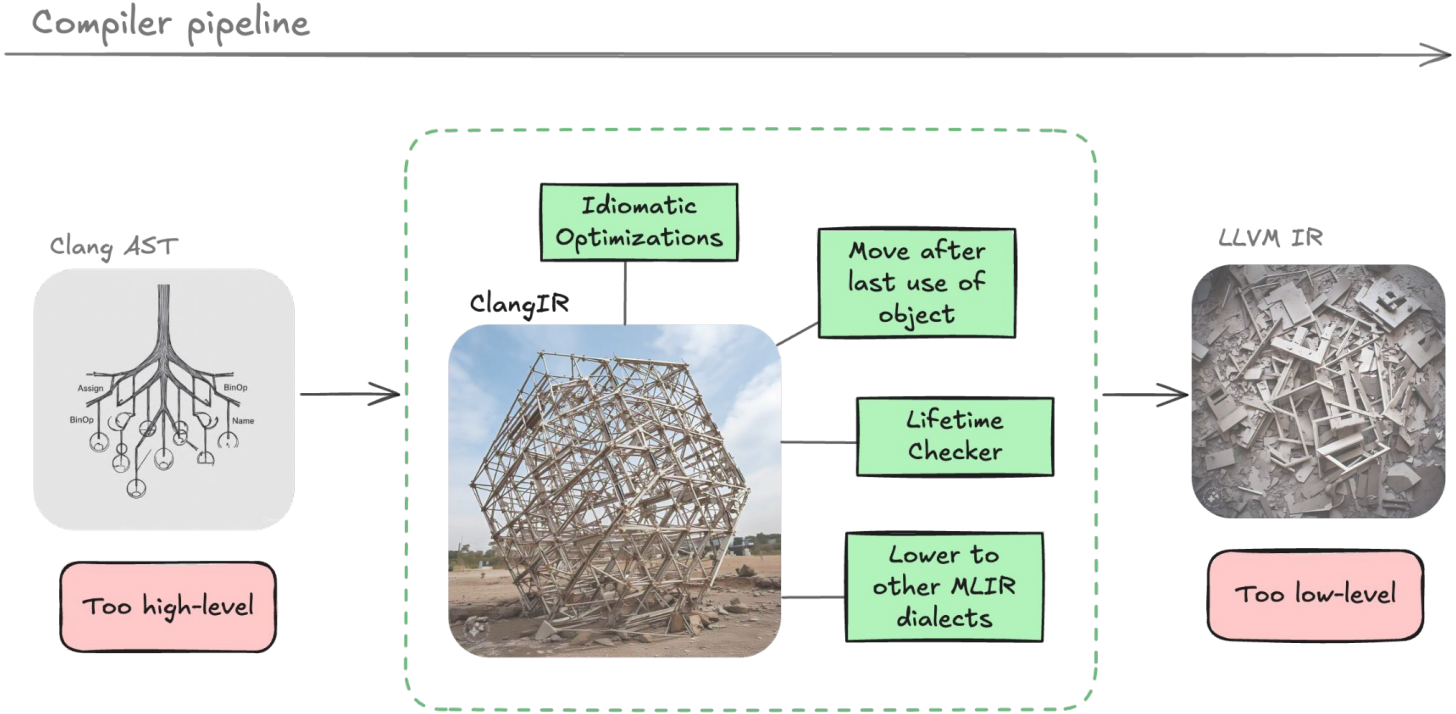
ClangIR: Upstreaming an Incubator Project

