `[slug]` Page Zero Page-Top Await	Sync page export `<Suspense>` boundary Forwarded `params`	`export default function Page({ params }: { params: Promise<{ slug: string }> }) { return <Shell><Suspense fallback={<PostSkeleton />}><PostBody params={params} /></Suspense></Shell> }` — await `const { slug } = await params` strictly inside `PostBody`.
3. Authenticated Nav Shell (No Hook Block) Fixes Client Hook Error	Static outer `<nav>` Isolated client leaf `usePathname()`	Keep brand logo and layout links static in `RootLayout`. Wrap only `<ActiveRouteTabs />` (which calls `usePathname()` / `useSearchParams()`) and `<UserMenu />` inside `<Suspense fallback={<NavTabsSkeleton />}>` so dynamic routes never hit `blocking-prerender-client-hook`.

4. Cached Catalog + Mutation Revalidation
Rich Static Shell
	`"use cache"`
	`cacheLife('hours')`
	`cacheTag` + `updateTag`
		`async function Catalog({ cat }: { cat: string }) { 'use cache'; cacheLife('hours'); cacheTag('cat-' + cat); return <Grid items={await getItems(cat)} />; }`
		In your Server Action: mutate DB, then call `updateTag('cat-' + cat)` and verify under `next start`.

5. Opted-In Prefetch + `ensureStatic` Lock
Next.js 16.4 Control
	`ensureStatic`
	`partialPrefetching: true`
	`await navigation()`
		Add `export const ensureStatic = 'navigation'` on marketing/blog layouts to fail builds on accidental dynamic reads; behind `<Link prefetch={true}>`, call `await navigation()` in heavy child components to skip prefetch bloat.

Composition Power Move: Combine Recipe 2 + Recipe 4 for Instant Dynamic Pages

Forwarding the `params` promise into a `<Suspense>` child (Recipe 2) keeps your shared App Shell reusable across all URLs, while calling a `"use cache"` data helper inside that child (Recipe 4) lets `<Link prefetch={true}>` resolve the URL's cached content before the user even clicks.

Regression Proof with `@next/playwright` `instant()`

Timers are flaky and `next dev` does not prefetch. Guard your instant navigations with `instant()` from `@next/playwright` —a deterministic dynamic-data lock running against a real production build.

1 Expose Testing API in Test Rigs Only

```
// next.config.ts (gated to E2E/CI test builds)
export default {
  cacheComponents: true,
  partialPrefetching: true,
  experimental: {
    exposeTestingApiInProductionBuild:
      process.env.PLAYWRIGHT_TEST === '1',
  },
}
```

Run tests against `next build && next start`. Keep `@next/playwright` on the exact same release version as `next`.

2 The Present & Instant + Non-Blank Contract

- **Under the lock (`instant()`):** Dynamic data streams are paused. Your meaningful shell marker MUST be visible (`toBeVisible()`), and deferred dynamic data MUST still be absent (`toHaveCount(0)`) to prevent vacuous passes!
- **After lock release:** Dynamic content streams in (`toBeVisible()`).
- **RED → GREEN differential:** Prove the test fails RED before your fix, turns GREEN after, and stays non-blank visually.

Production Tests: Soft Client Navigation vs. Hard Initial Load

A. Soft Client Navigation (`<Link>` Click)

```
import { test, expect } from '@playwright/test'
import { instant } from '@next/playwright'

test('dashboard soft nav is instant', async ({ page }) => {
  await page.goto('/projects')
  const link = page.getByTestId('nav-analytics-link')
  await expect(link).toBeVisible()

  await instant(page, async () => {
    await link.click()
    await expect(page.getByTestId('analytics-
shell')).toBeVisible()
    await expect(page.getByTestId('live-
metrics')).toHaveCount(0)
  })
  await expect(page.getByTestId('live-
metrics')).toBeVisible()
})
```

B. Hard Initial Load (`page.goto` + `storageState`)

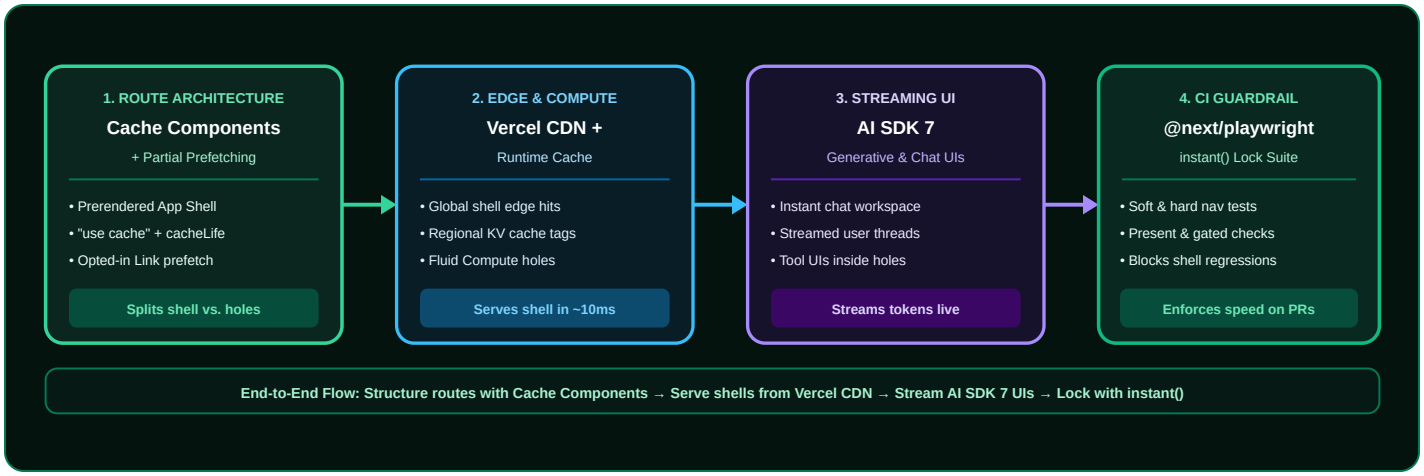
```
// Inject session via storageState — NEVER call page.goto()
// to log in before instant(), or auth runs unmeasured!
test('analytics hard load serves shell', async ({ page }) =>
{
  const url = 'http://127.0.0.1:3000/analytics'
  await instant(
    page,
    async () => {
      await page.goto(url)
      await expect(page.getByTestId('analytics-
shell')).toBeVisible()
      await expect(page.getByTestId('live-
metrics')).toHaveCount(0)
    },
    { baseURL: new URL(url).origin }
  )
  await expect(page.getByTestId('live-
metrics')).toBeVisible()
})
```

