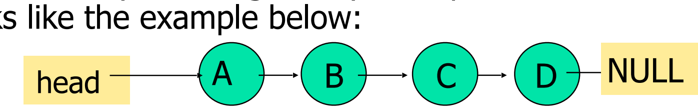
Lists and SkipList

Alon Efrat
Computer Science Department
University of Arizona

A (singly connected) link list

- Set of cells in memory. Each cell contains a key, and a pointer to the next cell.
- A pointer is the address of the next cell in memory. (in java, it is the reference)
- There is a variable **(head)** storing the address of the first cell
- The last element points to NULL.
- We could think about the memory as a large array, so a possible interpretation might looks like the example below:



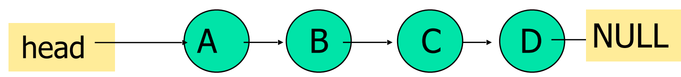
Head=106

Memory Snapshot:

Cell address	102	104	106	108	110	112	114	116	118
Key		D	A		B				C
Next cell		0 null	110		118				104

A (singly connected) linked list

- Set of cells in memory. Each cell contains a key, and a pointer to the next cell.
- A pointer is the address of the next cell in memory. (in java, it is the reference)
- There is a variable **(head)** storing the address of the first cell
- The last element points to NULL.
- We could think about the memory as a large array, so a possible interpretation might looks like the example below:



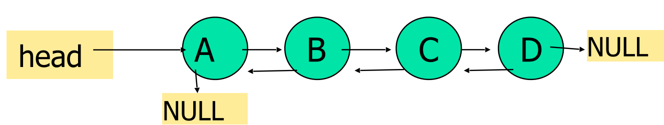
Memory Snapshot: Head=106

Cell address	102	104	106	108	110	112	114	116	118
Key		D	A		B				C
Next cell		0 null	110		118				104

- Constant time to move from a cell to the next cell
- No efficient way to move to the previous cell, or to find a key. Require linear scan.

A (doubly connected) link list

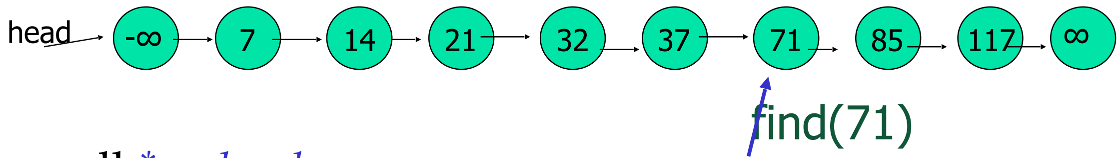
- Set of cells in memory. Each cell contains a key, and a pointer to the **next** cell and a pointer to the previous cell (**prev)**
- A pointer is the address of the next cell in memory. (in java, it is the reference)
- There is a variable **(head)** storing the address of the first cell
- The last element points to NULL.
- We could think about the memory as a large array, so a possible interpretation might looks like the example below:



Cell address	102	104	106	108	110	112	114	116	118
Key		D	A		B				C
Next cell		0 null	110		118				104
Prev cell		118	0 null		106				110

- Constant time to move from a cell to the next cell or to the previous cell
- No efficient to find a key. Require linear scan.

Searching a key *x* in a sorted linked list



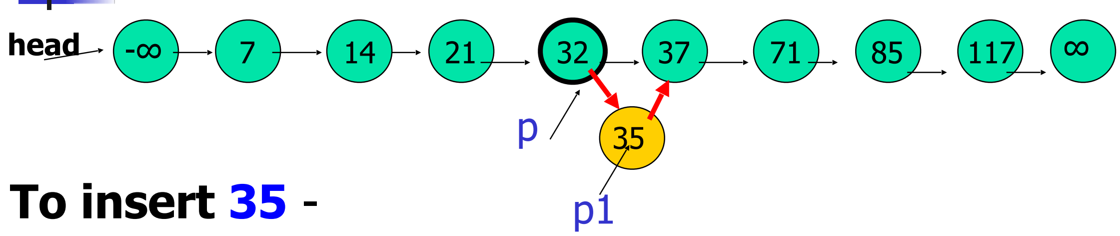
1. cell **p =head ;*
2. while *(p → key < **x**) p=p →next ;*
3. return *p ;* / / (which is either equal or larger than ***x***)

