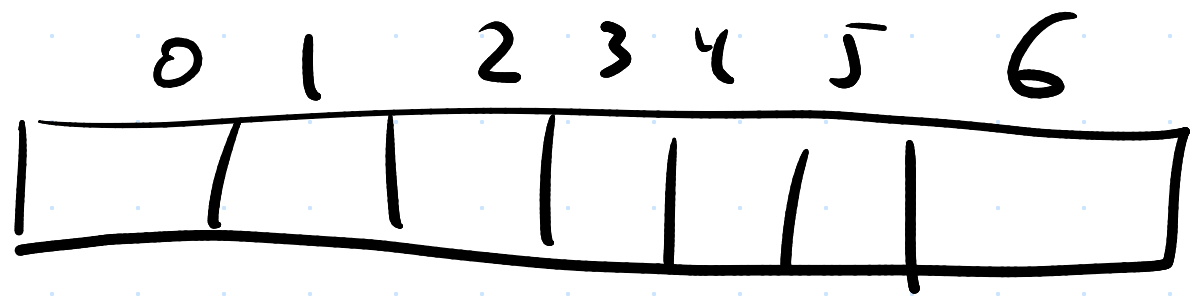
Linked chains (lists) as an alternative implentation technique

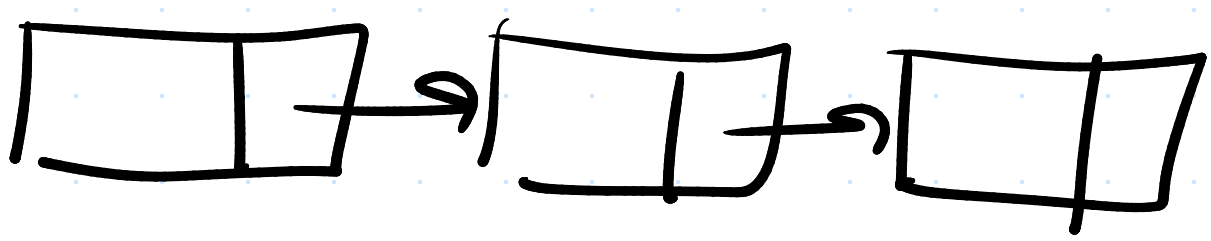
Today: stack

Later: the world!

arrays (Kotlin lists)



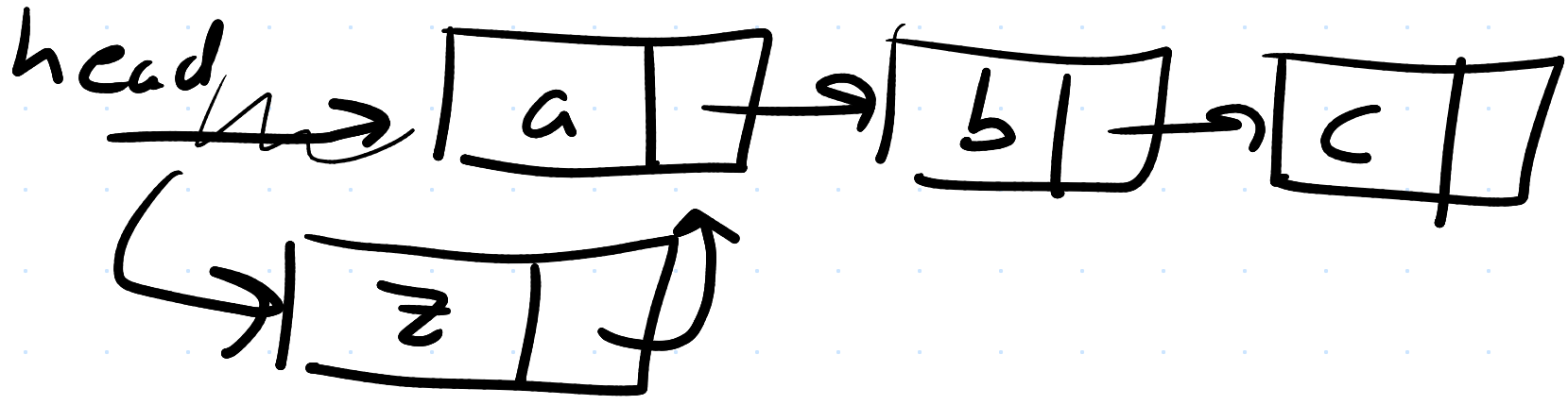
linked chain



- key advantage: you can insert at the beginning really quickly

For a Kotlin array / list to insert at beginning everything has to get copied down O(n)

