an efficient data structure for Must-Alias Analysis

George Kastrinis

George Balatsouras
Kostas Ferles
Nefeli Prokopaki
Yannis Smaragdakis

an efficient data structure for Must-Alias Analysis

```
x = y
y.f = z
```

aliasing expressions refer to the same memory

```
x ~ y
y.f ~ z
x.f ~ z
```

an efficient data structure for Must-Alias Analysis

Find the LEGO

Truth

May (Over)

Must (under)


I don't always report aliasing...
But when I do, I'm sure!

Insights for efficiency

Must-Alias: an Equivalence Relation

$$\frac{x \sim y \qquad y \sim z}{x \sim z}$$

Must-Alias: an Equivalence Relation

$$\frac{\begin{array}{c} x \sim y \\ y \sim z \end{array}}{x \sim z}$$

⚠️ **N aliasing elements** $\rightarrow$ **N^2 pairs**

Implicit access path extension

$$\frac{x \sim y}{\begin{array}{c} x.f \sim y.f \\ x.g \sim y.g \\ x.g.k \sim y.g.k \end{array}}$$

⚠️ **c fields, k max length** → **$\Omega(c^k)$ access paths**

May-Alias is NOT an equivalence relation

```
x = a
if (...)
  y = c
else
  y = a
```

$$\frac{x \sim a \qquad y \sim c \qquad y \sim a}{x \sim y \qquad x \; ? \; c}$$

how much?

an **efficient** data structure

for Must-Alias Analysis

Datalog Naive (the old)

Explicitly represent alias pairs

Explicitly extend access paths (max len)

