BLAZINGLY FAST

BLAZINGLY FAST

WEBSITES WITH PARALLEL COMPOSITION

Created by Piotr Hałas from Crazy Goat Software

AGENDA

AGENDA

1. WHOAMI
2. What is page composition
3. Evolution of available technologies
4. Technology today - where we stand
5. Live demo x2 - hope it works
6. Looking Ahead: go-mesi project
7. Q&A

WHOAMI --ENV=DEV

- Professionally coding in PHP since 2007
  - Symfony from version 1.0
  - Explored Slim and Zend frameworks.
  - Workerman/Webman i (Open)Swoole
- Hobby
  - C/C++ embedded, ESP-01/12, Arduino
  - Golang - tooling, PHP + Golang
- Small DevOps - CI/CD, Docker, Systems Architecture
- Open Source contributor
- Engineering mindset over pure theory

WHOAMI --ENV=LIVE

- Proud Husband and Father of Two Daughters
- Run a small farm Hałaśliwa Zagroda
  - Farm animals: horse, donkeys, goats, sheep, pigs, ducks, geese, chickens, guinea fowl; domestic: dogs, cats, fish
  - We produce goat cheese and sell eggs, organize school trips, birthday parties for children
- Passion for Woodworking
- Enjoy Welding Projects
- Love for Classic Vehicles
- Youtuber

THE FIRST RIDDLE

THE FIRST RIDDLE

WHICH ANIMAL IS NOT ON OUR FARM?

THE FIRST RIDDLE

WHICH ANIMAL IS NOT ON OUR FARM?

Horse	Duck	Dog
Cat	Goose	Goat
Donkey	Fish	Cow

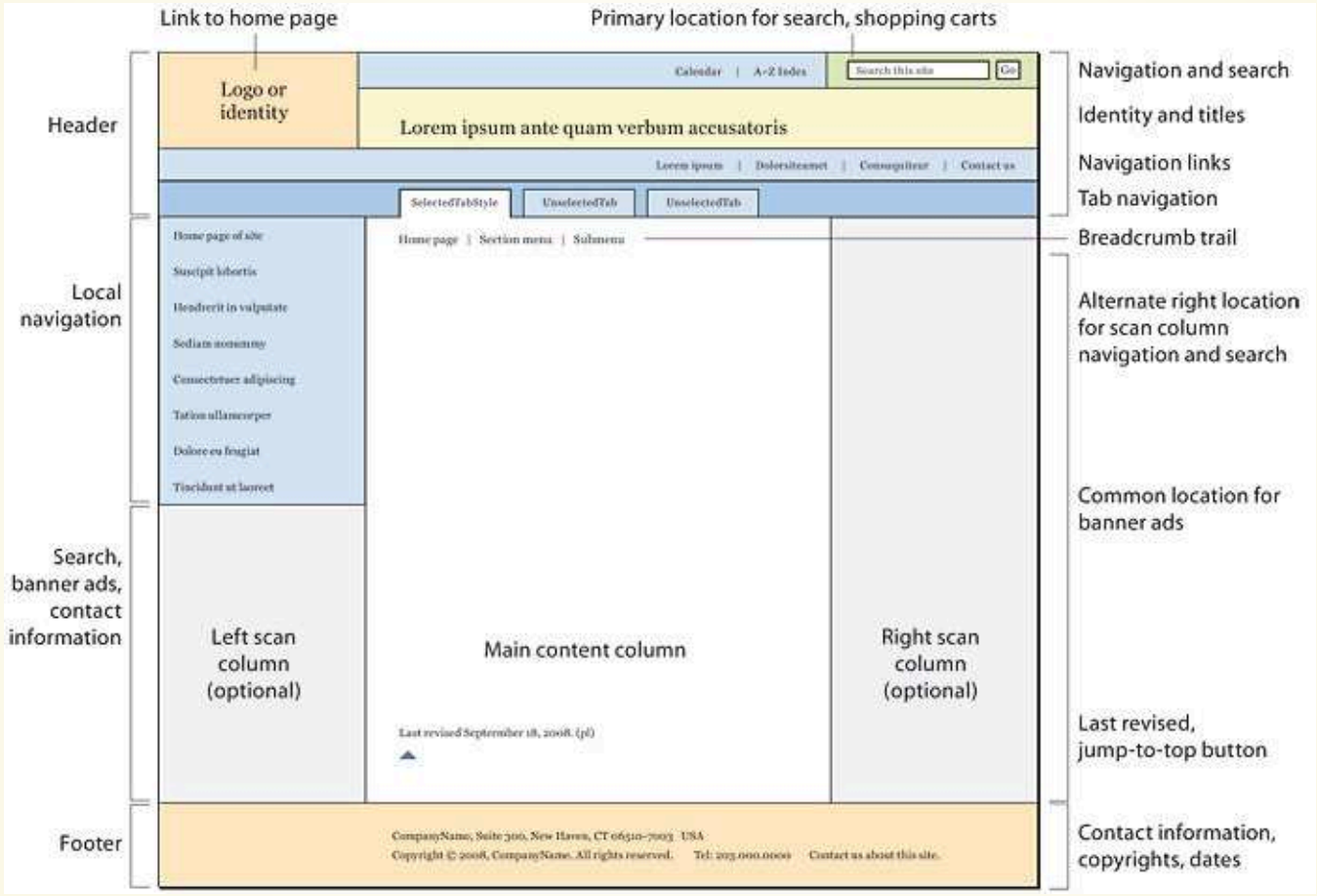
COW

PAGE COMPOSITION?

PAGE COMPOSITION?

Page composition is now primarily associated with web development, referring to the process of assembling a webpage from multiple independent components. This approach allows for better modularity, scalability, and performance optimization.

PAGE COMPOSITION - WEB EXAMPLE



https://webstyleguide.com/wsg3/6-page-structure/3-site-design.html

PAGE COMPOSITION

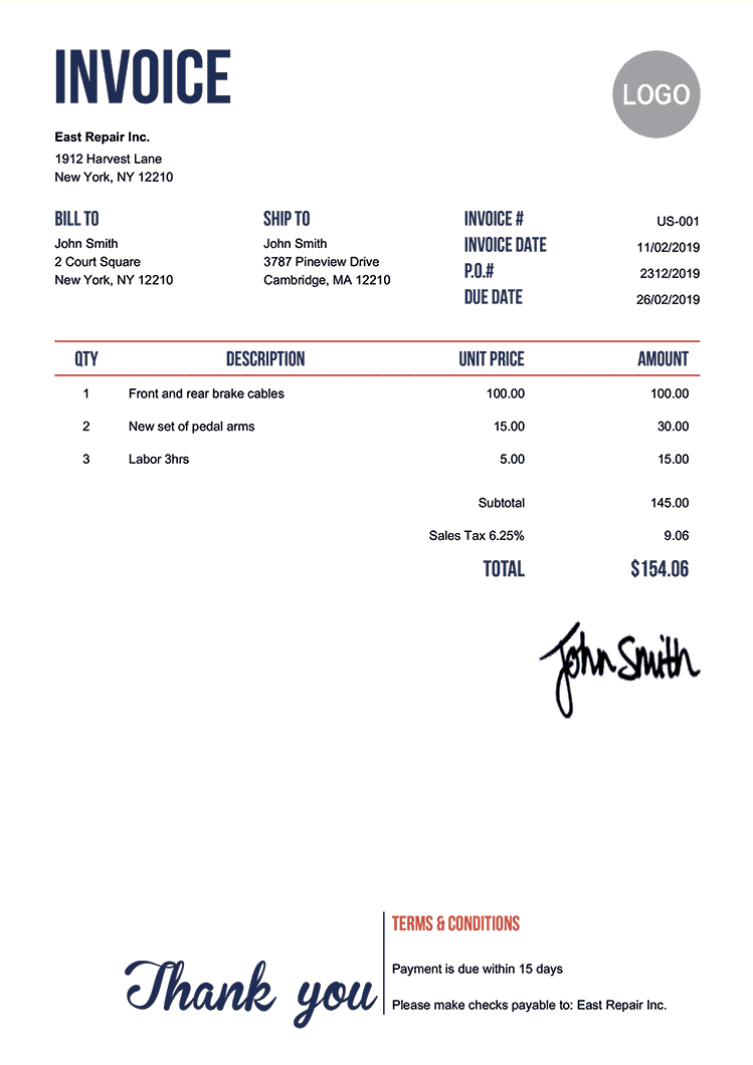
Page composition in reality, it is an old concept that has been used for centuries in:

- **Books & Newspapers** – Consistent layout with fixed sections like headers, footers, and sidebars.
- **Business Letterheads** – Predefined structure with repeating elements (e.g., company logo, recipient details, footer).
- **Invoices** - Structured format with recurring elements like company details, recipient information, and itemized lists.

PAGE COMPOSITION - BOOKS

PAGE COMPOSITION - BUSINESS LETTERHEAD

PAGE COMPOSITION - INVOICE



INVOICE

LOGO

East Repair Inc.
1912 Harvest Lane
New York, NY 12210

BILL TO
John Smith
2 Court Square
New York, NY 12210

SHIP TO
John Smith
3787 Pineview Drive
Cambridge, MA 12210

INVOICE #	US-001
INVOICE DATE	11/02/2019
P.O.#	2312/2019
DUE DATE	26/02/2019

QTY	DESCRIPTION	UNIT PRICE	AMOUNT
1	Front and rear brake cables	100.00	100.00
2	New set of pedal arms	15.00	30.00
3	Labor 3hrs	5.00	15.00
		Subtotal	145.00
		Sales Tax 6.25%	9.06
		TOTAL	$154.06

John Smith

Thank you

TERMS & CONDITIONS

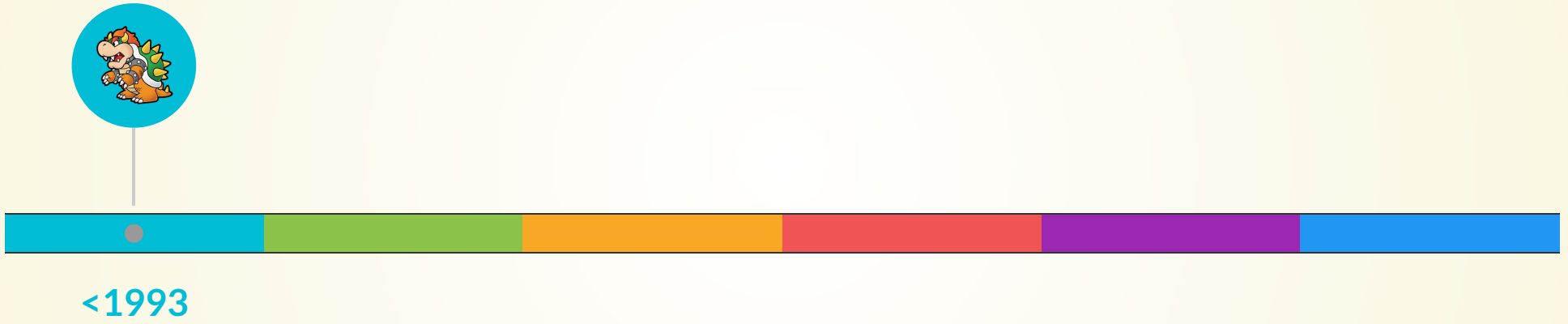
Payment is due within 15 days

Please make checks payable to: East Repair Inc.

