The Future of AST-Matcher based Refactoring

Stephen Kelly
EuroLLVM 2019
steveire.wordpress.com
@steveire

Stephen Kelly

- @steveire
- steveire.wordpress.com
- KDE
- Qt
- CMake
- Clang

ASTMatcher-based Refactoring

- Scale and Distribute refactoring task
- Makes intractable problems tractable
- Allows creating generic reusable tools
- C++

ASTMatcher-based Refactoring

- Learning curve is very steep
  - Hit complexity very fast
  - Requires existing knowledge of Clang APIs
  - Discovery is difficult
  - Multiple domains of input information
    - AST Nodes, Matchers, Source Locations
- Takes lots of slow developer iteration
  - No plugin System
  - C++

Becoming More Novice-Friendly

- More documentation
- More presentations
- Collaboration
- New features in existing tools
  - Workflow
  - Discovery
  - Debugging
- New tools
  - Speed
- New APIs

Becoming More Novice-Friendly

- More documentation
- More presentations
- Collaboration
- New features in existing tools
  - Workflow
  - Discovery
  - Debugging
- New tools
  - Faster iteration
- New APIs

Parallel Efforts

- ASTER
  - Generate AST Matchers from example code
- `clang::tooling::Transformation`
  - Specify changes based on matched Nodes
- Syntax Tree
  - Syntactic Representation and manipulation

