Jämförelsesortering och räknesortering
Weiss, avsnitt 7.8–7.9

Peter Ljunglöf

DAT036, Datastrukturer

10 dec 2012

Hur många jämförelser krävs?

Antag att vi ska sortera talen 1–n:

- Talen 1–n har $n!$ möjliga permutationer
- En algoritm som gör $f(n)$ jämförelser kan inte skilja på fler än $2^{f(n)}$ olika fall
- Alltså måste $2^{f(n)} \geq n!$, dvs $f(n) \geq \log_2(n!)$
- Men $\log_2(n!) = \Omega(n \log n)$ (enligt beviset till teorem 7.7 i avsnitt 7.8.1)
- Alltså måste sortering vara $\Omega(n \log n)$!

http://en.wikipedia.org/wiki/Comparison_sort#Number_of_comparisons_required_to_sort_a_list

Bucket sort [enligt Weiss]
Counting sort [enl. Wikipedia]

Antag att vi ska sortera en lista som innehåller n heltal, där alla ligger mellan 0 och $m-1$:

```
public int[] counting_sort(int list[], int max) {
    int[] counts = new int[max]
    for (int nr : list)
        counts[nr]++
    int position = 0
    for (int nr = 0; nr < counts.length; n++)
        for (int i = 0; i < counts[nr]; i++) {
            list[position] = nr
            position++
        }
}
```

Bucket sort [enligt Weiss]
Counting sort [enl. Wikipedia]

Antag att vi ska sortera en lista som innehåller n heltal, där alla ligger mellan 0 och $m-1$:

```
public int[] counting_sort(int list[], int max) {
    int[] counts = new int[max]
    for (int nr : list)
        counts[nr]++
    int position = 0
    for (int nr = 0; nr < counts.length; n++)
        for (int i = 0; i < counts[nr]; i++) {
            list[position] = nr
            position++
        }
    }
```

O(n), eftersom |list| = n

O(m), eftersom |counts| = m

O(n), eftersom $\sum$counts = n

Analys av counting sort

Den första loopen läser varje element i listan

- alltså O(n)

Den andra loopen går igenom alla möjliga värden

- alltså O(m)

Men den uppdaterar också varje position i listan

- alltså O(n)

Summa summarum, komplexiteten blir alltså

- O(n) + (m) = O(n + m)

Hur går det ihop med $\Omega(n \log n)$???

$O(n \log n)$ vs $O(n + m)$

Counting sort gör inga *jämförelser*!

- alltså gäller inte $\Omega(n \log n)$-argumentet
- en jämförelse är binär
  - större/mindre: 2 olika möjligheter
- counting sort använder mer information
  - 0, 1, …, m–1: m olika möjligheter

Bucket sort [enligt Wikipedia]

Detta är en generalisering av counting sort som funkar för alla objekt, och för godtyckligt många.

Från Wikipedia:

- Set up an array of initially empty "buckets."
- *Scatter*: Go over the original array, putting each object in its bucket.
  - (detta kräver någon form av partitionering)
- Sort each non-empty bucket.
- *Gather*: Visit the buckets in order and put all elements back into the original array.
  - (detta kräver att partitioneringen är ordnad)

