Lecture 4

Solving Recurrences (contd.), Hashing

Another Recurrence

Another Recurrence

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 3.T(n/4) + c'n^2, & \text{otherwise} \end{cases}$$

Another Recurrence

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 3.T(n/4) + c'n^2, & \text{otherwise} \end{cases}$$

Computing $T(n)$:

Another Recurrence

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 3.T(n/4) + c'n^2, & \text{otherwise} \end{cases}$$

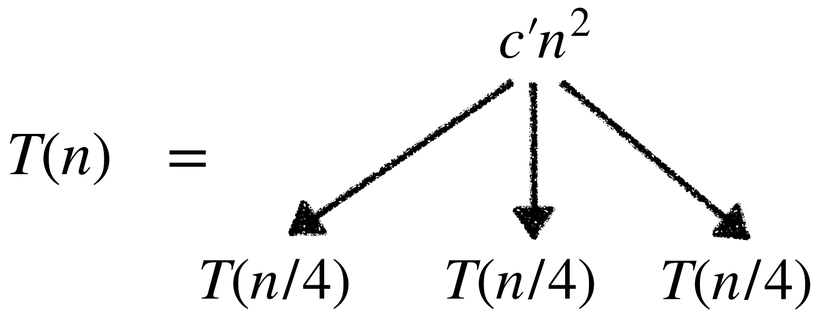
Computing $T(n)$:

$T(n)$

Another Recurrence

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 3.T(n/4) + c'n^2, & \text{otherwise} \end{cases}$$

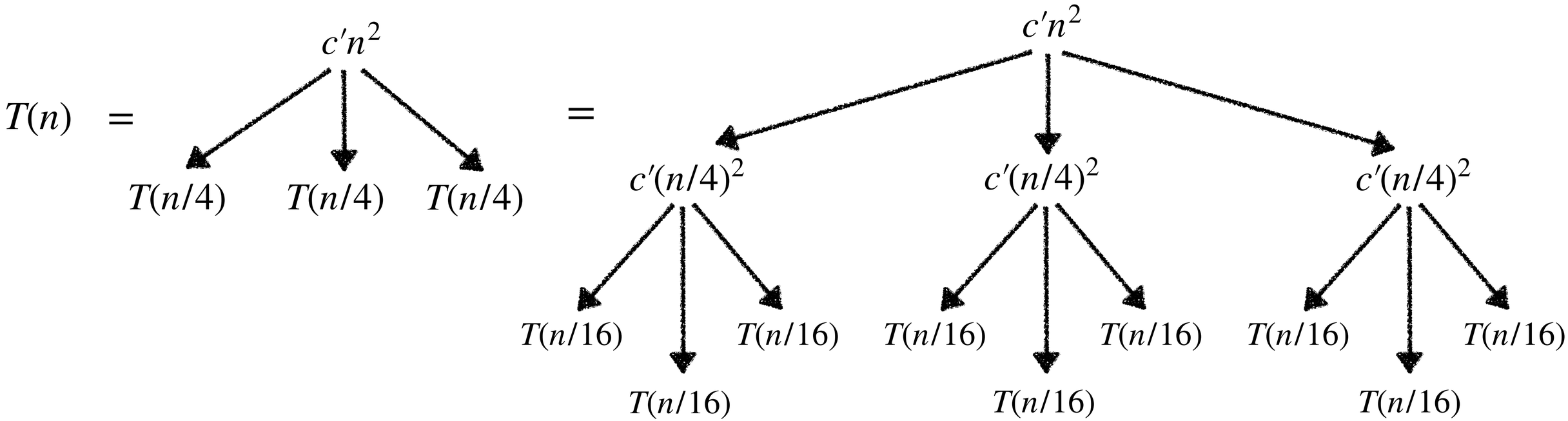
Computing $T(n)$:



Another Recurrence

$$T(n) = \begin{cases} c, & \text{if } n \leq 1 \\ 3.T(n/4) + c'n^2, & \text{otherwise} \end{cases}$$

Computing $T(n)$:



