C++ now

Glean

Code Indexing at Meta

Michael Park

2024

Agenda

1. Motivation
2. Architecture Overview
3. C++ Macros: Understanding Locations
4. Angle: Schema/Query Language
5. C++ Macro Navigation

Motivation

rocksdb.cpp

```
11
12  #include <rocksdb/db.h>
13
14  namespace facebook {
15  namespace glean {
16  namespace rocks {
17
18  std::shared_ptr<Cache> newCache(size_t capacity) {
19    return rocksdb::NewLRUCache(capacity);
20  }
21
22  std::unique_ptr<Container> open(
23      const std::string& path,
24      Mode mode,
25      bool cache_index_and_filter_blocks,
26      folly::Optional<std::shared_ptr<Cache>> cache) {
27    return std::make_unique<impl::ContainerImpl>(
28        path, mode, cache_index_and_filter_blocks, std::move(cache));
29  }
30
31  }
32  }
```

Code from https://github.com/facebookincubator/Glean/blob/main/glean/rocksdb/rocksdb.cpp

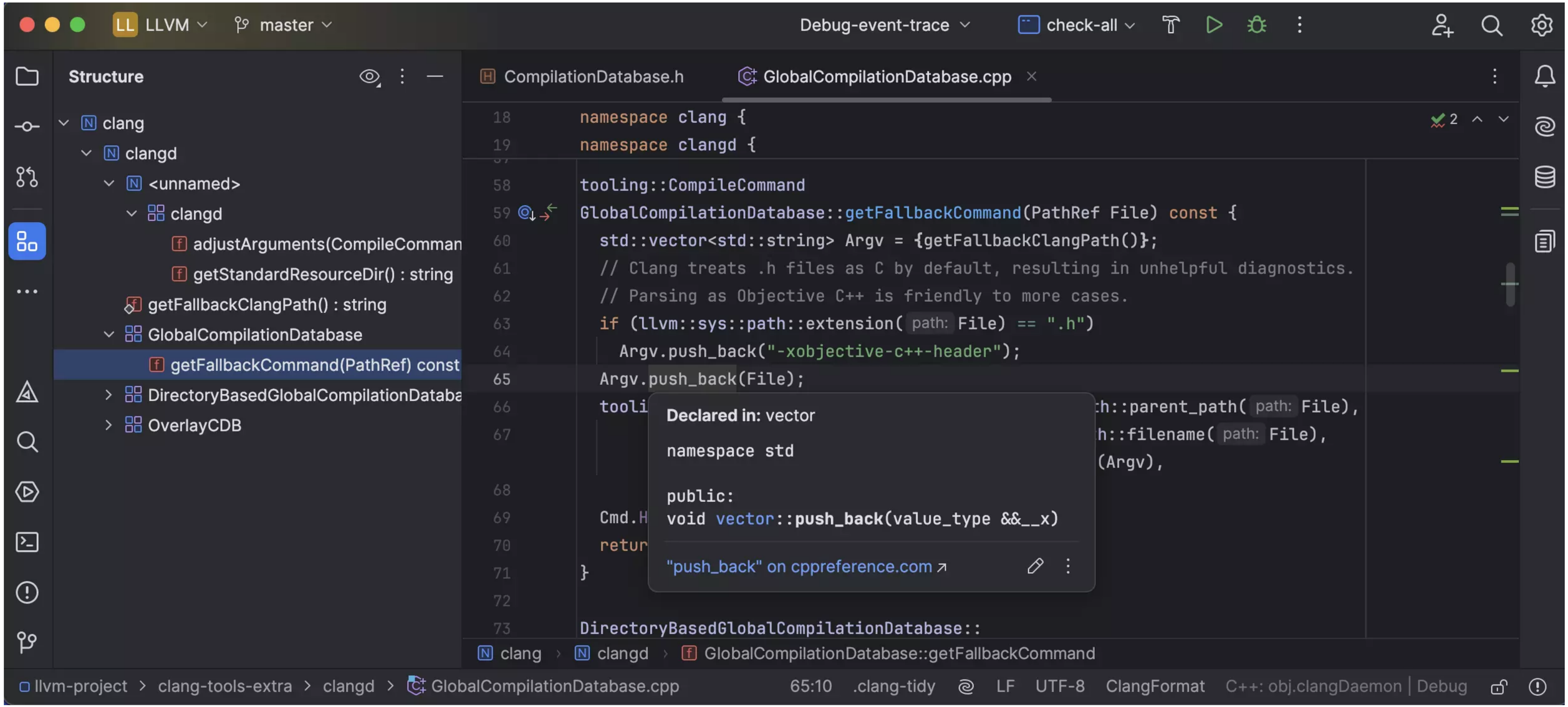


Image from https://www.jetbrains.com/clion/

◆ stripFromIncludePath()

QCString stripFromIncludePath (const **QCString** & path)

strip part of *path* if it matches one of the paths in the **Config_getList(INCLUDE_PATH)** list

Definition at line **337** of file **util.cpp**.

```
338 {
339   return stripFromPath(path,Config_getList(STRIP_FROM_INC_PATH));
340 }
```

References **Config_getList**, and **stripFromPath()**.

Referenced by **addIncludeFile()**, and **findFileDef()**.

Image from https://www.doxygen.nl/index.html#crossreferencing

Many Developer Tools

- IDEs (e.g. VSCode, CLion, IntelliJ)
- Code Browsers (e.g. codebrowser.dev)
- Documentation Generation (e.g. Doxygen)
- Symbol Search
- Code Analysis (e.g. Linters, Dead Code Detection)
- API Usage Tracking (for Library Maintainers)
- and more!

Various Limitations

- Many tools are language-specific
- Scaling to large codebases often challenging
- The collected data is not easily available for general queries
  - "Find variables named `size`"
  - "Hey documentation tool, what percentage of our functions were documented?"

Scaling Up

- Support **many** languages
- Scale to **large** codebases
- Power **many** developer tools
- Serve **complex** queries
- Extensible to collect **more** detailed data

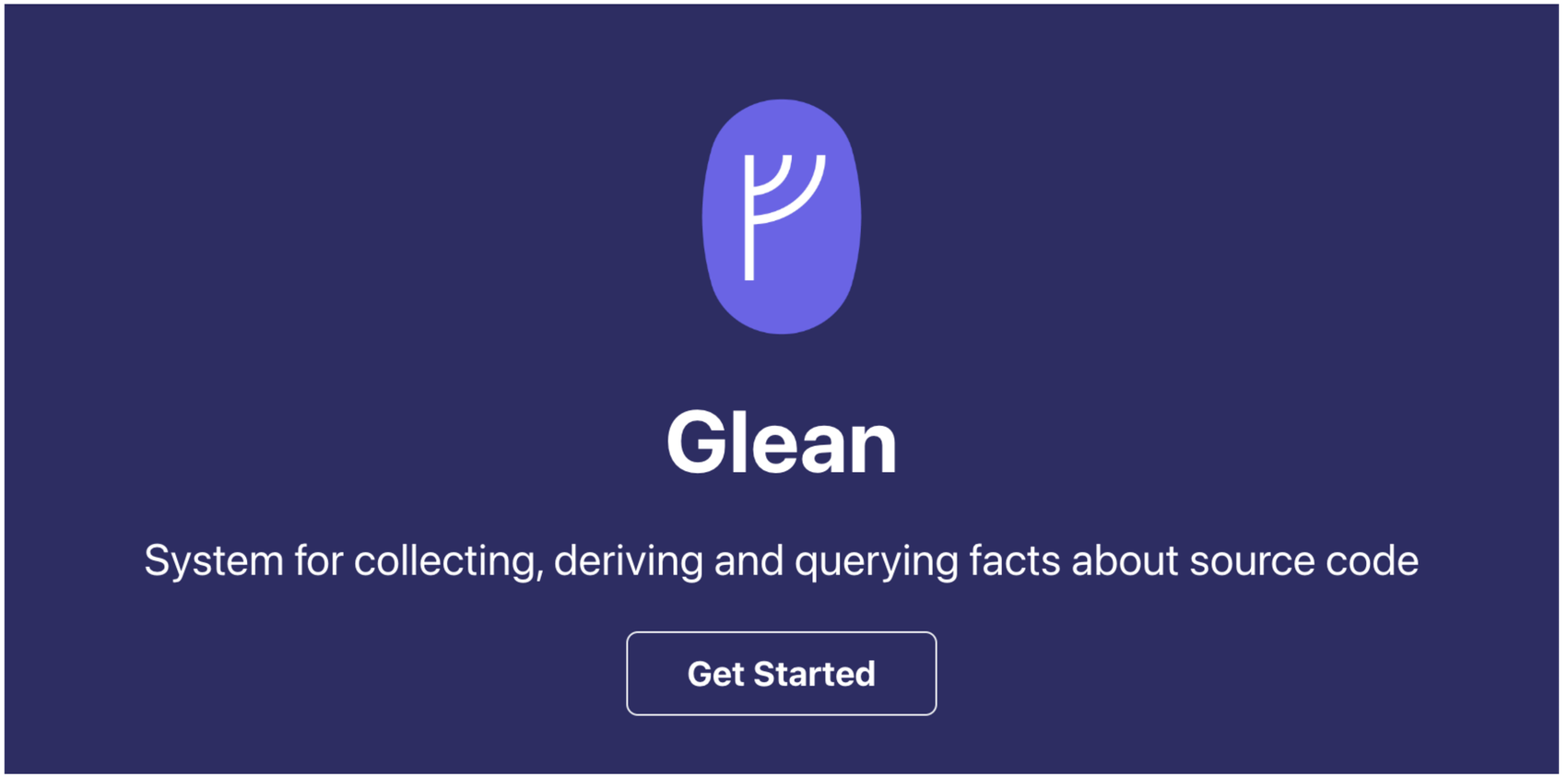
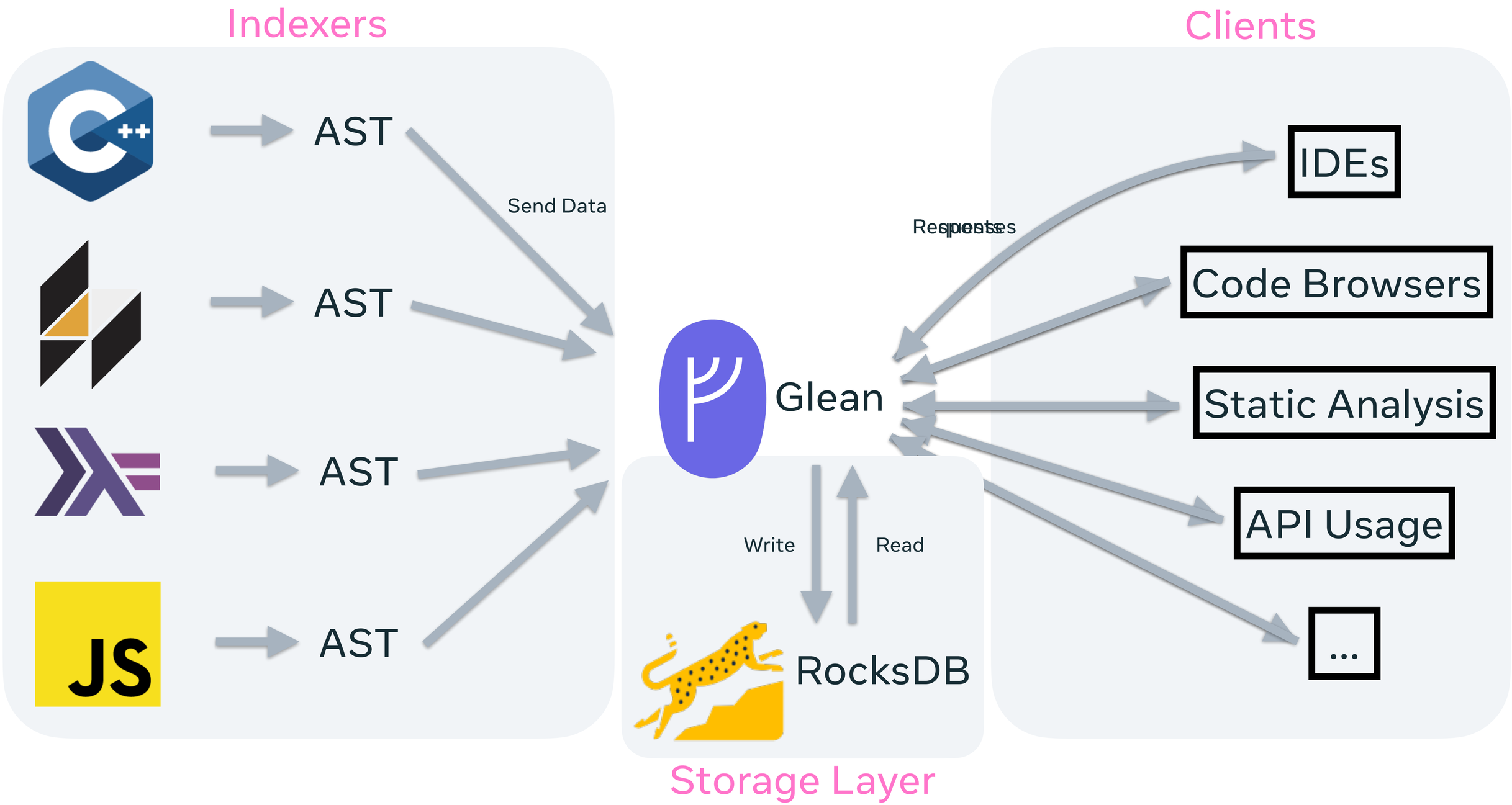


