Faux-Press Hardening Strategy

A full plan calibrated to what's actually in the codebase. Items are ordered by **regression-elimination ROI** — each item's purpose is to either prevent a class of bug or make a class of bug instantly localizable.

Conventions:

- **VERIFIED** — claim grounded in cited file:line during this session
- **EXTRAPOLATED** — plausible but not directly verified; treat as hypothesis until validated by the locality benchmark
- **NEW** — invention not in either source; mine

Part I — Foundations

These are prerequisites. Everything else depends on them landing first.

1. Locality benchmark — do this before anything else

Why first: Every claim about "regression spread" is unfounded until measured. Without this, every subsequent decision is dressed-up intuition.

Mechanism:

- Add `scripts/bench-locality.sh` next to `bench-render.sh`.
- Render baseline → mutate corpus (e.g., insert one word at paragraph N) → re-render → diff at the geometry level.
- Metric: median + p95 number of pages whose `PlacementNodeInstance` rect set changes.
- Three corpus tiers: `small_prose-20`, `large_book-1000`, `usability-report.md` (the one that hung).
- Report as part of `bench-render.sh` output.

Acceptance criteria:

- Baseline numbers checked in (not pass/fail yet, just the table).
- Once 3+ commits have data, set a regression alarm: "p95 pages affected jumped from N to N+M, why?"

Effort: 1-2 days. **Owner:** anyone. **Blocks:** items 14, 15.

2. StructuralParentForest newtype — close the bug class permanently

Why: The 8-min hang we just patched is one instance. The same anti-pattern (cross-cutting writers to a shared map) likely lives elsewhere in `MeasurementGraphIndex`.

Mechanism (verified code, ready to drop in):

```
Rust
pub struct StructuralParentForest {
    parent_of: BTreeMap<OccupantId, ParentId>,
}

impl StructuralParentForest {
    pub fn try_from_edges(edges: impl IntoIterator<Item =
(ParentId, OccupantId)>)
```

```
        → Result<Self, ForestError>;
}

pub enum ForestError {
    SelfLoop { node: OccupantId },
    MultipleParents { node, old, new },
    Cycle { path: Vec<OccupantId> },
}
```

File: `crates/layout-measurement-lowerer/src/lib.rs:2206`. Replace the raw map. Update `from_graph` (line 2271) to collect edges into a `Vec<(ParentId, OccupantId)>`, then call `try_from_edges` once. Cycle audit becomes free (constructor enforces it).

Validate-once exits:

- Remove the read-side `HashSet` cycle guard (no longer needed).
- Remove the `audit_structural_parent_acyclic` runtime walk (no longer needed).
- Both are kept-for-history but moved to a `cfg(test)` consistency check.

Audit other index maps in the same file (`component_publication_subjects`, `media_resource_by_publication_subject`, list-body-occupant maps). For each: is the implicit invariant a forest, a bag, or a function? Promote to a typed wrapper if it has any structure.

Effort: 2-3 days. **Owner:** layout-measurement-lowerer. **Blocks:** none — independent. **Replaces:** the recent defensive patch with a stronger guarantee.

3. Stage fingerprint snapshots in –trace mode

Why: Today, “something changed” is opaque. With per-stage fingerprints, regressions become “Stage 17’s fingerprint changed” — instant localization.

Mechanism:

- Define `StageSnapshot { stage_name, ordinal, fingerprint_128, node_counts_by_kind, page_count, severe_diags, ip_summary }`.
- After each `renderer-pipeline` stage, serialize the *graph slice that stage owns* canonically (sort by stable IngestId), feed to BLAKE3, store the 128-bit hash + counts.
- `--trace` writes a sidecar `<output>.stages.jsonl`.
- A new CI gate: `scripts/check_stage_fingerprints.sh` — fails if a PR changes a fingerprint that the diff didn’t touch (i.e., changing stage 13 must not perturb stage 7).

Implementation note: the canonical serialization is the work. Use a fixed schema crate (`stage-snapshot/`); each stage emits the slice via a trait method.

Effort: 1 week. **Owner:** renderer-pipeline. **Depends on:** item 14 (stage trait) accelerates this. **Blocks:** items 6, 7, 17.

