Static Analysis of Accessed Regions in Recursive Data Structures

Stephen Chong
Radu Rugina

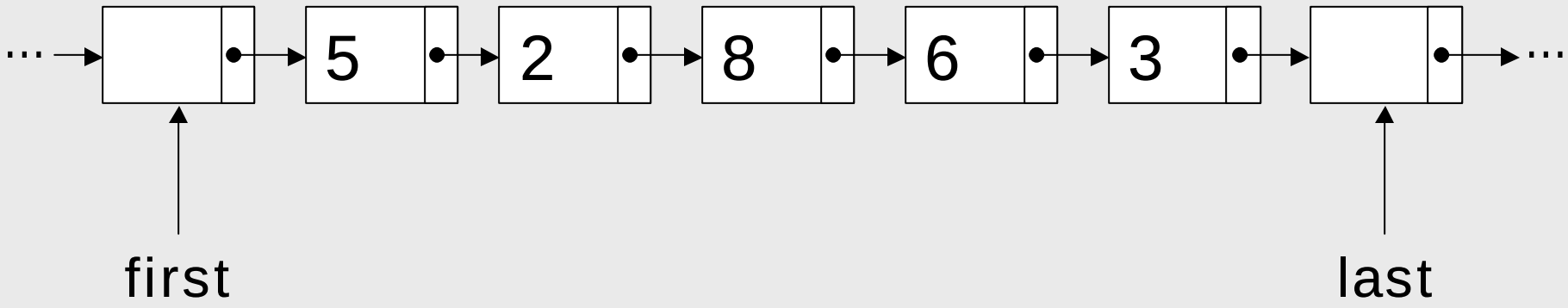
Cornell University

What This Talk is About

- Problem: Precise characterization of regions accessed by statements and procedures
  - For recursive programs with destructive updates
  - Fine-grained notion of regions: substructures within recursive data structures.
    - E.g. sublists within lists, sub trees within trees
- How we do it:
  - Context sensitive interprocedural analysis algorithm
  - Precise shape information
  - Region access information
- Uses:
  - Parallelization, Program Understanding, Correctness

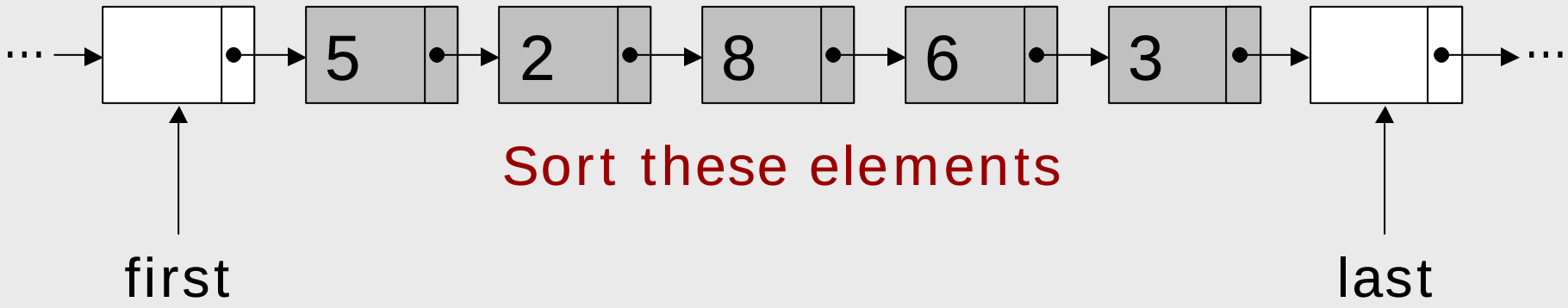
Quicksort Example

- Sorts a sublist in place (i.e. with destructive updates)



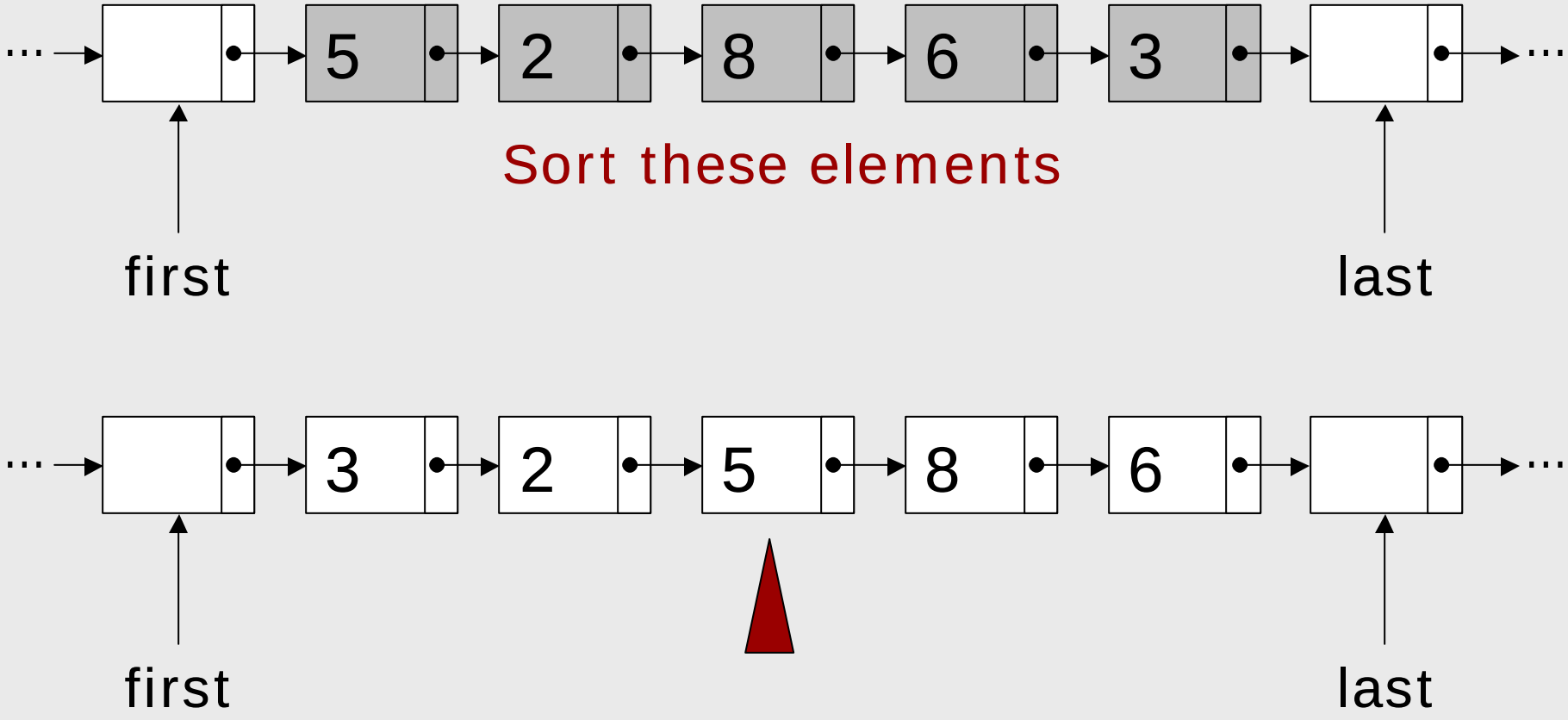
Quicksort Example

- Sorts a sublist in place (i.e. with destructive updates)

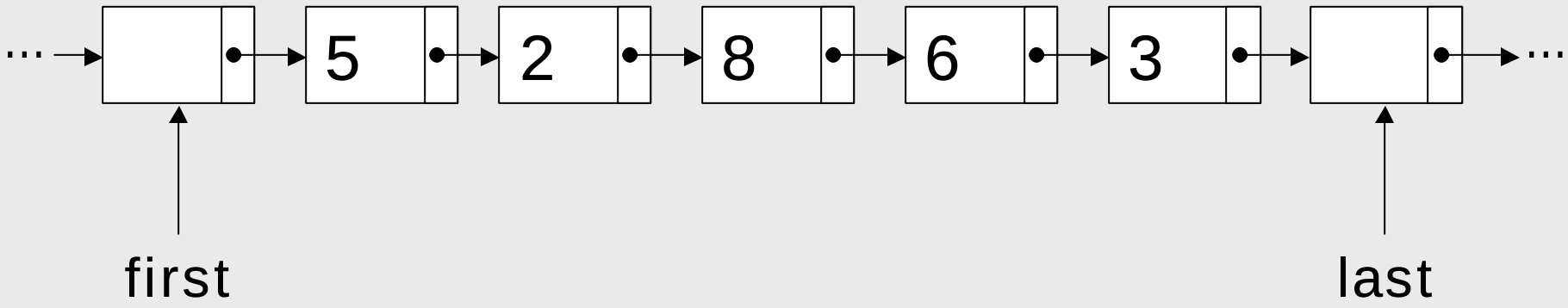


Quicksort Example

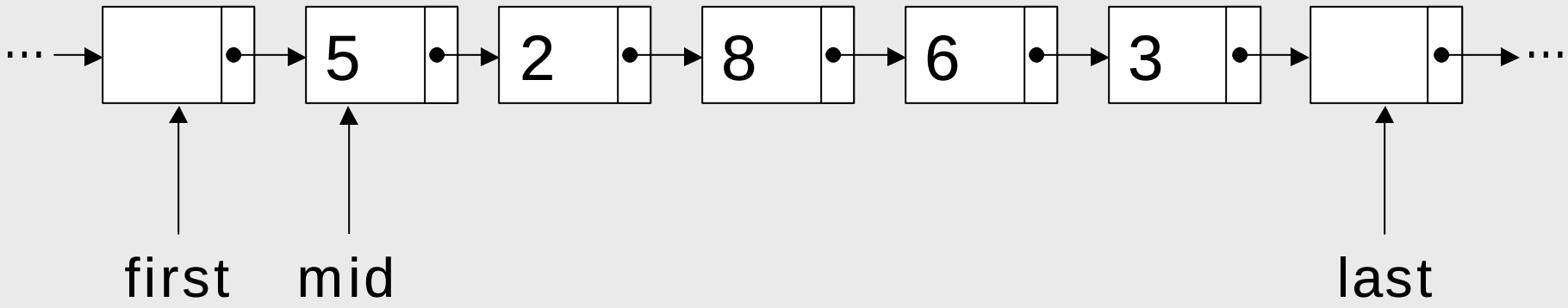
- Chooses a pivot value
- Partitions list into sublists destructively



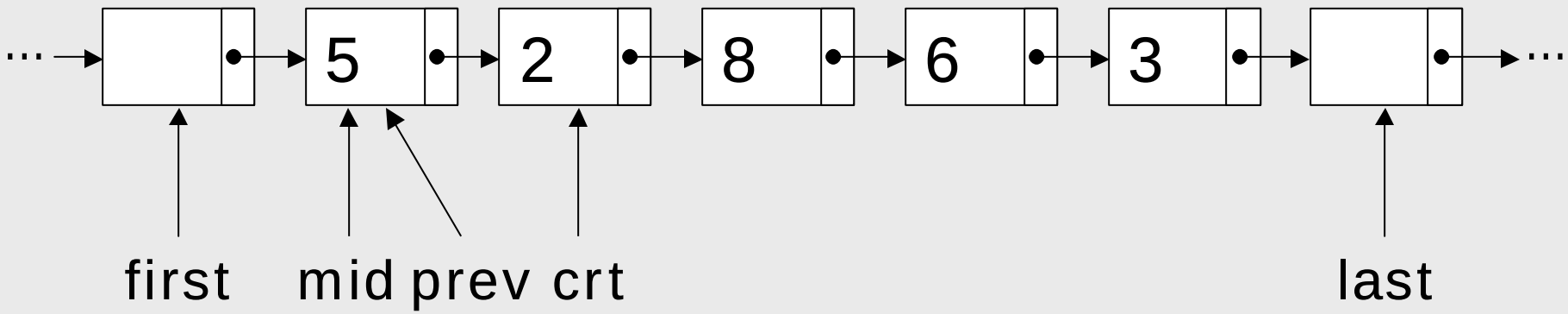
Quicksort Example: Partitioning



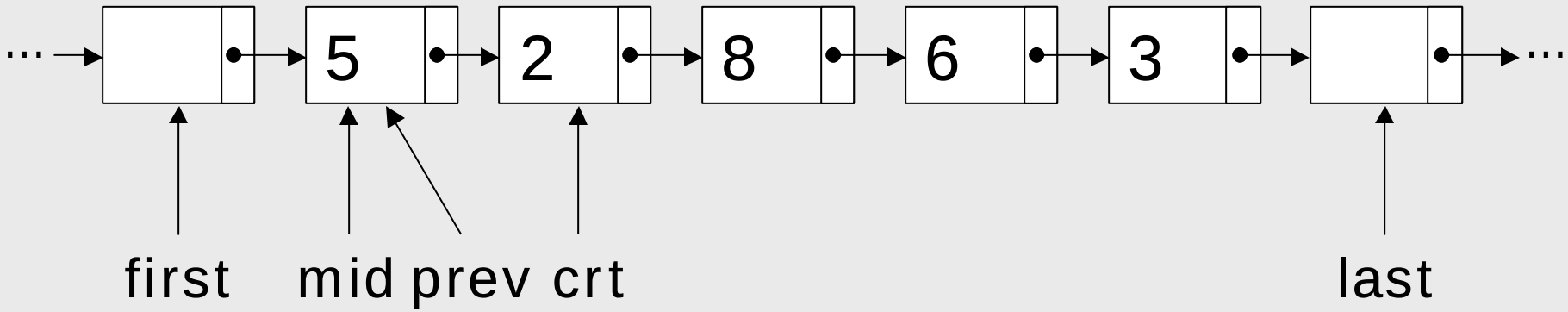
Quicksort Example: Partitioning



Quicksort Example: Partitioning

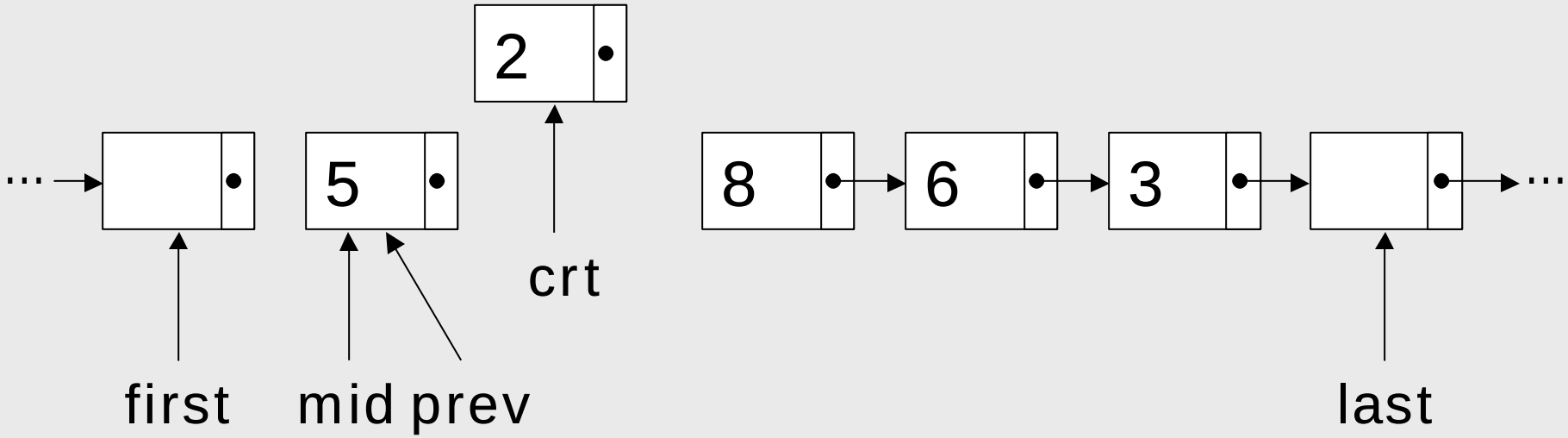


Quicksort Example: Partitioning



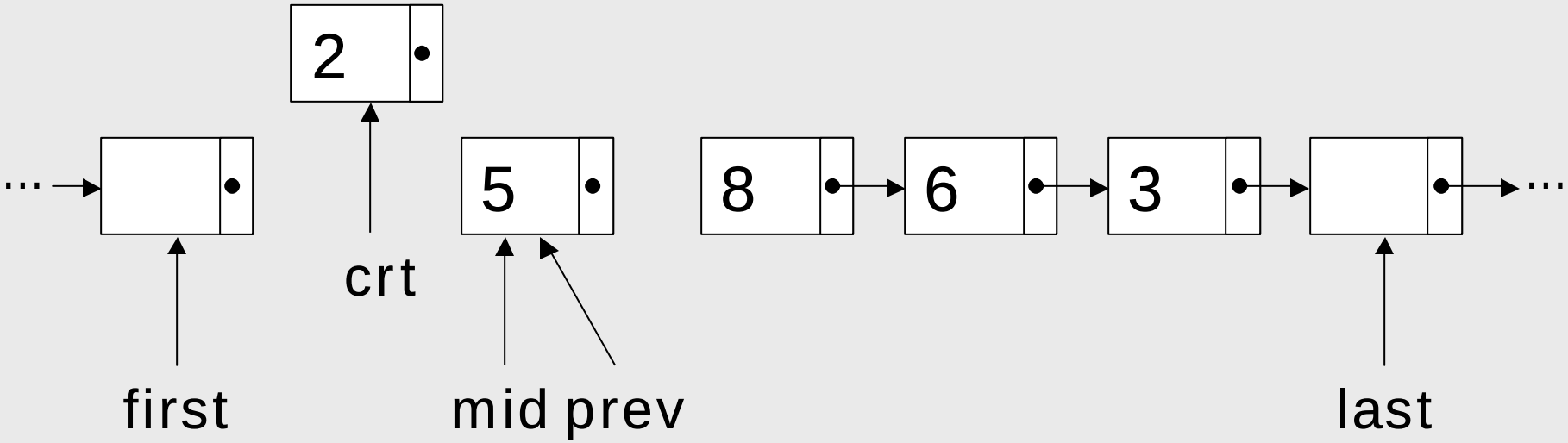
mid.val > crt.val ? Yes!

