Loop Passes

Adding New Features While Reducing Technical Debt

Michael Zolotukhin

Adding New Features

- Implementation
- Testing
- Review
- Maintenance
- Paying off technical debt

Technical Debt

- Bugs

  Oops! Didn’t see it!

- Non-optimal solutions

  Works for now! It’s just a temporary solution.

- Unfinished solutions

  That’s enough for my case! We’ll finish it… one day.

Loop Unrolling
New Heuristics

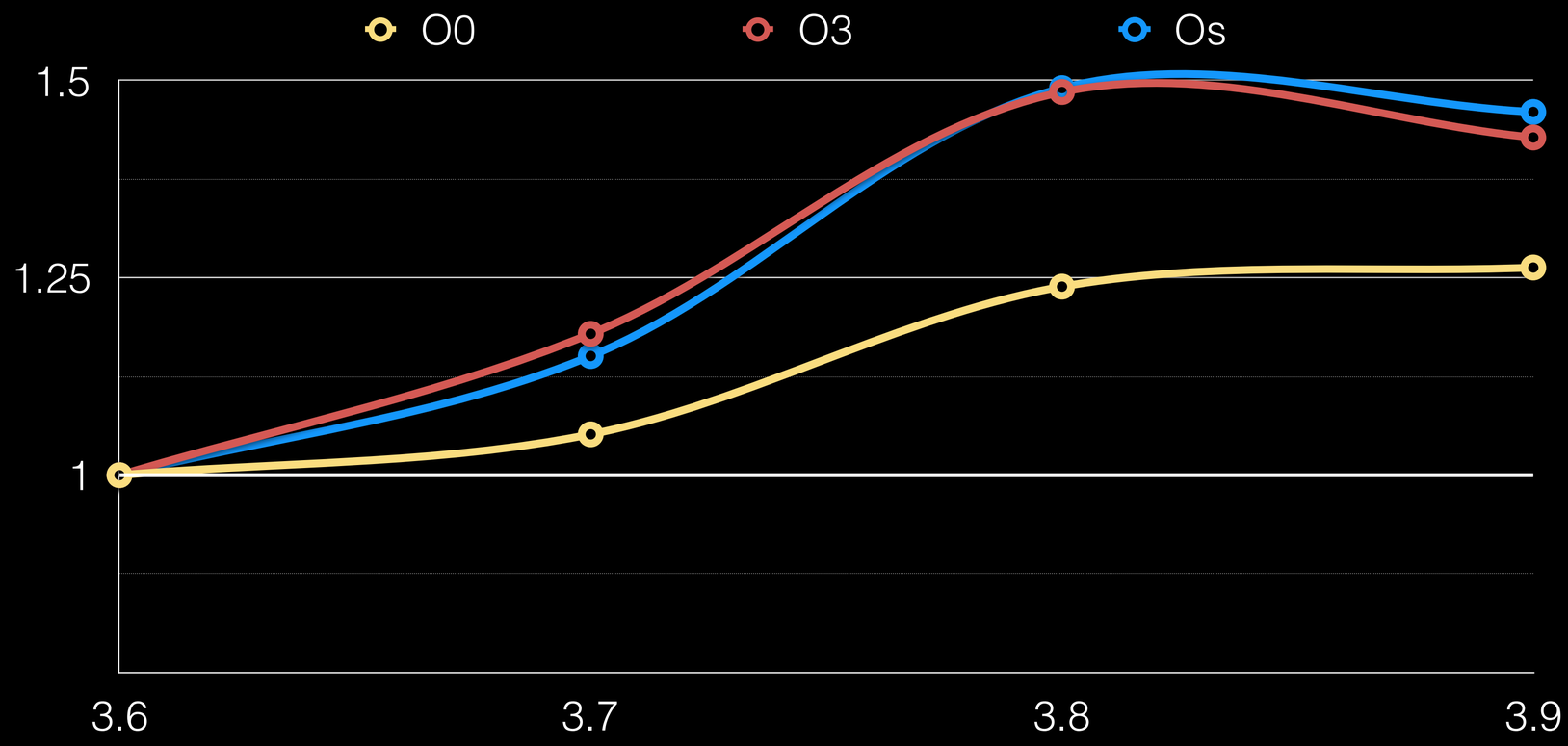
- Analyze loop body
- Predict potential outcomes of other optimizations if we unroll

LLVMDev 2015: “Advances in Loop Analysis Frameworks and Optimizations”

Loop Unrolling
Effects

- Performance improves
- Code size increases
- Compile time increases

Compile Time



Loop Unrolling
Stress Testing

- Check compile time impact
- Tune optimization thresholds
- Look for potential problems

Pass Structure

```
MPM.add(createLoopRotatePass());
MPM.add(createCFGSimplificationPass());
MPM.add(createIndVarSimplifyPass());
MPM.add(createSimpleLoopUnrollPass());
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  SimplifyCFG();
  IndVars(L);
  Unroll(L);
}
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
}
SimplifyCFG();
Prepare(F);
for (Loop *L : F) {
  IndVars(L);
}
Prepare(F);
for (Loop *L : F) {
  Unroll(L);
}
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
}
SimplifyCFG();
Prepare(F);
for (Loop *L : F) {
  IndVars(L);
}
Prepare(F);
for (Loop *L : F) {
  Unroll(L);
}
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
}
SimplifyCFG();
Prepare(F);
for (Loop *L : F) {
  IndVars(L);
}
Prepare(F);
for (Loop *L : F) {
  Unroll(L);
}
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
}
SimplifyCFG();
Prepare(F);
for (Loop *L : F) {
  IndVars(L);
  Unroll(L);
}
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
}
SimplifyCFG();
Prepare(F);
for (Loop *L : F) {
  IndVars(L);
  Unroll(L);
}
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  LoopSimplifyCFG(L);
  IndVars(L);
  Unroll(L);
}
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  LoopSimplifyCFG(L);
  IndVars(L);
  Unroll(L);
}
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  LoopSimplifyCFG(L);
  IndVars(L);
  Unroll(L);
}
```

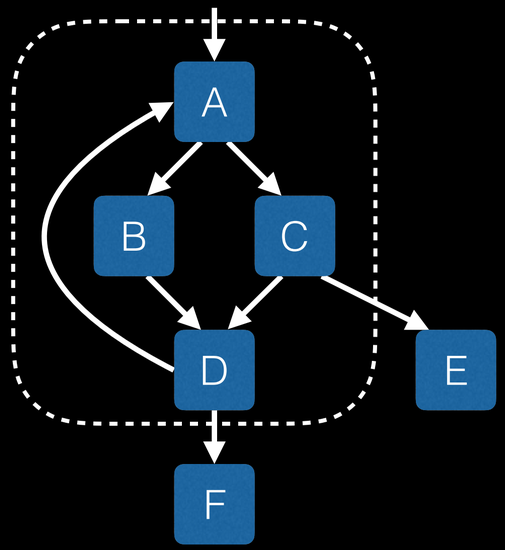
```
BuildDT(F);
BuildLCSSA(F);
SimplifyLoops(F);
```

Pass Structure

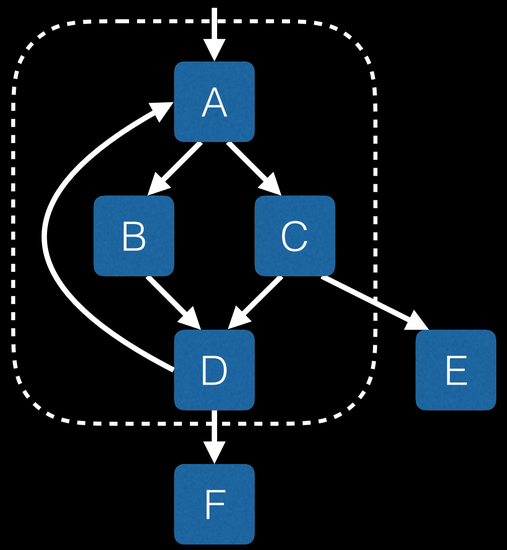
```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  SimplifyCFG();
  IndVars(L);
  Unroll(L);
}
```

