Global Value Numbering and Control Flow Optimizations

15-411/15-611 Compiler Design

Ben L. Titzer

April 2, 2026

About Me

- Associate Research Professor in S3D
- At CMU since 2022
- Google 2010-2019
  - Worked on V8 on 3 compilers: Crankshaft, TurboFan, and Liftoff
  - Co-founded WebAssembly
- Sun Labs
  - Worked on Maxine VM: C1X optimizing compiler
- Other
  - Virgil I-II compiler (2003-2007)
  - Virgil III compiler (2009-)
  - Wizard baseline compiler (2023-)

Today

- Extended basic blocks
- Local and Global Value Numbering
- Control flow optimizations
  - Jump threading
  - Tail duplication
- Instruction Scheduling

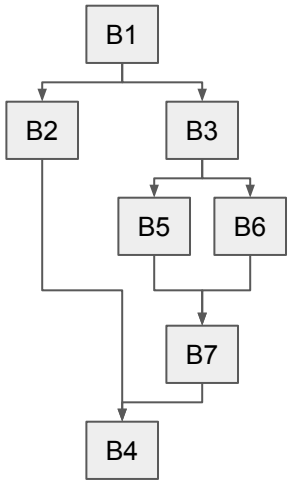
Extended Basic Blocks

- Traditional definition of basic block
  - Straight-line sequence with single entry and single exit (SESE)
  - Implies no internal control flow
- Extended basic block (EBB)
  - A *set* of blocks B_1, B_2 … B_n where only B1 has multiple predecessors and all other blocks have exactly one predecessor in the sequence

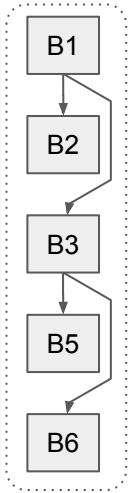
Extended Basic Blocks

- Extended basic block (EBB)
  - A *set* of blocks B_1, B_2 … B_n where only B1 has multiple predecessors and all other blocks have exactly one predecessor in the sequence

CFG



EBB



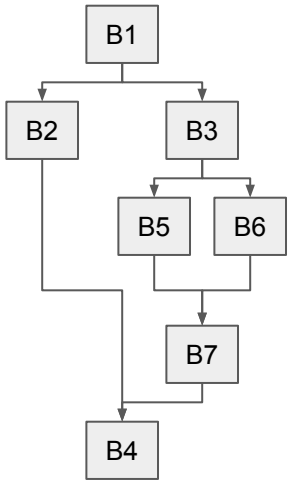
EBB

EBB

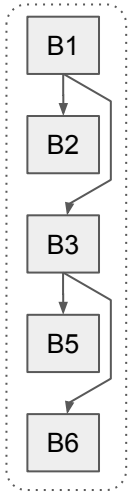
Extended Basic Blocks

- Extended basic block (EBB)
  - A *set* of blocks B_1, B_2 … B_n where only B1 has multiple predecessors and all other blocks have exactly one predecessor in the sequence

CFG



EBB



EBB

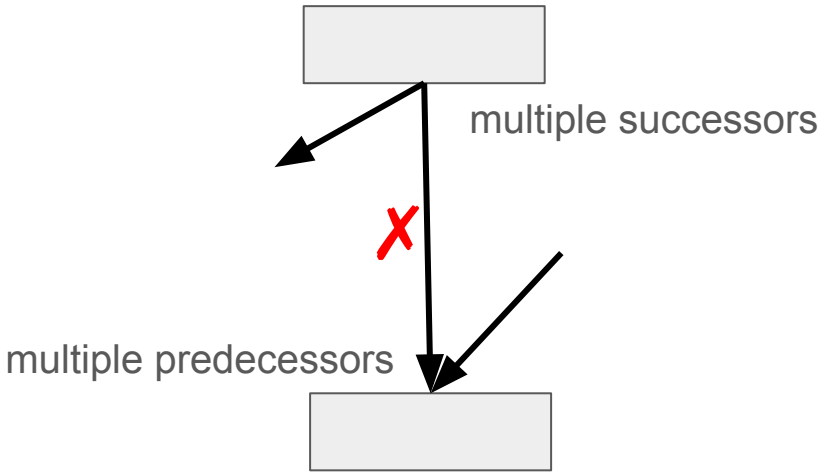
EBB

What about critical edges?

Splitting Critical Edges

- Recall from (deconstructing) SSA

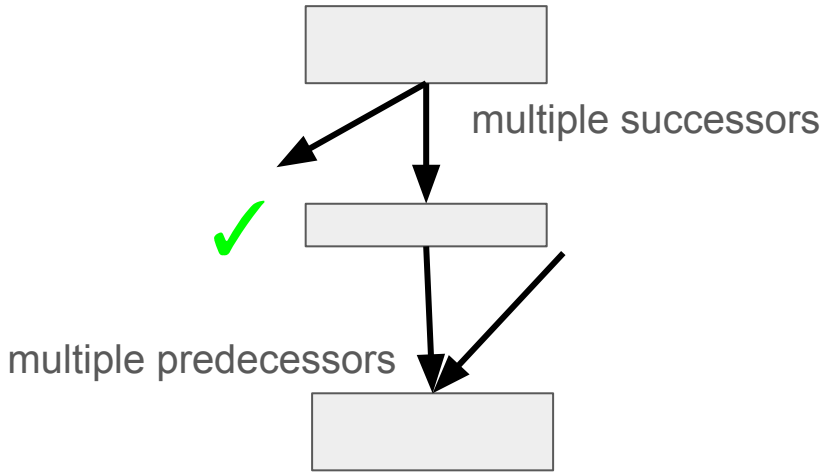
A *critical edge* is any edge that connects a block with multiple successors to a block with multiple predecessors.



Splitting Critical Edges

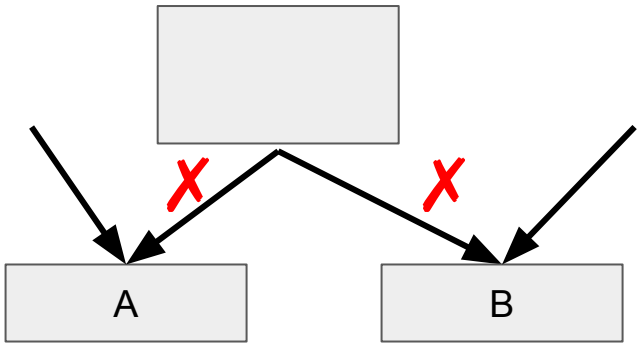
- Splitting critical edges is required for correct SSA deconstruction.
- Also benefits some optimizations like lazy code motion.

Splitting critical edges is an easy local transformation.



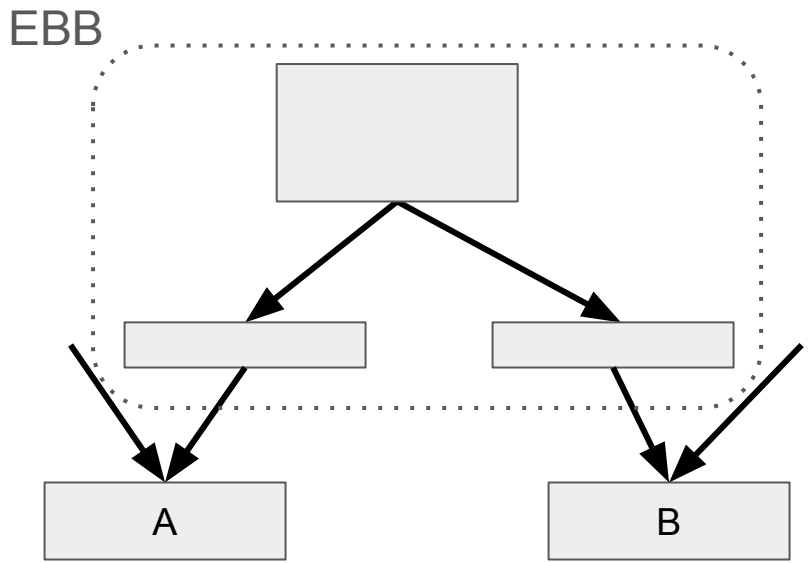
Splitting Critical Edges

Critical edges make conditional control flow seem unnecessarily general.



Splitting Critical Edges

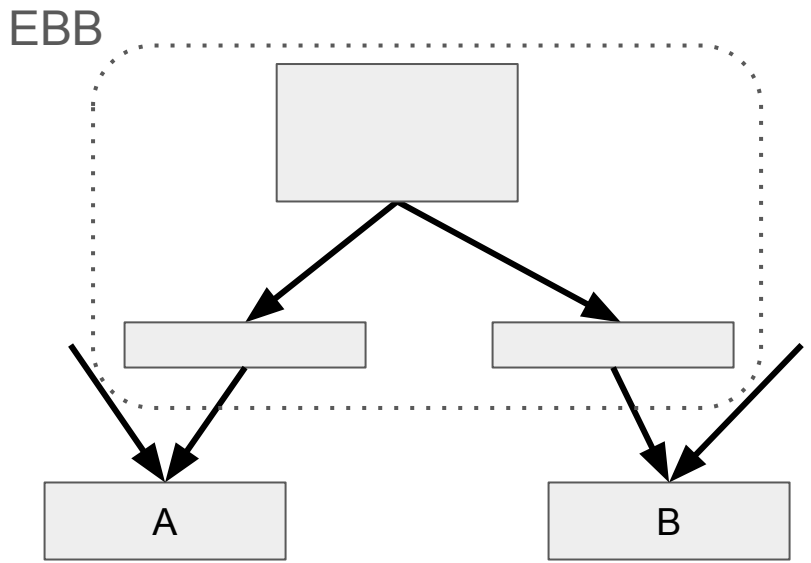
Splitting critical edges can make for larger EBBs.



Splitting Critical Edges

Splitting critical edges can make for larger EBBs.

Conditional branches will then always look like an `if`, with dominated successors with just one predecessor.

