van Emde Boas Trees

The Dynamic Predecessor Problem

Goal:

Design a data stucture that maintains a *dynamic* set S of *integers* from a universe $\{0, \ldots, u-1\}$, supporting the following operations:

- Insert(x): Add x into S.
- Delete(x): Remove x from S.
- Find(x): report whether $x \in S$.

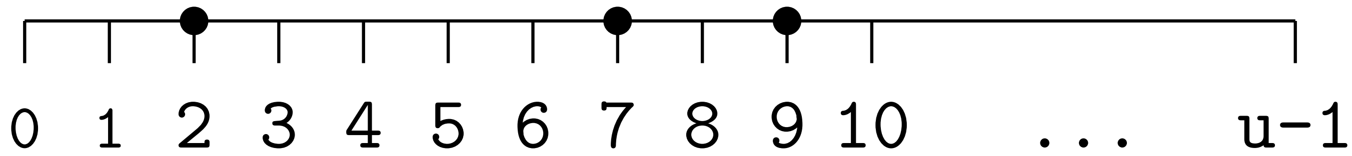
The Dynamic Predecessor Problem

Goal:

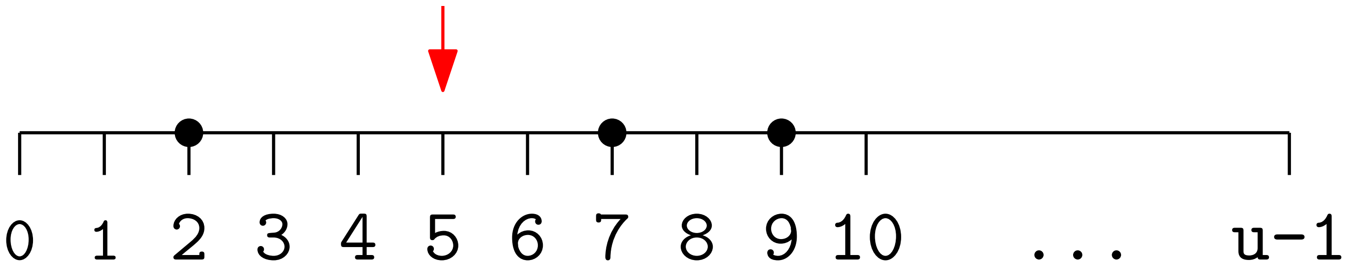
Design a data stucture that maintains a *dynamic* set S of *integers* from a universe $\{0, \ldots, u-1\}$, supporting the following operations:

- Insert(x): Add x into S.
- Delete(x): Remove x from S.
- Find(x): report whether $x \in S$.
- Predecessor(x): return the largest integer $y < x$ in S (if any).
- Successor(x): return the smallest integer $y > x$ in S (if any).

A Geometric Intepretation

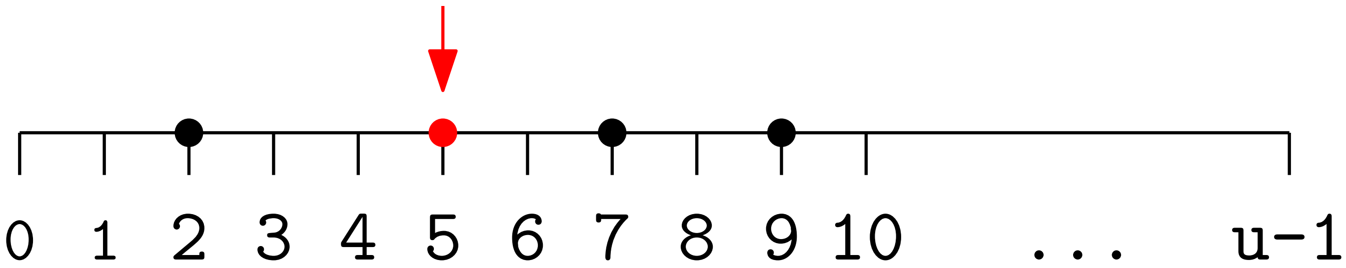


A Geometric Intepretation



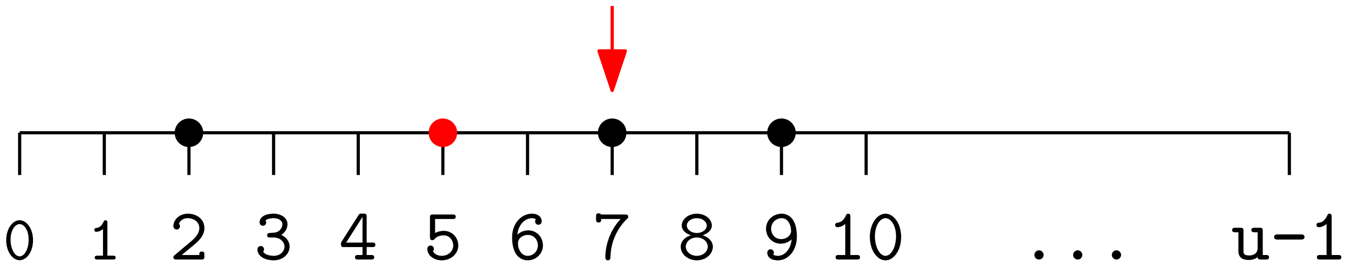
- Insert(5)

A Geometric Intepretation



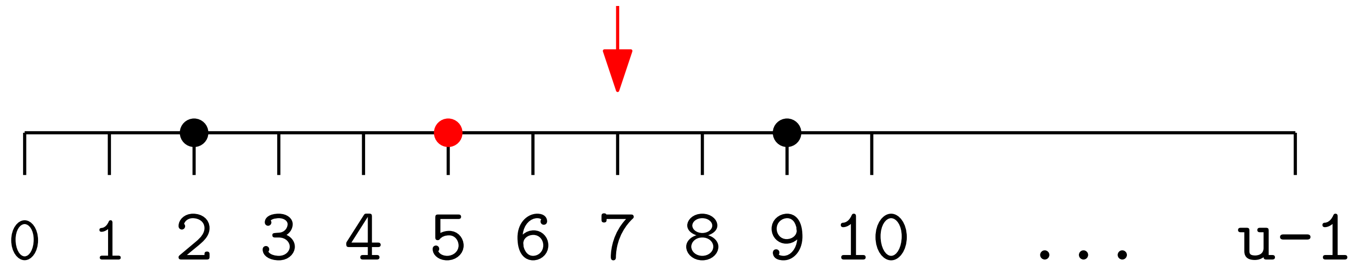
- Insert(5)

A Geometric Intepretation



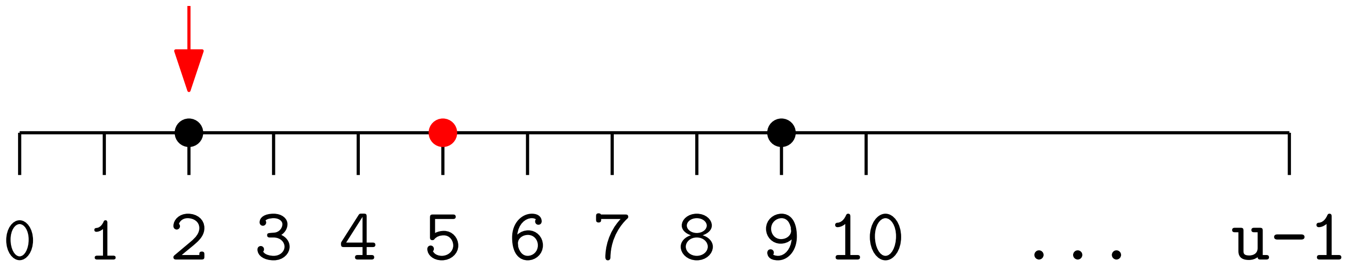
- Insert(5)
- Delete(7)

A Geometric Intepretation



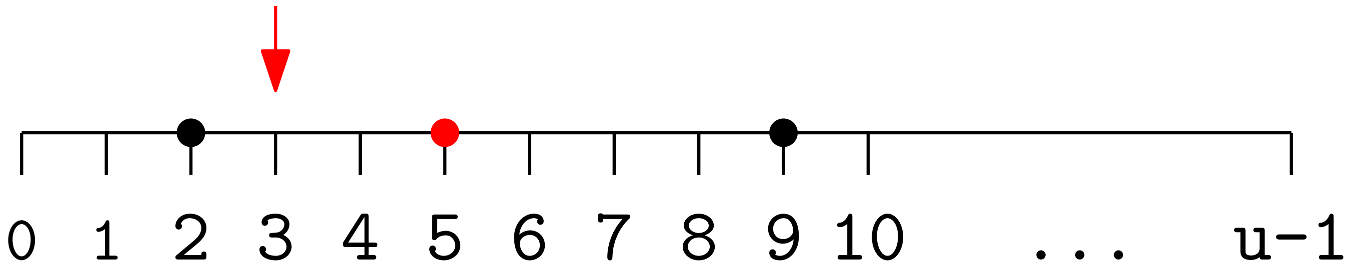
- Insert(5)
- Delete(7)

A Geometric Intepretation



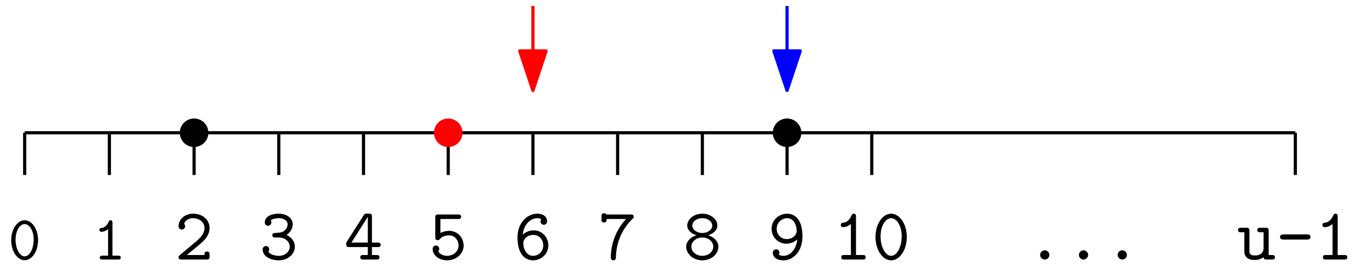
- Insert(5)
- Delete(7)
- Find$(2) = \top$

A Geometric Intepretation



- Insert(5)
- Delete(7)
- Find$(2) = \top$
- Find$(3) = \bot$

A Geometric Intepretation



- Insert(5)
- Delete(7)
- Find$(2) = \top$
- Find$(3) = \bot$
- Successor(6)=9

The Dynamic Predecessor Problem

Goal:

Design a data stucture that maintains a *dynamic* set S of *integers* from a universe $\{0, \ldots, u-1\}$, supporting the following operations:

- Insert(x): Add x into S.
- Delete(x): Remove x from S.
- Find(x): report whether $x \in S$.
- Predecessor(x): return the largest integer $y < x$ in S (if any).
- Successor(x): return the smallest integer $y > x$ in S (if any).

The Dynamic Predecessor Problem

Goal:

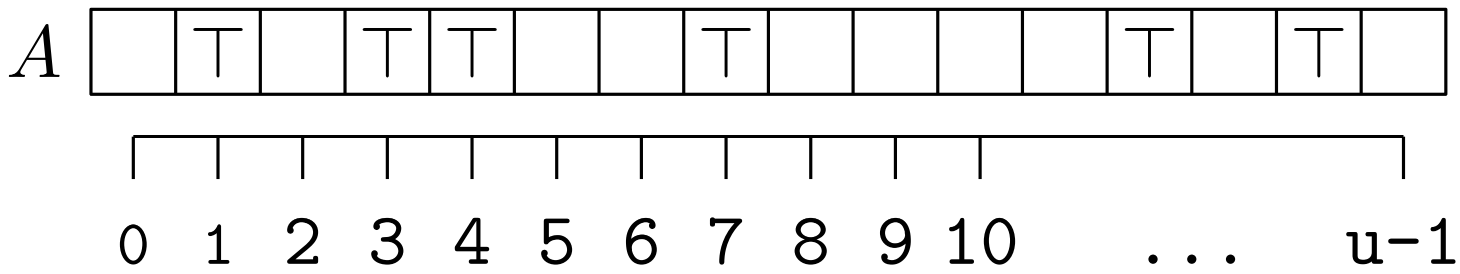
Design a data stucture that maintains a *dynamic* set S of *integers* from a universe $\{0, \ldots, u-1\}$, supporting the following operations:

- Insert(x): Add x into S.
- Delete(x): Remove x from S.
- Find(x): report whether $x \in S$.
- Predecessor(x): return the largest integer $y < x$ in S (if any).
- Successor(x): return the smallest integer $y > x$ in S (if any).

Assume: $u = 2^w$, for some even positive integer w.

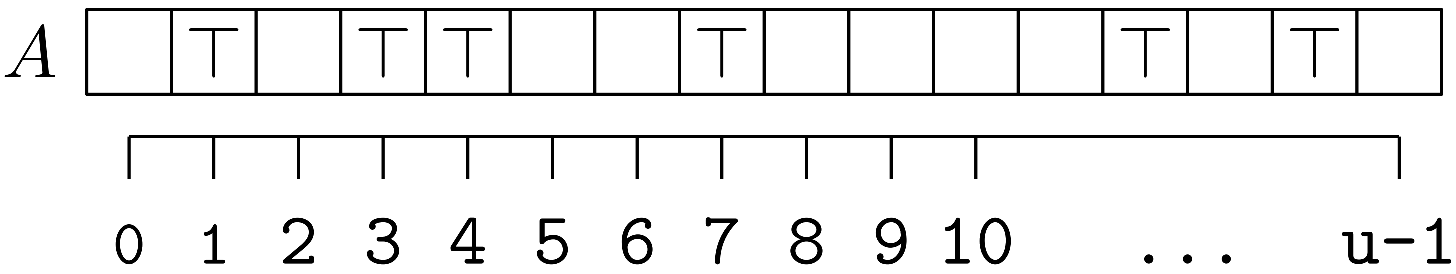
A Trivial Solution

Store a Boolean vector $A[0 : u-1]$ where $A[x] = \top$ iff $x \in S$.



A Trivial Solution

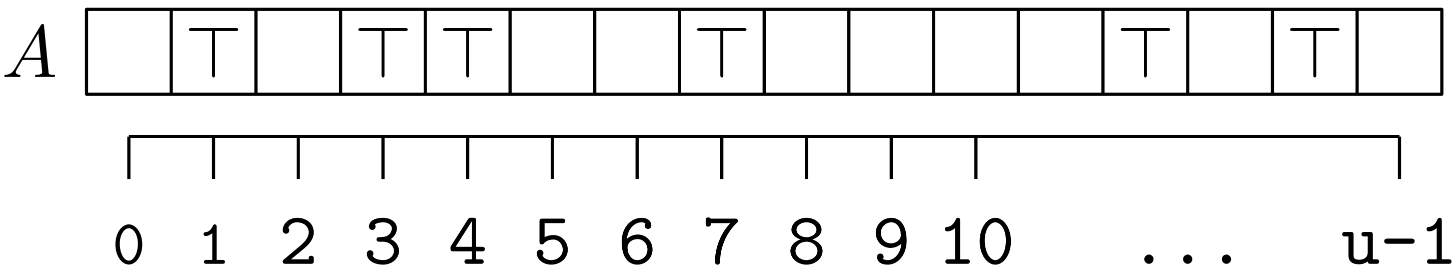
Store a Boolean vector $A[0:u-1]$ where $A[x] = \top$ iff $x \in S$.



- Insert(x): $A[x] \leftarrow \top$ Time: $O(1)$

A Trivial Solution

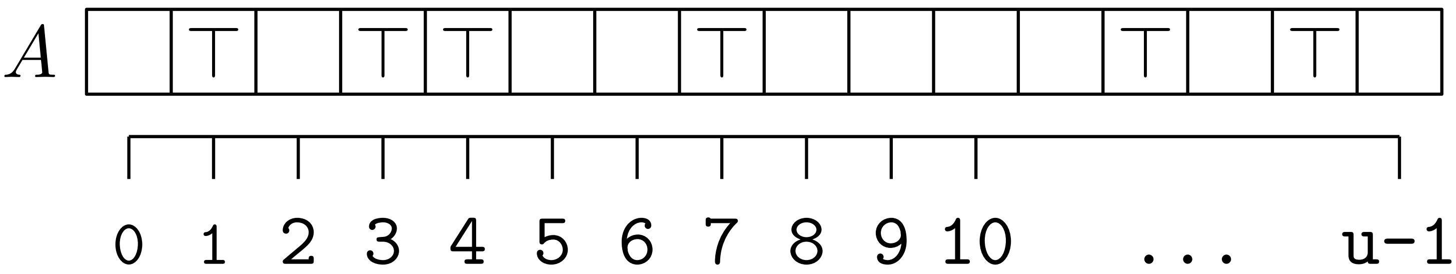
Store a Boolean vector $A[0 : u-1]$ where $A[x] = \top$ iff $x \in S$.



