Check-in

Review Lecture: Flowgraphs

Draw the CFG of this procedure

```
[BEGIN v LOCALS]
a (formal arg of 8)
tmp0 (tmp var of 8 bytes)
tmp1 (tmp var of 8 bytes)
tmp2 (tmp var of 8 bytes)
[END v LOCALS]
L0:    enter v
L1:    getarg 1 [a]
L2:    [tmp0] := [a] LT64 3
L3:    IFZ [tmp0] GOTO L12
L4:    nop
L5:    [tmp1] := [a] LT64 4
L6:    IFZ [tmp1] GOTO L10
L7:    [tmp2] := [a] ADD64 1
L8:    [a] := [tmp2]
L9:    goto L4
L10:   nop
L11:   REPORT [a]
L12:   nop
L13:   leave v
```

Announcements

Administrivia

Drew Davidson | University of Kansas

EECS 665

COMPILER

CONSTRUCTION

Dataflow

Previously…

Review Lecture: Flowgraphs

Control flow graphs:

A hybrid IR/ a structural overlay

- Rationale

  Useful for visualizing program flow

- Construction

  Identify basic blocks (BBLs)

  Connect edges on control transfer

- Uses

  Program understanding

You should know

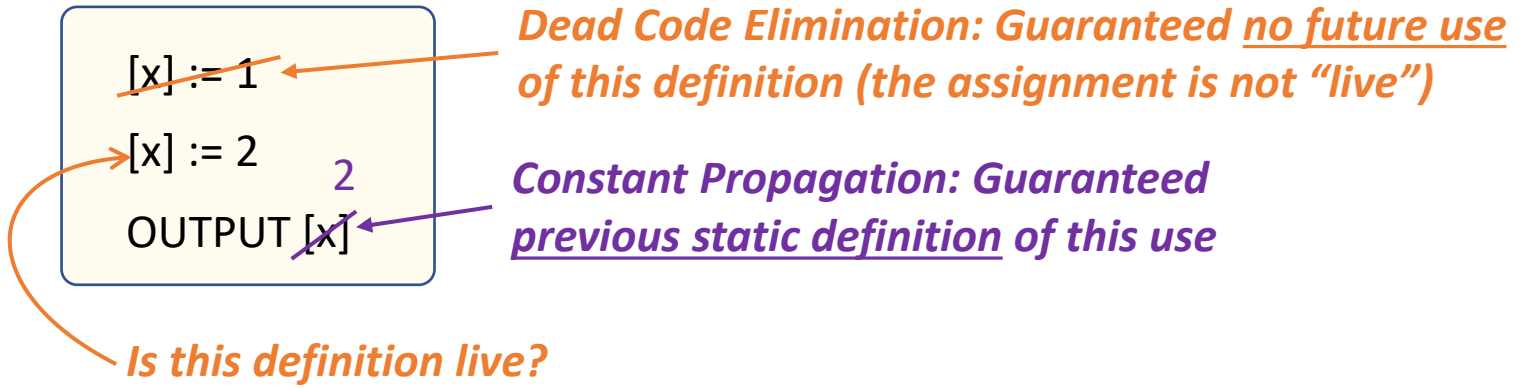
- Basic Blocks
- How to build a CFG
- The idea of some local optimizations
  - Dead Code Elimination
  - Common Subexpression Elimination
  - Constant/Copy Propagation



Optimization

Recall: Some Local Optimizations

Review - Basic Block Optimization



¯_(ツ)_/¯

Without knowing x's use *outside this block*
We have to keep it

Today's Outline

Dataflow

Dataflow analysis

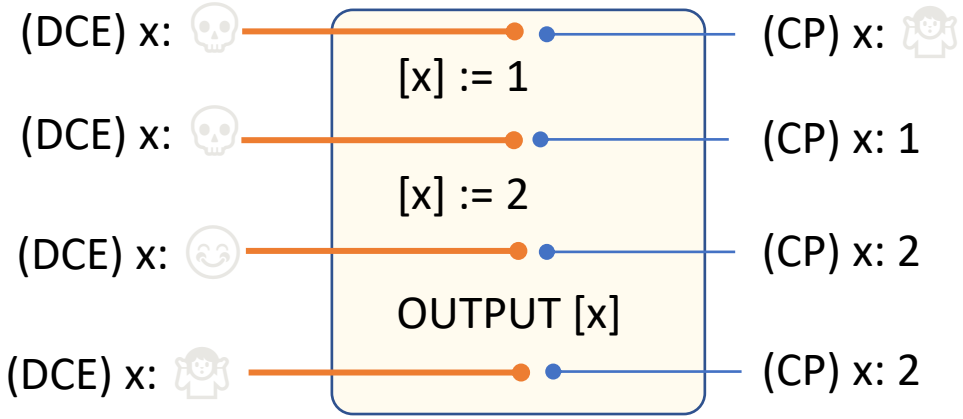
- Intuition
- Concepts
- Dataflow frameworks



Optimization

Consider What Info We Know

Basic Block Optimization



Dead Code Elimination: Guaranteed no future use of this definition (the assignment is not "live")

Constant Propagation: Guaranteed previous static definition of this use

For Dead Code Elimination, definition could be marked

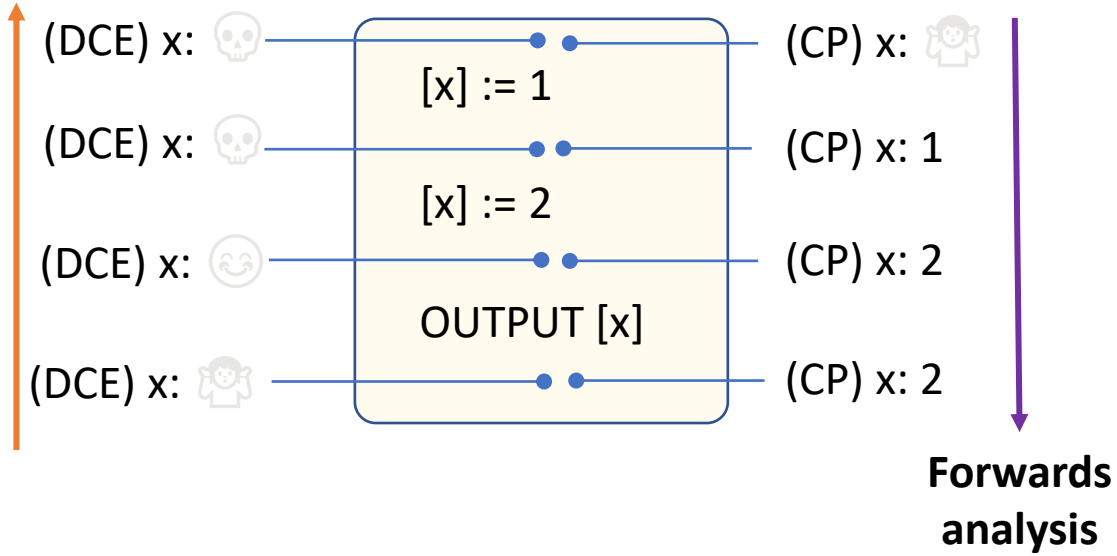
Known Live	Known Dead	Not Enough Info

For Constant Propagation, use could be marked

Guaranteed Constant	Guaranteed Non-Constant	Not Enough Info
<value>	> 1 value or	

Consider Where We Learn Info

Basic Block Optimization



Dead Code Elimination: Guaranteed no future use of this definition (the assignment is not "live")

