Contrary to Popular Belief Incremental Discretization can be Sound, Computationally Efficient and Extremely Useful for Streaming Data

By Geoffrey I. Webb

Faculty of Information Technology, Monash University, Victoria, Australia

Presented by Massy BOURENNANI

29 November 2017

Soft Skill Seminar, D&K 2017

The wolf of wall street: the french adventure
Script by : D&K 2017 students

SPOILER ALERT

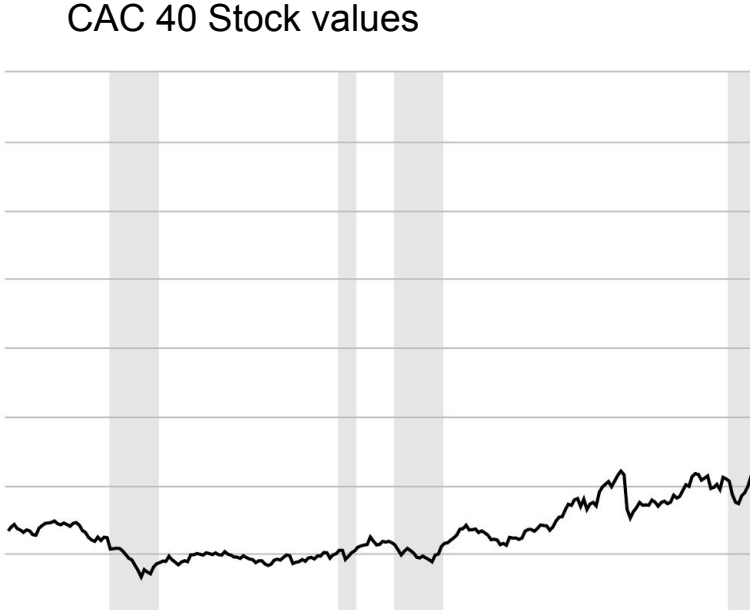
ML

The wolf of wall street: the french adventure
Script by : D&K 2017 students


MovieStillsDB.com

2 years later (… paperwork) he went to France

Motivation



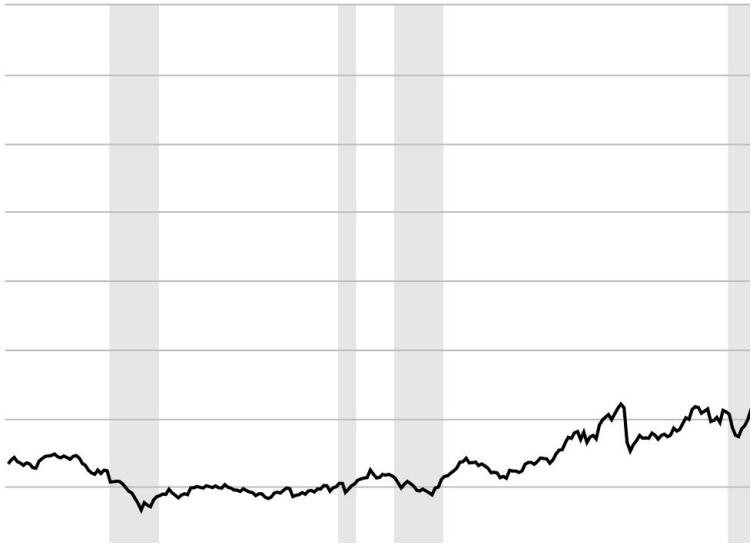
Motivation

- discretization : is to divide a continuous data into a set of buckets (bins)

- bins
  ] -∞, b_1],]b_2,b_3] …]b_{m-1},+∞ [

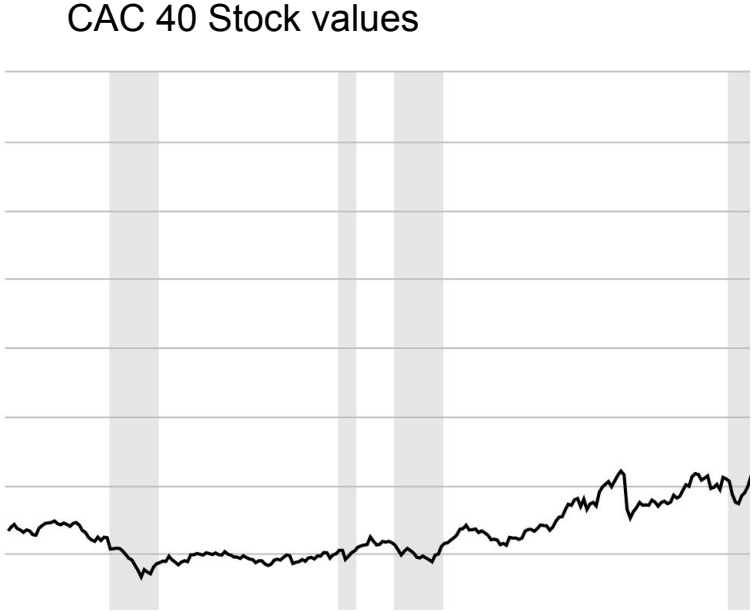
- **b_1 b_2...b_{m-1}** the cut points

CAC 40 Stock values

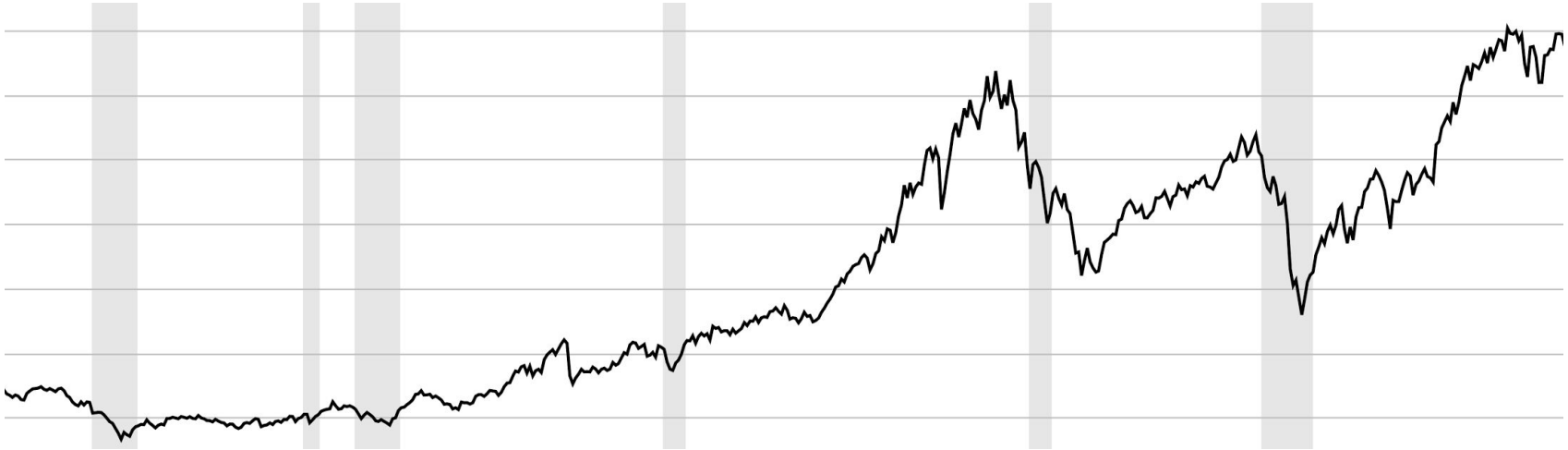


Issues

- cut points may vary over time
- unknown distribution of the data
- not easy to determine relevant intervals
- static intervals : loss of information
- some learning algorithms requires “same” bins over time



Issues



CAC 40 Stock values

Solution

- find a way to synchronize the discretization with the data distribution
- you want to keep the bins as informative as possible about the original data