- Insert(x): $A[x] \leftarrow \top$ Time: $O(1)$
- Find(x): Return $A[x]$ Time: $O(1)$

A Trivial Solution

Store a Boolean vector $A[0 : u-1]$ where $A[x] = \top$ iff $x \in S$.

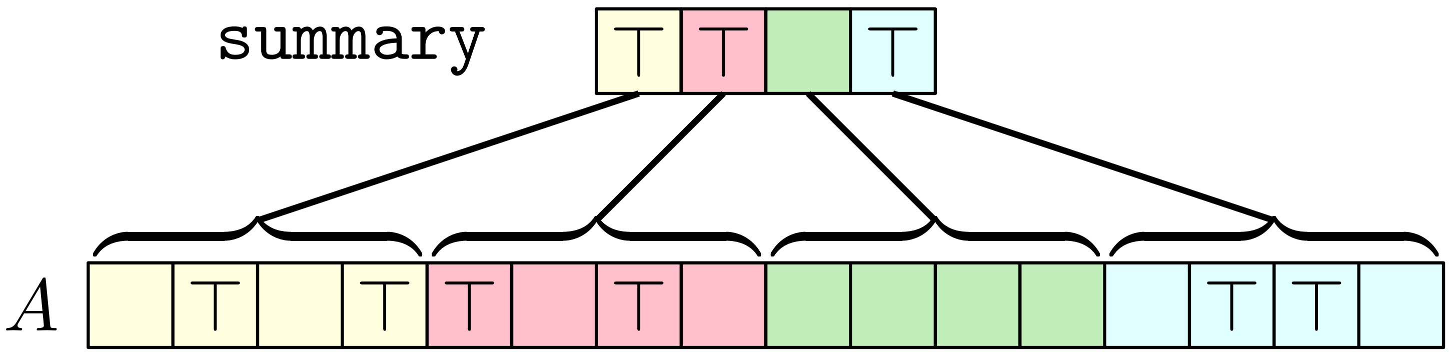


- Insert(x): $A[x] \leftarrow \top$ Time: $O(1)$
- Find(x): Return $A[x]$ Time: $O(1)$
- Successor(x):

Return the smallest $y > x$ with $A[y] = \top$, or $+\infty$ if no such y exists.

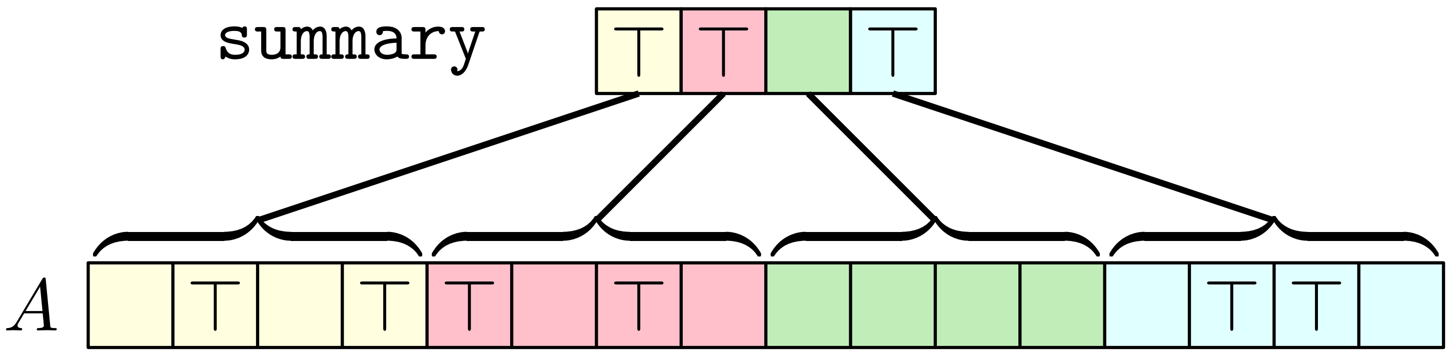
Time: $O(u)$

Speeding up Successor Queries



Split A into $\approx k$ clusters of $\approx \frac{u}{k}$ elements each

Speeding up Successor Queries

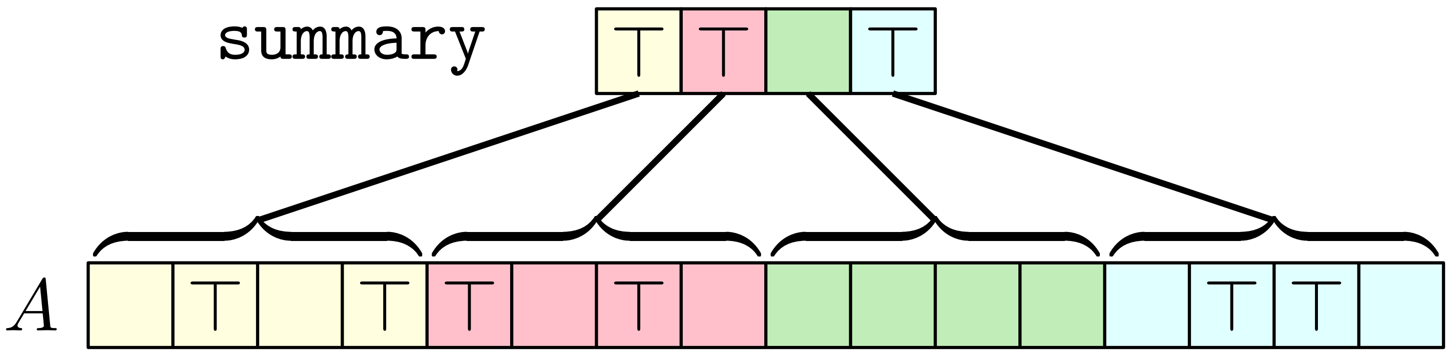


Split A into $\approx k$ clusters of $\approx \frac{u}{k}$ elements each

Let $c_x = \lfloor x/k \rfloor$ be the index of the cluster of x.

Store a `summary` vector. $\texttt{summary}[i] = \top$ if $\exists\ x$, s.t. $c_x = i$ and $A[x] = \top$.

Speeding up Successor Queries



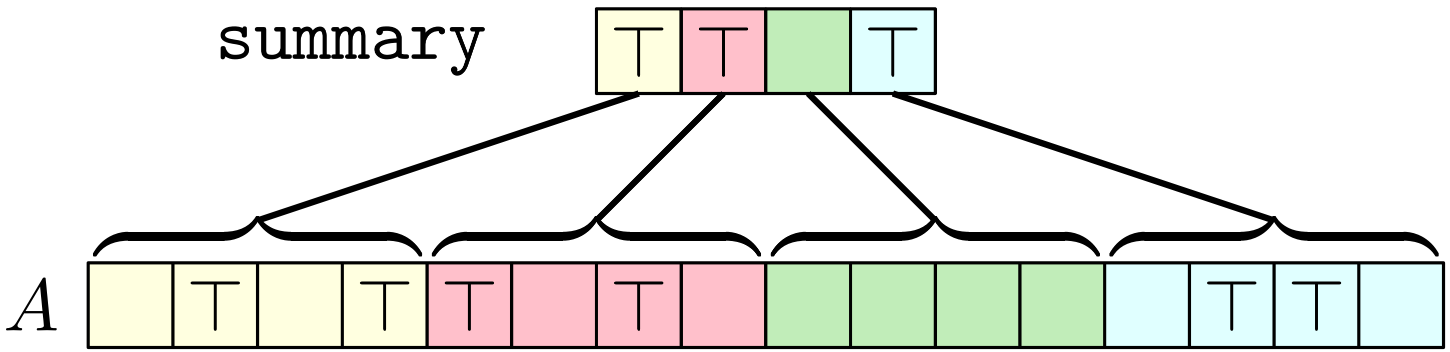
Split A into $\approx k$ clusters of $\approx \frac{u}{k}$ elements each

Let $c_x = \lfloor x/k \rfloor$ be the index of the cluster of x.

Store a `summary` vector. $\texttt{summary}[i] = \top$ if $\exists\ x$, s.t. $c_x = i$ and $A[x] = \top$.

- Find(x): No changes — Time: $O(1)$
- Insert(x): $A[x] \leftarrow \top$; summary$[i] \leftarrow \top$ — Time: $O(1)$

Speeding up Successor Queries



Split A into $\approx k$ clusters of $\approx \frac{u}{k}$ elements each

Let $c_x = \lfloor x/k \rfloor$ be the index of the cluster of x.

Store a `summary` vector. $\texttt{summary}[i] = \top$ if $\exists\, x$, s.t. $c_x = i$ and $A[x] = \top$.

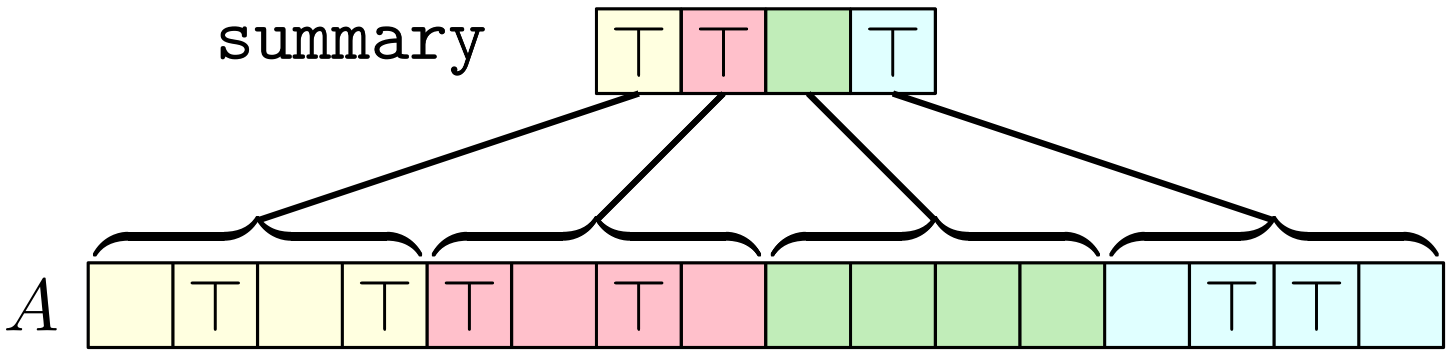
- Successor(x):

  Return the smallest $y > x$ with $c_y = c_x$ and $A[y] = \top$, if any.

  Search for the first cluster $j > c_x$ with $\texttt{summary}[j] = \top$.

  Return the smallest y with $c_y = j$ and $A[y] = \top$ (or $+\infty$).

Speeding up Successor Queries



Split A into $\approx k$ clusters of $\approx \frac{u}{k}$ elements each

Let $c_x = \lfloor x/k \rfloor$ be the index of the cluster of x.

Store a `summary` vector. $\texttt{summary}[i] = \top$ if $\exists\, x$, s.t. $c_x = i$ and $A[x] = \top$.

- Successor(x):

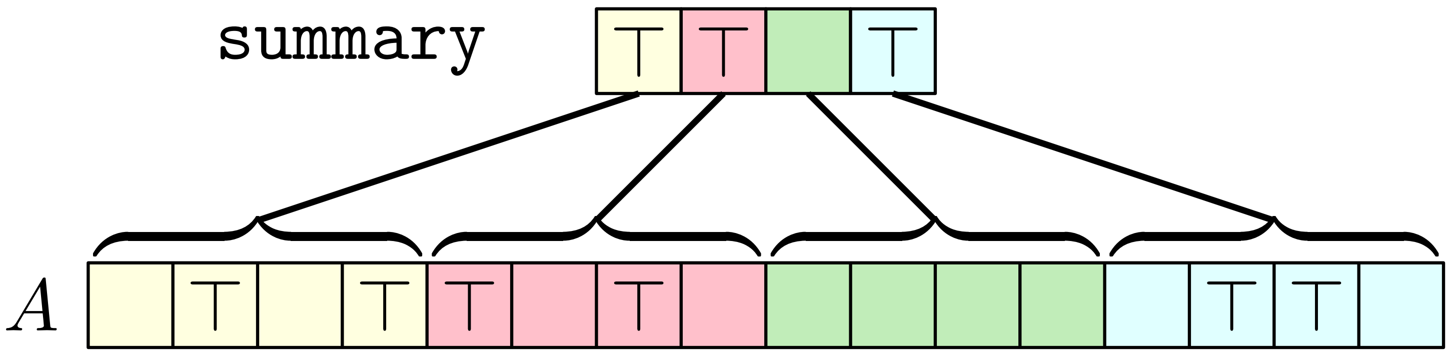
  Return the smallest $y > x$ with $c_y = c_x$ and $A[y] = \top$, if any. $O(u/k)$

  Search for the first cluster $j > c_x$ with $\texttt{summary}[j] = \top$. $O(k)$

  Return the smallest y with $c_y = j$ and $A[y] = \top$ (or $+\infty$). $O(u/k)$

Time: $O(u/k + k)$

Speeding up Successor Queries



Split A into $\approx k$ clusters of $\approx \frac{u}{k}$ elements each

Let $c_x = \lfloor x/k \rfloor$ be the index of the cluster of x.

Store a `summary` vector. $\texttt{summary}[i] = \top$ if $\exists\ x$, s.t. $c_x = i$ and $A[x] = \top$.

- Successor(x):

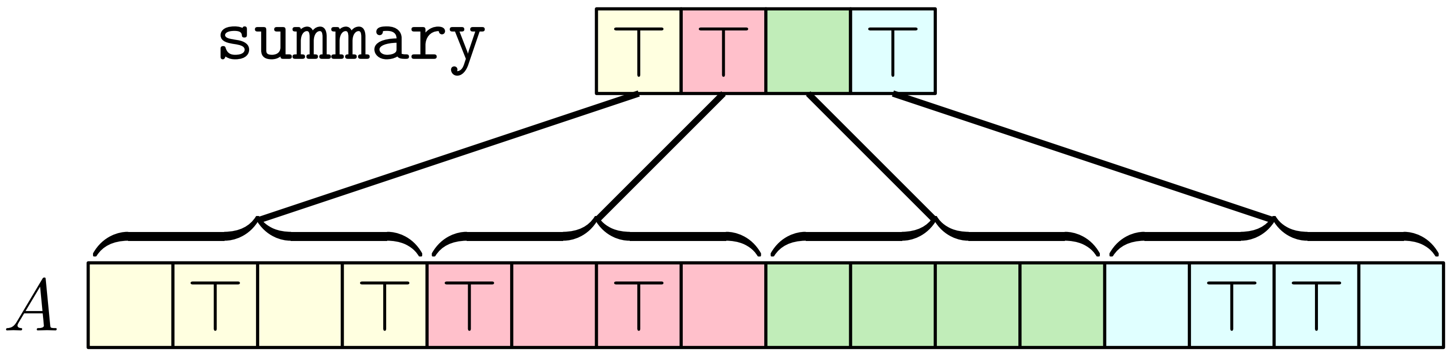
  Return the smallest $y > x$ with $c_y = c_x$ and $A[y] = \top$, if any. $O(u/k)$

  Search for the first cluster $j > c_x$ with $\texttt{summary}[j] = \top$. $O(k)$

  Return the smallest y with $c_y = j$ and $A[y] = \top$ (or $+\infty$). $O(u/k)$

Time: $O(u/k + k)$ How to optimize k?

Speeding up Successor Queries



Split A into $\approx k$ clusters of $\approx \frac{u}{k}$ elements each

Let $c_x = \lfloor x/k \rfloor$ be the index of the cluster of x.

Store a summary vector. $\texttt{summary}[i] = \top$ if $\exists\ x$, s.t. $c_x = i$ and $A[x] = \top$.

- Successor(x):

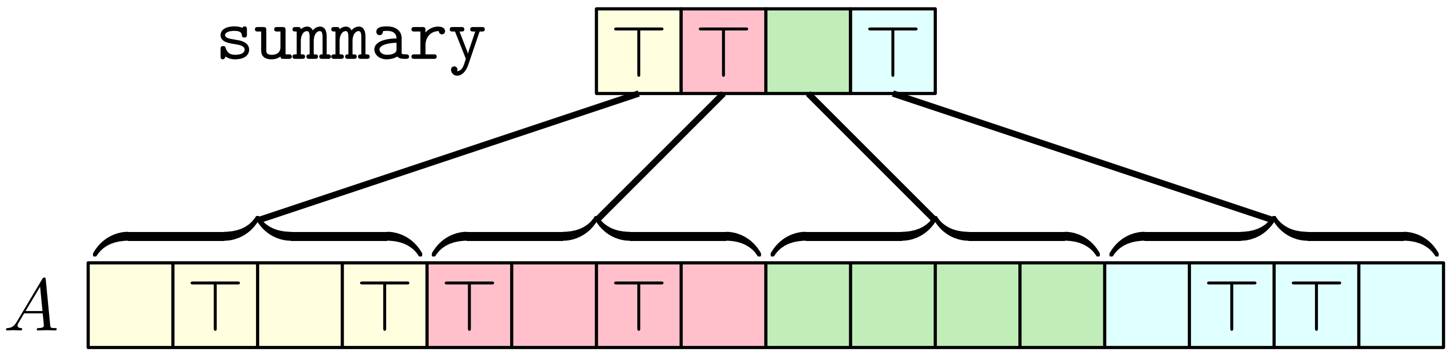
  Return the smallest $y > x$ with $c_y = c_x$ and $A[y] = \top$, if any. $O(u/k)$

  Search for the first cluster $j > c_x$ with $\texttt{summary}[j] = \top$. $O(k)$

  Return the smallest y with $c_y = j$ and $A[y] = \top$ (or $+\infty$). $O(u/k)$

Time: $O(u/k + k)$ How to optimize k? Pick $k = \sqrt{u}$.

