# AlOoTa*
# (Absolutely Lame OoTA)

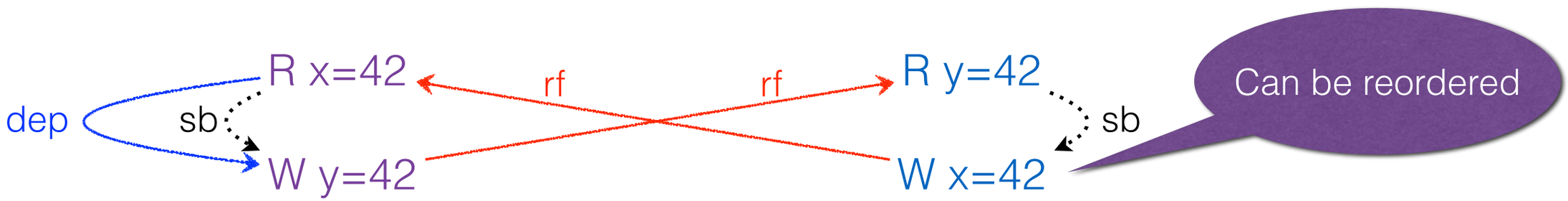
*: Ali’s OoTA

# Good

*Program*

```
r1 = x;          ||          r2 = y;
y = r1;                      x = 42;
```

## Execution

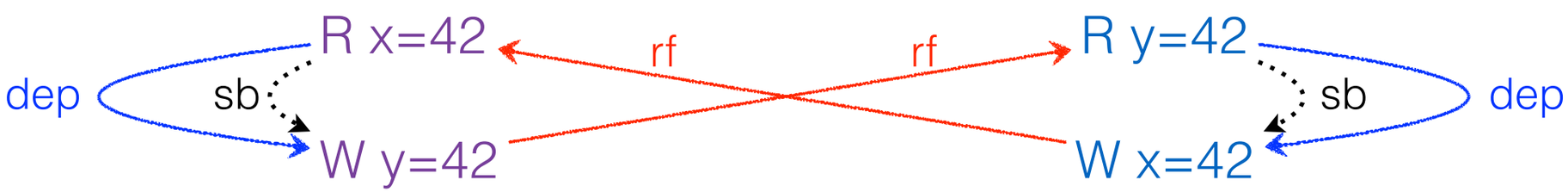


# Still Good?

*Program*

```
r1 = x;                  ||   r2 = y;
if (r1>0)                ||   if (r2>0)
    y = r1;              ||       x = 42;
```

## Execution




❌ SC Guarantee: $[x_i{=}y_i{=}0 \Rightarrow x{=}0] \wedge [y_i{>}0 \Rightarrow x{=}42]$

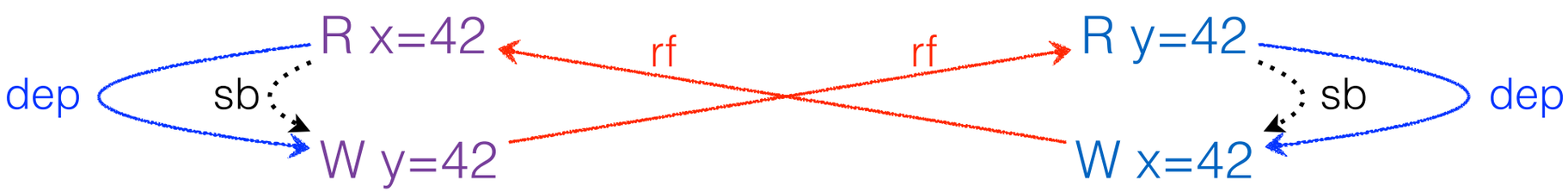
✅ Actual Guarantee: $[x_i{=}y_i{=}0 \Rightarrow x{=}0 \vee x{=}42] \wedge [y_i{>}0 \Rightarrow x{=}42]$

# Bad

*Program*

```
r1 = x;                  ||      r2 = y;
if (r1>0)                ||      if (r2>0)
    y = r1;              ||          x = r2;
```

## Execution




❌ SC Guarantee: $[y_i>0 \Rightarrow x=y_i \vee x=x_i]$

✅ Actual Guarantee: true {x can have any value}

# Under-approximation FTW

- Remove “bad” executions.


- Keep “good” executions.

- Don't worry about the “questionable” executions (yet).

# Terminology (really short)

| Instructions | Potential Instantiations |
|---|---|
| `r1 = x;` | { R x=0 , R x=1 , R x=2 , R x=3 , ... } |
| `x = r1;` | { W x=0 , W x=1 , W x=2 , W x=3 , ... } |
| `r1 = k;` | { R x=*k* } |
| `x = k;` | { W x=*k* } |
| `r1 = Expr(`$r_i$`);` | { R x=*k* \| *k* ∈ ⟦Expr($r_i$)⟧ } |
| `x = Expr(`$r_i$`);` | { W x=*k* \| *k* ∈ ⟦Expr($r_i$)⟧ } |

