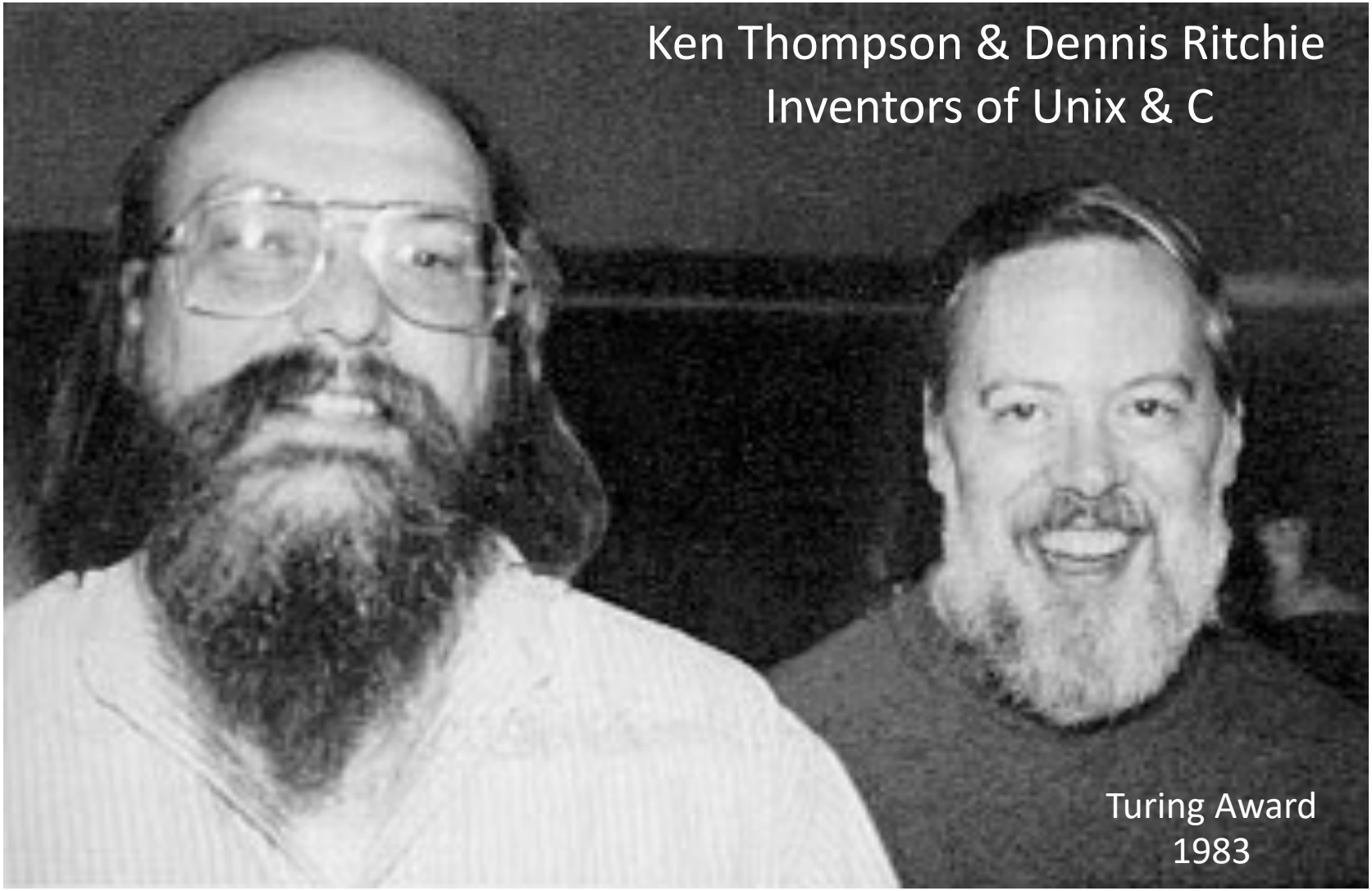
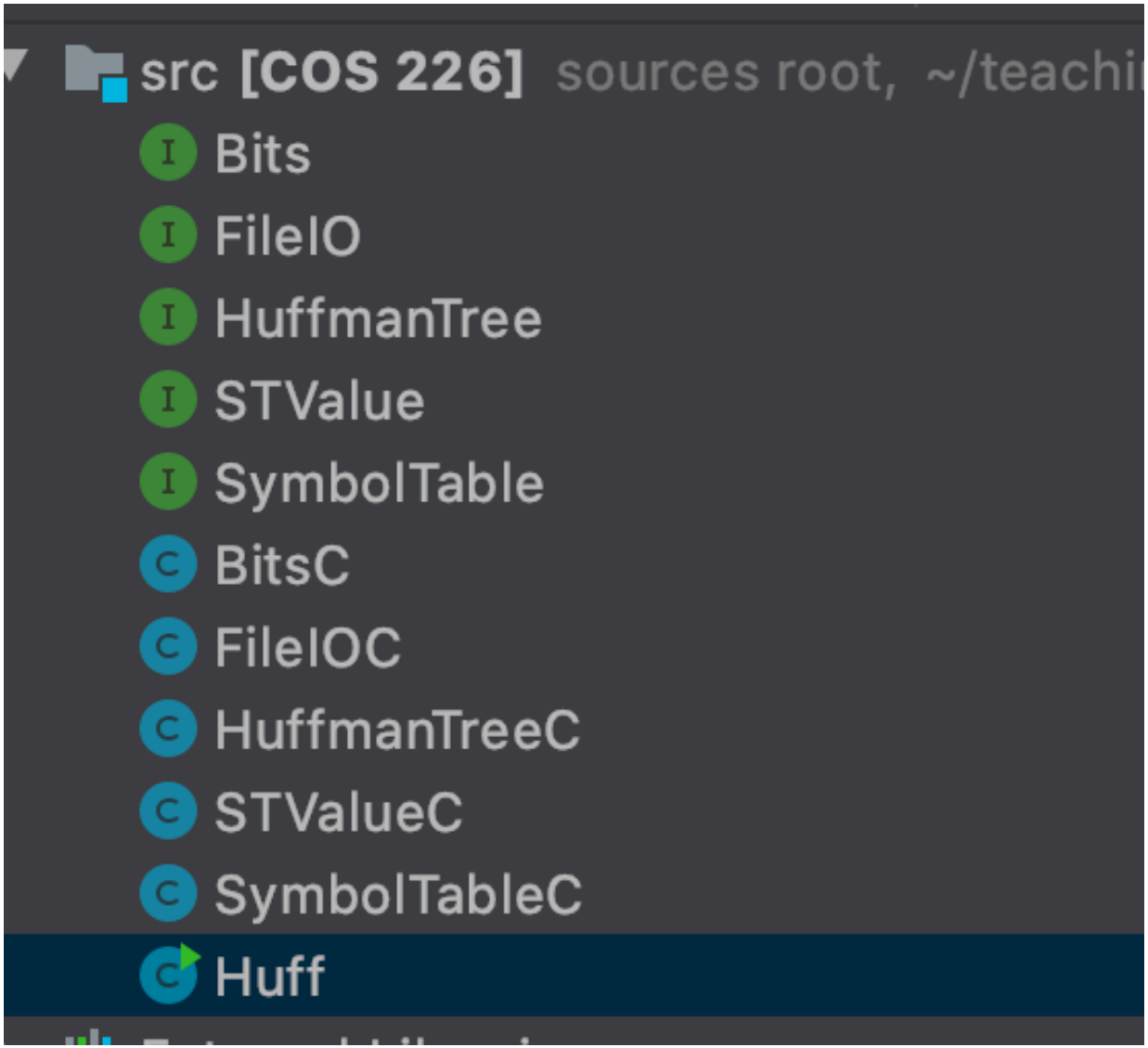
CSCI 1102 Computer Science 2

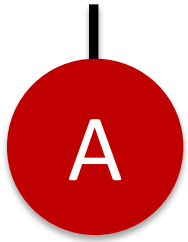
Meeting 21: Thursday 4/15/2021

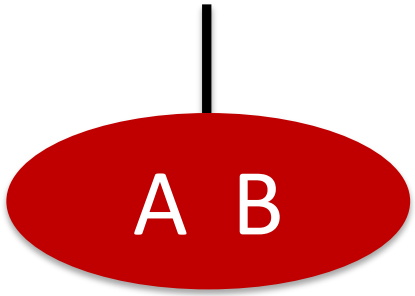
More on Red Black Trees

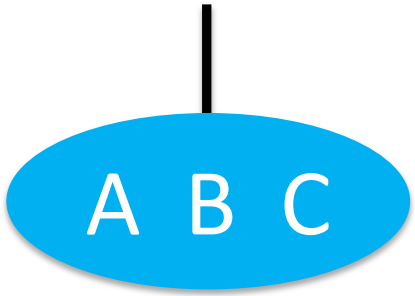

Ken Thompson & Dennis Ritchie
Inventors of Unix & C
Turing Award
1983

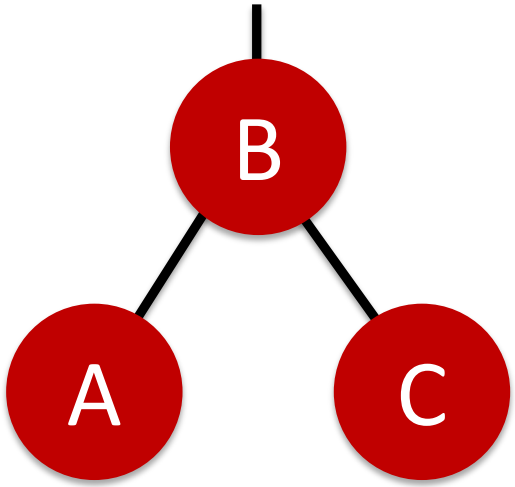

src [COS 226] sources root, ~/teachi
Bits
FileIO
HuffmanTree
STValue
SymbolTable
BitsC
FileIOC
HuffmanTreeC
STValueC
SymbolTableC
Huff

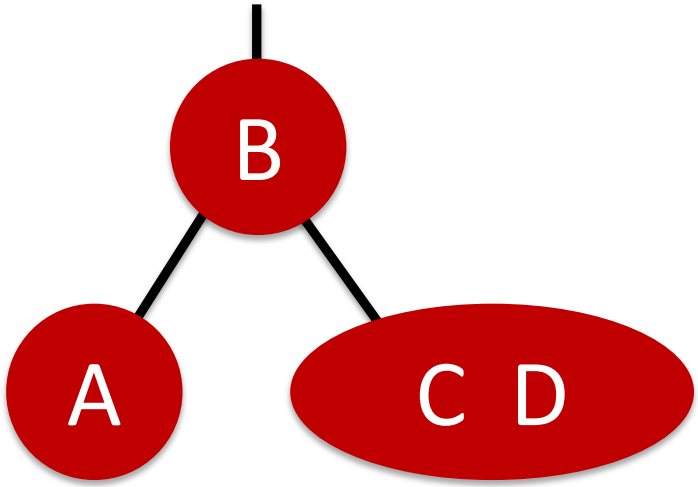
Show all of the 2-3 trees resulting from inserting the keys ABCDEFG

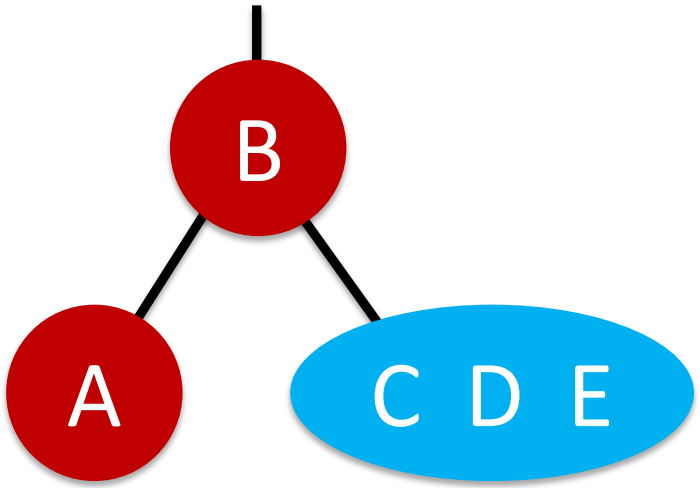

A

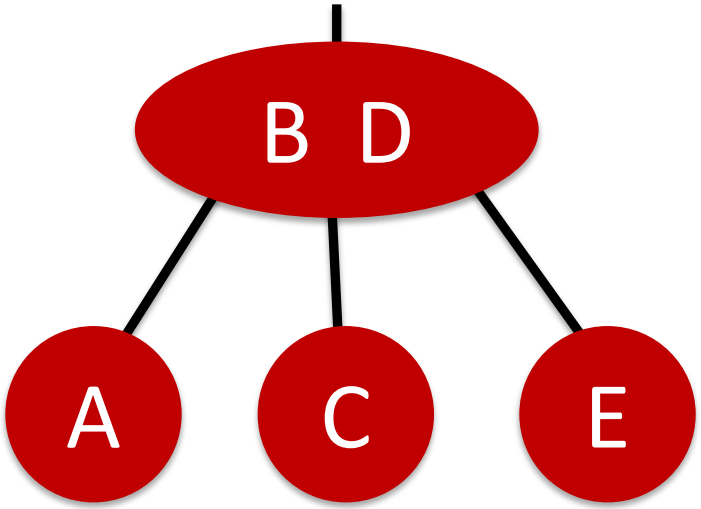

A B

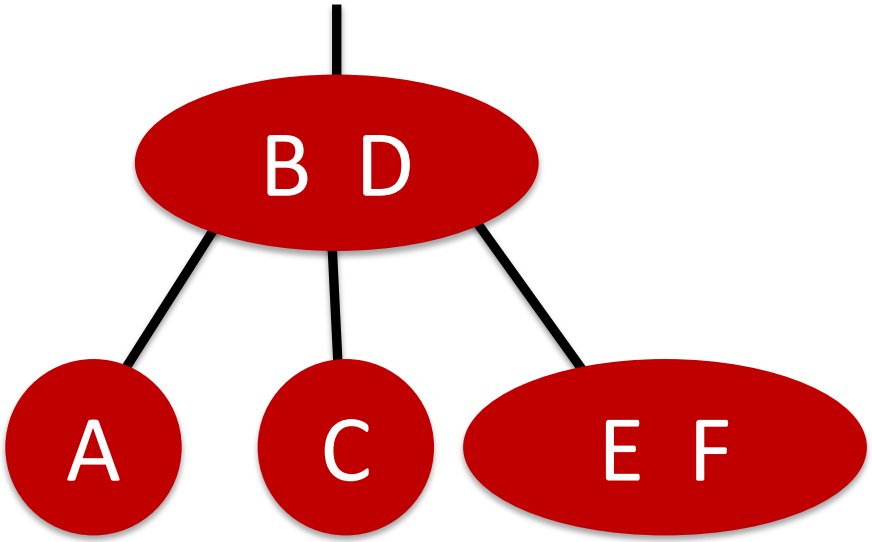

A B C

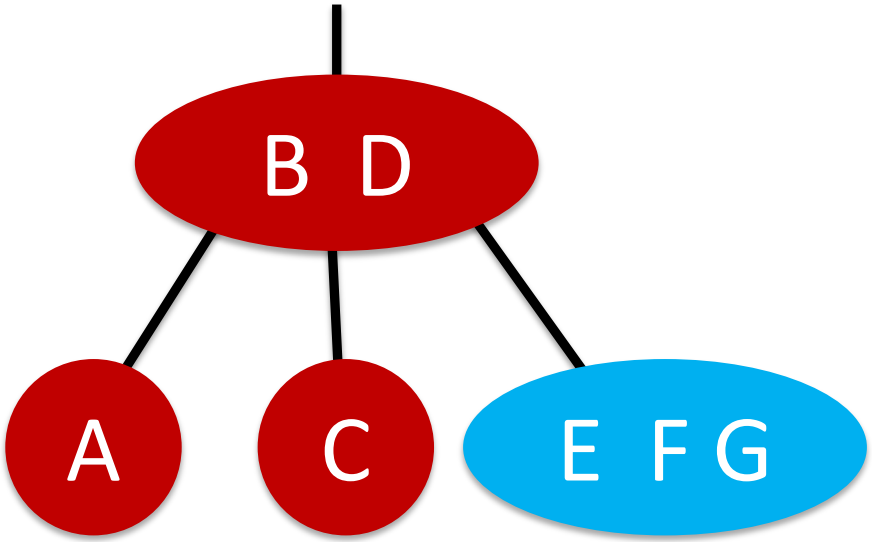

B
A
C

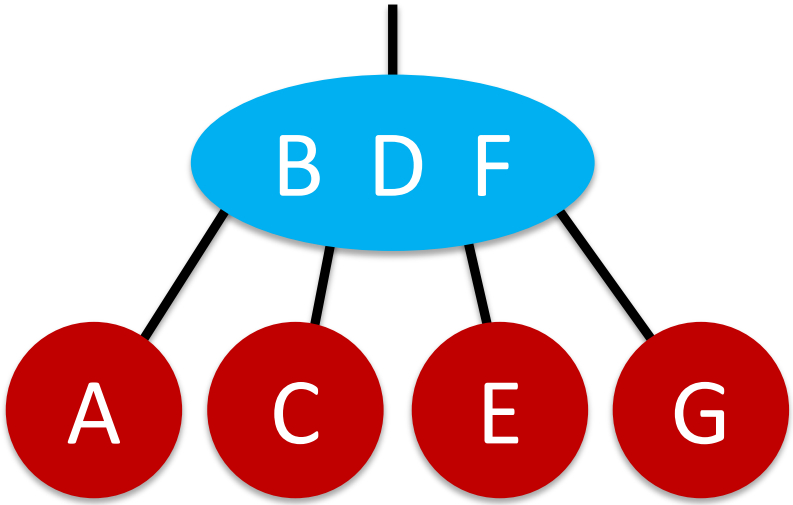

B
A
C D

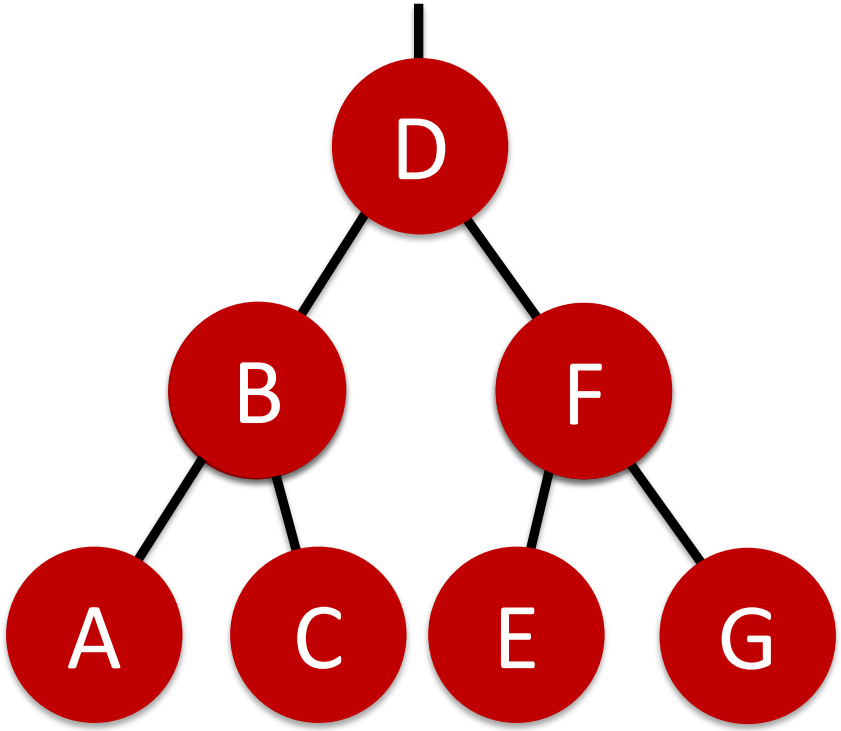

B
A
C D E


B D
A
C
E


B D
A
C
E F

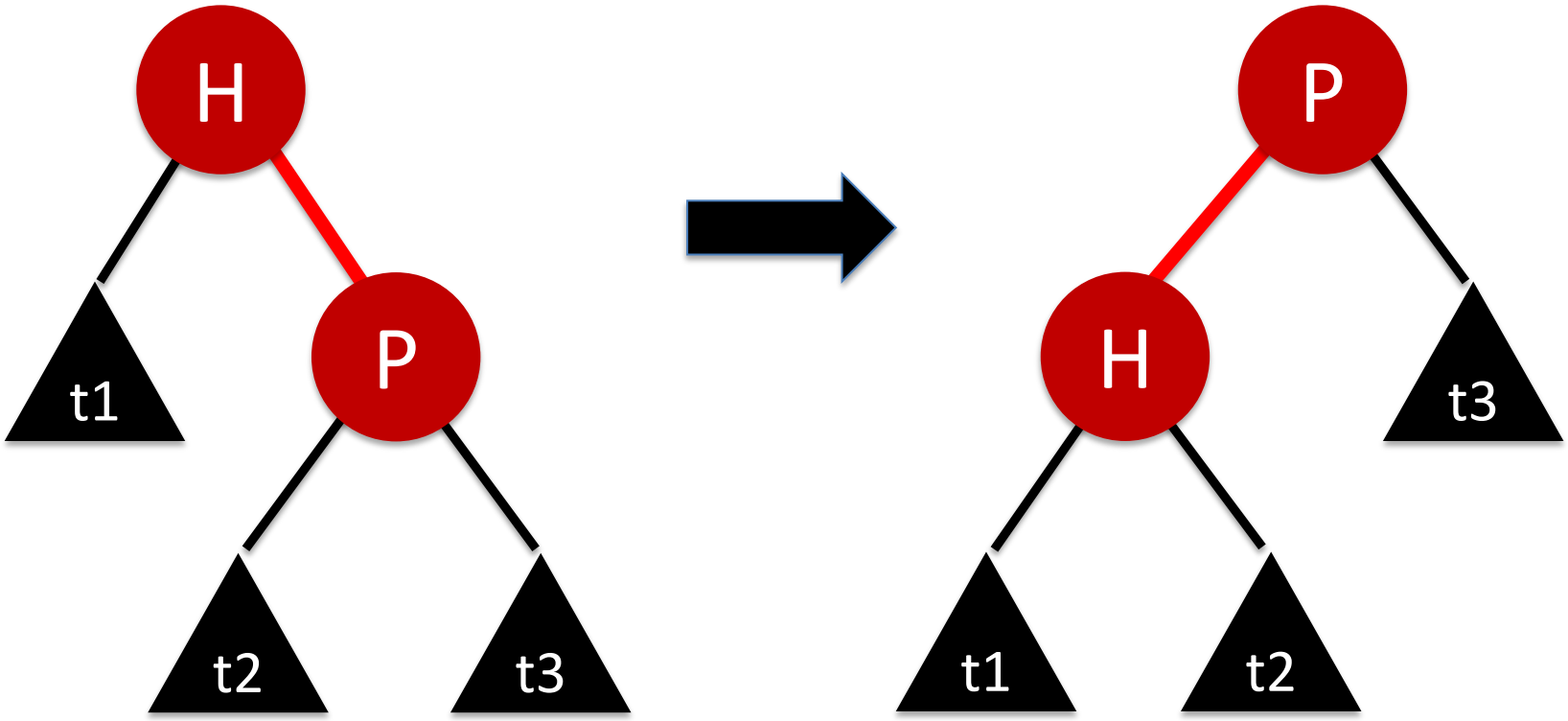

B D
A
C
E F G


B D F
A
C
E
G

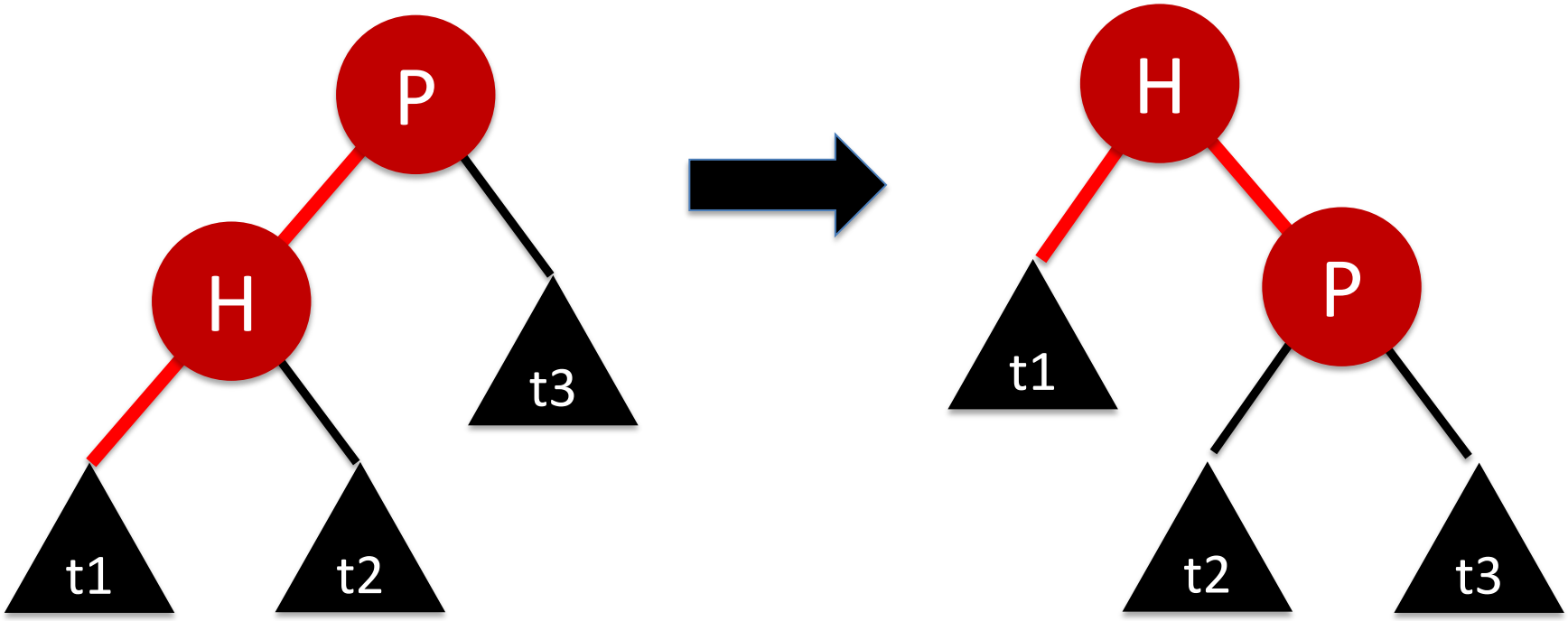

D
B
F
A
C
E
G

Left-Leaning Red-Black Trees

Rotate Left



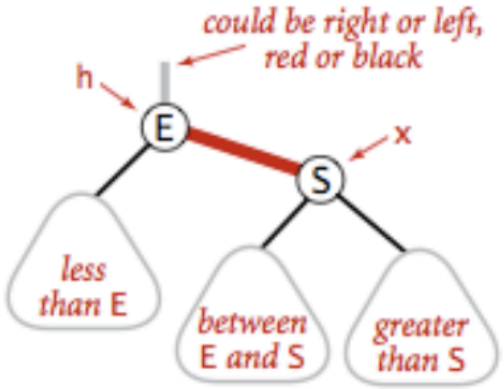
Rotate Right



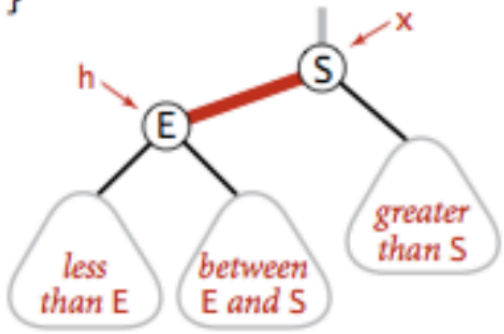
```java
// insert the key-value pair in the subtree rooted at h
private Node put(Node h, Key key, Value val) {
  if (h == null) return new Node(key, val, RED, 1);

  int cmp = key.compareTo(h.key);
  if (cmp < 0) h.left = put(h.left, key, val);
  else if (cmp > 0) h.right = put(h.right, key, val);
  else h.val = val;

  // fix-up any right-leaning links
  if (isRed(h.right) && !isRed(h.left)) h = rotateLeft(h);
  if (isRed(h.left) && isRed(h.left.left)) h = rotateRight(h);
  if (isRed(h.left) && isRed(h.right)) flipColors(h);
  h.size = size(h.left) + size(h.right) + 1;

  return h;
}
```

```
// insert the key-value pair in the subtree rooted at h
private Node put(Node h, Key key, Value val) {
  if (h == null) return new Node(key, val, RED, 1);

  int cmp = key.compareTo(h.key);
  if (cmp < 0) h.left = put(h.left, key, val);
  else if (cmp > 0) h.right = put(h.right, key, val);
  else h.val = val;

  // fix-up any right-leaning links
  if (isRed(h.right) && !isRed(h.left)) h = rotateLeft(h);
  if (isRed(h.left) && isRed(h.left.left)) h = rotateRight(h);
  if (isRed(h.left) && isRed(h.right)) flipColors(h);
  h.size = size(h.left) + size(h.right) + 1;

  return h;
}
```

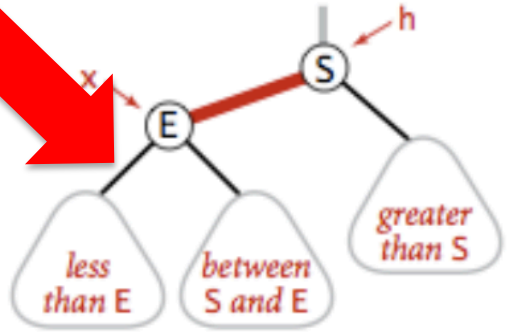
Always Red



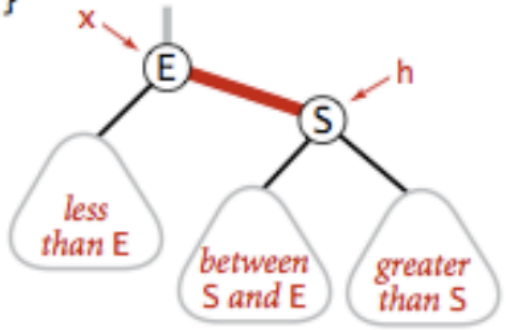

```
Node rotateLeft(Node h)
{
   Node x = h.right;
   h.right = x.left;
   x.left = h;
   x.color = h.color;
   h.color = RED;
   x.N = h.N;
   h.N = 1 + size(h.left)
           + size(h.right);
   return x;
}
```



