A Decision Procedure for Skiplists with Unbounded Height and Length

Alejandro Sánchez[1]   César Sánchez[1,2]

[1]IMDEA Software Institute, Spain

[2]Spanish Council for Scientific Research (CSIC), Spain

SVARM'11, Tallinn, 31 March 2012

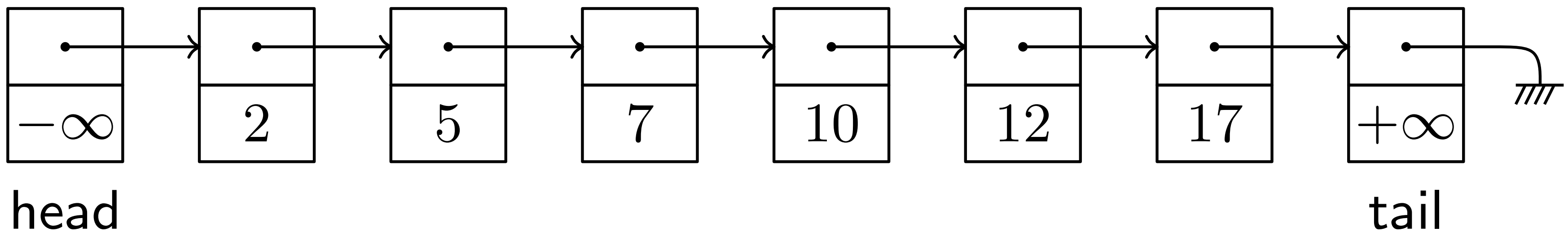
Skiplists

Skiplists

- Sorted list of elements

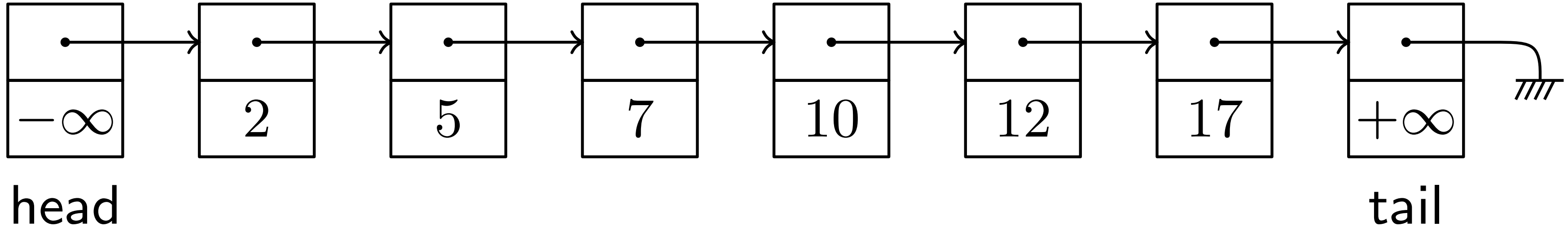
Skiplists

- Sorted list of elements



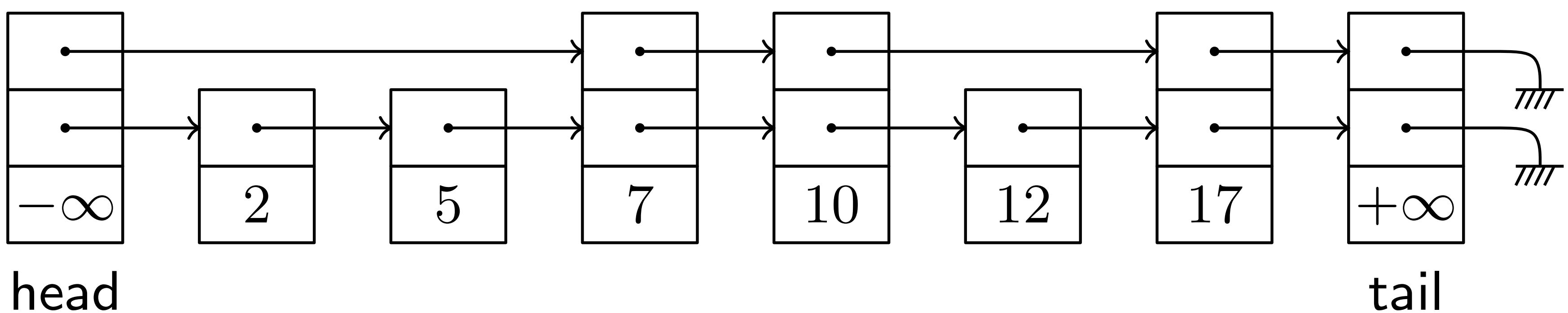
Skiplists

- Sorted list of elements
- Hierarchy of linked lists



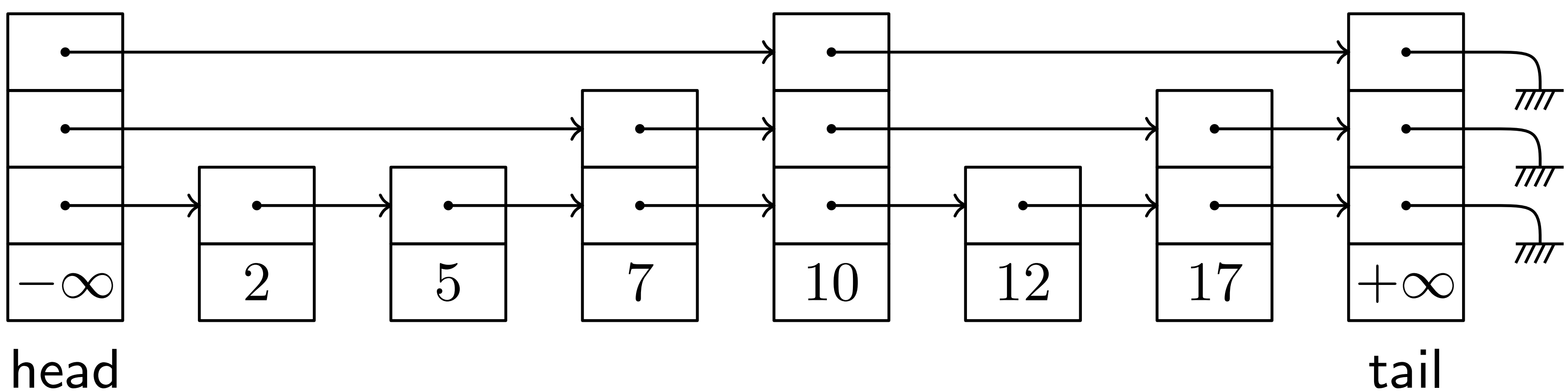
Skiplists

- Sorted list of elements
- Hierarchy of linked lists



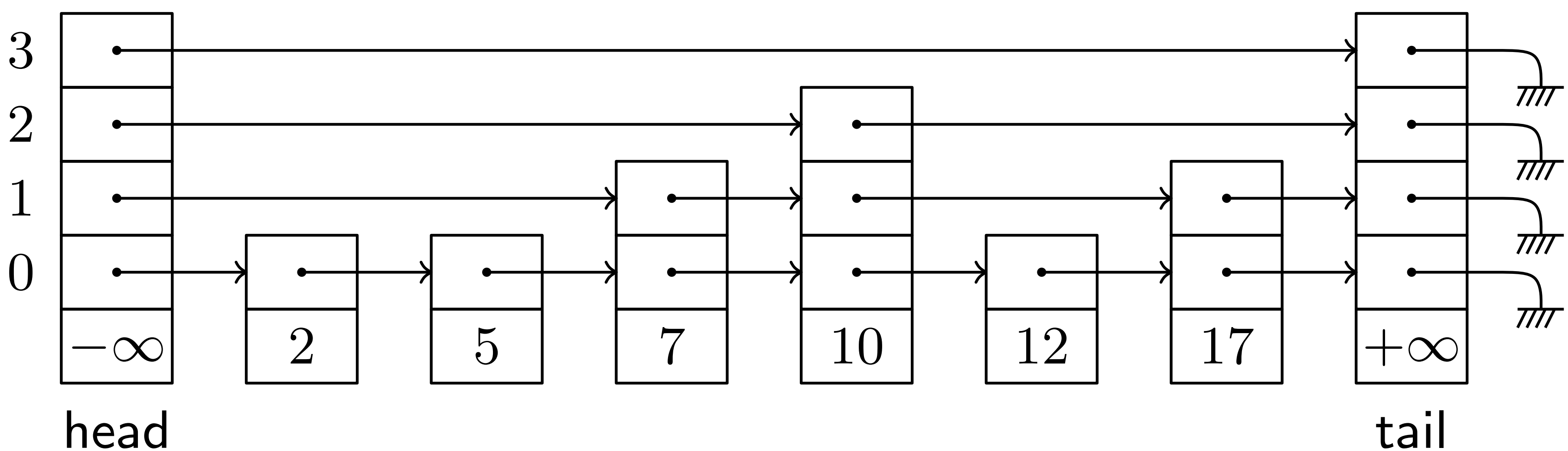
Skiplists

- Sorted list of elements
- Hierarchy of linked lists



Skiplists

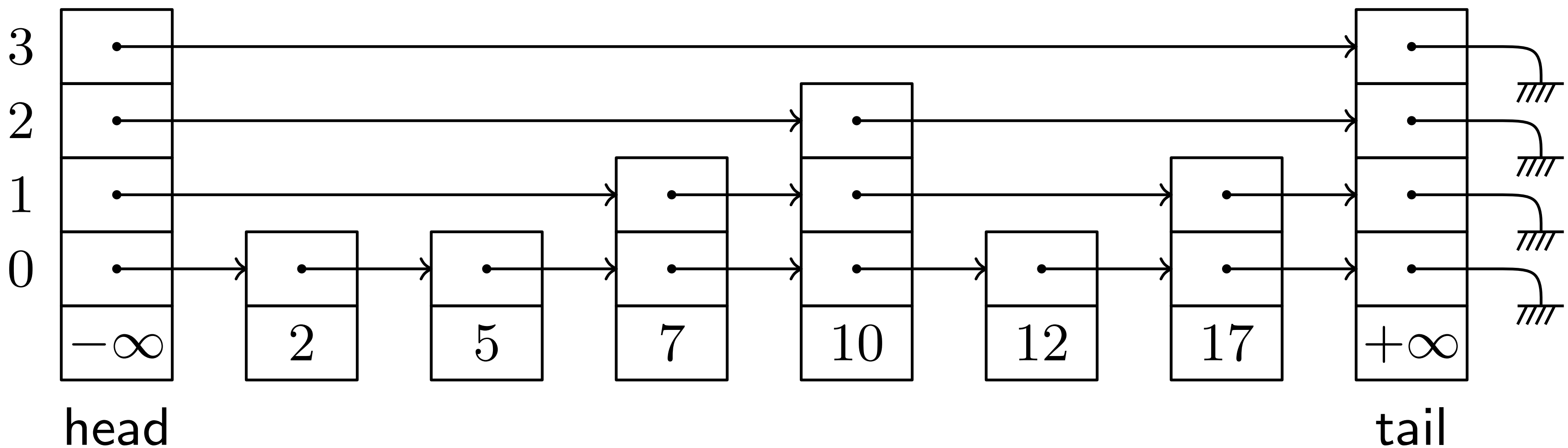
- Sorted list of elements
- Hierarchy of linked lists



Skiplists

- Sorted list of elements
- Hierarchy of linked lists

```
class Skiplist {              class Node {
        Node* head;                   Value v;
        Node* tail;                   Key k;
        int maxLevel;                 Array<Node*>(4) next;
                              }
}
```



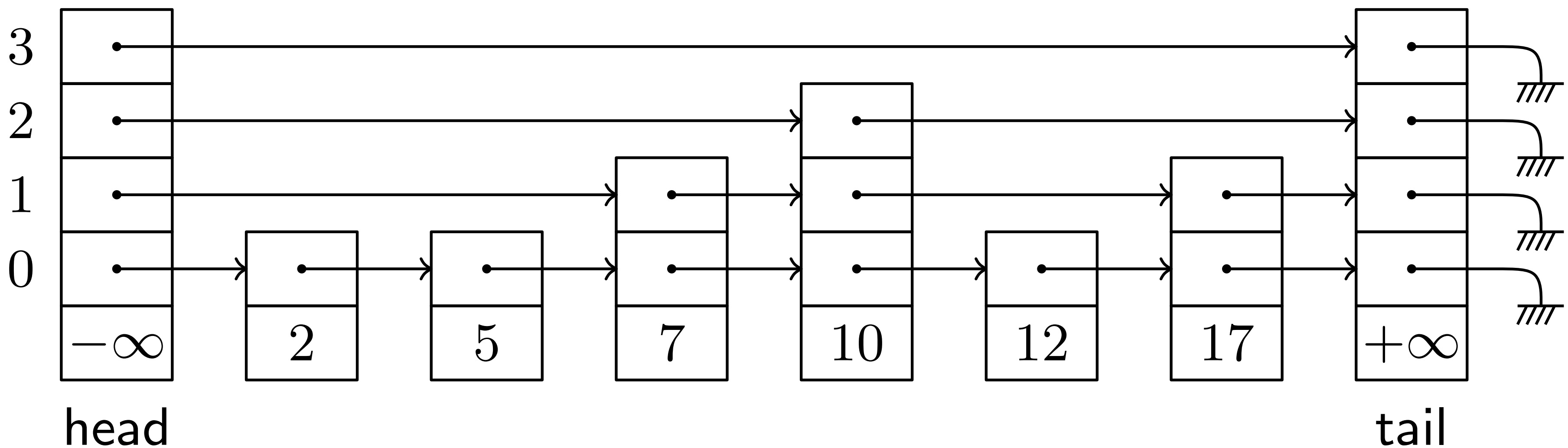
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```



Skiplists

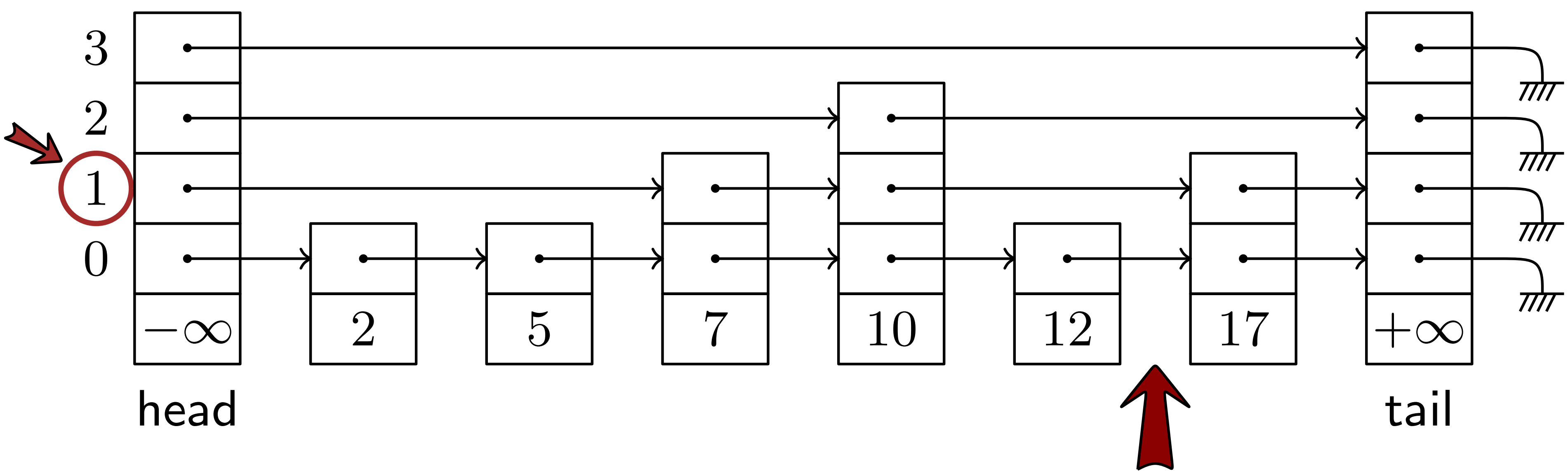
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

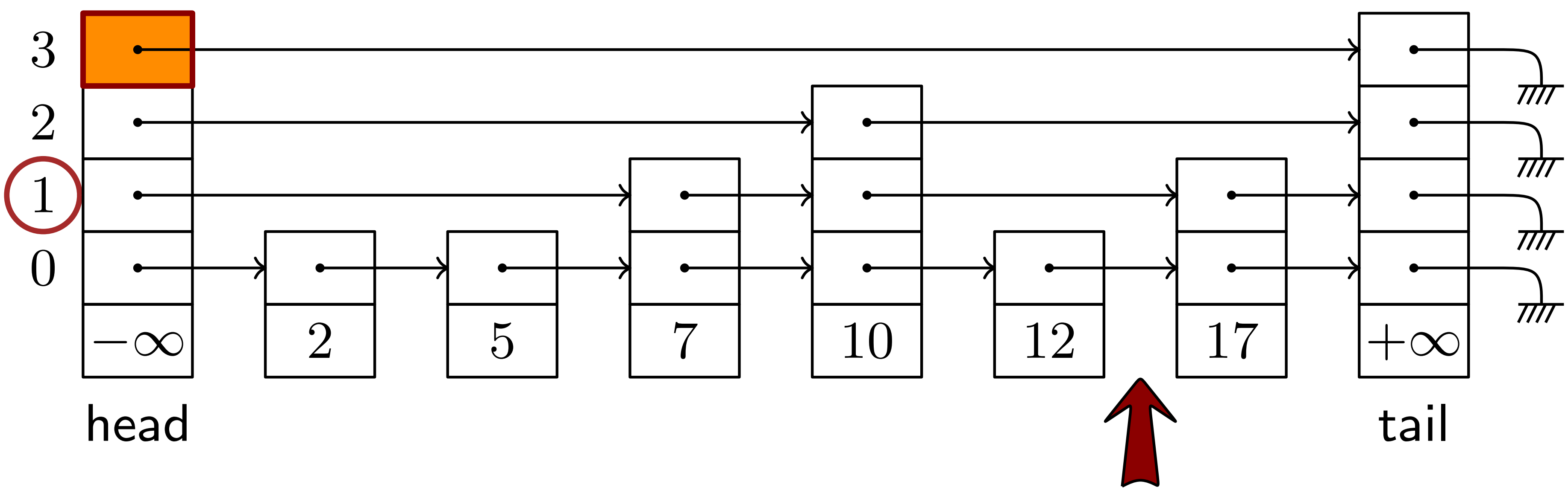
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

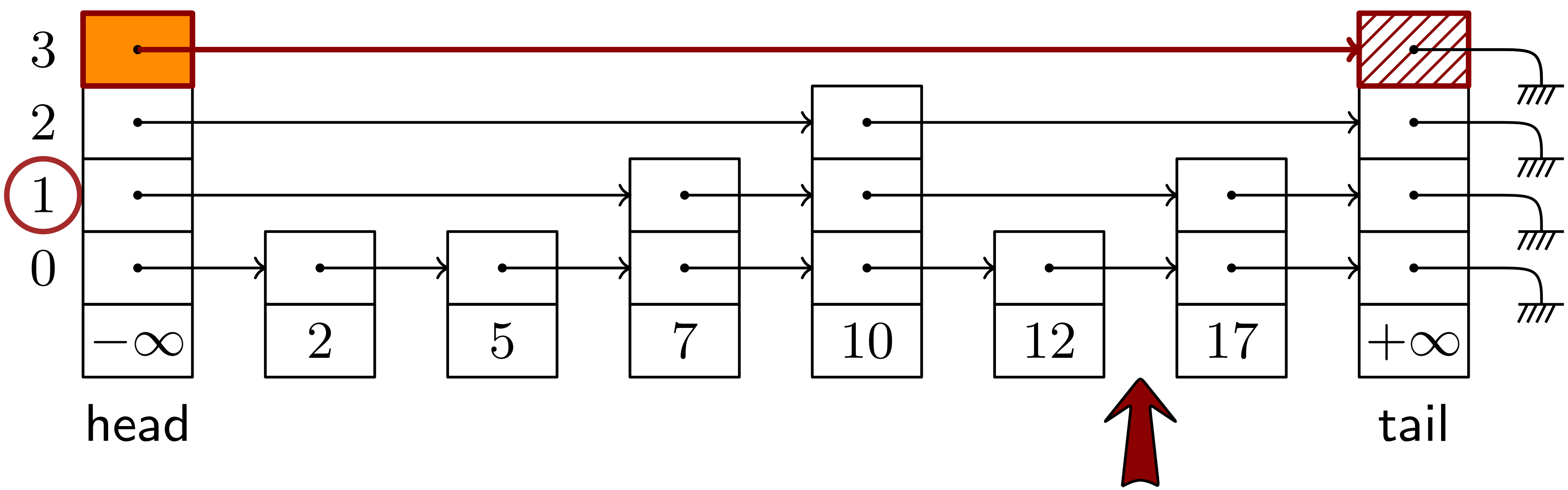
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



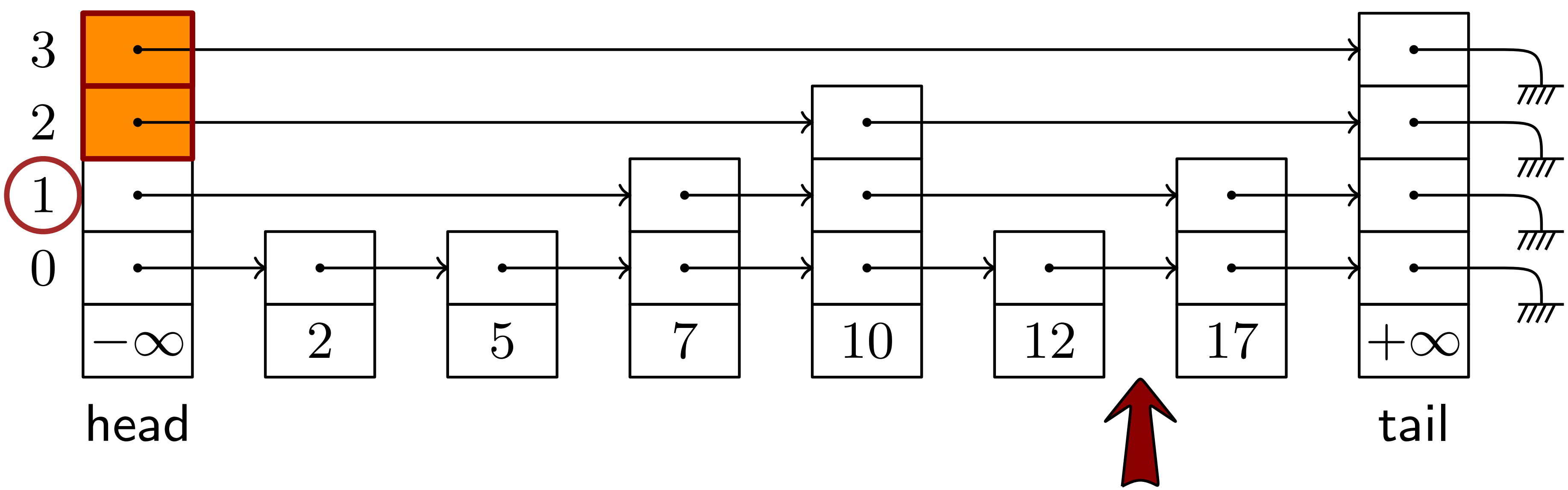
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;
}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

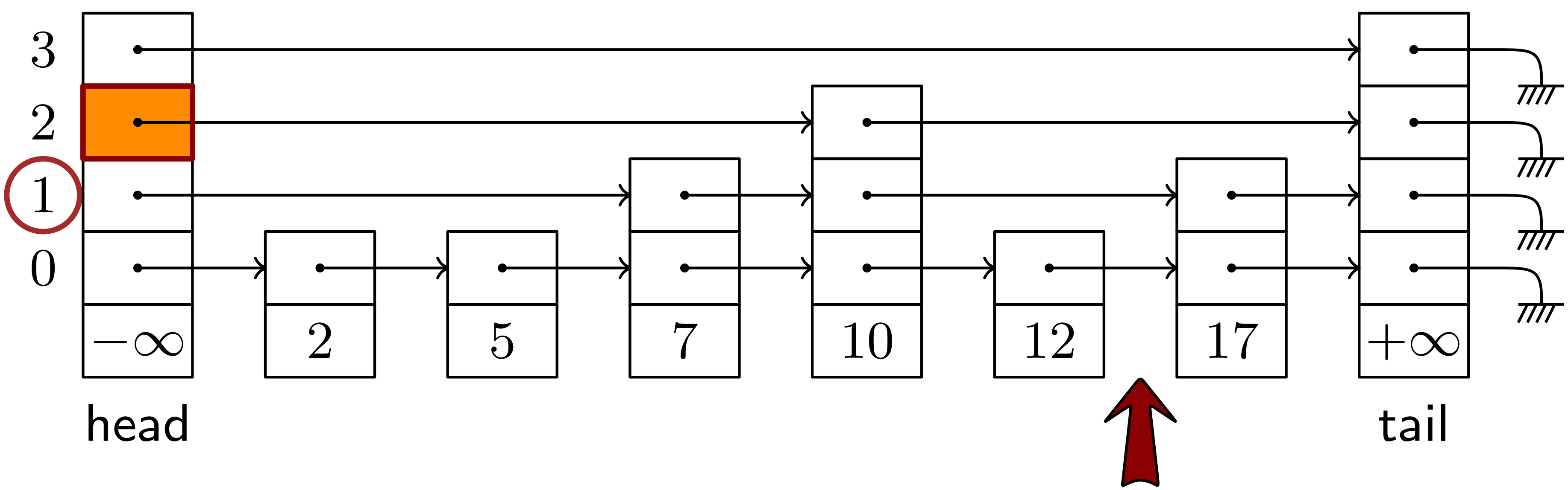
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

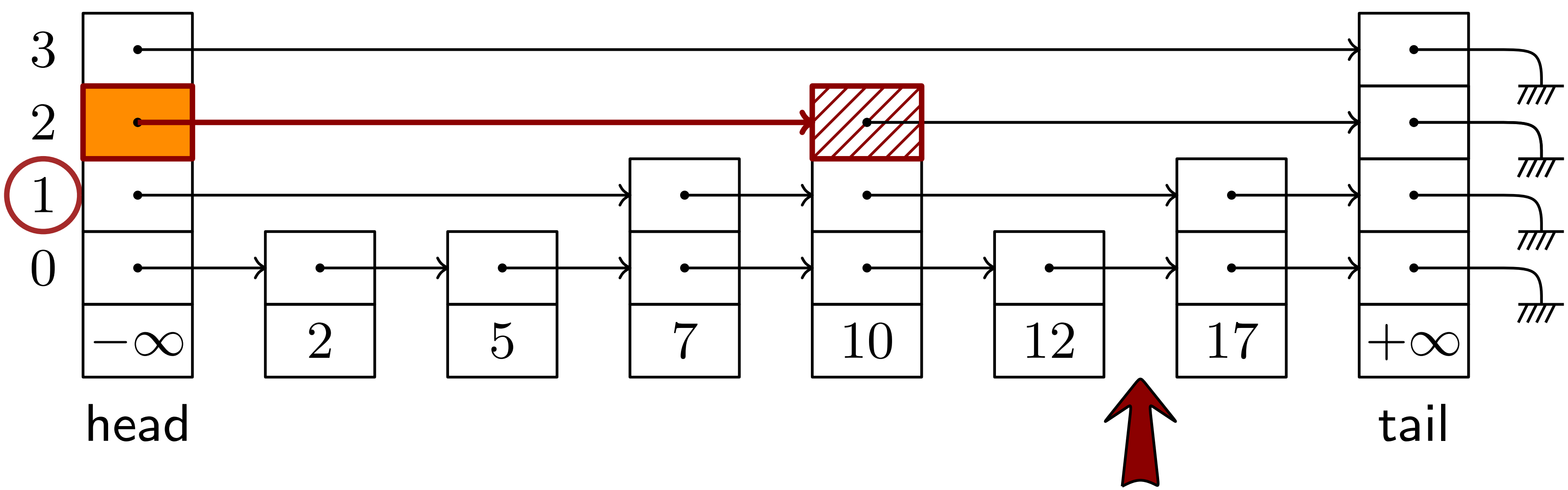
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

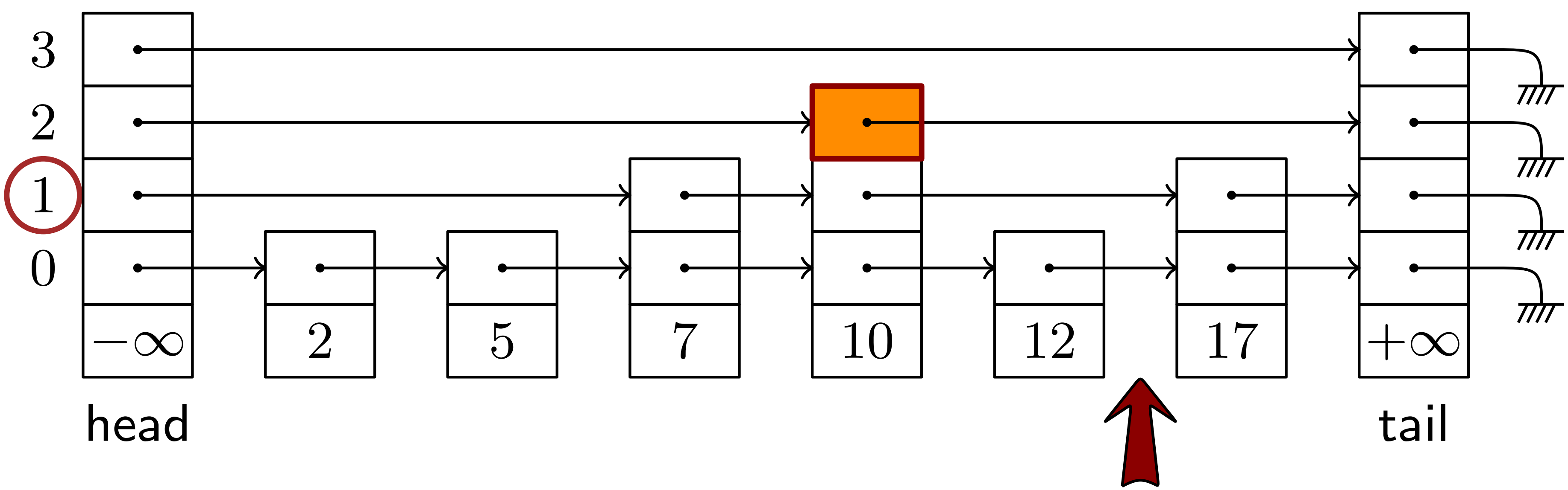
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

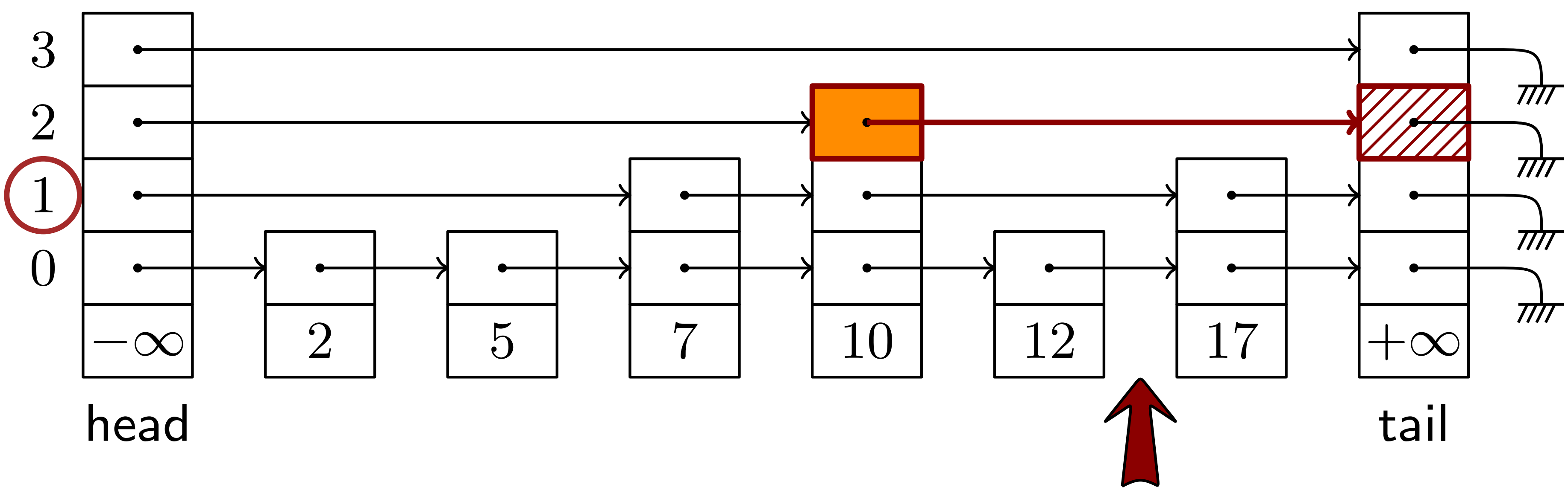
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1

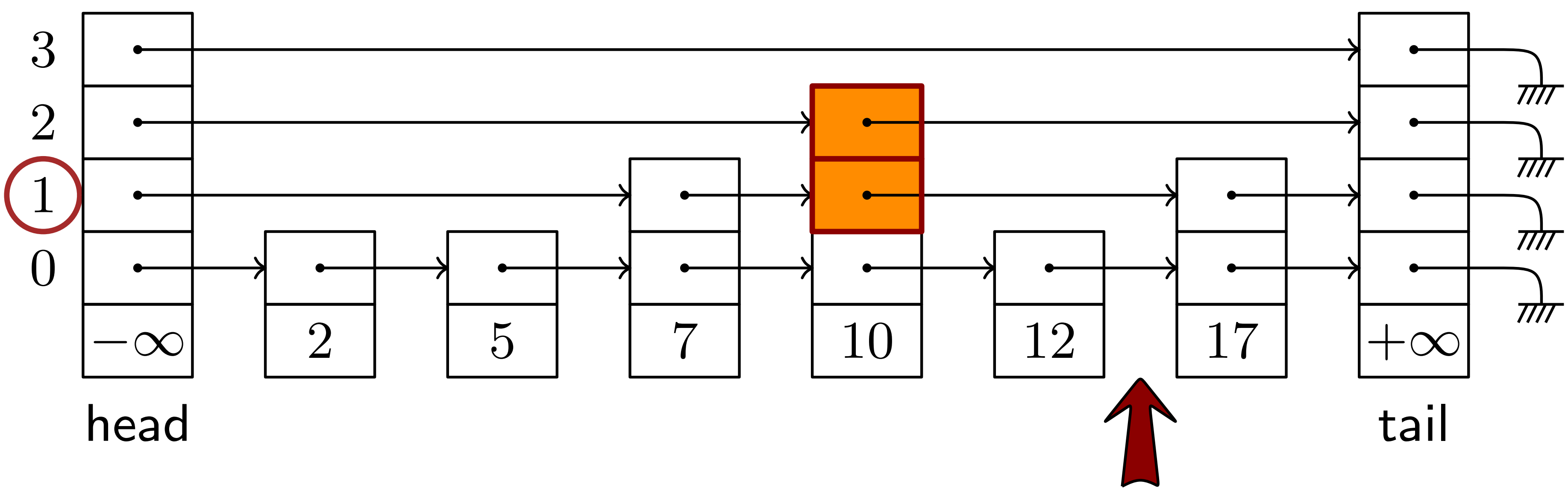


Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {                class Node {
        Node* head;                     Value v;
        Node* tail;                     Key k;
        int maxLevel;                   Array<Node*>(4) next;
                                }
}
```

insert(14) with height 1



Skiplists

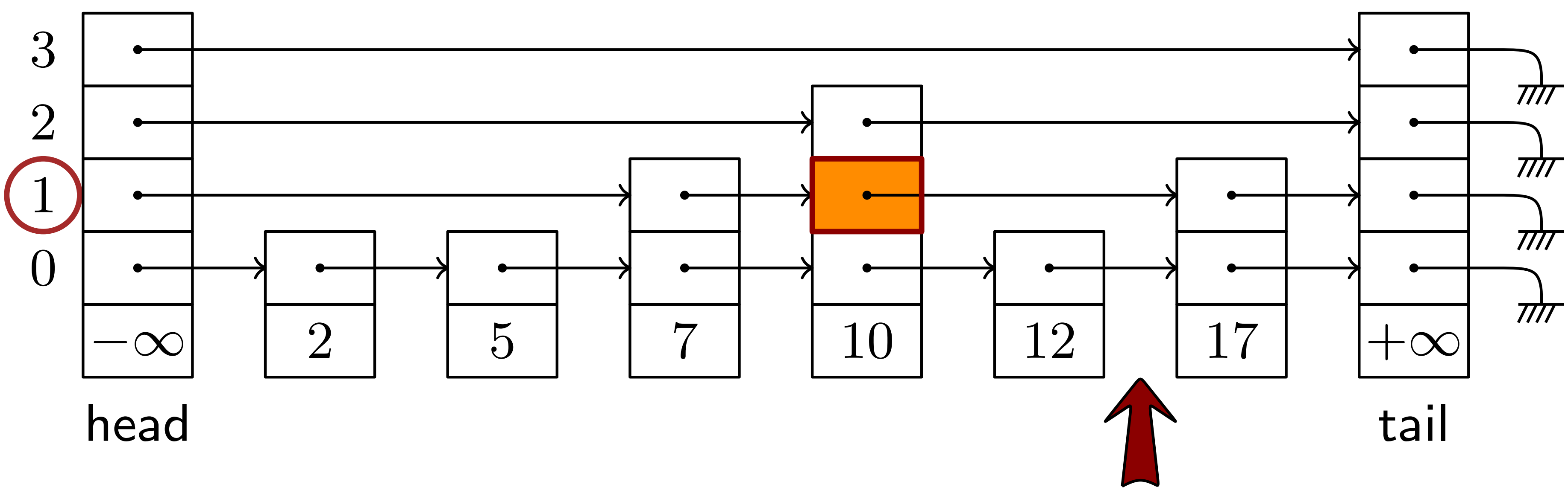
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

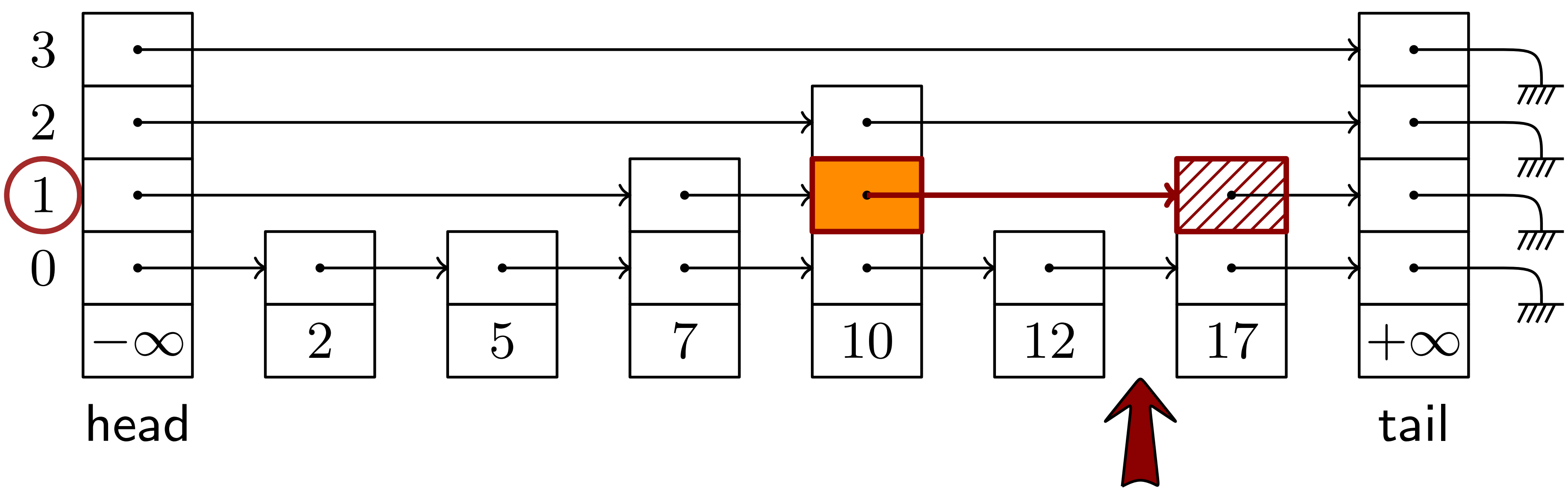
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

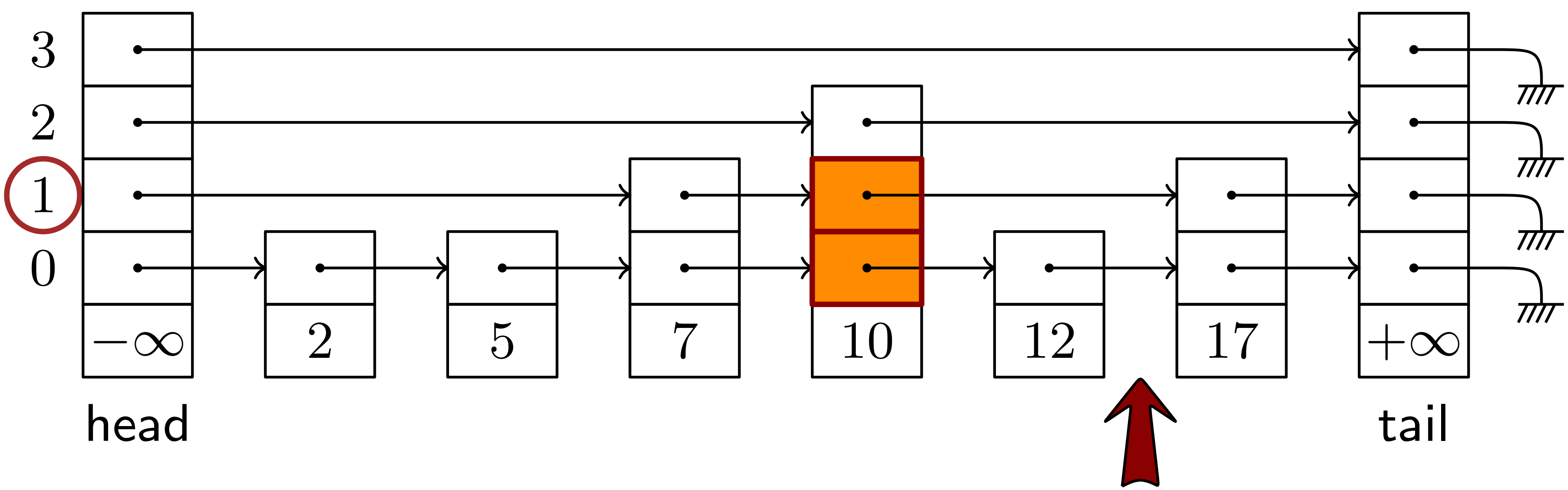
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

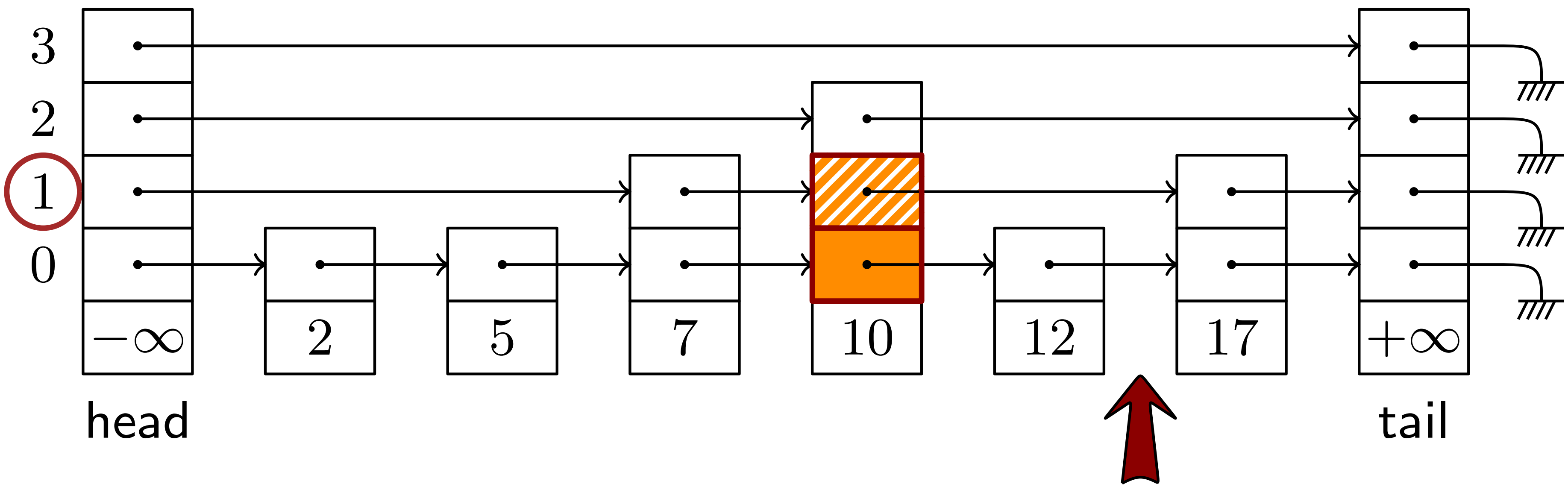
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

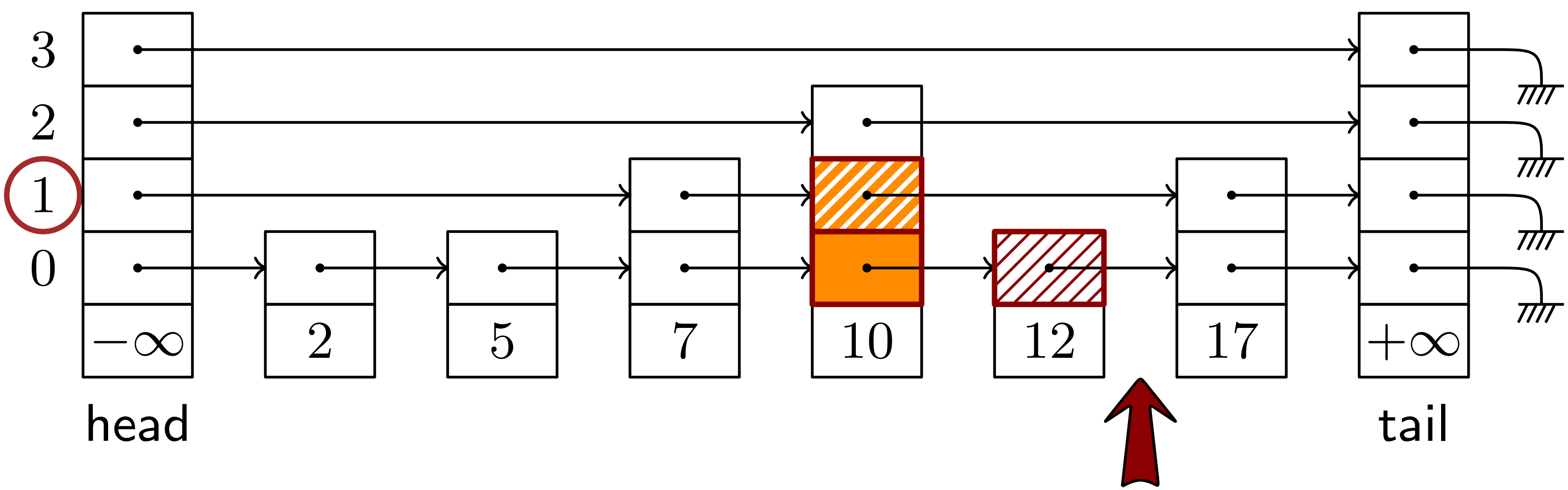
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

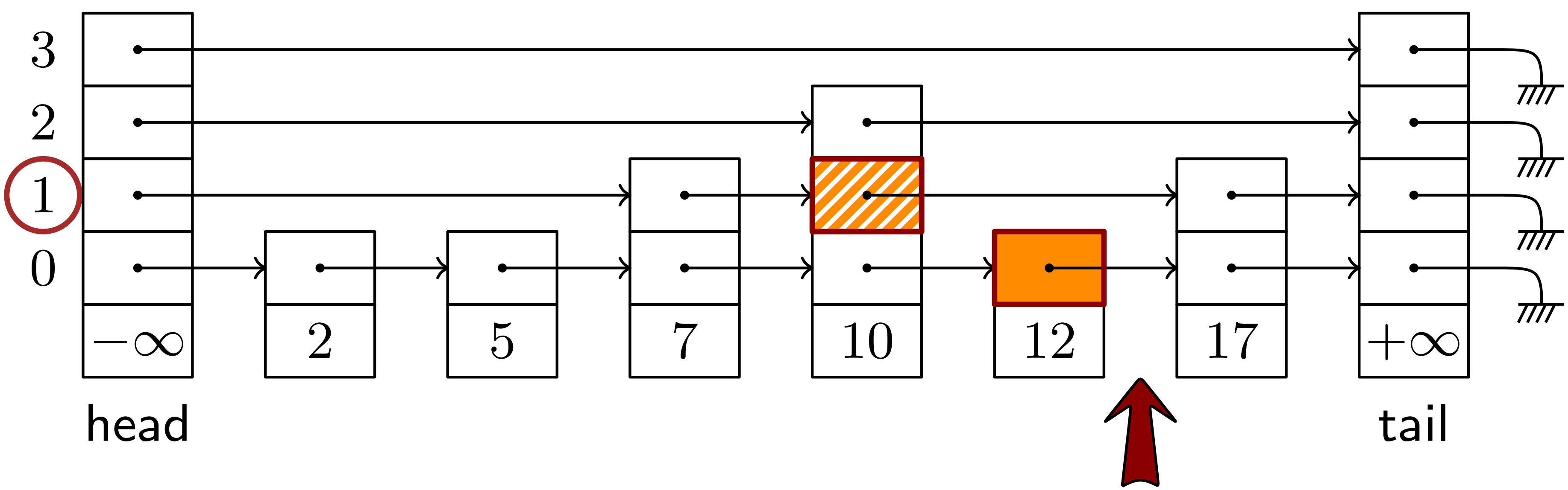
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

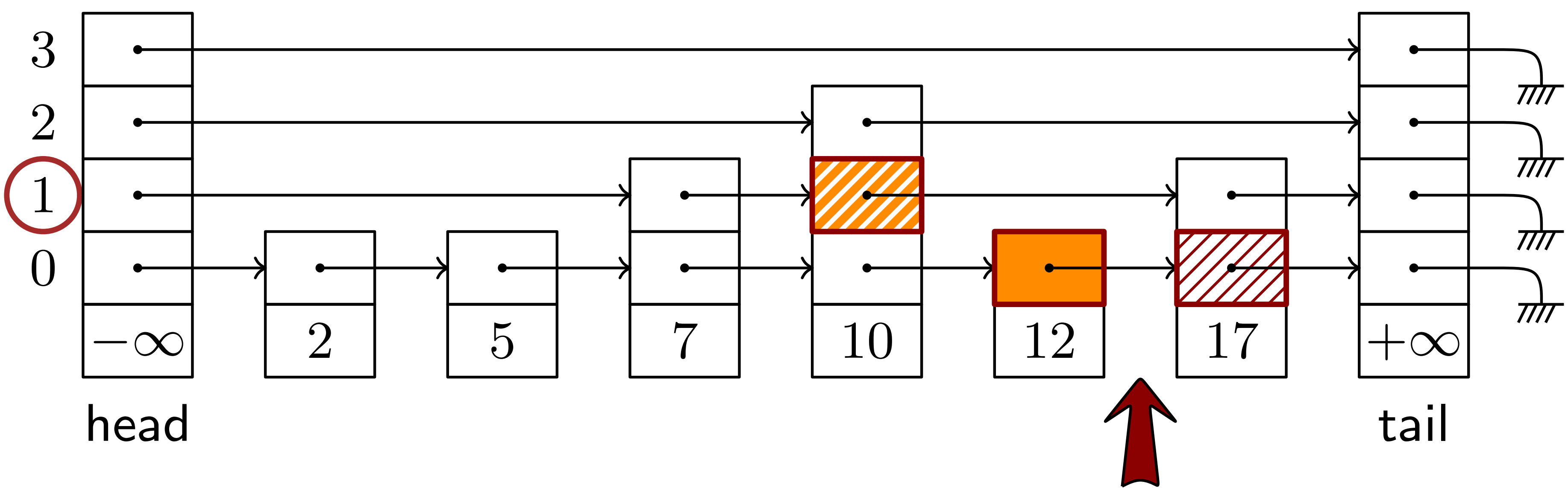
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

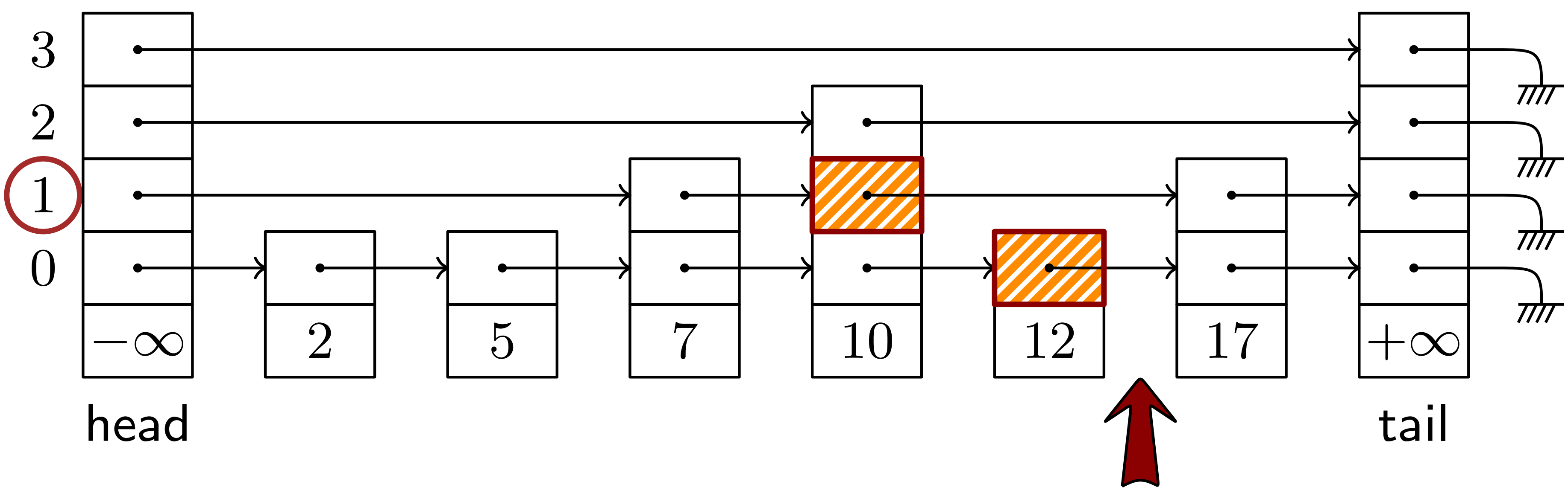
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

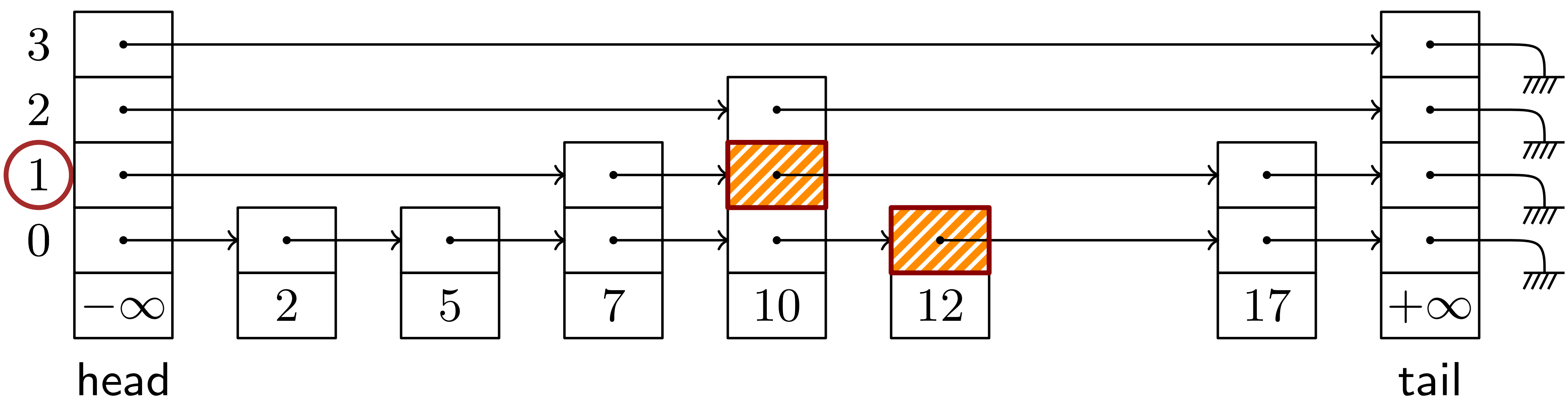
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

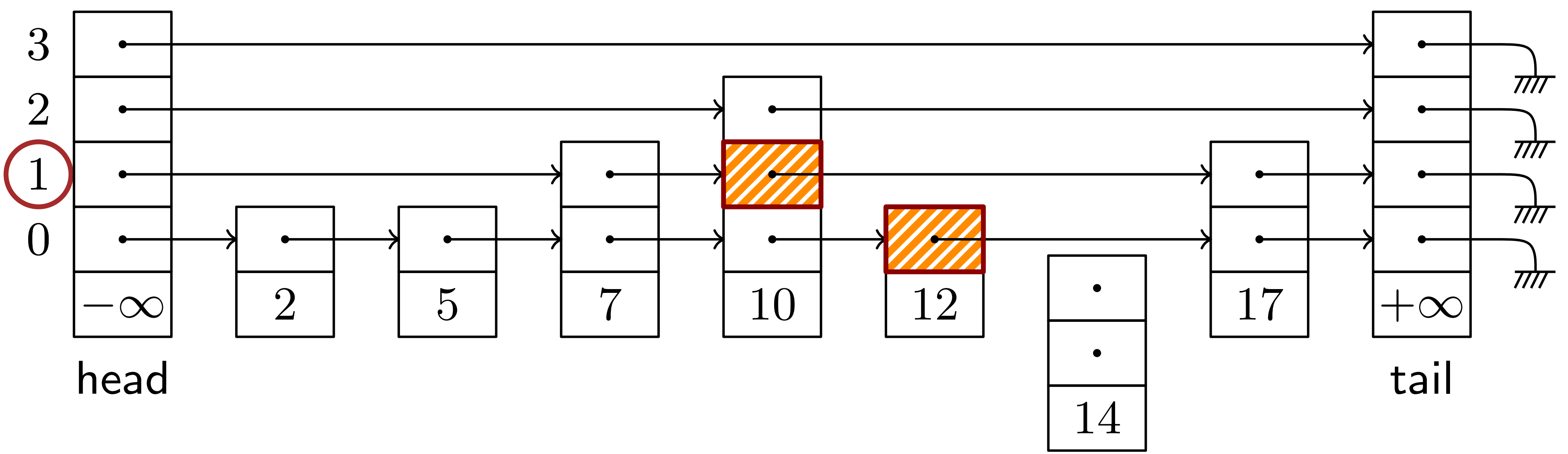
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

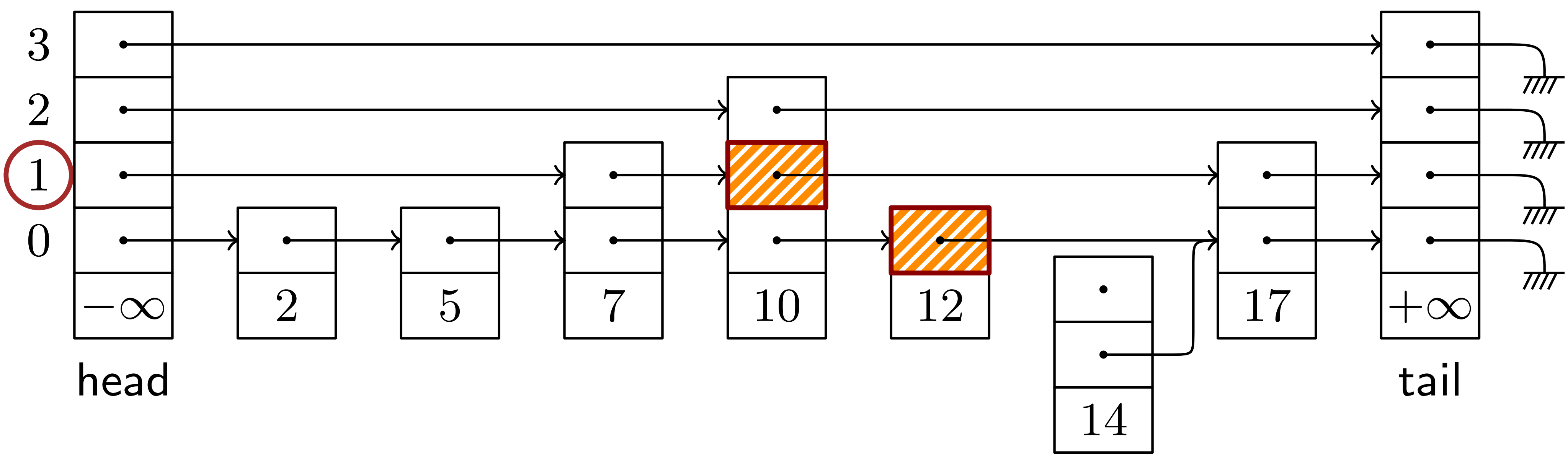
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

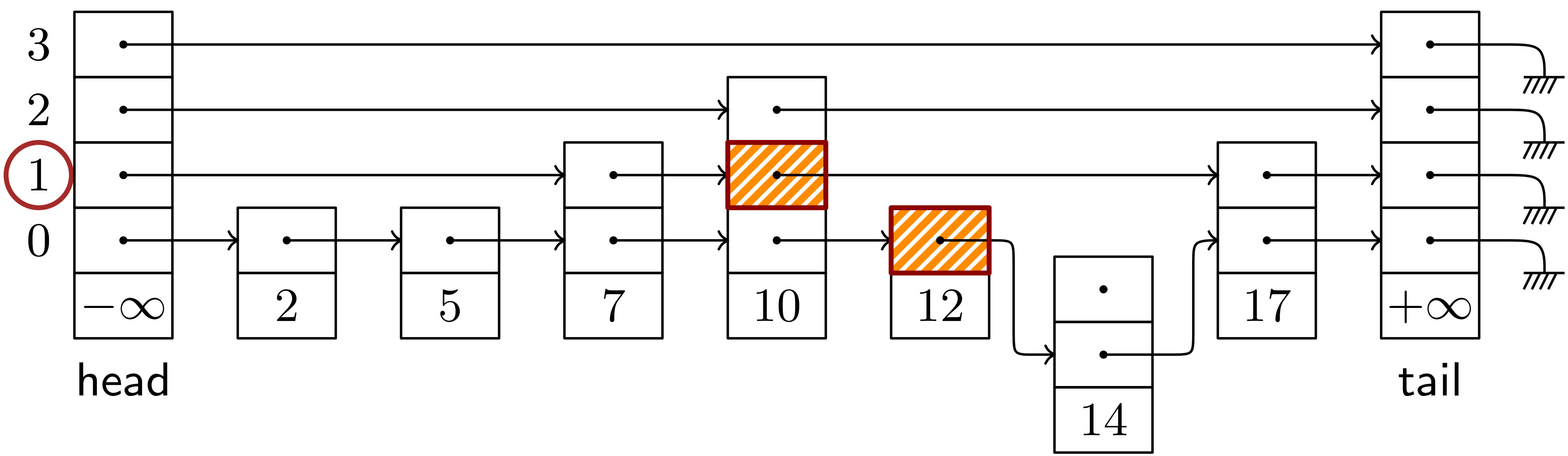
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



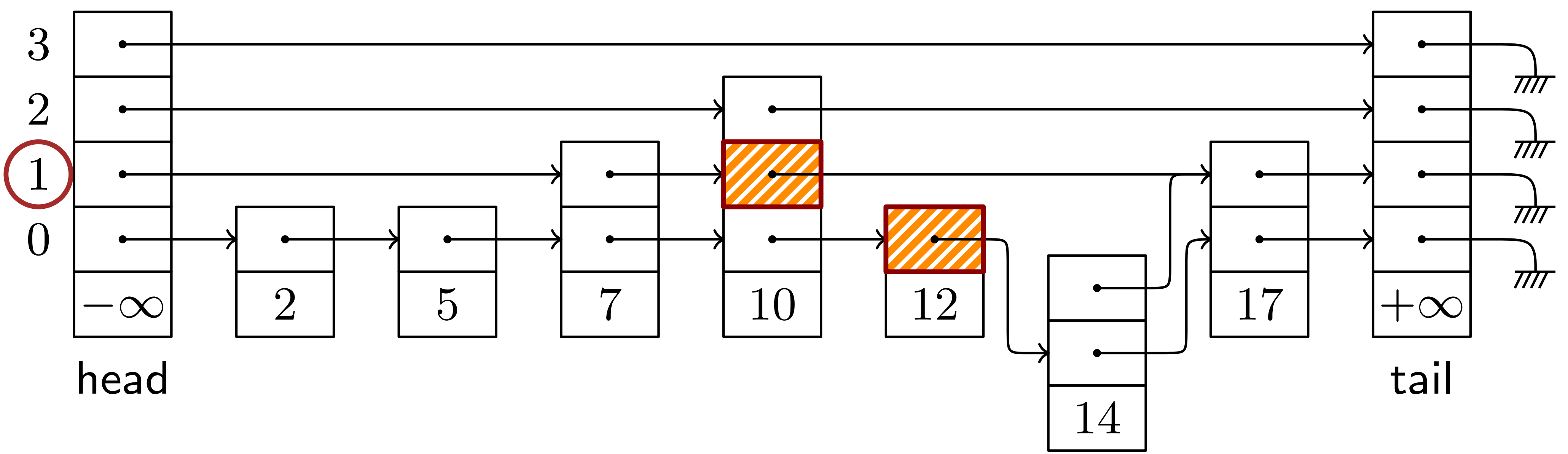
Skiplists

- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;
}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

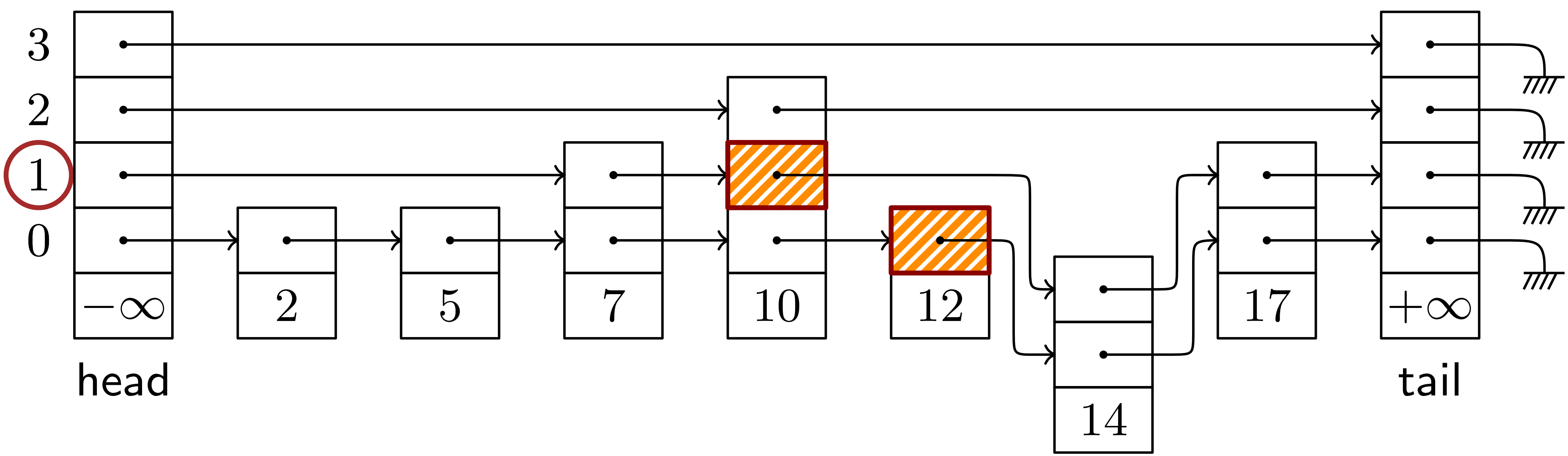
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

insert(14) with height 1



Skiplists

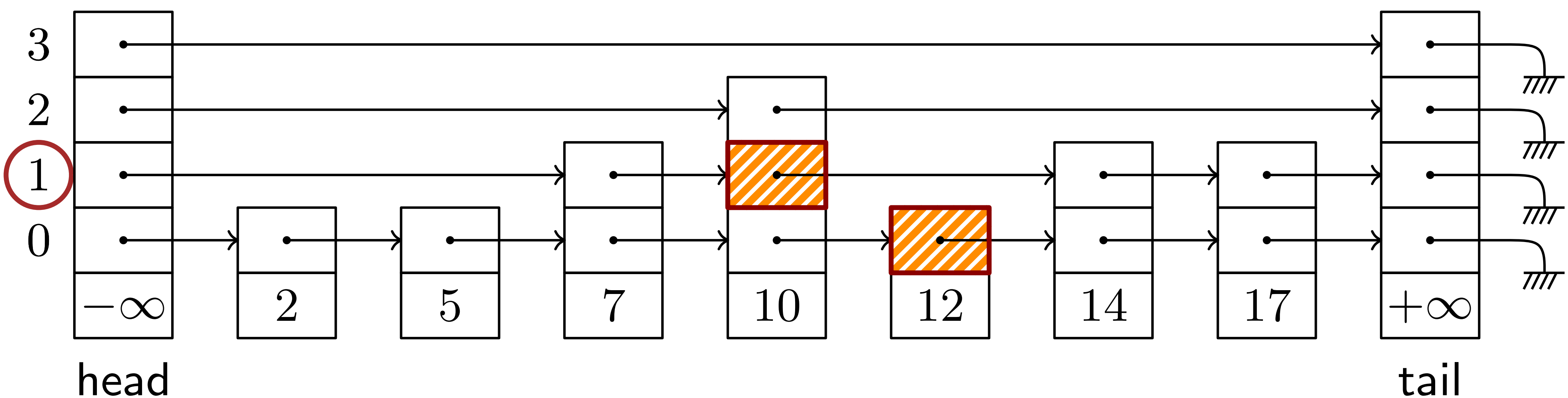
- Sorted list of elements
- Hierarchy of linked lists
- Efficiency comparable to balanced binary search trees

```
class Skiplist {
        Node* head;
        Node* tail;
        int maxLevel;

}
```

```
class Node {
        Value v;
        Key k;
        Array<Node*>(4) next;
}
```

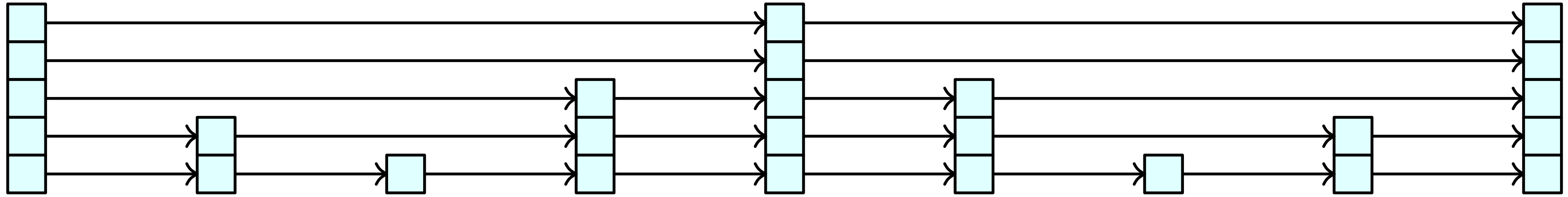
insert(14) with height 1



Why skiplists of unbounded height and length?

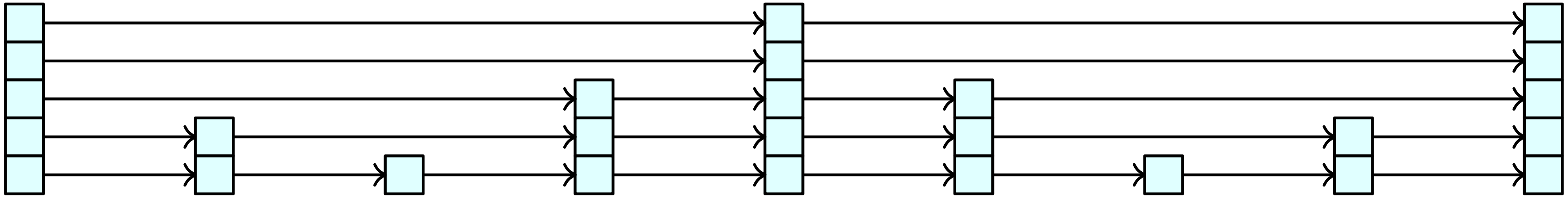
Why skiplists of unbounded height and length?

- While visited EPFL, we developed $\mathbf{TSL_K}$



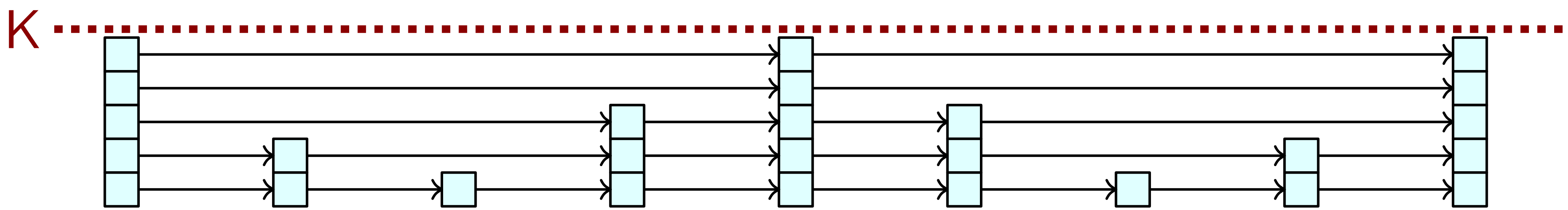
Why skiplists of unbounded height and length?

- While visited EPFL, we developed **TSL$_K$**
  - Show it decidable by finite model property
  - Arbitrary length...



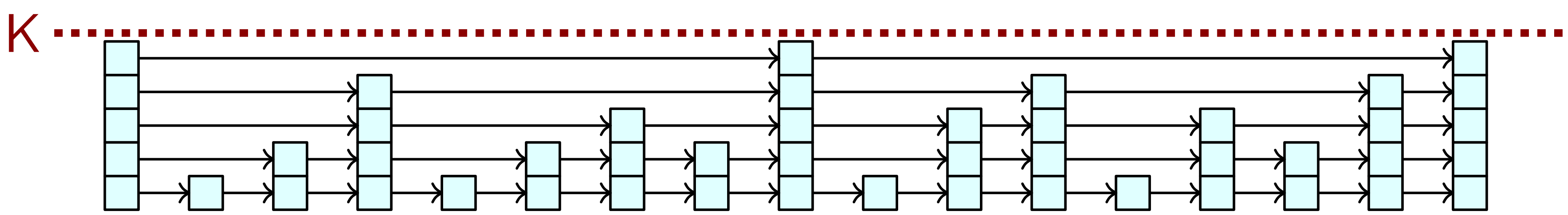
Why skiplists of unbounded height and length?

- While visited EPFL, we developed **TSL_K**
  - Show it decidable by finite model property
  - Arbitrary length...
  - ... but bounded height!



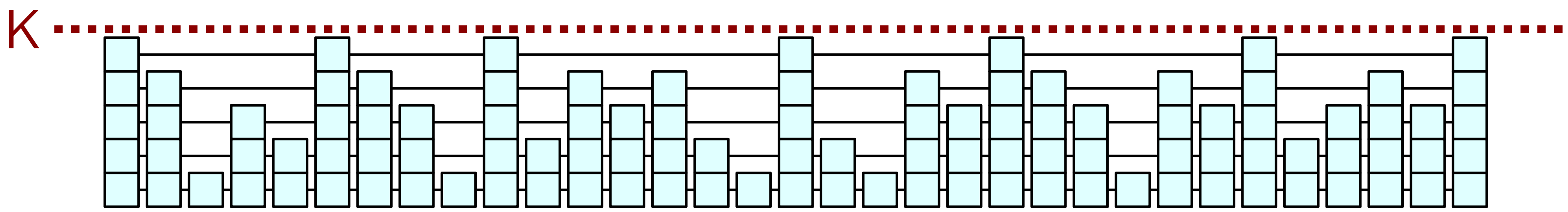
Why skiplists of unbounded height and length?

- While visited EPFL, we developed **TSL_K**
  - Show it decidable by finite model property
  - Arbitrary length...
  - ... but bounded height!



Why skiplists of unbounded height and length?

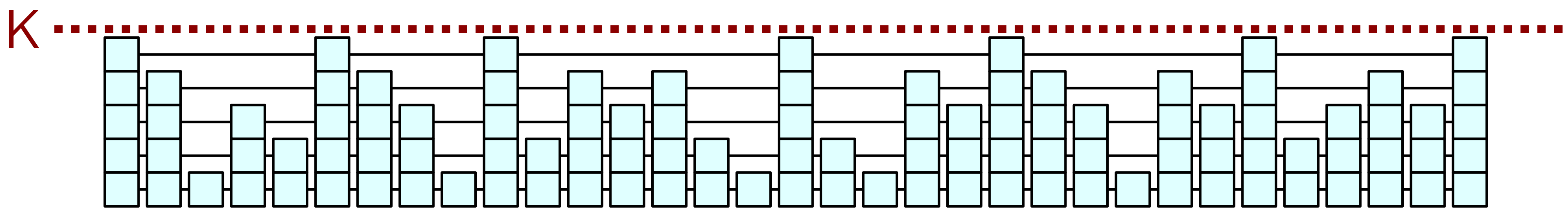
- While visited EPFL, we developed **TSL$_K$**
  - Show it decidable by finite model property
  - Arbitrary length...
  - ... but bounded height!



Why skiplists of unbounded height and length?

- While visited EPFL, we developed **$\mathsf{TSL_K}$**
  - Show it decidable by finite model property
  - Arbitrary length...
  - ... but bounded height!

In practice, **performance is lost!**



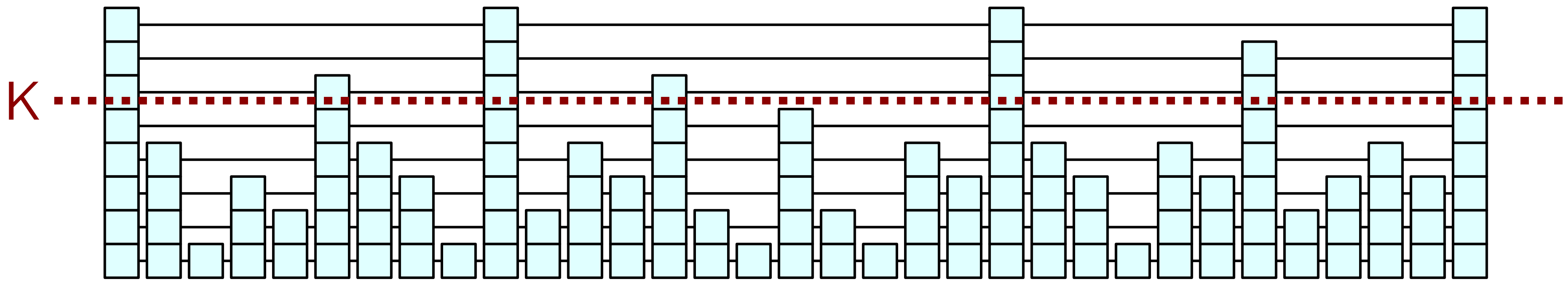
Why skiplists of unbounded height and length?

- While visited EPFL, we developed **TSL**~~$_K$~~
  - Show it decidable by finite model property
  - Arbitrary length...
  - ... but bounded height!

TSL

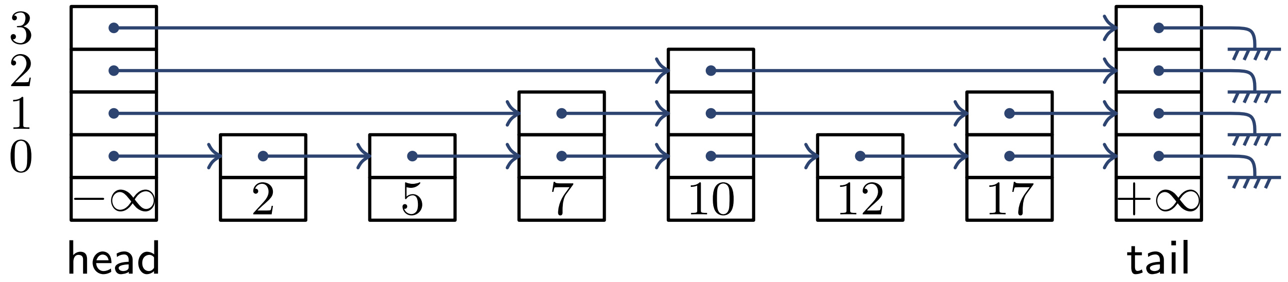
In practice, **performance is lost!**

Dynamic height is required



Verification of Skiplists

$max_H = 3$

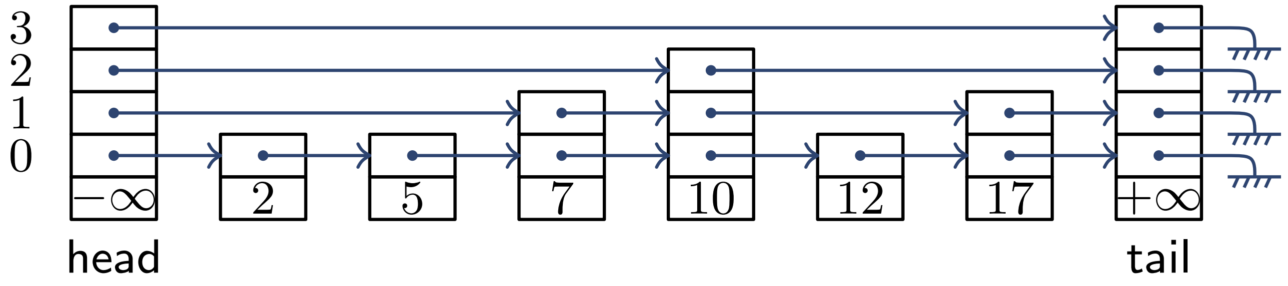


Verification of Skiplists

- **Skiplist** shape **preservation** : $\Box SkipList(h, sl, r)$

$SkipList(h, sl, r) \hat{=}$

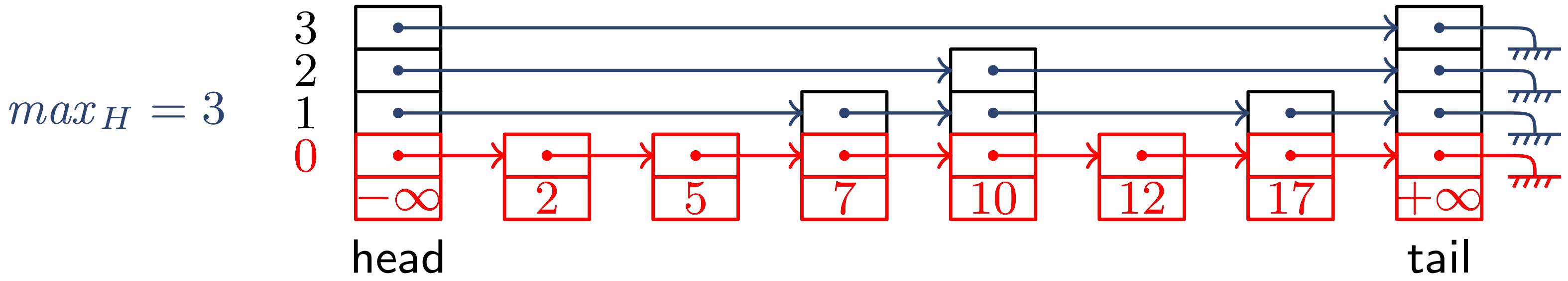
$max_H = 3$



Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$SkipList(h, sl, r) \mathrel{\hat{=}} ordList(m, getp(heap, sl.head, sl.tail, 0))$$

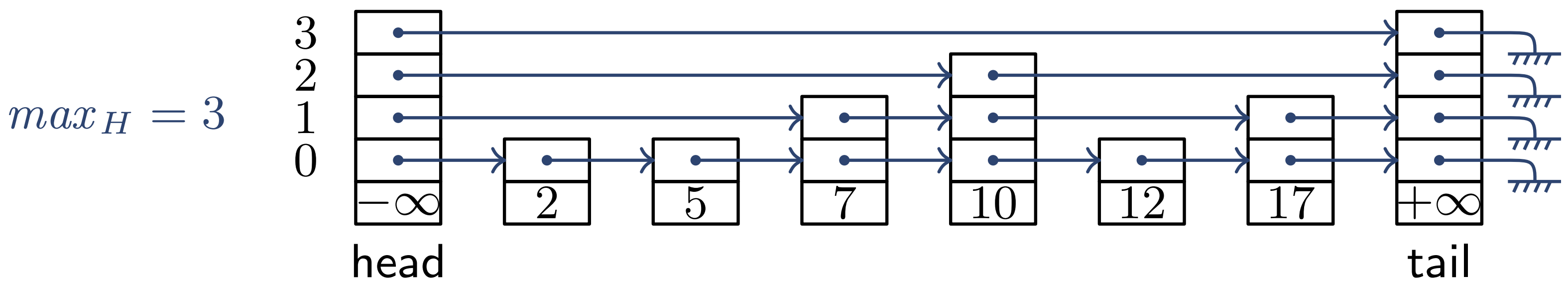


Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$SkipList(h, sl, r) \hat{=}$$

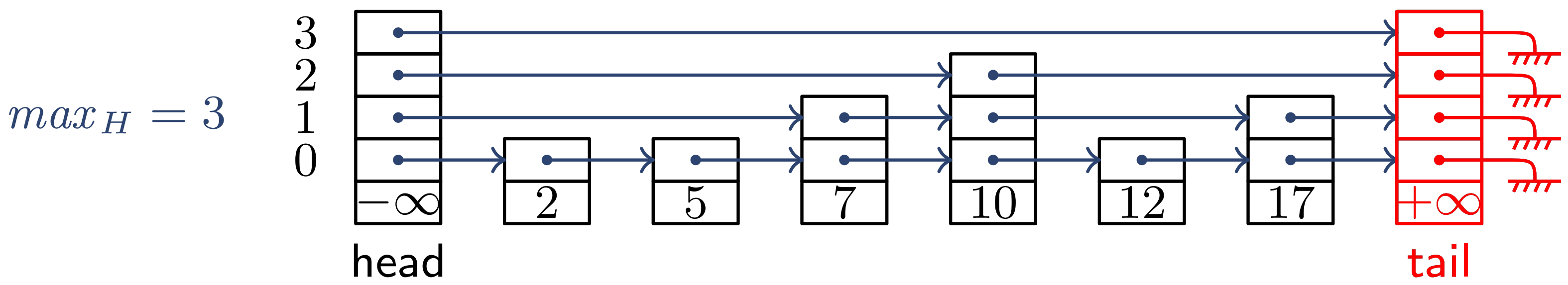
$$ordList(m, getp(heap, sl.head, sl.tail, 0)) \quad \wedge$$
$$r = p2s(getp(heap, sl.head, sl.tail, 0))$$



Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

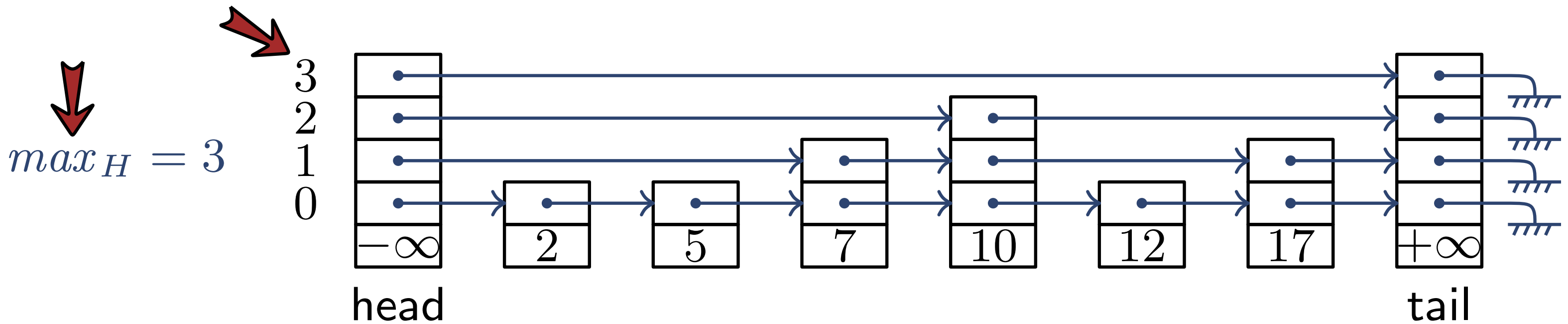
$$
\begin{aligned}
SkipList(h, sl, r) \hat{=}\ & \\
& ordList(m, getp(heap, sl.head, sl.tail, 0)) \quad \wedge \\
& r = p2s(getp(heap, sl.head, sl.tail, 0)) \quad \wedge \\
& rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null
\end{aligned}
$$



Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

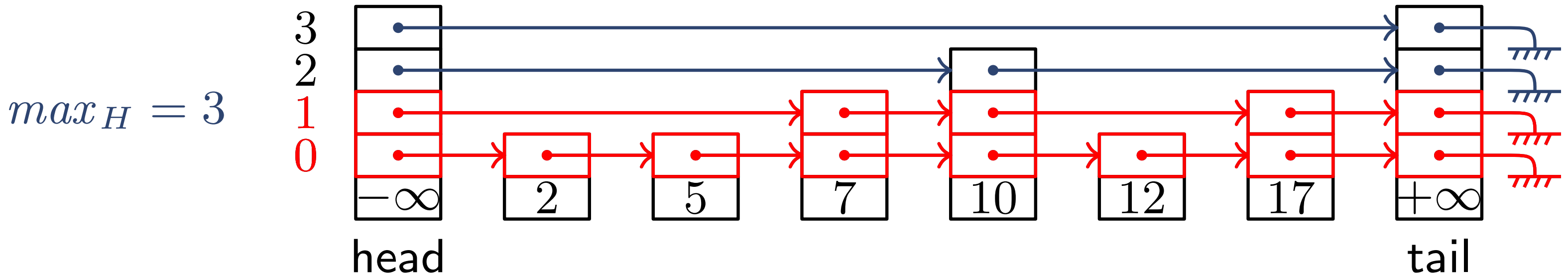
$$
\begin{aligned}
SkipList(h, sl, r) \hat{=} \quad & ordList(m, getp(heap, sl.head, sl.tail, 0)) \quad \land \\
& r = p2s(getp(heap, sl.head, sl.tail, 0)) \quad \land \\
& rd(heap, sl.tail).arr[0] = null \land \cdots \land rd(heap, sl.tail).arr[max_H]) = null \quad \land \\
& a \in r \rightarrow rd(heap, a).level \leq max_H
\end{aligned}
$$



Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$
\begin{aligned}
SkipList(h, sl, r) \mathrel{\hat{=}} & \\
& ordList(m, getp(heap, sl.head, sl.tail, 0)) \quad \wedge \\
& r = p2s(getp(heap, sl.head, sl.tail, 0)) \quad \wedge \\
& rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null \quad \wedge \\
& a \in r \rightarrow rd(heap, a).level \leq max_H \quad \wedge \\
\bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i + 1)) \subseteq p2s(getp(heap, head, tail, i))
\end{aligned}
$$

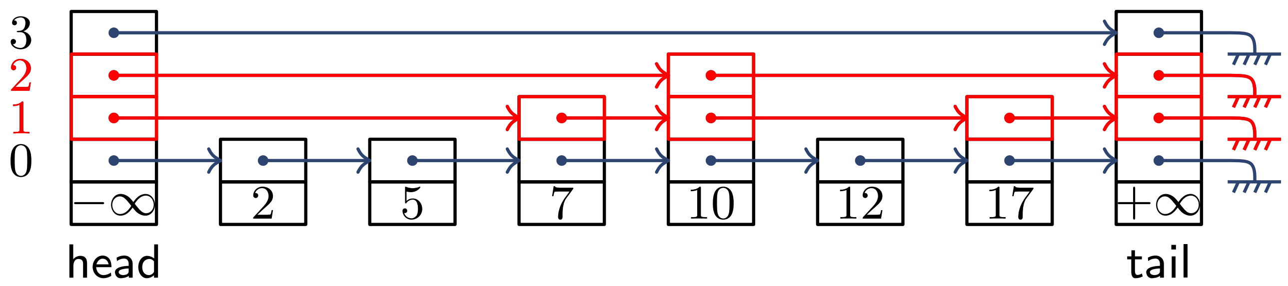


Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$
\begin{array}{rll}
SkipList(h, sl, r) \mathrel{\hat{=}} & & \\
& ordList(m, getp(heap, sl.head, sl.tail, 0)) & \wedge \\
& r = p2s(getp(heap, sl.head, sl.tail, 0)) & \wedge \\
& rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null & \wedge \\
& a \in r \rightarrow rd(heap, a).level \leq max_H & \wedge \\
\bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i + 1)) \subseteq p2s(getp(heap, head, tail, i)) &
\end{array}
$$

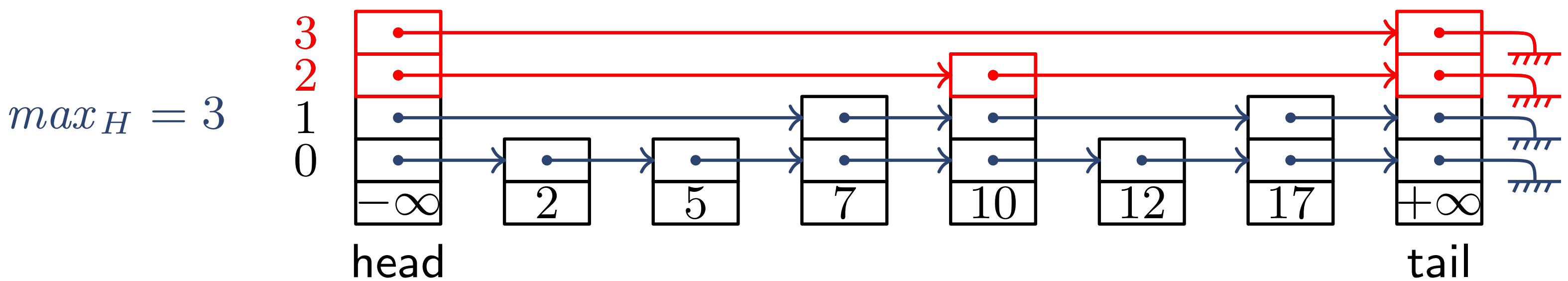
$max_H = 3$



Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$
\begin{array}{lll}
SkipList(h, sl, r) \hat{=} & & \\
& ordList(m, getp(heap, sl.head, sl.tail, 0)) & \wedge \\
& r = p2s(getp(heap, sl.head, sl.tail, 0)) & \wedge \\
& rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null & \wedge \\
& a \in r \to rd(heap, a).level \leq max_H & \wedge \\
\bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i + 1)) \subseteq p2s(getp(heap, head, tail, i)) &
\end{array}
$$



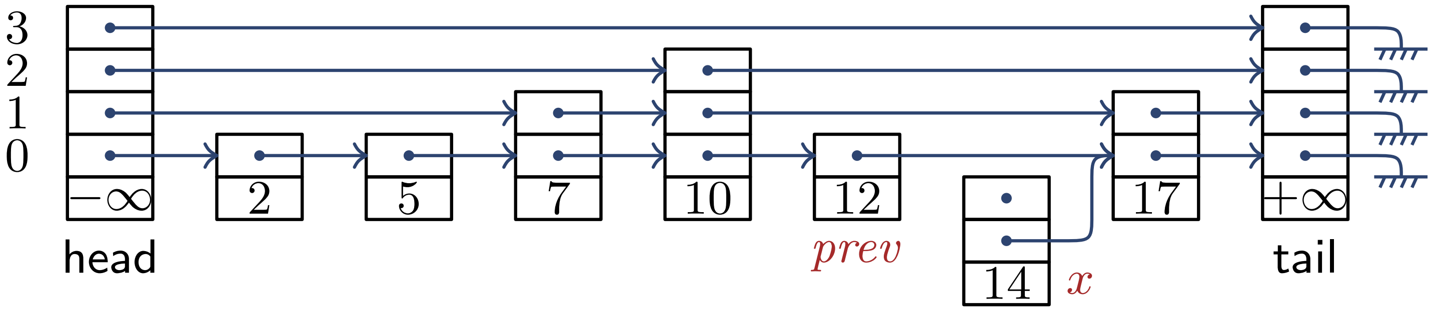
Verification of Skiplists

- **Skiplist** shape **preservation** : $\Box SkipList(h, sl, r)$

$$
\begin{array}{lll}
SkipList(h, sl, r) \mathrel{\hat{=}} & & \\
 & ordList(m, getp(heap, sl.head, sl.tail, 0)) & \wedge \\
 & r = p2s(getp(heap, sl.head, sl.tail, 0)) & \wedge \\
 & rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null & \wedge \\
 & a \in r \rightarrow rd(heap, a).level \leq max_H & \wedge \\
\bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i+1)) \subseteq p2s(getp(heap, head, tail, i)) &
\end{array}
$$

- Program **transitions** :

```
35: . . .
36: prev.arr[0] := x
37: . . .
```



Verification of Skiplists

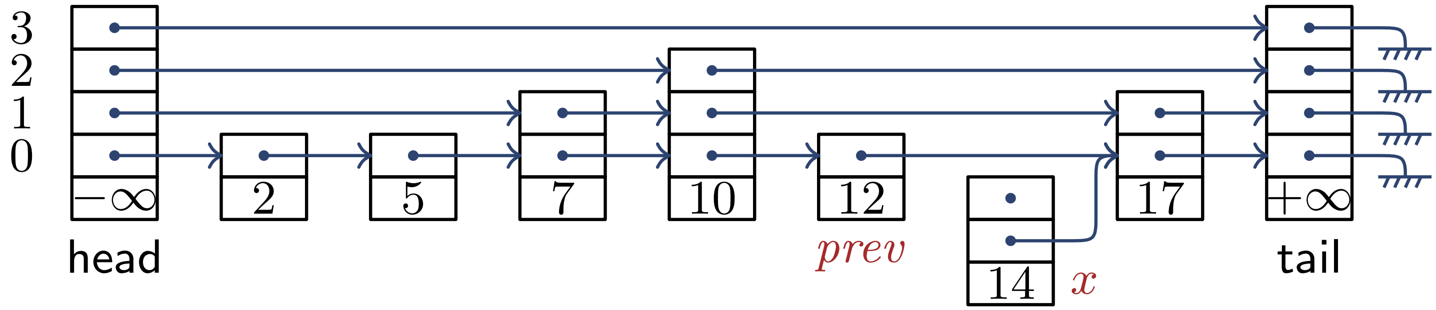
- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$
\begin{aligned}
SkipList(h, sl, r) \mathrel{\hat{=}} \quad & \\
& ordList(m, getp(heap, sl.head, sl.tail, 0)) & \wedge \\
& r = p2s(getp(heap, sl.head, sl.tail, 0)) & \wedge \\
& rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null & \wedge \\
& a \in r \rightarrow rd(heap, a).level \leq max_H & \wedge \\
\textstyle\bigwedge_{i \in 0 \ldots (max_H - 1)} \quad & p2s(getp(heap, head, tail, i + 1)) \subseteq p2s(getp(heap, head, tail, i)) &
\end{aligned}
$$

- Program **transitions** : $SL(h, sl, r)$

$SkipList(h, sl, r)$

```
35: . . .
36: prev.arr[0] := x
37: . . .
```



Verification of Skiplists

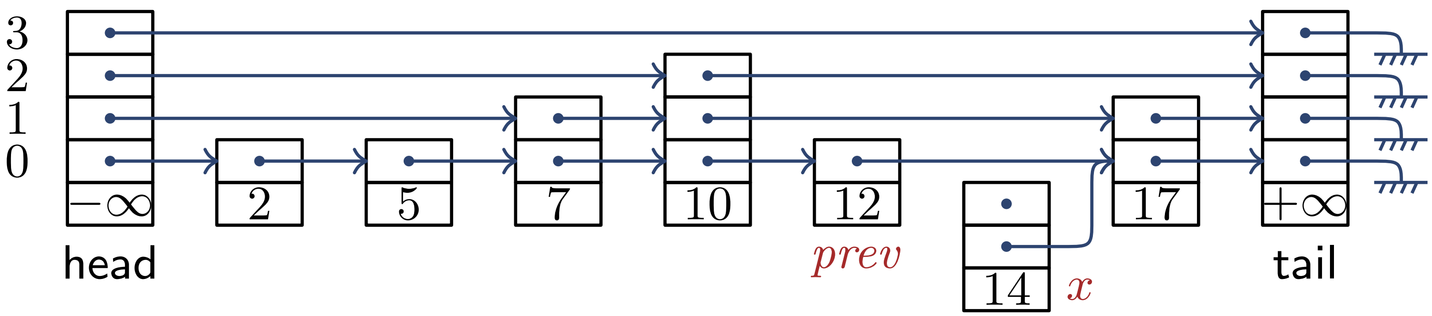
- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$SkipList(h,sl,r) \mathrel{\hat{=}} \begin{array}{ll} & ordList(m, getp(heap, sl.head, sl.tail, 0)) \quad \wedge \\ & r = p2s(getp(heap, sl.head, sl.tail, 0)) \quad \wedge \\ & rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null \quad \wedge \\ & a \in r \to rd(heap, a).level \leq max_H \quad \wedge \\ \bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i+1)) \subseteq p2s(getp(heap, head, tail, i)) \end{array}$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux}$

$$SkipList(h, sl, r) \wedge \left(\begin{array}{rcll} x.key & = & 14 & \wedge \\ prev.key & < & 14 & \wedge \\ x.arr[0].key & > & 14 & \wedge \\ prev.arr[0] & = & x.arr[0] & \wedge \\ & & x \notin r \wedge 0 \leq 0 \leq 3 & \end{array} \right)$$

```
35: . . .
36: prev.arr[0] := x
37: . . .
```



Verification of Skiplists

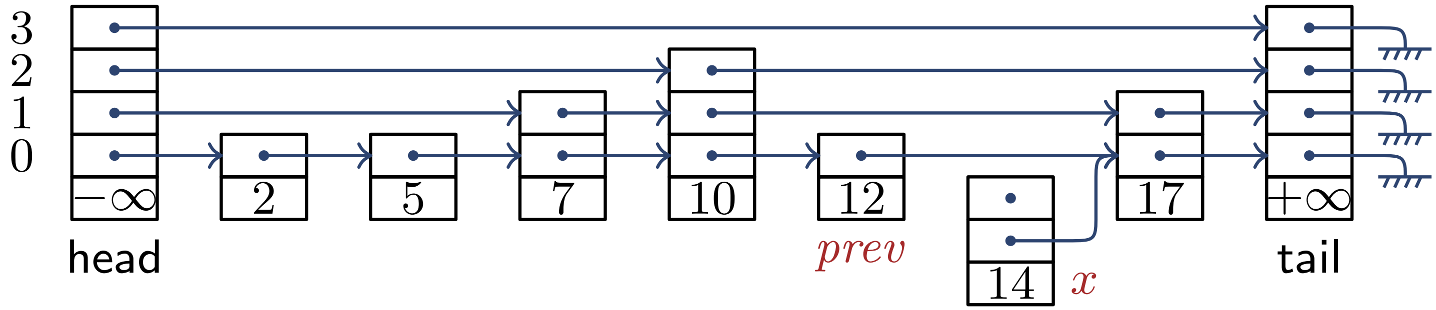
- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$
\begin{aligned}
SkipList(h, sl, r) \mathrel{\hat{=}} & \\
& ordList(m, getp(heap, sl.head, sl.tail, 0)) \wedge \\
& r = p2s(getp(heap, sl.head, sl.tail, 0)) \wedge \\
& rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null \wedge \\
& a \in r \rightarrow rd(heap, a).level \leq max_H \wedge \\
\bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i+1)) \subseteq p2s(getp(heap, head, tail, i))
\end{aligned}
$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux} \wedge \rho_{36}(V, V')$

$$
SkipList(h, sl, r) \wedge
\begin{pmatrix}
x.key & = & 14 & \wedge \\
prev.key & < & 14 & \wedge \\
x.arr[0].key & > & 14 & \wedge \\
prev.arr[0] & = & x.arr[0] & \wedge \\
\multicolumn{3}{c}{x \notin r \wedge 0 \leq 0 \leq 3} &
\end{pmatrix}
\wedge
\begin{pmatrix}
at_{36} & \wedge \\
prev'.arr[0] = x & \wedge \\
at'_{37} & \wedge \\
h' = h \wedge sl = sl' & \wedge \\
r' = r \cup \{x\} \wedge x' = x & \ldots
\end{pmatrix}
$$

```
35: . . .
36: prev.arr[0] := x
37: . . .
```



Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

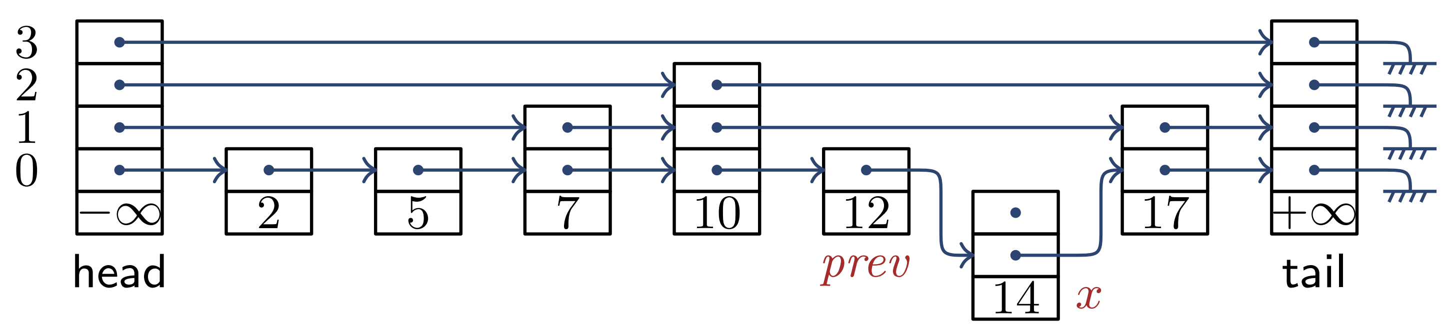
$$
\begin{array}{rll}
SkipList(h, sl, r) \mathrel{\hat{=}} & & \\
 & ordList(m, getp(heap, sl.head, sl.tail, 0)) & \wedge \\
 & r = p2s(getp(heap, sl.head, sl.tail, 0)) & \wedge \\
 & rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null & \wedge \\
 & a \in r \to rd(heap, a).level \leq max_H & \wedge \\
 \bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i+1)) \subseteq p2s(getp(heap, head, tail, i)) &
\end{array}
$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux} \wedge \rho_{36}(V, V')$

$$
SkipList(h, sl, r) \wedge
$$

$$
\left(
\begin{array}{rcll}
x.key & = & 14 & \wedge \\
prev.key & < & 14 & \wedge \\
x.arr[0].key & > & 14 & \wedge \\
prev.arr[0] & = & x.arr[0] & \wedge \\
\multicolumn{3}{c}{x \notin r \wedge 0 \leq 0 \leq 3} &
\end{array}
\right)
\wedge
\left(
\begin{array}{cl}
at_{36} & \wedge \\
prev'.arr[0] = x & \wedge \\
at'_{37} & \wedge \\
h' = h \wedge sl = sl' & \wedge \\
r' = r \cup \{x\} \wedge x' = x & \ldots
\end{array}
\right)
$$

```
35: . . .
36: prev.arr[0] := x
37: . . .
```



Verification of Skiplists

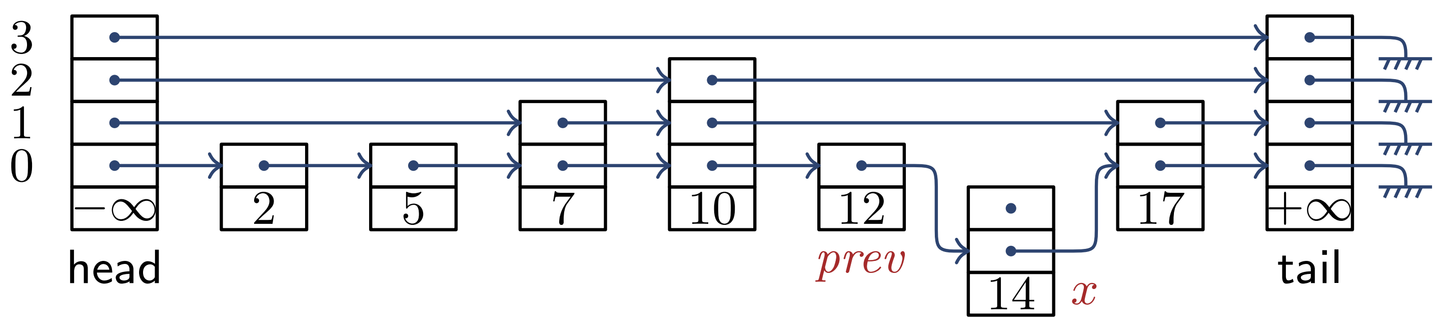
- **Skiplist** shape **preservation** : $\Box SkipList(h, sl, r)$

$$
\begin{array}{lll}
SkipList(h,sl,r) \hat{=} & & \\
 & ordList(m, getp(heap, sl.head, sl.tail, 0)) & \wedge \\
 & r = p2s(getp(heap, sl.head, sl.tail, 0)) & \wedge \\
 & rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null & \wedge \\
 & a \in r \rightarrow rd(heap, a).level \leq max_H & \wedge \\
 \bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i+1)) \subseteq p2s(getp(heap, head, tail, i)) &
\end{array}
$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux} \wedge \rho_{36}(V, V')$

$$
SkipList(h,sl,r) \wedge
\begin{pmatrix}
x.key & = & 14 & \wedge \\
prev.key & < & 14 & \wedge \\
x.arr[0].key & > & 14 & \wedge \\
prev.arr[0] & = & x.arr[0] & \wedge \\
\multicolumn{3}{c}{x \notin r \wedge 0 \leq 0 \leq 3} &
\end{pmatrix}
\wedge
\begin{pmatrix}
at_{36} & \wedge \\
prev'.arr[0] = x & \wedge \\
at'_{37} & \wedge \\
h' = h \wedge sl = sl' & \wedge \\
r' = r \cup \{x\} \wedge x' = x & \ldots
\end{pmatrix}
$$

```
35: . . .
36: prev.arr[0] := x
37: . . .
```



Verification of Skiplists

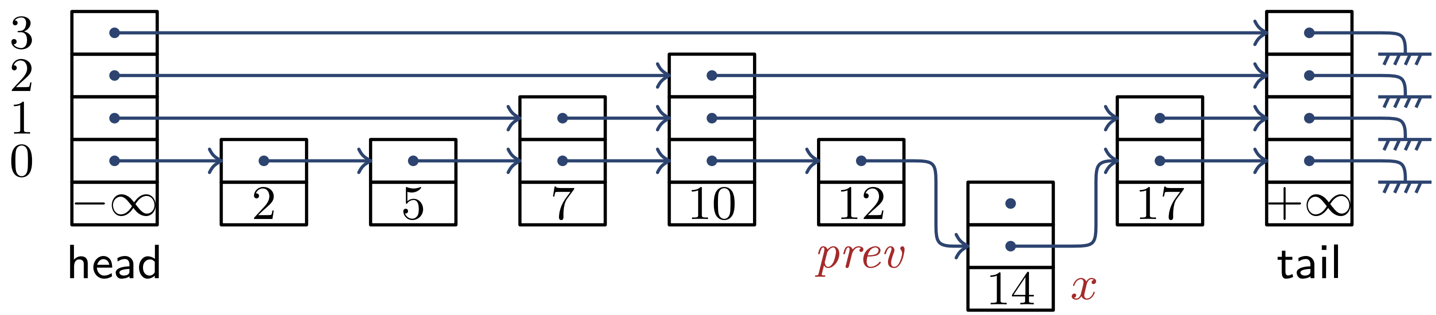
- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$
\begin{aligned}
SkipList(h, sl, r) \mathrel{\hat{=}} & \\
& ordList(m, getp(heap, sl.head, sl.tail, 0)) && \wedge \\
& r = p2s(getp(heap, sl.head, sl.tail, 0)) && \wedge \\
& rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null && \wedge \\
& a \in r \rightarrow rd(heap, a).level \leq max_H && \wedge \\
\bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i + 1)) \subseteq p2s(getp(heap, head, tail, i))
\end{aligned}
$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux} \wedge \rho_{36}(V, V') \rightarrow SL(h', sl', r')$

$$
SkipList(h, sl, r) \wedge
\begin{pmatrix}
x.key & = & 14 & \wedge \\
prev.key & < & 14 & \wedge \\
x.arr[0].key & > & 14 & \wedge \\
prev.arr[0] & = & x.arr[0] & \wedge \\
\multicolumn{3}{c}{x \notin r \wedge 0 \leq 0 \leq 3} &
\end{pmatrix}
\wedge
\begin{pmatrix}
at_{36} & \wedge \\
prev'.arr[0] = x & \wedge \\
at'_{37} & \wedge \\
h' = h \wedge sl = sl' & \wedge \\
r' = r \cup \{x\} \wedge x' = x & \ldots
\end{pmatrix}
\rightarrow SkipList(h', sl', r')
$$

```
35: . . .
36: prev.arr[0] := x
37: . . .
```



Verification of Skiplists

- **Skiplist** shape **preservation** : $\Box \mathit{SkipList}(h, sl, r)$

$$
\begin{array}{lll}
\mathit{SkipList}(h, sl, r) \mathrel{\hat{=}} & & \\
& \mathit{ordList}(m, \mathit{getp}(\mathit{heap}, sl.\mathit{head}, sl.\mathit{tail}, 0)) & \wedge \\
& r = \mathit{p2s}(\mathit{getp}(\mathit{heap}, sl.\mathit{head}, sl.\mathit{tail}, 0)) & \wedge \\
& \mathit{rd}(\mathit{heap}, sl.\mathit{tail}).\mathit{arr}[0] = \mathit{null} \wedge \cdots \wedge \mathit{rd}(\mathit{heap}, sl.\mathit{tail}).\mathit{arr}[\mathit{max}_H]) = \mathit{null} & \wedge \\
& a \in r \rightarrow \mathit{rd}(\mathit{heap}, a).\mathit{level} \leq \mathit{max}_H & \wedge \\
\bigwedge_{i \in 0 \ldots (\mathit{max}_H - 1)} & \mathit{p2s}(\mathit{getp}(\mathit{heap}, \mathit{head}, \mathit{tail}, i+1)) \subseteq \mathit{p2s}(\mathit{getp}(\mathit{heap}, \mathit{head}, \mathit{tail}, i)) &
\end{array}
$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux} \wedge \rho_{36}(V, V') \rightarrow SL(h', sl', r')$

$$
\mathit{SkipList}(h, sl, r) \wedge
$$

$$
\left(\begin{array}{rcll}
x.\mathit{key} & = & 14 & \wedge \\
\mathit{prev}.\mathit{key} & < & 14 & \wedge \\
x.\mathit{arr}[0].\mathit{key} & > & 14 & \wedge \\
\mathit{prev}.\mathit{arr}[0] & = & x.\mathit{arr}[0] & \wedge \\
\multicolumn{3}{c}{x \notin r \wedge 0 \leq 0 \leq 3} &
\end{array}\right) \wedge
\left(\begin{array}{cl}
\mathit{at}_{36} & \wedge \\
\mathit{prev}'.\mathit{arr}[0] = x & \wedge \\
\mathit{at}'_{37} & \wedge \\
h' = h \wedge sl = sl' & \wedge \\
r' = r \cup \{x\} \wedge x' = x & \ldots
\end{array}\right) \rightarrow \mathit{SkipList}(h', sl', r')
$$

reason about

Verification of Skiplists

- **Skiplist** shape **preservation** : $\Box SkipList(h, sl, r)$

$$
\begin{array}{rll}
SkipList(h, sl, r) \doteq & & \\
& ordList(m, getp(heap, sl.head, sl.tail, 0)) & \wedge \\
& r = p2s(getp(heap, sl.head, sl.tail, 0)) & \wedge \\
& rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null & \wedge \\
& a \in r \rightarrow rd(heap, a).level \leq max_H & \wedge \\
\bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i+1)) \subseteq p2s(getp(heap, head, tail, i)) &
\end{array}
$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux} \wedge \rho_{36}(V, V') \rightarrow SL(h', sl', r')$

$$
SkipList(h, sl, r) \wedge
$$

$$
\left(\begin{array}{rclc}
x.key & = & 14 & \wedge \\
prev.key & < & 14 & \wedge \\
x.arr[0].key & > & 14 & \wedge \\
prev.arr[0] & = & x.arr[0] & \wedge \\
\multicolumn{3}{c}{x \notin r \wedge 0 \leq 0 \leq 3} &
\end{array}\right)
\wedge
\left(\begin{array}{cc}
at_{36} & \wedge \\
prev'.arr[0] = x & \wedge \\
at'_{37} & \wedge \\
h' = h \wedge sl = sl' & \wedge \\
r' = r \cup \{x\} \wedge x' = x & \ldots
\end{array}\right)
\rightarrow SkipList(h', sl', r')
$$

reason about

ordered values + notion of ordered list

Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$
\begin{array}{lll}
SkipList(h, sl, r) \hat{=} & & \\
 & ordList(m, getp(heap, sl.head, sl.tail, 0)) & \wedge \\
 & r = p2s(getp(heap, sl.head, sl.tail, 0)) & \wedge \\
 & rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H] = null & \wedge \\
 & a \in r \to rd(heap, a).level \leq max_H & \wedge \\
\bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i + 1)) \subseteq p2s(getp(heap, head, tail, i)) &
\end{array}
$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux} \wedge \rho_{36}(V, V') \to SL(h', sl', r')$

$$
SkipList(h, sl, r) \wedge
$$

$$
\begin{pmatrix}
x.key & = & 14 & \wedge \\
prev.key & < & 14 & \wedge \\
x.arr[0].key & > & 14 & \wedge \\
prev.arr[0] & = & x.arr[0] & \wedge \\
x \notin r \wedge 0 \leq 0 \leq 3 & & &
\end{pmatrix}
\wedge
\begin{pmatrix}
at_{36} & \wedge \\
prev'.arr[0] = x & \wedge \\
at'_{37} & \wedge \\
h' = h \wedge sl = sl' & \wedge \\
r' = r \cup \{x\} \wedge x' = x & \ldots
\end{pmatrix}
\to SkipList(h', sl', r')
$$

reason about

levels

Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$
\begin{aligned}
SkipList(h, sl, r) \triangleq & \\
& ordList(m, getp(heap, sl.head, sl.tail, 0)) & \wedge \\
& r = p2s(getp(heap, sl.head, sl.tail, 0)) & \wedge \\
& rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null & \wedge \\
& a \in r \rightarrow rd(heap, a).level \leq max_H & \wedge \\
\bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i+1)) \subseteq p2s(getp(heap, head, tail, i))
\end{aligned}
$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux} \wedge \rho_{36}(V, V') \rightarrow SL(h', sl', r')$

$$
SkipList(h, sl, r) \wedge
$$

$$
\begin{pmatrix}
x.key & = & 14 & \wedge \\
prev.key & < & 14 & \wedge \\
x.arr[0].key & > & 14 & \wedge \\
prev.arr[0] & = & x.arr[0] & \wedge \\
x \notin r \wedge 0 \leq 0 \leq 3 & & &
\end{pmatrix}
\wedge
\begin{pmatrix}
at_{36} & \wedge \\
prev'.arr[0] = x & \wedge \\
at'_{37} & \wedge \\
h' = h \wedge sl = sl' & \wedge \\
r' = r \cup \{x\} \wedge x' = x & \ldots
\end{pmatrix}
\rightarrow SkipList(h', sl', r')
$$

reason about

arrays

Verification of Skiplists

- **Skiplist** shape **preservation** : $\Box SkipList(h, sl, r)$

$$
SkipList(h, sl, r) \mathrel{\hat{=}}
\begin{array}{ll}
 & ordList(m, getp(heap, sl.head, sl.tail, 0)) \quad \wedge \\
 & r = p2s(getp(heap, sl.head, sl.tail, 0)) \quad \wedge \\
 & rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null \quad \wedge \\
 & a \in r \to rd(heap, a).level \leq max_H \quad \wedge \\
\bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i+1)) \subseteq p2s(getp(heap, head, tail, i))
\end{array}
$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux} \wedge \rho_{36}(V, V') \to SL(h', sl', r')$

$$
SkipList(h, sl, r) \wedge
\left(
\begin{array}{rcll}
x.key & = & 14 & \wedge \\
prev.key & < & 14 & \wedge \\
x.arr[0].key & > & 14 & \wedge \\
prev.arr[0] & = & x.arr[0] & \wedge \\
\multicolumn{4}{c}{x \notin r \wedge 0 \leq 0 \leq 3}
\end{array}
\right)
\wedge
\left(
\begin{array}{cl}
at_{36} & \wedge \\
prev'.arr[0] = x & \wedge \\
at'_{37} & \wedge \\
h' = h \wedge sl = sl' & \wedge \\
r' = r \cup \{x\} \wedge x' = x & \ldots
\end{array}
\right)
\to SkipList(h', sl', r')
$$

reason about

regions (sets)

Verification of Skiplists

- **Skiplist** shape **preservation** : $\square SkipList(h, sl, r)$

$$SkipList(h, sl, r) \hat{=}$$

$$\begin{array}{lr} & ordList(m, getp(heap, sl.head, sl.tail, 0)) \quad \wedge \\ & r = p2s(getp(heap, sl.head, sl.tail, 0)) \quad \wedge \\ & rd(heap, sl.tail).arr[0] = null \wedge \cdots \wedge rd(heap, sl.tail).arr[max_H]) = null \quad \wedge \\ & a \in r \to rd(heap, a).level \leq max_H \quad \wedge \\ \bigwedge_{i \in 0 \ldots (max_H - 1)} & p2s(getp(heap, head, tail, i + 1)) \subseteq p2s(getp(heap, head, tail, i)) \end{array}$$

- Program **transitions** : $SL(h, sl, r) \wedge \varphi_{aux} \wedge \rho_{36}(V, V') \to SL(h', sl', r')$

$$SkipList(h, sl, r) \wedge$$

$$\begin{pmatrix} x.key & = & 14 & \wedge \\ prev.key & < & 14 & \wedge \\ x.arr[0].key & > & 14 & \wedge \\ prev.arr[0] & = & x.arr[0] & \wedge \\ & x \notin r \wedge 0 \leq 0 \leq 3 & & \end{pmatrix} \wedge \begin{pmatrix} at_{36} & \wedge \\ prev'.arr[0] = x & \wedge \\ at'_{37} & \wedge \\ h' = h \wedge sl = sl' & \wedge \\ r' = r \cup \{x\} \wedge x' = x & \ldots \end{pmatrix} \to SkipList(h', sl', r')$$

reason about

memory, cells

Our Contribution

- **TSL**, a theory for skiplists of **arbitrary length and height**
- We show TSL **decidable**...
- ...by reducing **TSL satisfiability** to **$\mathbf{TSL}_K$ satisfiability**.

TSL: A Theory for Skiplists of Arbitrary Height

TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

Σ_{addr}

TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}}$

TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}}$$

TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}}$$

TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}}$$

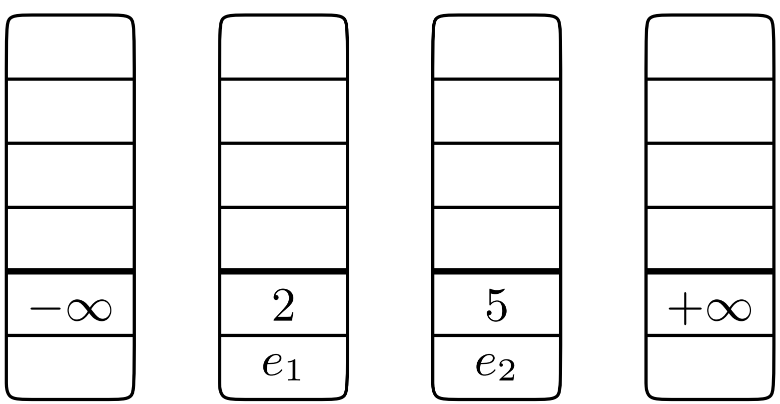
$\mathbf{TSL}_\mathbf{K}$	**TSL**

TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$$\Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}}$$

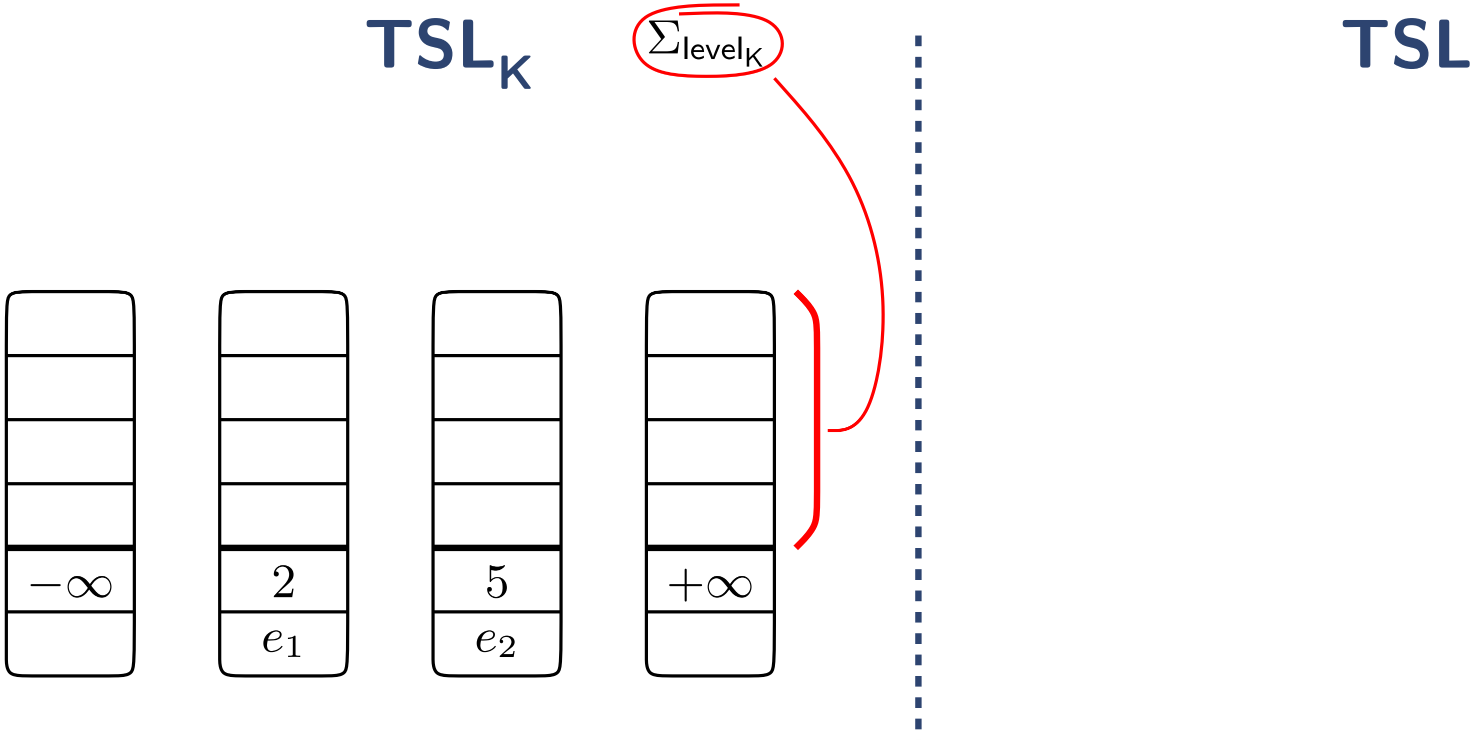
TSL_K | **TSL**



TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

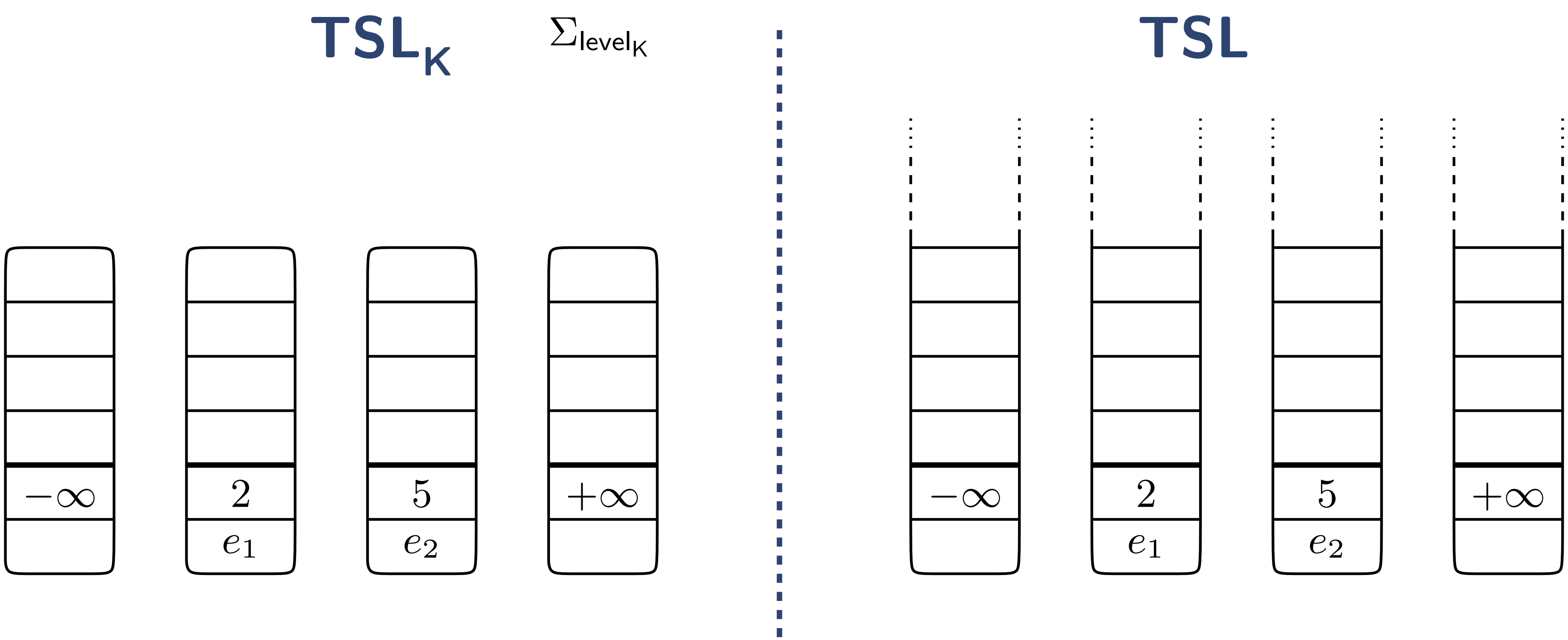
$$\Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}}$$



TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

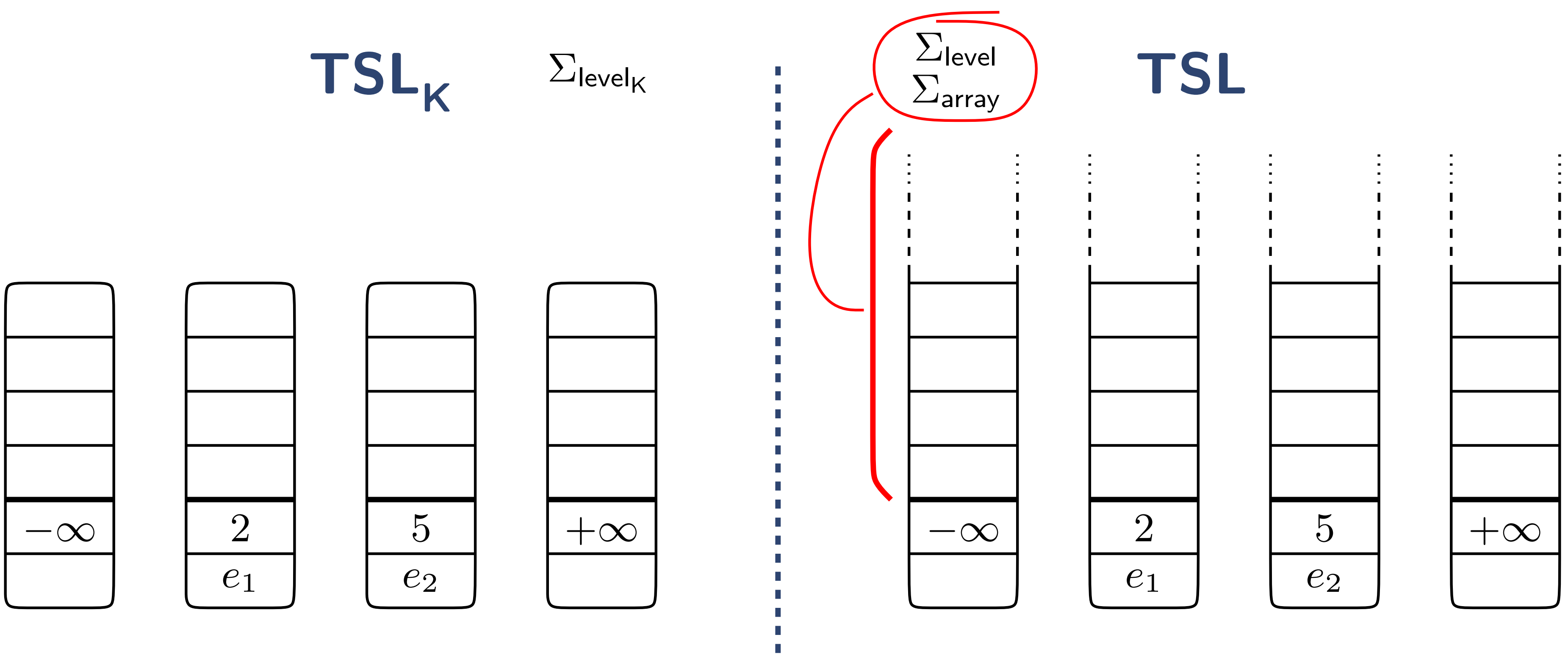
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}}$$



TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

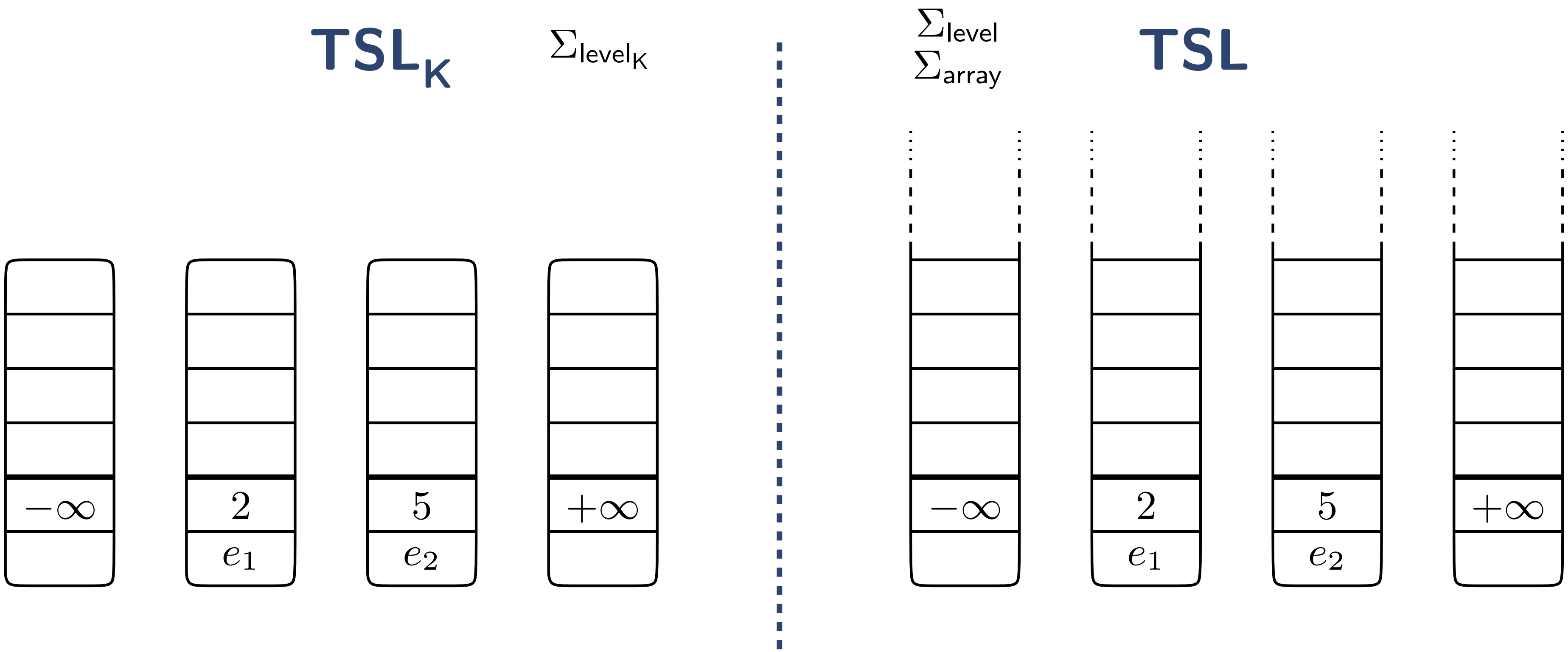
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}}$$



TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like $\mathsf{TSL_K}$, is a **union of other theories**

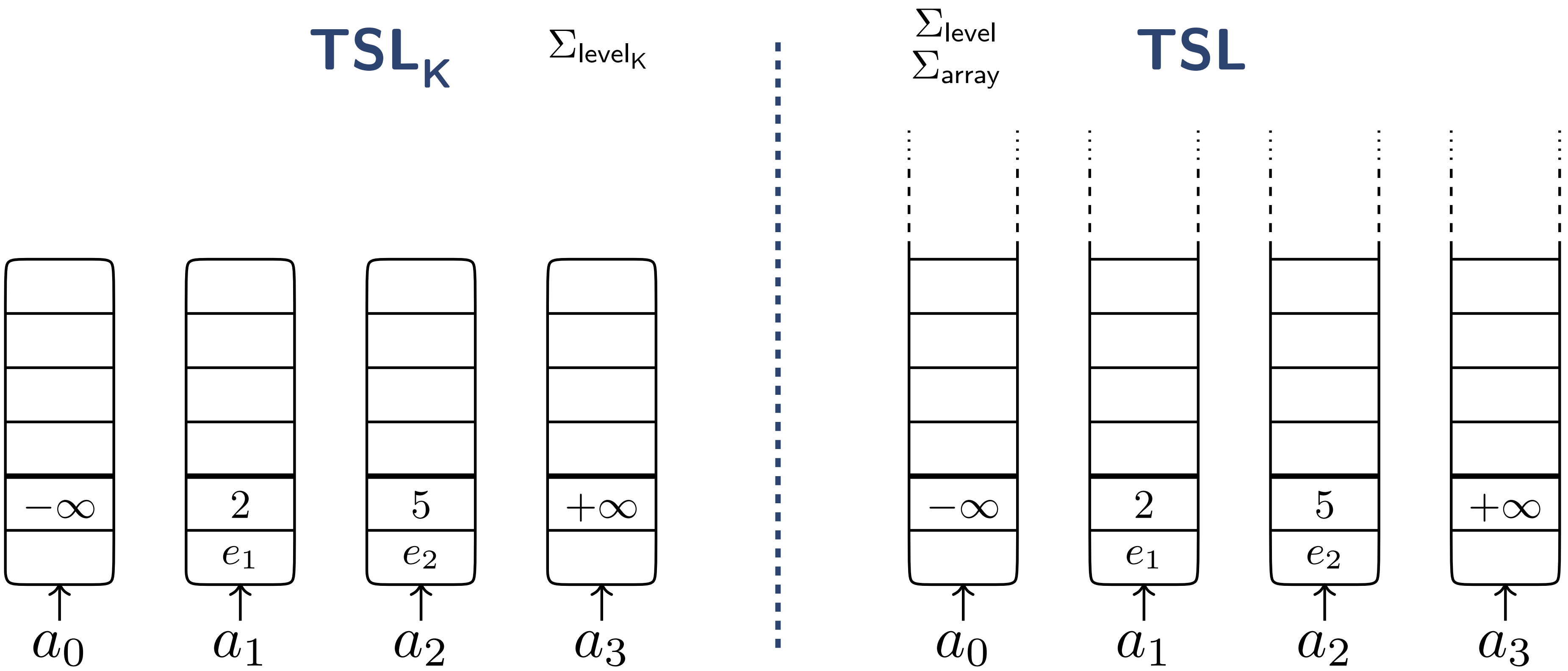
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}}$$



TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

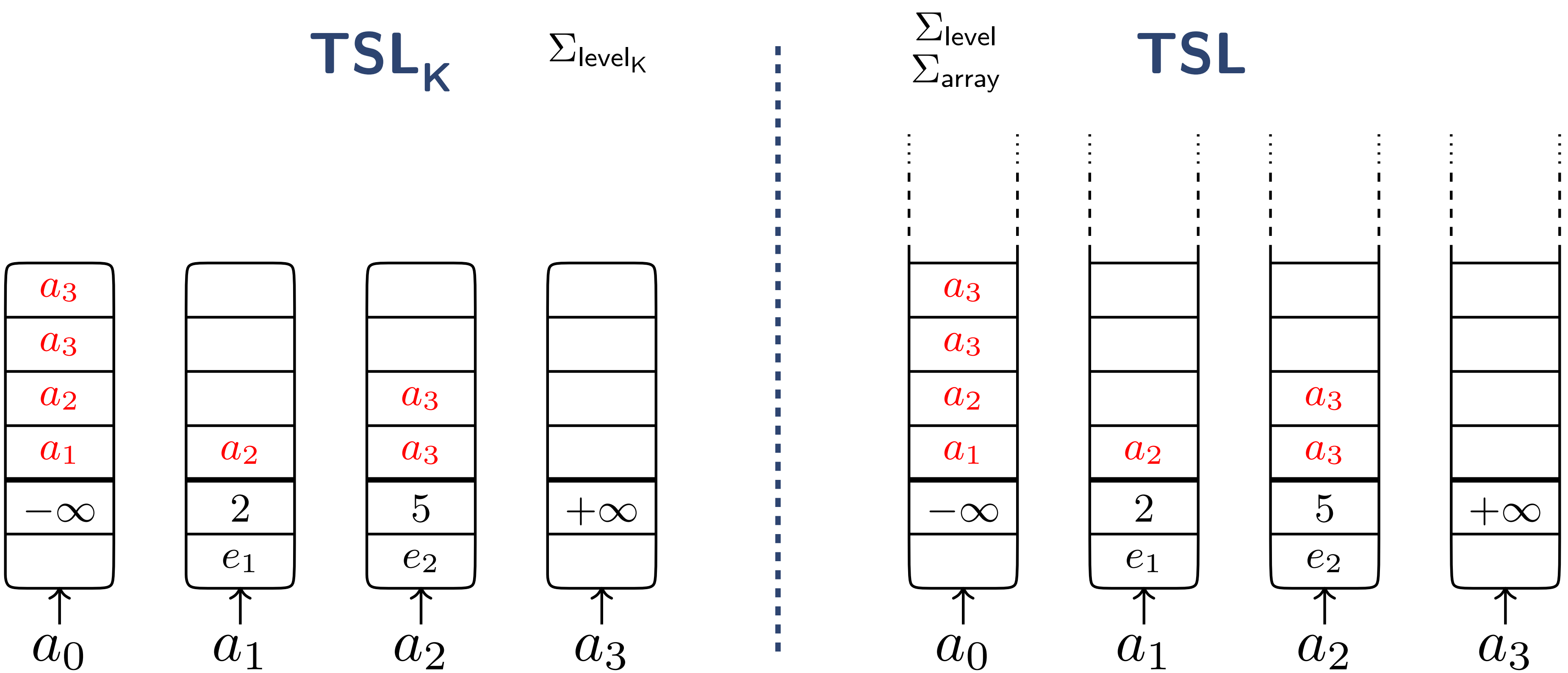
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}}$$



TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

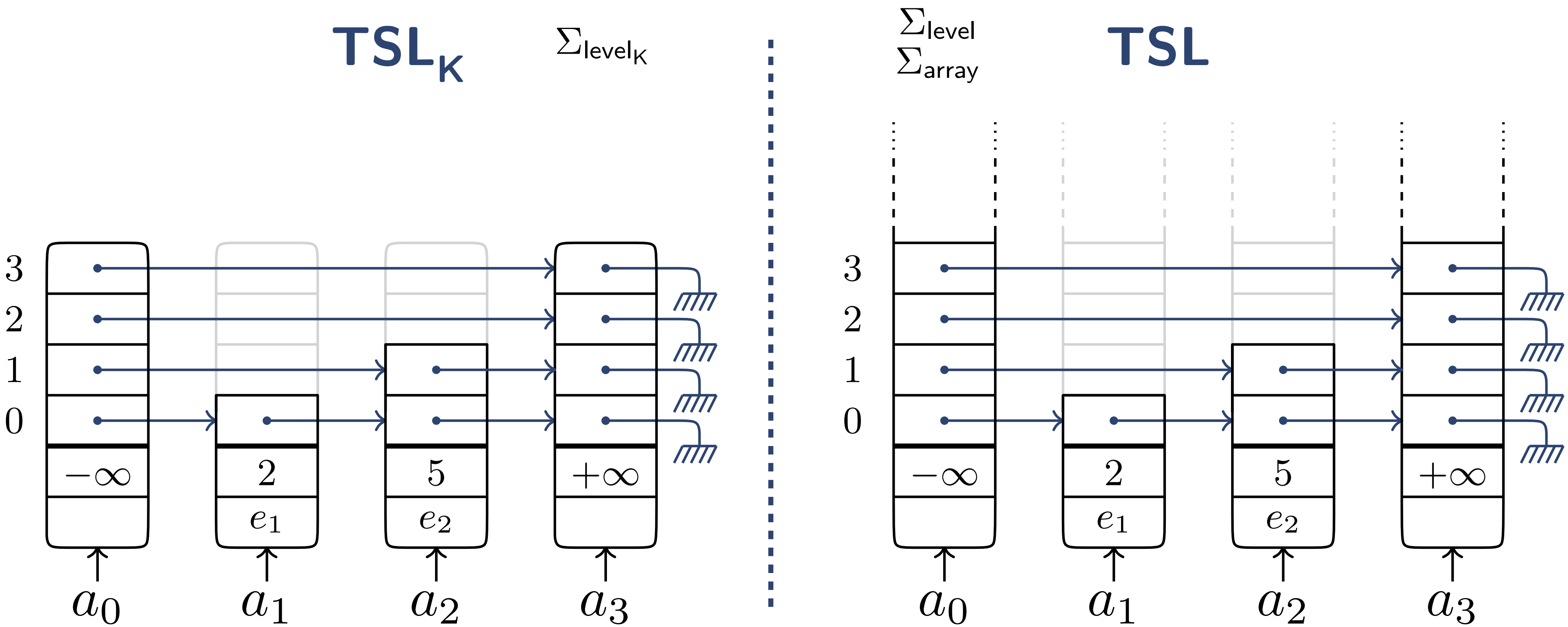
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}}$$



TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

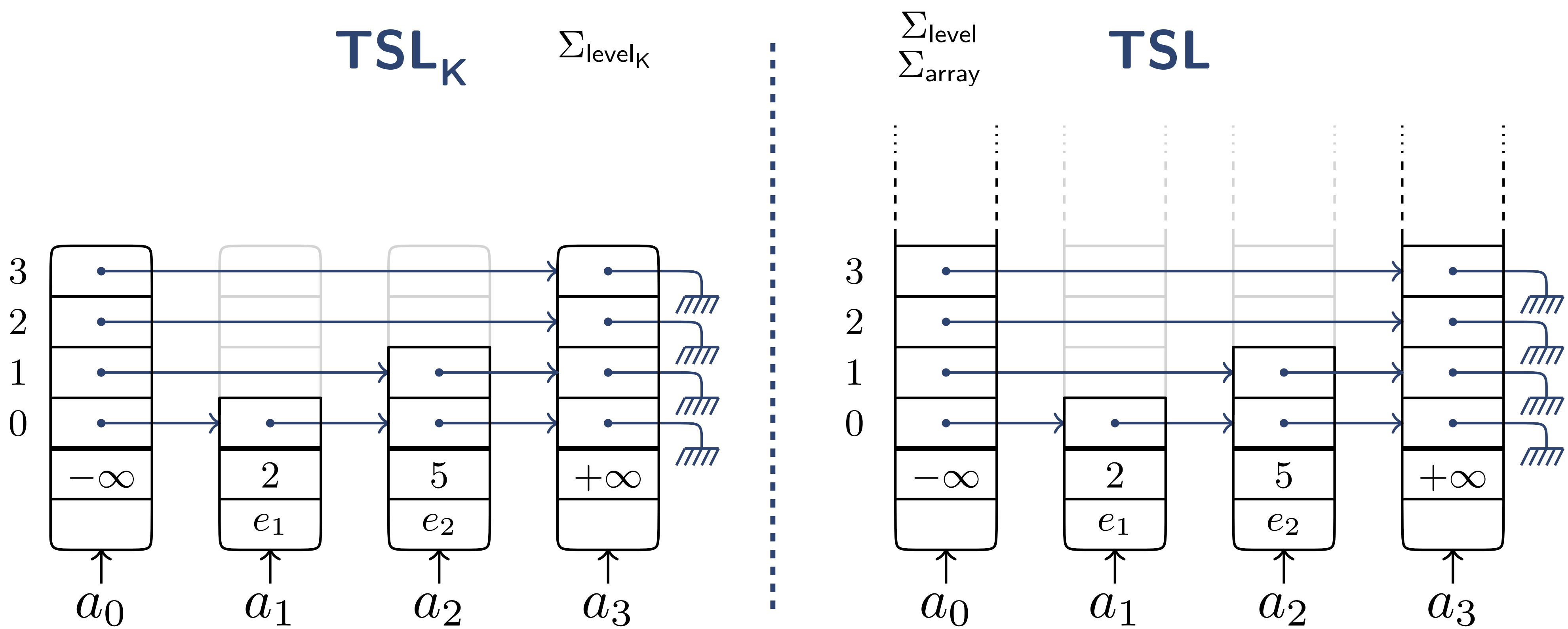
$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}}$$



TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}}$$



TSL: A Theory for Skiplists of Arbitrary Height

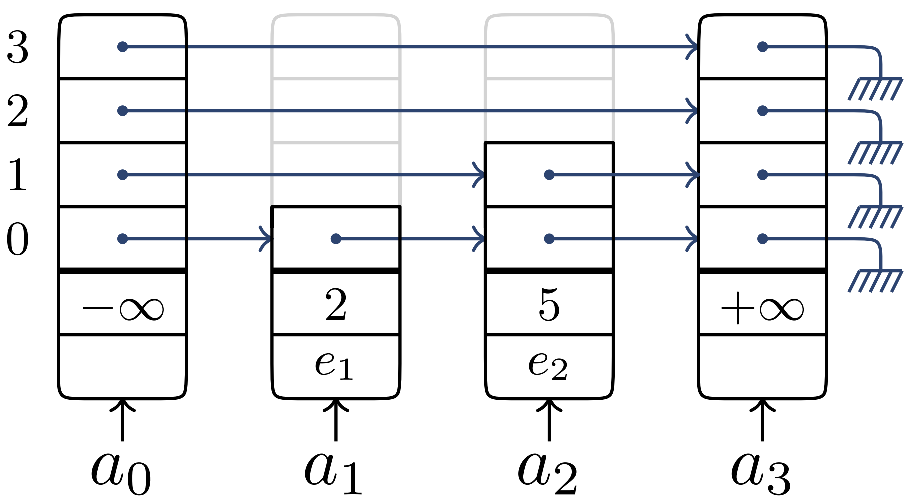
- TSL, like $\mathsf{TSL_K}$, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{reachability}}$$

path = a non-repeating sequence of addresses

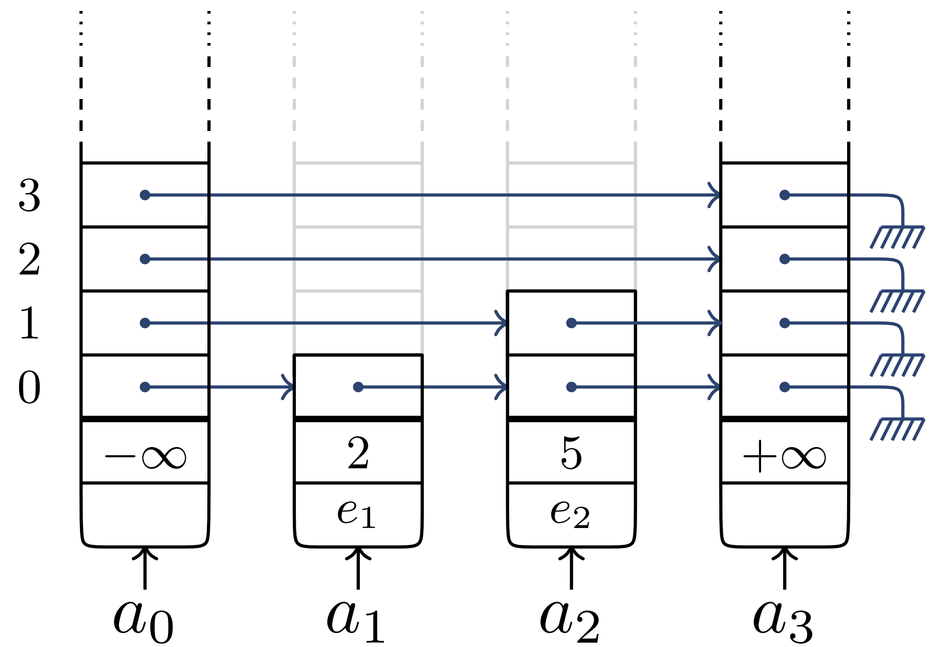
$$[a_1, a_2, a_3]$$

$\mathsf{TSL_K}$ $\Sigma_{\mathsf{level_K}}$



Σ_{level}
Σ_{array}

TSL

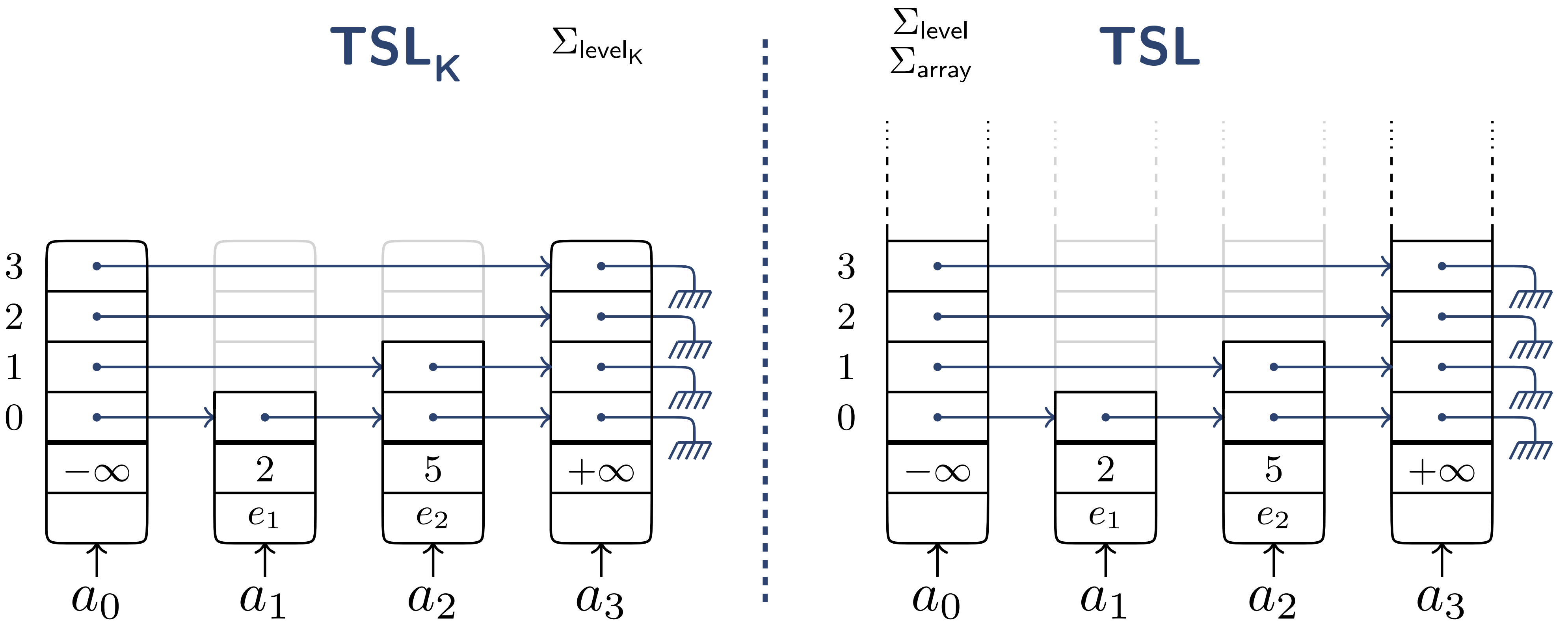


TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like $\mathsf{TSL_K}$, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{reachability}}$$

$$append([a_1, a_2], [a_3], [a_1, a_2, a_3])$$

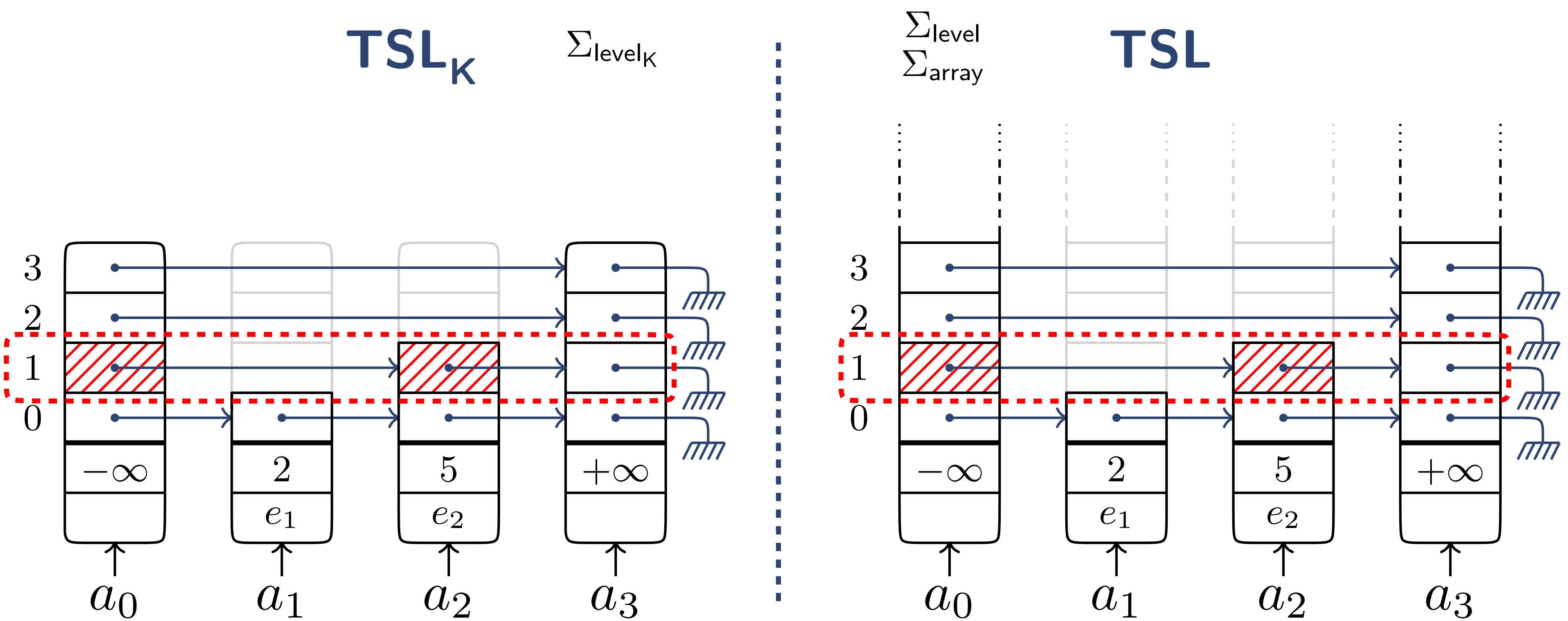


TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{reachability}}$$

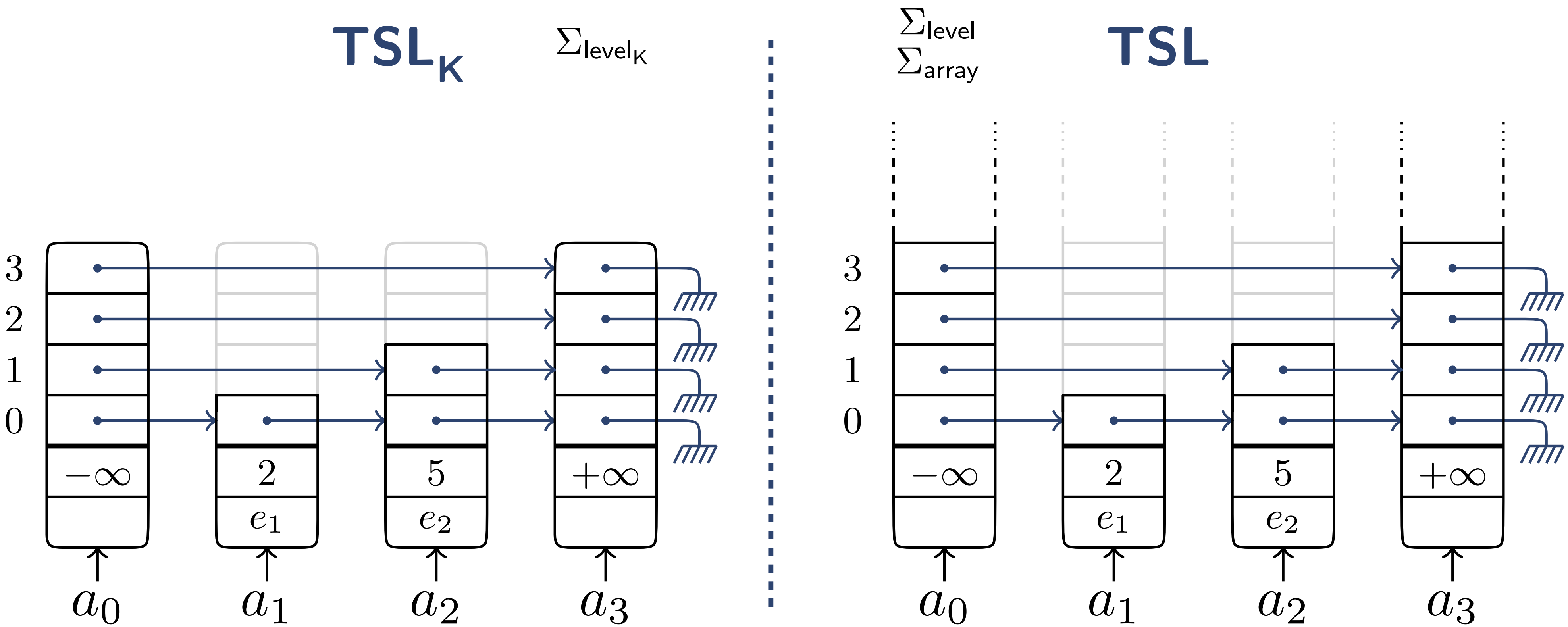
$$reach(a_0, a_3, 1, [a_0, a_2])$$



TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{reachability}} \cup \Sigma_{\mathsf{bridge}}$$

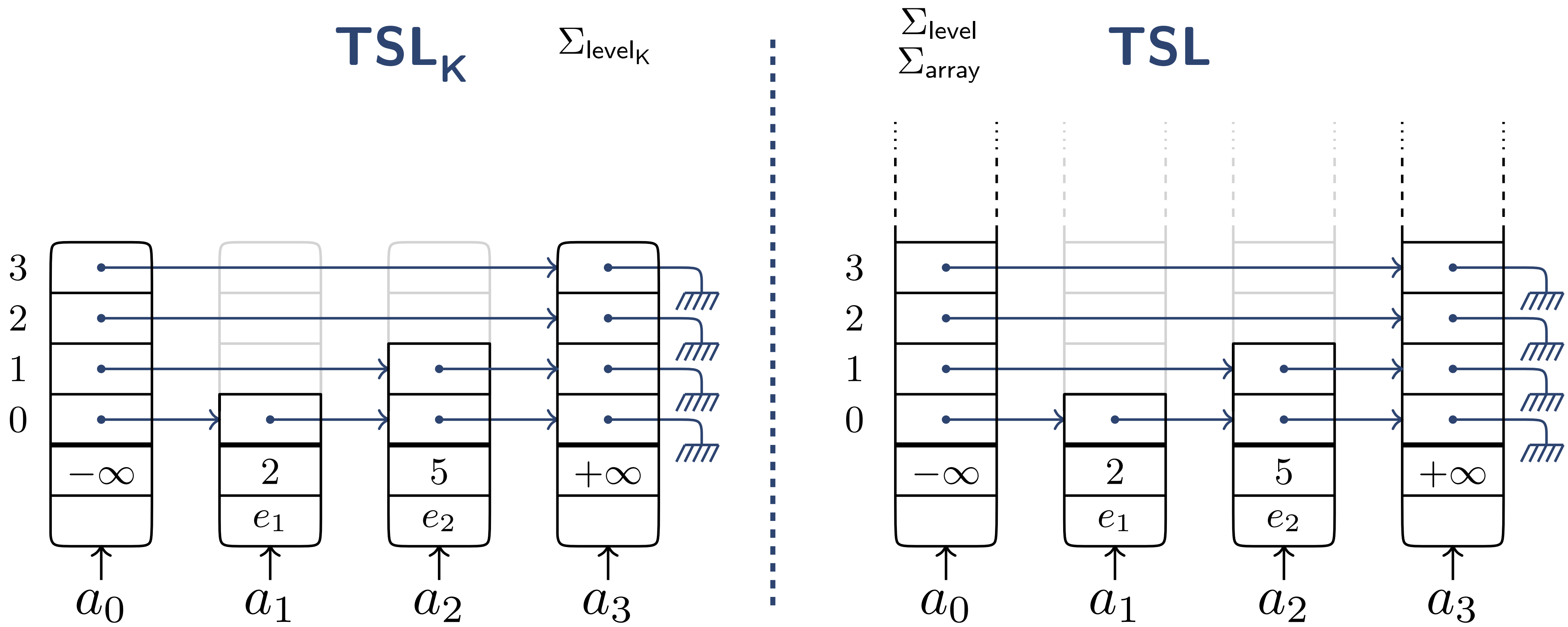


TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like $\mathsf{TSL_K}$, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{reachability}} \cup \Sigma_{\mathsf{bridge}}$$

$$path2set([a_2, a_3]) = \{a_2, a_3\}$$

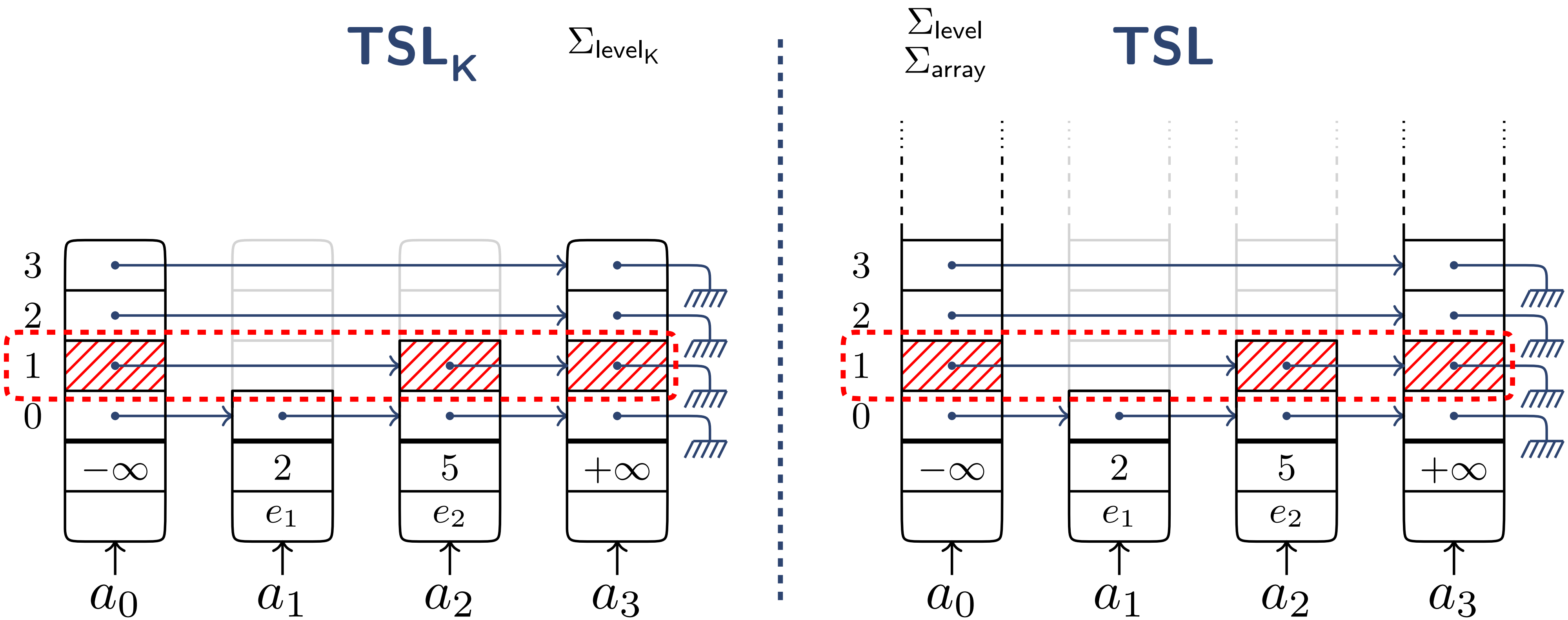


TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{reachability}} \cup \Sigma_{\mathsf{bridge}}$$

$$addr2set(a_0, 1) = \{a_0, a_2, a_3\}$$

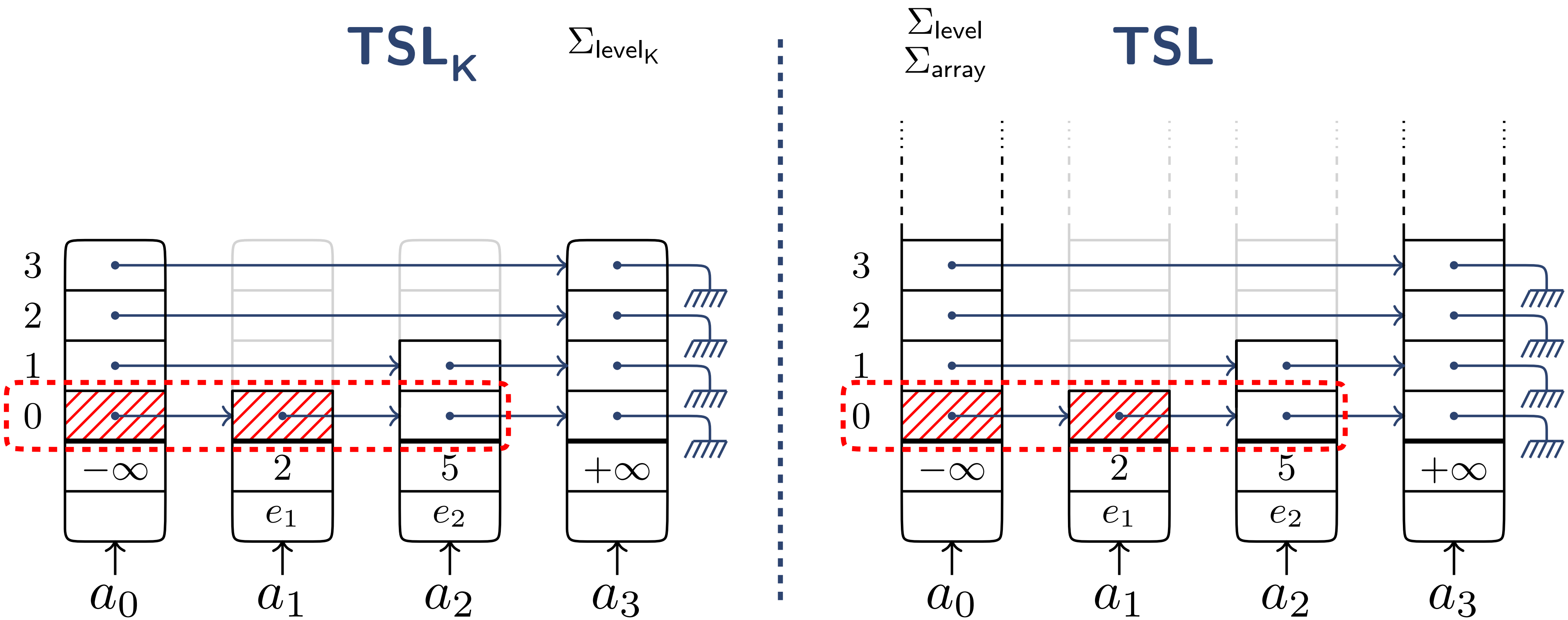


TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like $\mathsf{TSL_K}$, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{reachability}} \cup \Sigma_{\mathsf{bridge}}$$

$$getp(a_0, a_2, 0) = [a_0, a_1]$$

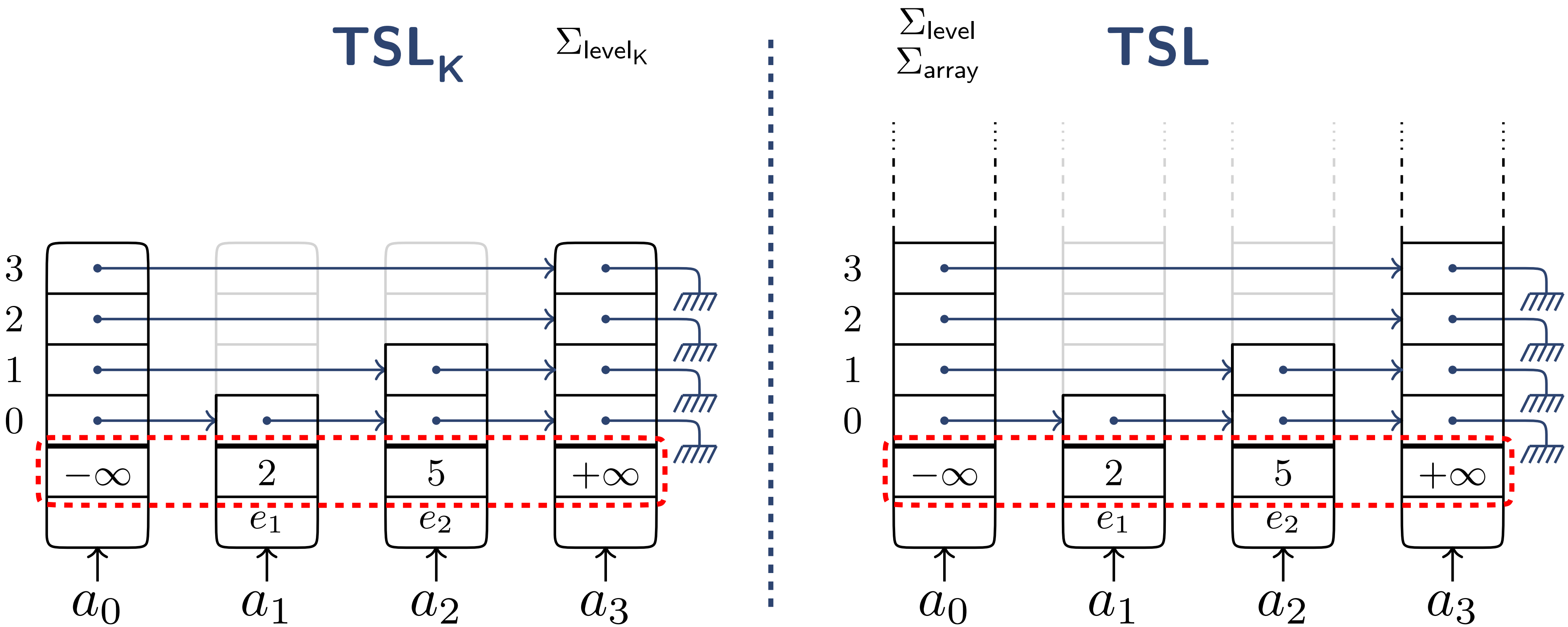


TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$$\Sigma_{\text{addr}} \cup \Sigma_{\text{elem}} \cup \Sigma_{\text{ord}} \cup \Sigma_{\text{cell}} \cup \Sigma_{\text{mem}} \cup \Sigma_{\text{set}} \cup \Sigma_{\text{reachability}} \cup \Sigma_{\text{bridge}}$$

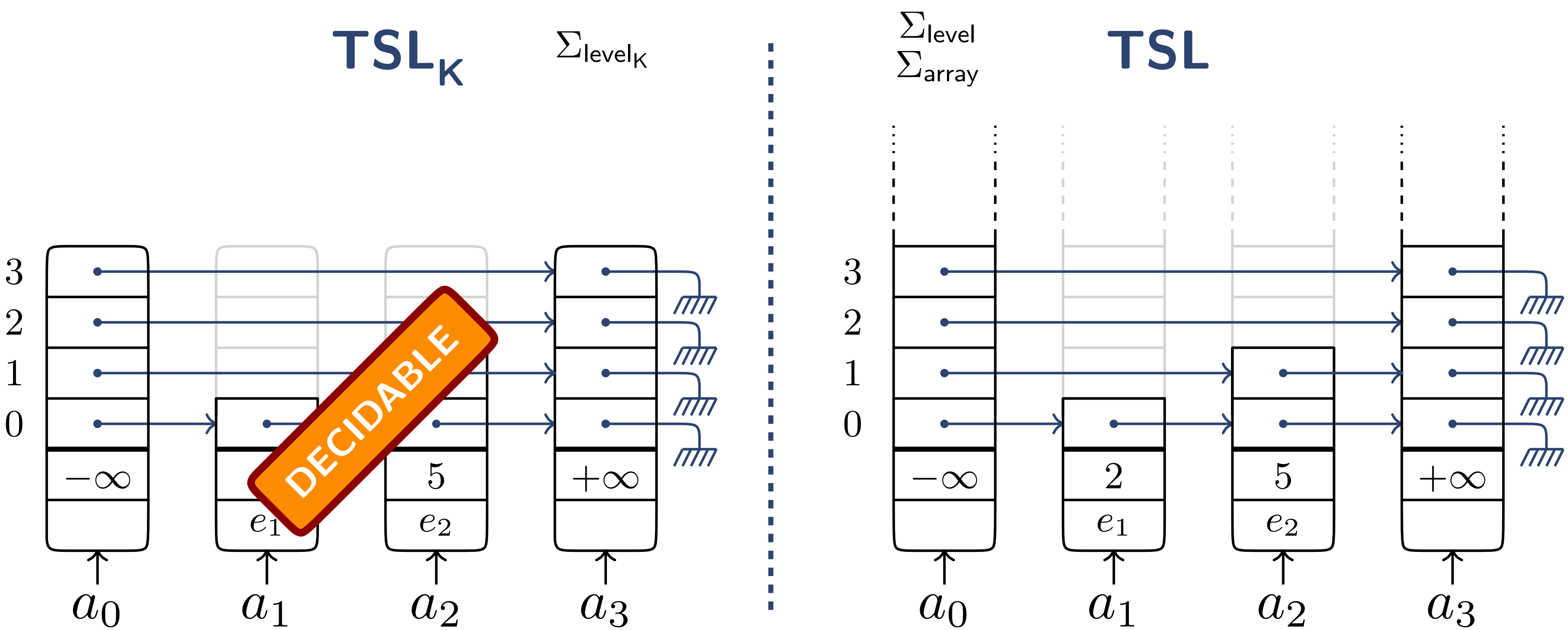
$$ordList([a_0, a_1, a_2, a_3])$$



TSL: A Theory for Skiplists of Arbitrary Height

- TSL, like TSL_K, is a **union of other theories**

$$\Sigma_{\mathsf{addr}} \cup \Sigma_{\mathsf{elem}} \cup \Sigma_{\mathsf{ord}} \cup \Sigma_{\mathsf{cell}} \cup \Sigma_{\mathsf{mem}} \cup \Sigma_{\mathsf{set}} \cup \Sigma_{\mathsf{reachability}} \cup \Sigma_{\mathsf{bridge}}$$



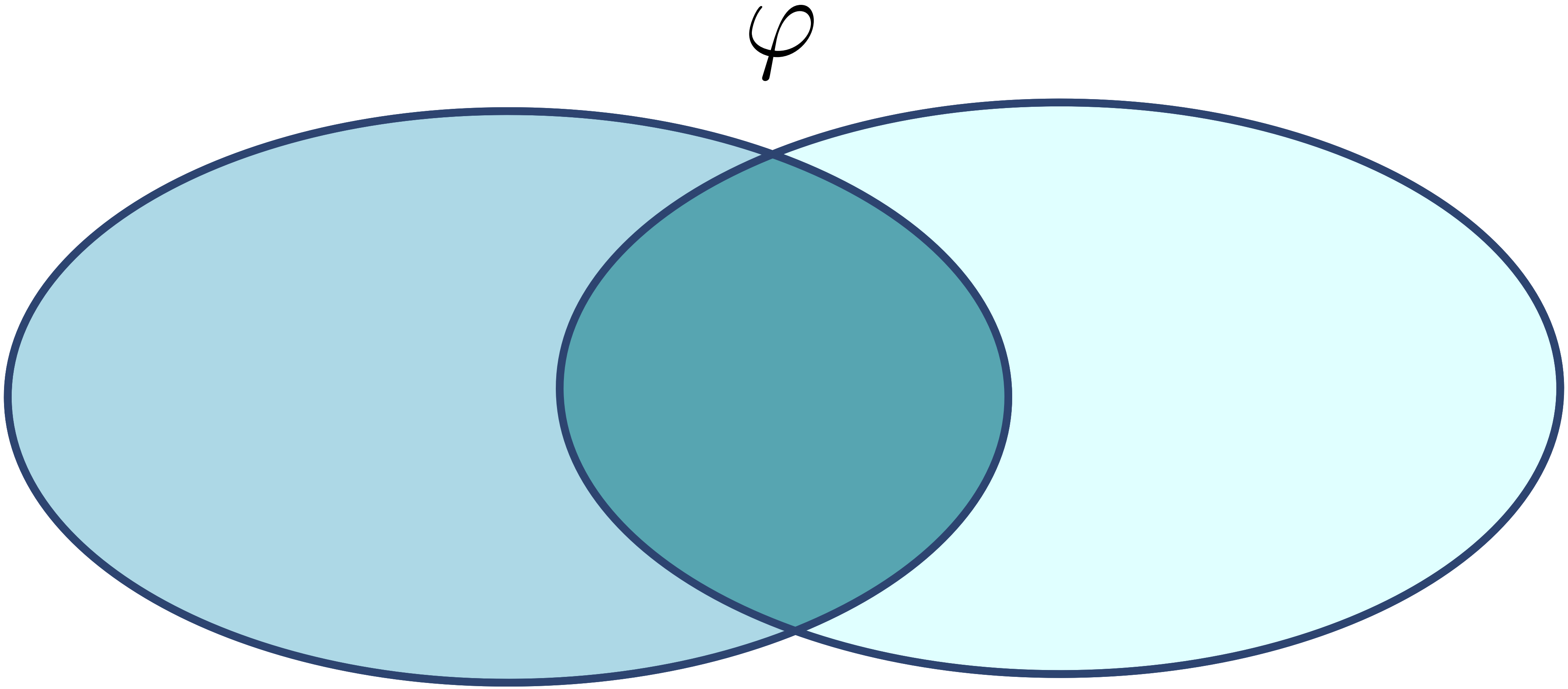
Decision Procedure for TSL

Decision Procedure for TSL

- Let φ be a normalized TSL formula

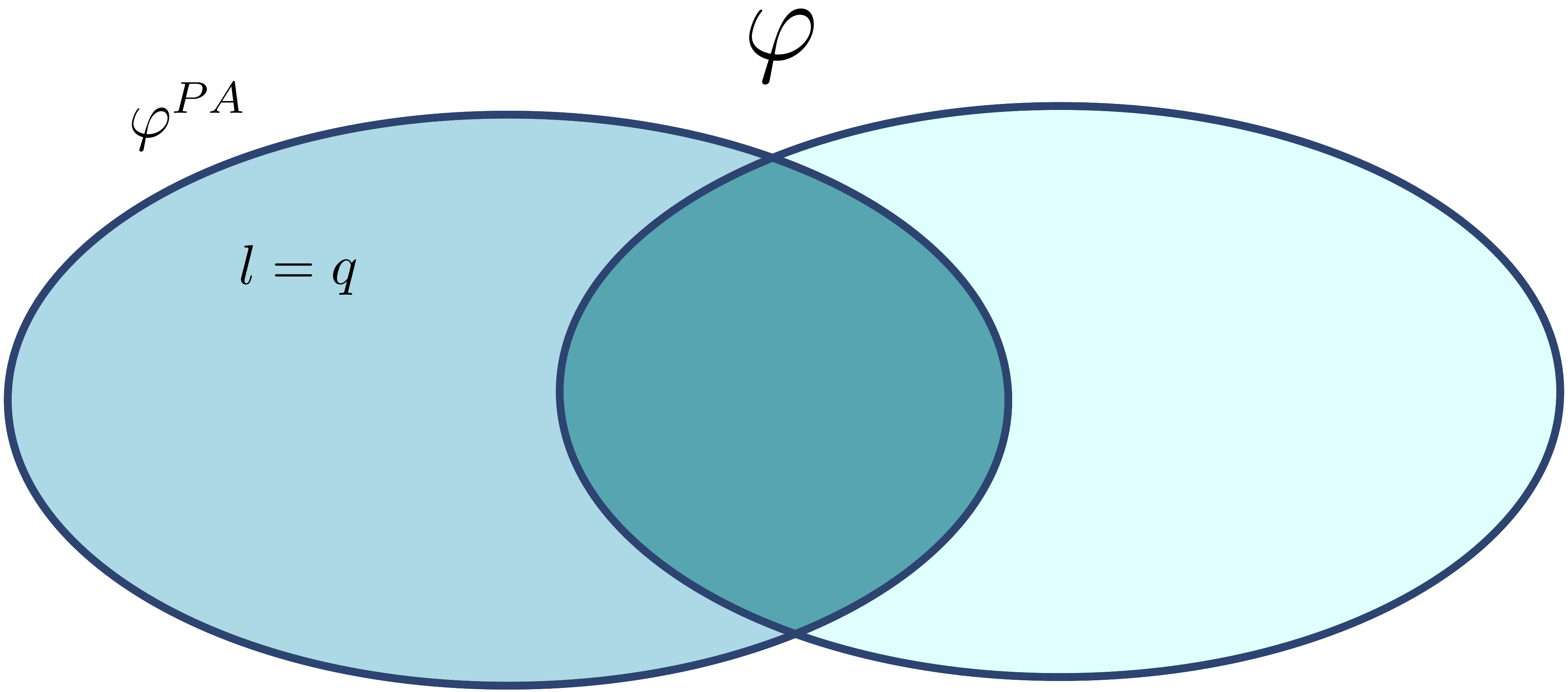
Decision Procedure for TSL

- Let φ be a normalized TSL formula



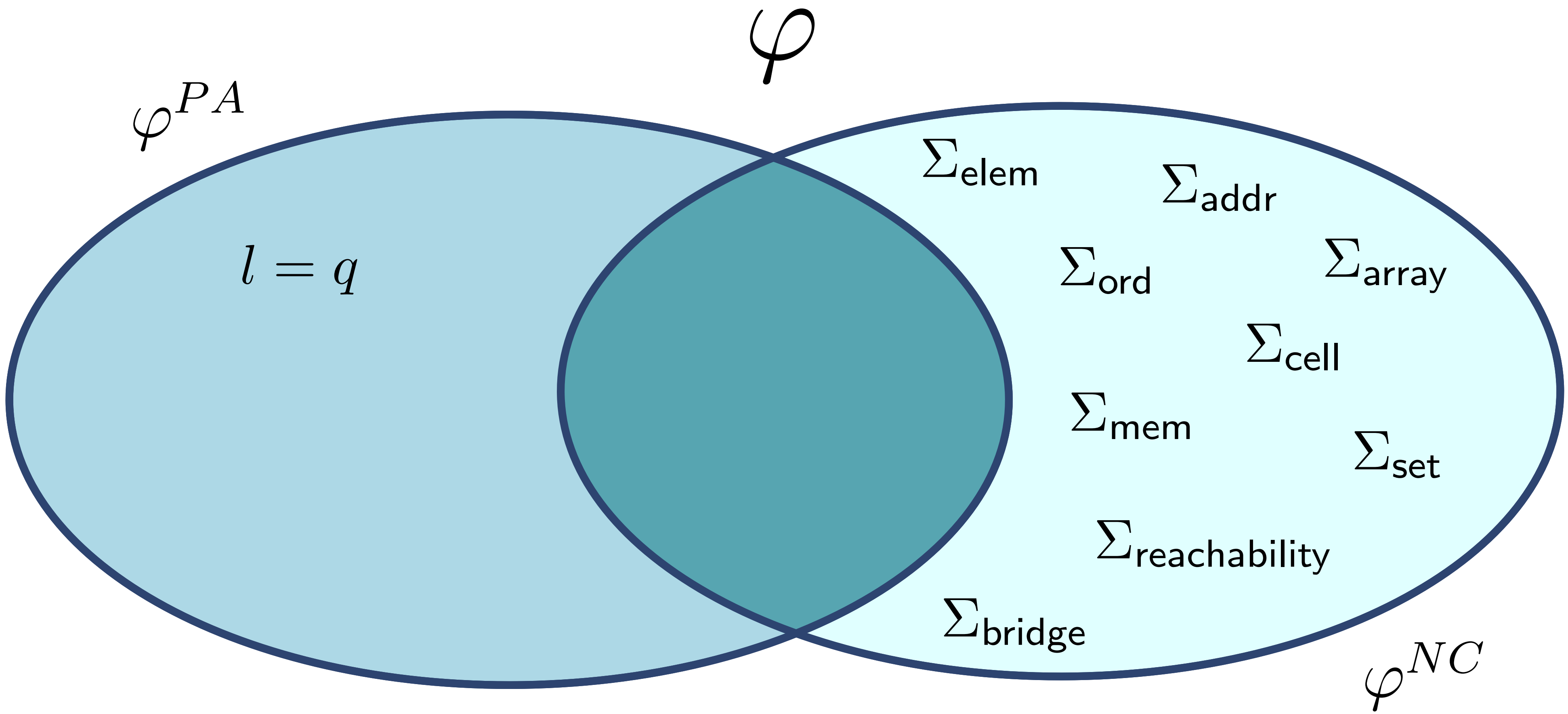
Decision Procedure for TSL

- Let φ be a normalized TSL formula



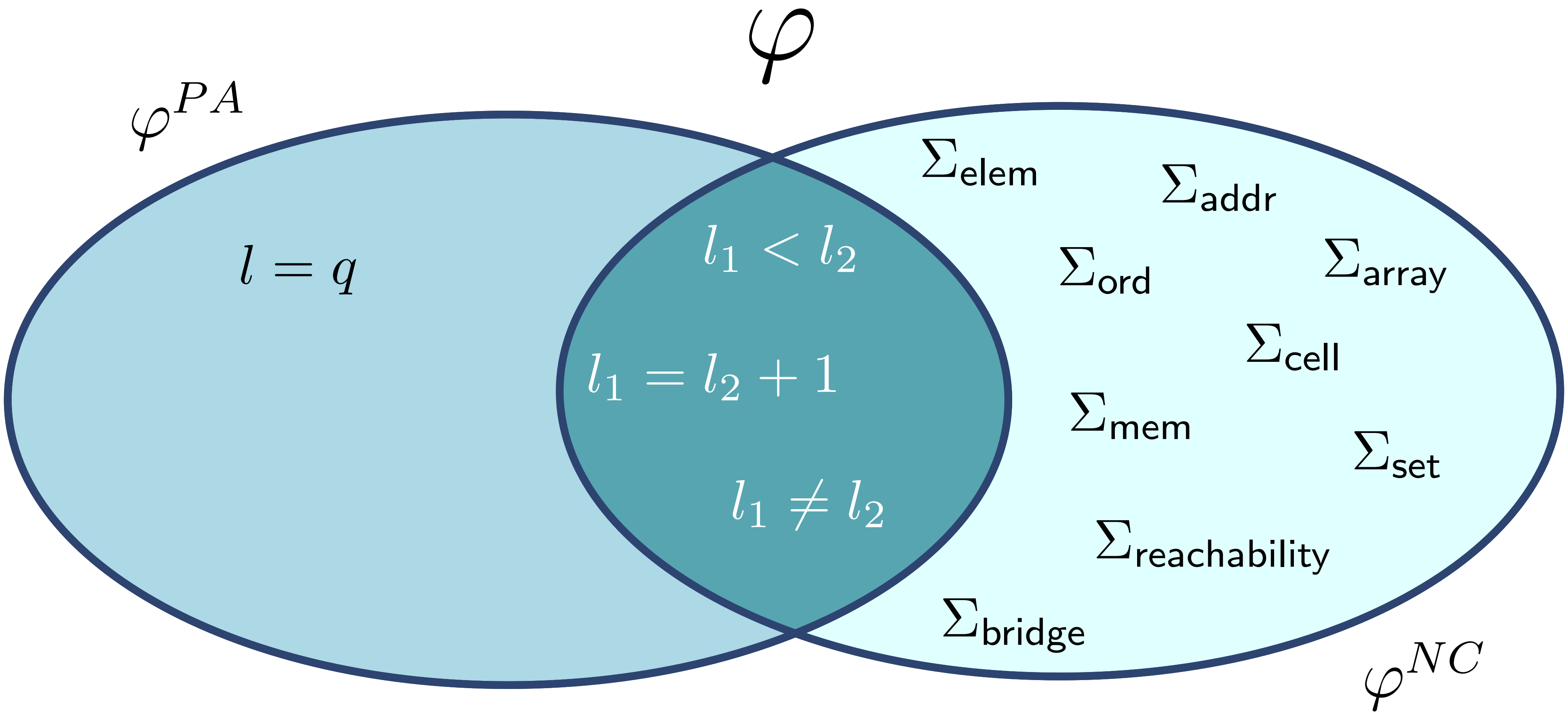
Decision Procedure for TSL

- Let φ be a normalized TSL formula



Decision Procedure for TSL

- Let φ be a normalized TSL formula



Decision Procedure for TSL

- Let φ be a normalized TSL formula

$$\varphi$$

Decision Procedure for TSL

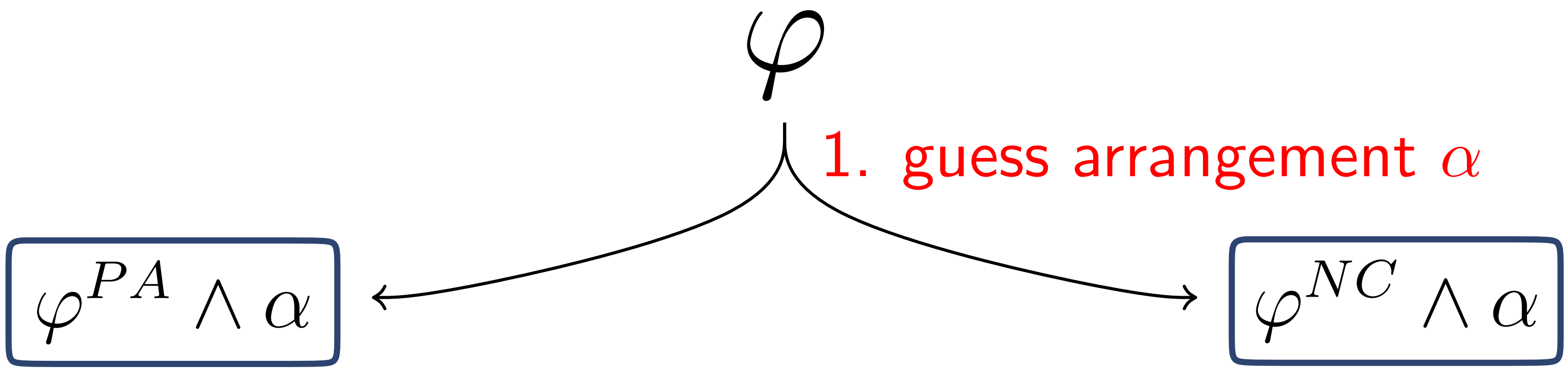
- Let φ be a normalized TSL formula

$$\varphi$$

1. guess arrangement α

Decision Procedure for TSL

- Let φ be a normalized TSL formula



Decision Procedure for TSL

- Let φ be a normalized TSL formula

$$\varphi$$

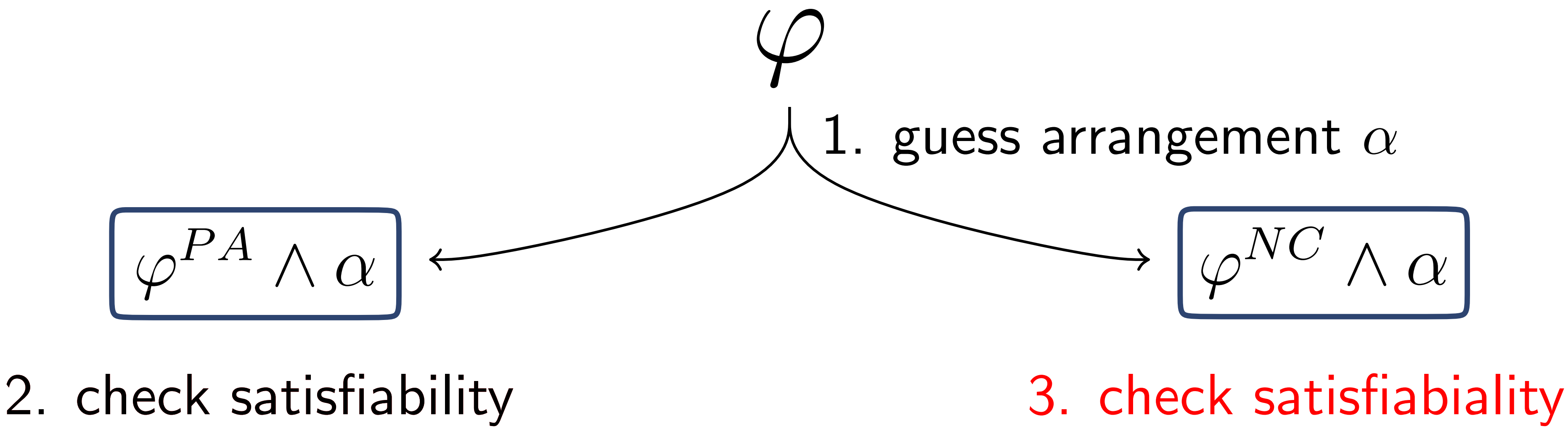
1. guess arrangement α

$$\varphi^{PA} \wedge \alpha \qquad \varphi^{NC} \wedge \alpha$$

2. check satisfiability

Decision Procedure for TSL

- Let φ be a normalized TSL formula



Decision Procedure for TSL: Correctness

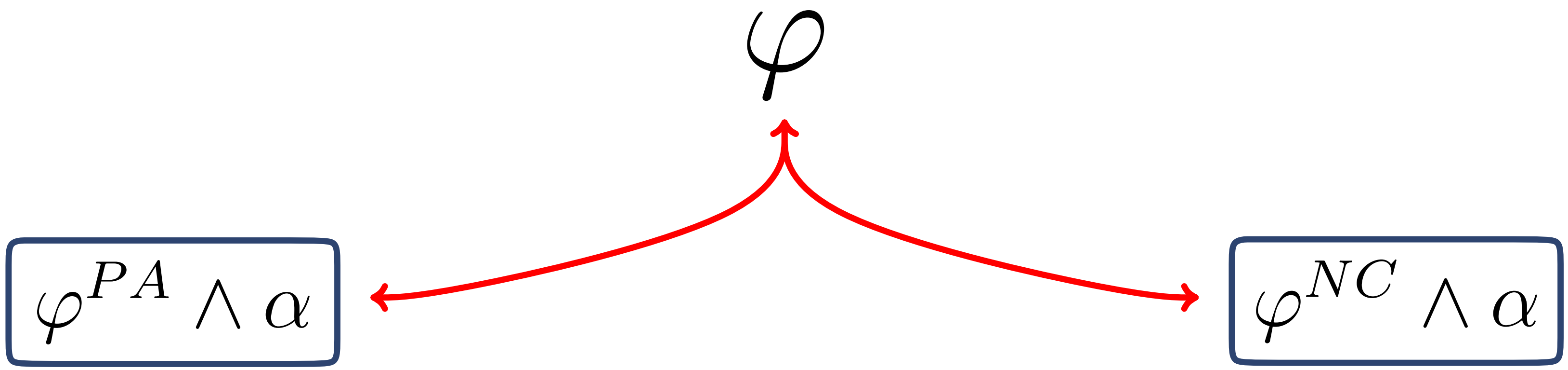
- Let φ be a normalized TSL formula

$$\varphi$$

$$\varphi^{PA} \land \alpha \quad \longleftarrow \quad \varphi \quad \longrightarrow \quad \varphi^{NC} \land \alpha$$

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



Theorem:

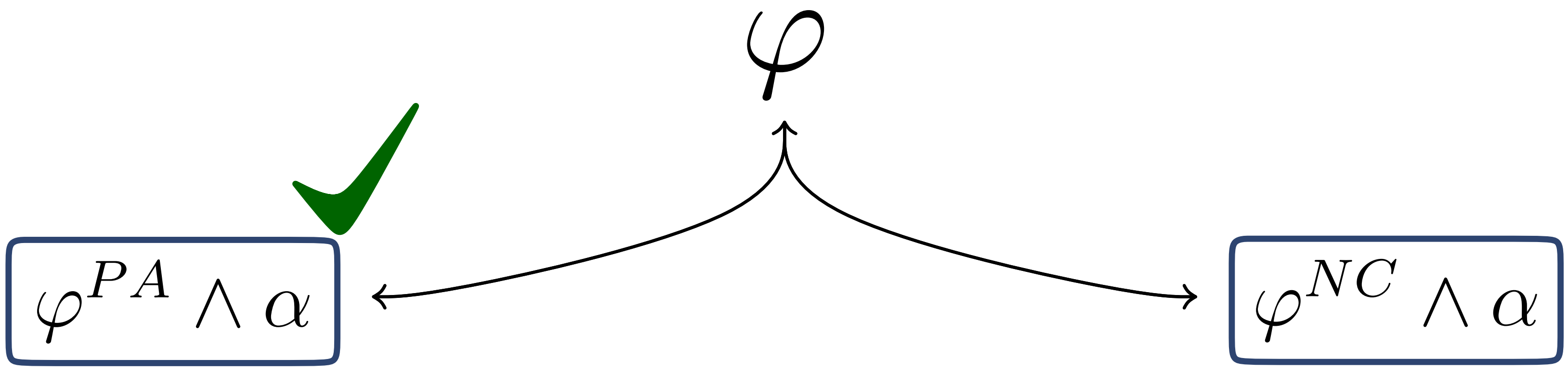
$\varphi : TSL$ formula is satisfiable

iff

$(\varphi^{PA} \wedge \alpha)$ is satisfiable and $(\varphi^{NC} \wedge \alpha)$ is satisfiable

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



Presburguer Arithmetic

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula

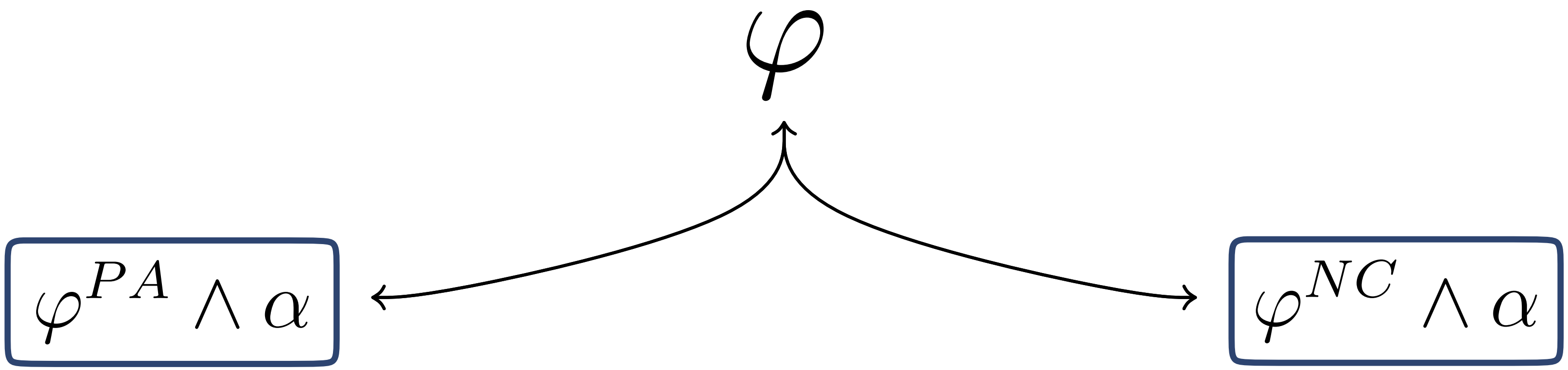
$$\varphi$$

$$\varphi^{PA} \land \alpha \qquad \longleftrightarrow \qquad \longleftrightarrow \qquad \varphi^{NC} \land \alpha$$

- **Gapless model**: we stay only with **interesting levels**

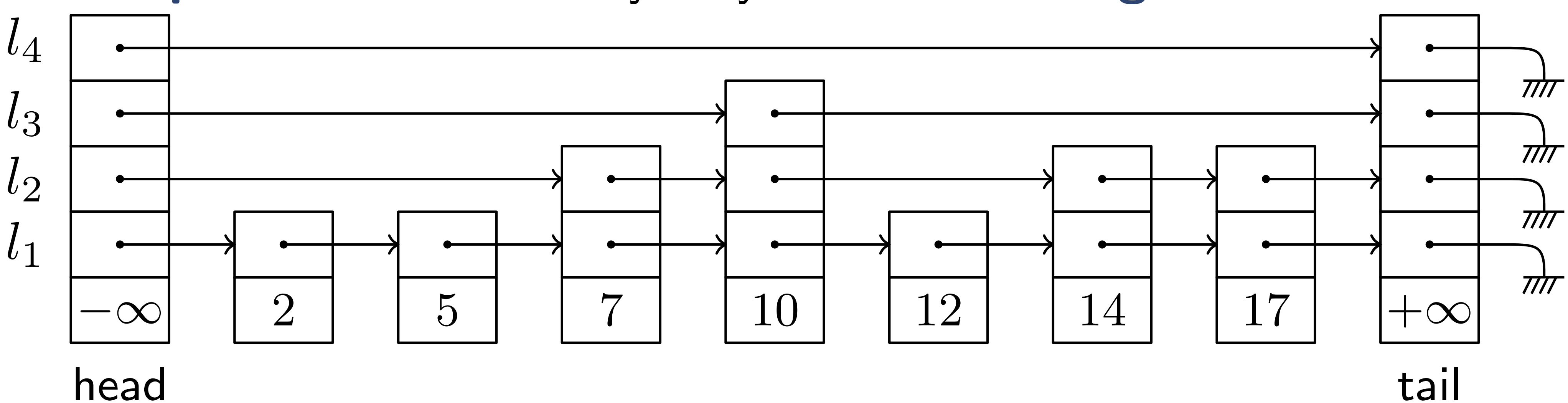
Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



$$V_{\text{level}}(\varphi^{NC} \wedge \alpha) = \{l_1, l_3\}$$

- **Gapless model**: we stay only with **interesting levels**



Decision Procedure for TSL: Correctness

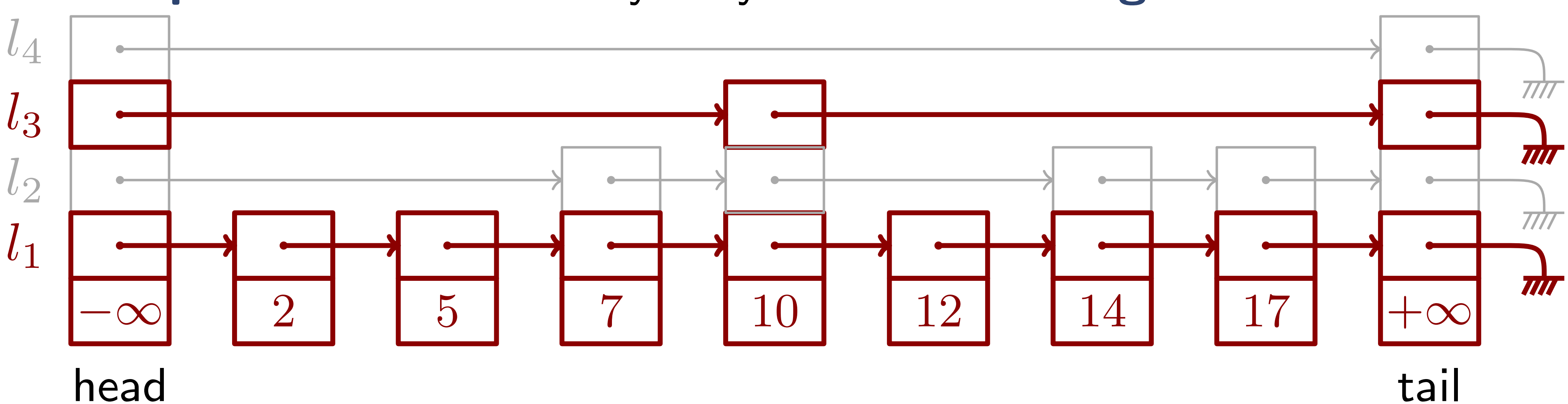
- Let φ be a normalized TSL formula

$$\varphi$$

$$\boxed{\varphi^{PA} \wedge \alpha} \longleftarrow \varphi \longrightarrow \boxed{\varphi^{NC} \wedge \alpha}$$

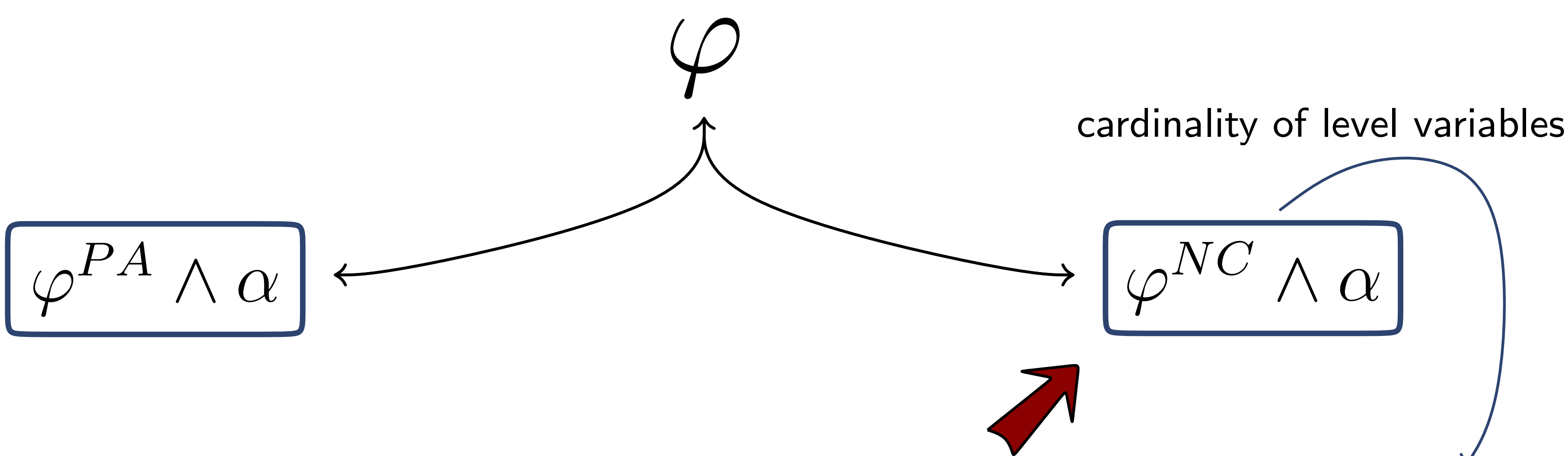
$$V_{\mathsf{level}}(\varphi^{NC} \wedge \alpha) = \{l_1, l_3\}$$

- **Gapless model**: we stay only with **interesting levels**



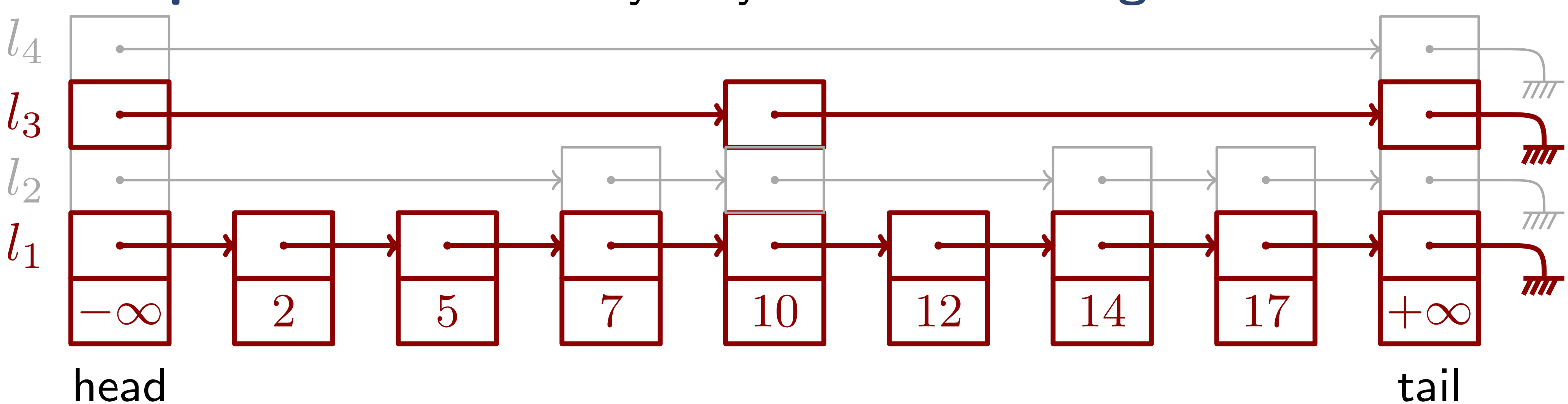
Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



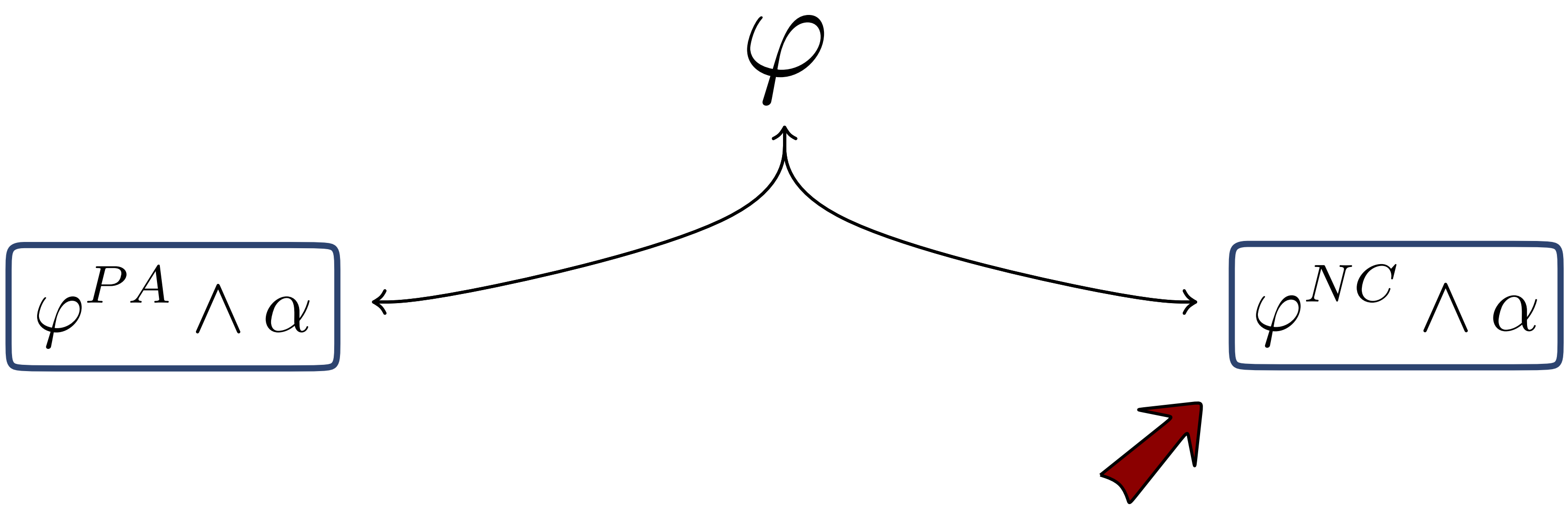
We reduce the formula to $\ulcorner \varphi^{NC} \wedge \alpha \urcorner$: TSL_K

- **Gapless model**: we stay only with **interesting levels**



Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



We reduce the formula to $\ulcorner \varphi^{NC} \wedge \alpha \urcorner$: TSL_K

Theorem:

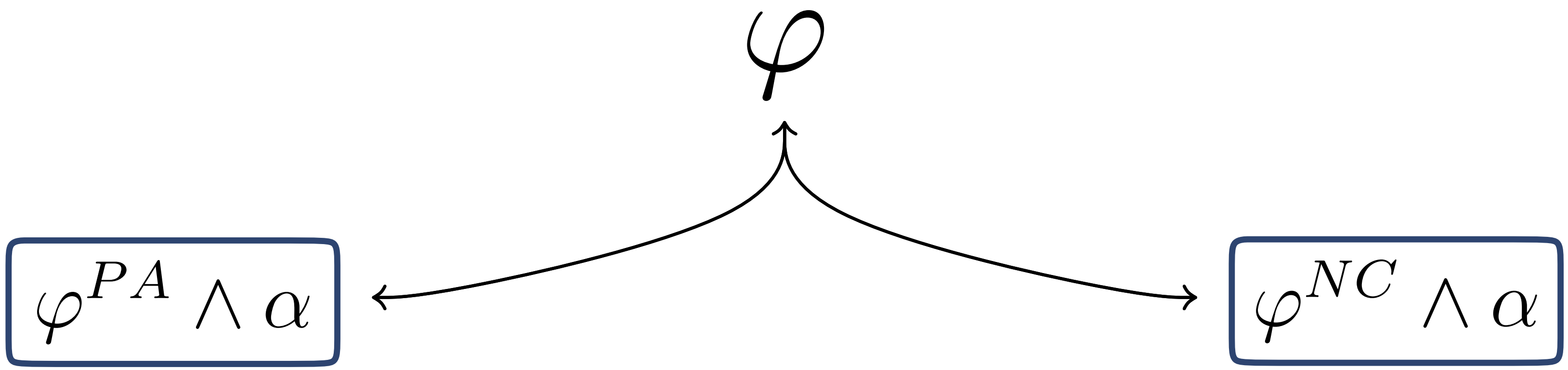
$\varphi : TSL$ formula without constant levels is satisfiable

iff

$\ulcorner \varphi \urcorner$: TSL_K is satisfiable

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



- Reduction $\quad \mathsf{TSL} \longrightarrow \mathsf{TSL}_{\mathsf{K}}$

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula

$$\varphi$$

$$\boxed{\varphi^{PA} \wedge \alpha} \longleftrightarrow \varphi \longleftrightarrow \boxed{\varphi^{NC} \wedge \alpha}$$

- Reduction

$$\mathsf{TSL} \longrightarrow \mathsf{TSL}_{\mathsf{K}}$$

$$\ulcorner c = mkcell(e, k, A, l) \urcorner \qquad c = (e, k, v_{A[0]}, \ldots, v_{A[K-1]})$$

$$v_{A[l]} = A(l)$$

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula

$$\varphi$$

$$\varphi^{PA} \wedge \alpha \longleftarrow \varphi \longrightarrow \varphi^{NC} \wedge \alpha$$

- Reduction

$$\mathsf{TSL} \longrightarrow \mathsf{TSL}_{\mathsf{K}}$$

$\ulcorner c = mkcell(e, k, A, l) \urcorner$	$c = (e, k, v_{A[0]}, \ldots, v_{A[K-1]})$
$\ulcorner a = A[l] \urcorner$	$\bigwedge_{i=0\ldots\mathsf{K}-1} l = i \rightarrow a = v_{A[i]}$

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula

$$\varphi$$

$$\varphi^{PA} \wedge \alpha \quad \longleftarrow \quad \varphi \quad \longrightarrow \quad \varphi^{NC} \wedge \alpha$$

- Reduction

TSL	$\longrightarrow$ $\mathsf{TSL_K}$
$\ulcorner c = mkcell(e,k,A,l) \urcorner$	$c = (e, k, v_{A[0]}, \ldots, v_{A[K-1]})$
$\ulcorner a = A[l] \urcorner$	$\bigwedge_{i=0 \ldots \mathsf{K}-1} l = i \rightarrow a = v_{A[i]}$
$\ulcorner B = A\{l \leftarrow a\} \urcorner$	$\left(\bigwedge_{i=0 \ldots \mathsf{K}-1} l = i \rightarrow a = v_{B[i]} \right) \wedge \left(\bigwedge_{j=0 \ldots \mathsf{K}-1} l \neq j \rightarrow v_{B[j]} = v_{A[j]} \right)$

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula

$$\varphi$$

$$\boxed{\varphi^{PA} \wedge \alpha} \longleftrightarrow \varphi \longleftrightarrow \boxed{\varphi^{NC} \wedge \alpha}$$

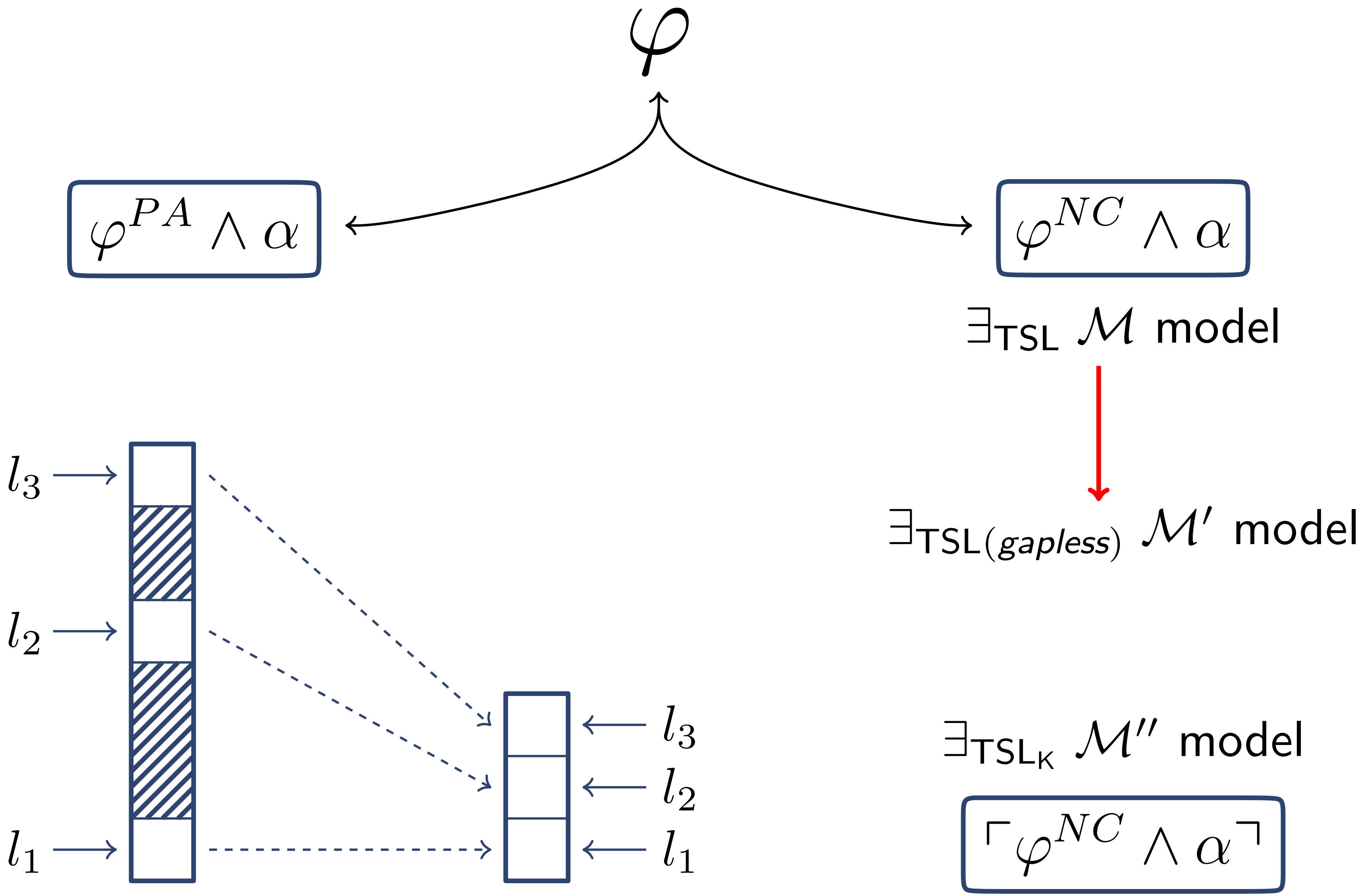
$\exists_{\mathsf{TSL}}\ \mathcal{M}$ model

$\exists_{\mathsf{TSL_K}}\ \mathcal{M}''$ model

$$\boxed{\ulcorner \varphi^{NC} \wedge \alpha \urcorner}$$

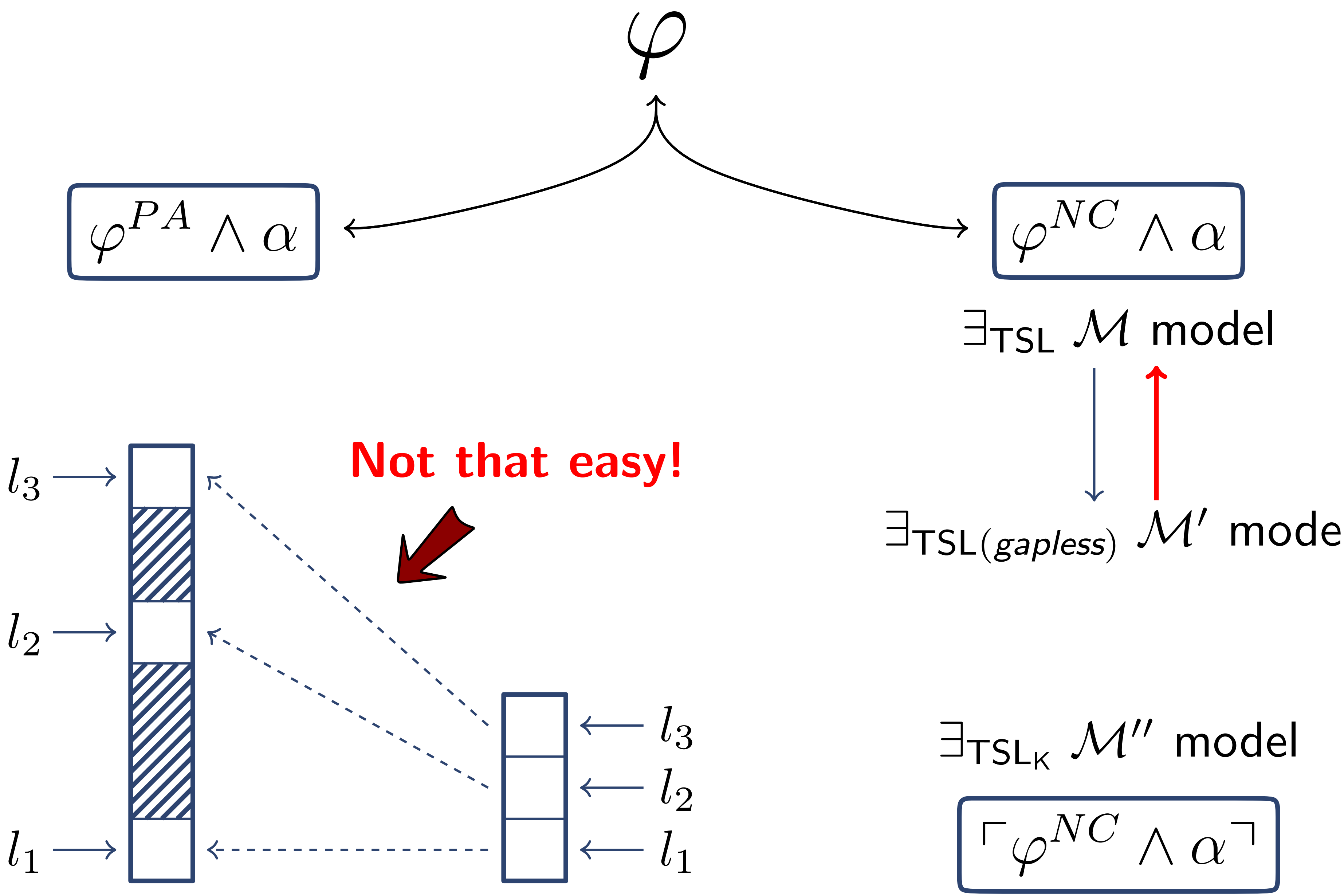
Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula

$$\varphi$$

$$\varphi^{PA} \wedge \alpha \qquad \qquad \varphi^{NC} \wedge \alpha$$

$\exists_{\mathsf{TSL}}\ \mathcal{M}$ model

$$B = A\{l_1 \leftarrow a\}$$

$\exists_{\mathsf{TSL}(gapless)}\ \mathcal{M}'$ model

$\exists_{\mathsf{TSL_K}}\ \mathcal{M}''$ model

$$\ulcorner \varphi^{NC} \wedge \alpha \urcorner$$

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula

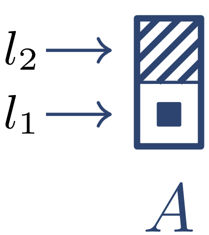
$$\varphi$$

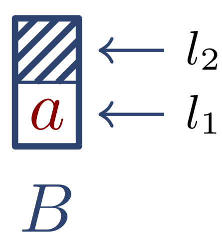
$$\boxed{\varphi^{PA} \wedge \alpha} \qquad \boxed{\varphi^{NC} \wedge \alpha}$$

$\exists_{\mathsf{TSL}}\ \mathcal{M}$ model

$B = A\{l_1 \leftarrow a\}$

$\exists_{\mathsf{TSL}(gapless)}\ \mathcal{M}'$ model





$\exists_{\mathsf{TSL}_\mathsf{K}}\ \mathcal{M}''$ model

$$\boxed{\ulcorner \varphi^{NC} \wedge \alpha \urcorner}$$

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula

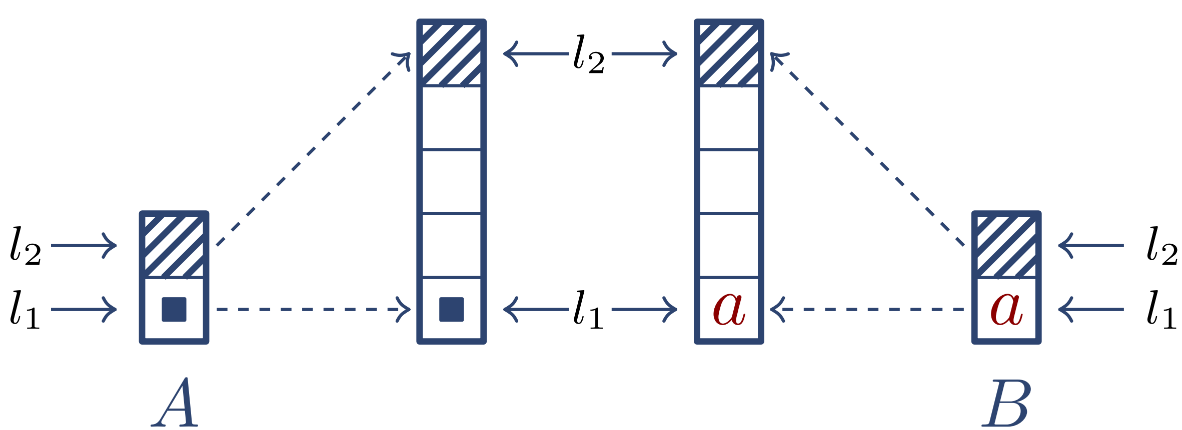
$$\varphi$$

$$\varphi^{PA} \wedge \alpha \qquad \qquad \varphi^{NC} \wedge \alpha$$

$\exists_{\mathsf{TSL}}$ $\mathcal{M}$ model

$B = A\{l_1 \leftarrow a\}$

$\exists_{\mathsf{TSL}(gapless)}$ $\mathcal{M}'$ model

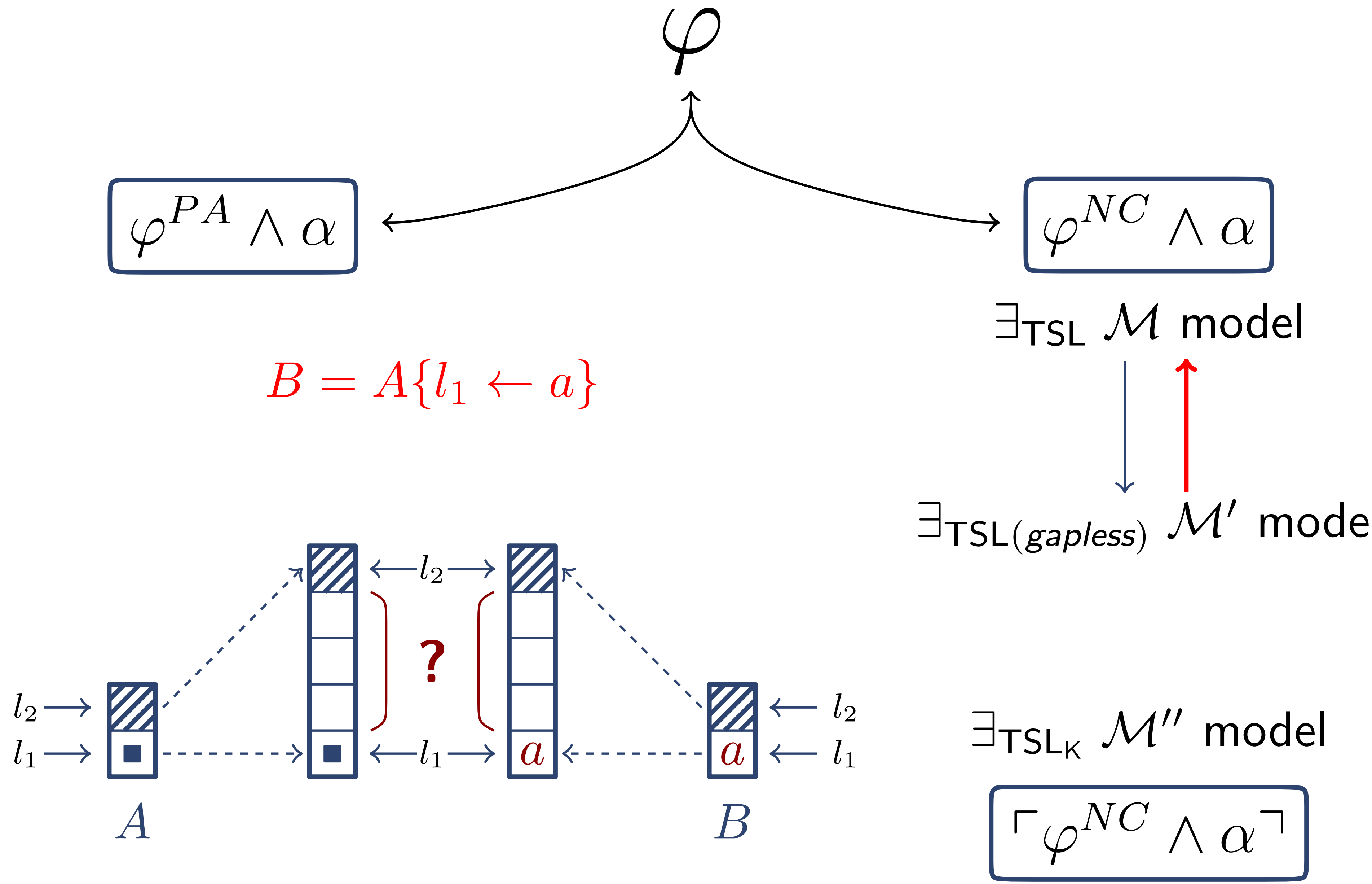


$\exists_{\mathsf{TSL_K}}$ $\mathcal{M}''$ model

$$\ulcorner \varphi^{NC} \wedge \alpha \urcorner$$

Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula

$$\varphi$$

$\boxed{\varphi^{PA} \land \alpha}$ $\quad$ $\boxed{\varphi^{NC} \land \alpha}$

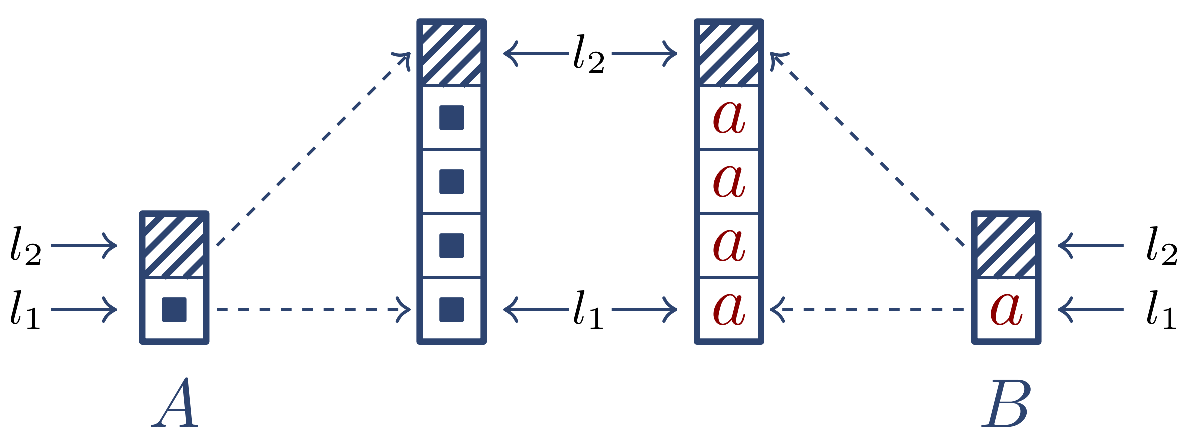
$\exists_{\mathsf{TSL}}\ \mathcal{M}$ model

$\exists_{\mathsf{TSL}(\textit{gapless})}\ \mathcal{M}'$ model

$\exists_{\mathsf{TSL_K}}\ \mathcal{M}''$ model

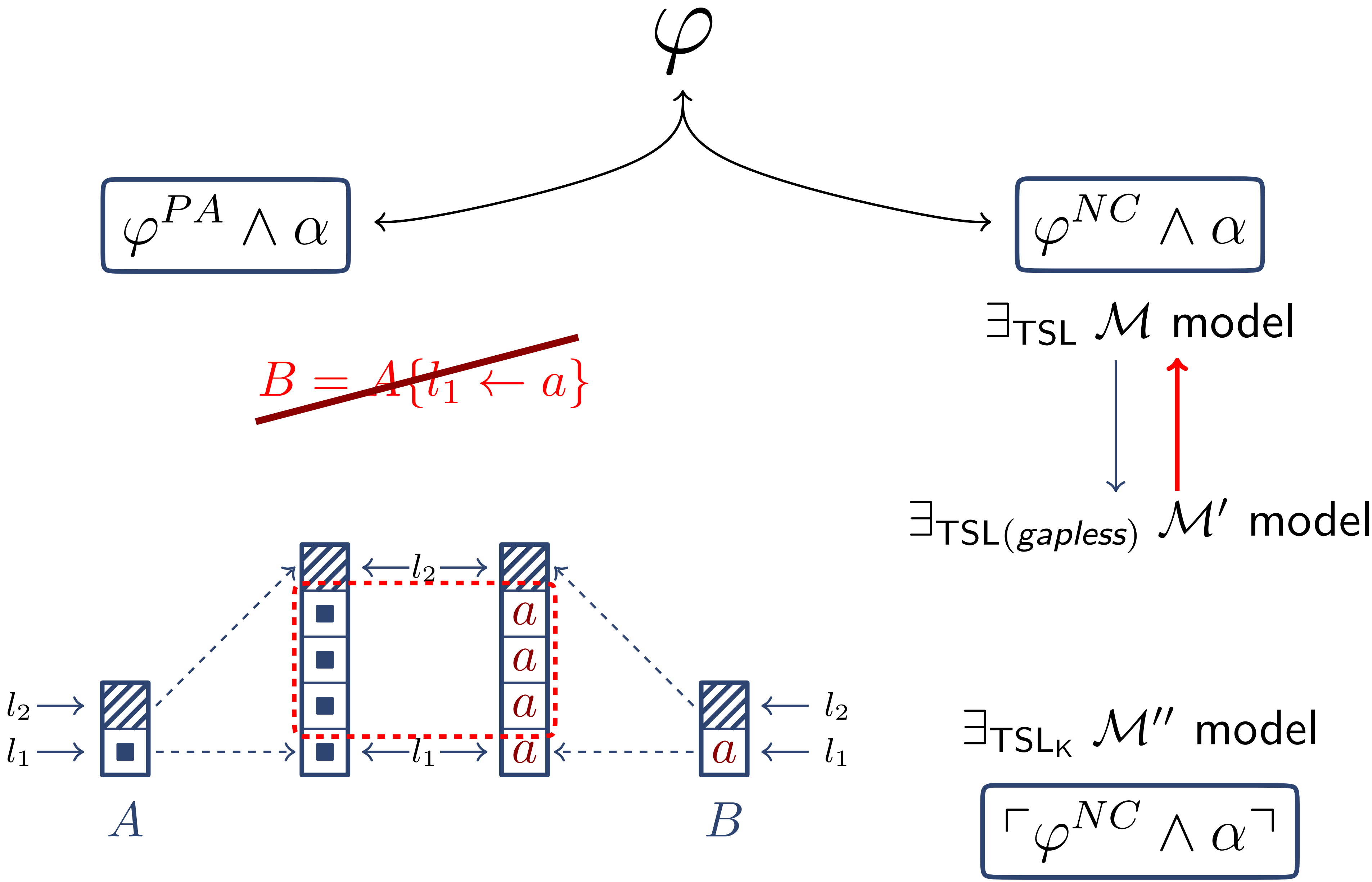
$\boxed{\ulcorner \varphi^{NC} \land \alpha \urcorner}$

$$B = A\{l_1 \leftarrow a\}$$



Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula

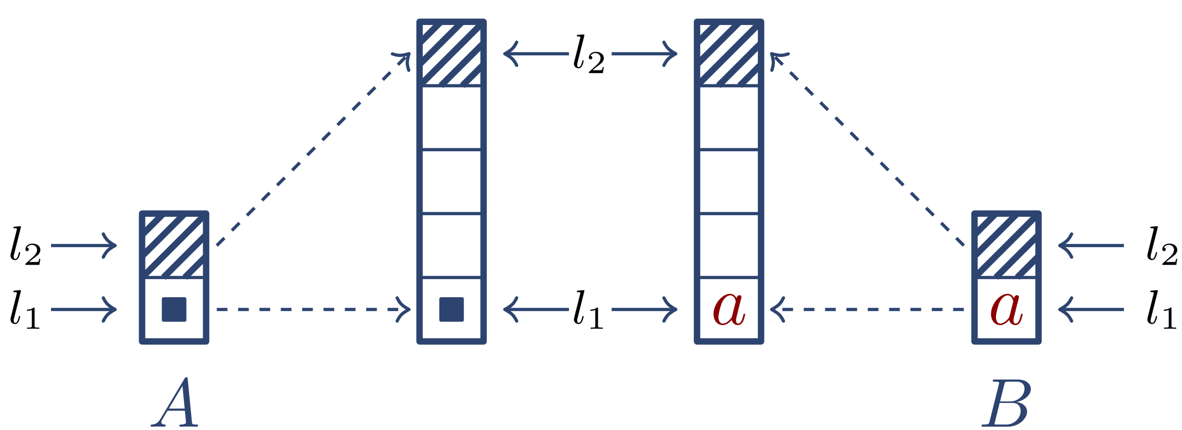
$$\varphi$$

$$\varphi^{PA} \wedge \alpha \qquad\qquad \varphi^{NC} \wedge \alpha$$

$\exists_{\mathsf{TSL}}\ \mathcal{M}$ model

$$B = A\{l_1 \leftarrow a\}$$

$\exists_{\mathsf{TSL}(gapless)}\ \mathcal{M}'$ model

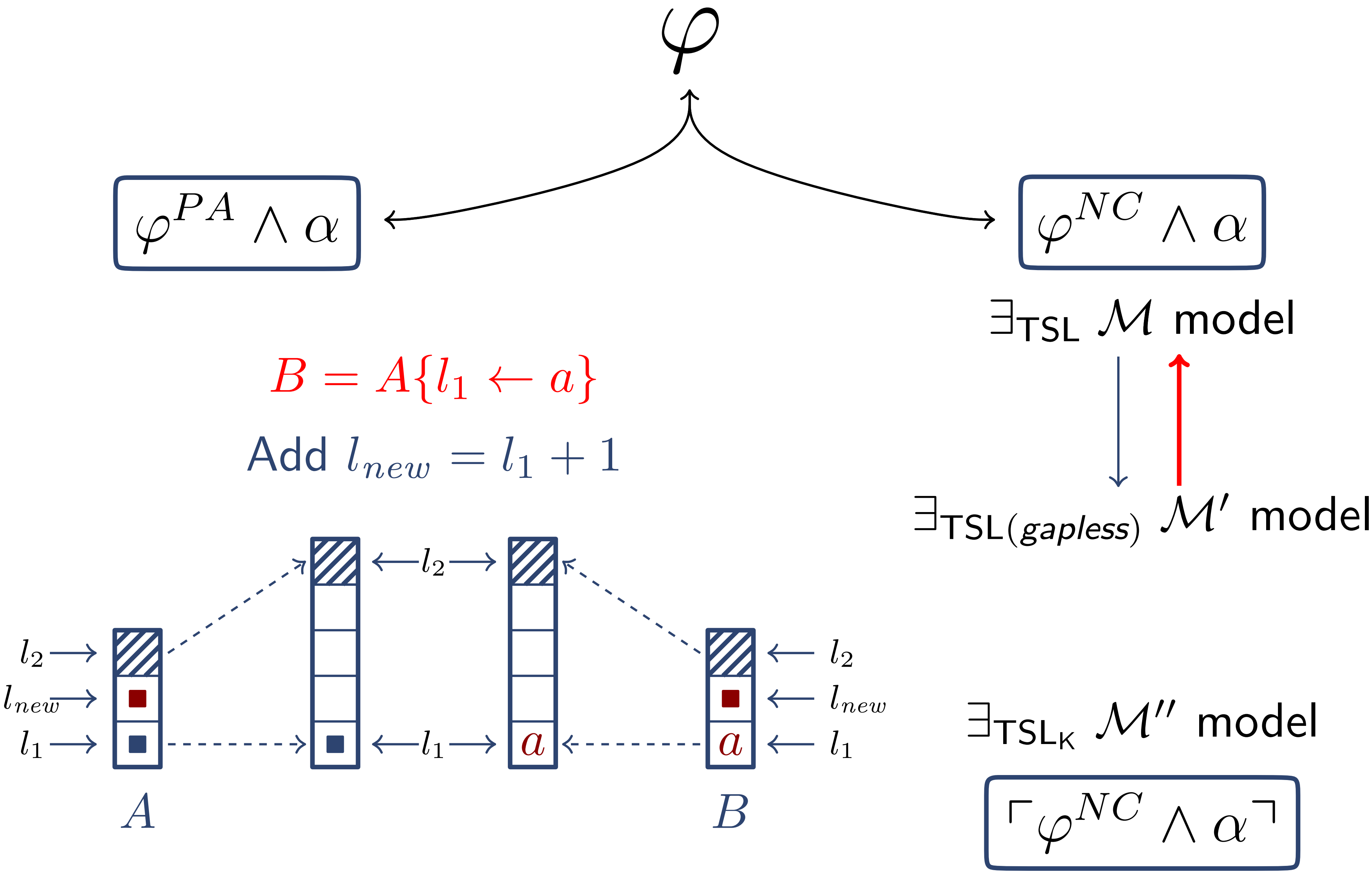


$\exists_{\mathsf{TSL}_\mathsf{K}}\ \mathcal{M}''$ model

$$\ulcorner \varphi^{NC} \wedge \alpha \urcorner$$

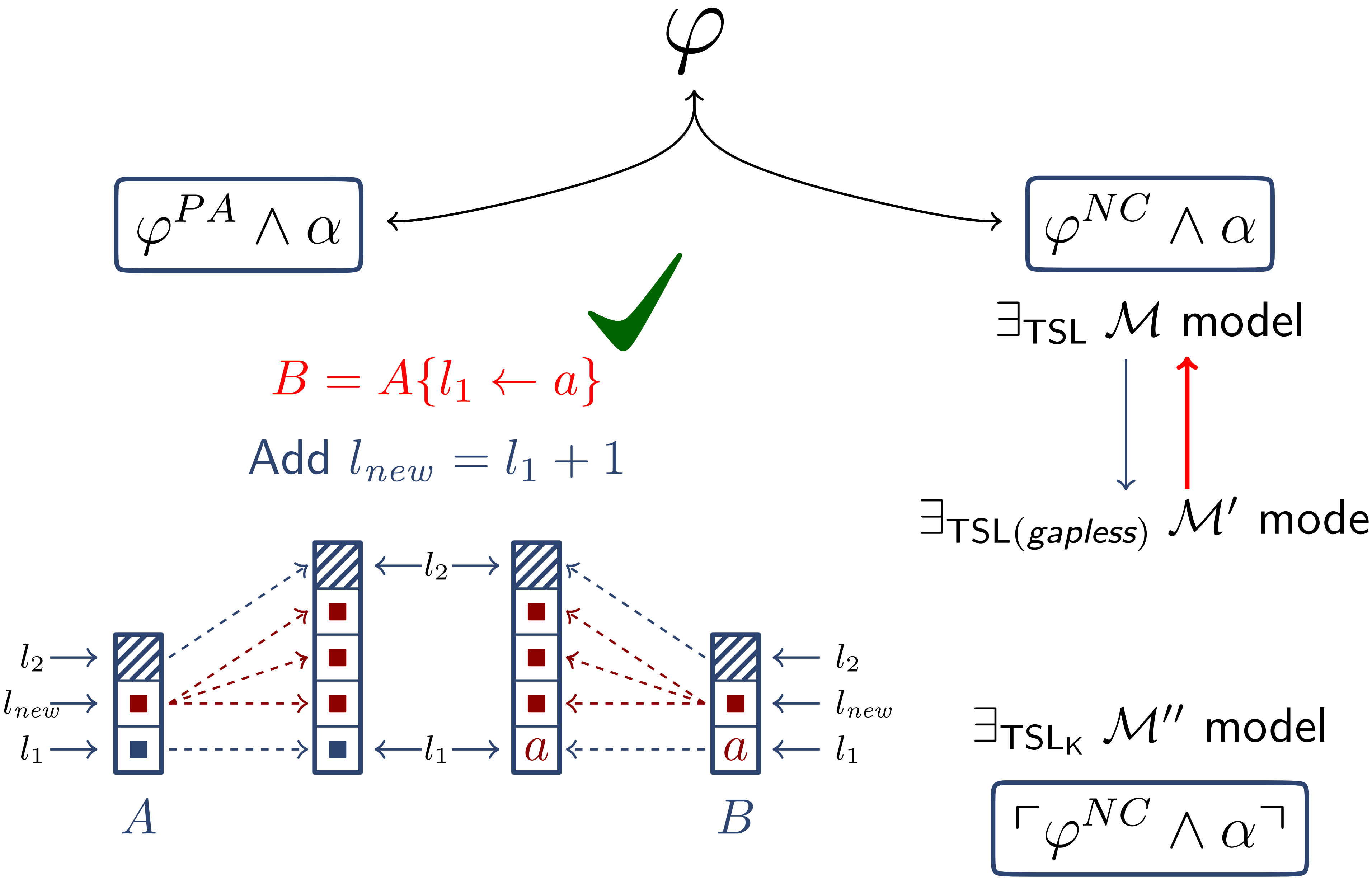
Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



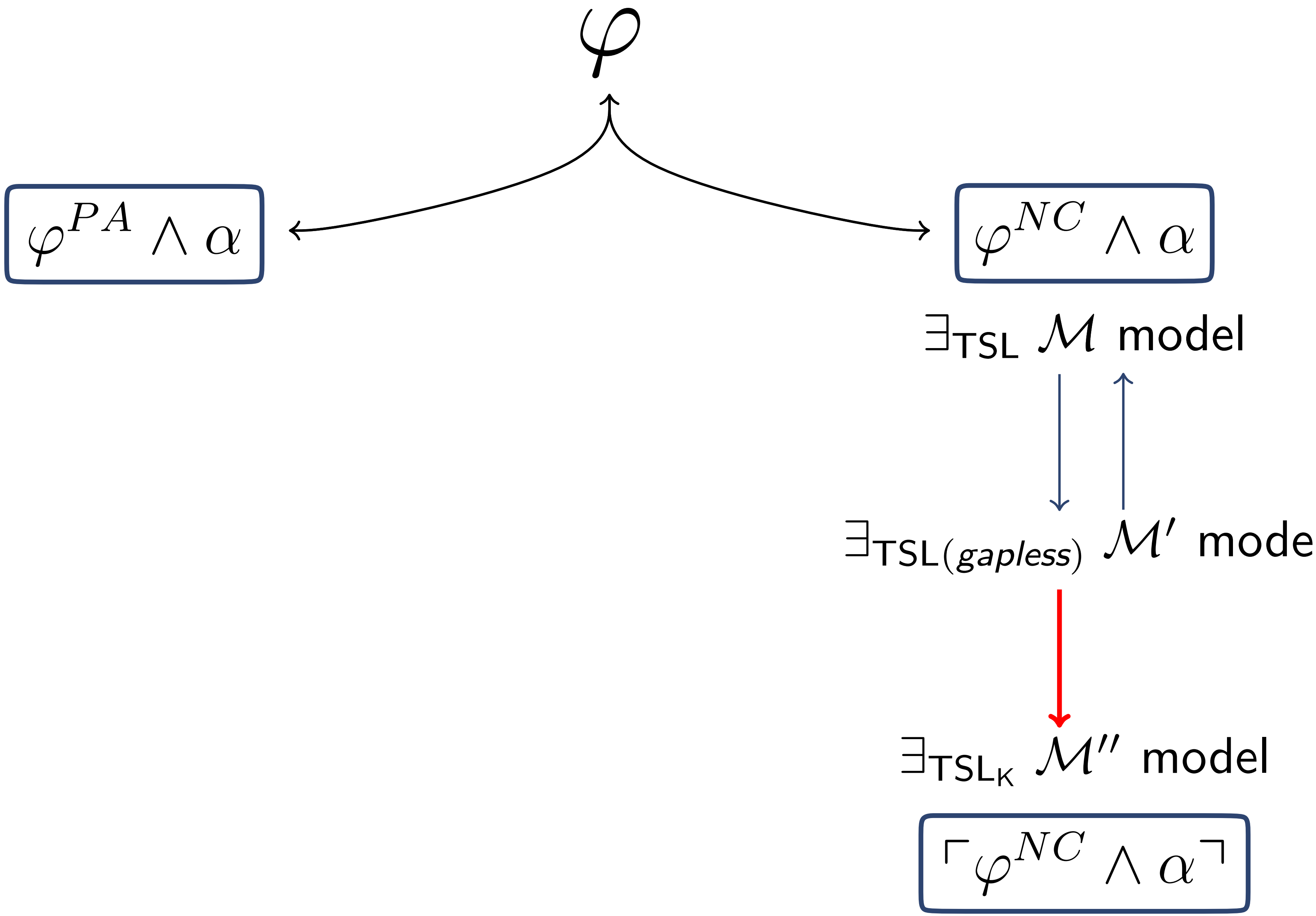
Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



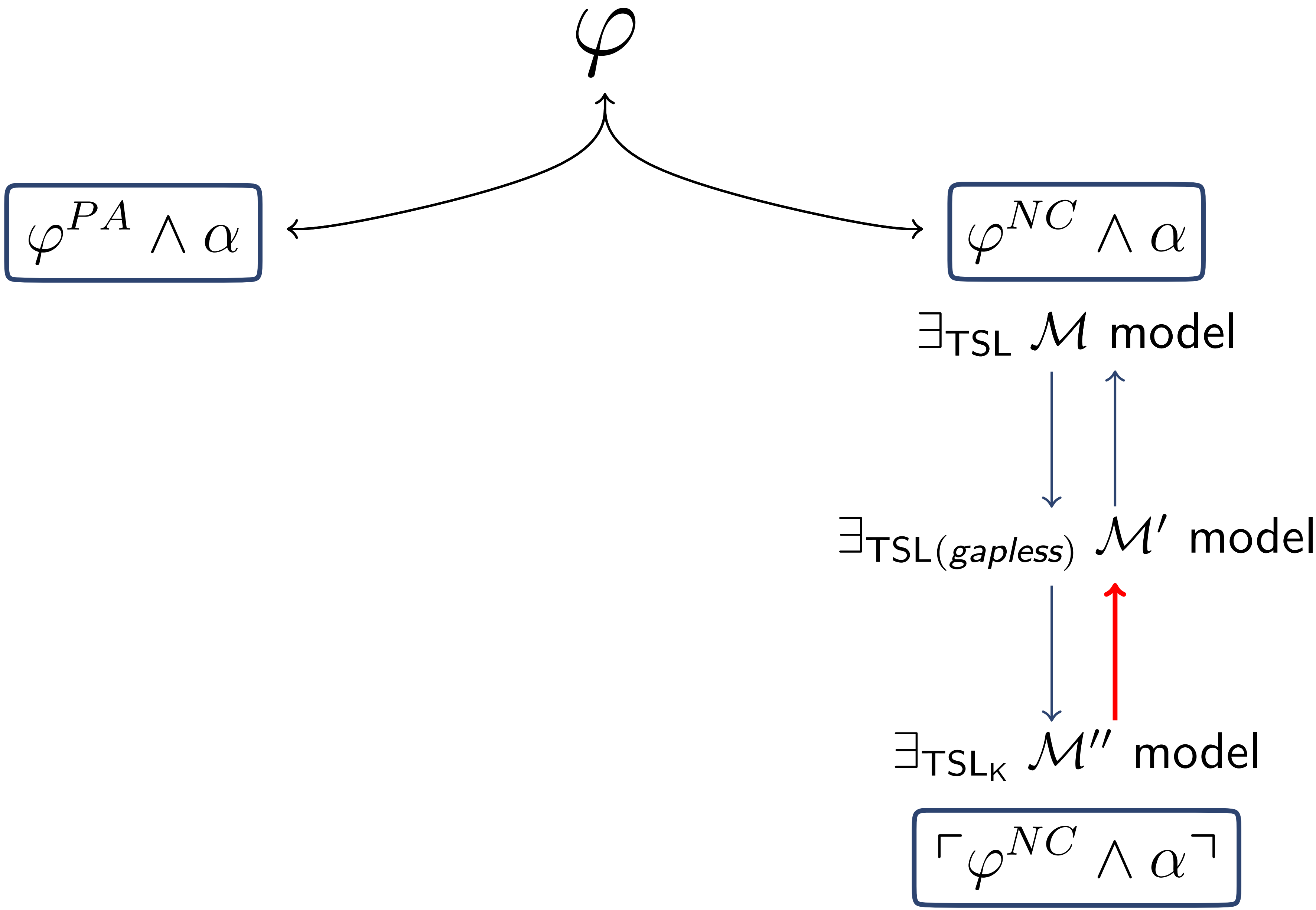
Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



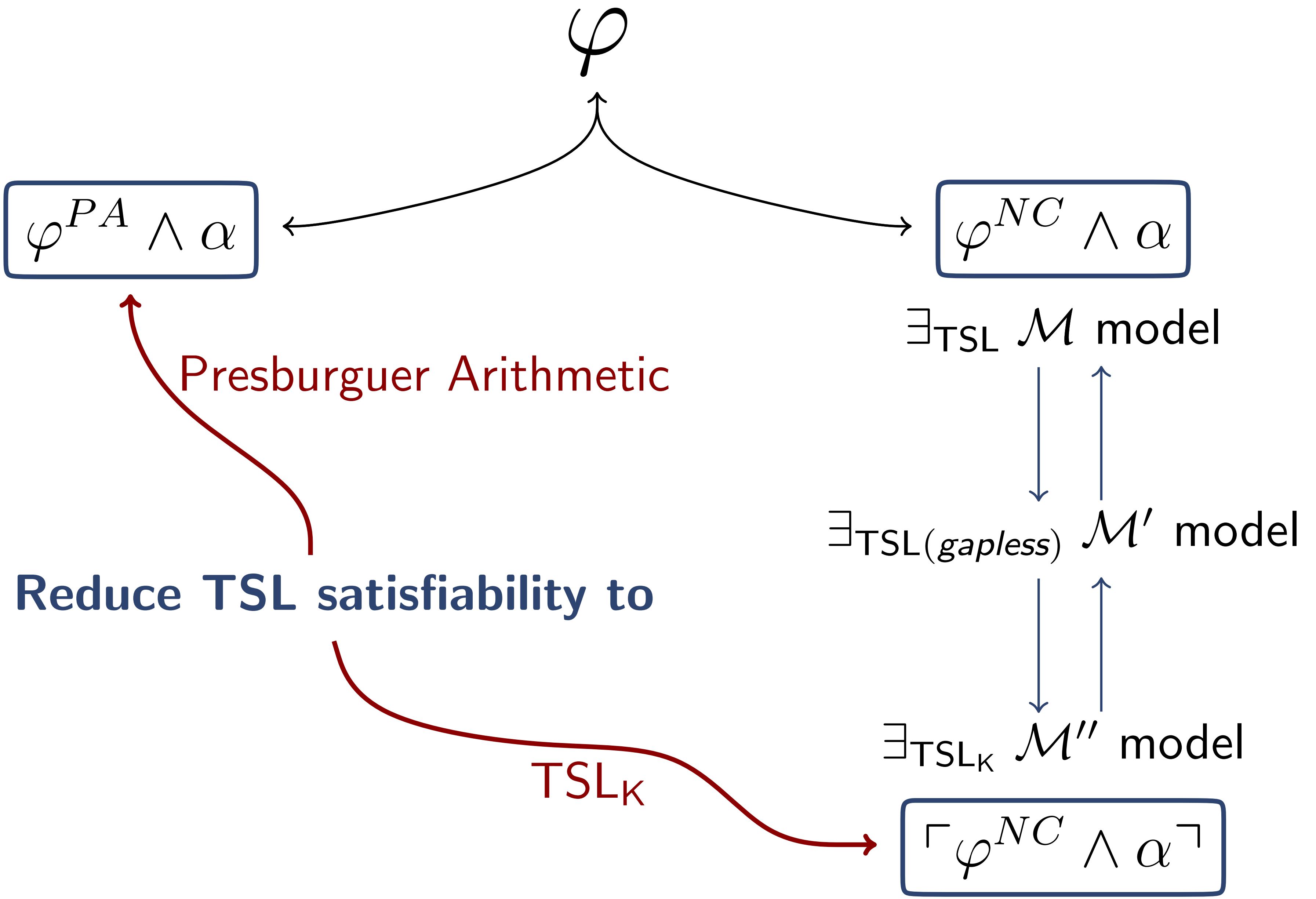
Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



Decision Procedure for TSL: Correctness

- Let φ be a normalized TSL formula



Conclusions

- We defined **TSL**, a theory for skiplists of arbitrary height
- We proved TSL **decidable**...
- ... by reducing to **Presburguer Arithmetic** and **TSL_K**
- **TSL_K** can reason about memory, cells, pointers, regions, reachability, ordered lists and sublists
- **Current and future** work:
  have implementation of DP for TSL_K
  building DP for TSL
  thinking on DP for concurrent skiplists
- Many possible **collaborations**:
  DPs as combination, SMTs, implementation