Image from https://glean.software

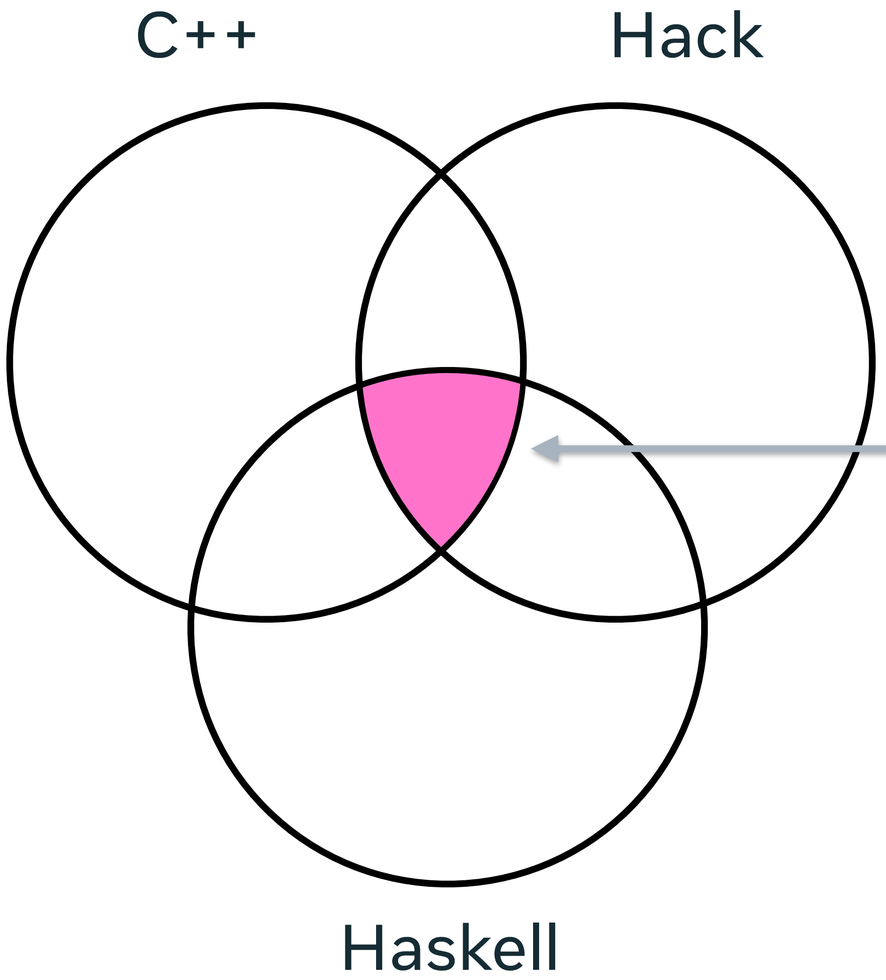
Architecture Overview


Indexers
AST
AST
AST
AST
Send Data
Glean
Write
Read
RocksDB
Storage Layer
Requests
Clients
IDEs
Code Browsers
Static Analysis
API Usage
...

Defining the Schema

- Glean does not enforce a single schema
- Each language has their own nuances
  - Even something as universal as variable declarations vary!
    e.g. In C++: variable templates, `constexpr`, initialization used
- Different clients need different data
  - Static analysis needs precise information, but
    code navigation can still be useful with heuristics

Least Common Denominator

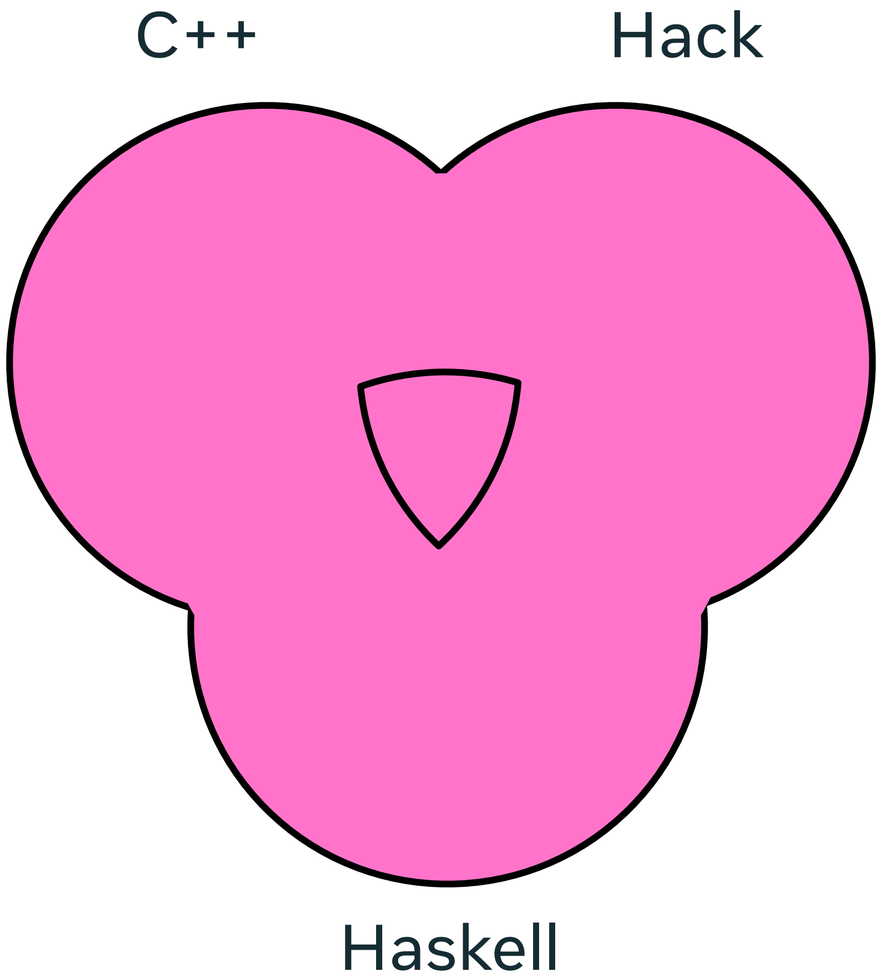


Dead Include Removal for C++

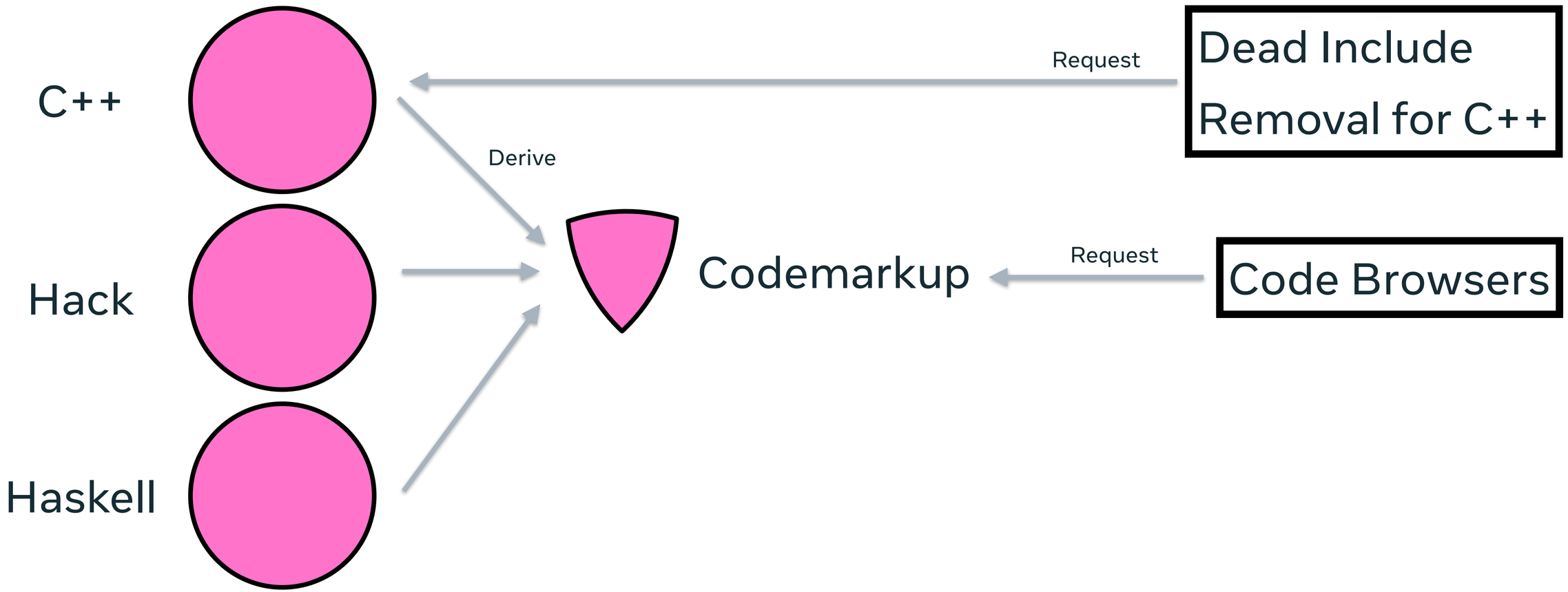
Request

Code Browsers

Union of all Languages



Multiple Schemas



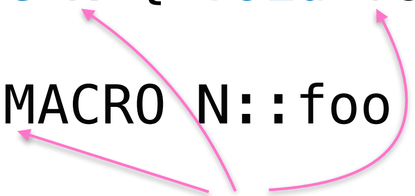
Example: Static Analysis vs Code Navigation

```
namespace N { void foo() {} }

#define MACRO N::foo

void bar() { MACRO(); }
```

```
namespace N { void foo() {} }

#define MACRO N::foo

void bar() { MACRO(); }
```

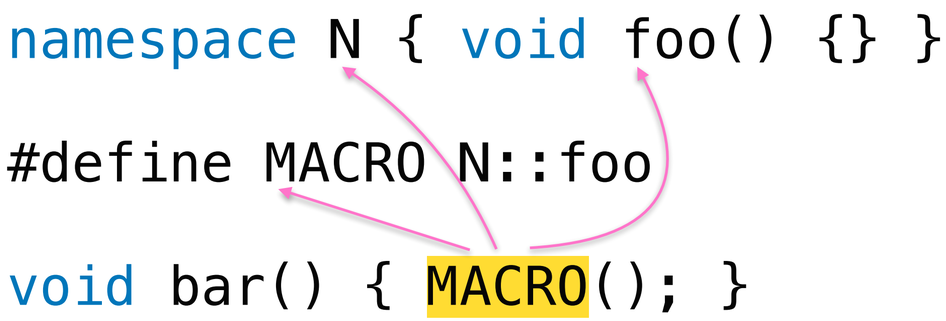
Example: Static Analysis vs Code Navigation

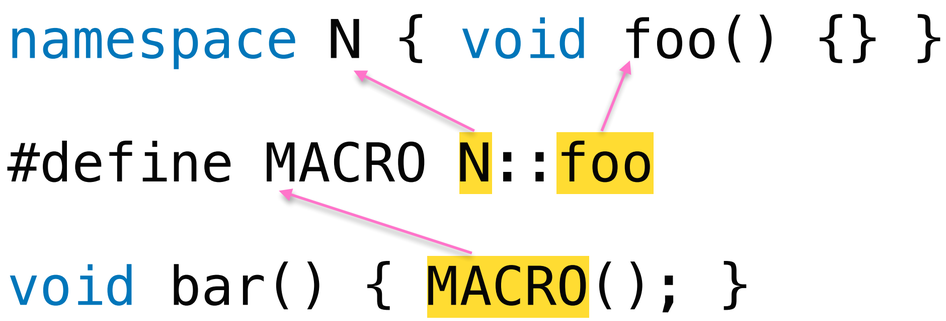
```
namespace N { void foo() {} }

#define MACRO N::foo

void bar() { MACRO(); }
```



