Paths towards unifying LLVM and MLIR

Nicolai Hähnle

AMD
together we advance_

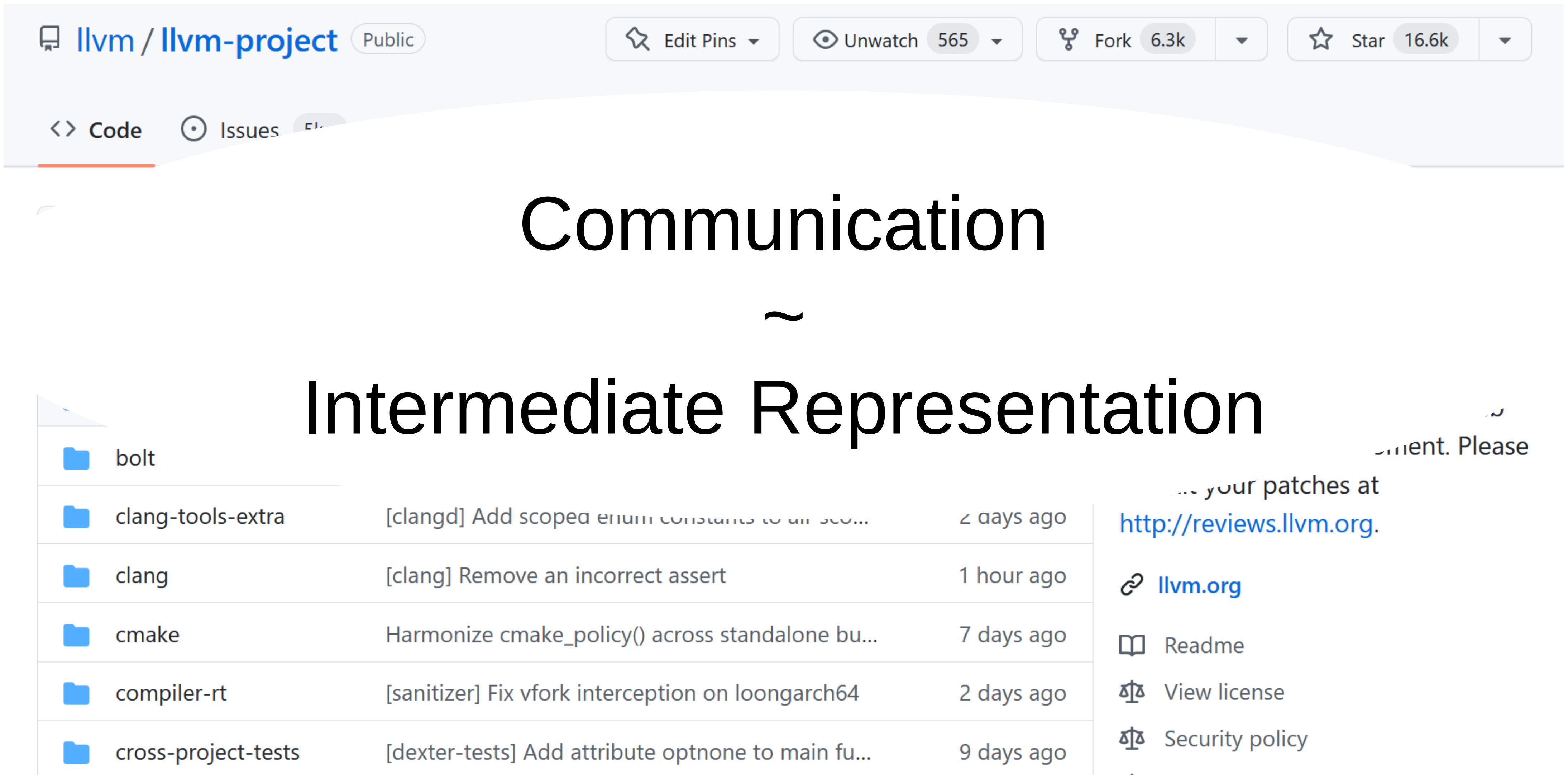
Part I

Where do you see yourself / the LLVM project

10 Years From Now?

“The Road Ahead” by DALL-E 2

“The LLVM project”



bolt		
clang-tools-extra	[clangd] Add scoped enum constants to all sco...	2 days ago
clang	[clang] Remove an incorrect assert	1 hour ago
cmake	Harmonize cmake_policy() across standalone bu...	7 days ago
compiler-rt	[sanitizer] Fix vfork interception on loongarch64	2 days ago
cross-project-tests	[dexter-tests] Add attribute optnone to main fu...	9 days ago

ment. Please
your patches at
http://reviews.llvm.org.

llvm.org

Readme

View license

Security policy

Communication

~

Intermediate Representation

arith func MLIR builtin vector

tensor pdl scf pdl_interp

async amx shape memref

math cf amdgpu gpu

sparse_tensor linalg index dlti

nvvm acc ml_program tosa

x86vector bufferization spirv omp

nvgpu arm_sve llvm complex

affine arm_neon rocdl emitc

quant transform

LLVM IR

Core MIR G-MIR RISCV-MIR

AArch64-MIR X86-MIR AMDGPU-MIR

WebAssembly-MIR ARC-MIR

PowerPC-MIR Hexagon-MIR

M68k-MIR SPIRV-MIR NVPTX-MIR

BPF-MIR ARM-MIR

CSKY-MIR Sparc-MIR Mips-MIR

SystemZ-MIR VE-MIR AVR-MIR

LoongArch-MIR MSP430-MIR

Lanai-MIR XCore-MIR

“The Tower of Babel” by DALL-E 2

arith func MLIR builtin vector
tensor pdl scf pdl_interp
async amx shape memref
math cf amdgpu gpu
sparse_tensor linalg index dlti
nvvm acc ml_program tosa
x86vector bufferization spirv omp
nvgpu arm_sve llvm complex
affine arm_neon rocdl emitc
quant transform

LLVM IR

Core MIR G-MIR RISCV-MIR
AArch64-MIR X86-MIR AMDGPU-MIR
WebAssembly-MIR ARC-MIR
Hexagon-MIR PowerPC-MIR
SPIRV-MIR NVPTX-MIR
M68k-MIR BPF-MIR
ARM-MIR CSKY-MIR
Sparc-MIR Mips-MIR SystemZ-MIR
VE-MIR AVR-MIR
LoongArch-MIR MSP430-MIR
Lanai-MIR XCore-MIR

“The Tower of Babel” by DALL-E 2

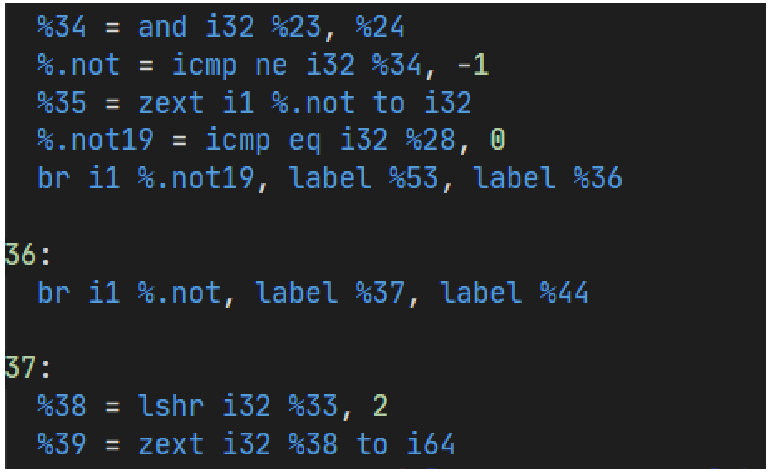
arith func MLIR builtin vector
tensor pdl scf pdl_interp
async amx shape memref
math cf amdgpu gpu
sparse_tensor linalg index dlti
nvvm acc ml_program tosa
x86vector bufferization spirv omp
nvgpu arm_sve llvm complex
affine arm_neon rocdl emitc
quant transform

SatMath Vector VP CheckedMath
ExtMath Core LLVM MemModel
LibC Swift FixPointMath
Coro GC Matrix llvm.amdgcn
FPEnv EH BitManip llvm.aarch64
VarArgs llvm.arm llvm.x86 llvm.ve
llvm.bpf llvm.nvvm llvm.ppc
llvm.s390 llvm.dx llvm.spv
llvm.wasm llvm.xcore llvm.riscv
llvm.hexagon llvm.mips llvm.r600

Core MIR G-MIR RISCV-MIR
AArch64-MIR X86-MIR AMDGPU-MIR
WebAssembly-MIR ARC-MIR
PowerPC-MIR Hexagon-MIR NVPTX-MIR
M68k-MIR SPIRV-MIR
BPF-MIR ARM-MIR
CSKY-MIR Sparc-MIR Mips-MIR
SystemZ-MIR
VE-MIR AVR-MIR
LoongArch-MIR
MSP430-MIR
Lanai-MIR XCore-MIR

“The Tower of Babel” by DALL-E 2

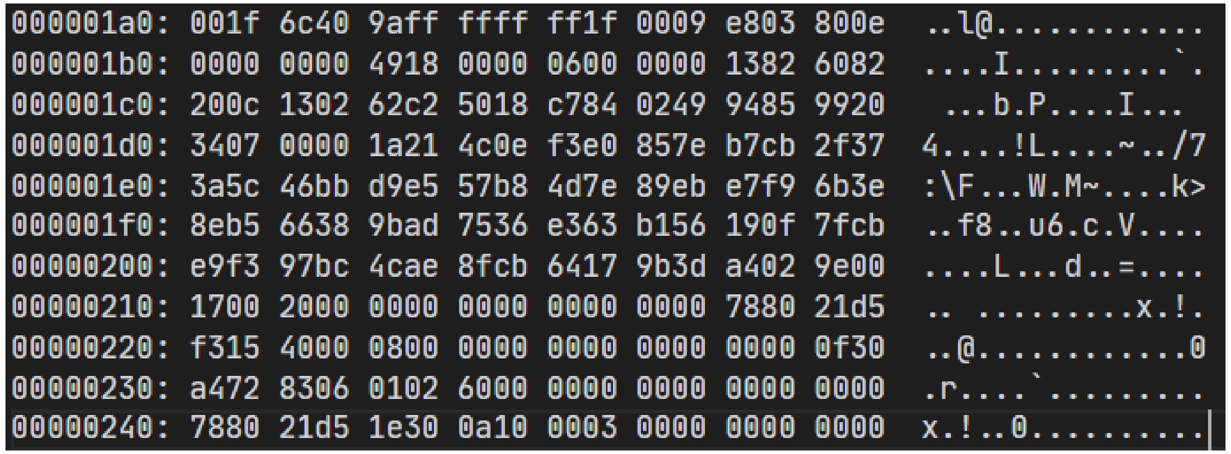
What is "LLVM IR"?



