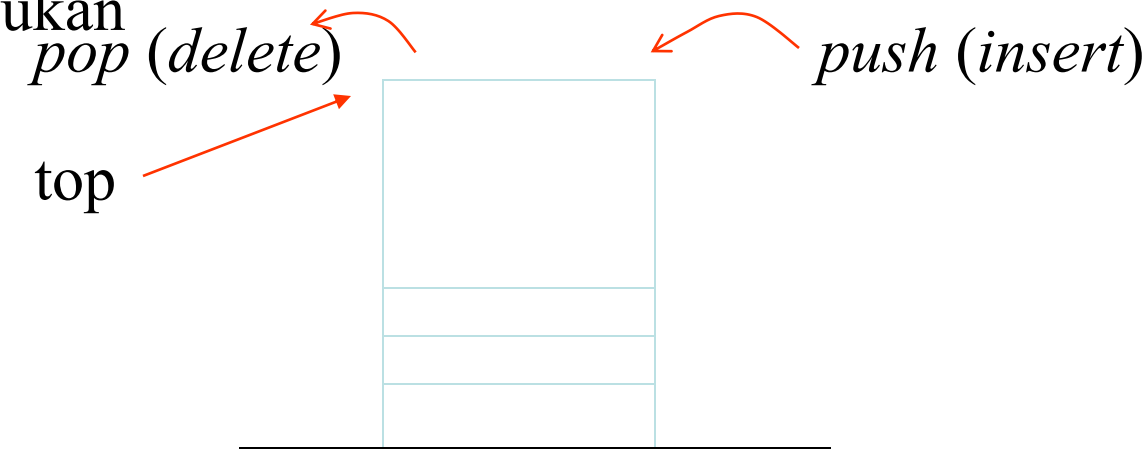
Algoritma dan Struktur Data

Stack

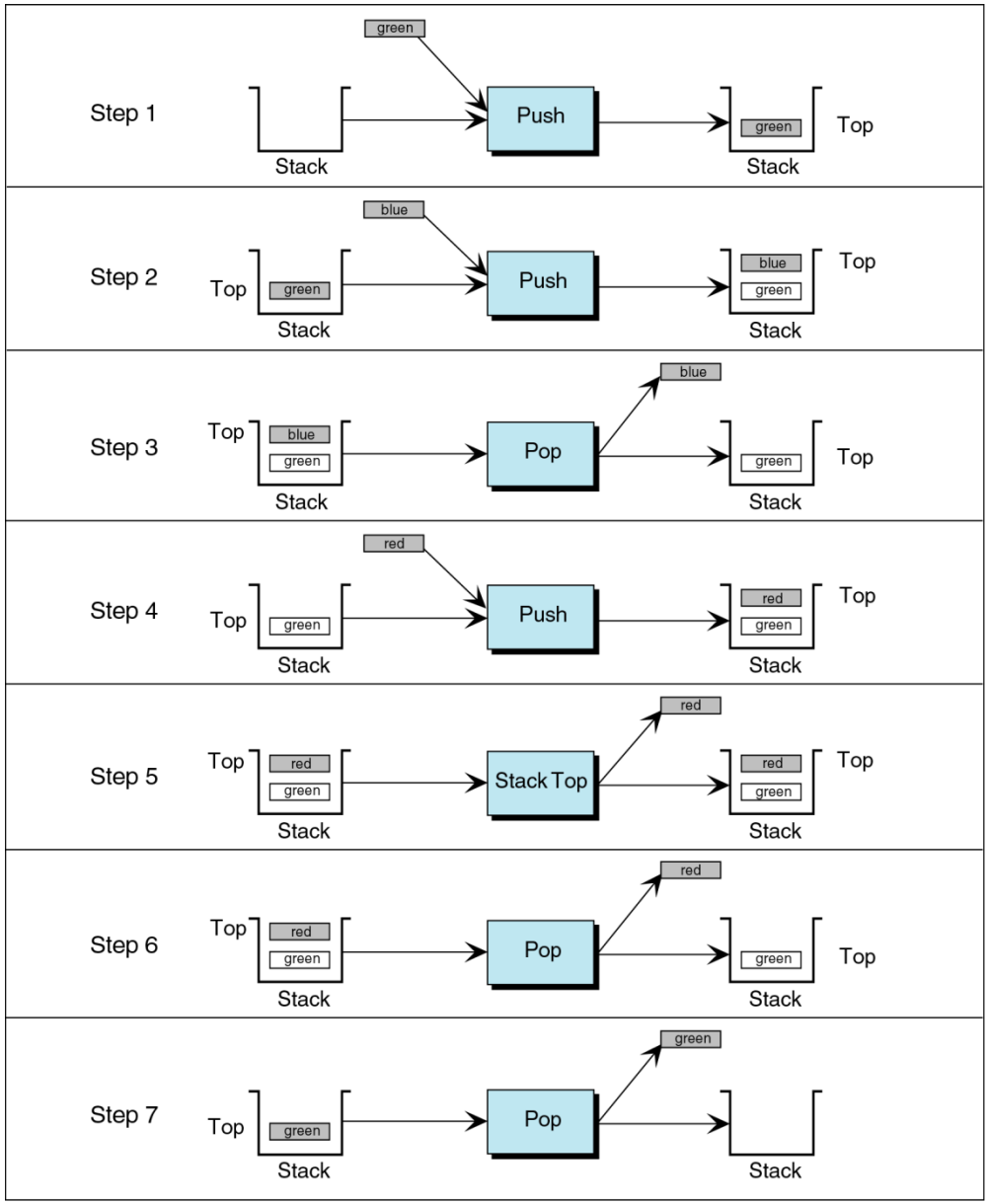
Tumpukan piring

ADT Stack

- Stack merupakan list dengan operasi insert dan delete sedemikian hingga node yang terakhir diinsertkan merupakan node yang pertama kali didelete
- Stack biasanya digambarkan sebagai tumpukan barang di mana operasi insert dan delete dilakukan di bagian puncak tumpukan

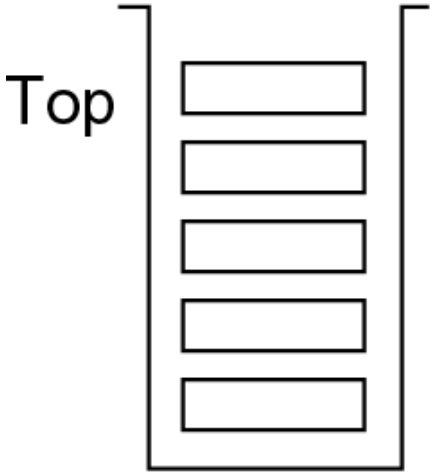


- Operasi stack : *isFull*, *isEmpty*, *makeEmpty, push*, *pop*, *top*.

