

CF Optimizations And Instruction Scheduling

15-411/15-611 Compiler Design

Ben L. Titzer and Seth Goldstein

April 17, 2025

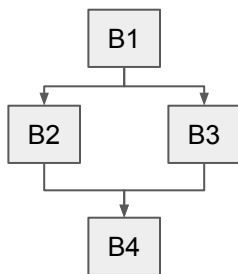
Today

- Extended basic blocks
- Control flow optimizations
 - Jump threading
 - Tail duplication
- Instruction Scheduling

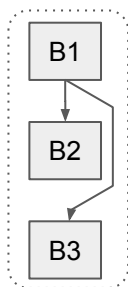
Extended Basic Blocks

- 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 $B_1, B_2 \dots B_n$ where only B_1 has multiple predecessors and all other blocks have exactly one predecessor in the sequence

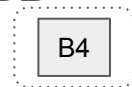
CFG



EBB



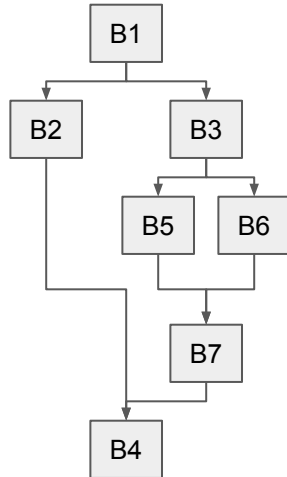
EBB



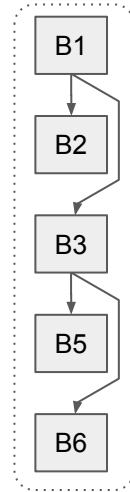
Extended Basic Blocks

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

CFG



EBB



EBB



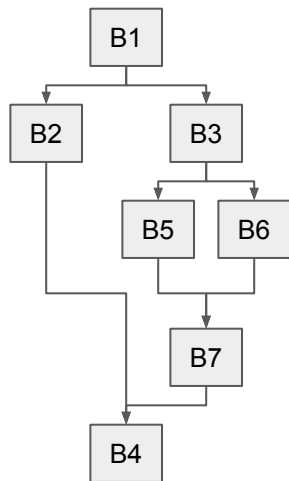
EBB



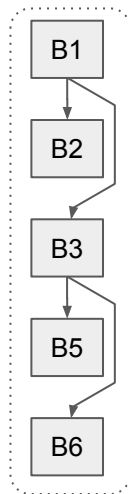
Extended Basic Blocks

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

CFG



EBB



EBB



EBB

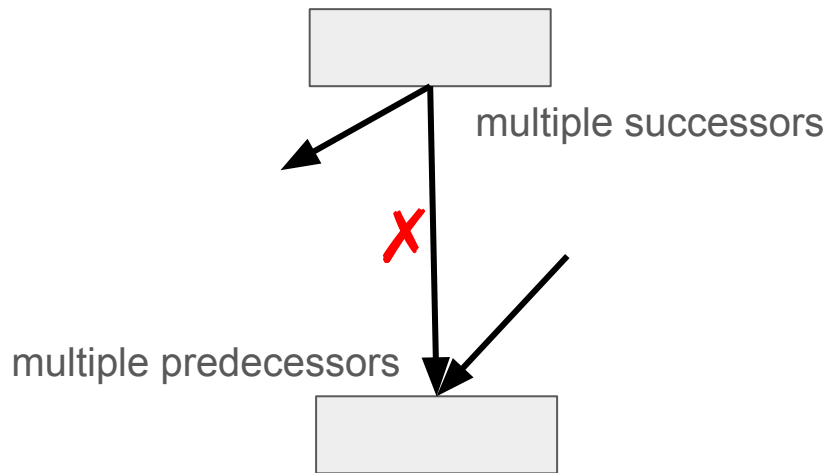


What about critical edges?

Splitting Critical Edges

- Recall from (deconstructing) SSA

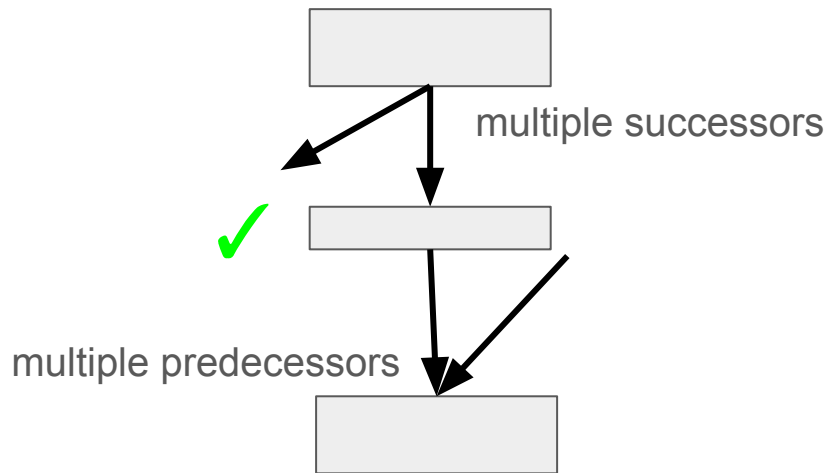
A critical edge is any edge that connects a block with multiple successors to a block with multiple predecessors.



Splitting Critical Edges

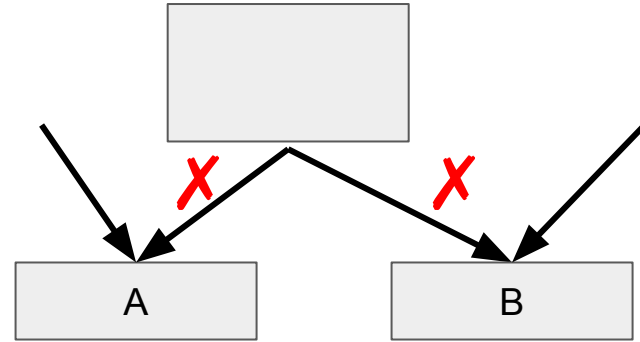
- Splitting critical edges is required for correct SSA deconstruction.
- Also benefits some optimizations like lazy code motion.

Splitting critical edges is an easy local transformation.



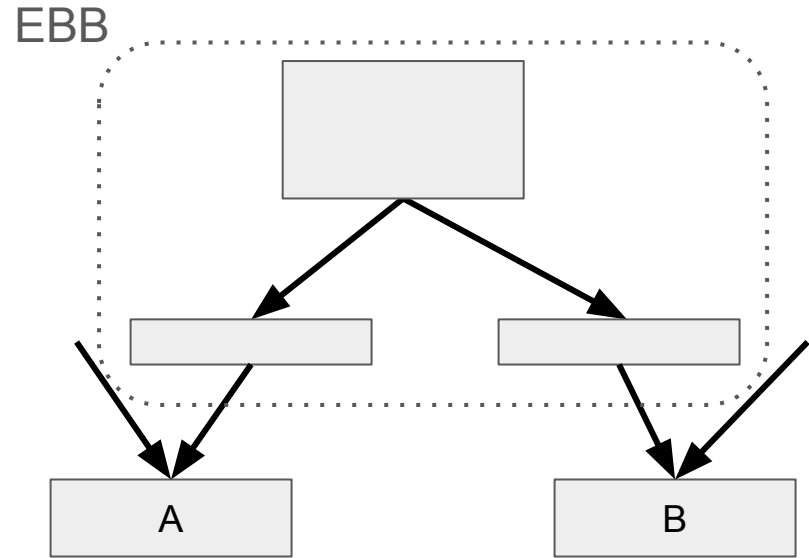
Splitting Critical Edges

Critical edges make conditional control flow seem unnecessarily general.



Splitting Critical Edges

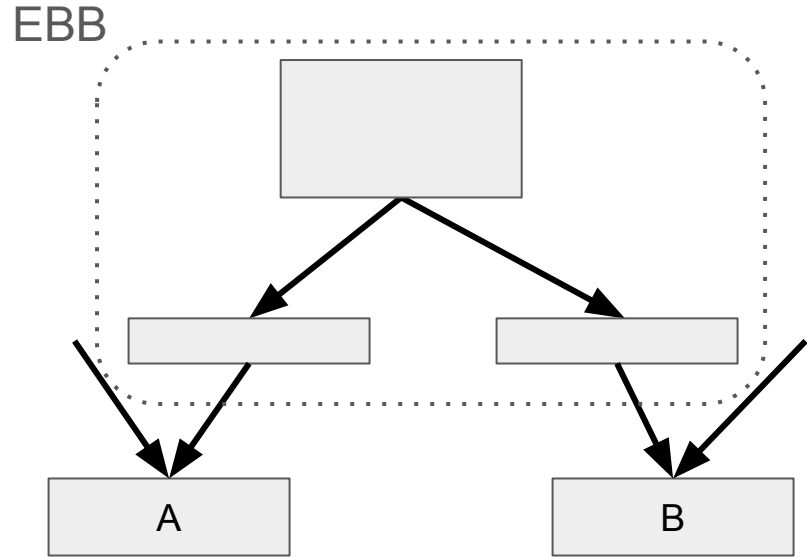
Splitting critical edges can make for larger EBBs.



