A Concurrent Program Logic with a Future and History

Roland Meyer, Thomas Wies, Sebastian Wolff

[OOPSLA'22]

Motivation

"***Programs = Algorithms + Data Structures***"

–Niklaus Wirth, 1978

Motivation

"***Programs = Algorithms + Data Structures***"

–Niklaus Wirth, 1978

Verification via Separation Logic (SL)

Motivation

"***Programs = Algorithms + Data Structures***"

–Niklaus Wirth, 1978

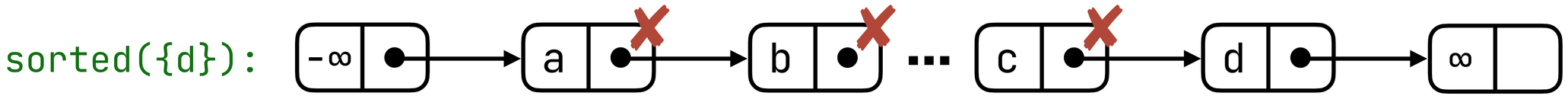
Verification via Separation Logic (SL)

Automation

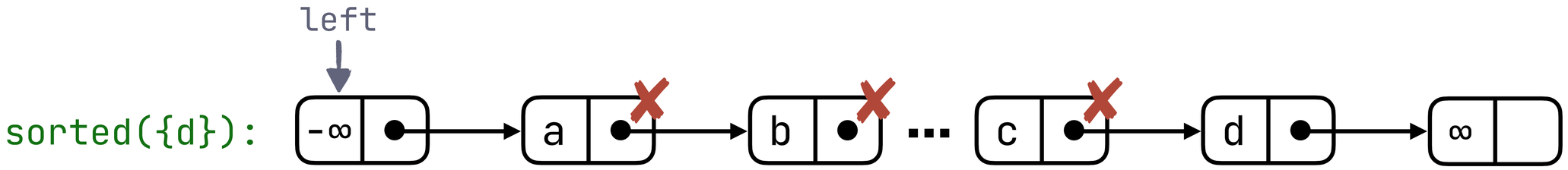
Verification Challenge 1: **Complex Unbounded Updates**

Verification Challenge 2: **Linearizability**

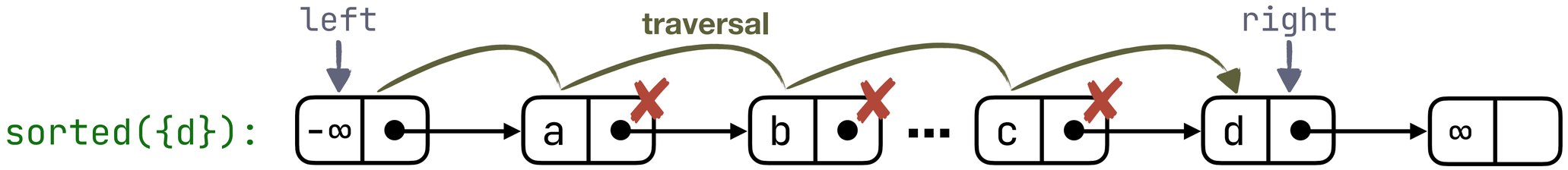
Example: Unbounded Removal



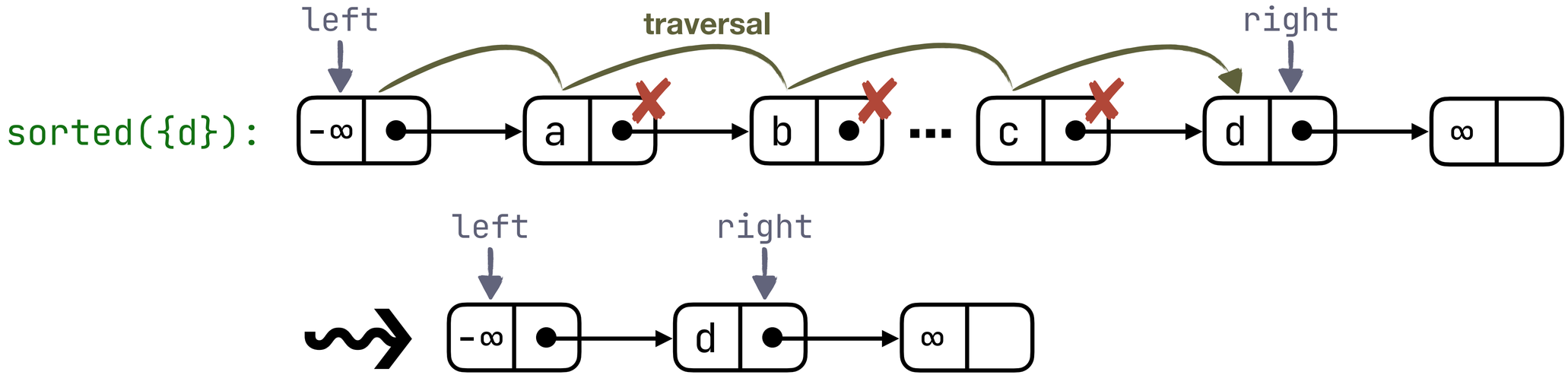
Example: Unbounded Removal



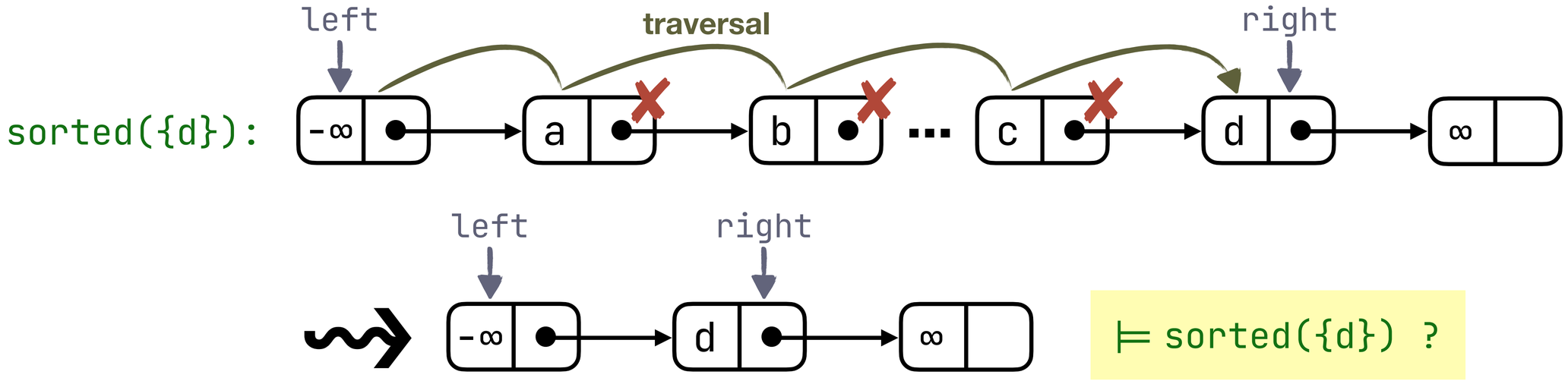
Example: Unbounded Removal



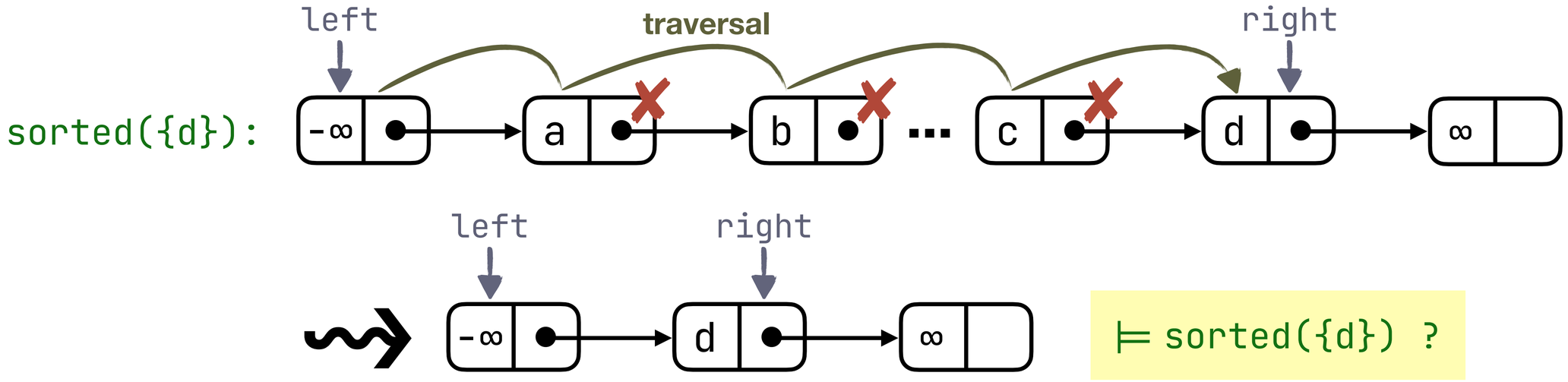
Example: Unbounded Removal



Example: Unbounded Removal



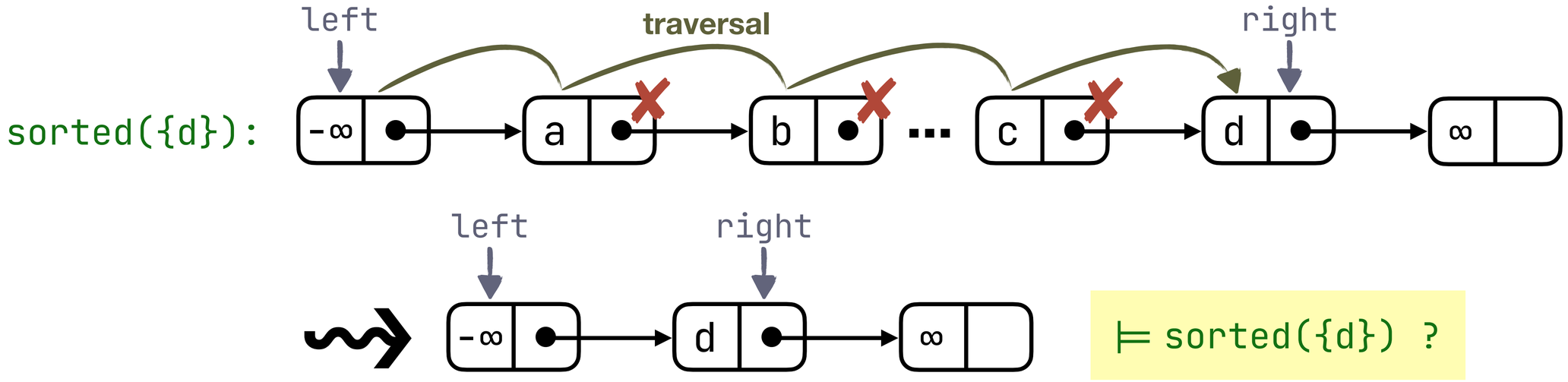
Example: Unbounded Removal



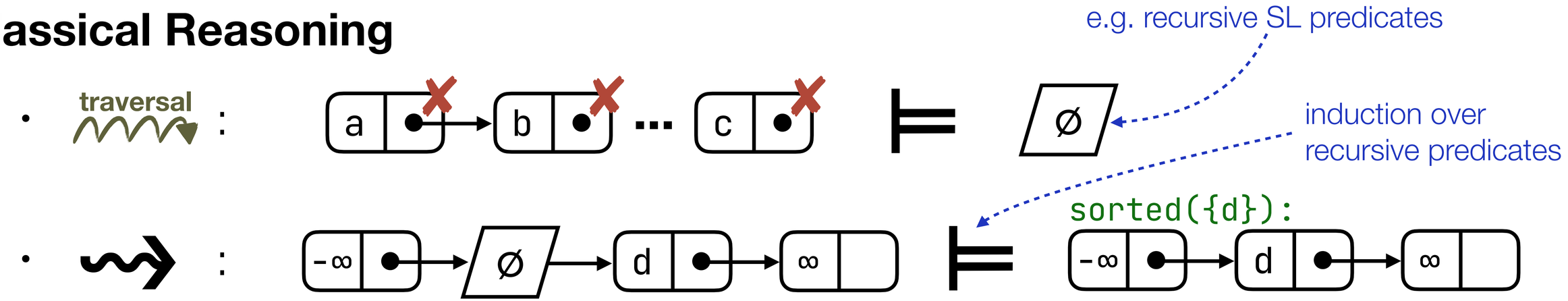
Classical Reasoning

- traversal : 

