

# Memoization with Pure Functions

---



**Zoran Horvat**

CEO AT CODING HELMET

@zoranh75

<http://csharpmentor.com>



# Memoization as a Trade-off

**Caller**

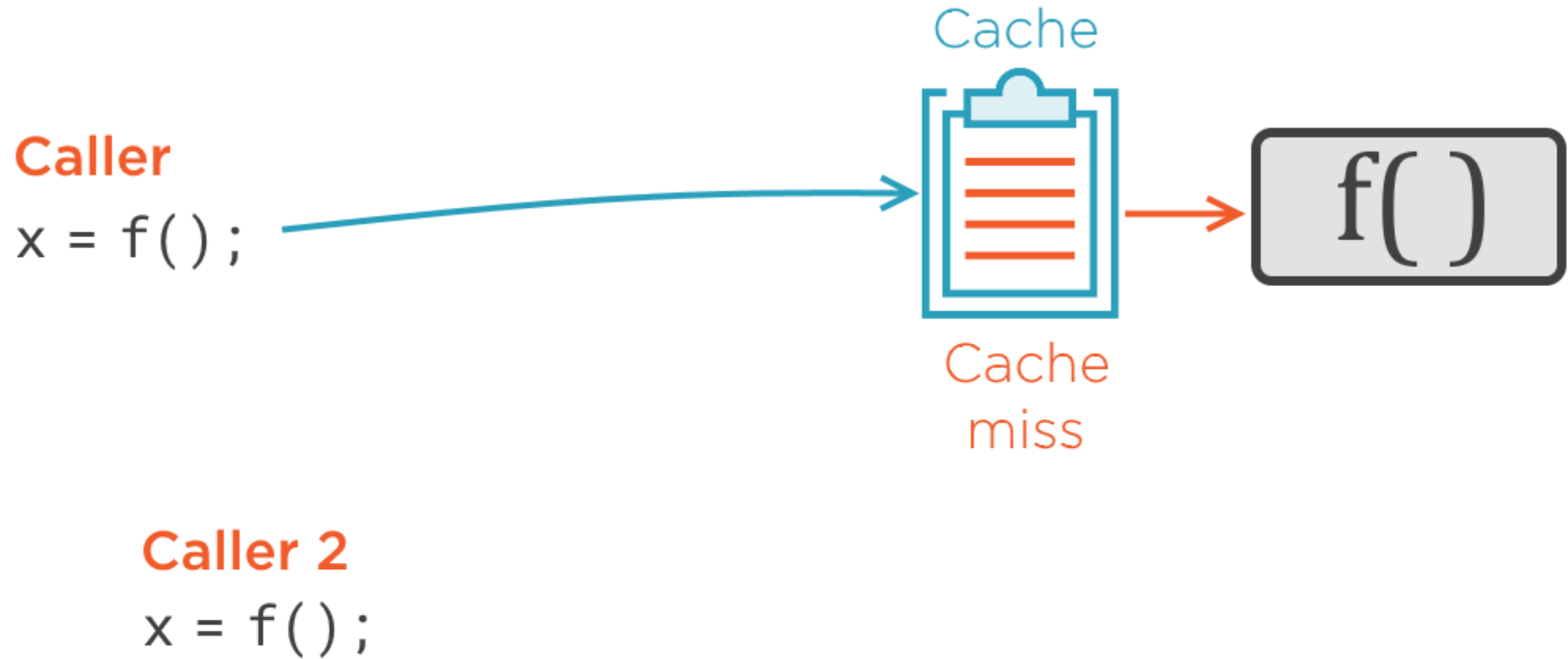
`x = f();`

**Caller 2**

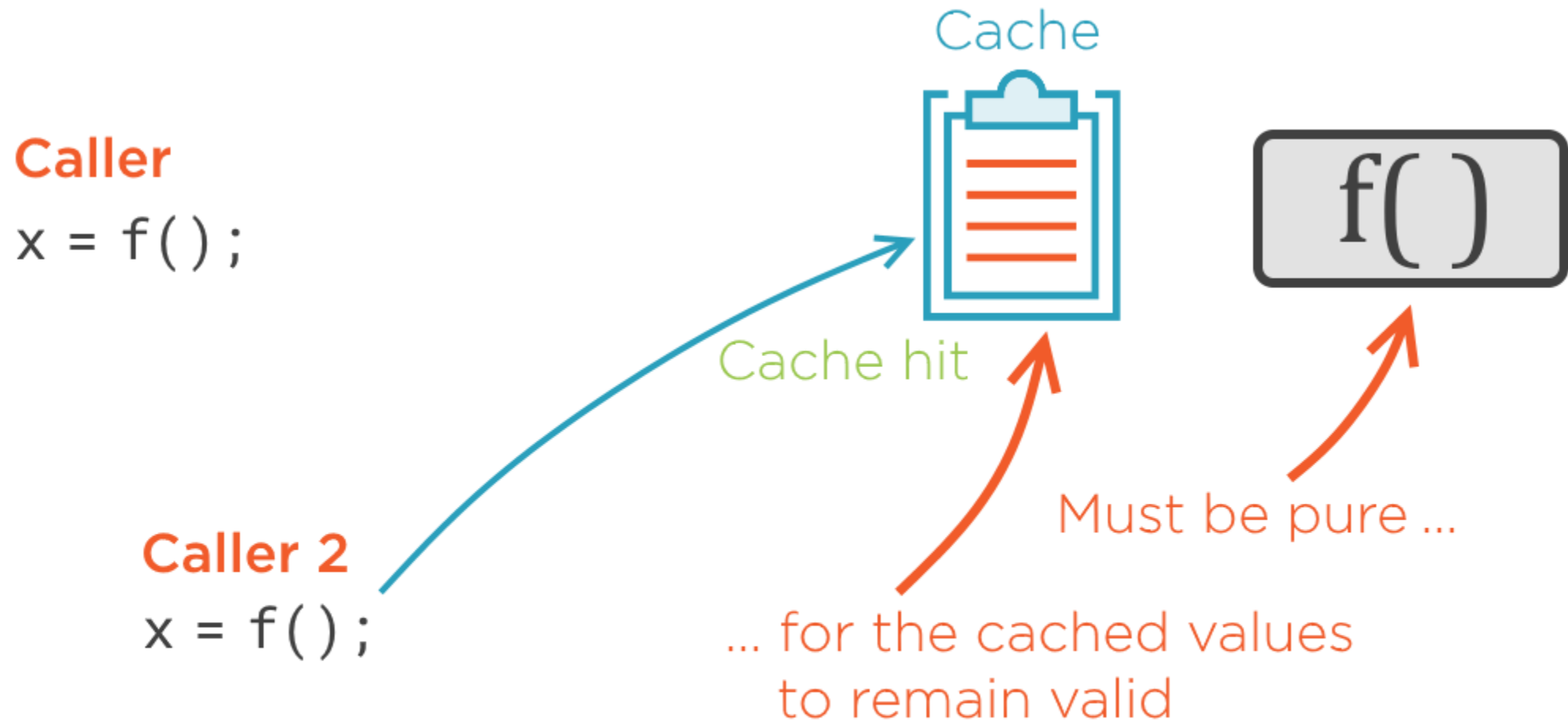
`x = f();`



# Memoization as a Trade-off



# Memoization as a Trade-off



# Memoization as a Trade-off

**Caller**

`x = f();`

Cache



Larger memory footprint  
Possible defects

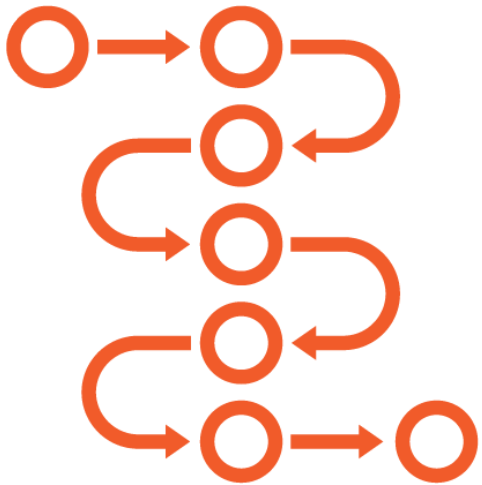
**Caller 2**

`x = f();`

*You have a problem.  
You choose to solve it by caching results.  
Now you have two problems.*



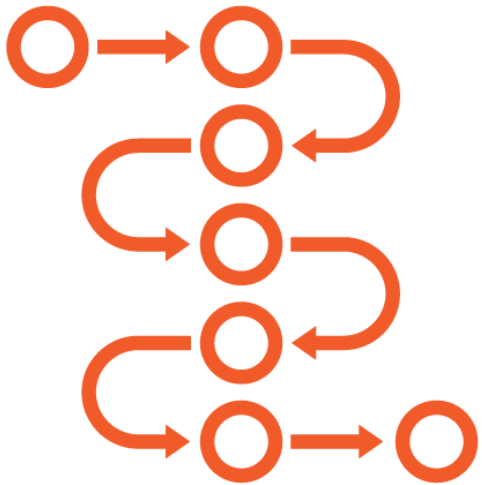
# Dynamic Programming Defined



Dynamic programming (a.k.a. dynamic optimization) is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions.

[https://en.wikipedia.org/wiki/Dynamic\\_programming](https://en.wikipedia.org/wiki/Dynamic_programming)

# Dynamic Programming Defined

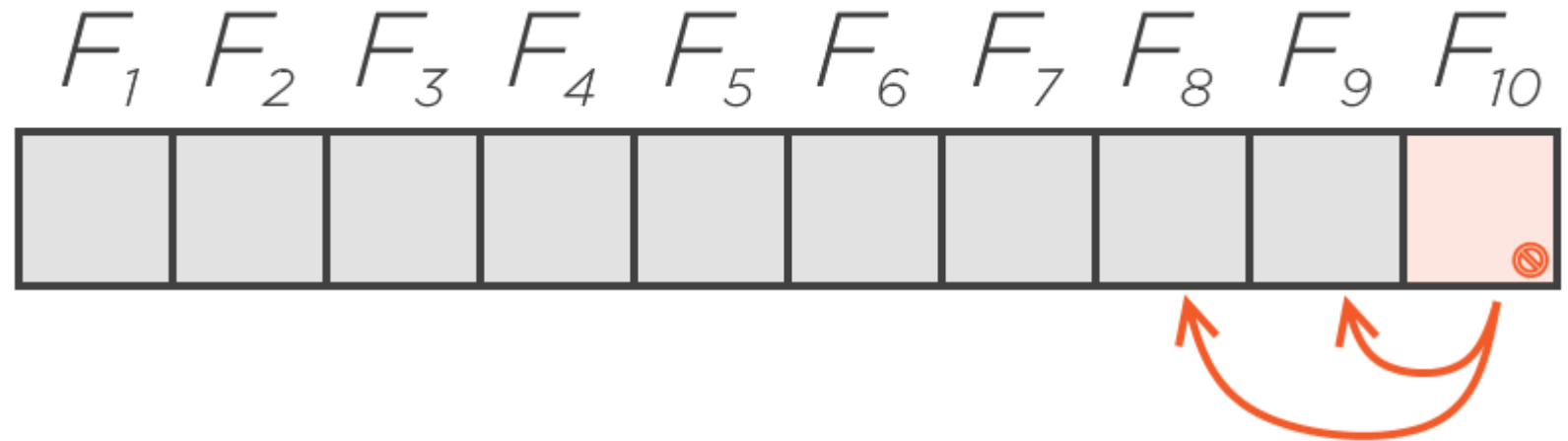
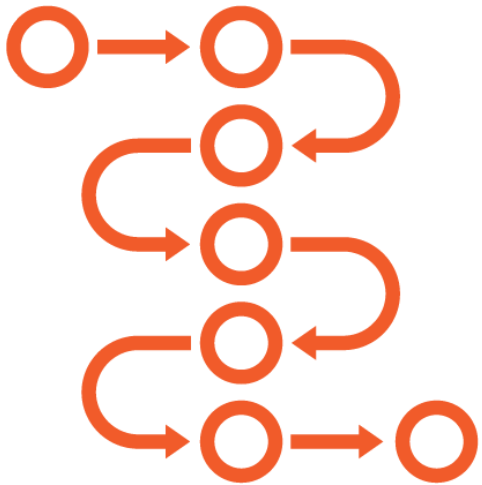


Dynamic programming (a.k.a. dynamic optimization) is a method for solving a **complex problem** by breaking it down into a collection of simpler **subproblems**, solving each of those subproblems just **once**, and **storing** their solutions.

[https://en.wikipedia.org/wiki/Dynamic\\_programming](https://en.wikipedia.org/wiki/Dynamic_programming)

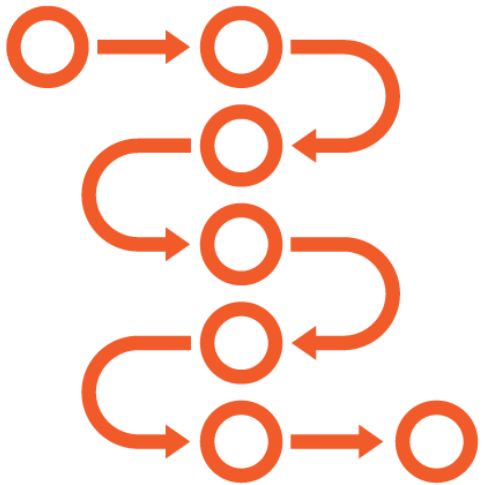


# Dynamic Programming Defined

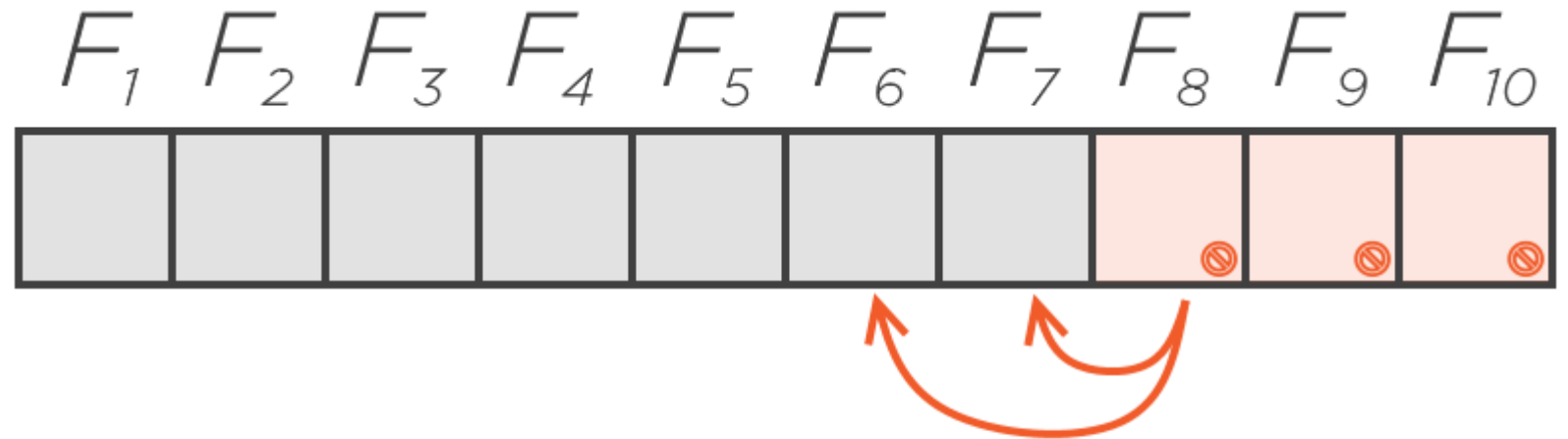
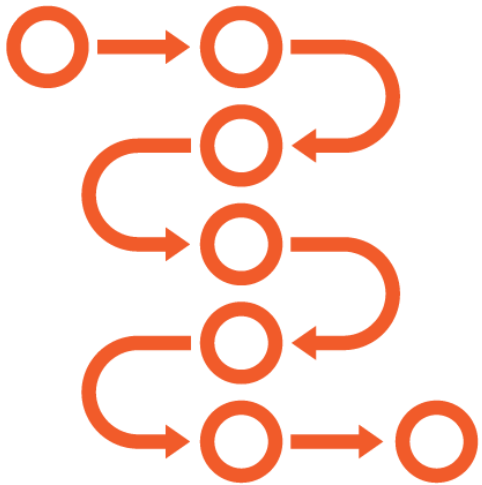




