LLVM's First 25 Years - and the Road Ahead

Chris Lattner
LLVM Cofounder
CEO, Modular Inc.

LLVM Dev Meeting, Tokyo
June 2025

Today is a Historic Gathering

 Celebrating LLVM's impact across Asia-Pacific

 Next generation of LLVM contributors

 32nd LLVM Developer Meeting!

Looking Back & Celebrating

25 years of compiler evolution


Celebrating the community behind LLVM


Where will you take LLVM next?

2000–2005
Growing out of Academia

LLVM Prehistory (2000)

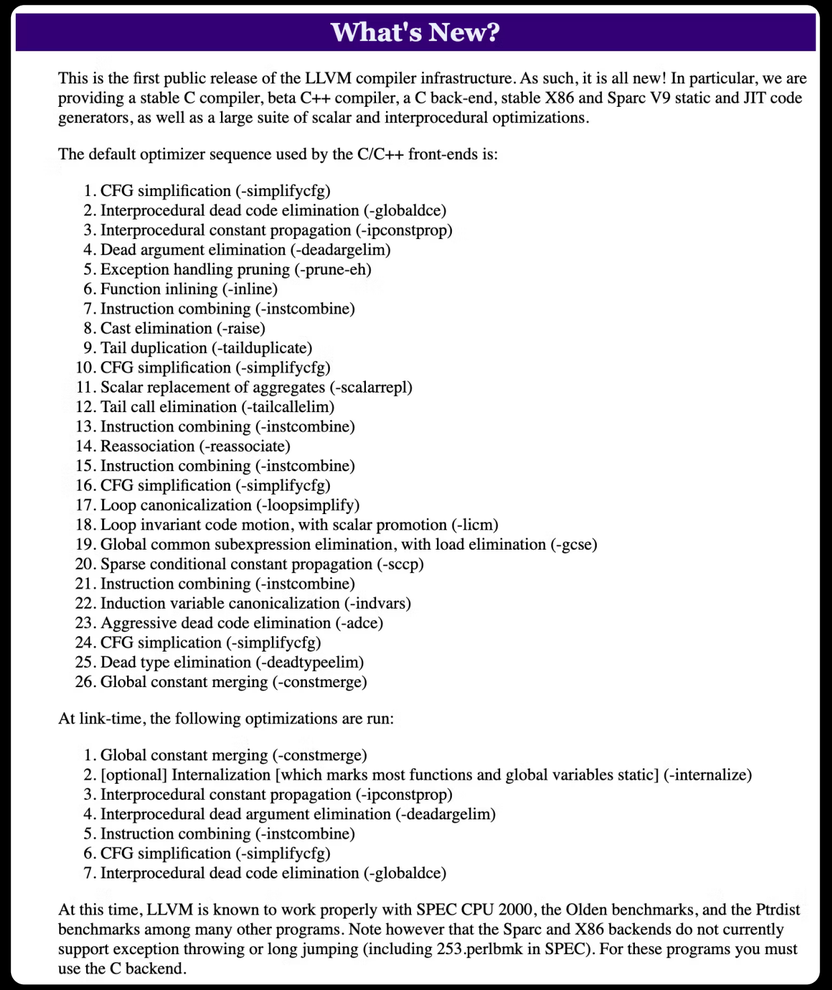
- Graduate research at University of Illinois under Vikram Adve
- Solve multi-stage compilation + optimization

"A brief history of LLVM"
/devmtg/2007-05/

LLVM 1.0 (2003)

- Sparc, X86, and C Backend
- llvm-gcc: "3.4-llvm 20030827 (experimental)"
- Worked: SPEC CPU2000, Olden, Ptrdist, ...
- 125K lines of code

Now open sourced!



What's New?

This is the first public release of the LLVM compiler infrastructure. As such, it is all new! In particular, we are providing a stable C compiler, beta C++ compiler, a C back-end, stable X86 and Sparc V9 static and JIT code generators, as well as a large suite of scalar and interprocedural optimizations.

The default optimizer sequence used by the C/C++ front-ends is:

1. CFG simplification (-simplifycfg)
2. Interprocedural dead code elimination (-globaldce)
3. Interprocedural constant propagation (-ipconstprop)
4. Dead argument elimination (-deadargelim)
5. Exception handling pruning (-prune-eh)
6. Function inlining (-inline)
7. Instruction combining (-instcombine)
8. Cast elimination (-raise)
9. Tail duplication (-tailduplicate)
10. CFG simplification (-simplifycfg)
11. Scalar replacement of aggregates (-scalarrepl)
12. Tail call elimination (-tailcallelim)
13. Instruction combining (-instcombine)
14. Reassociation (-reassociate)
15. Instruction combining (-instcombine)
16. CFG simplification (-simplifycfg)
17. Loop canonicalization (-loopsimplify)
18. Loop invariant code motion, with scalar promotion (-licm)
19. Global common subexpression elimination, with load elimination (-gcse)
20. Sparse conditional constant propagation (-sccp)
21. Instruction combining (-instcombine)
22. Induction variable canonicalization (-indvars)
23. Aggressive dead code elimination (-adce)
24. CFG simplication (-simplifycfg)
25. Dead type elimination (-deadtypeelim)
26. Global constant merging (-constmerge)

At link-time, the following optimizations are run:

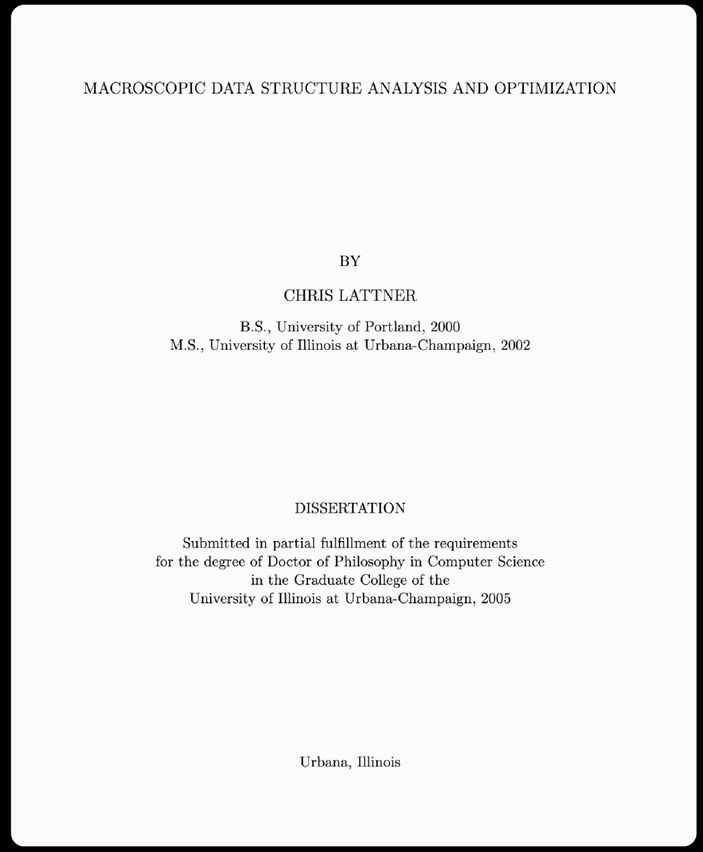
1. Global constant merging (-constmerge)
2. [optional] Internalization [which marks most functions and global variables static] (-internalize)
3. Interprocedural constant propagation (-ipconstprop)
4. Interprocedural dead argument elimination (-deadargelim)
5. Instruction combining (-instcombine)
6. CFG simplification (-simplifycfg)
7. Interprocedural dead code elimination (-globaldce)

At this time, LLVM is known to work properly with SPEC CPU 2000, the Olden benchmarks, and the Ptrdist benchmarks among many other programs. Note however that the Sparc and X86 backends do not currently support exception throwing or long jumping (including 253.perlbmk in SPEC). For these programs you must use the C backend.

releases.llvm.org/1.0/docs/ReleaseNotes.html

UIUC PhD (2005)

- Described LLVM + intermediate representation (IR)
- Proposed a foundation for modern compiler architecture
- Capstone showing pointer analysis research using LLVM



MACROSCOPIC DATA STRUCTURE ANALYSIS AND OPTIMIZATION

BY

CHRIS LATTNER

B.S., University of Portland, 2000
M.S., University of Illinois at Urbana-Champaign, 2002

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

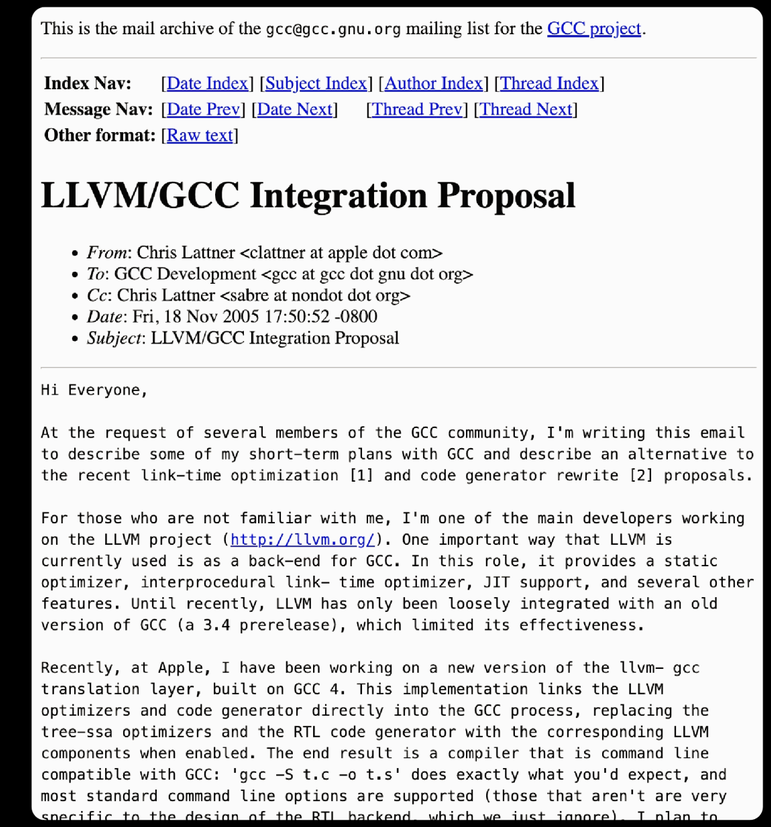
Urbana, Illinois

ideals.illinois.edu/items/11014

Should LLVM join GNU? (2005)

- Bridge between GCC and LLVM ecosystems
- Prove LLVM's viability as production backend
- Enabled migration path for existing codebases

... GCC / FSF decided they weren't interested



This is the mail archive of the gcc@gcc.gnu.org mailing list for the GCC project.