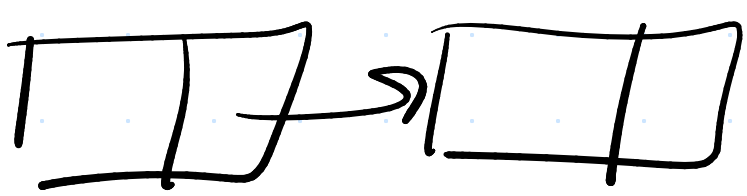
For a linked chain: O(1)

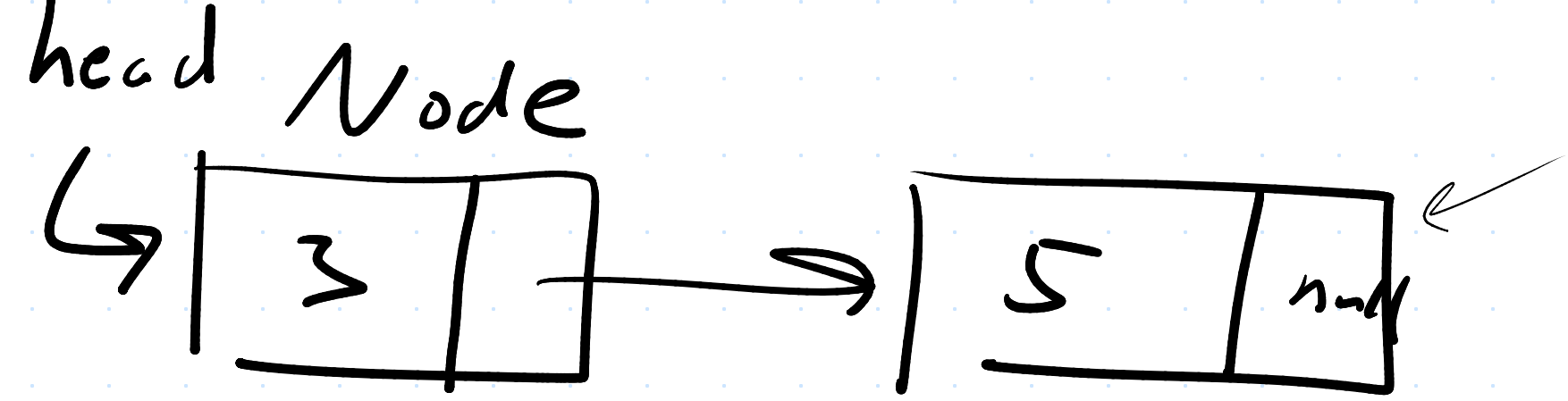


Downside of linked chains:
- you can't get to a particular item without walking down the list to get there