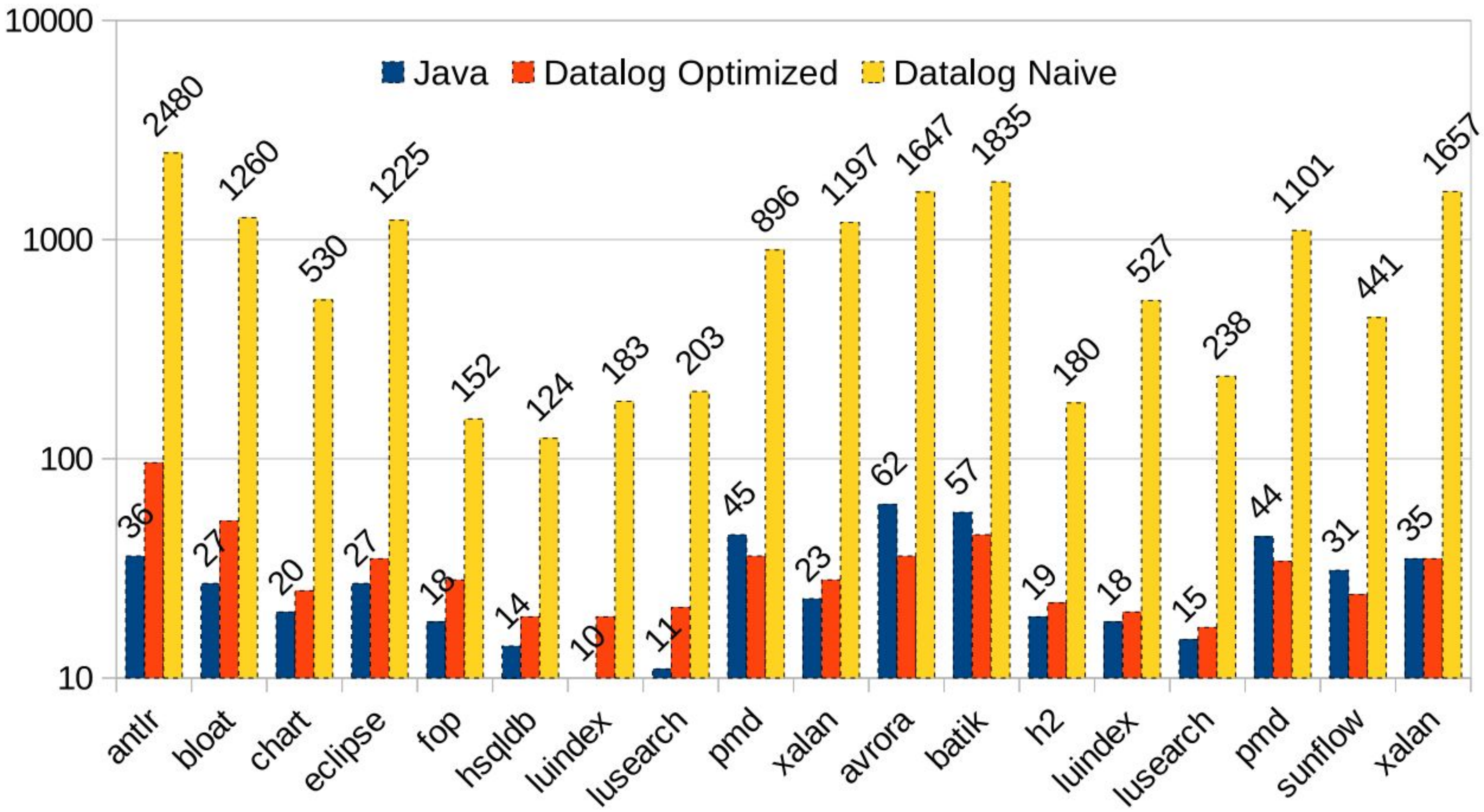
Java

Data structure for implicit representation of both points

Datalog Opt

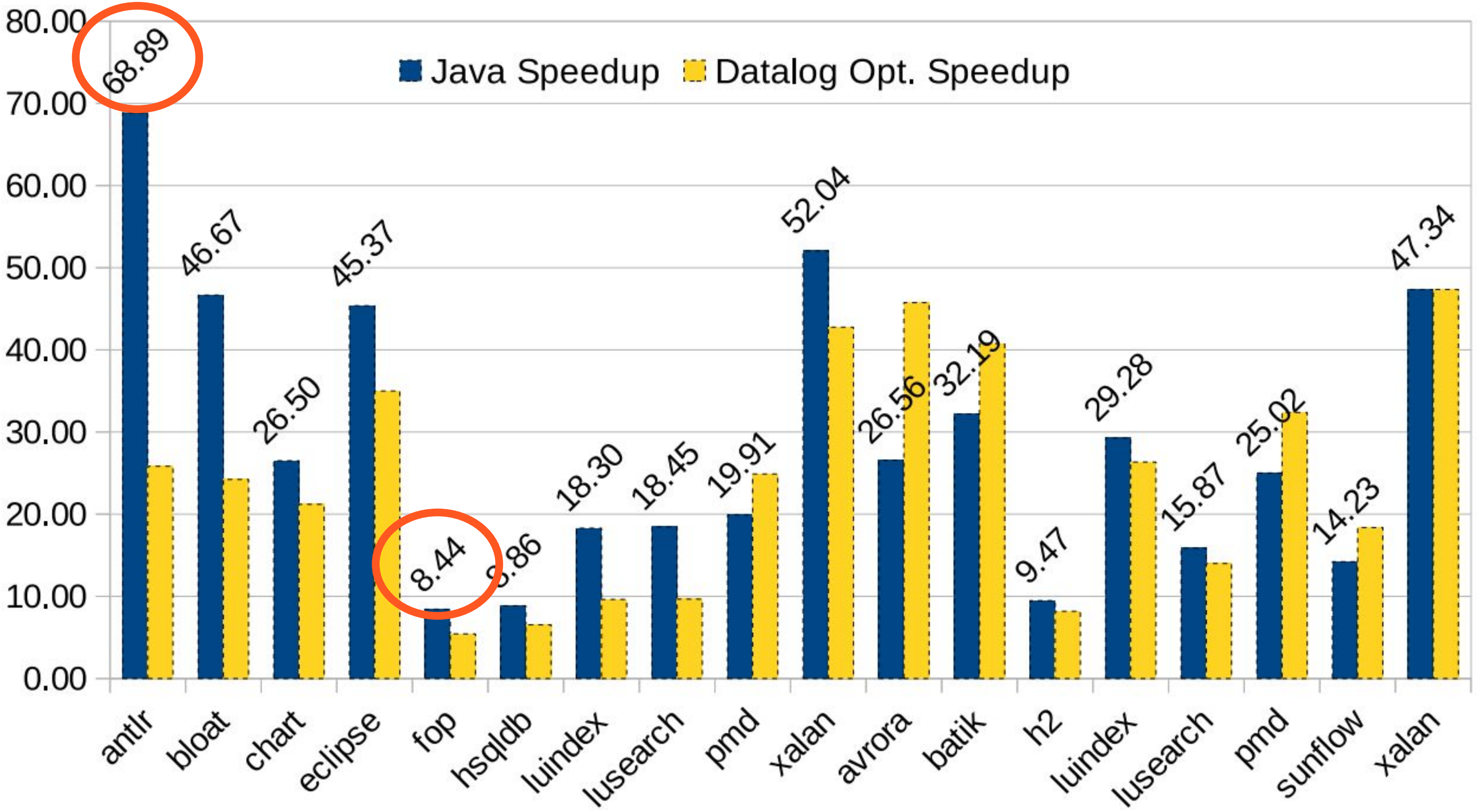
Simulated in a purely declarative setup


Java
Datalog Opt
Datalog Naive



10000
1000
100
10
Java
Datalog Optimized
Datalog Naive
36
2480
27
1260
20
530
27
1225
18
152
14
124
10
183
11
203
45
896
23
1197
62
1647
57
1835
19
180
18
527
15
238
44
1101
31
441
35
1657
antlr
bloat
chart
eclipse
fop
hsqldb
luindex
lusearch
pmd
xalan
avrora
batik
h2
luindex
lusearch
pmd
sunflow
xalan

Time

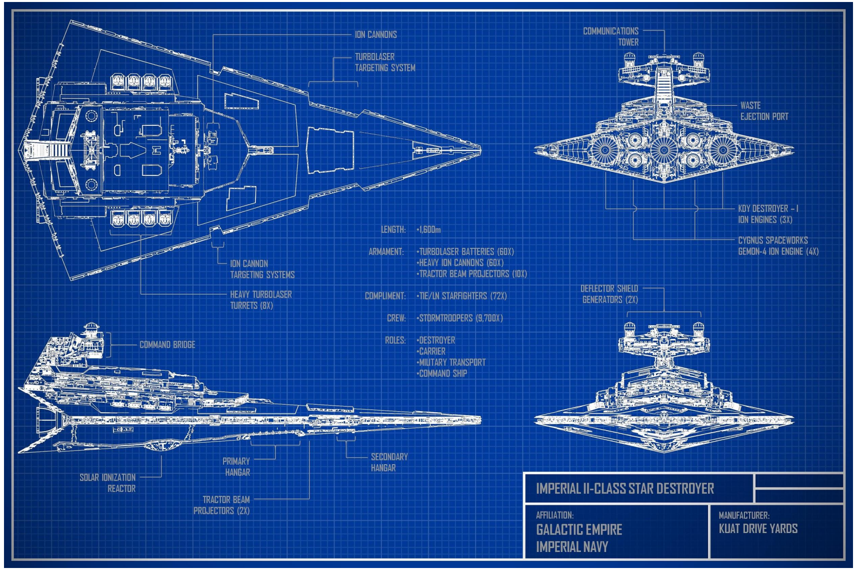


Java Speedup
Datalog Opt. Speedup
80.00
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0.00
68.89
46.67
26.50
45.37
8.44
18.30
18.45
19.91
52.04
26.56
32.19
9.47
29.28
15.87
25.02
14.23
47.34
antlr
bloat
chart
eclipse
fop
hsqldb
luindex
lusearch
pmd
xalan
avrora
batik
h2
luindex
lusearch
pmd
sunflow
xalan

Speedup

how?

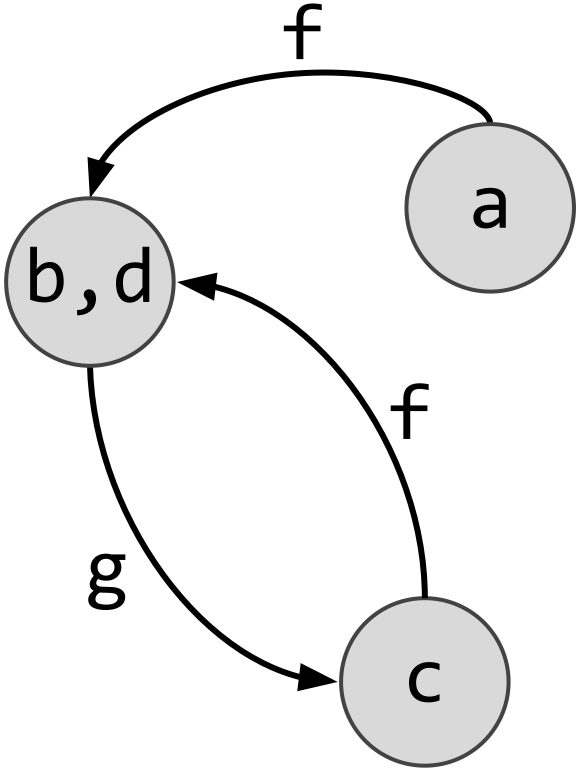
an efficient data structure for Must-Alias Analysis



Alias Graph

- **directed** graph
- invent abstract objects (**nodes**) for what a variable points to
- **edges** represent fields
- access paths are **paths** in the graph
- paths to same node ⟶ **aliases**
- **merge** aliasing variables

- one graph per (instruction x calling context)
- copy from one instruction to the next and apply semantics
- up until a fixpoint



b.g ~ c

d.g ~ c

a.f ~ b

c.f ~ b

a.f ~ c.f

a.f.g ~ c

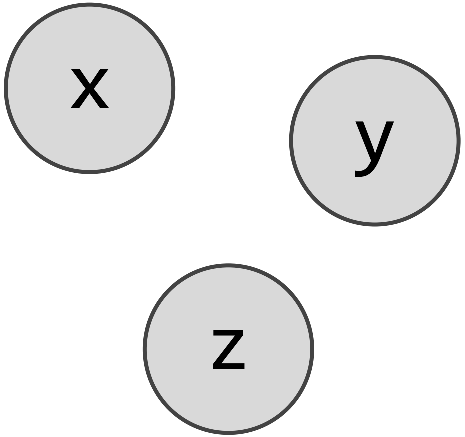
a.f.g.f ~ b

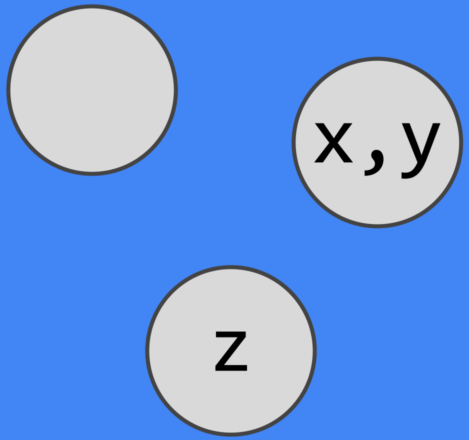
a.f.g ~ b.f

etc

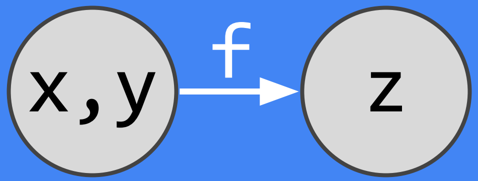
Operations

x = y

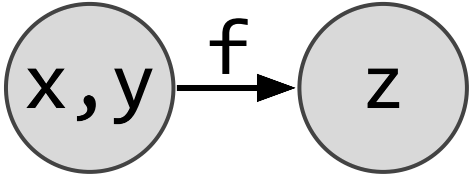


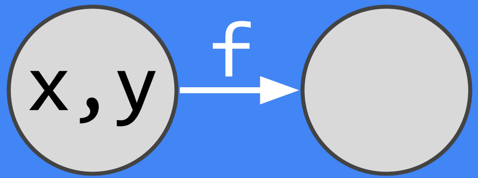


x.f = z

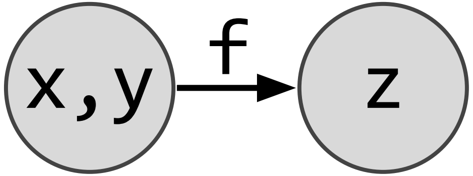


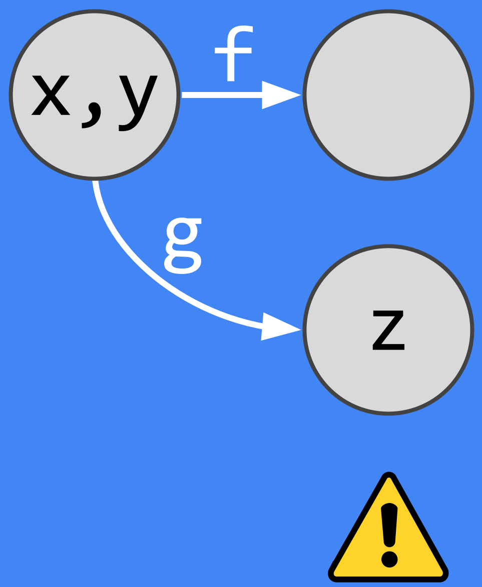
z = y.g

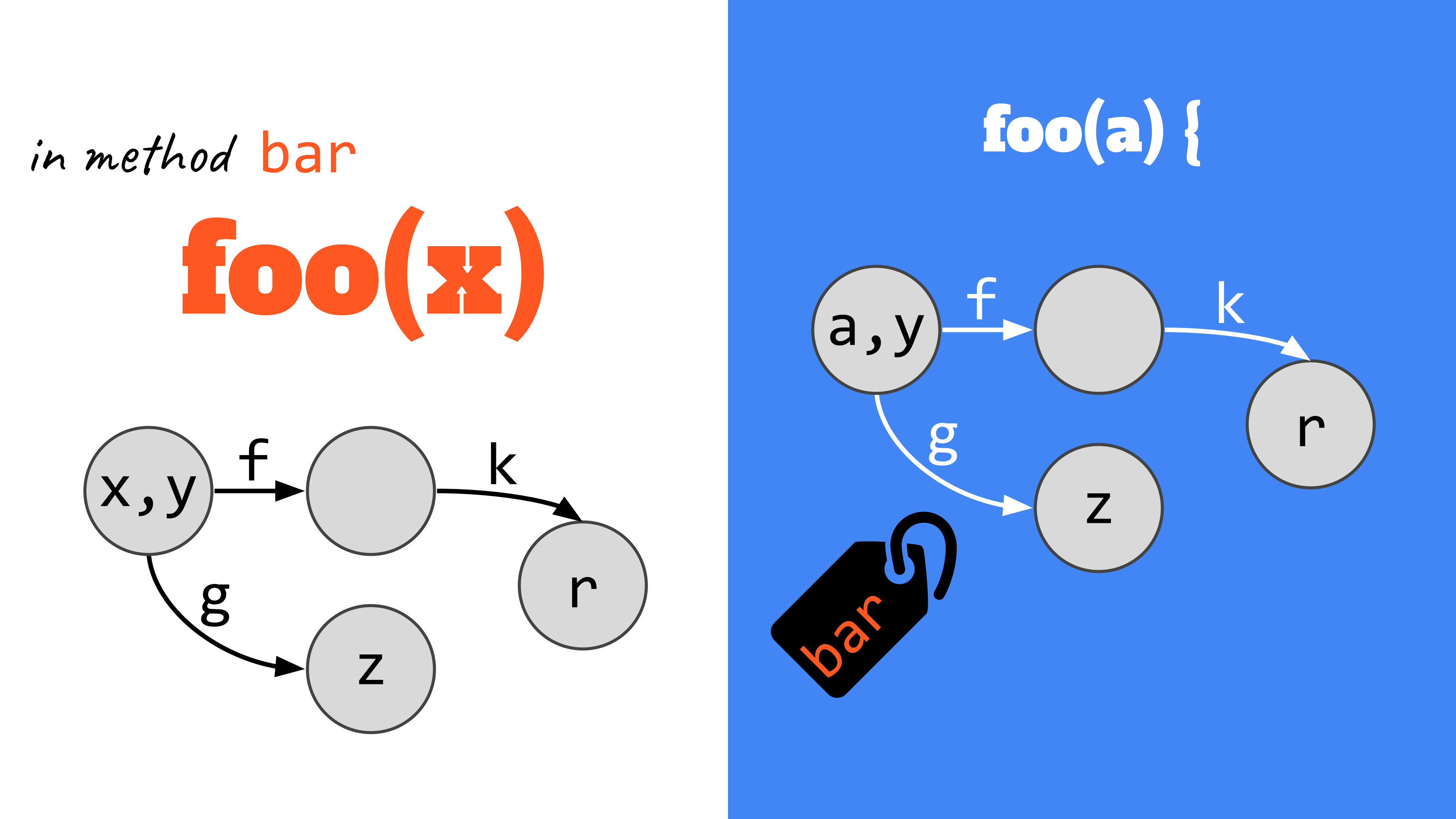




z = y.g

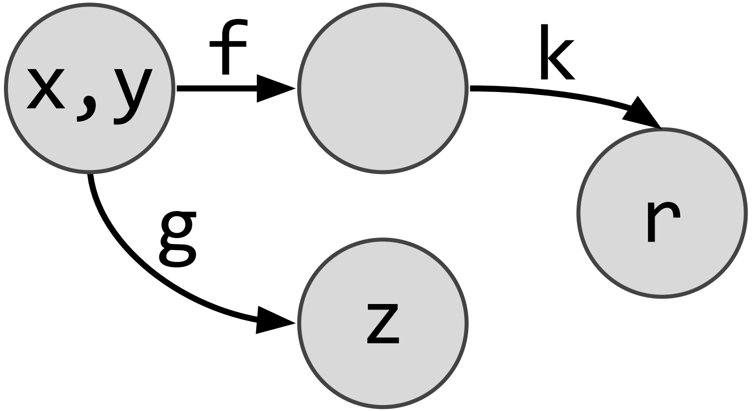




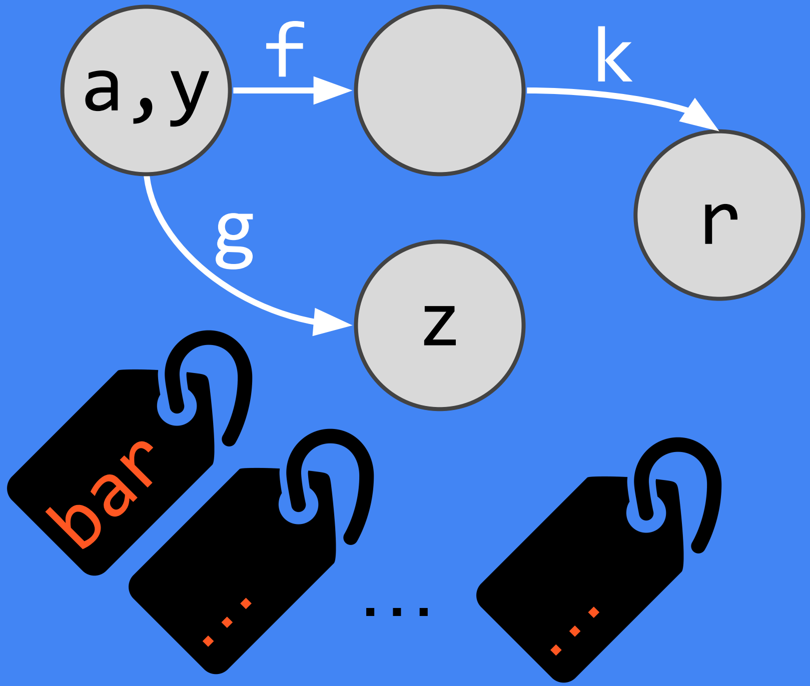

in method bar
foo(x)
x, y
f
k
g
z
r
foo(a) {
a, y
f
k
g
z
r
bar

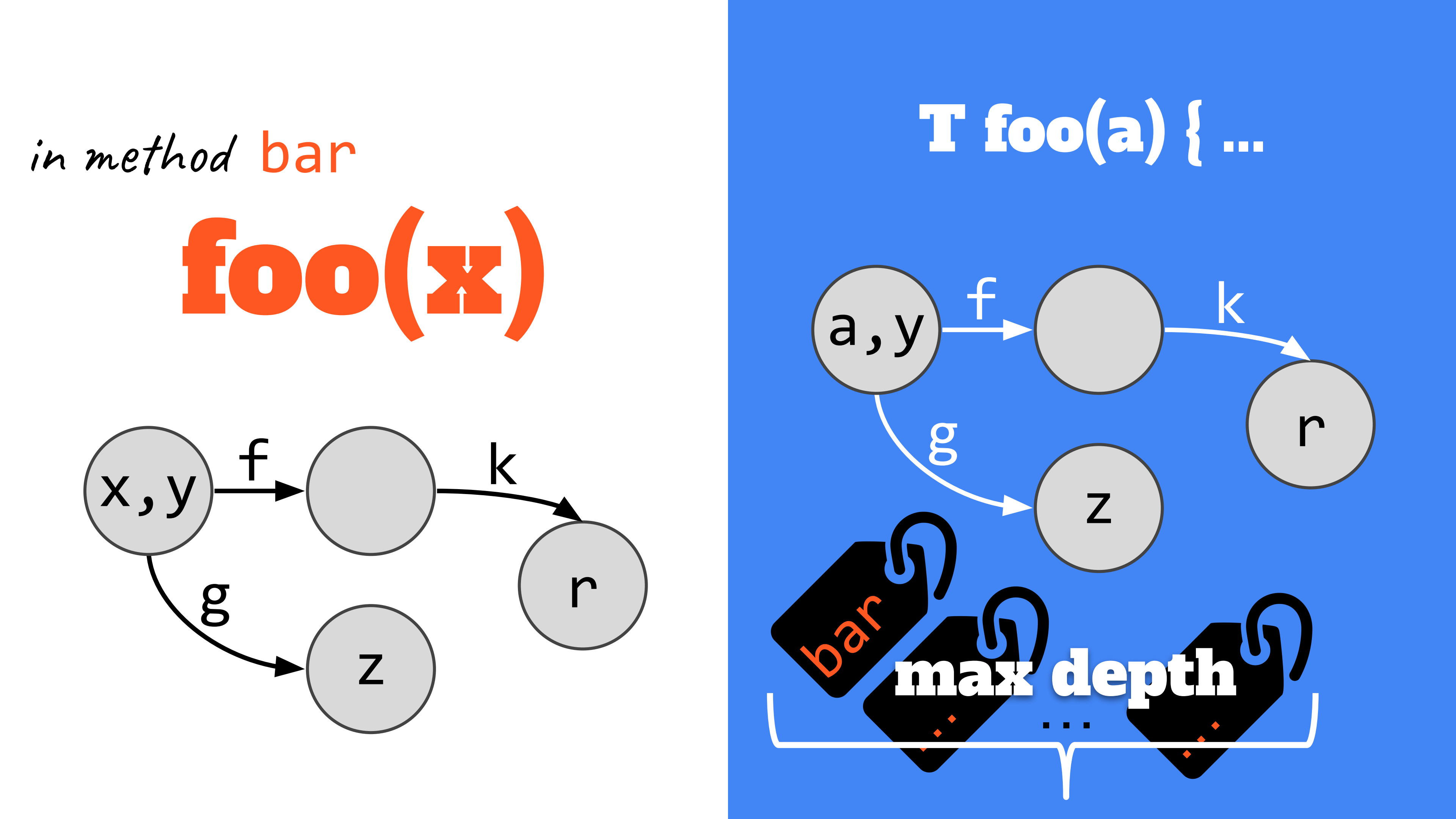
in method bar

foo(x)



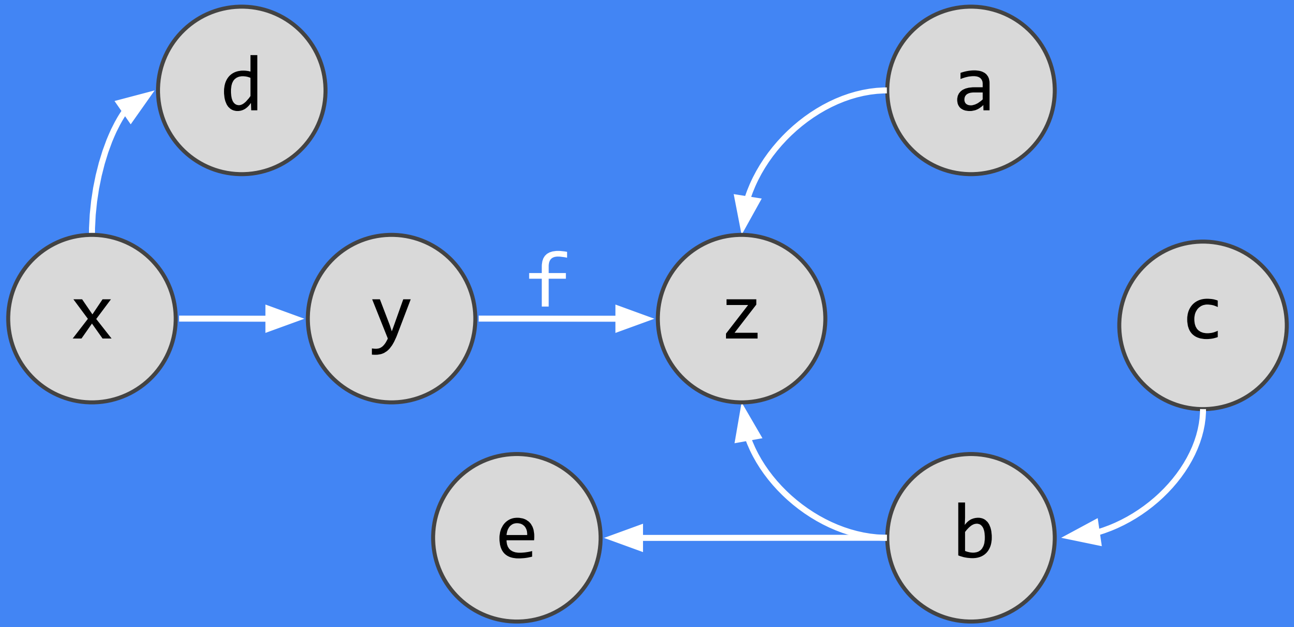
T foo(a) { ...



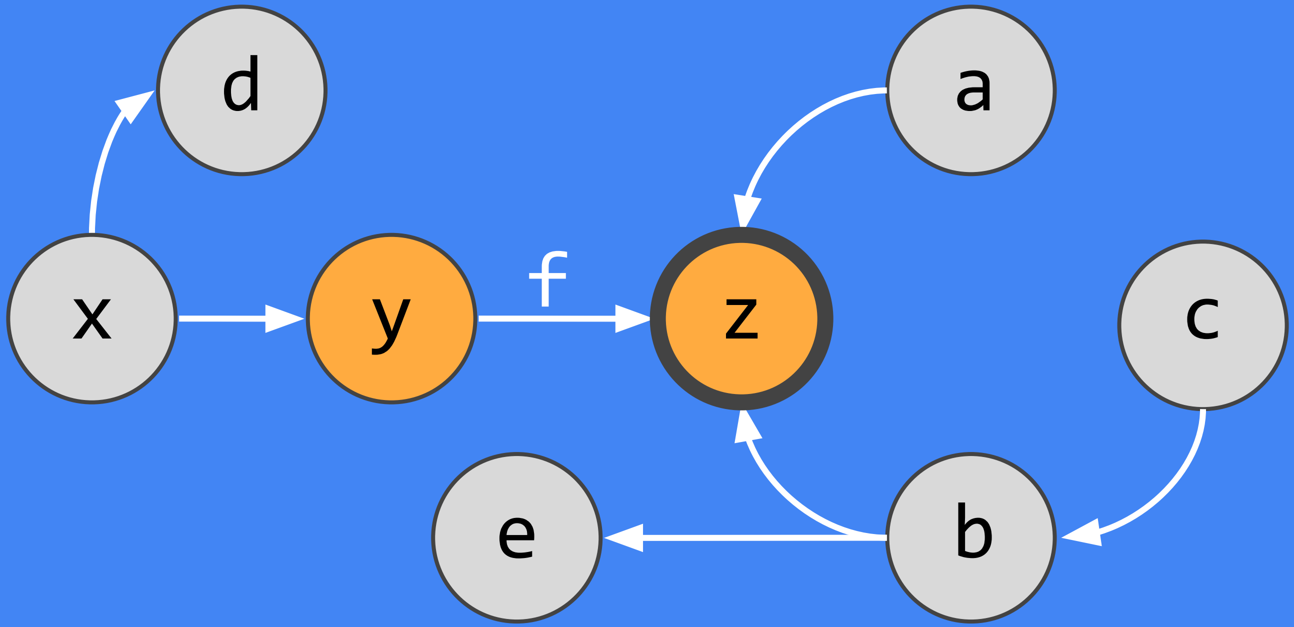

in method bar
foo(x)
x, y
f
k
g
z
r
T foo(a) { ...
a, y
f
k
g
z
r
bar
max depth
...

Algorithms

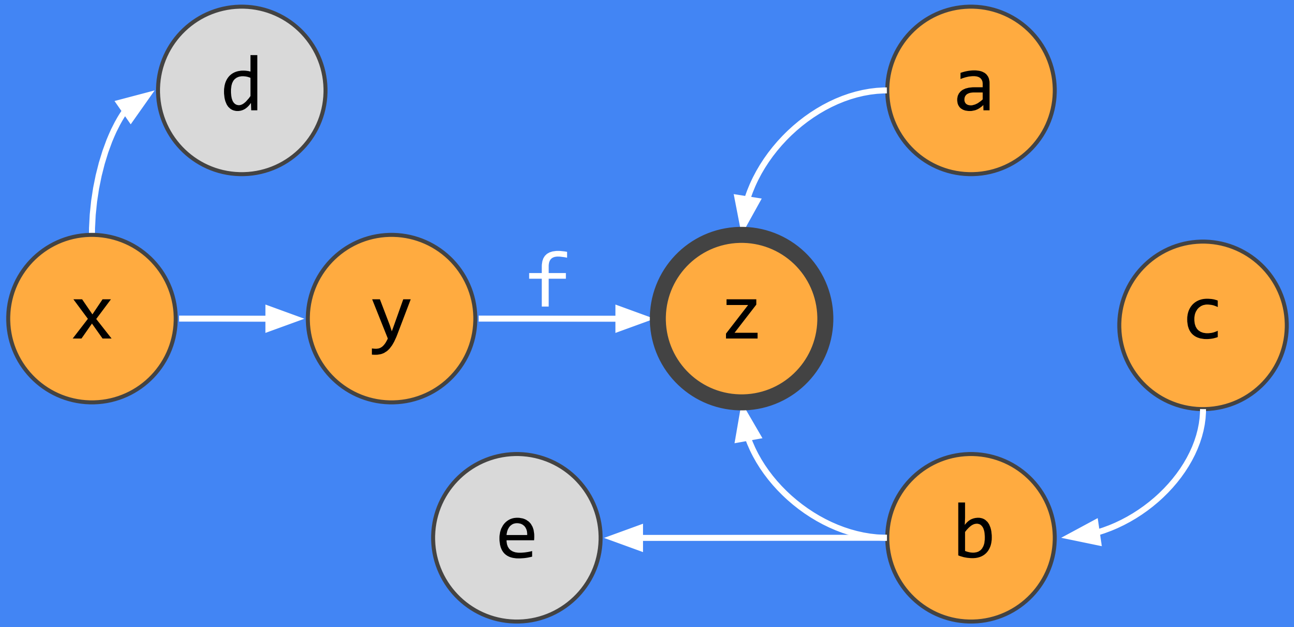
allAliases(y.f, len)



allAliases(y.f, len)

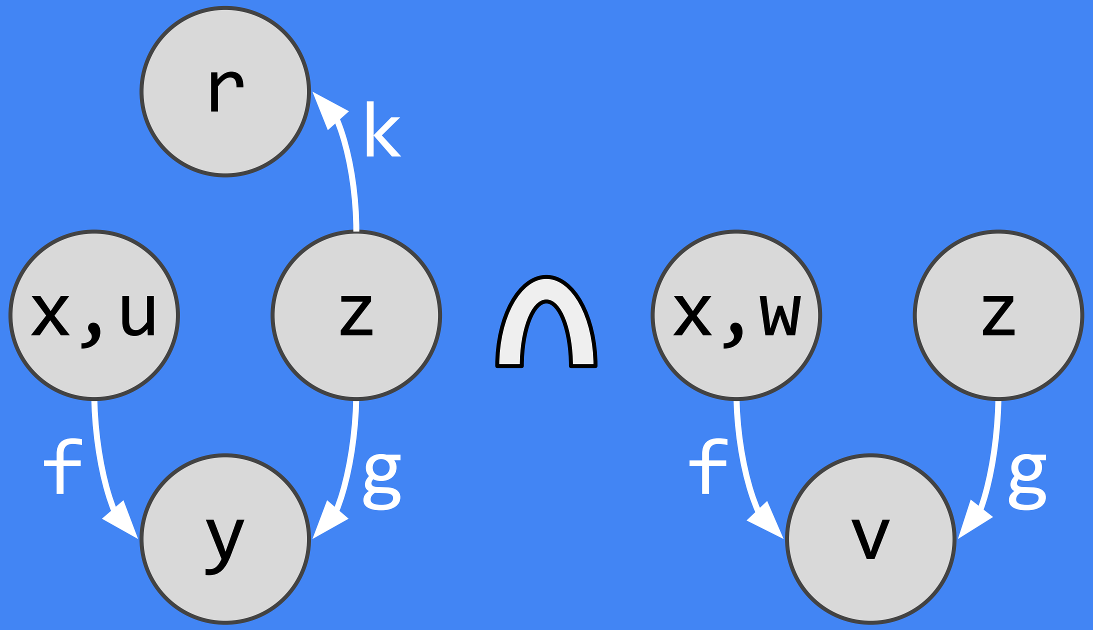


allAliases(y.f, len)