HISTORY OVERVIEW

HISTORY OVERVIEW

Dark ages



<1993

HISTORY OVERVIEW

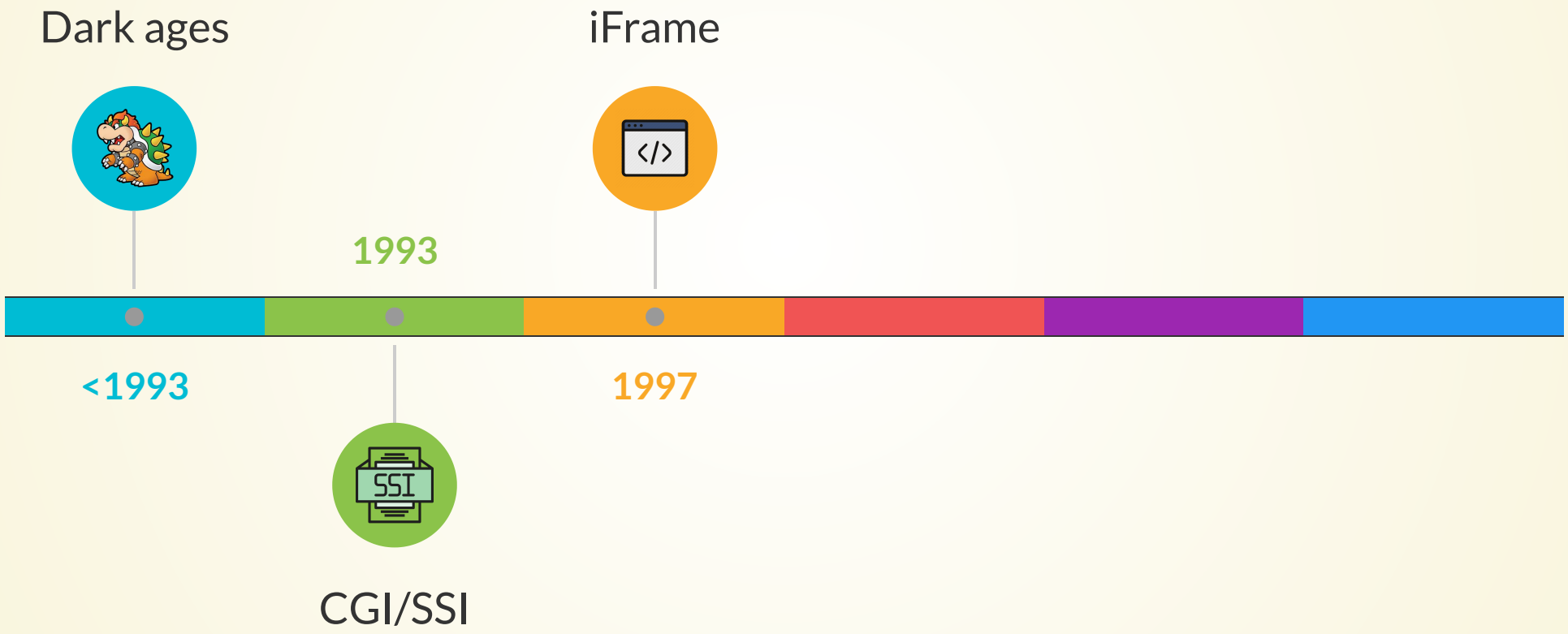
Dark ages

1993

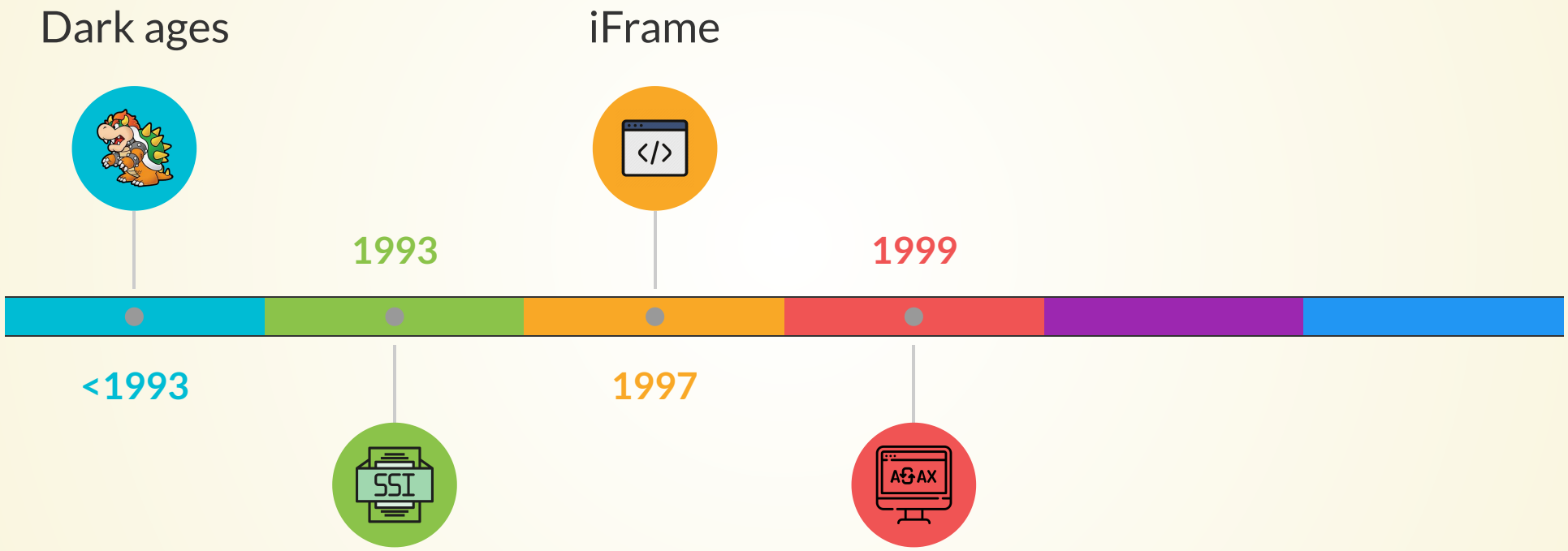
<1993

CGI/SSI

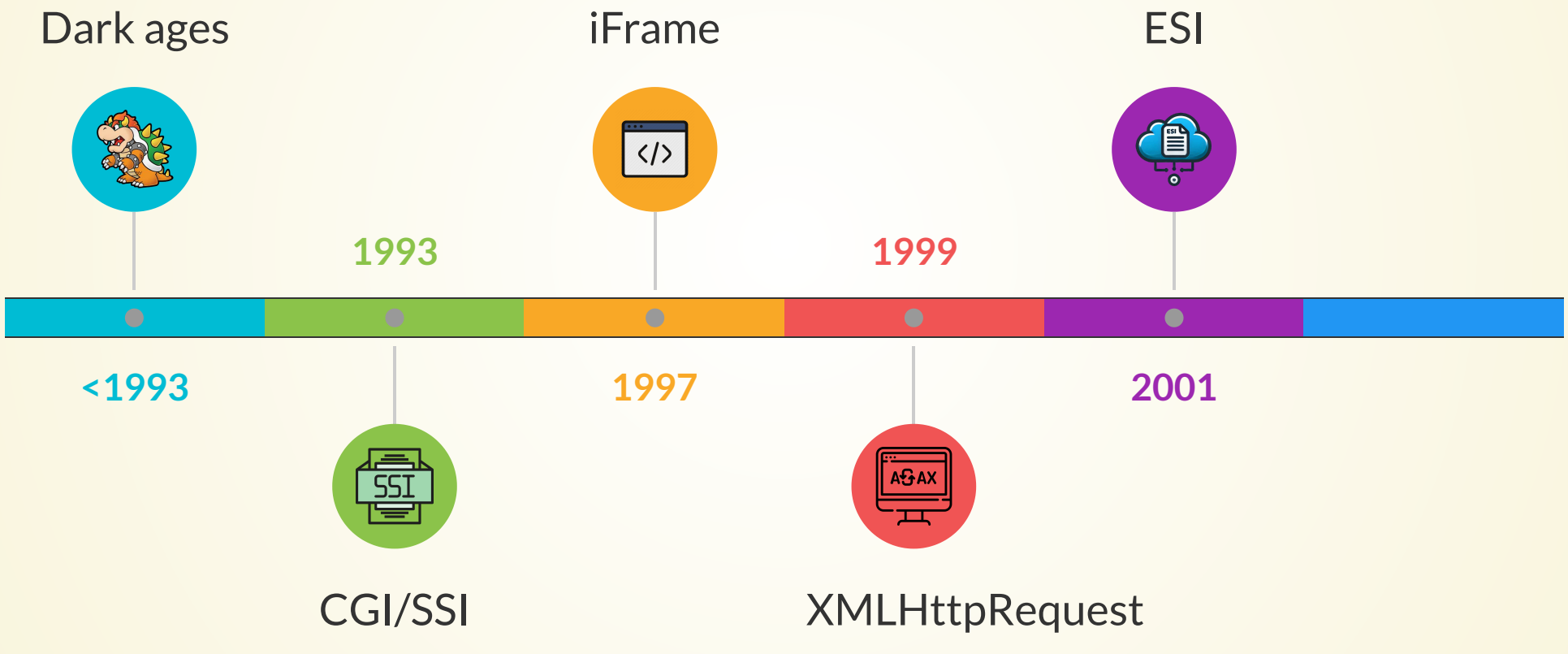
HISTORY OVERVIEW



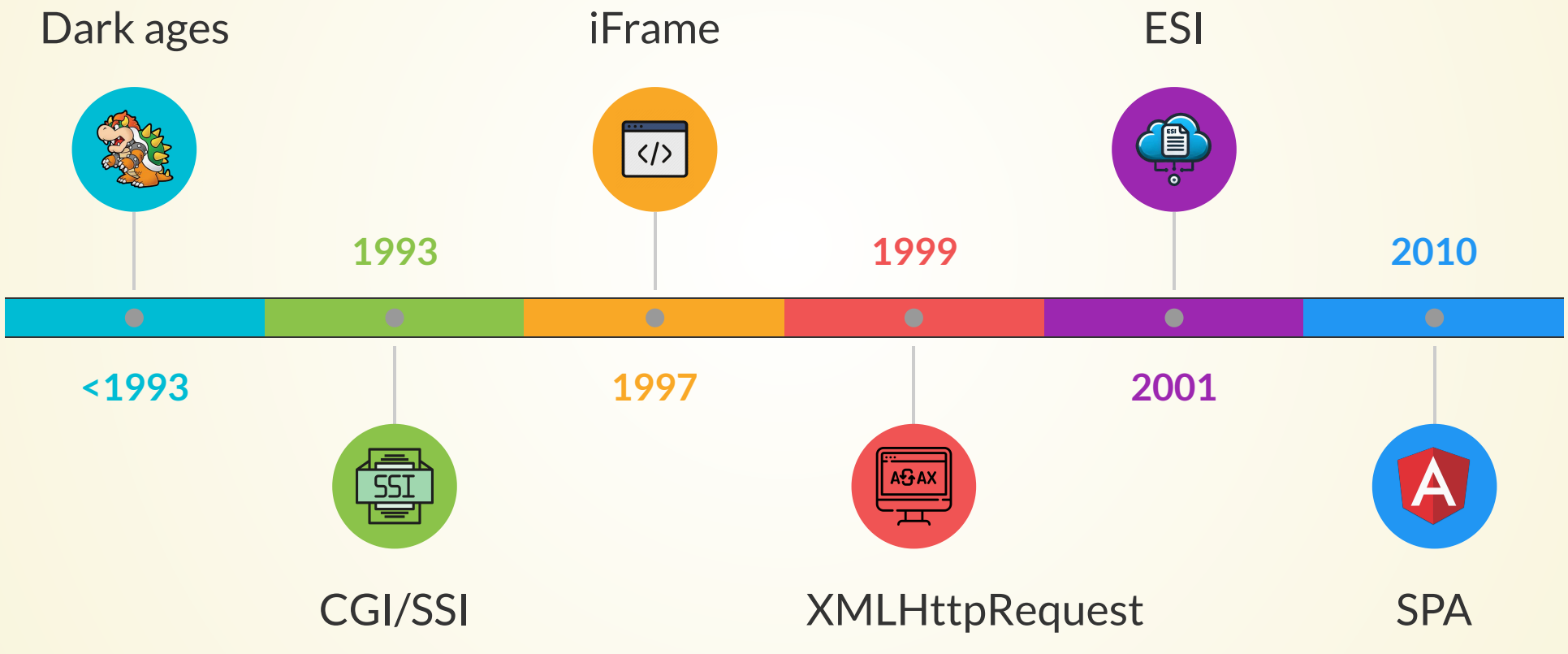
HISTORY OVERVIEW



HISTORY OVERVIEW



HISTORY OVERVIEW



SECOND RIDDLE

SECOND RIDDLE

WHICH COMPANY CREATED THE IFRAME

SECOND RIDDLE

WHICH COMPANY CREATED THE IFRAME

Hint: Don't let the name fool you, it wasn't Apple

MICROSOFT

The <iframe> element was introduced by Microsoft in 1997 as part of Internet Explorer 3.0. It officially became part of the HTML 4.0 standard, which was approved by the W3C (World Wide Web Consortium) in December 1997.

TECHNOLOGY OVERVIEW

TECHNOLOGY OVERVIEW

Page composition can be categorized into four types based on where the assembly occurs: backend composition, server-side composition, edge-side composition, and client-side composition. Each approach determines where and how different components are combined to render the final webpage, impacting performance, flexibility, and caching strategies.

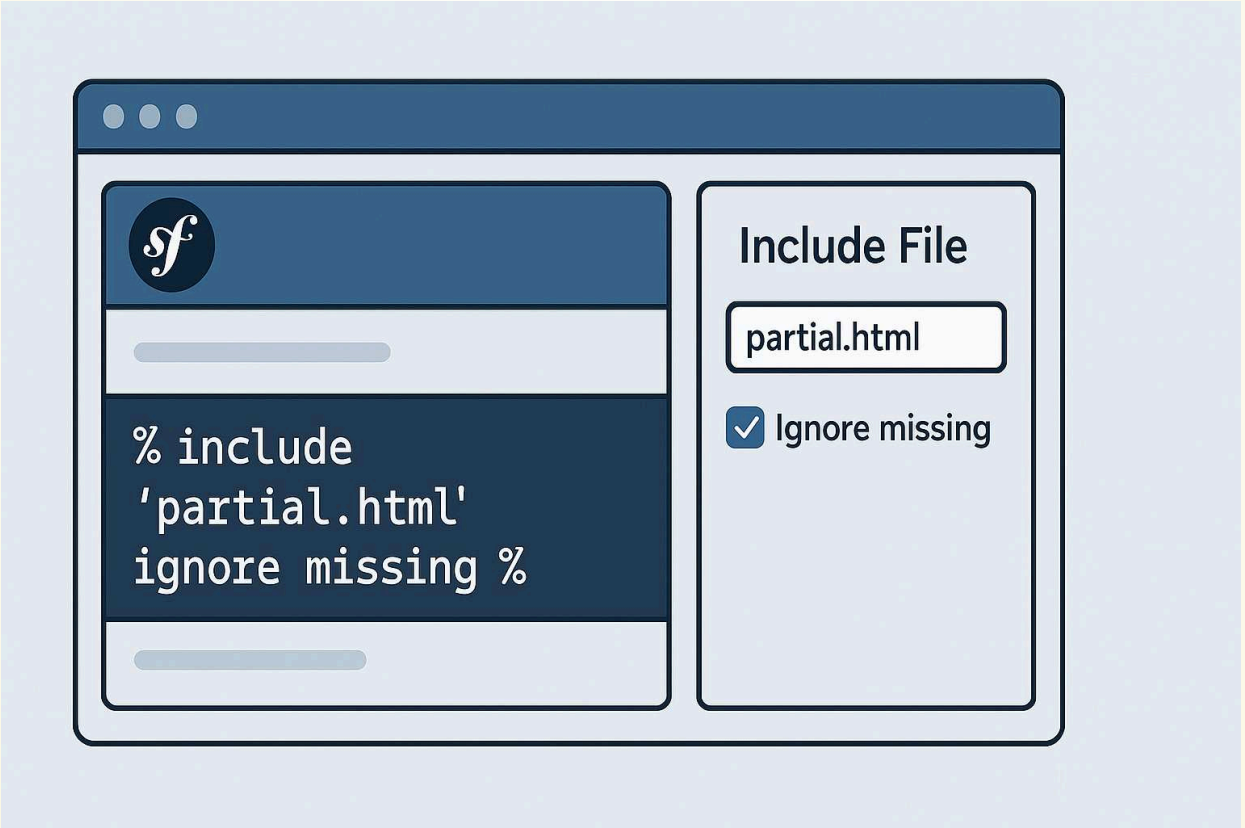
Backend	Server Side Include	Edge Side Include	Client Side
Symfony, node.js, django	Apache, NGINX	Akamai, fastly, VARNISH CACHE	htmx, AJAX