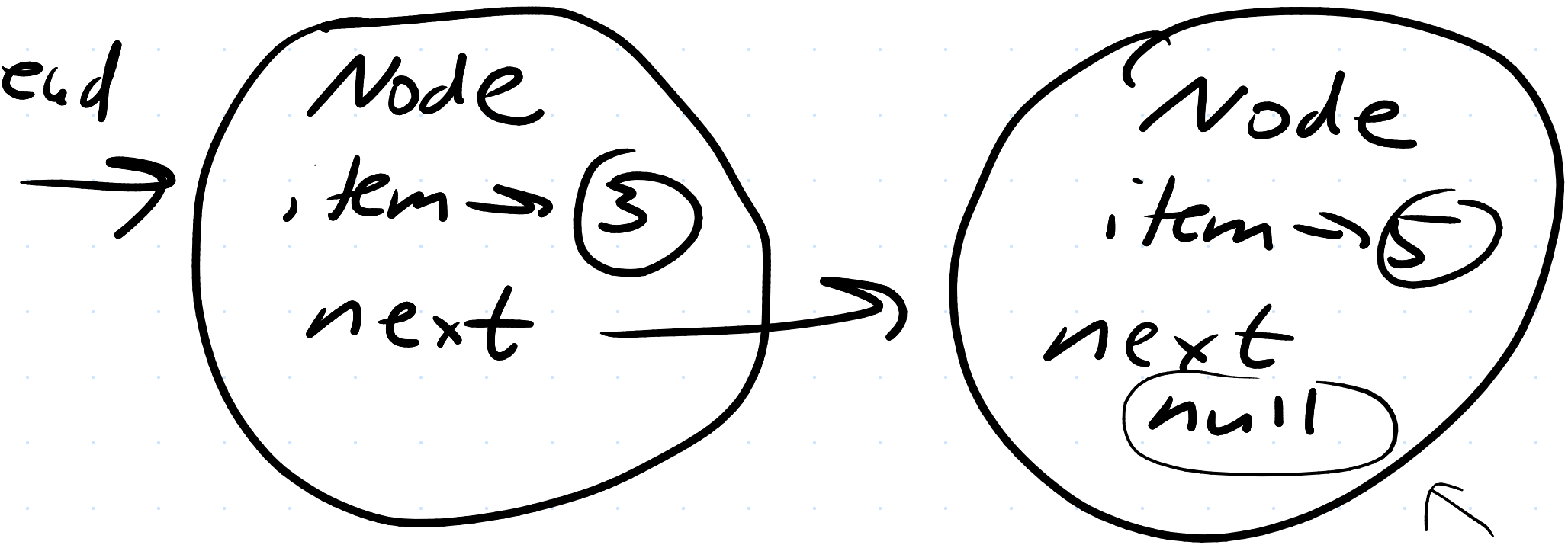
"Get item number 100"
array/kotlin list: O(1)
linked chain: O(n)

