VPlan: Status Update and Future Roadmap

Florian Hahn, Compilers @ Apple
Ayal Zaks, Compilers @ Mobileye

2023 US LLVM Developers' Meeting | Oct. 11, 2023

Context: Vectorization Plan in the Loop Vectorizer

Ongoing incremental effort to upgrade the Loop Vectorizer's infrastructure and extend its capabilities

Extending LoopVectorizer towards supporting OpenMP4.5 SIMD and outer loop auto-vectorization
2016 US LLVM Developers' Meeting, *H. Saito*

Introducing VPlan to the Loop Vectorizer
2017 Euro LLVM Developers' Meeting, *G. Rapaport*

Vectorizing Loops with VPlan – Current State and Next Steps
2017 US LLVM Developers' Meeting, *A. Zaks*

Extending LoopVectorize to Support Outer Loop Vectorization Using VPlan
2018 Euro LLVM Developers' Meeting, *D. Caballero, S. Guggilla*

Documentation: https://www.llvm.org/docs/VectorizationPlan.html

Context: Vectorization Plan in the Loop Vectorizer

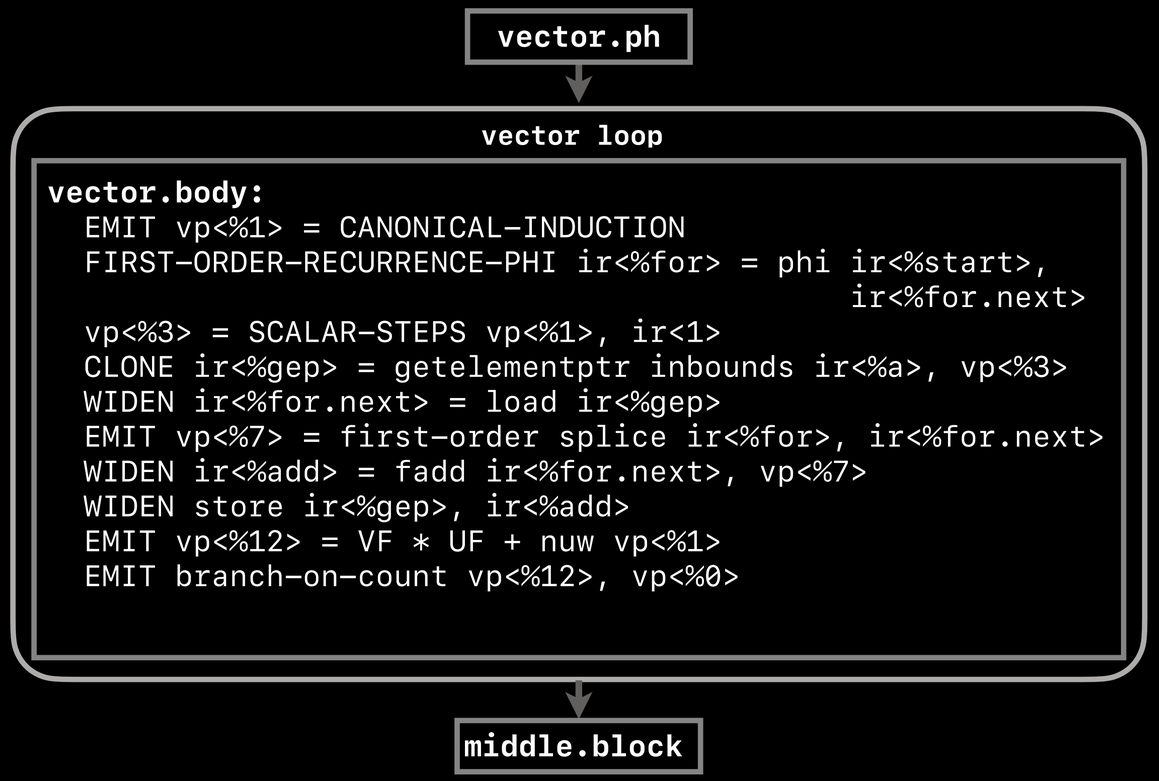
VPlan is an explicit model for describing vectorization candidates

Context: Vectorization Plan in the Loop Vectorizer

Input LLVM-IR

```
entry:
  br label %loop

loop:
  %for = phi float [ %start, %entry ], [ %for.next, %loop ]
  %iv = phi i64 [ 0, %entry ], [ %iv.next, %loop ]
  %iv.next = add nuw nsw i64 %iv, 1
  %gep = getelementptr inbounds float, ptr %a, i64 %iv
  %for.next  = load float, ptr %gep
  %add = fadd float %for.next, %for
  store float %add, ptr %gep
  %ec = icmp eq i64 %iv.next, %n
  br i1 %ec, label %exit, label %loop

exit:
  ret void
```

VPlan for VF={4},UF>=1



Context: Vectorization Plan in the Loop Vectorizer

VPRegionBlock

Input LLVM-IR

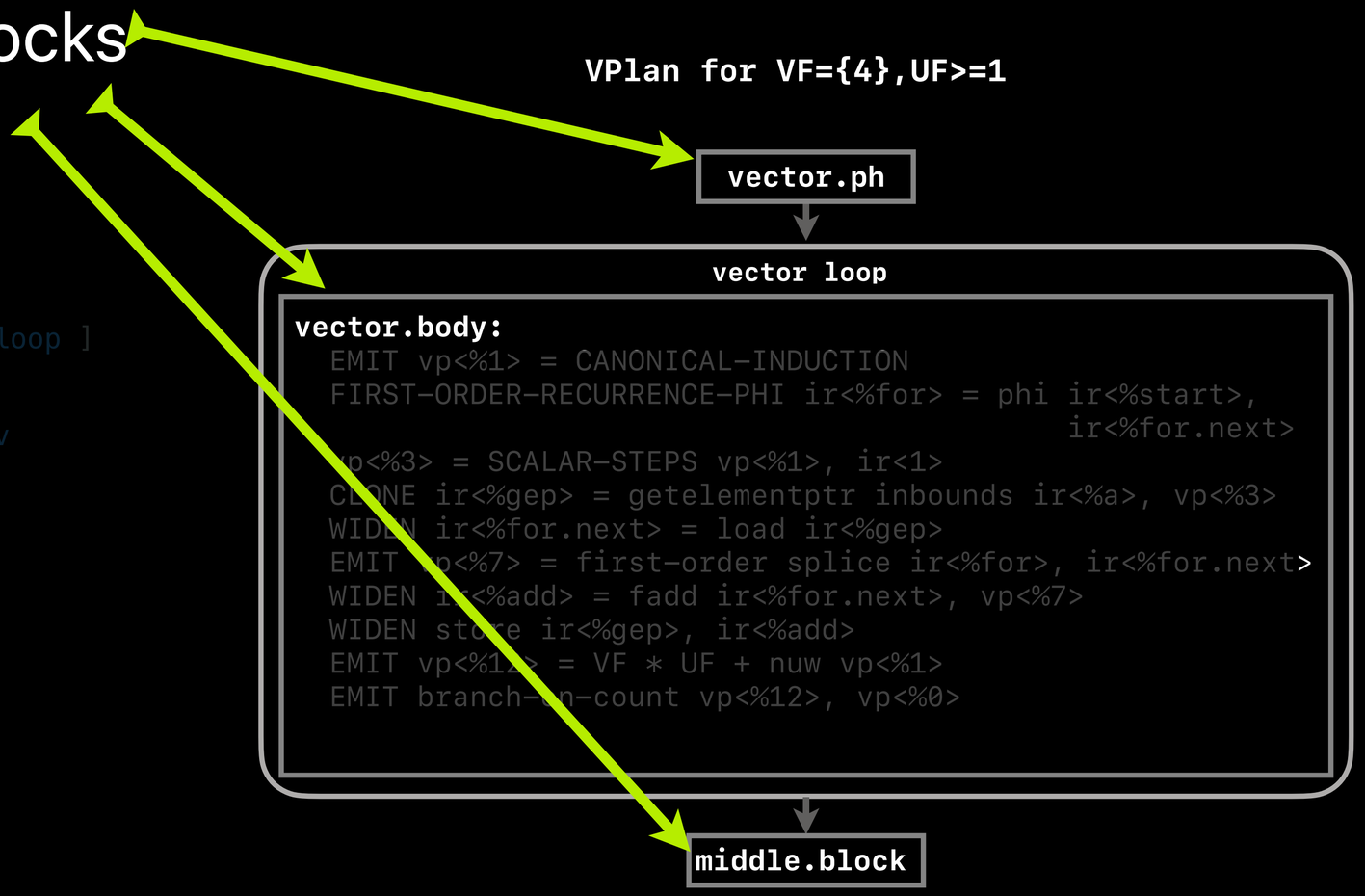
```
entry:
  br label %loop

loop:
  %for = phi float [ %start, %entry ], [ %for.next, %loop ]
  %iv = phi i64 [ 0, %entry ], [ %iv.next, %loop ]
  %iv.next = add nuw nsw i64 %iv, 1
  %gep = getelementptr inbounds float, ptr %a, i64 %iv
  %for.next  = load float, ptr %gep
  %add = fadd float %for.next, %for
  store float %add, ptr %gep
  %ec = icmp eq i64 %iv.next, %n
  br i1 %ec, label %exit, label %loop

exit:
  ret void
```

VPlan for VF={4},UF>=1

Context: Vectorization Plan in the Loop Vectorizer

Input LLVM-IR

```
entry:
  br label %loop

loop:
  %for = phi float [ %start, %entry ], [ %for.next, %loop ]
  %iv = phi i64 [ 0, %entry ], [ %iv.next, %loop ]
  %iv.next = add nuw nsw i64 %iv, 1
  %gep = getelementptr inbounds float, ptr %a, i64 %iv
  %for.next  = load float, ptr %gep
  %add = fadd float %for.next, %for
  store float %add, ptr %gep
  %ec = icmp eq i64 %iv.next, %n
  br i1 %ec, label %exit, label %loop

exit:
  ret void
```



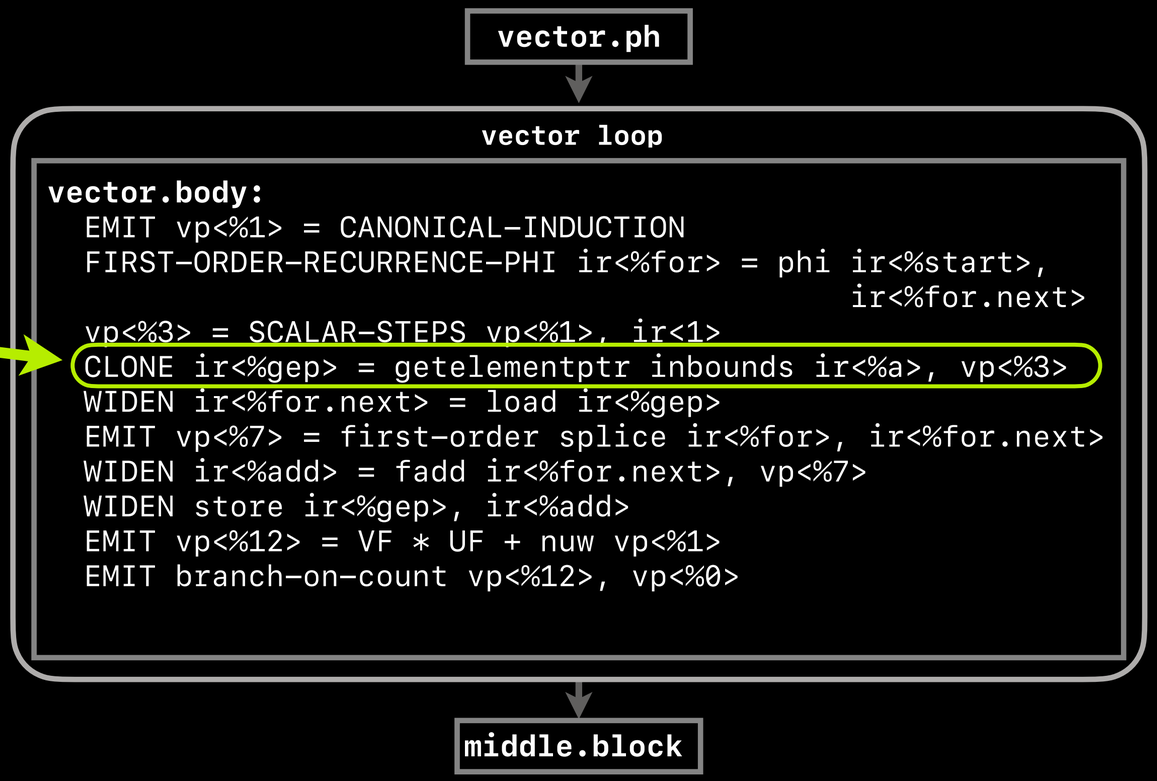
Context: Vectorization Plan in the Loop Vectorizer

Input LLVM-IR

```
entry:
  br label %loop

loop:
  %for = phi float [ %start, %entry ], [ %for.next, %loop ]
  %iv = phi i64 [ 0, %entry ], [ %iv.next, %loop ]
  %iv.next = add nuw nsw i64 %iv, 1
  %gep = getelementptr inbounds float, ptr %a, i64 %iv
  %for.next = load float, ptr %gep
  %add = fadd float %for.next, %for
  store float %add, ptr %gep
  %ec = icmp eq i64 %iv.next, %n
  br i1 %ec, label %exit, label %loop

exit:
  ret void
```

VPRecipes

VPlan for VF={4},UF>=1



Context: Vectorization Plan in the Loop Vectorizer

Input LLVM-IR

```
entry:
  br label %loop

loop:
  %for = phi float [ %start, %entry ], [ %for.next, %loop ]
  %iv = phi i64 [ 0, %entry ], [ %iv.next, %loop ]
  %iv.next = add nuw nsw i64 %iv, 1
  %gep = getelementptr inbounds float, ptr %a, i64 %iv
  %for.next  = load float, ptr %gep
  %add = fadd float %for.next, %for
  store float %add, ptr %gep
  %ec = icmp eq i64 %iv.next, %n
  br i1 %ec, label %exit, label %loop

exit:
  ret void
```

VPRecipes

VPlan for VF={4},UF>=1

Context: Vectorization Plan in the Loop Vectorizer

Vectorization Plan is an explicit model for describing vectorization candidates

Input LLVM-IR

```
entry:
  br label %loop

loop:
  %for = phi float [ %start, %entry ], [ %for.next, %loop ]
  %iv = phi i64 [ 0, %entry ], [ %iv.next, %loop ]
  %iv.next = add nuw nsw i64 %iv, 1
  %gep = getelementptr inbounds float, ptr %a, i64 %iv
  %for.next  = load float, ptr %gep
  %add = fadd float %for.next, %for
  store float %add, ptr %gep
  %ec = icmp eq i64 %iv.next, %n
  br i1 %ec, label %exit, label %loop

exit:
  ret void
```

Context: Vectorization Plan in the Loop Vectorizer

Vectorization Plan is an explicit model for describing vectorization candidates

Input LLVM-IR

```
entry:
  br label %loop

loop:
  %for = phi float [ %start, %entry ], [ %for.next, %loop ]
  %iv = phi i64 [ 0, %entry ], [ %iv.next, %loop ]
  %iv.next = add nuw nsw i64 %iv, 1
  %gep = getelementptr inbounds float, ptr %a, i64 %iv
  %for.next  = load float, ptr %gep
  %add = fadd float %for.next, %for
  store float %add, ptr %gep
  %ec = icmp eq i64 %iv.next, %n
  br i1 %ec, label %exit, label %loop

exit:
  ret void
```

