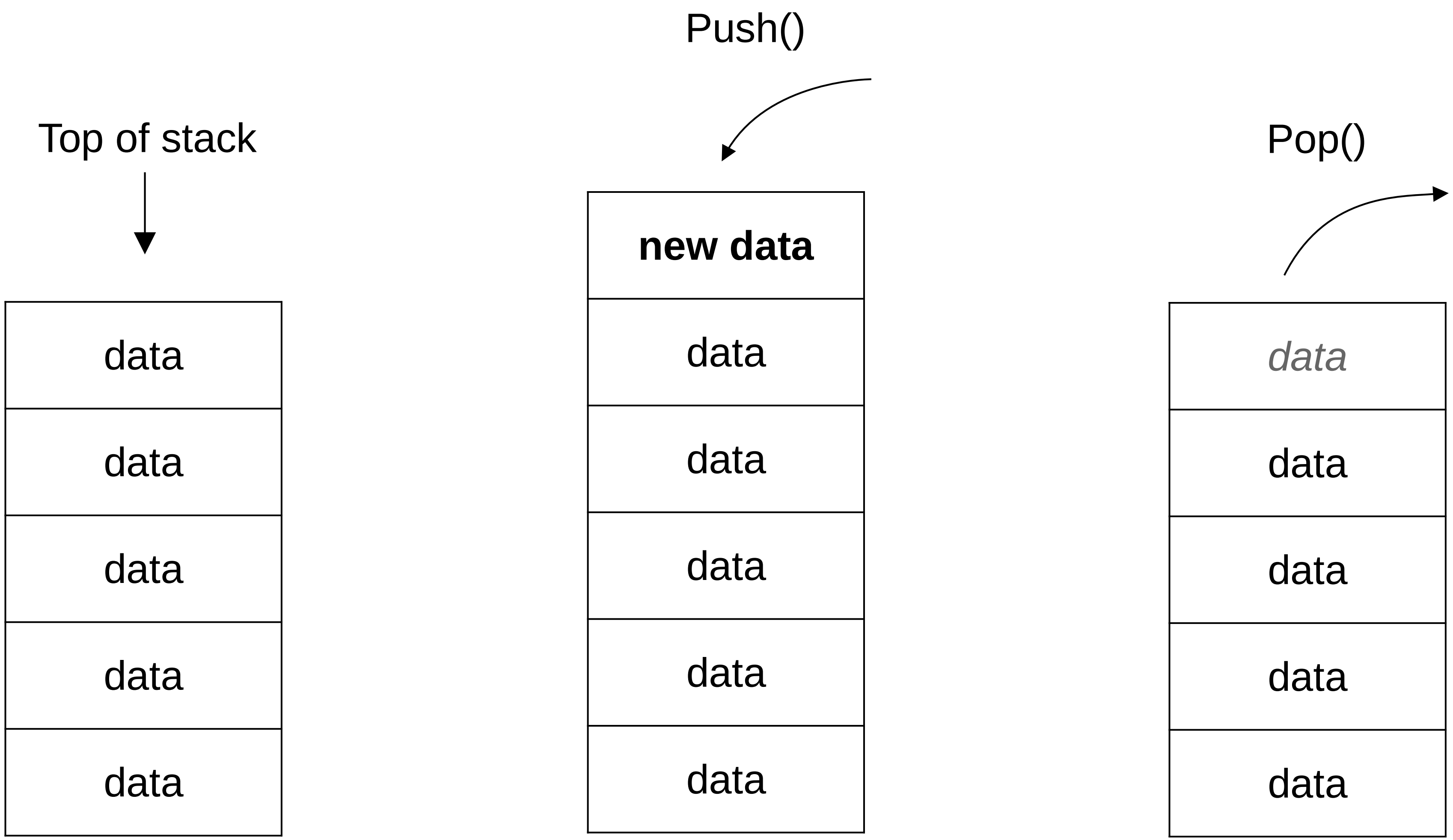
Stack, queue and hashtable

Simeon Monov

Stack



Top of stack

data

data

data

data

data

Push()

new data

data

data

data

data

data

Pop()

