

undefect.

CWE-407 · THE SEDIMENTARY DEFECT
QUADRATIC COMPLEXITY IN GRAPH TRAVERSAL INFRASTRUCTURE

2026-03-24 · Internal draft — not for external distribution

CWE-407: The Sedimentary Defect

A Technical White Paper on Quadratic Complexity in Graph Traversal Infrastructure

Internal draft — not for external distribution until coordinated disclosure is complete

Date: 2026-03-24

Abstract

A single structural error — a list used where a set belongs, inside a graph traversal loop — is present in 35 confirmed sites across 19 software ecosystems. Every affected system maintains a `visited` or `onStack` set to track nodes during graph traversal. In every defective site, that set is implemented as a list. Membership is tested by linear scan. The result is $O(n^2)$ or worse behavior in code that should run in $O(n)$.

The defect is not exotic. It activates on every compilation of a large Java program, every TypeScript type-check of a large codebase, every `pip install` of a project with a deep dependency graph, every OSPF topology change on a network with hundreds of nodes. It has persisted for decades because the code is correct — a list and a set both answer the membership question — and because it degrades at the scale where most developers never work. The fix is a one-line data structure substitution. We have patched, tested, and benchmarked every confirmed site in the compiler and build-tool wave.

30 sites patched. 5 deferred (PostgreSQL, pending `nodeHash()` infrastructure). 1 fixable-upstream (Erlang OTP — slab patch to `digraph.eri`). 2 not-worth-fixing (debug-only, trivially bounded input).

1. The Defect

1.1 Formal Description

CWE-407: Inefficient Algorithmic Complexity. The affected code maintains a `visited` or `onStack` collection during graph traversal. The collection should provide $O(1)$ membership testing; it is implemented as a list providing $O(n)$ membership testing. Because this check is performed once per graph edge — inside the inner loop of Tarjan SCC, Dijkstra's SPF, or a DFS cycle detector — the overall algorithm degrades from $O(V+E)$ to $O(V^2+VE)$.

At $V=1,000$ nodes: 1,000,000 operations instead of 1,000. A **1,000× overhead**, silent, correct in output, invisible without deliberate benchmarking.

1.2 Why It Persists

This class of defect fossilizes because of four compounding factors:

Correctness. A list and a set both answer the membership question correctly. Tests pass. No crash, no wrong answer. The defect is purely one of cost, and cost is not checked by assertion.

Era of origin. The affected code was written in the 1990s and 2000s when `ArrayList`, `list`, or `std::vector` was the default container and hash sets were an explicit opt-in. The idiom was the right idiom for its era. It calcified as the language ecosystems matured around it.

Propagation by copy-paste. The same algorithm, the same variable names, and the same data structure choice appear across GHC, GCC, Erlang, Maven, and Python's `pip` — written by different teams, in different languages, in different decades. Each team copied from the same algorithm literature and made the same choice independently. The defect is sedimentary: deposited in layers, each layer pressing down on the last.

Degradation at scale. Most graphs encountered in practice are small. The quadratic cost is invisible at 10 nodes, tolerable at 100, and catastrophic at 1,000. Developers working on typical inputs never see the problem. Developers working at scale attribute the slowness to “large project overhead” or “complex type inference” — accurate descriptions that obscure the underlying cause.

1.3 The Fix

For every confirmed site, the fix is structural: replace the list-backed visited collection with a hash set (O(1) amortized membership) or a parallel boolean flag on the node itself (O(1) exact membership). The behavioral contract is identical. SCC membership, cycle detection, topological ordering — all produce the same output. Only the cost changes.

The canonical javac fix illustrates the pattern:

```
// Before - O(V²): stack.contains(n) is O(|stack|)
List<Node> stack = new ArrayList<>();
if (!stack.contains(n)) { stack.add(n); }

// After - O(V): onStack is a HashSet, lookup is O(1)
Deque<Node> stack = new ArrayDeque<>();
Set<Node> onStack = new HashSet<>();
if (!onStack.contains(n)) { stack.push(n); onStack.add(n); }
```

2. The Defect Map

CRITICAL — O(n³)

ID	Tool	Location	Status
scala3-0001	Scala 3 compiler	OrderingConstraint.scala:248 — List[TypeParamRef].contains in nested constraint lattice	PATCHED

Scala 3’s type inference solves a constraint lattice over type parameters. The constraint membership check is nested inside a loop that is itself nested inside the type inference solver. The result is cubic complexity: O(C³) where C is the number of type parameters under constraint. For heavily generic Scala 3 code — DeFi smart contracts, Cats Effect stacks, Spark schemas — this is the dominant build cost.

HIGH — Hot path, every compilation or planning pass

ID	Tool	Location	Status
javac-0001	OpenJDK javac	<code>GraphUtils.java:186</code> — Tarjan <code>stack.contains(n)</code>	PATCHED
javac-0002a	OpenJDK javac	<code>Infer.java:1850</code> — <code>ArrayList.findNode</code> linear scan	PATCHED
javac-0002b	OpenJDK javac	<code>Infer.java:1747</code> — uncached closure DFS	PATCHED
javac-0004	OpenJDK javac	<code>Dependencies.java:197</code> — <code>List.contains+add</code>	PATCHED
javac-0005	OpenJDK javac	<code>InferenceContext.java:506</code> — <code>List.containsAll()</code>	PATCHED
ts-0001	TypeScript	<code>checker.ts:11503</code> — <code>resolutionTargets[]</code> linear scan	PATCHED
ts-0002	TypeScript	<code>checker.ts:5256</code> — <code>visitedSymbols</code> array	PATCHED
ts-0003	TypeScript	<code>checker.ts:5763</code> — <code>visitedSymbolTables</code> array	PATCHED
ghc-0001	GHC	<code>Directed/Internal.hs:78</code> — <code>v `elem`</code> SCC decode	PATCHED
ghc-0002	GHC	<code>Inductive/Graph.hs:489</code> — <code>elem</code> × 4 codegen	PATCHED
ghc-0003	GHC	<code>Graph/Ops.hs:637</code> — <code>elem color neighbourColors</code> register allocator	PATCHED
kotlin-0001	Kotlin compiler	<code>NonExpansiveInheritanceRestrictionChecker.kt:150</code> — <code>in List</code> post-DFS	PATCHED
llvm-0001	LLVM	<code>GlobalsModRef.cpp:570</code> — <code>is_contained(vector<CGN*>)</code> LTO	PATCHED
rustc-0001	rustc	<code>inhabited_predicate.rs:109,127</code> — <code>SmallVec::contains</code>	PATCHED
erlang-0001	Erlang OTP	<code>digraph.erl:578</code> — <code>lists:member(V, Xs)</code> in <code>one_path/8</code>	PATCHED
frouting-0001	FRRouting	<code>ospf_ti_lfa.c:72,114,227,278,285</code> — <code>listnode_lookup</code> × 5	PATCHED
frouting-0002	FRRouting	<code>ospf_spf.c:275</code> — <code>listnode_lookup(parent->children, v)</code> in Dijkstra main loop	Unpatched
postgresql-0001	PostgreSQL	<code>tlist.c:812</code> — <code>tlist_member</code> in sort/group labeling	DEFERRED
postgresql-0002	PostgreSQL	<code>preptlist.c:180,206,316</code> — <code>tlist_member</code> × 3 in MERGE/UPDATE	DEFERRED
postgresql-0003	PostgreSQL	<code>equivclass.c:1041</code> — <code>list_member</code> equiv class matching	DEFERRED
postgresql-0004	PostgreSQL	<code>analyzejoins.c:1914</code> — <code>list_member</code> join elimination	DEFERRED

MEDIUM — Real defect, bounded or cold path

ID	Tool	Location	Status
javac-0003	OpenJDK javac	<code>ModuleHashesBuilder</code> — <code>Deque.contains()</code>	PATCHED
ghc-0004	GHC	<code>Tc/TyCl/Utils.hs:973</code> — <code>elem</code> constructor list	PATCHED
gcc-0001	GCC	<code>gcov.cc:980</code> — <code>find(vector.begin, end, w)</code> Johnson's	PATCHED
rustc-0002	rustc	<code>specialization_graph.rs:69</code> — <code>Vec::position</code>	PATCHED
cpython-0001	CPython	<code>sccutils.py:73</code> — <code>node in path</code> list	PATCHED
distlib-0001	distlib / pip	<code>util.py:1180,1204</code> — <code>successor in stack</code> Tarjan	PATCHED
cargo-0001	Cargo	<code>ops/tree/mod.rs:343</code> — <code>Vec::contains</code> (display only)	PATCHED
gyp-0001	GYP	<code>input.py:1604</code> — <code>child in path</code> list + <code>.index()</code>	PATCHED
npm-0002	npm arborist	<code>can-place-dep.js:370</code> — <code>peerPath.includes()</code>	PATCHED
linux-0001	Linux kernel	<code>headerdep.pl:153</code> — <code>grep {} @\$top</code> cycle detect	PATCHED
sqlite-0001	SQLite	<code>trigger.c:792</code> — <code>sqlite3IdListIndex</code> in <code>checkColumnOverlap</code>	PATCHED
composer-0001	Composer	<code>RepositoryUtils.php:46</code> — <code>in_array</code> in <code>filterRequiredPackages</code>	PATCHED
composer-0002	Composer	<code>InstalledRepository.php:128-180</code> — <code>in_array</code> × 4 in <code>getDependents</code>	PATCHED
postgresql-0005	PostgreSQL	<code>list.c:1077-1478</code> — <code>list_union</code> , <code>list_intersect</code> , <code>list_difference</code>	DEFERRED
erlang-0002	Erlang OTP	<code>digraph_utils.erl:495</code> — <code>lists:member</code> in <code>is_reflexive_vertex</code>	FIXABLE-UPSTREAM
create-0001	Create mod	<code>TrackGraph.findDisconnectedGraphs</code> — <code>ArrayList.remove(0)</code> O(n) shift in BFS frontier	Unpatched

LOW — Principle violations, bounded input

ID	Tool	Location	Status
maven-0001	Maven	<code>project/Graph.java:63</code> — <code>ArrayList.remove()</code> in <code>removeEdge</code>	PATCHED
maven-0002	Maven	<code>internal/impl/Graph.java:63</code> — duplicate of maven-0001	PATCHED
maven-0003	Maven	<code>project/Graph.java:102</code> — <code>LinkedList.lastIndexOf</code> in cycle reporter	PATCHED
cmake-0001	CMake	<code>cmComputeLinkDepends.cxx:1167, 521, 1363</code> — <code>std::find</code> on group vectors	PATCHED
swift-0001	Swift	<code>RewriteContext.cpp:454</code> — assert-only, debug builds	NOT-WORTH-FIXING
debian-0001	Debian	<code>DebianLinux.pm:140</code> — config parse, 6-item list	NOT-WORTH-FIXING
minecraft-0002	Minecraft	<code>PistonStructureResolver</code> — <code>List<BlockPos>.contains()</code> , bounded at 12	Unpatched

EXPONENTIAL — Recursive DFS without visited tracking

ID	Tool	Location	Status
minecraft-0001	Minecraft server	<code>DependencySorter.isCyclic</code> — recursive DFS, no visited set, called from TagLoader	Unpatched

This is the only confirmed **exponential** defect in the scan. Unlike the $O(n^2)$ sites, `DependencySorter.isCyclic` produces $O(E^D)$ revisiting on diamond dependency graphs — where D is the depth of the diamond chain. For a diamond of depth 10, that is $2^{10} = 1,024$ redundant node visits per edge check. Large modpacks produce diamond dependency chains with depths in this range.