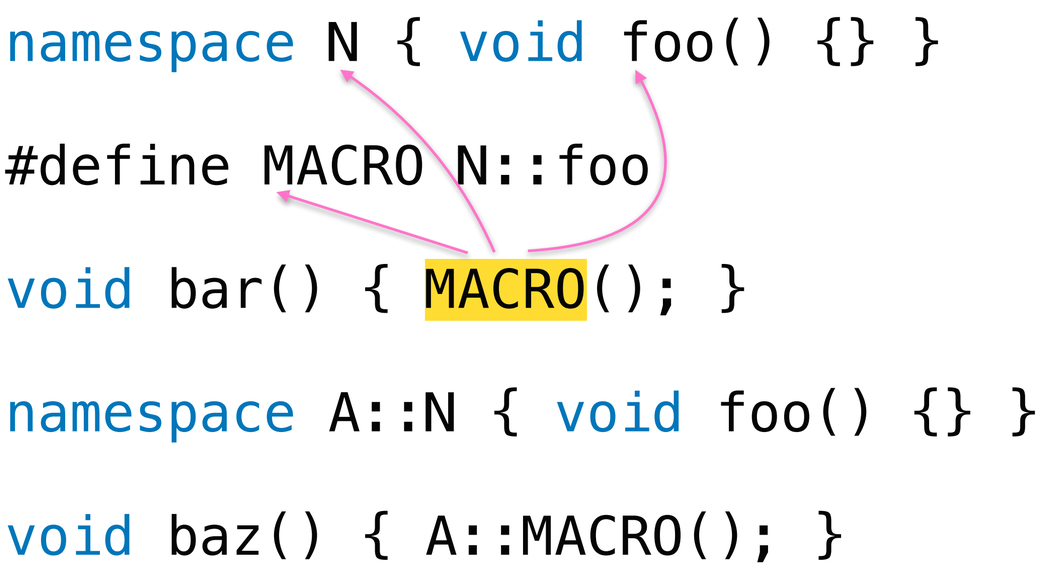
```
namespace N { void foo() {} }

#define MACRO N::foo

void bar() { MACRO(); }
```

Example: Static Analysis vs Code Navigation

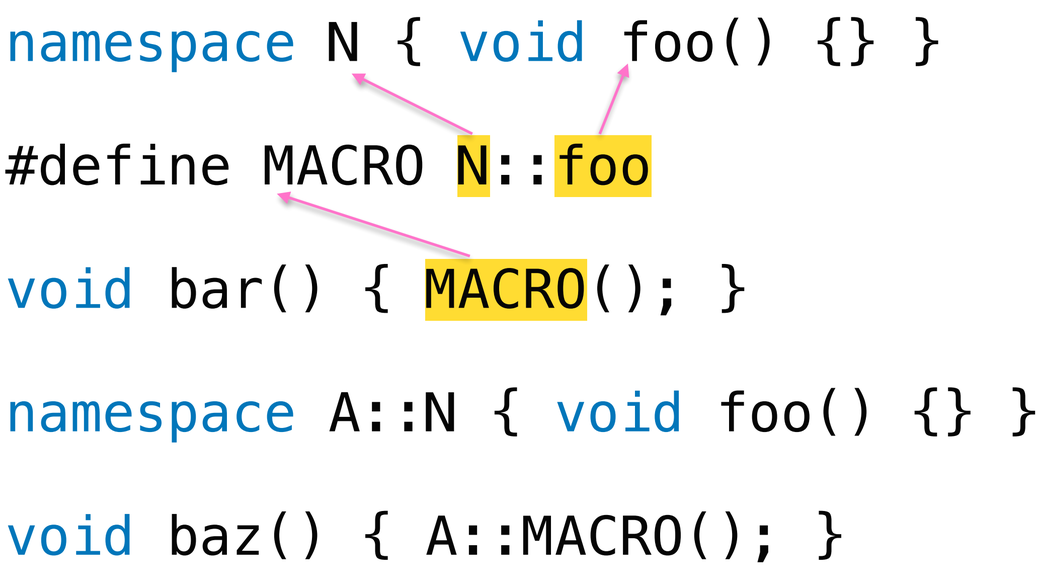




Example: Static Analysis vs Code Navigation



```
namespace N { void foo() {} }

#define MACRO N::foo

void bar() { MACRO(); }

namespace A::N { void foo() {} }

void baz() { A::MACRO(); }
```



```
namespace N { void foo() {} }

#define MACRO N::foo

void bar() { MACRO(); }

namespace A::N { void foo() {} }

void baz() { A::MACRO(); }
```

Example: Static Analysis vs Code Navigation

```
namespace N { void foo() {} }

#define MACRO N::foo

void bar() { MACRO(); }

namespace A::N { void foo() {} }

void baz() { A::MACRO(); }
```

```
namespace N { void foo() {} }

#define MACRO N::foo

void bar() { MACRO(); }

namespace A::N { void foo() {} }

void baz() { A::MACRO(); }
```

Example: Static Analysis vs Code Navigation

```
namespace N { void foo() {} }

#define MACRO N::foo

void bar() { MACRO(); }

namespace A::N { void foo() {} }

void baz() { A::MACRO(); }
```

```
namespace N { void foo() {} }

#define MACRO N::foo

void bar() { MACRO(); }

namespace A::N { void foo() {} }

void baz() { A::MACRO(); }
```

C++ Macros: Understanding Locations

stats.cpp

```
260
261 void init_stats(stat_t* stats) {
262 #define STAT(_name, _type, _aggregate, _data_assignment) \
263   {                                                      \
264     stat_t& s = stats[_name##_stat];                     \
265     s.name = #_name;                                     \
266     s.group = GROUP;                                     \
267     s.type = _type;                                      \
268     s.aggregate = _aggregate;                            \
269     s.data _data_assignment;                             \
270   }
271 #define STUI(name, value, agg) STAT(name, stat_uint64, agg, .uint64 = value)
272 #define STSI(name, value, agg) STAT(name, stat_int64, agg, .int64 = value)
273 #define STSS(name, value, agg) \
274   STAT(name, stat_string, agg, .string = (char*)value)
275 #define EXTERNAL_STAT(name) \
```

Code from https://github.com/facebook/mcrouter/blob/main/mcrouter/stats.cpp#L261-L275

Types of Locations

```
void foo() {}

#define MACRO foo

#define REF(x) x
```

```
void f() {
  foo();

  MACRO();

  REF(foo)();
}
```

Types of Locations

```
void foo() {}
     ^^^ file

#define MACRO foo


#define REF(x) x
```

```
void f() {
  foo();
  ^^^ file
  MACRO();

  REF(foo)();

}
```

Types of Locations

```
void foo() {}

#define MACRO foo

#define REF(x) x
```

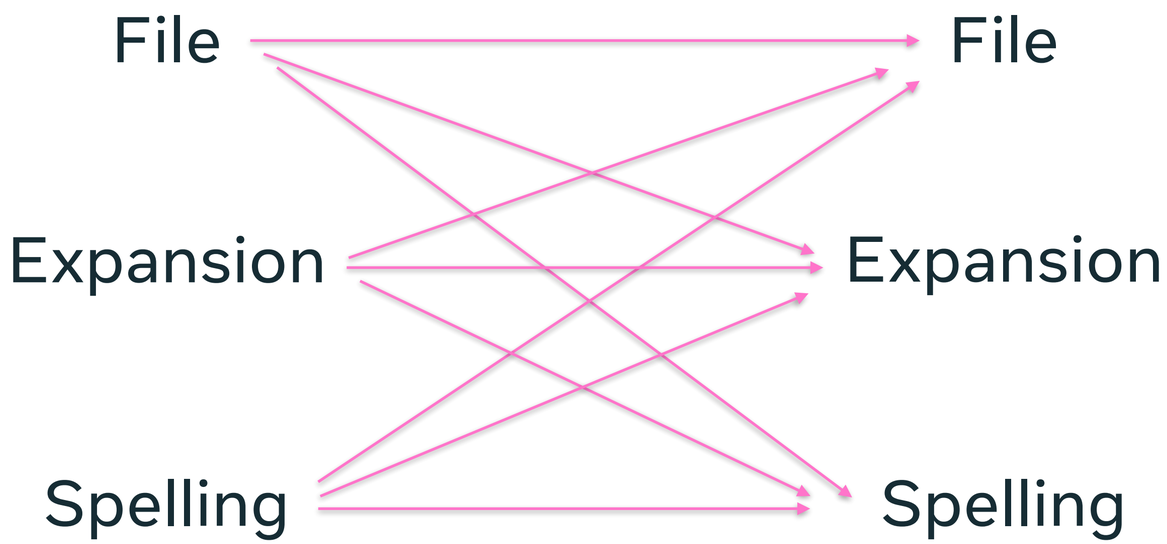
```
void f() {
  foo();

  MACRO();
  ^^^^^ expansion
  REF(foo)();
  ^^^^^^^^ expansion
}
```

Types of Locations

```
void foo() {}


#define MACRO foo
              ^^^ spelling

#define REF(x) x
```

```
void f() {
  foo();

  MACRO();

  REF(foo)();
      ^^^ spelling
}
```

Cross References



Common Examples

```
void foo() {}

void baz() {
  foo();
}
```

File to File

File to File

```
#define DEFINE(name) \
  void name() {}

DEFINE(bar)

void baz() {
  bar();
}
```

File to Spelling
(Macro Argument)

File to Spelling

Common Examples

```
int foo() {}

#define MACRO_FOO foo


void baz() {
  MACRO_FOO();
  assert(foo > 0);
}
```

Spelling (Macro Body) to File

Spelling (Macro Argument) to File

Spelling to File

```
#define UNWRAP(x)                  \
  ({                               \
    auto result = x;               \
    if (!result) {                 \
      return result.error(); \
    }                              \
    result.value();                \
  })
```

Spelling to Spelling

Spelling to Spelling

Angle: Schema/Query Language

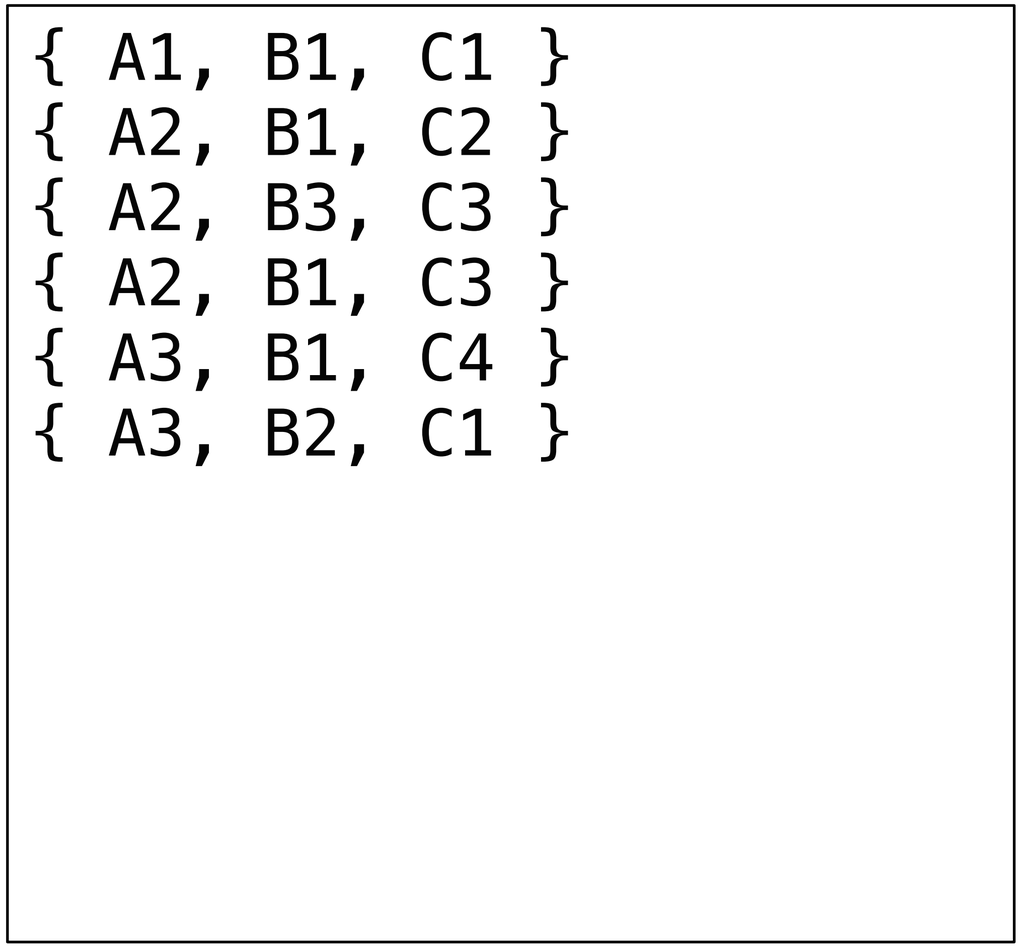
Predicates and Facts

```
schema example {
  predicate Name : string

  predicate QualifiedName : {
    name : Name,
    namespace : Name,
  }
}
```