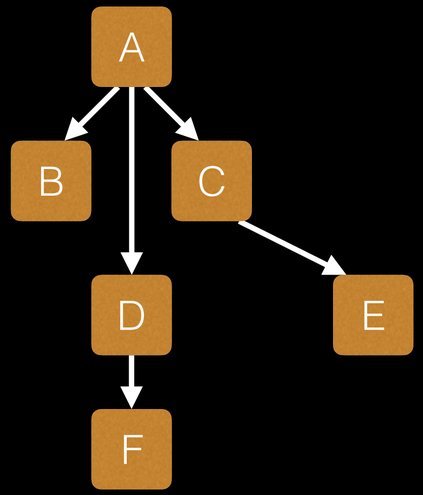
```
Transform(L);
RebuildDT(F);
```

Updating DomTree

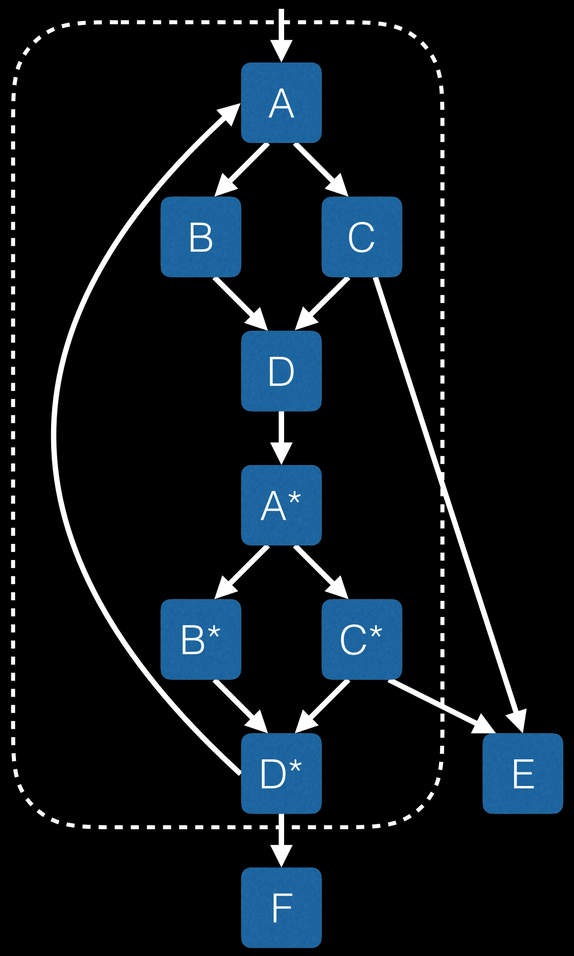


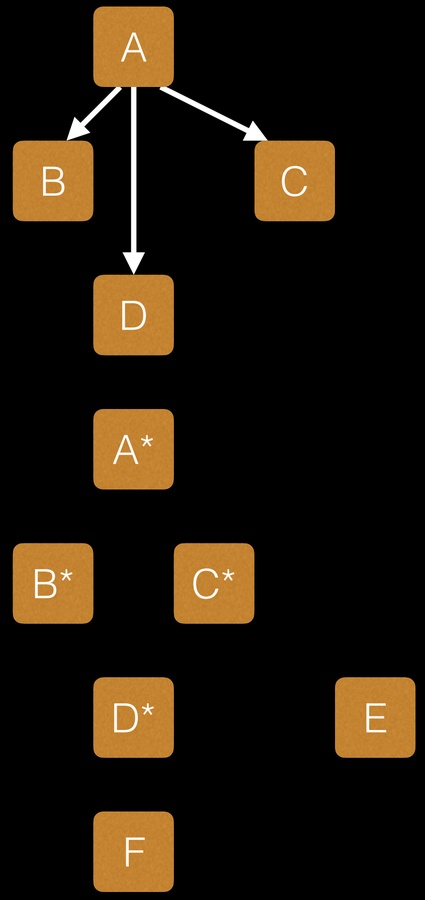
Updating DomTree



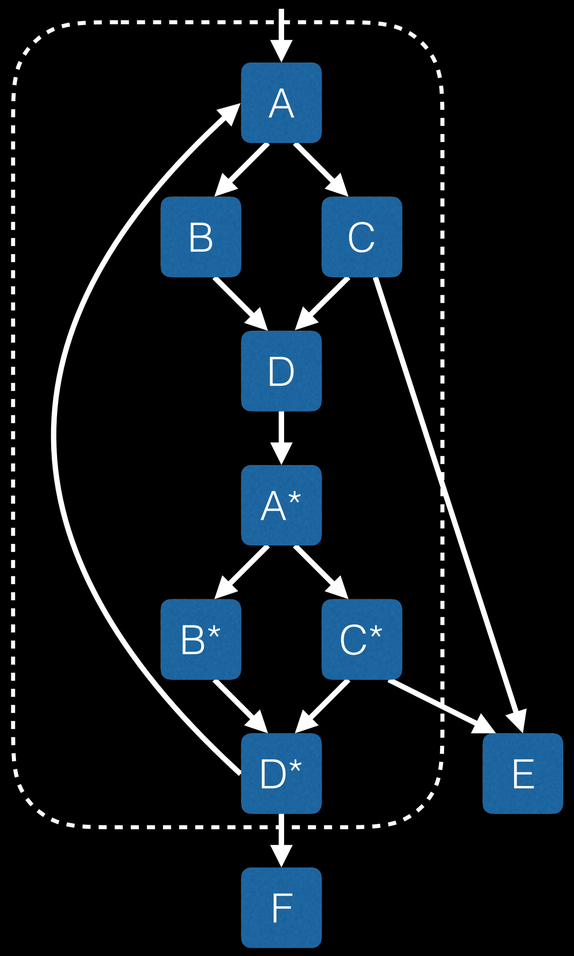


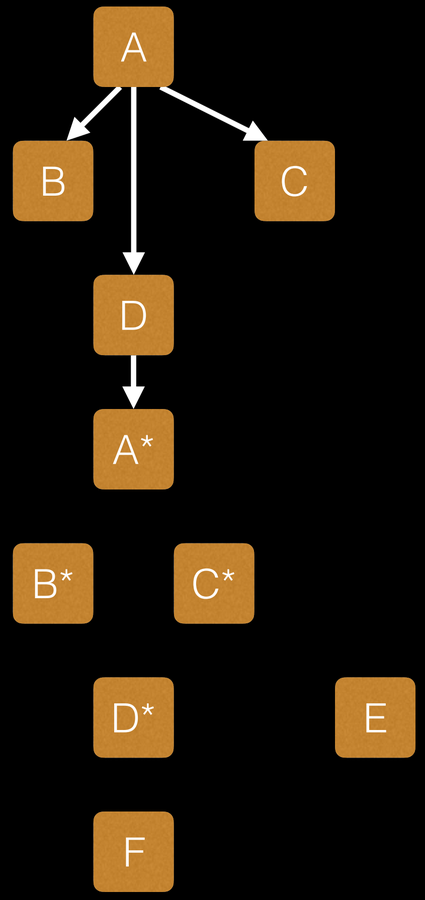
Updating DomTree



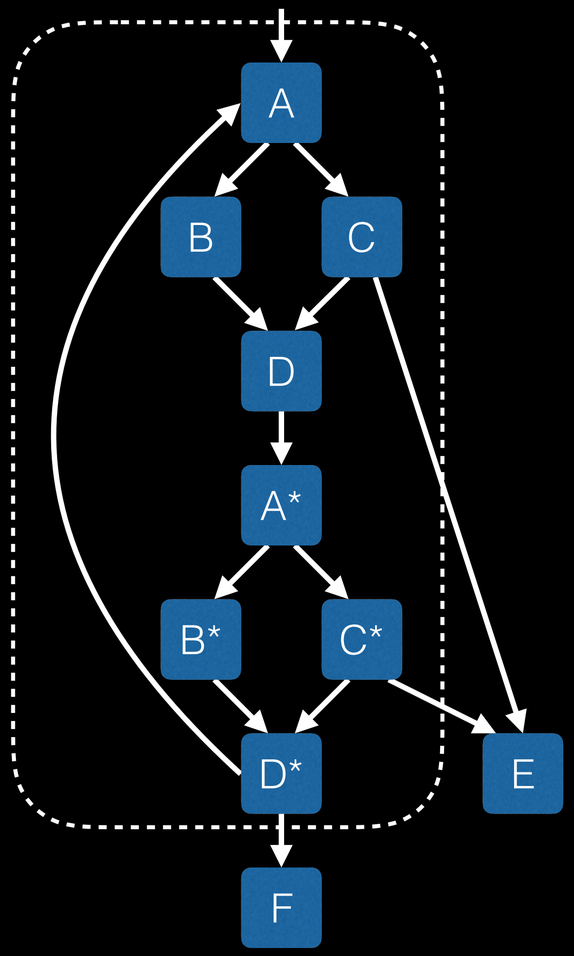


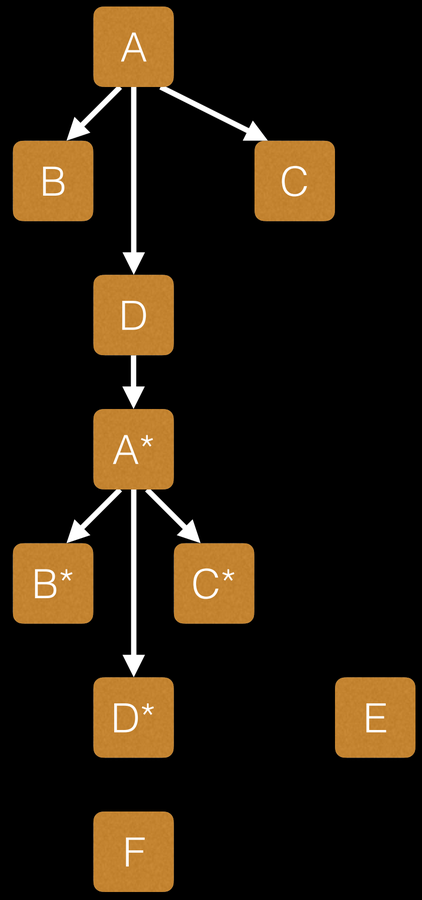
Updating DomTree



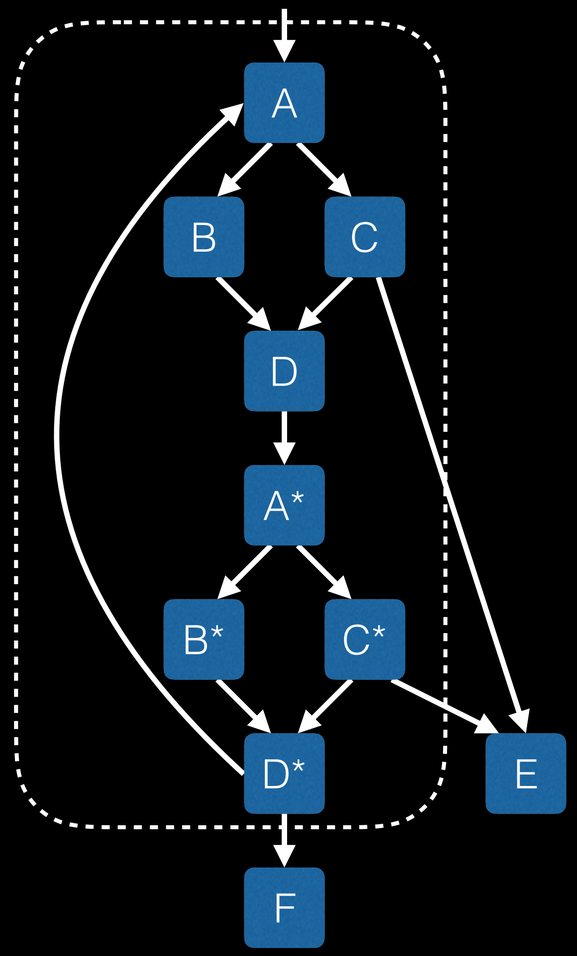


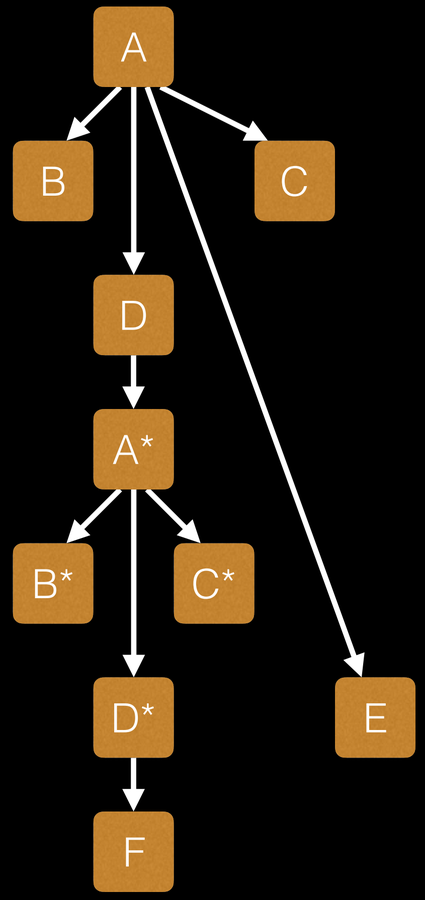
Updating DomTree





Updating DomTree





Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  SimplifyCFG();
  IndVars(L);
  Unroll(L);
}
```

```