Index Nav: [Date Index] [Subject Index] [Author Index] [Thread Index]
Message Nav: [Date Prev] [Date Next] [Thread Prev] [Thread Next]
Other format: [Raw text]

LLVM/GCC Integration Proposal

- *From*: Chris Lattner <clattner at apple dot com>
- *To*: GCC Development <gcc at gcc dot gnu dot org>
- *Cc*: Chris Lattner <sabre at nondot dot org>
- *Date*: Fri, 18 Nov 2005 17:50:52 -0800
- *Subject*: LLVM/GCC Integration Proposal

Hi Everyone,

At the request of several members of the GCC community, I'm writing this email to describe some of my short-term plans with GCC and describe an alternative to the recent link-time optimization [1] and code generator rewrite [2] proposals.

For those who are not familiar with me, I'm one of the main developers working on the LLVM project (http://llvm.org/). One important way that LLVM is currently used is as a back-end for GCC. In this role, it provides a static optimizer, interprocedural link- time optimizer, JIT support, and several other features. Until recently, LLVM has only been loosely integrated with an old version of GCC (a 3.4 prerelease), which limited its effectiveness.

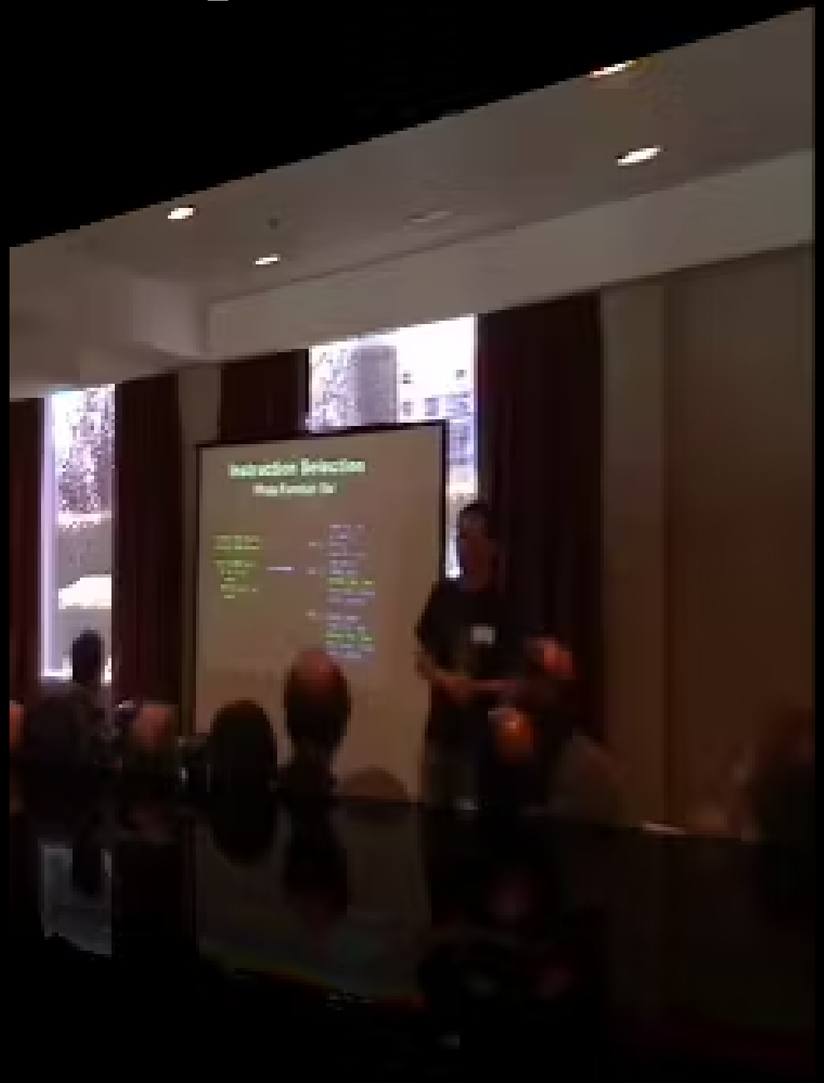
Recently, at Apple, I have been working on a new version of the llvm- gcc translation layer, built on GCC 4. This implementation links the LLVM optimizers and code generator directly into the GCC process, replacing the tree-ssa optimizers and the RTL code generator with the corresponding LLVM components when enabled. The end result is a compiler that is command line compatible with GCC: 'gcc -S t.c -o t.s' does exactly what you'd expect, and most standard command line options are supported (those that aren't are very specific to the design of the RTL backend, which we just ignore). I plan to

gcc.gnu.org/legacy-ml/gcc/2005-11/msg00888.html

2005–2008
Growing a community driven compiler

Code Generator (2007)

- Presenter: Evan Cheng
- Foundation for LLVM's multi-architecture support
- Unified framework for instruction selection and register allocation
- Enabled rapid addition of new processor backends



"The LLVM Code Generator"
llvm.org/devmtg/2007-05/

Clang Frontend (2007)

- Presenter: Steve Naroff
- Modern C/C++ frontend architecture
- Superior error messages and diagnostics
- Fast compilation times 🏎️

"New LLVM C Front-end"
llvm.org/devmtg/2007-05/

Clang Static Analyzer (2008)