data

data

data

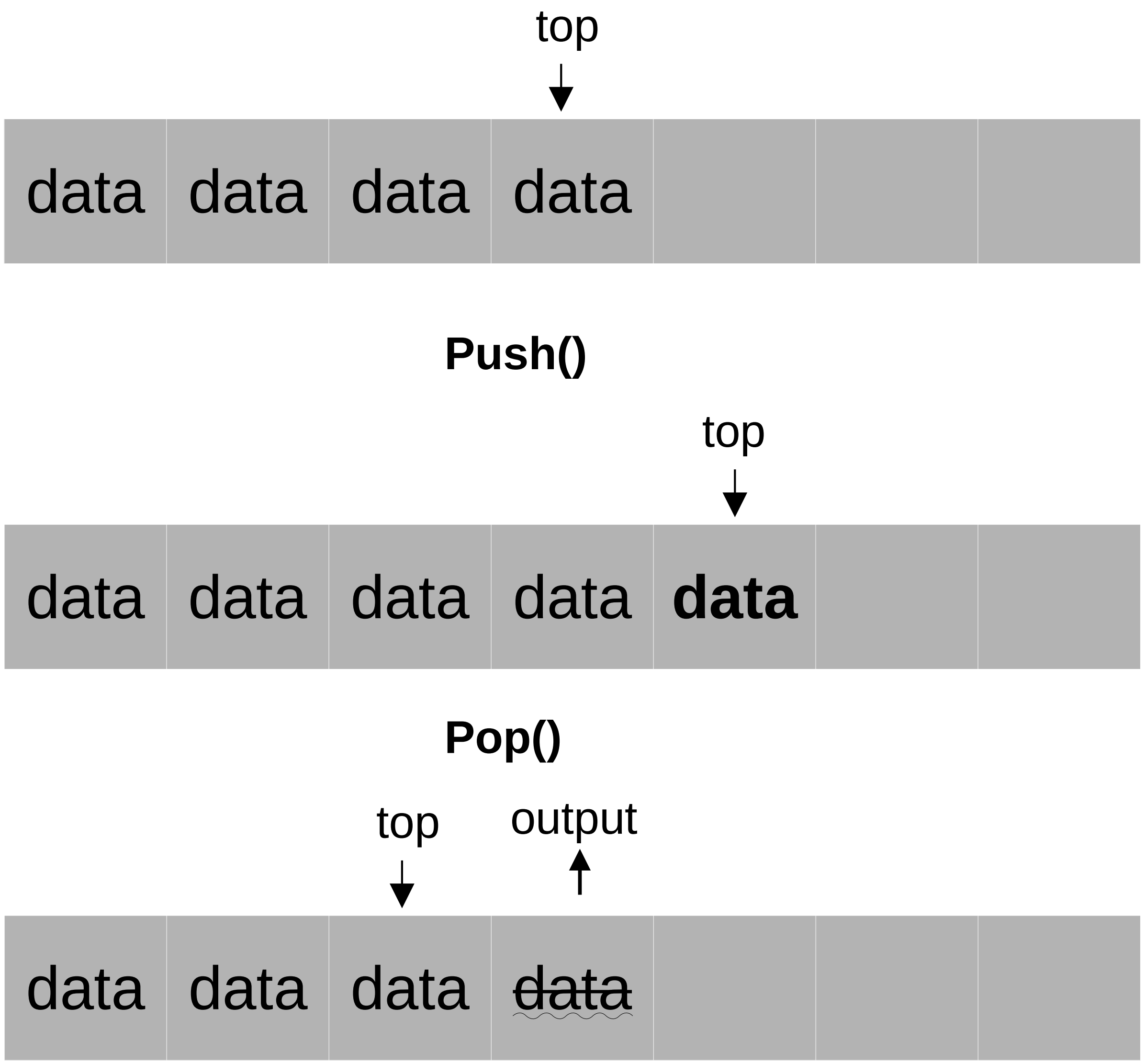
data

data

Stack operations

- **Push()** - pushes element at the top of the stack
- **Pop()** - retrieves and removes (pops) element from the top of the stack
- **Peek()** - retrieves the top element of the stack without removing it
- **Count** – number of elements in the stack

Representing stack with array



top

data data data data

Push()

top

data data data data **data**

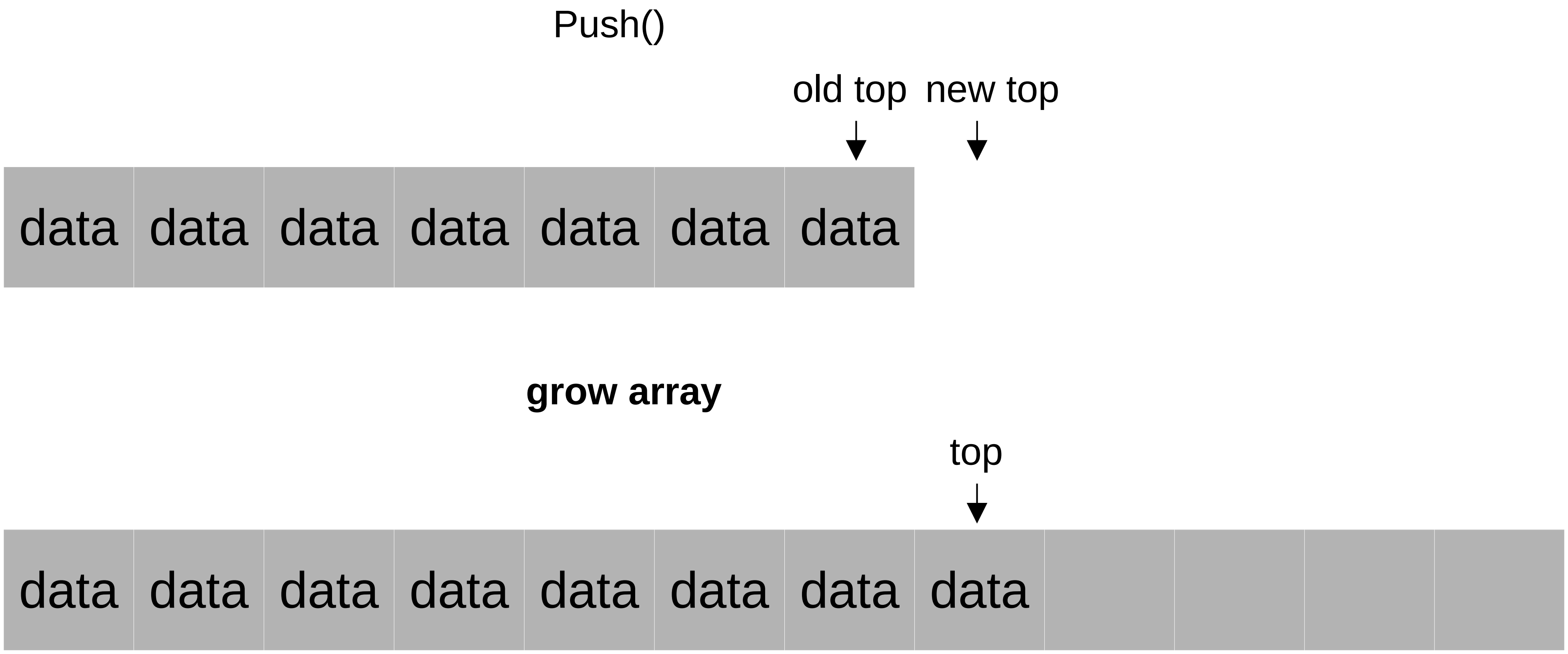
Pop()

