Symbolic Evaluation

symbolic evaluation

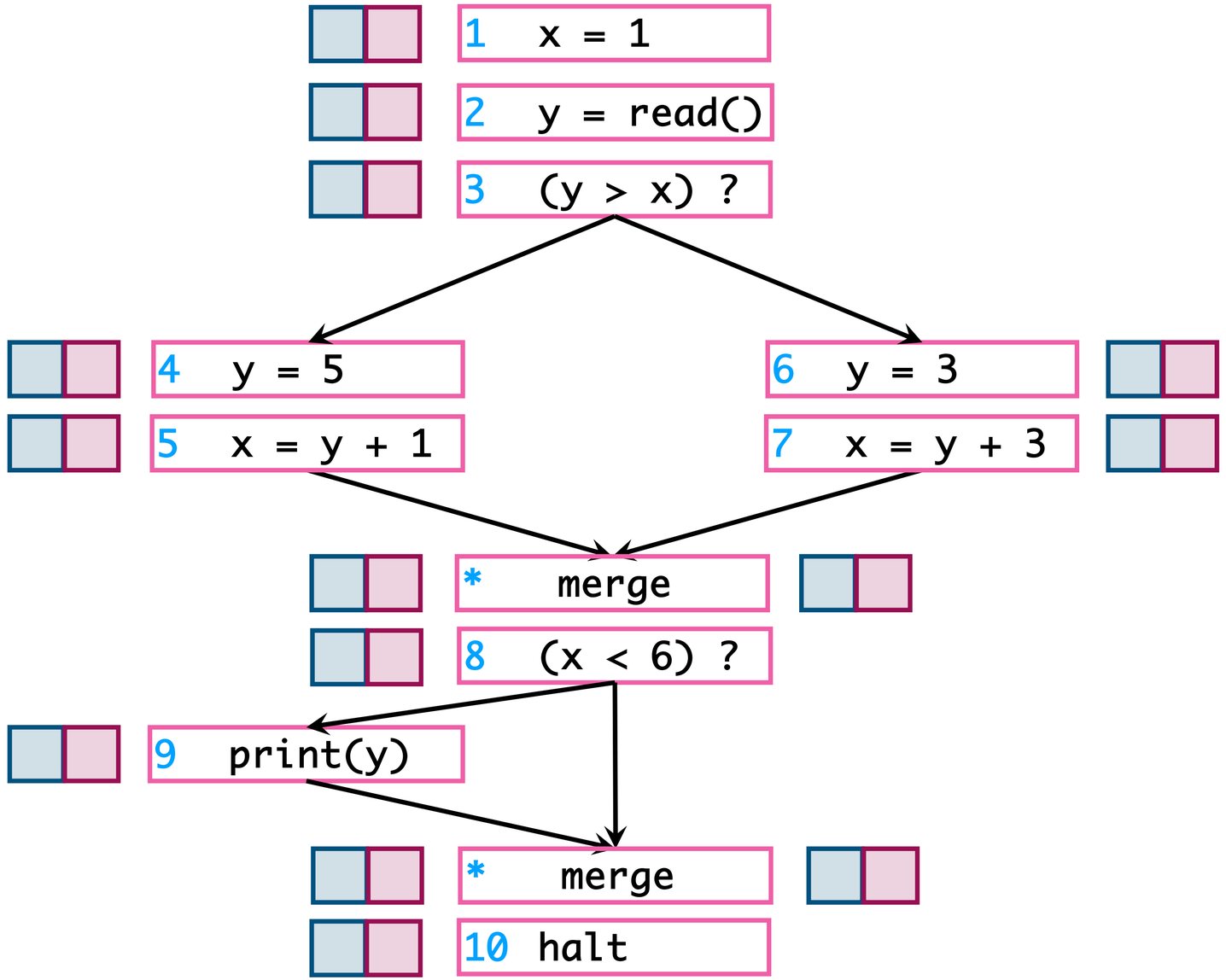
- Introduce *symbolic* values for each variable at each program point

⊥ No information about this variable

v Some constant value v (a particular constant)

⊤ Definitely not a constant

- Before execution begins, have no information (except will assume that variables are definitely not constants at the beginning of the program)



symbolic evaluation

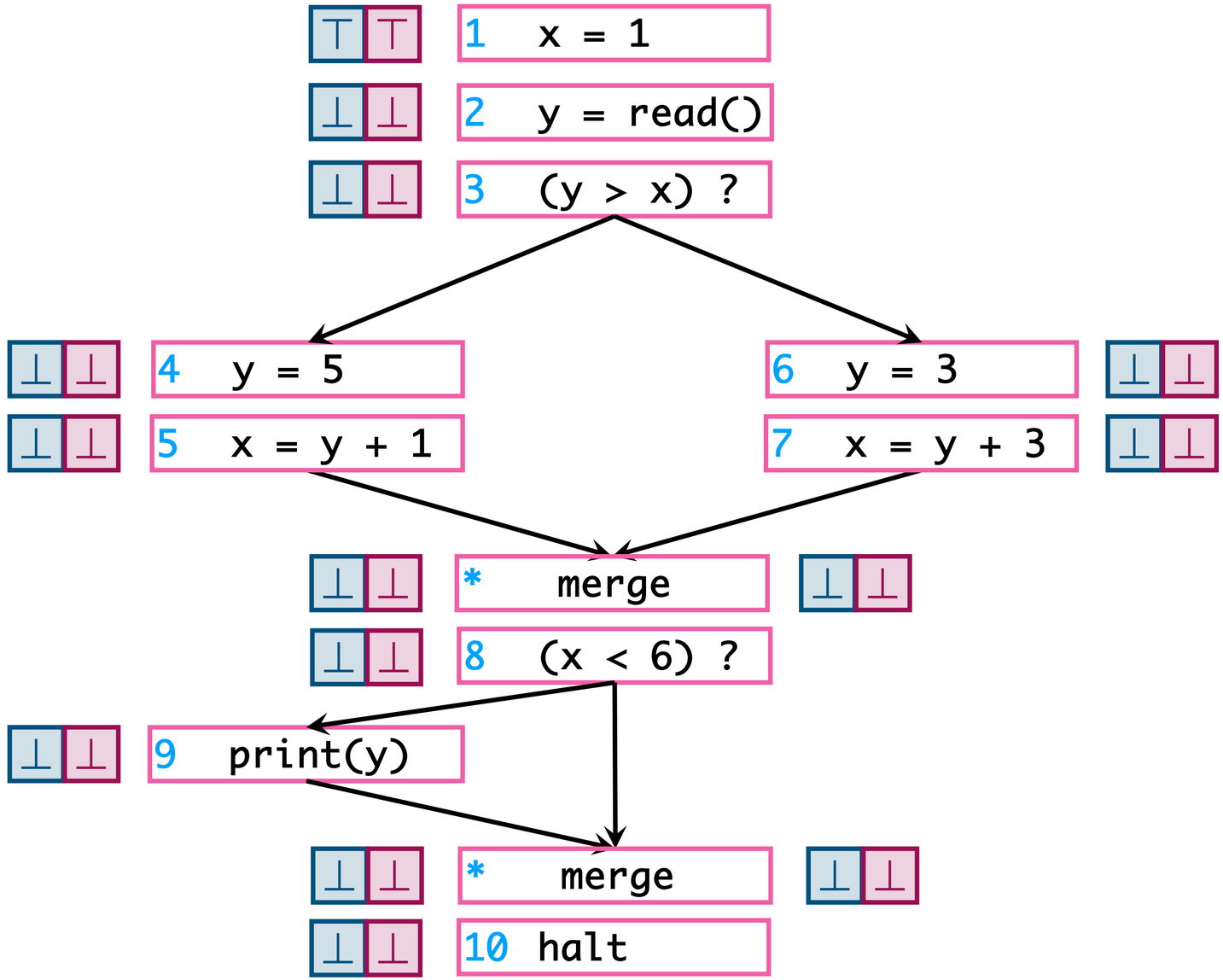
- Introduce *symbolic* values for each variable at each program point