Solution: IDA & IDAW

- **I**ncremental **D**iscretization **A**lgorithme (IDA)
- approximate **quantile-based** discretization on the entire data stream encountered to date
- keeps the meaning of the bins even though the cut values drift over time

Solution: IDA & IDAW

- **I**ncremental **D**iscretization **A**lgorithme (IDA)
- approximate **quantile-based** discretization on the entire data stream encountered to date
- keeps the meaning of the bins even though the cut values drift over time

- **I**ncremental **D**iscretization **A**lgorithme with **W**indow (IDAW)
- allow the cut points to track a non stationary distribution

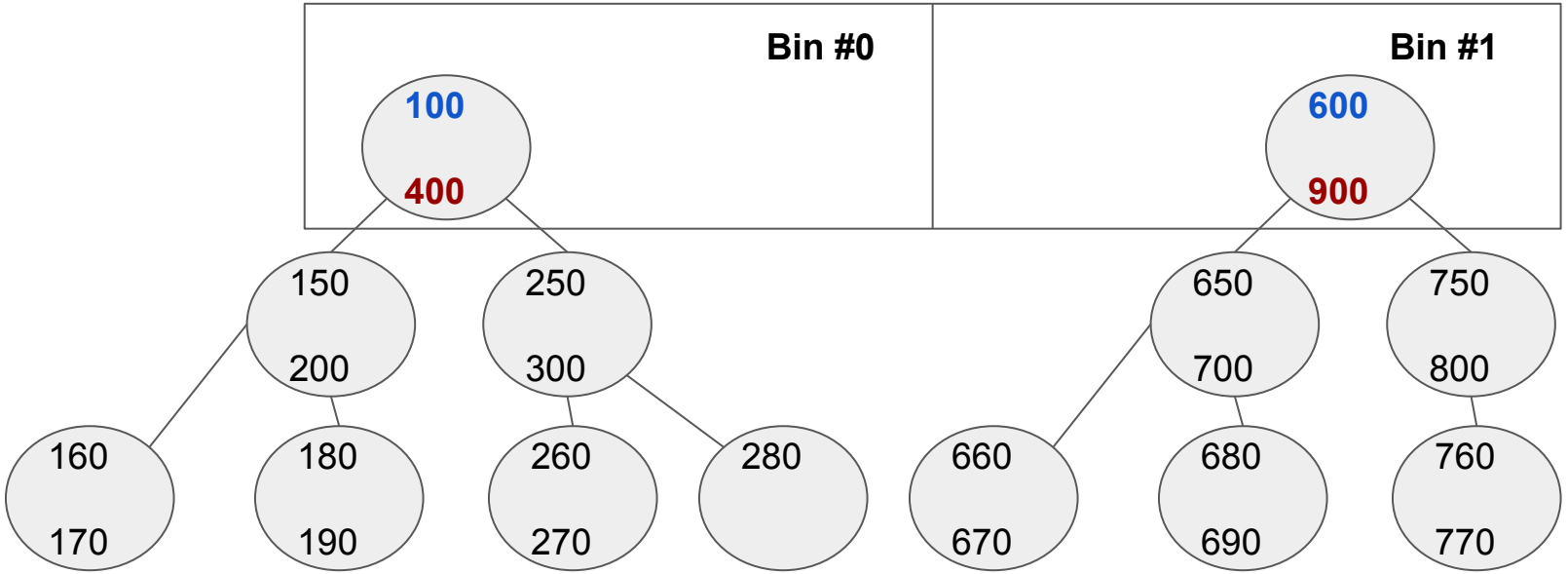
Solution: IDA

- don’t consider the entire data stream
- samples the data using reservoir sampling
  - more efficient
  - not feasible to store all the encountered data
  - put tight bounds on the estimated quantiles
- store each bin in an interval heap
- the discretization of an attribute is represented with a vector of heaps

IDA example

size of the reservoir : 28
number of bins: 2

(25: 290) (26: 780) (27: 790) …

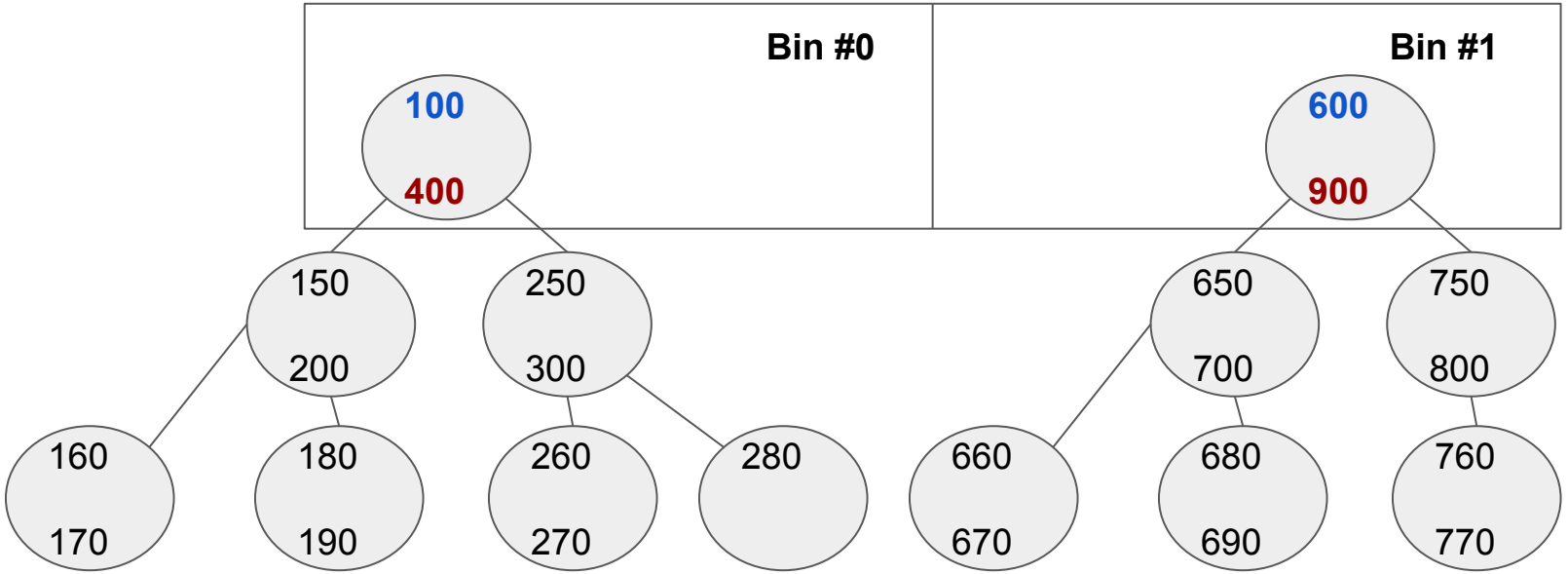


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 25
t = i mod m = 25 mod 2 = 1
j = 0
---
insert 290 in Bin #0
move max Bin #0 to Bin #1
reorganize the heaps