Always Assert Both Halves of the Self-Validating Lock

If `exposeTestingApiInProductionBuild` was accidentally omitted, `instant()` silently becomes a no-op and a shell-only assertion passes falsely! Checking `toHaveCount(0)` on a known dynamic node inside the lock guarantees the lock actually engaged.

Cross-Tool Synergy: From Static Shell to Live AI UI

Cache Components and Partial Prefetching form the presentation foundation of modern Vercel applications—pairing edge-delivered App Shells with Runtime Cache, AI SDK 7 streaming interfaces, and automated CI locks.



How These Layers Click Together in a Production AI App

1 Edge Shells + AI SDK 7 Streaming

In an AI workspace, the sidebar, model picker, and prompt composer belong in your prerendered App Shell via `"use cache"`, served straight from the Vercel CDN. Only the user's active conversation history and **AI SDK 7** token streams sit inside `<Suspense>`, so clicking any thread opens immediately.

2 Runtime Cache + CI Lock Discipline

Tag-based invalidation (`updateTag` / `revalidateTag`) coordinates across Next.js `"use cache"` and Vercel Runtime Cache, while running `@next/playwright` `instant()` against preview deployments catches accidental top-level `cookies()` or `await params` regressions before merge.

Design Takeaway: Separate Public Infrastructure From Personalized Streamed State

When your route boundaries mirror how data actually changes—public chrome in the edge shell, shared reference data in `"use cache"`, and user-specific AI or auth data behind tight `<Suspense>` bays—every layer of the Vercel platform reinforces the next.

Troubleshooting Clinic & Pre-Flight Checklist

Most Cache Components and Partial Prefetching snags trace to one of seven concrete patterns. Match your error below for the ten-second fix, then verify your app against the pre-flight checklist.

Symptom / Error	Root Cause	10-Second Fix
1. `experimental.dynamicIO` aborts startup	Legacy experimental flag was replaced by top-level `cacheComponents` in Next.js 16.3+.	Delete `experimental.dynamicIO` (and deprecated `experimental.useCache`) from `next.config.*` and set top-level `cacheComponents: true`.
2. Top-level `await params` blocks shared shell	Awaiting `params` or `searchParams` before `<Suspense>` ties the shell to one URL or fails prerender.	Remove `await` from the page/layout function; forward the `params` promise into a `<Suspense>` child and `await` it inside that child.
3. Sync-IO breaks an `instant = false` route	`new Date()`, `Date.now()`, `Math.random()`, or `crypto.randomUUID()` ignores opt-outs.	Run `next build --debug-prerender --debug-build-paths="..."` to locate the call; move into `"use cache"`, behind `<Suspense>`, or use `connection()`.
4. `cookies()` throws inside `use cache` under `next start`	`next build` passed, but a cached helper called `cookies()`, `headers()`, or auth session at runtime.	Read session/cookies *outside* `"use cache"` inside a `<Suspense>` parent, and pass serializable primitives (like `tenantId`) as arguments.
5. `Only async functions allowed` with `"use cache"` + `instant`	A file with top-level `"use cache"` also exported `const instant = false` or read request data.	Remove file-level `"use cache"` from the route file; place `"use cache"` strictly on pure async data/UI functions inside helper modules or components.
6. Codemod reports `0 ok` or misses client-hook error	Codemod targeted wrong dir (`./app` vs `./src/app`), or a shared nav calls `usePathname()`.	Re-run codemod against the real app folder (check count > 0), and wrap `usePathname()` / `useSearchParams()` client leaves in `<Suspense>`.
7. `<Link prefetch={true}>` returns shell-only	Destination lacks `export const prefetch = 'partial'` (or `partialPrefetching: true`), or URL data isn't cached.	Add `export const prefetch = 'partial'` during rollout (or enable `partialPrefetching: true`), wrap URL reads in `"use cache"`, and test under `next start`.

6-Point Pre-Flight Checklist Before Merging to Main

- ✓ **Legacy Configs Cleared:** Zero remaining `experimental.dynamicIO`, `dynamic`, `revalidate`, or `fetchCache` exports across `app/`.
- ✓ **Top-Down Opt-Outs Removed:** Every adopted feature (starting from root layout) has zero leftover `export const instant = false` markers.
- ✓ **`next-dev-loop` & Insights Clean:** Turbopack `/_next/mcp` reports zero compilation or blocking errors and zero amber Insights warnings.
- ✓ **Rich Non-Blank Shells Verified:** First paint shows real layout chrome, headings, and CLS-matched skeletons—never a full-page blank spinner.
- ✓ **Production `next start` Tested:** Every new `"use cache"` boundary, mutation `updateTag()`, and `<Link prefetch={true}>` verified under `next start`.
- ✓ **`instant()` Lock Passing in CI:** Both soft-nav and hard-load `@next/playwright` `instant()` tests assert shell visible AND deferred content `toHaveCount(0)`.