Data-Flow Analysis II

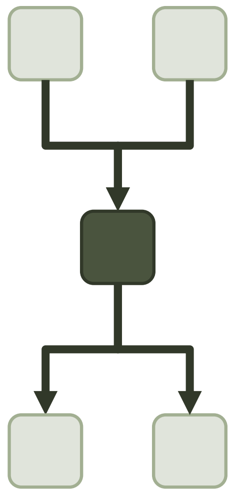
Data-Flow Analysis Review

Goal: Model program state along all program paths.

Concern: ***Undecidable***. Also, number of paths is exponential.

Approach:

- Consider subset of state (***data-flow value***).
- Reduce paths: $\text{IN}[b] = \bigwedge_{a \text{ precedes } b} \text{OUT}[a]$ (***meet operator***).
- Compute: $\text{OUT}[b] = f_b(\text{IN}[b])$ (***transfer function***).
- Necessarily approximate solution.



The Meet Operator and Its Domain

Property	Definition
Idempotent	$x \wedge x = x$
Commutative	$x \wedge y = y \wedge x$
Associative	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$

Element	Definition
Top (⊤)	$\forall x. \top \wedge x = x$
Bottom (⊥)	$\forall x. \bot \wedge x = \bot$

The Meet Operator and Its Domain

Property	Definition
Idempotent	$x \wedge x = x$
Commutative	$x \wedge y = y \wedge x$
Associative	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$

Implementation detail

	Definition
Top (⊤)	$\forall x.\ \top \wedge x = x$
Bottom (⊥)	$\forall x.\ \bot \wedge x = \bot$

Needed for termination

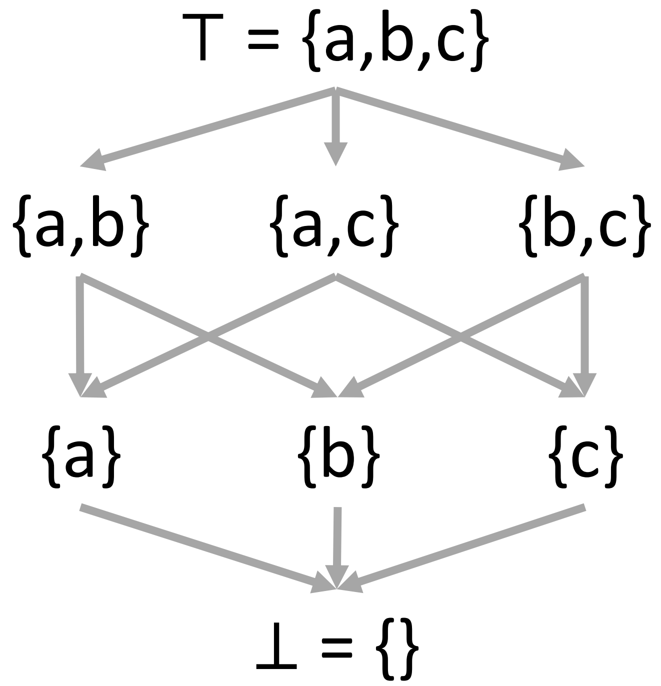
Meet Semilattices

We can define a partial order:

- Reflexive, antisymmetric, transitive.
- $x \leq y \equiv x \wedge y = x$

Greatest Lower Bound (glb)

- $glb(x, y) = x \wedge y$



$$x \subseteq y \equiv x \cap y = x$$

Transfer Functions

Property	Definition
Identity Function	$\exists I \in F. \forall x \in V. I(x) = x$
Closed under Composition	$\forall f, g \in F. h(x) = g\big(f(x)\big) \Rightarrow h \in F$

Monotone (1)	$\forall x, y \in V. \forall f \in F$ $f(x \wedge y) \leq f(x) \wedge f(y)$
Monotone (2)	$\forall x, y \in V. \forall f \in F$ $x \leq y \Rightarrow f(x) \leq f(y)$

Transfer Functions

Property	Definition
Identity Function	$\exists I \in F. \forall x \in V. I(x) = x$
Closed under Composition	$\forall f, g \in F. h(x) = g\big(f(x)\big) \Rightarrow h \in F$

Monotone (1)	$\forall x, y \in V. \forall f \in F$ $f(x \wedge y) \leq f(x) \wedge f(y)$
Monotone (2)	$\forall x, y \in V. \forall f \in F$ $x \leq y \Rightarrow f(x) \leq f(y)$

Needed for termination

Statements vs. Basic Blocks

We often define transfer functions for ***statements*** instead of ***basic blocks***.

- If basic block $B = \langle s_1, s_2, \dots s_n \rangle$, then $f_B = f_{s_n} \circ \cdots \circ f_{s_2} \circ f_{s_1}$

Data-flow analysis does not require ***maximal*** blocks.

- Same result if each block is one statement.

Basic blocks are an ***optimization***: fewer nodes in the graph.

Forward Data-Flow Algorithm

Given:

- V: values of lattice
- $\wedge$: meet operator
- F: set of transfer functions
- CFG with unique ENTRY and EXIT nodes
- v_{ENTRY}: data-flow value for ENTRY node

For each block b, OUT[b] = T

OUT[ENTRY] = v_{ENTRY}

While any OUT changes

  For each block b except ENTRY

    IN[b] = $\wedge_a$ OUT[a]

    OUT[b] = f_b(IN[b])

Backward Data-Flow Algorithm

Given:

- V: values of lattice
- $\wedge$: meet operator
- F: set of transfer functions
- CFG with unique ENTRY and EXIT nodes
- v_{EXIT}: data-flow value for EXIT node

For each block b, IN[b] = T

IN[EXIT] = v_{EXIT}

While any IN changes

  For each block b except EXIT

    OUT[b] = $\wedge_c$ IN[c]

