Agenda

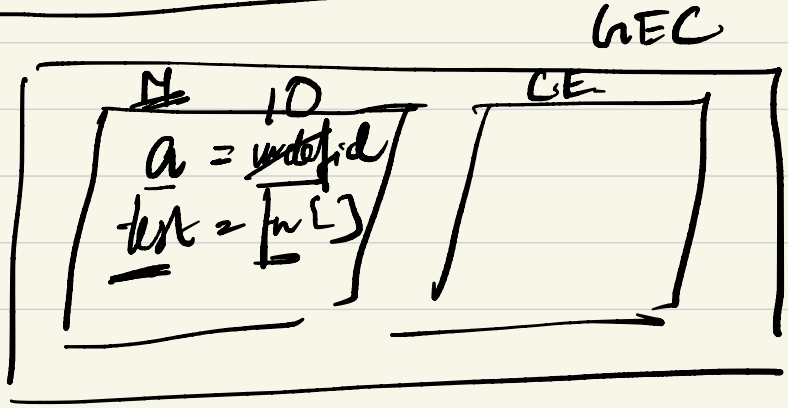
1. Recap of EEC
2. Scope Chaining & Lexical Scope
3. Closures

Recap

1. Execution Context
   - Global EC
   - Function EC
2. Hoisting → initial value undefined
3. Temporal Dead Zone

```
console.log(a)
var a = 10
test()
function test() {
    console.log("this is a test.");
}
test()
```

undefined ✓
this is a test ✓

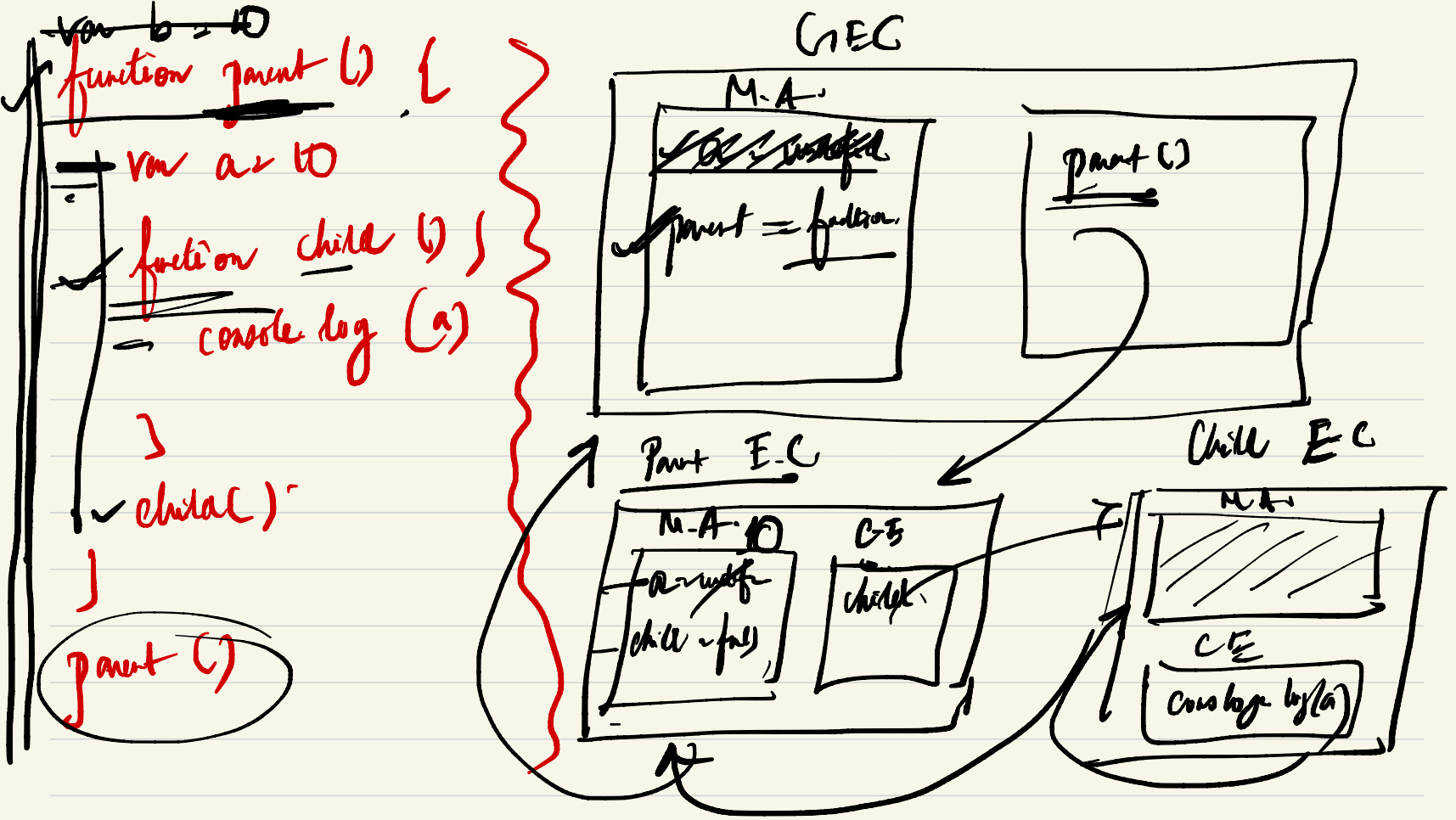



```
test()
const test = function() {
var
}
```

```
function parent() {
  var a = 10
  function child() {
    console.log(a)
  }
  child()
}
```