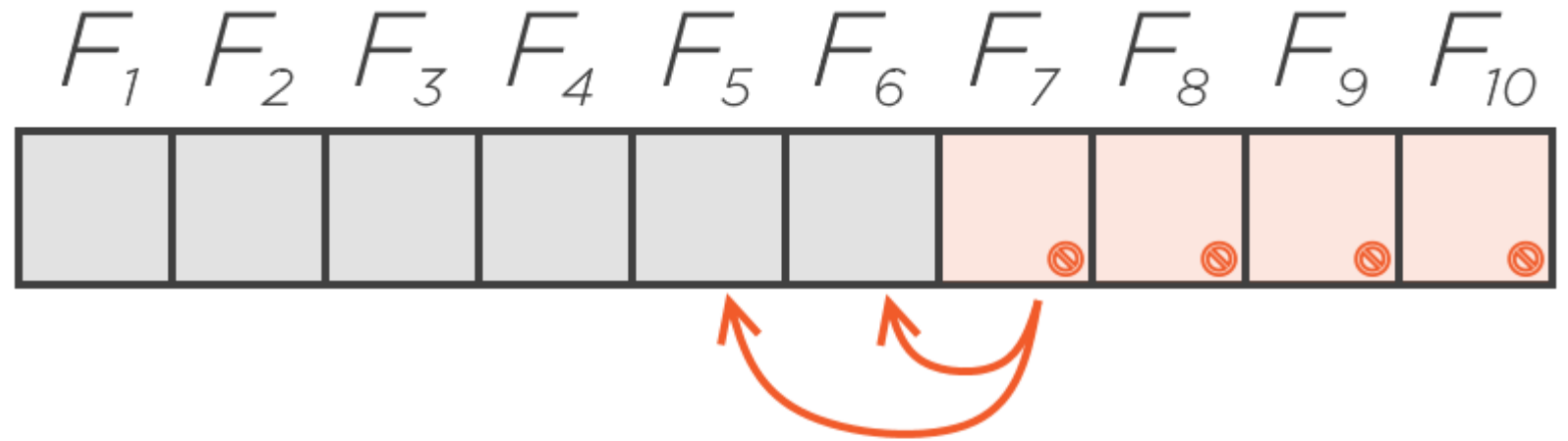
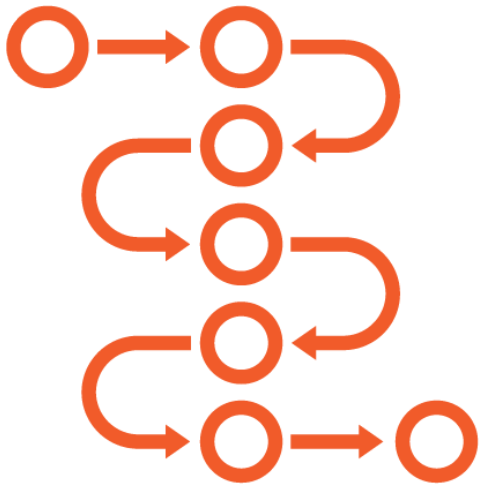
# Dynamic Programming Defined



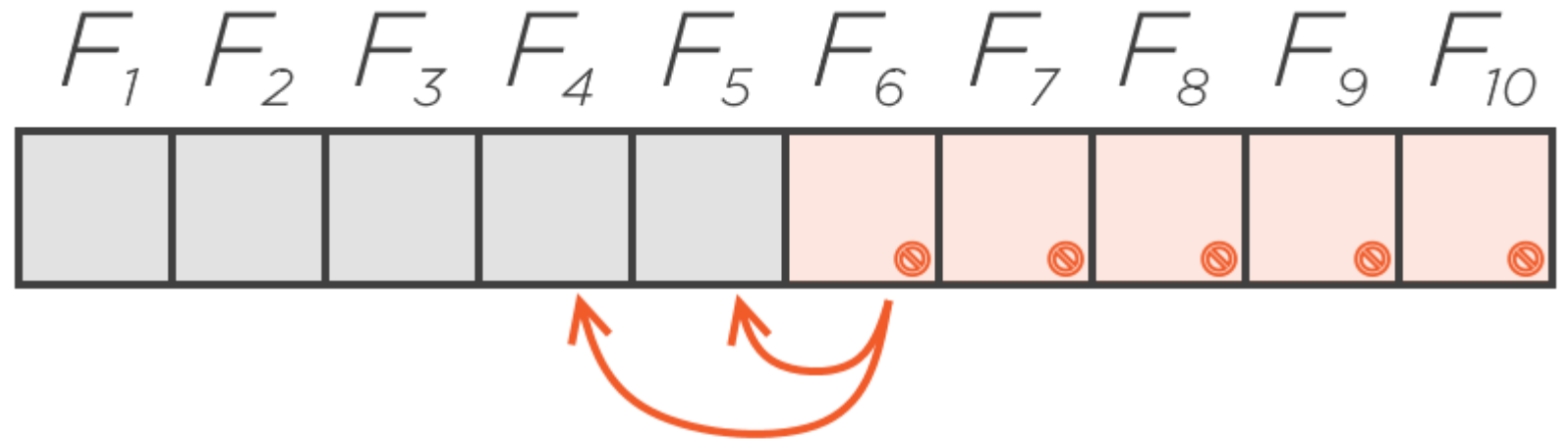
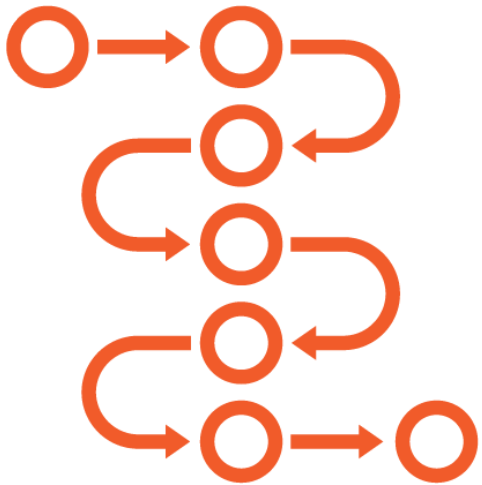
# Dynamic Programming Defined



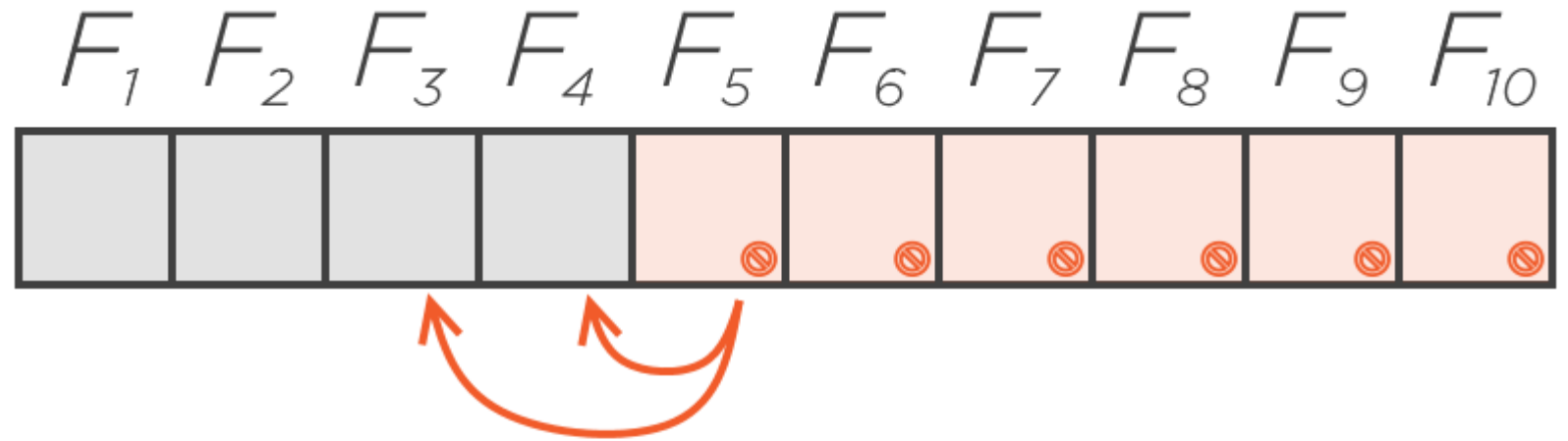
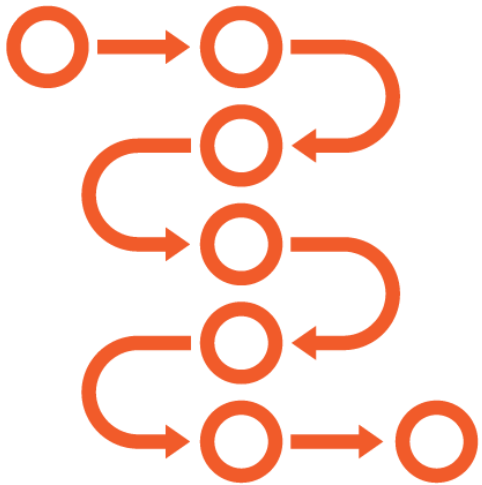
# Dynamic Programming Defined



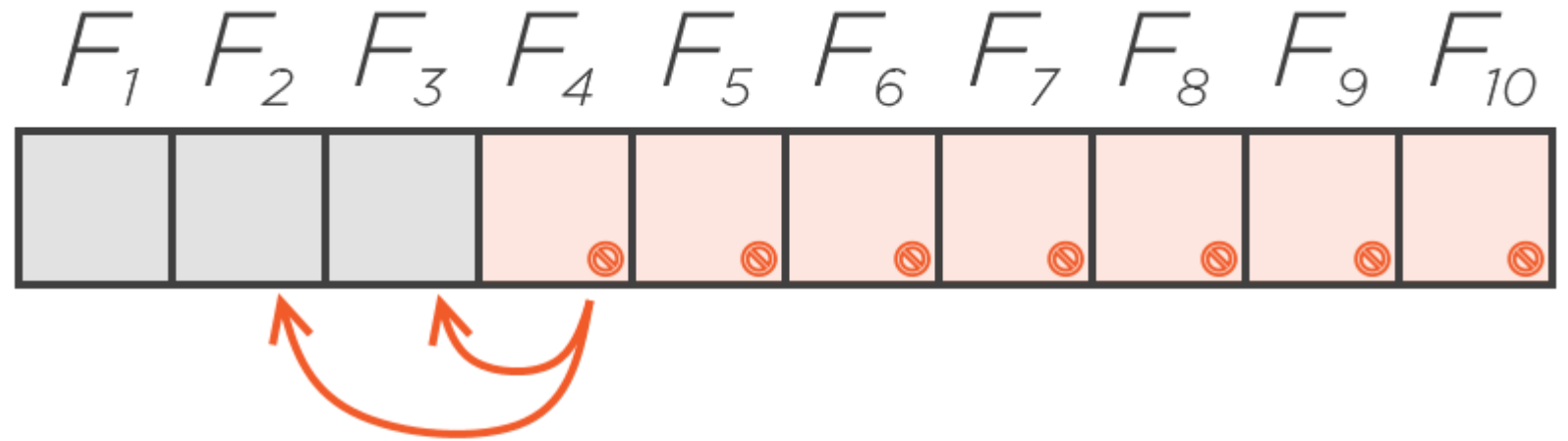
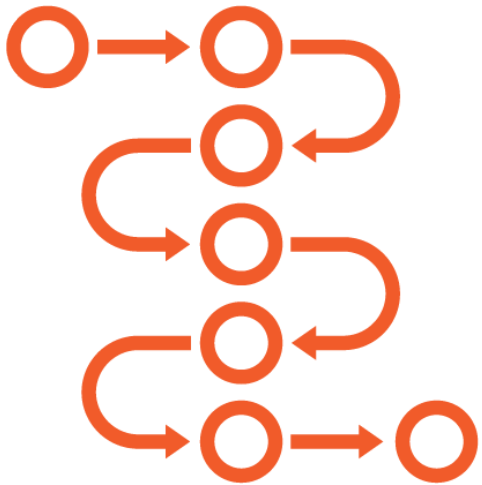
# Dynamic Programming Defined



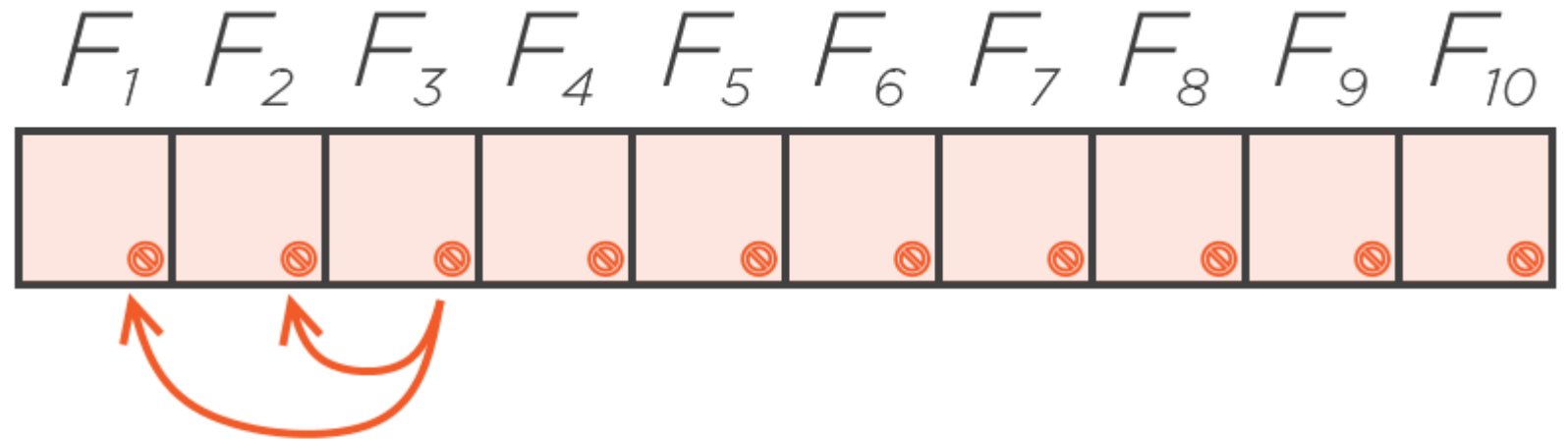
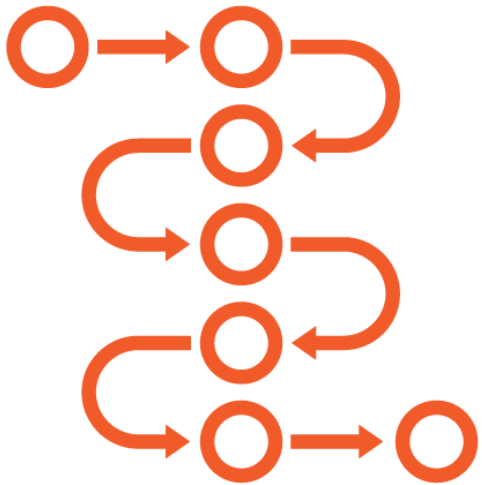
# Dynamic Programming Defined



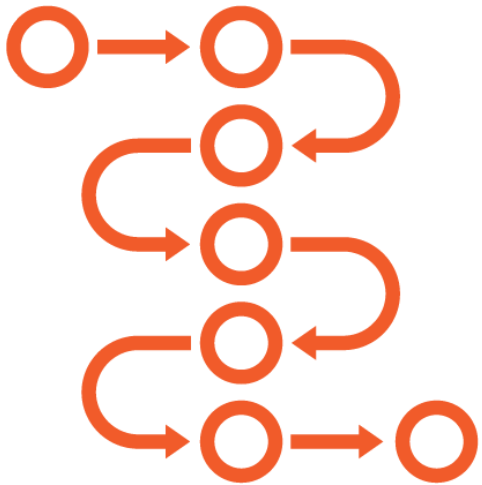
# Dynamic Programming Defined



# Dynamic Programming Defined



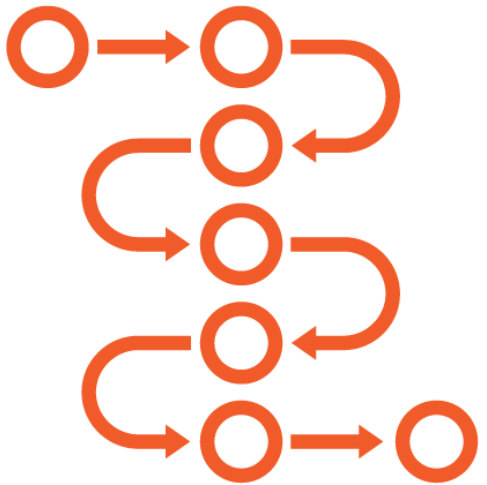
# Dynamic Programming Defined



$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$
1 ✓	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗

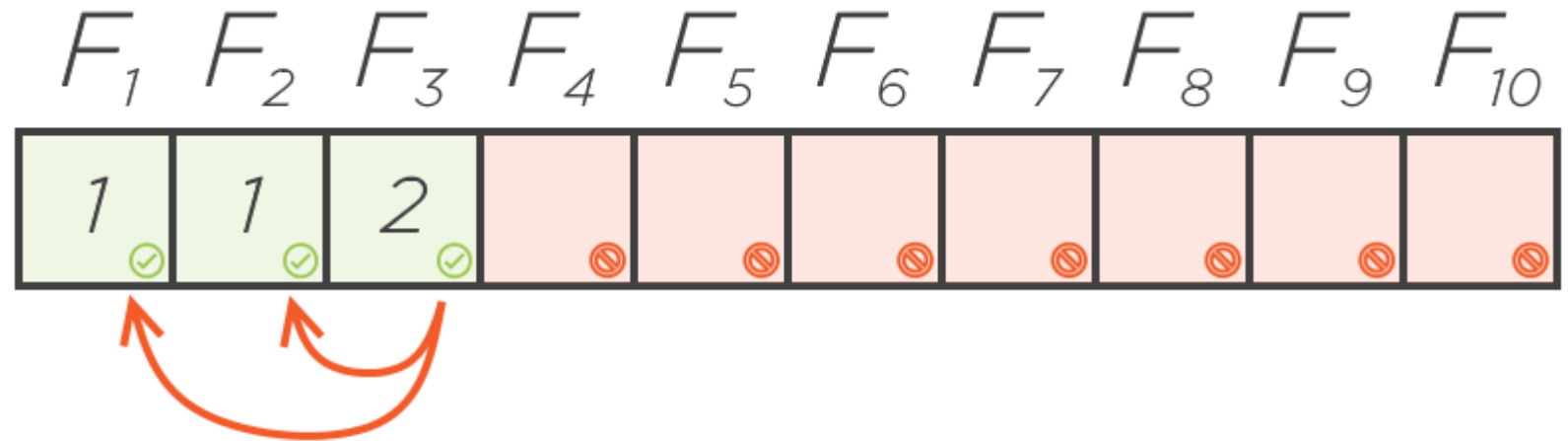
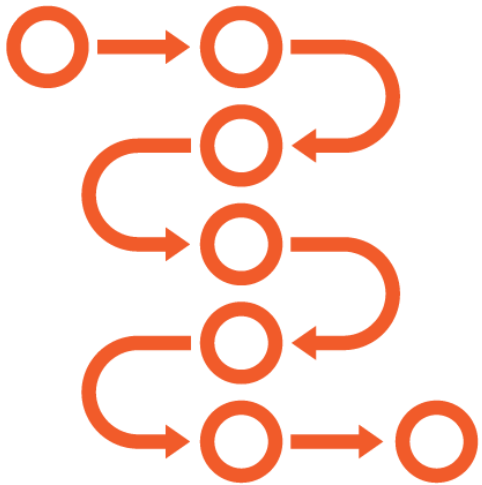


# Dynamic Programming Defined

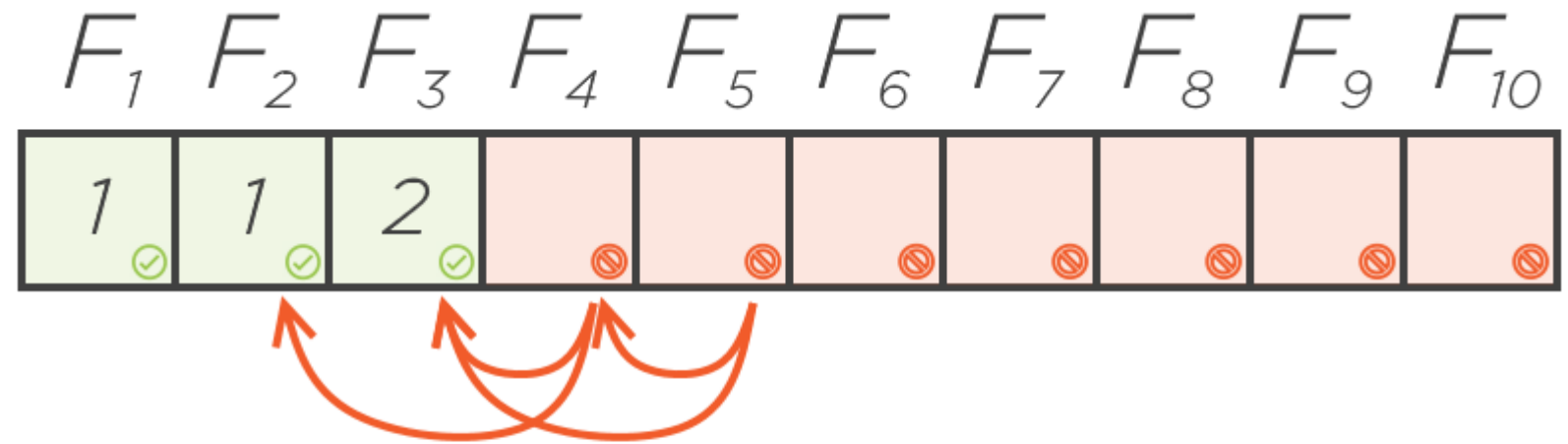
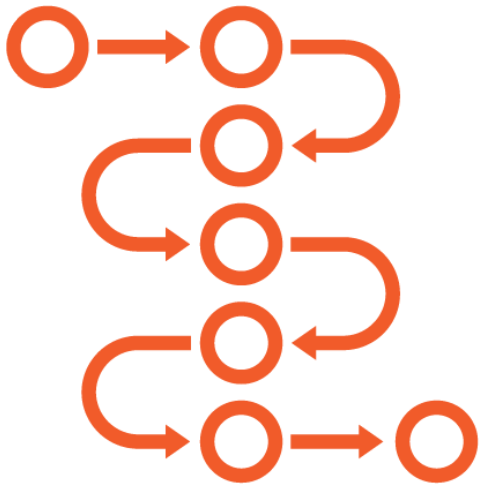


$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$
1	1								
✓	✓	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗

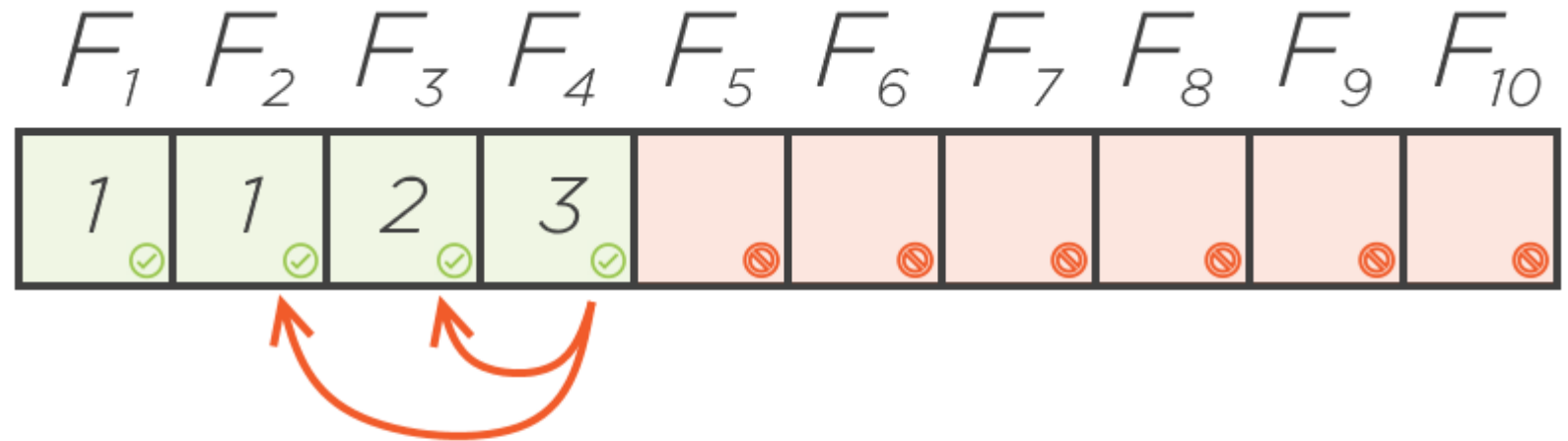
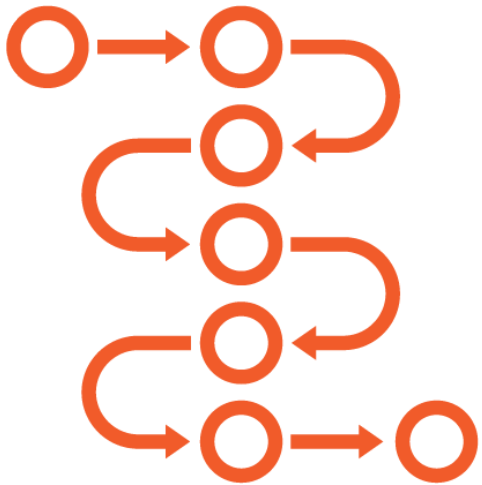
# Dynamic Programming Defined



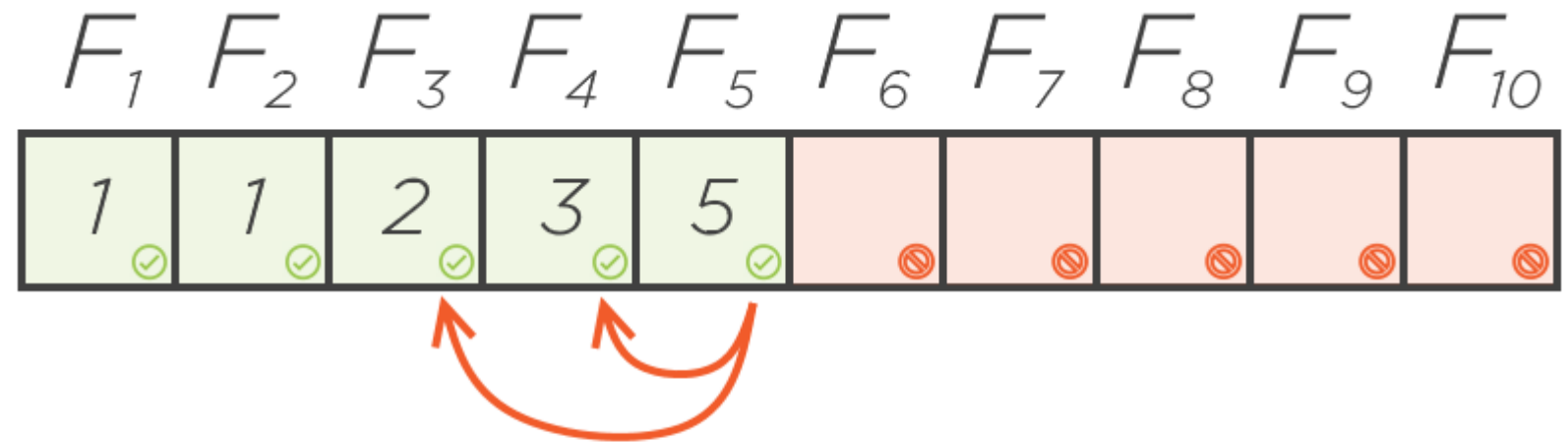
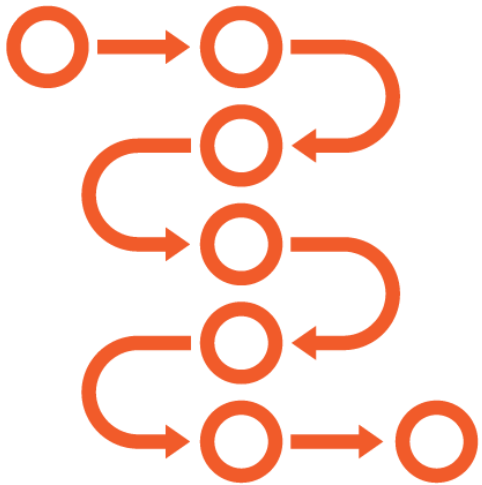
# Dynamic Programming Defined



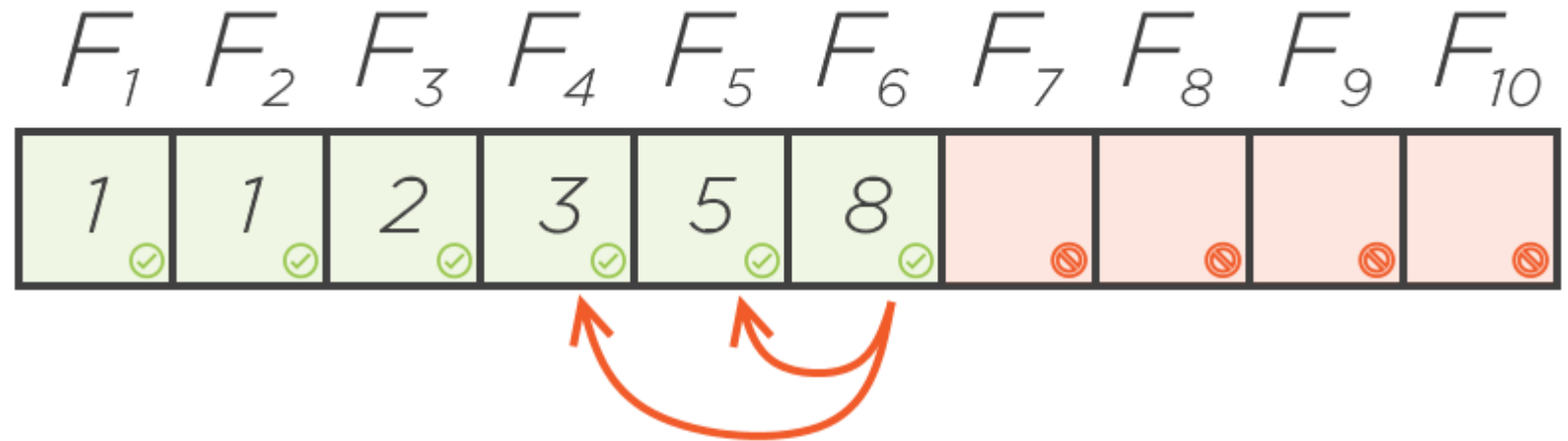
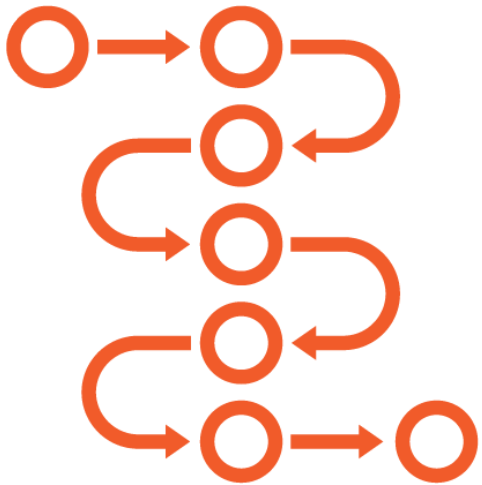
# Dynamic Programming Defined