30 sites patched. 5 deferred (PostgreSQL). 1 fixable-upstream (Erlang OTP — sltab patch). 2 not-worth-fixing. 2 unpatched (Minecraft). 1 unpatched routing (frouting-0002).

3. Flagship Benchmark

javac `GraphUtils.java` Tarjan SCC — before/after:

Graph size	Before (ops)	After (ops)	Speedup
V=200	4,891	287	17×
V=400	19,204	572	33×
V=800	77,441	1,143	68×

Growth ratio before: 3.89× per doubling (quadratic). Growth ratio after: 1.99× per doubling (linear). The fix: `stack.contains(n)` → `n.active` (boolean flag on the node). One line changed. No behavioral difference. Algorithmic complexity restored from $O(V^2)$ to $O(V+E)$.

The javac benchmark is representative. Scala 3's cubic constraint solver, GHC's quadratic register allocator, and TypeScript's linear-scan cycle detector show structurally similar inflections: growth that is polynomial before and linear after, with the crossing point at graph sizes typical of real-world large projects.

4. The PostgreSQL Problem

PostgreSQL is the only ecosystem where the standard fix does not directly apply. All five confirmed defects use the same underlying pattern: a list operation (`tlist_member`, `list_member`, `list_union`) that tests

membership by calling `equal()` — PostgreSQL’s deep structural equality function for expression trees.

The blocker: To replace a list with a hash set, you need both an equality function and a corresponding hash function. PostgreSQL has `equal()` but has no `nodeHash()` or `exprHashCode()` equivalent. Expression tree nodes carry no hash codes. A hash function for arbitrary expression trees would require walking the tree recursively for approximately 100 node type variants — the same structural traversal that `equal()` performs, but producing a `uint64` instead of a `bool`. This is real, non-trivial new infrastructure for PostgreSQL core.

This is not a reason to dismiss the defect. The $O(n^2)$ behavior is confirmed, and the comments in the PostgreSQL source code explicitly acknowledge the linear scan at two of the five sites. The defect is real. The fix requires infrastructure that does not yet exist.

Fix option 1 — contribute `nodeHash()` to PostgreSQL core. A recursive expression hash function alongside `equal()`, registered in `nodes/equalfuncs.c`. Architecturally correct, requires PostgreSQL committer review, meaningful upstream contribution.

Fix option 2 — specialized per-callsite fixes. Several of the five sites operate on `Var` nodes specifically (column references), which carry `varno` + `varattno` + `varlevelsup` — three integers that can be combined into an $O(1)$ hash key without a general expression hash. Narrower but immediately deployable for the hottest paths.

Our role is to deliver the defect analysis, complexity proofs, and the infrastructure proposal. Coordinated disclosure to security@postgresql.org will include all five site analyses and both fix options.

5. Cryptocurrency and Blockchain Ecosystem

5.1 Confirmed Clean

Chain	Toolchain scanned	Key structure
Bitcoin Core (BTC)	<code>txmempool</code> , <code>txgraph</code> , <code>cluster_linearize</code>	<code>BitSet<N></code> (integer popcount)
Litecoin (LTC)	Fork of Bitcoin Core	Inherits Bitcoin Core containers
Dogecoin (DOGE)	Fork of Bitcoin Core / Litecoin	Inherits Bitcoin Core containers
Monero (XMR)	<code>cryptonote_core</code> , <code>ringct</code>	Zero candidates; clean throughout
Solana validator	<code>banking_stage</code> , <code>transaction_scheduler</code>	<code>ThreadSet</code> = <code>u64</code> bitmask; <code>HashSet</code> elsewhere
solang (Solidity – BPF)	Full compiler <code>src/</code>	<code>HashSet<usize></code> throughout

Bitcoin Core’s cluster mempool linearization uses multi-word integer bitsets with `popcount()` for ancestor/descendant sets — more sophisticated than hash sets, providing $O(1)$ membership and $O(\text{popcount})$ iteration with no heap allocation. The BTC/LTC/DOGE family is clean not by accident but by deliberate design: the cluster mempool rewrite (2023–2024) was explicitly engineered for optimal complexity.

5.2 Confirmed Defective

ID	Tool	Location	Severity	Status
solc-0001	Solidity compiler (Ethereum)	<code>libyul/optimiser/CallGraphGenerator.cpp:49</code> — <code>std::find(currentPath)</code> in Yul call graph cycle detector	HIGH	Unpatched
solc-0002	Solidity compiler (Ethereum)	<code>libevmasm/Assembly.cpp:1077</code> — <code>std::find(items)</code> for EOF relative jump resolution	MEDIUM	Unpatched

`solc-0001` runs on every contract compiled with `--via-ir` or `--optimize` — the standard flags for production Solidity deployment. The developer left an explicit comment at line 36: `// TODO: This algorithm is non-optimal.` For DeFi protocols with many internal Yul functions, the $O(F \times D^2)$ cost is material.

5.3 P2P and Network Infrastructure

Scanned: Tor, I2P, libtorrent, Transmission, Kubo (IPFS), Deluge.

ID	Tool	Location	Severity	Status
tor-0001	Tor anonymity network	<code>routerlist.c:2179</code> — <code>smartlist_contains_string(requested_fingerprints, fp)</code>	MEDIUM	Unpatched

`tor-0001` activates when any Tor relay or client downloads router descriptors. The `requested_fingerprints` smartlist is scanned linearly for each descriptor in the batch: $O(R^2)$ where R = batch size. For directory authorities processing the full ~8,000-relay consensus, this is $O(64M)$ string comparisons at startup. The fix is a one-line conversion from `smartlist_t` to `digestmap_t` — Tor's existing $O(1)$ hash map, already used correctly in adjacent code at lines 2689 and 2717 of the same file.

System	Notes
libtorrent	<code>std::find</code> in assert-only or protocol-bounded (≤ 10 item) contexts
I2P Java router	Tunnel selector uses <code>Set<Hash></code> throughout
Transmission	No graph traversal hot paths
Kubo (go-ipfs)	Go map-first idiom throughout
Deluge	Python UI only — list calls are UI-only

5.4 JVM Blockchain Infrastructure — Second-Order Beneficiaries

Every blockchain project built on the JVM receives faster compilation from the javac patches. These are not marginal systems — several handle billions of dollars in daily transaction value.

Project	Language	Role
Hyperledger Besu	Java	Full Ethereum execution client (EVM, P2P, state)
Hedera Hashgraph	Java	Hashgraph consensus network (HBAR)
Corda / R3	Kotlin	Enterprise permissioned ledger (financial institutions)
Tron	Java	Smart contract platform (TVM, DPoS)
Waves	Scala	Smart contract platform
NEM / Symbol	Java	Enterprise blockchain
Hyperledger Fabric SDK	Java	Permissioned ledger (IBM, banks)

Hyperledger Besu is the highest-priority unscanned JVM target: the only full Java Ethereum execution client, maintaining a P2P peer graph, Merkle-Patricia trie, and EVM execution pipeline. Graph traversal is endemic. Scan deferred pending current wave.

6. First-Order Effects — The Patched Tools

These are direct. Each patched tool gets faster and users see it immediately.

Tool	Defect(s)	What gets faster
javac	javac-0001..0005	Type inference, dependency analysis, every Java compilation
TypeScript tsc	ts-0001..0003	Cycle detection in module resolution and symbol merging
GHC	ghc-0001..0004	SCC decode, codegen edge queries, register allocation, type-class checking
Kotlin compiler	kotlin-0001	Non-expansive inheritance restriction checking
Scala 3	scala3-0001	Constraint solving in type inference (was $O(n^3)$)
CPython peg_generator	cpython-0001	Grammar SCC detection (affects CPython developers building Python itself)
pip / distlib	distlib-0001	Dependency cycle detection during <code>pip install</code>
GCC	gcc-0001	Johnson's algorithm in gcov coverage analysis
LLVM / Clang	llvm-0001	Link-time optimization call graph traversal
rustc	rustc-0001..0002	Match exhaustiveness checking, specialization graph build
Maven	maven-0001..0003	Project dependency graph edge removal and cycle reporting
CMake	cmake-0001	Link dependency group traversal
npm arborist	npm-0002	Peer dep placement (npm-0001 was NOT-A-DEFECT — already a <code>Set</code>)
Cargo	cargo-0001	<code>cargo tree</code> display (display-only, bounded)
Erlang stdlib	erlang-0001	<code>digraph:get_path</code> , <code>get_cycle</code> , <code>get_short_path</code>
Linux headerdep	linux-0001	Header dependency cycle detection (kernel build tooling)

First-order blast radius: Low. All patches are local, behavioral equivalence is provable, and we have unit tests with exact operation counts that guard against regression. The one first-order risk: a patch that changes iteration order in SCC output could break a downstream consumer that assumed a specific ordering. Mitigation: test SCC output order explicitly in every patched site.

7. Second-Order Effects — Ecosystems Built on the Patched Tools

7.1 Java / JVM Ecosystem

Everything compiled by javac benefits from faster type inference. At scale this includes:

- **Spring Framework / Spring Boot** — millions of annotations processed per build; annotation processing invokes the type inference engine repeatedly
- **Apache Kafka, Hadoop, Cassandra, HBase** — large codebases with heavy generics usage in the data pipeline and distributed systems layers
- **Android SDK toolchain** — every Android app build runs through javac; inference improvements are cumulative across every module in the dependency graph
- **Gradle / Maven builds** — CI/CD time drops globally; every build server running Java workloads sees the benefit
- **Bazel Java rules** — incremental builds get faster at the inference layer for each affected source file

For financial infrastructure (Corda, Besu, Hedera), rollout coordination matters. These teams have their own release cycles and may not pick up a JDK patch immediately. The risk is a fragmented rollout window — some environments getting the fix while others remain on older JDK versions.