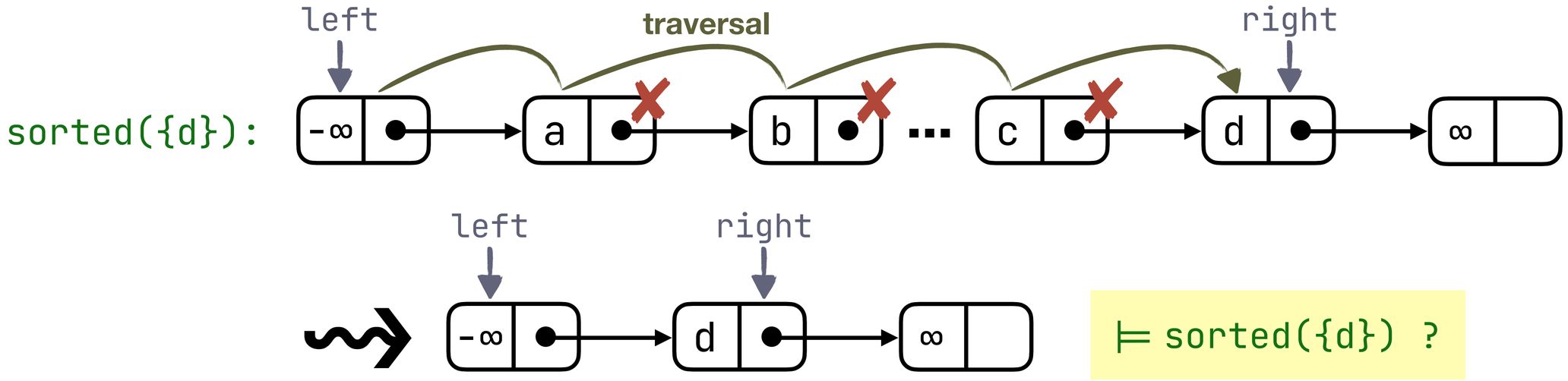
Example: Unbounded Removal



Classical Reasoning

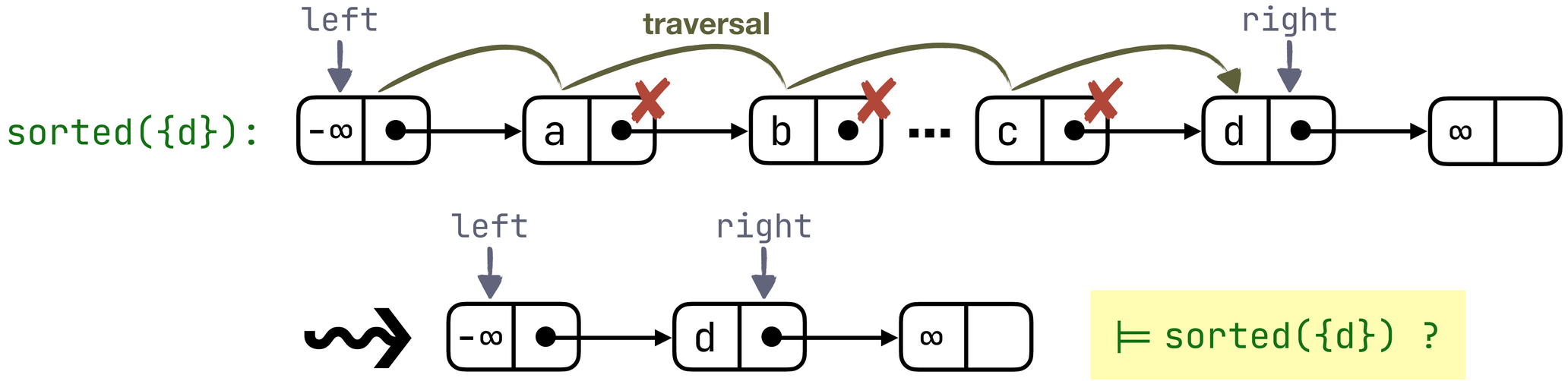


Example: Unbounded Removal



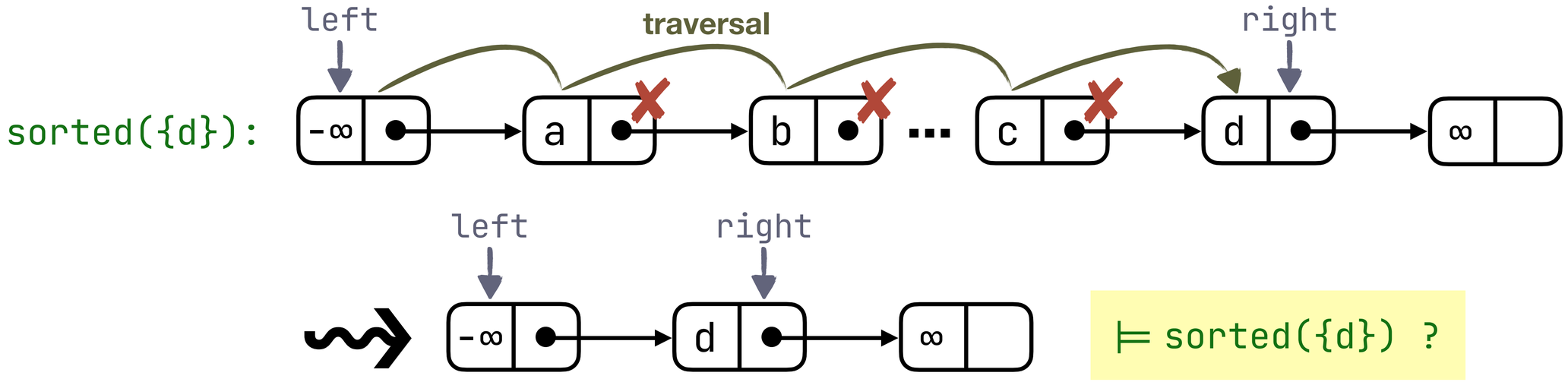
Contribution: Ghost Update Chunks

Example: Unbounded Removal

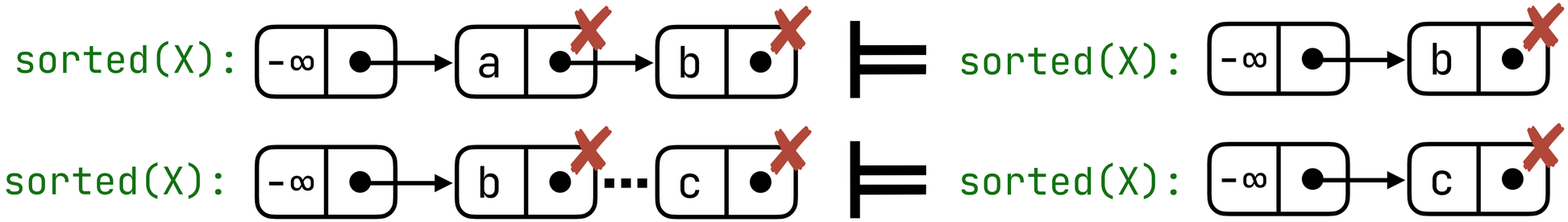


Contribution: Ghost Update Chunks

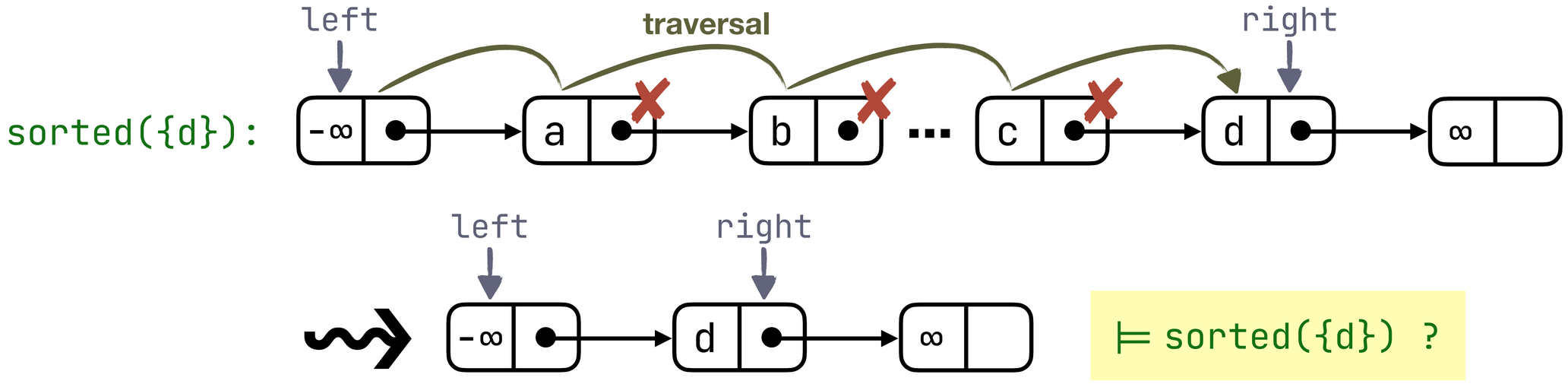
Example: Unbounded Removal



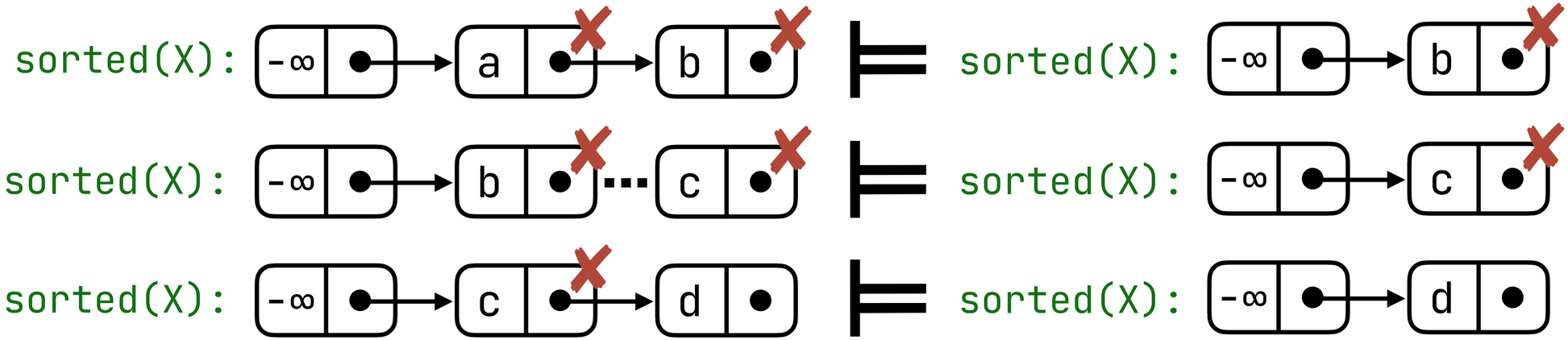
Contribution: Ghost Update Chunks



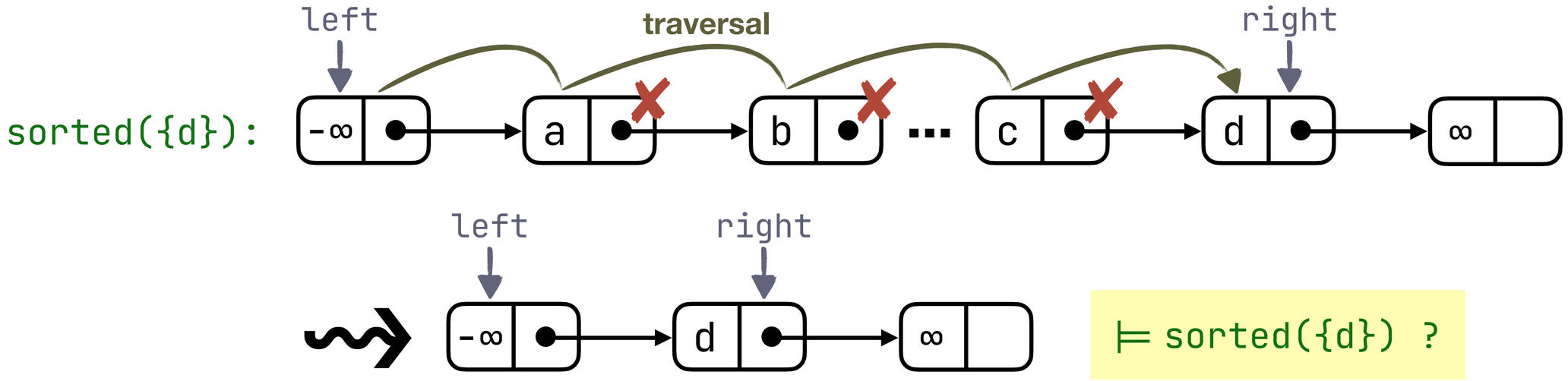
Example: Unbounded Removal



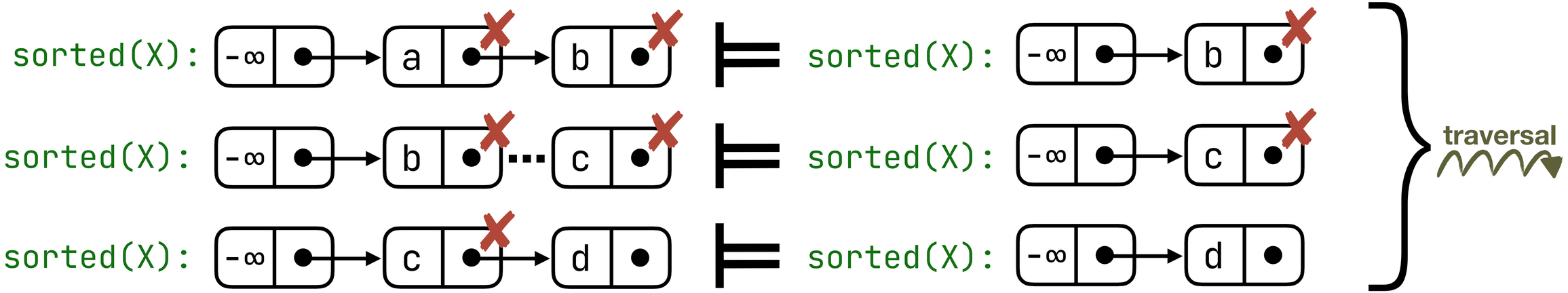
Contribution: Ghost Update Chunks



Example: Unbounded Removal

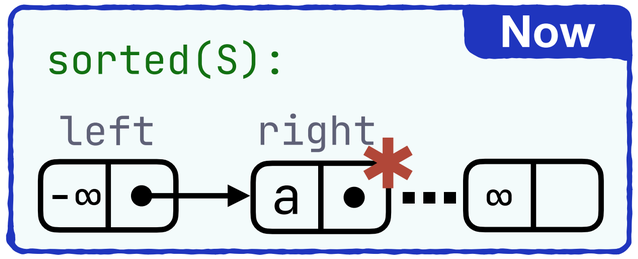


Contribution: Ghost Update Chunks



Ghost Update Chunks

```
// traversal step 1
```

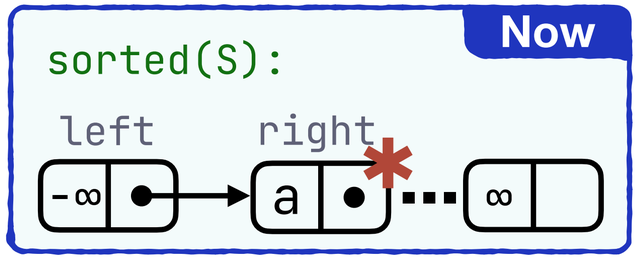



```
assume(right→mark);
next := right→next;
```