Speeding up Successor Queries



Split A into $\approx k$ clusters of $\approx \frac{u}{k}$ elements each

Let $c_x = \lfloor x/k \rfloor$ be the index of the cluster of x.

Store a summary vector. $\texttt{summary}[i] = \top$ if $\exists\, x$, s.t. $c_x = i$ and $A[x] = \top$.

- Successor(x):

  Return the smallest $y > x$ with $c_y = c_x$ and $A[y] = \top$, if any. $O(u/k)$

  Search for the first cluster $j > c_x$ with $\texttt{summary}[j] = \top$. $O(k)$

  Return the smallest y with $c_y = j$ and $A[y] = \top$ (or $+\infty$). $O(u/k)$

Time: $O(u/k + k) = O(\sqrt{u})$ Pick $k = \sqrt{u}$.

A Binary View

Split A into $\sqrt{u}$ clusters of $\sqrt{u}$ elements each.

$$x = (c_x \cdot \sqrt{u}) + (x \bmod \sqrt{u}) = (c_x \cdot 2^{w/2}) + (x \bmod 2^{w/2}).$$

$x =$ `0 1 0 0 1 1 0 1`

A Binary View

Split A into $\sqrt{u}$ clusters of $\sqrt{u}$ elements each.

$$x = (c_x \cdot \sqrt{u}) + (x \bmod \sqrt{u}) = (c_x \cdot 2^{w/2}) + (x \bmod 2^{w/2}).$$

$$x = \underbrace{\texttt{0 1 0 0}}_{w/2 \text{ bits}} \; \underbrace{\texttt{1 1 0 1}}_{w/2 \text{ bits}}$$

A Binary View

Split A into $\sqrt{u}$ clusters of $\sqrt{u}$ elements each.

$$x = (c_x \cdot \sqrt{u}) + (x \bmod \sqrt{u}) \ = (c_x \cdot 2^{w/2}) + (x \bmod 2^{w/2}).$$

$$x = \underbrace{\texttt{0 1 0 0}}_{w/2 \text{ bits}} \ \underbrace{\texttt{1 1 0 1}}_{w/2 \text{ bits}}$$

$\text{high}(x) = \lfloor x / \sqrt{u} \rfloor$ Index of the cluster of x

$\text{low}(x) = x \bmod \sqrt{u}$ Index x within its cluster

$\text{idx}(i, j) = i \cdot \sqrt{u} + j$ Gets x from $i = \text{high}(x)$ and $j = \text{low}(x)$

Recursion!

How do we use `summary` ?

$\text{high}(x)$

- We need to be able to add elements from $\{0, \ldots, \sqrt{u} - 1\}$.
- We need to find the next non-empty cluster.

Recursion!

How do we use `summary` ?

$\text{high}(x)$

- We need to be able to add elements from $\{0, \ldots, \sqrt{u} - 1\}$.
- We need to find the next non-empty cluster.

This is a successor query!

Recursion!

How do we use `summary` ?

- We n… high(x) … $u - 1\}$.
- W… empty cluster.

Replace summary with a (smaller) data structure for the universe $\{0, \ldots, \sqrt{u} - 1\}$.

This is a successor query!

Recursion!

How do we use `summary` ?

(x)

- We n… $-1\}$.
- W… empty cluster.

Replace summary with a (smaller) data structure for the universe $\{0,\ldots,\sqrt{u}-1\}$.

This is a successor query!

What can we do in each cluster?

- Add an element $\{0,\ldots,\sqrt{u}-1\}$
- Find an element low(x)
- Find the next element in the cluster

Recursion!

How do we use `summary` ?

- W... high(x) ... $u - 1\}$.
- W... empty cluster.

Replace summary with a (smaller) data structure for the universe $\{0, \ldots, \sqrt{u} - 1\}$.

This is a successor query!

What can we do in each cluster?

- Add an element $\{0, \ldots, \sqrt{u} - 1\}$
- Find an element low(x)
- Find the next element in the cluster

This is a successor query!

Recursion!

How do we use `summary` ?

- We n... (x) ... $u - 1\}$.
- W... empty cluster.

> Replace summary with a (smaller) data structure for the universe $\{0, \ldots, \sqrt{u} - 1\}$.

This is a successor query!

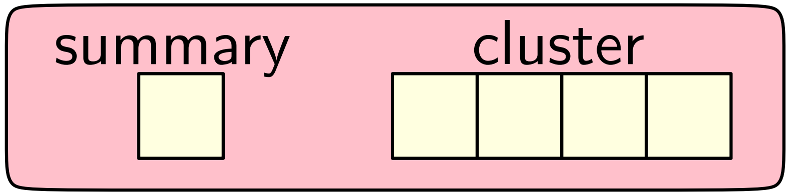
What can we do in each cluster?

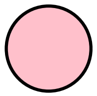
- Add an element $\{0, \ldots, \sqrt{u} - 1\}$
- Fi...
- Fi... in the cluster

> Replace each cluster with a (smaller) data structure for the universe $\{0, \ldots, \sqrt{u} - 1\}$.

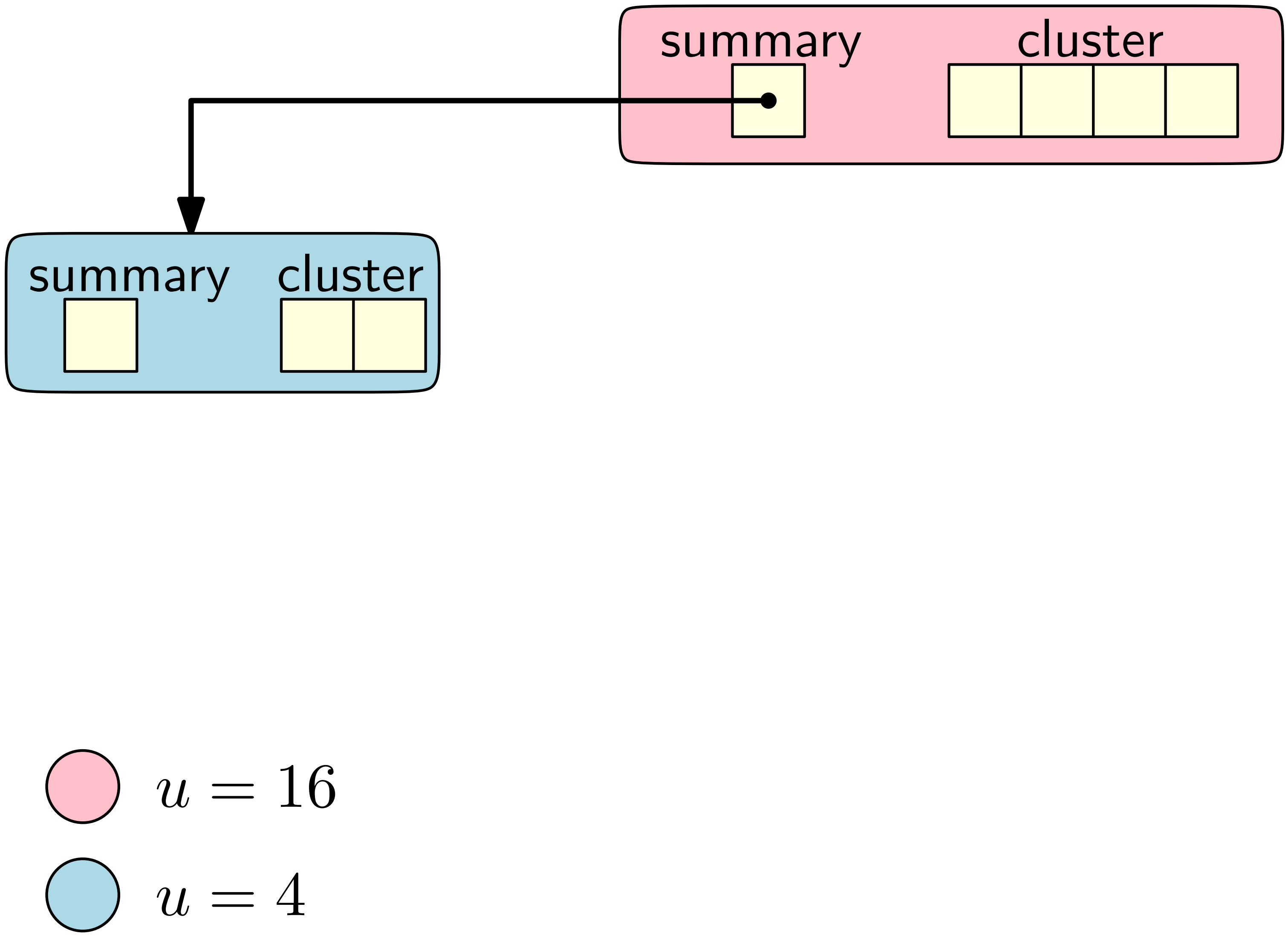
This is a successor query!