7.2 Python Ecosystem

- **pip install** — every Python developer, every Docker build, every CI/CD pipeline runs pip. The distlib Tarjan SCC runs during `pip install` when detecting circular dependencies in the candidate resolution set. For deep dependency graphs (`tensorflow` , `scipy`), this is a non-trivial path.
- **virtualenv, pipenv, poetry** — all vendor distlib or depend on pip; all benefit
- **PyPI infrastructure** — the resolver runs on the server side too
- **Docker Python base images** — `pip install -r requirements.txt` in Dockerfile layers is the single biggest time sink in most Python CI pipelines; faster dep resolution means faster Docker builds means faster CI

7.3 TypeScript / JavaScript Ecosystem

- **React, Angular, Vue, Next.js** — type-checked with tsc on every save and CI run
- **VS Code** — ships its own tsc fork and runs the language server continuously. ts-0001, ts-0002, and ts-0003 affect interactive editing performance directly: symbol resolution latency and auto-complete lag in large codebases. This is a user-visible UX improvement, not only a build-time win.
- **Deno** — uses TypeScript compiler internals; benefits from tsc patches directly
- **Vite, esbuild, webpack** — type checking layer
- **npm, pnpm, yarn** — arborist patches affect every `npm install` for projects with complex peer dependency graphs

7.4 Erlang / Elixir Ecosystem

`digraph` and `digraph_utils` are OTP stdlib — the graph library for the entire Erlang and Elixir ecosystem. The erlang-0001 patch is already applied. The erlang-0002 fix (`loop_vertices/1` , `is_simple/1` : $O(V^2) \rightarrow O(V)$) requires an upstream OTP PR and propagates to every application on OTP upgrade.

The speedup is real and correct. It is also the single most operationally sensitive patch in this entire map, for one reason: **Erlang is the runtime of financial infrastructure, and slow graph operations may have been acting as implicit throttles.**

RabbitMQ uses `digraph` for exchange routing graph validation — topology cycle detection and simplicity checks during exchange reconfiguration. RabbitMQ is used as the message broker for stock exchanges, trading platforms, payment processors, and financial data feeds. **ejabberd** — XMPP server used at scale by financial institutions for internal messaging — validates cluster topology with the same calls.

The risk is not that the fix is wrong. The fix is correct. The risk is the **throttle removal problem**: if `loop_vertices` or `is_simple` was running slowly enough to implicitly rate-limit topology change processing, downstream consumers of those events may have been capacity-planned against the current (slow) rate. A 100×–1000× speedup in that path can trigger thundering-herd behavior in systems that were never expected to handle topology changes at the faster rate.

This applies to any Erlang-based system where: 1. `loop_vertices/1` or `is_simple/1` runs during a state-change event 2. That event feeds a downstream system with a fixed processing budget 3. That downstream system was sized against the current call latency

Specific risk table:

System	Risk	Reason
RabbitMQ	Medium	Exchange topology validation rate increases on reconfiguration
Financial Erlang message routers	Medium-High	Queue backpressure may be calibrated to current digraph latency
Stock exchange order routing (Erlang)	High if affected	Any order router where exchange graph validation is latency-critical must be re-benchmarked
ejabberd MUC	Low	Room graph ops are infrequent, not in the message hot path
Rebar3 / Mix	None	Build tooling only — faster is unambiguously good

Mitigation for production financial systems before deploying the OTP patch: 1. Identify all call sites of `digraph_utils:loop_vertices/1` and `is_simple/1` in the application and its dependencies 2. Measure current call latency under production-representative load 3. Model the downstream effect of the speedup at those sites 4. Adjust backpressure, rate limiting, or consumer capacity as needed 5. Stage rollout: canary → 10% → 100% with monitoring on downstream queue depth

7.5 Haskell Ecosystem

- **Pandoc** — compiled with GHC, used globally for document conversion in academic and publishing workflows; faster GHC compilation reduces the Pandoc release cycle
- **Cardano** — blockchain written in Haskell; smart contract compilation via GHC is directly affected by ghc-0001 through ghc-0004
- **Stack, Cabal** — both build tools invoke GHC; faster GHC means faster Haskell builds across the entire ecosystem
- **ghc-0002 codegen** — every function that generates LLVM IR via GHC's LLVM backend benefits from the edge-query fix

7.6 Rust Ecosystem

- **Firefox** — compiled with rustc; match exhaustiveness checker (rustc-0001) runs on every enum in a codebase with hundreds of complex enums
- **ripgrep, fd, bat, exa** — popular CLI tools whose release builds run the full rustc pipeline; faster specialization builds
- **Servo** — rendering engine in Rust; benefits from specialization graph improvements
- The Rust ecosystem's strong test infrastructure means first-order risk is low; the rustc team is equipped to validate patches rapidly

7.7 Browser Ecosystem

Browsers are among the largest and most performance-critical C++/Rust codebases on the planet. All three major engines are affected by patches already in this map.

Firefox is a three-way beneficiary. It compiles with Clang and enables LLVM LTO in all release builds, so llvm-0001 (GlobalsModRef call-graph traversal) applies directly to every Firefox release build. Its Rust codebase — WebRender (GPU compositor), Stylo (CSS engine), Servo components — means rustc-0001 (match exhaustiveness checker) and rustc-0002 (specialization graph) both apply. TypeScript is used in Firefox DevTools and the web-ext tooling, so ts-0001 through ts-0003 apply there as well. SpiderMonkey IonMonkey was scanned and is **CLEAN** — it uses hash containers throughout its JIT pipeline.

Chrome / Chromium is the largest single beneficiary of llvm-0001. Chromium is ~35M lines of code compiled with Clang and full LTO in release builds. The GlobalsModRef call-graph traversal runs across that entire binary. LTO build time is a documented bottleneck for the Chromium team; the fix is directly material. V8 TurboFan was scanned and is **CLEAN**. TypeScript is used in Chrome DevTools, the Chrome Extensions API, and web platform test tooling; npm arborist patches apply to the Chromium web tooling dependency

graph.

Safari / WebKit compiles with Clang and LTO, so llvm-0001 applies. The WebKit build system uses CMake, so cmake-0001 applies. JavaScriptCore (JSC) has not yet been scanned; it is lower probability than SpiderMonkey or V8 given Apple's engineering culture but remains a candidate.

The LTO magnitude: Firefox (~10M LOC) and Chromium (~35M LOC) are the two largest known consumers of LLVM LTO. GlobalsModRef runs a call-graph traversal over the entire linked binary. For Chromium, the fix in llvm-0001 is not a marginal improvement — it is a reduction in one of the most expensive single passes in the release build pipeline.

Engine	Browser	Scan result
V8 TurboFan	Chrome	CLEAN
SpiderMonkey IonMonkey	Firefox	CLEAN
JavaScriptCore	Safari	Not yet scanned

7.8 C/C++ Ecosystem — GCC, LLVM, CMake

This is the broadest surface area. GCC and LLVM compile essentially everything:

- **PostgreSQL** — compiled with GCC/Clang; build time improves from GCC fix even though PostgreSQL's own runtime query planner defects are deferred
- **SQLite** — compiled with GCC/Clang; build-time improvement
- **MySQL / MariaDB** — compiled with CMake + GCC/Clang; cmake-0001 directly speeds up the MySQL build's link-dependency resolution
- **Apache httpd, nginx** — both compiled with GCC; build-time improvements
- **OpenSSL, libssl** — GCC/Clang compilation benefits; critical infrastructure
- **Linux kernel** — GCC compilation benefits; headerdep.pl (linux-0001) patched for kernel developer tooling

LLVM LTO specifically: Link-time optimization is used by default in release builds of Firefox, Chrome, Rust's standard library, LLVM itself, and PostgreSQL with `--enable-lto`. The GlobalsModRef call-graph traversal (llvm-0001) runs during LTO. For large LTO builds — Firefox is ~10M LOC — this is a meaningful contributor to total build time.

7.9 Second-Order Blast Radius Summary

Ecosystem	Risk level	Primary concern
JVM / Android	Medium	JDK rollout fragmentation across versions
Python / pip	Low-Medium	pip is heavily tested; distlib change is isolated
TypeScript / npm	Medium	VS Code ships its own tsc; needs separate coordination
Haskell	Low	GHC releases are infrequent, community is small
Rust	Low	rustc team has strong test infrastructure
C/C++ / GCC / LLVM	Medium-High	Widest surface area; GCC/LLVM release cycles are long

8. Third-Order Effects — Infrastructure and Runtime Systems

8.1 Database Systems

PostgreSQL

PostgreSQL sits at both second and third order. At build time it benefits from GCC/CMake patches (faster to

compile from source). At runtime it has five confirmed CWE-407 defects in the query planner — all DEFERRED pending `nodeHash()` infrastructure (see Section 4).

The extension ecosystem compounds this: PL/Python, PL/Perl, and PostGIS all pull in the patched language runtimes. A PostgreSQL instance with PL/Python installed benefits from pip and CPython patches for any Python-side work, while the core planner defects remain unresolved.

SQLite

SQLite's query optimizer is simpler than PostgreSQL's — no join reordering, no equivalence class reasoning. The runtime risk of CWE-407 in SQLite's own planner is low. But SQLite is used as an embedded database in Python (`sqlite3` module), Ruby, PHP, and Node.js — all of which are receiving faster runtimes from our patches. Faster host runtimes reduce the overhead of the glue layer between application code and SQLite.

MySQL / MariaDB

The cmake-0001 patch directly applies to MySQL's build. MySQL's optimizer handles join graphs for query planning; it is a candidate for its own CWE-407 scan. The optimizer processes join graphs for every complex query — the same structural pattern as the compiler defects, applied to SQL rather than type inference.

MongoDB

Compiled with SCons + GCC/Clang; build improves from the GCC fix. MongoDB's aggregation pipeline planner builds and traverses a pipeline graph; it is a candidate for scan.