Bruno Cardoso Lopes, Meta
Andy Kaylor, NVIDIA

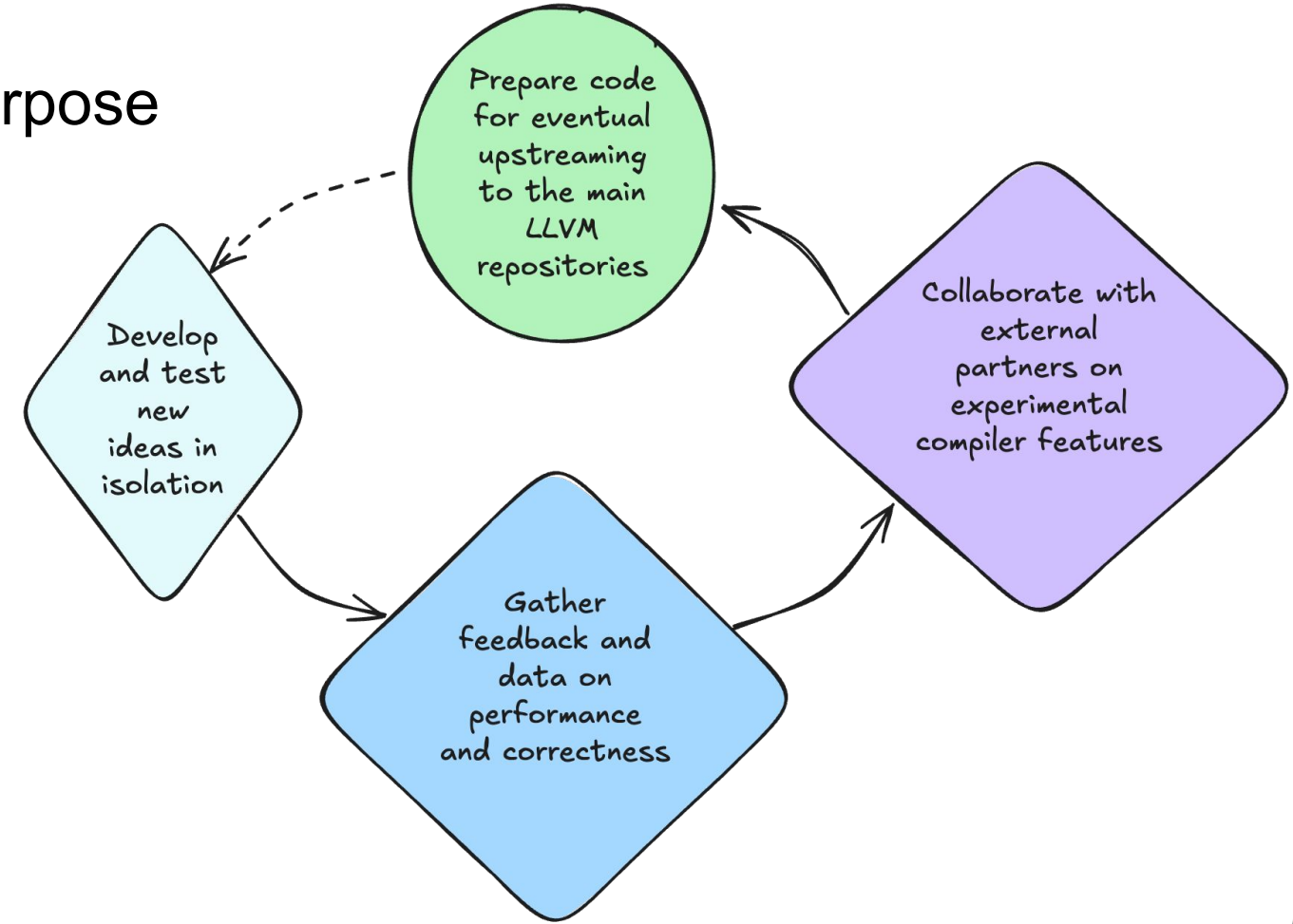
ClangIR recap

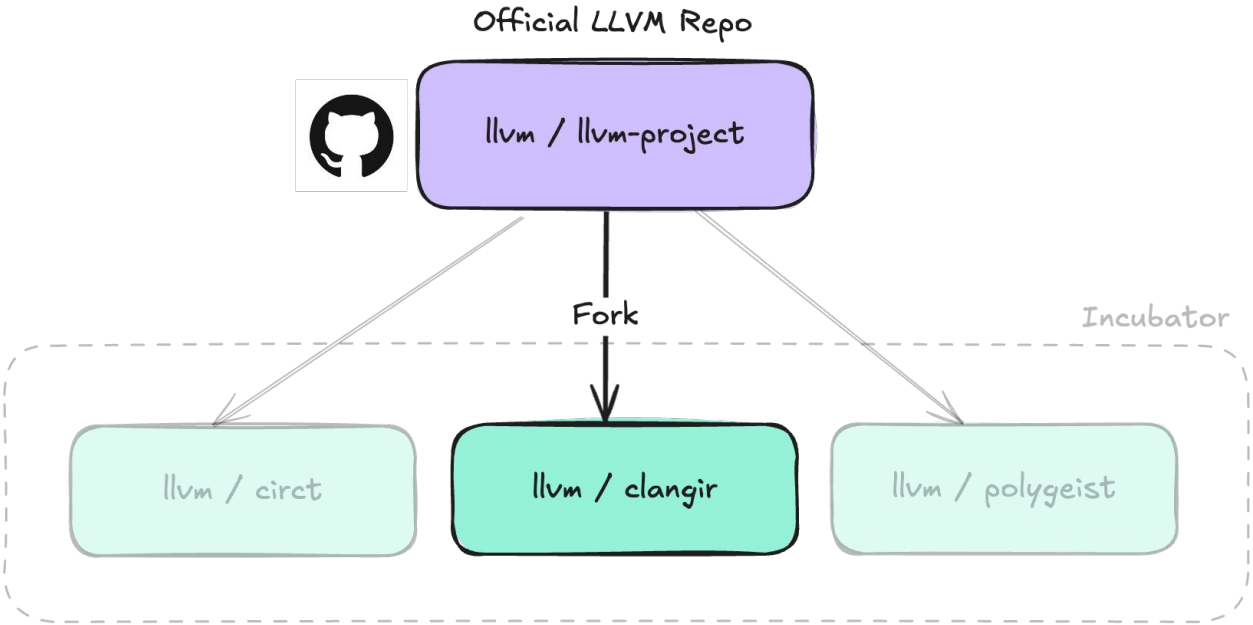


ClangIR timeline

ClangIR timeline

Incubator's purpose




Official LLVM Repo
llvm / llvm-project
Fork
Incubator
llvm / circt
llvm / clangir
llvm / polygeist

From RFC to incubator

C/C++ MLIR frontend working group.

Move to LLVM incubator.

Incubation advantages

Ability to use CI and other LLVM project infrastructure

Project visibility

- Path for building upstreaming agreement
- The LLVM umbrella attracts more contributors