Quicksort Example: Partitioning



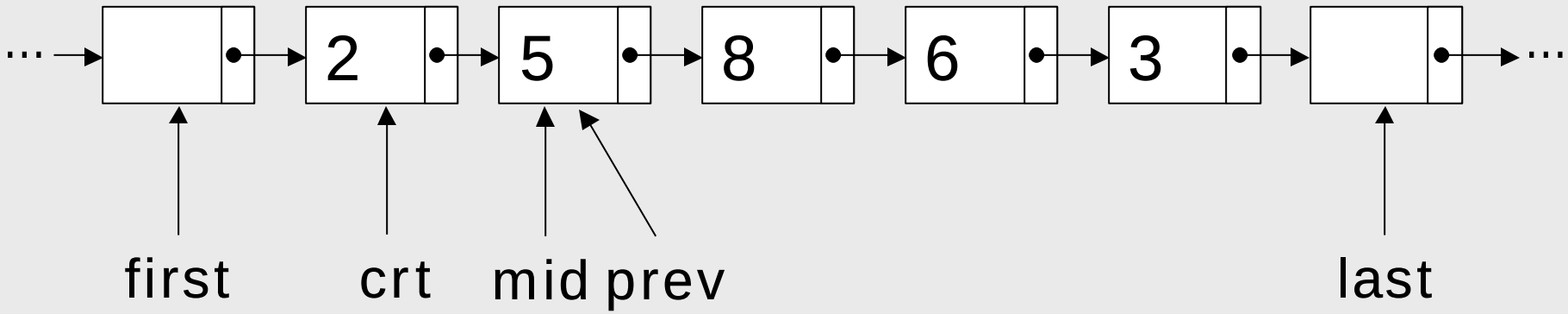
mid.val > crt.val ? Yes!

Quicksort Example: Partitioning



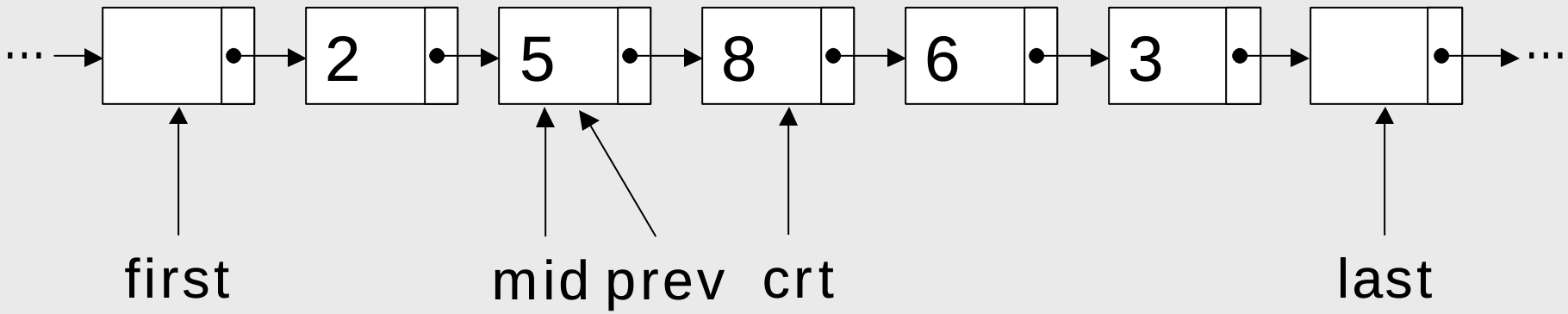
mid.val > crt.val ? Yes!

Quicksort Example: Partitioning



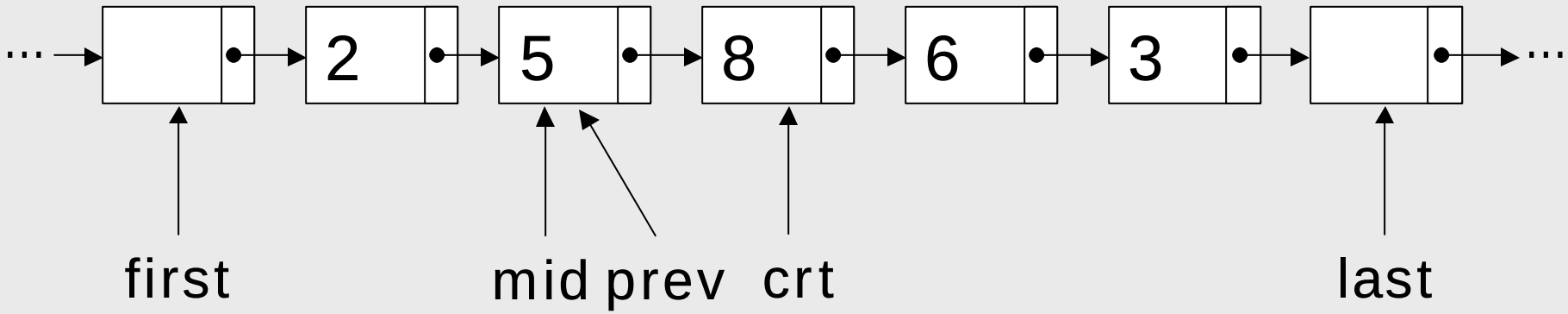
mid.val > crt.val ? Yes!

Quicksort Example: Partitioning



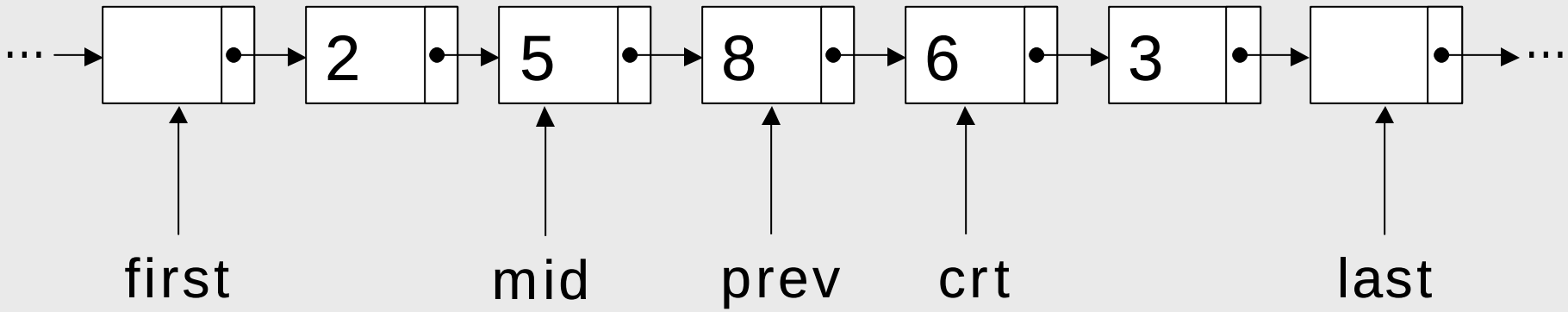
crt = prev->next

Quicksort Example: Partitioning



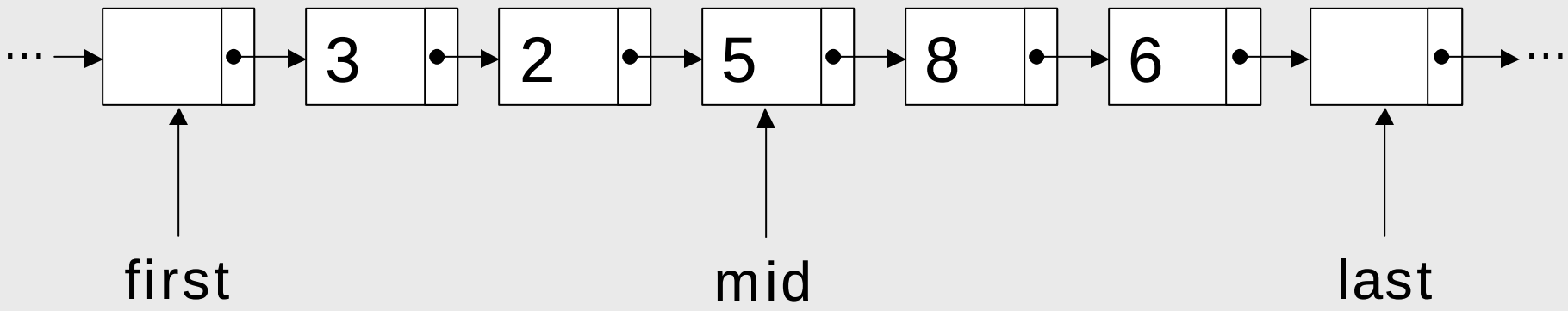
mid.val > crt.val ? No!

Quicksort Example: Partitioning

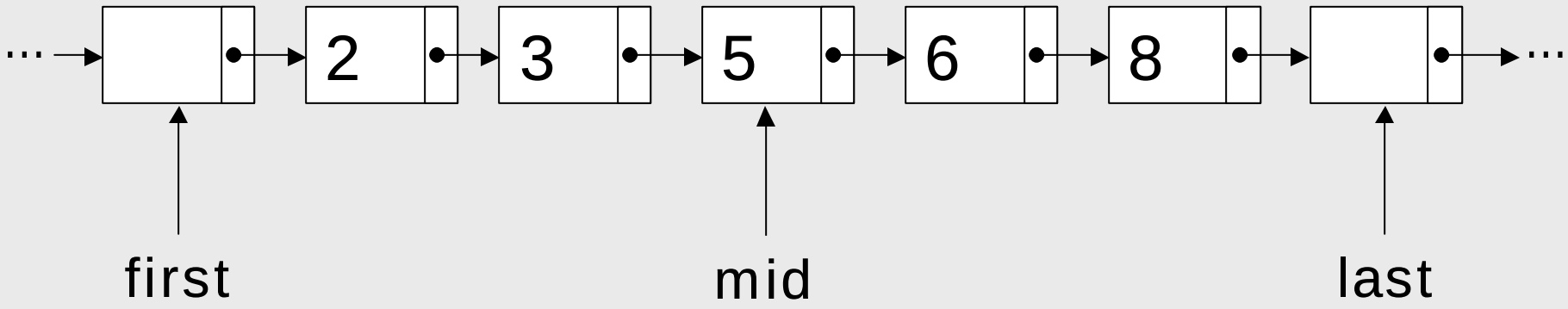


crt = prev->next

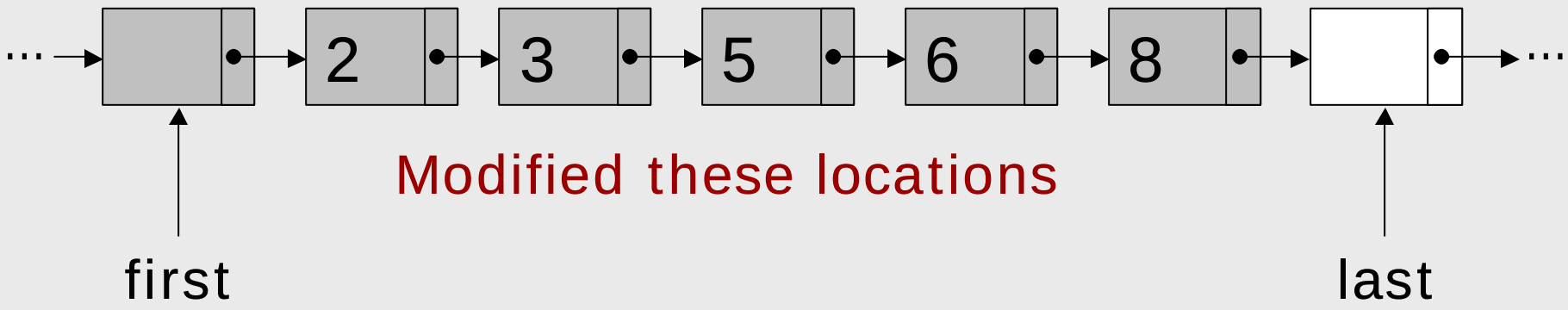
Quicksort Example: Partitioning



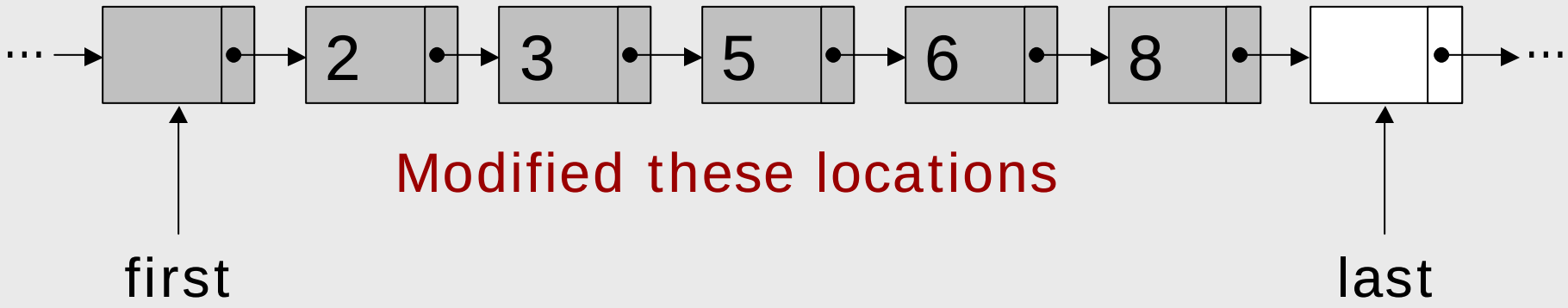
Quicksort Example: Recursing

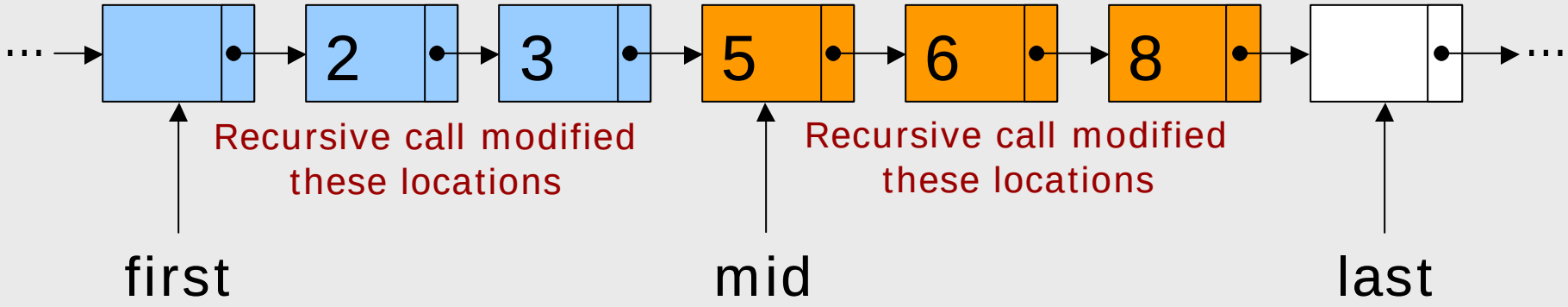


Quicksort Example: Accessed Regions



Quicksort Example: Accessed Regions





Quicksort Example

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

Quicksort Example

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

Base cases

Quicksort Example

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

List partitioning

Quicksort Example

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

Recursive calls

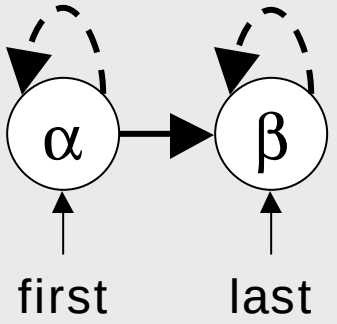
Quicksort Example

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

Goal: Automatically determine that the procedure accesses only the sublist between first and last.