Transform(L);
RebuildDT(F);
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  SimplifyCFG();
  IndVars(L);
  Unroll(L);
}
```

```
Transform(L);
UpdateDT(L);
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  SimplifyCFG();
  IndVars(L);
  Unroll(L);
}
```

```
Transform(L);
UpdateDT(L);
RebuildLCSSA(F);
```

Loop Closed SSA

- All definitions are used only inside the same loop or in phis in exit blocks
- Predecessors of all exit blocks belong to this loop

Updating LCSSA

```
do {
  a = …;


  do {


    b = a + 1;
  } while (inner);


  c = b * 4;
} while (outer);
```

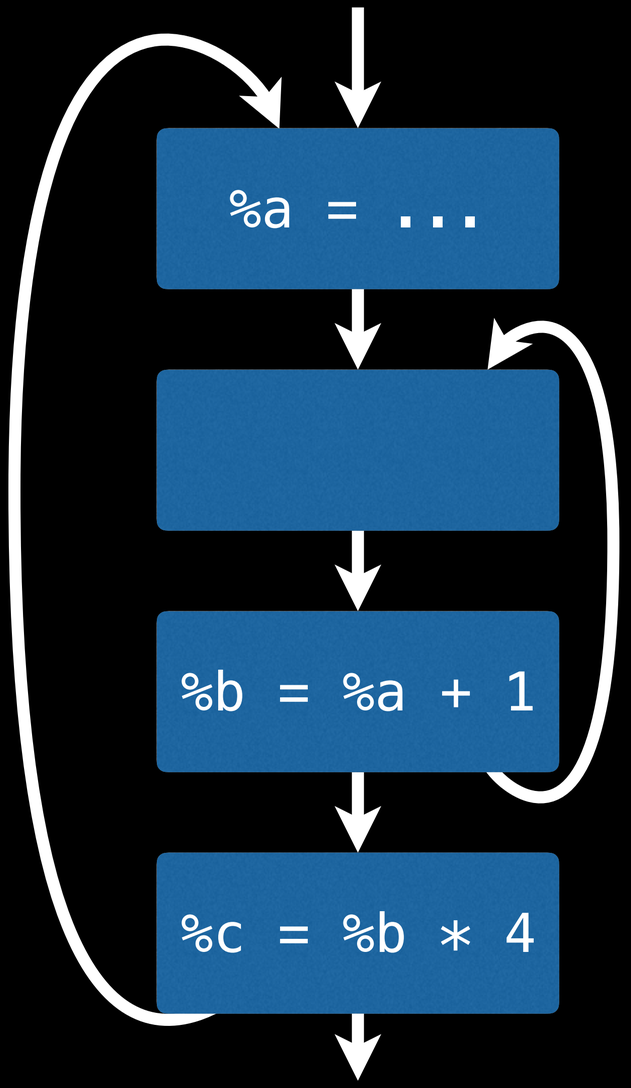
Updating LCSSA