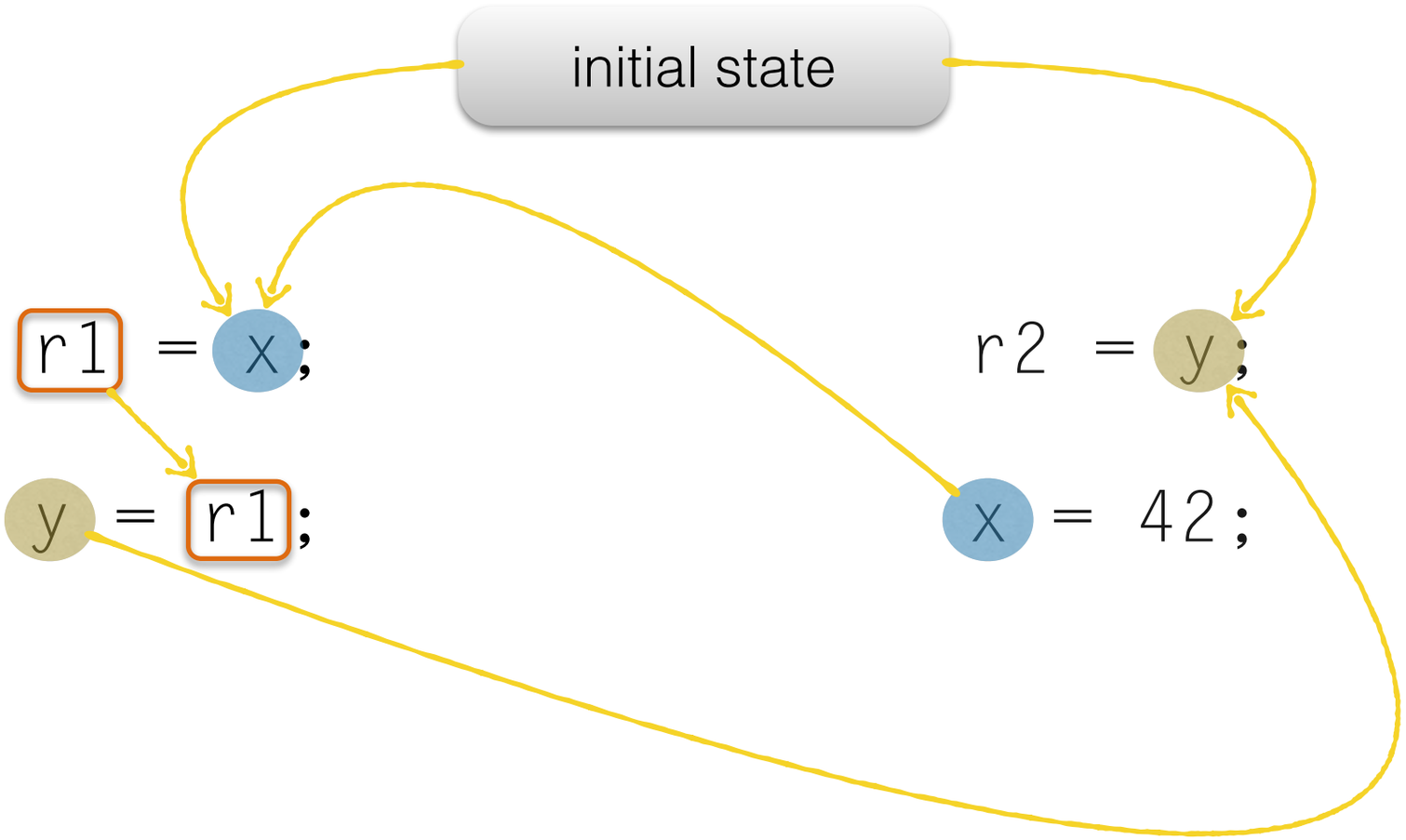
# Terminology (really short)

An instruction is *determined* if its set of possible instantiations is a singleton.

| | | |
|---|---|---|
| determined | `r1 = k;` | { R x=*k* } |
| undetermined | `x = r1;` | { W x=0, W x=1, W x=2, W x=3,... } |
| determined | `r1 = Expr(`$r_i$`);` | { R x=*k* \| *k* ∈ ⟦Expr($r_i$)⟧ } if \|⟦Expr($r_i$)⟧\| =1 |
| undetermined | `r1 = Expr(`$r_i$`);` | { R x=*k* \| *k* ∈ ⟦Expr($r_i$)⟧ if \|⟦Expr($r_i$)⟧\| >1 |

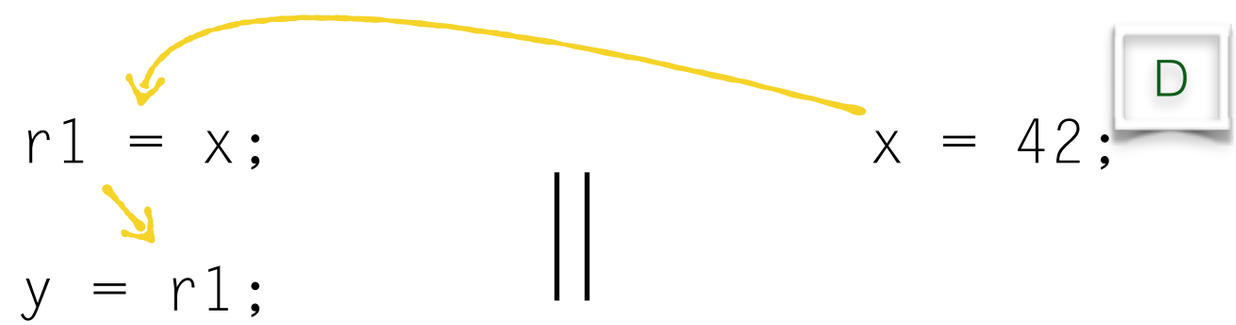
# Terminology (really short)

An instruction *i* is a provider for another instruction *j* if *i* modifies a register/location which *j* uses.