Constant Propagation: Guaranteed previous static definition of this use

For Dead Code Elimination, definition could be marked

Known Live	Known Dead	Not Enough Info

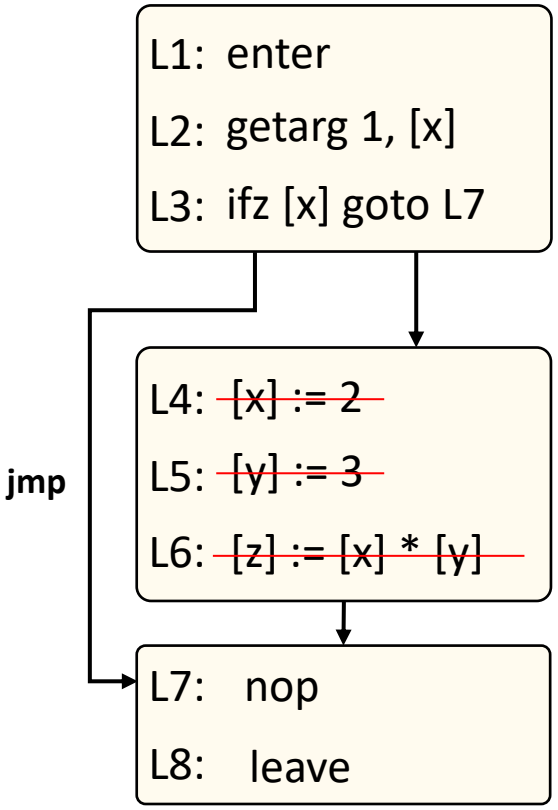
For Constant Propagation, use could be marked

Guaranteed Constant	Guaranteed Non-Constant	Not Enough Info
<value>	> 1 value or	

Beyond Local Optimization

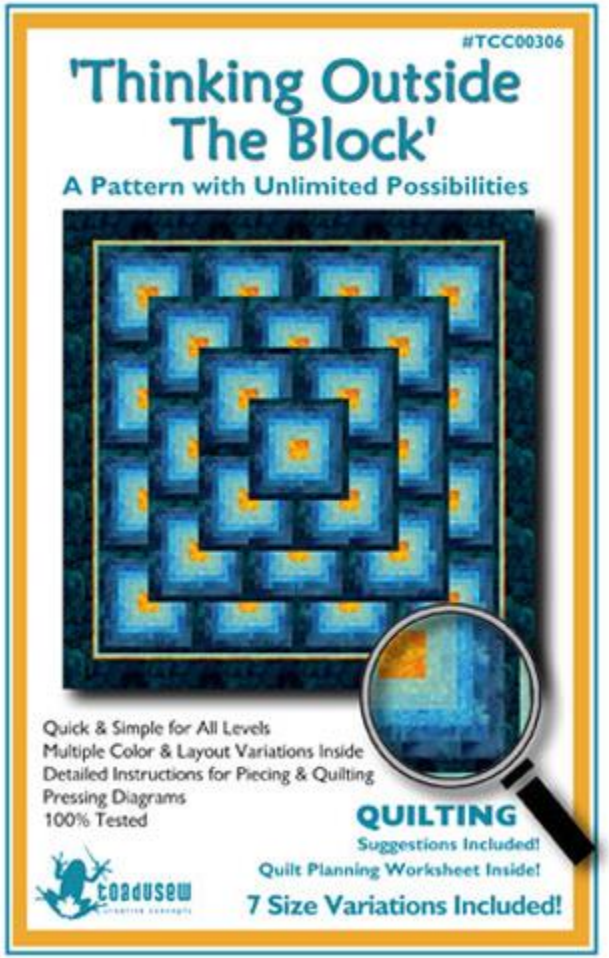
Dataflow

One possible CFG



L6 is dead!

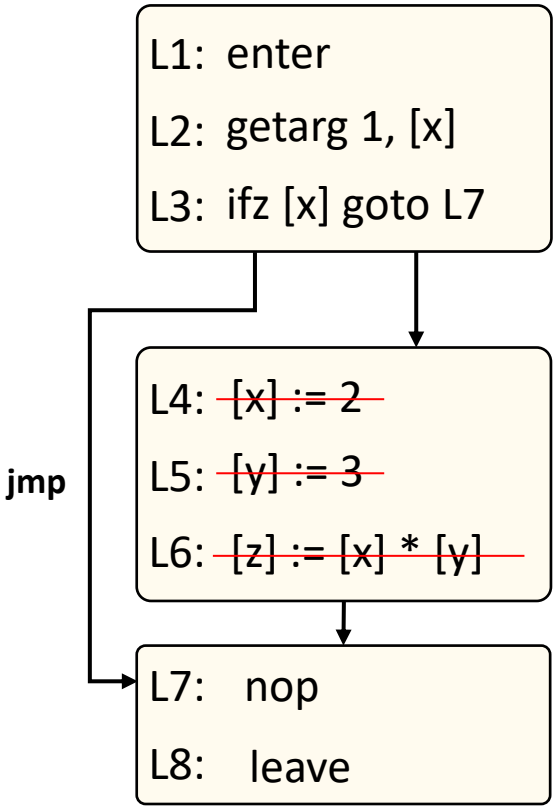
(causes L4 and L5 to be dead)



Beyond Local Optimization

Dataflow

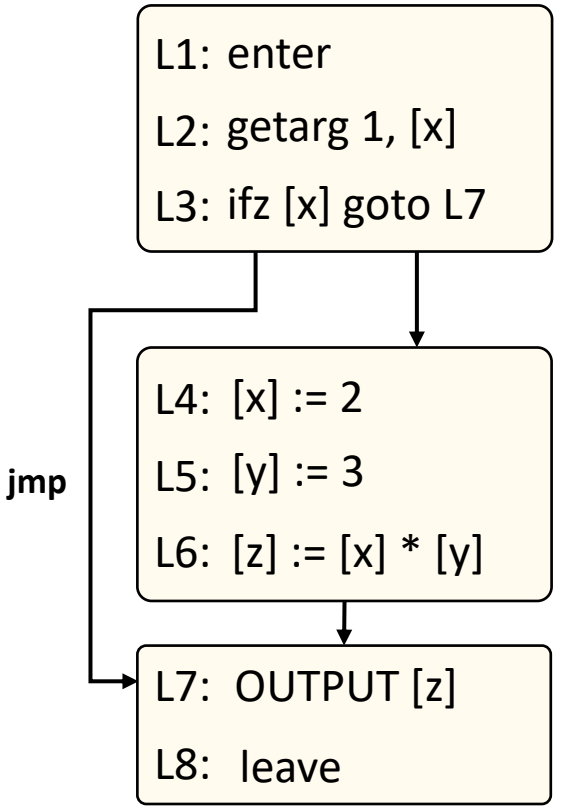
One possible CFG



L6 is dead!