⊥ No information about this variable

v Some constant value v (a particular constant)

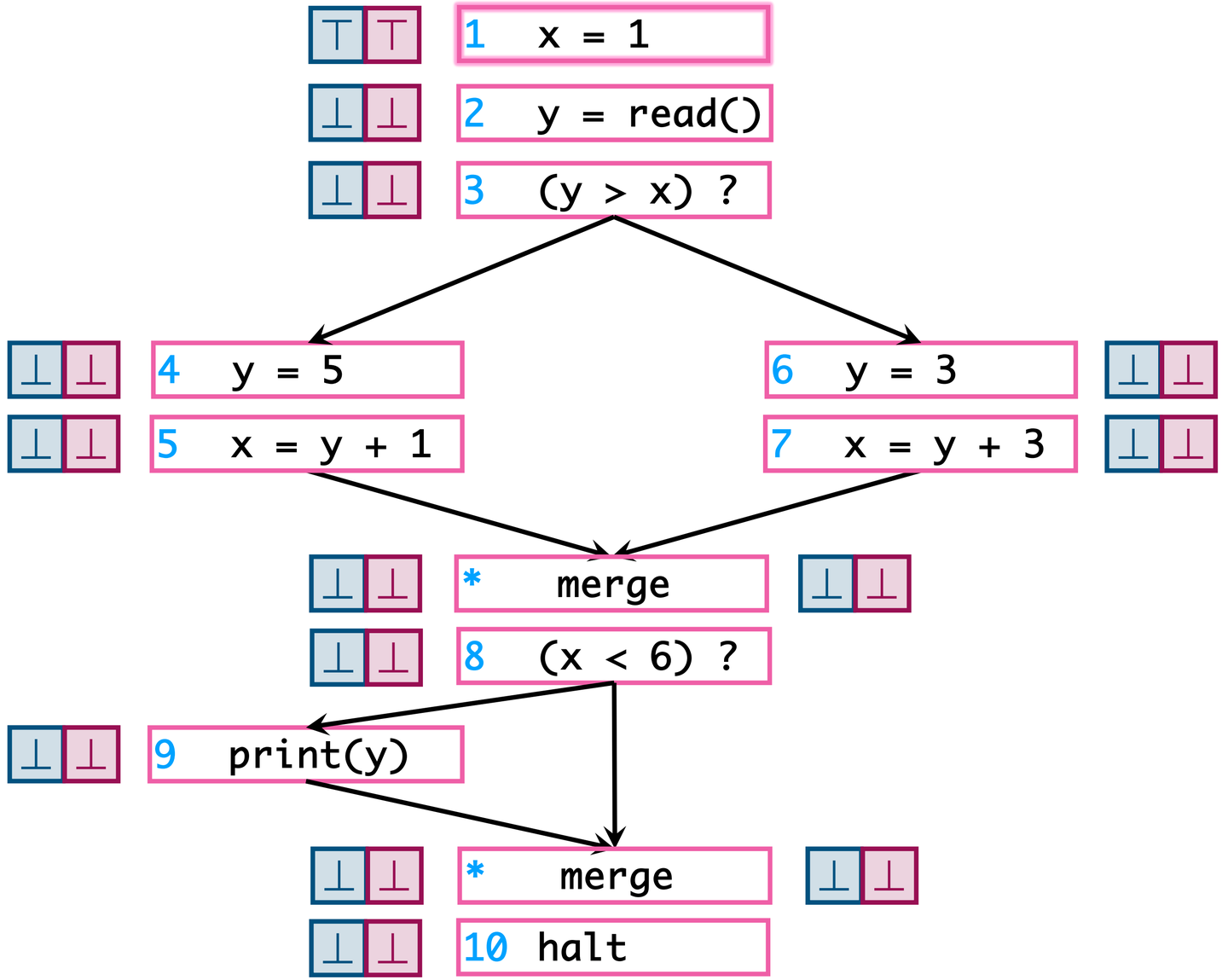
⊤ Definitely not a constant

- Before execution begins, have no information (except will assume that variables are definitely not constants at the beginning of the program)



