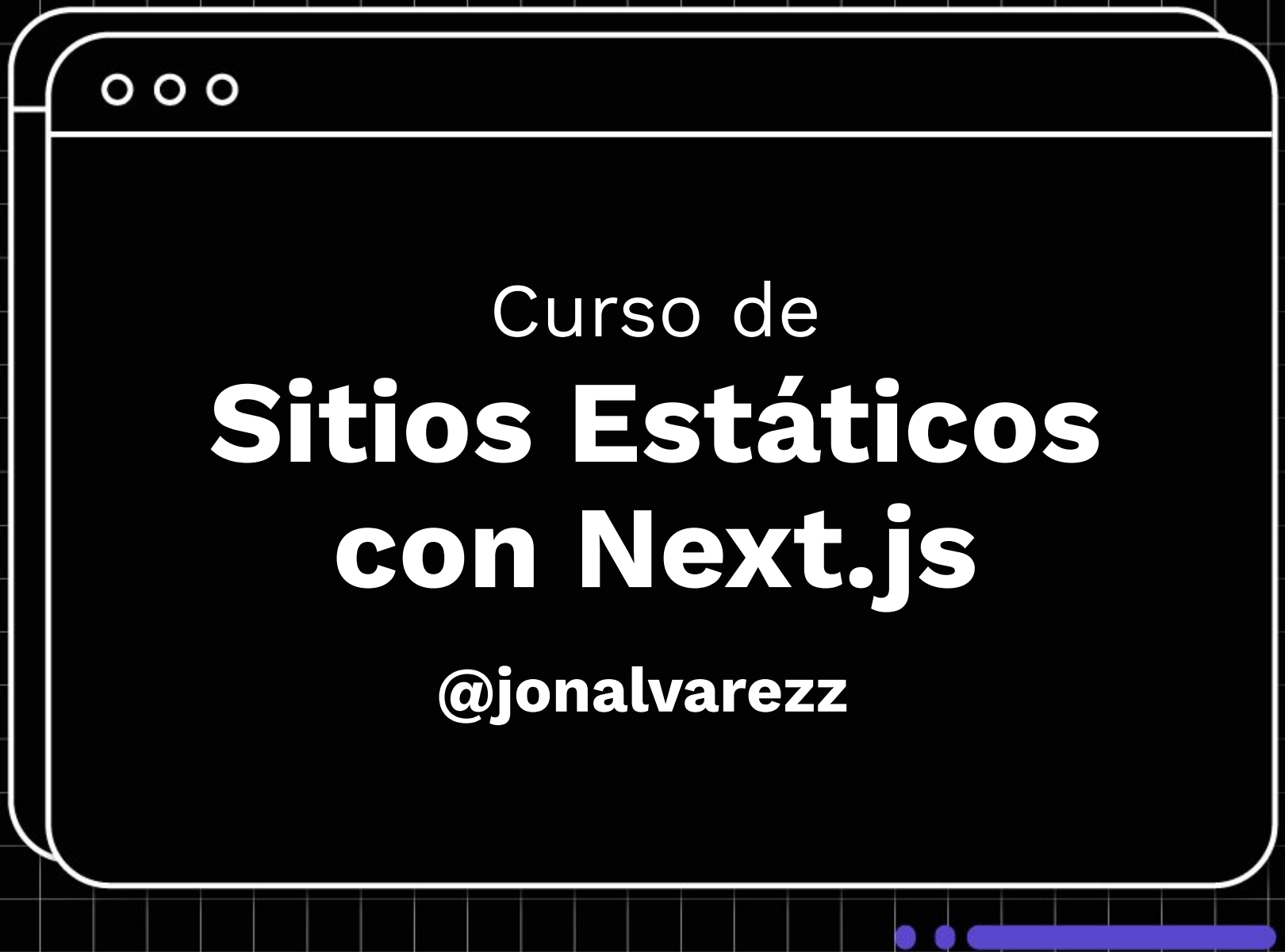
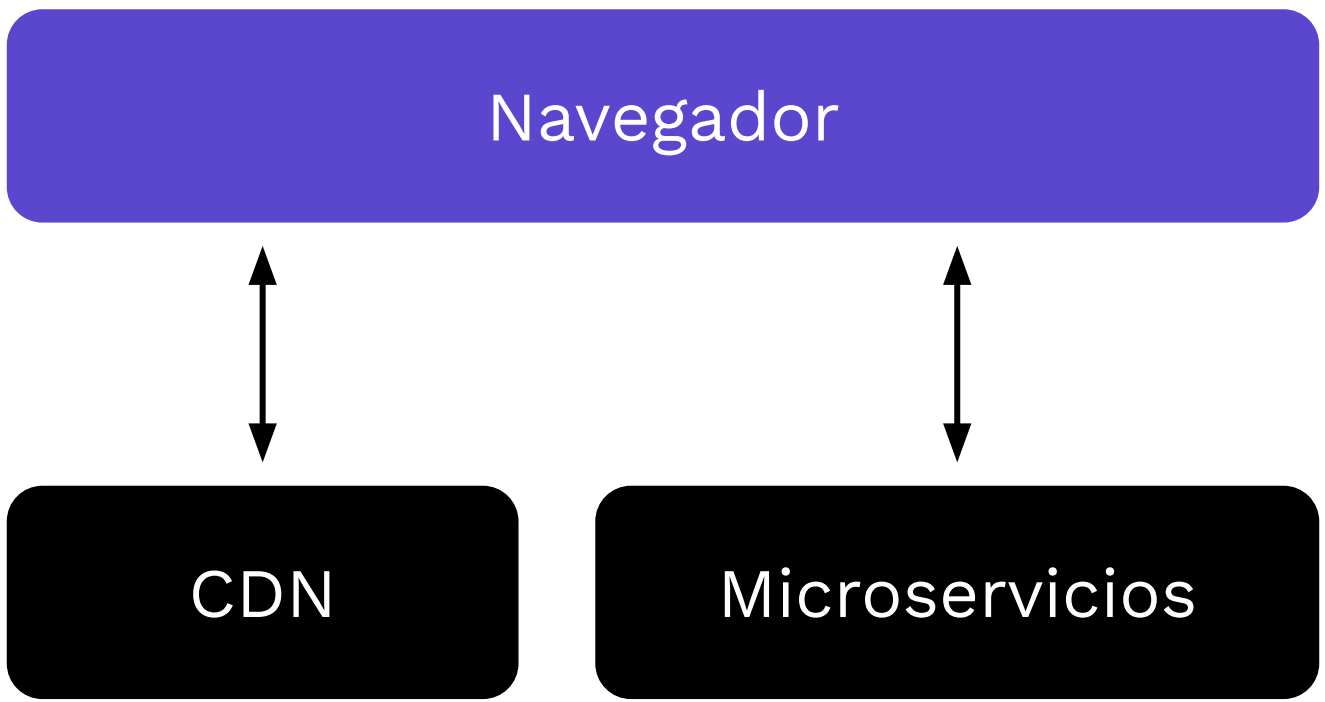
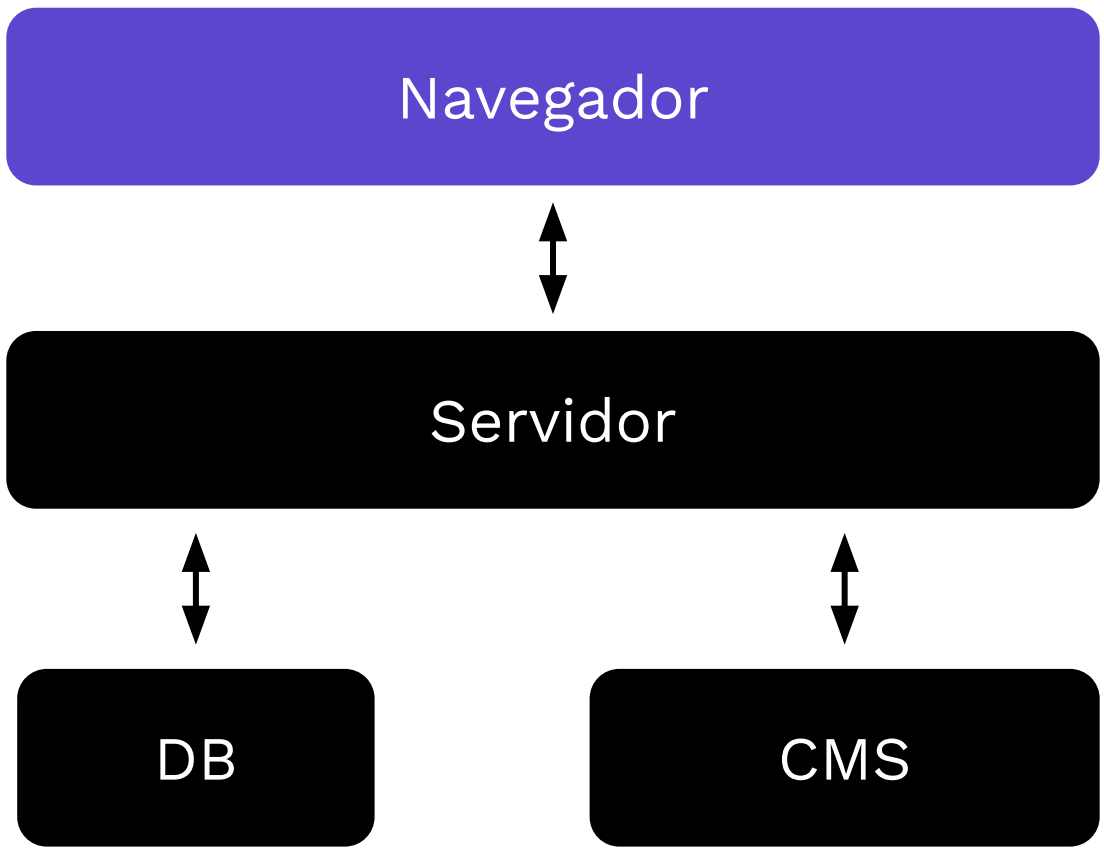