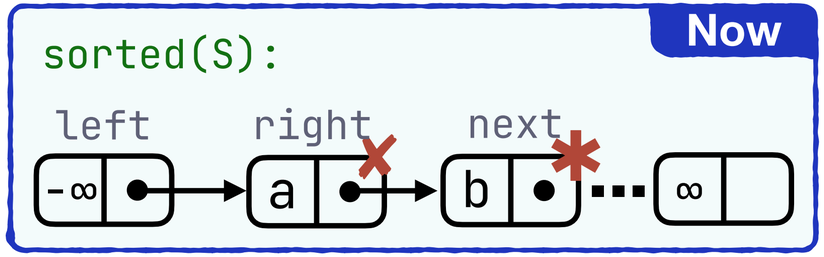
```
right := next;
```

Ghost Update Chunks

```
// traversal step 1
```



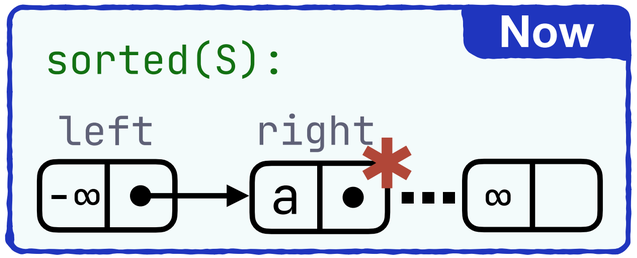

```
assume(right→mark);
next := right→next;
```



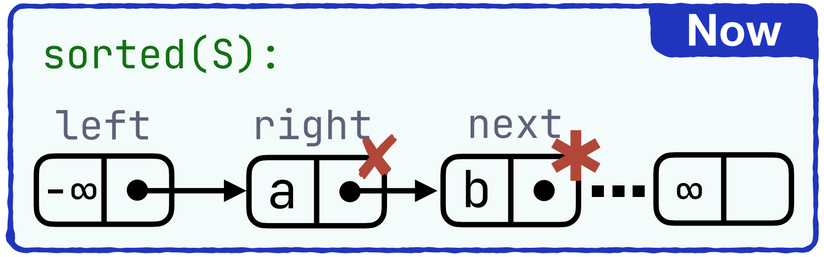

```
right := next;
```

Ghost Update Chunks

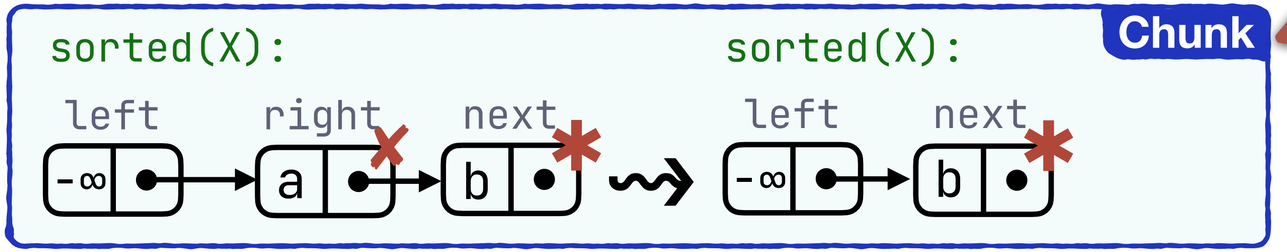
```
// traversal step 1
```




```
assume(right→mark);
next := right→next;
```



*

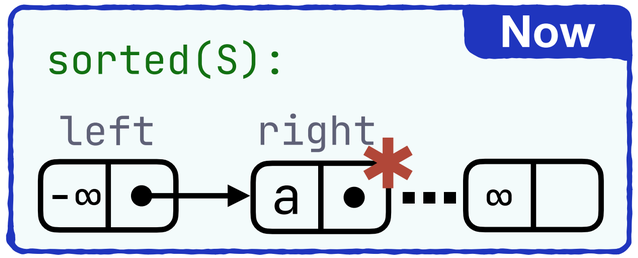


INTRO

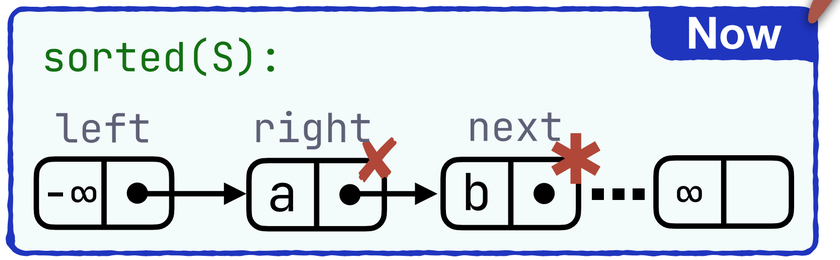
```
right := next;
```

Ghost Update Chunks

```
// traversal step 1
```

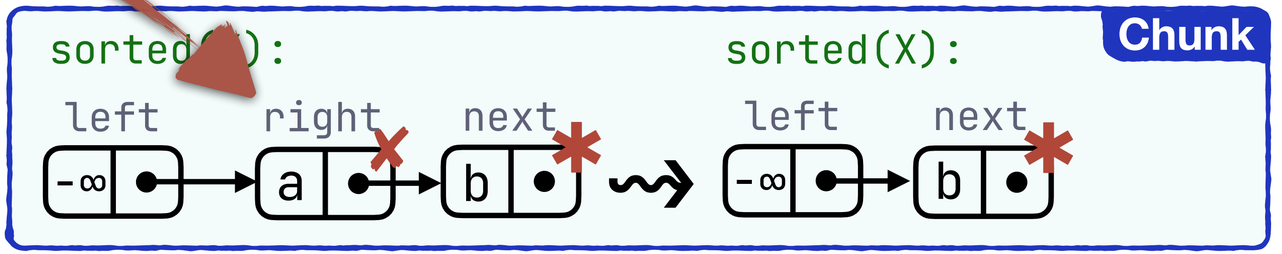



```
assume(right→mark);
next := right→next;
```



DISCHARGE

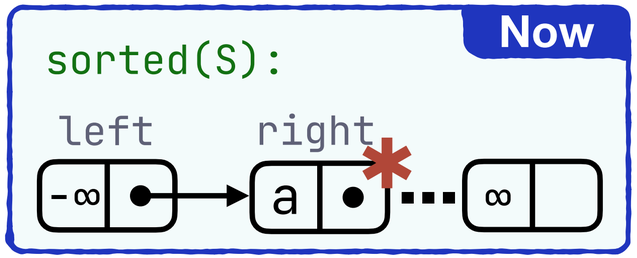
*



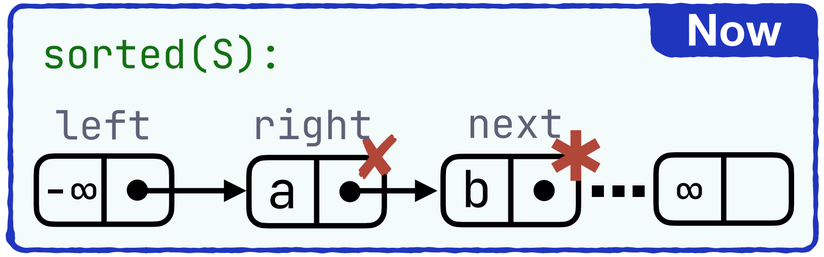

```
right := next;
```

Ghost Update Chunks

```
// traversal step 1
```

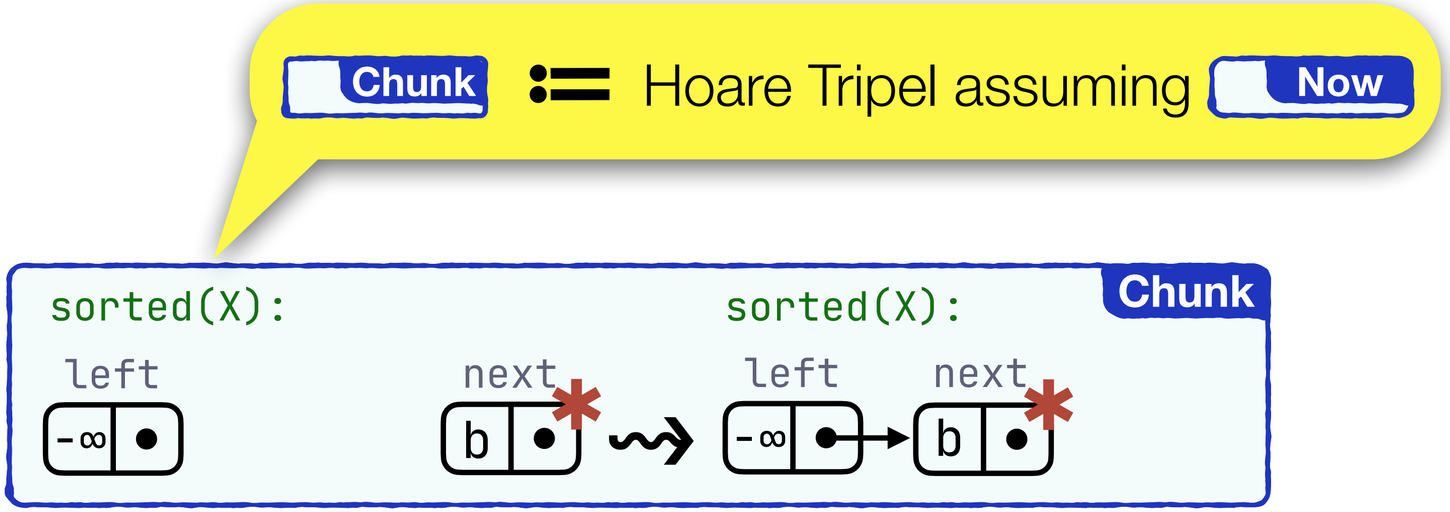



```
assume(right→mark);
next := right→next;
```



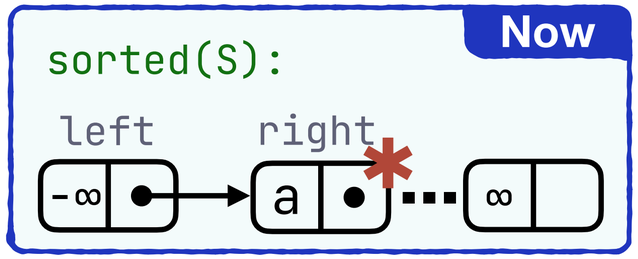

```
right := next;
```

*

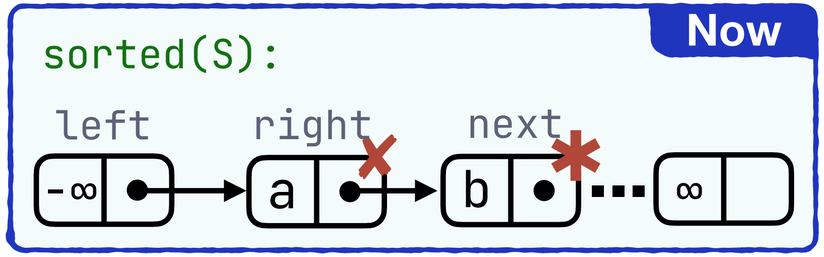


Ghost Update Chunks

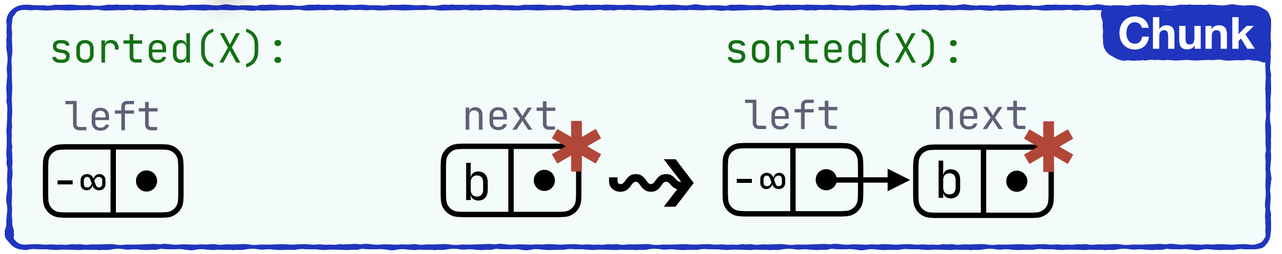
```
// traversal step 1
```




```
assume(right→mark);
next := right→next;
```

