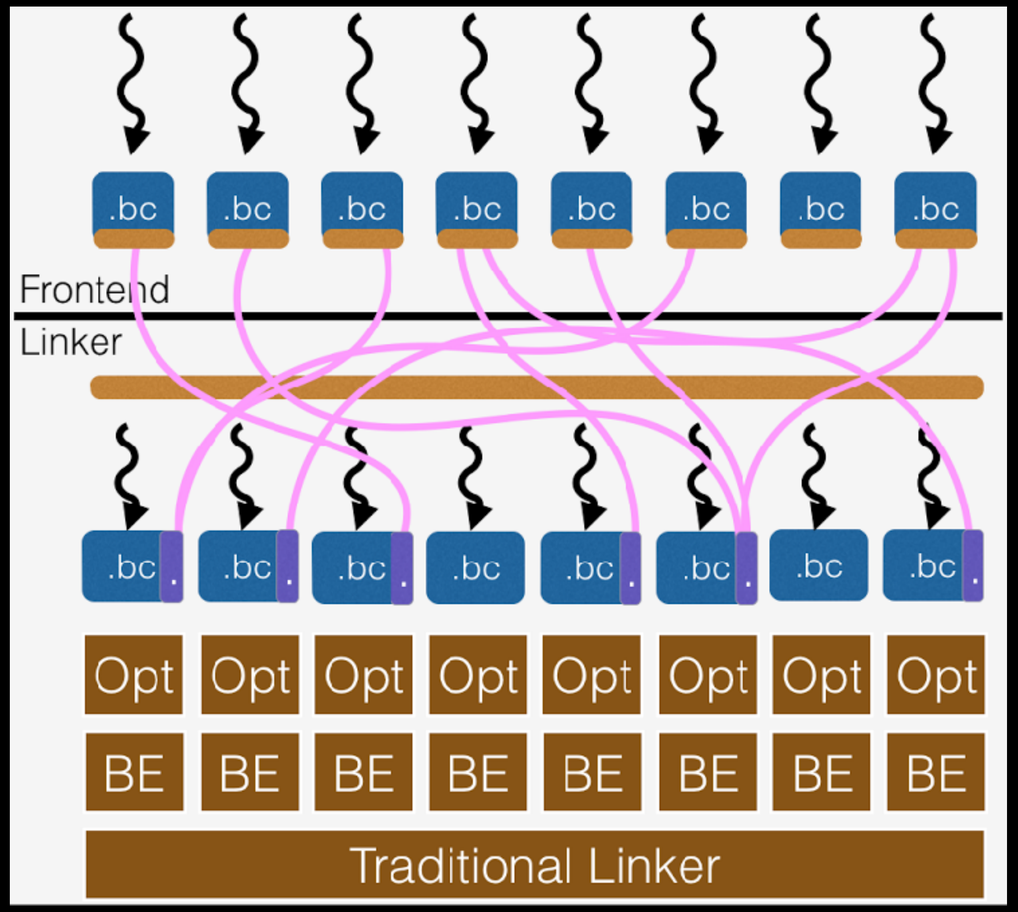
ThinLTO Summaries in JIT Compilation

Stefan Gränitz, Apple

LLVM Developers' Meeting, San Jose, Oct. 2018

ThinLTO Module Summaries

https://clang.llvm.org/docs/ThinLTO.html



Allows us to reconstruct global call-graph without parsing bitcode

JIT Overview

IR → MC: Performance compared to precompiled execution

JIT Overview

IR → MC: Performance compared to precompiled execution

	Load-time	Runtime
Eager	slow	native
Lazy		

JIT Overview

IR → MC: Performance compared to precompiled execution

	Load-time	Runtime
Eager	slow	native
Lazy	fast	slow

JIT Overview

IR → MC: Performance compared to precompiled execution

	Load-time	Runtime
Eager	slow	native
Summary Based	fastest	fast
