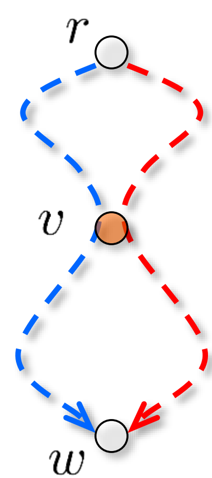
Dominators in a Flowgraph

Flowgraph: $G = (V, E, r)$; each v in V is reachable from r

v dominates w if every path from r to w includes v



Application areas : Program optimization, VLSI testing, theoretical biology, distributed systems, constraint programming, memory profiling, analysis of diffusion networks...

Dominators in a Flowgraph

Flowgraph: $G = (V, E, r)$; each v in V is reachable from r

v dominates w if every path from r to w includes v

Set of dominators: $Dom(w) = \{ v \mid v \text{ dominates } w \}$

Trivial dominators: $\forall\, w \neq r,\ \ w, r \in Dom(w)$

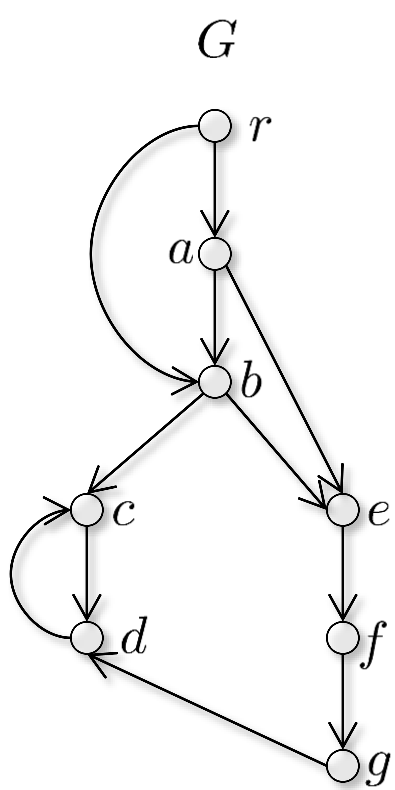
Immediate dominator: $idom(w) \in Dom(w) - w$ and dominated by every v in $Dom(w) - w$

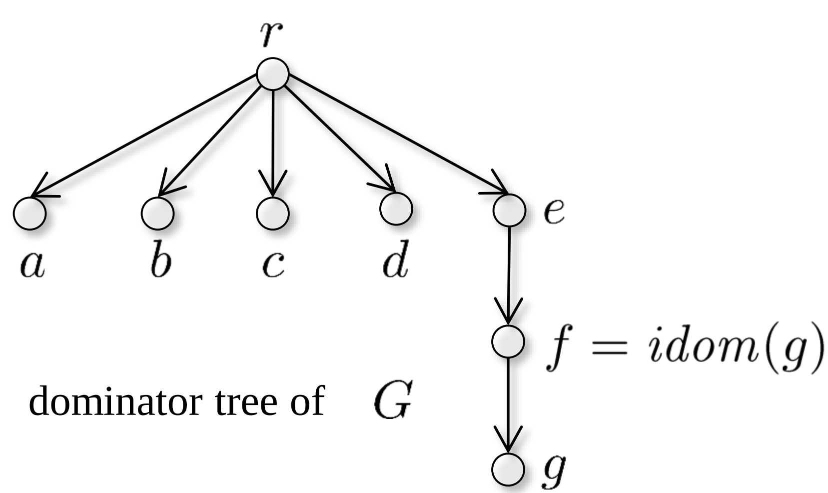
Goal: Find $idom(v)$ for each v in V

Dominators in a Flowgraph

Flowgraph: $G = (V, E, r)$; each v in V is reachable from r

v **dominates** w if every path from r to w includes v





$O(m\alpha(m,n))$ algorithm: [Lengauer and Tarjan '79]

$O(m+n)$ algorithms:

[Alstrup, Harel, Lauridsen, and Thorup '97]

[Buchsbaum, Kaplan, Rogers, and Westbrook '04]

[G., and Tarjan '04]

[Buchsbaum et al. '08]

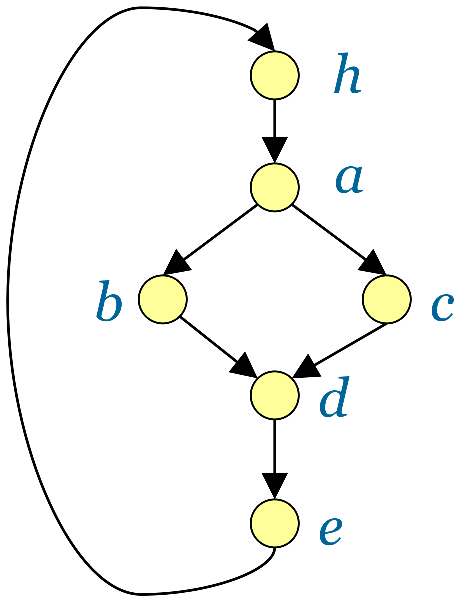
Application: Loop Optimizations

Loop optimizations typically make a program much more efficient since a large fraction of the total running time is spent on loops.

Dominators can be used to detect loops.

Application: Loop Optimizations

Loop L



There is a node $h \in L$ (loop header), such that

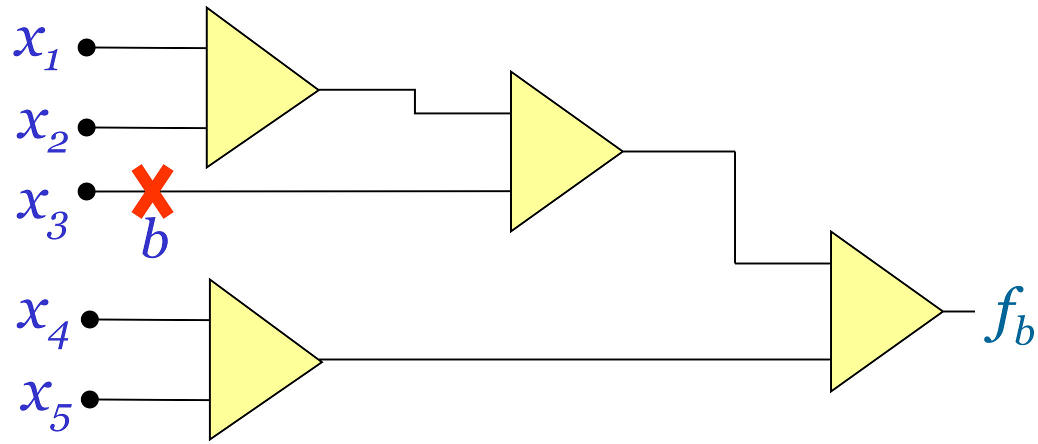
- There is a (v,h) for some $v \notin L$
- For any $w \in L\text{-}h$ there is no (v,w) for $v \notin L$
- h is reachable from every $w \in L$
- h reaches every $w \in L$

Thus h dominates all nodes in L.

Loop back-edge: $(v,h) \in A$ and h dominates v.

Application: Identifying Functionally Equivalent Faults

Consider a circuit C:



Inputs: $x_1, \ldots, x_n$

Output: $f(x_1, \ldots, x_n)$

Suppose there is a fault in wire a. Then $C = C_a$ evaluates f_a instead.

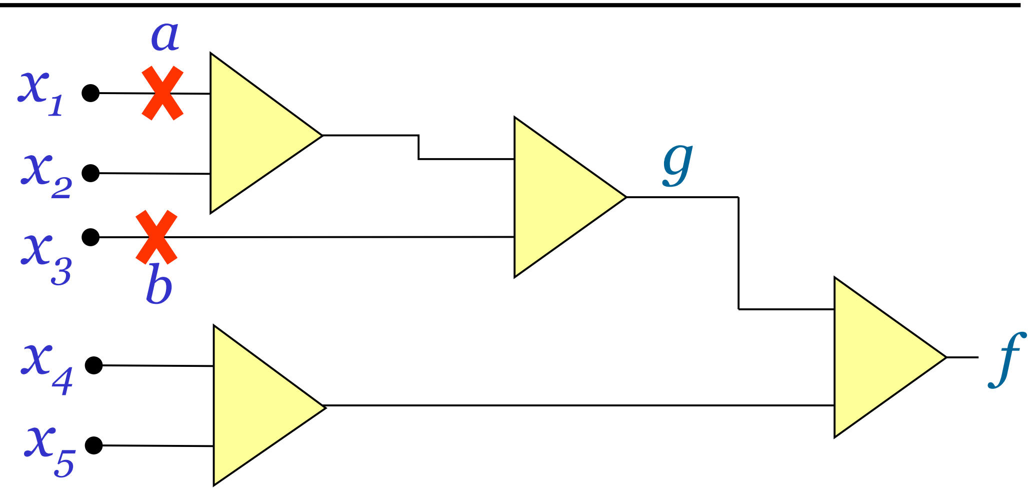
Fault a and fault b are functionally equivalent iff

$$f_a(x_1, \ldots, x_n) = f_b(x_1, \ldots, x_n)$$

Such pairs of faults are indistinguishable and we want to avoid spending time to distinguish them (since it is impossible).

Application: Identifying Functionally Equivalent Faults

Consider a circuit C:



Inputs: $x_1, \ldots, x_n$

Output: $f(x_1, \ldots, x_n)$

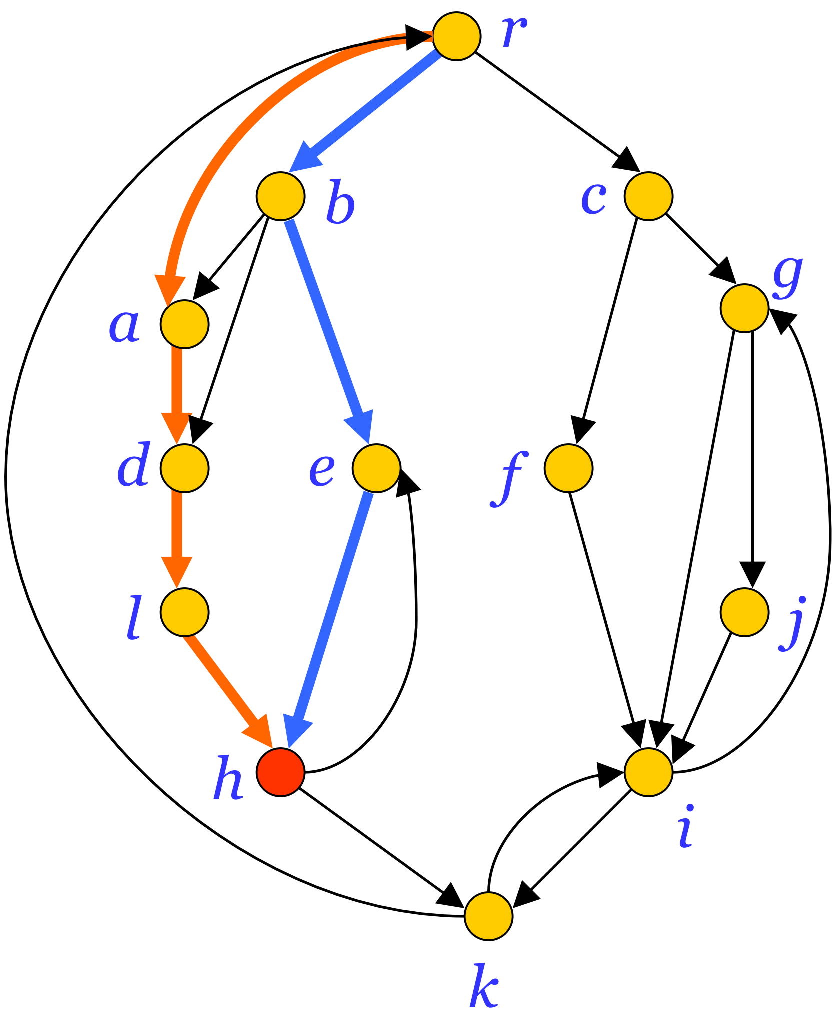
Suppose there is a fault in wire a. Then $C = C_a$ evaluates f_a instead.

Fault a and fault b are functionally equivalent iff

$$f_a(x_1, \ldots, x_n) = f_b(x_1, \ldots, x_n)$$

It suffices to evaluate the output of a gate g that dominates a and b. This is can be faster than evaluating f since g may have fewer inputs.

Example



$Dom(r) = \{ r \}$

$Dom(b) = \{ r, b \}$

$Dom(c) = \{ r, c \}$

$Dom(a) = \{ r, a \}$

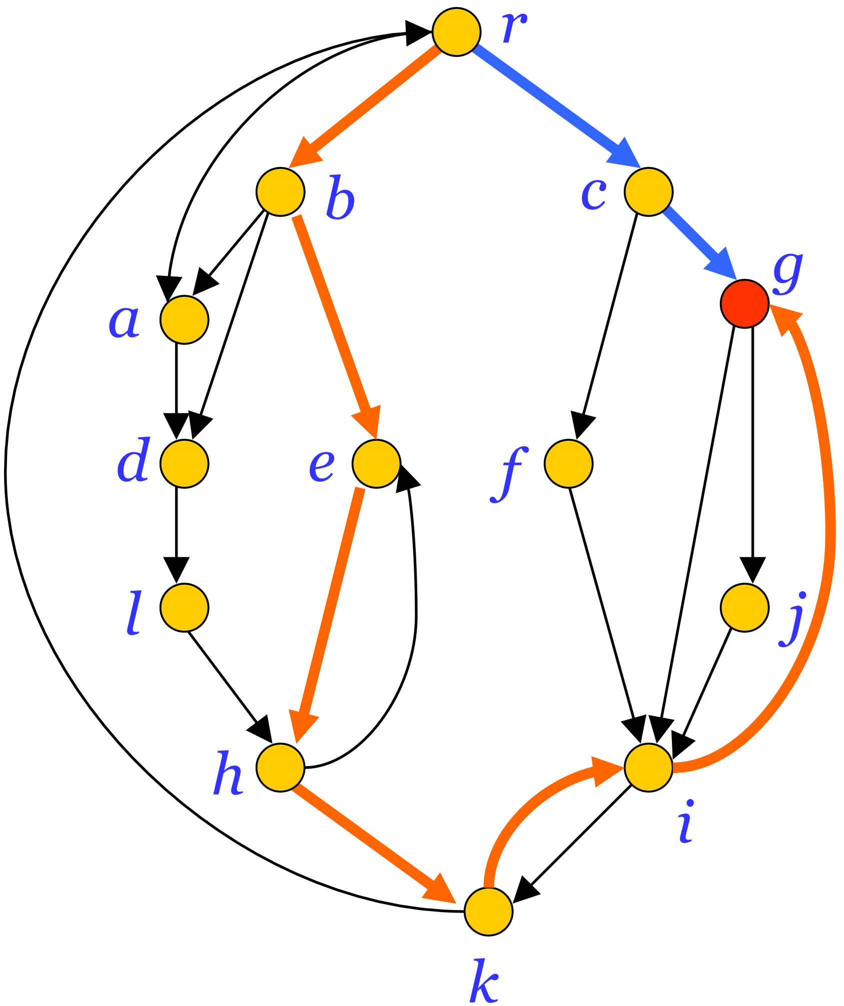
$Dom(d) = \{ r, d \}$

$Dom(e) = \{ r, e \}$

$Dom(l) = \{ r, d, l \}$

$Dom(h) = \{ r, h \}$

Example



$Dom(r) = \{ r \}$

$Dom(b) = \{ r, b \}$

$Dom(c) = \{ r, c \}$

$Dom(a) = \{ r, a \}$

$Dom(d) = \{ r, d \}$

$Dom(e) = \{ r, e \}$

$Dom(l) = \{ r, d, l \}$

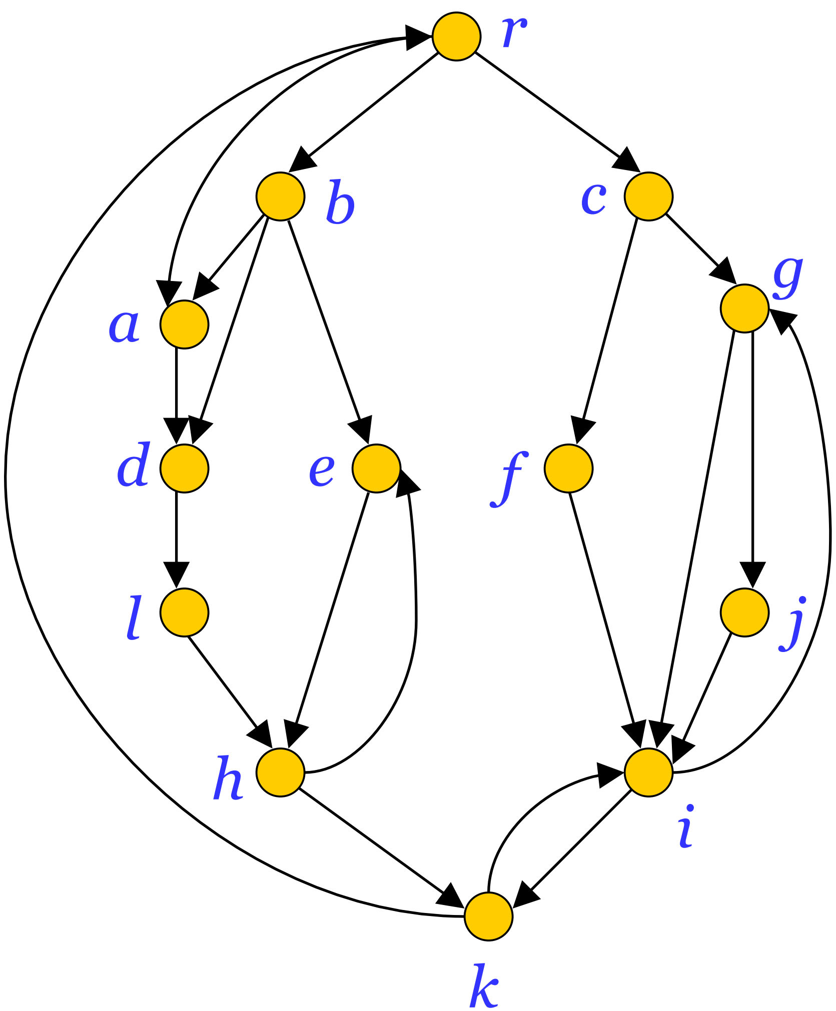
$Dom(h) = \{ r, h \}$

$Dom(k) = \{ r, k \}$

$Dom(f) = \{ r, c, f \}$

$Dom(g) = \{ r, g \}$

Example



$Dom(b) = \{ r, b \}$

$Dom(c) = \{ r, c \}$

$Dom(a) = \{ r, a \}$

$Dom(d) = \{ r, d \}$

$Dom(e) = \{ r, e \}$

$Dom(l) = \{ r, d, l \}$

$Dom(h) = \{ r, h \}$

$Dom(k) = \{ r, k \}$

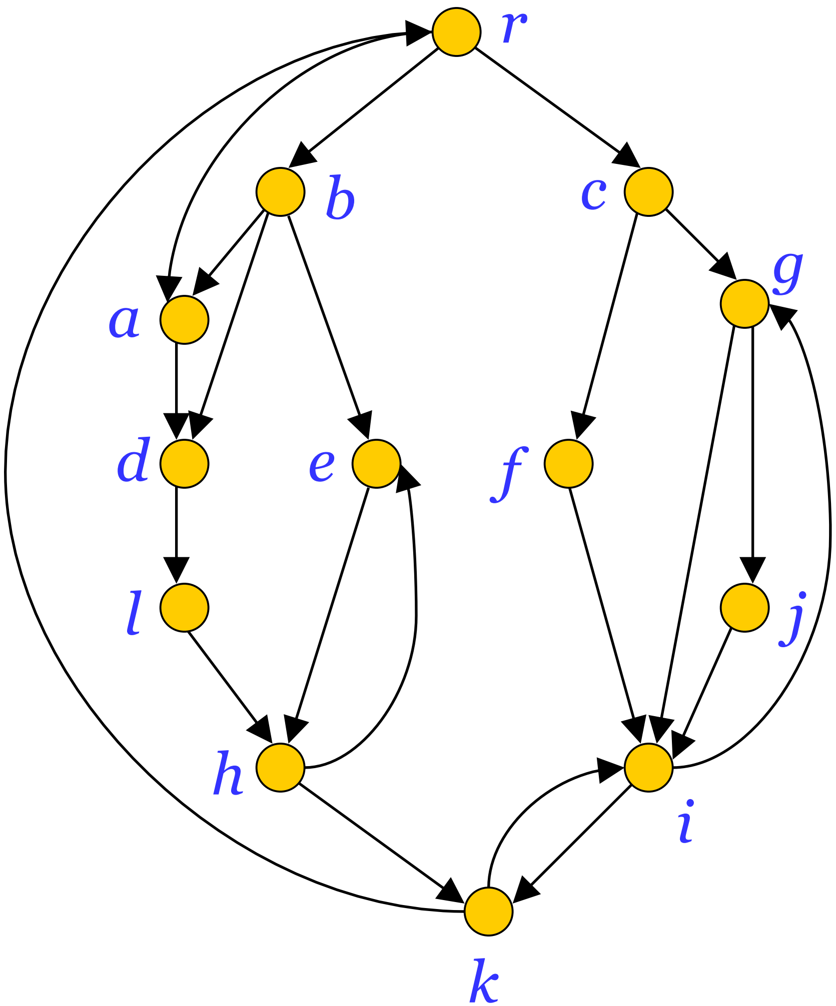
$Dom(f) = \{ r, c, f \}$

$Dom(g) = \{ r, g \}$

$Dom(j) = \{ r, g, j \}$

$Dom(i) = \{ r, i \}$

Example



$idom(b) = r$

$idom(c) = r$

$idom(a) = r$

$idom(d) = r$

$idom(e) = r$

$idom(l) = d$

$idom(h) = r$

$idom(k) = r$

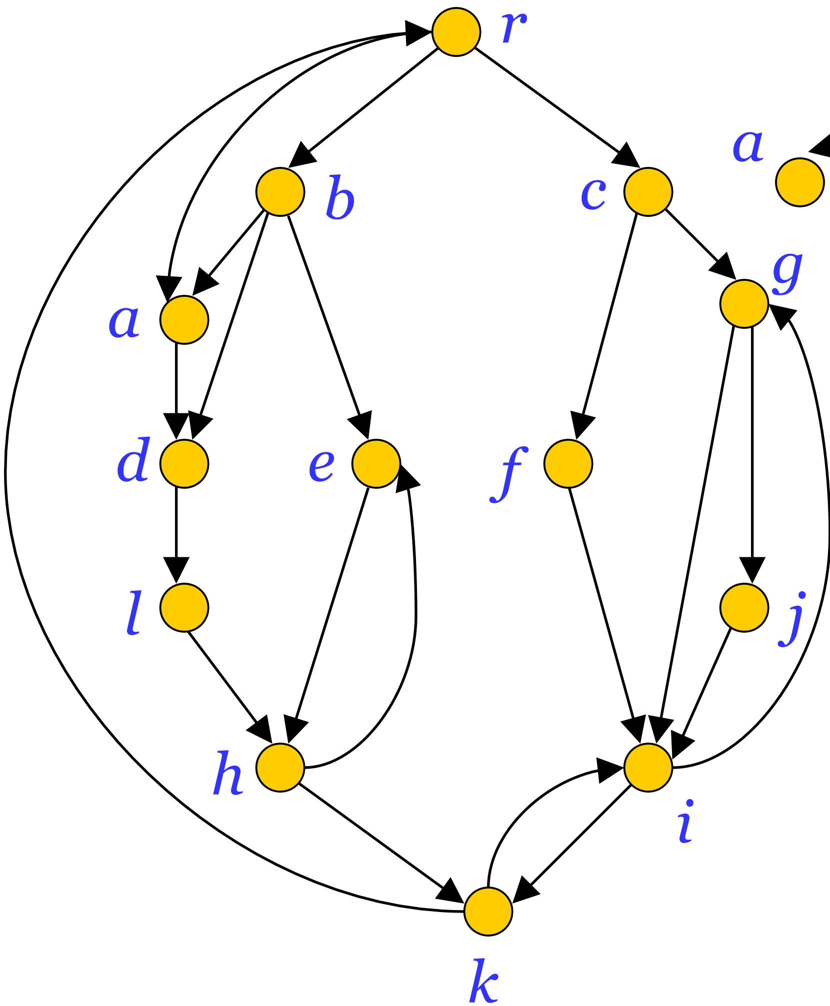
$idom(f) = c$

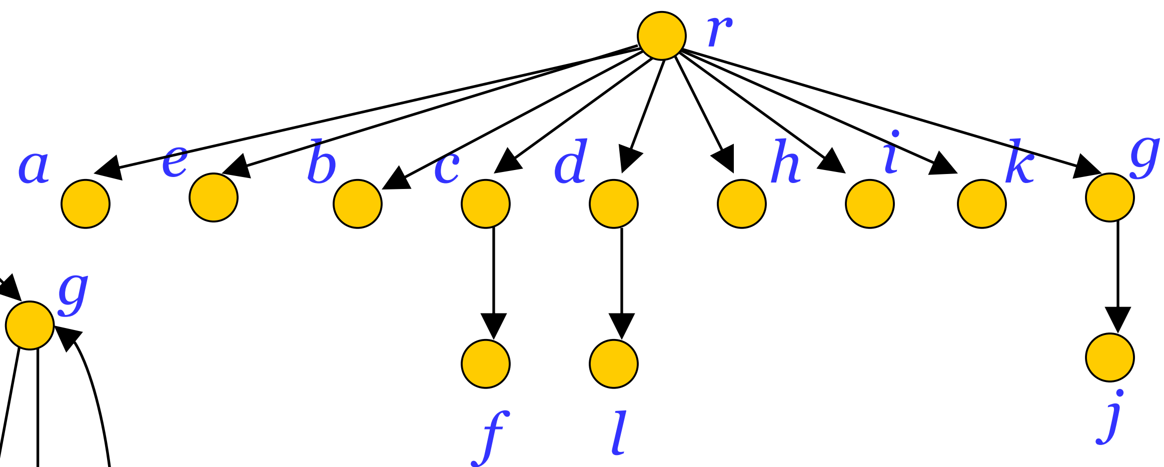
$idom(g) = r$

$idom(j) = g$

$idom(i) = r$

Example





Dominator Tree D

$$D = (\, V,\ \cup_{w \neq r}\, (idom(w), w)\,)$$

A Straightforward Algorithm

Purdom-Moore [1972]:

> **for all** v in $V - r$ **do**
>   remove v from G
> $R(v) \leftarrow$ unreachable vertices
>   **for all** u in $R(v)$ **do**
> $Dom(u) \leftarrow Dom(u) \cup \{ v \}$
>   **done**
> **done**

The running time is $O(n \cdot m)$. Also very slow in practice.

Iterative Algorithm

Dominators can be computed by solving iteratively the set of equations [Allen and Cocke, 1972]

$$Dom(v) = \left(\cap_{(u,v) \in A} Dom(u) \right) \cup \{v\}, v \neq r$$

Initialization

$$Dom(r) = \{r\}$$

$$Dom(v) = \emptyset, v \neq r$$

In the intersection we consider only the nonempty *Dom(u)*.

Each *Dom(v)* set can be represented by an *n*-bit vector.
Intersection ≡ bit-wise AND.

Requires n^2 space. Very slow in practice (but better than PM).

Iterative Algorithm

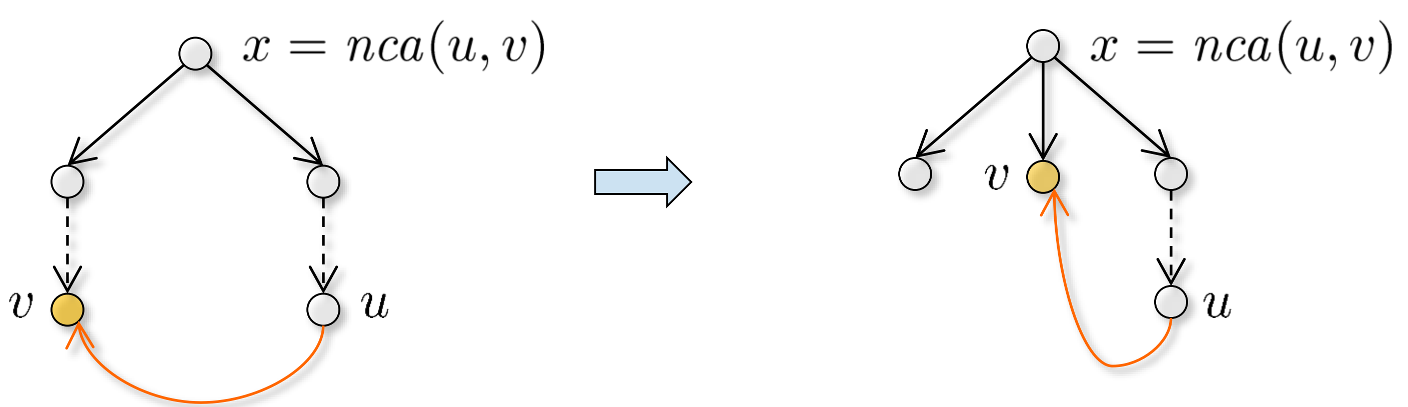
Dominators can be computed by solving iteratively the set of equations [Allen and Cocke, 1972]

$$Dom(v) = \left(\bigcap_{(u,v)\in A} Dom(u) \right) \cup \{v\}, v \neq r$$

Efficient implementation [Cooper, Harvey and Kennedy 2000]:

Maintain tree T; process the edges until a fixed-point is reached.

Process (u,v): compute $x =$ nearest common ancestor of u and v in T.
If x is ancestor of parent of v, make x new parent of v.



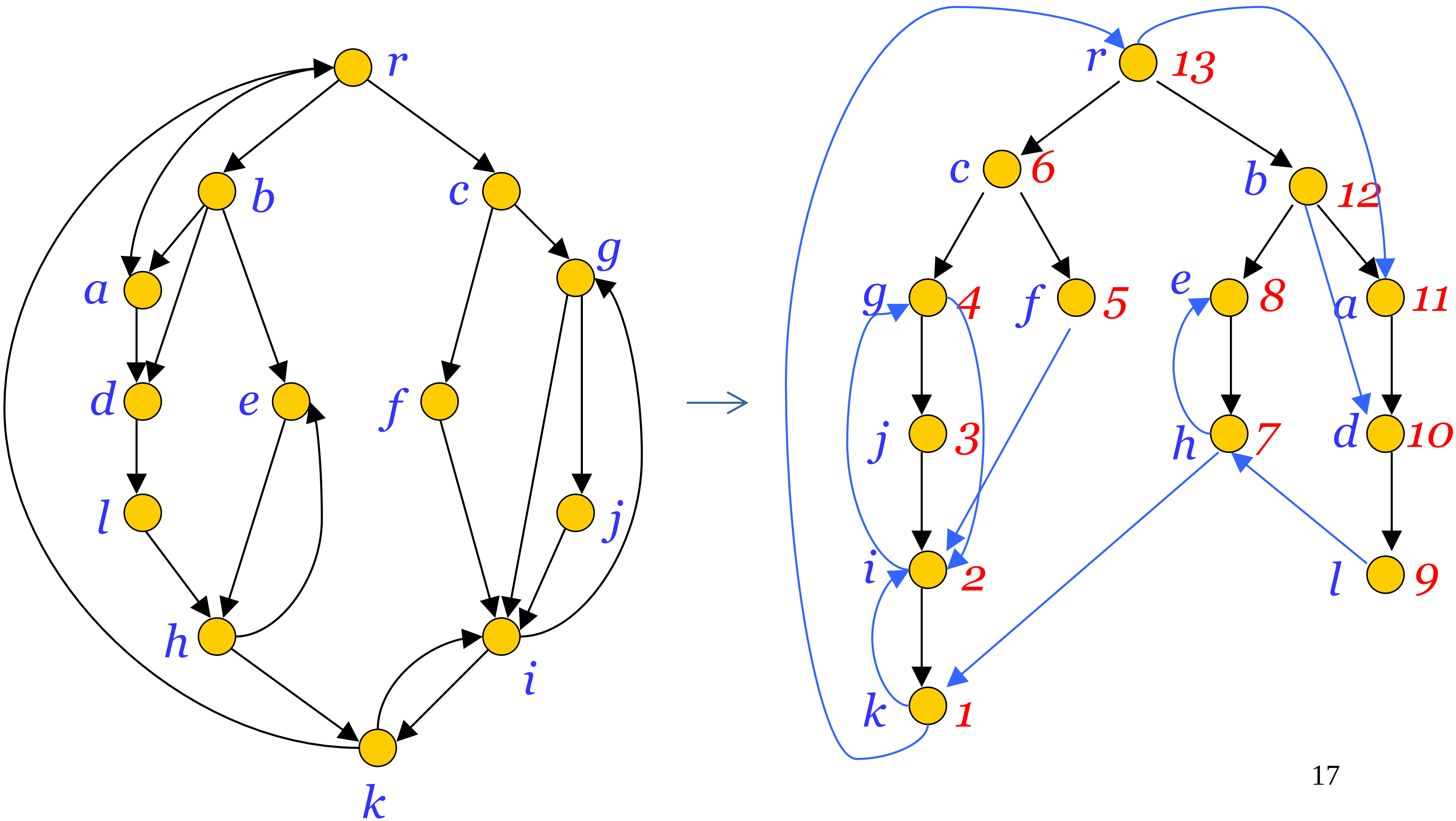
Iterative Algorithm

Efficient implementation [Cooper, Harvey and Kennedy 2000]

```
dfs(r)
T ← {r}
changed ← true
while ( changed ) do
    changed ← false
    for all v in V − r in reverse postorder do
        x ← nca(pred(v))
        if x ≠ parent(v) then
            parent(v) ← x
            changed ← true
        end
    done
done
```

Iterative Algorithm: Example

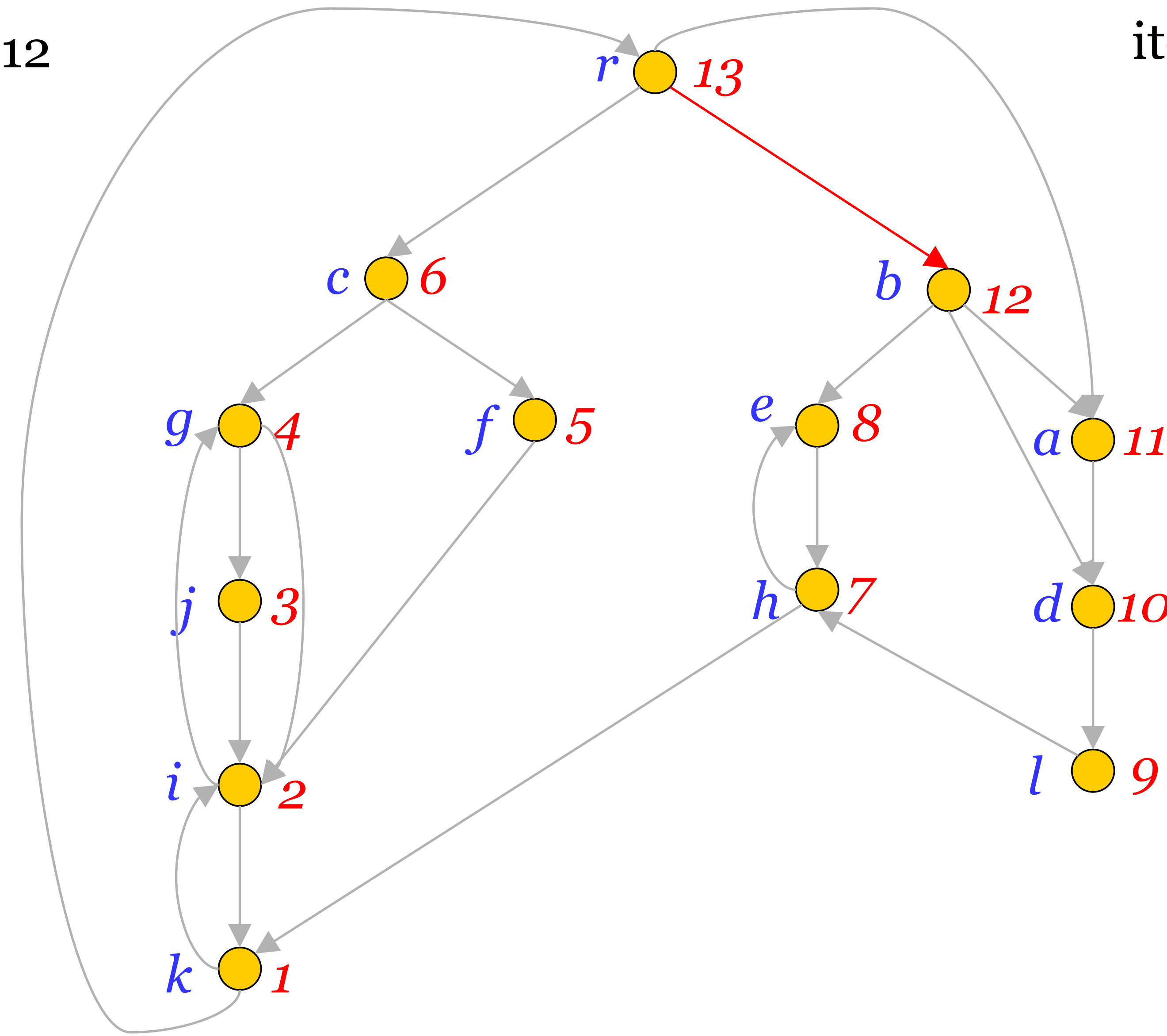
Perform a depth-first search on G $\Rightarrow$ postorder numbers