Output LLVM-IR

```
vector.ph:
  %n.mod.vf = urem i64 %n, 4
  %n.vec = sub i64 %n, %n.mod.vf
  %vector.recur.init = insertelement <4 x float> …
  br label %vector.body

vector.body:
  %index = phi i64 [ 0, %vector.ph ], [ %index.next, …
  %vector.recur = phi <4 x float> [ %vector.recur….
  %0 = add i64 %index, 0
  %1 = getelementptr inbounds float, ptr %a, i64 %0
  %2 = getelementptr inbounds float, ptr %1, i32 0
  %wide.load = load <4 x float>, ptr %2, align 4
  %3 = shufflevector <4 x float> %vector.recur, …
  %4 = fadd <4 x float> %wide.load, %3
  store <4 x float> %4, ptr %2, align 4
  %index.next = add nuw i64 %index, 4
  %5 = icmp eq i64 %index.next, %n.vec
  br i1 %5, label %middle.block, label %vector.body,…

middle.block:
  %cmp.n = icmp eq i64 %n, %n.vec
  %vector.recur.extract = extractelement <4 x float>…
  br i1 %cmp.n, label %exit, label %scalar.ph
```

Status Update

VPlan in 2018

VPlan for VF={4},UF>=1

```
vector.body:
  WIDEN-PHI %for = phi %start, %for.next
  WIDEN-INDUCTION %iv = phi 0, %iv.next
  CLONE %gep = getelementptr %a, %iv
  WIDEN %for.next = load %arrayidx32, ir<%gep>
  CLONE %arrayidx34 = getelementptr %b, %iv
  WIDEN
    %add = fadd %for.next, %for
  WIDEN store %add, %gep, ir<%gep>
```

VPlan in 2018

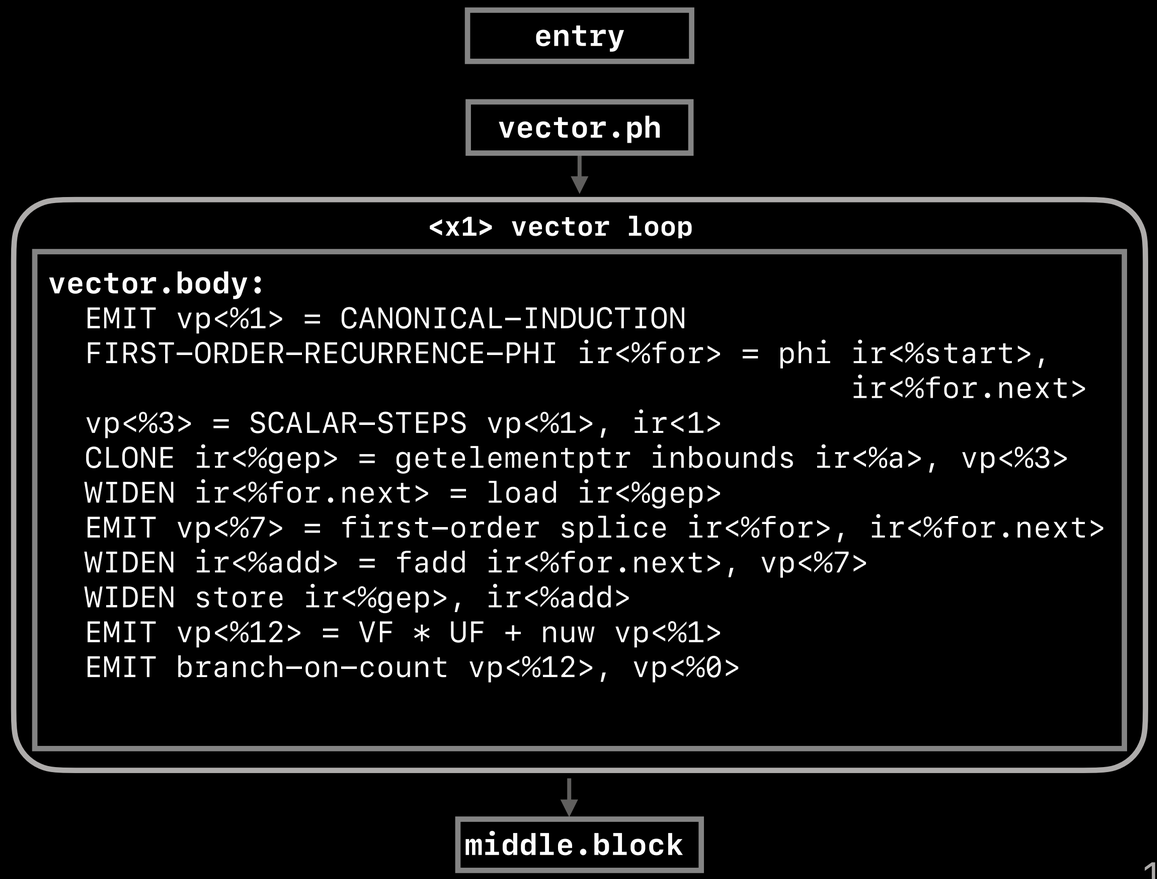
VPlan for VF={4},UF>=1

```
vector.body:
  WIDEN-PHI %for = phi %start, %for.next
  WIDEN-INDUCTION %iv = phi 0, %iv.next
  CLONE %gep = getelementptr %a, %iv
  WIDEN %for.next = load %arrayidx32, ir<%gep>
  CLONE %arrayidx34 = getelementptr %b, %iv
  WIDEN
    %add = fadd %for.next, %for
  WIDEN store %add, %gep, ir<%gep>
```

VPlan in 2023

VPlan for VF={4},UF>=1

```
Live-in vp<%0> = vector-trip-count
```



VPlan in 2018

VPlan in 2023

More gradual & precise Recipes

VPlan for VF={4},UF>=1

```
vector.body:
  WIDEN-PHI %for = phi %start, %for.next
  WIDEN-INDUCTION %iv = phi 0, %iv.next
  CLONE %gep = getelementptr %a, %iv
  WIDEN %for.next = load %arrayidx32, ir<%gep>
  CLONE %arrayidx34 = getelementptr %b, %iv
  WIDEN
    %add = fadd %for.next, %for
  WIDEN store %add, %gep, ir<%gep>
```

VPlan for VF={4},UF>=1

```
Live-in vp<%0> = vector-trip-count
```

entry

vector.ph

<x1> vector loop

```
vector.body:
  EMIT vp<%1> = CANONICAL-INDUCTION
  FIRST-ORDER-RECURRENCE-PHI ir<%for> = phi ir<%start>,
                                            ir<%for.next>
  vp<%3> = SCALAR-STEPS vp<%1>, ir<1>
  CLONE ir<%gep> = getelementptr inbounds ir<%a>, vp<%3>
  WIDEN ir<%for.next> = load ir<%gep>
  EMIT vp<%7> = first-order splice ir<%for>, ir<%for.next>
  WIDEN ir<%add> = fadd ir<%for.next>, vp<%7>
  WIDEN store ir<%gep>, ir<%add>
  EMIT vp<%12> = VF * UF + nuw vp<%1>
  EMIT branch-on-count vp<%12>, vp<%0>
```

middle.block

VPlan in 2018

VPlan in 2023

Def-Use chains modeled in directly in VPlan

VPlan for VF={4},UF>=1

```
vector.body:
  WIDEN-PHI %for = phi %start, %for.next
  WIDEN-INDUCTION %iv = phi 0, %iv.next
  CLONE %gep = getelementptr %a, %iv
  WIDEN %for.next = load %arrayidx32, ir<%gep>
  CLONE %arrayidx34 = getelementptr %b, %iv
  WIDEN
    %add = fadd %for.next, %for
  WIDEN store %add, %gep, ir<%gep>
```

VPlan for VF={4},UF>=1

```
Live-in vp<%0> = vector-trip-count
```

entry

vector.ph

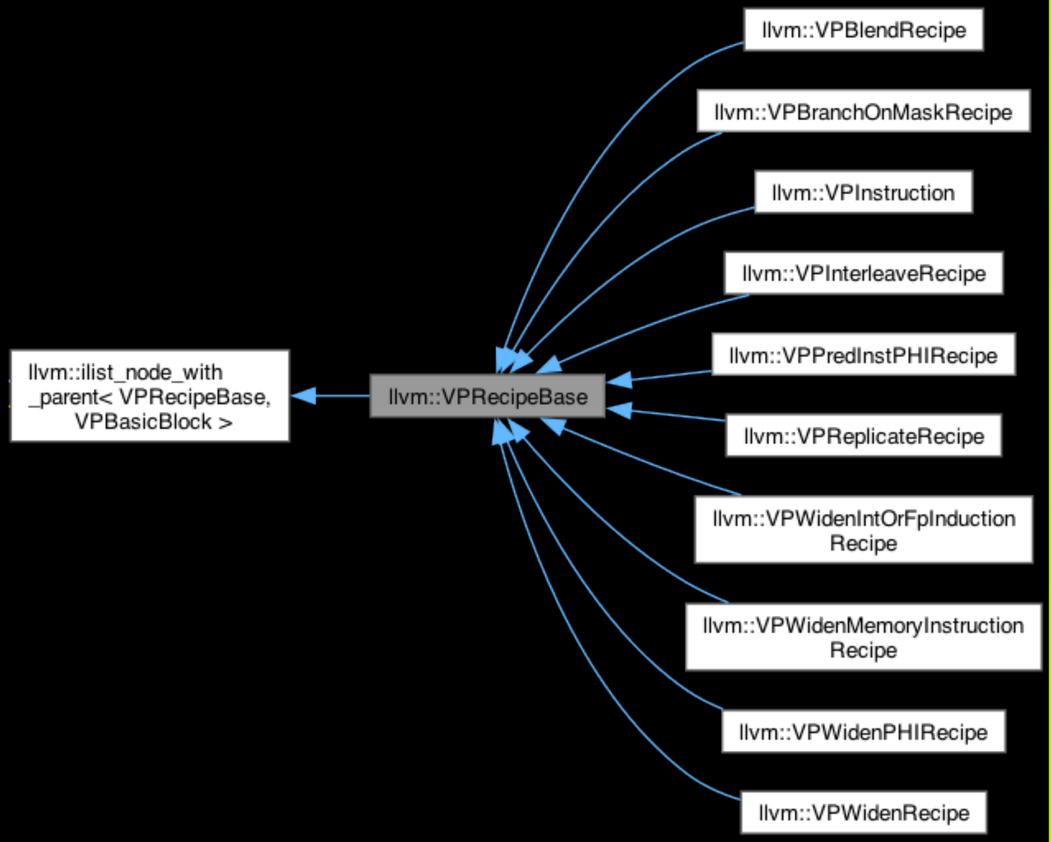
<x1> vector loop

```
vector.body:
  EMIT vp<%1> = CANONICAL-INDUCTION
  FIRST-ORDER-RECURRENCE-PHI ir<%for> = phi ir<%start>,
                                            ir<%for.next>
  vp<%3> = SCALAR-STEPS vp<%1>, ir<1>
  CLONE ir<%gep> = getelementptr inbounds ir<%a>, vp<%3>
  WIDEN ir<%for.next> = load ir<%gep>
  EMIT vp<%7> = first-order splice ir<%for>, ir<%for.next>
  WIDEN ir<%add> = fadd ir<%for.next>, vp<%7>
  WIDEN store ir<%gep>, ir<%add>
  EMIT vp<%12> = VF * UF + nuw vp<%1>
  EMIT branch-on-count vp<%12>, vp<%0>
```

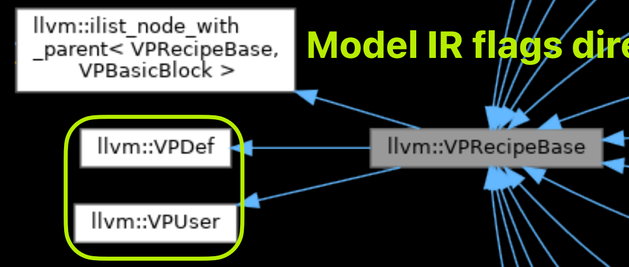
middle.block

and Many More Improvements

Recipes in 2018



Recipes in 2023

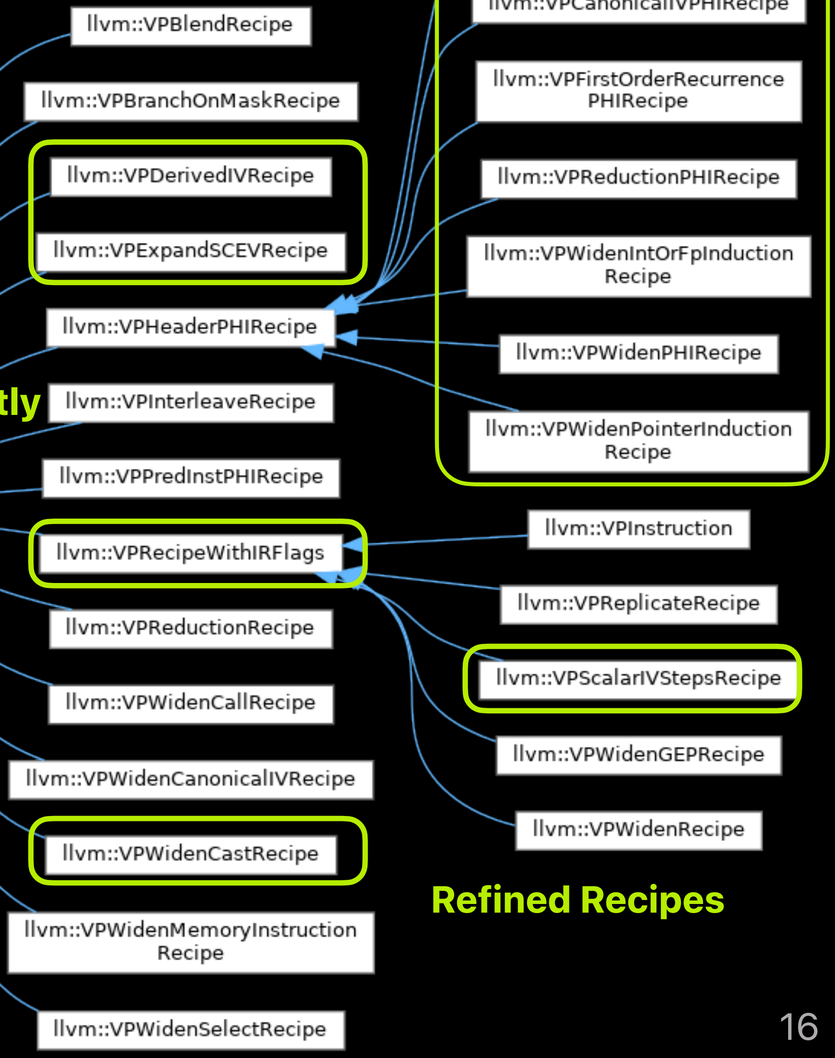


Model IR flags directly

Each recipe can define 0, 1 or more values via VPDef

Operands are managed via VPUser

Clarified header PHI Recipes



Refined Recipes

discourse.llvm.org/t/vplan-progress-update-august-2023/73003

VPlan in 2018

VPlan in 2023

VPlan scope was extended

VPlan for VF={4},UF>=1

```
vector.body:
  WIDEN-PHI %for = phi %start, %for.next
  WIDEN-INDUCTION %iv = phi 0, %iv.next
  CLONE %gep = getelementptr %a, %iv
  WIDEN %for.next = load %arrayidx32, ir<%gep>
  CLONE %arrayidx34 = getelementptr %b, %iv
  WIDEN
    %add = fadd %for.next, %for
  WIDEN store %add, %gep, ir<%gep>
```