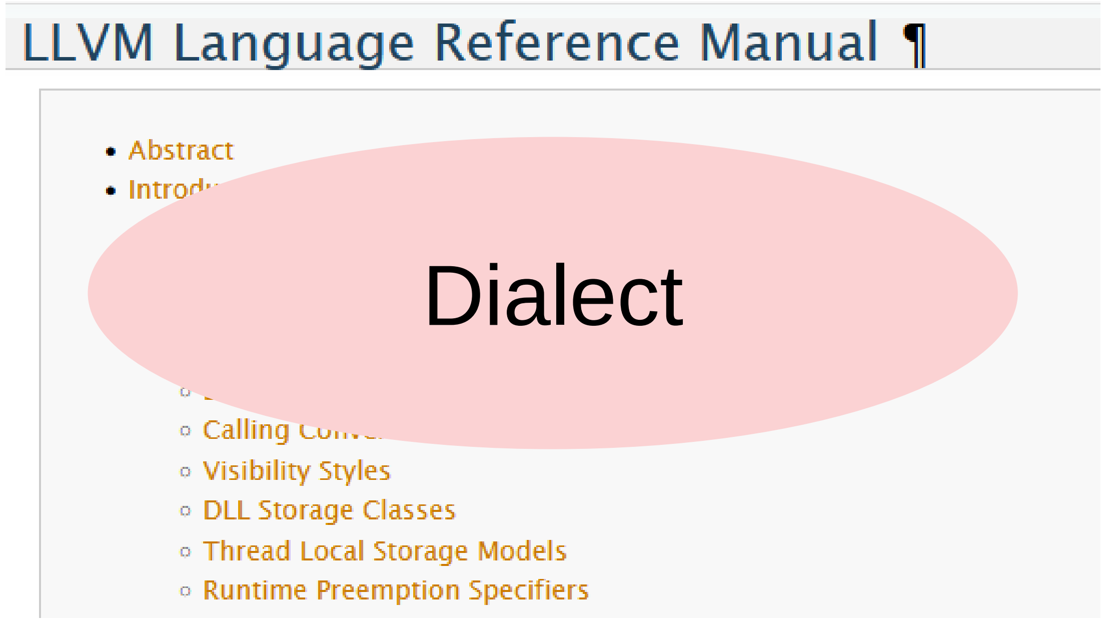
```
  %34 = and i32 %23, %24
  %.not = icmp ne i32 %34, -1
  %35 = zext i1 %.not to i32
  %.not19 = icmp eq i32 %28, 0
  br i1 %.not19, label %53, label %36

36:
  br i1 %.not, label %37, label %44

37:
  %38 = lshr i32 %33, 2
  %39 = zext i32 %38 to i64
```



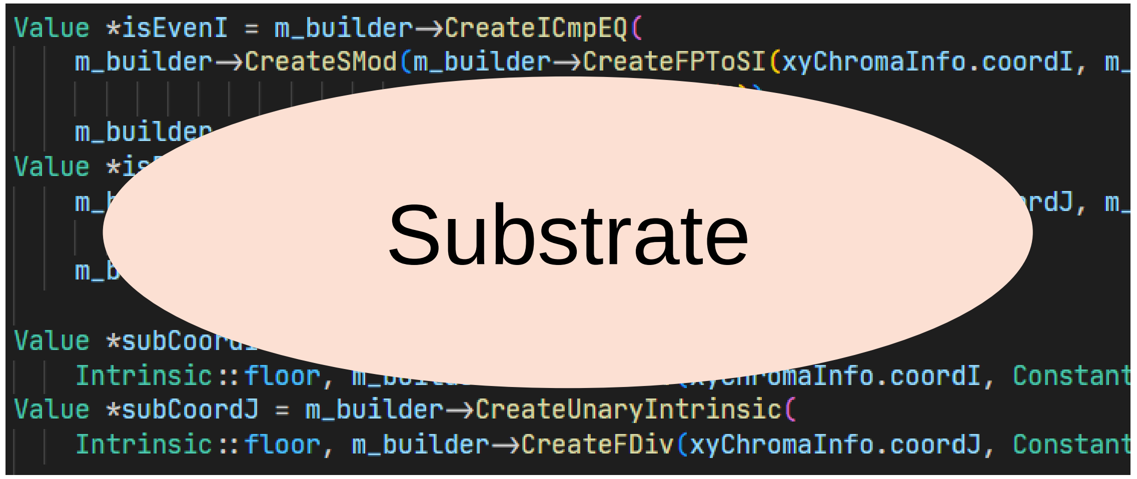
```
000001a0: 001f 6c40 9aff ffff ff1f 0009 e803 800e  ..l@............
000001b0: 0000 0000 4918 0000 0600 0000 1382 6082  ....I.........`.
000001c0: 200c 1302 62c2 5018 c784 0249 9485 9920   ...b.P....I...
000001d0: 3407 0000 1a21 4c0e f3e0 857e b7cb 2f37  4....!L....~../7
000001e0: 3a5c 46bb d9e5 57b8 4d7e 89eb e7f9 6b3e  :\F...W.M~....k>
000001f0: 8eb5 6638 9bad 7536 e363 b156 190f 7fcb  ..f8..u6.c.V....
00000200: e9f3 97bc 4cae 8fcb 6417 9b3d a402 9e00  ....L...d..=....
00000210: 1700 2000 0000 0000 0000 0000 7880 21d5  .. .........x.!.
00000220: f315 4000 0800 0000 0000 0000 0000 0f30  ..@............0
00000230: a472 8306 0102 6000 0000 0000 0000 0000  .r....`.........
00000240: 7880 21d5 1e30 0a10 0003 0000 0000 0000  x.!..0..........
```



LLVM Language Reference Manual ¶

- Abstract
- Introd
  - Calling Conv
  - Visibility Styles
  - DLL Storage Classes
  - Thread Local Storage Models
  - Runtime Preemption Specifiers

Dialect



```
Value *isEvenI = m_builder->CreateICmpEQ(
    m_builder->CreateSMod(m_builder->CreateFPToSI(xyChromaInfo.coordI, m_

    m_builder
Value *is
    m_                                                        rdJ, m_

    m_b

Value *subCoord
    Intrinsic::floor, m_                        omaInfo.coordI, Constant
Value *subCoordJ = m_builder->CreateUnaryIntrinsic(
    Intrinsic::floor, m_builder->CreateFDiv(xyChromaInfo.coordJ, Constant
```

Substrate

Dialect

Instruction set
Types
Semantics

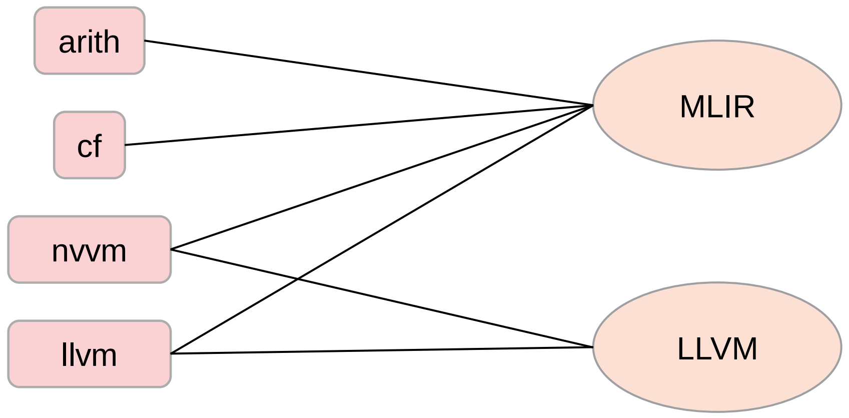
E.g.: LangRef.rst, mlir/Dialect.td

Substrate

Set of C++ classes to represent and manipulate code in one or more dialects

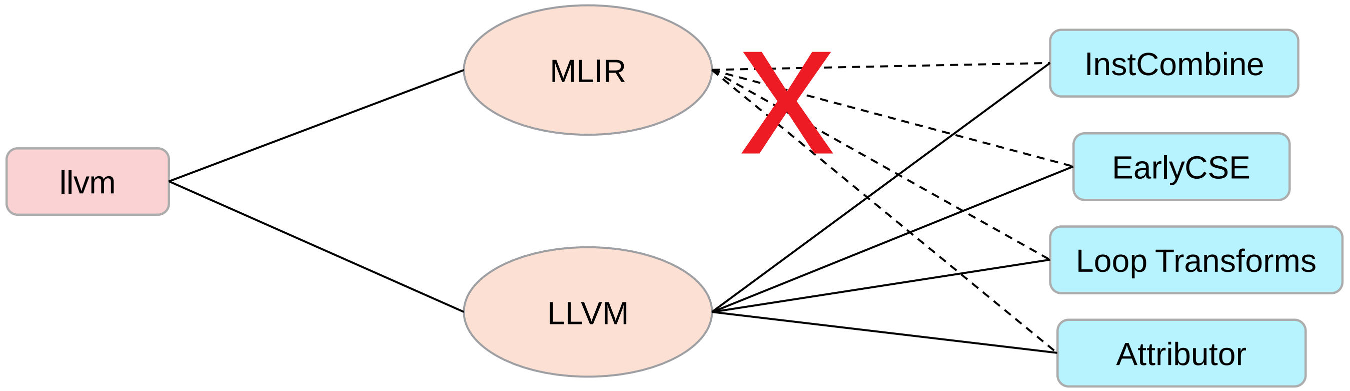
E.g.: llvm::Instruction, llvm::Value, llvm::BasicBlock, …

