

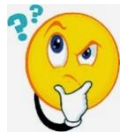
Рекурсії

Замикання

Навіщо потрібна рекурсія?



Навіщо потрібна рекурсія?



Задача: побудувати піраміду !



Навіщо потрібна рекурсія?



Задача: побудувати піраміду !



Навіщо потрібна рекурсія?



Задача: побудувати піраміду !



Це дуже
складно !!!



Навіщо потрібна рекурсія?



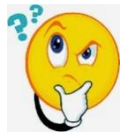
Задача: побудувати піраміду !



Це дуже
складно !!!



Навіщо потрібна рекурсія?



Задача: побудувати піраміду !



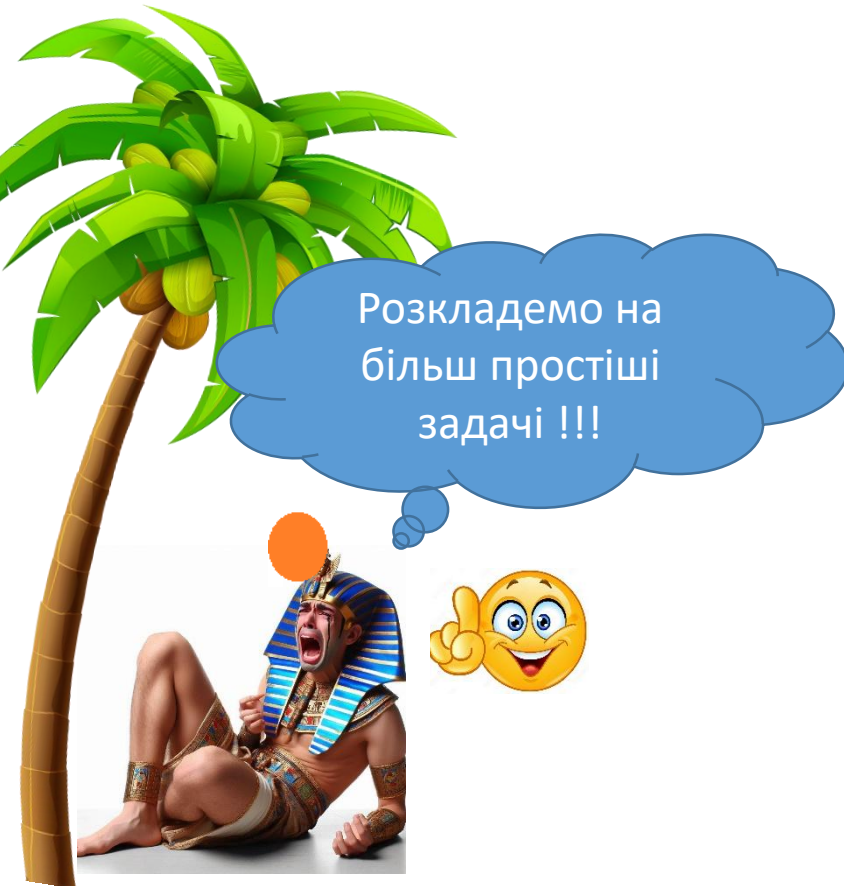
Це дуже
складно !!!



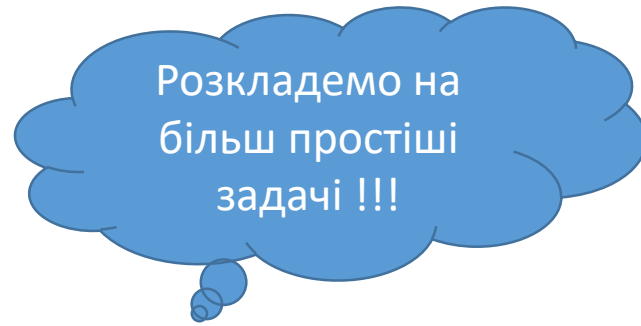
Навіщо потрібна рекурсія?



Задача: побудувати піраміду !



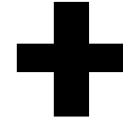
Навіщо потрібна рекурсія?



Побудувати піраміду з 4 поверхів!



Навіщо потрібна рекурсія?

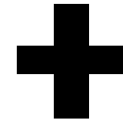


Побудувати піраміду з 4 поверхів!

Побудувати піраміду з 3 поверхів!



Навіщо потрібна рекурсія?



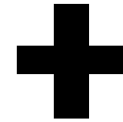
Побудувати піраміду з 4 поверхів!

Побудувати піраміду з 3 поверхів!



Побудувати піраміду з 2 поверхів!

Навіщо потрібна рекурсія?



Побудувати піраміду з 4 поверхів!

Побудувати піраміду з 3 поверхів!

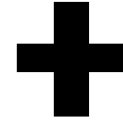


Побудувати піраміду з 2 поверхів!



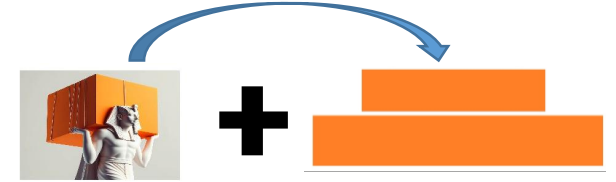
Побудувати піраміду з 1 поверху!

Навіщо потрібна рекурсія?

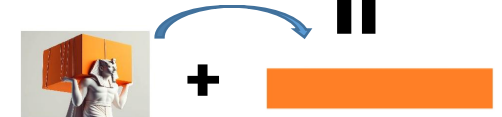


Побудувати піраміду з 4 поверхів!

Побудувати піраміду з 3 поверхів!



Побудувати піраміду з 2 поверхів!



Побудувати піраміду з 1 поверху!



Поставити 1 блок!

Рекурсія

Рекурсія – це такий спосіб організації обчислювального процесу, за якого функція звертається сама до себе. Такі звернення називаються *рекурсивними викликами*, а функція, що містить рекурсивні виклики, – *рекурсивною*.

НАМАГАЙТЕСЬ УНИКАТИ РЕКУРСІЇ !!!

Рекурсія

Рекурсію використовують у ситуаціях, коли легко звести вихідну задачу до задачі того ж виду, але з іншими вихідними даними.

Приклад. Найпростішим прикладом такої задачі може стати обчислення факторіала.

$$n! = 1 * 2 * 3 * 4 * 5 * \dots * n$$

Так обчислення факторіала може бути здійснене у відповідності до наступного рекурентного правила:

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Тобто, наприклад, при обчисленні $5!$ можемо записати

$$5! = 1 * 2 * 3 * 4 * 5 = \underbrace{1 * 2 * 3 * 4}_{4!} * 5 = 4! * 5$$

$$5! = 1 * 2 * 3 * 4 * 5 = 4! * 5$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Прямий хід рекурсії

$$5! = 1 * 2 * 3 * 4 * 5 = \underbrace{4!}_{1 * 2 * 3 * 4} * 5$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Прямий хід рекурсії

$$5! = 1 * 2 * 3 * 4 * 5 = \underbrace{4!}_{\underbrace{1 * 2 * 3}_{3!} * 4} * 5$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Прямий хід рекурсії

$$5! = 1 * 2 * 3 * 4 * 5 = \underbrace{4!}_{3! * 4} * 5$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Прямий хід рекурсії

$$5! = 1 * 2 * 3 * 4 * 5 = \underbrace{4!}_{\underbrace{3! * 4}_{\underbrace{2! * 3}}}$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Прямий хід рекурсії

$$\begin{aligned}
 5! &= 1 * 2 * 3 * 4 * 5 = \underbrace{4!}_{\text{dashed box}} * 5 \\
 &\quad \underbrace{3!}_{\text{dashed box}} * 4 \\
 &\quad \underbrace{2!}_{\text{dashed box}} * 3 \\
 &\quad \underbrace{1!}_{\text{dashed box}} * 2
 \end{aligned}$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Прямий хід рекурсії

$$\begin{aligned}
 5! &= 1 * 2 * 3 * 4 * 5 = \underbrace{4!}_{\substack{3! * 4 \\ \underbrace{2! * 3}_{\underbrace{1! * 2}_{\underbrace{1}}}}} * 5
 \end{aligned}$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Прямий хід рекурсії

$$\begin{aligned}
 5! &= 1 * 2 * 3 * 4 * 5 = \underbrace{4!}_{\text{dashed box}} * 5 \\
 &\quad \underbrace{3!}_{\text{dashed box}} * 4 \\
 &\quad \underbrace{2!}_{\text{dashed box}} * 3 \\
 &\quad \underbrace{1!}_{\text{dashed box}} * 2 \\
 &\quad \underbrace{1}_{\text{dashed box}}
 \end{aligned}$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Зворотній хід рекурсії

$$5! = 1 * 2 * 3 * 4 * 5 = \underbrace{4!}_{\text{dashed box}} * 5$$

$$\underbrace{3!}_{\text{dashed box}} * 4$$

$$\underbrace{2!}_{\text{dashed box}} * 3$$

$$\underbrace{1}_{\text{dashed box}} * 2 = 2$$



$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Зворотній хід рекурсії

$$5! = 1 * 2 * 3 * 4 * 5 = \underbrace{4!}_{\text{}} * 5$$

$$\underbrace{3!}_{\text{}} * 4$$

$$\underbrace{2}_{\text{}} * 3 = 6$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Зворотній хід рекурсії

$$5! = 1 * 2 * 3 * 4 * 5 = \underbrace{4!}_{6} * 5$$

$6 * 4 = 24$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Зворотній хід рекурсії

$$5! = 1 * 2 * 3 * 4 * 5 = 24 * 5$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Зворотній хід рекурсії

$$5! = 1 * 2 * 3 * 4 * 5 = 24 * 5 = 120$$

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

Зворотній хід рекурсії

Нагадаємо, що таке ***контекст виконання***

Execution context (контекст виконання)

Контекст виконання – середовище у якому браузер виконує програми JavaScript.

Компоненти контексту виконання :

- 1) Компонент пам'яті - **Memory** (колекція пар ключ-значення : *назва_змінної* : *значення_змінної*)
- 2) Компонент коду - **Thread of Execution** (місце де виконуються команди)

| Memory | Thread of Execution |
|--------|---------------------|
| | |

Приклад.

| Memory | Thread of Execution |
|--|--|
| <pre>num1 : 2 num2 : 3 ... s : 5</pre> | <pre>var num1 = 2 var num2 = 3 var s = num1 + num2 document.write('Summa = ' + s)</pre> |

Фази виконання програм JavaScript

- 1) Фаза виділення пам'яті (виділення пам'яті для змінних і функцій)
- 2) Фаза виконання команд (поступове виконання команд по одній за раз)

Приклад.

| Фаза вик. | Стан контексту виконання | |
|--|---|---------------------|
| 1. Виділення пам'яті для: 1) змінних(змінн і мають початкове значення <u>undefined</u>) 2)функцій (function-declaration) | Memory | Thread of Execution |
| | <div><div>x1: undefined, x2: undefined, x3: undefined, x4: undefined, s1: undefined, s2: undefined, sum: function{...}</div></div> | |
| 2) Виконання команд | | |

```
function sum(a, b) {  
  res = a + b  
  return res  
}  
  
let x1 = 2, x2 = 3,  
    x3 = 7, x4 = 8  
  
let s1 = sum(x1, x2)  
let s2 = sum(x3, x4)  
  
document.write(`s1 = ${s1}, s2=${s2}`)
```

Фази виконання програм JavaScript

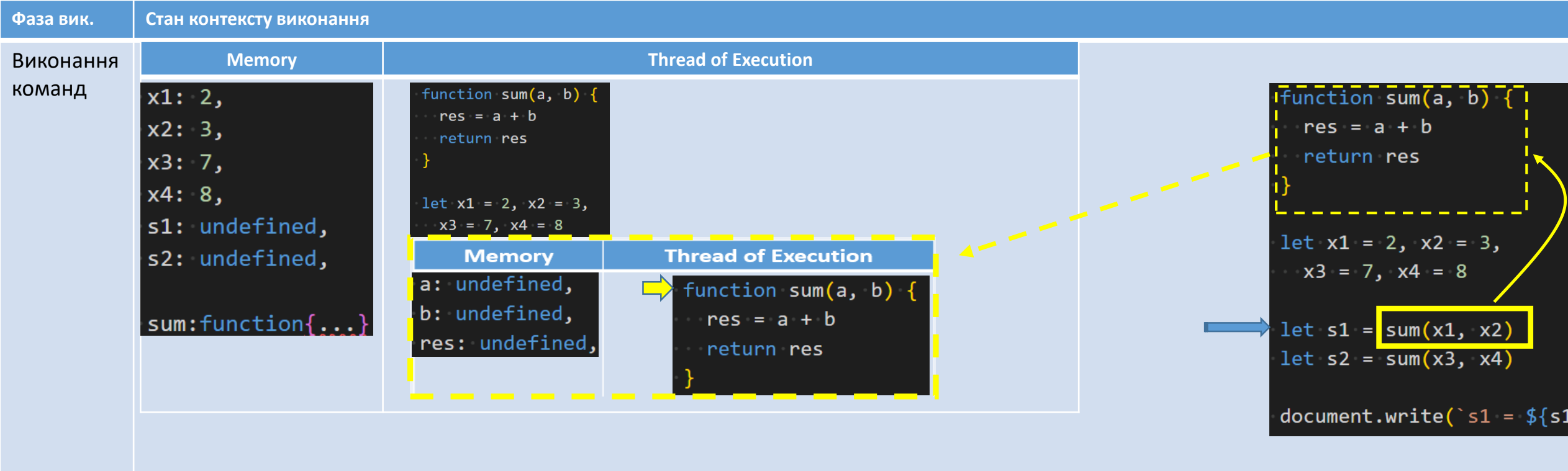
1) **Фаза виділення пам'яті** (виділення пам'яті для змінних і функцій)

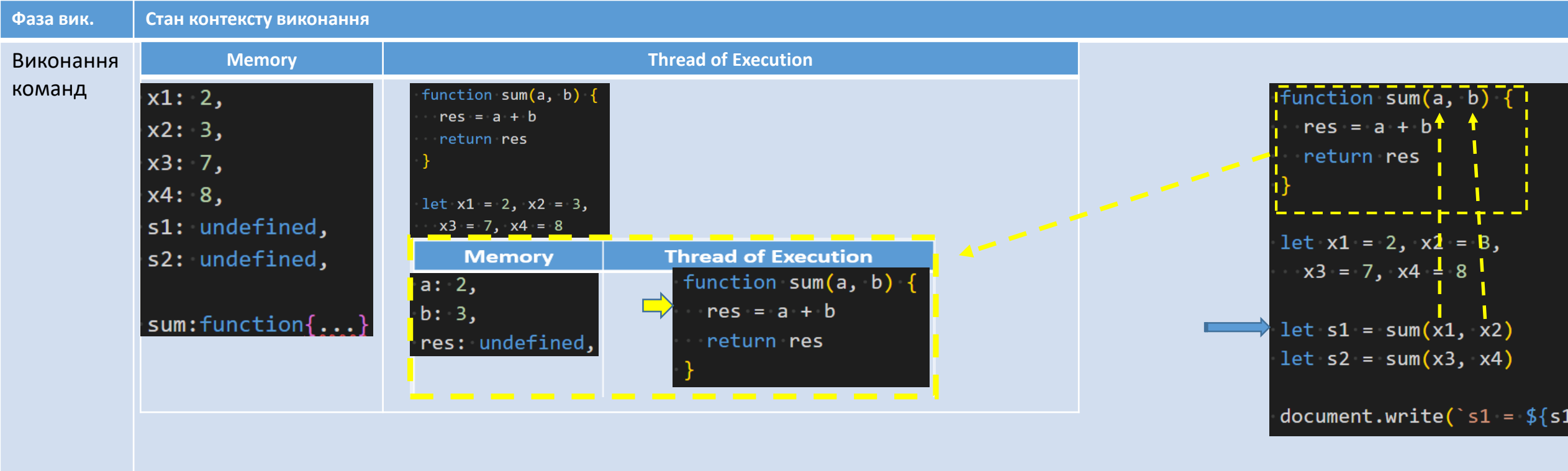
2) **Фаза виконання команд** (поступове виконання команд по одній за раз)

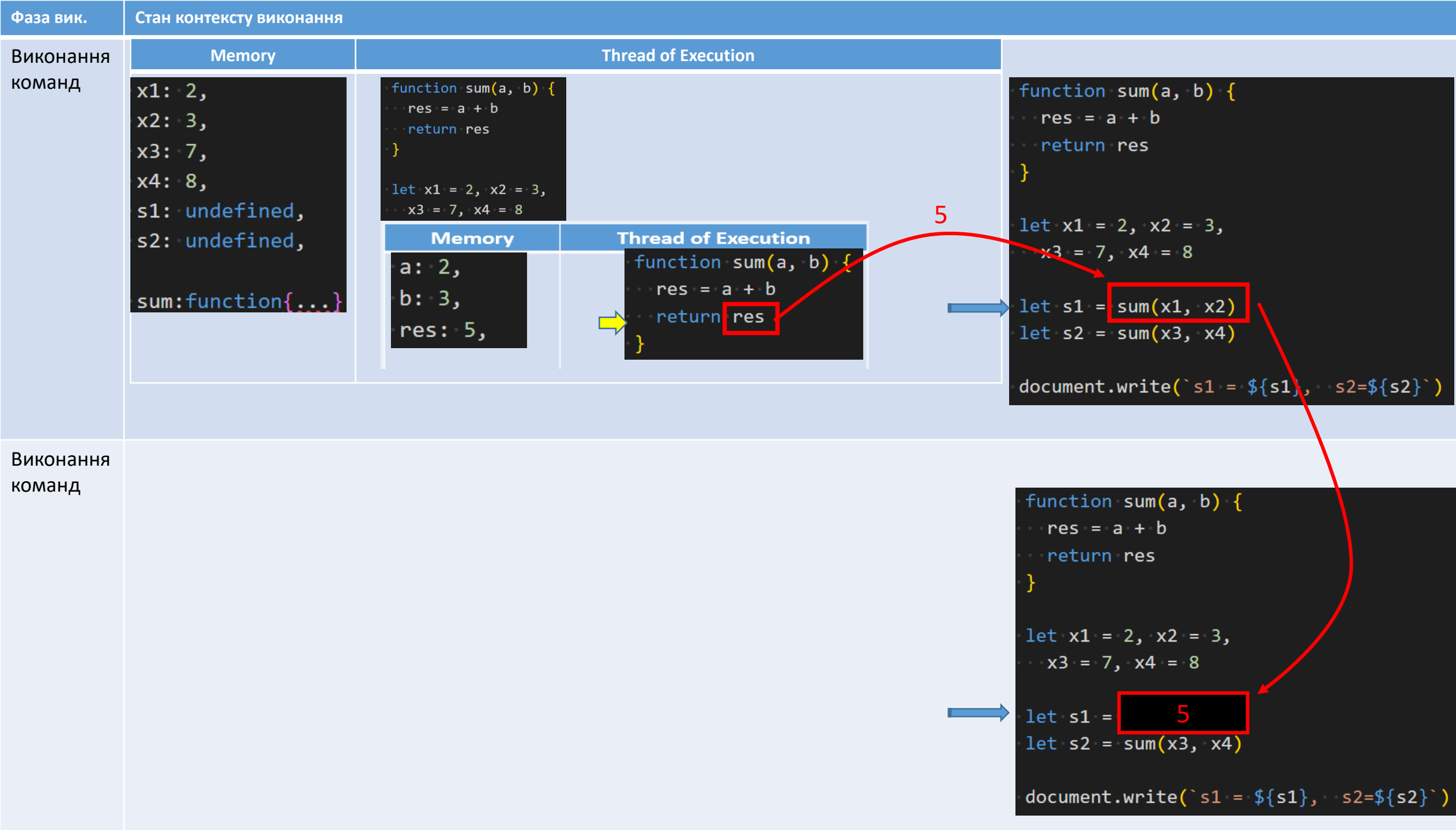
Приклад.