(causes L4 and L5 to be dead)

Another possible CFG



L6 is live!

Cannot be removed

Today's Outline

Dataflow

Dataflow analysis

- Intuition
- Concepts
- Dataflow frameworks



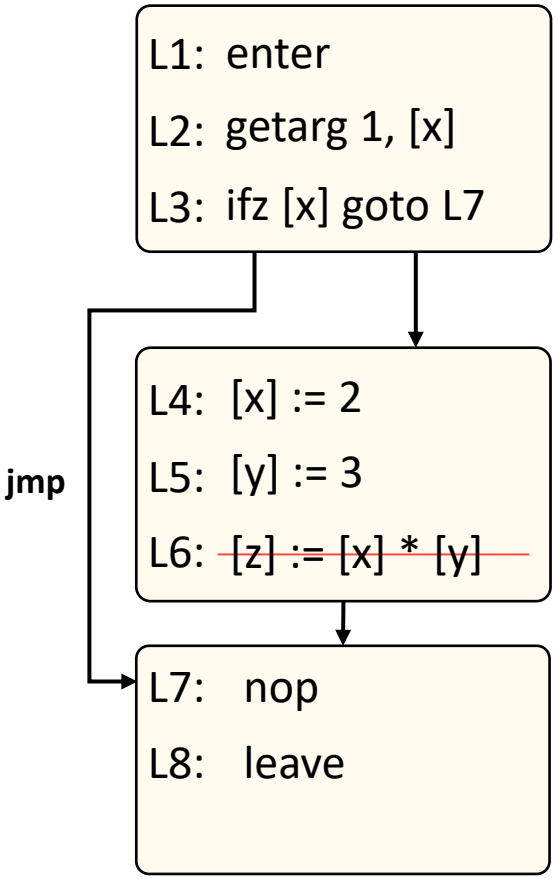
Optimization

Generalizing Dataflow Intuition

Dataflow Intuition

Let's revisit the example, and ask some leading questions

One possible CFG



Why is L6 dead? ≡ Why isn't L6 live?



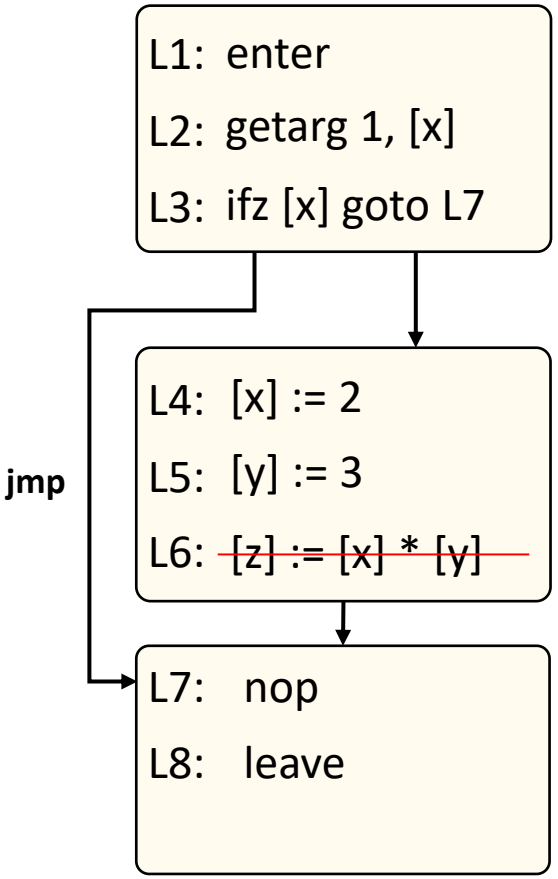
Returning to the scene of the crime

Generalizing Dataflow Intuition

Dataflow Intuition

Let's revisit the example, and ask some leading questions

One possible CFG



Why is L6 dead? ≡ Why isn't L6 live?

The thing defined was no longer useful

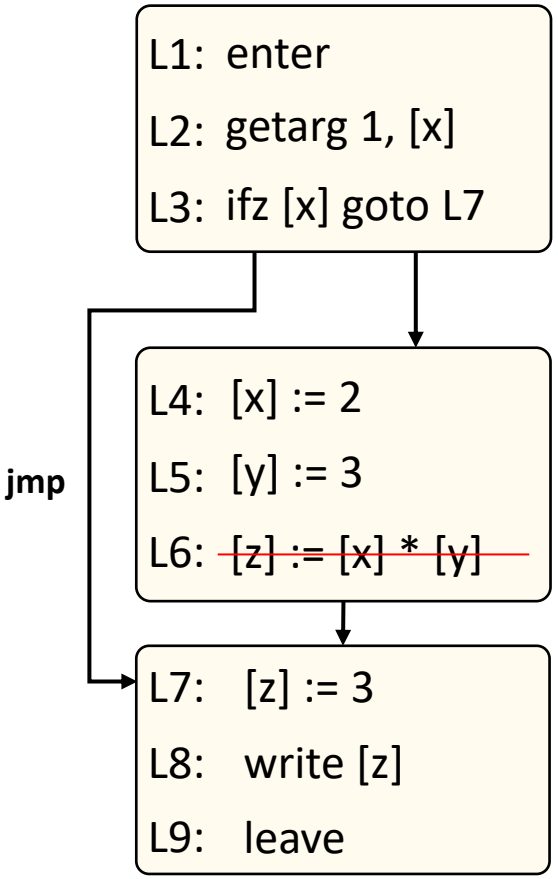
"died of natural causes"

Generalizing Dataflow Intuition

Dataflow Intuition

Let's revisit the example, and ask some leading questions

One possible CFG



Why is L6 dead? ≡ Why isn't L6 live?

The thing defined was no longer useful

"died of natural causes"

The thing defined was redefined before use

"it was killed!"

Need to *gather some facts* to tell if a statement is dead

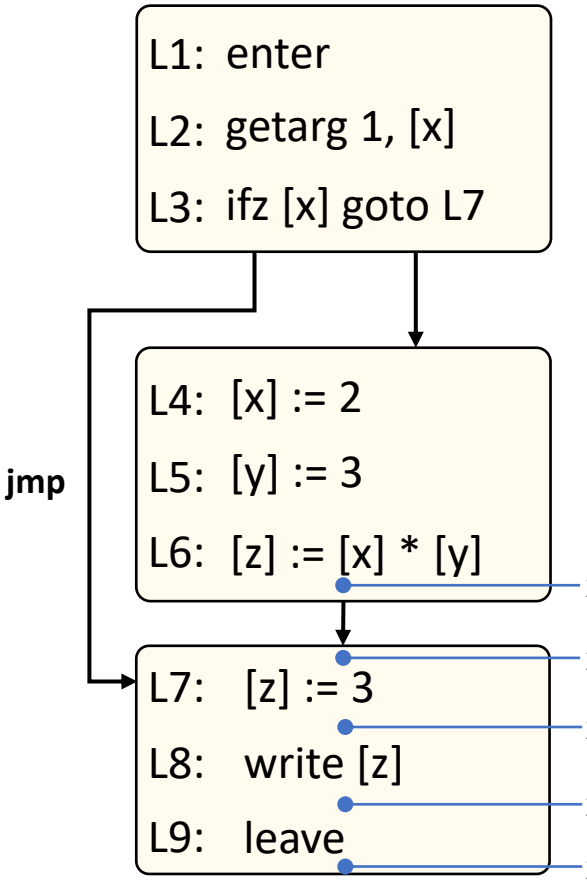
What variables are useful at each program point?

