[301] Dictionary Nesting

Tyler Caraza-Harter

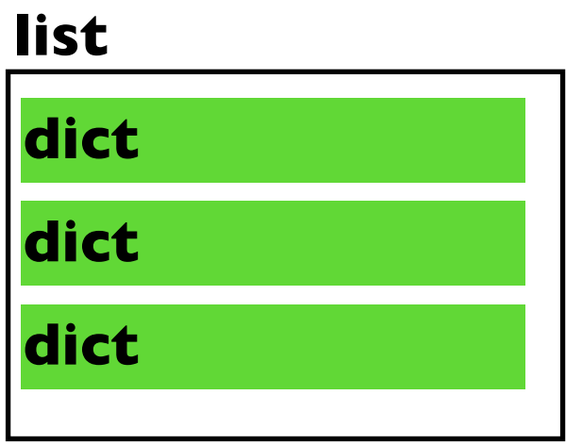
Learning Objectives Today

More dictionary operations

- len, in, for loop
- d.keys(), d.values()
- defaults for get and pop

Syntax for nesting (dicts inside dicts, etc)

- indexing/lookup
- step-by-step resolution



Understand common use cases for nesting

- binning/bucketing (`list` in `dict`)
- a more convenient table representation (`dict` in `list`)
- a more convenient table representation (`dict` in `dict`)

one of the most common
data analysis tasks

Today's Outline

Dictionary Ops

Binning (dict of list)

Table Representation (list of dict)

Table Representation (dict of dict)

Creation of Empty Dict

Non-empty dict:

```
d = {“a”: “alpha”, “b”: “beta”}
```

Empty dict (way 1):

```
d = {}
```

Empty dict (way 2):

```
d = dict()
```

similar for lists: `L = list()`

similar for sets: `s = set()`

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}

print(len(num_words))

print(1 in num_words)

print(“one” in num_words)

for x in num_words:
    print(x)
```

4

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}

print(len(num_words))

print(1 in num_words)

print(“one” in num_words)

for x in num_words:
    print(x)
```

4

?

?

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(len(num_words))
```

4

```
print(1 in num_words)
```

True

```
print(“one” in num_words)
```

False

(it is only checking keys, not vals)

```
for x in num_words:
    print(x)
```

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(len(num_words))
```

4

```
print(1 in num_words)
```

True

```
print(“one” in num_words)
```

False

(it is only checking keys, not vals)

```
for x in num_words:
    print(x)
```

?
?
?
?

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(len(num_words))
```

4

```
print(1 in num_words)
```

True

```
print(“one” in num_words)
```

False

(it is only checking keys, not vals)

```
for x in num_words:
    print(x)
```

2
1
0
3

(for iterates over keys, not vals)

(note there is no order here)

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(len(num_words))
```

4

```
print(1 in num_words)
```

True

```
print(“one” in num_words)
```

False

(it is only checking keys, not vals)

```
for x in num_words:
    print(x, num_words[x])
```

2 two
1 one
0 zero
3 three

you can iterate over values
by combining a **for loop** with **lookup**

Extracting keys and values

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(type(num_words.keys()))
```

<class 'dict_keys'>

```
print(type(num_words.values()))
```

<class 'dict_values'>

don’t worry about these new types, because we can force them to be lists

Extracting keys and values

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(type(num_words.keys()))
```

<class 'dict_keys'>

```
print(type(num_words.values()))
```

<class 'dict_values'>

```
print(list(num_words.keys()))
```

[3, 1, 2, 0]

```
print(list(num_words.values()))
```

[“one”, “two”, “zero”, “three”]

Defaults with get and pop

```
suffix = {1:“st”, 2:”nd”, 3:”rd”}
```

```
suffix.pop(0) # delete fails, because no key 0
```

```
suffix[4] # lookup fails because no key 4
```

Defaults with get and pop

```
suffix = {1:“st”, 2:”nd”, 3:”rd”}
```

```
suffix.pop(0) # delete fails, because no key 0
```

```
suffix[4] # lookup fails because no key 4
```

```
suffix.get(4, “th”) # returns “th” because no key 4
```

specify a default if
key cannot be found

Defaults with get and pop

```
suffix = {1:"st", 2:"nd", 3:"rd"}
```

specify a default if
key cannot be found

```
suffix.pop(0) # delete fails, because no key 0
```

```
suffix[4] # lookup fails because no key 4
```

```
suffix.get(4, "th") # returns "th" because no key 4
```

specify a default if
key cannot be found

Defaults with get and pop

```
suffix = {1:"st", 2:"nd", 3:"rd"}
```

specify a default if
key cannot be found

```
suffix.pop(0, "th") # returns "th" because no key 0
```

```
suffix[4] # lookup fails because no key 4
```

```
suffix.get(4, "th") # returns "th" because no key 4
```

specify a default if
key cannot be found

Defaults with get and pop

```
suffix = {1:“st”, 2:”nd”, 3:”rd”}


for num in range(6):
    print(str(num) + suffix.get(num, “th”))
```

```
0th
1st
2nd
3rd
4th
5th
```