Curso de
Sitios Estáticos con Next.js
@jonalvarezz

Sitios estáticos y Jamstack

Jamstack



Web tradicional



“

En general, solo hay **dos formas*** de renderizar una página web: en el navegador *(cliente)* o en el servidor.

”

Modos de rendering en Next.js

“

¿Rendering?

”

 Rendering

Procesar *pedazos de código* y *datos*
para mostrar su resultado.

 Rendering

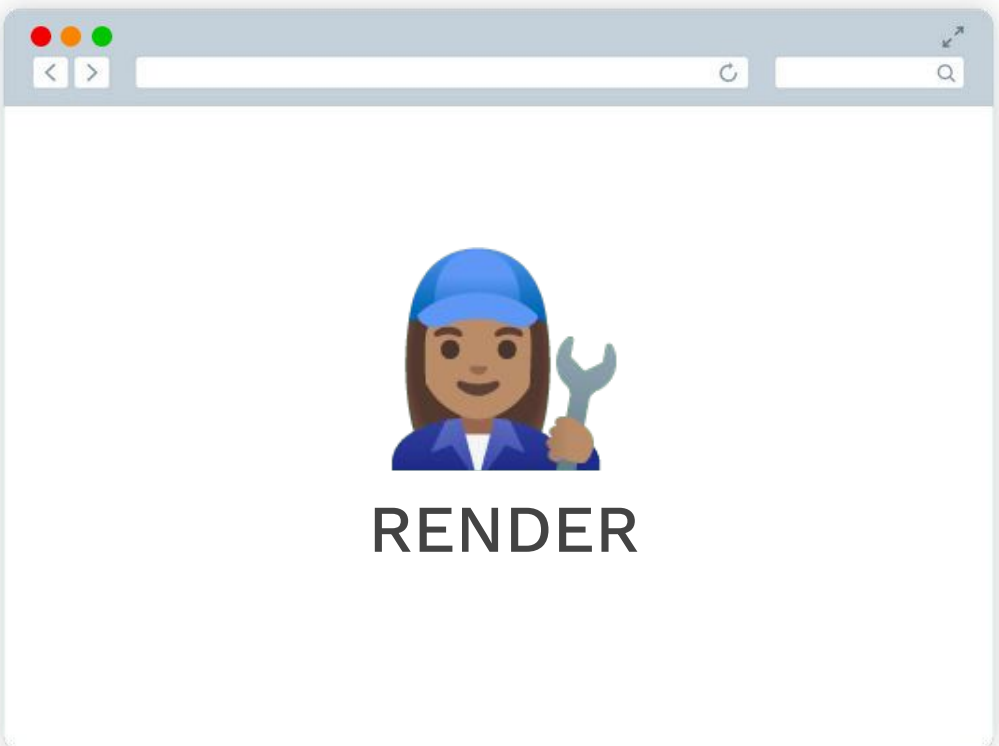
JavaScript

Procesar *~~pedazos de código~~* y *datos*
para mostrar ~~su resultado~~. HTML

Modos rendering

- Client-side
- Server-side
- Static rendering

Client-side rendering



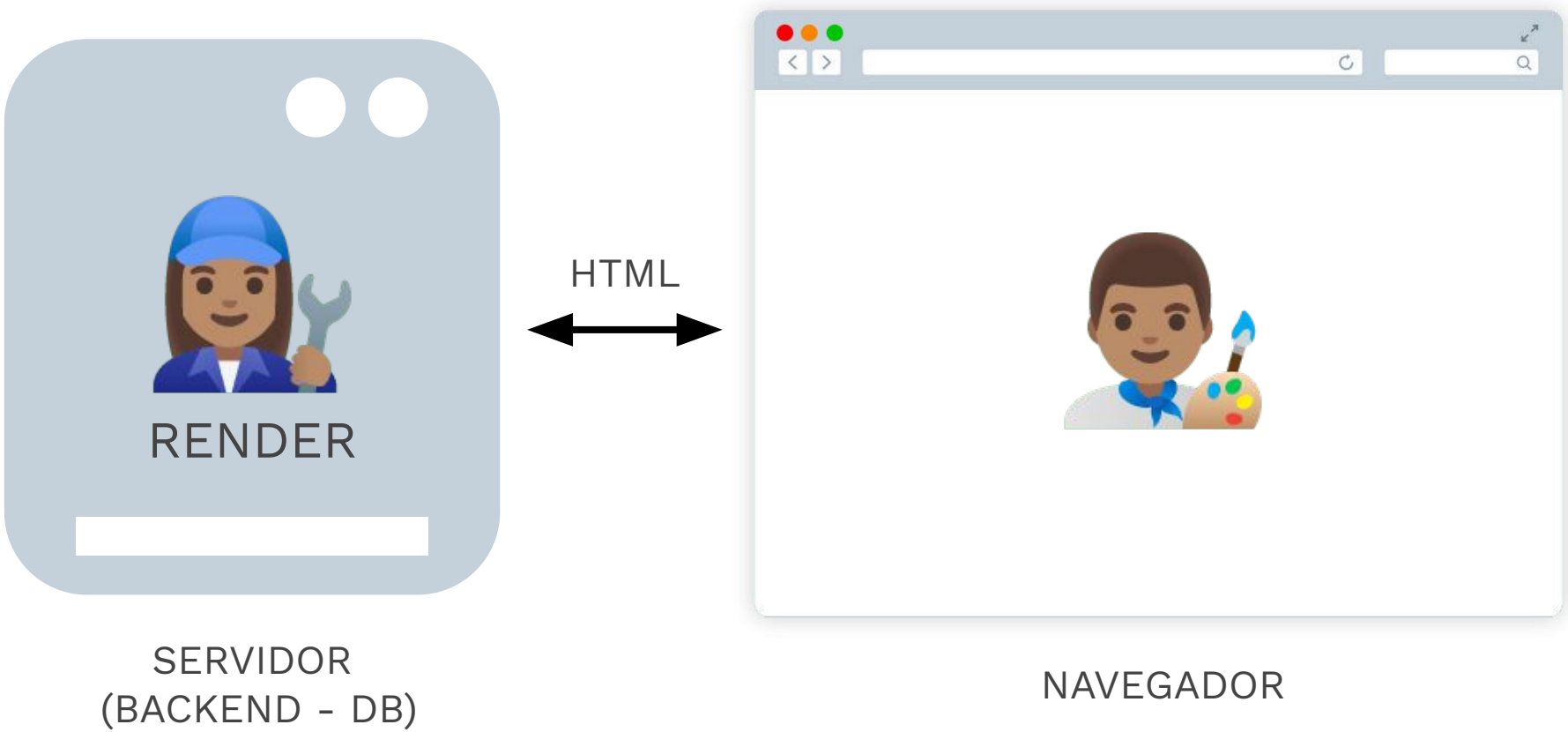
Client-side rendering