WHAT WE WANT TO ACHIEVE

1. Concurrent rendering
2. Microservices/different programming languages
3. Easy caching

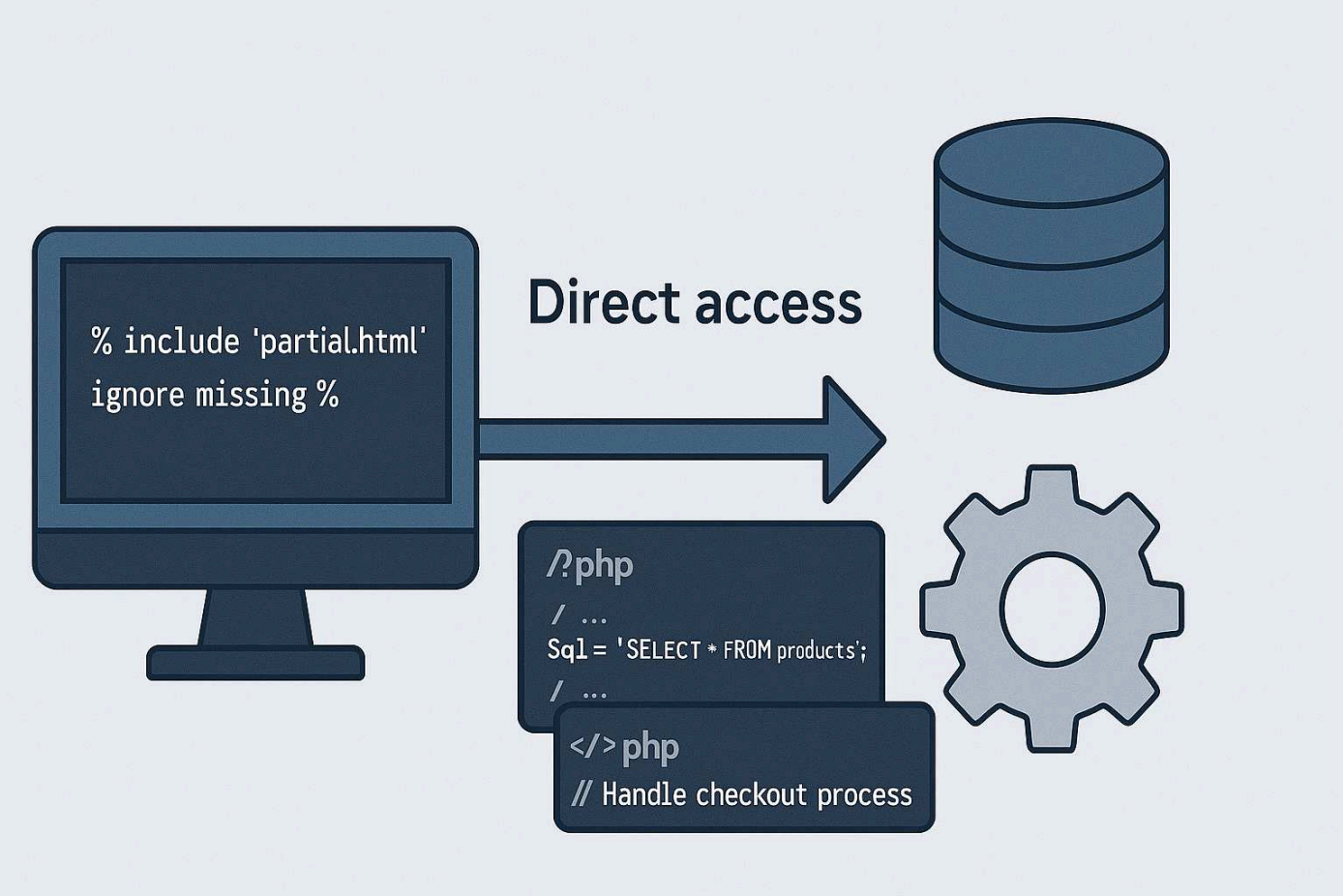
BACKEND COMPOSITION ✅ PROS

Easy Implementation – Simple to develop and maintain within a monolithic backend.



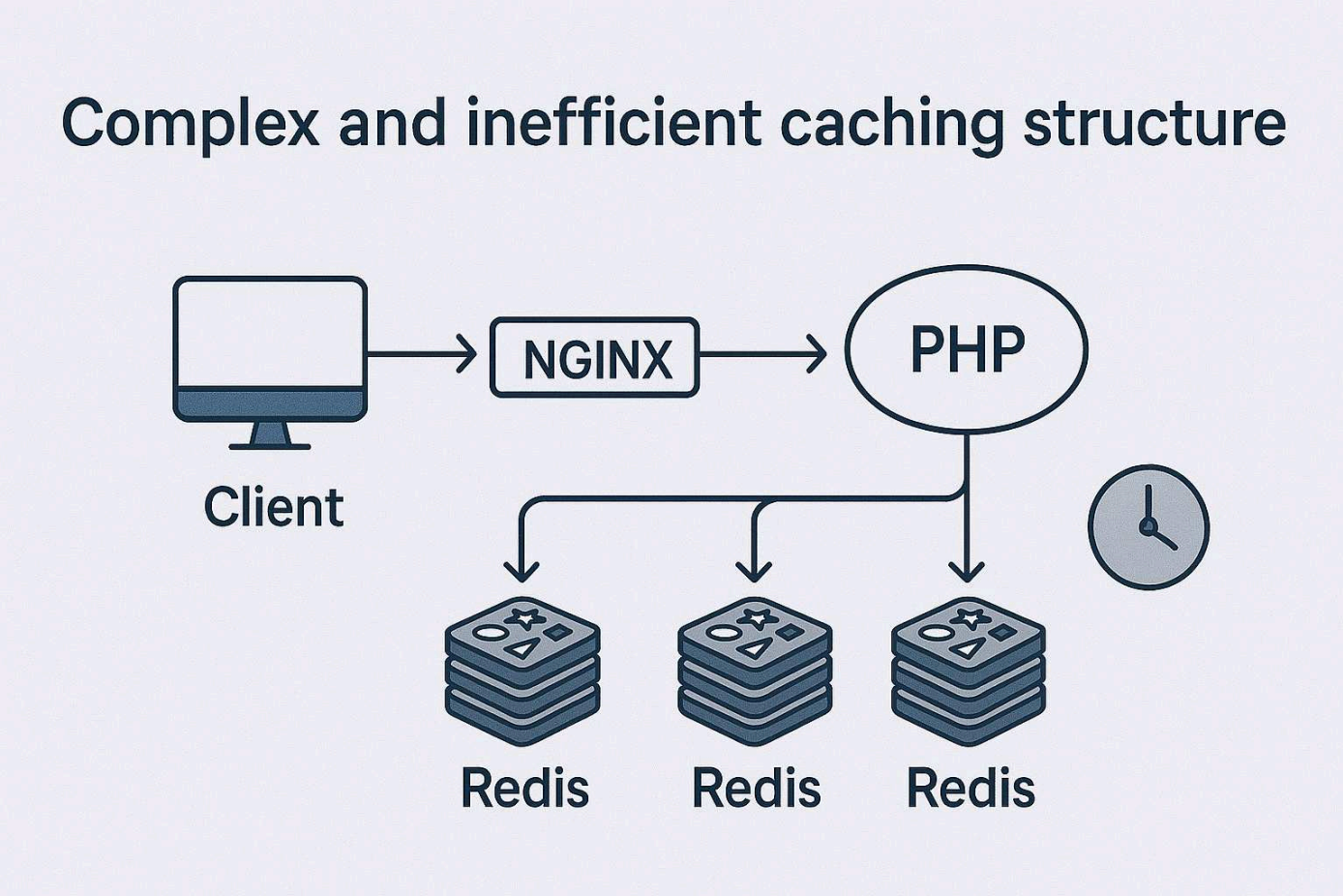
BACKEND COMPOSITION ✅ PROS

Strong Integration – Direct access to databases and business logic.



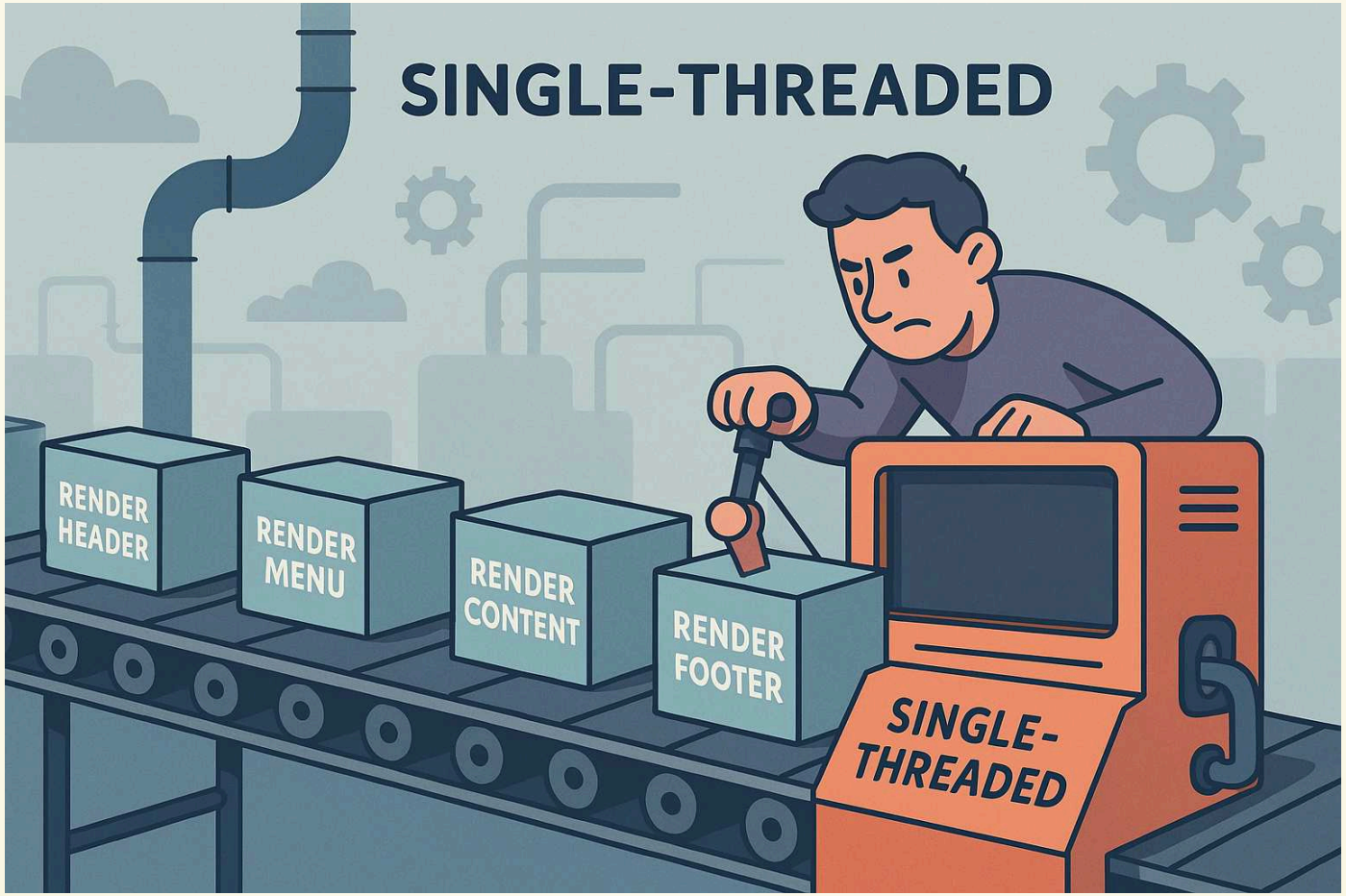
BACKEND COMPOSITION ❌ CONS

Caching Challenges – Difficult to cache dynamic content efficiently.



BACKEND COMPOSITION ❌ CONS

Performance Issues – Most web languages are single-threaded.



BACKEND COMPOSITION ❌ CONS

Monolithic Architecture – Encourages tightly coupled systems.

TECHNOLOGY OVERVIEW - BACKEND COMPOSITION

✅ Pros:

- **Easy Implementation** – Simple to develop and maintain within a monolithic backend.
- **Strong Integration** – Direct access to databases and business logic.

❌ Cons:

- **Caching Challenges** – Difficult to cache dynamic content efficiently.
- **Performance Issues** – Most web languages are single-threaded, potentially causing bottlenecks.
- **Monolithic Architecture** – Encourages tightly coupled systems, reducing flexibility and scalability.