In a chain a node





really an object



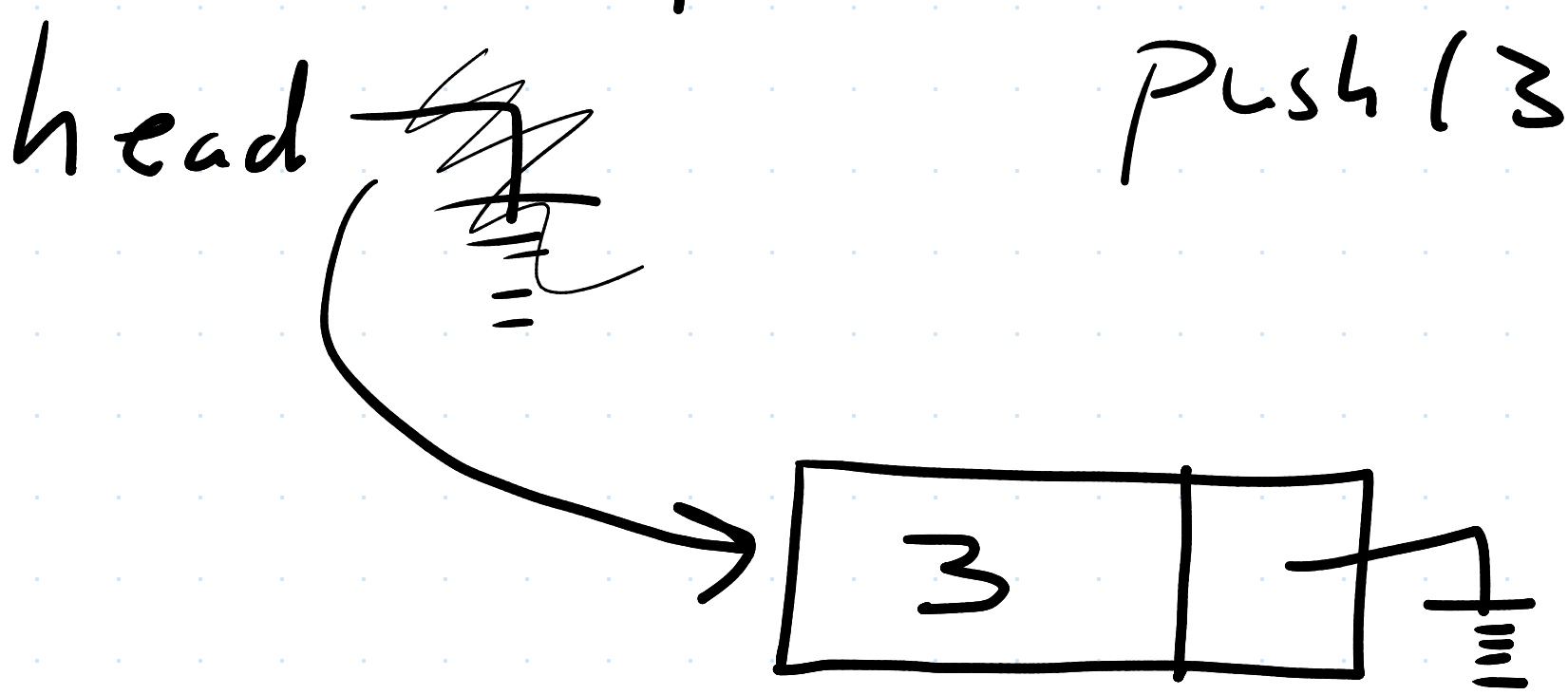
---

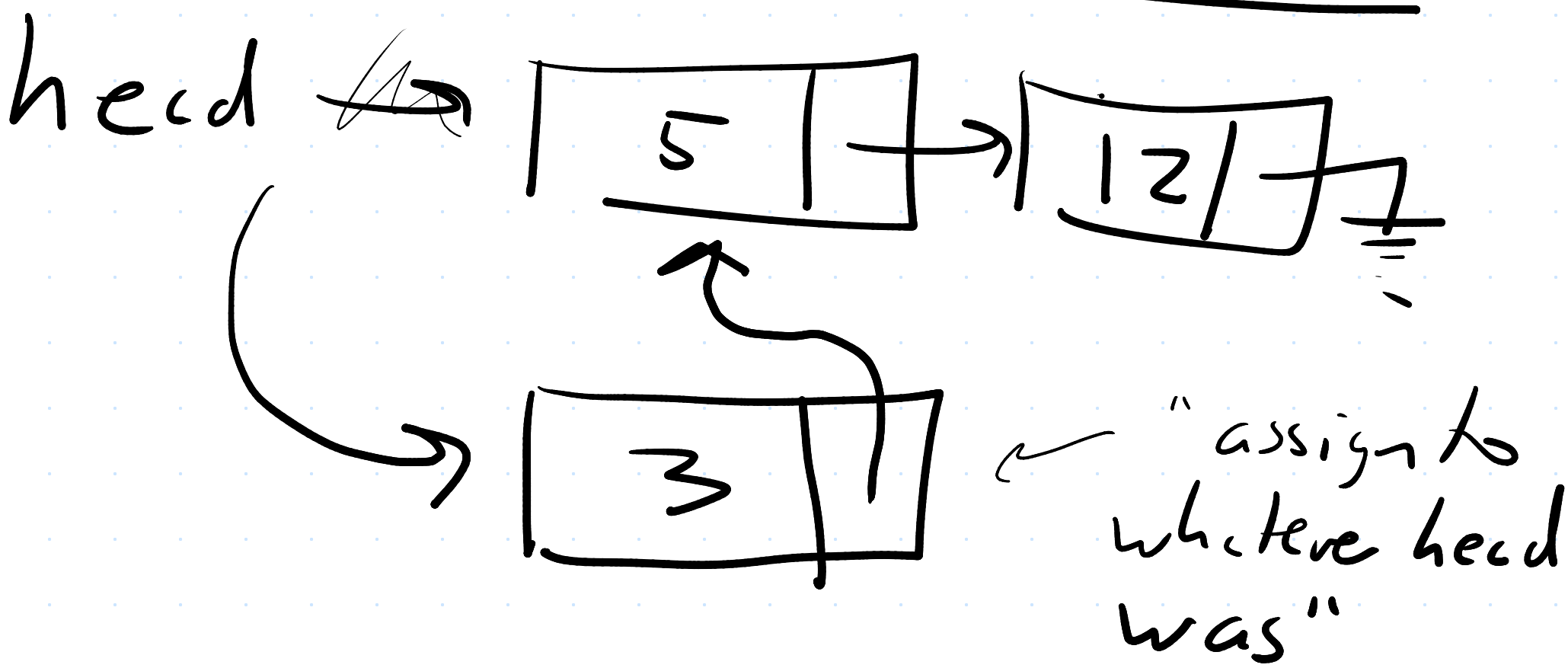
empty case

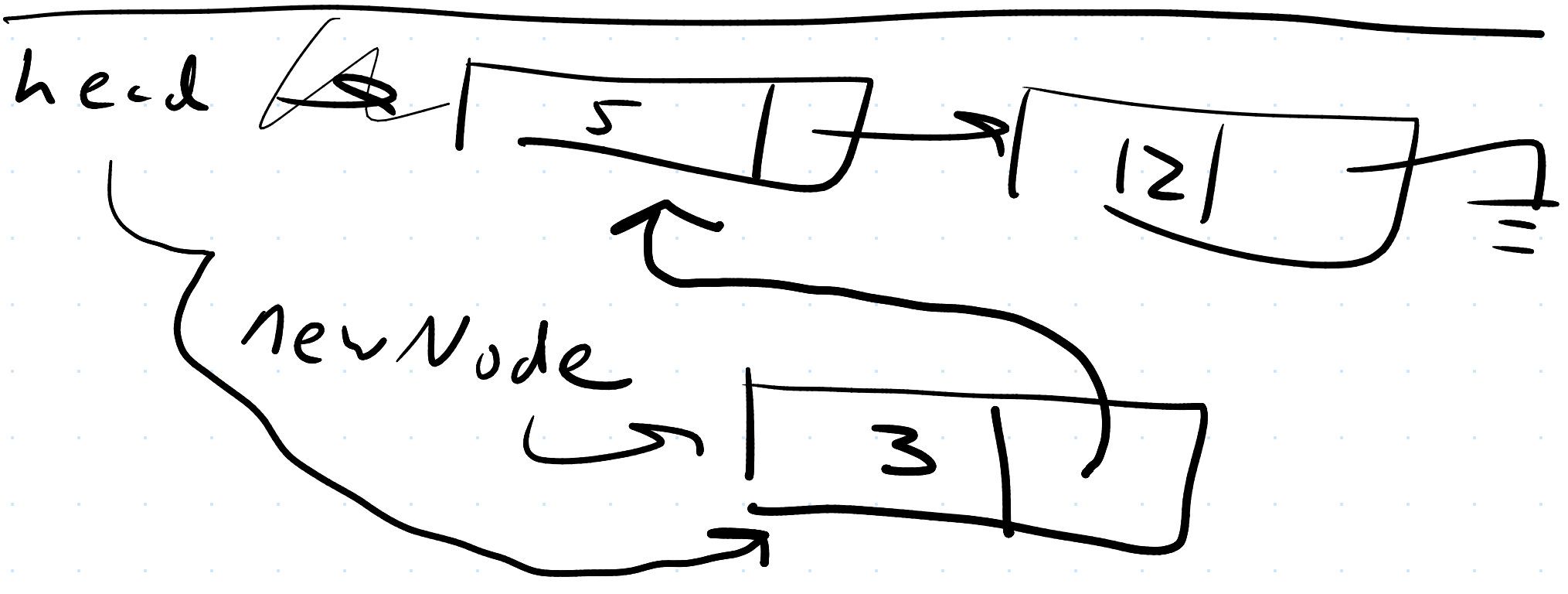
head ⏚

we will use the first item as the top of the stack

What does push look like?







A detour on terminology

ADT vs implementation

a description of a data type (e.g. a Stack), without how you actually implement it

e.g. A Stack is an ADT
a abstract data type

"A stack is a data structure w/ following operations:
push: put an item on top
....."