```
do {
  a = …;


  do {


    b = a + 1;
  } while (inner);


  c = b * 4;
} while (outer);
```



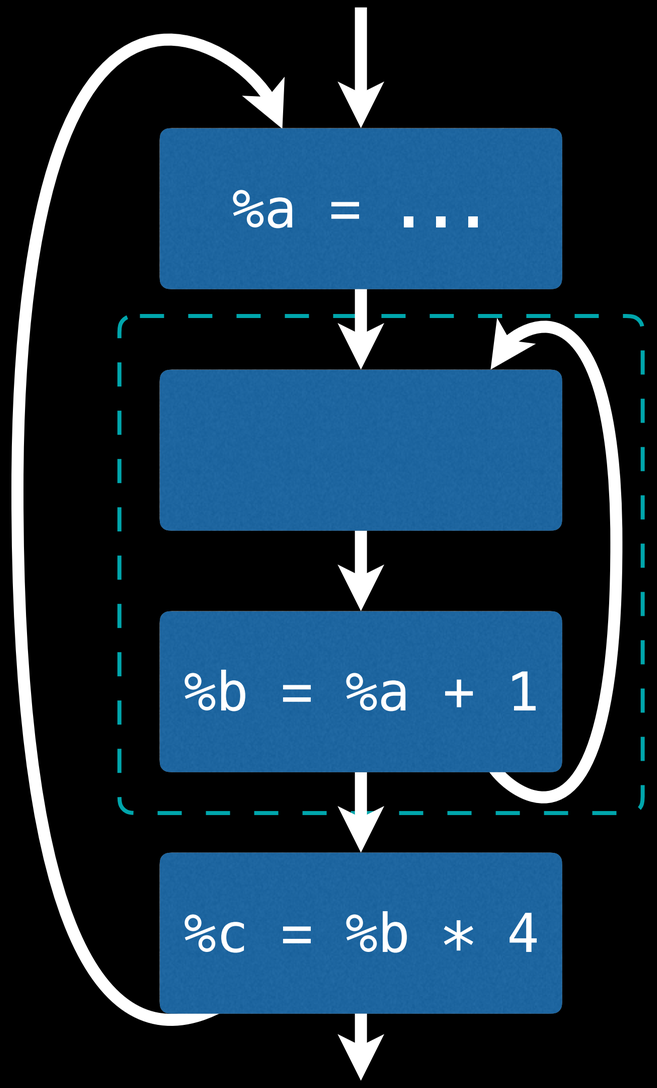
Updating LCSSA

```
do {
  a = …;

  do {


    b = a + 1;
  } while (inner);

  c = b * 4;
} while (outer);
```



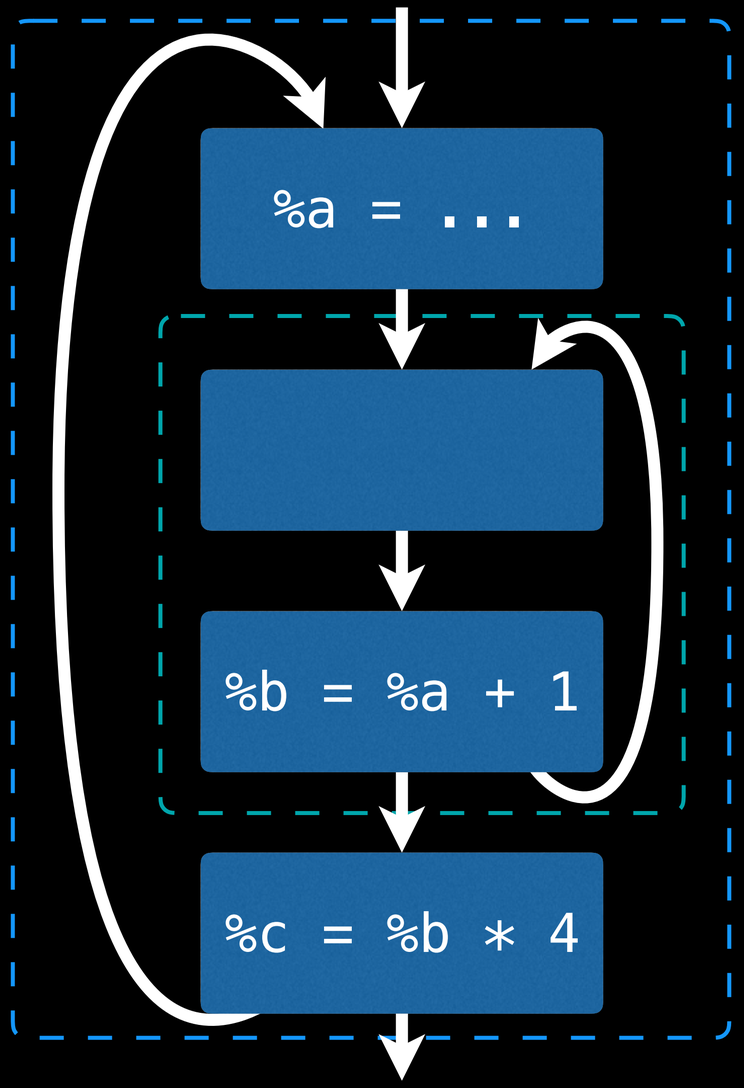
Updating LCSSA

```
do {
  a = …;

  do {


    b = a + 1;
  } while (inner);

  c = b * 4;
} while (outer);
```



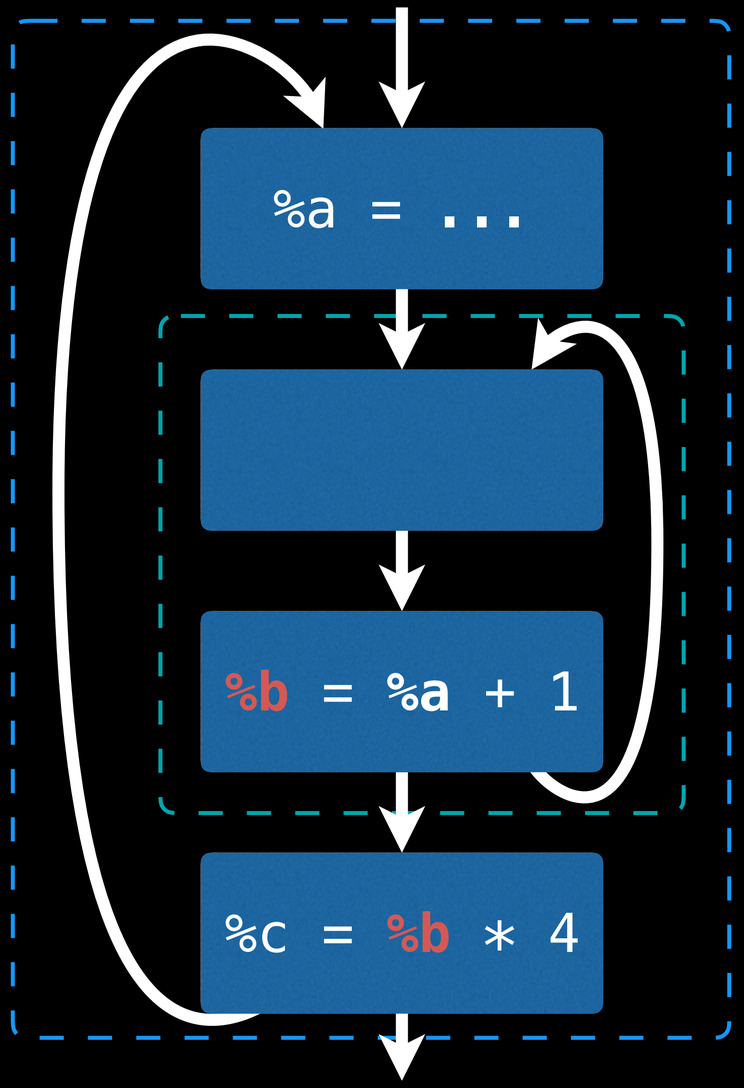
Updating LCSSA