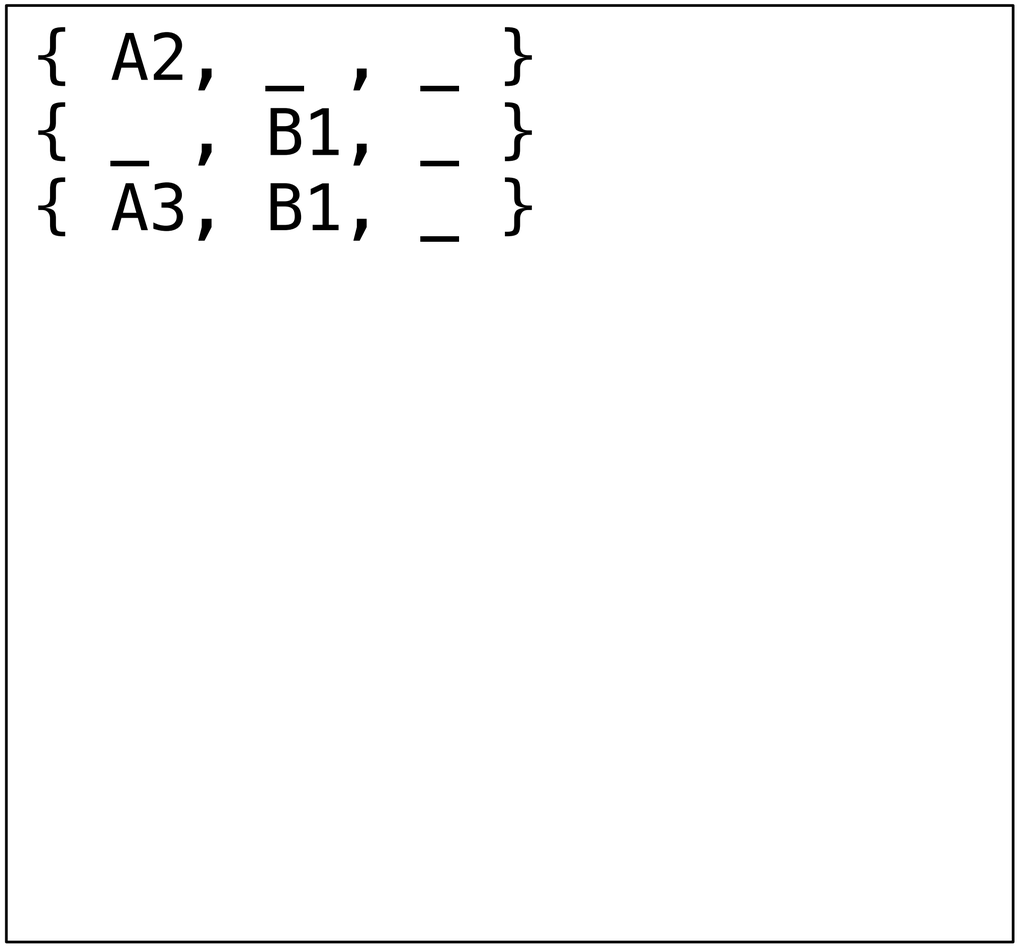
example.angle

- Predicates are types of facts
- Each fact is assigned an ID
- Unique facts are stored exactly once
- Facts are sorted by prefix for efficient queries
- Queries generate facts in a sequence

