Type-Based Aliasing Control for DDC

Tran Ma
University of New South Wales
FP-SYD 2012/11/15

Disciple and DDC

- Strict by default

- Regions, effects and closure typing

$$\texttt{updateInt :: [r1 r2 : \%]. Mutable r1} \Rightarrow \texttt{Int r1} \rightarrow \texttt{Int r2} \rightarrow \texttt{()}$$

- Upcoming major DDC release!

disciple.ouroborus.net

Aliasing Problem

```
next = Λ r1 r2. λ (w1 : Mutable r1). λ (v : Int r1) (c : Int r2).
       letregion r3 in
       let (x : Int r3) = addInt    [r1] [r2] [r3] v c
           _            = updateInt [r1] [r2] <w1> v x
       in  addInt [r2] [r3] [r2] c (1 [r3] ())
```

$$\texttt{addInt} \;::\; [\texttt{r1 r2 r3} : \%].\ \texttt{Int r1} \to \texttt{Int r2} \xrightarrow{Read\ r1\ +\ Read\ r2\ +\ Alloc\ r3\ |\ Use\ r1} \texttt{Int r3}$$

$$\texttt{updateInt} \;::\; [\texttt{r1 r2} : \%].\ \texttt{Mutable r1} \Rightarrow \texttt{Int r1} \to \texttt{Int r2} \xrightarrow{Read\ r_2\ +\ Write\ r_1\ |\ Use\ r_1} \texttt{Unit}$$

Aliasing Problem

```
next = Λ r1 r2. λ (w1 : Mutable r1). λ (v : Int r1) (c : Int r2).
       letregion r3 in
       let (v1 : Int#)    = unboxInt   [r1] v                       Read r1
           (c1 : Int#)    = unboxInt   [r2] c                       Read r2
           (x  : Int r3)  = boxInt     [r3] (add# v1 c1)            ⊥
           _              = updateInt  [r1] [r3] <w1> v x           Read r3 + Write r1
           (c2 : Int#)    = unboxInt   [r2] c                       Read r2
       in  boxInt r2 (add# c2 1#)                                   ⊥
```

$$\texttt{boxInt} \;::\; \texttt{[r : \%]. Int\#} \xrightarrow{Alloc\ r\ \mid\ Use\ r} \texttt{Int r}$$

$$\texttt{unboxInt} \;::\; \texttt{[r : \%]. Int r} \xrightarrow{Read\ r\ \mid\ \$0} \texttt{Int\#}$$

$$\texttt{add\#} \;::\; \texttt{Int\#} \rightarrow \texttt{Int\#} \rightarrow \texttt{Int\#}$$

Witnesses to the Rescue

```
next = Λ r1 r2.λ (w1 : Mutable r1) (w2 : Const r2). λ (v : Int r1) (c : Int r2). ...
```

Too strict!

Witnesses to the Rescue

```
next = Λ r1 r2.λ (w1 : Mutable r1) (w2 : Const r2). λ (v : Int r1) (c : Int r2). ...
```

Too strict!

```
next = Λ r1 r2.λ (w1 : Mutable r1) (w2 : Distinct r1 r2). λ (v : Int r1) (c : Int r2).
       letregion r3 with { w : Const r3 } in
       let (v1 : Int#)   = unboxInt [r1] v                      Read r1
           (c1 : Int#)   = unboxInt [r2] c                      Read r2
           (x  : Int r3) = boxInt [r3] (add# v1 c1)             ⊥
           _             = updateInt [r1] [r3] <w1> v x         Read r3 + Write r1
       in  boxInt r2 (add# c1 1#)                               ⊥
```

Get to it already!

```
Distinct :: % ~> % ~> @
```

$$\frac{r \notin \Delta \qquad \Delta \mid r \vdash \overline{w_j : \tau_j} \text{ well-formed} \qquad \Delta,\ r : \% \mid \Gamma,\ \overline{w_j : \tau_j} \vdash t : \tau\ ;\ \sigma\ ;\ \gamma \qquad \Delta \vdash \tau : * \qquad r \notin fv(t)}{\Delta \mid \Gamma \vdash \textbf{letregion}\ r\ \textbf{with}\ \{\overline{w_j : \tau_j}\}\ \textbf{in}\ t : \tau\ ;\ \sigma - \mathit{Read}\ r - \mathit{Write}\ r - \mathit{Alloc}\ r\ ;\ \mathit{cutT}\ r\ \gamma}$$

TyLetR

Get to it already!

```
Distinct :: % ~> % ~> @
```

$$\frac{r \notin \Delta \qquad \Delta \mid r \vdash \overline{w_j : \tau_j} \text{ well-formed} \qquad \Delta,\ r : \% \mid \Gamma,\ \overline{w_j : \tau_j} \vdash t : \tau \ ; \sigma \ ; \gamma \qquad \Delta \vdash \tau : * \qquad r \notin fv(t)}{\Delta \mid \Gamma \vdash \textbf{letregion}\ r\ \textbf{with}\ \{\overline{w_j : \tau_j}\}\ \textbf{in}\ t : \tau \ ; \sigma - \mathit{Read}\ r - \mathit{Write}\ r - \mathit{Alloc}\ r \ ; \ \mathit{cutT}\ r\ \gamma}$$

TyLetR

```
λ r1. letregion r2 with {w : Distinct r1 r2} in .. ✓
λ r1. letregion r2 with {w : Distinct r2 r2} in .. ✗
λ r1. letregion r2 with {w : Distinct r1 r1} in .. ✗
```

In Layman's Terms...

$$\frac{}{\Delta \mid r \vdash \emptyset \text{ well-formed}} \textsc{WfEmpty}$$

$$\frac{\Delta \mid r \vdash \Gamma \text{ well-formed} \qquad \mathit{Const}\ r \notin \Gamma}{\Delta \mid r \vdash \Gamma,\ w : \mathit{Mutable}\ r \text{ well-formed}} \textsc{WfMutable}$$

$$\frac{\Delta \mid r \vdash \Gamma \text{ well-formed} \qquad \mathit{Mutable}\ r \notin \Gamma}{\Delta \mid r \vdash \Gamma,\ w : \mathit{Const}\ r \text{ well-formed}} \textsc{WfConst}$$

$$\frac{r_2 : \% \in \Delta \qquad r_1 \notin \Delta}{\Delta \mid r_1 \vdash \Gamma,\ w : \mathit{Distinct}\ r_1\ r_2 \text{ well-formed}} \textsc{WfDistinct1}$$

$$\frac{r_2 : \% \in \Delta \qquad r_1 \notin \Delta}{\Delta \mid r_1 \vdash \Gamma,\ w : \mathit{Distinct}\ r_2\ r_1 \text{ well-formed}} \textsc{WfDistinct2}$$

Multi-way Distinctness

```
Distinct3 :: % ⇝ % ⇝ % ⇝ @
Distinct4 :: % ⇝ % ⇝ % ⇝ % ⇝ @
...
```

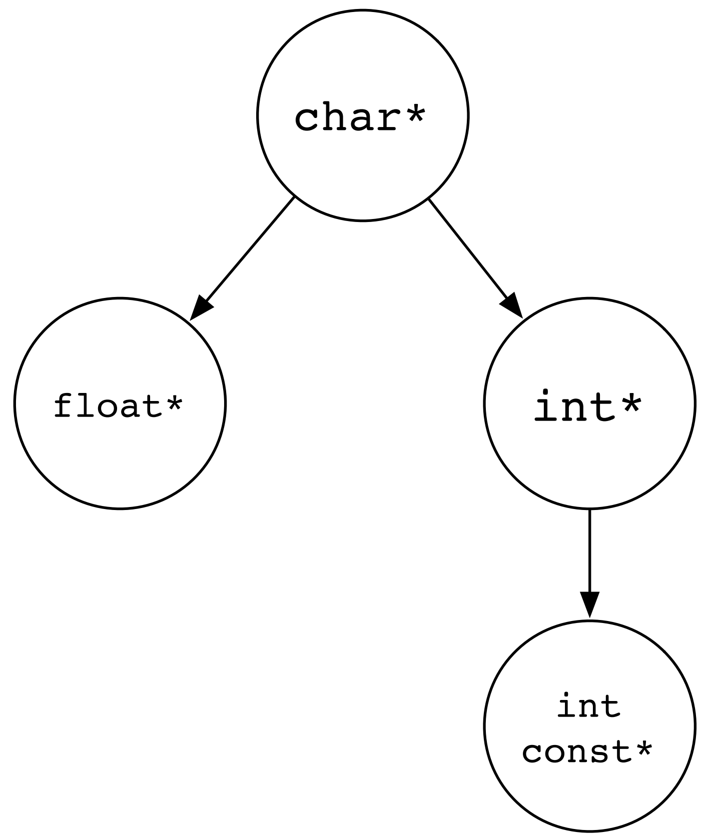
```
Λ r1 r2 r3.
letregions r4 with { w : Distinct4 r1 r2 r3 r4 }
in ...
```

```
letregions r1 r2 r3 r4 with { w : Distinct4 r1 r2 r3 r4 }
in ...
```

