Advancing Clangd

Bringing persisted indexing to Clang tooling

Marc-André Laperle, Ericsson

AGENDA

1. Introductions
2. What’s new in Clangd
3. The road to persisted indexing
4. Current state
5. Future and challenges

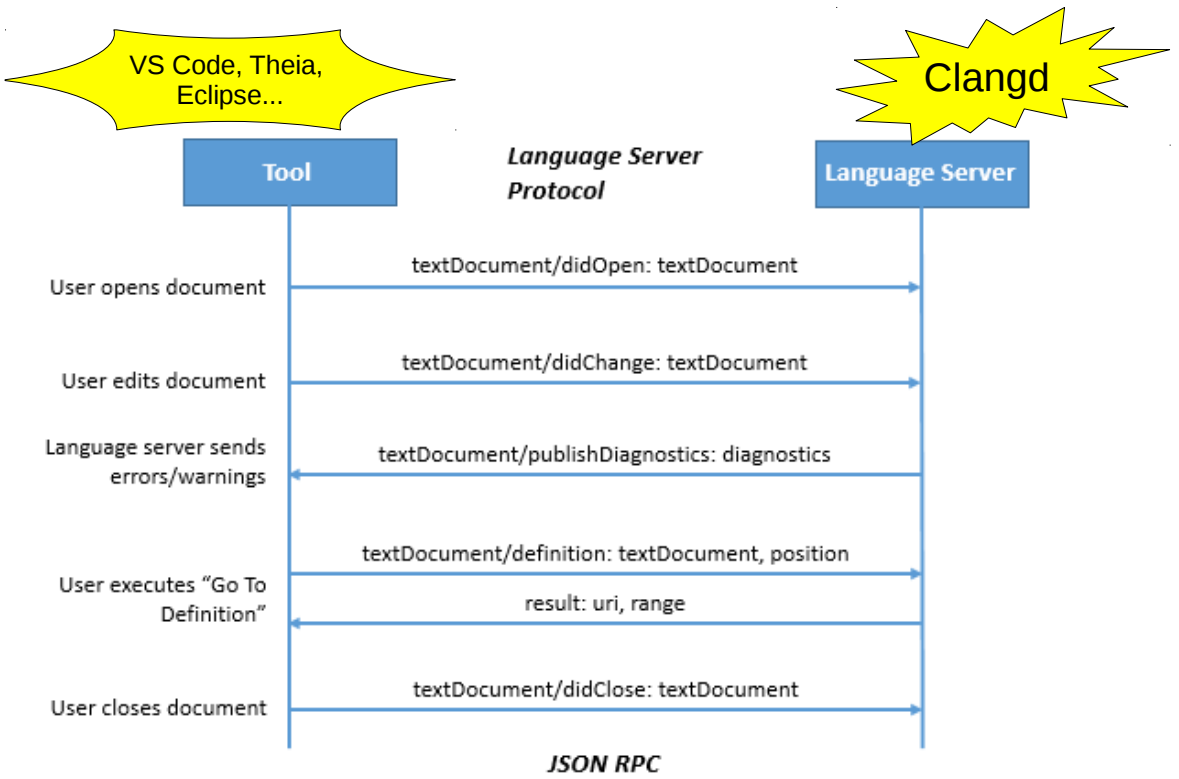
Introductions

- Marc-André Laperle
  - Software Developer at Ericsson since 2013
  - Eclipse committer for CDT (C/C++) and several other projects
  - New-ish LLVM/Clang contributor (early 2017)
  - Enthusiastic about C/C++, IDEs and tooling in general (Not a compiler expert! Yet?)

What is Clangd

- Tool in Clang “Extras”
- Implements the Language Server Protocol (**LSP**) for C/C++
  - Protocol in **JSON-RPC**
  - Code “smartness” features like code completion, find references, etc.
- Integrations with Visual Studio Code, Theia and Eclipse (and more?)