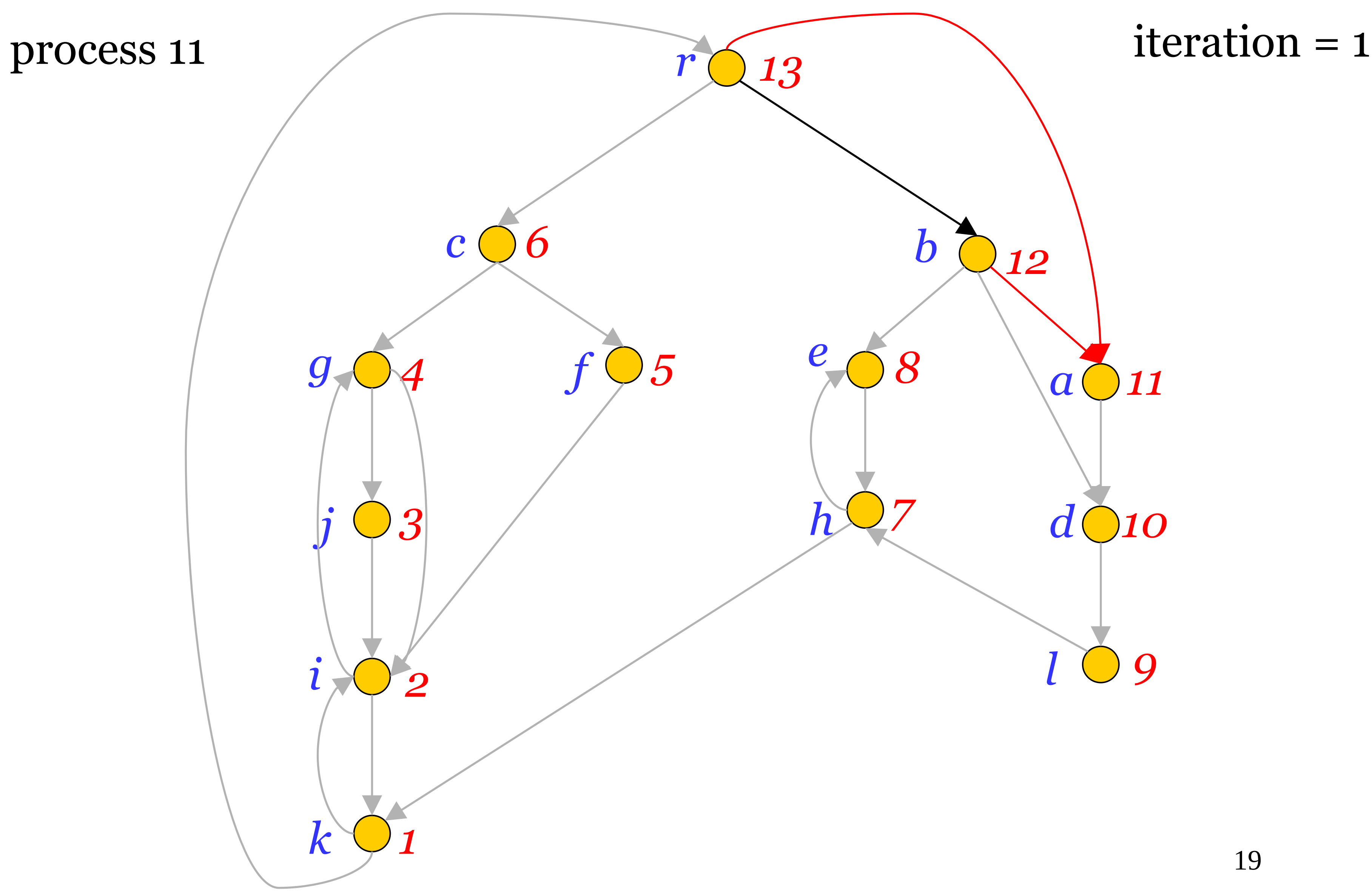
Iterative Algorithm: Example

process 12

iteration = 1



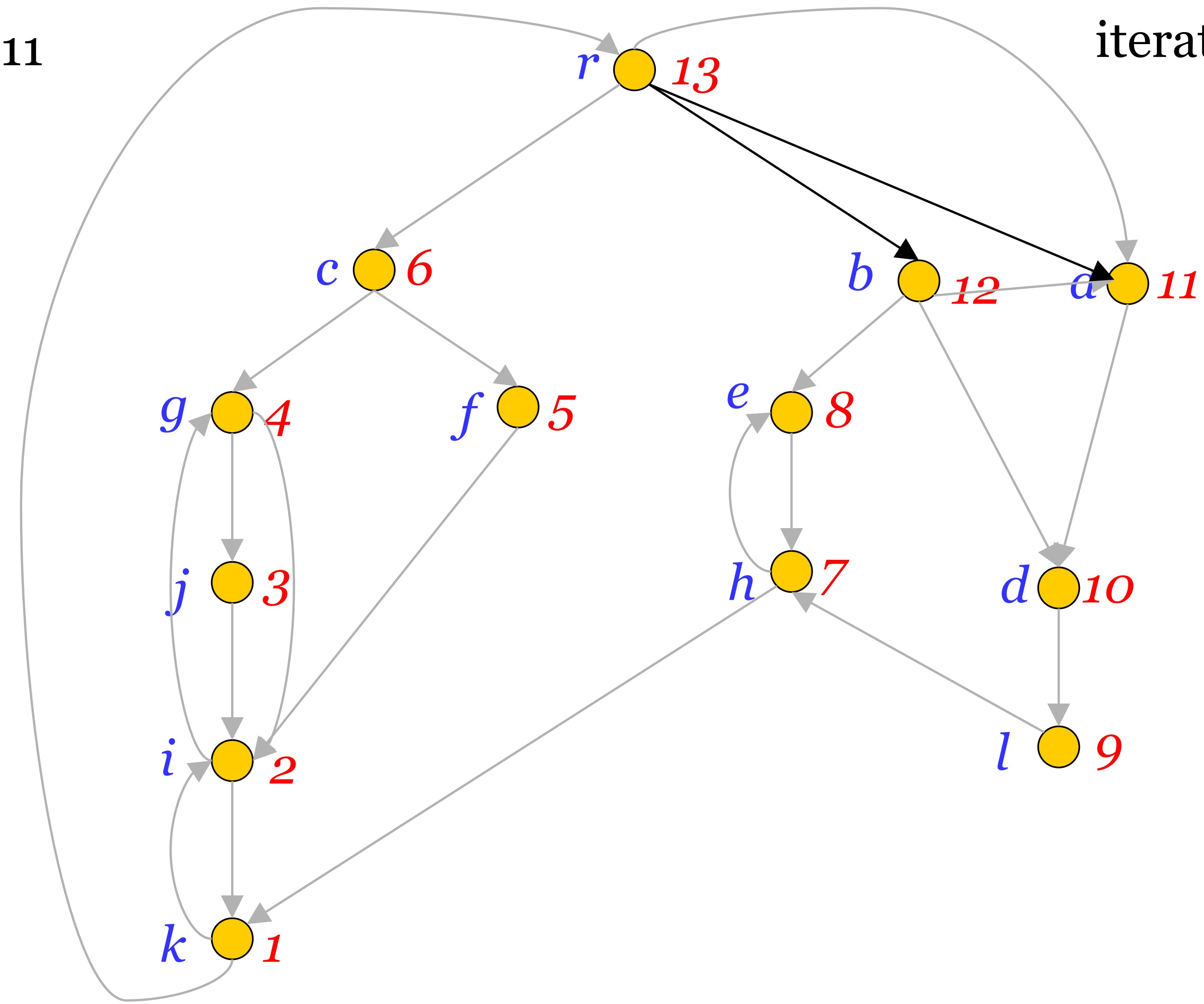
Iterative Algorithm: Example



Iterative Algorithm: Example

process 11

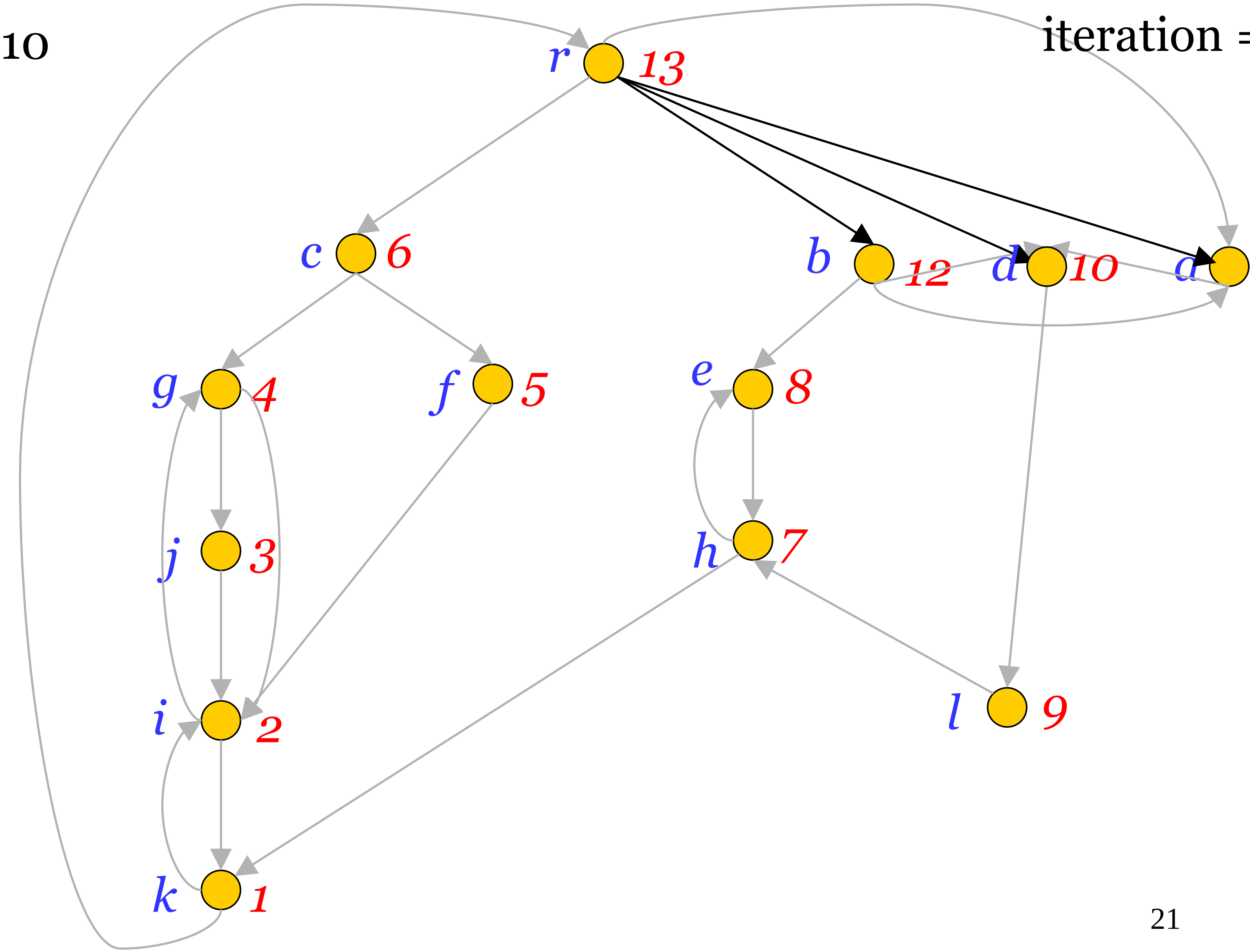
iteration = 1



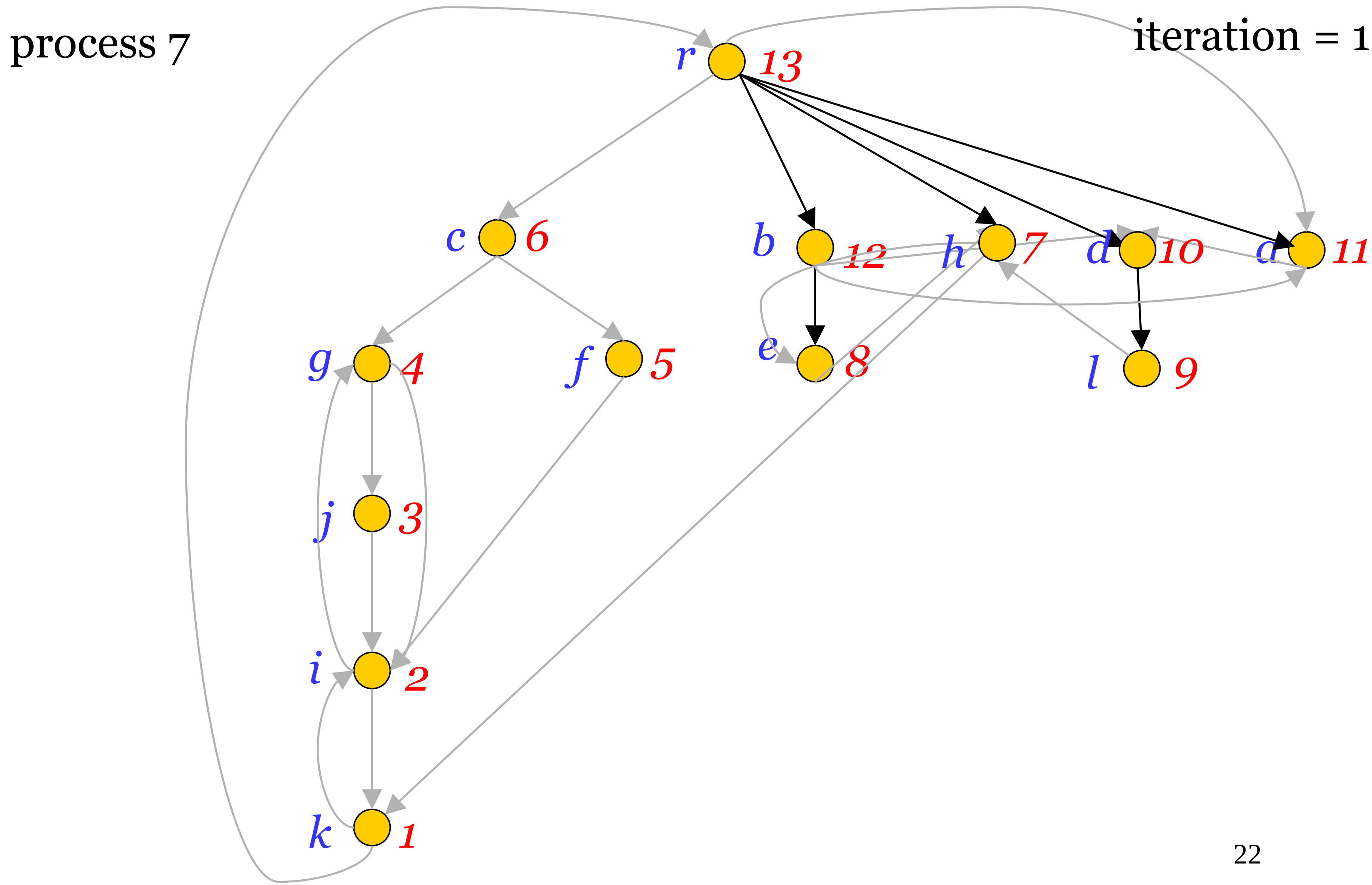
Iterative Algorithm: Example

process 10

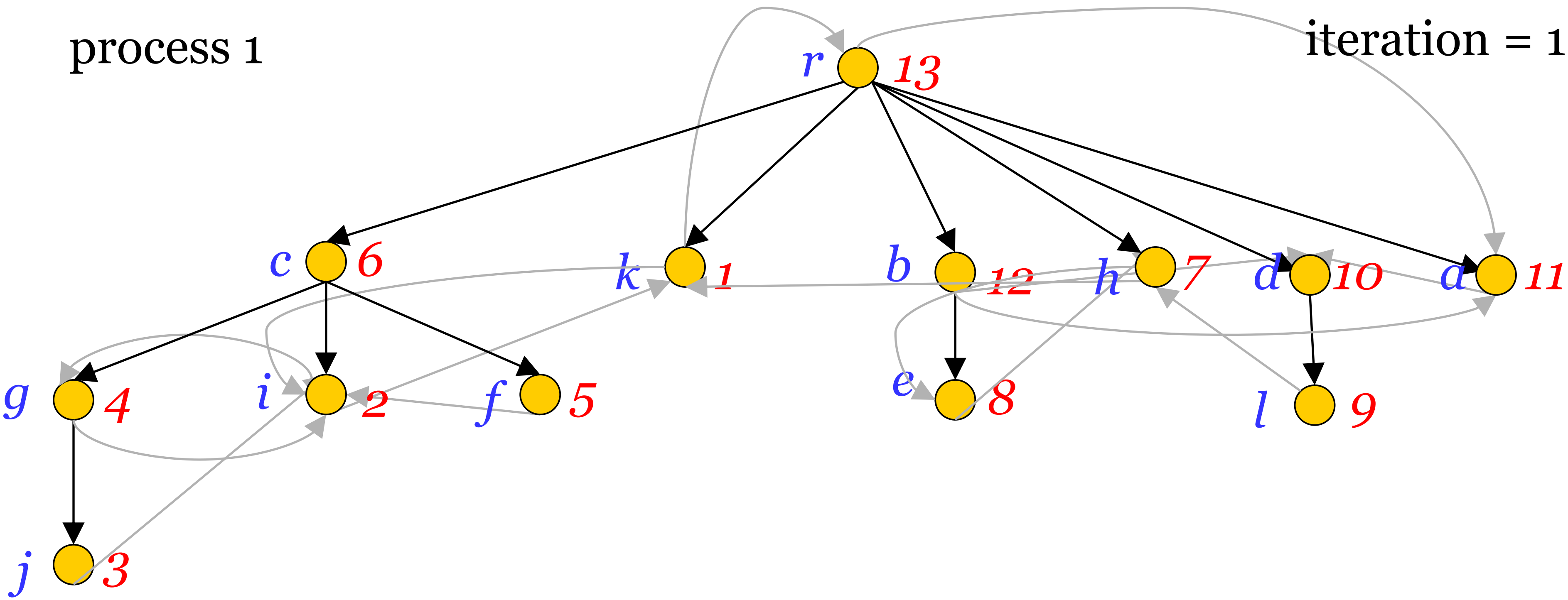
iteration = 1



Iterative Algorithm: Example



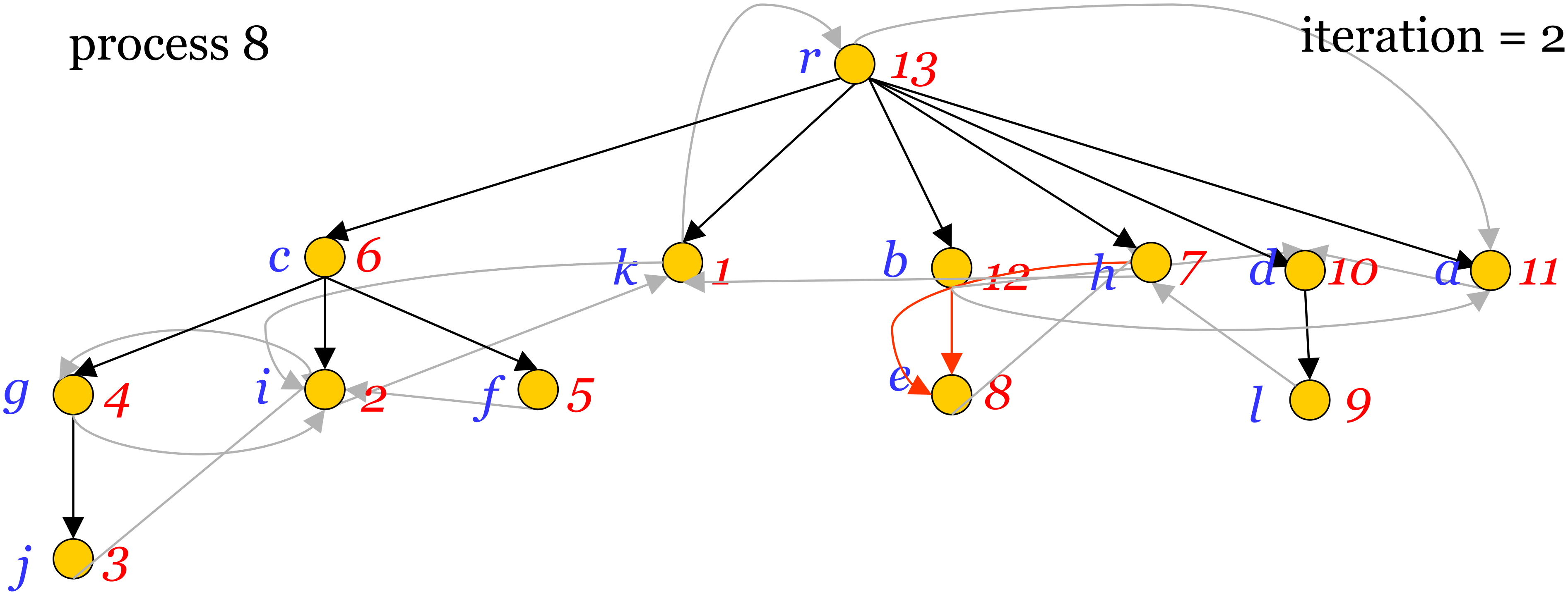
Iterative Algorithm: Example



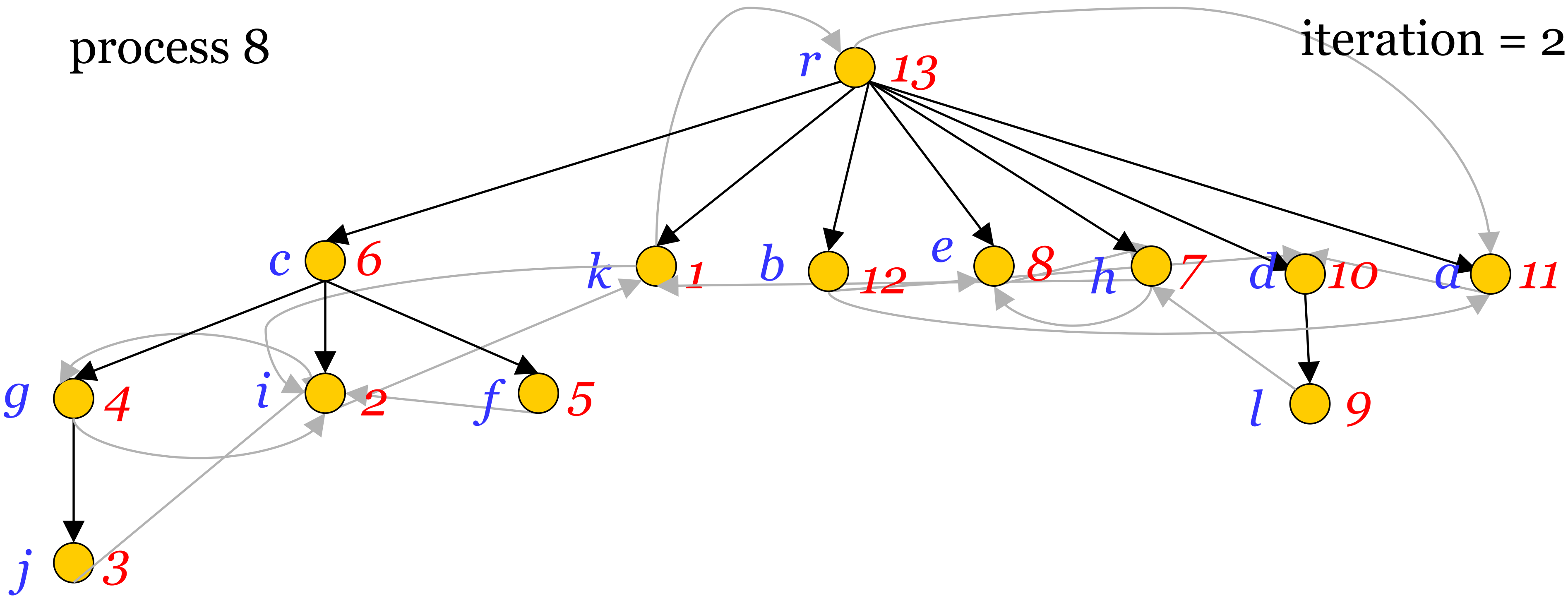
Iterative Algorithm: Example

process 8

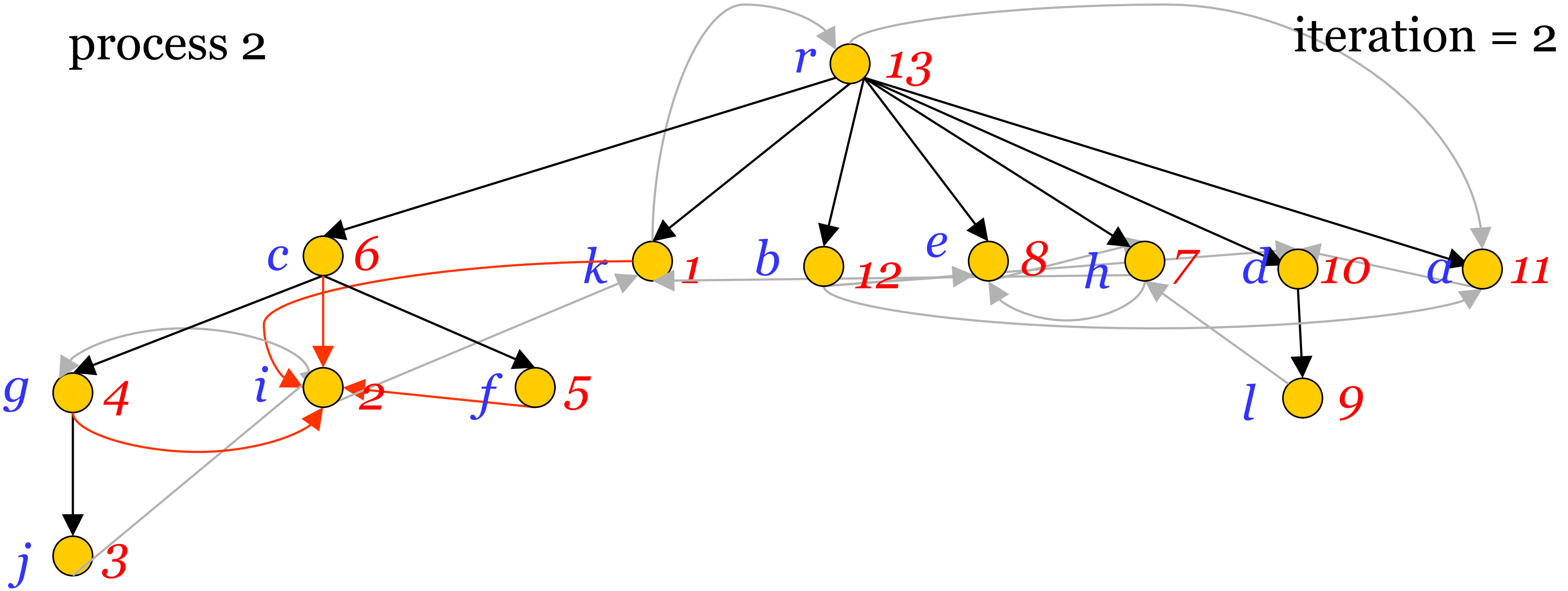
iteration = 2



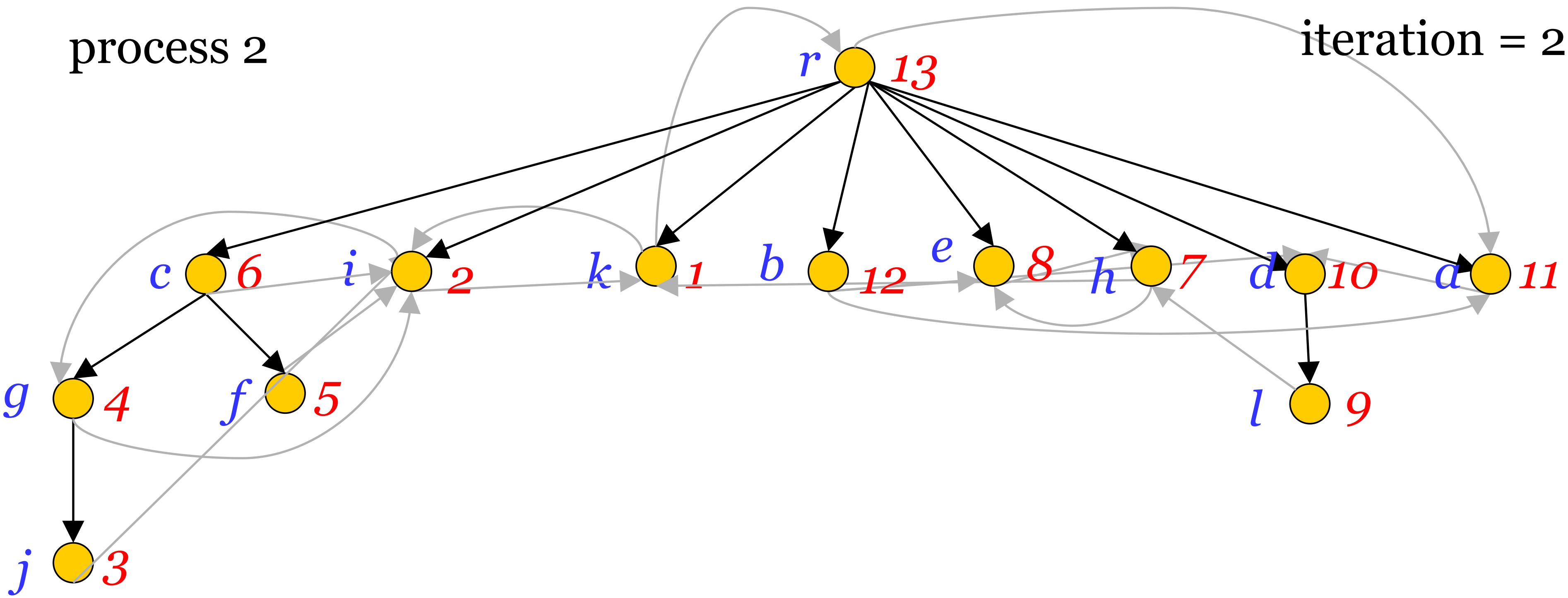
Iterative Algorithm: Example



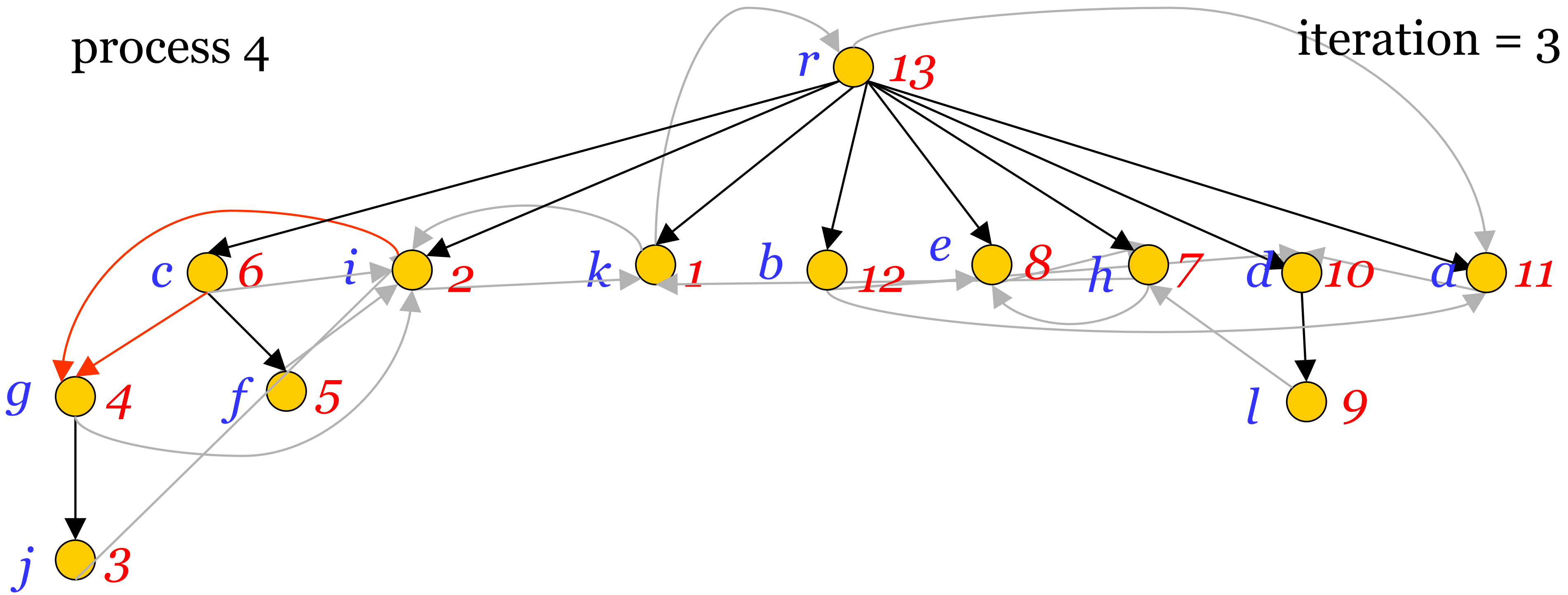
Iterative Algorithm: Example



Iterative Algorithm: Example



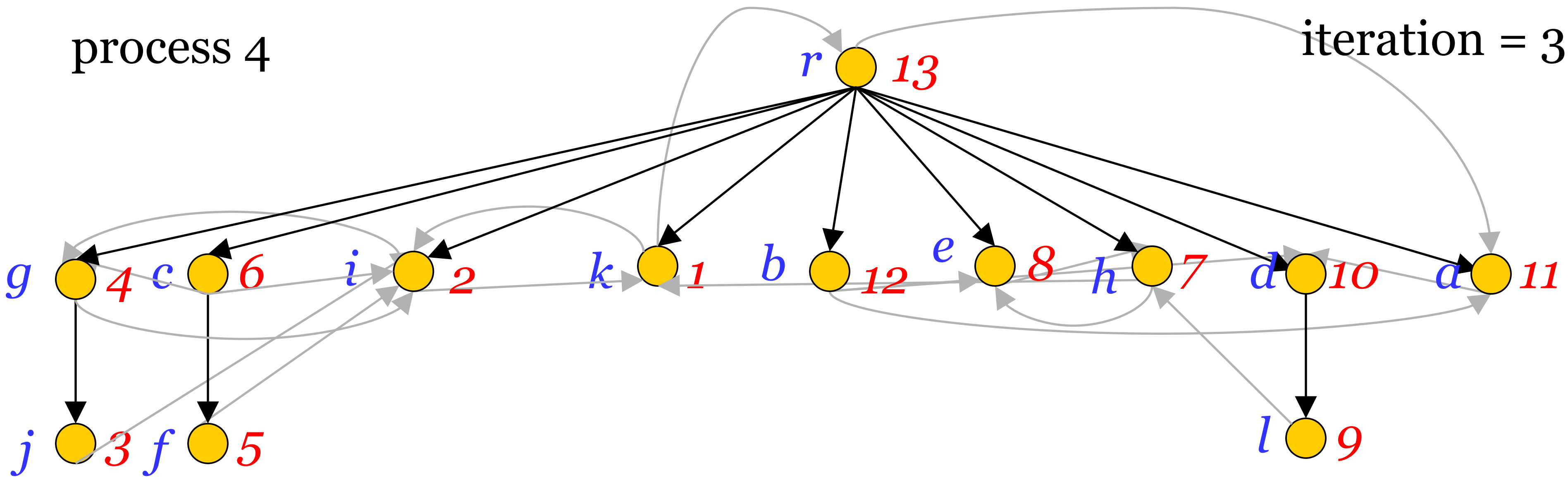
Iterative Algorithm: Example



Iterative Algorithm: Example

process 4

iteration = 3



DONE!