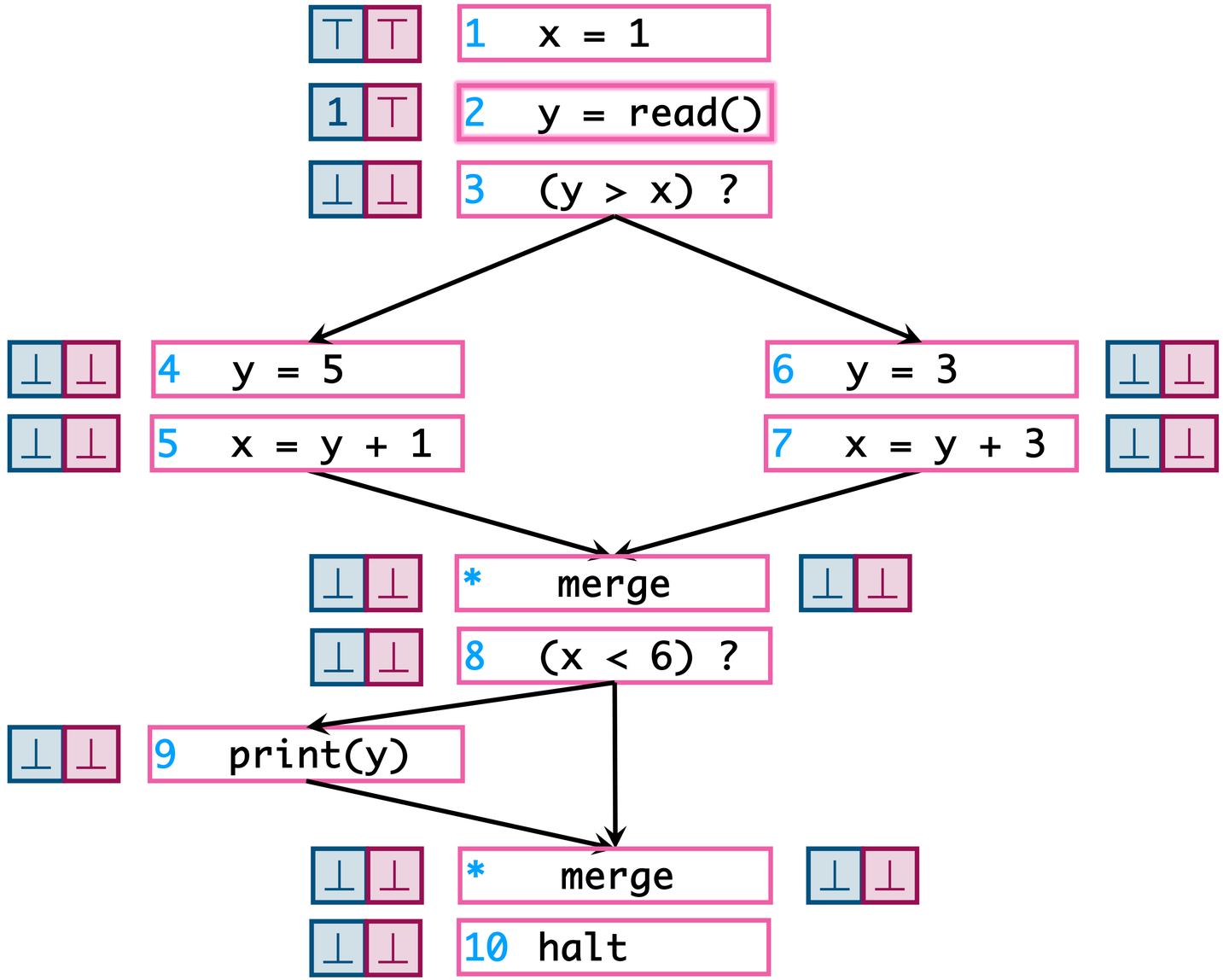
symbolic evaluation

- *Symbolically* evaluate expressions
- Evaluate expression with special rules:
  - If result of the expression is constant, set output to that constant
  - If not constant because of $\bot$ or $\top$, emit $\bot$ or $\top$



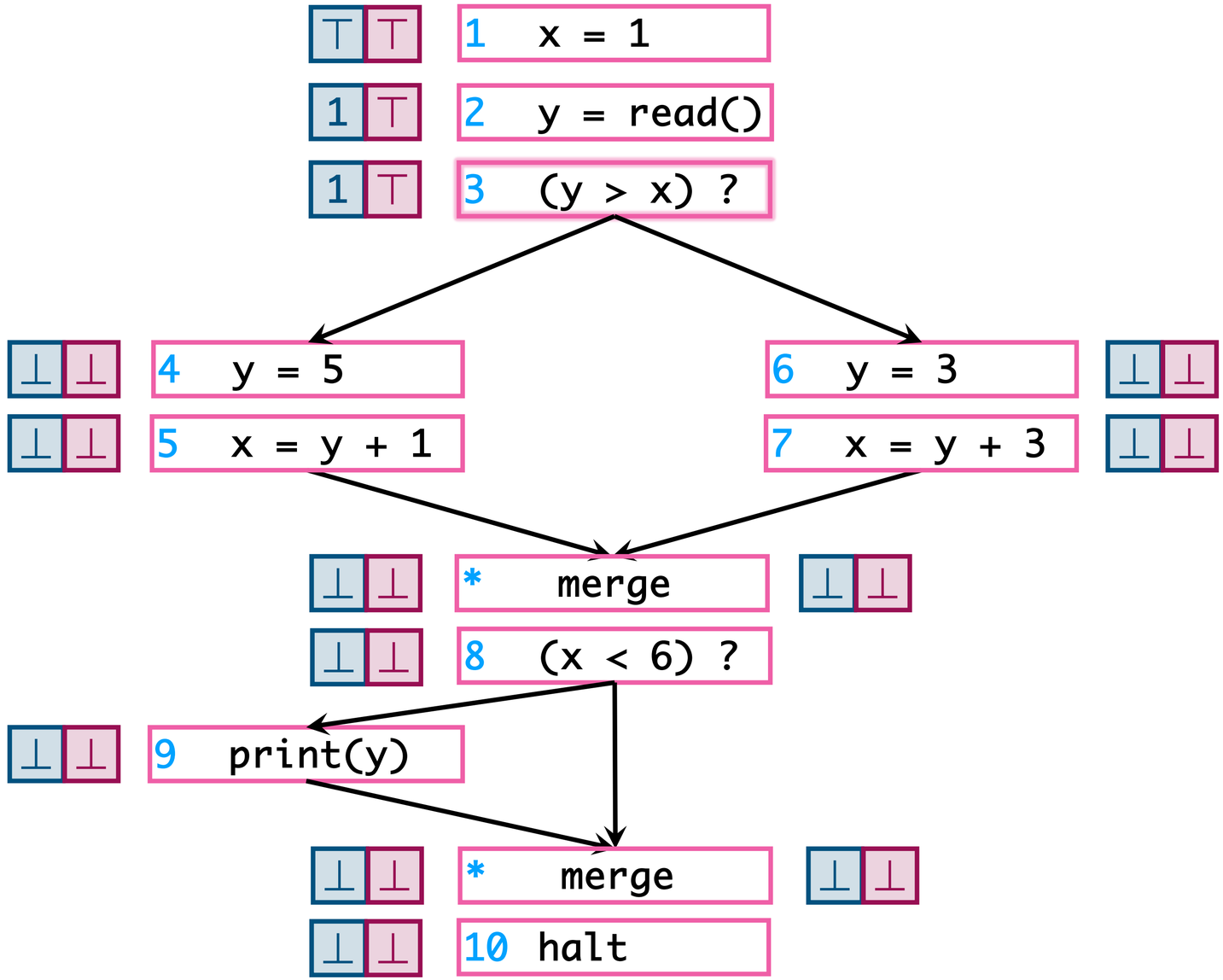
symbolic evaluation

- *Symbolically* evaluate expressions
- Evaluate expression with special rules:
  - If result of the expression is constant, set output to that constant
  - If not constant because of ⊥ or ⊤, emit ⊥ or ⊤