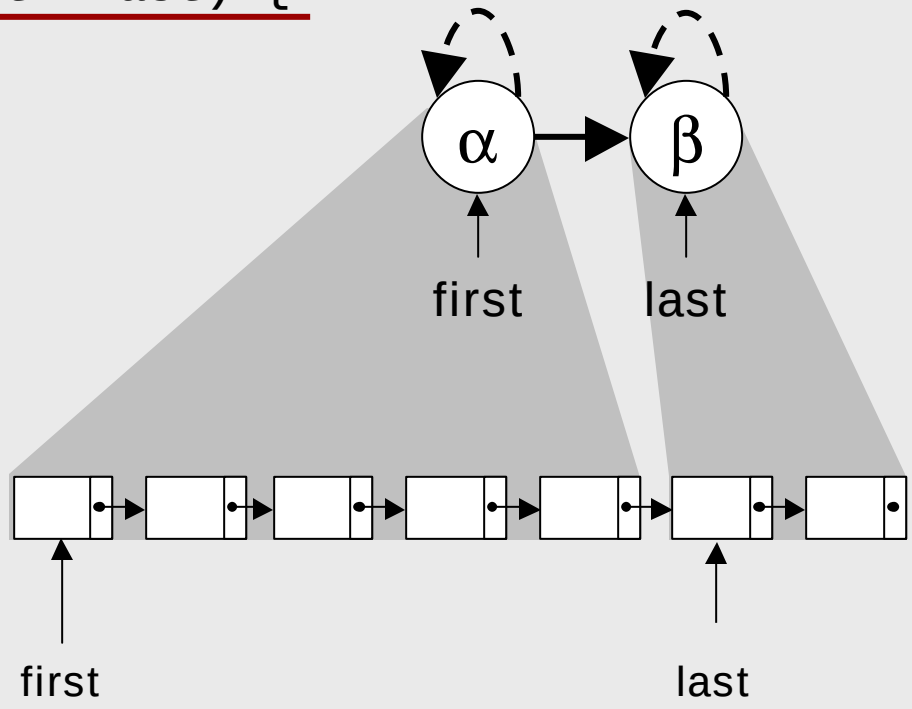
Quicksort Example: Abstraction

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



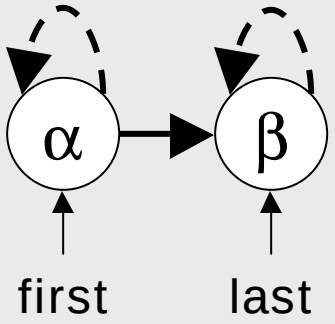
Quicksort Example: Abstraction

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



Quicksort Example: Abstraction

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



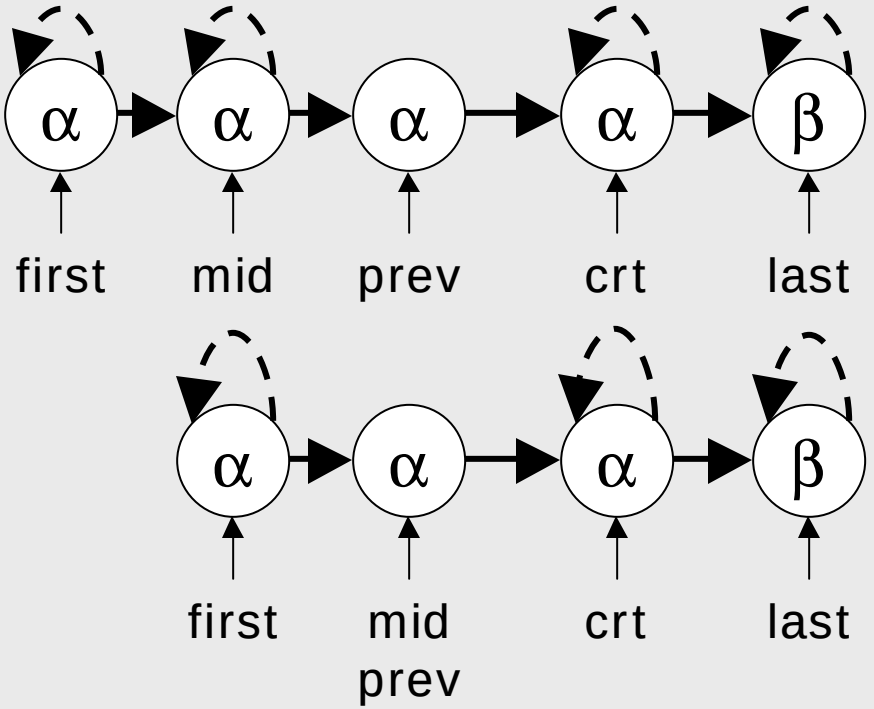
Effects: Reads: α
Writes: α

Quicksort Example: Abstraction

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

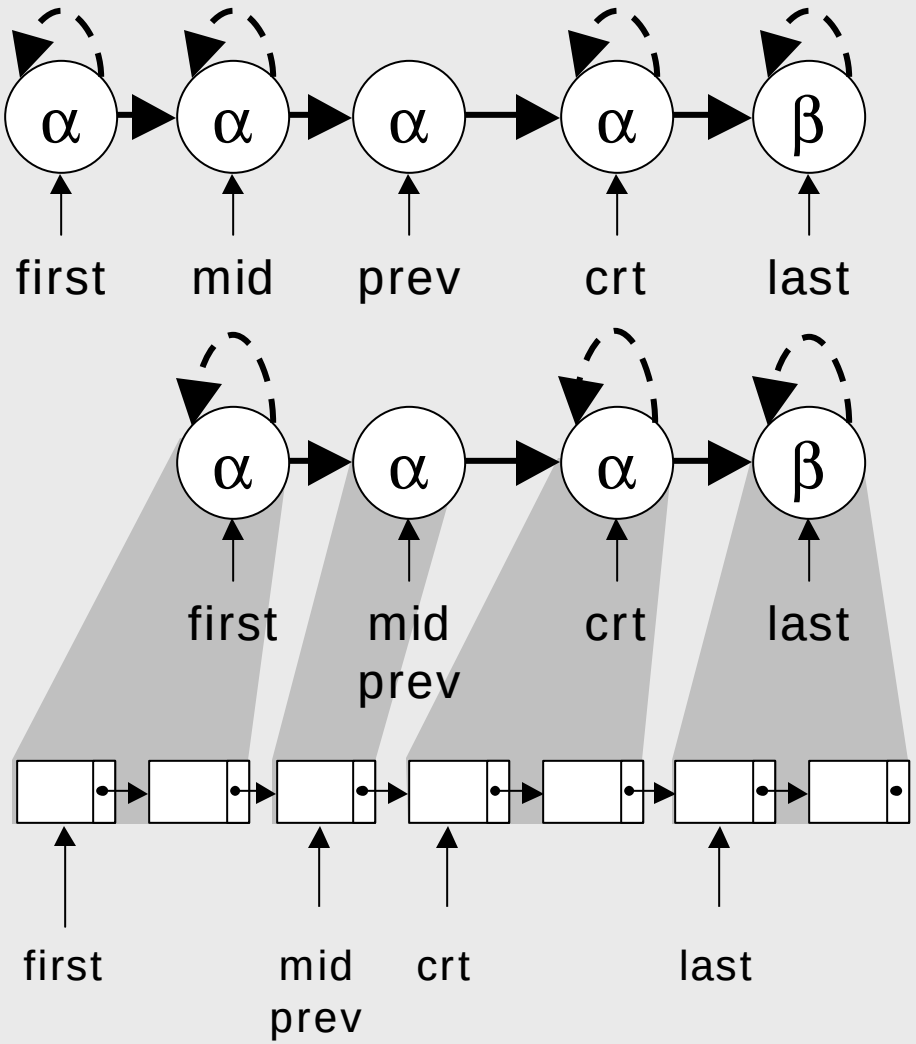
Quicksort Example: Abstraction

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



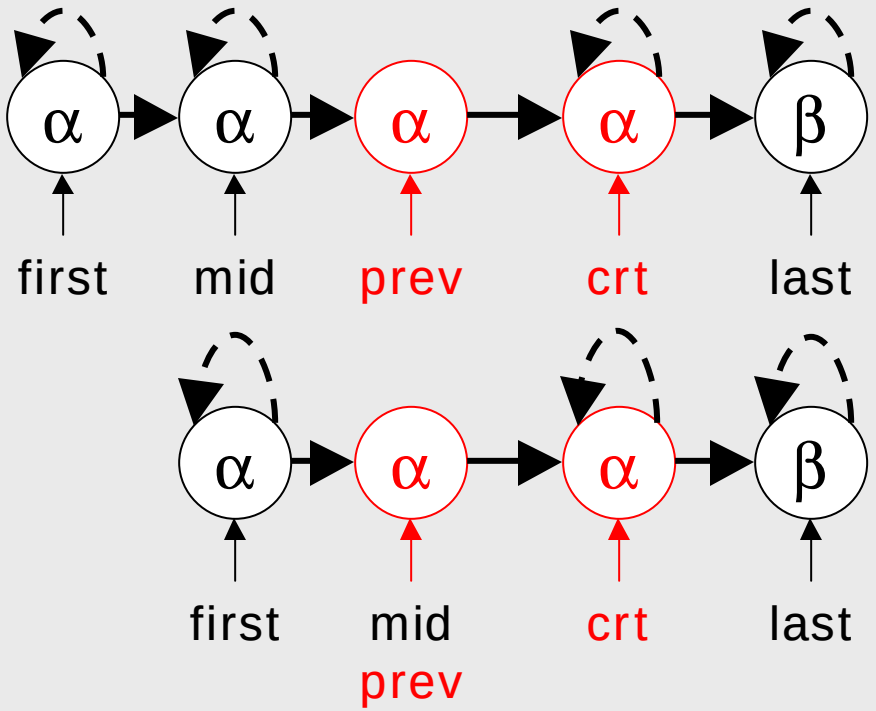
Quicksort Example: Abstraction

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



Quicksort Example: Abstraction

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



Reads: α
Writes: α

Details of the Analysis

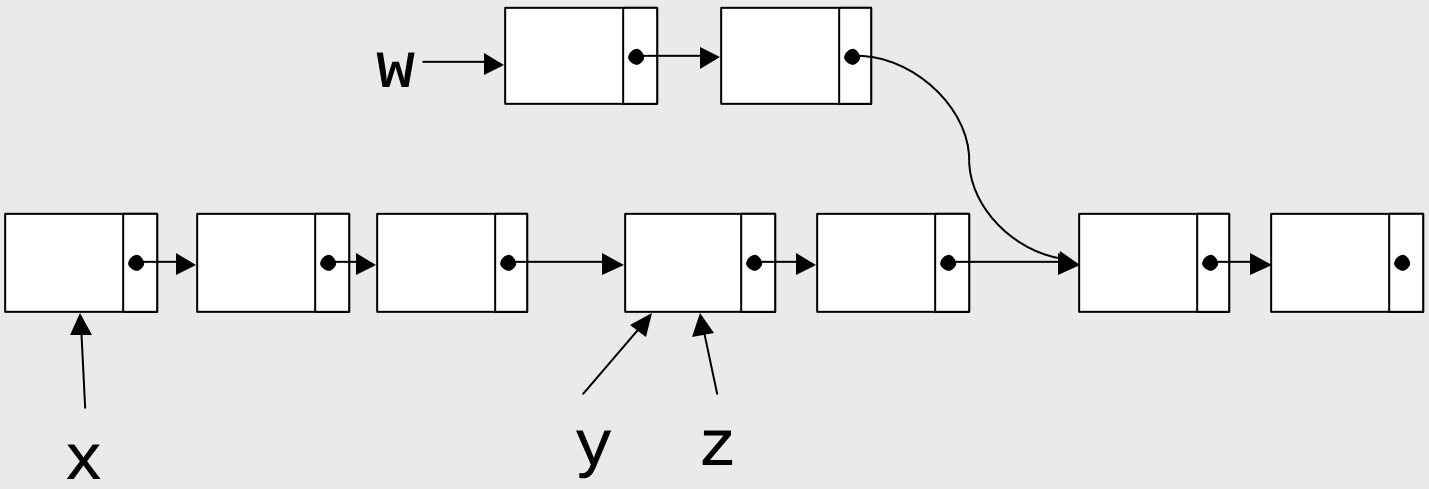
Outline of the Analysis

- Abstraction
- Intraprocedural Analysis
  - Shape Analysis
  - Region Analysis
- Interprocedural Analysis

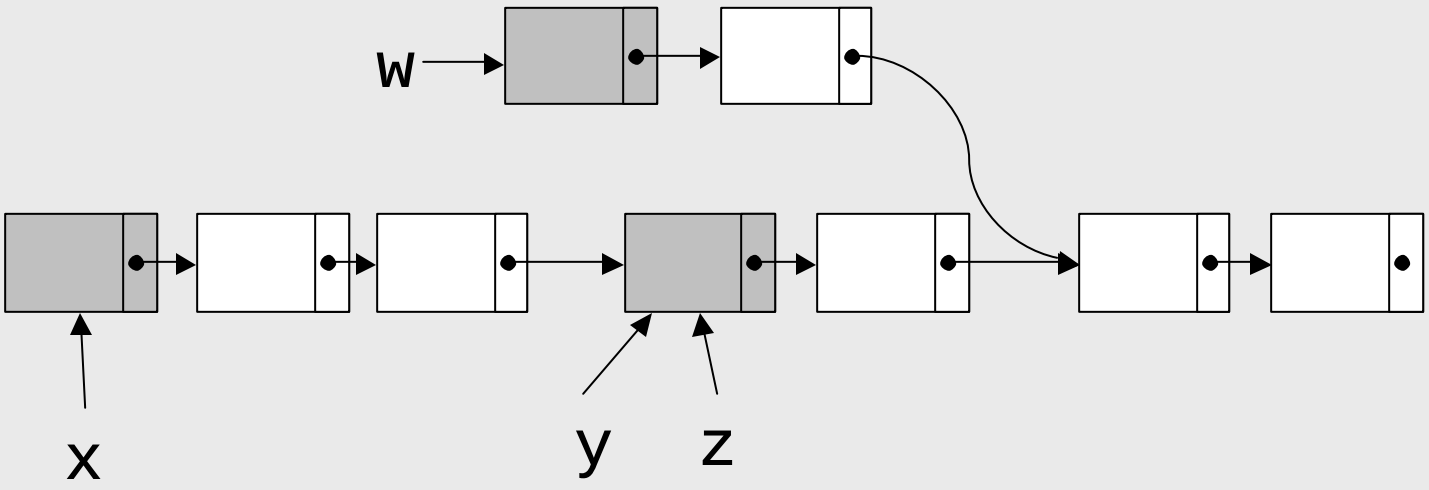
Shape Abstraction

- A heap is:
  - an (unbounded) number of locations
  - Each location may have at most one out-going pointer
  - Stack pointers point to heap locations
- Need a finite abstraction for heaps
  - Uses summary nodes to denote regions
  - Based on reachability from stack pointers