Lexical Scope (Confinement)

(Where it is defined)


var b = 10
function parent () {
var a = 10
function child () {
console.log (a)
}
child()
}
parent ()
GEC
M.A
parent = function
parent ()
Parent E.C
M.A
a = undefined
child = full
C.E
child
Child E.C
M.A
C.E
Console.log(a)

Lexical Scope

Lexical scope is the ability for a function scope to access variables from parent scope (or ancestor).

We say that the child function is lexically bound to that of the parent.

When function keyword is used,

I will be able to access it before even defining it,

BECAUSE the entire function gets stored in the M.M phase.

Temporal Dead Zone.

For let & const, we cannot access the value before we declare it. (Although, it is stored as undefined)

Steps to create an execution context:

M.A

① Figure out what variables are declared in the current E.C

② If there is any function keyword, then store the value in M.A phase. (the entire function body)

③ If not, then assign memory location for all variables but store the value as undefined.

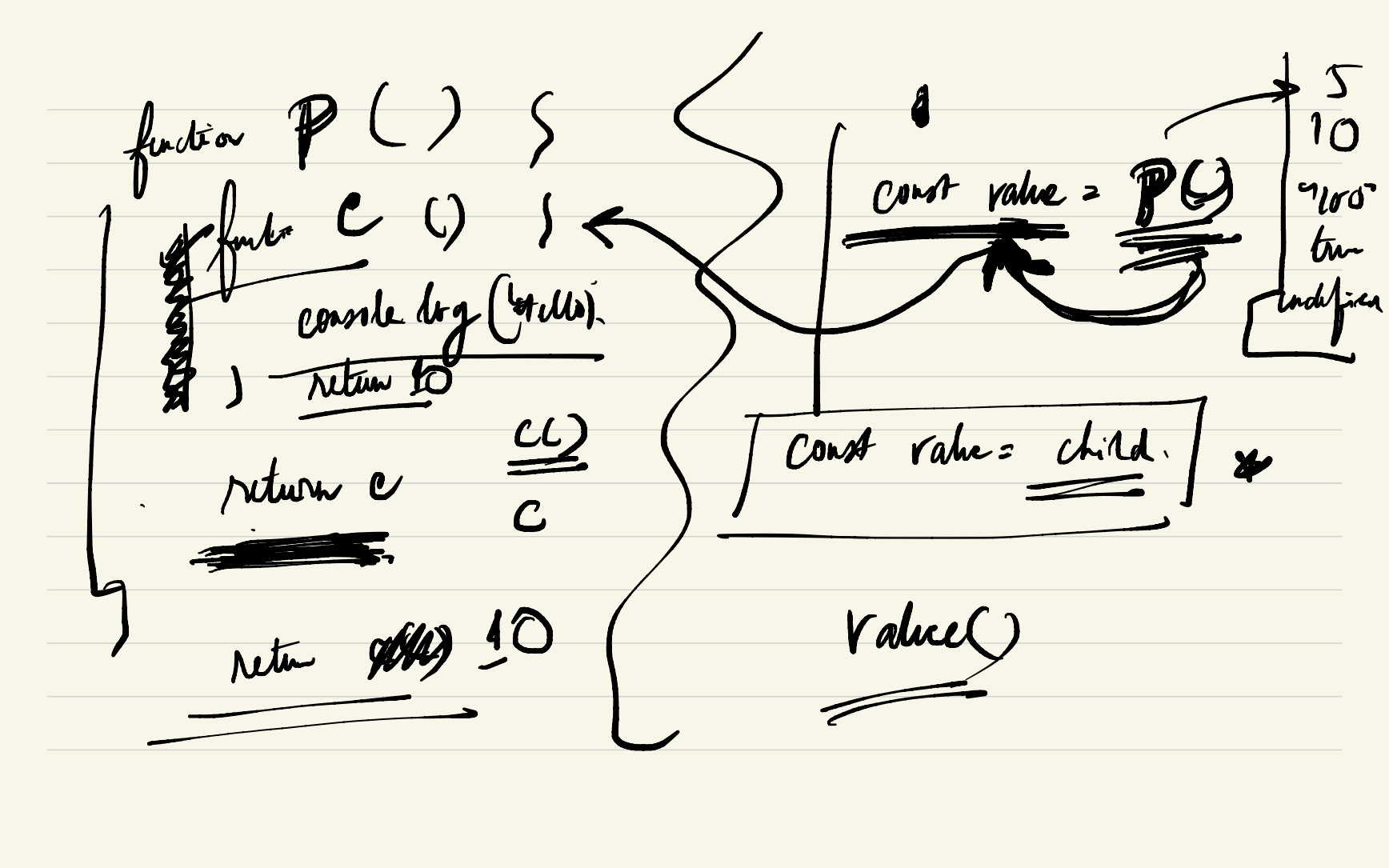
C.E.