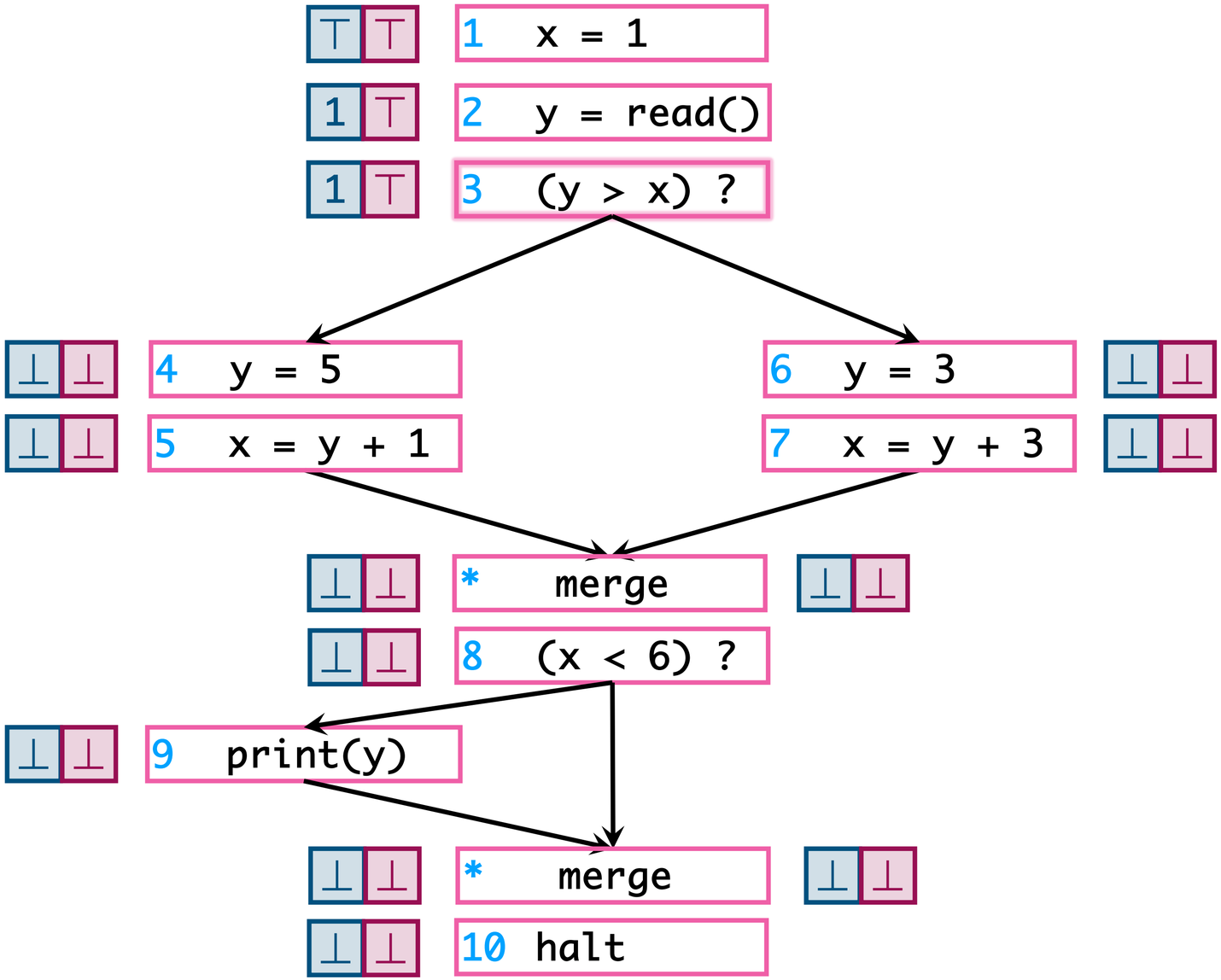
symbolic evaluation

- *Symbolically* evaluate expressions
- Evaluate expression with special rules:
  - If result of the expression is constant, set output to that constant
  - If not constant because of $\bot$ or $\top$, emit $\bot$ or $\top$