top output

data data data ~~data~~

Representing stack with array

- Sometimes we need to grow the array to accommodate more data in the stack



Push()

old top new top

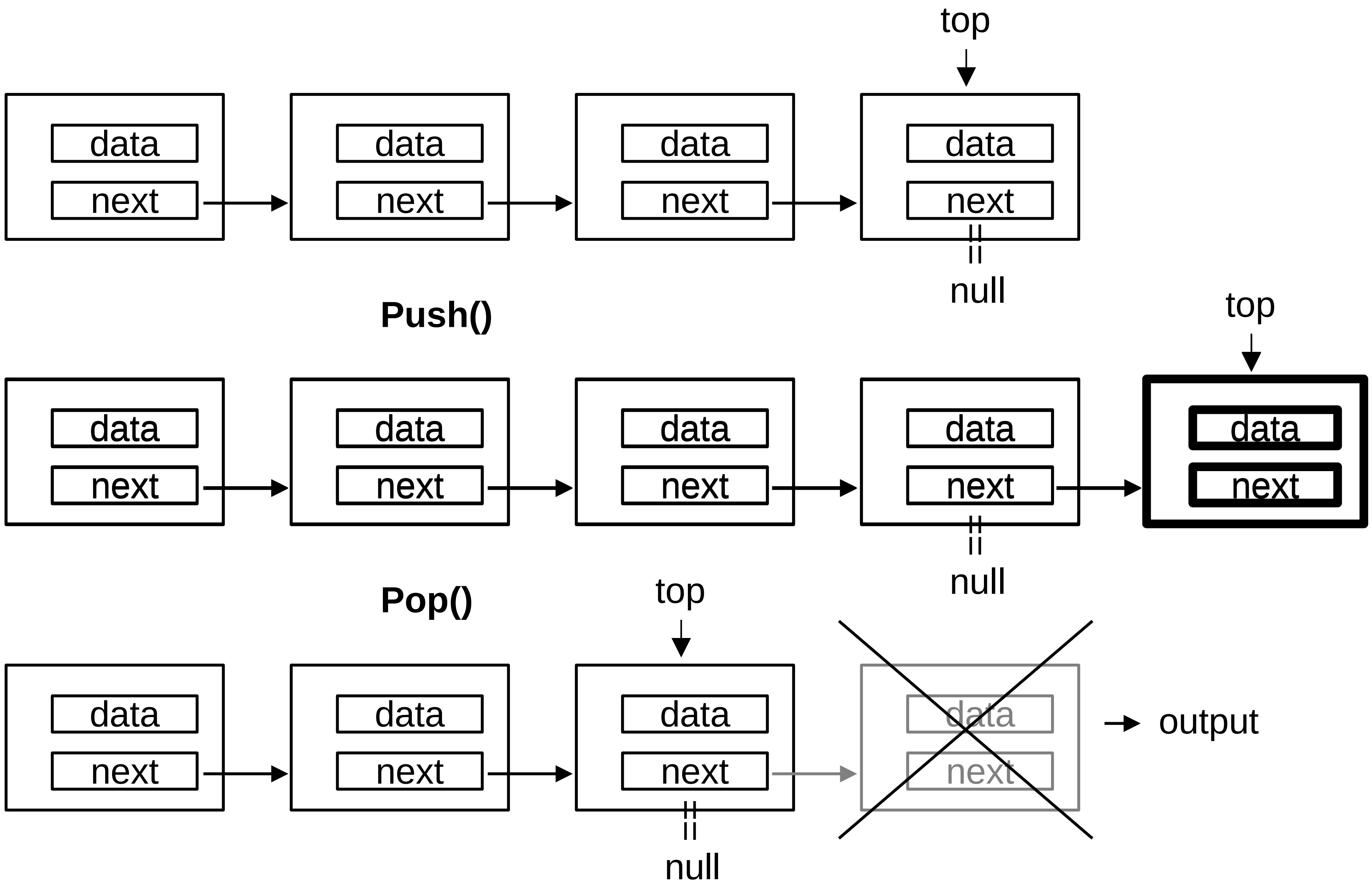
data	data	data	data	data	data	data