Lazy	fast	slow

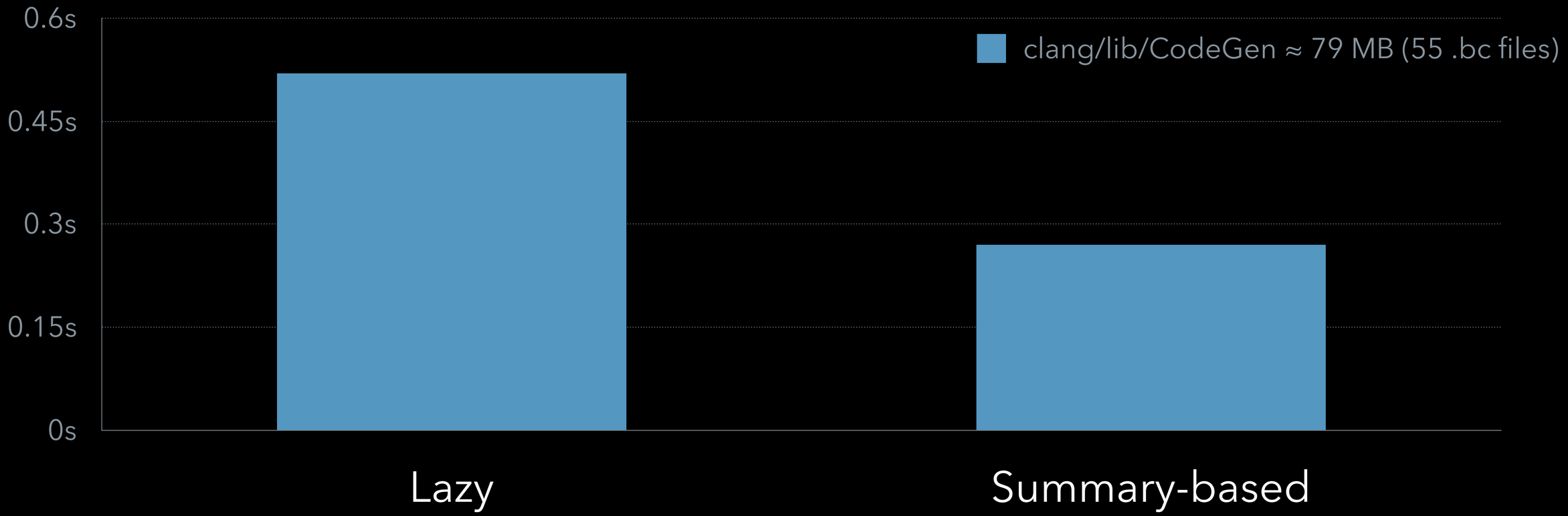
JIT Overview

IR → MC: Performance compared to precompiled execution

	Load-time	Runtime
Eager	slow	native
Summary Based	fastest	fast
Lazy	fast	slow

Load-time Performance Benchmarks*

Time required to locate symbols in modules and determine linkage types



* https://github.com/weliveindetail/ThinLtoJitBench

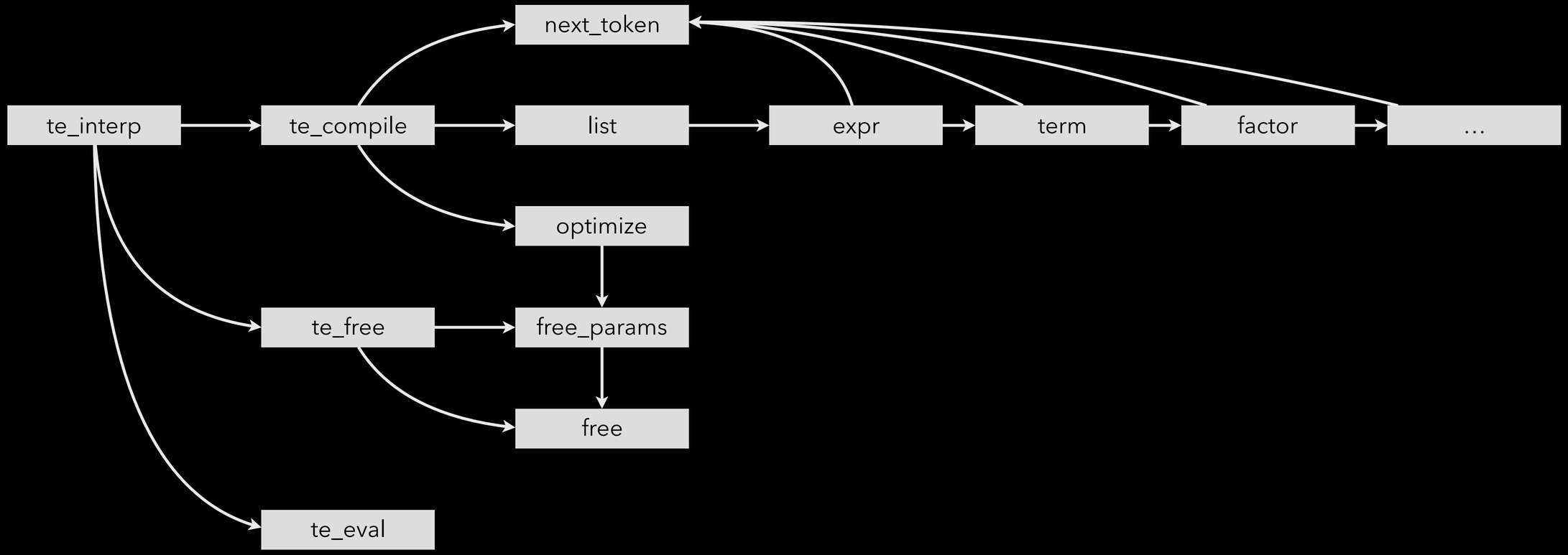
JIT Overview

IR → MC: Performance compared to precompiled execution

	Load-time	Runtime
Eager	slow	native
Summary Based	faster	fast
Lazy	fast	slow

Anticipate Compiler Interceptions

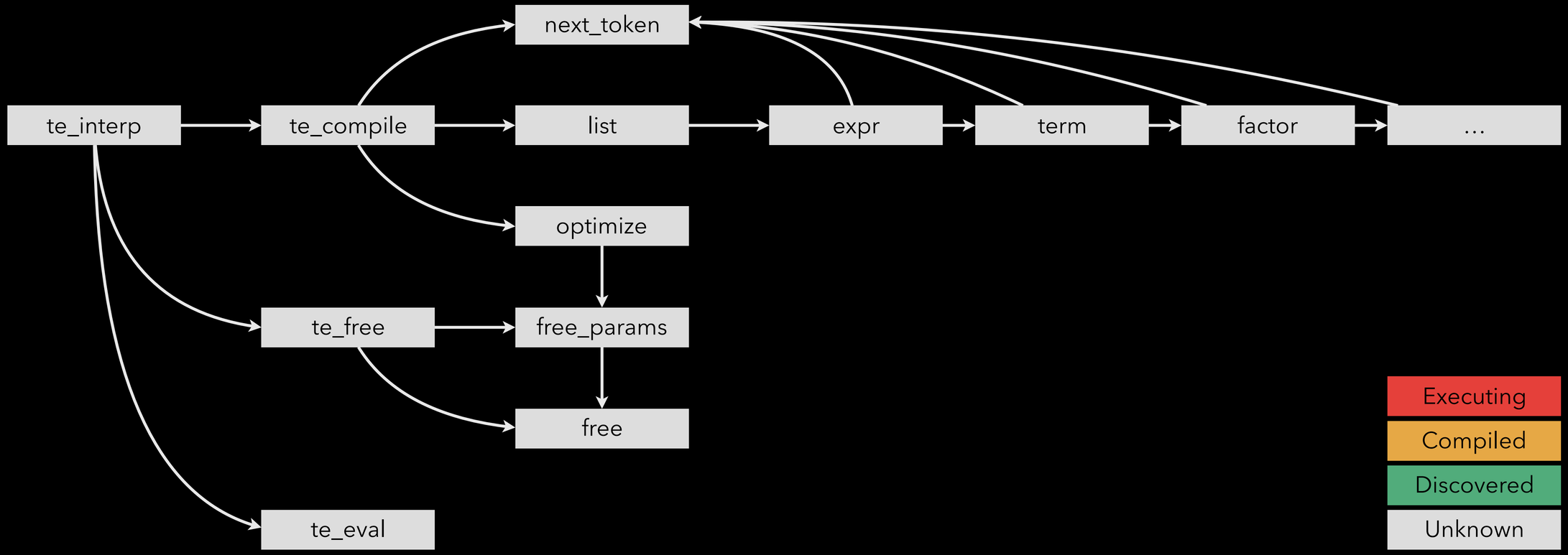
Subset of call graph for TinyExpr* evaluation engine for arithmetic expressions