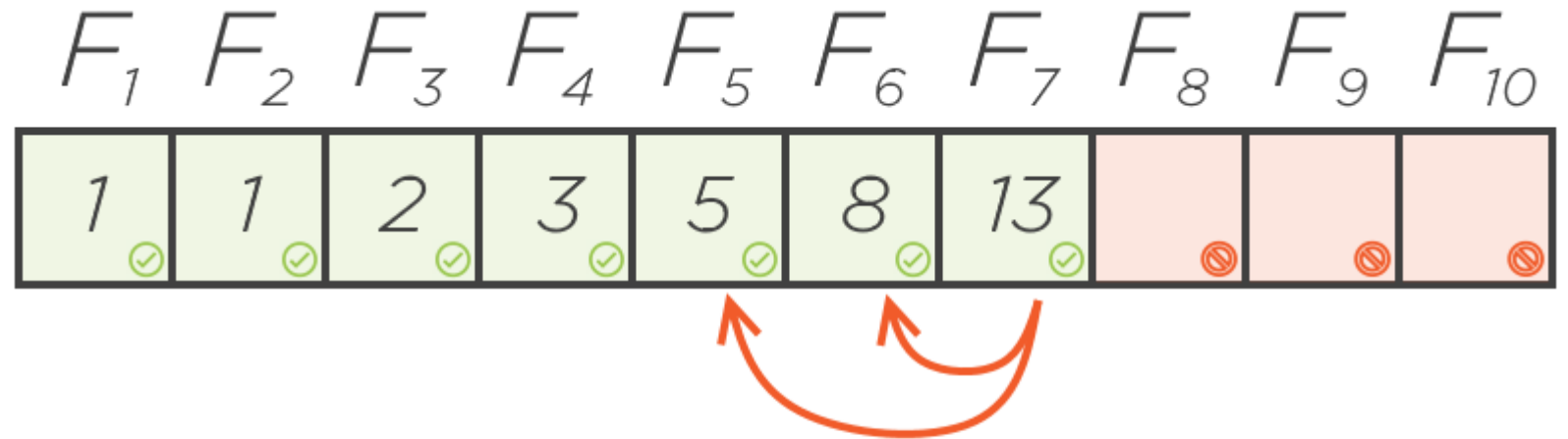
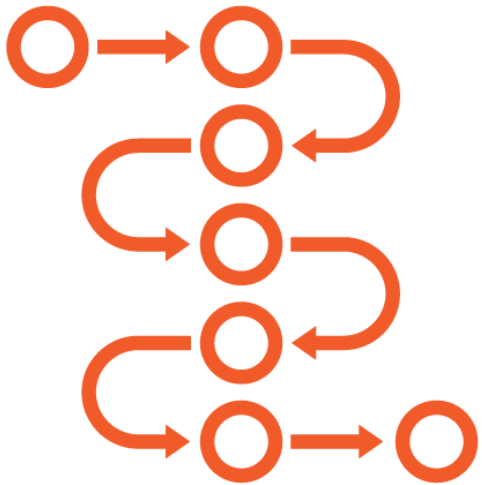
# Dynamic Programming Defined



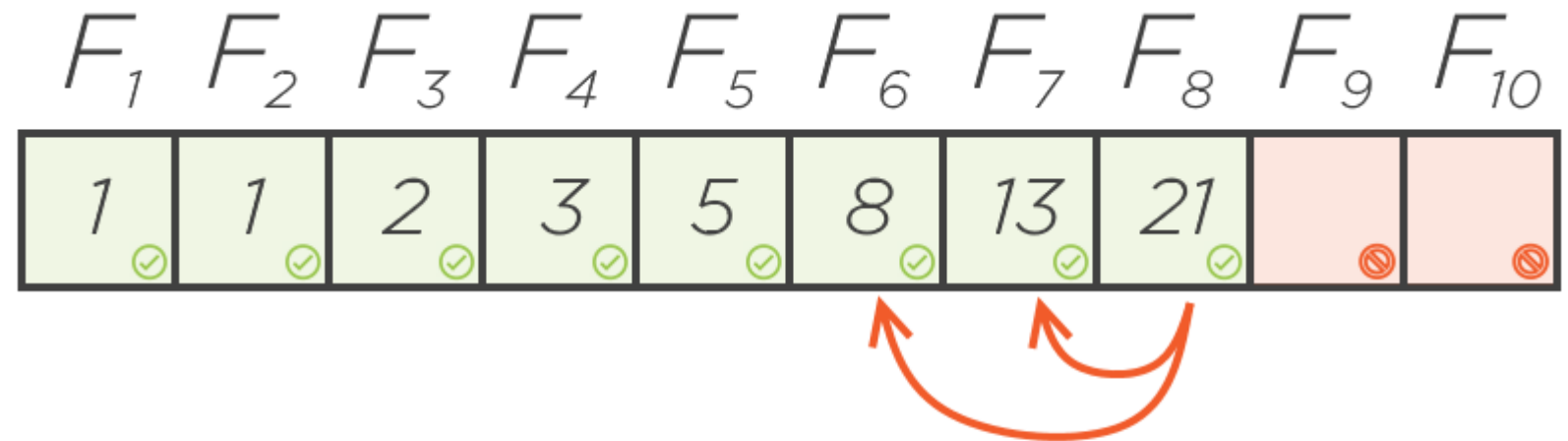
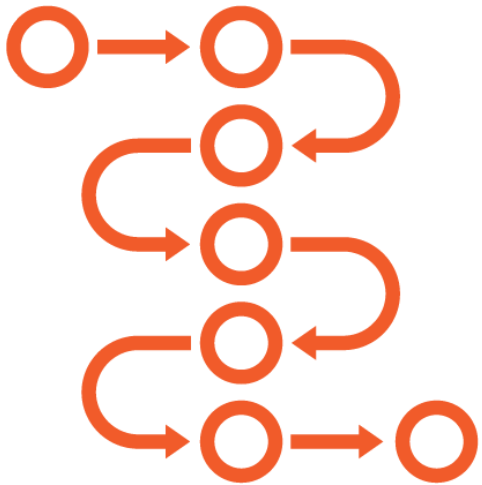
# Dynamic Programming Defined



# Dynamic Programming Defined

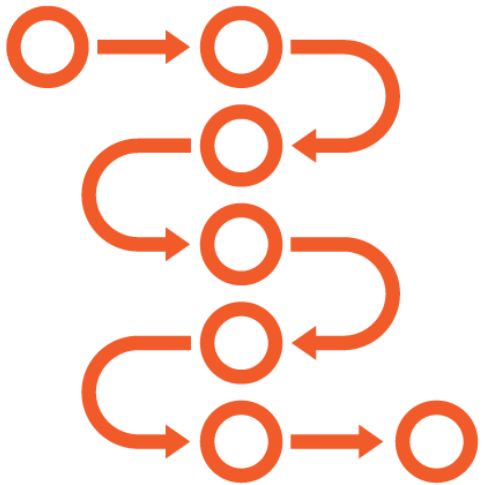


# Dynamic Programming Defined





# Dynamic Programming Defined

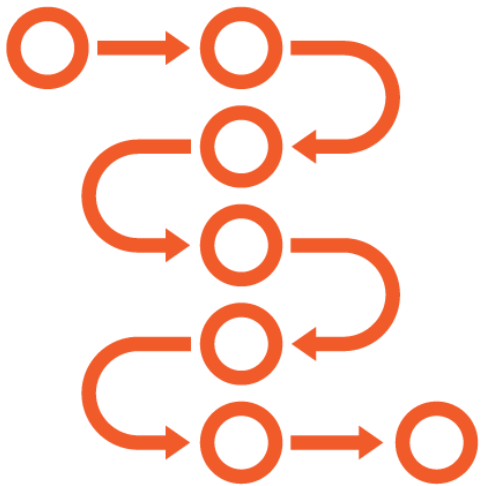


$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$
1	1	2	3	5	8	13	21	34	

Diagram illustrating the Fibonacci sequence values stored in an array, with arrows indicating the recurrence relation  $F_n = F_{n-1} + F_{n-2}$  for  $n \geq 3$ .

The array contains values for  $F_1$  through  $F_9$ , with  $F_{10}$  being empty. The values are:  $F_1=1$ ,  $F_2=1$ ,  $F_3=2$ ,  $F_4=3$ ,  $F_5=5$ ,  $F_6=8$ ,  $F_7=13$ ,  $F_8=21$ ,  $F_9=34$ . The last cell ( $F_{10}$ ) is highlighted in red and contains a red 'X' icon, indicating it is the current state or a target state.

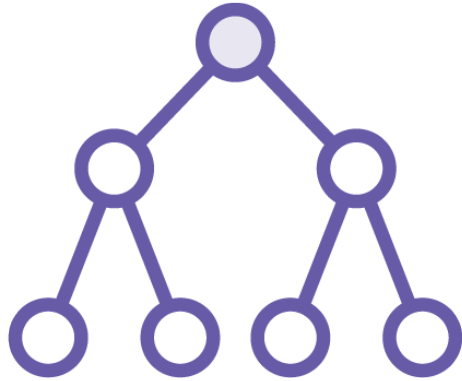
# Dynamic Programming Defined



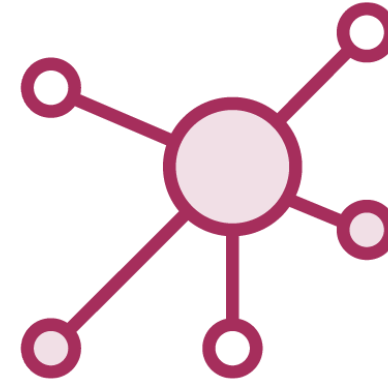
$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$	$F_7$	$F_8$	$F_9$	$F_{10}$
1	1	2	3	5	8	13	21	34	55