VPlan for VF={4},UF>=1

```
Live-in vp<%0> = vector-trip-count
```

entry

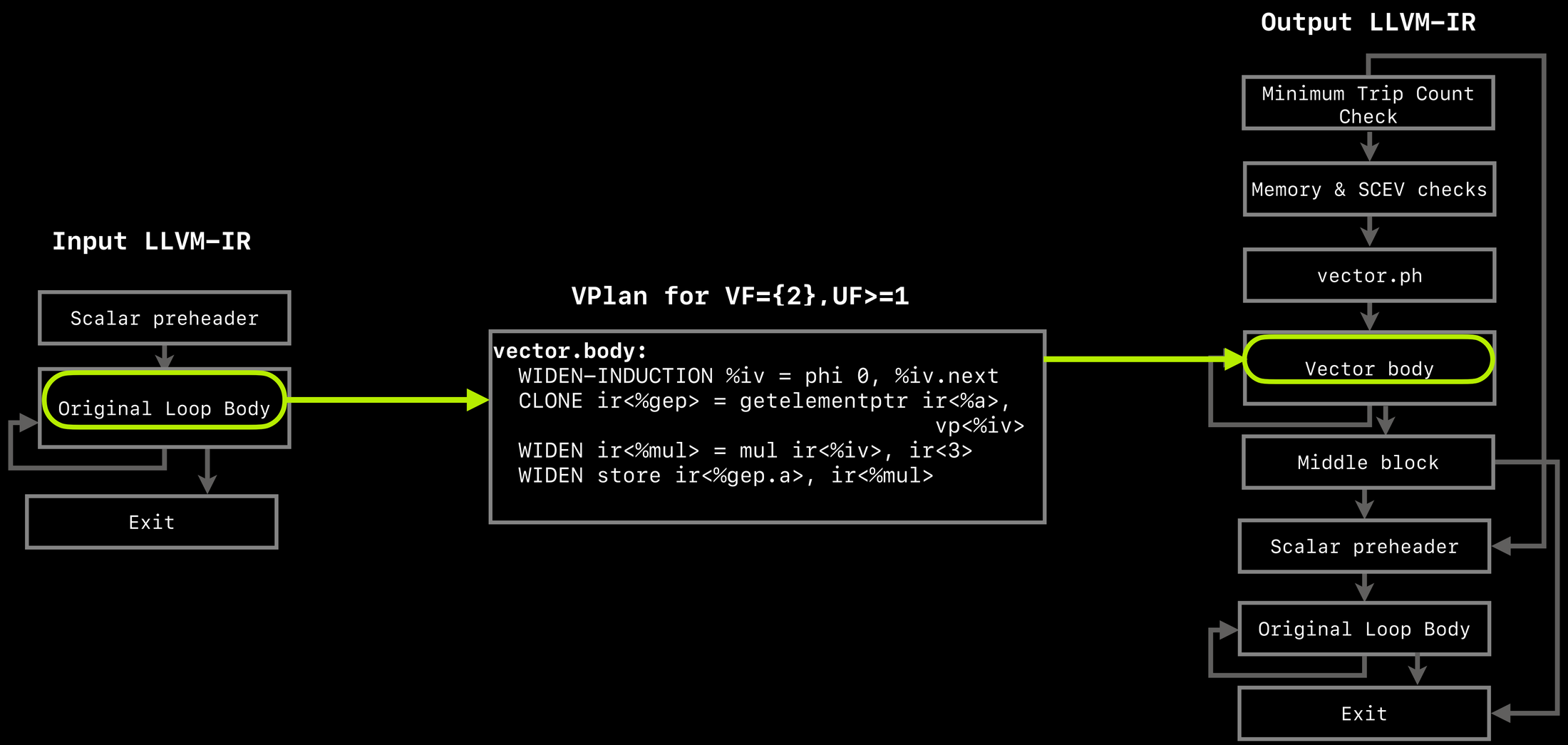
vector.ph

<x1> vector loop

```
vector.body:
  EMIT vp<%1> = CANONICAL-INDUCTION
  FIRST-ORDER-RECURRENCE-PHI ir<%for> = phi ir<%start>,
                                            ir<%for.next>
  vp<%3> = SCALAR-STEPS vp<%1>, ir<1>
  CLONE ir<%gep> = getelementptr inbounds ir<%a>, vp<%3>
  WIDEN ir<%for.next> = load ir<%gep>
  EMIT vp<%7> = first-order splice ir<%for>, ir<%for.next>
  WIDEN ir<%add> = fadd ir<%for.next>, vp<%7>
  WIDEN store ir<%gep>, ir<%add>
  EMIT vp<%12> = VF * UF + nuw vp<%1>
  EMIT branch-on-count vp<%12>, vp<%0>
```

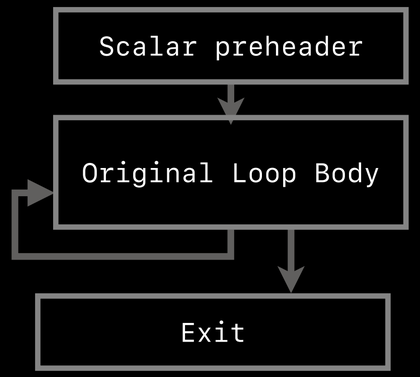
middle.block

VPlan Scope in 2018



VPlan Scope in 2018

Input LLVM-IR

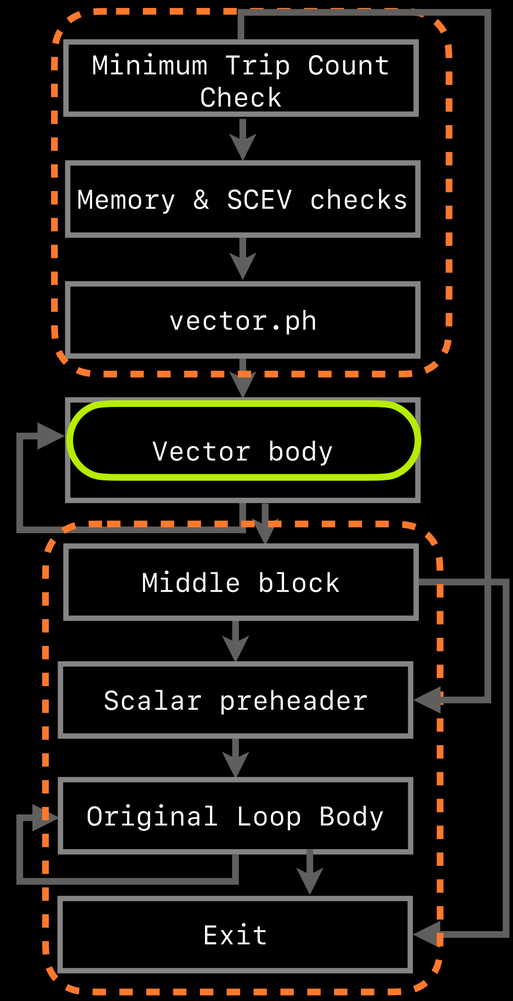


VPlan for VF={2},UF>=1

```
vector.body:
  WIDEN-INDUCTION %iv = phi 0, %iv.next
  CLONE ir<%gep> = getelementptr ir<%a>,
                                 vp<%iv>
  WIDEN ir<%mul> = mul ir<%iv>, ir<3>
  WIDEN store ir<%gep.a>, ir<%mul>
```