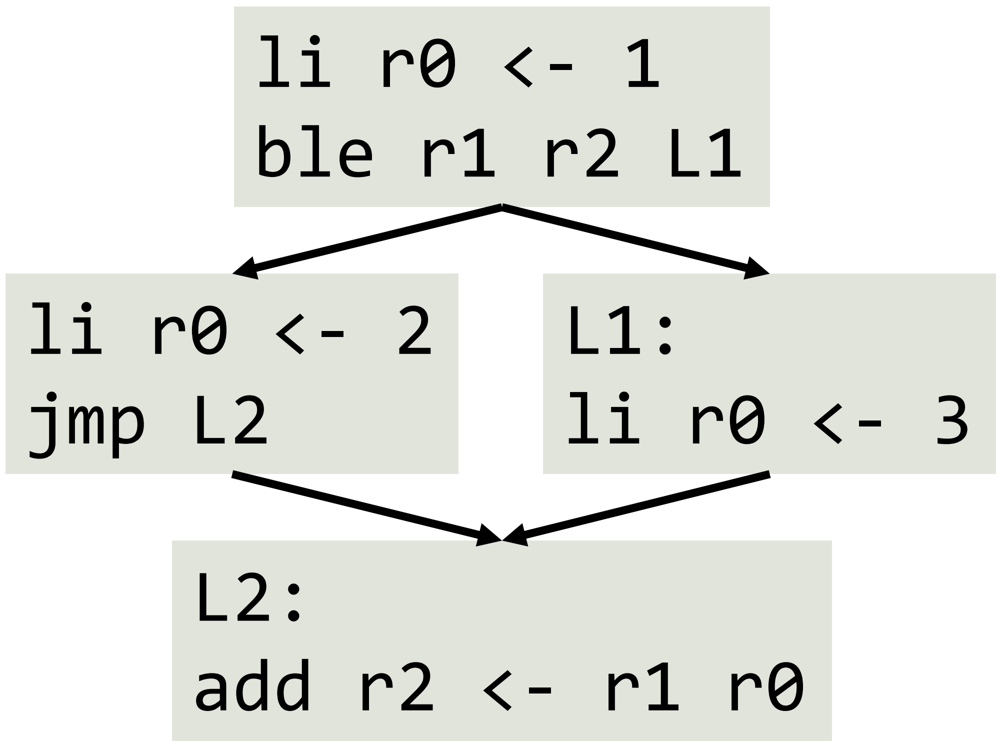
    IN[b] = f_b(OUT[b])

Live Variable Analysis

Goal: Determine range of statements in which a value may be needed.

Used in:

- Dead code elimination.
- Register allocation.



Live Variable Analysis

Direction: Backward

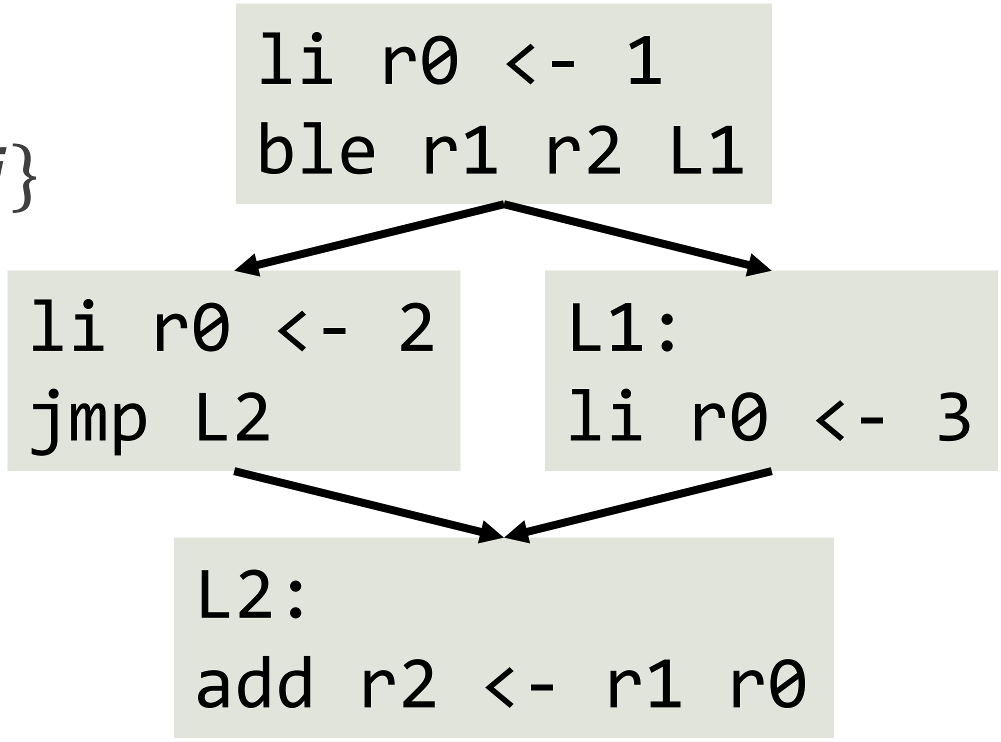
Values: Set of live locations.

- $V \subseteq \{\mathrm{r}i | 0 \leq i \leq 7\} \cup \{\mathrm{sp}[j] | 0 \leq j\}$

