252-210: Compiler Design

Again: Busy expressions

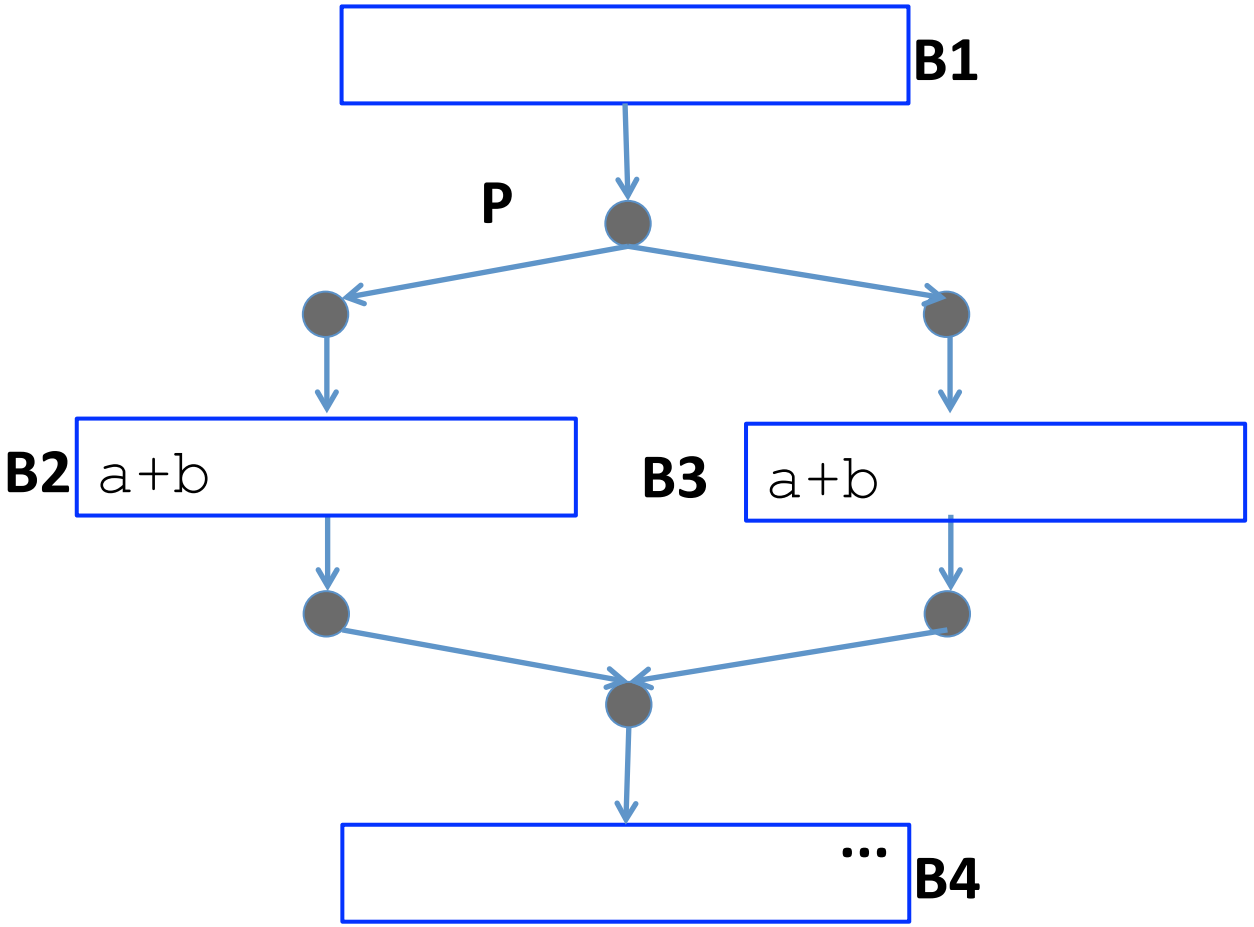
Thomas R. Gross

Computer Science Department
ETH Zurich, Switzerland

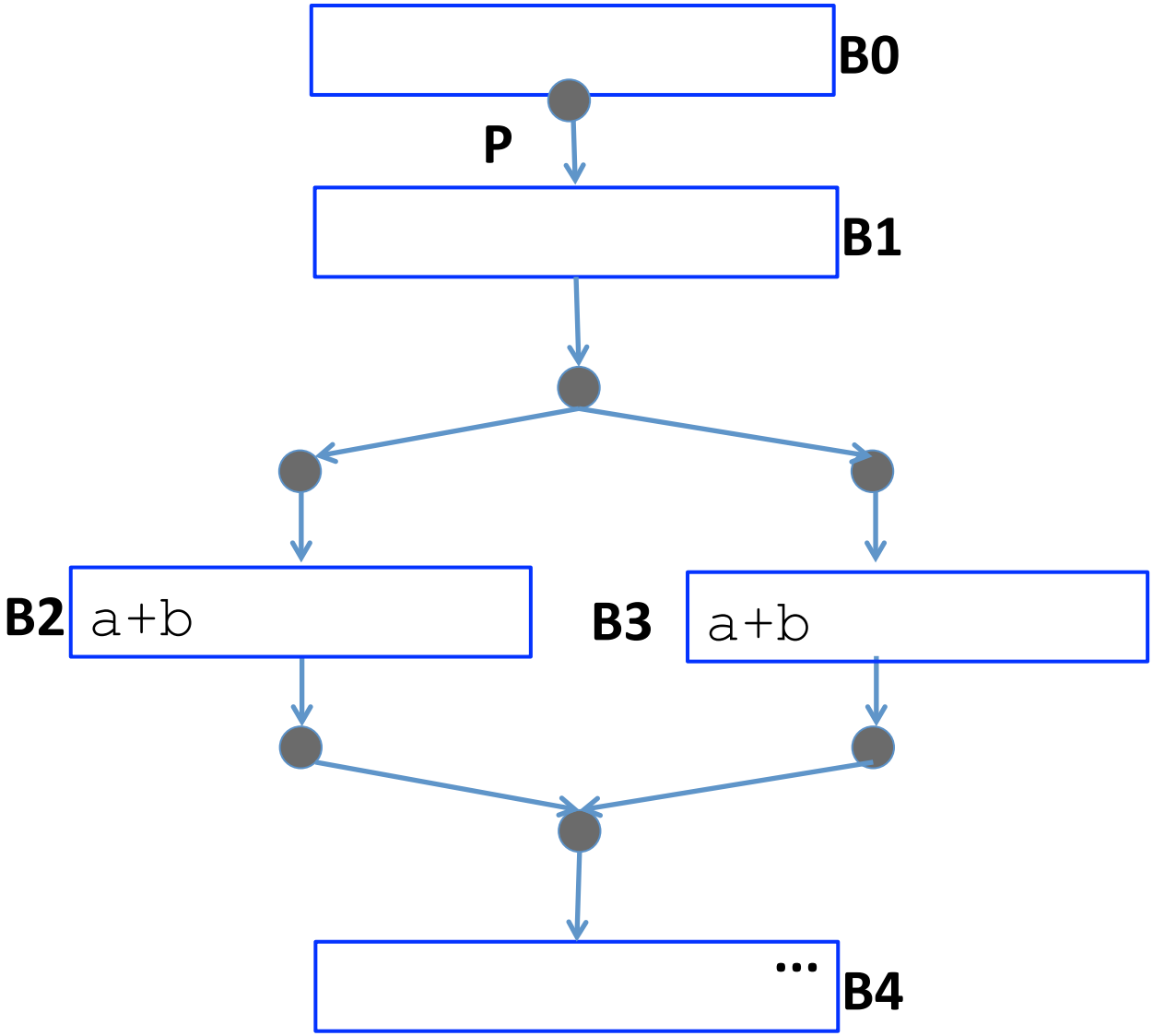
- **Idea: an expression E is *very busy* at Point P if no matter what path is taken from P, the expression E is evaluated before any of its operands are defined.**

- **Idea: an expression E is very busy at Point P if no matter what path is taken from P, the expression E is evaluated before any of its operands are defined.**
- **An expression `a+b` is *very busy* at a point P if `a+b` is evaluated on all paths from P to EXIT and there is no definition of `a` or `b` on a path between P and an evaluation of `a+b`**
  - Interested in set of expressions available at the start of a basic block B
  - Set depends on paths that start at P_{before_B}