Another Recurrence

Another Recurrence

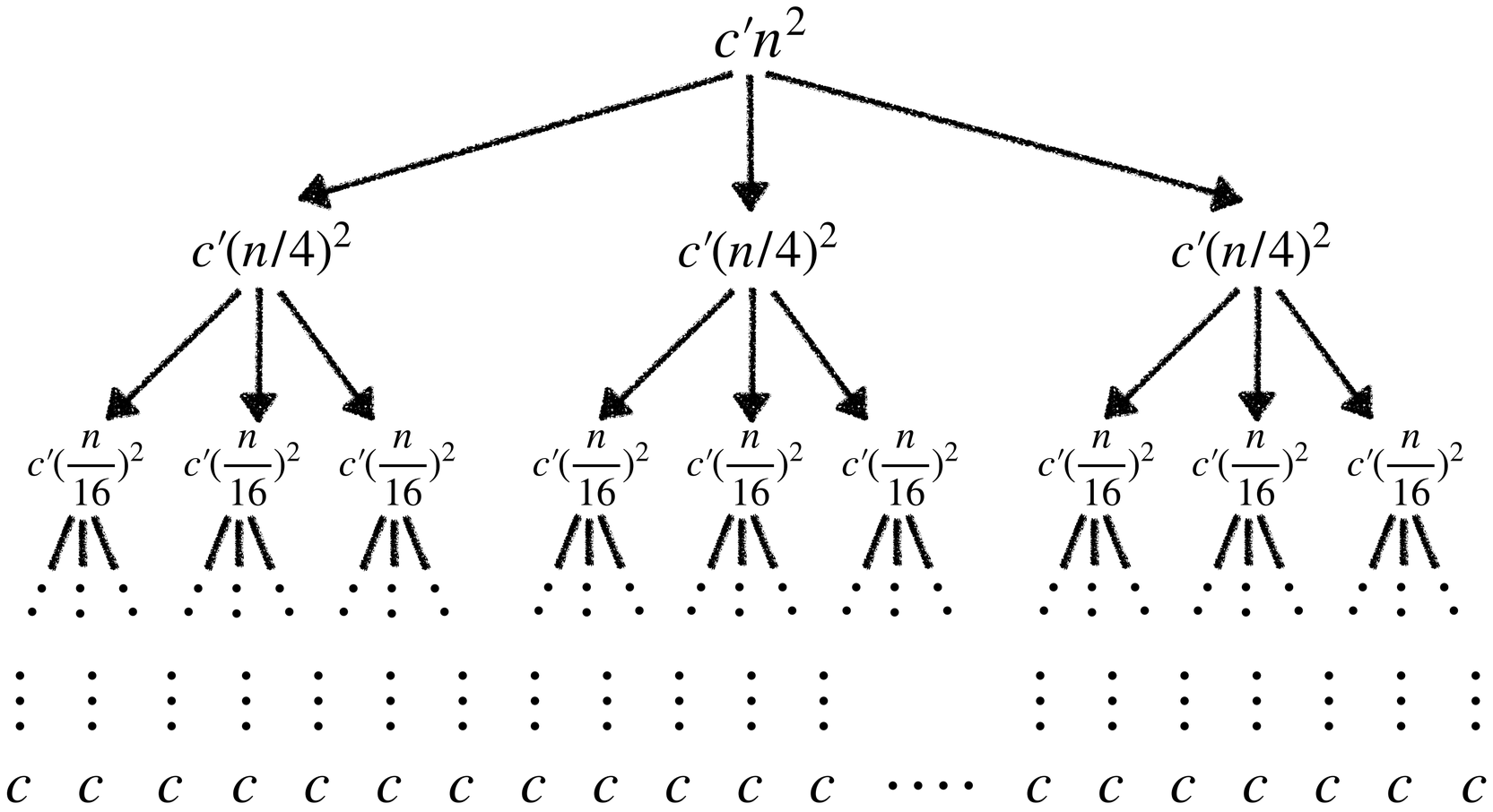
$T(n)$

Another Recurrence

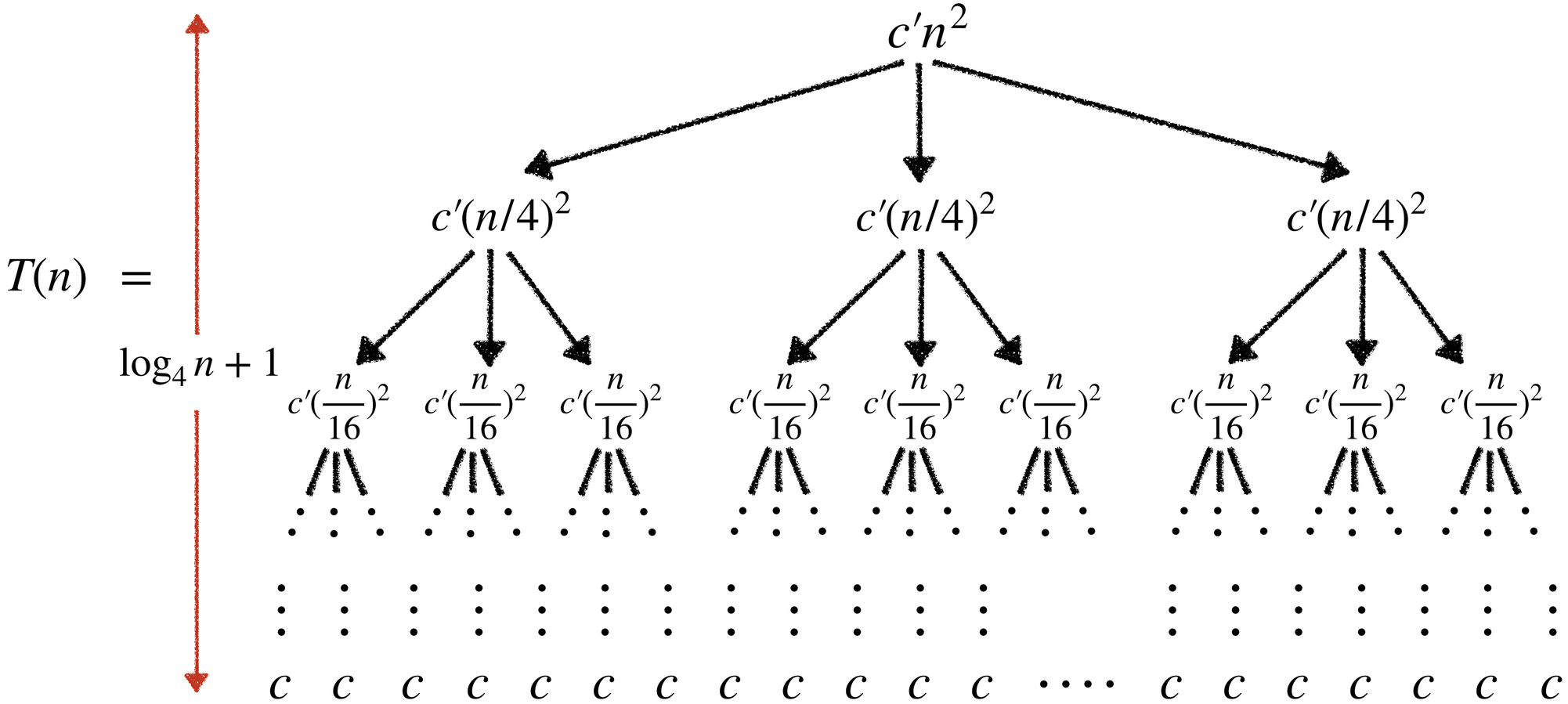
$T(n) \quad =$

Another Recurrence

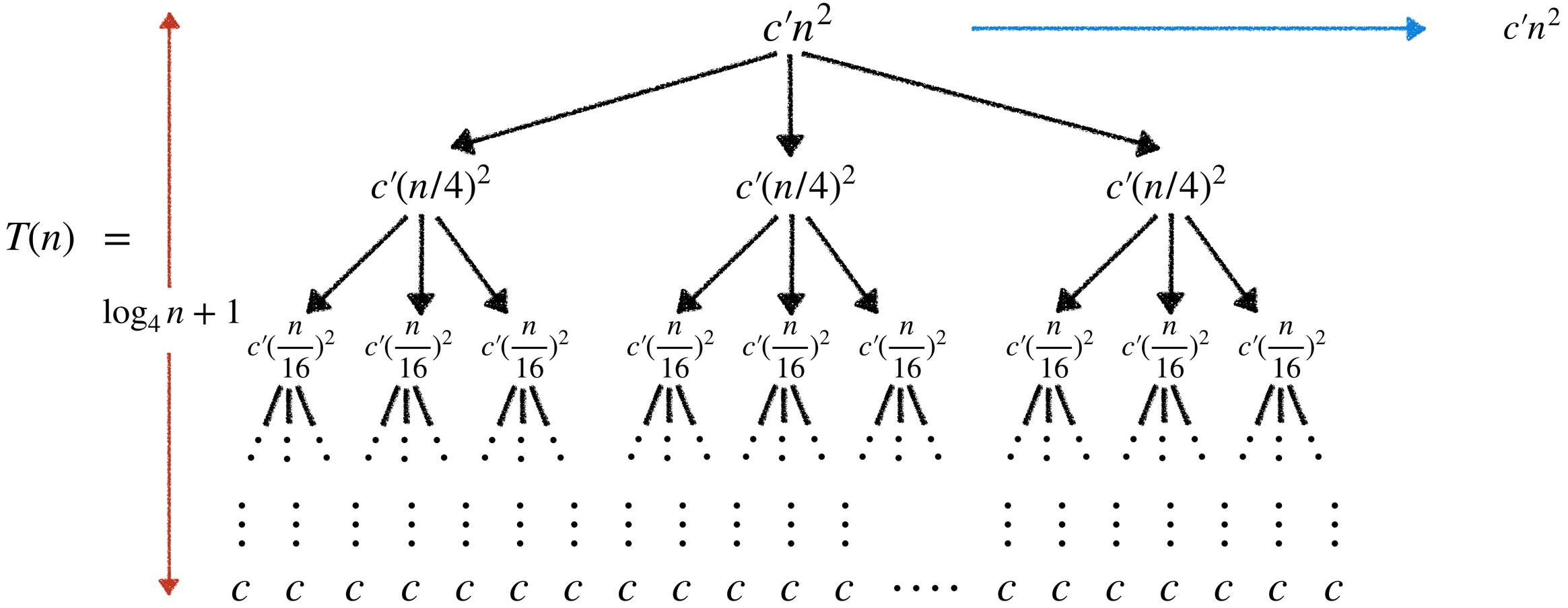
$T(n) =$



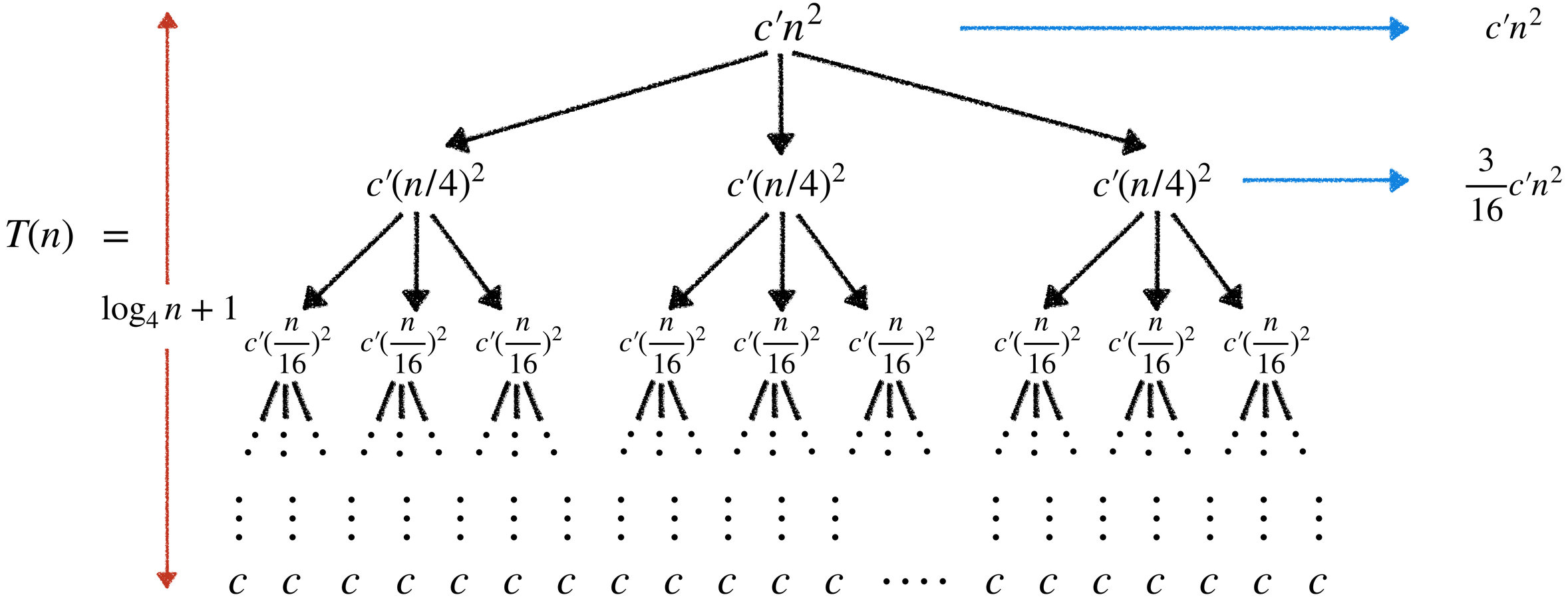
Another Recurrence



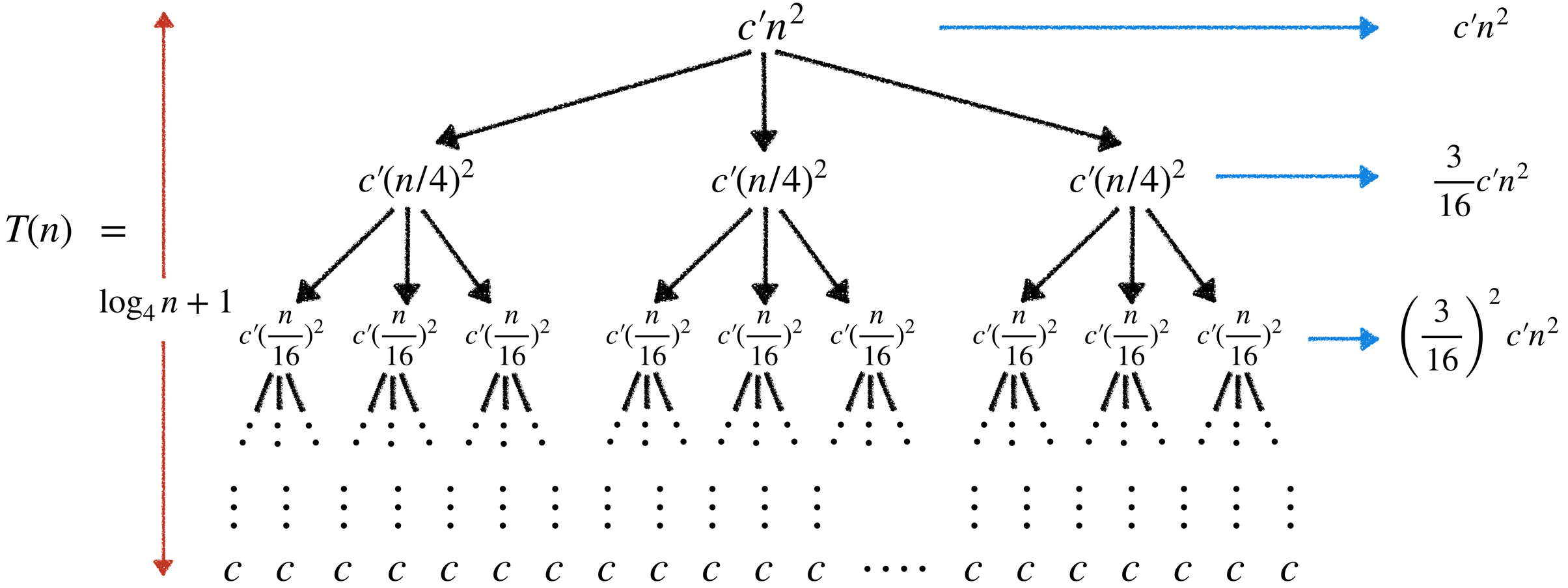
Another Recurrence



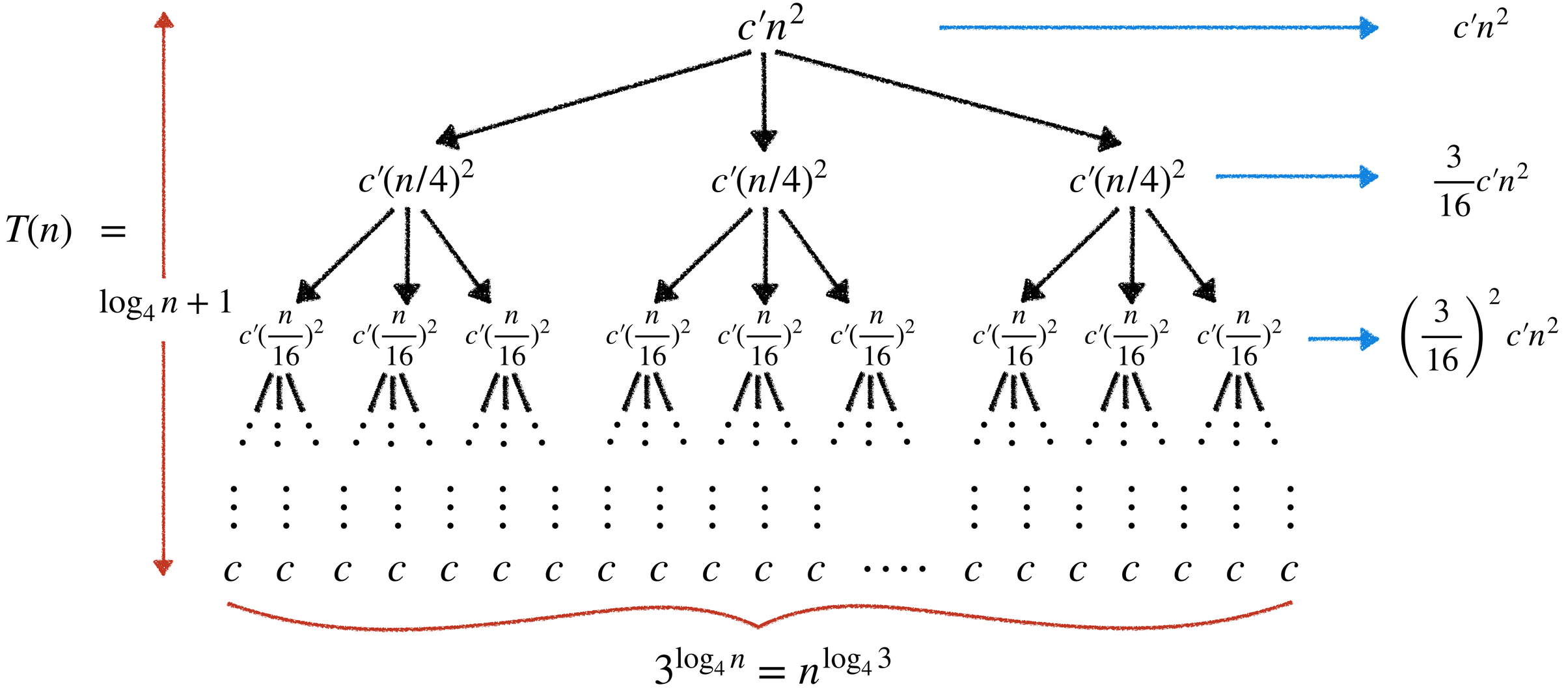
Another Recurrence



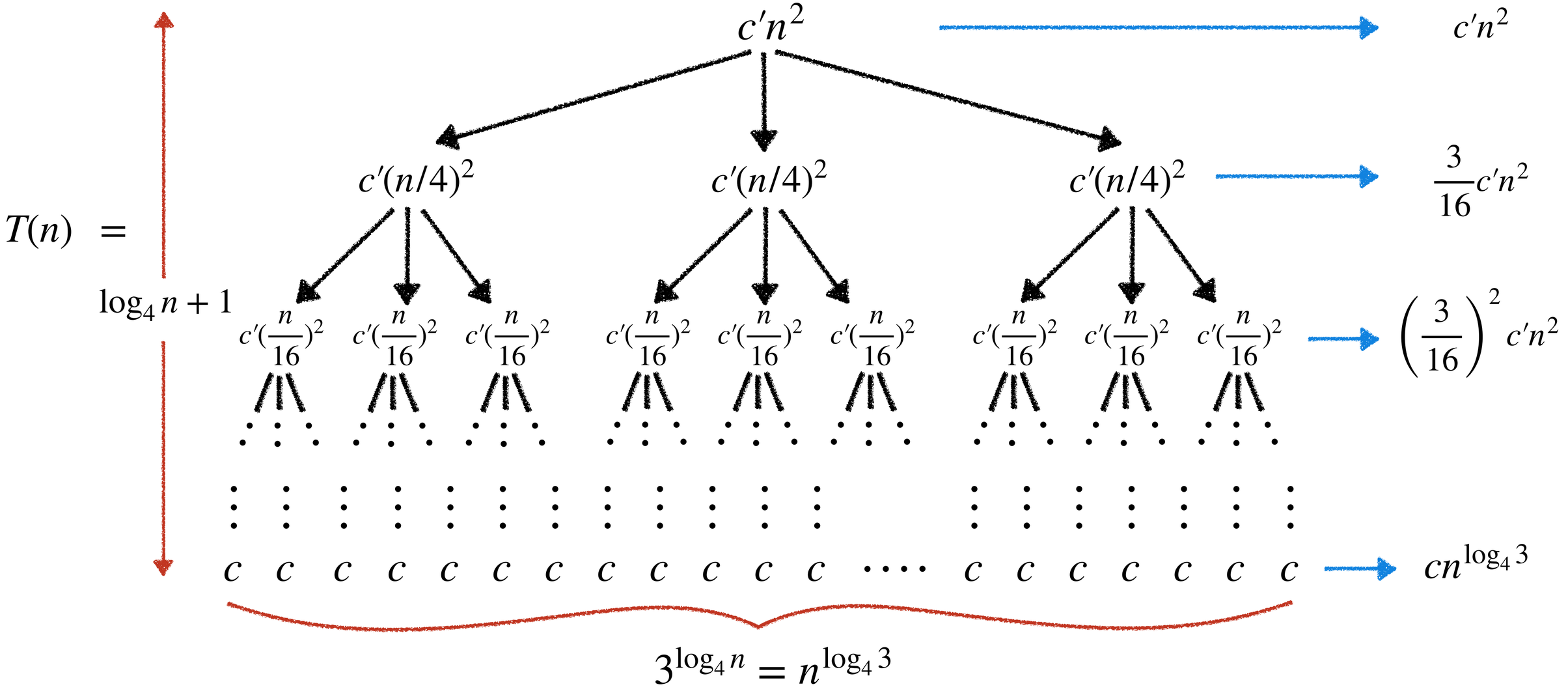
Another Recurrence



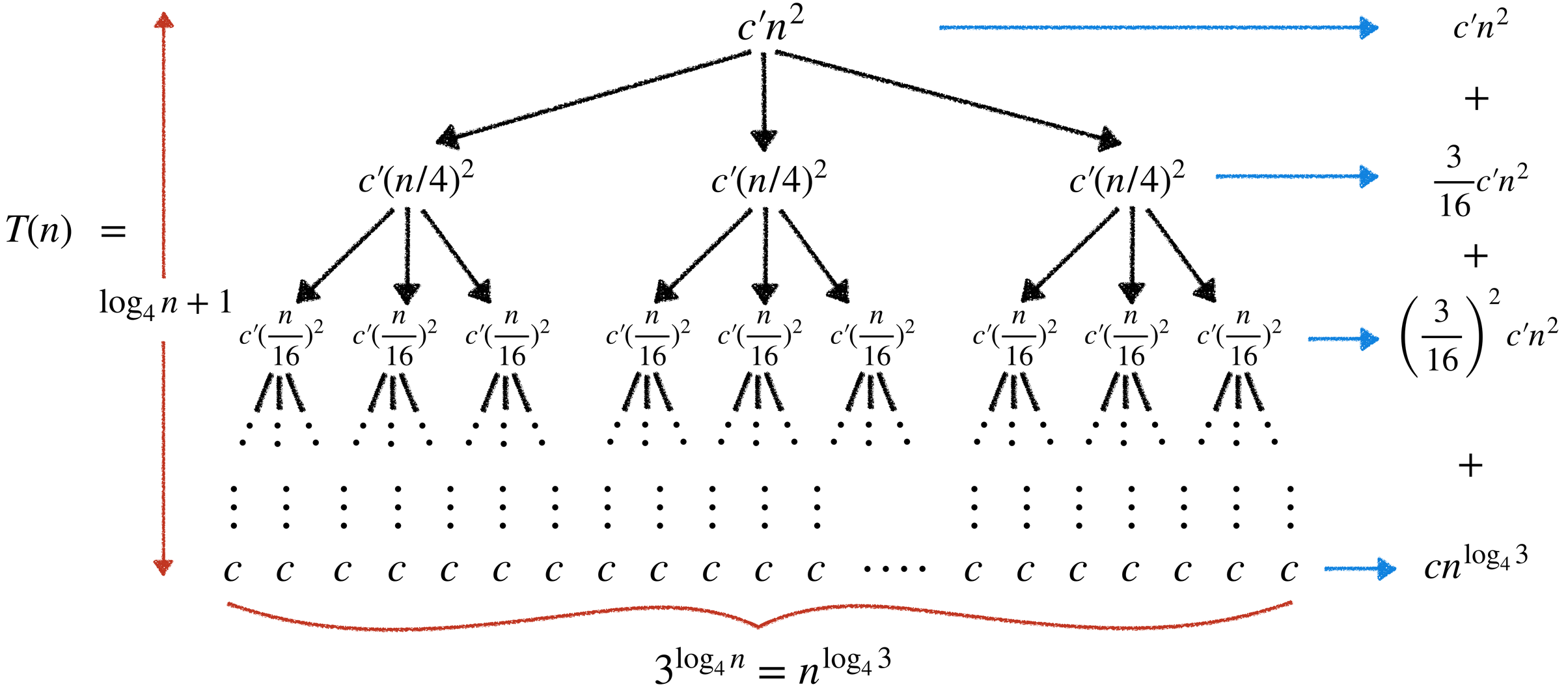
Another Recurrence



Another Recurrence



Another Recurrence



Another Recurrence

Another Recurrence

$$T(n) = c'n^2 + \frac{3}{16}c'n^2 + \left(\frac{3}{16}\right)^2 c'n^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n} c'n^2 + cn^{\log_4 3}$$

Another Recurrence

$$T(n) = c'n^2 + \frac{3}{16}c'n^2 + \left(\frac{3}{16}\right)^2 c'n^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n} c'n^2 + cn^{\log_4 3}$$

$$= c'n^2 \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i + cn^{\log_4 3}$$

Another Recurrence

$$T(n) = c'n^2 + \frac{3}{16}c'n^2 + \left(\frac{3}{16}\right)^2 c'n^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n} c'n^2 + cn^{\log_4 3}$$

$$= c'n^2 \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i + cn^{\log_4 3}$$

$$< c'n^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + cn^{\log_4 3}$$

Another Recurrence

$$T(n) = c'n^2 + \frac{3}{16}c'n^2 + \left(\frac{3}{16}\right)^2 c'n^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n} c'n^2 + cn^{\log_4 3}$$

$$= c'n^2 \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i + cn^{\log_4 3}$$

$$< c'n^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + cn^{\log_4 3}$$

$$= c'n^2 \left(\frac{16}{13}\right) + cn^{\log_4 3}$$

Another Recurrence

$$T(n) = c'n^2 + \frac{3}{16}c'n^2 + \left(\frac{3}{16}\right)^2 c'n^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n} c'n^2 + cn^{\log_4 3}$$

$$= c'n^2 \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i + cn^{\log_4 3}$$

$$< c'n^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + cn^{\log_4 3}$$

$$= c'n^2 \left(\frac{16}{13}\right) + cn^{\log_4 3} \quad \left(\because a + ar + ar^2 + \ldots = \frac{a}{1-r}, \text{ when } |r| < 1\right)$$

Another Recurrence

$$T(n) = c'n^2 + \frac{3}{16}c'n^2 + \left(\frac{3}{16}\right)^2 c'n^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n} c'n^2 + cn^{\log_4 3}$$

$$= c'n^2 \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i + cn^{\log_4 3}$$

$$< c'n^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + cn^{\log_4 3}$$

$$= c'n^2 \left(\frac{16}{13}\right) + cn^{\log_4 3} \qquad \left(\because a + ar + ar^2 + \ldots = \frac{a}{1-r}, \text{ when } |r| < 1\right)$$

$$= O(n^2)$$

Hashing: Introduction

Hashing: Introduction

Hashing is typically used to maintain a dynamic set that supports three operations:

Hashing: Introduction

Set of elements where every element has a key.

Hashing is typically used to maintain a dynamic set that supports three operations:

Hashing: Introduction

Set of elements where every element has a key.

Hashing is typically used to maintain a dynamic set that supports three operations:

- Insert an element.

Hashing: Introduction

Set of elements where every element has a key.

Hashing is typically used to maintain a dynamic set that supports three operations:

- Insert an element.
- Search for an element with the key k.

Hashing: Introduction

Set of elements where every element has a key.

Hashing is typically used to maintain a dynamic set that supports three operations:

- Insert an element.
- Search for an element with the key k.
- Delete an element.

Direct-address Tables

Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Then, we can use an array or a direct-address table

Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Then, we can use an array or a direct-address table, denoted by $T[0:m-1]$, to represent

Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

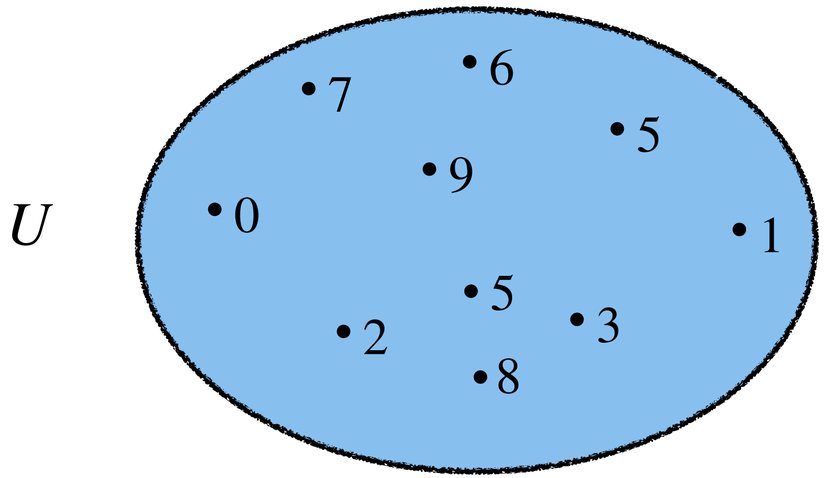
Then, we can use an array or a direct-address table, denoted by $T[0 : m-1]$, to represent the dynamic set, in which element x resides in $T[x\,.\,key]$.

Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Then, we can use an array or a direct-address table, denoted by $T[0 : m-1]$, to represent the dynamic set, in which element x resides in $T[x\,.\,key]$.

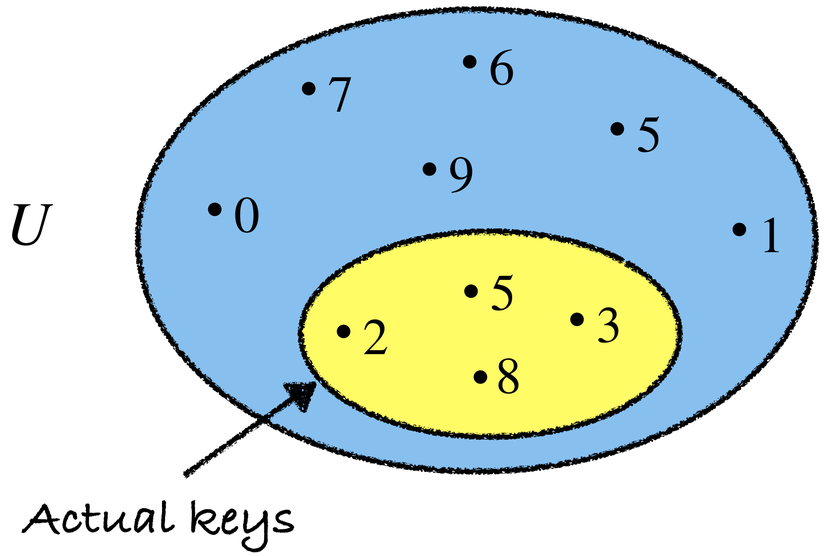


Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Then, we can use an array or a direct-address table, denoted by $T[0 : m-1]$, to represent the dynamic set, in which element x resides in $T[x\,.\,key]$.