N:M relationship

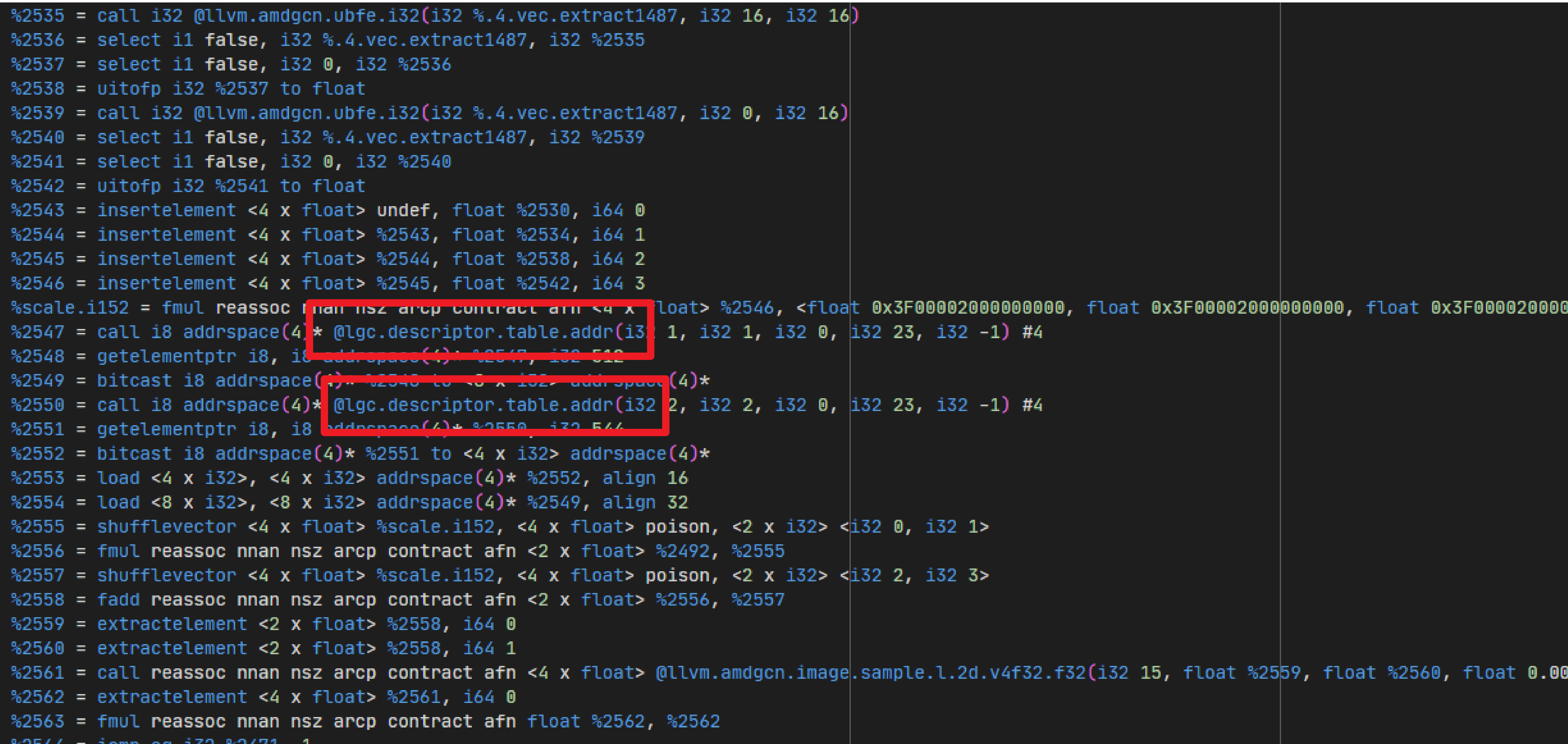


Substrates Matter

- Code artefacts (transforms, analyses, helpers, …) are written against a specific substrate
- Can represent llvm in MLIR, but can't run InstCombine or loop transforms on it



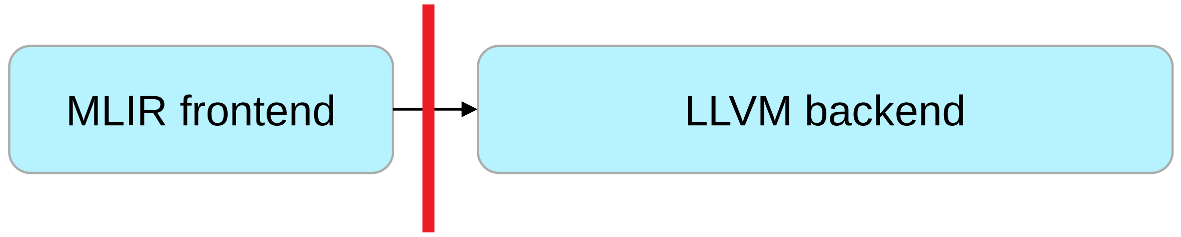
Substrates Matter



```
%2535 = call i32 @llvm.amdgcn.ubfe.i32(i32 %.4.vec.extract1487, i32 16, i32 16)
%2536 = select i1 false, i32 %.4.vec.extract1487, i32 %2535
%2537 = select i1 false, i32 0, i32 %2536
%2538 = uitofp i32 %2537 to float
%2539 = call i32 @llvm.amdgcn.ubfe.i32(i32 %.4.vec.extract1487, i32 0, i32 16)
%2540 = select i1 false, i32 %.4.vec.extract1487, i32 %2539
%2541 = select i1 false, i32 0, i32 %2540
%2542 = uitofp i32 %2541 to float
%2543 = insertelement <4 x float> undef, float %2530, i64 0
%2544 = insertelement <4 x float> %2543, float %2534, i64 1
%2545 = insertelement <4 x float> %2544, float %2538, i64 2
%2546 = insertelement <4 x float> %2545, float %2542, i64 3
%scale.i152 = fmul reassoc nnan nsz arcp contract afn <4 x float> %2546, <float 0x3F00002000000000, float 0x3F00002000000000, float 0x3F000020000
%2547 = call i8 addrspace(4)* @lgc.descriptor.table.addr(i32 1, i32 1, i32 0, i32 23, i32 -1) #4
%2548 = getelementptr i8, i8 addrspace(4)* %2547, i32 512
%2549 = bitcast i8 addrspace(4)* %2548 to <8 x i32> addrspace(4)*
%2550 = call i8 addrspace(4)* @lgc.descriptor.table.addr(i32 2, i32 2, i32 0, i32 23, i32 -1) #4
%2551 = getelementptr i8, i8 addrspace(4)* %2550, i32 544
%2552 = bitcast i8 addrspace(4)* %2551 to <4 x i32> addrspace(4)*
%2553 = load <4 x i32>, <4 x i32> addrspace(4)* %2552, align 16
%2554 = load <8 x i32>, <8 x i32> addrspace(4)* %2549, align 32
%2555 = shufflevector <4 x float> %scale.i152, <4 x float> poison, <2 x i32> <i32 0, i32 1>
%2556 = fmul reassoc nnan nsz arcp contract afn <2 x float> %2492, %2555
%2557 = shufflevector <4 x float> %scale.i152, <4 x float> poison, <2 x i32> <i32 2, i32 3>
%2558 = fadd reassoc nnan nsz arcp contract afn <2 x float> %2556, %2557
%2559 = extractelement <2 x float> %2558, i64 0
%2560 = extractelement <2 x float> %2558, i64 1
%2561 = call reassoc nnan nsz arcp contract afn <4 x float> @llvm.amdgcn.image.sample.l.2d.v4f32.f32(i32 15, float %2559, float %2560, float 0.00
%2562 = extractelement <4 x float> %2561, i64 0
%2563 = fmul reassoc nnan nsz arcp contract afn float %2562, %2562
%2564 = icmp eq i32 %2471, 1
```