Shape Abstraction Example

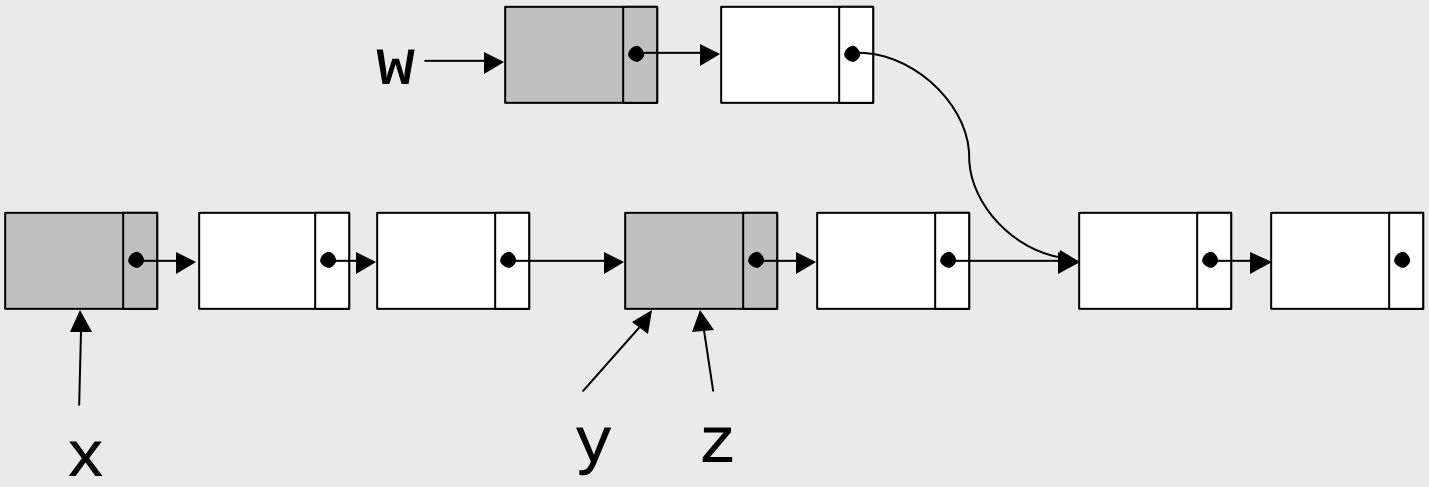


Shape Abstraction Example



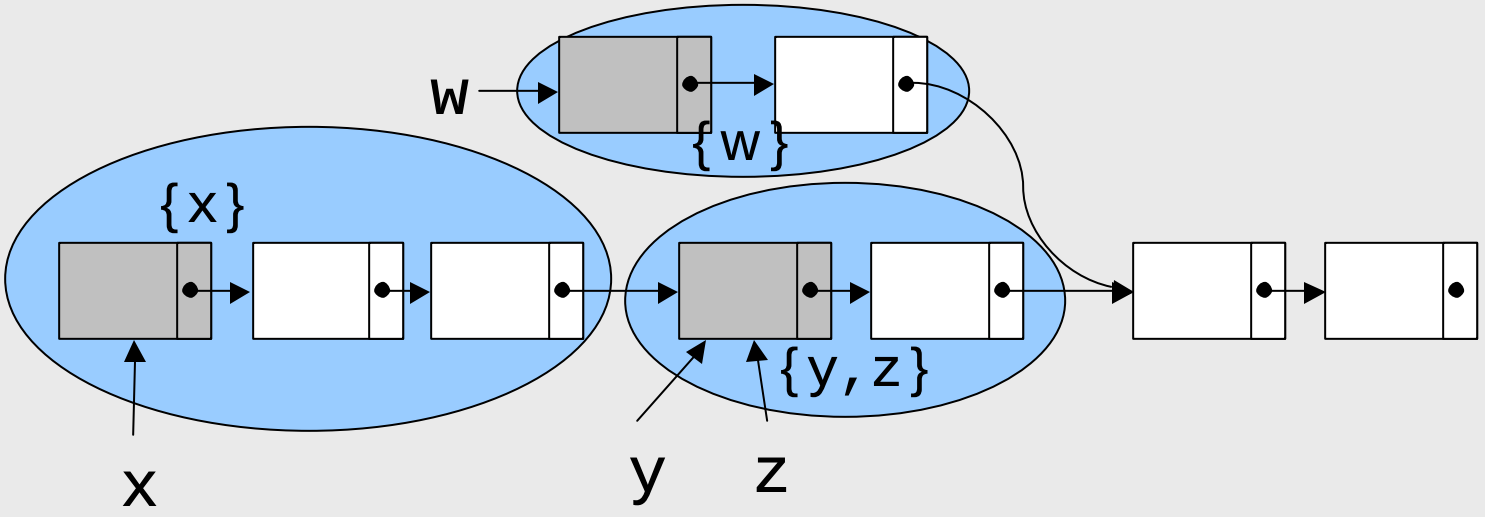
- *Root locations* are immediately pointed to by stack pointers

Shape Abstraction Example



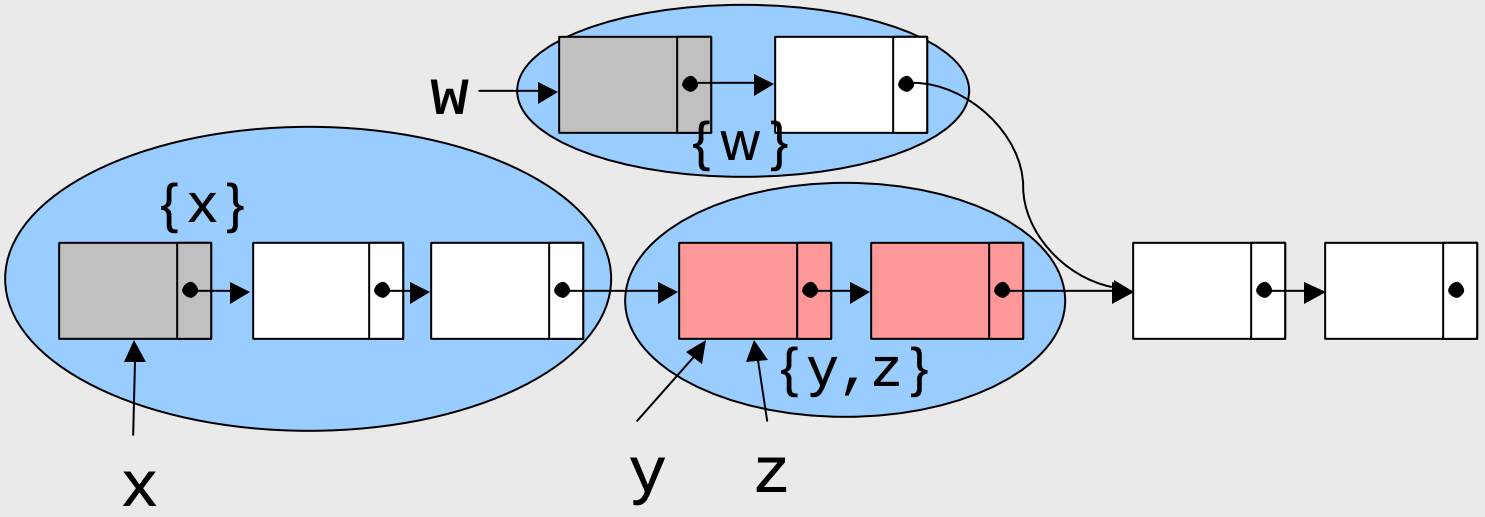
- A location *h* is *owned* by a set of stack pointers *S*, if all paths from a stack pointer to the location *h* must go through the root of *S*

Shape Abstraction Example



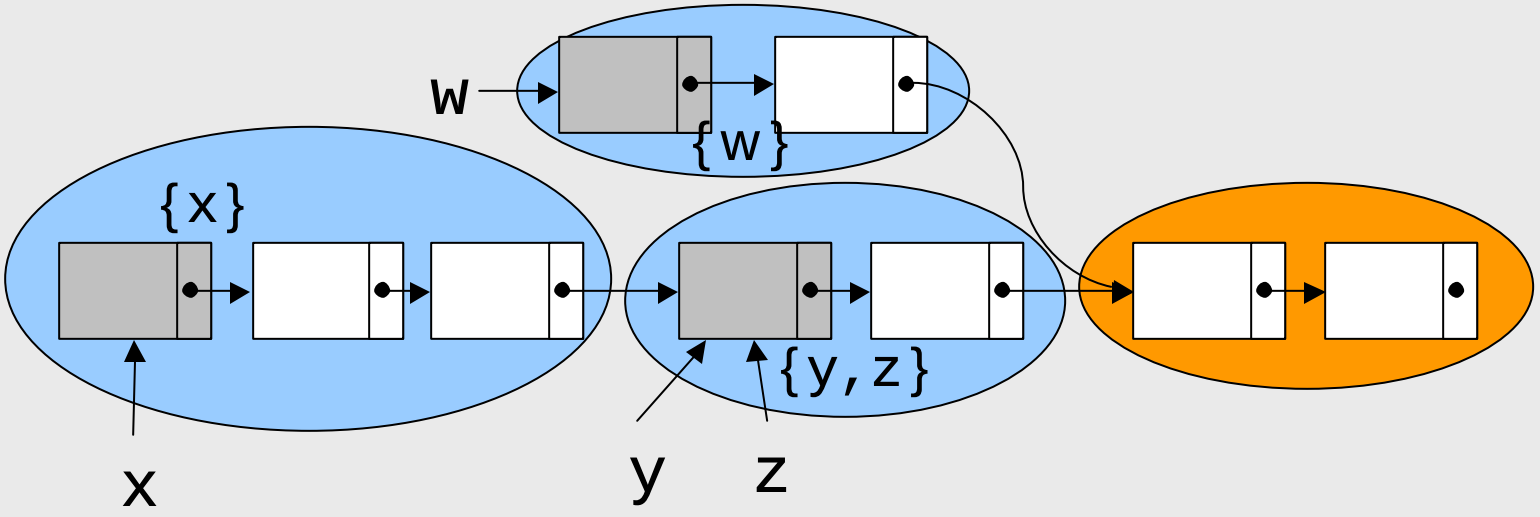
- A location *h* is *owned* by a set of stack pointers *S*, if all paths from a stack pointer to the location *h* must go through the root of *S*

Shape Abstraction Example



- A location *h* is *owned* by a set of stack pointers *S*, if all paths from a stack pointer to the location *h* must go through the root of *S*

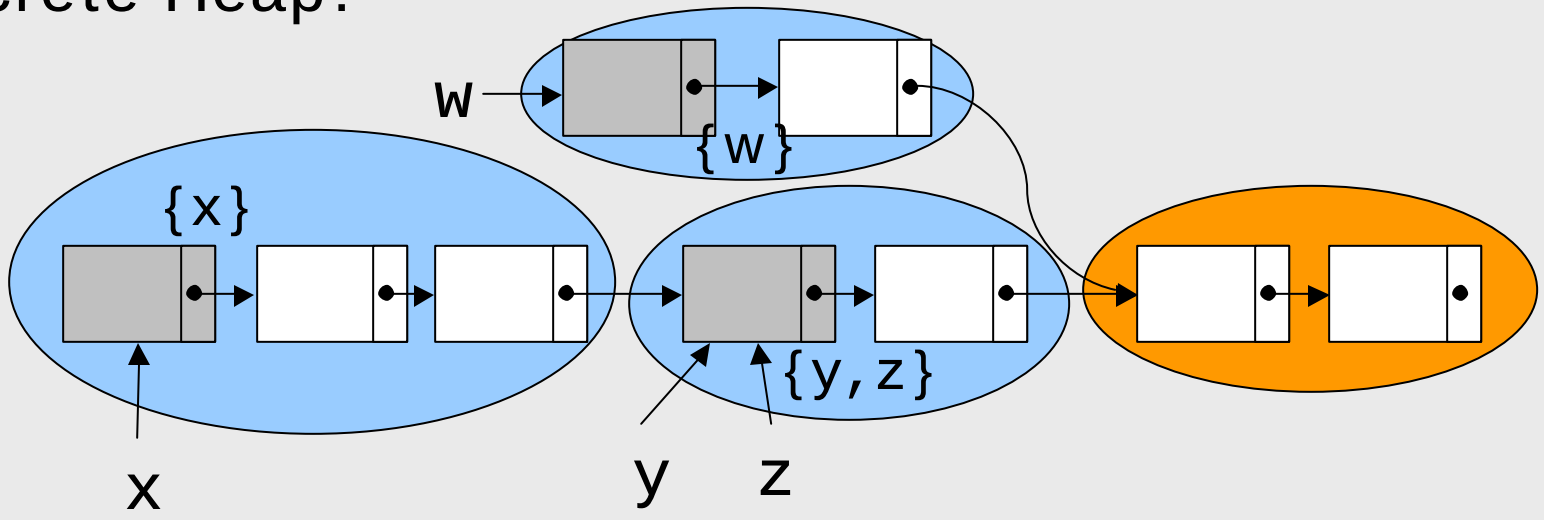
Shape Abstraction Example



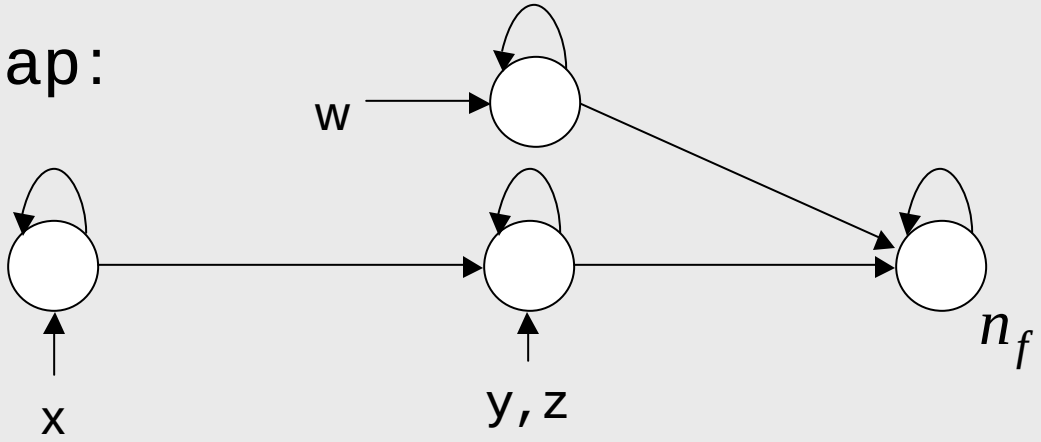
- A location *h* is *owned* by a set of stack pointers *S*, if all paths from a stack pointer to the location *h* must go through the root of *S*

Shape Abstraction Example

Concrete Heap:



Abstract Heap:



Intraprocedural Shape Analysis

Intraprocedural Shape Analysis

- Shape Analysis is formulated as a dataflow analysis
  - Set of shape graphs computed for each program point
- A shape graph is a tuple (N, E, C), where:
  - N: set of summary nodes
  - $E \in N \times N \rightarrow \{0, ½, 1\}$: edges with reachability info
  - $C \in N \rightarrow \{0, ½, 1\}$: cyclicity info for nodes

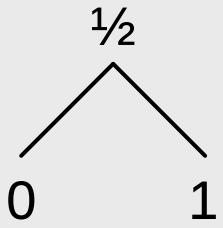
- Transfer functions defined for `x=malloc()`, `x=y->next`, `x=NULL`, `x->next=y`, `x->next=NULL`, `x=y`
- Merge operation defined for shape graphs

Merge Operation

- $(N_1, E_1, C_1) \sqcup (N_2, E_2, C_2) = (N, E, C)$ where:
  - $N = N_1 \cup N_2$
  - $E(x, y) = E_1(x, y) \sqcup_3 E_2(x, y)$ if $x, y \in N_1 \cap N_2$
  - $C(x) = C_1(x) \sqcup_3 C_2(x)$ if $x \in N_1 \cap N_2$

- $\sqcup_3$ is the merge operation for logic values:

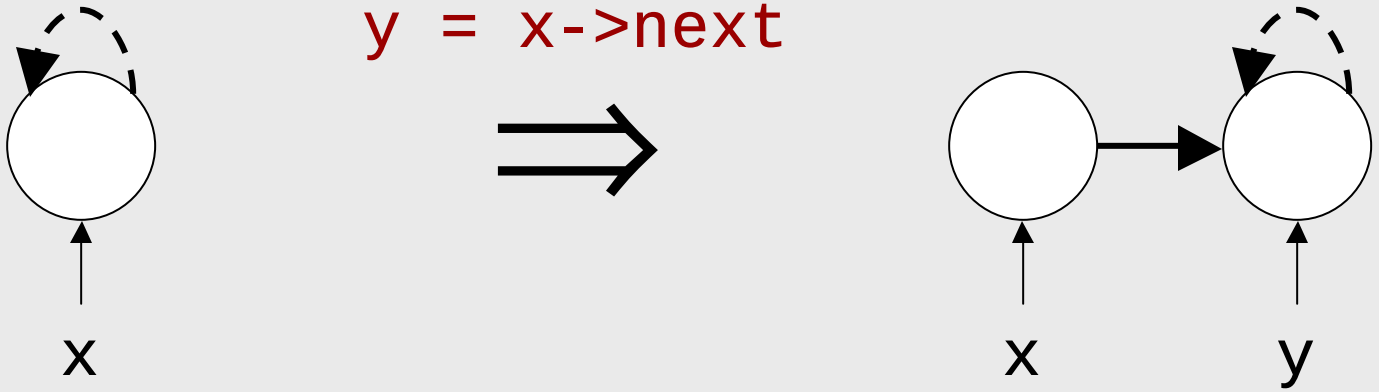
$\sqcup_3$	0	½	1
0	0	½	½
½	½	½	½
1	½	½	1



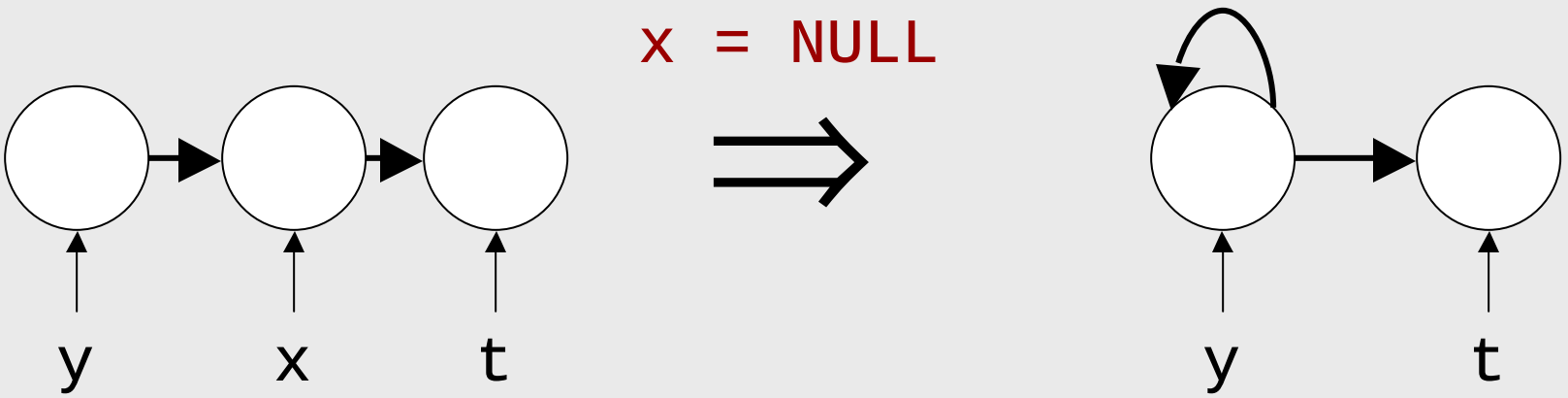
Materialization and Summarization

- Standard shape analysis techniques [Sagiv et al., POPL'96]
- Materialization: creating a new summary node from a summary node
  - a result of traversing a self-edge
  - E.g. `y=x->next`
- Summarization: combining summary nodes together
  - a result of nullifying a stack pointer
  - E.g. `x=NULL`

Materialization

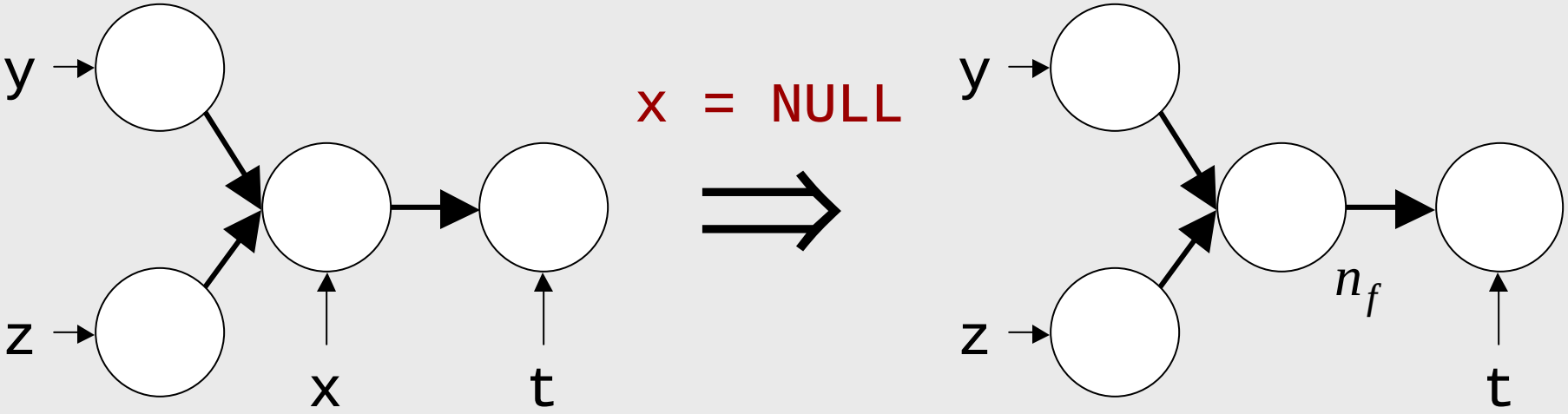


Summarization



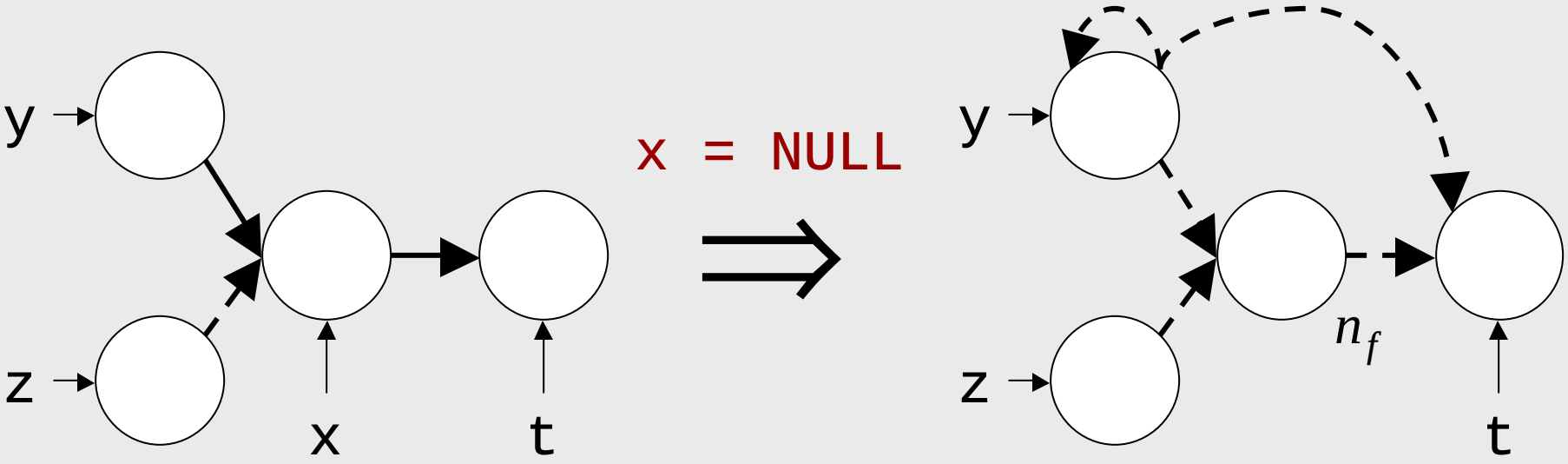
Summarization

- Harder case:



Summarization

- Even harder case:



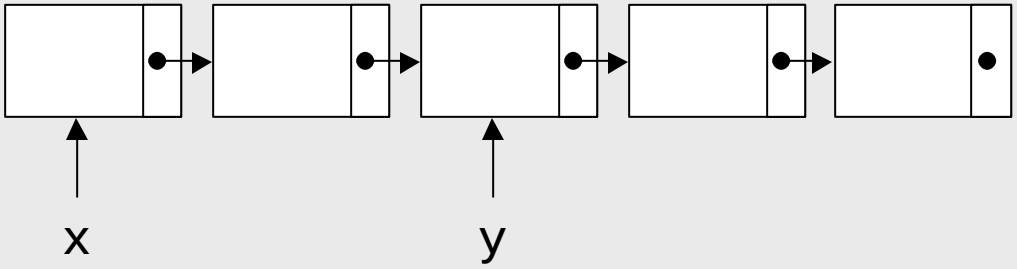
Intraprocedural Region Analysis

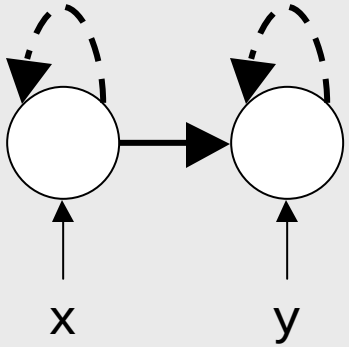
Regions

- Extend shape abstraction to analyze which regions a procedure accesses.
  - Summarize effects of procedures and express results in terms of regions

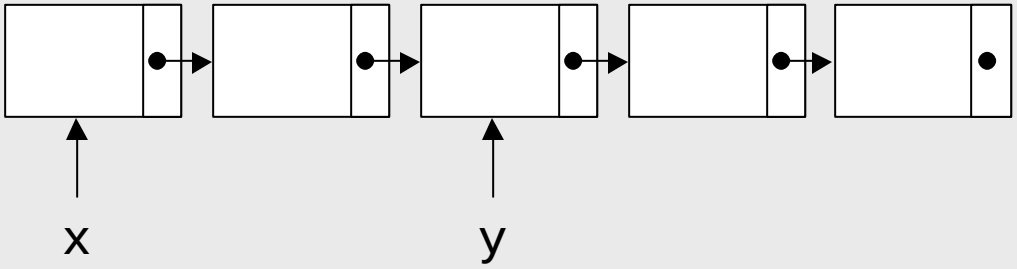
- Problem: summary nodes may represent different heap locations at different program points
  - A heap location may be owned by different stack pointers at different program points