Using distinctness for low-level optimisation

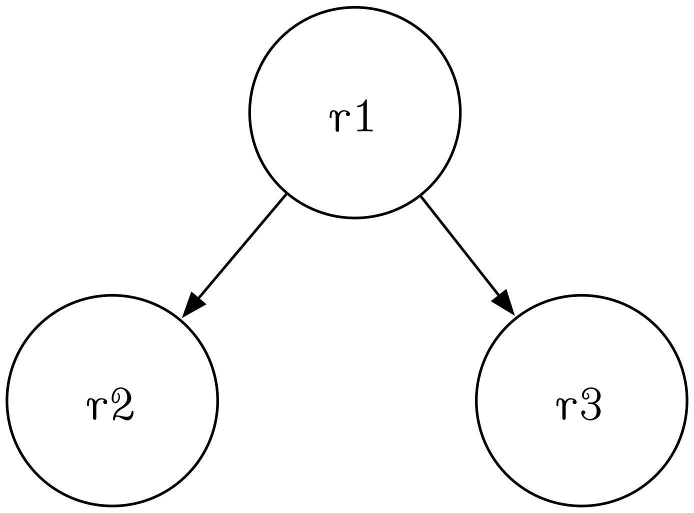
```
%a = load i32 %x
store %y %a
%b = load i32 %x
```

LLVM Alias Analysis Metadata



LLVM Alias Analysis Metadata

```
{r1, r2, r3}

Distinct r2 r3
```




```
!0 = metadata !{metadata !"r1", null, i32 0}

!1 = metadata !{metadata !"r2", metadata !0, i32 0}

!2 = metadata !{metadata !"r3", metadata !0, i32 1}
```

Translation from Witnesses to Metadata

S : set of regions
A : alias relation in DDC
A’ : alias relation in LLVM

Translation from Witnesses to Metadata

S : set of regions
A : alias relation in DDC
A’ : alias relation in LLVM

Safe

$$\forall\ r_1, r_2 \in \mathcal{S}\ (r_1 \neq r_2).\ (r_1, r_2) \in \mathcal{A} \implies (r_1, r_2) \in \mathcal{A}'$$

Translation from Witnesses to Metadata

S : set of regions
A : alias relation in DDC
A' : alias relation in LLVM

Safe

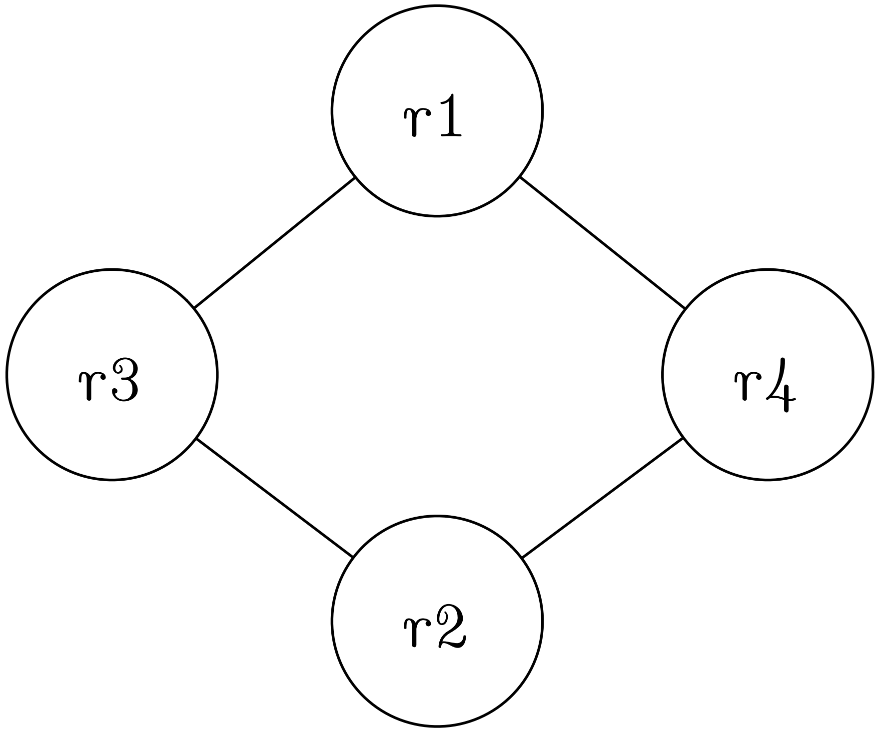
$$\forall\ r_1, r_2 \in \mathcal{S}\ (r_1 \neq r_2).\ (r_1, r_2) \in \mathcal{A} \implies (r_1, r_2) \in \mathcal{A}'$$

Optimal

$$\forall \mathcal{R}.\ safe(\mathcal{R}) \implies \mathcal{R} \geq \mathcal{A}'$$