Incubation downsides

Less visibility than llvm-project

Users prefer llvm-project over a fork

Investment risk

- Might never meet the bar
- Never upstreamed

Incubation process is not formalized, consensus can be challenging

Life in clangir incubator

clangir.org guides

LLVM practices enforced ⚠️

Paving the way for upstreaming

Additional freedom for exploration 👩🏽‍🔬

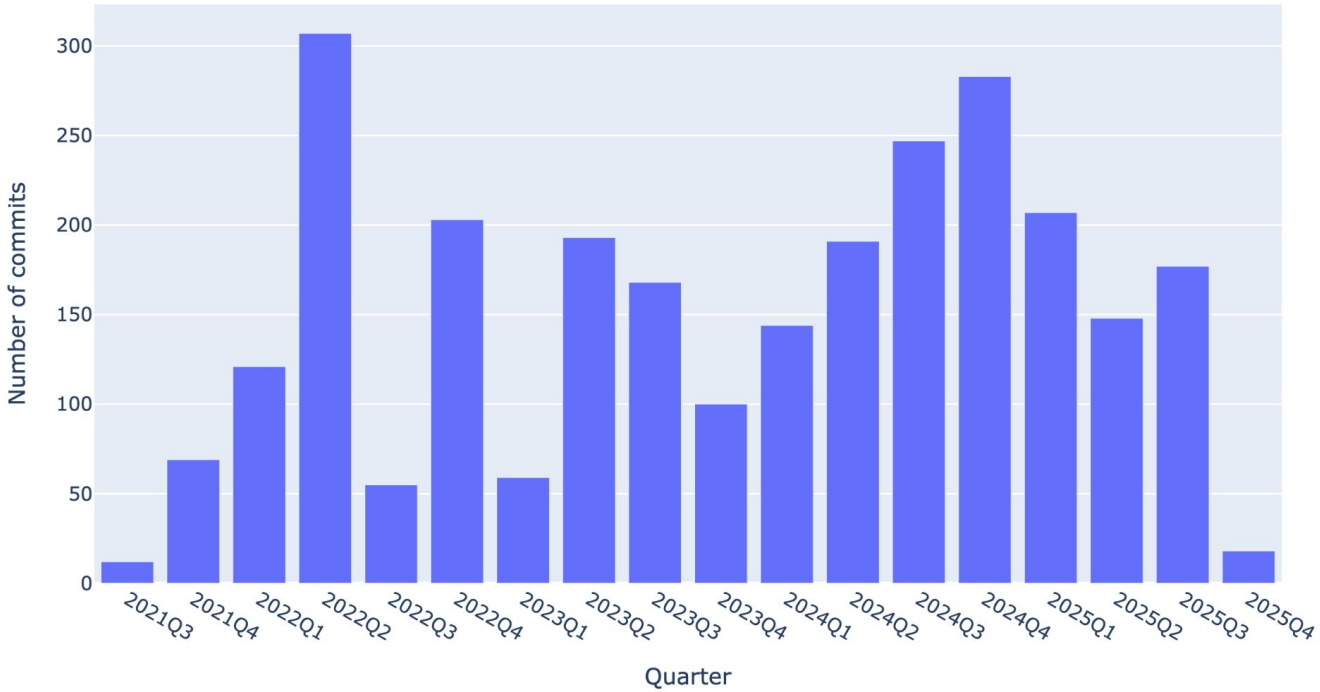
Challenges

Wait! I heard this was working

Always in motion is top-of-tree 🏃

Upstream and incubator differences

Contributions



Authors come and go ✈️

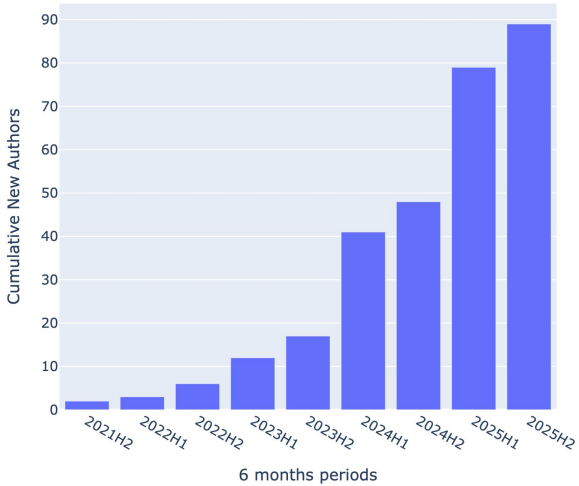
Contribution span varies

Good for incremental pieces (e.g. builtins)