An Example for $u = 16$

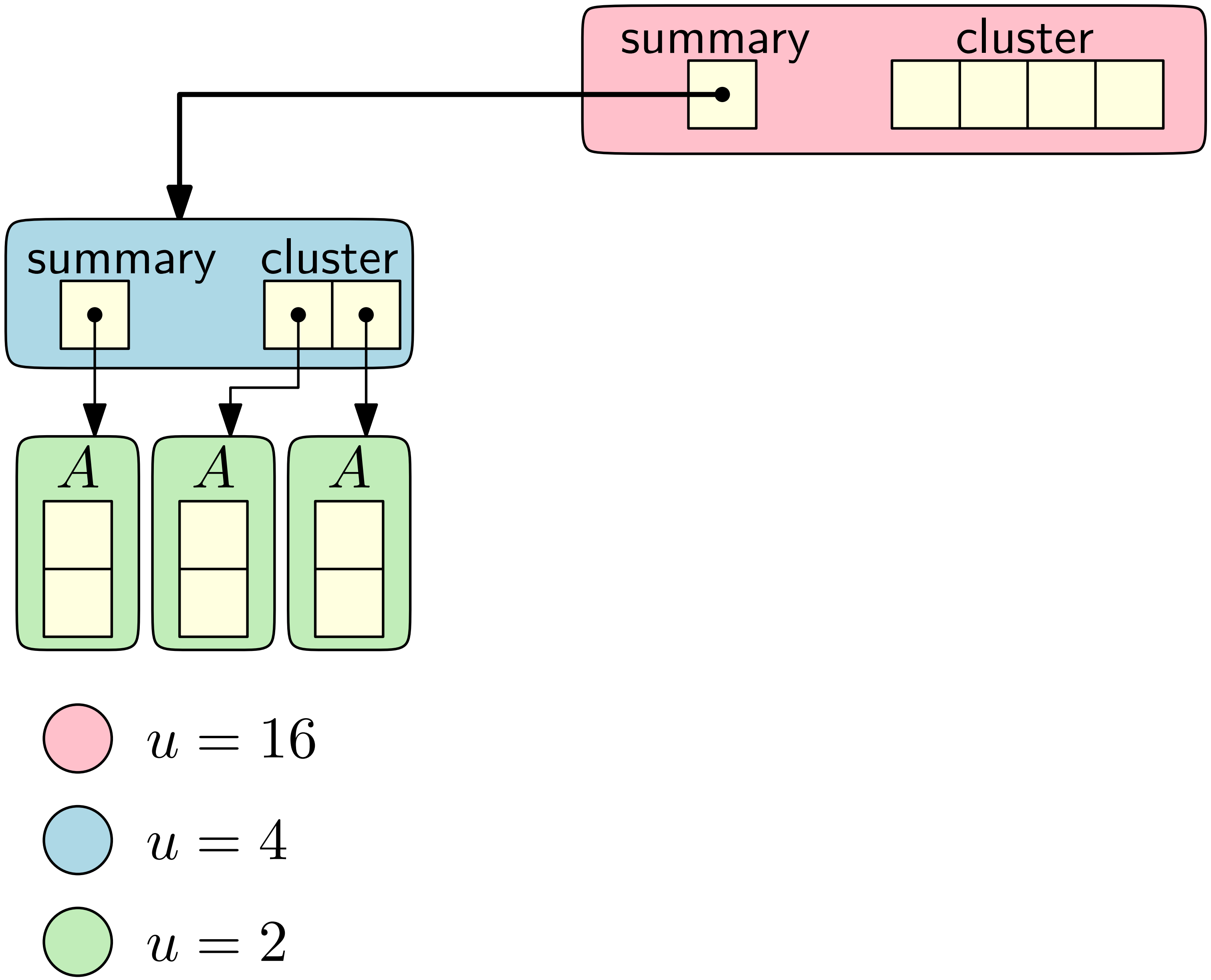


 $u = 16$

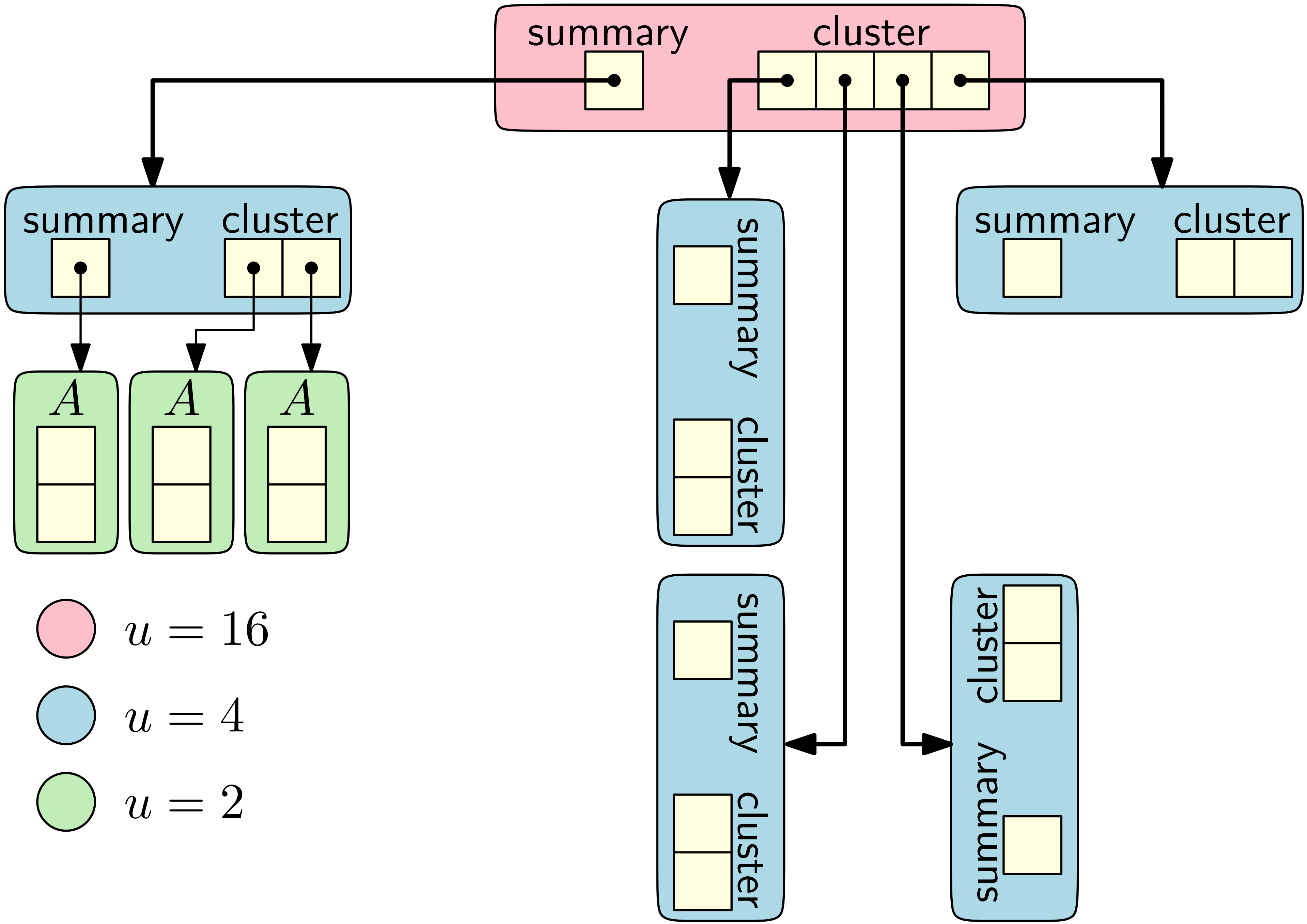
An Example for $u = 16$



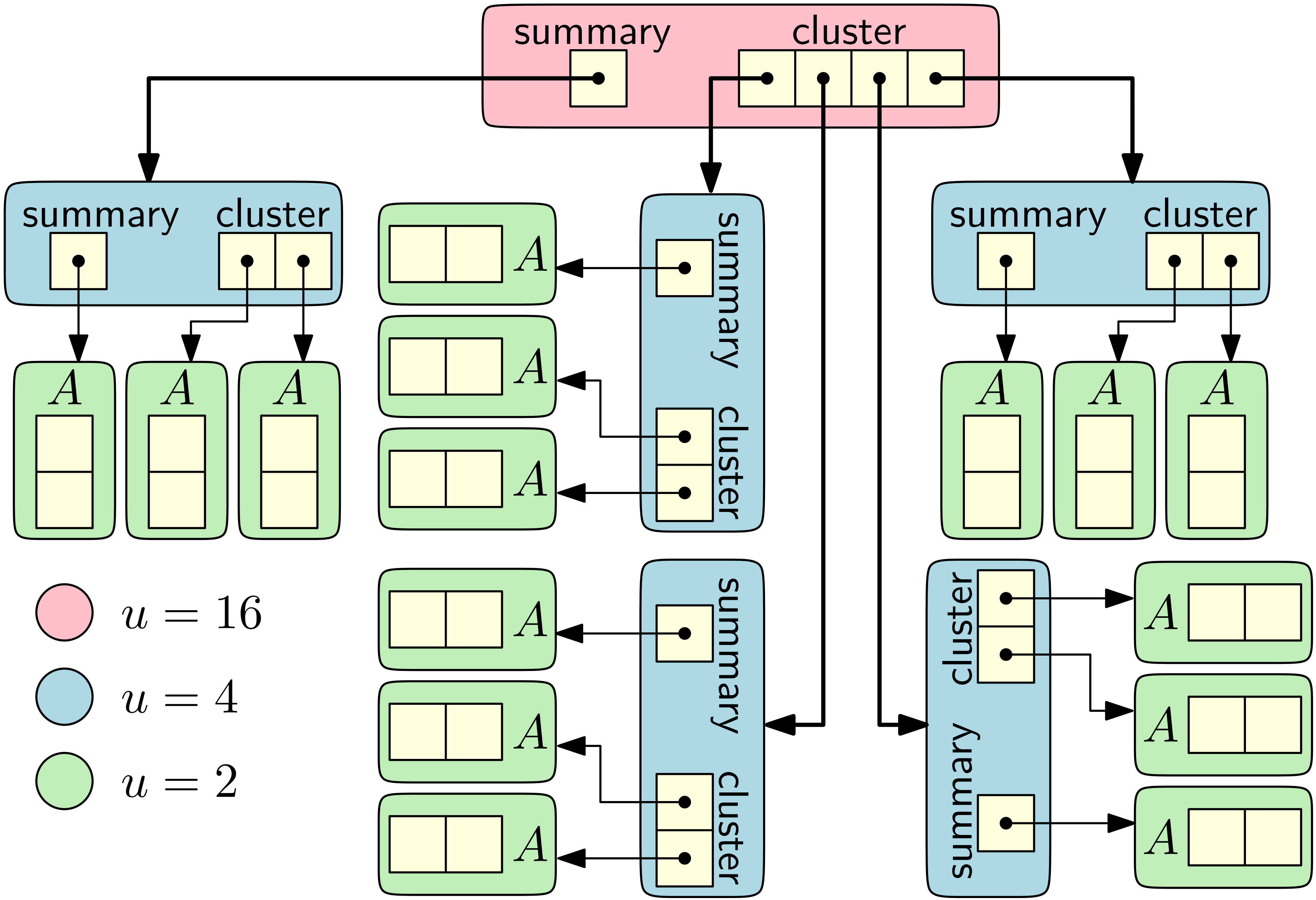
An Example for $u = 16$



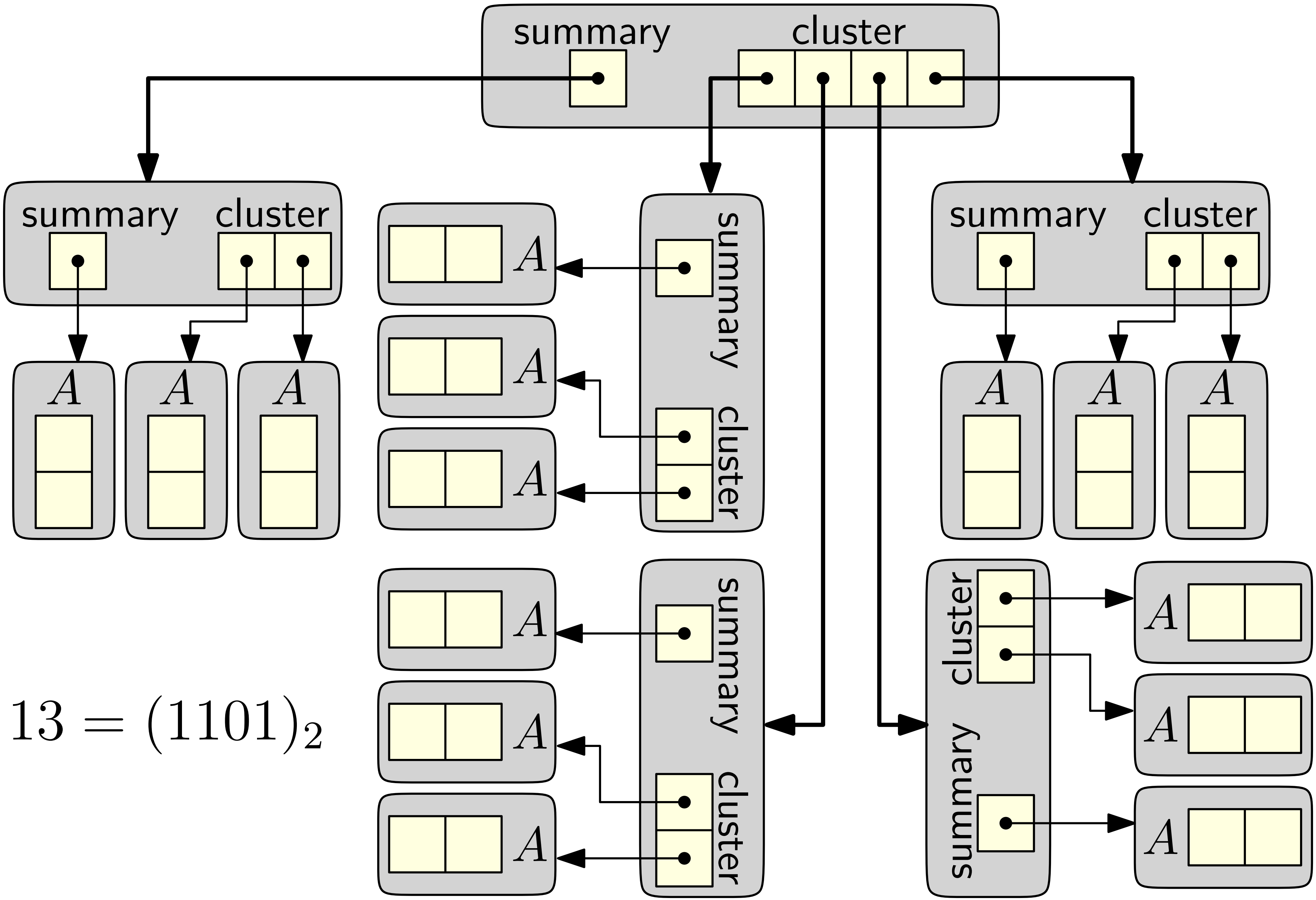
An Example for $u = 16$



An Example for $u = 16$

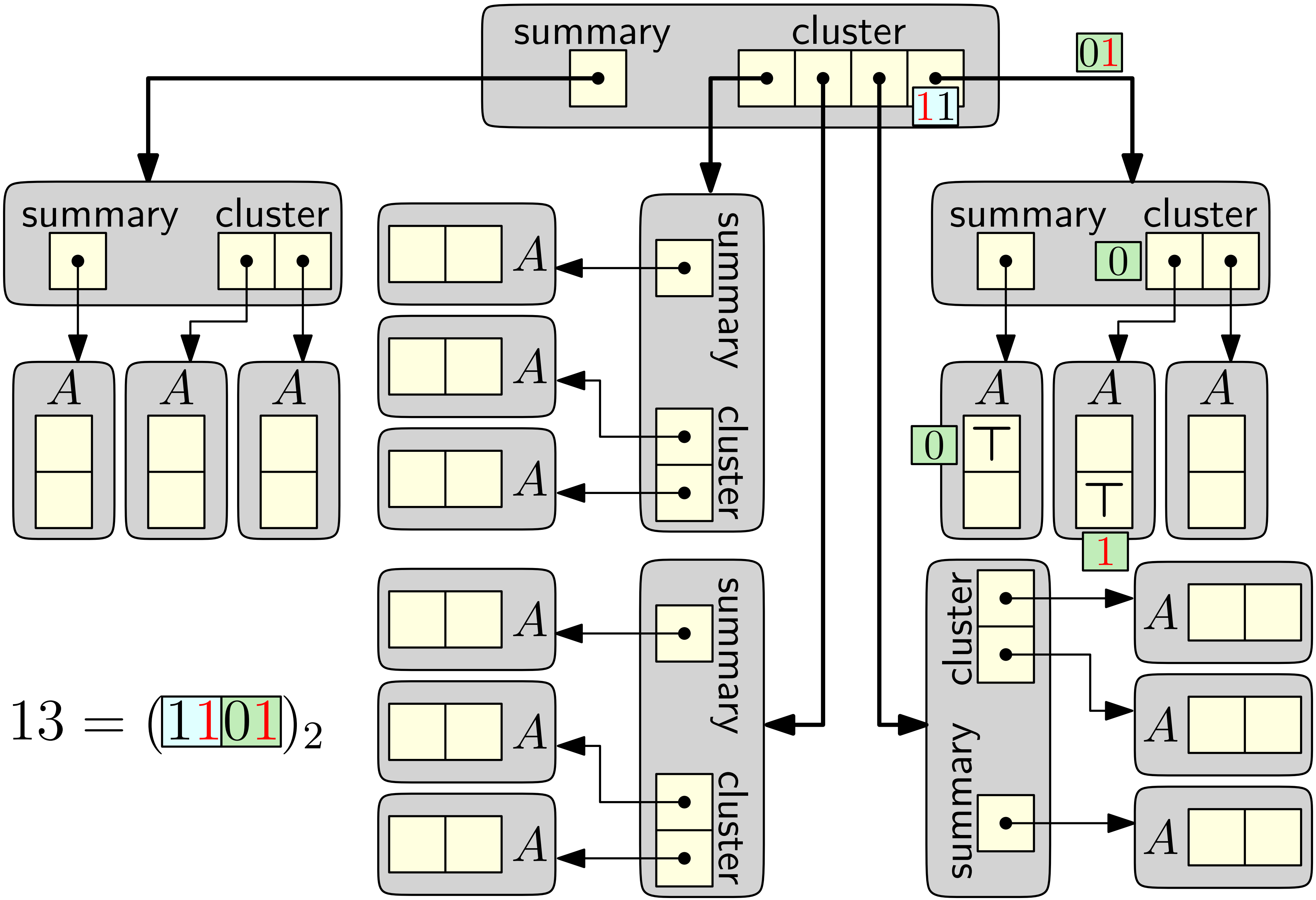


An Example for $u = 16$



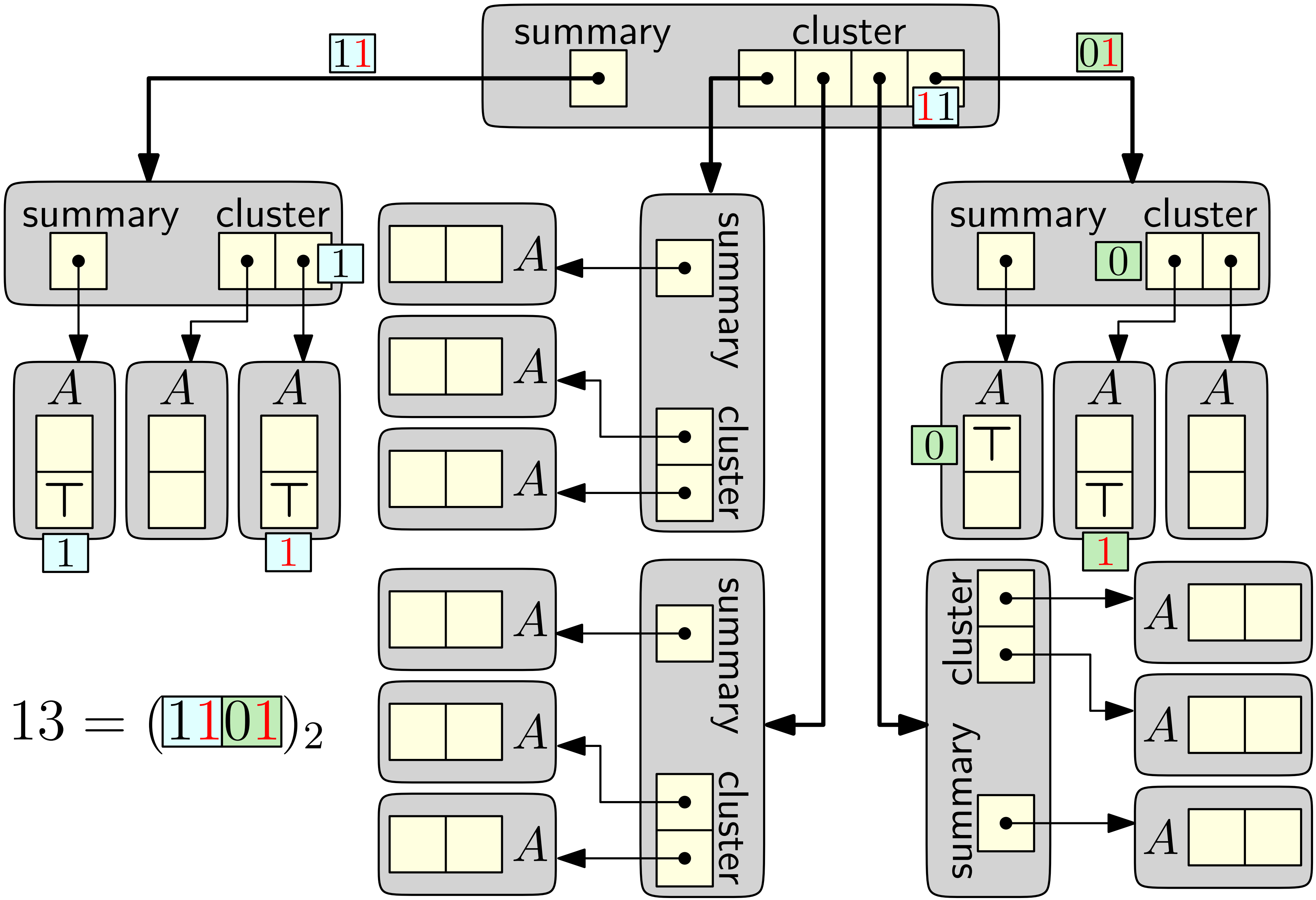
$13 = (1101)_2$

An Example for $u = 16$



$$13 = (1101)_2$$

An Example for $u = 16$



$$13 = (1101)_2$$

Space Usage

Let $S(u)$ be the space usage of a structure for the universe $\{0, \ldots, u-1\}$

- `summary` occupies $S(\sqrt{u})$ space.
- `cluster` occupies $\sqrt{u} \cdot S(\sqrt{u})$ space.
- $O(1)$ additional information (e.g., the value of u)
- (Charge pointers to the inner structures)

Space Usage

Let $S(u)$ be the space usage of a structure for the universe $\{0, \ldots, u-1\}$

- `summary` occupies $S(\sqrt{u})$ space.
- `cluster` occupies $\sqrt{u} \cdot S(\sqrt{u})$ space.
- $O(1)$ additional information (e.g., the value of u)
- (Charge pointers to the inner structures)

$$S(u) \leq (1 + \sqrt{u}) \cdot S(\sqrt{u}) + O(1)$$

$$S(u) = O(1) \text{ when } u = O(1)$$

Space Usage

Let $S(u)$ be the space usage of a structure for the universe $\{0, \ldots, u-1\}$

- `summary` occupies $S(\sqrt{u})$ space.
- `cluster` occupies $\sqrt{u} \cdot S(\sqrt{u})$ space.
- $O(1)$ additional information (e.g., the value of u)
- (Charge pointers to the inner structures)

$$S(u) \leq (1+\sqrt{u}) \cdot S(\sqrt{u}) + O(1)$$

$$S(u) = O(1) \text{ when } u = O(1)$$

Solution: $S(u) = O(u)$

Space Usage

There is a c such that:

- $S(u) \leq c \quad$ for $u < 9$
- $S(u) \leq (1 + \sqrt{u}) \cdot S(\sqrt{u}) + c$

We want to show that $S(u) = O(u)$.