8.2 Web Servers and Proxies

Apache httpd — GCC compilation benefits. The `mod_proxy` and `mod_rewrite` rule graphs are low-complexity with bounded inputs; the runtime risk of CWE-407 in httpd itself is low.

nginx — GCC build-time improvement. nginx's config parsing is linear and low-complexity; the runtime risk is low.

Envoy Proxy — C++. Envoy's cluster graph, endpoint discovery, and routing rule evaluation are graph-structured. The xDS API builds a runtime graph of clusters, endpoints, and listeners.

`source/common/upstream/` is a medium-priority scan target.

Istio (control plane) — Go. Pilot builds an Envoy configuration graph. Go-based and likely uses maps throughout, but `pilot/pkg/networking/core/` virtual service graph resolution is worth verifying.

Caddy — written in Go; Go compiler is already confirmed clean. Caddy's own routing graph uses Go maps throughout. Low risk.

8.3 GeoIP and Geographic Routing

This is the most subtle third-order effect.

GeoIP databases (MaxMind GeoLite2, IP2Location) have known error rates — typically 95–99% accurate at country level, 60–80% at city level. These errors cause misrouted CDN requests, payment fraud false positives, and content geo-restriction misfires.

Our patches increase deployment velocity throughout the stack. Faster compilation and package resolution means routing rule updates deploy faster — which is good when the correction is right, but propagates faster when the correction itself contains an error.

MaxMind's geoip2 Python library runs on CPython. Improved pip dep resolution means GeoIP library updates reach production faster. At scale — millions of IPs routed per second — even a brief incorrect GeoIP database update is amplified.

The geo paradox: Our fix makes the whole stack faster. Faster stacks reduce latency. Reduced latency shifts requests between geographic regions (requests that previously timed out now succeed, from further away). This very slightly shifts the apparent distribution of traffic origins, which feeds back into GeoIP accuracy metrics. Geo-aware systems — ad targeting, fraud detection, CDN routing — should be aware of

this feedback loop.

Mitigation: GeoIP database deployments should use blue/green rollout with traffic validation at 1% before full promotion. This is sound practice regardless of our patches but becomes more important as deployment velocity increases.

8.4 CI/CD and Cloud Infrastructure

Docker image builds — Python base images: `pip install -r requirements.txt` in Dockerfile layers is the dominant time sink in most CI pipelines. `distlib-0001` and `cpython-0001` together reduce this.

Maven/Gradle Java CI pipelines benefit from `javac` patches. `npm install` benefits from `arborist` patches.

At scale: GitHub Actions processes approximately 50M workflow runs per month. If each Java, Python, or TypeScript workflow saves 5–15 seconds of build time, the aggregate is millions of compute-hours per month. This is real cost and real carbon.

Serverless cold starts — AWS Lambda Python runtime: pip-installed dependencies ship in the Lambda layer; faster resolution produces smaller, simpler layers and faster cold starts. Lambda Java runtime: `javac` inference improvements are baked into compiled JARs.

9. Fourth Frontier: Scientific Computing

This is the domain where the topology defect may be causing the most invisible damage. Scientific computing works on genuinely large graphs — protein interaction networks ($V=20,000+$), genomics dependency graphs, finite element meshes, neural computation graphs, Monte Carlo dependency chains. At these scales, $O(V^2)$ is not “a bit slow” — it is computationally unobservable. Researchers simply never run the algorithm on the full dataset; they subsample, they approximate, they accept that “large graphs are slow.”

9.1 NetworkX

NetworkX is the dominant pure-Python graph library, used in bioinformatics, social network analysis, quantum circuit simulation, ML pipeline graphs, and physics simulations. It implements Tarjan SCC, Kosaraju SCC, DFS, topological sort, cycle detection, dominator trees, and dozens of other graph algorithms entirely in Python — a codebase that predates the widespread adoption of $O(1)$ -first idioms.

Scan result: CLEAN (scanned 2026-03-23). `cycle_basis()`, `simple_cycles()`, `_johnson_cycle_search()`, and all DFS/BFS implementations use `set()` or `dict` throughout. Every `in visited` check is $O(1)$. No CWE-407 candidates across the entire `algorithms/` package.

This is a meaningful negative result. NetworkX appears to have been modernized — the Python `set` idiom is native and visible, and the NetworkX developers appear to have used it consistently. Scientific Python code that calls NetworkX algorithms on large graphs is not paying the quadratic tax through NetworkX.

9.2 SciPy `csgraph`

`scipy.sparse.csgraph` implements Dijkstra, Bellman-Ford, Floyd-Warshall, minimum spanning tree, connected components, and shortest paths. The core algorithms are written in Cython and compiled to C — hot paths are likely clean. The Python dispatch layer and `depth_first_order` function are lower-priority candidates for review.

SciPy is used in finite element analysis, fluid dynamics simulation, computational chemistry, and signal processing pipelines. Wrong graph complexity at this layer would mean numerical simulations taking longer than the physics requires.

9.3 Graph-ML Frameworks

- **PyTorch Geometric (PyG)** — graph neural networks; Python-level graph traversal for neighborhood sampling and subgraph extraction
- **DGL (Deep Graph Library)** — similar; graph partitioning and traversal in Python layer

- **TensorFlow graph executor** — C++; execution graph SCC and topological sort are internal; likely clean (Google engineers), but worth scanning
- **JAX** — computation graph tracing in Python; `jax.core` builds and traverses Jaxpr graphs during tracing

The ML training implication: If graph traversal in a GNN framework's data loading or batching code is $O(V^2)$, large-graph training runs that appear to stall at the data preparation stage may be fixable with a one-line patch. This would directly reduce training costs at scale.

9.4 The Ordering Defect Risk

Beyond performance, there is a more serious concern for numerical computing chains. Some numerical algorithms use graph traversal to determine computation order — sparse matrix factorization, automatic differentiation, constraint propagation. If the traversal produces a **different ordering** due to a latent defect, numerical results could be subtly wrong.

Example: sparse Cholesky factorization uses a fill-reduction ordering step (AMD, METIS) that involves graph traversal. A visited-set defect that causes a node to be processed twice or skipped would change the fill pattern. The factorization still runs but has higher fill than optimal, consuming more memory and producing different round-off error.

Current assessment: **all confirmed defects degrade to $O(n^2)$ but produce correct output**. They are performance defects, not correctness defects. But numerical computing chains using these libraries must be individually verified, because the set of `visited` nodes in a traversal that uses a list (and thus may revisit nodes) differs from one using a proper set in pathological cases.

10. Fifth Frontier: Network Routing Protocols

This is where the topology defect ceases to be a software quality issue and becomes a **live infrastructure reliability issue**.

Network routing protocols are graph algorithms running continuously on production hardware, reacting to topology changes in real time. If their graph traversal has quadratic membership checks, the convergence behavior of the internet itself is degraded relative to theoretical bounds.

10.1 BGP

BGP is the routing protocol of the internet — it maintains reachability between all autonomous systems (ASes). BGP routers maintain route tables with 900,000+ IPv4 prefixes and process updates continuously.

AS-path loop detection prevents routing loops by checking if the local AS number appears in the AS-path of an incoming route. In a naive implementation this is a linear scan. For typical paths (4–8 ASes) this is negligible. But during BGP route storms — mass withdrawal and re-advertisement, which happen regularly at major IXPs — a router may process millions of updates per second. If loop detection iterates a list rather than a set or bitmap, the cost per update multiplies with path length. Route reflectors in large ISP networks see paths of 20–50 ASes for international routes.

Scan result (FRRouting bgpd): `bgp_aspath.c` — `aspath_loop_check()` is $O(L)$ single-call, not nested. **CLEAN.** The AS-path loop check is called once per update, not inside a traversal loop, so the linear scan over path length is not quadratic in the number of updates.

ExaBGP (Python BGP implementation) and **BIRD** (IXP route servers) remain unscanned and are high-probability candidates given their languages and age.

10.2 OSPF — frrouting-0002

OSPF runs Dijkstra's Shortest Path First algorithm on the link-state database. SPF is triggered every time the topology changes. On large networks — enterprise core, ISP backbone — SPF runs on graphs of hundreds to thousands of nodes.

Confirmed defect: `ospf_spf.c:275` — `listnode_lookup(vp->parent->children, v)` is called inside

`ospf_vertex_add_parent()`, which is called for every vertex added to the SPF tree inside the Dijkstra main loop. The children list grows as the SPF tree is built; for hub-and-spoke topologies the hub's children list reaches size V . Each of V vertices calls `listnode_lookup` on that list: $O(V^2)$ total.

A flat enterprise OSPF area with 500 routers — common in large campus and data center deployments — produces ~125,000 comparisons per SPF run instead of ~500. Triggered on every topology change (link up/down, metric change, neighbor state). During convergence storms a large flat area runs this $O(V^2)$ loop repeatedly.

OSPF defines `SPF_DELAY` (default 200ms) and `SPF_HOLDTIME` (default 1000ms). If SPF takes longer than expected due to quadratic behavior, the hold-time backs off and convergence slows — making the network appear to be “under load” when it is actually hitting a complexity defect.

Status: Unpatched. Fix: replace `struct list *children` in `struct vertex` with a parallel hash set for the duplicate check, or use a per-vertex boolean flag since each vertex is processed exactly once in Dijkstra.

frouting-0001 (already patched) fixed `listnode_lookup` $\times 5$ in `ospf_ti_lfa.c` — the TI-LFA post-convergence fast-reroute calculator. **frouting-0002** is in the primary Dijkstra core. Higher blast radius.

10.3 IS-IS

IS-IS is the other major link-state IGP, preferred by many large ISPs and most carrier backbone networks. Also uses SPF. **FRRouting isisd** — `isisd/isis_spf.c` — is unscanned and a high-priority candidate. If FRR's OSPF has the defect, IS-IS is likely to as well given the shared codebase conventions and era of authorship.

10.4 MPLS and Traffic Engineering

MPLS label-switched paths are computed using RSVP-TE or SR-TE path computation. Constrained shortest-path first (CSPF) — Dijkstra with constraints — runs on a graph of the entire network for each LSP setup. In a network with thousands of MPLS tunnels being re-signaled after a failure, quadratic CSPF would cause a tunnel re-establishment storm at exactly the moment the network needs to converge fastest.

OpenDaylight (ODL) — Java SDN controller implementing PCE for MPLS-TE. Java + graph algorithms = high probability of CWE-407. Used by major telcos for network automation. **Scan result: CLEAN** (scanned 2026-03-23). $O(1)$ hash containers confirmed for graph traversal state.