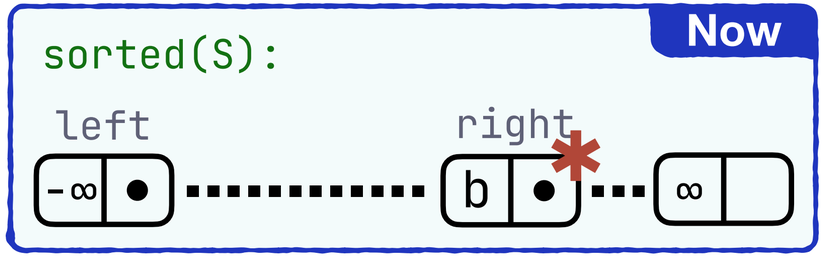


*

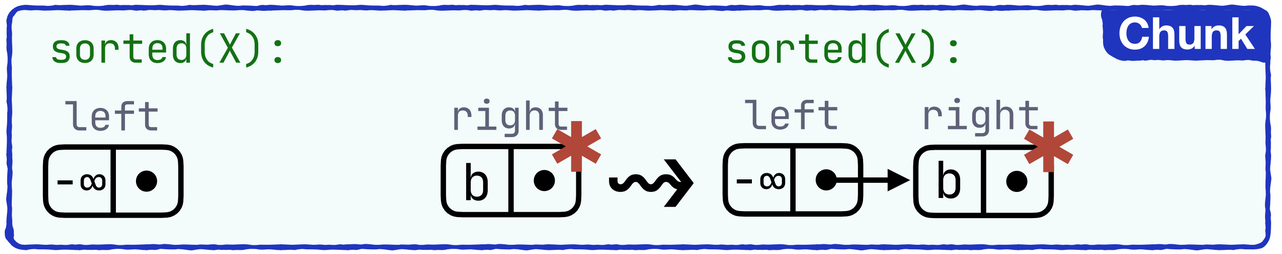


Chunk ≔ Hoare Tripel assuming Now

```
right := next;
```

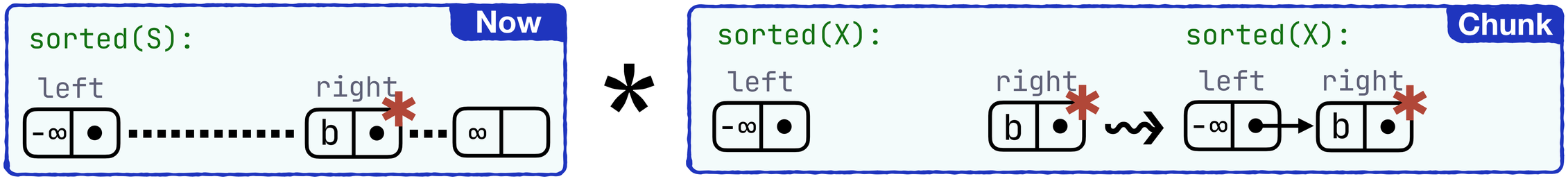


*



Ghost Update Chunks

```
// traversal step 1 2
```

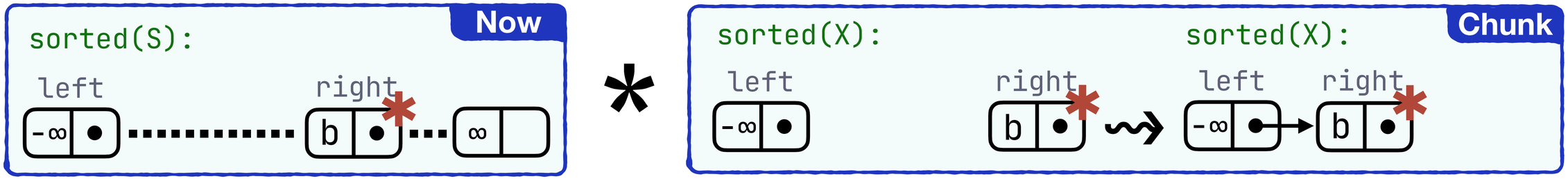



```
assume(right→mark);
next := right→next;
```

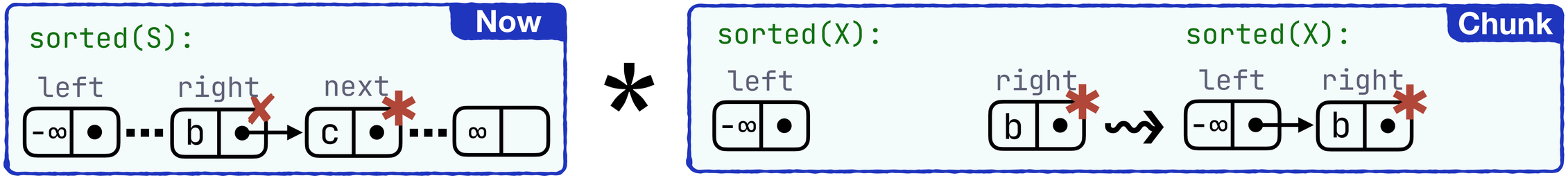
```
right := next;
```

Ghost Update Chunks

```
// traversal step 1 2
```



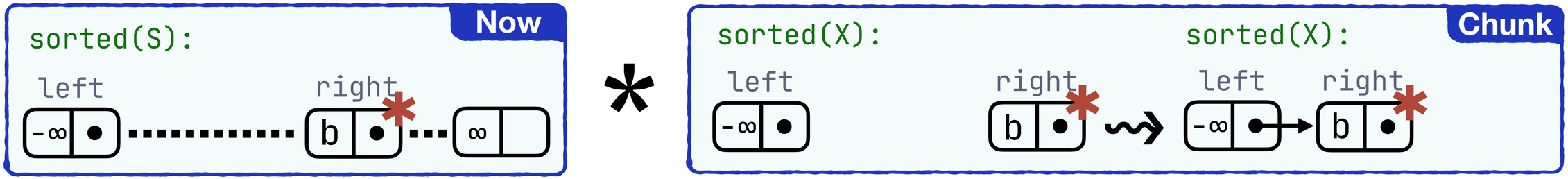

```
assume(right→mark);
next := right→next;
```



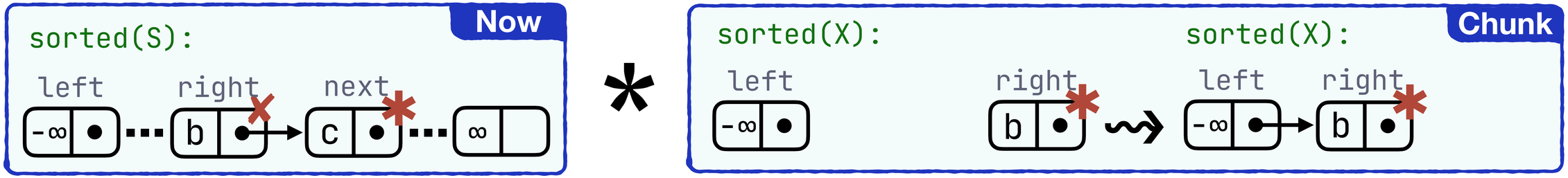

```
right := next;
```

Ghost Update Chunks

```
// traversal step 1 2
```

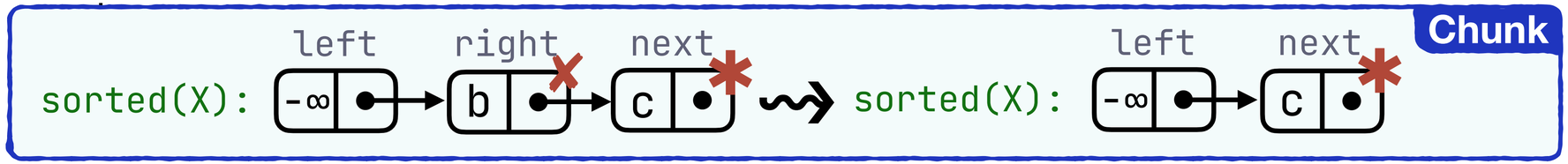


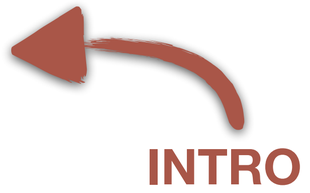

```
assume(right→mark);
next := right→next;
```




```
right :=
```

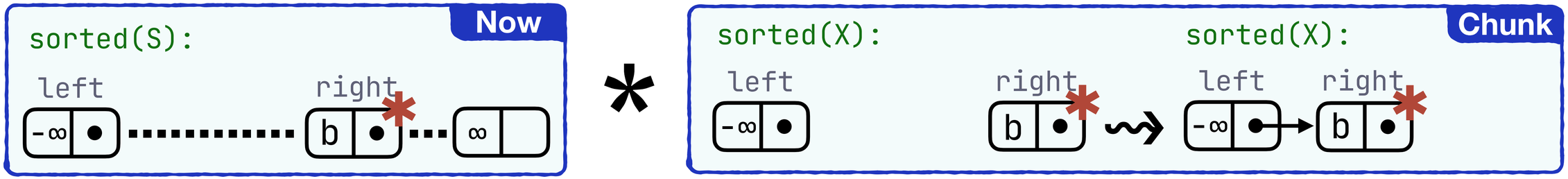
*



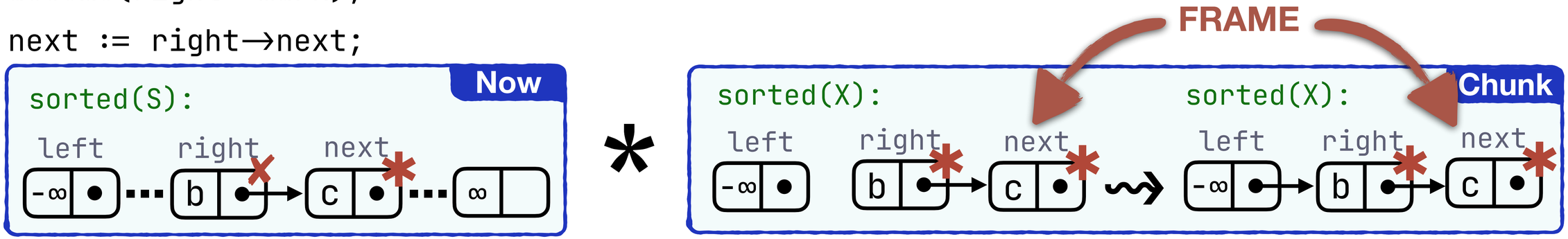


Ghost Update Chunks

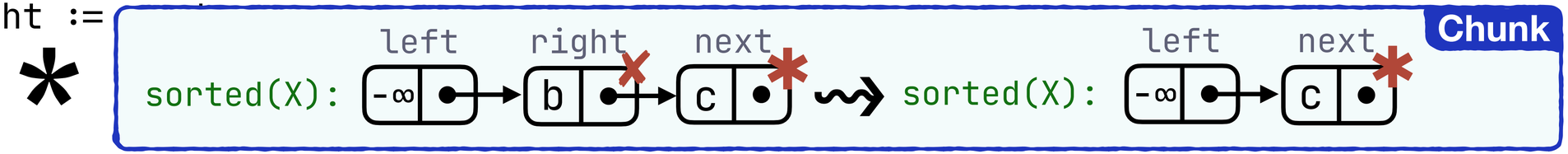
```
// traversal step 1 2
```




```
assume(right→mark);
next := right→next;
```

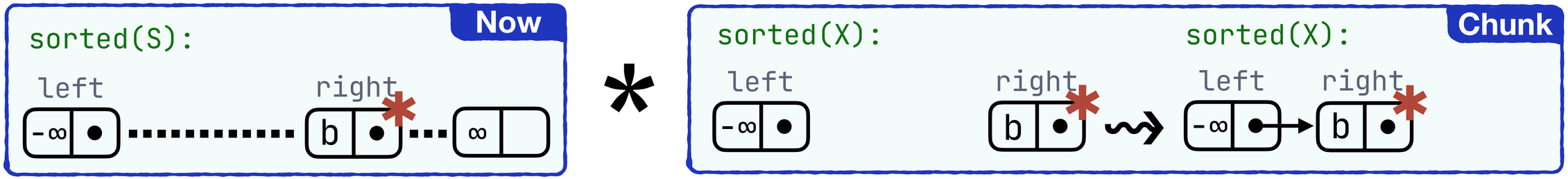



```
right :=
```

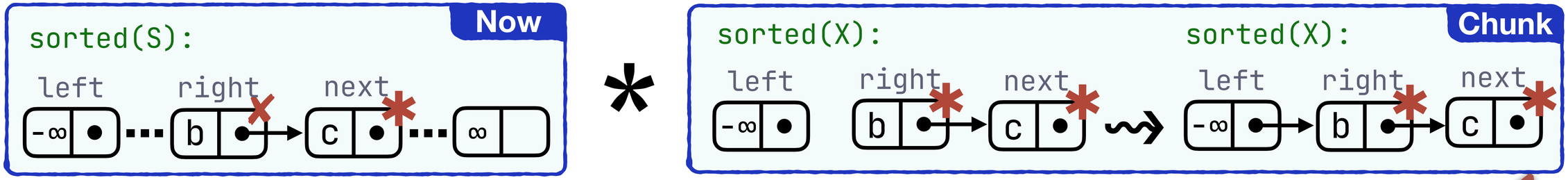


Ghost Update Chunks

```
// traversal step 1 2
```

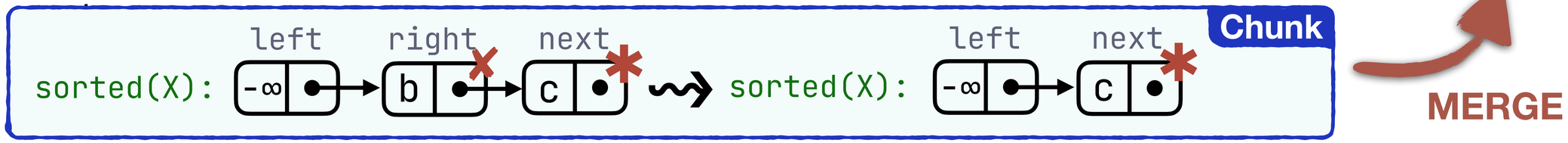



```
assume(right→mark);
next := right→next;
```