```
do {
  a = …;

  do {


    b = a + 1;
  } while (inner);

  c = b * 4;
} while (outer);
```



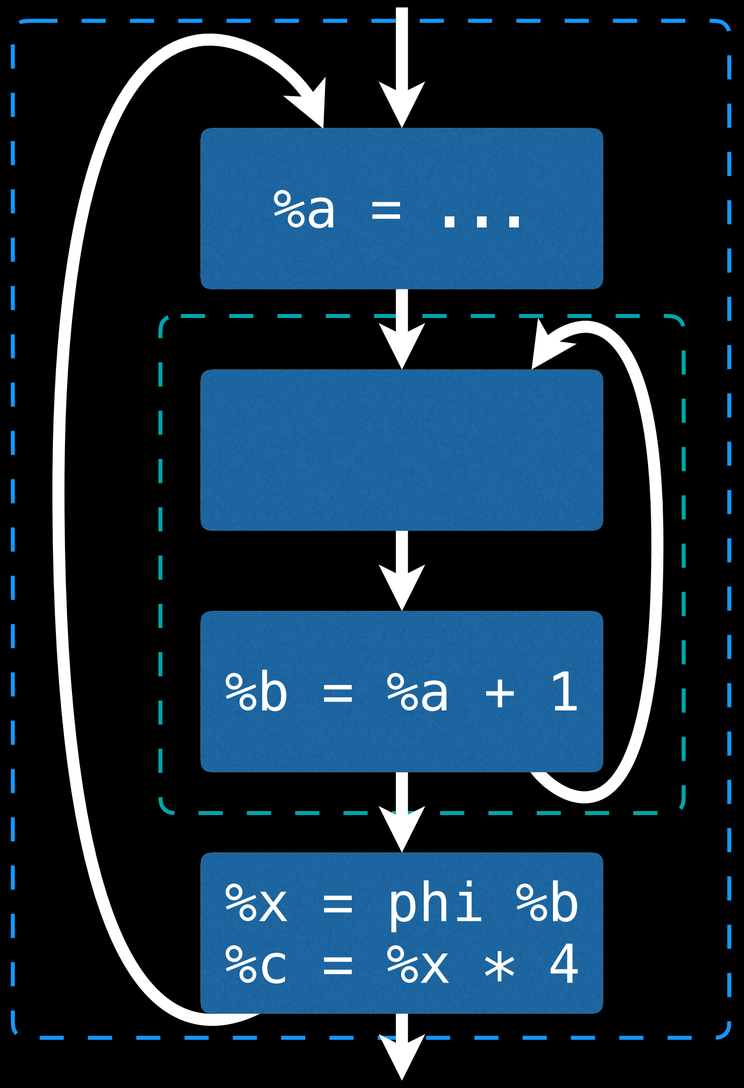
Updating LCSSA

```
do {
  a = …;

  do {


    b = a + 1;
  } while (inner);

  c = b * 4;
} while (outer);
```



Updating LCSSA

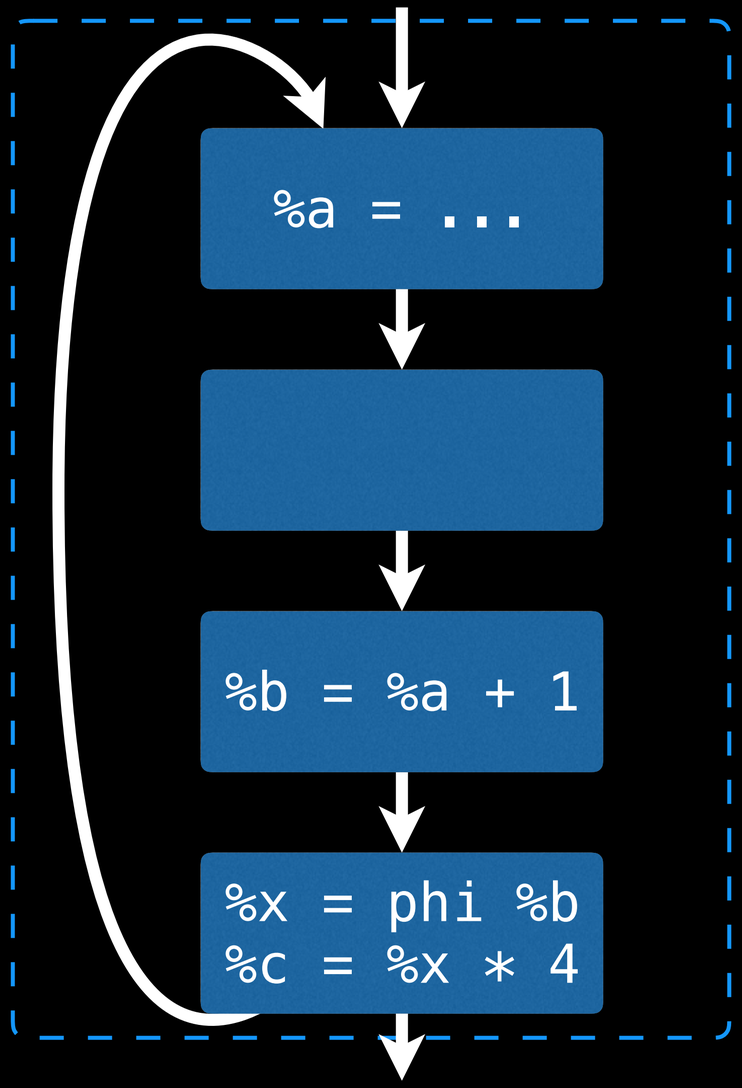
```
do {
  a = …;




  b = a + 1;



  c = b * 4;
} while (outer);
```



Updating LCSSA

```
while (outer) {
  a = …;