But we need one more iteration to verify that nothing changes

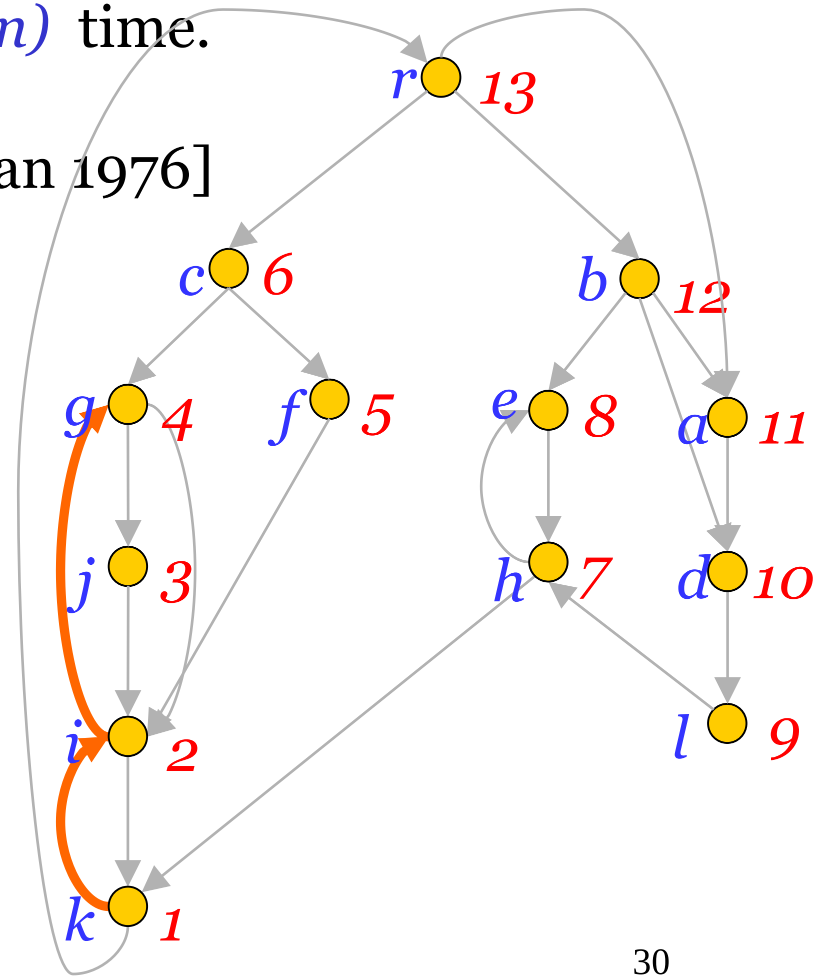
Iterative Algorithm

Running Time

Each pairwise intersection takes $O(n)$ time.

#iterations $\le d + 3$. [Kam and Ullman 1976]

d = max #back-edges in any cycle-free path of G



$d = 2$

Iterative Algorithm

Running Time

Each pairwise intersection takes $O(n)$ time.

The number of iterations is $\leq d + 3$.

d = max #back-edges in any cycle-free path of G

= $O(n)$

Running time = $O(mn^2)$

This bound is tight, but very pessimistic in practice.

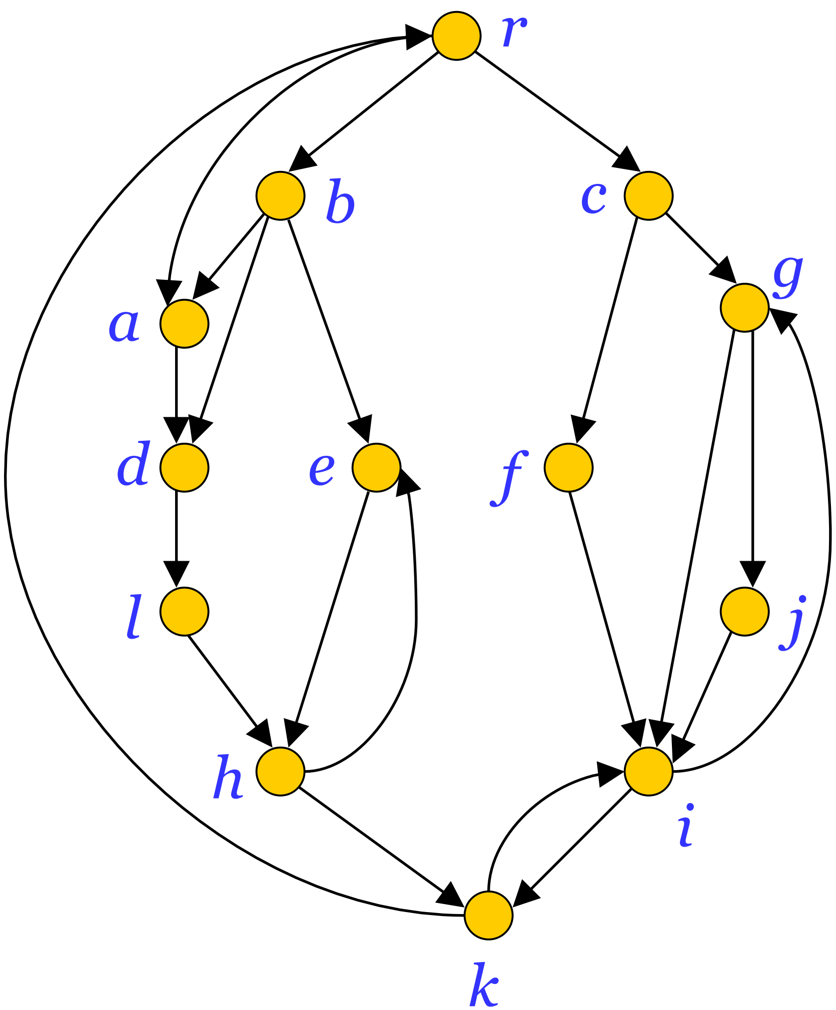
A Fast Dominator Algorithm

Lengauer-Tarjan [1979]: $O(m \cdot \alpha(m,n))$ time

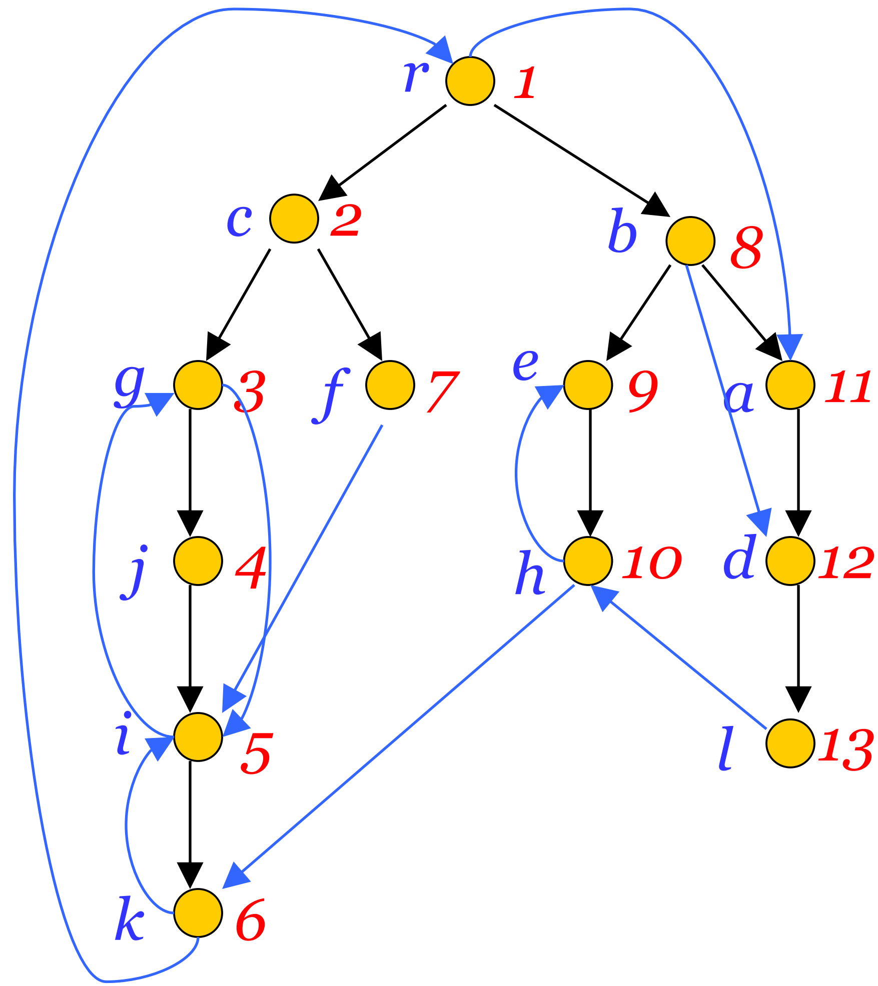
A simpler version runs in $O(m \cdot \log_{2+\lfloor m/n \rfloor} n)$ time

The Lengauer-Tarjan Algorithm: Depth-First Search

Perform a depth-first search on $G \Rightarrow$ DFS-tree T



$\rightarrow$



The Lengauer-Tarjan Algorithm: Depth-First Search

Depth-First Search Tree T :

We refer to the vertices by their DFS numbers:

$v < w$: v was visited by DFS before w

Notation

$v \overset{*}{\rightarrow} w$: v is an ancestor of w in T

$v \overset{+}{\rightarrow} w$: v is a proper ancestor of w in T

$parent(v)$: parent of v in T

Property 1

$$\forall\, v, w \text{ such that } v \leq w,\quad (v,w) \in E \quad \Rightarrow \quad v \overset{*}{\rightarrow} w$$

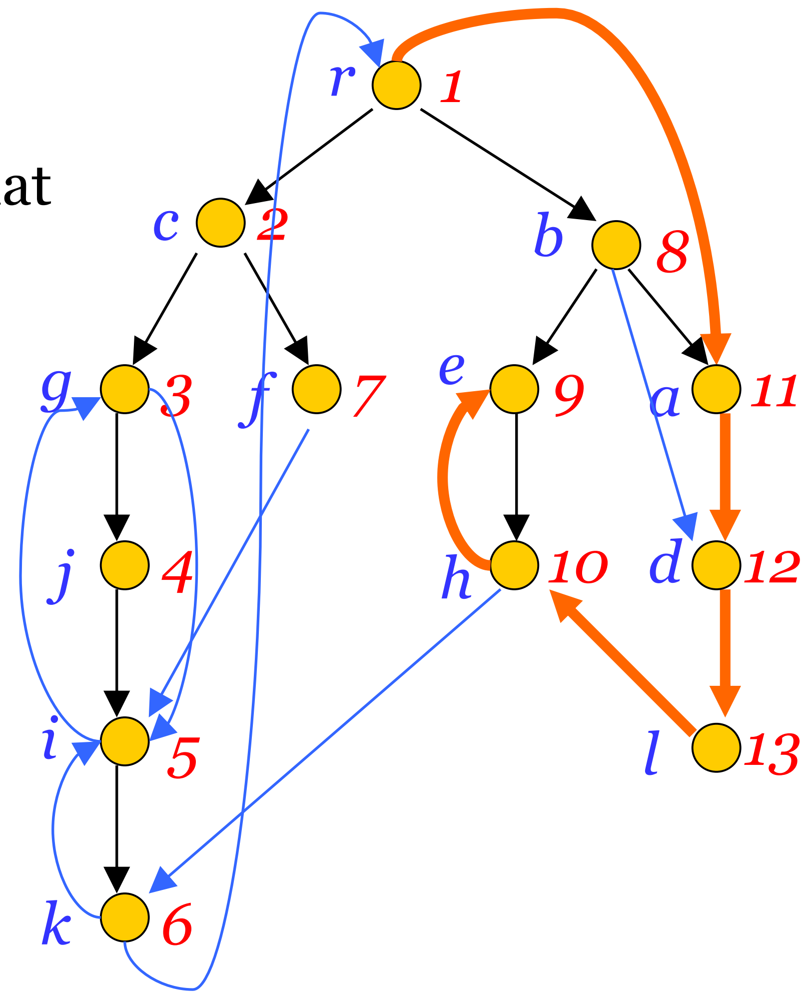
The Lengauer-Tarjan Algorithm: Semidominators

Semidominator path (SDOM-path):

$P = (v_0 = v, v_1, v_2, \ldots, v_k = w)$ such that

$v_i > w$, for $1 \le i \le k\text{-}1$

(r, a, d, l ,h, e) is an SDOM-path for e



The Lengauer-Tarjan Algorithm: Semidominators

Semidominator path (SDOM-path):

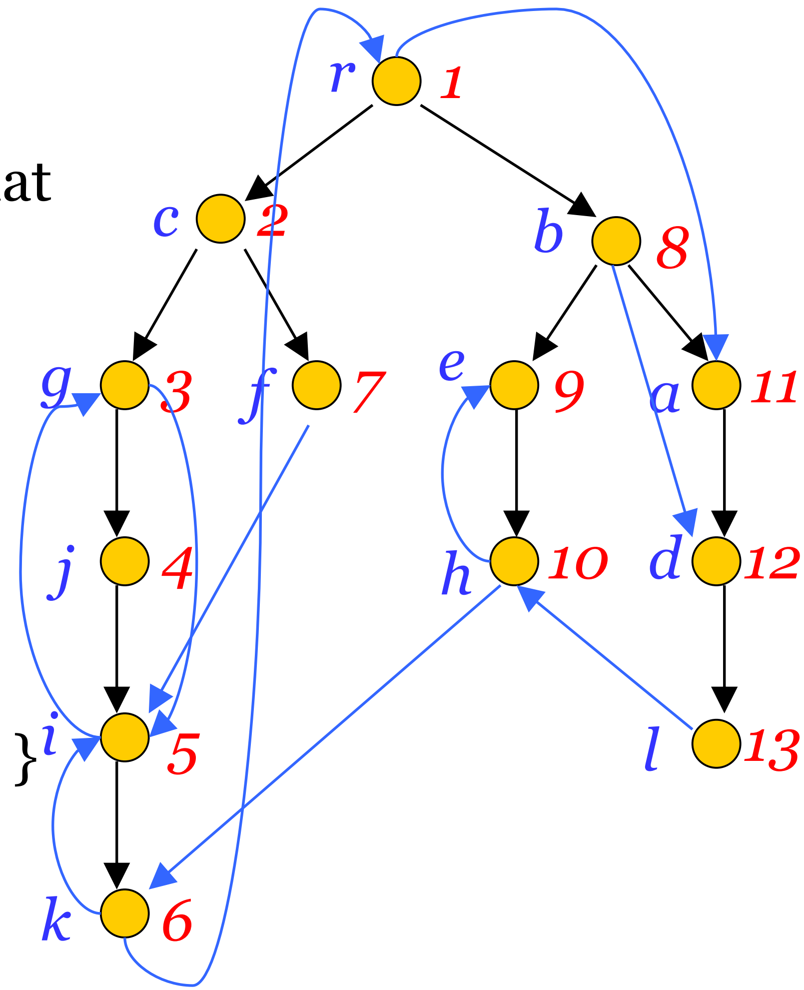
$P = (v_0 = v, v_1, v_2, \ldots, v_k = w)$ such that

$v_i > w$, for $1 \leq i \leq k\text{-}1$

Semidominator:

$sdom(w) =$

$\min \{ v \mid \exists$ SDOM-path from v to $w \}$



The Lengauer-Tarjan Algorithm: Semidominators

Semidominator path (SDOM-path):

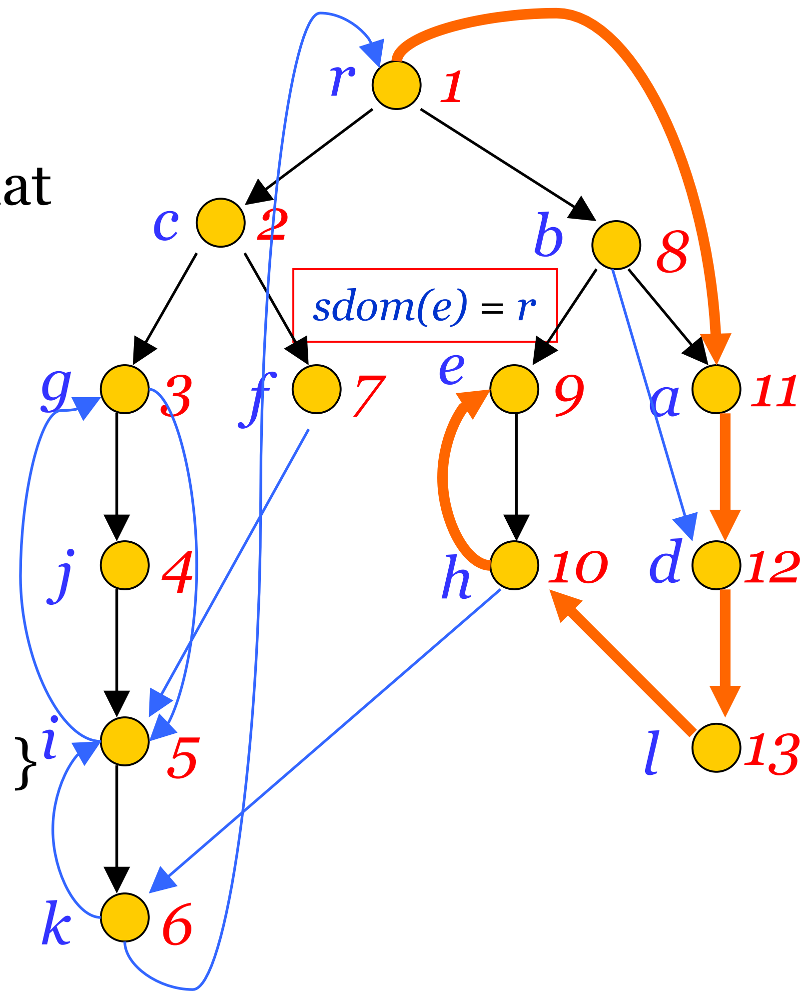
$P = (v_0 = v, v_1, v_2, \ldots, v_k = w)$ such that

$v_i > w$, for $1 \leq i \leq k-1$

Semidominator:

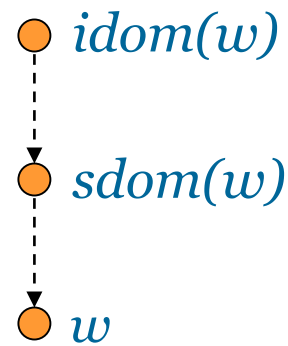
$sdom(w) =$

$\min \{ v \mid \exists$ SDOM-path from v to $w \}$



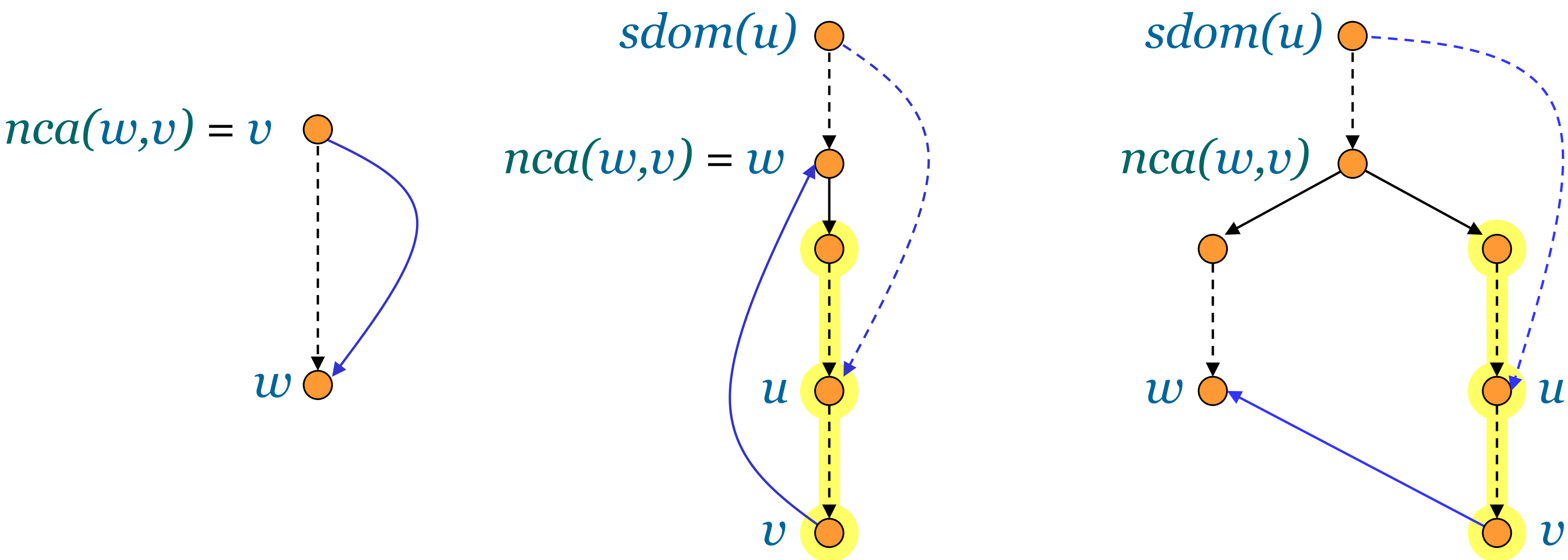
The Lengauer-Tarjan Algorithm: Semidominators

- For any $w \neq r$, $idom(w) \overset{*}{\rightarrow} sdom(w) \overset{+}{\rightarrow} w$.



The Lengauer-Tarjan Algorithm: Semidominators

- For any $w \neq r$, $idom(w) \overset{*}{\rightarrow} sdom(w) \overset{+}{\rightarrow} w$.
- $sdom(w) = \min \left(\{ v \mid (v, w) \in E \text{ and } v < w \} \cup \{ sdom(u) \mid u > w \text{ and } \exists (v, w) \in E \text{ such that } u \overset{*}{\rightarrow} v \} \right)$.

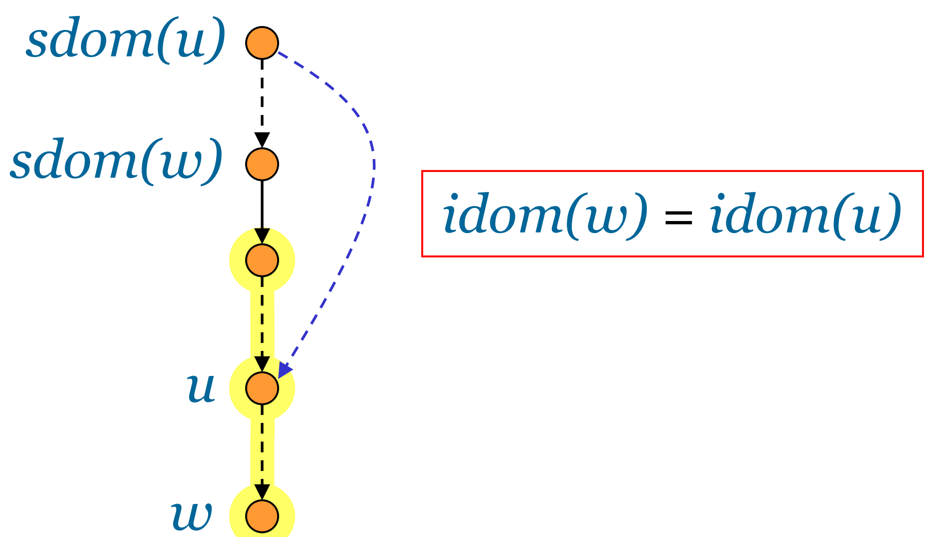


The Lengauer-Tarjan Algorithm: Semidominators

- For any $w \neq r$, $idom(w) \overset{*}{\rightarrow} sdom(w) \overset{+}{\rightarrow} w$.
- $sdom(w) = \min\left(\{ v \mid (v, w) \in E \text{ and } v < w \} \cup \right.$

  $\left. \{ sdom(u) \mid u > w \text{ and } \exists\, (v, w) \in E \text{ such that } u \overset{*}{\rightarrow} v \} \right)$.

- Let $w \neq r$ and let u be any vertex with min $sdom(u)$ that satisfies $sdom(w) \overset{+}{\rightarrow} u \overset{*}{\rightarrow} w$. Then $idom(w) = idom(u)$.

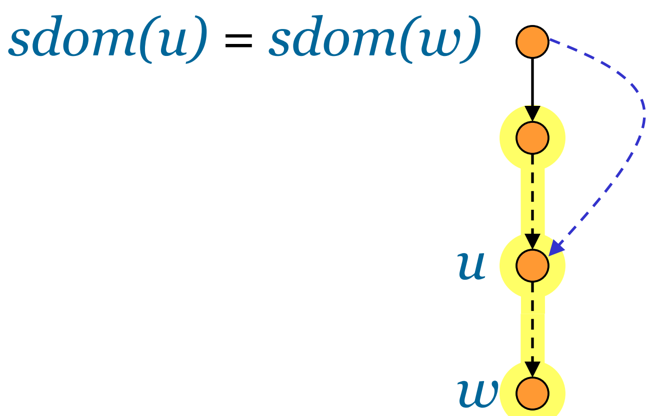


The Lengauer-Tarjan Algorithm: Semidominators

- For any $w \neq r$, $idom(w) \overset{*}{\rightarrow} sdom(w) \overset{+}{\rightarrow} w$.
- $sdom(w) = \min(\{ v \mid (v, w) \in E \text{ and } v < w \} \cup$

  $\{ sdom(u) \mid u > w \text{ and } \exists (v, w) \in E \text{ such that } u \overset{*}{\rightarrow} v \})$.

- Let $w \neq r$ and let u be any vertex with $\min sdom(u)$ that satisfies $sdom(w) \overset{+}{\rightarrow} u \overset{*}{\rightarrow} w$. Then $idom(w) = idom(u)$. Moreover, if $sdom(u) = sdom(w)$ then $idom(w) = sdom(w)$.



$\boxed{idom(w) = sdom(w)}$

Overview of the Algorithm

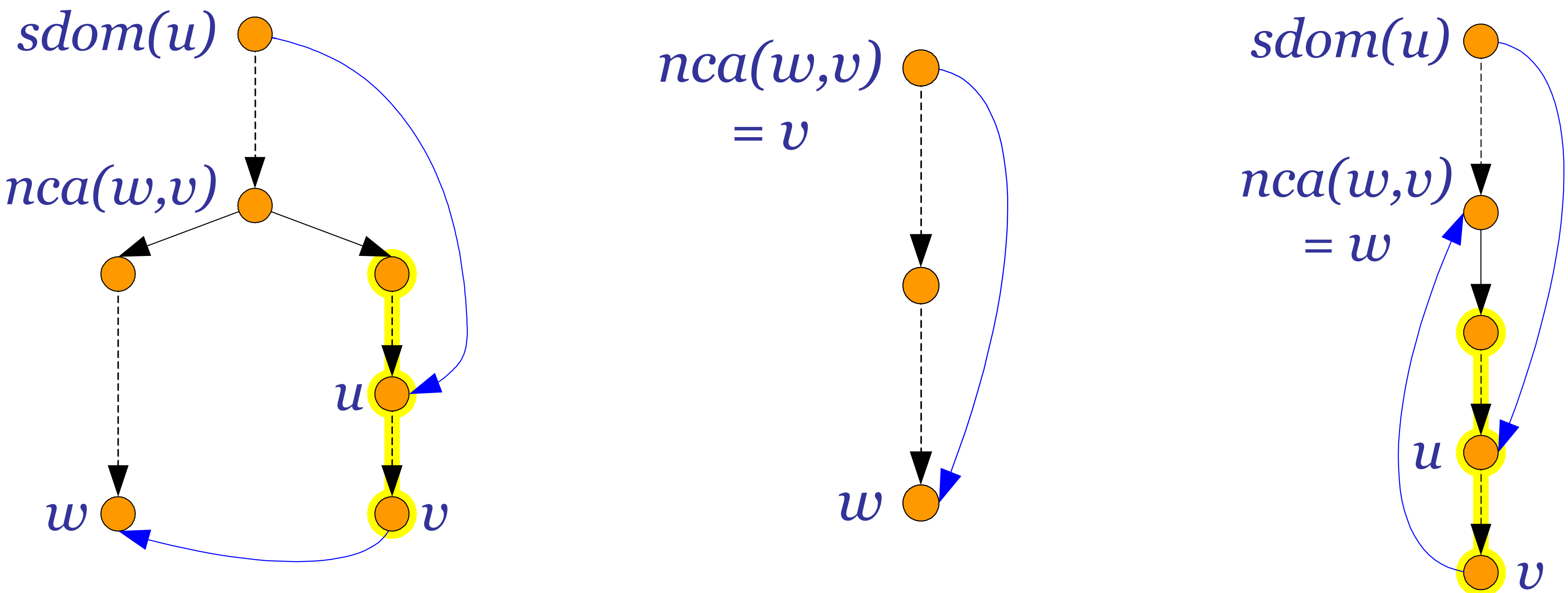
1. Carry out a DFS.

2. Process the vertices in reverse preorder. For vertex w, compute $sdom(w)$.

3. Implicitly define $idom(w)$.

4. Explicitly define $idom(w)$ by a preorder pass.

Evaluating minima on tree paths



If we process vertices in reverse preorder then the *sdom* values we need are known.

Evaluating minima on tree paths

Data Structure: Maintain forest F and supports the operations:

link(v, w): Add the edge (v,w) to F.

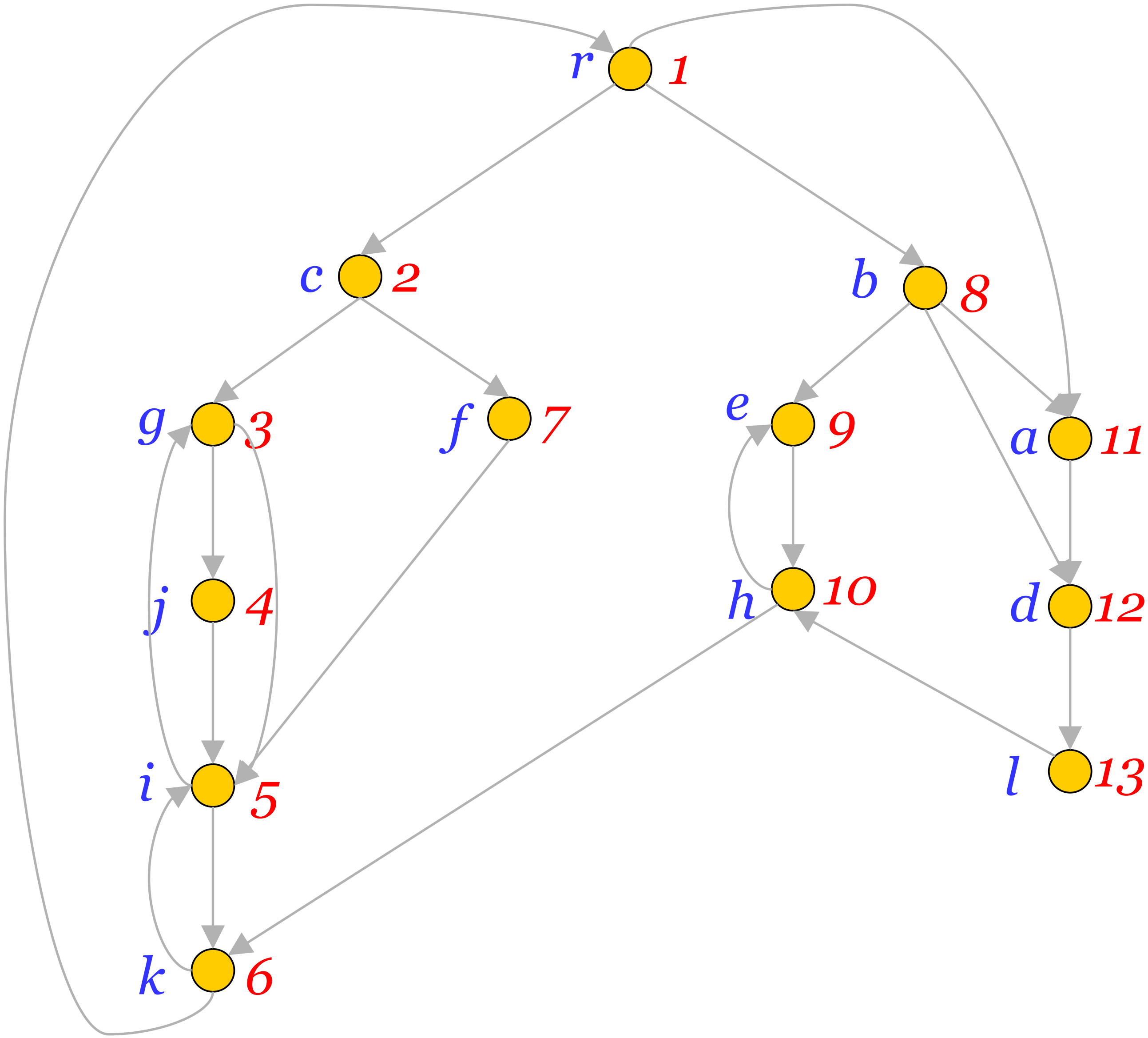
eval(v): Let r be the root of the tree that contains v in F.
If $v = \mathrm{r}$ then return v. Otherwise return any vertex with minimum $sdom$ among the vertices u that satisfy $r \xrightarrow{+} u \xrightarrow{*} v$.

Initially every vertex in V is a root in F.