WHAT CAN WE FIX

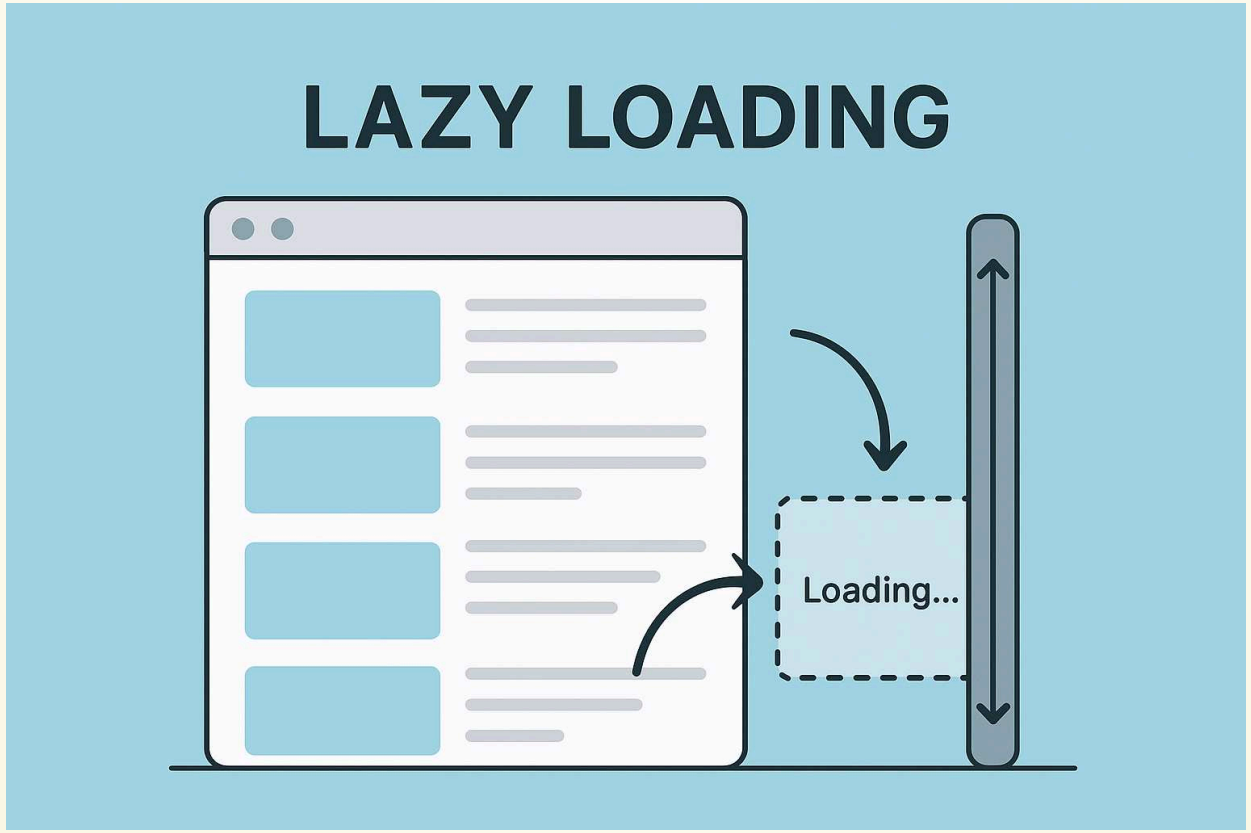
PERFORMANCE ISSUES

MONOLITHIC ARCHITECTURE


STOP!
JS
SLOW WEBSITES

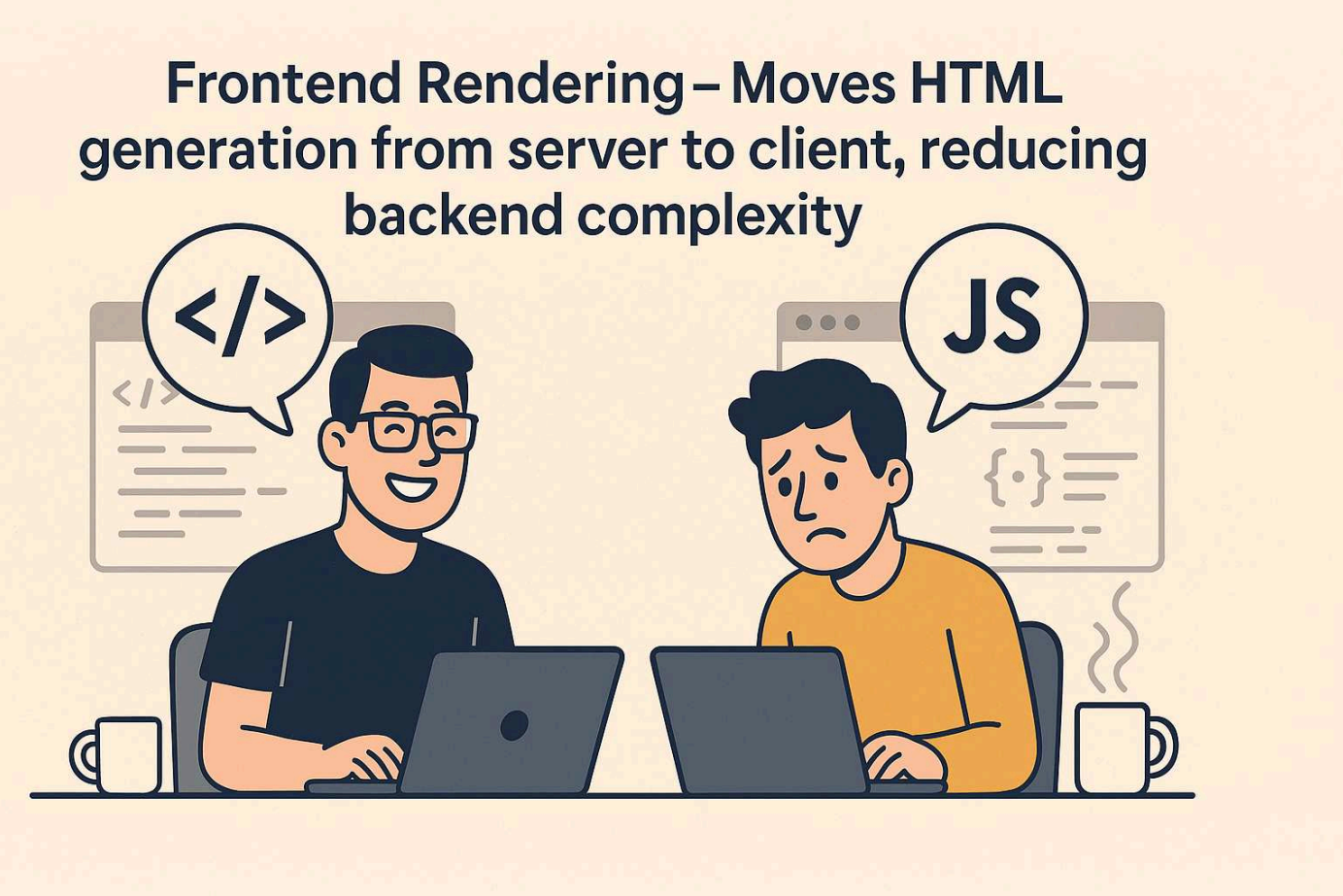
CLIENT SIDE ✅ PROS

Lazy Loading – Loads only the necessary components, improving performance and user experience.



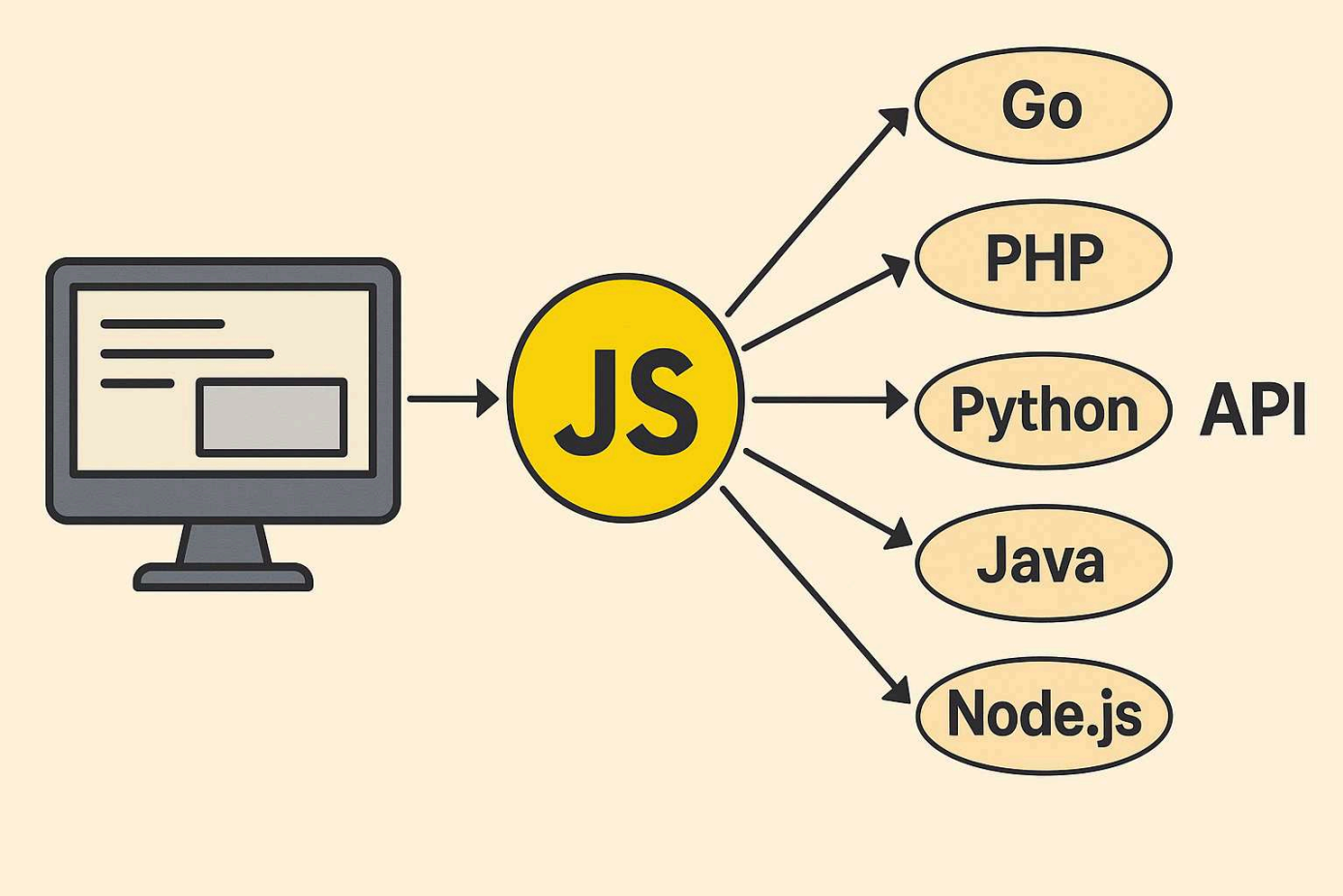
CLIENT SIDE ✅ PROS

Frontend Rendering – Moves HTML generation from server to client.



CLIENT SIDE ✅ PROS

Flexible and Scalable – Works well with microservices and APIs.

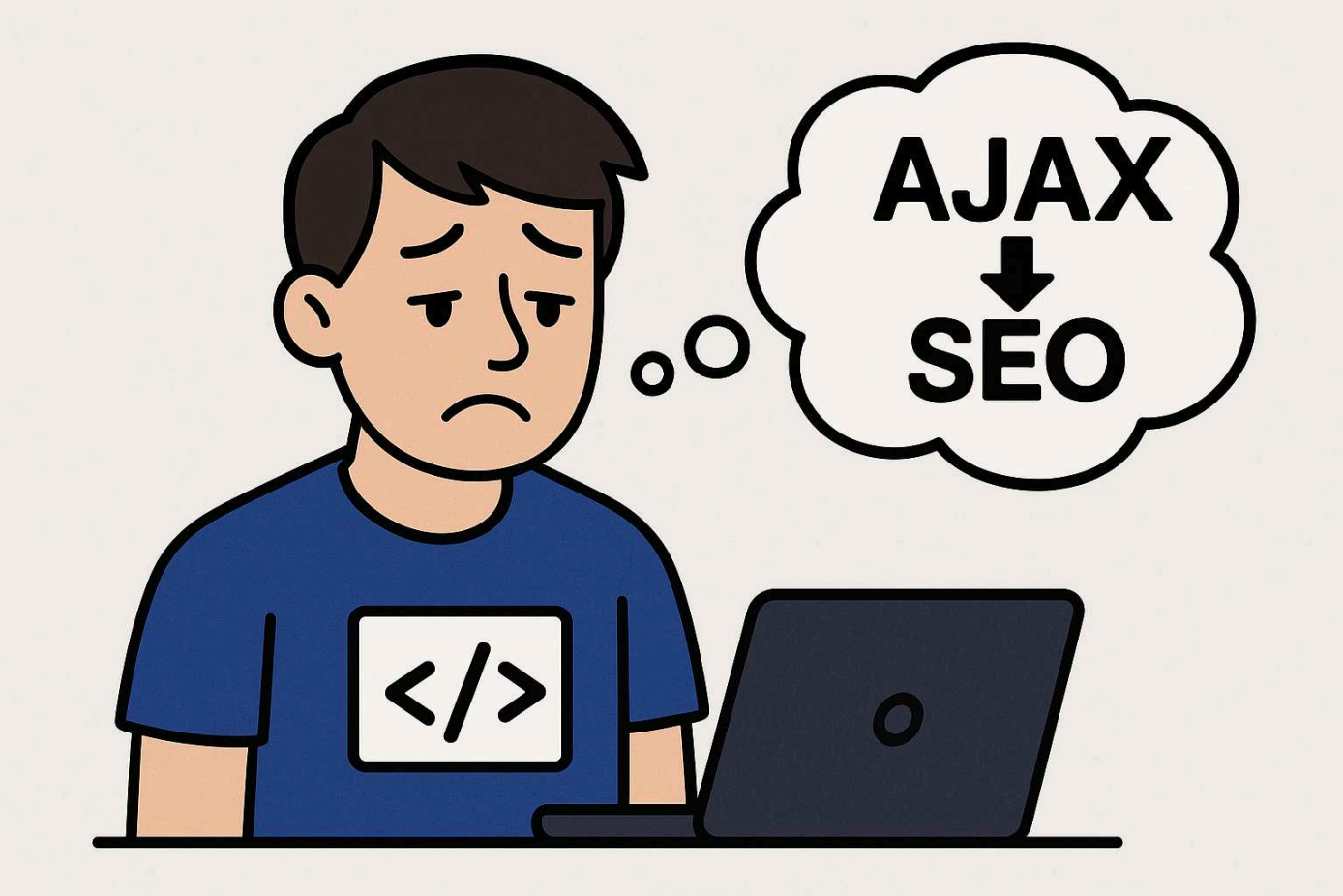


CLIENT SIDE ❌ CONS

Slower Initial Load – More processing happens in the browser.

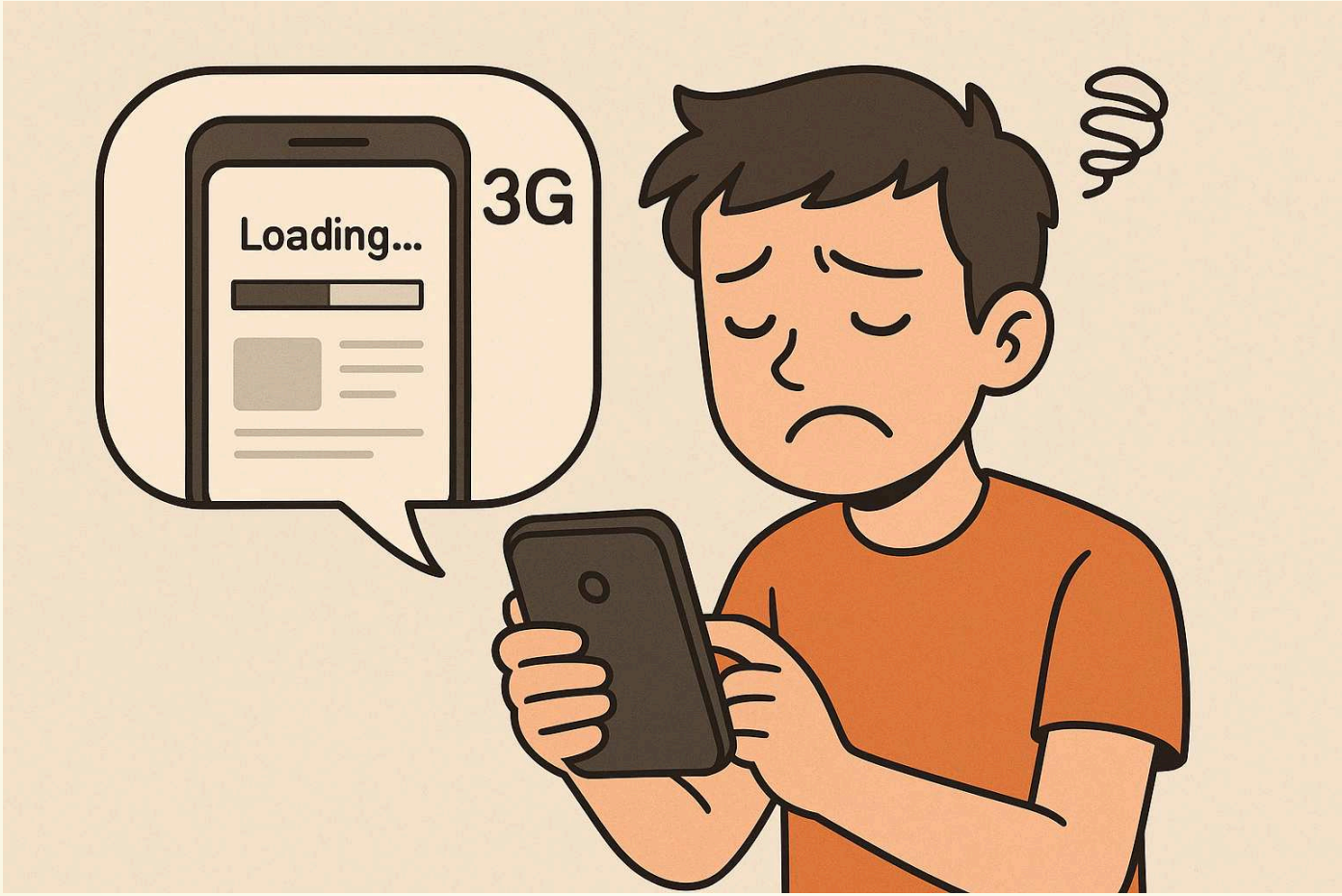
CLIENT SIDE ❌ CONS

SEO Challenges – Search engines may struggle.