* https://github.com/codeplea/tinyexpr

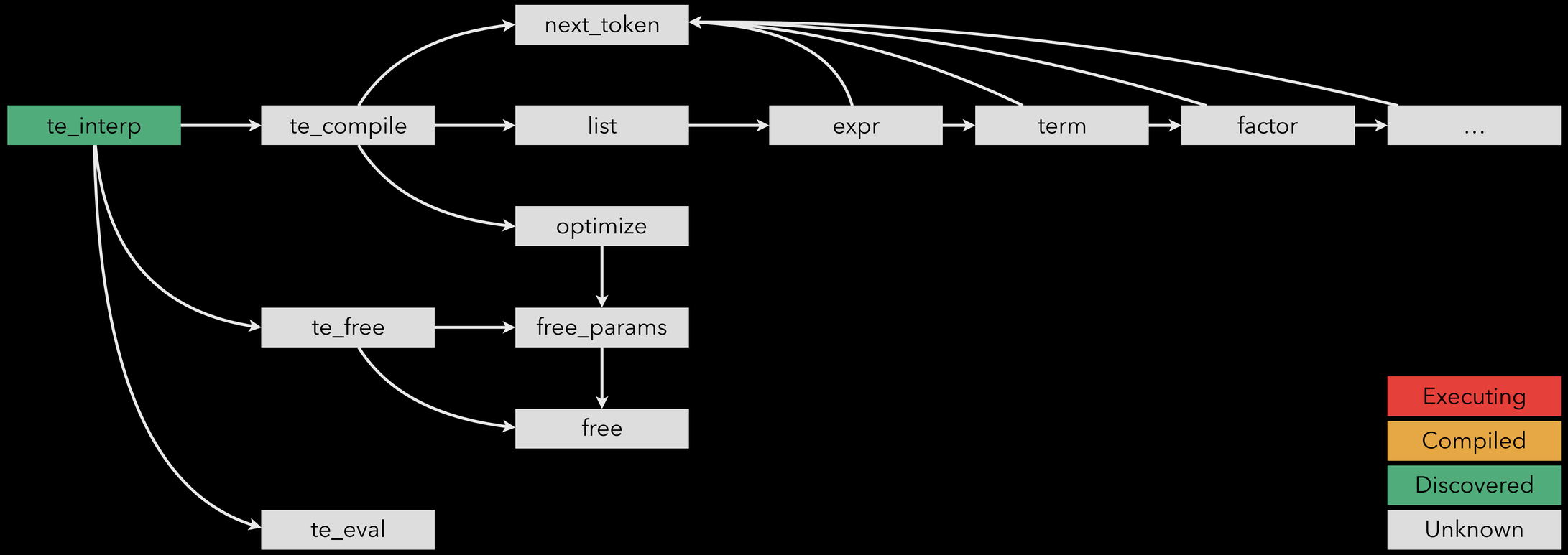
Anticipate Compiler Interceptions

Subset of call graph for TinyExpr evaluation engine for arithmetic expressions



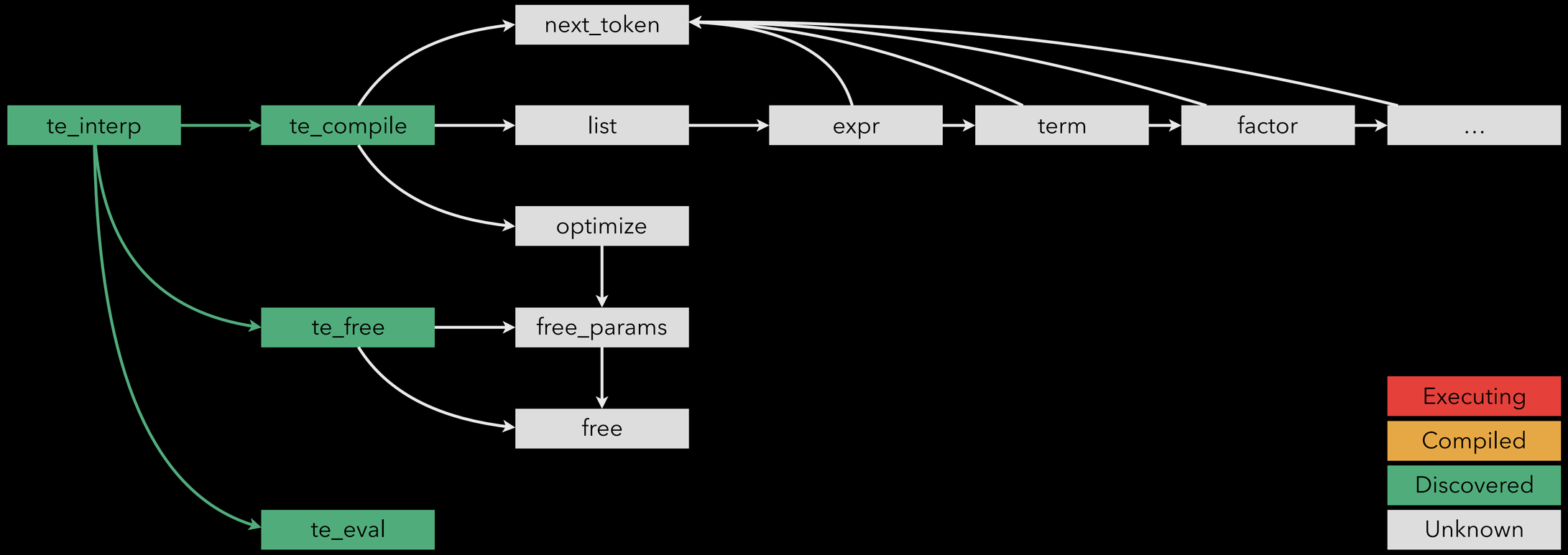
Anticipate Compiler Interceptions

Find entry point: Lookup te_interp in combined summary index



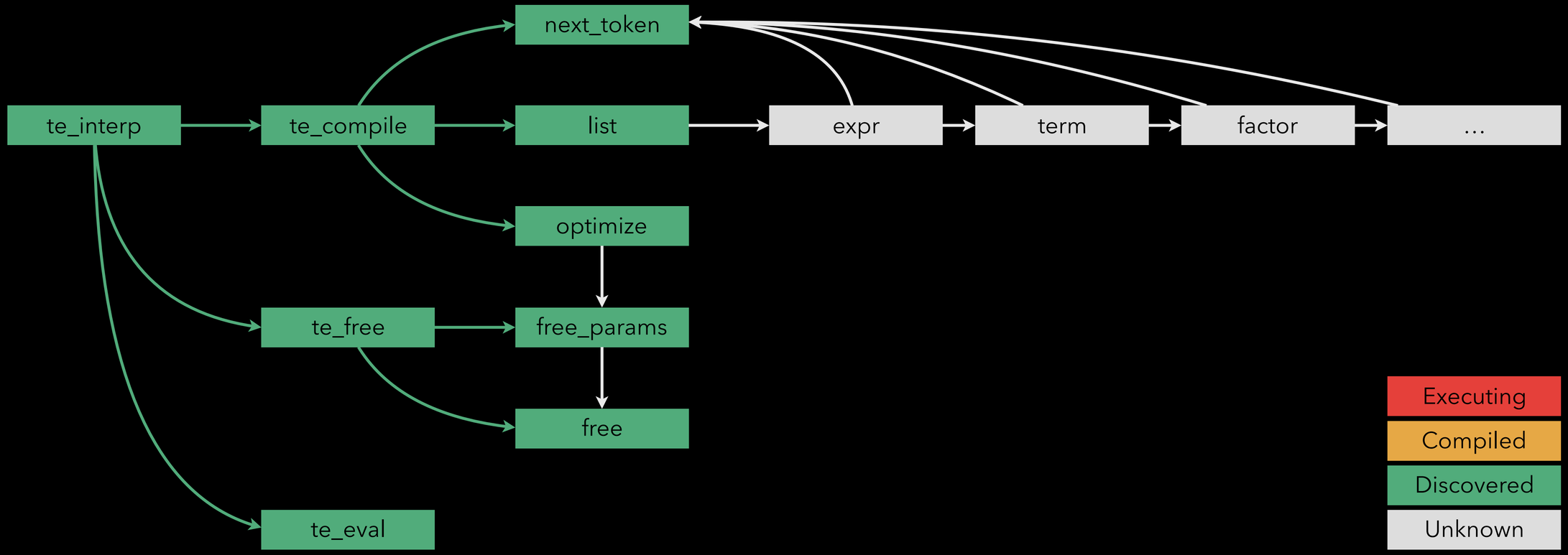
Anticipate Compiler Interceptions