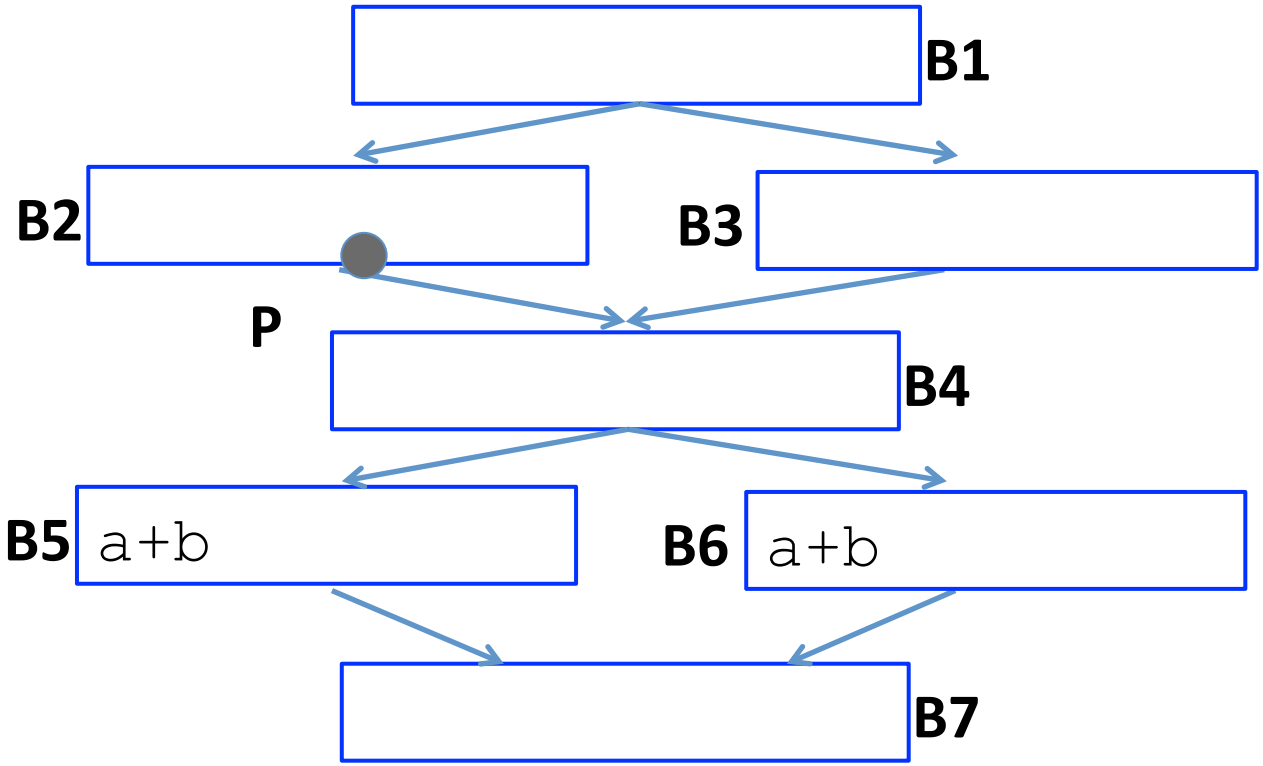
Examples



Examples



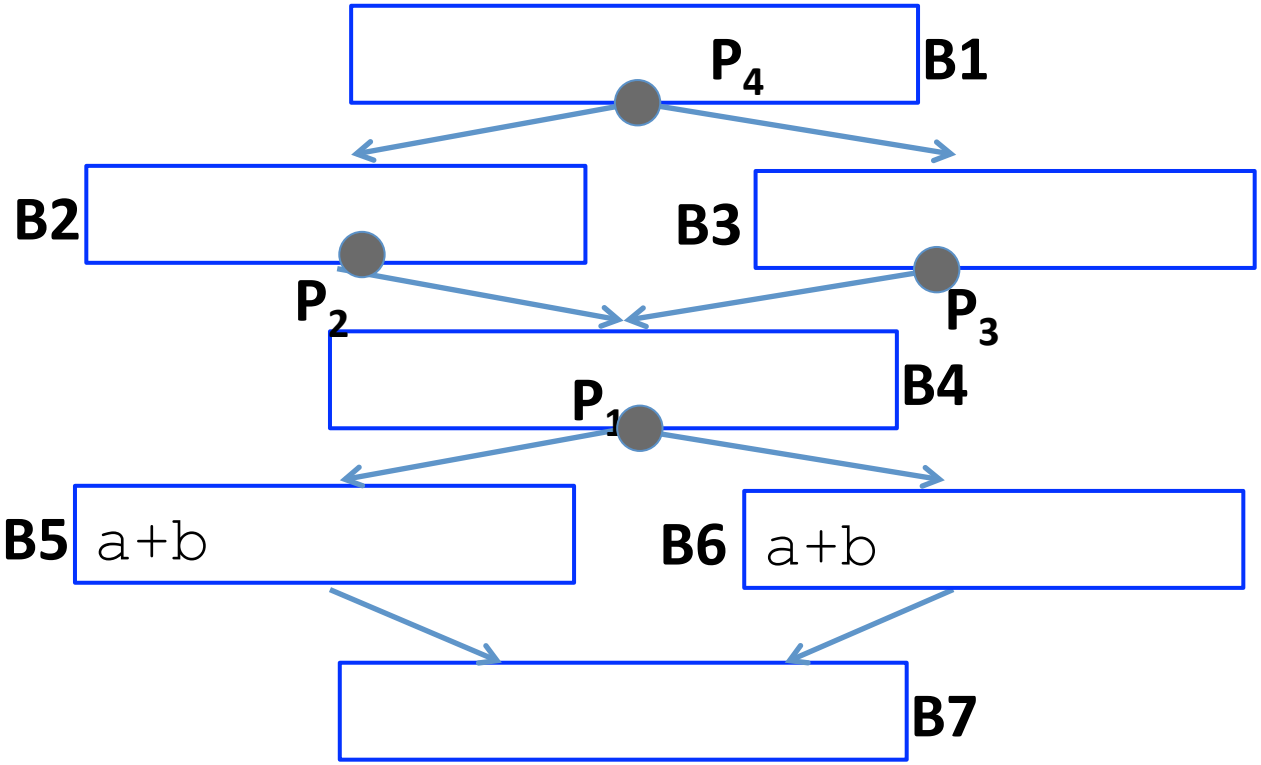
Examples



Very busy expressions