grow array

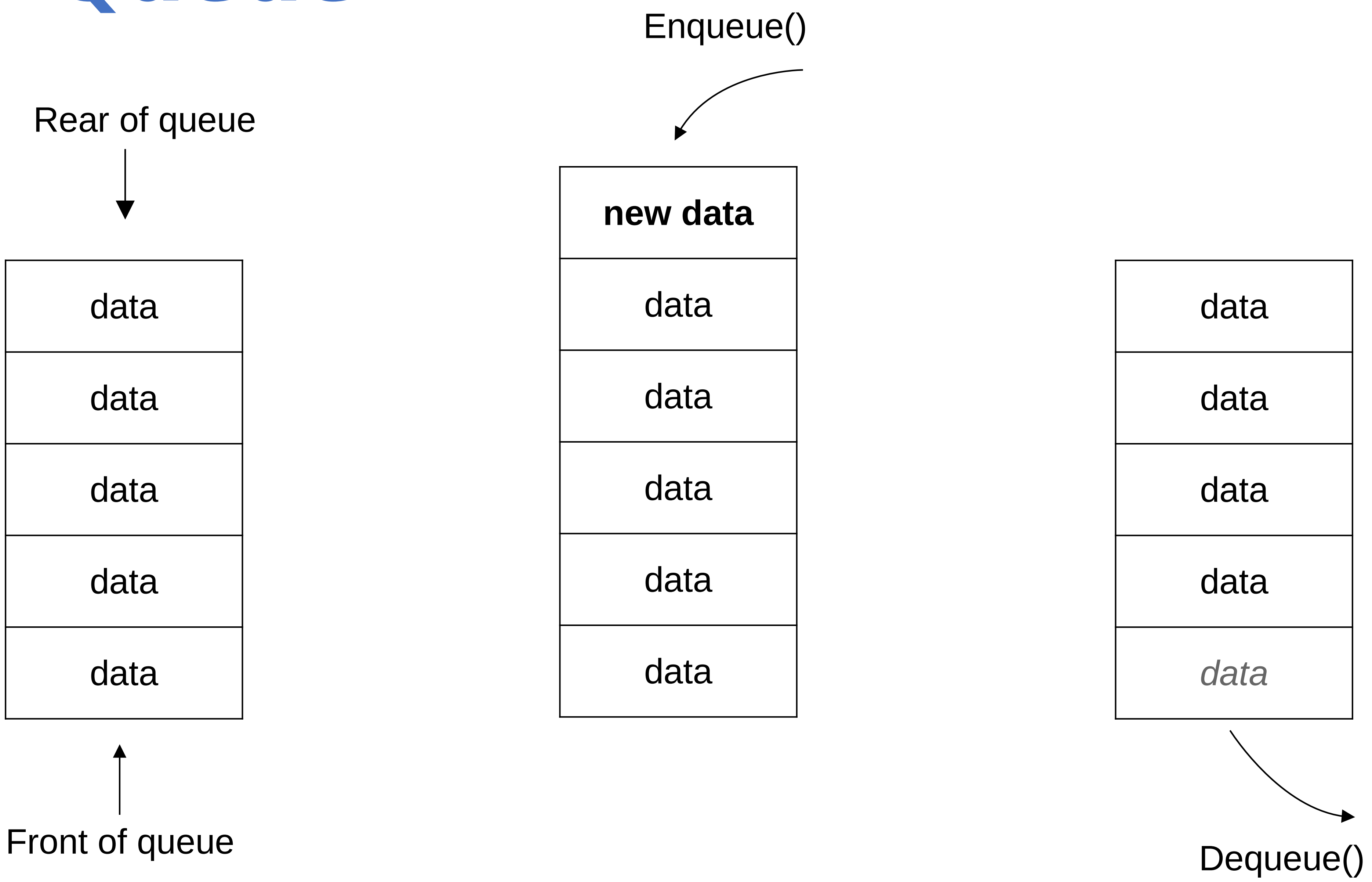
top

data	data	data	data	data	data	data	data				

Representing stack with dynamically linked list



Queue



Rear of queue

data

data

data

data

data

Front of queue

Enqueue()