4. Convergence-result enum for silently-failing passes

Why: Smell #6 (exclusion reflow). Same pattern probably exists in font fallback chain, table column re-solve, KP relaxation. Silent fallback = invisible corruption.

Mechanism:

```rust
pub enum ConvergenceResult<T> {
    Converged(T),
    DegradedAfter { result: T, passes: usize, residual: f64 },
    Diverged { last_attempt: T, reason: ConvergenceReason },
}
```

Every pass that retries with a max-pass cap returns this. The shell's diagnostic collector treats `Diverged` as `Severity::Error` and `DegradedAfter` as `Severity::Warning`.

Audit targets:

- `reflow_text_compositions_with_page_exclusions` (renderer-pipeline:1555)
- `refine_table_layouts_from_shape` (renderer-pipeline:1551) — verify if it can fail
- KP relaxation ladder in `linebreak-core/src/solver.rs`
- Font fallback walk in `font-resolver`

Effort: 1 week. **Owner:** each pass owner. **Blocks:** none.

5. Geometry-extraction visual diff replaces pixel diff

Why: Smell — current `pdf_visual_audit.py` rasterizes and compares pixels. Pixel diffs are noisy. The IR carries the geometry losslessly.

Mechanism: Build `scripts/structural-diff.py` that:

1. Reads the EmitIR snapshot (item 17 makes this an artifact).

2. Extracts geometry per the table from the deep-research report (page rect, composition break IDs, fragment IDs+rects, glyph-run bboxes, table cell matrix, image rects, paint op order, accessibility role tree).
3. Diffs structural geometry first; falls back to pixel diff only when the structural pass says “same shape but bytes differ.”

Output: “Page 8 has 1 different fragment boundary and 3 text bboxes shifted >0.5pt” — not “screenshot is red again.”

Effort: 2 weeks. **Owner:** scripts/. **Depends on:** item 17. **Replaces:** pixel-level scrutiny in `pdf_visual_audit.py`.

Part II — Contracts

These promote implicit conventions to enforced types.

6. Typed slot-class distinction in component-realizer

Why: Smell #3's *upstream* fix. Today `slots_for_publication` (component-realizer/src/lib.rs:2167) seeds the primary slot with the component's own publication identity. When that primary slot's name happens to also be a structural-parent slot ("title", "items", "item_body", "media"), measurement-lowerer mis-infers a self-loop.

Mechanism: Two distinct slot-class types.

```Rust
pub enum SlotClass {
    PrimaryContent,           // self-seeded; describes "this component's own content"
    StructuralChildren,       // contains nested publications; never self-seeded
    Relationship(RelKind),    // describes/labels/etc.
}

pub struct ComponentSlotInstance {
    pub class: SlotClass,
    pub name: SlotName,
    pub occupants: Vec<IdentityRef>,
}
```

`structural_parent_slot()` in measurement-lowerer becomes `slot.class == StructuralChildren` — typed instead of name-string-matched.

Effort: 3-5 days. **Owner:** component-realizer + layout-measurement-lowerer. **Replaces:** the write-side self-loop skip (item 2's defense becomes obsolete and can be removed). **Verifies via:** item 3 (no fingerprint change in stages downstream of measurement).

7. Rename crates to match their layers

Why: Smell #13. `.arch/layers.toml` documents that `layout-measurement-lowerer = "text-traits"` because it does both. Names that lie are documentation rot.

Mechanism:

- `layout-measurement-lowerer` → split into `layout-measurement` (geometry)

    - `text-traits` (run prep). Or: keep one crate, name it `layout-text-traits`.
- `text-shaping-lowerer` → `text-composition` (per layers.toml).
- `composition-fragmenter` → consider `pagination-fragments` since the work is pagination-specific.

Caveat: crate renames are big diffs. Defer unless the names are actively confusing newcomers (which they currently are). Pair with item 8.

Effort: 1 week. **Owner:** anyone with merge access. **Blocks:** none. **Risk:** historic git-blame becomes harder; mitigate with `git log --follow`.