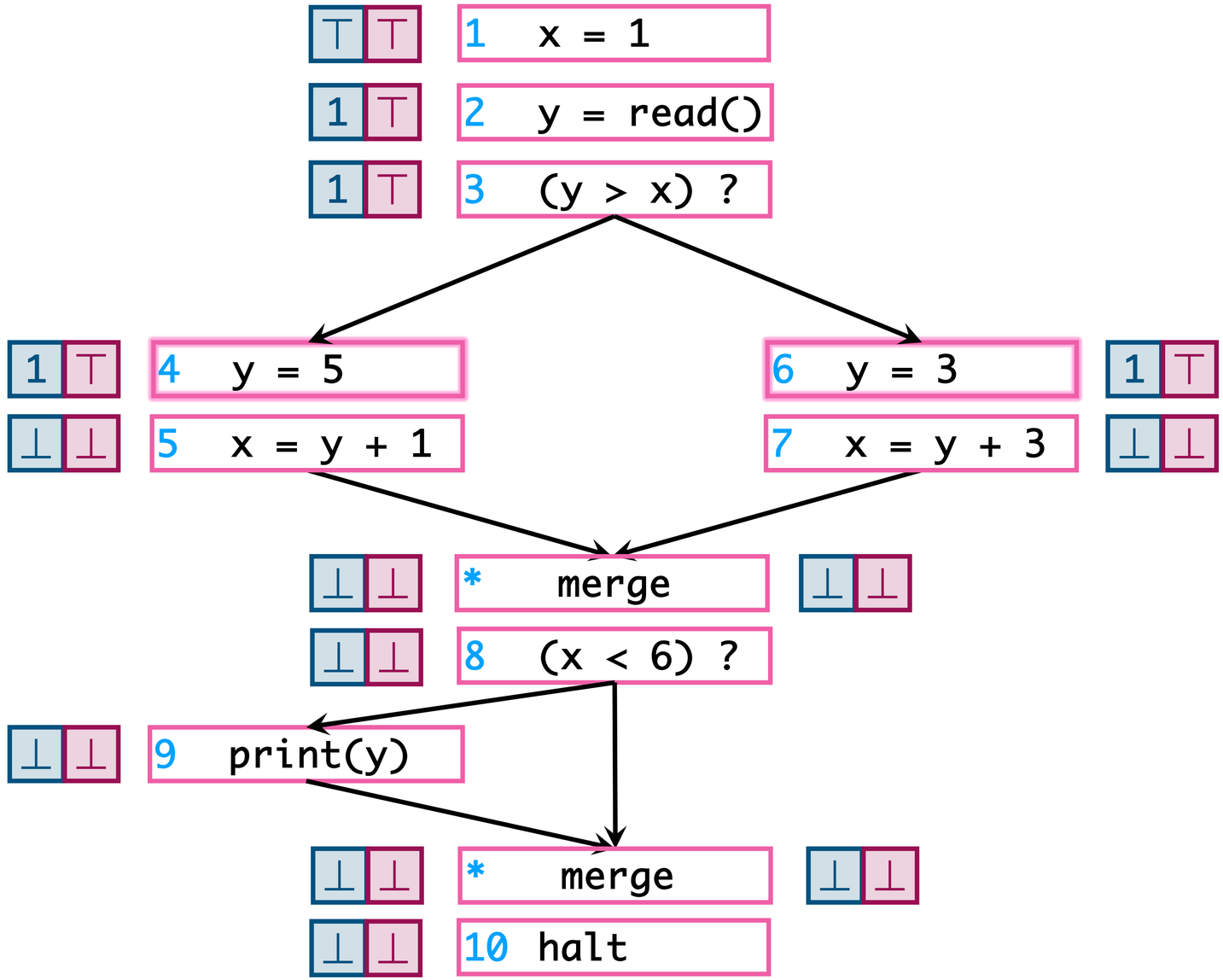
symbolic evaluation

- What if we cannot determine which way a branch goes?
- Magic of symbolic evaluation: **evaluate both branches**



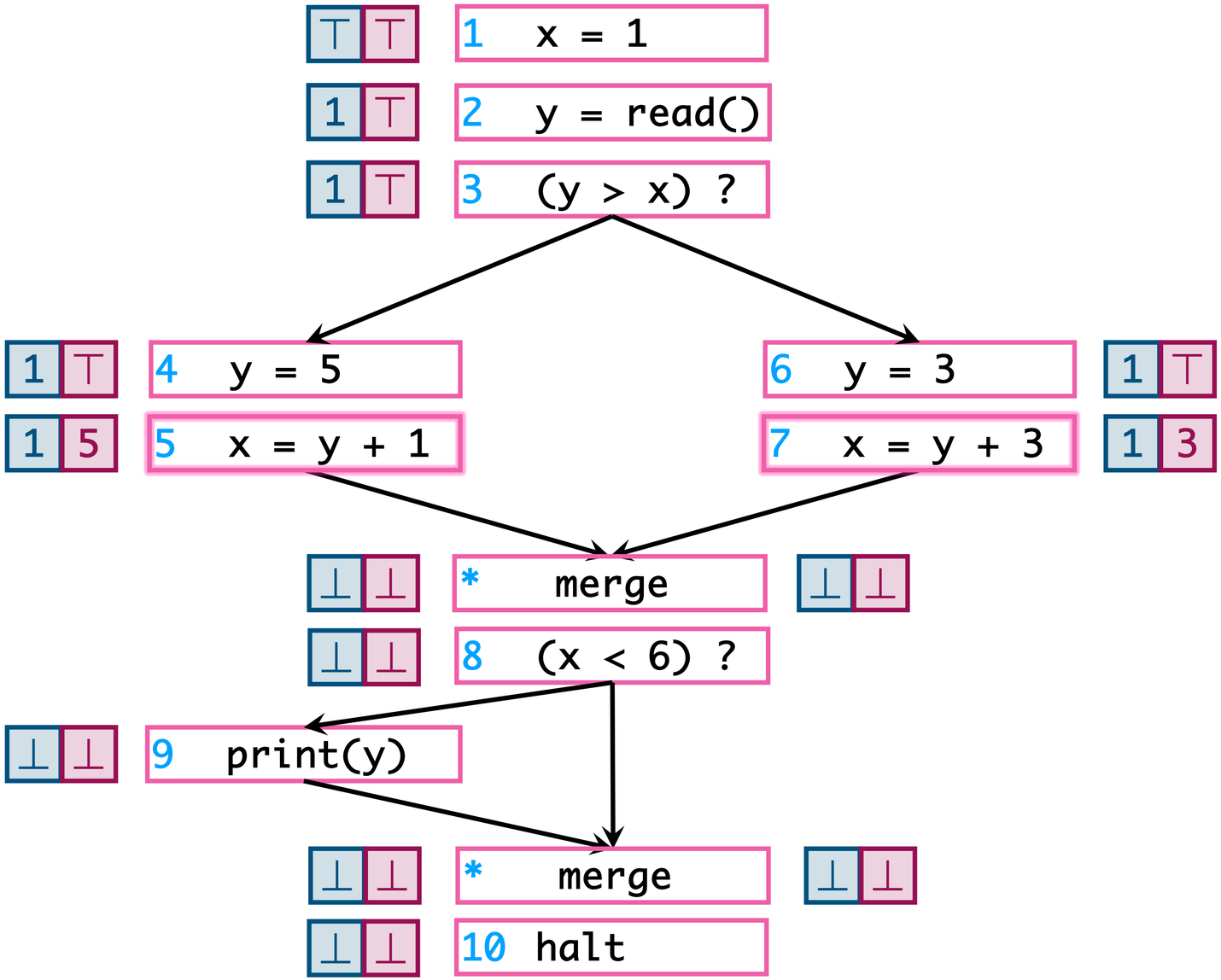
symbolic evaluation

- What if we cannot determine which way a branch goes?
- Magic of symbolic evaluation: **evaluate both branches**