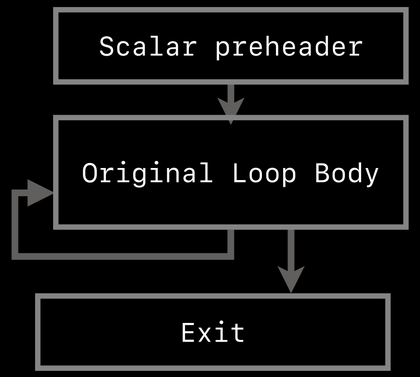
Yet to model in VPlan '18

Output LLVM-IR



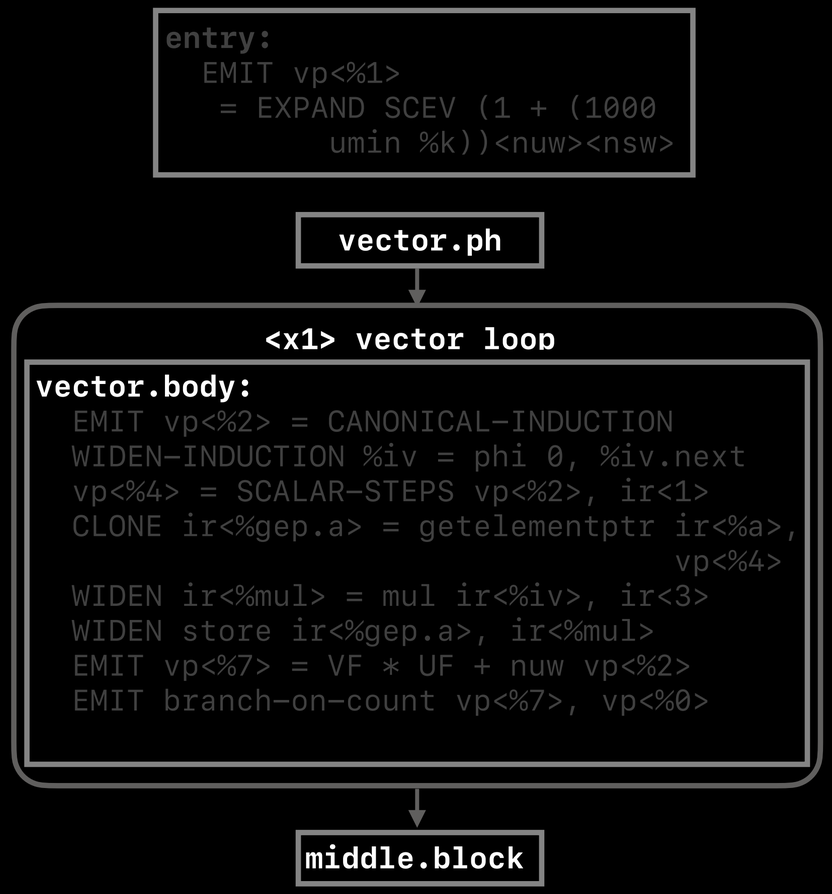
VPlan Scope in 2023

Input LLVM-IR



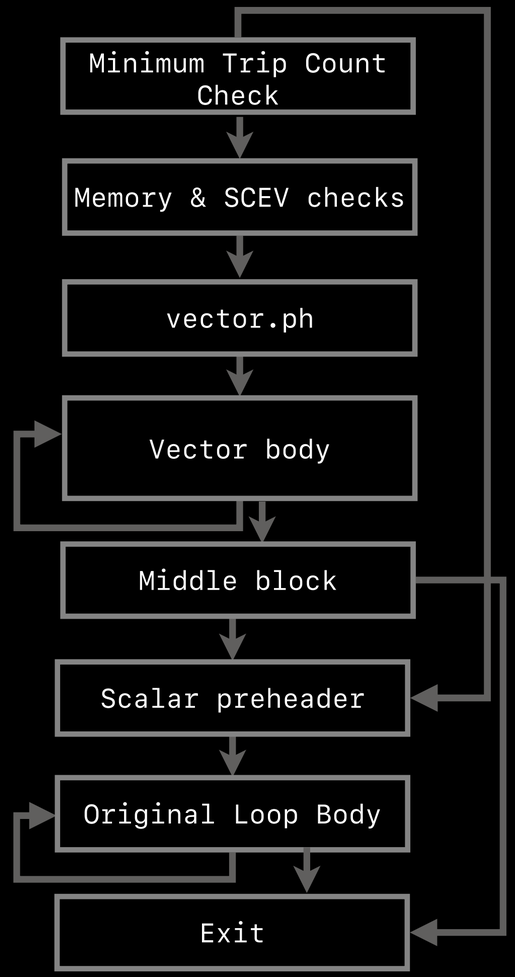
VPlan for VF={2},UF>=1

Live-in vp<%0> = vector-trip-count

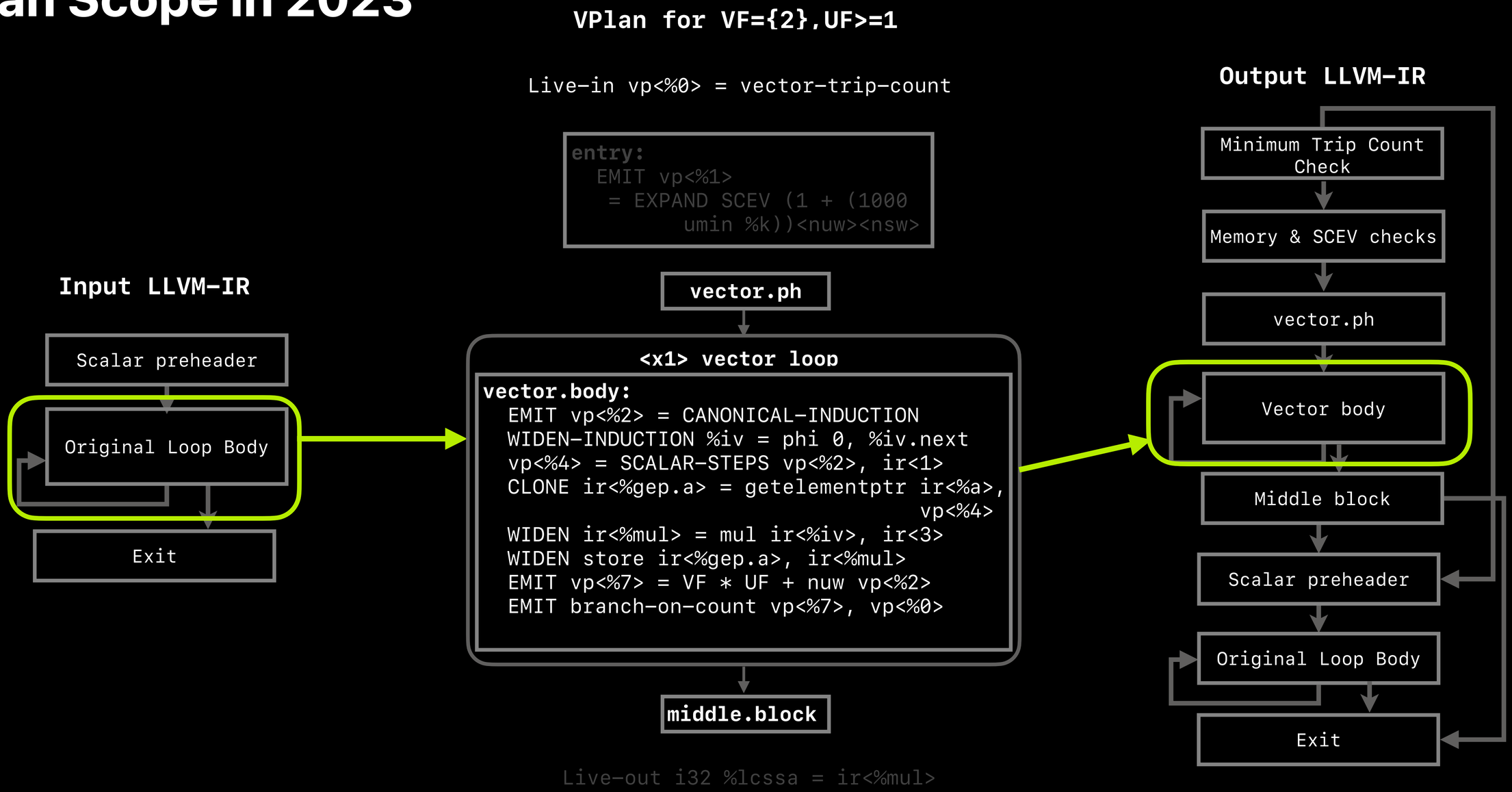


Live-out i32 %lcssa = ir<%mul>

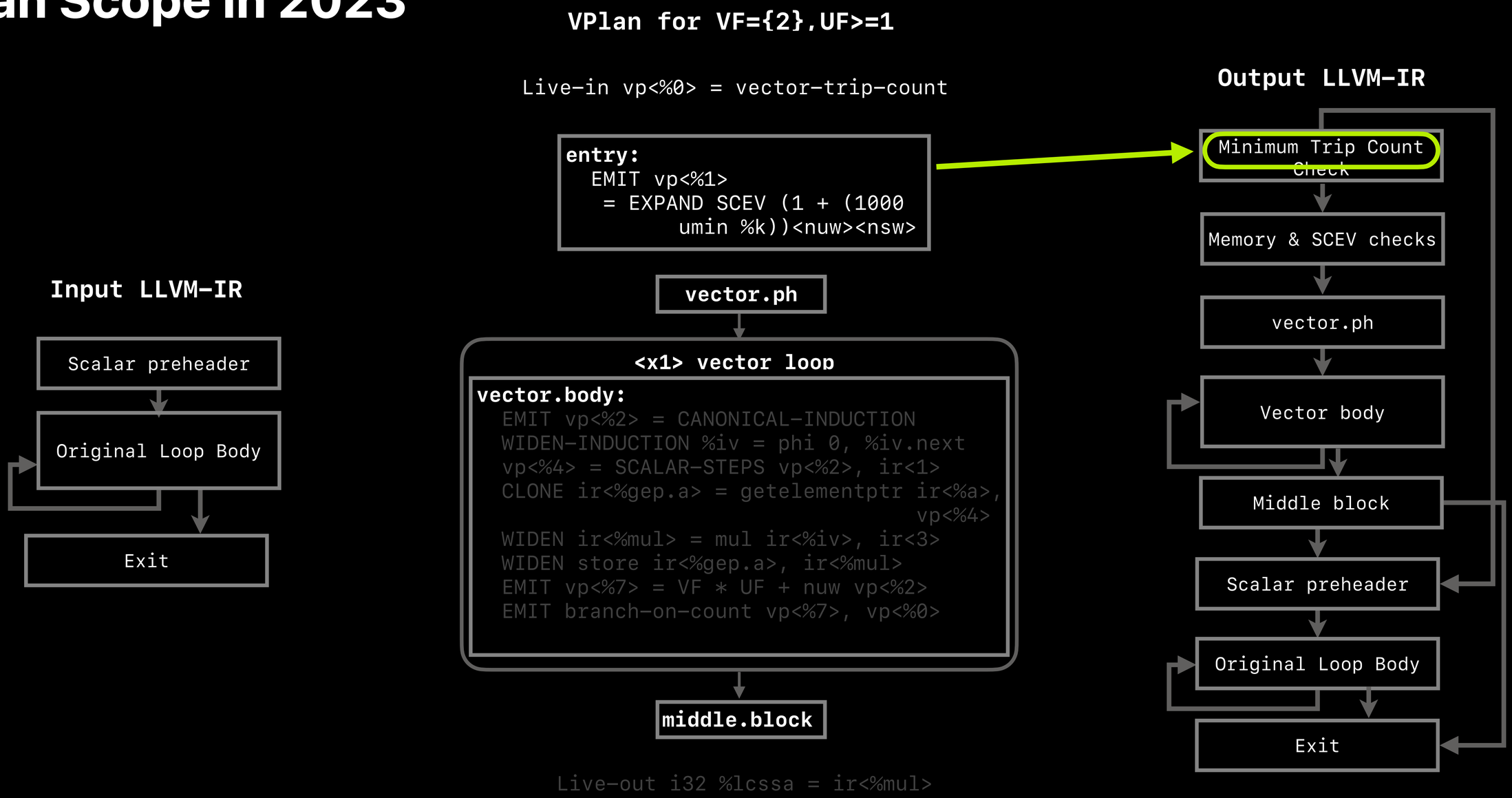
Output LLVM-IR



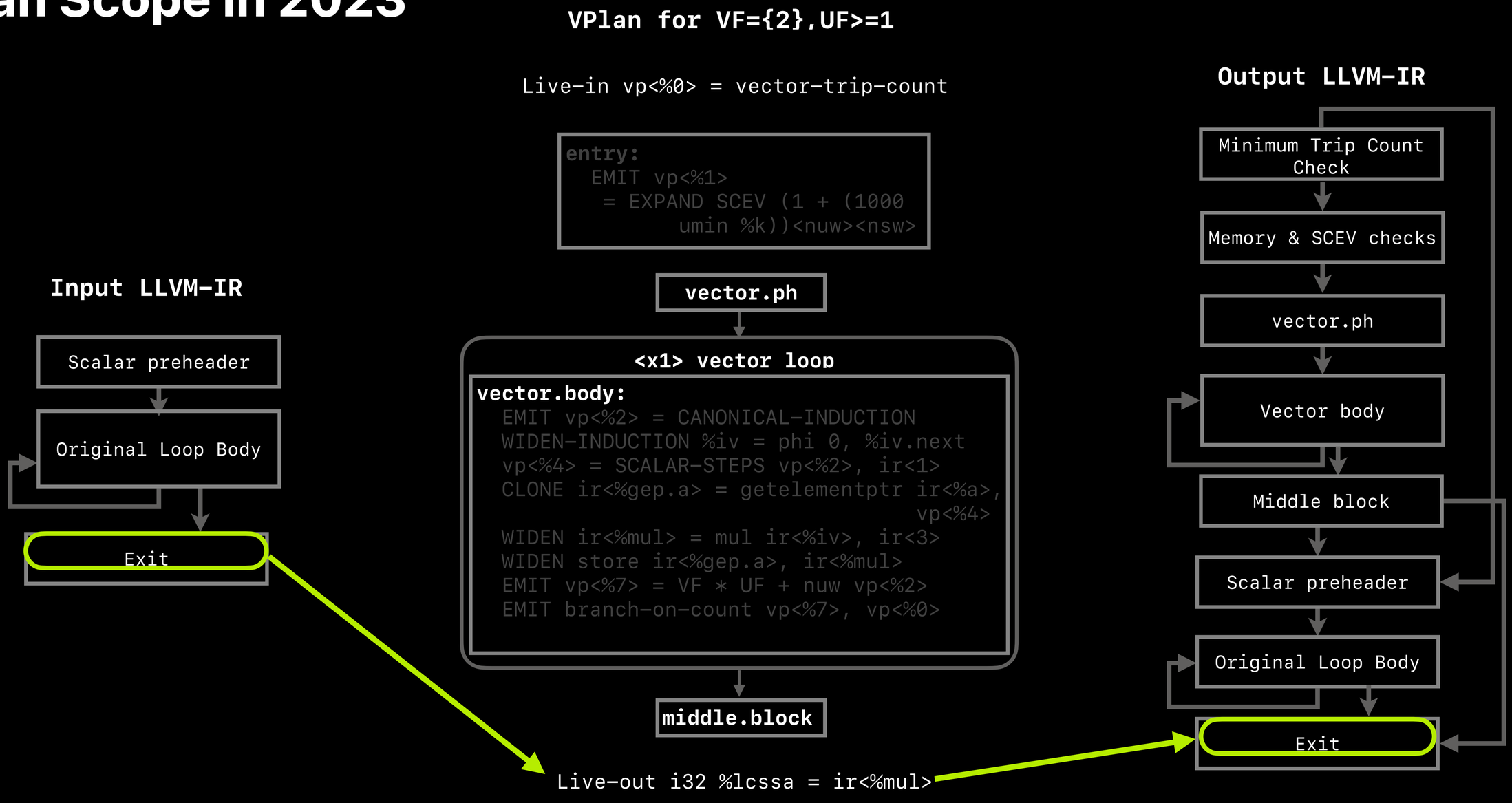
VPlan Scope in 2023



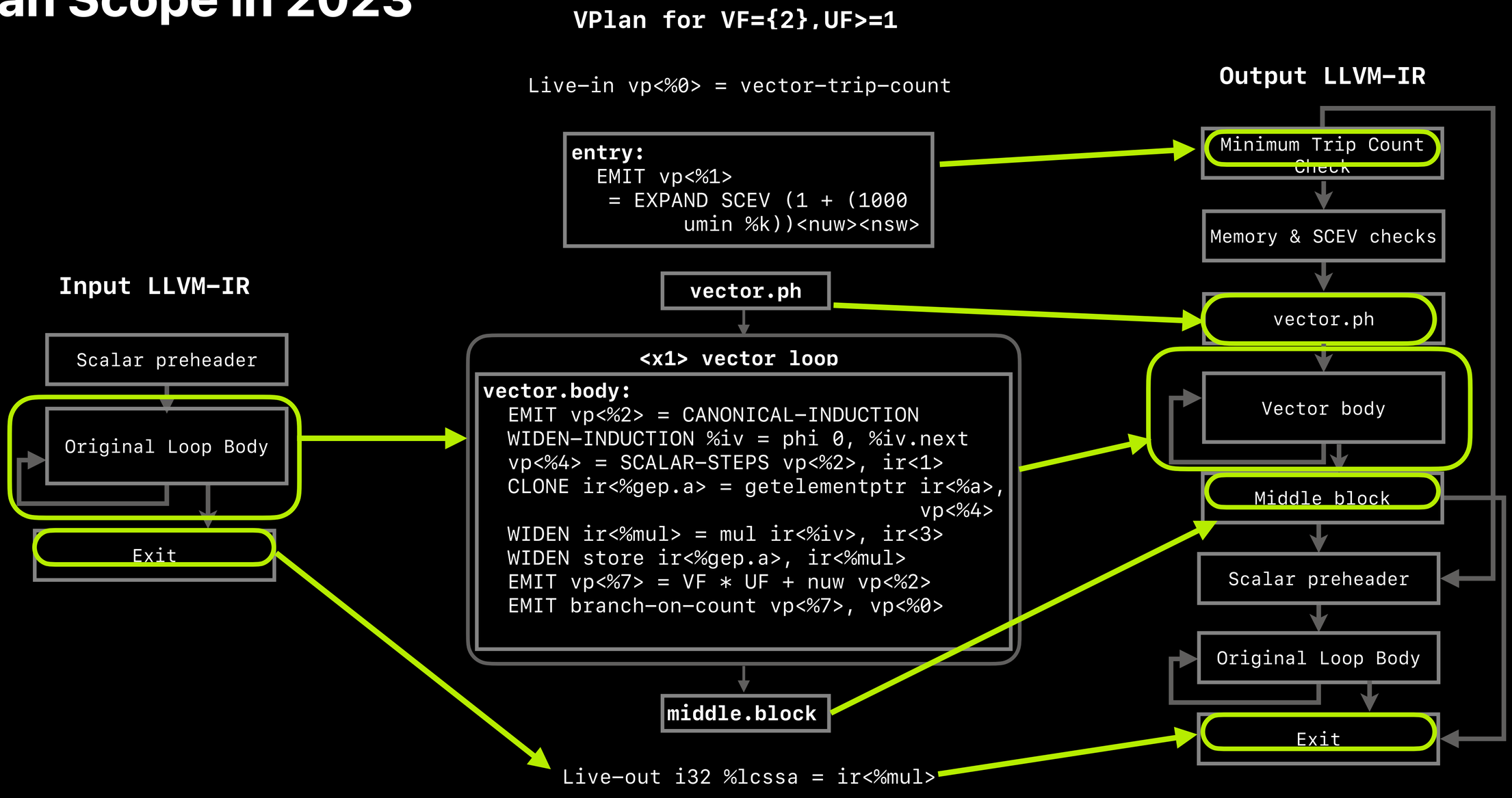
VPlan Scope in 2023



VPlan Scope in 2023

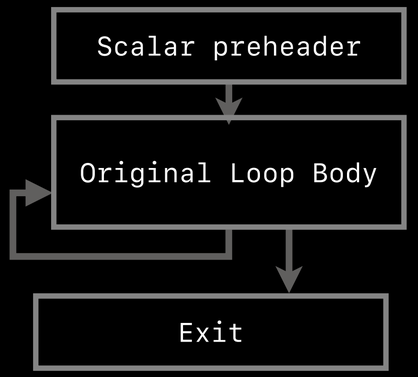


VPlan Scope in 2023



VPlan Scope in 2023

Input LLVM-IR

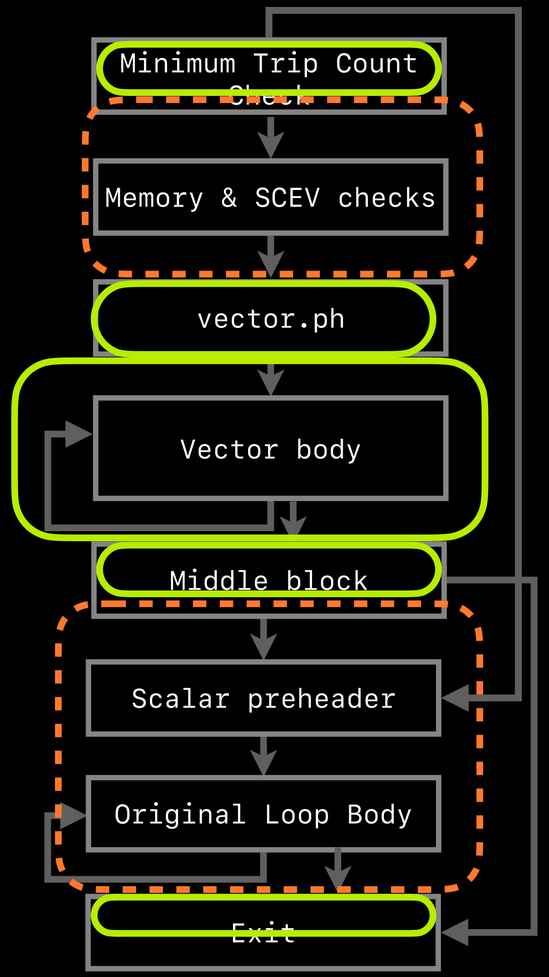


VPlan for VF={2},UF>=1

```
Live-in vp<%0> = vector-trip-count

entry:
  EMIT vp<%1>
   = EXPAND SCEV (1 + (1000
          umin %k))<nuw><nsw>

vector.ph

<x1> vector loop
vector.body:
  EMIT vp<%2> = CANONICAL-INDUCTION
  WIDEN-INDUCTION %iv = phi 0, %iv.next
  vp<%4> = SCALAR-STEPS vp<%2>, ir<1>
  CLONE ir<%gep.a> = getelementptr ir<%a>,
                                   vp<%4>
  WIDEN ir<%mul> = mul ir<%iv>, ir<3>
  WIDEN store ir<%gep.a>, ir<%mul>
  EMIT vp<%7> = VF * UF + nuw vp<%2>
  EMIT branch-on-count vp<%7>, vp<%0>

middle.block

Live-out i32 %lcssa = ir<%mul>
```

Yet to model in VPlan '23

Output LLVM-IR



Improved Modeling Resolved A Number Of Crashes

✅ #46687 opt -loop-vectorize crashes with "PHINode should have one entry for each predecessor of its parent basic block!"

✅ #51366 Instruction does not dominate all uses! Crash with -O2 in llvm/lib/Transforms/Vectorize/LoopVectorize.cpp:1047

✅ #54867 [LoopVectorizer] Assertion `hasVectorValue (Def, Instance.Part)' failed.

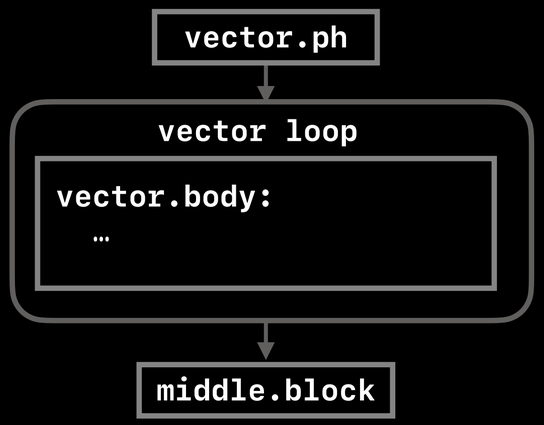
✅ #55167 [LoopVectorizer] Miscompile due to incorrect tail computation

✅ #55459 [LoopVectorizer] Assertion `!verifyFunction(*L->getHeader()->getP arent(), &dbgs())' failed.