What is Clangd



Original image source:
https://github.com/Microsoft/language-server-protocol/blob/master/protocol.md

Current status

C/C++ Editor feature	LSP	Clangd
Formatting	Yes	Yes
Completion	Yes	Yes
Diagnostics	Yes	Yes
Fix-its	Yes	Yes
Go to Definition	Yes	Yes*
Source hover	Yes	Soon
Signature Help	Yes	Yes
Find References	Yes	No, index needed
Document Highlights	Yes	Soon
Rename	Yes	No, index needed
Code Lens	Yes	No, index needed

* Actually just goes to declarations or definitions visible in the same unit

Current status

C/C++ Editor feature	LSP	Clangd
Syntax and Semantic Coloring	No	No
Code folding	No	No
Call hierarchy	No	No, index needed
Type hierarchy	No	No, index needed
Organize Includes	No	No
Quick Assist	No	No
Extract Local Variable	No	No
Extract Function/Method	No	No
Hide Method	No	No
Implement Method	No	No, index needed
Gen. Getters/Setters	No	No, index needed

What’s new

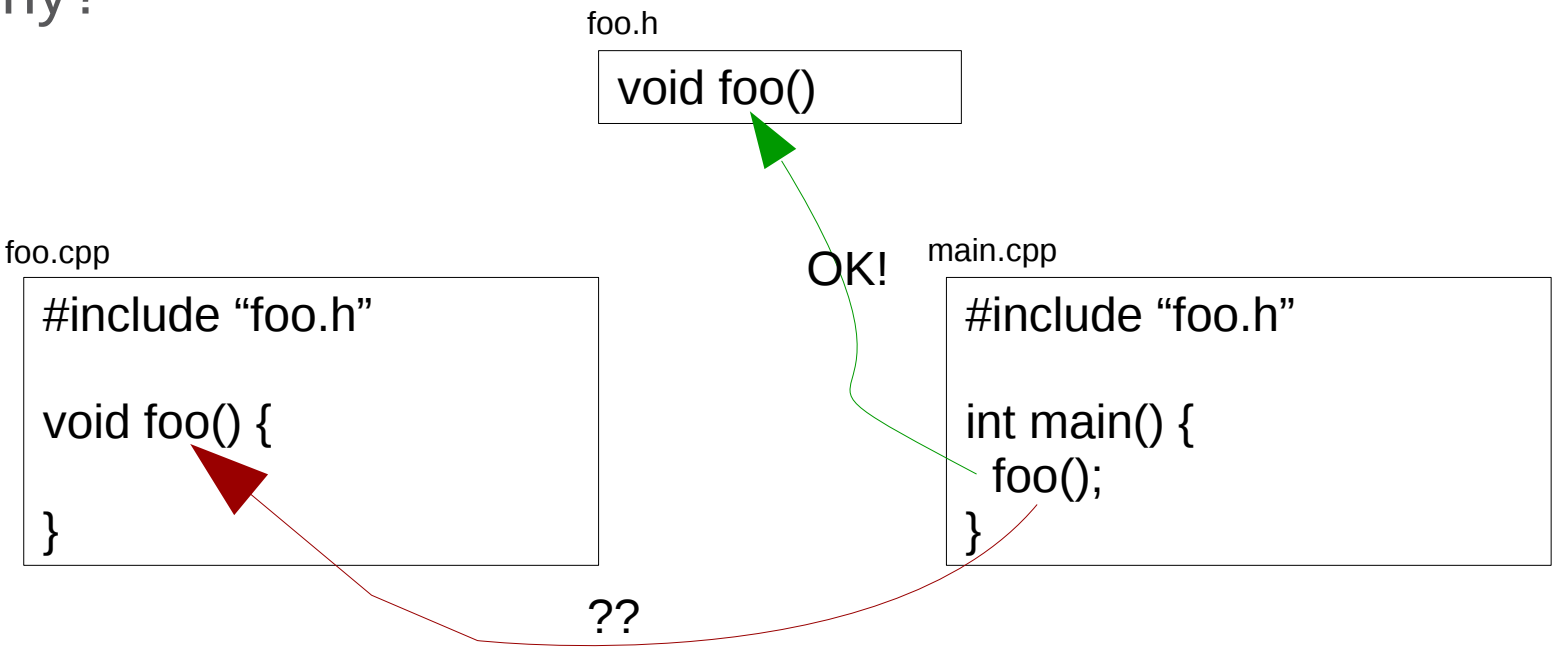
- Code Completion
- Go to Declaration
- Signature Help
- Switch between source/header
- More command-line options (resource-dir, compilation database)

What’s new

- Performance improvements (multiple thread workers, preamble management, etc)
- A lot of bug fixes and clean-ups!
- Growing community!

Indexing

› Why?