0 When you're trying to access a variable —

a) Can you access this variable? → X

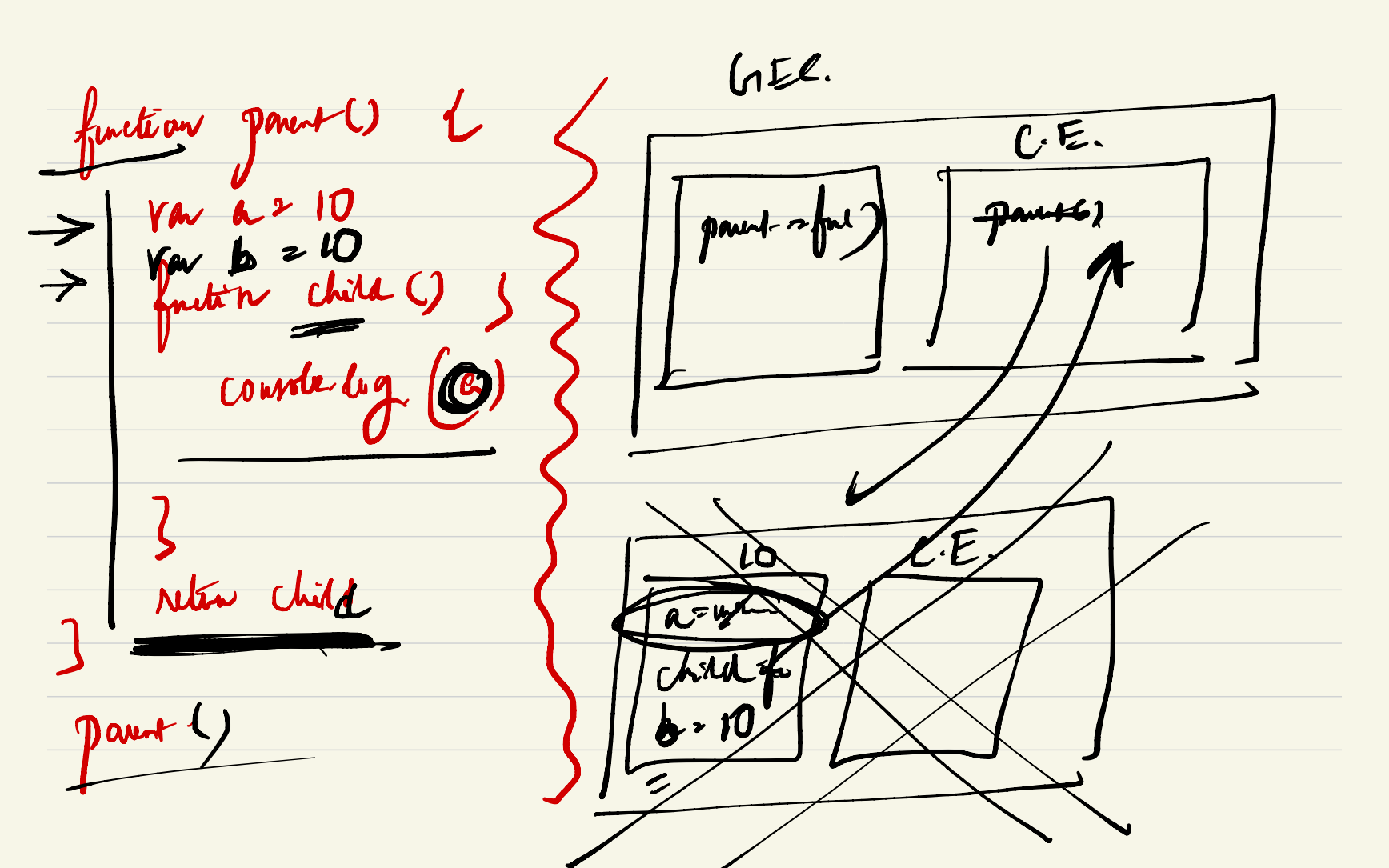
(If let & const — we can't access before initialization) • TDZ

b) If we can, then figure out what is the current value.