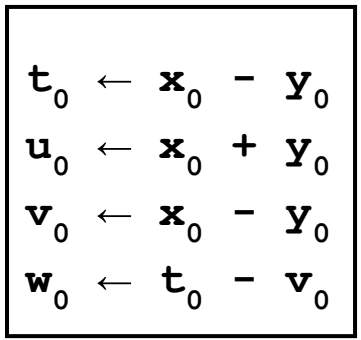


Using Extended Basic Blocks

- Extended basic blocks allow many local analyses to work on a larger scope
- Example: Local Value numbering to Global Value Numbering

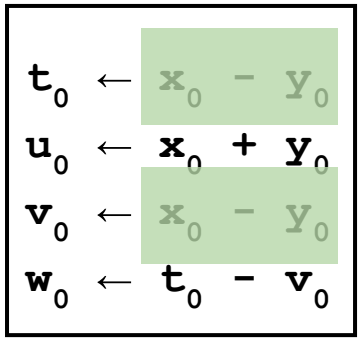
Common Subexpression Elimination

- Often, repeated computations appear due to optimization



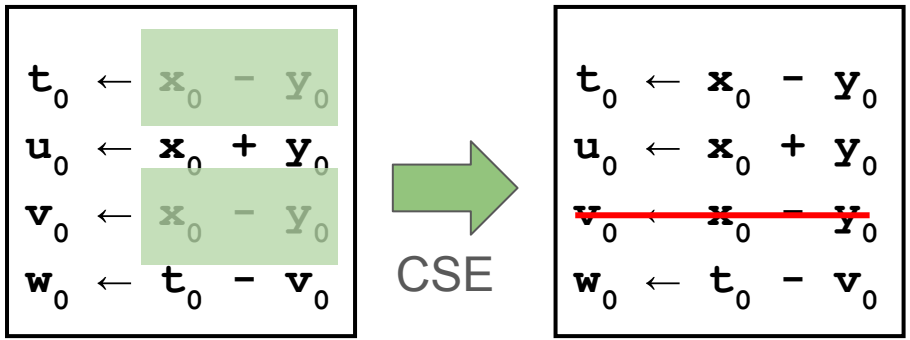
Common Subexpression Elimination

- Often, repeated computations appear due to optimization



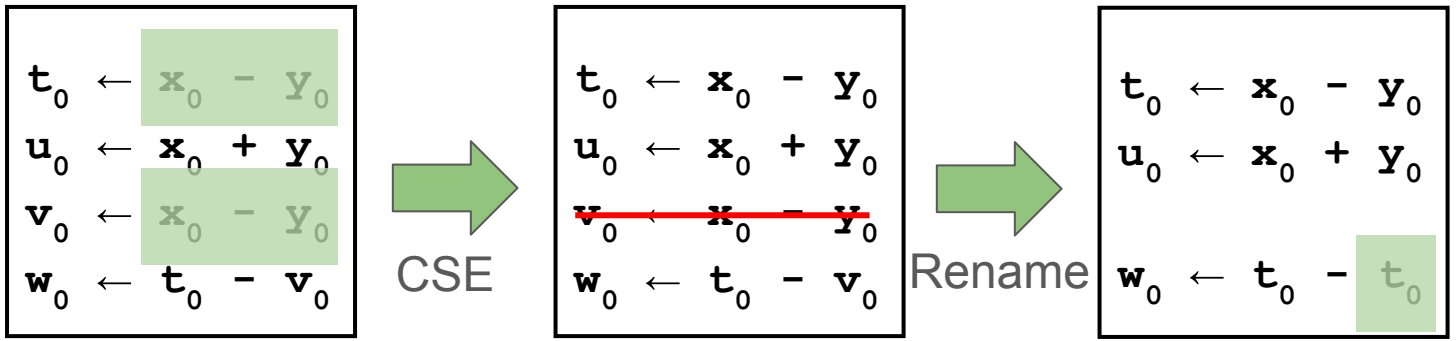
Common Subexpression Elimination

- Often, repeated computations appear due to optimization



Common Subexpression Elimination

- Often, repeated computations appear due to optimization



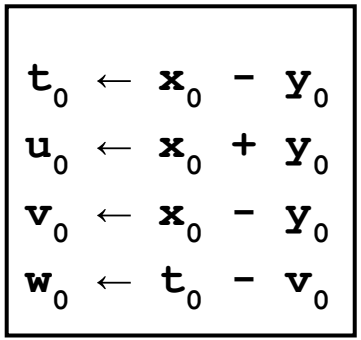
Common Subexpression Elimination

- A family of optimization techniques accomplish CSE

- Typical approaches
  - Local value numbering
  - Global value numbering
  - Lazy code motion
  - (loop invariant code motion)
  - (code hoisting)

Local Value Numbering

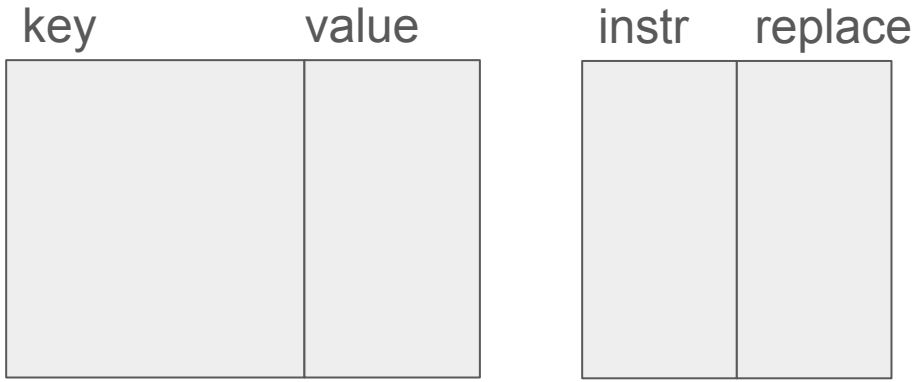
- Local: works on a basic block at a time
- Value *numbering*: going to use HashMaps (!?)
- Easy to implement
- Good results in practice
- Can be run multiple times (before/after elaboration)



- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

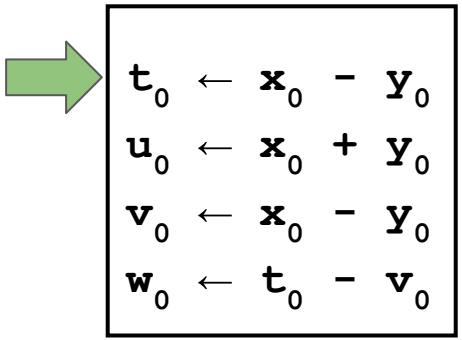
LVN map

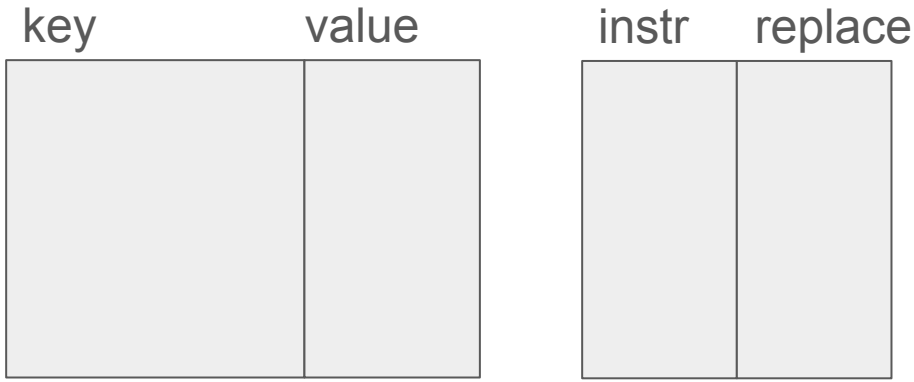
$t_0 \leftarrow x_0 - y_0$
$u_0 \leftarrow x_0 + y_0$
$v_0 \leftarrow x_0 - y_0$
$w_0 \leftarrow t_0 - v_0$



- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

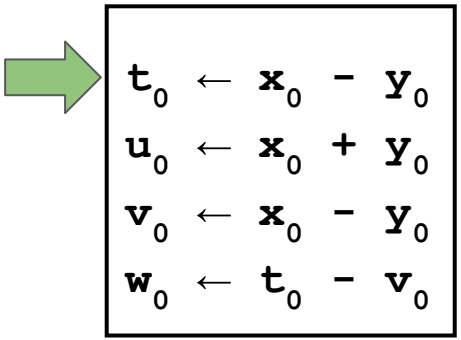
LVN map

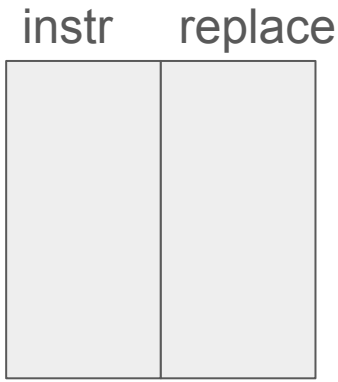




- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

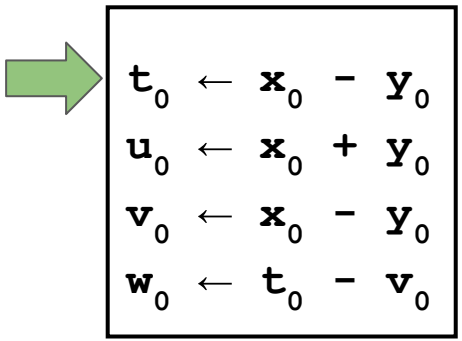
LVN map

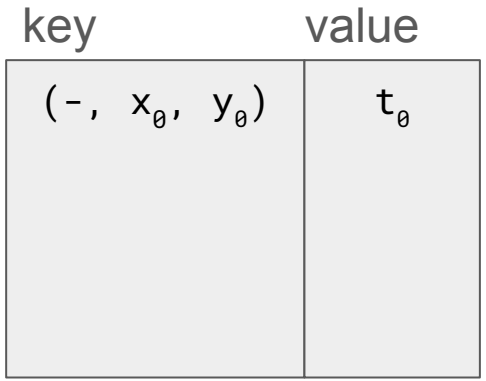


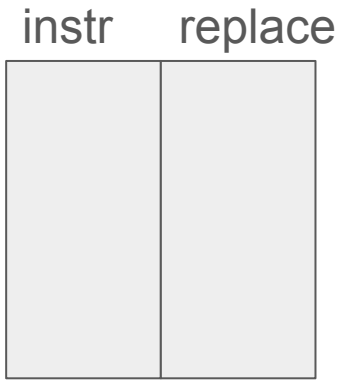


- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

LVN map

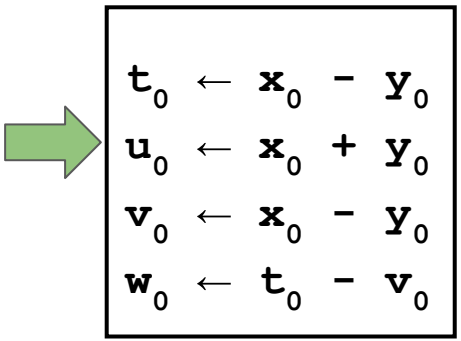


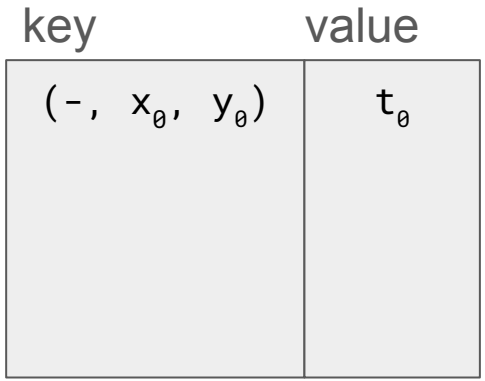


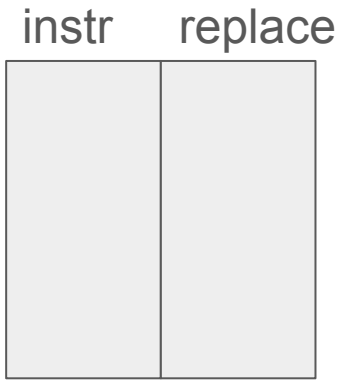


- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

LVN map

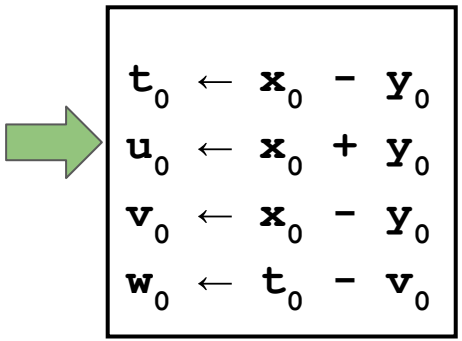


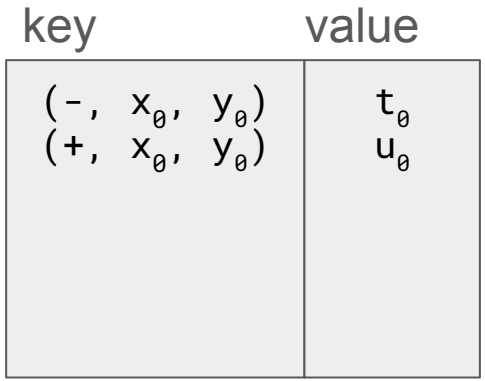


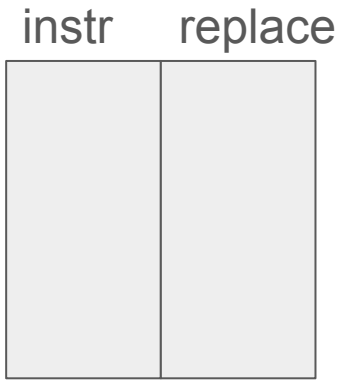


- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

LVN map

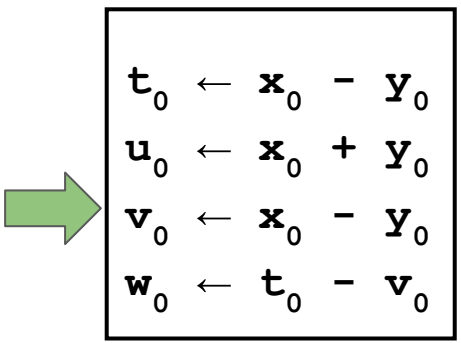


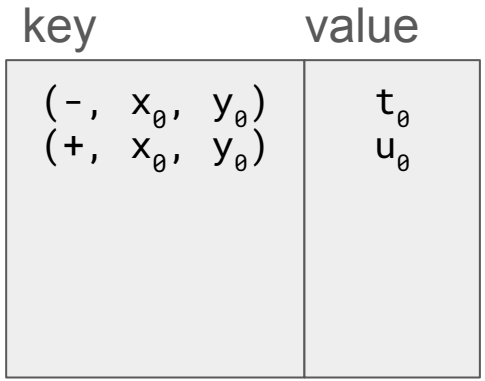


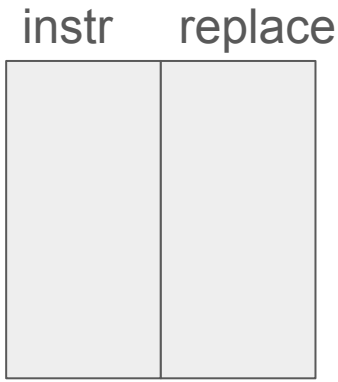


- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

LVN map

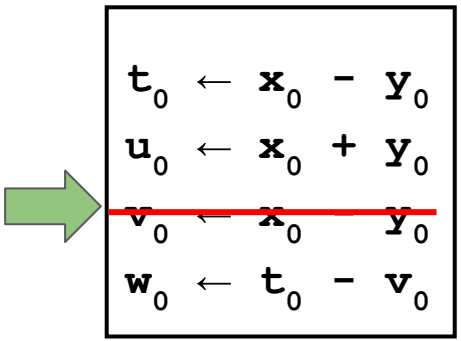






- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

LVN map

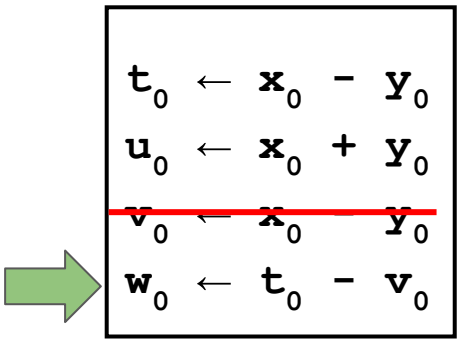


key	value
$(-, x_0, y_0)$	t_0
$(+, x_0, y_0)$	u_0

instr	replace
v_0	t_0

- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

LVN map

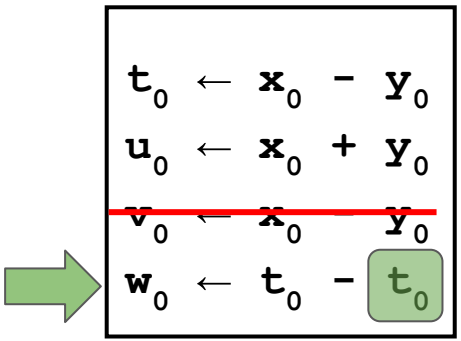


key	value
$(-, x_0, y_0)$	t_0
$(+, x_0, y_0)$	u_0

instr	replace
v_0	t_0

- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

LVN map

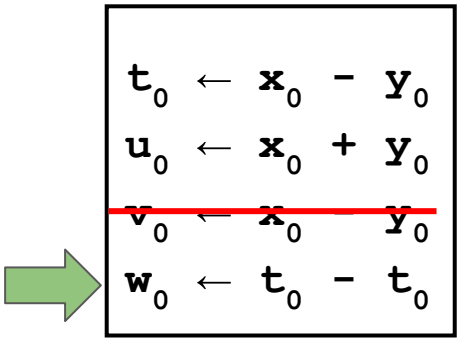


key	value
$(-, x_0, y_0)$	t_0
$(+, x_0, y_0)$	u_0

instr	replace
v_0	t_0

- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

LVN map



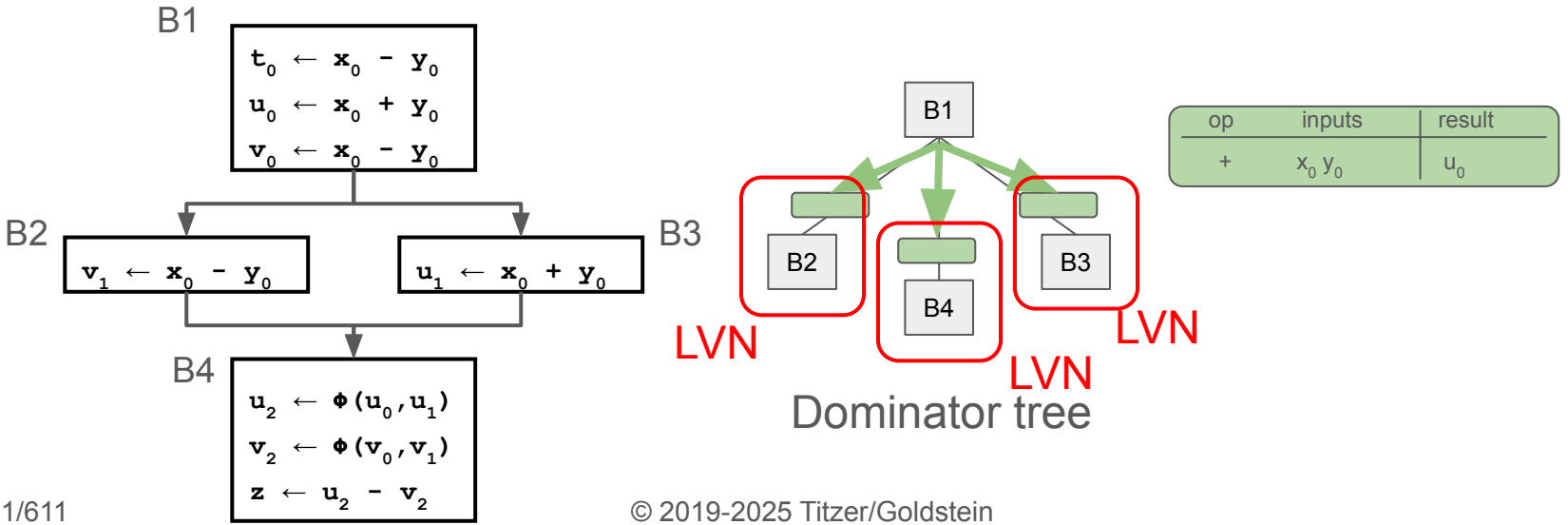
key	value
$(-, x_0, y_0)$	t_0
$(+, x_0, y_0)$	u_0
$(-, t_0, t_0)$	w_0

instr	replace
v_0	t_0

- Forward pass over instructions in a basic block
- Maintain:
  - map from (op, inputs) to instruction
  - map from instruction to instruction
- Rename RHS inputs
- Lookup RHS
  - If found, replace with result
  - If not found and *pure*, add to map (op, inputs)

Global Value Numbering using Dominators

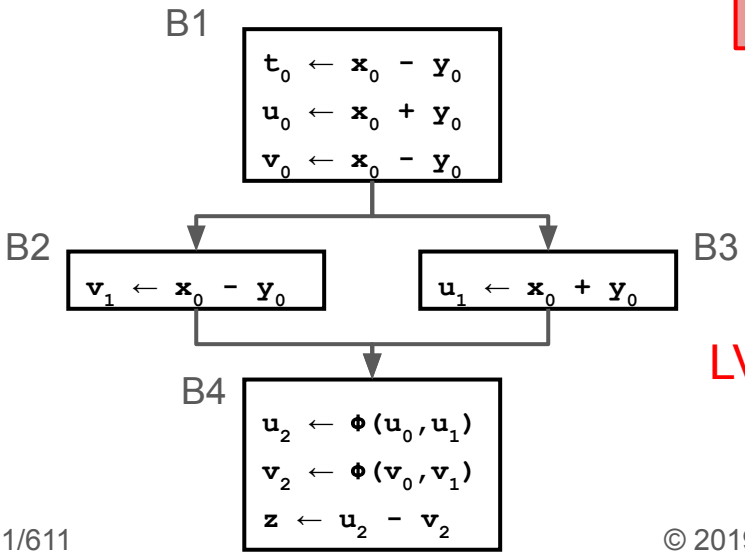
- LVN finds redundant computations in a basic block
- Dominator-based GVN algorithm generalizes LVN
- Propagate local value numbering map from dominator to dominated blocks

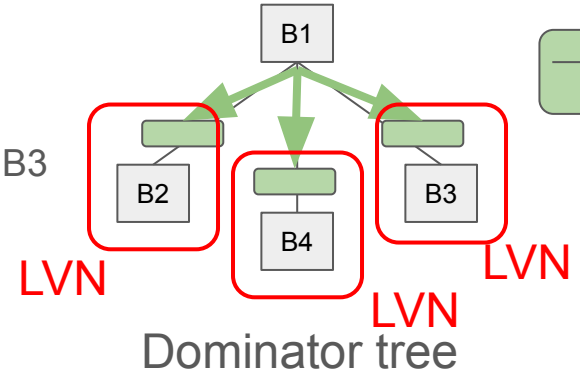


Global Value Numbering using Dominators

- LVN finds redundant computations in a basic block
- Dominator-based GVN algorithm generalizes LVN
- Propagate local value numbering map from dominator to dominated blocks

What if we don't have a dominator tree (yet)?