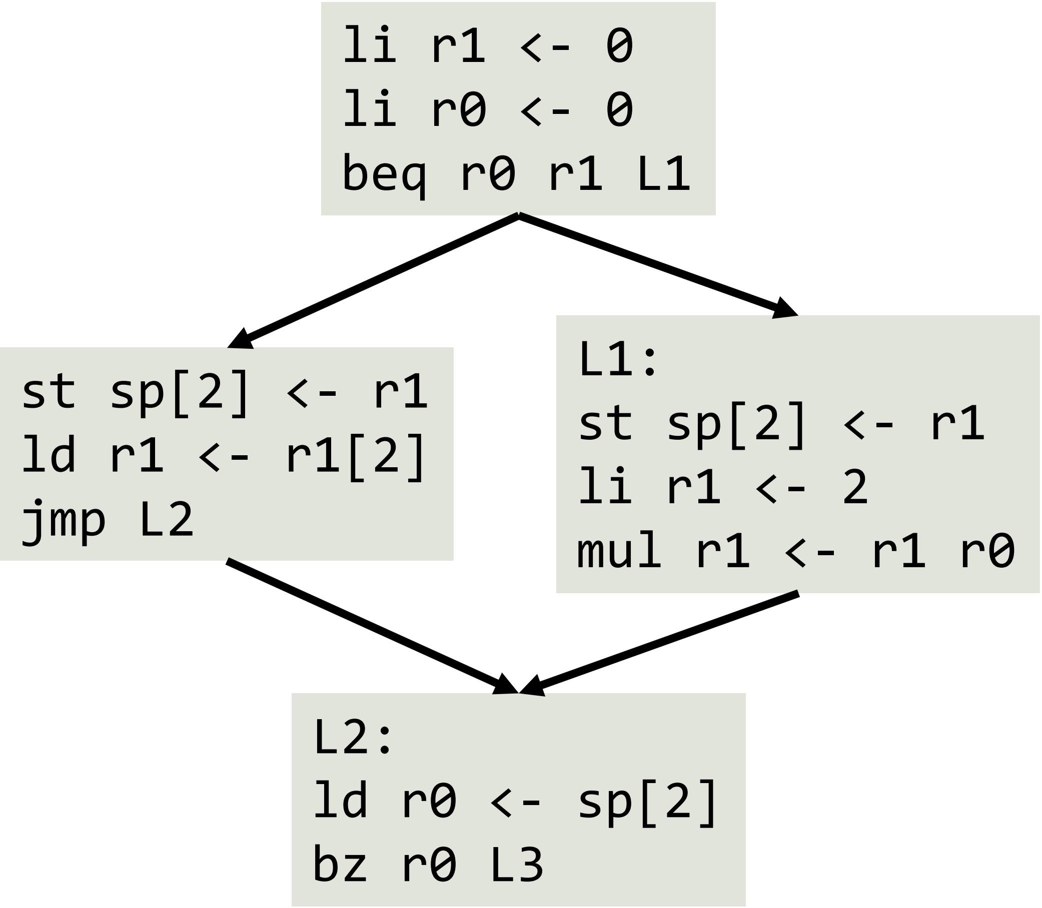
Meet operator: set union

Transfer functions:

- `op ra <- rb rc`
- $f(x) = \{\mathrm{rb}, \mathrm{rc}\} \cup (x - \{\mathrm{ra}\})$



Constant Propagation



Direction: Forward

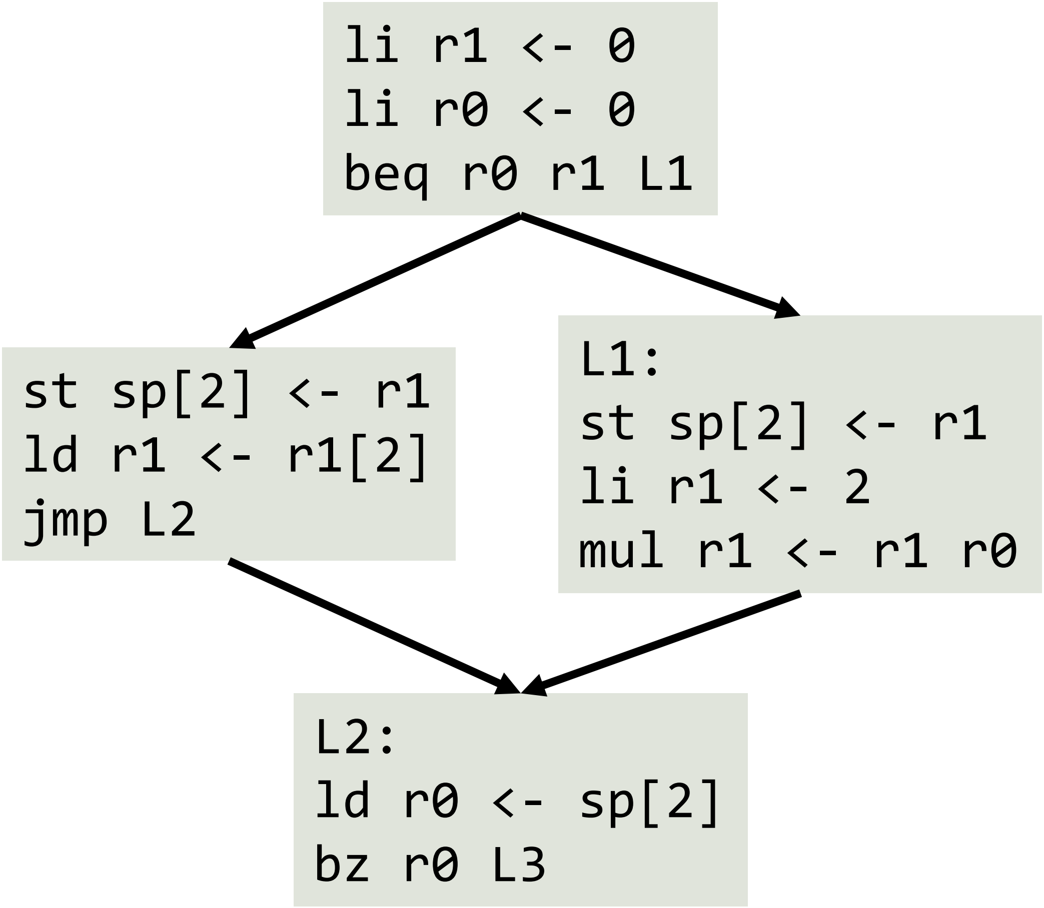
Values:

- $\langle$r0, r1, … r7, sp[j]…$\rangle$
- $v_i \in \{\top(\text{unknown}), \bot(\text{nac})\} \cup \mathbb{Z}$

Meet operator:

- $\langle \dots, x_i, \dots \rangle \wedge \langle \dots, y_i, \dots \rangle =$
  - Usual rules for ⊤ and ⊥
  - c if $x_i = y_i = c$
  - ⊥ otherwise