The Lengauer-Tarjan Algorithm

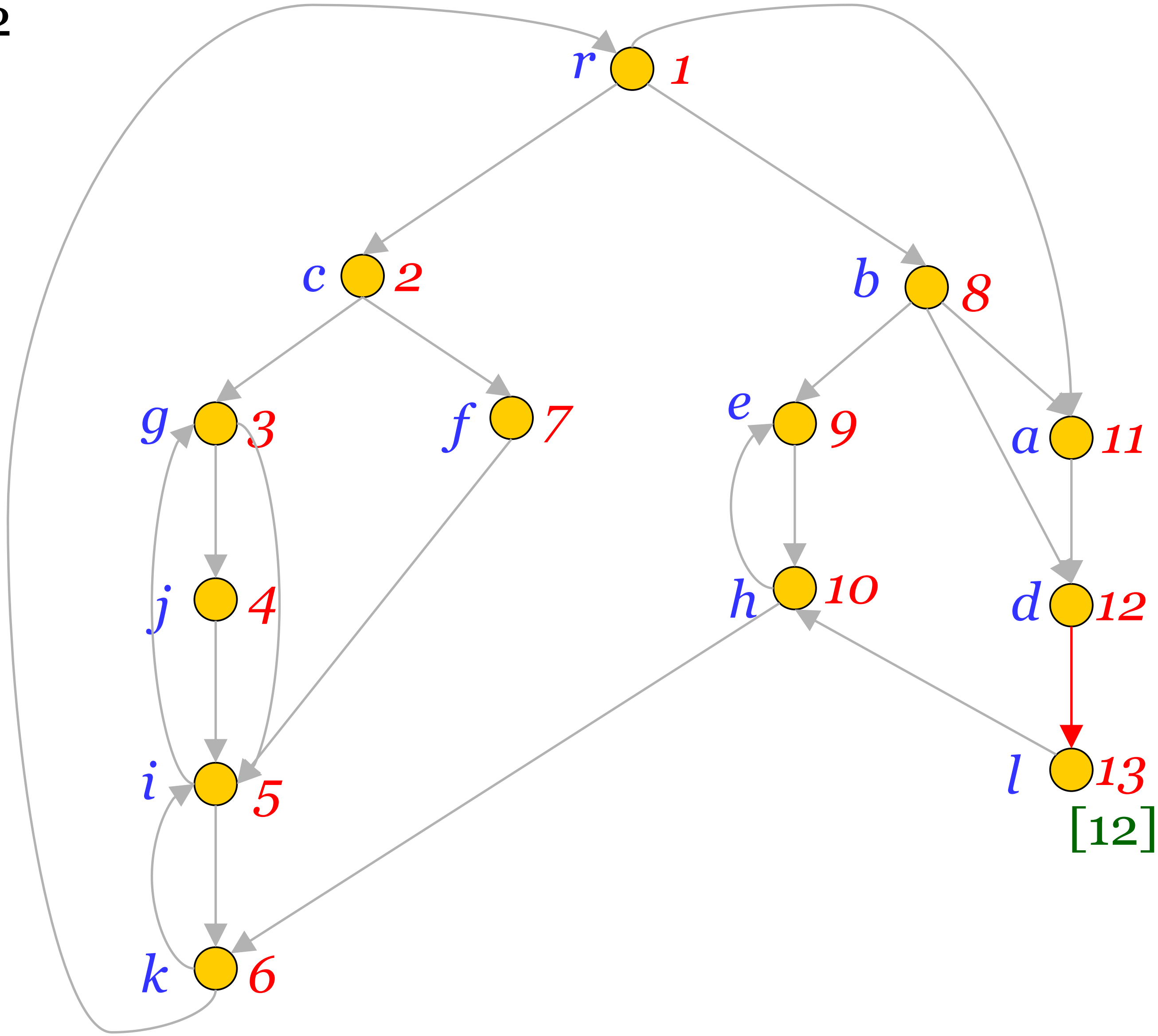
```
dfs(r)
for all w ∈ V in reverse preorder do
        for all v ∈ pred(w) do
                u ← eval(v)
                if semi(u) < semi(w) then semi(w) ← semi(u)
        done
        add w to the bucket of semi(w)
        link(parent(w), w)
        for all v in the bucket of parent(w) do
                delete v from the bucket of parent(w)
                u ← eval(v)
                if semi(u) < semi(v) then dom(v) ← u
                else dom(v) ← parent(w)
        done
done
for all w ∈ V in reverse preorder do
        if dom(w) ≠ semi(w) then dom(w) ← dom(dom(w))
done
```

The Lengauer-Tarjan Algorithm: Example



The Lengauer-Tarjan Algorithm: Example

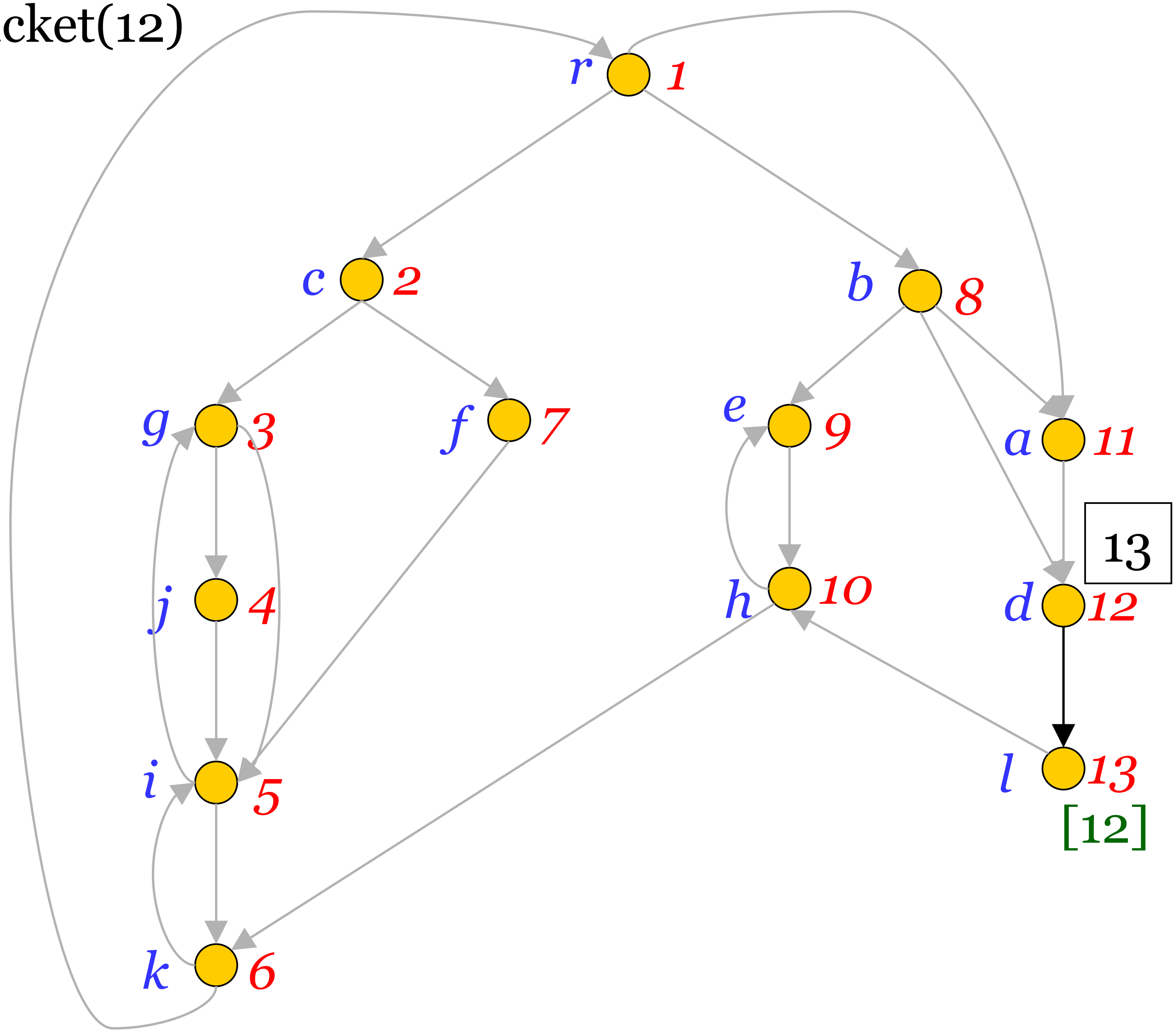
eval(*12*) = 12



The Lengauer-Tarjan Algorithm: Example

add 13 to bucket(12)

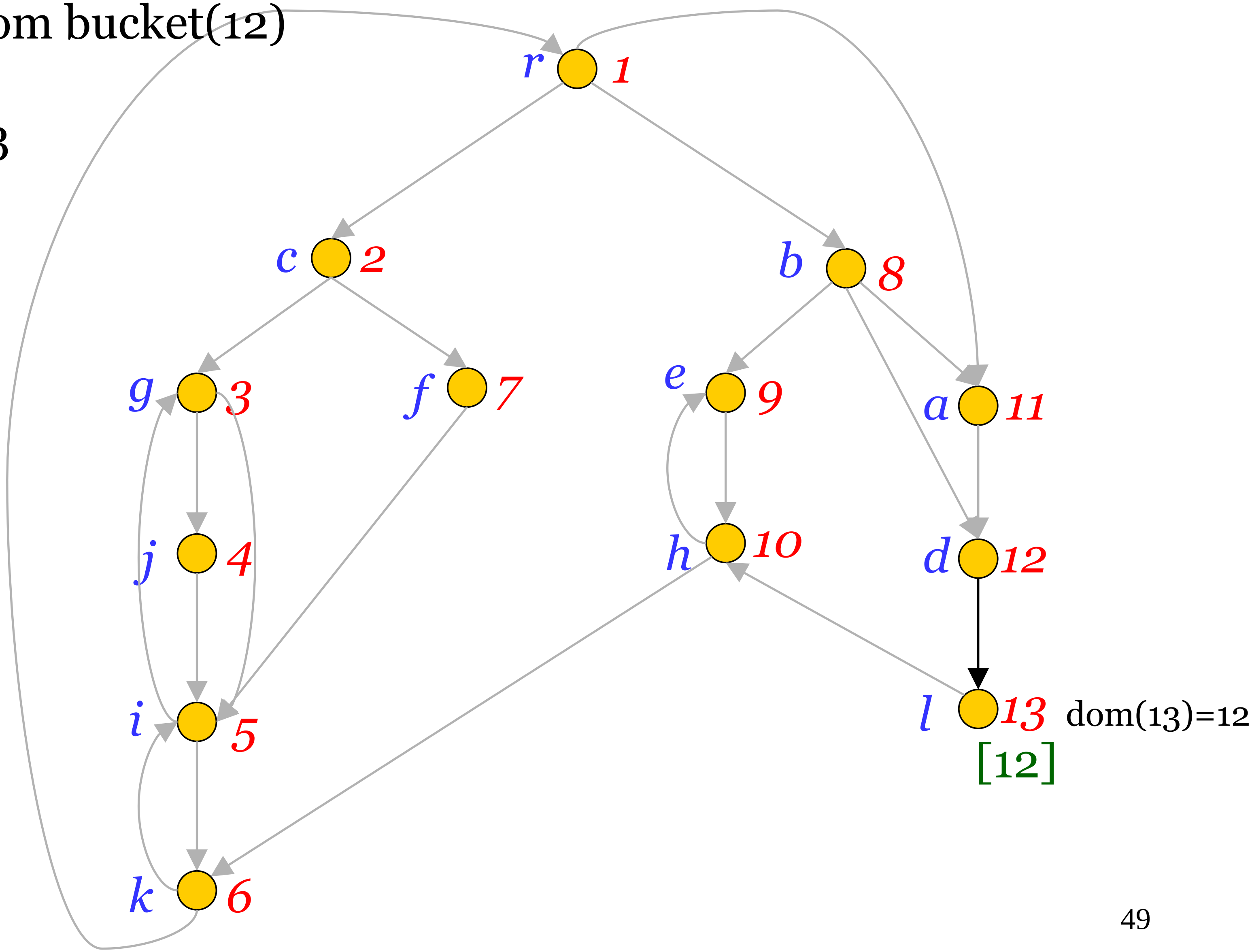
link(*13*)



The Lengauer-Tarjan Algorithm: Example

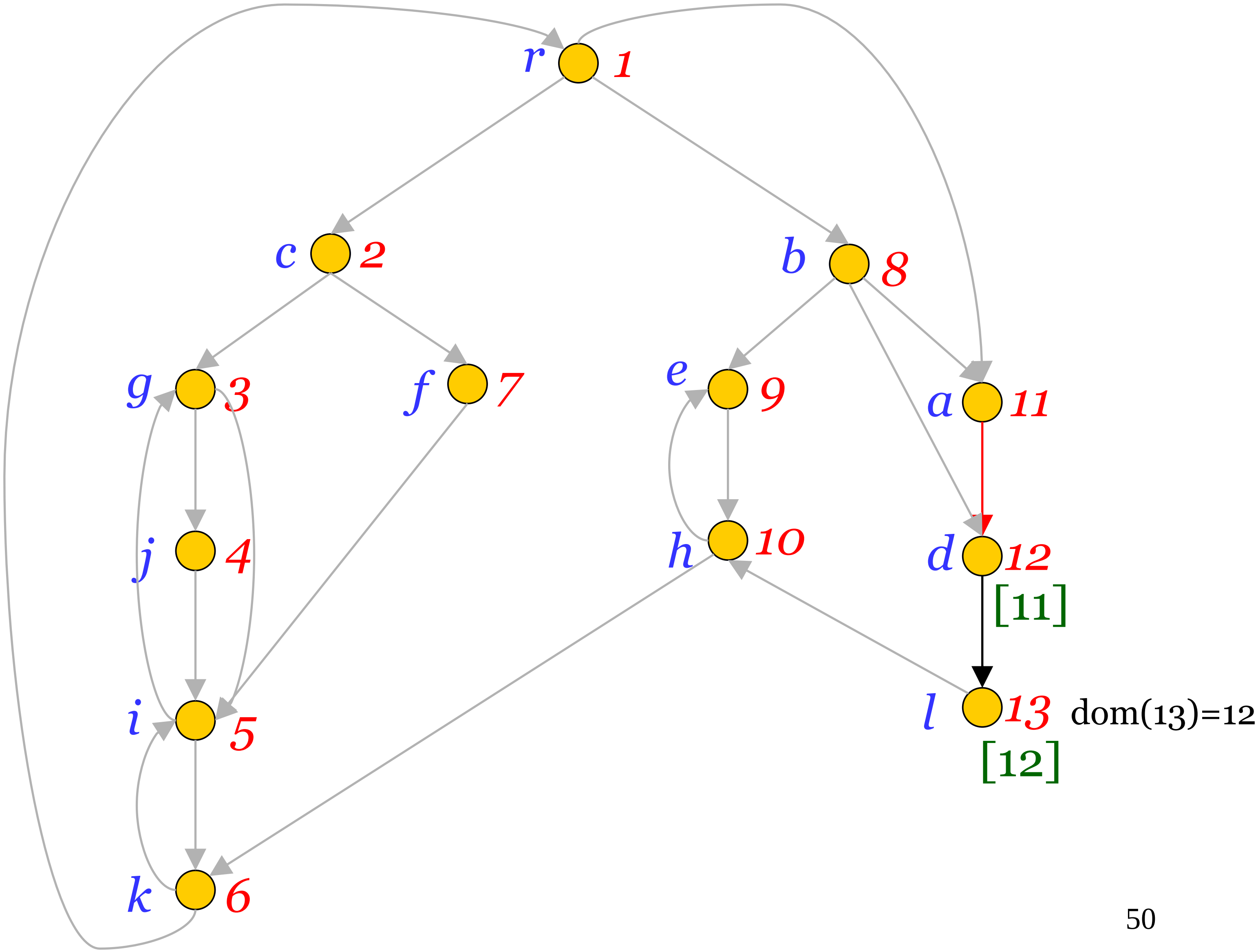
delete 13 from bucket(12)

eval(13) = 13



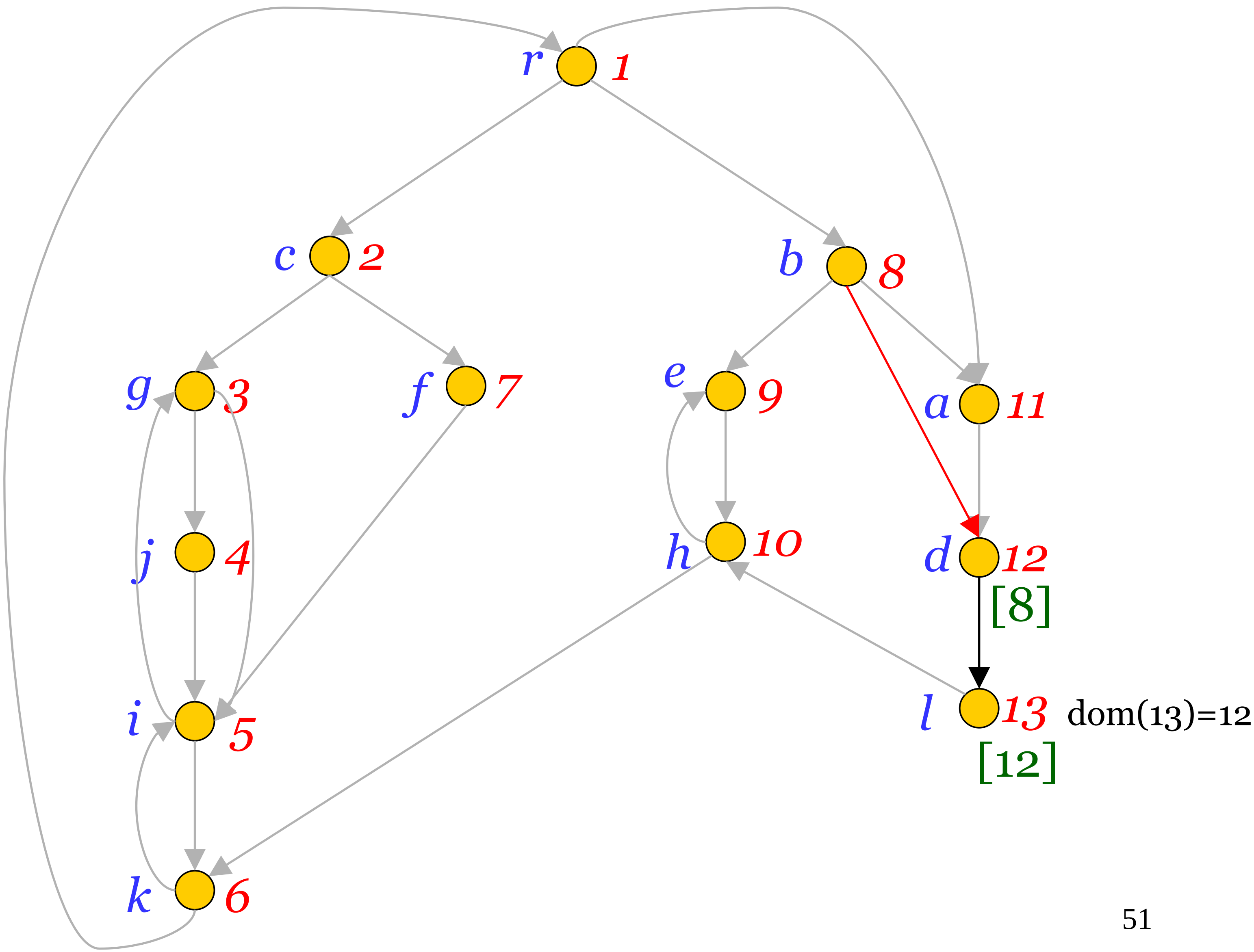
The Lengauer-Tarjan Algorithm: Example

eval(11) = 11



The Lengauer-Tarjan Algorithm: Example

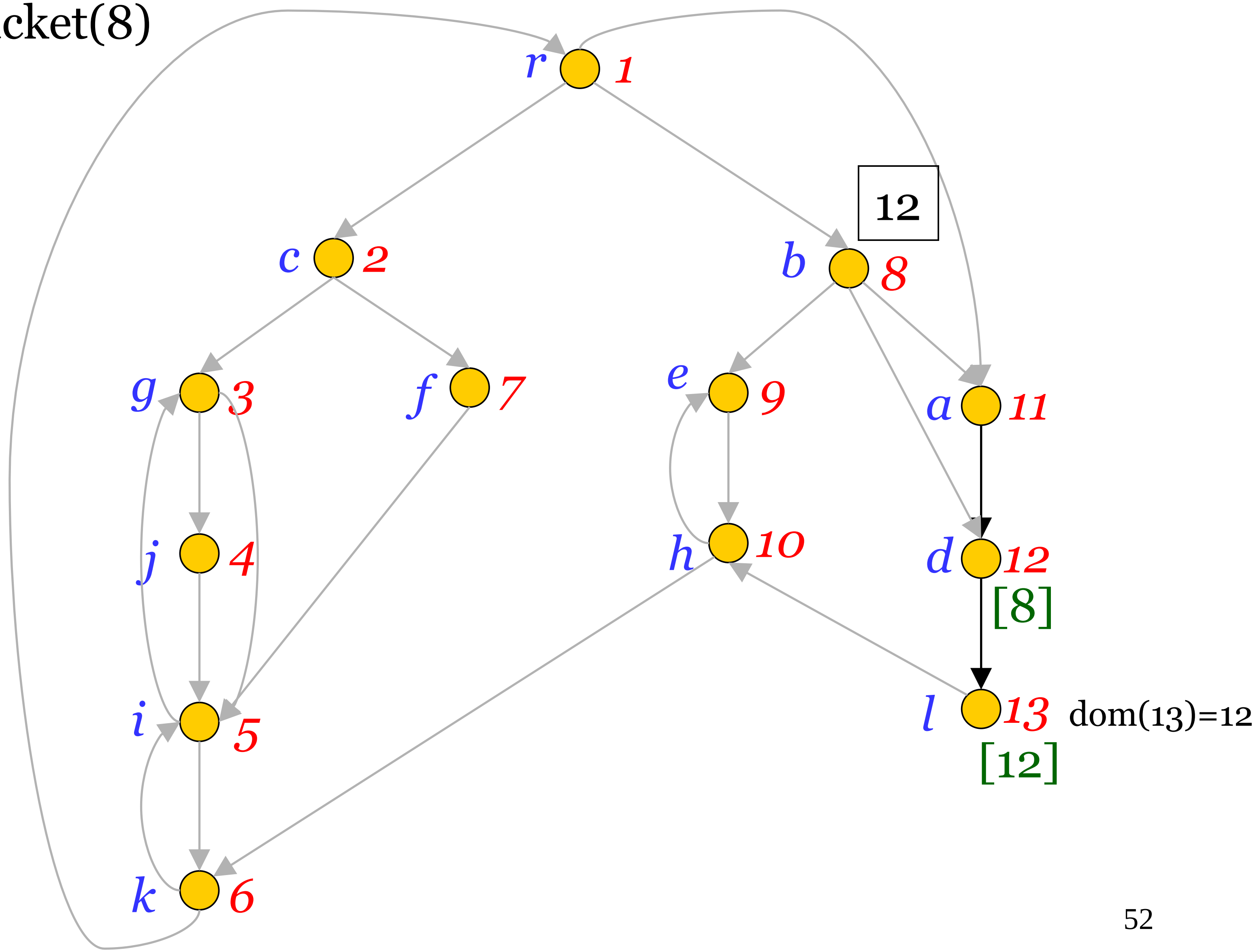
eval(8) = 8



The Lengauer-Tarjan Algorithm: Example

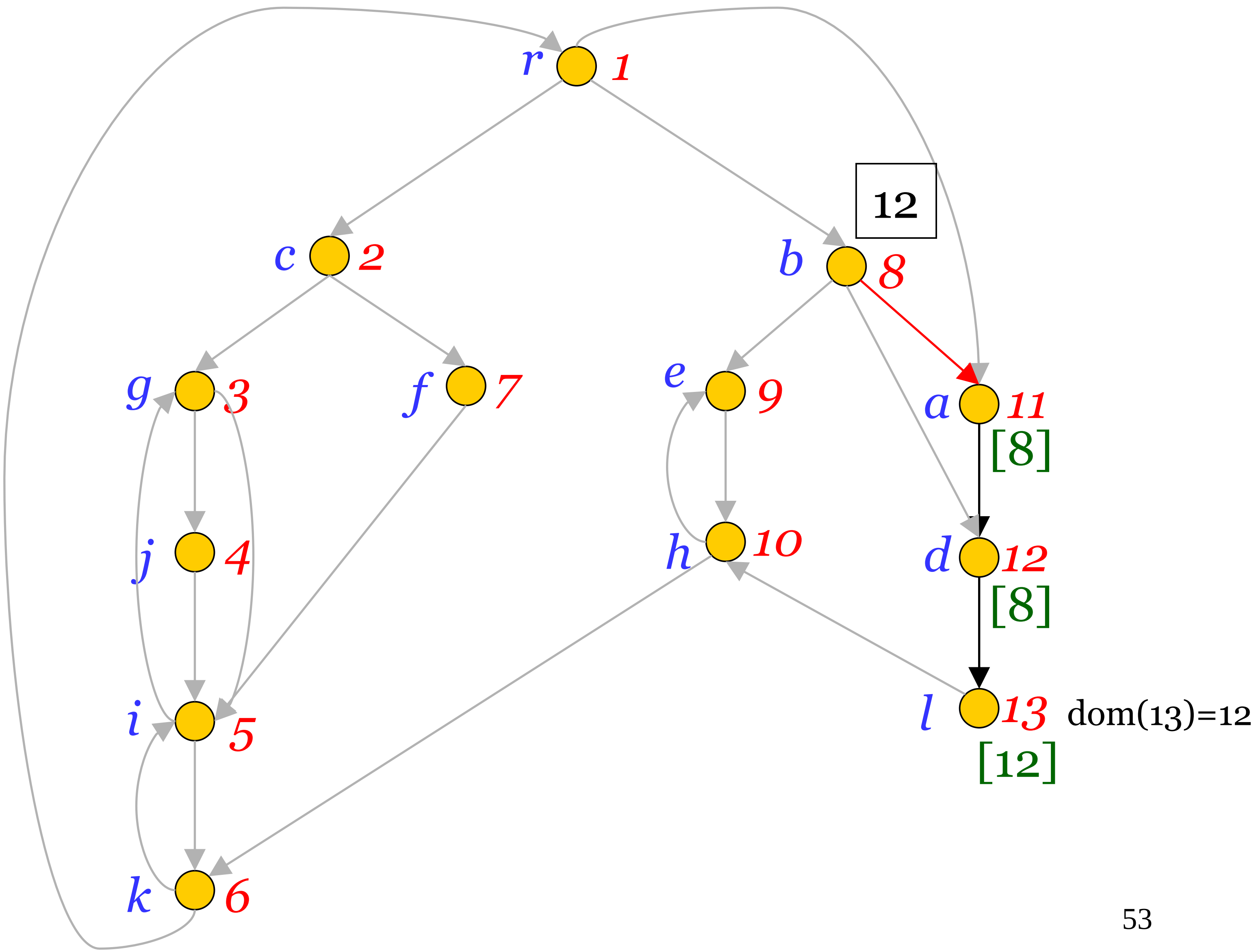
add 12 to bucket(8)

link(12)



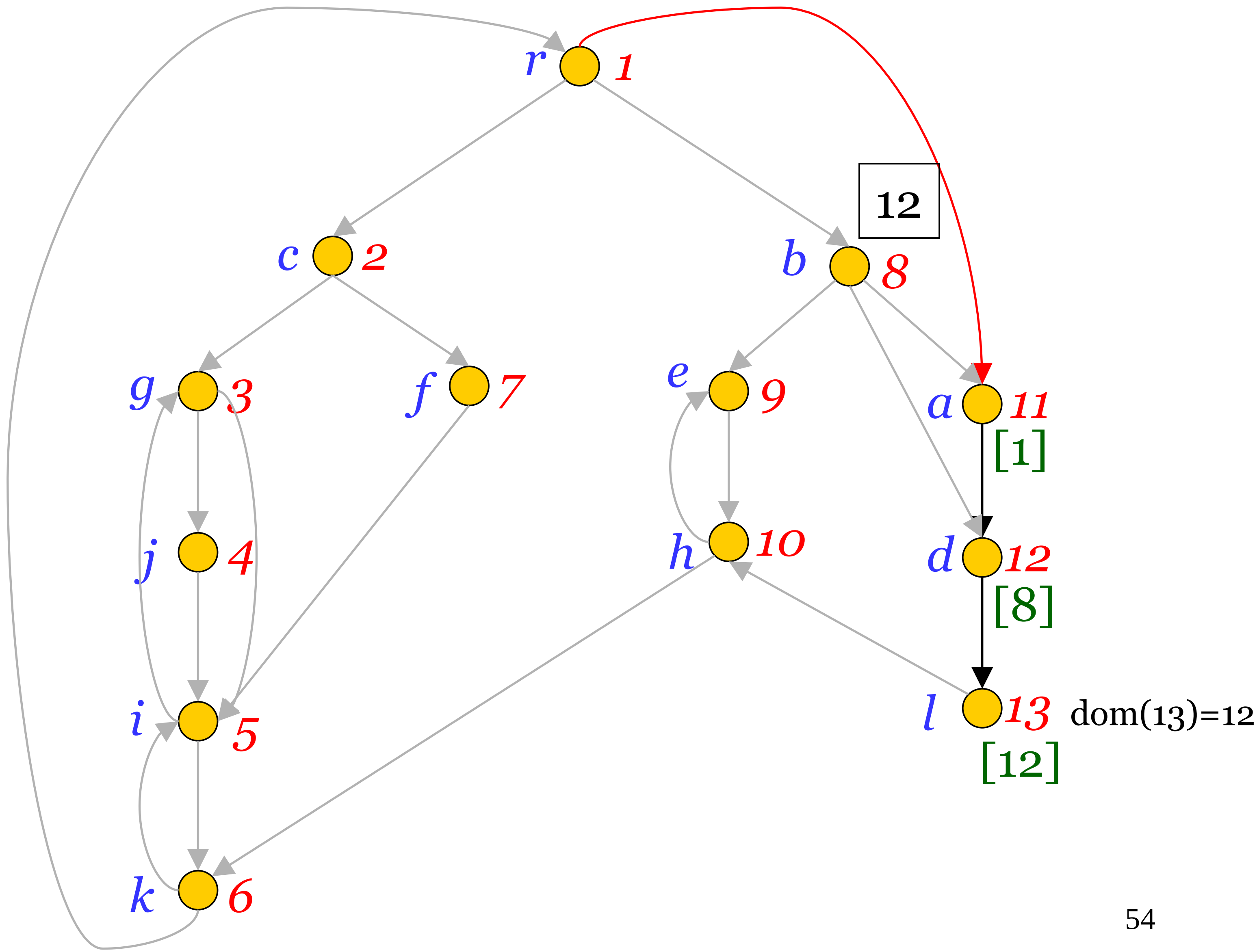
The Lengauer-Tarjan Algorithm: Example

eval(8)=8



The Lengauer-Tarjan Algorithm: Example

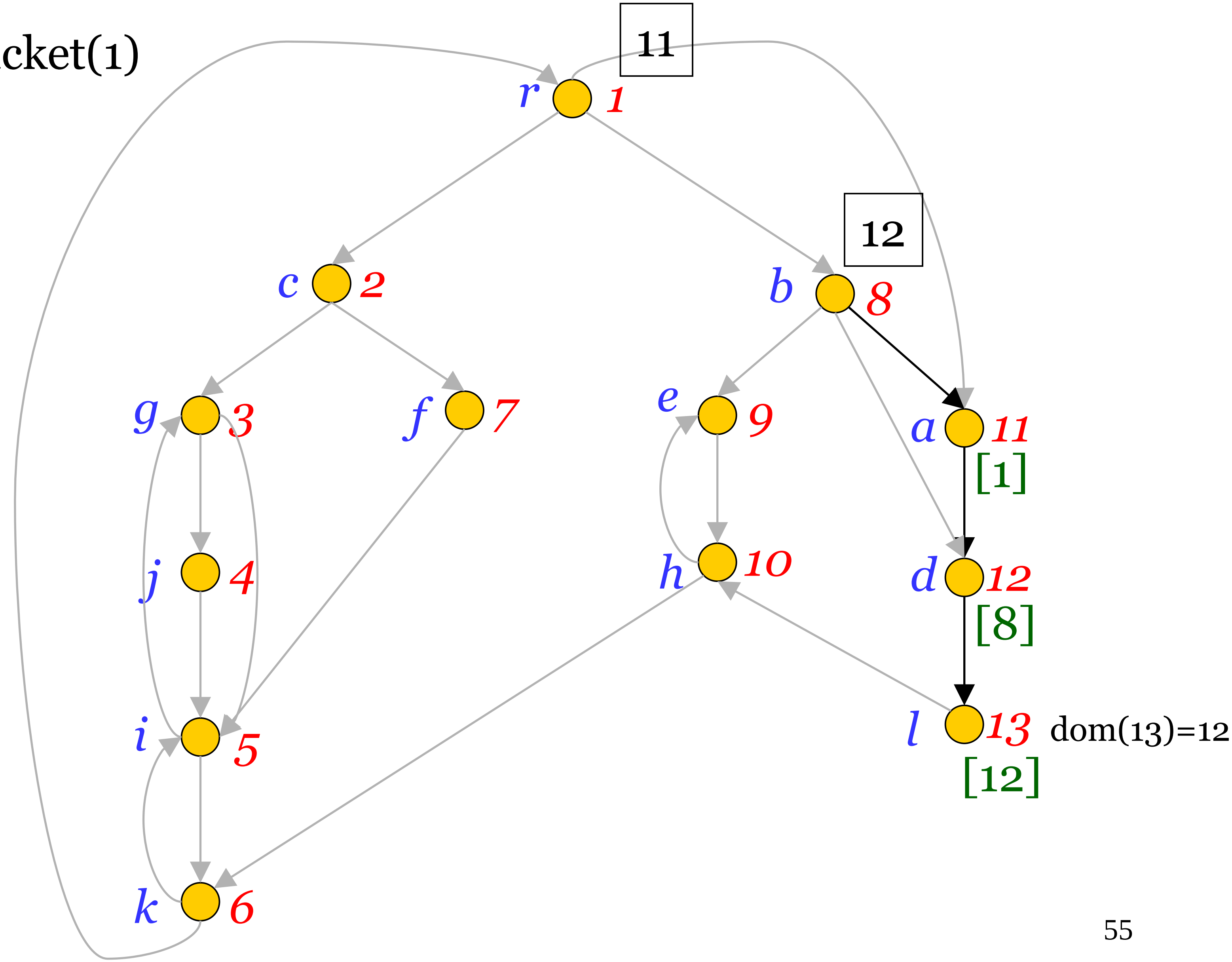
eval(1)=1



The Lengauer-Tarjan Algorithm: Example

add 11 to bucket(1)

