Efficiency Comparison — allegroit.dk

Old stack (live at `allegroit.dk`): WordPress + Elementor + Polylang on Simply.com shared hosting
New stack (live at `designtest.allegroit.dk`): Plain PHP + Parsedown on Hetzner VPS (Docker)
Measurements taken on the homepage on 2026-04-15 against the public production URLs.

TL;DR

Metric	WordPress site	New stack	Improvement
Time to first byte (TTFB)	325 ms	123 ms	**2.6× faster**
DOMContentLoaded	526 ms	448 ms	~15% faster
Full page load	716 ms	490 ms	**1.5× faster**
HTTP requests (first load)	100	14	**7.1× fewer**
HTML page weight (gzipped)	26.5 KB	7.9 KB	**3.4× smaller**
Files in the project	~12,500	125	**~100× fewer**
Database	MySQL required	none	–
DB queries per page	30–100+	0	–
Third-party plugins	8	0	–
Public attack endpoints	7+	0	–
Security headers grade	D	**A**	–

1. Server response time

Phase	WordPress	New stack
TTFB (server think time)	325 ms	123 ms
DOMContentLoaded	526 ms	448 ms
Full `load` event	716 ms	490 ms

The WordPress TTFB is what you'd expect from a shared-host PHP stack booting the full WP kernel, resolving 8 active plugins, and running 30–100 MySQL queries before it can emit a single byte. The new stack reads one Markdown file, renders it through Parsedown, and writes the response — no database round-trips, no plugin init, no filter hooks.

2. HTTP requests

Asset type	WordPress	New stack
HTML	1	1
Stylesheets	37	1
Scripts	45	0
Images	~15	~12
Fonts	~2	0
Total	**100**	**14**

Every request is a round-trip: TCP + TLS + bytes back. Cutting 100 to 14 means fewer round-trips, fewer chances for a slow asset to block rendering, and a much simpler network waterfall. The WP site loads 37 separate stylesheets because every plugin (Elementor, Polylang, contact form, cookie banner, …) ships its own CSS. The new stack has one hand-written `style.css`.

3. HTML page weight

Measurement	WordPress	New stack
Transferred (gzipped)	26.5 KB	7.9 KB
Decoded (raw HTML)	155 KB	30.8 KB

The WP HTML is full of inline Elementor markup (`data-settings`, nested wrapper divs, generated class names, auto-injected preload hints). The new stack emits hand-written templates — every element has a reason to be there.

4. Code footprint

Count	WordPress	New stack
Tracked files	~12,500 (estimate)	125
Active plugins	8	0
Themes	1 (+ child)	0
Database tables	~50 (wp_* core)	0
Admin UI	`/wp-admin/`	none

The WordPress count is an estimate based on a typical install: WP core (~3,000 files) + Elementor (~3,500) + Polylang (~500) + translation `.mo` files + theme files + plugin assets. The new stack is `git ls-files | wc -`

`l` → **125**. A fresh `git clone` produces a fully working site in seconds; WordPress needs a database import, a `wp-config`, file permissions, a plugin reactivation pass, and a cache warmup.

5. Database

Aspect	WordPress	New stack
Database engine	MySQL (required)	none
Tables	~50 (`wp_posts`, `wp_options`…)	0
Queries per page	30–100+ (depends on plugins)	0
Backup surface	SQL dump + uploads directory	`git`

No database means no backup job, no migration pain, no "the site broke because `wp_options` got bloated", no SQL injection attack surface, no "the host upgraded MySQL and something drifted". The backup strategy is literally `git push`.

6. Security / attack surface

We probed the 7 most common WordPress attack endpoints on both sites:

Endpoint	WordPress site	New stack
`/wp-admin/`	gated (Simply.com WAF)	**404**
`/wp-login.php`	gated	**404**
`/xmlrpc.php`	gated	**404**
`/wp-json/`	**200 OK** — REST index public	**404**
`/wp-json/wp/v2/users`	gated	**404**
`/?author=1`	gated	**404**
`/wp-content/`	gated	**404**

On the new stack there is literally nothing to attack — no admin URL, no login form, no REST API, no plugin auto-update endpoint, no PHP entry point other than a single hardened front controller. An attacker scanning for WordPress CVEs simply gets 404 everywhere and moves on.