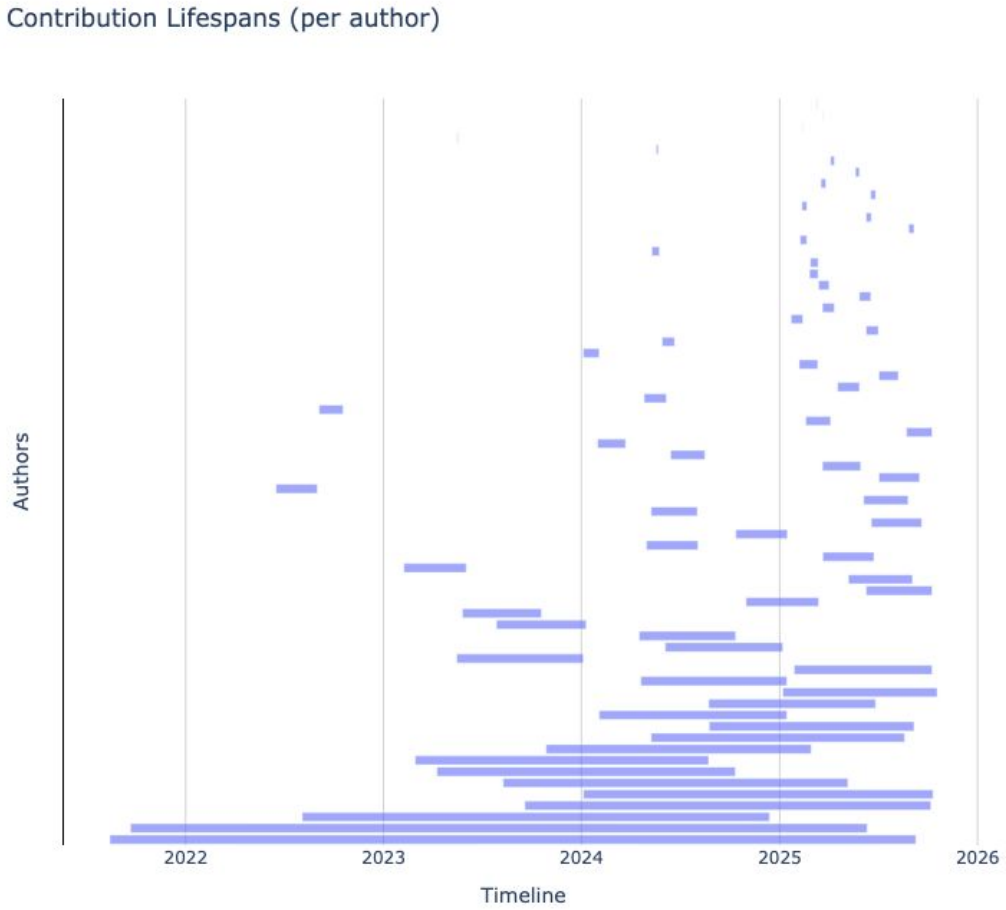
    Especially attractive to newcomers

Bad for solidifying expertise

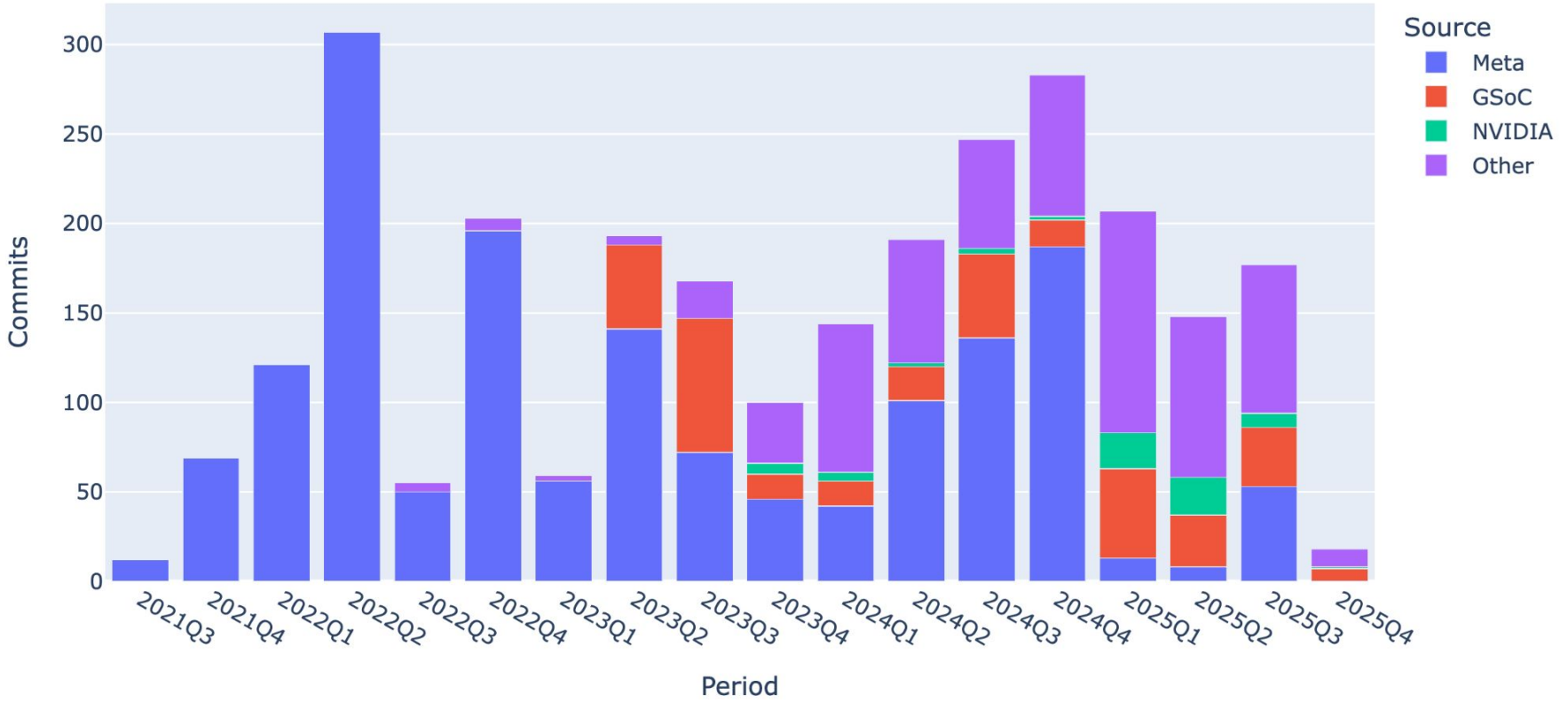
Some couldn't wait for upstreaming



23 only contributed once



Incubator: Grouped Commits per Quarter



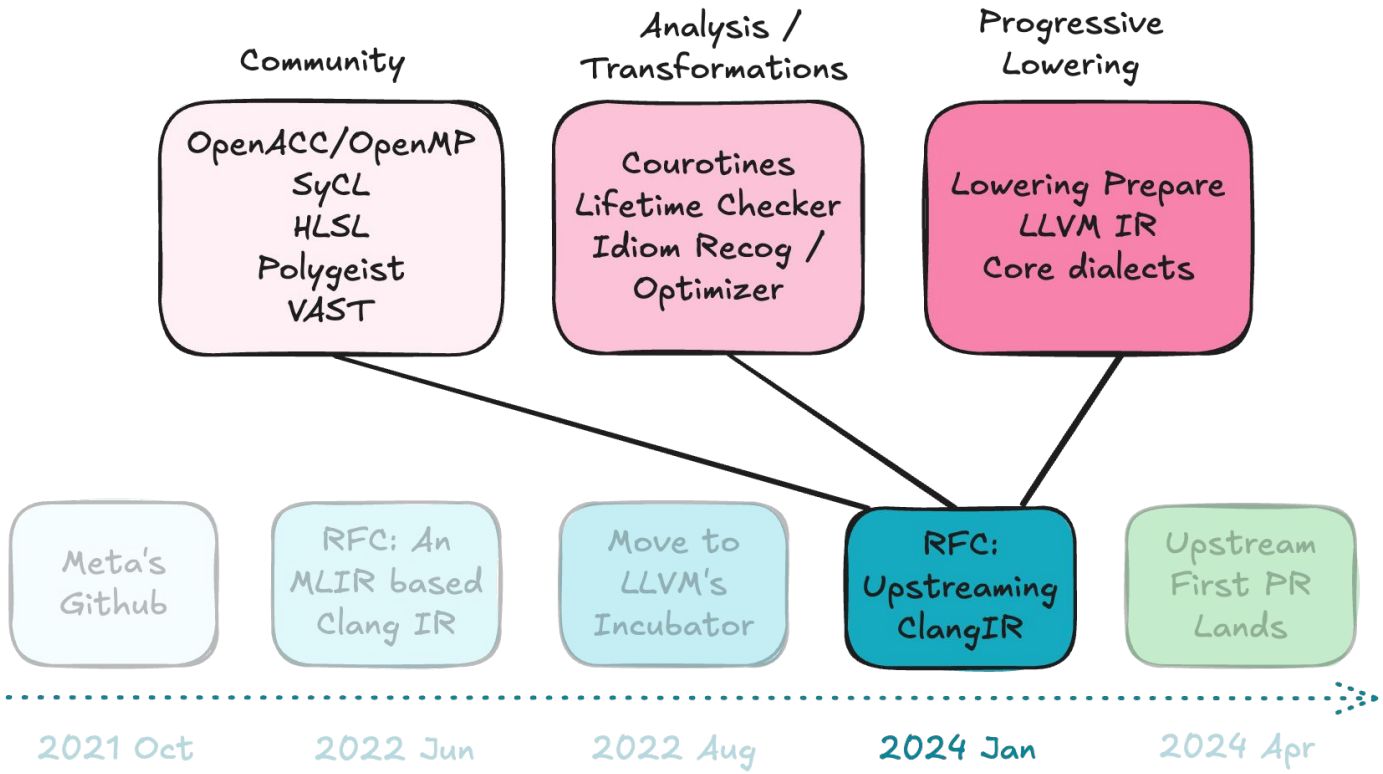
Incubator - In hindsight

Challenging

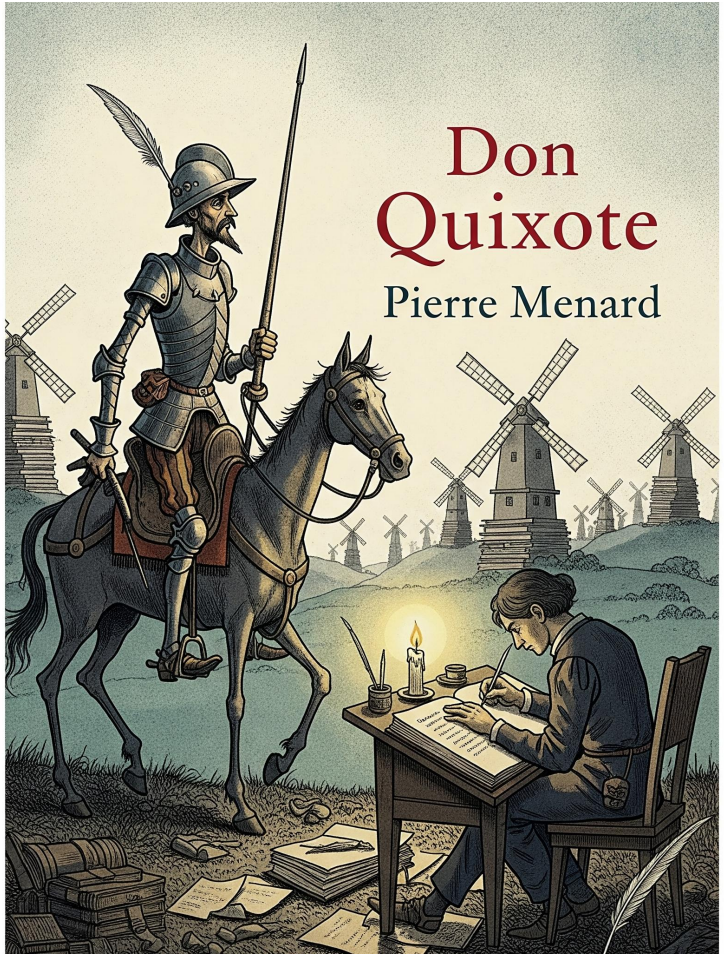
… but worth it

Community up and running

Steady progress in upstreaming


Community
Analysis / Transformations
Progressive Lowering
OpenACC/OpenMP
SyCL
HLSL
Polygeist
VAST
Courotines
Lifetime Checker
Idiom Recog / Optimizer
Lowering Prepare
LLVM IR
Core dialects
Meta's Github
RFC: An MLIR based Clang IR
Move to LLVM's Incubator
RFC: Upstreaming ClangIR
Upstream First PR Lands
2021 Oct
2022 Jun
2022 Aug
2024 Jan
2024 Apr