Note:

- The -∞ and ∞ elements are not "real" keys.
  - They are in the list to prevent checking special cases
- Sometimes we prefer to return the element proceeding the one containing ***x***. **Then line 2 is replaced with**

while *(p→next→key < **x**) p=p →next*

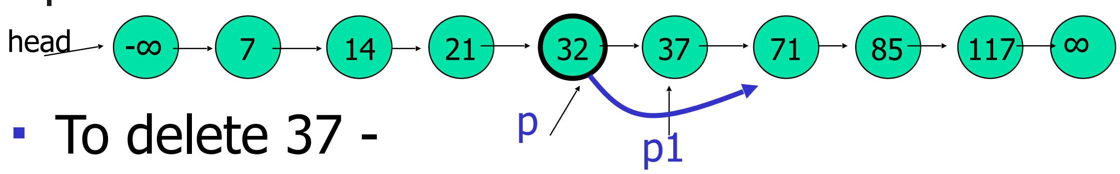
inserting a key into a Sorted linked list



To insert 35 -

- p= find(35); // find the proceeding element – the next one is > 35
- CELL *p1 = (CELL *) malloc(sizeof(CELL));
- p1 →key=35;
- p1→next = p →next ;
- p→next = p1 ;

deleteing a key from a sorted list

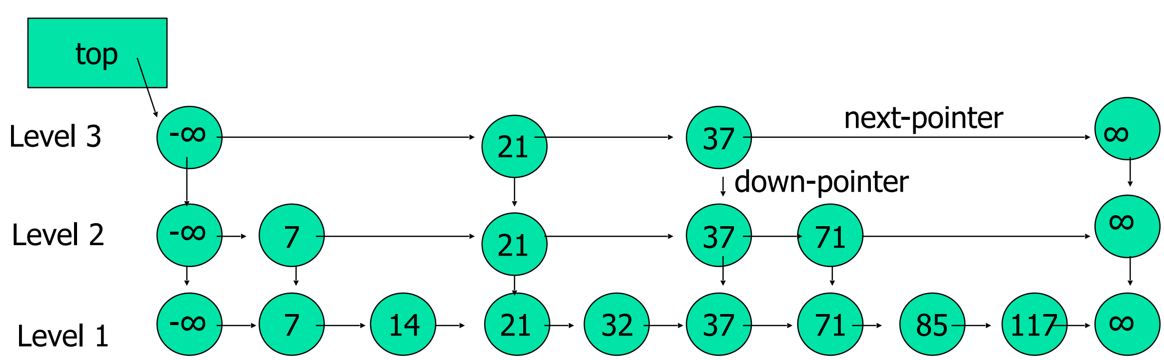


- To delete 37 -
- p=find(37); // Again find proceeding element
- CELL *p1 =p →next;
- p →next = p1→next ;
- free(p1);

SKIP LIST - A data structure for maintaining keys in a sorted order

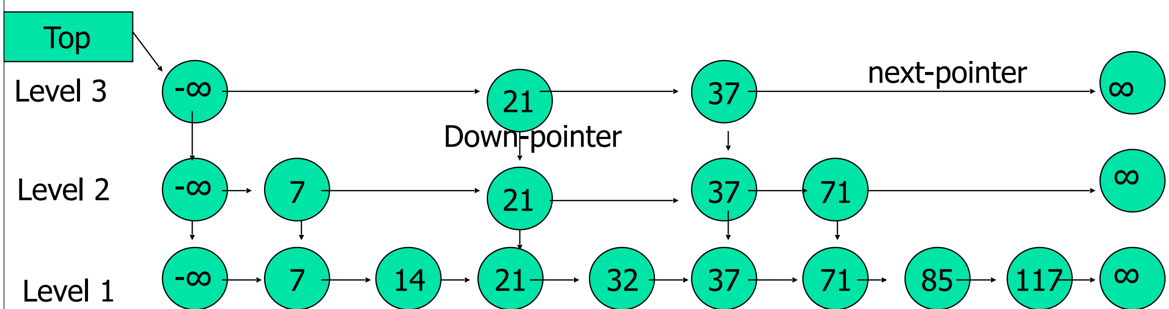
Rules:

- Consists of several **levels.**
- All keys appear in level 1
- Each level is a sorted list.
- If key ***x*** appears in level ***i***, then it also appears in all levels below level ***i***
- First element in each level has key **-∞** .
- Last element has key **+∞**
- First element in upper level is pointed to by variable ***top.***



More rules

- An element in level ***i >1*** points (via down pointer) to the element with the same key in the level below.
- Elements in the lowest level have ***down-pointer=NULL***
- Also maintain a counter specifying the number of levels.



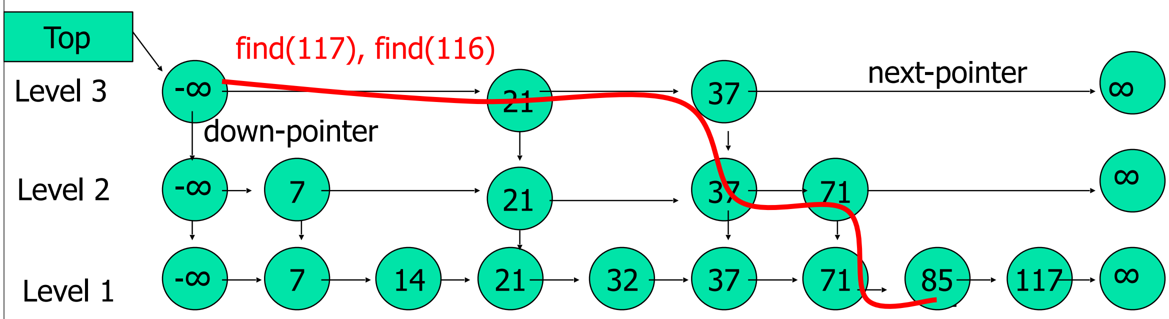
An empty SkipList

Finding **an** element with key x

- p=*top* ;
- while(1){
  - while (p→*next*→*key* $\leq$ $\mathbf{x}$) p=p→*next*;
  - if (p→*down* == *NULL*) **return** p
  - p=p→*down* ;
- }

If the key x is in SL, we return a pointer to the lowest element contain x.

If x is not in SL, return pointer to lowest predecessor.



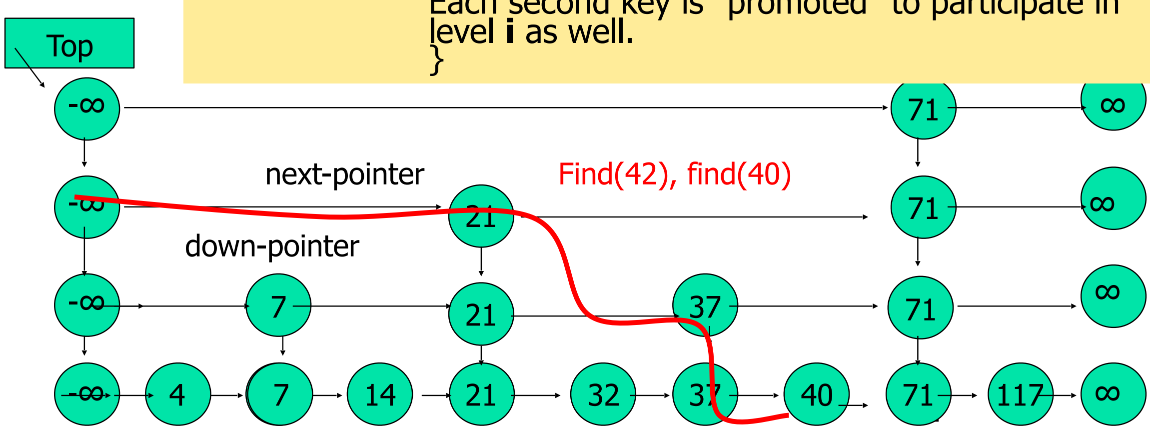
A “perfect” SkipList

Most SL as re not perfect.
Hard to maintain

A SL is Perfect if between every two consecutive keys of level **i** there is exactly one key of level **i-1.**