- Sucede bajo demanda en el navegador.
- Ej.: Cada que agrego JavaScript usando la etiqueta src.
- Ej.: create-react-app.

Client-side rendering

- Sucede bajo demanda en el navegador.
- Ej.: Cada que agrego JavaScript usando la etiqueta src.
- Ej.: create-react-app.

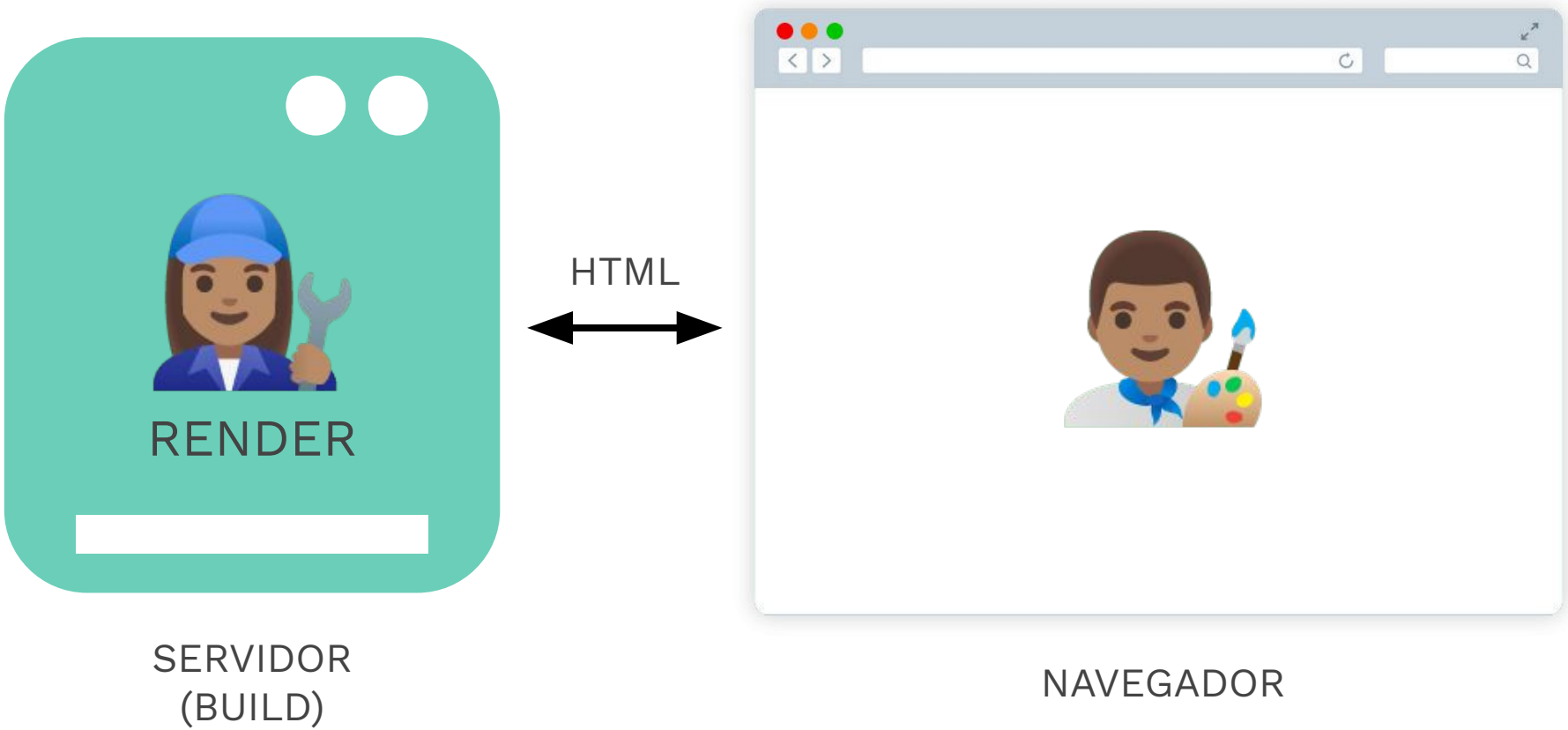
Server-side rendering



Server-side rendering

- Sucede bajo demanda en el servidor.
- La gran mayoría de lenguajes y frameworks backend.
- Symfony (PHP), Wordpress clásico (PHP), Flask (Python), Django (Python), etc.

Static generation rendering



Static generation

- Sucede una única vez en “build time”.
- Jekyll, Wintersmith, Gatsby, Hugo, Next.js.