Space Usage

There is a c such that:

- $S(u) \leq c$ for $u < 9$
- $S(u) \leq (1 + \sqrt{u}) \cdot S(\sqrt{u}) + c$

We want to show that $S(u) = O(u)$.

Substitution method

Prove by induction that, for $u \geq 3$, $S(u) \leq c(u-2)$.

- Trivially true for $3 \leq u < 9$: $S(u) \leq c \leq c(u-2)$

Space Usage

There is a c such that:

- $S(u) \le c \quad$ for $u < 9$
- $S(u) \le (1 + \sqrt{u}) \cdot S(\sqrt{u}) + c$

We want to show that $S(u) = O(u)$.

Substitution method

Prove by induction that, for $u \ge 3$, $S(u) \le c(u-2)$.

- Trivially true for $3 \le u < 9$: $S(u) \le c \le c(u-2)$
- For $u \ge 9$: $S(u) \le (1 + \sqrt{u}) \cdot S(\sqrt{u}) + c$

Space Usage

There is a c such that:

- $S(u) \leq c \quad$ for $u < 9$
- $S(u) \leq (1 + \sqrt{u}) \cdot S(\sqrt{u}) + c$

We want to show that $S(u) = O(u)$.

Substitution method

Prove by induction that, for $u \geq 3$, $S(u) \leq c(u - 2)$.

- Trivially true for $3 \leq u < 9$: $S(u) \leq c \leq c(u - 2)$
- For $u \geq 9$: $S(u) \leq (1 + \sqrt{u}) \cdot c(\sqrt{u} - 2) + c$

Space Usage

There is a c such that:

- $S(u) \leq c \quad$ for $u < 9$
- $S(u) \leq (1 + \sqrt{u}) \cdot S(\sqrt{u}) + c$

We want to show that $S(u) = O(u)$.

Substitution method

Prove by induction that, for $u \geq 3$, $S(u) \leq c(u - 2)$.

- Trivially true for $3 \leq u < 9$: $S(u) \leq c \leq c(u - 2)$
- For $u \geq 9$: $S(u) \leq (2 + \sqrt{u}) \cdot c(\sqrt{u} - 2) + c$

Space Usage

There is a c such that:

- $S(u) \le c \quad$ for $u < 9$
- $S(u) \le (1 + \sqrt{u}) \cdot S(\sqrt{u}) + c$

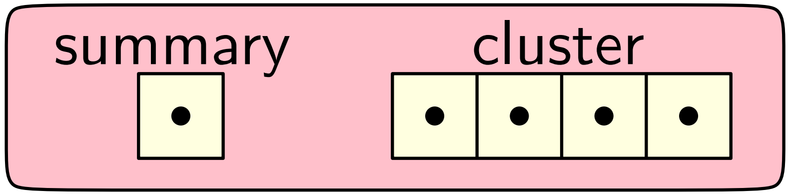
We want to show that $S(u) = O(u)$.

Substitution method

Prove by induction that, for $u \ge 3$, $S(u) \le c(u - 2)$.

- Trivially true for $3 \le u < 9$: $S(u) \le c \le c(u-2)$
- For $u \ge 9$: $S(u) \le (2 + \sqrt{u}) \cdot c(\sqrt{u} - 2) + c$
  $$\le c(u - 4) + c < c(u - 2).$$

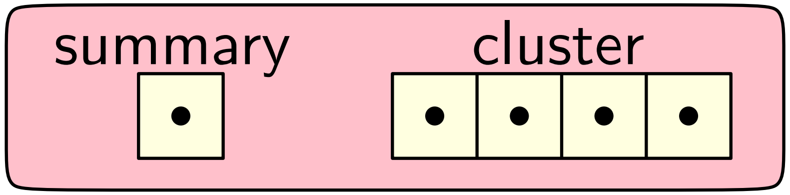
Implementing Find



Find(x):

- $h, \ell \leftarrow$ high(x), low(x)
- Return `cluster`$[h]$.find(ℓ)

Implementing Find

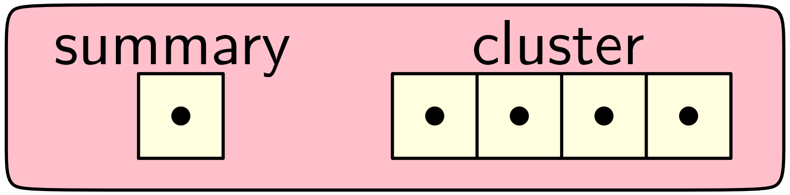


Find(x):

- $h, \ell \leftarrow$ high(x), low(x)
- Return `cluster`$[h]$.find(ℓ)

Time: $T(u) = T(\sqrt{u}) + O(1)$

Implementing Find



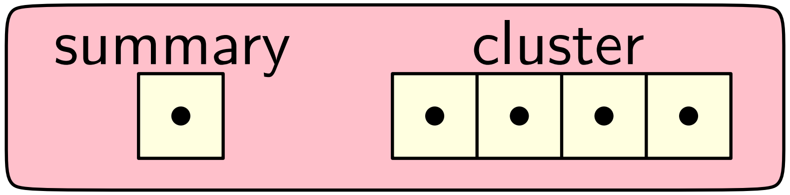
Find(x):

- $h, \ell \leftarrow$ high(x), low(x)
- Return `cluster`$[h]$.find(ℓ)

Time: $T(u) = T(\sqrt{u}) + O(1)$

$W(z) = T(2^z) = T(2^{z/2}) + O(1)$

Implementing Find



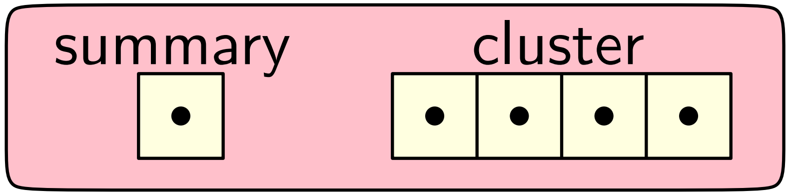
Find(x):

- $h, \ell \leftarrow$ high(x), low(x)
- Return `cluster`$[h]$.find(ℓ)

Time: $T(u) = T(\sqrt{u}) + O(1)$

$W(z) = T(2^z) = T(2^{z/2}) + O(1) = W(z/2) + O(1)$

Implementing Find



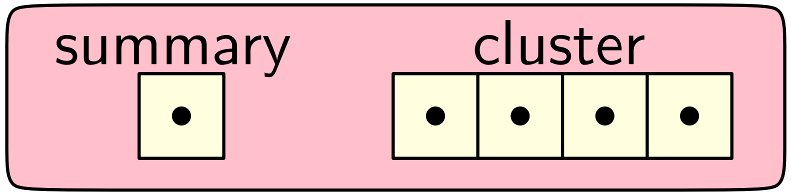
Find(x):

- $h, \ell \leftarrow$ high(x), low(x)
- Return `cluster`$[h]$.find(ℓ)

Time: $T(u) = T(\sqrt{u}) + O(1)$

$$W(z) = T(2^z) = T(2^{z/2}) + O(1) = W(z/2) + O(1) = \Theta(\log z)$$

Implementing Find



Find(x):

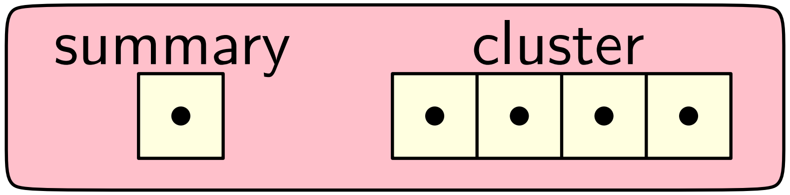
- $h, \ell \leftarrow$ high(x), low(x)
- Return `cluster`[h].find(ℓ)

Time: $T(u) = T(\sqrt{u}) + O(1)$

$$W(z) = T(2^z) = T(2^{z/2}) + O(1) = W(z/2) + O(1) = \Theta(\log z)$$

$$T(u) = W(\log u) = \Theta(\log \log u)$$ ☺

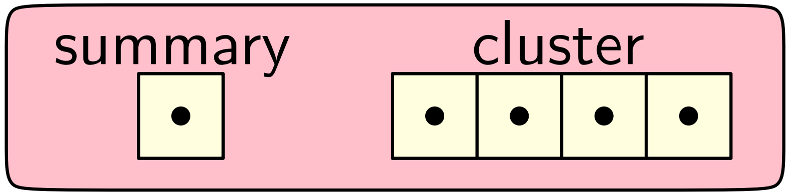
Implementing Insert



Insert(x):

- $h, \ell \leftarrow$ high(x), low(x)
- `summary`.insert(h)
- `cluster`[h].insert(ℓ)

Implementing Insert

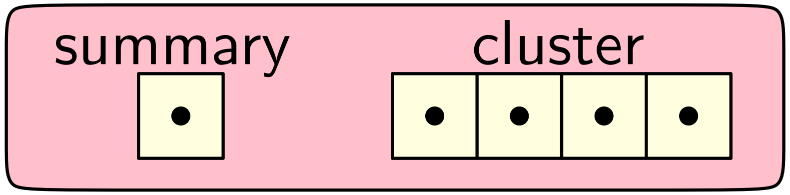


Insert(x):

- $h, \ell \leftarrow$ high(x), low(x)
- `summary`.insert(h)
- `cluster`[h].insert(ℓ)

Time: $T(u) = 2T(\sqrt{u}) + O(1)$

Implementing Insert



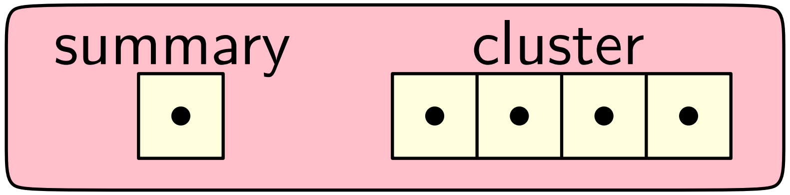
Insert(x):

- $h, \ell \leftarrow$ high(x), low(x)
- `summary`.insert(h)
- `cluster`[h].insert(ℓ)

Time: $T(u) = 2T(\sqrt{u}) + O(1)$

$W(z) = T(2^z) = 2T(2^{z/2}) + O(1)$

Implementing Insert



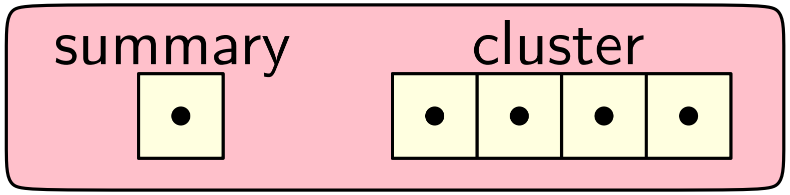
Insert(x):

- $h, \ell \leftarrow$ high(x), low(x)
- `summary`.insert(h)
- `cluster`[h].insert(ℓ)

Time: $T(u) = 2T(\sqrt{u}) + O(1)$

$W(z) = T(2^z) = 2T(2^{z/2}) + O(1) = 2W(z/2) + O(1)$

Implementing Insert



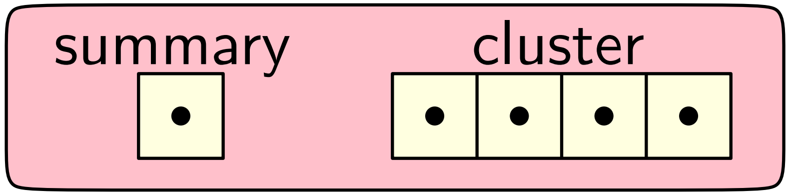
Insert(x):

- $h, \ell \leftarrow$ high(x), low(x)
- `summary`.insert(h)
- `cluster`[h].insert(ℓ)

Time: $T(u) = 2T(\sqrt{u}) + O(1)$

$$W(z) = T(2^z) = 2T(2^{z/2}) + O(1) = 2W(z/2) + O(1) = \Theta(z)$$

Implementing Insert



Insert(x):

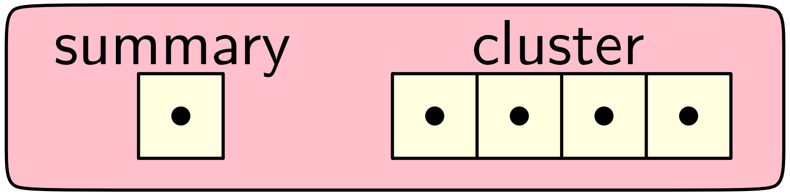
- $h, \ell \leftarrow$ high(x), low(x)
- `summary`.insert(h)
- `cluster`[h].insert(ℓ)

Time: $T(u) = 2T(\sqrt{u}) + O(1)$

$W(z) = T(2^z) = 2T(2^{z/2}) + O(1) = 2W(z/2) + O(1) = \Theta(z)$

$T(u) = W(\log u) = \Theta(\log u)$ ☹

Implementing Insert



Insert(x):

- $h, \ell \leftarrow$ high(x), low(x)
- `summary`.insert(h)
- `cluster`[h].insert(ℓ)

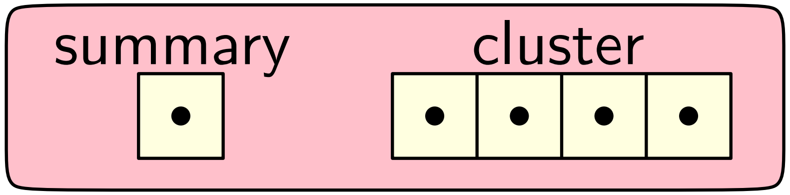
Too many recursive calls

Time: $T(u) = 2T(\sqrt{u}) + O(1)$

$$W(z) = T(2^z) = 2T(2^{z/2}) + O(1) = 2W(z/2) + O(1) = \Theta(z)$$

$$T(u) = W(\log u) = \Theta(\log u)$$ ☹

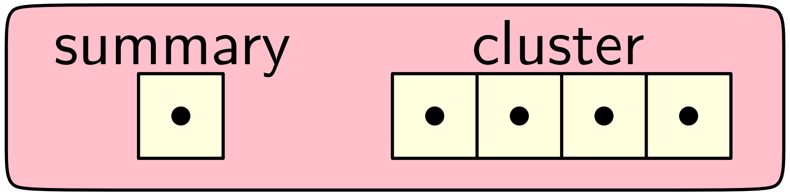
Implementing Successor



Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- $\ell' \leftarrow$ `cluster`[h].successor(ℓ) `// Check` x`'s cluster`
- If $\ell' \neq +\infty$: Return Idx(h, ℓ')

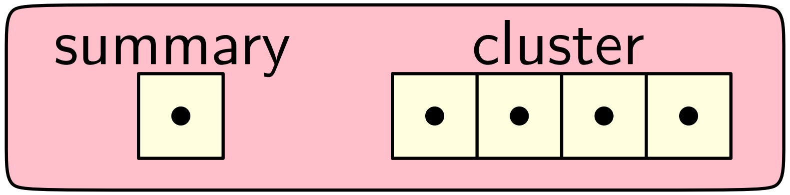
Implementing Successor



Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- $\ell' \leftarrow$ `cluster`$[h]$.successor(ℓ) `// Check` x`'s cluster`
- If $\ell' \neq +\infty$: Return Idx(h, ℓ')
- $h' \leftarrow$ `summary`.successor(h) `// Find next non-empty cluster`
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`$[h']$.successor($-\infty$)
  - Return Idx(h', ℓ')
- Return $+\infty$

Implementing Successor



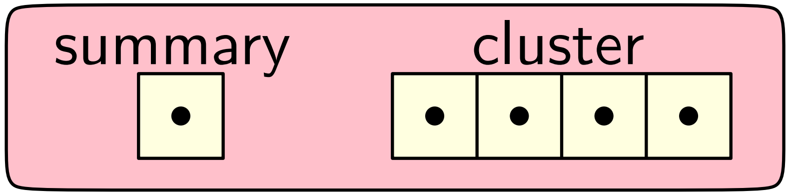
Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- $\ell' \leftarrow$ `cluster`[h].successor(ℓ) `// Check` x`'s cluster`
- If $\ell' \neq +\infty$: Return Idx(h, ℓ')
- $h' \leftarrow$ `summary`.successor(h) `// Find next non-empty cluster`
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`[h'].successor($-\infty$)
  - Return Idx(h', ℓ')
- Return $+\infty$