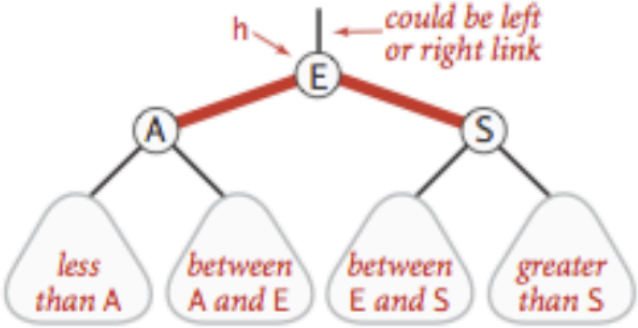
Left rotate (right link of h)



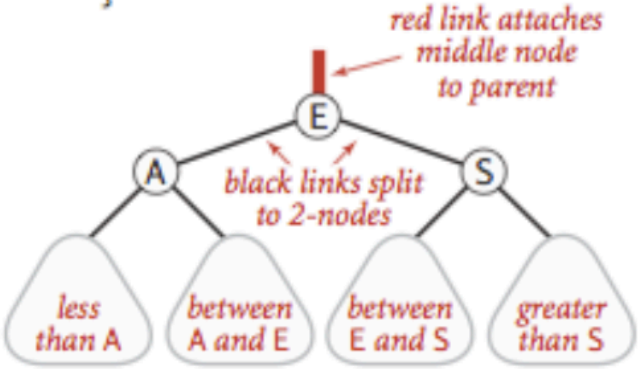

```
Node rotateRight(Node h)
{
   Node x = h.left;
   h.left = x.right;
   x.right = h;
   x.color = h.color;
   h.color = RED;
   x.N = h.N;
   h.N = 1 + size(h.left)
           + size(h.right);
   return x;
}
```



Right rotate (left link of h)

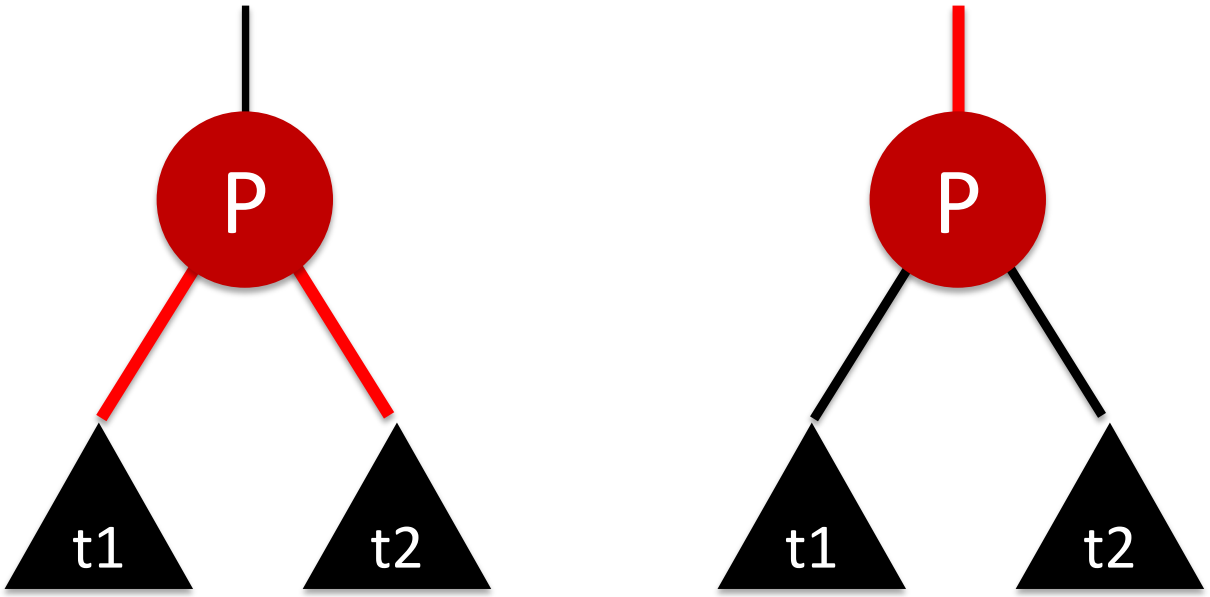



```
void flipColors(Node h)
{
   h.color = RED;
   h.left.color = BLACK;
   h.right.color = BLACK;
}
```



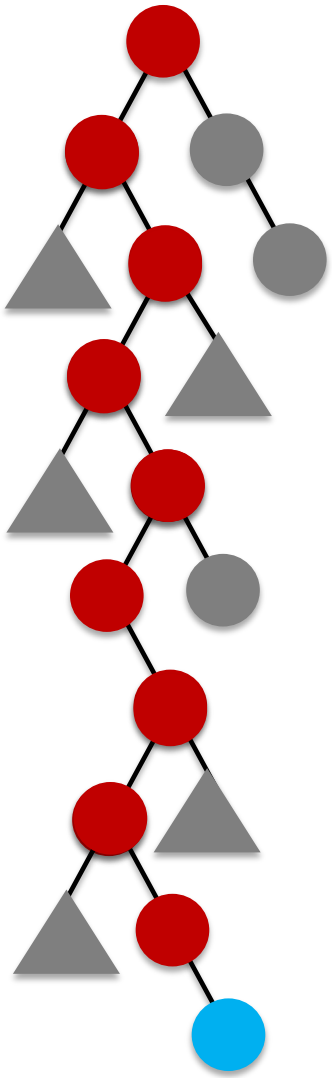
Flipping colors to split a 4-node

Flip



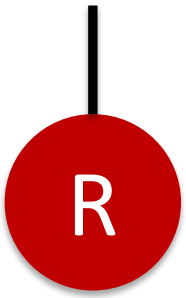
Fixup on the way out

- Insertion point in blue
- Fixup code pending at each (red) node on the path from the insertion point back to the root.

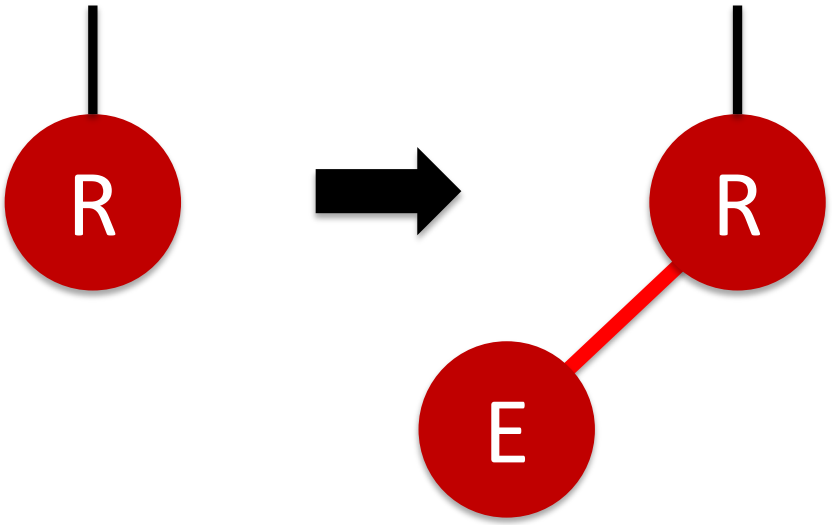


REDBLACK

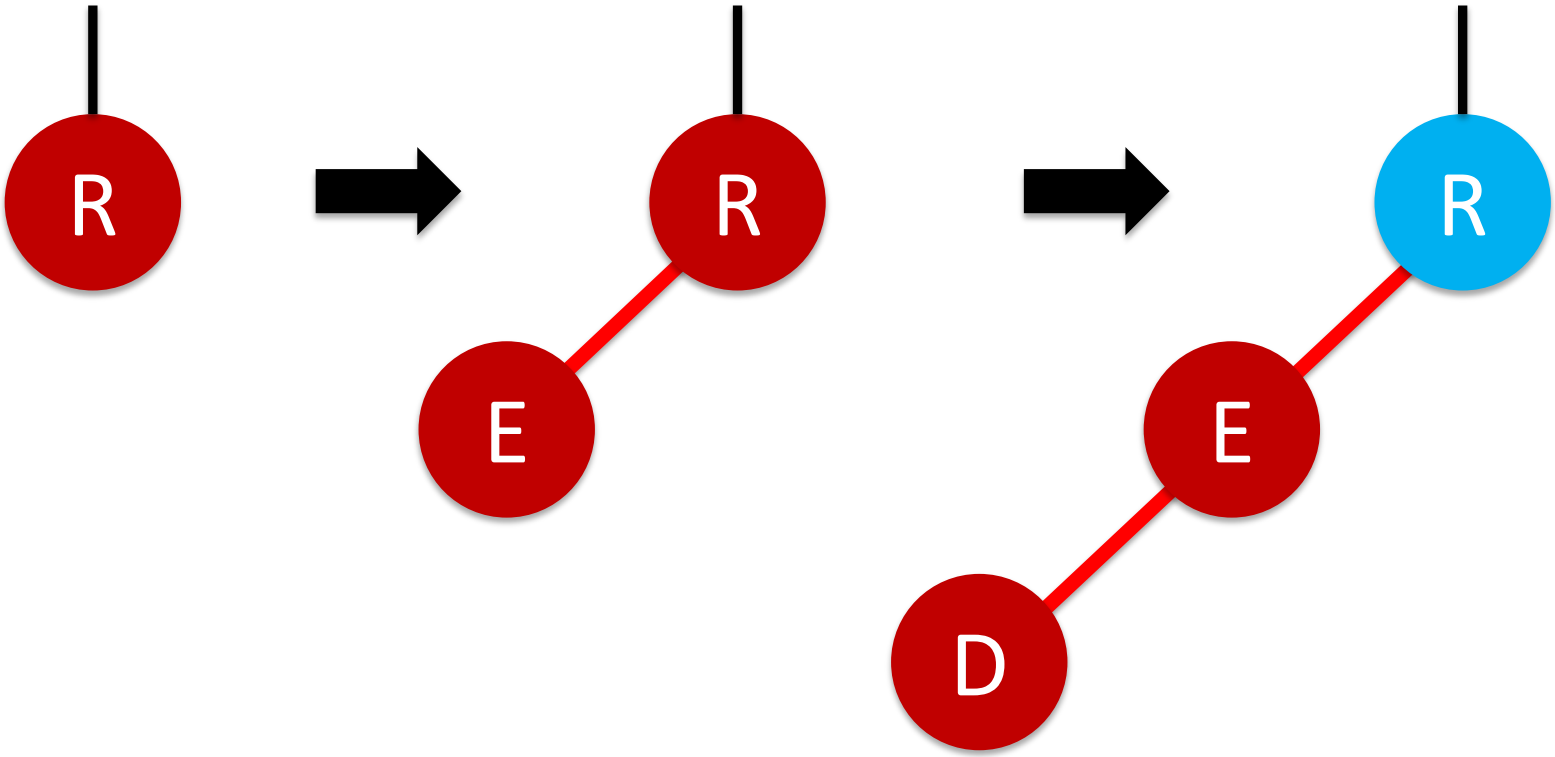

R
E
D
B
L
A
C
K

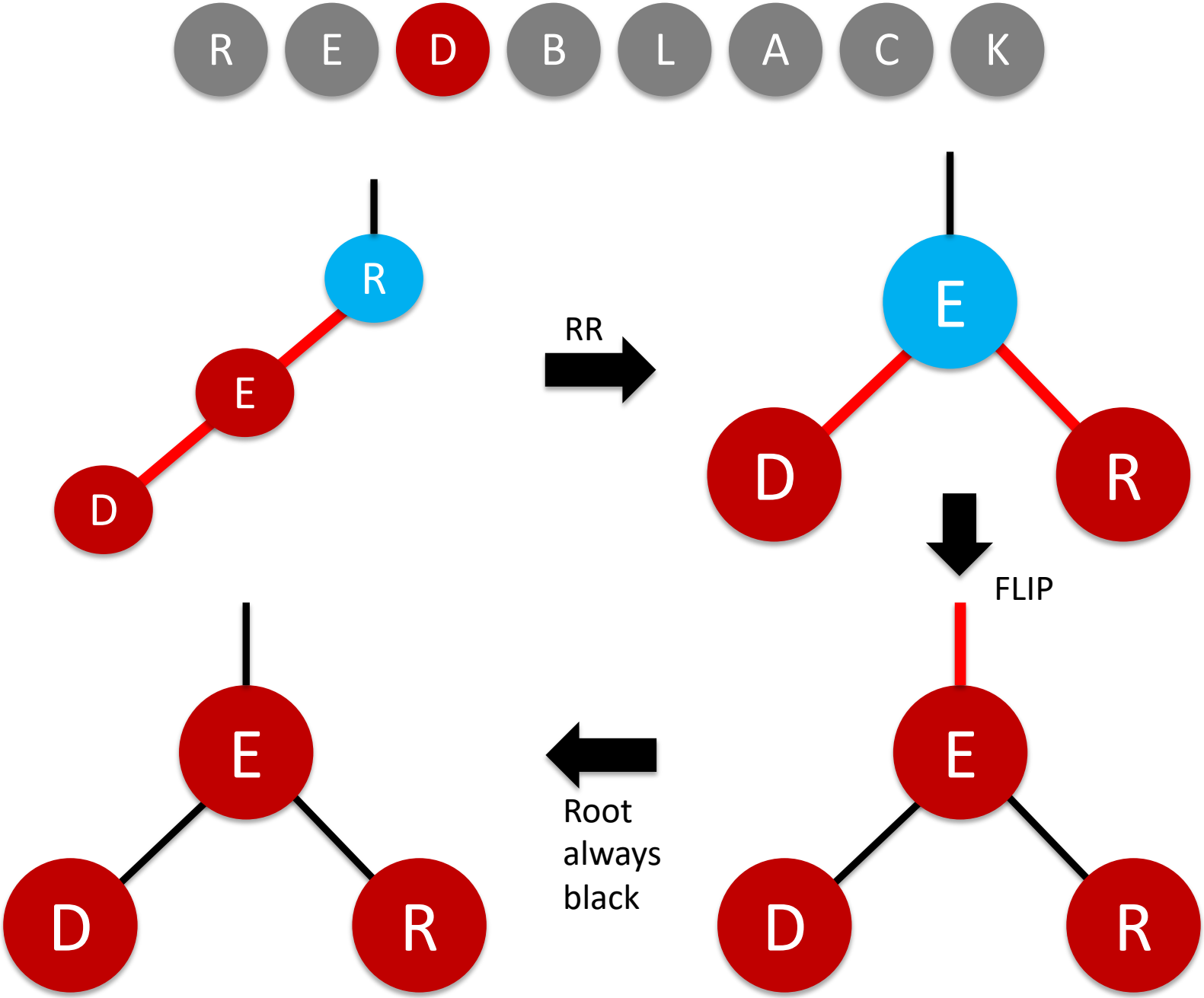

R

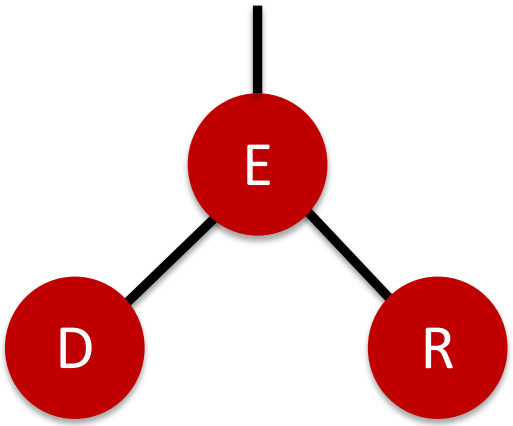

R
E
D
B
L
A
C
K


R
R
E


R
E
D
B
L
A
C
K

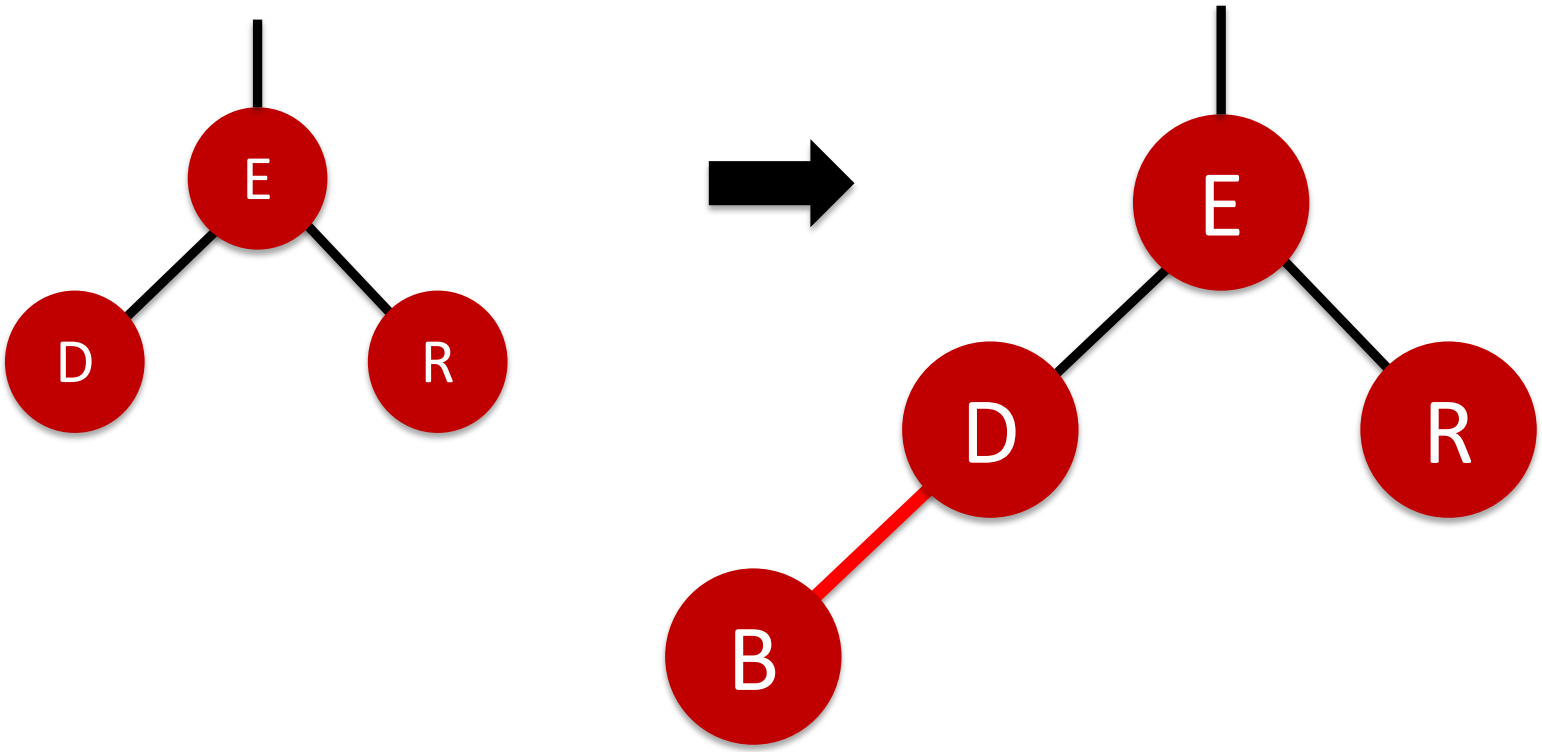

R
R
E
R
E
D


R E D B L A C K
R
E
D
RR
E
D
R
FLIP
E
D
R
Root
always
black
E
D
R

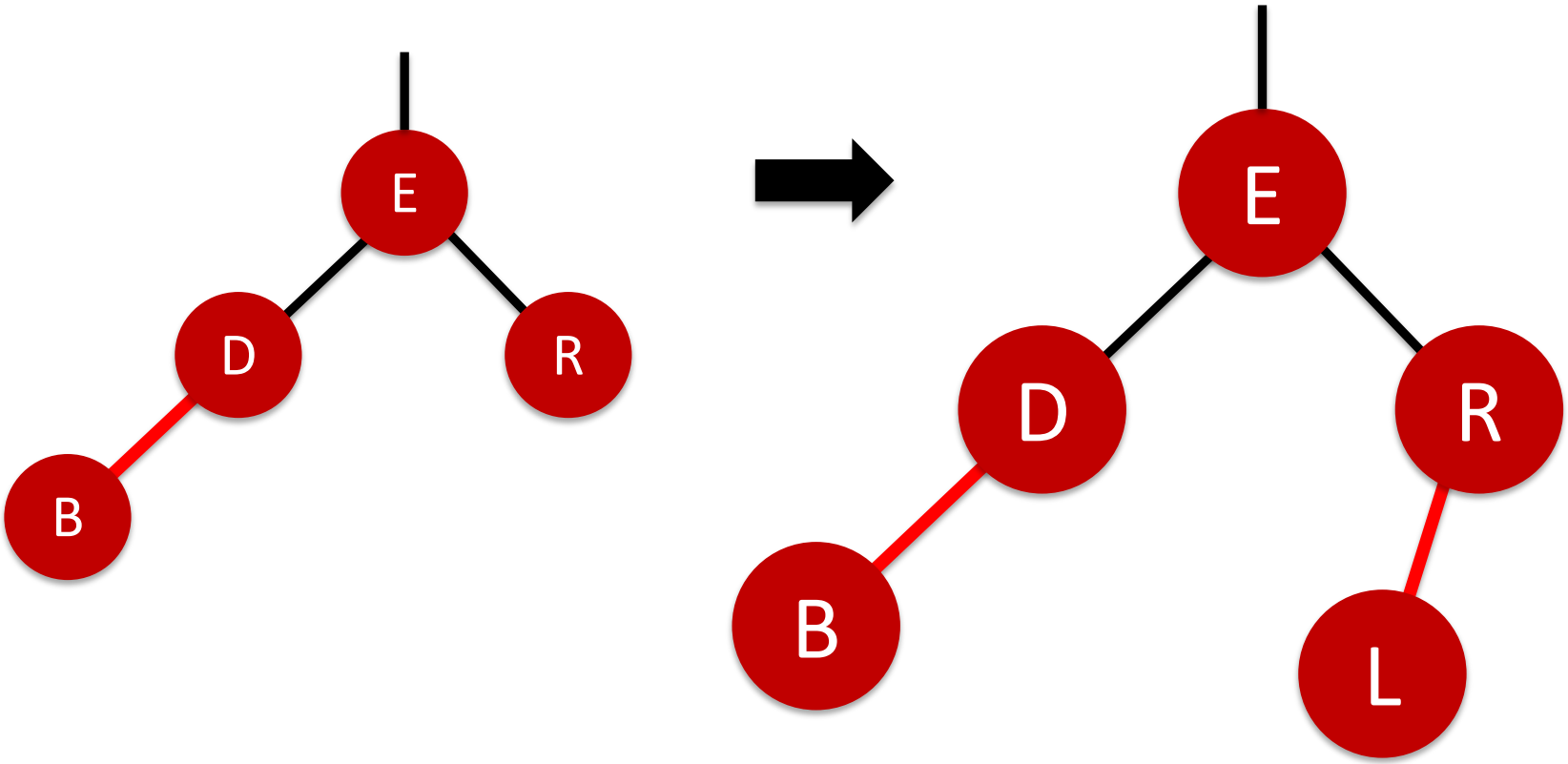


Finish the sequence of insertions showing all tree transformations.