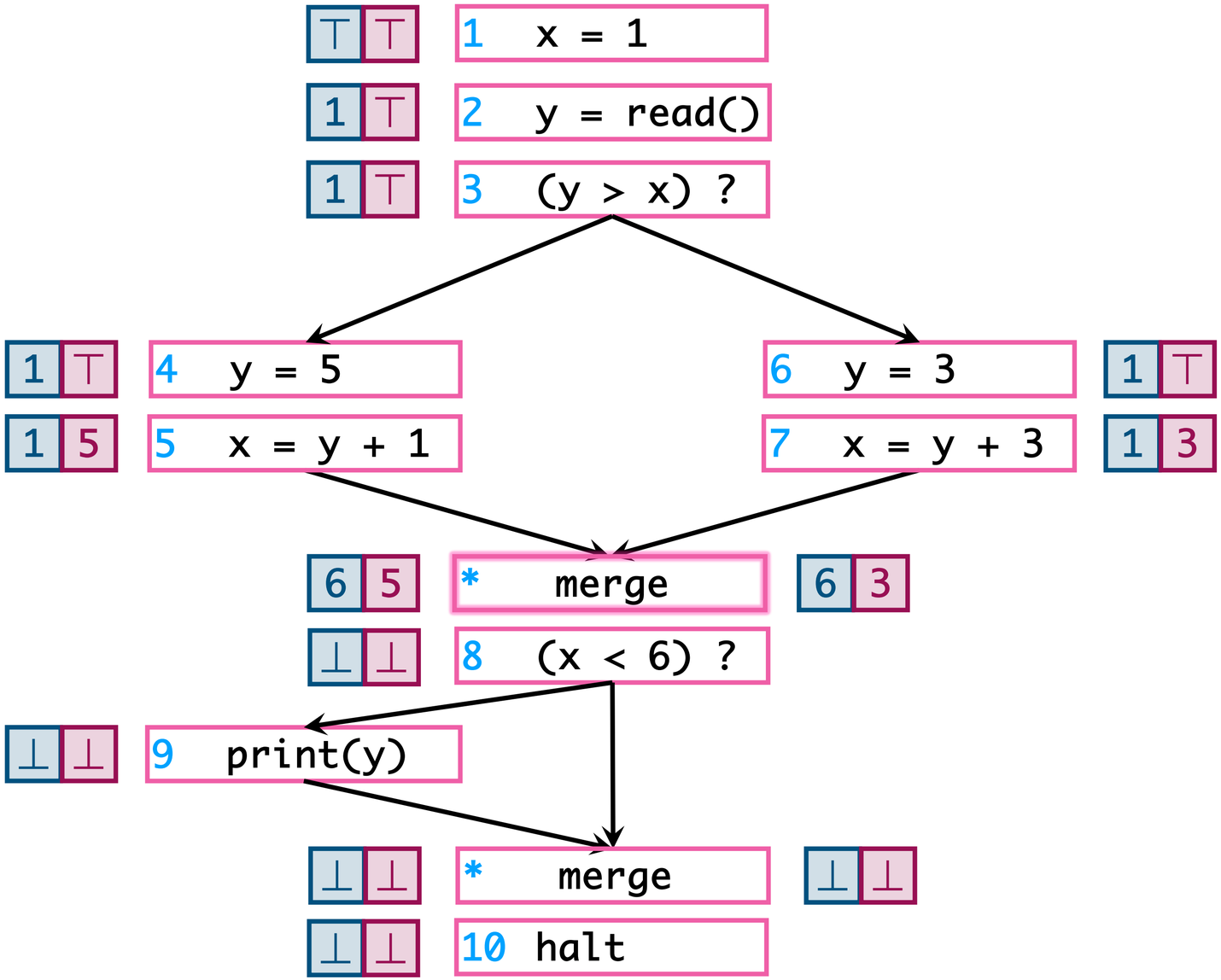
symbolic evaluation

- What if we cannot determine which way a branch goes?
- Magic of symbolic evaluation: **evaluate both branches**



symbolic evaluation

- What if we cannot determine which way a branch goes?
- Magic of symbolic evaluation: **evaluate both branches**

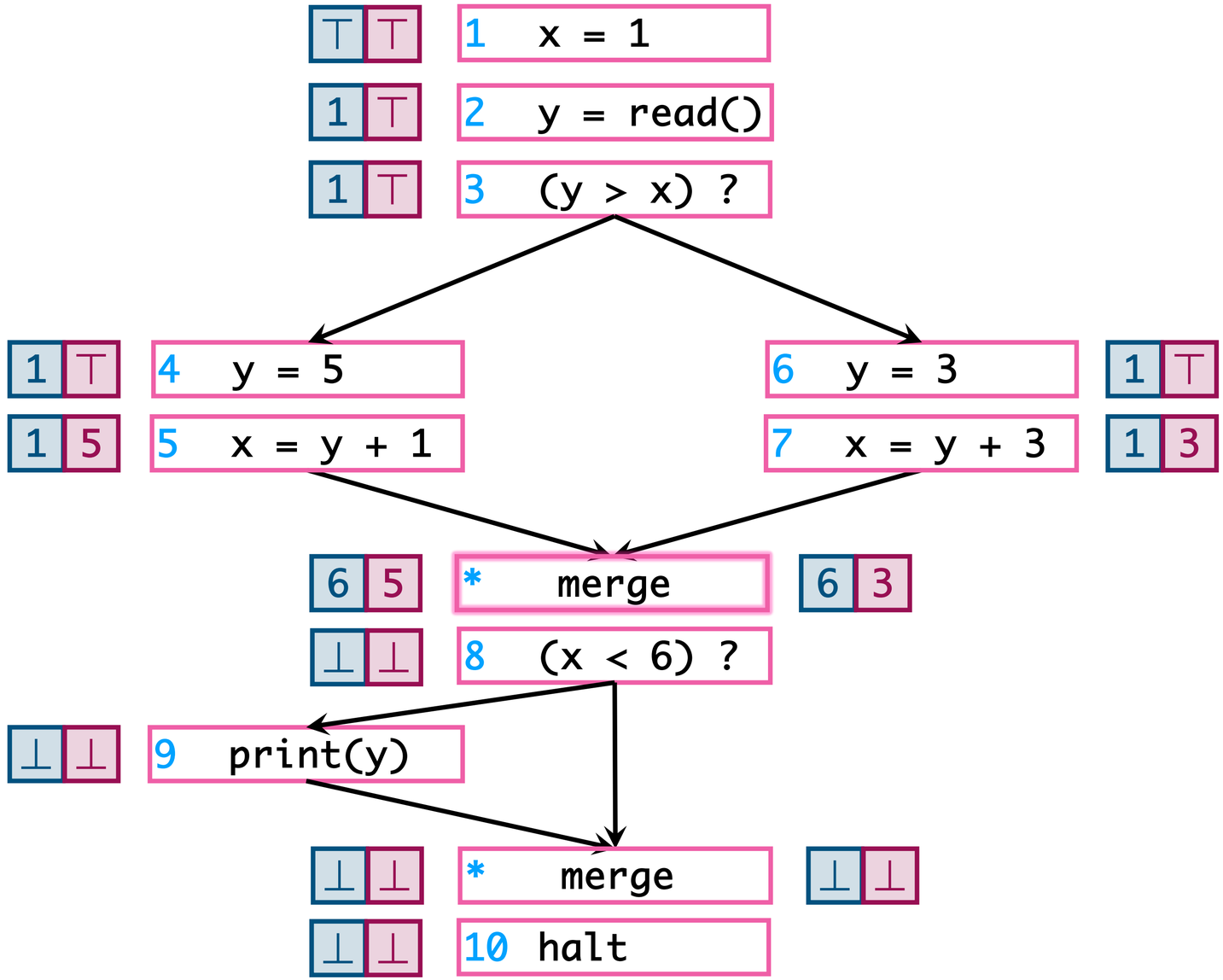


symbolic evaluation

- What do we do at merge points? Execution coming from more than one path

- Come up with a rule to **merge** information coming from two paths

1. v_1 vs. $v_1 \rightarrow v_1$
2. $\top$ vs. $* \rightarrow \top$
3. $\bot$ vs. $* \rightarrow *$
4. v_1 vs. $v_2 \rightarrow \top$