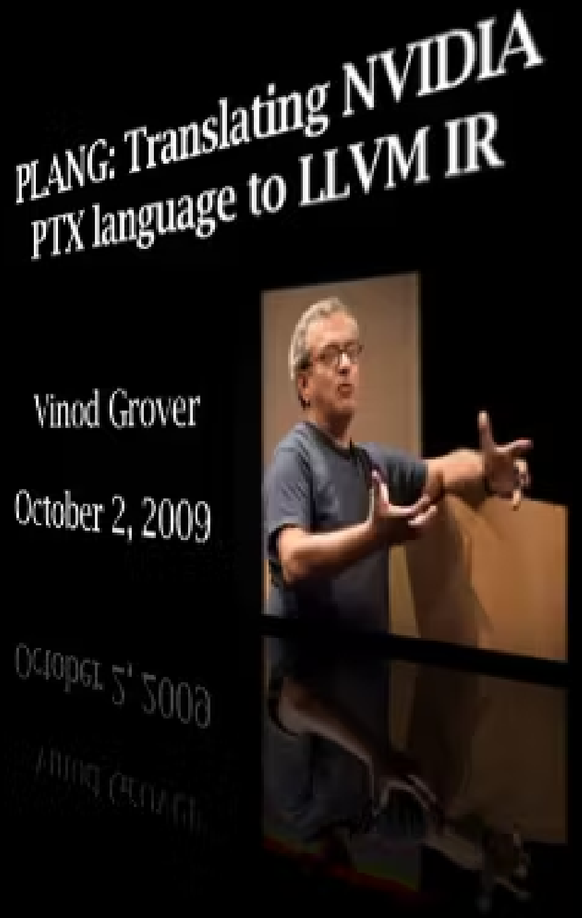
- Presenter: Ted Kremenek
- Advanced static analysis capabilities
- Path-sensitive bug detection
- Enhanced code quality and safety

"Finding Bugs with the Clang Static Analyzer"
llvm.org/devmtg/2008-08/

2009-2016
Growth & Adoption

NVIDIA PTX Backend (2009)

- Presenter: Vinod Grover
- GPU computing integration with LLVM
- NVIDIA PTX language support
- Heterogeneous computing capabilities



"PLANG: Translating NVIDIA PTX language to LLVM IR Machine"
llvm.org/devmtg/2009-10/

LLDB (2010)

- Presenter: Greg Clayton
- Modern debugger architecture
- Deep LLVM/Clang integration
- Superior debugging experience

"LLDB: Modular Debugging Infrastructure"
llvm.org/devmtg/2010-11/

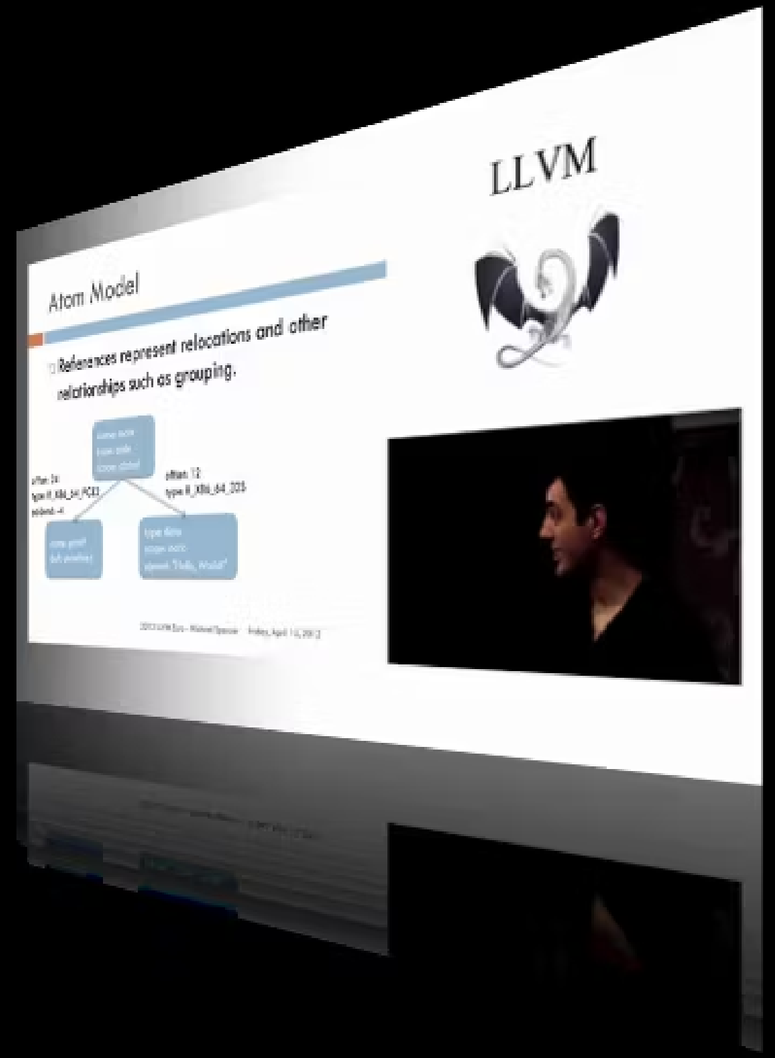
Sanitizers (2011)

- Presenter:
  Konstantin Serebryany
- Revolutionary bug detection tech
- Memory safety and thread safety analysis
- Production-ready dynamic analysis

"Finding races and memory errors with LLVM instrumentation"
llvm.org/devmtg/2011-11/#talk12

LLD (2012)

- Presenter: Michael Spencer
- Cross-platform, high-performance linker
- Native LLVM integration = faster linking than traditional linkers
- EuroLLVM's first full conference!