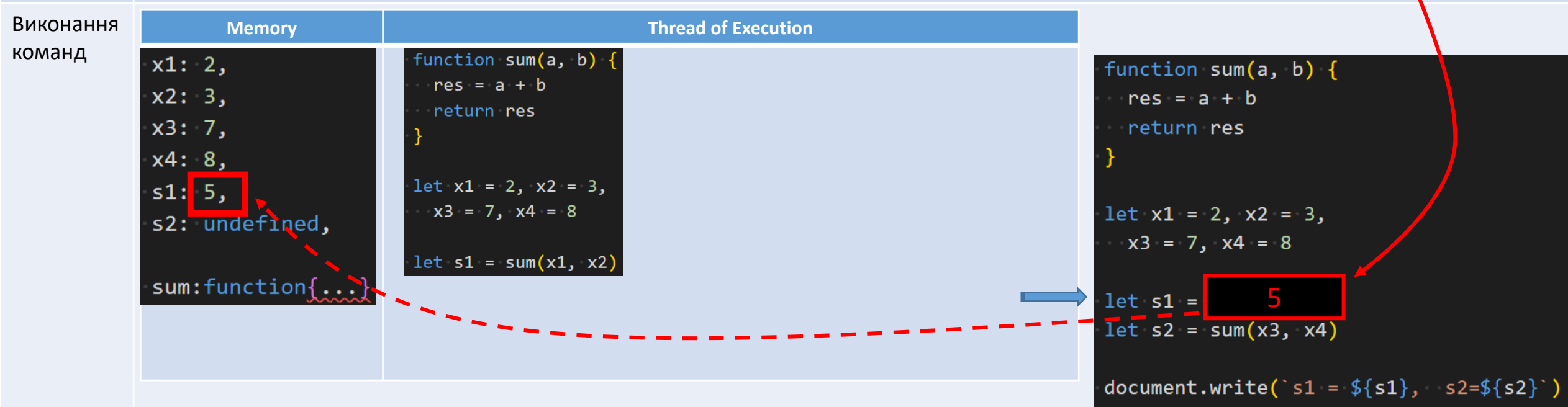
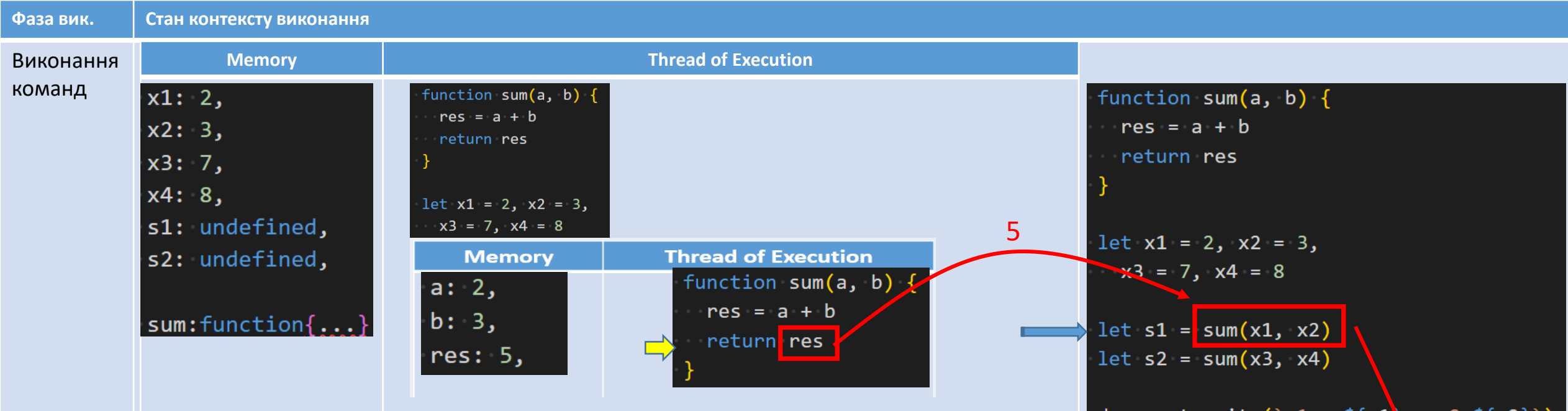
| Фаза вик. | Стан контексту виконання | |
|---|--|---|
| 1. Виділення пам'яті для: 1) змінних(змінн і мають початкове значення undefined) 2)функцій (function-declaration) | Memory | Thread of Execution |
| | <pre>x1: undefined, x2: undefined, x3: undefined, x4: undefined, s1: undefined, s2: undefined, sum: function{...}</pre> | |
| 2) Виконання команд | Memory | Thread of Execution |
| | <pre>x1: 2, x2: 3, x3: 7, x4: 8, s1: undefined, s2: undefined, sum: function{...}</pre> | <pre>function sum(a, b) { res = a + b return res } let x1 = 2, x2 = 3, x3 = 7, x4 = 8 let s1 = sum(x1, x2) let s2 = sum(x3, x4) document.write(`s1 = \${s1}, s2=\${s2}`)</pre> |

| Фаза вик. | Стан контексту виконання | |
|------------------|---|--|
| Виконання команд | Memory | Thread of Execution |
| | <pre>x1: 2, x2: 3, x3: 7, x4: 8, s1: undefined, s2: undefined, sum:function{...}</pre> | <pre>function sum(a, b) { res = a + b return res } let x1 = 2, x2 = 3, x3 = 7, x4 = 8</pre> |
| | | <pre>function sum(a, b) { res = a + b return res } let x1 = 2, x2 = 3, x3 = 7, x4 = 8 let s1 = sum(x1, x2) let s2 = sum(x3, x4) document.write(`s1 = \${s1}`)</pre> |

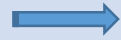


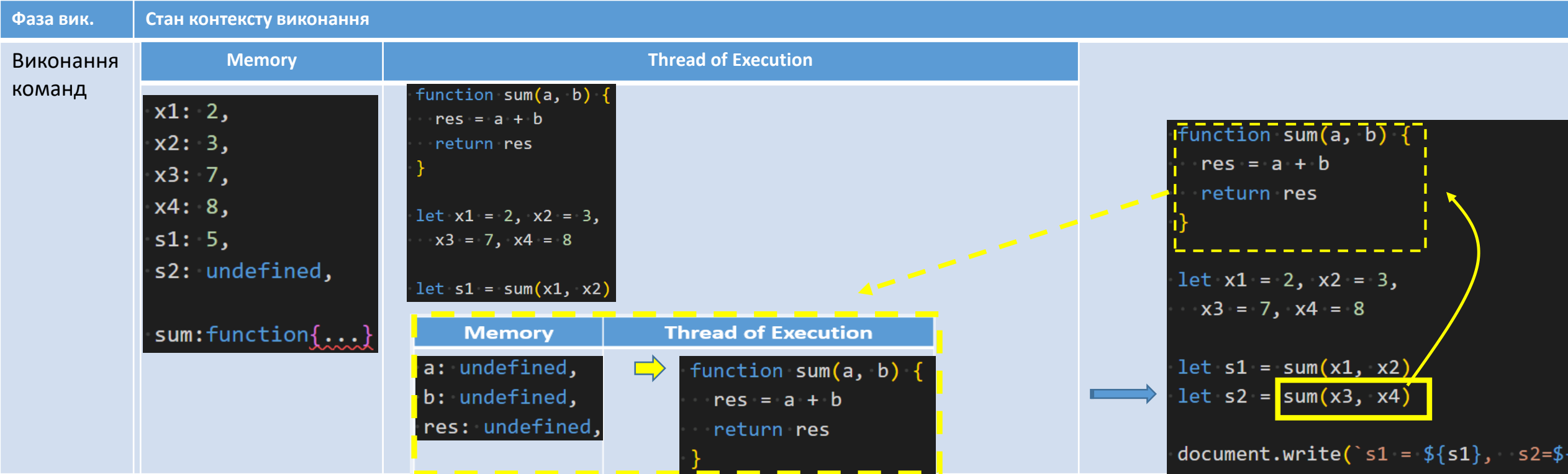


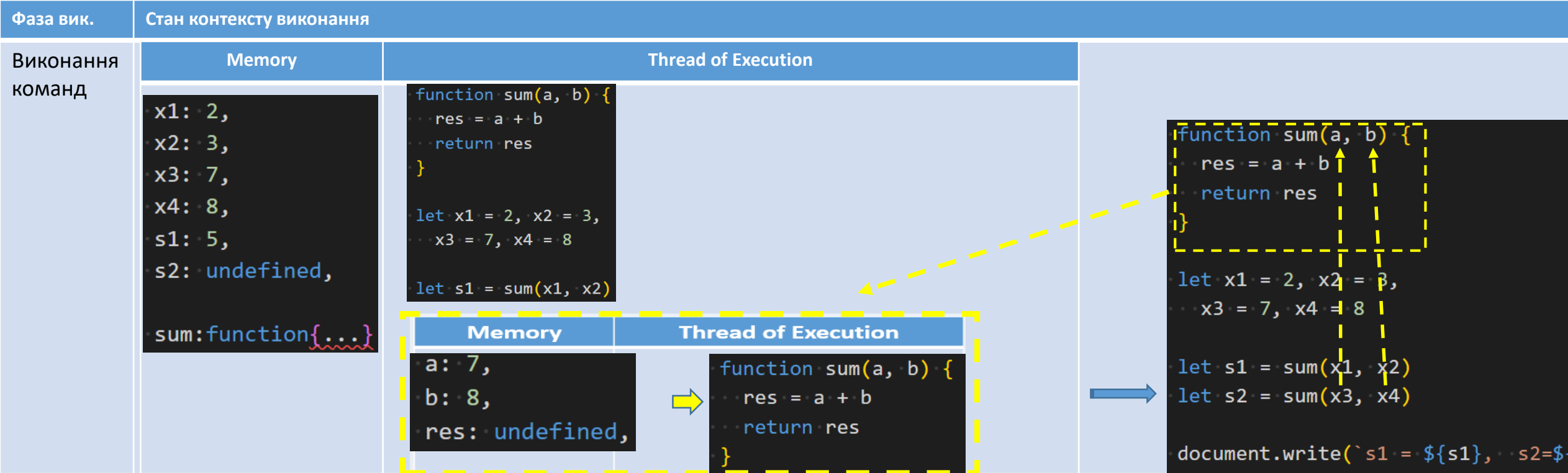




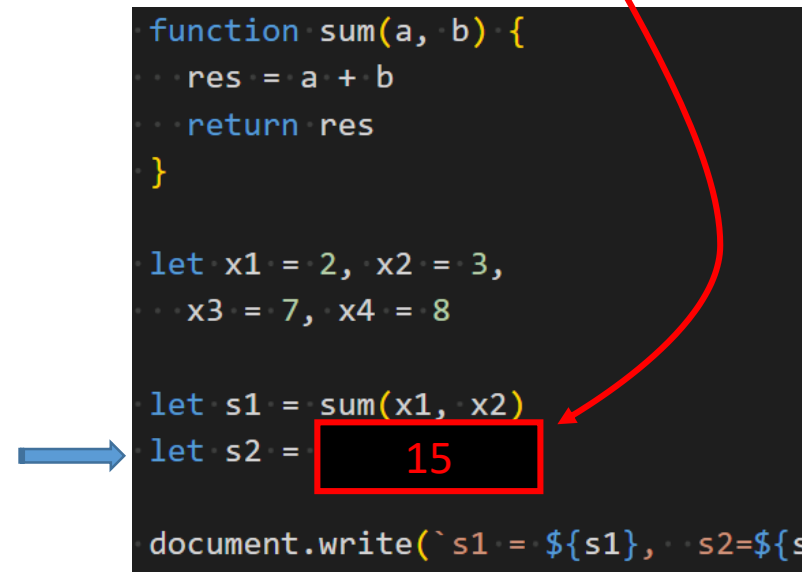
| Фаза вик. | Стан контексту виконання | | |
|------------------|---|--|--|
| Виконання команд | Memory | Thread of Execution | |
| | <pre> x1: 2, x2: 3, x3: 7, x4: 8, s1: 5, s2: undefined, sum: function {...} </pre> | <pre> function sum(a, b) { res = a + b return res } let x1 = 2, x2 = 3, x3 = 7, x4 = 8 let s1 = sum(x1, x2) </pre> | |
| | | <div>→</div> <pre> function sum(a, b) { res = a + b return res } let x1 = 2, x2 = 3, x3 = 7, x4 = 8 let s1 = sum(x1, x2) let s2 = sum(x3, x4) document.write(`s1 = \${s1}, s2=\${s2}`) </pre> | |

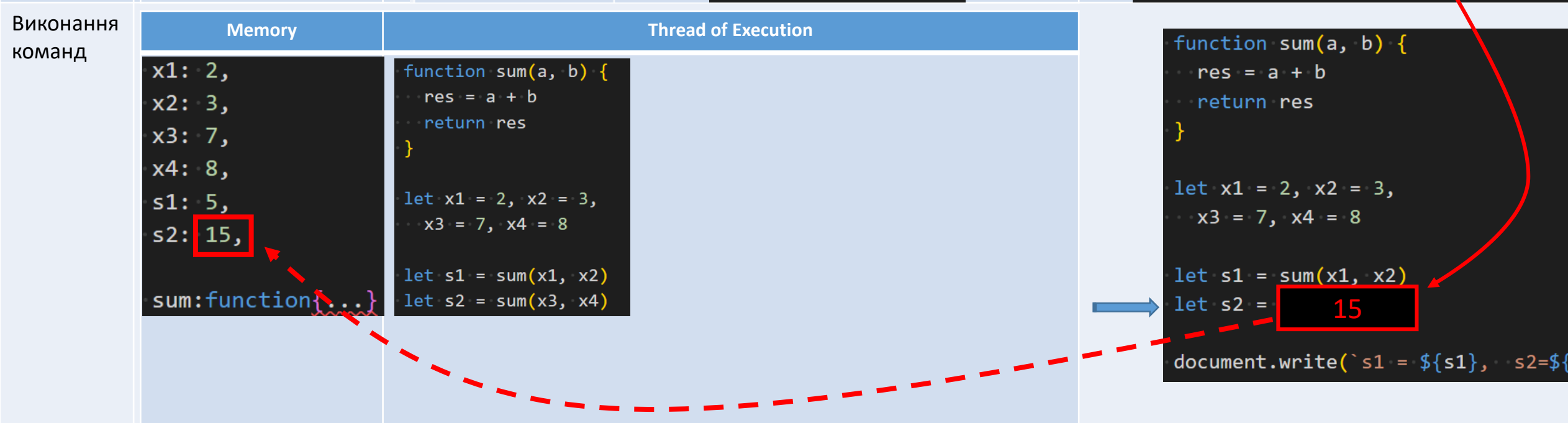
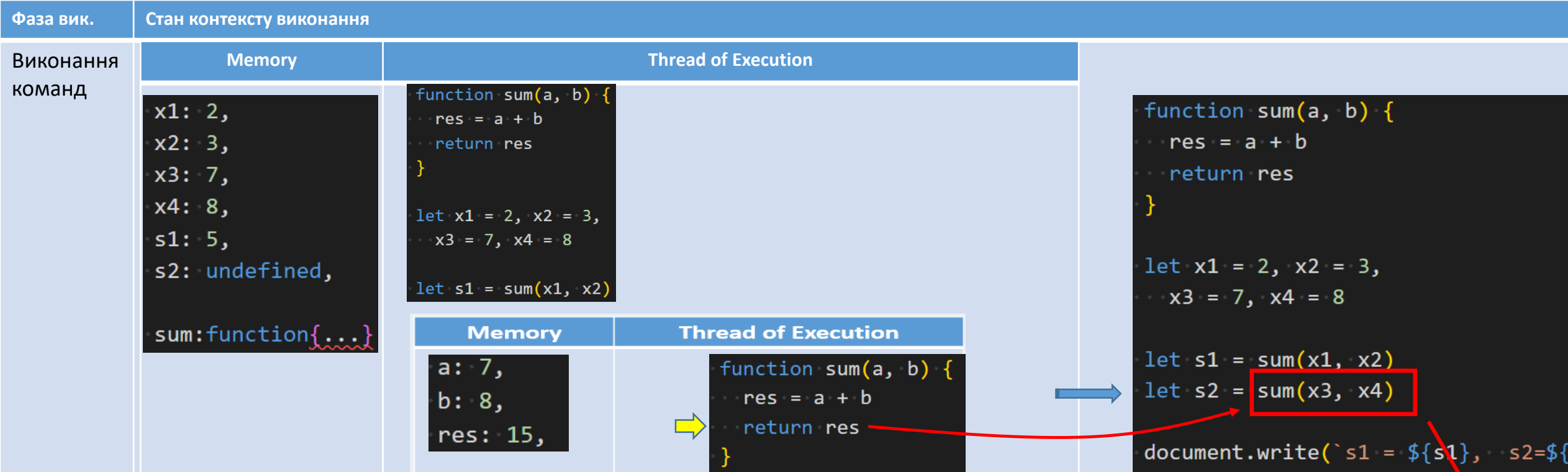
| Фаза вик. | Стан контексту виконання | | |
|------------------|--|--|---|
| Виконання команд | Memory | Thread of Execution | |
| | <pre>x1: 2, x2: 3, x3: 7, x4: 8, s1: 5, s2: undefined, sum: function{...}</pre> | <pre>function sum(a, b) { res = a + b return res } let x1 = 2, x2 = 3, x3 = 7, x4 = 8 let s1 = sum(x1, x2)</pre> | <div><pre>function sum(a, b) { res = a + b return res } let x1 = 2, x2 = 3, x3 = 7, x4 = 8 let s1 = sum(x1, x2) let s2 = sum(x3, x4) document.write(`s1 = \${s1}, s2=\${s2}`)</pre></div> |





| Фаза вик. | Стан контексту виконання | | | | | |
|---------------------------------|--|---|--------|---------------------|---------------------------------|--|
| Виконання команд | Memory | Thread of Execution | | | | |
| | <pre>x1: 2, x2: 3, x3: 7, x4: 8, s1: 5, s2: undefined, sum: function{...}</pre> | <pre>function sum(a, b) { res = a + b return res } let x1 = 2, x2 = 3, x3 = 7, x4 = 8 let s1 = sum(x1, x2)</pre> | | | | |
| | | <table><tr><th>Memory</th><th>Thread of Execution</th></tr><tr><td><pre>a: 7, b: 8, res: 15,</pre></td><td><pre>function sum(a, b) { res = a + b return res }</pre></td></tr></table> | Memory | Thread of Execution | <pre>a: 7, b: 8, res: 15,</pre> | <pre>function sum(a, b) { res = a + b return res }</pre> |
| Memory | Thread of Execution | | | | | |
| <pre>a: 7, b: 8, res: 15,</pre> | <pre>function sum(a, b) { res = a + b return res }</pre> | | | | | |





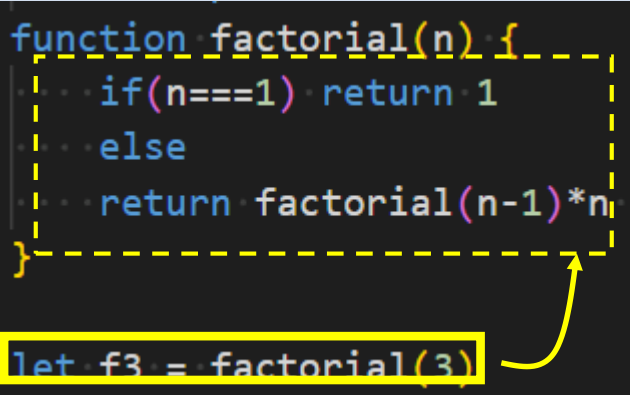
Рекурсивний алгоритм знаходження факторіала

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

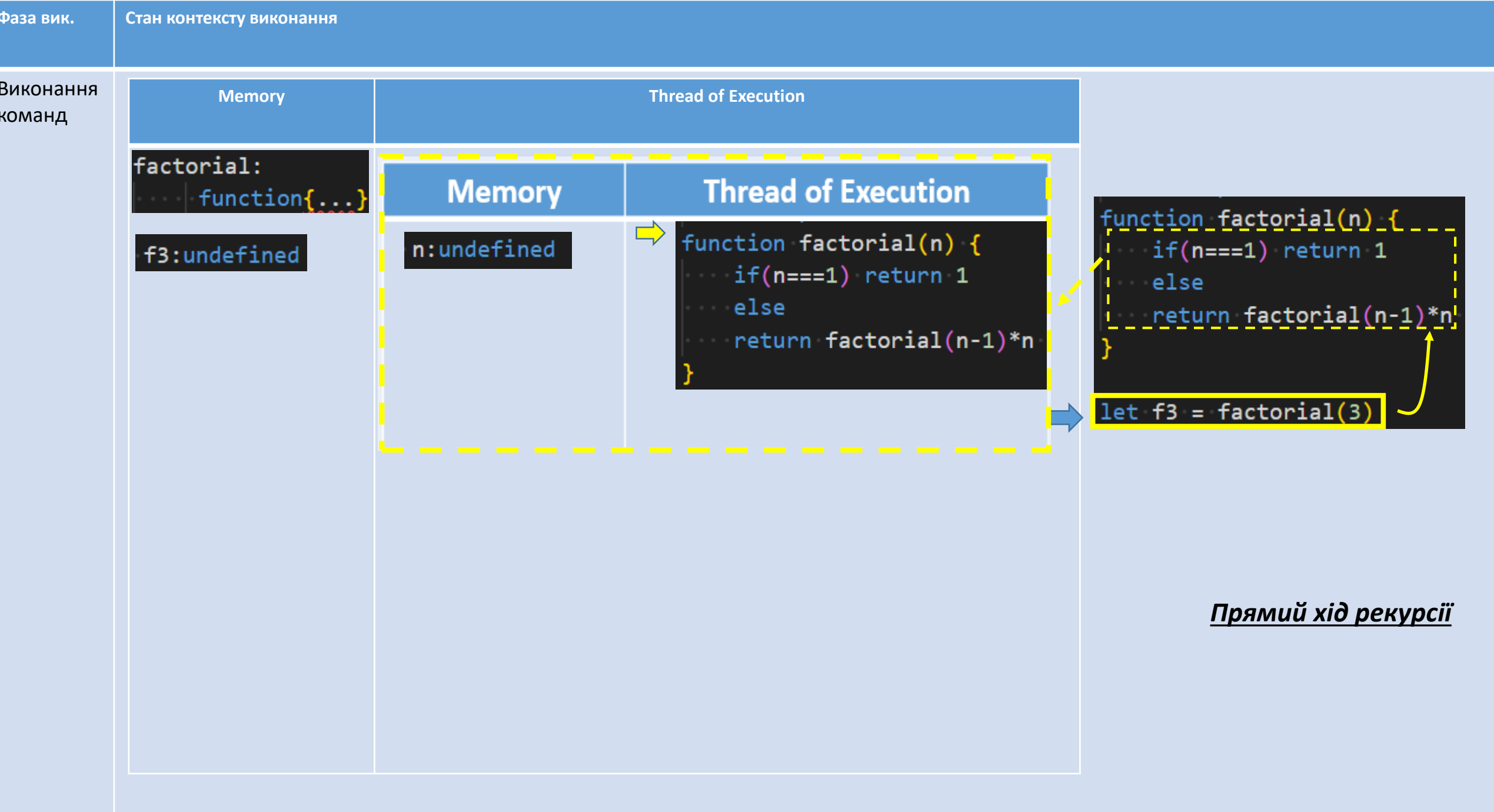
```
function factorial(n) {  
  ... if(n===1) return 1  
  ... else  
  ... return factorial(n-1)*n  
}  
  
let f3 = factorial(3)
```