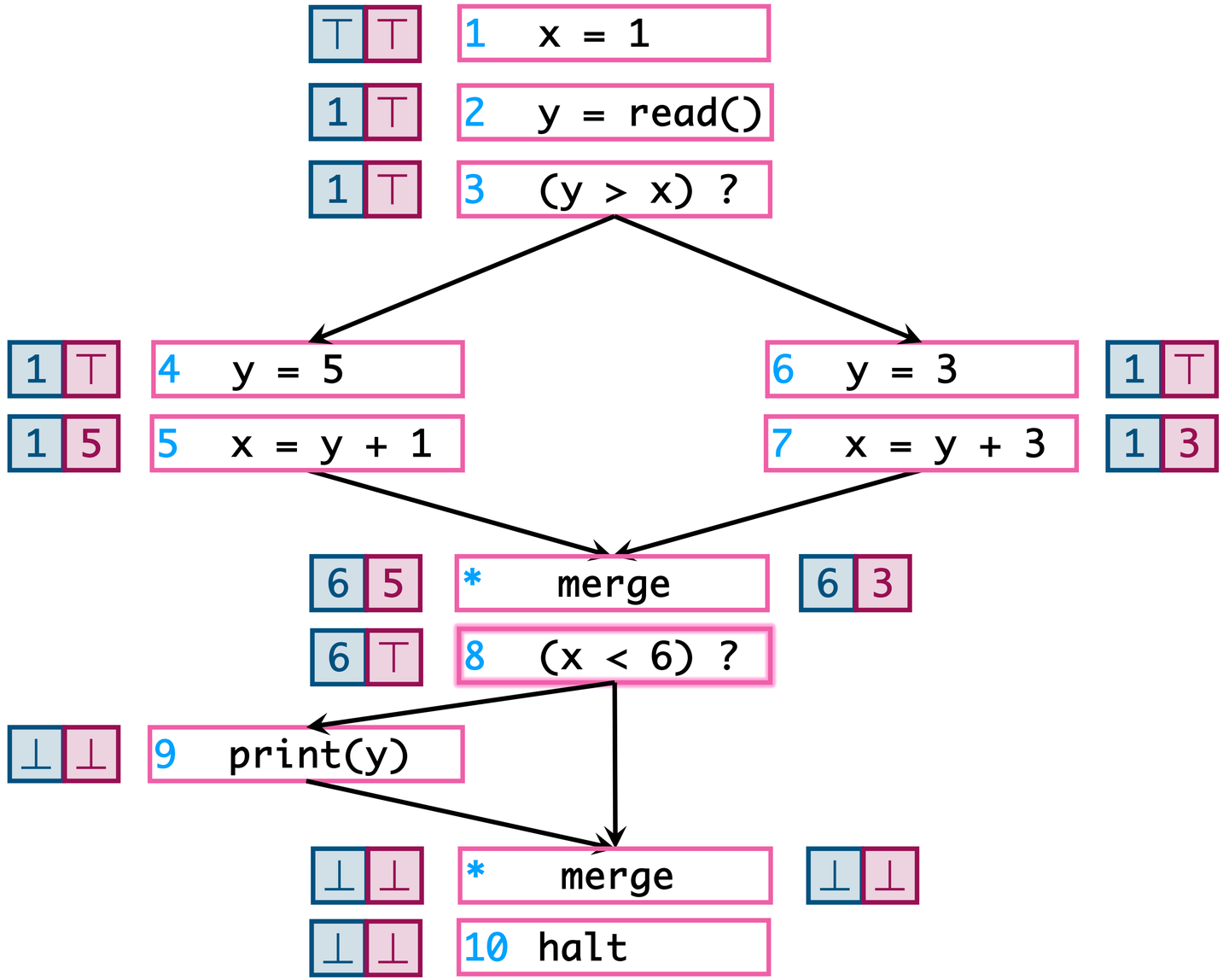


symbolic evaluation

- What do we do at merge points? Execution coming from more than one path

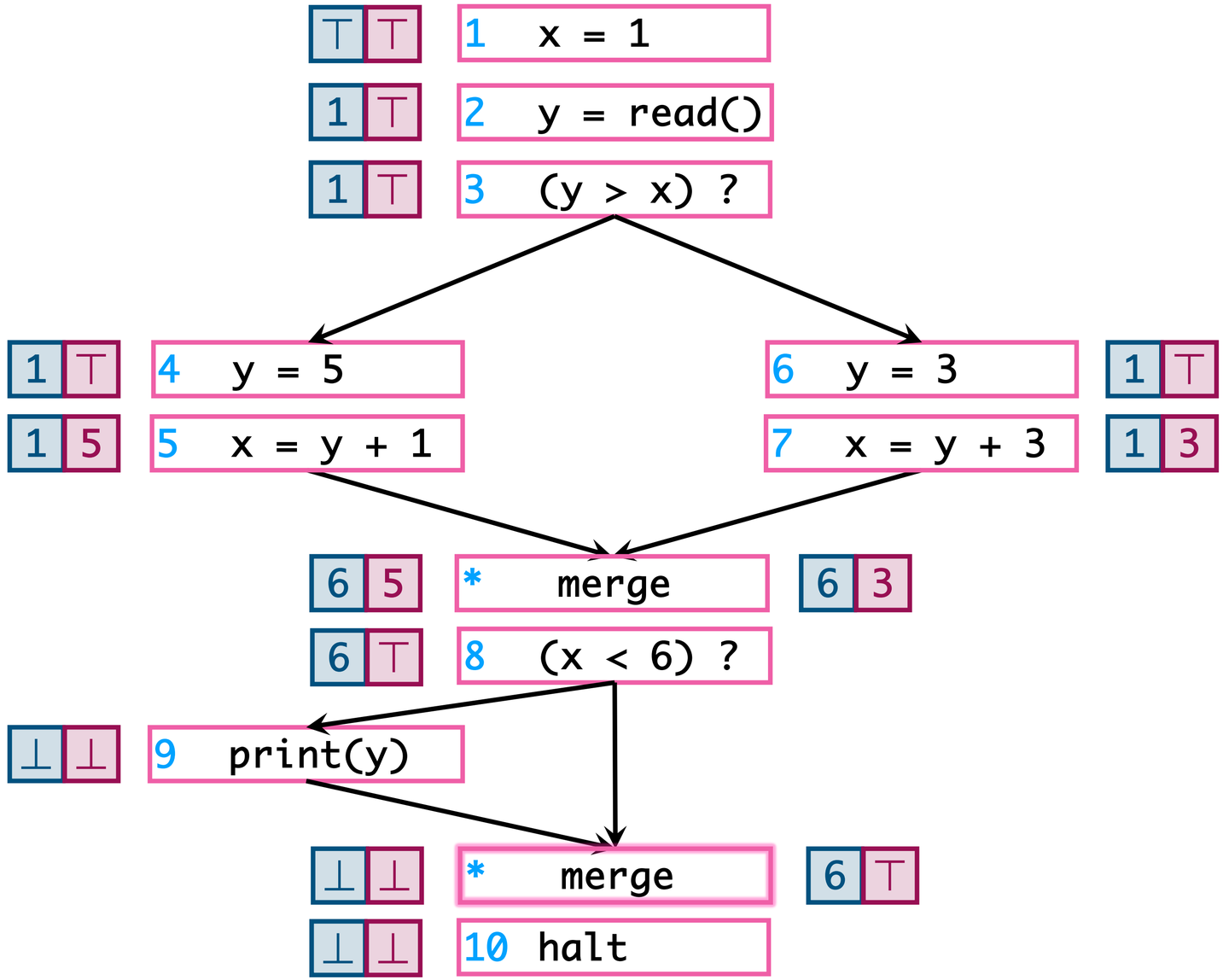
- Come up with a rule to **merge** information coming from two paths

1. v_1 vs. $v_1 \rightarrow v_1$
2. $\top$ vs. $* \rightarrow \top$
3. $\bot$ vs. $* \rightarrow *$
4. v_1 vs. $v_2 \rightarrow \top$



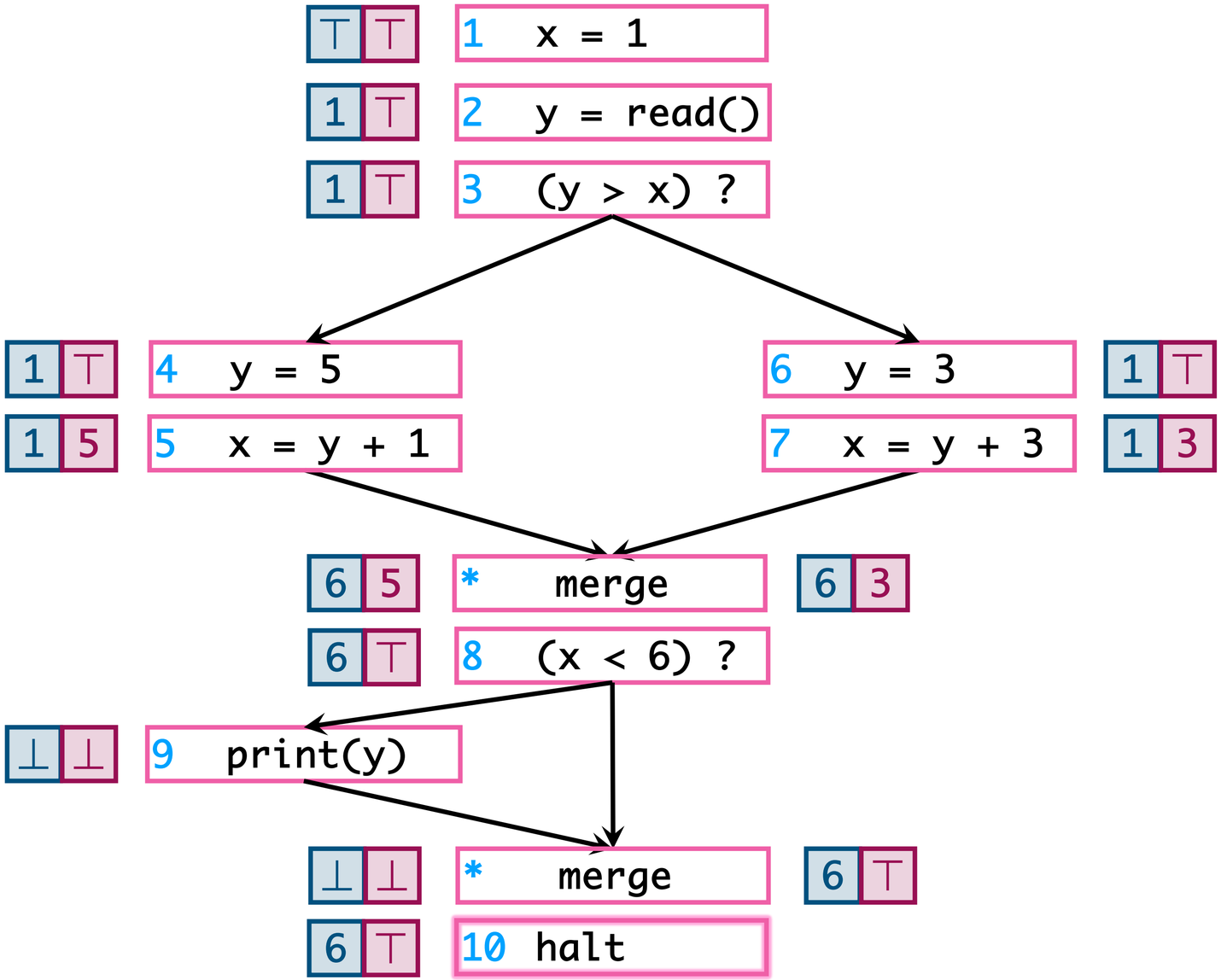
symbolic evaluation

- Keep executing until no more changes



symbolic evaluation

- Keep executing until no more changes



what about loops?

- Symbolically execute each statement in the program

- Treat loops as a **fixpoint** problem
  - If the inputs to a statement change, re-execute statement
  - Keep going until inputs stop changing

- Claim: this will handle loops
- Claim: inputs will eventually stop changing

next: loops and fixpoints