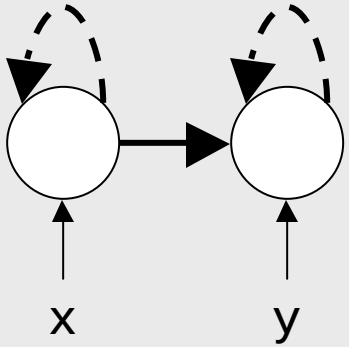
Regions: Problem





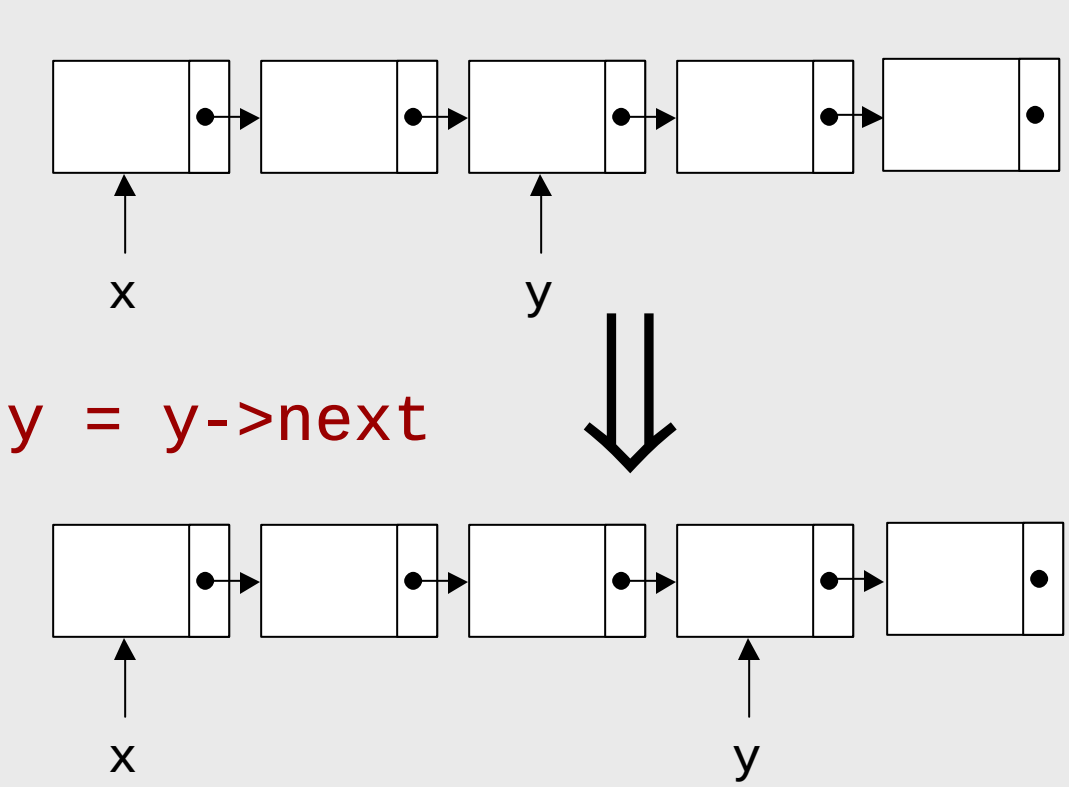
Regions: Problem

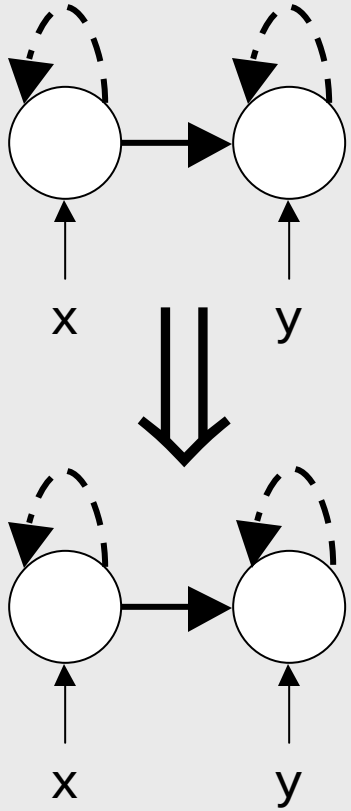




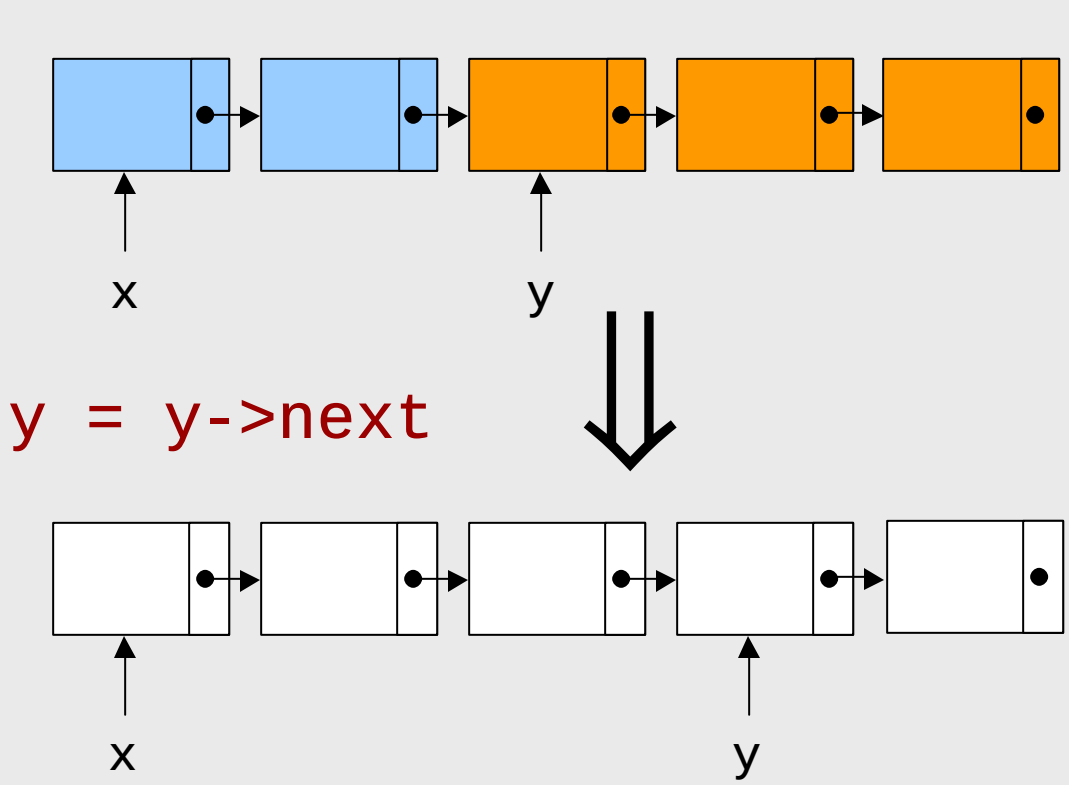

```
y = y->next
```

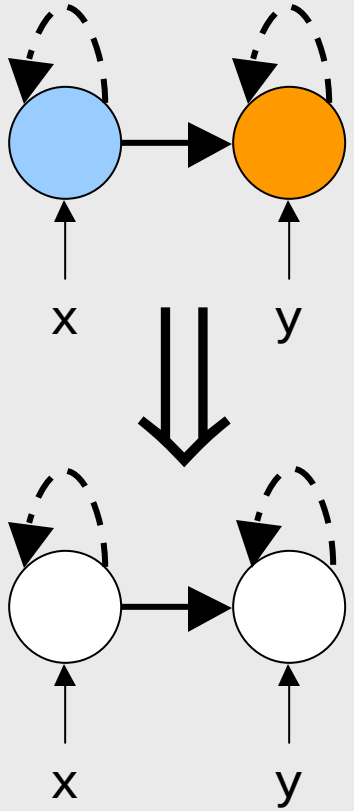
Regions: Problem



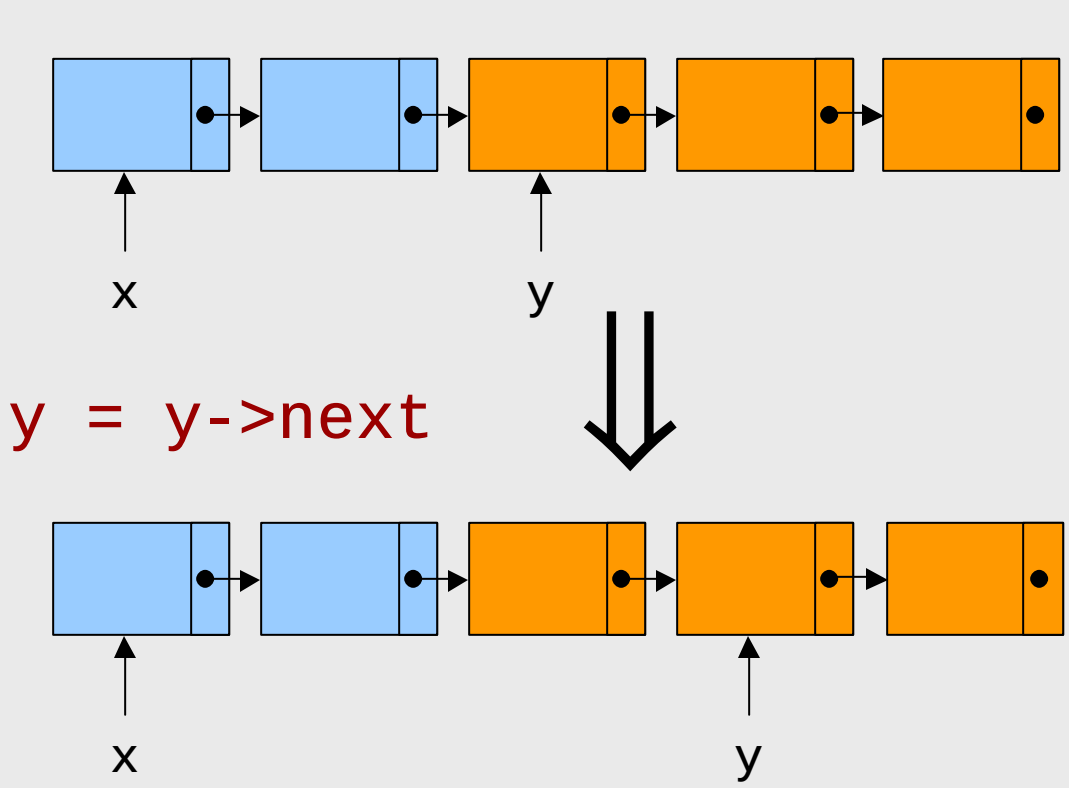


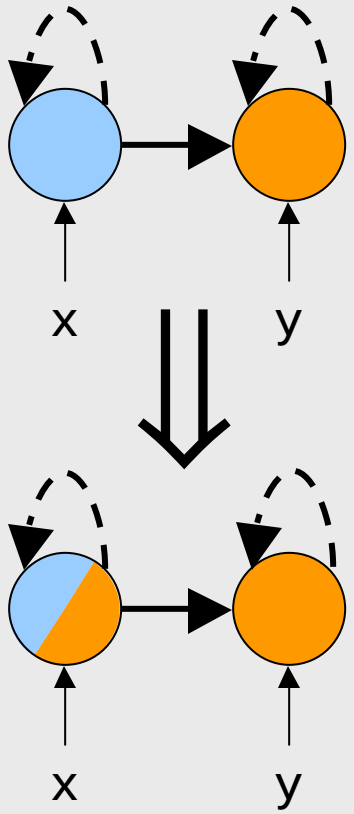
Regions: Problem





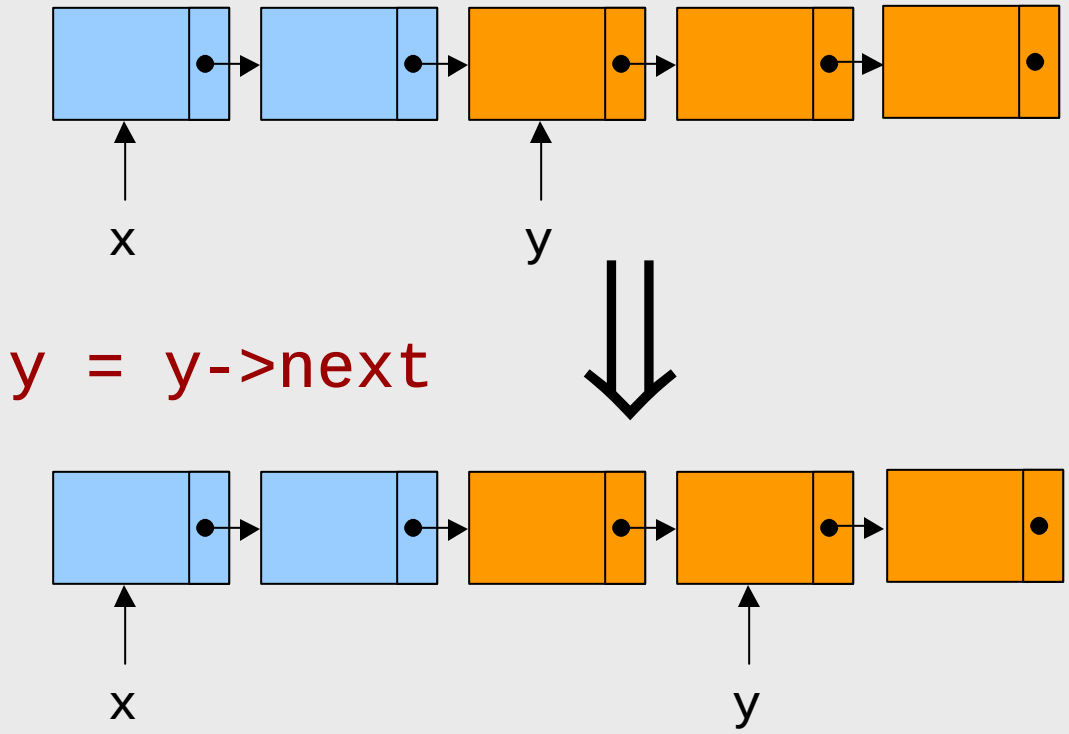
Regions: Problem

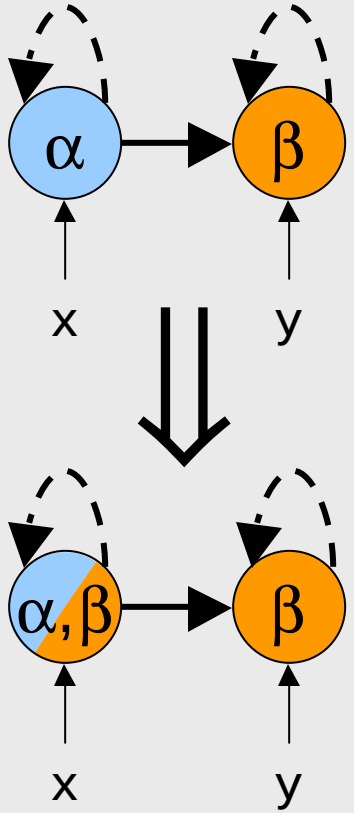




Regions: Solution

- Use labels on summary nodes to indicate the regions they represent.



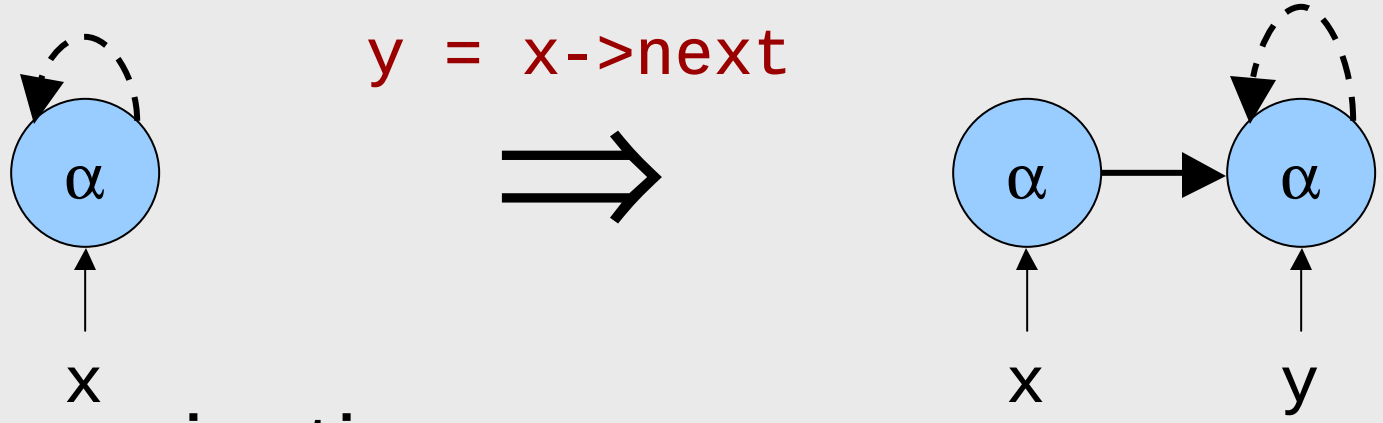


Region Analysis

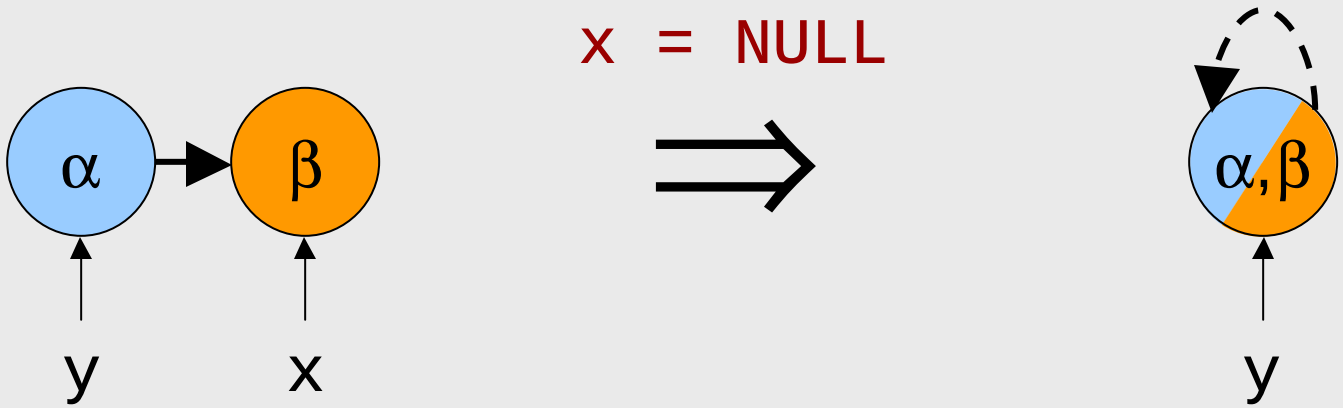
- Fresh region labels are assigned at the start of a procedure, and used throughout the analysis of procedure
  - $\Rightarrow$Region labels on shape graphs refer to regions at the beginning of the procedure
- Transfer functions defined for region labels
  - Interesting cases are materialization and summarization

Region Analysis

- Materialization

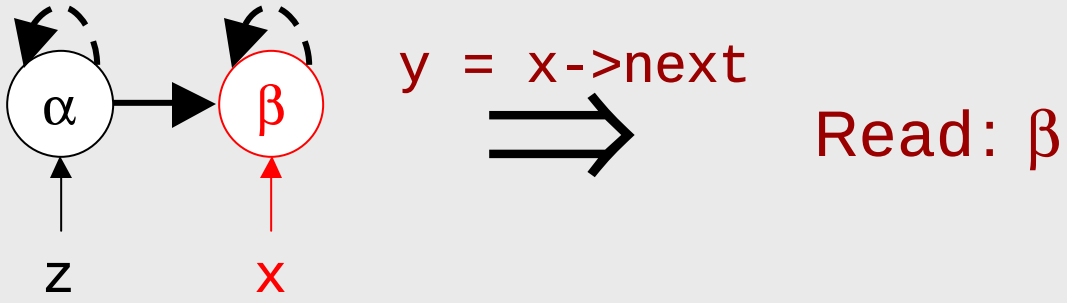


- Summarization



Region Accesses

- Can use the region information to track which regions are read and written by a procedure
  - Write regions:
    - `x->next=NULL, x->next=y`
    - Add the region(s) for the x node to the write set
  - Read regions:
    - `y=x->next`
    - Add the region(s) for x node to the read set
- E.g.