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.

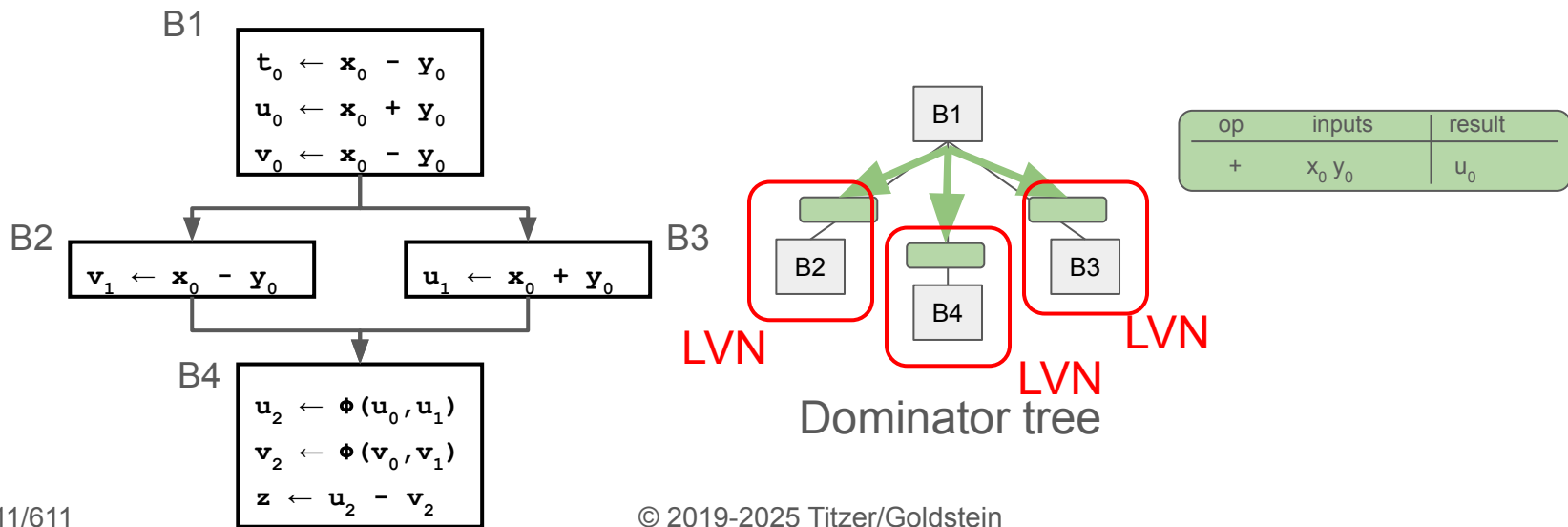


Using Extended Basic Blocks

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

Global Value Numbering using Dominators

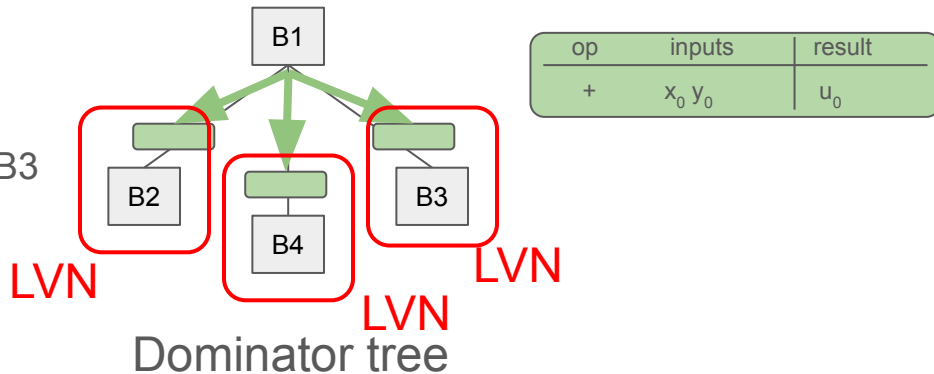
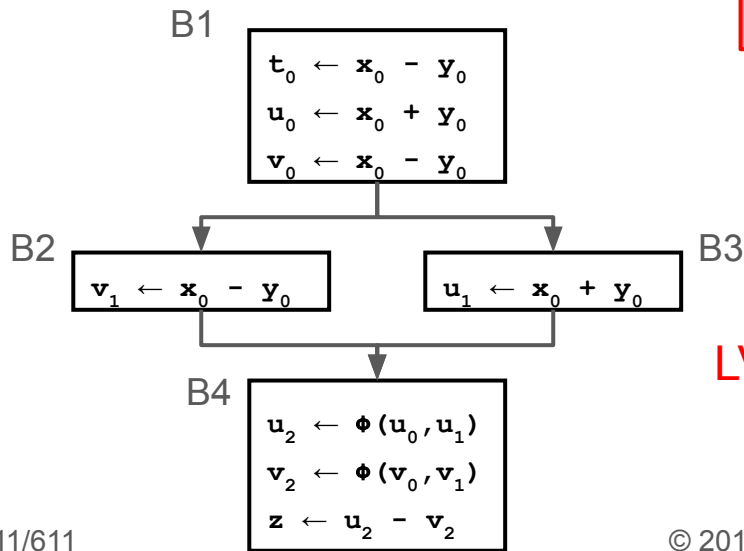
- Recall: LVN finds redundant computations in a basic block
- Dominator-based GVN algorithm trades accuracy for speed
- Propagate local value numbering map from dominator to dominated nodes



Global Value Numbering using Dominators

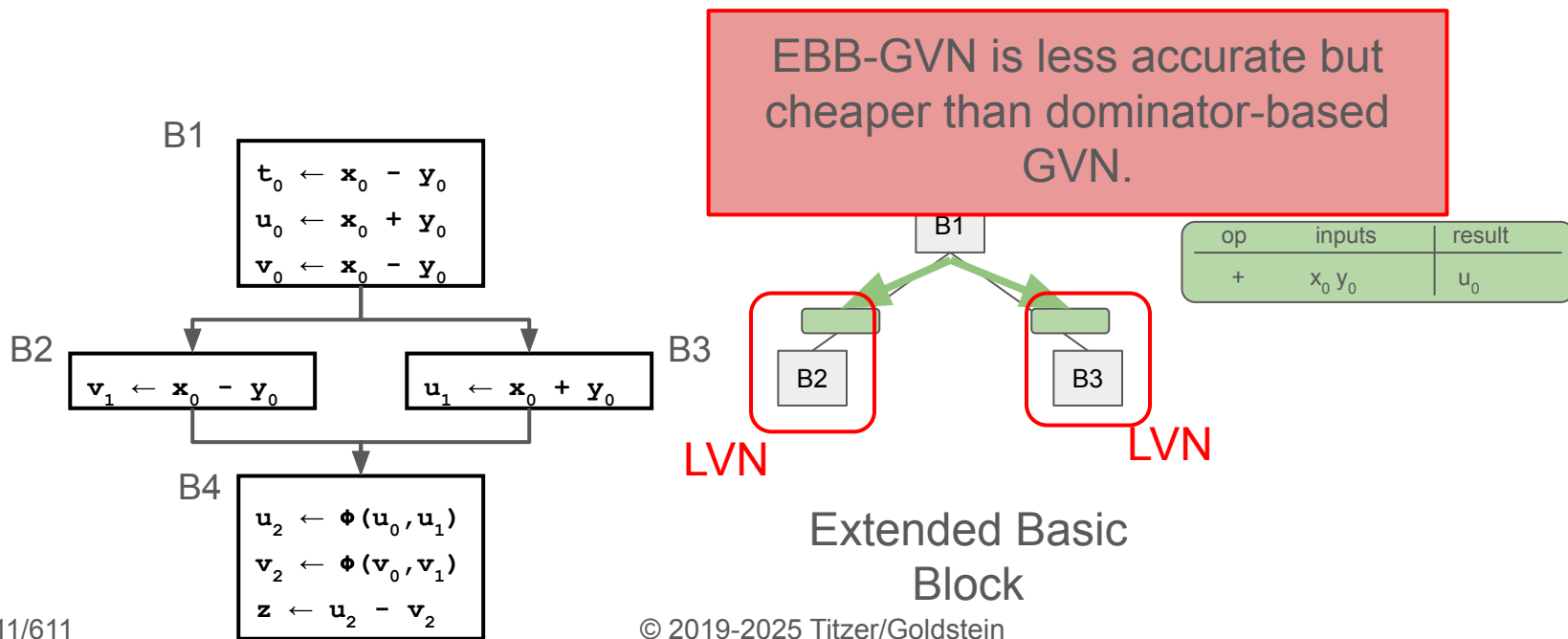
- Recall: LVN finds redundant computations in a basic block
- Dominator-based GVN algorithm trades accuracy for speed
- Propagate local value numbering map from dominator to dominated nodes