What variables are killed at each program point?

Generalizing Dataflow Intuition

Dataflow Intuition

One possible CFG



Need to *gather some facts* to tell if a statement is dead

What things are useful at each program point?

What things are killed at each program point?

x: 💀, y: 💀, z: 💀 nothing live at end of block, because nothing live at **entry to successors**

x: 💀, y: 💀, z: 💀 Previous definitions of z are not useful, because L7 killed them

x: 💀, y: 💀, z: 🧠 z is useful, because it's used later on (in L8)

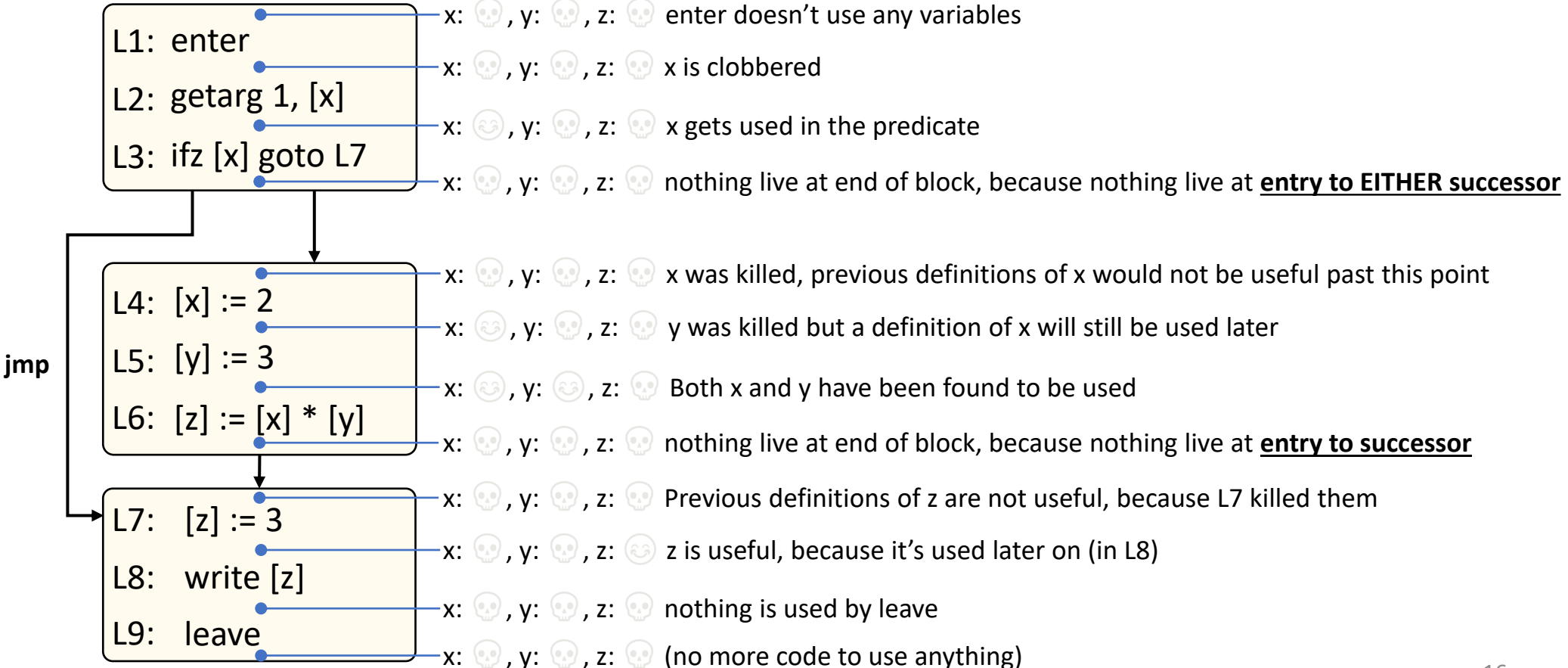
x: 💀, y: 💀, z: 💀 nothing is used by leave

x: 💀, y: 💀, z: 💀 (no more code to use anything)