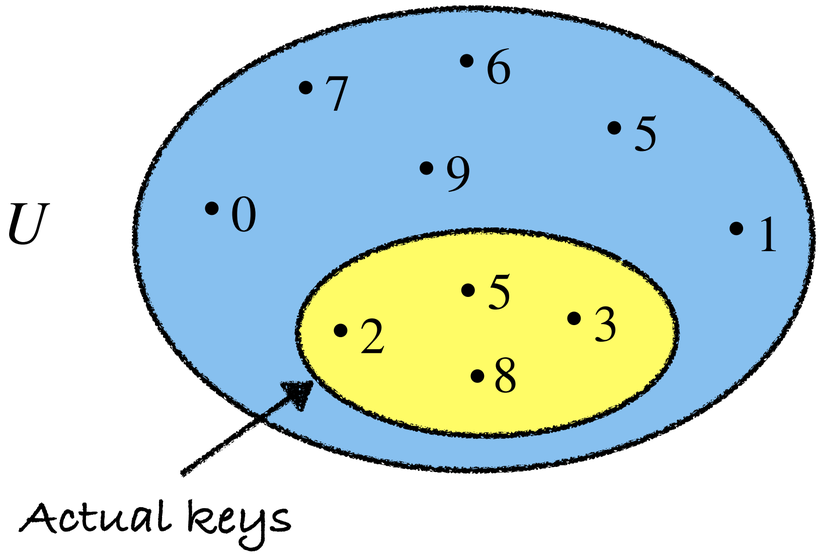


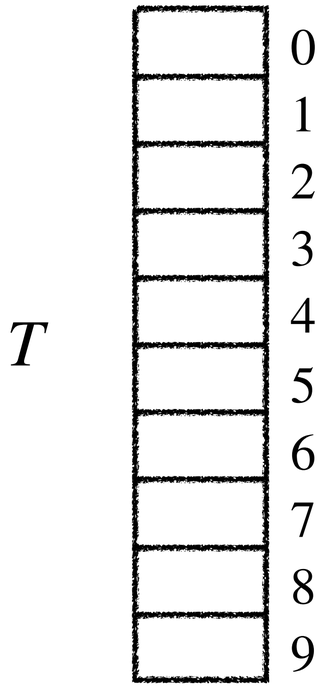
Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Then, we can use an array or a direct-address table, denoted by $T[0 : m-1]$, to represent the dynamic set, in which element x resides in $T[x\,.\,key]$.



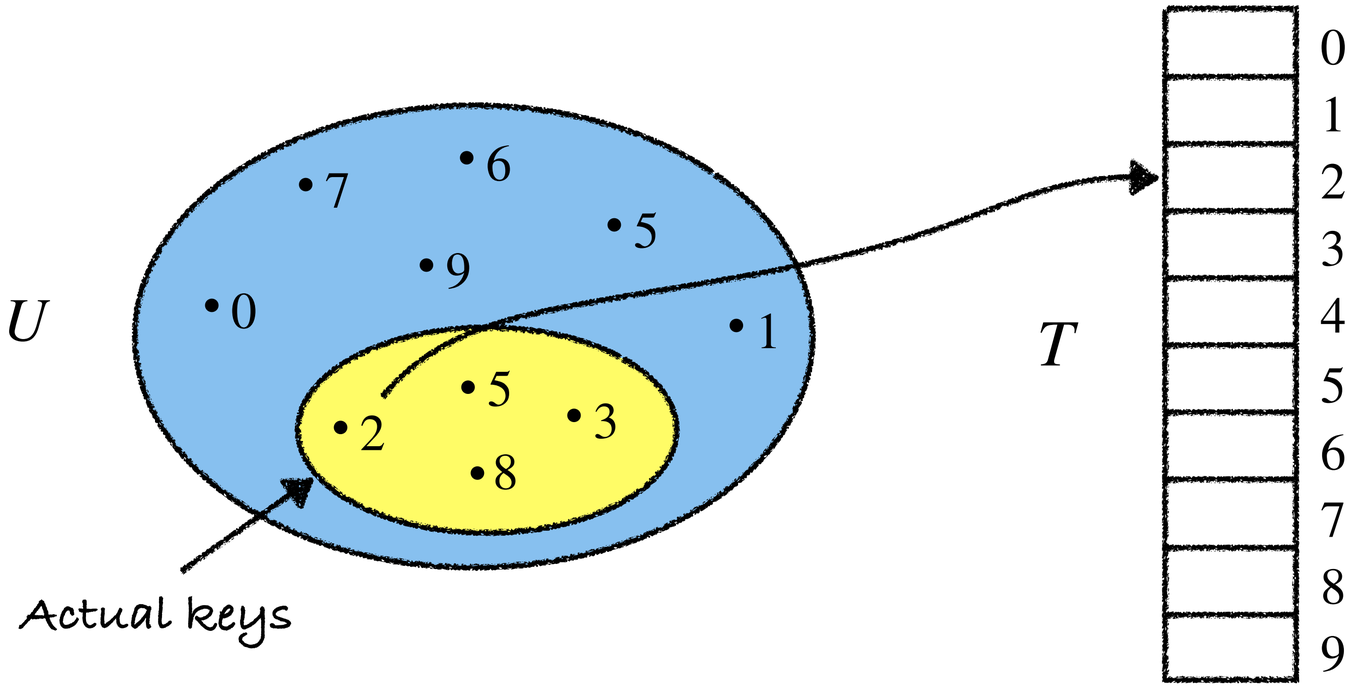


Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Then, we can use an array or a direct-address table, denoted by $T[0 : m-1]$, to represent the dynamic set, in which element x resides in $T[x\,.\,key]$.

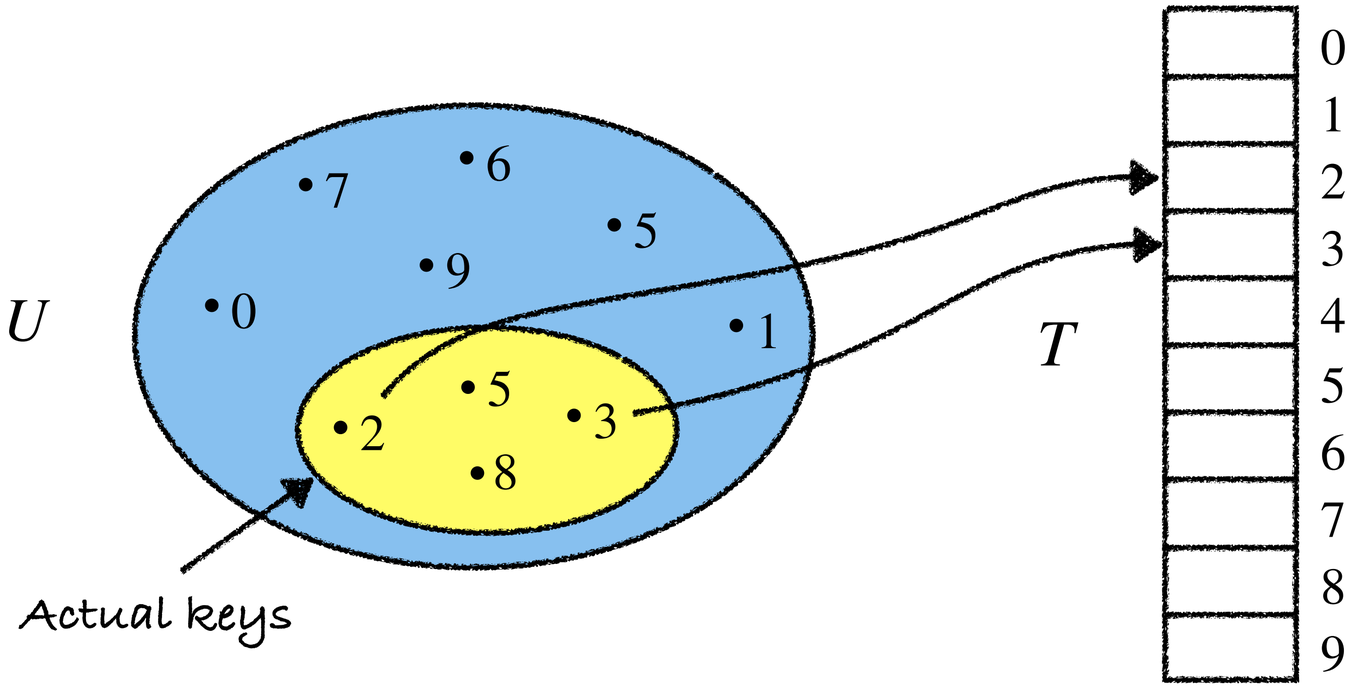


Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Then, we can use an array or a direct-address table, denoted by $T[0 : m-1]$, to represent the dynamic set, in which element x resides in $T[x\,.\,key]$.

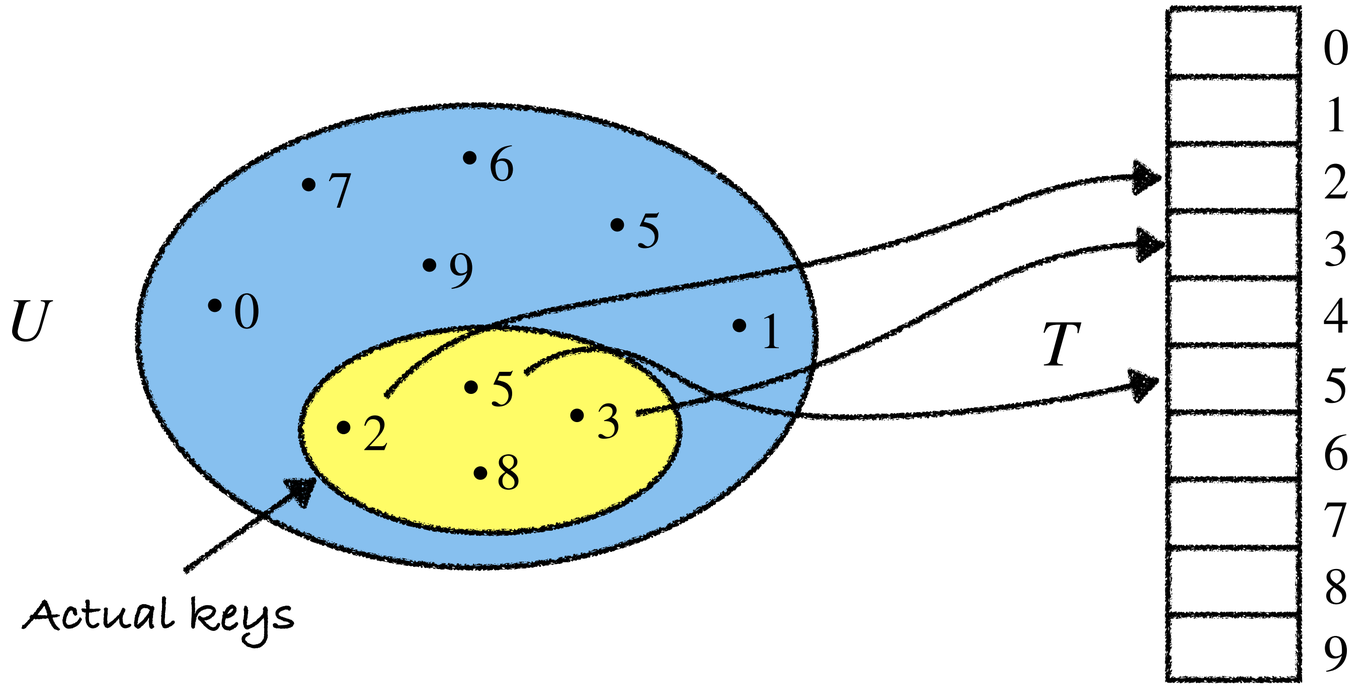


Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Then, we can use an array or a direct-address table, denoted by $T[0 : m-1]$, to represent the dynamic set, in which element x resides in $T[x\,.\,key]$.

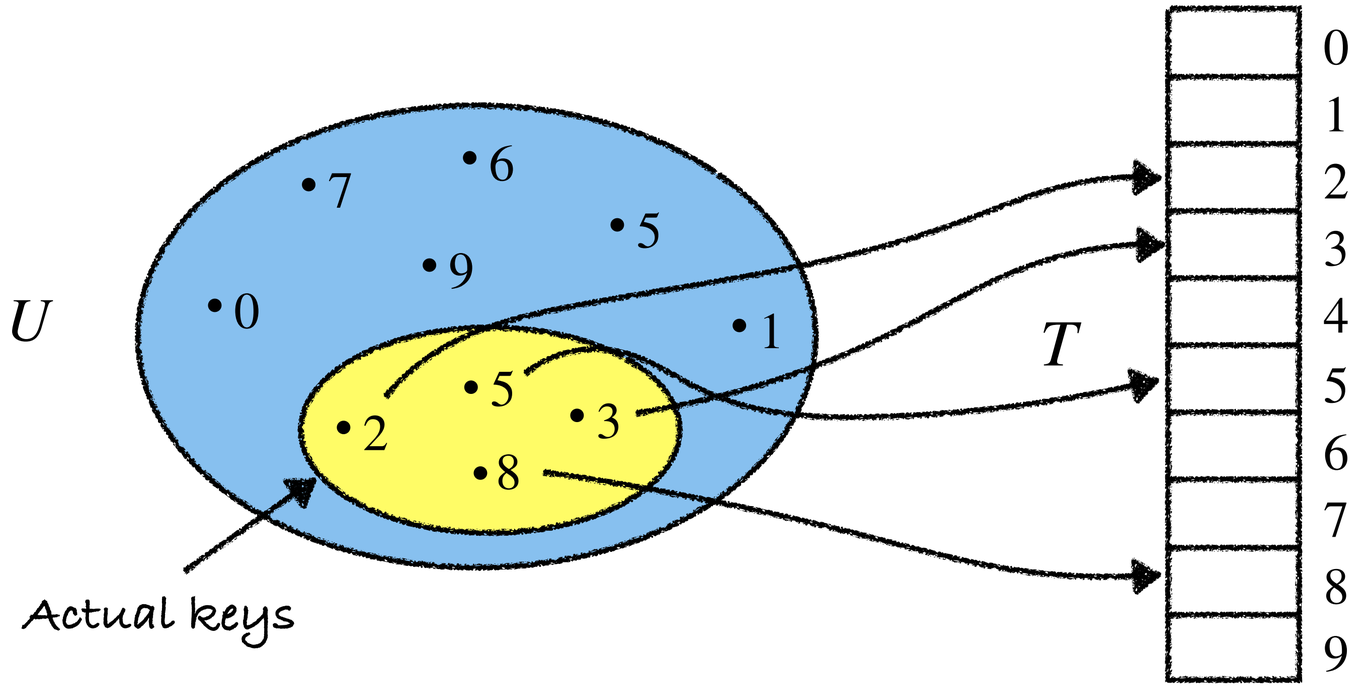


Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Then, we can use an array or a direct-address table, denoted by $T[0 : m-1]$, to represent the dynamic set, in which element x resides in $T[x\,.\,key]$.

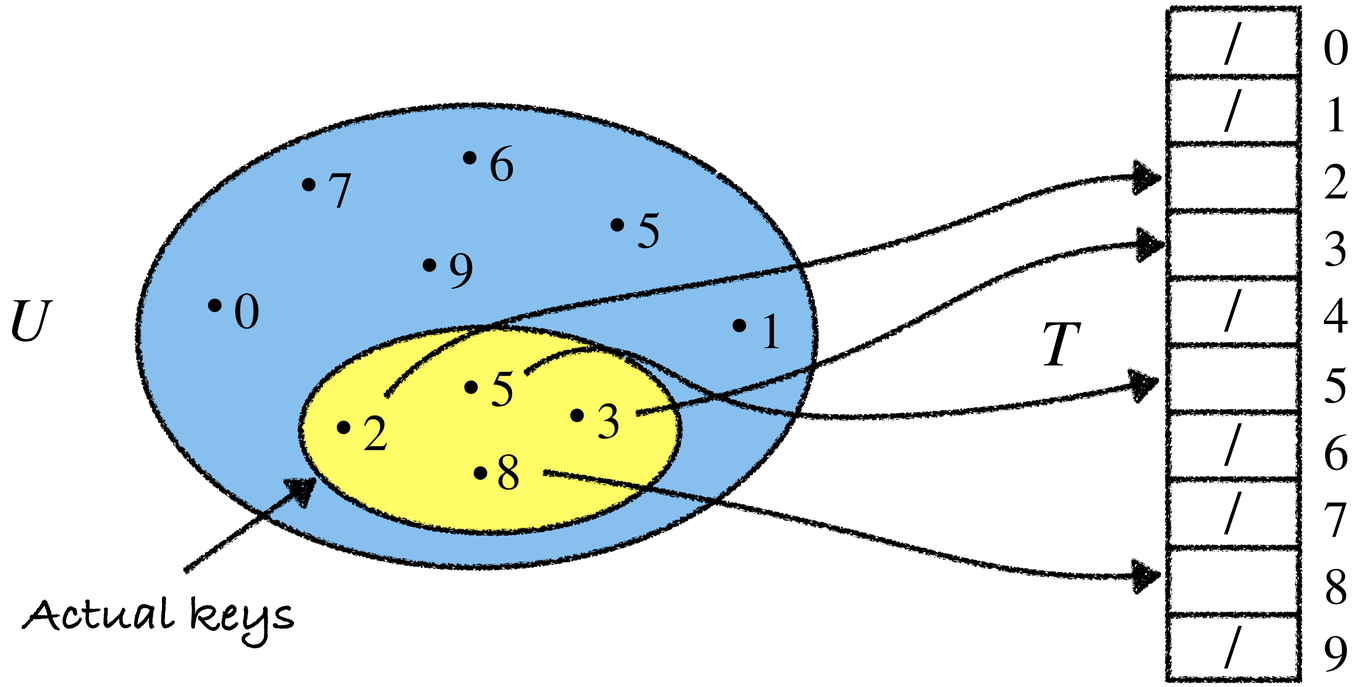


Direct-address Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,m-1\}$$

Then, we can use an array or a direct-address table, denoted by $T[0 : m-1]$, to represent the dynamic set, in which element x resides in $T[x\,.\,key]$.