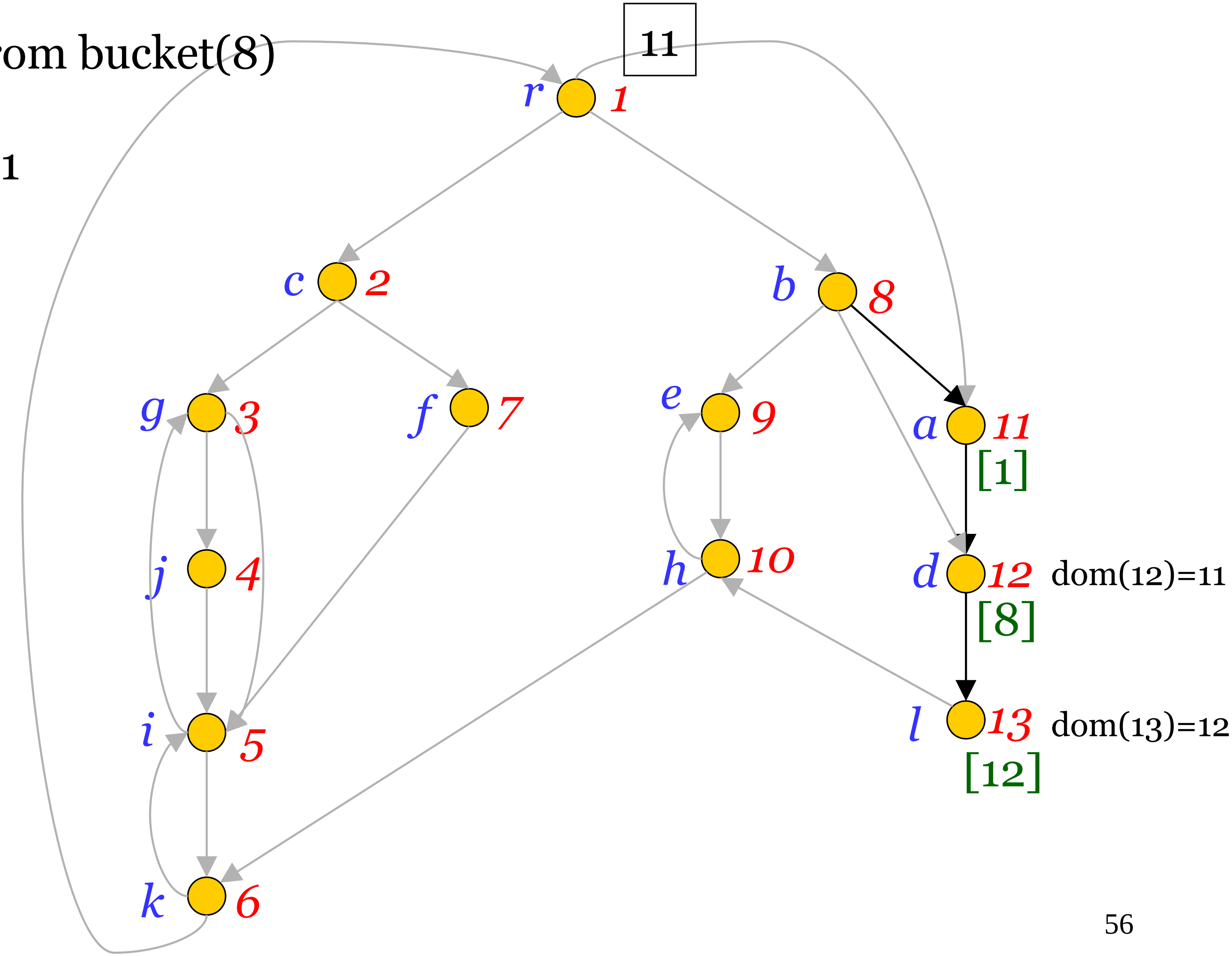
link(11)



The Lengauer-Tarjan Algorithm: Example

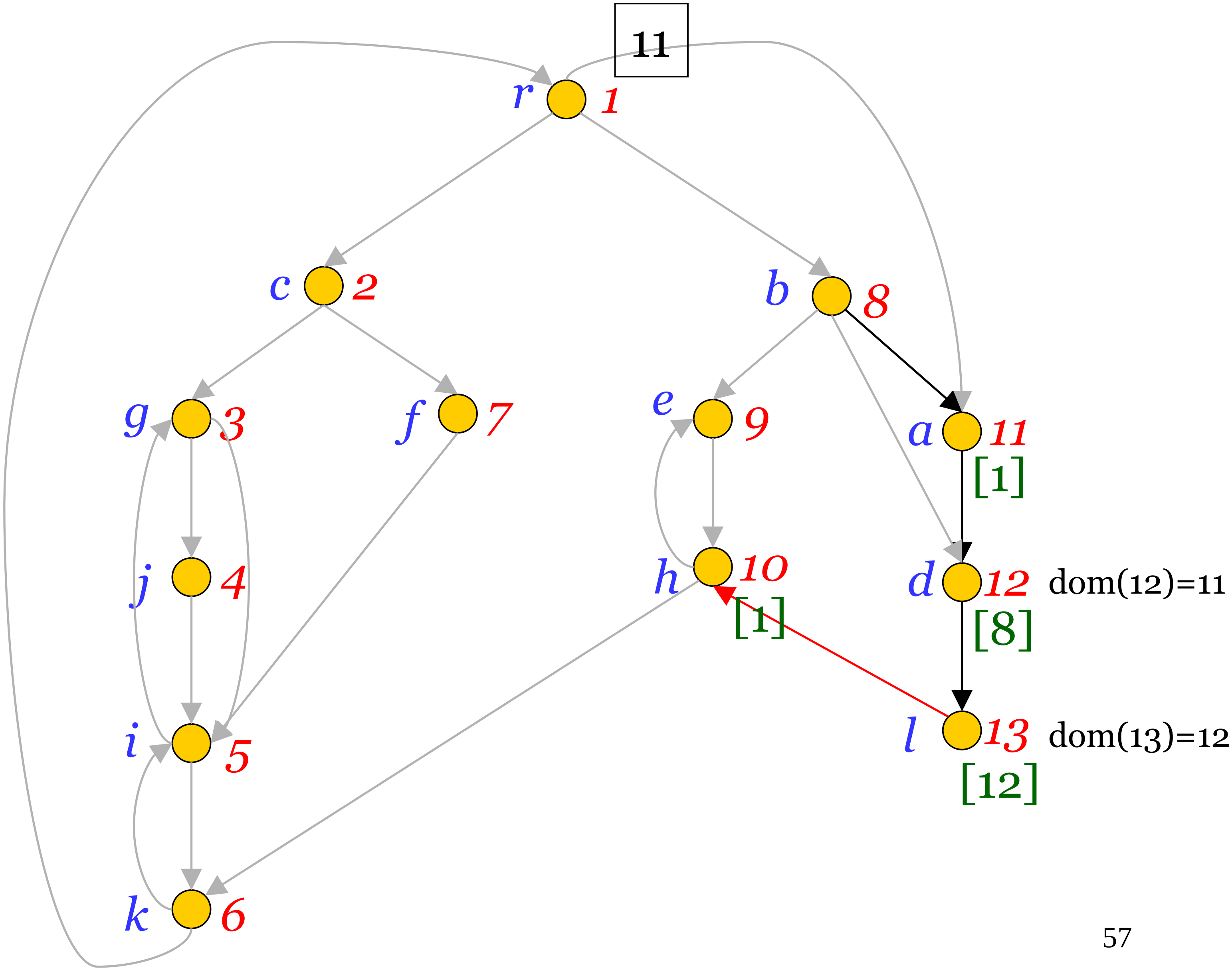
delete 12 from bucket(8)

eval(12) = 11



The Lengauer-Tarjan Algorithm: Example

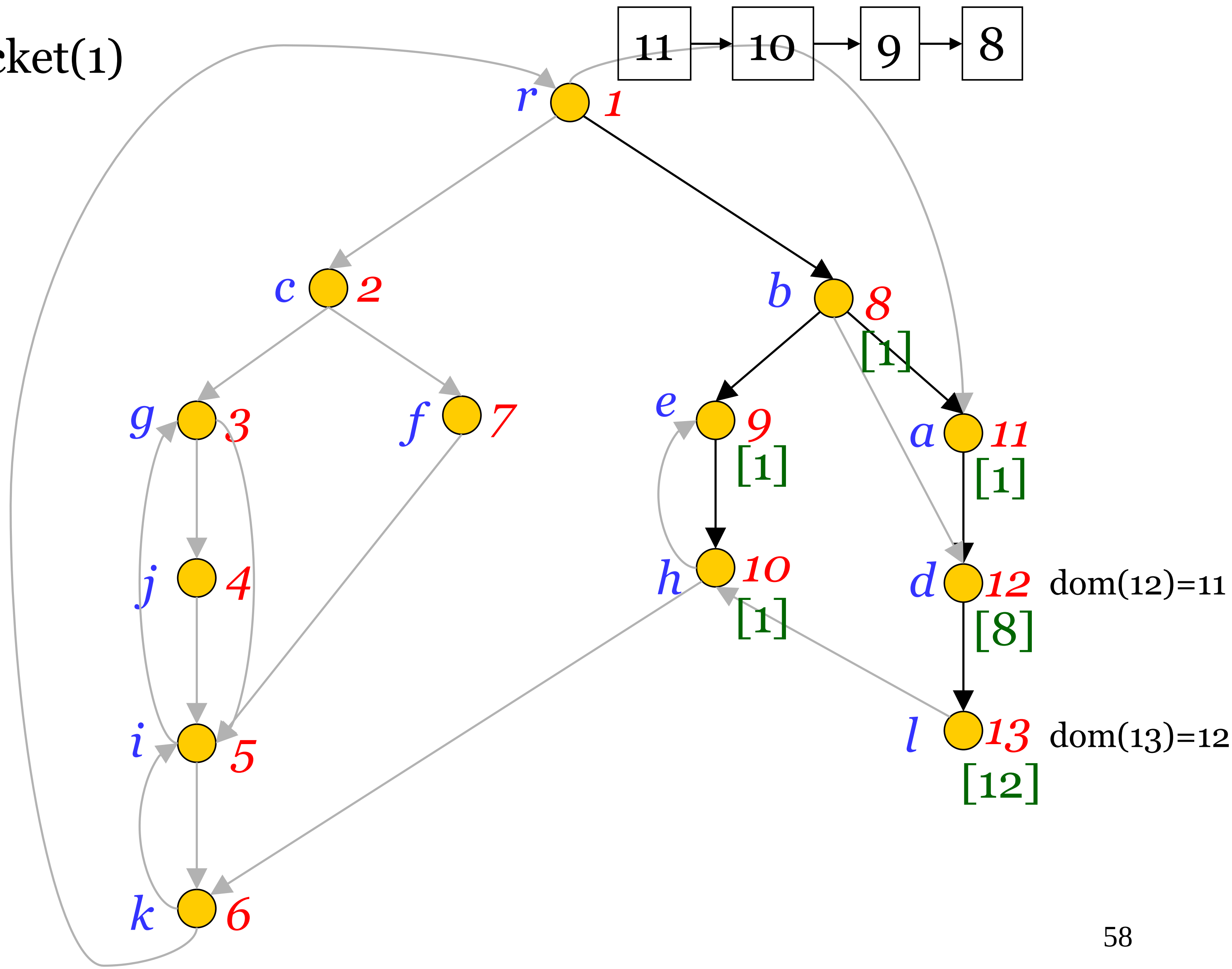
eval(13) = 11



The Lengauer-Tarjan Algorithm: Example

add 8 to bucket(1)

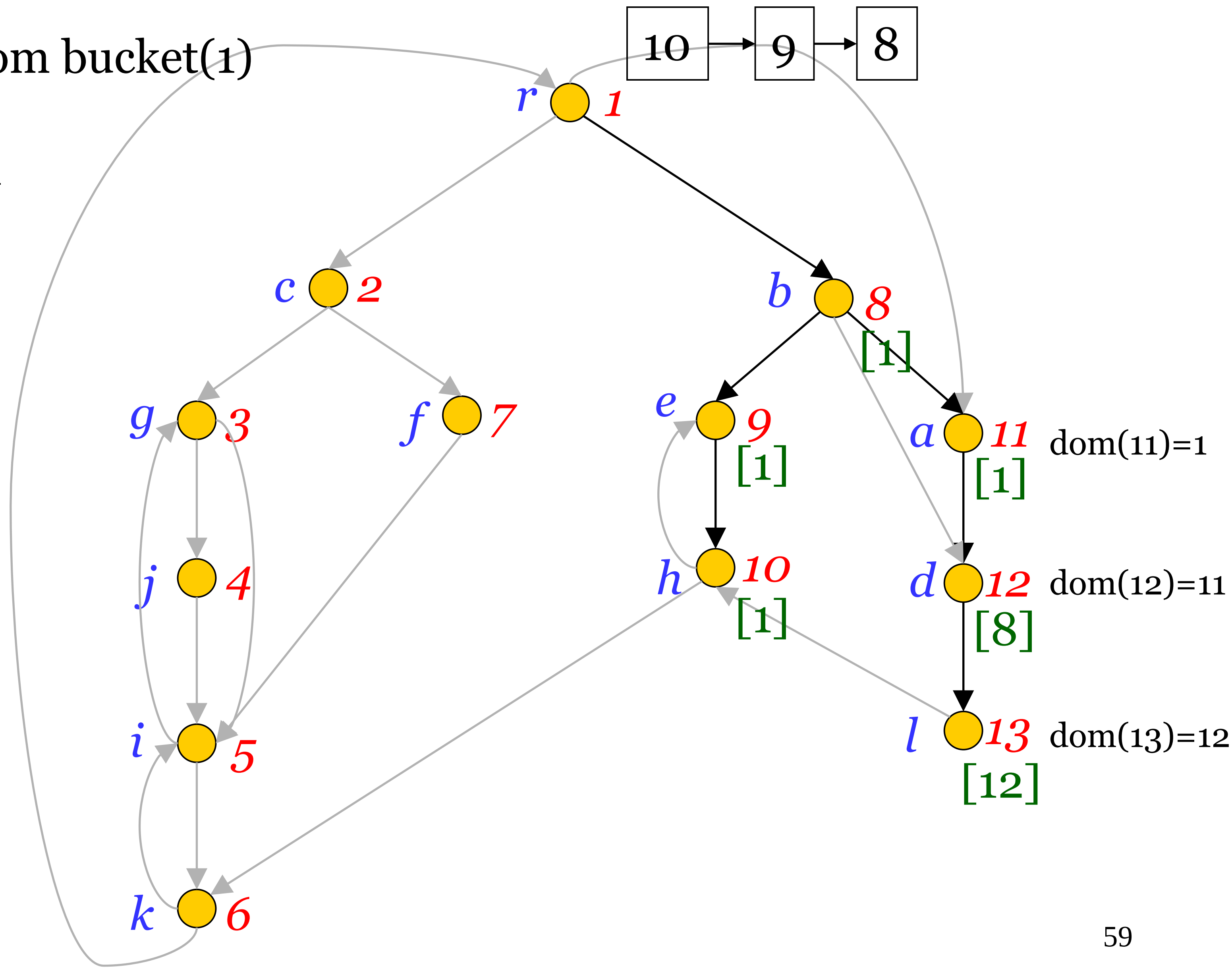
link(8)



The Lengauer-Tarjan Algorithm: Example

delete 11 from bucket(1)

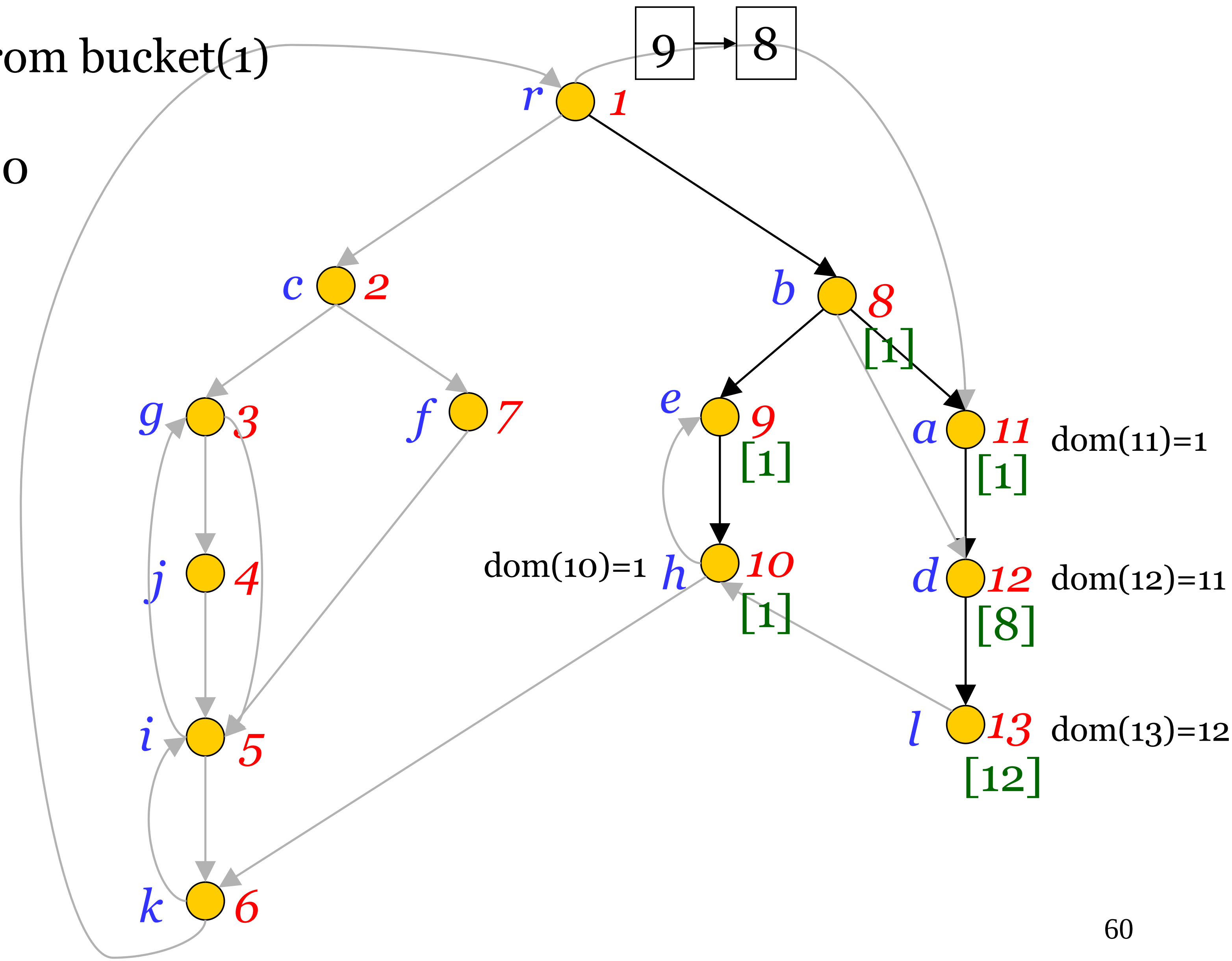
eval(11) = 11



The Lengauer-Tarjan Algorithm: Example

delete 10 from bucket(1)

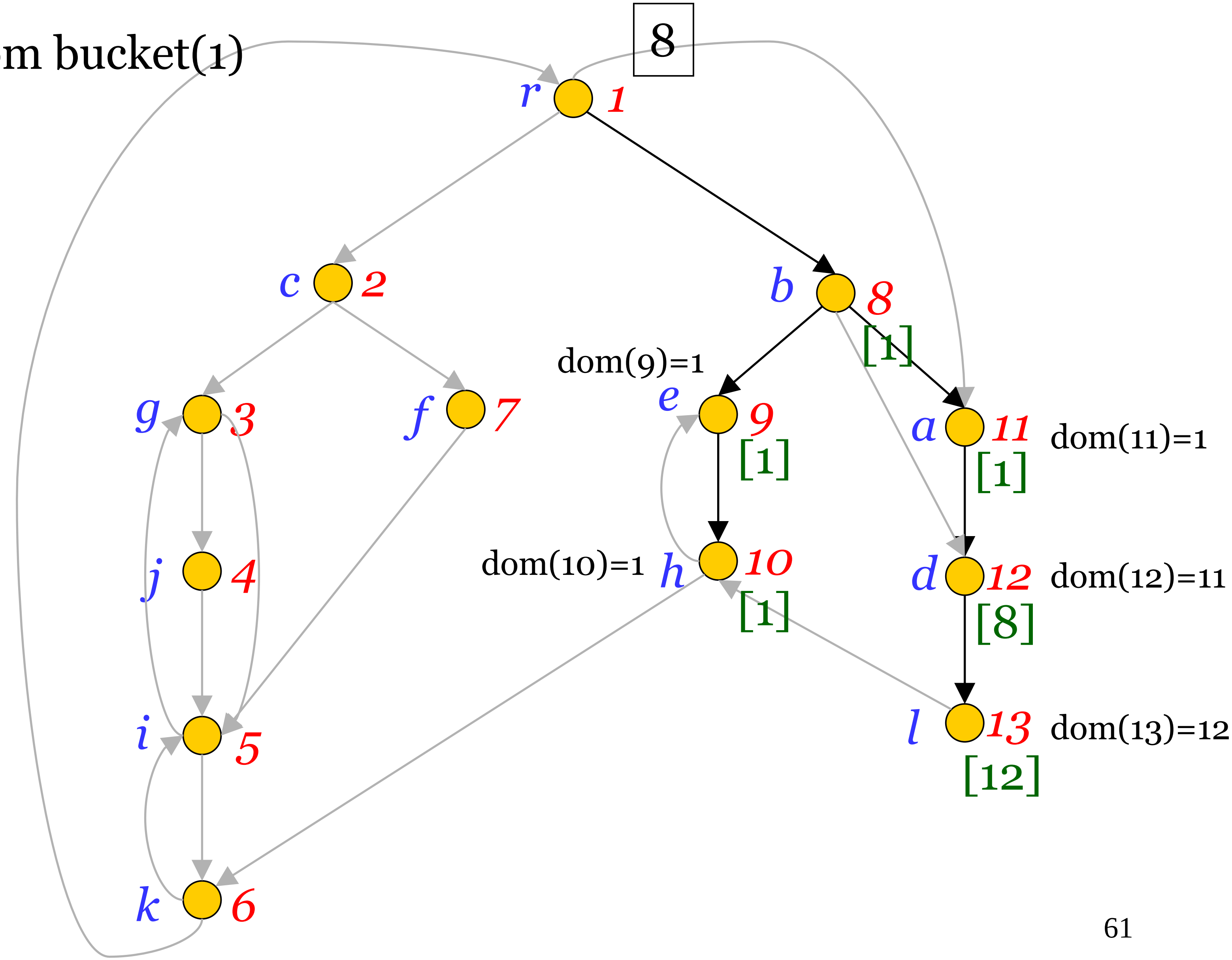
eval(10) = 10



The Lengauer-Tarjan Algorithm: Example

delete 9 from bucket(1)

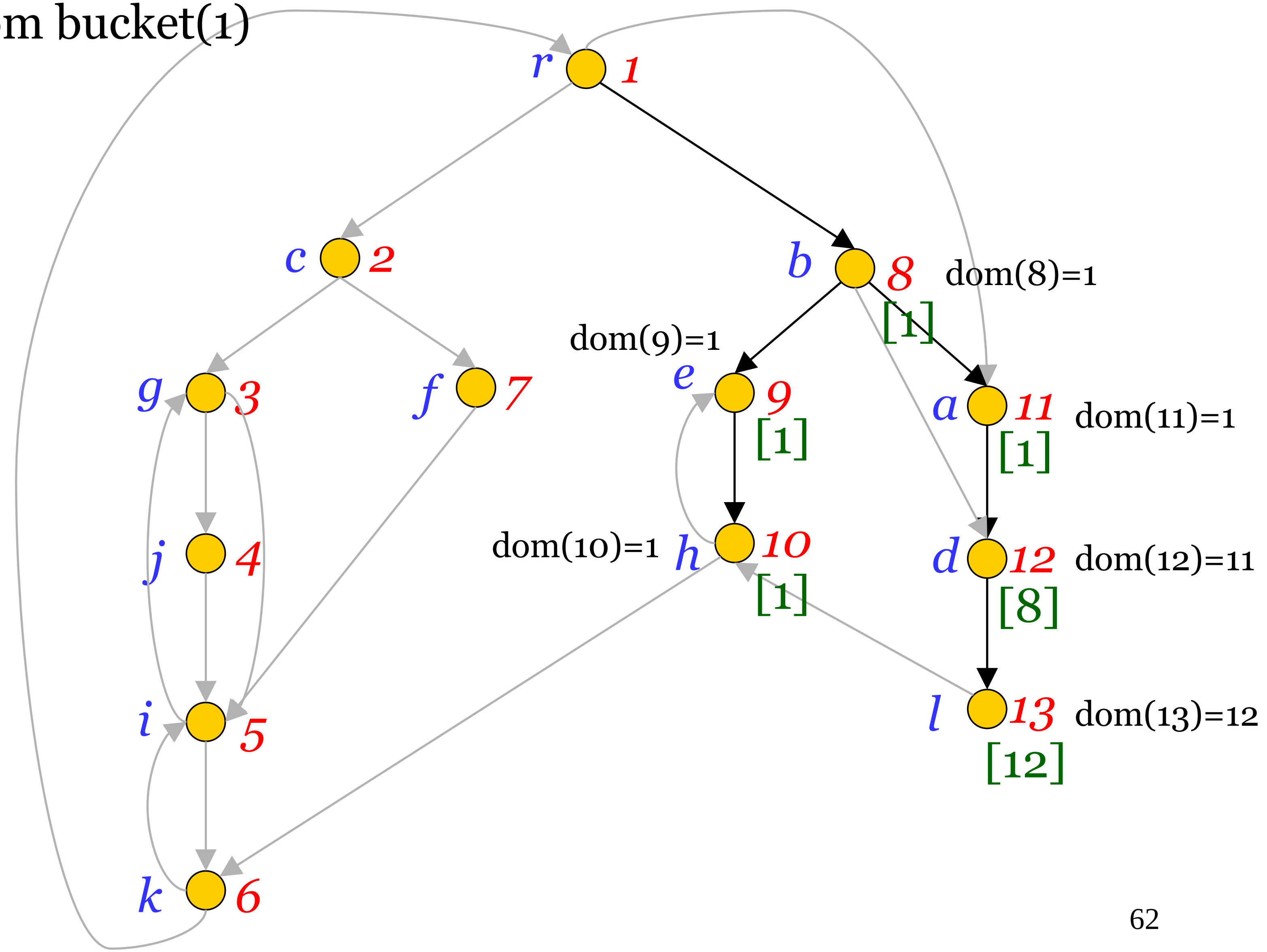
eval(9) = 9



The Lengauer-Tarjan Algorithm: Example

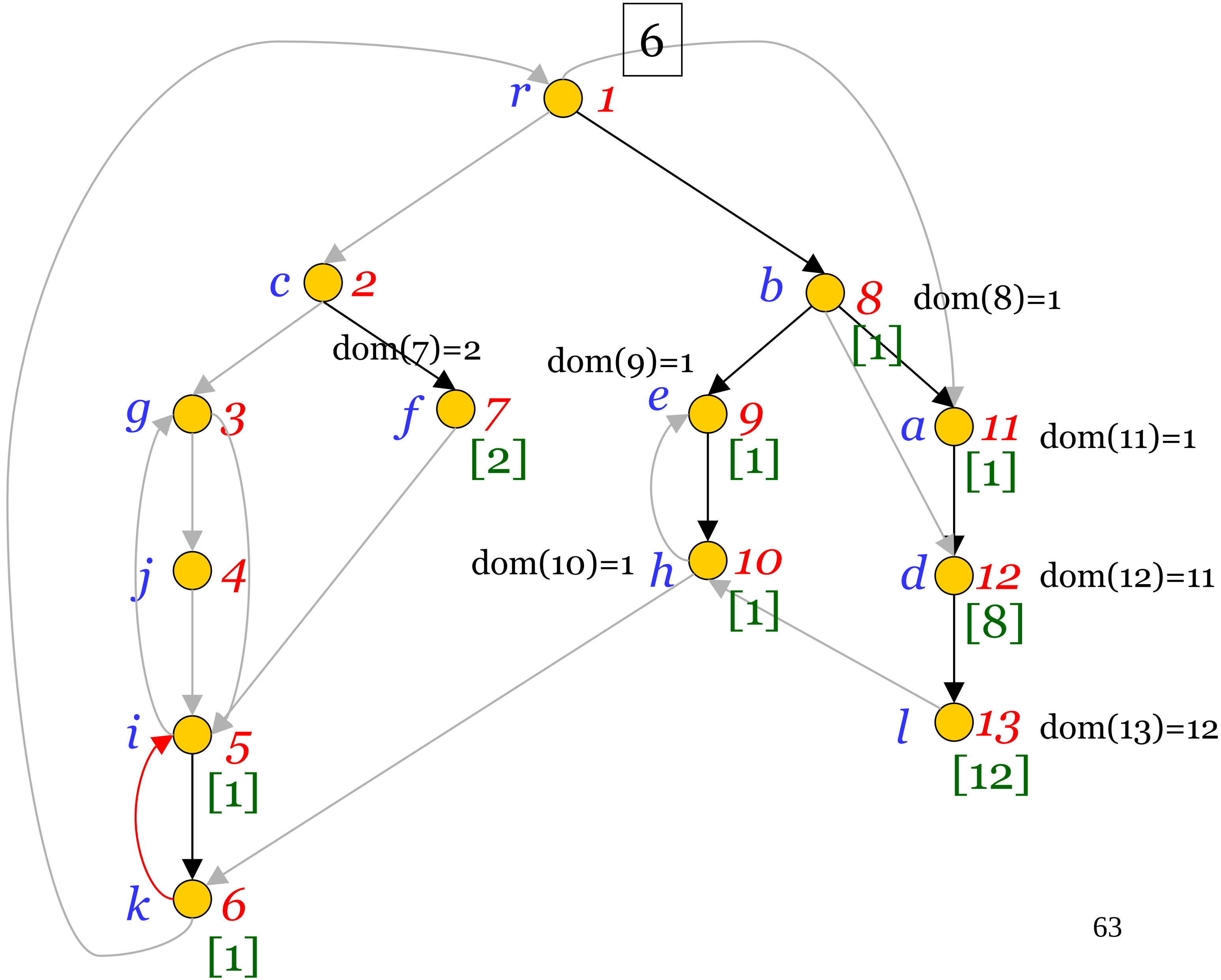
delete 8 from bucket(1)

eval(8) = 8



The Lengauer-Tarjan Algorithm: Example

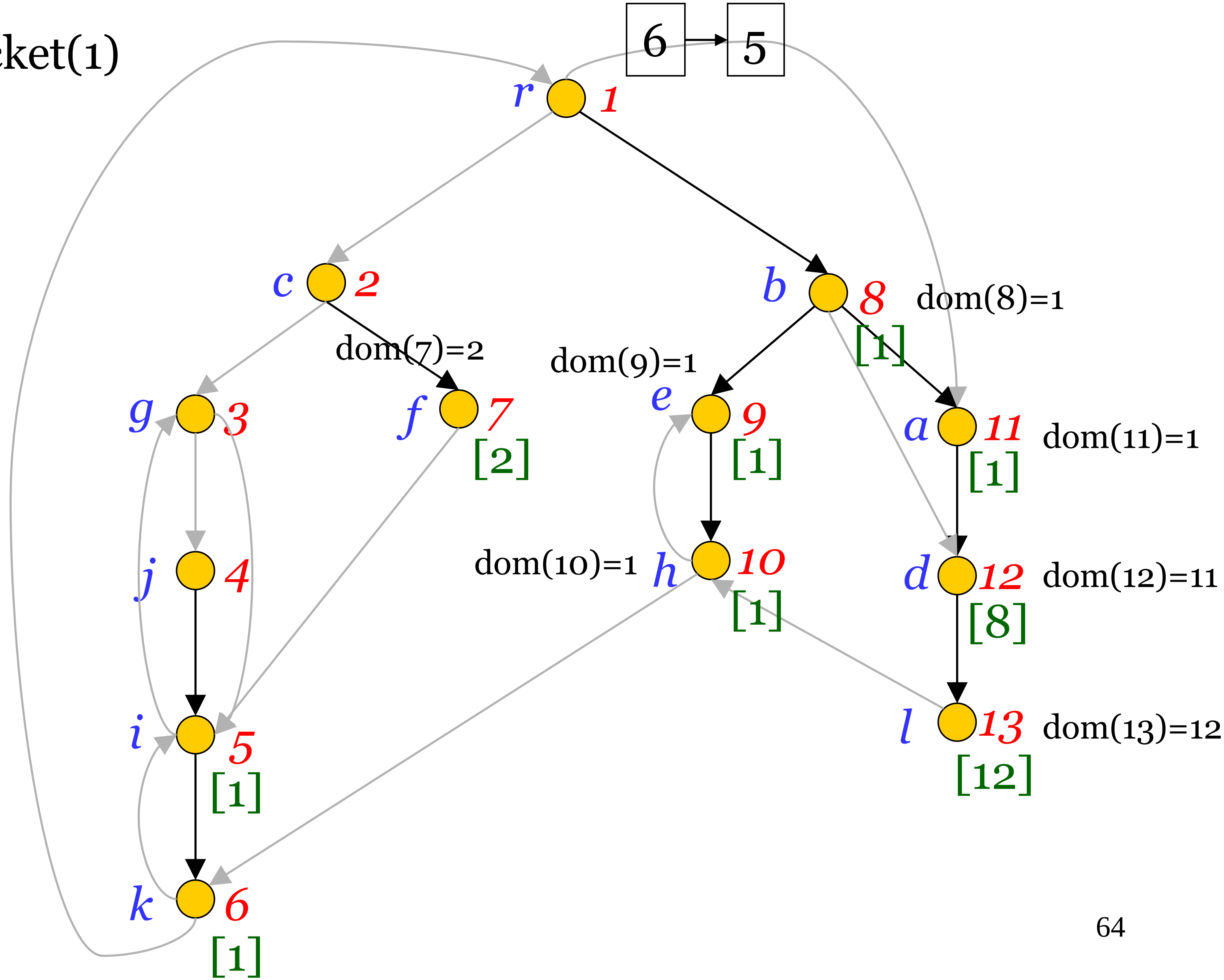
eval(6) = 6



The Lengauer-Tarjan Algorithm: Example

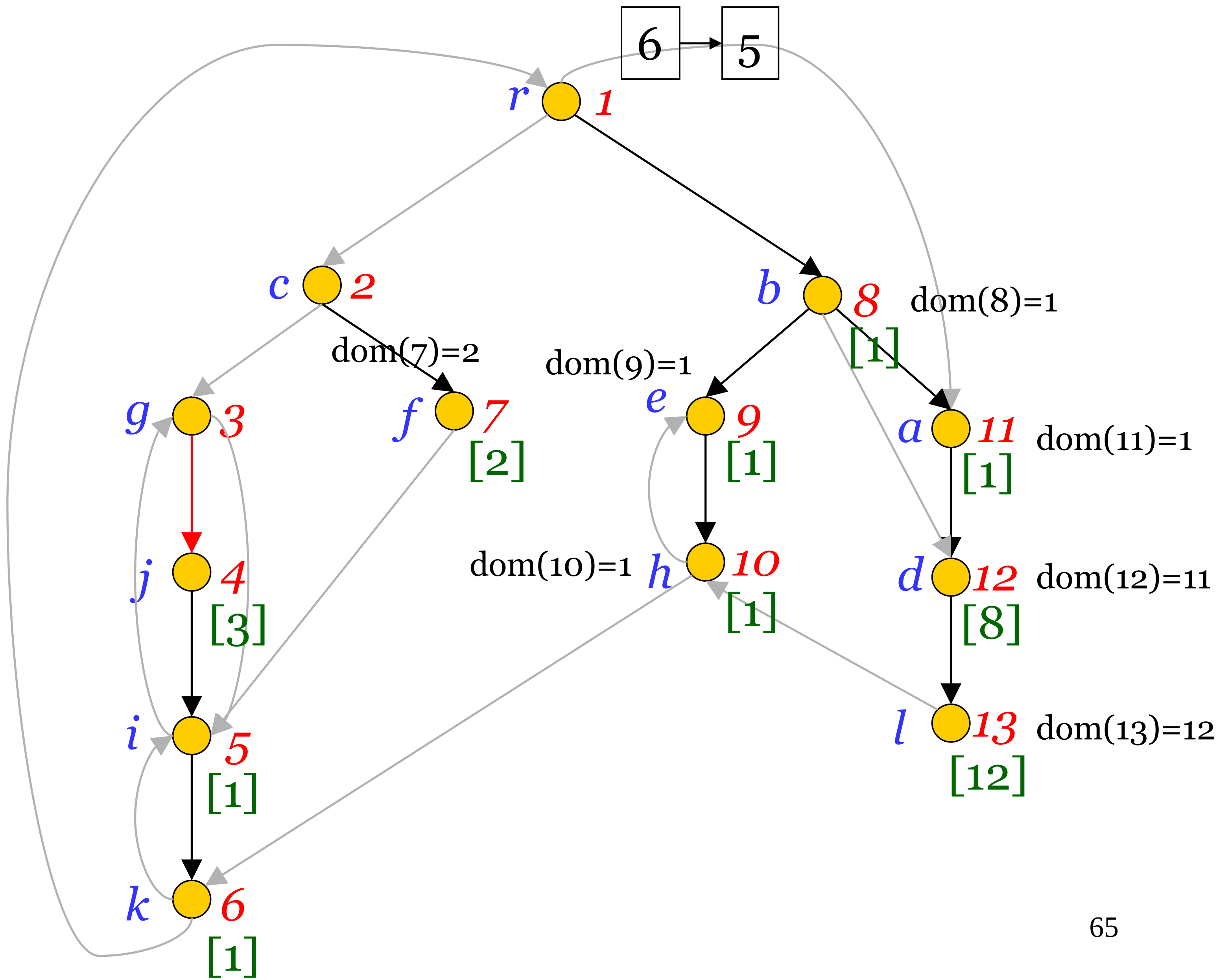
add 5 to bucket(1)

link(5)