Benefits of a Unified Substrate

- Interoperability of code
- Eliminate a hard pass-ordering boundary

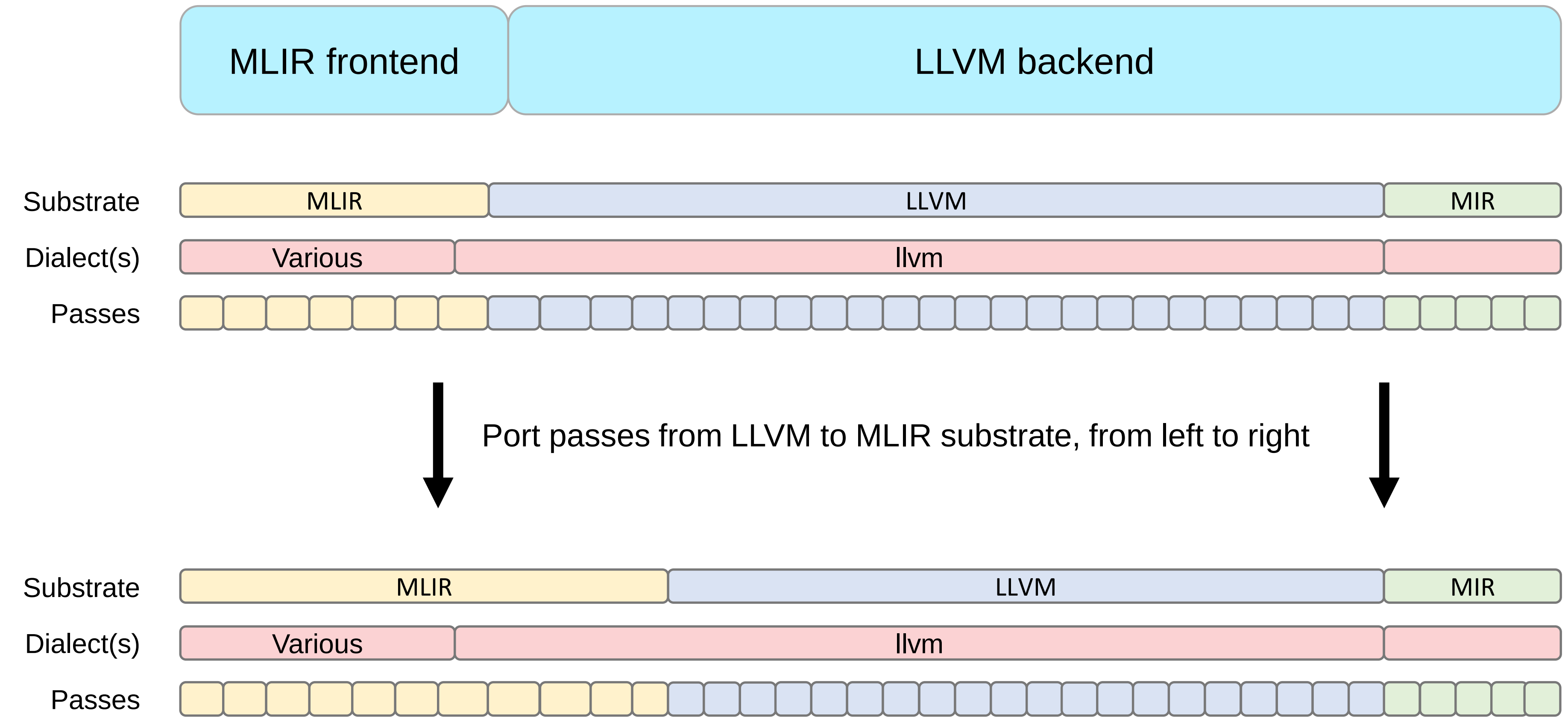


- Accessibility for developers
  - Shared knowledge and shared “language” within the LLVM project
  - Reduced friction for “full (compiler) stack developers”

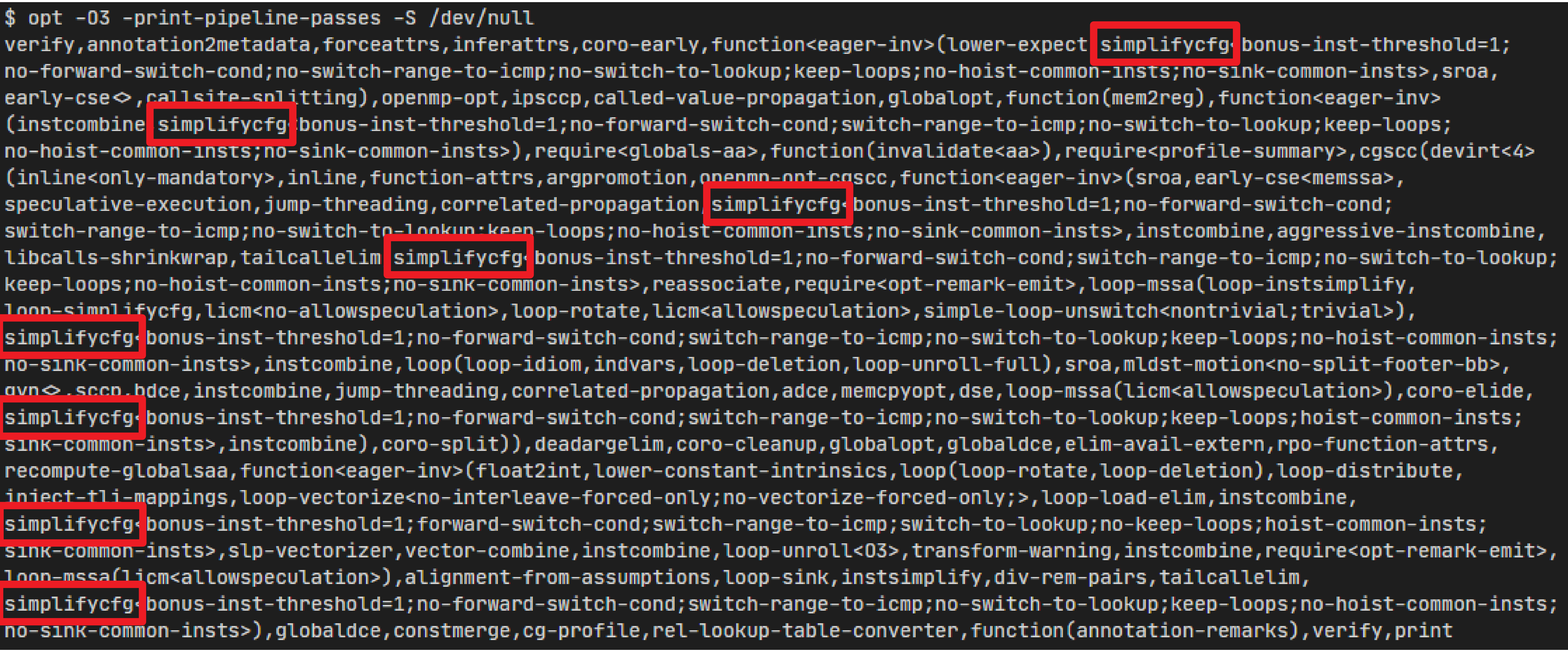
Part II
The Unification Landscape and the Delta Matrix