**Memoization**  $F_{10}=55$   
as a special kind of  
**Dynamic Programming**

# Dynamic Programming

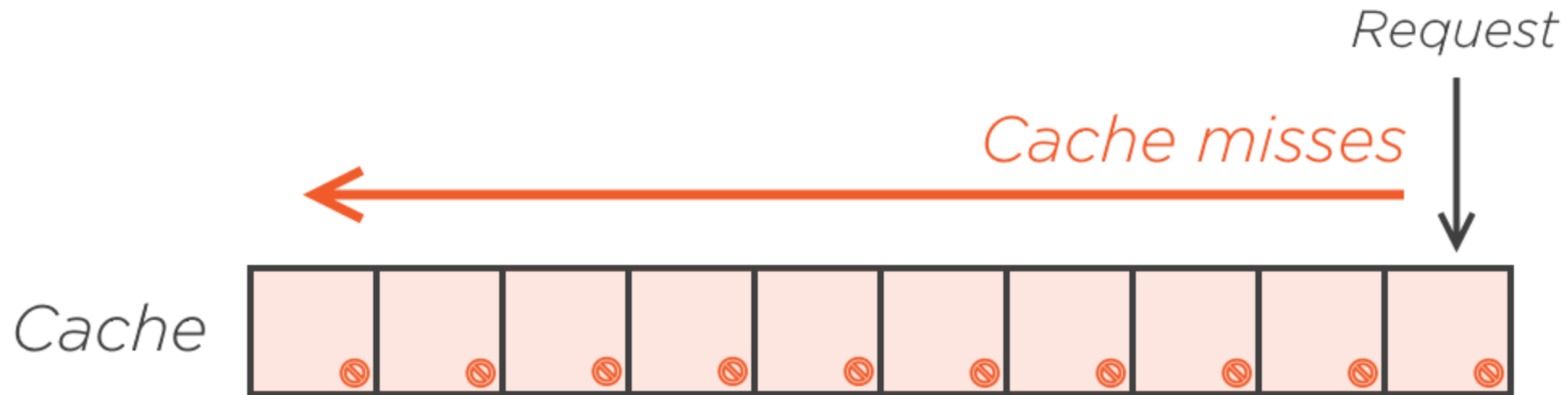


Memoization is  
a special case of  
Dynamic Programming

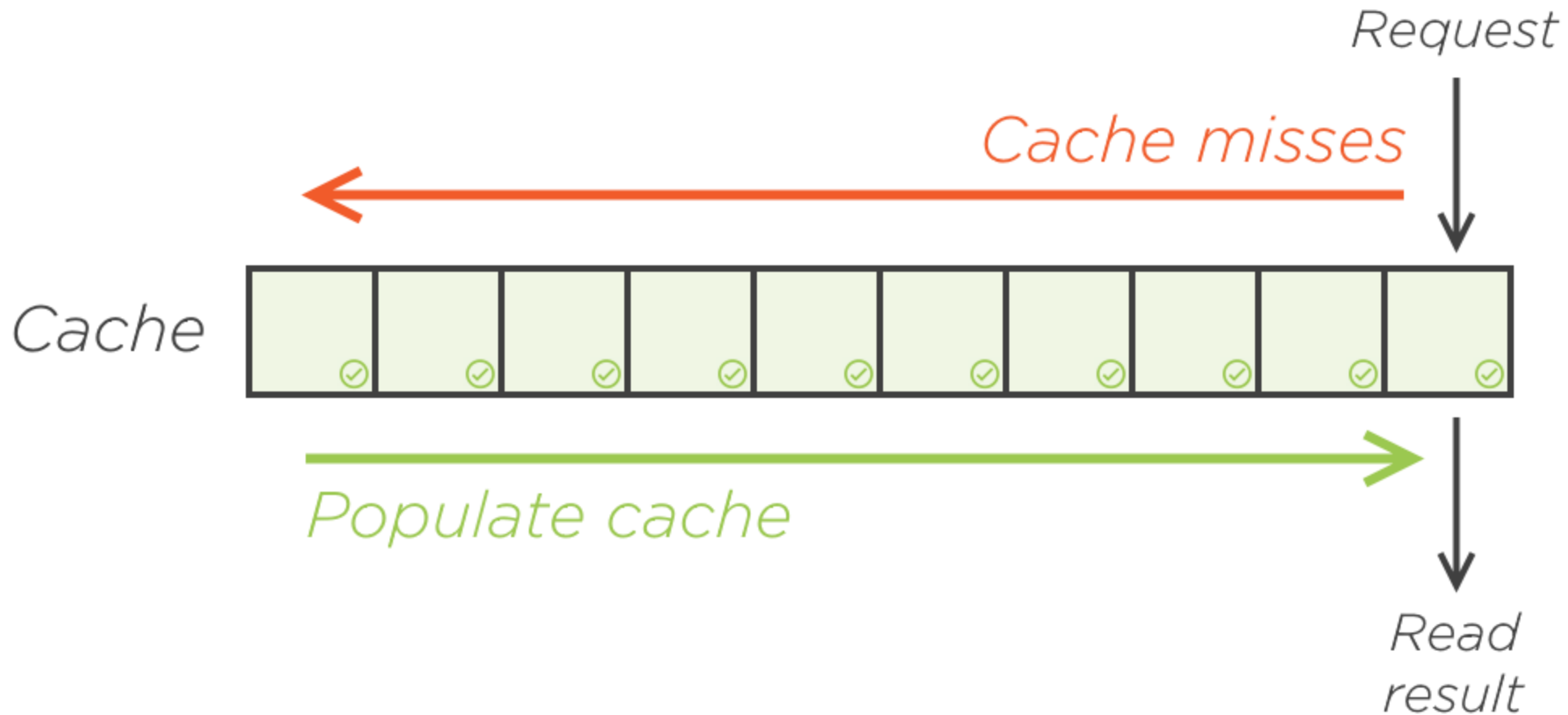


Dynamic Programming is  
a generalization of  
Memoization

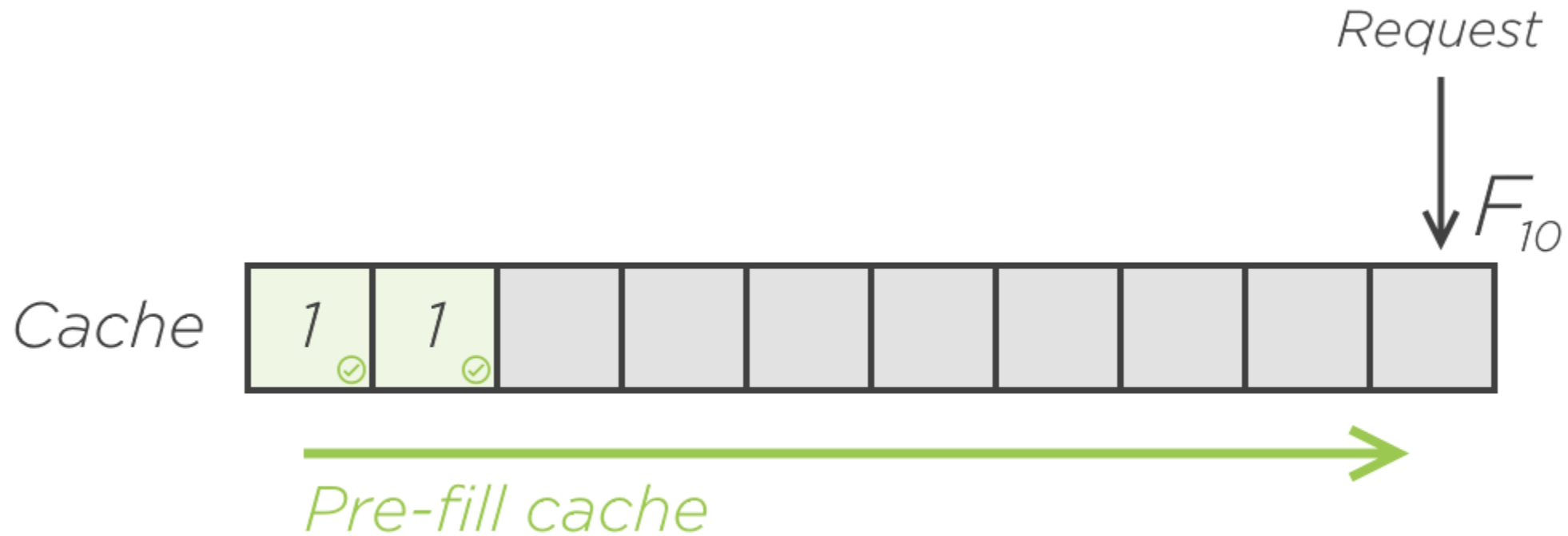
# Dynamic Programming



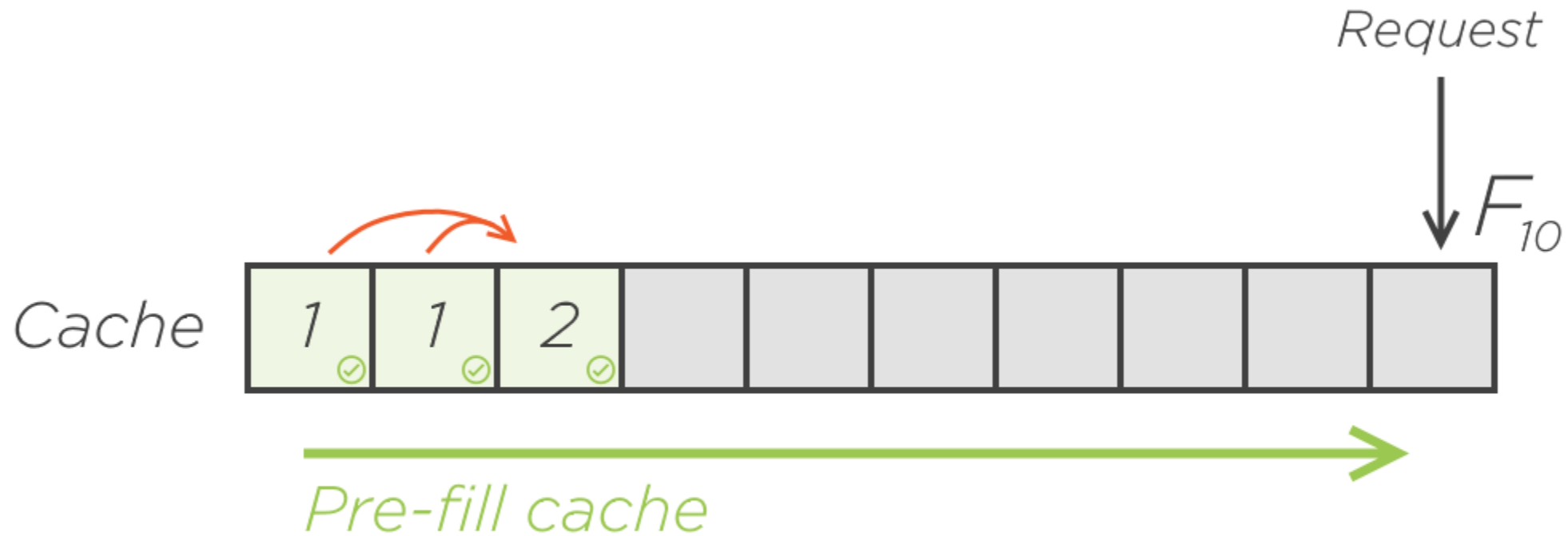
# Dynamic Programming



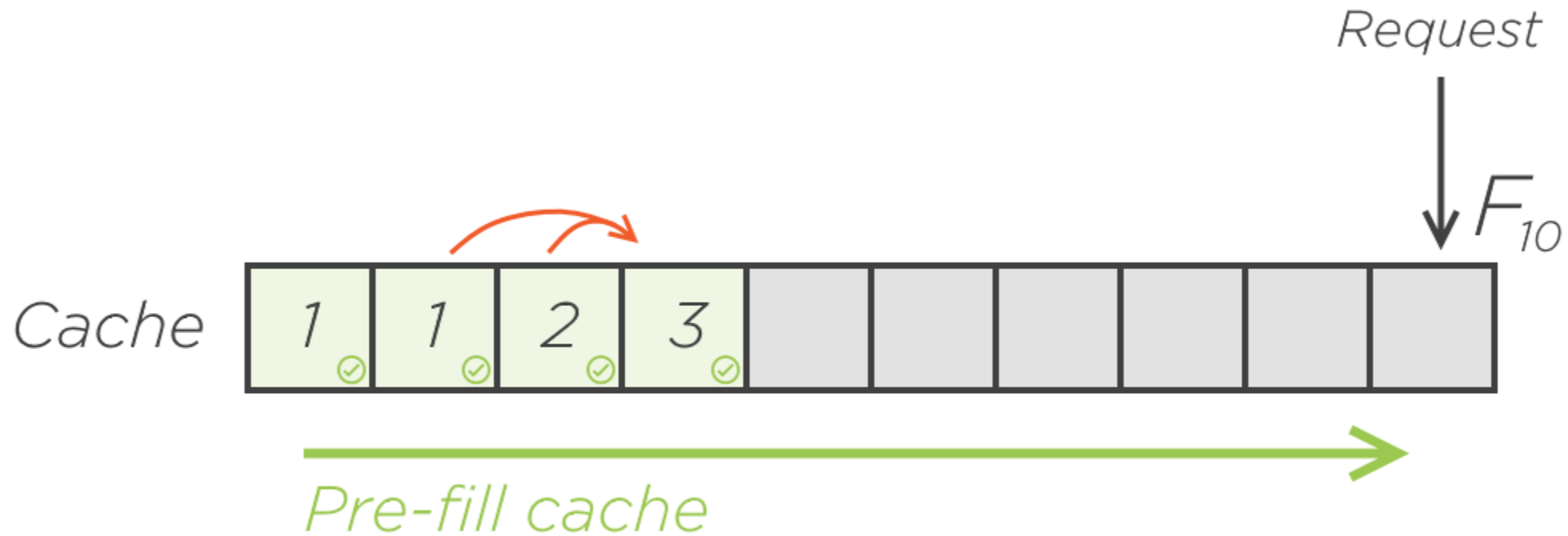
# Dynamic Programming



# Dynamic Programming

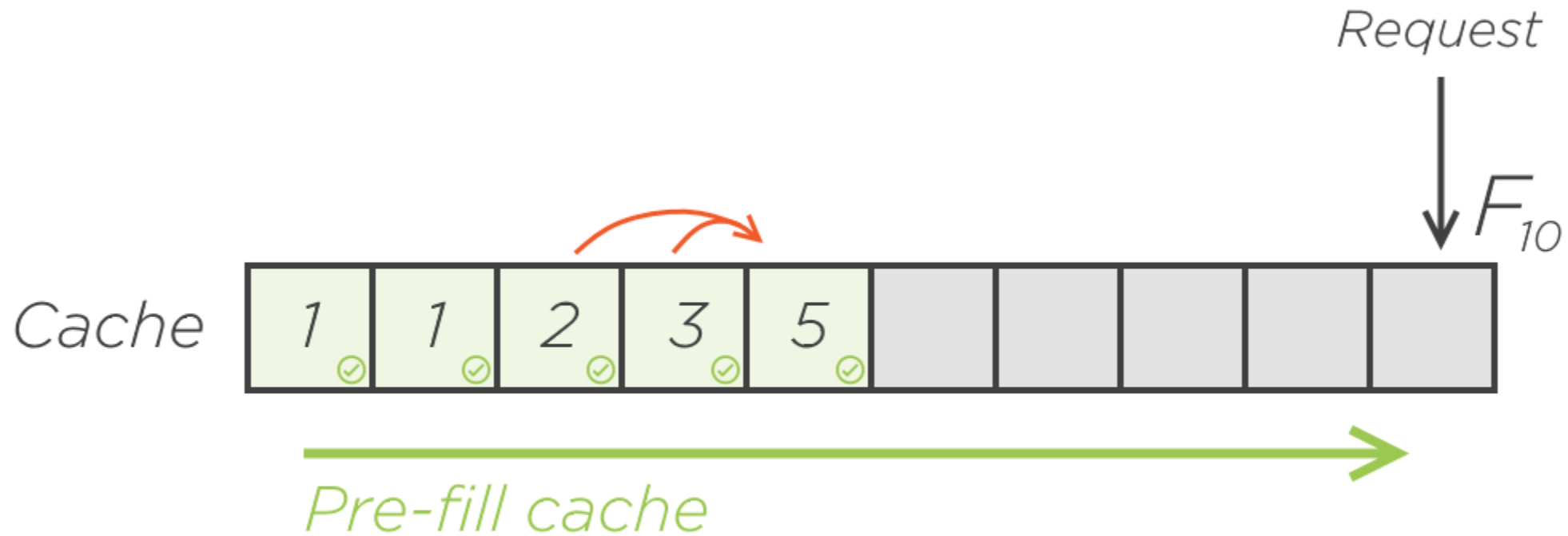


# Dynamic Programming

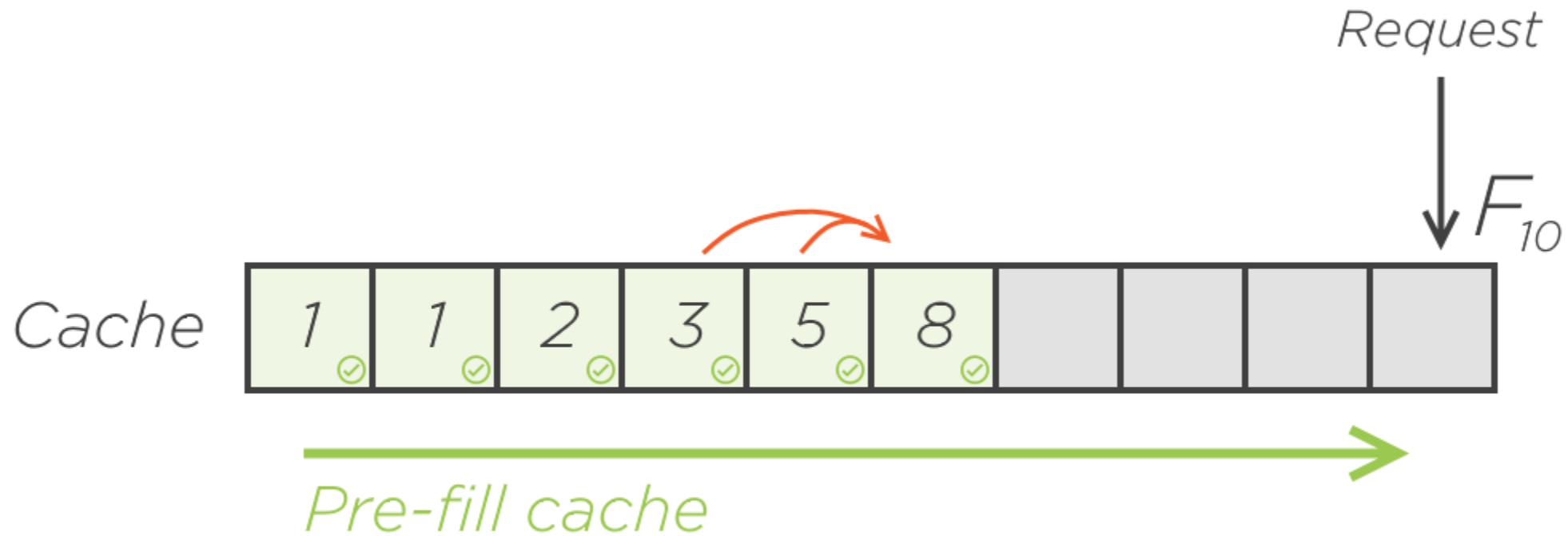




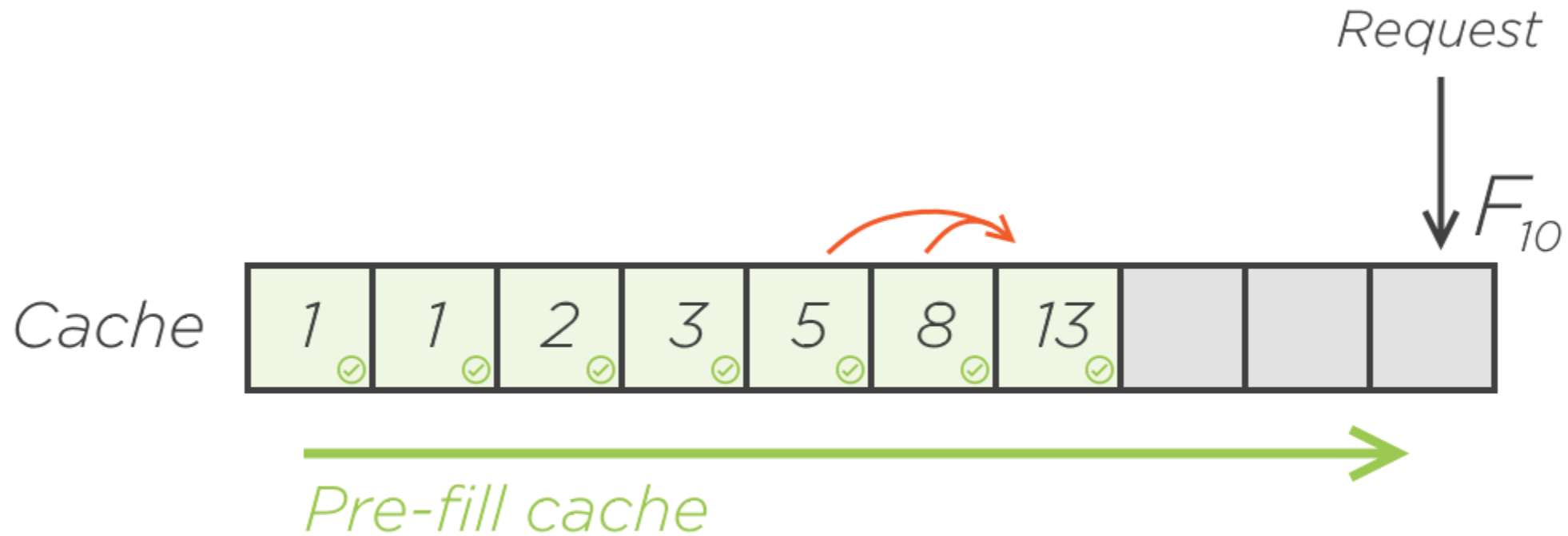