Bucket sort för strängar

cat *boar* *cradle* *dear* *bear* *cart* *boat*

- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

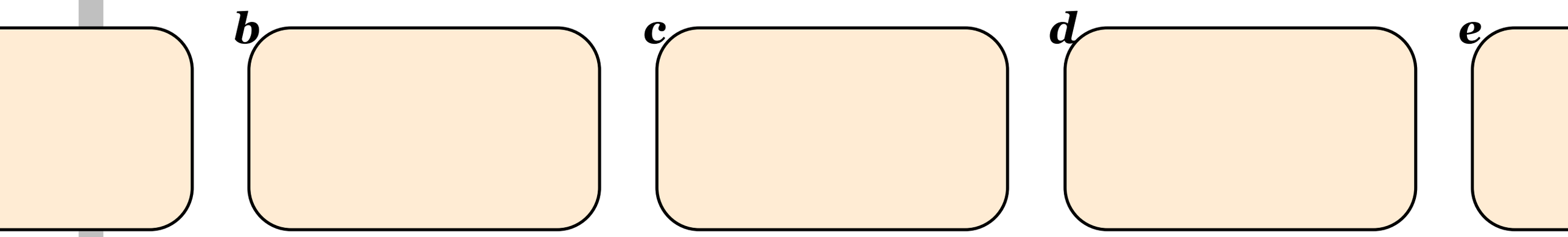
Bucket sort för strängar

cat *boar* *cradle* *dear* *bear* *cart* *boat*

- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar

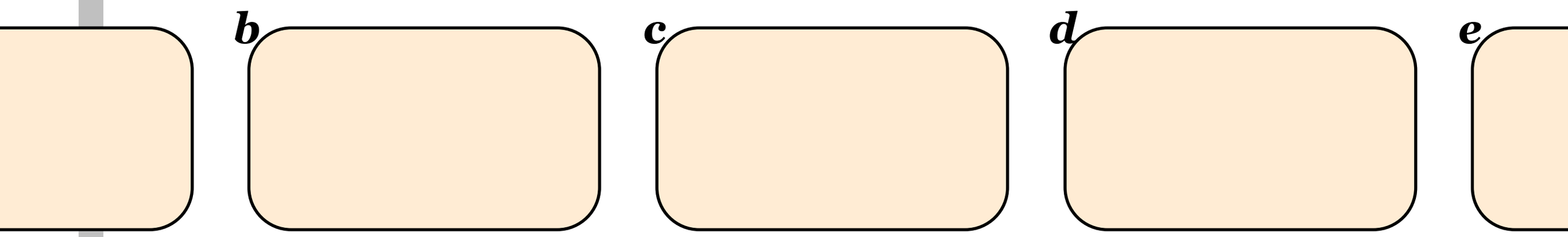
cat *boar* *cradle* *dear* *bear* *cart* *boat*



- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar

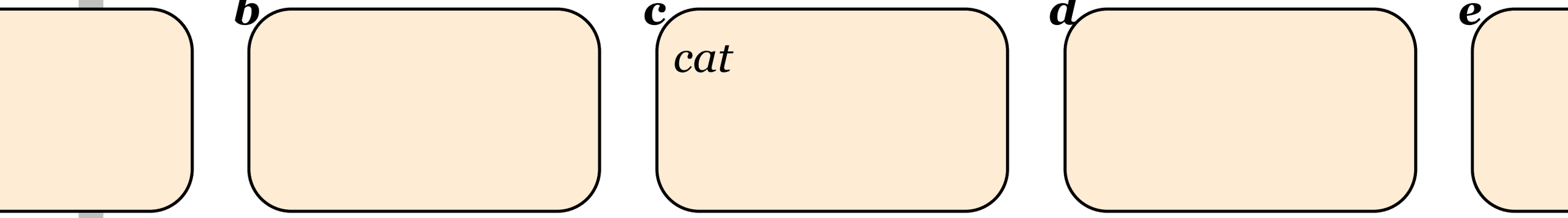
cat *boar* *cradle* *dear* *bear* *cart* *boat*



- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar

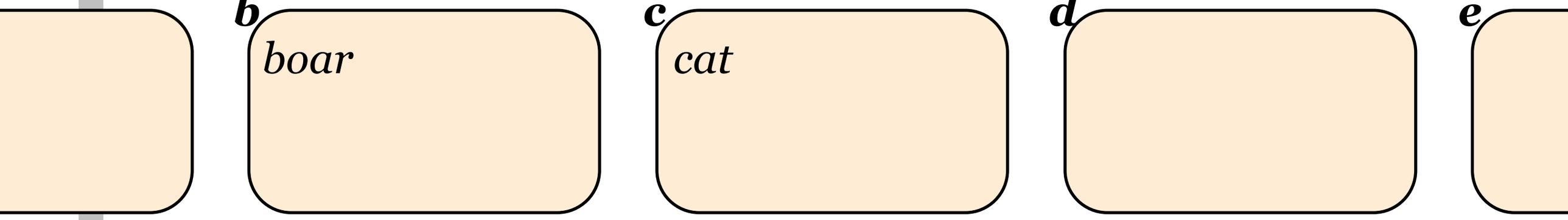
boar *cradle* *dear* *bear* *cart* *boat*



- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar

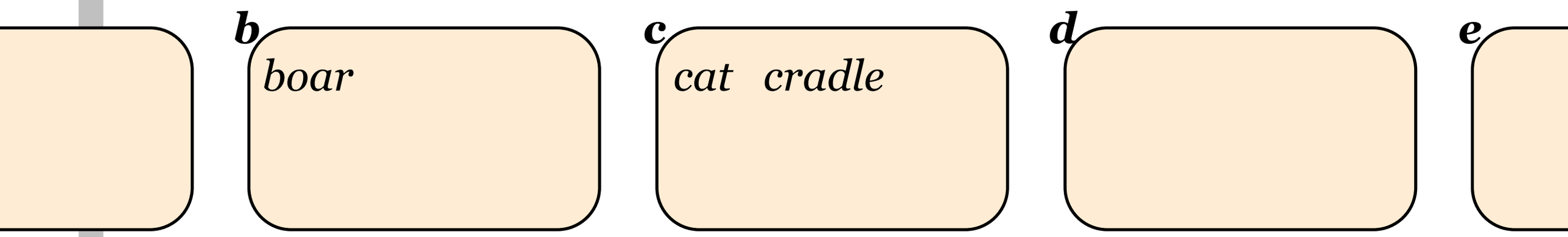
cradle *dear* *bear* *cart* *boat*



- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar

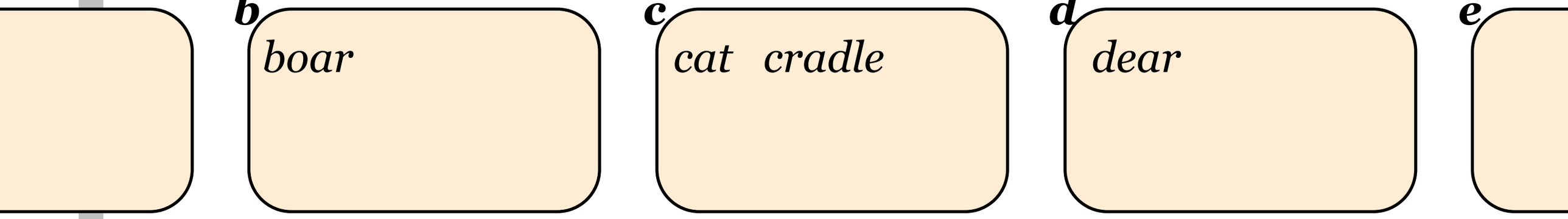
dear *bear* *cart* *boat*



- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

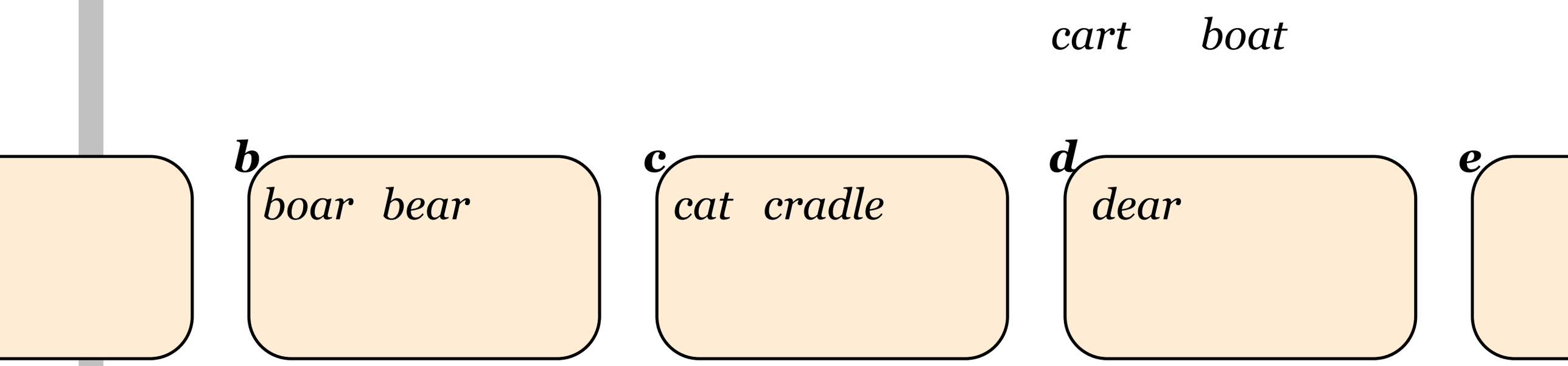