8. Split handlers.rs (29,811 lines) by command

Why: Smell #11. Stage 4 (generated matter), Stage 5 (entry), Stage 33 (PDF metadata), Stage 34 (krilla wrapper) all share a file with 50+ other things.

Mechanism:

- `crates/faux-press/src/cmd/render.rs` (~3-5k lines)
- `crates/faux-press/src/cmd/bench.rs`
- `crates/faux-press/src/cmd/inspect.rs`
- `crates/faux-press/src/cmd/report_render.rs`
- `crates/faux-press/src/cmd/mdbook.rs`
- `crates/faux-press/src/handlers.rs` becomes a thin re-export.

Effort: 2-3 days mechanical. **Owner:** anyone. **Blocks:** items 9, 11.

9. Split renderer-pipeline/src/lib.rs (5,866 lines) by stage

Why: Smell #12. One file holds all wiring. Per-stage submodules let each stage have its own input/output type as the only crate-level boundary.

Mechanism:

- `crates/renderer-pipeline/src/stage/mod.rs` — defines `Stage` trait
- `stage/source_to_publication.rs`
- `stage/components.rs`
- `stage/layout.rs`
- `stage/measurement.rs`
- `stage/shaping.rs` (sub-modules: `segment`, `shape`, `table_refine`, `exclusion_reflow`)

- `stage/composition.rs`
- `stage/fragmentation.rs`
- `stage/placement.rs`
- `stage/paint.rs`
- `stage/emit_ir.rs`
- `lib.rs` becomes a wiring file (~500 lines).

Effort: 1 week. **Owner:** renderer-pipeline. **Depends on:** item 14 (Stage trait first).

10. Delete dead –engine flag

Why: Smell #1. Parsed, ignored, leaks ambiguity.

Mechanism: Remove from clap; add to release notes. If someone passes `--engine legacy`, error with: "the legacy engine was removed in Phase 5; identity is the only engine."

Effort: 30 minutes. **Owner:** anyone.

11. Generated-matter explicit two-pass contract

Why: Smell #2 + the deep-research report's strongest contribution. Stage 4 today *mutates the markdown string* and re-enters Stage 5. There's no contract on what survives between passes.

Mechanism:

```rust
pub struct DraftFacts {
    pub page_map: BTreeMap<IdentityRef, PageNo>,
    pub object_targets: BTreeMap<String, TargetLoc>,
    pub bookmarks: BookmarkTree,
    pub cross_refs: BTreeMap<RefId, ResolvedRef>,
}

pub fn render_draft(md: &str, cx: &RenderContext)
    -> Result<DraftFacts>;

pub fn materialise_generated_matter(md: &str, facts: &DraftFacts)
    -> String;

pub fn render_final(materialised_md: &str, cx: &RenderContext,
facts: Option<&DraftFacts>)
    -> Result<EmitBundle>;
```

`DraftFacts` becomes a snapshot artifact — golden test fodder. The contract is now: "render_final may use facts but must not depend on `render_draft` side effects."

Effort: 3-5 days. **Owner:** faux-press (handlers.rs:17448). **Depends on:** item 8.

12. Mandatory RenderContext, mandatory shape cache

Why: Smell #4 + verified: `Option<&PersistentShapeCache>` is the API today; the cache is opt-in, the fast path is not the default.

Mechanism:

```Rust
pub struct RenderContext {
    pub fonts: Arc<InMemoryFontInventory>,
    pub images: Arc<InMemoryImageInventory>,
    pub media: Arc<InMemoryMediaResourceRegistry>,
    pub shape_cache: Arc<PersistentShapeCache>,    // not Option
    pub unicode: UnicodeServices,
    pub diagnostics: DiagnosticsSink,
}
```

Shape-cache disabling becomes an explicit knob (`--no-cache` flag) instead of "you forgot to wire it." Every signature in `renderer-pipeline` that takes `Option<&Cache>` takes `&RenderContext` instead.

Effort: 2 weeks (high diff churn). **Owner:** renderer-pipeline + handlers.rs. **Depends on:** item 9.

13. Split text-shaping stats into 3 buckets

Why: Smell #5 — Stages 15, 16, 17 share `TextShapingStats`. Stats come back as one number; impossible to tell which sub-pass is hot.