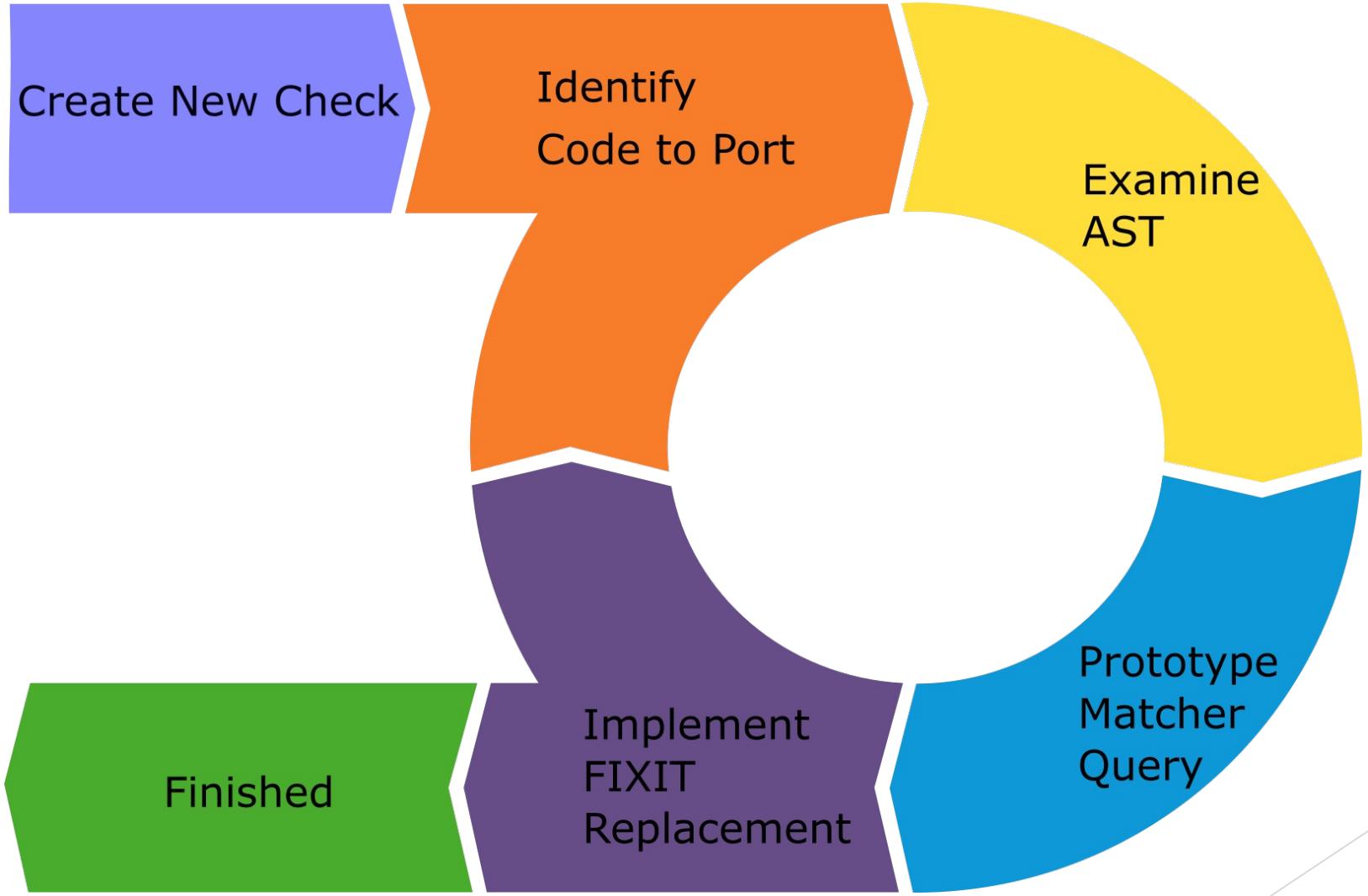
Resources and Collaboration

- clang-query helps, but not referenced well
- My vcblog series
  - 3 Part series aimed at Novices
- clang-query explorer
  - http://ce.steveire.com/z/pcARNO
  - Upstreaming to godbolt.org