The Lengauer-Tarjan Algorithm: Example

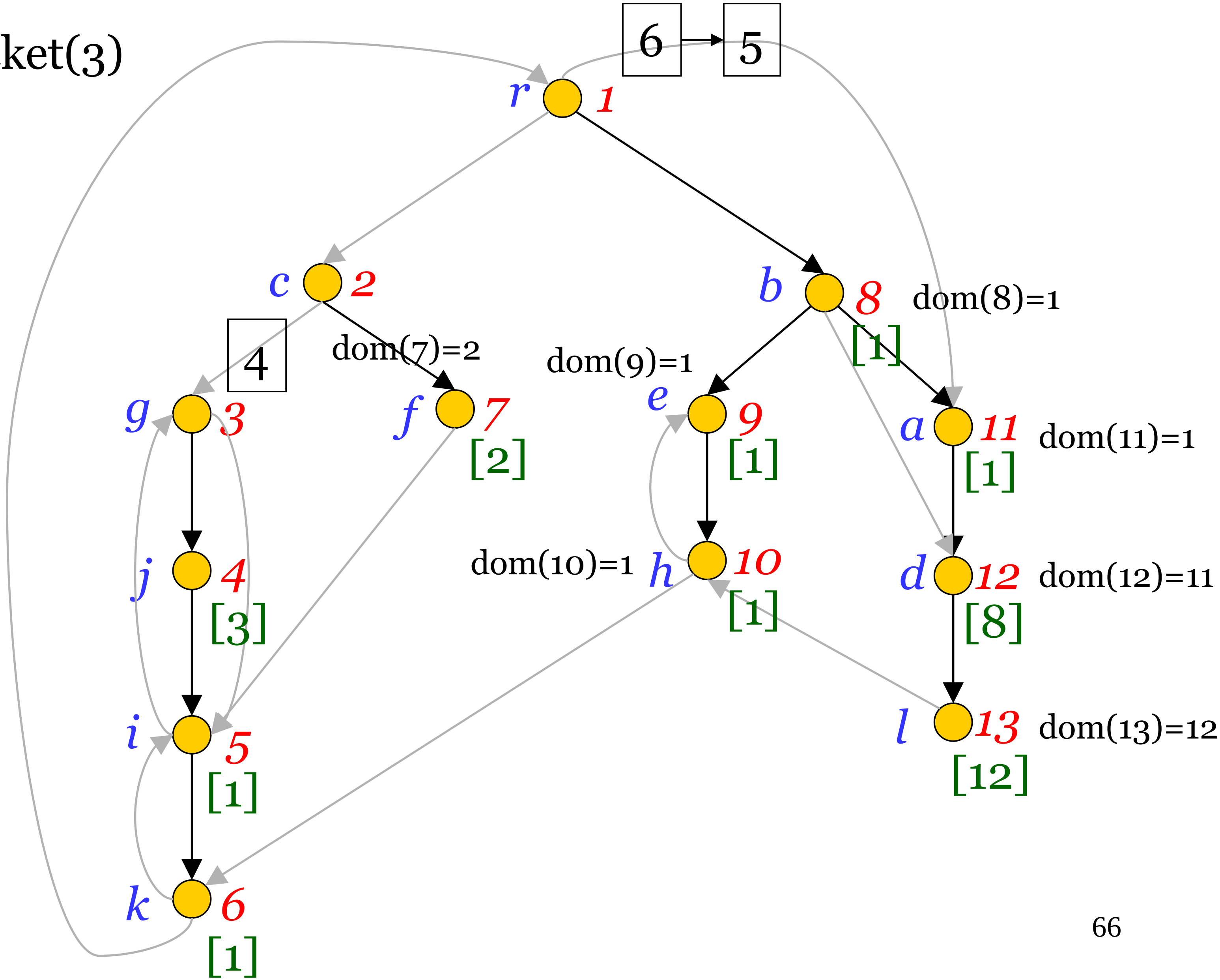
eval(3) = 3



The Lengauer-Tarjan Algorithm: Example

add 4 to bucket(3)

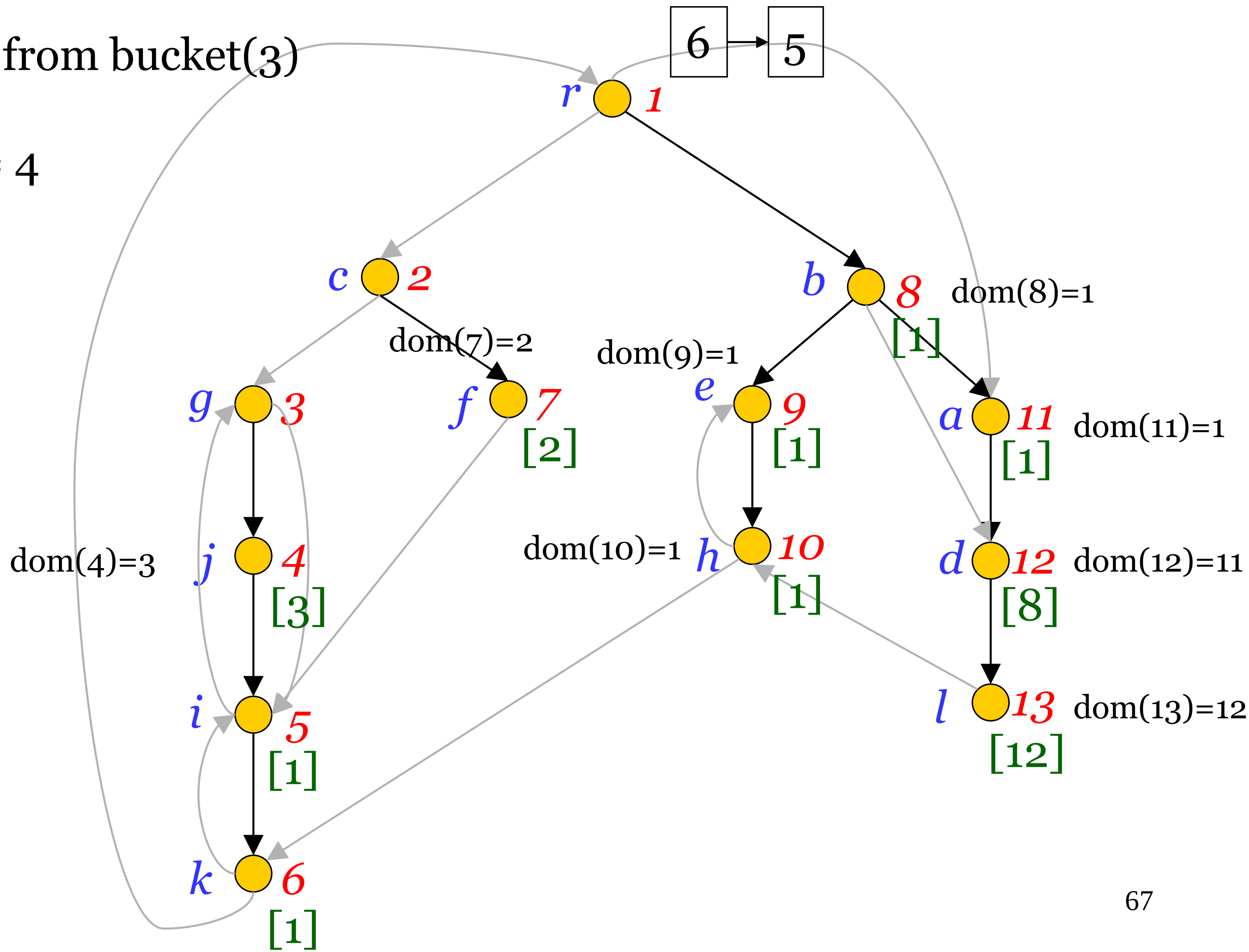
link(4)



The Lengauer-Tarjan Algorithm: Example

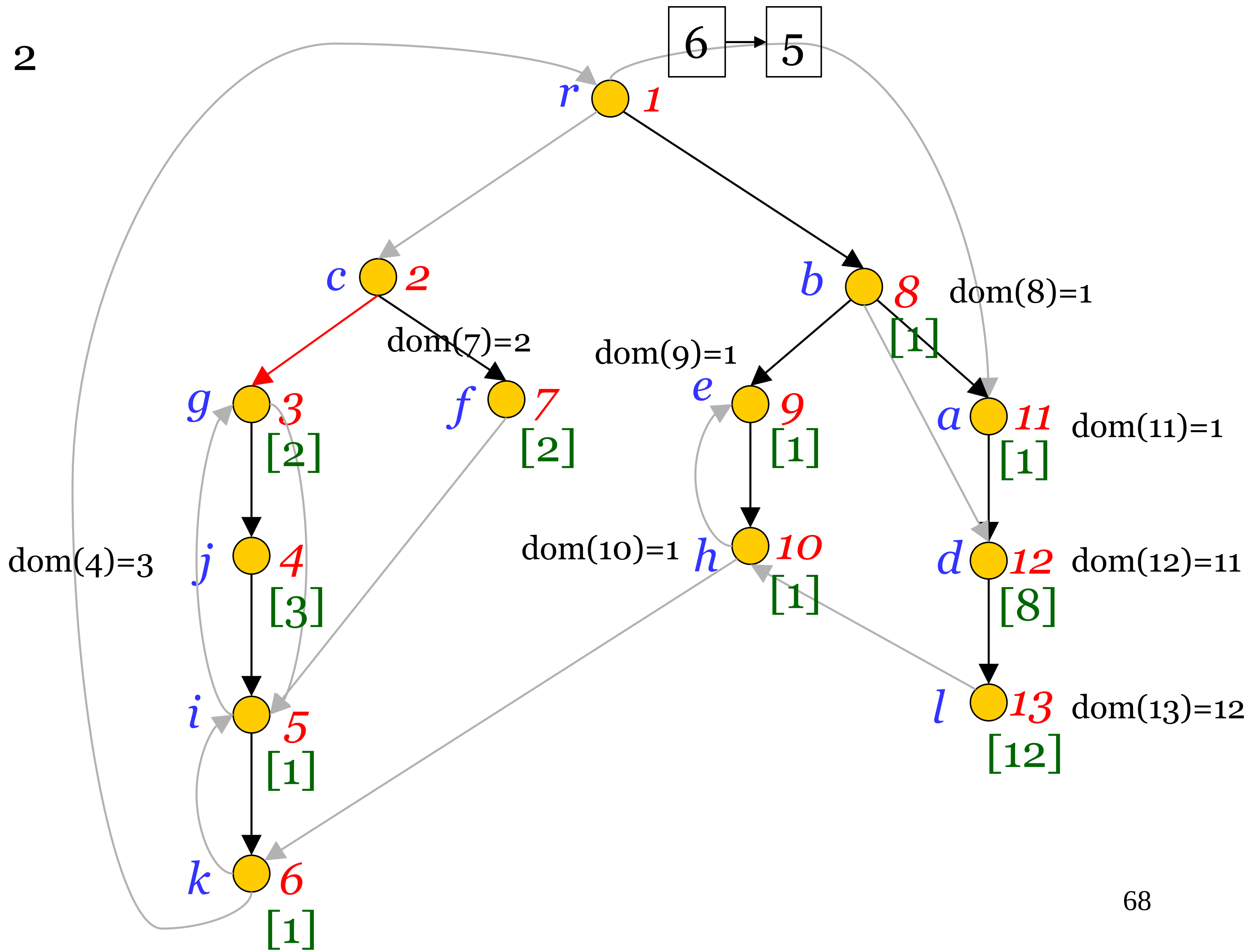
delete 4 from bucket(3)

eval(4) = 4



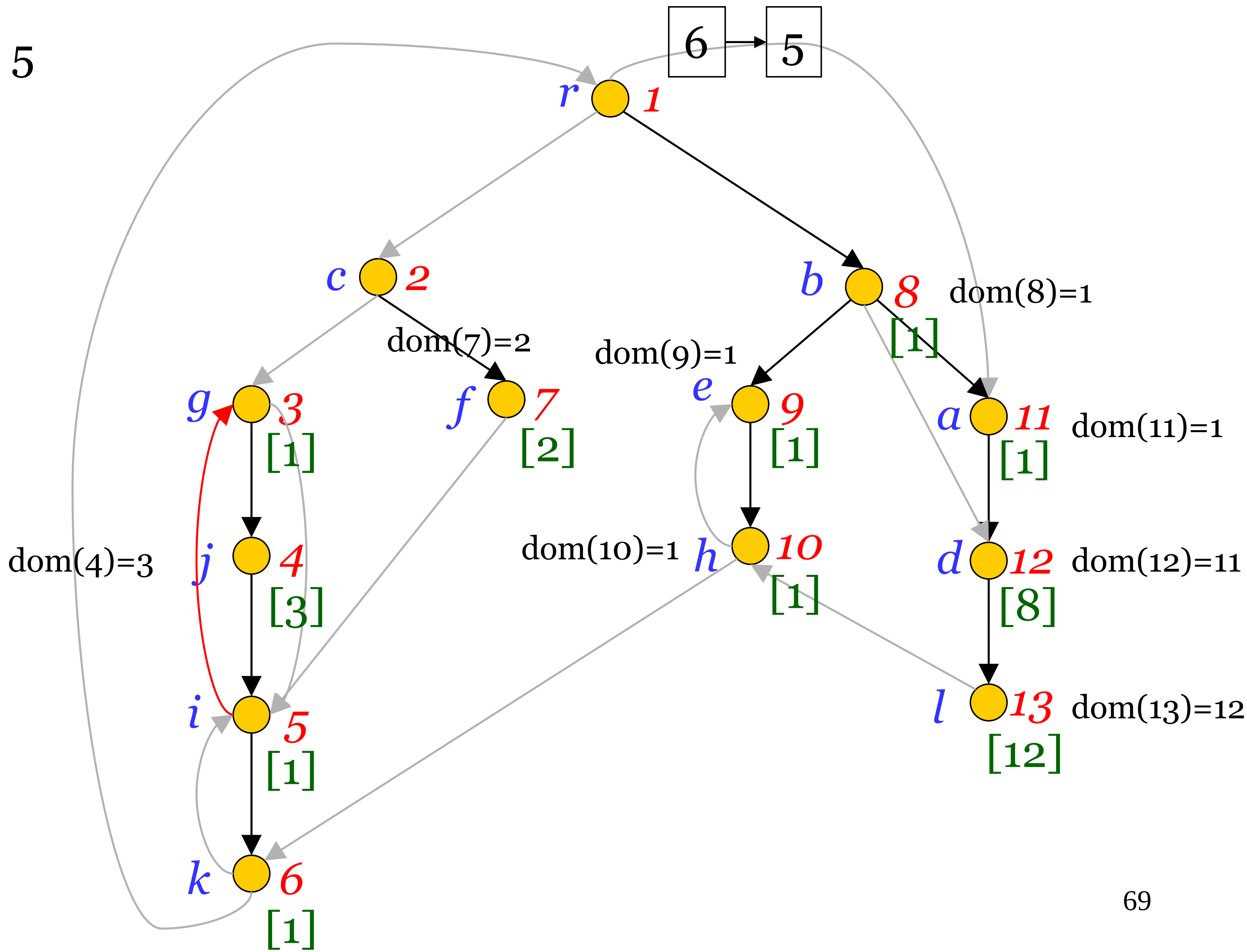
The Lengauer-Tarjan Algorithm: Example

eval(2) = 2



The Lengauer-Tarjan Algorithm: Example

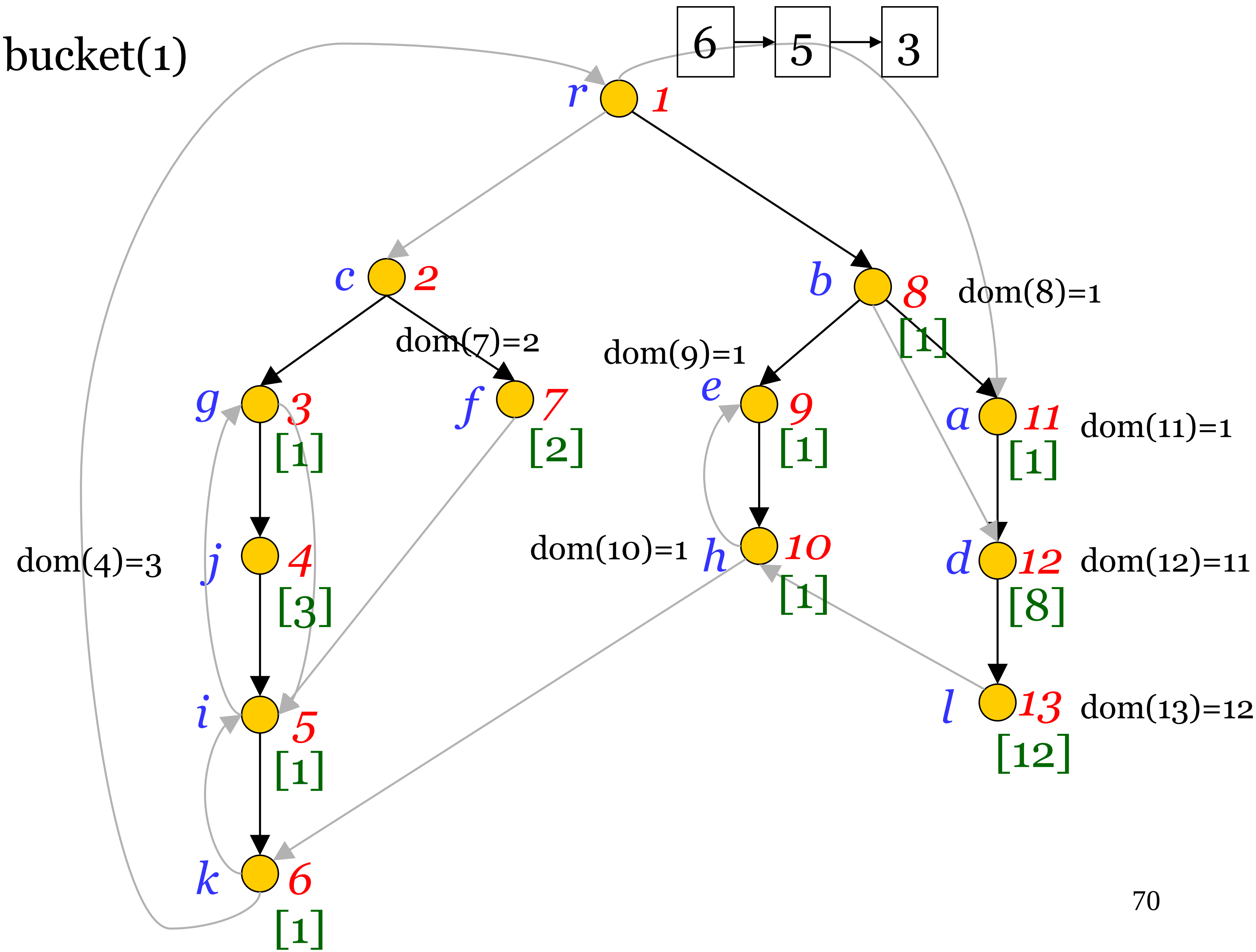
eval(5) = 5



The Lengauer-Tarjan Algorithm: Example

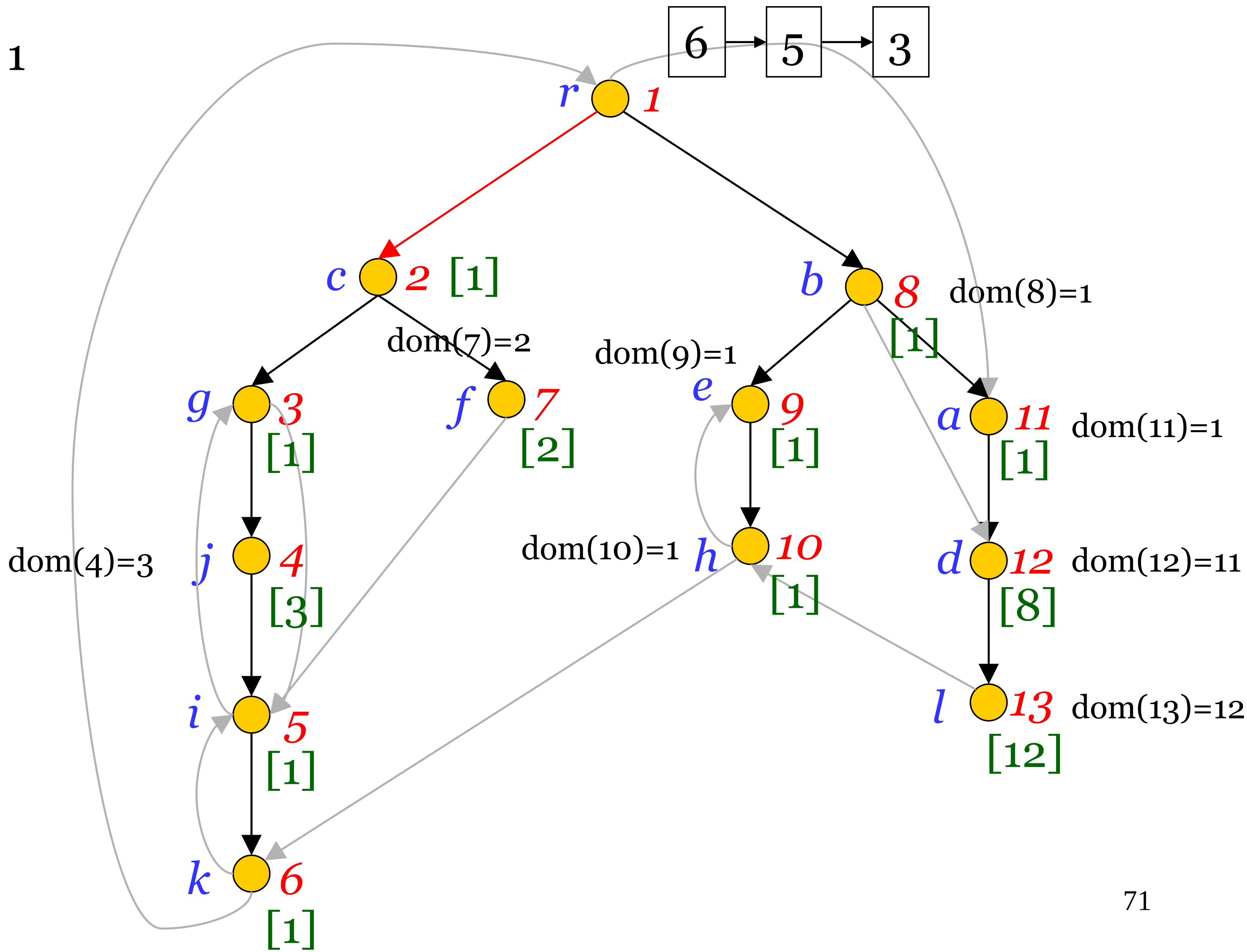
add 3 to bucket(1)

link(3)



The Lengauer-Tarjan Algorithm: Example

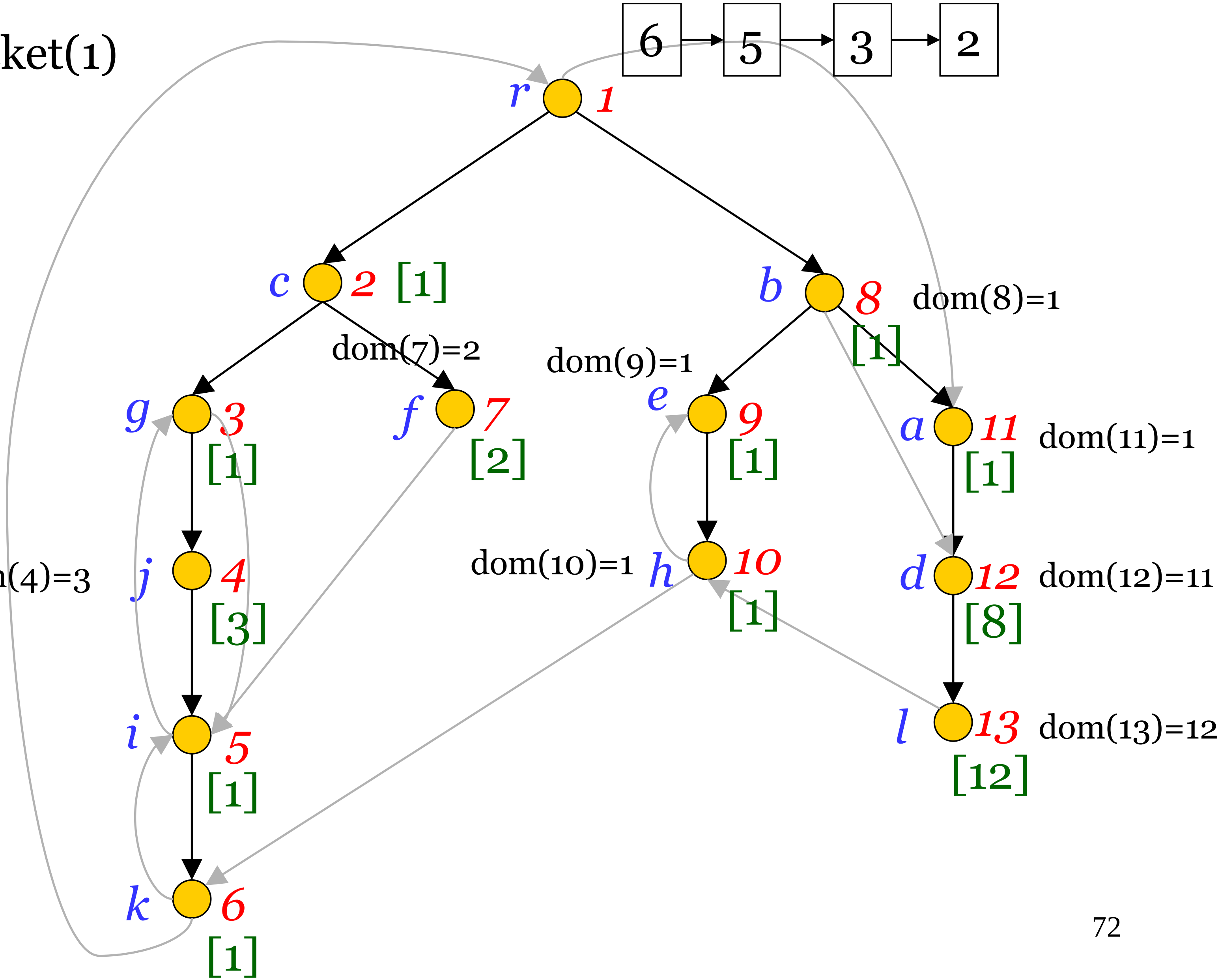
eval(1) = 1



The Lengauer-Tarjan Algorithm: Example

add 2 to bucket(1)

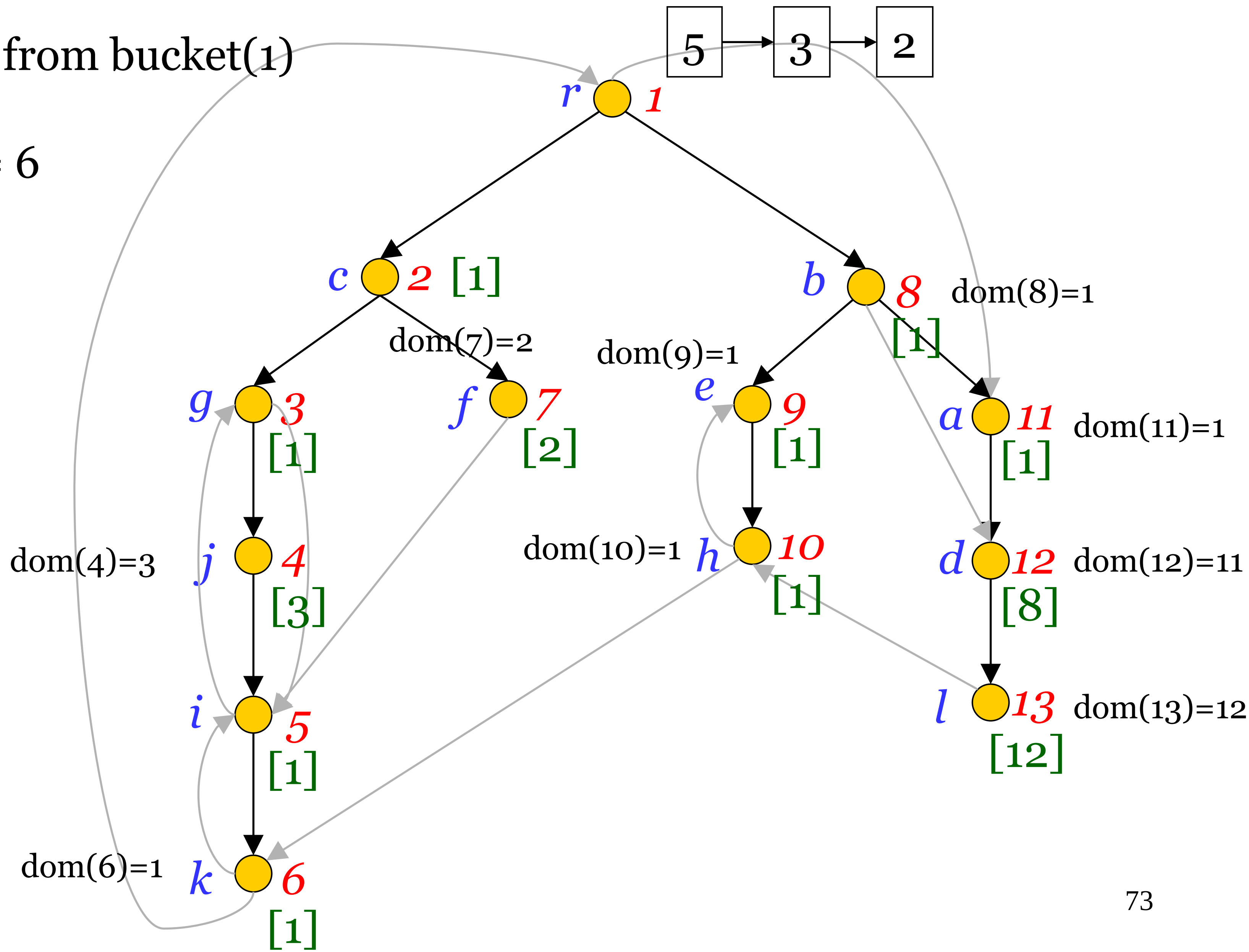
link(2)



The Lengauer-Tarjan Algorithm: Example

delete 6 from bucket(1)

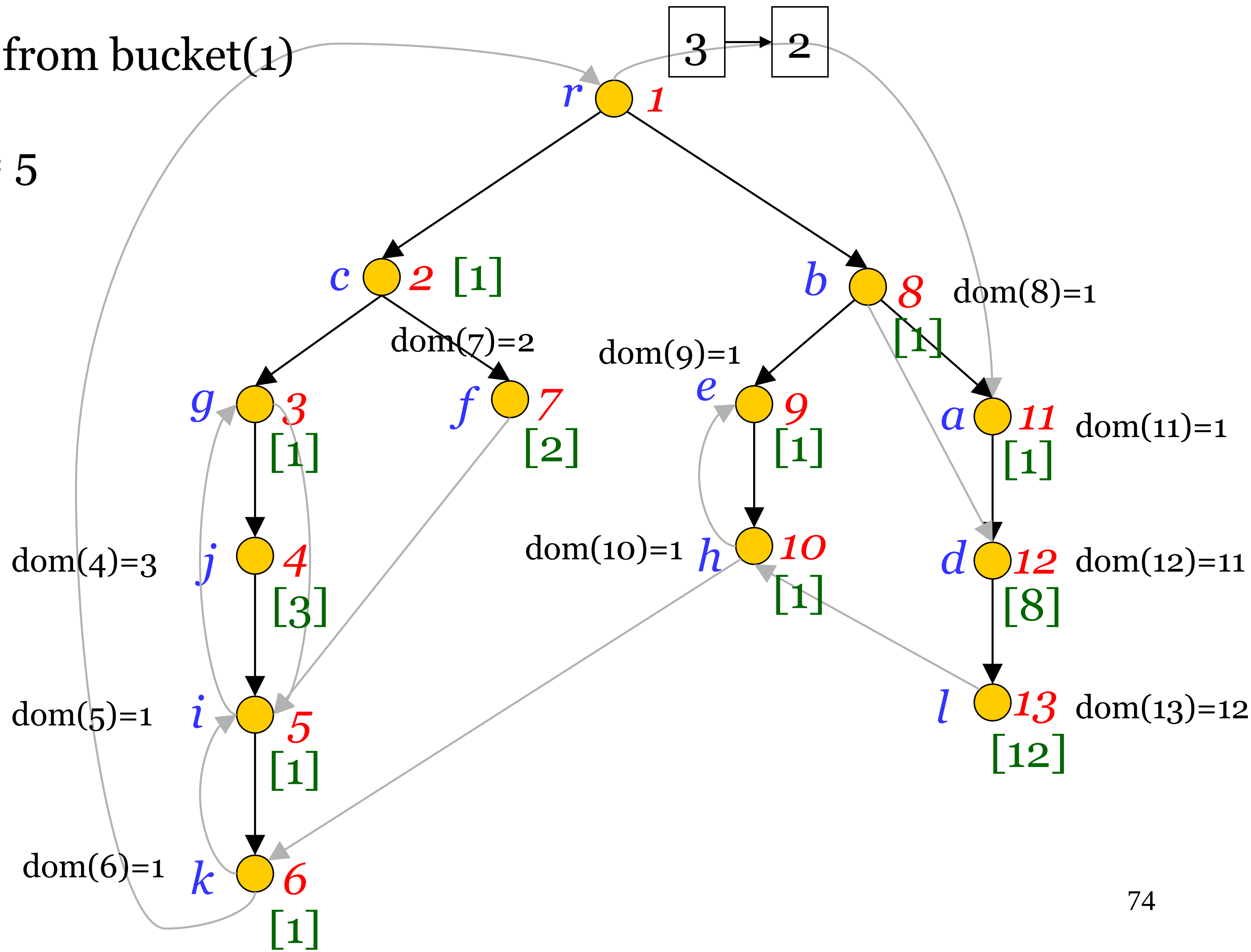
eval(6) = 6



The Lengauer-Tarjan Algorithm: Example

delete 5 from bucket(1)