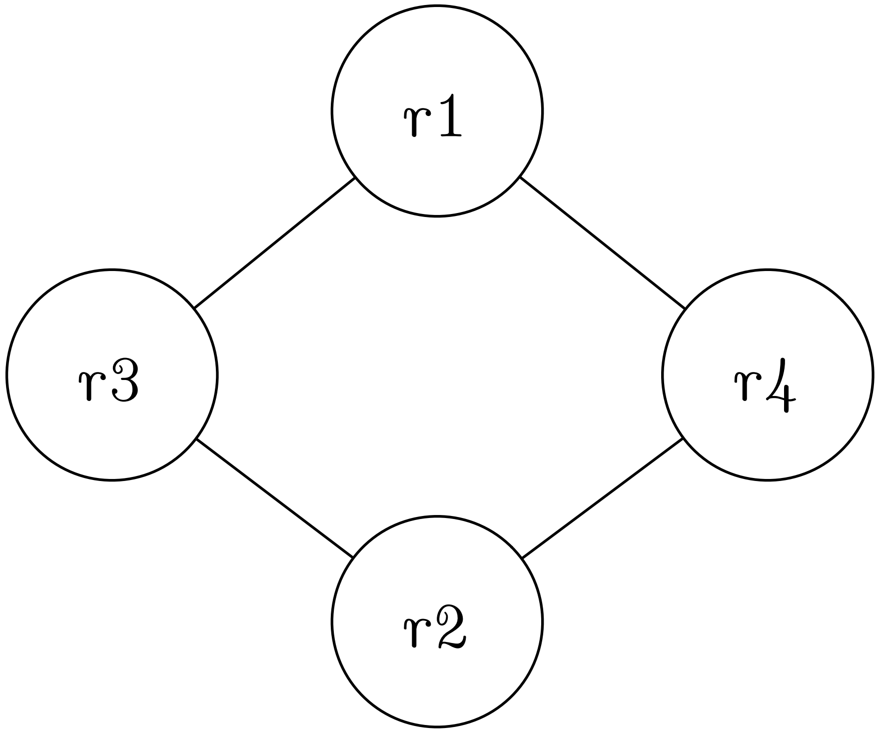
DDC Alias as Adjacency

$$S = \{r1,\ r2,\ r3,\ r4\}$$
$$D = \{Distinct\ r1\ r2,\ Distinct\ r3\ r4\}$$



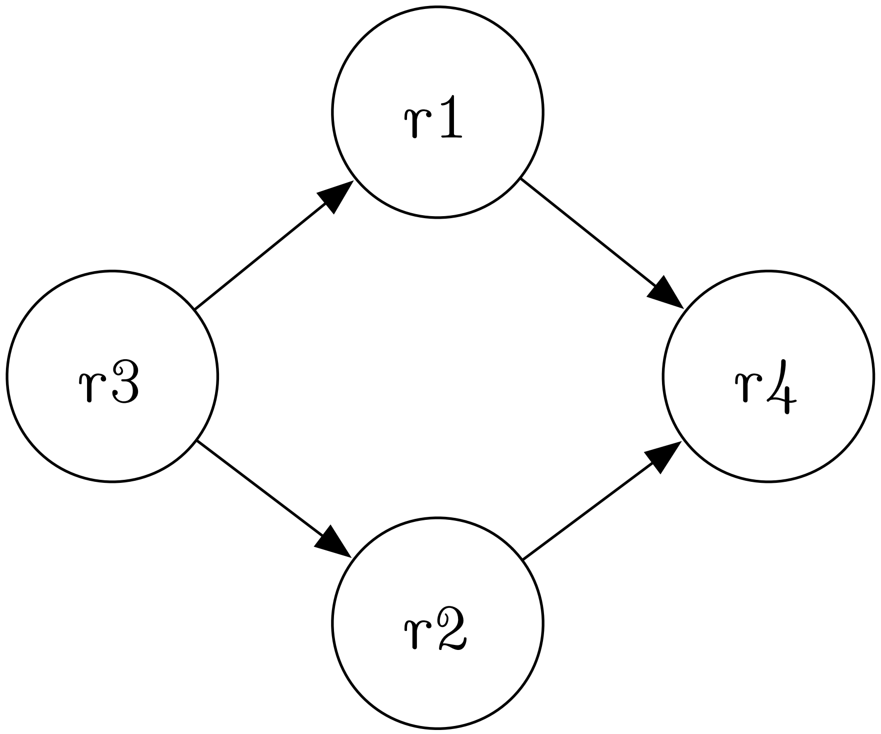
DDC Alias as Adjacency

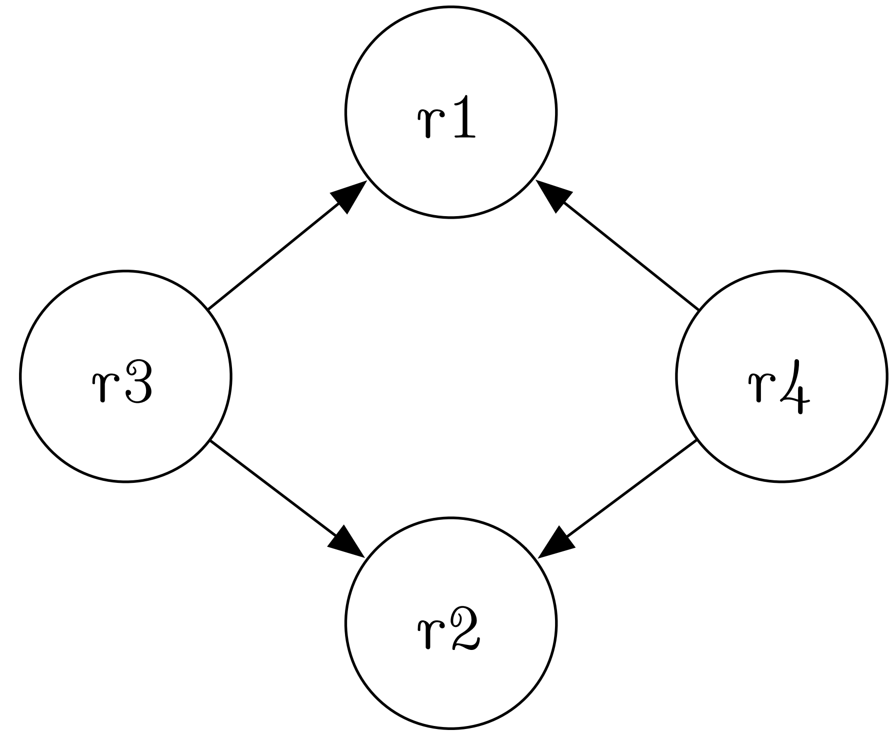
$$S = \{r1, r2, r3, r4\}$$
$$D = \{Distinct\ r1\ r2,\ Distinct\ r3\ r4\}$$



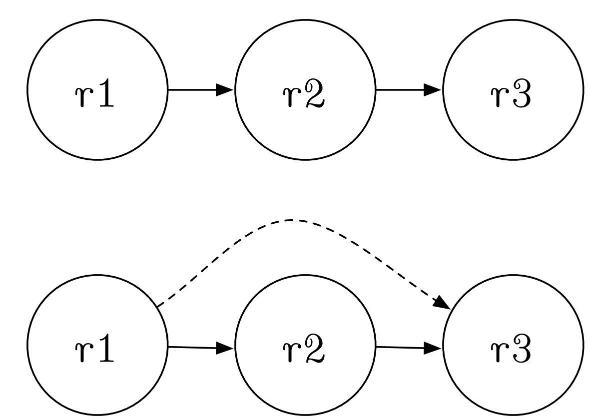
Orientation and Partition
preserving safety and optimality