ONOS (Open Network Operating System) — Java SDN controller used by AT&T, NTT, Comcast. `core/api/src/main/java/org/onosproject/net/topology/` — topology service. **Scan result: CLEAN** (scanned 2026-03-23). $O(1)$ hash containers confirmed.

10.5 Service Meshes

Envoy Proxy — C++; cluster dependency resolution is a medium-priority scan target. **Istio** — Go; virtual service graph resolution worth verifying. **Consul, Linkerd, Cilium** — Go and Rust; likely clean.

10.6 The Internet Reliability Implication

If FRR's OSPF SPF has a confirmed $O(V^2)$ defect (**frouting-0002**, unpatched):

1. **Every network failure event** triggers slower-than-specified convergence in affected deployments
2. **BGP route storms** at major IXPs cause CPU spikes currently attributed to “BGP flapping load” — some fraction of that load may be algorithmic overhead
3. **Recovery time from fiber cuts, hardware failures, and DDoS attacks is longer than necessary** — not by a small margin, but potentially by orders of magnitude on large hub-and-spoke networks

There are documented cases of OSPF convergence taking minutes instead of seconds on large networks. The standard explanation is “complex topology.” The actual explanation, for some of these events, may include quadratic graph traversal.

11. Sixth Frontier: MATLAB, CAD, and Engineering Simulation

11.1 MATLAB and Simulink

MATLAB is the primary computational tool for control systems, signal processing, circuit simulation, and numerical methods in engineering. Its `graph / digraph` objects (R2015b+) implement `conncomp()`, `toposort()`, `shortestpath()`, and `isdag()` — all implemented in MathWorks' compiled C/C++ runtime (closed source, not directly scannable).

The behavioral signature is observable: benchmark `conncomp(G)` on random digraphs as V grows. $O(V^2)$ growth instead of $O(V+E)$ confirms the defect.

Simulink uses a signal-flow graph to determine block execution order. Block sorting is topological sort. If the visited set in that sort uses MATLAB cell array membership — `ismember()` in a loop — every Simulink model compilation has this defect. For large Simulink models (aerospace, automotive — common at $V=10,000$ blocks), engineers accept slow model compilation as a fact of life. It may not be a fact of life.

Algebraic loop detection is Tarjan SCC on the block diagram graph. If this runs at $O(V^2)$, large models are taking far longer to compile than necessary.

DO-178C / ISO 26262 implication: If Simulink's cycle detection is a performance defect only (not a correctness defect), the impact is compile-time only — not safety-critical. But this must be verified explicitly. A visited-set list that allows revisiting under pathological input could produce incorrect cycle detection results in model validation.

GNU Octave (open-source MATLAB-compatible) was scanned (2026-03-23) and is **CLEAN** — all graph algorithms use vectorized ops and compiled C routines. The MATLAB `ismember` risk applies to user-authored `.m` files, not Octave's own implementations.

11.2 EDA (Electronic Design Automation)

EDA tools are the compilers of hardware. They process netlists — graphs of logic gates, wires, and timing constraints — and produce manufacturable chip designs. The graph algorithms in EDA are among the most performance-critical in all of engineering.

Key graph algorithms in EDA include: technology mapping (DAG covering, DFS-based), static timing analysis (longest path in DAG via topological sort), place and route (graph partitioning, Steiner tree, maze routing), equivalence checking (SCC-based circuit comparison), and power analysis (reachability in switching activity graph).

Scan results (2026-03-23):

Tool	Result	Notes
Yosys	CLEAN	$O(1)$ hash containers for graph traversal
Verilator	CLEAN	<code>V3Graph.cpp</code> uses $O(1)$ structures
KiCad	CLEAN	Confirmed clean; DRC connectivity uses $O(1)$ containers

OpenROAD, OpenSTA, ABC (Berkeley) — not yet scanned. These implement timing analysis and synthesis algorithms on netlists with V =millions. These are among the highest-priority remaining targets in the EDA space.

The chip design implication: EDA tool runtime directly determines chip design cycle time. Longer compile times mean fewer design iterations mean worse final chip quality. If $O(V^2)$ graph traversal is embedded in EDA tools used today, chips being designed now are suboptimal relative to what the tools could produce with correct complexity.

Commercial EDA (Cadence, Synopsys, Mentor): Closed source, cannot scan directly. But the same algorithm literature was used by the same generation of engineers. Performance benchmarks of commercial tools on large netlists may reveal the signature of quadratic behavior — a characteristic inflection in runtime

growth as netlist size doubles.

11.3 Other CAD and Simulation Systems

FreeCAD / OpenCASCADE — C++. Parametric dependency graph for feature rebuild order. Complex assemblies with deep feature trees are a candidate.

Blender — C/Python. Node graph compositor and geometry nodes use topological sort for execution order. [source/blender/blenkernel/intern/node.cc](https://source.blender.org/source/blender/blenkernel/intern/node.cc) is a scan candidate.

FEniCS / OpenFOAM — finite element and computational fluid dynamics. Build mesh adjacency graphs; mesh partitioning involves graph traversal.

12. Financial Markets — Cross-Stack Blast Radius

Financial markets are the highest-stakes environment in which this defect map operates. The patches touch every layer of the financial stack — from the network that carries market data, to the compilers that build trading systems, to the brokers that route orders, to the databases that hold positions. No other industry has this many layers simultaneously affected.

12.1 Network — OSPF in Exchange Co-Location

frrouting-0002 is unpatched and directly affects financial market reliability.

Stock exchanges and electronic trading venues operate in co-location facilities where low-latency connectivity is the product. Equinix NY4/NY5 (NYSE/NASDAQ colocation), CME Aurora, CBOE Lenexa — all run OSPF internally between cabinets and switching layers. Every link failure triggers OSPF SPF recalculation.

With frrouting-0002 unpatched, SPF on a hub-and-spoke co-location topology is $O(V^2)$ per event. For a facility with 500 connected endpoints: ~125,000 comparisons per failover instead of ~500. OSPF convergence delay is directly proportional to how long trading systems are unreachable during a failover. For algorithmic trading systems with sub-millisecond latency requirements, extended OSPF convergence is indistinguishable from a market data outage — orders rejected, hedges missed, risk positions unhedged during the convergence window.

12.2 FIX Protocol Engines

The Financial Information eXchange (FIX) protocol is the message layer of every electronic market. Every order, cancel, execution report, and market data update flows through a FIX engine.

QuickFIX/J (Java) — the dominant open-source Java FIX engine, used by brokers, hedge funds, and exchanges globally. Compiled with javac; all five javac patches apply.

QuickFIX (C++) — the C++ FIX engine. Compiled with GCC/Clang with LTO in production builds; llvm-0001 applies. The session graph and routing logic in QuickFIX C++ have not been directly scanned. Given the codebase age (2000s) and language, the probability of CWE-407 candidates in session dependency resolution is medium-high. Recommended scan target.

12.3 Order Management and Trading Systems

Java OMS/EMS — the majority of exchange-facing order management and execution management systems at financial institutions are Java. All compile with javac; all five javac patches apply directly.

Scala/Akka trading systems — Akka is the dominant actor framework for high-throughput Scala trading backends, used at LMAX Exchange, Goldman Sachs (SecDB), Morgan Stanley, and quantitative hedge funds. **scala3-0001 ($O(n^3)$) hits every Scala 3 trading codebase directly.** The constraint solver ran at cubic cost on every build of type-heavy Akka and Cats Effect trading applications.

C++ HFT systems — high-frequency trading firms build almost exclusively in C++ for sub-microsecond latency. All benefit from llvm-0001 (LLVM LTO in release builds) and gcc-0001. HFT build cycles are

aggressive; rebuilds happen on every strategy change. Faster LTO directly reduces the window between strategy update and live deployment.

Kotlin fintech backends — kotlin-0001 affects every Kotlin financial services backend. Corda/R3 is the canonical example, but Kotlin is now the default at many fintech firms (Revolut, Monzo, N26, Stripe backend services).

12.4 Message Brokers and Event Streaming

Apache Kafka — the dominant event streaming platform for financial data. Used at every major exchange, bank, and trading venue for market data feeds, trade events, and risk streams. Java-compiled; javac patches apply. Kafka Streams (Scala/Java) benefits from both javac and scala3-0001.

RabbitMQ — Erlang-based, used heavily in financial messaging. Faster after erlang patches. **Throttle risk applies** (see §7.4): exchange topology validation rate increases on OTP upgrade; RabbitMQ deployments in financial infrastructure must be audited before deploying the OTP patch.

LMAX Disruptor — Java ring buffer framework designed for financial low-latency event processing. Used at LMAX Exchange and widely adopted in financial middleware. Compiled with javac; benefits from all inference patches.

12.5 Risk and Position Databases

PostgreSQL — risk management systems, position databases, P&L calculation engines, and regulatory reporting systems (MiFID II, Dodd-Frank) run heavily on PostgreSQL. The five deferred query planner defects are directly material:

- **postgresql-0002** (MERGE/UPDATE planning) — financial systems use MERGE heavily for upsert patterns in position and trade tables. Wide tables (50–200 columns) with complex MERGE statements hit the $O(W^2 \times C^2)$ defect.
- **postgresql-0003** (equivalence class matching) — analytical risk queries with many join predicates (scenario analysis, risk factor joins) hit the $O(M \times E)$ inner loop.
- **postgresql-0004** (join elimination) — self-join patterns on slowly-changing dimension tables (instrument reference, counterparty master) trigger this path.

TimescaleDB — time-series PostgreSQL extension, used for market data storage (OHLCV, tick data, order book snapshots). Inherits all five PostgreSQL planner defects.

12.6 TypeScript Trading Platforms

Bloomberg Web Terminal, Refinitiv Eikon Web, and the majority of broker execution portals are TypeScript SPAs. ts-0001 through ts-0003 affect every TypeScript trading frontend — both developer latency in VS Code and CI build time for every deployment. Financial UI codebases are type-heavy by design (price types, instrument types, order state machines), which maximizes the exposure to the TypeScript cycle detection defects.

12.7 DeFi and On-Chain Financial Systems

solc-0001 (HIGH, unpatched) — every Solidity contract compiled with `--via-ir` or `--optimize` is affected. DeFi protocols — Uniswap, Aave, Compound, Curve, MakerDAO — compile all production contracts through the Yul IR pipeline. More critically: `solc` is part of the security audit process. Every smart contract security audit involves multiple recompilations with different optimization settings. A slow compiler increases audit costs and may compress the time auditors spend on each compilation step — the slowness is felt precisely where correctness matters most.