Formal Treatment

- Transfer functions defined for all statements (including materialization and summarization cases)
- Theoretical results:
  - Termination
    - Transfer functions monotonic over a finite height lattice
  - Soundness
    - Transfer functions sound with respect to our abstraction function

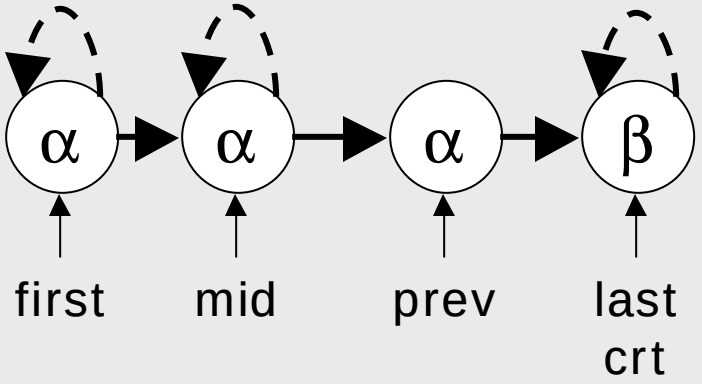
Interprocedural Analysis

Interprocedural Analysis

- Performs context-sensitive interprocedural analysis
- Can handle recursive procedures
- At each call site:
  1. Map current analysis information into name space of invoked procedure
  2. Analyze procedure for the calling context
  3. Unmap results

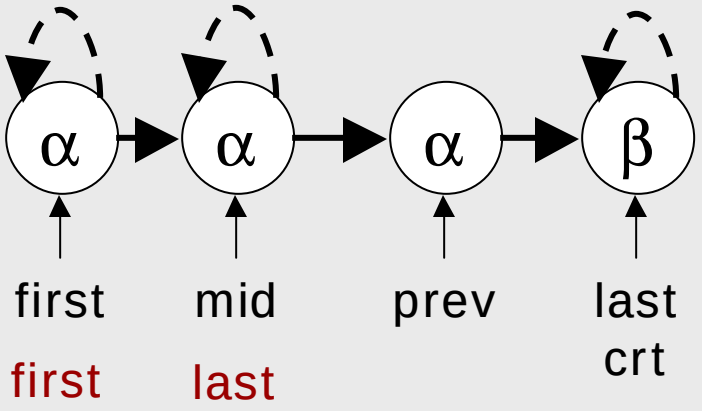
Example

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



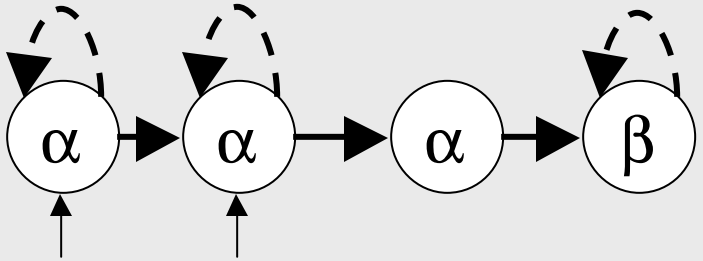
Example: Mapping

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



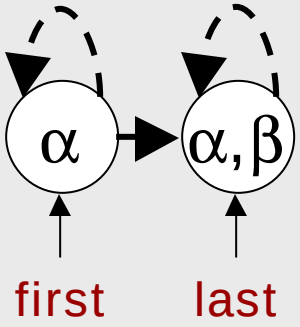
Example: Mapping

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



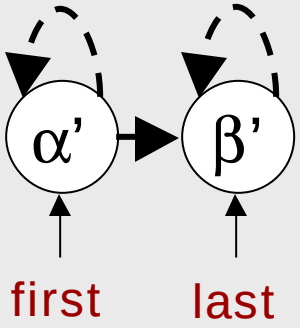
Example: Mapping

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



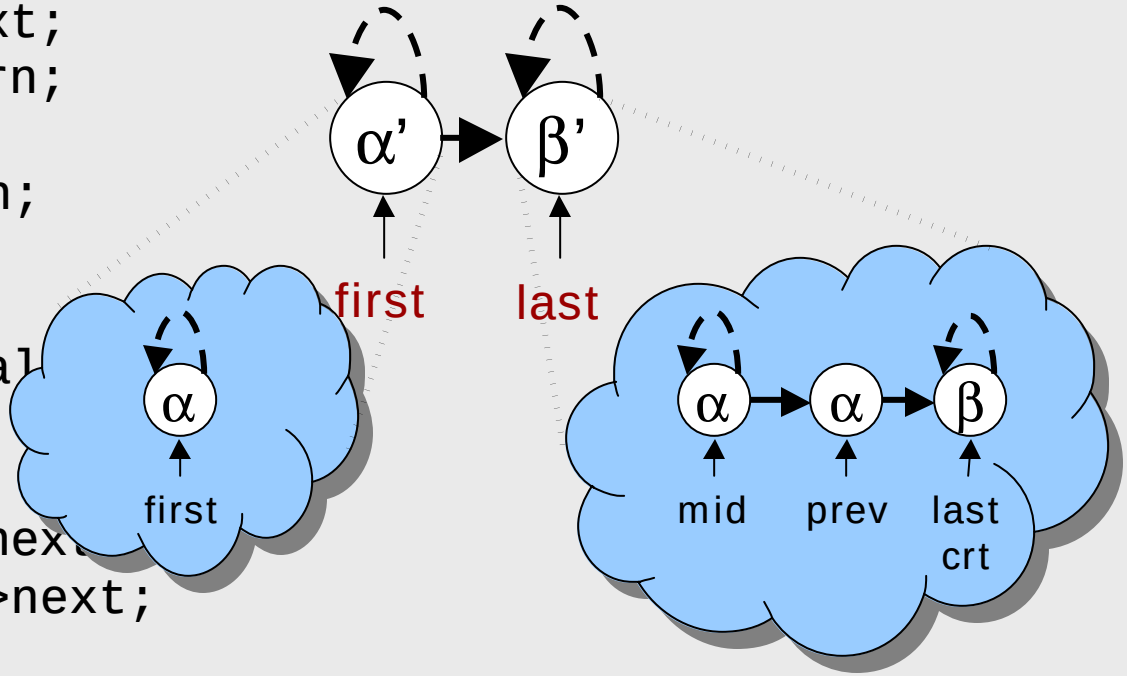
Example: Mapping

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



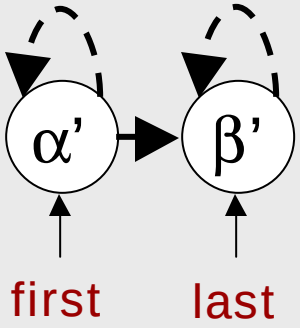
Example: Mapping

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val
      prev = crt;
    } else {
      prev->next = crt->next
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



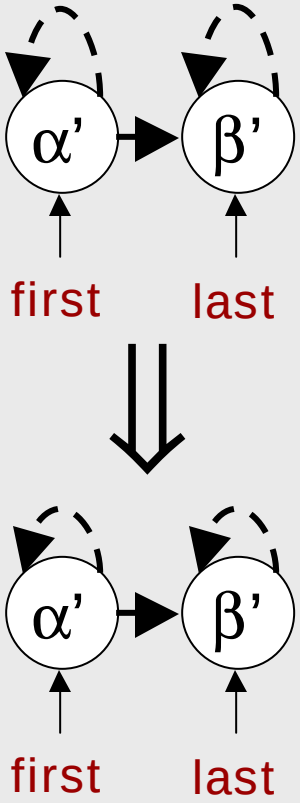
Example: Mapping

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



Example: Analysis

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

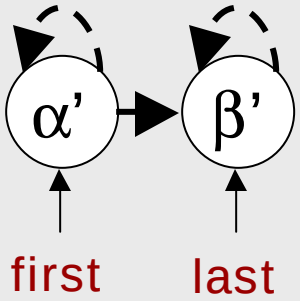


Read = {α'}
Write = {α'}

Example: Unmapping

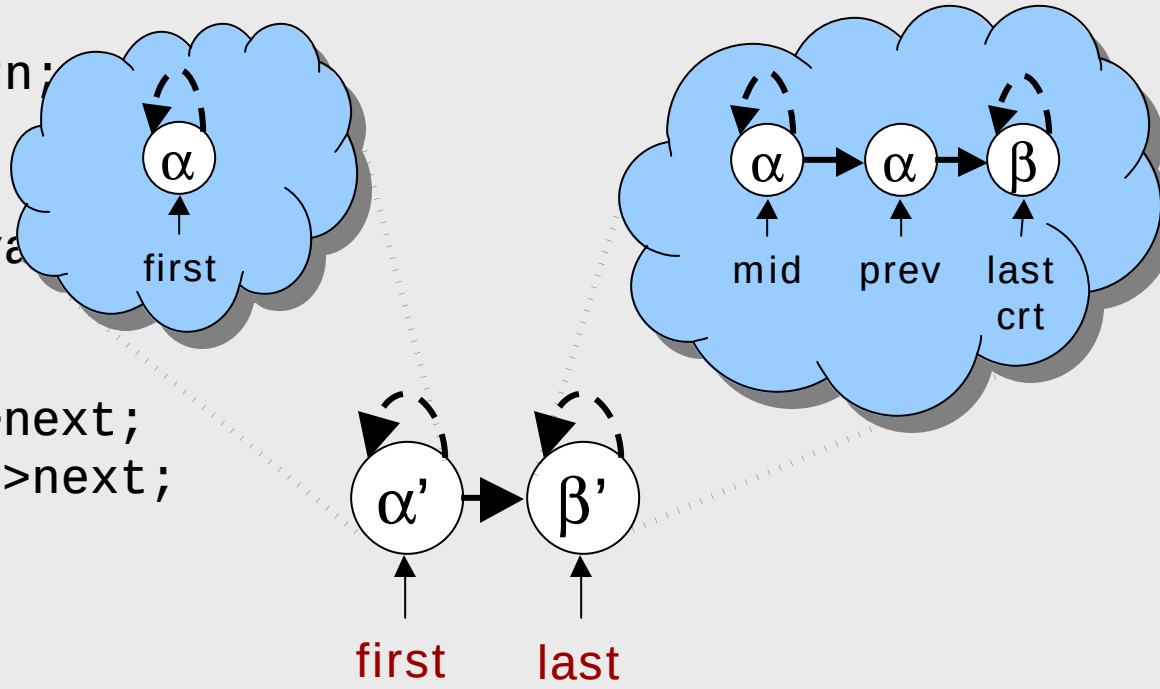
```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

Read = {α'}
Write = {α'}



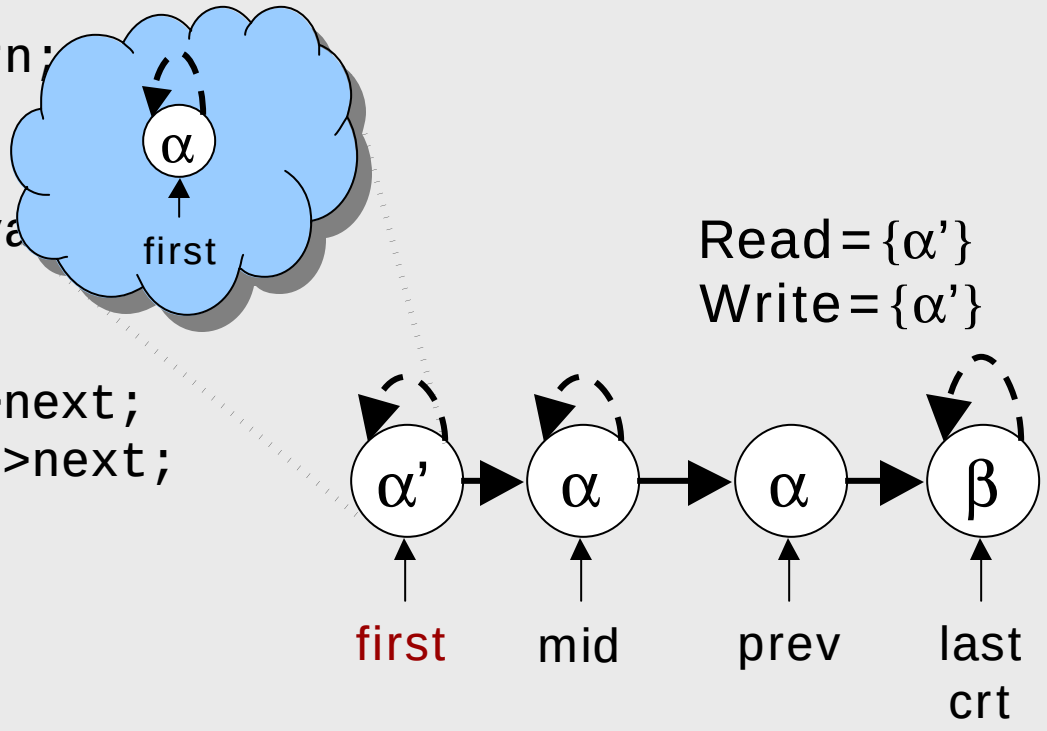
Example: Unmapping

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->va
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



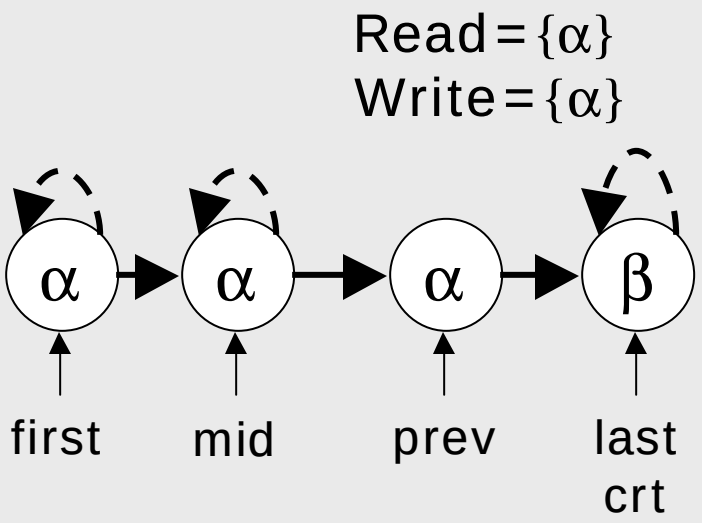
Example: Unmapping

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->va
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



Example: Unmapping

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



Extensions

- Multiple Selectors
  - Extend analysis to deal with more than a single selector name:
    - Annotate edges with selector sets
    - Add cyclicity and sharedness info for selector sets
- Refining the n_f node
  - n_f currently represents all heap locations not owned by any stack pointers
  - Could use different *shared nodes* s_X (X a subset of stack pointers), that represents all heap locations reachable from all roots of pointers in X

Analysis Uses

- Parallelization
  - Statements accessing disjoint heap regions can be executed in parallel
- Program Understanding
  - The shape graph and region output of the analysis can aid understanding of the effect of procedures on heap structures
- Correctness
  - Analysis can verify programmer-supplied specifications

Using Analysis for Correctness

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while ( crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

Using Analysis for Correctness

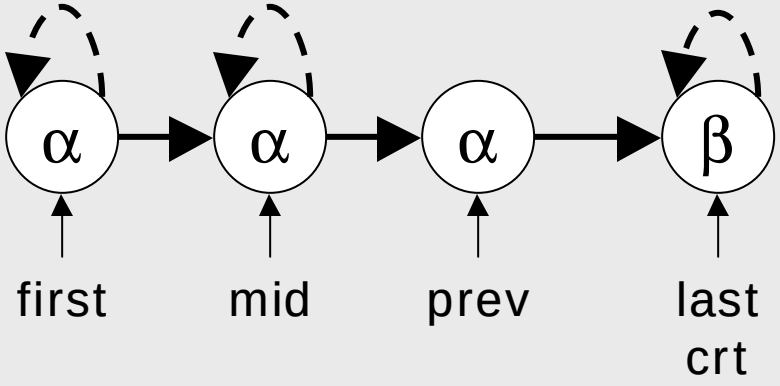
```
void quicksort(list *first, list *last) Reads: a  Writes: a {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while ( crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

Using Analysis for Correctness

```
void quicksort(list *first, list *last) Reads: a  Writes: a {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (prev != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```

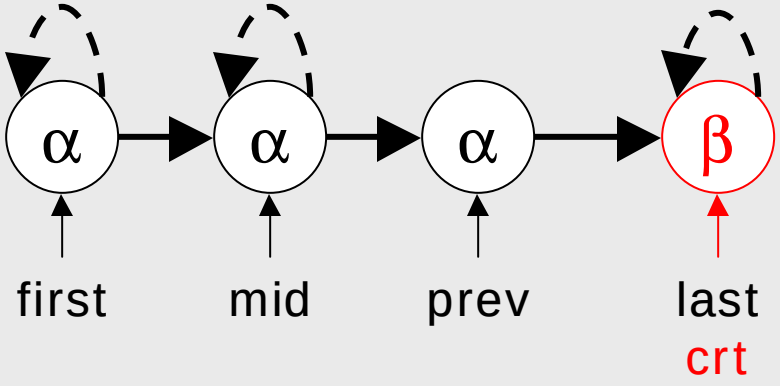
Using Analysis for Correctness

```
void quicksort(list *first, list *last) Reads: a  Writes: a {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (prev != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