Orientation



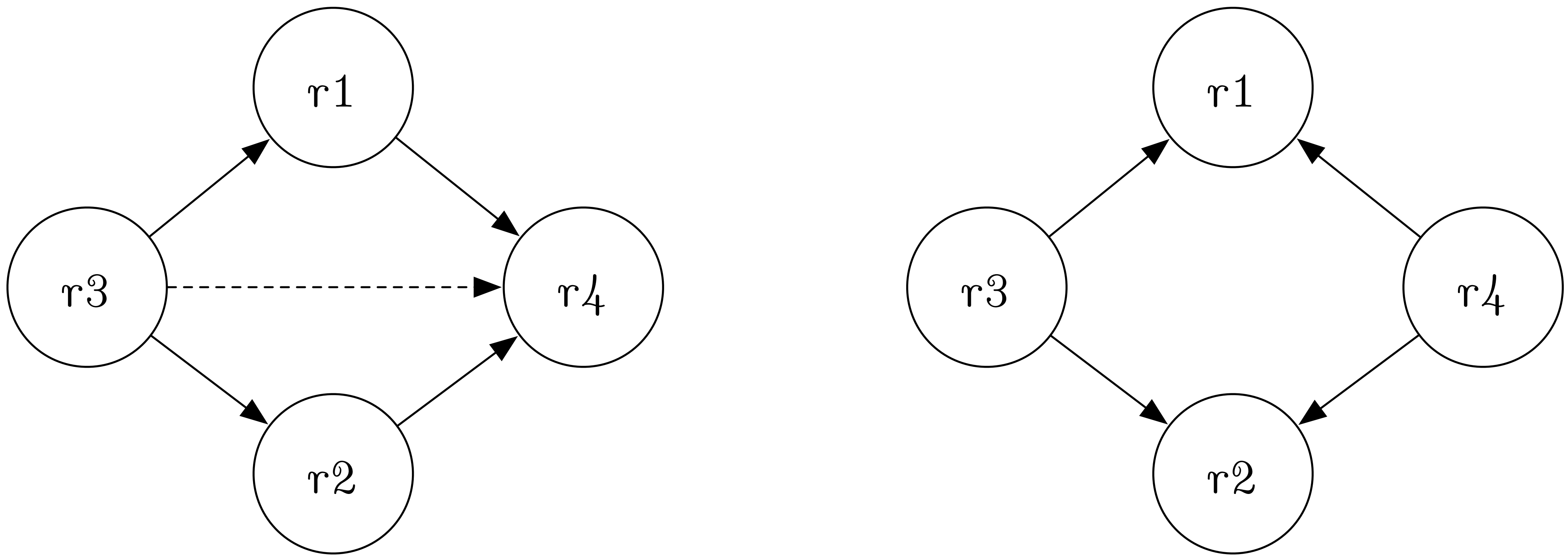


Measure of Aliasing
=
Number of edges in the transitive closure of D



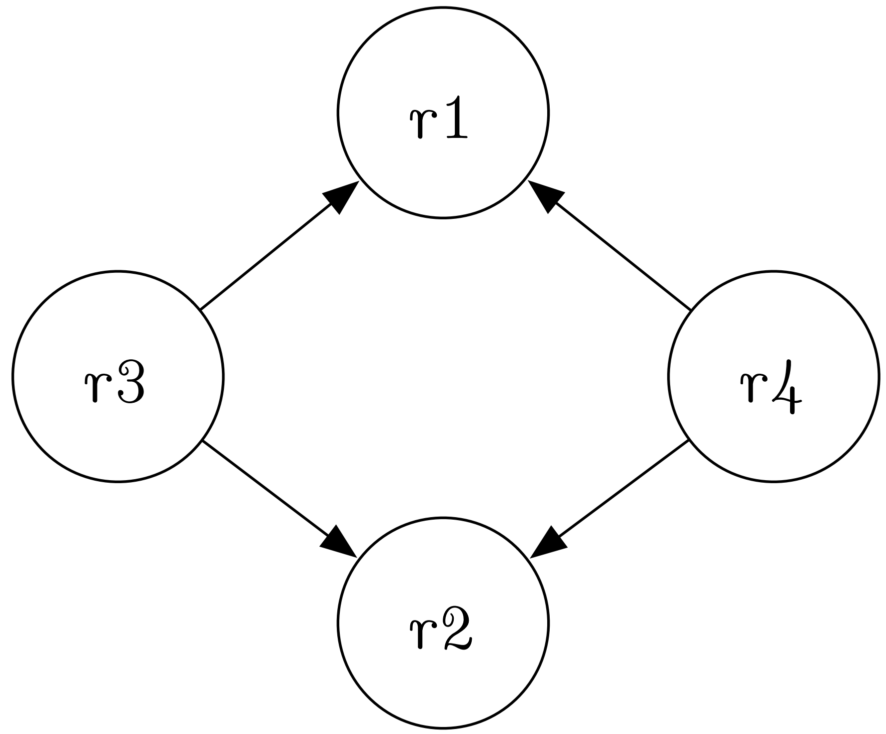
Transitive closure

Orientation



Bruteforce = combinatorial explosion

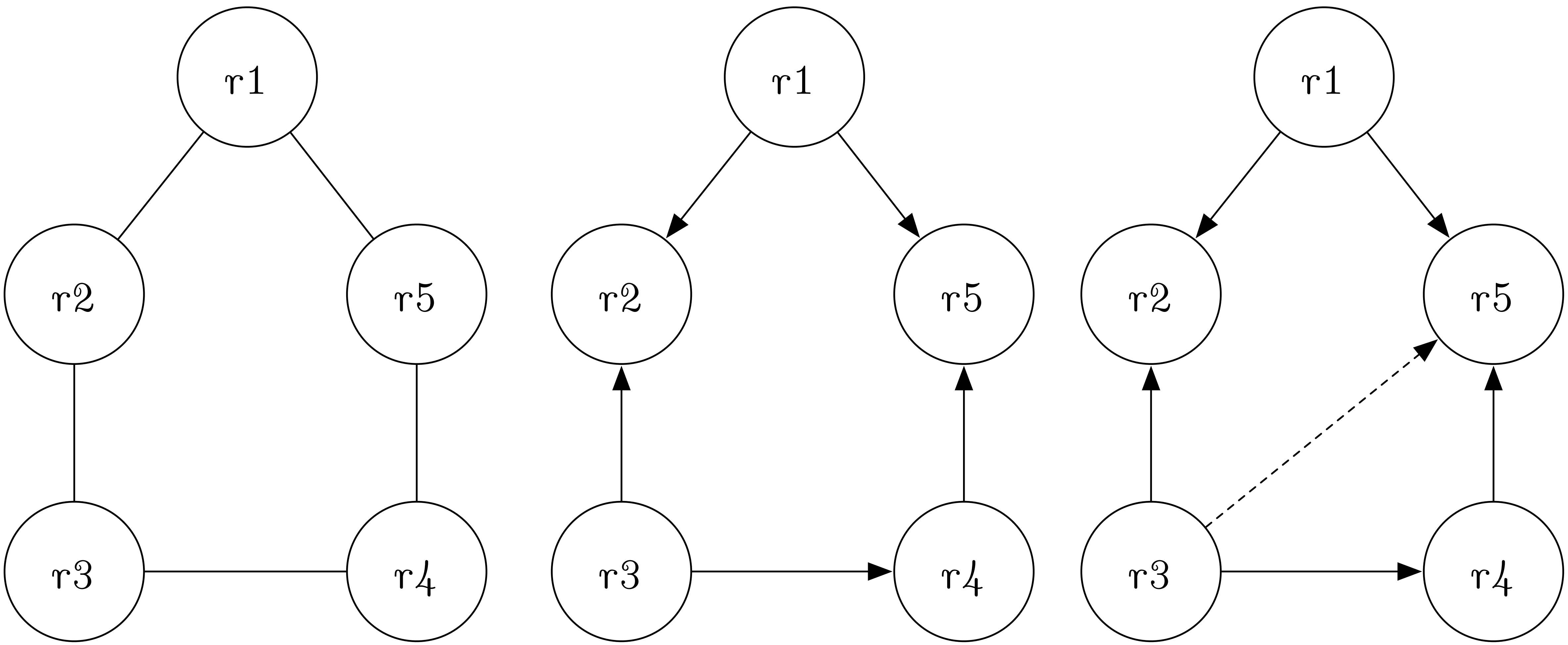
Orientation - Happy Case



Transitive Orientation

O(n)