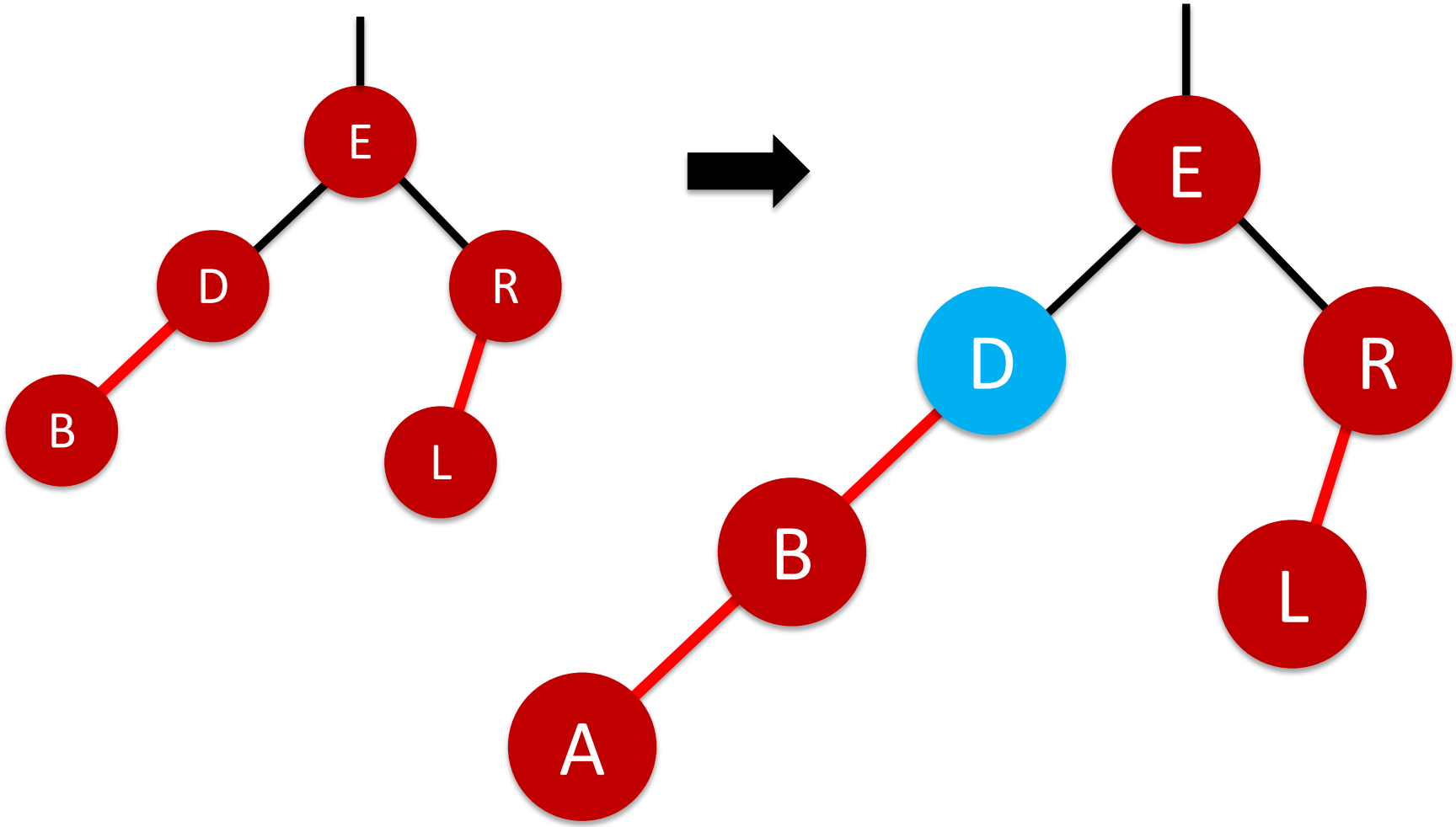

R
E
D
B
L
A
C
K


E
D
R
E
D
R
B

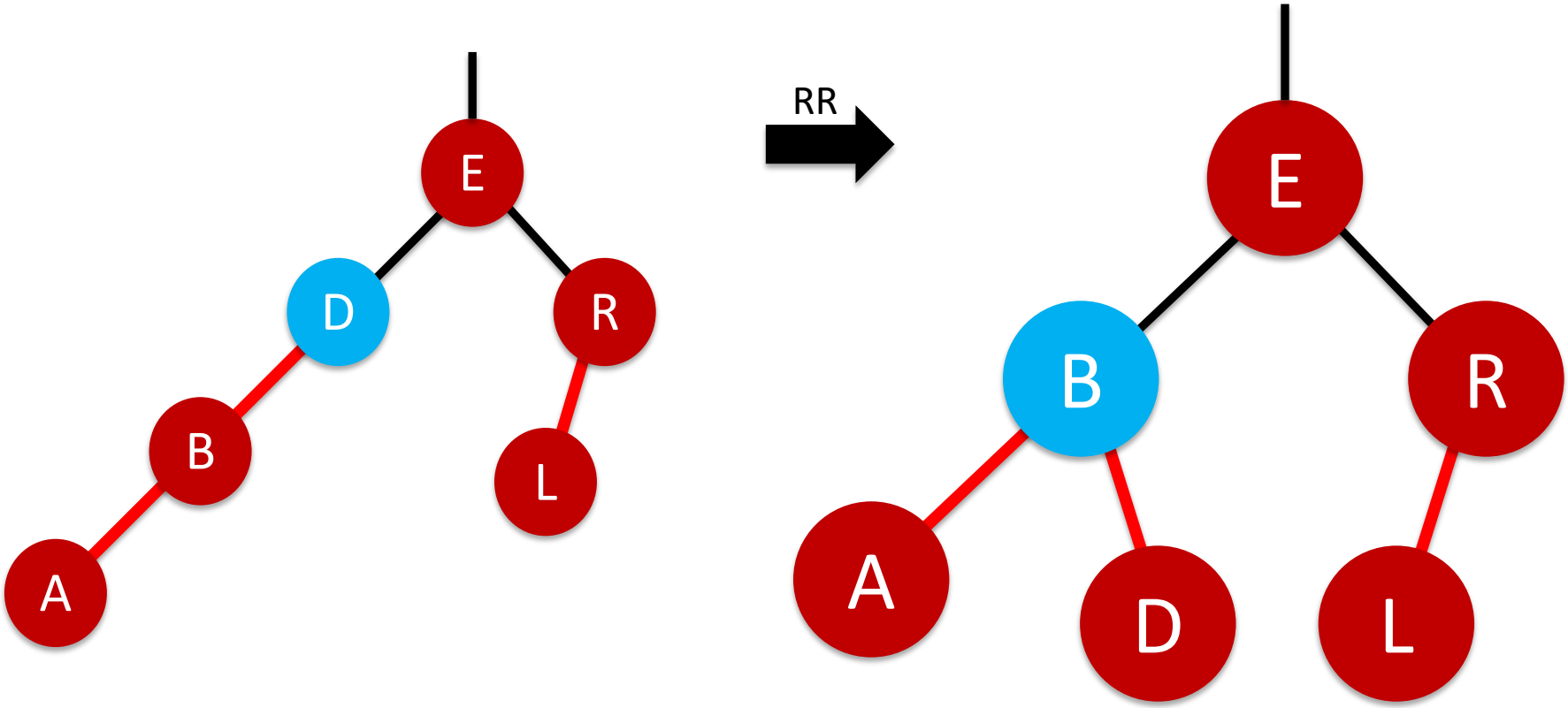

R
E
D
B
L
A
C
K


E
D
R
B
E
D
R
B
L

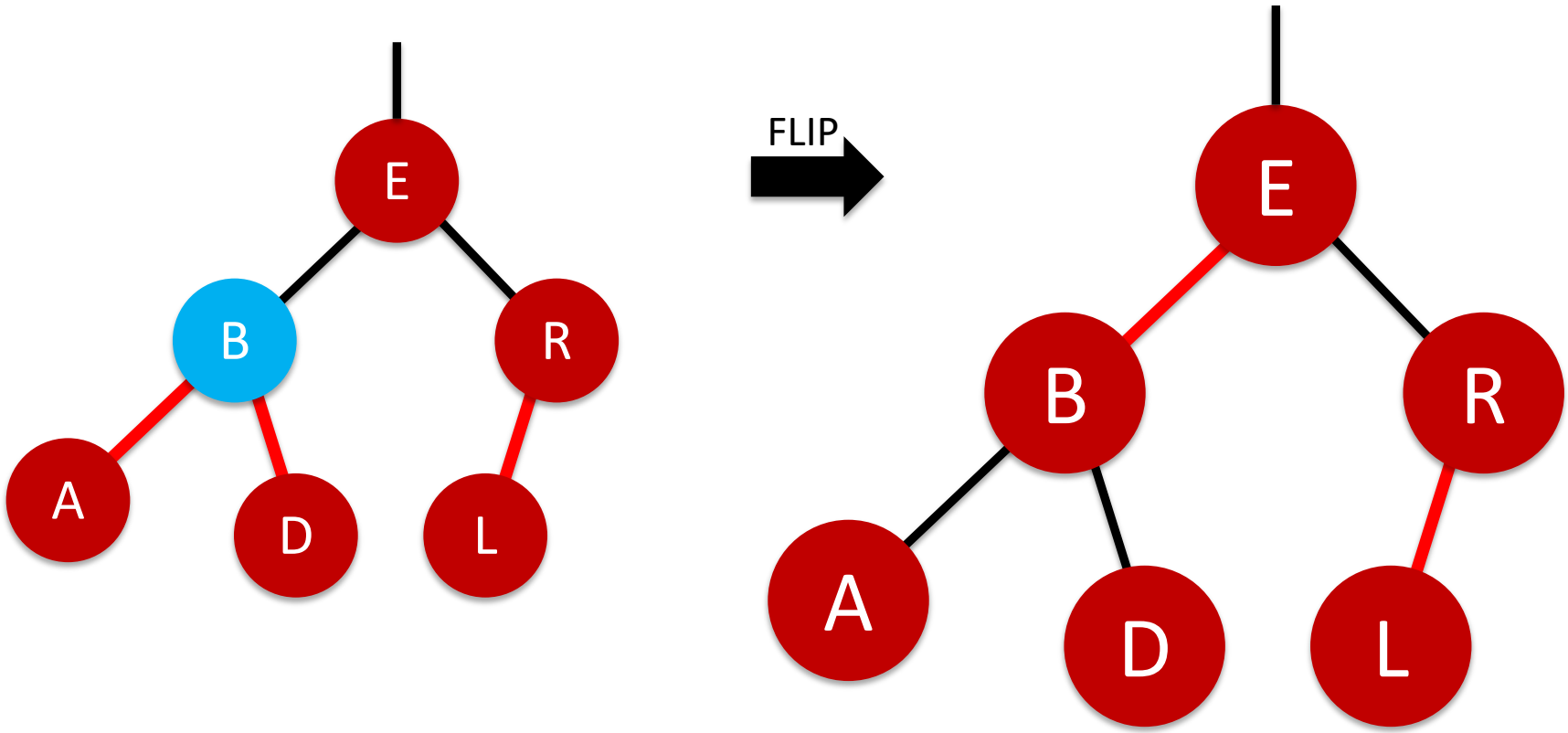

R
E
D
B
L
A
C
K


E
D
R
B
L
E
D
R
B
L
A

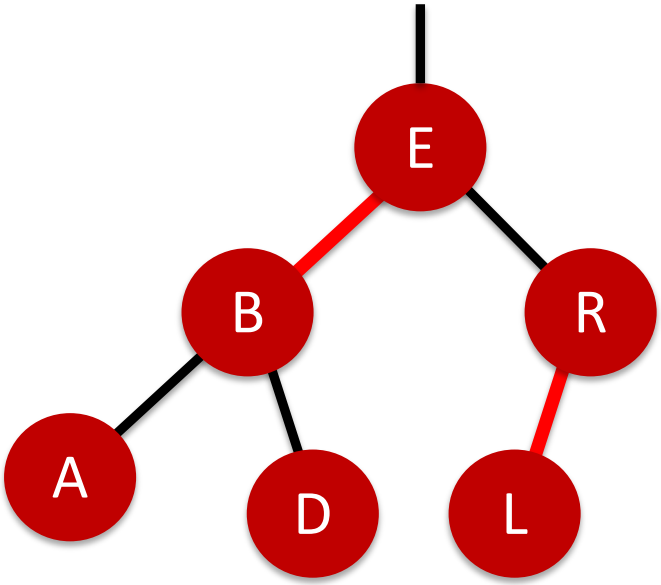

R
E
D
B
L
A
C
K

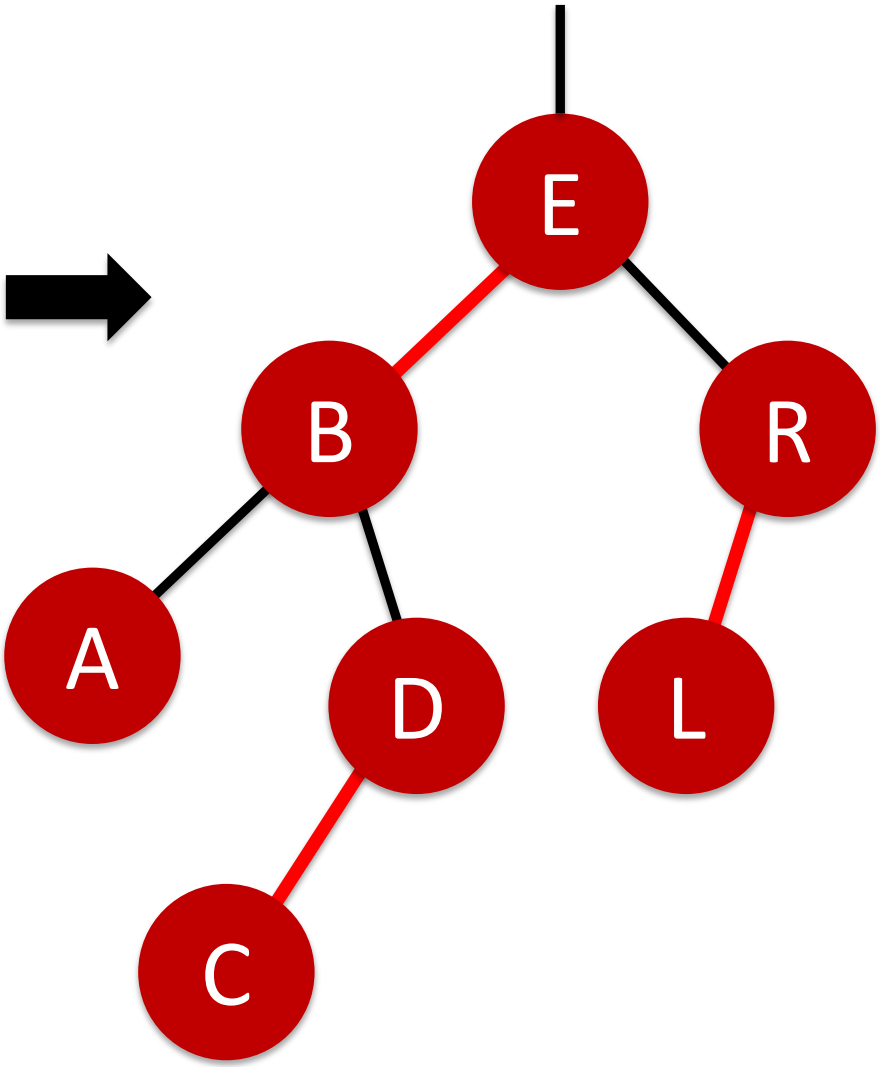

E
D
R
B
L
A
RR
E
B
R
A
D
L


R
E
D
B
L
A
C
K

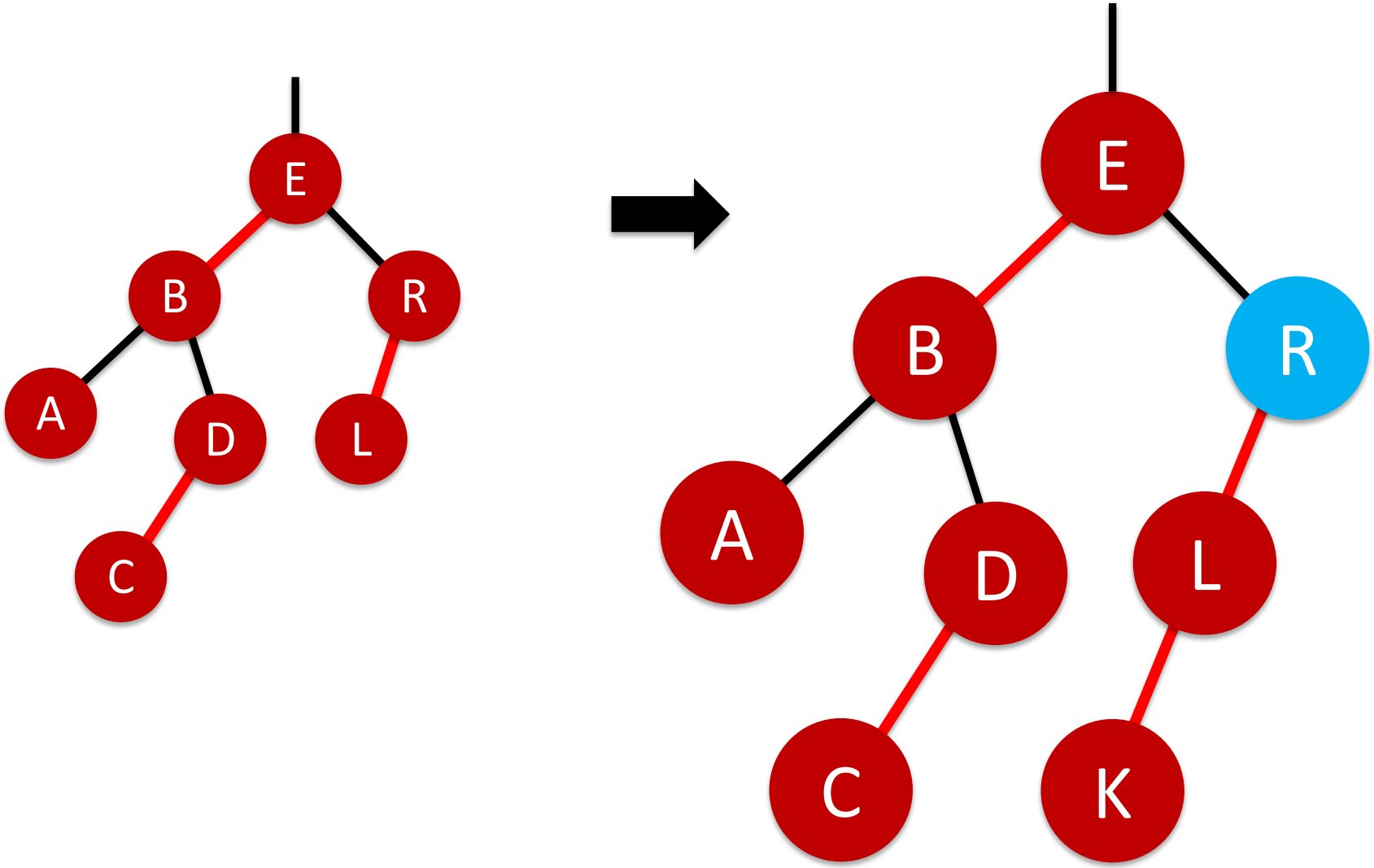

E
B
R
A
D
L
FLIP
E
B
R
A
D
L


R
E
D
B
L
A
C
K

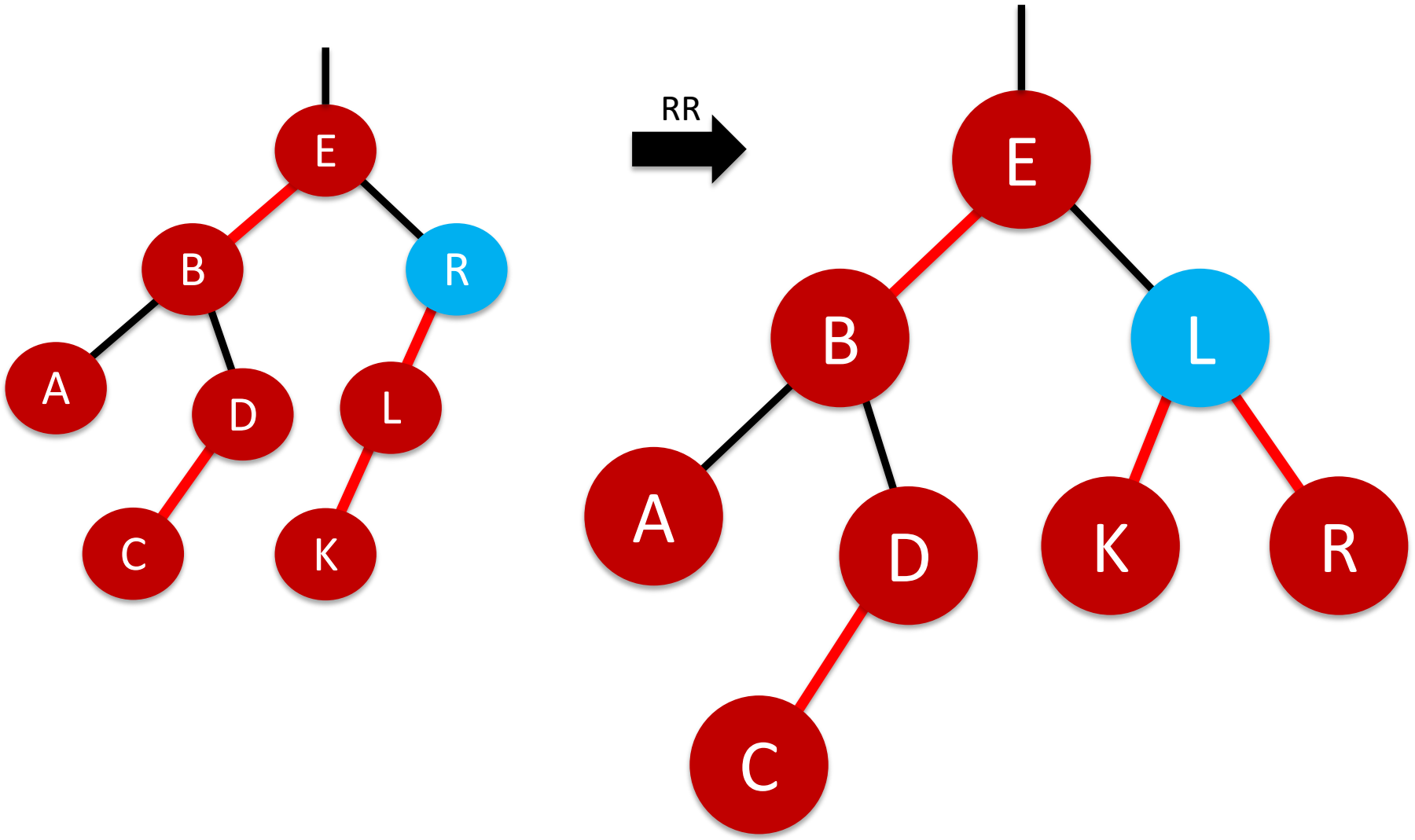

E
B
R
A
D
L


E
B
R
A
D
L
C

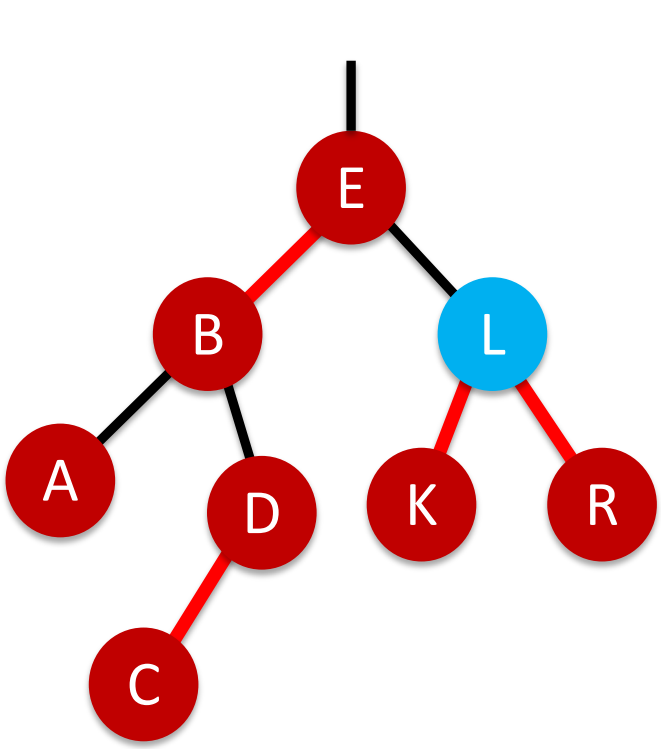

R
E
D
B
L
A
C
K

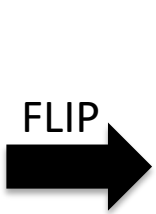

E
B
R
A
D
L
C
E
B
R
A
D
L
C
K


R
E
D
B
L
A
C
K

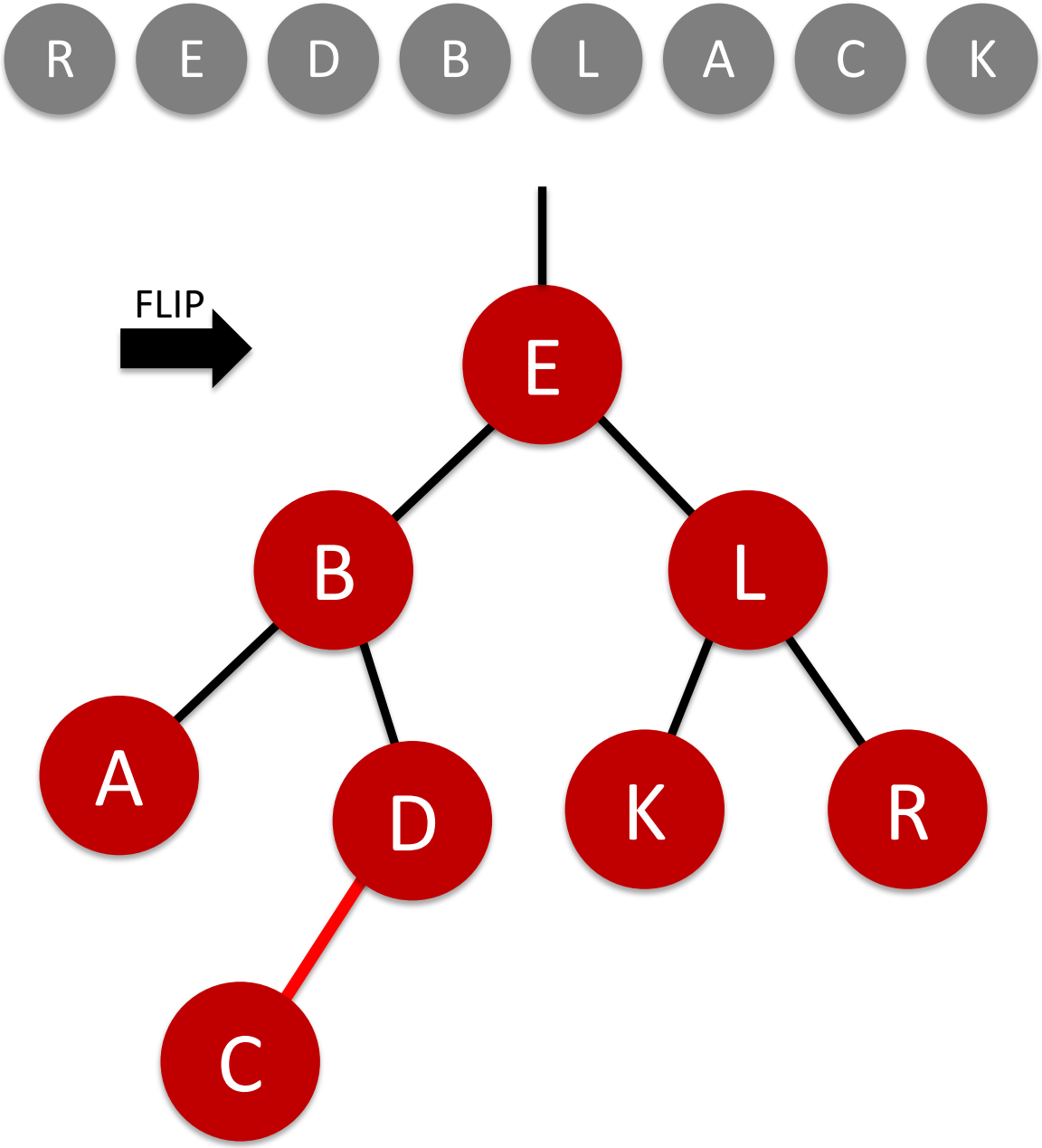

E
B
R
A
D
L
C
K
RR
E
B
L
A
D
K
R
C


R
E
D
B
L
A
C
K

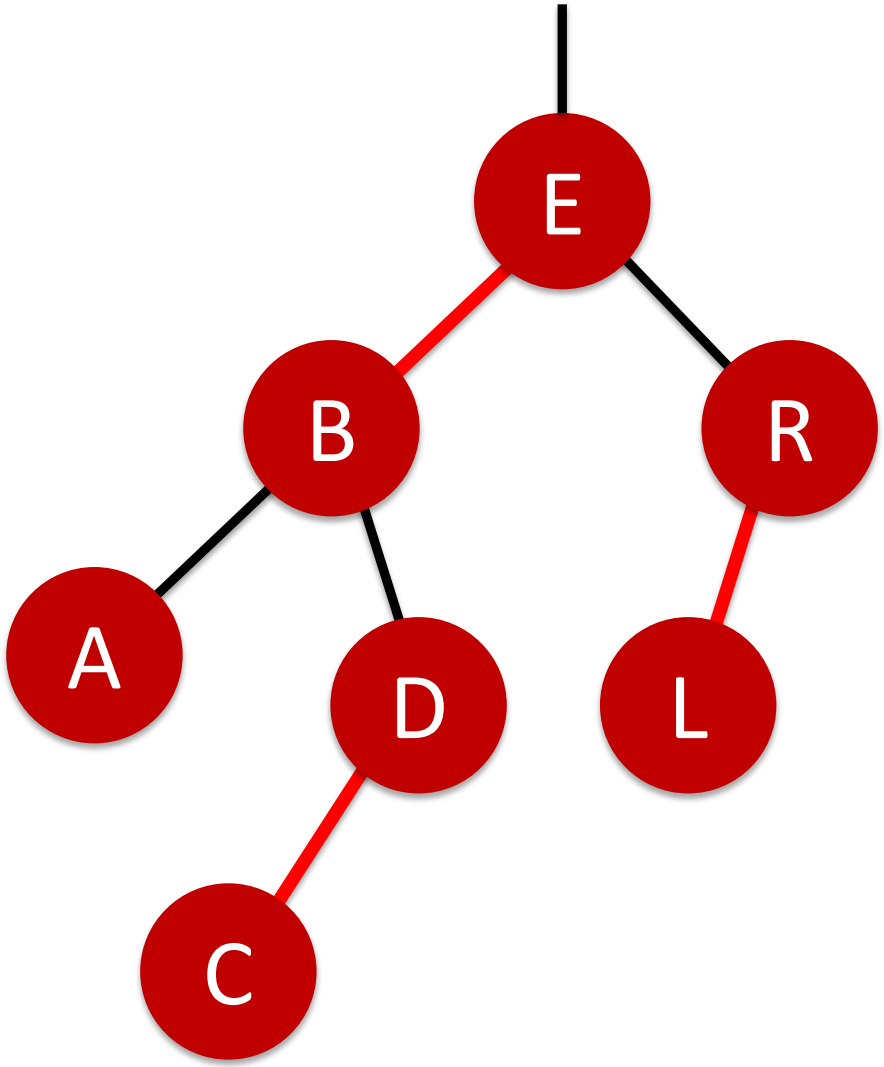

E
B
L
A
D
K
R
C


FLIP

E
B
L
A
D
K
R
C

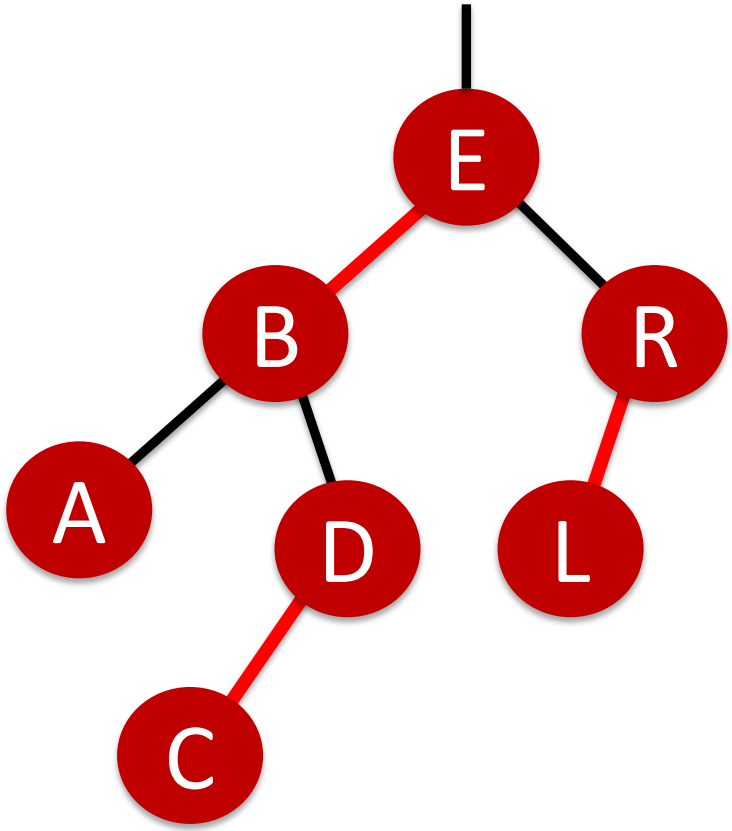

R
E
D
B
L
A
C
K
FLIP
E
B
L
A
D
K
R
C

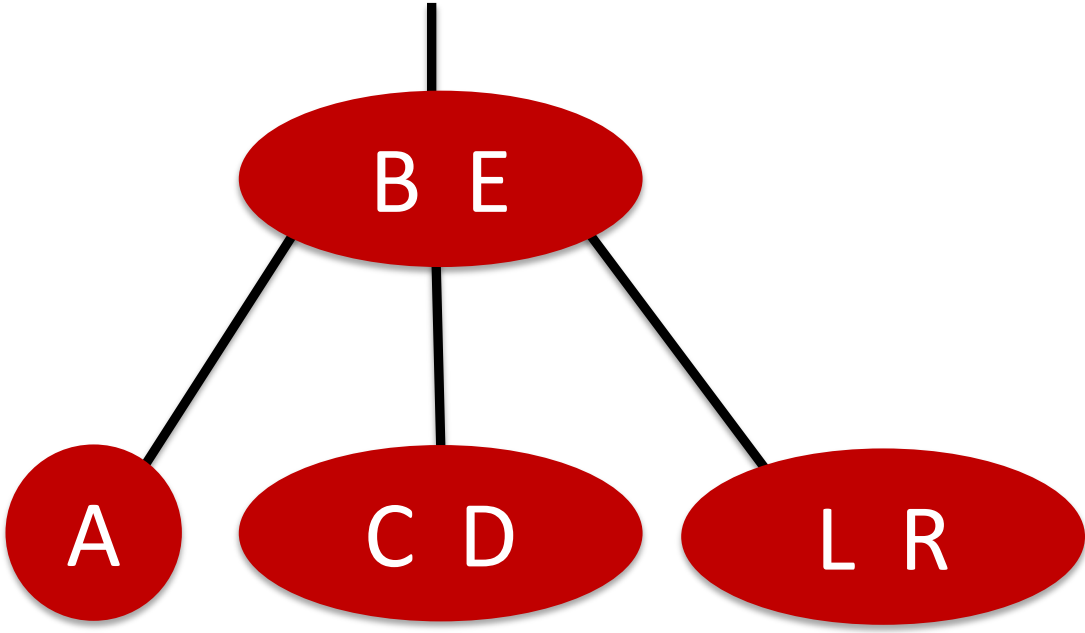
R E D B L A C K

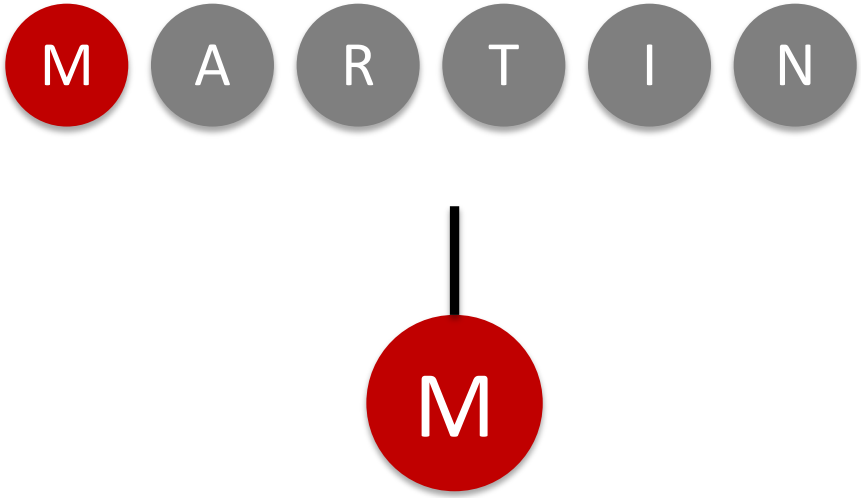


Draw this RB tree as a 2-3 tree.