Facts Ordering Example

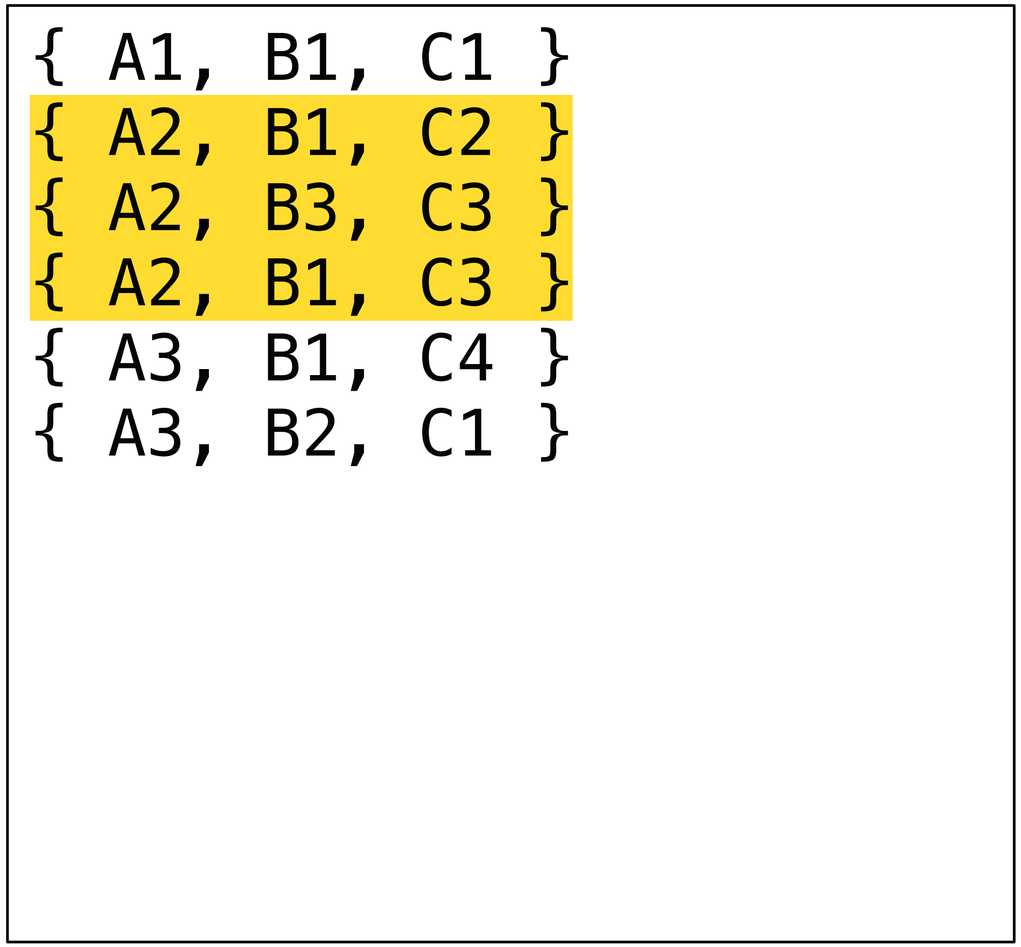


Database

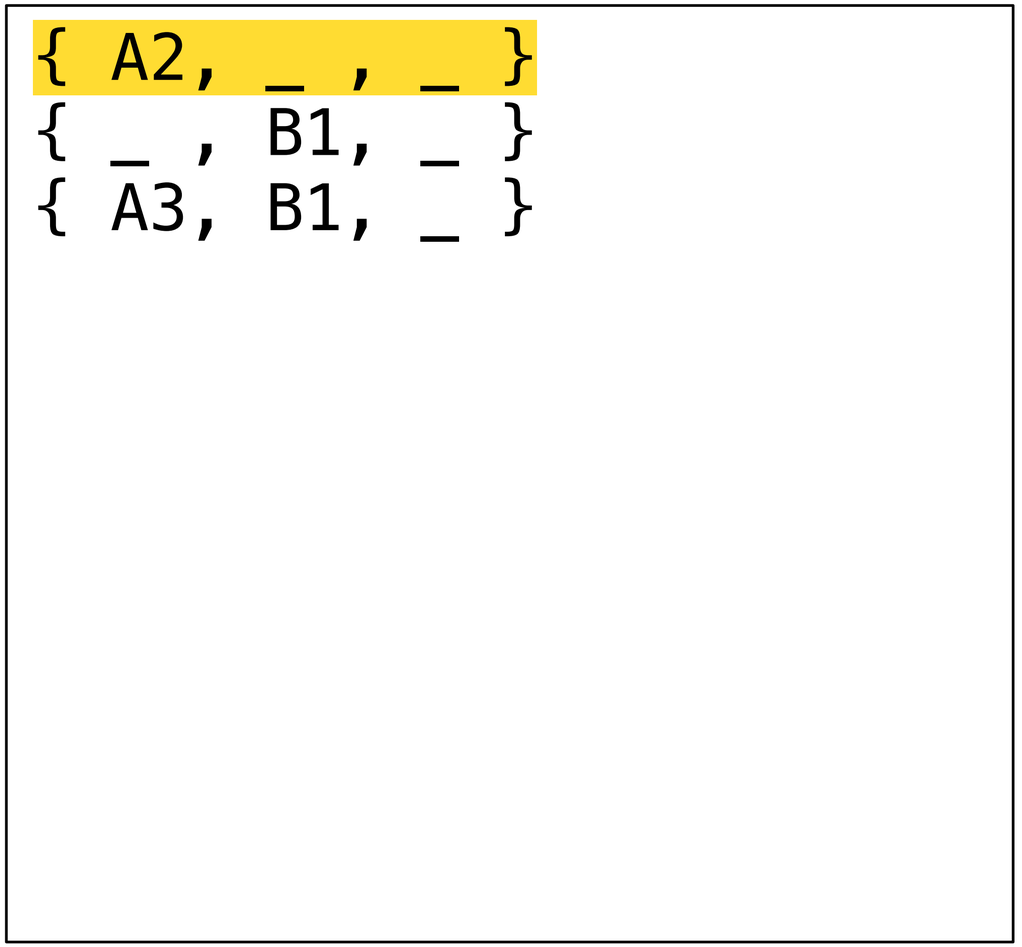


Queries

Facts Ordering Example

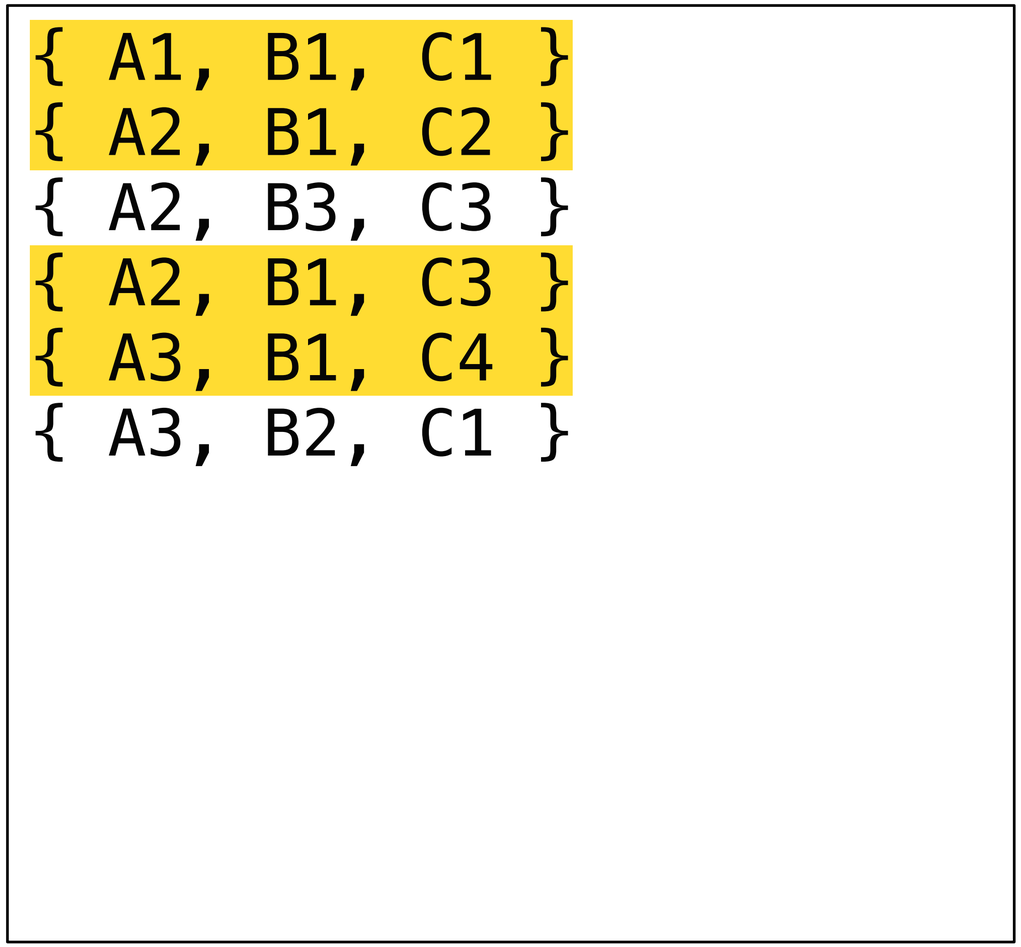


Database

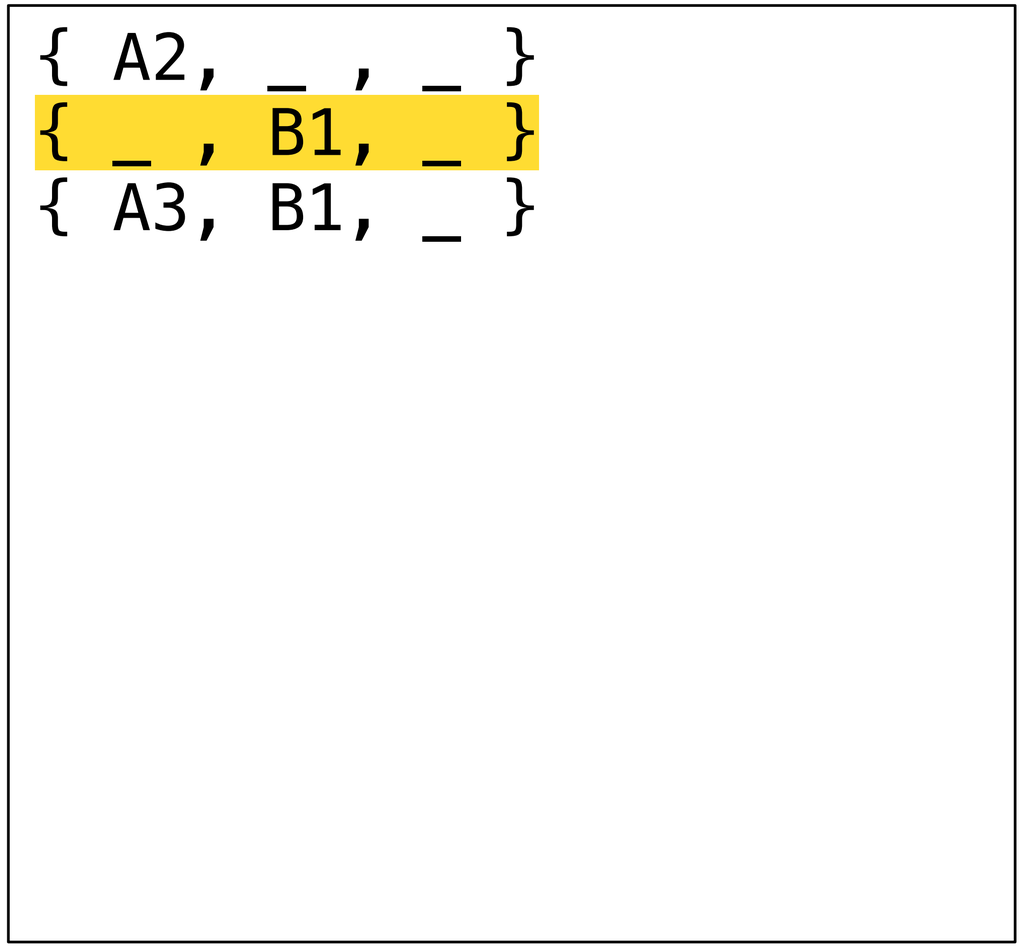


Queries

Facts Ordering Example

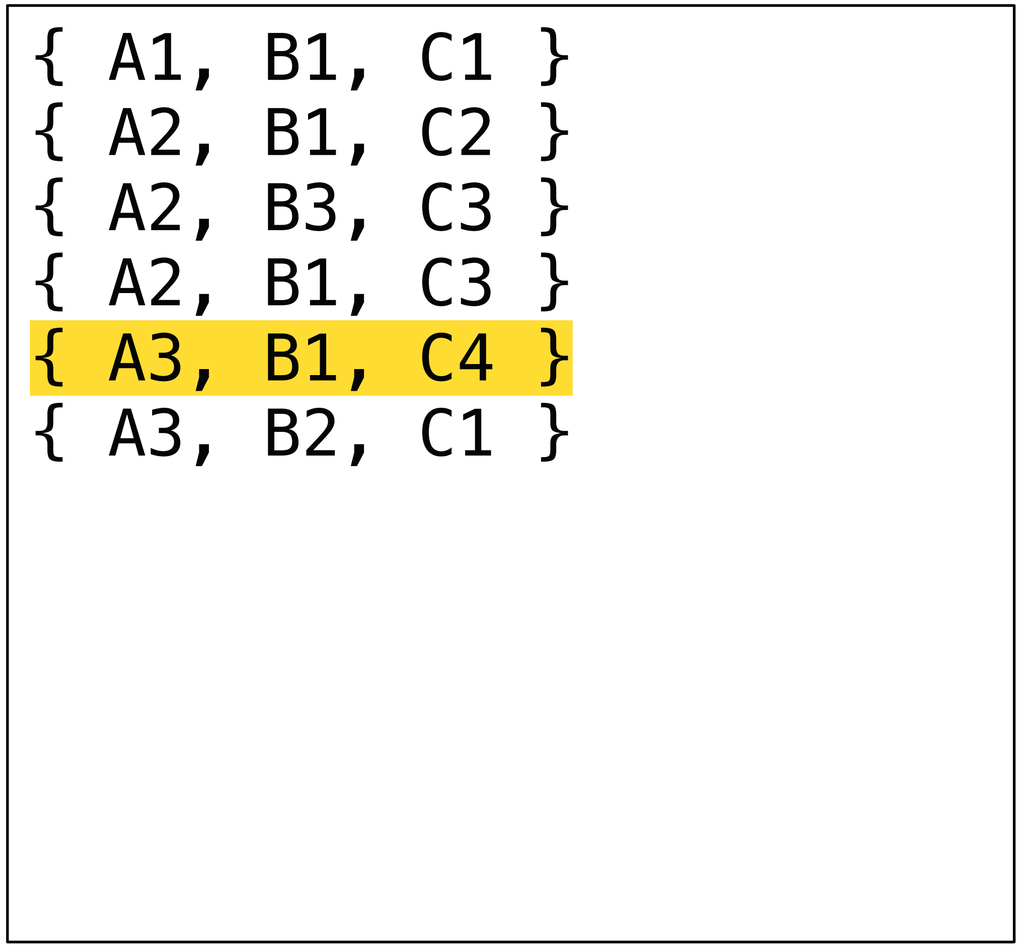


Database

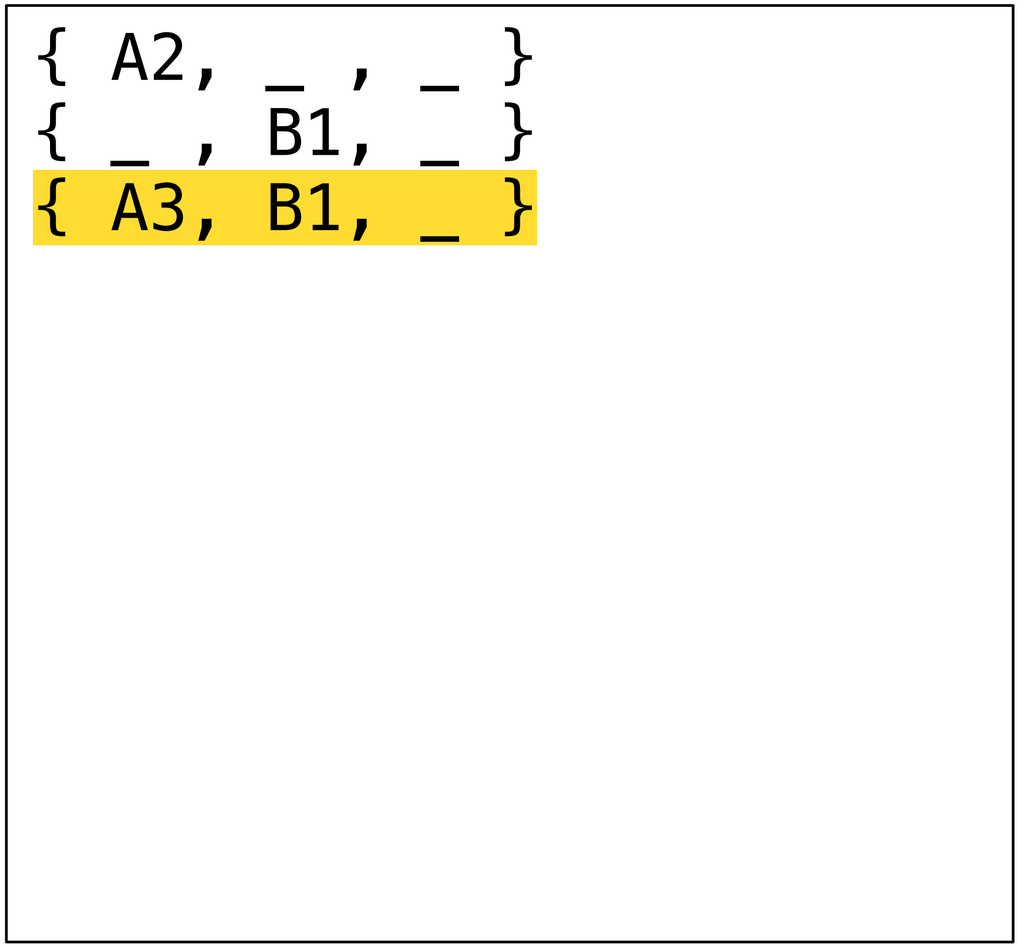


Queries

Facts Ordering Example



Database



Queries

Predicates and Facts

```
schema example {
  predicate Name : string

  predicate QualifiedName : {
    name : Name,
    namespace : Name,
  }
}
```

example.angle

Predicates and Facts

```
schema example {
  predicate Name : string

  predicate QualifiedName : {
    name : Name,
    namespace : Name,
  }
}
```

example.angle

```
#include <glean/lang/clang/schema.h>

bool VisitNamespaceDecl(
    const clang::NamespaceDecl *decl) {
  db.fact<example::Name>(decl->getName());
  return true;
}
```

Predicates and Facts

```
schema example {
  predicate Name : string

  predicate QualifiedName : {
    name : Name,
    namespace : Name,
  }
}
```

example.angle

```
#include <glean/lang/clang/schema.h>

bool VisitNamespaceDecl(
    const clang::NamespaceDecl *decl) {
  db.fact<example::Name>(decl->getName());
  return true;
}
```

Predicates and Facts

```
1   namespace N1 {
2     class Widget {};
3   }
4
5   namespace N2 {
6     struct Widget {};
7   }
8
9   N1::Widget w1;
10  N1::Widget w2;
```

widget.cpp

```
example.Name { id: 0, key: "N1" }
example.Name { id: 1, key: "Widget" }
example.Name { id: 2, key: "N2" }
```

Predicates and Facts

```
1   namespace N1 {
2     class Widget {};
3   }
4
5   namespace N2 {
6     struct Widget {};
7   }
8
9   N1::Widget w1;
10  N1::Widget w2;
```

widget.cpp

```
example.Name { id: 0, key: "N1" }
example.Name { id: 1, key: "Widget" }
example.Name { id: 2, key: "N2" }

example.QualifiedName { // N1::Widget
  id: 3,
  key: {
    name: { id: 1 },
    namespace: { id: 0 },
  },
}

example.QualifiedName { // N2::Widget
  id: 4,
  key: {
    name: { id: 1 },
    namespace: { id: 2 },
  },
}
```

Predicates and Facts

```
1   namespace N1 {
2     class Widget {};
3   }
4
5   namespace N2 {
6     struct Widget {};
7   }
8
9   N1::Widget w1;
10  N1::Widget w2;
```

widget.cpp

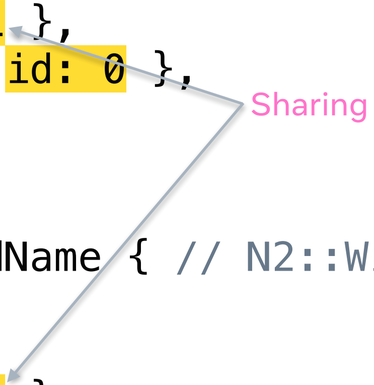
```
example.Name { id: 0, key: "N1" }
example.Name { id: 1, key: "Widget" }
example.Name { id: 2, key: "N2" }

example.QualifiedName { // N1::Widget
  id: 3,
  key: {
    name: { id: 1 },
    namespace: { id: 0 },
  },
}

example.QualifiedName { // N2::Widget
  id: 4,
  key: {
    name: { id: 1 },
    namespace: { id: 2 },
  },
}
```

Predicates and Facts

```
1   namespace N1 {
2     class Widget {};
3   }
4
5   namespace N2 {
6     struct Widget {};
7   }
8
9   N1::Widget w1;
10  N1::Widget w2;
```

widget.cpp

```
example.Name { id: 0, key: "N1" }
example.Name { id: 1, key: "Widget" }
example.Name { id: 2, key: "N2" }

example.QualifiedName { // N1::Widget
  id: 3,
  key: {
    name: { id: 1 },
    namespace: { id: 0 },
  },
}

example.QualifiedName { // N2::Widget
  id: 4,
  key: {
    name: { id: 1 },
    namespace: { id: 2 },
  },
}
```



Glean Interactive Shell

```
$ glean shell --db example
example> example.Name _
{ id: 0, key: "N1" }
{ id: 1, key: "Widget" }
{ id: 2, key: "N2" }

example> example.QualifiedName { name = "Widget" }
{ id: 3, key: { name: { id: 1, key = "Widget" }, namespace: { id: 0, key = "N1" } } }
{ id: 4, key: { name: { id: 1, key = "Widget" }, namespace: { id: 2, key = "N2" } } }

example> NS where example.QualifiedName { name = "Widget", namespace = NS }
{ id: 0, key = "N1" }
{ id: 2, key = "N2" }

example> example.Name "N1" | "N2"
{ id: 0, key: "N1" }
{ id: 2, key: "N2" }
```