CLIENT SIDE ❌ CONS

Inconsistent Performance – Depends on user device and network



TECHNOLOGY OVERVIEW - CLIENT SIDE

✅ Pros:

- **Lazy Loading** – Loads only the necessary components, improving performance and user experience.
- **Frontend Rendering** – Moves HTML generation from server to client, reducing backend complexity.
- **Flexible and Scalable** – Works well with microservices and APIs, allowing independent frontend updates.

❌ Cons:

- **Slower Initial Load** – More processing happens in the browser, which may delay rendering.
- **SEO Challenges** – Search engines may struggle to index client-rendered content without proper server-side support.
- **Inconsistent Performance** – Heavily depends on the user's device and network speed.

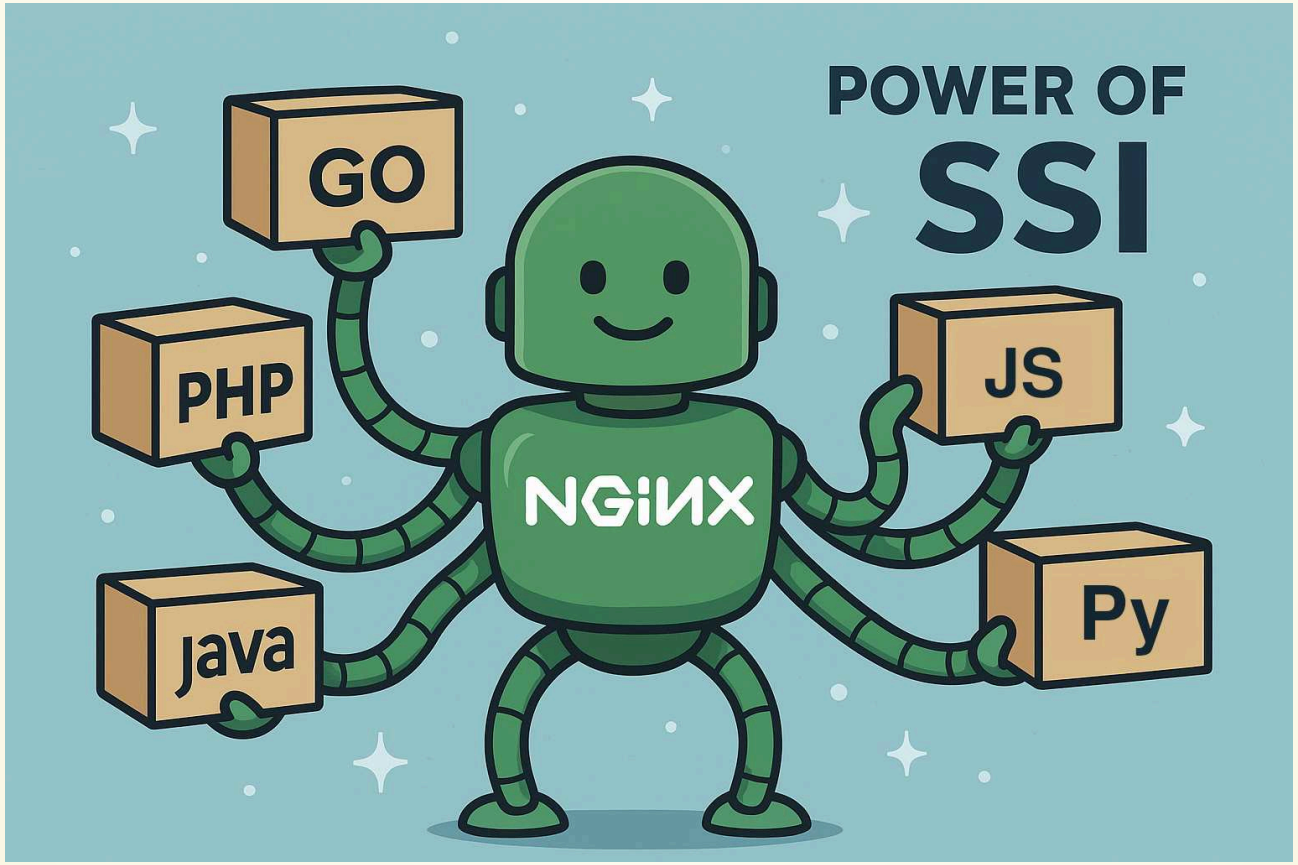
WHAT CAN WE FIX

INCONSISTENT PERFORMANCE

SLOWER INITIAL LOAD

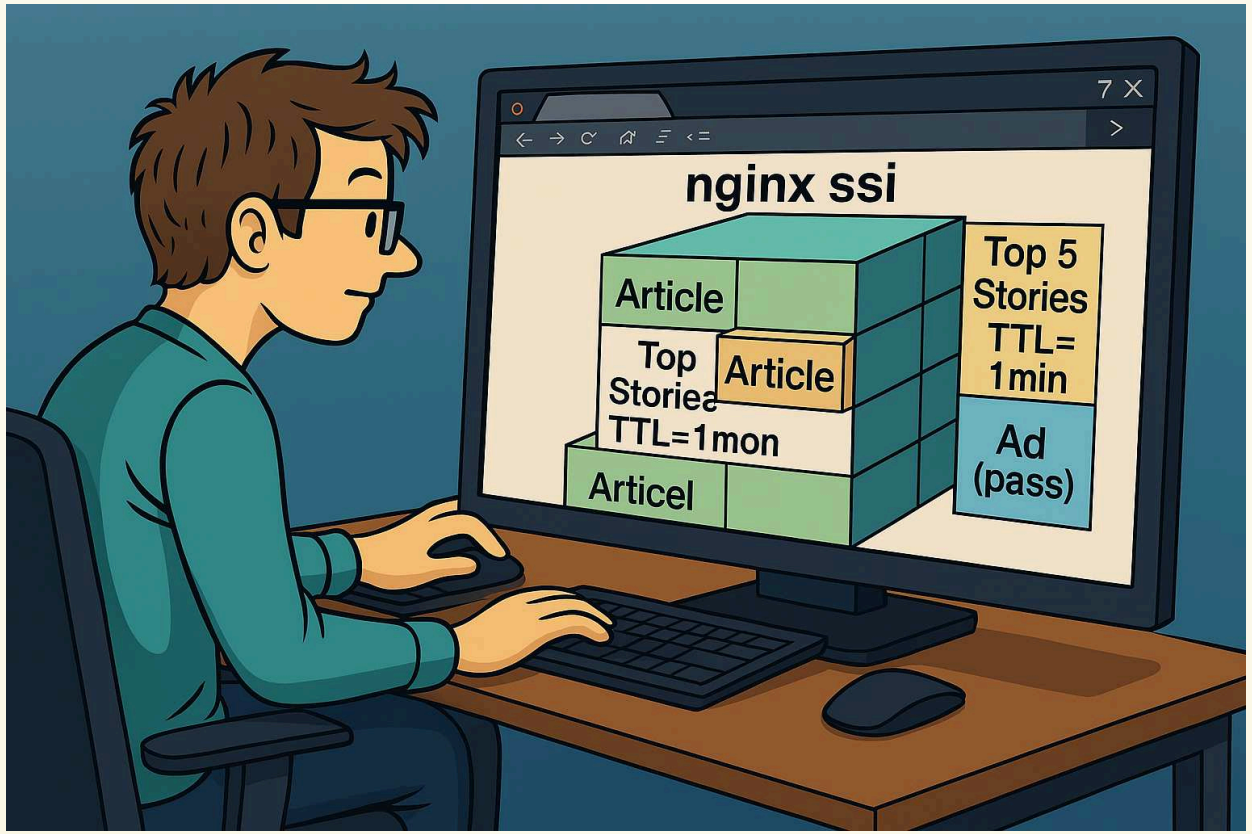
And SEO

SERVER SIDE INCLUDE (SSI)



SERVER SIDE INCLUDE (SSI)

Server Side Includes (SSI) is a server-side feature that allows web pages to include content from other sources before being sent to the client.



SERVER SIDE INCLUDE (SSI)

It works by processing special SSI directives, such as:

```
<!--# include file="header.html" -->
```

which enable the inclusion of external files or subrequests. It is supported by Apache, LiteSpeed, Nginx and IIS servers

NGINX SSI

In Nginx, configuring Server Side Includes (SSI) can be more complex because the **`ngx_http_ssi_module`** is not compiled by default on some distributions and may need to be enabled during the build process.

Enable:

```
./configure --with-http_ssi_module
```

Config:

```
location / {
    ssi on;
}
```

BASIC USAGE

Include file:

```
<body>
    <!--# include file="header.html" -->
    <!--# include file="content.html" -->
    <!--# include file="footer.html" -->
</body>
```

Include request:

```
<body>
    <!--# include virtual="/_component/v1/header" -->
    <!--# include virtual="/_component/v1/content" -->
    <!--# include virtual="/_component/v1/footer" -->
</body>
```

DEMO TIME

TRY YOURSELF

You can run demo by pulling docker container

```
docker run -p 8888:80 crazygoat/nginx-ssi-example
```

And visit

```
http://127.0.0.1:8888/
```

ORIGINAL CODE

```
<header><!--?php include "_component/header.php" ?--></header>
<div class="container">
    <nav class="left-menu"><!--?php include "_component/left_menu.php" ?--></nav>
    <div class="content-area">
        <div class="breadcrumbs"><!--?php include "_component/breadcrumbs.php" ?--></div>
        <div class="content">
            <div class="content1"><!--?php include "_component/content.php" ?--></div>
            ...
        </div>
    </div>
</div>
<footer><!--?php include "_component/footer.php" ?--></footer>
```

SSI CODE

```
<header><!--# include virtual="/_component/header.php" --></header>
<div class="container">
    <nav class="left-menu"><!--# include virtual="/_component/left_menu.php" --></nav>
    <div class="content-area">
        <div class="breadcrumbs"><!--# include virtual="/_component/breadcrumbs.php" --></div>
        <div class="content">
            <div class="content1"><!--# include virtual="/_component/content.php" --></div>
            ...
        </div>
    </div>
</div>
<footer><!--# include virtual="/_component/footer.php" --></footer>
```

COMPONENT CODE

_component/breadcrumbs.php

```php
<?php usleep(100000); echo "Breadcrumbs";
```

NGINX CONFIG

```
server {
    ...
    ssi on;

    location ~ \.php$ {
        fastcgi_pass   php_fpm;
        fastcgi_param  SCRIPT_FILENAME /var/www/$fastcgi_script_name;
        include        fastcgi_params;
    }
}
```

NGINX CONFIG - CACHE

Nginx configuration:

```
fastcgi_cache_path /tmp/nginx-cache levels=1 keys_zone=articles_cache:10m max_size=1024m inactive=1h;

location ~/_component/*\.php$ {
    fastcgi_cache_key $scheme$host$uri$request_method;
    fastcgi_cache articles_cache;
}
```

Cache ttl is controlled by Cache-Control Header:

```
header("Cache-Control: public, max-age=10");
```

Important! only public cache is stored

IMPORTANT LIMITATIONS

- Page with layout must be really fast, with minimal logic
- All parameters to the component must be passed in the query string
- By default, headers from the main request are not passed
- Better prepare a looooot of PHP workers
- SSI requests can only be performed within the server*
- Using session can be problematic (session lock)

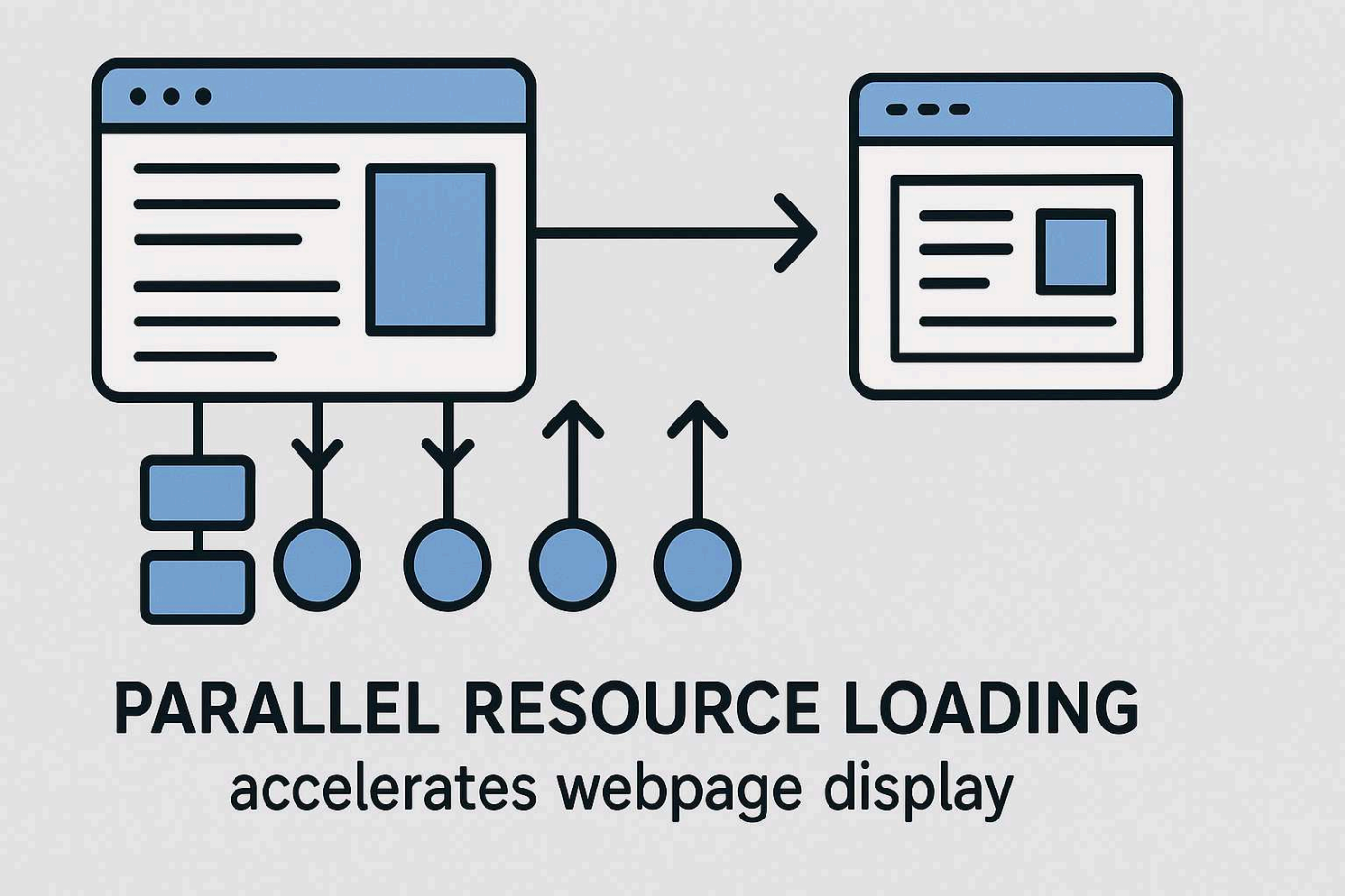
IMPORTANT LIMITATIONS - REMOTE REQUESTS

```
upstream components_backend {
    server app1.server.internal:80;
    server app2.server.internal:80;
    ...
}

server {
    location /_components/ {
        proxy_pass http://components_backend;
        proxy_set_header Host $host;
        ...
     }
}
```