“

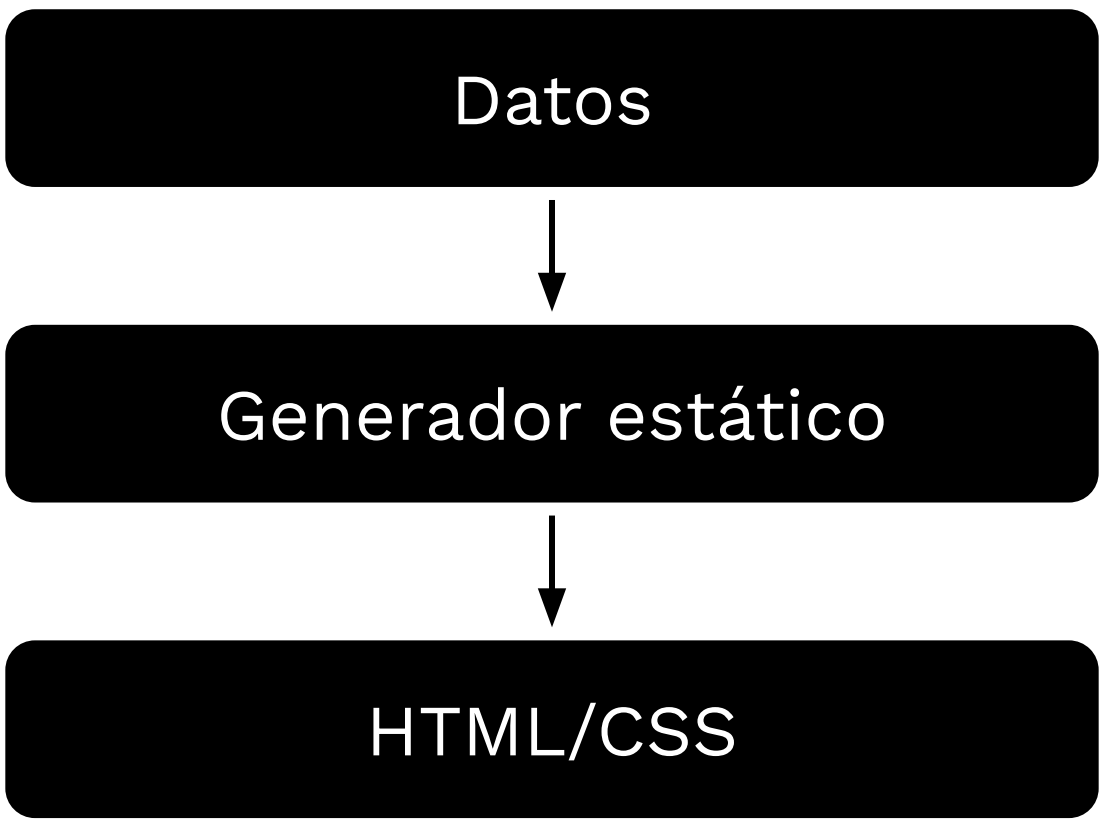
Next.js te permite crear aplicaciones híbridas

”