"lld - the LLVM Linker"
llvm.org/devmtg/2012-04-12/

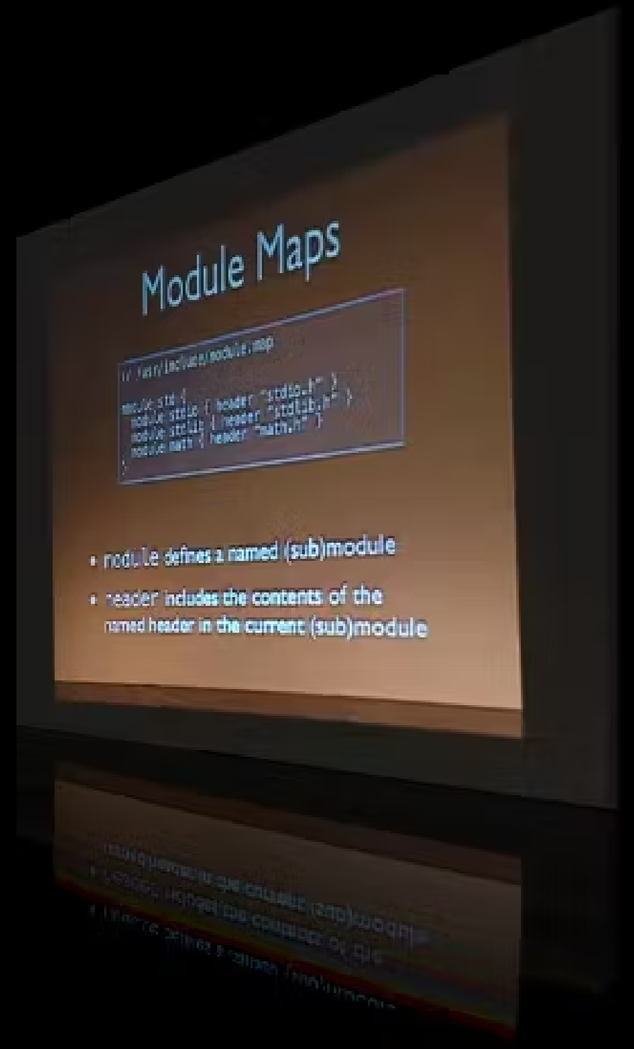
AArch64 Backend (2012)

- Presenter: Tim Northover
- Support for ARM's next-generation architecture
- 64-bit ARM processor enablement
- Mobile and embedded systems advancement

"The AArch64 backend: status and plans"
llvm.org/devmtg/2012-11/#talk2

Modules (2012)

- Presenter: Doug Gregor
- Solve header-file compilation problems
- Introduced Module Maps
- Set the path towards C++ standardization



"Modules"
llvm.org/devmtg/2012-11/#talk6

OpenMP (2013)

- Presenters: Andrey Bokhanko, Alexey Bataev
- Direct response to the growing need for multi-core and multi-threaded performance in C++
- Hat-tip to Polly for first experimenting with OpenMP!

"Towards OpenMP Support in LLVM"
llvm.org/devmtg/2013-04/#talk2

New Pass Manager (2014)

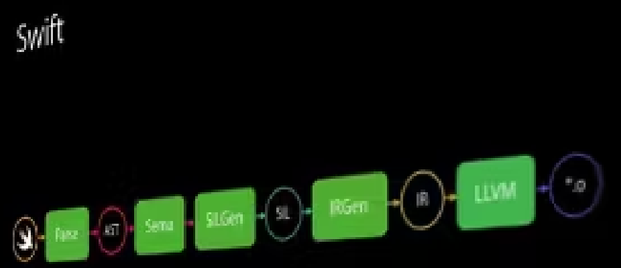
- Presenter: Chandler Carruth
- Improved optimization potential
  - Enabled efficient reuse of analysis results
  - Foundation for improved pass scheduling and composition
- Better optimization composition

```
class FunctionPassManager {
  typedef detail::PassConcept<Function *> FunctionPassConcept;

  template <typename PassT>
  struct FunctionPassModel : detail::PassModel<Function *, PassT> {
    FunctionPassModel(PassT Pass)
        : detail::PassModel<Function *, PassT>(std::move(Pass)) {}
  };

  std::vector<std::unique_ptr<FunctionPassConcept>> Passes;

public:
  template <typename FunctionPassT>
  void addPass(FunctionPassT Pass) {
    Passes.emplace_back(
        new FunctionPassModel<FunctionPassT>(std::move(Pass)));
  }

  void run(Function *F) {
    for (const auto &P : Passes)
      P->run(F);
  }
};
```

"The LLVM Pass Manager, Part 2"
llvm.org/devmtg/2014-10/#talk11

Swift & SIL (2015)