Lookahead: Traverse call-graph to discover future callees



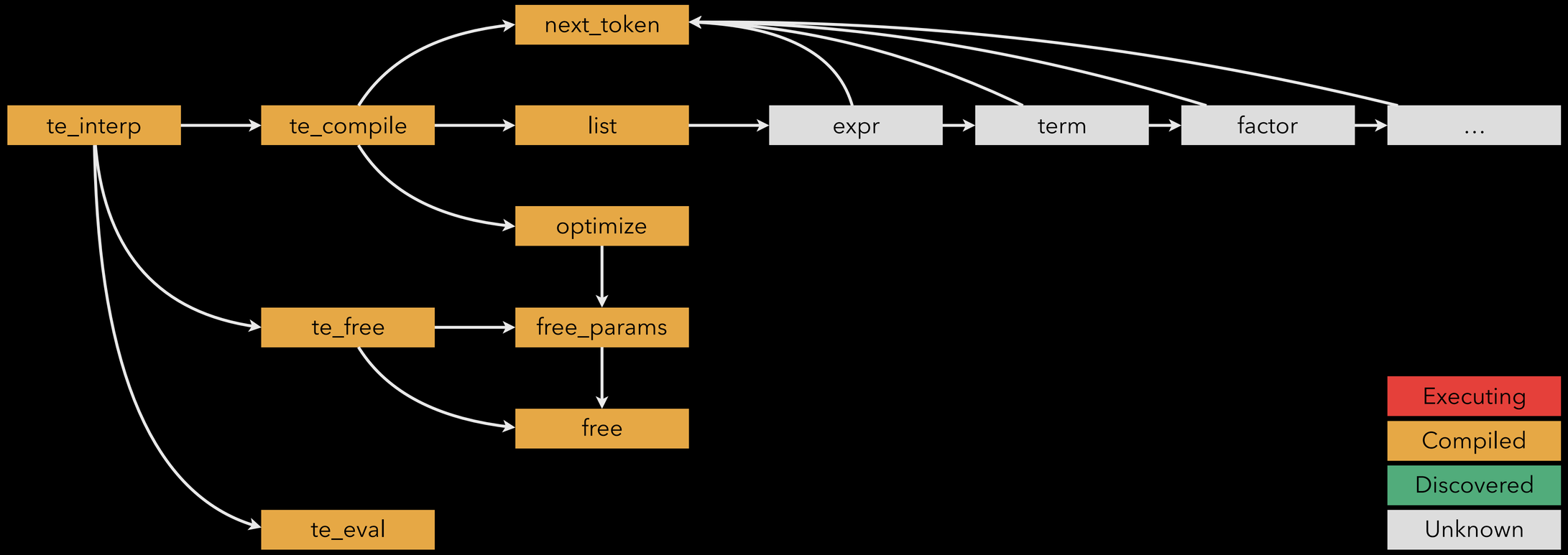
Anticipate Compiler Interceptions

Lookahead: Traverse call-graph to discover future callees



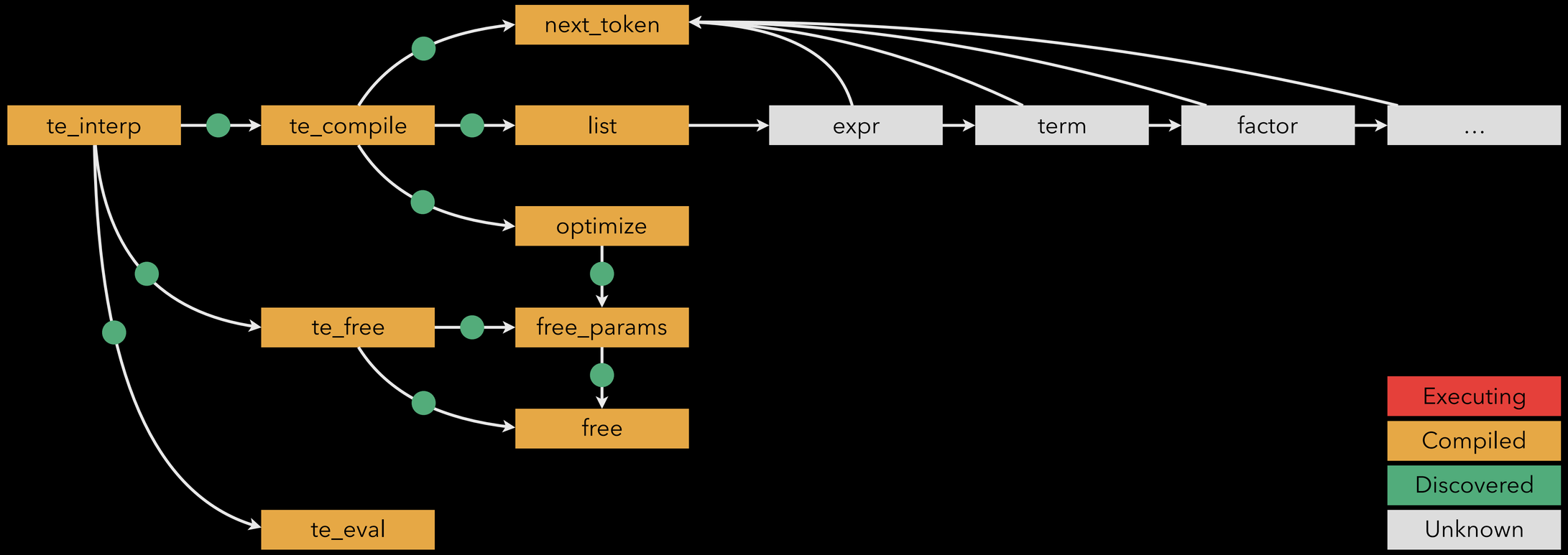
Anticipate Compiler Interceptions

Parse bitcode and compile all in one go



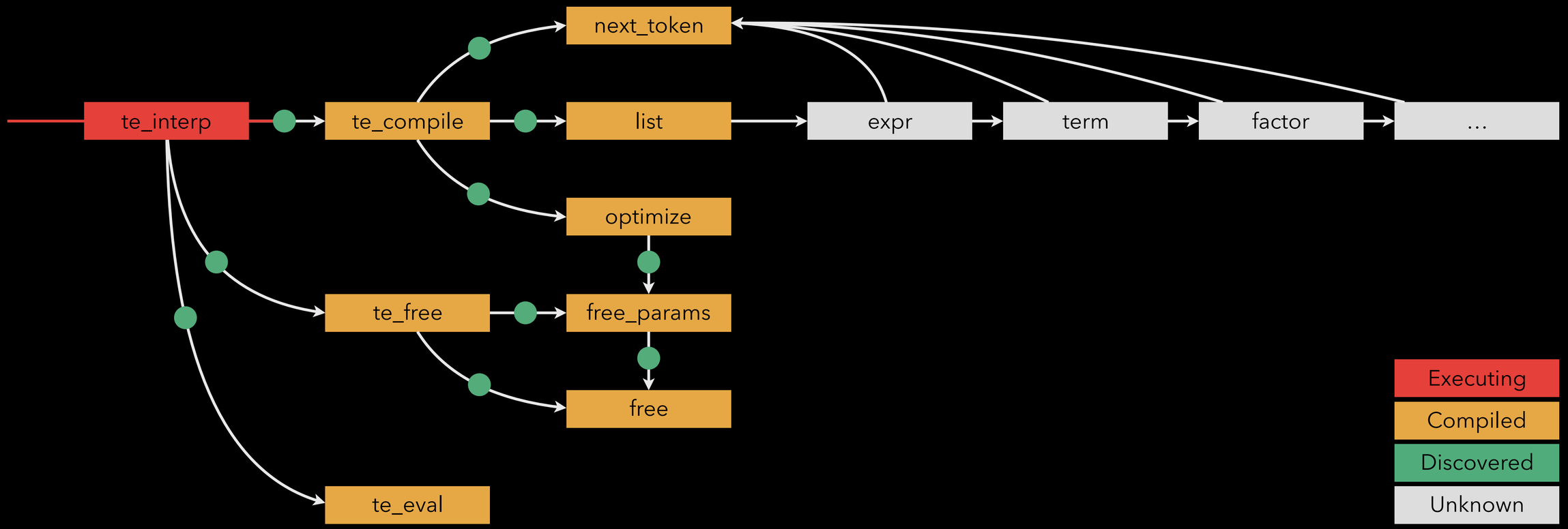
Anticipate Compiler Interceptions