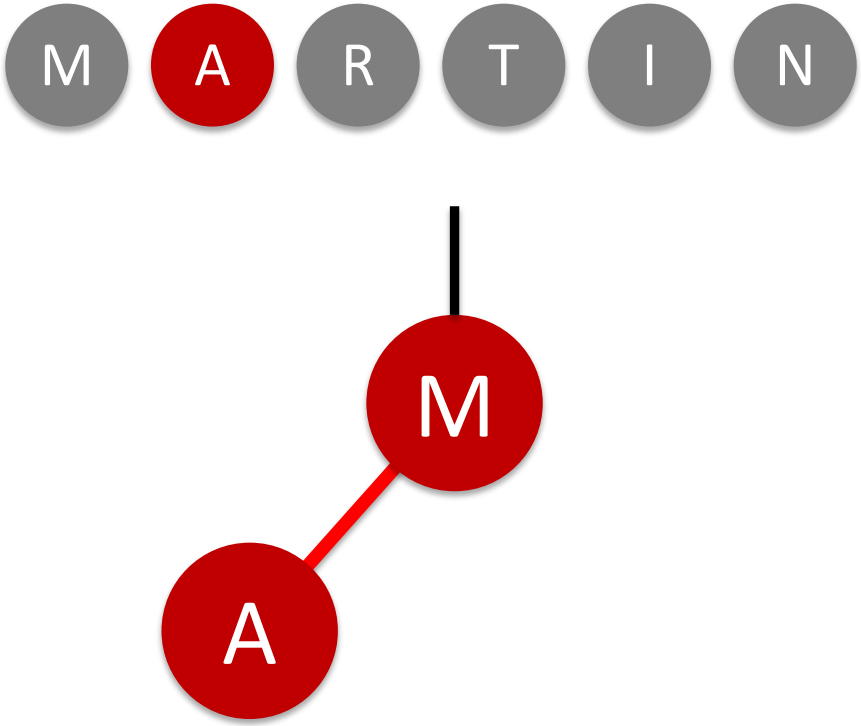

R E D B L A C K


E
B
R
A
D
L
C

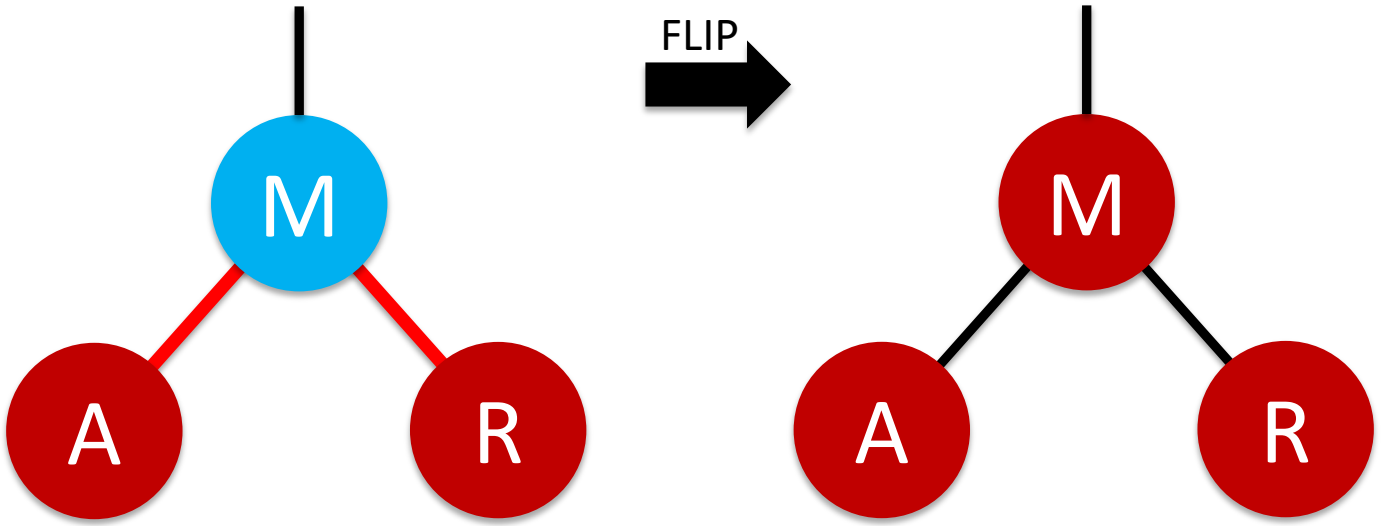

B E
A
C D
L R



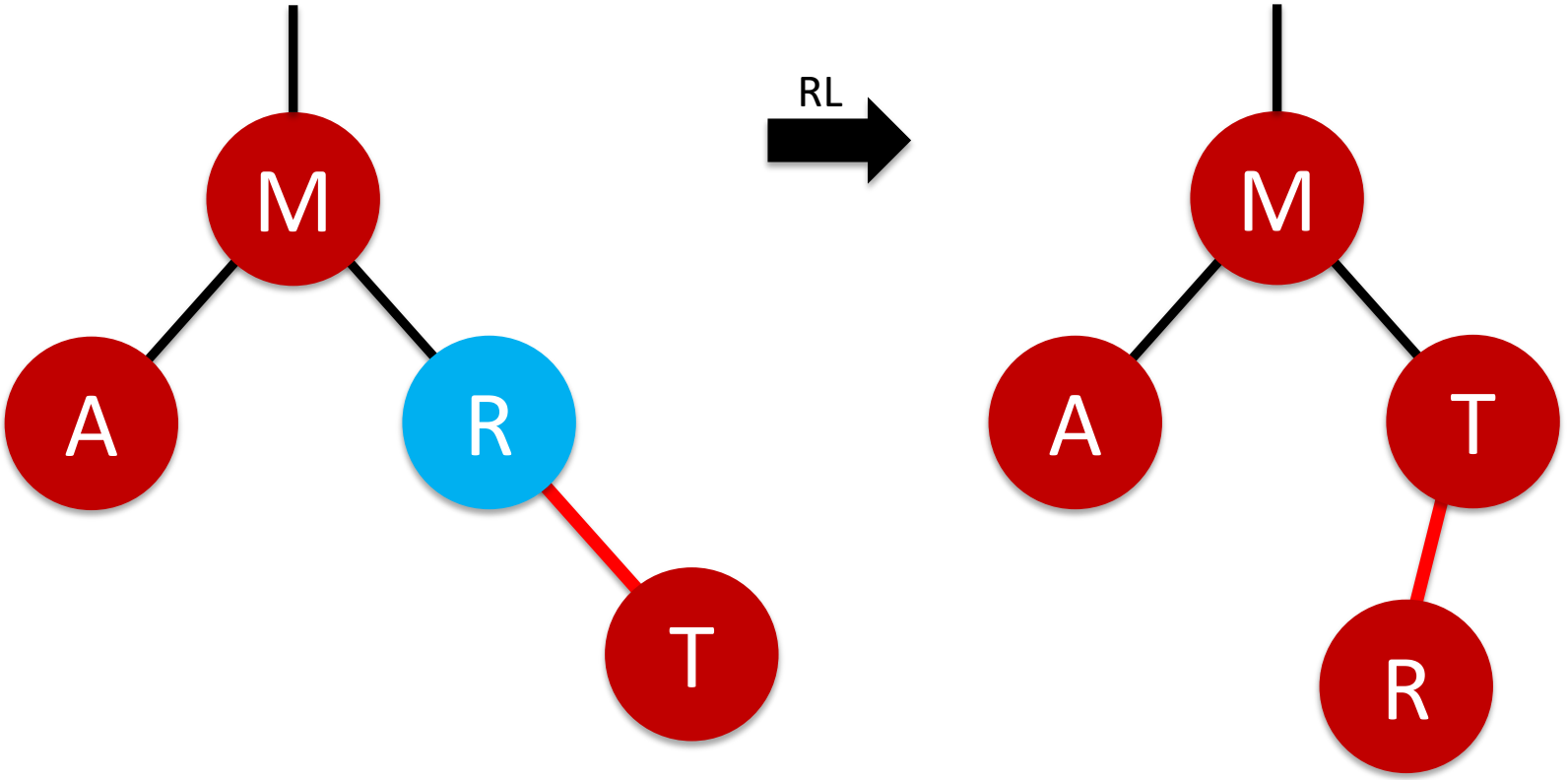
Finish this sequence.


M
A
R
T
I
N
M
A

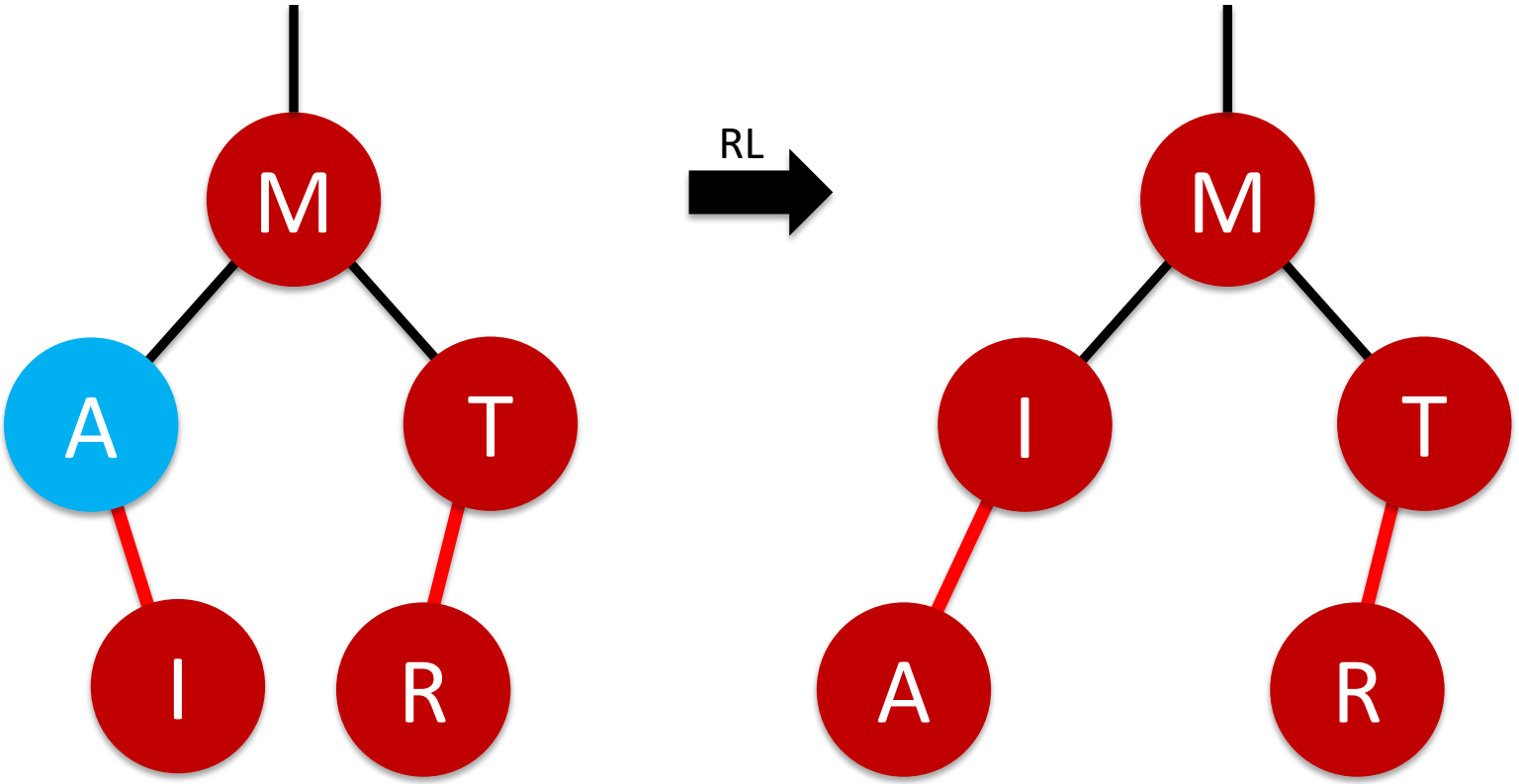

M
A
R
T
I
N


M
A
R
FLIP
M
A
R

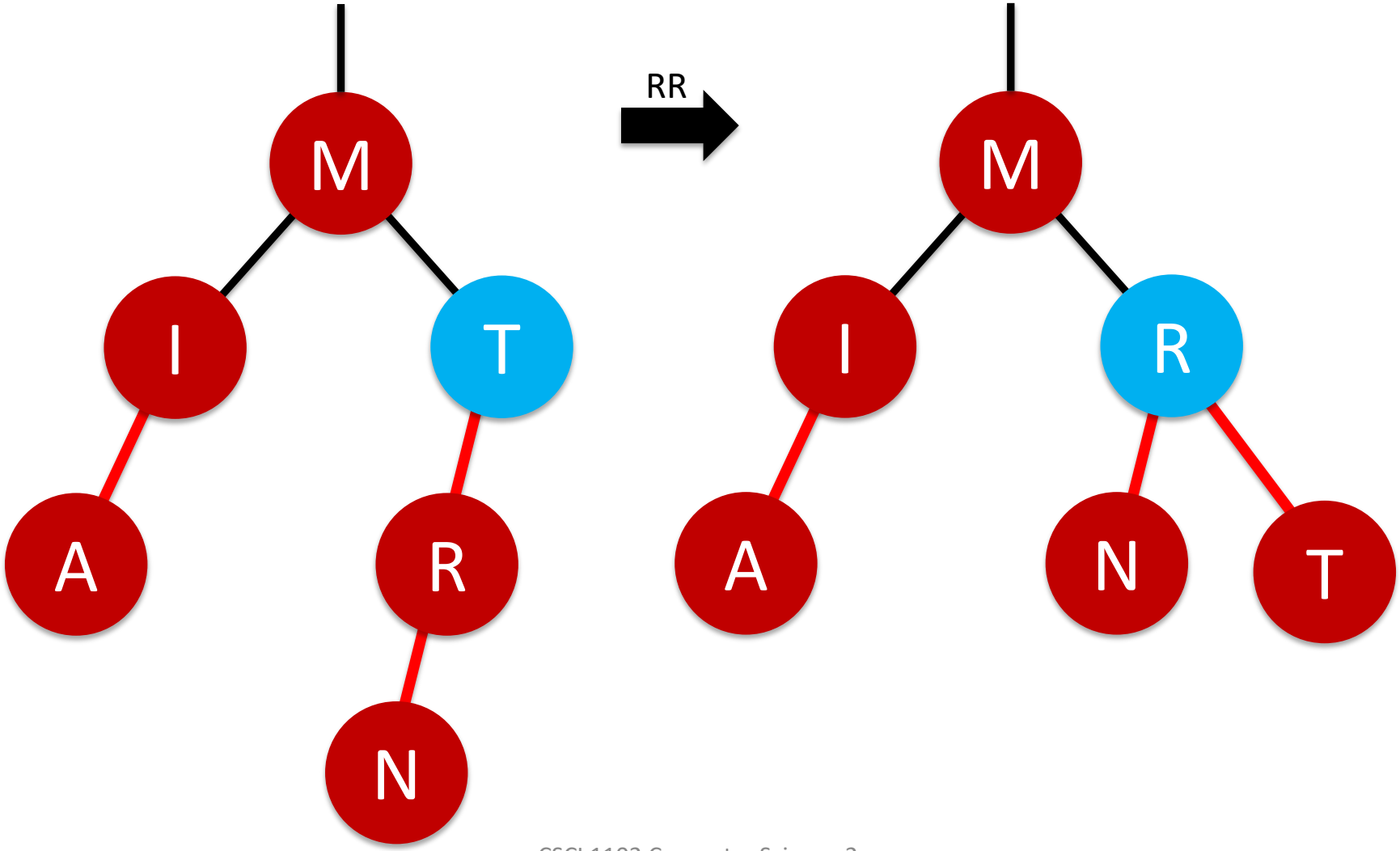

M
A
R
T
I
N

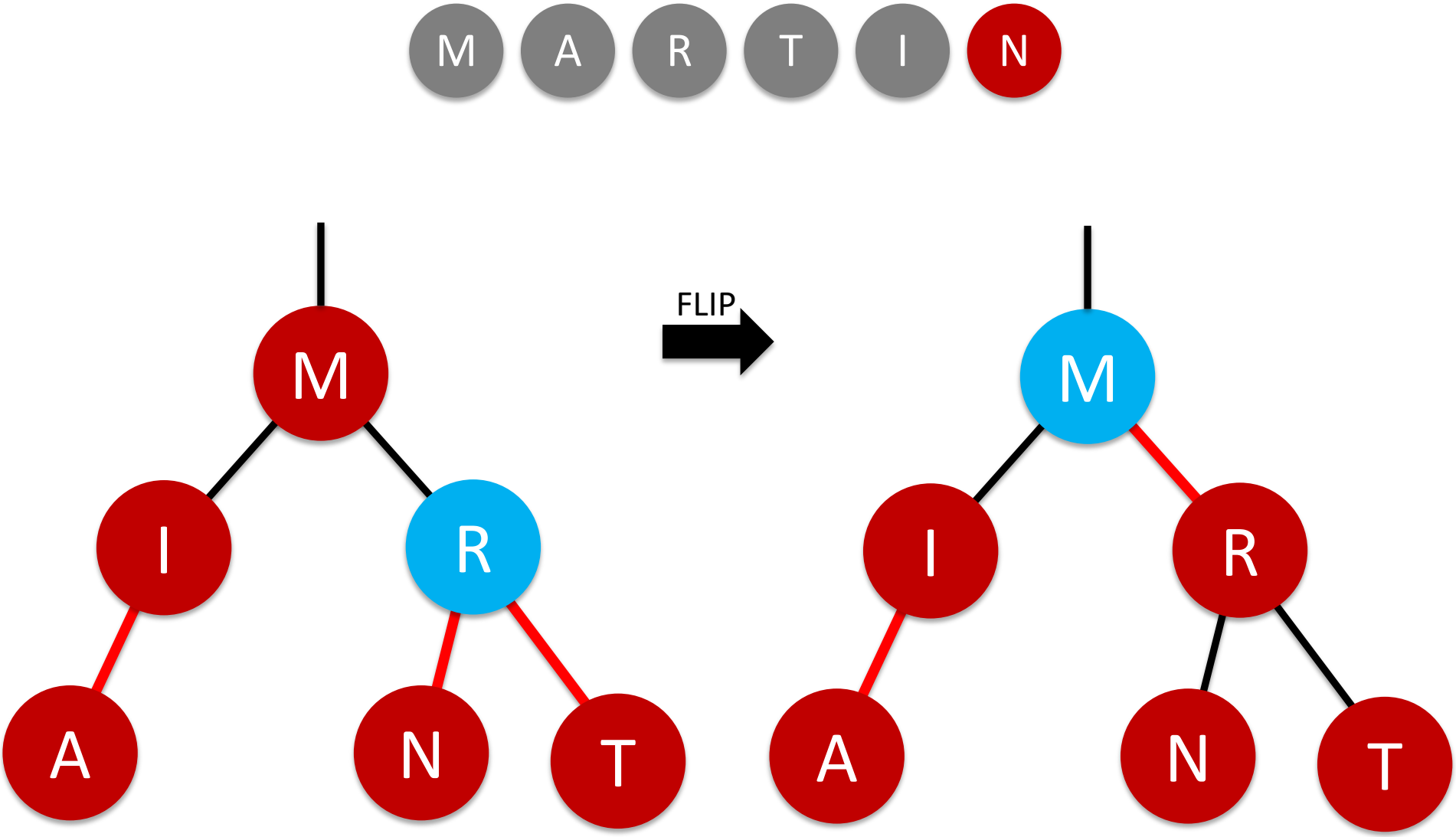

M
A
R
T
RL
M
A
T
R

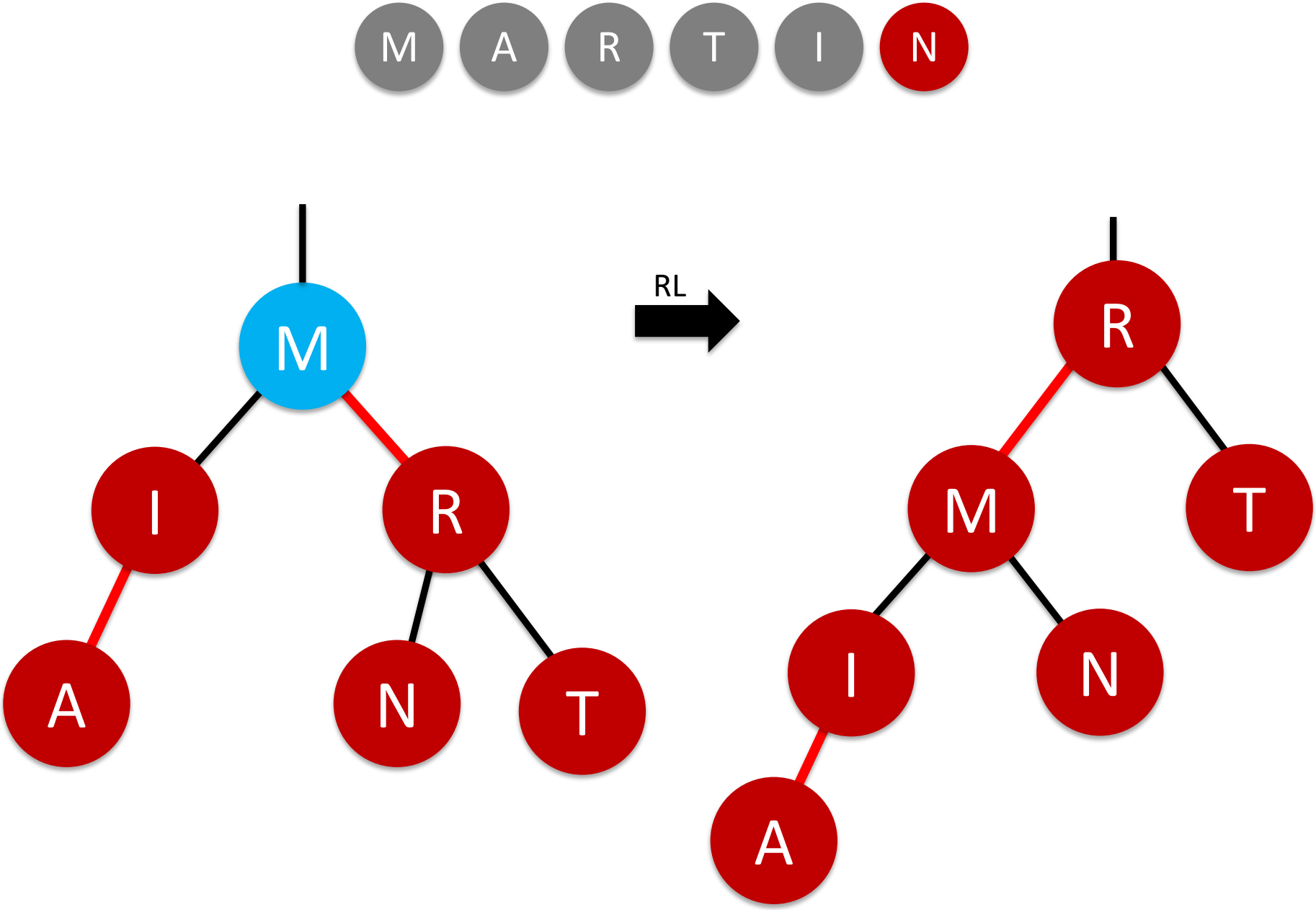

M
A
R
T
I
N

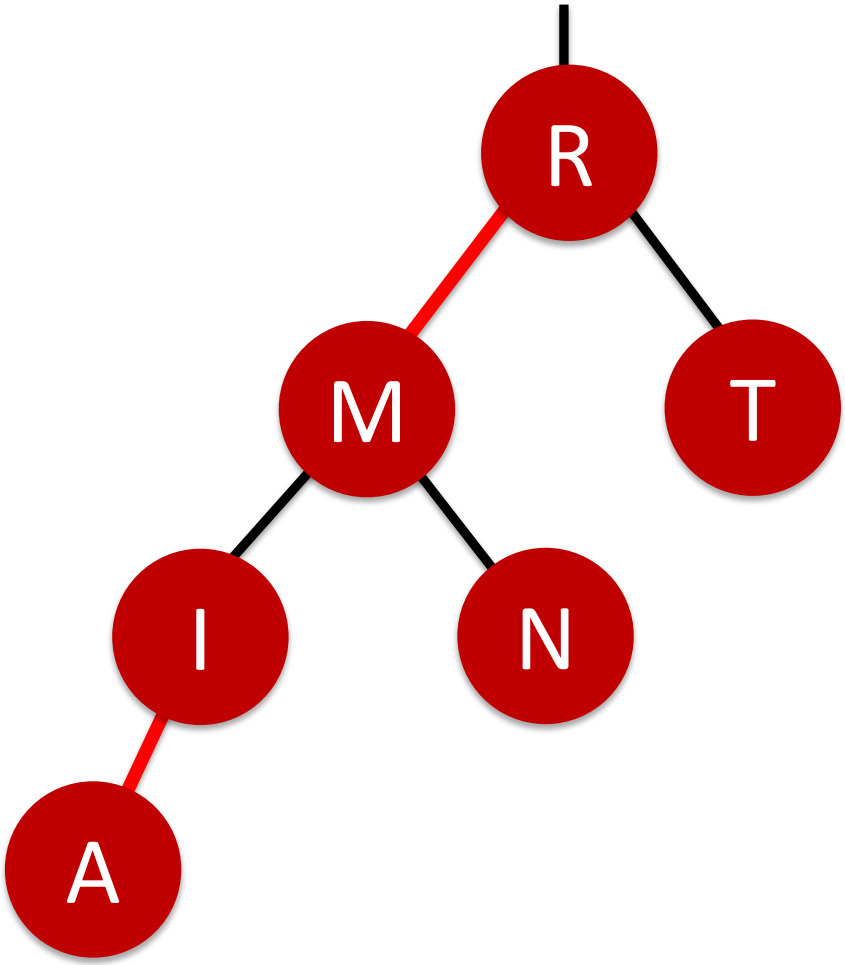

M
A
T
I
R
RL
M
I
T
A
R


M
A
R
T
I
N


M
I
T
A
R
N
RR
M
I
R
A
N
T

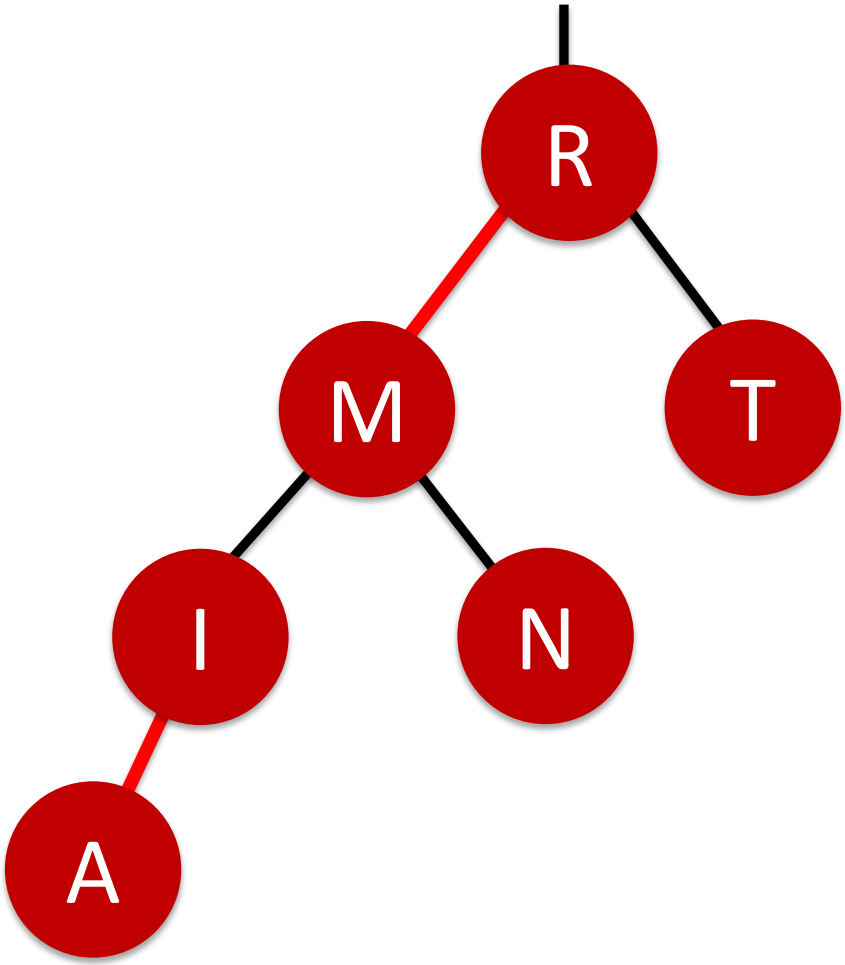

M
A
R
T
I
N
FLIP
M
I
R
A
N
T
M
I
R
A
N
T

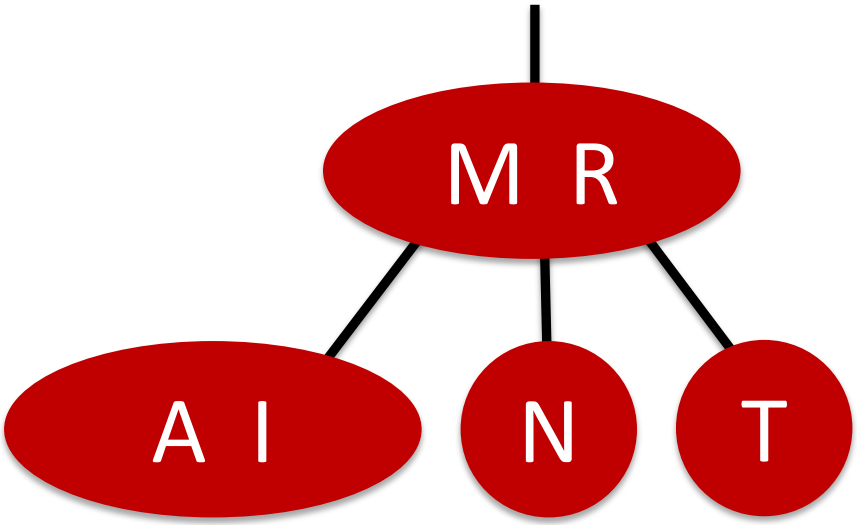

M A R T I N
M
I
R
A
N
T
RL
R
M
T
I
N
A

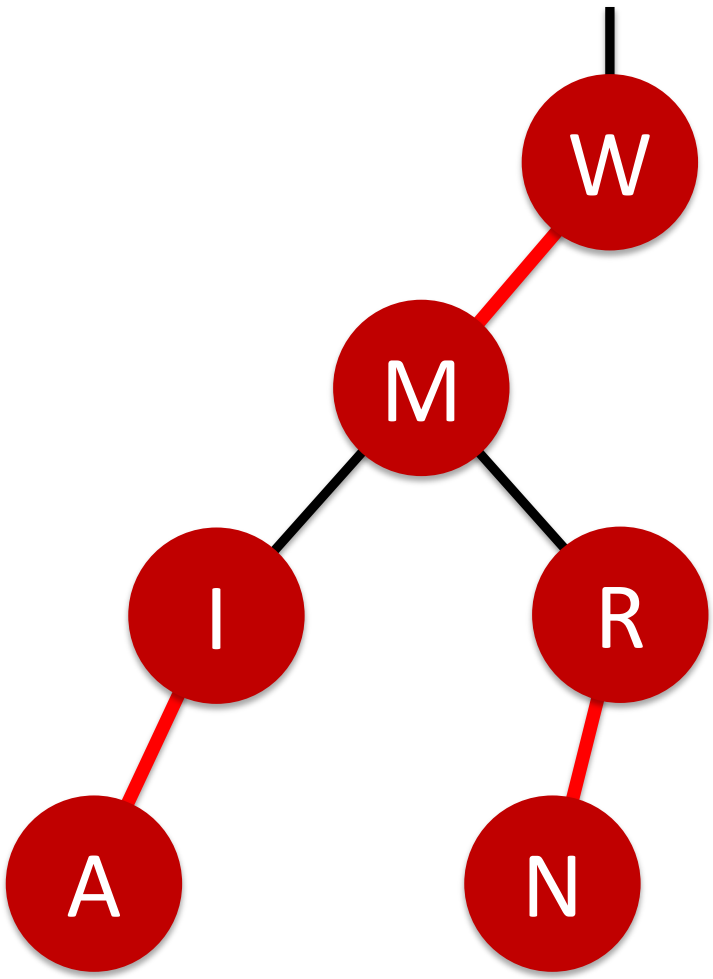


Draw this RB tree as a 2-3 tree.