Upstreaming begins 🚀

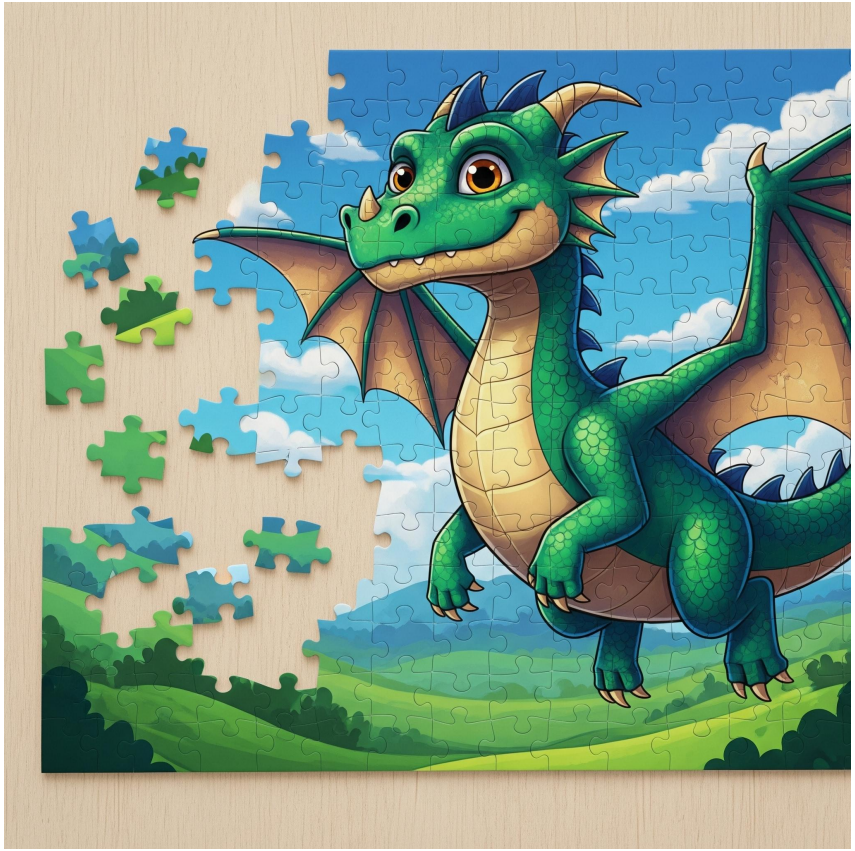


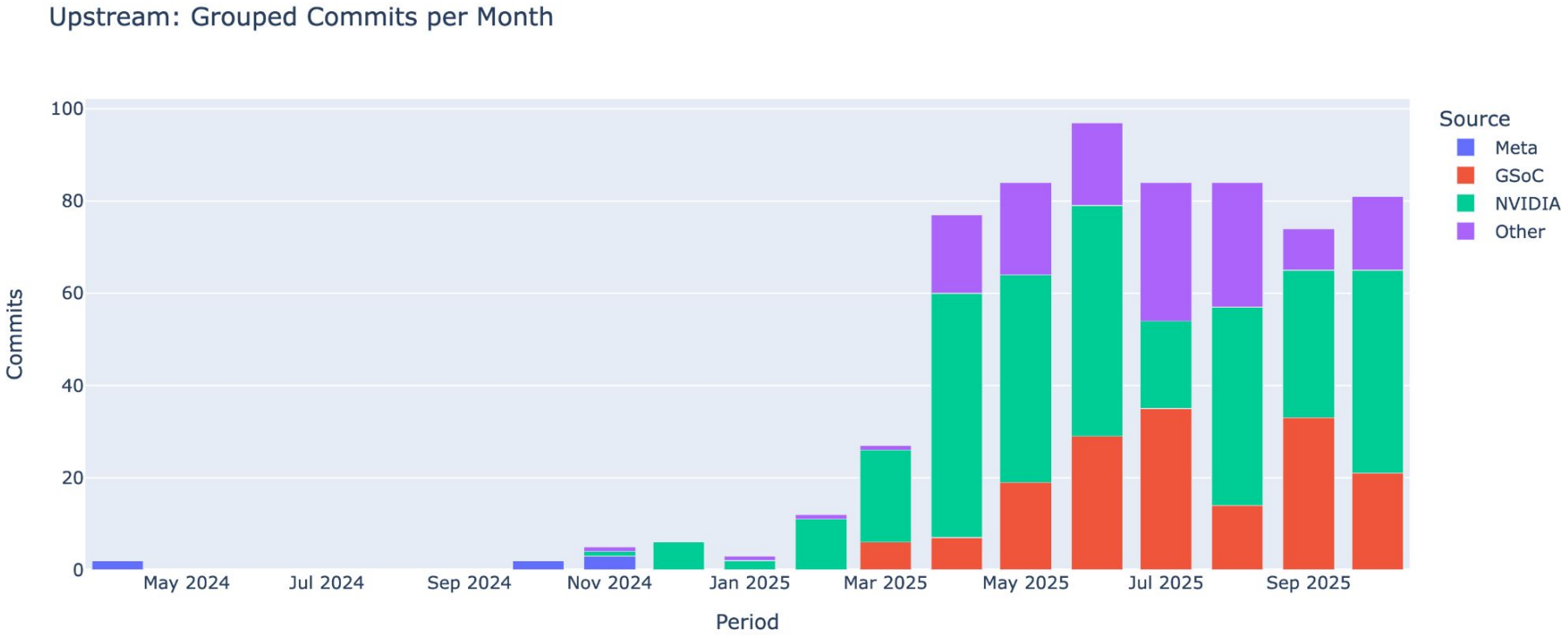
A useful accident

- The contributors to the upstreaming effort are mostly different than the original contributors to the incubator
- Pierre Menard: Author of the Quixote
  - New contributors creating the same code
  - Fresh perspectives bring new context
- Wax on, wax off
  - Learning by going through the motions
- Result: twice as many people have deep understanding of the code

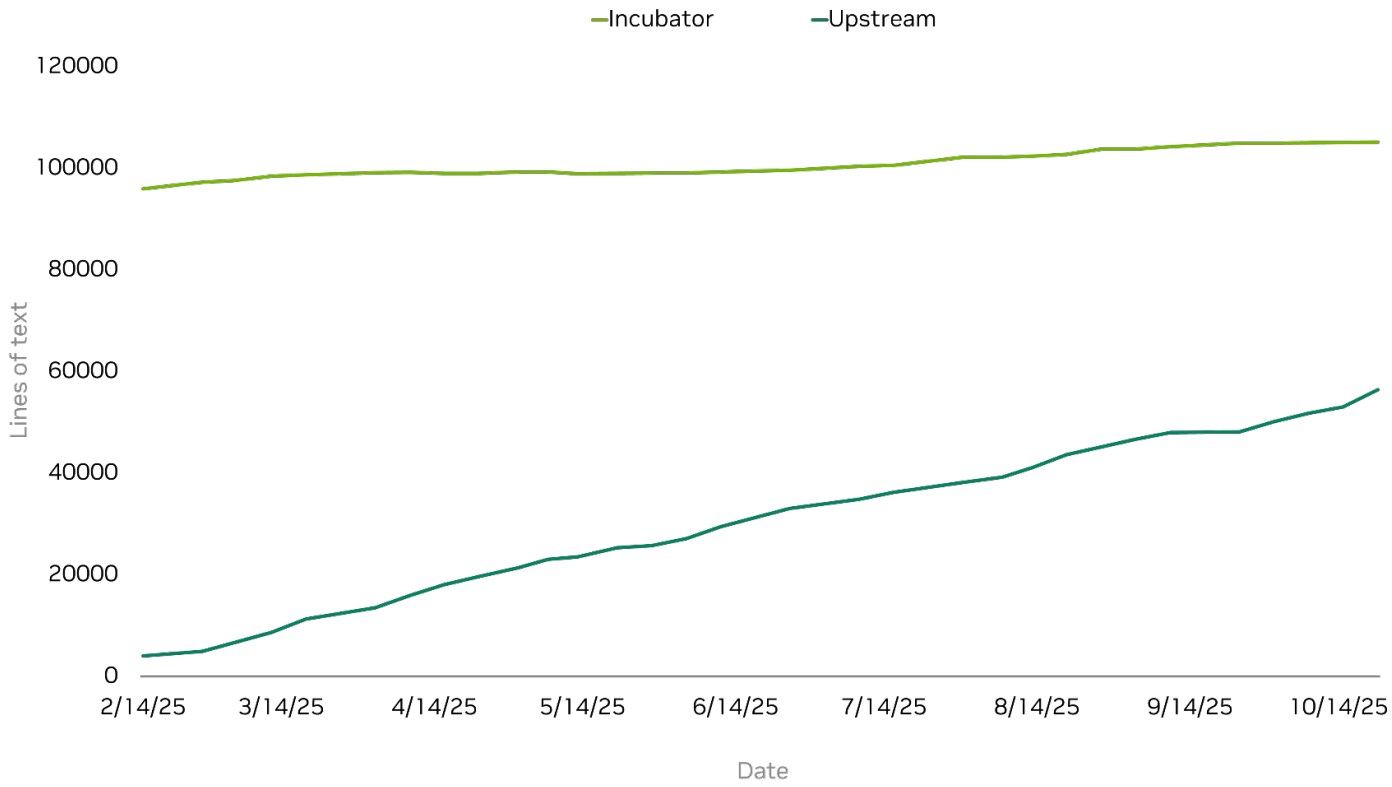
Going live!