- An expression E can be very busy at many points P_1, P_2, P_3,

Examples

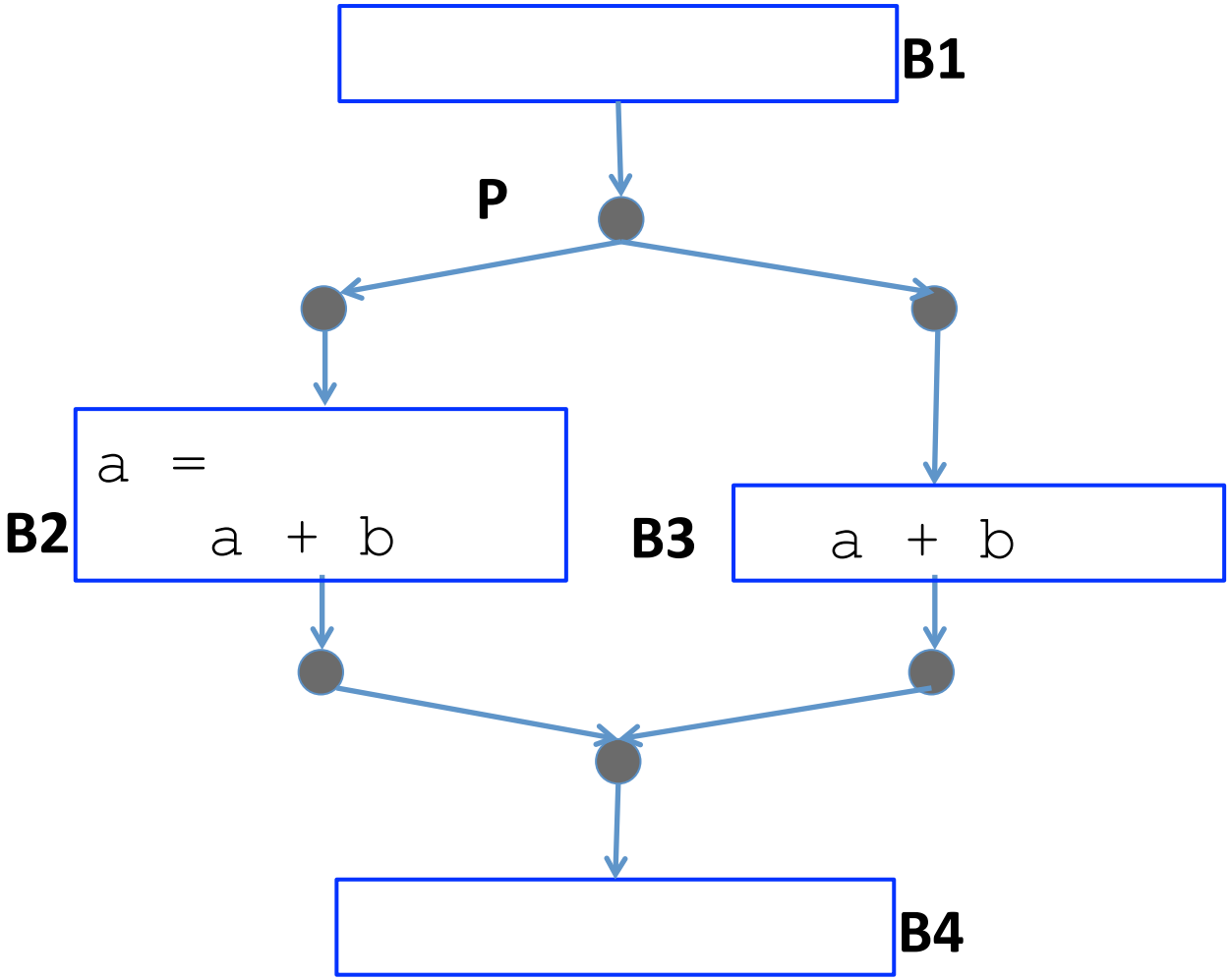