(25: 290) (26: 780) (27: 790) …

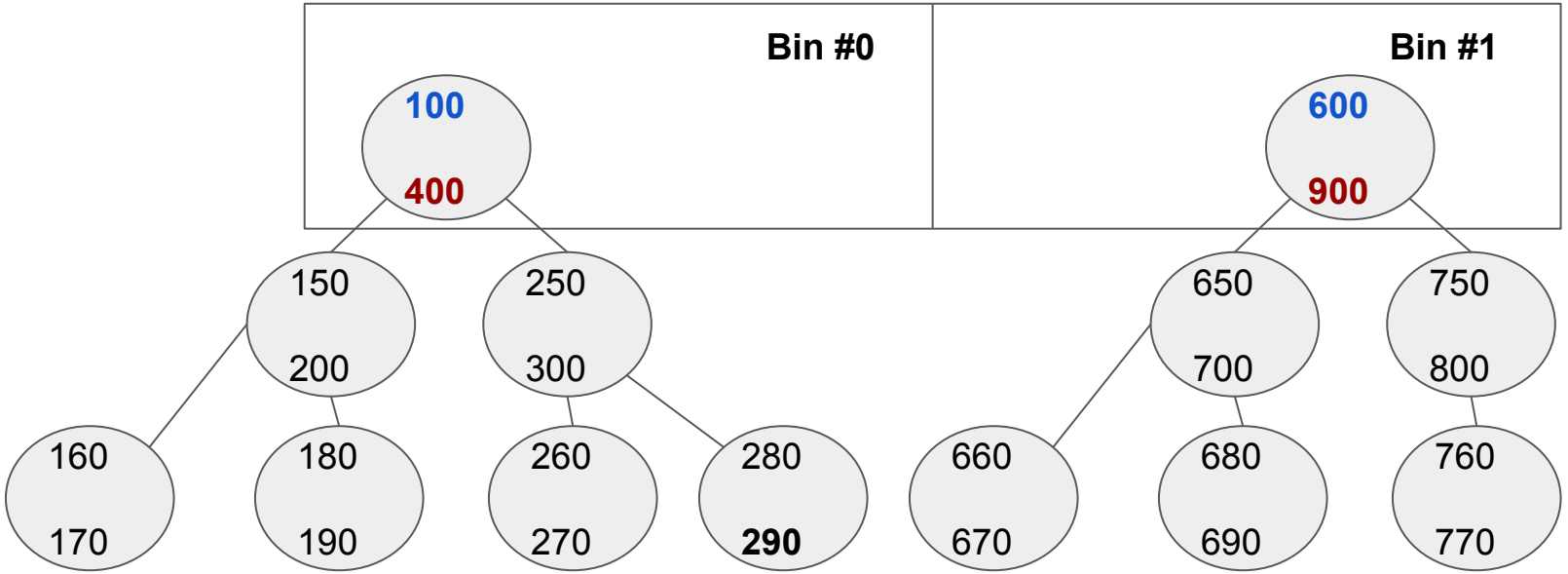


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 25
t = i mod m = 25 mod 2 = 1
j = 0
---
insert 290 in Bin #0
move max Bin #0 to Bin #1
reorganize the heaps

~~(25: 290)~~ (26: 780) (27: 790) …

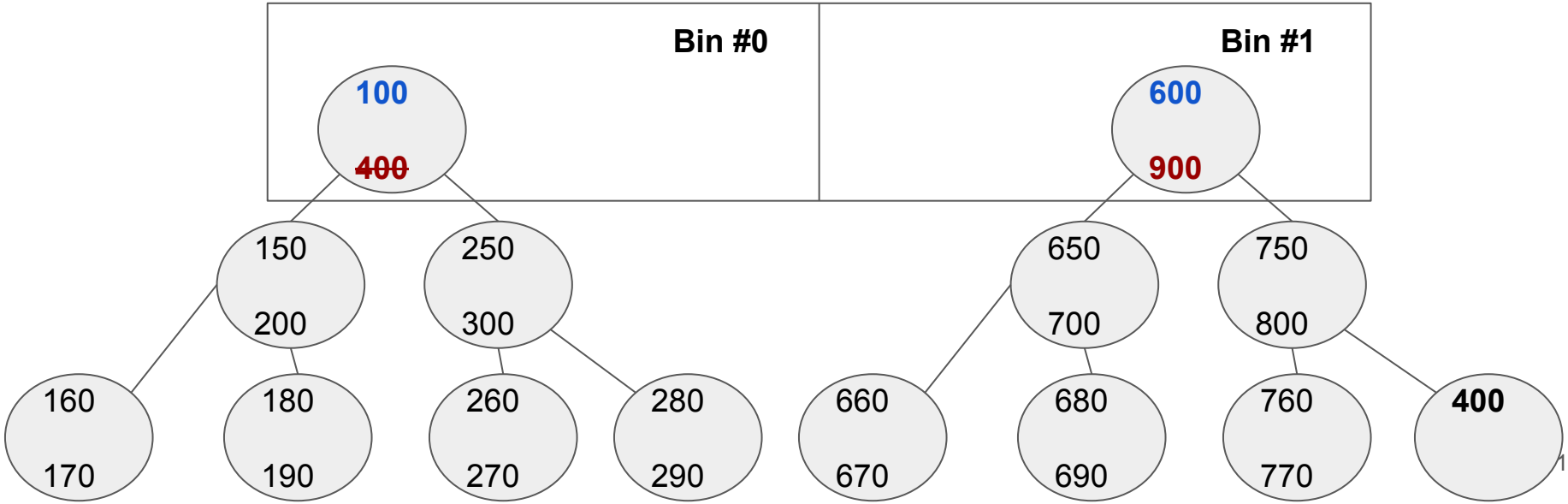


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 25
t = i mod m = 25 mod 2 = 1
j = 0
---
insert 290 in Bin #0
move max Bin #0 to Bin #1
reorganize the heaps

~~(25: 290)~~ (26: 780) (27: 790) …

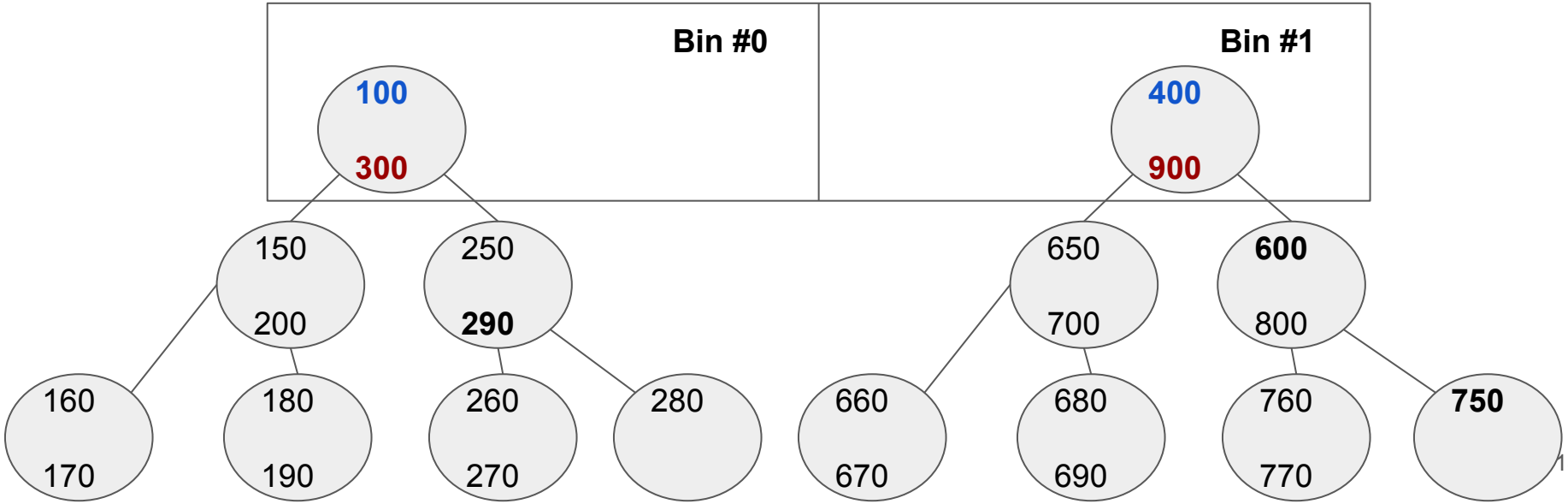


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 25
t = i mod m = 25 mod 2 = 1
j = 0
---
insert 290 in Bin #0
move max Bin #0 to Bin #1
reorganize the heaps