Direct-address Tables: Operations

Direct-address Tables: Operations

DAT-INSERT(T, x)**:**

Direct-address Tables: Operations

DAT-INSERT(T, x)**:**

1. $T[x\,.\,key] = x$

Direct-address Tables: Operations

DAT-INSERT(T, x)**:**

1. $T[x\,.\,key] = x$

DAT-SEARCH(T, k)**:**

Direct-address Tables: Operations

DAT-INSERT(T, x)**:**

1. $T[x\,.\,key] = x$

DAT-SEARCH(T, k)**:**

1. **return** $T[k]$

Direct-address Tables: Operations

DAT-INSERT(T, x)**:**

1. $T[x\,.\,key] = x$

DAT-SEARCH(T, k)**:**

1. **return** $T[k]$

DAT-DELETE(T, x)**:**

Direct-address Tables: Operations

DAT-INSERT(T, x)**:**

1. $T[x\,.\,key] = x$

DAT-SEARCH(T, k)**:**

1. **return** $T[k]$

DAT-DELETE(T, x)**:**

1. $T[x\,.\,key] = NIL$

Direct-address Tables: Operations

DAT-INSERT(T, x)**:**

1. $T[x\,.\,key] = x$

Time complexity: $O(1)$

DAT-SEARCH(T, k)**:**

1. **return** $T[k]$

Time complexity: $O(1)$

DAT-DELETE(T, x)**:**

1. $T[x\,.\,key] = NIL$

Time complexity: $O(1)$

Direct-address Tables: Cons

Direct-address Tables: Cons

A downside of direct-address tables:

Direct-address Tables: Cons

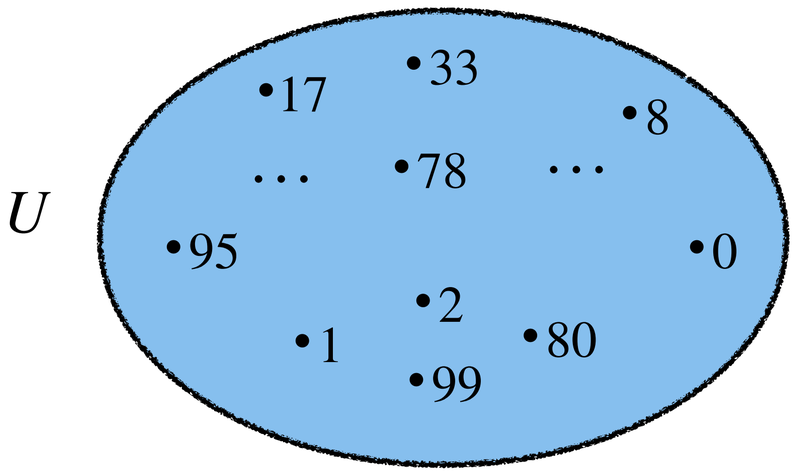
A downside of direct-address tables:

Large universe of keys but small number of elements causes wastage of space.

Direct-address Tables: Cons

A downside of direct-address tables:

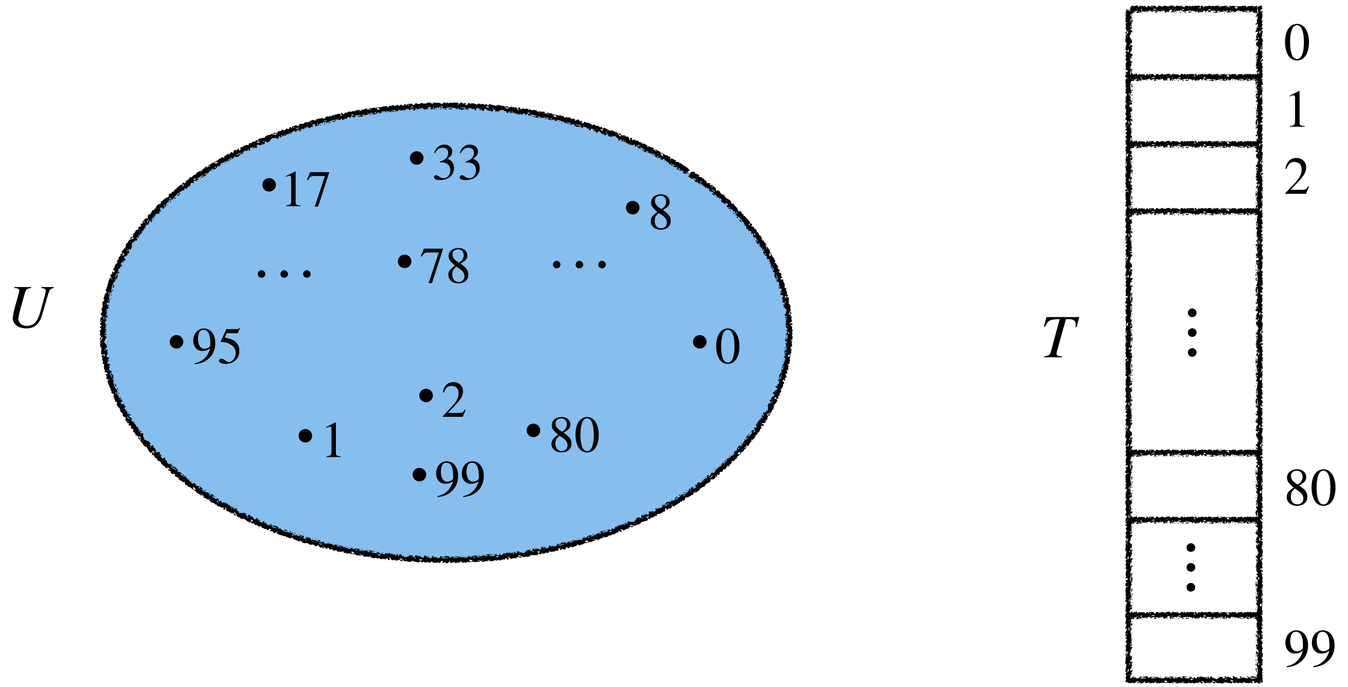
Large universe of keys but small number of elements causes wastage of space.



Direct-address Tables: Cons

A downside of direct-address tables:

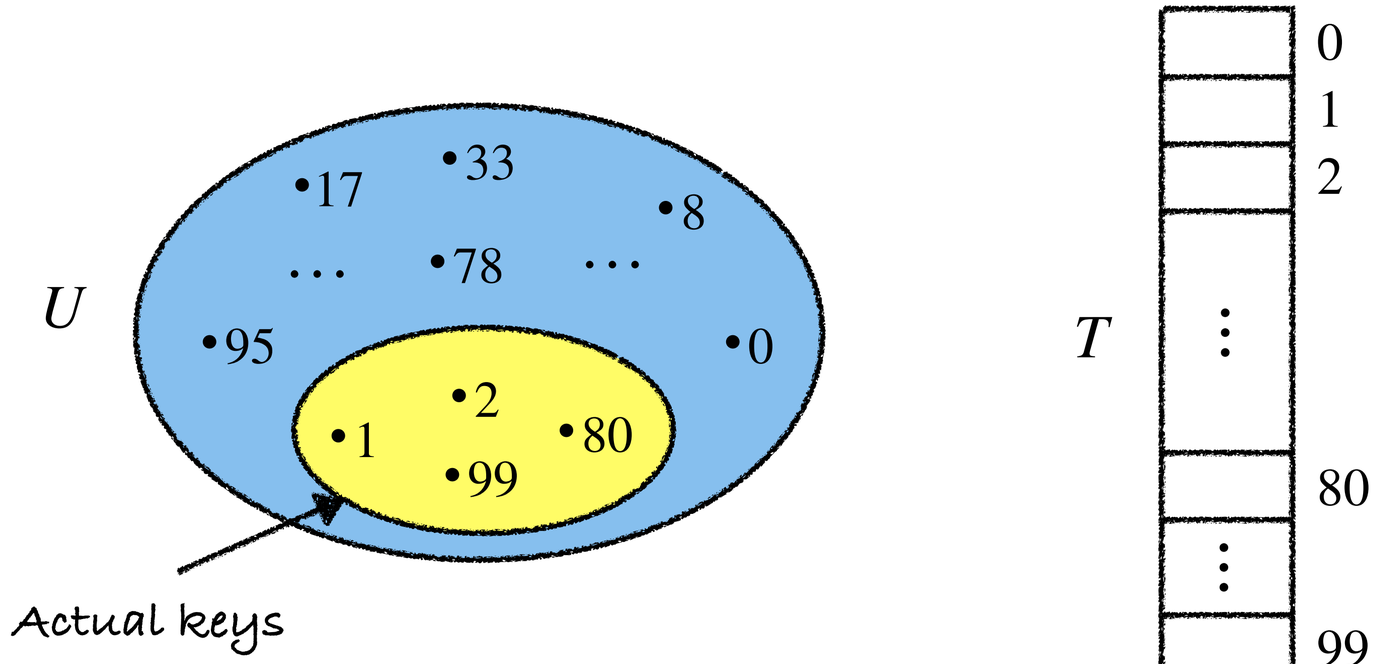
Large universe of keys but small number of elements causes wastage of space.



Direct-address Tables: Cons

A downside of direct-address tables:

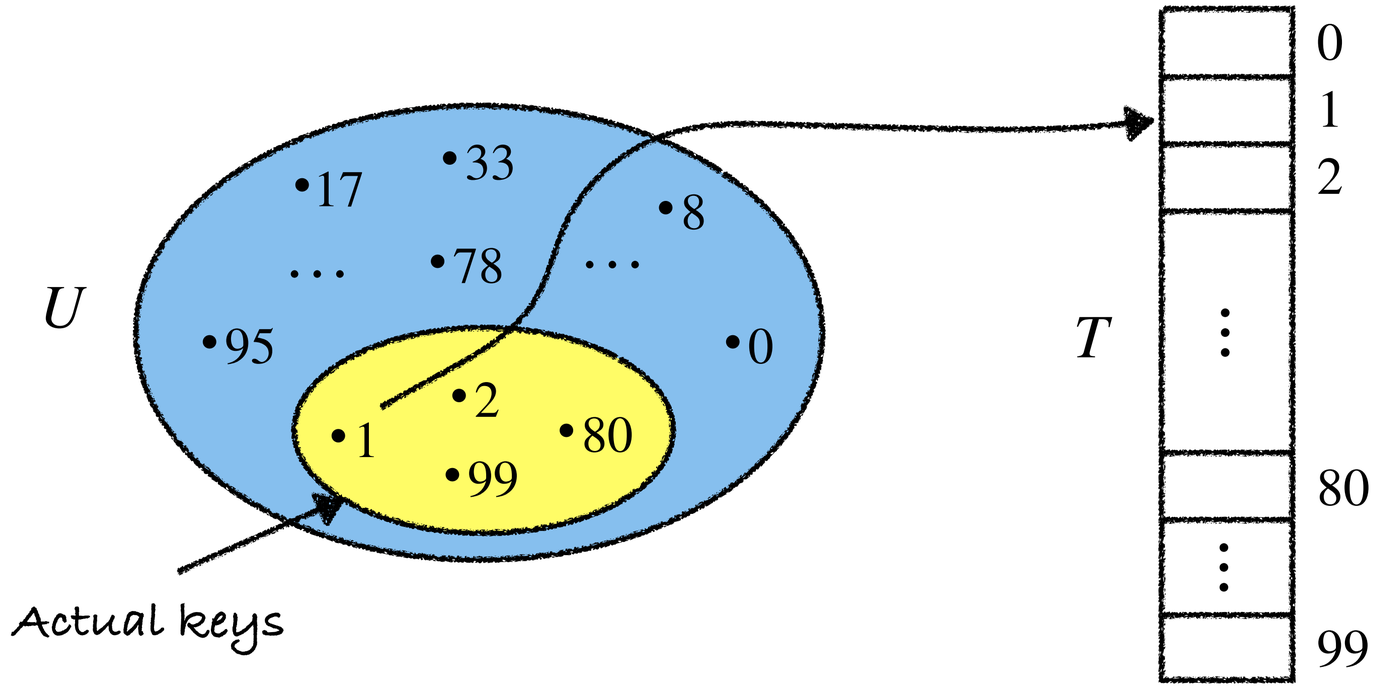
Large universe of keys but small number of elements causes wastage of space.



Direct-address Tables: Cons

A downside of direct-address tables:

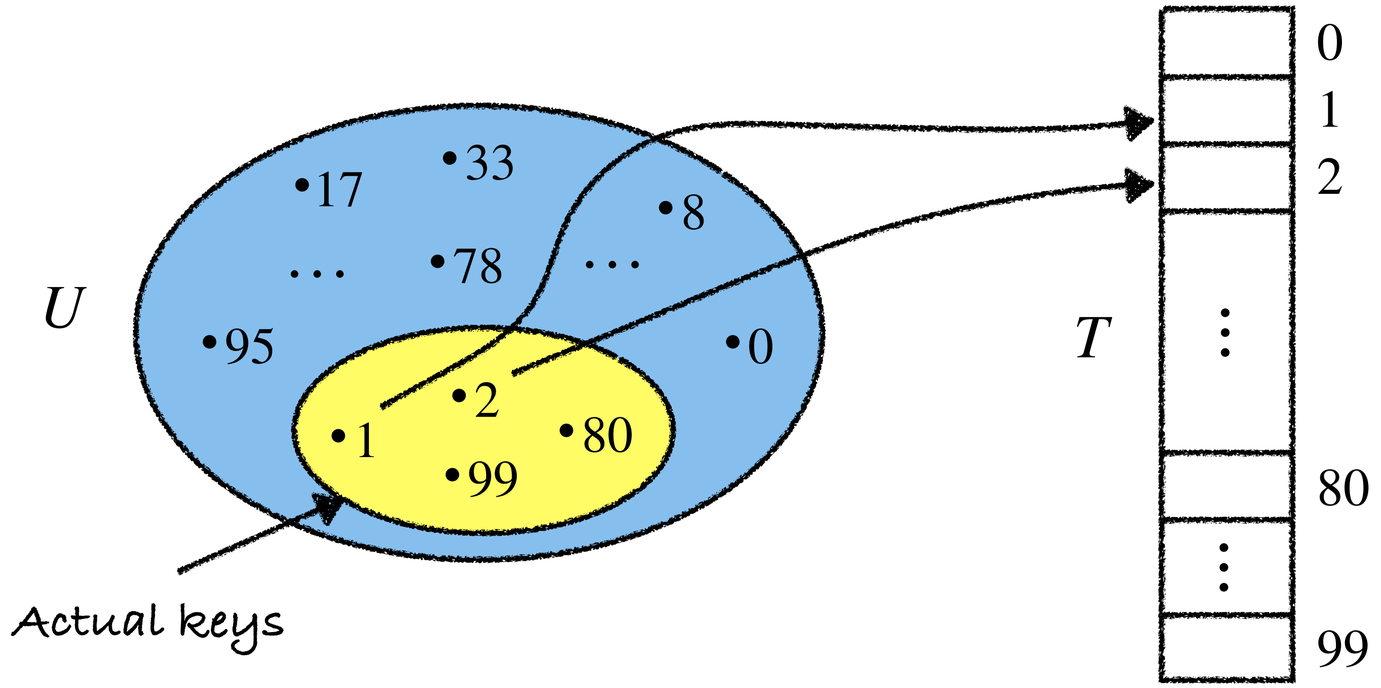
Large universe of keys but small number of elements causes wastage of space.



Direct-address Tables: Cons

A downside of direct-address tables:

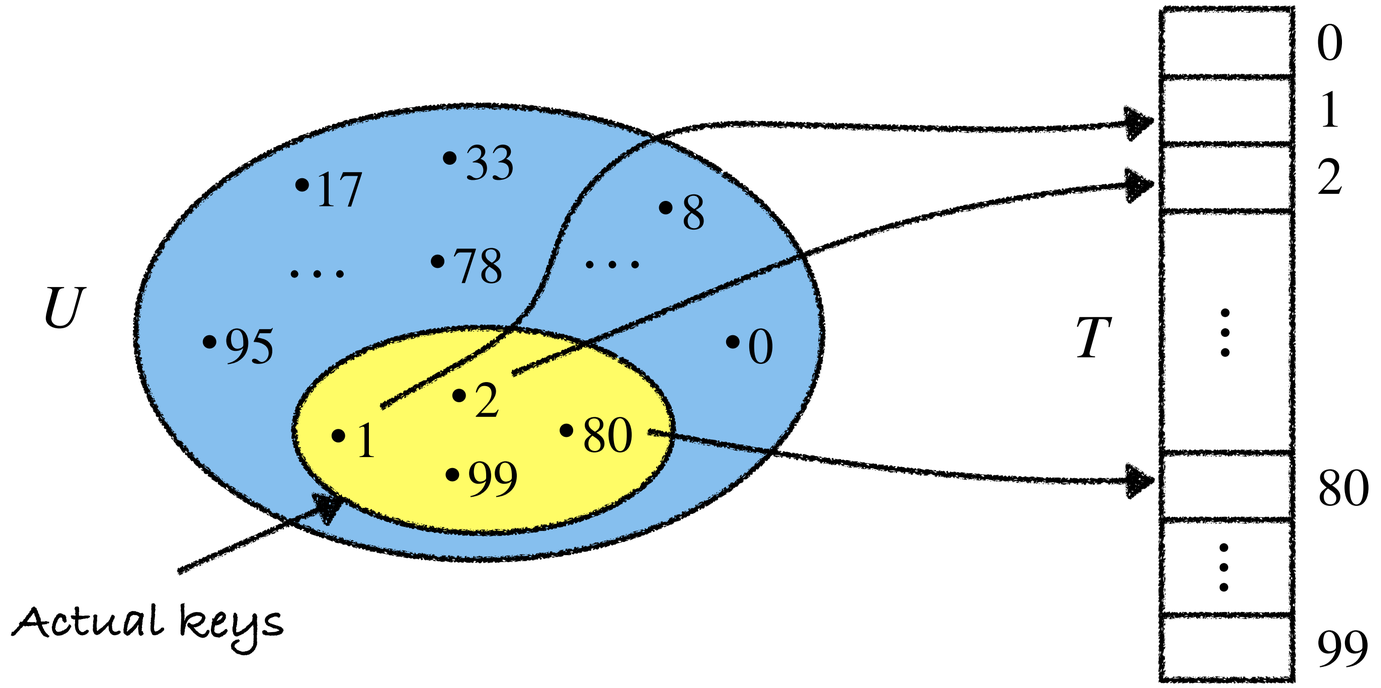
Large universe of keys but small number of elements causes wastage of space.