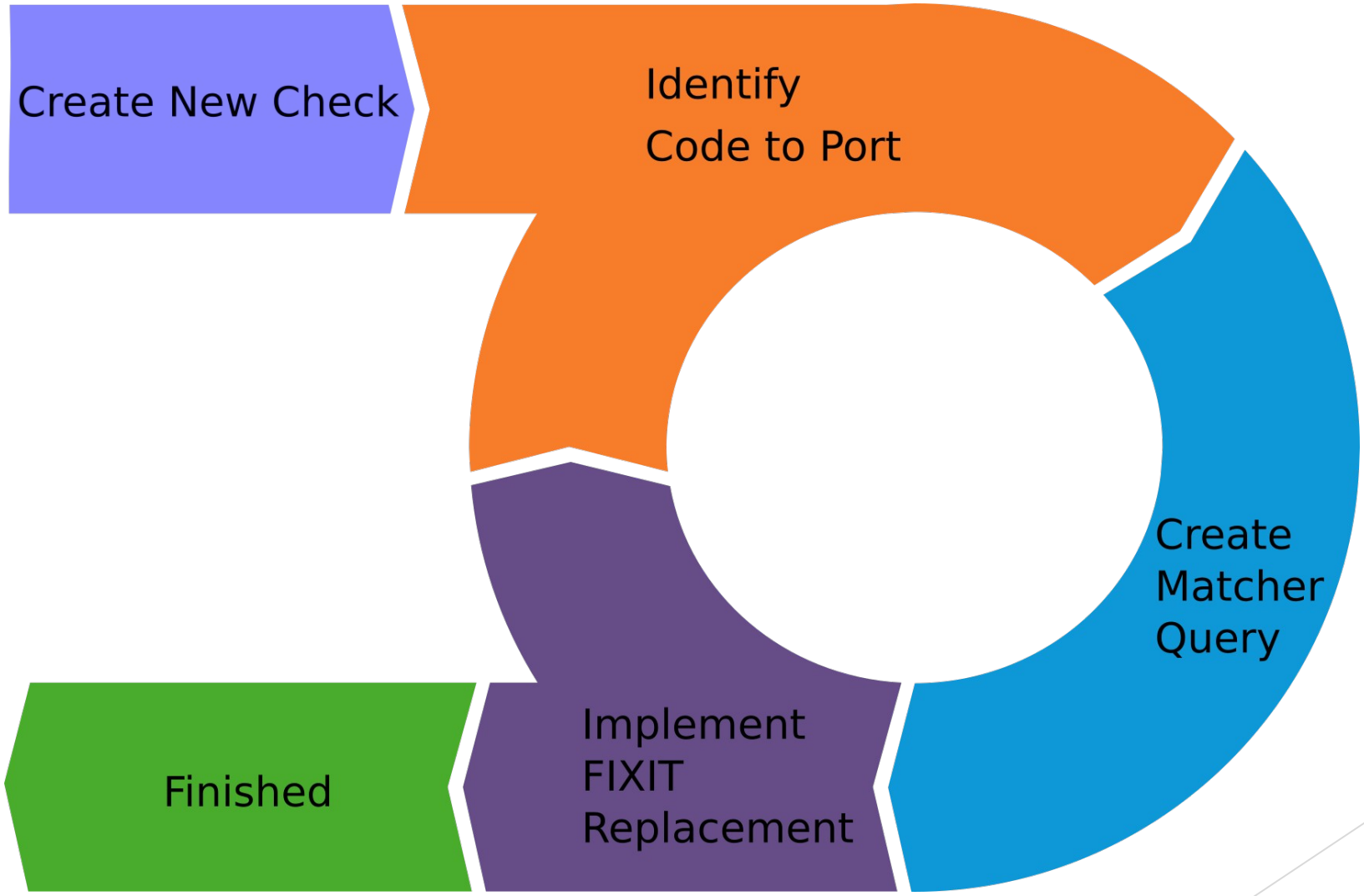
Reduced noise for Novices

- Simplified AST to discover top-level Matchers:
  - http://ce.steveire.com/z/sjyYUJ
- Detailed AST still available:
  - http://ce.steveire.com/z/OpLliE
- Remove ‘invisible’ AST nodes
  - http://ce.steveire.com/z/lHYwEH
  - `ignoringImplicit()` is not enough
  - http://ce.steveire.com/z/EdnWVg

Workflow (today)



Workflow (future)



Discovery

- Close knowledge gap
  - Novice mental model <=> Clang reality
- Discover Matchers
  - http://ce.steveire.com/z/lDNQCx
- Discover Source Locations
  - http://ce.steveire.com/z/JysGF8