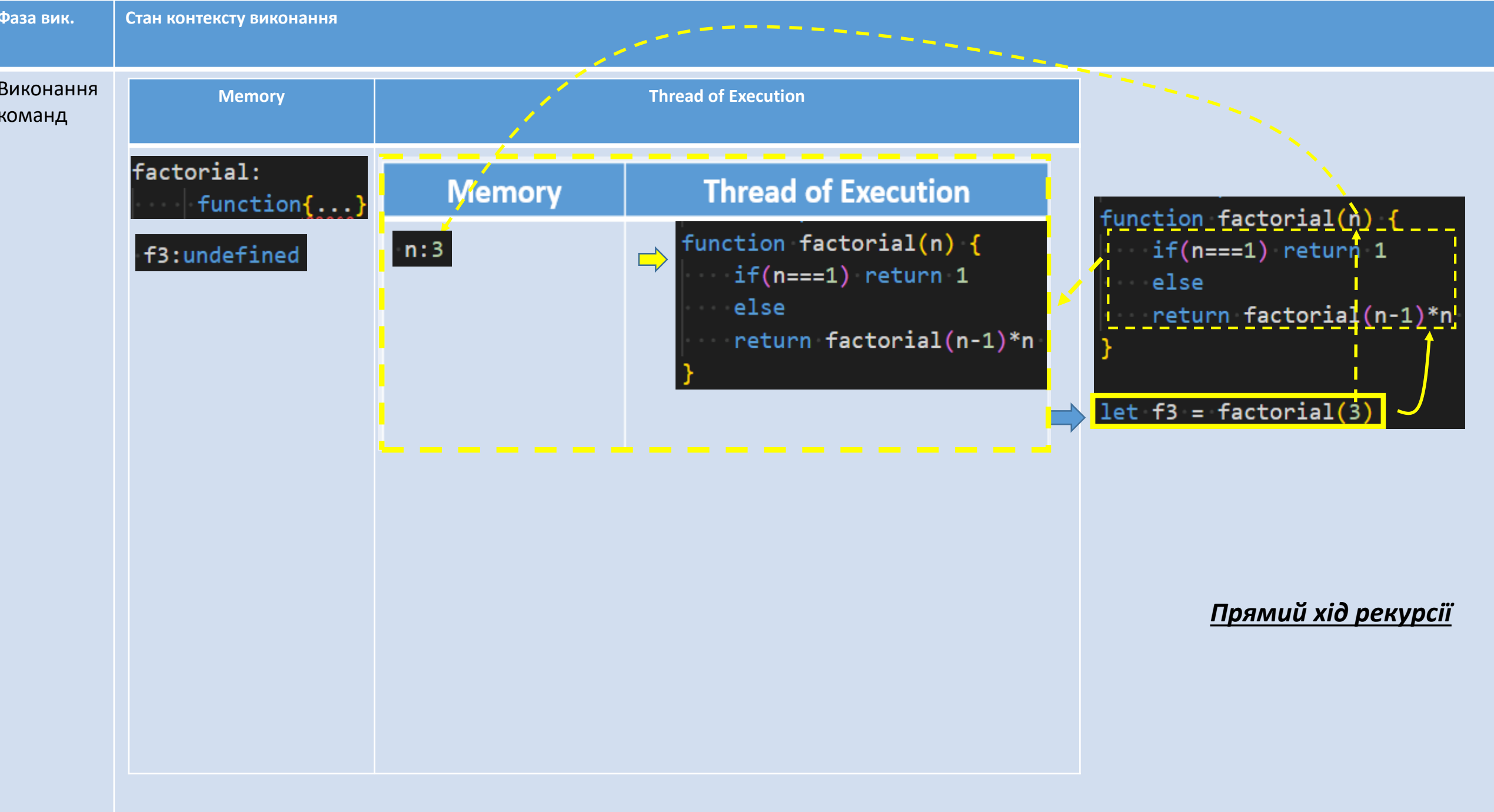
Приклад. Виконання рекурсивних функцій

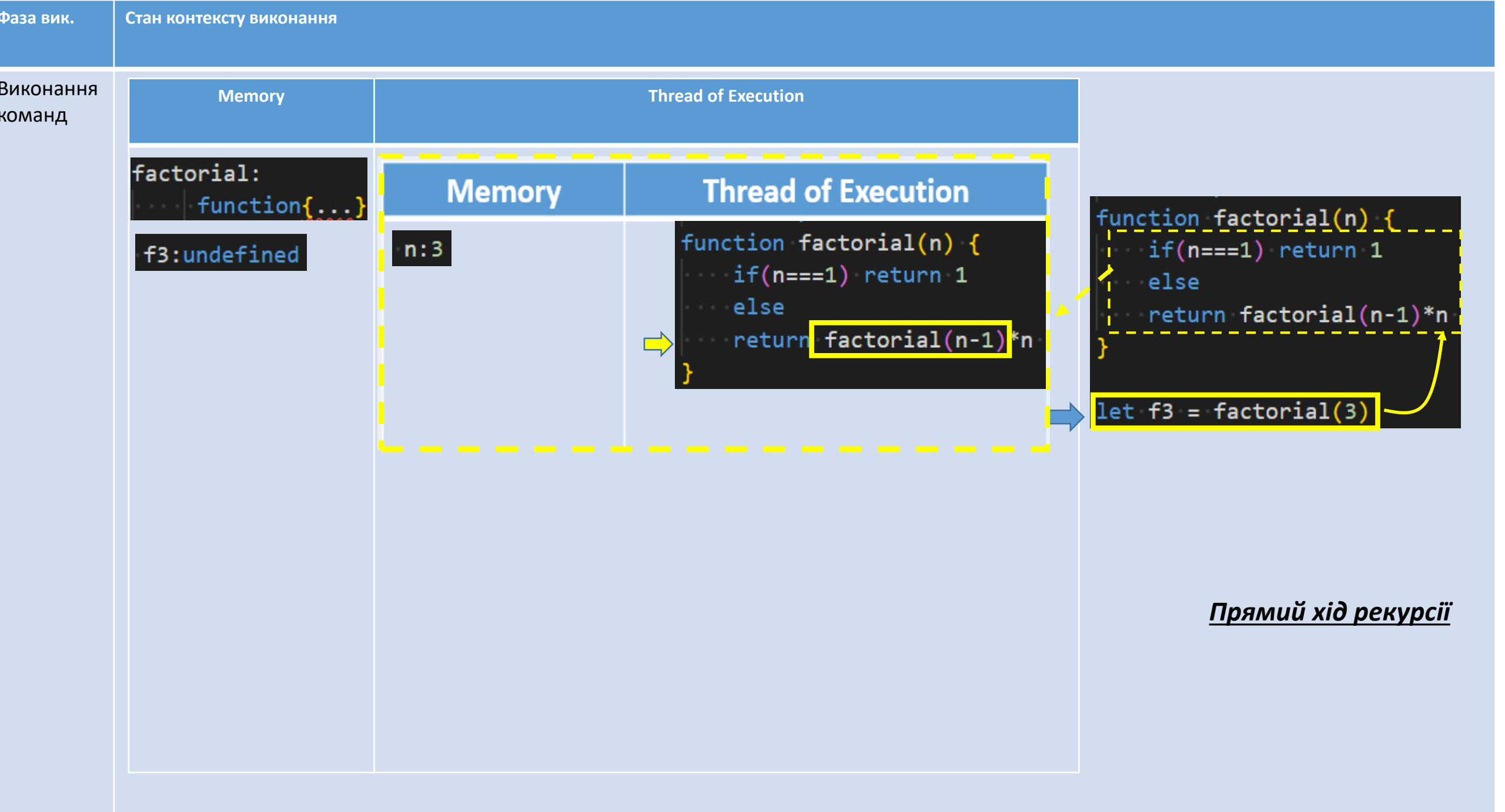
| Фаза вик. | Стан контексту виконання | |
|---|---|---|
| 1. Виділення пам'яті для: 1) змінних(змінн і мають початкове значення <u>undefined</u>) 2)функцій (function-declaration) | Memory | Thread of Execution |
| | <pre>factorial: ... function{...} f3:undefined</pre> | |
| 2) Виконання команд | Memory | Thread of Execution |
| | <pre>factorial: ... function{...} f3:undefined</pre> | <pre>function factorial(n) { ... if(n===1) return 1 ... else ... return factorial(n-1)*n } let f3 = factorial(3)</pre> |

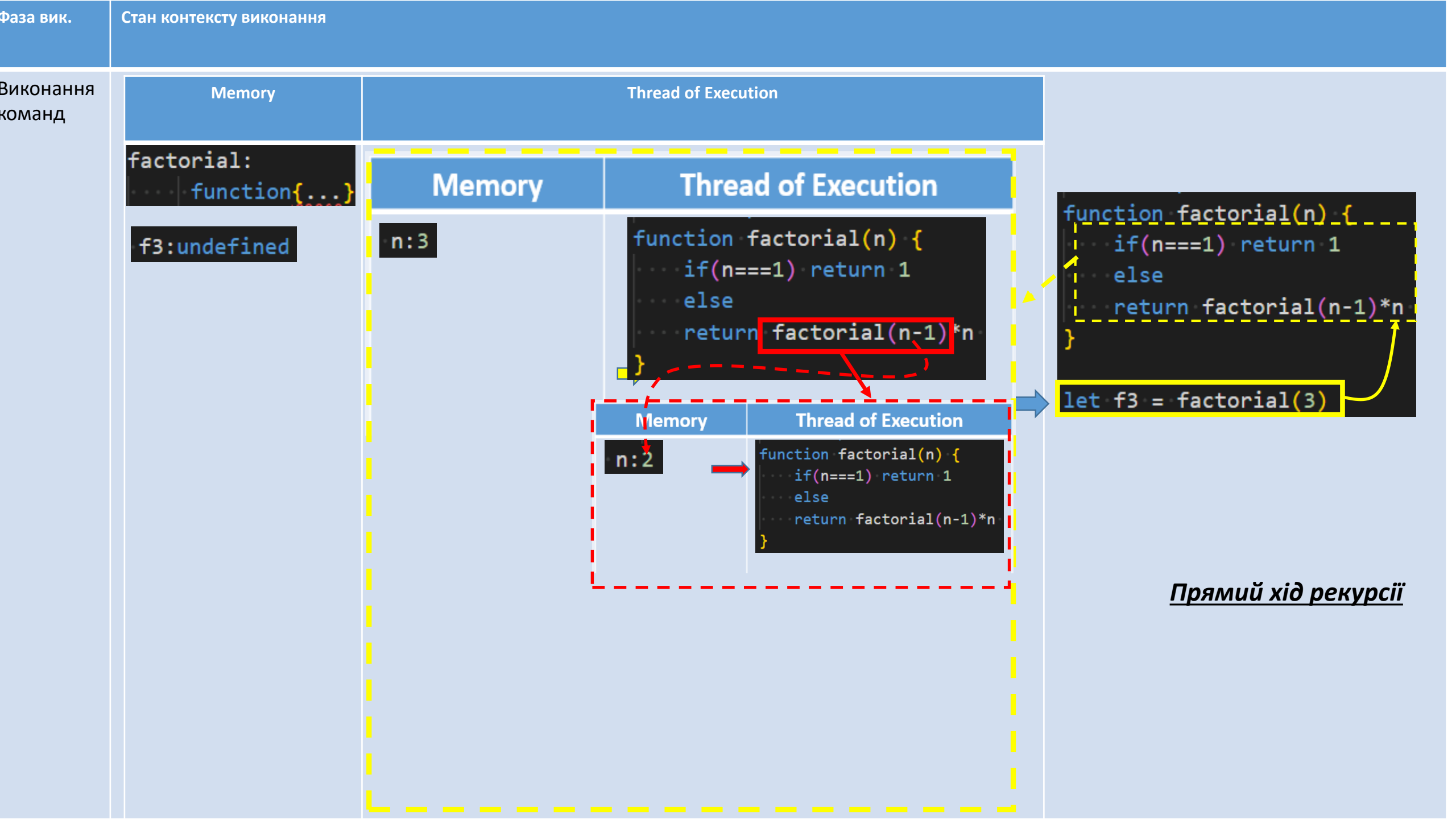
| Фаза вик. | Стан контексту виконання | | |
|------------------|--|---|--|
| Виконання команд | Memory | Thread of Execution | |
| | <pre>factorial: ... function{...} f3: undefined</pre> |  | |