# Dynamic Programming



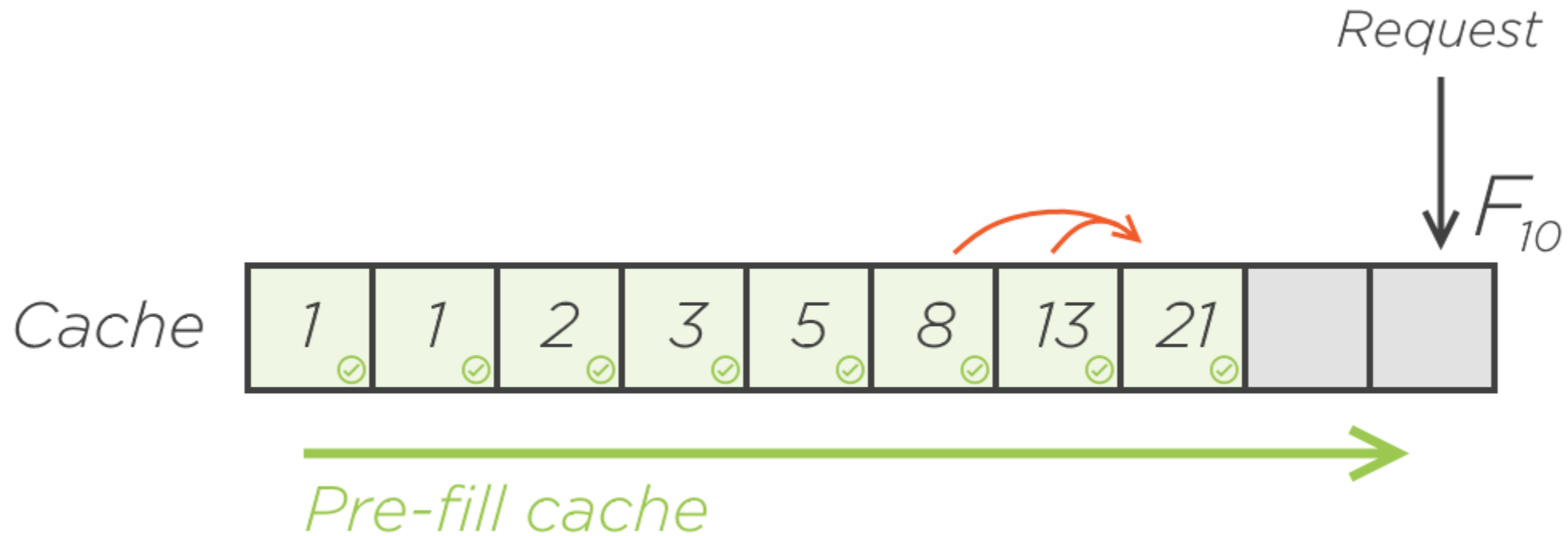
# Dynamic Programming



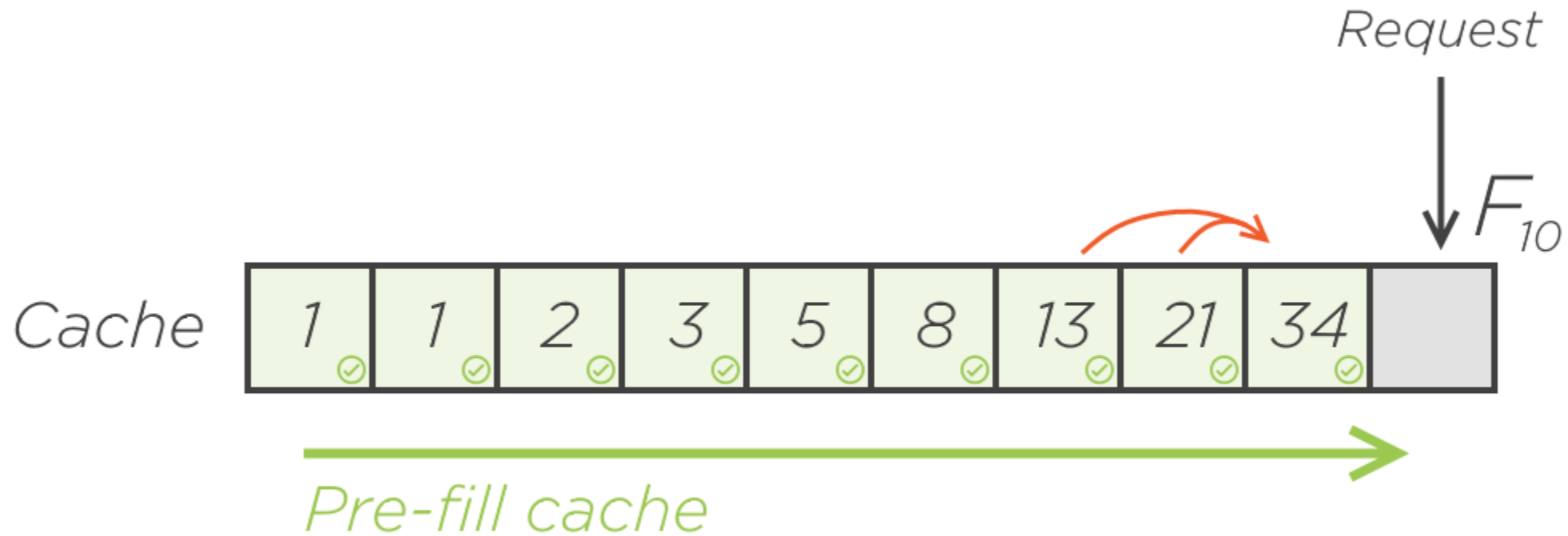
# Dynamic Programming



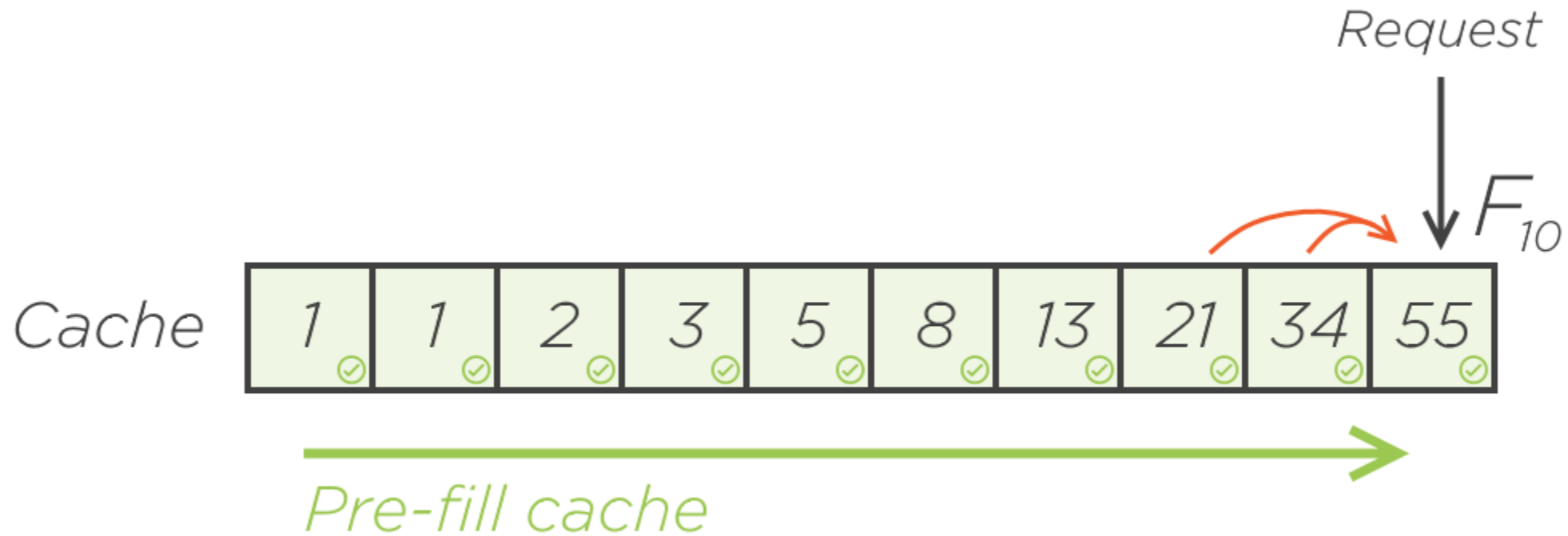
# Dynamic Programming



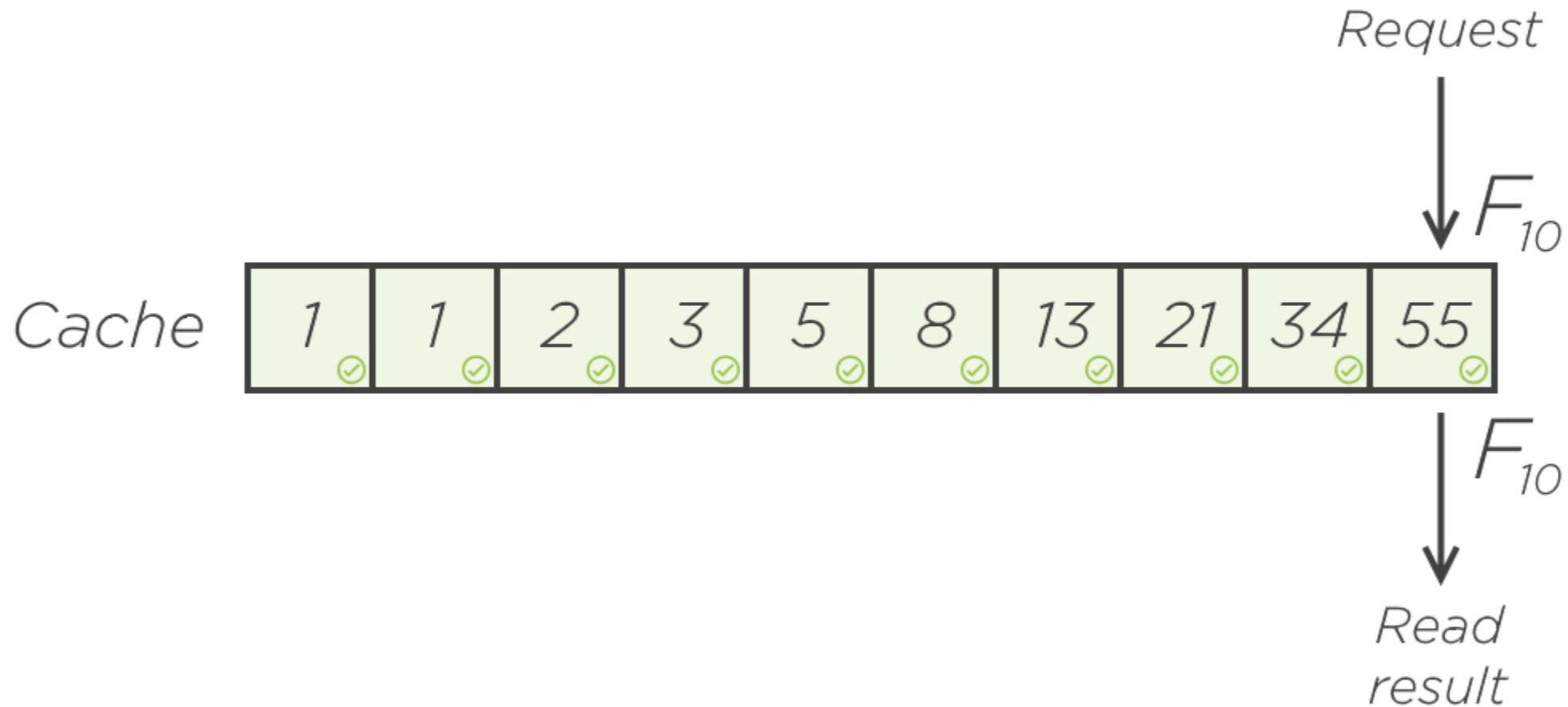
# Dynamic Programming



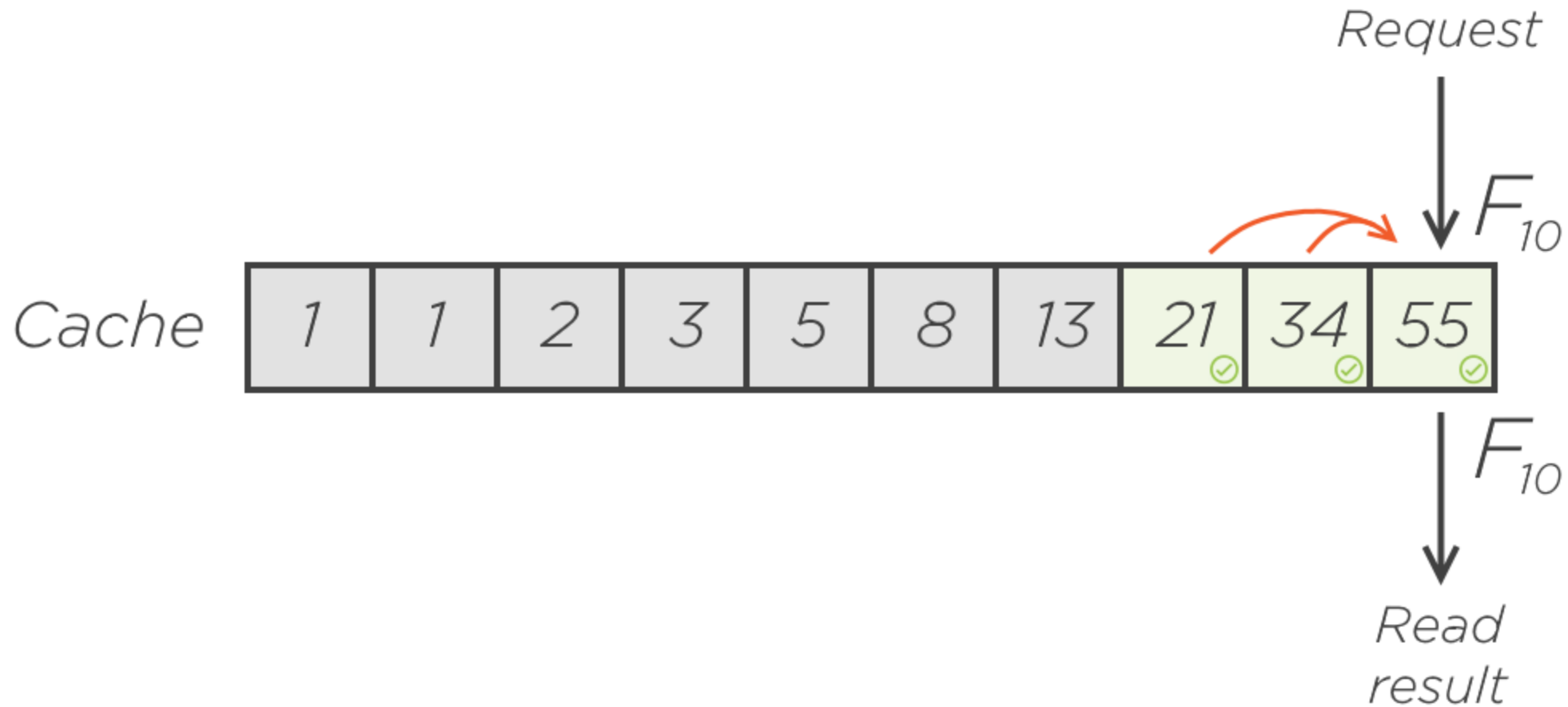
# Dynamic Programming



# Dynamic Programming



# Inventing an Algorithm





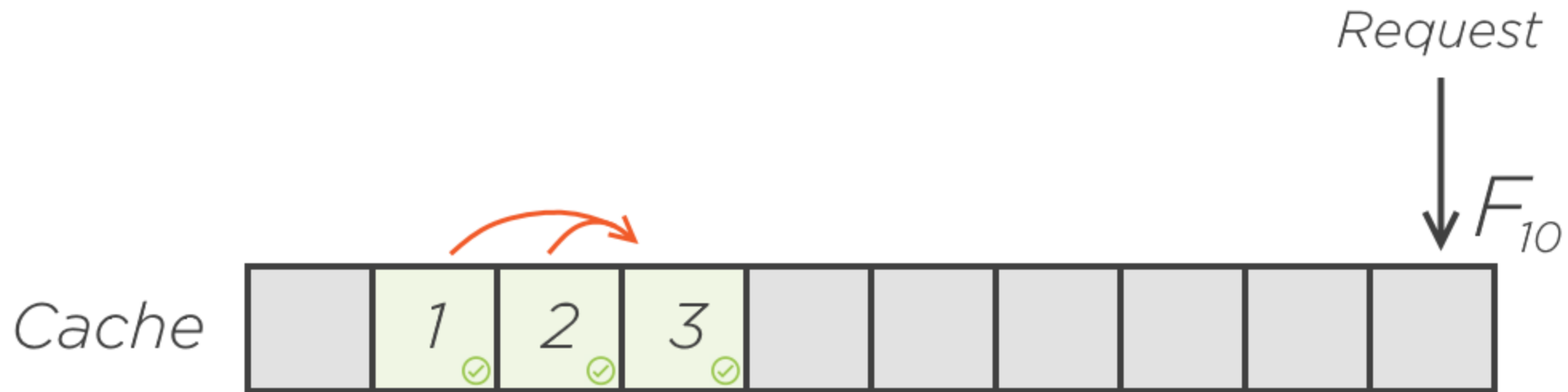
# Inventing an Algorithm



# Inventing an Algorithm



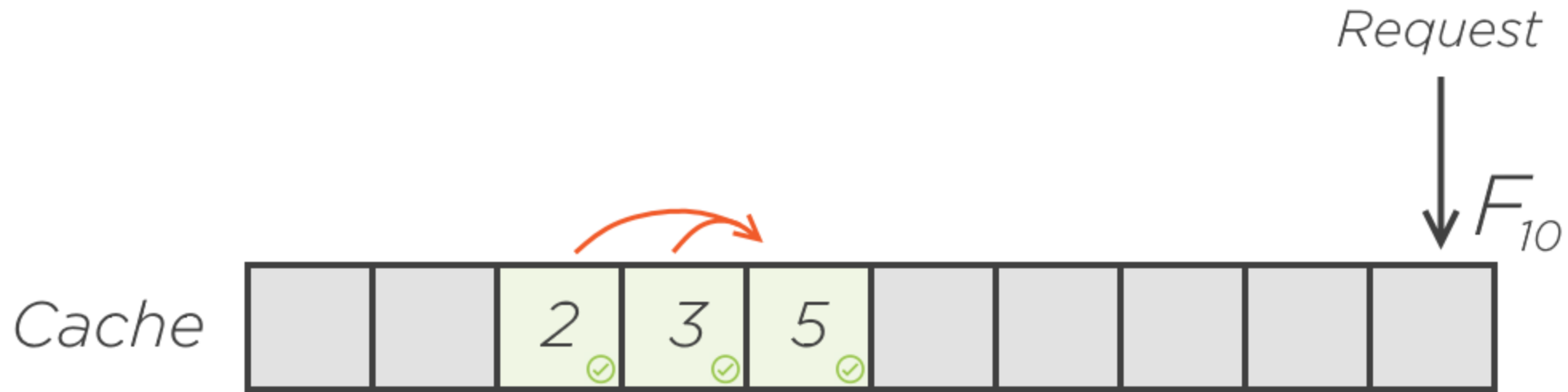
# Inventing an Algorithm



# Inventing an Algorithm



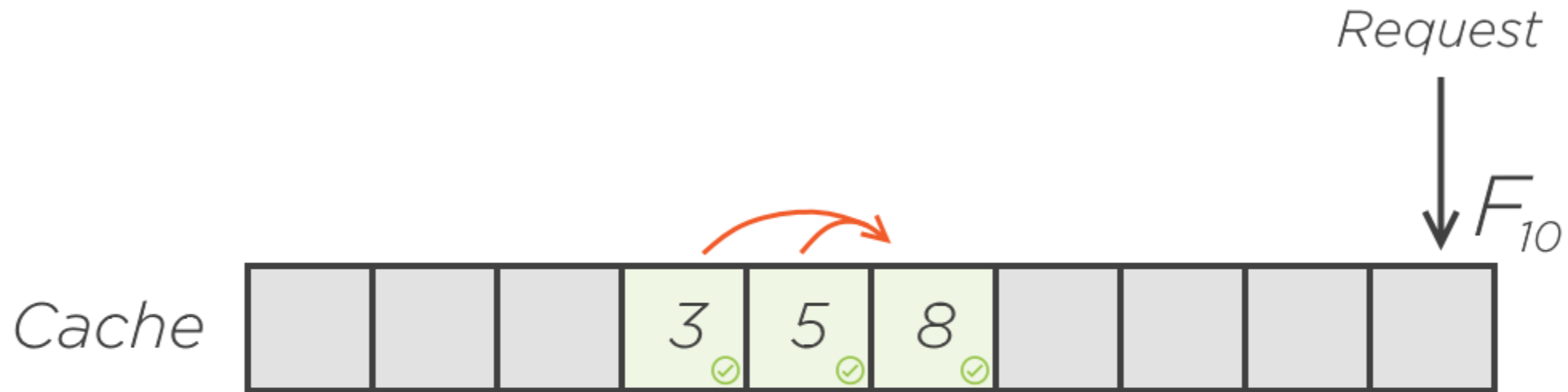