Mechanism: Three explicitly named stats structs + three explicit stage names in trace output:

- `ShapeStats` (Stage 15)
- `TableRefineStats` (Stage 16)
- `ExclusionReflowStats` (Stage 17, now returns `ConvergenceResult`)

Effort: 1 day. **Owner:** text-shaping-lowerer. **Depends on:** items 4, 9.

Part III — Architecture

14. Stage trait — formalize what every stage promises

Why: Today every stage is a free-function. Adding fingerprints, snapshots, owned-kinds enforcement requires a trait so they can be applied uniformly.

Mechanism:

```rust
pub trait Stage<Input, Output> {
    fn name(&self) → &'static str;
    fn ordinal(&self) → u16;

    /// Kinds this stage may add or mutate.
    /// Asserted via fingerprint diff outside the owned set.
    fn owned_kinds(&self) → &'static [IrKind];

    fn run(&self, input: Arc<Input>, cx: &RenderContext)
        → Result<StageResult<Output>>;
}

pub struct StageResult<O> {
    pub output: Arc<O>,
    pub fingerprint: [u8; 16],
    pub stats: StageStats,
    pub diagnostics: Vec<Diagnostic>,
}
```

A debug-mode wrapper diffs the input vs output graph and asserts “no nodes outside `owned_kinds()` were modified.” This is **automatic enforcement** of the layered architecture, not just `arch gate` linting.

Effort: 1 week. **Owner:** renderer-pipeline. **Blocks:** items 3, 5, 9.

15. Run segmentation pre-pass

Why: Two reasons:

- **Correctness:** HarfBuzz’s manual is explicit — it does *not* do bidi, line breaking, hyphenation. Today text-shaping-lowerer pretends it does, which works for Latin and breaks subtly for Thai/Khmer/Arabic/emoji-modifier sequences.
- **Caching:** Runs become canonically keyed on `(font, script, lang, direction, features, axes)`, dramatically improving cache hit rate for repeated content.

Mechanism: Insert `Stage 14.5` between Layout-Measurement (Stage 13) and Font-Resolution (Stage 14):

```rust
pub fn segment_text_runs(graph: &mut RendererIrGraph, cx:
&RenderContext)
    → SegmentationStats;

pub struct SegmentedRun {
    pub key: TextRunKey,
    pub text: SmolStr,
    pub break_opportunities: Vec<UAX14BreakClass>,
}

pub struct TextRunKey {
    font_intent: FontIntent,
    script: Script,           // from unicode-script
    lang: Lang,
    dir: Direction,           // from unicode-bidi
    features: SmallVec<[FeatureTag; 8]>,
    axes: SmallVec<[VariationAxis; 4]>,
}
```

Dependencies: `unicode-script`, `unicode-bidi`, `xi-unicode` (UAX-14). All are dependency-stable Rust crates.

Effort: 3-4 weeks. **Owner:** new `text-segmentation` crate. **High value for non-Latin scripts.**

Defer this if the user's corpus is Latin-only. Validate via item 1 (locality benchmark) on Thai/Arabic/Hebrew test cases.

16. Stable total-order merge key for chapter-parallel

Why: Today's "sort by IngestId" works (lossless gate passes), but it's a single-axis order. Future parallelism beyond chapters (per-section, per-page) needs a richer key.

Mechanism:

```rust
#[derive(Clone, Eq, Ord, PartialEq, PartialOrd)]
pub struct StableOrdKey {
    pub stage: u16,
    pub chapter: u32,
    pub page: u32,
    pub ingest: u64,
```

```
    pub identity: u128,
    pub seq: u32,
}
```

Every merge boundary uses this key. Future parallelism granularities (page-parallel, section-parallel) just add fields.

Effort: 3 days. **Owner:** chapter_parallel.rs. **Verifies via:** lossless gate stays green + item 1 (locality bench unchanged).

Caveat: don't do this until the use case is real. Today single-axis is enough.

17. EmitBundle replaces parallel BackendOutputNode + EmitIR

Why: Smell #9. `backend-output-lowerer` and `graph-emit-ir` both summarize Paint IR; the manifest from the former is rarely consumed.