First prototypes

- 1) Outside LLVM/Clang community
  - Before Clangd
  - Separate Github repo
  - Experimentations with code completion and RecursiveASTVisitor
  - libjsoncpp + hand-written JSON-RPC handling

- 2) “Migration” to Clangd
  - Same repo (but in a branch) as other Clang “extra” tools: more collaboration!
  - LLVM’s YAML parser (for JSON), hand-written JSON-RPC handling
  - “Open Declaration” feature, no index
  - No Clang/Index code reused, only “RecursiveASTVisitor”

- 3) Introducing persistence
  - “RecursiveASTVisitor” still used to find “cursor” location
  - Clang/Index’s IndexDataConsumer used to feed the index
  - Simple, Very Slow Index file format (linear dump)
  - No definition in headers, only function definitions

“SVSI” file format (Simple Very Slow Index)

	Num of files
+ 4	File path length
+ 4	File path
+ filepath length	Num of symbols
	Symbol name length
+ 4	Symbol name
+ 4	Symbol loc start
+ name length	Symbol loc end
+ 4	...More symbols
+ 4	...More files

- Read through the whole file to read a symbol a the end!
- On source file change => Rebuild the whole file!

- 4) Better persistence and code reuse:
  - IndexDataConsumer used for both “cursor” location and feeding the index
  - ClangdIndexDataStorage, BTree introduced
  - No definition in headers, only function definitions
  - Source file changes handled (but not headers)
  - Dependents of headers stored in index (can now use source file’s compilation database when opening header file)

ClangdIndexDataStorage

- Malloc-like interface to writing in a file
- Stores raw bytes, ints, string and pointers (to other locations in the file)
- Inspired from Eclipse CDT’s database

ClangdIndexDataStorage

File layout

0	File version
4	Linked list to free blocks of 8 bytes
8	Linked list to free blocks of 16 bytes
...	
2048	Linked list to free blocks of 4096B
2052	“User” data (Blocks)

Data block

0	Block size
2..size	Any “user” data

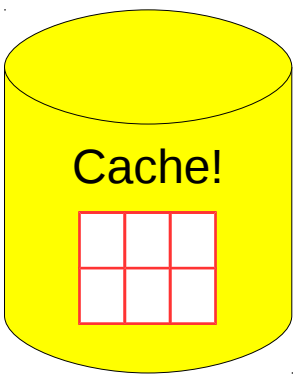
Free block

0	Free block size
2	Pointer to next free block of same size
6..size	Unused (until it becomes a data block)

ClangdIndexDataStorage

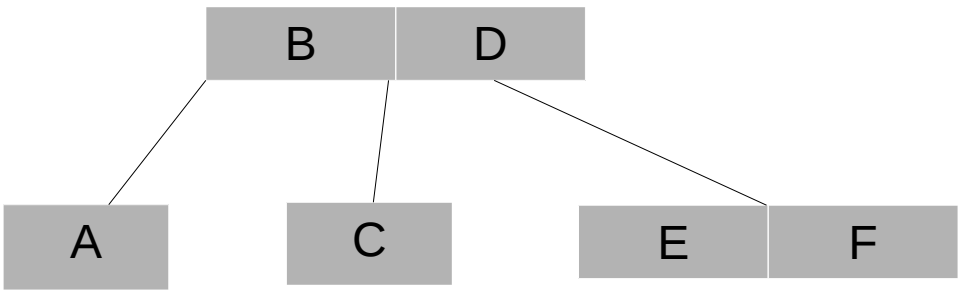
4K pieces

Data block			Free block	
Free		Data	Data	Free	Data
Data	Free		Data	
Data				
Data	Data	Free	Data	Data
Data	Free	Free		Data