(**26: 780**) (27: 790) …

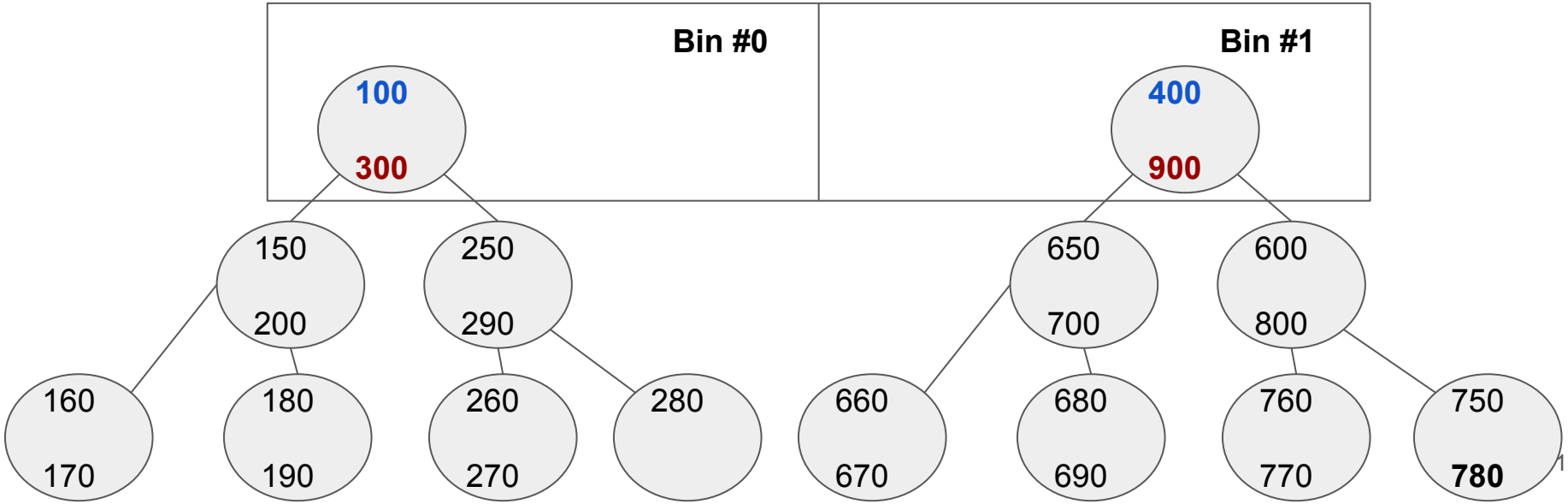


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 26
t = i mod m = 26 mod 2 = 0
j = 1
---
insert 780 in Bin #1
move min Bin #1 to Bin #0
reorganize the heaps

(~~26: 780~~) (27: 790) …

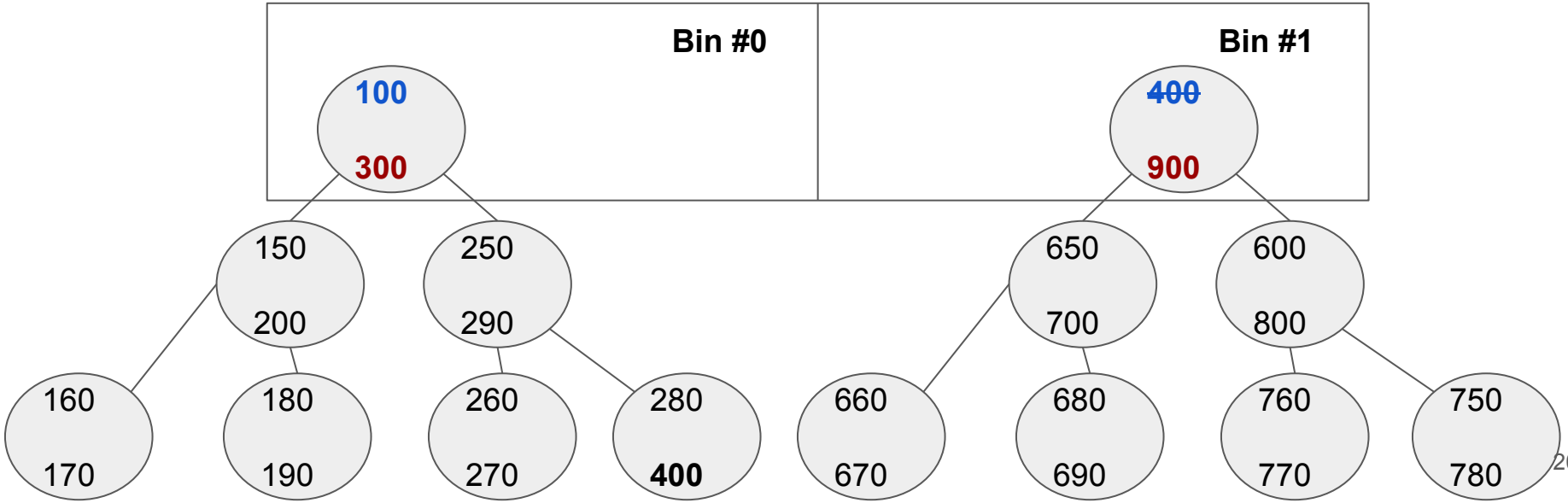


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 26
t = i mod m = 26 mod 2 = 0
j = 1
---
insert 780 in Bin #1
move min Bin #1 to Bin #0
reorganize the heaps

(~~26: 780~~) (27: 790) …

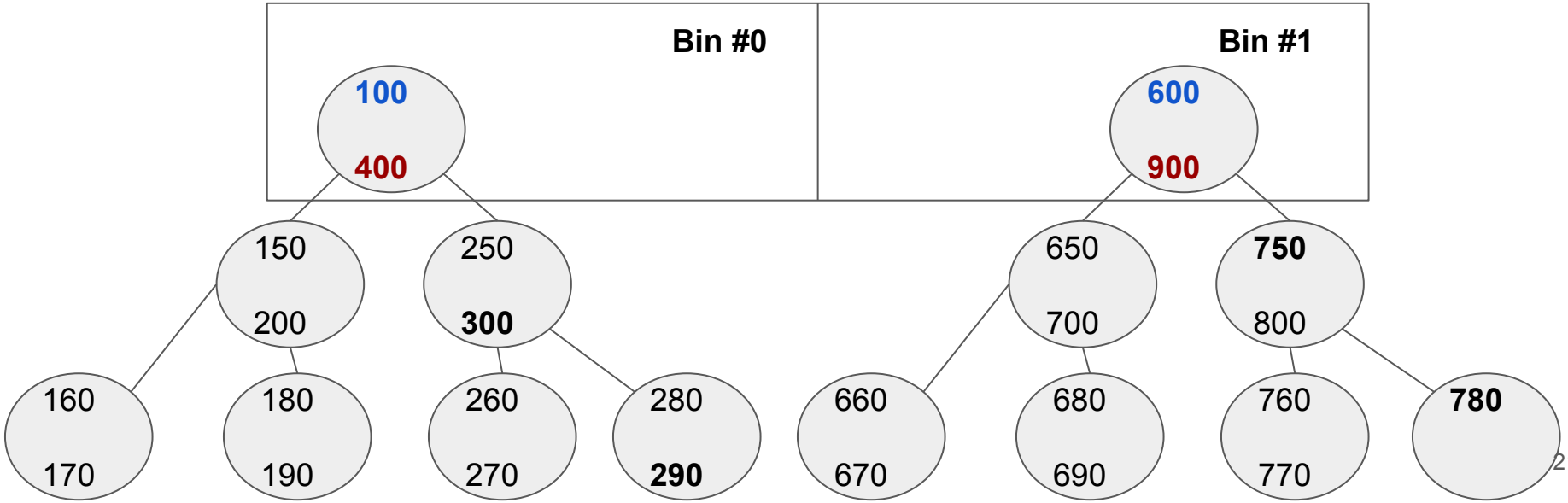


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 26
t = i mod m = 26 mod 2 = 0
j = 1
---
insert 780 in Bin #1
move min Bin #1 to Bin #0
reorganize the heaps