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



Global Value Numbering using Dominators

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



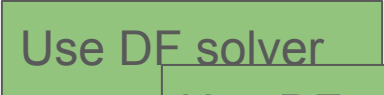

Control Flow Optimizations: Tail duplication

- 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 ϕ s, harder to reason through

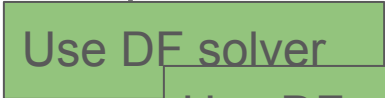
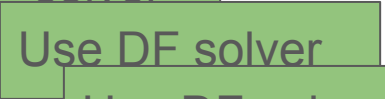

Control Flow Optimizations: Tail duplication

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





Control Flow Optimizations: Tail duplication

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

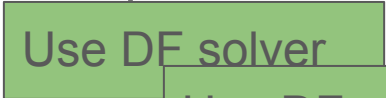
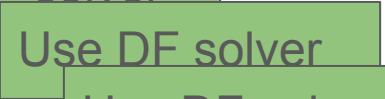


Control Flow Optimizations: Tail duplication

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

Control Flow Optimizations: Tail duplication

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

Control Flow Optimizations: Tail duplication

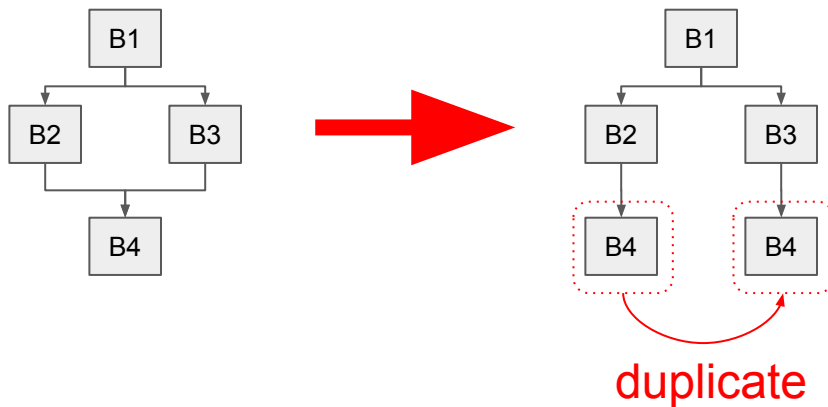
- Often, merges in the control flow complicate analyses
- Dataflow equations with unions  Use DF solver
- Missed value numbering opportunities  Use DF solver
- Merges are not in predecessors' EBBs  Use DF solver
- Merges often have ϕ s, harder to reason through  Use DF solver

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

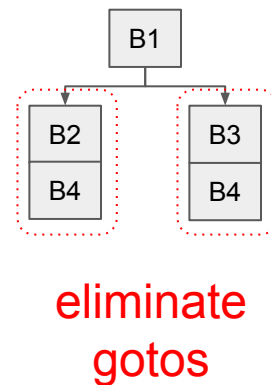
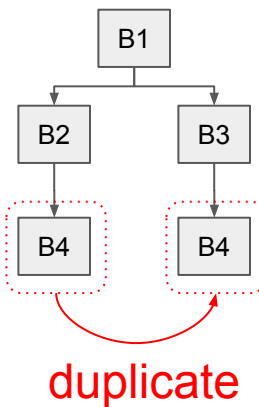
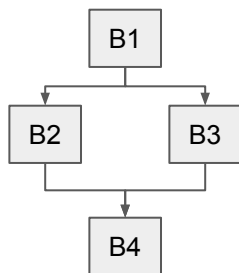
CFG