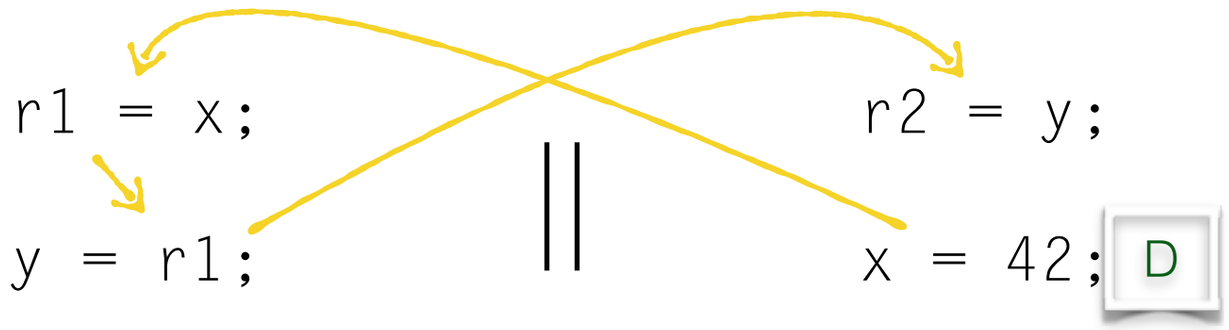
# Terminology (really short)

An instantiation in an execution is resolved if it belongs to a determined instruction or each of its operands are provided by resolved instantiations.

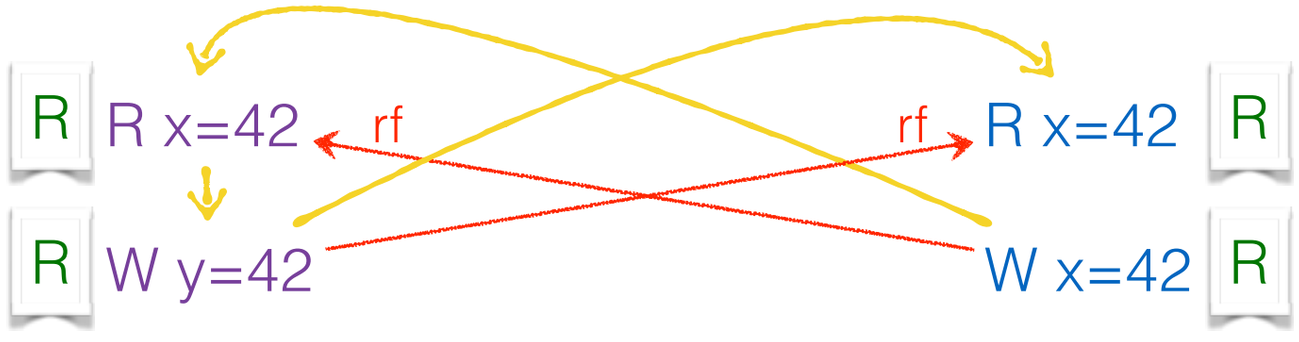



# Terminology (really short)

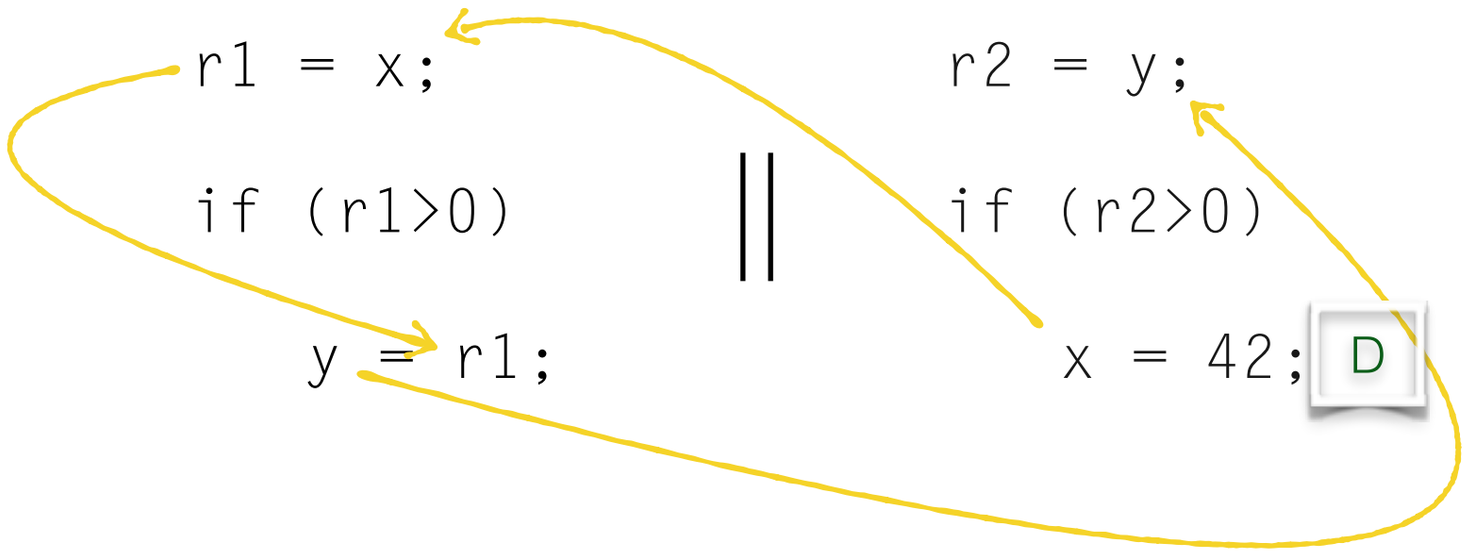
An execution does not have AIOoTA relative to *R* if it is possible to mark each node as resolved by edges that do not conflict with *R*.