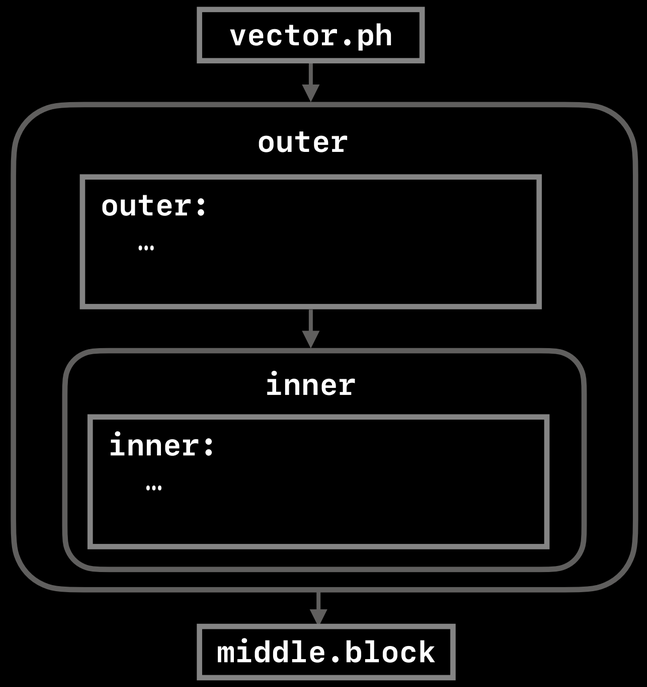
✅ #58811 Clang failed with -O3: Assertion `DT.dominates(RHead, LHead) && "No dominance between recurrences used by one SCEV?"' failed.