new data

data

data

data

data

data

data

data

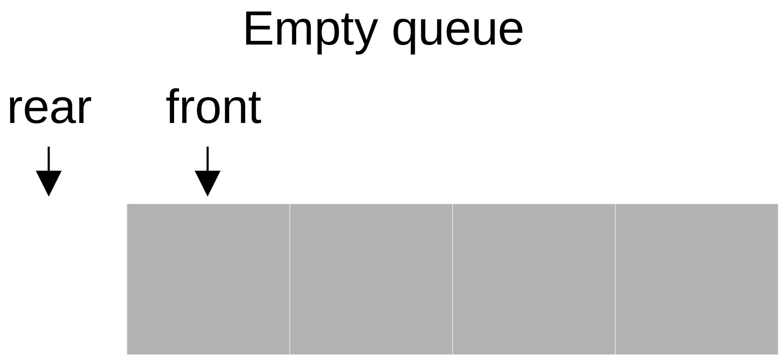
data

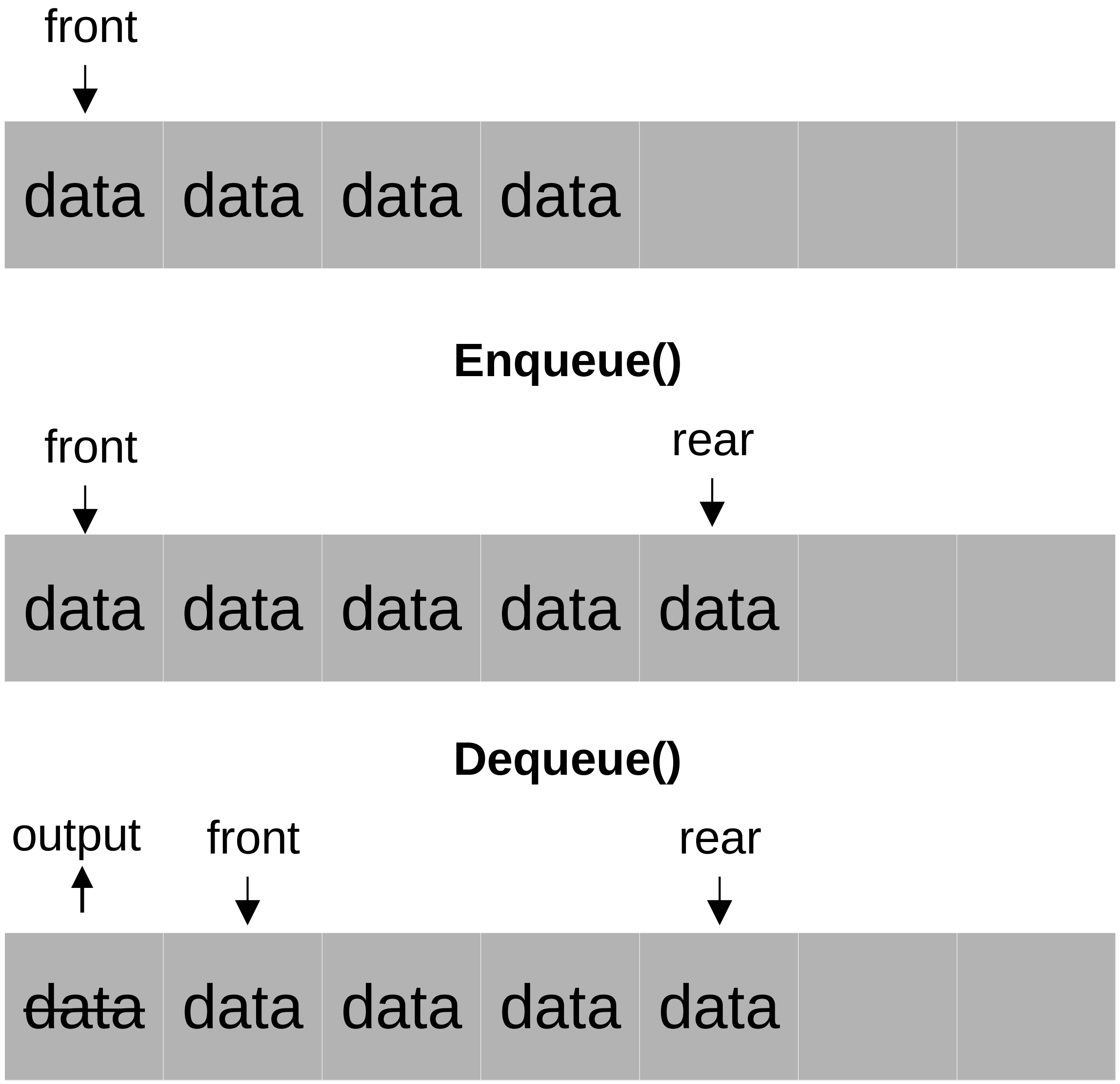
data

data

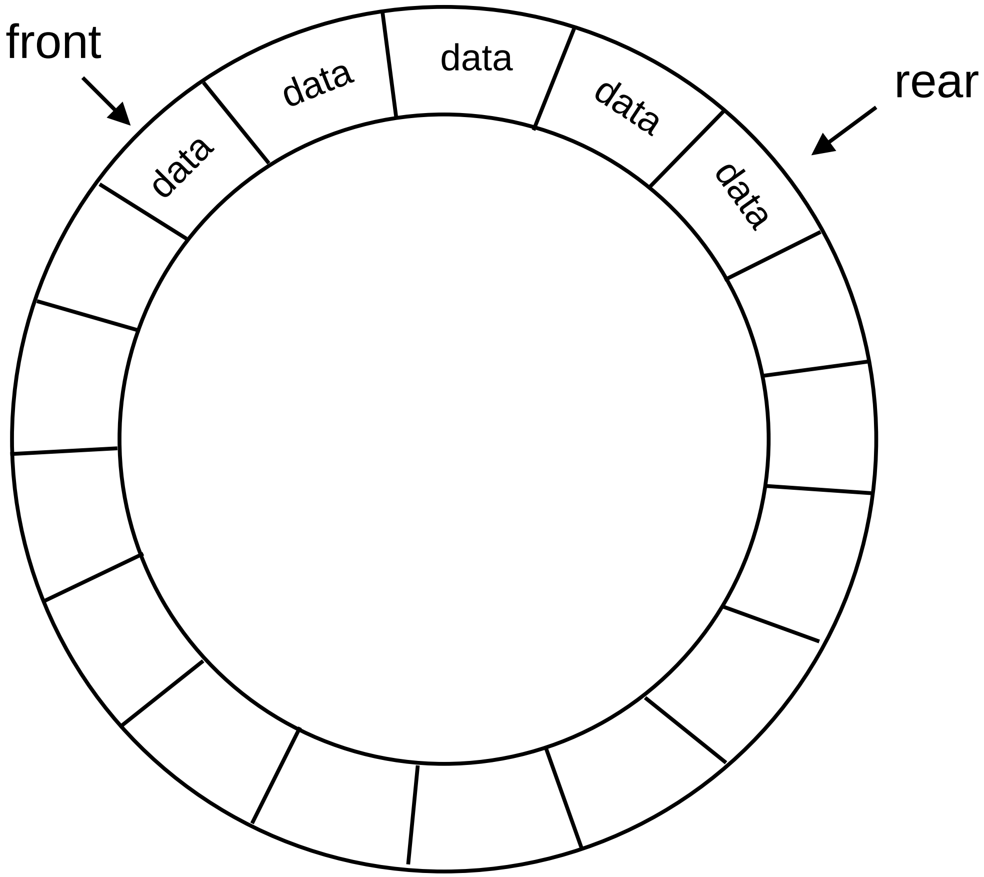
Dequeue()