Scheme for creation a well-performing SL

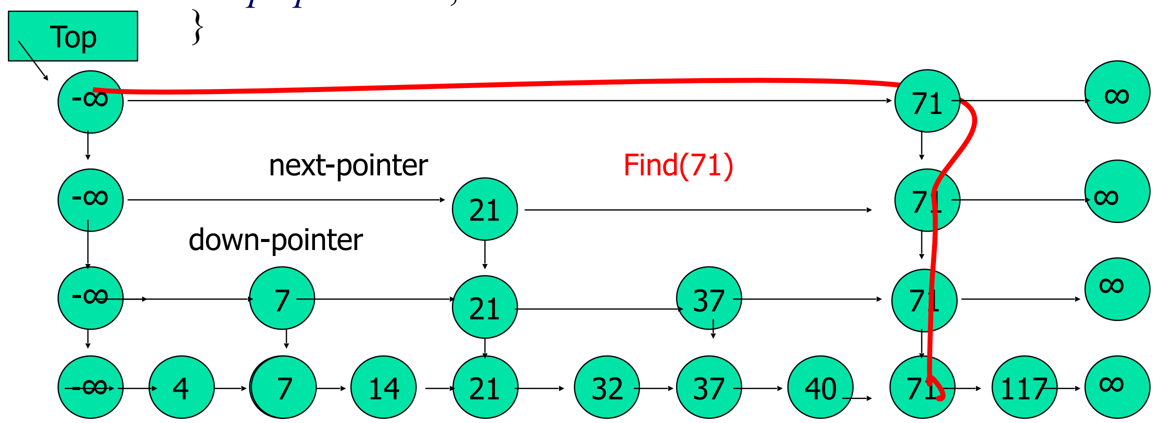
- Start from Level 1 (lowers level)
- For **i=2,3**…
  - Generation of Level **i**: }
    - we scan the keys in level **i-1**.
    - Each second key is “promoted” to participate in level **i** as well.
    - }



Search in a "perfect" SkipList

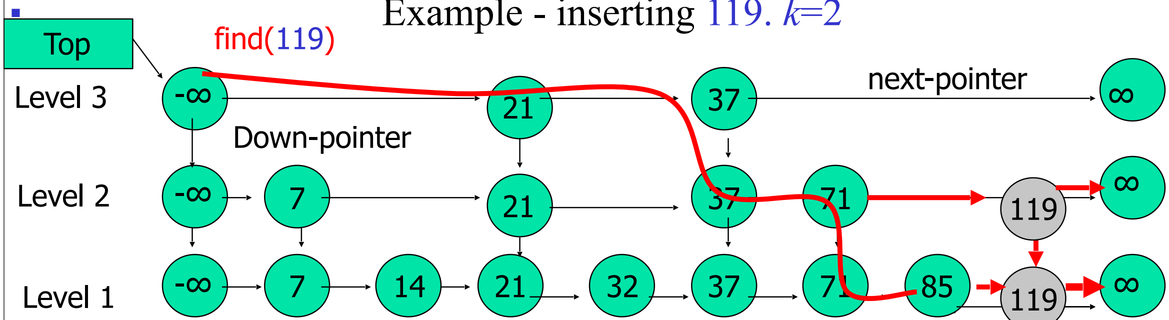
Another example

```
p=top ;
while(1){
    while (p→next→key ≤ x )
        p=p→next;
    if (p→down == NULL ) return p
    p=p→down ;
}
```