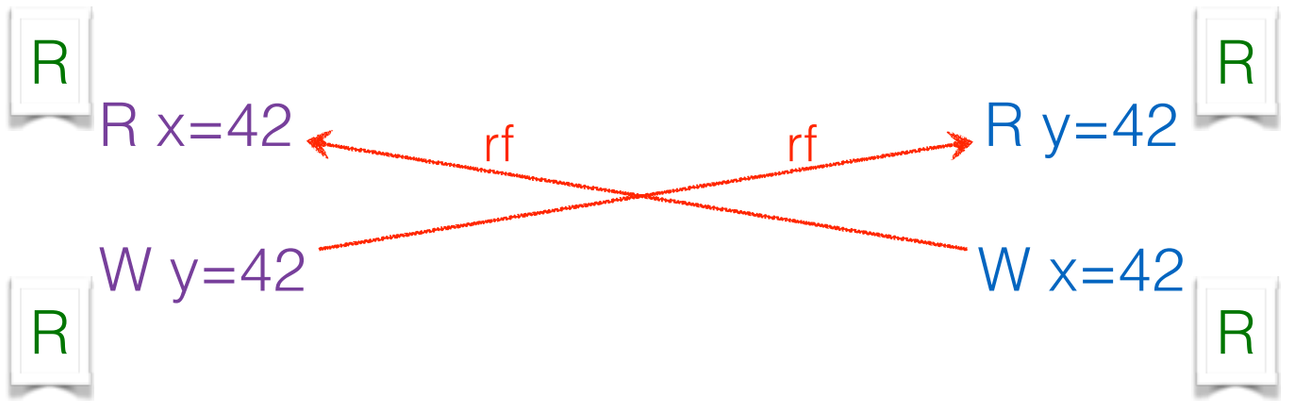

*R*=rf



# Example 1

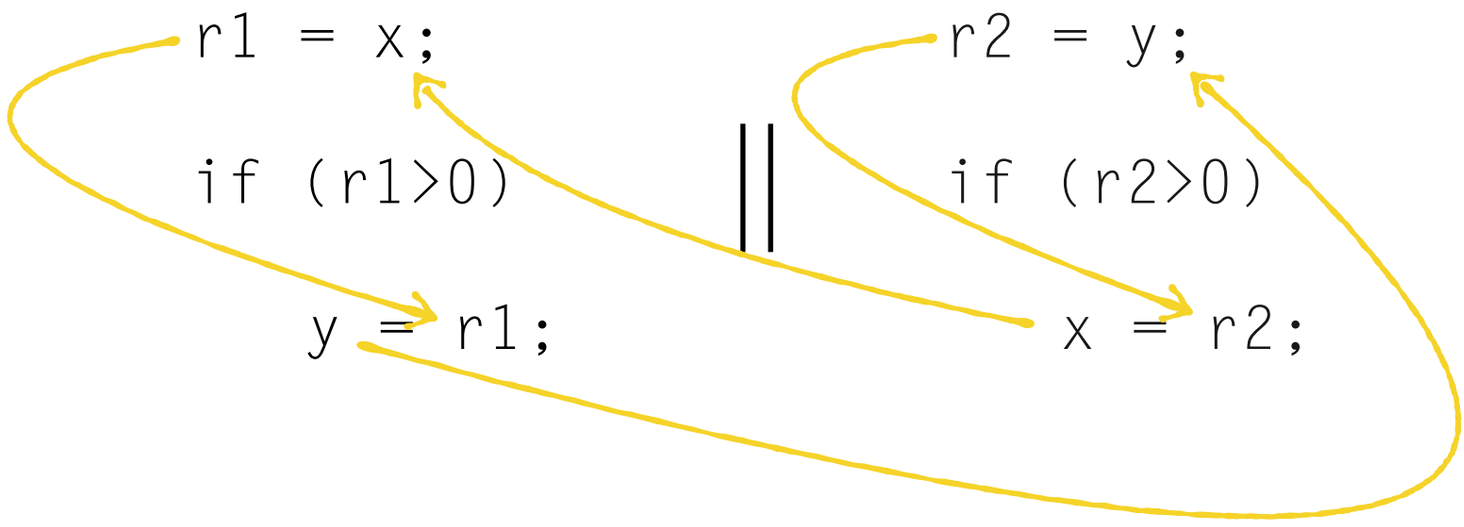



*R*=rf

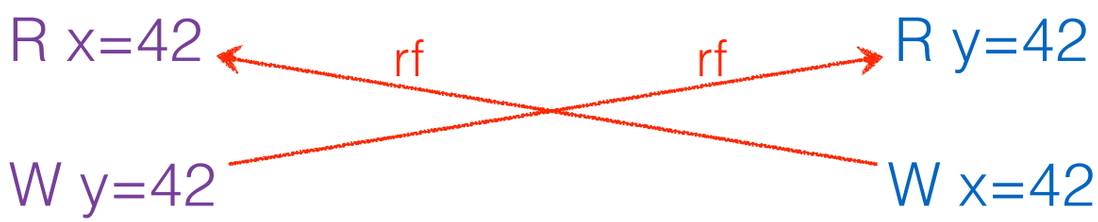



No AlOoTA

# Example 2



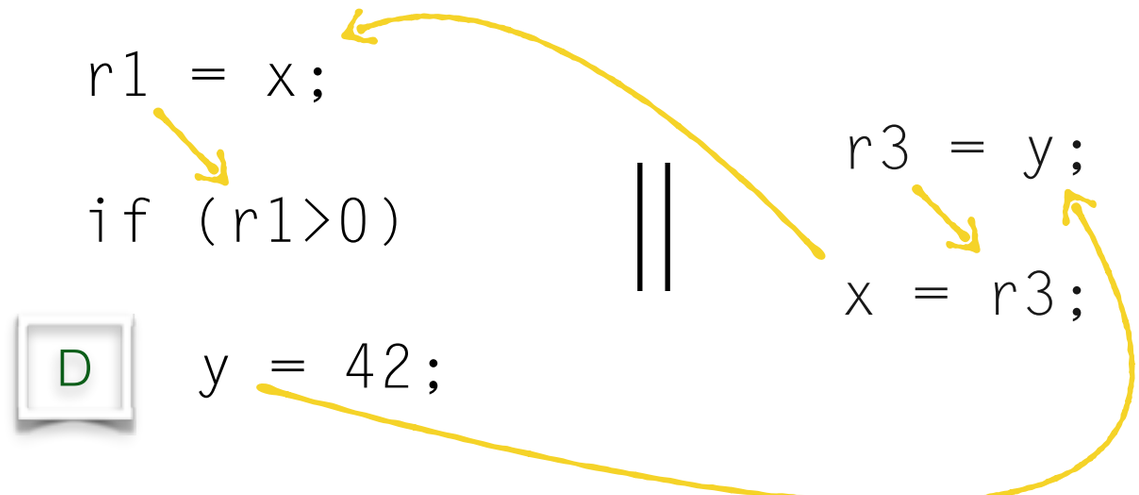

*R*=rf



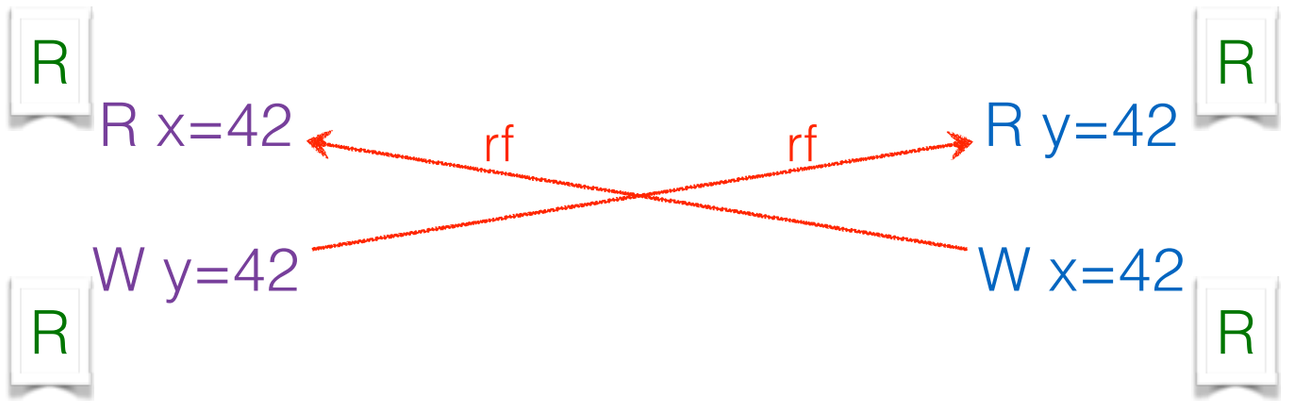

AIOoTA

# Example 3

{x=23,y=23}




*R*=rf



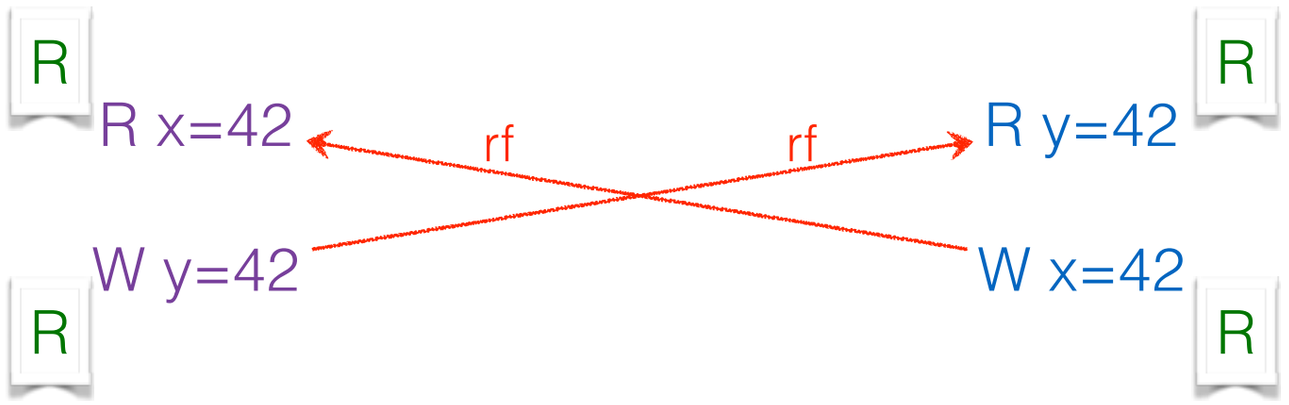

No AlOoTA

# Example 4

```
r1 = x;          ||   r4 = y;
r2 = 2*r1;       ||
r3 = r2 % 2;     ||   x = r4;
y = 42+r3;
```

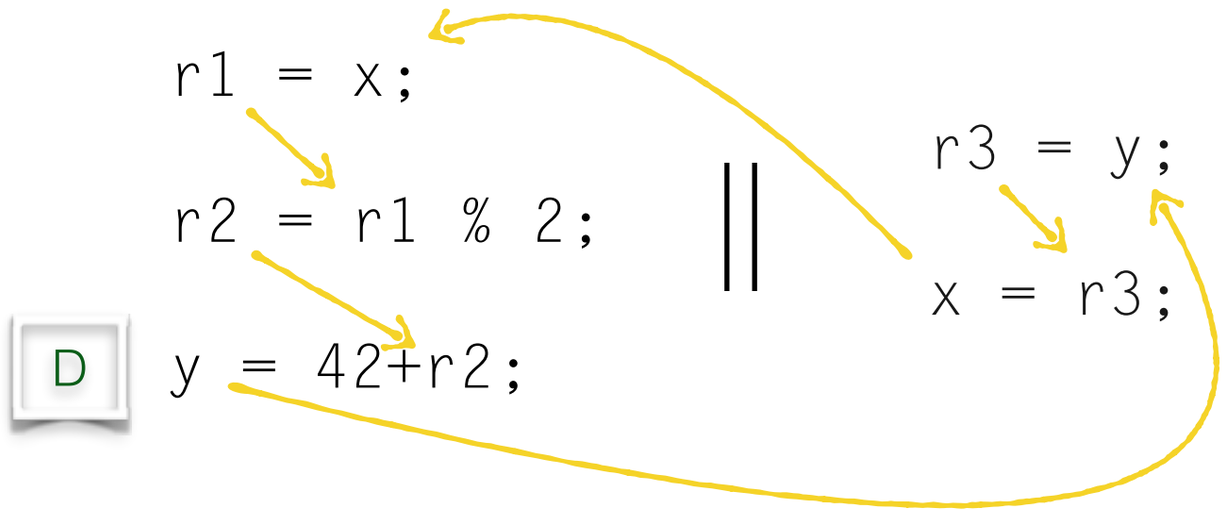
D

*R*=rf



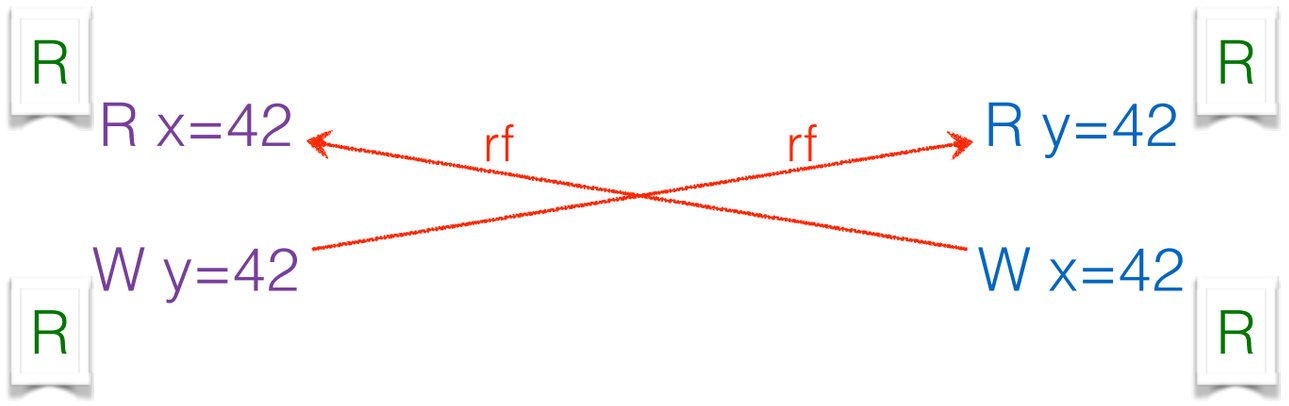

No AlOoTA

# Example 5



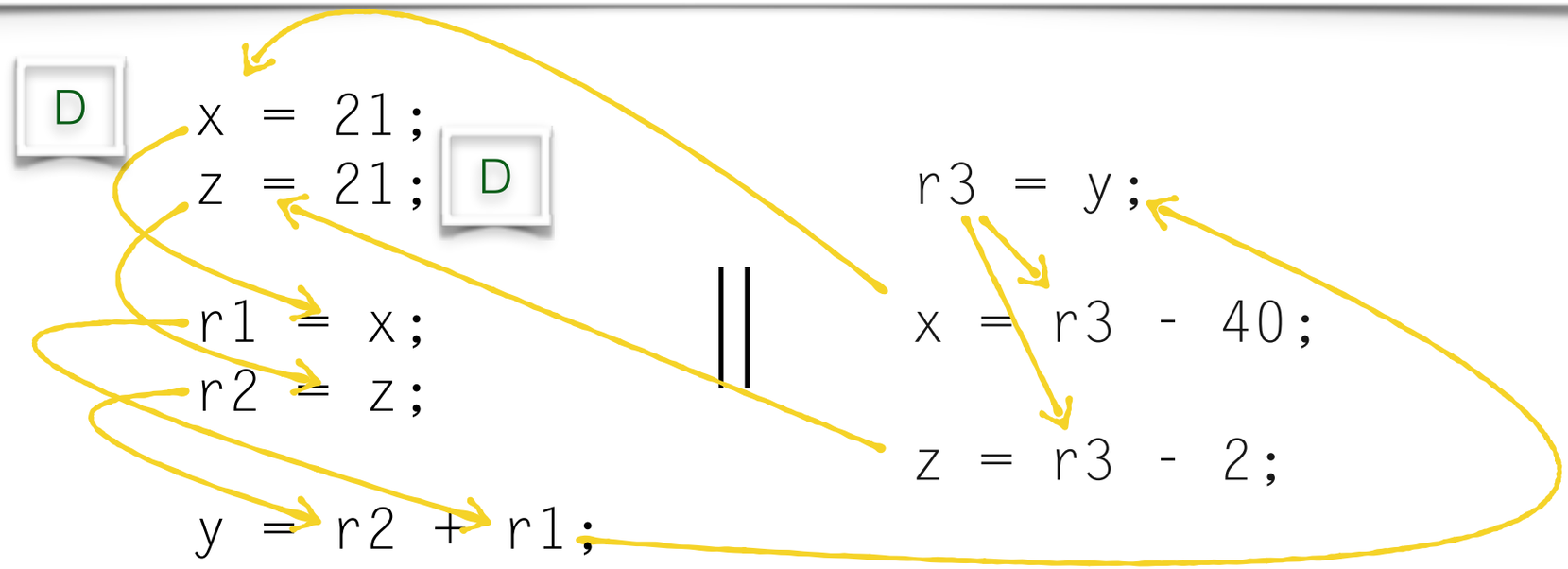

D

*R*=rf

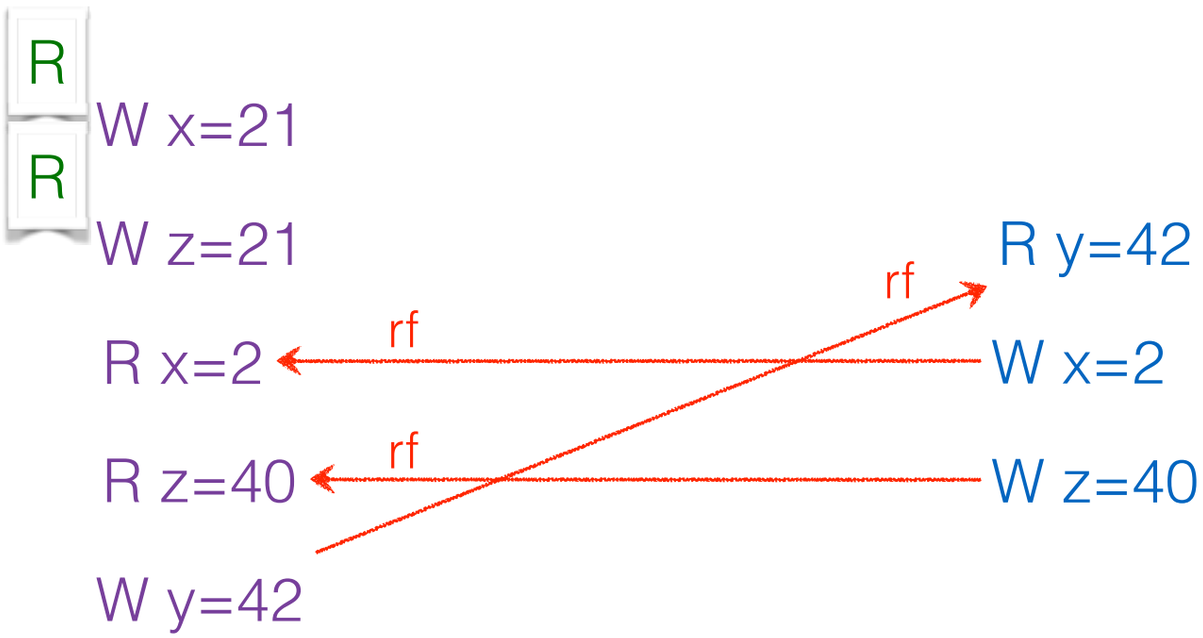



No AlOoTA

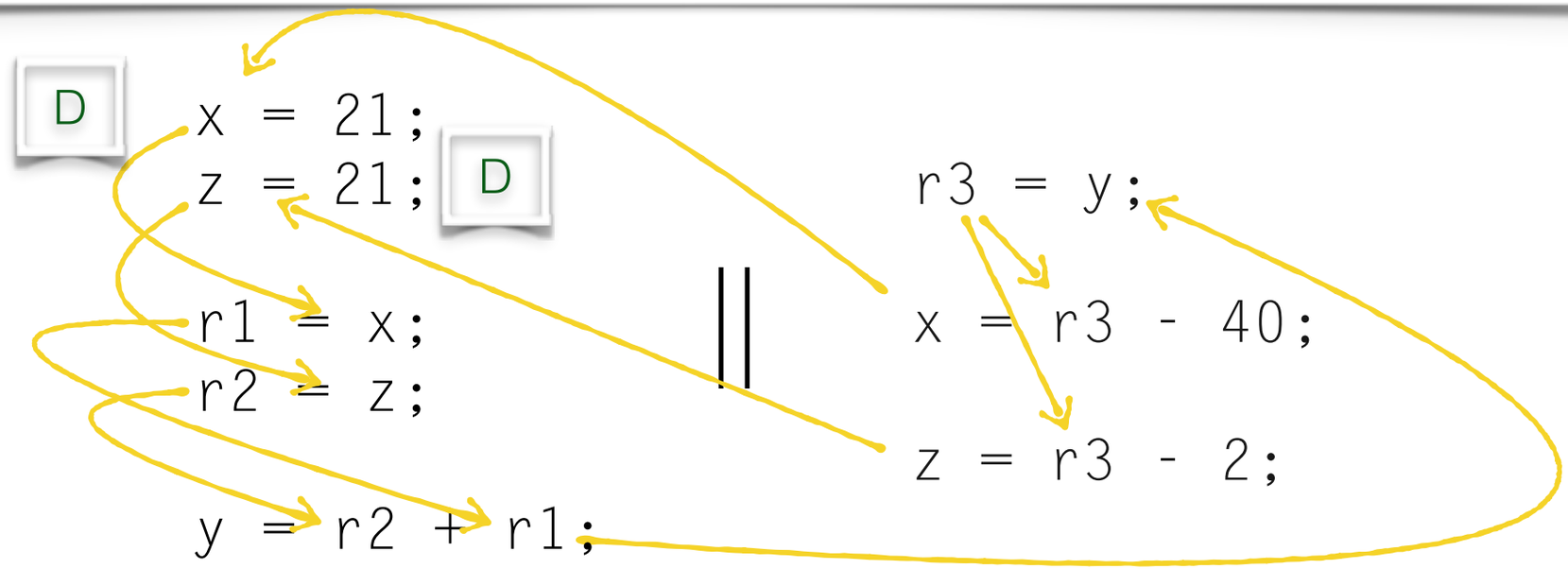
# To AlOoTA or To not AlOoTa?



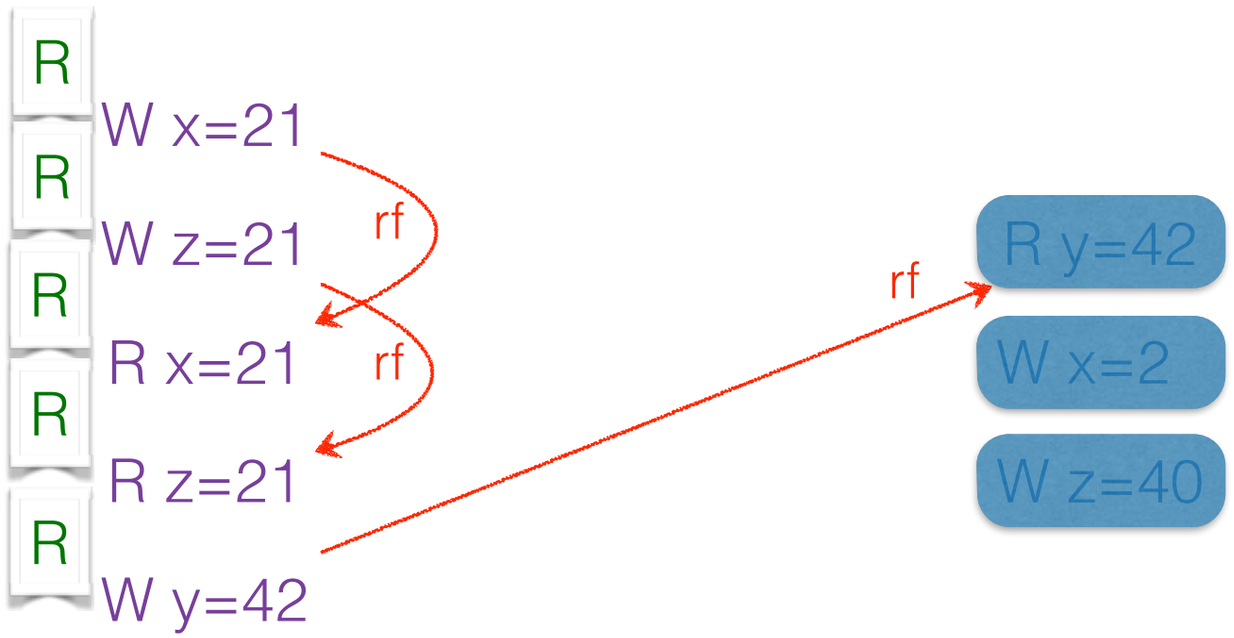

*R*=rf



# To AlOoTA or To not AlOoTa?




*R*=rf



# Final Words

- Speculations, how to “resolve”?
- Is there a way to quantify how far we are off relative to SC guarantees?
- Examples, examples, examples.