Today's Outline

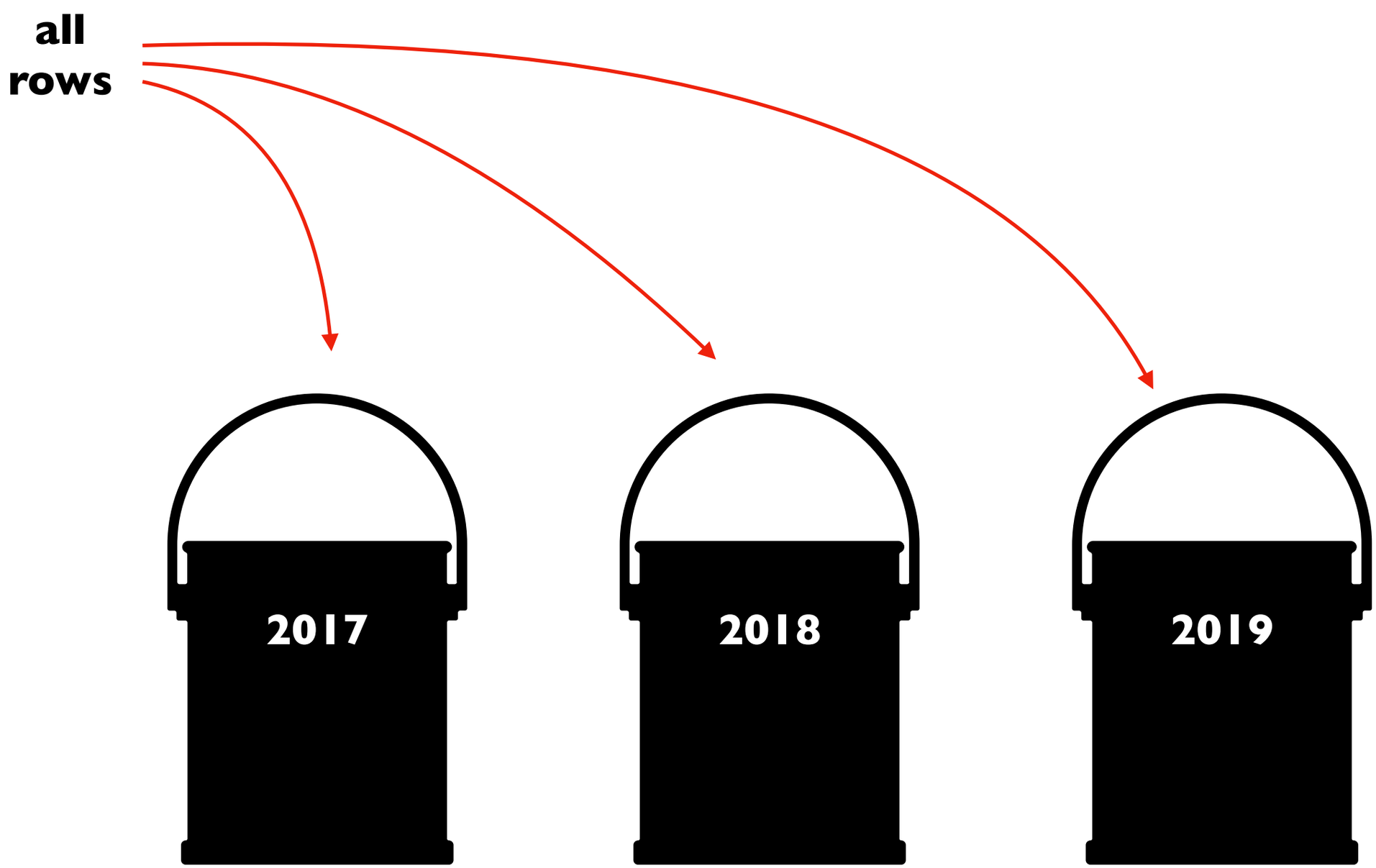
Dictionary Ops

Binning (dict of list)

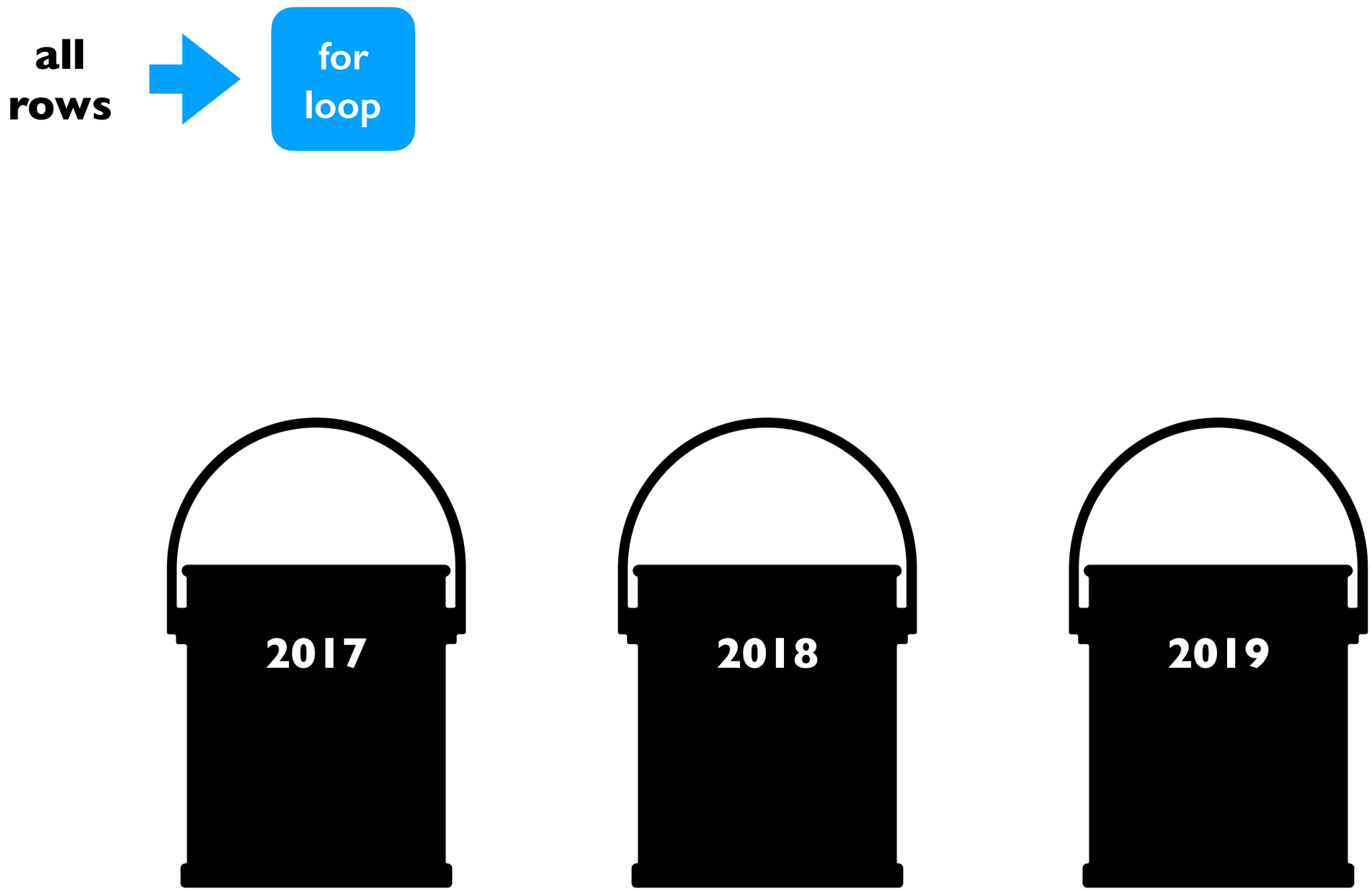
Table Representation (list of dict)