Representing queue with array

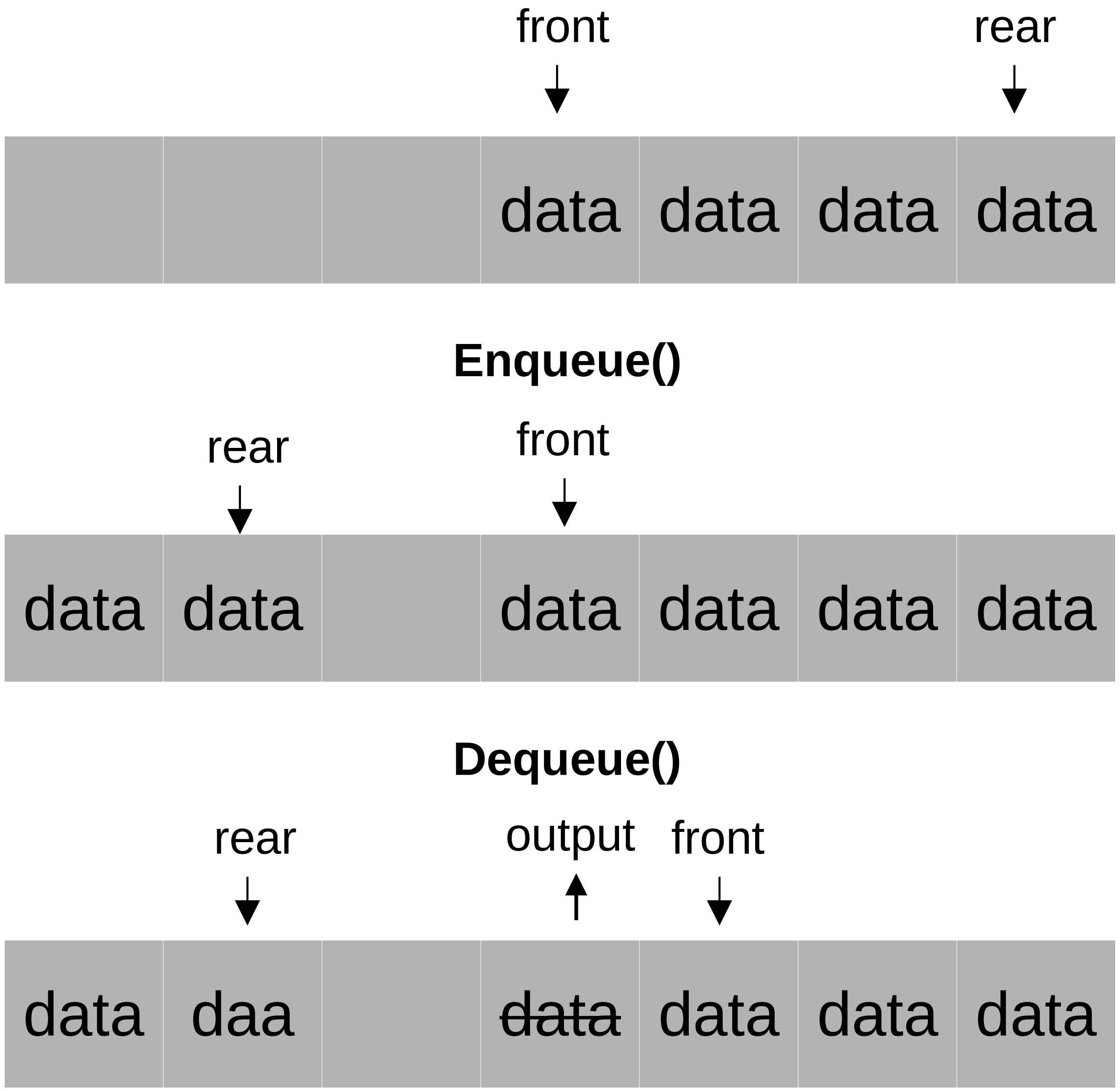




Representing queue with circular array



Representing queue with circular array



			data	data	data	data

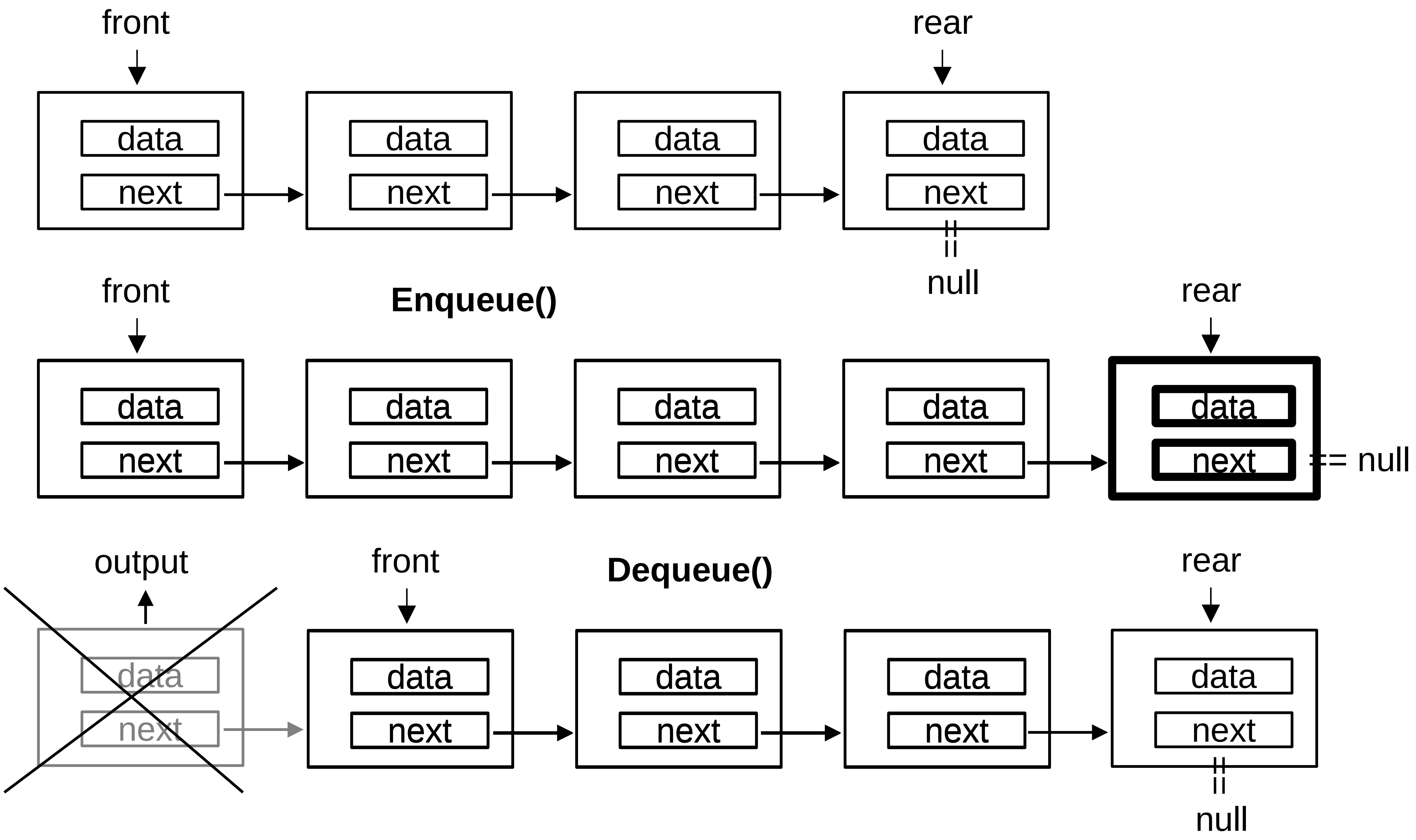
Enqueue()

data	data		data	data	data	data

Dequeue()

data	daa		~~data~~	data	data	data

Representing queue with dynamically linked list



front

rear

data | next → data | next → data | next → data | next == null

front

Enqueue()