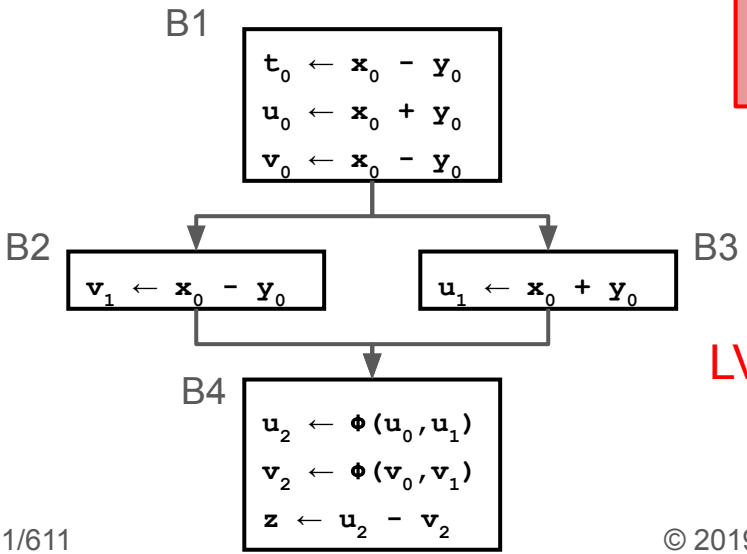




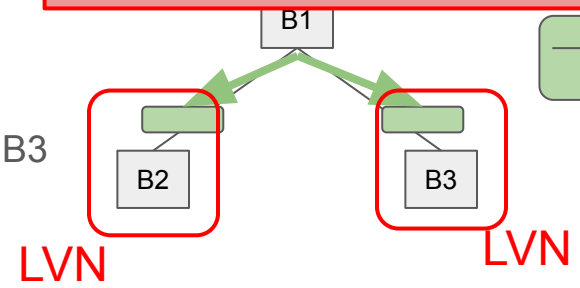
op	inputs	result
+	x_0 y_0	u_0

Global Value Numbering using Dominators

- LVN finds redundant computations in a basic block
- EBB-based GVN algorithm trades accuracy for speed
- Propagate local value numbering map along EBB edges



EBB-GVN is less accurate but cheaper than dominator-based GVN.



op	inputs	result
+	x_0 y_0	u_0

Extended Basic Block

Control Flow Optimizations: Tail duplication

- Often, merges in the control flow complicate analyses
- Dataflow equations with unions
- Missed value numbering opportunities
- Merges are not in predecessors' EBBs
- Merges often have φs, harder to reason through

Control Flow Optimizations: Tail duplication

- Often, merges in the control flow complicate analyses
- Dataflow equations with unions
- Missed value numbering opportunities
- Merges are not in predecessors' EBBs
- Merges often have φs, harder to reason through

Use DF solver

Control Flow Optimizations: Tail duplication

- Often, merges in the control flow complicate analyses
- Dataflow equations with unions
- Missed value numbering opportunities
- Merges are not in predecessors' EBBs
- Merges often have φs, harder to reason through

Use DF solver

Use DF solver

Control Flow Optimizations: Tail duplication

- Often, merges in the control flow complicate analyses
- Dataflow equations with unions
- Missed value numbering opportunities
- Merges are not in predecessors' EBBs
- Merges often have φs, harder to reason through

Use DF solver

Use DF solver

Use DF solver

Control Flow Optimizations: Tail duplication

- Often, merges in the control flow complicate analyses
- Dataflow equations with unions
- Missed value numbering opportunities
- Merges are not in predecessors' EBBs
- Merges often have φs, harder to reason through

Use DF solver

Use DF solver

Use DF solver

Use DF solver

Control Flow Optimizations: Tail duplication

- Often, merges in the control flow complicate analyses
- Dataflow equations with unions
- Missed value numbering opportunities
- Merges are not in predecessors' EBBs
- Merges often have φs, harder to reason through

Use DF solver

Use DF solver

Use DF solver

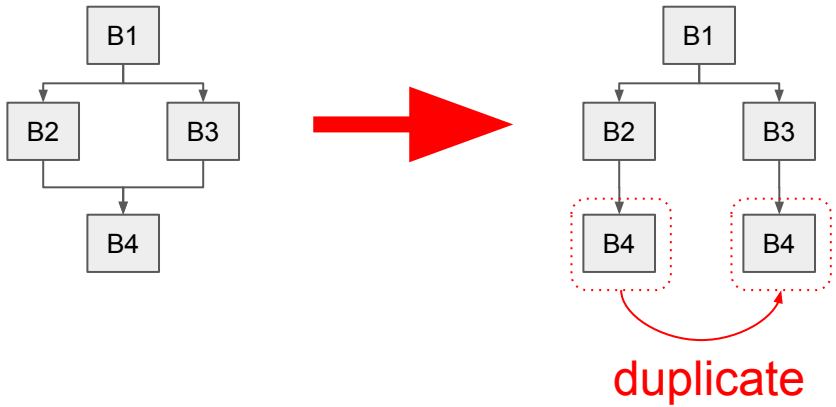
Use DF solver

What about problems that are not necessarily dataflow problems?

Tail Duplication

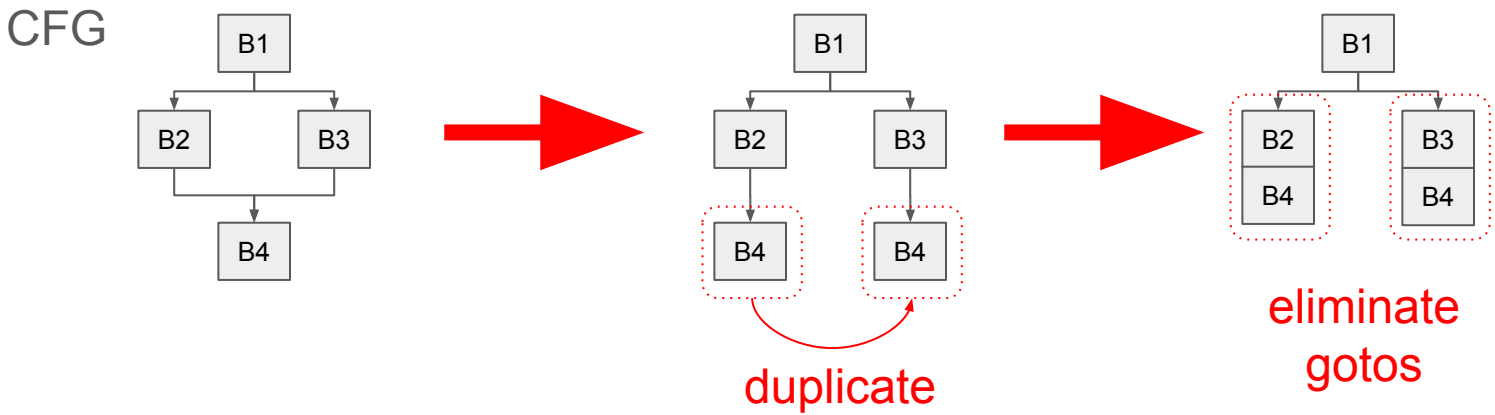
- One solution: duplicate code at joins
  - Eliminate the source of imprecision in dataflow analysis
  - Generates new specialization opportunities
  - Eliminates jumps

CFG

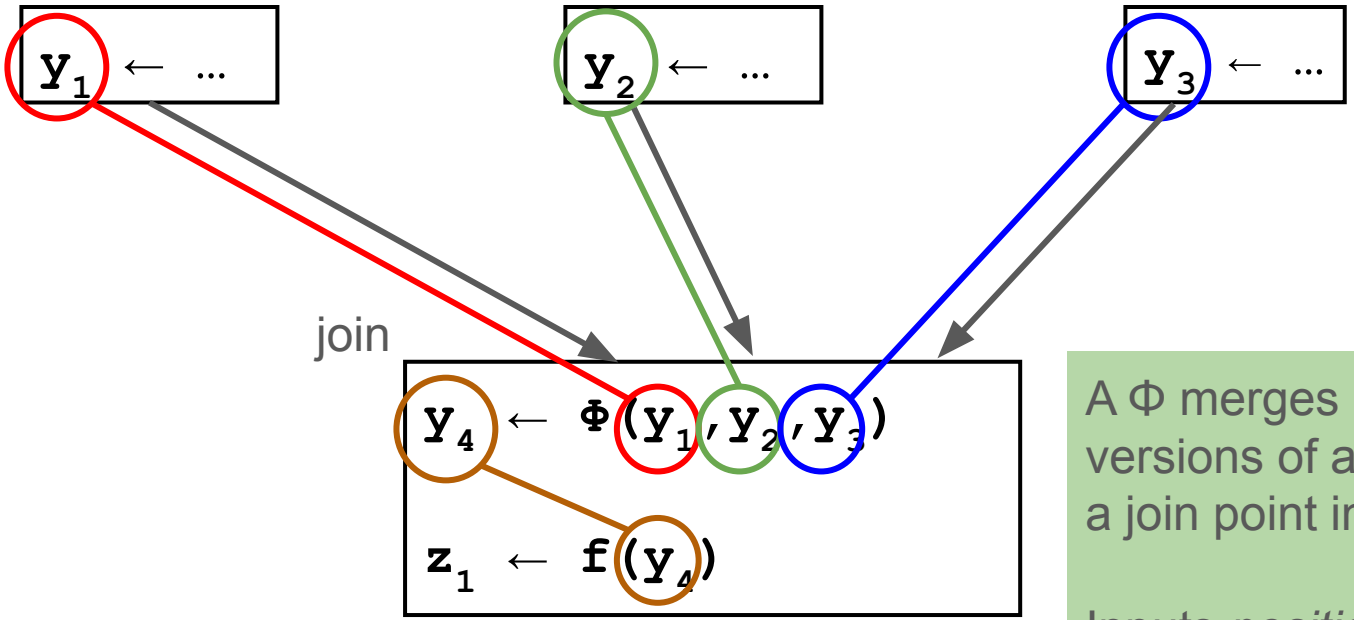


Tail Duplication

- One solution: duplicate code at joins
  - Eliminate the source of imprecision in dataflow analysis
  - Generates new specialization opportunities
  - Eliminates jumps



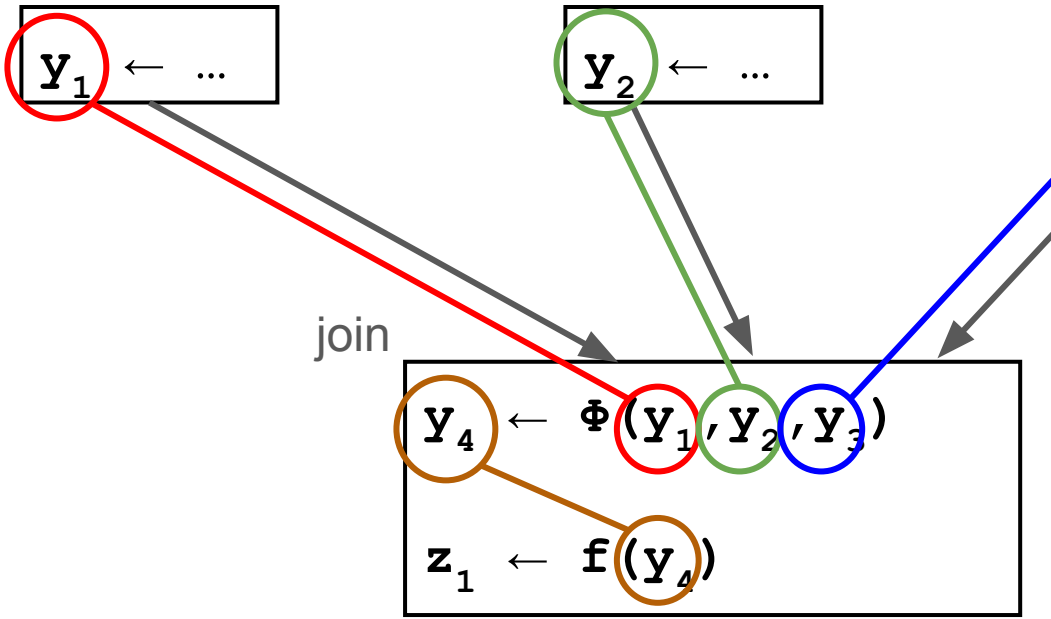
Tail Duplication Transformation



A Φ merges multiple versions of a variable at a join point in the CFG.