(~~26: 780~~) (27: 790) …

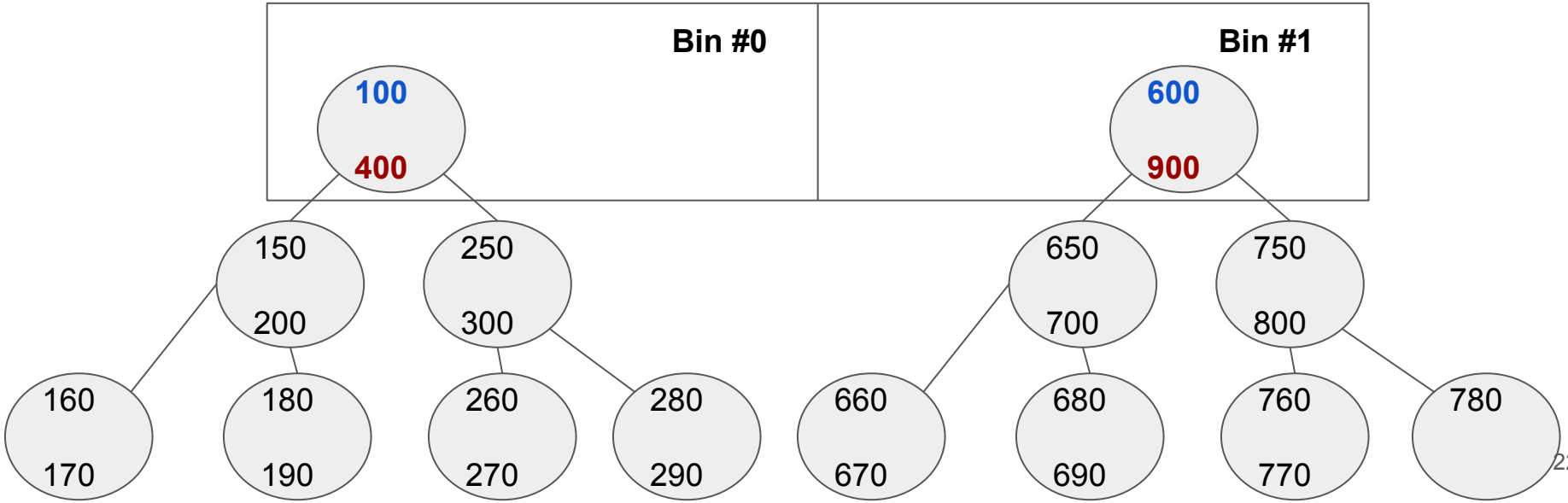


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 27
t = i mod m = 27 mod 2 = 1
j = 1
---
insert 790 in Bin #1
move min Bin #1 to Bin #0
reorganize the heaps

(**27: 790**) …

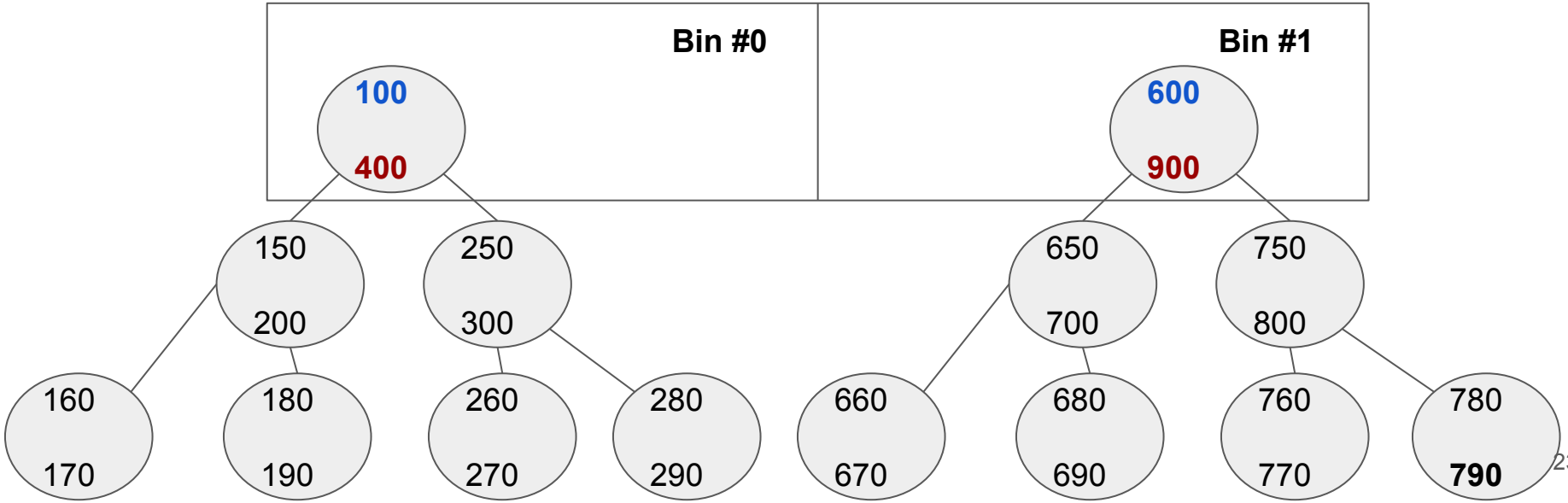


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 27
t = i mod m = 27 mod 2 = 1
j = 1
---
insert 790 in Bin #1
move min Bin #1 to Bin #0
reorganize the heaps

(~~27: 790~~) …

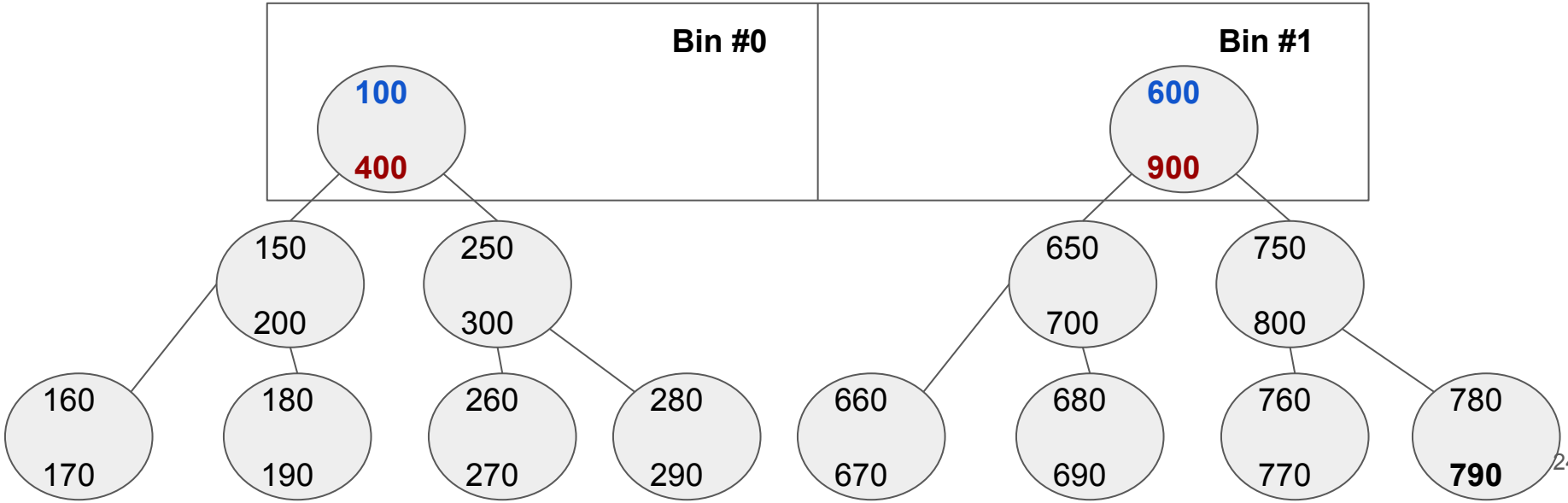


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 27
t = i mod m = 27 mod 2 = 1
j = 1
---
insert 790 in Bin #1
~~**move min Bin #1 to Bin #0**~~ **because t=j**
~~**reorganize the heaps**~~