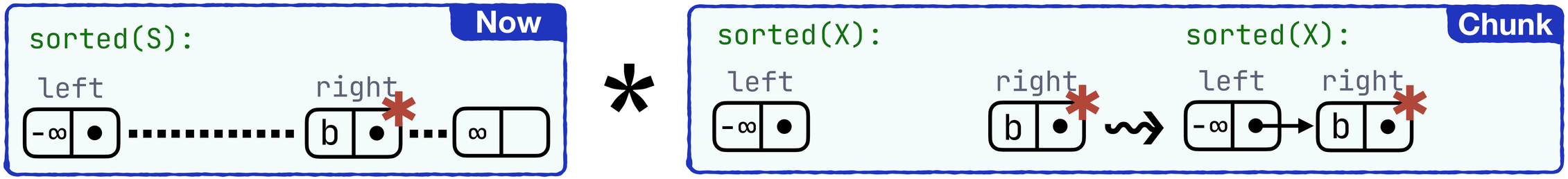

```
right :=
```

*

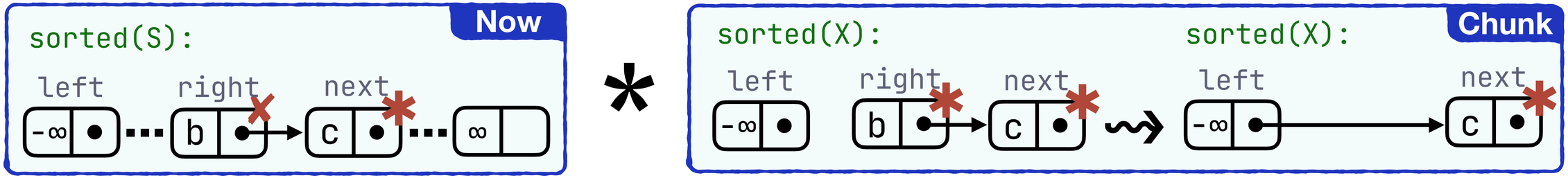


Ghost Update Chunks

```
// traversal step 1 2
```



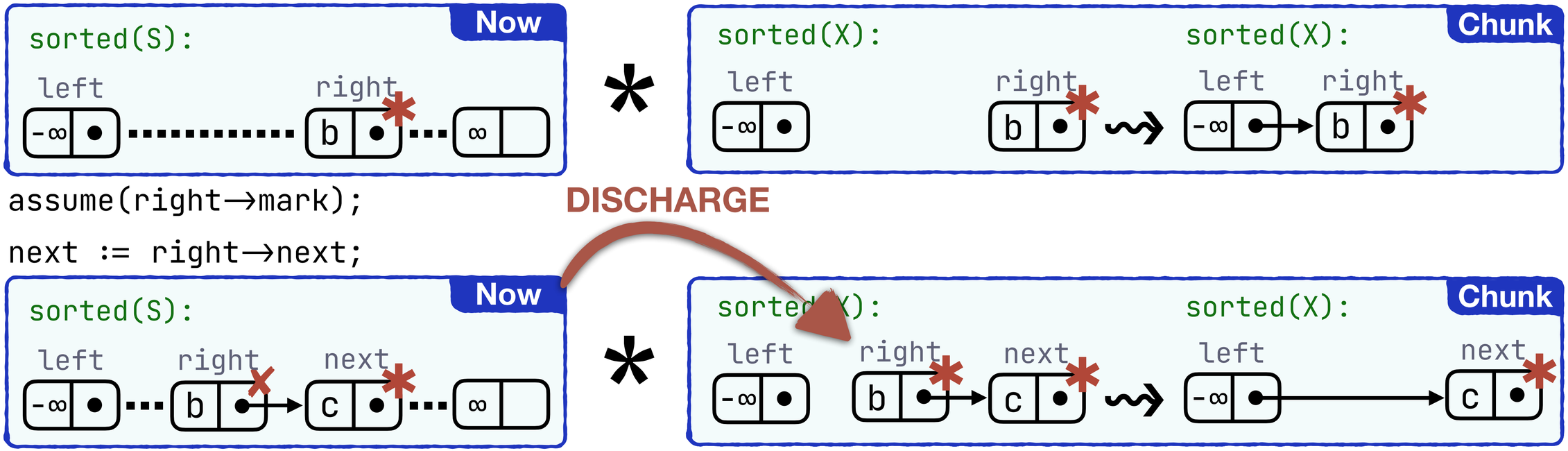

```
assume(right→mark);
next := right→next;
```




```
right := next;
```

Ghost Update Chunks

```
// traversal step 1 2
```




```
assume(right→mark);
```

DISCHARGE

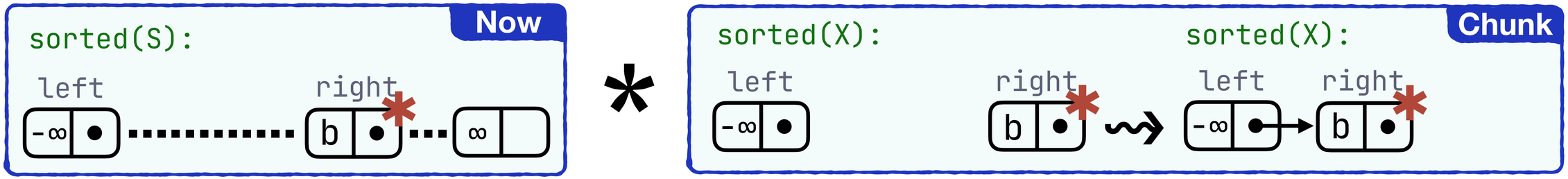
```
next := right→next;
```



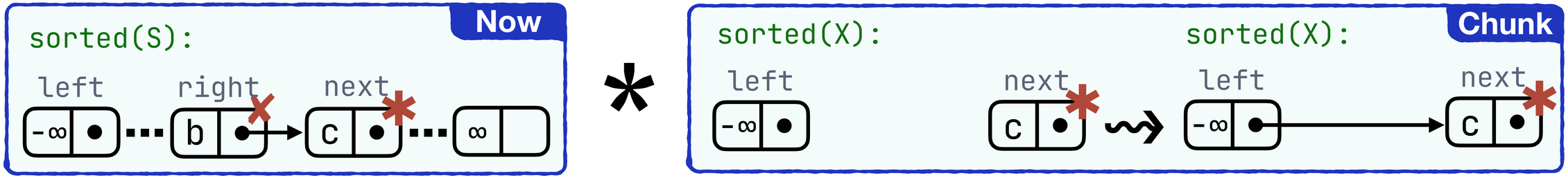
```
right := next;
```

Ghost Update Chunks

```
// traversal step 1 2
```



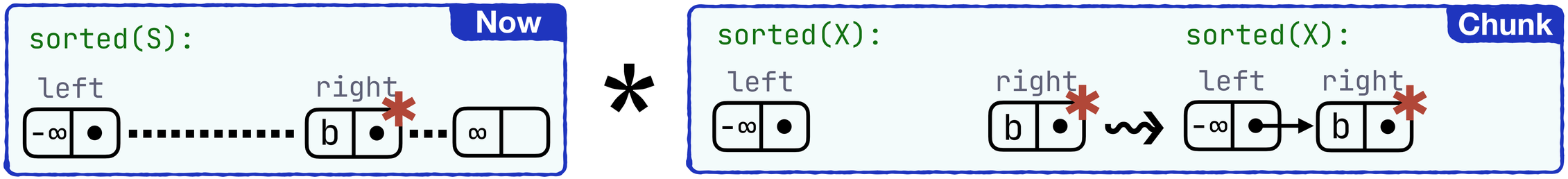

```
assume(right→mark);
next := right→next;
```



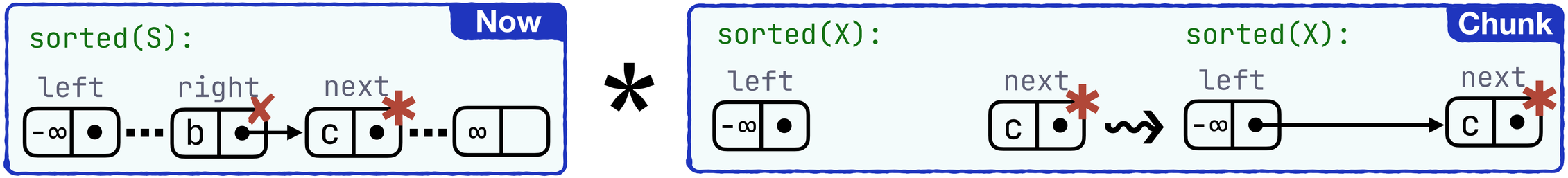

```
right := next;
```

Ghost Update Chunks

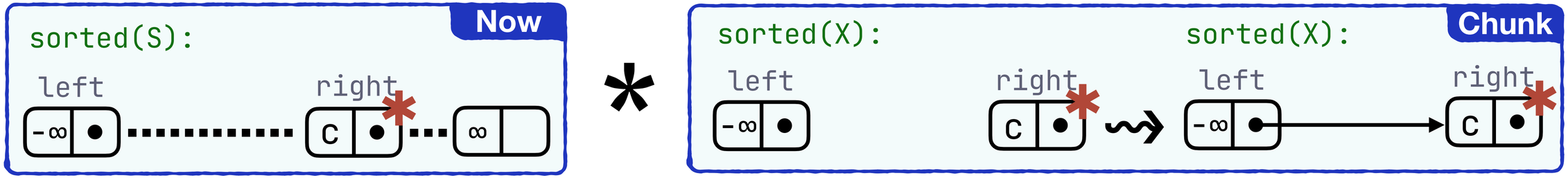
```
// traversal step 1 2
```




```
assume(right→mark);
next := right→next;
```

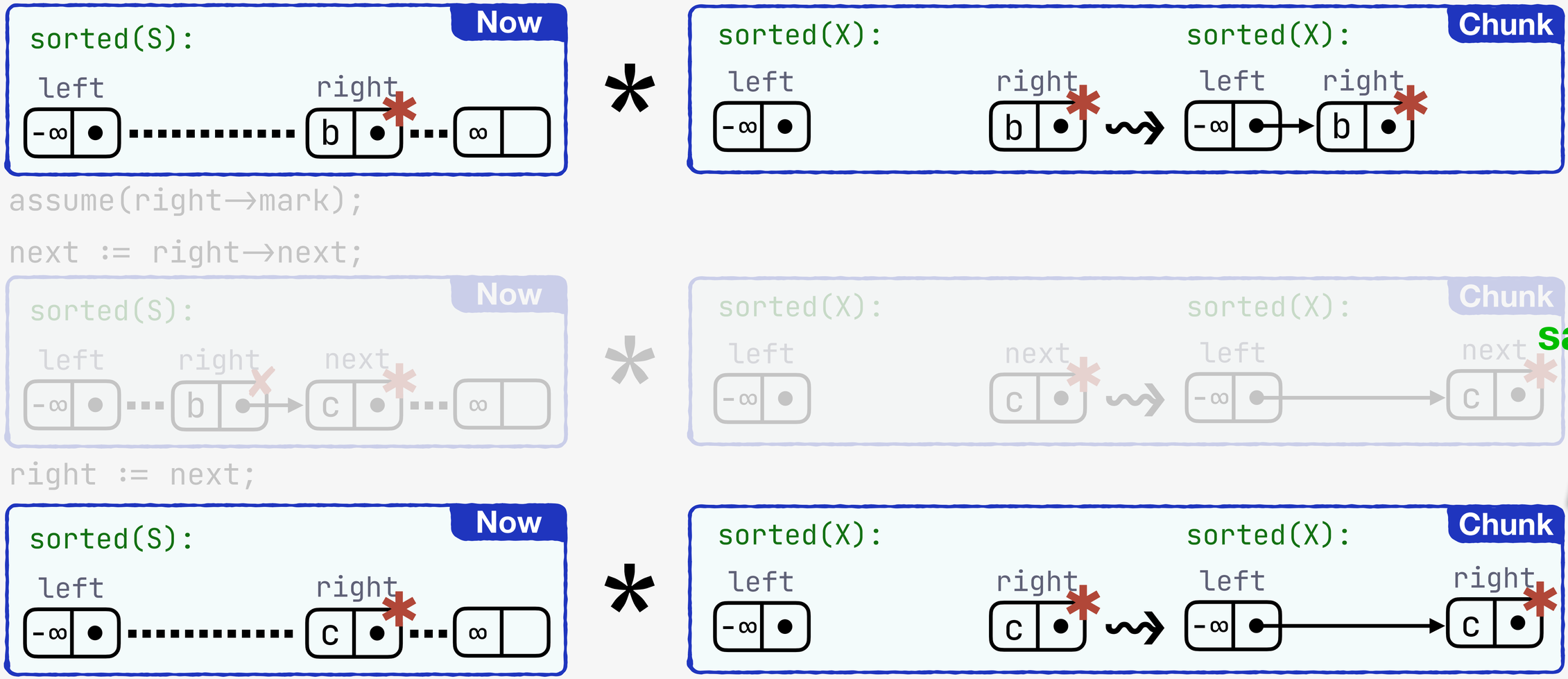



```
right := next;
```

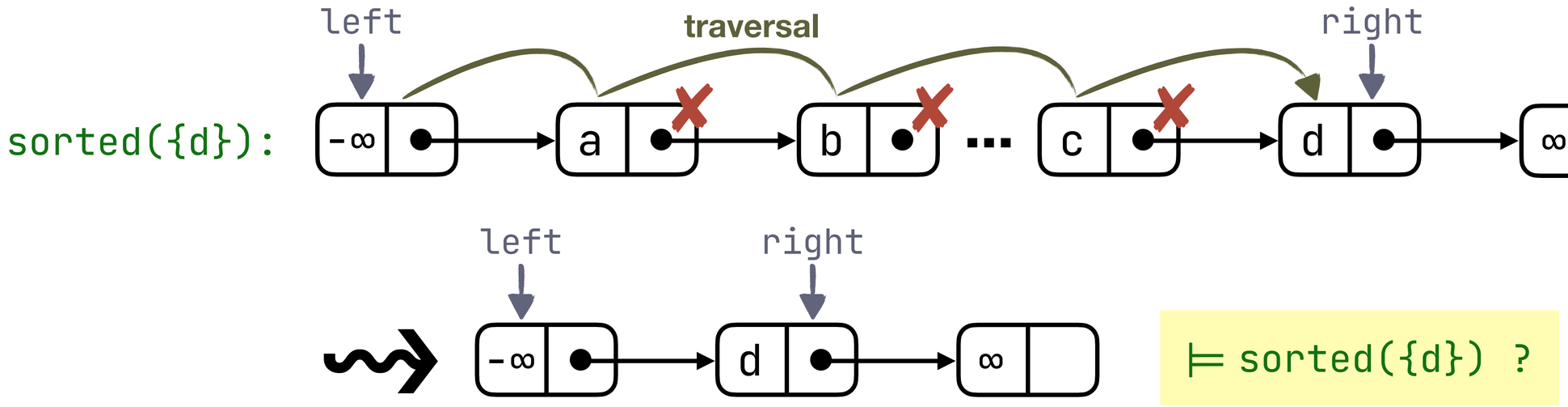


Ghost Update Chunks

```
// traversal step 1 2 k+1
```



Example: Unbounded Removal



Contribution: Ghost Update Chunks

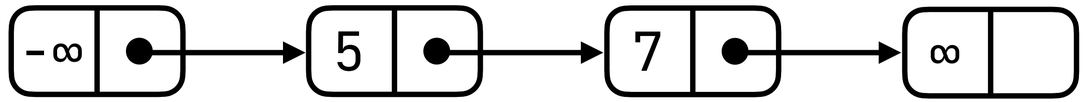
- formalization in abstract SL
- usage patterns: INTRO, MERGE, FRAME, DISCHARGE
- implementation