Constant Propagation

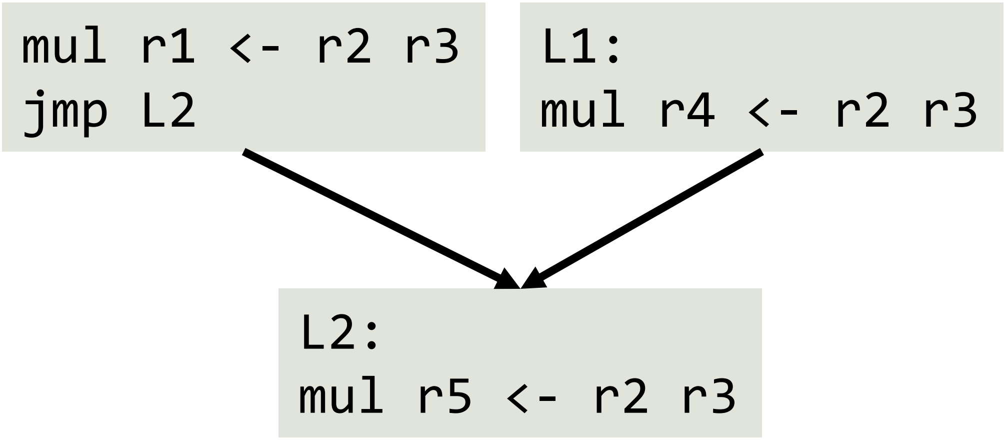


Transfer Functions:

Statement	Value
li ri <- c	?
ld ri <- sp[j]	?
st sp[i] <- rj	?
mul ra <- rb rc	?
call ri	?

Redundant Expressions

Global Common Expressions



Redundant Expressions

Global Common Expressions

```
mul r1 <- r2 r3
jmp L2
```

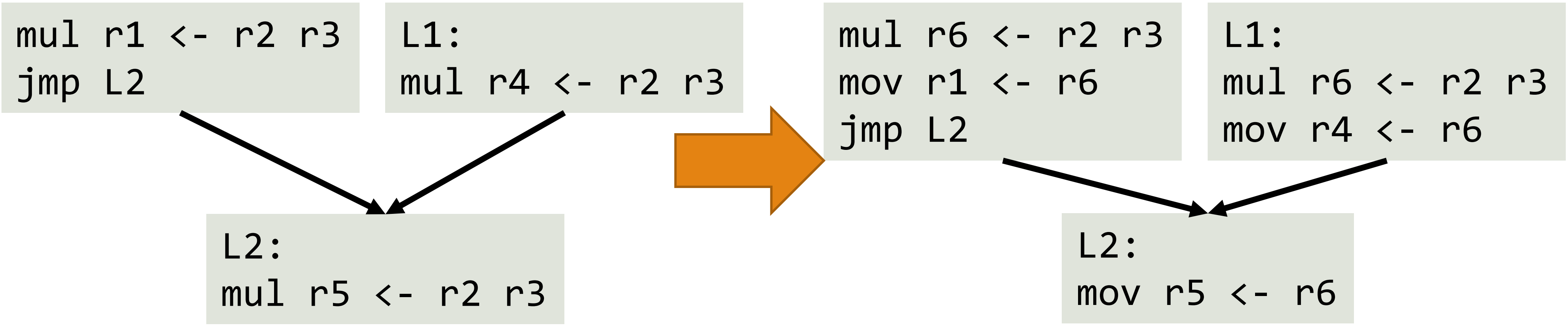
```
L1:
mul r4 <- r2 r3
```

```
L2:
mul r5 <- r2 r3
```