Arquitectura del proyecto

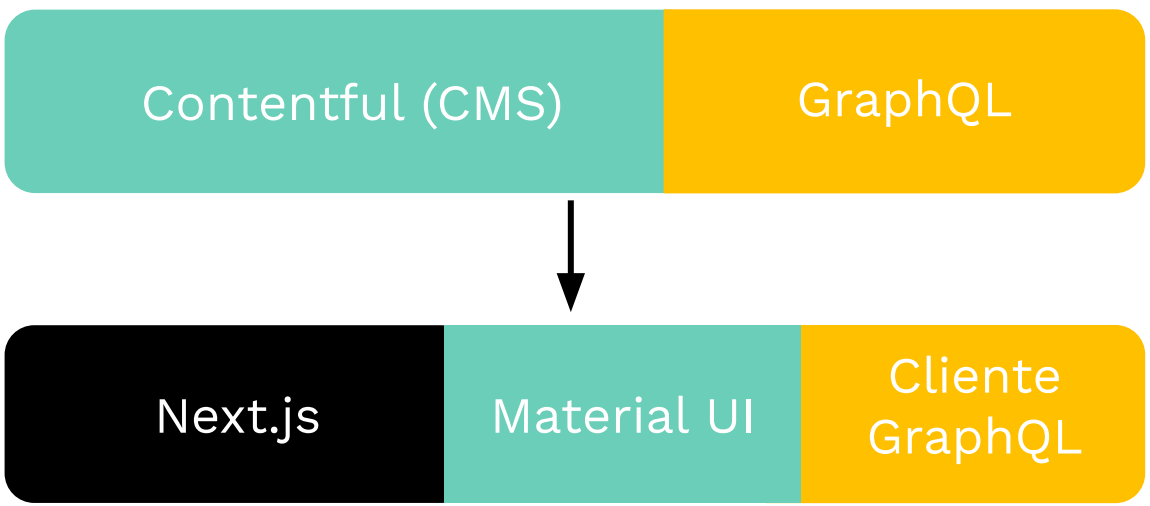
Arquitectura de nuestra App y GraphQL

Arquitectura Jamstack

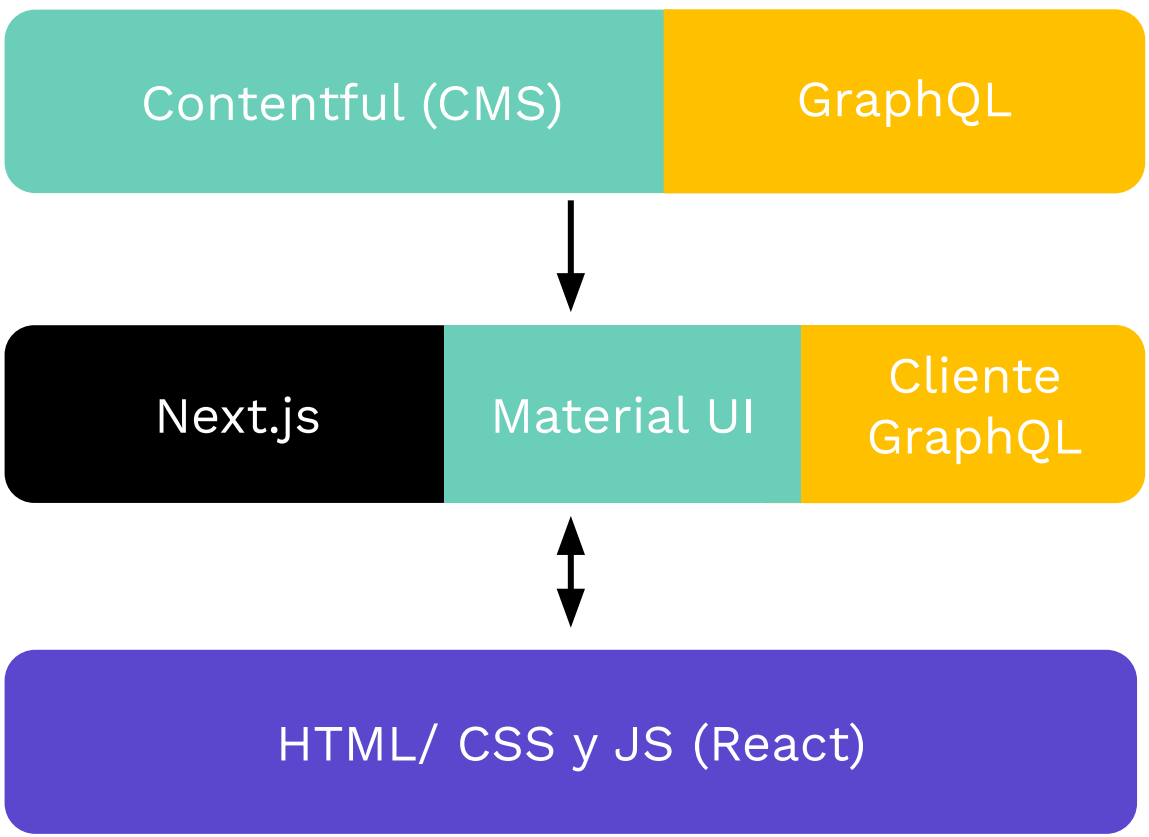


Nuestra app

Nuestra app


Contentful (CMS)
GraphQL
Next.js
Material UI
Cliente GraphQL

Nuestra app


Contentful (CMS)
GraphQL
Next.js
Material UI
Cliente GraphQL
HTML/ CSS y JS (React)

“

En la capa de datos, puedes usar cualquier otro CMS o fuente de datos.

”


Configurando
Contentful y
Tokens


contentful

Importando contenido desde CLI

Contenful

- Configurar tipos de datos y relaciones.
- Crear contenido.
- El **CLI import** del curso carga más de 400 registros en Contenful.


Explorando
nuestra app de
Next.js

Nuestra App

- Node.js y React (Next.js) + TypeScript.
- **ui:** Primitivos de UI.
- **components:** UI + algo de lógica de negocio.
- **api/resources.d.ts:** Tipo de nuestra app.

GraphQL API y autogeneración de código

Reto

Configurar desde 0 el auto generador de GraphQL y todas las queries de la API.

graphql-code-generator.com

Página Home: lista


Moviendo al servidor: getStaticProps

Reto

Realizar algo similar con los autores o categorías.

Página de entrada: detalle

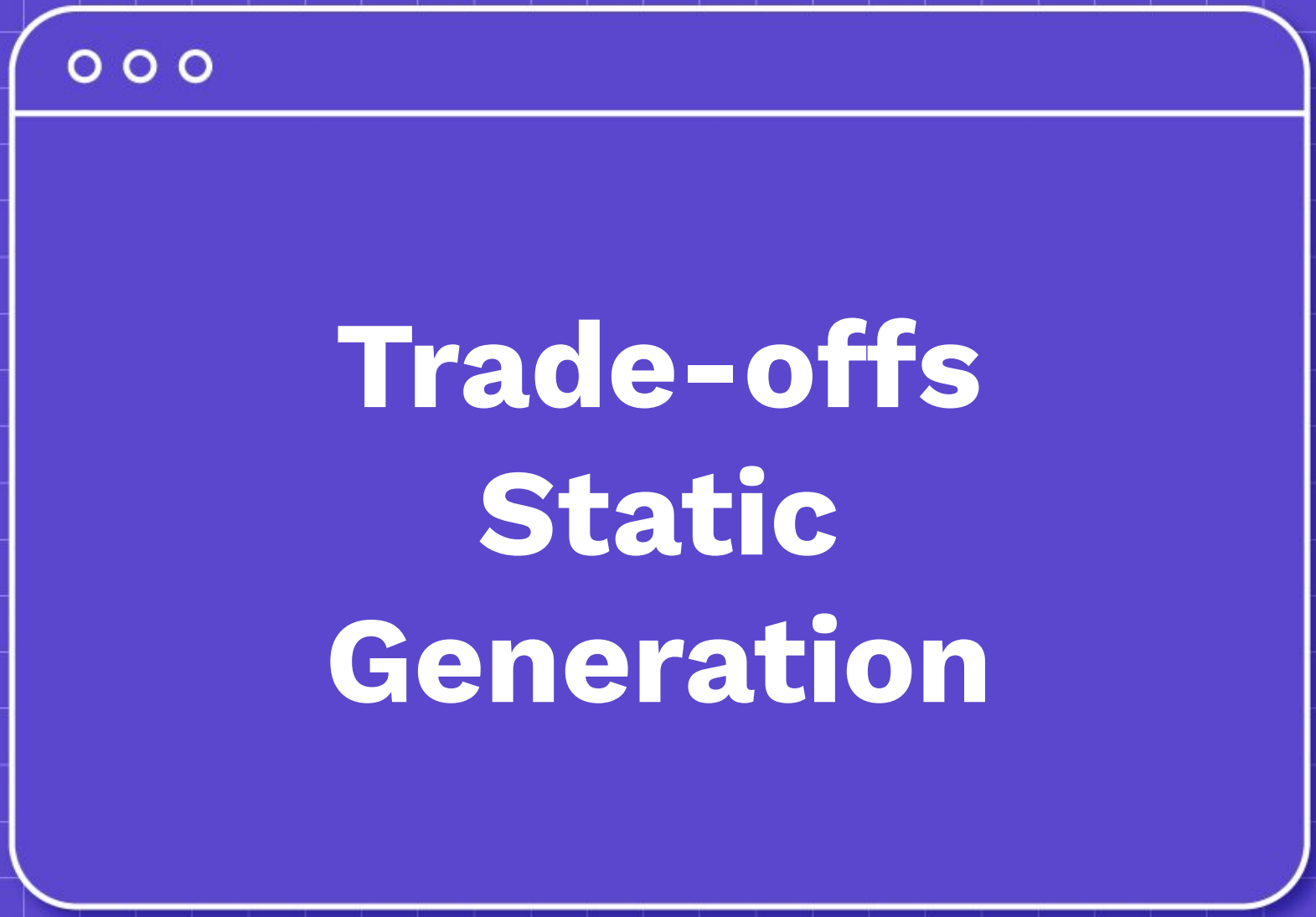

Moviendo al servidor: getStaticPaths

Reto

Realizar algo similar con los autores.

Ej: Sidebar con Entradas Recientes.