Predicates and Facts

```
schema example {
  type Kind = enum {
    Struct | Class | Union
  };

  predicate RecordDeclaration : {
    name : QualifiedName,
    kind : Kind,
    file : string,
    line : nat,
  }
}
```

example.angle

```
#include <glean/lang/clang/schema.h>

bool VisitCXXRecordDecl(
    const clang::CXXRecordDecl *decl) {
  auto name = db.fact<example::Name>(decl->getName());
  auto namespace_name = db.fact<example::Name>(
    /* decl->getDeclContext() */);

  auto qname = db.fact<example::QualifiedName>(
    name, namespace_name);

  auto kind = [&] {
    switch (decl->getTagKind()) {
      case clang::TTK_Struct: return Kind::Struct;
      case clang::TTK_Class: return Kind::Class;
      case clang::TTK_Union: return Kind::Union;
      default: std::unreachable();
    }
  }();

  db.fact<example::RecordDeclaration>(
    qname, kind, /* decl->getSourceRange() */);

  return true;
}
```

Predicates and Facts

```
schema example {
  type Kind = enum {
    Struct | Class | Union
  };

  predicate RecordDeclaration : {
    name : QualifiedName,
    kind : Kind,
    file : string,
    line : nat,
  }
}
```

example.angle

```
#include <glean/lang/clang/schema.h>

bool VisitCXXRecordDecl(
    const clang::CXXRecordDecl *decl) {
  auto name = db.fact<example::Name>(decl->getName());
  auto namespace_name = db.fact<example::Name>(
    /* decl->getDeclContext() */);

  auto qname = db.fact<example::QualifiedName>(
    name, namespace_name);

  auto kind = [&] {
    switch (decl->getTagKind()) {
      case clang::TTK_Struct: return Kind::Struct;
      case clang::TTK_Class: return Kind::Class;
      case clang::TTK_Union: return Kind::Union;
      default: std::unreachable();
    }
  }();

  db.fact<example::RecordDeclaration>(
    qname, kind, /* decl->getSourceRange() */);

  return true;
}
```

Predicates and Facts

```
schema example {
  type Kind = enum {
    Struct | Class | Union
  };

  predicate RecordDeclaration : {
    name : QualifiedName,
    kind : Kind,
    file : string,
    line : nat,
  }
}
```

example.angle

```
#include <glean/lang/clang/schema.h>

bool VisitCXXRecordDecl(
    const clang::CXXRecordDecl *decl) {
  auto name = db.fact<example::Name>(decl->getName());
  auto namespace_name = db.fact<example::Name>(
    /* decl->getDeclContext() */);

  auto qname = db.fact<example::QualifiedName>(
    name, namespace_name);

  auto kind = [&] {
    switch (decl->getTagKind()) {
      case clang::TTK_Struct: return Kind::Struct;
      case clang::TTK_Class: return Kind::Class;
      case clang::TTK_Union: return Kind::Union;
      default: std::unreachable();
    }
  }();

  db.fact<example::RecordDeclaration>(
    qname, kind, /* decl->getSourceRange() */);

  return true;
}
```

Predicates and Facts

```
schema example {
  type Kind = enum {
    Struct | Class | Union
  };

  predicate RecordDeclaration : {
    name : QualifiedName,
    kind : Kind,
    file : string,
    line : nat,
  }
}
```

example.angle

```
#include <glean/lang/clang/schema.h>

bool VisitCXXRecordDecl(
    const clang::CXXRecordDecl *decl) {
  auto name = db.fact<example::Name>(decl->getName());
  auto namespace_name = db.fact<example::Name>(
    /* decl->getDeclContext() */);

  auto qname = db.fact<example::QualifiedName>(
    name, namespace_name);

  auto kind = [&] {
    switch (decl->getTagKind()) {
      case clang::TTK_Struct: return Kind::Struct;
      case clang::TTK_Class: return Kind::Class;
      case clang::TTK_Union: return Kind::Union;
      default: std::unreachable();
    }
  }();

  db.fact<example::RecordDeclaration>(
    qname, kind, /* decl->getSourceRange() */);

  return true;
}
```

Predicates and Facts

```
1   namespace N1 {
2     class Widget {};
3   }
4
5   namespace N2 {
6     struct Widget {};
7   }
8
9   N1::Widget w1;
10  N1::Widget w2;
```

widget.cpp

```
example.RecordDeclaration {
  id: 5,
  key: {
    name: { id: 3 }, // N1::Widget
    kind: Class,
    file: "widget.cpp",
    line: 2
  },
}

example.RecordDeclaration {
  id: 6,
  key: {
    name: { id: 4 }, // N2::Widget
    kind: Struct,
    file: "widget.cpp",
    line: 6
  },
}
```

Predicates and Facts

```
1   namespace N1 {
2     class Widget {};
3   }
4
5   namespace N2 {
6     struct Widget {};
7   }
8
9   N1::Widget w1;
10  N1::Widget w2;
```

widget.cpp

```
example.RecordDeclaration {
  id: 5,
  key: {
    name: { id: 3 }, // N1::Widget
    kind: Class,
    file: "widget.cpp",
    line: 2
  },
}

example.RecordDeclaration {
  id: 6,
  key: {
    name: { id: 4 }, // N2::Widget
    kind: Struct,
    file: "widget.cpp",
    line: 6
  },
}
```

Predicates and Facts

```
1   namespace N1 {
2     class Widget {};
3   }
4
5   namespace N2 {
6     struct Widget {};
7   }
8
9   N1::Widget w1;
10  N1::Widget w2;
```

widget.cpp

```
example.RecordDeclaration {
  id: 5,
  key: {
    name: { id: 3 }, // N1::Widget
    kind: Class,
    file: "widget.cpp",
    line: 2
  },
}

example.RecordDeclaration {
  id: 6,
  key: {
    name: { id: 4 }, // N2::Widget
    kind: Struct,
    file: "widget.cpp",
    line: 6
  },
}
```

Predicates and Facts

```
schema example {
  type XRef = {
    target : RecordDeclaration,
    lines : [nat],
  }

  predicate FileXRefs : {
    file : string,
    xrefs : [XRef],
  }
}
```

example.angle

```
example.FileXRefs {
  id: 7,
  key: {
    file: "widget.cpp",
    xrefs: [
      {
        target: { id: 5 }, // N1::Widget
        lines: [9, 10],
      }
    ]
  }
}
```

Glean Interactive Shell

```
$ glean shell --db example
example> example.FileXRefs { file = "widget.cpp" }
{
  id: 7,
  key: {
    file: "widget.cpp",
    xrefs: [
      {
        target: {
          id: 5,
          key: {
            name: { id: 3, key: { name: { id: 1, key: "Widget" }, namespace: { id: 0, key: "N1" } } },
            kind: Class,
            file: "widget.cpp",
            line: 2,
          },
        },
        lines: [9, 10],
      }
    ]
  }
}
```

Derived Predicates

Derivation Example: `FileToFile`

```
schema example {
  type XRef = {
    target : RecordDeclaration,
    lines : [nat],
  }

  predicate FileXRefs : {
    file : string,
    xrefs : [XRef],
  }
}
```

example.angle

Derivation Example: `FileToFile`

```
schema example {
  predicate FileToFile : {
    from : string,
    to : string,
  }









}
```

example.angle

```
$ glean shell --db example
example> { From, To }
         where
         FileXRefs {
           file = From, xrefs = XRefs
         };
         { target = { file = To } } = XRefs[..]
{
  id: 11,
  key: {
    tuplefield0: "main.cpp",
    tuplefield1: "foo.h",
  }
}
{
  id: 12,
  key: {
    tuplefield0: "main.cpp",
    tuplefield1: "bar.h",
  }
}
```

Derivation Example: `FileToFile`

```
schema example {
  predicate FileToFile : {
    from : string,
    to : string,
  }
  { From, To }
  where
  FileXRefs {
    file = From, xrefs = XRefs
  };
  { target = { file = To } } = XRefs[..]
}
```

example.angle

```
$ glean shell --db example
example>
```

```
{
  id: 11,
  key: {
    tuplefield0: "main.cpp",
    tuplefield1: "foo.h",
  }
}
{
  id: 12,
  key: {
    tuplefield0: "main.cpp",
    tuplefield1: "bar.h",
  }
}
```

Derivation Example: `FileToFile`

```
schema example {
  predicate FileToFile : {
    from : string,
    to : string,
  }
  { From, To }
  where
  FileXRefs {
    file = From, xrefs = XRefs
  };
  { target = { file = To } } = XRefs[..]
}
```

example.angle

```
$ glean shell --db example
example> example.FileToFile _
```

```
{
  id: 11,
  key: {
    tuplefield0: "main.cpp",
    tuplefield1: "foo.h",
  }
}
{
  id: 12,
  key: {
    tuplefield0: "main.cpp",
    tuplefield1: "bar.h",
  }
}
```

Derivation Example: `FileToFile`

```
schema example {
  predicate FileToFile : {
    from : string,
    to : string,
  }
  { From, To }
  where
  FileXRefs {
    file = From, xrefs = XRefs
  };
  { target = { file = To } } = XRefs[..]
}
```

example.angle

```
$ glean shell --db example
example> example.FileToFile _
```

```
{
  id: 11,
  key: {
    from: "main.cpp",
    to: "foo.h",
  }
}
{
  id: 12,
  key: {
    from: "main.cpp",
    to: "bar.h",
  }
}
```

Derivation Example: `FileToFile`

```
schema example {
  predicate FileToFile : {
    from : string,
    to : string,
  }
  stored
  { From, To }
  where
  FileXRefs {
    file = From, xrefs = XRefs
  };
  { target = { file = To } } = XRefs[..]
}
```

example.angle

```
$ glean shell --db example
example> example.FileToFile _
```

```
{
  id: 11,
  key: {
    from: "main.cpp",
    to: "foo.h",
  }
}
{
  id: 12,
  key: {
    from: "main.cpp",
    to: "bar.h",
  }
}
```

Derivation Example: `FileToFile`

```
schema example {
  predicate FileToFile : {
    from : string,
    to : string,
  }
  stored
  { From, To }
  where
  FileXRefs {
    file = From, xrefs = XRefs
  };
  { target = { file = To } } = XRefs[..]
}
```

example.angle

```
$ glean shell --db example
example> example.FileToFile _
```

```
{
  id: 11,
  key: {
    from: "main.cpp",
    to: "foo.h",
  }
}
{
  id: 12,
  key: {
    from: "main.cpp",
    to: "bar.h",
  }
}
```

Derivation Example: TargetUses

```
schema example {
  predicate TargetUses : {
    target : RecordDeclaration,
    file : string,
    lines : [nat],
  }
  stored
  { Target, File, Lines }
  where
  FileXRefs {
    file = File,
    xrefs = XRefs
  };
  {
    target = Target,
    lines = Lines
  } = XRefs[..];
}
```

example.angle

```
$ glean shell --db example
example> example.TargetUses {
  target = {
    name = { name = "Widget", namespace = "N1" },
  },
}

{
  id: 14,
  key: {
    target: { id: 5 },
    file: "widget.cpp",
    lines: [9, 10],
  }
}
```

Deriving the Codemarkup Layer

```
import example
import cxx
import hack

schema codemarkup {
  predicate FileXRef {
    file : string,
    line : nat,
    to_file : string,
    to_line : nat,
  }
  { File, Line, ToFile, ToLine }
  where
  ( example.FileXRefs { file = File, xrefs = XRefs };
    { target = { file = ToFile, line = ToLine }, lines = Lines } = XRefs[..];
    Line = Lines[..];
  ) | (
    cxx.FileXRefs { … };
    cxx.PPXRefs { … };
    cxx.DeclDefnXRefs { … };
    …
  ) | (
    hack.FileXRefs { … };
    …
  )
}
```

codemarkup.angle

Deriving the Codemarkup Layer

```
import codemarkup.example
import codemarkup.cxx
import codemarkup.hack

schema codemarkup {
  predicate FileXRef {
    file : string,
    line : nat,
    to_file : string,
    to_line : nat,
  }
  { File, Line, ToFile, ToLine }
  where
  codemarkup.example.FileXRef { File, Line, ToFile, ToLine } |
  codemarkup.cxx.FileXRef { File, Line, ToFile, ToLine } |
  codemarkup.hack.FileXRef { File Line, ToFile, ToLine } |
  …
}
```

codemarkup.angle

C++ Macro Navigation

Extending the Source Location

```
type XRef = {
  target : RecordDeclaration,
  lines : [nat],




}
```

example.angle

```
struct S {};

#define MACRO S


void f() {
  S s1;


  MACRO s2;

}
```

Extending the Source Location

```
type XRef = {
  target : RecordDeclaration,
  from : {
    spans : [Span],
    expansions: [Span],
    spellings: [Span],
  },
}
```

example.angle

```
struct S {};

#define MACRO S


void f() {
  S s1;


  MACRO s2;

}
```

Extending the Source Location

```
type XRef = {
  target : RecordDeclaration,
  from : {
    spans : [Span],
    expansions: [Span],
    spellings: [Span],
  },
}
```

example.angle

```
struct S {};

#define MACRO S
               ^ spelling

void f() {
  S s1;
  ^ file

  MACRO s2;
  ^^^^^ expansion
}
```