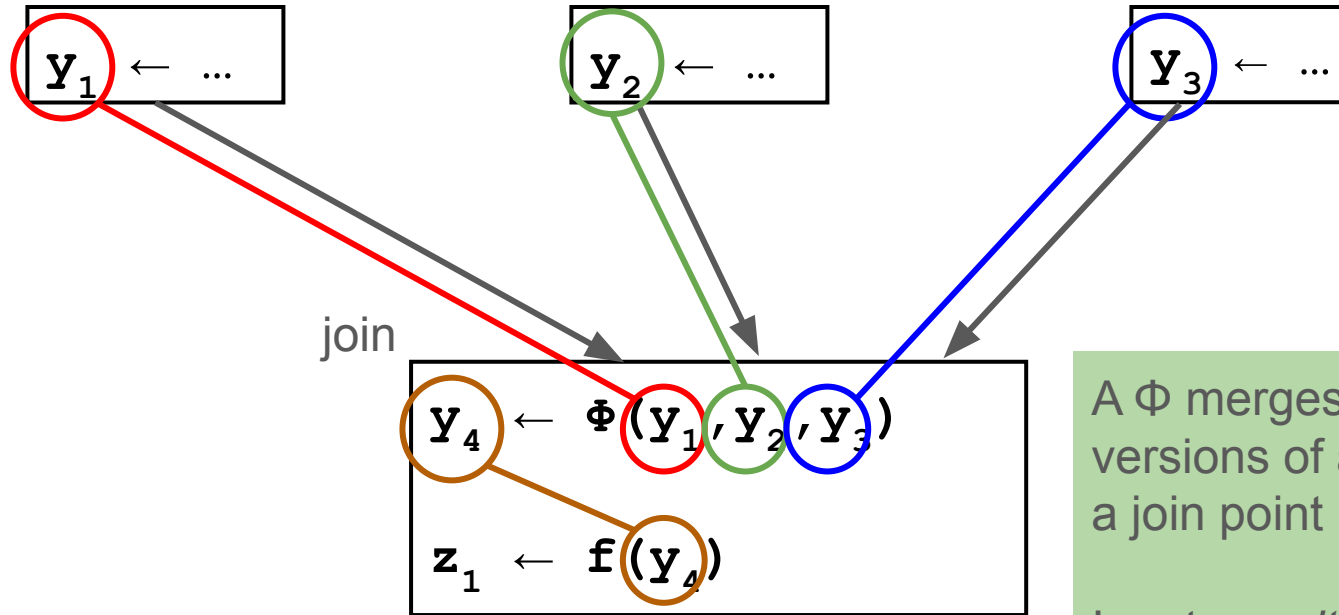
Tail Duplication

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

CFG



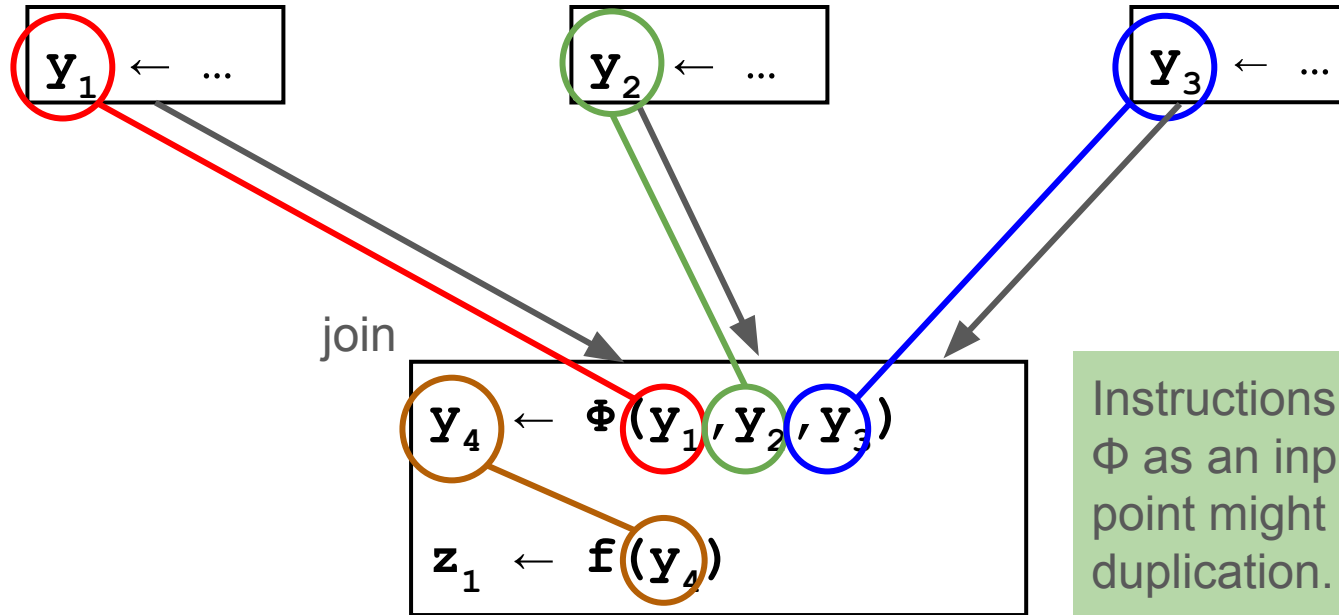
Tail Duplication Transformation



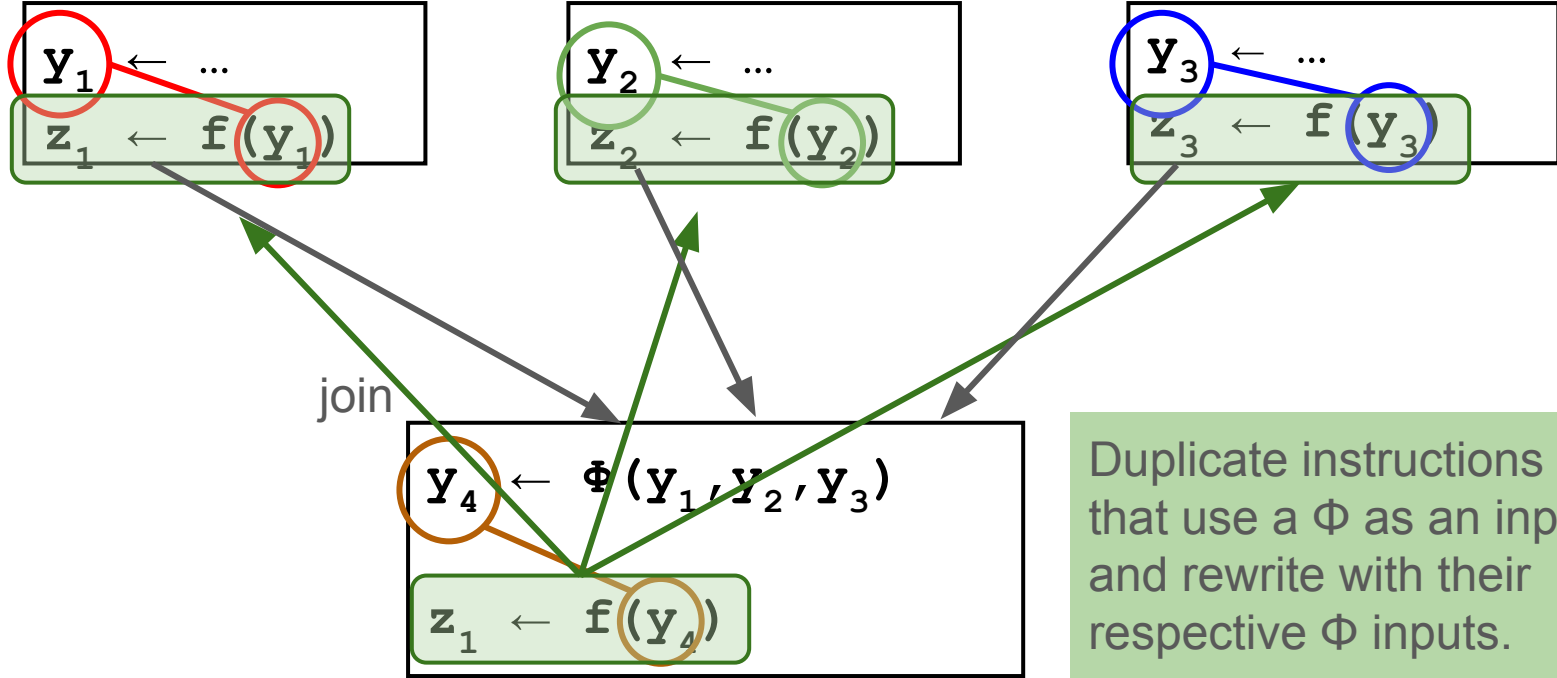
A Φ merges multiple versions of a variable at a join point in the CFG.

Inputs *positionally correspond* with predecessor edges.