Explorando la flexibilidad de getStaticPaths con fileSystem


Trade-offs
Static
Generation

Trade-offs

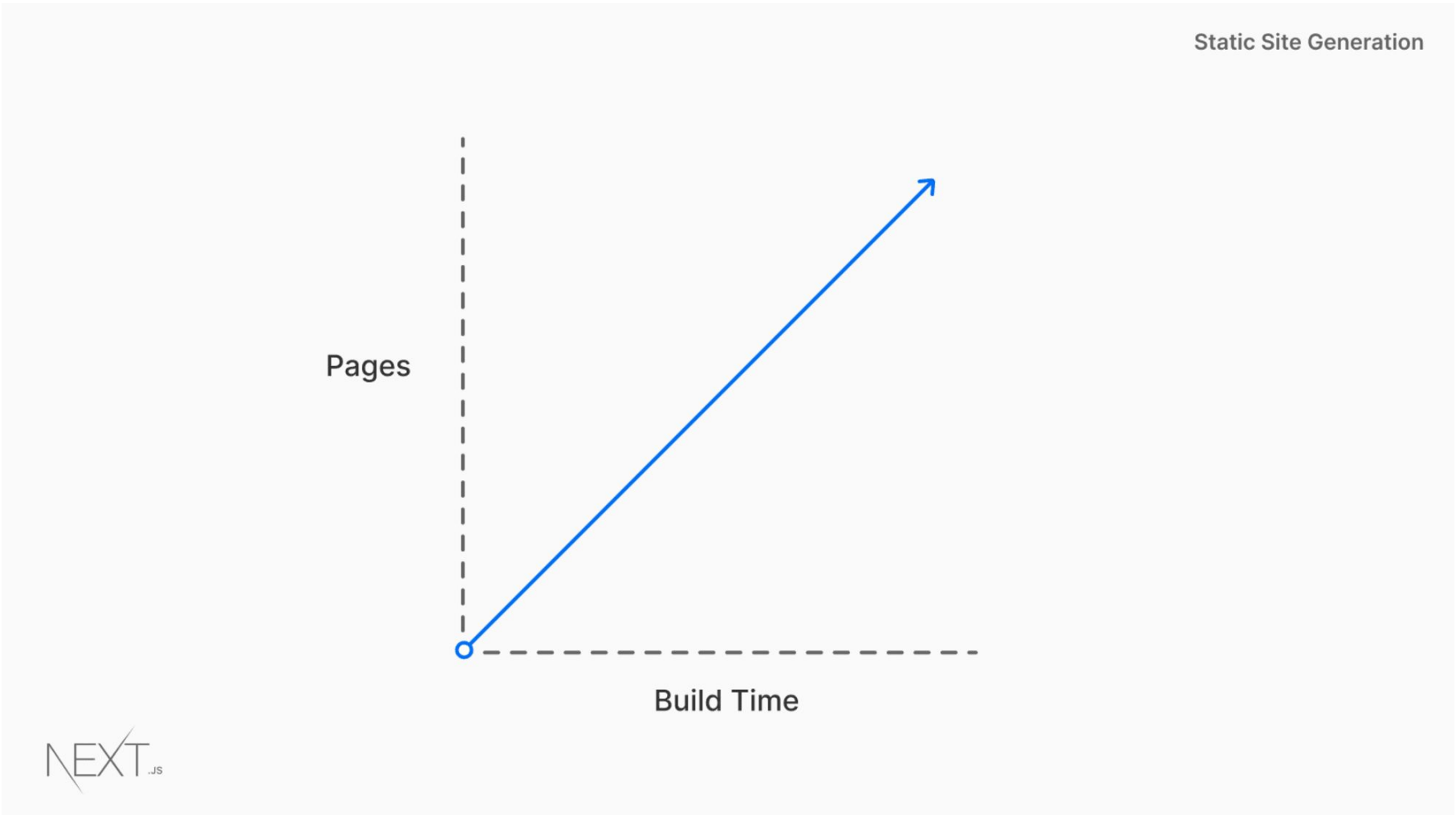
- 😄 Al final, solo son archivos estáticos (html, css y js): el deployment es el más fácil y se puede hacer en cualquier servidor.
- 😄 El SEO y performance de carga serán de los mejores.

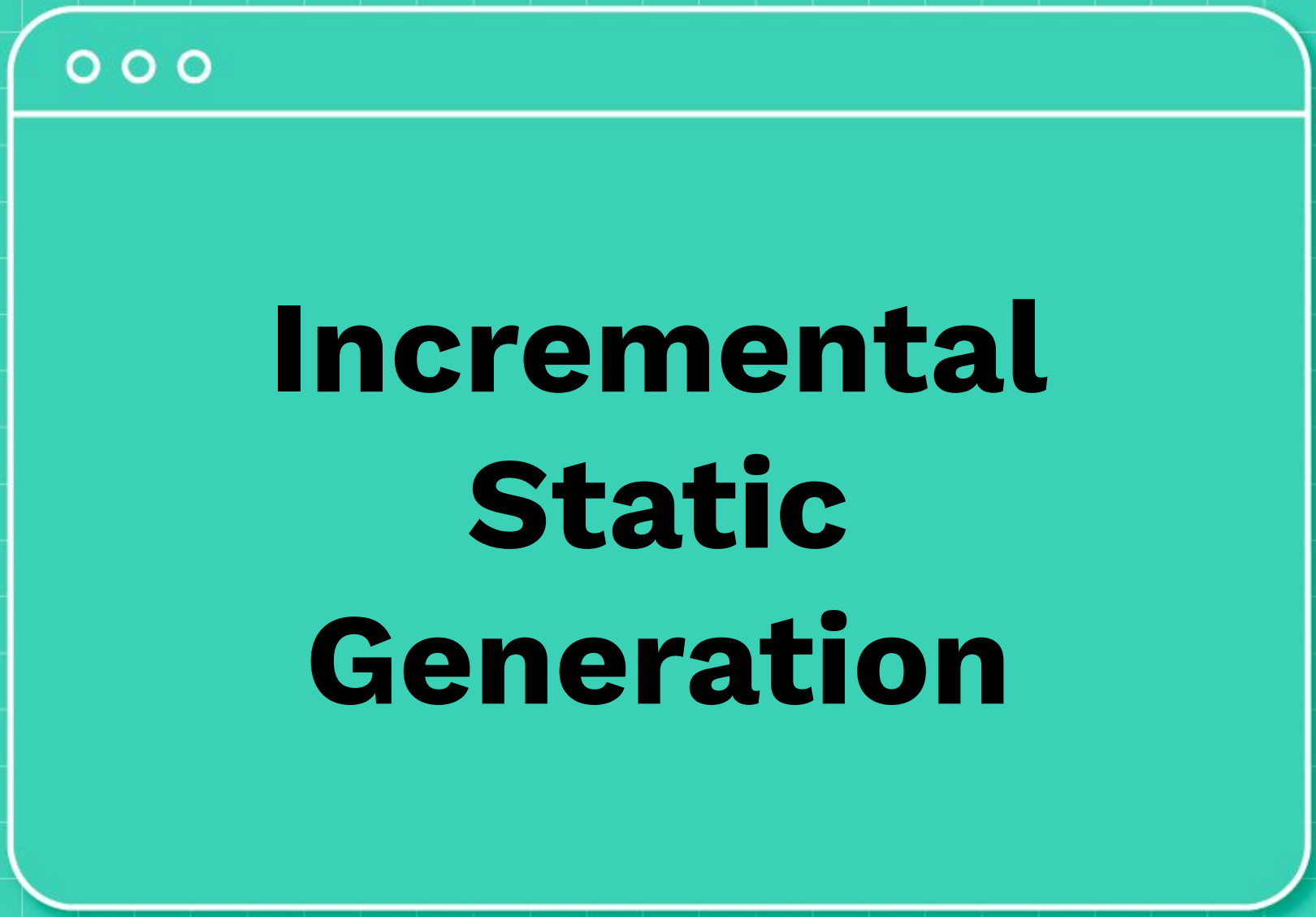
Trade-offs

- ☹️ No todos los sitios se pueden generar de forma estática. Ej.: páginas de usuario o información personalizada, un dashboard.