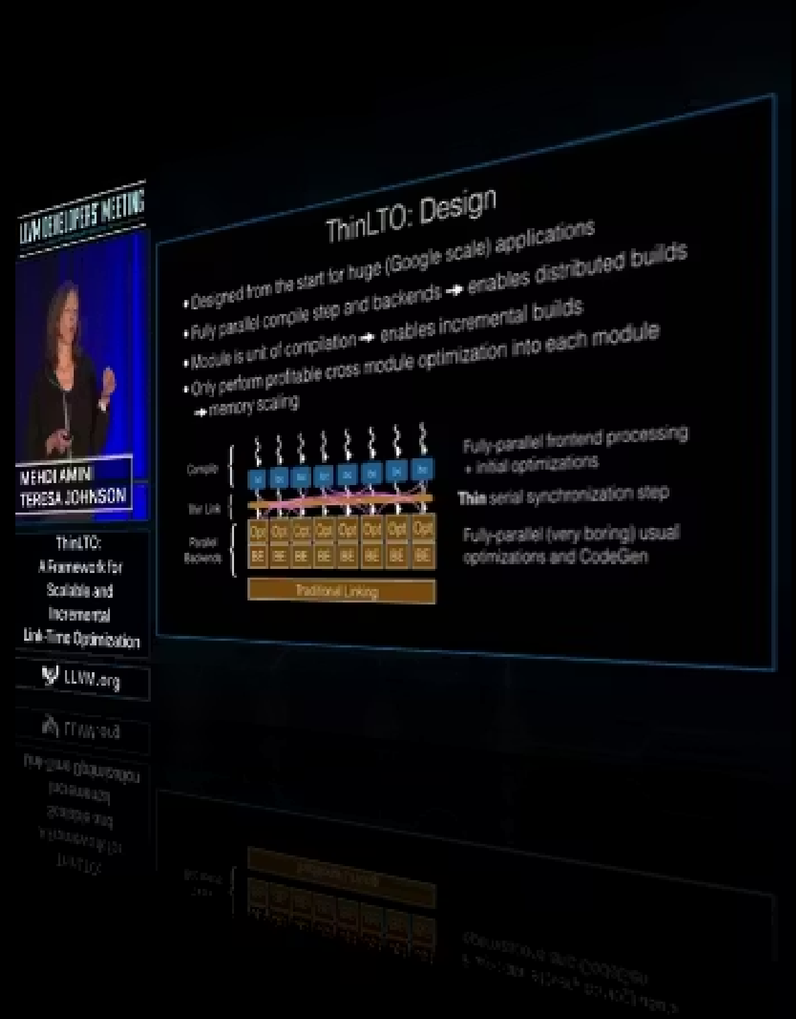
- Presenters: Joseph Groff, Chris Lattner
- High-level semantic analysis, devirtualization, and memory management optimizations
- Showed how LLVM IR is not enough: domain-specific IRs are important too



"Swift's High-Level IR: A Case Study of Complementing LLVM IR with Language-Specific Optimization"
llvm.org/devmtg/2015-10/#talk7

LTO Infrastructure (2016)

- Presenters: Teresa Johnson, Mehdi Amini
- Enabled hyperscale whole-program optimization
- Introduced summary-based analysis for efficient cross-module insights
- Significantly improved LTO's memory footprint and build times

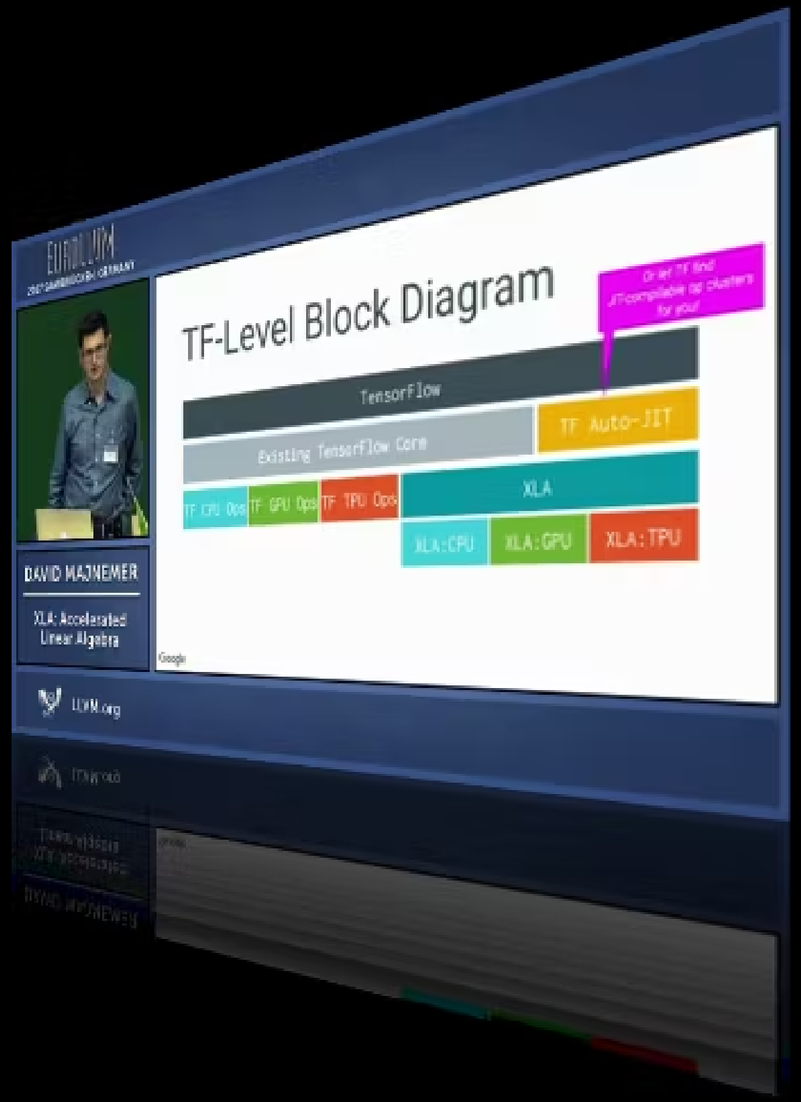


"ThinLTO: Scalable and Incremental LTO"
llvm.org/devmtg/2016-11/#talk12

2017–Present
MLIR and the Modern Era

XLA (2017)