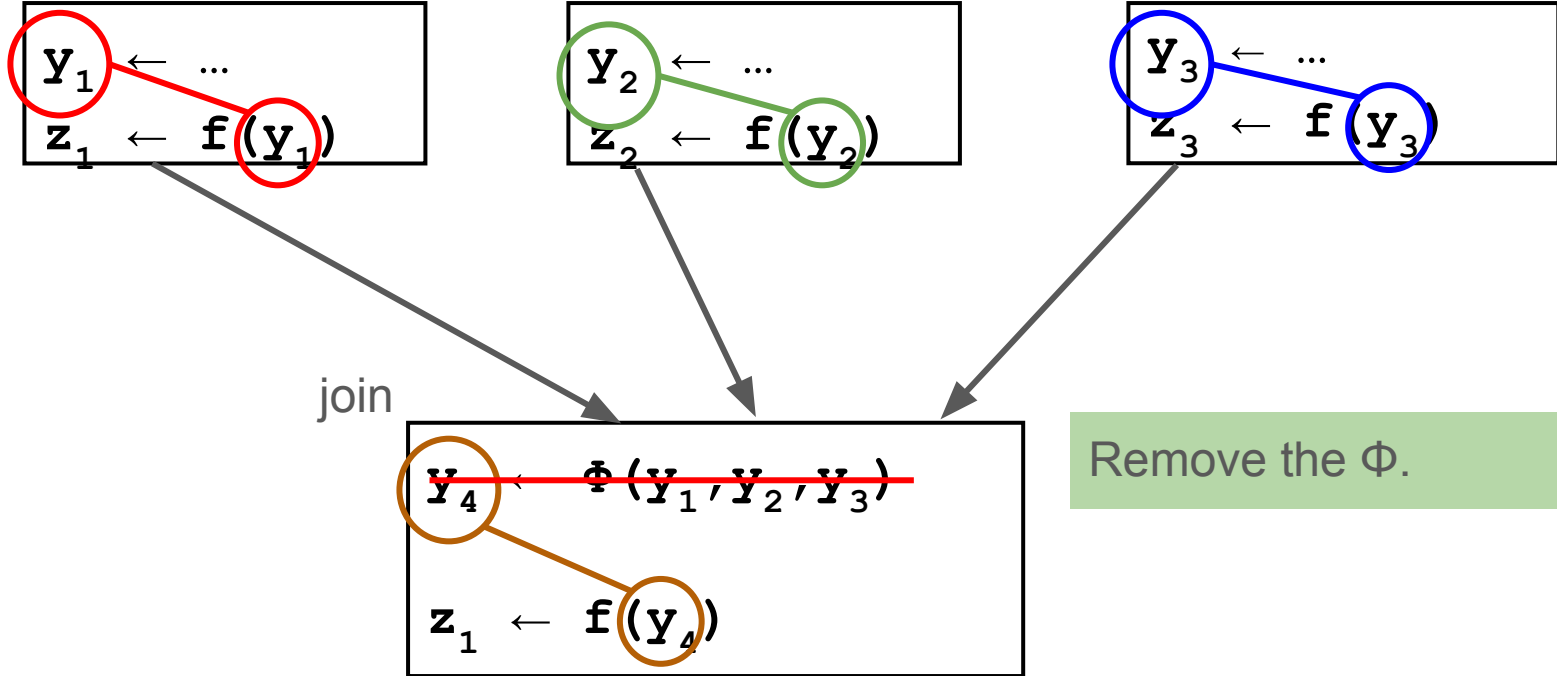
Tail Duplication Transformation



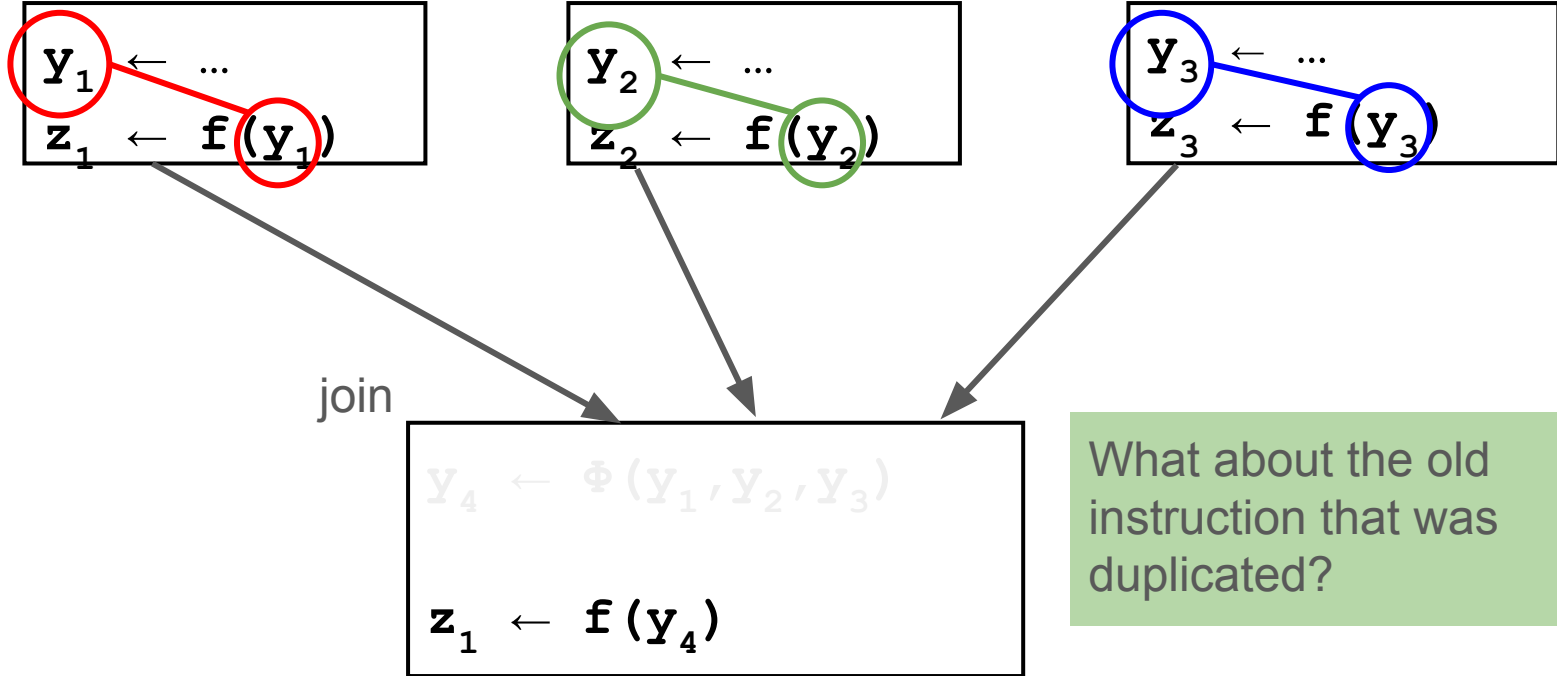
Tail Duplication Transformation



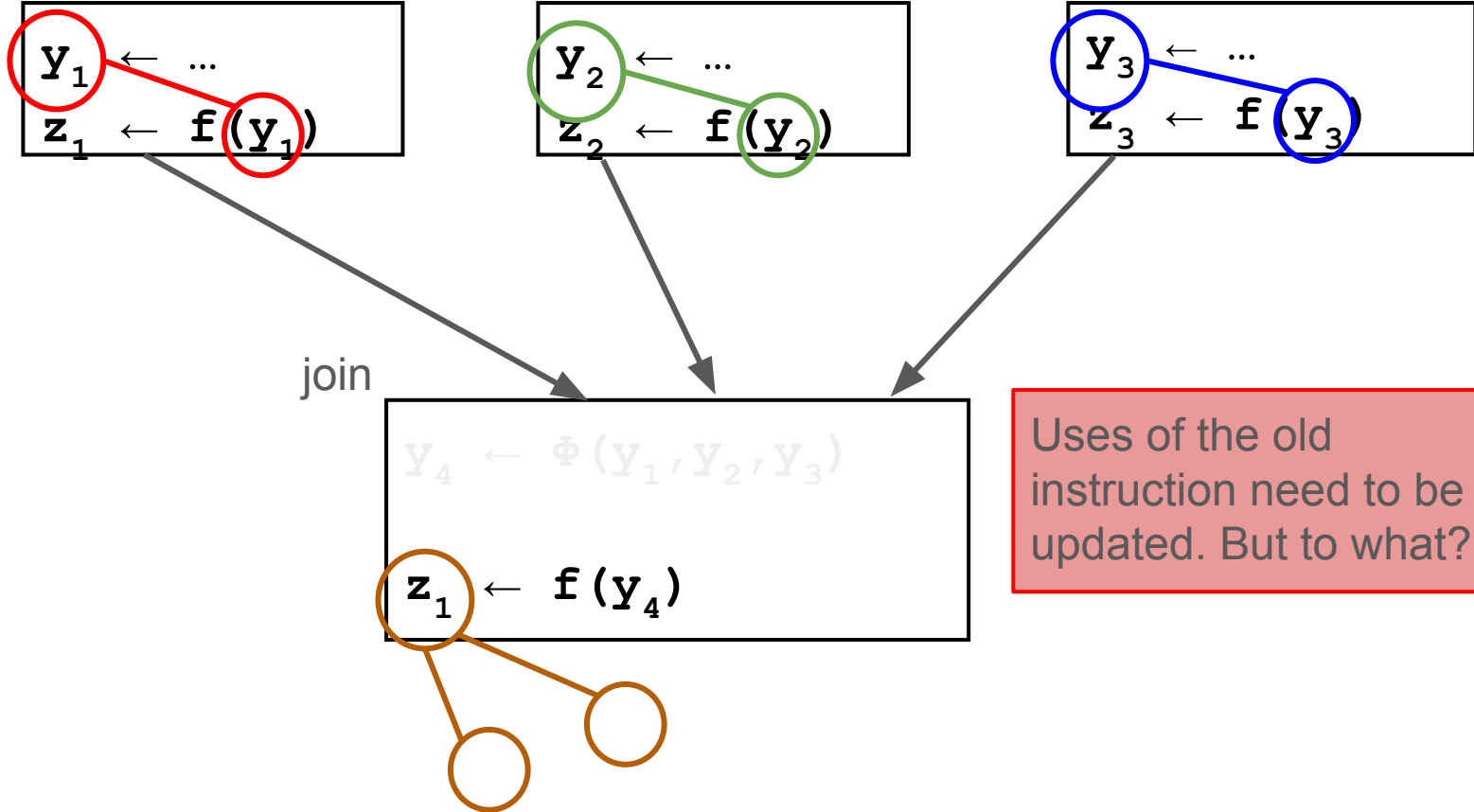
Tail Duplication Transformation



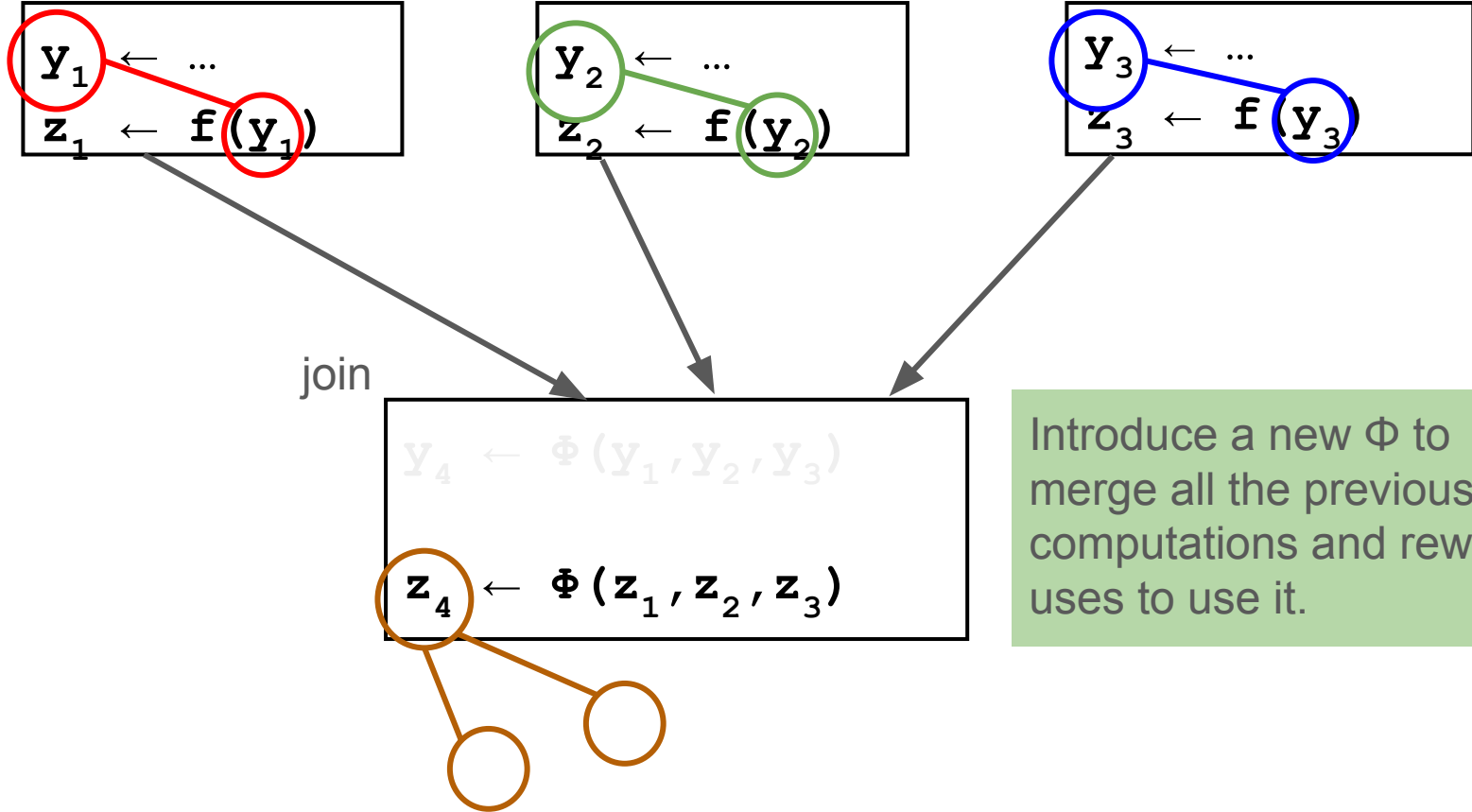
Tail Duplication Transformation



Tail Duplication Transformation

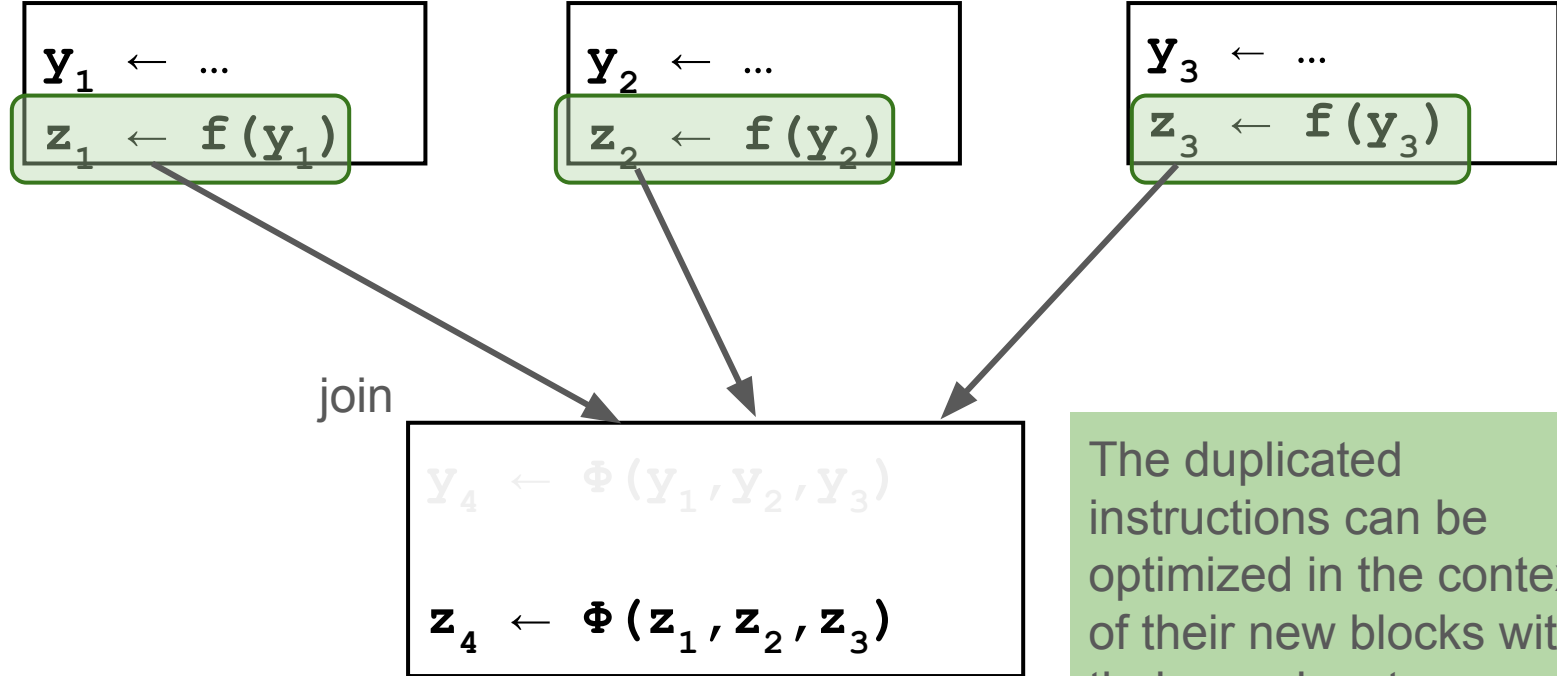


Tail Duplication Transformation



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