A Step Towards Unifying Inner and Outer Loop Codepaths

VPlan for inner loop

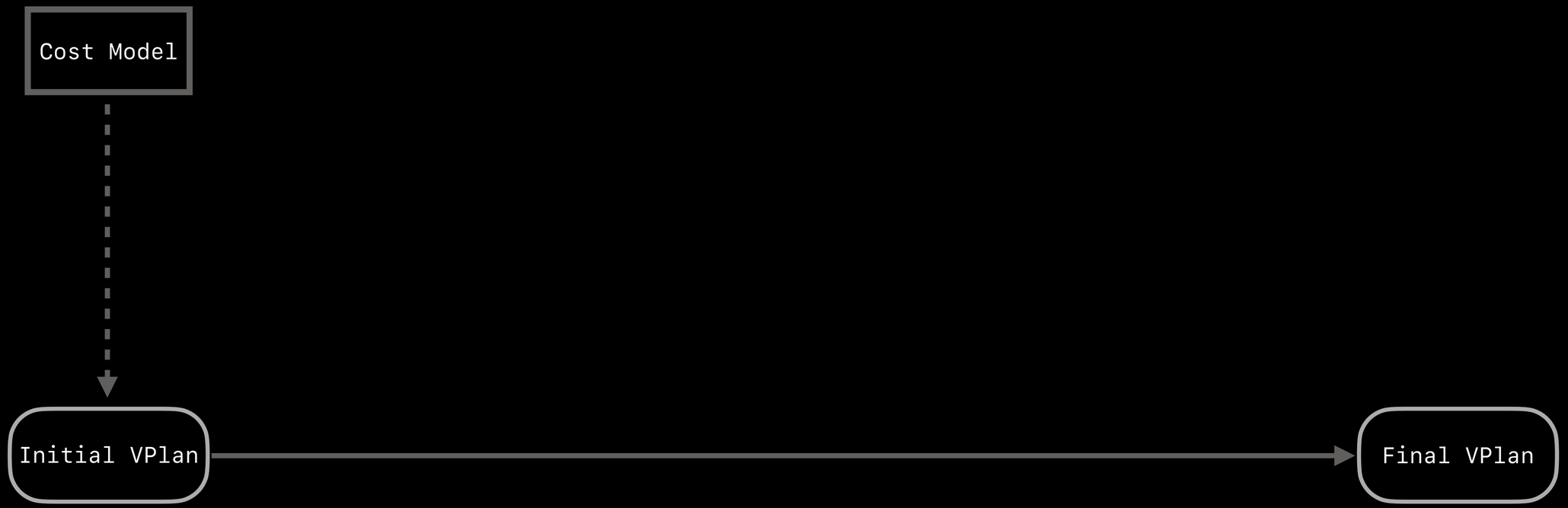


VPlan for outer loop

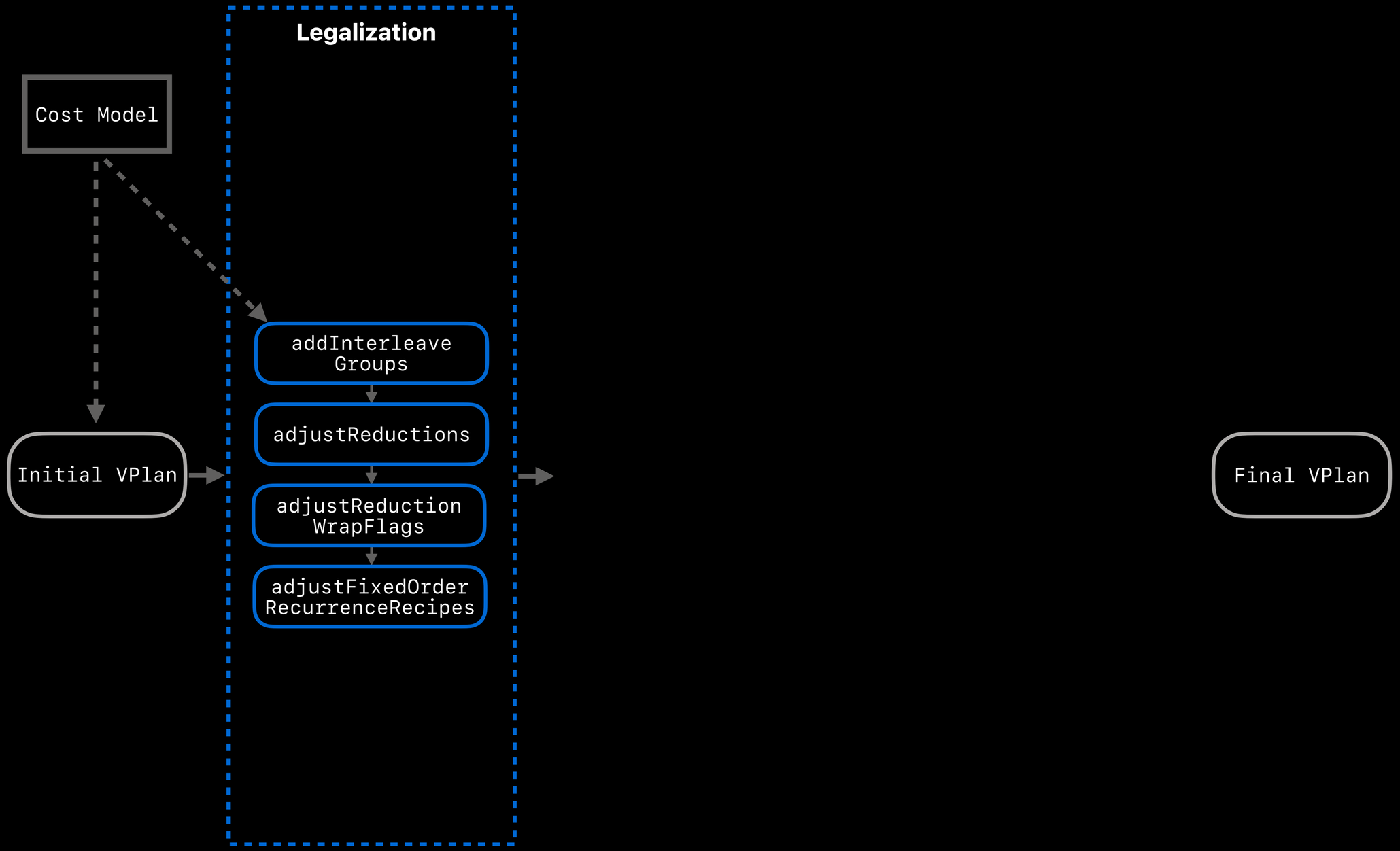


VPlans share the same structure and codegen path

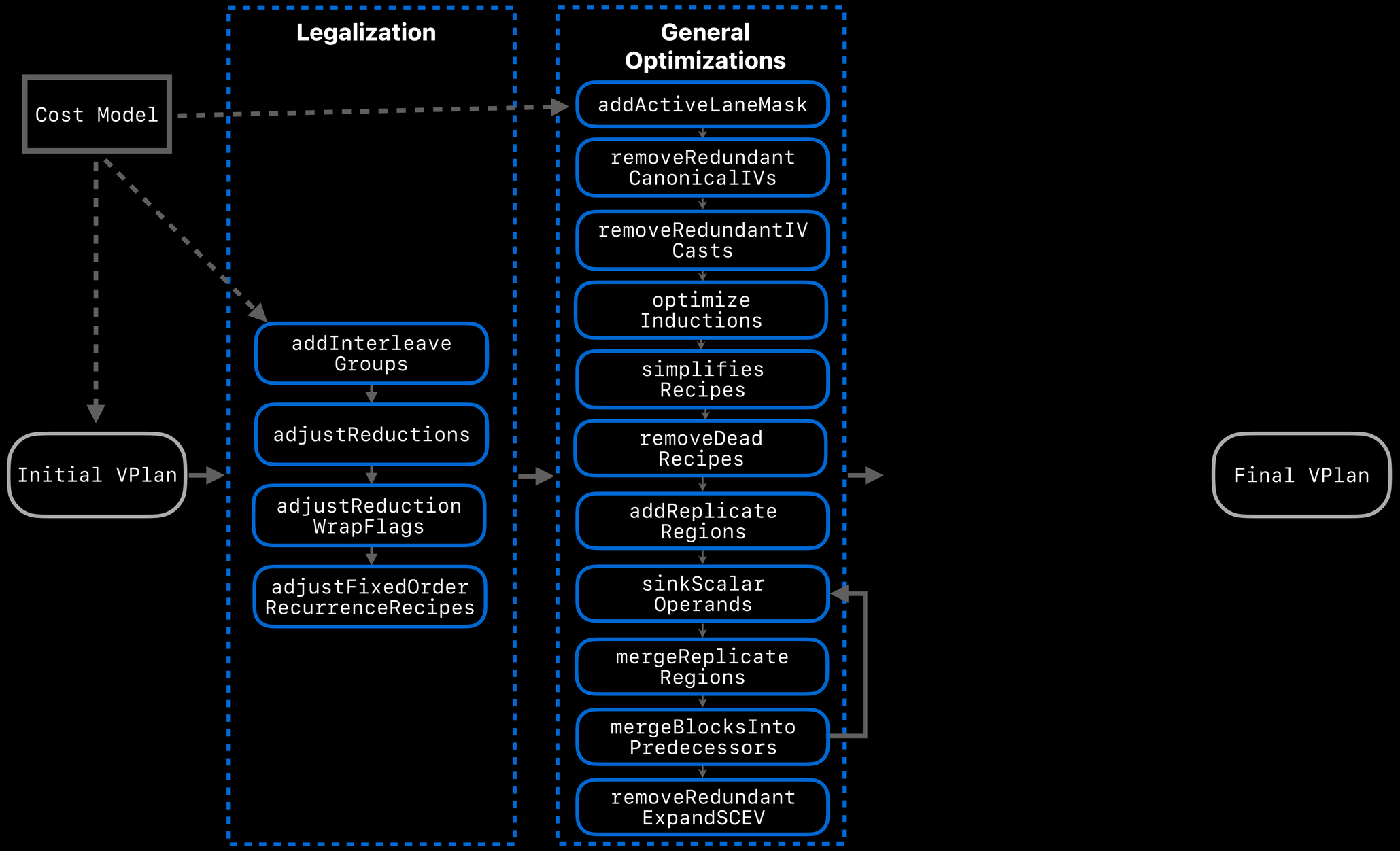
VPlan Transformation Pipeline



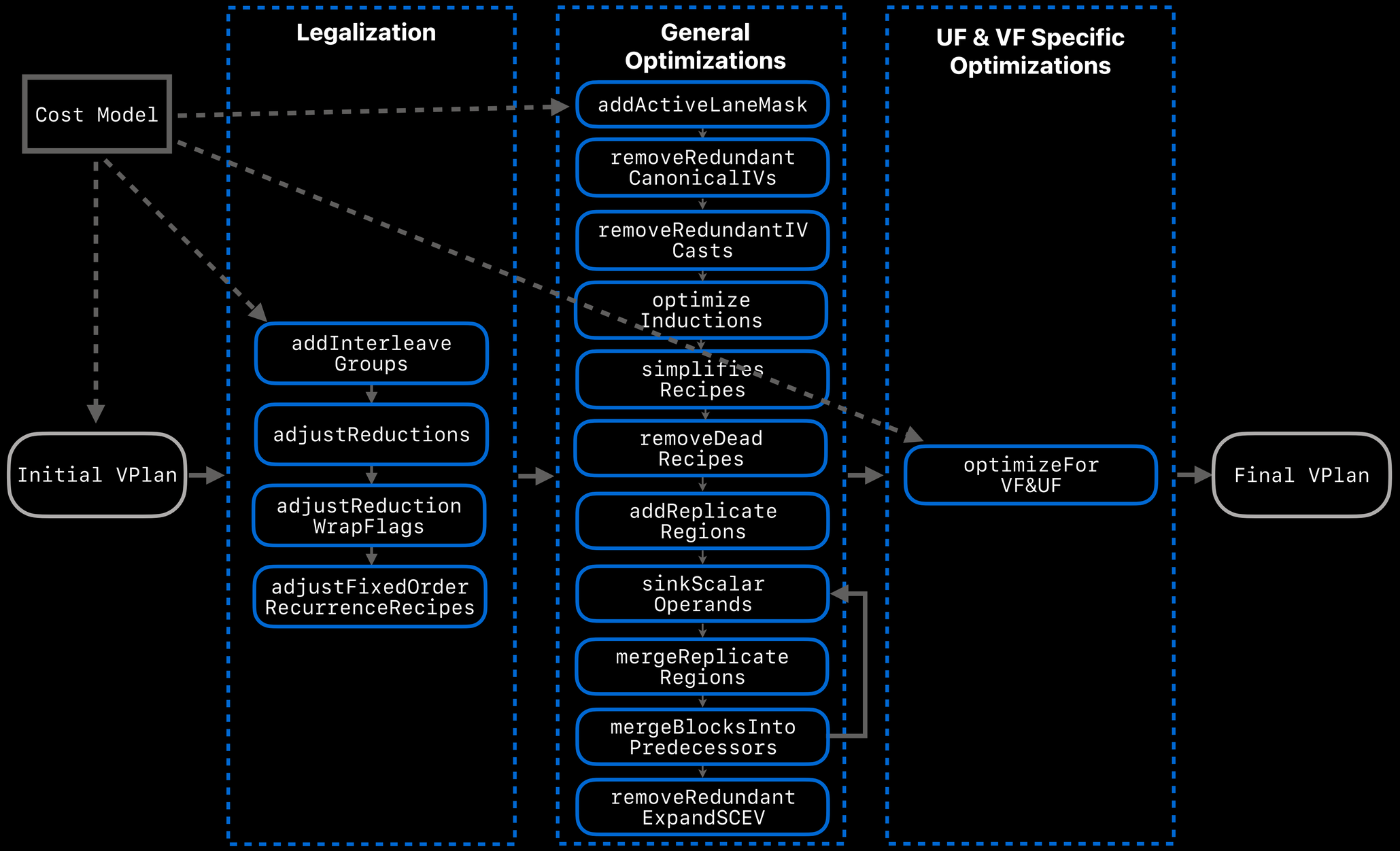
VPlan Transformation Pipeline



VPlan Transformation Pipeline



VPlan Transformation Pipeline

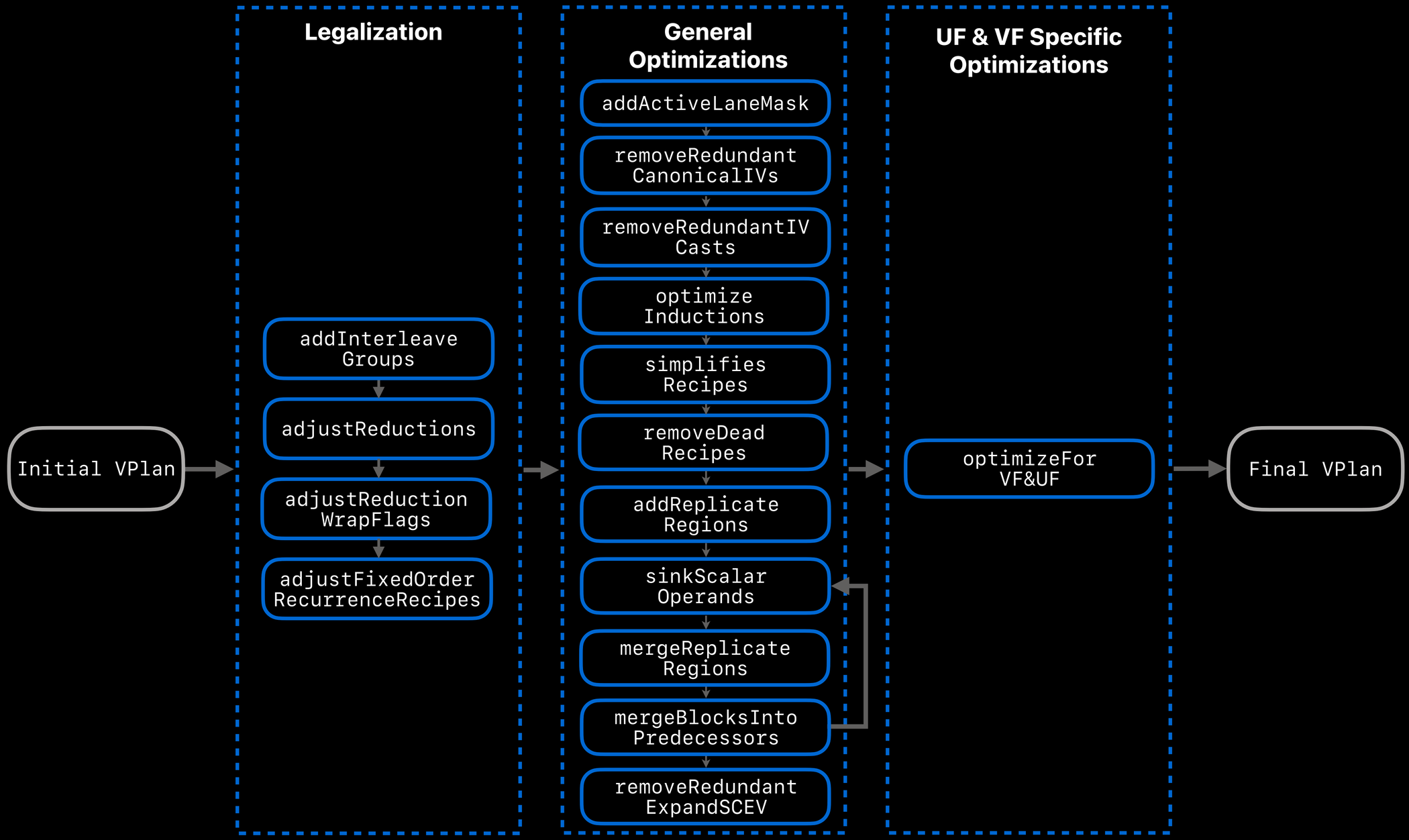


Guiding Principles of VPlan Development

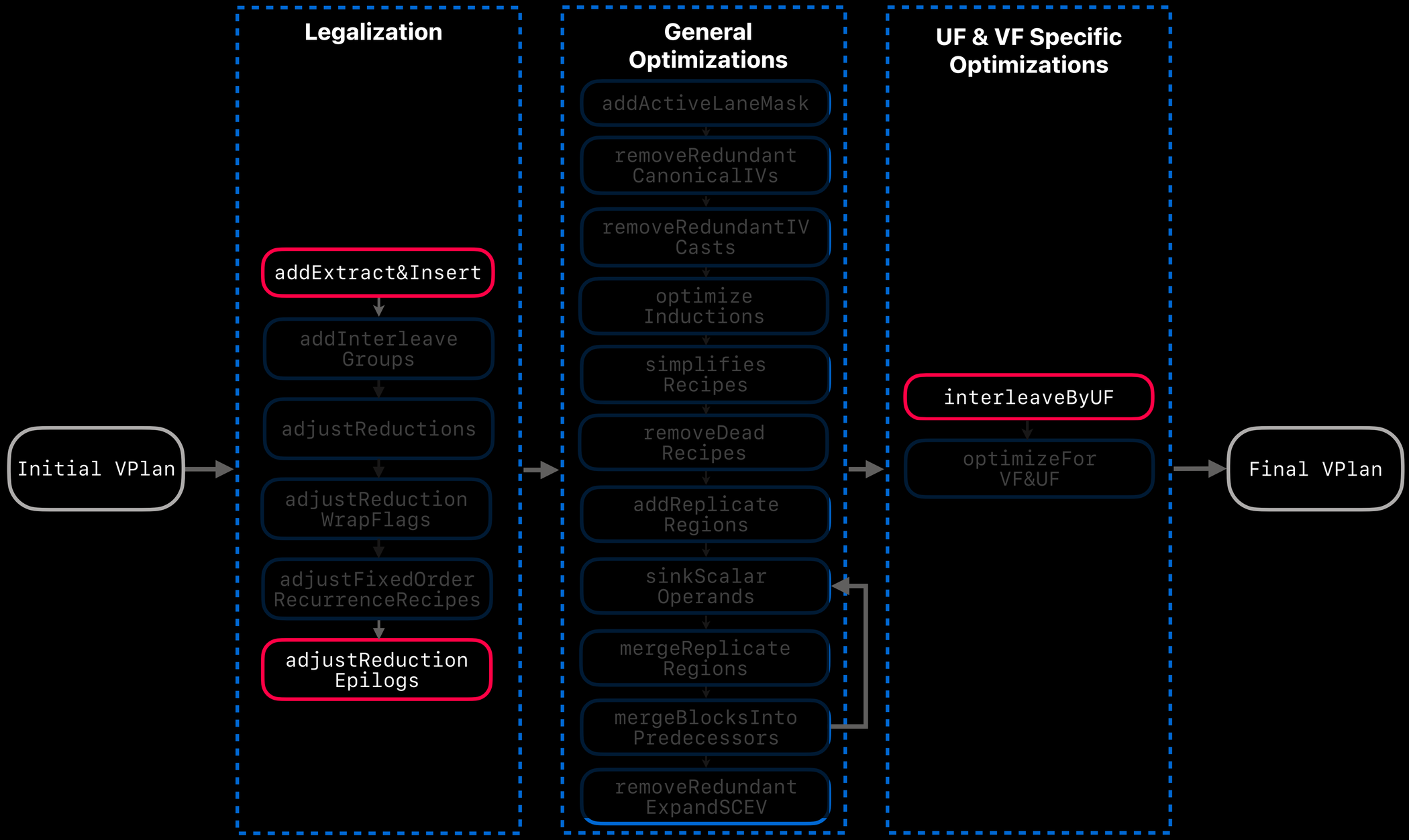
- Always-on in-place refactoring to retain quality and integrate continuously
- Gradual refactoring into multiple VPlan-to-VPlan transforms to reduce complexity
  - Simplify VPlan creation: generating initial VPlan from IR
  - Canonicalize then Specialize
  - Simplify VPlan execution: generating IR from final VPlan
- Self-contained as much as possible to promote modularity
  - Independent of underlying IR
  - Independent of cost model & legality
  - Model information explicitly in VPlan as needed

Future Roadmap

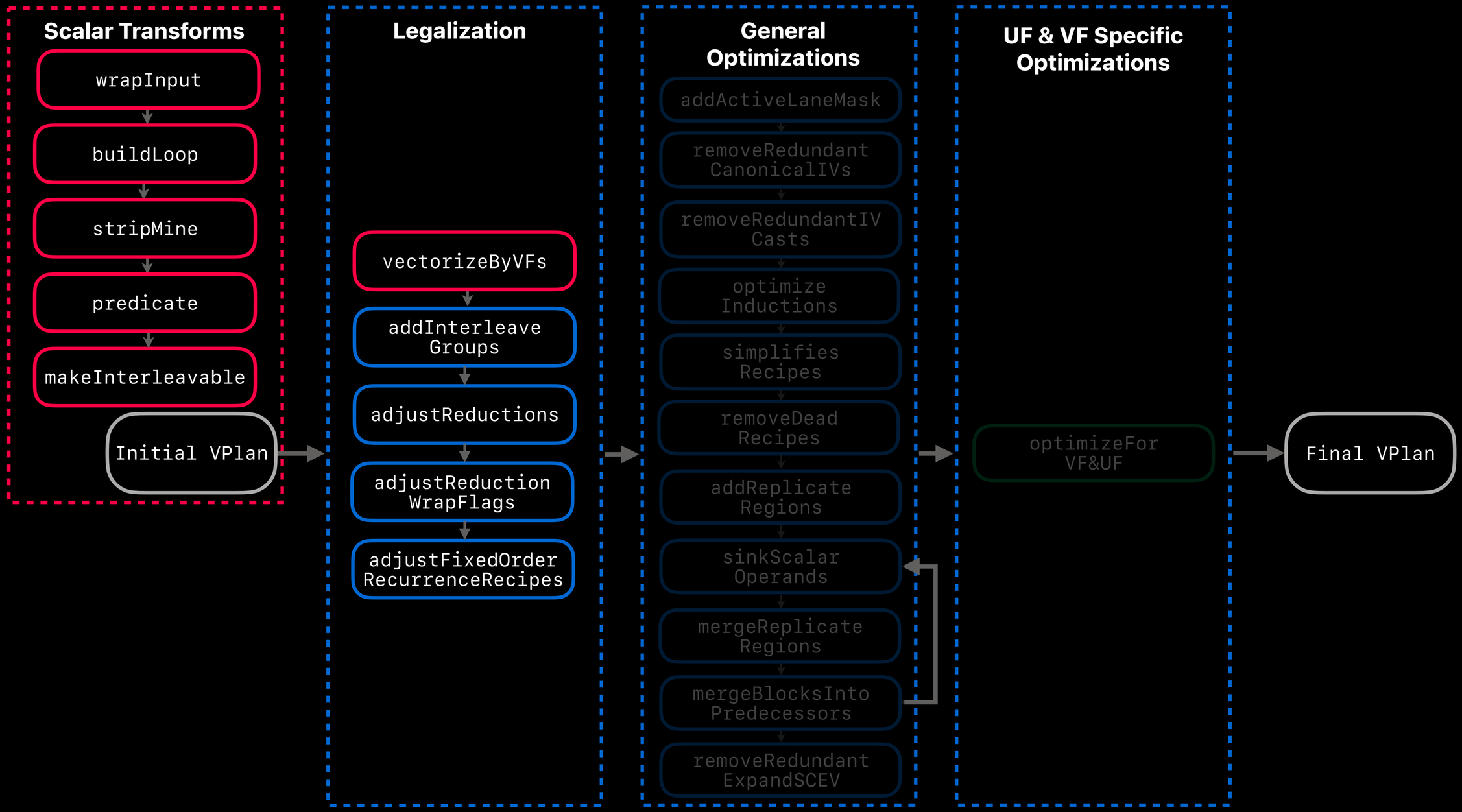
Recap: VPlan Transformation Pipeline '23



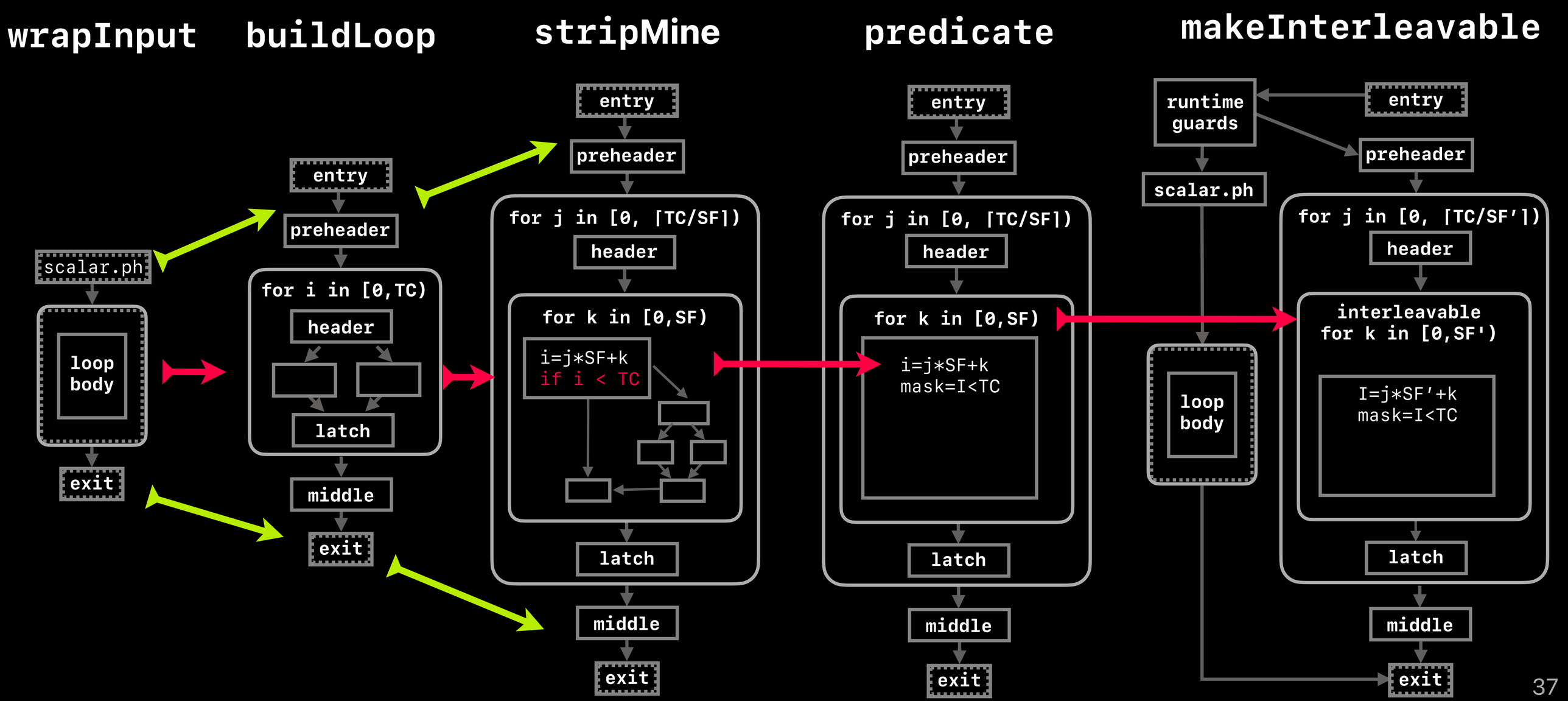
Roadmap Direction 0: Simplify VPlan Execution