HTTP security headers

Header	WordPress	New stack
`X-Content-Type-Options`	✓	✓
`Strict-Transport-Security`	✗	✓
`X-Frame-Options`	✗	✓
`Referrer-Policy`	✗	✓
`Permissions-Policy`	✗	✓
`Content-Security-Policy`	✗	✓
securityheaders.com grade	**D**	**A**

On the new stack, adding the full modern security-header set was a 10-line block in `docker/nginx.conf` — the site went from no headers to an A grade in a single commit. On the WP site, fixing this is a shared-host config change that requires Simply.com's cooperation and touches every site on the same host.

7. Operational model

Concern	WordPress	New stack
Deploy	FTP + DB import + plugin activation	`git push` → GitHub Actions → Docker pull
Rollback	Restore DB snapshot + files	`git reset --hard {prev} && docker compose up`
Backup	Daily SQL dump + uploads tarball	`git` (files) + Outlook (mail log)
Plugin updates	Monthly, manual, breaks things	n/a
Core upgrades	Every 3 months, risk of white screen	`apk upgrade` in base image on next build
Content edits	WP admin (WYSIWYG, risk of drift)	Markdown file + `git commit`
Multilingual	Polylang plugin	Two Markdown files per page, one slug map
Contact form	Contact Form 7 plugin	80 lines of PHP + `fsockopen` SMTP

8. What about WordPress's classic advantages?

The traditional list of WordPress advantages is three things: a WYSIWYG editor anyone can use, a massive plugin ecosystem, and 20 years of well-trodden terrain. In 2026, none of them weigh as much as they used to.

1. **"Anyone can edit in WP admin."** A truth with caveats — and there's a reason web agencies are the ones building and maintaining WordPress sites in the first place. In practice, WP admin still needs a technical person: Elementor and other page builders have their own learning curve, plugin updates break things on a regular basis, conflicts need resolving, backups need to run, and SEO and security plugins

need configuring. The "anyone can edit" promise is mostly marketing — most WP customers call their agency anyway when anything non-trivial needs to change.

And once you have a developer (or an agency) in the loop anyway, the argument flips. **Git is actually better than WordPress at handling multiple concurrent editors** — two people editing the same page in WP admin create conflicts and lost changes; git merges cleanly and leaves a revision history down to the line. Markdown is easy to learn for anyone mildly technical. And most importantly: nobody has to type the changes themselves. Describe what needs to change in plain English — *"update the price in the table, add a new reference, swap the image"* — and Claude Code handles it, commits it, and pushes. The WYSIWYG editor solves a problem that's no longer a problem.

2. **"There's always a plugin for that."** WordPress has a plugin for almost everything — and that's also what creates eight external codebases in your install, each with its own attack surface, update cycle and CVE tail. When you instead ask Claude to build a feature specifically for your site — booking, newsletter signup, filters, whatever — it comes out smaller, faster and more precise than a generic plugin that has to cover everyone's use cases. The question is no longer *"is there a plugin?"* but *"can it be built quickly?"* — and the answer is usually yes.

   A concrete example from a client WordPress site we worked on: the site was firing duplicate API requests — the same call multiple times per page load. We tried to debug it with AI assistance and **had to give up**. Not because the AI wasn't good enough, but because the code wasn't ours — it was a handful of plugins, each hooking into WordPress's filters without knowing about each other. You can't fix what you don't own. On a stack like this one, Claude Code has full access to a codebase we wrote every line of; that kind of debugging is trivial.

3. **"When WordPress breaks, you can Google the fix."** True — WordPress is 20 years old and there's plenty of help out there. But when *your* site breaks, you don't want to Google; you want it fixed. **That's exactly the kind of maintenance we do for clients.** If you're running a plain-PHP site built with Claude Code and it goes sideways one day, call us. 24-hour response time.

One honest caveat remains: the new homepage ships ~7.4 MB of images because the design uses large hero/screenshot imagery. This is a content choice, not an architectural limit — either site could be trimmed by switching hero images to WebP and serving responsive variants.

Summary

The new stack is a **single-purpose marketing site** built for the work it actually does. It's faster, smaller, cheaper to run, has no database, no admin panel, no plugins, and no WordPress attack surface.

And it was built end-to-end — from empty repo to live, bilingual, GDPR-compliant, A-grade-security production site — in approximately 15 hours of work with Claude Code. That covers all template code, the content migration from WordPress, the contact form and SMTP integration, the deploy pipeline, the security-header block, and the favicon + OG-image set. The number is falsifiable — the git log is public.

The classic "WordPress is for non-technical editors" argument dissolves once an AI coding tool is in the loop — Claude Code handles edits, features, and debugging against a codebase you own every line of. That last part matters: full ownership of the code is what makes AI-assisted maintenance actually work.