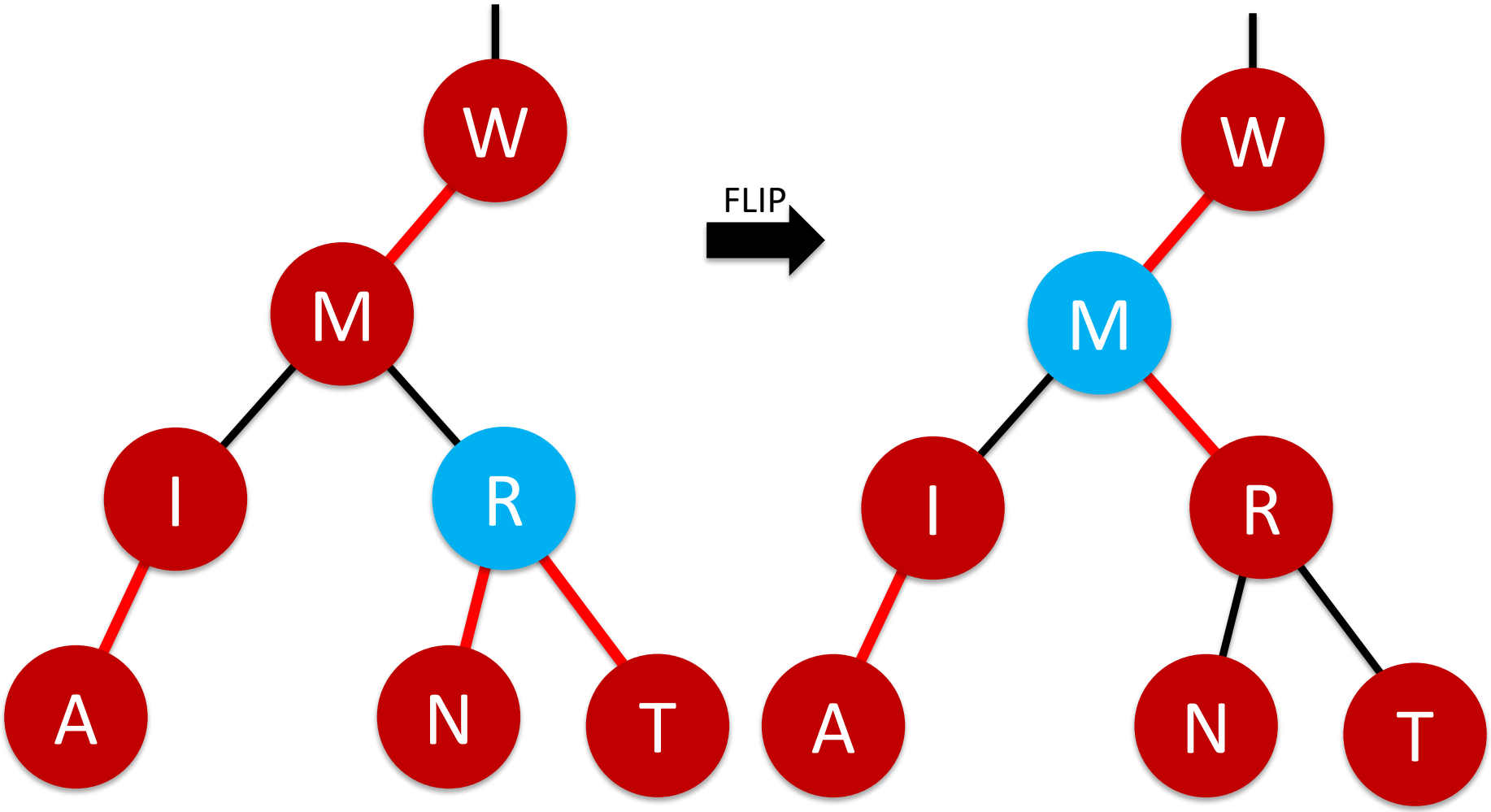

M
A
R
T
I
N

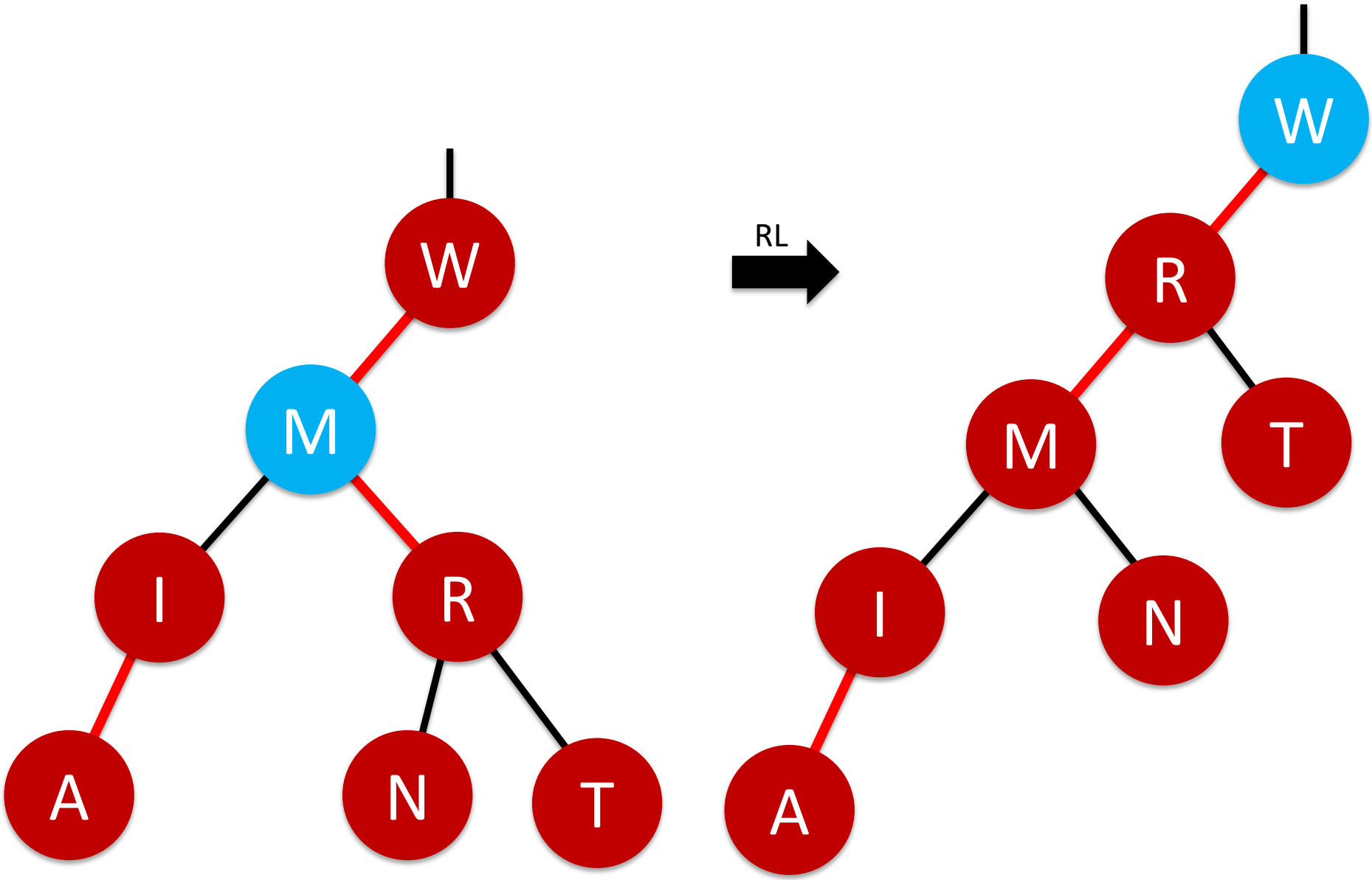

R
M
T
I
N
A

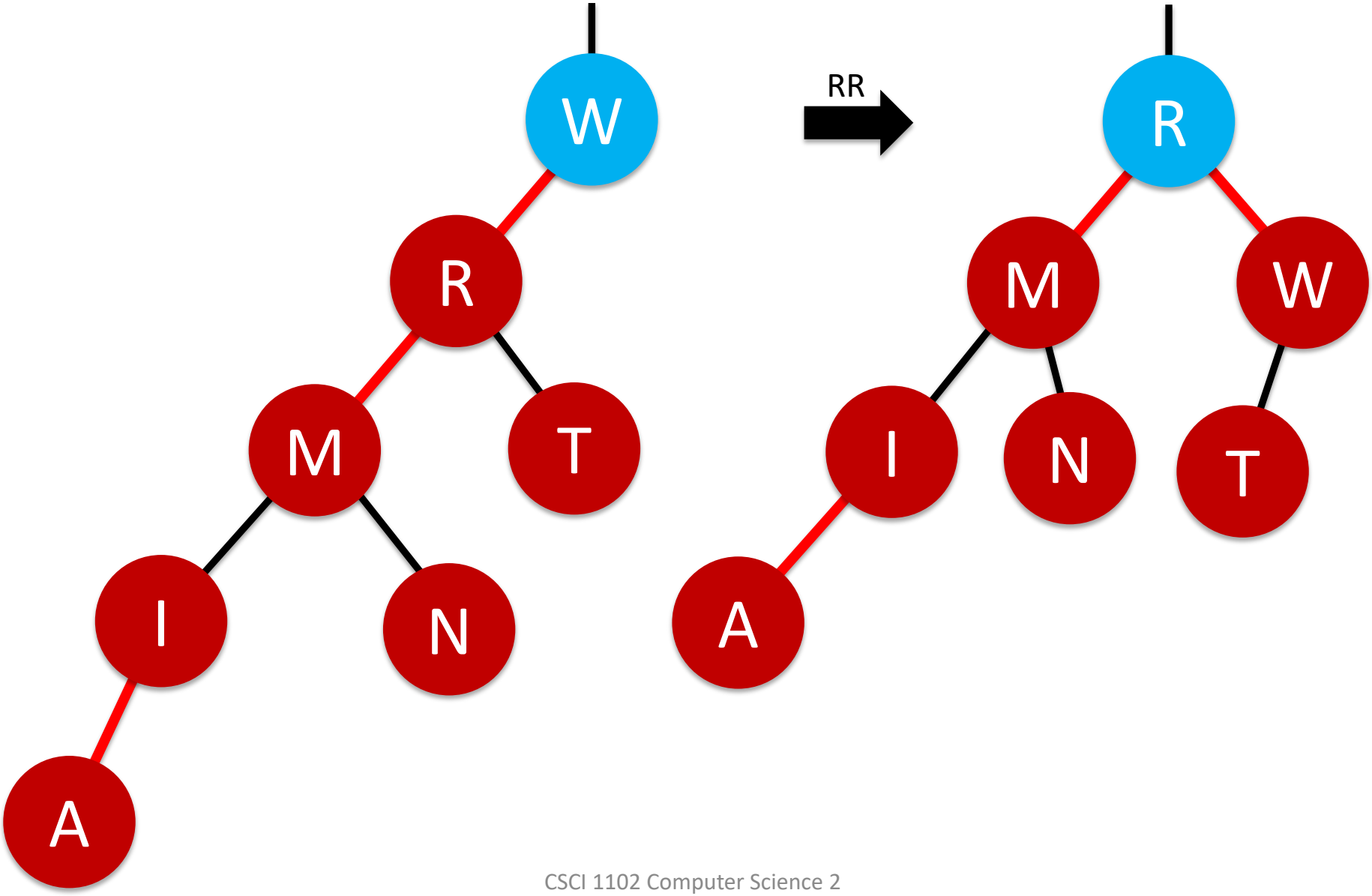

M R
A I
N
T

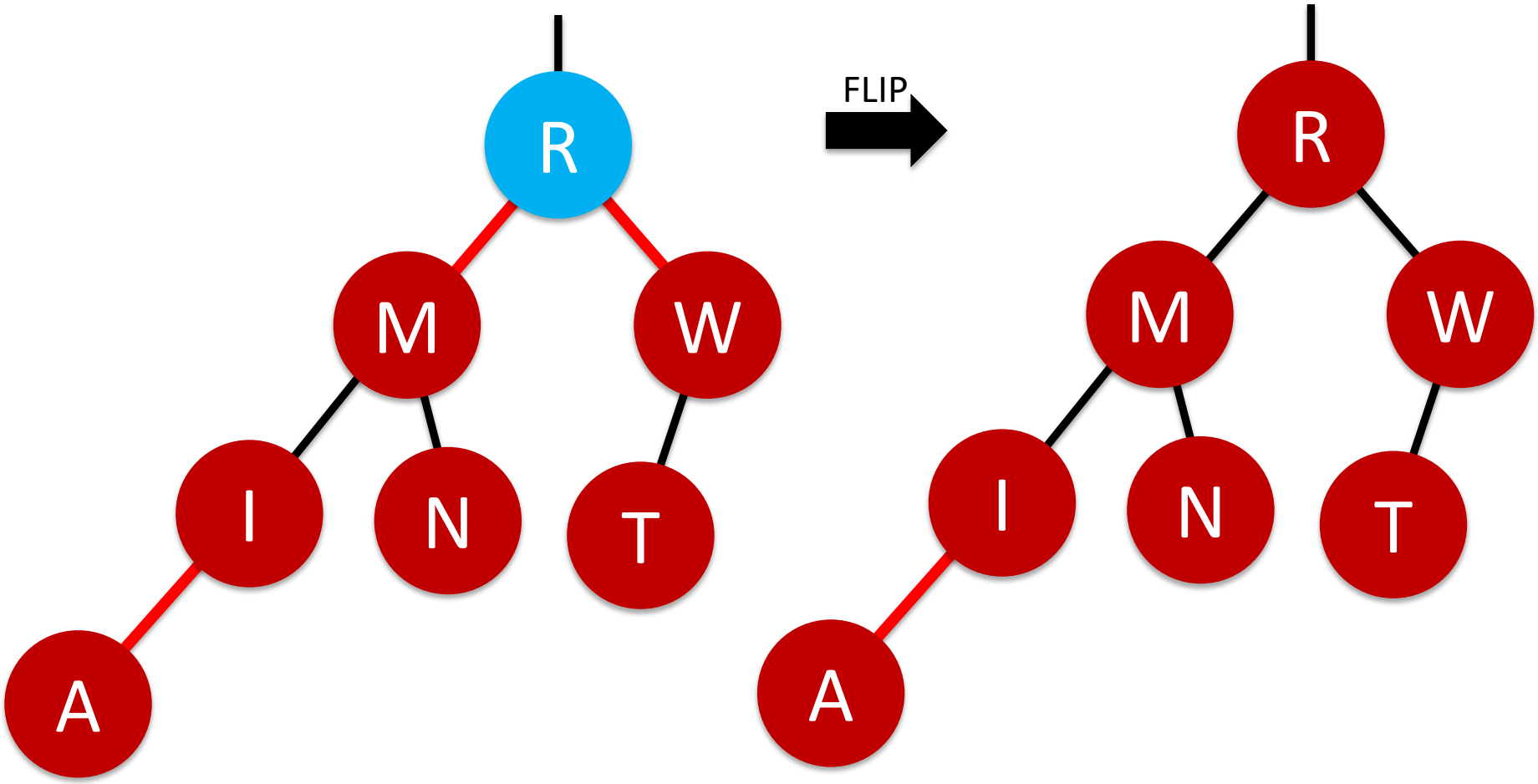


Insert the letter T into this RB tree.


W
M
I
R
A
N
T
FLIP
W
M
I
R
A
N
T

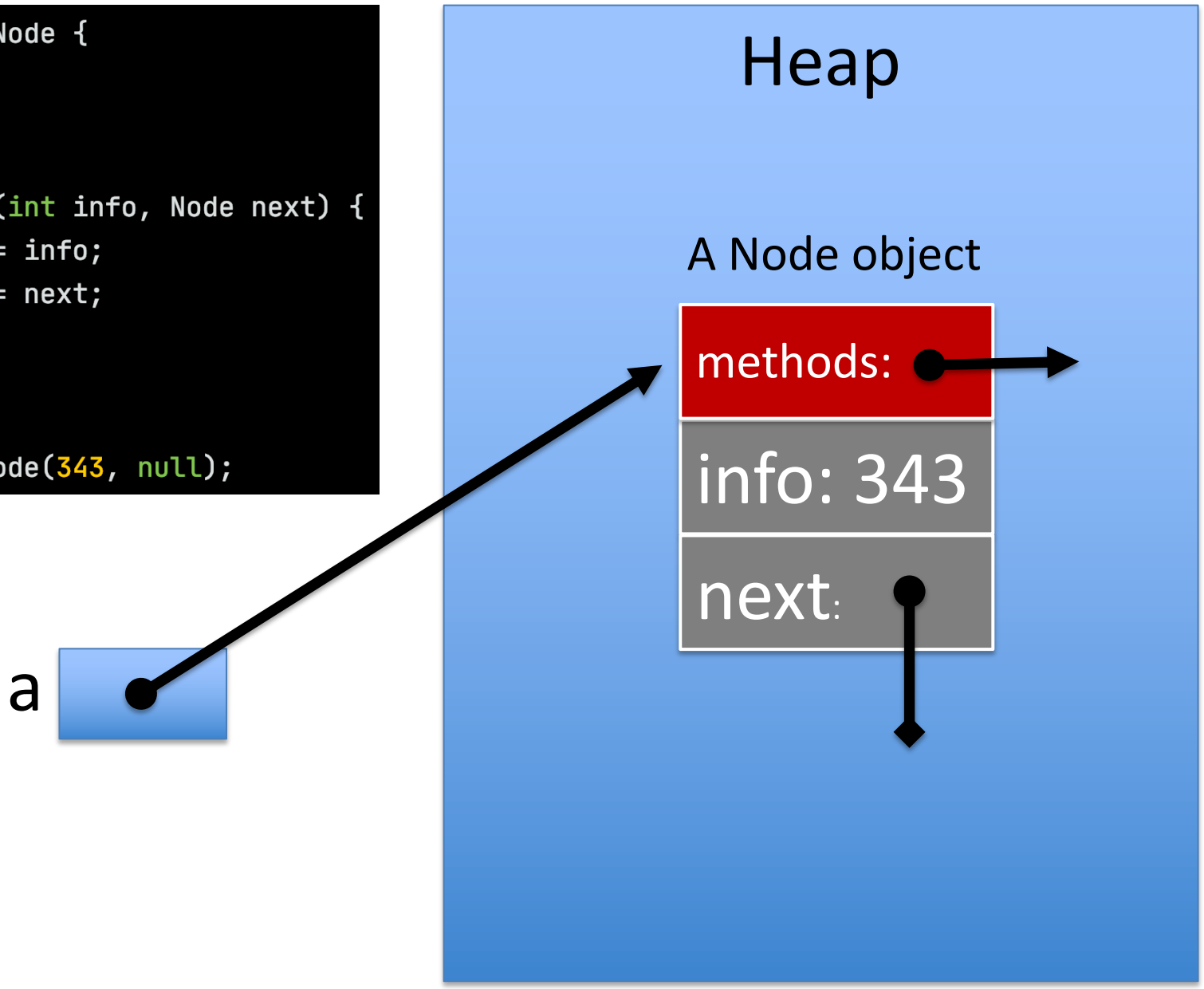

W
M
I
R
A
N
T
RL
W
R
M
T
I
N
A


W
R
M
T
I
N
A
RR
R
M
W
I
N
T
A

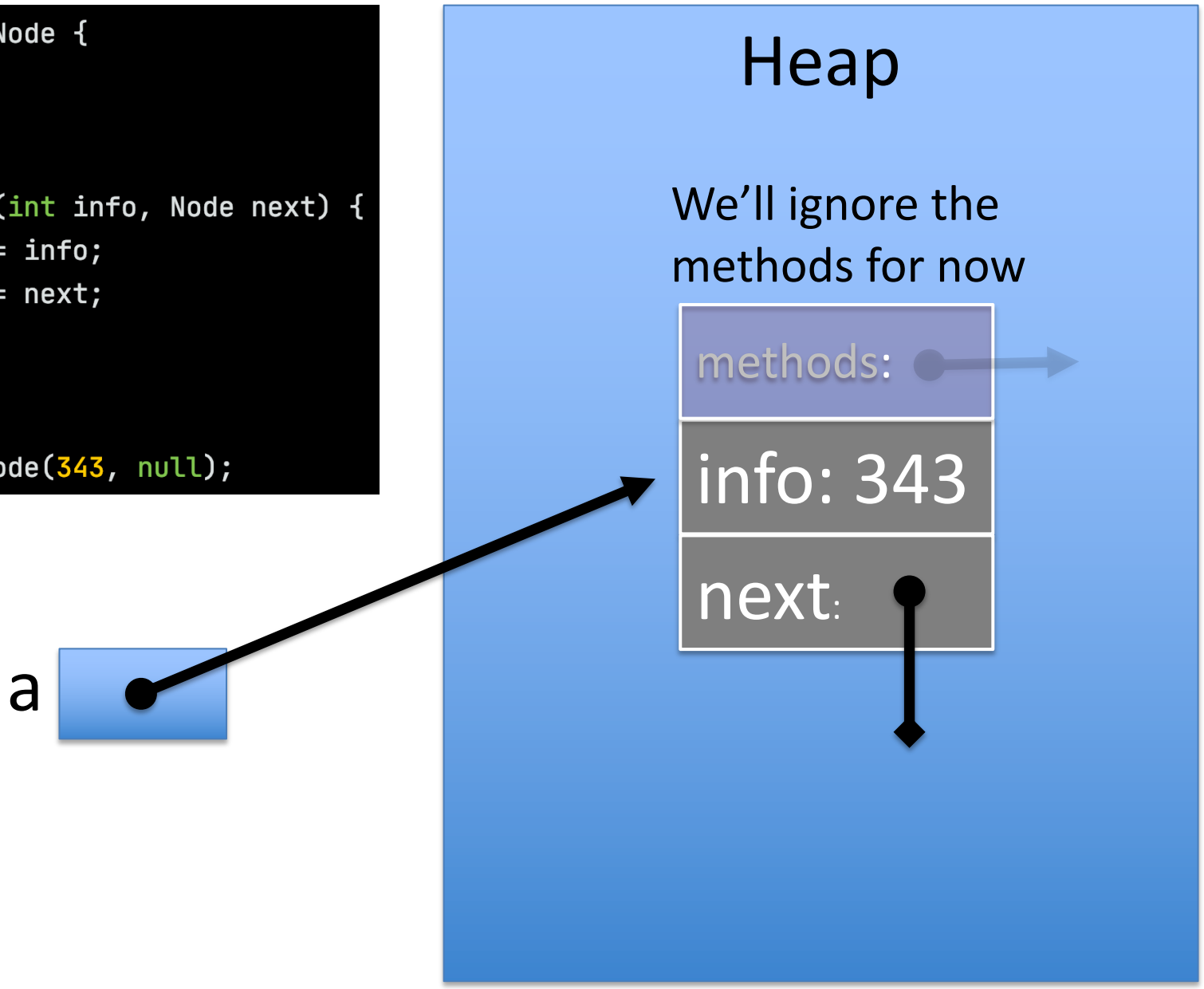

R
M
W
I
N
T
A
FLIP
R
M
W
I
N
T
A

Review of this Storage Diagrams

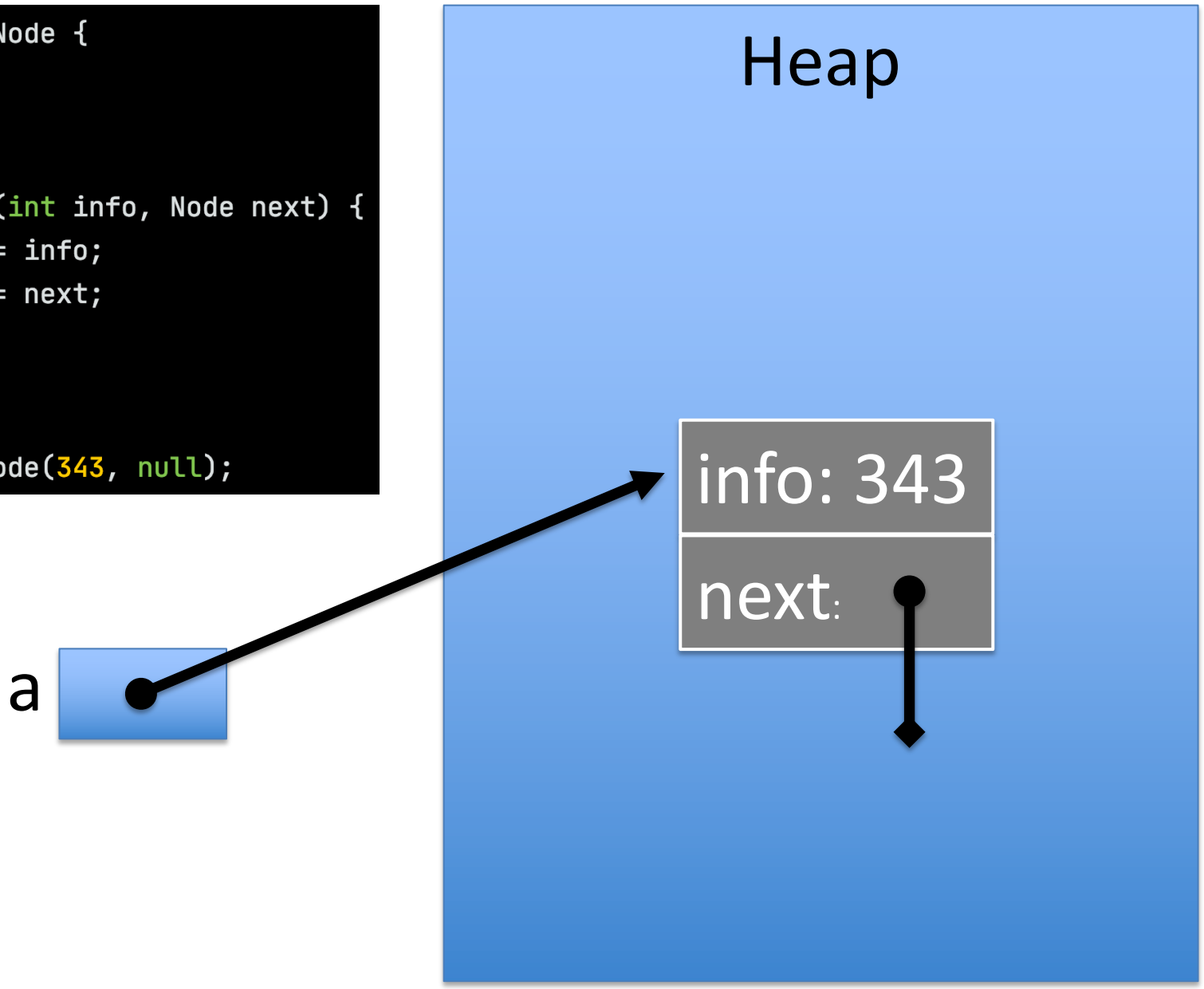
```java
private class Node {
  int info;
  Node next;

  private Node(int info, Node next) {
    this.info = info;
    this.next = next;
  }
}

Node a = new Node(343, null);
```