function P ( ) {
function C ( ) {
console.log ("Hello").
return 10
return C
C()
C
return 10
const value = P()
5
10
"100"
const value = child.
value()

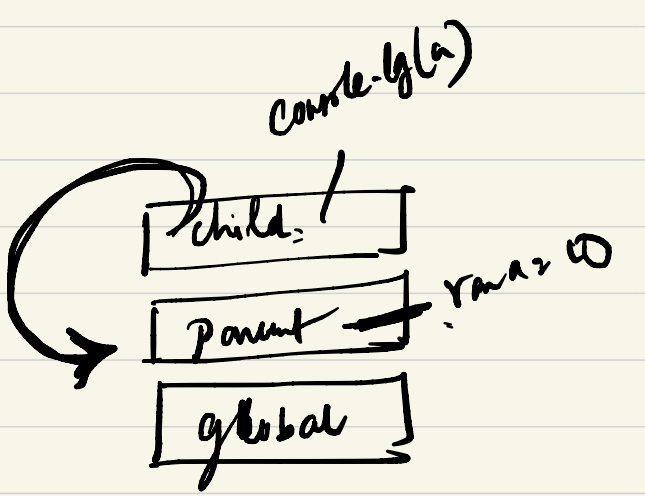
func() — execution

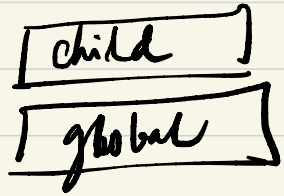
func → referring to the body.

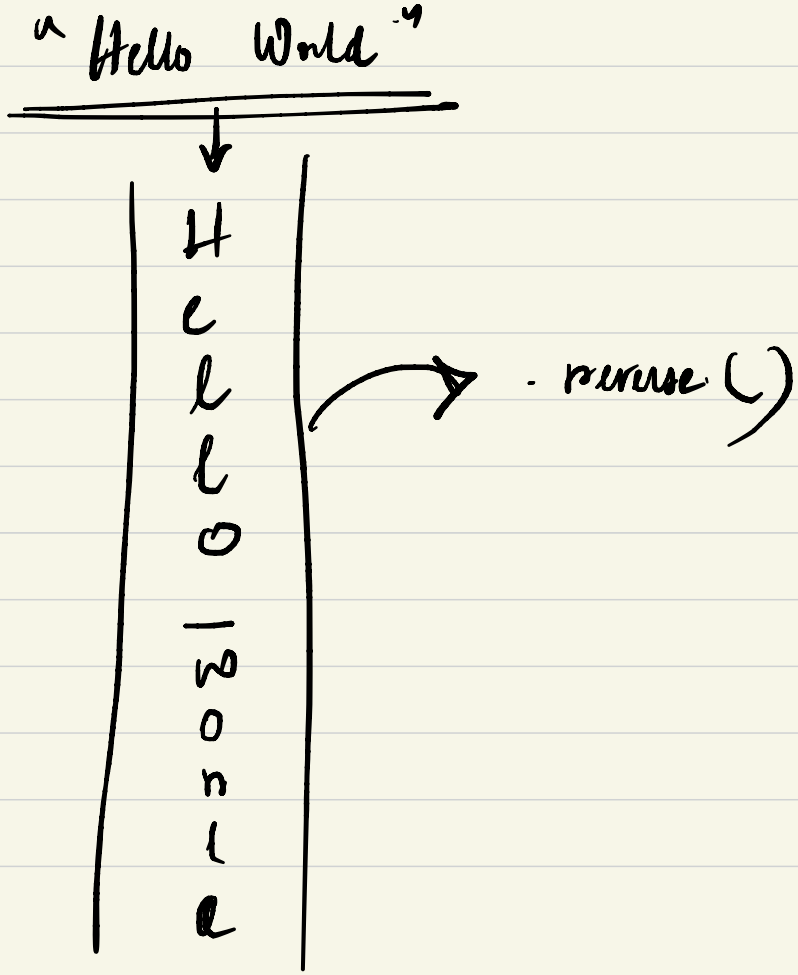

function parent() {
var a = 10
var b = 10
function child() {
console.log(a)
}
return child
}
parent()
GEC.
C.E.
parent()
parent()
LO
C.E.
a = 10
child fn
b = 10

Closure → A closure is a feature of JS using which a child function can have access to parent's lexical scope although the parent E.C has been destroyed.

Example 1






"Hello World"
H
e
l
l
o
_
w
o
r
l
d
.reverse()