Mechanism:

```
Rust
pub struct EmitBundle {
    pub emit_ir: EmitIR,                    // the canonical op list
    pub manifest: BackendManifest,           // counts, losses, unsupported features
    pub bookmarks: BookmarkTree,
    pub stage_snapshots: Vec<StageSnapshot>, // for diff/debug
}
```

Fold `BackendOutputNodeInstance` into `BackendManifest` inside `EmitBundle`. The graph-emit-ir crate becomes the sole producer of "summary of backend-relevant output." `backend-output-lowerer` either disappears or becomes a sub-module of `graph-emit-ir`.

Effort: 1 week. **Owner:** graph-emit-ir + backend-output-lowerer. **Blocks:** item 5.

18. Collapse duplicated accessibility emission

Why: Smell #8. Stages 23 and 26 both emit `AccessibilityNodeInstance`. Two emission points = two opportunities to disagree.

Mechanism:

- Stage 23 produces `AccessibilityPlan` (typed, not a NodeInstance — internal-only).
- Stage 26 reads `AccessibilityPlan` + paint identities and produces the *single* `AccessibilityNodeInstance` set.
- One emit point. The plan is intermediate.

Effort: 3-5 days. **Owner:** renderer-pipeline + renderer-metadata-lowerer.

Part IV — Solvers

19. Windowed page planning — gate behind locality benchmark

Why: Smell #14. The composition planner's serial cursor is the architectural bottleneck for parallelism, *if* it's a bottleneck.

Mechanism:

- Reify the implicit "current page state" from `plan_composition` into an explicit `BoundaryState { page_num, body_fill, open_floats, footnote_carry, widow_orphan_context }`.
- Solver becomes: pick a window of `k=5` .. `10` pages, solve in isolation, commit first page (or first spread), slide forward.
- Windows can run in parallel; merge by `StableOrdKey` (item 16).

Decision criterion: Only do this if item 1 (locality benchmark) shows that today's serial cursor produces high-ripple changes. If a one-paragraph edit perturbs <3 pages today, skip — the cursor isn't the problem.

Effort: 4-6 weeks. **Owner:** composition-planner. **Risk:** high — globally-aware aesthetics may degrade. Run side-by-side with current planner under a feature flag for 4 weeks before flipping default.

20. CP-SAT for local hard zones — narrow scope

Why: Tables-with-many-rows + repeated headers + height constraints, exclusion-heavy pages, footnote packing. These are integer-grid problems already (`RectPtMilli`).

Mechanism:

- Add `or-tools-rs` or wrap CP-SAT C++ via FFI.
- Define `LocalLayoutProblem` enum with variants: `FloatPlacement`, `TableContinuation`, `ExclusionPackingPage`, `FootnotePacking`.
- Heuristic planner runs first; if defect threshold exceeded → CP-SAT runs with `AddHint(heuristic_result)` warm-start.
- Time-bounded (e.g. 100ms per problem); falls back to heuristic on timeout.

Hard scope rules:

- Never used for whole-book pagination.
- Never used outside the pre-defined enum variants.
- Build-time wobble tolerance: <500ms added to CI render of `large_book-1000`.

Effort: 3-4 weeks per problem variant. **Owner:** new `pagination-cpsat` crate.

21. Skip Monge/SMAWK paragraph breaking until proven applicable

Why: The current cost stack — hyphen + class + widow + orphan + float — almost certainly doesn't satisfy total monotonicity. Implementing SMAWK without a proof gives "very fast wrong answers."

Mechanism (only if pursued):

1. Empirical study: for the actual corpus, sample 1000 paragraphs, evaluate cost matrix, check `M[i,j] + M[i+1,j+1] ≤ M[i,j+1] + M[i+1,j]` for all (i,j,k,l).
2. Report monotonicity rate. If <80%, skip.
3. If ≥80%, implement guarded solver: matrix-monotonicity check at runtime, SMAWK on monotone, exact KP fallback.

Don't implement until step 1 is done. The study is ~3 days; SMAWK is 2+ weeks.