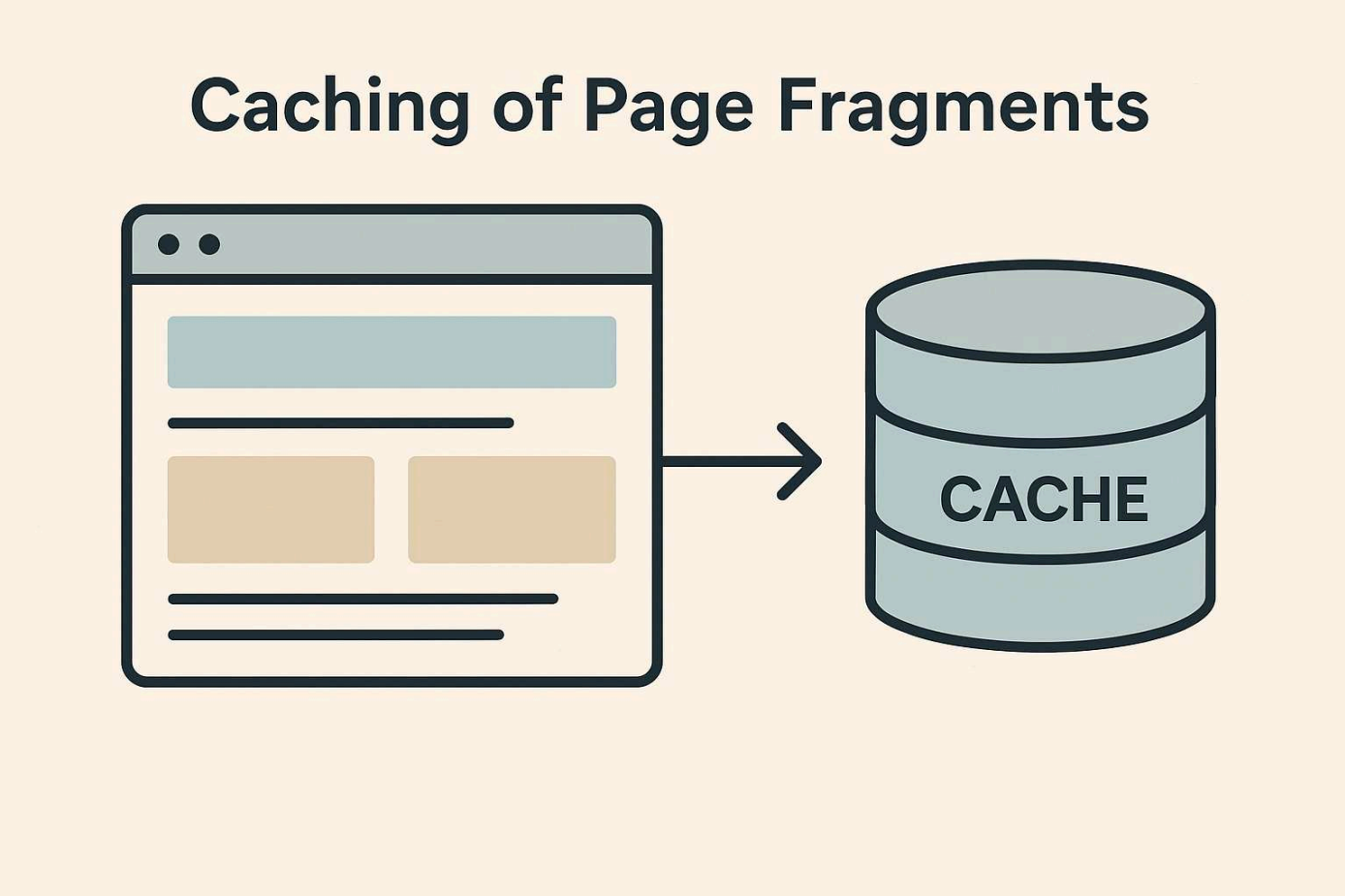
SERVER SIDE COMPOSITION ✅ PROS

Efficient Content Assembly – Components are fetched in parallel



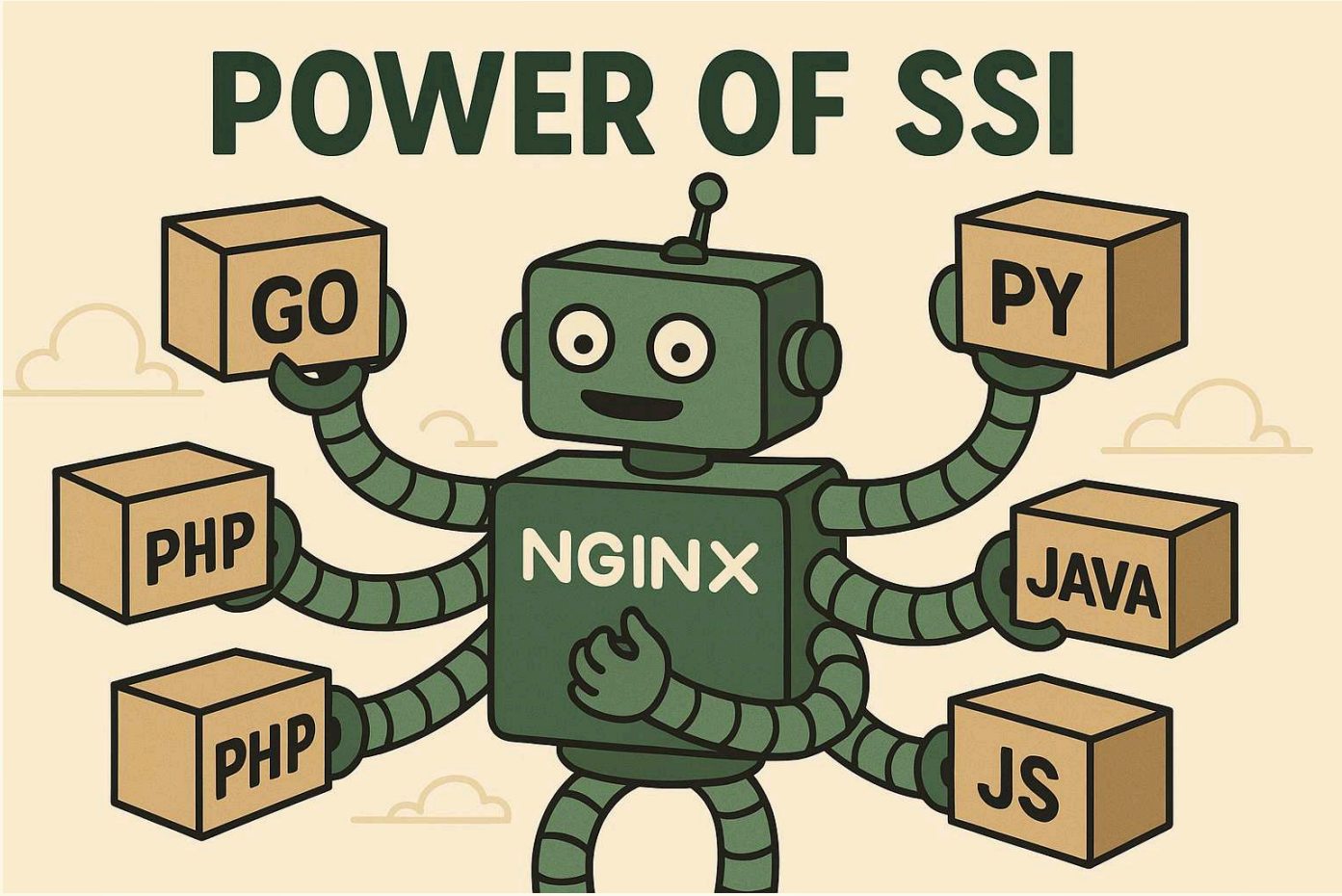
SERVER SIDE COMPOSITION ✅ PROS

Improved Performance – partial page caching, reducing backend load.



SERVER SIDE COMPOSITION ✅ PROS

Multi-Language Support – Components in different languages

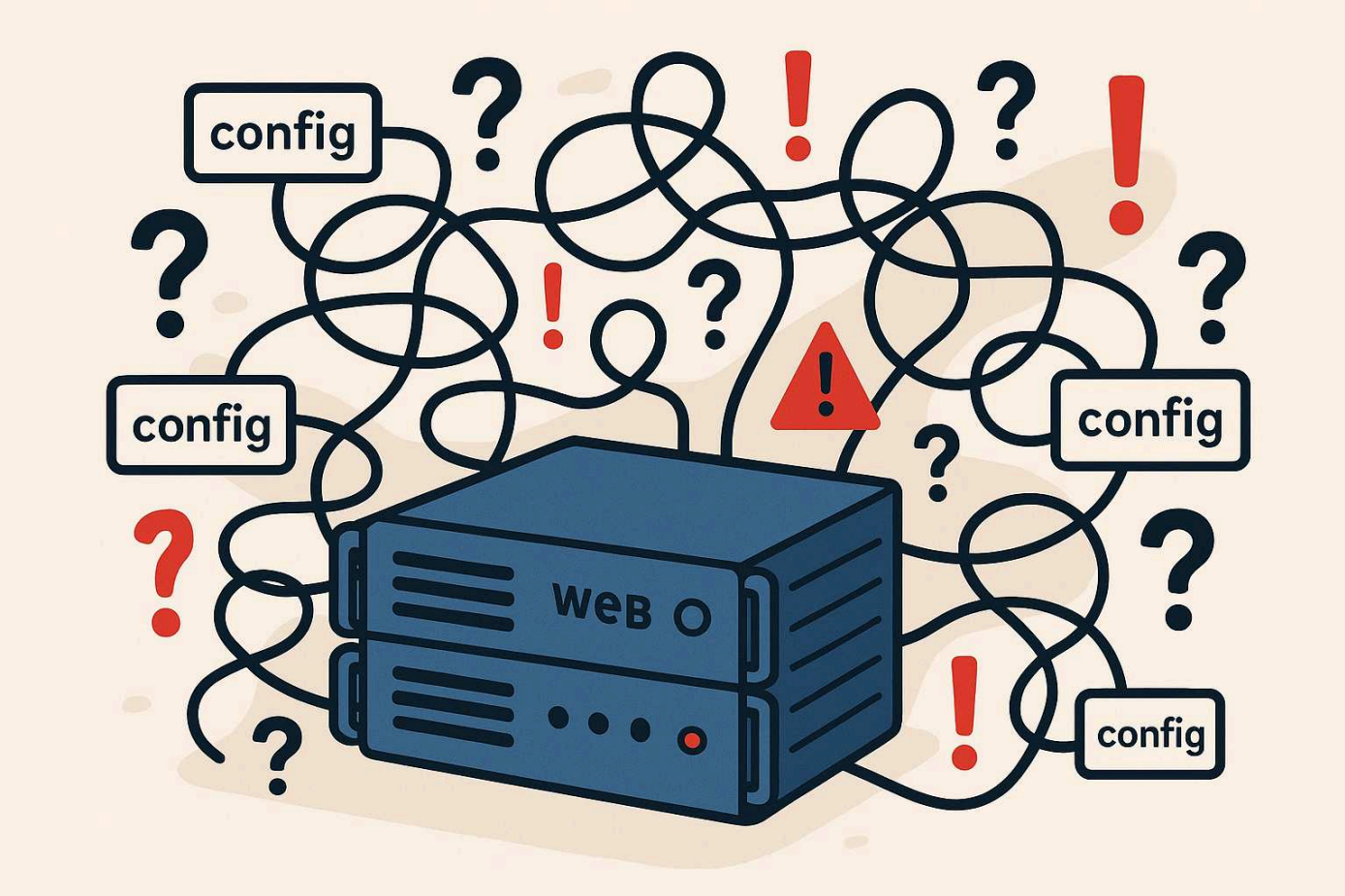


SERVER SIDE COMPOSITION ❌ CONS

Limited Flexibility – Requires server-side support

SERVER SIDE COMPOSITION ❌ CONS

Increased Complexity – not defined by the code.