(~~27: 790~~) …

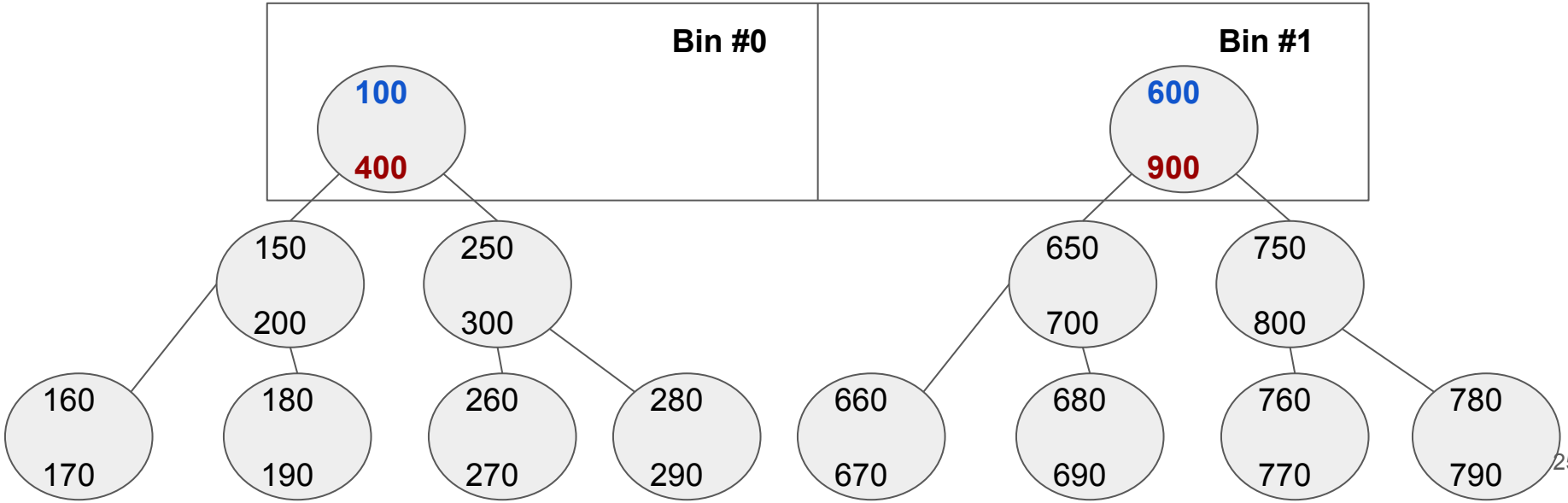


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 28
keep = random () < s/29
---
if keep :
  remove a random element from Bin #0 U Bin #1
  insert the new element
else skip the element

(28: 850)….

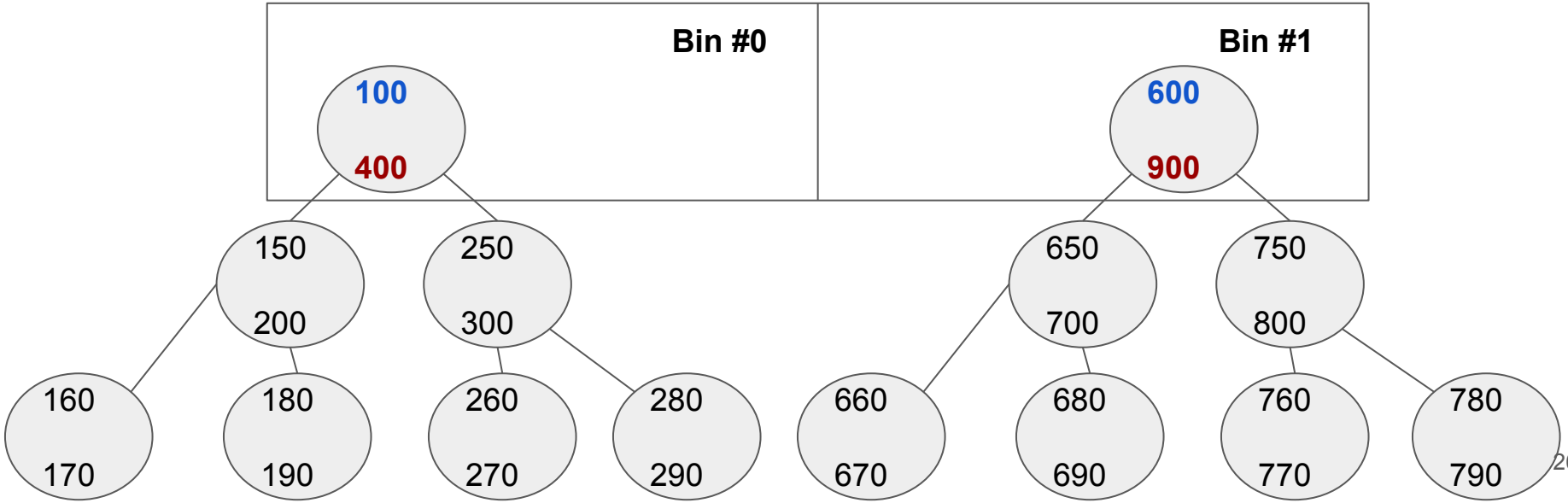


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 28
keep = random () < s/29 #true
---
if keep :
   remove a random element from Bin #0 U Bin #1
   insert the new element
else skip the element

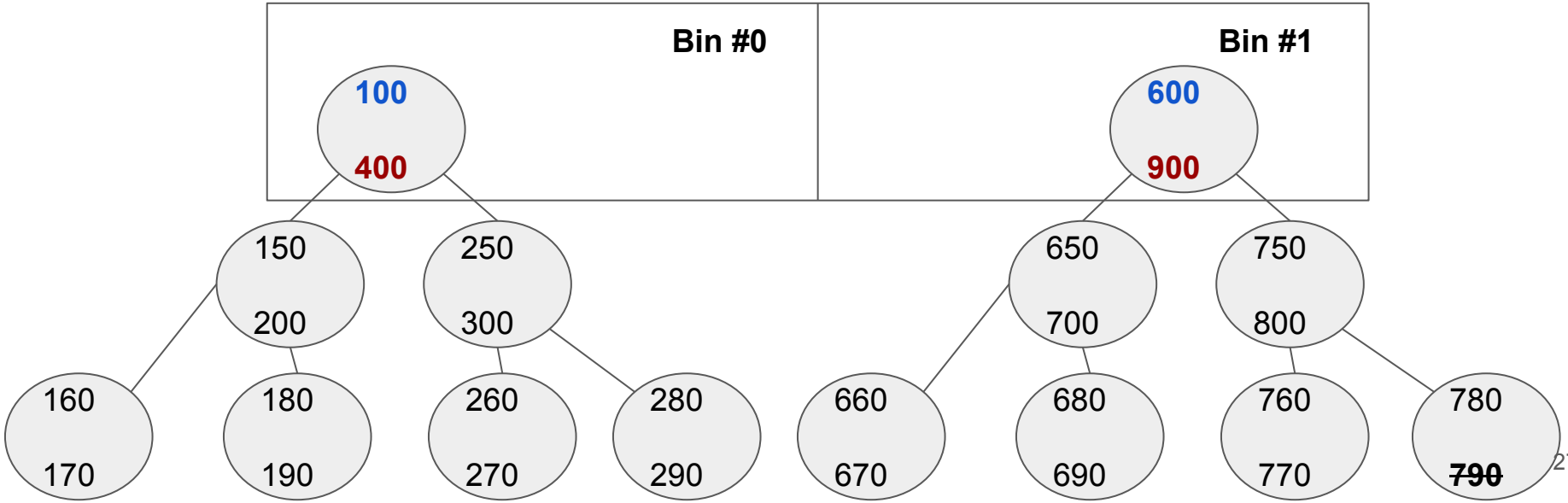
(~~28: 850~~)….



IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 28
keep = random () < s/29 #true
---
if keep : (~~28: 850~~)….
remove a random element from Bin #0 U Bin #1
insert the new element
else skip the element

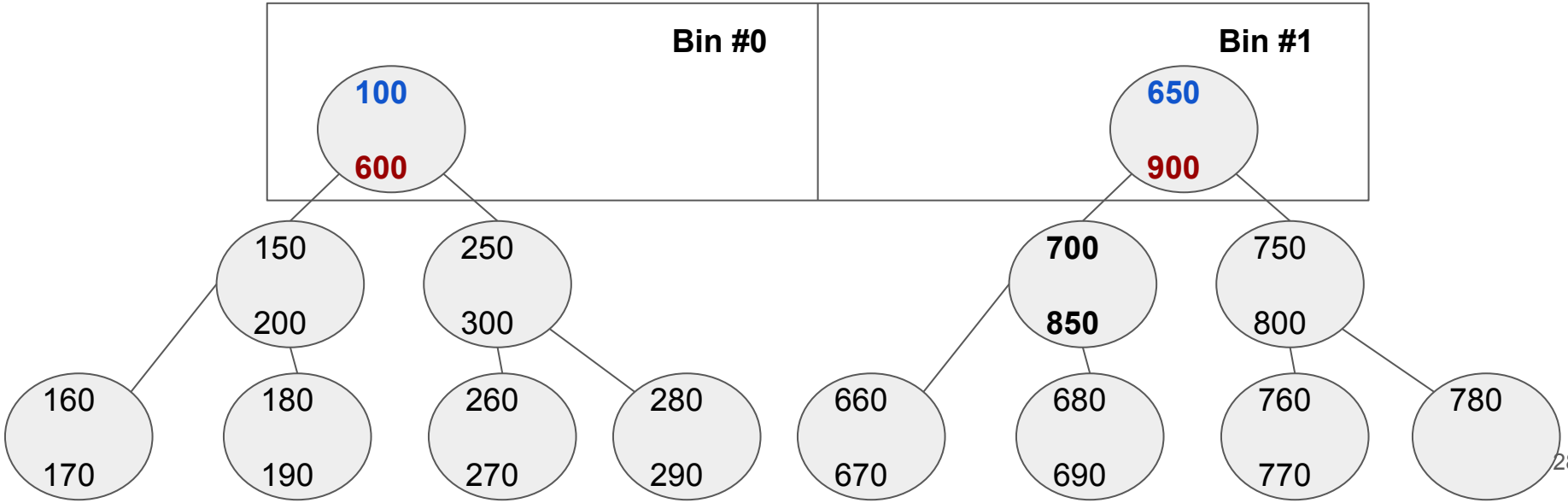


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 28
keep = random () < s/29 #true
---
if keep :
   remove a random element from Bin #0 U Bin #1
   insert the new element
else skip the element

(~~28: 850~~)….

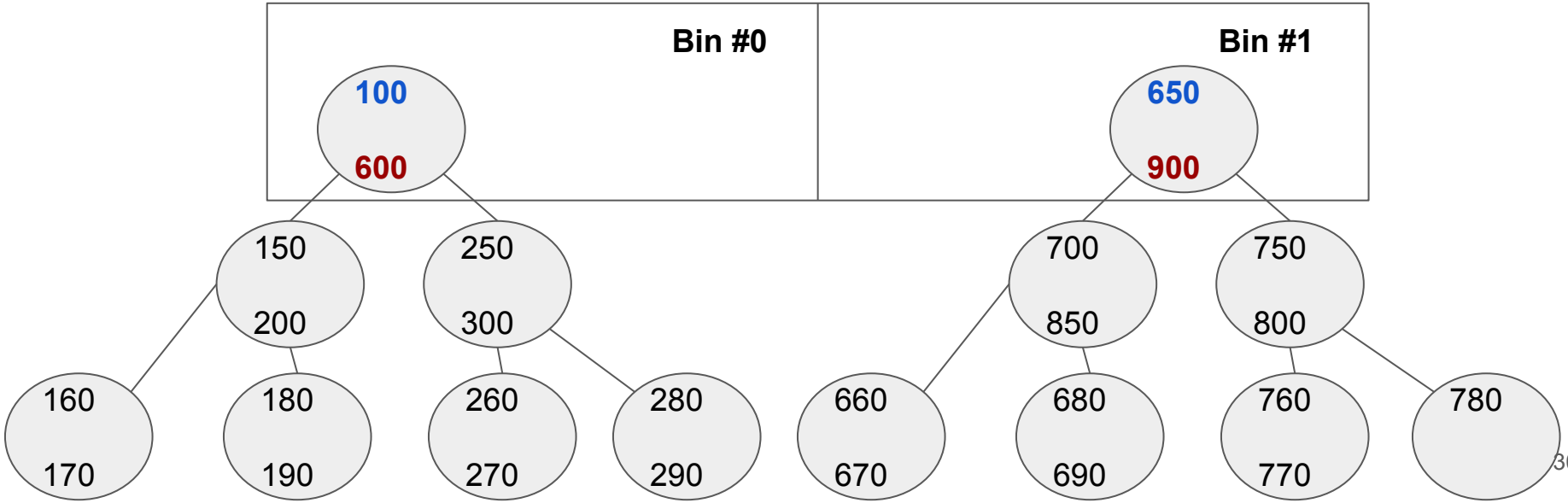


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 999
keep = random () < s/1000
---
if keep :
  remove a random element from Bin #0 U Bin #1
  insert the new element
else skip the element

(**999**: 890)….

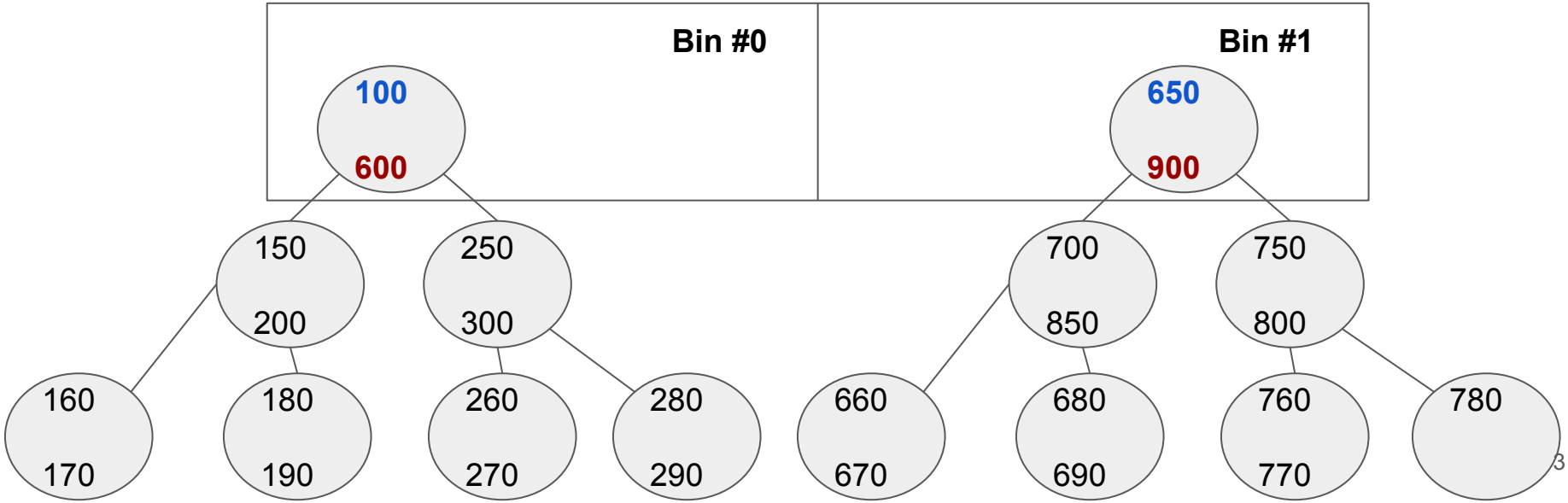


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 999
keep = random () < s/1000 # false
---
if keep :
   remove a random element from Bin #0 U Bin #1
   insert the new element
else skip the element

(**999**: 890)….