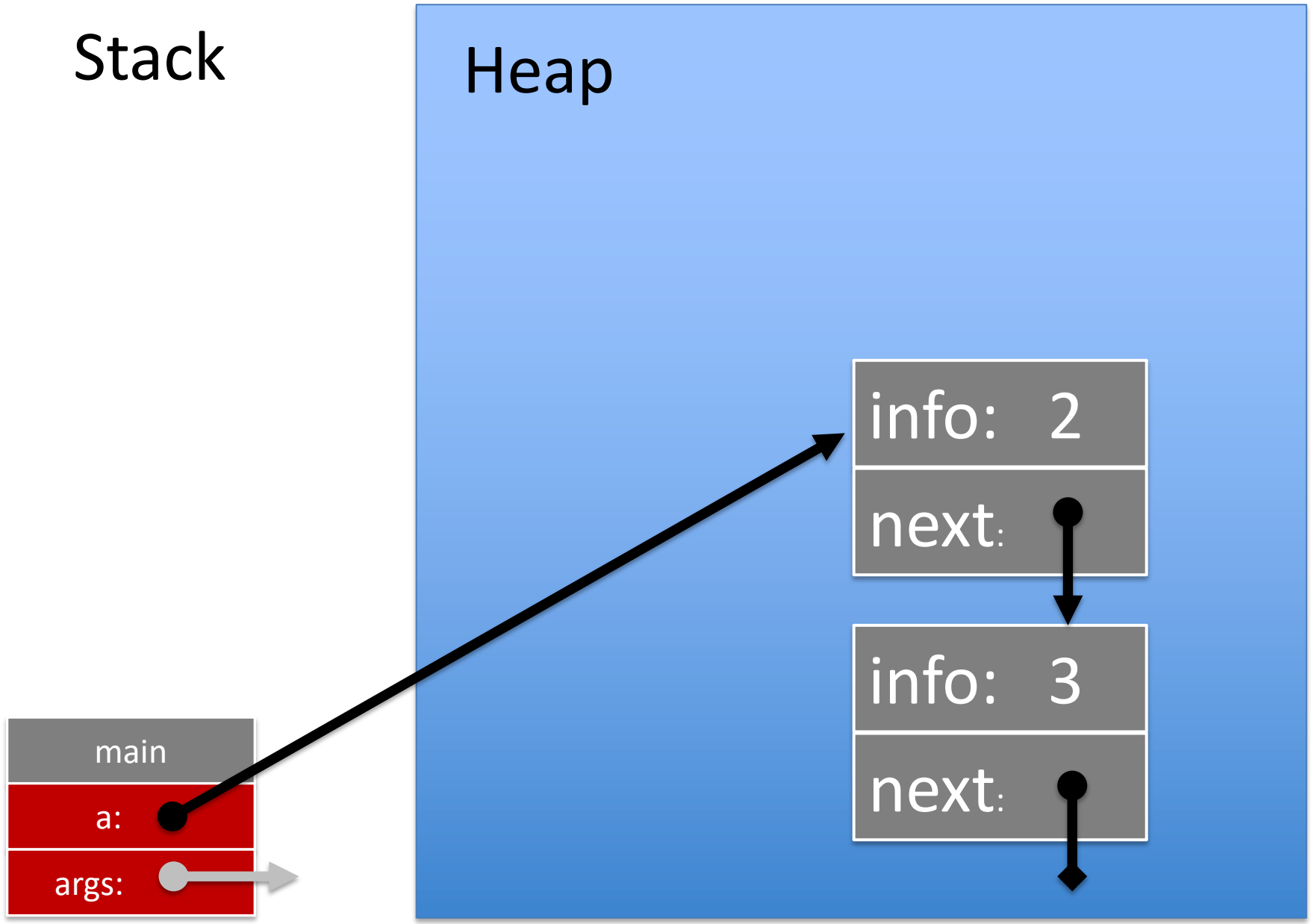
```
private class Node {
  int info;
  Node next;

  private Node(int info, Node next) {
    this.info = info;
    this.next = next;
  }
}

Node a = new Node(343, null);
```

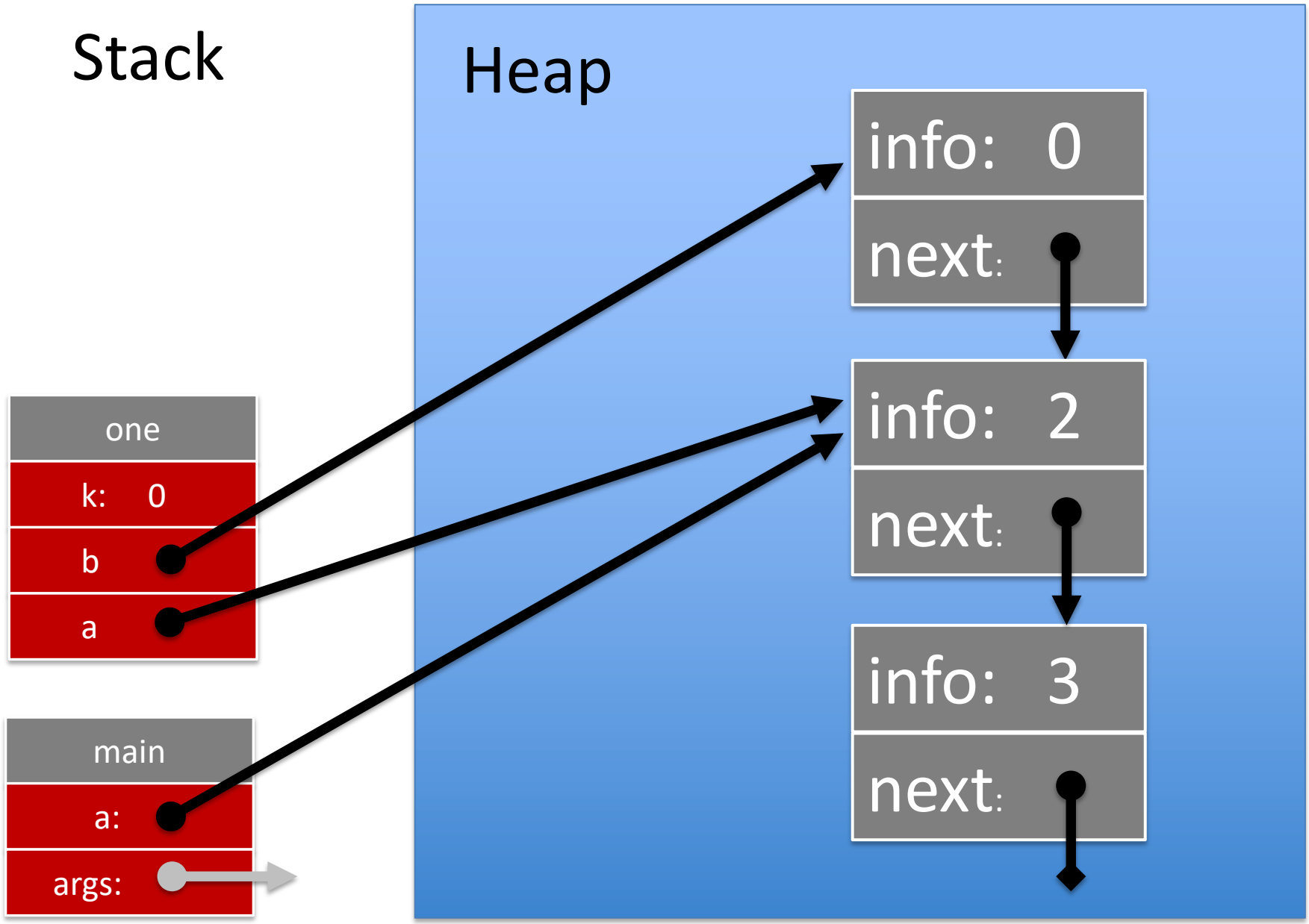


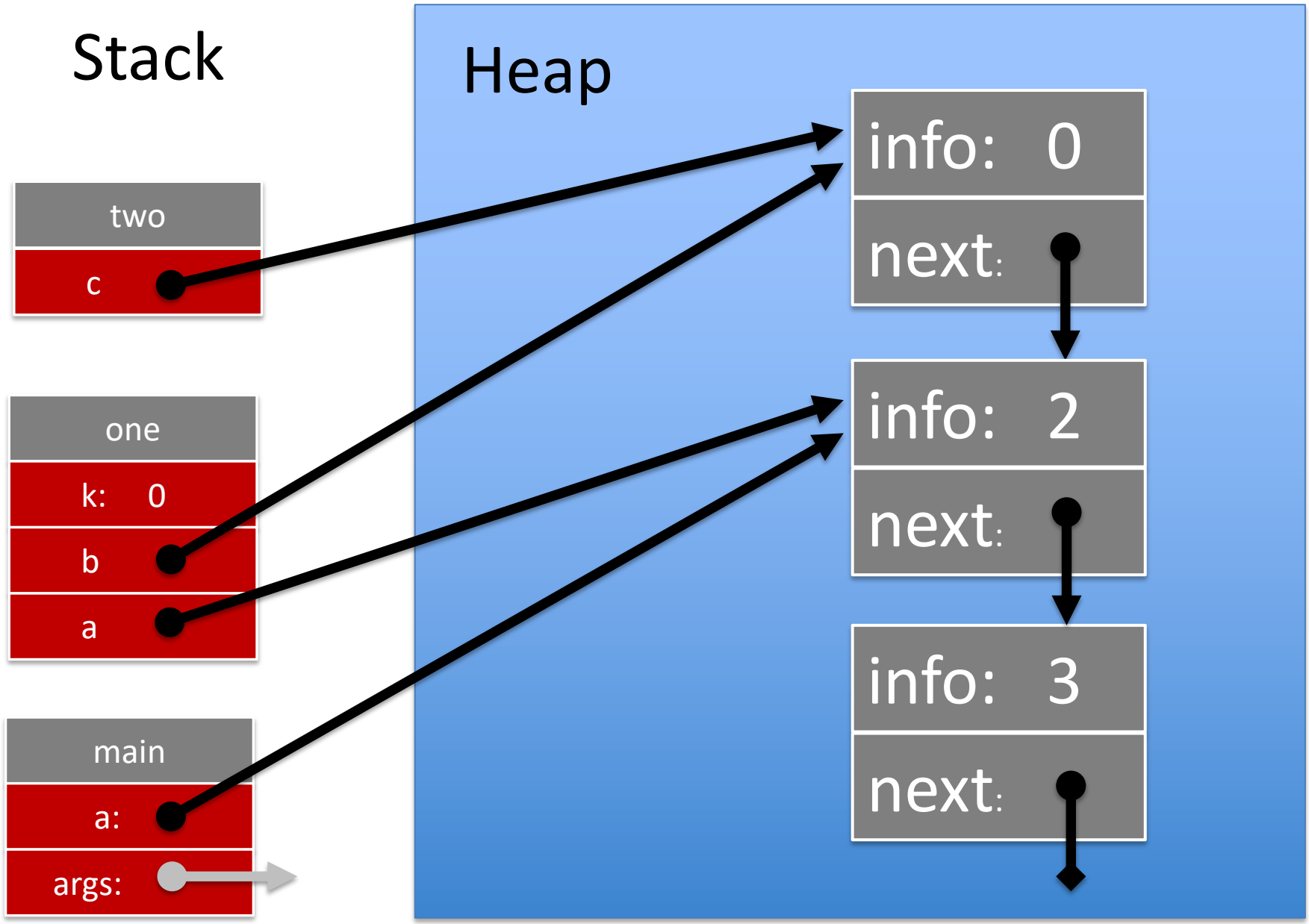
```java
private class Node {
  int info;
  Node next;

  private Node(int info, Node next) {
    this.info = info;
    this.next = next;
  }
}

Node a = new Node(343, null);
```

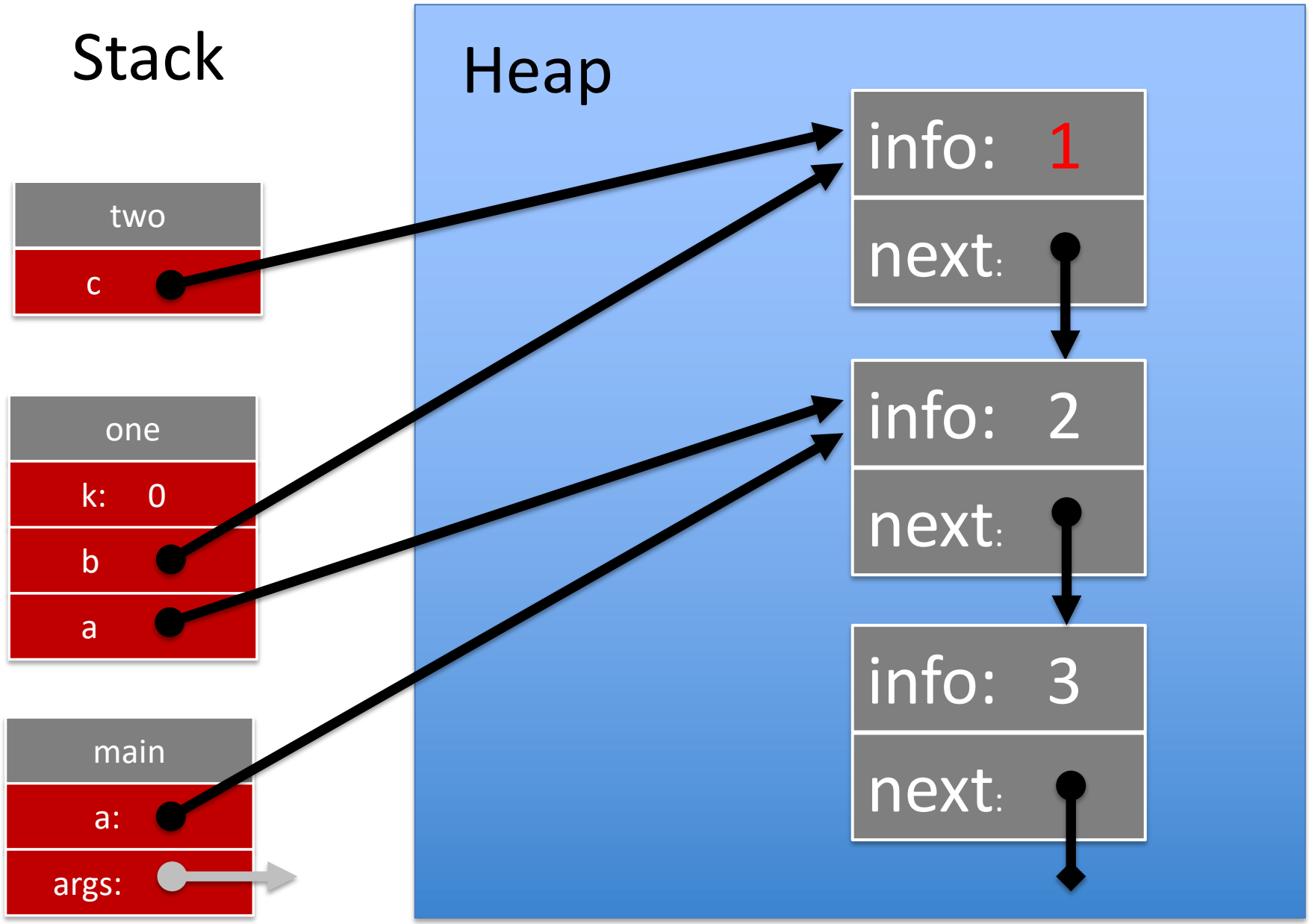


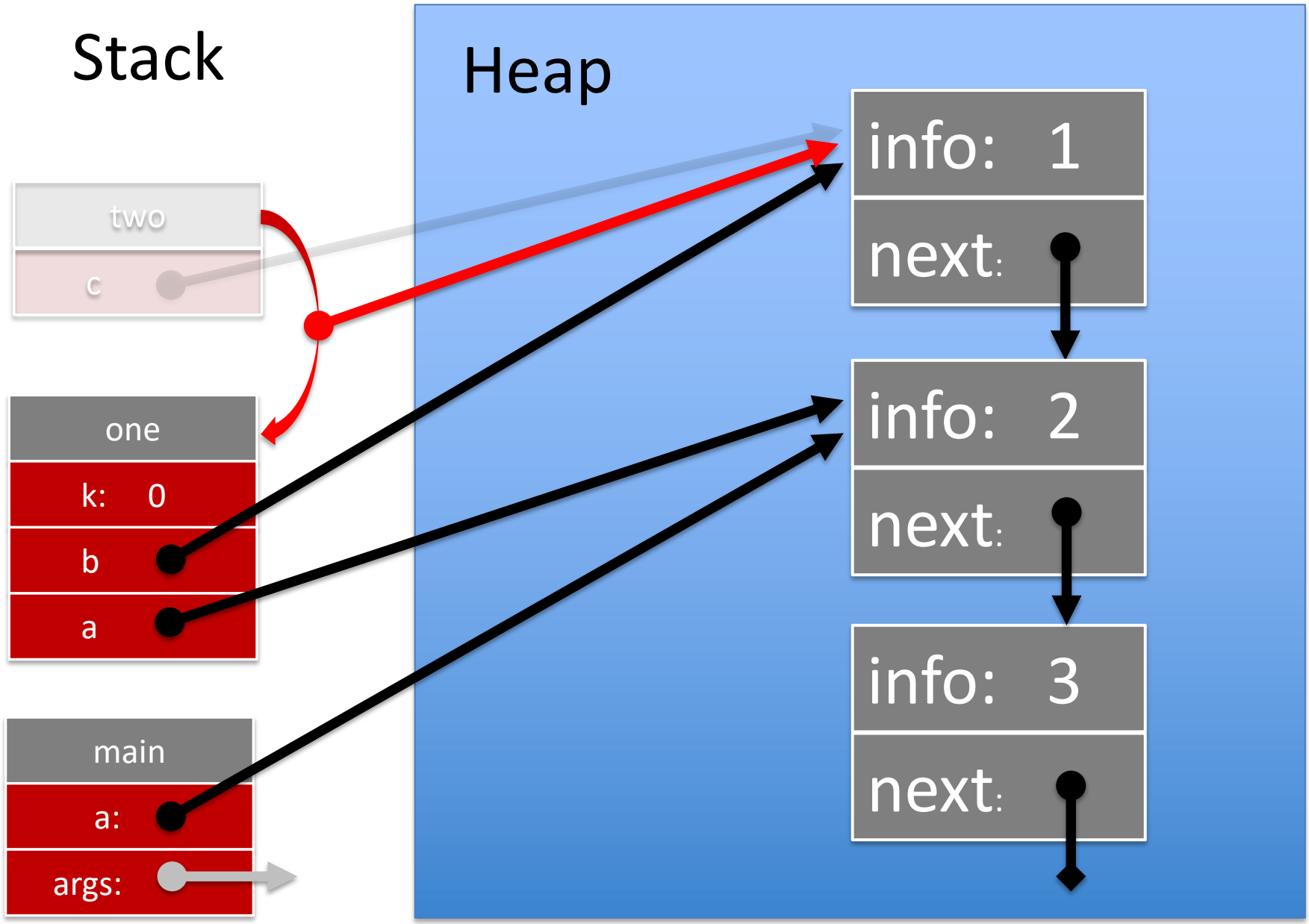
```
private static Node two(Node c) {
  c.info = c.info + 1;
  return c;
}

private static Node one(int k, Node a) {
  Node b = new Node(k, a);
  return two(b);
}

public static void main(String[] args) {
  Node a = new Node(2, new Node(3, null));
  a = one(0, a);
}
```

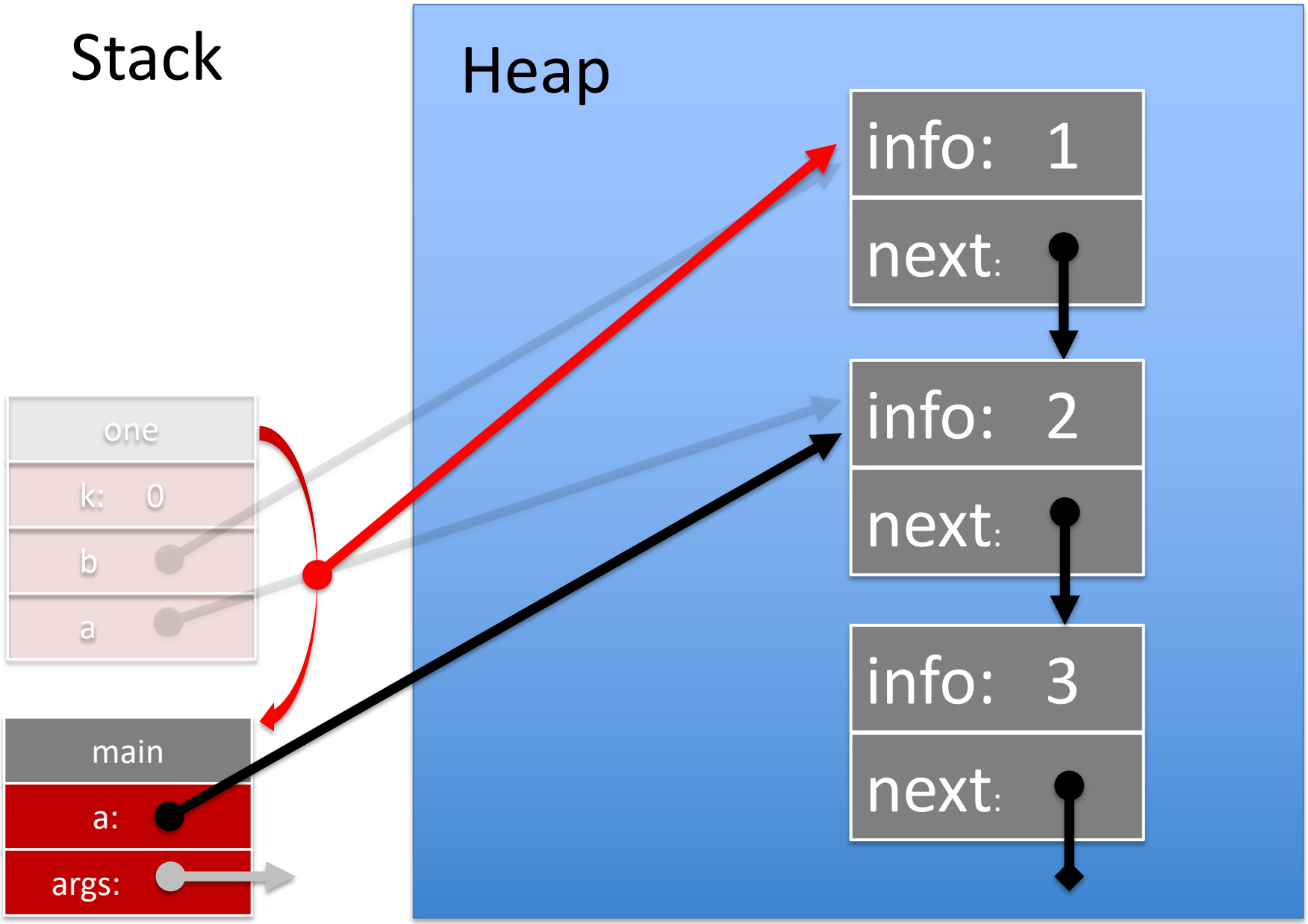

Stack
Heap
info: 2
next:
info: 3
next:
main
a:
args:

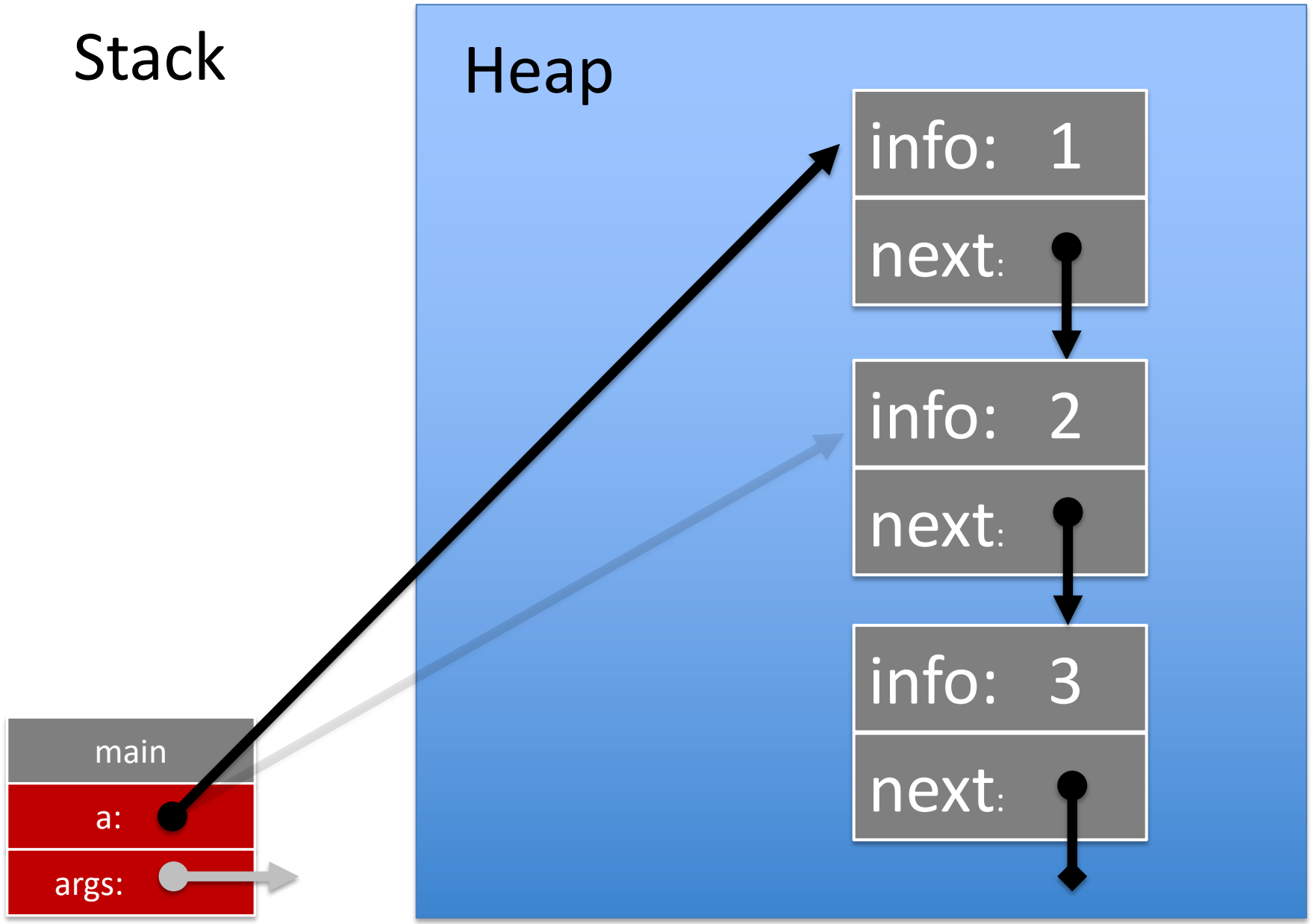

Stack
Heap
info: 0
next:
info: 2
next:
info: 3
next:
one
k: 0
b
a
main
a:
args:


Stack
Heap
two
c
one
k: 0
b
a
main
a:
args:
info: 0
next:
info: 2
next:
info: 3
next:


Stack
Heap
two
c
one
k: 0
b
a
main
a:
args:
info: 1
next:
info: 2
next:
info: 3
next:


Stack
Heap
two
c
one
k: 0
b
a
main
a:
args:
info: 1
next:
info: 2
next:
info: 3
next:

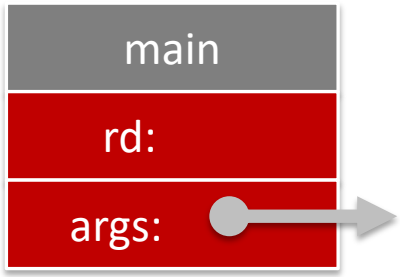

Stack
Heap
info: 1
next:
info: 2
next:
info: 3
next:
one
k: 0
b
a
main
a:
args:

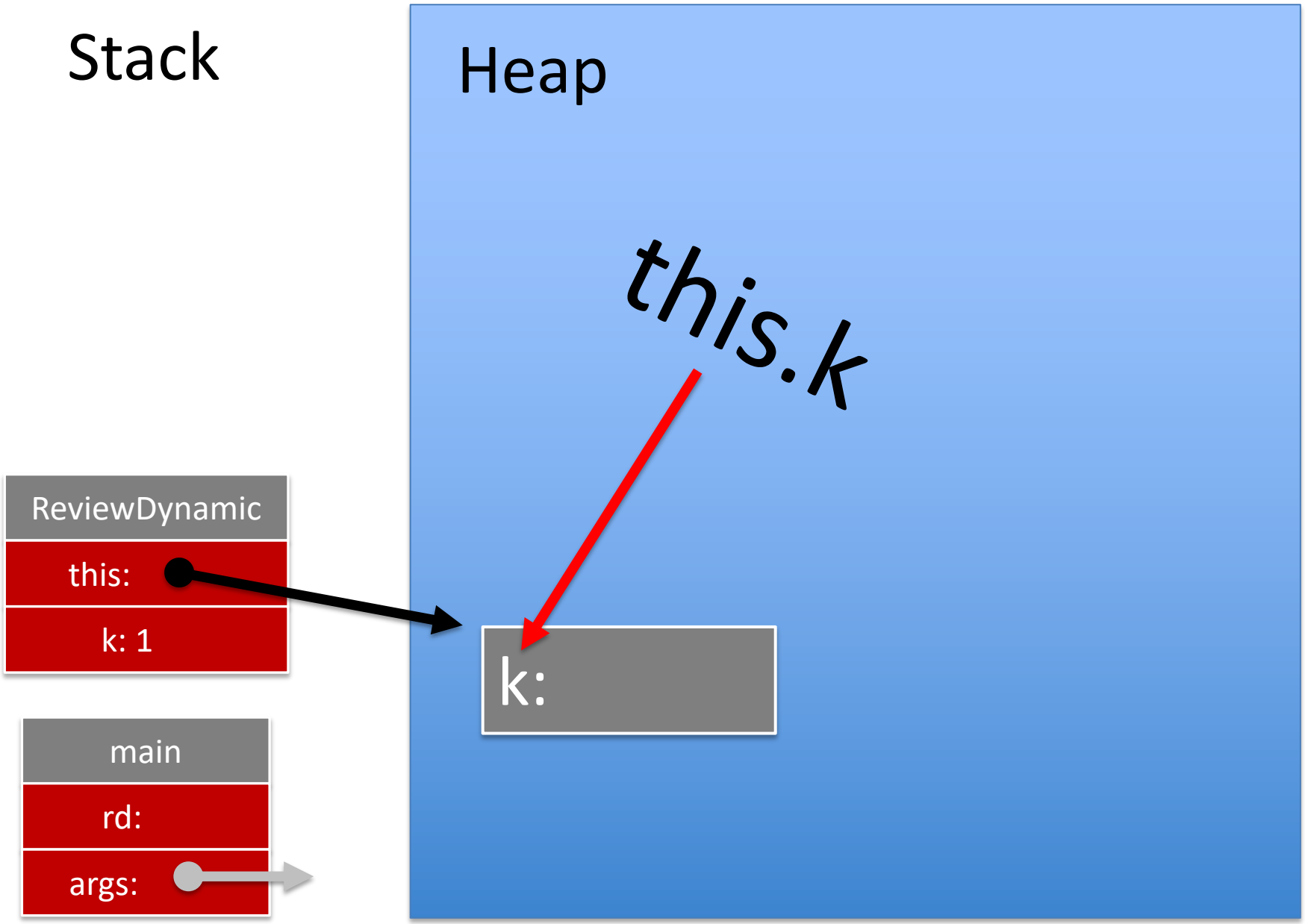

Stack
Heap
info: 1
next:
info: 2
next:
info: 3
next:
main
a:
args:

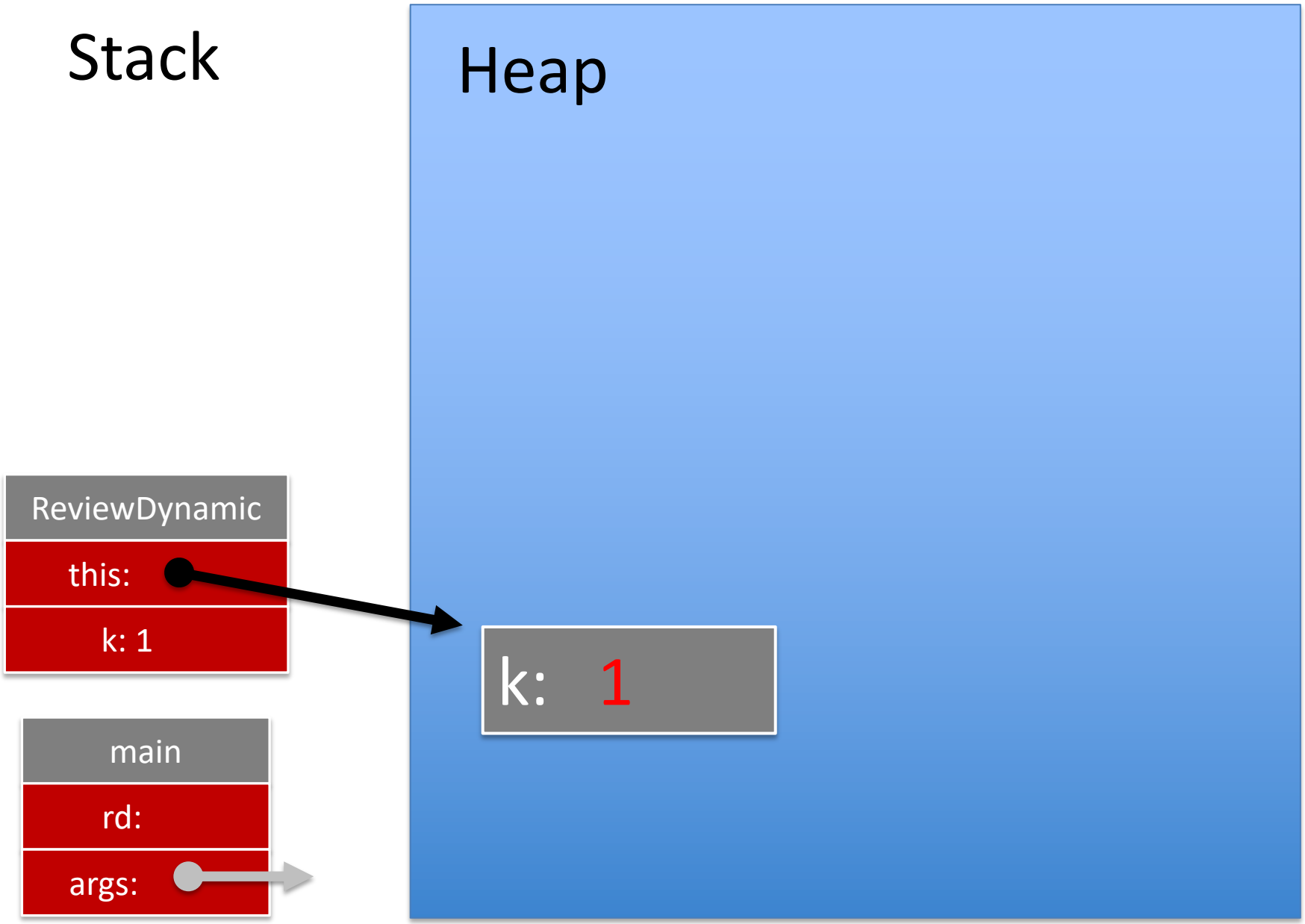
```java
private int k;

public ReviewDynamic(int k) {
  this.k = k;
}

private Node two(Node b) {
  Node c = new Node(this.k, node);
  return c;
}

private void one() {
  Node a = new Node(2, new Node(3, null));
  a = this.two(a);
  return a;
}

public static void main(String[] args) {
  ReviewDynamic rd = new ReviewDynamic(1);
  rd.one();
}
```

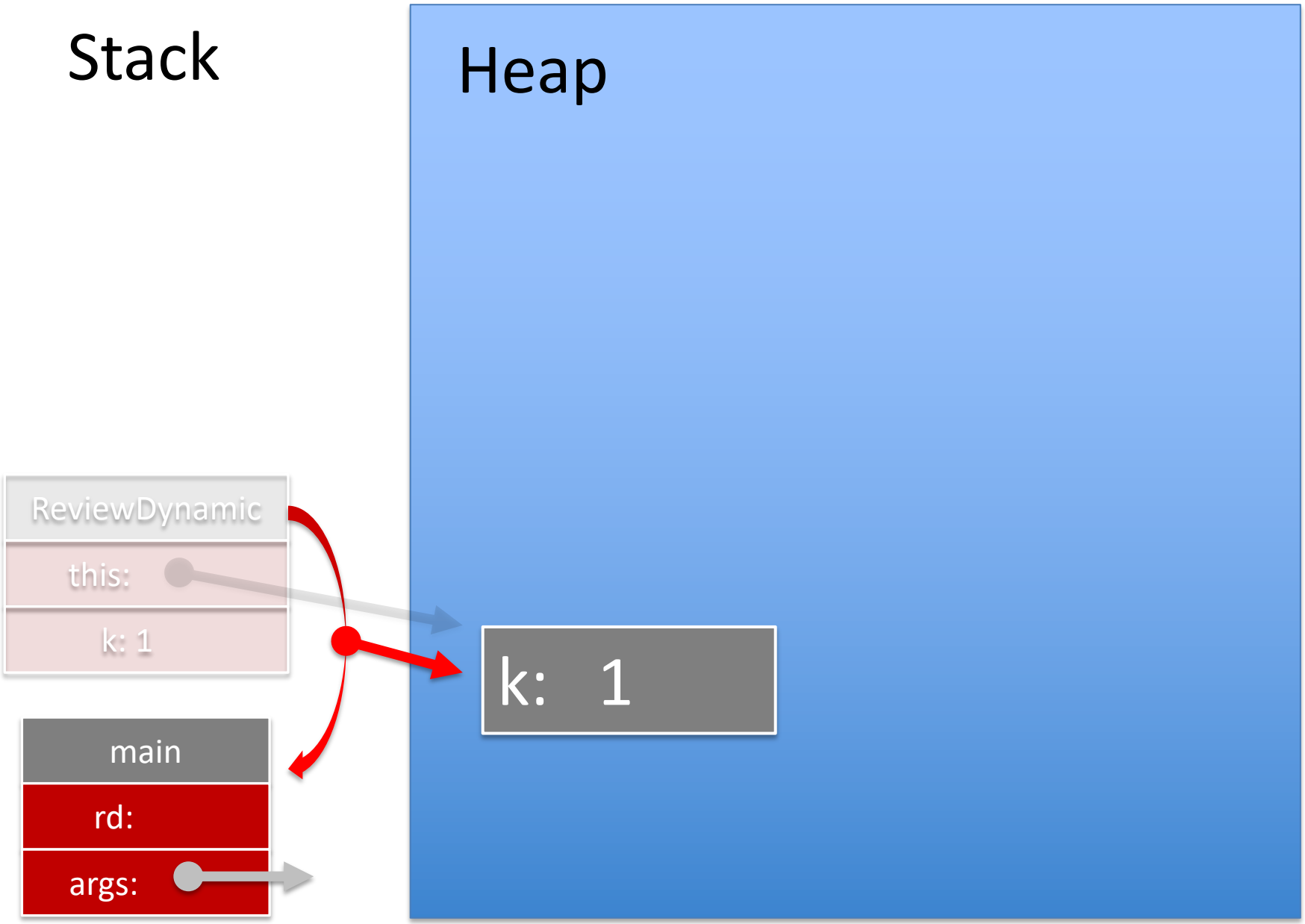
Stack

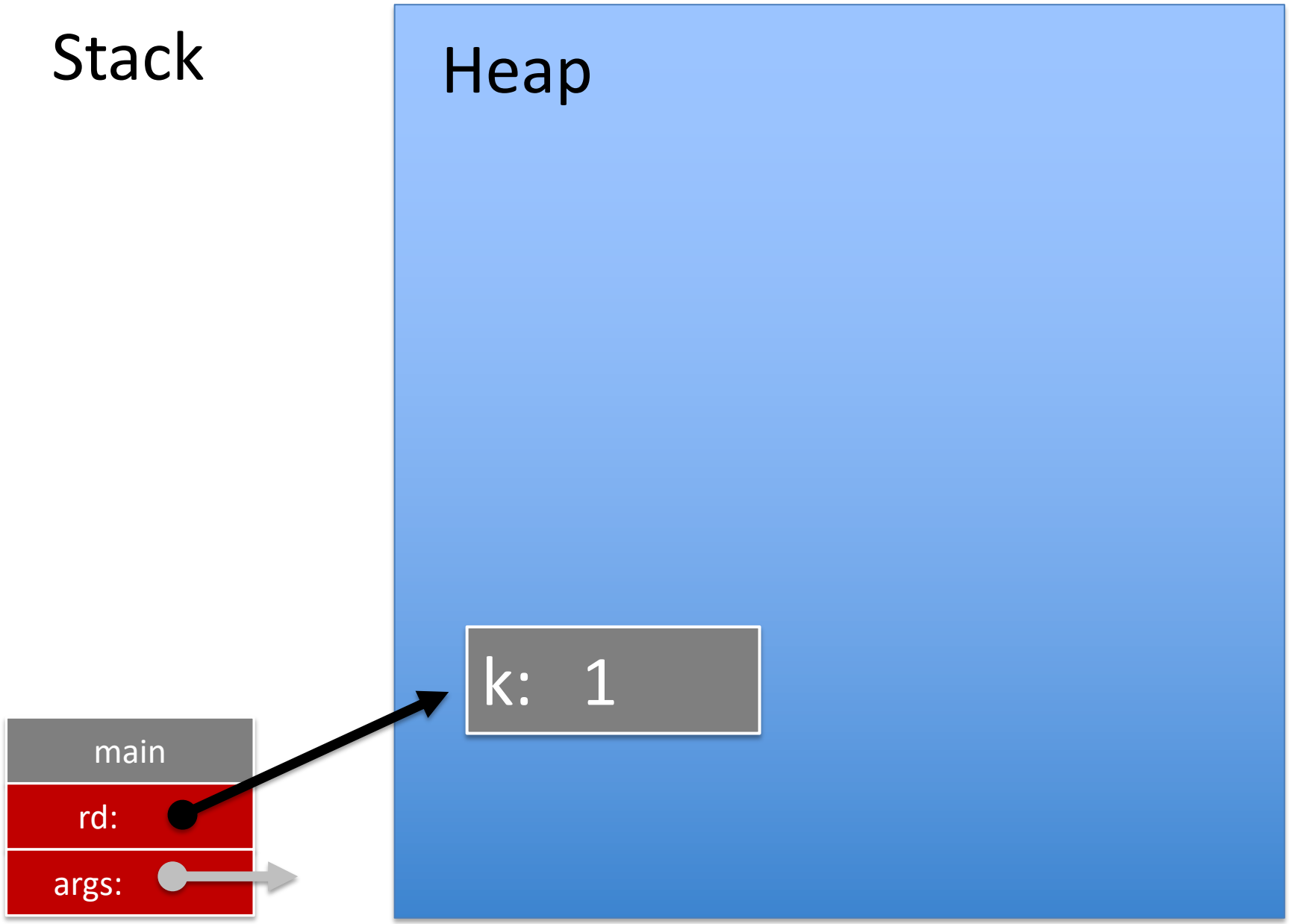
Heap

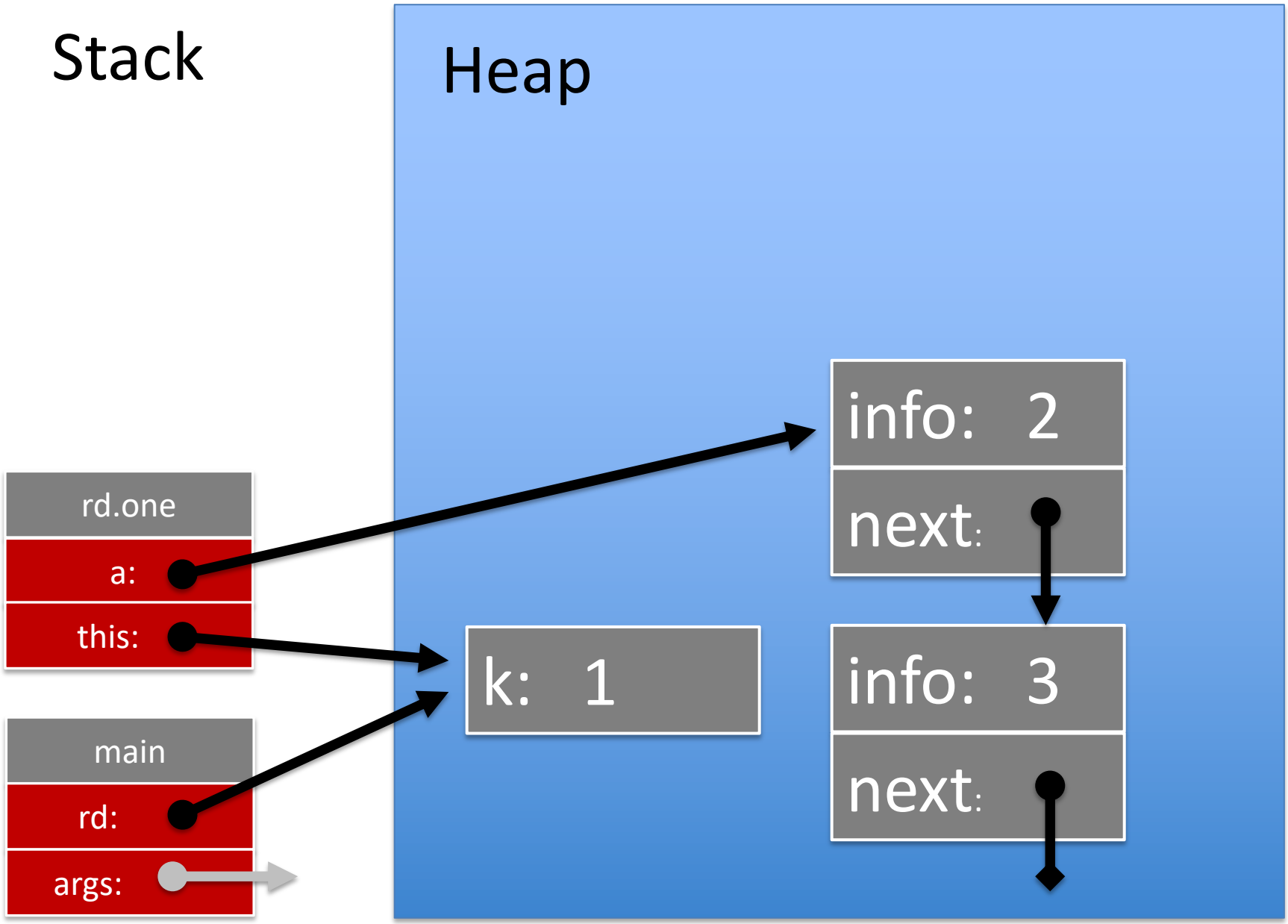



Stack
Heap
this.k
ReviewDynamic
this:
k: 1
k:
main
rd:
args:

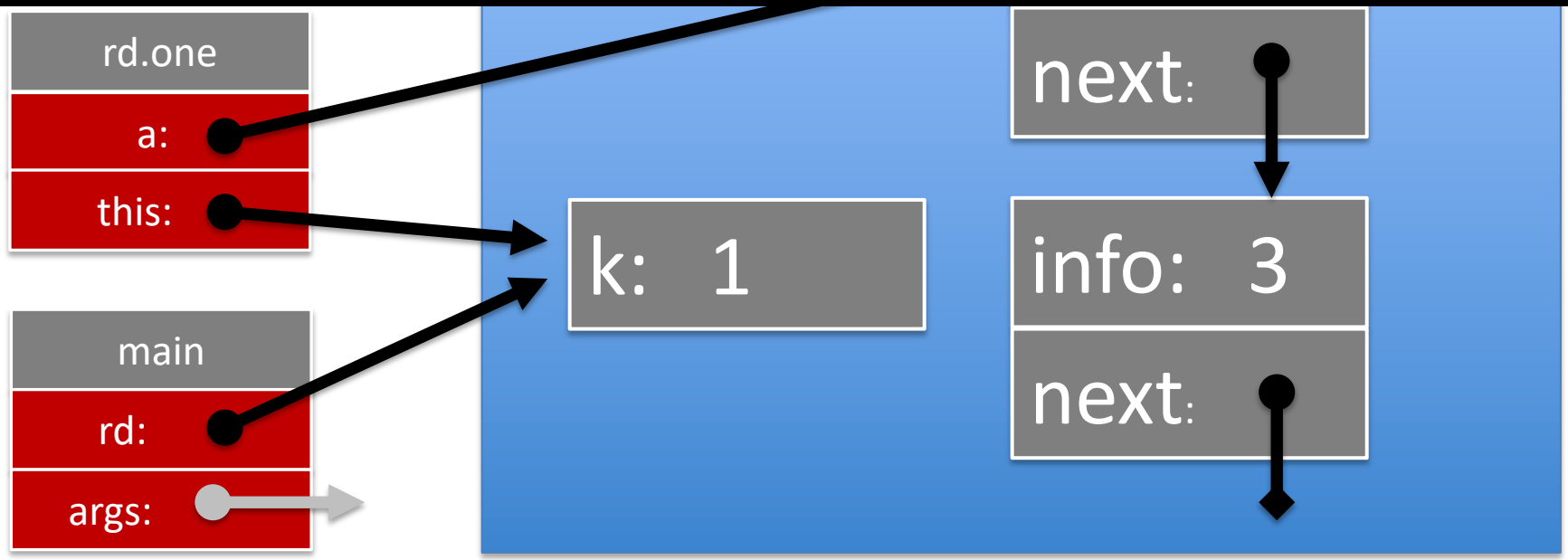

Stack
Heap
ReviewDynamic
this:
k: 1
k: 1
main
rd:
args:


Stack
Heap
ReviewDynamic
this:
k: 1
k: 1
main
rd:
args:


Stack
Heap
k: 1
main
rd:
args:

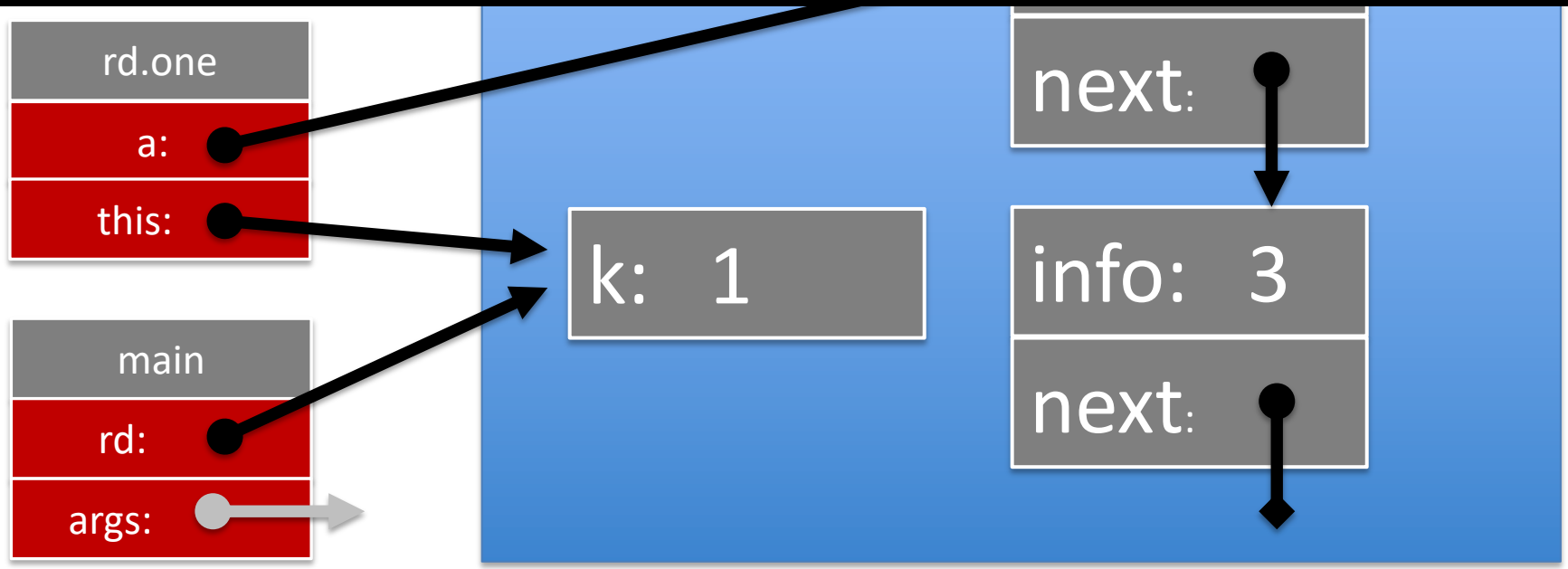

Stack
Heap
rd.one
a:
this:
main
rd:
args:
k: 1
info: 2
next:
info: 3
next:

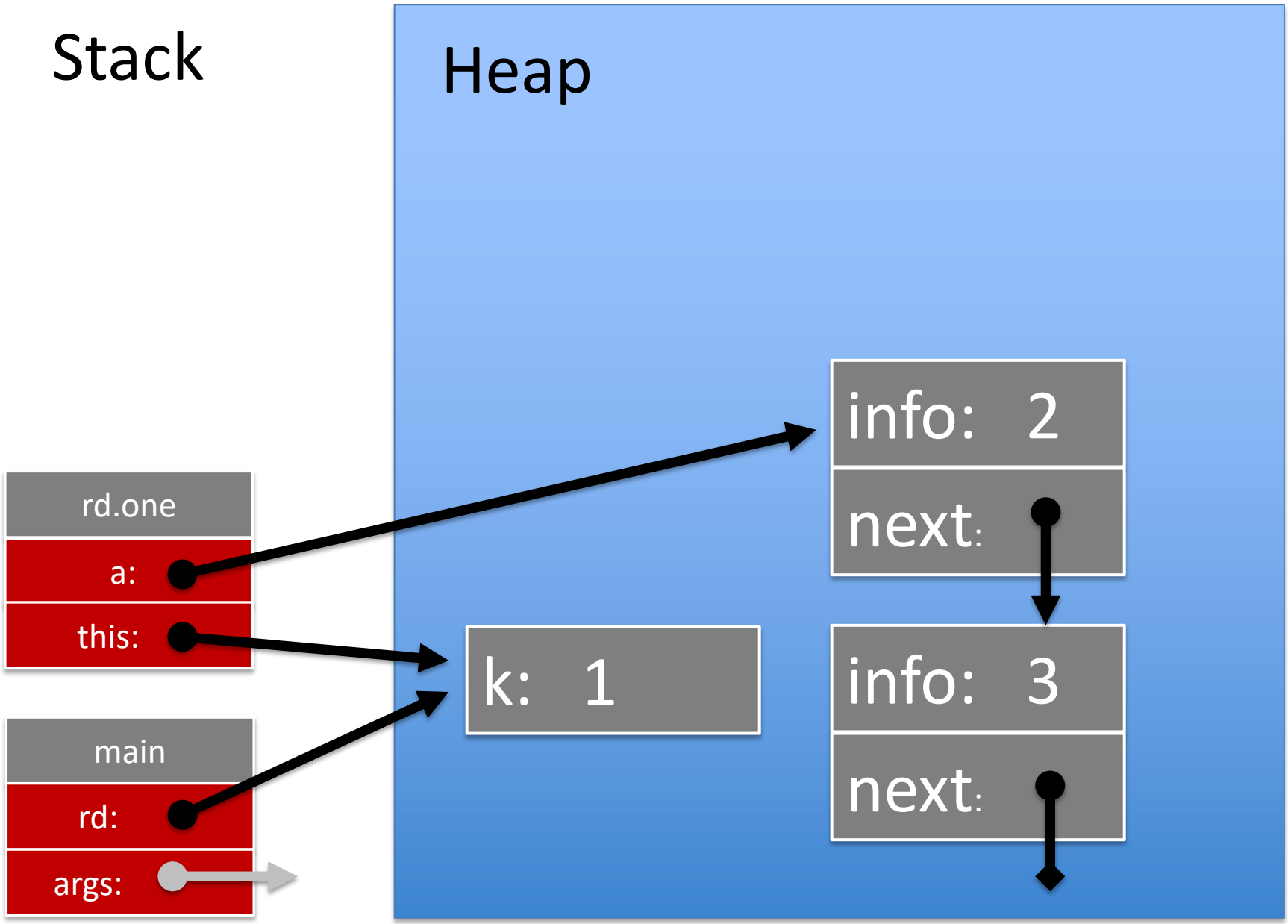
```java
private void one() {
  Node a = new Node(2, new Node(3, null));
  a = two(a);
  return a;
}

public static void main(String[] args) {
  ReviewDynamic rd = new ReviewDynamic(1);
  rd.one();
}
```