Extending the Target Location

```
predicate RecordDeclaration : {
  name : QualifiedName,
  kind : Kind,
  file : string,
  line : nat,


}
```

```
struct S {};


#define DEFINE_S struct S {}


DEFINE_S;
```

example.angle

Extending the Target Location

```
predicate RecordDeclaration : {
  name : QualifiedName,
  kind : Kind,
  file : string,
  span : Span,


}
```

```
struct S {};


#define DEFINE_S struct S {}


DEFINE_S;
```

example.angle

Extending the Target Location

```
predicate RecordDeclaration : {
  name : QualifiedName,
  kind : Kind,
  file : string,
  span : Span,


}
```

example.angle

```
struct S {};
^^^^^^^^^^^ file

#define DEFINE_S struct S {}
```

```
DEFINE_S;
^^^^^^^^ expansion
```

Extending the Target Location

```
predicate RecordDeclaration : {
  name : QualifiedName,
  kind : Kind,
  file : string,
  span : Span,


}
```

example.angle

```
struct S {};
^^^^^^^^^^^ file

#define DEFINE_S struct S {}
                 ^^^^^^^^^^^
                 spelling???

DEFINE_S;
^^^^^^^^ expansion
```

Extending the Target Location

```
predicate RecordDeclaration : {
  name : QualifiedName,
  kind : Kind,
  file : string,
  span : Span,


}
```

example.angle

```
#define STRUCT struct
#define OPEN_BRACE {
#define CLOSE_BRACE }


#define EMPTY_BODY \
  OPEN_BRACE CLOSE_BRACE

#define DEFINE_S \
  STRUCT S EMPTY_BODY

DEFINE_S;
^^^^^^^^ expansion
```

Extending the Target Location

```
predicate RecordDeclaration : {
  name : QualifiedName,
  kind : Kind,
  file : string,
  span : Span,


}
```

example.angle

```
#define STRUCT struct
#define OPEN_BRACE {
#define CLOSE_BRACE }
^^^^^^^^^^^^^^^^^^^^^^^ spelling?

#define EMPTY_BODY \
  OPEN_BRACE CLOSE_BRACE

#define DEFINE_S \
  STRUCT S EMPTY_BODY

DEFINE_S;
^^^^^^^^ expansion
```

Extending the Target Location

```
predicate RecordDeclaration : {
  name : QualifiedName,
  kind : Kind,
  file : string,
  span : Span,
  name_file : string,
  name_span : Span,
}
```

example.angle

```
struct S {};

#define DEFINE_S struct S {}

DEFINE_S;
```

Extending the Target Location

```
predicate RecordDeclaration : {
  name : QualifiedName,
  kind : Kind,
  file : string,
  span : Span,
  name_file : string,
  name_span : Span,
}
```

example.angle

```
struct S {};
       ^ file

#define DEFINE_S struct S {}
                        ^
                   spelling



DEFINE_S;
```

stats.cpp

```
261  void init_stats(stat_t* stats) {
262  #define STAT(_name, _type, _aggregate, _data_assignment) \
263    {                                                      \
264      stat_t& s = stats[_name##_stat];                     \
265      s.name = #_name;                                     \
266      s.group = GROUP;                                     \
267      s.type = _type;                                      \
268      s.aggregate = _aggregate;                            \
269      s.data _data_assignment;                             \
270    }
271  #define STUI(name, value, agg) STAT(name, stat_uint64, agg, .uint64 = value)
272  #define STSI(name, value, agg) STAT(name, stat_int64, agg, .int64 = value)
273  #define STSS(name, value, agg) \
274    STAT(name, stat_string, agg, .string = (char*)value)
275  #define EXTERNAL_STAT(name) \
276    {}
```

Code from https://github.com/facebook/mcrouter/blob/main/mcrouter/stats.cpp#L261-L276

Still Lots To Do!

- Templates are indexed, but implicit template instantiations are currently not.

```
struct S {
  void foo() {}
};

void bar(auto x) {
  x.foo();
}

bar(S{});
```

Still Lots To Do!

- Templates are indexed, but implicit template instantiations are currently not.
- Dealing with local edits is challenging.
  - Better integration with ClangD? Can `libIncremental` help?
- Support for various platforms and environments.
  - Windows: Index files that can only build on Windows.
    Code navigation of those files on a Linux machine!
- More tools!
  - Build time insights, Help migration to Modules, Detect ODR violations?

Thank you!

Michael Park
Software Engineer

Meta

LSP (Language Server Protocol)

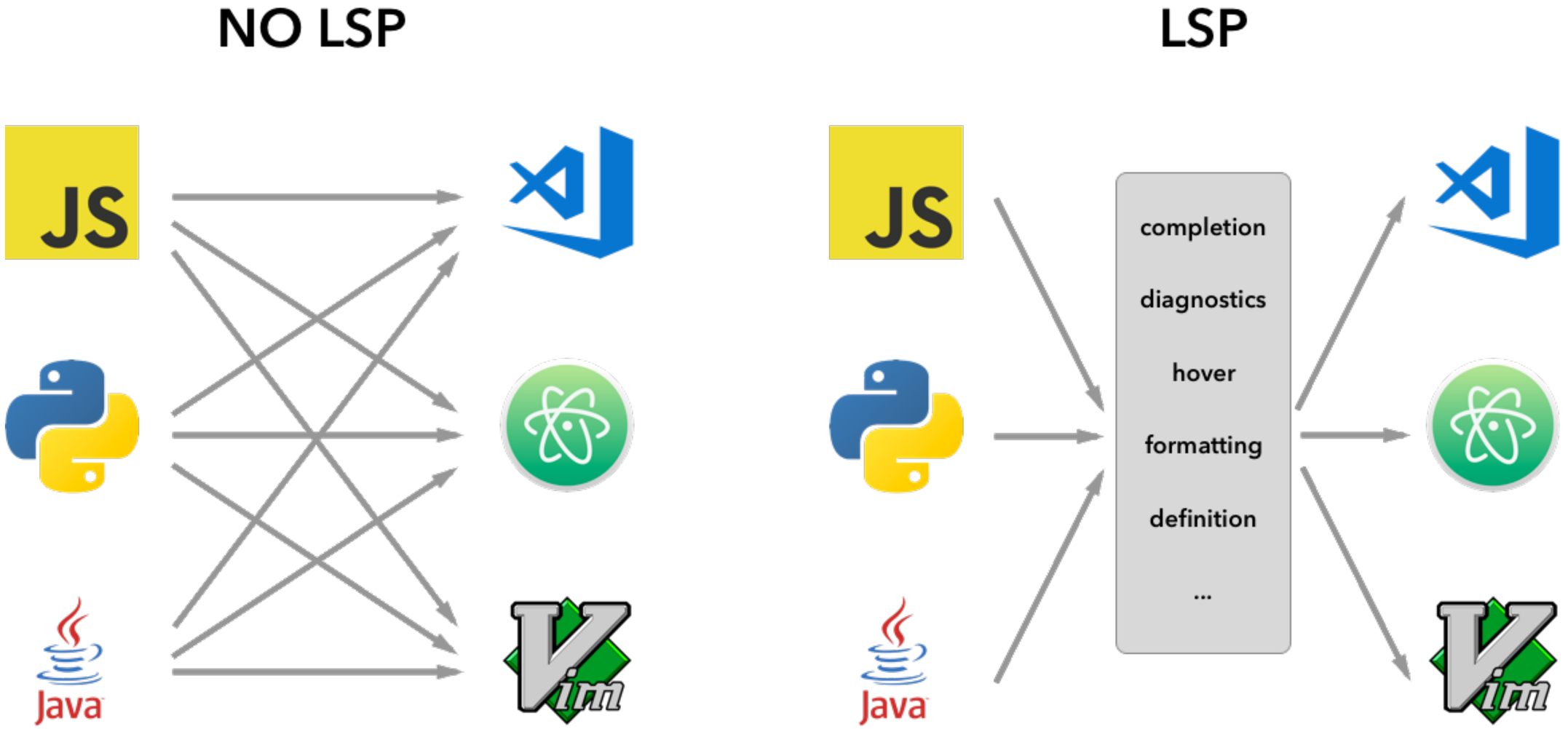
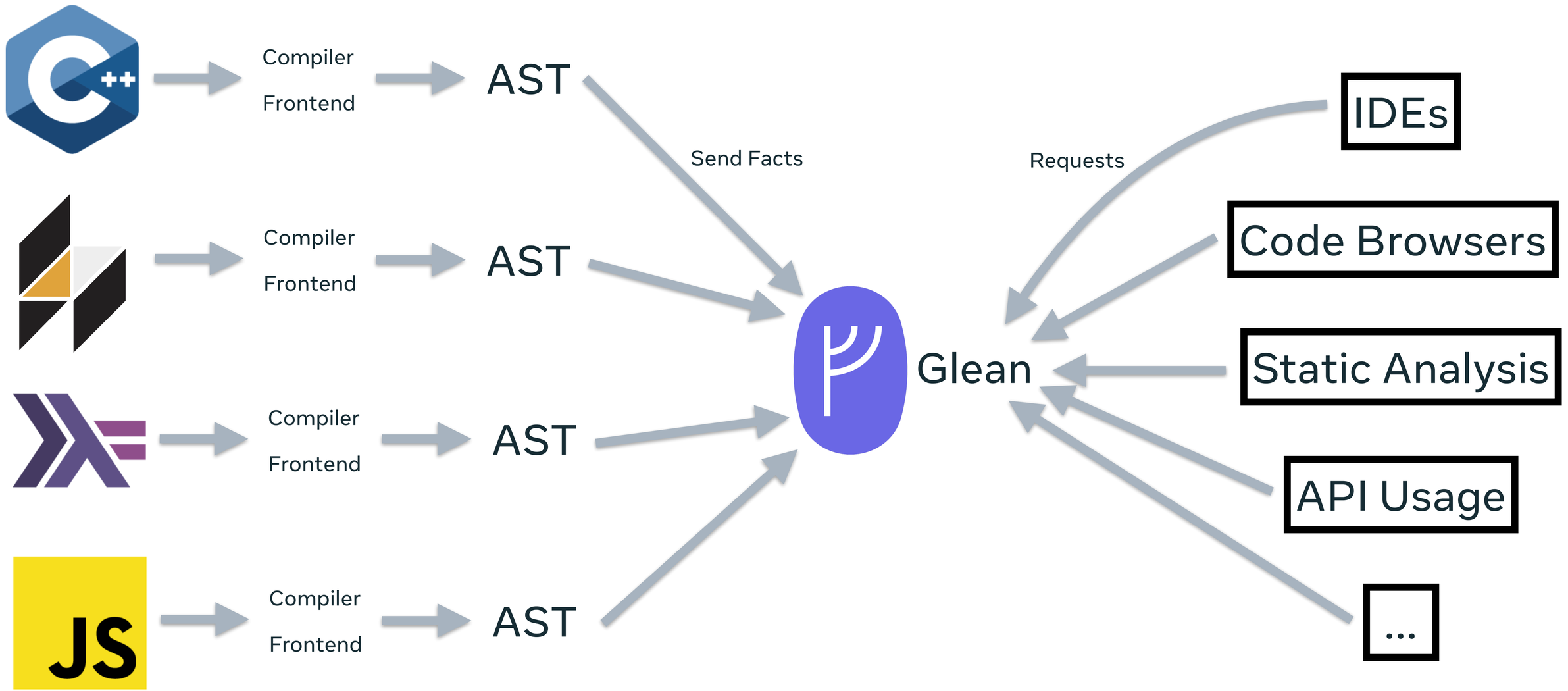
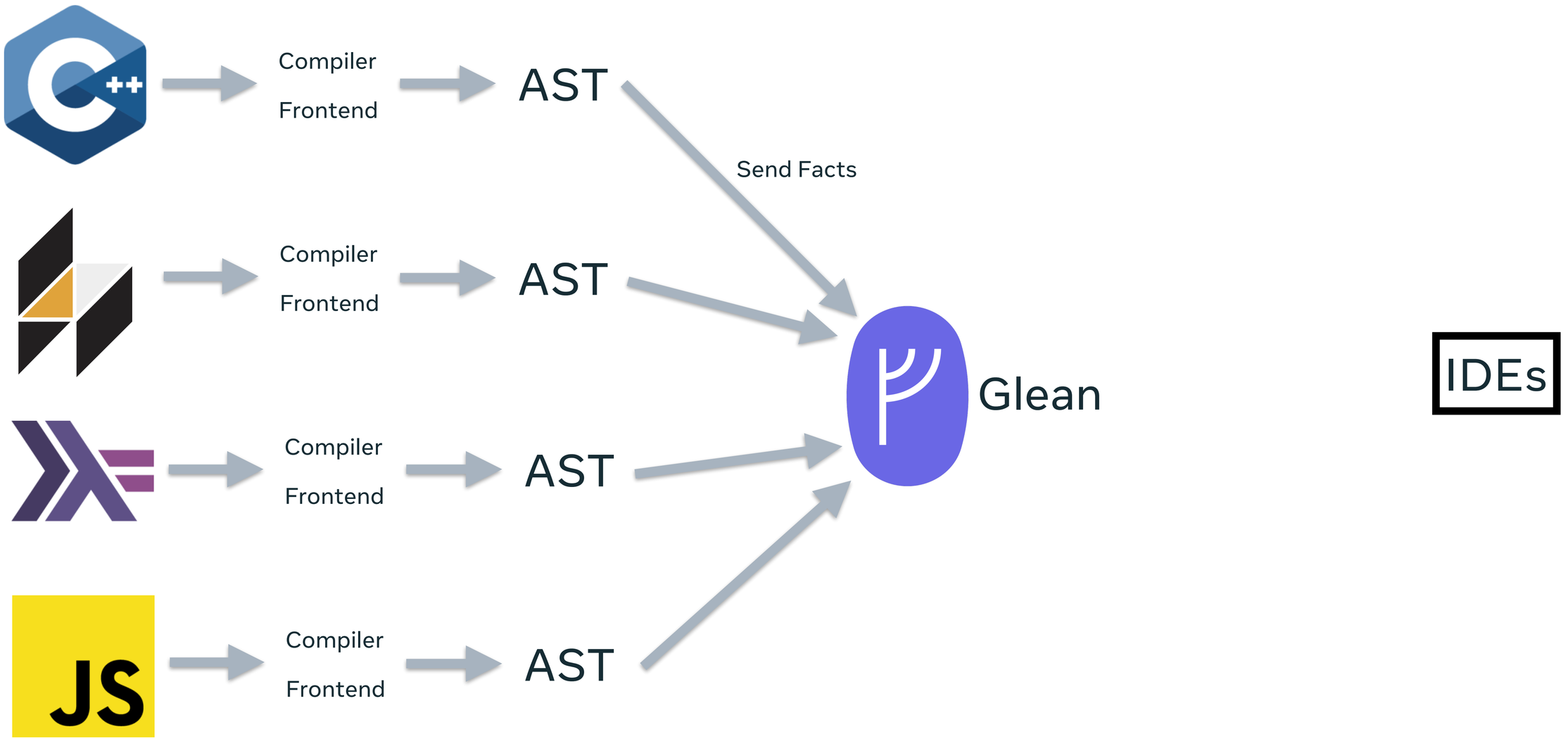
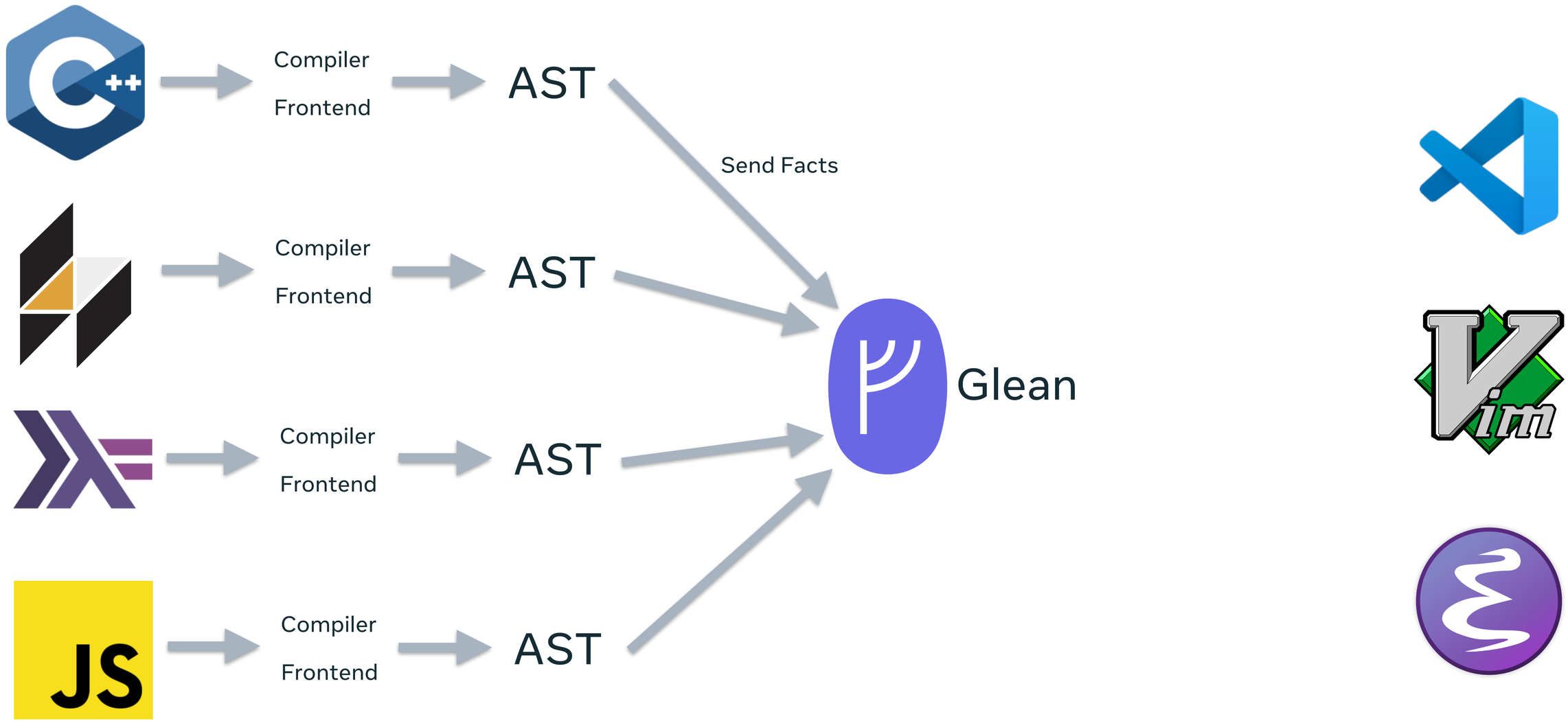
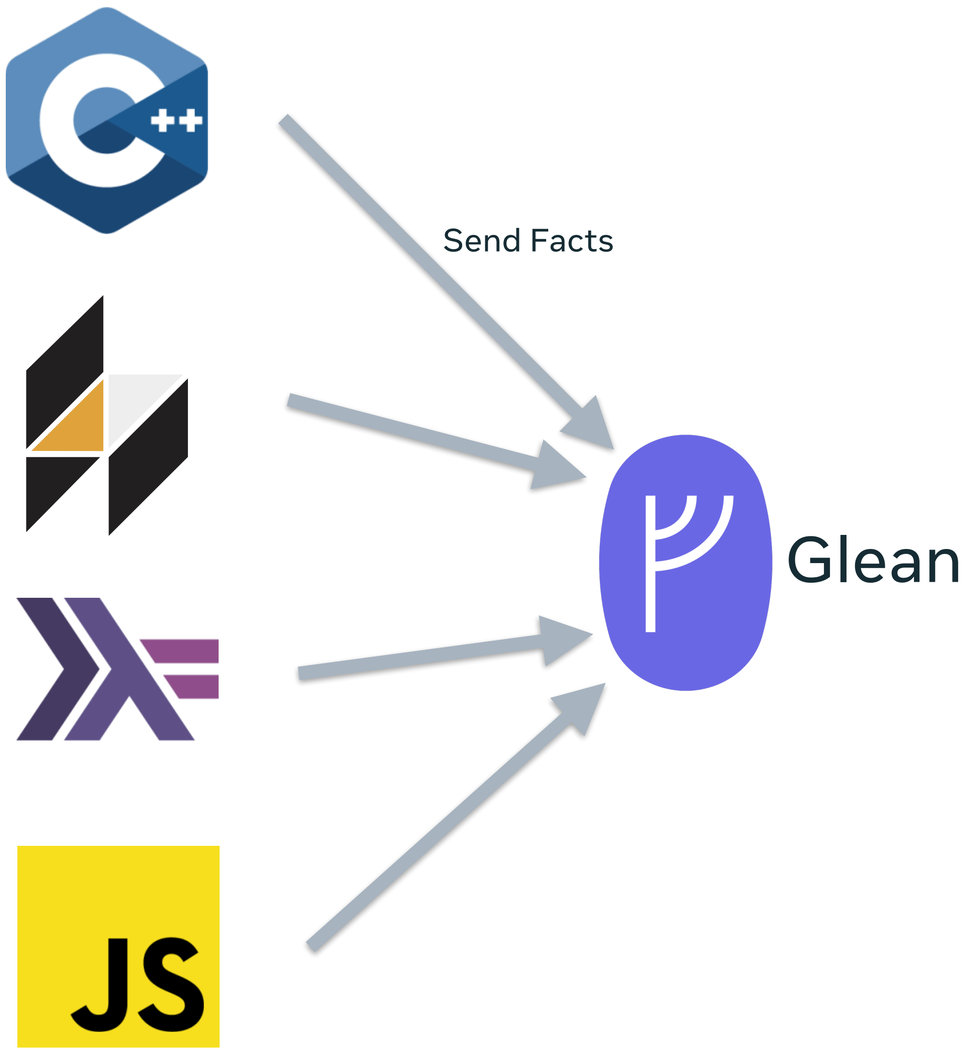