“Friendly Wyverns Shaking Hands” by DALL-E 2

Thought Experiment: Port LLVM Code to the MLIR Substrate

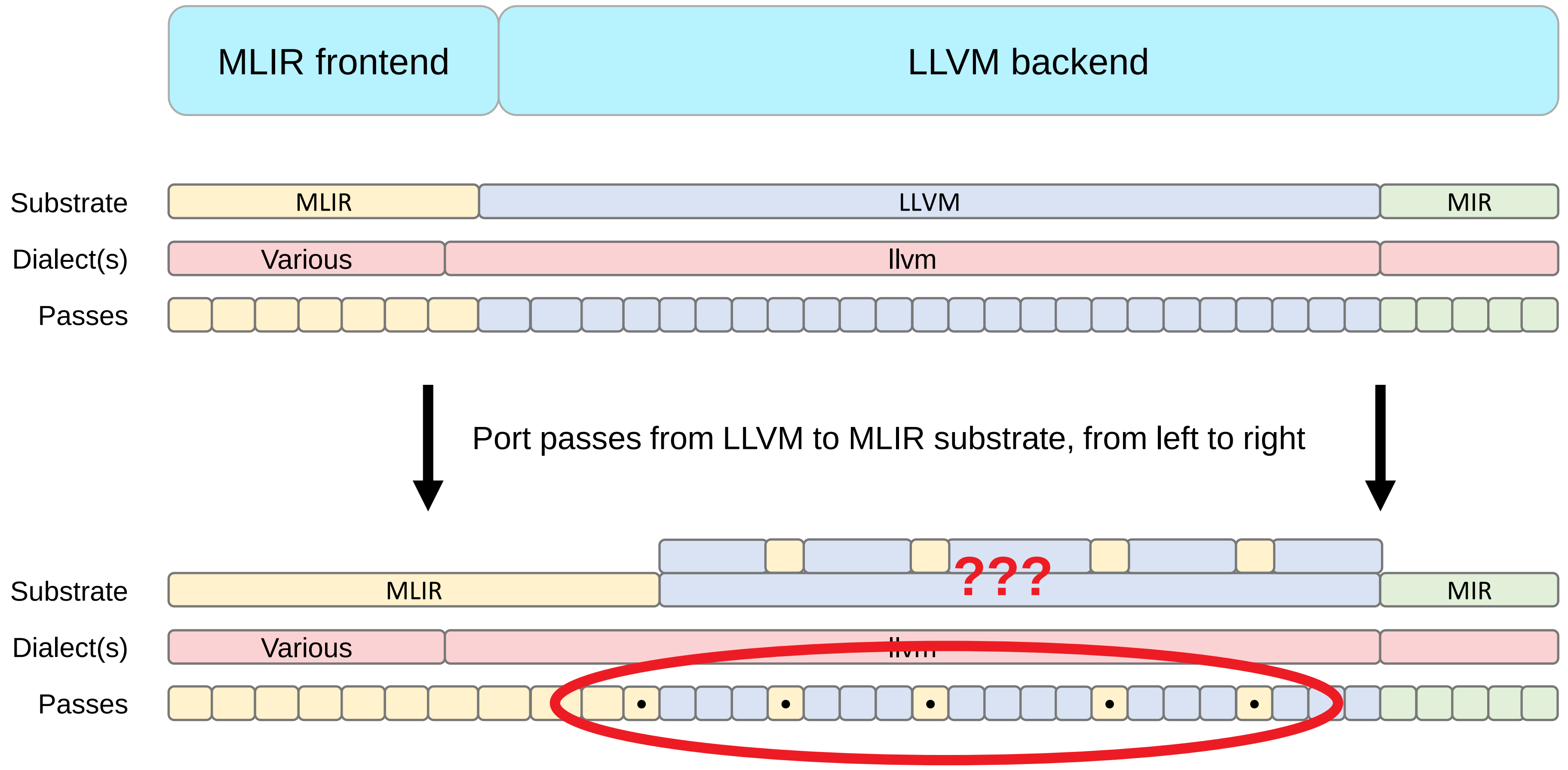


Reality Check: The Optimization Pass Pipeline



```
$ opt -O3 -print-pipeline-passes -S /dev/null
verify,annotation2metadata,forceattrs,inferattrs,coro-early,function<eager-inv>(lower-expect,simplifycfg<bonus-inst-threshold=1;
no-forward-switch-cond;no-switch-range-to-icmp;no-switch-to-lookup;keep-loops;no-hoist-common-insts;no-sink-common-insts>,sroa,
early-cse<>,callsite-splitting),openmp-opt,ipsccp,called-value-propagation,globalopt,function(mem2reg),function<eager-inv>
(instcombine,simplifycfg<bonus-inst-threshold=1;no-forward-switch-cond;switch-range-to-icmp;no-switch-to-lookup;keep-loops;
no-hoist-common-insts;no-sink-common-insts>),require<globals-aa>,function(invalidate<aa>),require<profile-summary>,cgscc(devirt<4>
(inline<only-mandatory>,inline,function-attrs,argpromotion,openmp-opt-cgscc,function<eager-inv>(sroa,early-cse<memssa>,
speculative-execution,jump-threading,correlated-propagation,simplifycfg<bonus-inst-threshold=1;no-forward-switch-cond;
switch-range-to-icmp;no-switch-to-lookup;keep-loops;no-hoist-common-insts;no-sink-common-insts>,instcombine,aggressive-instcombine,
libcalls-shrinkwrap,tailcallelim,simplifycfg<bonus-inst-threshold=1;no-forward-switch-cond;switch-range-to-icmp;no-switch-to-lookup;
keep-loops;no-hoist-common-insts;no-sink-common-insts>,reassociate,require<opt-remark-emit>,loop-mssa(loop-instsimplify,
loop-simplifycfg,licm<no-allowspeculation>,loop-rotate,licm<allowspeculation>,simple-loop-unswitch<nontrivial;trivial>),
simplifycfg<bonus-inst-threshold=1;no-forward-switch-cond;switch-range-to-icmp;no-switch-to-lookup;keep-loops;no-hoist-common-insts;
no-sink-common-insts>,instcombine,loop(loop-idiom,indvars,loop-deletion,loop-unroll-full),sroa,mldst-motion<no-split-footer-bb>,
gvn<>,sccp,bdce,instcombine,jump-threading,correlated-propagation,adce,memcpyopt,dse,loop-mssa(licm<allowspeculation>),coro-elide,
simplifycfg<bonus-inst-threshold=1;no-forward-switch-cond;switch-range-to-icmp;no-switch-to-lookup;keep-loops;hoist-common-insts;
sink-common-insts>,instcombine),coro-split)),deadargelim,coro-cleanup,globalopt,globaldce,elim-avail-extern,rpo-function-attrs,
recompute-globalsaa,function<eager-inv>(float2int,lower-constant-intrinsics,loop(loop-rotate,loop-deletion),loop-distribute,
inject-tli-mappings,loop-vectorize<no-interleave-forced-only;no-vectorize-forced-only;>,loop-load-elim,instcombine,
simplifycfg<bonus-inst-threshold=1;forward-switch-cond;switch-range-to-icmp;switch-to-lookup;no-keep-loops;hoist-common-insts;
sink-common-insts>,slp-vectorizer,vector-combine,instcombine,loop-unroll<O3>,transform-warning,instcombine,require<opt-remark-emit>,
loop-mssa(licm<allowspeculation>),alignment-from-assumptions,loop-sink,instsimplify,div-rem-pairs,tailcallelim,
simplifycfg<bonus-inst-threshold=1;no-forward-switch-cond;switch-range-to-icmp;no-switch-to-lookup;keep-loops;no-hoist-common-insts;
no-sink-common-insts>),globaldce,constmerge,cg-profile,rel-lookup-table-converter,function(annotation-remarks),verify,print
```

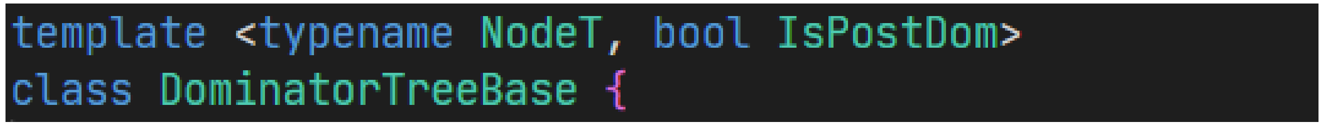
Thought Experiment: Port LLVM Code to the MLIR Substrate



- Many substrate round-trips (compile-time cost!) or code duplication (maintenance cost!)

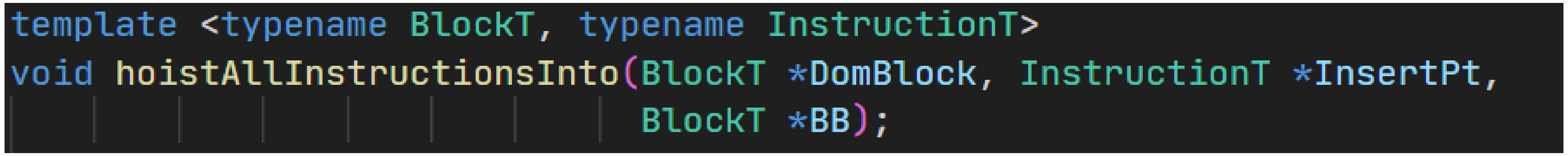
Thought Experiment: Make Code Generic over the Substrate

- Generic code already exists. For example:



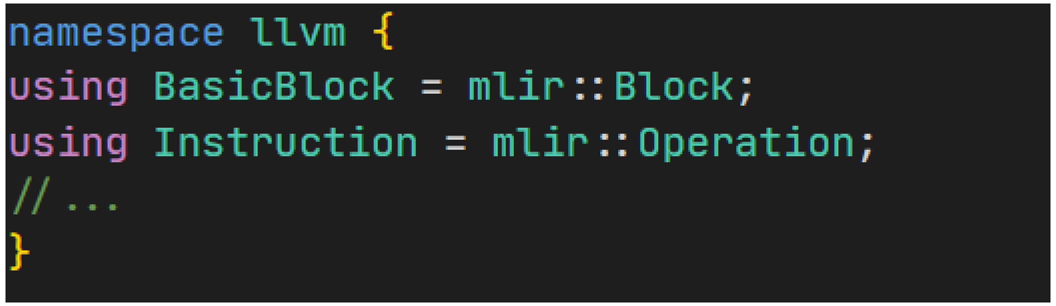
```
template <typename NodeT, bool IsPostDom>
class DominatorTreeBase {
```

Expand use of this technique. For example:

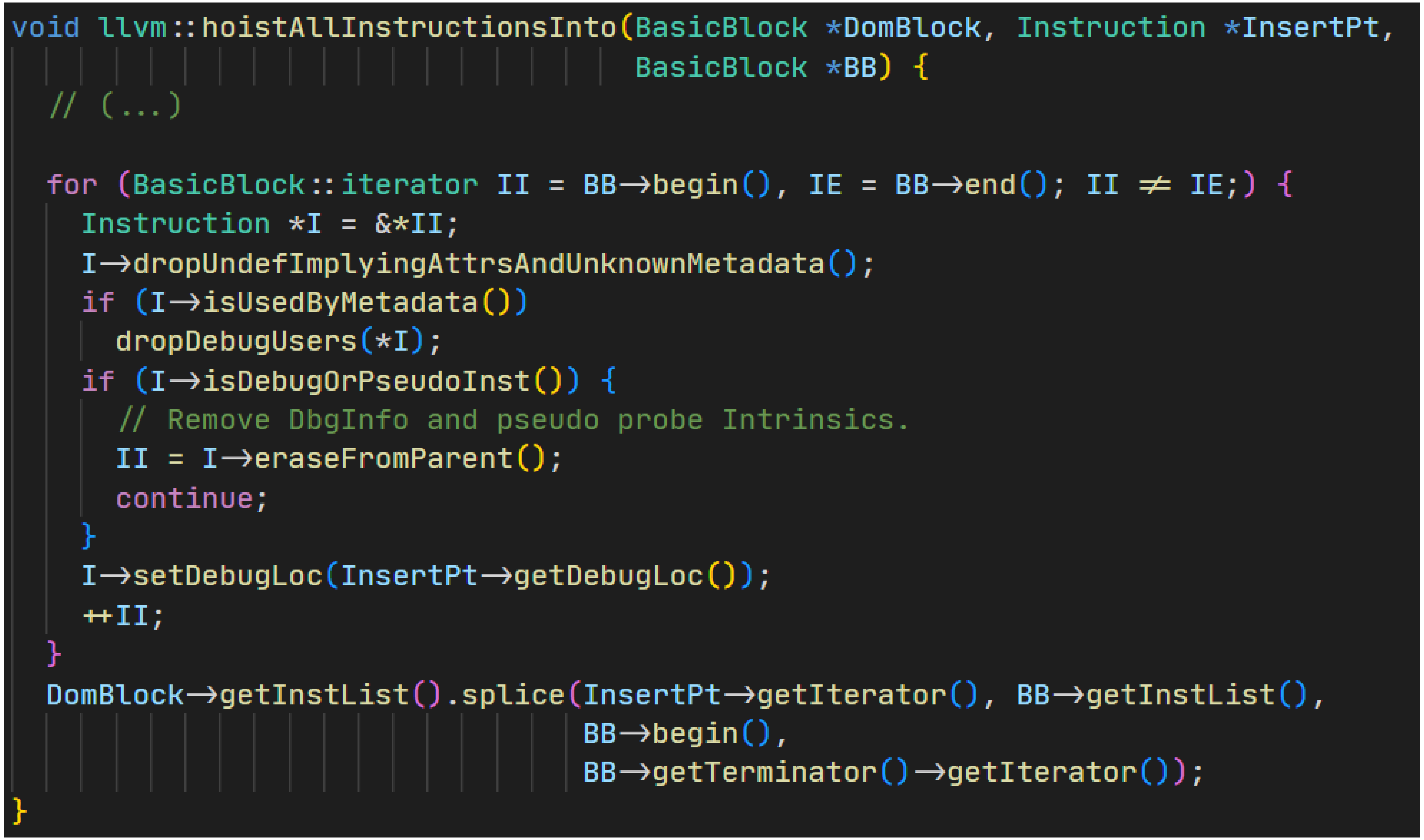


```
template <typename BlockT, typename InstructionT>
void hoistAllInstructionsInto(BlockT *DomBlock, InstructionT *InsertPt,
                              BlockT *BB);
```

Eventually:

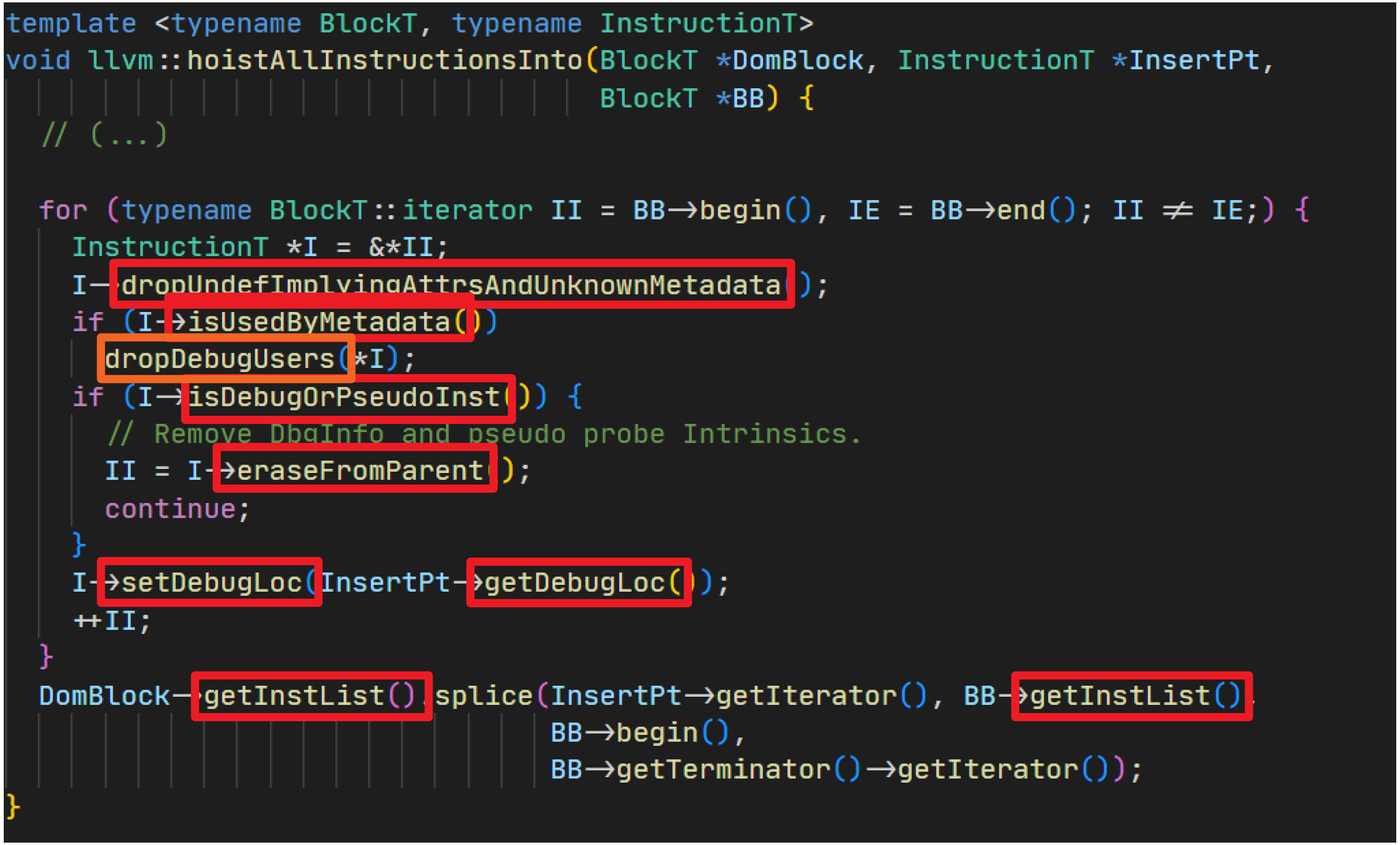


```
namespace llvm {
using BasicBlock = mlir::Block;
using Instruction = mlir::Operation;
// ...
}
```



```cpp
void llvm::hoistAllInstructionsInto(BasicBlock *DomBlock, Instruction *InsertPt,
                                    BasicBlock *BB) {
  // (...)

  for (BasicBlock::iterator II = BB->begin(), IE = BB->end(); II != IE;) {
    Instruction *I = &*II;
    I->dropUndefImplyingAttrsAndUnknownMetadata();
    if (I->isUsedByMetadata())
      dropDebugUsers(*I);
    if (I->isDebugOrPseudoInst()) {
      // Remove DbgInfo and pseudo probe Intrinsics.
      II = I->eraseFromParent();
      continue;
    }
    I->setDebugLoc(InsertPt->getDebugLoc());
    ++II;
  }
  DomBlock->getInstList().splice(InsertPt->getIterator(), BB->getInstList(),
                                 BB->begin(),
                                 BB->getTerminator()->getIterator());
}
```

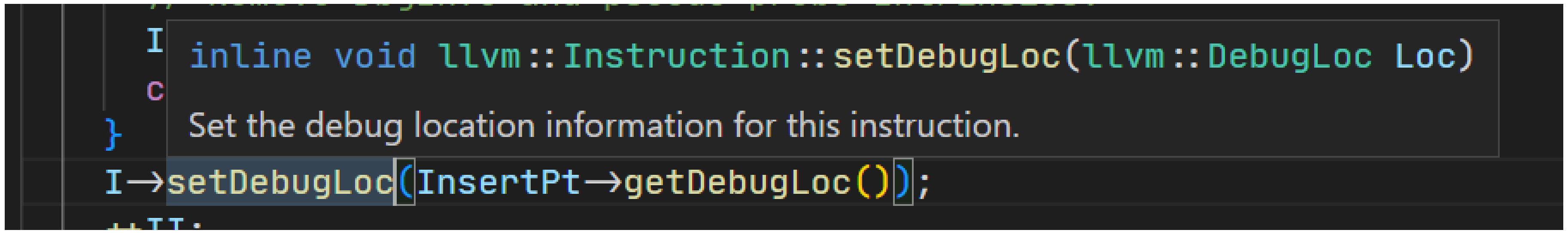


```
template <typename BlockT, typename InstructionT>
void llvm::hoistAllInstructionsInto(BlockT *DomBlock, InstructionT *InsertPt,
                                    BlockT *BB) {
  // (...)

  for (typename BlockT::iterator II = BB->begin(), IE = BB->end(); II != IE;) {
    InstructionT *I = &*II;
    I->dropUndefImplyingAttrsAndUnknownMetadata();
    if (I->isUsedByMetadata())
      dropDebugUsers(*I);
    if (I->isDebugOrPseudoInst()) {
      // Remove DbgInfo and pseudo probe Intrinsics.
      II = I->eraseFromParent();
      continue;
    }
    I->setDebugLoc(InsertPt->getDebugLoc());
    ++II;
  }
  DomBlock->getInstList().splice(InsertPt->getIterator(), BB->getInstList(),
                                 BB->begin(),
                                 BB->getTerminator()->getIterator());
}
```

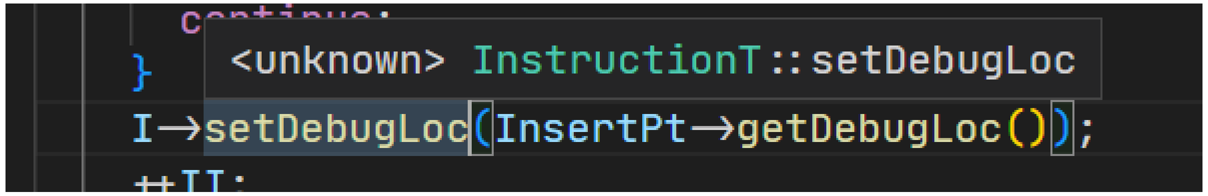
Tooling for Templates is Weak

- Error messages
- Compile times
- Language servers



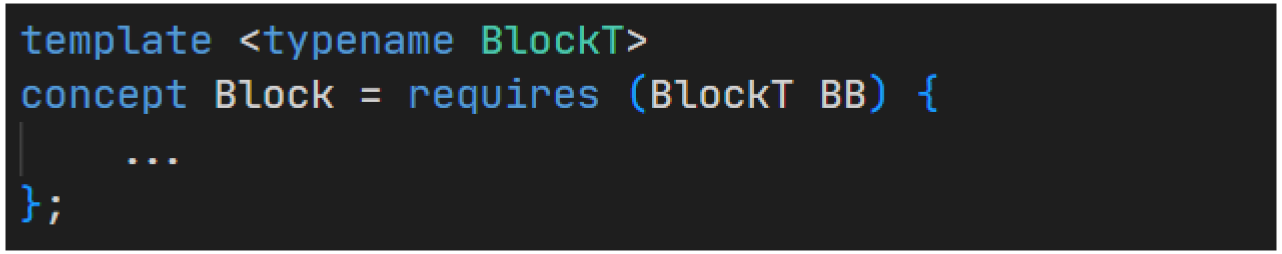
```
inline void llvm::Instruction::setDebugLoc(llvm::DebugLoc Loc)
Set the debug location information for this instruction.
I->setDebugLoc(InsertPt->getDebugLoc());
```

vs.

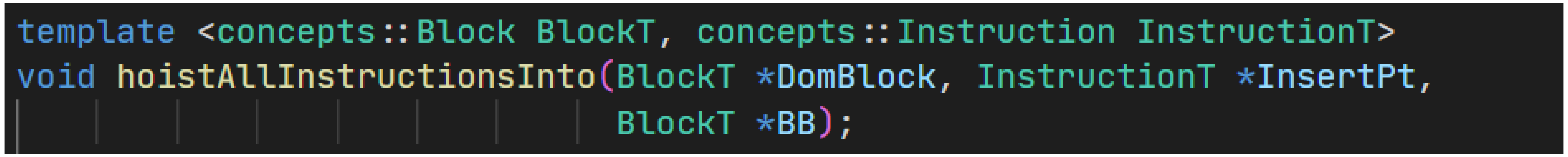


```
<unknown> InstructionT::setDebugLoc
I->setDebugLoc(InsertPt->getDebugLoc());
```

C++ concepts



```
template <typename BlockT>
concept Block = requires (BlockT BB) {
    ...
};
```



```
template <concepts::Block BlockT, concepts::Instruction InstructionT>
void hoistAllInstructionsInto(BlockT *DomBlock, InstructionT *InsertPt,
                              BlockT *BB);
```

- Example: “Block” concept
  - CFG predecessors and successors
  - Instruction list and terminators
  - Basic block arguments
  - …
- Not as good as type classes or traits; and requires C++20

The Delta Matrix

Clang									
Flang									
SimplifyCFG									
InstCombine									
Attributor									
Coroutines									
AddressSanitizer									
Alias analysis									
Target codegen									
Assembly printer & parser									
Bitcode writer & reader									
IR linker									
lldb									
JIT infrastructure									
llvm-reduce									
…									

The Delta Matrix

	Constants	Multiple defined values	Phi nodes vs. BB args	Builder interface	Regions	Attributes	Debug info	Metadata	…
Clang									
Flang									
SimplifyCFG									
InstCombine									
Attributor									
Coroutines									
AddressSanitizer									
Alias analysis									
Target codegen									
Assembly printer & parser									
Bitcode writer & reader									
IR linker									
lldb									
llvm-reduce									
lli									
…									

llvm::Value

“Dragon on a Pile of Gold Coins” by DALL-E 2

Part III
Column Refactorings

“Columns with Excavator” by DALL-E 2

Example 1: Dialects in LLVM

1. First-class support for extended instructions and types, with near-MLIR-compatible definitions
2. Cleanup of “the LLVM dialect” into constituent dialects
3. Extensibility for frontends, including external frontends

First-class support for extended instructions and types

- Intrinsics are second-class citizens in textual IR

```
%x = add i32 %y, %z
```

vs.

```
%x = call i32 @llvm.smin.i32(i32 %y, i32 %z)
```

- Intrinsics are second-class citizens in C++

```
if (auto *Store = dyn_cast<StoreInst>(I)) {
  use(Store->getPointerOperand());
}
```

vs.

```
if (auto *Intr = dyn_cast<IntrinsicInst>(I)) {
  if (Intr->getIntrinsicID() == Intrinsic::prefetch) {
    use(Intr->getArgOperand(0));
  }
}
```

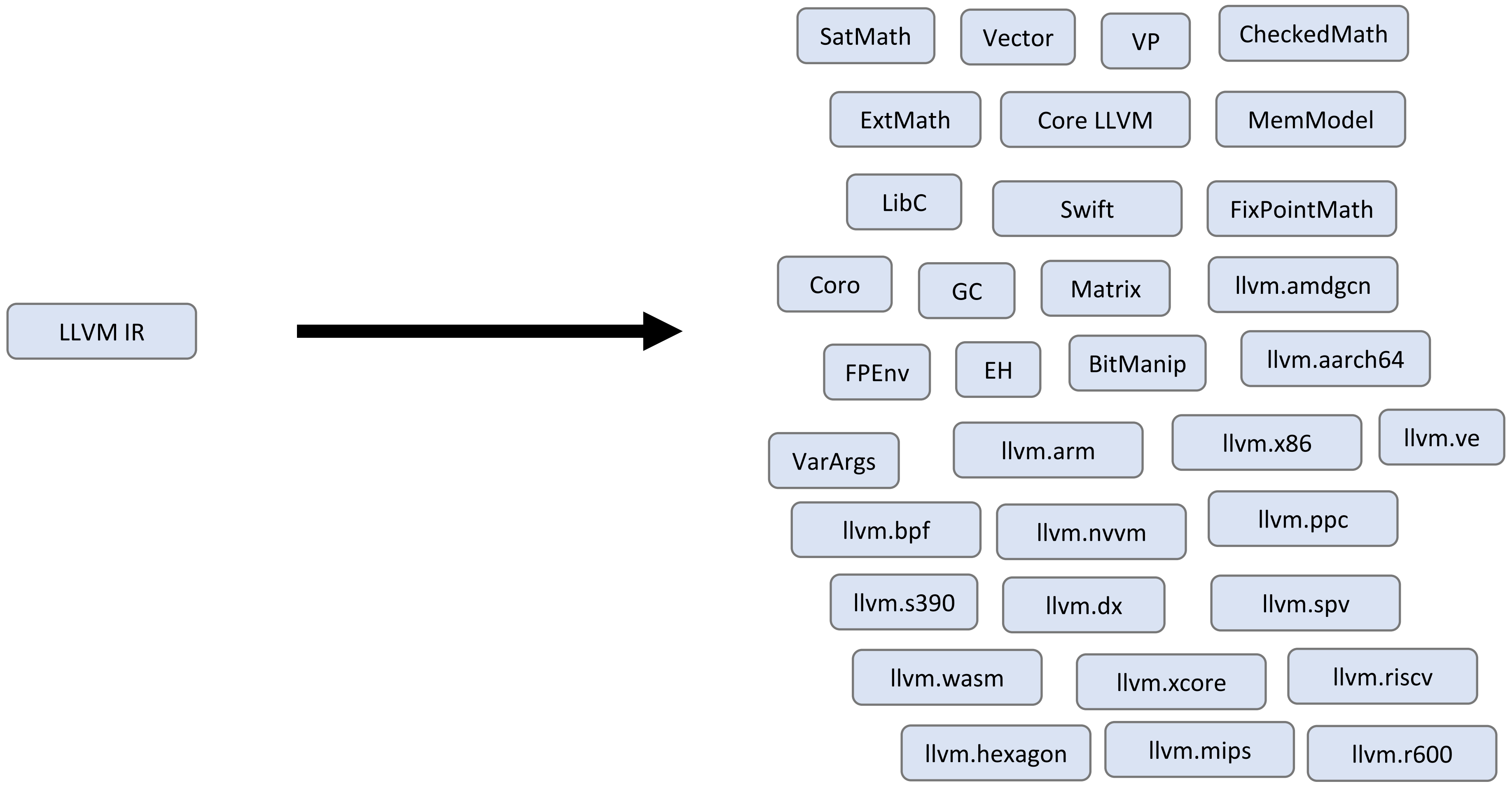
- Intrinsics are compile-time and -space inefficient

```
store volatile i32 %x, ptr %p
```

vs.

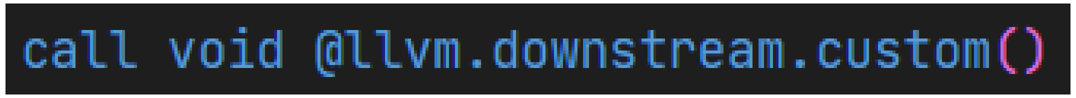
```
call void @llvm.memset.p0.i64(ptr %p, i8 %y, i64 4, i1 true)
```

Cleanup of “the LLVM dialect”

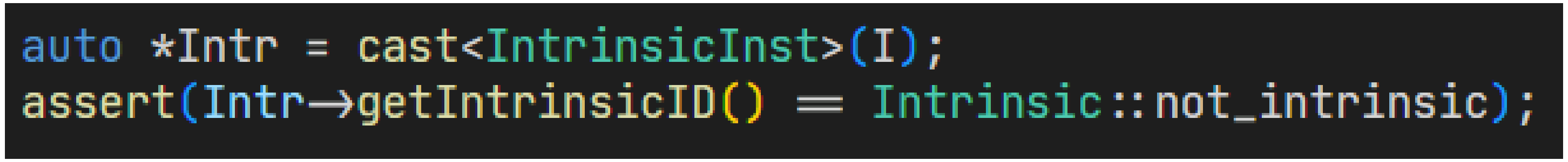


Extensibility

Today:

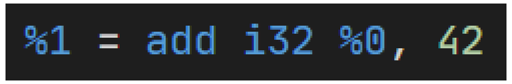


```
call void @llvm.downstream.custom()
```



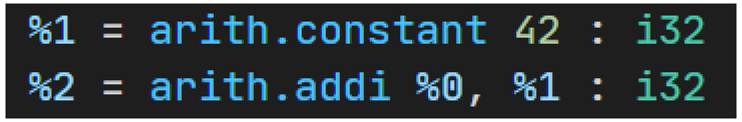
```
auto *Intr = cast<IntrinsicInst>(I);
assert(Intr->getIntrinsicID() == Intrinsic::not_intrinsic);
```

Example 2: Constant values vs. materialization



```
%1 = add i32 %0, 42
```

vs.



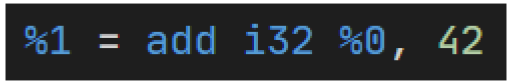
```
%1 = arith.constant 42 : i32
%2 = arith.addi %0, %1 : i32
```

- Value → User → Constant
  - ConstantData
    - ConstantInt, ConstantFP, …
    - ConstantDataSequential
  - ConstantAggregate
    - ConstantStruct, ConstantArray, ConstantVector
  - ConstantExpr
  - GlobalValue
    - GlobalIndirectSymbol
    - GlobalObject
      - Function
      - GlobalVariable
  - BlockAddress
  - DSOLocalEquivalent
  - UndefValue
  - PoisonValue

vs.

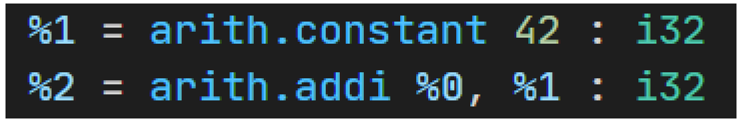
(N/A)

Deciding on a Unified Design



```
%1 = add i32 %0, 42
```

vs.



```
%1 = arith.constant 42 : i32
%2 = arith.addi %0, %1 : i32
```

- Constant equality via object identity

- Easier extensibility
- Debug info on constants
- Multi-threading

What is the right design in the long term?
Where is the (near-term) value?

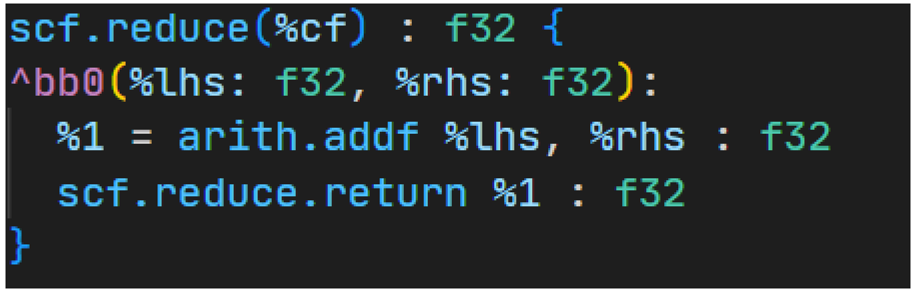
Eliminating llvm::Constant

- Value → User → Constant
  - ConstantData
    - ConstantInt, ConstantFP, …
    - ConstantDataSequential
  - ConstantAggregate
    - ConstantStruct, ConstantArray, ConstantVector
  - ConstantExpr
  - GlobalValue
    - GlobalIndirectSymbol
    - GlobalObject
      - Function
      - GlobalVariable
  - BlockAddress
  - DSOLocalEquivalent
  - UndefValue
  - PoisonValue

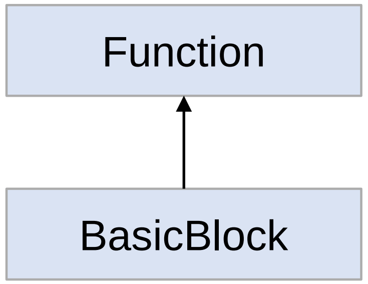
1. Remove from GlobalVariable initializers
   - Standalone representation of constant data
2. Remove from metadata
3. Introduce symbol reference instruction(s), detach GlobalValue from the Value hierarchy
4. Introduce constant instruction(s) for plain data, remove Constant

Example 3: Regions

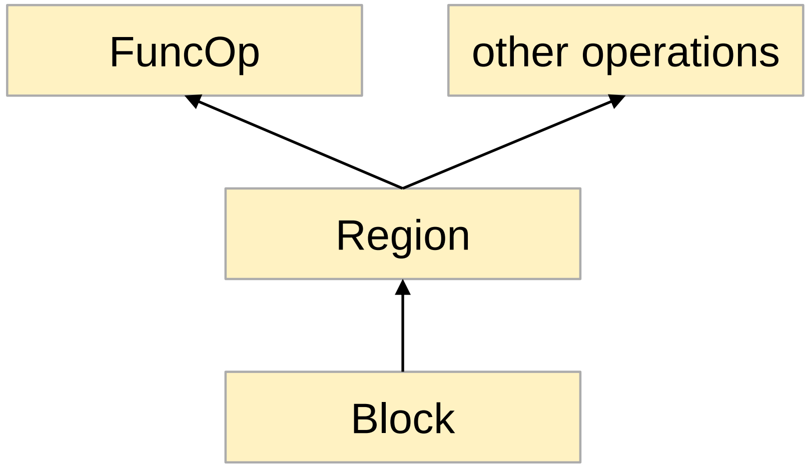
(N/A) vs.



```
scf.reduce(%cf) : f32 {
^bb0(%lhs: f32, %rhs: f32):
  %1 = arith.addf %lhs, %rhs : f32
  scf.reduce.return %1 : f32
}
```



vs.

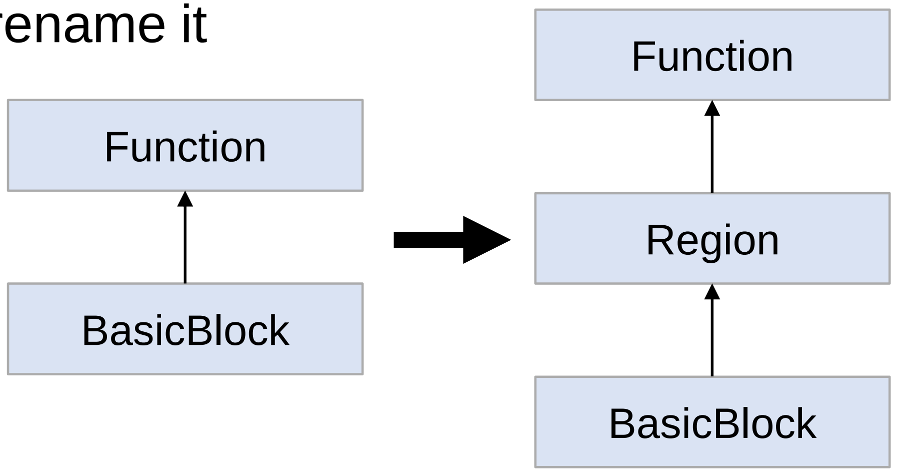


The near-term value of regions in LLVM

- Wave-wide mode in AMDGPU: side-stepping regular control flow
  - Analogous to “unmasked” in ISPC
- Waterfall loops in AMDGPU: highly regular loop with unusual and complex loop condition that does not benefit from generic loop transforms
- Frontends for C-like languages: scopes and lambdas

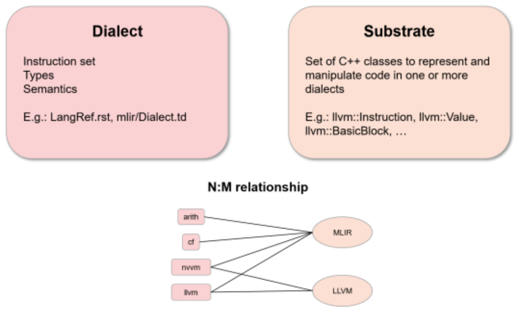
Regions for the LLVM substrate

1. llvm::Region exists but means something quite different; rename it
2. Insert Region into the Function/BasicBlock relationship
   - Change BB->getParent() to return Region instead of Function; replace by BB->getFunction() where necessary and appropriate
   - Iterate over F->getBody() instead of *F
3. Define a concept of region passes
4. Audit existing function passes: which ones should really be region passes?
5. Add regions to instructions



Recap

- Dialects and substrates
- Benefits of a unified substrate
- Thought experiments on unification of LLVM and MLIR
- The delta matrix
- Feasibility and near-term value of some column refactorings



Dialect

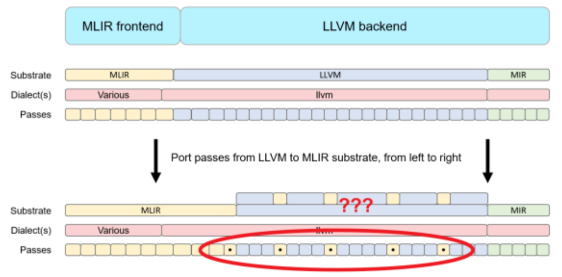
Instruction set
Types
Semantics

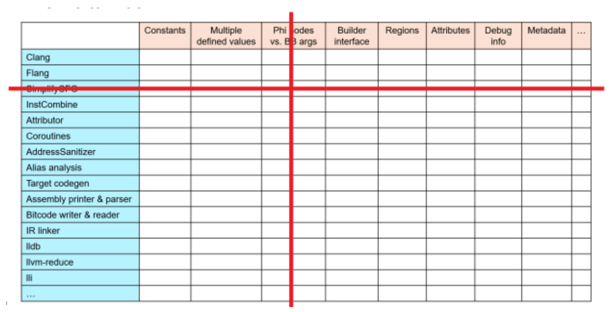
E.g.: LangRef.rst, mlir/Dialect.td

Substrate

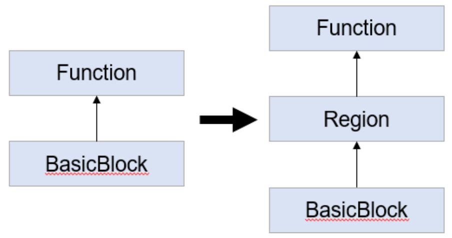
Set of C++ classes to represent and manipulate code in one or more dialects

E.g.: llvm::Instruction, llvm::Value, llvm::BasicBlock, …





	Constants	Multiple defined values	Phi nodes vs. BB args	Builder interface	Regions	Attributes	Debug info	Metadata	…
Clang									
Flang									
SimplifyCFG									
InstCombine									
Attributor									
Coroutines									
AddressSanitizer									
Alias analysis									
Target codegen									
Assembly printer & parser									
Bitcode writer & reader									
IR linker									
lldb									
llvm-reduce									
lli									
…									



The End

"Light at the End of the Tunnel" by DALL-E 2

COPYRIGHT AND DISCLAIMER

Dialect implementation that is generic over the substrate

- Using a generic builder to create a generic instruction.
- Behind the scenes, the template parameter is resolved to the substrate-specific instruction class

```
builder.template create<generic::AddInst>(lhs, rhs);
```

- Using dyn_cast<T> with a generic instruction.
- The result type is a function of both the generic target instruction and the substrate-specific operand type, so the result type can be substrate-specific as well.

```
for (auto &inst : basicBlock) {
    if (auto *alloca = dyn_cast<generic::AllocaInst>(&inst)) {
      ...
    }
}
```