An ADT says nothing about big-O

An implementation is actual code.
We just built a Stack implementation in Kotlin

ADT: Stack
Implementations: we built three of them
ArrayStack.kt
ListStack.kt
LinkedStack.kt

Dave's term: Implementation strategy

"A stack as a linked chain"

ADTs: Stack | LunaLander

Implementation strategies: array list, linked chain (list)

arrayth.t
gets expanded and copied

Implementations: ~.kt, ~.kt, ~.kt | LunarLander.kt

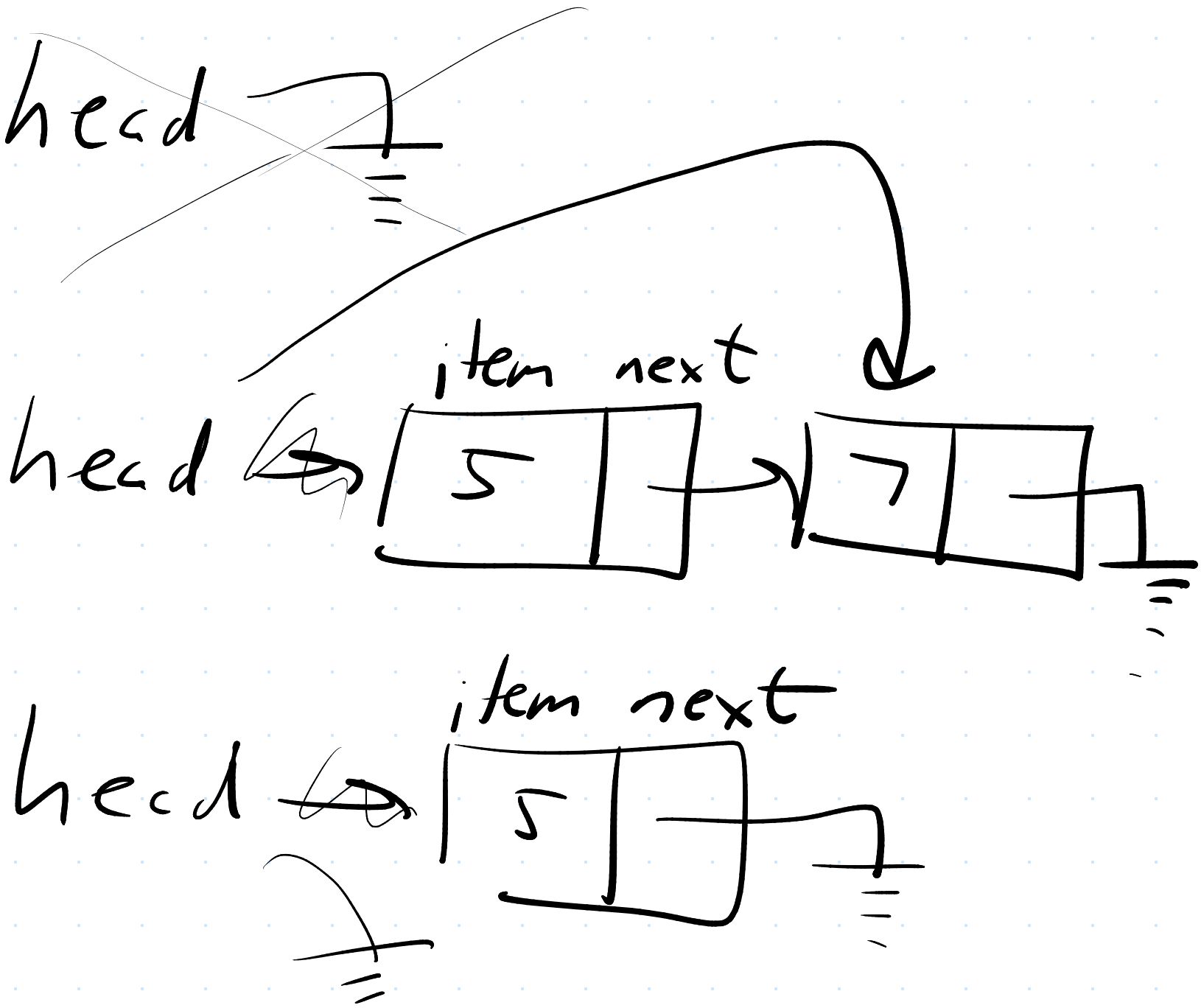
instance variables and some calculations