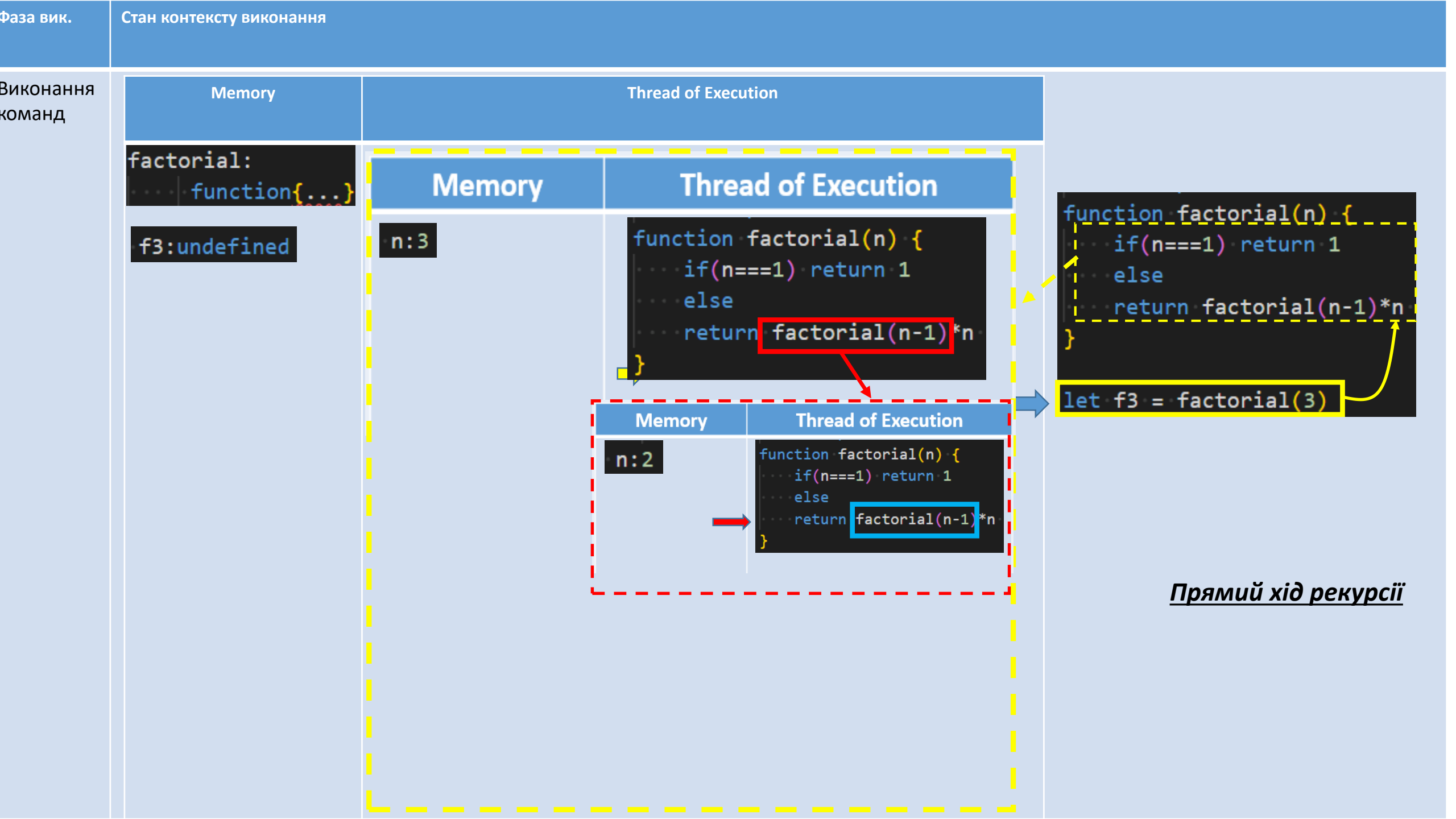
Прямий хід рекурсії

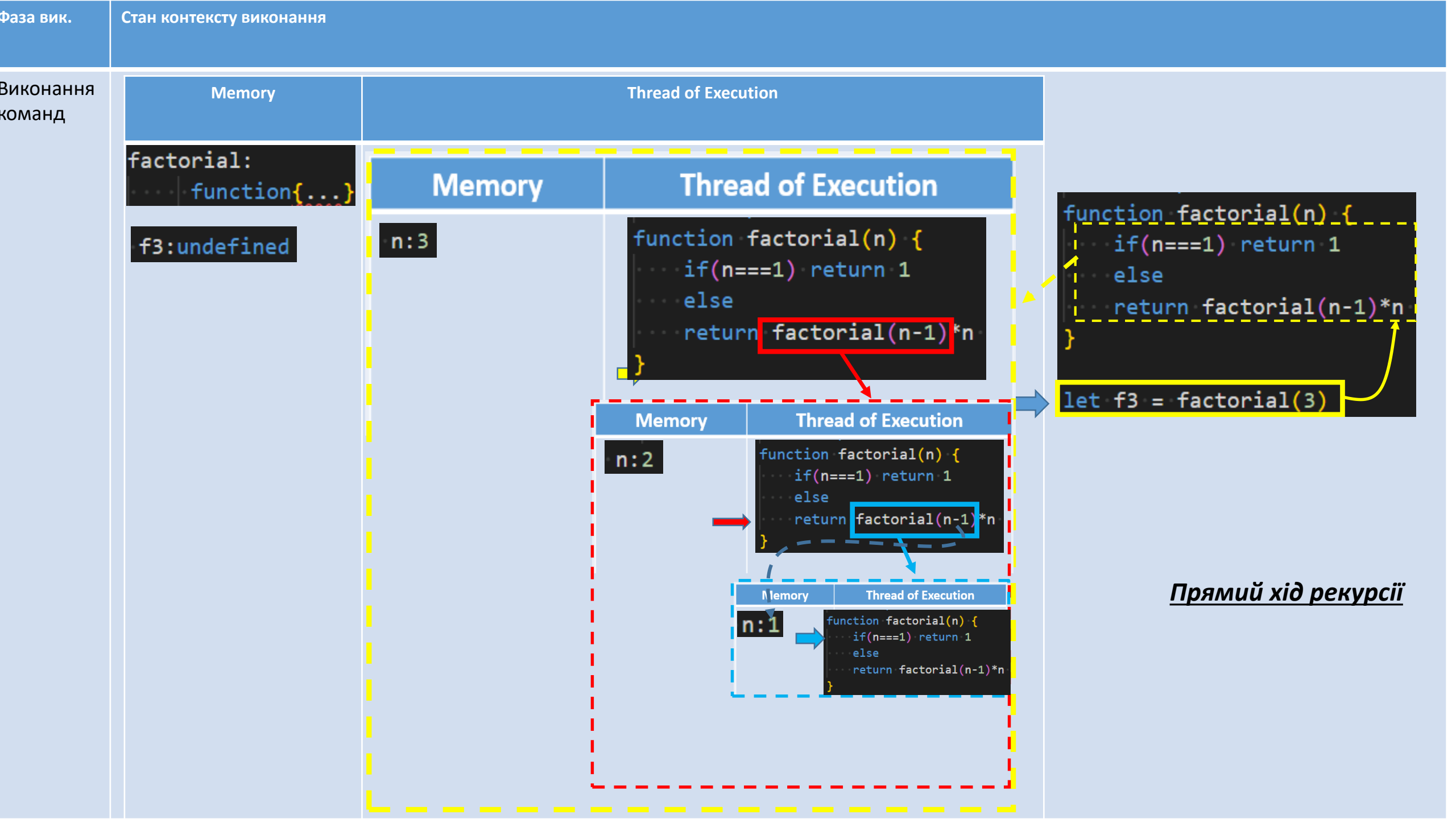


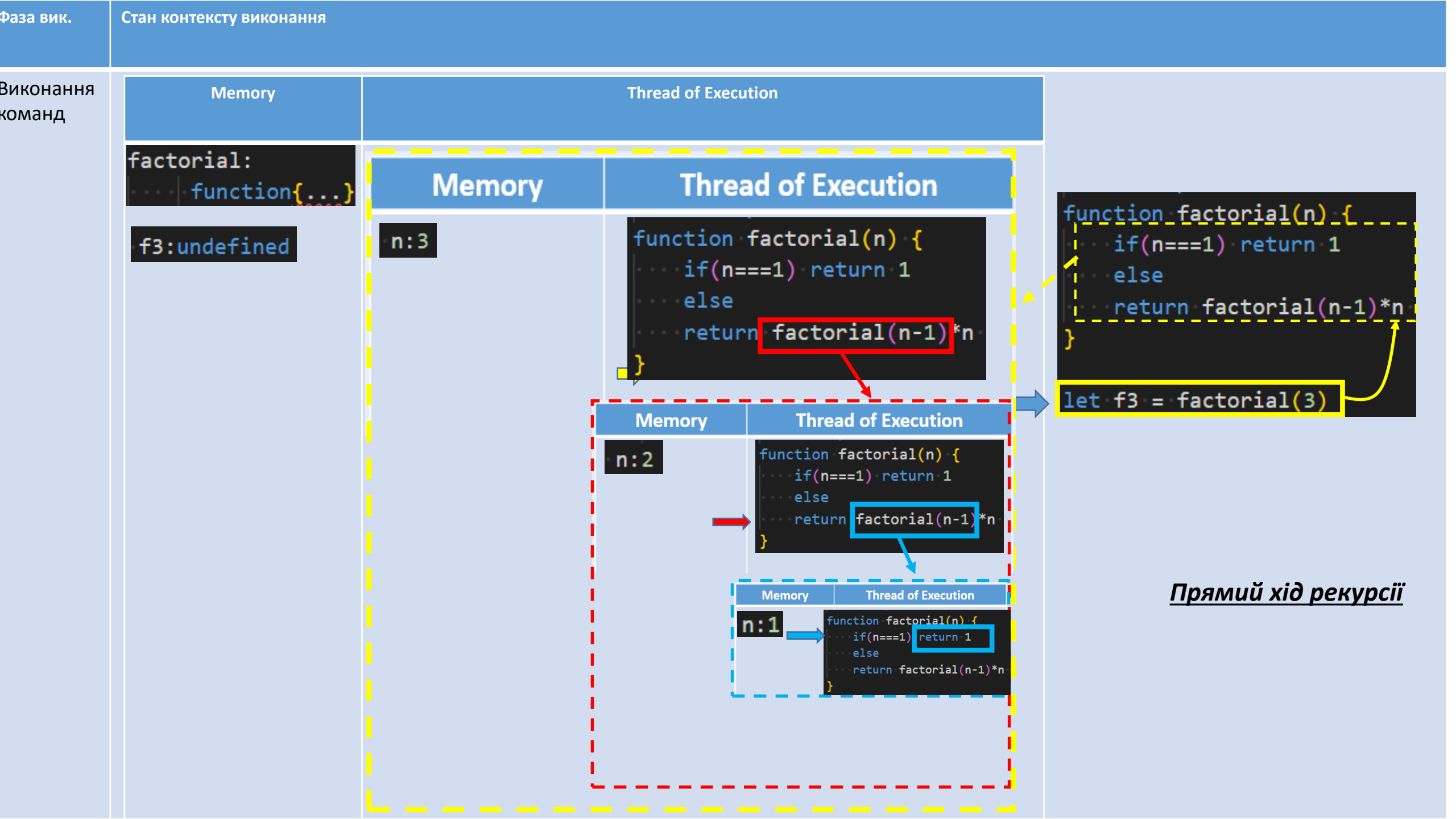


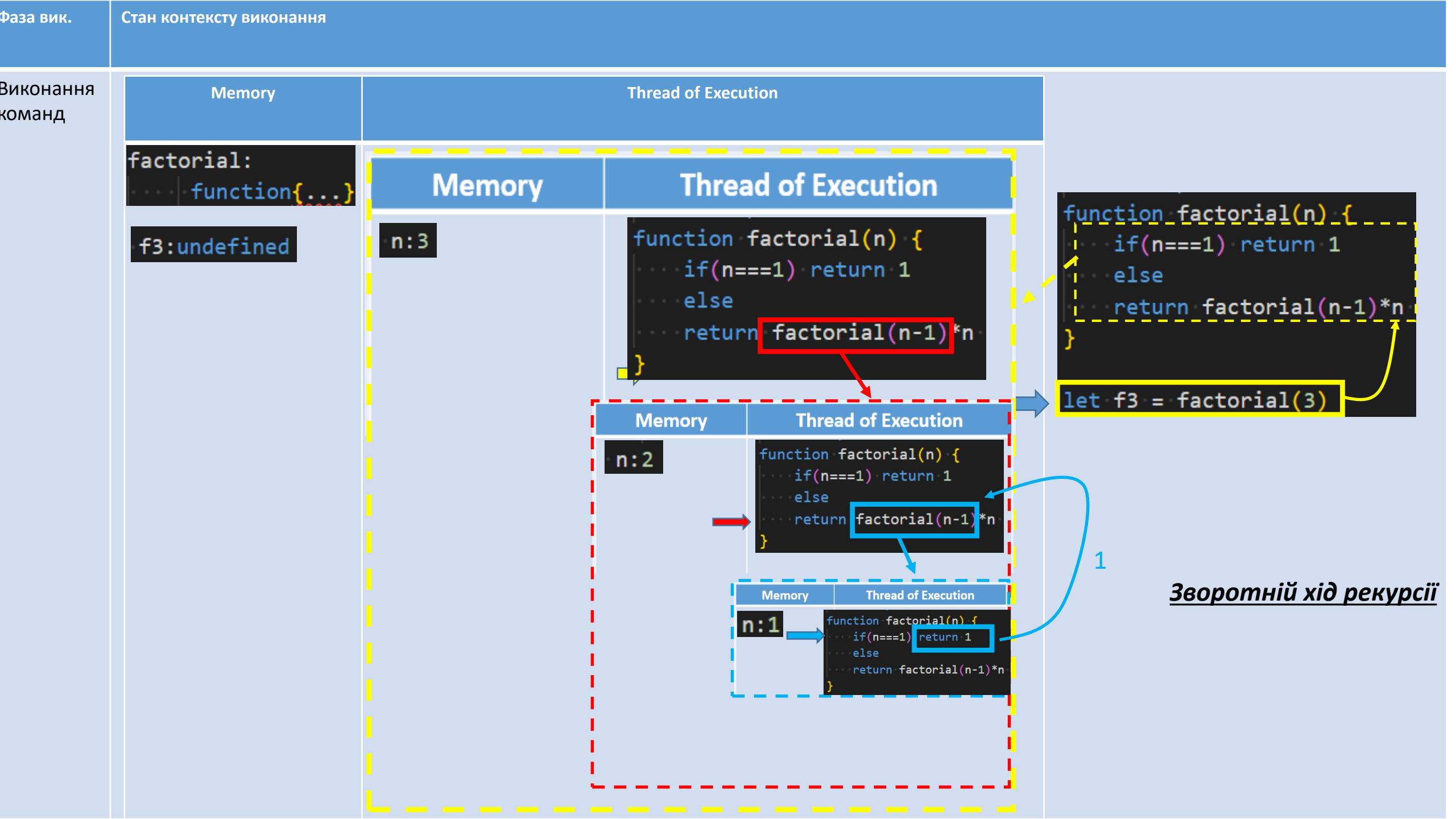






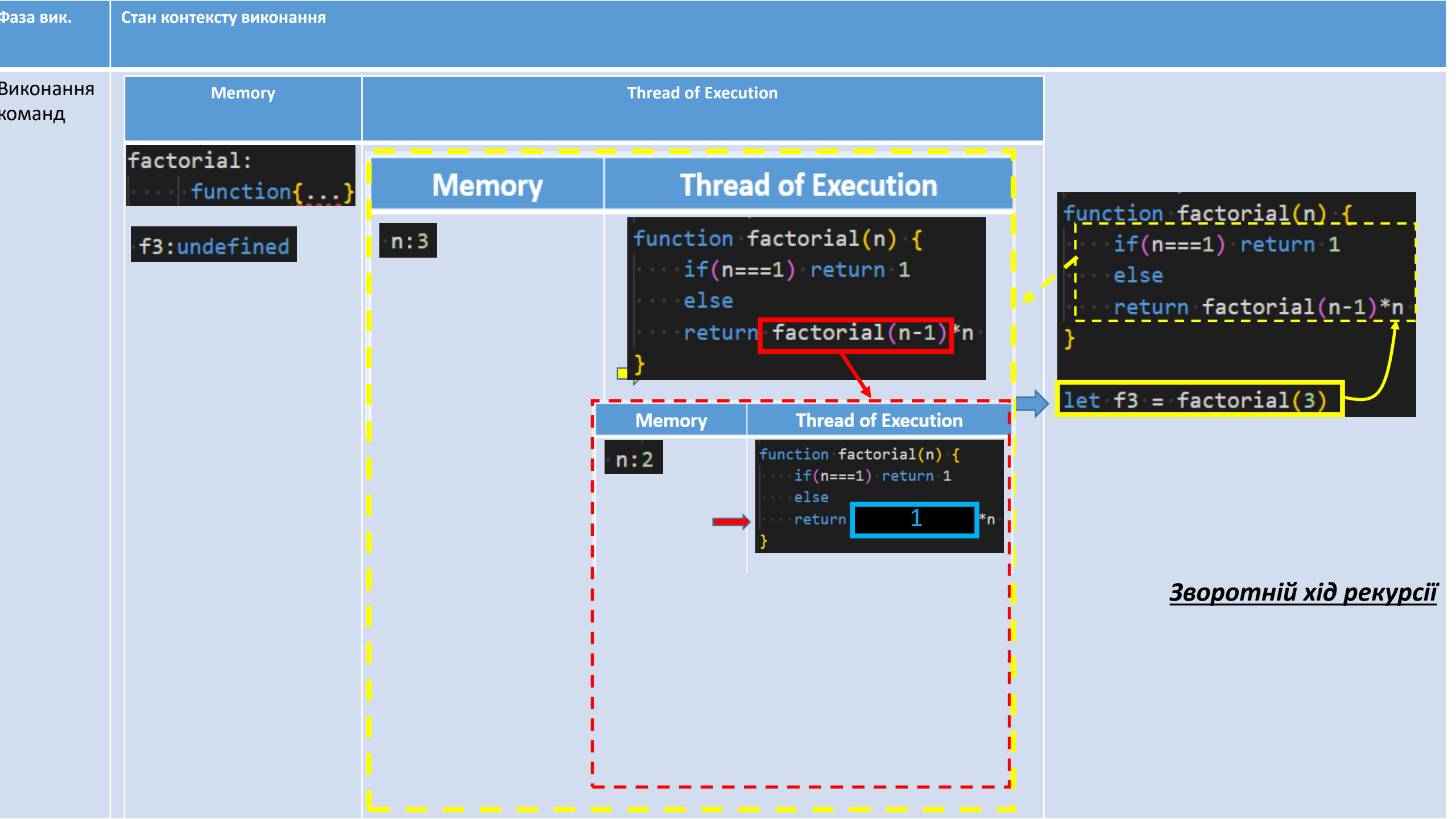


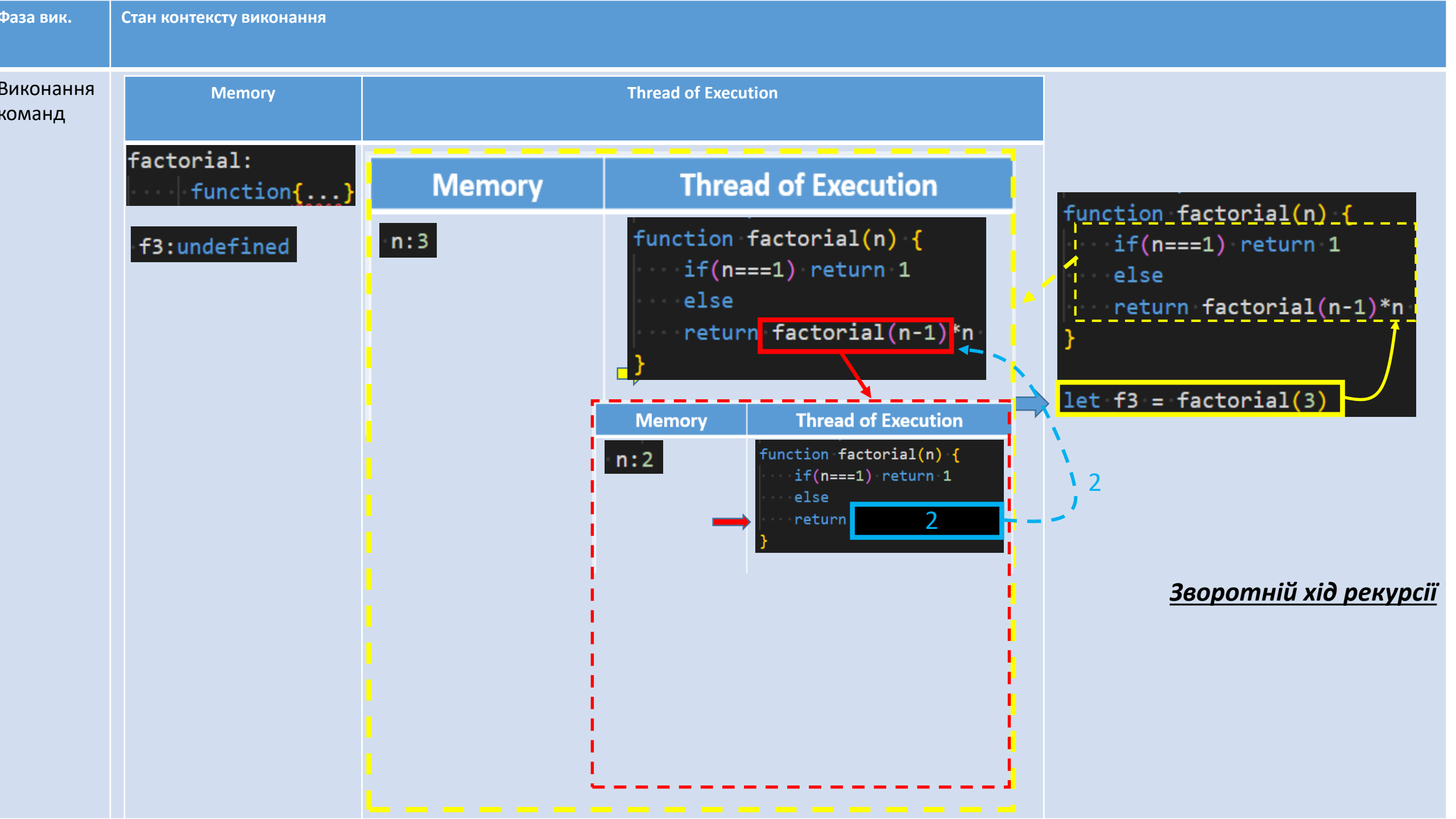


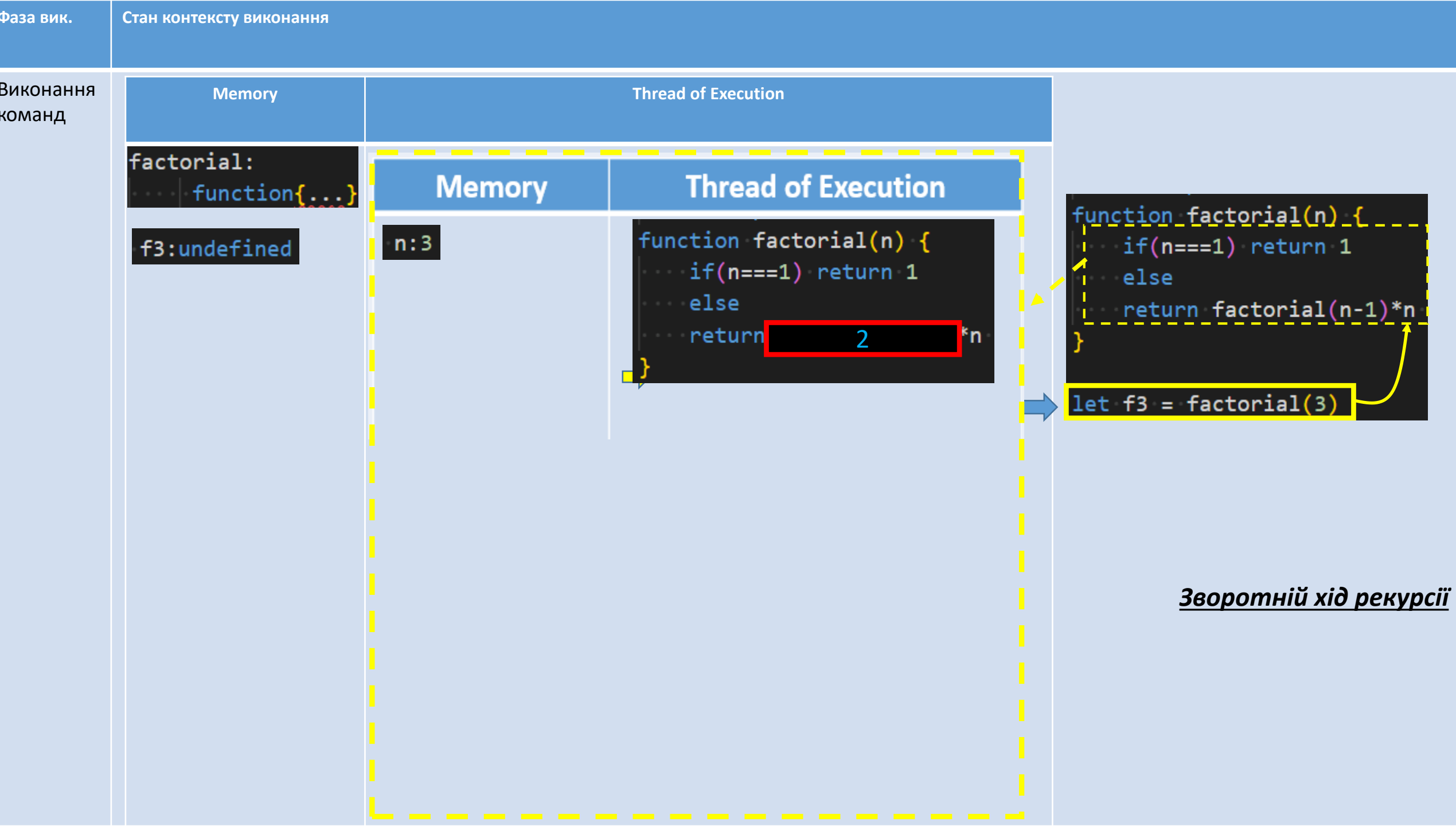


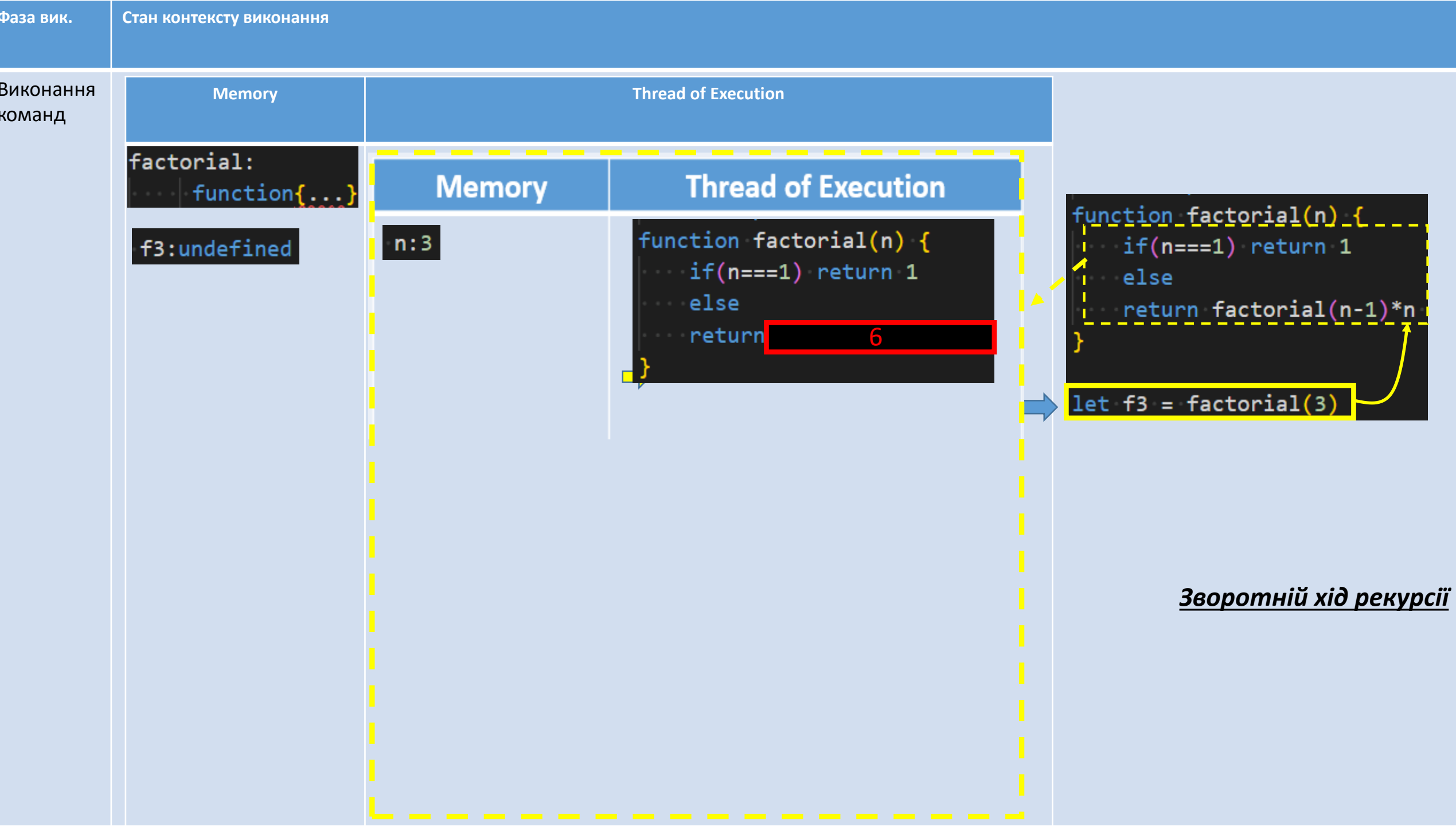
1

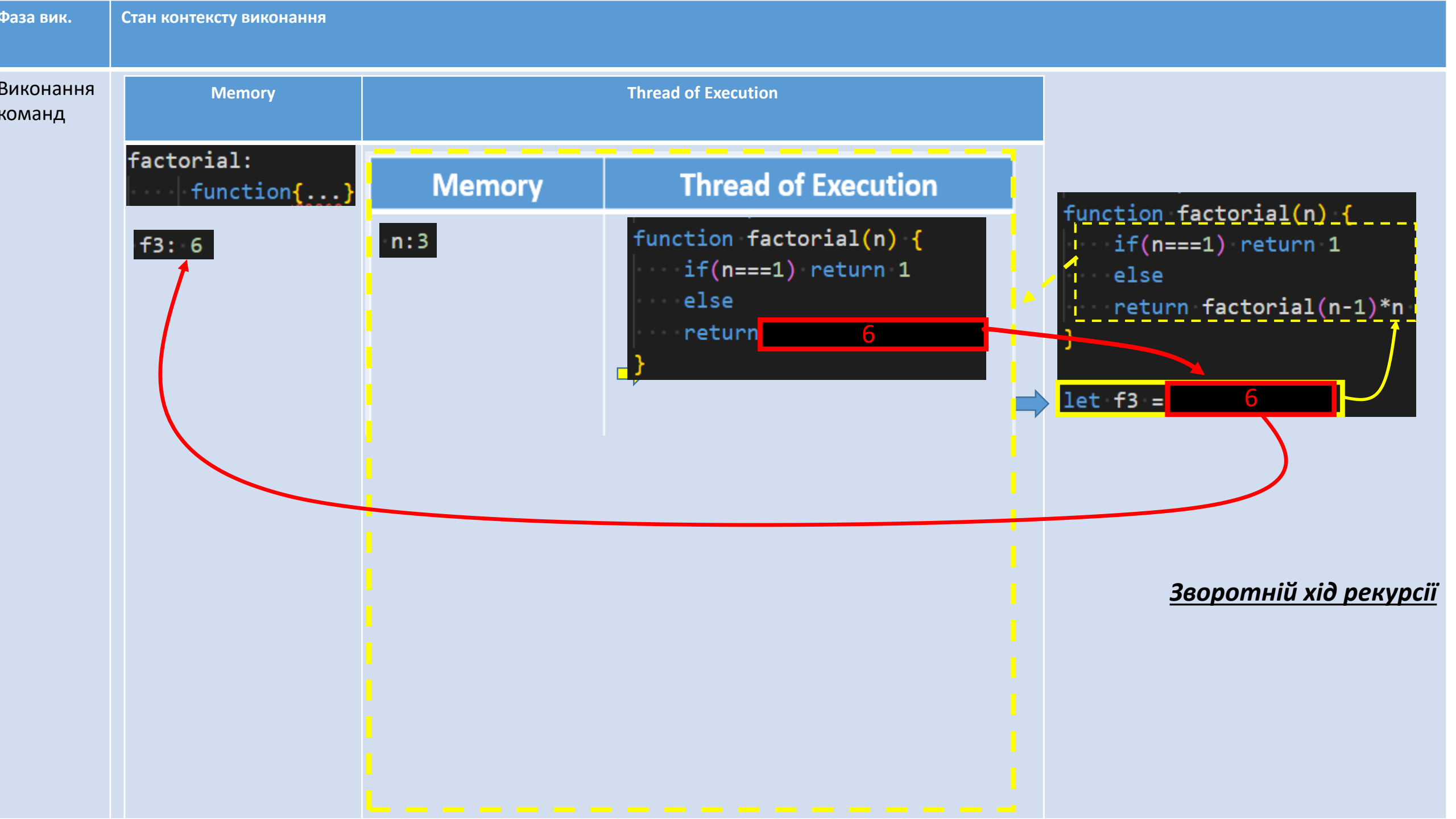
Зворотній хід рекурсії











| Фаза вик. | Стан контексту виконання | | |
|------------------|--|---|---|
| Виконання команд | Memory | Thread of Execution | |
| | <pre>factorial: ... function{...} f3: 6</pre> | <pre>function factorial(n) { ... if(n===1) return 1 ... else ... return factorial(n-1)*n } let f3 = factorial(3)</pre> | <pre>function factorial(n) { ... if(n===1) return 1 ... else ... return factorial(n-1)*n } let f3 = factorial(3)</pre> |

Знаходження факторіала

1) Рекурсивний алгоритм знаходження факторіала

$$n! = \begin{cases} (n-1)! \cdot n, & \text{якщо } n > 1; \\ 1, & \text{якщо } n = 1. \end{cases}$$

```
function factorial(n) {  
  ... if(n===1) return 1  
  ... else  
  ... return factorial(n-1)*n  
}  
  
let f3 = factorial(3)
```

2) Нерекурсивний алгоритм знаходження факторіала

```
function factorial(n) {  
  ... let res = 1  
  ... for (let i = 2; i <= n; i++)  
  ... |   res *= i  