Direct-address Tables: Cons

A downside of direct-address tables:

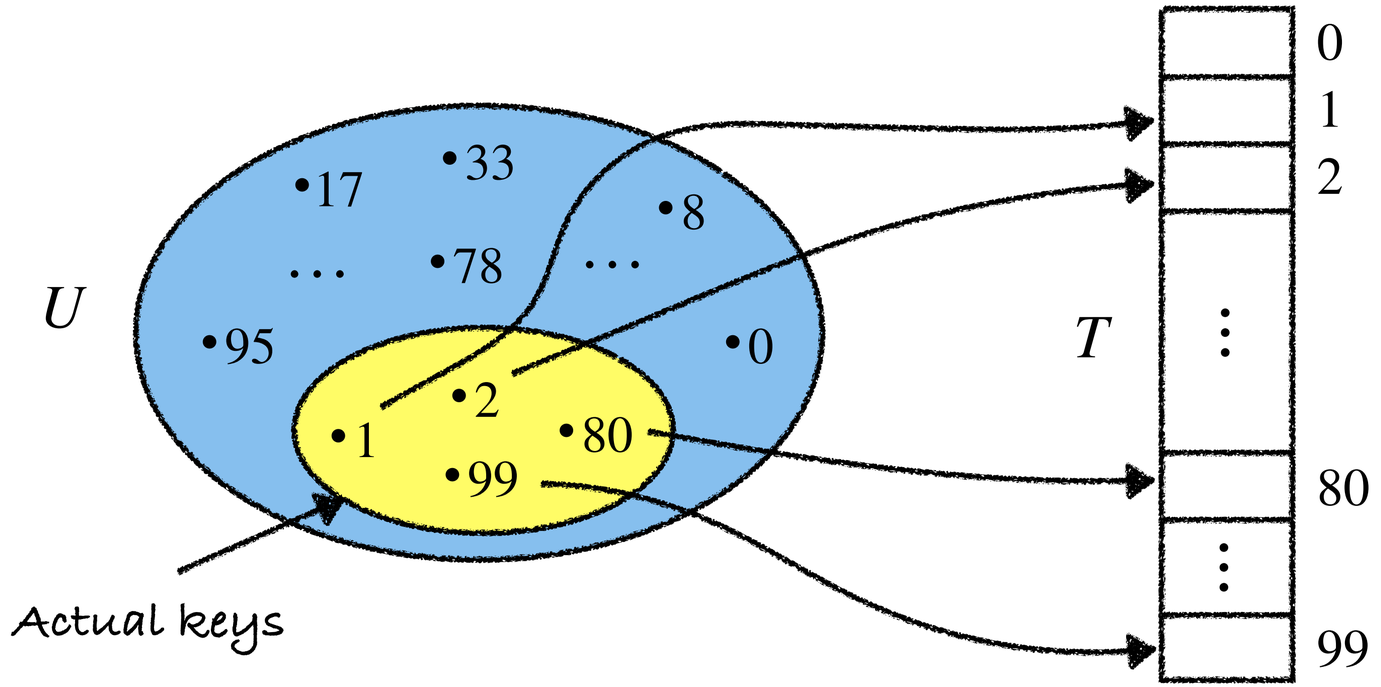
Large universe of keys but small number of elements causes wastage of space.



Direct-address Tables: Cons

A downside of direct-address tables:

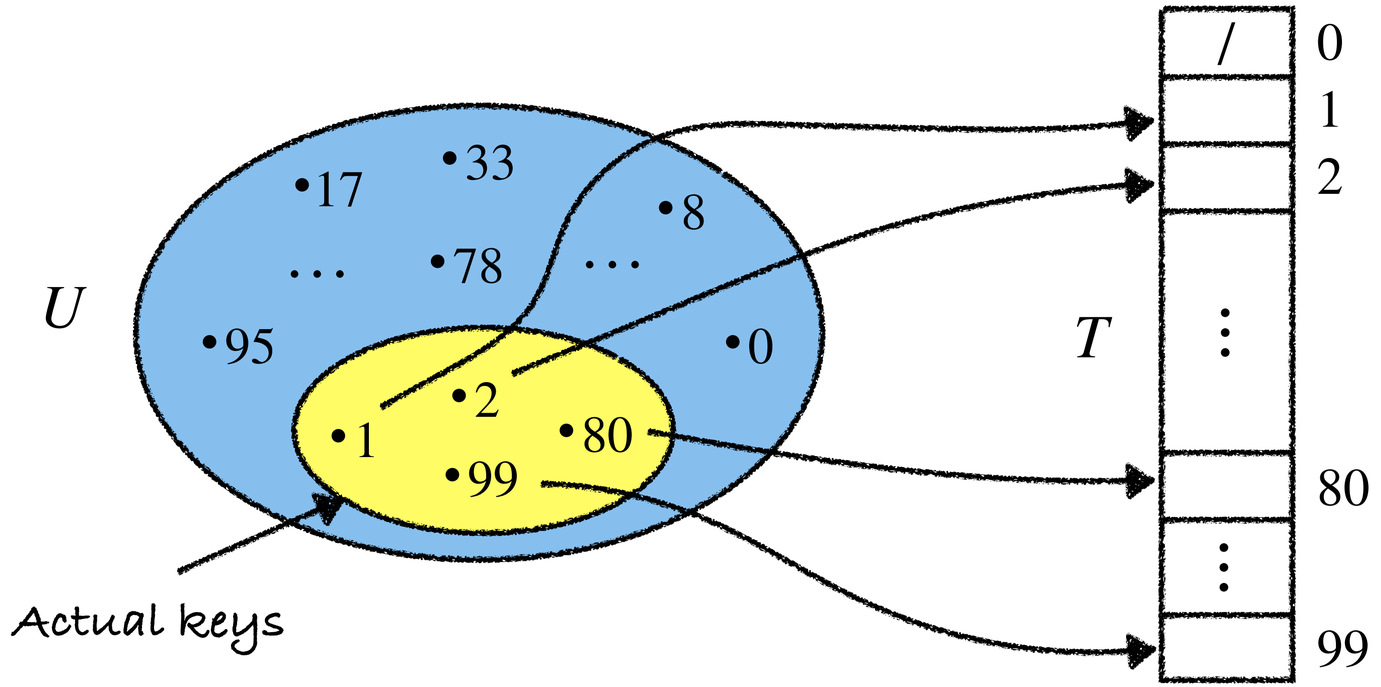
Large universe of keys but small number of elements causes wastage of space.



Direct-address Tables: Cons

A downside of direct-address tables:

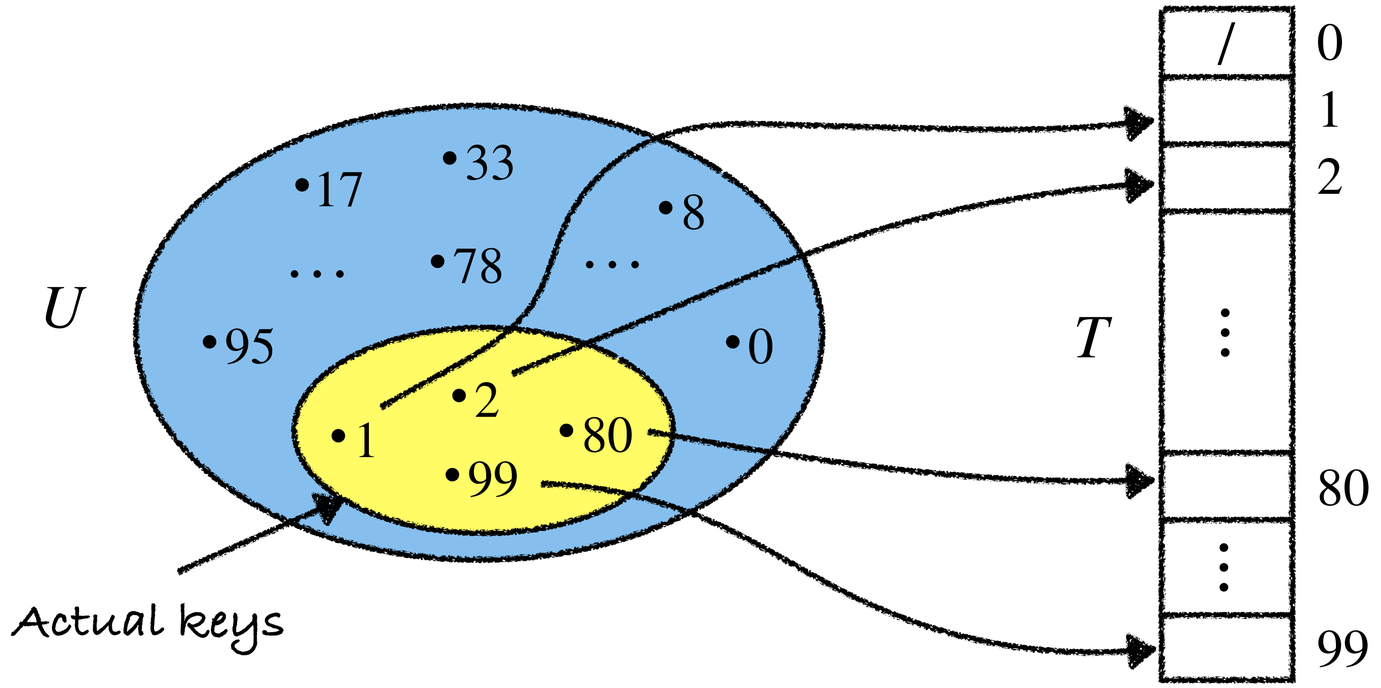
Large universe of keys but small number of elements causes wastage of space.



Direct-address Tables: Cons

A downside of direct-address tables:

Large universe of keys but small number of elements causes wastage of space.



96 slots are wasted

Hash Tables

Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$,

Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x\,.\,key)]$,

Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0:m-1]$, where $m < n$, element x resides in $T[h(x.key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x\,.\,key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.

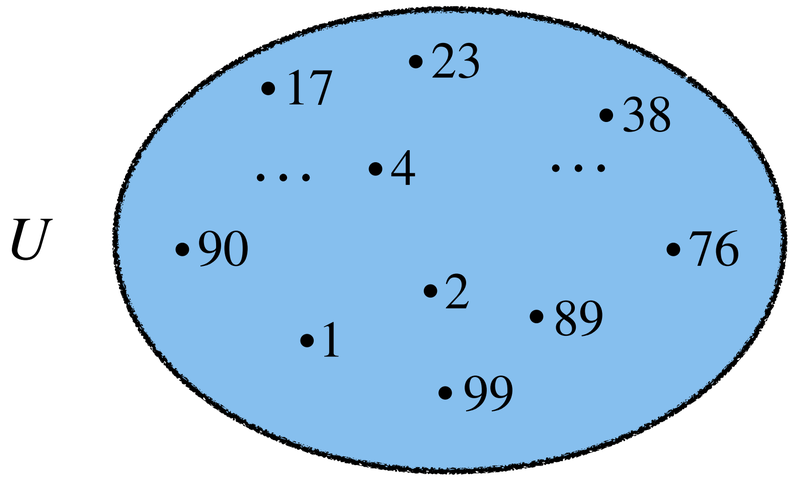
Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x\,.\,key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



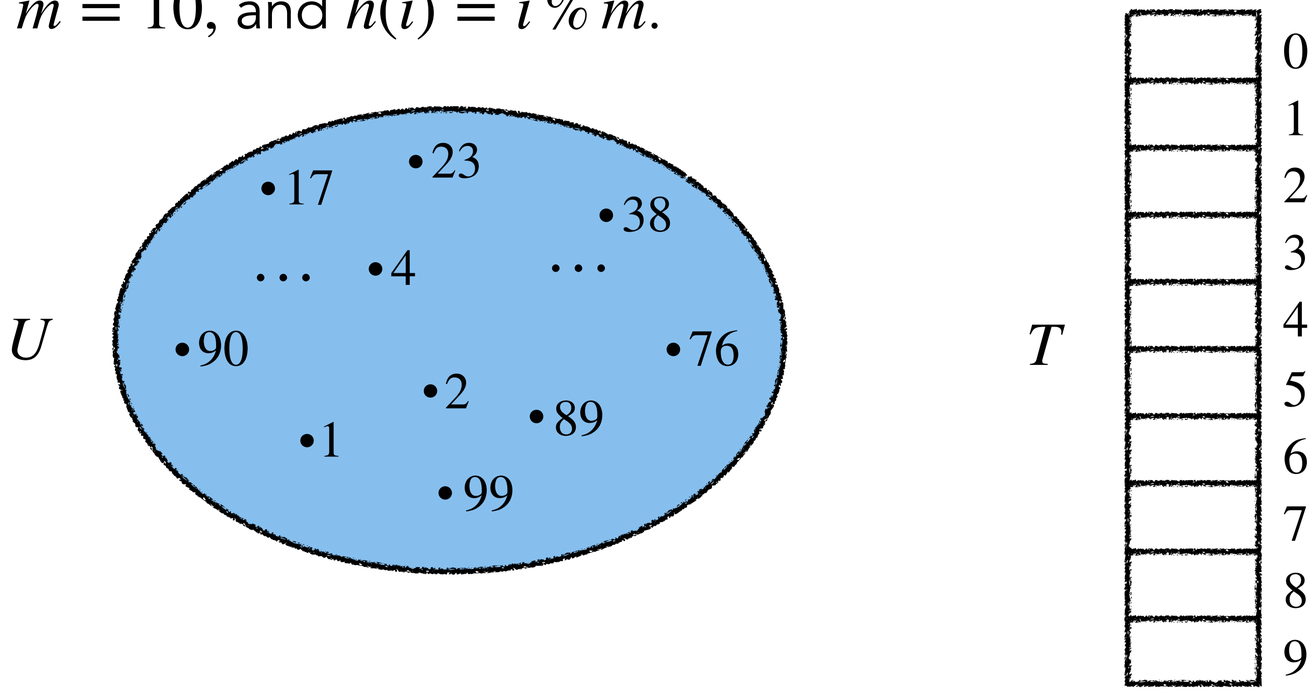
Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x.key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



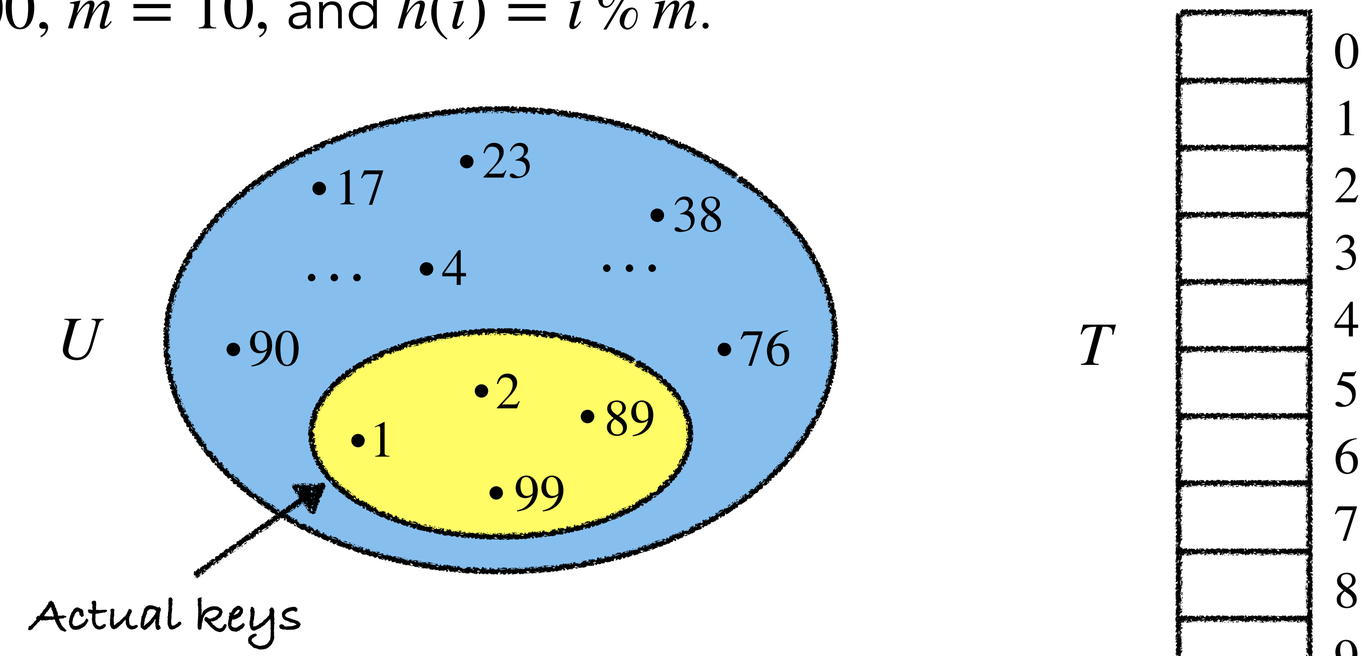
Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x\,.\,key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



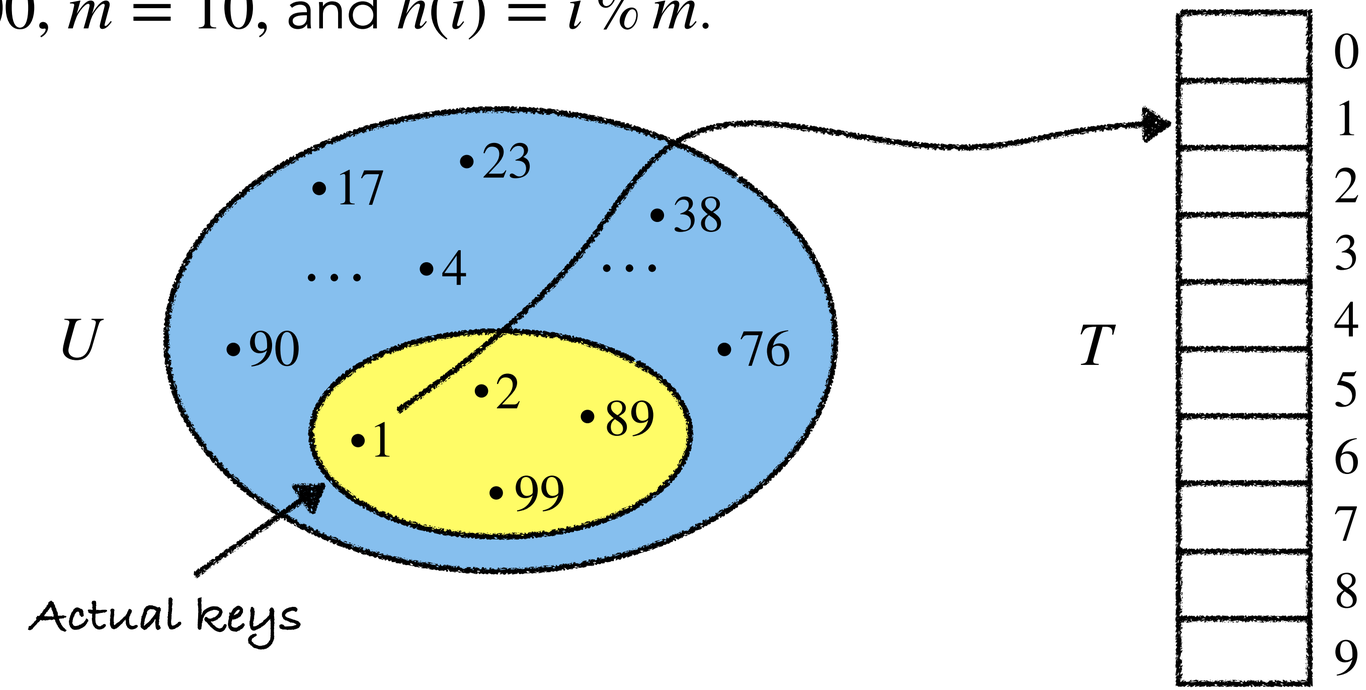
Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x\,.\,key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