- Doing everything *as if* from scratch
  - The clang maintainers didn't want the incubator project merged en masse
  - There was no practical way to "replay" the incubator commits
- Each PR for upstreaming code is treated exactly as if it were a new contribution
- Each PR is held to the same standards as any LLVM contribution
  - Contributor must thoroughly understand the code
  - All functionality must be tested
  - Conformance to coding standards
  - Single purpose
  - Isolated change, as small as possible

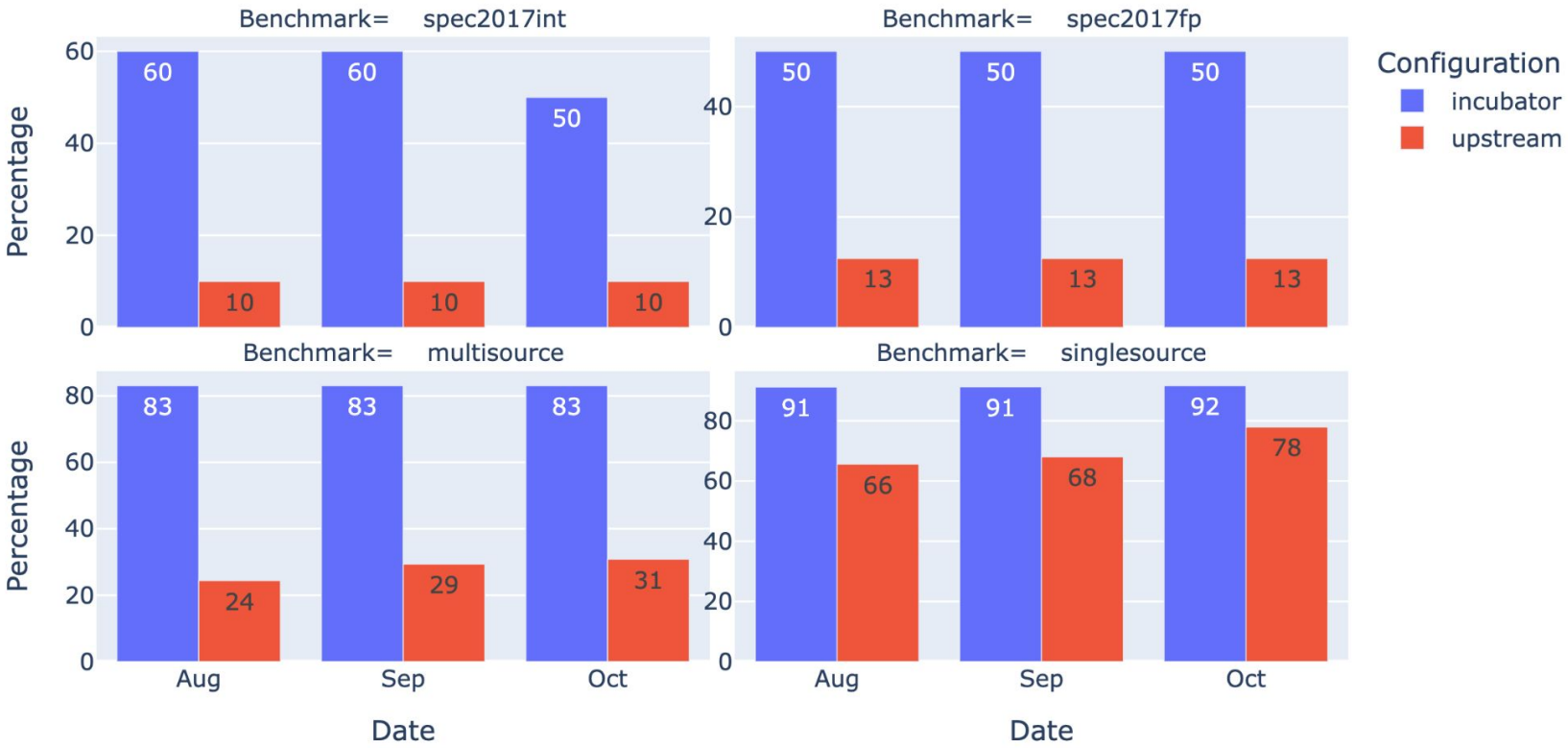



Upstream: Grouped Commits per Month
Commits
100
80
60
40
20
0
May 2024
Jul 2024
Sep 2024
Nov 2024
Jan 2025
Mar 2025
May 2025
Jul 2025
Sep 2025
Period
Source
Meta
GSoC
NVIDIA
Other

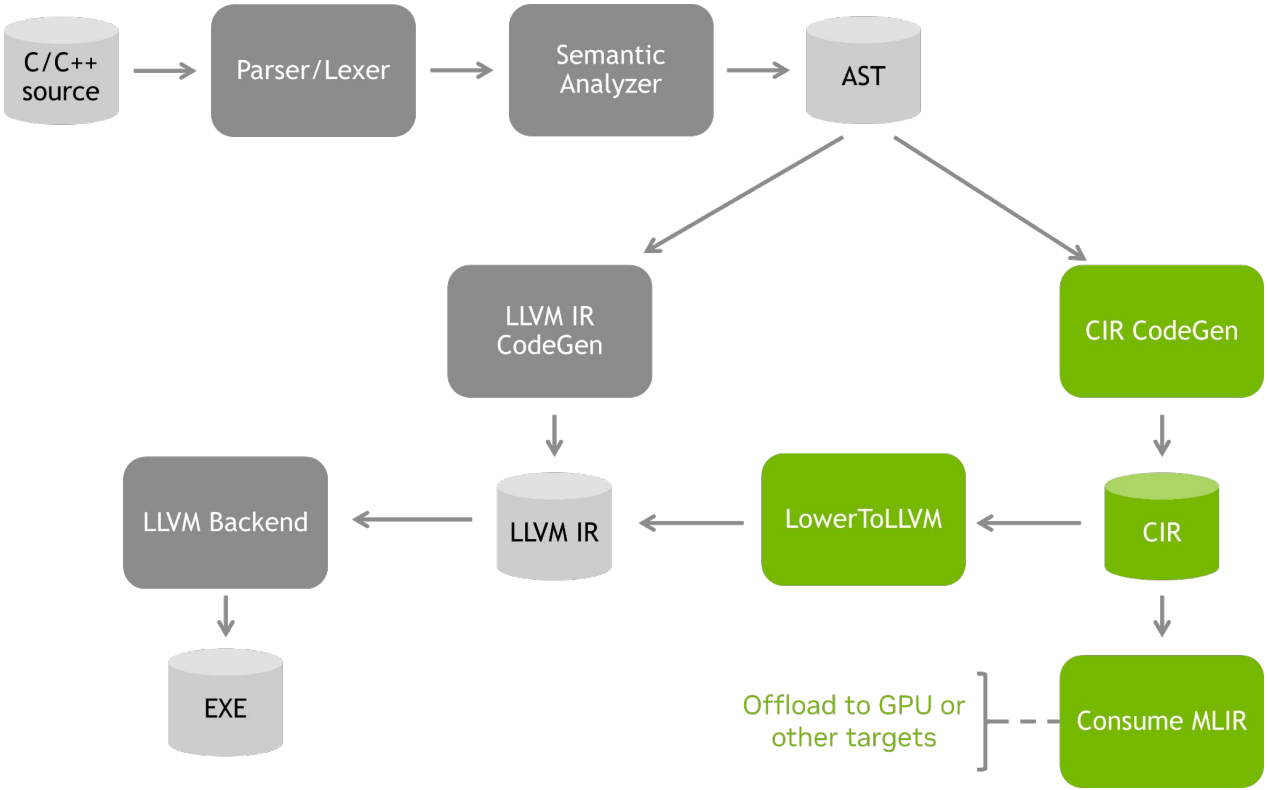
Upstreaming progress



Passing Benchmarks (% of completion)