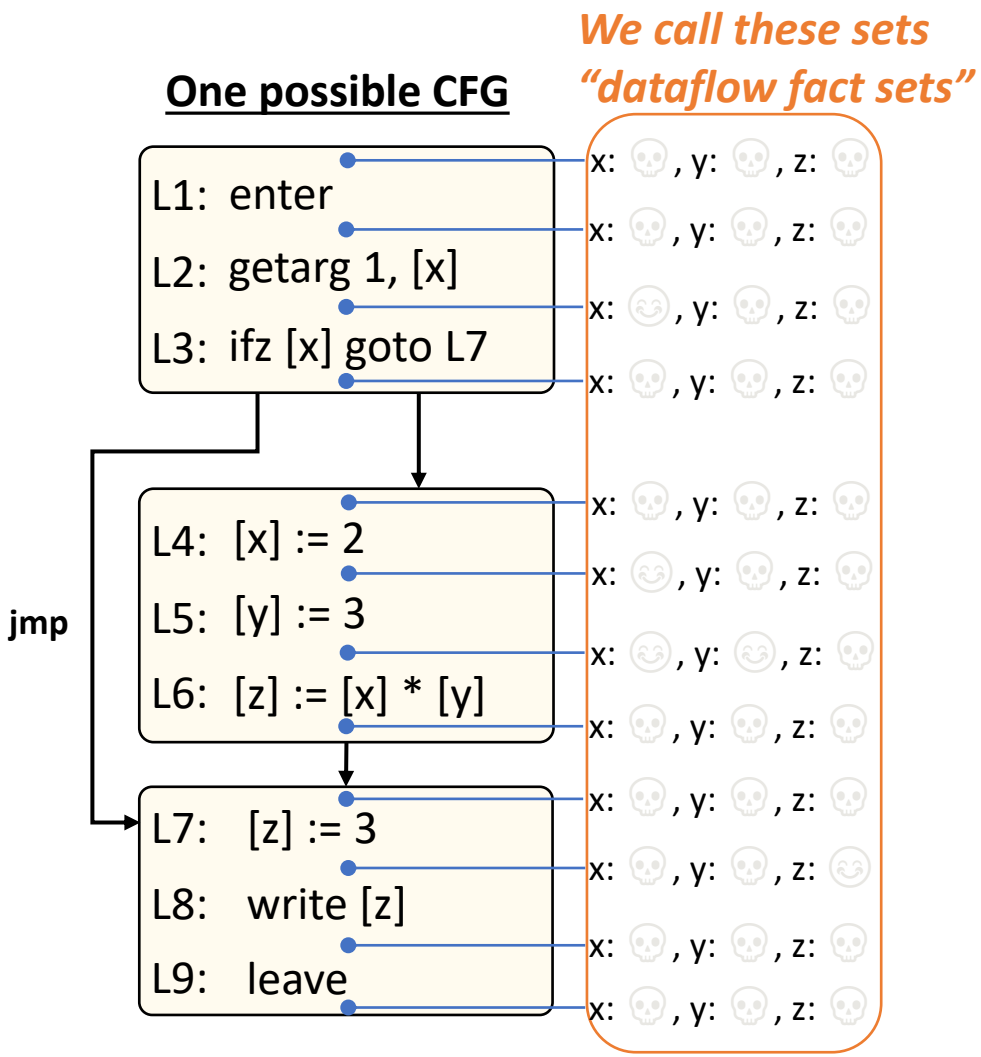
Generalizing Dataflow Intuition

Dataflow Intuition

One possible CFG



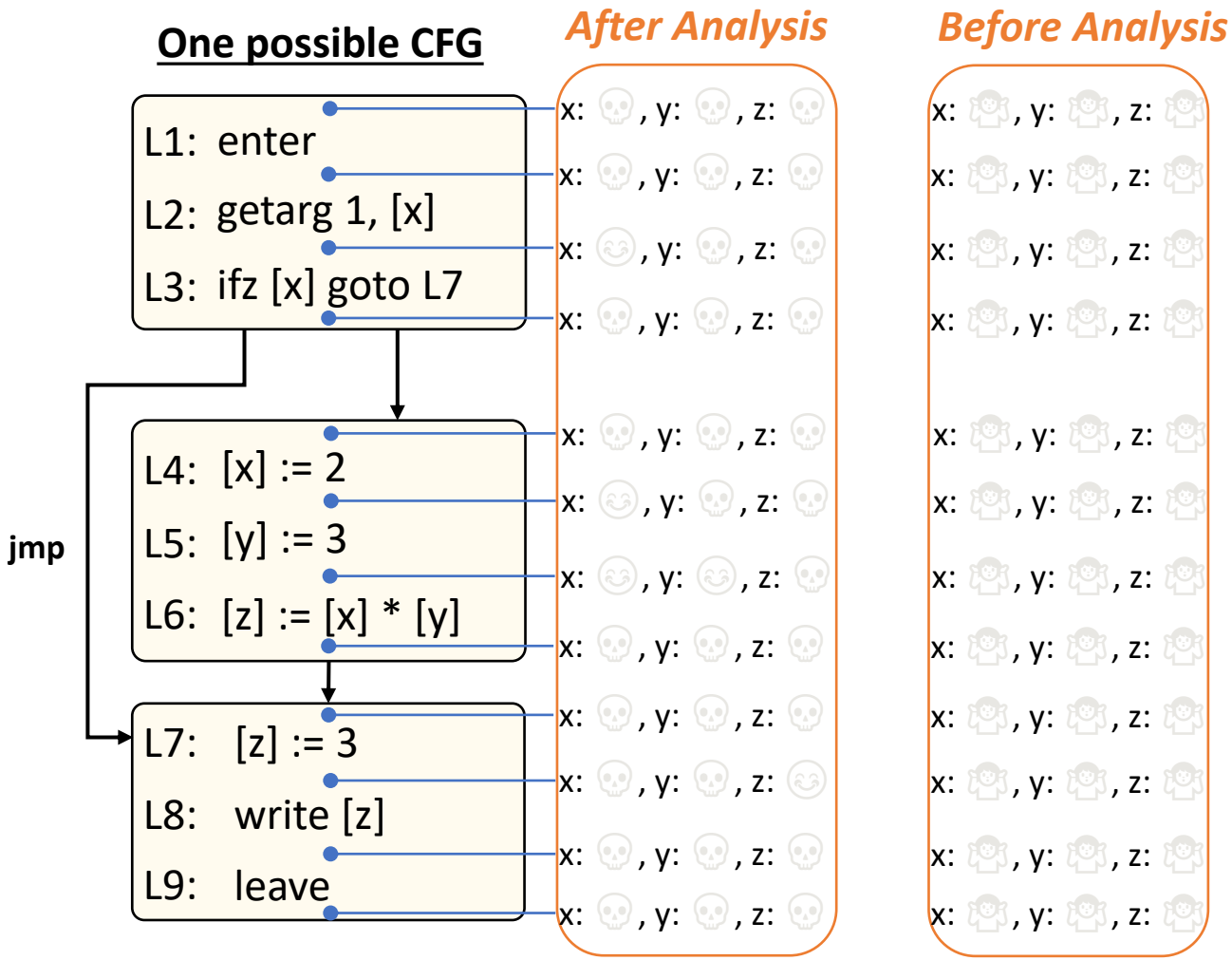
Generalizing Dataflow Intuition



Initializing Fact Sets

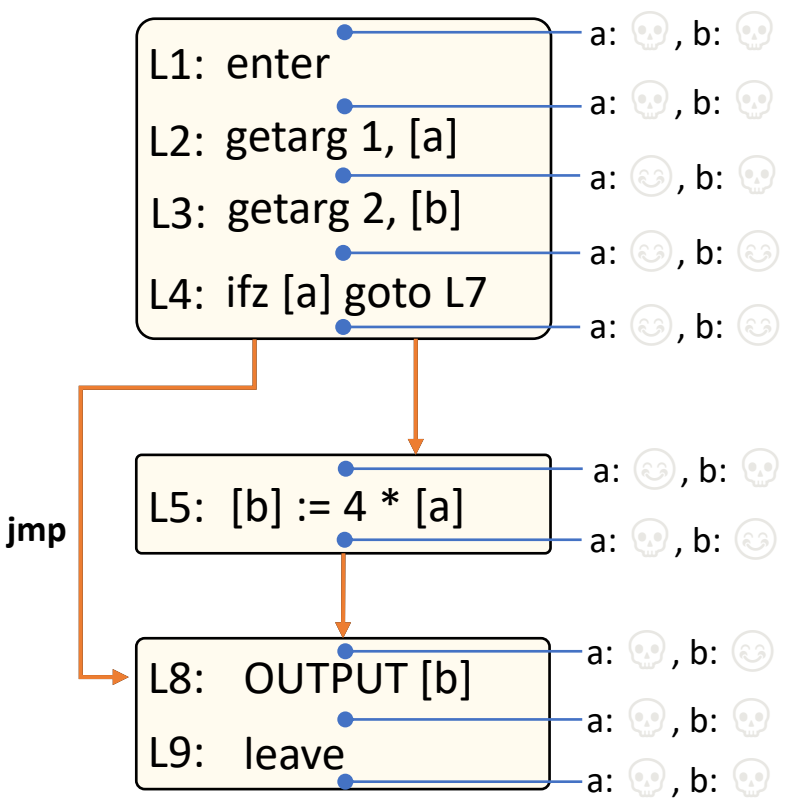
Dataflow Intuition

Technically, we should start all fact sets as "Not enough info" (💀). This will matter later