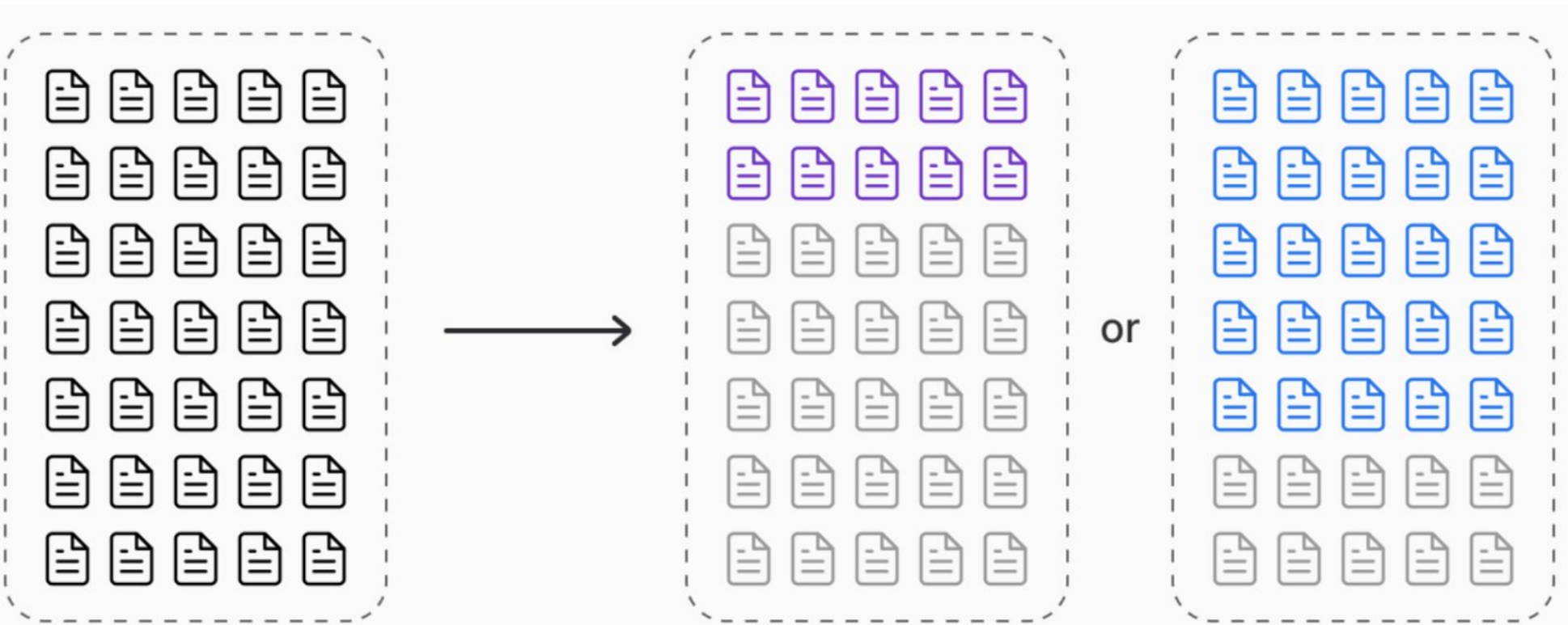
- ☹️ Build time: entre más páginas, más lento el proceso.

Trade-offs




Incremental
Static
Generation

Habilitando Incremental Static Generation

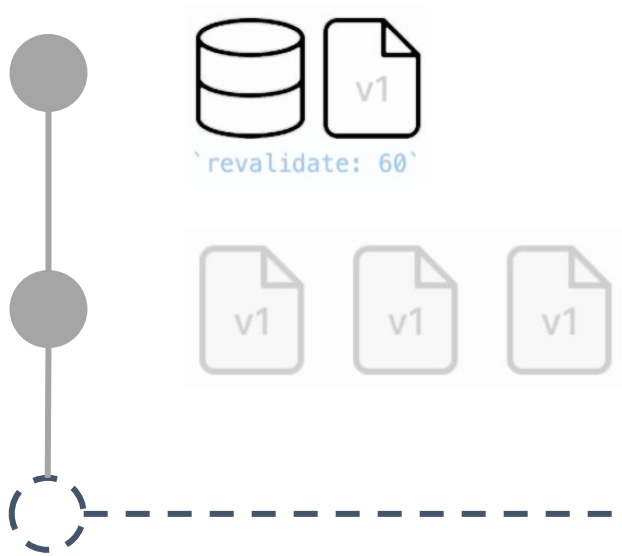

or

Estrategia Fallback vs. Bloqueante

El enfoque stale-while-revalidate

Generación de páginas (build)