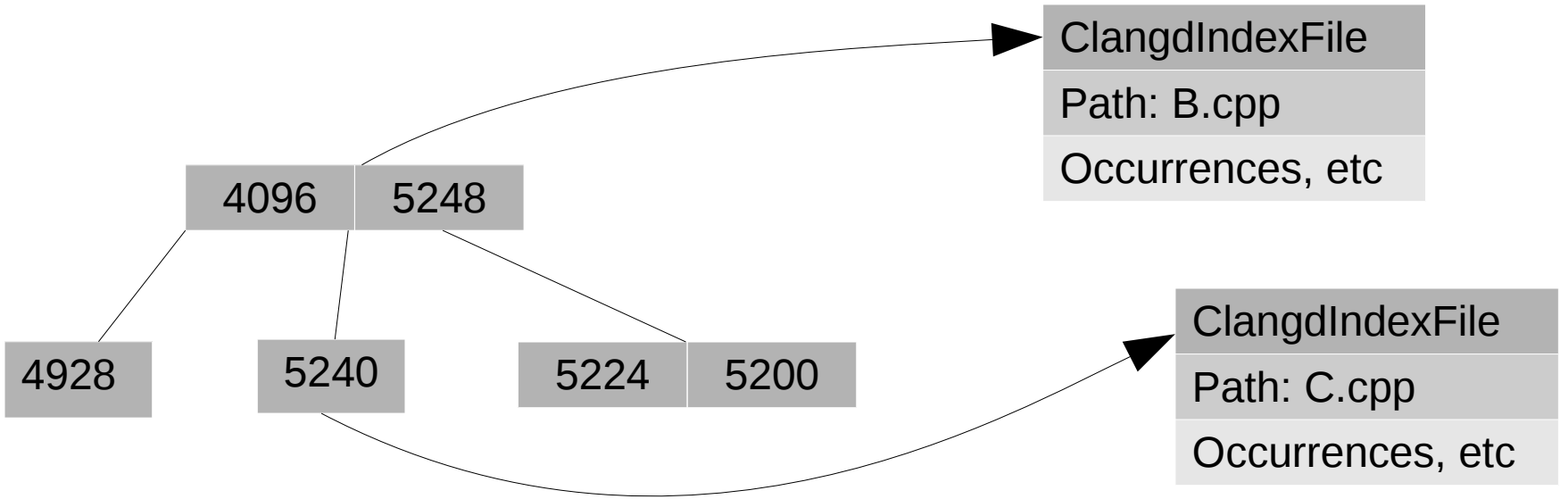
BTree

- Tree with nodes containing multiple children
- Balanced
- Logarithmic insertion, search, deletion