Verification Challenge 1: Complex Unbounded Updates

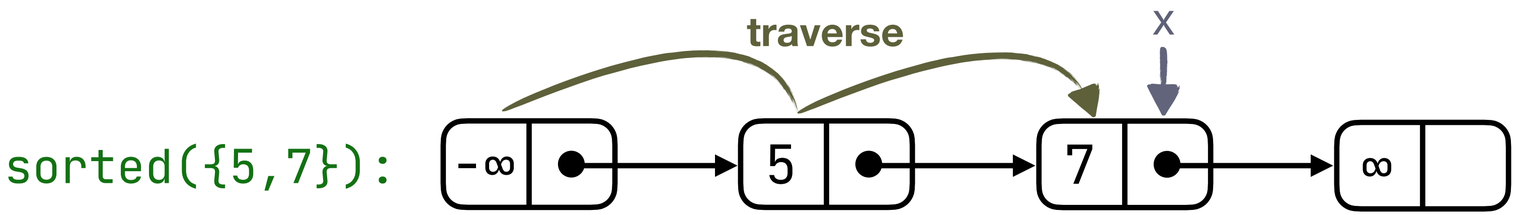
Verification Challenge 2: **Linearizability**

Example: `contains(7)`

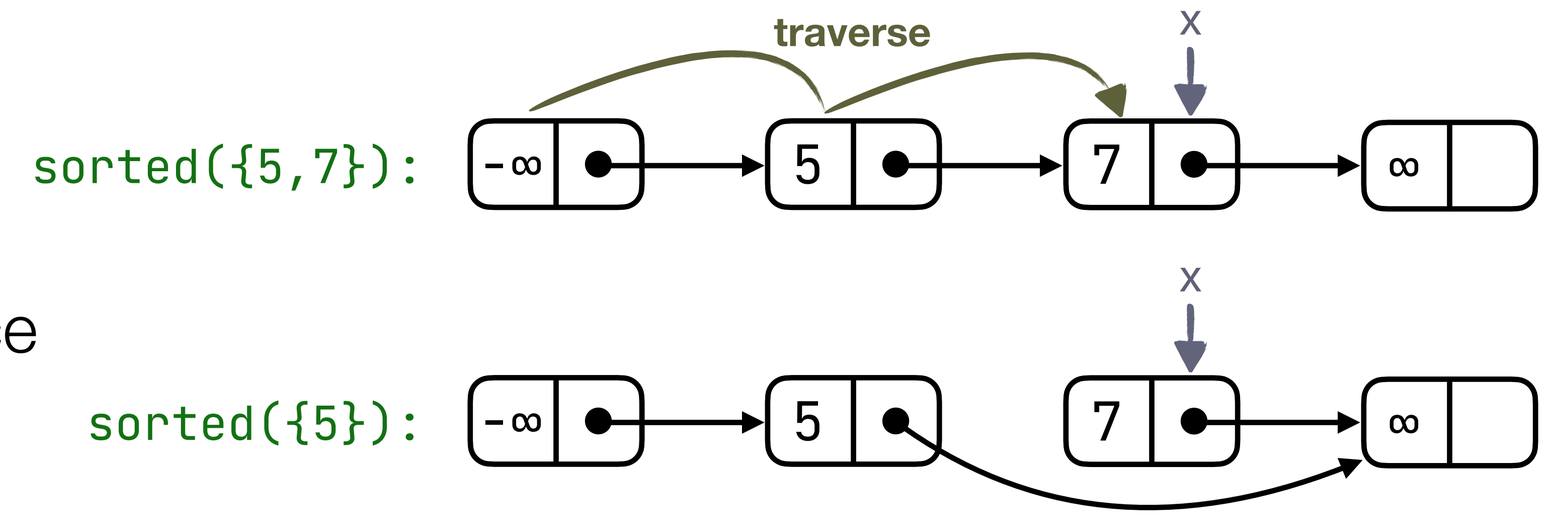
`sorted({5,7}):`



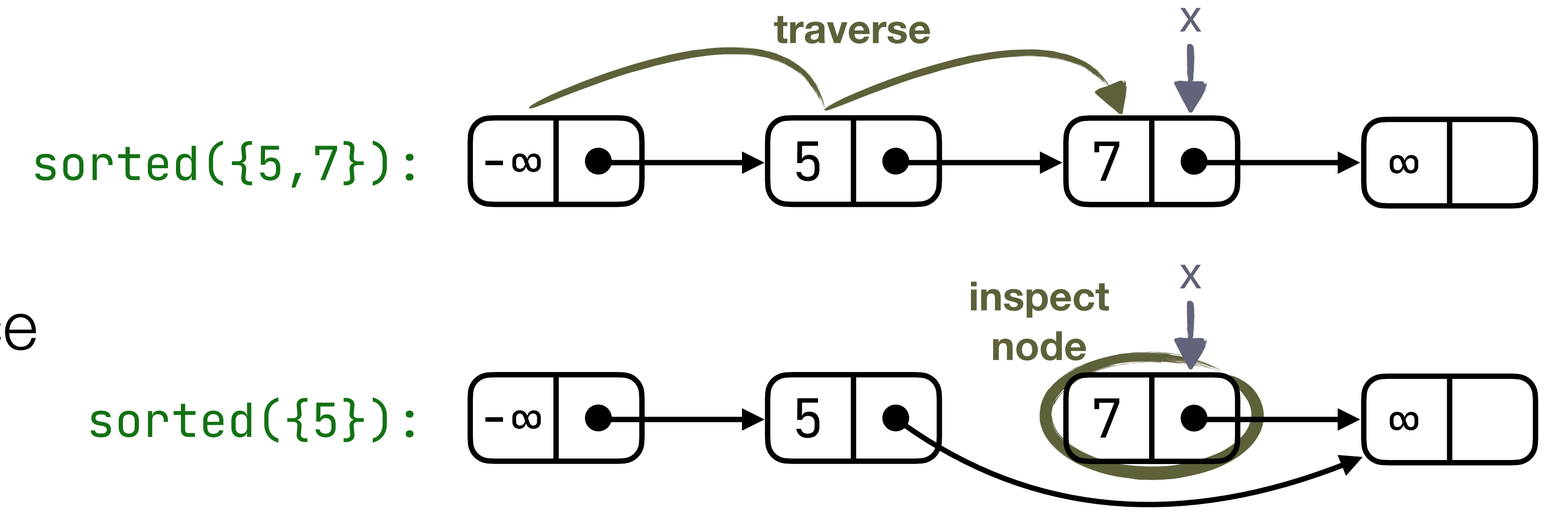
Example: `contains(7)`



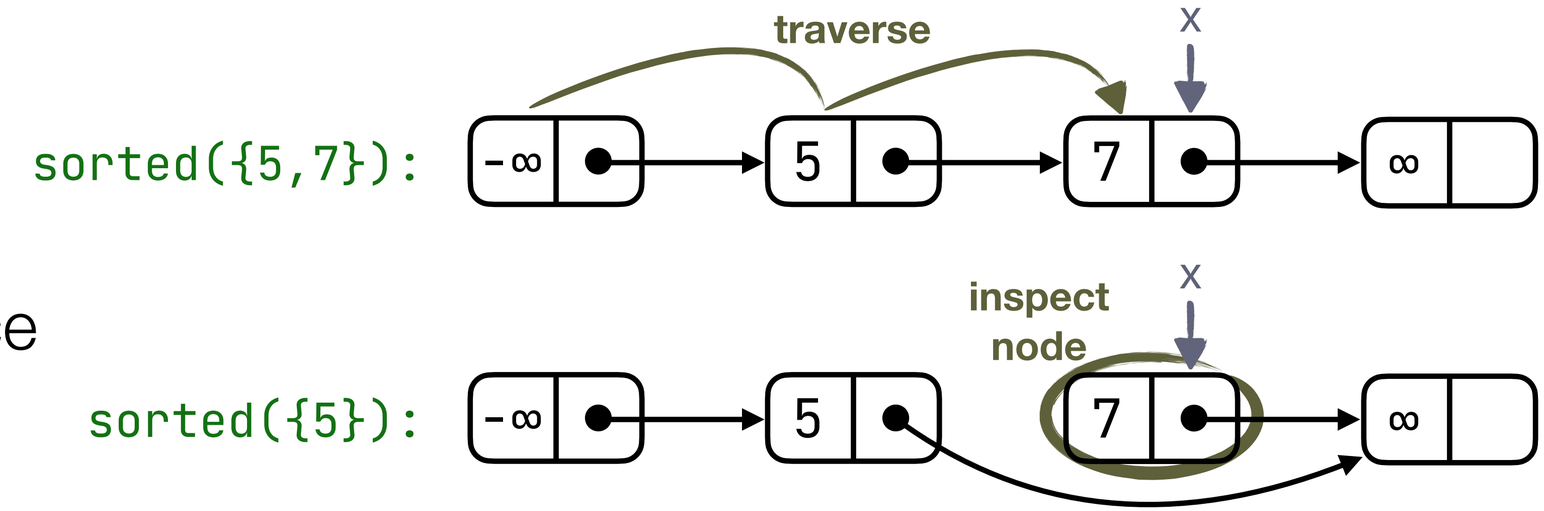
Example: `contains(7)`



Example: `contains(7)`

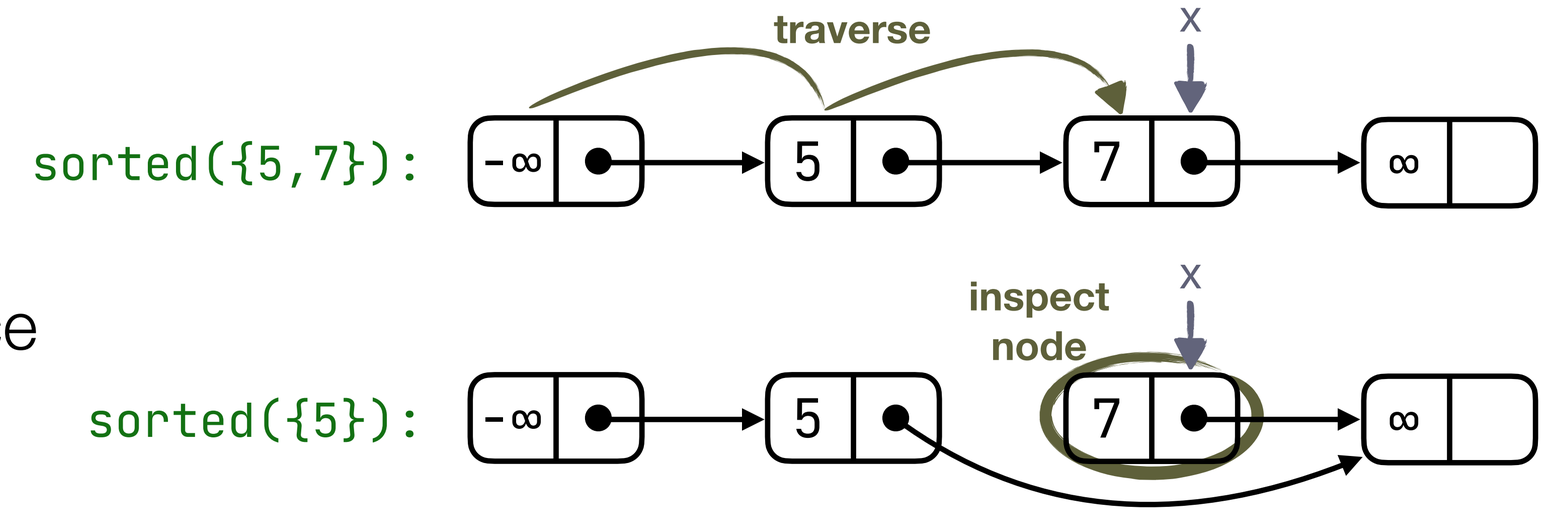


Example: `contains(7)`



$\models$ `sorted({7,...})` at some point ?

Example: `contains(7)`



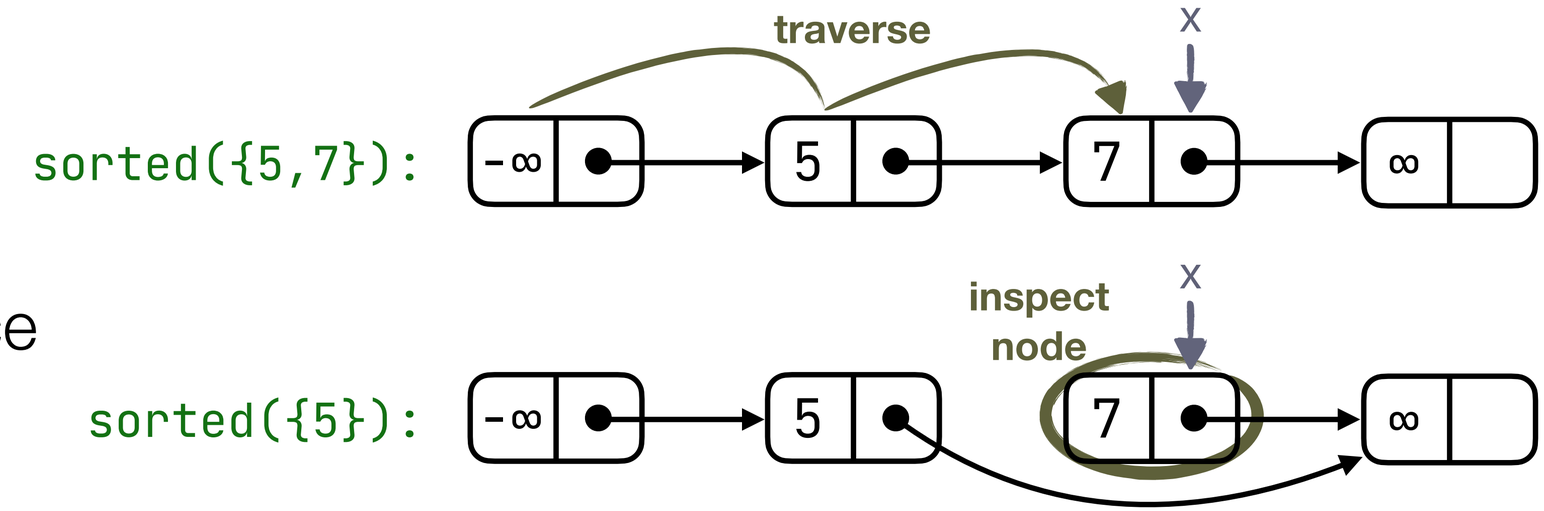
$\models$ `sorted({7,...})` `at some point ?`

Classical Reasoning

- traverse : prophecy about outcome of inspect node ("case split")

automation ?