`We are cheating a little here.`

find(0) or successor(0)

Implementing Successor



Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- $\ell' \leftarrow$ `cluster`[h].successor(ℓ) `// Check` x`'s cluster`
- If $\ell' \neq +\infty$: Return Idx(h, ℓ')
- $h' \leftarrow$ `summary`.successor(h) `// Find next non-empty cluster`
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`[h'].successor($-\infty$)
  - Return Idx(h', ℓ')
- Return $+\infty$

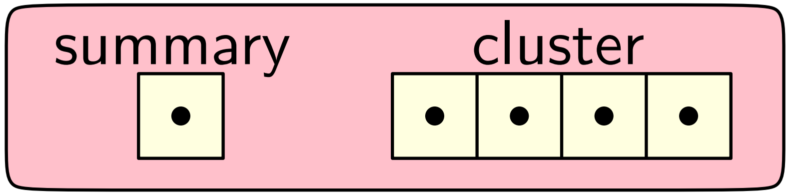
`We are cheating a little here.`

find(0) or successor(0)

Time: $T(u) = 3T(\sqrt{u}) + O(\log \log u)$

Implementing Successor

Successor(x):



- $h, \ell \leftarrow$ high(x), low(x)
- $\ell' \leftarrow$ `cluster`$[h]$.successor(ℓ) `// Check x's cluster`
- If $\ell' \neq +\infty$: Return Idx(h, ℓ')
- $h' \leftarrow$ `summary`.successor(h) `// Find next non-empty cluster`
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`$[h']$.successor($-\infty$)
  - Return Idx(h', ℓ')
- Return $+\infty$

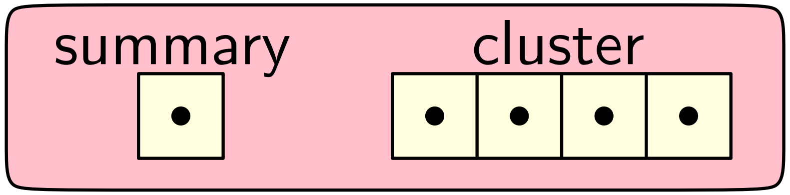
`We are cheating a little here.`

find(0) or successor(0)

Time: $T(u) = 3T(\sqrt{u}) + O(\log \log u)$

$W(z) = T(2^z) = 3T(2^{z/2}) + O(\log \log 2^z) = 3W(z/2) + O(\log z)$

Implementing Successor



Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- $\ell' \leftarrow$ `cluster`$[h]$.successor(ℓ)   `// Check` x`'s cluster`
- If $\ell' \neq +\infty$: Return Idx(h, ℓ')
- $h' \leftarrow$ `summary`.successor(h) `// Find next non-empty cluster`
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`$[h']$.successor($-\infty$)
  - Return Idx(h', ℓ')
- Return $+\infty$

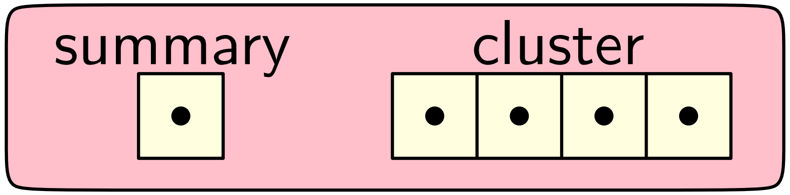
`We are cheating a little here.`

find(0) or successor(0)

Time: $T(u) = 3T(\sqrt{u}) + O(\log \log u)$

$W(z) = \Theta\left(z^{\log_2 3}\right)$

Implementing Successor



Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- $\ell' \leftarrow$ `cluster`[h].successor(ℓ) `// Check` x`'s cluster`
- If $\ell' \neq +\infty$: Return Idx(h, ℓ')
- $h' \leftarrow$ `summary`.successor(h) `// Find next non-empty cluster`
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`[h'].successor($-\infty$)
  - Return Idx(h', ℓ')
- Return $+\infty$

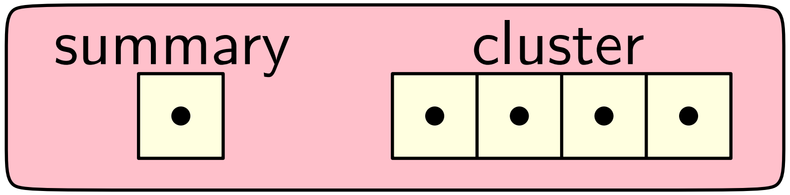
`We are cheating a little here.`

find(0) or successor(0)

Time: $T(u) = 3T(\sqrt{u}) + O(\log\log u) = \Theta\left((\log u)^{\log_2 3}\right)$ ☹

$W(z) = \Theta\left(z^{\log_2 3}\right)$

Implementing Successor



Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- $\ell' \leftarrow$ `cluster`$[h]$.successor(ℓ)
- If $\ell' \neq +\infty$: Return Idx(h, ℓ')
- $h' \leftarrow$ `summary`.successor(h)
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`$[h']$.successor($-\infty$)
  - Return Idx(h', ℓ')
- Return $+\infty$

Too many recursive calls

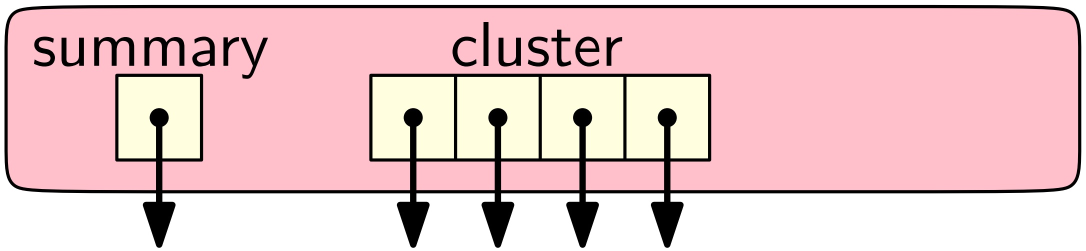
Time: $T(u) = 3T(\sqrt{u}) + O(\log\log u) = \Theta\left((\log u)^{\log_2 3}\right)$ ☹

$W(z) = \Theta\left(z^{\log_2 3}\right)$

Speeding Up the Operations (again)

Idea 1: Maintain the minimum separately!

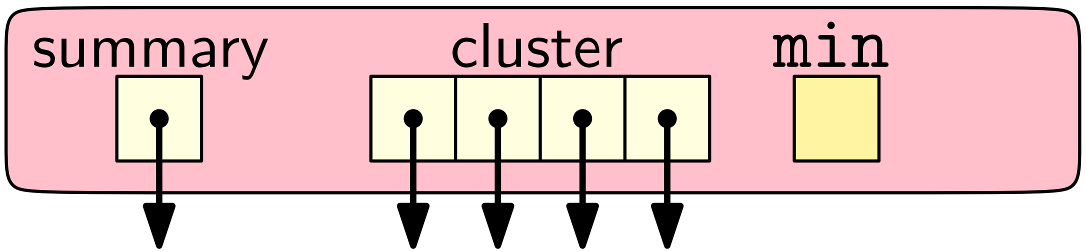
- Store a new field `min`
- `min` is no longer stored in `cluster`
- `min` does not affect `summary`
- (If S is empty, `min` $= +\infty$)



Speeding Up the Operations (again)

Idea 1: Maintain the minimum separately!

- Store a new field `min`
- `min` is no longer stored in `cluster`
- `min` does not affect `summary`
- (If S is empty, $\texttt{min} = +\infty$)



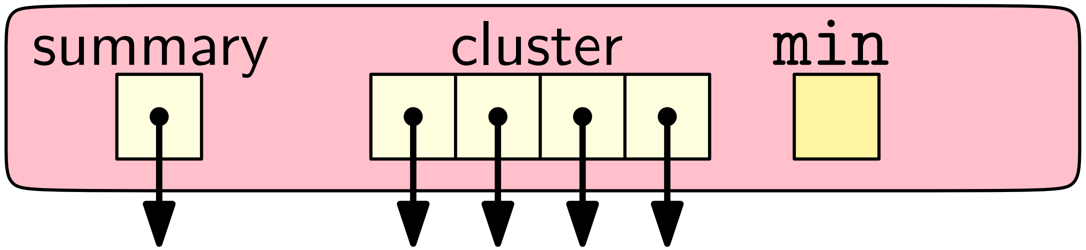
Speeding Up the Operations (again)

Idea 1: Maintain the minimum separately!

- Store a new field `min`
- `min` is no longer stored in `cluster`
- `min` does not affect `summary`
- (If S is empty, $\texttt{min} = +\infty$)

Idea 2: Keep track of the maximum.

- Store a new field `max`
- `max` is still stored in `cluster` and affects `summary` as usual



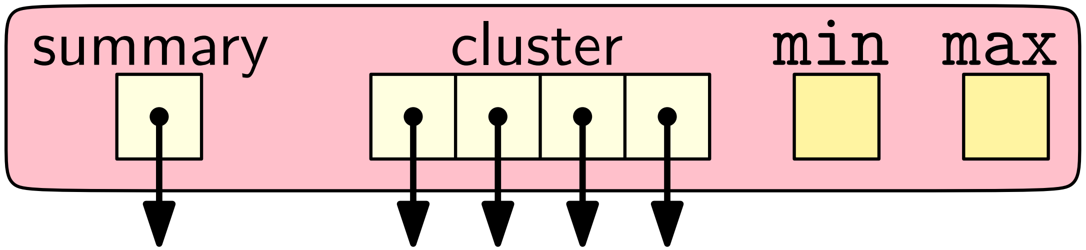
Speeding Up the Operations (again)

Idea 1: Maintain the minimum separately!

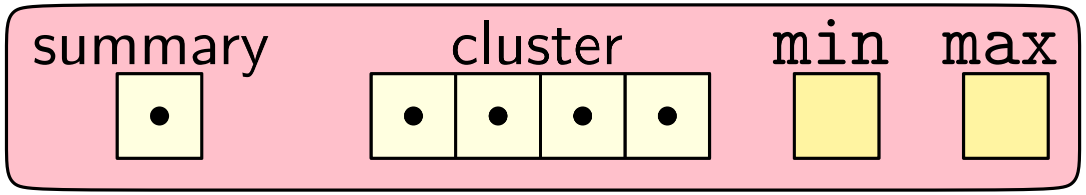
- Store a new field `min`
- `min` is no longer stored in `cluster`
- `min` does not affect `summary`
- (If S is empty, `min` $= +\infty$)

Idea 2: Keep track of the maximum.

- Store a new field `max`
- `max` is still stored in `cluster` and affects `summary` as usual



Implementing Find (again)



Find(x):

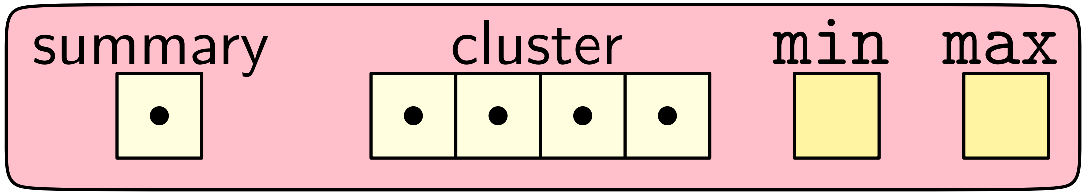
- $h, \ell \leftarrow$ high(x), low(x)
- Return `cluster`$[h]$.find(ℓ)

Time: $T(u) = T(\sqrt{u}) + O(1)$

$W(z) = T(2^z) = T(2^{z/2}) + O(1) = W(z/2) + O(1) = \Theta(\log z)$

$T(u) = W(\log u) = \Theta(\log \log u)$ ☺

Implementing Find (again)



Find(x):

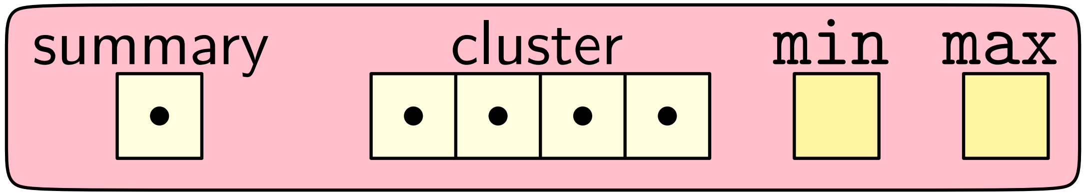
- If $x = \texttt{min}$: return $\top$
- $h, \ell \leftarrow$ high(x), low(x)
- Return $\texttt{cluster}[h]$.find(ℓ)

Time: $T(u) = T(\sqrt{u}) + O(1)$

$$W(z) = T(2^z) = T(2^{z/2}) + O(1) = W(z/2) + O(1) = \Theta(\log z)$$

$$T(u) = W(\log u) = \Theta(\log \log u)$$ ☺

Implementing Insert (again)

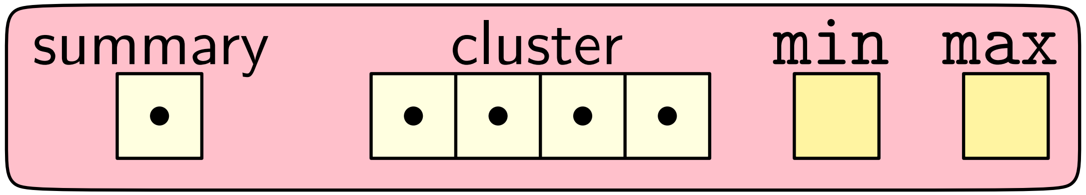


Insert(x):

- $h, \ell \leftarrow$ high(x), low(x)
- `summary`.insert(h)
- `cluster`[h].insert(ℓ)

Time: $T(u) = 2\,T(\sqrt{u}) + O(1)$ ☹

Implementing Insert (again)

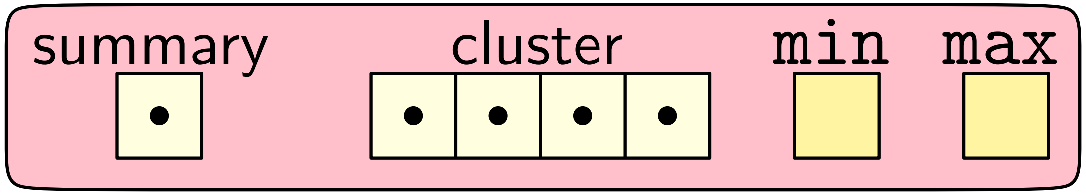


Insert(x):

- If $\texttt{min} = +\infty$: $\texttt{min} \leftarrow \texttt{max} \leftarrow x$. Return
- If $x > \texttt{max}$: $\texttt{max} \leftarrow x$
- If $x < \texttt{min}$: swap x and $\texttt{min}$
- $h, \ell \leftarrow$ high(x), low(x)
- $\texttt{summary}$.insert(h)
- $\texttt{cluster}[h]$.insert(ℓ)

Time: $T(u) = 2\,T(\sqrt{u}) + O(1)$ ☹

Implementing Insert (again)

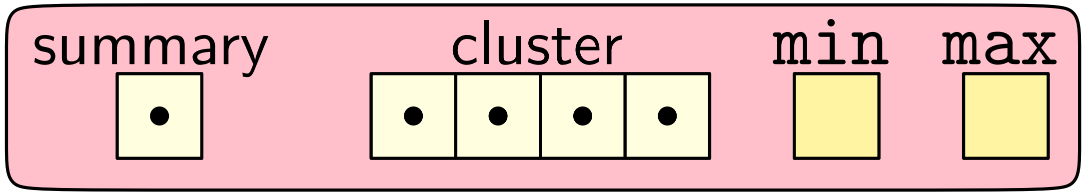


Insert(x):

- If $\texttt{min} = +\infty$: $\texttt{min} \leftarrow \texttt{max} \leftarrow x$. Return
- If $x > \texttt{max}$: $\texttt{max} \leftarrow x$
- If $x < \texttt{min}$: swap x and $\texttt{min}$
- $h, \ell \leftarrow$ high(x), low(x)
- If $\texttt{cluster}[h].\texttt{min} = +\infty$:
  - $\texttt{summary}$.insert(h)
- $\texttt{cluster}[h]$.insert(ℓ)

Time: $T(u) = 2\,T(\sqrt{u}) + O(1)$ ☹

Implementing Insert (again)