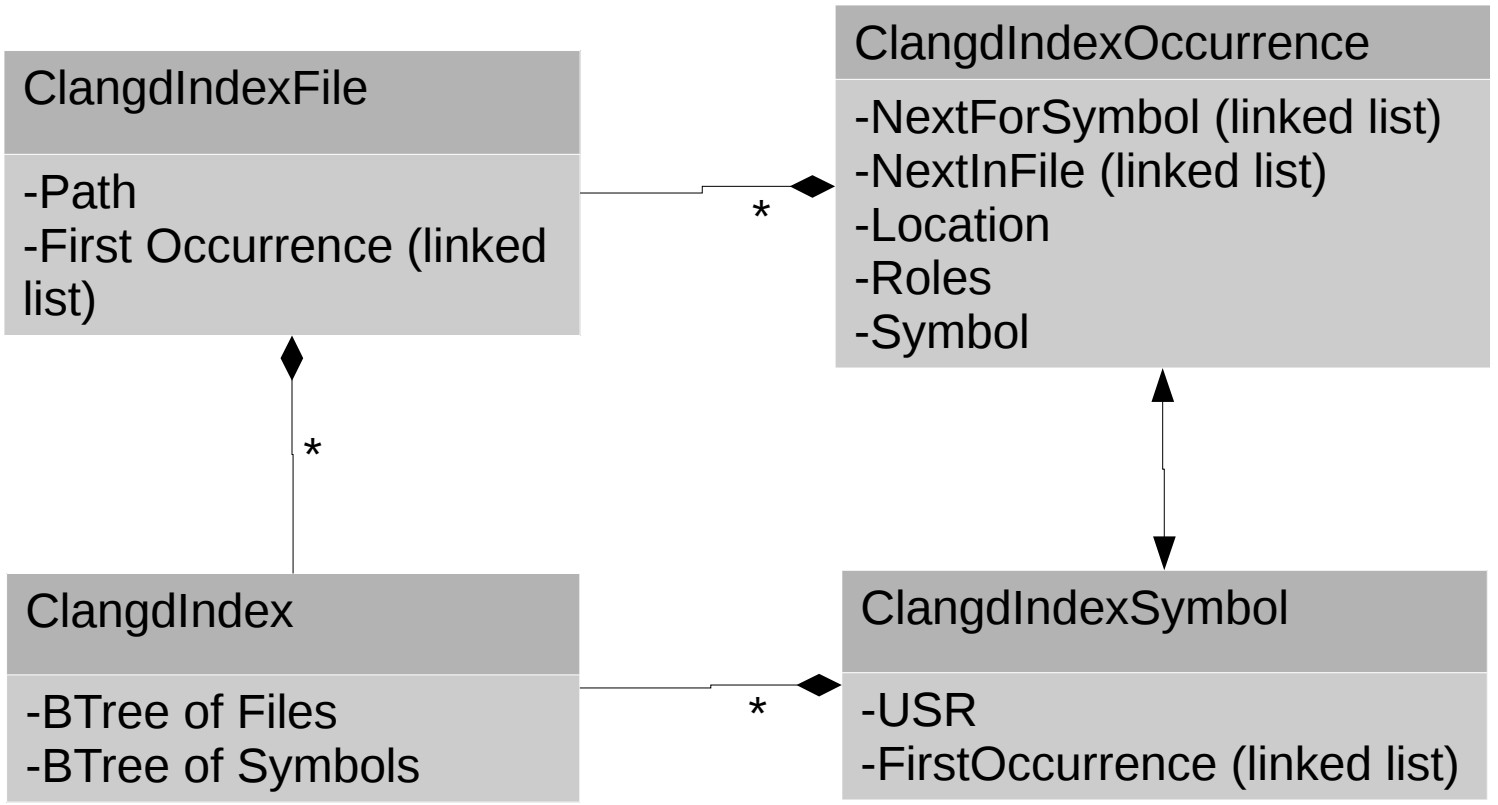
BTree

- Keys are pointers to data in ClangdIndexDataStorage



- 5) Richer model, file handling
  - Time stamps for all files (source and headers) recorded
  - Dependent source files re-indexed on header change
  - Declarations, Definitions, References all in index
  - Index model closer to “Index-While-Building” proposal
  - Working “Open definition” and “Find references”
  - … crashes when indexing Clangd

Model


ClangdIndexFile
-Path
-First Occurrence (linked list)
ClangdIndexOccurrence
-NextForSymbol (linked list)
-NextInFile (linked list)
-Location
-Roles
-Symbol
*
*
ClangdIndex
-BTree of Files
-BTree of Symbols
ClangdIndexSymbol
-USR
-FirstOccurrence (linked list)
*

Demo

(VS Code, Theia, Eclipse)

Challenges

Lack of header caching

- Use Precompiled headers? Pretokenized Headers?
- How to use cached header information when included in a different context? (different order of includes, etc)
- Heuristics for using cached header even in different context?
- Split indexing in multiple phases? (Skip bodies at first, etc)
- Reuse “Modules” feature in Clang

Challenges

File event handling

- Server vs client file watching
- Watching a large number of files (beyond the inotify max)
- Re-indexing everything when a top level header changes

Collaborations

Indexing

- “Index-While-Building” feature in Xcode 9 could be reused. Reuse the index store (instead of ClangdIndexStorage). [1] (Also presented today)
- Move some indexing logic to Clang/Index instead of Clangd, for others to reuse
- Make indexing extensible enough so that other tools can add information to the index (Clang Static Analyzer)
- Use liblmdb, for symbol mapping, similar to Xcode

[1] https://docs.google.com/document/d/1cH2sTpgSnJZCkZtJl1aY-rzy4uGPcrI-6RrUpdATO2Q/

Collaborations

Refactoring

- Specifying LSP well enough to express the refactorings (with options, other languages, etc)
- Reuse refactoring framework introduced with Xcode 9 [1]

  (Also presented today)
- Contribute more refactorings to Clang
- Add code generations (Generate Getters/Setters, Constructors, etc)
- Quick assist support, possibly reusing refactoring components

[1] http://lists.llvm.org/pipermail/cfe-dev/2017-June/054286.html

References

- Clangd: https://clang.llvm.org/extra/clangd.html
- Language Server Protocol: https://github.com/Microsoft/language-server-protocol
- Clang mailing list: https://lists.llvm.org/mailman/listinfo/cfe-dev
- Github branch hosting indexing prototype: https://github.com/MarkZ3/clang-tools-extra/tree/IndexFunctionsPrototype
- Theia IDE: https://github.com/theia-ide/theia

Q&A