Skip entirely unless the locality benchmark or KP profiling shows paragraph breaking is a hot path. Project memory says text-shaping (not KP) was the recent perf bottleneck.

22. Skip vision-in-the-loop verifier

Why: PaperFit's value is "compile correctness from external rendered output." Faux-press has full IR introspection — overlap, clipping, gutter violations, spread asymmetry are all checkable from `PlacementNodeInstance` directly. Item 5 (structural visual diff) covers this.

Don't pursue until: structural visual diff is in CI for 3+ months, and a real defect has been missed by it that would have been caught by raster.

Part V — Testing

23. Stage-diff CI gate

Why: The whole point of items 3, 5, 14 — make the regression localizable.

Mechanism:

- Per-PR CI:
  - Render canonical corpus (`small_prose-20`, `usability-report.md`, `large_-book-1000`).
  - Compare stage fingerprints vs. `main`.
  - Fail if fingerprints differ in stages outside the diff's blast radius.
  - Pass with annotation if the changed stage's fingerprint changed *and* the PR touched that stage.
- `--bless` mode for accepting expected fingerprint changes.

Effort: 1 week (after item 3). **Blocks:** none.

24. Golden frontier tests

Why: The planner emits candidate cost vectors *before* the fragmenter commits. Test the candidates, not just the output.

Mechanism:

- For corpus paragraphs: snapshot the KP frontier (legal breakpoints + cost vector + winning sequence).
- For corpus pages: snapshot `PageCandidate` / `SpreadCandidate` lists with cost vectors.
- For corpus emit: snapshot `EmitIR` op count + ordered stable keys.

A solver "regression" is a fingerprint-stable change in the frontier — detected even if the final PDF looks identical.

Effort: 1 week. **Owner:** new `crates/golden-frontier-tests/`. **Depends on:** item 3.

25. cargo-fuzz harnesses

Why: Coverage-guided fuzzing finds the edge cases unit tests don't. Targets are the boundary-crossing functions:

- Markdown ingest → graph

- Component-realizer slot population
- Measurement index builder (specifically `from_graph`)
- Exclusion reflow convergence
- Chapter partition + merge
- Graph → EmitIR lowering
- PDF emit with weird-but-valid EmitIR

Mechanism:

- `cargo install cargo-fuzz`.
- `crates/fuzz/` with one fuzz_target per boundary.
- Properties to assert (panics, infinite loops, broken invariants):
  - No self-loop in StructuralParentForest (auto-enforced by item 2)
  - Stable total order under parallel merge
  - Deterministic output for same input + seed
  - No negative widths/heights
  - Page numbers monotonic
  - Every accessibility node maps back to a valid identity
  - Every emitted glyph run references a loaded font

Effort: 1 week initial setup + ongoing corpus growth. **Owner:** new `crates/fuzz/`.

26. Property tests via Proptest

Why: Properties that hold for *any* input — fuzz finds these slowly; proptest finds them with shrinking.

Mechanism: Per-crate `prop` test modules. Examples:

- `linebreak-core`: KP terminates for any item sequence.
- `composition-planner`: any candidate cost vector is non-negative.
- `fragment-placer`: every fragment's rect is within page bounds.
- `chapter_parallel`: serial render and parallel render produce byte-identical PDFs (already a script; promote to proptest with random chapter splits).

Effort: 1-2 days per crate. **Owner:** crate owners.

27. Fuzz seed corpus discipline

Why: Good seed corpus = effective fuzzing. Bad seed corpus = wasted CPU.

Mechanism: Maintain `crates/fuzz/corpus/` with:

- Generated-matter doc with `@toc` + crossrefs
- Mixed LTR/RTL paragraph
- Thai or Khmer text
- Table with repeated headers and narrow labels
- Sidebar/pull-quote exclusion page
- Emoji modifier sequences
- Chapter boundary with widow/orphan pressure
- Missing-glyph stress case
- Pathological inputs that have caused historical bugs (the usability-report.md added)

Every CI failure that leads to a fix → corpus addition.

Effort: ongoing.