Very busy expressions

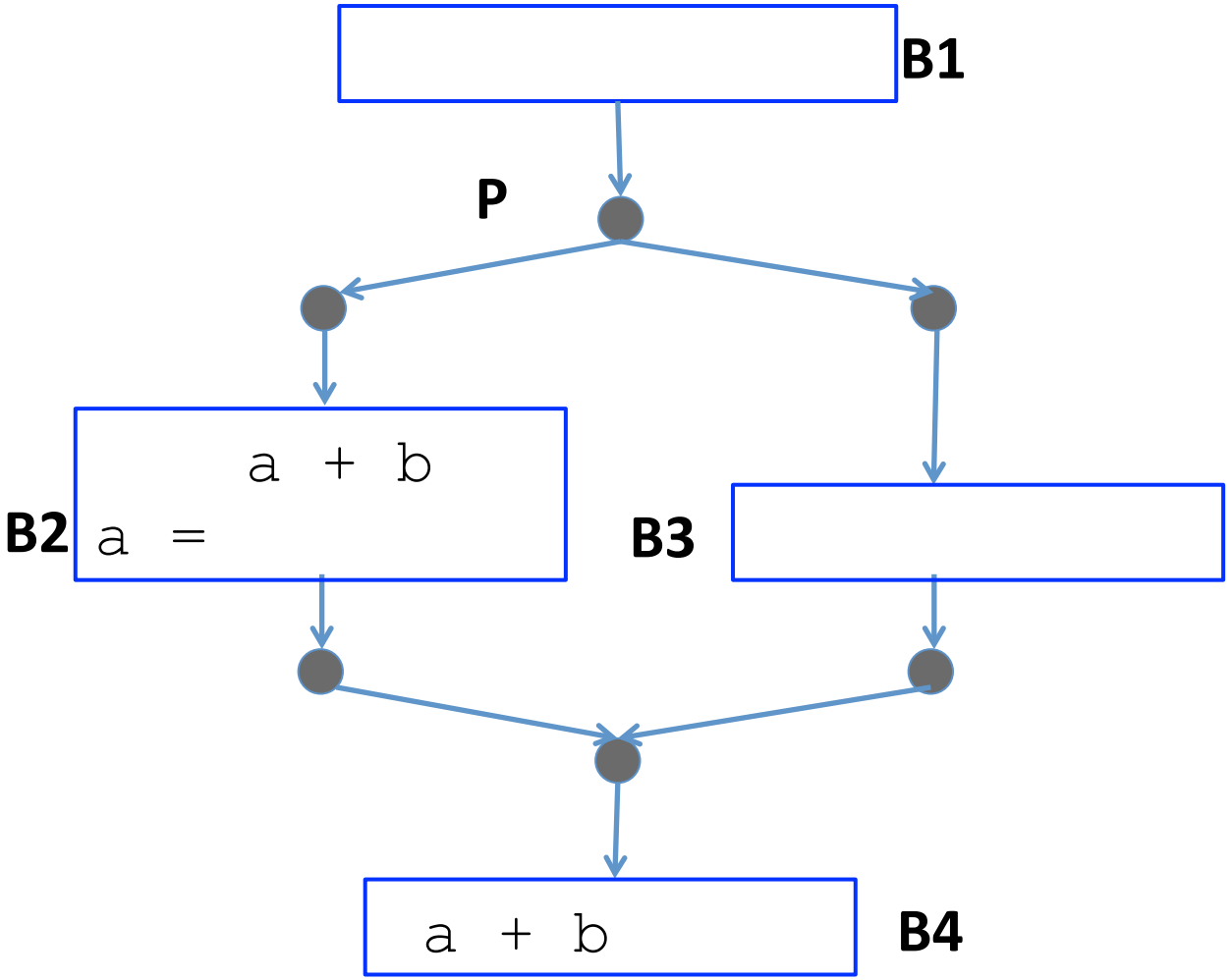
- **An expression E can be very busy at many points P_1, P_2, P_3, …**