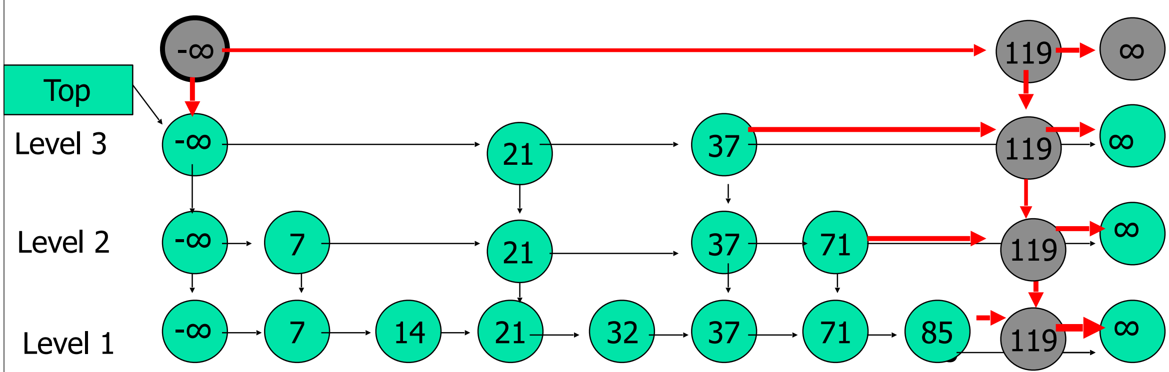
Inserting new element x
(the resulting SL will not be perfect)

- Determine ***k* ≥1** defined as the number of levels in which ***x*** participates (explained later how)
- Perform ***find(x)***, but once the search path is in one of the lowest ***k*** levels:
  - ***x*** is inserted after the elements at which the search path branches down or terminates.
  - The *next-pointer* behave like a "standard" linked list
  - The *down pointer(s)* point between themselves.
- Example - inserting 119. *k*=2



Inserting an element - cont.

- If ***k*** is larger than the current number of levels, add new levels (and update *top*, and *num_of_levels* counter)
- Example - **insert(119)** when ***k=4***
- `Heuristic: Add at most one new level (not needed for the analysis)`

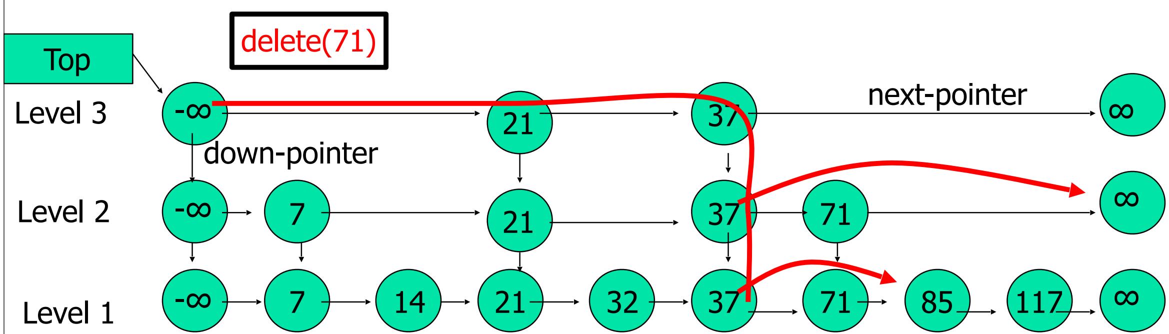


Determining k

- ***k*** **-** the number of levels at which an element *x* participate.
- Use a random function *OurRnd()* --- returns 1 or 0 (True/False) with equal probability.
  - ***k****=1 ;*
  - *While(OurRnd()==1)* ***k****++ ;*

Deleteing a key *x*

- Find ***x*** in all the levels it participates, using find(***x***).
- During the "find", delete ***x*** from each level it participates using the standard "delete from a linked list" method.
- If one or more of the upper levels become empty, remove them (and update *top* and *num_of_levels*)



"expected" space requirement

- **Claim:** The expected number of elements is O(*n*).
- The term "**expected**" here refers to the experiments we do while tossing the coin (or calling *OurRnd()*). No assumption about input distribution.
- So imagine a given set, given set of operations insert/del/find, but we repeat many time the experiments of constructing the SL, and count the #elements.