28. CI regime — layered, not monolithic

Per-PR (5-10 min budget):

- Unit + property tests
- `arch gate` (already runs)
- Stage fingerprint diff on small corpus (item 23)
- 3× determinism rerun (item 16)
- Lint: stable ordering checks (CI fails if `HashMap` :: `iter` is used in production code without sort)

Nightly (60 min budget):

- Full corpus render
- Structural visual diff (item 5)
- Fuzz corpus replay
- 15-30 min guided fuzzing
- `bench-render.sh` + `bench-locality.sh` (item 1)

Pre-release (manual trigger):

- Vision verifier — *if* item 22 ever lands
- Backend parity (PDF vs Canvas vs EPUB byte equivalence on safe-subset corpus)
- Accessibility audit (PDF/UA-1 conformance)

Effort: 1 week to set up gating thresholds. **Depends on:** items 3, 5, 23, 25.

Part VI — Process

29. Stage ownership document

Why: Today, no one owns "shaping" — it's a function. With 36 stages, ownership matters.

Mechanism: Extend `docs/design/pipeline-walkthrough.md` per stage with:

- Owner (a name)
- Performance budget (e.g., "Stage 13 must complete in <2s on `large_book-1000`")
- Explicit failure modes
- Invariants

Owner = first reviewer required for any change to that stage's crate.

Effort: 2 days. **Owner:** the team collectively.

30. No-silent-fixes commit rule

Why: A bug fix without a regression test guarantees the bug returns.

Mechanism: Add to CLAUDE.md:

> Every bug fix must include:
> - The fix itself
> - One regression test (unit or proptest)
> - One invariant assertion or typed encoding (if applicable)
> - One stage-fingerprint snapshot update (if behavior changed)
>
> A bug fix without a test is not a fix; it's a future regression with prep work.

Pre-commit hook can lint: "this commit modifies a `*-lowerer` crate but adds no test files."

Effort: 30 min. **Verifies:** every recurring bug stops recurring.

31. Documentation rot guard

Why: Both the architecture doc and pipeline-walkthrough cite file:line. Files move; line numbers drift; docs lie.

Mechanism:

- `scripts/check-doc-citations.py` — parses all `crate-name/path/file.rs:LINE` references in `docs/` and verifies each line exists and the function name still matches (use rustc-style `#[doc]` markers).
- CI gate on PRs that touch `crates/`.
- `docs/design/pipeline-walkthrough.md` carries a "Last verified against commit X" stamp; CI compares to current SHA and warns if stale.

Effort: 3 days. **Verifies:** docs stay tied to code.

32. Performance budget per stage

Why: Project memory shows real wins (text-shaping 14.8s → 1.83s, markdown-parse 26.4s → 51ms, pipeline 29s → 12s on `large_book-1000`). These need to be defended.

Mechanism: Per-stage budget table in pipeline-walkthrough:

Stage	Budget (large_book-1000)	Current	Headroom
7 (markdown ingest)	100ms	51ms	2×
13 (measurement)	500ms	?	?
15 (shaping)	2s	1.83s	<10%
17 (exclusion reflow)	200ms	?	?
19 (composition)	3s	?	?
34 (PDF emit)	6s	5.7s	<5%

CI fails if any budget is exceeded by 25%.

Effort: 1 week to populate, then ongoing. **Owner:** stage owners (item 29).

33. Diagnostics promotion ladder

Why: Today diagnostics are best-effort warnings. Some are critical (cycle detection, missing glyph in production output, accessibility violations).

Mechanism: Severity escalation via `--strict-diagnostics`:

- Default: `Severity::Warning` doesn't gate; `Severity::Error` does.
- `--strict-diagnostics`: all warnings escalate to errors.
- CI uses `--strict-diagnostics` on `print-final` mode.
- Production fast path (`--mode preview-fast`) keeps warnings warnings.

Categories that *always* error:

- Structural-parent forest cycle (post-item 2, this becomes a constructor error, not even a runtime check)
- Glyph render-misses with no fallback registered
- PDF/UA-1 tag tree malformed
- Convergence Diverged (from item 4)

Effort: 3 days. **Depends on:** item 4.