r2*r3 is already computed on every path

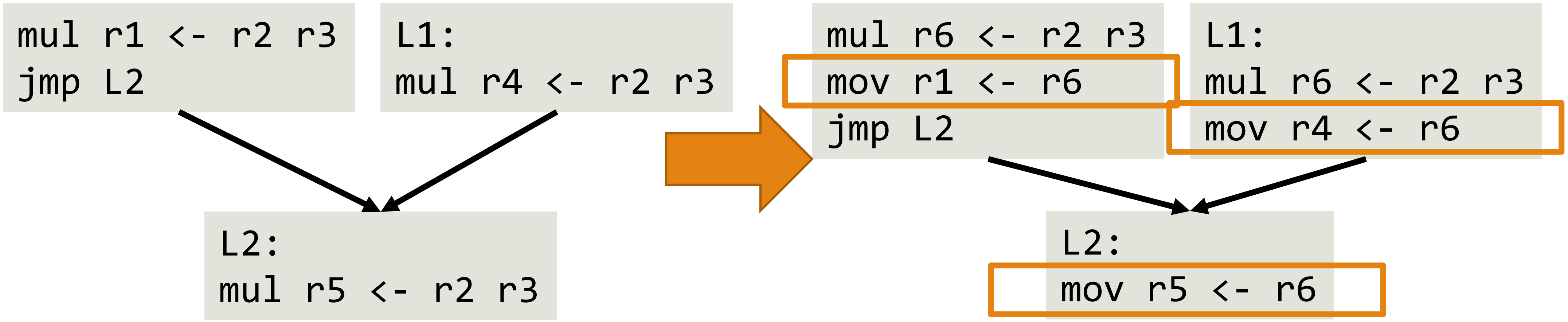
Redundant Expressions

Global Common Expressions



Redundant Expressions

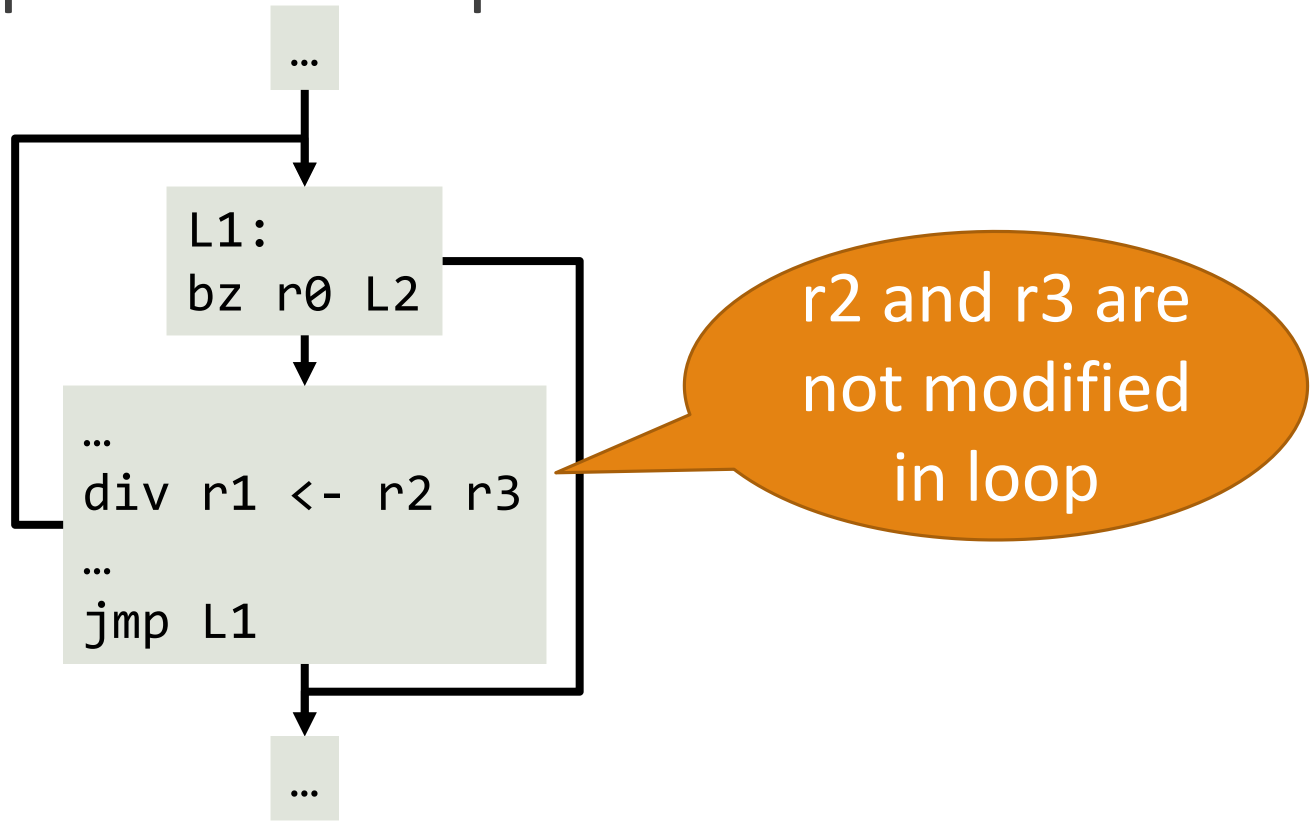
Global Common Expressions



Clean up with copy propagation.