Facts about SL

- **Def:** The **height** of the SL is the number of levels
- **Claim:** The expected number of levels is O(log n)
- (here n is the number of keys)
- **"≅ Proof"** (*A rigorous proof coming later)*
  - The number of elements participate in the lowest level is n.
  - Since the probability of an element to participates in level 2 is ½, the **expected** number of elements in level 2 is **n/2**.
  - Since the probability of an element to participates in level 3 is *1/4*, the expected number of elements in level 3 is *n/4*.
  - …
  - The probability of an element to participate in level j is $(1/2)^{j-1}$ so number of elements in this level is $n/2^{j-1}$
  - So after log(n) levels, no element is left.

Facts about SL

- **Claim:** The expected number of elements is O(n).
- (here n is the number of keys)
- **"≅ Proof"** (Real proof – later)
  - The total number of elements is

    $n+n/2+n/4+n/8\ldots \le n(1+1/2+1/4+1/8\ldots) = 2n$. QED

And a real proof . Lets $x_{i,l}$ denote a random variable which is 1 if key k_i participates in level l, and $x_{i,l}$=0 if this key does not participate in this level.

The number of elements in the SL is clearly $\sum_{i=1}^{n} \sum_{l=1}^{MaxLevel} x_{i,l}$.

Remember that the probability of a key to make it to level ℓ is $\frac{1}{2^{\ell-1}}$

The expected number of elements is

$$E\left(\sum_{i=1}^{n} \sum_{l=1}^{MaxLevel} x_{i,l}\right) = \sum_{i=1}^{n} \sum_{l=1}^{MaxLevel} E(x_{il}) = \sum_{i=1}^{n} \sum_{l=1}^{MaxLevel} Pr(x_{i,l}=1) = \sum_{i=1}^{n} \sum_{l=1}^{MaxLevel} \frac{1}{2^{l-1}} \le \sum_{i=1}^{n} 2 = 2n$$

To reduce the worst case scenario, we verify during insertion that k (the number of levels that an element participates) in) is $\le \log n$

"Conclusion": The expected storage is $O(n)$

More facts

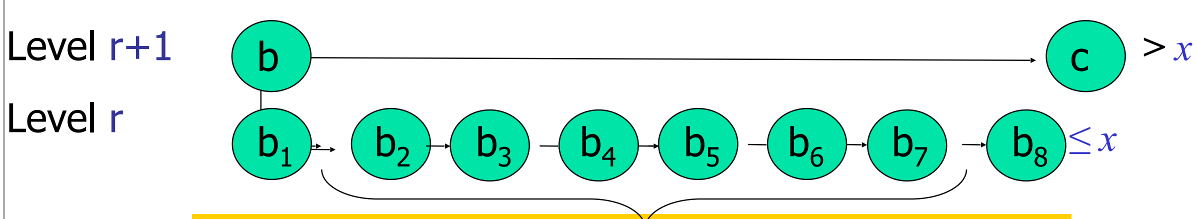
- **Thm**: The expected time for find/insert/delete is O(log n)

- **Proof** For all Insert and Delete, the time is ≤ expected #elements scanned during find(x) operation.

- Will show: Need to scan expected O(log n) elements.

Thm: Expected time for `**find'** operation is O(log n)

- ≅**Proof** – we know that there are O(log n) levels. Will show that we spend O(1) time in each level.
- Assume during find(x), we scanned t elements, (for t>8) in level r. Assume first that r is not the upper level.
  - (the search visited **b**, branched down to $\mathbf{b_1}$ and then visited $\mathbf{b_{2...}b_8}$ (not sure what happed before or after)



All smaller than x
None of these 7 elements reached level r+1 (why?)

The probability that none of these 7 elements reached level r+1 is $1/2^7$. For larger value of 7 – very slim.