Using Analysis for Correctness

```
void quicksort(list *first, list *last) Reads: a  Writes: a {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (prev != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



Read = {α, β} !
Write = {α, β} !

Related Work

Related Work

- Shape Analysis
  - [Horwitz,Pfeiffer,Reps, PLDI'89], [Chase,Wegman,Zadek,PLDI'90], [Ghiya,Hendren,POPL96], [Sagiv,Reps,Wilhelm,TOPLAS'98, TOPLAS'02], With reachability:[Dor,Rodeh,Sagiv,SAS'00]
  - Interprocedural:[Rinetzky,Sagiv,CC'01], [Kuncak,Rinard,POPL02]

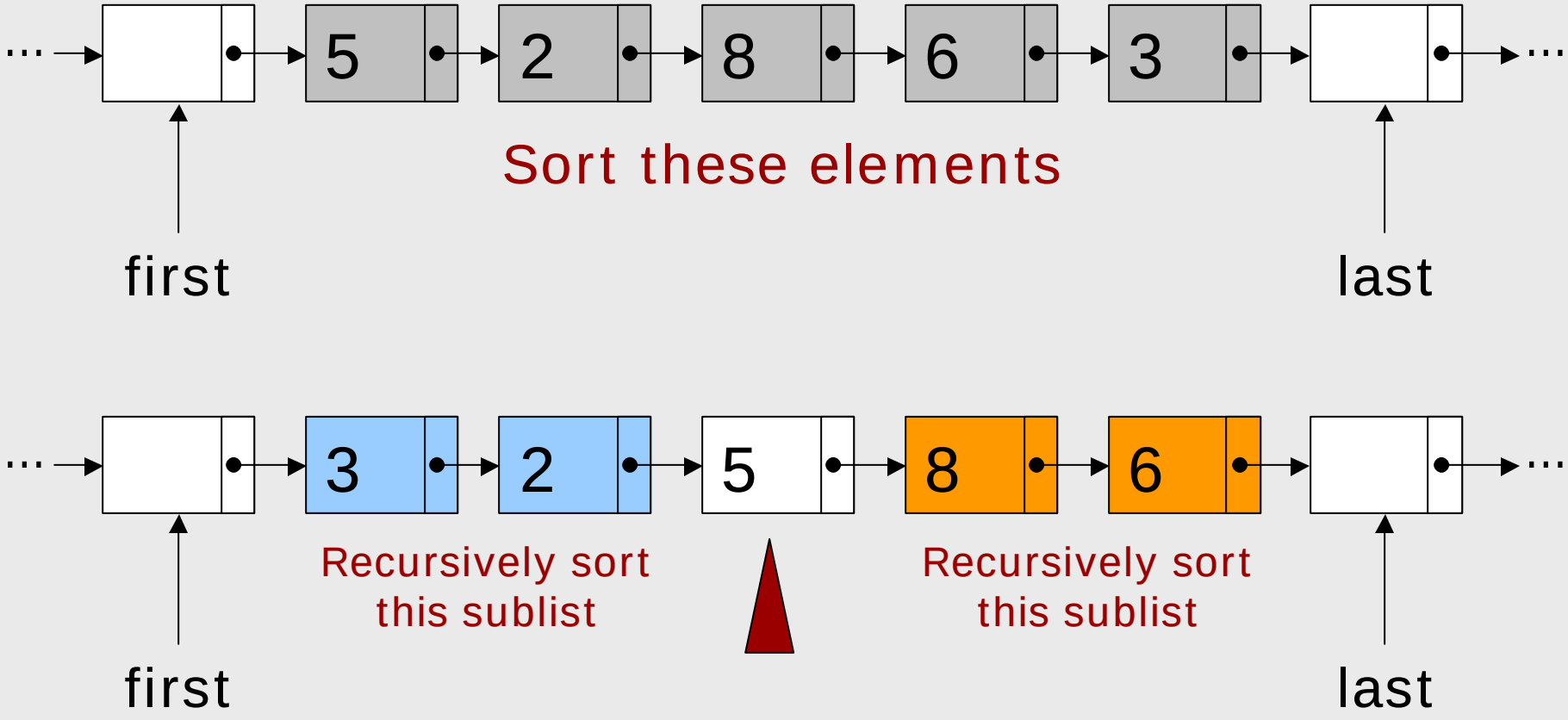
- Regions
  - Language support: RC[Gay,Aiken,PLDI'98], Vault[DeLine,Fahndrich,PLDI'01], Cyclone[Grossman et.al.,PLDI'02]
  - Region Inference: [Tofte,Talpin, POPL'94], [Lattner,Adve MSP'02]

Conclusions

- Analysis of accessed regions in recursive data structures
- Regions = sublists, subtrees, etc.
- Dataflow analysis formulation
- Interprocedural analysis
- Applies to recursive programs with destructive heap updates

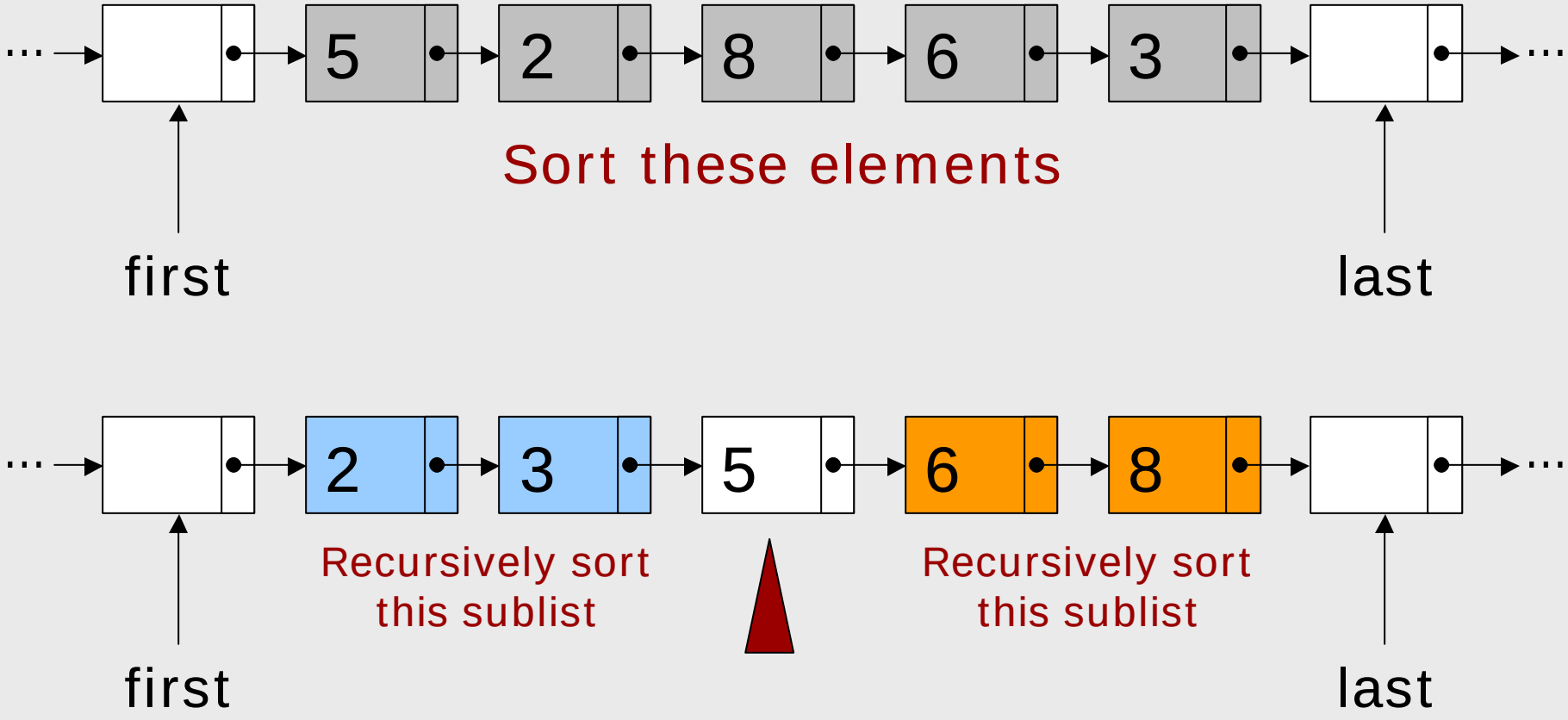
Quicksort Example

- Chooses a pivot value
- Partitions list into sublists destructively

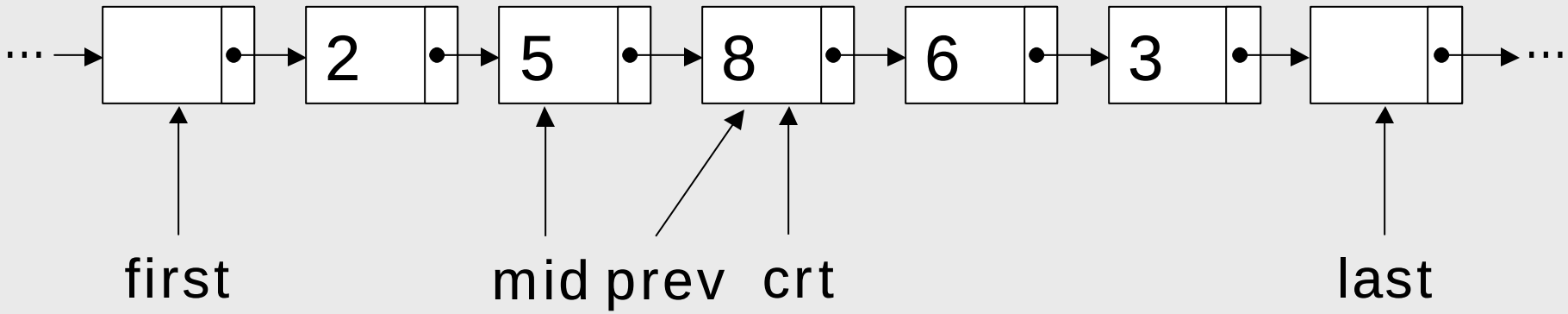


Quicksort Example

- Chooses a pivot value
- Partitions list into sublists destructively



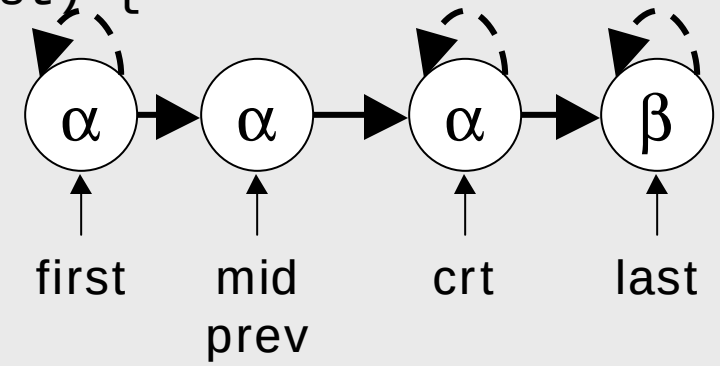
Quicksort Example: Partitioning

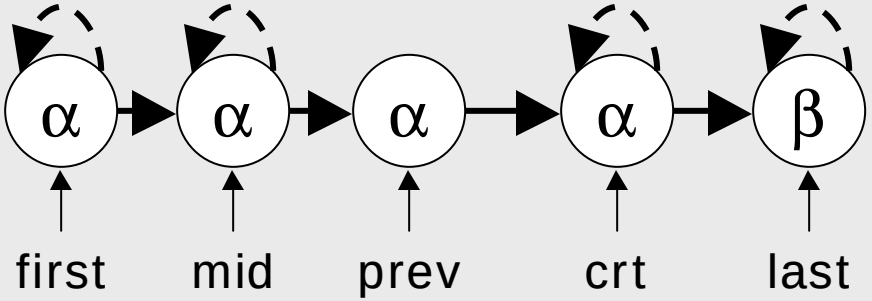


mid.val > crt.val ? No!

Quicksort Example: Abstraction

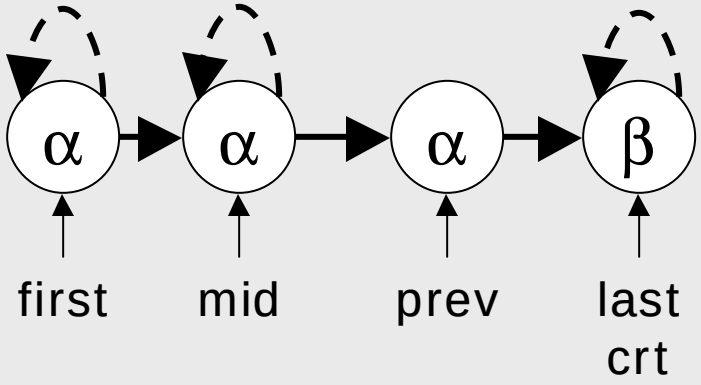
```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```





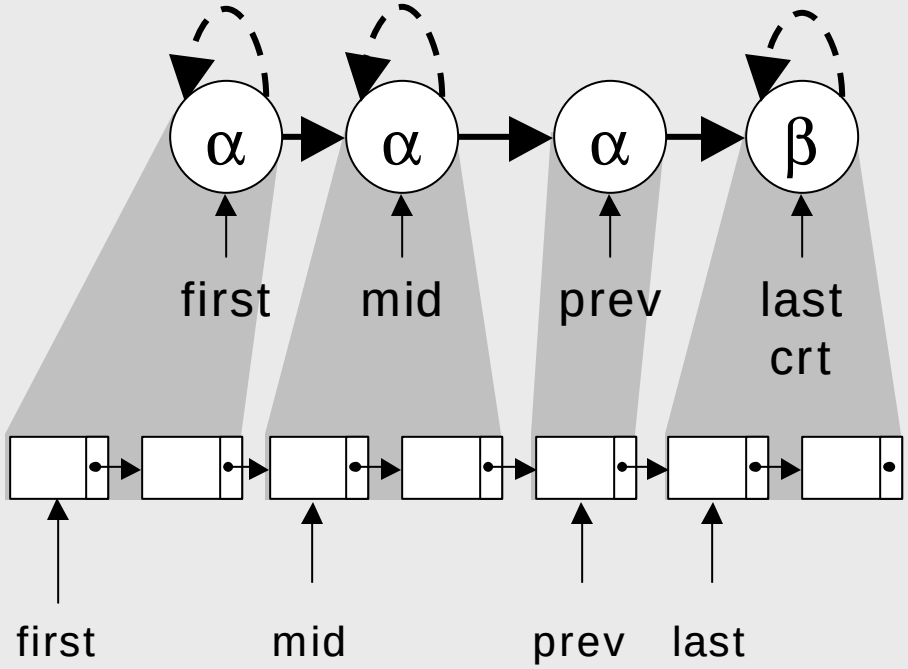
Quicksort Example: Abstraction

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



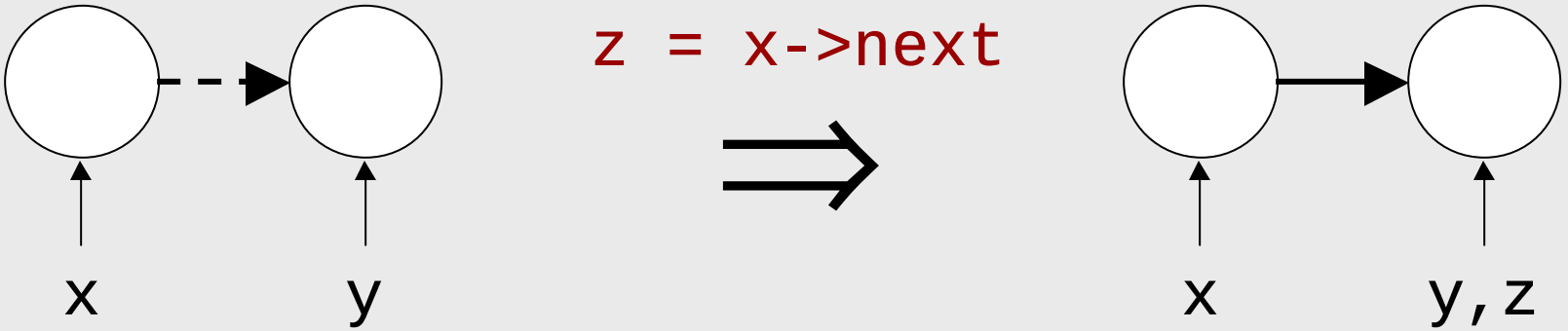
Quicksort Example: Abstraction

```
void quicksort(list *first, list *last) {
  list *mid, *crt, *prev;
  mid = prev = first->next;
  if (mid == last)  return;
  crt = prev->next;
  if (crt == last) return;

  while (crt != last) {
   if (crt->val > mid->val) {
      prev = crt;
    } else {
      prev->next = crt->next;
      crt->next = first->next;
      first->next = crt;
    }
    crt = prev->next;
  }
  quicksort(first, mid);
  quicksort(mid, last);
}
```



Traversing May-Edges

- Traversing a may-edge makes it a must-edge
  - E.g.



  - In any execution where z=x->next succeeds, then the root of z is definitely reachable from the root of x

Related Work

- Effect Systems
  - FX-87[Gifford,Jouvelot,Lucassen, POPL88],
  - Broadway[Guyer,Lin,LCPC'00,SAS'03],
  - Array accesses[Rugina,Rinard,CC'01],
  - Cyclone[Morrisett et. al.,USENIX'02],
  - Roles[Kuncak,Rinard, POPL'02]