Install notification stubs to report execution path back to JIT



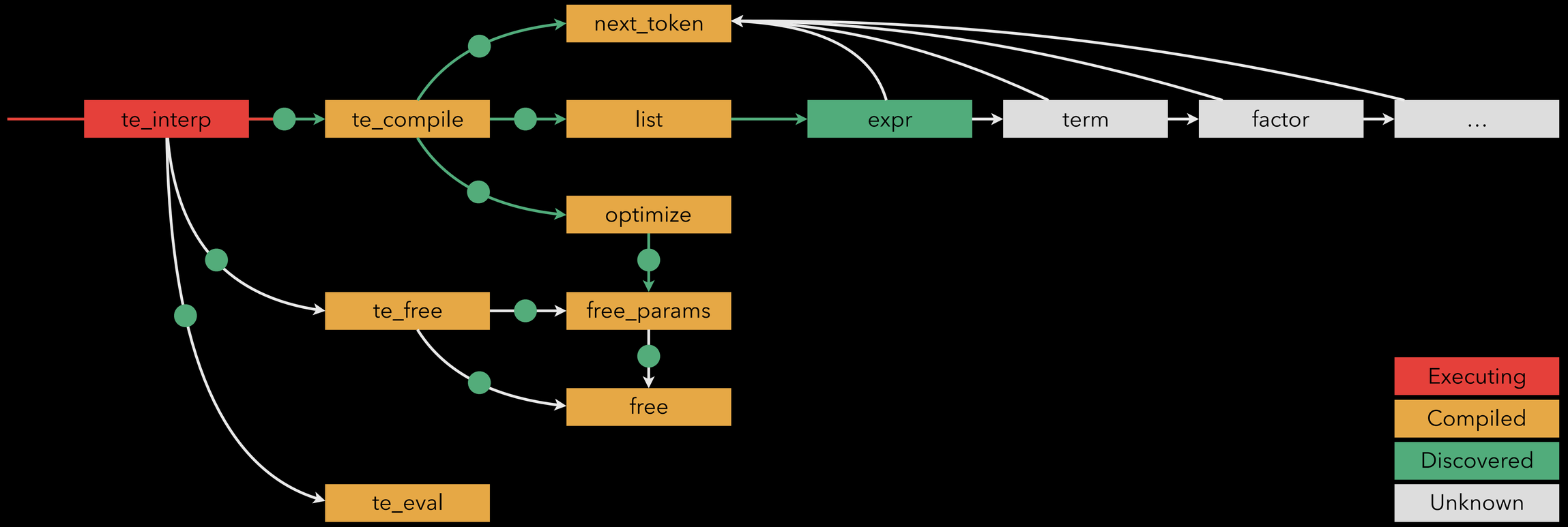
Anticipate Compiler Interceptions

Pass through notification stubs at runtime



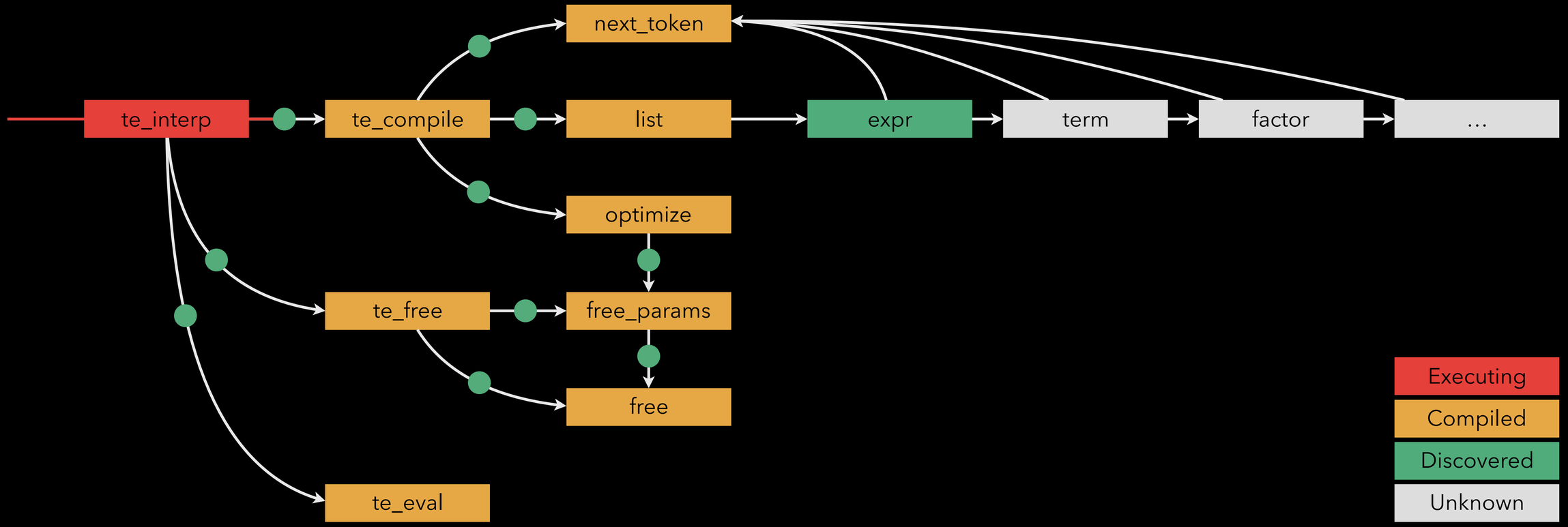
Anticipate Compiler Interceptions

Spawn new discovery in the background



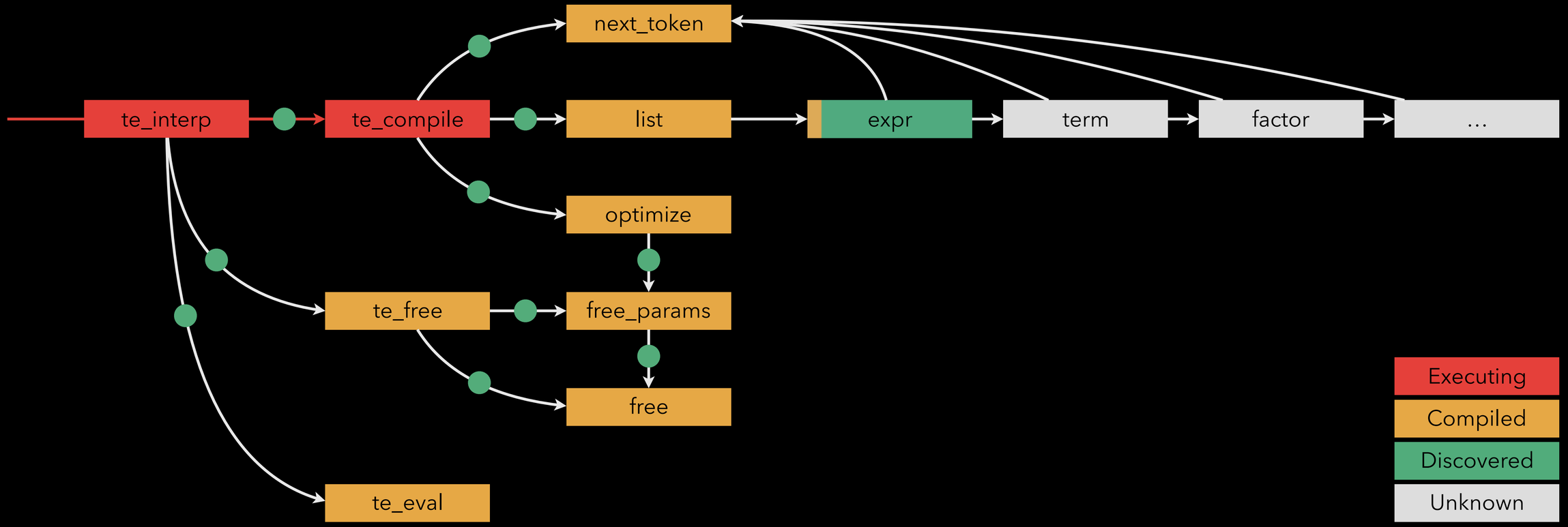
Anticipate Compiler Interceptions

Execution continues seamlessly