Example: `contains(7)`



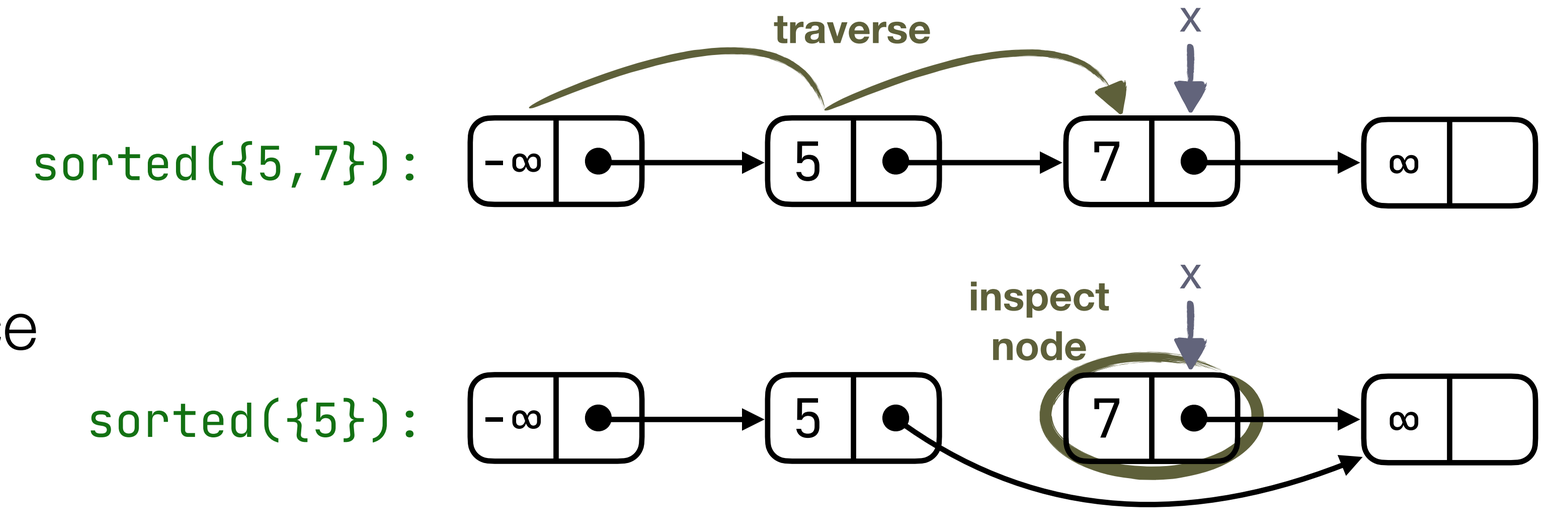
$\models$ `sorted({7,...})` at some point ?

Classical Reasoning

- traverse : prophecy about outcome of inspect node ("case split")
- inspect node : reconcile outcome with prophecy

automation ?

Example: `contains(7)`

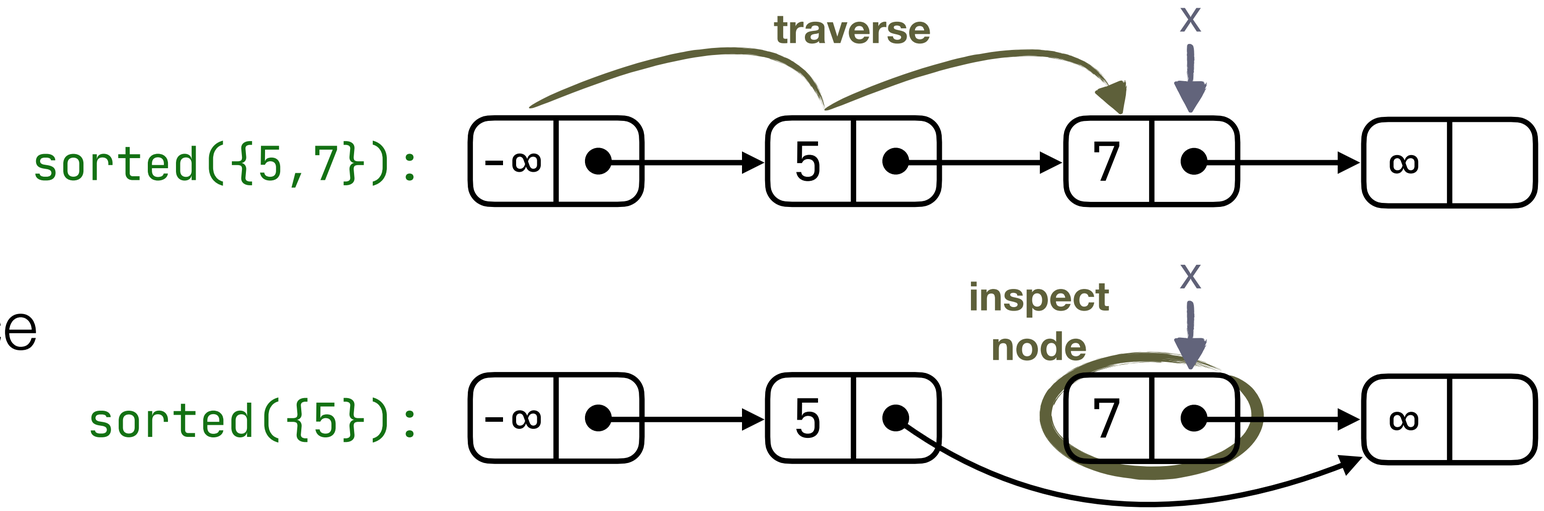


$\models$ `sorted({7,...})` at some point ?

Contribution: Past Reasoning

- traverse : record past state

Example: `contains(7)`



interference

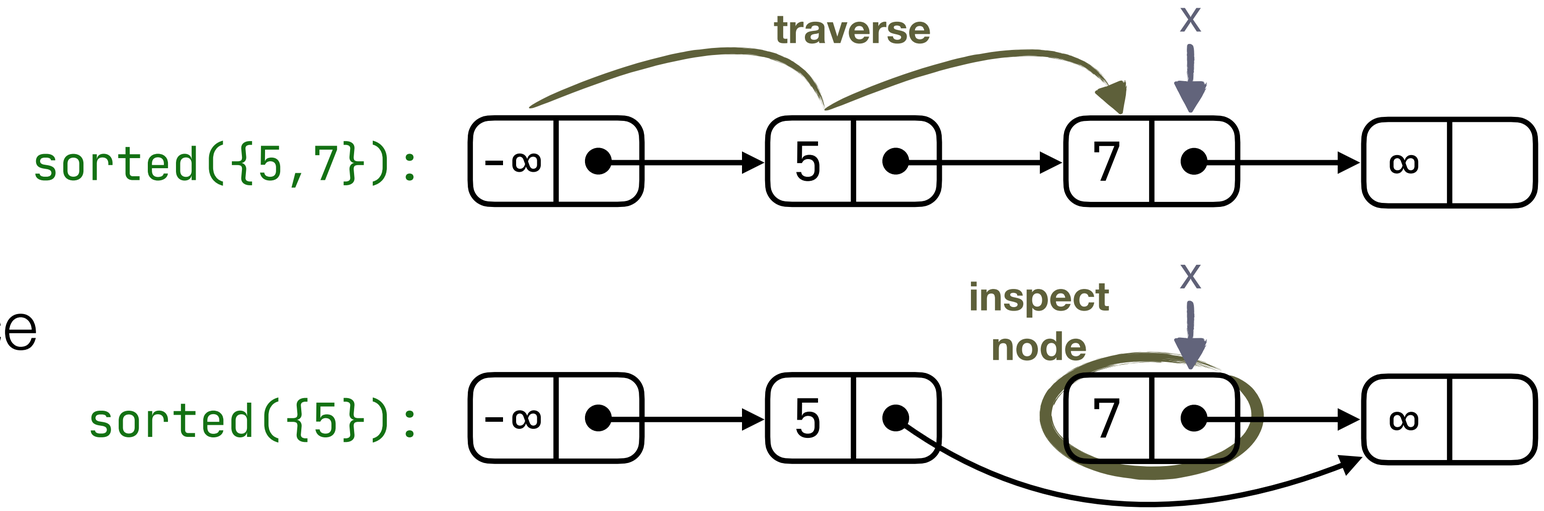
$\rightsquigarrow$

$\models$ `sorted({7,...})` at some point ?

Contribution: Past Reasoning

- traverse : record past state
- inspect node : propagate outcome into past state

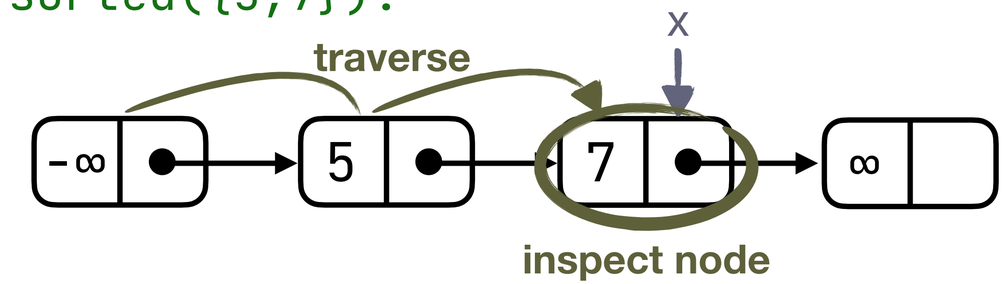
Example: `contains(7)`



⊨ `sorted({7,...})` at some point ?

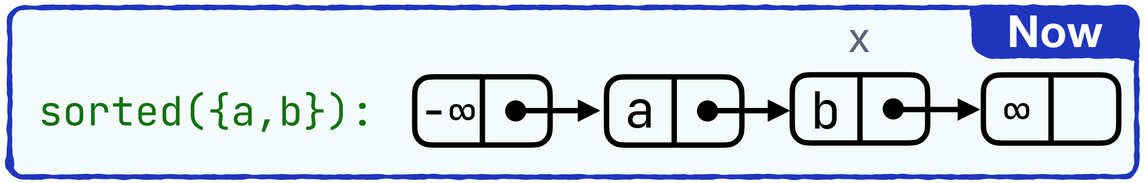
Contribution: Past Reasoning

- traverse : record past state
- inspect node : propagate outcome into past state



Past Reasoning

```
//  ...
x = /* a→next */
```



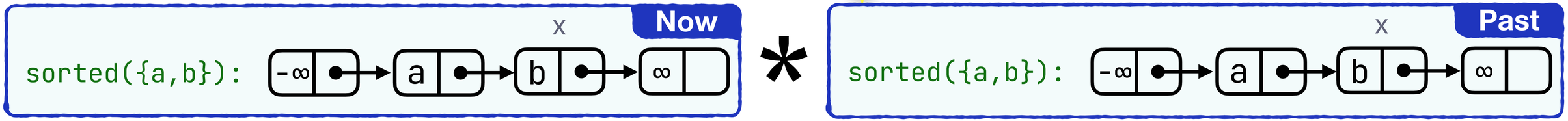

```
if (x→key == k) {
```

```
   return true;
} else { /* continue traversal */ }
```

Past Reasoning

Past ≔ Now in some (previous) state of the execution

```
// ...
x = /* a→next */
```

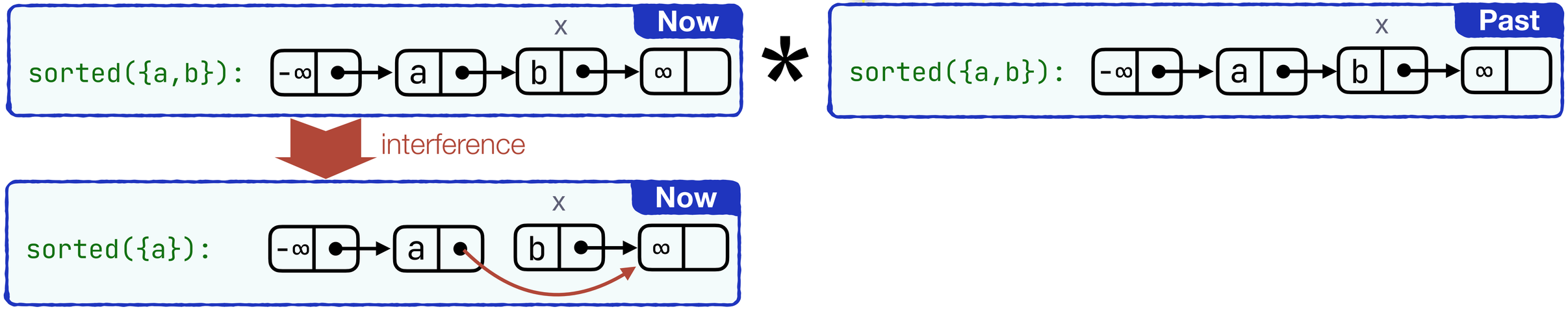



```
if (x→key == k) {
   return true;
} else { /* continue traversal */ }
```

Past Reasoning

Past ≔ Now in some (previous) state of the execution

```
// ...
x = /* a→next */
```

