The Context Isolation Trap: Why Standard i18n Breaks in MDX

The Problem

MDX renders as static HTML. It is isolated from the server request lifecycle.

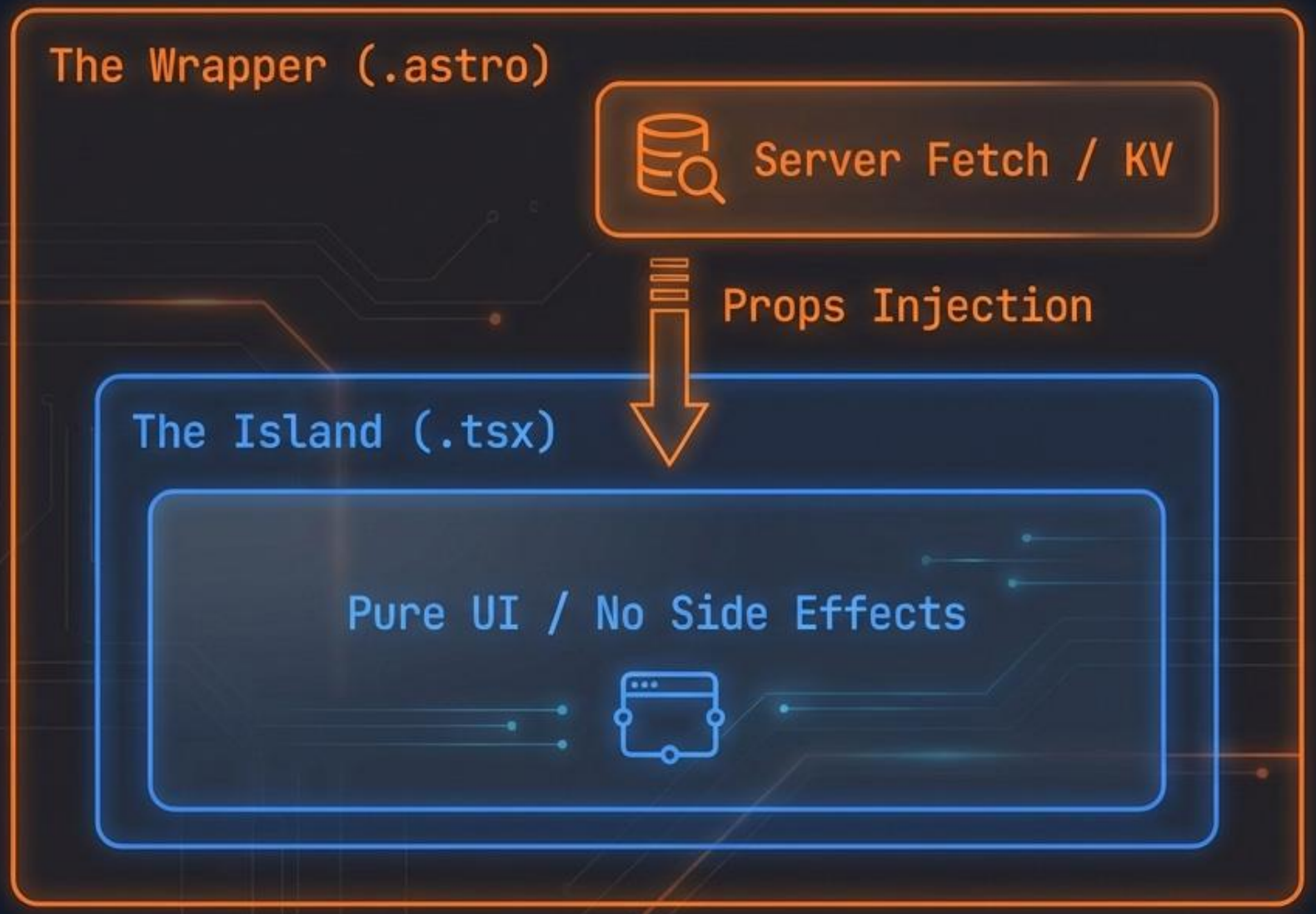
The Fail State

Standard `useTranslation()` hooks return `undefined` because the Context Provider is missing in the island.

The Bloat

Client-side fixes require shipping large JSON bundles, blocking hydration.

The Pattern: Server Controller, Pure View



- Separation of Concerns: Split logic into a Data Controller and a Pure View.
- **Server-Side Fetching:** The `.astro` wrapper runs entirely on the server (SSR).
- **Prop Drilling:** Translations are passed as `serializable` props.
- **Zero Network Requests:** The component hydrates with data already present.

Implementation Step 1: The Bridge (.astro)

This file lives in the components folder but acts as the glue between MDX and Edge data.

- Access `translationLocale` from `Astro.locals`.
- Fetch namespaces (e.g., 'blog') via `fetchTranslations`.
- Render the React component with `client:visible`.
- Pass the dictionary as the `t` prop.

```astro
---
// src/components/blog/LocalizedCounterWrapper.astro
import { LocalizedCounter } from '@/components/islands/LocalizedCounter'
import { fetchTranslations } from '@/domain/i18n/fetcher'

const { translationLocale, runtime } = Astro.locals
const { blog } = await fetchTranslations(runtime, translationLocale, ['blog'])
const t = blog.counter
---

<LocalizedCounter
  client:visible
  t={t}
  locale={translationLocale}
  labels={blog.counter}
/>
```

Implementation Step 2: The Pure UI Component (.tsx)

src/components/islands/LocalizedCounter.tsx

```tsx
// src/components/islands/LocalizedCounter.tsx
interface LocalizedCounterProps {
  t: I18n.Schema['blog']['counter'] // Typed Dictionary
  initial?: number
  locale: string
  labels: CounterLabels
}

export const LocalizedCounter = ({
  t,
  initial = 0,
  locale,
  labels,
}: LocalizedCounterProps) => {
  // ... Standard React Logic ...
  return (
    <div className="text-4xl">
       {t.title} {/* Used directly from props */}
    </div>
  )
}
```

Type Safety: Validated against JSON Schema

Zero Hooks: Data is just a prop

Implementation Step 3: Injecting the Wrapper into MDX

```astro
// pages/[lang]/blog/[...slug].astro
import LocalizedCounterWrapper from
'@/components/blog/LocalizedCounterWrapper.astro'

// ... fetch post logic ...

<BaseLayout>
  <Content
    components={{
      // Map MDX tag to the Astro Wrapper
      LocalizedCounter: LocalizedCounterWrapper,
    }}
  />
</BaseLayout>
```

The Magic Trick: MDX writes the tag, Astro renders the Wrapper.

The Edge-Native Advantage

Zero Client JS

Translations are baked into HTML during SSR. No hydration blocking.

Perfect Hydration

Components wake up immediately with valid props. No layout shift or flicker.

Resilient Fallbacks

Build-time safety ensures props are never undefined, even if the DB fails.

Developer Experience

Content creators write simple Markdown; Engineers control the data pipeline.

Ready to Ship Zero-JS i18n? 🚀

EdgeKits / astro-edgekits-core

The open-source starter for zero-JS edge translations.

★ Star the repo on GitHub

📖 The Full Deep Dive

Read the complete architecture breakdown on the EdgeKits.dev blog.

✉️ Join Early Birds

Get exclusive advanced Astro patterns & launch discounts.

𝕏 @GaryEdgeKits

Let's talk Edge stuff!