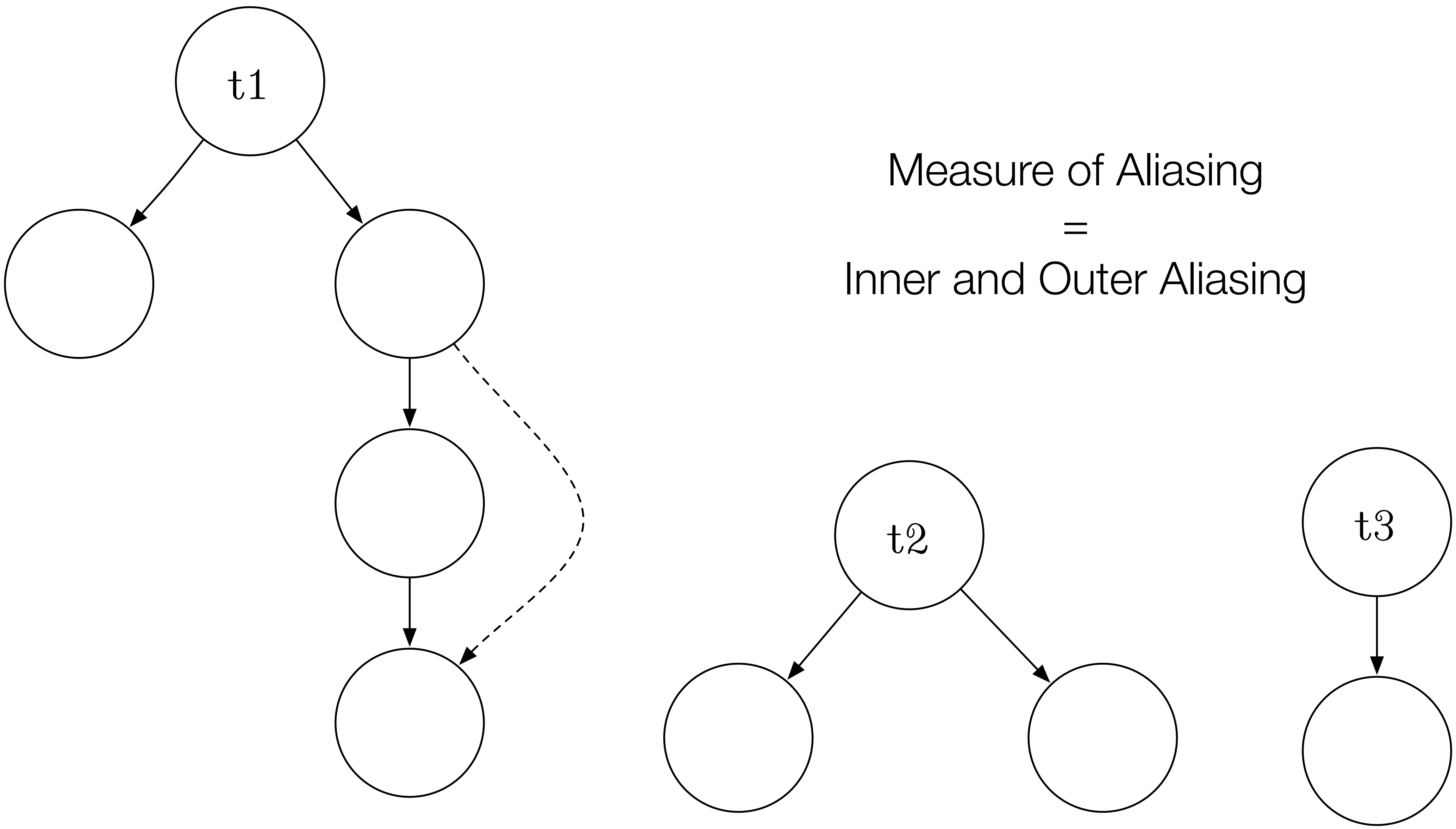
Orientation - Not So Happy Case



Orientation - Not So Happy Case

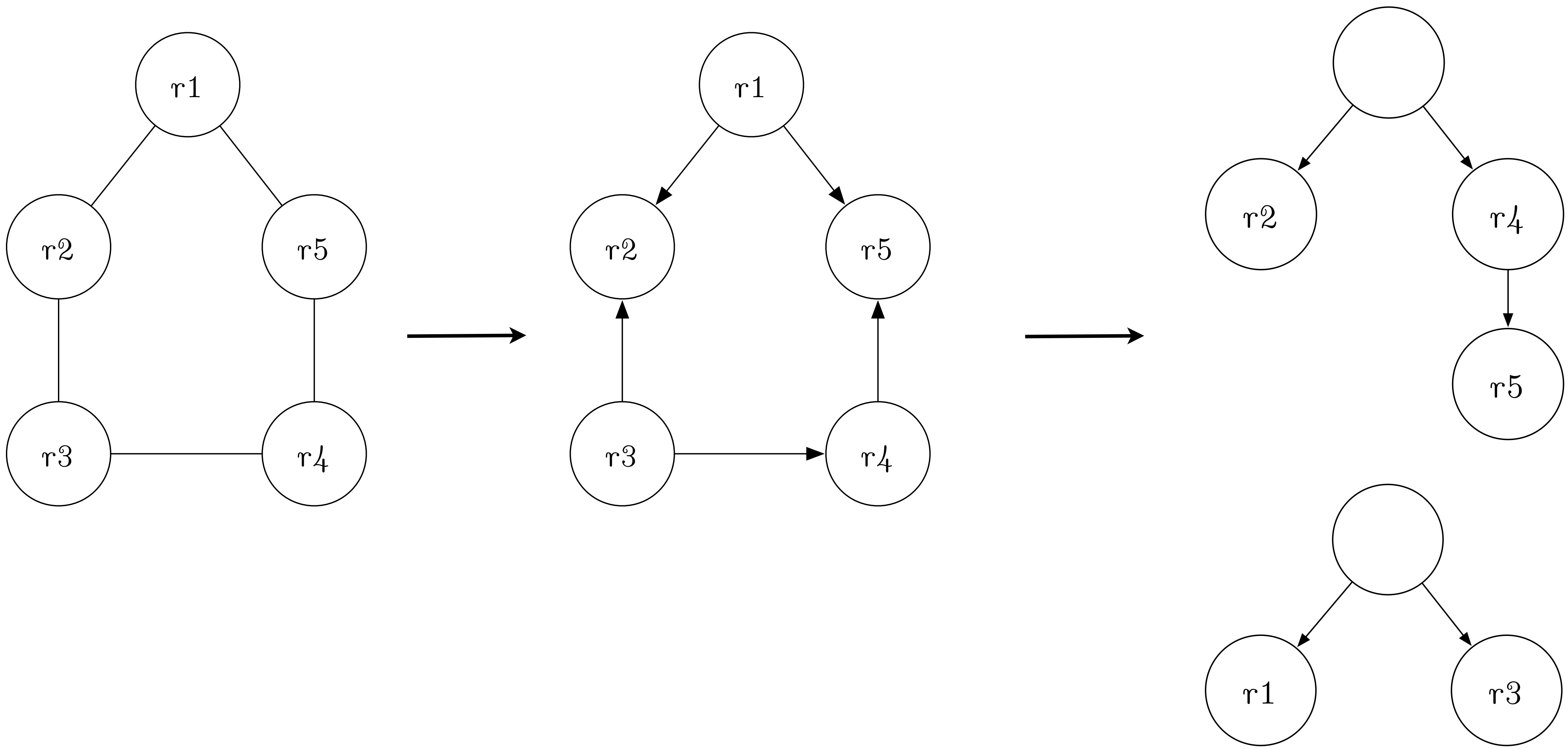
- Bruteforce with a threshold -- reasonable compromise
- Minimum “comparability completion” -- NP hard
  → Minimal comparability completion?

Partition

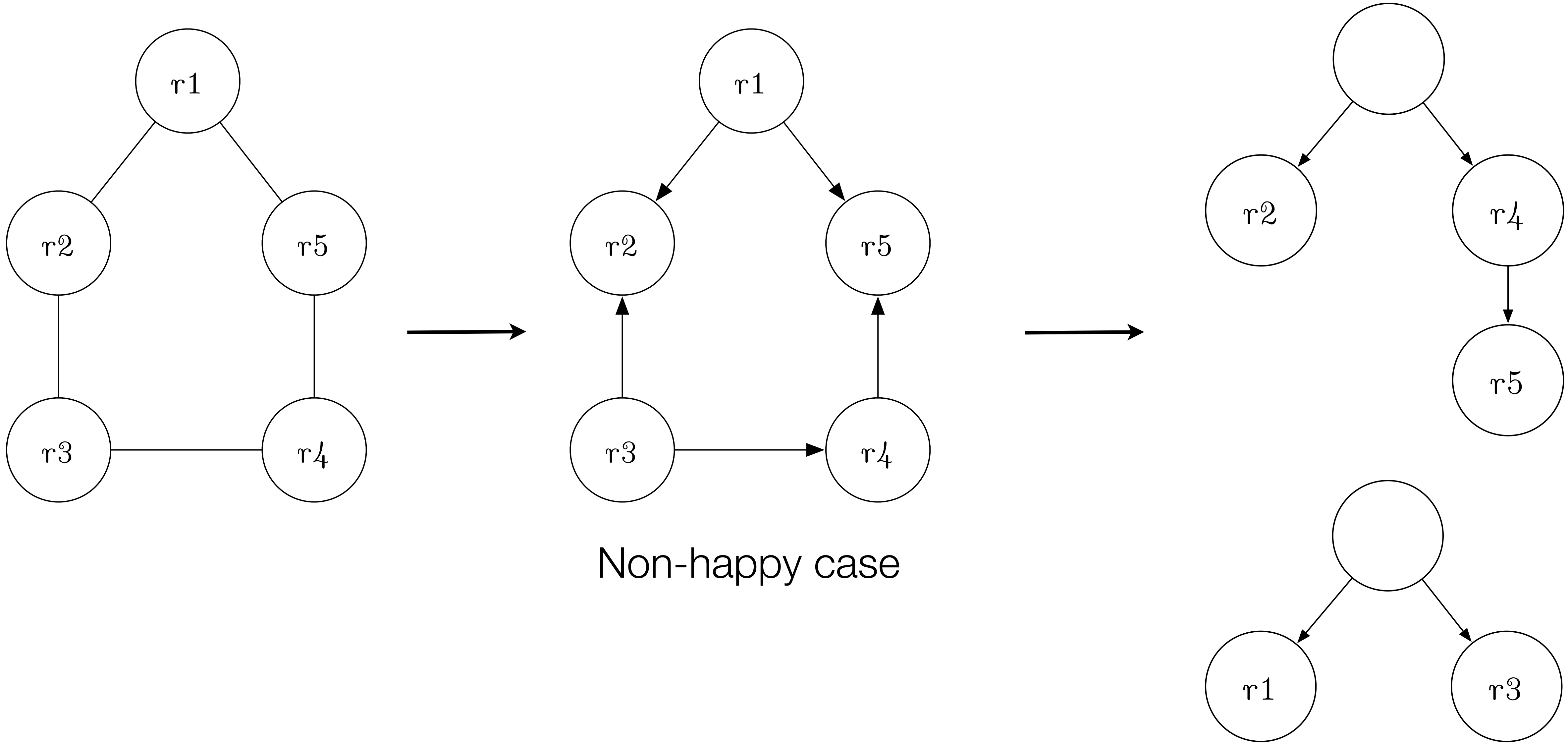



```
(outer-aliasing) + (inner-aliasing)
= (|tree1| * (|tree2| + |tree3|) + |tree2| * |tree3|) + (3 + 0 + 0)
= 17.
```