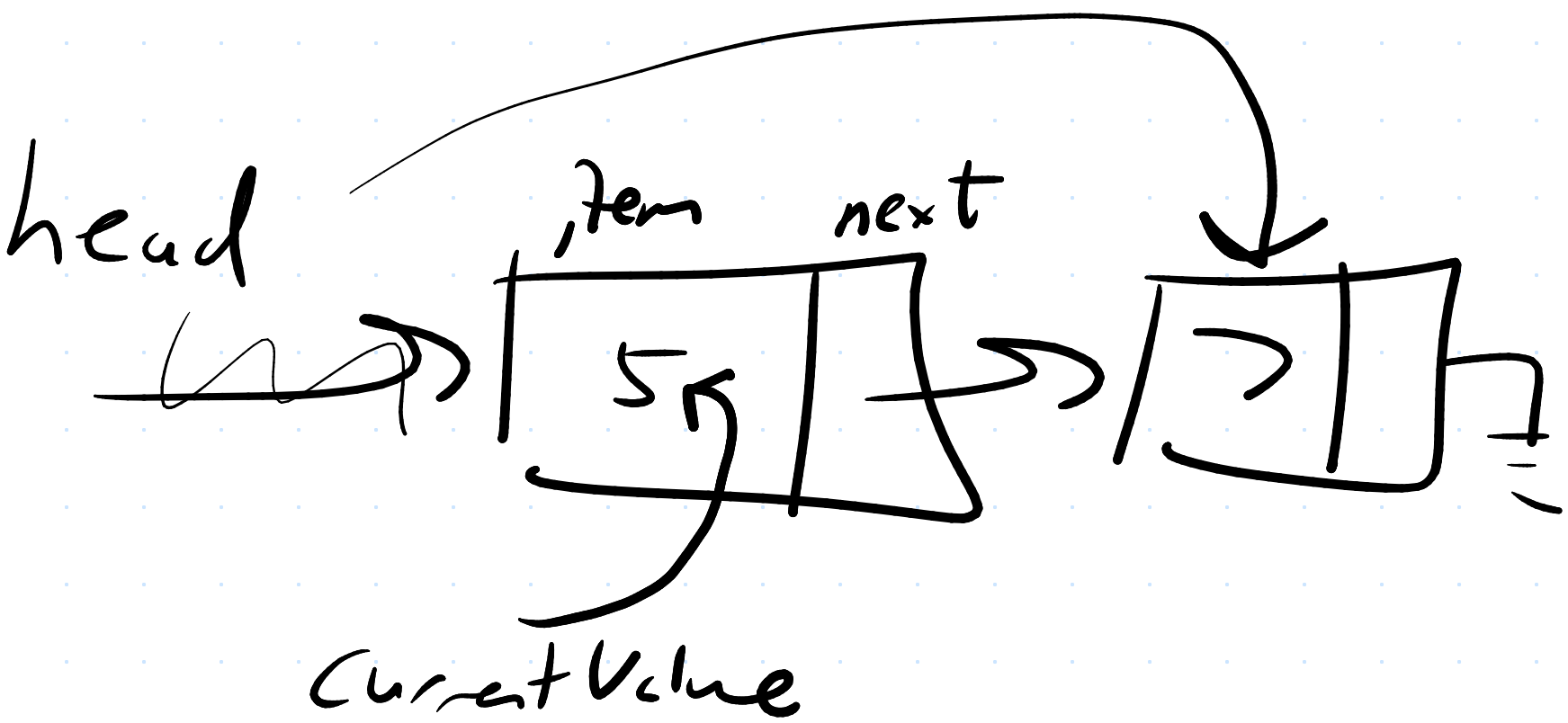
The word list is a disaster.
It is used, depeding on
context to mean

- an ADT
- an impl strategy ← linked (linke
- an implemetation
- a thing in Kotlin/Python

chain
d list)

— kotlin list
implemented
via an
array inside
it


head
item
next
head
5
7
item
next
head
5


head
item
next
5
7
CurrentValue