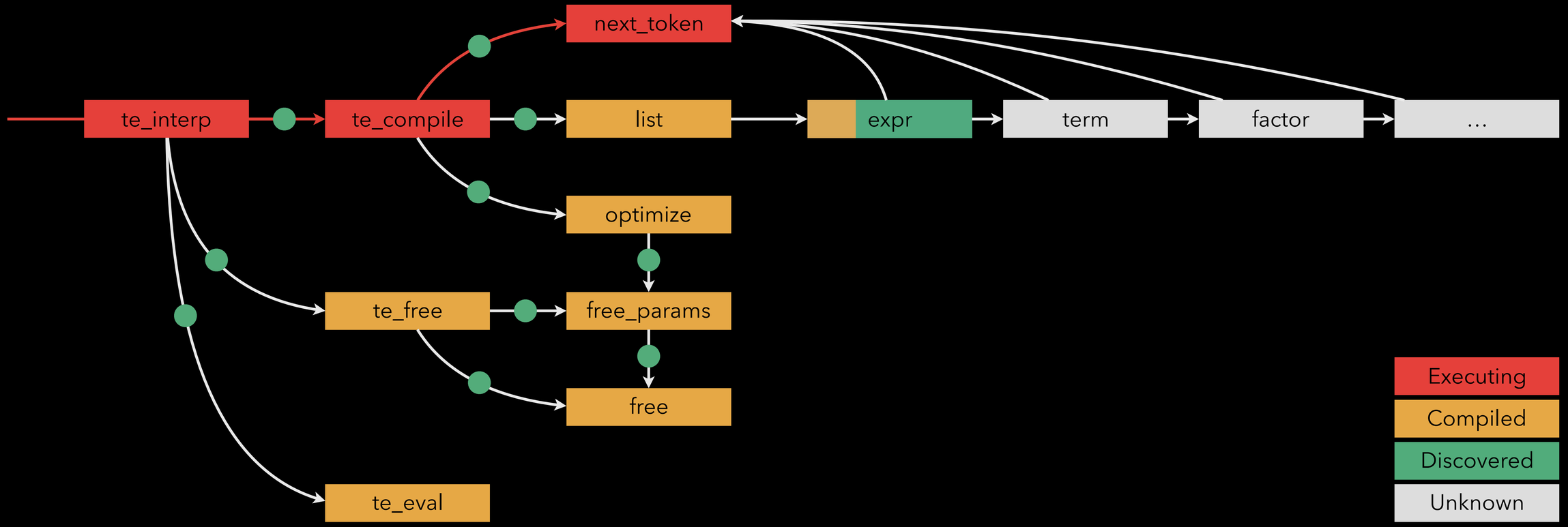
Anticipate Compiler Interceptions

Compile newly discovered functions in the background



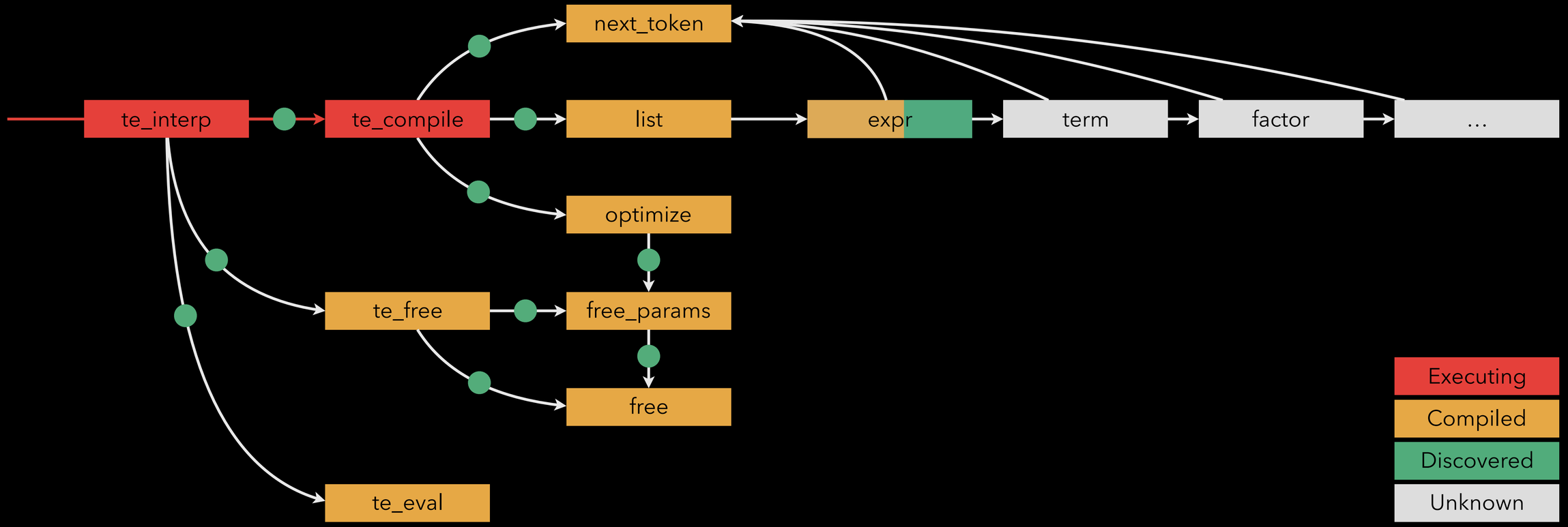
Anticipate Compiler Interceptions

Compile newly discovered functions in the background



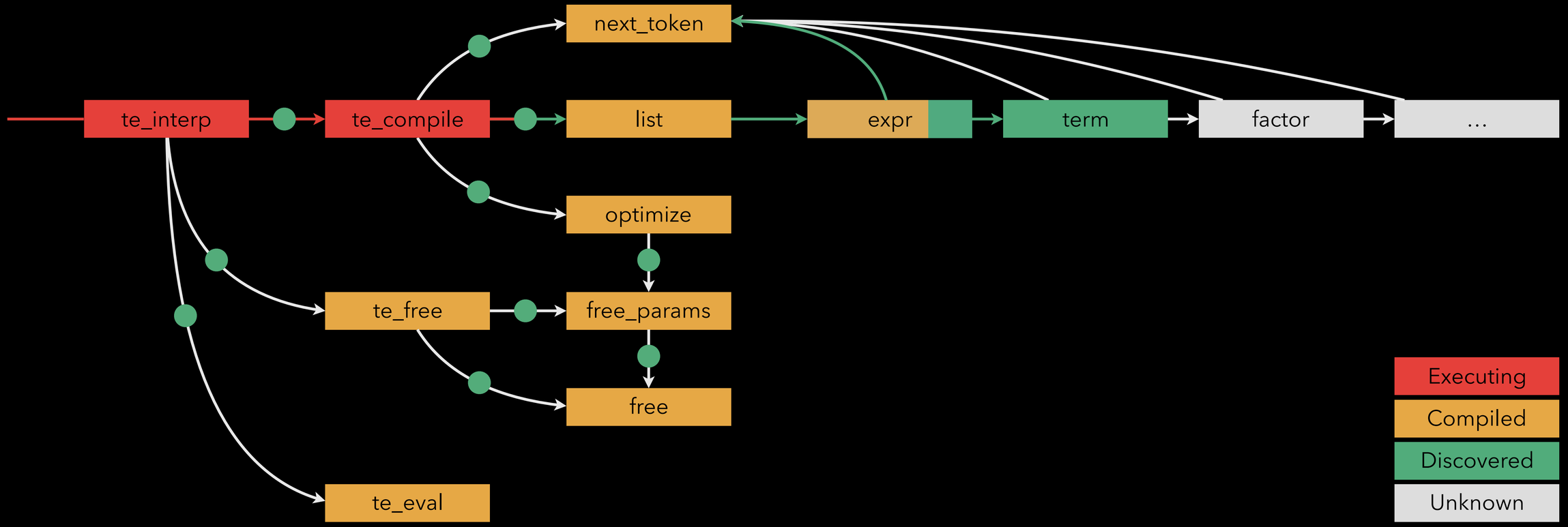
Anticipate Compiler Interceptions

Compile newly discovered functions in the background