Merging Fact Sets

Dataflow Intuition



Fact sets may be different when multiple successors/predecessors join

- Need to merge incoming fact sets

Merge as conservatively as possible

- Don't do anything without a guarantee!
- Plan for all possible flows

Example: is L3 live? *(consider both block paths)*

- L3 definition clobbered on the fallthrough branch (at L5)
- L3 definition not clobbered on the jump branch

Today's Outline

Dataflow

Dataflow analysis

- Intuition
- Dataflow frameworks
- Abstract Interpretation



Optimization

Harnessing Commonalities of Dataflow Analyses

Dataflow Frameworks

Basic algorithms for many dataflow analyses follow a common template with minor variations

- Idea: restate each analysis in terms of its variations
- Profit: reuse the same algorithm to get results



Harnessing Commonalities of Dataflow Analyses

Dataflow Frameworks

Basic algorithms for many dataflow analyses follow a common template with minor variations

- Idea: restate each analysis in terms of its variations
- Profit: reuse the same algorithm to get results

Variations

- What information is tracked
- How fact sets are merged
- The direction of the analysis

Templated Information Tracking

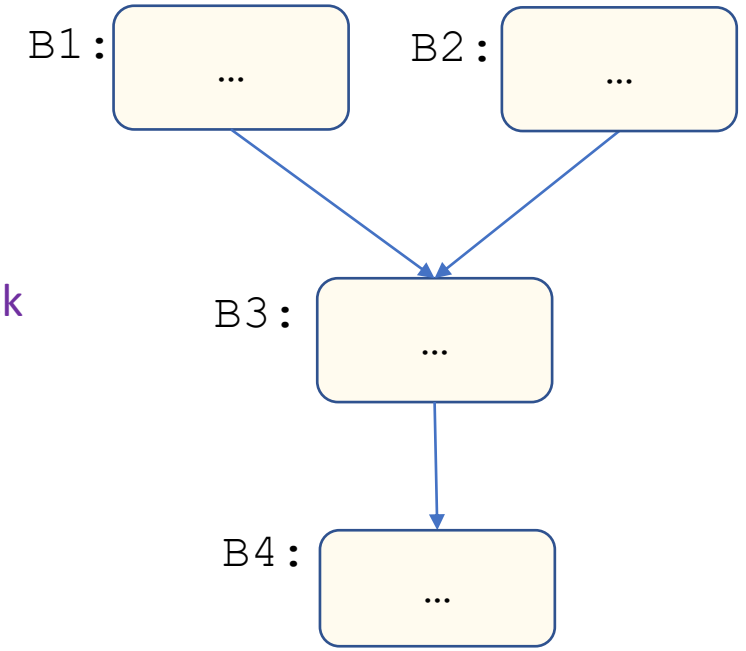
Dataflow Frameworks

Framework tracks the "interplay between data" at basic blocks boundaries

For a given basic block b:

- IN(b): facts true on entry to b
- OUT(b): facts true on exit from b
- GEN(b): facts created in b
- KILL(b): facts removed in b

For a *backwards analysis*
IN is at the bottom of the block
OUT is at the top of the block