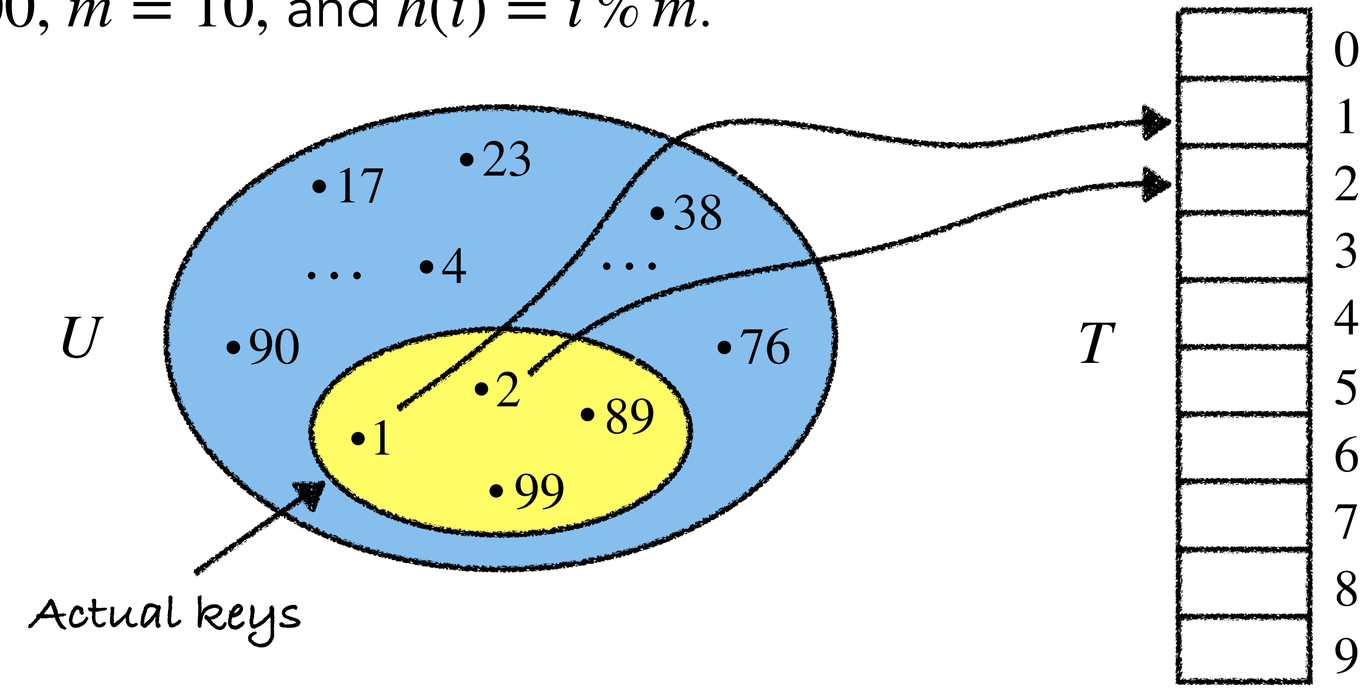
Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x.key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



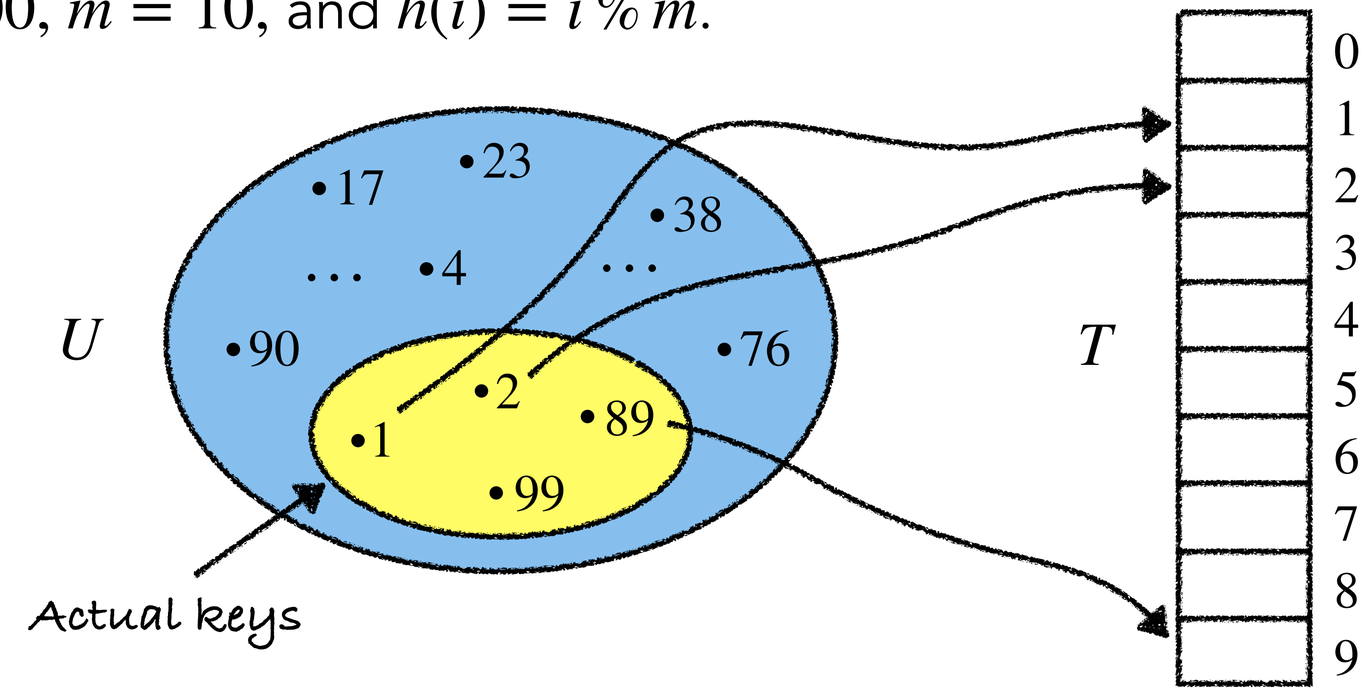
Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x\,.\,key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



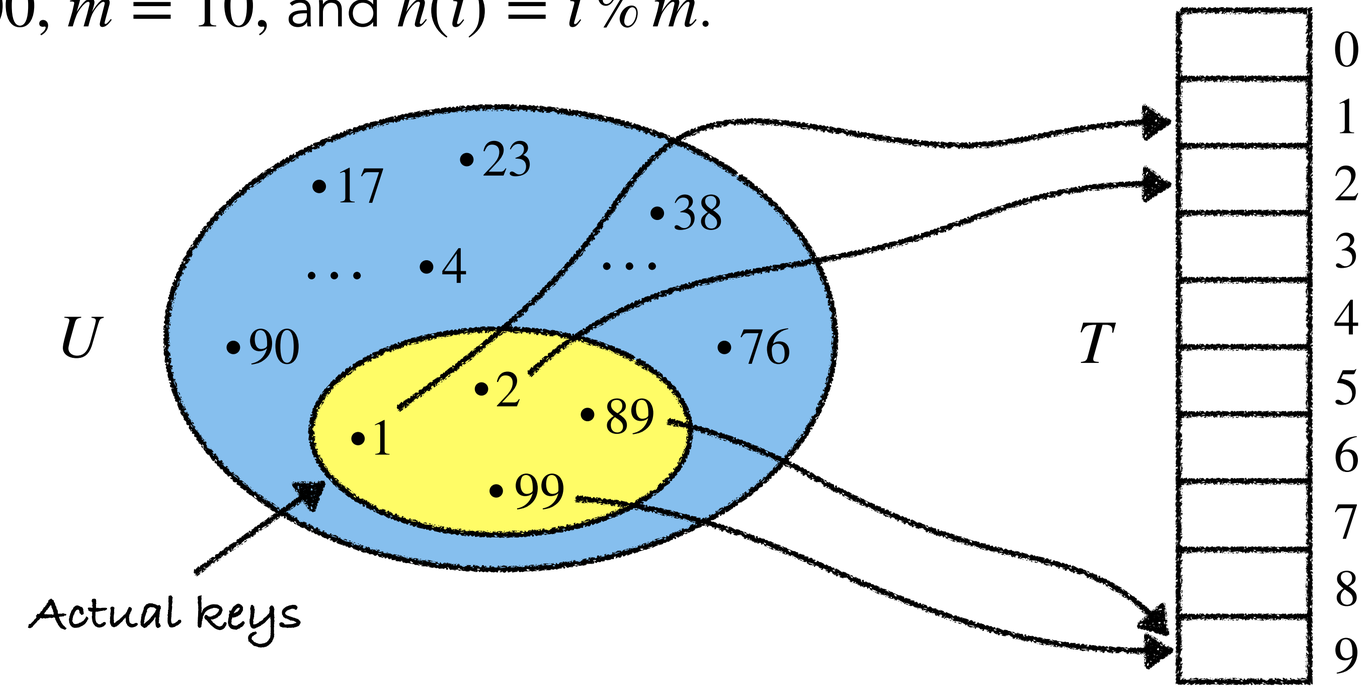
Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x.key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



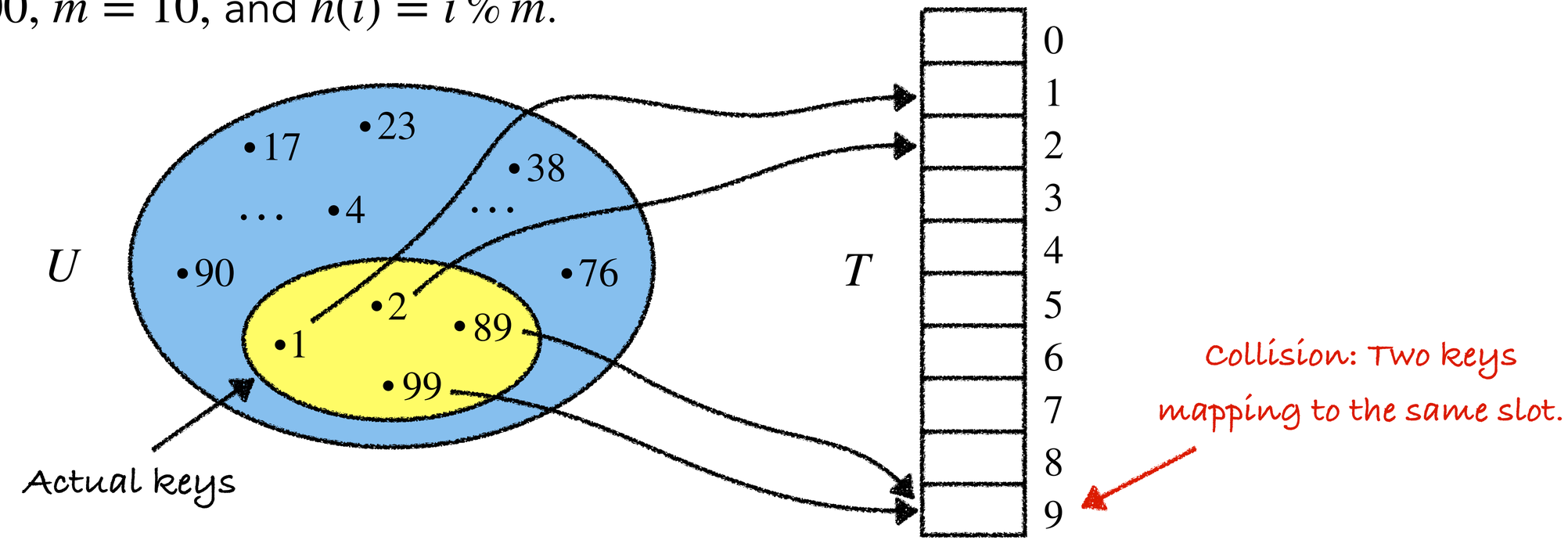
Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x.key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



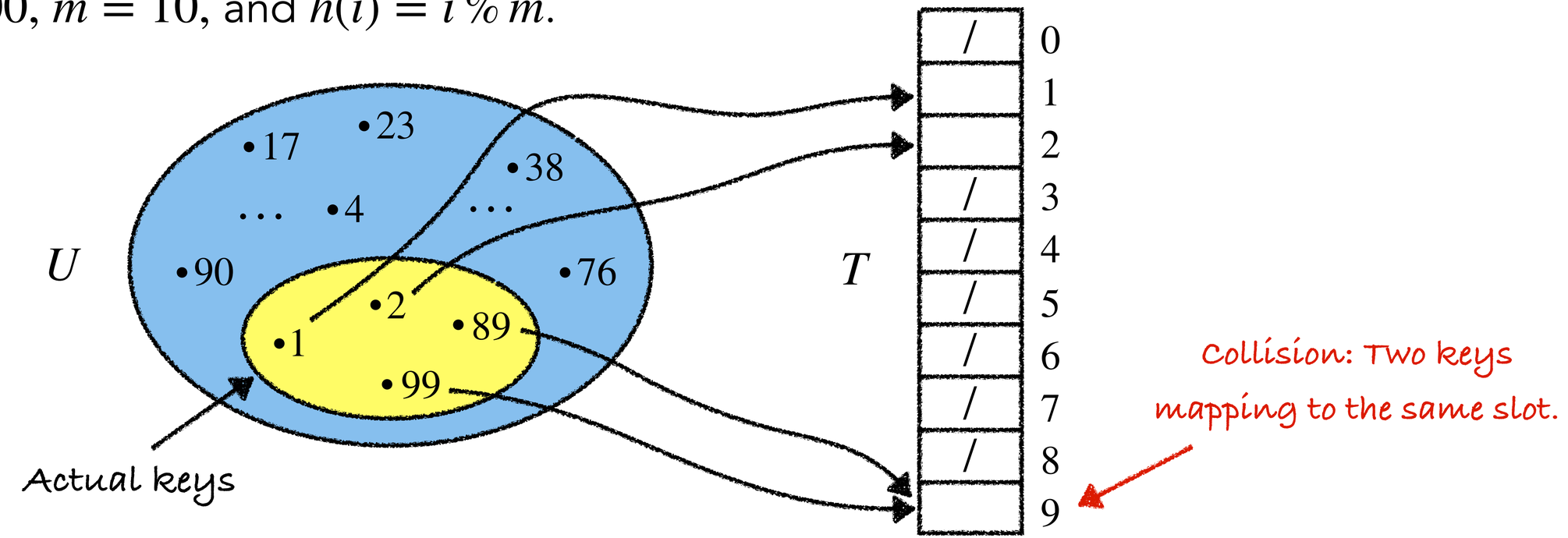
Hash Tables

Suppose every element of the dynamic set has a distinct key from the universe U.

$$U = \{0,1,\ldots,n-1\}$$

In a hash table, denoted by $T[0 : m-1]$, where $m < n$, element x resides in $T[h(x.key)]$, where $h\colon U \to \{0,1,\ldots,m-1\}$ is a hash function.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Hash Tables: Chaining

Chaining is a way to handle collisions,

Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of

Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

Hash Tables: Chaining

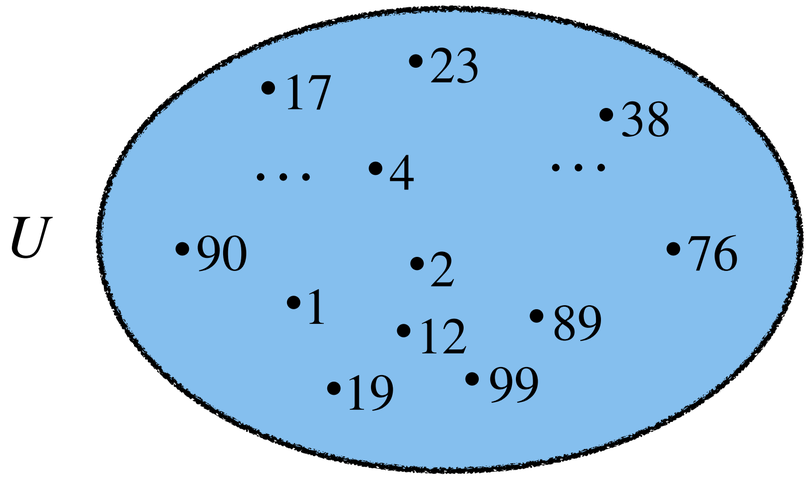
Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.

Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

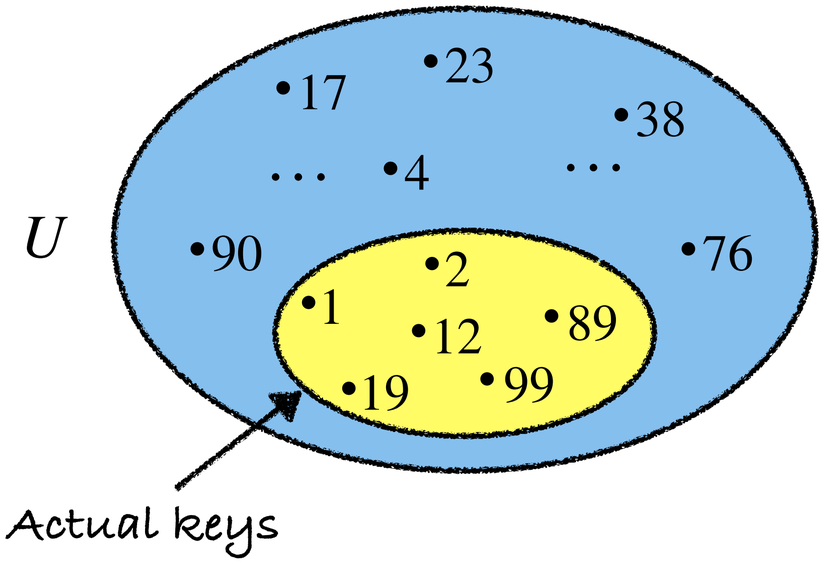
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

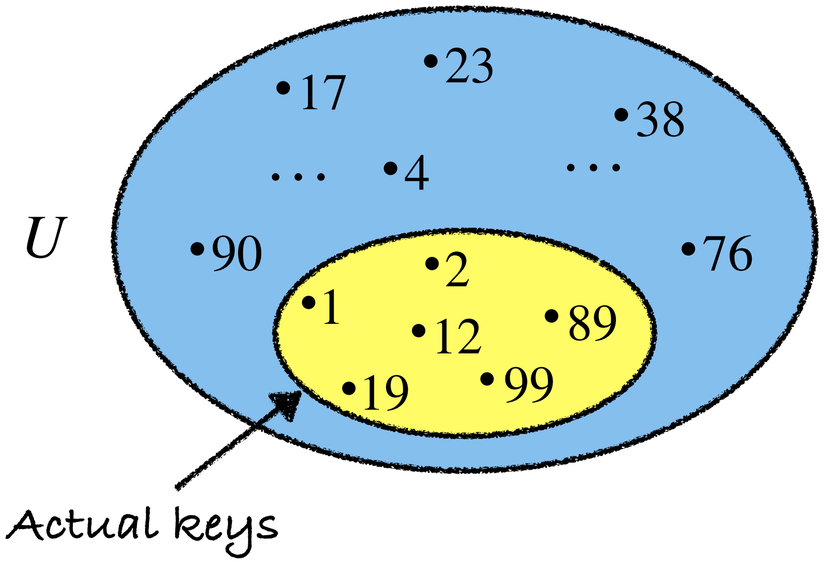
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.