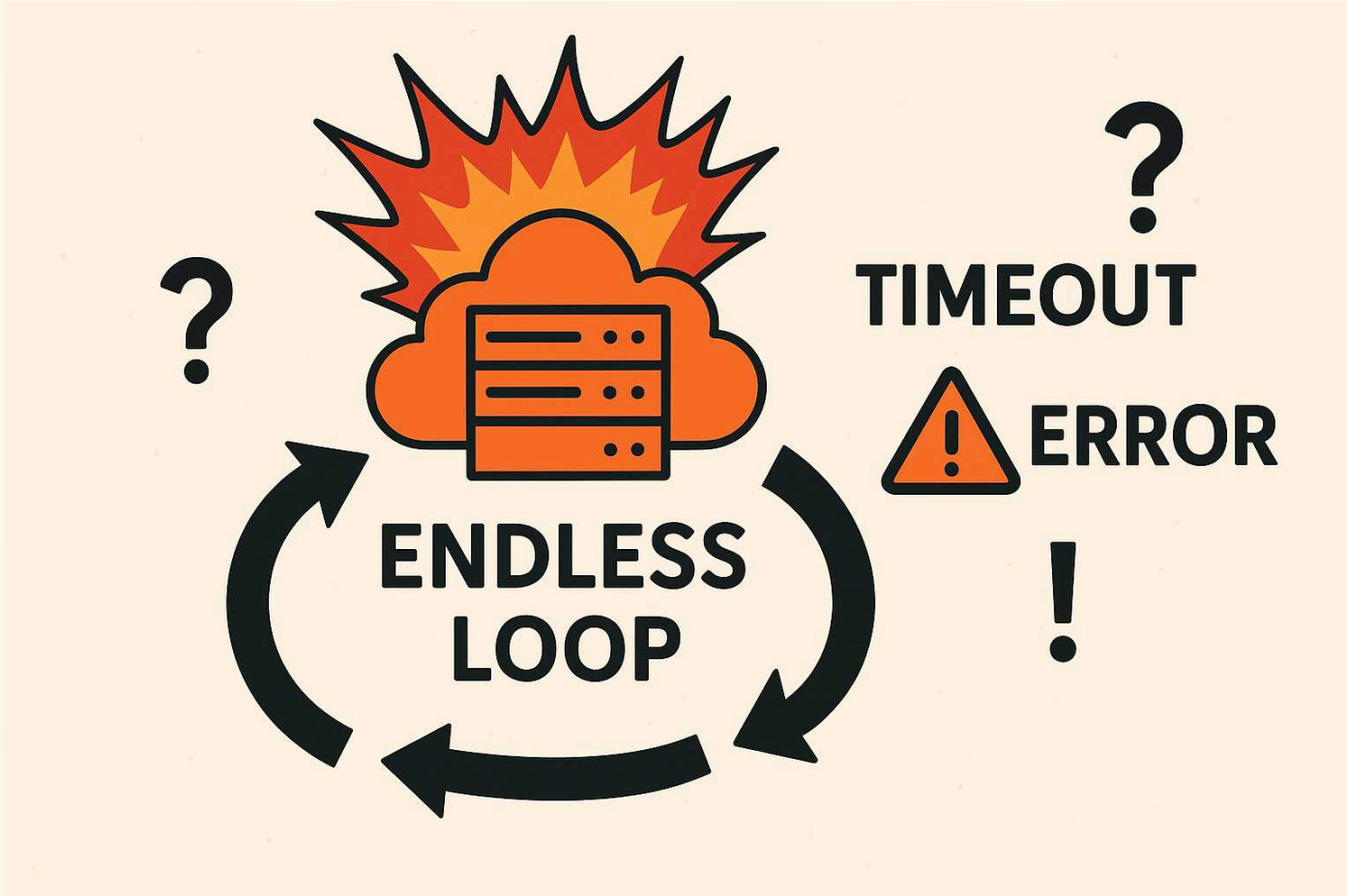


SERVER SIDE COMPOSITION ❌ CONS

Security Risks – exposing internal services.

SERVER SIDE COMPOSITION ❌ CONS

Other challenges – timeouts, ♾️ loops, error handling



TECHNOLOGY OVERVIEW - SERVER SIDE COMPOSITION

✅ Pros:

- **Efficient Content Assembly** – Components are fetched in parallel, speeding up page rendering.
- **Improved Performance** – Enables partial page caching, reducing backend load.
- **Multi-Language Support** – Components can be written in different programming languages and integrated seamlessly.

❌ Cons:

- **Limited Flexibility** – Requires server-side support and specific configurations.
- **Increased Complexity** – Managing dependencies between fragments can be challenging.
- **Security Risks** – Incorrect configuration may introduce vulnerabilities, such as exposing internal services or enabling injection attacks.
- **Other challenges** – timeouts, infinity loops, error handling.

WHAT CAN WE FIX

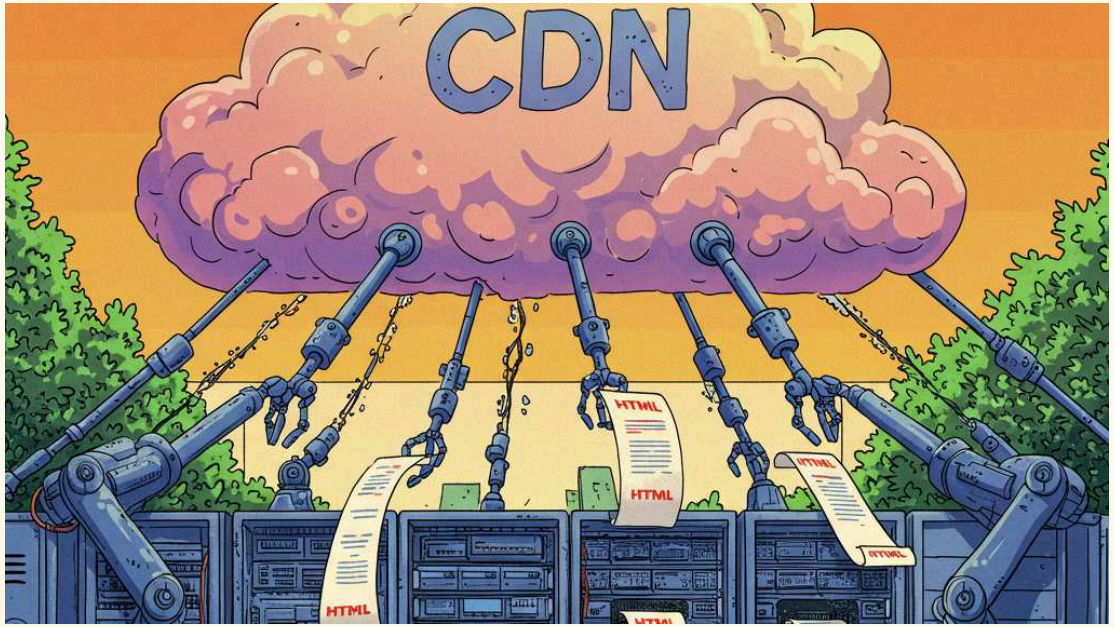
ERROR HANDLING

TIMEOUTS

EDGE SIDE INCLUDE (ESI)

EDGE SIDE INCLUDE (ESI)

Edge Side Includes (ESI) is a CDN feature that allows web pages to include content from other sources before being sent to the client.



EDGE SIDE INCLUDE (ESI)

It works by processing special ESI directives, such as:

```
<esi:include src="http://foo.bar/components/foo" />
```

which enable the inclusion of external subrequests.

ESI SUPPORT

CDNS

- **Akamai** - The original creator of ESI and provides full support.
- **Fastly** - Strong ESI support, Varnish-based.
- **CloudFlare*** - Offers ESI through their Workers platform.
- **AWS CloudFront*** - Limited ESI support through Lambda@Edge functions.

ESI SUPPORT

CACHE

- **Varnish Cache** - Robust ESI implementation including tags
- **Squid Cache** - Basic ESI functionality includes

ESI SUPPORT

PROXY & APPLICATION SERVERS

- **Traefik** - via go-esi plugin
- **Caddy** - via go-esi plugin
- **Roadrunner** - via go-esi plugin

THIRD RIDDLE

THIRD RIDDLE

Which of the following servers **does not** make subrequests in parallel?

1. **Fastly**
2. **Varnish**
3. **Traefik**

FASTLY

FASTLY

Sequential processing: ESI tags are processed sequentially, **not concurrently**, because when the parser encounters an ESI include tag, it will make the child request, wait for the HTTP response from the child request to complete...

VARNISH

GO-ESI

All server supported by go-esi plugin should work in parallel. Traefik, Caddy, roadrunner.

BASIC USAGE

Include request from different servers:

```
<body>
    <esi:include src="http://example.com/_component/v1/header" />
    <esi:include src="http://foo.com/bar/content" />
    <esi:include src="http://app.server.internal/_component/v1/footer" />
</body>
```

BASIC ERROR HANDLING

Simple fallback with alt attributes:

```
<esi:include src="http://example.com/_component/v2/header"
             alt="http://example.com/_component/v1/header" />
```

If `/v2/header` fails, then load `/v1/header`. Great for quick fallback.

ADVANCED FALLBACK

Advanced fallback with `<esi:try>` tag:

```
<esi:try>
  <esi:attempt>
    <esi:include src="http://example.com/_component/v2/content" />
  </esi:attempt>
  <esi:except>
    <div hx-get="http://example.com/_component/v1/content" hx-trigger="load" hx-swap="outerHTML">
      Loading content...
    </div>
  </esi:except>
</esi:try>
```

If `/v2/header` fails, then load `/v1/header` using htmx

DETECTING ESI SUPPORT

Most CDN providers add special HTTP headers to let you know ESI is available and active. Most popular is **`Surrogate-Capability`**, example:

```
Surrogate-Capability: varnish="ESI/1.0"
```

DETECTING ESI SUPPORT

Some html parsers can brake html structure

Original:

```
<div><esi:include src="http://foo.com/bar"/><p>some text</p></div>
```

Parsed:

```
<div><esi:include src="http://foo.com/bar"><p>some text</p></esi:include></div>
```

NOTIFYING CDN ABOUT ESI CONTENT

Parsing server responses for esi tags can be demanding. That's why most CDNs only parse responses that have the **`Edge-Control`** header set. Example:

```
Edge-control: dca=esi
```

SYMFONY SUPPORT

ESI is supported out of the box when using Symfony's HTTP Cache

```
# config/packages/framework.yaml
framework:
    esi: true
```

Then in twig template:

```
{# templates/static/layout.html.twig #}
{{ render_esi(controller('App\\Controller\\NewsController::latest', { 'page': 5 })) }}
```

DEMO TIME

TRAEFIKI CONFIGURATION

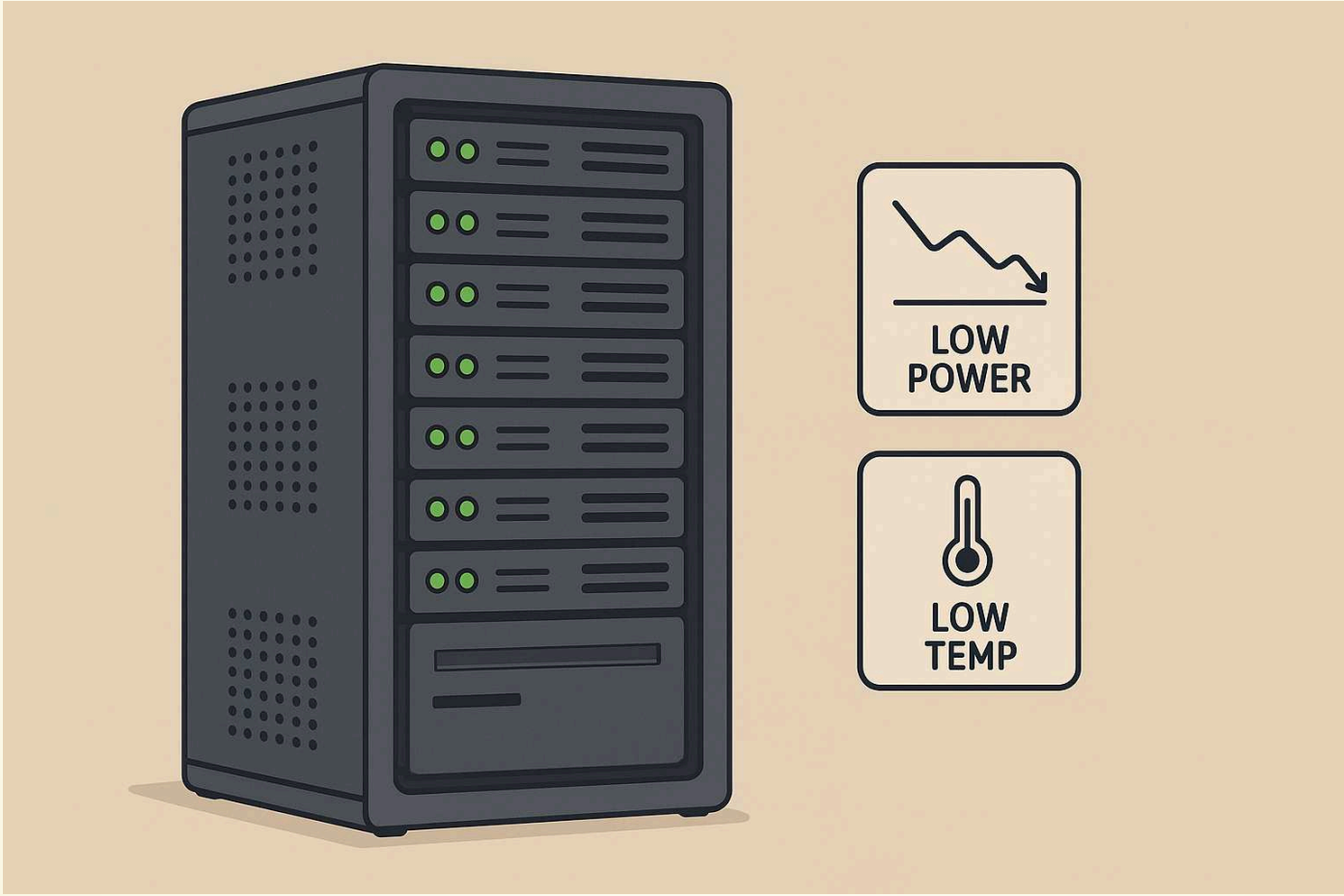
```
experimental:          #loading external plugin
  localPlugins:
    esi:
      moduleName: github.com/darkweak/go-esi
middlewares:           #configure esi plugin
  esi:
    plugin:
      esi: {}
http:
  routers:
    nginx:
      middlewares:
        - esi          #add esi middleware to service
      entrypoints:
      ...
```

DEMO TEMPLATE

```
<header><esi:include src="http://nginx/_component/header.php" /></header>
<div class="container">
    <nav class="left-menu"><esi:include src="http://nginx/_component/left_menu.php" /></nav>
    <div class="content-area">
        <div class="breadcrumbs"><esi:include src="http://nginx/_component/breadcrumbs.php" /></div>
        <h1>Use ESI to include components</h1>
        <div class="content">
            <div class="content1"><esi:include src="http://nginx/_component/content.php" /></div>
            ...
        </div>
    </div>
</div>
<footer><esi:include src="http://nginx/_component/footer.php" /></footer>
```