Tail Duplication Transformation



Tail Duplication Summary

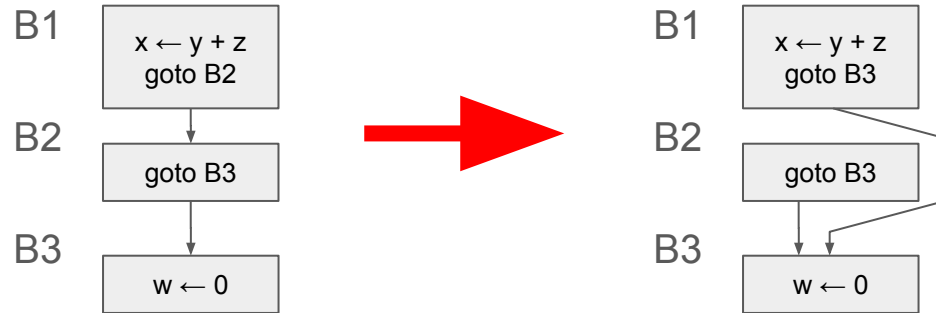
- Tail duplication copies join blocks to allow more optimization from context of predecessors.
- With SSA and edge split form, it's possible to do this without introducing new control flow.
- Like any optimization that copies code, it is a tradeoff
 - Increased code size, I-cache pressure
 - Specialization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