  ... return res  
}  
  
let f3 = factorial(3)
```



Хочу побудувати ще одну піраміду.
Треба золото !!!!!





Як знайти золото?
Де ж я ту заначку від Клеопатри заховав?





Як знайти золото?

Це неможливо знайти !!!
Це було 8 березня
З кумом заносили після святкувань





Що робити у кожній кімнат ???



Як знайти золото?



Що робити у
кожній кімнаті???



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

спочатку клона надсилати для пошуку зліва
потім клона надсилати для пошуку справа



Як знайти золото?



Що робити у
кожній кімнаті???



Як знайти золото?

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота



Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

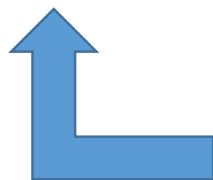
– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота



Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

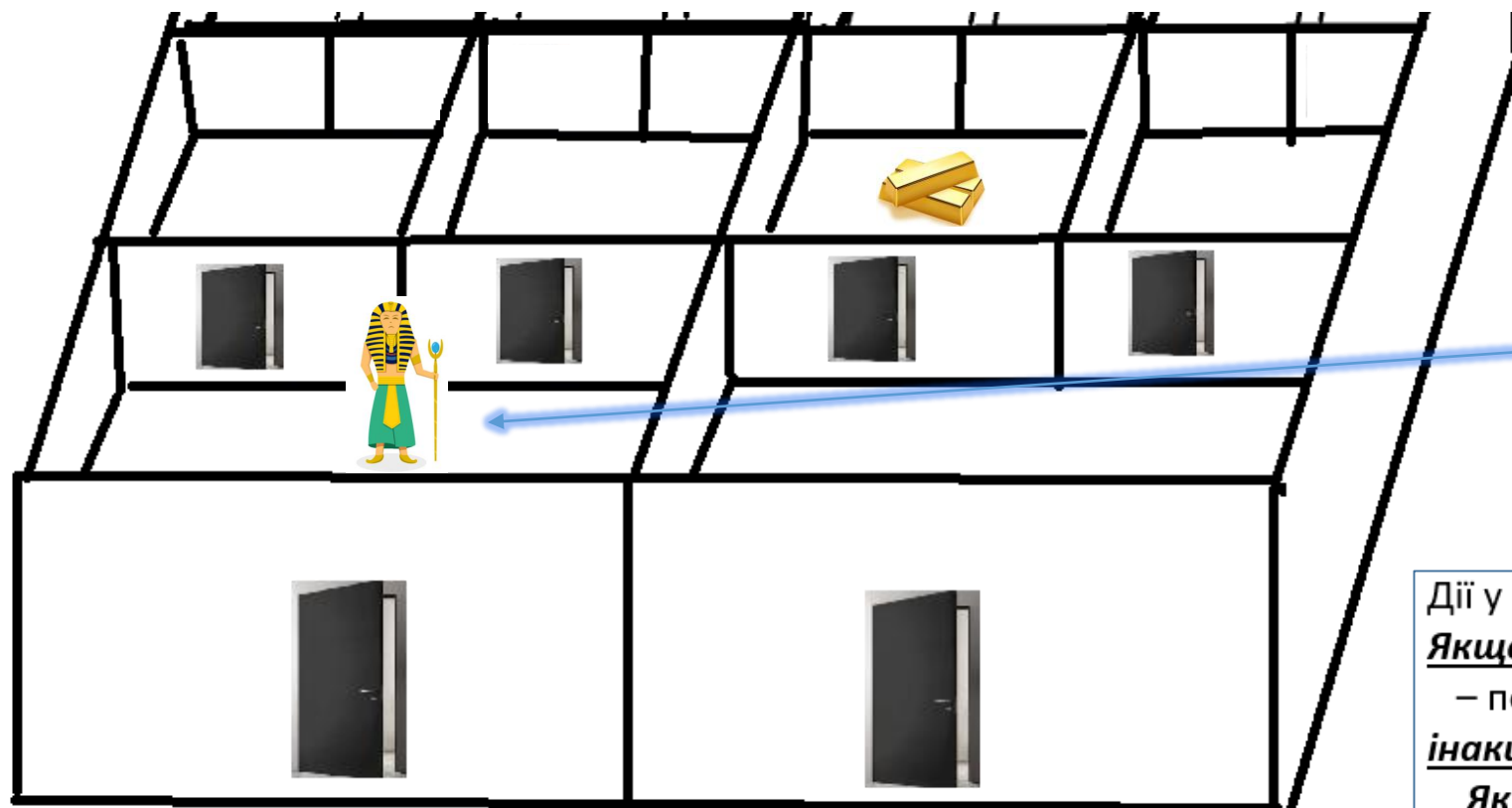
– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота



Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

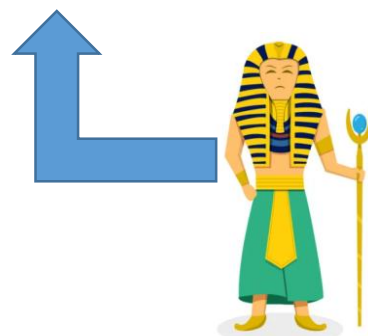
інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

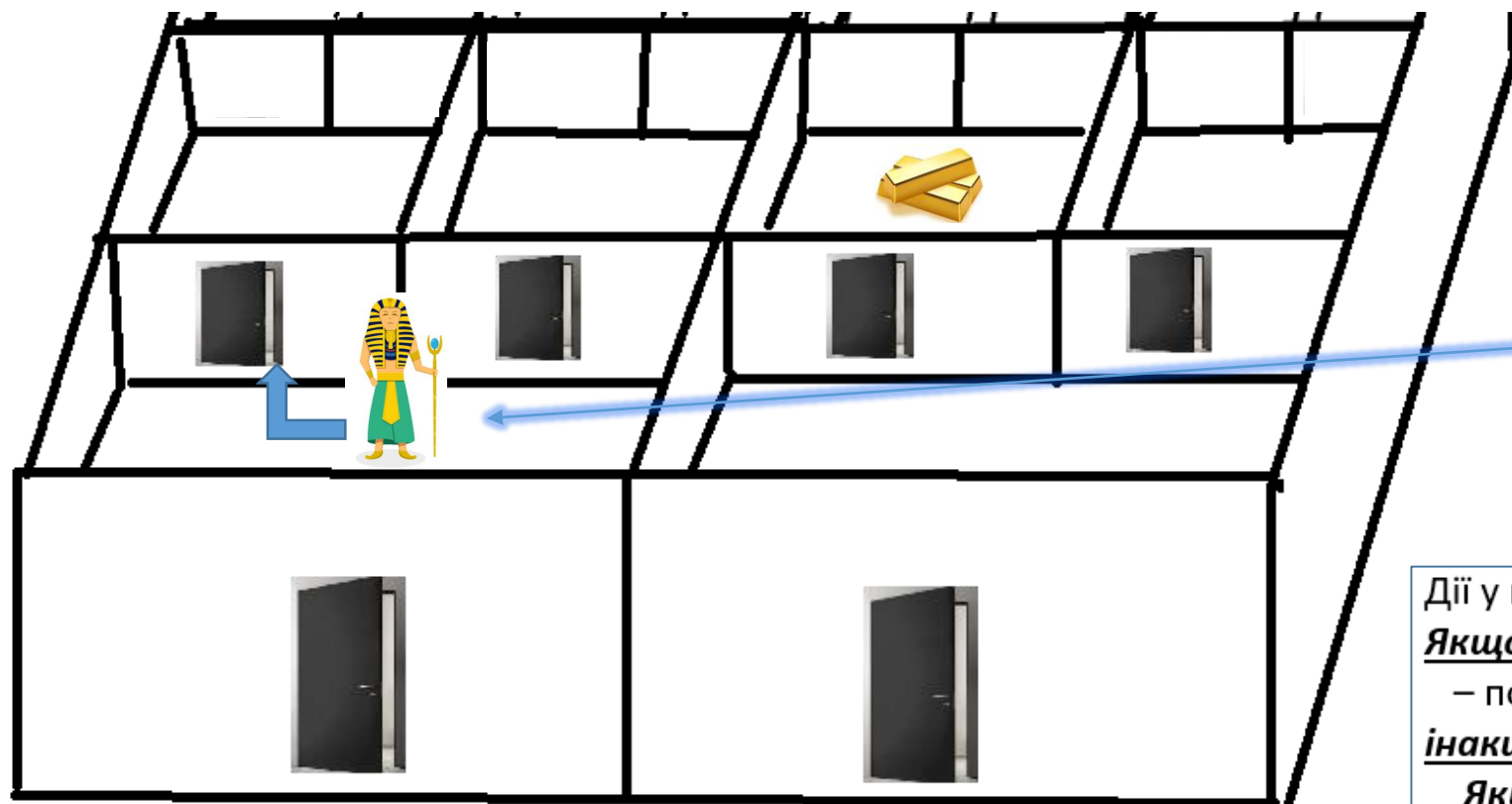
інакше

– повертаюсь звідки прийшов без золота





Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

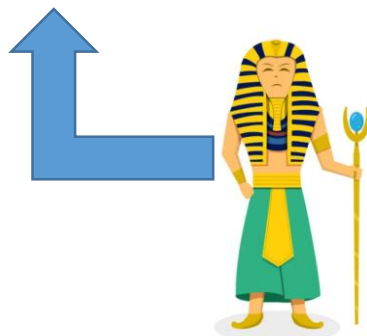
інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