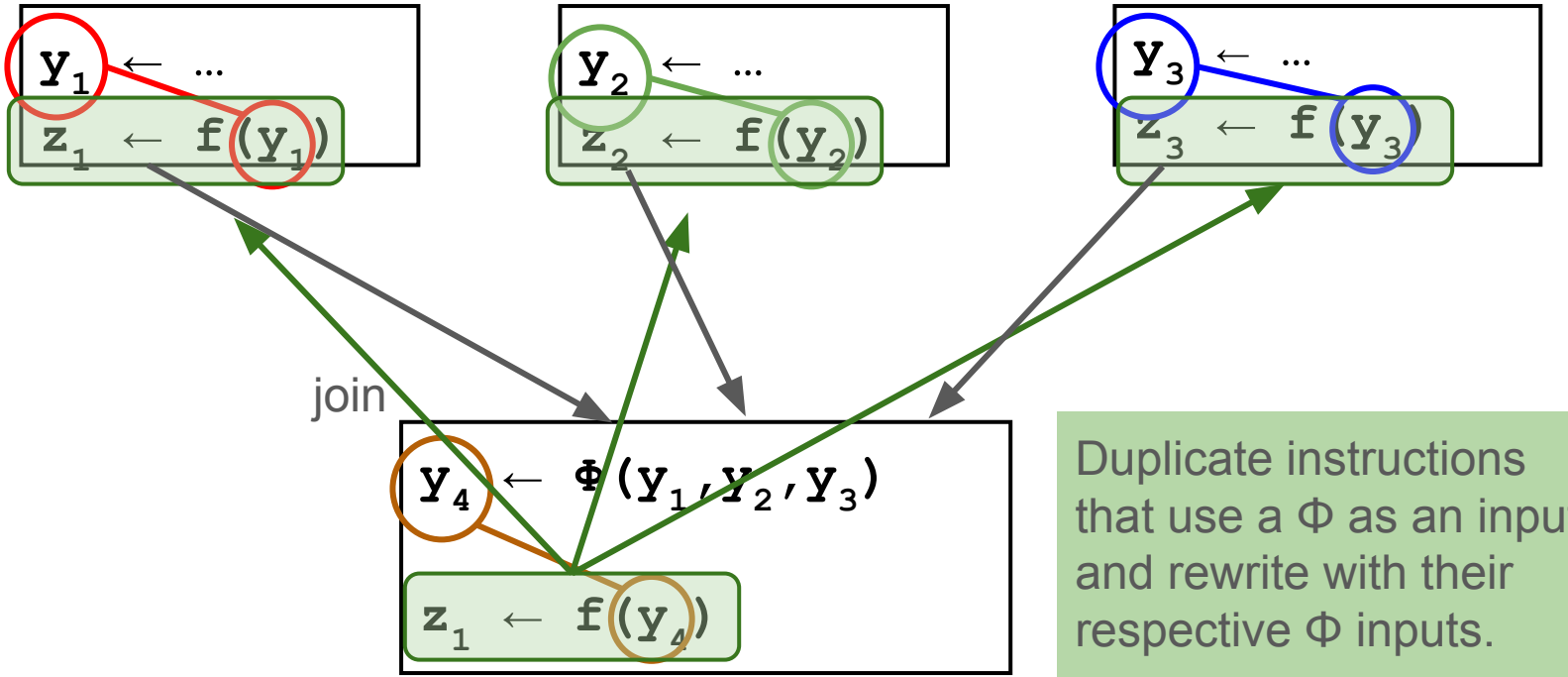
Inputs *positionally correspond* with predecessor edges.

Tail Duplication Transformation



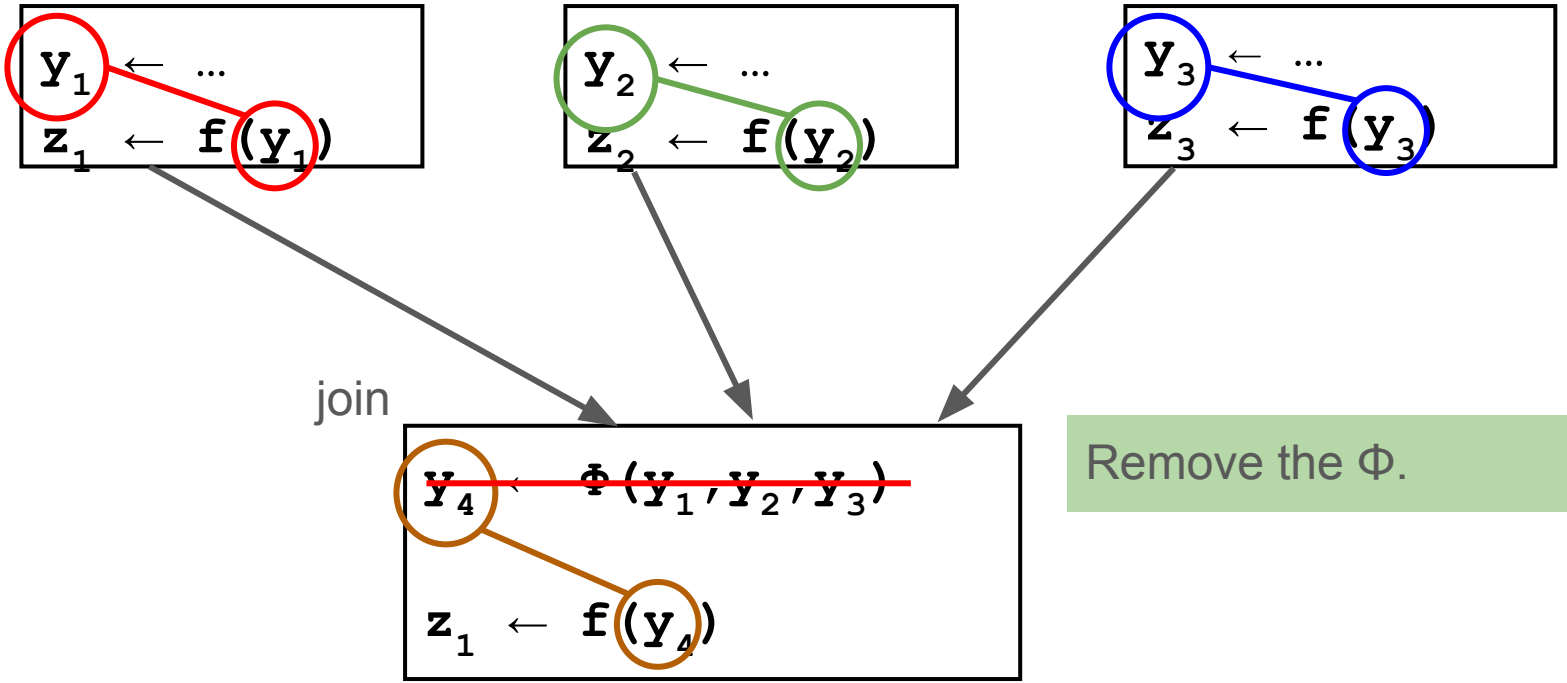
Instructions that have a Φ as an input at the join point might benefit tail duplication.

Tail Duplication Transformation



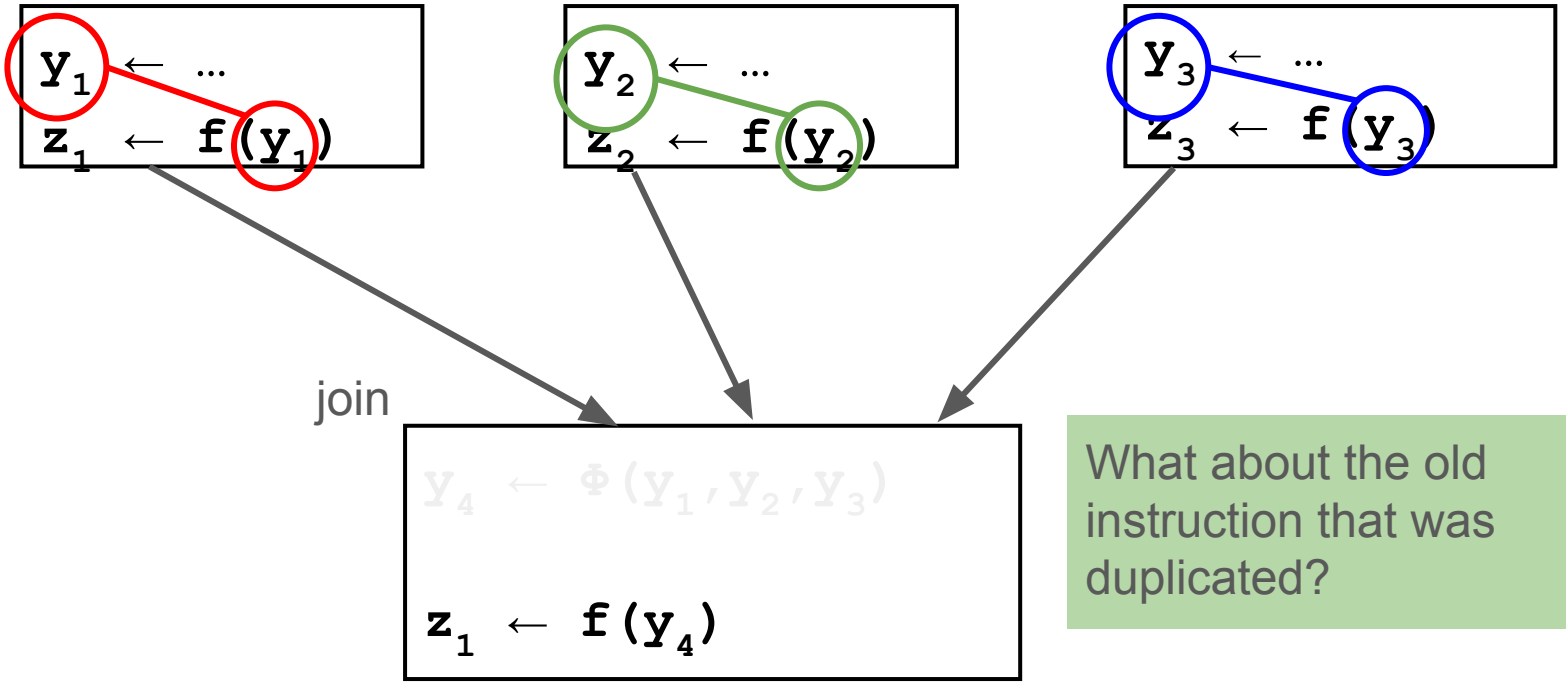
Duplicate instructions that use a Φ as an input and rewrite with their respective Φ inputs.