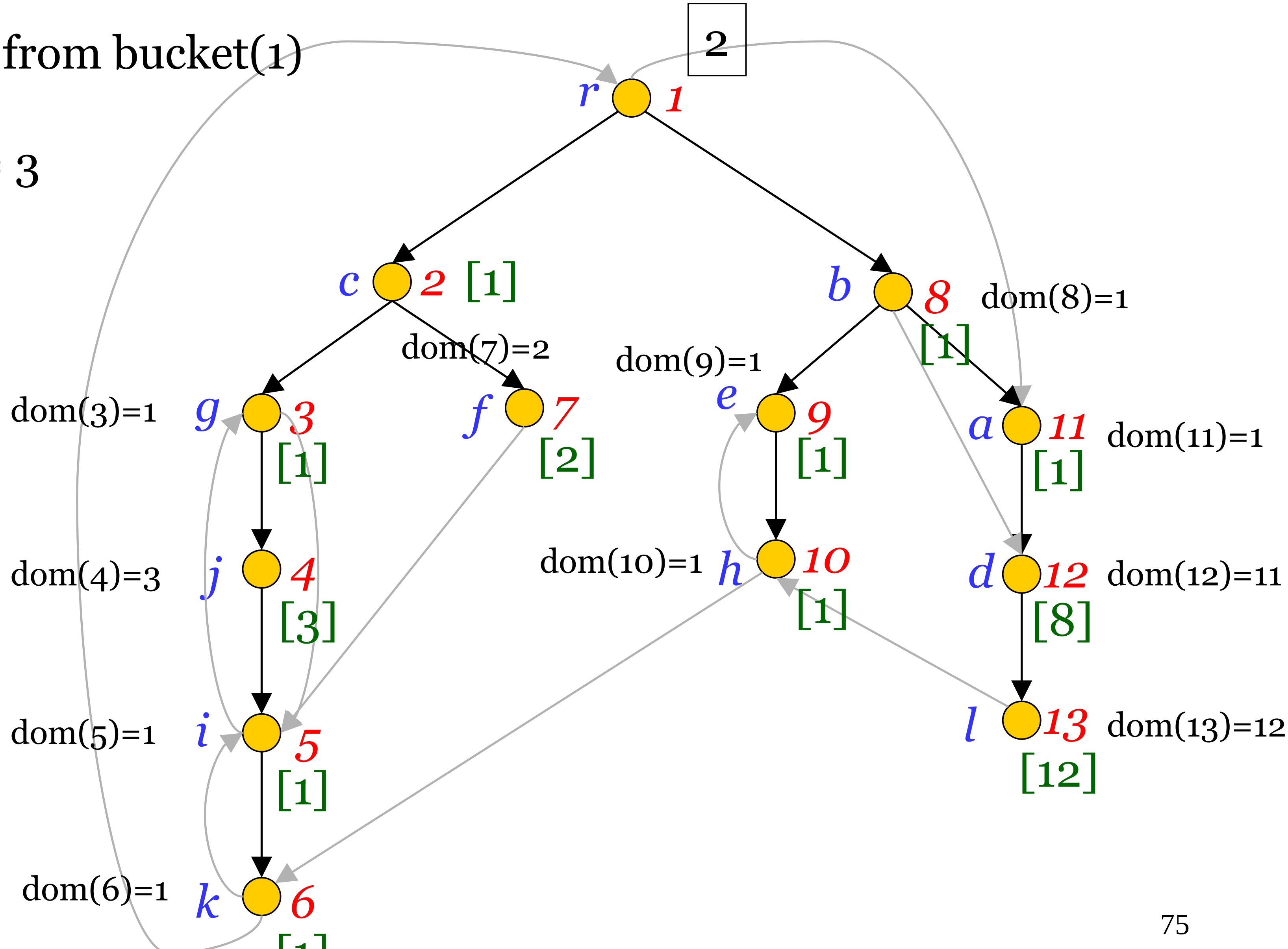
eval(5) = 5



The Lengauer-Tarjan Algorithm: Example

delete 3 from bucket(1)

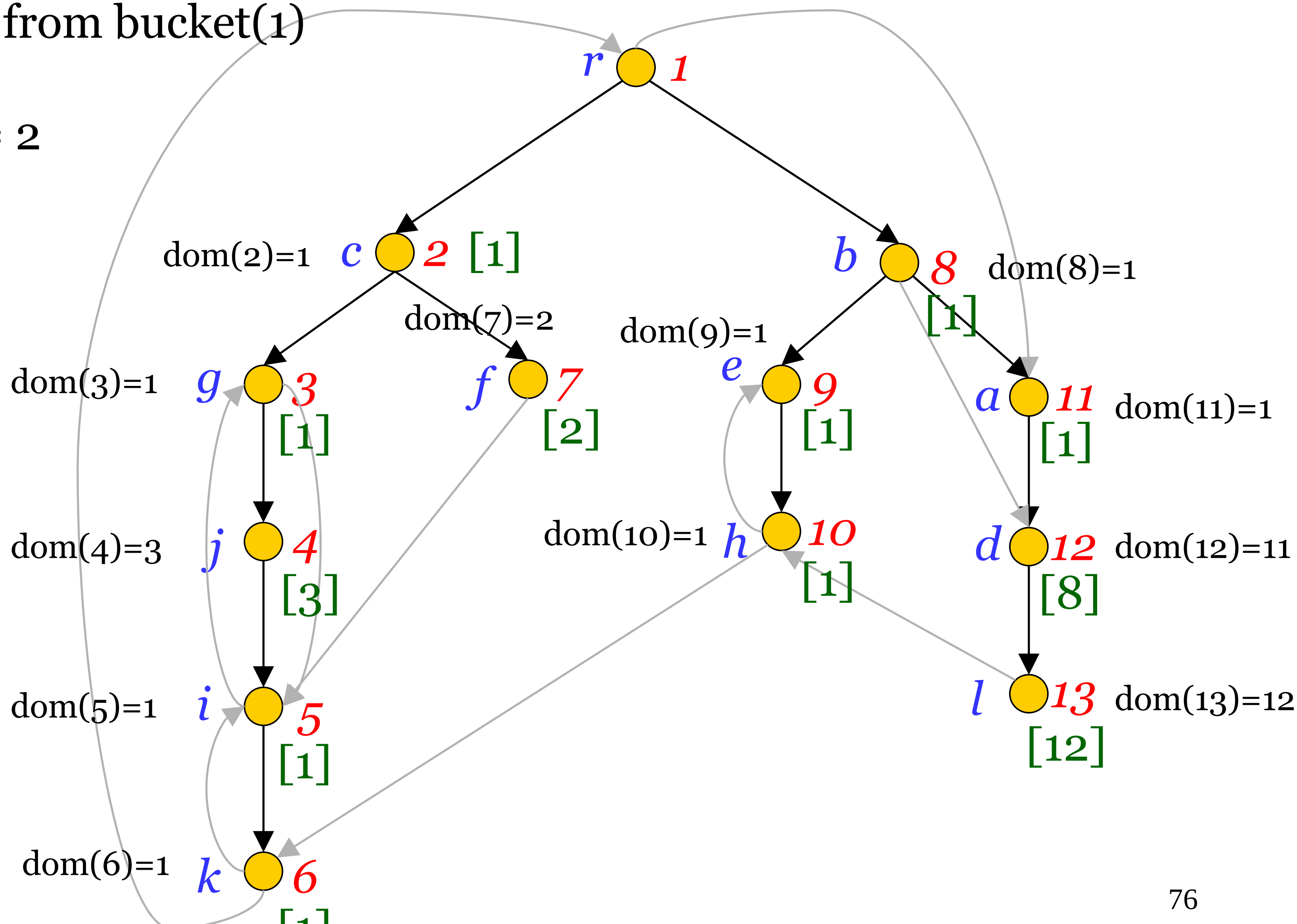
eval(3) = 3



The Lengauer-Tarjan Algorithm: Example

delete 2 from bucket(1)

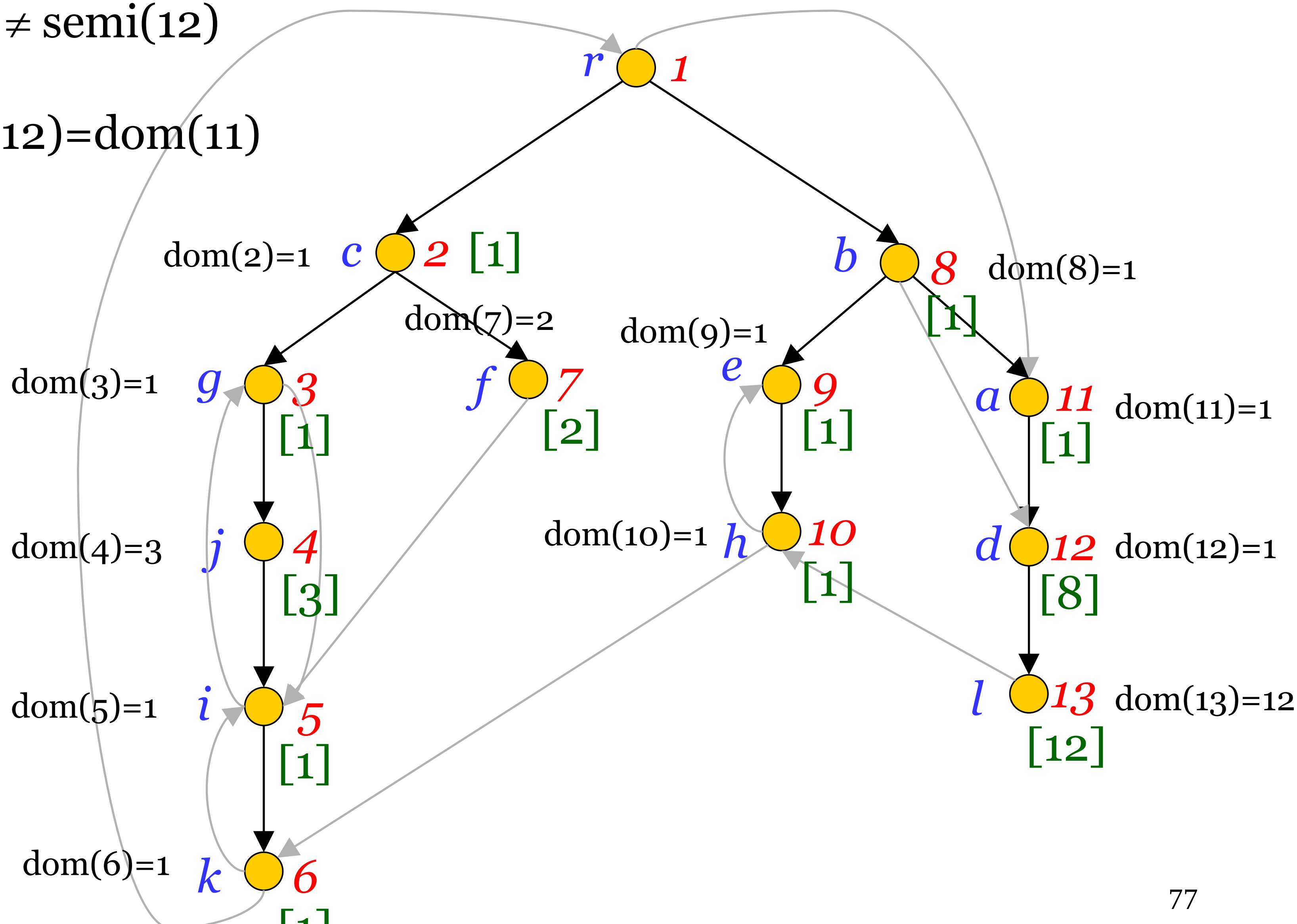
eval(2) = 2



The Lengauer-Tarjan Algorithm: Example

dom(12) ≠ semi(12)

set dom(12)=dom(11)



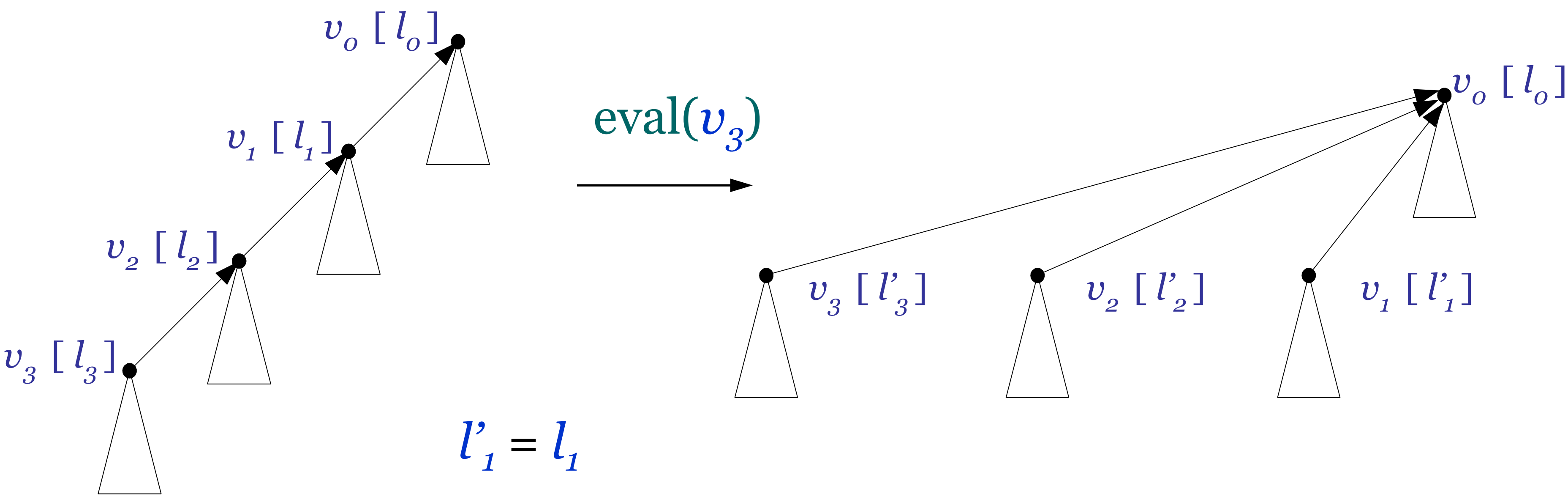
The Lengauer-Tarjan Algorithm

Running Time = $O(n + m)$ + Time for n-1 calls to link()

+ Time for m+n-1 calls to eval()

Data Structure for link() and eval()

We want to apply **Path Compression**:



$l'_1 = l_1$

$semi(l'_2) = \min \{ semi(l_1), semi(l_2) \}$

$semi(l'_3) = \min \{ semi(l_1), semi(l_2), semi(l_3) \}$

Data Structure for link() and eval()

We maintain a virtual forest VF such that:

1. For each T in F there is a corresponding VT in VF with the same vertices as T.

2. Corresponding trees T and VT have the same root with the same label.

3. If v is any vertex, eval(v, F) = eval(v, VF).

Representation: *ancestor(v)* = parent of v in VT.

Data Structure for link() and eval()

eval(v): Compress the path $r \overset{*}{\rightarrow} v$ and return the label of v.

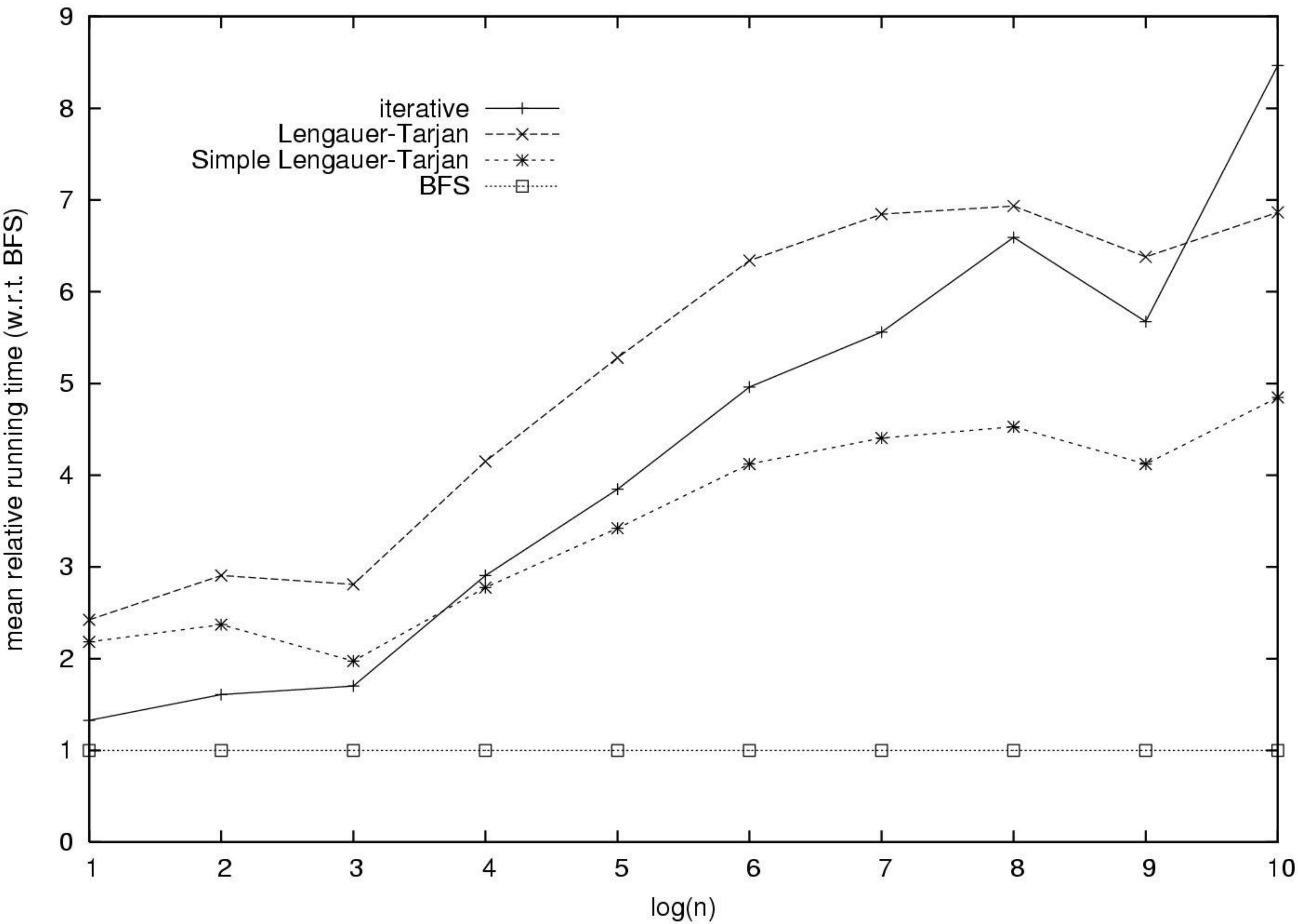
link(v,w): Make v the parent of w.

VF satisfies Properties 1-3.

Time for n-1 calls to link() +

Time for m+n-1 calls to eval() = $O(m \cdot \log_{2+\lfloor m/n \rfloor} n)$

Experimental Results (Small Graphs)



Experimental Results (Large Graphs)

GRAPH	n	m	density	height of dom. tree
NY	264346	733846	2.78	36
USA	23947347	58333344	2.44	241
ORACLE-16K	15678	48293	3.08	47
ORACLE-4M	4129899	14678595	3.55	1747
SAP-4M	4112973	12029328	2.92	6546
SAP-11M	11156904	36428546	3.27	4569
SAP-32M	32356252	81991806	2.53	7978
SAP-187M	186921678	556264024	2.98	180129504

GRAPH	SLT	LT	SNCA	IDFS	IBFS
NY	67.12	75.06	57.66	906.86	1301.80
USA	6544.00	7686.83	5868.11	393330.20	
ORACLE-16K	1.91	2.58	1.80	3.50	6.42
ORACLE-4M	1298.80	1511.77	1093.83	6827.96	327878.15
SAP-4M	1126.83	1445.78	965.85	1557961.15	1436782.57
SAP-11M	3035.53	3708.43	2610.60		
SAP-32M	7995.78	9843.50	6911.95		
SAP-187M	22422.59	28238.70	21281.76		

Running time in *ms*. Missing values correspond to execution time >1 h.

The Lengauer-Tarjan Algorithm: Correctness

Lemma 1: $\forall\, v, w$ such that $v \le w$, any path from v to w contains a common ancestor of v and w in T.

Follows from Property 1

Lemma 2: For any $w \neq r$, $idom(w)$ is an ancestor of w in T.

$idom(w)$ is contained in every path from r to w

The Lengauer-Tarjan Algorithm: Correctness

Lemma 3: For any $w \neq r$, $sdom(w)$ is an ancestor of w in T.

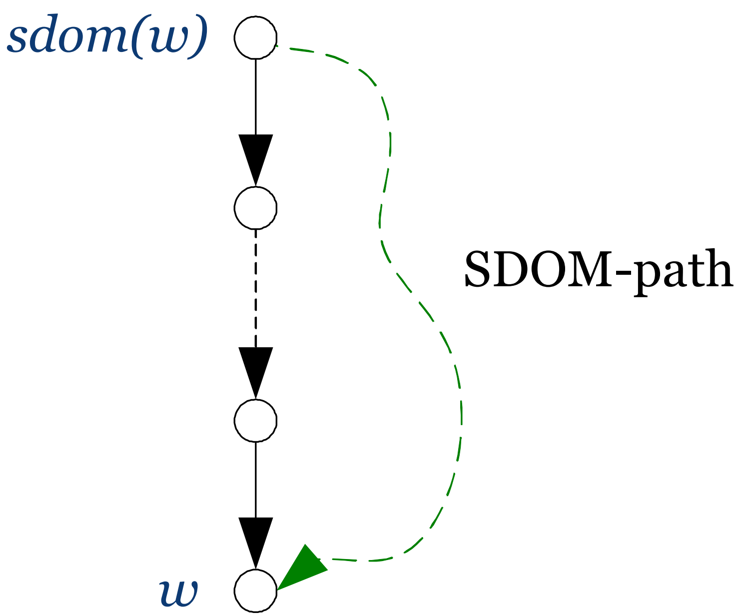
- $(parent(w), w)$ is an SDOM-path $\Rightarrow$ $sdom(w) \leq parent(w)$.
- SDOM-path $P = (v_0 = sdom(w), v_1, \ldots, v_k = w)$;

  Lemma 1 $\Rightarrow$ some v_i is a common ancestor of $sdom(w)$ and w.

  We must have $v_i \leq sdom(w) \Rightarrow v_i = sdom(w)$.

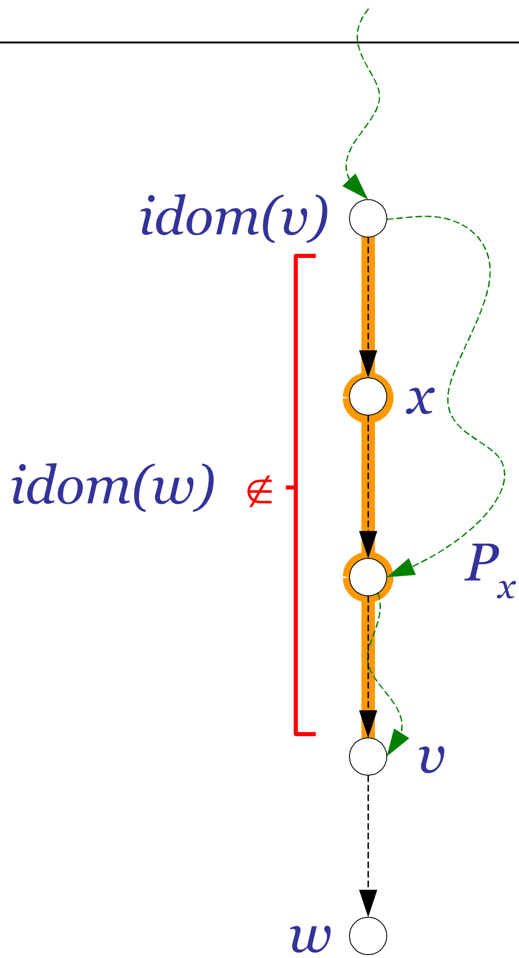
The Lengauer-Tarjan Algorithm: Correctness

Lemma 4: For any $w \neq r$, $idom(w)$ is an ancestor of $sdom(w)$ in T.



The SDOM-path from $sdom(w)$ to w avoids the proper ancestors of w that are proper descendants of $sdom(w)$.

Lemma 5: Let v, w satisfy $v \overset{*}{\rightarrow} w$. Then $v \overset{*}{\rightarrow} idom(w)$ or $idom(w) \overset{*}{\rightarrow} idom(v)$.



For each x that satisfies

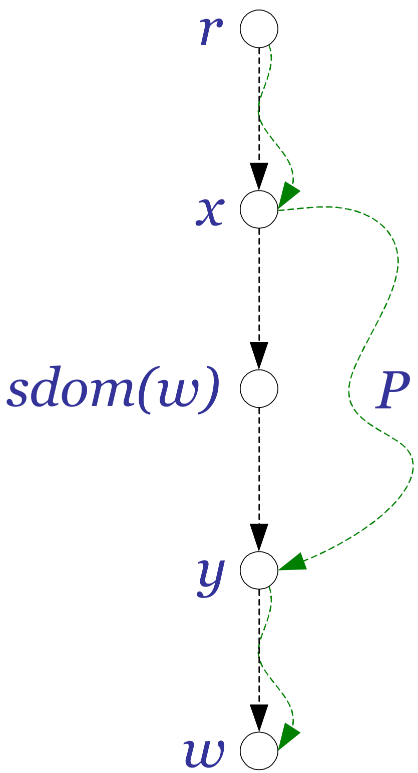
$$idom(v) \overset{+}{\rightarrow} x \overset{+}{\rightarrow} v$$

there is a path P_x from r to v that avoids x.

$P_x \cdot v \overset{*}{\rightarrow} w$ is a path from r to w that avoids $x \Rightarrow idom(w) \neq x$.

The Lengauer-Tarjan Algorithm: Correctness

Theorem 2: Let $w \neq r$. If $sdom(u) \geq sdom(w)$ for every u that satisfies $sdom(w) \xrightarrow{+} u \xrightarrow{*} w$ then $idom(w) = sdom(w)$.



Suppose for contradiction $sdom(w) \notin Dom(w)$

$\Rightarrow \exists$ path P from r to w that avoids $sdom(w)$.

x = last vertex $\in P$ such that $x < sdom(w)$

y = first vertex $\in P \cap sdom(w) \xrightarrow{*} w$

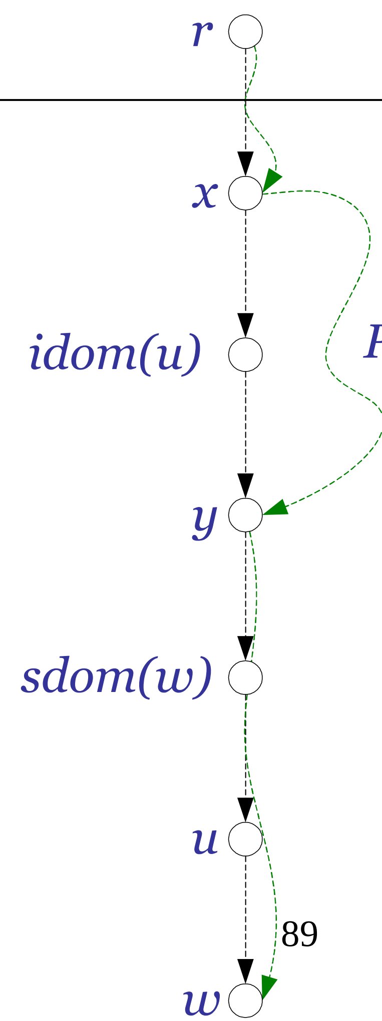
Q = part of P from x to y

Lemma 1 $\Rightarrow y < u, \forall u \in Q - \{x,y\}$

$\Rightarrow sdom(y) < sdom(w)$.

The Lengauer-Tarjan Algorithm: Correctness

Theorem 3: Let $w \neq r$ and let u be any vertex for which $sdom(u)$ is minimum among the vertices u that satisfy $sdom(w) \xrightarrow{+} u \xrightarrow{*} w$. Then $idom(u) = idom(w)$.



Lemma 4 and Lemma 5 $\Rightarrow idom(w) \xrightarrow{*} idom(u)$.

Suppose for contradiction $idom(u) \neq idom(w)$.

$\Rightarrow \exists$ path P from r to w that avoids $idom(u)$.

The Lengauer-Tarjan Algorithm: Correctness

Theorem 3: Let $w \neq r$ and let u be any vertex for which $sdom(u)$ is minimum among the vertices u that satisfy $sdom(w) \xrightarrow{+} u \xrightarrow{*} w$. Then $idom(u) = idom(w)$.

---

x = last vertex $\in P$ such that $x < idom(u)$.

y = first vertex $\in P \cap idom(u) \xrightarrow{*} w$.

Q = part of P from x to y.

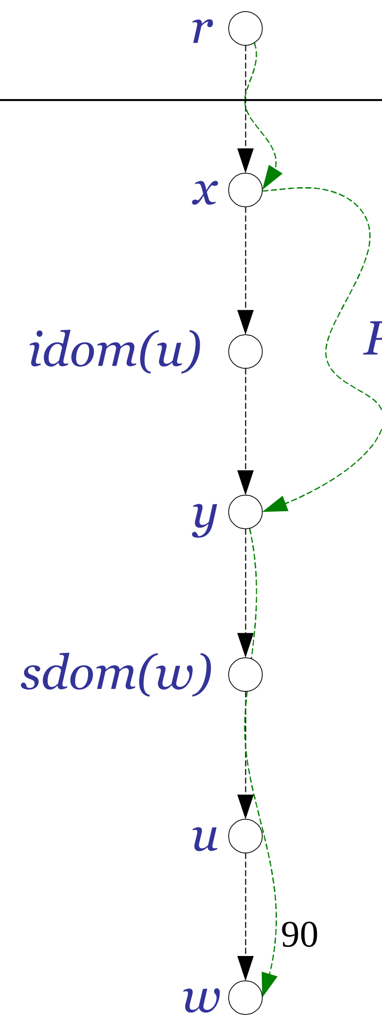
Lemma 1 $\Rightarrow$ $y < u$, $\forall\ u \in Q - \{x,y\}$

$\Rightarrow sdom(y) < idom(u) \leq sdom(u)$.

Therefore $y \neq v$ for any v that satisfies

$idom(u) \xrightarrow{+} v \xrightarrow{*} u$.

But y cannot be an ancestor of $idom(u)$.



The Lengauer-Tarjan Algorithm: Correctness

From Theorem 2 and Theorem 3 we have $sdom \Rightarrow idom$:

Corollary 1: Let $w \neq r$ and let u be any vertex for which $sdom(u)$ is minimum among the vertices u that satisfy $sdom(w) \overset{+}{\rightarrow} u \overset{*}{\rightarrow} w$. Then $idom(w) = sdom(w)$, if $sdom(w) = sdom(w)$ and $idom(w) = idom(u)$ otherwise.

We still need a method to compute $sdom$.

The Lengauer-Tarjan Algorithm: Correctness

Theorem 4: For any $w \neq r$,

$$sdom(w) = \min\big(\{\, v \mid (v, w) \in E \text{ and } v < w \,\} \cup \{\, sdom(u) \mid u > w \text{ and } \exists\, (v, w) \in E \text{ such that } u \overset{*}{\rightarrow} v \,\}\big).$$

Let $x = \min\big(\{\, v \mid (v, w) \in E \text{ and } v < w \,\} \cup \{\, sdom(u) \mid u > w \text{ and } \exists\, (v, w) \in E \text{ such that } u \overset{*}{\rightarrow} v \,\}\big)$.

We first show $sdom(w) \leq x$ and then $sdom(w) \geq x$.

The Lengauer-Tarjan Algorithm: Correctness

Theorem 4: For any $w \neq r$,

$$sdom(w) = \min\left(\{ v \mid (v, w) \in E \text{ and } v < w \} \cup \{ sdom(u) \mid u > w \text{ and } \exists\, (v, w) \in E \text{ such that } u \overset{*}{\rightarrow} v \} \right).$$

- $sdom(w) \leq x$

Assume $x = v$ such that $(v, w) \in E$ and $v < w \Rightarrow sdom(w) \leq x$.

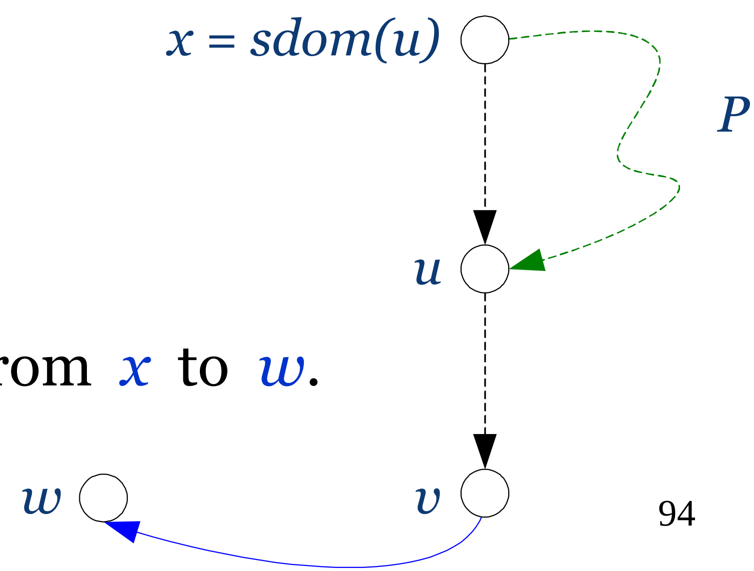
The Lengauer-Tarjan Algorithm: Correctness

Theorem 4: For any $w \neq r$,

$$sdom(w) = \min\left(\{ v \mid (v, w) \in E \text{ and } v < w \} \cup \{ sdom(u) \mid u > w \text{ and } \exists\, (v, w) \in E \text{ such that } u \overset{*}{\to} v \} \right).$$

- $sdom(w) \leq x$

Assume $x = sdom(u)$ such that $u > w$ and $(v, w) \in E$ for some descendant v of u in T.



P = SDOM-path from x to u $\Rightarrow$

$P \cdot u \overset{*}{\to} v \cdot (v, w)$ is an SDOM-path from x to w.

The Lengauer-Tarjan Algorithm: Correctness

Theorem 4: For any $w \neq r$,

$$sdom(w) = \min\left(\{ v \mid (v, w) \in E \text{ and } v < w \} \cup \{ sdom(u) \mid u > w \text{ and } \exists\, (v, w) \in E \text{ such that } u \overset{*}{\to} v \} \right).$$

- $sdom(w) \geq x$

Assume that $(sdom(w), w) \in E \Rightarrow sdom(w) \geq x$.

The Lengauer-Tarjan Algorithm: Correctness

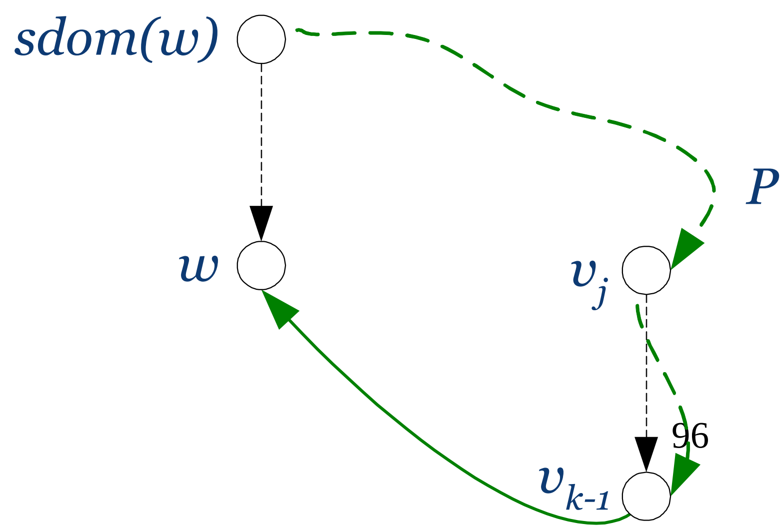
Theorem 4: For any $w \neq r$,

$$sdom(w) = \min (\{ v \mid (v, w) \in E \text{ and } v < w \} \cup \{ sdom(u) \mid u > w \text{ and } \exists (v, w) \in E \text{ such that } u \overset{*}{\rightarrow} v \}).$$

- $sdom(w) \geq x$

Assume that $P = (sdom(w) = v_0, v_1, \ldots, v_k = w)$ is a simple path

$v_i > w,\ 1 \leq i \leq k\text{-}1.$



$j = \min \{ i \geq 1 \mid v_i \overset{*}{\rightarrow} v_{k-1} \}.$

Lemma 1 $\Rightarrow\ v_i > v_j,\ 1 \leq i \leq j\text{-}1$

$\Rightarrow\ x \leq sdom(v_j) \leq sdom(w).$

The Lengauer-Tarjan Algorithm: Almost-Linear-Time Version

We get better running time if the trees in F are balanced (as in Set-Union).