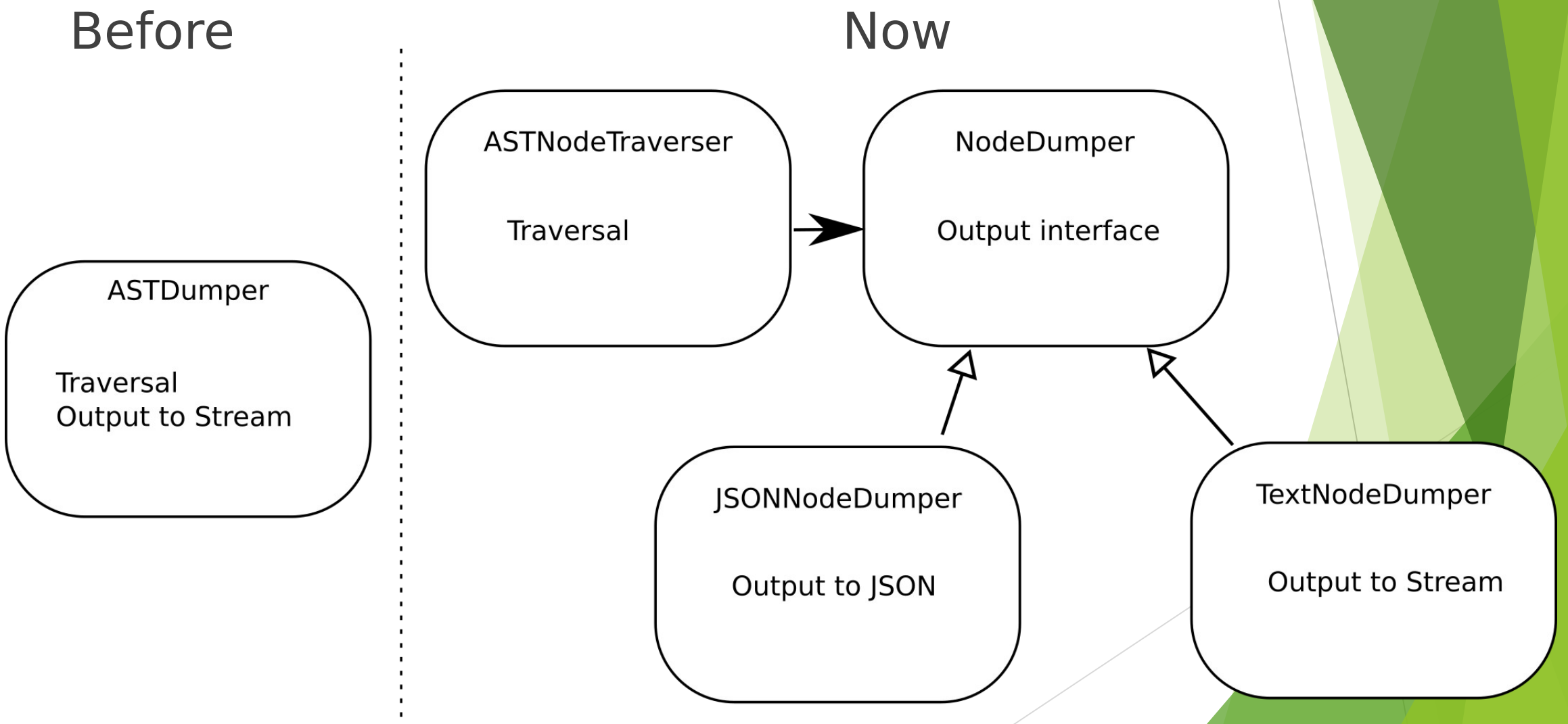
Developer Tooling

- Debugger
  - http://ce.steveire.com/z/JgMave
- Profiler
  - http://ce.steveire.com/z/wmMd3W

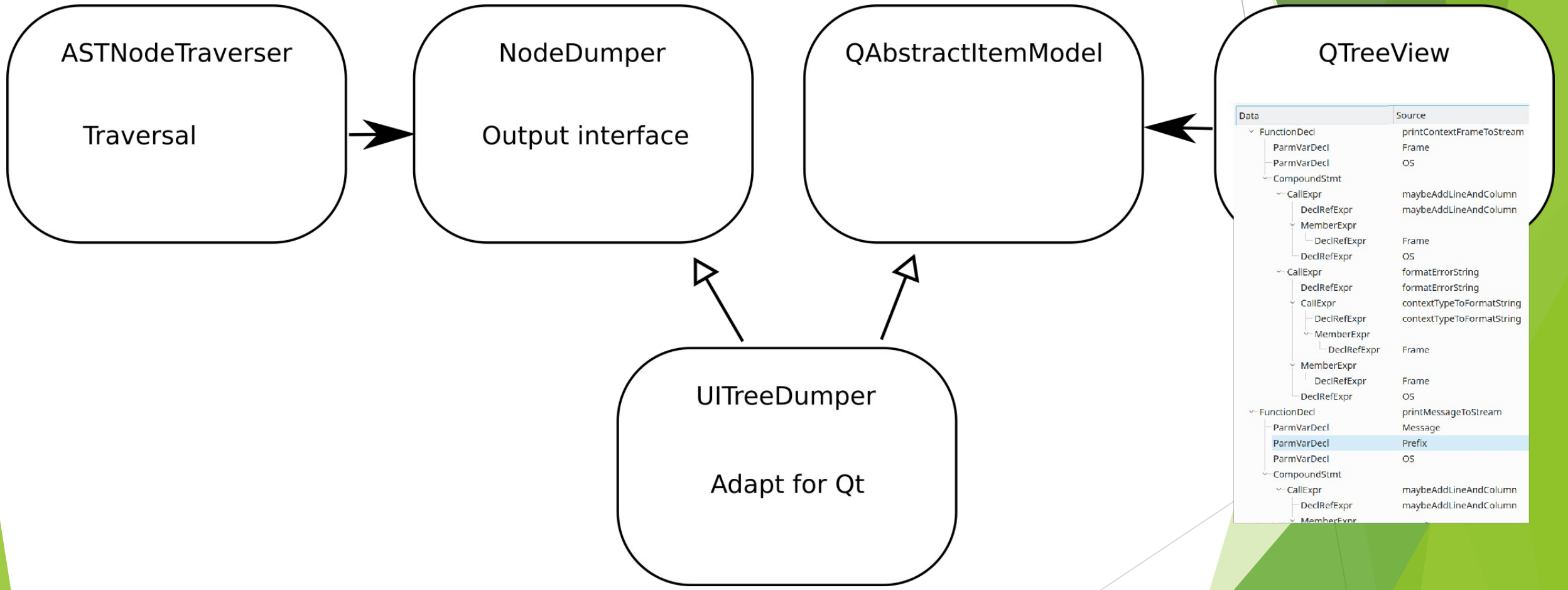
Output independent APIs

- Tooling APIs should be output-independent
  - Diagnostics is a good existing example
- Output independent AST dump traversal
  - New!