Tail Duplication Transformation



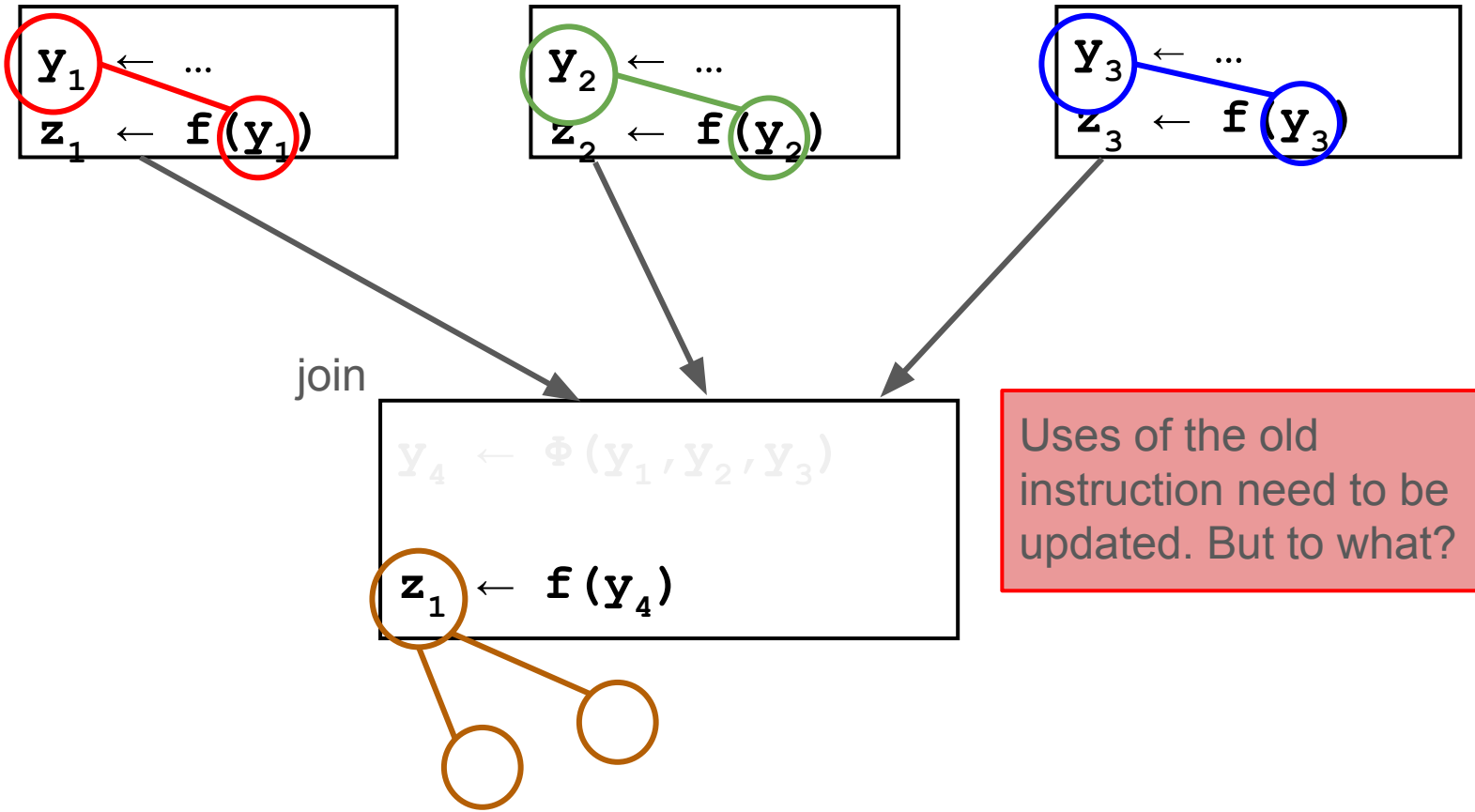
Remove the Φ.

Tail Duplication Transformation



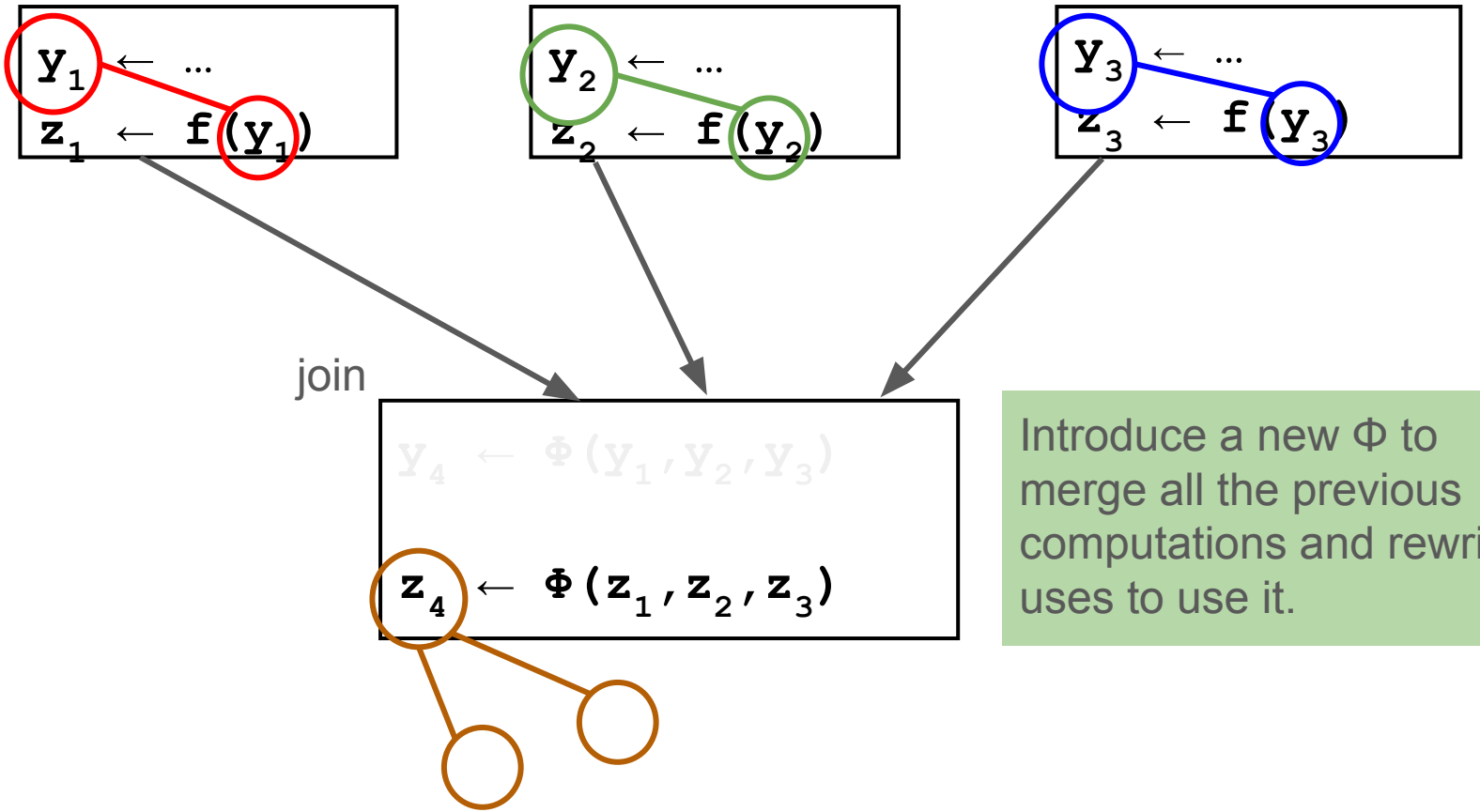
What about the old instruction that was duplicated?

Tail Duplication Transformation



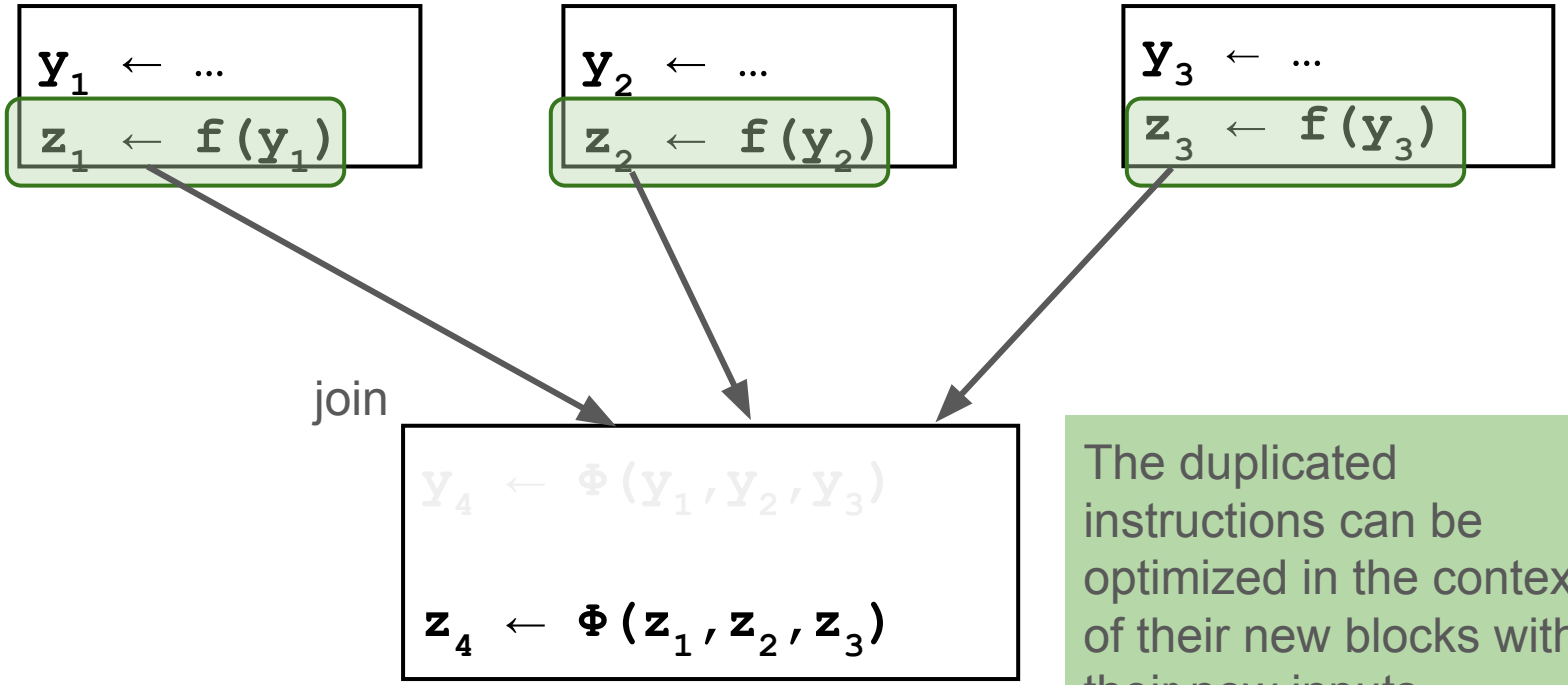
Uses of the old instruction need to be updated. But to what?