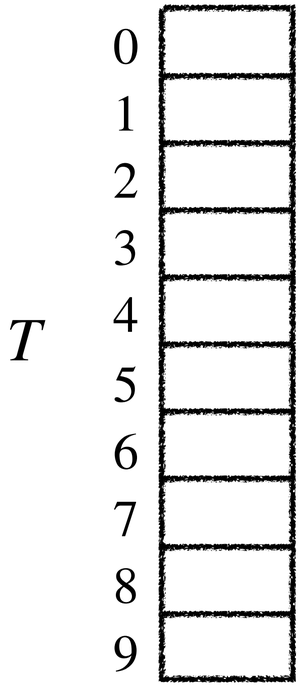


Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.

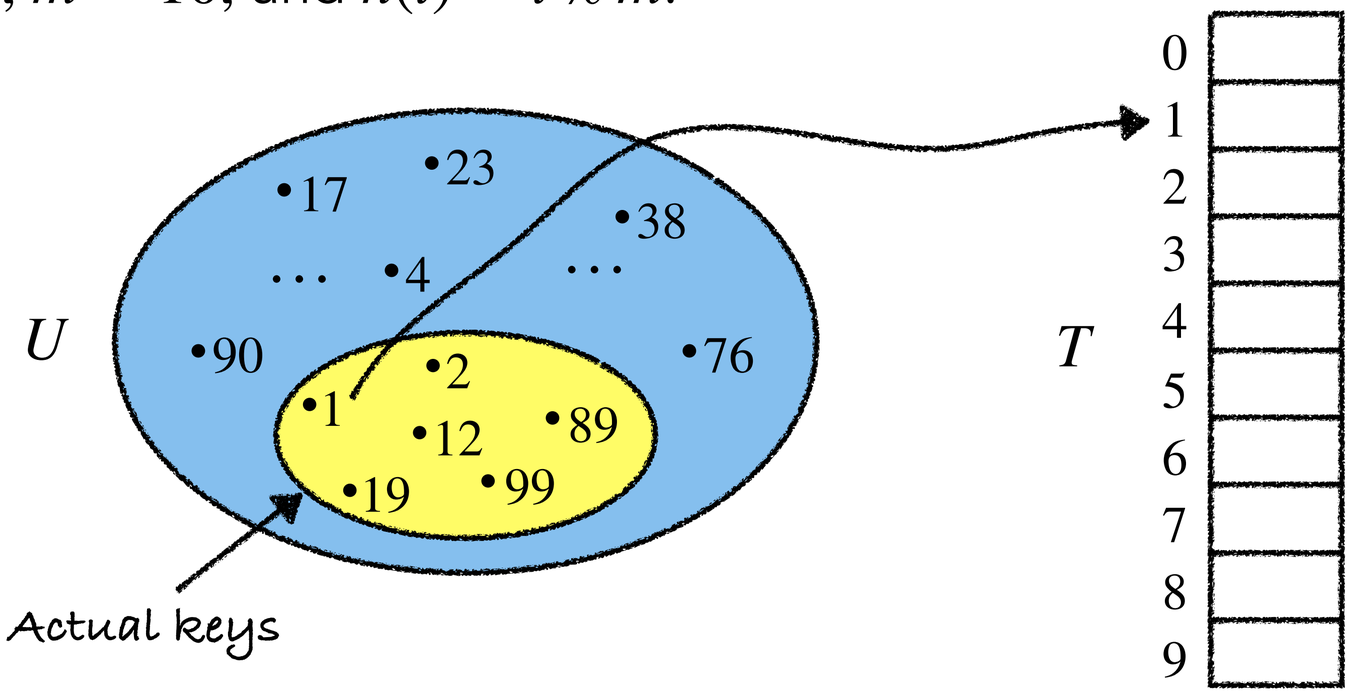




Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

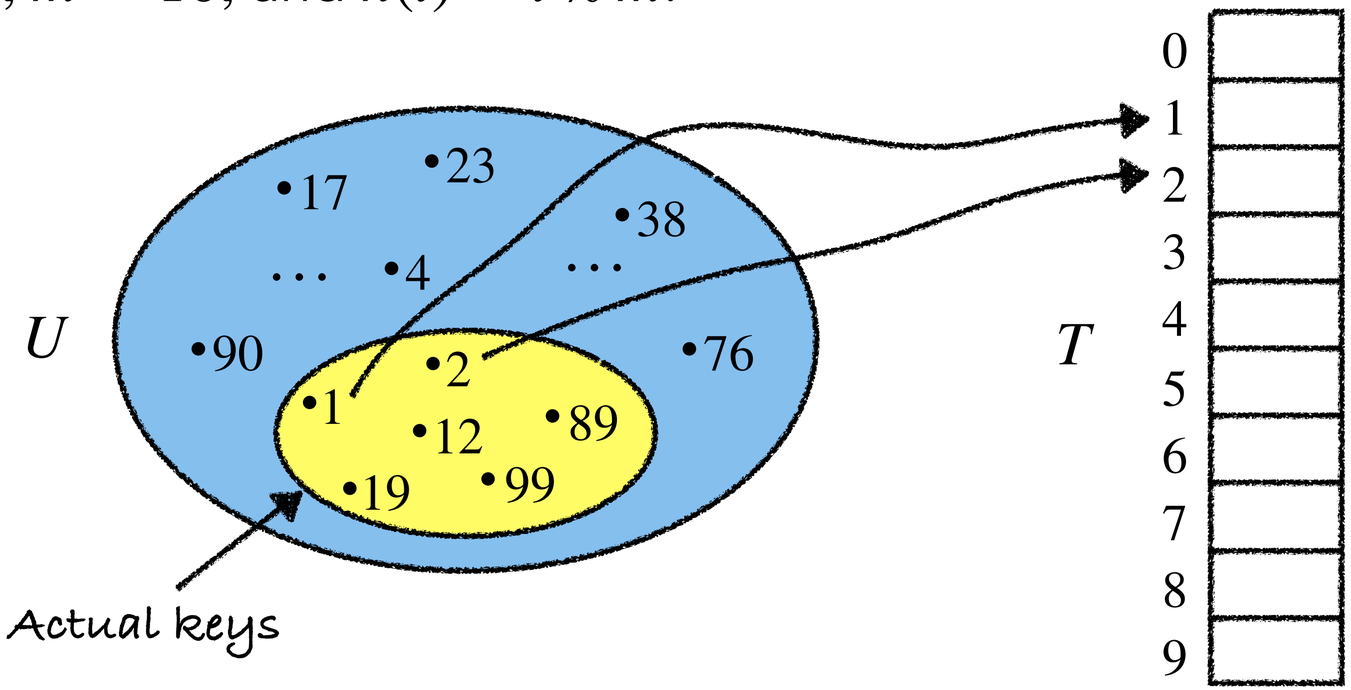
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

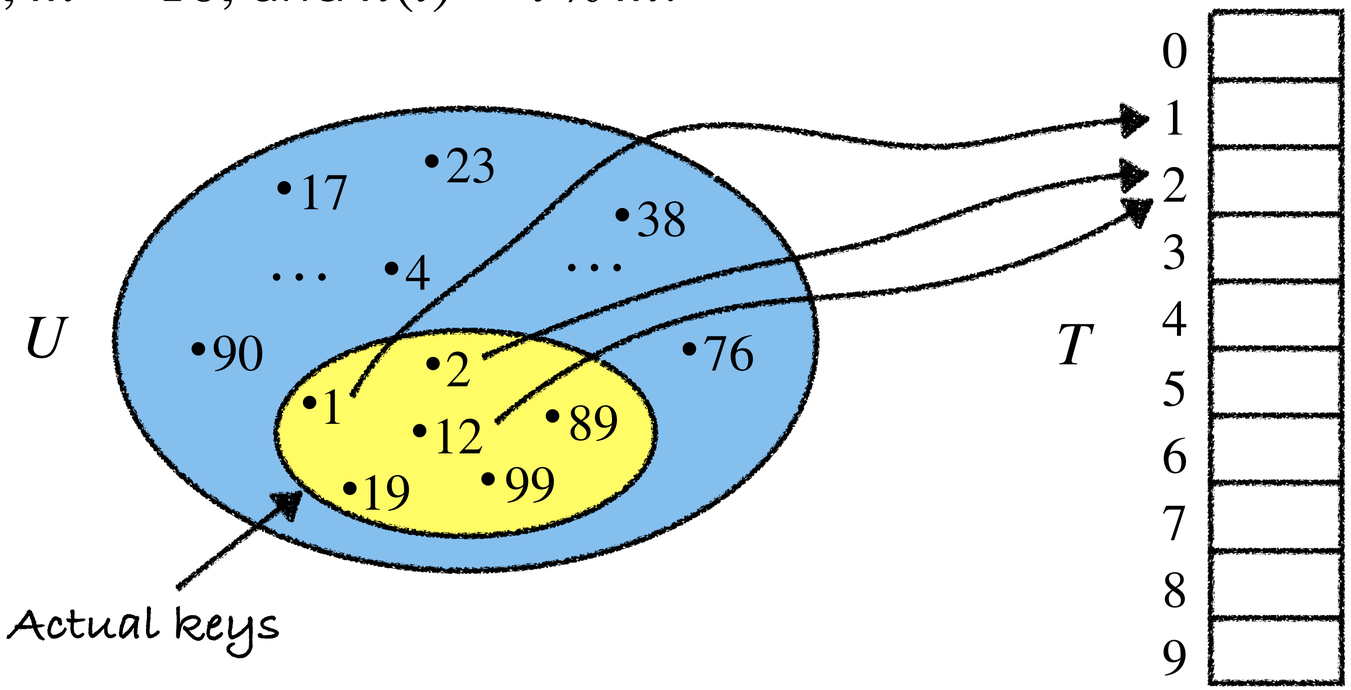
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

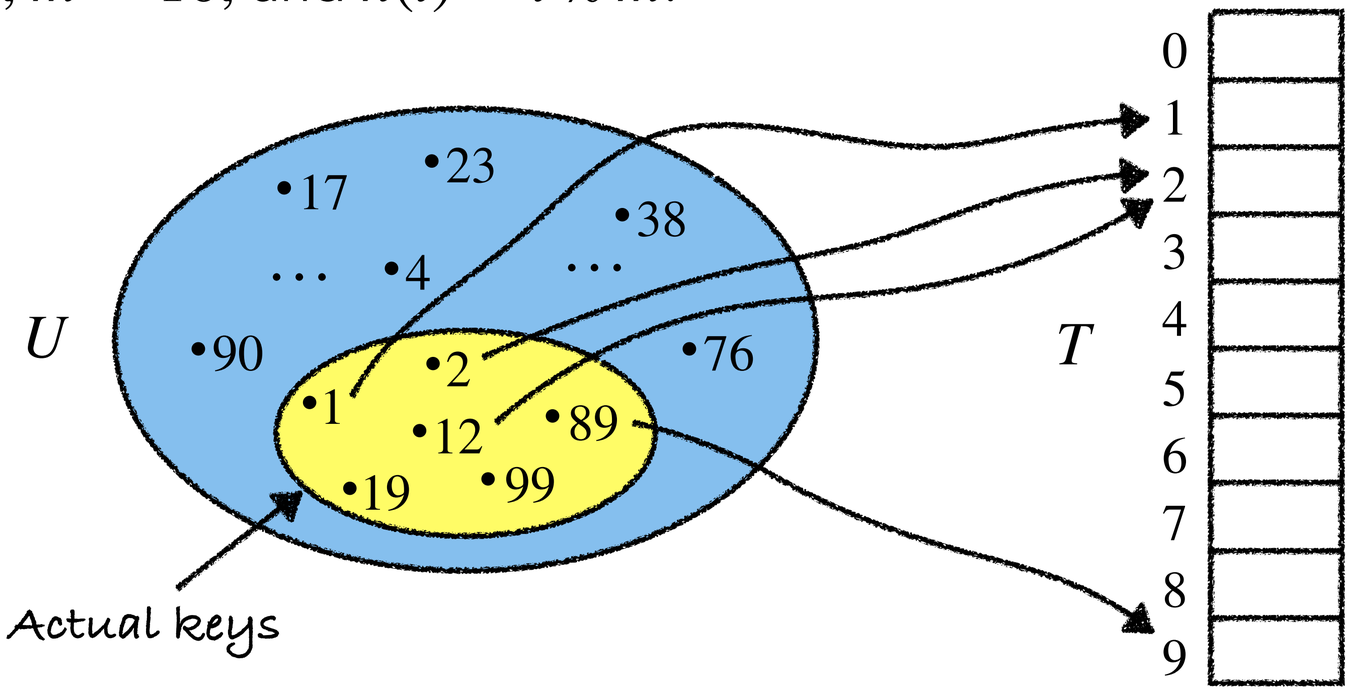
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

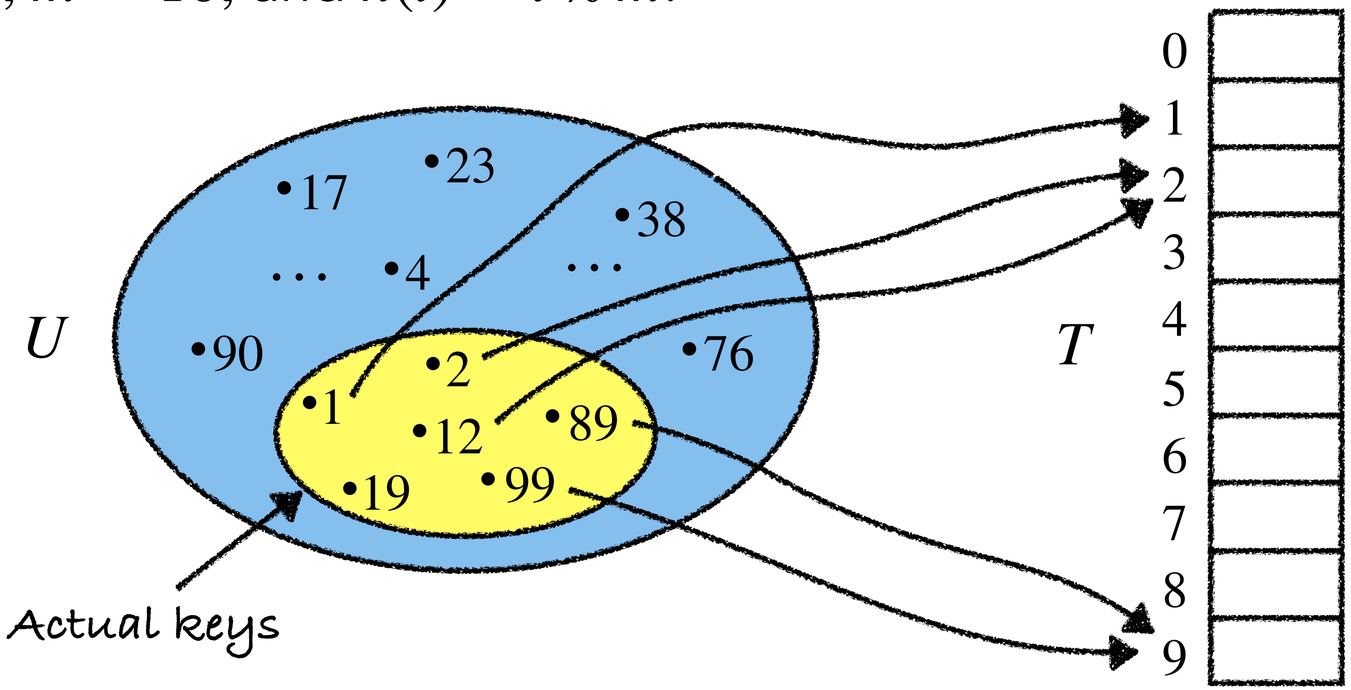
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