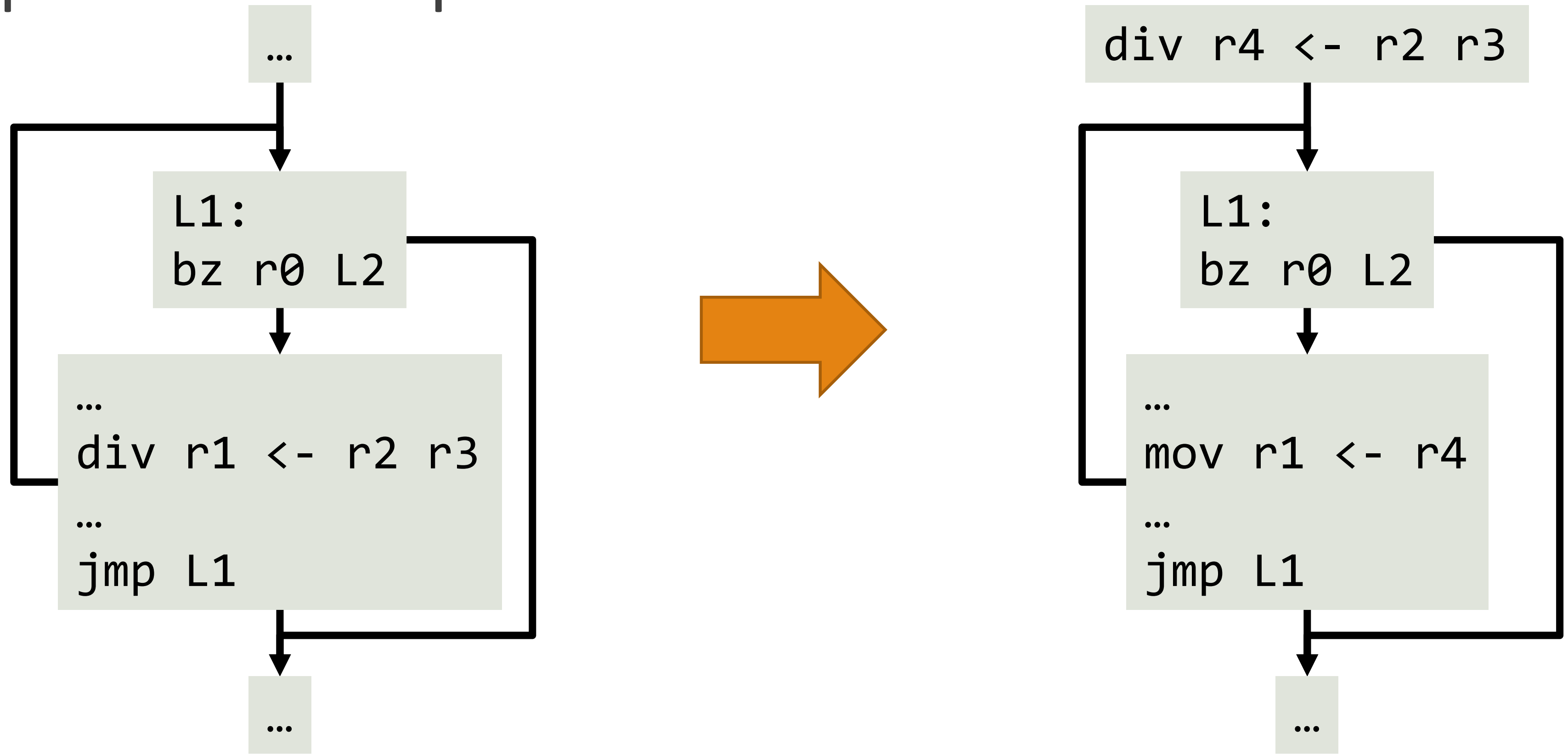
Redundant Expressions

Loop Invariant Expressions



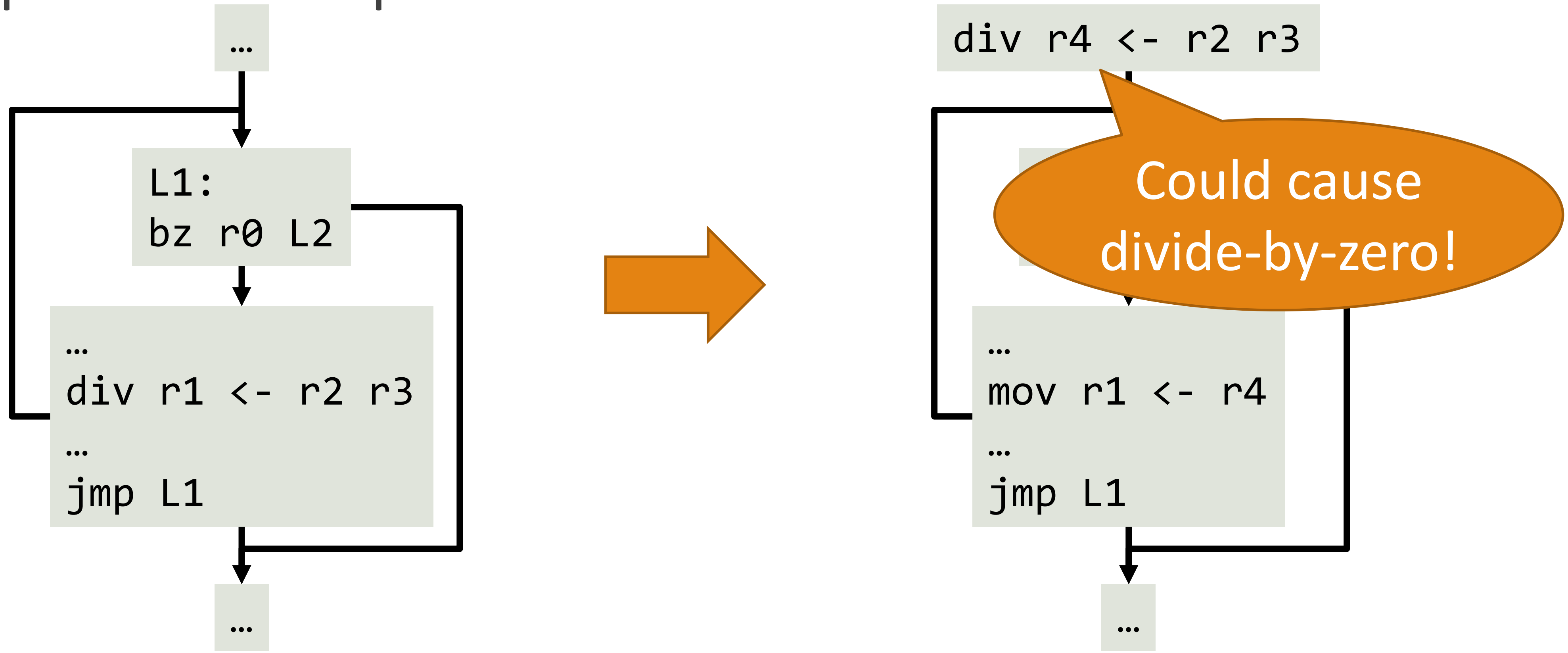
Redundant Expressions

Loop Invariant Expressions



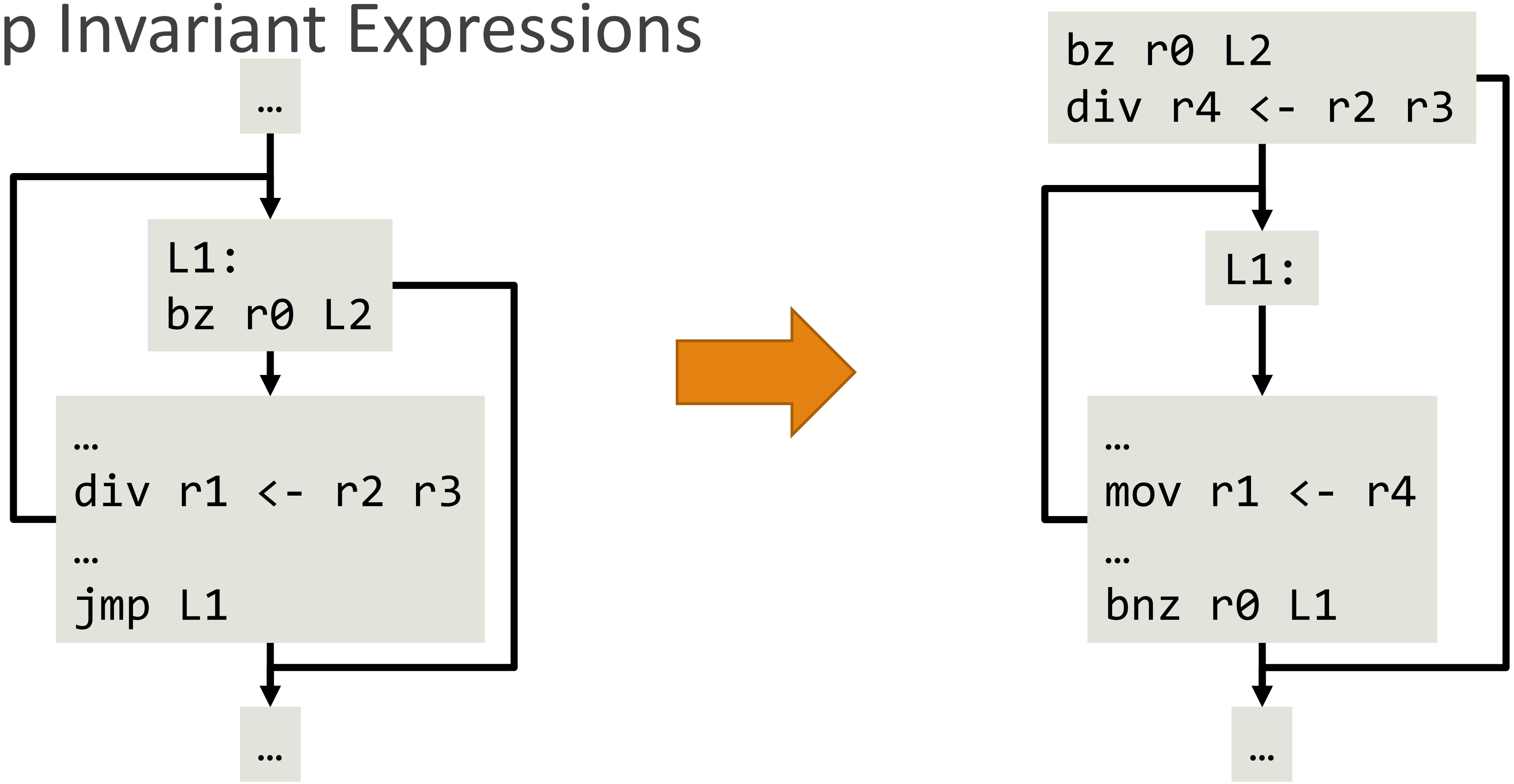
Redundant Expressions

Loop Invariant Expressions



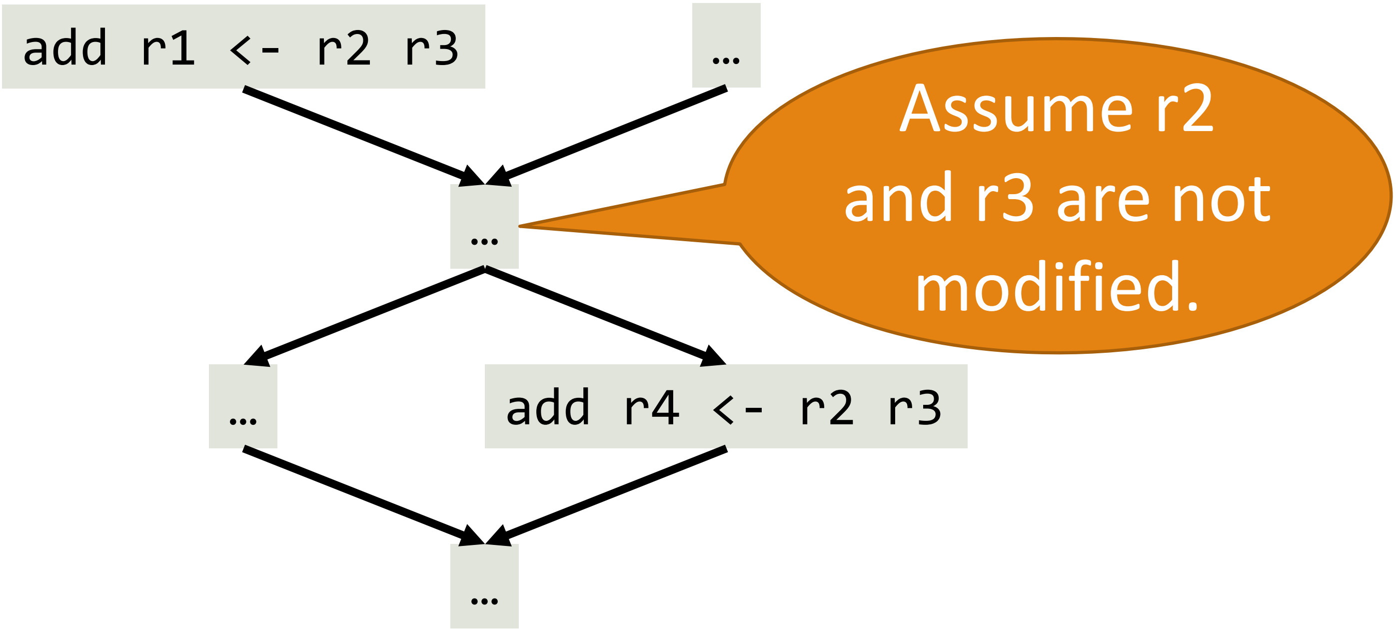
Redundant Expressions

Loop Invariant Expressions



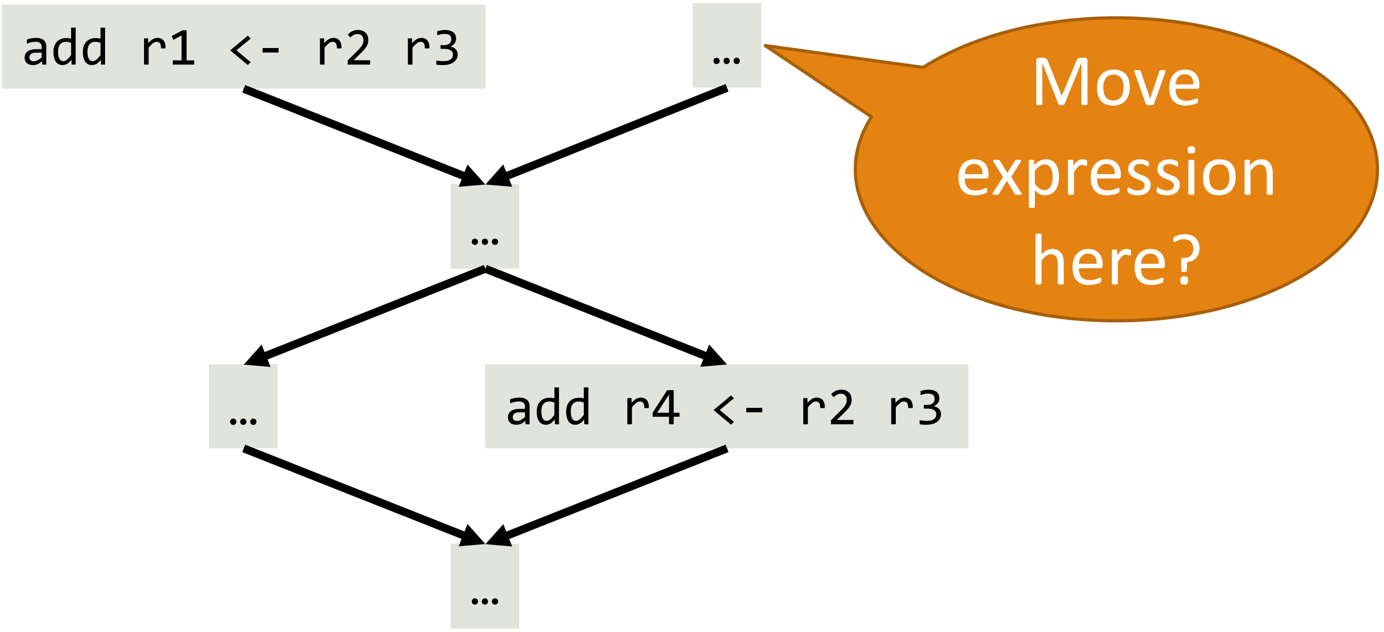
Redundant Expressions

Partially Redundant Expressions



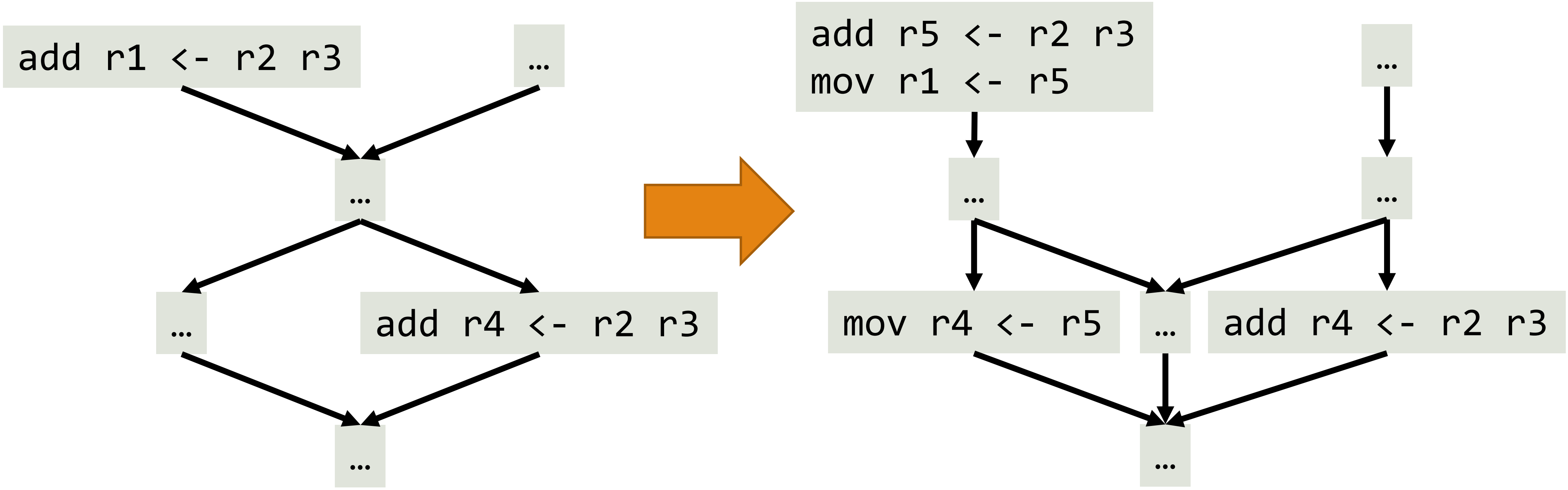
Redundant Expressions

Partially Redundant Expressions



Redundant Expressions

Partially Redundant Expressions



Code Motion and Debugging

We are changing the order of evaluation.

- "Don't break the build" – all ***valid runs must still be valid***.