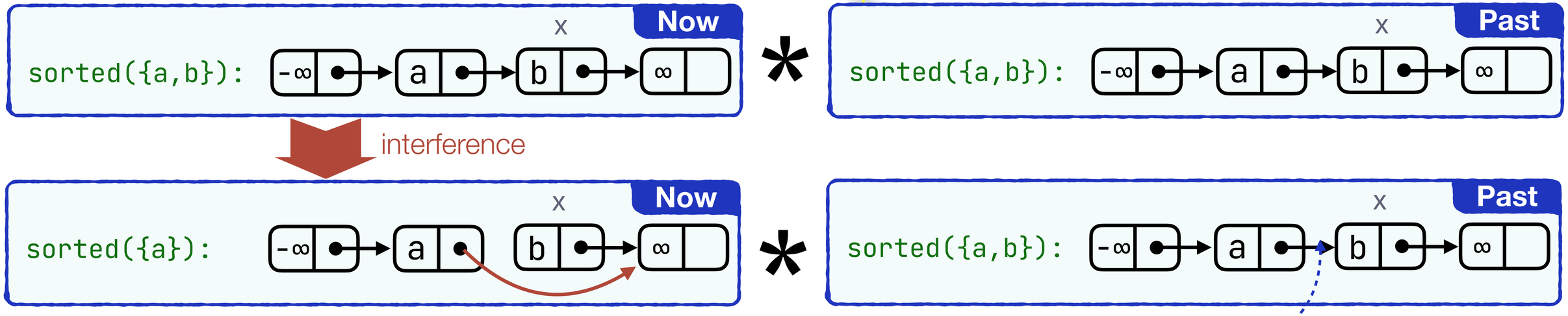



```
if (x→key == k) {
   return true;
} else { /* continue traversal */ }
```

Past Reasoning

Past ≔ Now in some (previous) state of the execution

```
// ...
x = /* a→next */
```

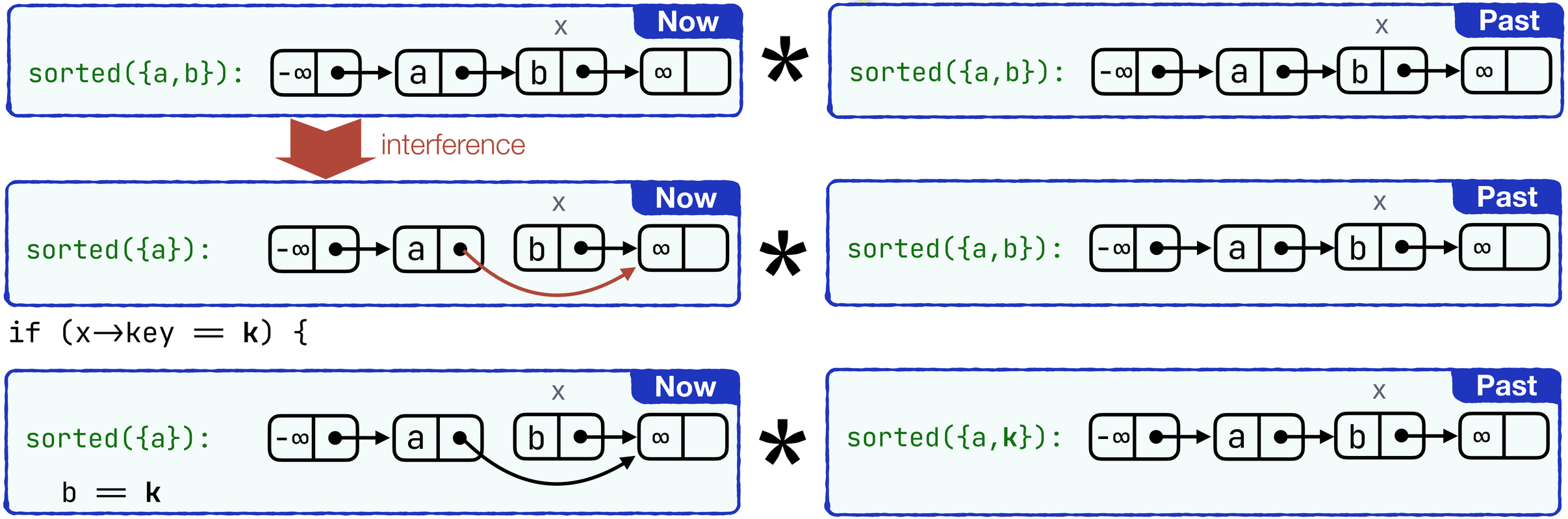



```
if (x→key == k) {
   return true;
} else { /* continue traversal */ }
```

Past Reasoning

Past ≔ Now in some (previous) state of the execution

```
// ...
x = /* a→next */
```




```
if (x→key == k) {
```

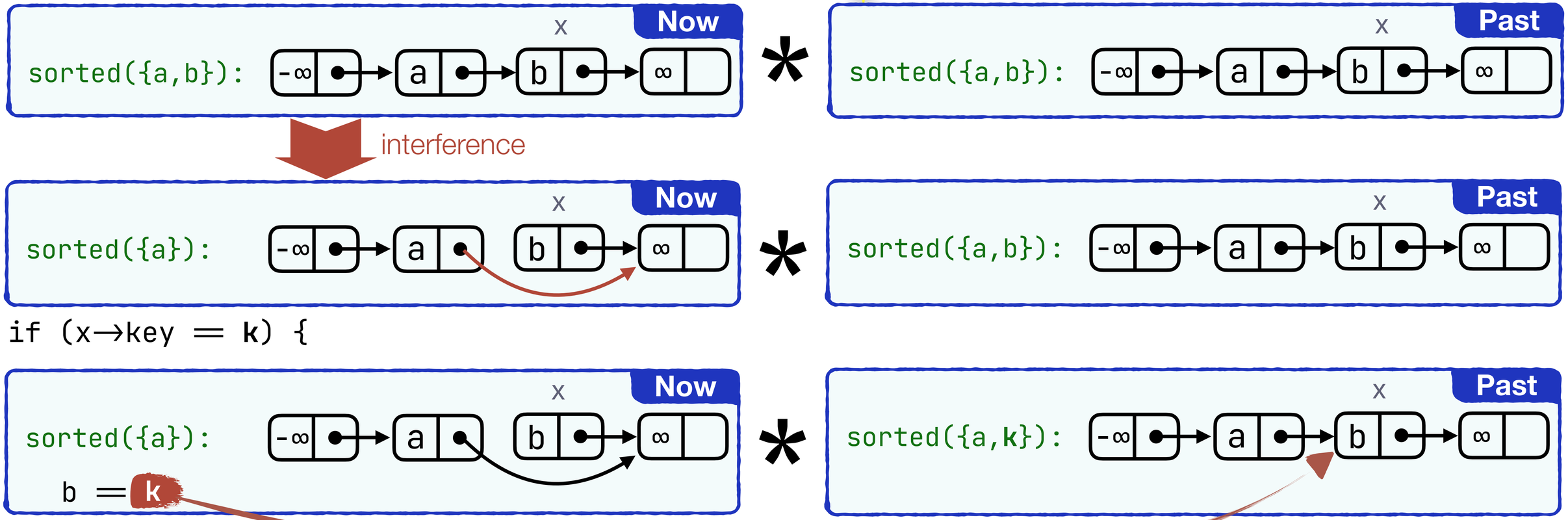


```
   return true;
} else { /* continue traversal */ }
```

Past Reasoning

Past ≔ Now in some (previous) state of the execution

```
// ...
x = /* a→next */
```

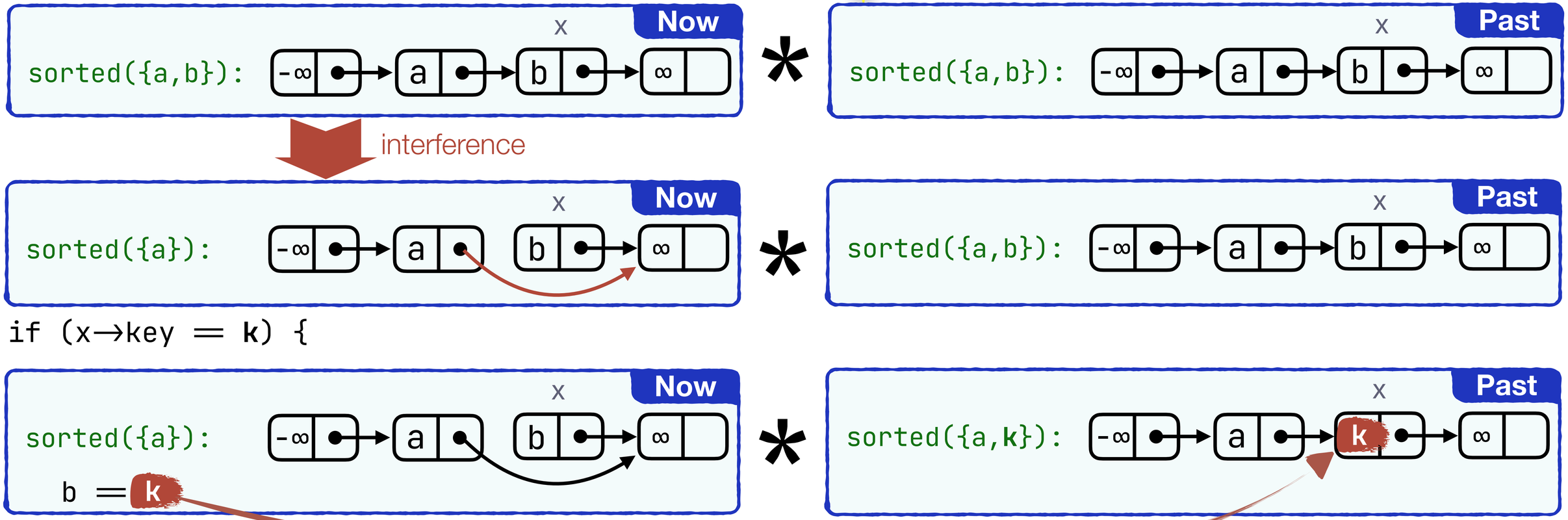



```
if (x→key == k) {
   return true;
} else { /* continue traversal */ }
```

Past Reasoning

Past ≔ Now in some (previous) state of the execution

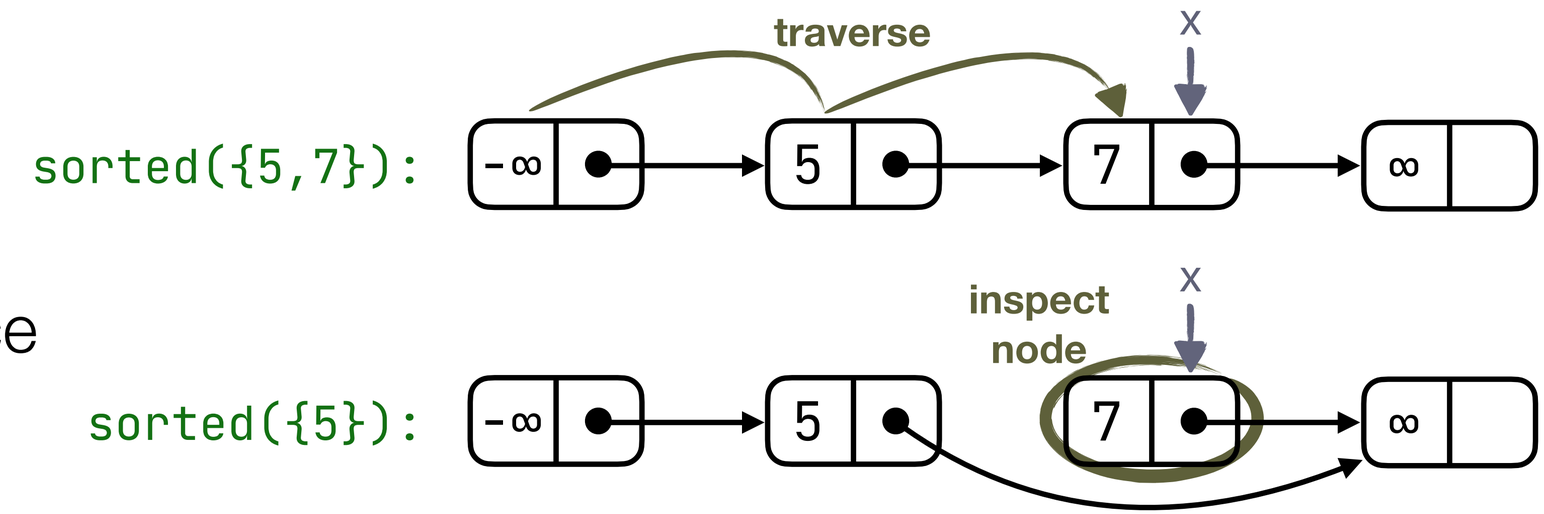
```
// ...
x = /* a→next */
```




```
if (x→key == k) {
   return true;
} else { /* continue traversal */ }
```

⊨ sorted({k,...}) at some point !

Example: `contains(7)`



interference

$\rightsquigarrow$

$\models$ `sorted({7,...})` at some point ?

Contribution: Past Reasoning

- formalization in abstract SL
- light-weight instance of hindsight reasoning
- implementation

Recap:
A Concurrent Program Logic with a Future and History

- program logic with
  - chunks ⇝ complex unbounded updates via simple ones
  - pasts ⇝ lightweight hindsight reasoning in SL
  - tool ⇝ first to verify challenging examples like Harris' set

Benchmark	Lineariz.
Fine-Grained set	46*s* ✓
Lazy set	77*s* ✓
FEMRS tree (no maintenance)	130*s* ✓
Vechev&Yahav 2CAS set	125*s* ✓
Vechev&Yahav CAS set	54*s* ✓
ORVYY set	47*s* ✓
Michael set	306*s* ✓
Harris set	1378*s* ✓

Recap:
A Concurrent Program Logic with a Future and History

Thanks

- program logic with
  - chunks ⇝ complex unbounded updates via simple ones
  - pasts ⇝ lightweight hindsight reasoning in SL
  - tool ⇝ first to verify challenging examples like Harris' set

Benchmark	Lineariz.
Fine-Grained set	46*s* ✓
Lazy set	77*s* ✓
FEMRS tree (no maintenance)	130*s* ✓
Vechev&Yahav 2CAS set	125*s* ✓
Vechev&Yahav CAS set	54*s* ✓
ORVYY set	47*s* ✓
Michael set	306*s* ✓
Harris set	1378*s* ✓