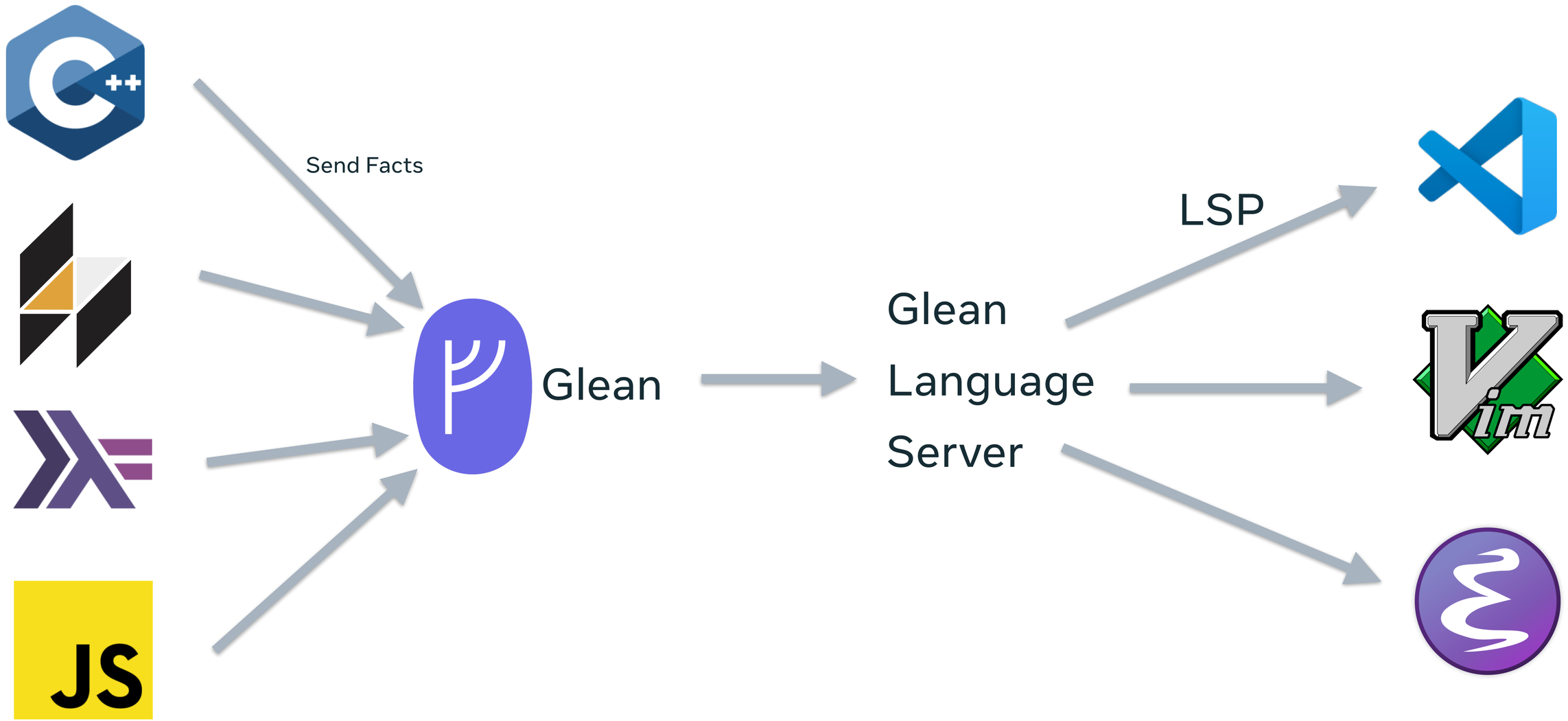
Image from https://code.visualstudio.com/api/language-extensions/language-server-extension-guide


Compiler
Frontend
AST
Compiler
Frontend
AST
Compiler
Frontend
AST
JS
Compiler
Frontend
AST
Send Facts
Glean
Requests
IDEs
Code Browsers
Static Analysis
API Usage
...


Compiler
Frontend
AST
Compiler
Frontend
AST
Compiler
Frontend
AST
Compiler
Frontend
AST
Send Facts
Glean
IDEs


Compiler
Frontend
AST
Compiler
Frontend
AST
Compiler
Frontend
AST
Compiler
Frontend
AST
Send Facts
Glean
JS
Vim


Send Facts
Glean
JS


Send Facts
Glean
Glean
Language
Server
LSP
JS