Table Representation (dict of dict)

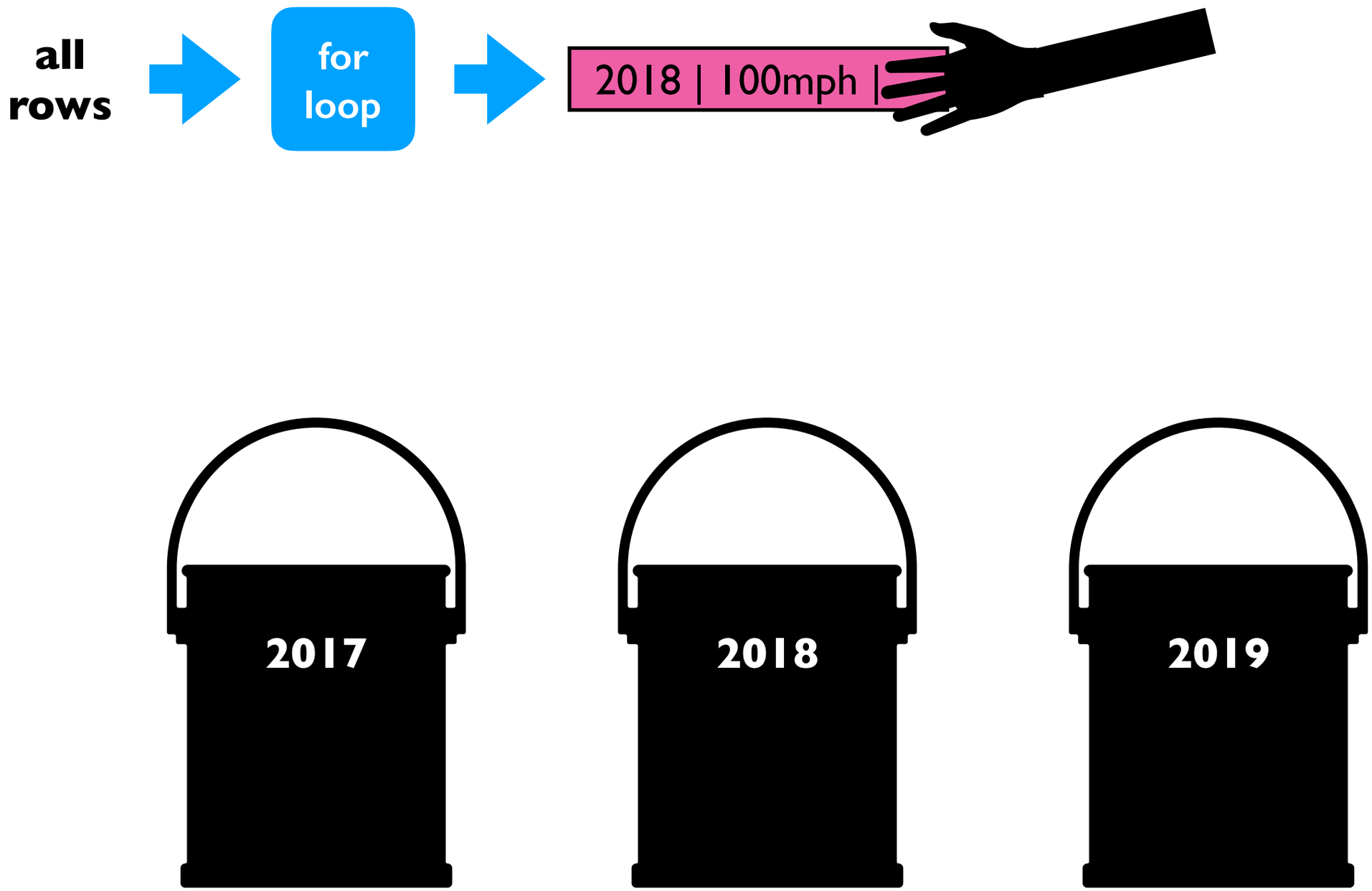
Bucketing/Binning



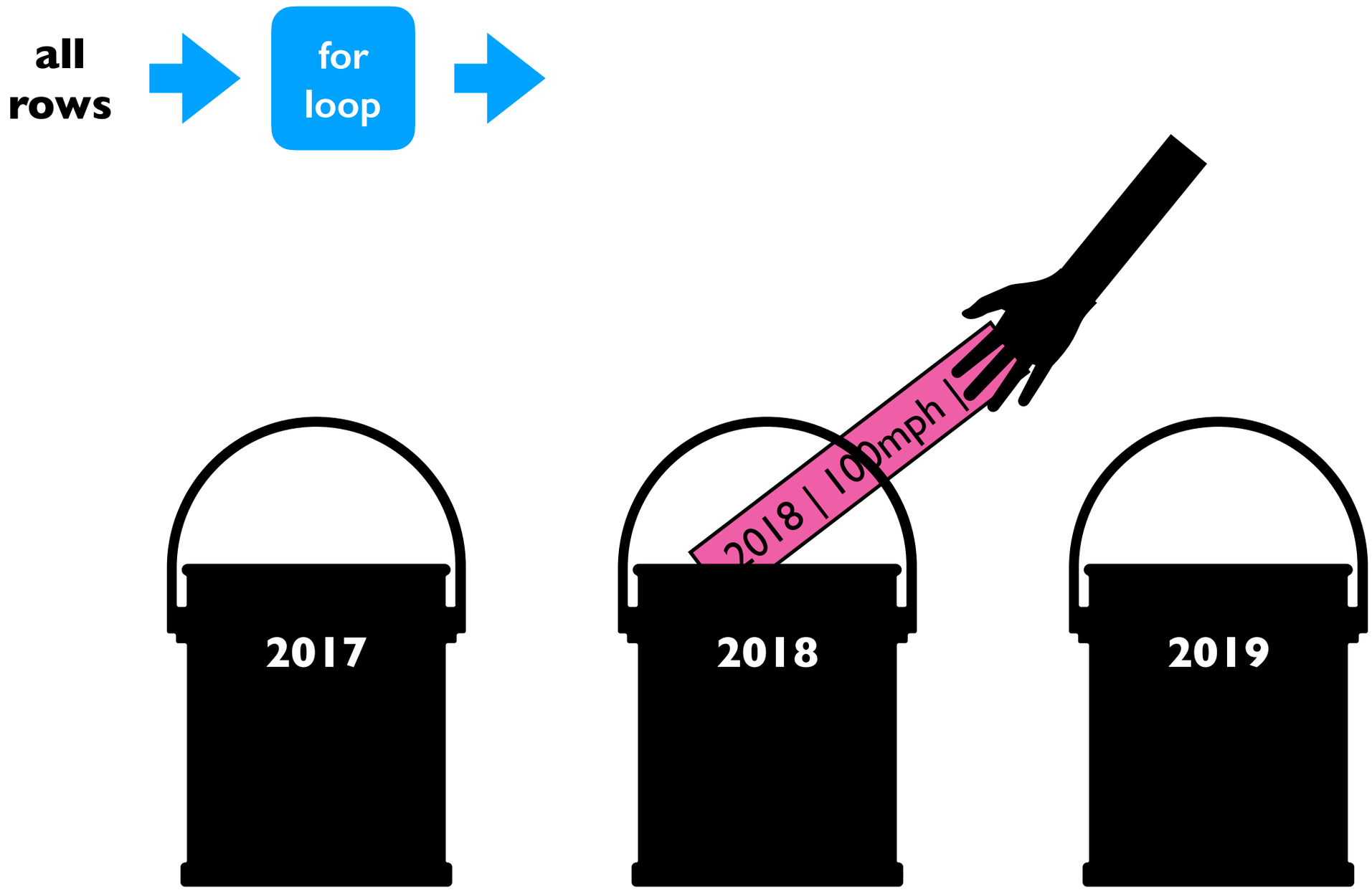
Bucketing/Binning



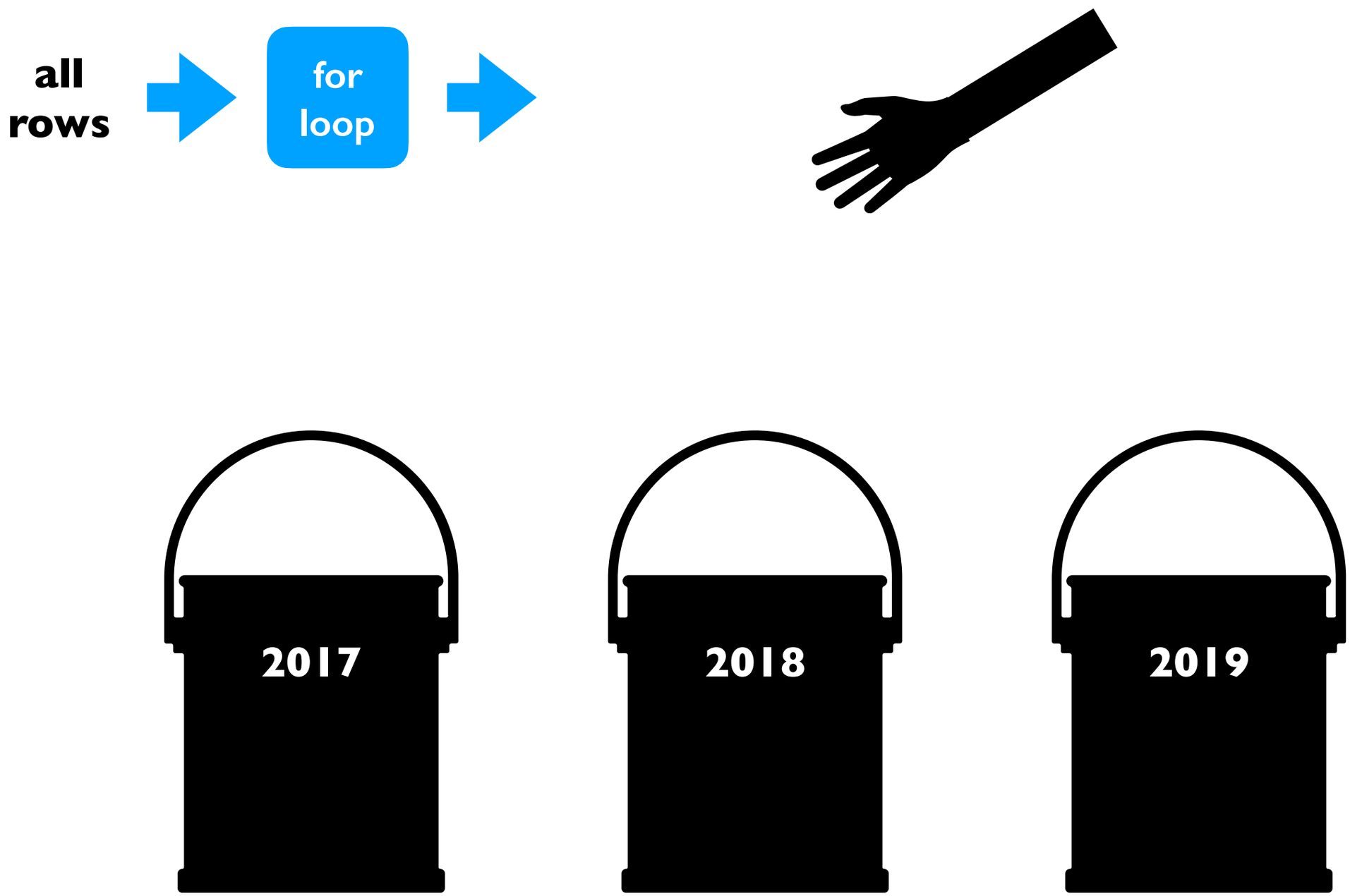
Bucketing/Binning



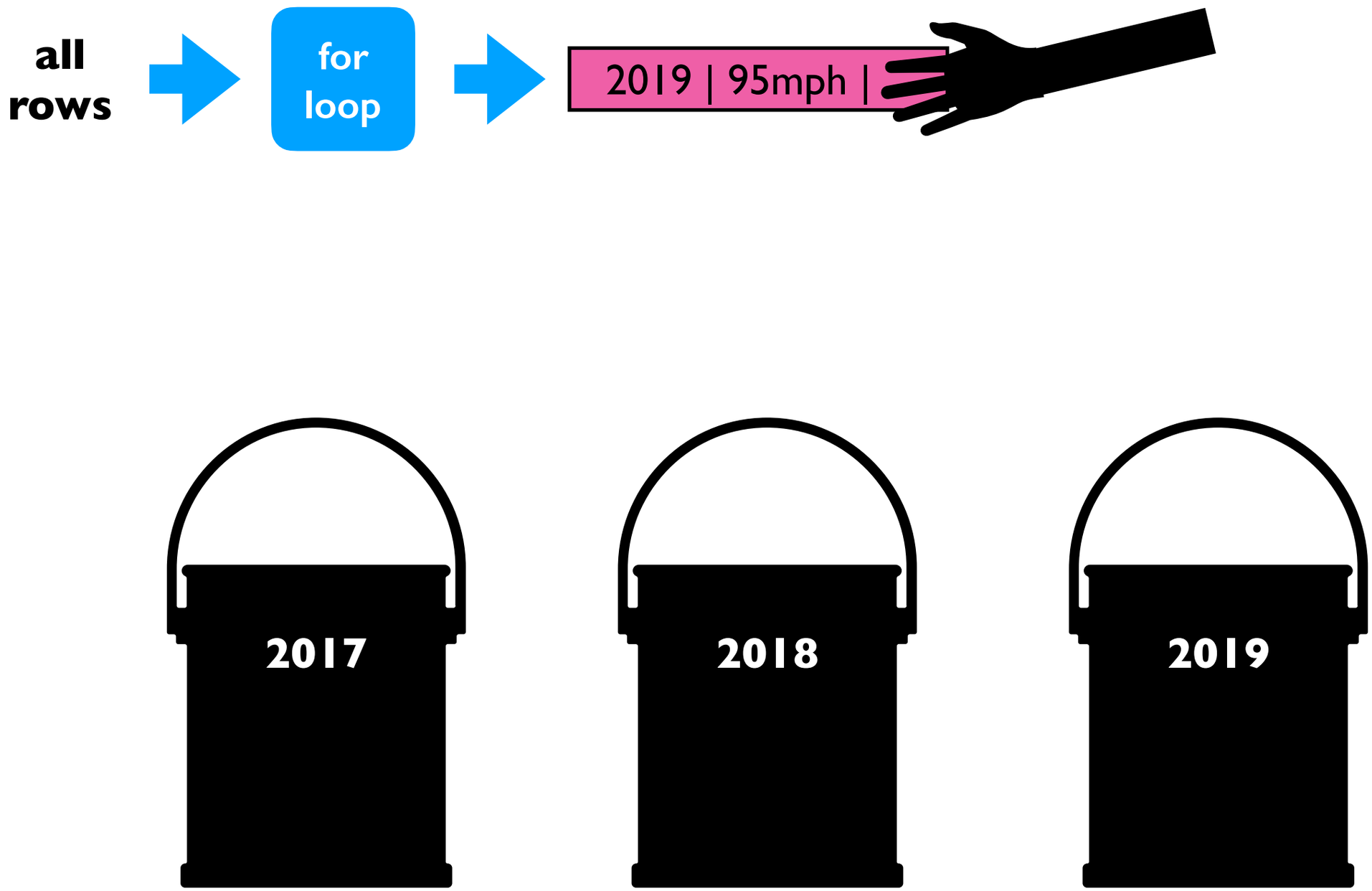
Bucketing/Binning



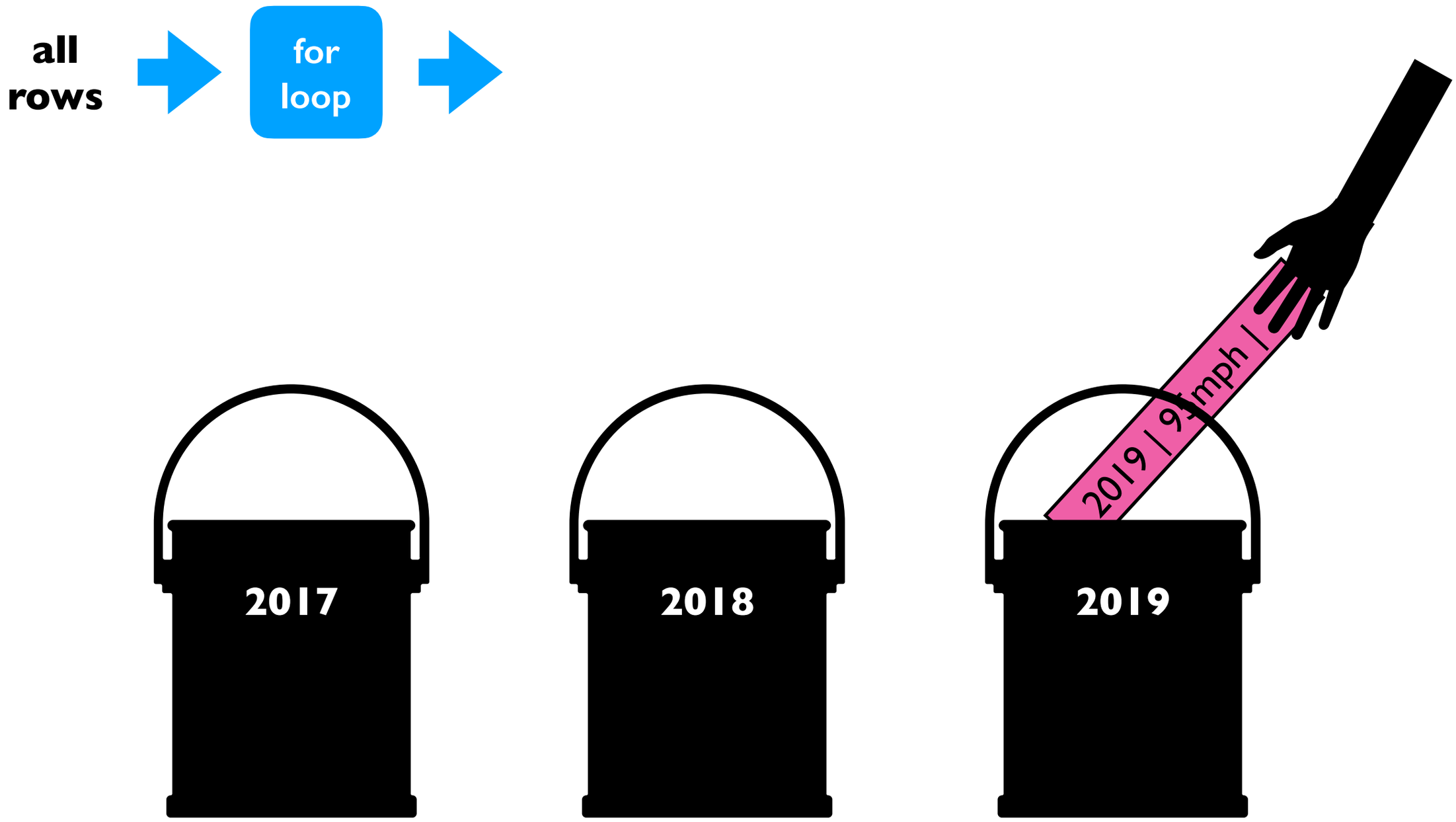
Bucketing/Binning



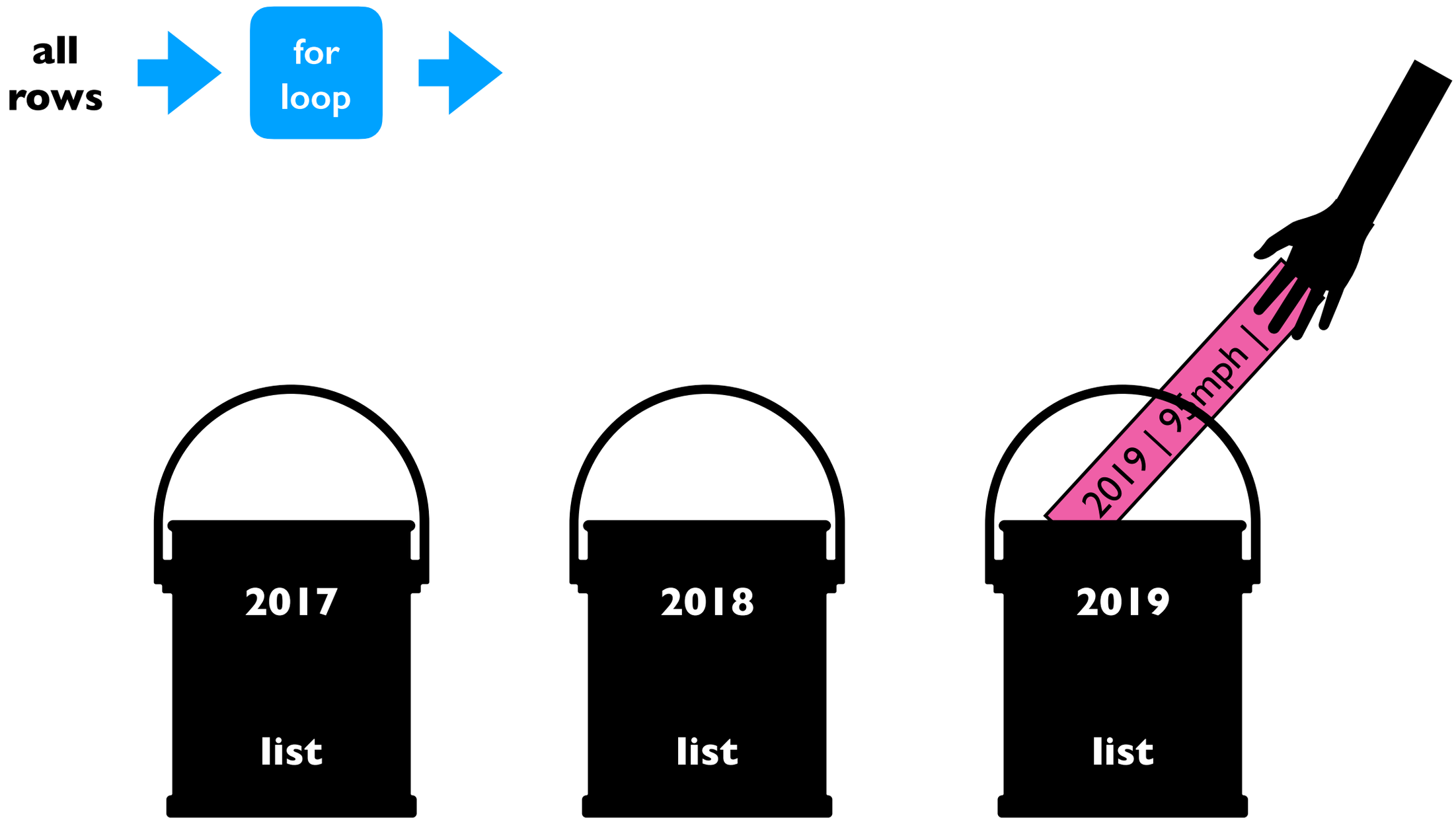
Bucketing/Binning



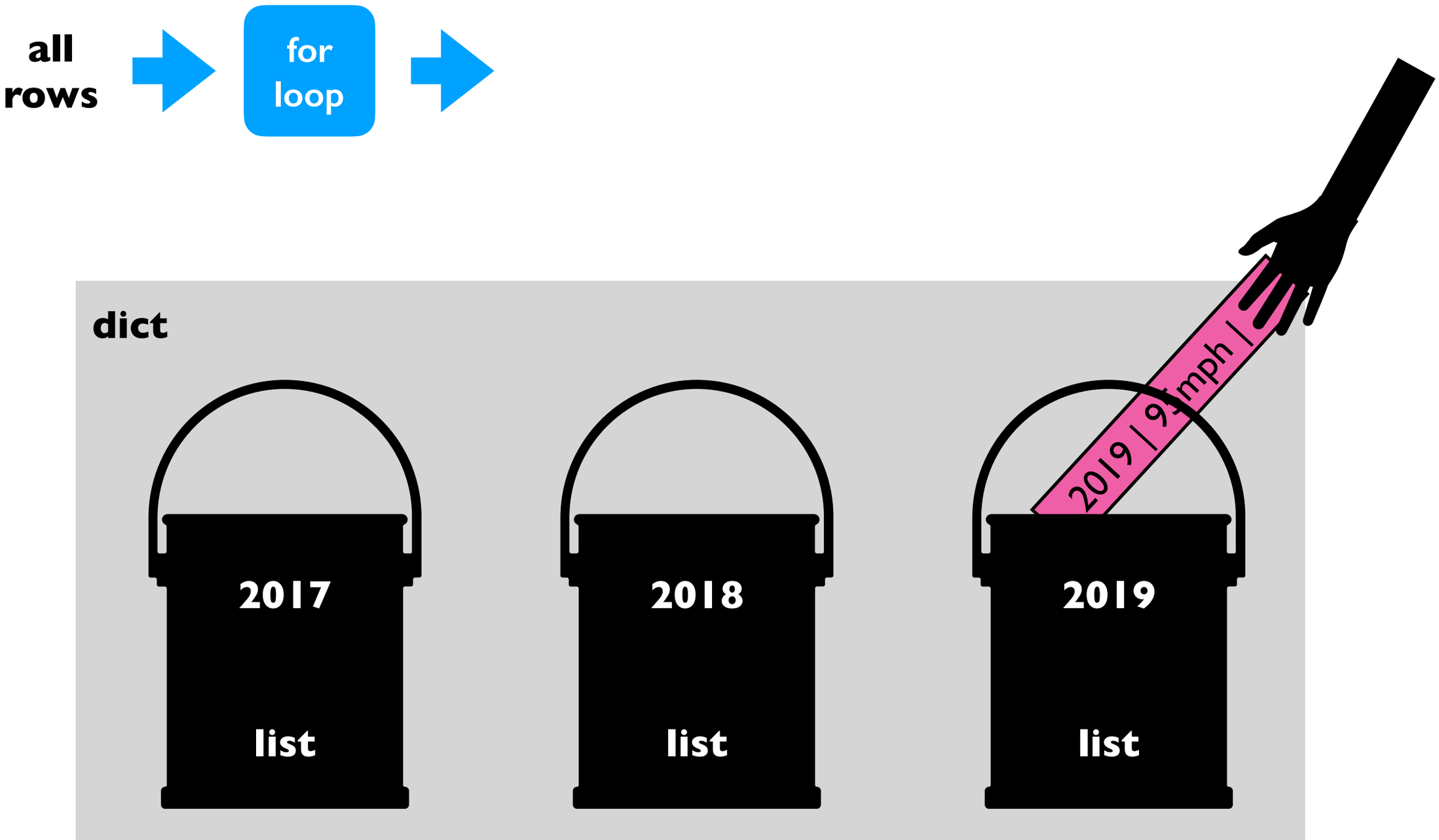
Bucketing/Binning



Bucketing/Binning



Bucketing/Binning



Bins with dicts and lists

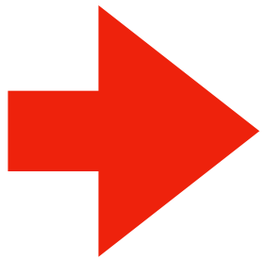
all data

```
rows = [
    [2014, “A”, 123],
    [2015, “B”, 120],
    [2015, “C”, 140],