  while (inner) {

    b = a + 1;
    if (exit)
      return;
  }

}
```

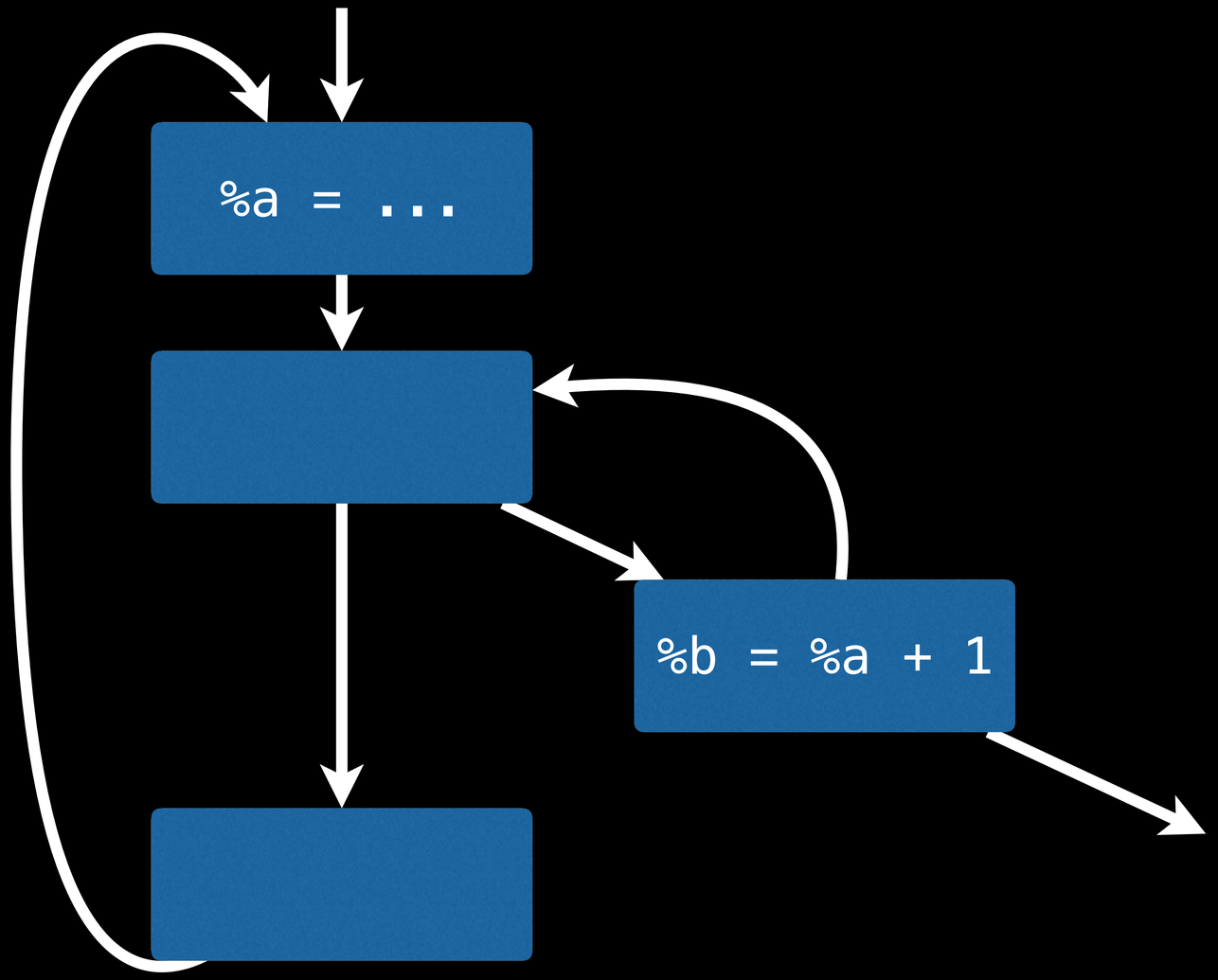
Updating LCSSA

```
while (outer) {
  a = …;


  while (inner) {

    b = a + 1;
    if (exit)
      return;
  }

}
```



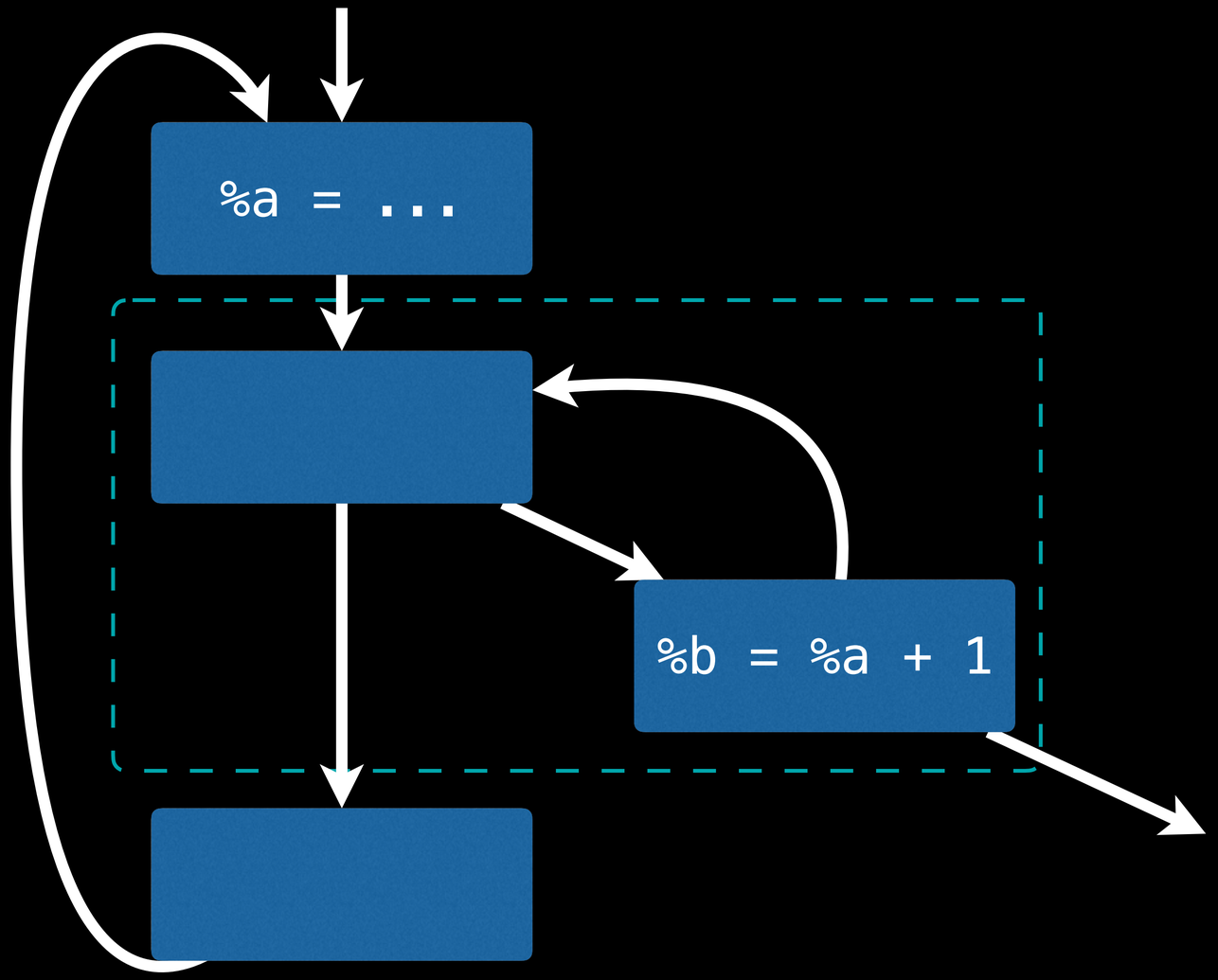
Updating LCSSA

```
while (outer) {
  a = …;