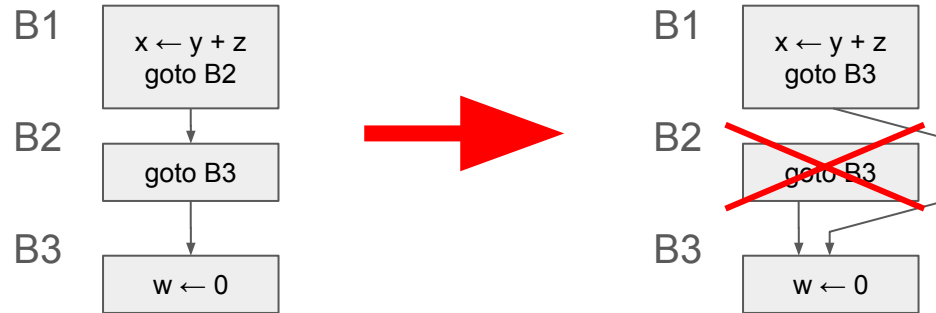
Jump Threading

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



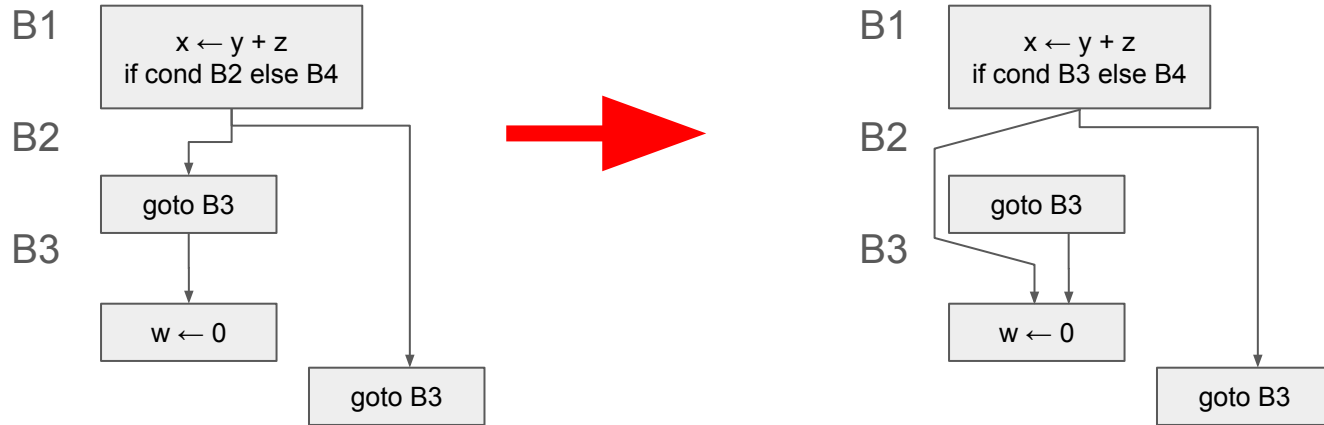
Jump Threading

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



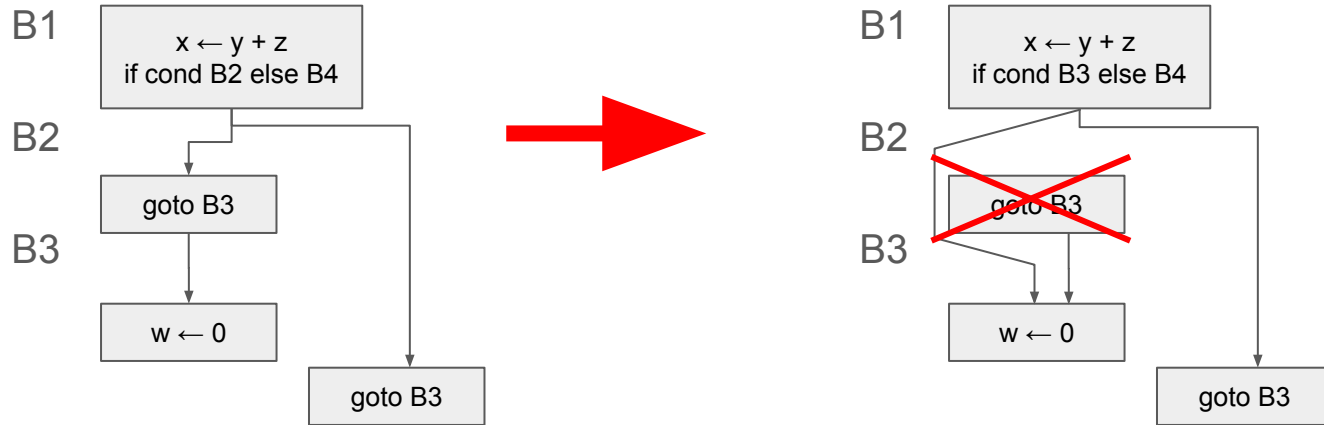
Jump Threading

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



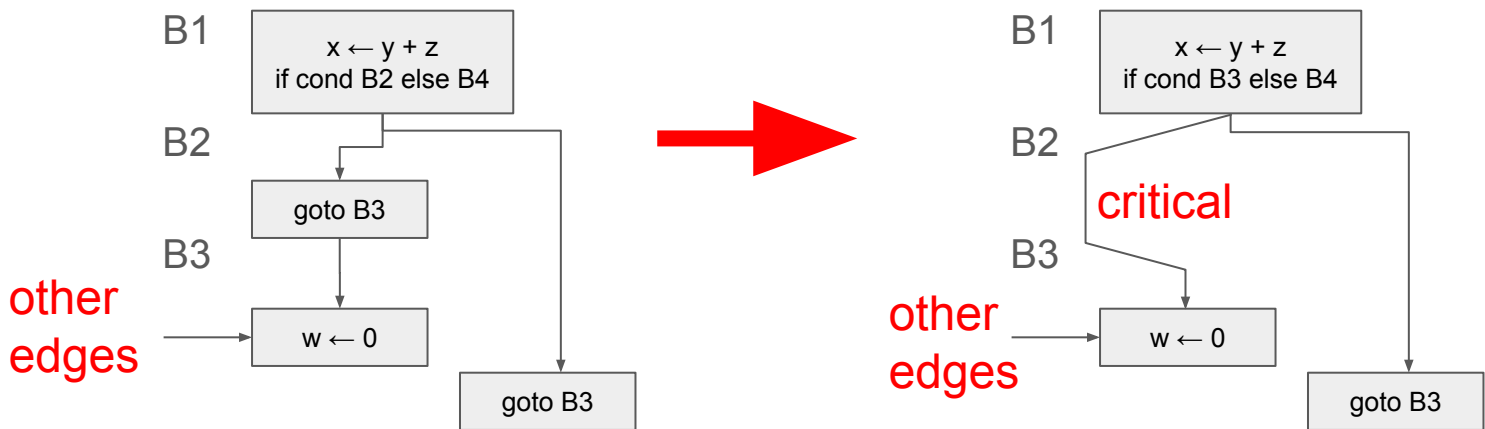
Jump Threading

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



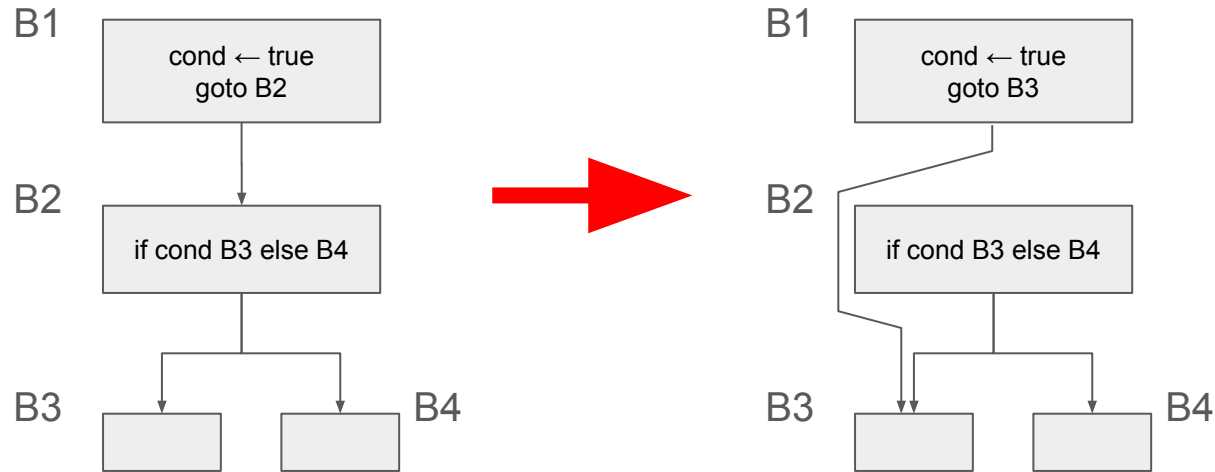
Jump Threading

- 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

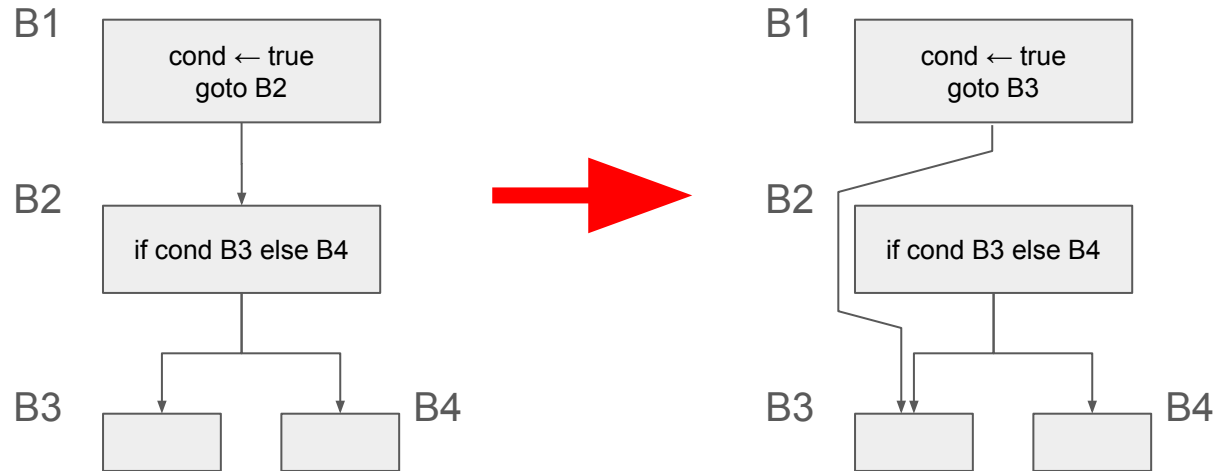
- Form 3: direct jump to block with conditional branch that will have a known outcome



Jump Threading

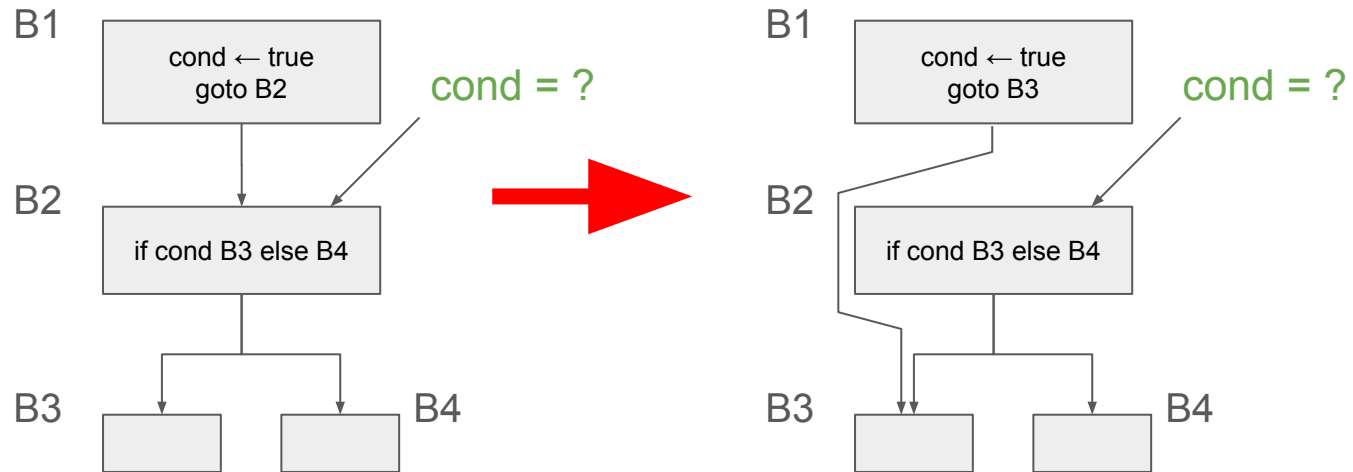
- Form 3: direct jump to block with conditional branch that will have a known outcome