$$\text{IN}(B) = \bigcup_{p \text{ in } pred(b)} \text{OUT}(p)$$

$$\text{OUT}(b) = \text{GEN}(b) \cup (\text{IN}(b) - \text{KILL}(b))$$

Dataflow Sets: Example

Dataflow: Formalization

IN(b): facts true on entry to b

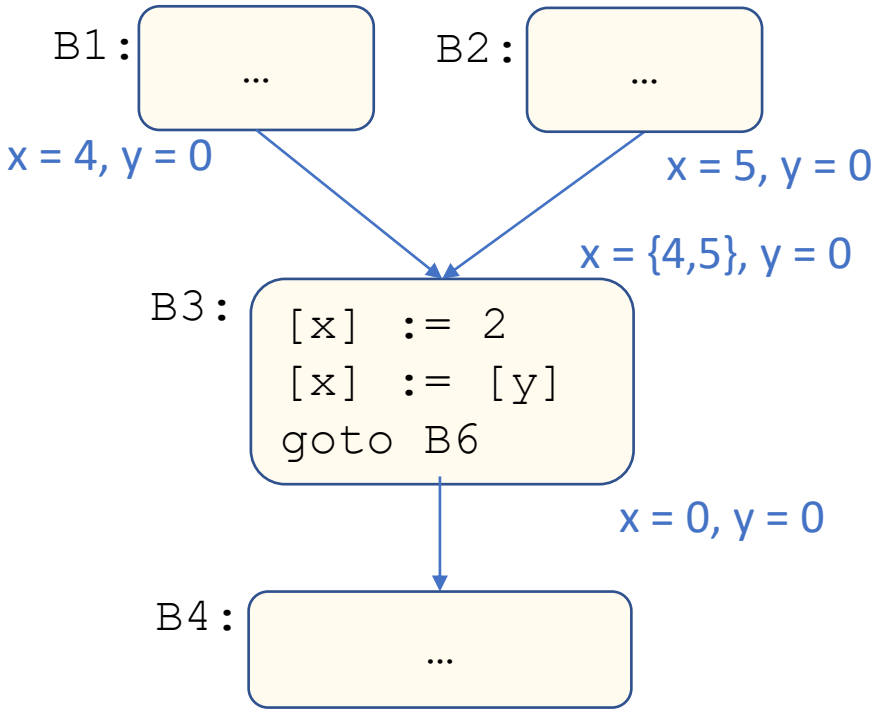
OUT(b): facts true on exit from b

GEN(b): facts created in b

KILL(b): facts removed in b

$$\text{IN}(B) = \bigcup_{p \text{ in } pred(b)} \text{OUT}(p)$$

$$\text{OUT}(b) = \text{GEN}(b) \cup (\text{IN}(b) - \text{KILL}(b))$$



Benefits of the Framework

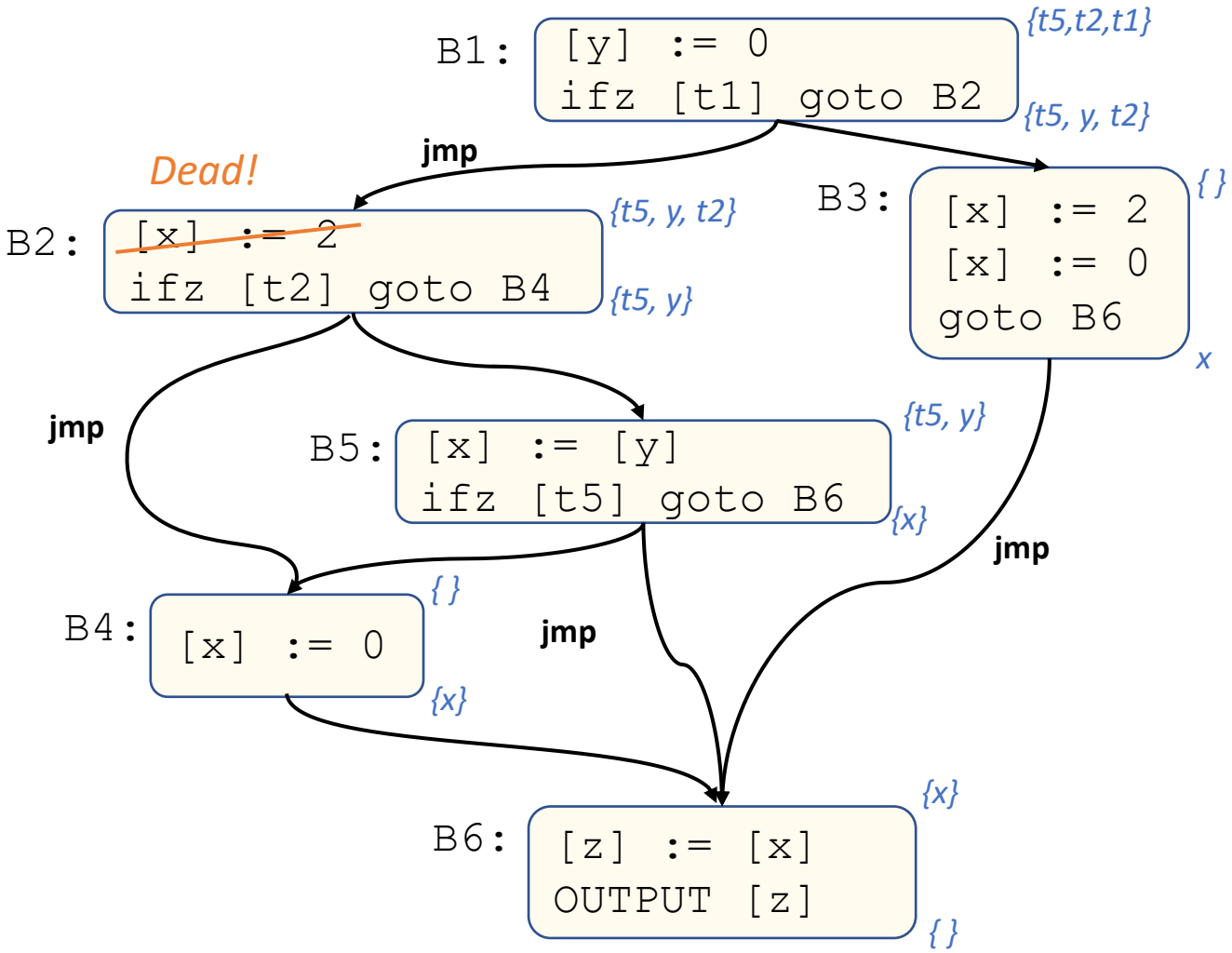
Dataflow Frameworks

When set up properly…

- Safety of the analysis is guaranteed
- Termination of the analysis is guaranteed
- Order of analysis (which block you process) is unimportant

Compute Live Variables

Dataflow: Formalization - Example



What values are live at B6?

Example Analyses

Dataflow: Formalization

Let's do some examples in this light

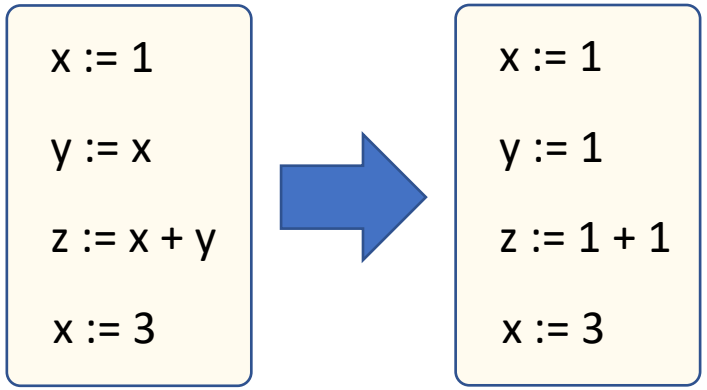
- ✔ A slightly bigger dead code elimination example
- Constant propagation
  - Recall: replace a variable with it's known constant value
  - Forward analysis
  - Fact sets: variable to (sets of) known values

Refresh Constant/Copy Propagation

Dataflow: Formalization

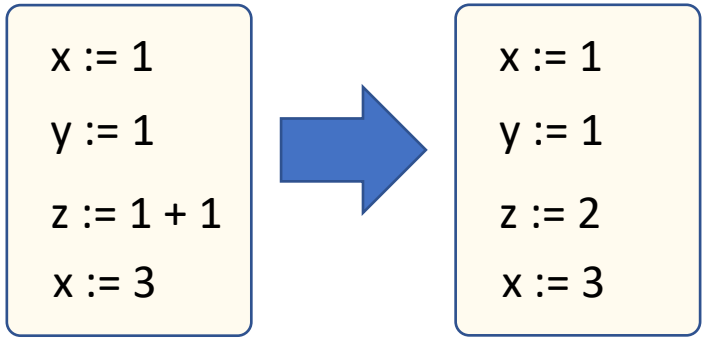
Copy Propagation

- Replace RHS of simple assigns with value of assign (if known)
- Forward analysis