CIR Won’t Always Go To LLVM IR



Growing pains

- Build problems
  - The ClangIR build sometimes gets broken by changes in MLIR or other parts of clang
  - Sometimes the incubator code is using AST interfaces that no longer exist upstream
    - This is part of the pain of rebasing the incubator
    - It also adds some overhead to the upstreaming process
- Chasing two moving targets
  - The incubator tried to follow the structure of clang's LLVM IR codegen
    - …at the time the incubator code was written
  - Development is still proceeding in the incubator
  - Upstreaming pulls from the incubator, but we also need to check for changes in LLVM IR codegen

Trying to synchronize with classic codegen

- We want generated CIR to be semantically equivalent to the LLVM IR clang generates
- LLVM IR codegen is always changing
- Strategies
  - We've tried to use the same code structures whenever possible
    - CIRGenFunction looks very much like CodeGenFunction
  - Most of our tests check LLVM IR lowering from CIR and direct LLVM IR codegen
    - This will alert us when something changes in the non-CIR path
  - Eventually we'd like to run all existing codegen tests with CIR enabled

Embarrassingly Parallel Code?

clang/lib/CIR/CodeGen/CIRGenExprScalar.cpp

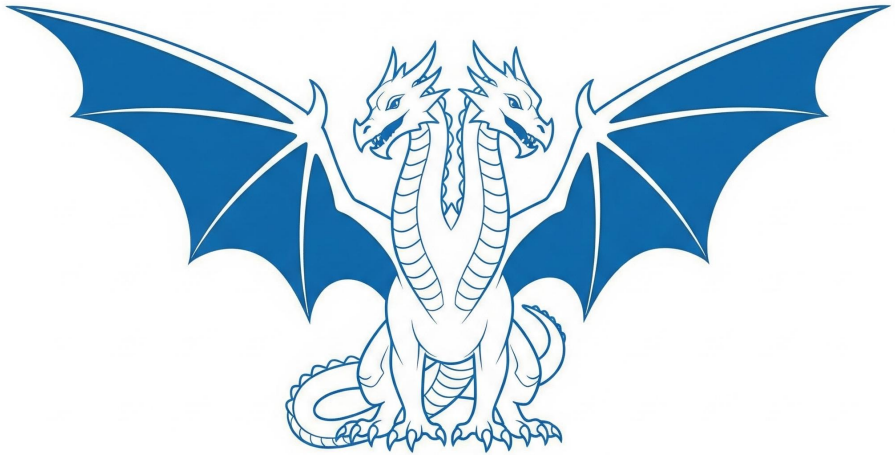
```
  // Unary Operators.
  mlir::Value VisitUnaryPostDec(const UnaryOperator *e) {
    LValue lv = cgf.emitLValue(e->getSubExpr());
    return emitScalarPrePostIncDec(e, lv,
cir::UnaryOpKind::Dec,
                                   false);
  }
  mlir::Value VisitUnaryPostInc(const UnaryOperator *e) {
    LValue lv = cgf.emitLValue(e->getSubExpr());
    return emitScalarPrePostIncDec(e, lv,
cir::UnaryOpKind::Inc,
                                   false);
  }
  mlir::Value VisitUnaryPreDec(const UnaryOperator *e) {
    LValue lv = cgf.emitLValue(e->getSubExpr());
    return emitScalarPrePostIncDec(e, lv,
cir::UnaryOpKind::Dec,
                                   true);
  }
  mlir::Value VisitUnaryPreInc(const UnaryOperator *e) {
    LValue lv = cgf.emitLValue(e->getSubExpr());
    return emitScalarPrePostIncDec(e, lv,
cir::UnaryOpKind::Inc,
                                   true);
  }
```

clang/lib/CodeGen/CGExprScalar.cpp

```
// Unary Operators.
Value *VisitUnaryPostDec(const UnaryOperator *E) {
  LValue LV = EmitLValue(E->getSubExpr());
  return EmitScalarPrePostIncDec(E, LV, false, false);
}
Value *VisitUnaryPostInc(const UnaryOperator *E) {
  LValue LV = EmitLValue(E->getSubExpr());
  return EmitScalarPrePostIncDec(E, LV, true, false);
}
Value *VisitUnaryPreDec(const UnaryOperator *E) {
  LValue LV = EmitLValue(E->getSubExpr());
  return EmitScalarPrePostIncDec(E, LV, false, true);
}
Value *VisitUnaryPreInc(const UnaryOperator *E) {
  LValue LV = EmitLValue(E->getSubExpr());
  return EmitScalarPrePostIncDec(E, LV, true, true);
}
```

Code sharing

- How do we keep CIR in sync with clang's LLVM IR codegen?
- Move common code into shared components
- Can we use templates or concepts to abstract IR-specific interfaces?



Getting help

- Upstreaming work requires discipline
- It doesn’t require prior in depth understanding
- A great way to learn!
- We’ve gotten a lot of help from great new contributors
- If you’d like to get involved, let us know!
- Come to the ClangIR roundtable tomorrow at 2:15

NVIDIA

Help Us Shape the Future

Meet us Tuesday during the LLVM Developers' Meeting 2025:

Tuesday: 3:00 Meet & Greet the LLVM Foundation Sponsors/California Ballroom

6:00 Evening Reception/Guildhouse

Scan the QR code to connect with us.
NVIDIA is the engine of the world's AI infrastructure. We are the world leader in accelerated computing.

Check out our open career opportunities here:
https://www.nvidia.com/en-us/about-nvidia/careers/

Meta

compilerjobs@meta.com