Bucket sort för strängar

bear *cart* *boat*



- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

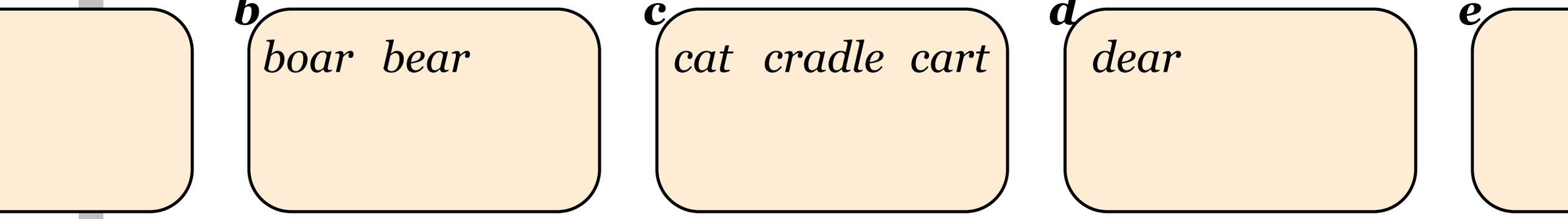
Bucket sort för strängar



- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

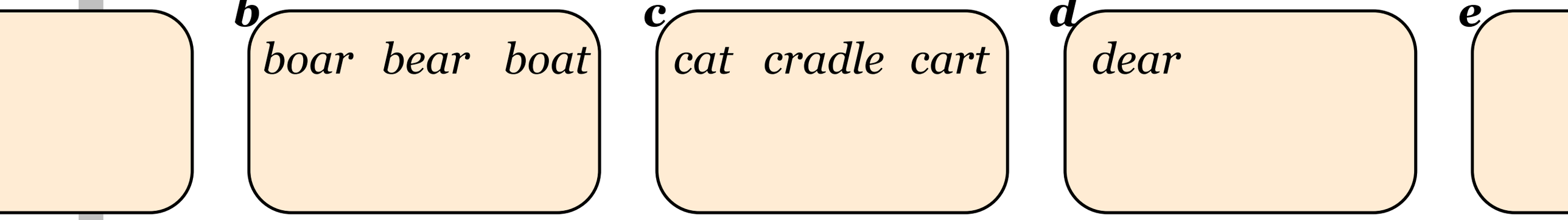
Bucket sort för strängar

boat



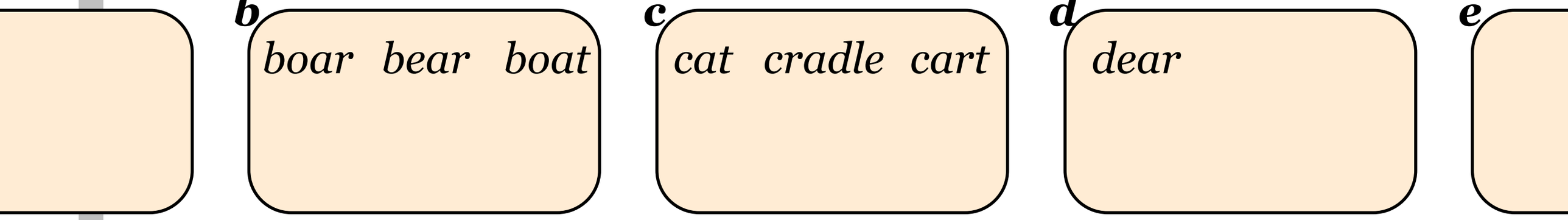
- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar



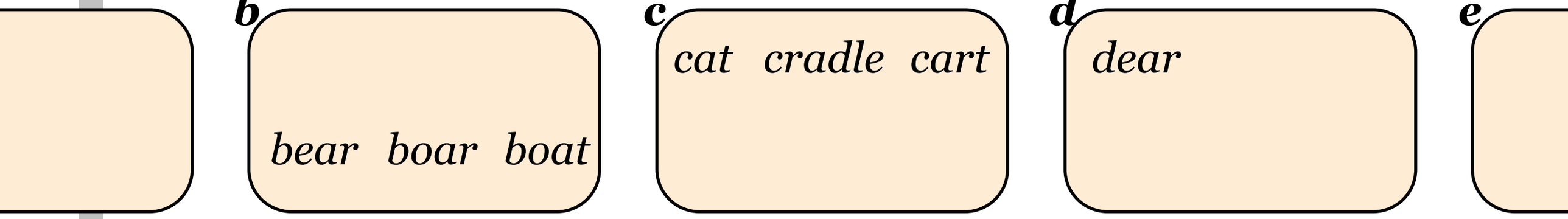
- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar



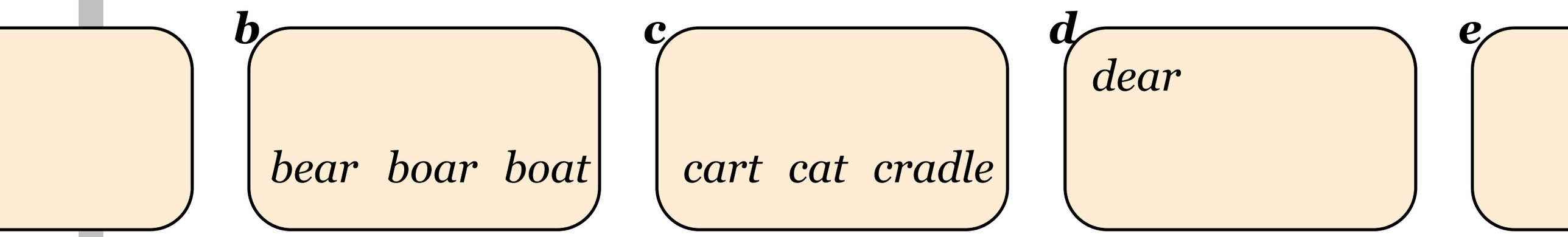
- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar



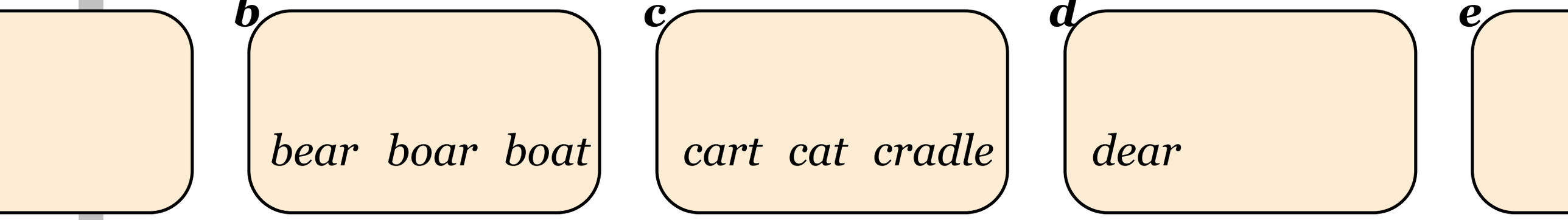
- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar



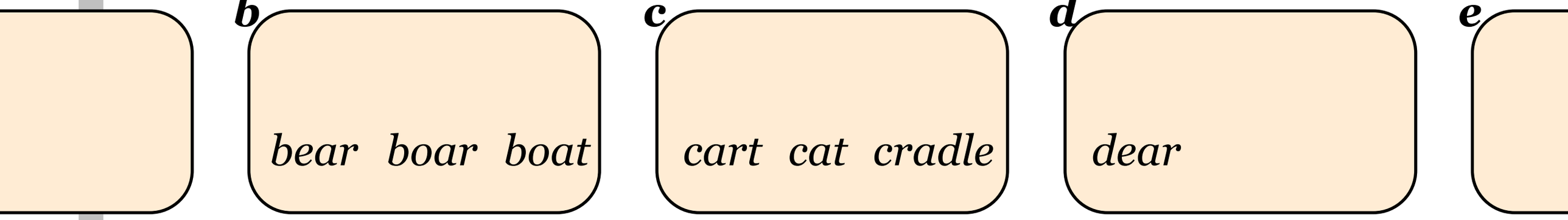
- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar



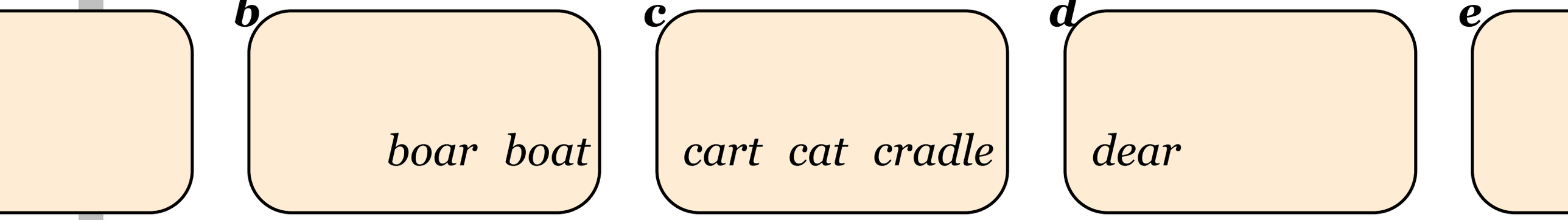
- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar



- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

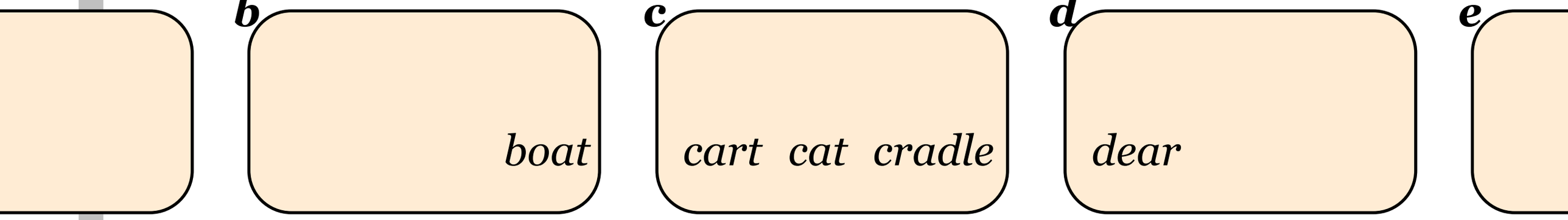
Bucket sort för strängar



bear

- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

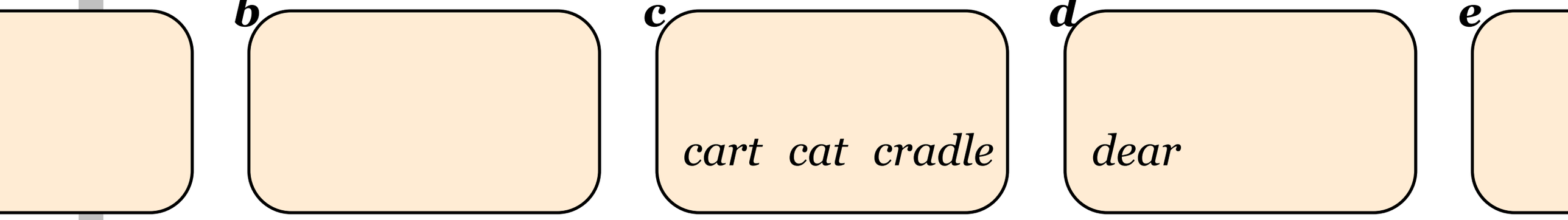
Bucket sort för strängar



bear boar

- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

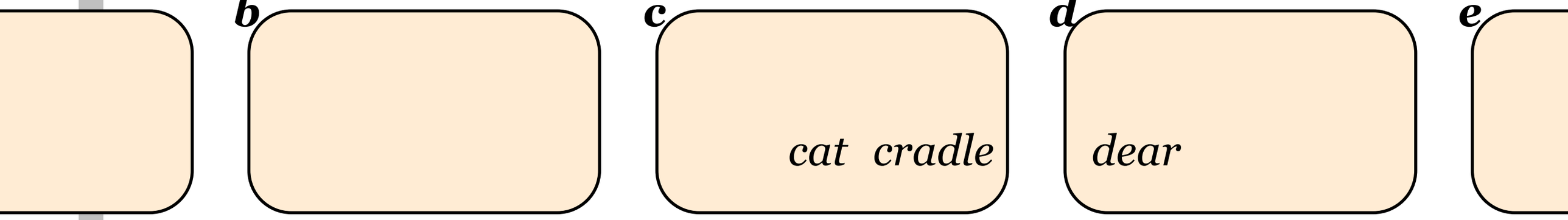
Bucket sort för strängar



bear boar boat

- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

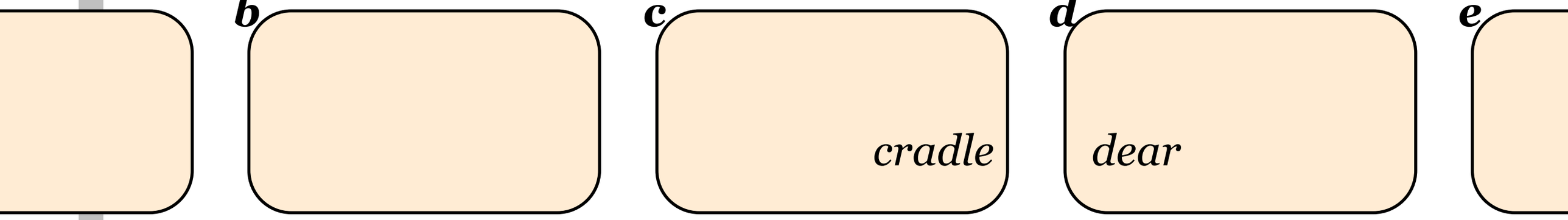
Bucket sort för strängar



bear boar boat cart

- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

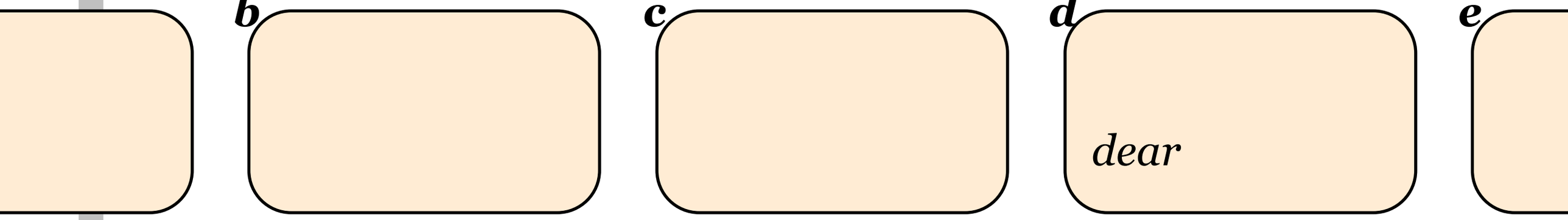
Bucket sort för strängar



bear *boar* *boat* *cart* *cat*

- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

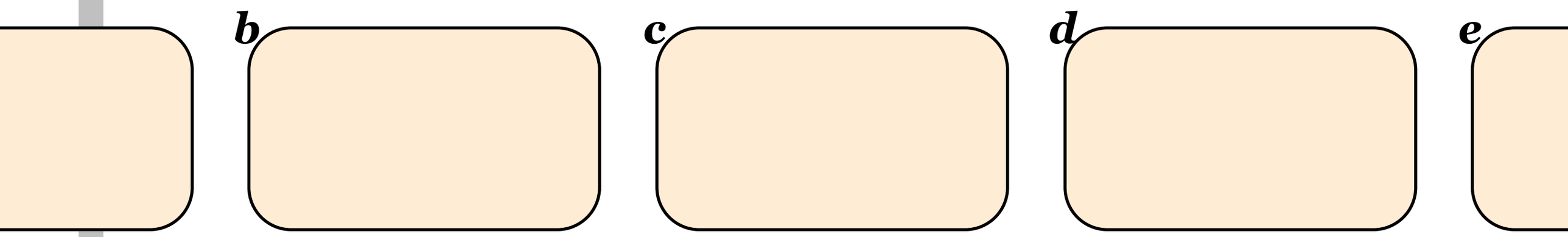
Bucket sort för strängar



bear *boar* *boat* *cart* *cat* *cradle*

- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

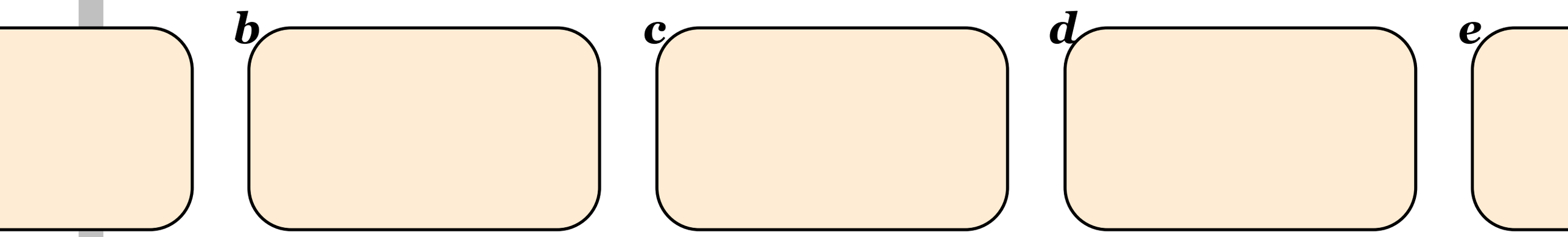
Bucket sort för strängar



bear *boar* *boat* *cart* *cat* *cradle* *dear*

- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Bucket sort för strängar



bear *boar* *boat* *cart* *cat* *cradle* *dear*

- Set up an array of initially empty "buckets."
- Put each object in its bucket.
- Sort each non-empty bucket.
- Visit the buckets in order

Varianter av bucket sort

Radix sort:

- en variant av bucket sort för heltal
- funkar på samma sätt som sträng-exemplet
- vid sorteringssteg *k* grupperar vi talen (strängarna) efter siffra (tecken) nummer *k*

Quicksort:

- kan ses som bucket sort med två buckets – ”*mindre-än-pivot*” och ”*större-än-pivot*”