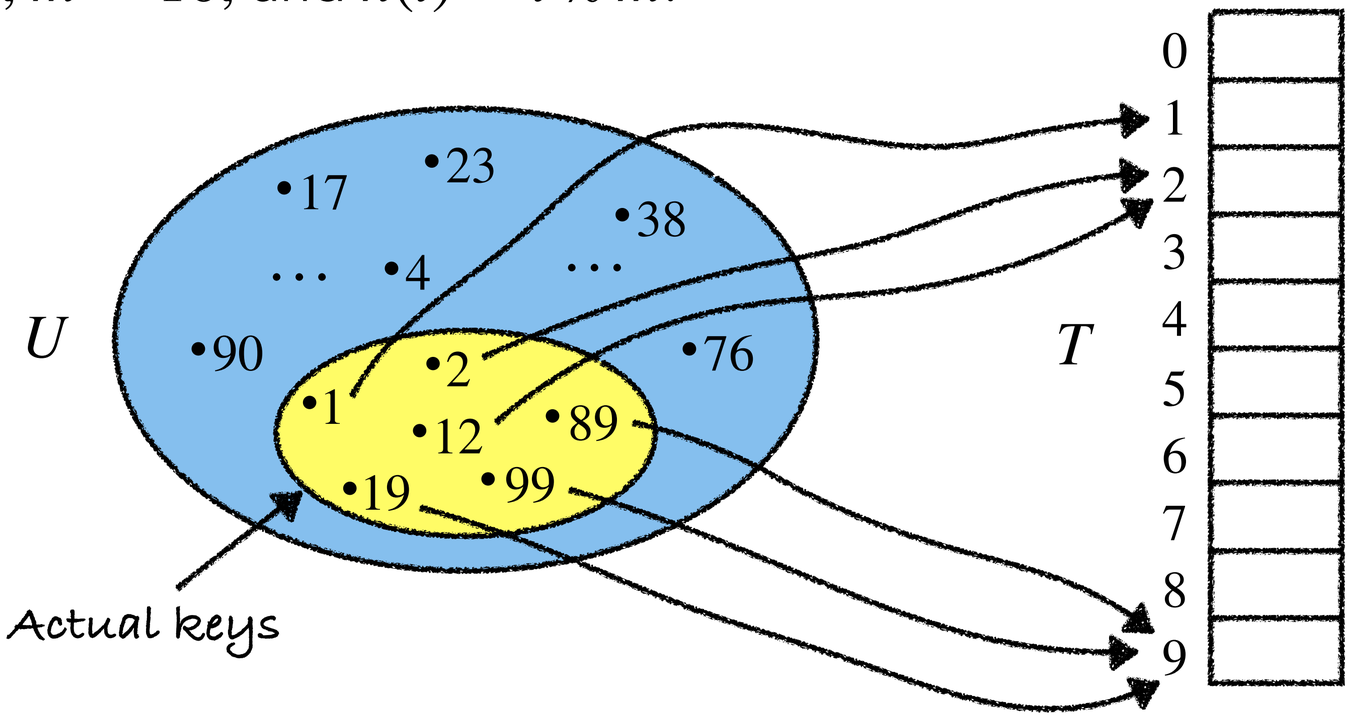
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

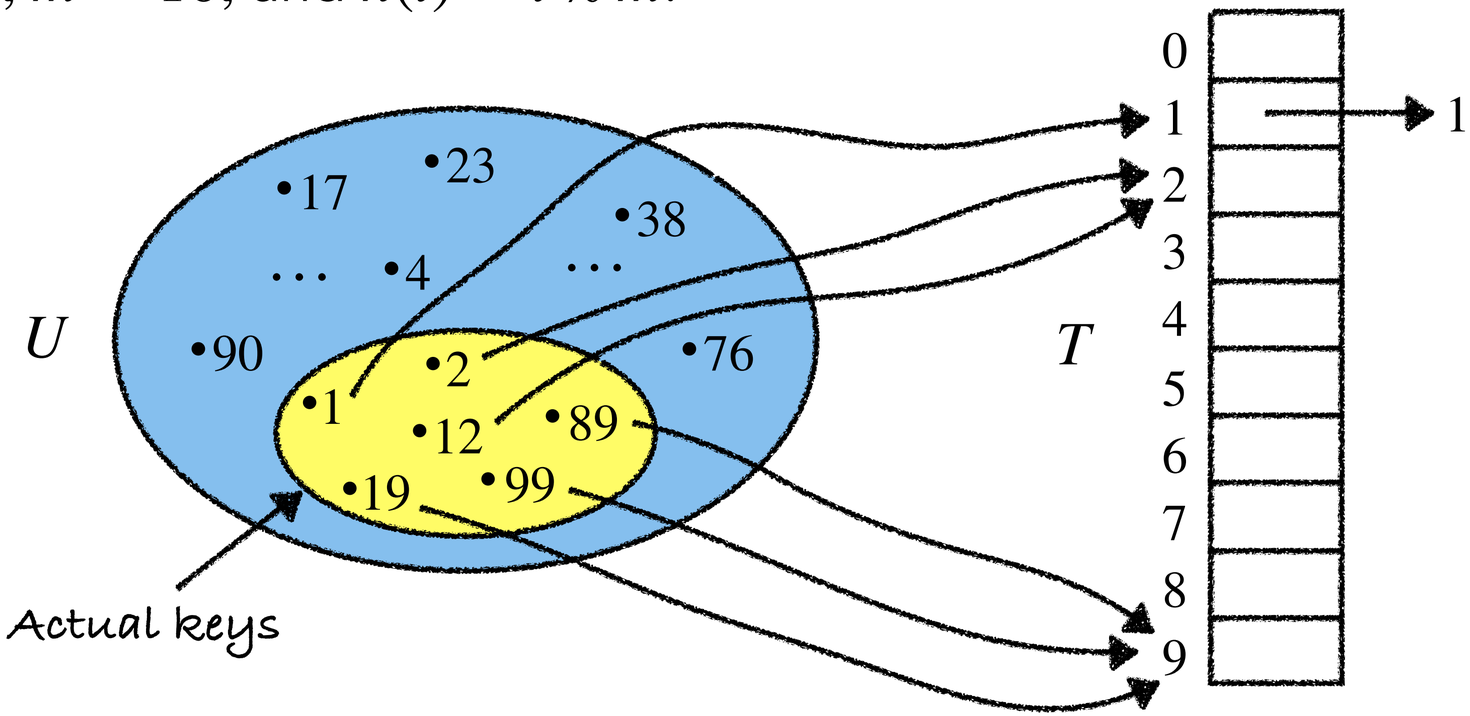
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

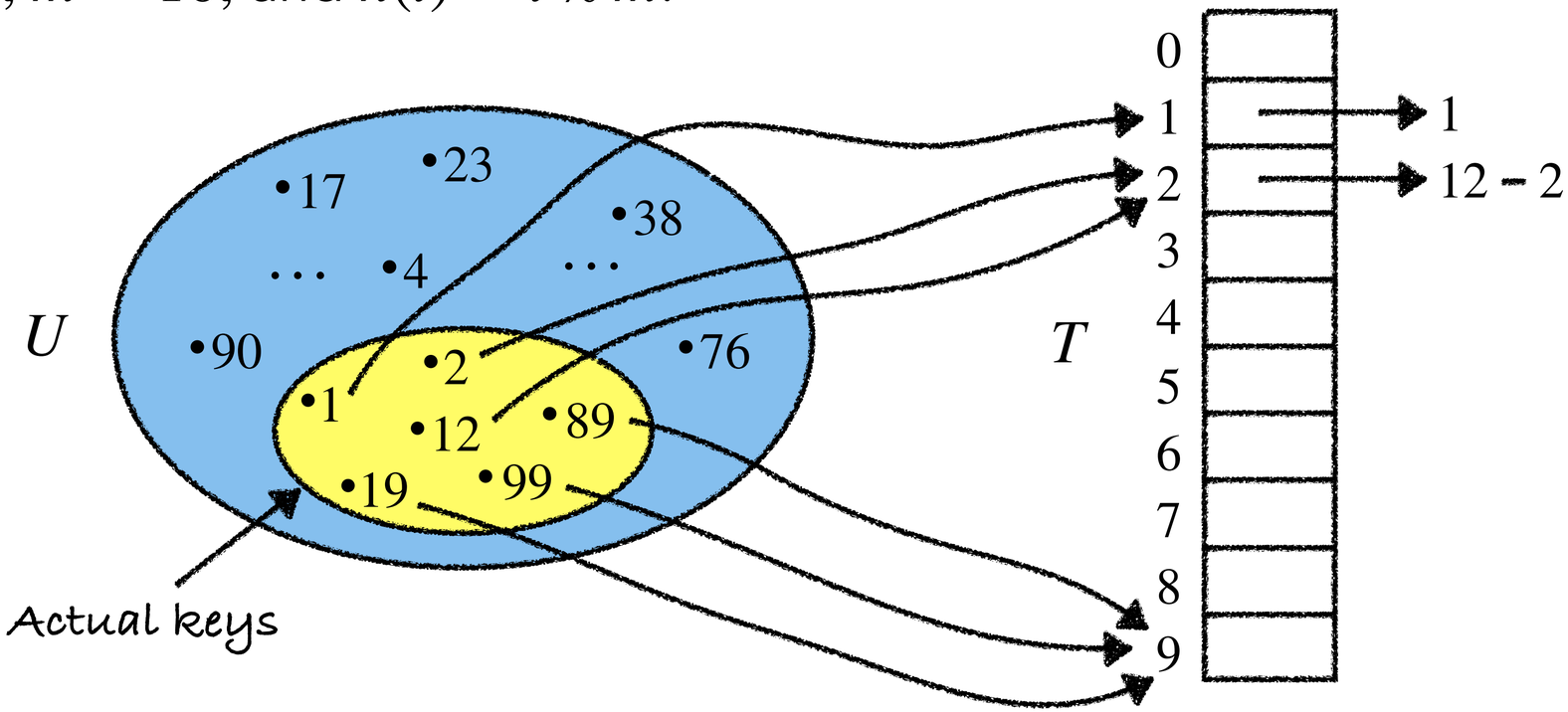
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

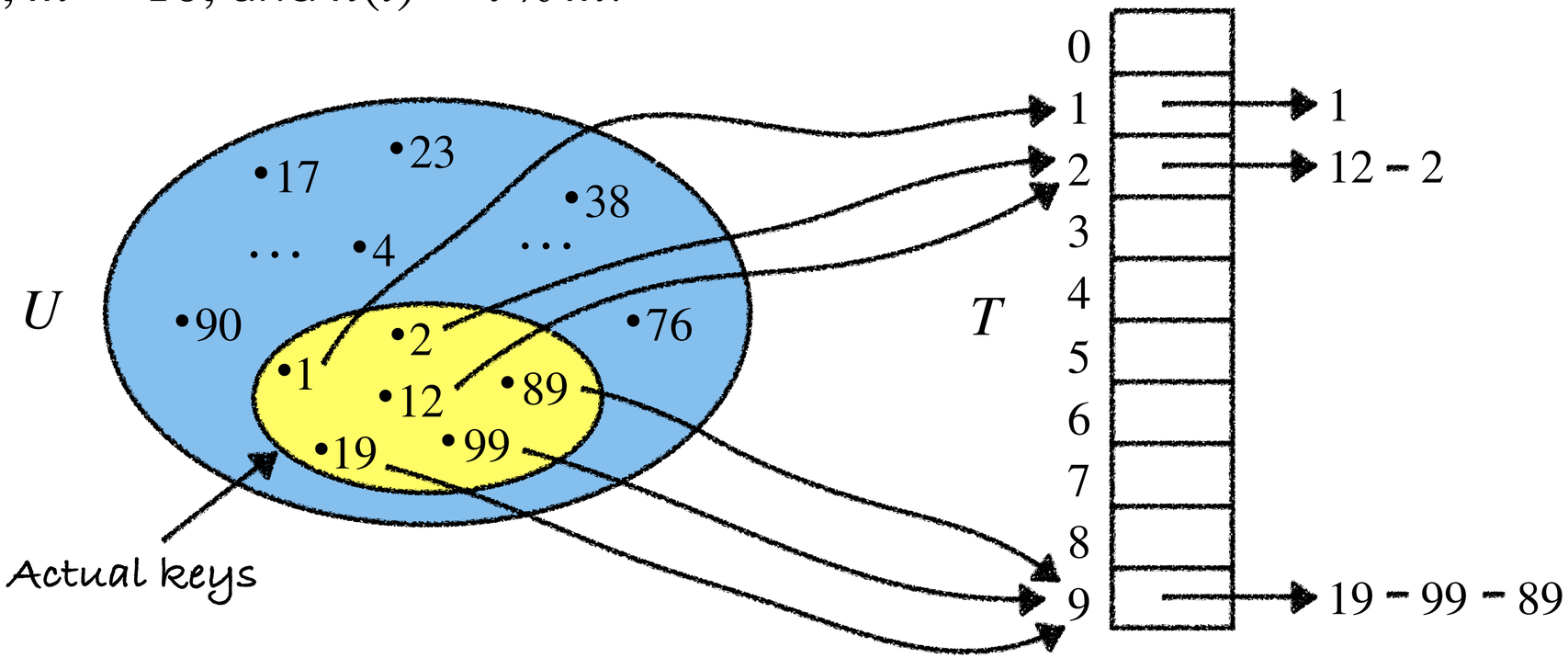
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

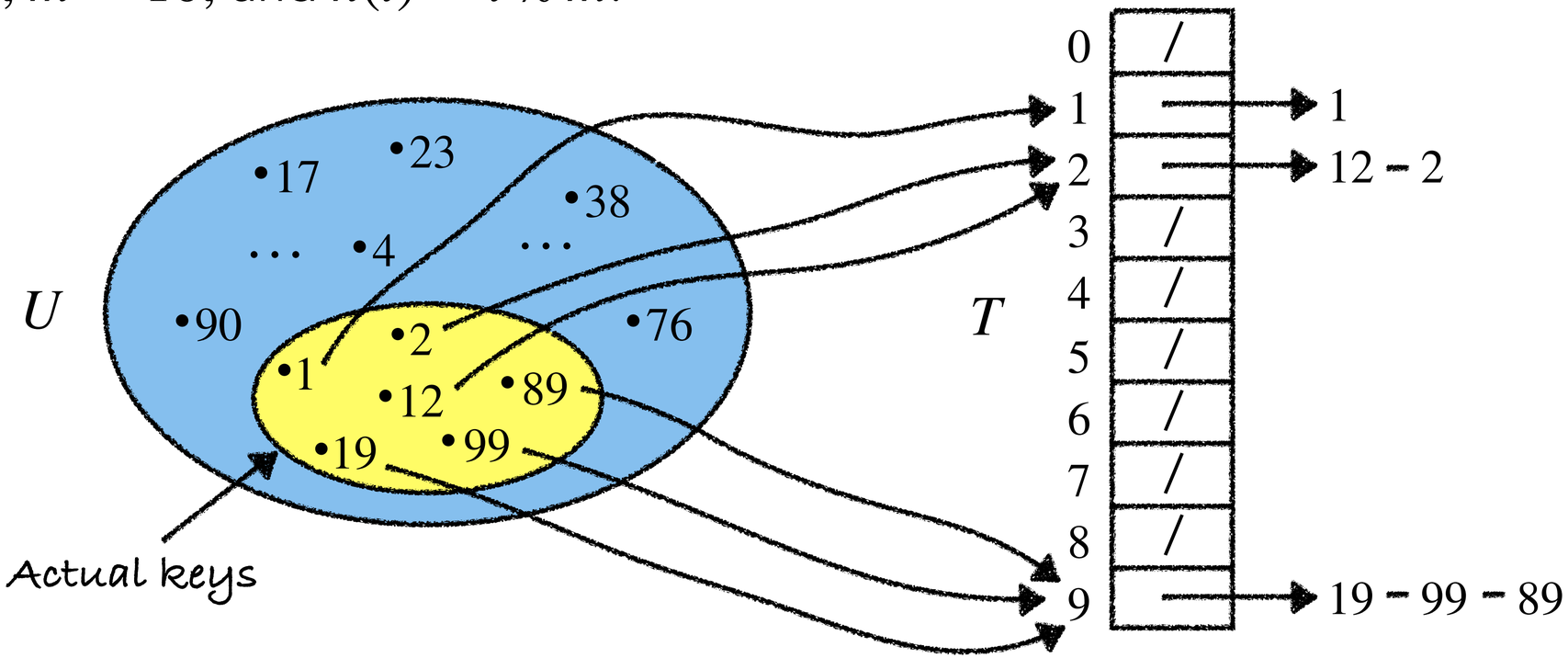
Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Hash Tables: Chaining

Chaining is a way to handle collisions, in which every slot $T[i]$ points to a linked list consisting of elements whose key hashes to the ith slot.

Example: $n = 100$, $m = 10$, and $h(i) = i \,\%\, m$.



Chained Hash Tables: Operations

Chained Hash Tables: Operations

CHAINED-HASH-INSERT(T, x)**:**

Chained Hash Tables: Operations

CHAINED-HASH-INSERT(T, x)**:**

1. **LIST-PREPEND**$(T[h(x\,.\,key)], x)$

Chained Hash Tables: Operations

CHAINED-HASH-INSERT(T, x)**:**

1. **LIST-PREPEND**$(T[h(x\,.\,key)], x)$

CHAINED-HASH-SEARCH(T, k)**:**

Chained Hash Tables: Operations

CHAINED-HASH-INSERT(T, x)**:**

1. **LIST-PREPEND**$(T[h(x\,.\,key)], x)$

CHAINED-HASH-SEARCH(T, k)**:**

1. **return LIST-SEARCH**$(T[h(k)], k)$

Chained Hash Tables: Operations

CHAINED-HASH-INSERT(T, x)**:**

1. **LIST-PREPEND**$(T[h(x\,.\,key)], x)$

CHAINED-HASH-SEARCH(T, k)**:**

1. **return LIST-SEARCH**$(T[h(k)], k)$

CHAINED-HASH-DELETE(T, x)**:**

Chained Hash Tables: Operations

CHAINED-HASH-INSERT(T, x)**:**

1. **LIST-PREPEND**$(T[h(x\,.\,key)], x)$

CHAINED-HASH-SEARCH(T, k)**:**

1. **return LIST-SEARCH**$(T[h(k)], k)$

CHAINED-HASH-DELETE(T, x)**:**

1. **LIST-DELETE**$(T[h(x\,.\,key)], x)$

Chained Hash Tables: Operations

CHAINED-HASH-INSERT(T, x)**:**

1. **LIST-PREPEND**$(T[h(x\,.\,key)], x)$

CHAINED-HASH-SEARCH(T, k)**:**

1. **return LIST-SEARCH**$(T[h(k)], k)$

CHAINED-HASH-DELETE(T, x)**:**

1. **LIST-DELETE**$(T[h(x\,.\,key)], x)$

Time complexity: $O(1)$

Chained Hash Tables: Operations

CHAINED-HASH-INSERT(T, x)**:**

1. **LIST-PREPEND**$(T[h(x\,.\,key)], x)$

Time complexity: $O(1)$

CHAINED-HASH-SEARCH(T, k)**:**

1. **return LIST-SEARCH**$(T[h(k)], k)$

Time complexity: $O(n)$ (n = size of dynamic set)

CHAINED-HASH-DELETE(T, x)**:**

1. **LIST-DELETE**$(T[h(x\,.\,key)], x)$

Chained Hash Tables: Operations

CHAINED-HASH-INSERT(T, x)**:**

1. **LIST-PREPEND**$(T[h(x\,.\,key)], x)$

Time complexity: $O(1)$

CHAINED-HASH-SEARCH(T, k)**:**

1. **return LIST-SEARCH**$(T[h(k)], k)$

Time complexity: $O(n)$ (n = size of dynamic set)

CHAINED-HASH-DELETE(T, x)**:**

1. **LIST-DELETE**$(T[h(x\,.\,key)], x)$

Time complexity: $O(1)$