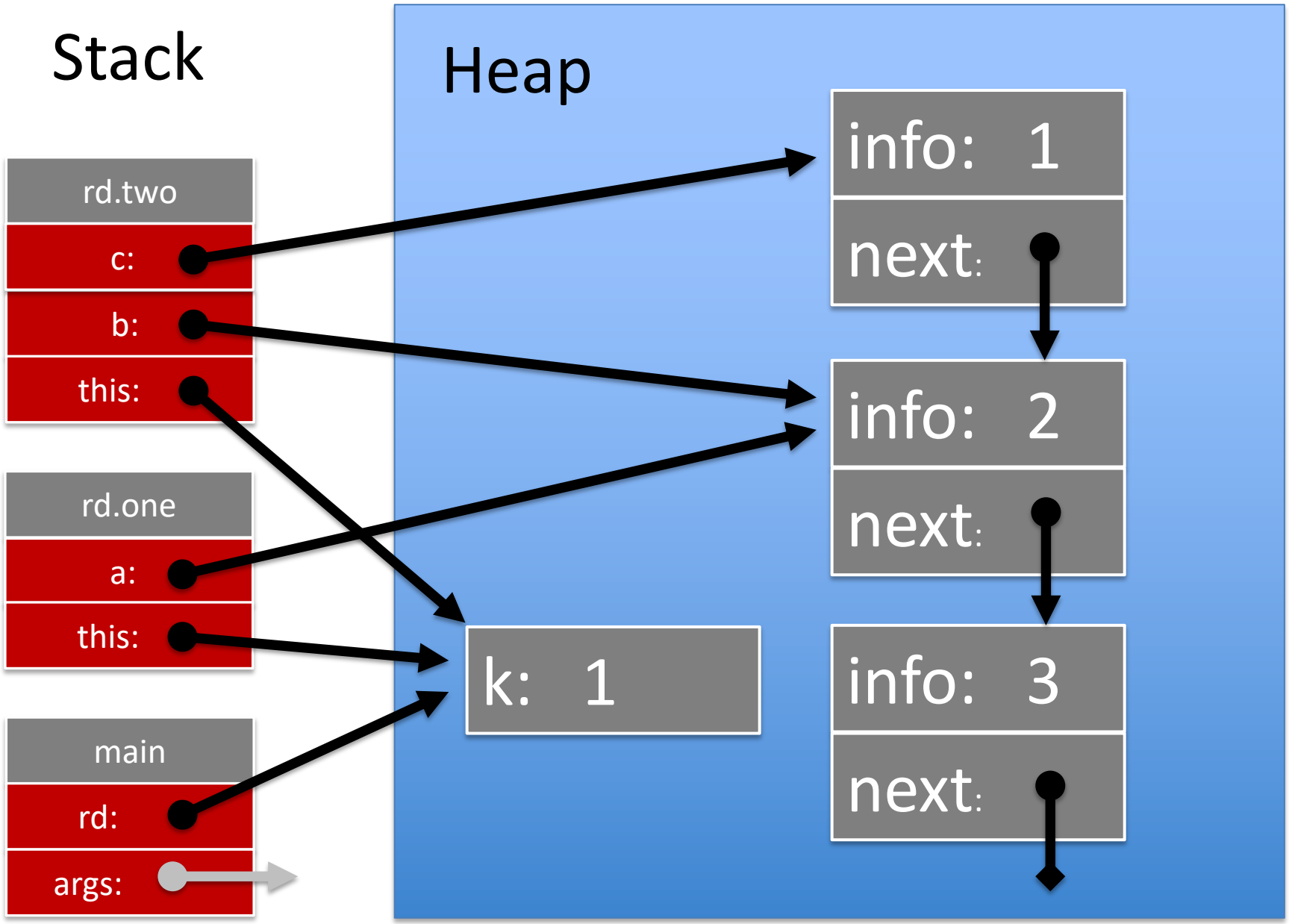


```
private void one(ReviewDynamic this) {
  Node a = new Node(2, new Node(3, null));
  a = two(a);
  return a;
}

public static void main(String[] args) {
  ReviewDynamic rd = new ReviewDynamic(1);
  rd.one(rd);
}
```

Compiler adds extra argument and this parameter

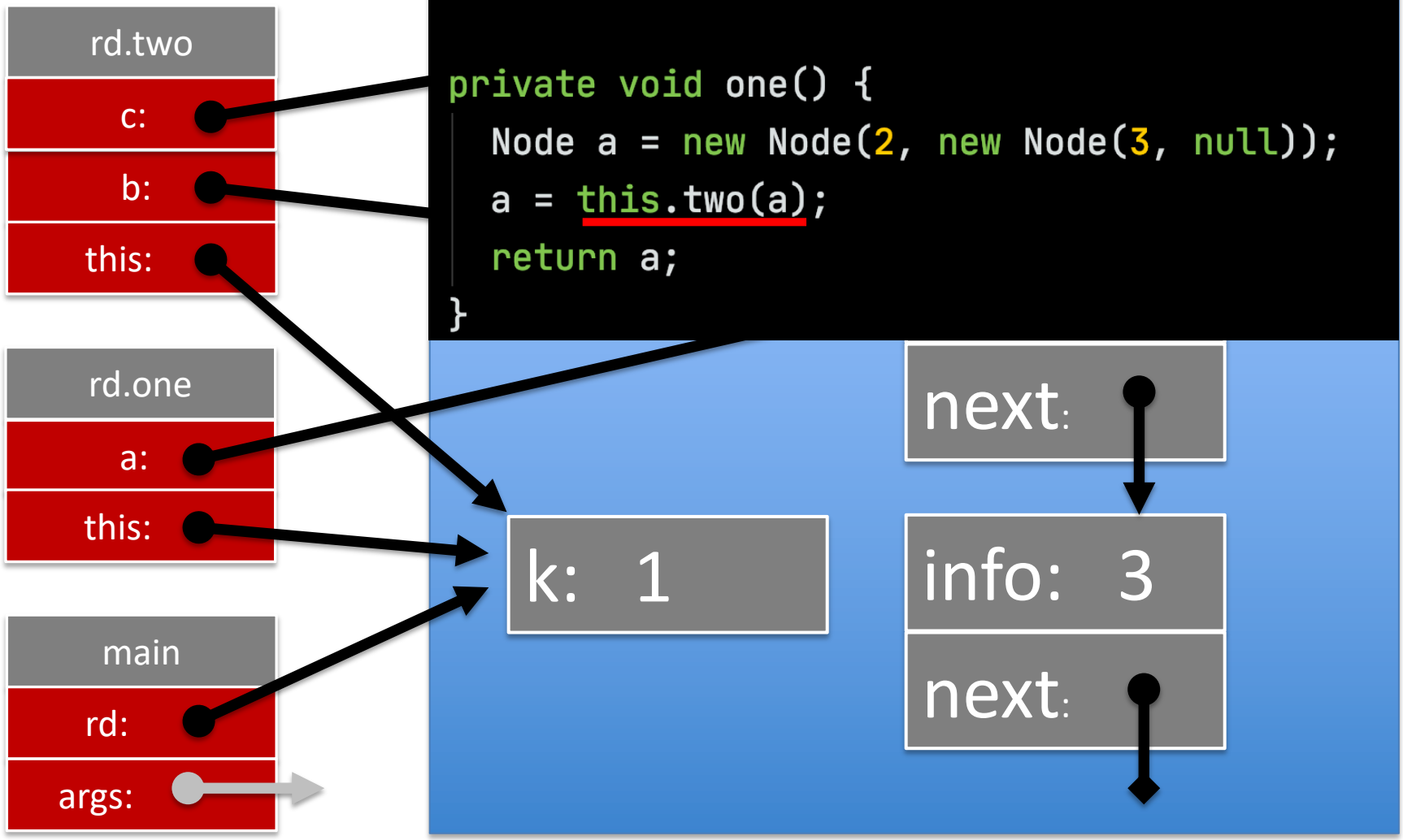



Stack
Heap
rd.one
a:
this:
main
rd:
args:
k: 1
info: 2
next:
info: 3
next:


Stack
Heap
rd.two
c:
b:
this:
rd.one
a:
this:
main
rd:
args:
k: 1
info: 1
next:
info: 2
next:
info: 3
next:

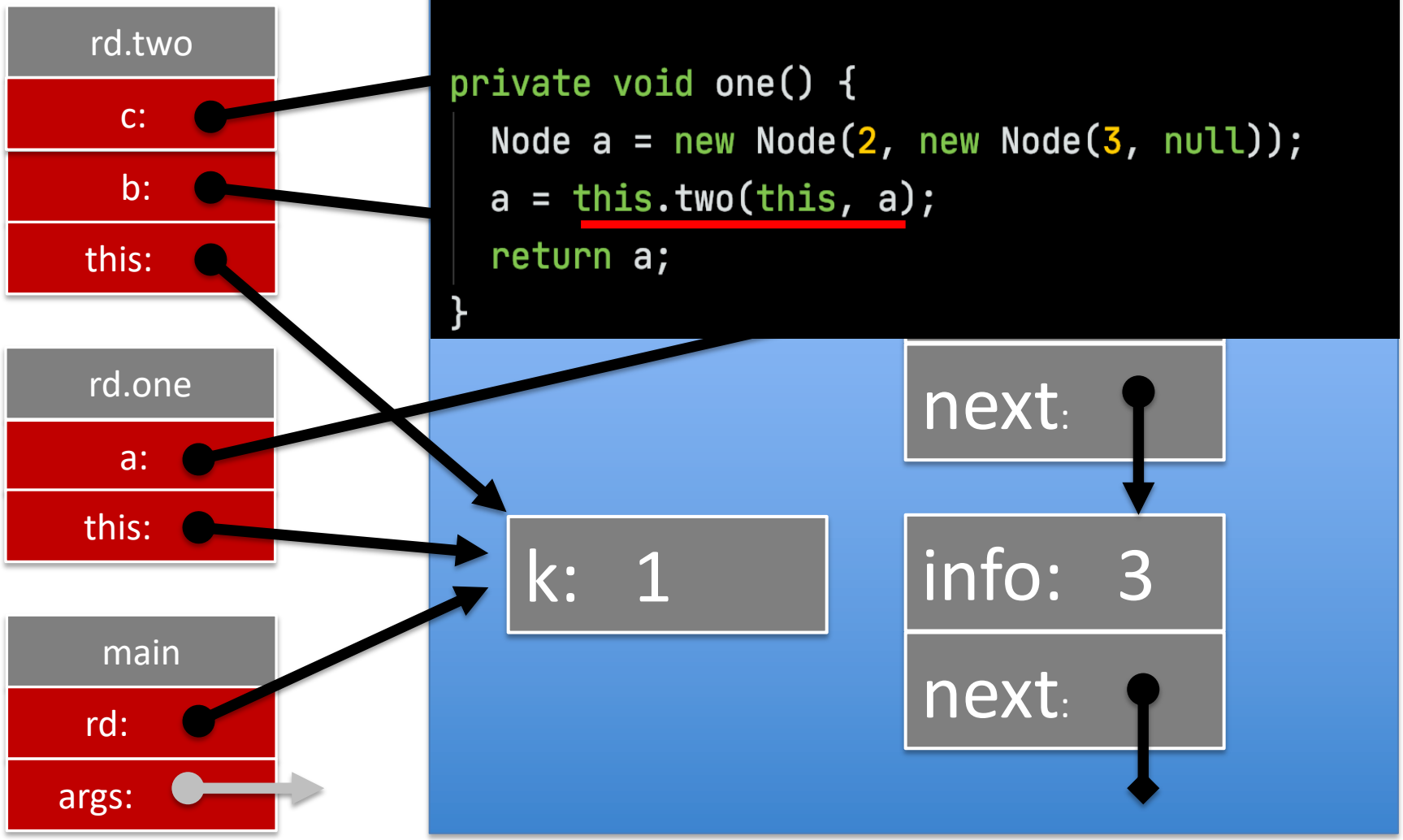
Stack

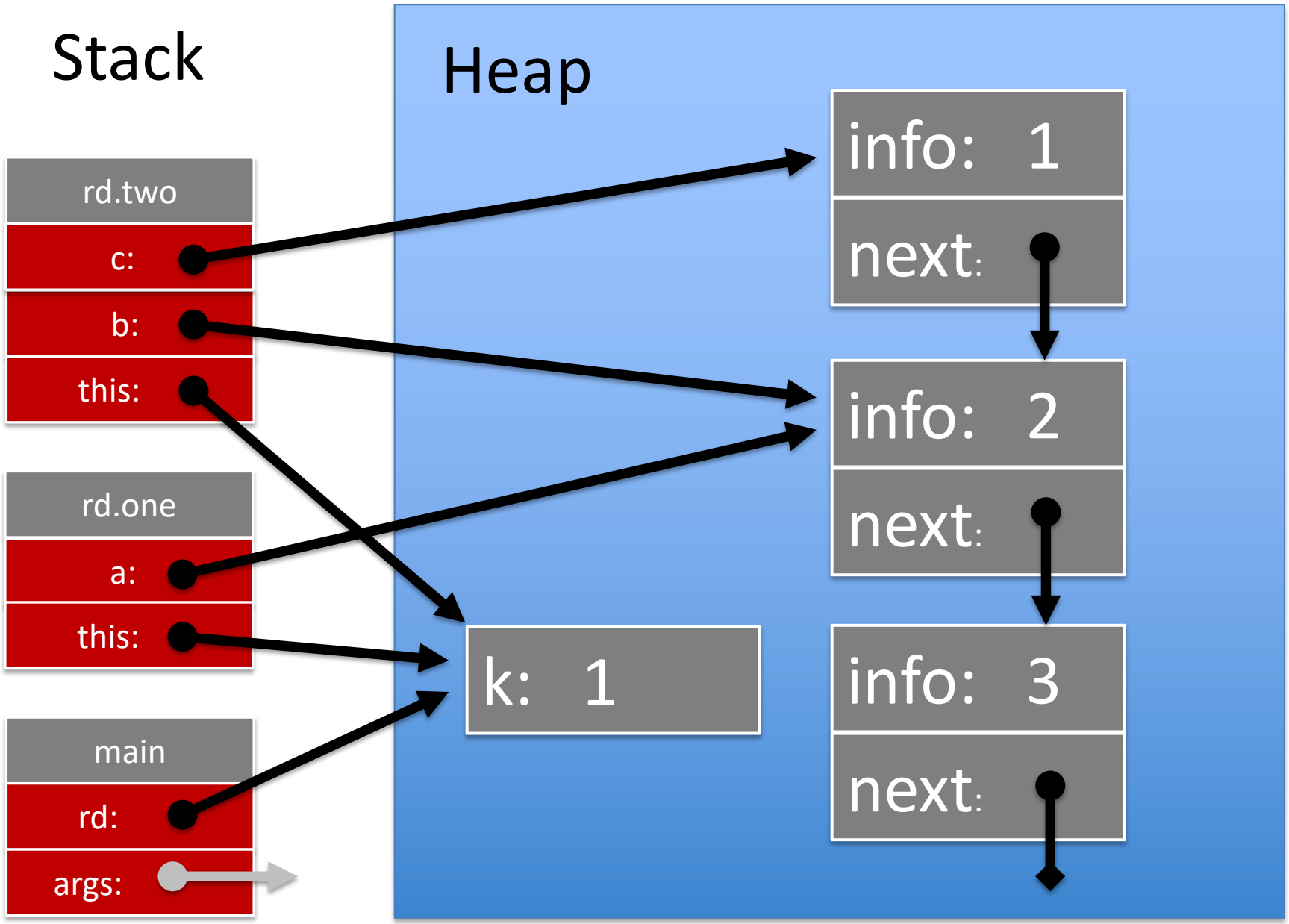
```
private Node two(Node b) {
  Node c = new Node(this.k, node);
  return c;
}

private void one() {
  Node a = new Node(2, new Node(3, null));
  a = this.two(a);
  return a;
}
```

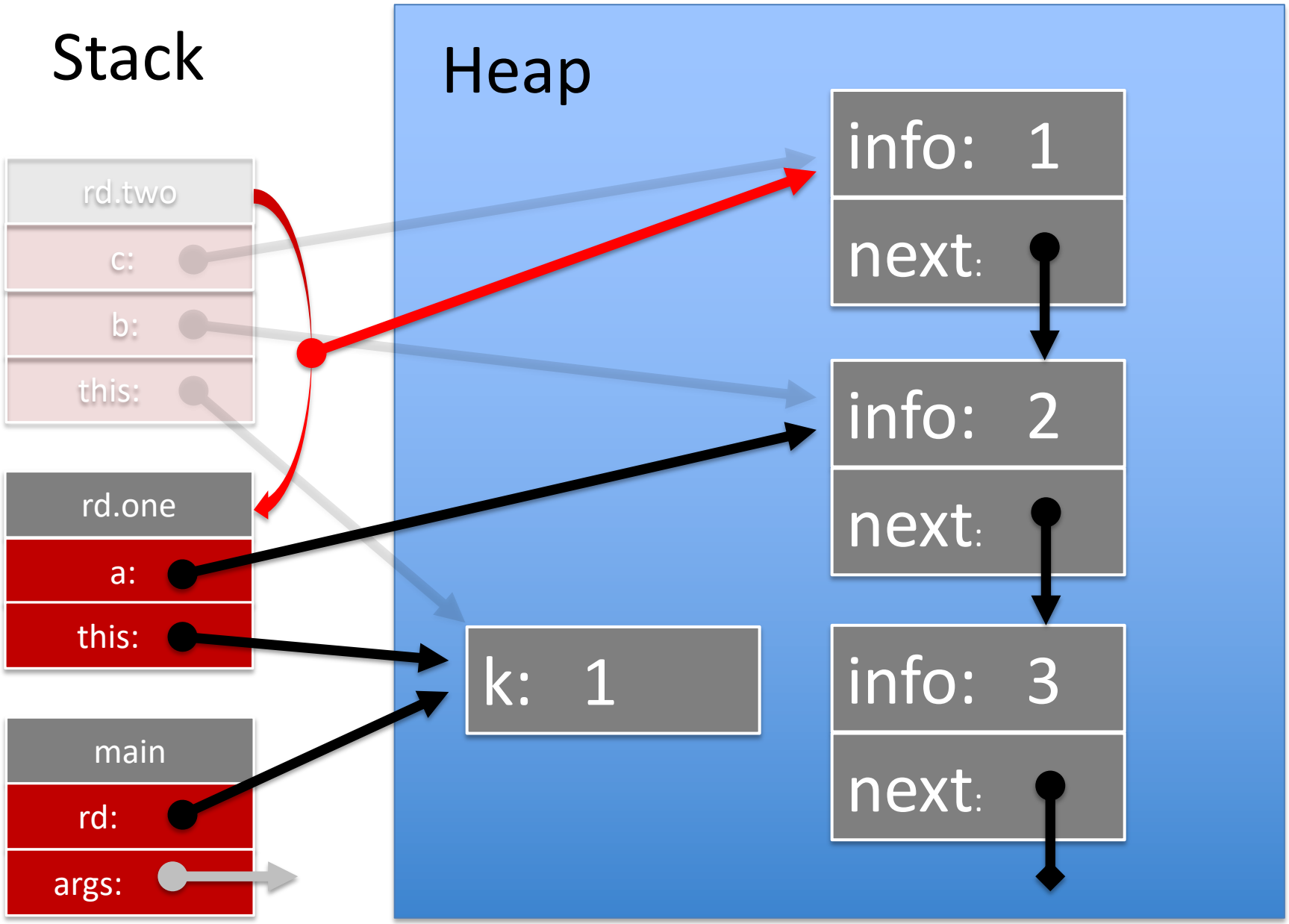


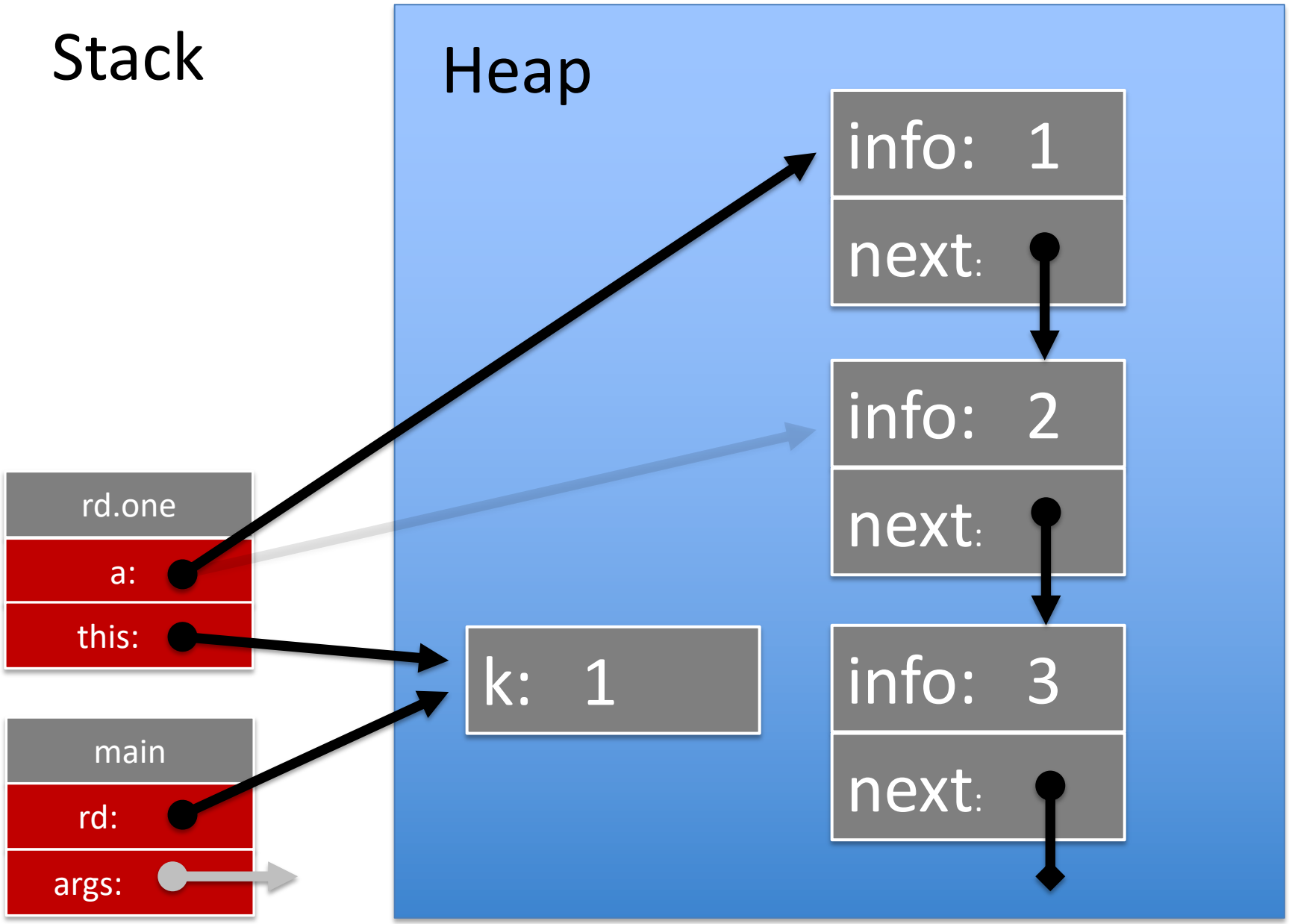
Stack

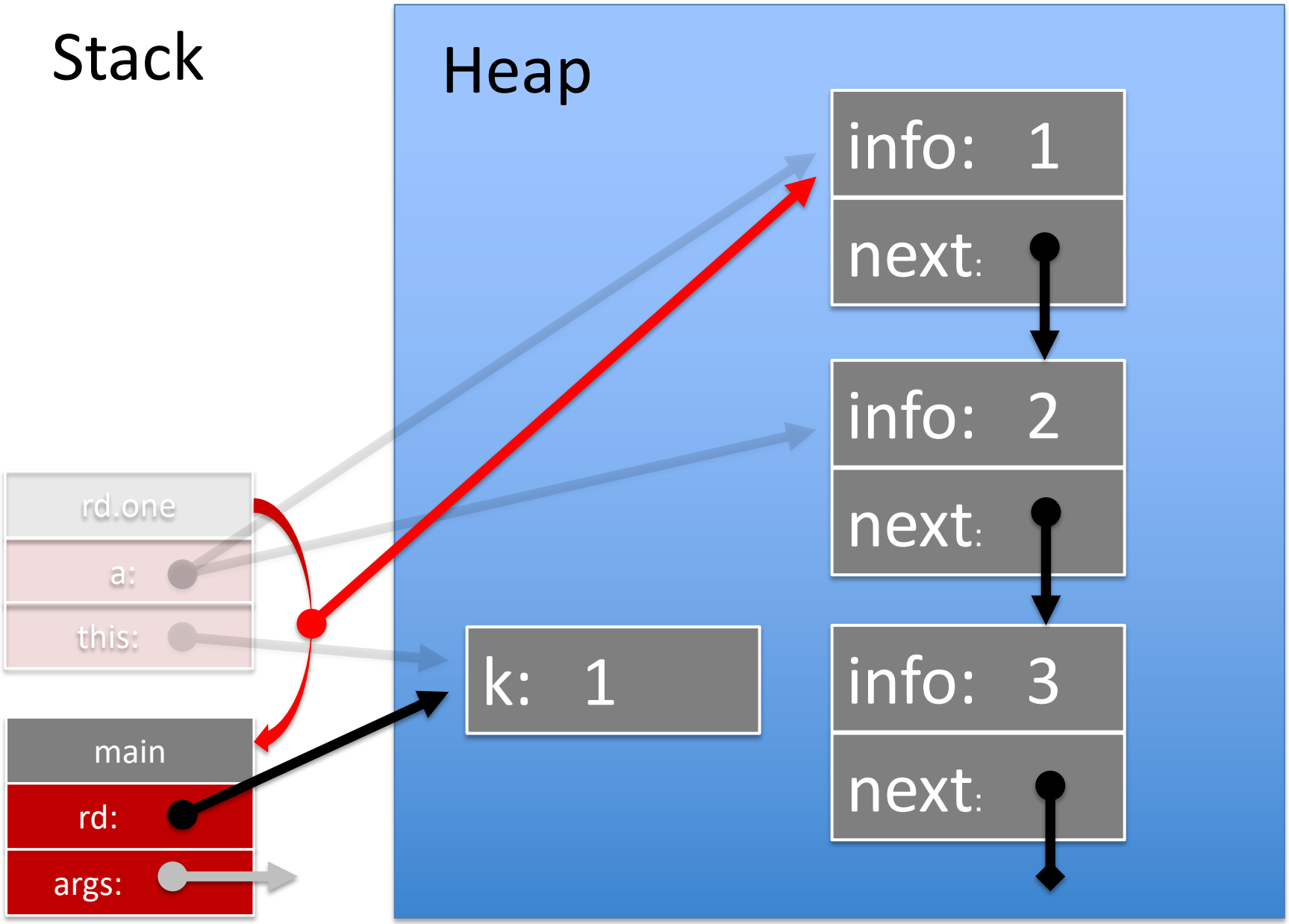
```
private Node two(ReviewDynamic this, Node b) {
  Node c = new Node(this.k, node);
  return c;
}

private void one() {
  Node a = new Node(2, new Node(3, null));
  a = this.two(this, a);
  return a;
}
```

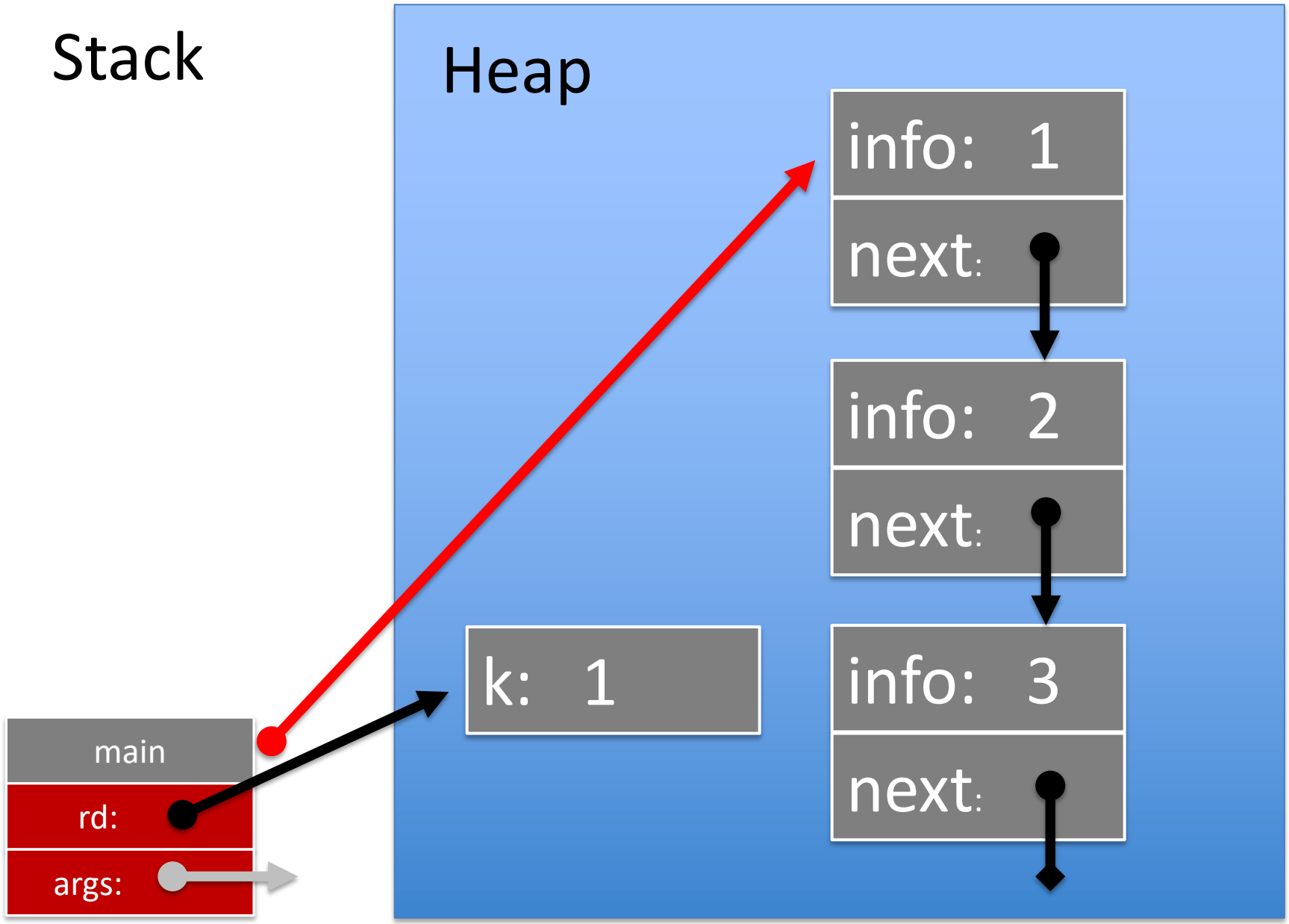



Stack
Heap
rd.two
c:
b:
this:
rd.one
a:
this:
main
rd:
args:
k: 1
info: 1
next:
info: 2
next:
info: 3
next:


Stack
Heap
rd.two
c:
b:
this:
rd.one
a:
this:
main
rd:
args:
k: 1
info: 1
next:
info: 2
next:
info: 3
next:


Stack
Heap
info: 1
next:
info: 2
next:
info: 3
next:
rd.one
a:
this:
k: 1
main
rd:
args:


Stack
Heap
info: 1
next:
info: 2
next:
info: 3
next:
rd.one
a:
this:
k: 1
main
rd:
args:


Stack
Heap
info: 1
next:
info: 2
next:
k: 1
info: 3
next:
main
rd:
args: