Verifying Linearizability with Hindsight

Peter O'Hearn
Noam Rinetzky | Queen Mary University of London

Martin Vechev
Eran Yahav
Greta Yorsh | IBM Watson Research Center

verification goal

- prove correctness of highly concurrent optimistic data structures implementation
  - memory safety
  - linearizability
  - progress

challenge: interference

- high degree of interference complicates
  - design
  - understanding
  - verification

~~challenge~~ blessing: interference

- high degree of interference ~~complicates~~ can simplify
  - design
  - understanding
  - verification

local design

- thread-local protocol for linearizability
  - efficiency use small atomic accesses
  - correctness maintains global structural invariant of shared state
    - correct: locally-maintainable invariant
      - counterexamples: subtle interactions
    - invariant + local observation → linearizability

proof strategy

- program logic proof local-protocol preserves global invariant (thread-local proof)

proof strategy

- program logic proof local-protocol preserves global invariant (thread-local proof)
- mathematical proof on good traces
  - not about algorithms / programs
- hindsight good trace allows to conclude properties of past states from current state
- local window concluded properties provide (enough) information for linearizability

case study: optimistic set

- list-based implementation [Heller et al.'05]
  - heap, pointers, destructive updates
  - unbounded # of threads
    - linearization points in other threads (contains)
- verified properties
  - memory safety
  - linearizability
  - wait-freedom of contains

sequential specification

$$\mathcal{A}(S) \subseteq N$$

$$\{\ \mathcal{A}(S)=A\ \}\ \ r=remove(k)\ \ \{\ r=k\in A \wedge \mathcal{A}(S)=A\backslash\{k\}\ \}$$

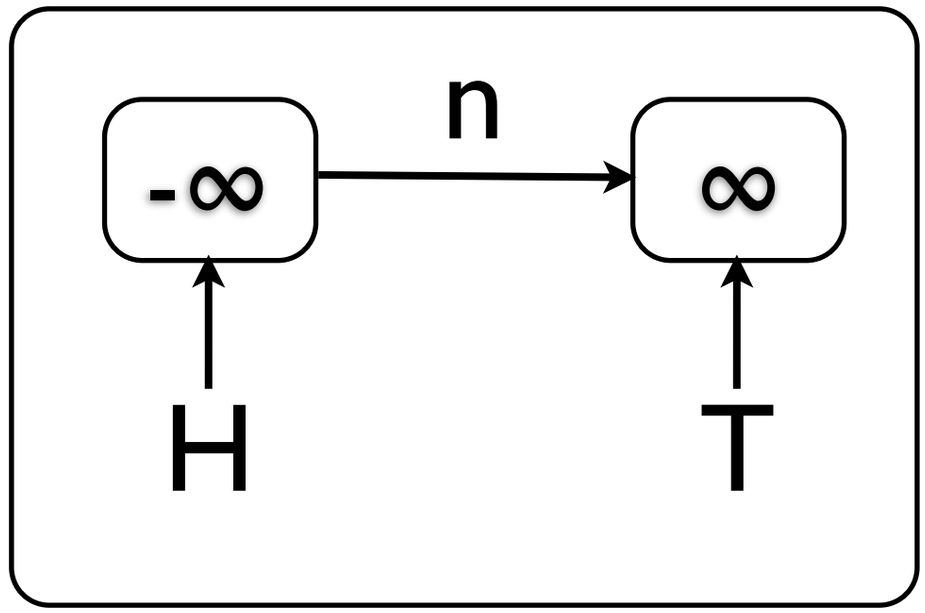
$$\{\ \mathcal{A}(S)=A\ \}\ \ r=add(k)\ \ \{\ r=k\notin A \wedge \mathcal{A}(S)=A \cup\{k\}\ \}$$

$$\{\ \mathcal{A}(S)=A\ \}\ \ r=contains(k)\ \ \{\ r=k\in A \wedge \mathcal{A}(S)=A\ \}$$

optimistic set

```
class N {
    int k;
    N n;
    boolean m;
}
```

N H,T;



initial state

- $\mathcal{A}(\sigma)$ = {keys of elements between H and T}

optimistic set: insert

```
bool insert(int k) {
  while (true){
    p,c=LOCATE(k);
    if (p.n==c && !p.m){
      if (c.k==k) return false;
      x = alloc N();
      x.m = false;
      x.k=k;  x.n=c;  p.n=x;
      return true
} }
```

LOCATE(k)

```
p = H;
c = H.n;
while (c.k < k) {
   p = c;
   c = c.n;
  }
```

optimistic set: remove

```
bool remove(int k) {
  while (true){
    p,c=LOCATE(k);
    if (p.n==c && !p.m){
      if (c.k!=k) return false;
      c.m=true;
      p.n=c.n;
      return true;
    }
} }
```

```
LOCATE(k)
  p = H;
  c = H.n;
  while (c.k < k) {
    p = c;
    c = c.n;
  }
```

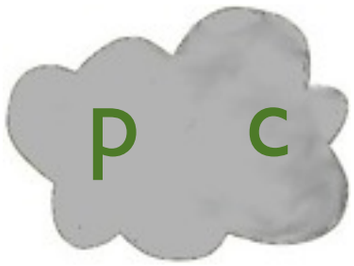
optimistic set

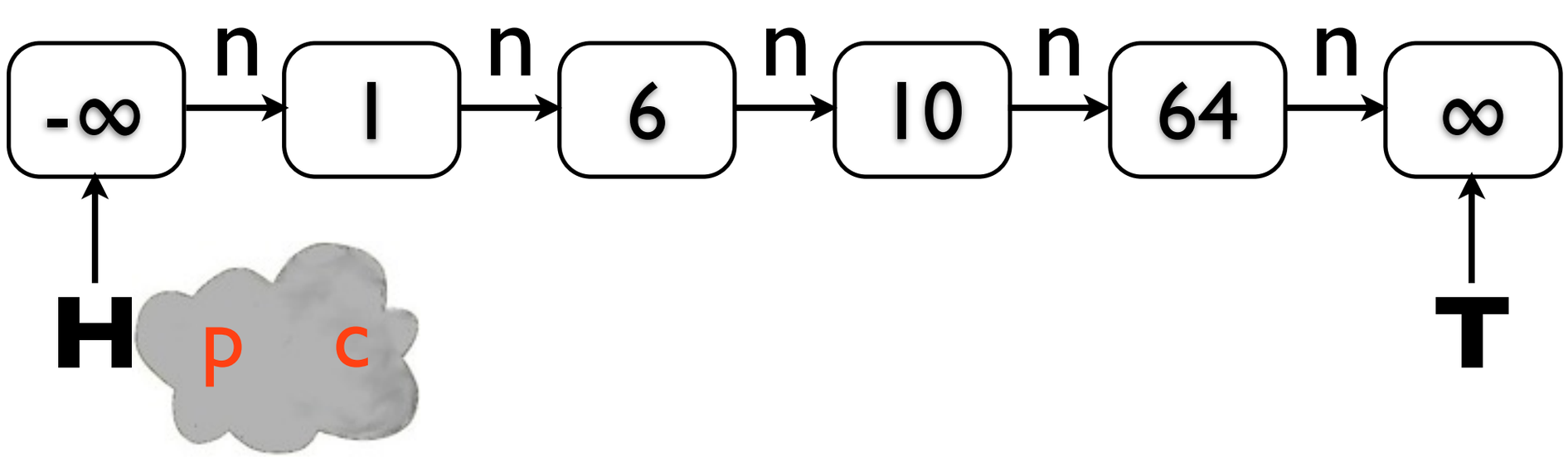
```
bool contains(int k) {

    p,c=LOCATE(k);
    return (c.k==k)

}
```

```
LOCATE(k)
    p = H;
    c = H.n;
    while (c.k < k) {
      p = c;
      c = c.n
    }
```

example 1

remove(6)

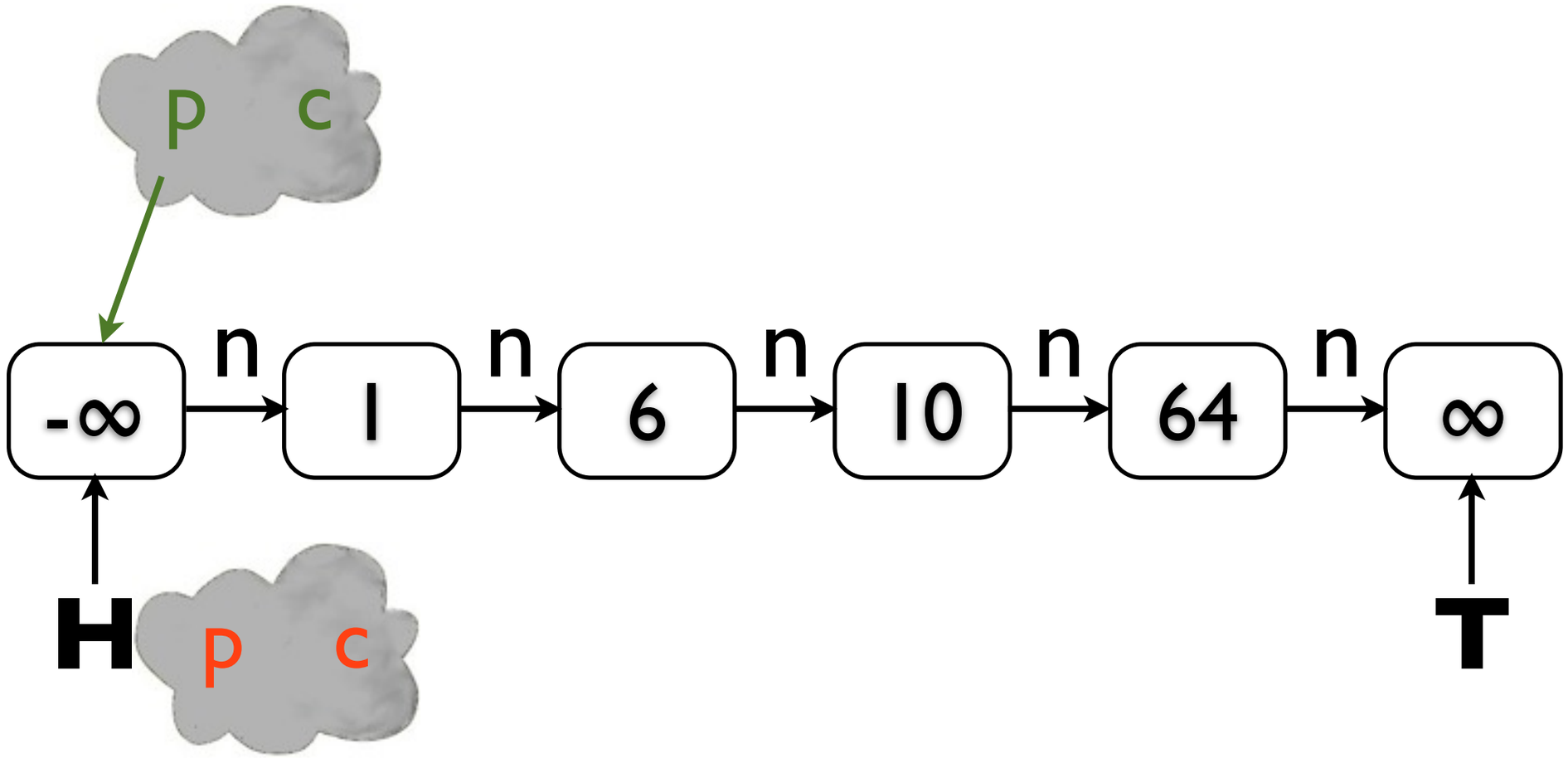




remove(10)

example 1

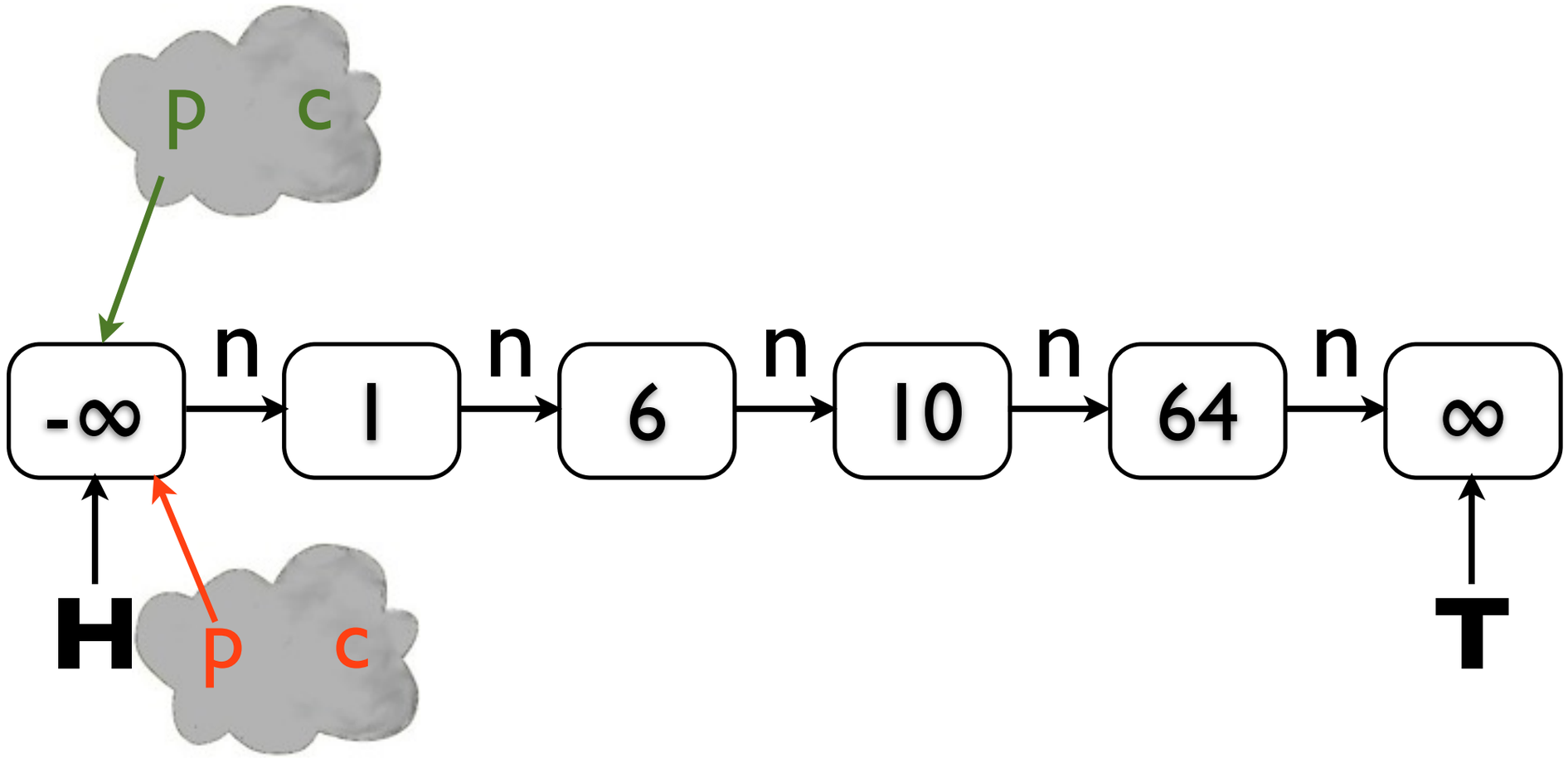
remove(6)



remove(10)

example 1

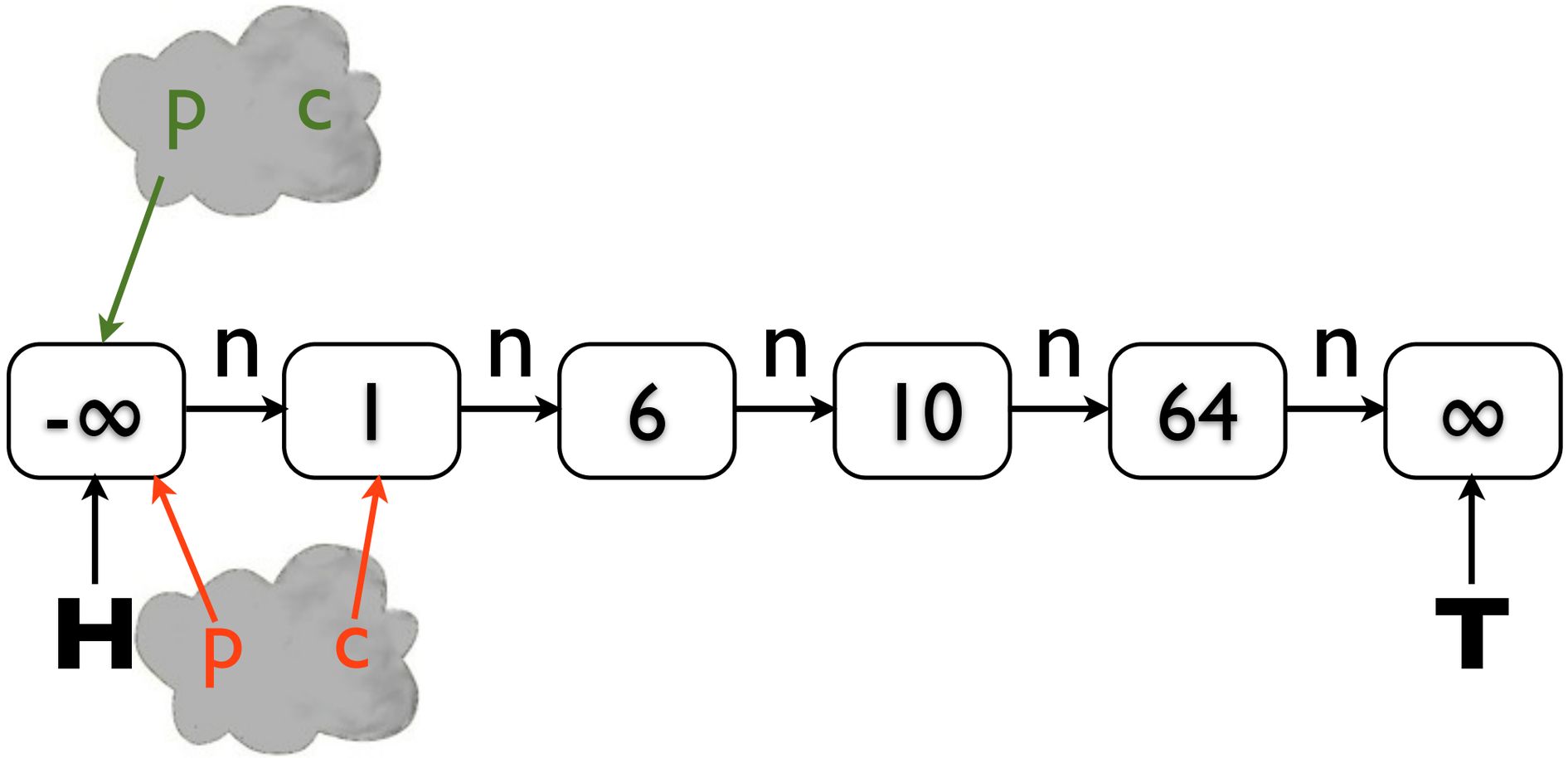
remove(6)



remove(10)

example 1

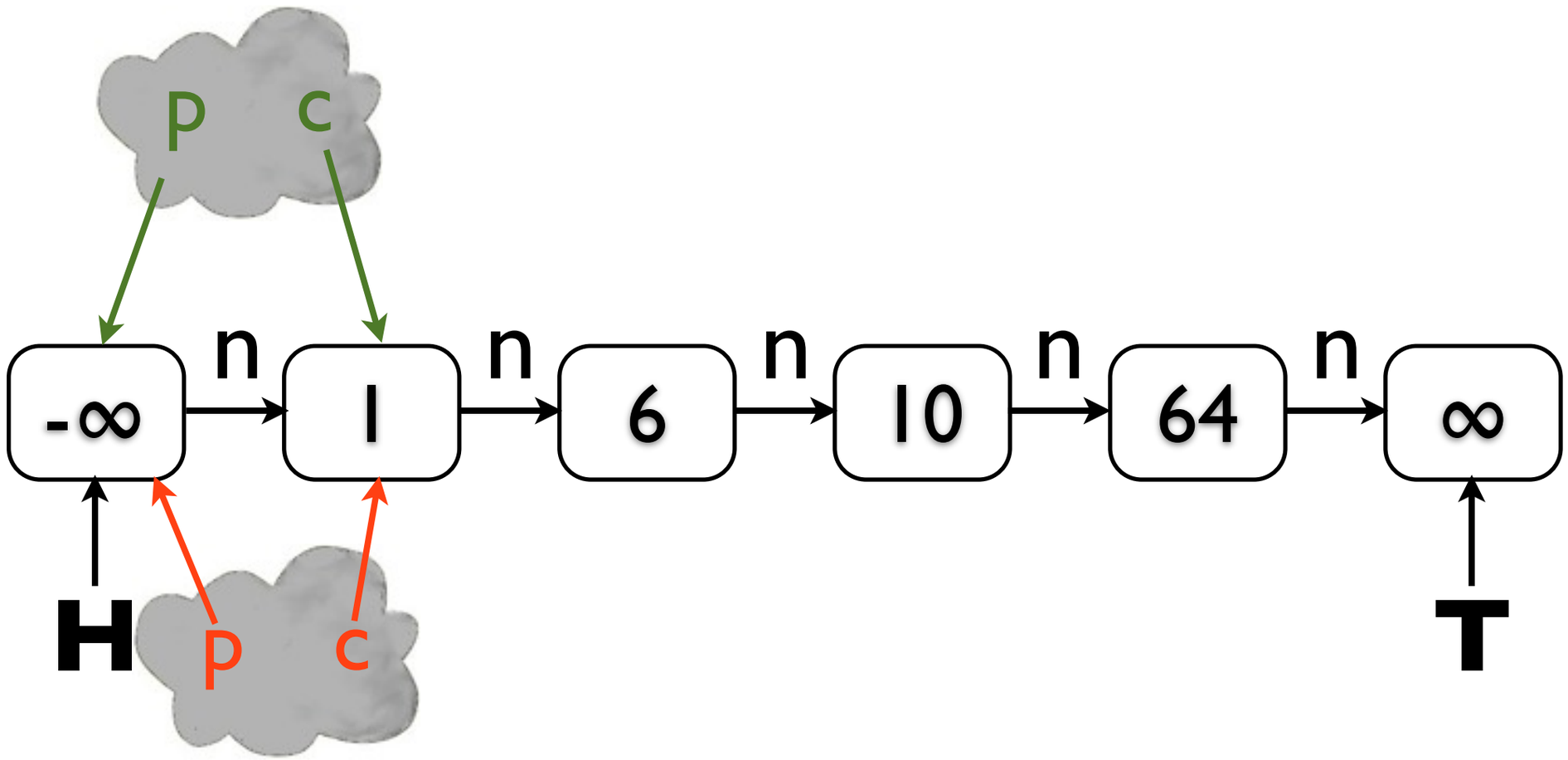
remove(6)



remove(10)

example 1

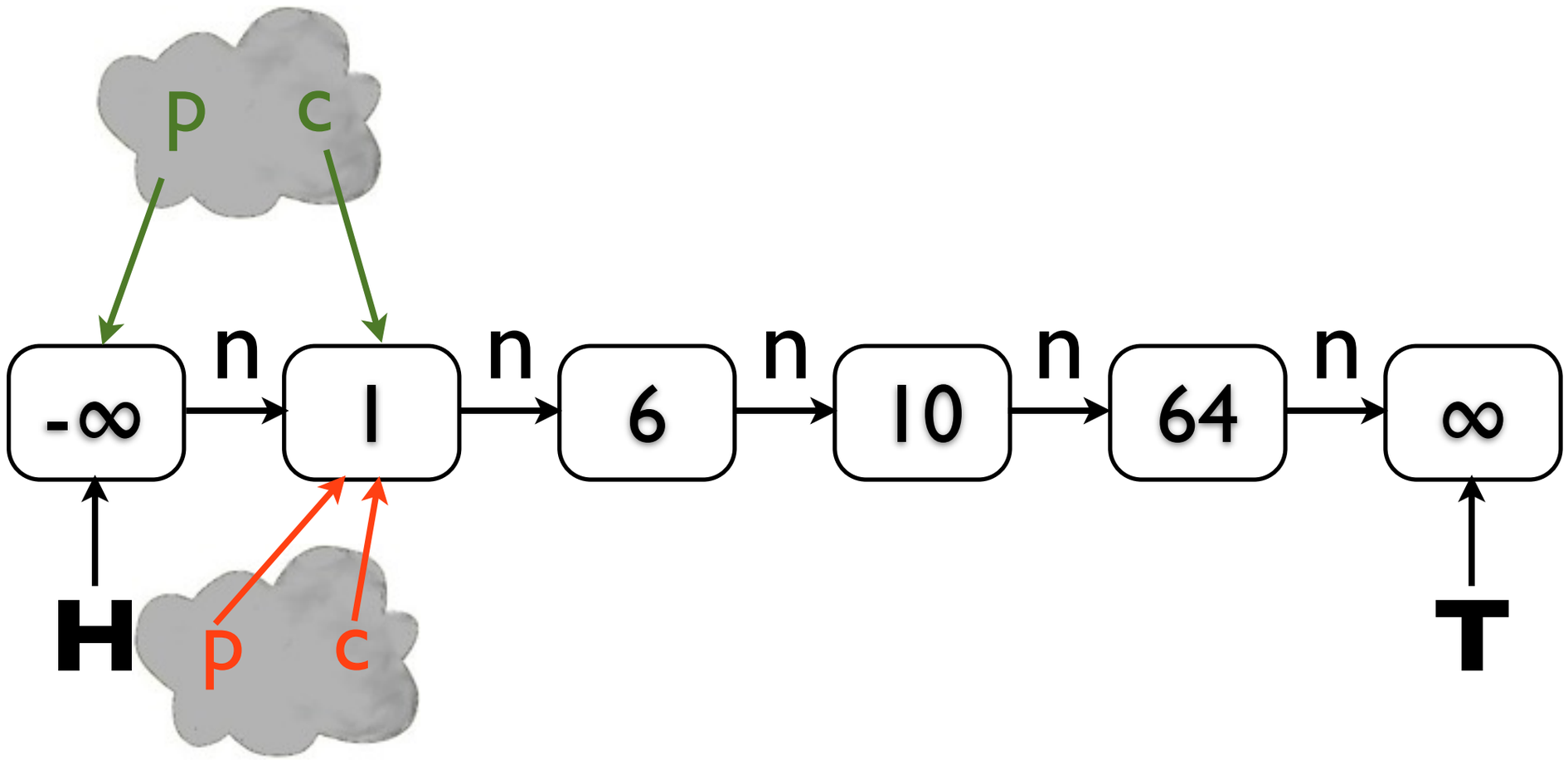
remove(6)



remove(10)

example 1

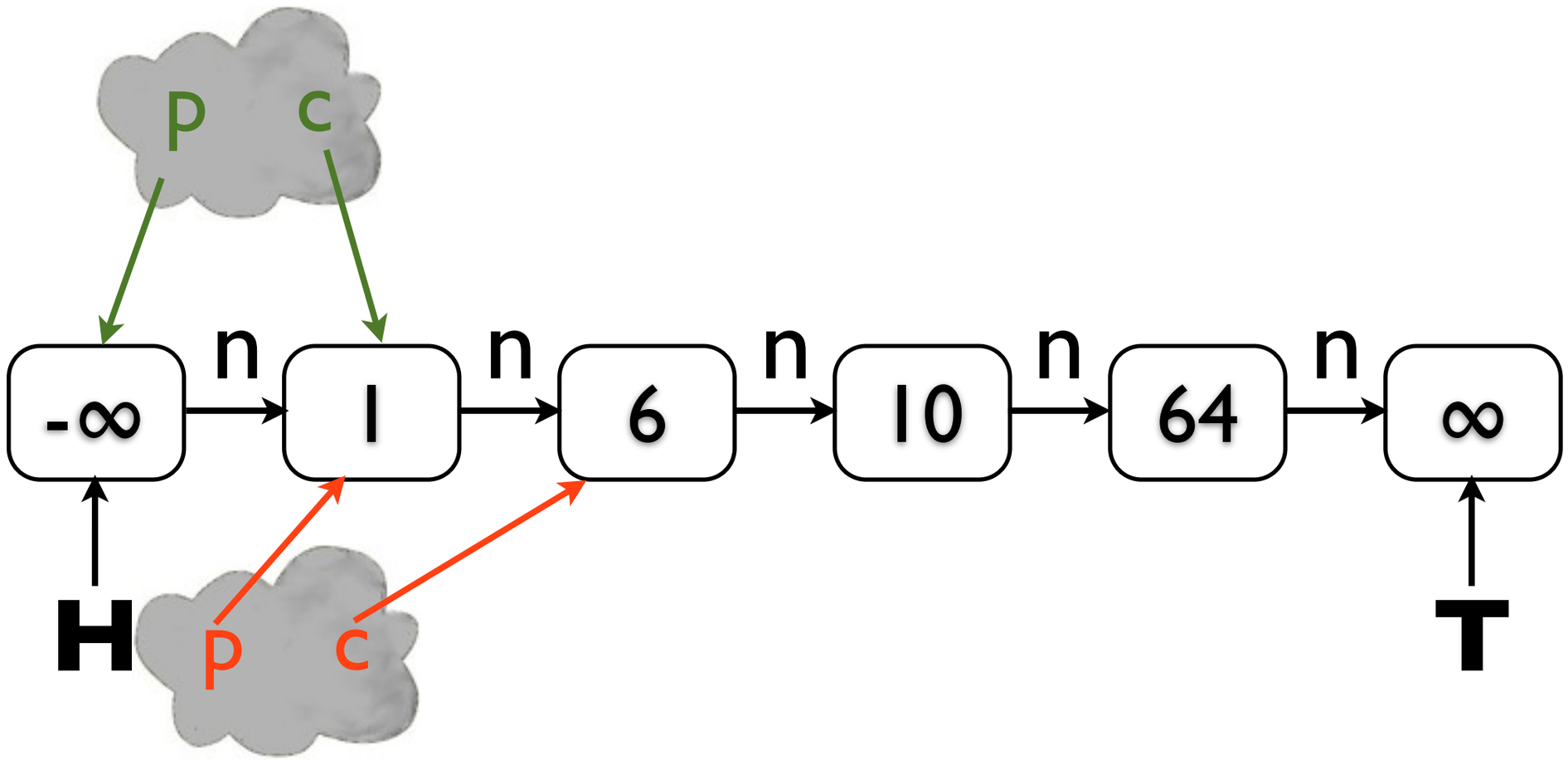
remove(6)



remove(10)

example 1

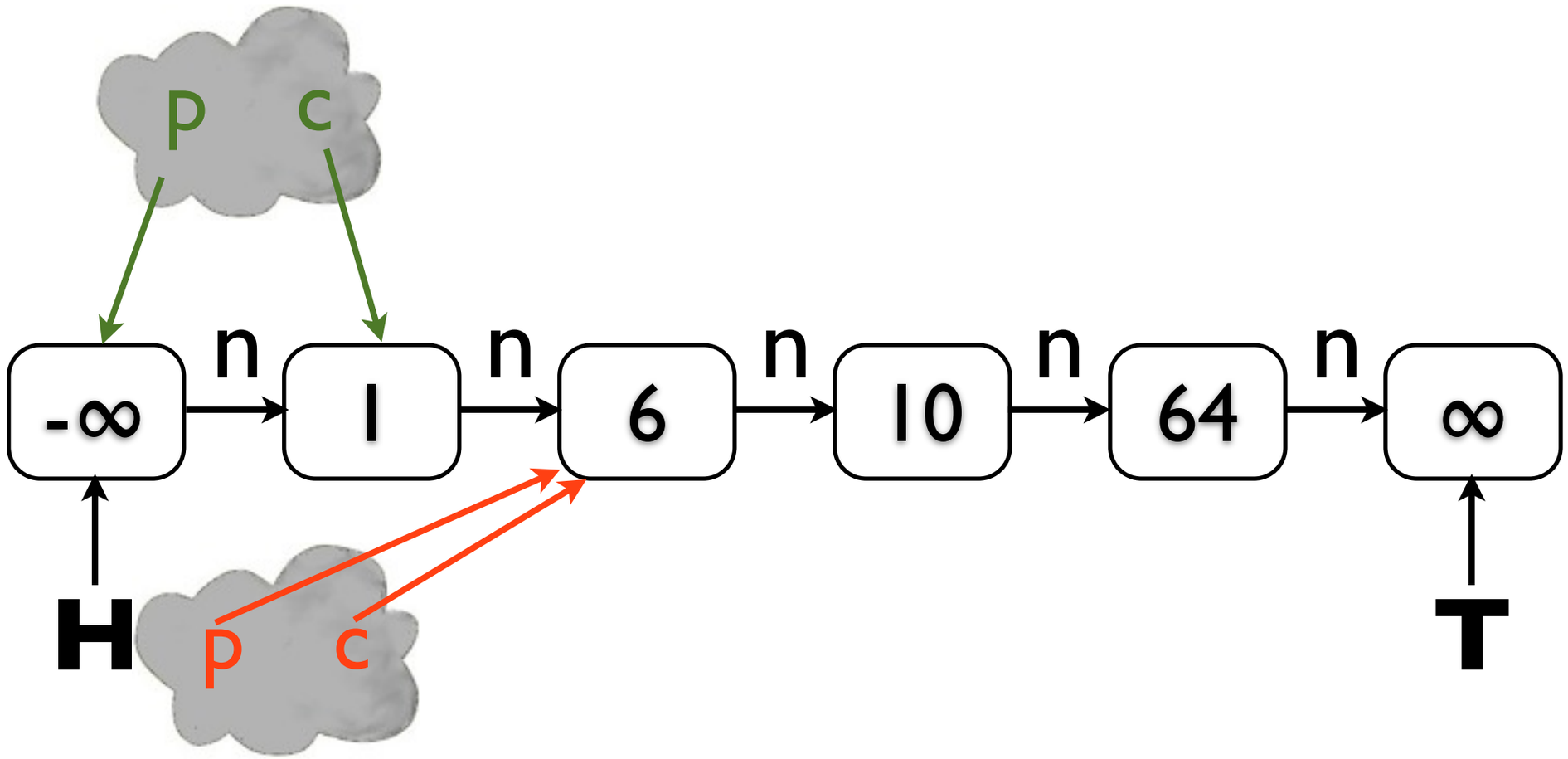
remove(6)



remove(10)

example 1

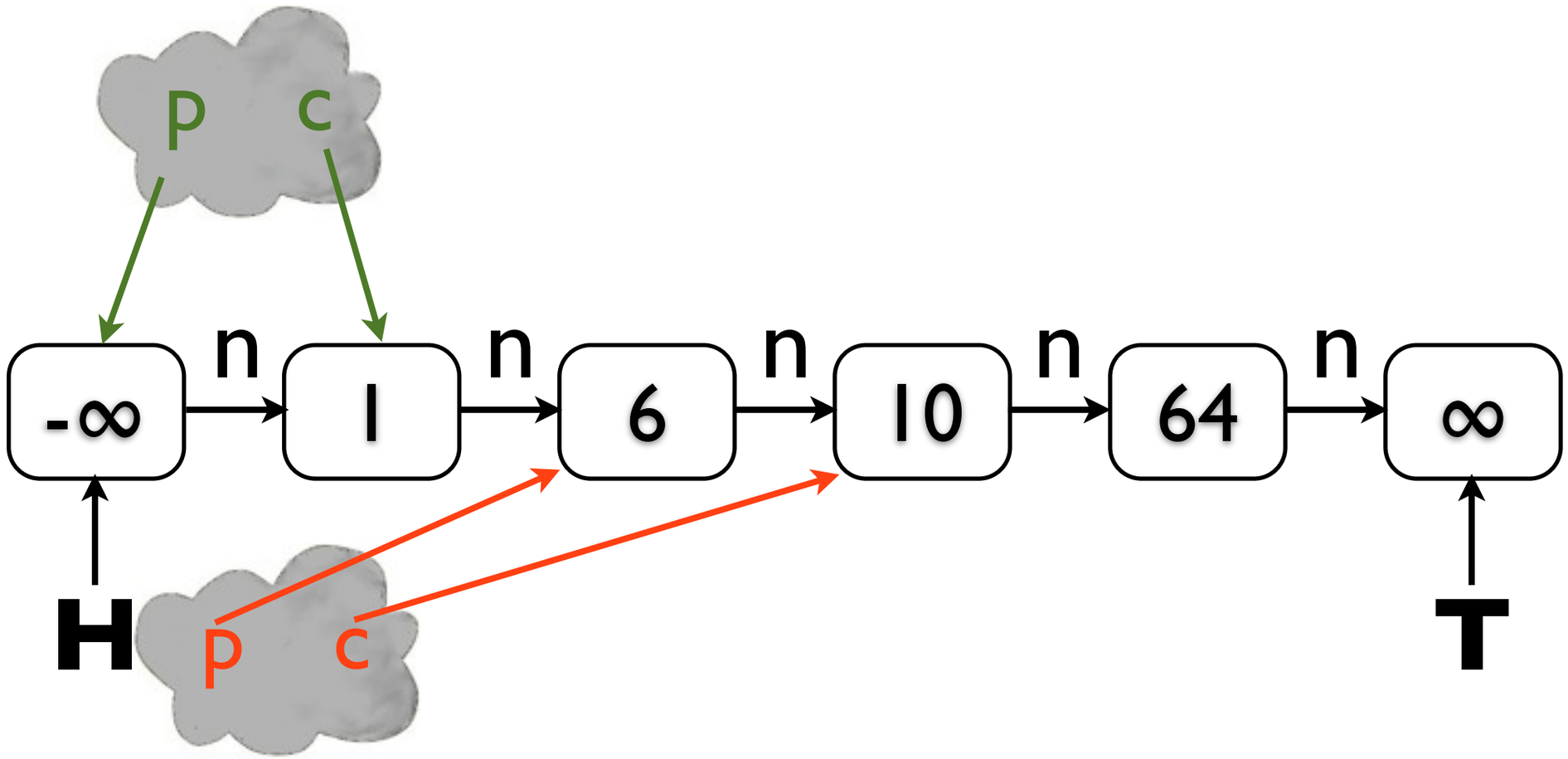
remove(6)



remove(10)

example 1

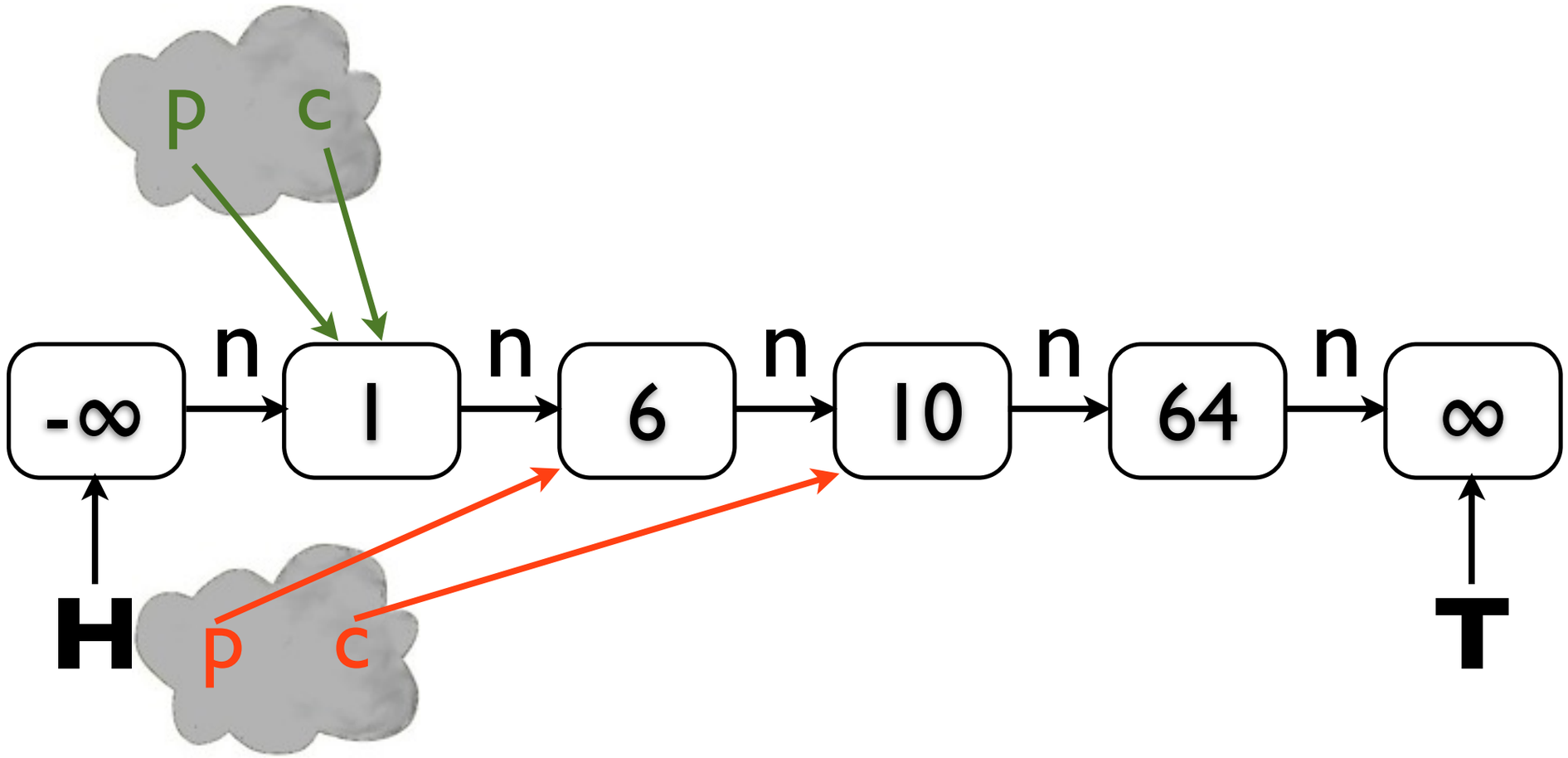
remove(6)



remove(10)

example 1

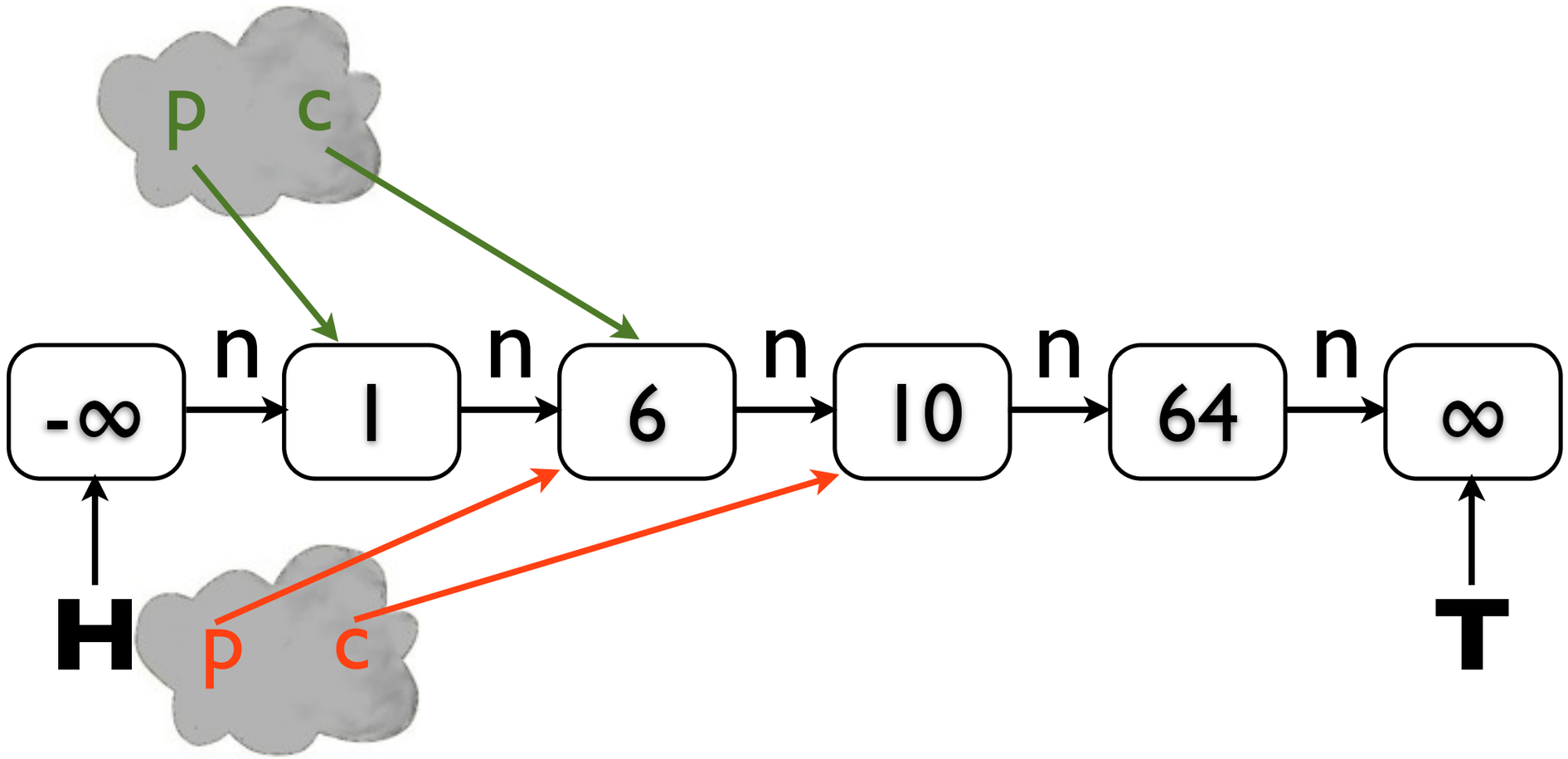
remove(6)



remove(10)

example 1

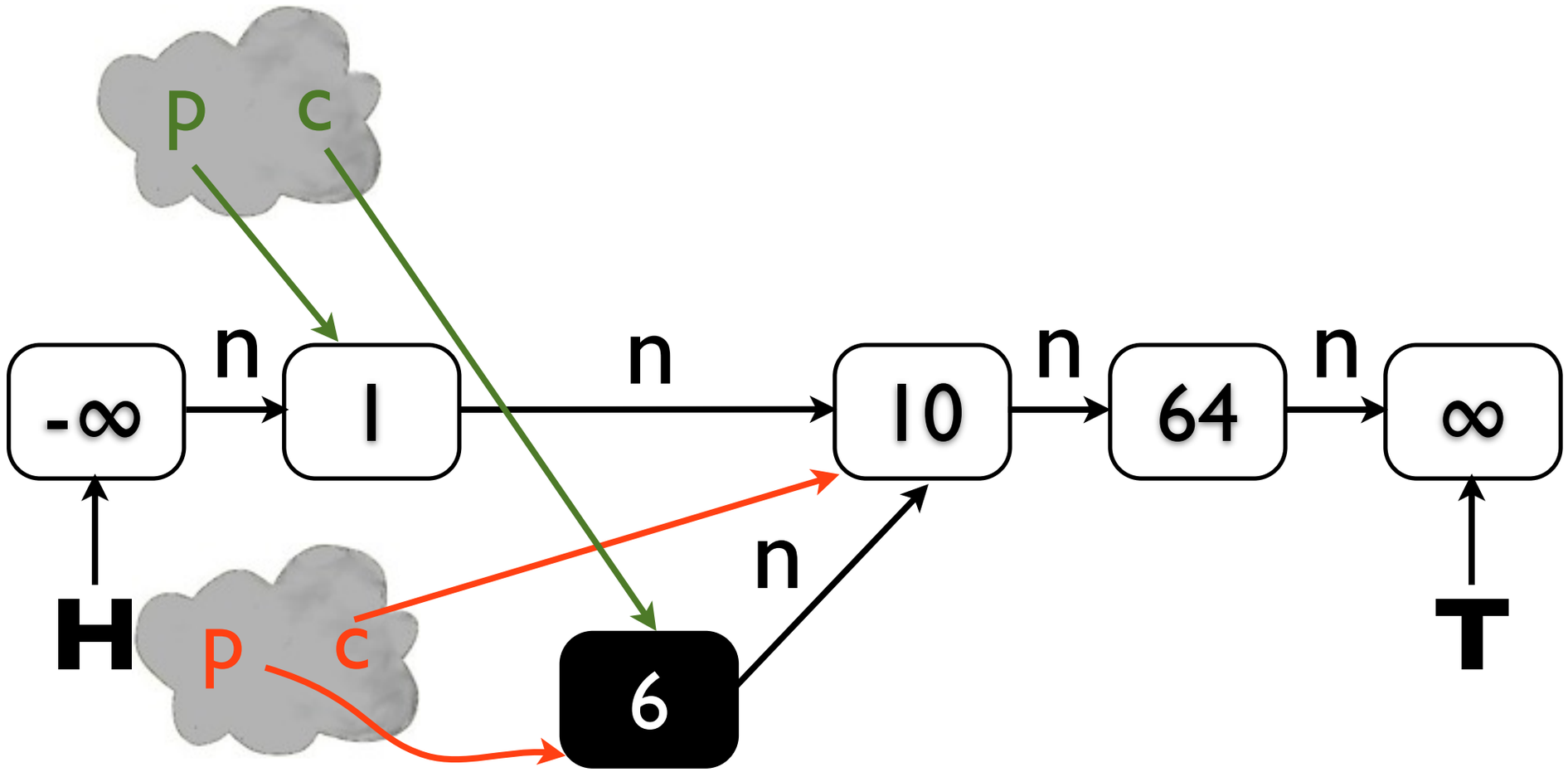
remove(6)



remove(10)

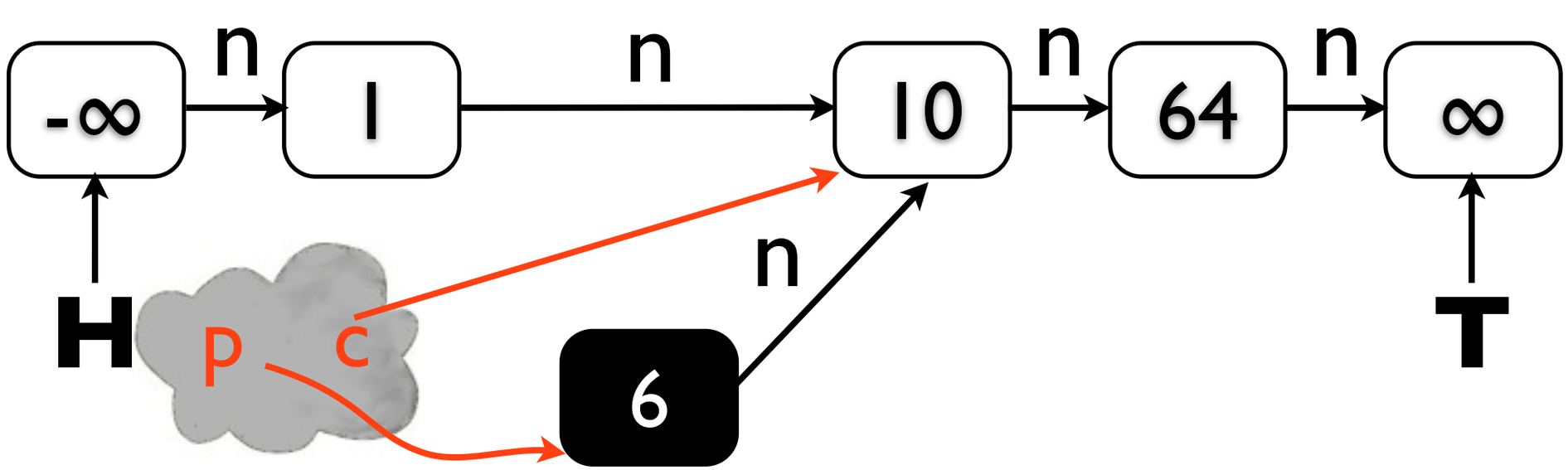
example 1

remove(6)



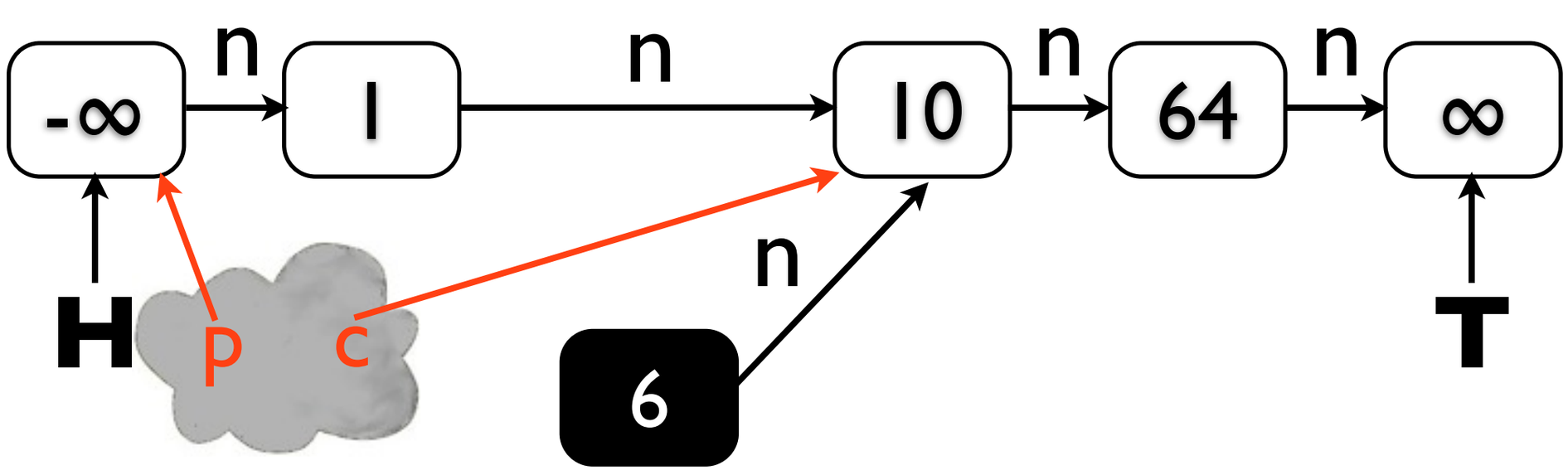
remove(10)

example 1



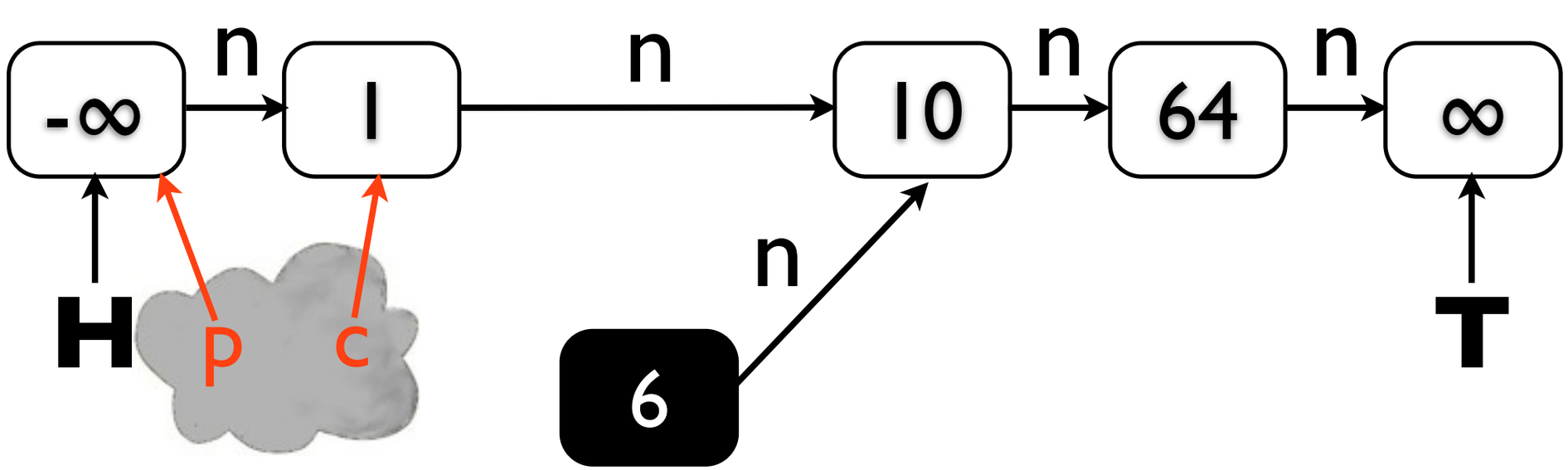
remove(10)

example 1



remove(10)

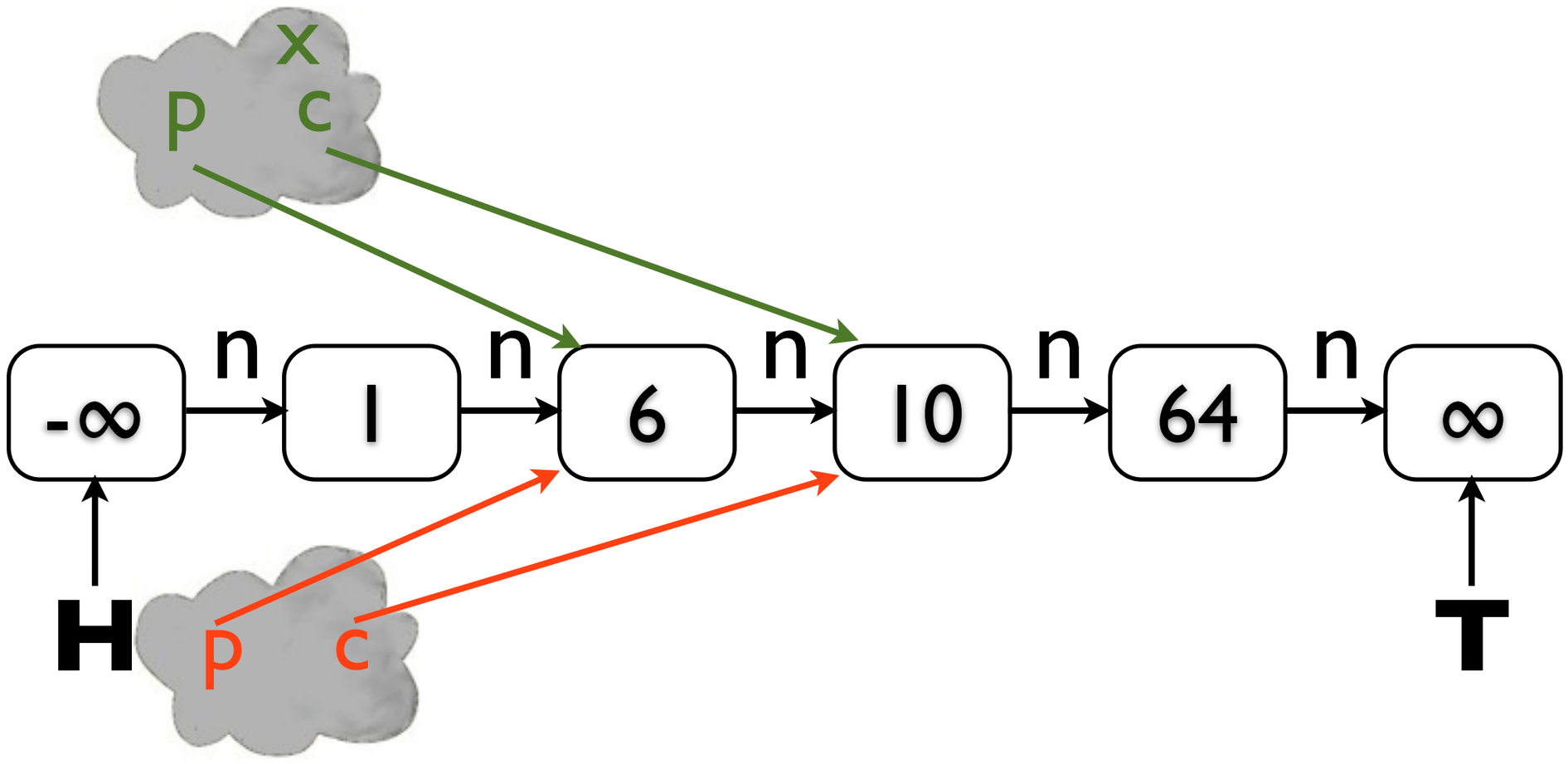
example 1



remove(10)

example 2

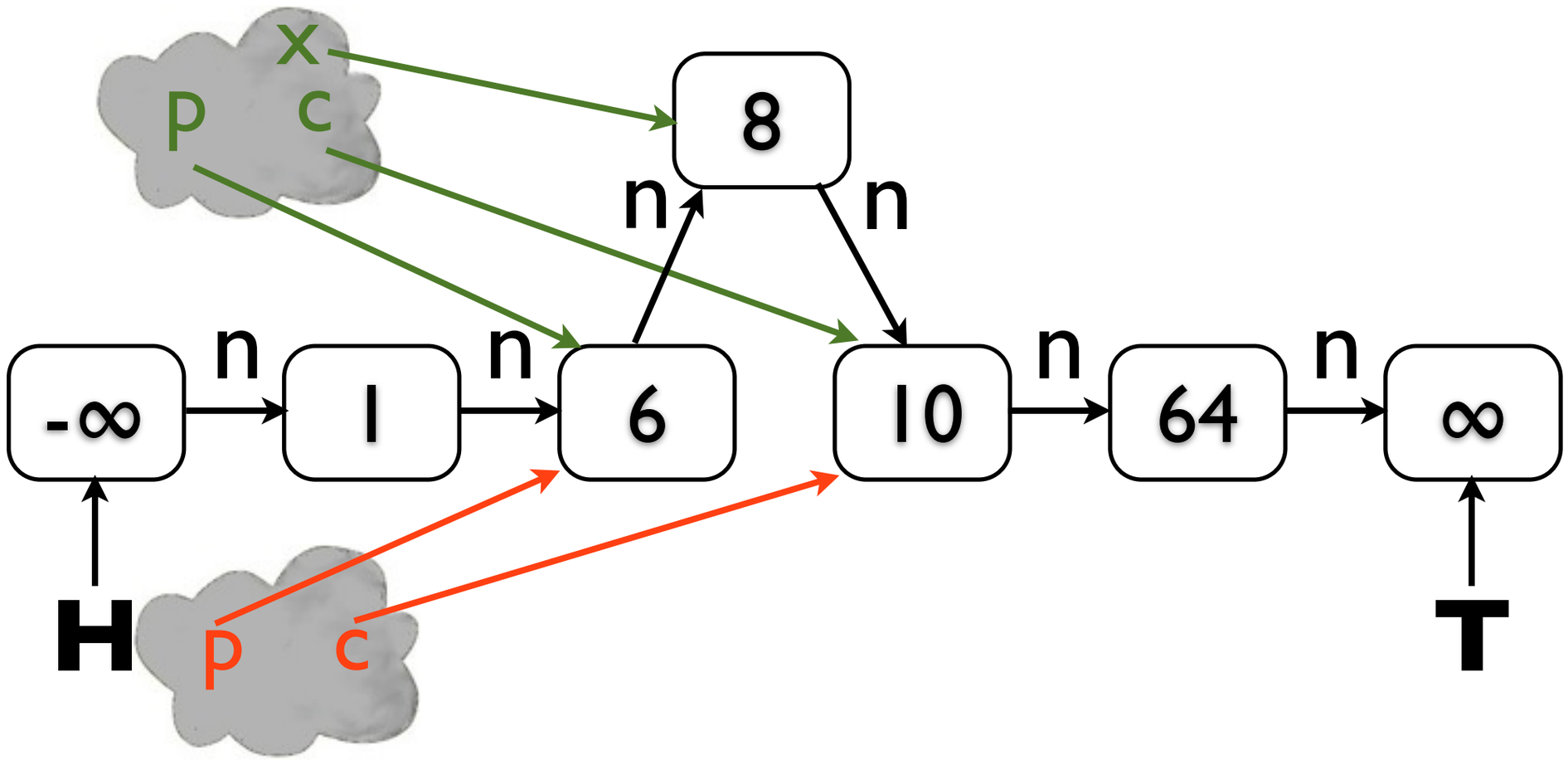
insert(8)



remove(10)

example 2

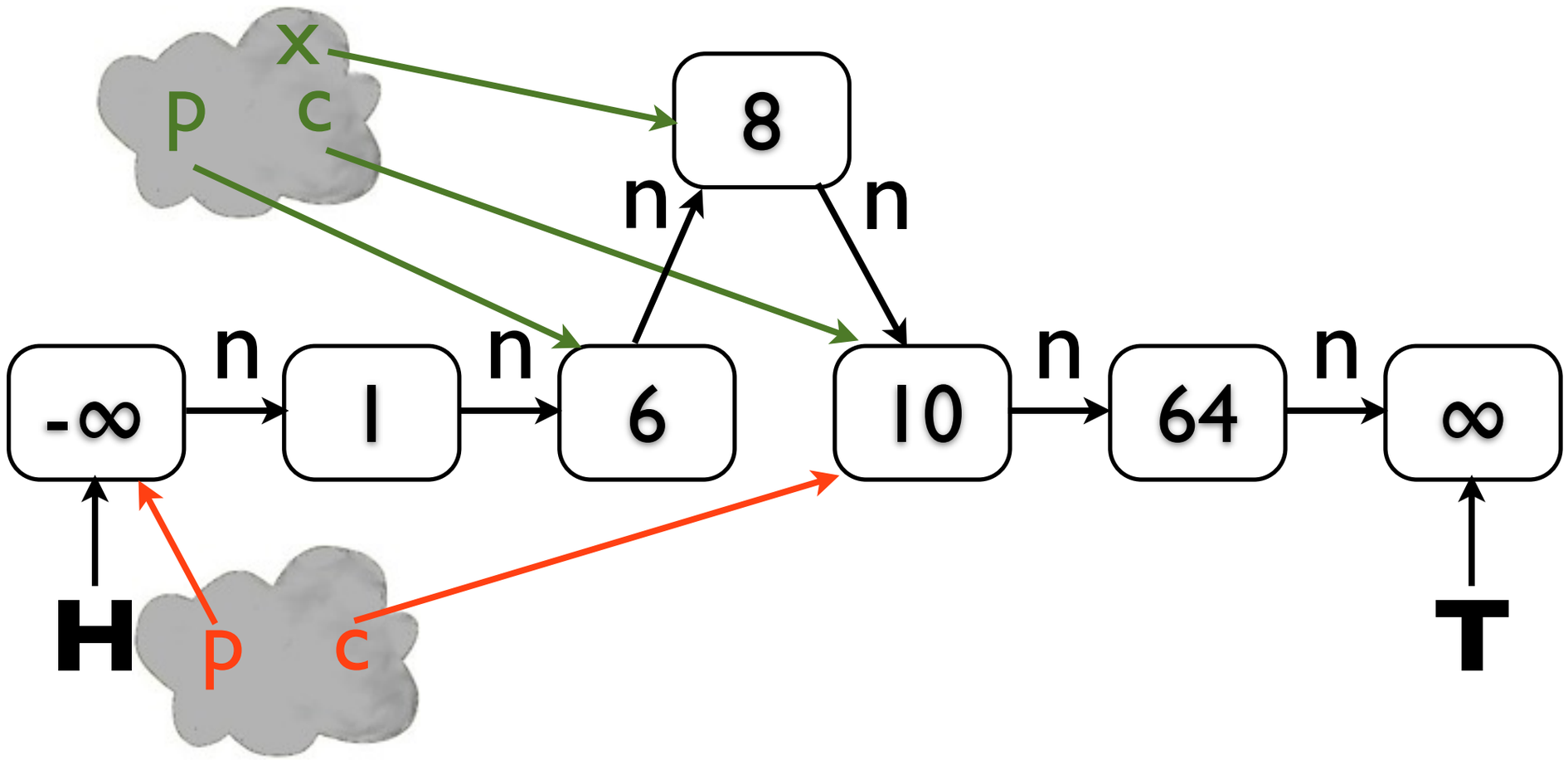
insert(8)



remove(10)

example 2

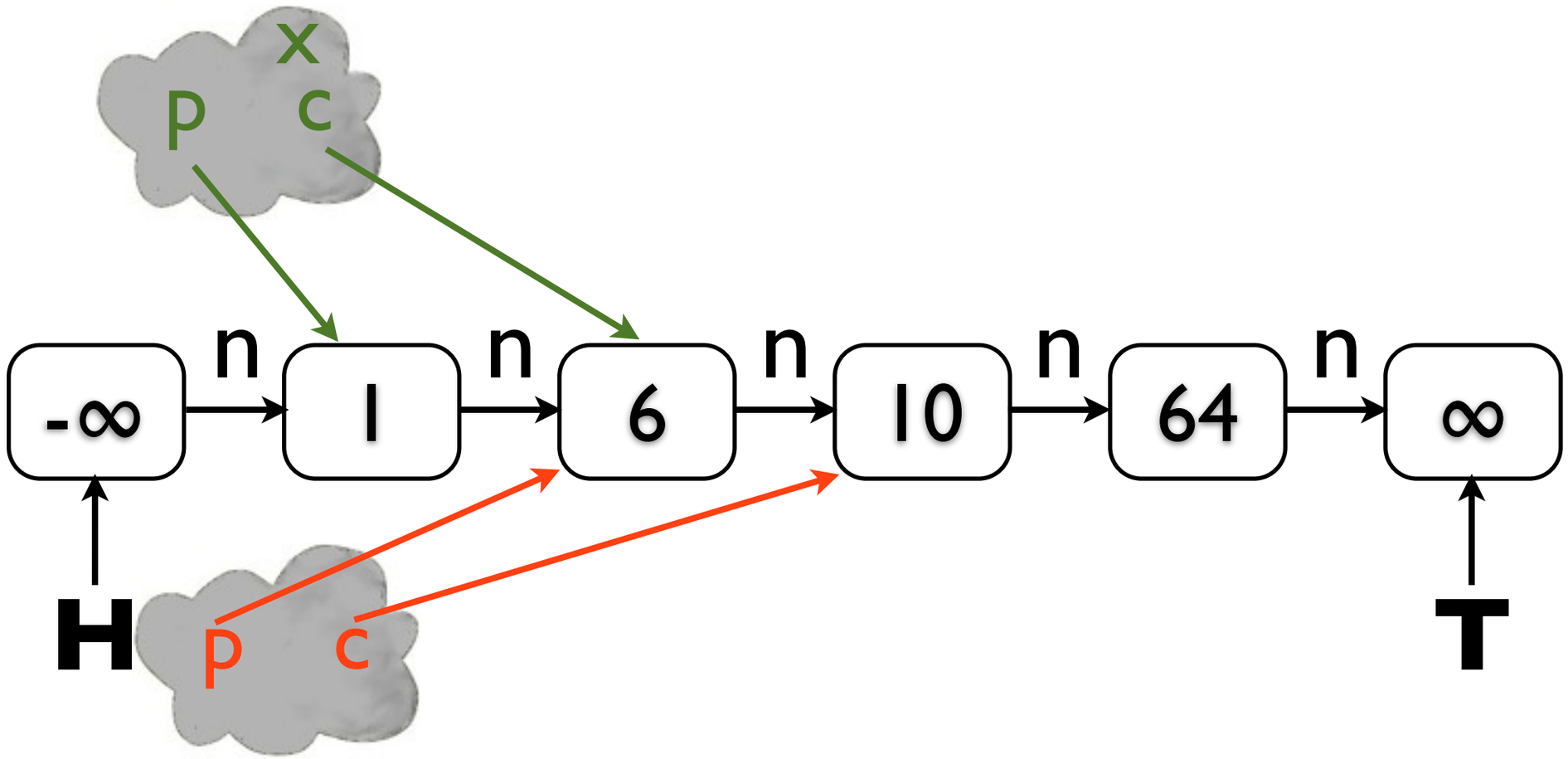
insert(8)



remove(10)

example 3

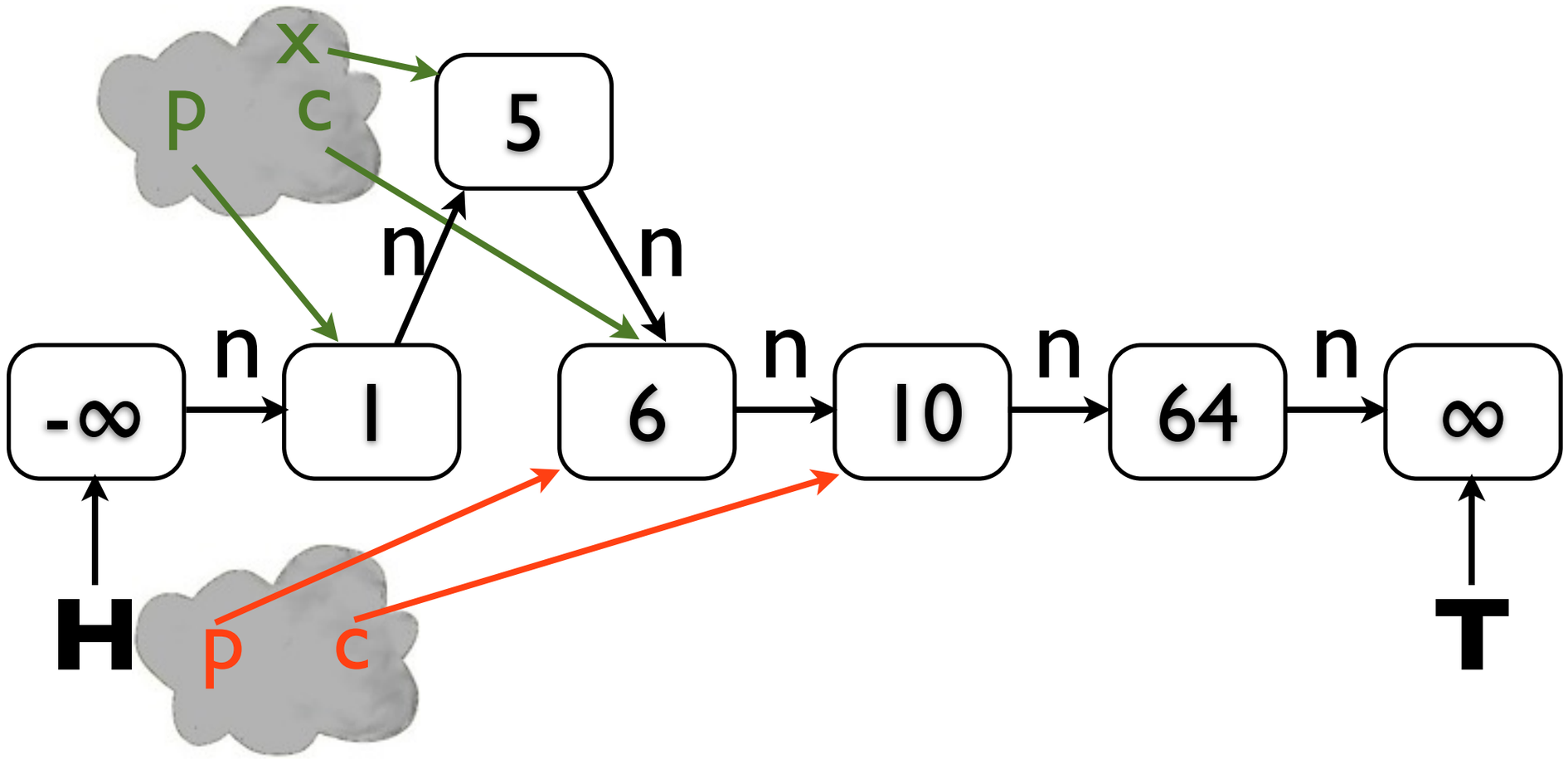
insert(5)



remove(10)

example 3

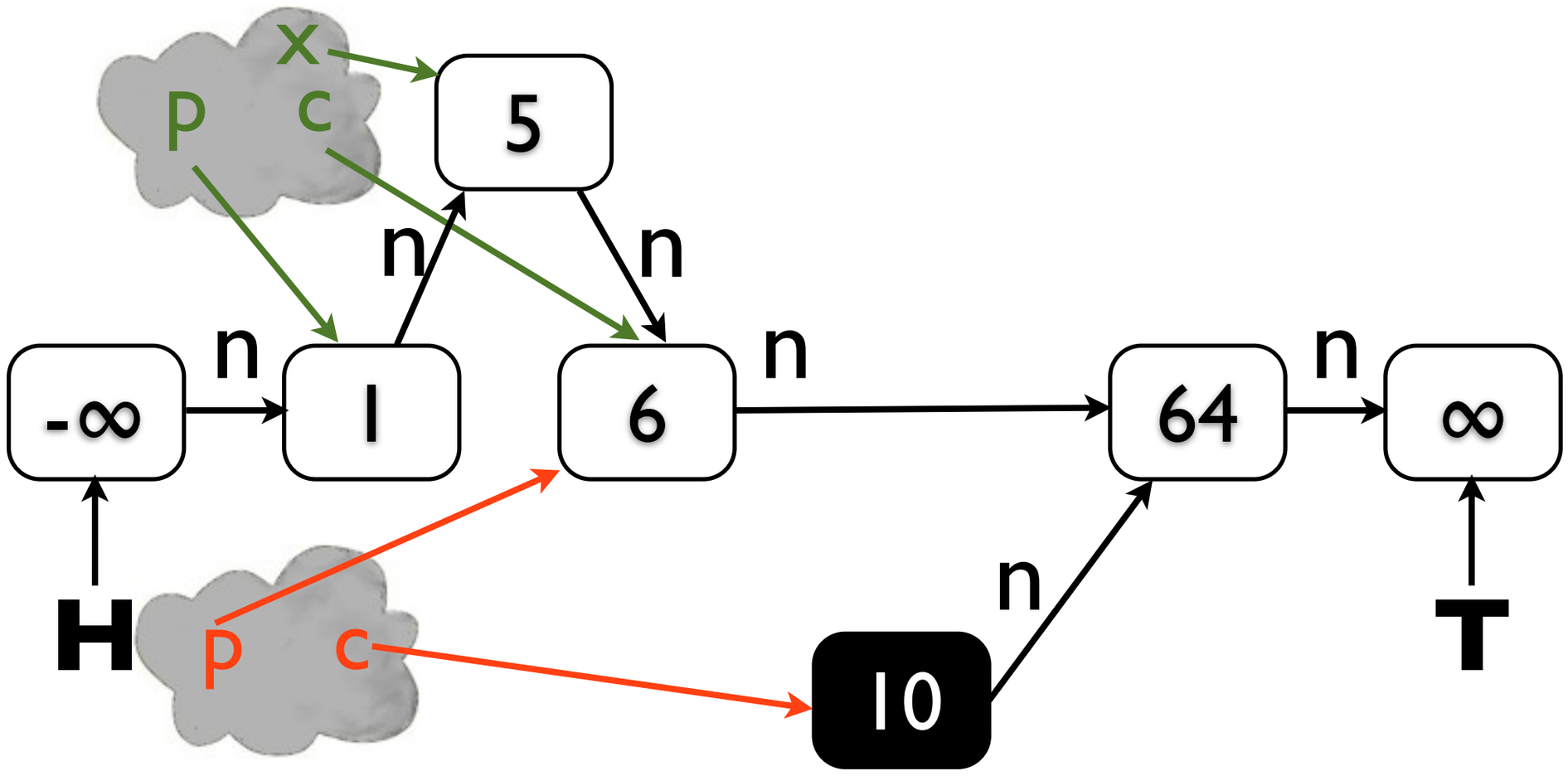
insert(5)



remove(10)

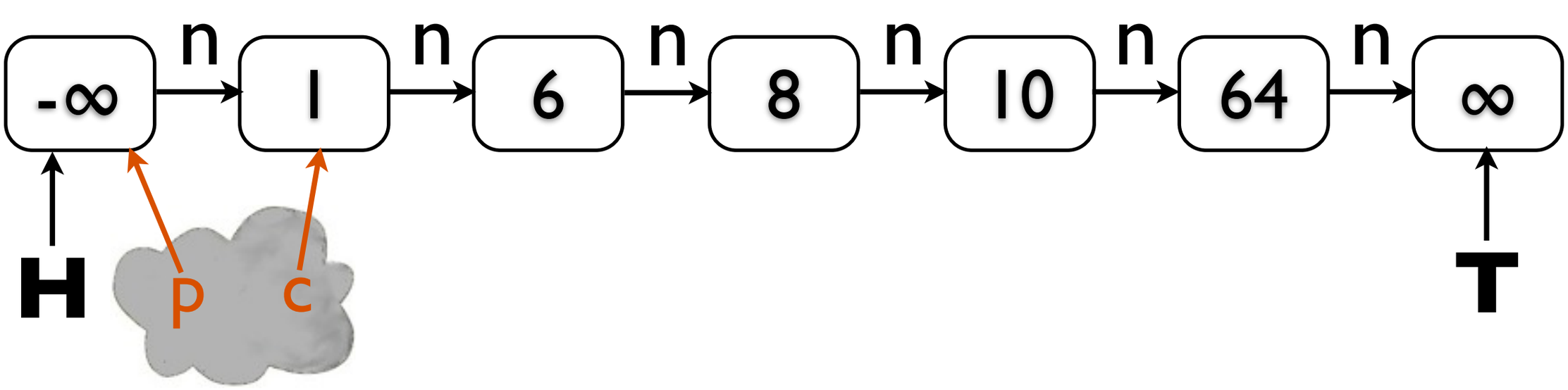
example 3

insert(5)



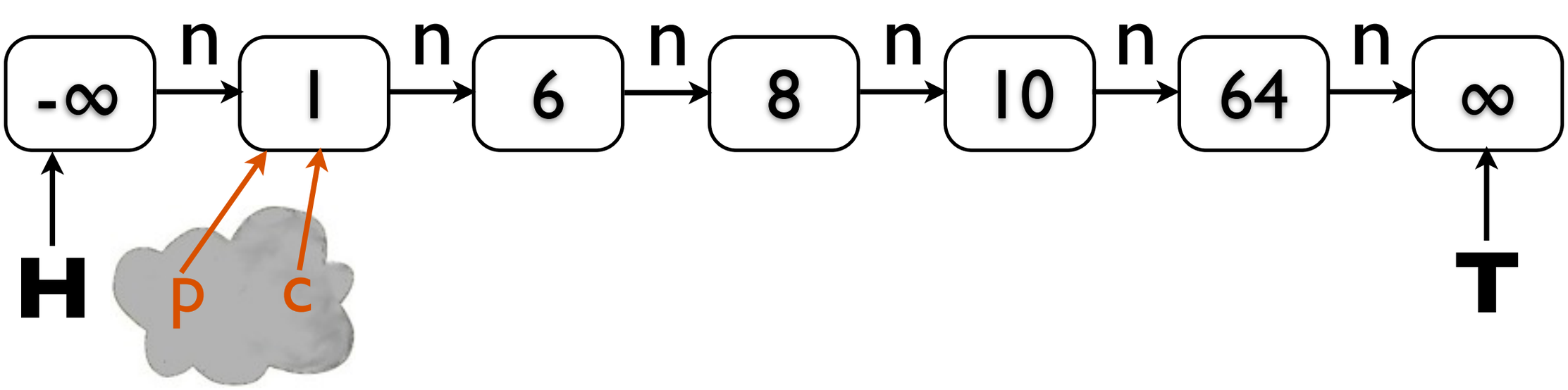
remove(10)

example 4



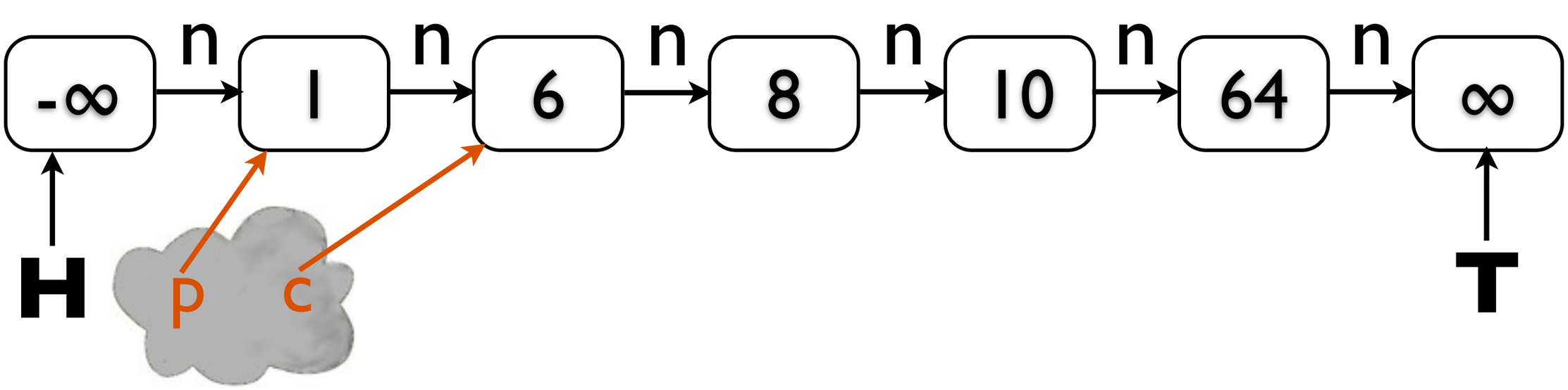
contains(8)

example 4



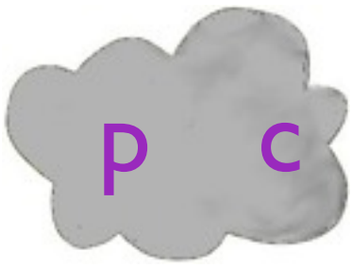
contains(8)

example 4

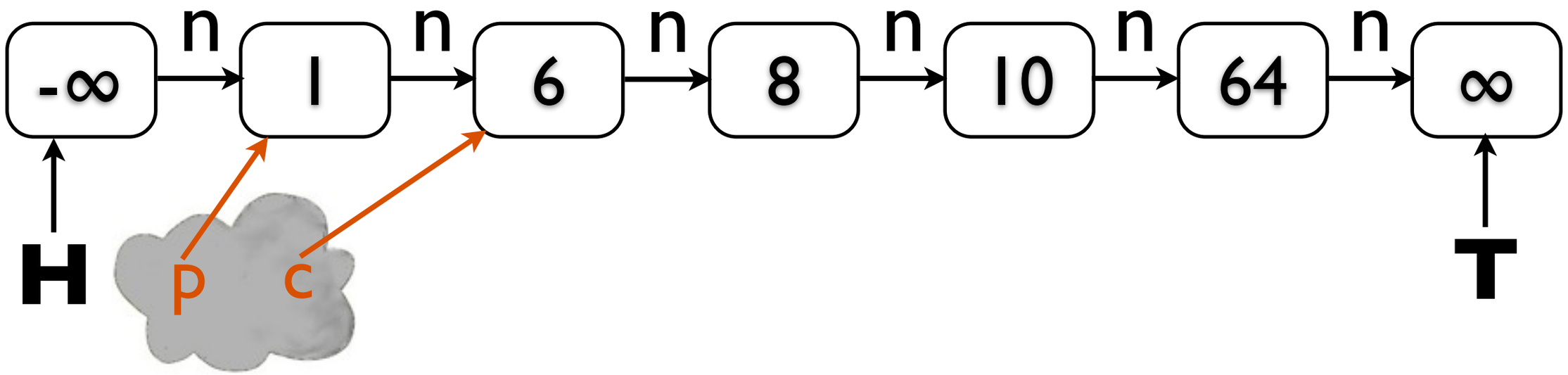


contains(8)

example 4

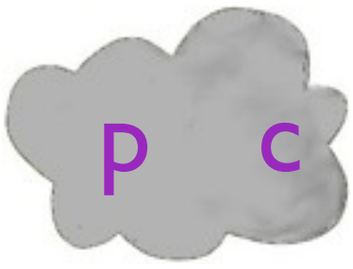


remove(8); remove(6); remove(10); add(8)

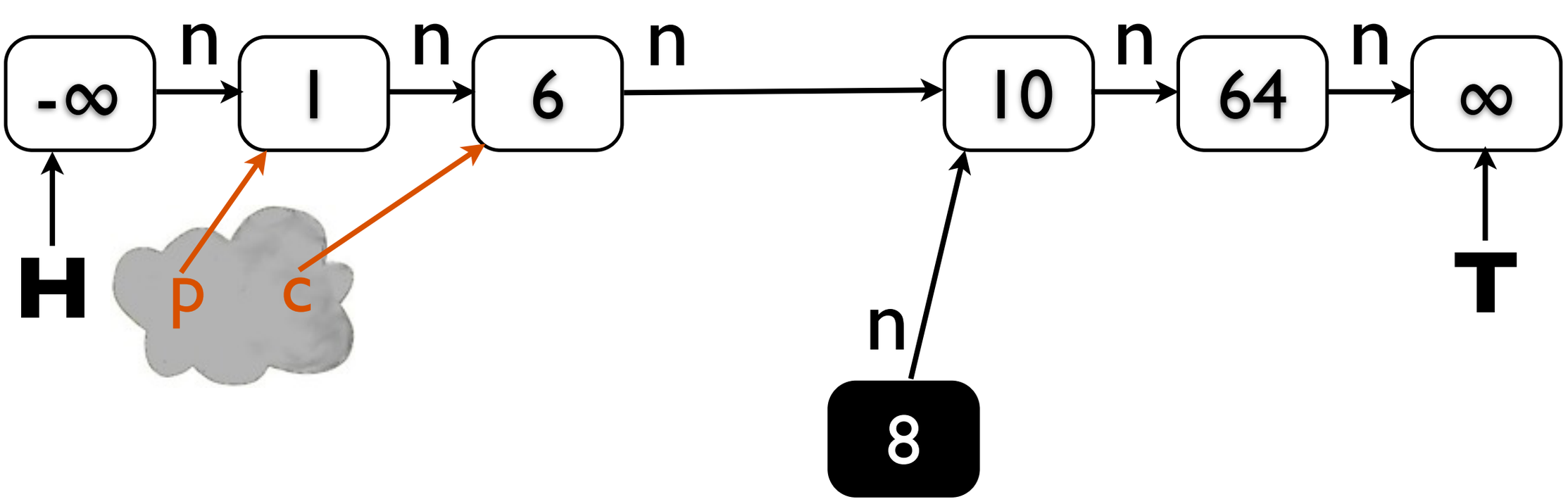


contains(8)

example 4



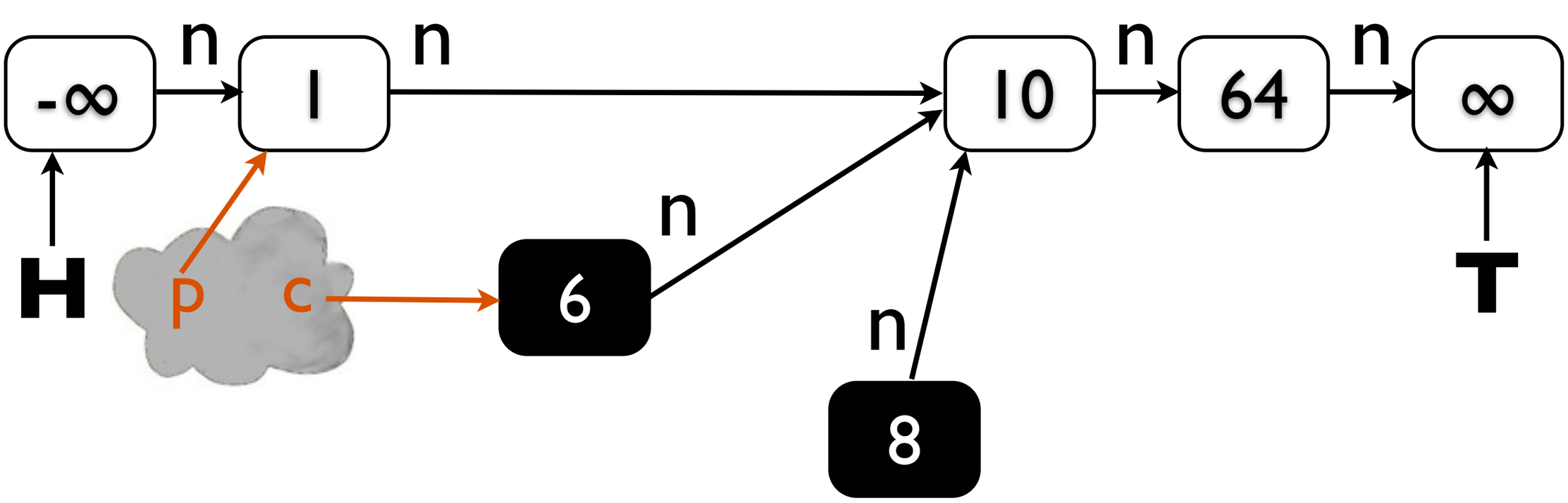
remove(8); remove(6); remove(10); add(8)



contains(8)

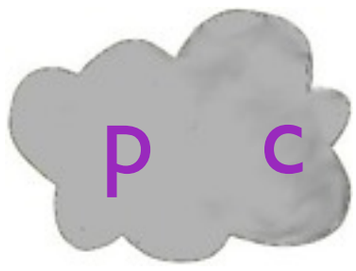
example 4

p c remove(8); remove(6); remove(10); add(8)

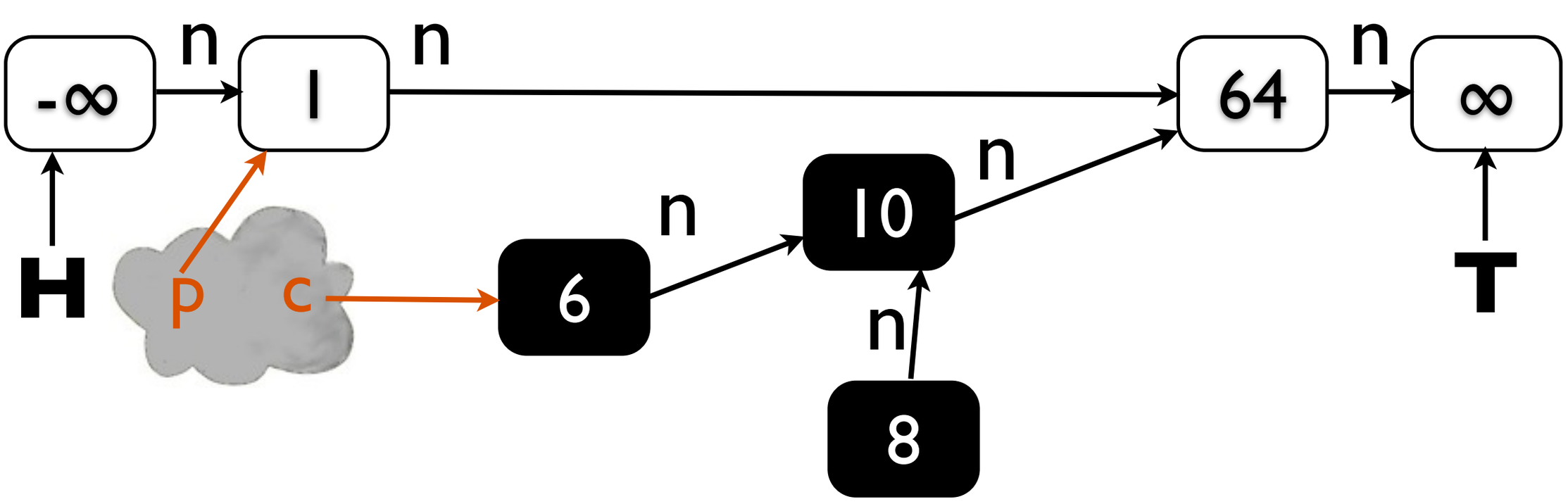


contains(8)

example 4

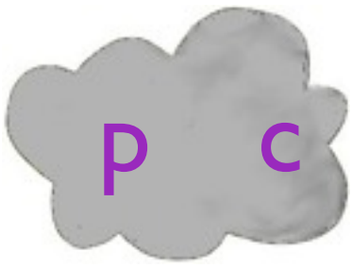


remove(8); remove(6); remove(10); add(8)

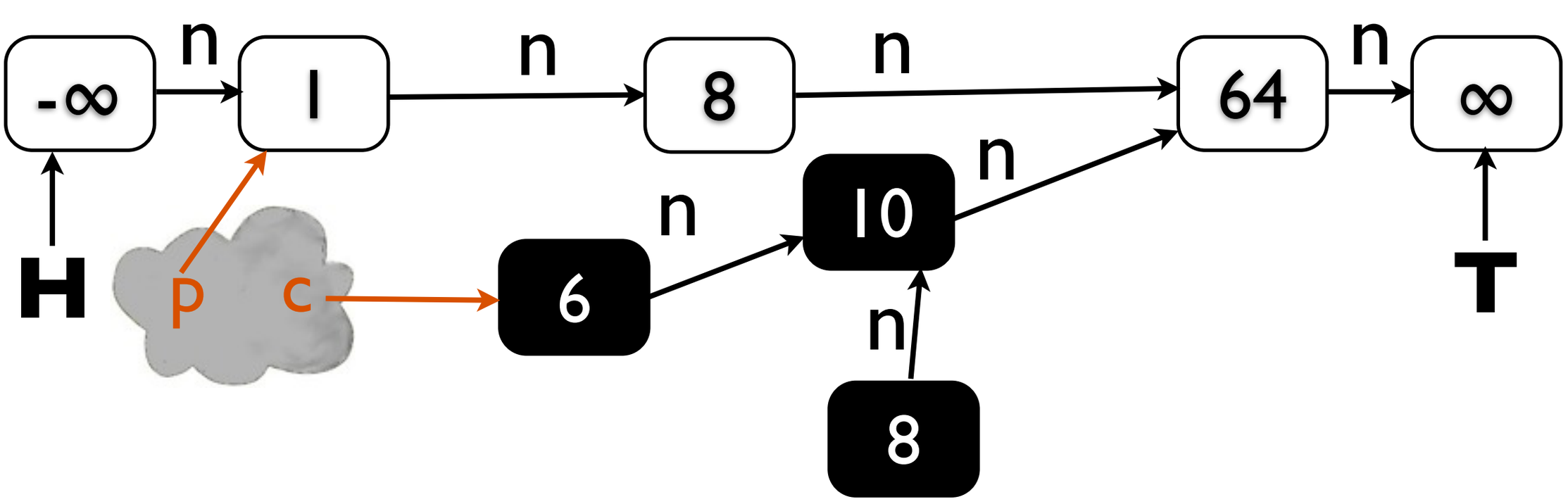


contains(8)

example 4

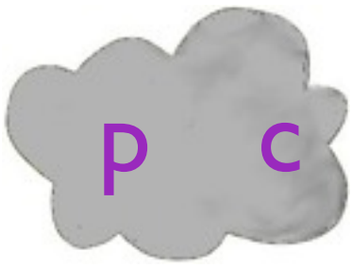


remove(8); remove(6); remove(10); add(8)

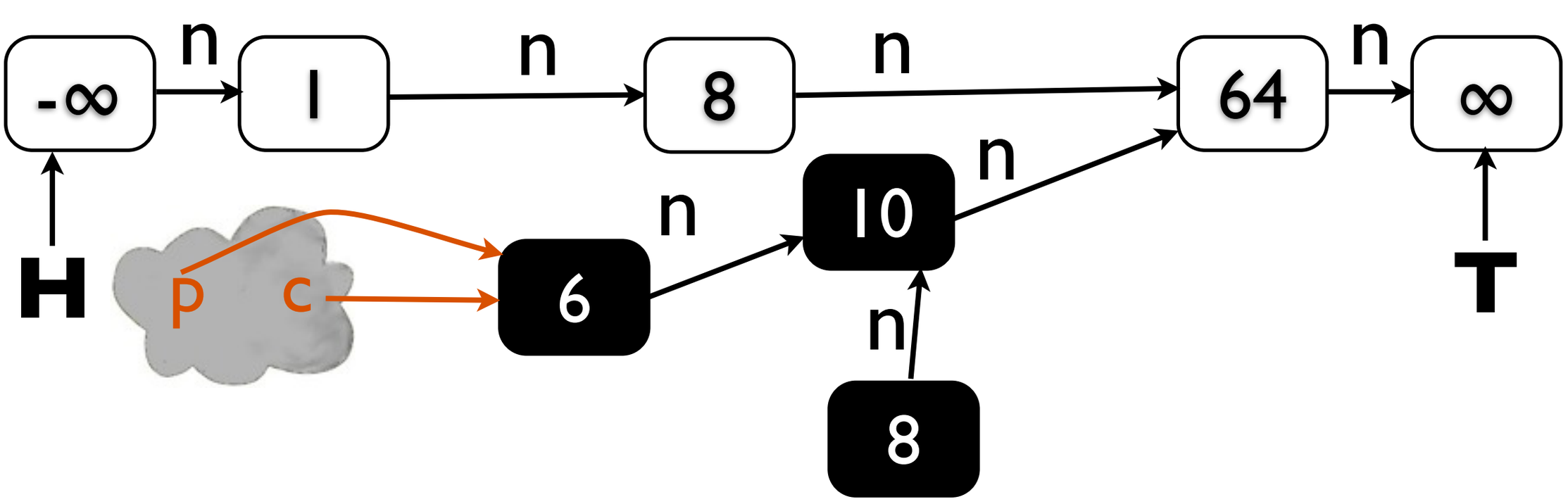


contains(8)

example 4

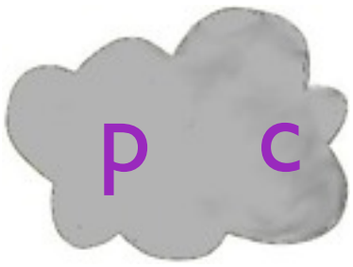


remove(8); remove(6); remove(10); add(8)

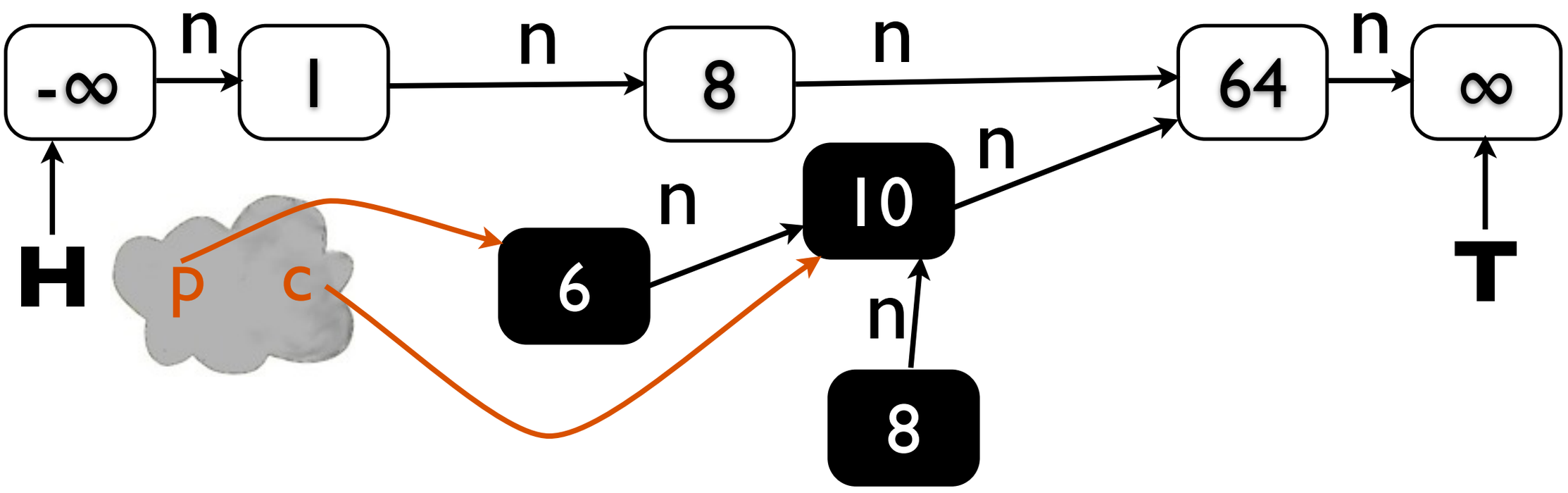


contains(8)

example 4



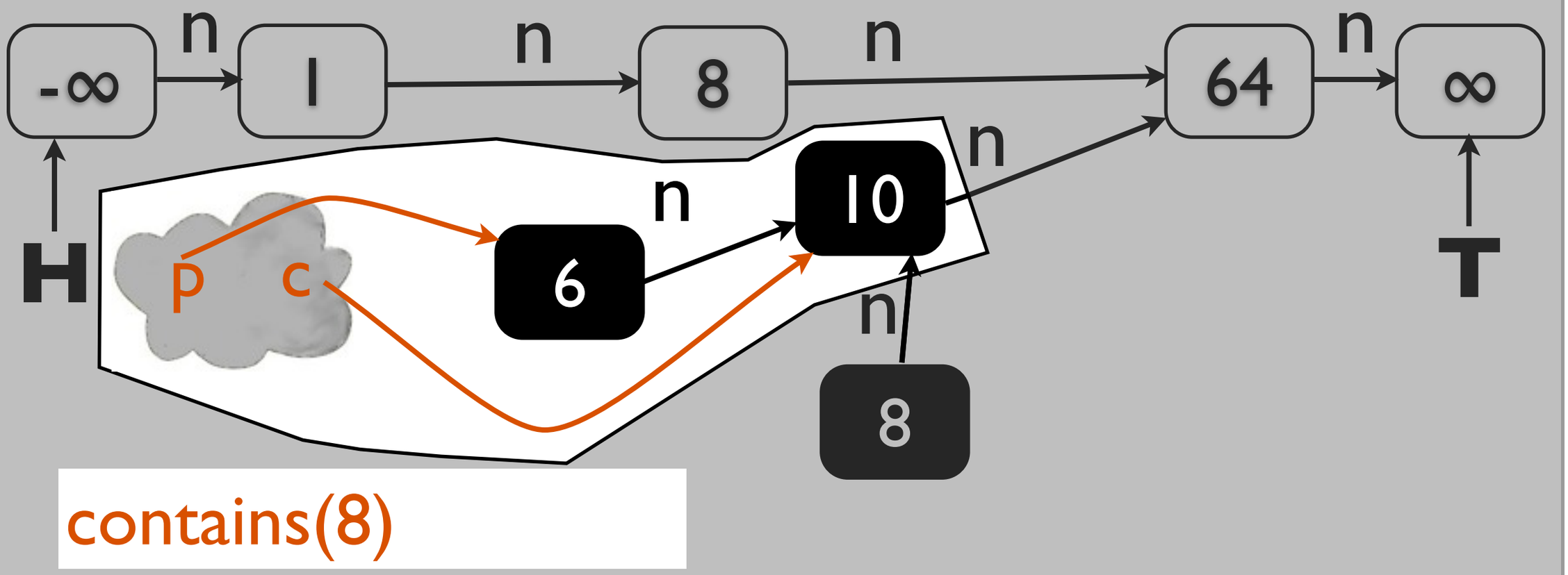
remove(8); remove(6); remove(10); add(8)



contains(8)

example 4

remove(8); remove(6); remove(10); add(8)

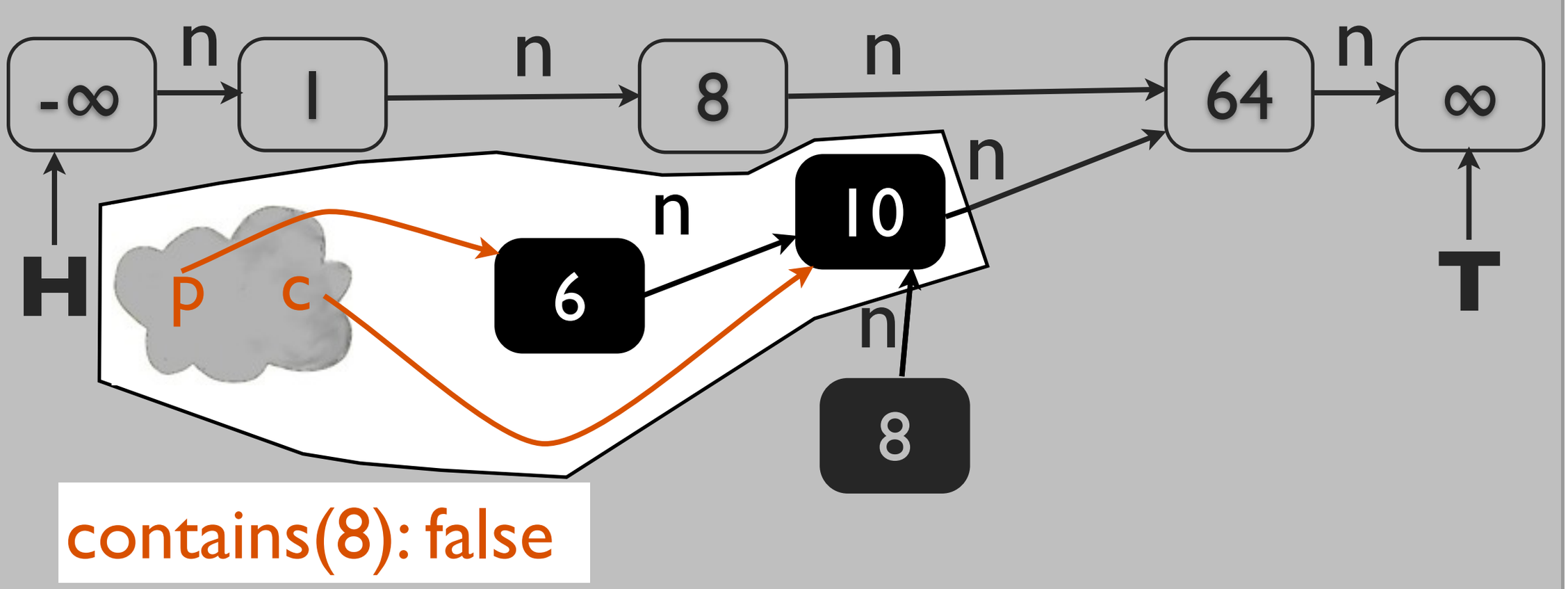


contains(8)

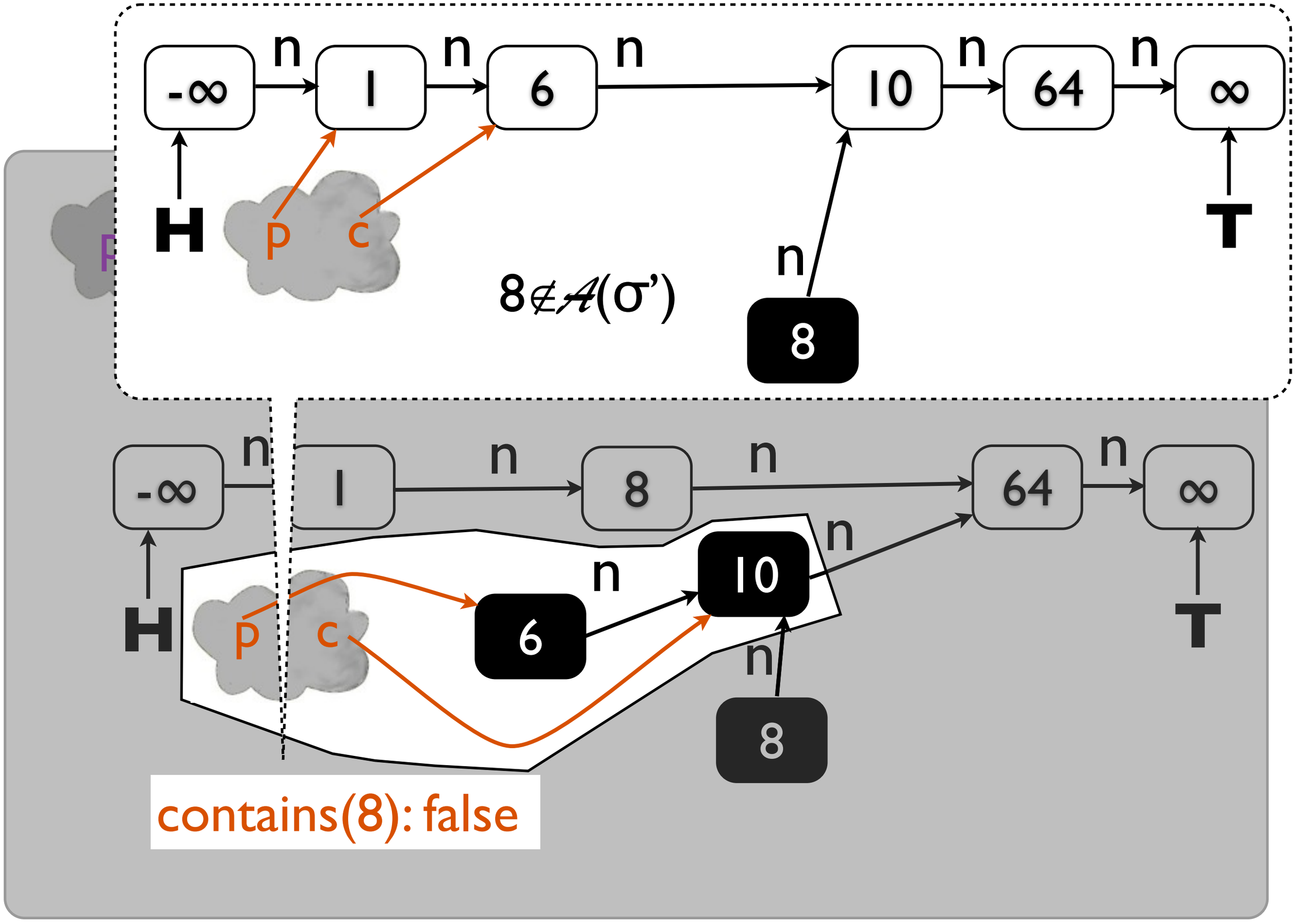
example 4

remove(8); remove(6); remove(10); add(8)

$8 \in \mathcal{A}(\sigma)$



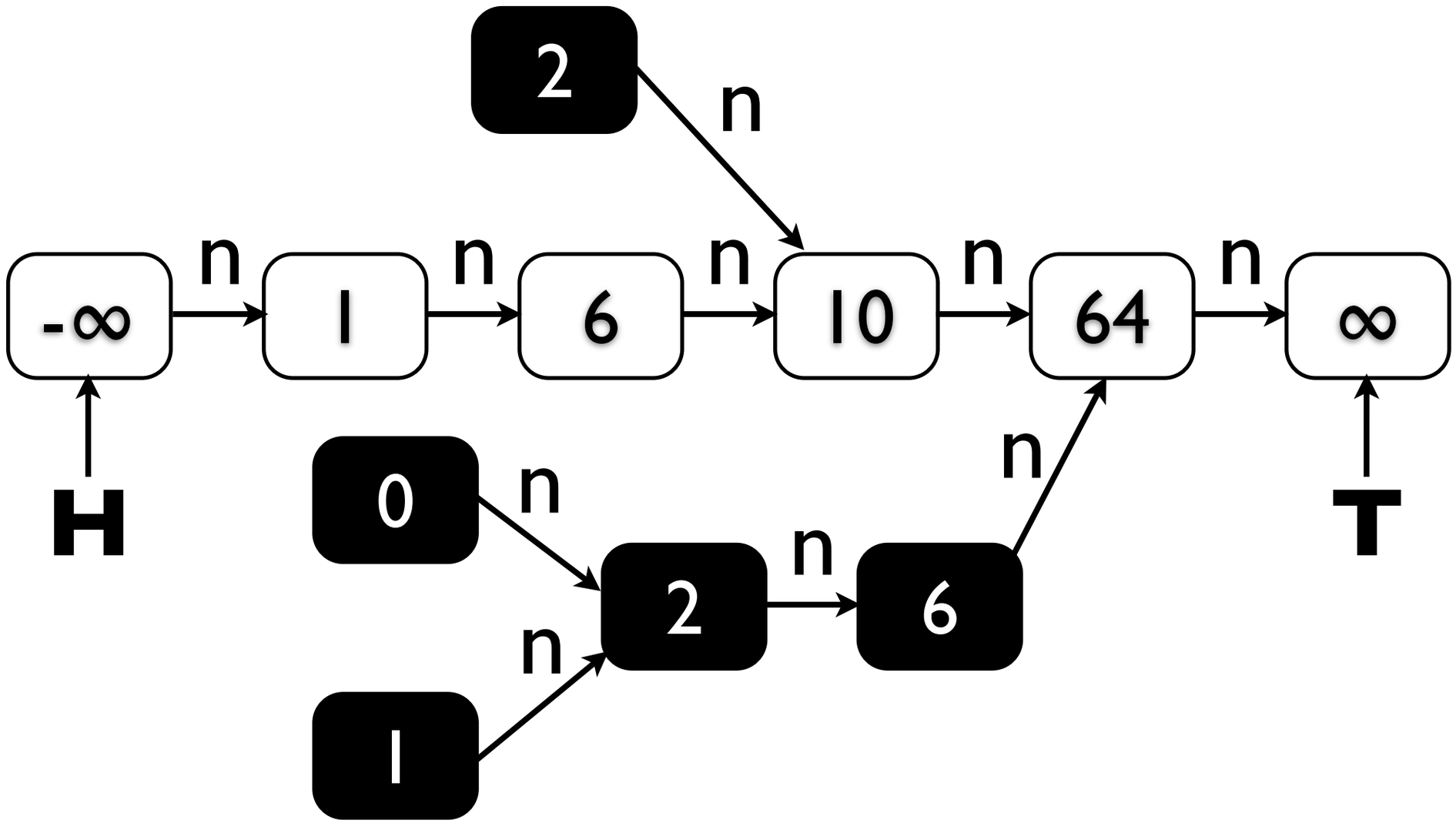
contains(8): false


-∞
n
1
n
6
n
10
n
64
n
∞
H
P
c
T
8∉𝒜(σ')
n
8
-∞
n
1
n
8
n
64
n
∞
H
p
c
6
n
10
n
n
8
contains(8): false

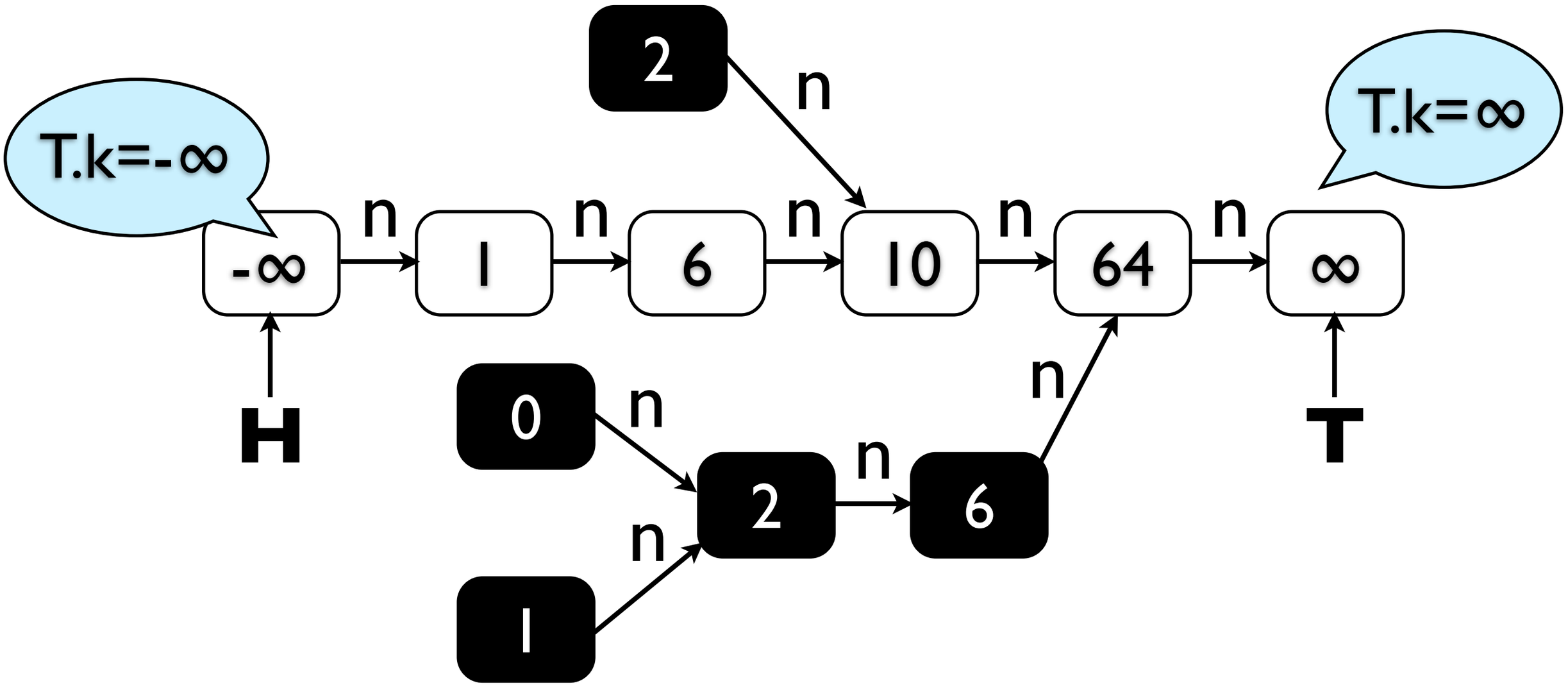
proof strategy

- program logic proof local-protocol preserves global invariant (thread-local proof)
- mathematical proof on good traces
  - not about algorithms / programs
- hindsight good traces allows to conclude properties of past states from current state
- local window concluded properties provide (enough) information for linearizability

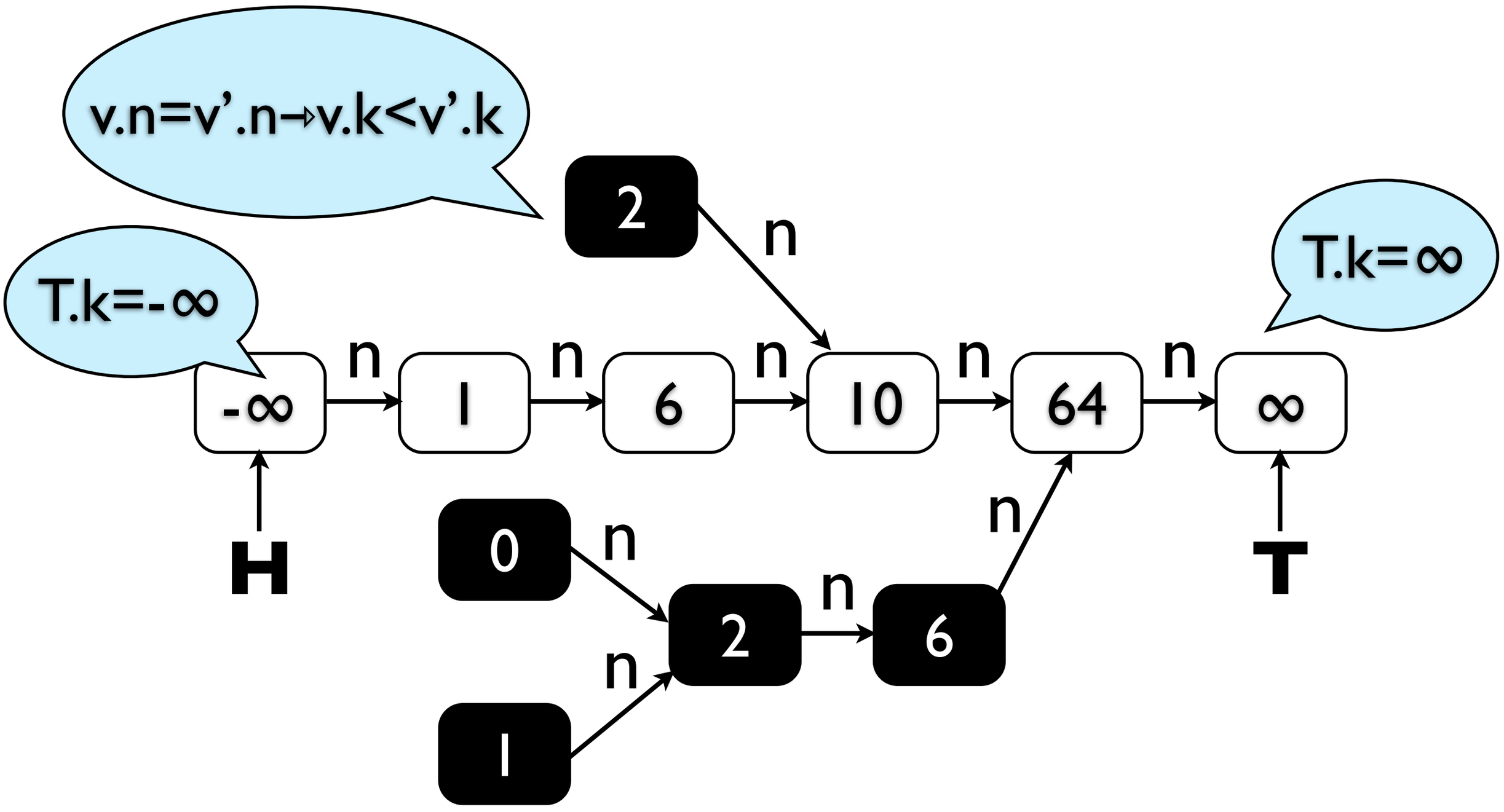
global state invariant



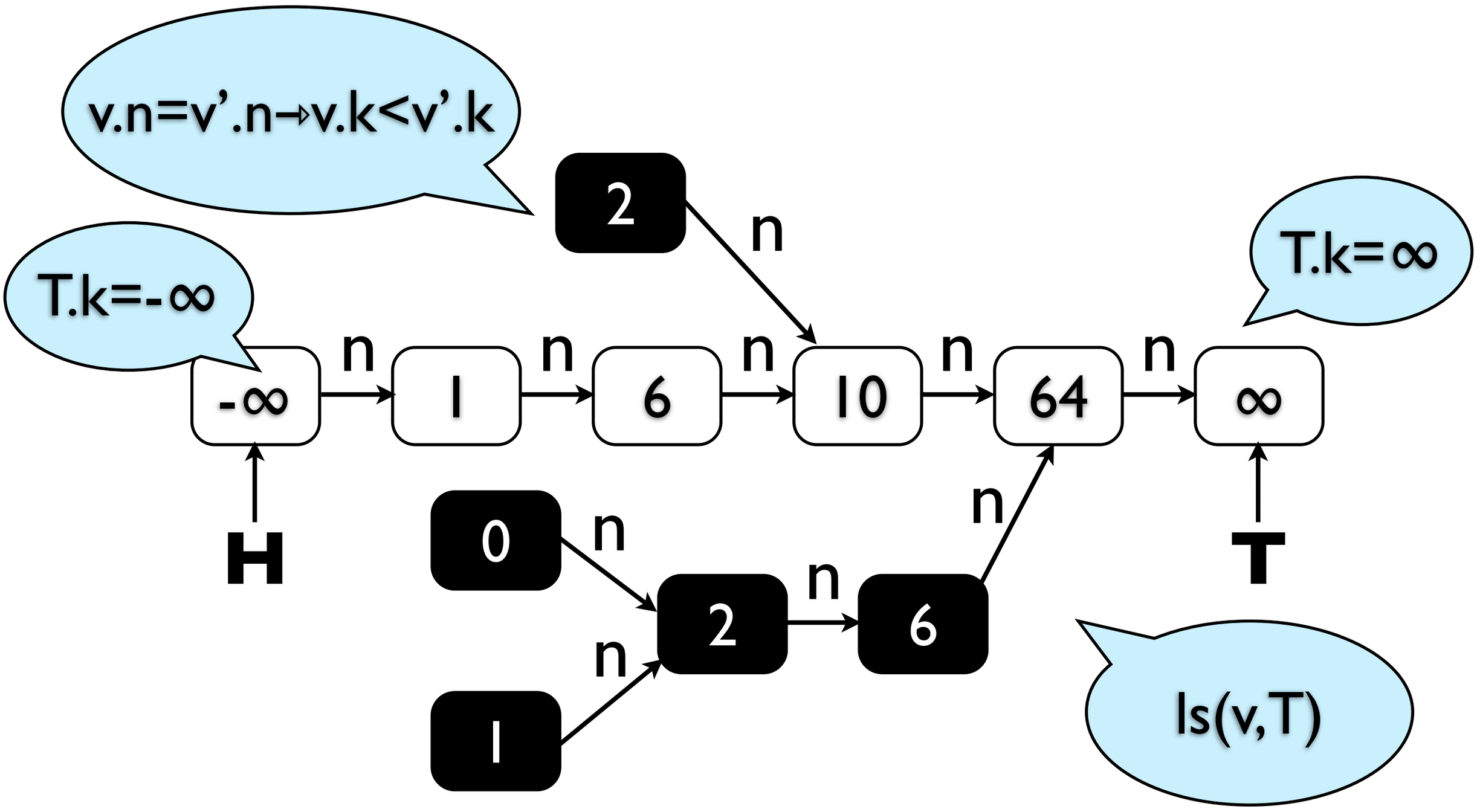
global state invariant



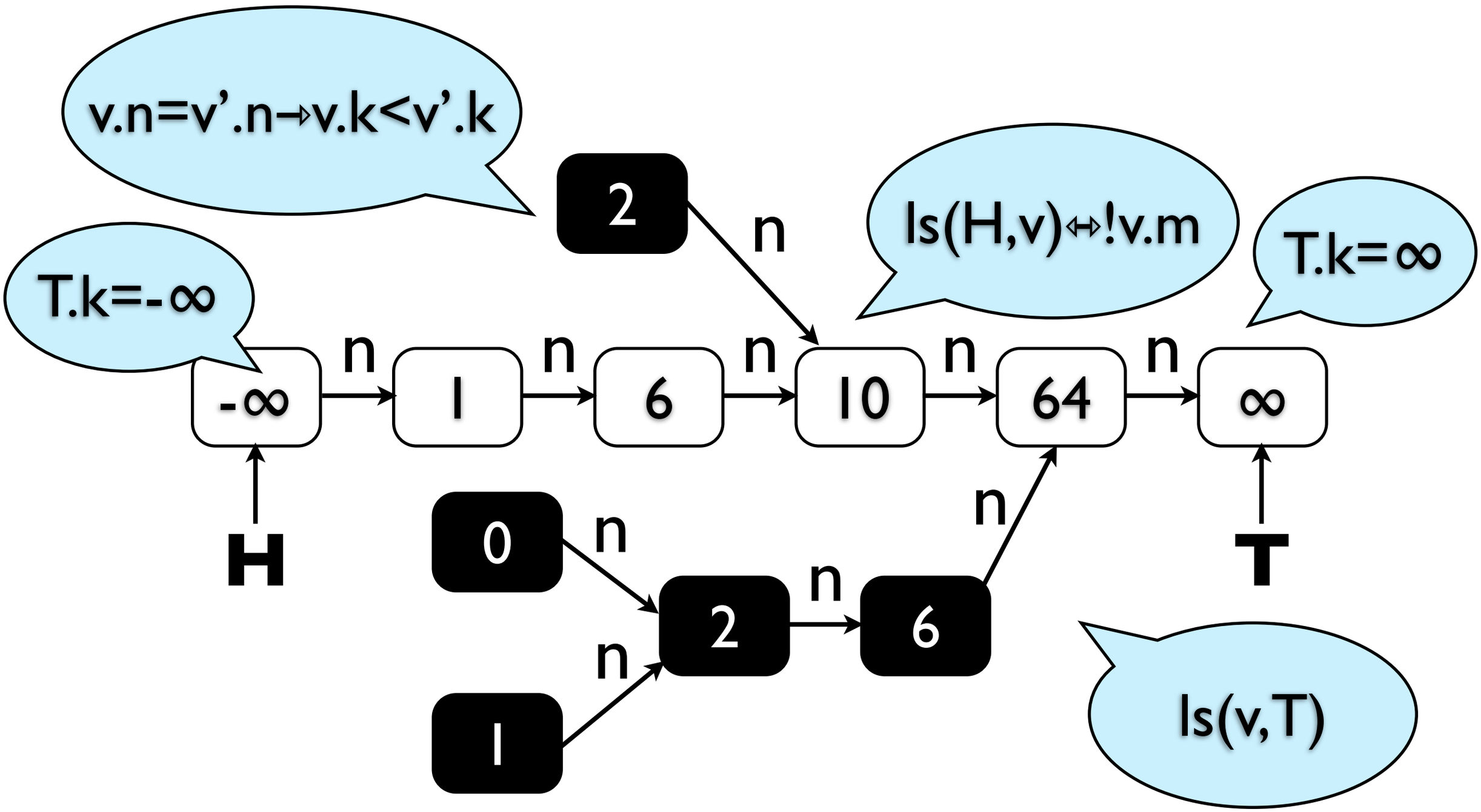
global state invariant



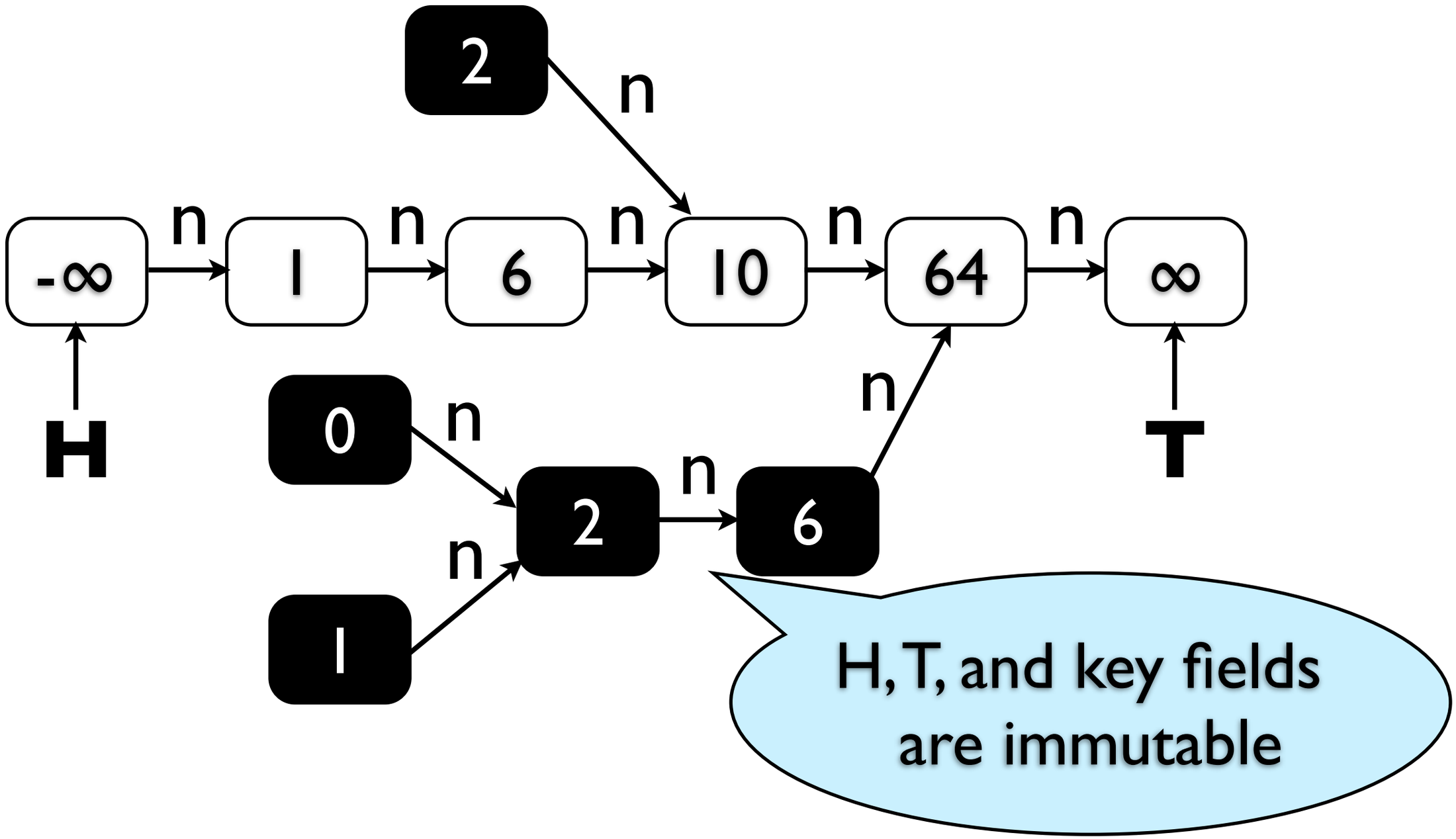
global state invariant



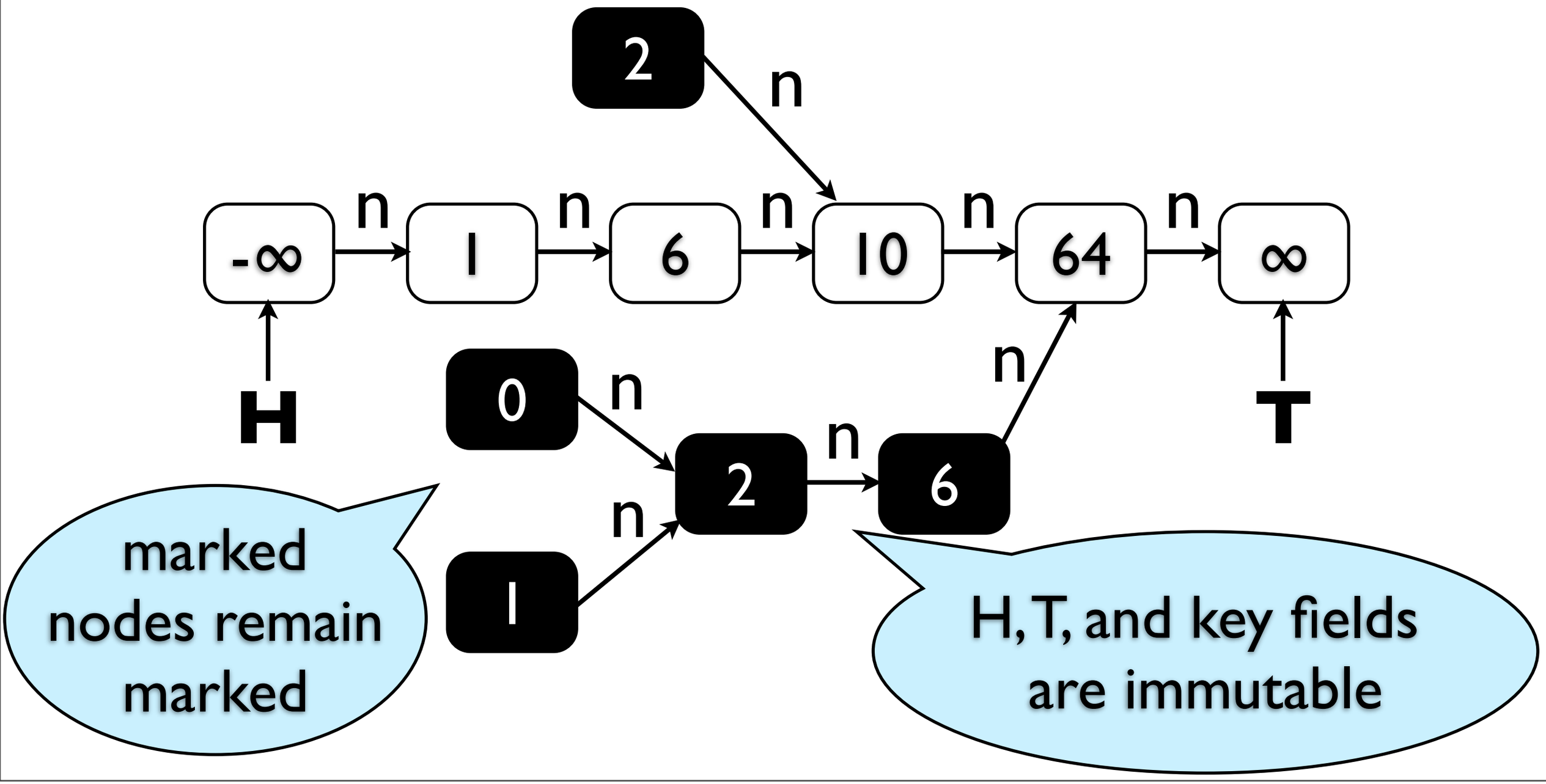
global state invariant



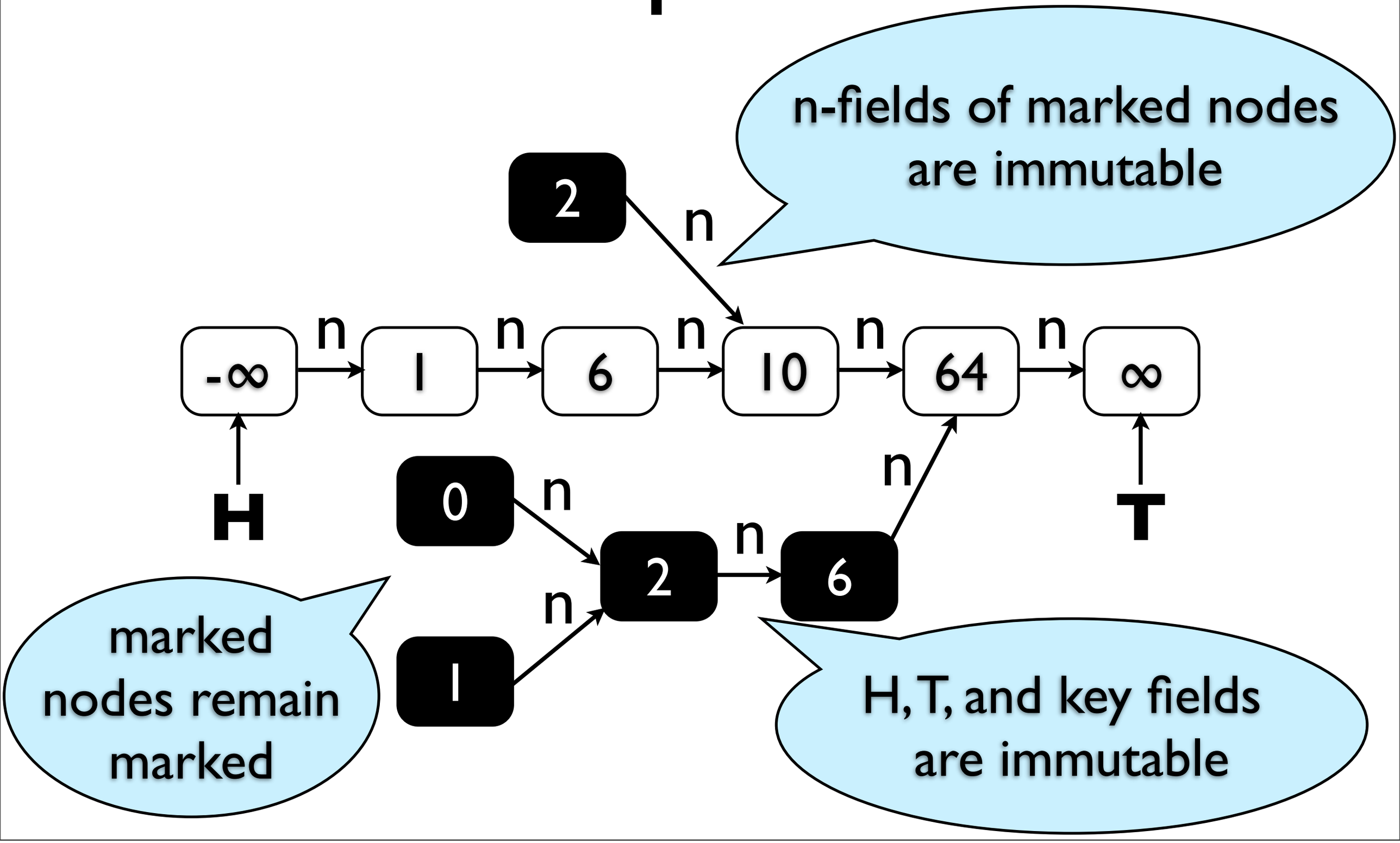
local step invariant



local step invariant



local step invariant

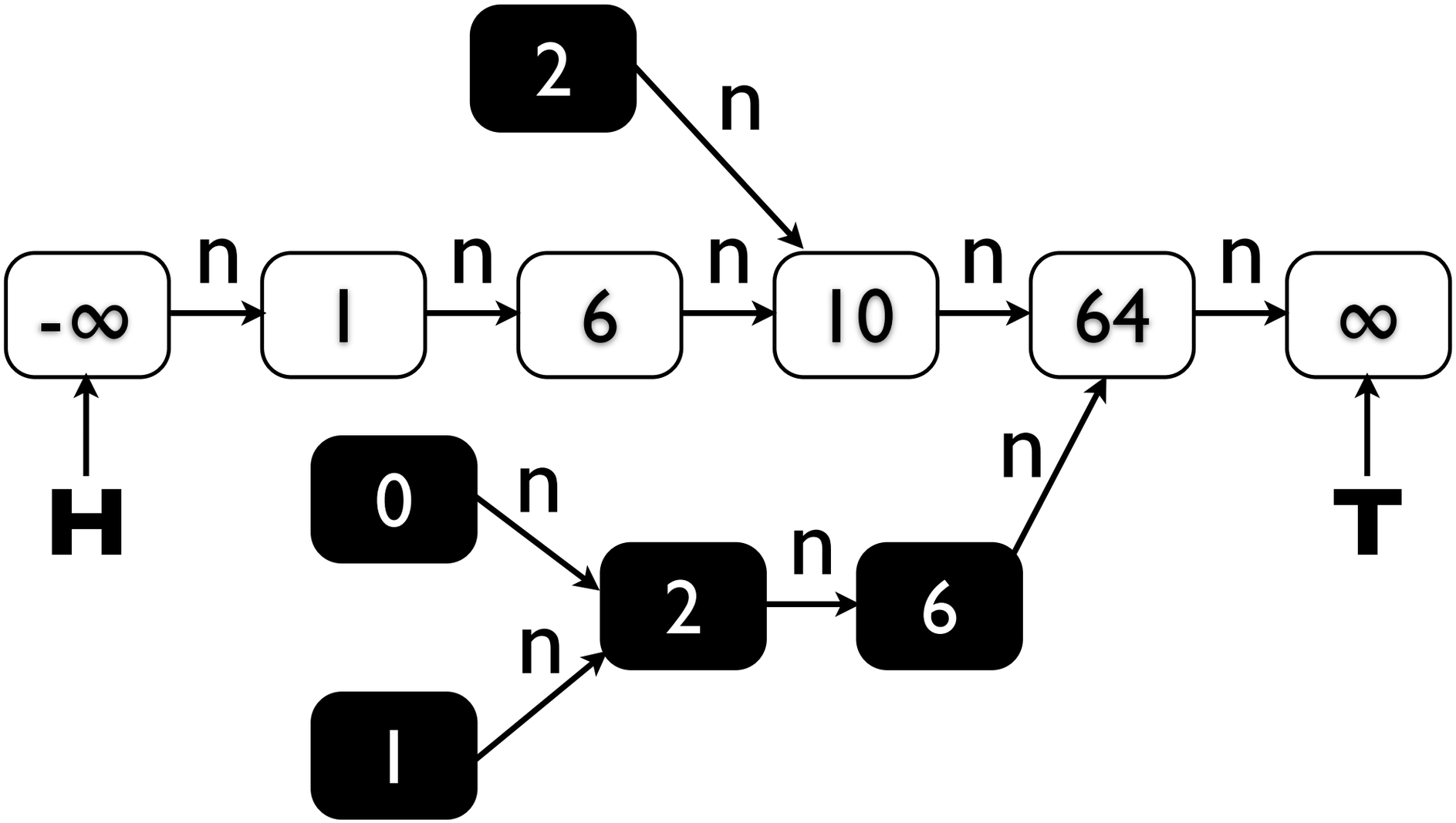


proof: using program logic

10 $\{\psi_s(-\infty, H, T, \infty)\}$
11 `bool remove(int k){`
12 $\quad\{\psi_s(-\infty, H, T, \infty)\}$
13 `while (true){`
14 $\quad\{\psi_s(-\infty, H, T, \infty)\}$
15 `E p = H;`
16 `E c = H.n;`
17 `while(c.k < k){`
18 `p = c;`
19 `c = p.n`
20 `}`
21 $\{k'_c \geq k \wedge \left(-\infty < k'_p < k \leq \infty \wedge -\infty \leq k'_p < k'_c \leq \infty \wedge p \mapsto \langle \_, k'_p, \_\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}\right) \wedge \psi_s(-\infty, H, T, \infty)\}$
$\quad\left|\begin{array}{l}\{\left(-\infty \leq k'_p < k \leq k'_c \leq \infty \wedge p \mapsto \langle \_, k'_p, \_\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}\right) \wedge \psi_s(-\infty, H, T, \infty)\}\\ \left\{\begin{array}{c}\left(\left(-\infty \leq k'_p < k \leq k'_c \leq \infty \wedge p \mapsto \langle \_, k'_p, c\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}\right) \wedge \psi_s(-\infty, H, T, \infty)\right) \vee\\ \left(\left(-\infty \leq k'_p < k \leq k'_c \leq \infty \wedge p \mapsto \langle \_, k'_p, p'\rangle * p' \mapsto \langle \_, k'', \_\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}\right) \wedge \psi_s(-\infty, H, T, \infty)\right)\end{array}\right\}\end{array}\right.$
22
23 `atomic{`
24 `if (p.n==c && !p.m){`
25 $\{\left(-\infty \leq k'_p < k = k'_c \leq \infty \wedge p \mapsto \langle \mathbf{false}, k'_p, c\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}\right) \wedge \psi_s(-\infty, H, T, \infty)\}$
26 `if (c.k==k){`
27 $\{\left(-\infty \leq k'_p < k = k'_c \leq \infty \wedge p \mapsto \langle \mathbf{false}, k'_p, c\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}\right) \wedge \psi_s(-\infty, H, T, \infty)\}$
28 $\{-\infty \leq k'_p < k = k'_c \leq \infty \wedge \psi_s(-\infty, H, p, k'_p) * p \mapsto \langle \mathbf{false}, k'_p, c\rangle * iT(p, k') * c \mapsto \langle \mathbf{false}, k'_c, c'\rangle * iT(c, k'_c) * \psi_s(k'_c, c', T,$
29 `c.m = true;`
30 $\{-\infty \leq k'_p < k = k'_c \leq \infty \wedge \psi_s(-\infty, H, p, k'_p) * p \mapsto \langle \mathbf{false}, k'_p, c\rangle * iT(p, k') * c \mapsto \langle \mathbf{true}, k'_c, c'\rangle * iT(c, k'_c) * \psi_s(k'_c, c', T,$
31 $\left\{\begin{array}{c}-\infty \leq k'_p < k = k'_c \leq \infty \wedge \psi_s(-\infty, H, p, k'_p) * p \mapsto \langle \mathbf{false}, k'_p, c\rangle * iT(p, k') * c \mapsto \langle \mathbf{true}, k'_c, c'\rangle * iT(c, k'_c) *\\ c' \mapsto \langle \mathbf{true}, k'_{c'}, c''\rangle * iT(c', k'_{c'}) * \psi_s(k'_{c'}, c'', T, \infty)\end{array}\right\}$
32 `p.n = c.n;`
33 $\left\{\begin{array}{c}-\infty \leq k'_p < k = k'_c \leq \infty \wedge \psi_s(-\infty, H, p, k'_p) * p \mapsto \langle \mathbf{false}, k'_p, c'\rangle * iT(p, k') * c \mapsto \langle \mathbf{true}, k'_c, c'\rangle * iT(c, k'_c) *\\ c' \mapsto \langle \mathbf{true}, k'_{c'}, c''\rangle * iT(c', k'_{c'}) * \psi_s(k'_{c'}, c'', T, \infty)\end{array}\right\}$
34 $\{(-\infty \leq k'_p < k = k'_c \leq \infty \wedge p \mapsto \langle \_, k'_p, \_\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}) \wedge \psi_s(-\infty, H, T, \infty)\}$
35 `return true`
36 `}`
37 $\{\left(-\infty \leq k'_p < k < k'_c \leq \infty \wedge p \mapsto \langle \mathbf{false}, k'_p, c\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}\right) \wedge \psi_s(-\infty, H, T, \infty)\}$
38 `else return false`
39 `}`
40 $\left(\left(-\infty \leq k'_p < k \leq k'_c \leq \infty \wedge p \mapsto \langle \_, k'_p, p'\rangle * p' \mapsto \langle \_, k'', \_\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}\right) \wedge \psi_s(-\infty, H, T, \infty)\right)$
41 `}`
42 $\left|\begin{array}{l}\{\left(-\infty \leq k'_p < k \leq k'_c \leq \infty \wedge p \mapsto \langle \_, k'_p, p'\rangle * p' \mapsto \langle \_, k'', \_\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}\right) \wedge \psi_s(-\infty, H, T, \infty)\}\\ \psi_s(-\infty, H, T, \infty)\}\end{array}\right.$
43 `}`
44 $\{\left((ret \iff k = k'_c) \wedge -\infty \leq k'_p < k \leq k'_c \leq \infty \wedge p \mapsto \langle \_, k'_p, \_\rangle * c \mapsto \langle \_, k'_c, \_\rangle * \mathbf{true}\right) \wedge \psi_s(-\infty, H, T, \infty)\}$

Figure 15. Resilient proof of `remove`.

verifying linearizability



$\mathcal{A}(\sigma) = \{1, 6, 10, 64\}$

verifying linearization with simulation

```
bool insert(int k) {
  while (true){
    p,c=LOCATE(k);
    if (p.n==c && !p.m){
      if (c.k==k) return false;
      x = alloc N();
      x.m = false;
      x.k=k;  x.n=c;  p.n=x;
      return true
} }
```

```
bool remove(int k) {
  while (true){
    p,c=LOCATE(k);
    if (p.n==c && !p.m){
      if (c.k!=k) return false;
      c.m=true;
      p.n=c.n;
      return true;
    }
} }
```

verifying linearization with hindsight

```
bool contains(int k) {

    p,c=LOCATE(k);
    return (c.k==k)

}
```

```
LOCATE(k)
    p = H;
    c = H.n
    while (c.k < k) {
        p = c;
        c = c.n;
    }
```

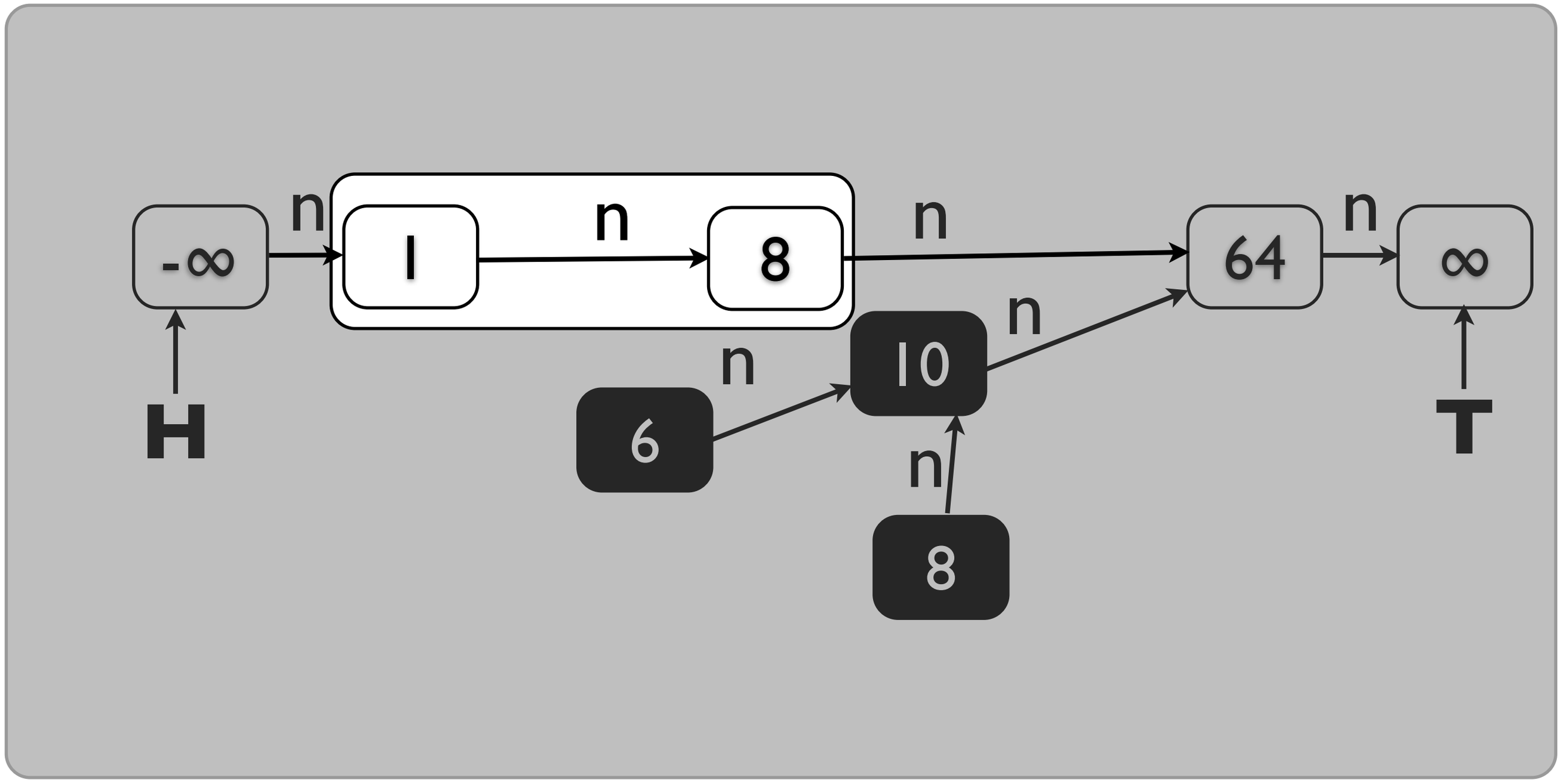
verifying linearizability with hindsight

- **Local Window to $\mathcal{A}(\sigma)$** if link $u \xrightarrow{n} v$ is reachable from H in state σ
  then $\mathcal{A}(\sigma) \cap \{u.k_\sigma,...,v.k_\sigma\} = \{u,v\} \setminus \{-\infty,\infty\}$

- **Hindsight** every link $u \xrightarrow{n} v$ that a thread traverses was reachable from H at some state which occur during the traversal

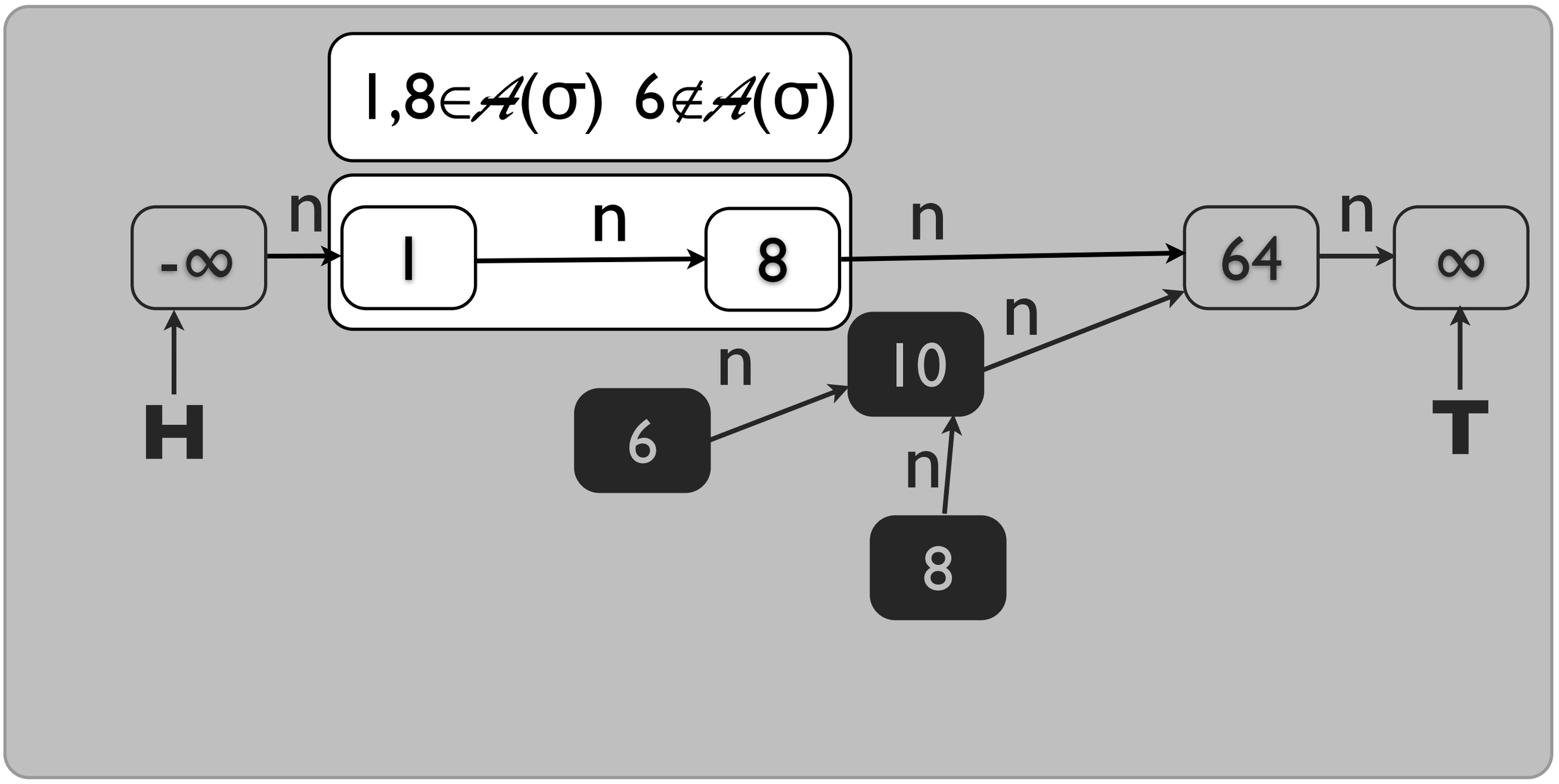
local window

$\mathcal{A}(\sigma) \cap \{u.k_\sigma,\ldots,v.k_\sigma\} = \{u,v\} \setminus \{-\infty,\infty\}$



local window

$\mathcal{A}(\sigma) \cap \{u.k_\sigma,\ldots,v.k_\sigma\} = \{u,v\} \setminus \{-\infty,\infty\}$



verifying linearizability with hindsight

- **Local Window to $\mathcal{A}(\sigma)$** if link $u \xrightarrow{n} v$ is reachable from H in state σ
  then $\mathcal{A}(\sigma) \cap \{u.k_\sigma,...,v.k_\sigma\} = \{u,v\} \setminus \{-\infty,\infty\}$

- **Hindsight** every link $u \xrightarrow{n} v$ that a thread traverses was reachable from H at some state which occur during the traversal

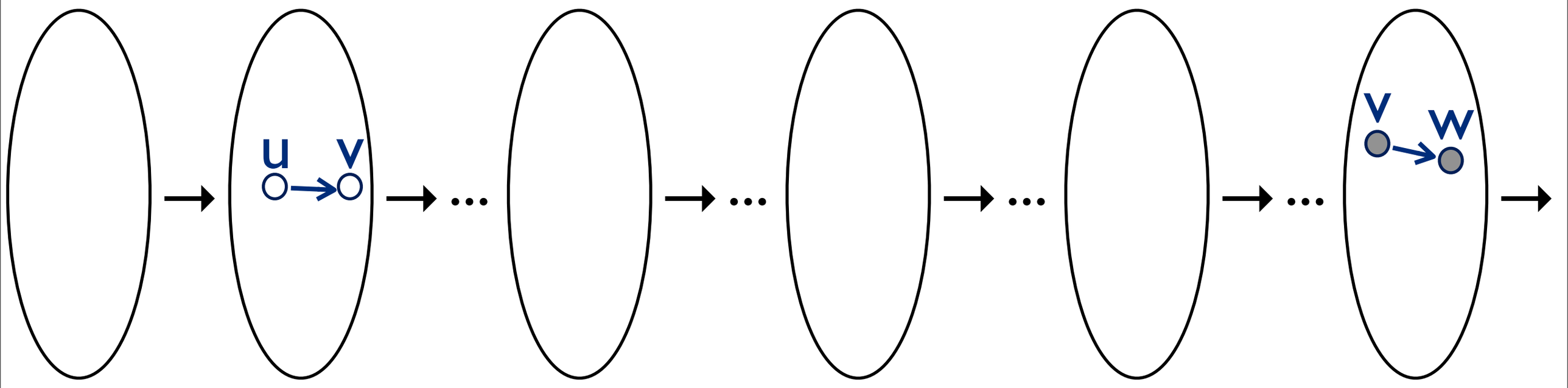
hindsight lemma

let $\pi = \cdots \sigma \cdots \sigma' \cdots$ be a trace which satisfies the state and step invariants

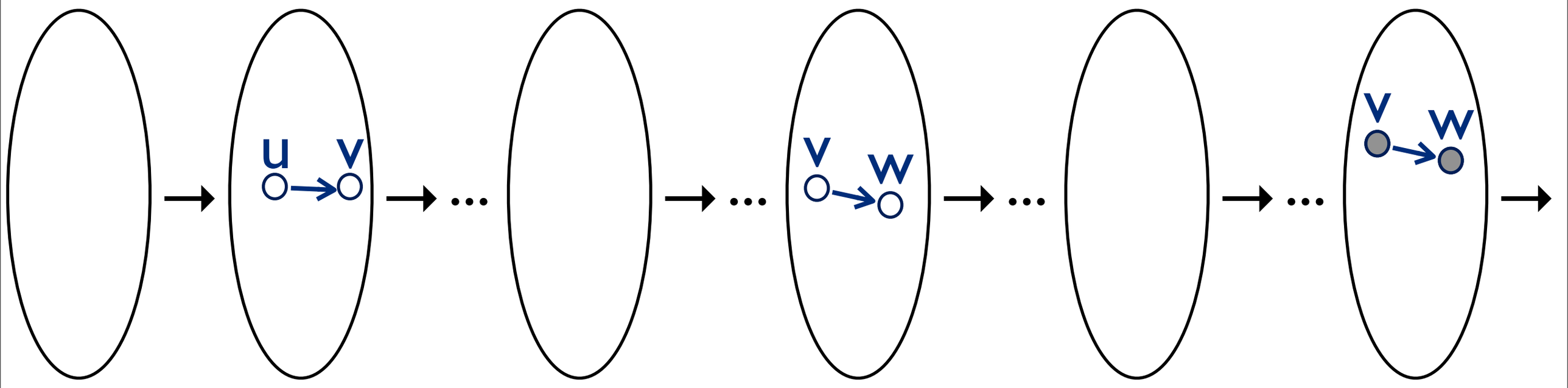
if in state $\sigma = \pi(i)$ link $u \xrightarrow{n} v$ is reachable from H and in state $\sigma' = \pi(j)$ there is a link $v \xrightarrow{n} w$ (where $i \leq j$)

then there exists a state $\sigma'' = \pi(k)$ for some $i \leq k \leq j$ such that in σ'' link $u \xrightarrow{n} v$ is reachable from H

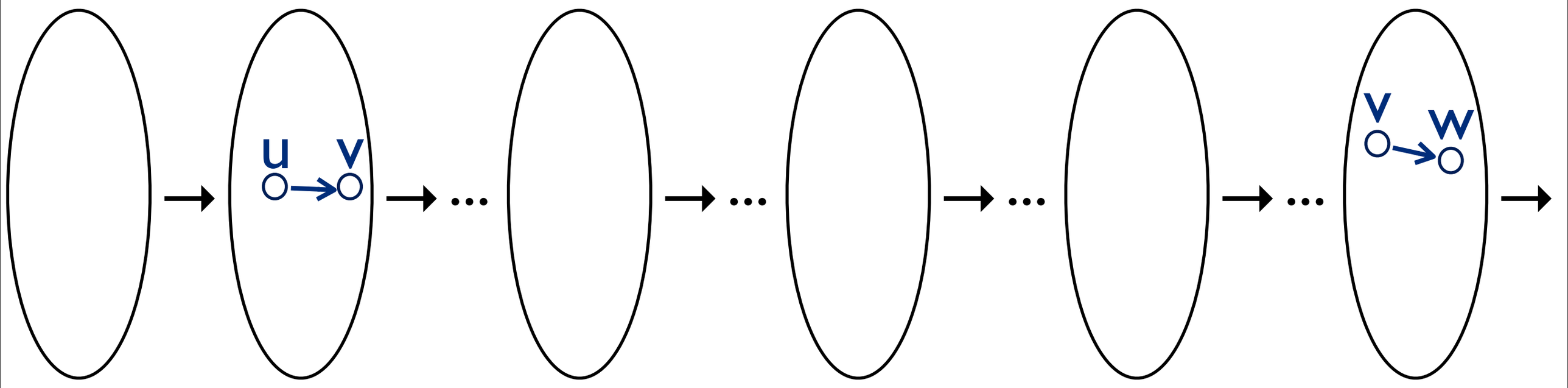
hindsight lemma



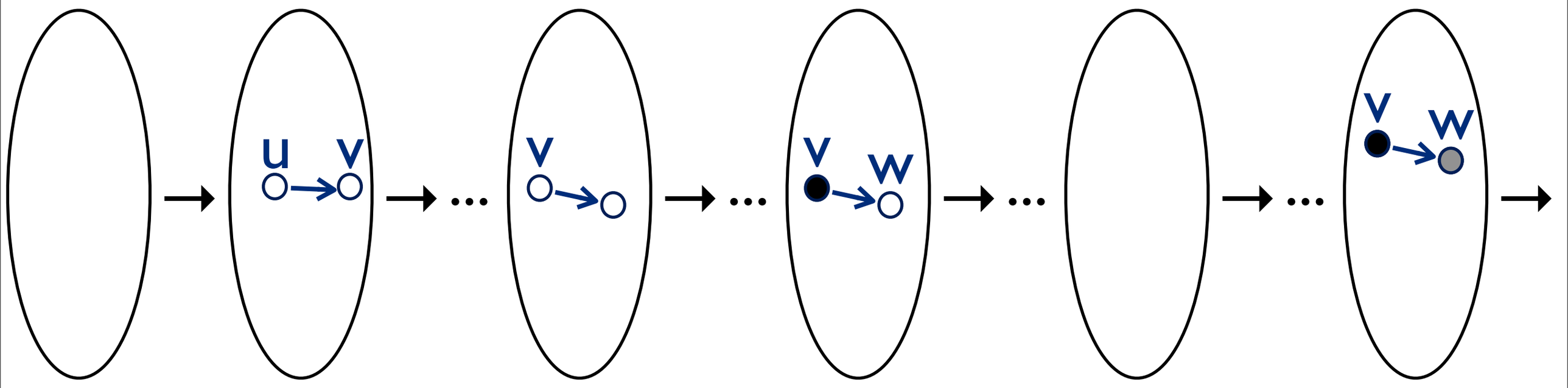
hindsight lemma



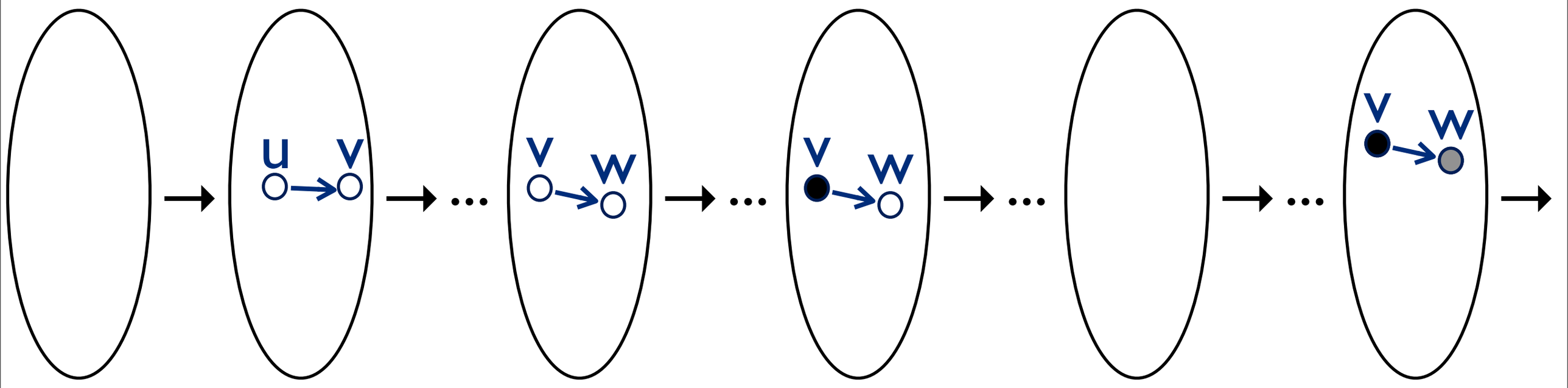
proof sketch: case 1



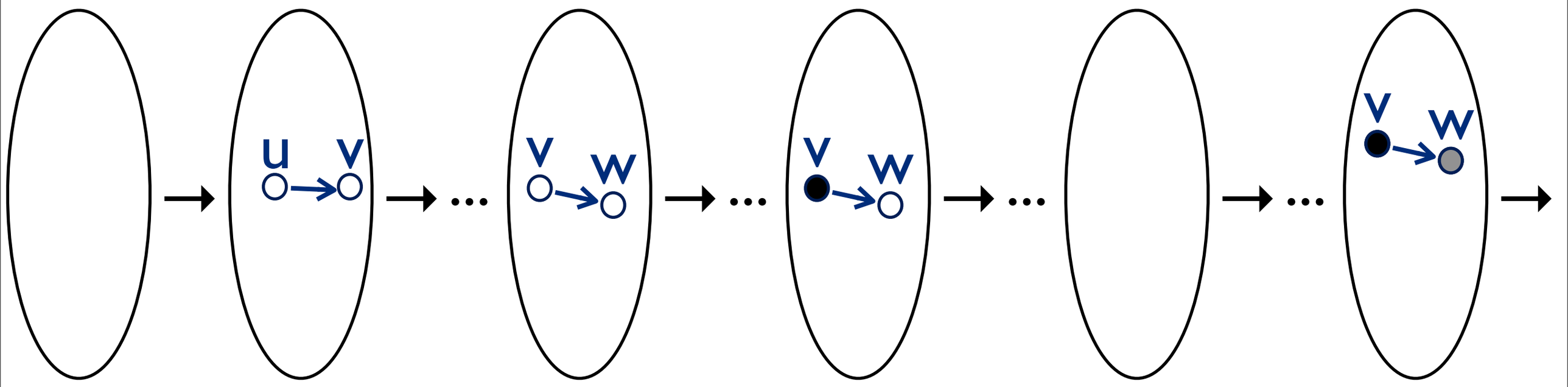
proof sketch: case 2



proof sketch: case 2

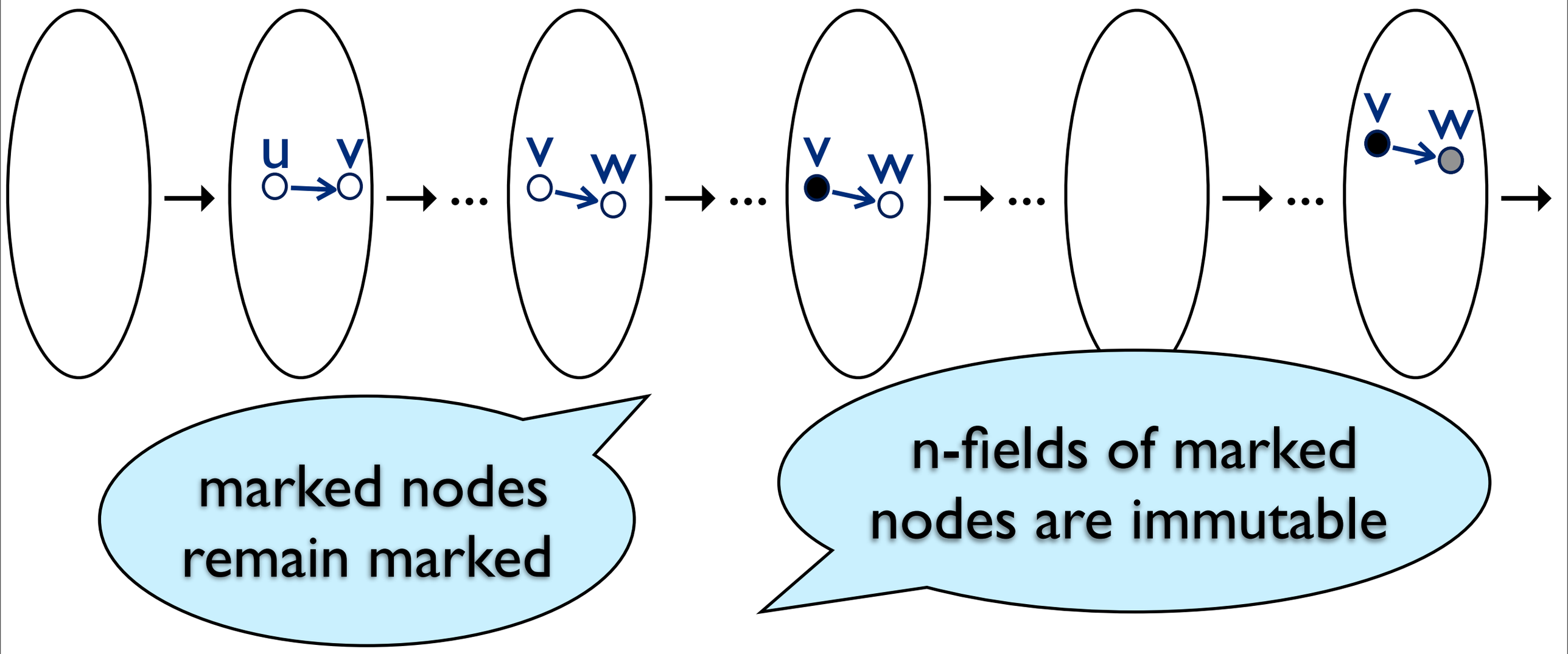


proof sketch: case 2

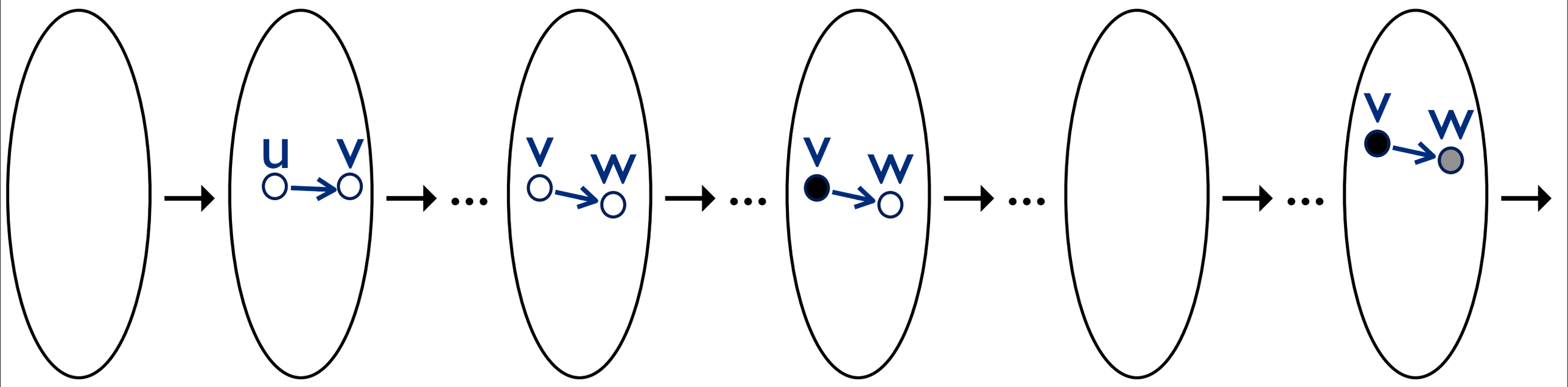


marked nodes
remain marked

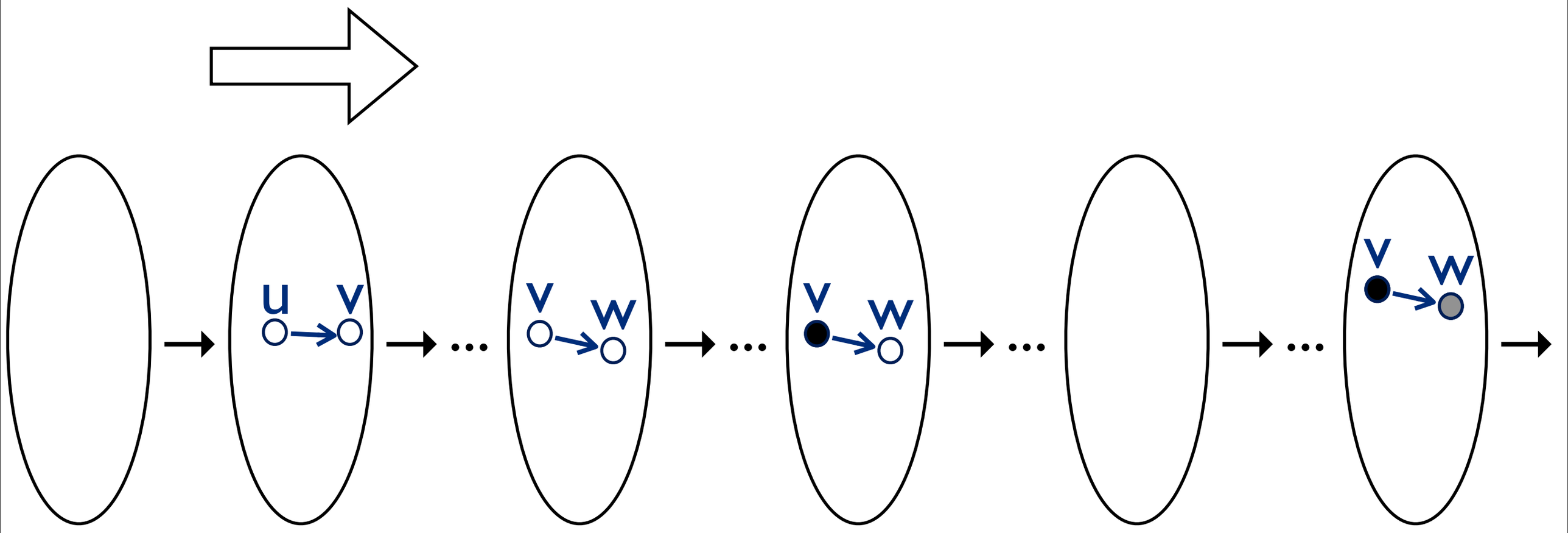
proof sketch: case 2



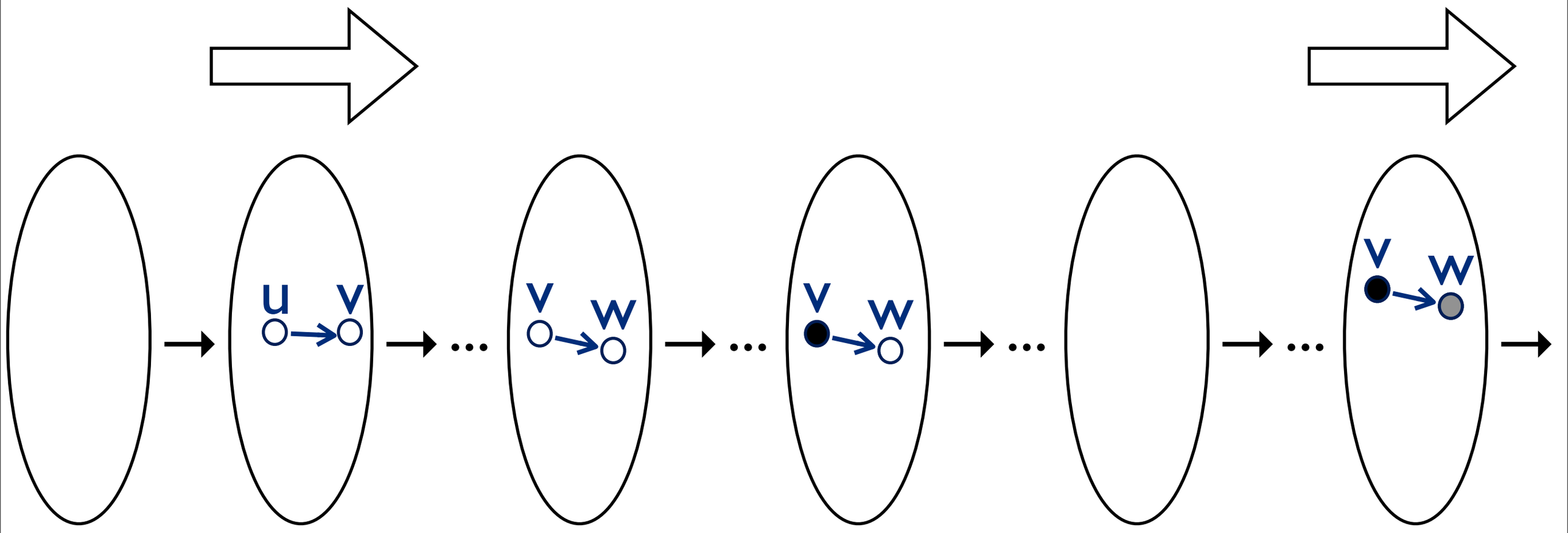
temporal traversal



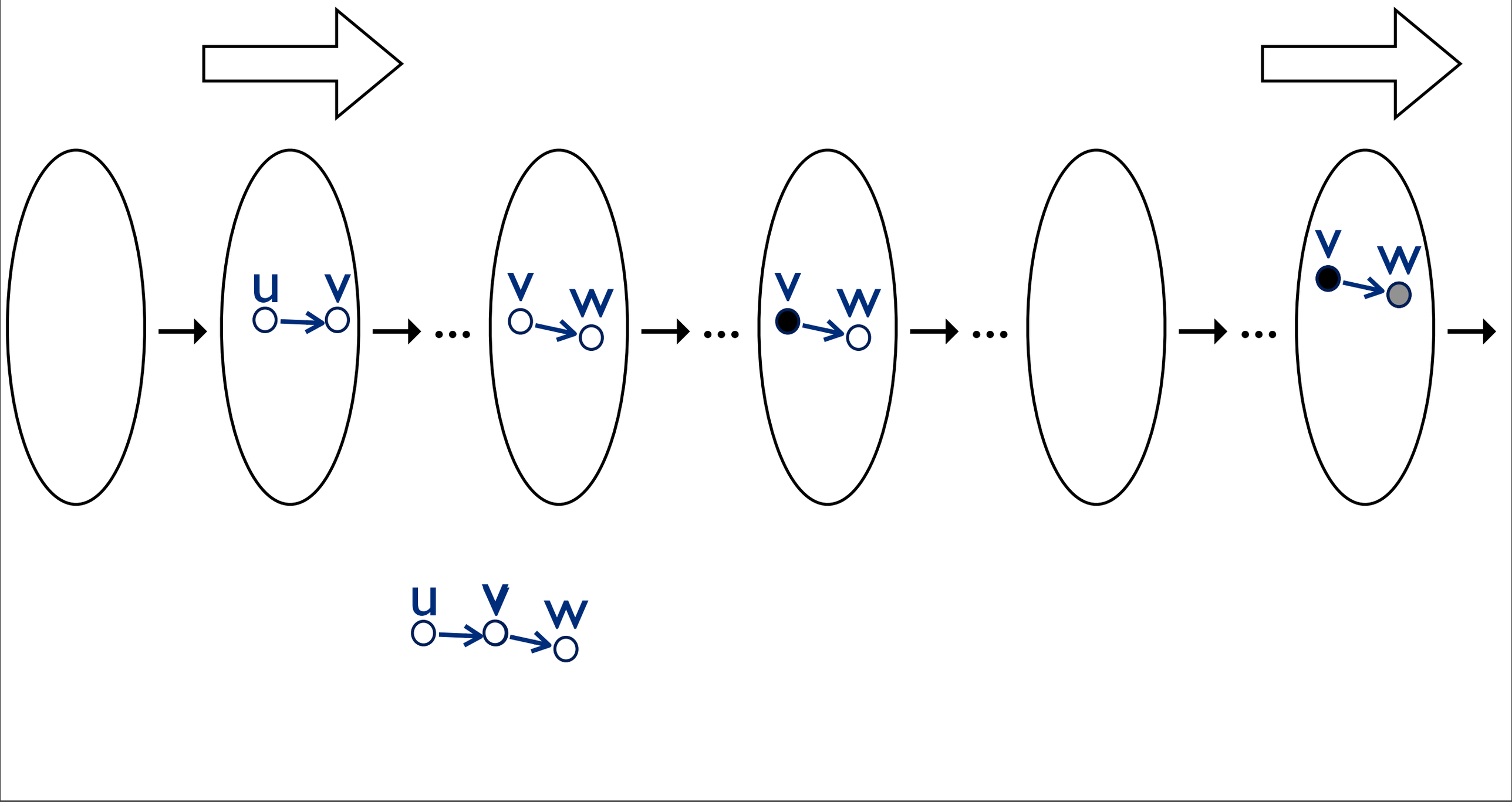
temporal traversal



temporal traversal



temporal traversal



summary

- program logic proof local-protocol preserves global invariant (thread-local proof)
- mathematical proof on good traces
  - not about algorithms / programs
- hindsight good traces allows to conclude properties of past states from current state
- local window concluded properties provide (enough) information for linearizability

~~challenge~~ blessing: interference

- high degree of interference ~~complicates~~ can simplify
  - design
  - understanding
  - verification

thank you(!)

www.eecs.qmul.ac.uk/~maon/pubs/PODC10-hindsight.pdf