Tail Duplication Transformation



Introduce a new Φ to merge all the previous computations and rewrite uses to use it.

Tail Duplication Transformation



The duplicated instructions can be optimized in the context of their new blocks with their new inputs.

Tail Duplication Summary

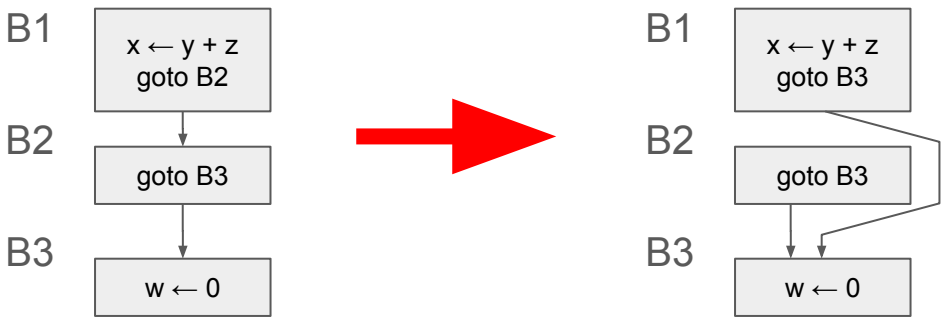
- Tail duplication copies join blocks to allow more optimization from context of predecessors.
- With SSA and edge split form, it's possible to do this without introducing new control flow.
- Like any optimization that copies code, it is a tradeoff
  - Increased code size, I-cache pressure
  - Specialization opportunities
  - More precise analyses

Jump Threading

- Often programs have “jumps to jumps”
  - Can show up late, after other transformations
  - Critical edge splitting can often produce empty blocks, but don’t know this until after SSA deconstruction or lazy code motion is done
- Sometimes a “jump to a branch” where the branch outcome will be known
  - Arises in compiling complex conditional expressions (&& and ||)
  - Can occur for other kinds of complex conditions
- Two approaches
  - “Thread” branches
  - Duplicate target block

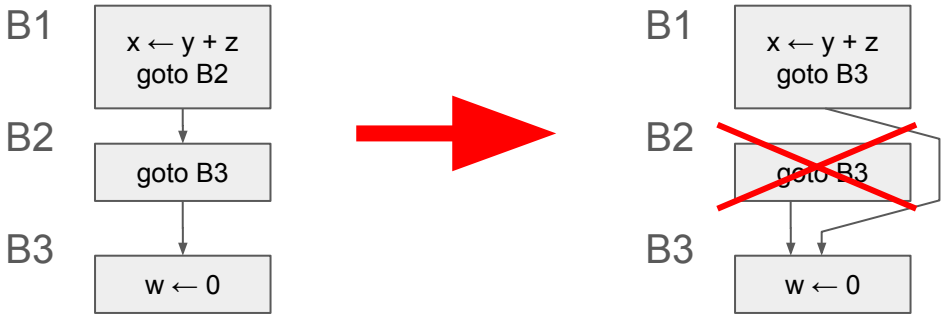
Jump Threading

- Form 1: direct jump to empty block with jump
  - Rewrite first jump with second jump's destination



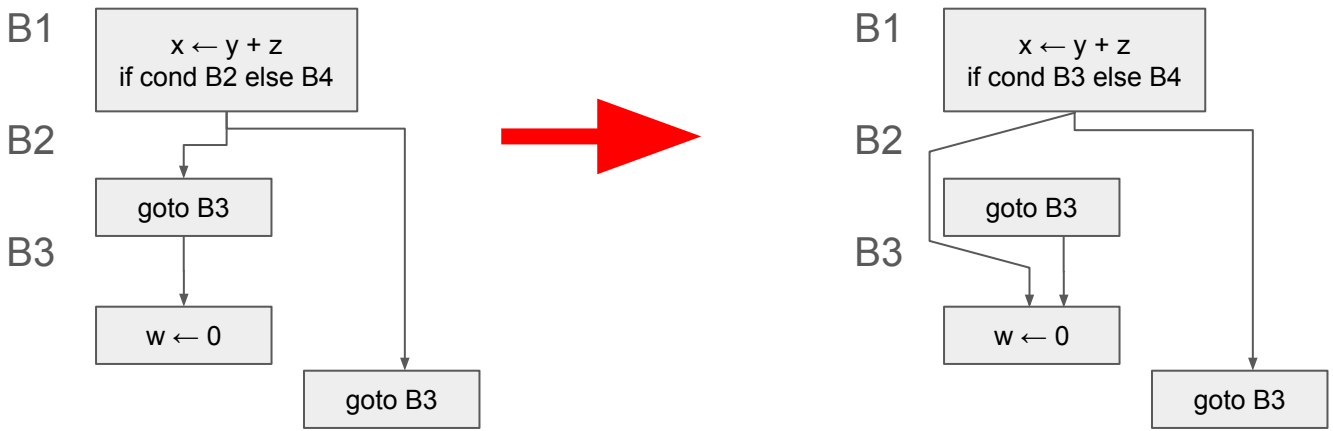
Jump Threading

- Form 1: direct jump to empty block with jump
  - Rewrite first jump with second jump's destination
  - Delete empty block if no remaining predecessors



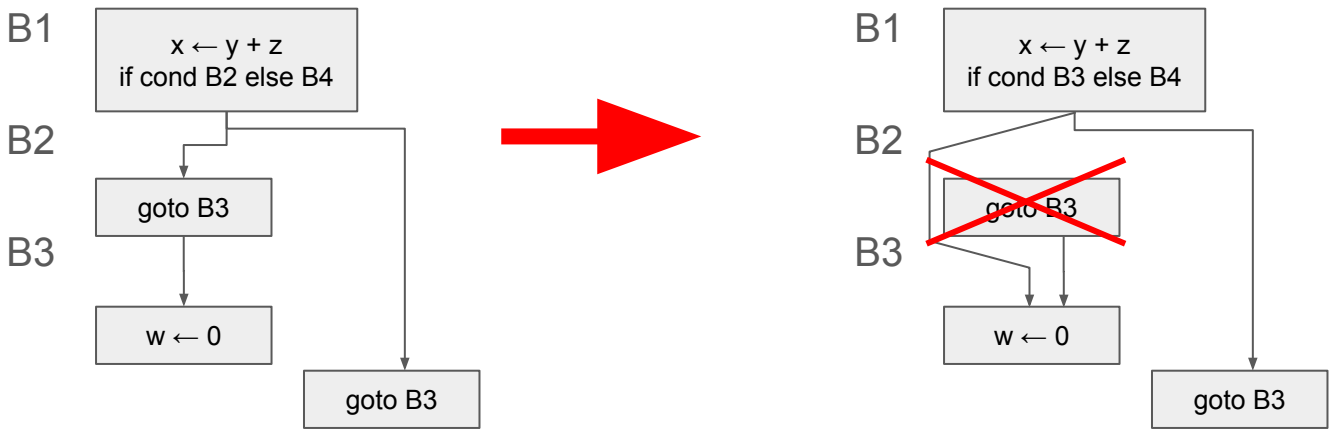
Jump Threading

- Form 2: conditional branch to empty block with jump
  - Rewrite first conditional branch with second jump's destination



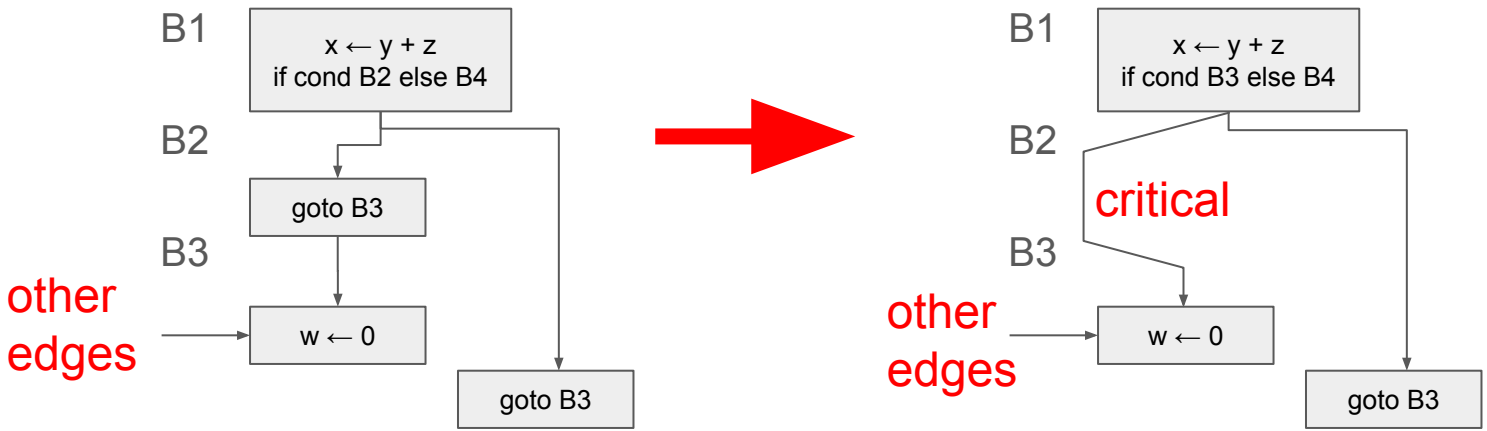
Jump Threading

- Form 2: conditional branch to empty block with jump
  - Rewrite first conditional branch with second jump's destination



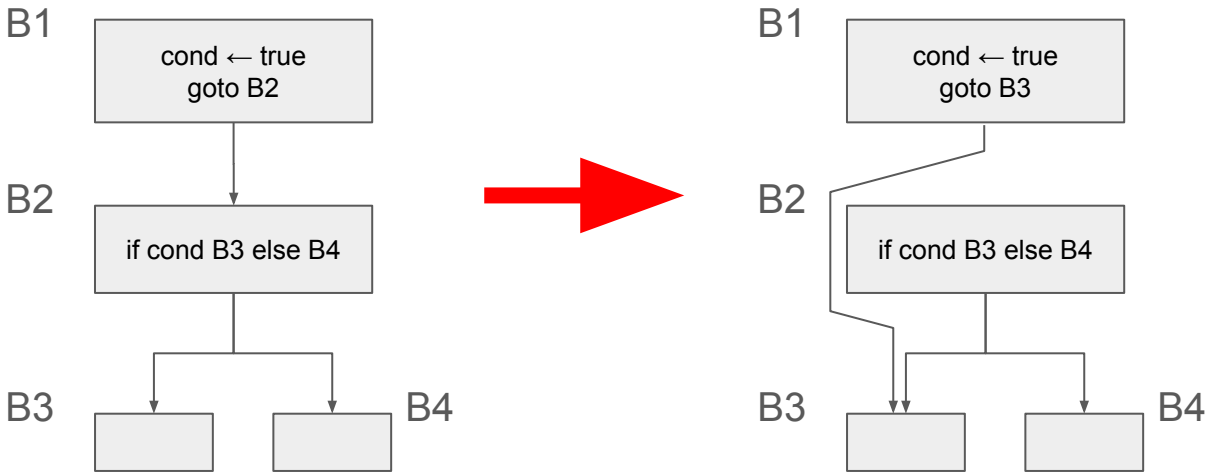
Jump Threading

- Form 2: conditional branch to empty block with jump
  - Rewrite first conditional branch with second jump's destination
  - Avoid introducing critical edges if necessary