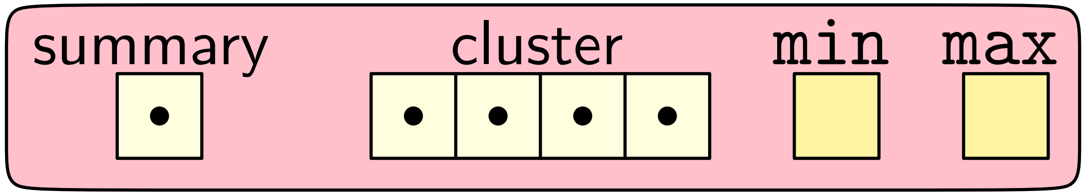


Insert(x):

- If `min` $= +\infty$: `min` $\leftarrow$ `max` $\leftarrow x$. Return
- If $x >$ `max`: `max` $\leftarrow x$
- If $x <$ `min`: swap x and `min`
- $h, \ell \leftarrow$ high(x), low(x)
- If `cluster`$[h]$.`min` $= +\infty$:
  - `summary`.insert(h) // If we execute this line...
- `cluster`$[h]$.insert(ℓ) // ...then this takes $O(1)$ time

Time: $T(u) = \cancel{2}T(\sqrt{u}) + O(1)$

Implementing Insert (again)

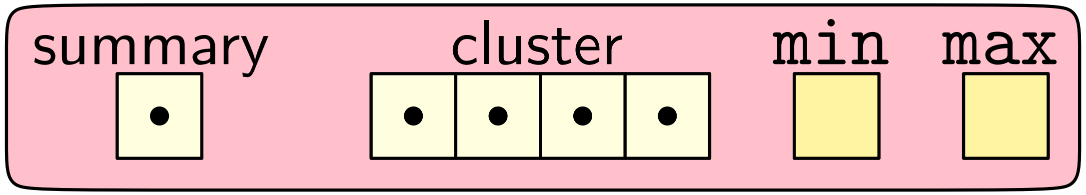


Insert(x):

- If `min` $= +\infty$: `min` $\leftarrow$ `max` $\leftarrow x$. Return
- If $x >$ `max`: `max` $\leftarrow x$
- If $x <$ `min`: swap x and `min`
- $h, \ell \leftarrow$ high(x), low(x)
- If `cluster`$[h]$.`min` $= +\infty$:
  - `summary`.insert(h)   `// If we execute this line...`
- `cluster`$[h]$.insert(ℓ)   `// ...then this takes` $O(1)$ `time`

Time: $T(u) = \cancel{2}T(\sqrt{u}) + O(1) \;= \Theta(\log \log u)$

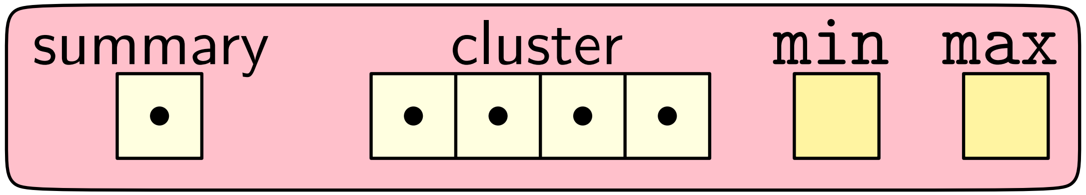
Implementing the Operations (again)



Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- $\ell' \leftarrow$ `cluster`$[h]$.successor(ℓ)
- If $\ell' \neq +\infty$: Return Idx(h, ℓ')
- $h' \leftarrow$ `summary`.successor(h)
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`$[h']$.successor($-\infty$)
  - Return idx(h', ℓ')
- Return $+\infty$

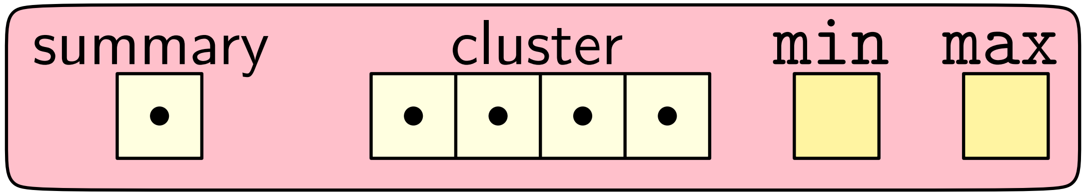
Implementing the Operations (again)



Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- If $\ell <$ `cluster`$[h]$.`max`:
  - Return idx(h, `cluster`$[h]$.successor(ℓ))
- $h' \leftarrow$ `summary`.successor(h)
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`$[h']$.successor($-\infty$)
  - Return idx(h', ℓ')
- Return $+\infty$

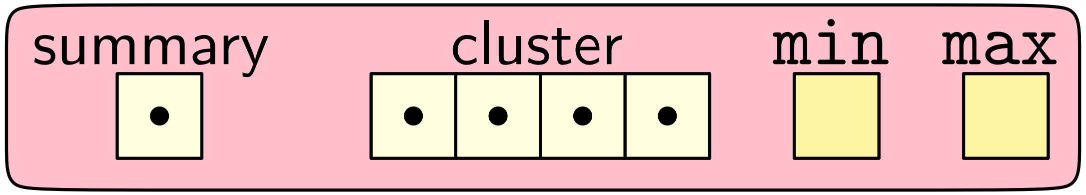
Implementing the Operations (again)



Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- If $\ell <$ `cluster`$[h]$.`max`:
  - Return idx(h, `cluster`$[h]$.successor(ℓ))
- $h' \leftarrow$ `summary`.successor(h)
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`$[h']$.`min`
  - Return idx(h', ℓ')
- Return $+\infty$

Implementing the Operations (again)



Successor(x):

- $h, \ell \leftarrow$ high(x), low(x)
- If $\ell <$ `cluster`$[h]$.`max`:
  - Return idx(h, `cluster`$[h]$.successor(ℓ))
- $h' \leftarrow$ `summary`.successor(h)
- If $h' \neq +\infty$:
  - $\ell' \leftarrow$ `cluster`$[h']$.`min`
  - Return idx(h', ℓ')
- Return $+\infty$

Time: $T(u) = T(\sqrt{u}) + O(1) = \Theta(\log \log u)$ ☺

The Dynamic Predecessor Problem

Goal:

Design a data stucture that maintains a *dynamic* set S of *integers* from a universe $\{0, \ldots, u-1\}$, supporting the following operations:

- Insert(x): Add x into S.
- Delete(x): Remove x from S.
- Find(x): report whether $x \in S$.
- Predecessor(x): return the largest integer $y < x$ in S (if any).
- Successor(x): return the smallest integer $y > x$ in S (if any).

Assume: $u = 2^w$, for some positive even integer w.

The Dynamic Predecessor Problem

Goal:

Design a data stucture that maintains a *dynamic* set S of *integers* from a universe $\{0, \ldots, u-1\}$, supporting the following operations:

- Insert(x): Add x into S.
- Delete(x): Remove x from S.
- Find(x): report whether $x \in S$.
- Predecessor(x): return the largest integer $y < x$ in S (if any).
- Successor(x): return the smallest integer $y > x$ in S (if any).

Assume: $u = 2^w$, for some positive even integer w.

Delete

Delete(x):

`// Last element?`

- If $x = \texttt{min} = \texttt{max}$: $\texttt{min} \leftarrow \texttt{max} \leftarrow +\infty$. Return

Delete

Delete(x):

// Last element?

- If $x = \texttt{min} = \texttt{max}$: $\texttt{min} \leftarrow \texttt{max} \leftarrow +\infty$. Return
- If $x = \texttt{min}$: // Are we deleting the minimum?
  - $\texttt{min} \leftarrow x \leftarrow \text{idx}(\texttt{summary.min}, \texttt{cluster}[\texttt{summary.min}].\texttt{min})$

Delete

Delete(x):

// Last element?

- If $x = \texttt{min} = \texttt{max}$: $\texttt{min} \leftarrow \texttt{max} \leftarrow +\infty$. Return
- If $x = \texttt{min}$: // Are we deleting the minimum?
  - $\texttt{min} \leftarrow x \leftarrow \text{idx}(\texttt{summary.min}, \texttt{cluster}[\texttt{summary.min}].\texttt{min})$
- $h, \ell \leftarrow \text{high}(x), \text{low}(x)$ // Actual deletion
- $\texttt{cluster}[h].\text{delete}(\ell)$
- If $\texttt{cluster}[h].\texttt{min} = +\infty$:
  - $\texttt{summary}.\text{delete}(h)$

Delete

Delete(x):

// Last element?

- If $x = \texttt{min} = \texttt{max}$: $\texttt{min} \leftarrow \texttt{max} \leftarrow +\infty$. Return
- If $x = \texttt{min}$: // Are we deleting the minimum?
  - $\texttt{min} \leftarrow x \leftarrow \text{idx}(\texttt{summary.min}, \texttt{cluster}[\texttt{summary.min}].\texttt{min})$
- $h, \ell \leftarrow \text{high}(x), \text{low}(x)$ // Actual deletion
- $\texttt{cluster}[h].\text{delete}(\ell)$ // ...then this took $O(1)$ time
- If $\texttt{cluster}[h].\texttt{min} = +\infty$:
  - $\texttt{summary}.\text{delete}(h)$ // If we execute this line...

Delete

Delete(x):

- If $x = \texttt{min} = \texttt{max}$: $\texttt{min} \leftarrow \texttt{max} \leftarrow +\infty$. Return `// Last element?`
- If $x = \texttt{min}$: `// Are we deleting the minimum?`
  - $\texttt{min} \leftarrow x \leftarrow \text{idx}(\texttt{summary.min}, \texttt{cluster}[\texttt{summary.min}].\texttt{min})$
- $h, \ell \leftarrow \text{high}(x), \text{low}(x)$ `// Actual deletion`
- $\texttt{cluster}[h].\text{delete}(\ell)$ `// ...then this took` $O(1)$ `time`
- If $\texttt{cluster}[h].\texttt{min} = +\infty$:
  - $\texttt{summary}.\text{delete}(h)$ `// If we execute this line...`

`// Recompute max from scratch`

- If $\texttt{summary.max} = +\infty$: $\texttt{max} \leftarrow \texttt{min}$. Return
- $\texttt{max} \leftarrow \text{idx}(\texttt{summary.max}, \texttt{cluster}[\texttt{summary.max}].\texttt{max})$

Recap

van Emde Boas trees: maintain a dynamic collection of integers from the universe $\{0, \ldots, u-1\}$

- Insert $O(\log\log u)$
- Delete $O(\log\log u)$
- Successor/Predecessor $O(\log\log u)$
- Min/Max $O(1)$
- Space $O(u)$ Not $O(n)$!!
- Supports satellite data

(n is the number of elements currently in the collection)

Recap

van Emde Boas trees: maintain a dynamic collection of integers from the universe $\{0, \ldots, u-1\}$

- Insert $O(\log \log u)$
- Delete $O(\log \log u)$
- Successor/Predecessor $O(\log \log u)$
- Min/Max $O(1)$
- Space $O(u)$ Not $O(n)$!!
- Supports satellite data

(n is the number of elements currently in the collection)

Space can be improved to $O(n)$.

(but the data structure becomes randomized & update times expected)

Reducing the Space Usage

Idea: Only store non-empty clusters!

- Replace `clusters` with a hash table
- Keys are high$(\cdot)$
- Values are pointers to the data-structures representing the clusters

$\ldots$ + additional tricks.

Reducing the Space Usage

Idea: Only store non-empty clusters!

- Replace `clusters` with a hash table
- Keys are high$(\cdot)$
- Values are pointers to the data-structures representing the clusters

$\ldots$ $+$ additional tricks.

Space: $O(n)$ (with *amortized expected* update times)

Reducing the Space Usage

Idea: Only store non-empty clusters!

- Replace `clusters` with a hash table
- Keys are high$(\cdot)$
- Values are pointers to the data-structures representing the clusters

$\ldots$ + additional tricks.

Space: $O(n)$ (with *amortized expected* update times)

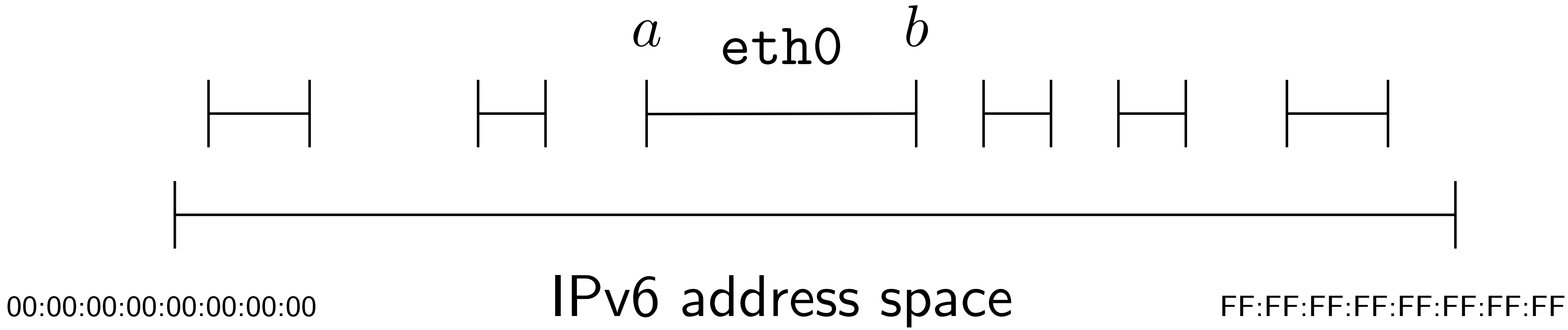
$\Omega(\log \log u)$ query time is needed when space is $O(n \operatorname{polylog} n)$

- Even when S is *static* and only Successor() is needed

In the cell probe model [M. Pătrașcu & M. Throup, STOC'06 & SODA'07]