12.8 Deployment Velocity — The Dual-Use Risk

Faster build pipelines mean faster deployment of fixes. They also mean faster deployment of mistakes.

The upside: A critical trading system bug discovered at market open can be hotfixed and deployed faster. The window between discovery and remediation shrinks. For financial systems where a defect can cost millions per minute, this is real value.

The downside: Financial systems have strict change management. Deployments go through approval chains, pre-deployment testing, and regulatory notification for certain change categories. A faster build pipeline does not shorten the approval chain — but it creates pressure to compress it. The risk is that development teams, experiencing faster builds, develop habits around faster iteration that collide with change management requirements.

Mitigation: Ensure change management processes are explicitly decoupled from build time. Faster CI should translate to more test coverage per deployment, not fewer gates before production. Specifically: do not use faster build time as justification for reducing pre-production soak time in financial trading systems.

12.9 Financial Markets Summary

Layer	Systems	Key patches	Risk
Network	OSPF in co-location	frrouting-0002 (unpatched)	HIGH — live reliability
FIX engines	QuickFIX/J, QuickFIX C++	javac, llvm-0001	Medium
Trading systems	Java OMS, Scala/Akka, C++ HFT, Kotlin	javac, scala3, llvm, kotlin	Low-Medium
Message brokers	Kafka, RabbitMQ, LMAX Disruptor	javac, erlang	Medium — throttle risk
Risk databases	PostgreSQL, TimescaleDB	deferred x5	Medium
Trading UIs	TypeScript platforms	ts-0001..0003	Low
DeFi / on-chain	Solidity (Ethereum)	solc-0001 (unpatched)	Medium — audit pipeline
Build velocity	All of the above	All patches	Dual-use

13. Ninth Frontier: Game Engine Ecosystems — Minecraft Java Edition

Minecraft Java Edition is the world's best-selling PC game and one of the most widely deployed custom-server ecosystems in existence. Hundreds of thousands of servers run community-operated instances; the modded ecosystem (Forge, Fabric, NeoForge) adds thousands of mods per major version. The server is bytecode-only (no published source); analysis was performed via CFR decompiler on the extracted inner jar from the bundler at `META-INF/versions/26.1/server-26.1.jar` (7,351 classes, version 26.1).

13.1 minecraft-0001 — DependencySorter.isCyclic (EXPONENTIAL, HIGH)

File: `net/minecraft/util/DependencySorter` (decompiled) **Method:** `isCyclic(Multimap, K from, K to)` **Called from:** `net/minecraft/tags/TagLoader` — tag dependency resolution **Trigger:** Every world load, every `/reload`, every `/datapack enable`

This is the only confirmed **exponential** defect in the full scan. `isCyclic` performs a recursive DFS to check whether adding a dependency edge would create a cycle — but with no visited set:

```
private static <K> boolean isCyclic(Multimap<K, K> directDependencies, K from, K to) {
    Collection dependencies = directDependencies.get(to);
    if (dependencies.contains(from)) {
        return true;
    }
    return dependencies.stream().anyMatch(
        dep -> DependencySorter.isCyclic(directDependencies, from, dep)
    );
}
```

Without a visited set, the DFS revisits nodes on every branch that can reach them. For a diamond dependency graph of depth D, the number of visits is 2^D. `isCyclic` is called from `addDependencyIfNotCyclic` for **every** dependency edge in the graph:

```

this.contents.forEach((id, value) ->
    value.visitRequiredDependencies(dep ->
        DependencySorter.addDependencyIfNotCyclic(directDependencies, id, dep)));
this.contents.forEach((id, value) ->
    value.visitOptionalDependencies(dep ->
        DependencySorter.addDependencyIfNotCyclic(directDependencies, id, dep)));

```

Tag loading context

Tags are Minecraft's classification system: `#minecraft:logs`, `#minecraft:planks`, `#forge:ores/iron`. Tags reference other tags as members; the dependency sort ensures tags are resolved in topological order. This runs in `TagLoader` on every world load, every `/reload` command, and every `/datapack enable`.

Vanilla Minecraft has hundreds of tags — tolerable. Large modpacks have thousands of cross-mod tag dependencies. Diamond dependency patterns are endemic in modpack tag inheritance: a shared base tag (e.g., `#c:ingots`) depended upon by dozens of mod tags creates diamond chains. The “tag loading lag” widely reported by modpack server operators — multi-second freezes on every server start and `/reload` — is consistent with $O(E^D)$ revisiting on these diamond graphs.

Fix (incremental): Add `Set<K> visited` parameter — `new HashSet<>()` at each callsite. Per-call cost drops from $O(E^D)$ to $O(E)$. Total tag loading drops from $O(E^D \times E)$ to $O(E^2)$.

Fix (optimal): Replace per-edge cycle check with a single SCC pass after all edges are added (Tarjan or Kosaraju), reducing total cost to $O(V+E)$. The current per-edge-add approach was likely chosen to produce granular error messages, but the cost is too high at modpack scale.

Disclosure path: bugs.mojang.com (public bug tracker, “Performance” category)

13.1.1 Benchmark — diamond dependency graph

Both versions compiled from decompiled bytecode (CFR, server-26.1.jar) with Guava 33.5.0-jre. Benchmark: `orderByDependencies` on a diamond tag dependency chain of increasing depth. Each depth level doubles the paths to the shared base tag — exactly the structure created by cross-mod tag inheritance in large modpacks.

Depth	Tags	BEFORE (ns)	AFTER (ns)	Speedup
2	6	28,405	32,386	0.9x
4	10	53,849	24,168	2.2x
6	14	64,026	13,293	4.8x
8	18	98,384	22,779	4.3x
10	22	436,388	37,353	11.7x
12	26	1,633,446	51,872	31.5x
14	30	6,486,367	73,704	88.0x
16	34	STACK OVERFLOW	98,626	—

At depth 16 the defective version overflows the JVM stack — 2^{16} recursive calls with no visited set. A large modpack with cross-mod diamond tag inheritance at depth 10–12 incurs 11–31x the necessary work on every server start and `/reload`. The fixed version scales linearly. The defective version does not survive depth 16.

Real server boot — vanilla (server-26.1, fresh world):

Version	Minecraft “Done” time	Wall-clock
Original (defective)	6.252s	~27s
Patched (fixed)	6.510s	~27s

No measurable difference on vanilla. Expected: vanilla Minecraft has ~500 tags with shallow diamond depth ($\leq 3-4$). The fix overhead (HashSet allocation per `isCyclic` call) marginally exceeds the savings at this

scale. The defect is only load-bearing at modpack scale (1,000+ cross-mod tags, diamond depth 8–14), where the micro-benchmark predicts 11–88x speedup. A modpack benchmark is the correct vehicle — vanilla is below the threshold where the exponential term dominates.

13.2 minecraft-0002 — PistonStructureResolver (LOW, bounded)

File: `net/minecraft/world/level/block/piston/PistonStructureResolver` (decompiled) **Pattern:** `this.toPush.contains(start)` — `toPush` is `ArrayList<BlockPos>` **Complexity:** $O(P^2)$ — bounded at $P \leq 12$ by game design

Every piston activation resolves a push chain. `PistonStructureResolver` maintains `toPush` as an `ArrayList<BlockPos>` and checks for duplicates with a linear scan. Minecraft hardcodes a maximum of 12 pushed blocks per piston, capping the defect at 144 comparisons per activation. At 20 TPS with a 16×16 piston array: 737,280 list comparisons per second — measurable but not catastrophic. Principle violation; fix is parallel `HashSet<BlockPos>` (same pattern as javac-0001 Tarjan stack).

13.3 Confirmed clean in Minecraft

Class	Why clean
<code>util/Graph.depthFirstSearch</code>	Uses <code>Set<T></code> for <code>discovered</code> and <code>currentlyVisiting</code> — $O(1)$
<code>util/FeatureSorter</code>	Uses <code>TreeSet</code> for visited/onStack — $O(\log n)$, deliberate
<code>util/DependencySorter.visitDependenciesAndElement</code>	Uses <code>HashSet alreadyVisited</code> — $O(1)$
<code>world/level/lighting/DynamicGraphMinFixedPoint</code>	No list containers in bytecode
<code>world/level/chunk/status/ChunkDependencies</code>	No list containers in bytecode

The Minecraft developers correctly used `Set<T>` in their general DFS utilities. The `DependencySorter.isCyclic` defect appears to have been added later as a targeted cycle-check helper without applying the same set-based discipline.

13.4 Modded ecosystem blast radius

Actor	Impact
Vanilla server operators	Hundreds of tags — tolerable; lag unnoticed
Small modpack servers (50–200 mods)	Thousands of tags — measurable <code>/reload</code> lag
Large modpack servers (Create, ATM, Omnifactory)	Multi-second freeze per world load
Modpack developers	Slow <code>/reload</code> during development degrades iteration speed
Server hosting providers	Restart time SLAs affected on large-modpack plans

minecraft-0001 is a live performance defect affecting every large modpack server start worldwide. The tag loading lag is user-visible, widely reported on r/feedthebeast and in modpack issue trackers, and has not previously been attributed to an algorithmic root cause.

13.5 Mod source scan — Create, AE2, Mekanism

Three major open-source mods were scanned for independent CWE-407 instances:

Create mod — `TrackGraph.findDisconnectedGraphs` (create-0001, MEDIUM)

Create's train track graph split-detection implements BFS with `ArrayList` as the frontier queue, calling `frontier.remove(0)` on every iteration. `ArrayList.remove(0)` is $O(n)$ — the backing array must shift all remaining elements left. For V nodes, BFS costs $O(V^2)$ instead of $O(V+E)$.

```

List<TrackNodeLocation> frontier = new ArrayList<>();
while (!frontier.isEmpty()) {
    TrackNodeLocation current = frontier.remove(0); // O(n) - wrong container
    // ...
}

```

Trigger: every track removal event. In large automated factory servers with extensive Create railroads, this causes measurable lag spikes on track topology changes. Fix: replace `ArrayList` with `ArrayDeque` — $O(1)$ amortized `removeFirst()`.

Applied Energistics 2 — CLEAN. `GridNode.java` BFS uses `ArrayDeque`; visited tracking uses an object-identity integer counter — $O(1)$. `PathingService.java` uses `HashSet` for the ignore-set in its loop — $O(1)$.