    [2016, “D”, 100],
    [2015, “E”, 130],
    [2016, “F”, 200],
]
```

Bins with dicts and lists

all data

```
rows = [
    [2014, “A”, 123],
    [2015, “B”, 120],
    [2015, “C”, 140],
    [2016, “D”, 100],
    [2015, “E”, 130],
    [2016, “F”, 200],
]
```



```
bins = {
    2014: [
        [2014, “A”, 123],
    ],
    2015: [
        [2015, “B”, 120],
        [2015, “C”, 140],
        [2015, “E”, 130],
    ],
    2016: [
        [2016, “D”, 100],
        [2016, “F”, 200],
    ]
}
```

Bins with dicts and lists

all data

```
rows = [
    [2014, “A”, 123],
    [2015, “B”, 120],
    [2015, “C”, 140],
    [2016, “D”, 100],
    [2015, “E”, 130],
    [2016, “F”, 200],
]
```

```
bins = {
    2014: [
        [2014, “A”, 123],
    ],
    2015: [
        [2015, “B”, 120],
        [2015, “C”, 140],
        [2015, “E”, 130],
    ],
    2016: [
        [2016, “D”, 100],
        [2016, “F”, 200],
    ]
}
```

median 123

median 130

median 150

Demo 1: Median Tornado Speed per Year

Goal: print **median speed** of tornados for each year

Input:

- Tornado CSV

Output:

- Median within each year

Example:

prompt> **python tornados.py**
...
2015: 130
2016: 123
2017: 90

Today's Outline

Dictionary Ops

Binning (dict of list)

Table Representation (list of dict)

Table Representation (dict of dict)

Table Representation (list of dict)

name	x	y
Alice	30	20