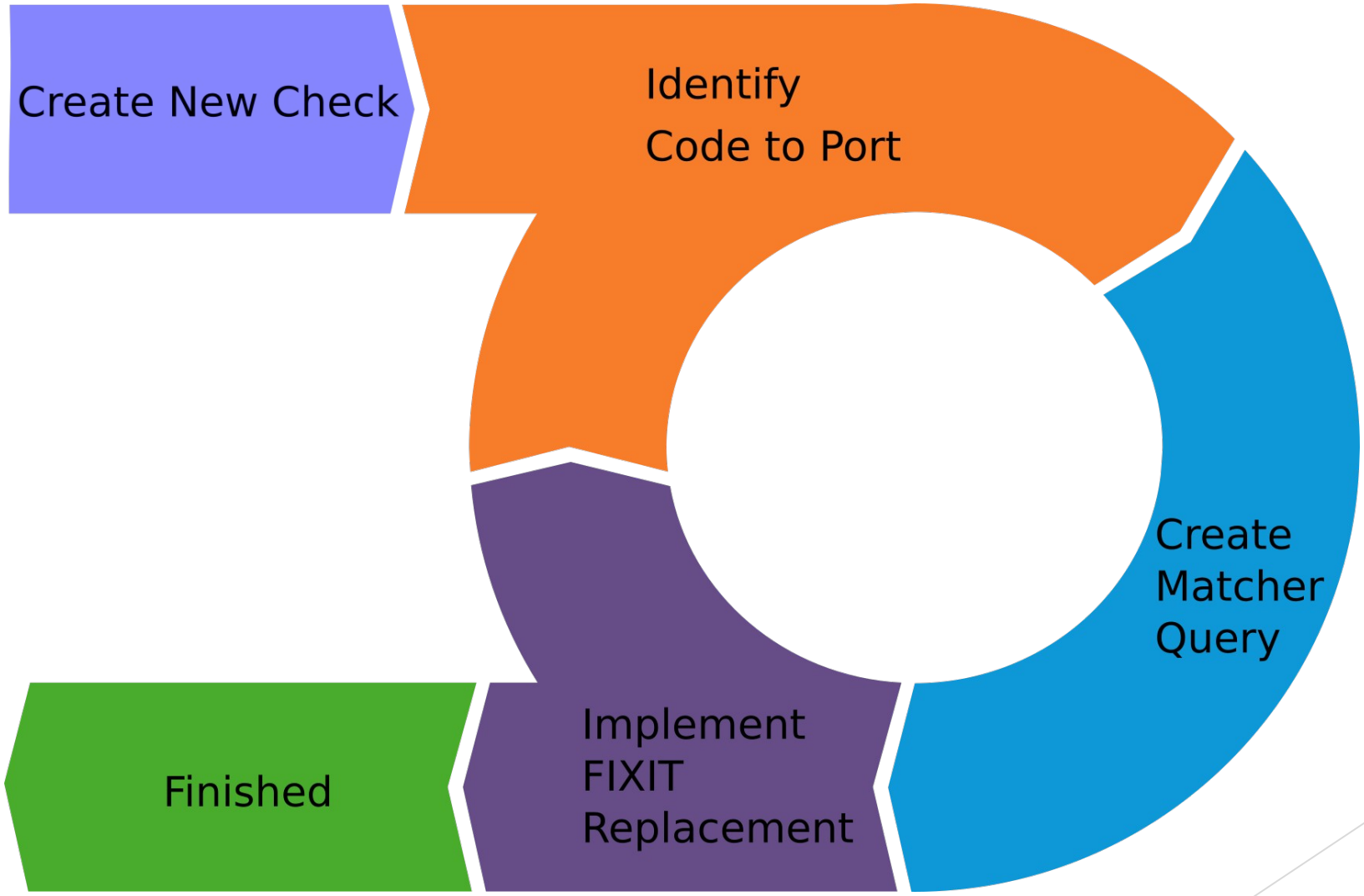
Output independent APIs



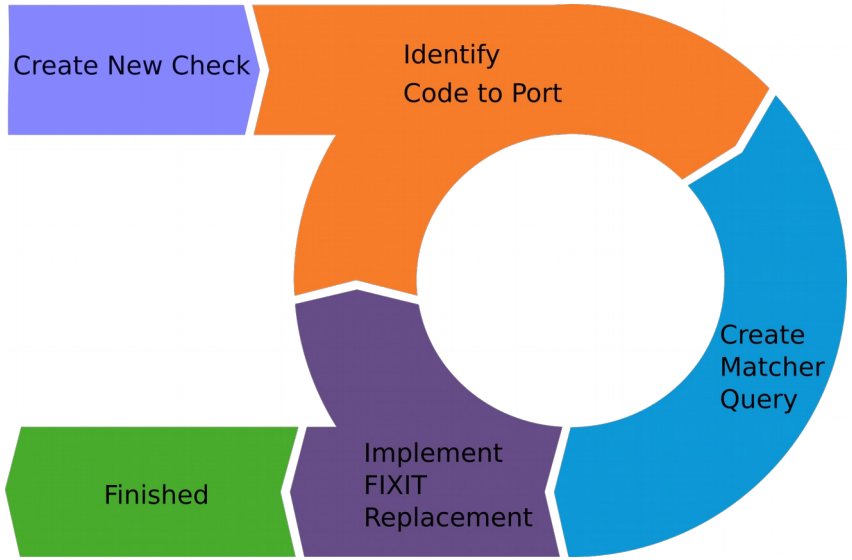
Output independent APIs



Workflow (future)



Workflow (more-future)



Pending Changes

- New Traversal Options
  - Ignore invisible nodes
  - Ignore template instantiations?
- Output possible Matchers from `clang-query`
  - Expose from `ast_matchers::dynamic::Registry`
- Debugger interface for `ASTMatchFinder`
  - Used for debugging and profiling
- AST introspection tool
  - Generate code for source locations etc

```
class DebuggerInterface {
  virtual void DeclareMatcher(
      const DynMatcherInterface *Matcher, llvm::StringRef
Name,
      const DynMatcherInterface *Parent) const = 0;

  virtual void CreateBinding(
    const DynMatcherInterface *Matcher, llvm::StringRef Name,
    const DynMatcherInterface *BindingMatcher) = 0;

  virtual void DebugMatch(
      const ast_type_traits::DynTypedNode &DynNode,
      const DynMatcherInterface *Matcher, bool IsMatch) = 0;
};
```

clang-ast-introspection

- New tool run at build-time
- Parses `clang/AST/AST.h`
- Uses AST-Matchers
- Generates
  - C++ API for source location texts
  - JSON data for Javascript bindings

libClangQuery

- Library-ify most of clang-query tool
- `struct QueryFactory`

```
{
  virtual Query *MakeMatchQuery(
        StringRef Source,
        const DynTypedMatcher &Matcher)
  { return new MatchQuery(Source, Matcher); }
  // etc...
};
```

ASTMatcher-based Refactoring

- Learning curve is very steep
  - Hit complexity very fast
  - Requires existing knowledge of Clang APIs
  - Discovery is difficult
  - Multiple domains of input information
    - AST Nodes, Matchers, Source Locations
- Takes lots of slow developer iteration
  - No plugin System
  - C++

Summary

- Mechanical refactoring enabled by Clang
- Barrier to entry too-great for Clang Novices
  - Must self-build Clang
  - Reduce verbosity of output by default
  - Add discovery features
  - Reduce domains of data (less AST)
- Shorten iteration time
  - Interpreted languages
  - Live result updates

What Now?

- Right analysis/Useful work?
- Is there interest in LLVM?
- Collaboratorators?

```
match questionDecl(
    hasAnswer(clearExpr().bind("Answer"))
    )

void check(auto const& Result)
{
    auto Answer =
        Result.Nodes->getAs<ClearExpr>("Answer");
    Answer->dump();
}
```