# Inventing an Algorithm



# Inventing an Algorithm



# Inventing an Algorithm

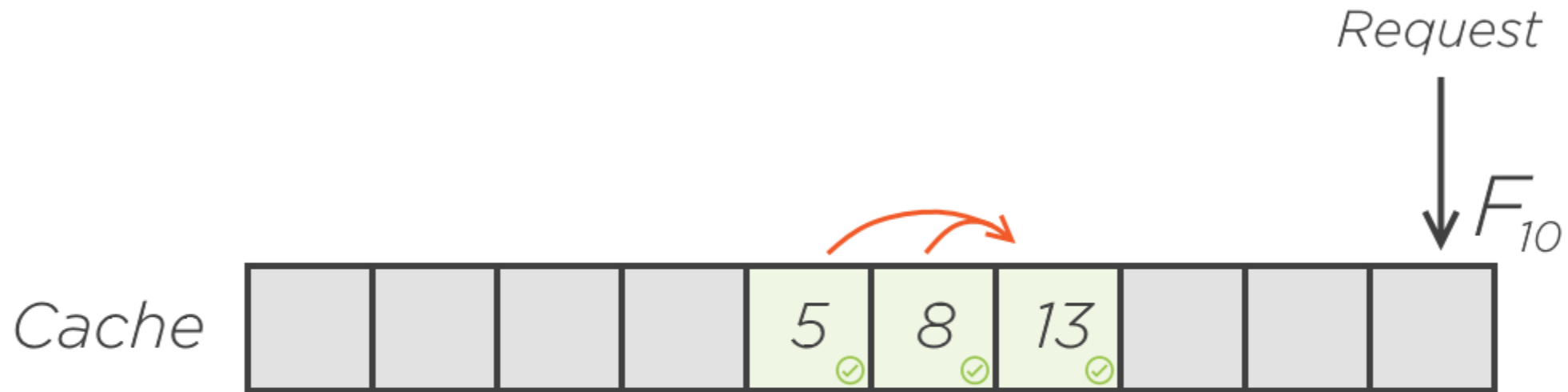


# Inventing an Algorithm





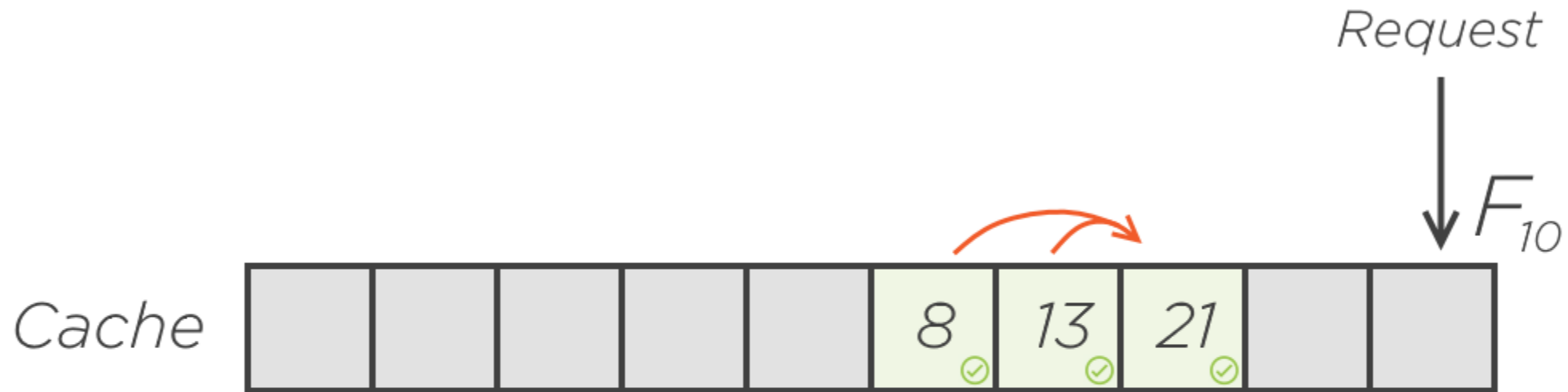
# Inventing an Algorithm



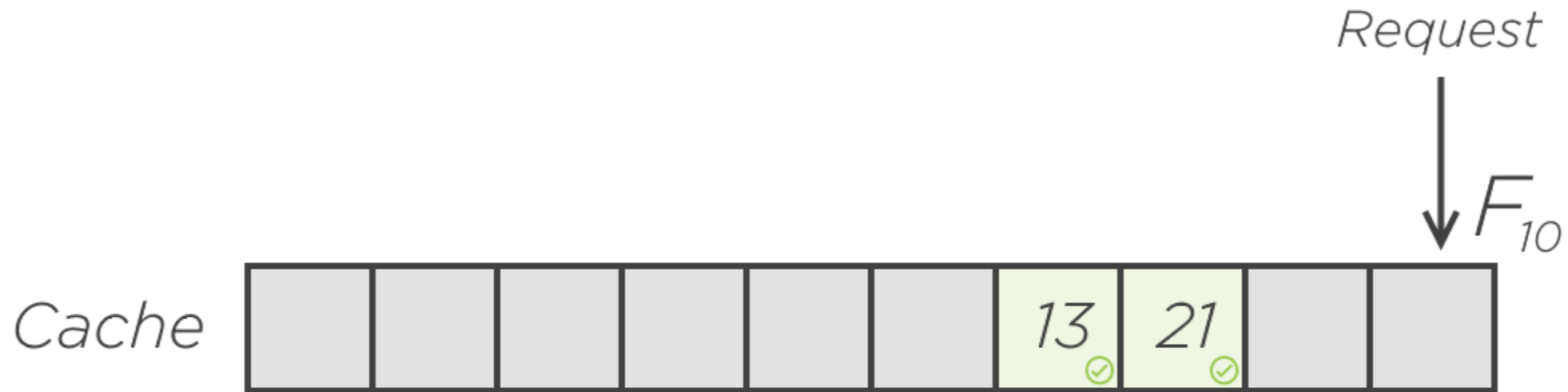
# Inventing an Algorithm



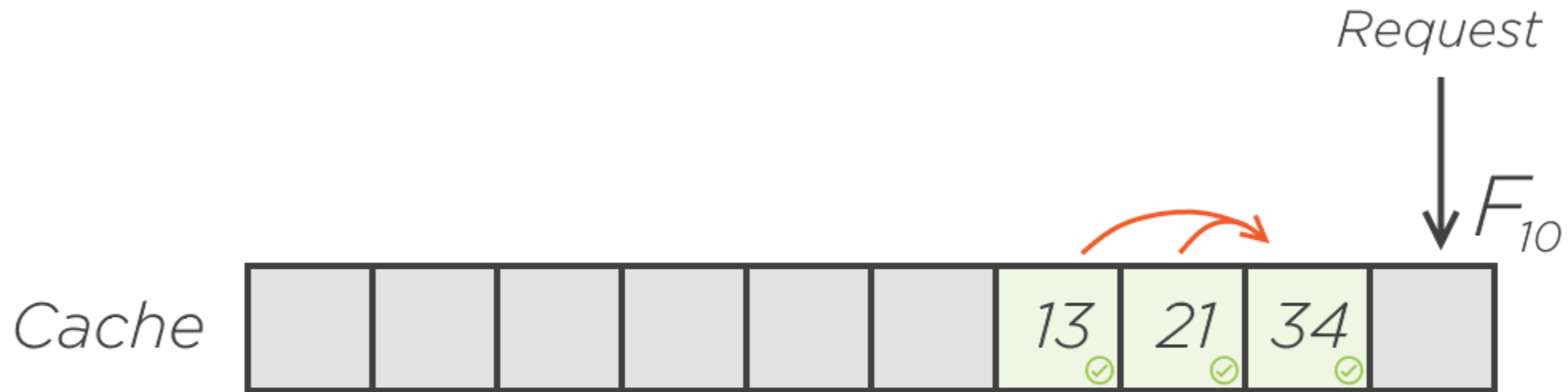
# Inventing an Algorithm



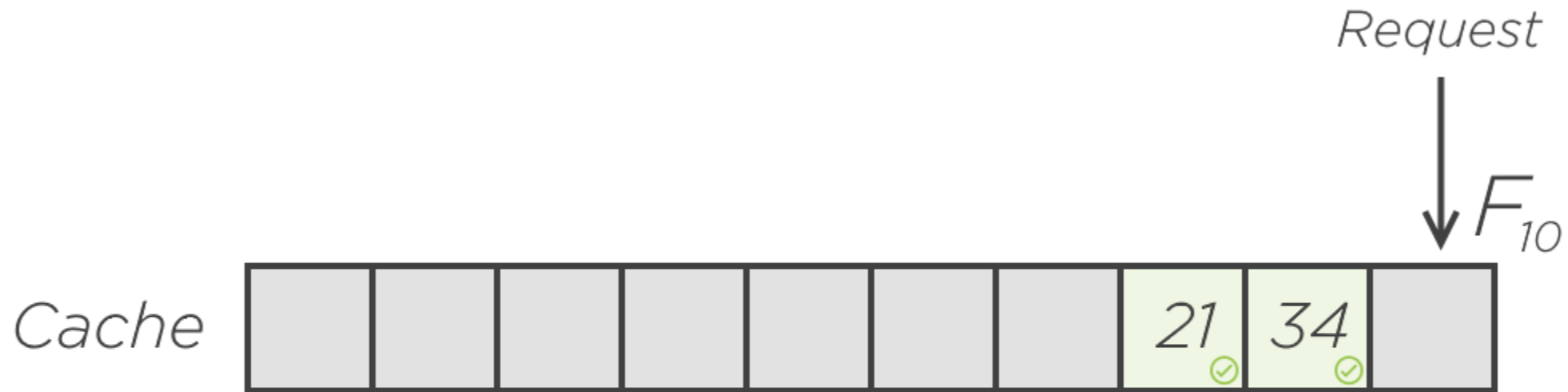
# Inventing an Algorithm



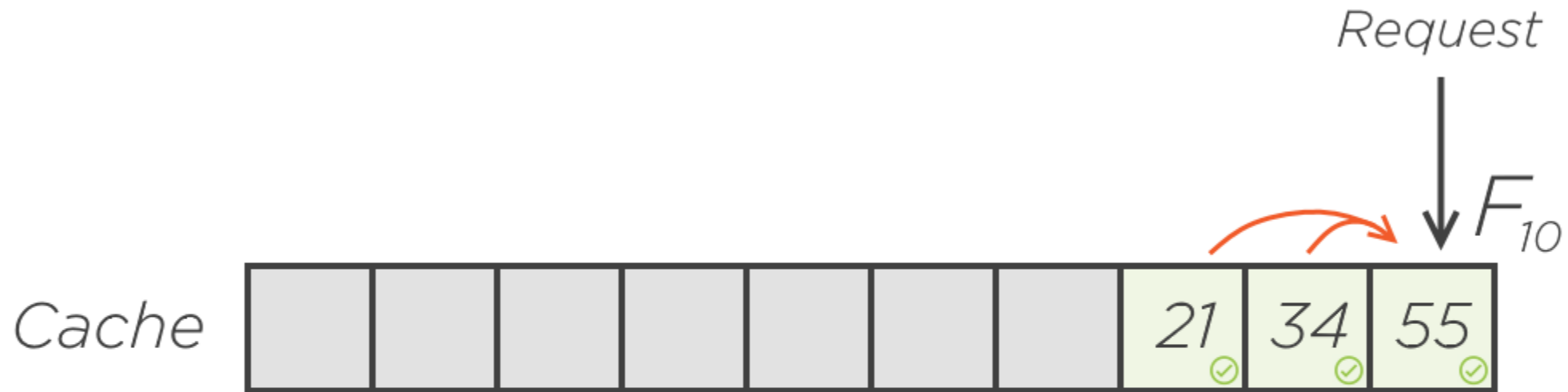
# Inventing an Algorithm



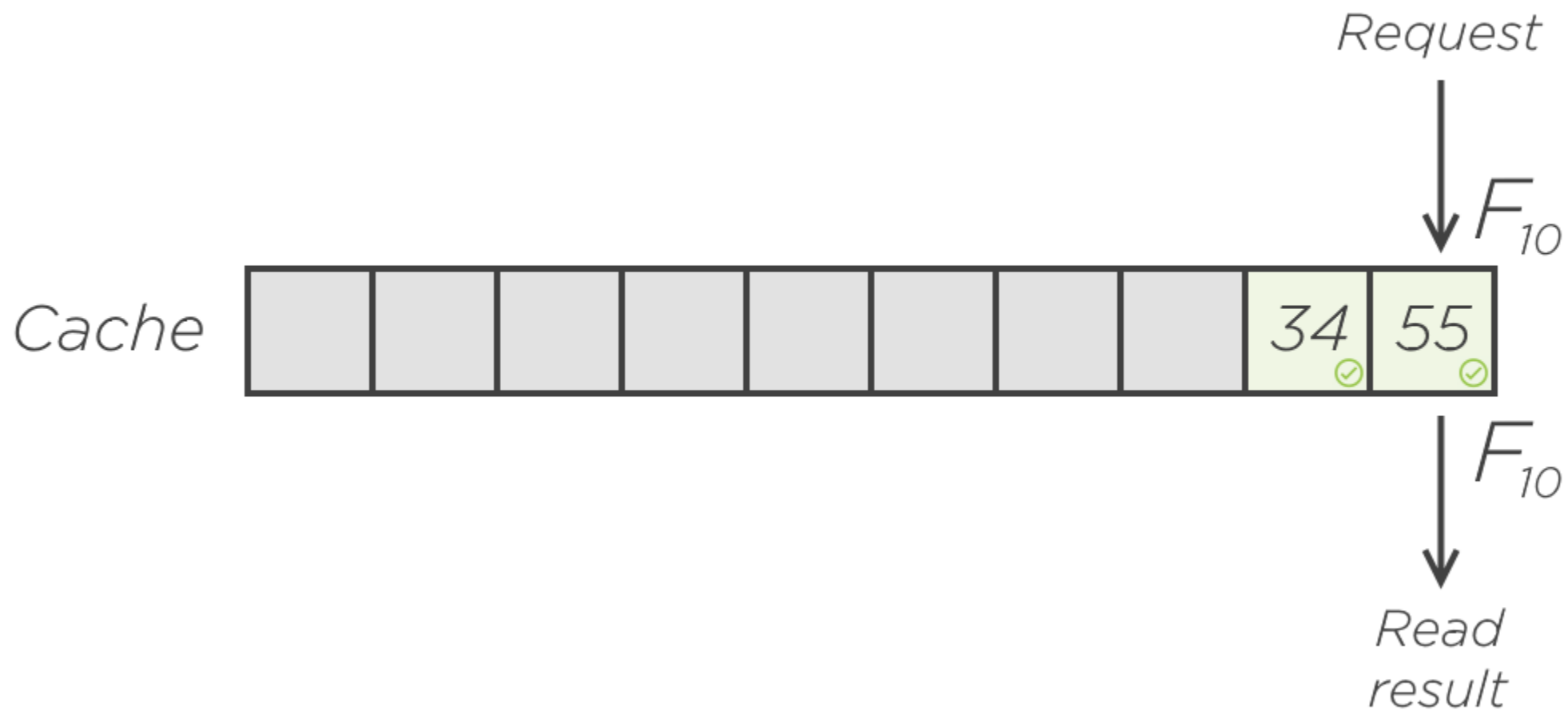
# Inventing an Algorithm



# Inventing an Algorithm



# Inventing an Algorithm





# Leveraging Memoization



Only applies to pure functions



Makes sense when there is noticeable performance drop