Bob	5	11
Cindy	-2	50

list of list representation

list of dict representation

```
header = [“name”, “x”, “y”]
rows = [
    [“Alice”, 30, 20],
    [“Bob”,   5,  11],
    [“Cindy”, -2, 50],
]
```

```
[
  {“name”:“Alice”, “x”:30, “y”:20”},
  {“name”:“Bob”,   “x”:5,  “y”:11”},
  {“name”:“Cindy”, “x”:-2, “y”:50”},
]
```

Table Representation (list of dict)

name	x	y
Alice	30	20
Bob	5	11
Cindy	-2	50

list of list representation

```
header = [“name”, “x”, “y”]
rows = [
    [“Alice”, 30, 20],
    [“Bob”,   5,  11],
2 → [“Cindy”, -2, 50],
]
```

2

rows[2][header.index(“y”)]

list of dict representation

```
[
  {“name”:“Alice”, “x”:30, “y”:20”},
  {“name”:“Bob”,   “x”:5,  “y”:11”},
2 → {“name”:“Cindy”, “x”:-2, “y”:50”},
]
```

"y"

rows[2][“y”]

Demo 2: Table Transform

Goal: create function that transforms list of lists table to a list of dicts table

Input:

- List of lists (from a CSV)

Output:

- List of dicts

Example:

```
>>> header = ["x","y"]
>>> rows = [[1,2], [3,4]]
>>> transform(header, rows)
[{"x":1,"y":2}, {"x":3,"y":4}]
```

Today's Outline

Dictionary Ops

Binning (dict of list)

Table Representation (list of dict)

Table Representation (dict of dict)

Table Representation (dict of dict)

name	x	y
Alice	30	20
Bob	5	11
Cindy	-2	50

list of list representation

list of dict representation

```
header = [“name”, “x”, “y”]
rows = [
    [“Alice”, 30, 20],
    [“Bob”,   5,  11],
2 → [“Cindy”, -2, 50],
]
                   ↑
                   2
```

```
{
  “Alice”: {“x”:30, “y”:20”},
  “Bob”:   {“x”:5,  “y”:11”},
"Cindy" → “Cindy”: {“x”:-2, “y”:50”},
}
                          ↑
                          "y"
```

rows[2][header.index(“y”)]

rows["Cindy"][“y”]

Demo 3: Table Transform (v2)

Goal: create function that transforms list of lists table to a dict of dicts table

Input:

- List of lists (from a CSV)

Output:

- List of dicts

Example:

```
>>> header = [“id”,“x”,”y”]
>>> rows = [["A",1,2], ["B",3,4]]
>>> transform(header, rows, key="id")
["A": {“x”:1, “y”:2}, "B": {“x”:3, “y”:4}]
```