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



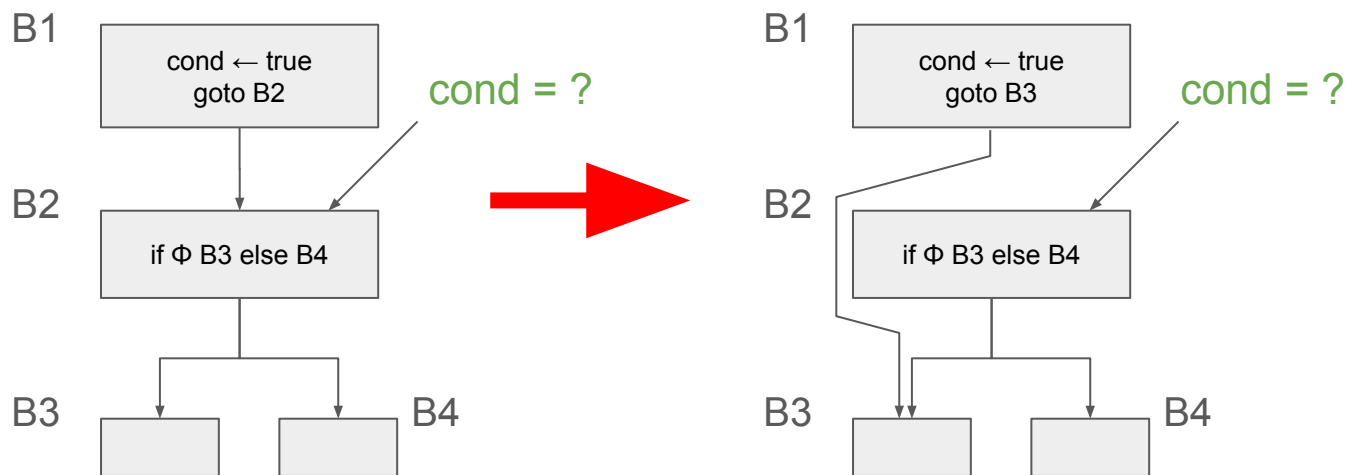
Jump Threading

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



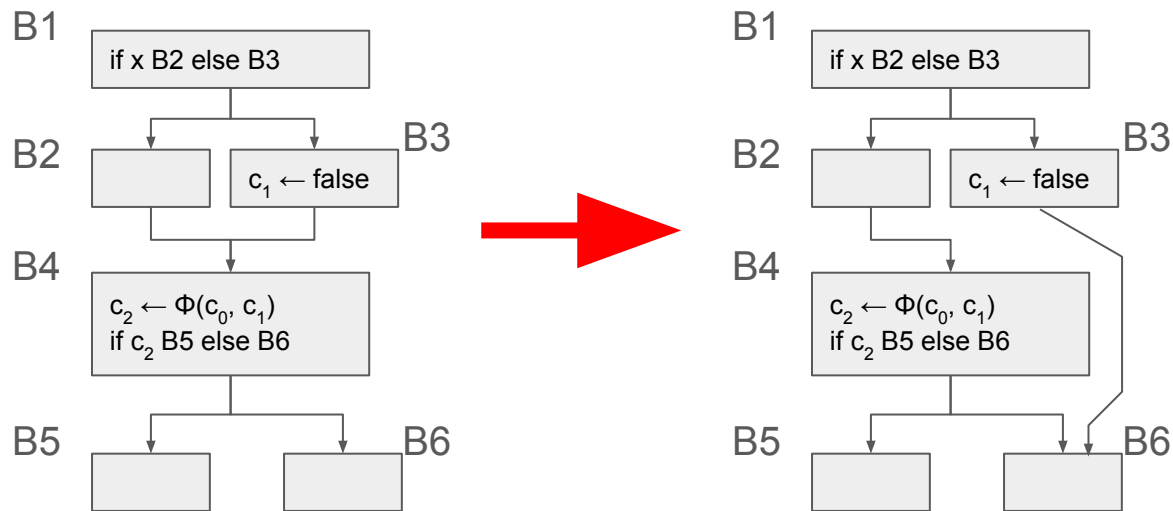
Jump Threading

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



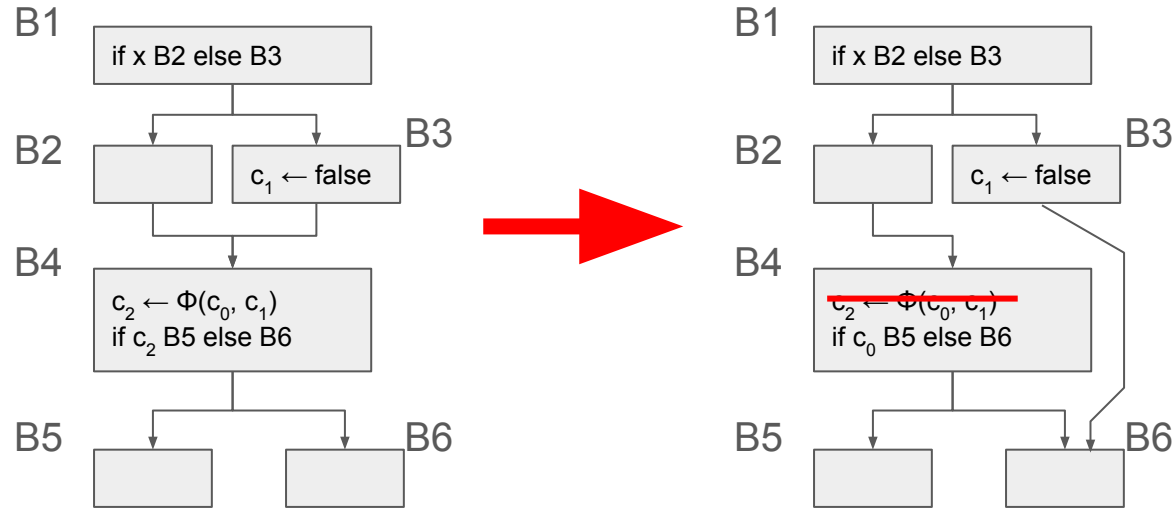
Jump Threading

- 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



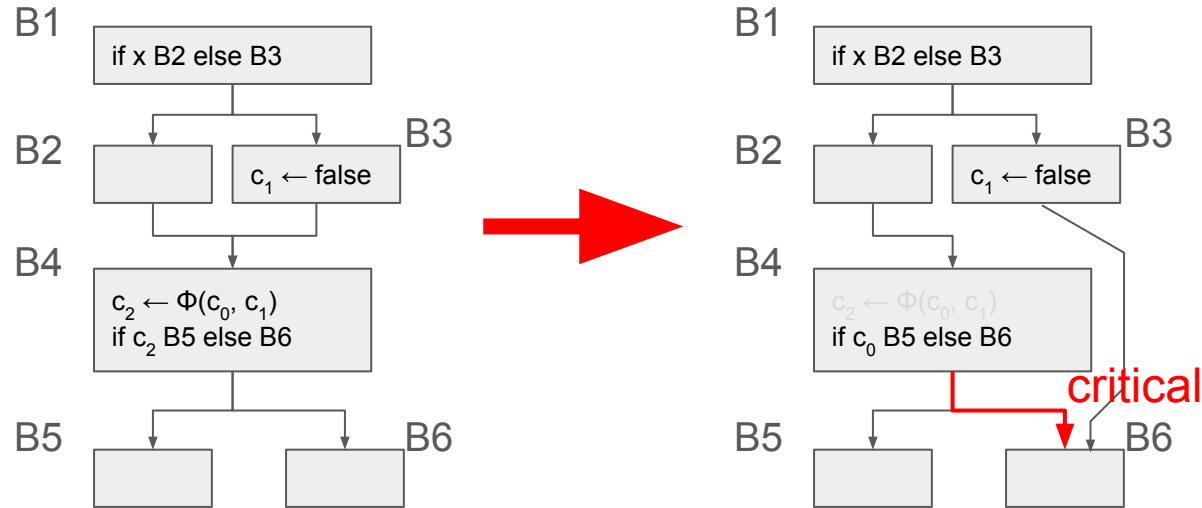
Jump Threading

- 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



Instruction Scheduling

- Instruction latency and instruction-level parallelism are critical to good performance
 - Instruction $x \leftarrow a + b$ cannot be executed until a and b are done
 - Latency of a and b determined by their operation
 - Latency of x determined by the $+$ operation
- Depends entirely on CPU's specific microarchitecture
 - Functional units: CPU resources able to execute different kinds of instructions
 - Pipelining: latency between starting an instruction and result available
 - Issue bandwidth: how many instructions can be “started” per cycle
 - Out-of-order execution: CPUs execute instructions when inputs are ready

Instruction Scheduling

- Compilers can model a specific CPU's characteristics and reorder sequences of instructions for better performance
- Basic block (local) or inter-block (superlocal) scopes are typical
 - Global instruction scheduling generally not used; too many interactions with other optimizations
- We'll focus on local instruction scheduling.

(Board)