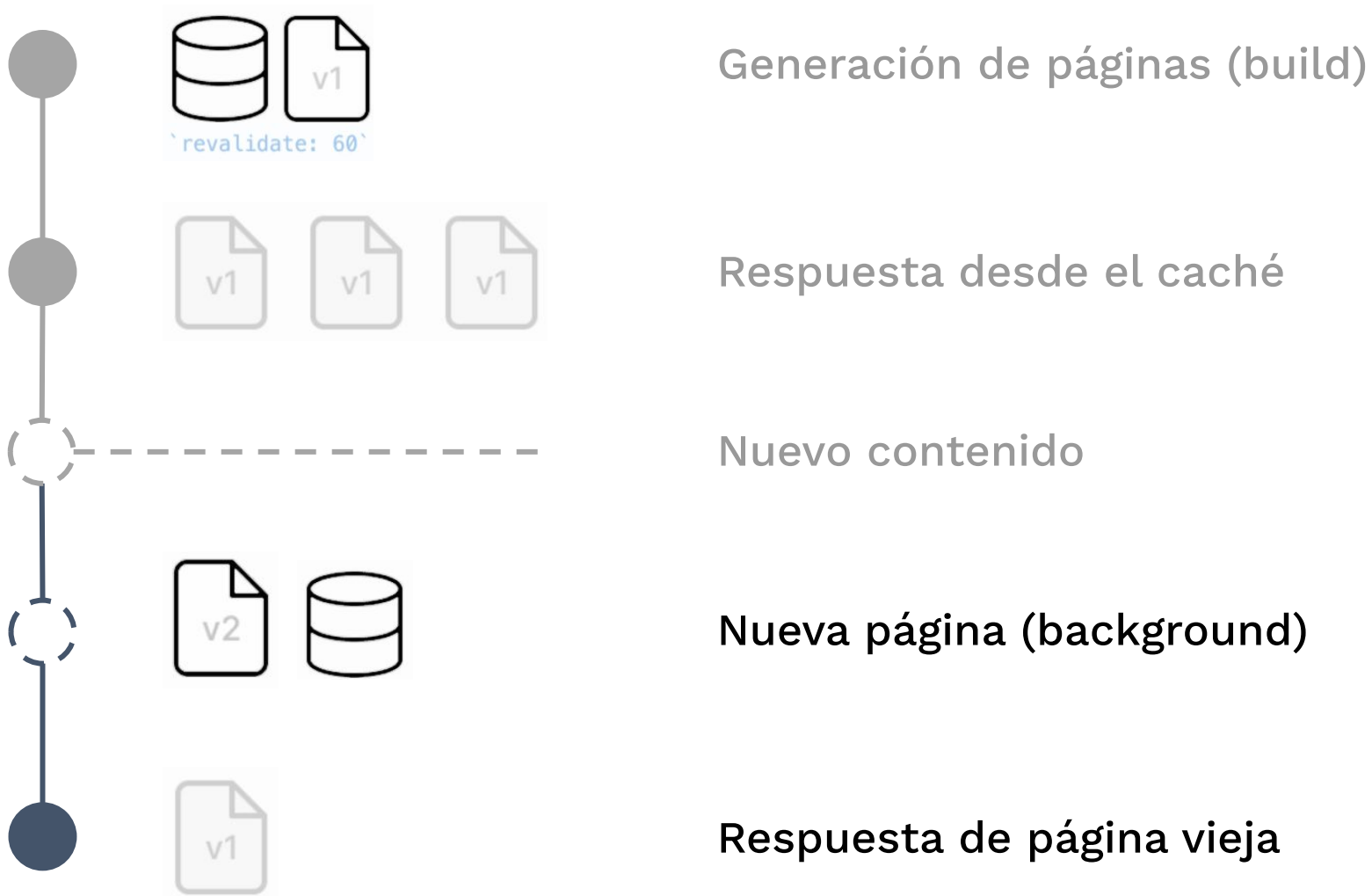
Respuesta desde el caché



Generación de páginas (build)

Respuesta desde el caché

Nuevo contenido


v1
`revalidate: 60`
Generación de páginas (build)
v1
v1
v1
Respuesta desde el caché
Nuevo contenido
v2
Nueva página (background)
v1
Respuesta de página vieja


v1
`revalidate: 60`
v1
v1
v1
v2
v1
v2
Generación de páginas (build)
Respuesta desde el caché
Nuevo contenido
Nueva página (background)
Respuesta de página vieja
Respuesta con la nueva página

Trade-offs Incremental Static Generation

¡ Trade-offs

- 😄 La flexibilidad de server-side rendering con las bondades de static generation.
- 🤦‍♀️ Requiere un servidor con Node.js.
- 😄 El build-time no aumenta con el número de páginas.

¡ Trade-offs

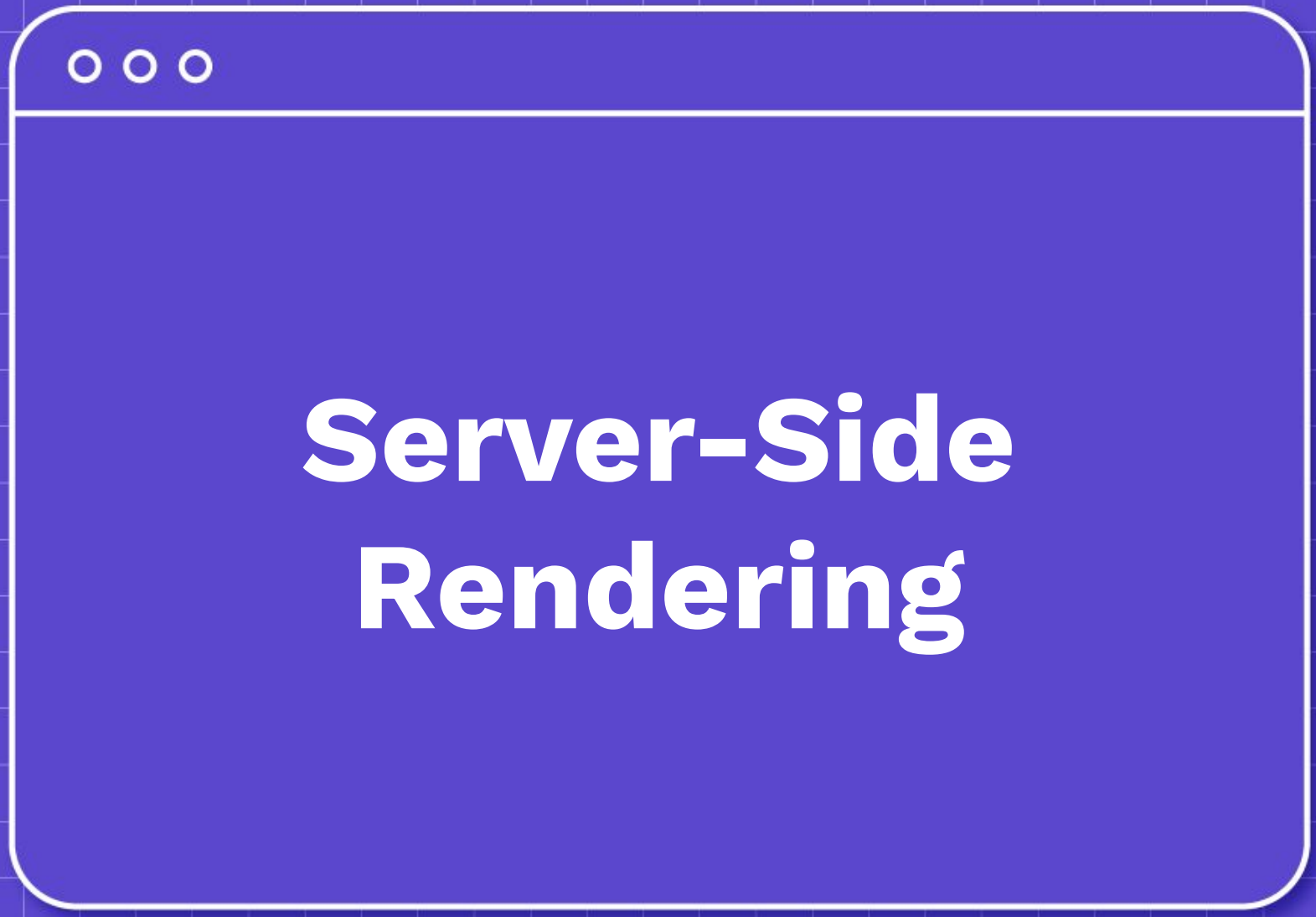
- 😁 La revalidación brinda mucha más flexibilidad.
- 🤦‍♀️ Así mismo puede ser peligroso, ej.: no poder ajustar los tiempos de revalidación ante un enlace que se vuelve *viral*.

Trade-offs

- ☹️ No es apto para todas las páginas. Ej.: páginas de usuario o información personalizada, dashboard en tiempo real.

- 🤔 No es un problema en sitios con pocas páginas.

Otras alternativas


Server-Side
Rendering

¡SSR

- 😁 La información siempre estará actualizada.
- 😁 Poder modificar la respuesta con base en la petición puede ser muy conveniente.
- 🤔 *Golpear* el servidor por cada petición puede ser costoso.


Exportar HTML

Exportar HTML

Se producen archivos estáticos y el servidor de Node.js es desacoplado.

¡ Trade-offs: Exportar

- 😁 No hay backend, por tanto no hay servidores por mantener.
- 😁 Se puede subir a servidores de archivos estáticos, como GitHub Pages.

¡ Trade-offs: Exportar

😕 Muchas funcionalidades se deshabilitarán:

- Server-side rendering (getServerSideProps)
- Incremental Site Generation
- Revalidación
- Rutas API
- Internacionalización
- next/image*

Siguientes pasos