- **Consider the point P_i that is *not* dominated by any other P_j as the target point for hoisting expression E**
  - Intuitively gives you the “closest” point P

Examples



Examples



`a + b` very busy at P?

- **Path from P to E such that operand (a) is modified**

Very busy ≠ hoistable

- **Need additional conditions that must be met if you want to hoist a very busy expression**
  - Same definitions reach P that reach occurrences of E that are hoisted
  - No definition of an operand along any path from P to E

Data flow vs. optimization

- **Data flow collects program properties**
- **Optimizations transform program**

- **Example**
  - Assignment statement `x = expression` in a loop
  - expression always evaluates to the same value

  - Question: can we hoist the assignment statement out of the loop?

```
while (…)  {
 …
 x = expr;
 …
}
```

```
x = expr;
while (…)  {
 …
}
```

- **Maybe the loop is never executed**
  - x has different value
- **Maybe x is never read outside the loop**
  - Don't care about about value of x but may lengthen execution
- **Maybe x is read in loop prior to assignment**
- **Maybe assignment is in a conditional statement**

```
while (…)  {
    …
    if (…) {x = expr; }
 …
}
```

- **Often cheaper to prune expressions from list of very busy expressions than to make global data flow analysis complicated**

Finding IN(B) and OUT(B)

- **N basic blocks, 2×N sets IN / OUT**
- **System with 2×N unknowns**
  - Solve by iterating until a fixed point is found

- **How to start iteration?**

  Safe assumption IN[EXIT] = $\varnothing$

  Nothing is very busy at the end

- **IN(B) = $\mathcal{U}$**
  - $\mathcal{U}$ set of all expressions in program
  - For all B ≠ EXIT

Computing very busy expressions

IN[EXIT] = $\varnothing$

Initialize IN[B] = $\mathcal{U}$ for $\forall$ B ≠ EXIT

while (changes to any IN(B)) {

  for (each basic block B ≠ EXIT) {

    OUT(B) = $\cap_{\text{Bi, Bi is successor of B in CFG}}$ IN(Bi)

    IN(B) = $gen_B \cup (OUT(B) - kill_B)$

  }

}

Greek

- αααααβδερργλλλμμμννννωωω
- •
- ⊤

∩