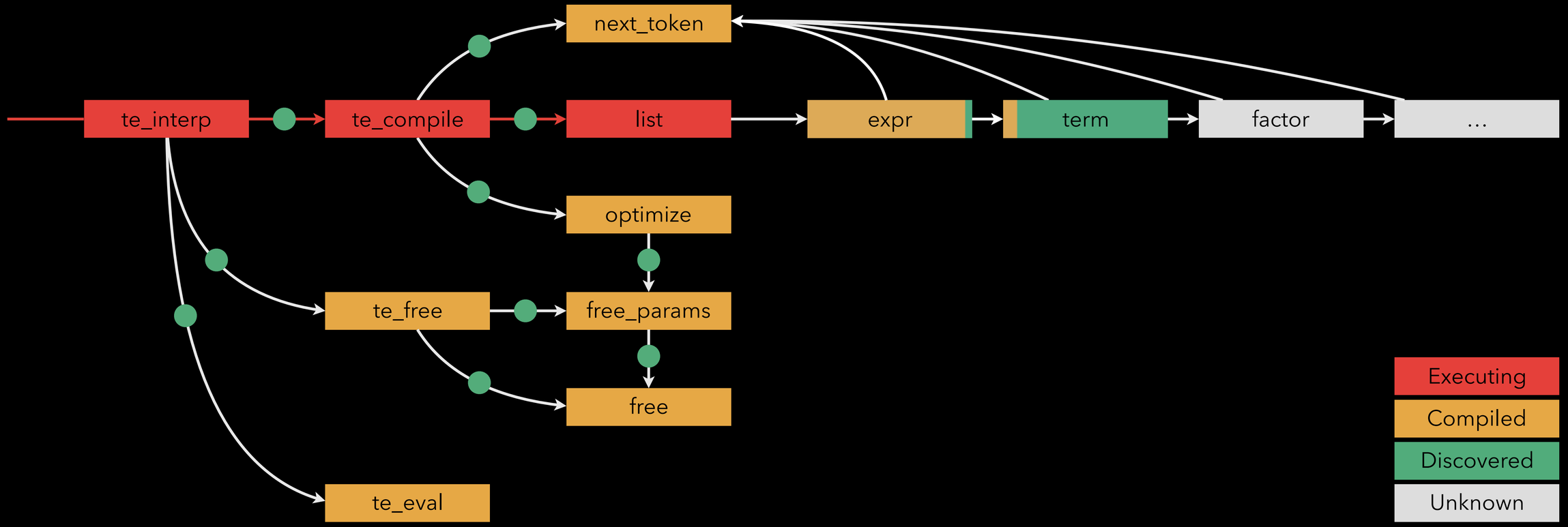
Anticipate Compiler Interceptions

Compile newly discovered functions in the background



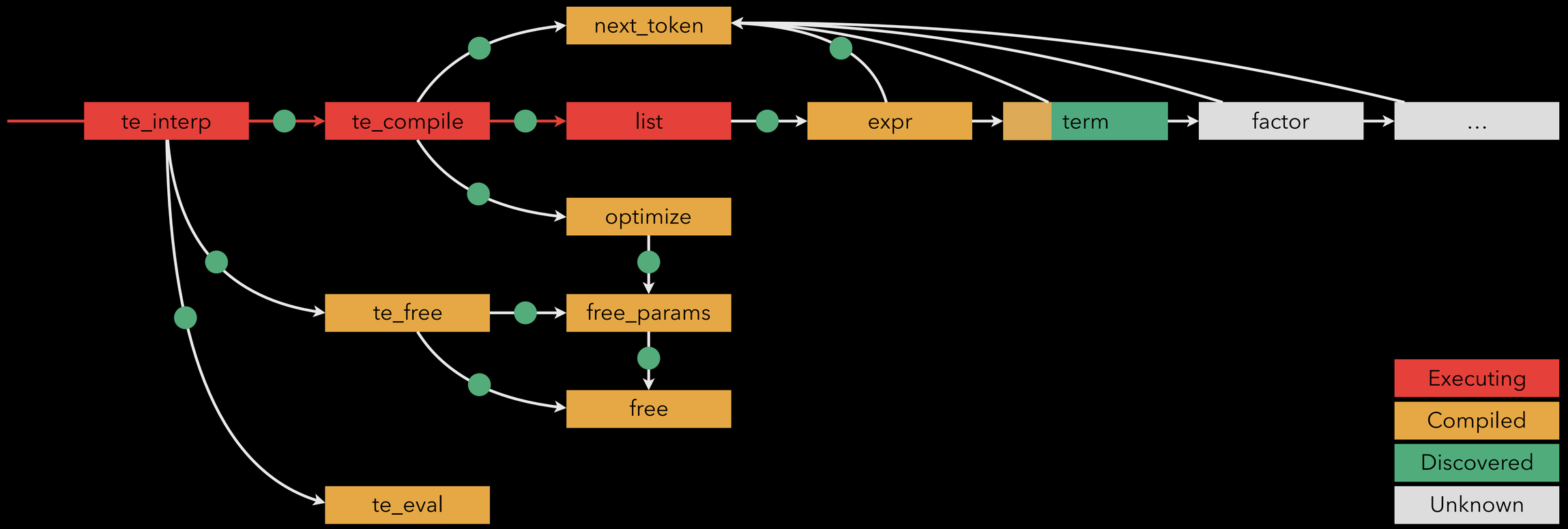
Anticipate Compiler Interceptions

Compile newly discovered functions in the background



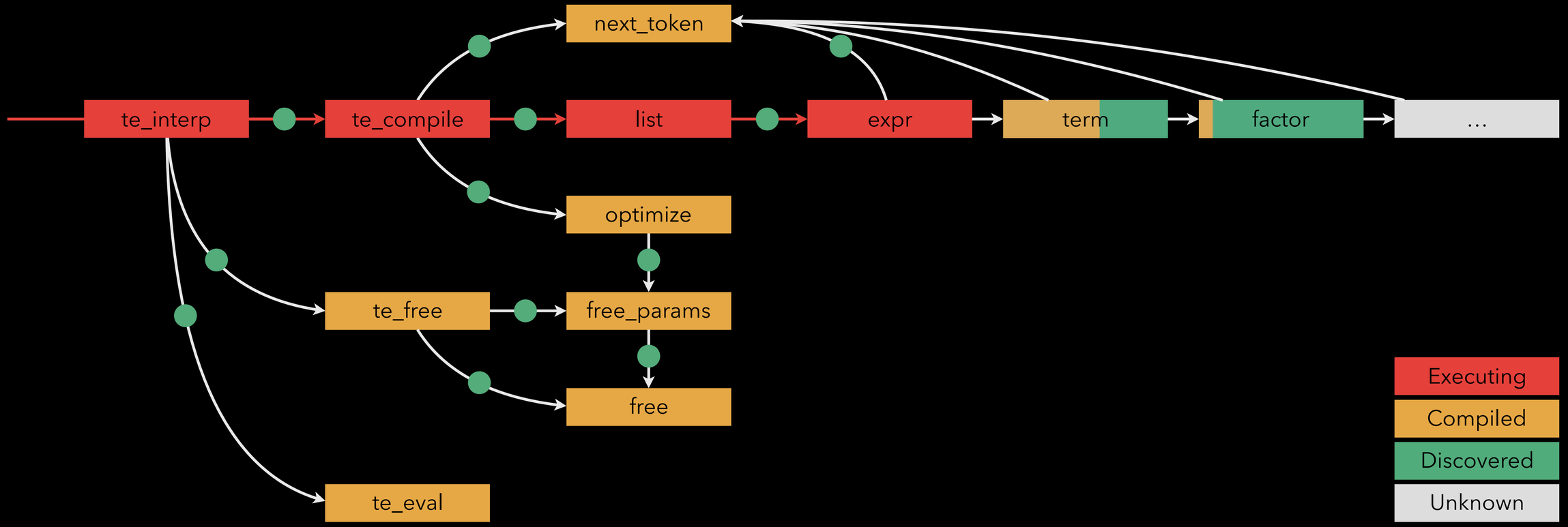
Anticipate Compiler Interceptions

Compile newly discovered functions in the background