Example



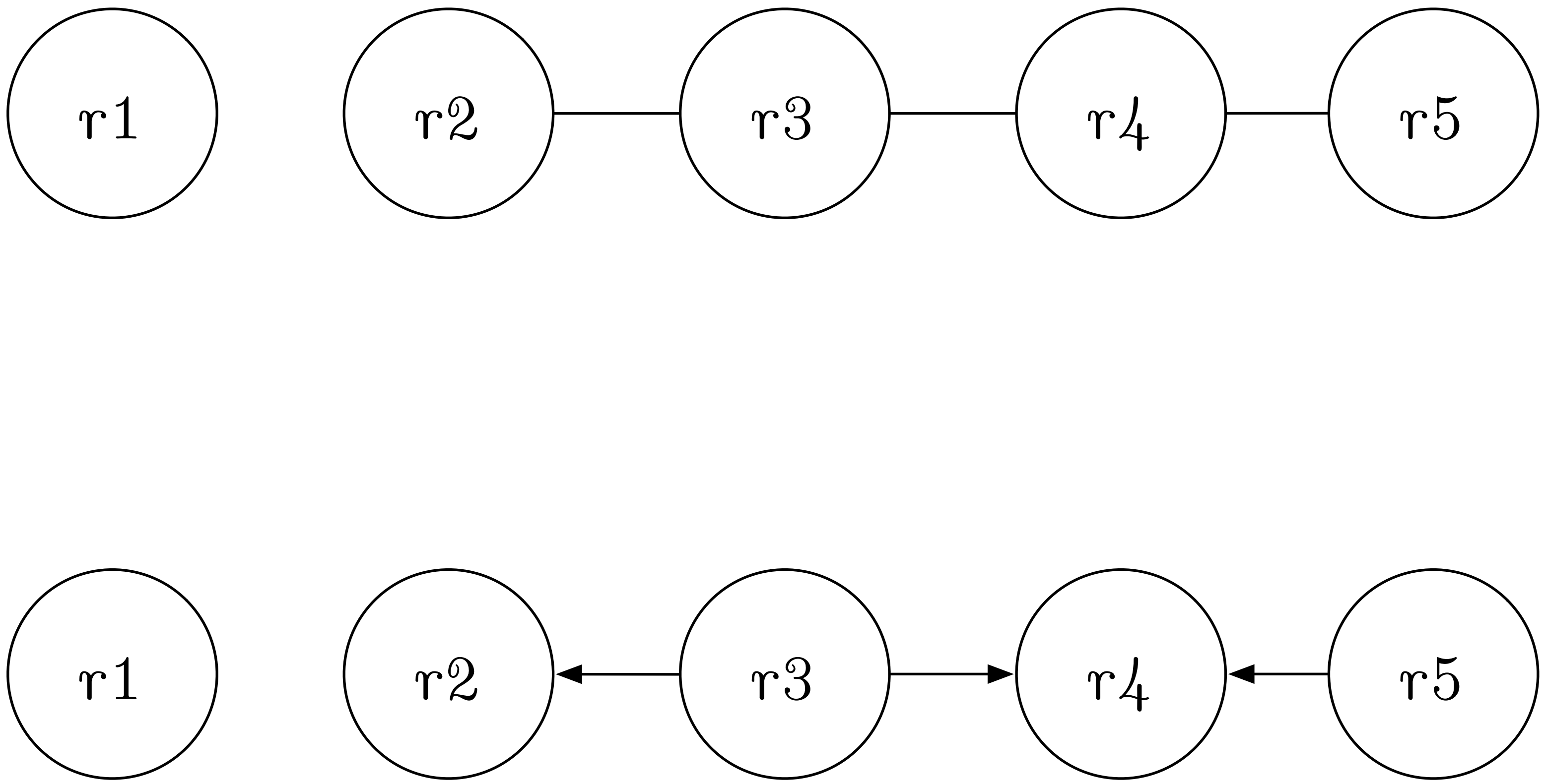
Example



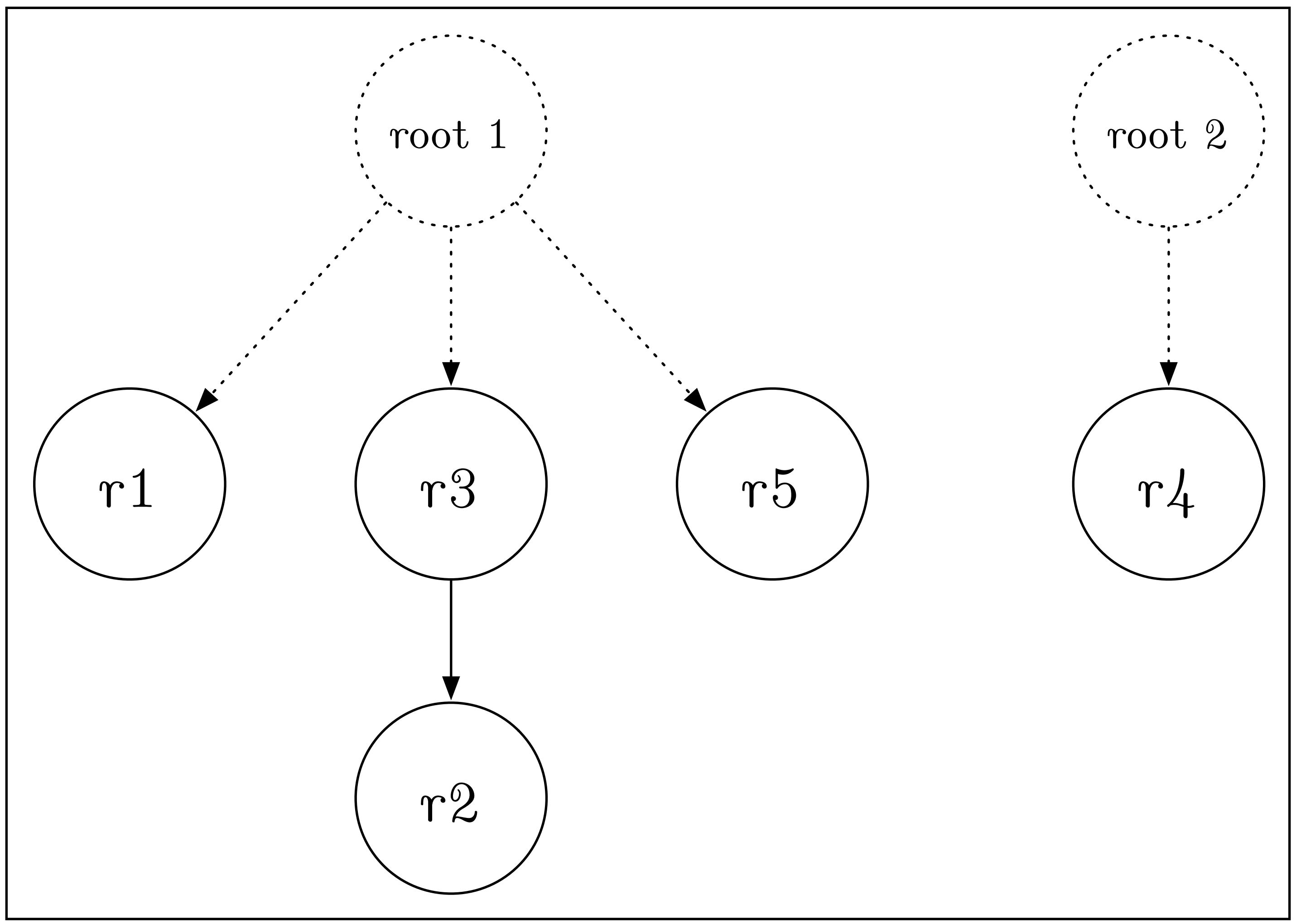
Non-happy case

Bruteforce -- but it's okay

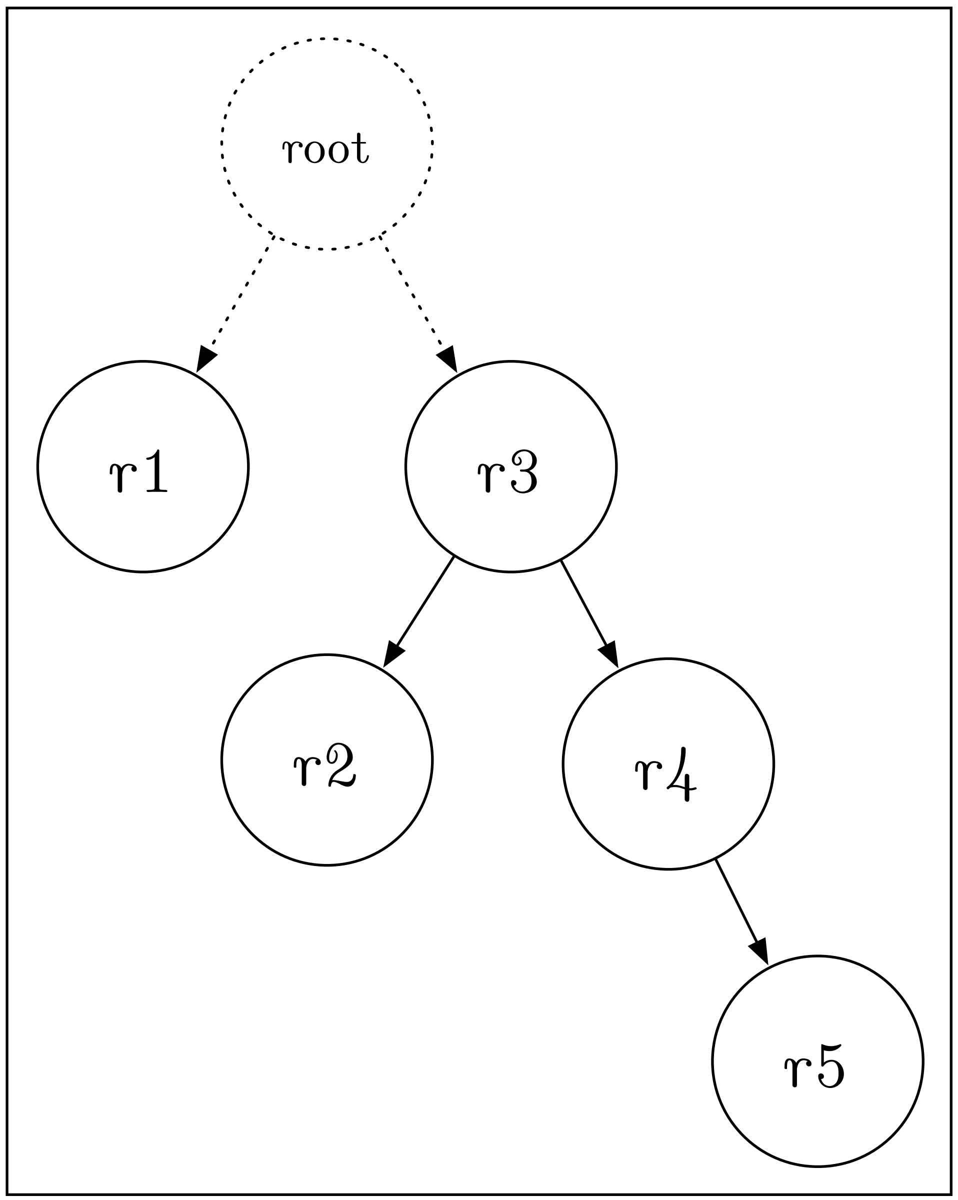
The price



The price



Aliasing = 5



Aliasing = 4

Get to the results!

```
peek# :: [r : %].[t : *].Ptr# r t → Nat# → t
poke# :: [r : %].[t : *].Ptr# r t → Nat# → t → Void#
```

```
-- Reads a pointer to a memory object in region r1, with offset 3
peek# [r1] [Obj]  x 3#;

-- Updates a pointer to an Int in region r2 with the new value 5
poke# [r2] [Int#] y 5 0#;
```

Results - GVN

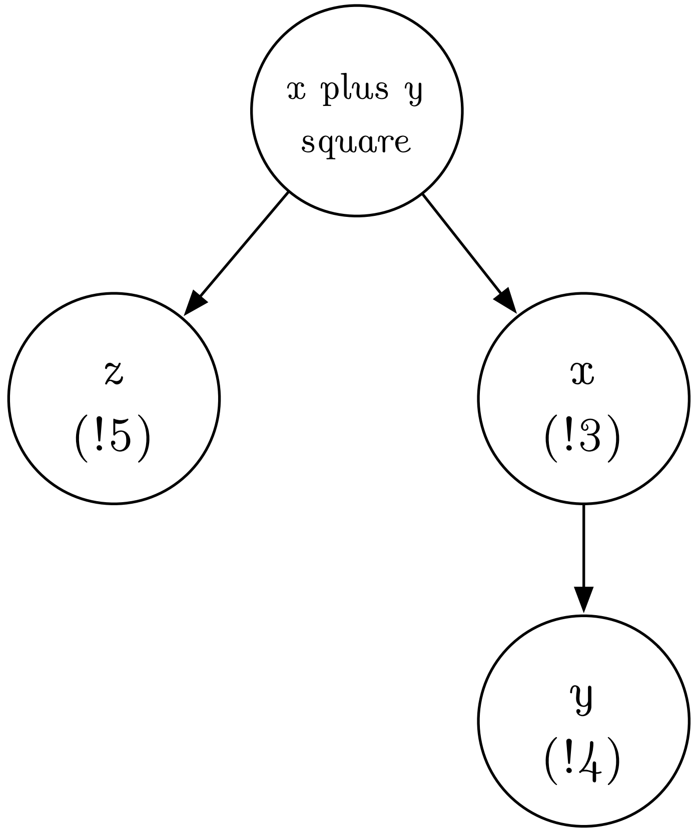
```
x_plus_y_square
    [rx ry rz : %]                              -- Takes three regions
    <w1 : Distinct rx rz>                       -- Two witnesses
    <w2 : Distinct ry rz>
    (x : Ptr# rx Int#)                          -- Three value arguments
    (y : Ptr# ry Int#)                          -- of type pointer to int
    (z : Ptr# rz Int#)
    : Int#
 = do { xval1 = peek# [rx] [Int#] x 0#;         -- Compute (x + y)
        yval1 = peek# [ry] [Int#] y 0#;
        a     = add# [Int#] xval1 yval1;

        poke# [rz] [Int#] z 0# a;               -- Modify z

        xval2 = peek# [rx] [Int#] x 0#;         -- Compute (x + y) again
        yval2 = peek# [ry] [Int#] y 0#;
        b     = add# [Int#] xval2 yval2;

        mul# [Int#] a b;                        -- Result is (x + y)^2
     };
```

Results - GVN

```
define external ccc i64 @x_plus_y_square(i64*  %x, i64*  %y, i64*  %z) align 8  {
l6.entry:
        (...Compute the pointer offset for peek# x...)
        %xval1         = load i64* %xval1.ptr,     !tbaa !3
        (...Compute the pointer offset for peek# y...)
        %yval1         = load i64* %yval1.ptr,     !tbaa !4
        %a             = add i64 %xval1, %yval1

        (...Compute the pointer offset for poke# z...)
        store i64 %a, i64* %_v9.ptr,    !tbaa !5                      -- modify z

        (...)
        %xval2         = load i64* %xval2.ptr,     !tbaa !3           -- redundant load
        (...)
        %yval2         = load i64* %yval2.ptr,     !tbaa !4           -- redundant load
        %b             = add i64 %xval2, %yval2

        %_v10          = mul i64 %a, %b
        ret i64 %_v10
}
```

Results - GVN

```
!5 = metadata !{metadata !"x_plus_y_square_rz", metadata !2, i32 0}
!4 = metadata !{metadata !"x_plus_y_square_ry", metadata !3, i32 0}
!3 = metadata !{metadata !"x_plus_y_square_rx", metadata !2, i32 0}
!2 = metadata !{metadata !"x_plus_y_square_ROOT_1", null, i32 1}
```



Results - GVN

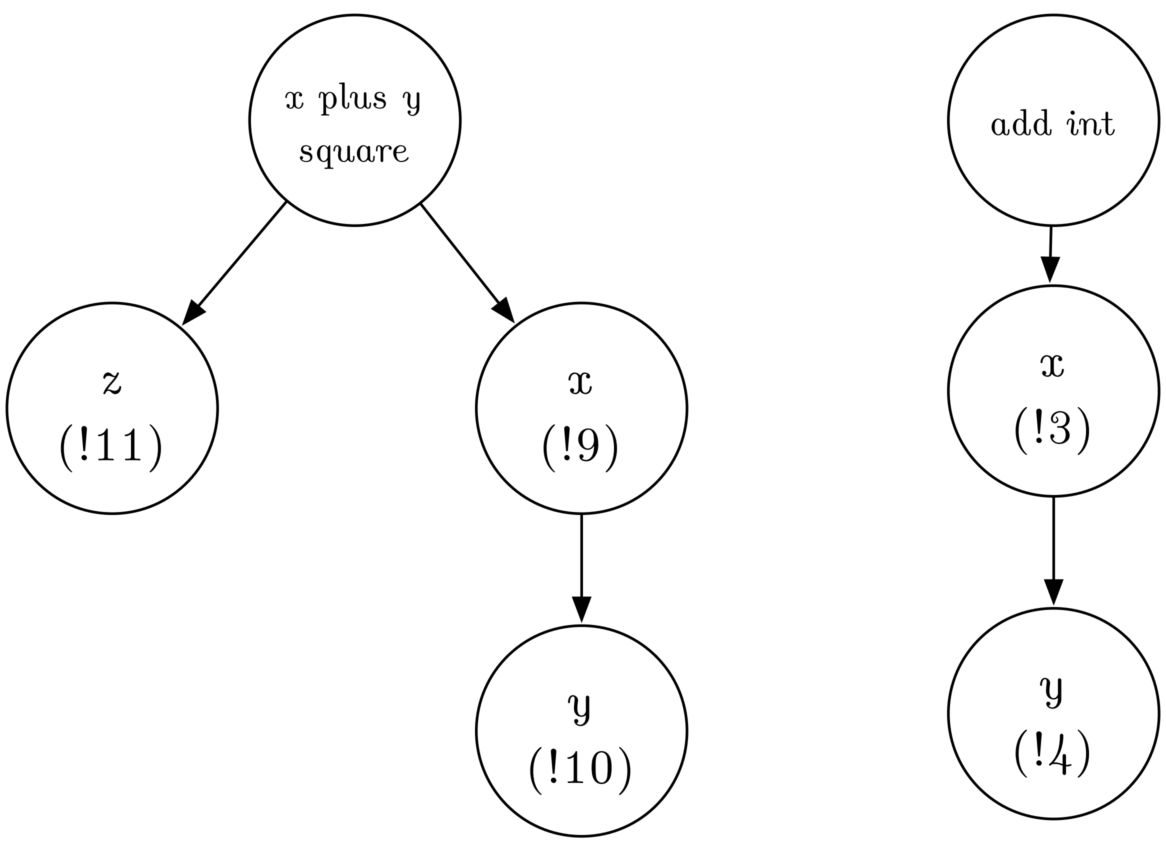
```
define i64 @x_plus_y_square(i64* %x, i64* %y, i64* %z) align 8 {
l6.entry:
        (...Compute the pointer offset for peek# x...)
        %xval1 = load i64* %xval1.ptr, !tbaa !0
        (...)
        %yval1 = load i64* %yval1.ptr, !tbaa !2
        %a = add i64 %xval1, %yval1

        (...Compute the pointer offset for poke# z...)
        store i64 %a, i64* %_v9.ptr, !tbaa !3                    -- modify z

        %_v10 = mul i64 %a, %a
        ret i64 %_v10
}

!0 = metadata !{metadata !"x_plus_y_square_rx", metadata !1, i32 0}
!1 = metadata !{metadata !"x_plus_y_square_ROOT_1", null, i32 1}
!2 = metadata !{metadata !"x_plus_y_square_ry", metadata !0, i32 0}
!3 = metadata !{metadata !"x_plus_y_square_rz", metadata !1, i32 0}
```

Without metadata:

```
GVN: load i64 %xval2 is clobbered by   store i64 %a, i64* %_v9.ptr
GVN: load i64 %yval2 is clobbered by   store i64 %a, i64* %_v9.ptr
```

Dependence on inlining and let-floating

```
define external ccc i64 @x_plus_y_square(i64*  %x, i64*  %y, i64*  %z) align 8  {
l12.entry:
        %a             = call i64 @add_int (i64* %x, i64* %y)
        (...)
        %_v14.addr2    = add i64 %_v13.addr1, 0
        (...)
        store i64 %a, i64* %_v15.ptr,    !tbaa !11
        %b             = call i64 @add_int (i64* %x, i64* %y)
        (...)
}
```



Results - LICM

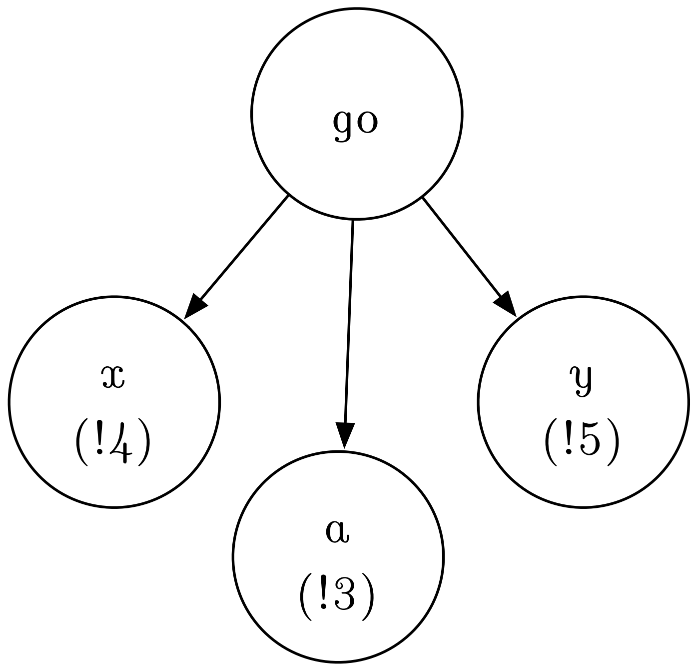
```
go  [ra rx ry : %]                              -- Takes three regions
    <w : Distinct3 ra rx ry>                    -- that are pair-wise distinct
    (a : Ptr# ra Nat#)                          -- Three pointers to natural numbers
    (x : Ptr# rx Nat#)
    (y : Ptr# ry Nat#)
    (i : Nat#)                                  -- Counter
    : Nat#
= case i of {
    42# -> i;
    _   ->
      do { yval      = peek# [ry] [Nat#] y 0#;  -- Loop invariant computation
           yplustwo = add# [Nat#] yval 2#;

           poke# [rx] [Nat#] x 0# yplustwo;     -- Loop invariant computation

           poke# [ra] [Nat#] a i i;             -- Loop-dependant computation

           nexti    = add# [Nat#] i 1#;
           go [ra] [rx] [ry] <w> a x y nexti;   -- Tail recursion
         };
  };
```

Results - LICM

```
define external ccc i64 @go(i64*  %a, i64*  %x, i64*  %y, i64  %i) align 8  {
l6.entry:
        switch i64 %i, label %l9.default [ i64 42,label %l7.alt ]
l7.alt:
        ret i64 %i
l9.default:
        (...)
        %yval        = load i64* %yval.ptr,    !tbaa !5
        %yplustwo    = add i64 %yval, 2
        (...)
        store i64 %yplustwo, i64* %_v12.ptr,    !tbaa !4
        (...)
        store i64 %i, i64* %_v15.ptr,    !tbaa !3
        %nexti       = add i64 %i, 1
        %_v16        = call i64 @go (i64* %a, i64* %x, i64* %y, i64 %nexti)
        ret i64 %_v16
}
```

Results - LICM

```
!5 = metadata !{metadata !"go_ry", metadata !2, i32 0}
!4 = metadata !{metadata !"go_rx", metadata !2, i32 0}
!3 = metadata !{metadata !"go_ra", metadata !2, i32 0}
!2 = metadata !{metadata !"go_ROOT_1", null, i32 1}
```



Results - LICM

```
define i64 @go(i64* %a, i64* %x, i64* %y, i64 %i) align 8 {
l6.entry:
  (...)

l9.default.lr.ph:                             ; preds = %l6.entry
  (...)
  %yval = load i64* %yval.ptr, !tbaa !0
  %yplustwo = add i64 %yval, 2
  (...)
  br label %l9.default

tailrecurse.l7.alt_crit_edge:                 ; preds = %l9.default
  %split = phi i64 [ %nexti, %l9.default ]
  store i64 %yplustwo, i64* %_v12.ptr
  br label %l7.alt

l7.alt:                                       ; preds = %tailrecurse.l7.alt_crit_edge, %l6.entry
  (...)

l9.default:                                   ; preds = %l9.default.lr.ph, %l9.default
  (...)
  store i64 %i.tr2, i64* %_v15.ptr, !tbaa !2
  (...)
}
```

Results - LICM

Just to make sure..

```
(...)
LICM hoisting to l9.default.lr.ph: %yplustwo = add i64 %yval, 2
(...)
LICM: Promoting value stored to in loop: %_v12.ptr = inttoptr i64
%_v10.addr1 to i64*
```

Integration with Rule-Based Rewriting

```
copyVector  :: [r1 r2 : %]. Vector r1 a → Buffer r2 a
copyVector' :: [r1 r2 : %]. Distinct r1 r2 → Vector r1 a → Buffer r2 a
```

can't get enough of aliasing control? http://www.cse.unsw.edu.au/~tranma/report.pdf