  while (inner) {

    b = a + 1;
    if (exit)
      return;
  }

}
```



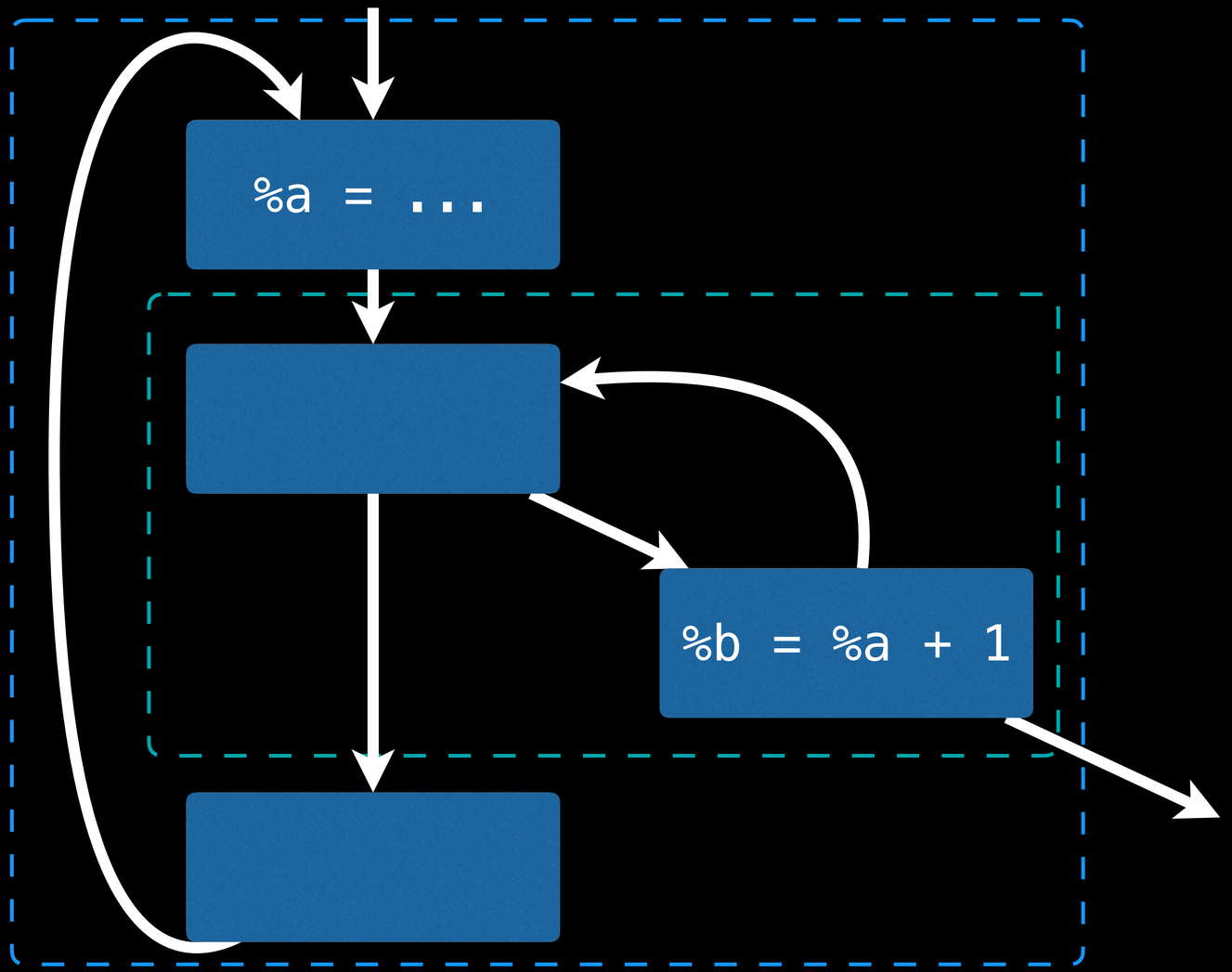
Updating LCSSA

```
while (outer) {
  a = …;

  while (inner) {

    b = a + 1;
    if (exit)
      return;
  }

}
```



Updating LCSSA

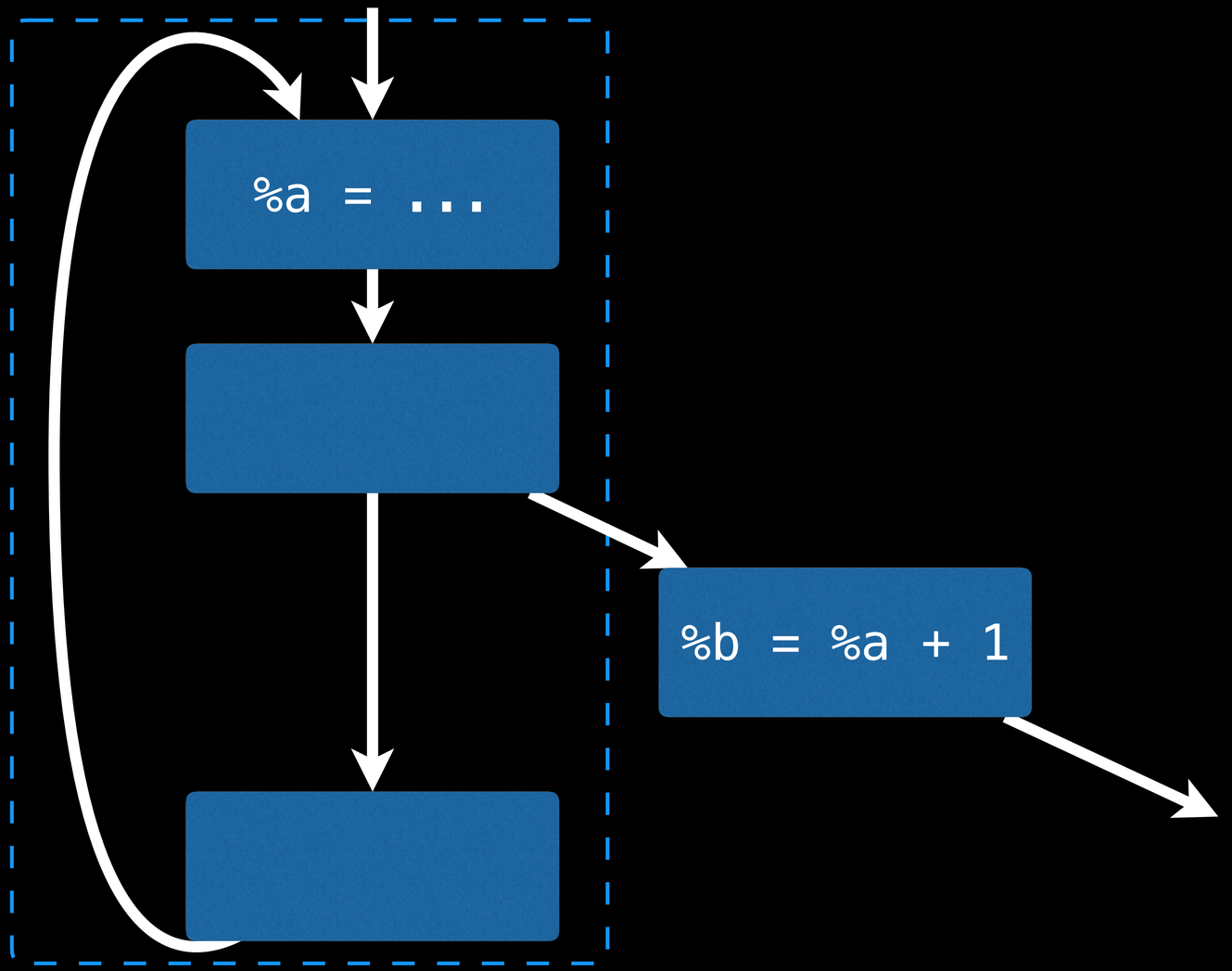
```
while (outer) {
  a = …;


  while (inner) {

    b = a + 1;

      return;
  }

}
```



Updating LCSSA

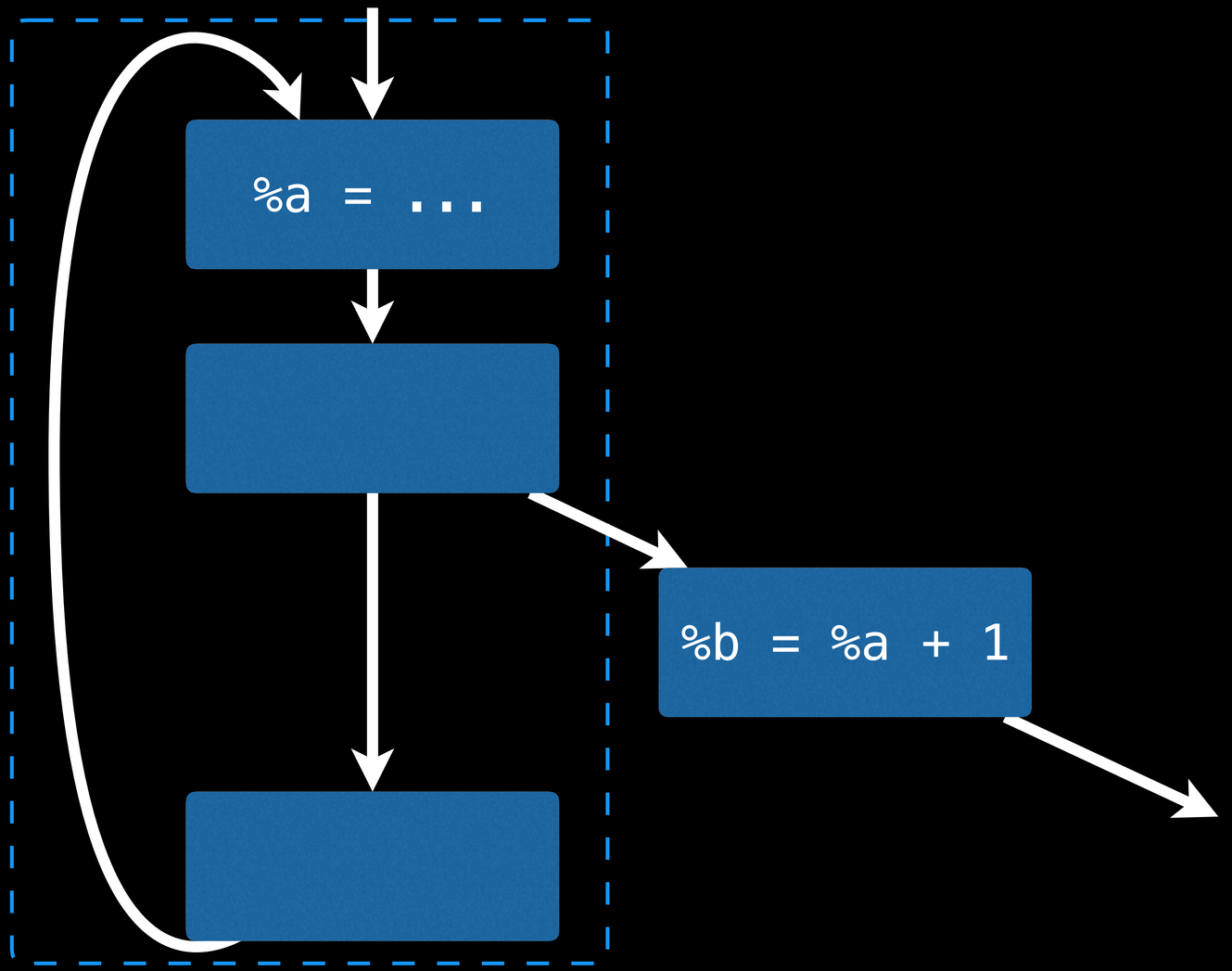
```
while (outer) {
  a = …;