Jump Threading

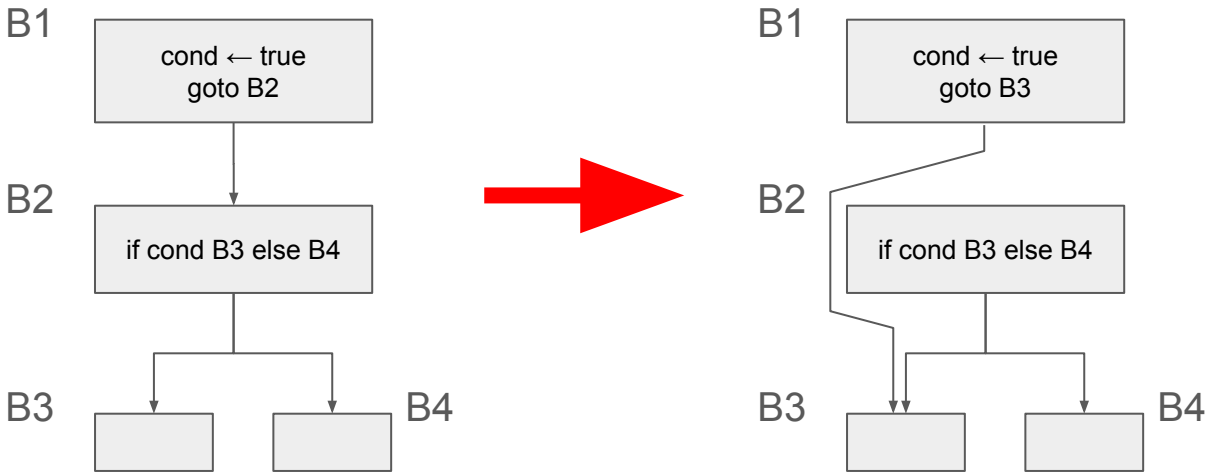
- Form 3: direct jump to block with conditional branch that will have a known outcome



Jump Threading

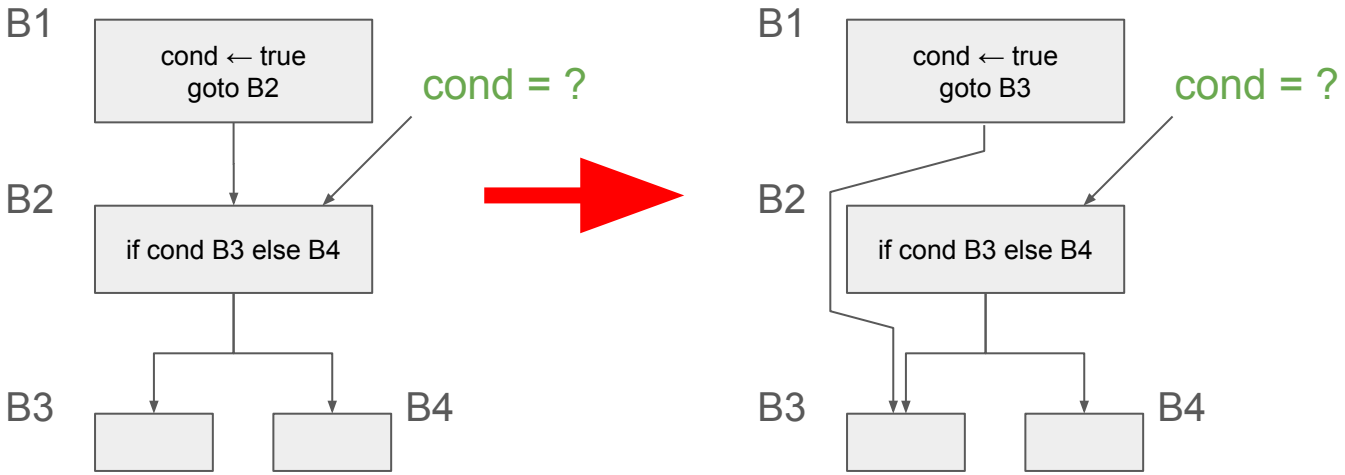
- Form 3: direct jump to block with conditional branch that will have a known outcome

Why wouldn't block merging and branch folding optimize this?



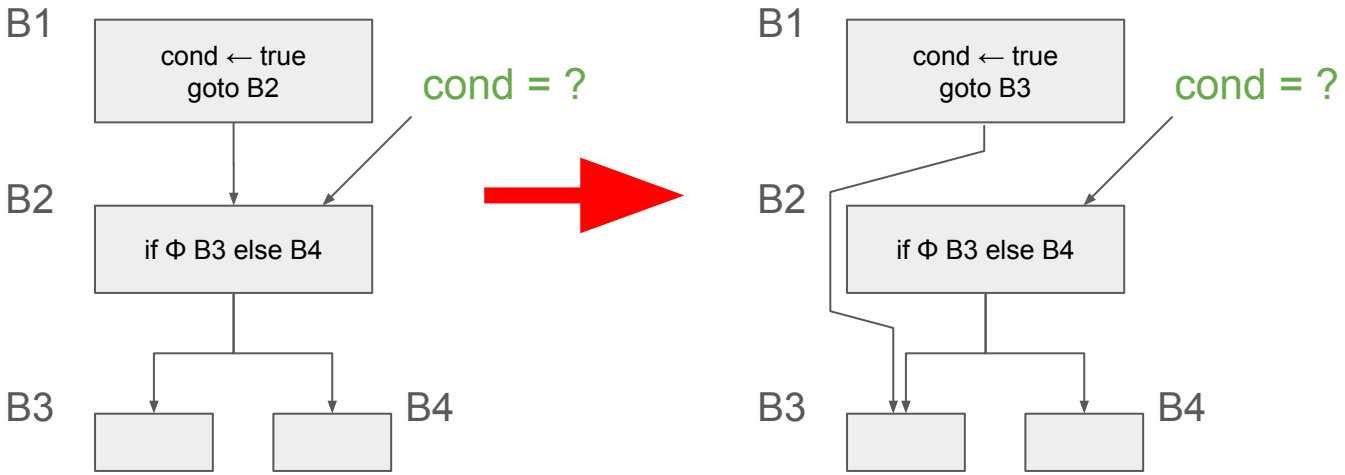
Jump Threading

- Form 3: direct jump to block with conditional branch that will have a known outcome
  - Target block may other predecessors; outcome only known on this path



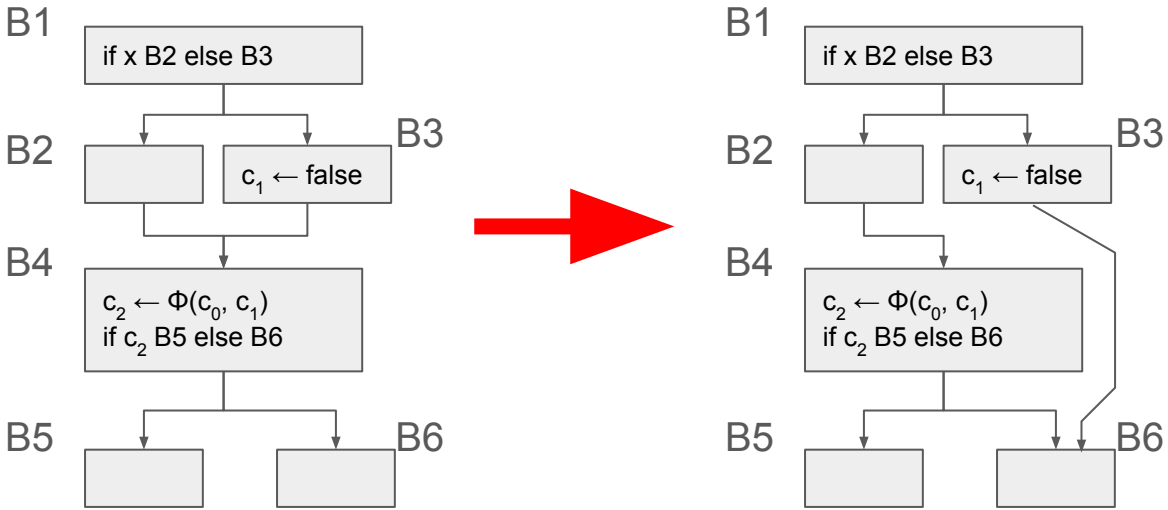
Jump Threading

- Form 3: direct jump to block with conditional branch that will have a known outcome
  - Target block may other predecessors; outcome only known on this path
  - In SSA, target block cond will thus be a **Φ**



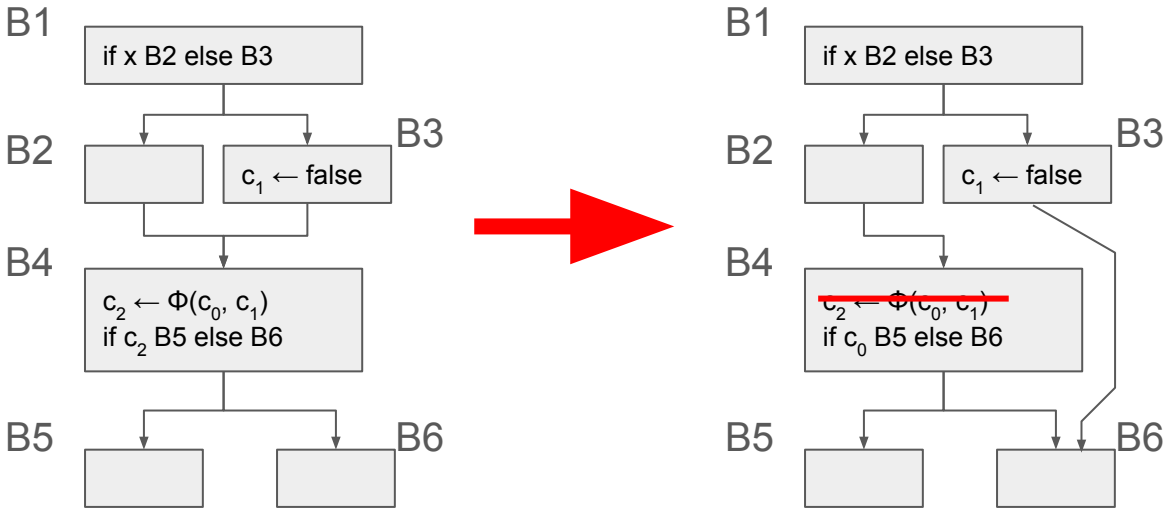
Jump Threading

- Form 3: direct jump to block with conditional branch that will have a known outcome
  - In SSA form with critical edges split, this looks like a double diamond with a Φ as the second condition



Jump Threading

- Form 3: direct jump to block with conditional branch that will have a known outcome
  - In SSA form with critical edges split, this looks like a double diamond with a Φ as the second condition



Jump Threading

- Form 3: direct jump to block with conditional branch that will have a known outcome
  - In SSA form with critical edges split, this looks like a double diamond with a Φ as the second condition