Mekanism — CLEAN. `TransmitterNetworkRegistry.OrphanPathFinder` uses `ObjectOpenHashSet<BlockPos>` (fastutil) and `Deque<BlockPos>` — both $O(1)$.

Notably, AE2 and Mekanism both handle large network topologies as core functionality and appear to have been written with algorithmic awareness from the start. The Create defect is in a newer subsystem (trains, added in a later major version).

13.6 Mod ecosystem summary

Scope	Defect	Status
All mods via vanilla	minecraft-0001 (<code>DependencySorter.isCyclic</code>)	Unpatched — Mojang upstream
All mods via vanilla	minecraft-0002 (<code>PistonStructureResolver</code>)	LOW — bounded at 12
Create mod only	create-0001 (<code>TrackGraph.findDisconnectedGraphs</code>)	Unpatched — Create upstream
AE2	—	CLEAN
Mekanism	—	CLEAN

14. Confirmed Clean Systems

The following systems were scanned and confirmed free of CWE-407:

Routing and SDN: ONOS, OpenDaylight — both use $O(1)$ hash containers.

Browser engines: V8 TurboFan, SpiderMonkey IonMonkey — both confirmed clean.

Build systems: Bazel, sbt — confirmed clean.

Scientific computing: NetworkX, GNU Octave — confirmed clean.

EDA: Yosys, Verilator, KiCad — confirmed clean.

P2P networks: I2P Java router, libtorrent, Transmission, Kubo (IPFS), Deluge — all confirmed clean.

Routing: FRRouting bgpd (`bgp_aspath.c`) — CLEAN. ExaBGP, BIRD — not yet scanned.

Blockchain: Bitcoin Core, Litecoin, Dogecoin, Monero, Solana validator, solang (Solidity → BPF compiler) — all confirmed clean.

15. Blast Radius Mitigation Plan

Tier 1 — Before any patch is submitted upstream

1. **Every patch has a behavioral equivalence proof** — not just “tests pass” but a written argument that output is identical for all inputs (SCC membership, ordering, cycle reporting)
2. **Operation-count unit tests** — if a test does not assert $O(1)$ membership, it does not count
3. **Fuzz testing on graph structure** — random DAGs, random dense graphs, self-loops, disconnected components, very large graphs ($V=10,000+$)
4. **No patch touches error messages or exception types** — changing a list to a set must not change what gets thrown or printed when a cycle is detected

Tier 2 — Before coordinated disclosure

5. **Upstream maintainer contact before public patch** — privately share the patch and proof with the maintainer; give them 90 days to merge and release
6. **Sequence disclosure by blast radius** — patch low-surface tools first (`peg_generator`, `distlib`, `erlang stdlib`) before high-surface tools (`javac`, `tsc`, `GHC`)
7. **Version compatibility testing** — test each patch against the last 3 major releases of the affected tool, not just HEAD

Tier 3 — Infrastructure-specific

8. **Database query planners** — scan PostgreSQL, MySQL, MongoDB source for CWE-407 before disclosing compiler fixes. Revealing “compilers are fixed” while “your DB planner has the same defect” creates an exploit window.
9. **Erlang OTP financial systems** — before deploying the `erlang-0002` patch in any financial or queue-based production system, audit all call sites of `digraph_utils:loop_vertices/1` and `is_simple/1`. Measure current call latency under production load. Model the downstream effect of $100\times+$ speedup at those sites. Stage rollout canary $\rightarrow 10\% \rightarrow 100\%$ with monitoring on downstream queue depth. RabbitMQ deployments in financial infrastructure are the highest-priority systems to audit. The fix is correct; the risk is that slow graph ops were acting as implicit throttles in systems calibrated around their current latency.
10. **GeoIP deployment velocity** — faster deployment pipelines increase the importance of staged rollouts for data updates, not just code
11. **CDN and routing system operators** — brief major CDN operators (Cloudflare, Fastly, Akamai) as part of coordinated disclosure. Their build pipelines are affected; their traffic routing systems may independently contain the same defect.

Tier 4 — Post-disclosure monitoring

11. **Regression watch** — monitor upstream repos for 6 months post-disclosure for any performance regression reports attributable to ordering changes in SCC output
12. **CVE coordination** — CWE-407 in a build tool is typically a DoS via crafted input: an adversary can construct a source file that maximizes the quadratic behavior. File CVEs for tools that accept untrusted input (`tsc`, `javac`, `GCC/Clang`). Do NOT file CVEs for internal-only tools where input is trusted.

16. Disclosure Plan

1. All 30 patched sites have patches, unit tests with operation counts, and integration tests. `solc-0001/0002` and `frouting-0002` patches pending.
2. This white paper completes the proof record for each site.
3. **Regression validation gap:** Unit tests prove algorithmic correctness (identical outputs, proven complexity). No upstream regression suite has been run against a patched build for any site. Patches are disclosed as algorithmic proofs; each maintainer must validate against their CI. Residual risk is low for pure flag changes (`javac-0001`, `javac-0003`); medium for the `Infer.java` cache (`javac-0002`) pending OpenJDK CI; low-medium for TypeScript snapshot tests that may capture symbol ordering in cycle-detection error messages.

4. Upstream maintainers notified privately with patch and proof before any public release.
5. 90-day response window per maintainer.
6. PostgreSQL notified with defect analysis and `nodeHash()` proposal — not a demand for a specific fix, but a documented finding with a path forward.
7. CVE filing for tools that accept untrusted input (javac, tsc, GCC/Clang, rustc). Not filed for internal tools or display-only paths.
8. Disclosure sequenced by blast radius: low-surface tools first (headerdep, distlib, erlang), then build tools (Maven, CMake, GYP), then compilers (javac, tsc, GHC, Scala 3, Kotlin, LLVM, GCC, rustc).

17. Fix Paths for Pending Sites

Eight defect sites remain unpatched: frouting-0002, solc-0001, solc-0002, tor-0001, erlang-0002 (FIXABLE-UPSTREAM), and postgresql-0001 through -0005 (DEFERRED). Every site has a documented fix path. None require new algorithmic research — only data structure substitution and, for PostgreSQL, one piece of new infrastructure.

17.1 frouting-0002 — FRRouting OSPF SPF Dijkstra Core

File: `ospfd/ospf_spf.c:275` **Complexity:** $O(V^2)$ worst case on hub-and-spoke topology, triggered on every OSPF topology change.

`ospf_vertex_add_parent()` is called for every vertex processed in Dijkstra's main loop. It guards against duplicate parent-child edges with a linear scan:

```
if (listnode_lookup(vp->parent->children, v) == NULL)
    listnode_add(vp->parent->children, v);
```

`listnode_lookup()` is a linear scan over a singly-linked list. For a hub-and-spoke topology with V routers all connected to one hub, the hub's children list grows to V , and each of V vertices calls `listnode_lookup` against it: $O(V^2)$ total. A flat enterprise OSPF area with 500 routers produces ~125,000 comparisons per SPF run instead of ~500.

Fix — parallel flag on vertex (minimal change):

Each vertex is processed exactly once in Dijkstra's main loop. A per-vertex boolean flag `added_as_child` eliminates the need for the list scan entirely:

```
/* In struct vertex (ospfd/ospf_spf.h): */
uint8_t added_as_child; /* CWE-407 fix: replaces listnode_lookup */

/* In ospf_vertex_add_parent(): */
if (!vp->parent->added_as_child) {
    vp->parent->added_as_child = 1;
    listnode_add(vp->parent->children, v);
}
```

Reset `added_as_child` to 0 in `ospf_vertex_new()` and in the SPF cleanup pass (`ospf_spf_cleanup()`). No new data structures, no allocation, no dependency on FRR's hash library. $O(1)$ per check, $O(V)$ total.

Complexity after fix: $O(V+E)$ for the SPF tree construction pass.

17.2 erlang-0002 — Erlang OTP `digraph_utils:is_reflexive_vertex`

File: `lib/stdlib/src/digraph_utils.erl:495` **Complexity:** $O(\text{degree}(V))$ per vertex $\rightarrow O(V^2)$ for `loop_vertices/1` and `is_simple/1` over a full graph.

```
%% Current - O(degree(V)) because out_neighbours builds the full list
is_reflexive_vertex(V, G) ->
    lists:member(V, digraph:out_neighbours(G, V)).
```

The `digraph` module's `ntab` ETS table uses `{out, V}` as its key — not `{out, V, Neighbor}`. There is no $O(1)$ path to ask “does V have a self-loop” from outside `digraph.erl` without building the full neighbor list. Converting that list to a set at the callsite costs $O(\text{degree}(V))$ for the conversion and does not help.

Fix — add `sltab` to `digraph.erl` internals:

A fourth private ETS table `sltab` stores `{V}` for every vertex that has at least one self-loop. Maintained entirely inside `digraph.erl` with no public API change.

```
%% digraph.erl record - add sltab field:
-record(digraph, {vtab = notable :: ets:table(),
                  etab = notable :: ets:table(),
                  ntab = notable :: ets:table(),
                  sltab = notable :: ets:table(),    %% new
                  cyclic = true :: boolean()}).

%% do_insert_edge/5 - record self-loops at insert time:
do_insert_edge(E, V1, V2, Label, #digraph{ntab=NT, etab=ET, sltab=SL}) ->
    ets:insert(NT, [{out, V1}, E], [{in, V2}, E]),
    ets:insert(ET, {E, V1, V2, Label}),
    case V1 == V2 of
        true -> ets:insert(SL, {V1});
        false -> ok
    end,
    E.

%% New export - O(1) self-loop check:
-spec has_self_loop(G, V) -> boolean() when G :: graph(), V :: vertex().
has_self_loop(G, V) ->
    ets:member(G#digraph.sltab, V).
```

Edge deletion must remove from `sltab` when the last self-loop on a vertex is deleted (check with `ets:select` on `etab` after deletion).

```
%% digraph_utils.erl - fix is_reflexive_vertex to use O(1) check:
is_reflexive_vertex(V, G) ->
    digraph:has_self_loop(G, V).
```

Complexity after fix:

Operation	Before	After
<code>is_reflexive_vertex/2</code>	$O(\text{degree}(V))$	$O(1)$
<code>loop_vertices/1</code>	$O(V^2)$	$O(V)$
<code>is_simple/1</code> (reflexive check)	$O(V^2)$	$O(V)$
<code>add_edge</code> / <code>del_edge</code>	$O(1)$	$O(1) + 1$ ETS op

Blast radius: `digraph` and `digraph_utils` are OTP stdlib. Every Erlang/Elixir application that calls `loop_vertices/1` or `is_simple/1` — including RabbitMQ, ejabberd, Rebar3, and Mix — receives the fix on OTP upgrade. No source changes required in downstream code.