rear

data | next → data | next → data | next → data | next → data | next == null

output

front

Dequeue()

rear

data | next → data | next → data | next → data | next → data | next == null

Hashtable

Hashtable represents key – value pairs (keys mapped to values)

Example:
Student grades:

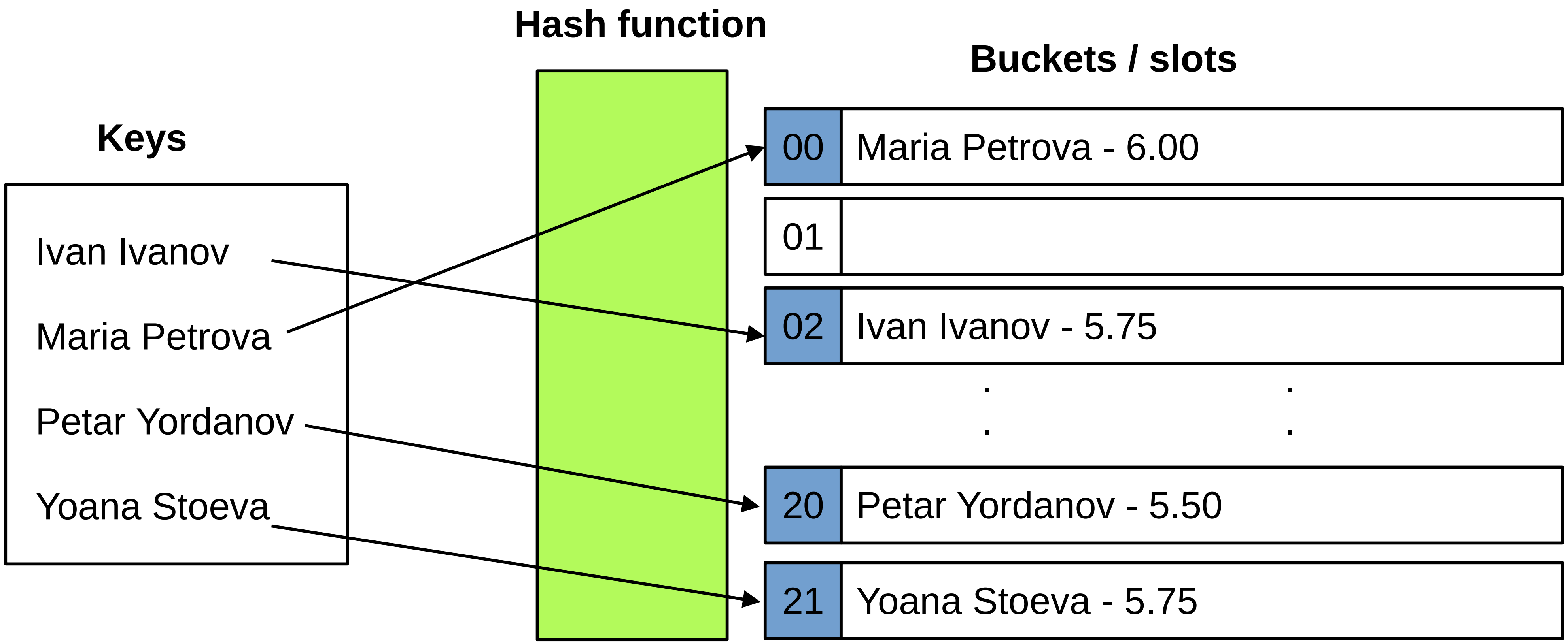
```
{
   “Ivan Ivanov”: 5.75,
   “Maria Petrova”: 6.00,
   “Petar Yordanov”: 5.50,
   “Yoana Stoeva”:  5.75
}
```