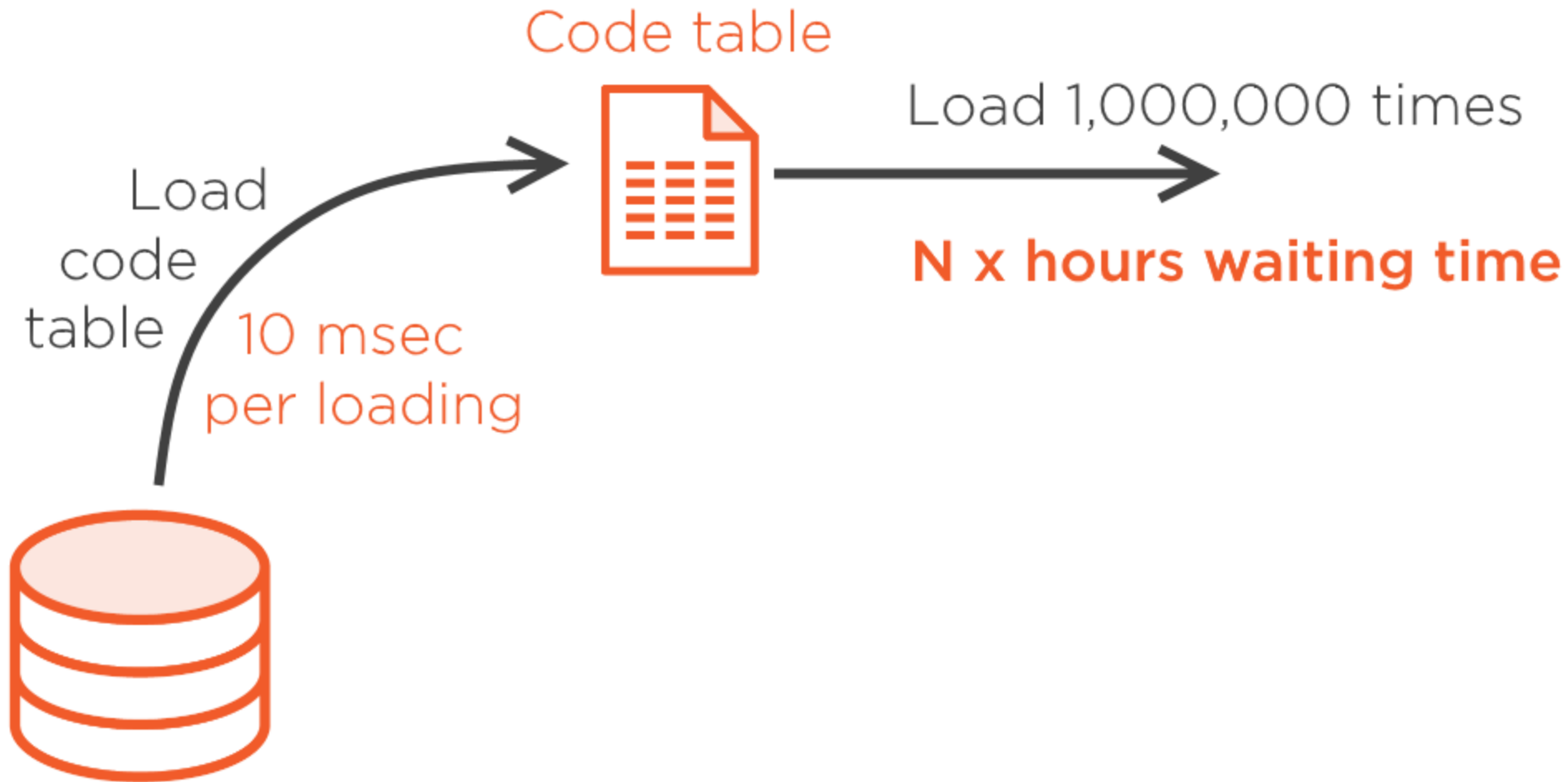


Trades time (CPU, I/O, etc.) for memory

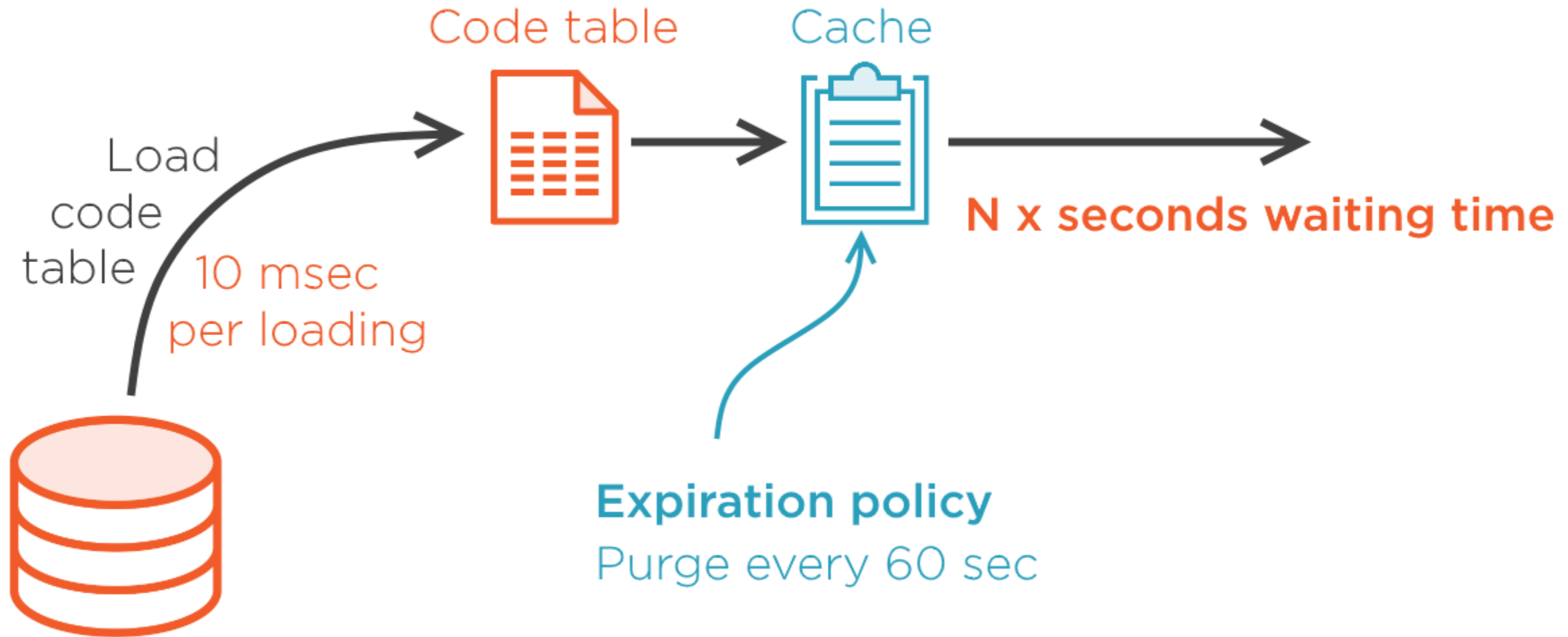
Don't forget to measure effects



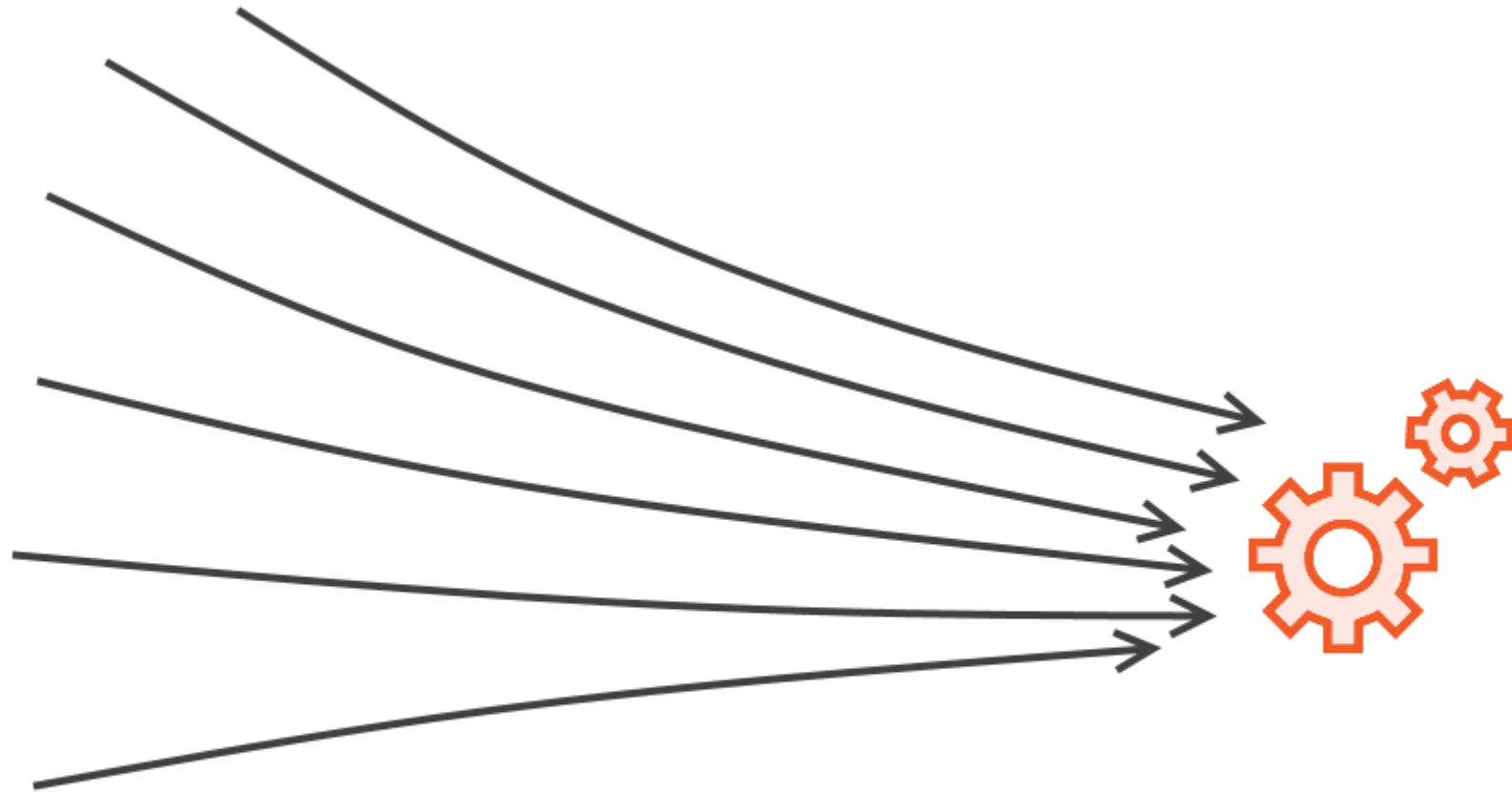
# Leveraging Memoization



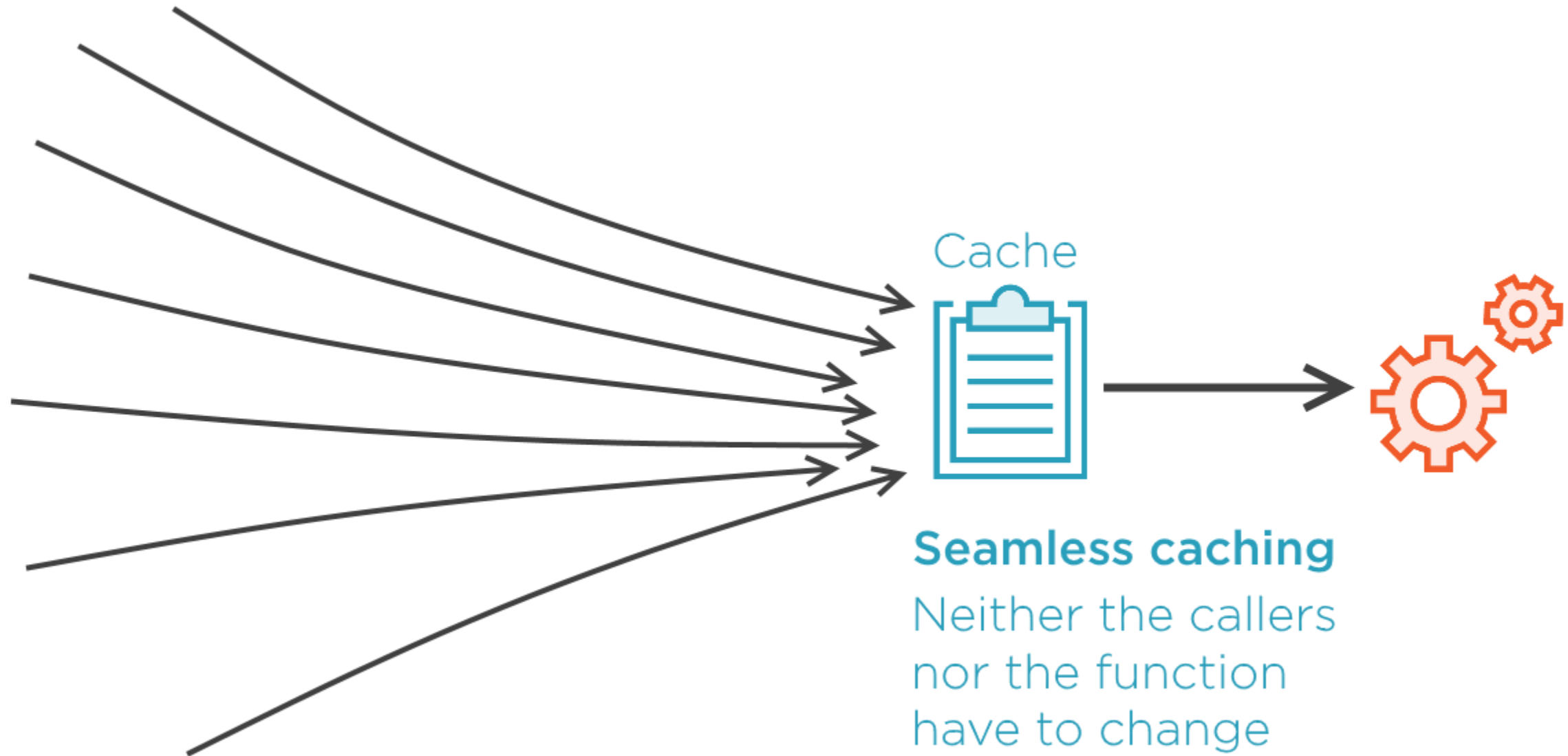
# Leveraging Memoization



# Leveraging Memoization



# Leveraging Memoization



# Summary



## Memoization of pure functions

- Special case of Dynamic Programming
- Every subproblem evaluated once
- Improves run time performance

## General memoization

- Wrap a function and cache its results
- PostSharp defines caching attribute
- Becomes orthogonal concern

**Next module:**

Working with Pure Member Functions