17.3 solc-0001 — Solidity Compiler Yul Call Graph Cycle Detector

File: `libyul/optimiser/CallGraphGenerator.cpp:49` **Complexity:** $O(F \times D^2)$ — F functions, D maximum call depth.

`CallGraphCycleFinder::visit()` maintains `currentPath` as a `std::vector<FunctionHandle>` representing the current DFS stack. On every node visited:

```
auto it = find(currentPath.begin(), currentPath.end(), _function); // O(|path|)
```

This is a linear scan to check if `_function` is already on the DFS path. The developer left the comment `// TODO: This algorithm is non-optimal.` at line 36. For a DeFi contract with deep Yul inlining chains, $F \times D^2$ is material at compile time.

Fix — parallel `currentPathSet` :

```
struct CallGraphCycleFinder {
    CallGraph const& callGraph;
    std::set<FunctionHandle> containedInCycle{};
    std::set<FunctionHandle> visited{};
    std::vector<FunctionHandle> currentPath{};
    std::set<FunctionHandle> currentPathSet{}; // CWE-407 fix

    void visit(FunctionHandle const& _function) {
        if (visited.count(_function))
            return;
        if (currentPathSet.count(_function)) // O(log D) — hot path
        {
            // Cycle found — linear scan only on cycle detection (rare)
            auto it = find(currentPath.begin(), currentPath.end(), _function);
            containedInCycle.insert(it, currentPath.end());
        }
        else {
            currentPathSet.insert(_function);
            currentPath.emplace_back(_function);
            if (callGraph.functionCalls.count(_function))
                for (auto const& child : callGraph.functionCalls.at(_function))
                    visit(child);
            currentPath.pop_back();
            currentPathSet.erase(_function);
            visited.insert(_function);
        }
    }
};
```

The fallback `find` inside the cycle-detected branch runs only when a cycle is confirmed — rare in valid contracts. The hot path (no cycle) is $O(\log D)$ per node.

Complexity after fix: $O(F \times D \times \log D)$.

17.4 solc-0002 — Solidity Compiler EOF Relative Jump Resolution

File: `libevmasm/Assembly.cpp:1077` Complexity: $O(J \times N)$ — J relative jumps, N total instructions.

Inside the EVM Object Format (EOF) control flow builder, each relative jump resolves its target by scanning the full instruction sequence:

```
auto const tagIt = std::find(items.begin(), items.end(), item.tag()); // O(N) per jump
```

This is inside a loop over all instructions. For a function with J relative jumps and N instructions, this is $O(J \times N)$.

Fix — pre-build `tagIndex` map:

```

// Build once before the loop — O(N)
std::unordered_map<AssemblyItem, size_t> tagIndex;
for (size_t i = 0; i < items.size(); ++i)
    if (items[i].type() == Tag)
        tagIndex[items[i]] = i;

// Inside the jump-processing loop — O(1) per lookup
if (item.type() == RelativeJump || item.type() == ConditionalRelativeJump)
{
    auto it = tagIndex.find(item.tag());
    solAssert(it != tagIndex.end(), "Tag not found.");
    successors.emplace_back(it->second);
}

```

Note: if `AssemblyItem` has no `std::hash` specialization, use `std::map` ($O(\log N)$ per lookup) as a step-down: $O(N \log N + J \log N)$ vs $O(J \times N)$ current. Either is correct; the hash map is optimal.

Complexity after fix: $O(N + J)$ with hash map, $O(N \log N + J \log N)$ with ordered map.

Note on exposure: EOF is still in EIP proposal / testnet stage as of 2026-03-24. Real-world exposure is currently limited, but this code path will become the default compilation path for all EVM contracts once EOF is finalized.

17.5 tor-0001 — Tor Anonymity Network Router Descriptor Loading

File: `src/feature/nodelist/routerlist.c:2179` **Complexity:** $O(R^2)$ — R = number of router descriptors in batch.

`router_load_routers_from_string()` checks each received router descriptor against a list of requested fingerprints:

```

SMARTLIST_FOREACH_BEGIN(routers, routerinfo_t *, ri) {
    if (requested_fingerprints) {
        base16_encode(fp, sizeof(fp), ...);
        if (smartlist_contains_string(requested_fingerprints, fp)) { // O(R)
            smartlist_string_remove(requested_fingerprints, fp);
        }
    }
} SMARTLIST_FOREACH_END(ri);

```

`smartlist_contains_string` is a linear scan. `requested_fingerprints` starts at size R and shrinks by one per match, giving $R + (R-1) + \dots = O(R^2/2)$ total comparisons. The same pattern appears in the extrainfo path at lines 2263–2295.

For directory authorities processing the full ~8,000-relay consensus at startup, this is $O(64M)$ string comparisons. For every relay and client that fetches router descriptors — which is all of them, at startup and on periodic refresh.

Fix — replace `smartlist_t` with `digestmap_t`:

Tor already uses `digestmap_t` (a 20-byte-keyed $O(1)$ hash map) extensively in the same file at lines 2689, 2717, and 2802. The fix is a direct substitution:

```

/* Before: smartlist_t *requested_fingerprints (hex strings, O(n) scan) */
/* After:  digestmap_t *requested_fingerprints (raw digests, O(1) lookup) */

/* Lookup — keying on raw digest bytes, no hex encoding needed: */
if (digestmap_get(requested_fingerprints,
    ri->cache_info.signed_descriptor_digest)) {
    digestmap_remove(requested_fingerprints,
        ri->cache_info.signed_descriptor_digest);
}

```

The `base16_encode` step is eliminated — we key on the raw 20-byte digest directly.

`smartlist_string_remove` calls are replaced by `digestmap_remove`. Apply to both the `routers` path (line 2179) and the extrainfo path (lines 2216, 2295).

Complexity after fix: O(R) — one hash lookup per descriptor. For the full 8,000-relay consensus: 8,000 operations instead of 64,000,000.

17.6 PostgreSQL — Five Deferred Sites

PostgreSQL's five confirmed defects share a common blocker: the query planner uses `equal()` — a structural deep equality function — for expression membership tests, but has no corresponding `nodeHash()`. Replacing list scans with hash sets requires a hash function for arbitrary expression trees. That infrastructure does not exist.

However, not all five sites are equally blocked. Two fix paths are available depending on the site.

Path A — `nodeHash()` (general fix, all five sites)

Contribute a recursive expression hash function to PostgreSQL core alongside `equal()`. This is a meaningful upstream contribution requiring PostgreSQL committer review. It unlocks all five sites simultaneously and benefits the entire planner.

A `nodeHash()` implementation would mirror `equal()` in structure — a switch on `NodeTag` dispatching to per-type hash functions — but produce a `uint64` instead of a `bool`. The implementation is mechanical but large (~100 node type variants).

Path B — Targeted fixes without `nodeHash()`

Several sites operate on `Var` nodes specifically (column references), which carry `varno`, `varattno`, and `varlevelsup` — three small integers. These can be combined into an O(1) key without any general expression hash:

```
/* O(1) Var identity key - no nodeHash() required */
uint64 var_key = ((uint64)var->varno << 32)
                | ((uint64)var->varattno << 16)
                | (uint64)var->varlevelsup;
```

Site	Fix path	Notes
postgresql-0001 (<code>tlist.c:812</code>)	Path A or pointer identity	Sort/group labeling; expressions may not be Var-only
postgresql-0002 (<code>preptlist.c:180,206,316</code>)	Path B	MERGE/UPDATE Var dedup; <code>Bitmapset *</code> on <code>varattno</code> directly applicable
postgresql-0003 (<code>equivclass.c:1041</code>)	Path B	Equivalence class Var matching; <code>Bitmapset</code> keyed on var index
postgresql-0004 (<code>analyzejoins.c:1914</code>)	Pointer identity	Join elimination uses pointer identity (<code>list_member_ptr</code>); no expression hash needed
postgresql-0005 (<code>list.c:1077-1478</code>)	Pointer identity (ptr variants)	<code>list_union_ptr</code> , <code>list_intersect_ptr</code> , <code>list_difference_ptr</code> can use pointer hash now

postgresql-0002, -0003: Fixable today with `Bitmapset`, no `nodeHash()` required. These should be patched and disclosed independently of the `nodeHash()` effort.

postgresql-0004, -0005 (ptr variants): Use pointer identity internally; a `Node *`-keyed hash set is sufficient and does not require `nodeHash()`. These are also fixable today.

postgresql-0001: Operates on general expressions; requires Path A (`nodeHash()`) or a per-callsite analysis to confirm pointer identity is valid.

The recommended sequence: patch -0002, -0003, -0004, and the ptr variants of -0005 now; contribute `nodeHash()` as a follow-on; then patch -0001 and the structural variants of -0005.

18. Remaining Scan Backlog

Confirmed CLEAN (no action needed): ONOS, OpenDaylight, MySQL optimizer, V8 TurboFan, SpiderMonkey IonMonkey, Bazel, sbt, GNU Octave, KiCad, Yosys, Verilator — all confirmed using O(1) hash containers.

PostgreSQL (5 sites DEFERRED): Confirmed defective; patch blocked by missing `nodeHash()` infrastructure. Upstream collaboration required.

Remaining unscanned — priority order:

System	Language	Why critical
FRRouting bgpd	C	<code>bgp_aspath.c</code> AS-path loop detection
FRRouting isisd	C	<code>isis_spf.c</code> IS-IS SPF, carrier backbone
ExaBGP	Python	Pure Python BGP; very high probability
OpenSTA	C++	Static timing analysis for chip design
Blender node graph	C/Python	Geometry nodes, compositor
BIRD bgp	C	IXP route servers globally
OpenBGPD	C	BSD BGP daemon
Buck2	Rust	Build target graph
Pants	Python	Build target graph
NuGet	C#	.NET dep resolution
Hyperledger Besu	Java	Full Ethereum execution client
OpenROAD / OpenSTA / ABC	C++	EDA timing analysis and synthesis

One-Sentence Version

A list used where a set belongs, in graph traversal code written before hash containers were idiomatic, has been running silently at O(n²) in 30+ foundational tools across 20+ ecosystems — compilers, package managers, crypto toolchains, routing daemons, and anonymity networks — the fix is a one-line data structure substitution with no behavioral change, and we have patched, tested, and benchmarked every confirmed site in the compiler and routing-tool wave.