EDGE SIDE COMPOSITION ✅ PROS

Reduced Backend Load – Pages assembled at the CDN



EDGE SIDE COMPOSITION ✅ PROS

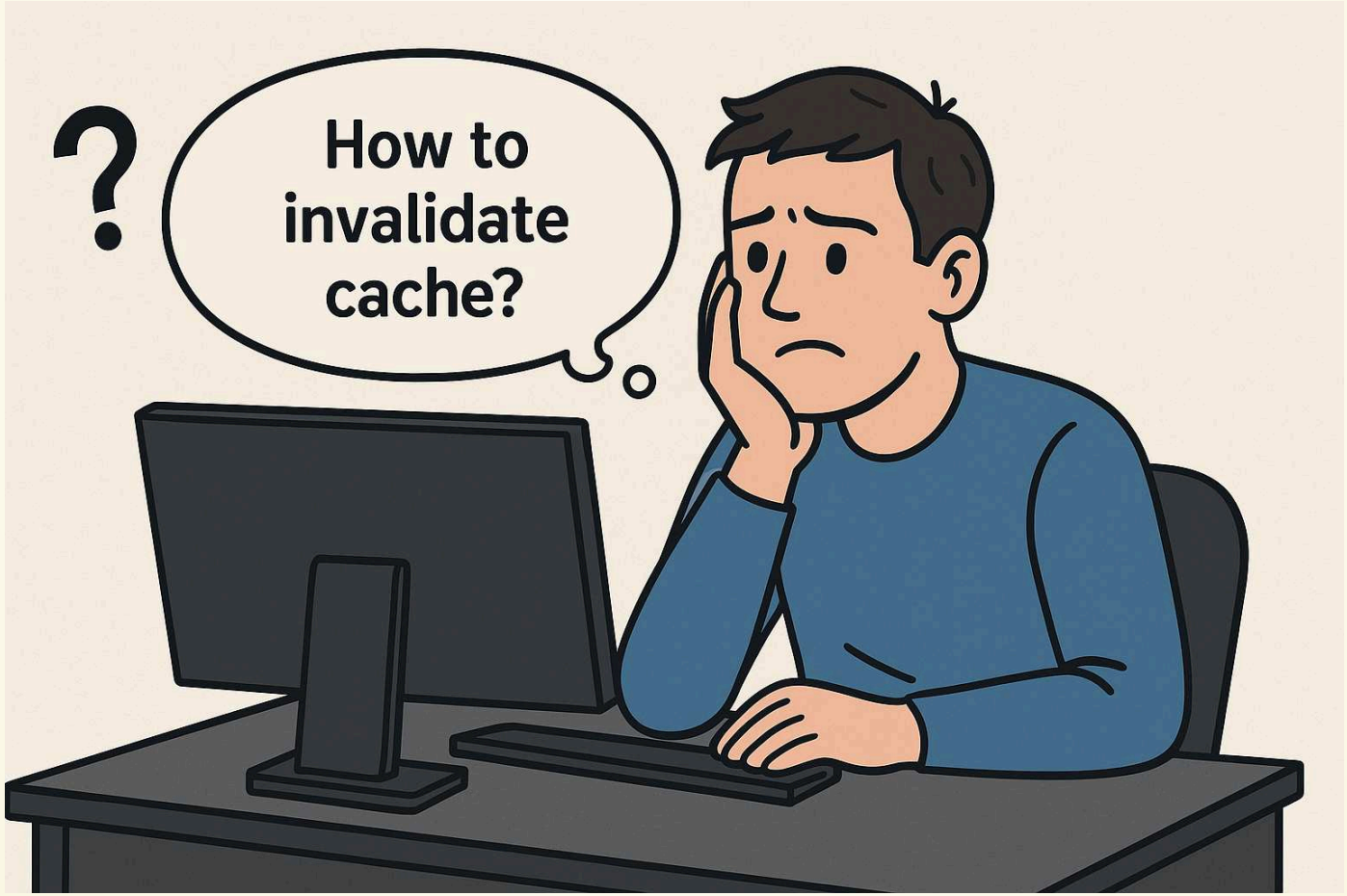
Faster Content Delivery – Components cache closer to the user

EDGE SIDE COMPOSITION ✅ PROS

Easy to Configure – Many CDNs offer built-in ESI

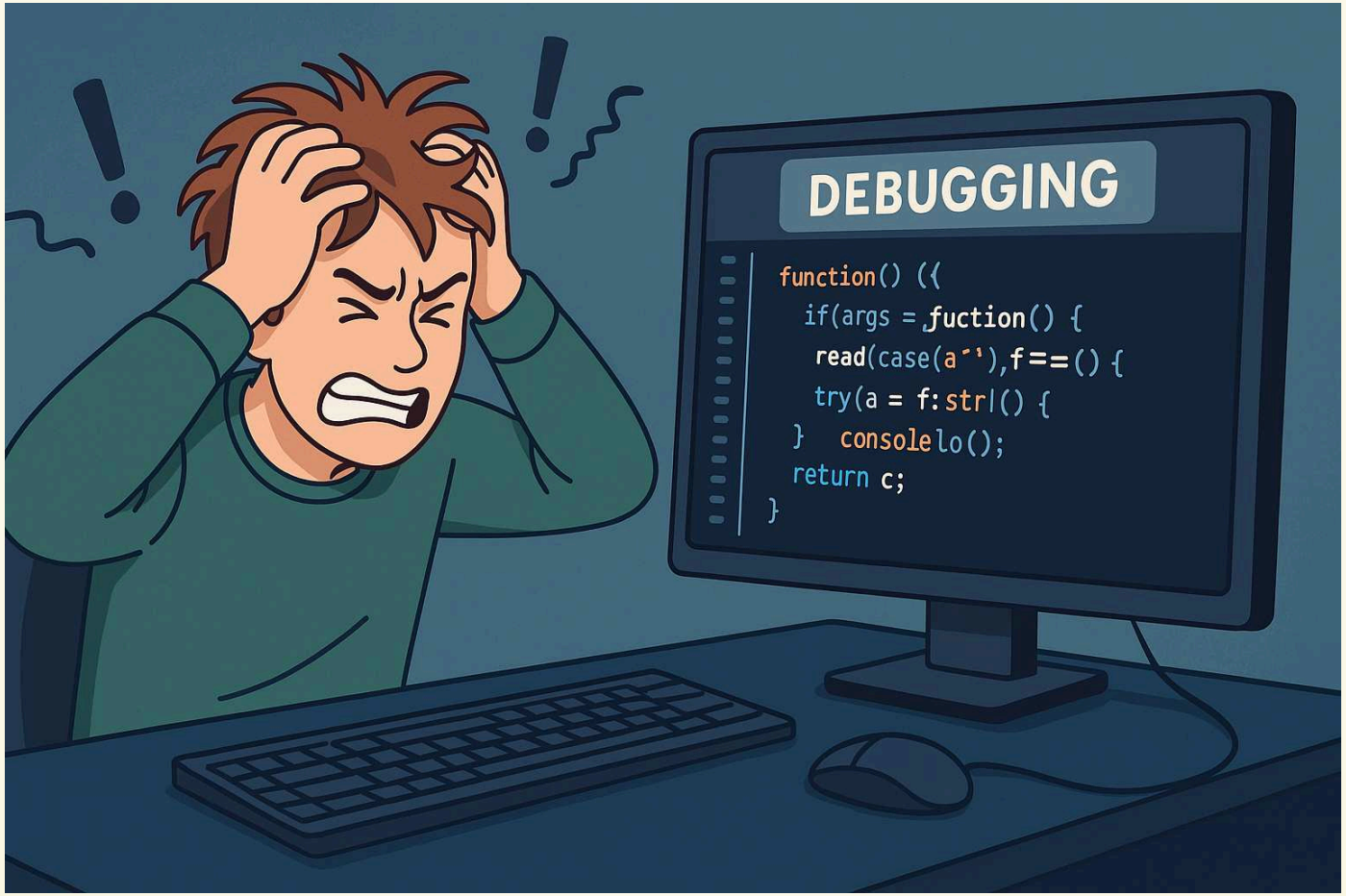
EDGE SIDE COMPOSITION ❌ CONS

Limited cache invalidation – hard to invalid single item



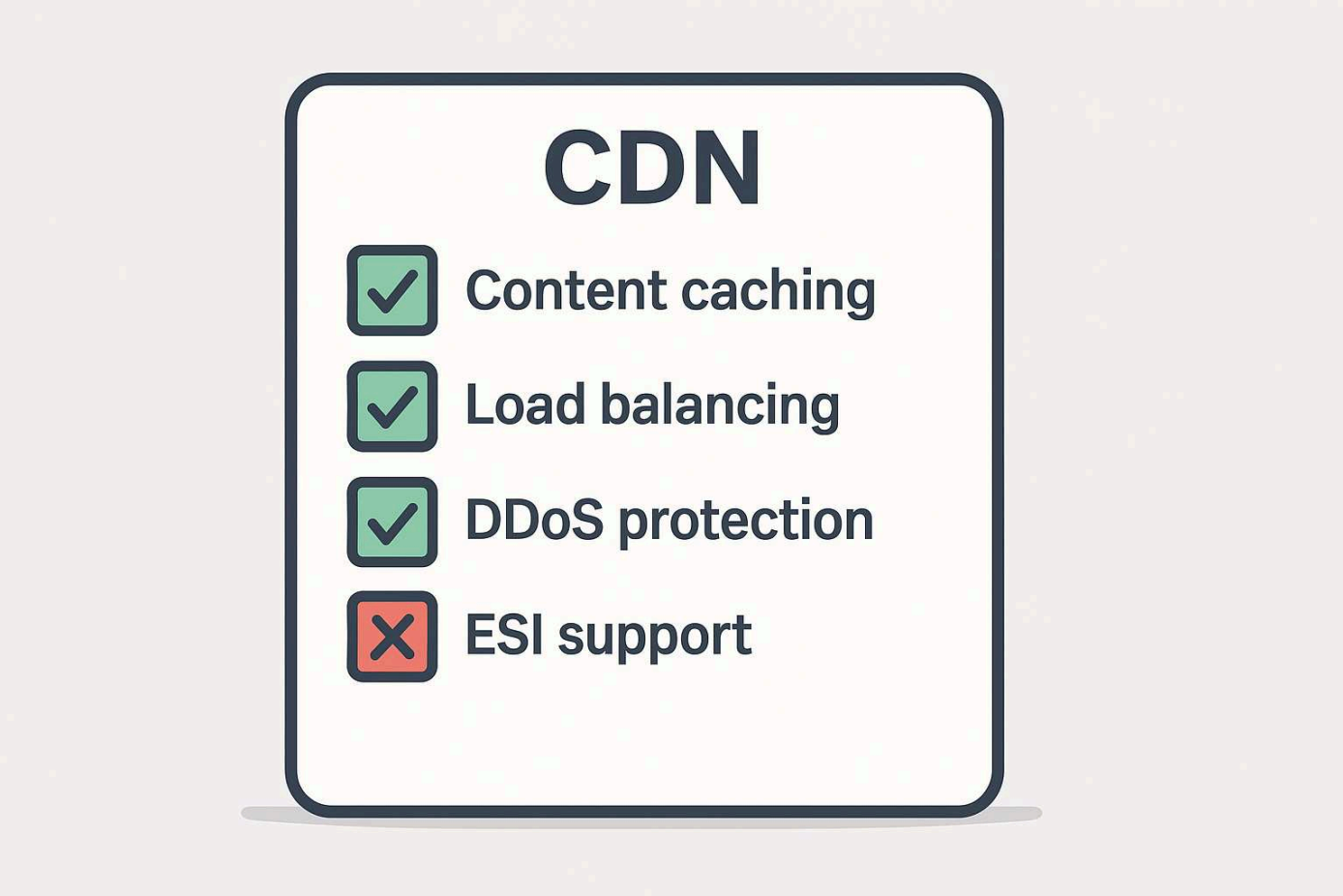
EDGE SIDE COMPOSITION ❌ CONS

Complex Debugging – harder to trace, processing done at the edge.



EDGE SIDE COMPOSITION ❌ CONS

CDN Dependency – Requires a CDN with ESI support.



EDGE SIDE COMPOSITION ❌ CONS

Additional Costs – ESI support can come with extra fees.

TECHNOLOGY OVERVIEW - EDGE SIDE COMPOSITION

✅ Pros:

- **Reduced Backend Load** – Pages are partially assembled at the CDN level, reducing server strain.
- **Faster Content Delivery** – Components can be cached closer to the user, improving response times.
- **Easy to Configure** – Many CDNs offer built-in ESI support with minimal setup required.

❌ Cons:

- **Limited cache invalidation** – Can be hard to invalid single cache item.
- **Complex Debugging** – Issues may be harder to trace due to processing happening at the edge.
- **CDN Dependency** – Requires a CDN with ESI support, limiting flexibility in infrastructure choices.
- **Additional Costs** – CDN services with ESI support can come with extra fees.

THE END

THE END

ALMOST

GO-MESI PROJECT

GO-MESI PROJECT

go-mESI sminimal Edge Side Includes) is a lightweight implementation of Edge Side Includes sESI) in Golang, designed to add ESI support to multiple web servers.

GO-MESI PROJECT

Features:

- **Parallel Fetching** - mESI supports parallel fetching of ESI fragments
- **Lightweight & Minimal** - focuses on essential ESI features
- **Multi-Server Support** - traefik, nginx, apache, caddy, roadrunner...
- **A/B testing** - test different version of component
- **Concurrent fetch** - if you need ultra performance
- **Timeout** - easily manage the page generation time
- **Fallback content** - content to be displayed on fail
- **Max depth** - Defines the maximum allowed recursion depth

GO-MESI - A/B TESTING

Allows to download with different probability from two different locations src and alt. In case of no alt attribute, the src location will always be downloaded.

```
<esi:include fetch-mode="ab" ab-ratio="90:10"
    src="http://foo.bar/A" alt="http://foo.bar/B" />
```

GO-MESI - MAXDEPTH

Defines the maximum allowed recursion depth for esi:include tags. This parameter prevents infinite loops that could occur when ESI templates reference each other.

```
<esi:include max-depth="2" src="http://foo.bar/recursive"/>
```

GO-MESI - TIMEOUT

Specifies the maximum time to wait for a server response when processing **`esi:include`** tags. The request will be terminated if this timeout is exceeded.

```
<esi:include timeout="0.2" src="http://foo.bar/some-long request"/>
```

GO-MESI - FALLBACK CONTENT

By default, the `esi:include` tag does not contain a body. mESI allows you to set the so-called fallback content, or the content that will be displayed in case the download of remote content fails

```
<esi:include timeout="0.25" src="https://foo.bar/can-be-slow">
  <div hx-trigger="load" hx-swap="outerHTML" hx-get="https://foo.bar/can-be-slow"></div>
</esi:include>
```

GO-MESI PROJECT

Gitlab: https://github.com/crazy-goat/go-mesi

THE END

THE END

QUESTIONS?

go-mESI

Slides