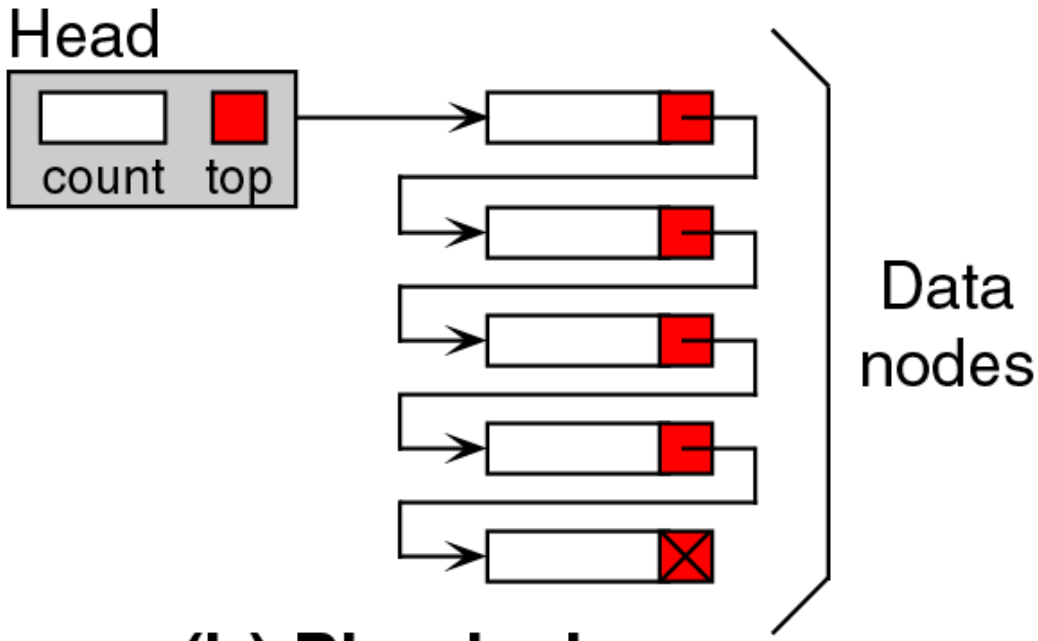

Step 1
green
Push
Stack
green
Top
Stack
Step 2
blue
Top
green
Push
Stack
blue
green
Top
Stack
Step 3
Top
blue
green
Pop
blue
Stack
green
Top
Stack
Step 4
red
Top
green
Push
Stack
red
green
Top
Stack
Step 5
red
Top
red
green
Stack Top
Stack
red
green
Top
Stack
Step 6
red
Top
red
green
Pop
Stack
red
green
Top
Stack
Step 7
green
Top
green
Pop
Stack
Stack

Implementasi Stack

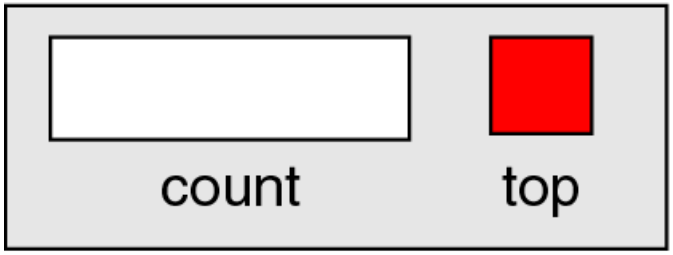
- Bisa menggunakan Array atau Linked List



(a) Conceptual



(b) Physical



Stack head structure

Stack node structure

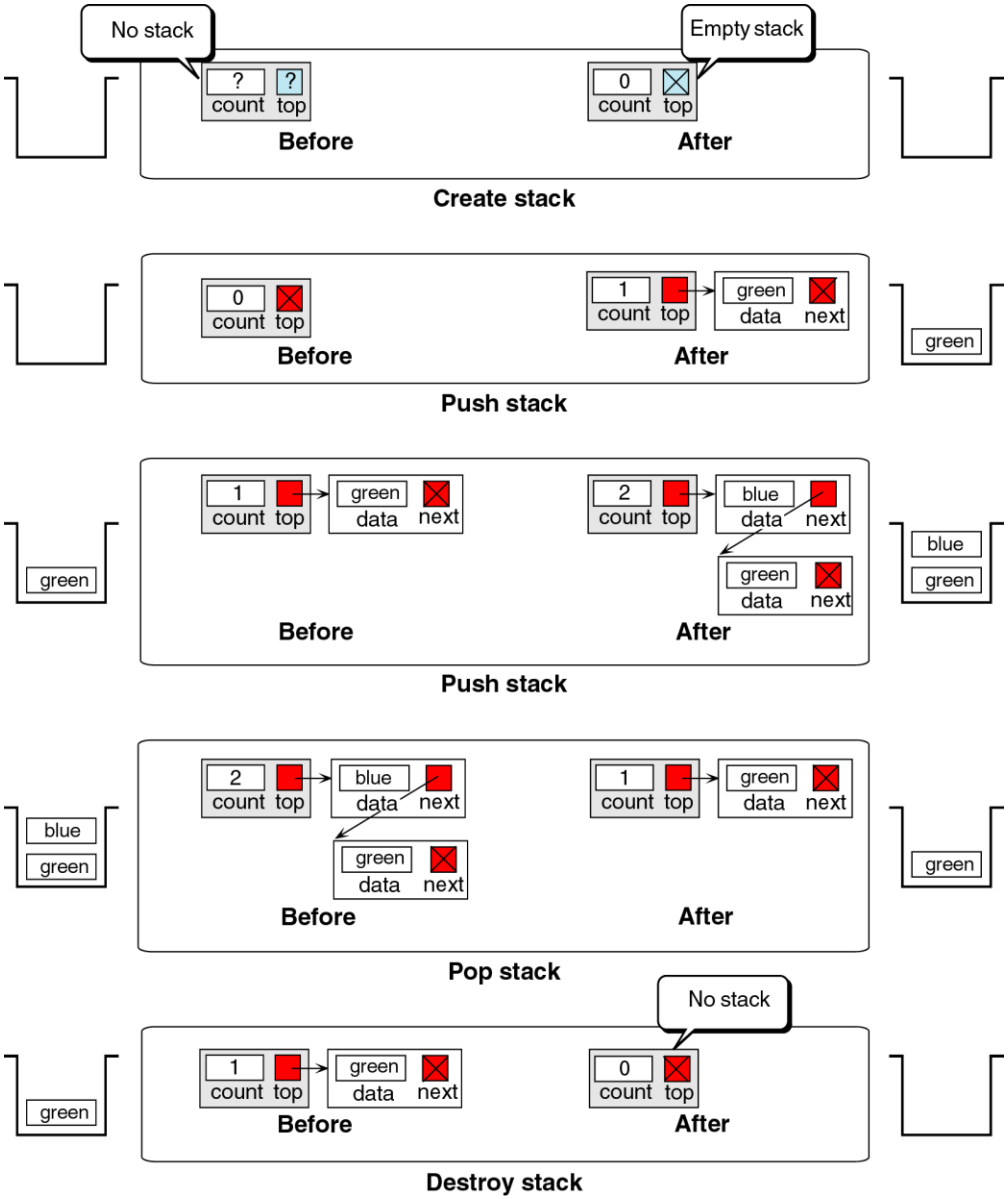
```
stack
  count  <integer>
  top    <node pointer>
end stack



node
  data   <dataType>
  next   <node pointer>
end node
```

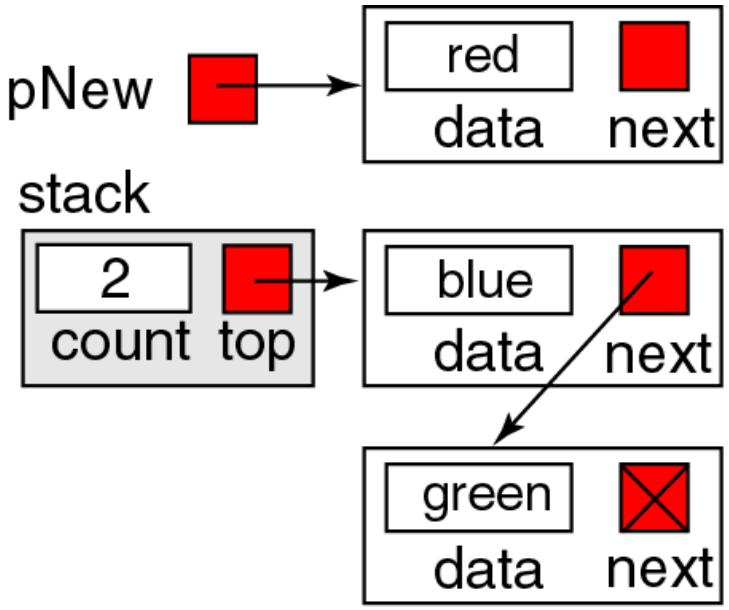
Operasi Stack

- Beberapa operasi stack
  - Push Stack
  - Pop Stack
  - Stack Top
  - Empty Stack
  - Full Stack
  - Stack Count
  - Destroy Stack

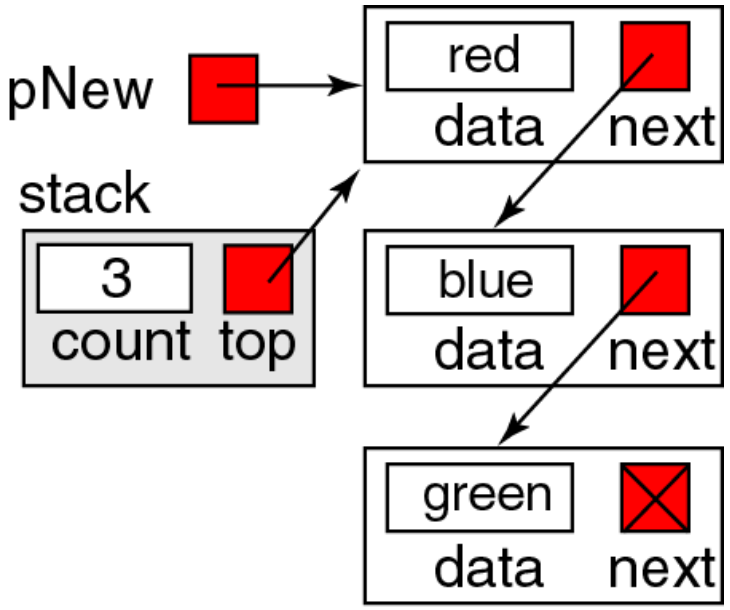

No stack
Empty stack
? count top
0 count top
Before
After
Create stack
0 count top
1 count top
green data next
Before
After
green
Push stack
1 count top
green data next
2 count top
blue data next
green data next
Before
After
green
blue
green
Push stack
2 count top
blue data next
green data next
1 count top
green data next
Before
After
blue
green
green
Pop stack
No stack
1 count top
green data next
0 count top
Before
After
green
Destroy stack

Create Stack

1. `stack.count = 0`
2. `stack.top = null`
3. `return`



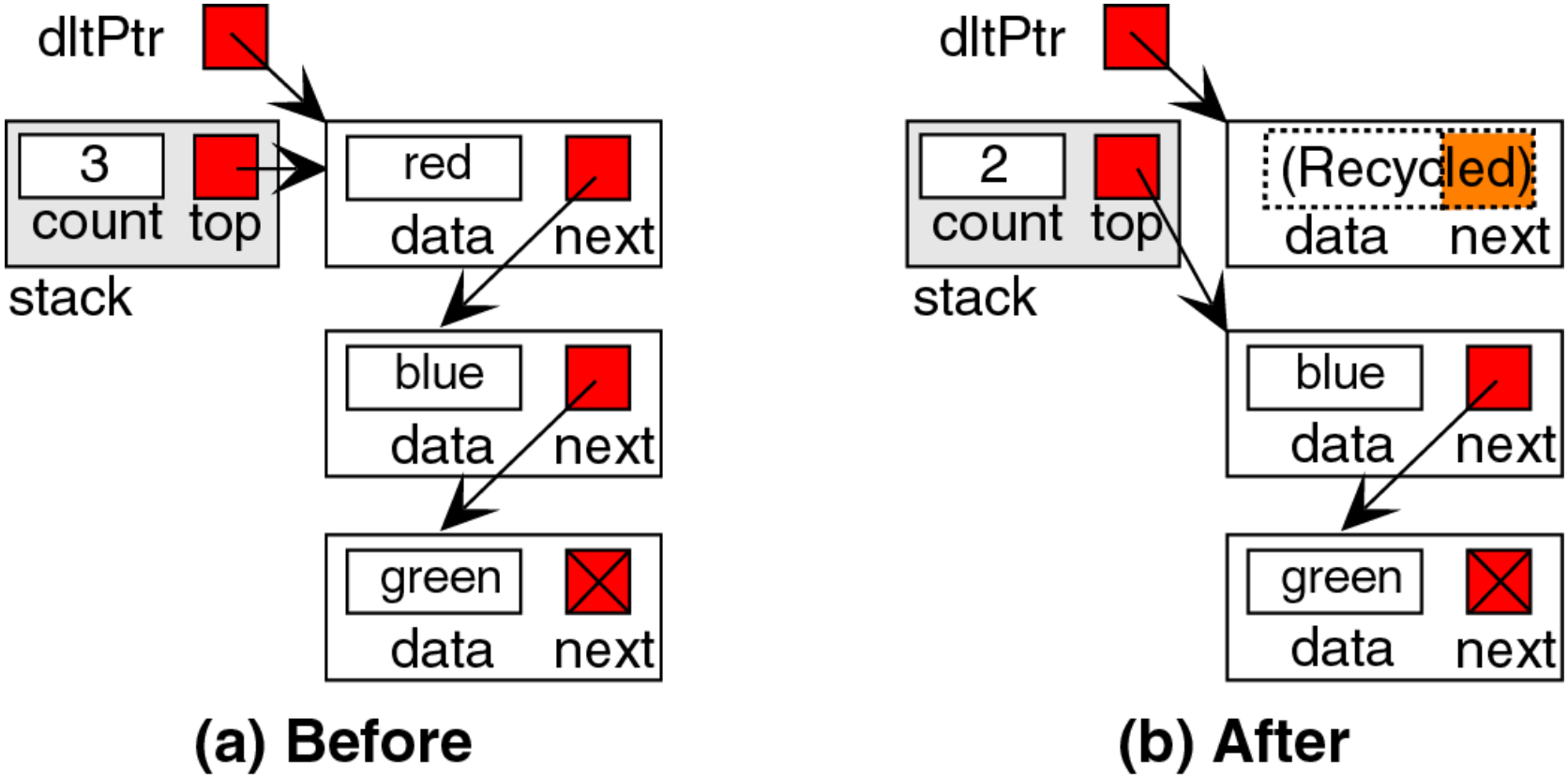
(a) Before



(b) After

Push

```
1. if (stack full)
   1.success = false
2. else
   1.allocate (newPtr)
   2.newPtr->data = data
   3.newPtr->next = stack.top
   4.stack.top = newPtr
   5.stack.count = stack.count + 1
   6.success = true
3. end if
4. return success
```


dltPtr
3
count
top
stack
red
data
next
blue
data
next
green
data
next
(a) Before
dltPtr
2
count
top
stack
(Recycled)
data
next
blue
data
next
green
data
next
(b) After

Pop

```
1.  if (stack empty)
    1. success = false
2.  else
    1. dltPtr = stack.top
    2. dataOut = stack.top->data
    3. stack.top = stack.top->next
    4. stack.count = stack.count - 1
    5. recycle (dltPtr)
    6. success = true
3.  end if
4.  return success
```

Stack Top

```
1. if (stack empty)
   1.success = false
2. else
   1.dataOut = stack.top->data
   2.success = true
3. end if
4. return success
```

Empty Stack

```
1. if (stack not empty)
   1.result = false
2. else
   1.result = true
3. end if
4. return result
```

Full Stack

```
1.  if (memory available)
    1. result = false
2.  else
    1. result = true
3.  end if
4.  return result
```

Stack Count

```
1. return (stack.count)
```

Destroy Stack

1. loop (stack.top not null)
   1. temp = stack.top
   2. stack.top = stack.top->next
   3. recycle (temp)
2. end loop
3. stack.count = 0
4. return