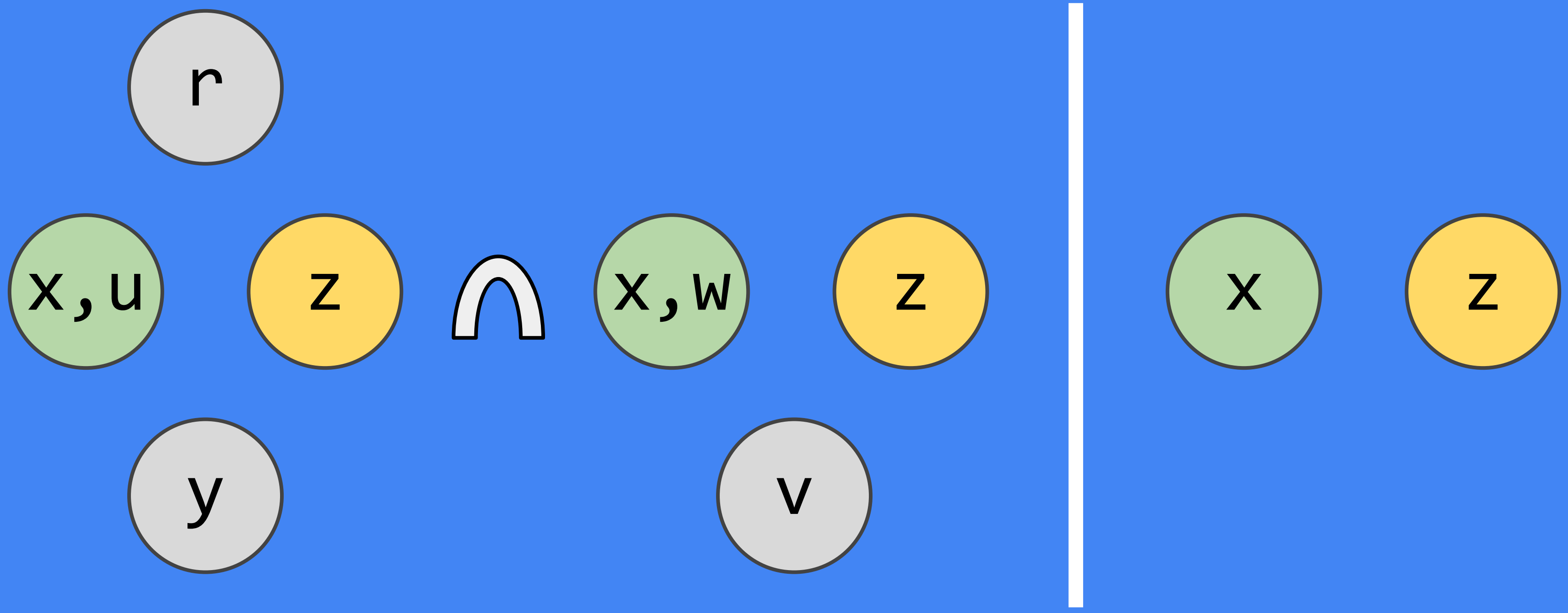


allAliases(y.f, len)

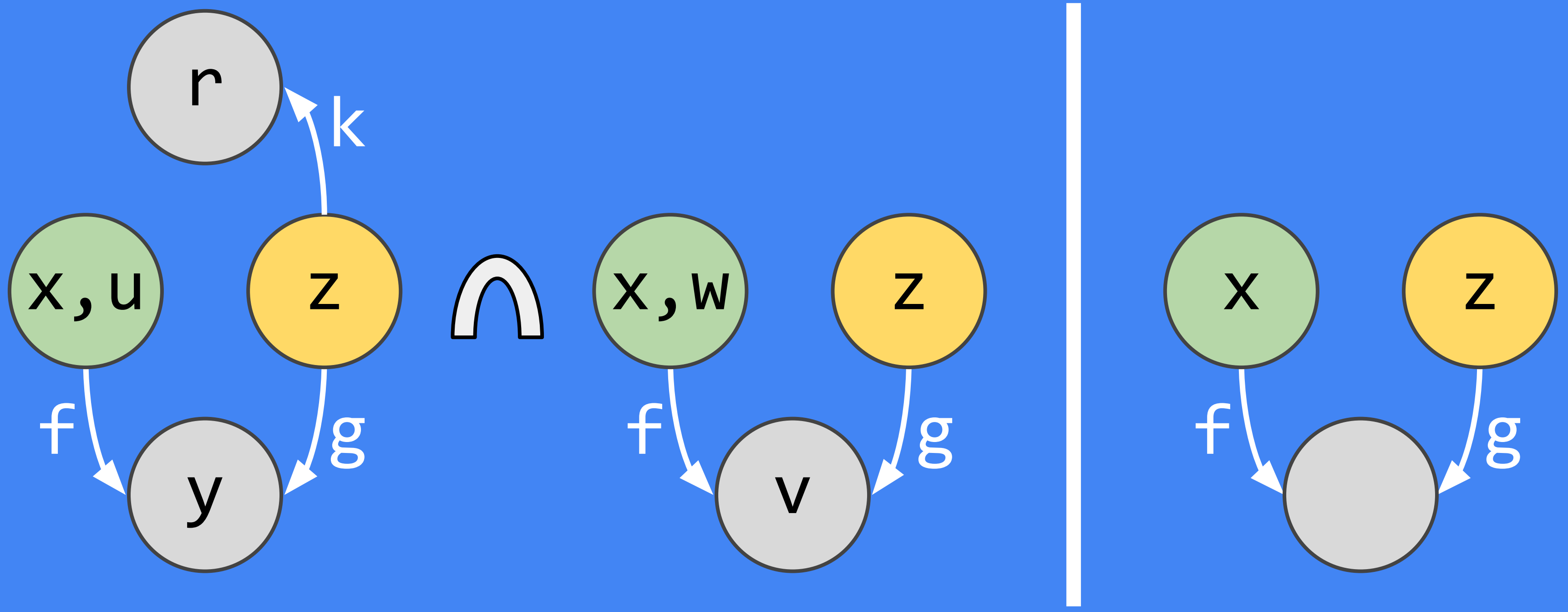
intersect(g1, g2)



intersect(g1, g2)

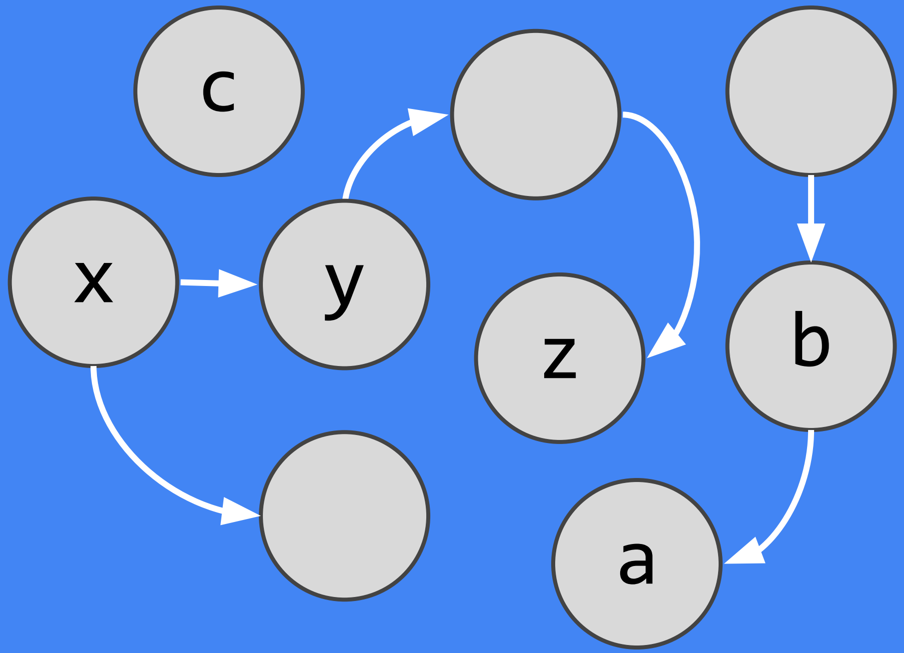


intersect(g1, g2)