- Evaluate expressions ***only if*** the naïve code would.

What about reordering invalid runs?

- E.g., an exception gets moved after database update.
- Need to ***maintain sequence of user-visible state changes***.

This is why debugging optimized code is not always obvious.

Lazy Code Motion

1. Find anticipated expressions at each program point *p*.
    - I.e., all *e* such that all paths from *p* eventually compute *e*.
2. Determine available expressions at each point *p*.
3. Postpone expressions as long as possible.
4. Eliminate unused temporaries.

Anticipated Expressions

Direction: Backward

Values: Sets of expressions

Meet operator: $\cap$

$v_{\text{EXIT}} = \{\}$

Transfer function:

- $f_b(x) = use_b \cup (x - kill_b)$

Use set:

- $use_b = \{e | e \text{ is computed in } b\}$

Kill set:

- $kill_b = \{e \mid \exists x \,.\, isop(x, e) \wedge def(x, b)\}$

Available Expressions

Direction: Forward

Values: Sets of expressions

Meet operator: ∩

v_{ENTRY} = {}

Transfer function:

- $f_b(x) = available[b] - kill_b$

After this analysis, insert expressions at points where the expression is first anticipated.

Postponable Expressions

Direction: Forward

Values: Sets of expressions

Meet operator: $\cap$

$v_{\text{ENTRY}} = \{\}$

Transfer function:

- $f_b = (earliest[b] \cup x) - use_b$

$$earliest[b] = anticipated[b] - available[b]$$

Used Expressions

Direction: Backward

Values: Sets of expressions

Meet operator: ∪

v_{EXIT} = {}

Transfer function:

- $f_b(x) = (use_b \cup x) - latest[b]$