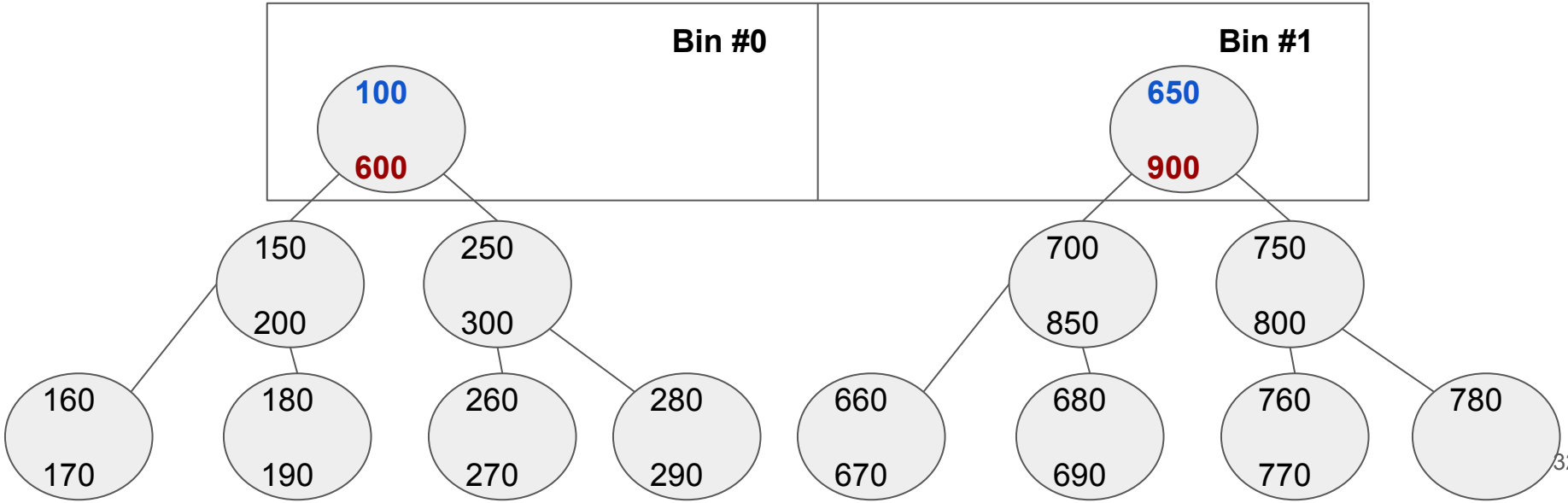


IDA example

size of the reservoir : 28
number of bins: 2

iteration i = 999
keep = random () < s/1000
---
if keep :
  remove a random element from Bin #0 U Bin #1
  insert the new element
else skip the element

(**999**: 890)….



IDA: performance

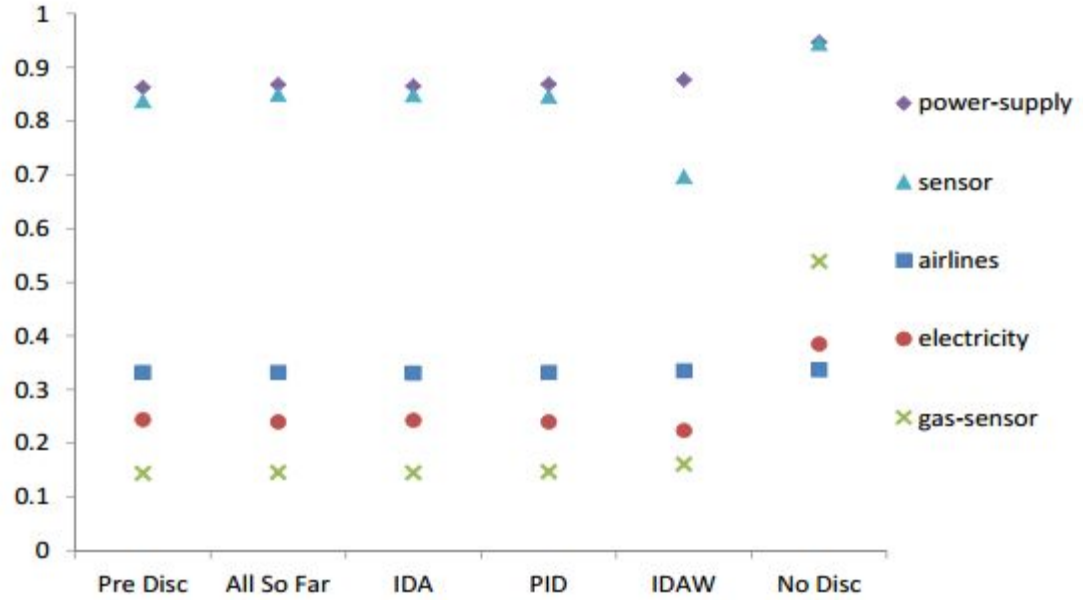
- constant time access for values of a bin (vector, heap)
- insertion/deletion in a bin O(log(s/m)) (heap)
- replacement in the vector of bins
  may require the update of m bins, O(m log(s/m))

- But, the complexity decreases over time because updates occur less often (reservoir sampling)

IDAW: performance

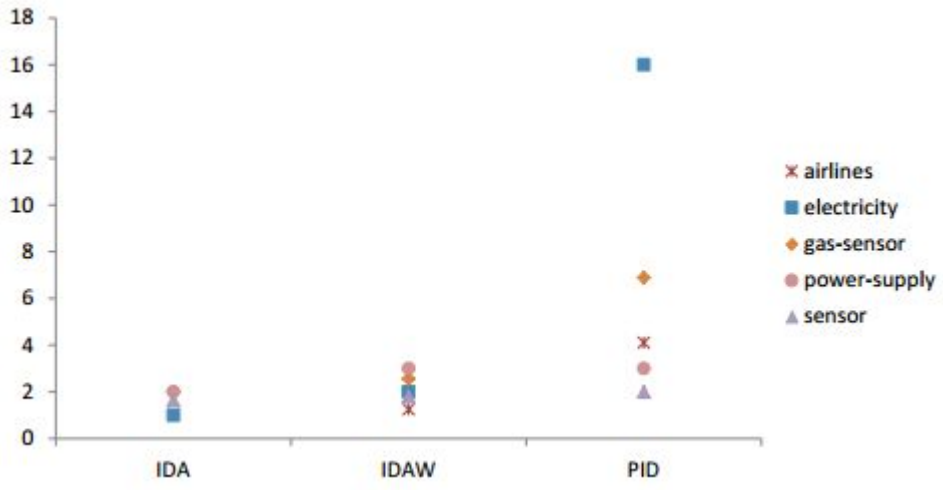
- we replace the oldest value (no randomness)
- a sliding window is used instead of a reservoir
- balanced binary trees for the bins
- the complexity for updates/accesses is O(log(s/m)) **every time**

Evaluation: with other approaches, LRSGD



Errors for each approach on each data stream

Evaluation: execution time, LRSGD



execution time for each approach, for each data set, expressed in terms of mutiples of the execution time taken without discretization

Conclusion

- IDA/IDAW maintain an equal frequency discretization
- Efficient computation (IDA) especially after a number of iterations thanks to the use of random sampling
- IDAW track more closely the distribution of the data
- discretization of the data can reduce the error for LRSGD with little cost on execution time