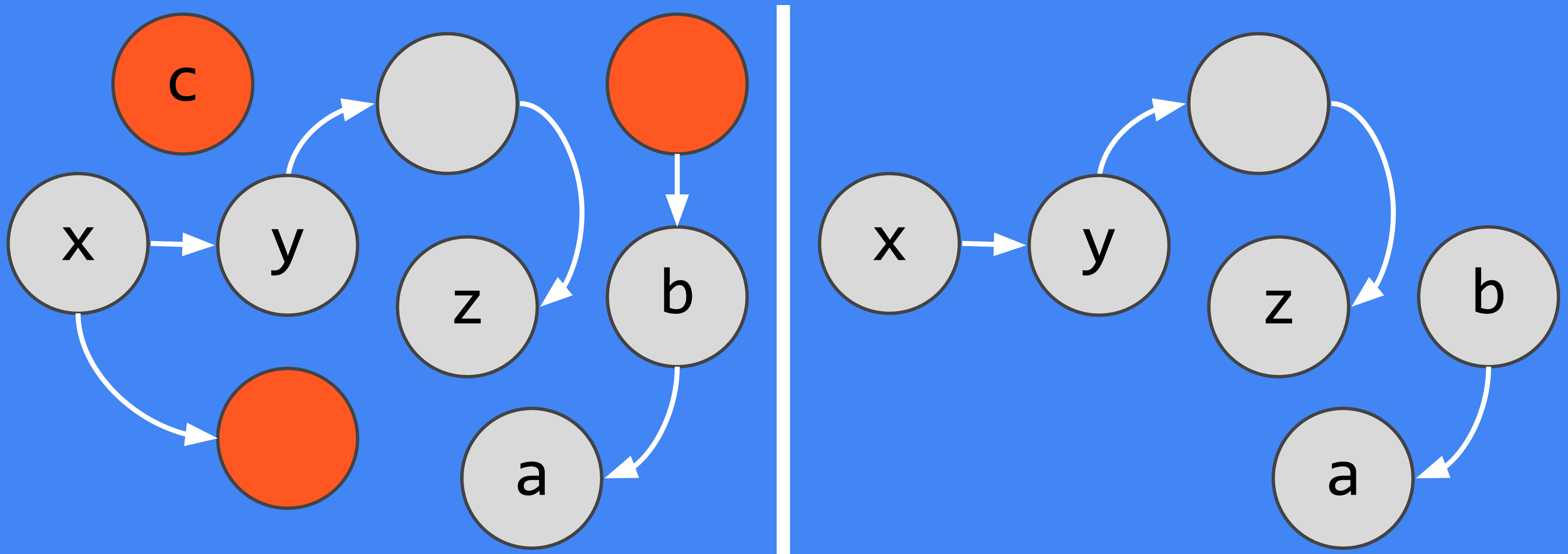


intersect(g1, g2)

gc(g)

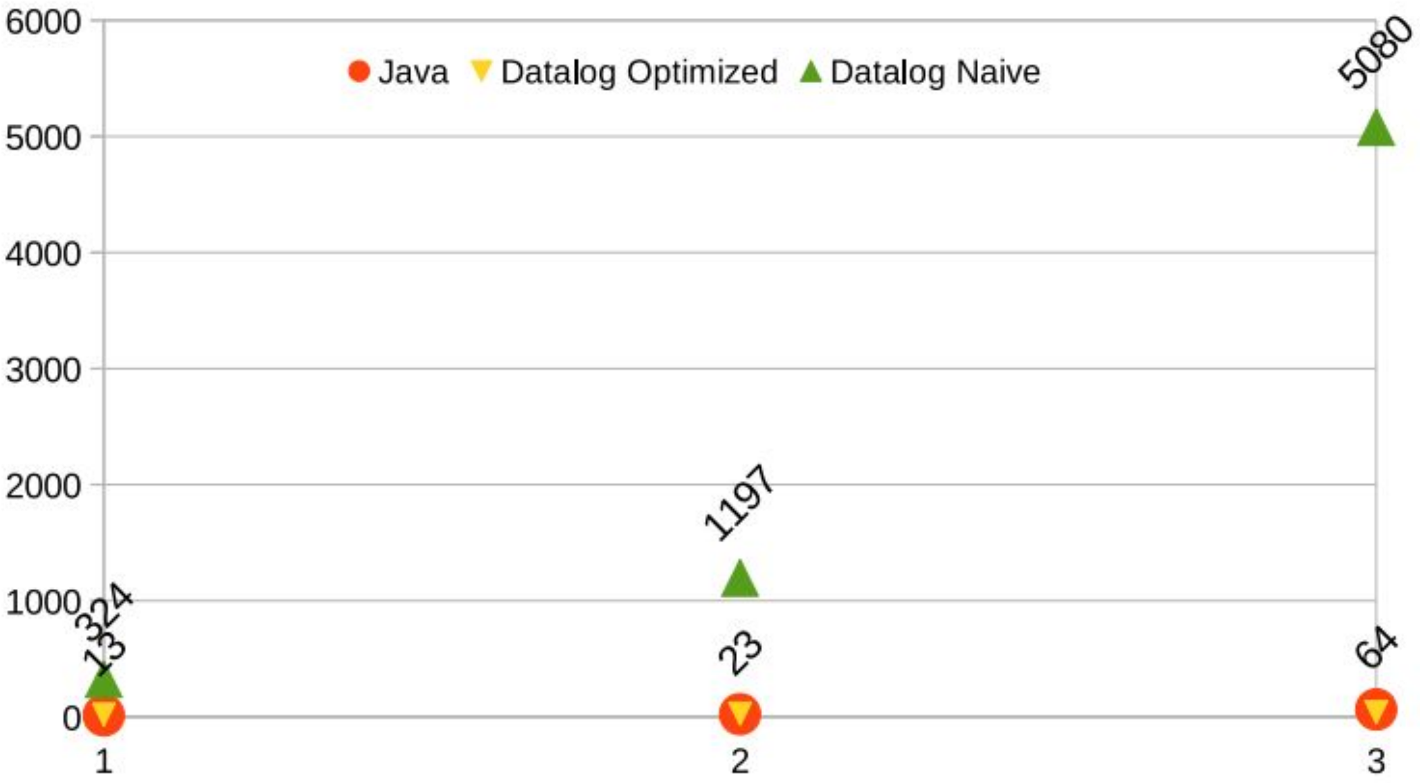


gc(g)



gc(g)

Max Access Path Length x Time



Max Context Depth x Time

an efficient data structure for Must-Alias Analysis

George Kastrinis

George Balatsouras
Kostas Ferles
Nefeli Prokopaki
Yannis Smaragdakis