F is balanced for constants $a > 1$, $c > 0$ if for all i we have:

$$\text{\# vertices in } F \text{ of height } i \leq cn/a^i$$

Theorem 5 [Tarjan 1975]: The total length of an arbitrary sequence of m path compressions in an n-vertex forest balanced for a, c is $O((m+n) \cdot \alpha(m+n, n))$, where the constant depends on a and c.

Linear-Time Algorithms

There are linear-time dominators algorithms both for the RAM Model and the Pointer-Machine Model.

- Based on LT, but much more complicated.
- First published algorithms that claimed linear-time, in fact didn't achieve that bound.

RAM: Harel [1985]

→ Alstrup, Harel, Lauridsen and Thorup [1999]

Pointer-Machine: Buchsbaum, Kaplan, Rogers and Westbrook [1998]

→ G. and Tarjan [2004],

→ Buchsbaum, G., Kaplan, Rogers, Tarjan and Westbrook [2008]

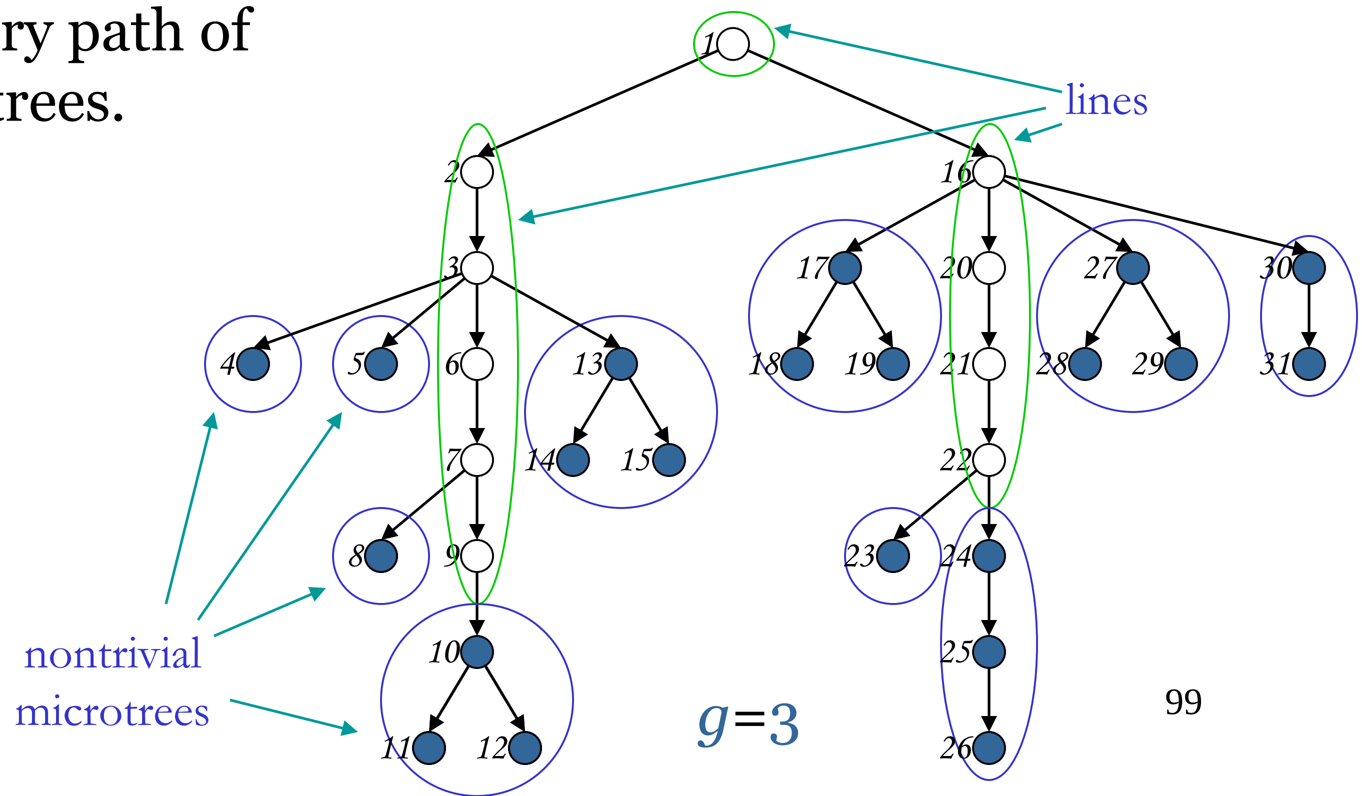
GT Linear-Time Algorithm: High-Level View

Partition DFS-tree D into nontrivial microtrees and lines.

Nontrivial microtree: Maximal subtree of D of size $\leq g$ that contains at least one leaf of D.

Trivial microtree: Single internal vertex of D.

Line: Maximal unary path of trivial microtrees.



GT Linear-Time Algorithm: High-Level View

Partition DFS-tree D into nontrivial microtrees and lines.

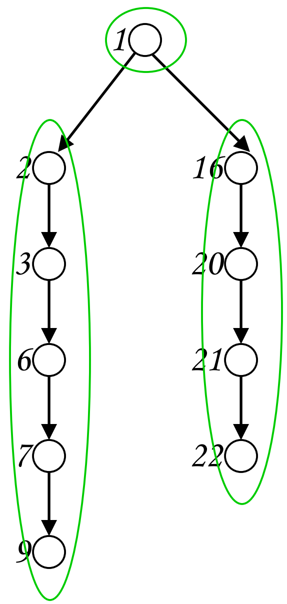
Nontrivial microtree: Maximal subtree of D of size $\leq g$ that contains at least one leaf of D.

Trivial microtree: Single internal vertex of D.

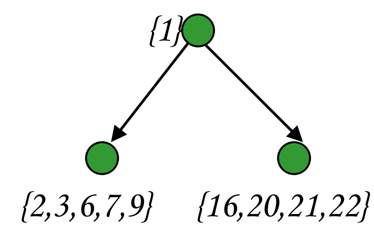
Line: Maximal unary path of trivial microtrees.

Core C: Tree D – nontrivial microtrees

C' : contract each line of C to a sigle vertex



C



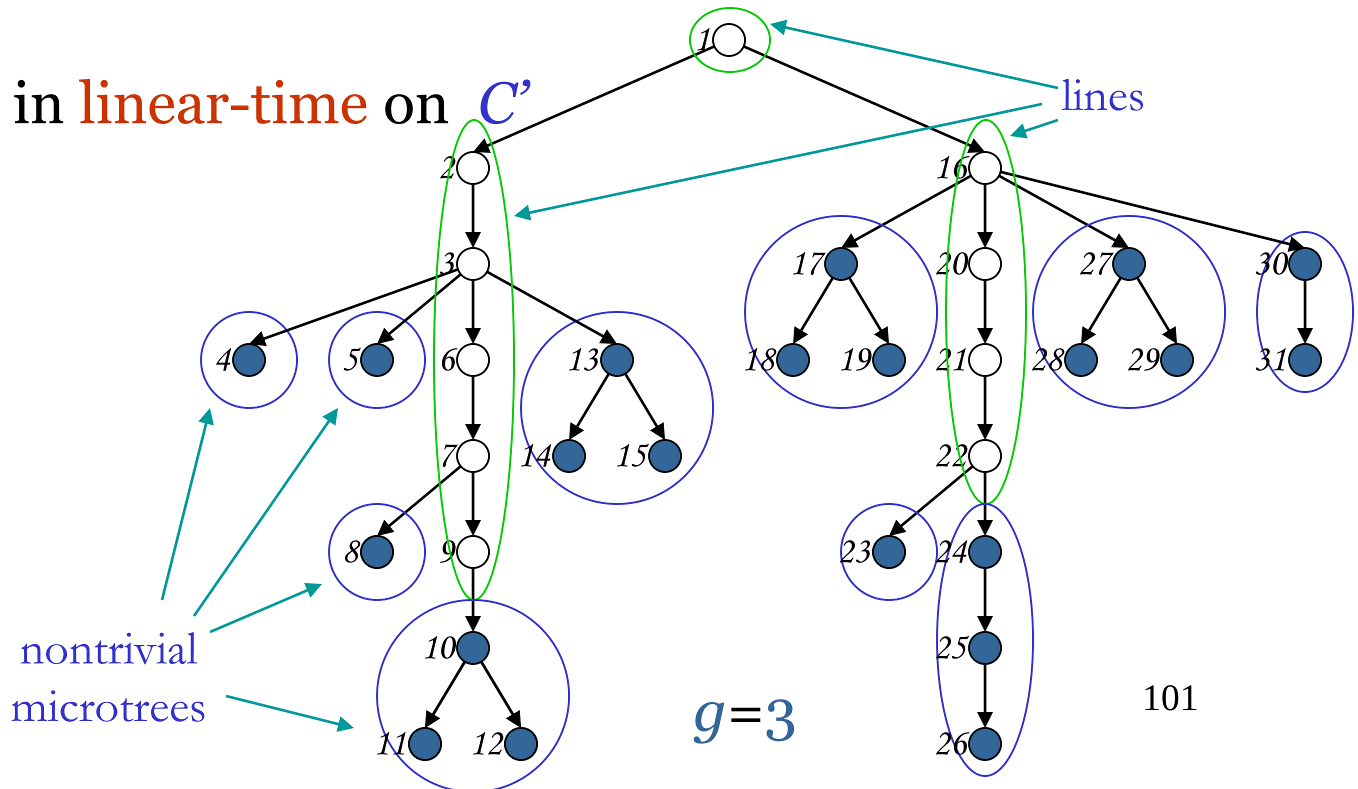
C'

GT Linear-Time Algorithm: High-Level View

Basic Idea: Compute external dominators in each nontrivial microtree and semidominators in each line, by running LT on C'

Precompute internal dominators in non-identical nontrivial microtrees.

Remark: LT runs in linear-time on C'



The Lengauer-Tarjan Algorithm: Almost-Linear-Time Version

Back to the $O(n \cdot \alpha(m,n))$-time version of LT…

We give the details of a data structure that achieves asymptotically faster link() and eval()

A Better Data Structure for link() and eval()

VF must satisfy one additional property:

1. For each T in F there is a corresponding VT in VF with the same vertices as T.

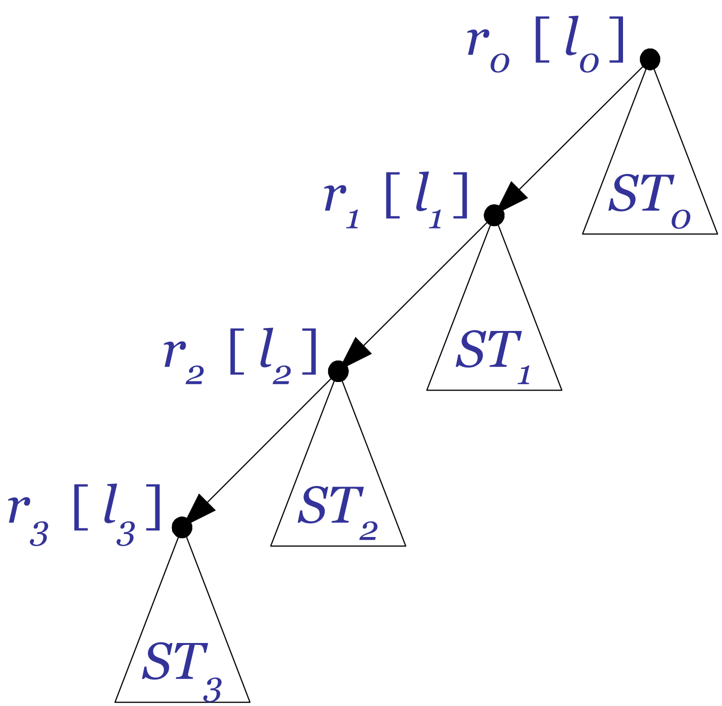
2. Corresponding trees T and VT have the same root with the same label.

3. If v is any vertex, eval(v, F) = eval(v, VF).

4. Each VT consists of subtrees ST_i with roots r_i, $0 \le i \le k$, such that $semi(label(r_j)) \ge semi(label(r_{j+1}))$, $1 \le j < k$.

A Better Data Structure for link() and eval()

4. Each VT consists of subtrees ST_i with roots r_i, $0 \le i \le k$, such that $semi(label(r_j)) \ge semi(label(r_{j+1}))$, $1 \le j < k$.

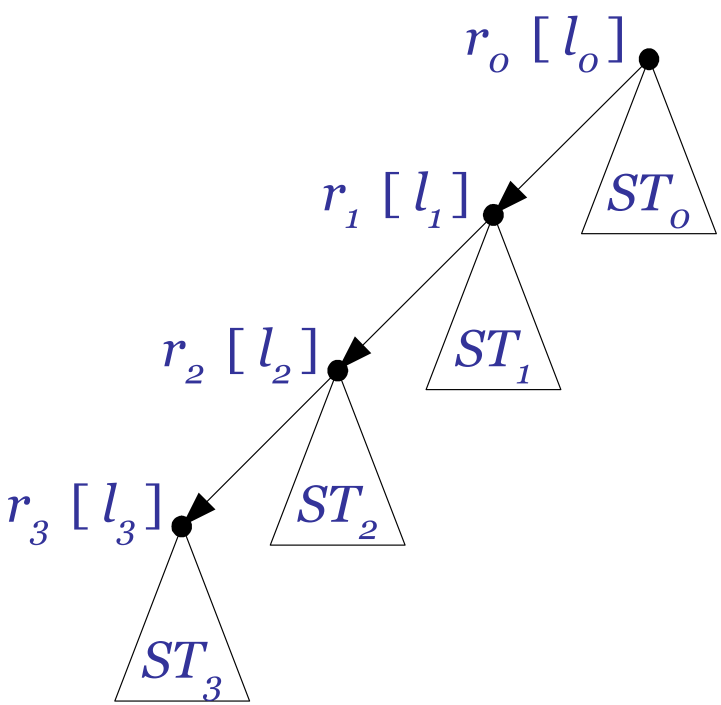


r_0 has not been processed yet
i.e., $semi(r_0) \ne sdom(r_0)$

$semi(l_1) \ge semi(l_2) \ge semi(l_3)$

A Better Data Structure for link() and eval()

4. Each VT consists of subtrees ST_i with roots r_i, $0 \leq i \leq k$, such that $semi(label(r_j)) \geq semi(label(r_{j+1}))$, $1 \leq j < k$.



We need an extra pointer per node:

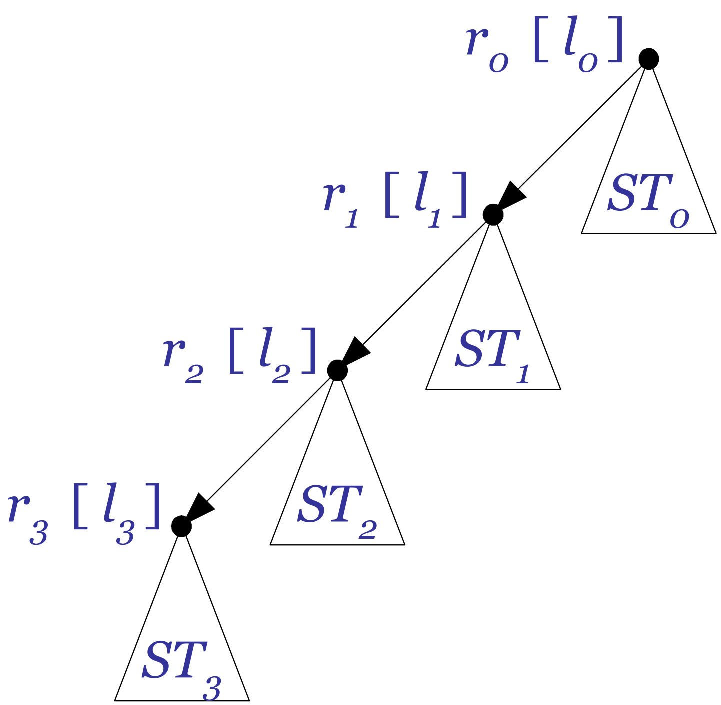
$child(r_j) = r_{j+1}$, $0 \leq j < k$

$ancestor(r_j) = 0$, $0 \leq j \leq k$

$semi(l_1) \geq semi(l_2) \geq semi(l_3)$

A Better Data Structure for link() and eval()

4. Each VT consists of subtrees ST_i with roots r_i, $0 \le i \le k$, such that $semi(label(r_j)) \ge semi(label(r_{j+1}))$, $1 \le j < k$.



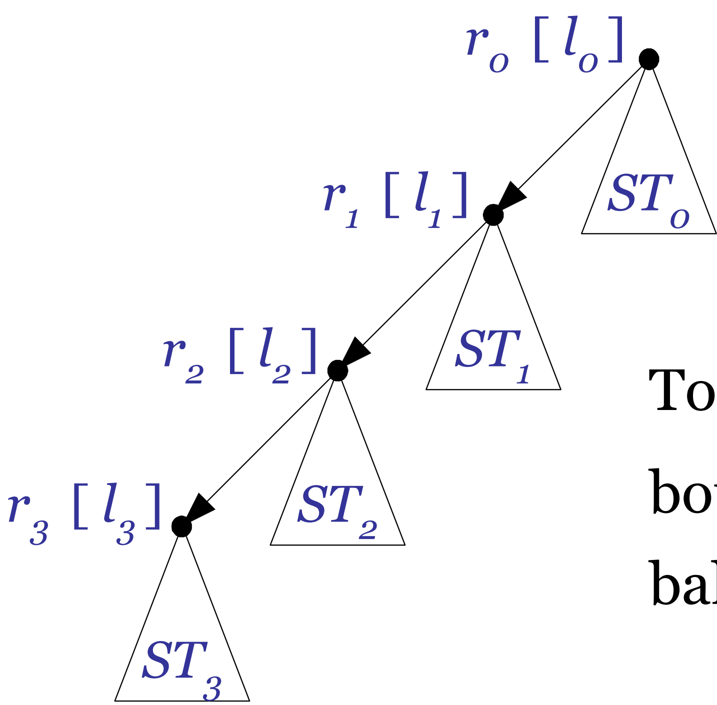
For any v in ST_j, eval(v) doesn't depend on $label(r_i)$, $i < j$.

$\Rightarrow$ We can use path compression inside each ST_j.

$semi(l_1) \ge semi(l_2) \ge semi(l_3)$

A Better Data Structure for link() and eval()

4. Each VT consists of subtrees ST_i with roots r_i, $0 \leq i \leq k$, such that $semi(label(r_j)) \geq semi(label(r_{j+1}))$, $1 \leq j < k$.

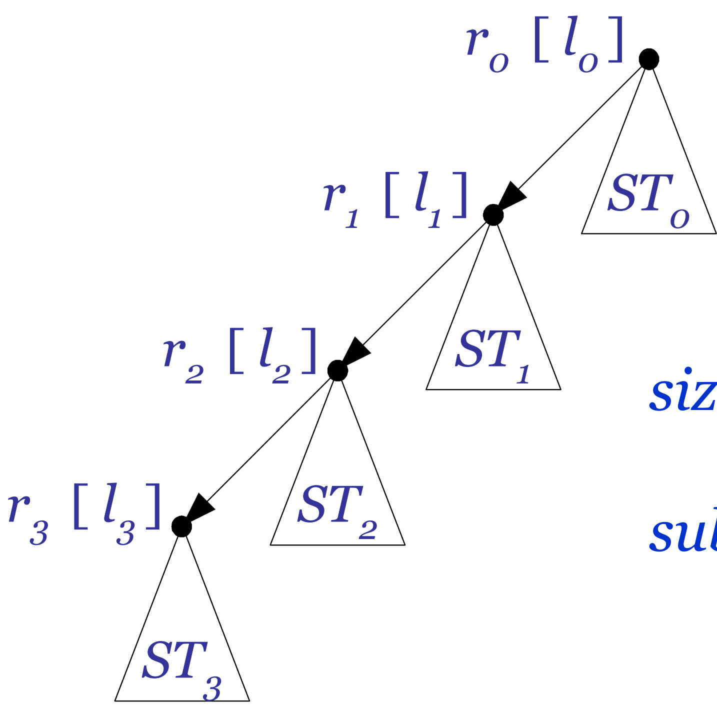


To get the $O((m+n) \cdot \alpha(m+n, n))$ time bound we want to keep each ST_j balanced.

$semi(l_1) \geq semi(l_2) \geq semi(l_3)$

A Better Data Structure for link() and eval()

4. Each VT consists of subtrees ST_i with roots r_i, $0 \le i \le k$, such that $semi(label(r_j)) \ge semi(label(r_{j+1}))$, $1 \le j < k$.



$size(r_j) = |ST_j| + |ST_{j+1}| + \ldots + |ST_k|$

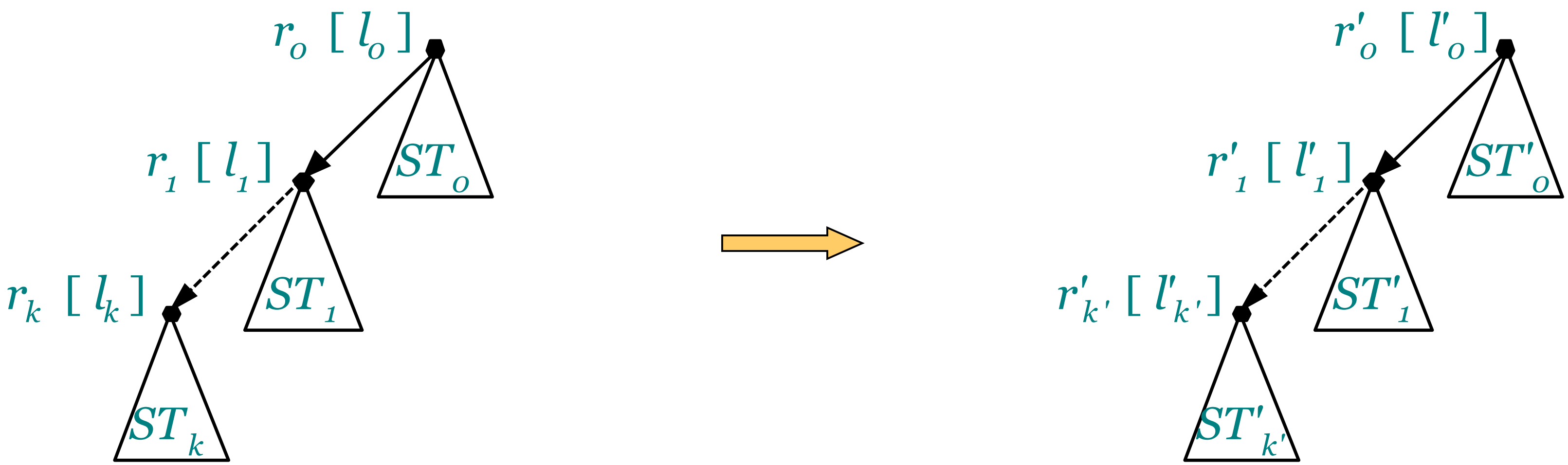
$subsize(r_j) = |ST_j| = size(r_j) - size(r_{j+1})$

$semi(l_1) \ge semi(l_2) \ge semi(l_3)$

A Better Data Structure for link() and eval()

First we implement the following auxiliary operation:

update(r): If r is a root in F and $l = label(r)$ then restore Property 4 for all subtree roots.

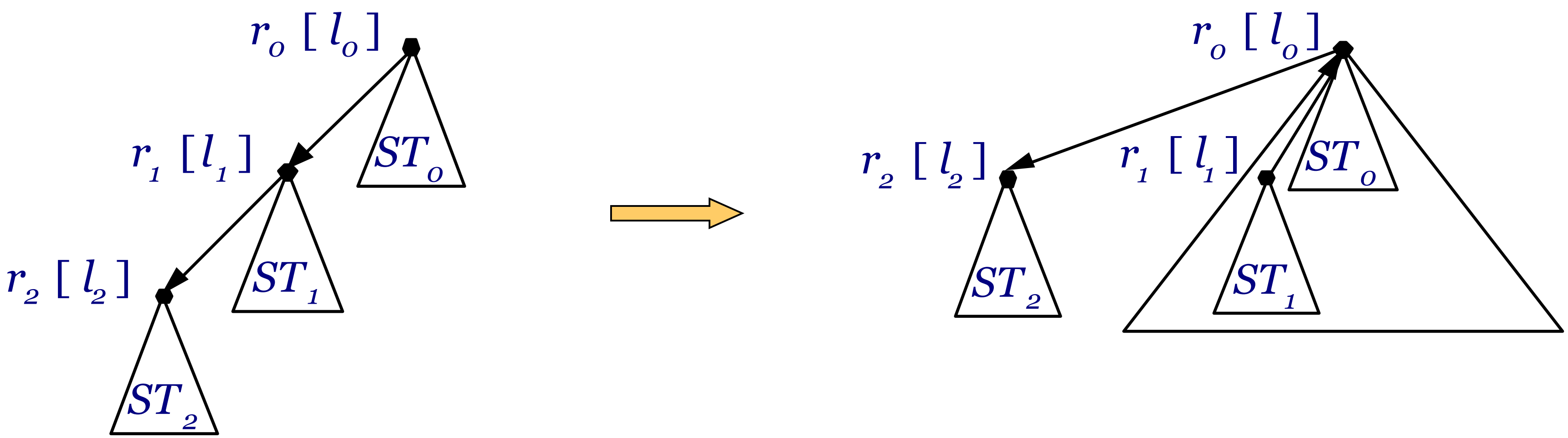


$semi(label(r_j)) \geq semi(label(r_{j+1}))$
$1 \leq j < k$.

$semi(label(r'_j)) \geq semi(label(r'_{j+1}))$
$0 \leq j < k'$.

Implementation of update()

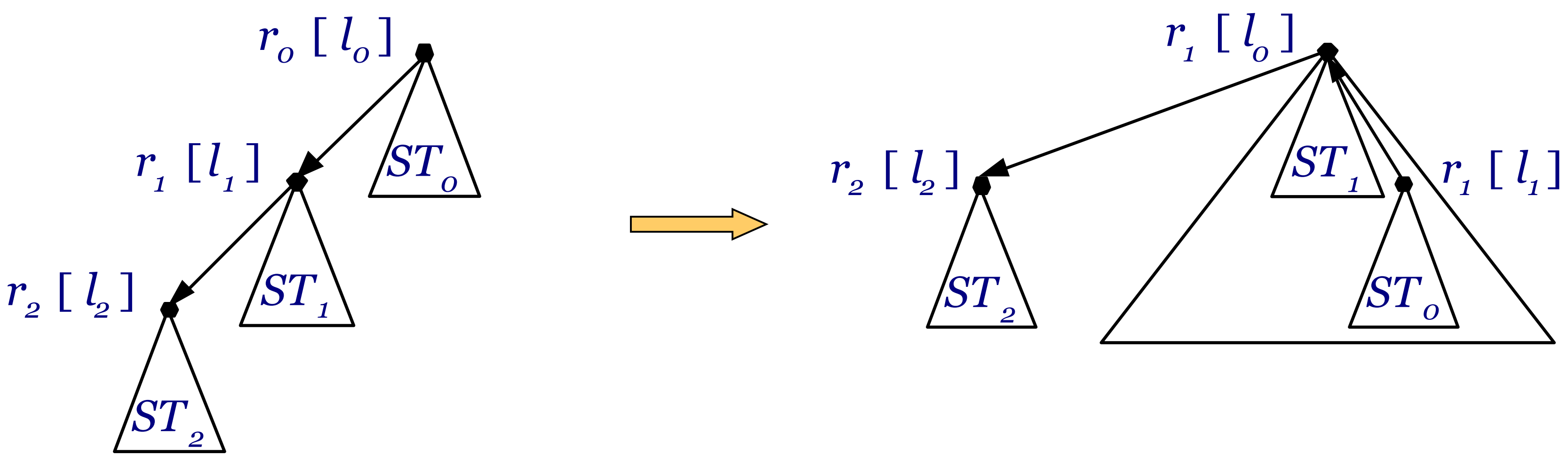
Suppose *semi*(l_0) < *semi*(l_1)



Case (a): $subsize(r_1) \geq subsize(r_2)$

Implementation of update()

Suppose $semi(l_0) < semi(l_1)$



Case (b): $subsize(r_1) < subsize(r_2)$

Implementation of update()

update(r): Let VT be the virtual tree rooted at $r = r_0$, with subtrees ST_i and corresponding roots r_i, $0 \le i \le k$.

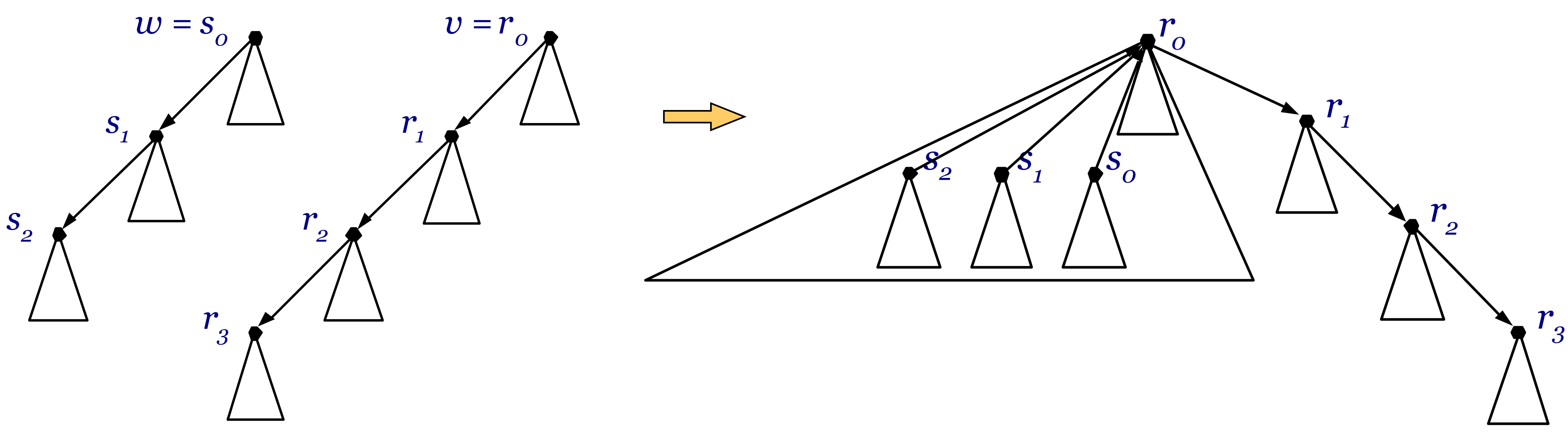
If $semi(label(r)) < semi(label(r_1))$ then combine ST_0 and ST_1 to a new subtree ST'_0. Repeat this process for ST_j, $i = 2, \ldots, j$, where $j=k$ or $semi(label(r)) \ge semi(label(r_j))$.

Set the label of the root r'_0 of the final subtree ST'_0 equal to $label(r)$ and set $child(r) = r'_1$.

Implementation of link()

link(v, w): update(w, $label(v)$)

Combine the virtual trees rooted at v and w.

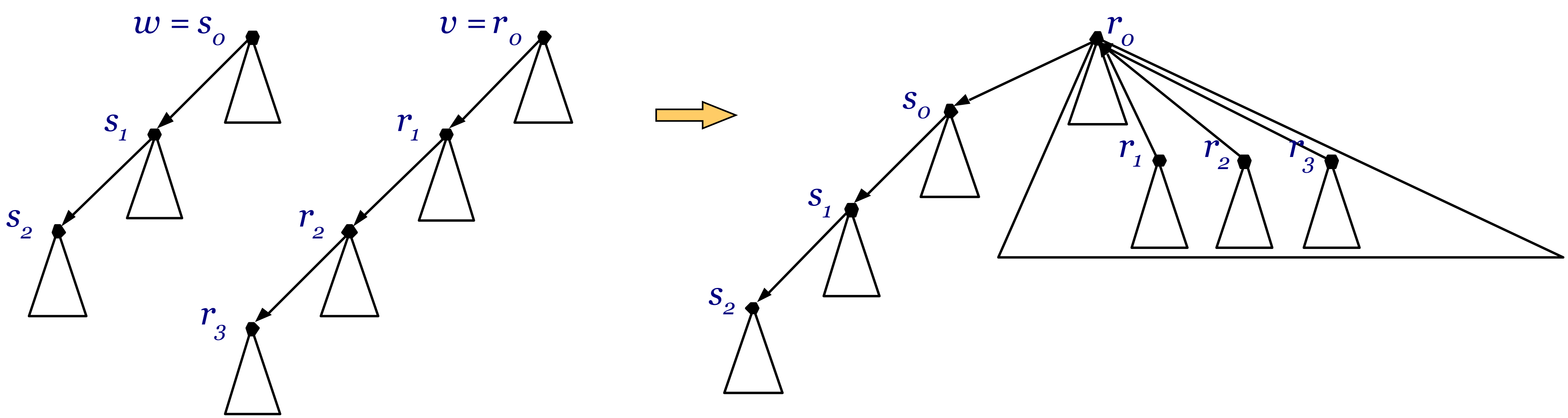


Case (a): $size(v) \geq size(w)$

Implementation of link()

link(v, w): update(w, *label(v)*)

Combine the virtual trees rooted at v and w.



Case (b): *size(v)* < *size(w)*

Implementation of link()

link(v, w): update(w, $label(v)$)

Let VT_1 be the virtual tree rooted at $v = r_0$, with subtree roots r_i, $0 \le i \le k$.

Let VT_2 be the virtual tree rooted at $w = s_0$, with subtree roots s_i, $0 \le i \le l$.

If $size(v) \ge size(w)$ make v parent of $s_0, s_1, \ldots, s_l$. Otherwise make v parent of $r_1, r_2, \ldots, r_k$ and make w the child of v.

Analysis of link()

We will show that the (uncompressed) subtrees built by link() are balanced.

U : uncompressed forest.

Just before $ancestor(y) \leftarrow x \;\Rightarrow\; x$ and y are subtree roots.
Then we add (x,y) to U. Let (x,y), $(y,z) \in U$.

(x,y) is **good** if $subsize(x) \geq 2 \cdot subsize(y)$

(y,z) is **mediocre** if $subsize(x) \geq 2 \cdot subsize(z)$

Analysis of link()

(x,y) is **good** if $subsize(x) \geq 2 \cdot subsize(y)$

(y,z) is **mediocre** if $subsize(x) \geq 2 \cdot subsize(z)$

Every edge added by update() is good.

Assume (x,y) and (y,z) are added outside update().

$$(y,z) \Rightarrow size(y) \geq 2 \cdot size(z)$$

$$(x,y) \Rightarrow subsize(x) \geq 2 \cdot subsize(z)$$

Thus, every edge added by link() is mediocre.

Analysis of link()

For any x, y and z in U such that $x \to y \to z$, we have $subsize(x) \geq 2 \cdot subsize(z)$.

It follows by induction that any vertex of height h in U has $\geq 2^{\lfloor h/2 \rfloor}$ descendants.

The number of vertices of height h in U is $\leq n/2^{\lfloor h/2 \rfloor}$ $\leq 2^{1/2}\, n/(2^{1/2})^h \;\Rightarrow\; U$ is balanced for constants $a = \sqrt{2},\ b = \sqrt{2}$.

Lengauer-Tarjan Running Time

By Theorem 5, m calls to eval() take $O((m+n) \cdot \alpha(m+n, n))$ time.

The total time for all the link() instructions is proportional to the number of edges in $U \Rightarrow O(n+m)$ time.

The Lengauer-Tarjan algorithm runs in $O(m \cdot \alpha(m,n))$ time.