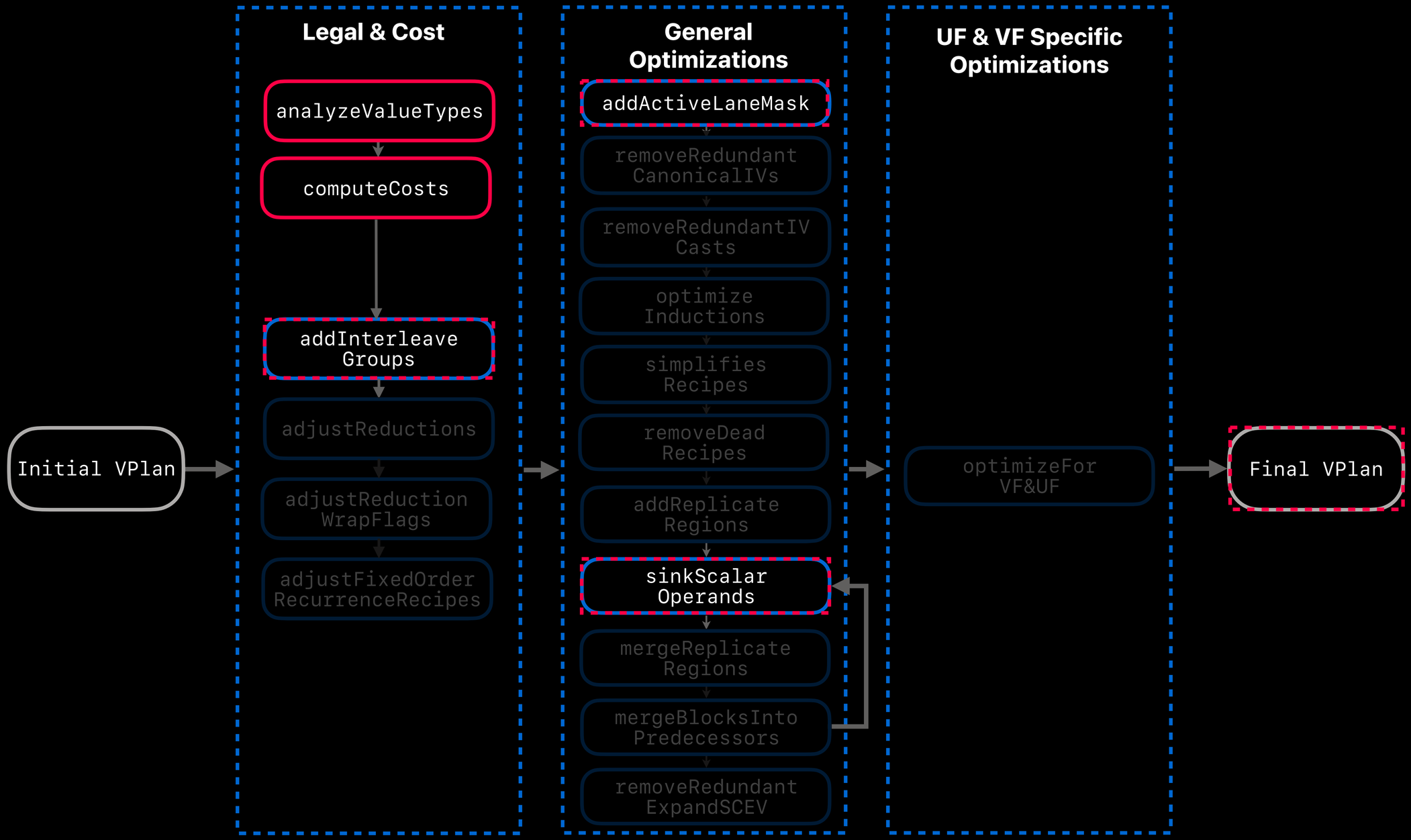
Roadmap Direction 1: Refactor Initial VPlan Creation



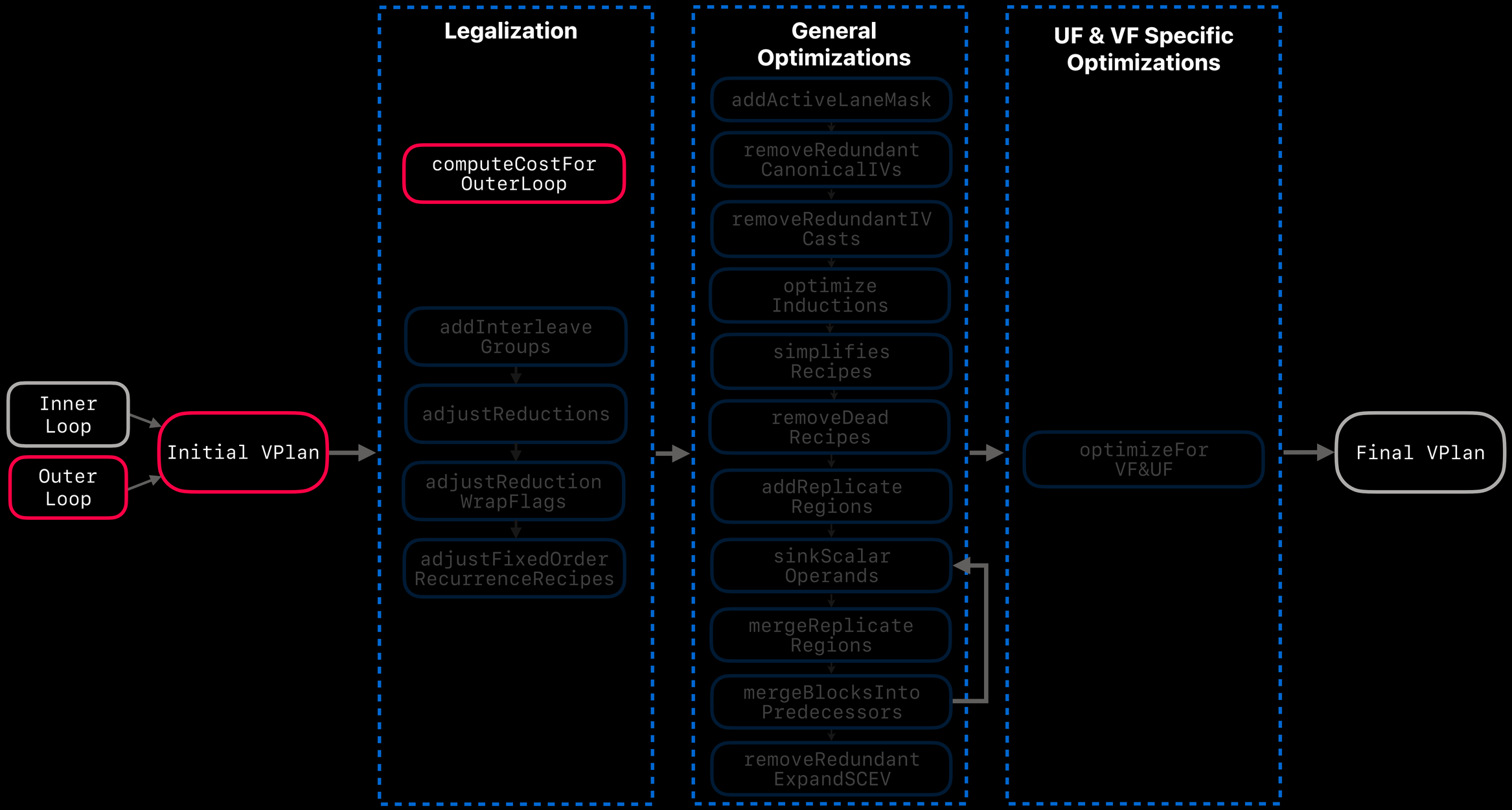
Direction 1: Refactor VPlan Creation



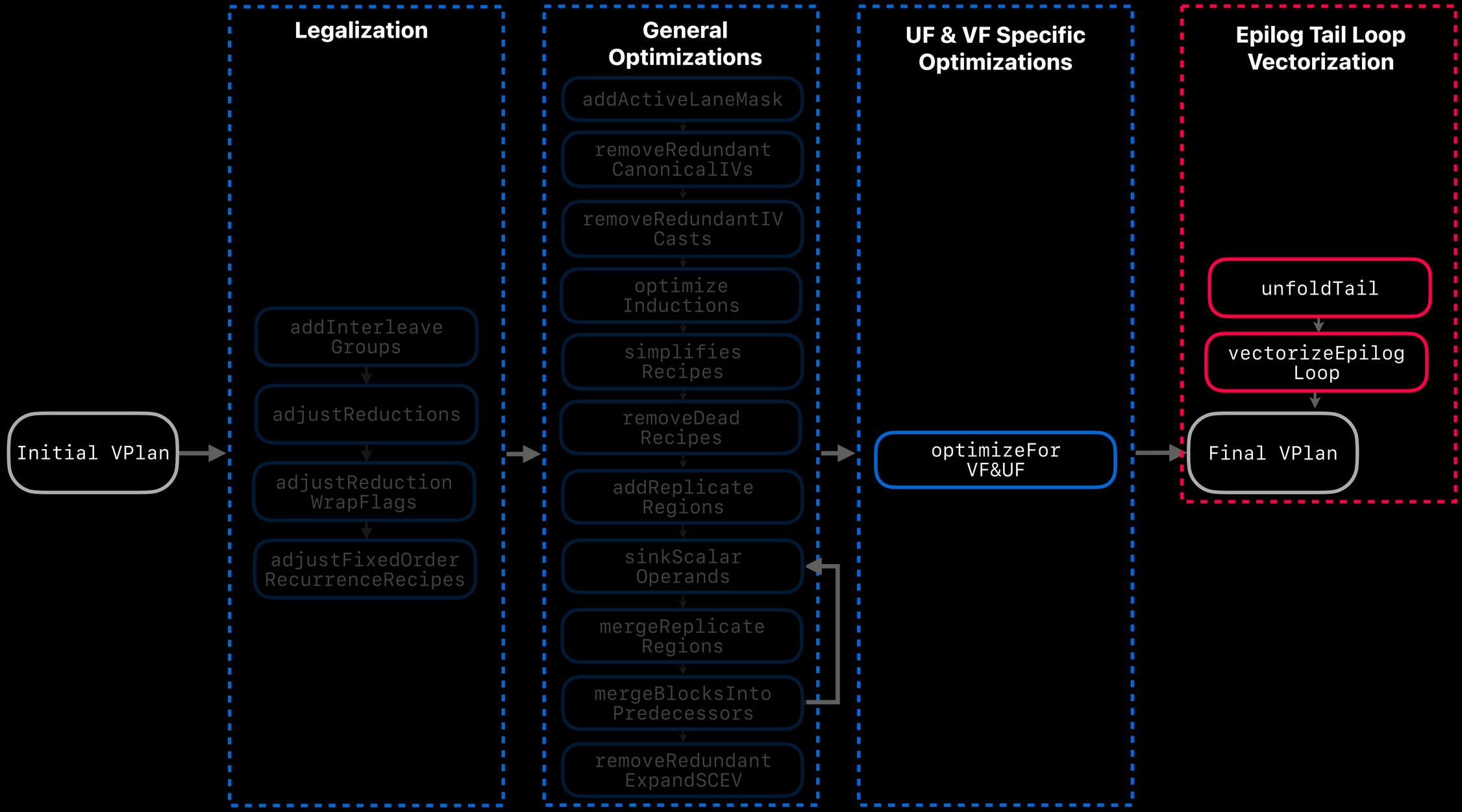
Roadmap Direction 2: Teach VPlan to Compute and use Cost



Roadmap Direction 3: Converge Inner and Outer loop pipelines

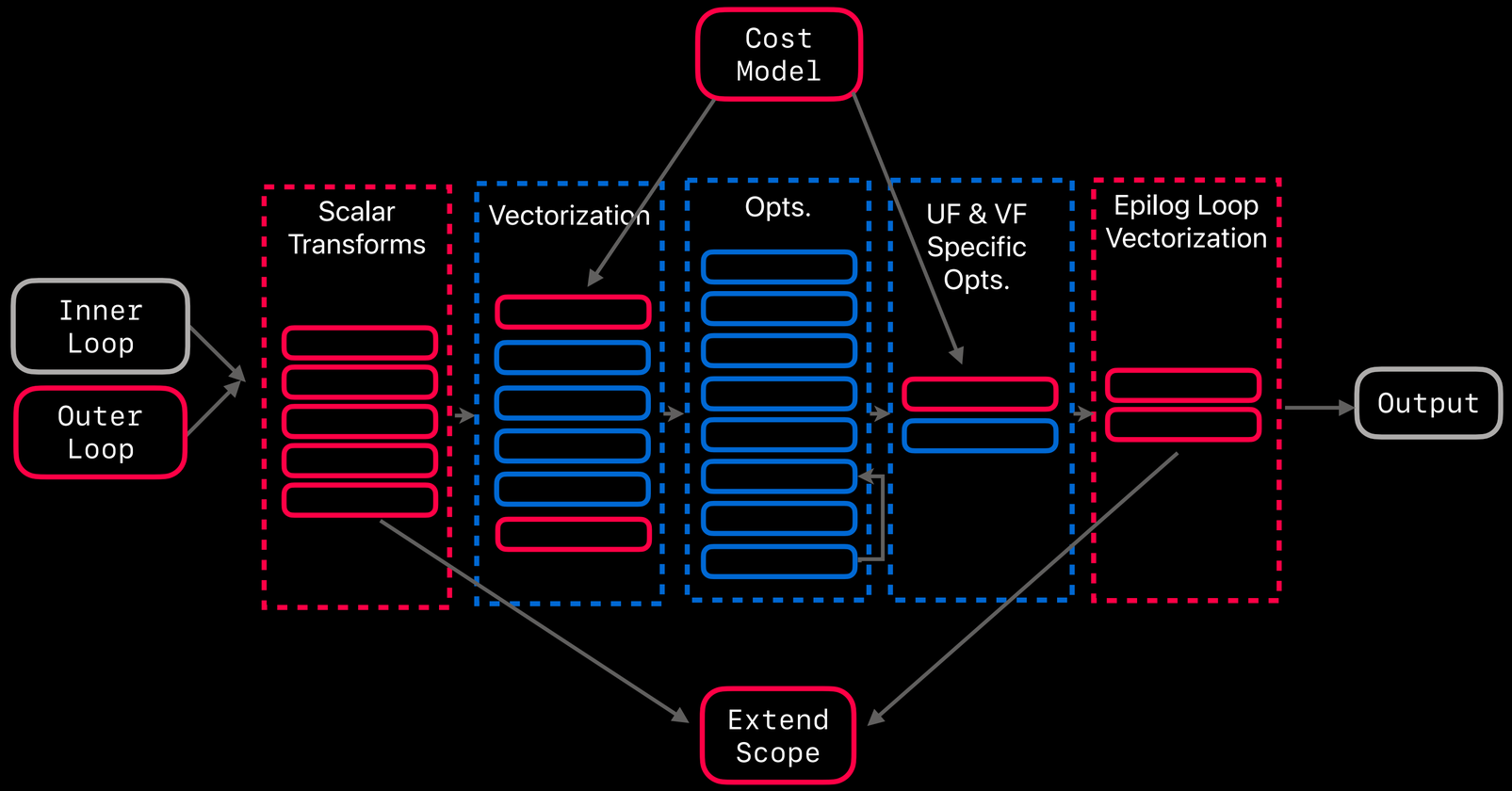


Roadmap Direction 4: Vectorize Epilog via VPlan Transform



Conclusions

- VPlan is ready for transformations now!
- Multiple advancements in recent years
- Multiple directions forward:

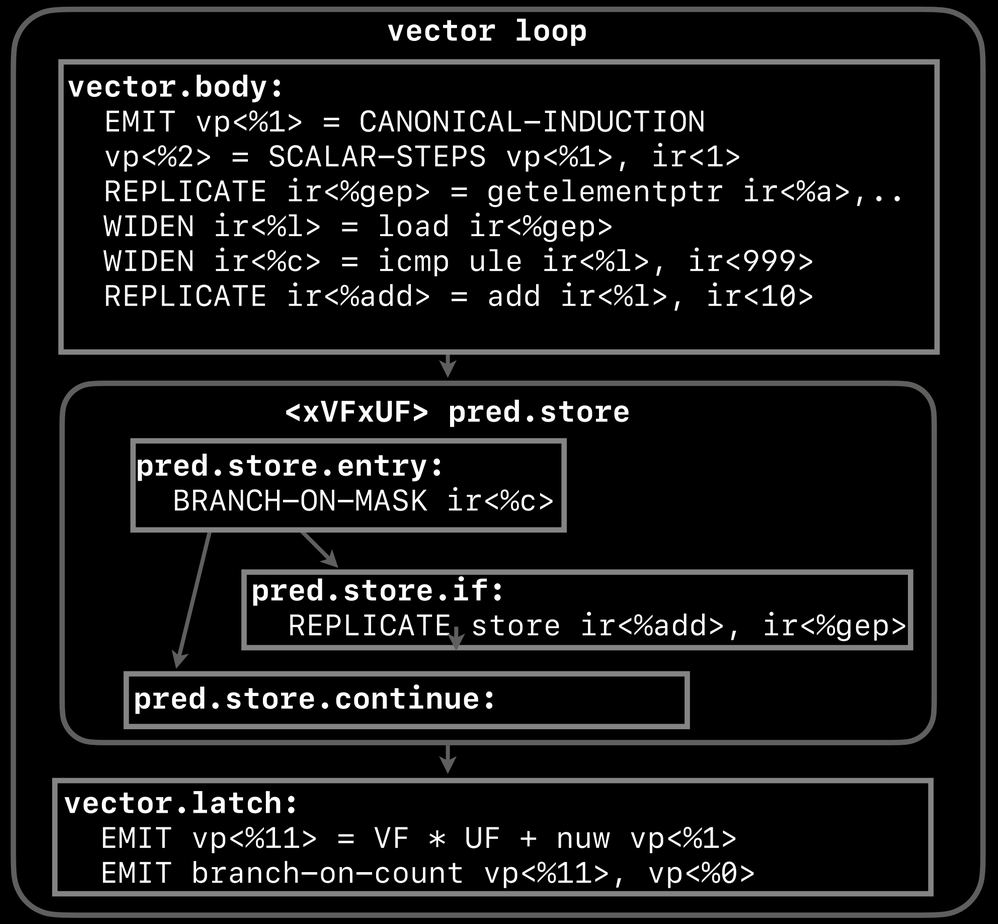


See you at the VPlan roundtable!
Oct 12 10:30 - 11:00

Questions?