Hashtable

- Bad implementation is to use a list of key/value pairs.
  This will take O(n) time for operations to retrieve value by key

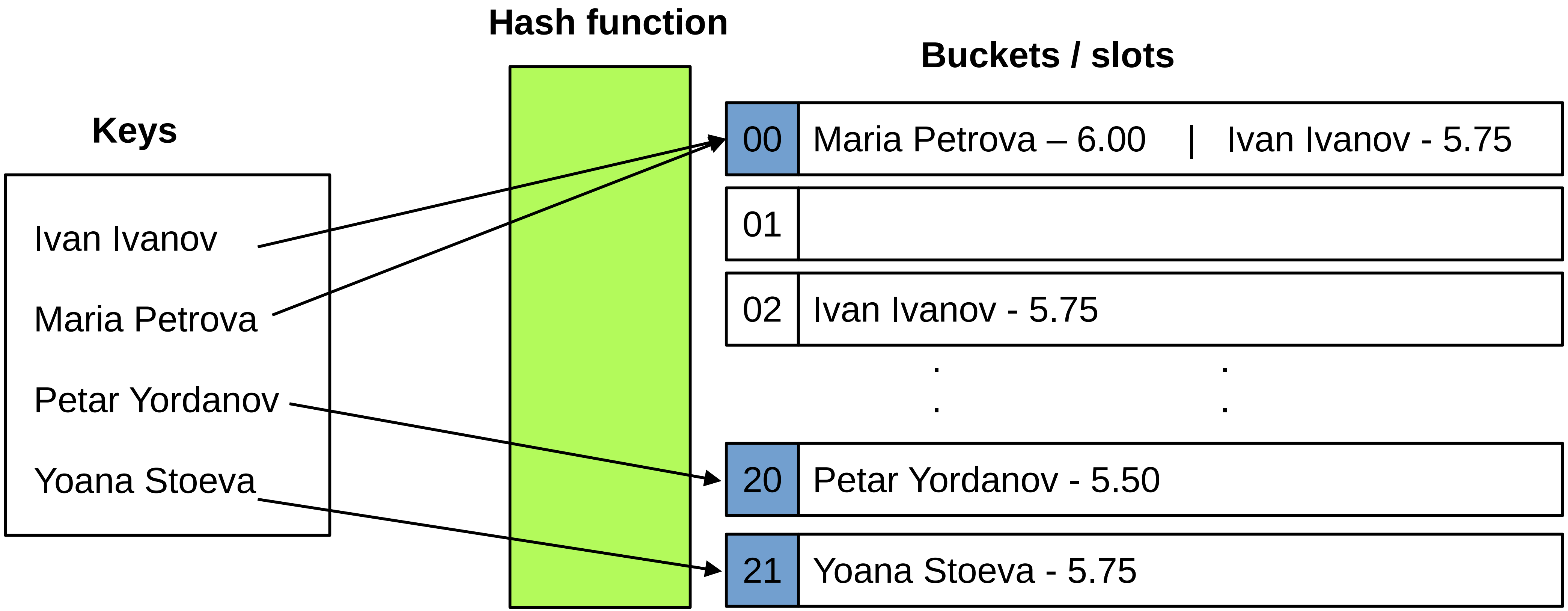
Key / value	Key / value	Key / value	Key / value	Key / value	Key / value	Key / value

Hashtable with hash function



Hashtable collisions

- Collision is when two keys has same hash function result
- It can be solved with linked list in the buckets



Hash function

Buckets / slots

Keys

Ivan Ivanov

Maria Petrova

Petar Yordanov

Yoana Stoeva

00 Maria Petrova – 6.00 | Ivan Ivanov - 5.75

01

02 Ivan Ivanov - 5.75

. .

. .

20 Petar Yordanov - 5.50

21 Yoana Stoeva - 5.75

Hashtable – deciding on hash function

- Best hash function is when it is fast (less operations) and creates minimal collisions

- Example hash functions:
  - Hashing by division: **hash(key) = f(key) mod n**, where f(k) is a function over the key and n is the number of buckets

  - Hashing by multiplication: **hash(key) = trunc(n * ((f(key) * a) – trunc(f(key) * a)))**, where a is a constant between 0 and 1