Bounding time for insert/delete/find

- Putting it together: The expected number of elements scanned in each level is O(1)
- There are O(log n) levels
- Total time is O(log n)
- As stated, getting bounds for time for insert/delete are similar

How likely is it to see a ``too-tall" SL ?

- We will prove a bound on the height. Similar bounds could be proven for similar properties.
- The question what is ``too-tall" is up to the user.
- Of course, the larger n is, the more level we expect to see. So lets ask the user to pick a value Z.
- We will compute the how likely is it that the the number of levels is is at least

$Z\log_2 n$, where $Z=1,2,3\ldots$

That is, we estimate the probability that the height of the SL is

- $\log_2 n$
- $2 \log_2 n$
- $3 \log_2 n$
- $4 \log_2 n$
  - …

Reminder from probability

- Assume that **A,B** are two events. Let
  - Pr(**A**) be the probability that **A** happens,
  - Pr(**B**) be the probability that **B** happens
  - Pr(**A** $\cup$ **B**) is the probability that either event **A** happens or event **B** happens (or both).
- So probably that at least one of them happened is

  Pr(**A**)+Pr(**B**)-Pr(**A** $\cap$ **B**) $\leq$ Pr(**A**)+Pr(**B**)

Similarly, for 3 Events $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{A_3}$. The probability that **at least** one of them happens

$$\Pr(\mathbf{A_1} \cup \mathbf{A_2} \cup \mathbf{A_3}) \leq \Pr(\mathbf{A_1}) + \Pr(\mathbf{A_2}) + \Pr(\mathbf{A_3})$$

Example: In a roulette, the result is a number **k** between **1..38**

- Event **A**: **k** is even. Pr(**A**)=Pr(**k** is even) = 19/38 = 0.5
- Event **B** : **k** is divided by 3. Pr(**B**)= 12/38=0.315
- Pr(**A** or **B**) = Pr(**A** $\cup$ **B**)=

  Pr((**k** is divided by 2) or (**k** is divided by 3)) $\leq$ 0.5+0.315=0.815

Pick your favorite number k.
What is the probability that the SL has >k levels ?

Answer: $\leq n/2^k$

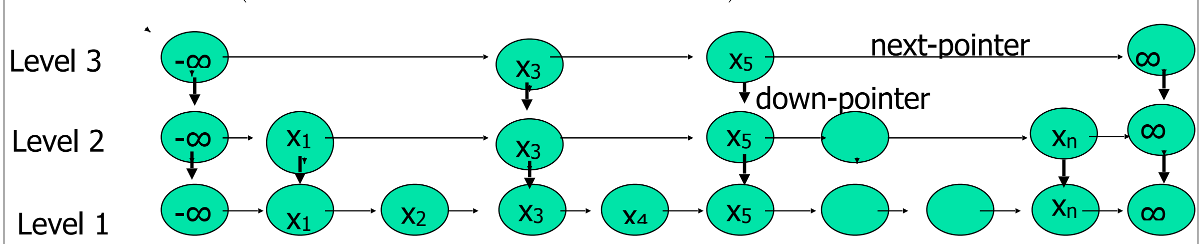
$$\Pr(\text{height of the SkipList} \geq k) =$$

$$\Pr\{(\mathbf{x}_1 \text{ participates in more than } k \text{ levels}) \text{ OR}$$
$$(\mathbf{x}_2 \text{ participates in more than } k \text{ levels}) \text{ OR}$$
$$(\mathbf{x}_3 \text{ participates in more than } k \text{ levels}) \text{ OR}$$
$$\vdots$$
$$(\mathbf{x}_n \text{ participates in more than } k \text{ levels})\}$$

$\leq$ /*Apply the principle from the previous slide*/

$$\Pr(\mathbf{x}_1 \text{ participates in more than } k \text{ levels}) + \quad 1/2^k +$$
$$\Pr(\mathbf{x}_2 \text{ participates in more than } k \text{ levels}) + \quad 1/2^k +$$
$$\Pr(\mathbf{x}_3 \text{ participates in more than } k \text{ levels}) + \quad 1/2^k +$$
$$\vdots$$
$$\Pr(\mathbf{x}_n \text{ participates in more than } k \text{ levels}) = \quad 1/2^k =$$

$$= n/2^k$$



But how likely is that the SL is too tall ?