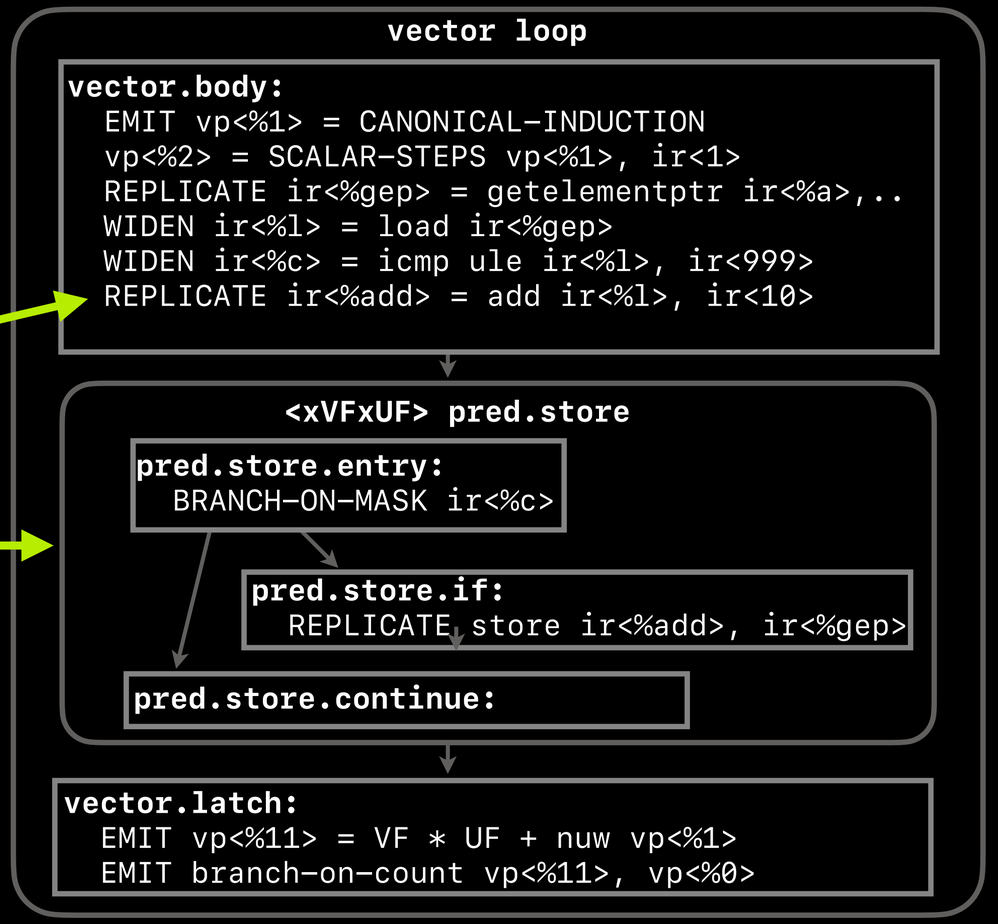
Example: Gradual Lowering for Replicate Regions

```
entry:
  br label %loop

loop:
  %iv = phi i32 [ 0, %entry ], [ %iv.next, %latch ]
  %gep = getelementptr i32, ptr %a, i32 %iv
  %l = load i32, ptr %gep
  %c = icmp ule i32 %l, 999
  br i1 %c, label %then, label %latch

then:
  %add = add i32 %l, 10
  store i32 %add, ptr %gep
  br label %latch

latch:
  %iv.next = add nuw nsw i32 %iv, 1
  %exitcond.not = icmp eq i32 %iv.next, %N
  br i1 %exitcond.not, label %exit, label %loop

exit:
```

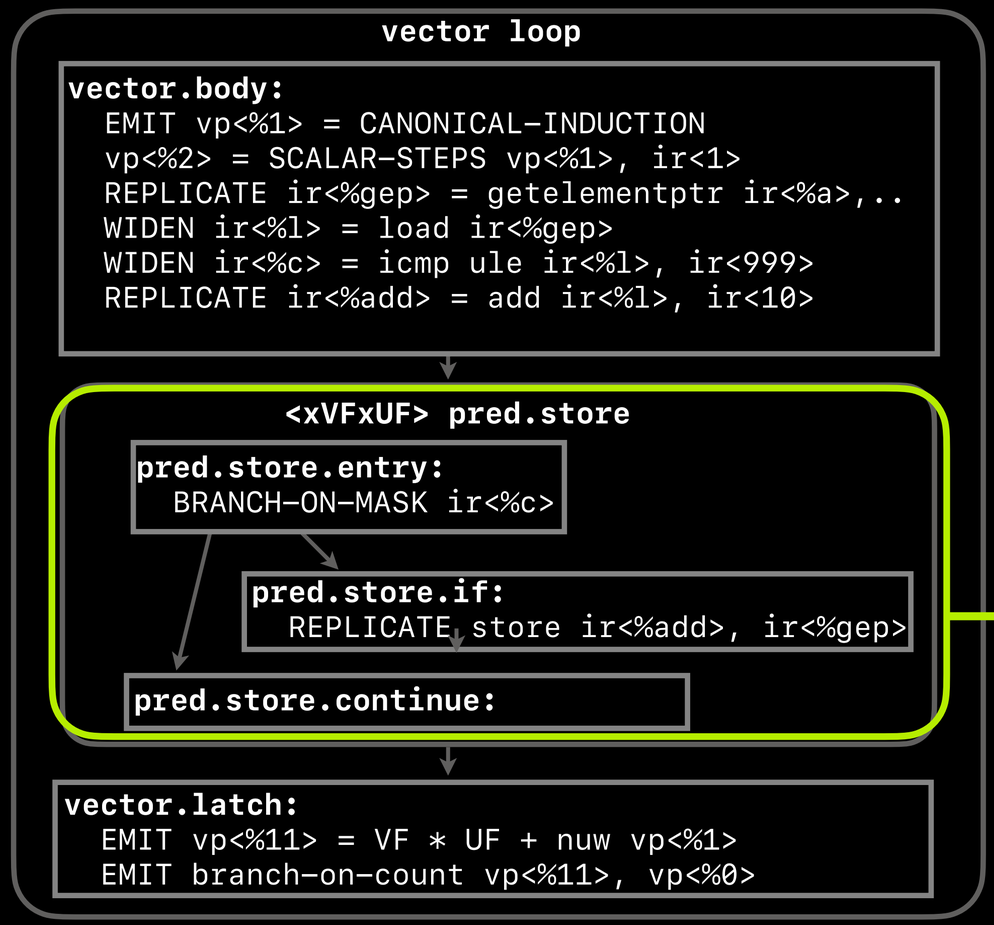


Example: Gradual Lowering for Replicate Regions

```
entry:
  br label %loop

loop:
  %iv = phi i32 [ 0, %entry ], [ %iv.next, %latch ]
  %gep = getelementptr i32, ptr %a, i32 %iv
  %l = load i32, ptr %gep
  %c = icmp ule i32 %l, 999
  br i1 %c, label %then, label %latch

then:
  %add = add i32 %l, 10
  store i32 %add, ptr %gep
  br label %latch

latch:
  %iv.next = add nuw nsw i32 %iv, 1
  %exitcond.not = icmp eq i32 %iv.next, %N
  br i1 %exitcond.not, label %exit, label %loop

exit:
```

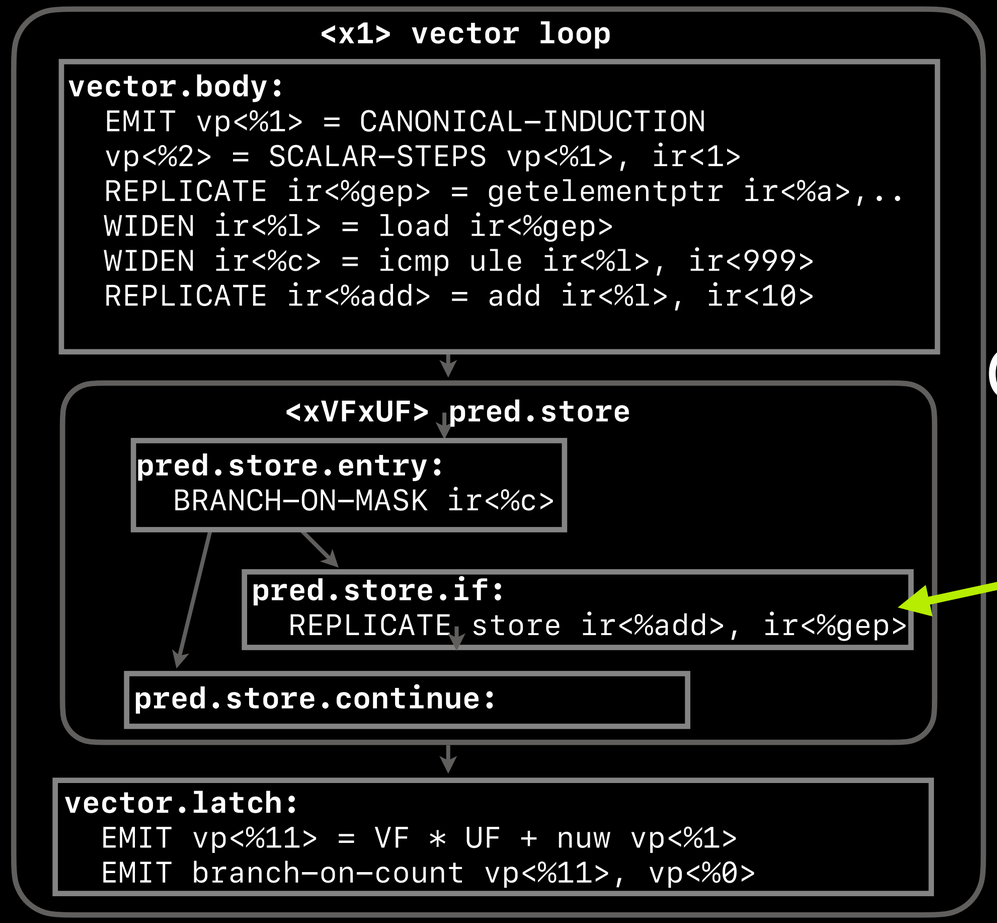


Example: Gradual Lowering for Replicate Regions




```
vector.body:
  %index = phi i32 [ 0, %vector.ph ], [ %index.next, %pred.store.continue5 ]
  %4 = add i32 %index, 0
  %5 = add i32 %index, 1
  %6 = getelementptr inbounds i32, ptr %a, i32 %4
  %7 = getelementptr inbounds i32, ptr %b, i32 %4
  %8 = getelementptr inbounds i32, ptr %b, i32 %5
  %9 = getelementptr inbounds i32, ptr %6, i32 0
  %wide.load = load <2 x i32>, ptr %9, align 4, !alias.scope !0
  %10 = icmp ule <2 x i32> %wide.load, <i32 999, i32 999>
  %11 = extractelement <2 x i1> %10, i32 0
  br i1 %11, label %pred.load.if, label %pred.load.continue

pred.store.if:
  %18 = extractelement <2 x i32> %wide.load, i32 0
  %19 = add i32 %18, %13
  store i32 %19, ptr %7, align 4, !alias.scope !3, !noalias !0
  br label %pred.store.continue

pred.store.continue:
  %20 = extractelement <2 x i1> %10, i32 1
  br i1 %20, label %pred.store.if4, label %pred.store.continue5

pred.store.if4:
  %21 = extractelement <2 x i32> %wide.load, i32 1
  %22 = add i32 %21, %16
  store i32 %22, ptr %8, align 4, !alias.scope !3, !noalias !0
  br label %pred.store.continue5

pred.store.continue5:
  %index.next = add nuw i32 %index, 2
  %23 = icmp eq i32 %index.next, %n.vec
  br i1 %23, label %middle.block, label %vector.body, !llvm.loop !5
```

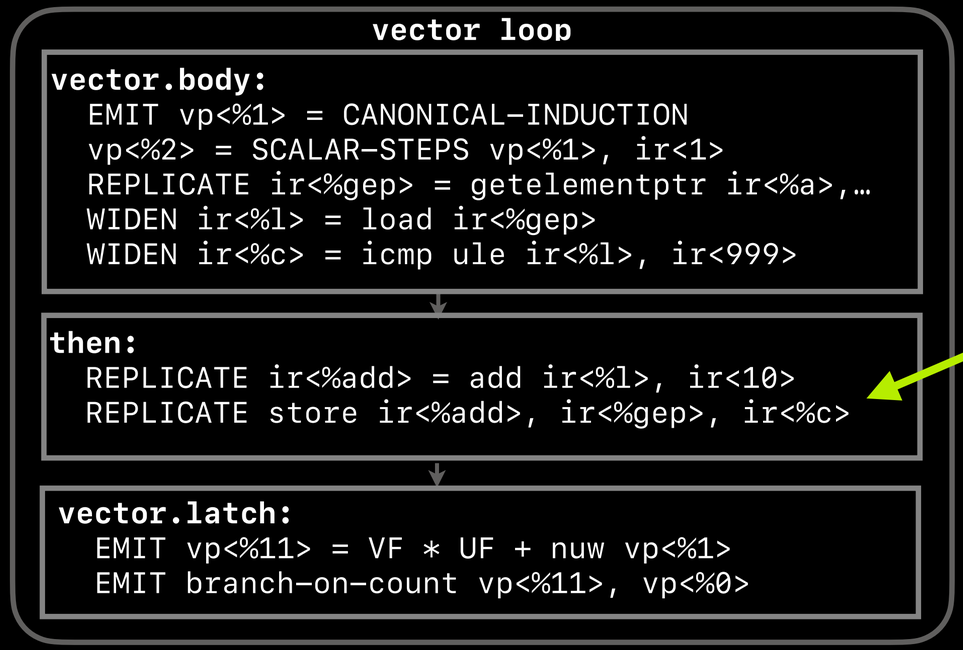
Example: Gradual Lowering for Replicate Regions



Consider sinking a recipe

after

Example: Gradual Lowering for Replicate Regions



Masked recipe

Example: Gradual Lowering for Replicate Regions