Anticipate Compiler Interceptions

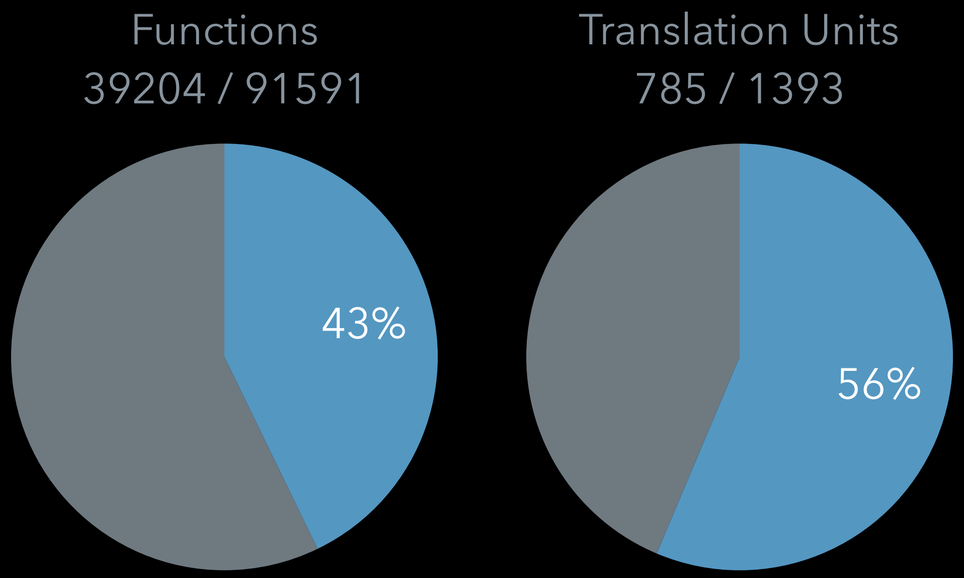
Continue execution with close-to native performance



Why not ...
build Clang Stage1 in-memory?

Build Clang Stage1 in-memory?

Compile less:
Stage1 code coverage building Stage2 (Clang-7)



... and benefit from pipelining!

Thank you!

ThinLTO Summaries in JIT Compilation
Stefan Gränitz, LLVM Developers' Meeting, San Jose, Oct. 2018