Part VII — Performance (defer; only if budgets are violated)

34. Continued IR-graph optimization patterns

Project memory documents a pattern: `graph.node(&id)` is O(nodes); calling it in a per-edge loop is O(N^2). Fixed in 4 places already. Pattern to maintain:

Lint-able rule: New code may not call `graph.node(&id)` inside `for edge in graph.edges` loops without first building a `node_by_id: HashMap<IdentityRef, &Node>` index.

Effort: 1-day clippy lint + ongoing review. **Owner:** anyone.

35. Krilla rayon feature already on; track upstream

Project memory: 1.3s saved on book-1000 from one feature flag. Periodically check for new krilla performance flags (next major release).

Effort: ongoing. **Owner:** emit-pdf.

36. Defer 1000-page-in-seconds goal

Why: Already at 5.7s on `large_book-1000`. Diminishing returns.

Skip until a real customer hits a bigger book (10k pages?) and current performance fails their budget.

Sequencing — what to actually do

Sprint 1 (2-3 weeks): Prove the problem before solving it

1. Locality benchmark (item 1)
2. Stage trait (item 14)
3. `StructuralParentForest` newtype (item 2)
4. Convergence-result enum (item 4)
5. Delete `--engine` (item 10)

Sprint 2 (3-4 weeks): Make regressions instantly localizable

6. Stage fingerprint snapshots (item 3)
7. Stage-diff CI gate (item 23)
8. Geometry-extraction visual diff (item 5)
9. Split text-shaping stats (item 13)
10. No-silent-fixes commit rule (item 30)
11. Stage ownership doc (item 29)

Sprint 3 (3-4 weeks): Type the implicit invariants

12. Slot-class typed distinction (item 6)
13. Generated-matter explicit contract (item 11)
14. EmitBundle (item 17)
15. Collapse accessibility emission (item 18)

Sprint 4 (2-3 weeks): Mandatory context

16. Split handlers.rs (item 8)
17. Split renderer-pipeline/lib.rs (item 9)
18. Mandatory RenderContext (item 12)

Sprint 5 (3-4 weeks): Testing infrastructure

19. cargo-fuzz harnesses (item 25)

20. Property tests (item 26)
21. Golden frontier tests (item 24)
22. Layered CI regime (item 28)
23. Doc citation guard (item 31)

Sprint 6+ (deferred — pull only if metrics demand)

- Run segmentation pre-pass (item 15) — **only if multilingual corpus**
- Windowed page planner (item 19) — **only if locality benchmark shows ripple**
- CP-SAT local solver (item 20) — **only if specific defect class persists**
- Stable total-order key extension (item 16) — **only if multi-axis parallelism added**
- Crate renames (item 7) — **only if onboarding pain reported**

Skip

- Monge/SMAWK (item 21)
- Vision-in-the-loop (item 22)
- Pure-stage structural copies (would 6× memory)
- Single unified KP cost function (loses precision)

Success criteria

After sprints 1-5 (~4 months), the system meets:

Criterion	Today	After
`sha256(input + version) → sha256(output)` deterministic	Yes (verified)	Yes (CI-enforced)
Stage-localized regressions	No	Yes (item 23)
Implicit invariants in maps	Yes (smell #3)	No (items 2, 6)
Silent fallbacks	Yes (smell #6)	No (items 4, 33)
Re-entrant generated matter	Yes (smell #2)	No (item 11)
handlers.rs >25k lines	Yes	No (item 8)
renderer-pipeline/lib.rs >5k lines	Yes	No (item 9)
Locality measured	No	Yes (item 1)
Stage ownership documented	No	Yes (item 29)
Property + fuzz testing	Minimal	Full (items 25, 26)
Per-stage performance budgets	Implicit	Explicit (item 32)
Doc citations stale-checked	No	Yes (item 31)

The thesis isn't "make faux-press SOTA." It's: **make every regression instantly localizable, every invariant typed, every fallback loud.** SOTA is downstream of that — once you've eliminated the noise, the genuinely-novel algorithmic problems become visible and tractable. Trying to leap to SOTA without doing this work first is how teams produce faster regressions.

If you want to start, the safe first commits are items 1 + 2 + 14 in parallel — different files, no conflicts, each independently valuable.

Last verified against commit `26652a6f` on `main` (2026-05-28).