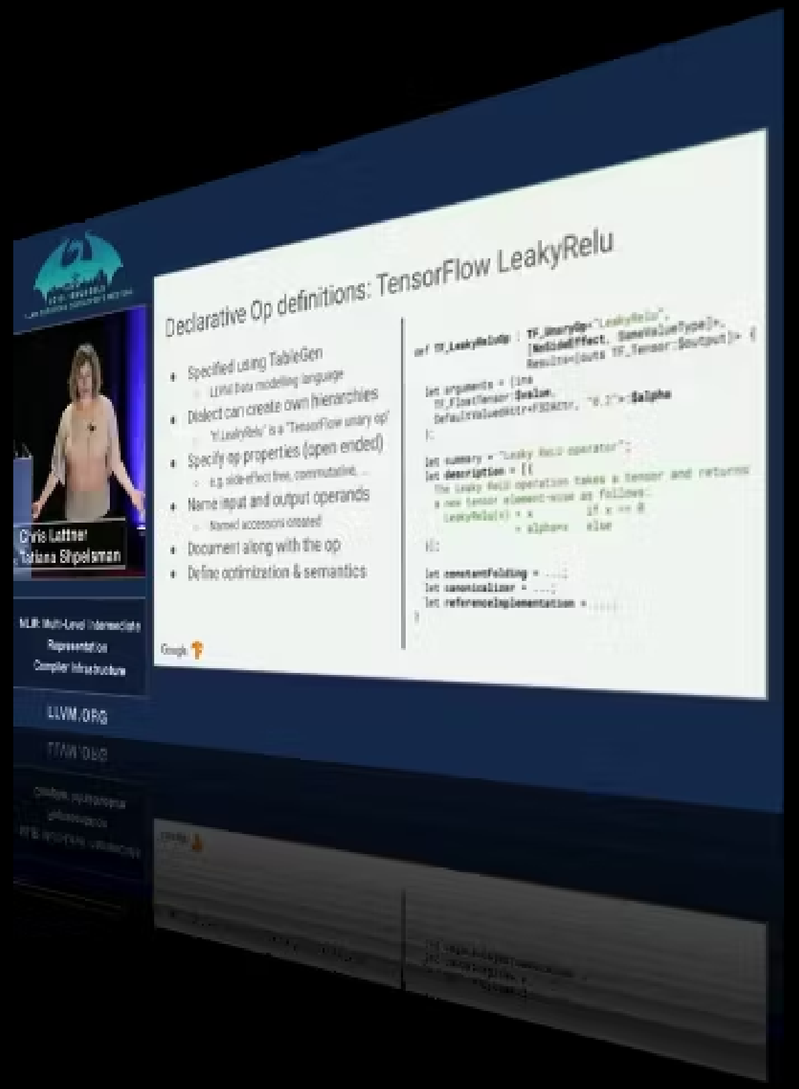
- Presenter: David Majnemer
- Pioneered domain-specific compilers built on LLVM infrastructure
- Automatic operation fusion and graph optimization
- JIT compilation targeting multiple hardware architectures



"XLA: Accelerated Linear Algebra"
llvm.org/devmtg/2017-03//2017/02/20/accepted-sessions.html#20

MLIR (2019)

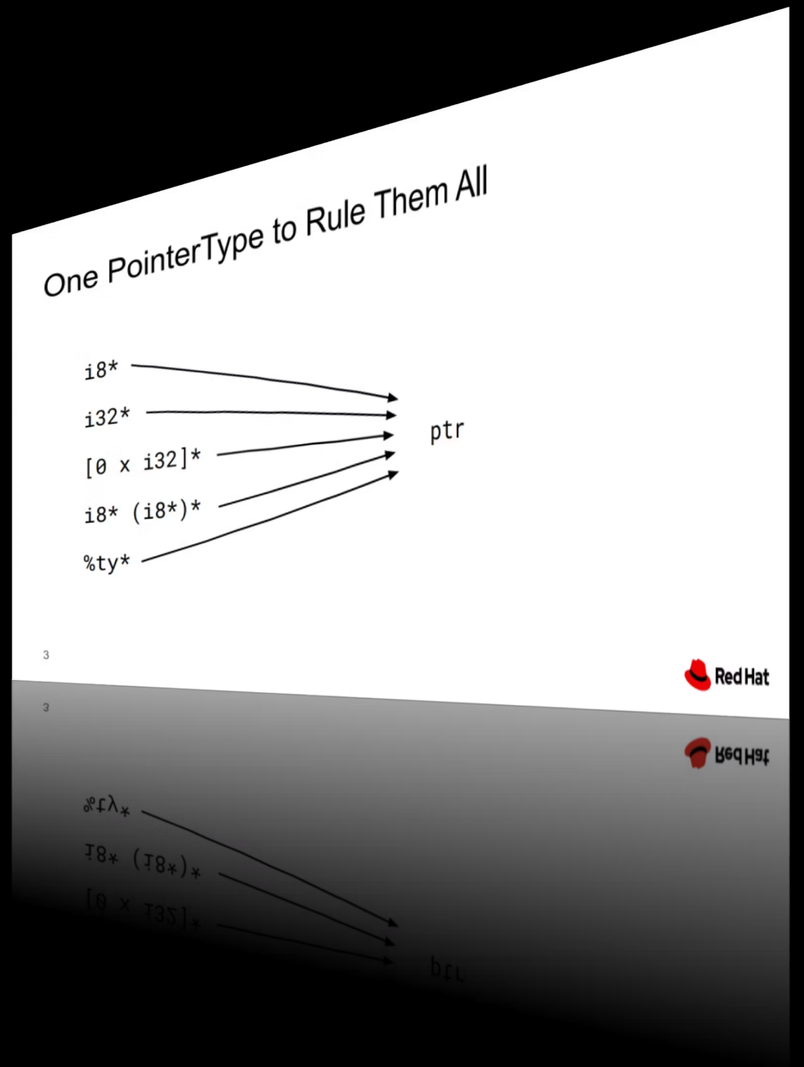
- Presenters: Tatiana Shpeisman, Chris Lattner
- Multi-level IR abstraction
- Extensible ops for modeling domain-specific abstractions
- Progressive lowering from high-level to machine-level IRs



"MLIR: Multi-Level Intermediate Representation"
llvm.org/devmtg/2019-04/talks.html#Keynote_1

Opaque Pointers (2022)

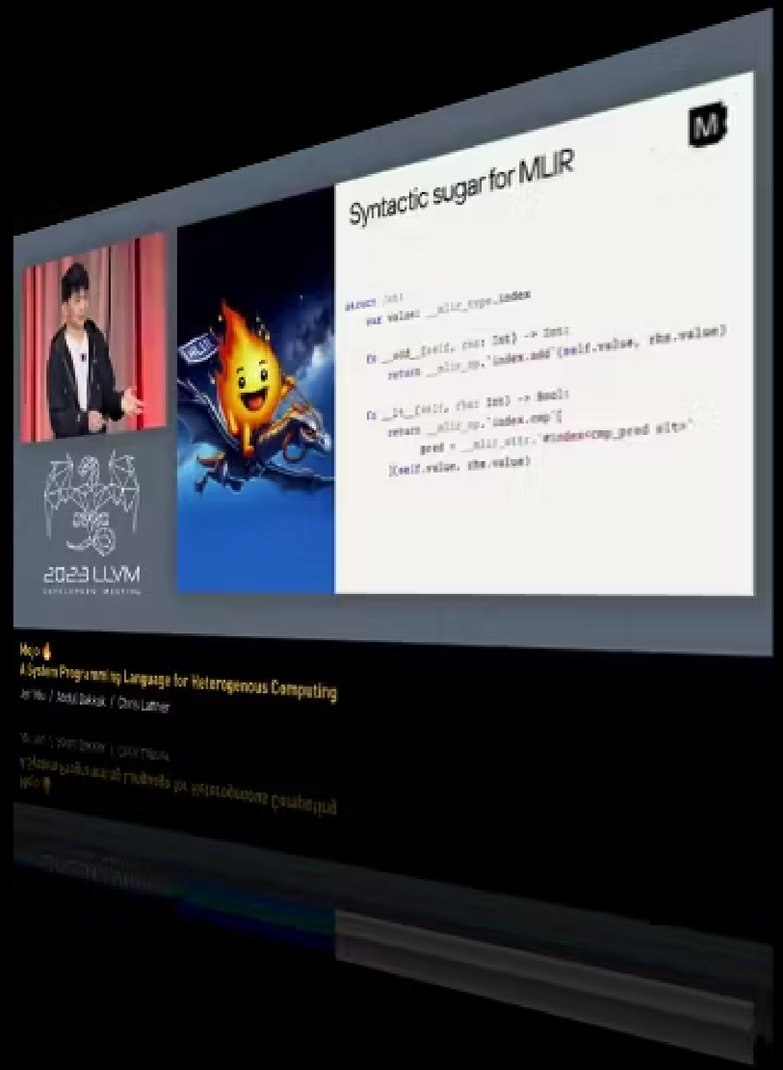
- Presenter: Nikita Popov
- Simplified LLVM IR design
- Reduced complexity in optimization passes
- True to a portable assembly



"Opaque Pointers Are Coming"
llvm.org/devmtg/2022-04-03/#keynote

Mojo 🔥 (2023)

- Presenters: Abdul Dakkak, Jeff Niu, Chris Lattner
- Python-family language design
- MLIR and LLVM foundation
- Heterogeneous computing capabilities



"Mojo 🔥: A system programming language for heterogenous computing"
llvm.org/devmtg/2023-10/

Rust ❤️ LLVM (2024)

- Presenter: Nikita Popov
- Dedicated Rust LLVM Working Group for systematic collaboration
- Active upstream issue resolution (inline assembly, optimization)
- LLVM 18 integration (PR #120055) with performance improvements

"Rust Heart LLVM"
llvm.org/devmtg/2024-10/

Today & The Road Ahead

May 9, 2025 – June 9, 2025

Period: 1 month

Overview

3,607 Active pull requests

1,261 Active issues

2,787	820	692	569
Merged pull requests	Open pull requests	Closed issues	New issues

Excluding merges, **667 authors** have pushed **3,372 commits** to main and **4,005 commits** to all branches. On main, **12,533 files** have changed and there have been **1,338,566 additions** and **554,592 deletions**.

2 Releases published by **1** person

llvmorg-20.1.5 LLVM 20.1.5
published last month

llvmorg-20.1.6 LLVM 20.1.6
published 2 weeks ago

2,787 Pull requests merged by **571** people

← Just one month on GitHub!

Thank You! Questions?

🙏 Thank you to the incredible LLVM community

🚀 25 years of innovation, and just getting started!

🗼 Excited for LLVM Asia's bright future