- Assume the keys in the SL are $\{x_1, x_2, \ldots x_n\}$.
- The probability that x_1 participates in $\geq$ **k+1** levels is 2^{-k} .
  - (same probability for all x_i).
  - Define: A_1 is the event that x_1 participates in $\geq$**k+1** levels.
  - $Pr(A_1)=2^{-k}$
  - Define: A_j is the event that x_j participates in $\geq$**k+1** levels.
  - $Pr(A_j) = 2^{-k}$ (for every **j**)
- If the height of SL $\geq$**k+1** then
  at least one of the x_j participates in $\geq$**k+1** levels.
- The probability that **any** x_i (one or more) participates in $\geq$ **k+1** levels is $\leq Pr(A_1) + Pr(A_2) + \ldots + Pr(A_n) = n\, 2^{-k}$
- **This is the probability that the height of the SL is $\geq$k+1.**

But how likely is that the SL is tall ?

- The probability that **any** x_i participates in at least **k** levels is $\leq n2^{-k}$. Then the height of the SL $\geq$ **k+1.**
- **Ignore the `+1'**
- If none of the x_i 's is at level $\geq$**k** then the height is $\leq$**k.**

- Recall $y^{(ab)}=(y^a)^b = (y^b)^a$
- $2^{\log_2 n} = n$ and $2^{5(\log_2 n)} = (n)^5$
- Write $k = Z \log_2 n$,

- Want to find: The probability that the height is **Z** times $\log_2 n$.
- That is, Twice $\log_2 n$, 3 time $\log_2 n$, 4 times $\log_2 n$ …

So how likely is it that the height of SL is $> Z \log n$

- The probability that **any** x_i participates in $>$ **k** levels is $\leq n/2^k$
- If none of the x_i 's is at level $\geq k$ then the height is $\leq k$.
- Recall $2^{(ab)} = (2^a)^b = (2^b)^a$
- Write $k = (\log_2 n) Z$
- Therefor $2^k = 2^{(\log_2 n) \cdot Z} = (2^{\log_2 n})^Z = n^Z$
- So the probability of seeing a SkipList with more than $Z \log n$ levels is

$$\leq n/2^k = n/n^Z = 1/n^{z-1}$$

- Lets play with some examples, to see if this is good news or bad news
- Lets pick **n=1000**.
- The probability that the heigh>**7** $\log_2 n$ is $\leq 1/1000^6 = 1/10^{18}$ … So the probability that the height$\leq$**7**$\log_2 n$ is $\geq$ **1-1/10**^{18}
- The prob. that the heigh**<10**$\log_2 n$ **is** $\geq$ **1-1/10**^{27}
- Conclusion: In this case (and in many other randomized algorithms) the probability of success is so high, that practically we can ignore it (higher chance of a lighting strike)

But how likely is that the SL is tall ?

- The probability that **any** x_i participates in at least **k** levels is $\leq n2^{-k}$ **.** Then the height of the SL $\geq$ **k+1.**
- Want to find: The probability that the height is **Z** times $\log_2 n$.
- Twice $\log_2 n$, 3 time $\log_2 n$, 4 times $\log_2 n$ …
- Then $2^{-k} = 2^{-(Z \log n)} = (2^{\log n})^{-Z} = n^{-Z} = 1/n^Z$
- So $n2^{-k} \leq n / n^Z = 1/n^{Z-1}$
- This is the probability that the height of SL $\geq Z \log_2 n$
- Example: **n=1000**.
- The probability that the heigh$\geq$**7** $\log_2 n$ is $\leq 1/1000^6 = 1/10^{18}$
- The probability that the height**<7**$\log_2 n$ is $\geq$ **1-1/10**^{18}
- The prob. that the heigh**<10**$\log_2 n$ **is** $\geq$ **1-1/10**^{27}