  if (inner) {

    b = a + 1;

    return;
  }

}
```



Updating LCSSA

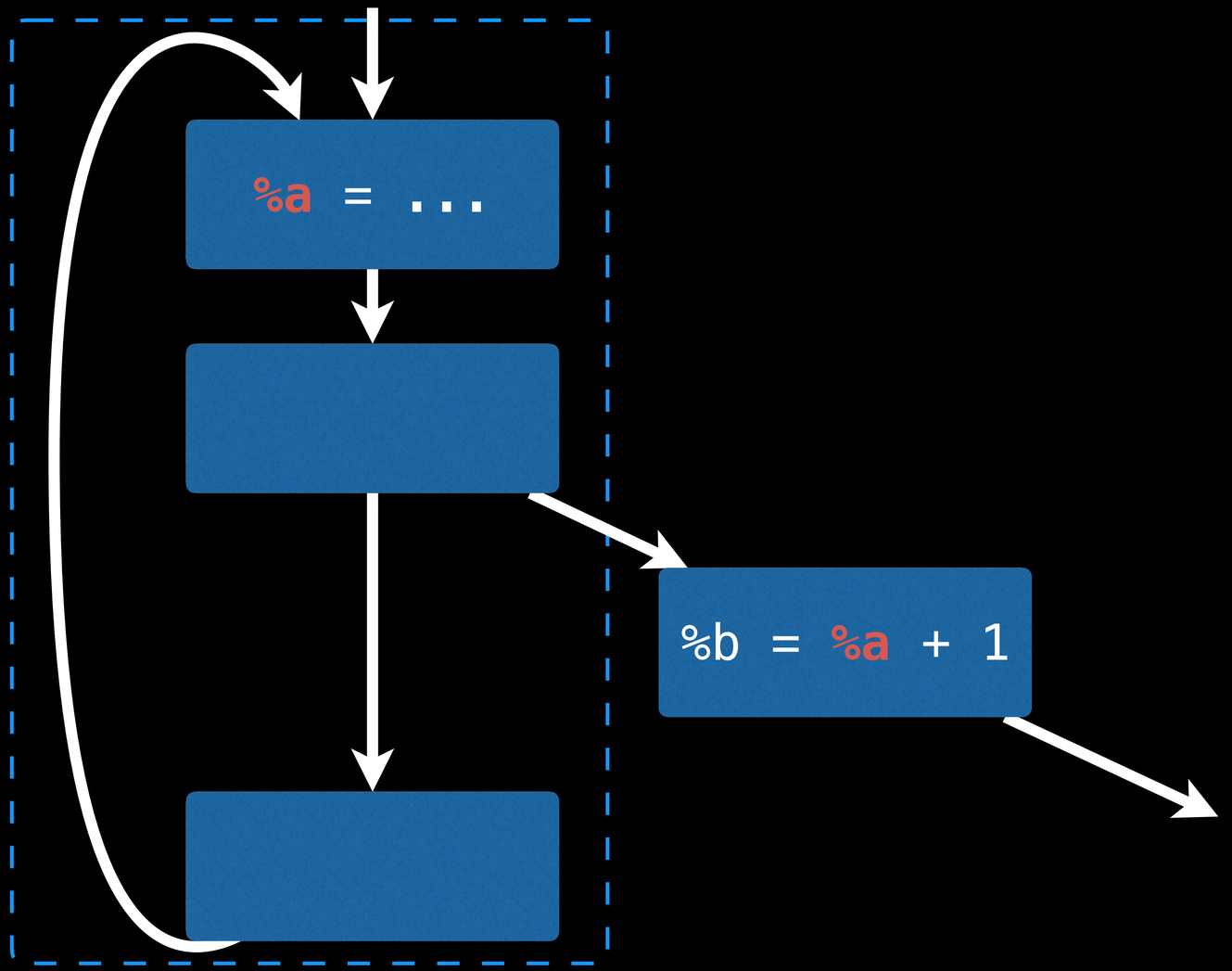
```
while (outer) {
  a = …;


  if (inner) {

    b = a + 1;

    return;
  }

}
```



Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  LoopSimplifyCFG(L);
  IndVars(L);
  Unroll(L);
}
```

```
Transform(L);
UpdateDT(L);
RebuildLCSSA(F);
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  LoopSimplifyCFG(L);
  IndVars(L);
  Unroll(L);
}
```

```
Transform(L);
UpdateDT(L);
if (ReallyReallyNeedToRebuild)
  RebuildLCSSA(F);
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  LoopSimplifyCFG(L);
  IndVars(L);
  Unroll(L);
}
```

```
Transform(L);
UpdateDT(L);
if (ReallyReallyNeedToRebuild)
  RebuildLCSSA(F);
VerifyLCSSA(F);
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  LoopSimplifyCFG(L);
  IndVars(L);
  Unroll(L);
}
```

```
Transform(L);
UpdateDT(L);
if (ReallyReallyNeedToRebuild)
  RebuildLCSSA(F);
VerifyLCSSA(F);          // FAIL!
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  LoopSimplifyCFG(L);
  IndVars(L);
  Unroll(L);
}
```

```
VerifyLCSSA(F);          // FAIL!
Transform(L);
UpdateDT(L);
if (ReallyReallyNeedToRebuild)
  RebuildLCSSA(F);
VerifyLCSSA(F);          // FAIL!
```

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  VerifyLCSSA(F);
  LoopSimplifyCFG(L);
  VerifyLCSSA(F);
  IndVars(L);
  VerifyLCSSA(F);
  Unroll(L);
  VerifyLCSSA(F);
}
```

Bugs Detected

Bug 25538 - clang crashes on valid code at -O2 on x86_64-linux-gnu

Bug 25578 - IndVarSimplify breaks LCSSA form, while saying it's preserved

Bug 26682 - crash on x86_64-linux-gnu at -O2 and above in both 32-bit and 64-bit modes (Assertion `L->isRecursivelyLCSSAForm(*DT) && "Indvars did not preserve LCSSA!"' failed)

Bug 26688 - Assert in LoopUnroll.cpp: Loops should be in LCSSA form after loop-unroll.

Bug 27157 - opt -O3 crashes with debug-only=loop-unroll

Bug 27945 - Compiler crash in "Induction Variable Simplification" for “-fno-exceptions"

Bug 28048 - crash at -Os, -O2 and -O3 in 32-bit and 64-bit mode on x86_64-linux-gnu (`L->isRecursivelyLCSSAForm(*DT) && "LCSSA required to run indvars!”')

Bug 28272 - LoopSimplify does not preserve LCSSA when separating nested loops.

Bug 28424 - Assertion `InLCSSA && "Requested to preserve LCSSA, but it's already broken."' failed.

...

Pass Structure

```
Prepare(F);
for (Loop *L : F) {
  Rotate(L);
  VerifyLoopAnalyses(F);
  LoopSimplifyCFG(L);
  VerifyLoopAnalyses(F);
  IndVars(L);
  VerifyLoopAnalyses(F);
  Unroll(L);
  VerifyLoopAnalyses(F);
}
```

Conclusion

- Use verifiers
- Test extensively
- Be aware of technical debt

Thank you!

Q&A