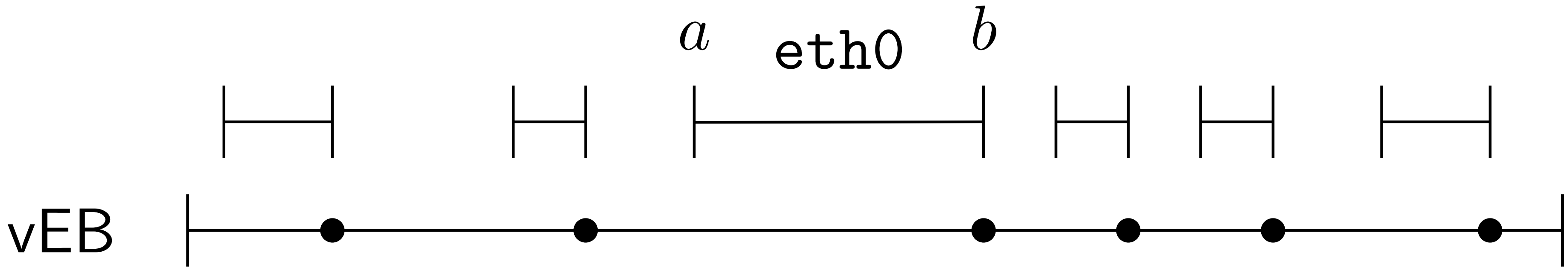
An Application

IP routing



An Application

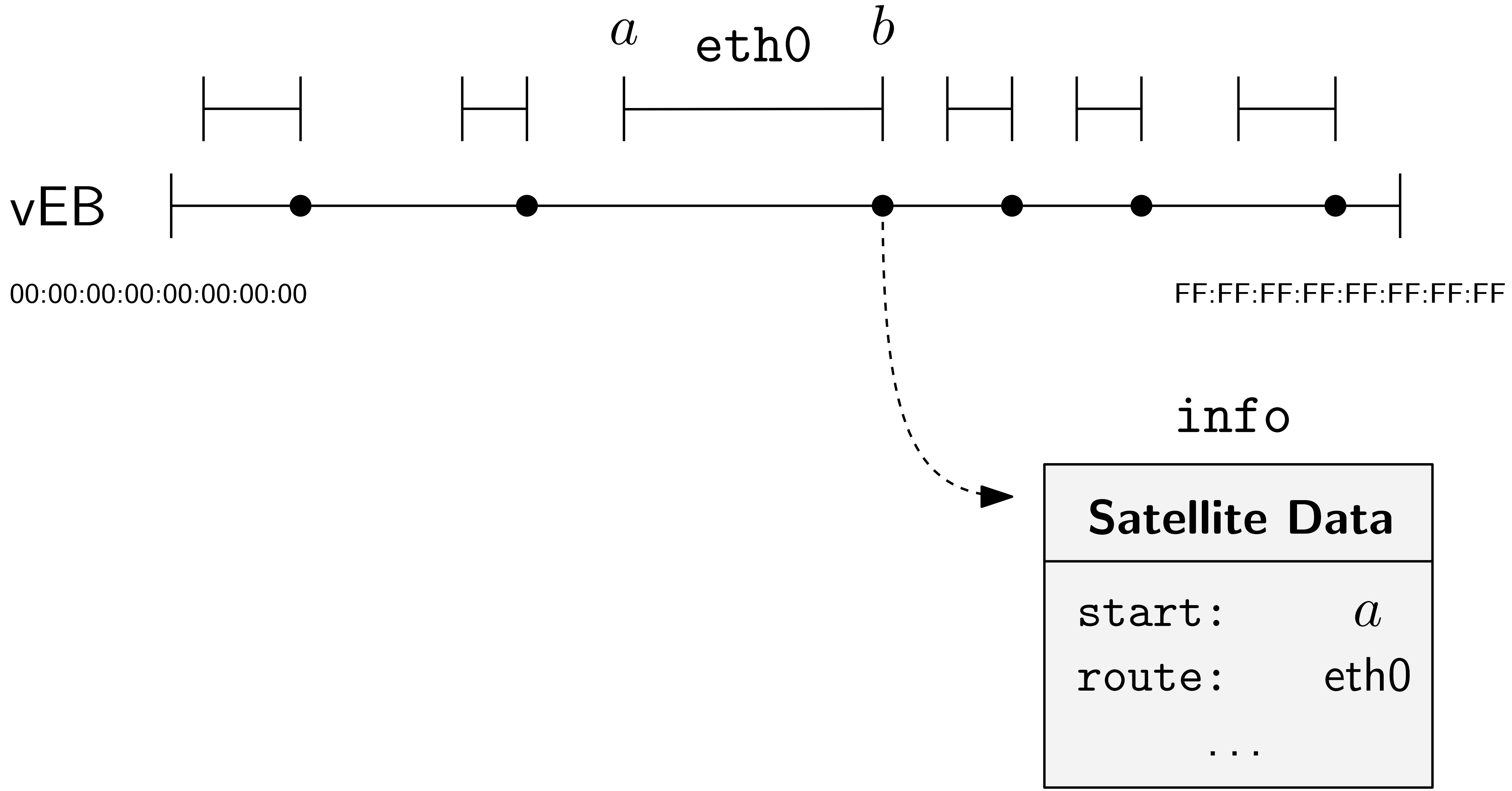
IP routing



00:00:00:00:00:00:00:00 IPv6 address space FF:FF:FF:FF:FF:FF:FF:FF

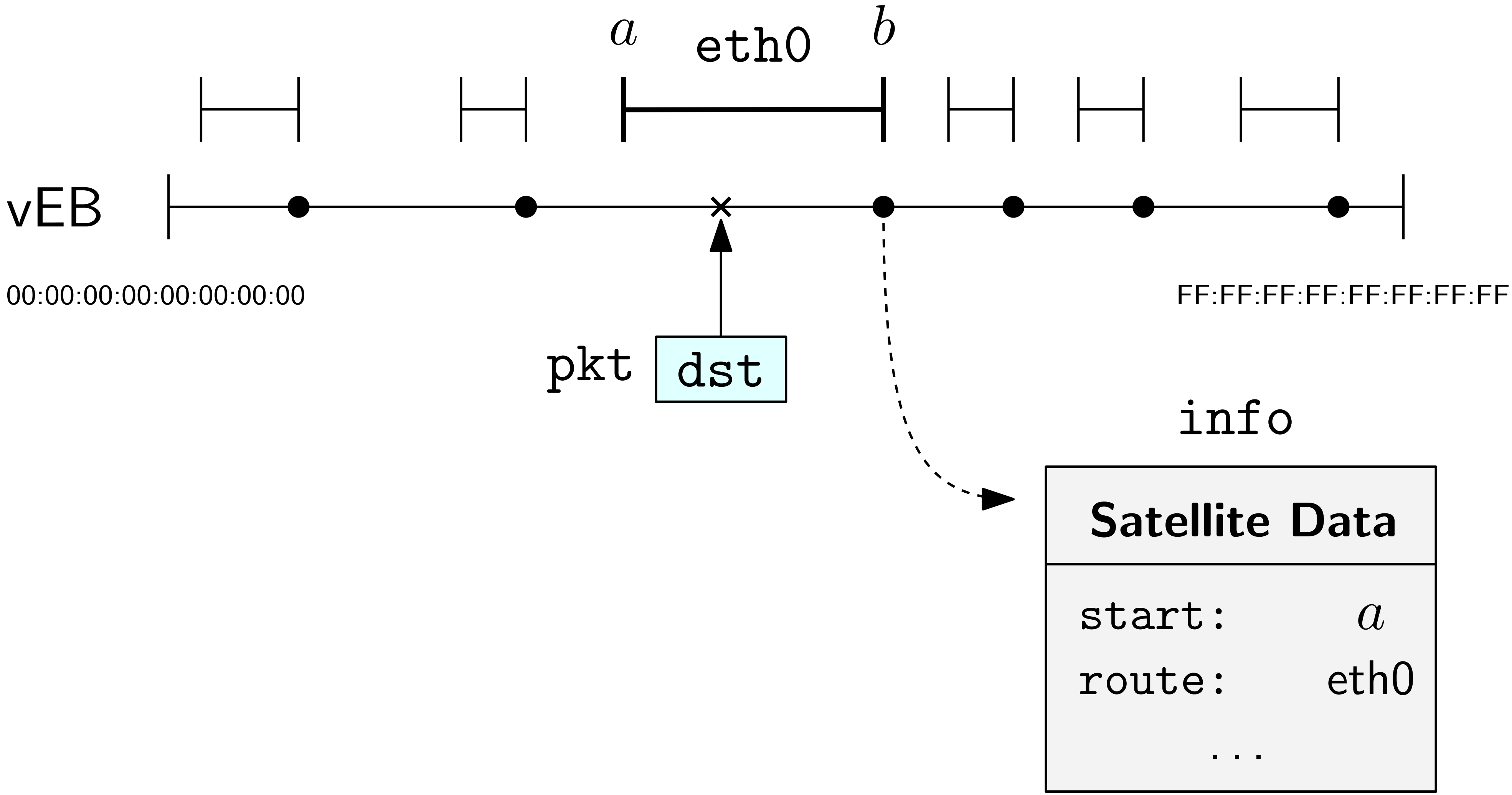
An Application

IP routing



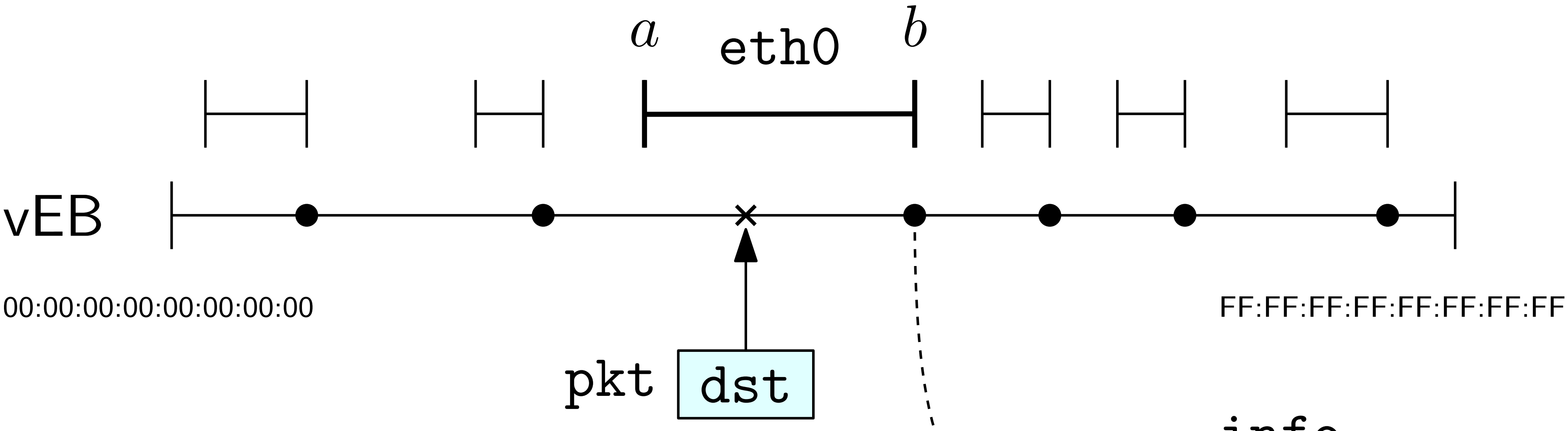
An Application

IP routing



An Application

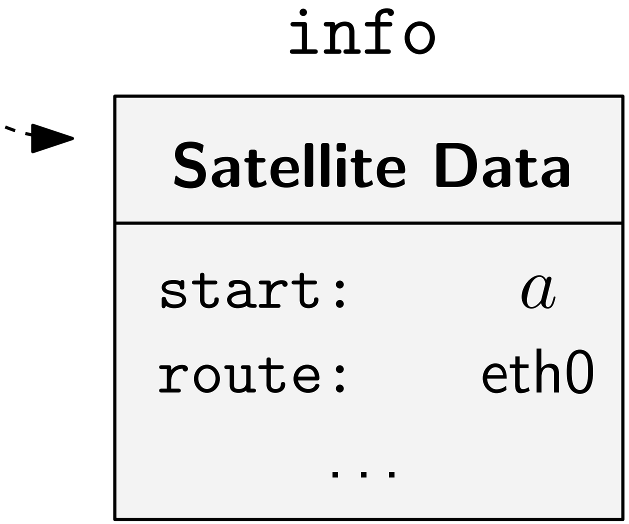
IP routing



$\texttt{info} \leftarrow \text{vEB.successor}(\texttt{dst})$

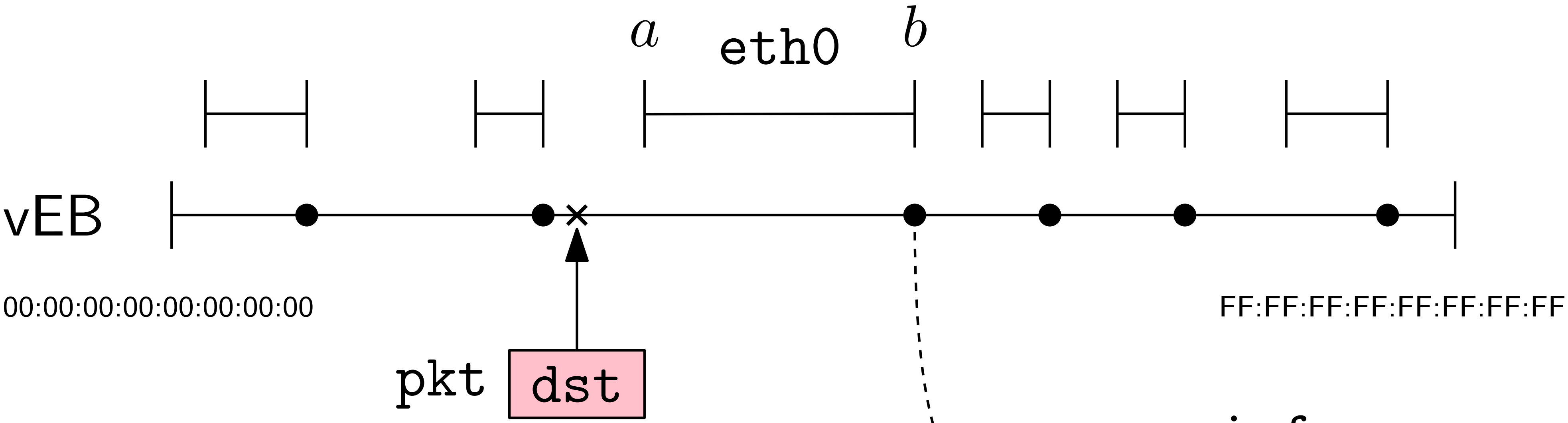
dst $\geq$ $\texttt{info.start}$?

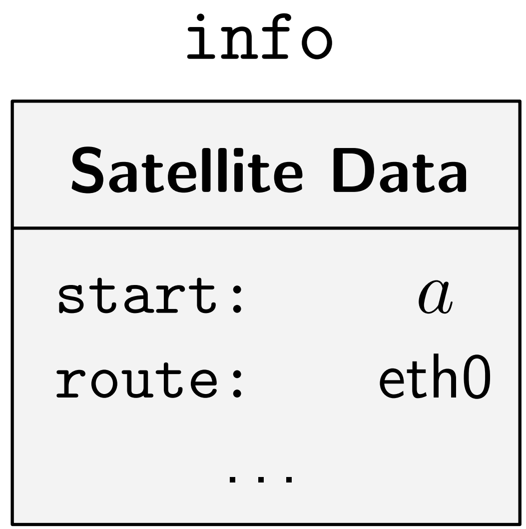
- **Yes**: send $\texttt{pkt}$ via $\texttt{info.route}$



An Application

IP routing





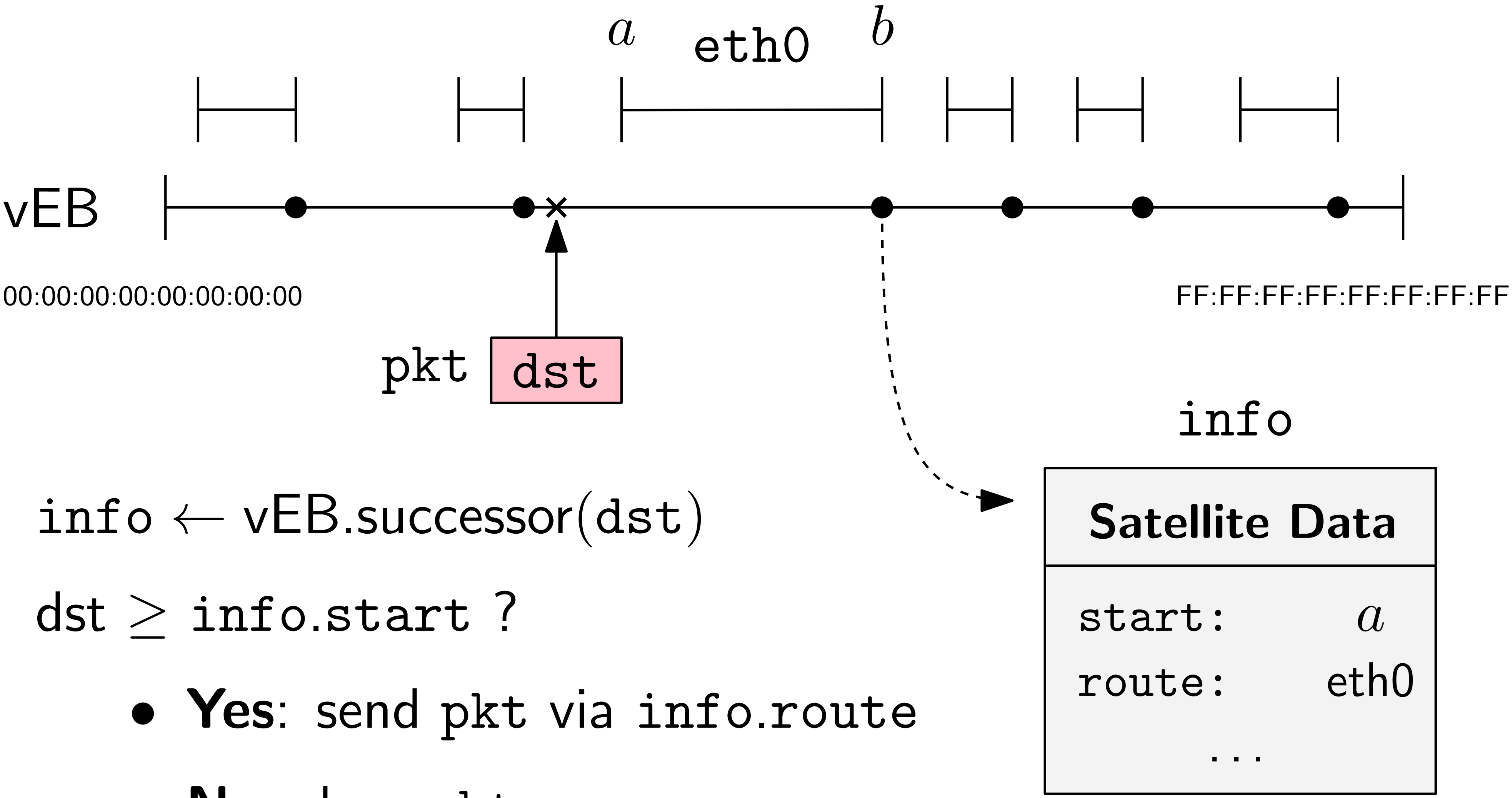
`info` $\leftarrow$ vEB.successor(`dst`)

dst $\geq$ `info.start` ?

- **Yes**: send `pkt` via `info.route`
- **No**: drop `pkt`

An Application

IP routing



info $\leftarrow$ vEB.successor(dst)

dst $\geq$ info.start ?

- **Yes**: send pkt via info.route
- **No**: drop pkt

Universe size: $u = 2^{128}$ $\log \log u = \log 128 = 7$