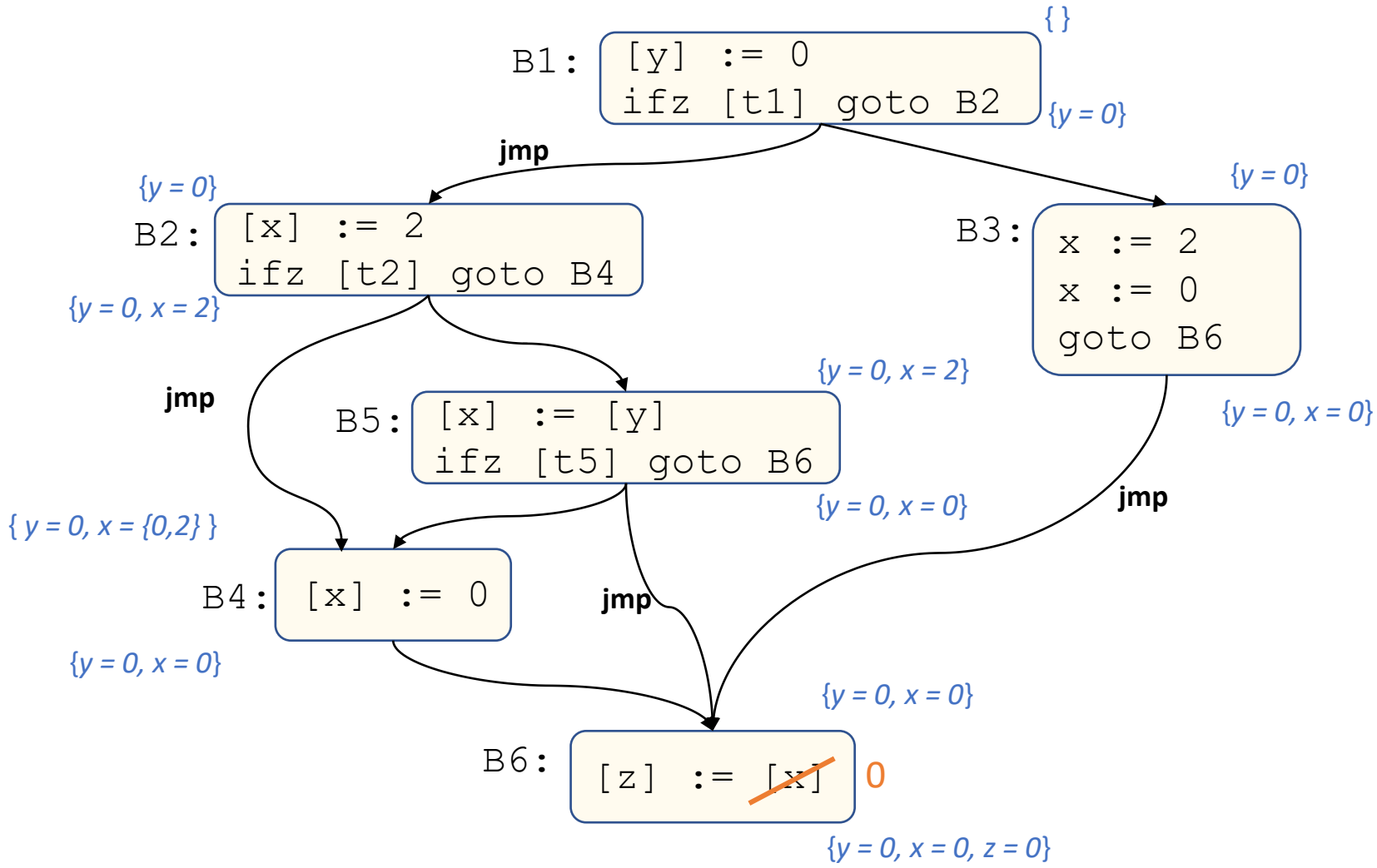
Constant folding

- Replace constant RHS expressions with value
- Traversal order isn't important



Example Constant Propagation

Dataflow: Formalization - Example



What values can x take on at B6?

Handling Practical Data Abstractions

Global Dataflow: Formalization

Global variables

- We only have visibility into 1 procedure
- Be conservative about the effect of other procedures
  - Reset fact sets across a call
  - Consider global variables live at function end

Analysis Termination

Dataflow: Formalization

In the previous examples, we completed in one pass over the CFG

- This won't always be the case, due to a fundamental construct…



Loops

Dataflow: Formalization

Loops complicate dataflow analysis

- Create cyclic dependencies
- Complicate fact sets



Oh bröther, you might have some lööps

Loops: Dependency cycles

Dataflow: Formalization

Solution: Saturate fact sets

- Start sets "TBD" ( ) value
- Run the algorithm until sets don't change

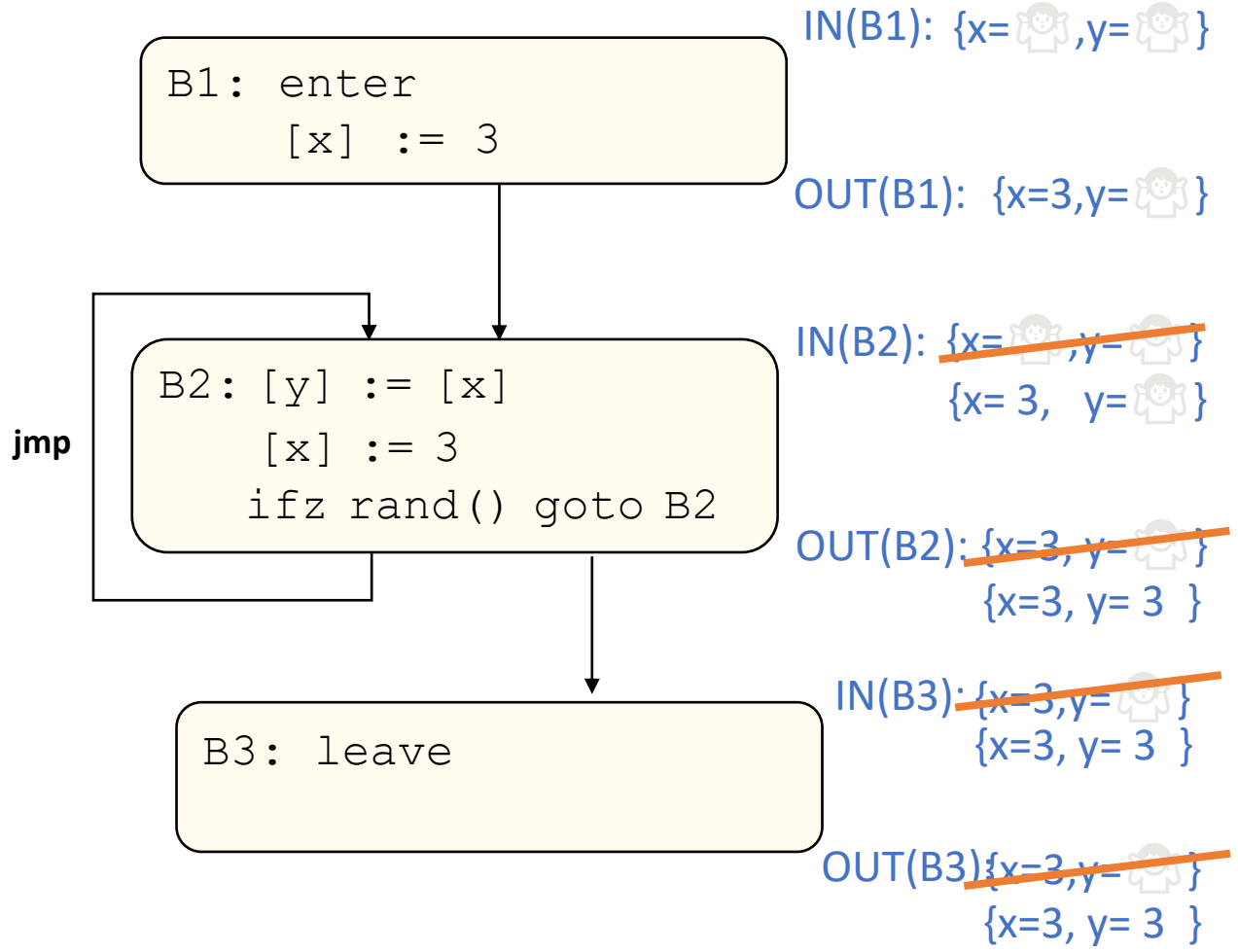
We've seen the saturation approach before

- (FIRST and FOLLOW sets)

Constant propagation

IN(B2) requires knowing OUT(B2)

OUT(B2) requires knowing IN(B2)



Summary

Underview

Covered some key optimization concepts

- Inter-block (global) analysis
- Dataflow frameworks:
  - Define fact sets and how they interact

Next Time – Static Single Assignment (SSA)

- A program form that eases and enhances optimization