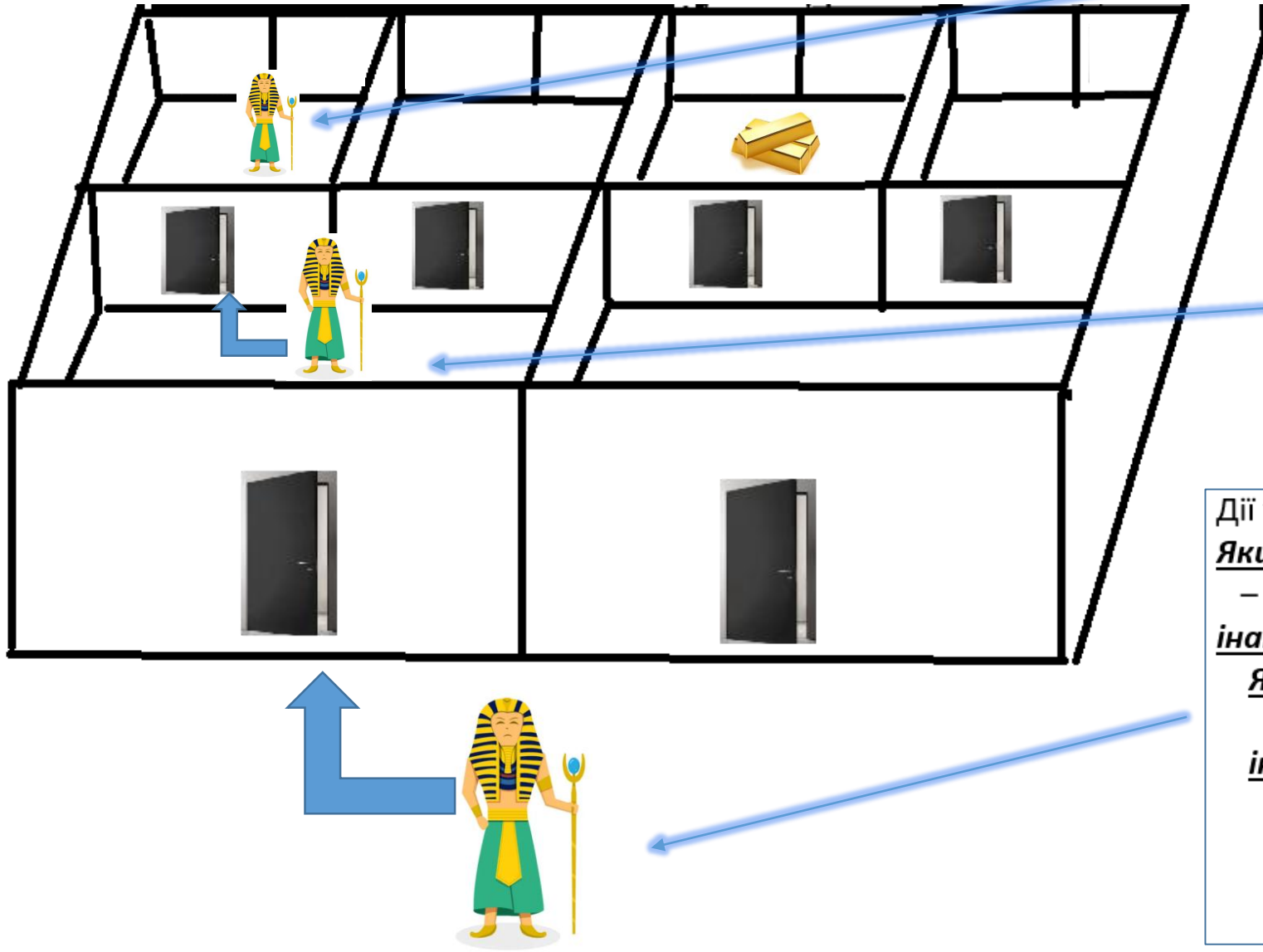
інакше

– повертаюсь звідки прийшов без золота





Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

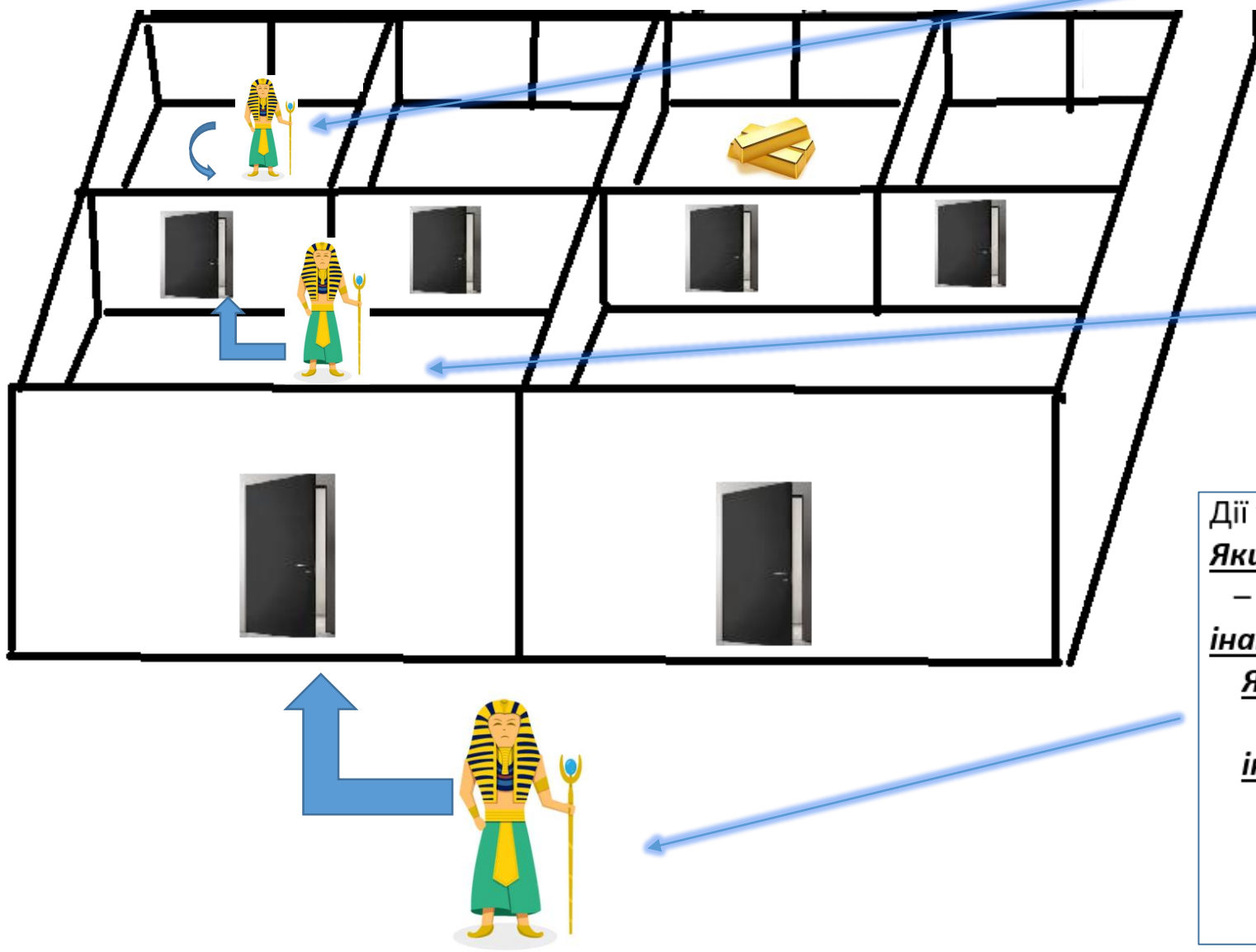
– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота



Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

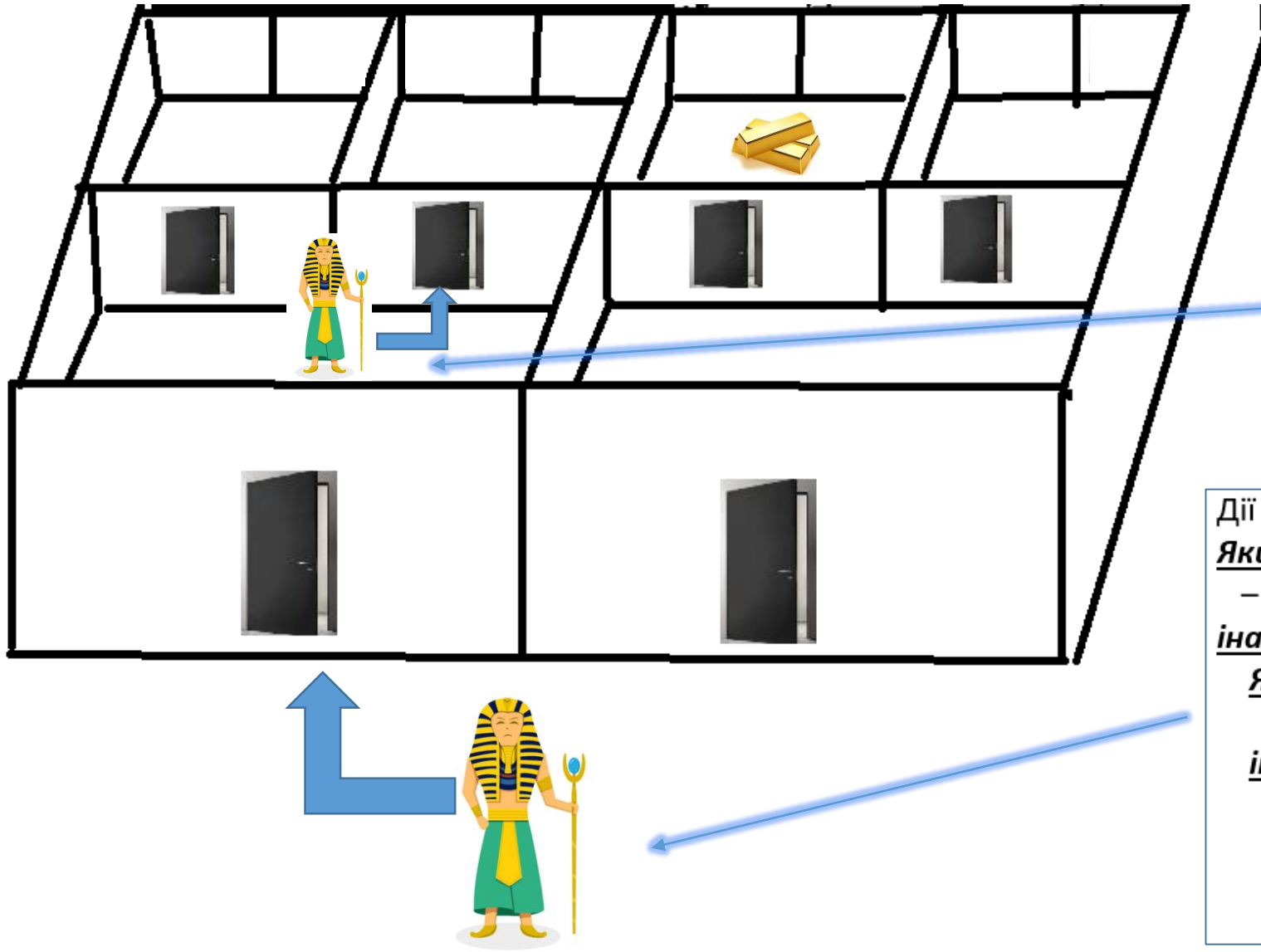
– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота



Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

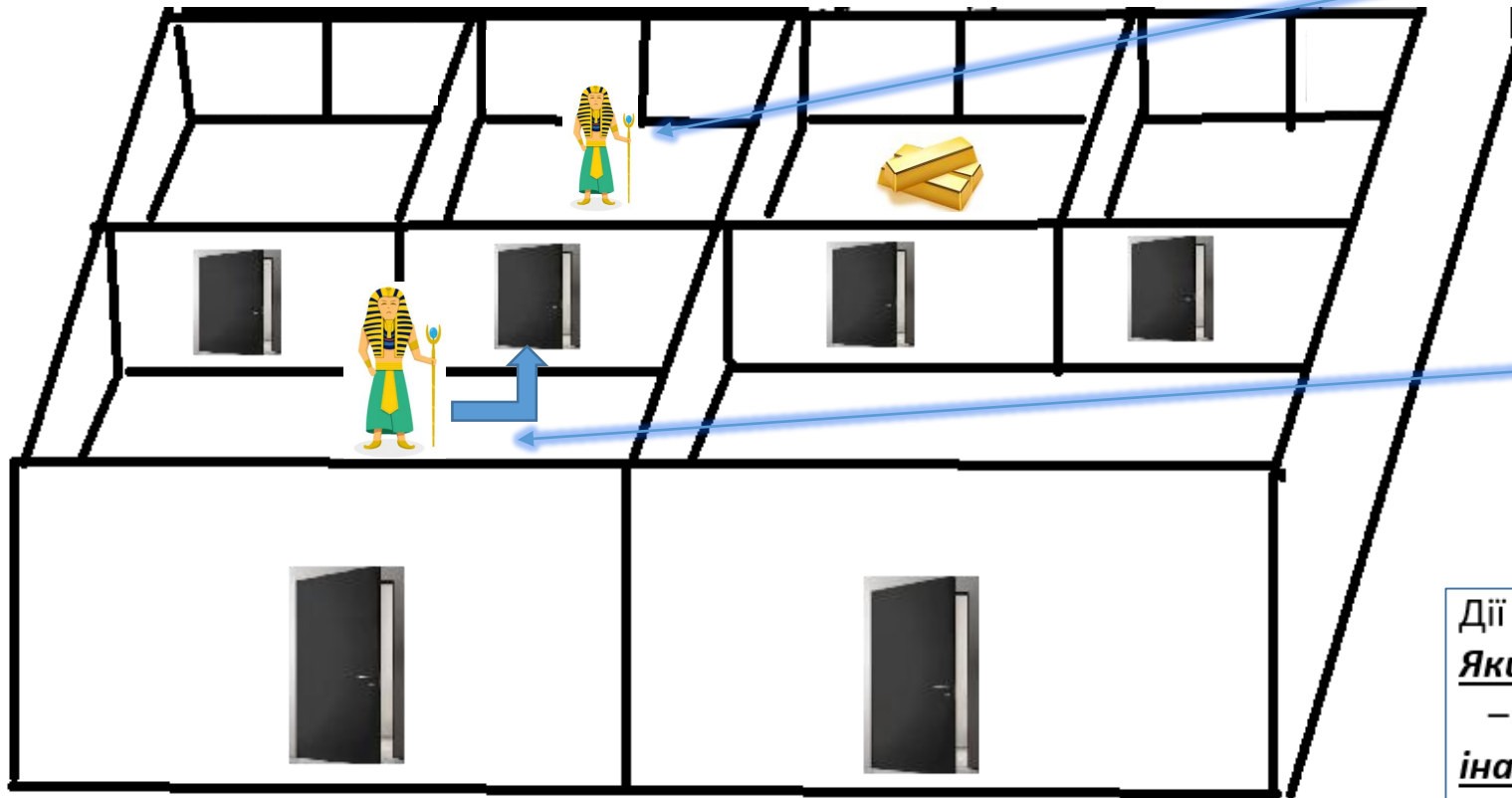
– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота



Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

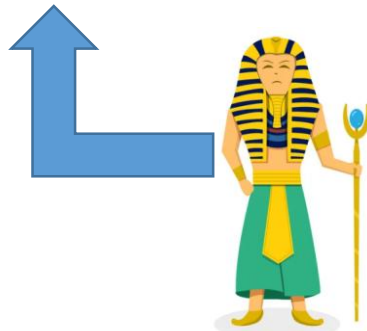
інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

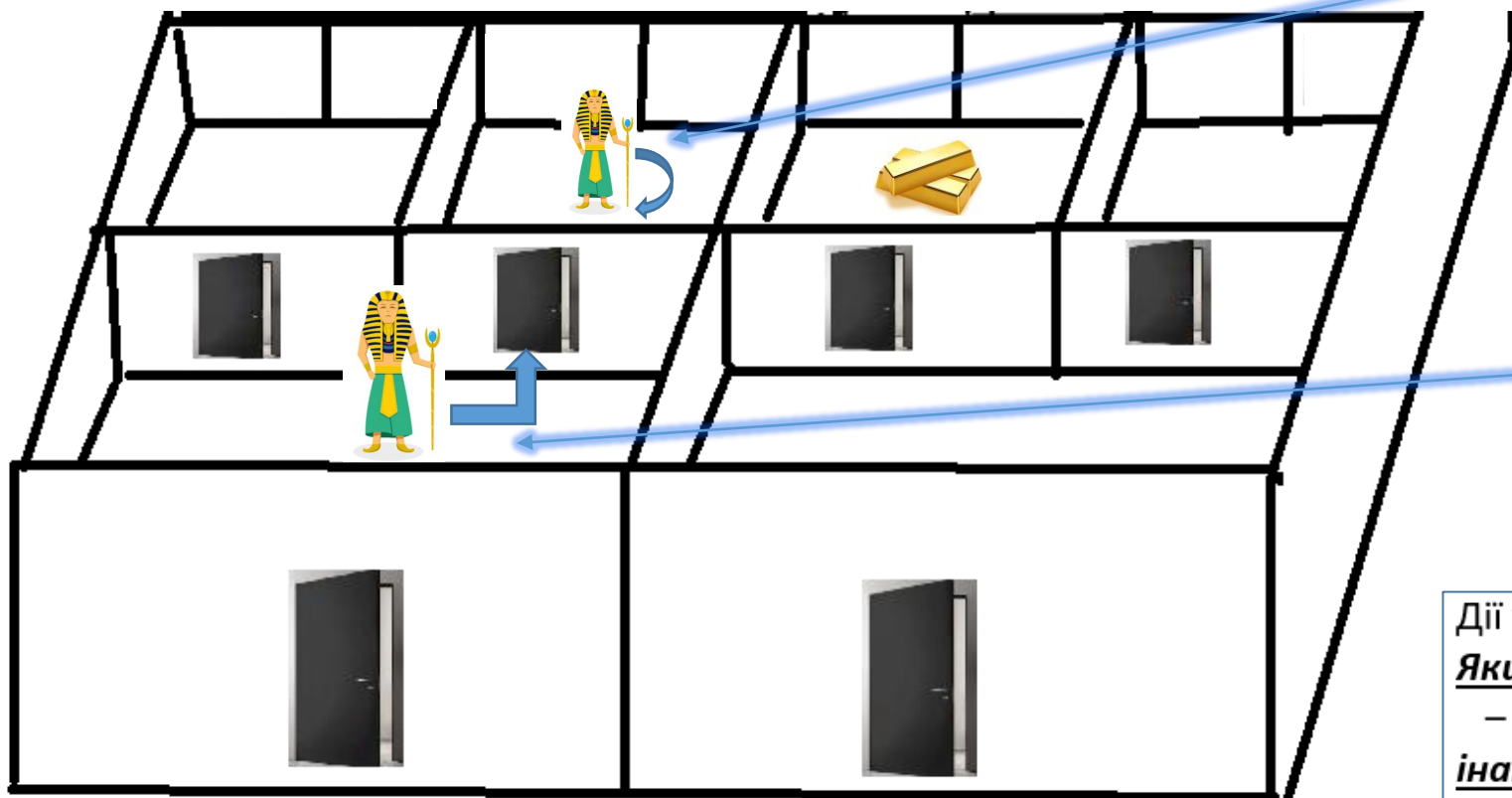
інакше

– повертаюсь звідки прийшов без золота





Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

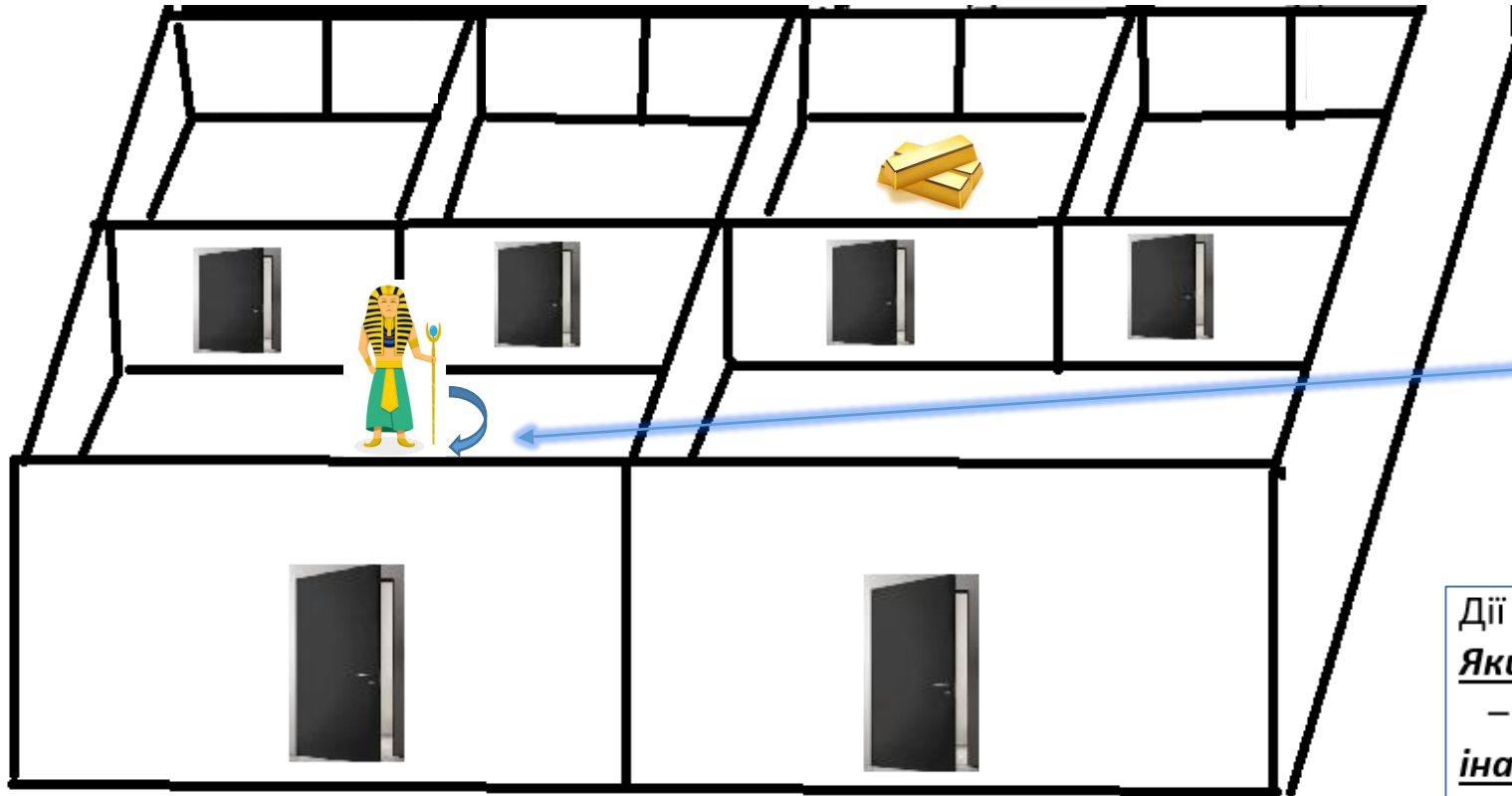
– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота



Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

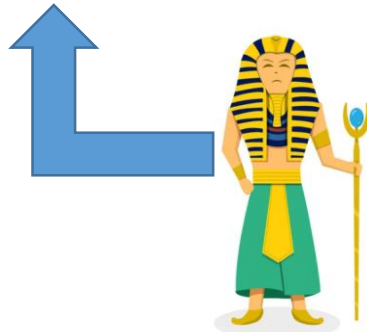
інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота





Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золотом



Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

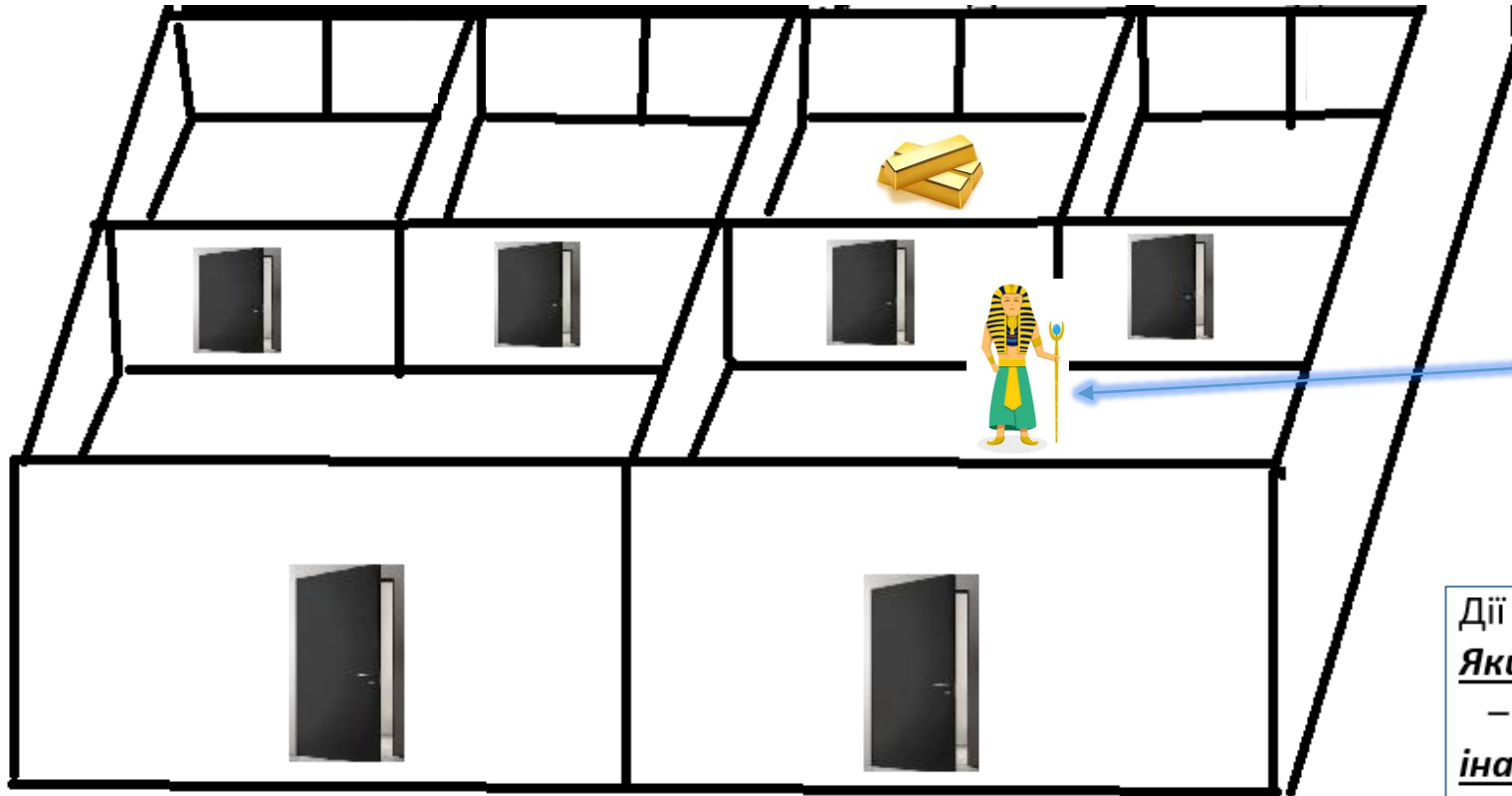
– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота



Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

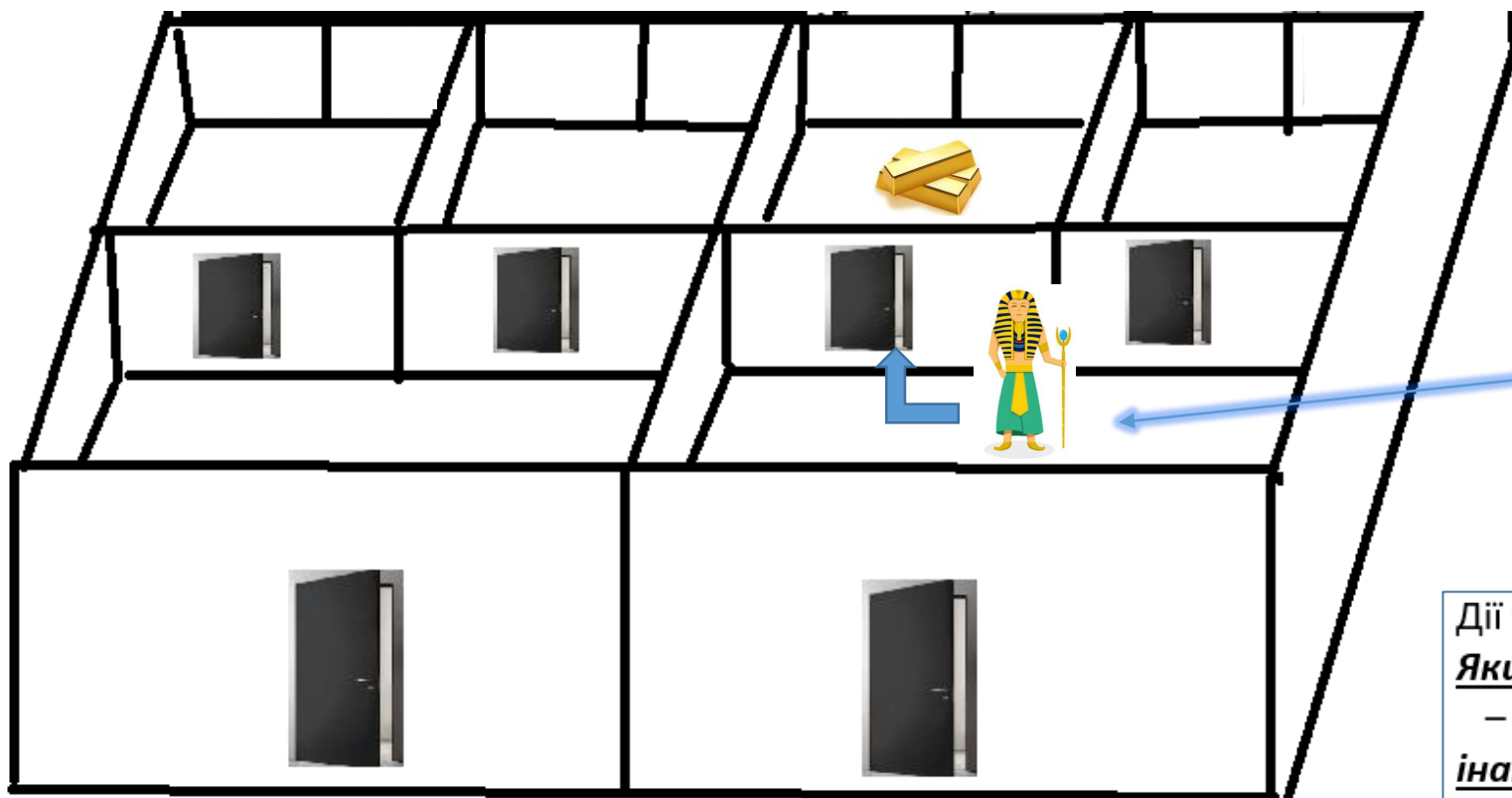
інакше

– повертаюсь звідки прийшов без золота





Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

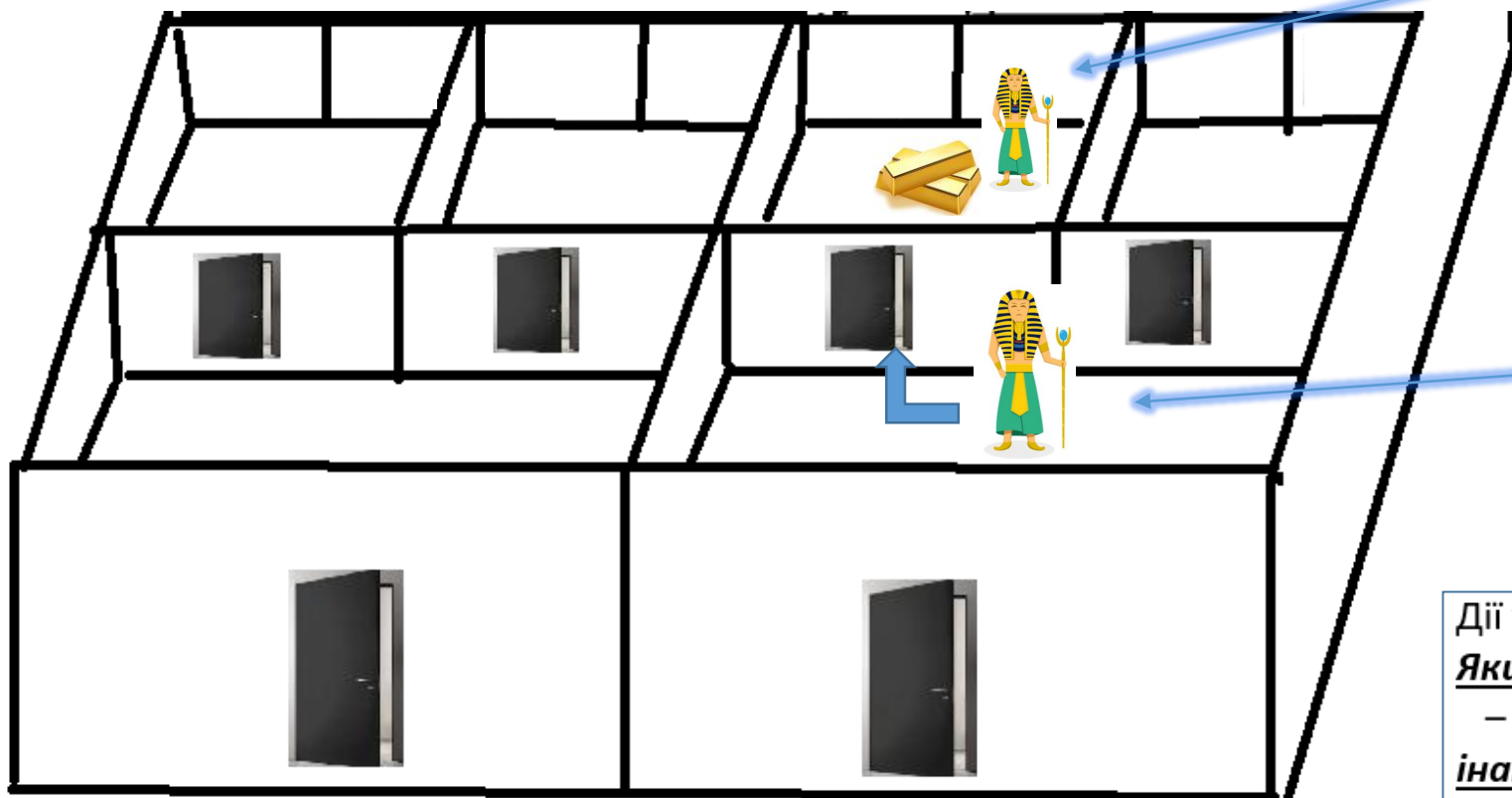
інакше

– повертаюсь звідки прийшов без золота





Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

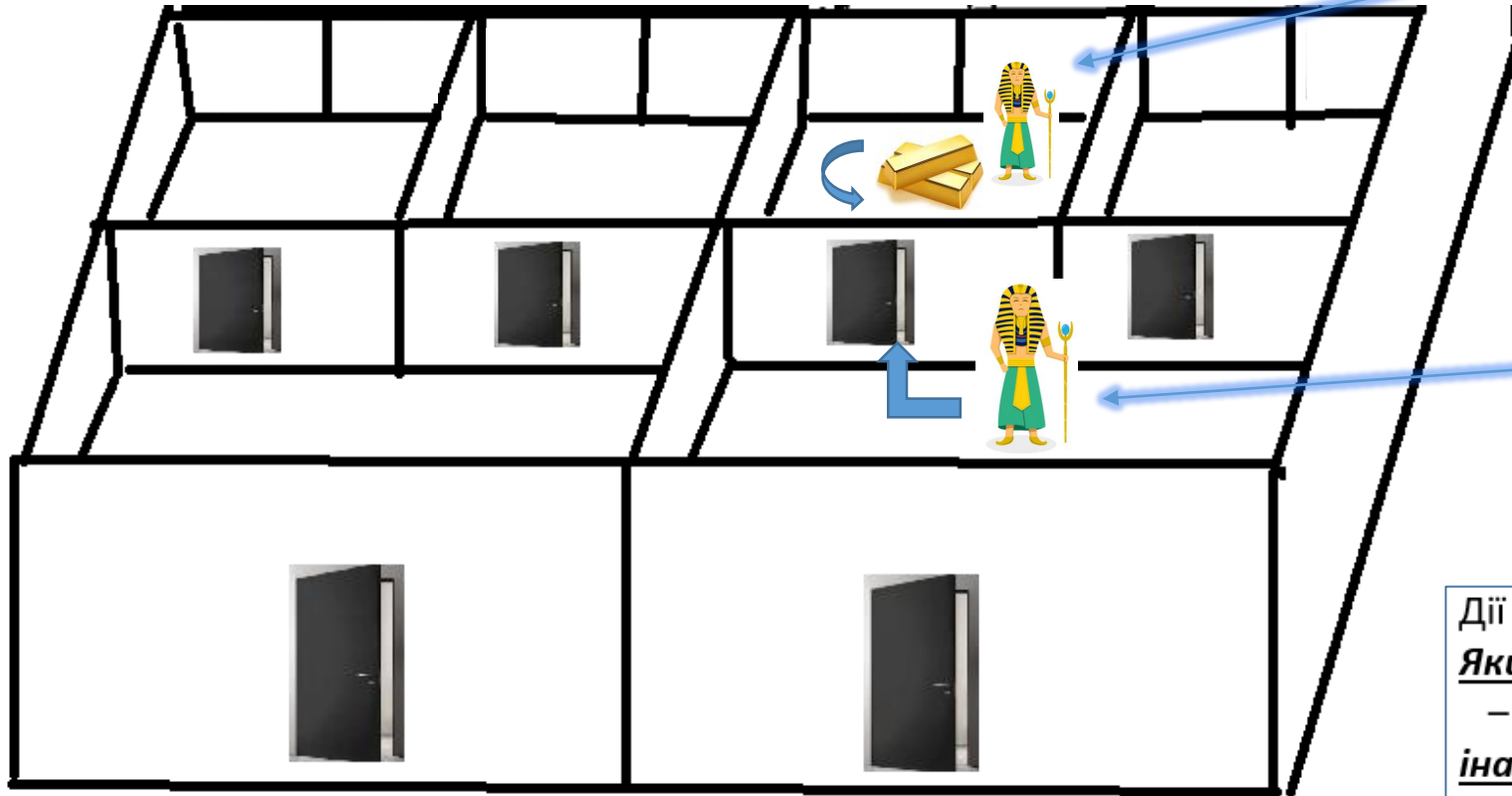
інакше

– повертаюсь звідки прийшов без золота





Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

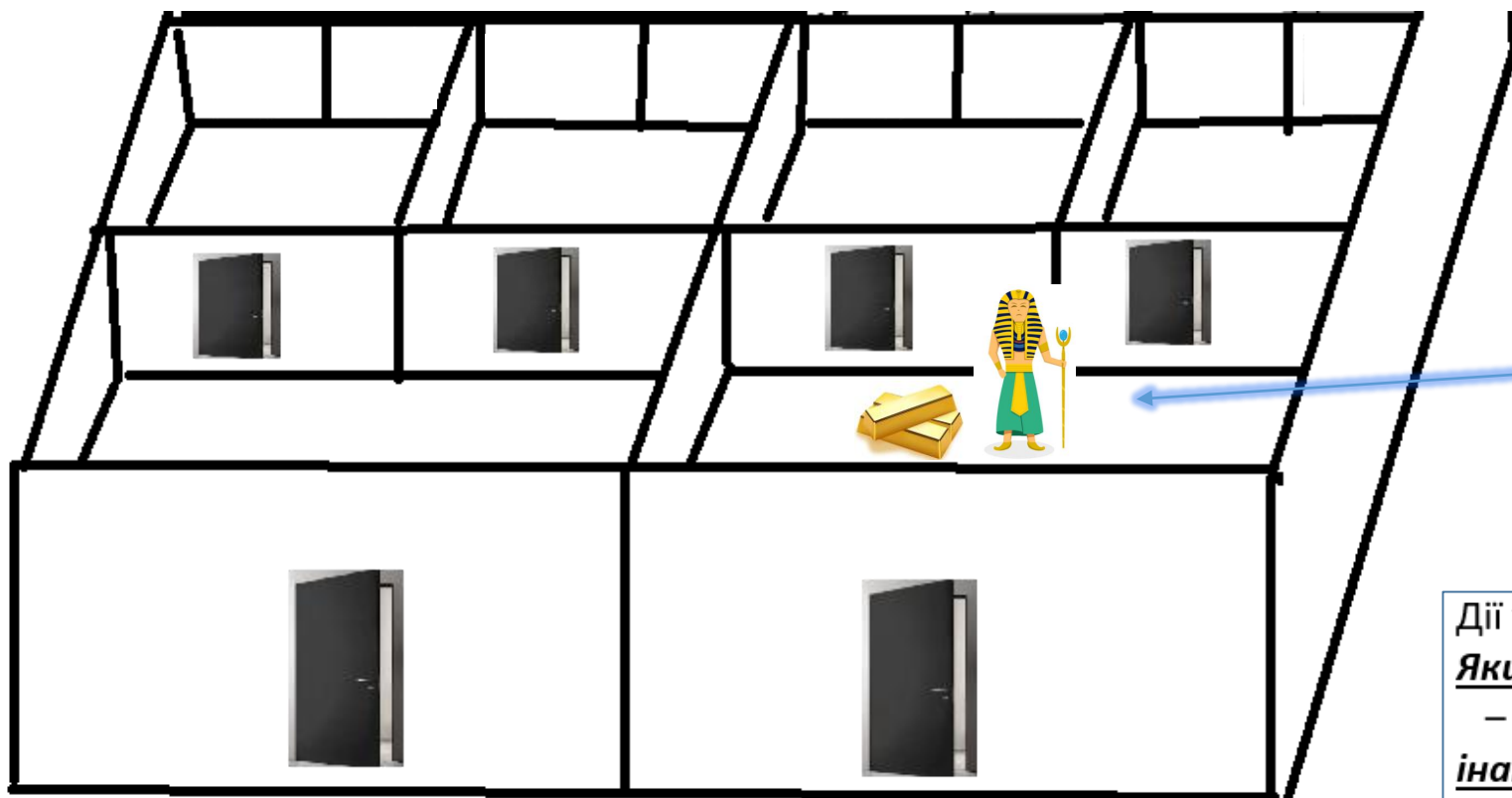
– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота



Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

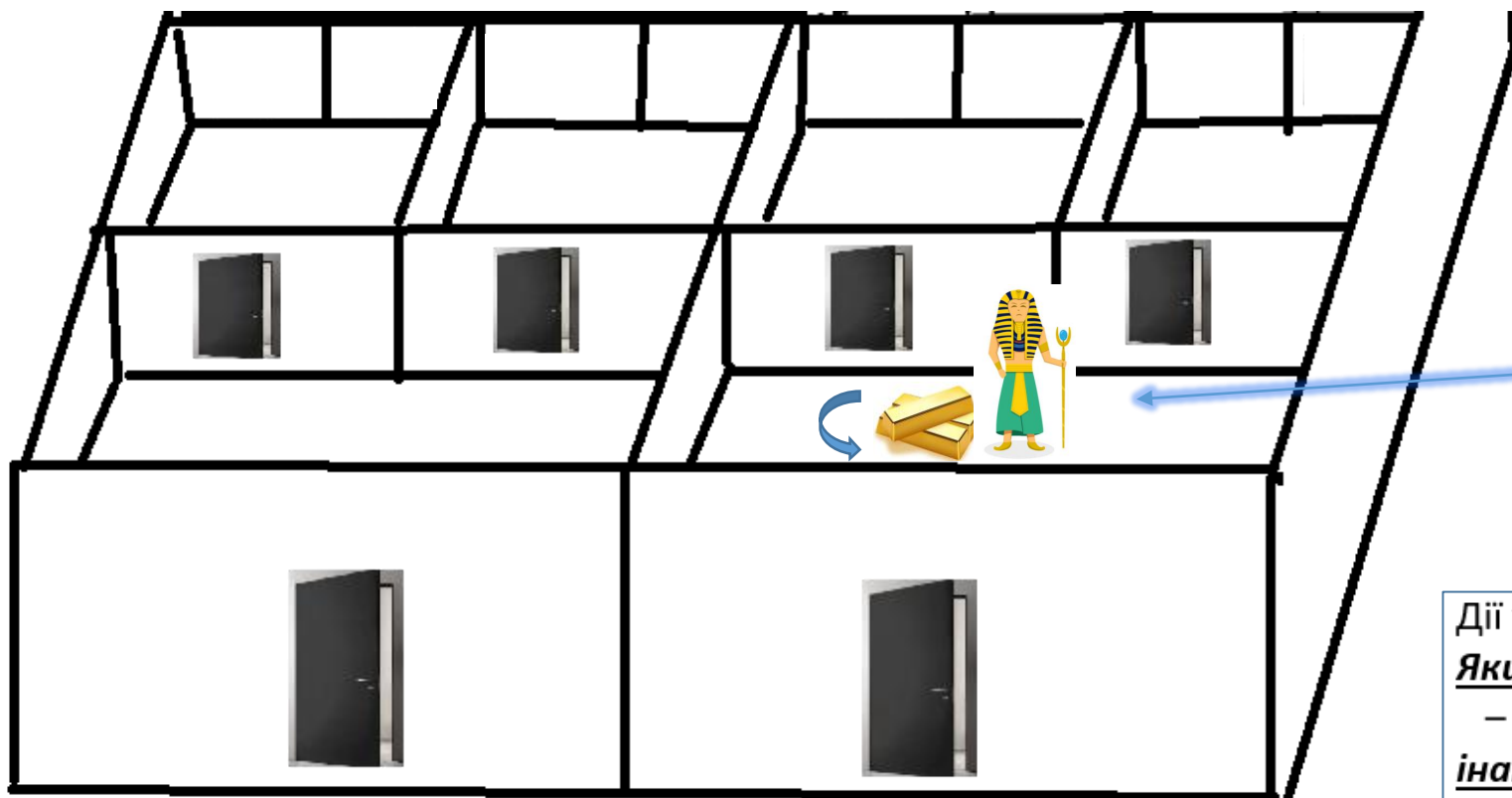
інакше

– повертаюсь звідки прийшов без золота





Як знайти золото?



Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

інакше

– повертаюсь звідки прийшов без золота

Дії у кожній кімнаті:

Якщо є золото

– повертаюсь звідки прийшов

інакше

Якщо клон знайде золото зліва

– повертаюсь звідки прийшов з золотом

інакше

Якщо клон знайде золото справа

– повертаюсь звідки прийшов з золотом

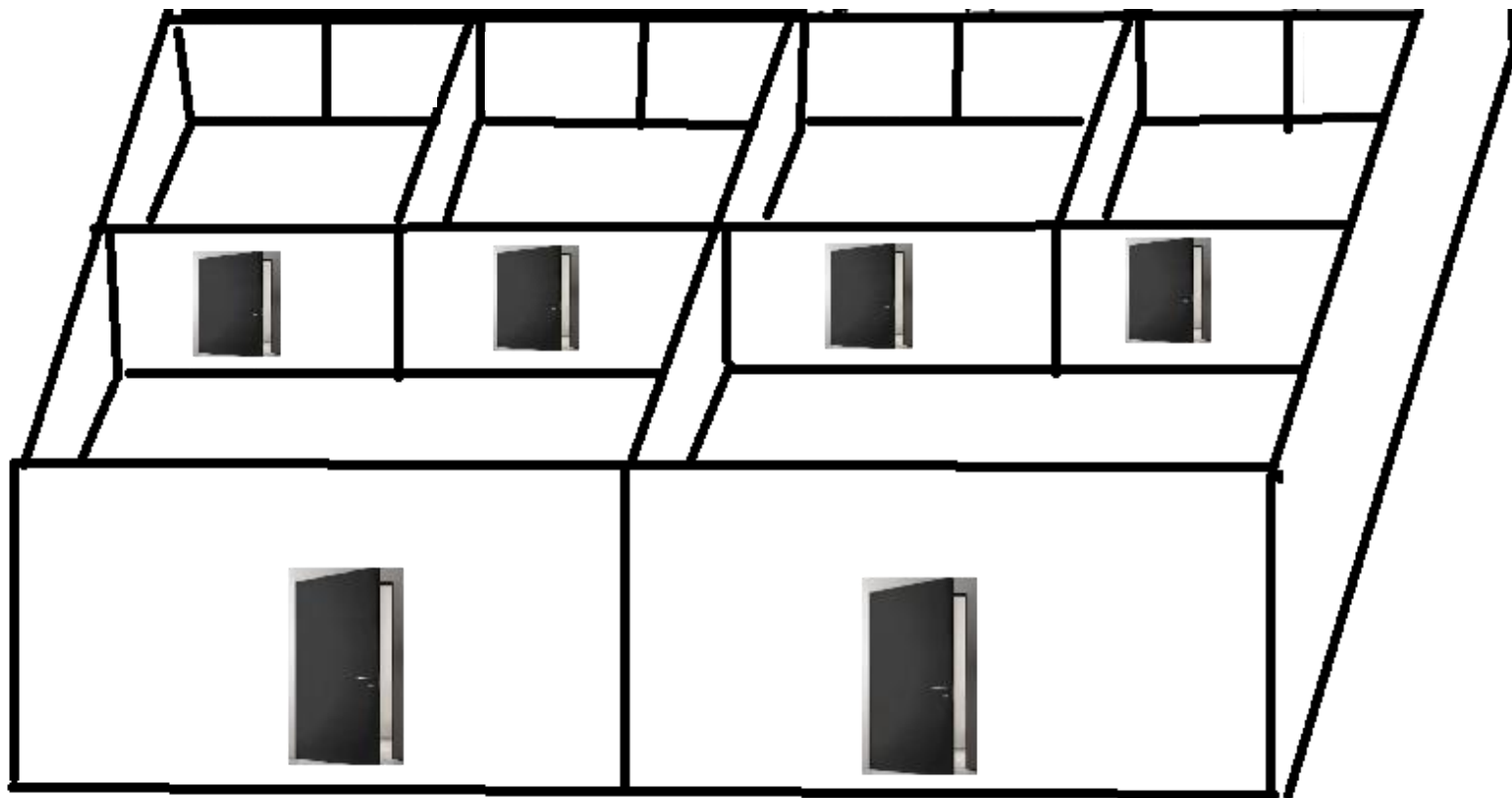
інакше

– повертаюсь звідки прийшов без золота





Як знайти золото?



Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0

0,0,1

0,1,0

0,1,1

1,0,0

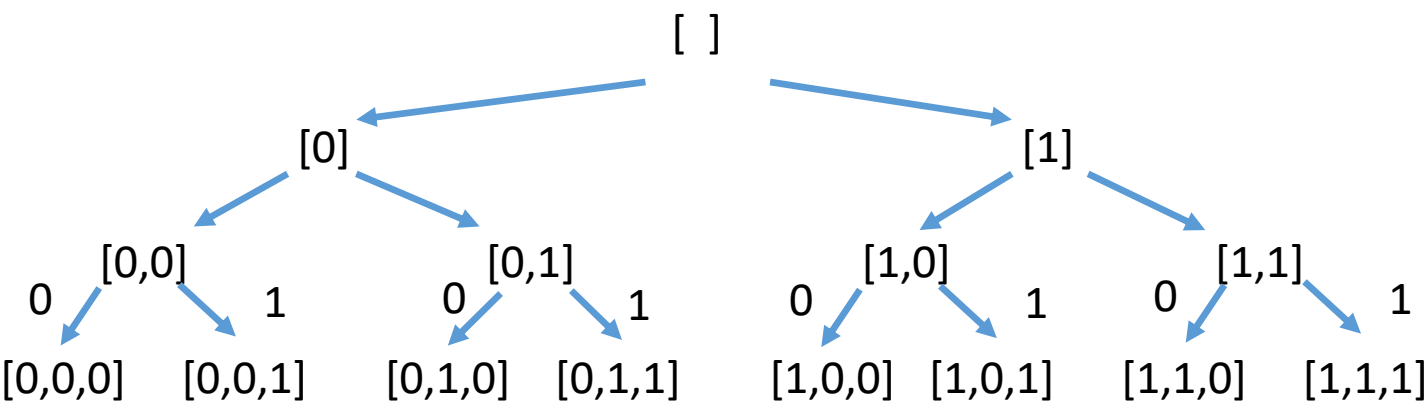
1,0,1

1,1,0

1,1,1

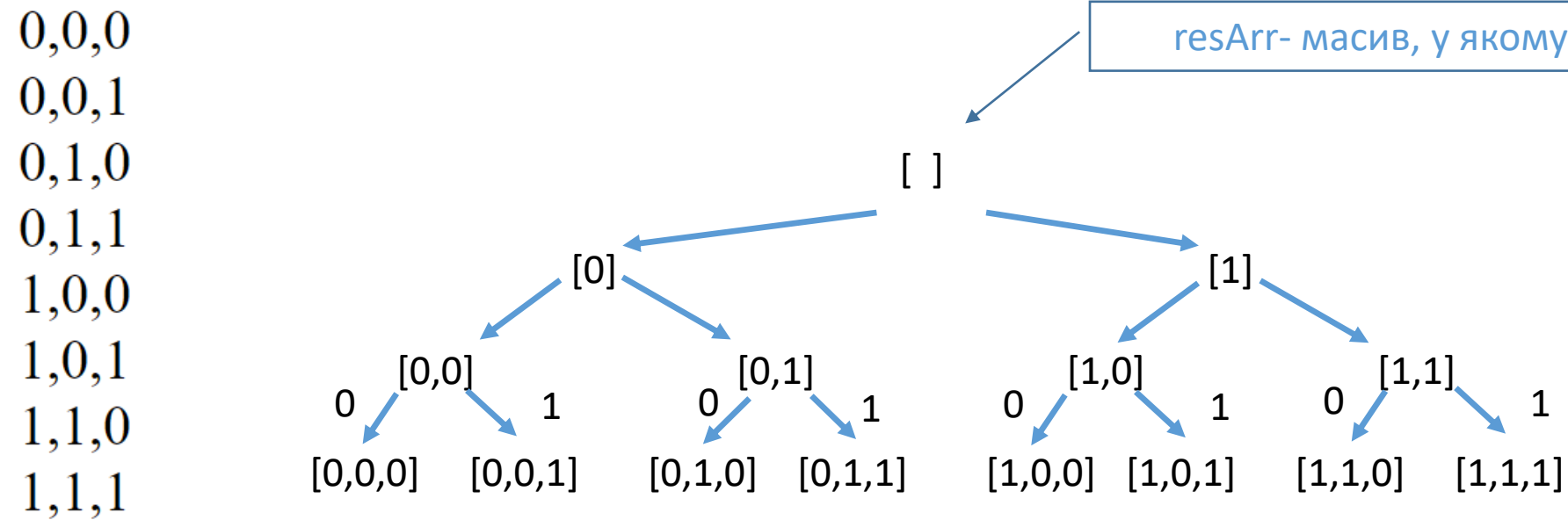
Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



step=0
step=1
step=2

Відобразити усі комбінації з нулів і одиниць у 3 позиціях



```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

```
nextB(0, [])
```

На кожному кроці виконуємо 2 команди:

Команда 1. До попереднього масиву додаємо 0 і переходимо до наступного кроку

Команда 2. До попереднього масиву додаємо 1 і переходимо до наступного кроку

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1

[]

resArr- масив, у якому будемо зберігати послідовність

```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

nextB(0, [])

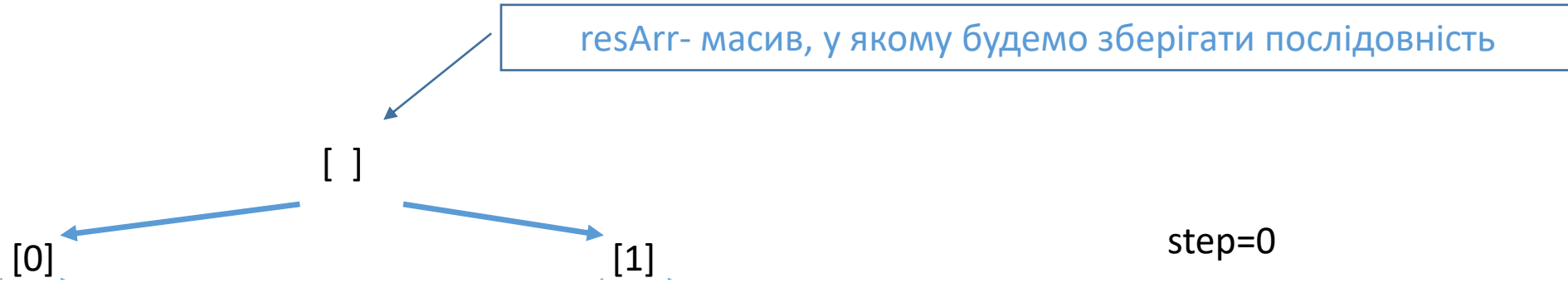
На кожному кроці виконуємо 2 команди:

Команда 1. До попереднього масиву додаємо 0 і переходимо до наступного кроку

Команда 2. До попереднього масиву додаємо 1 і переходимо до наступного кроку

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

nextB(0, [])

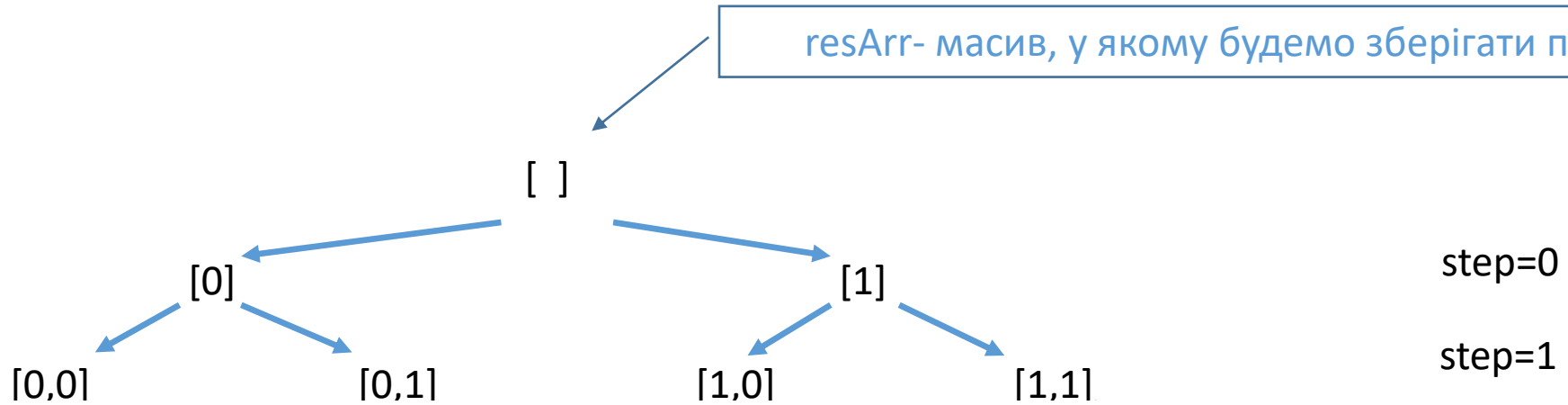
На кожному кроці виконуємо 2 команди:

Команда 1. До попереднього масиву додаємо 0 і переходимо до наступного кроку

Команда 2. До попереднього масиву додаємо 1 і переходимо до наступного кроку

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

nextB(0, [])

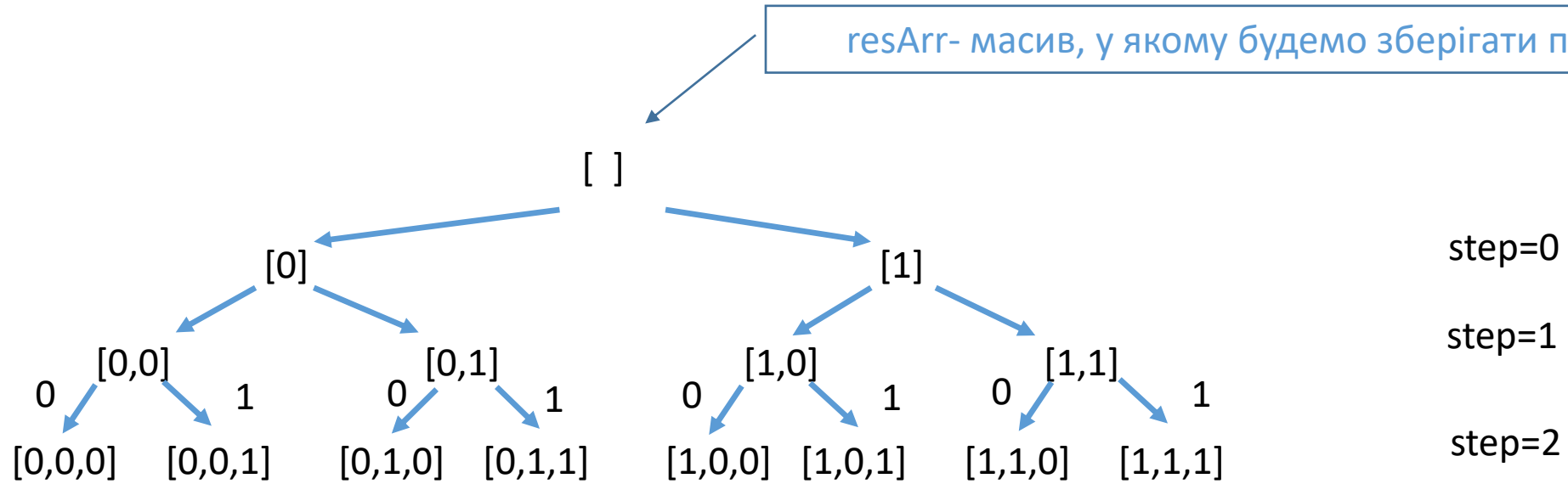
На кожному кроці виконуємо 2 команди:

Команда 1. До попереднього масиву додаємо 0 і переходимо до наступного кроку

Команда 2. До попереднього масиву додаємо 1 і переходимо до наступного кроку

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

nextB(0, [])

На кожному кроці виконуємо 2 команди:

Команда 1. До попереднього масиву додаємо 0 і переходимо до наступного кроку

Команда 2. До попереднього масиву додаємо 1 і переходимо до наступного кроку

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0

0,0,1

0,1,0

0,1,1

1,0,0

1,0,1

1,1,0

1,1,1

[]



resArr- масив, у якому будемо зберігати послідовність

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1

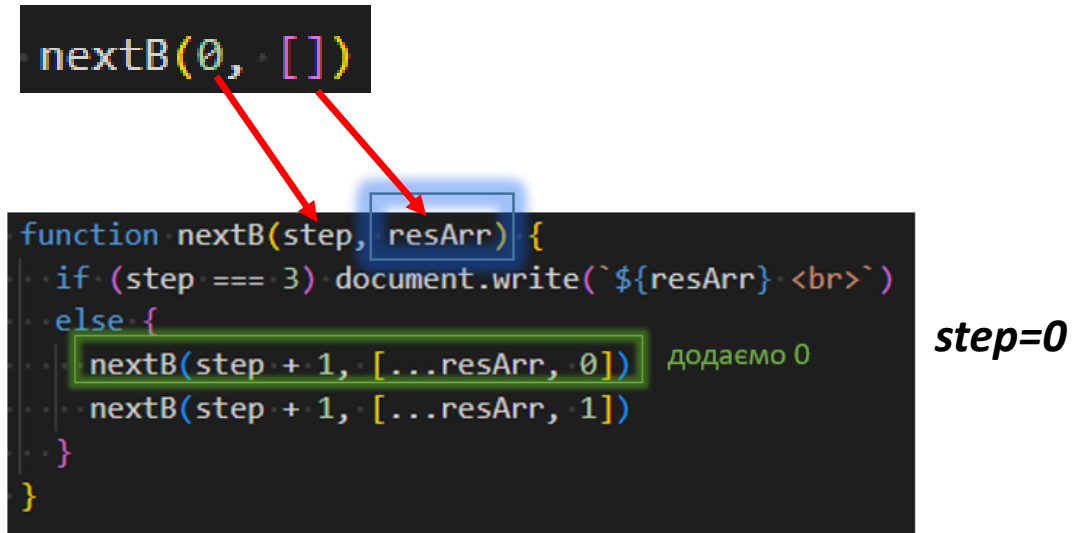


step=0

```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}  
  
nextB(0, [])
```


Відобразити усі комбінації з нулів і одиниць у 3 позиціях

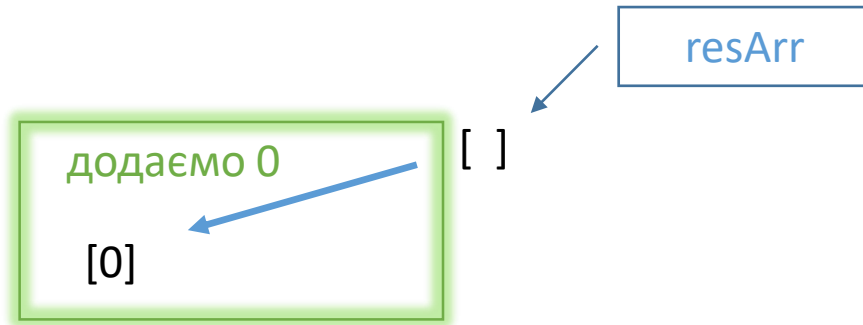
0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1

step=0



[]

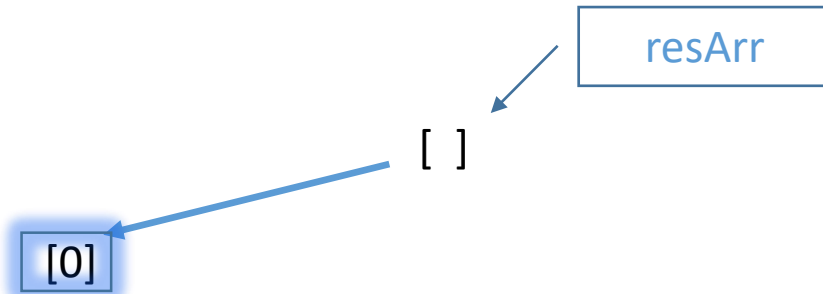
```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

додаємо 0

step=0

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

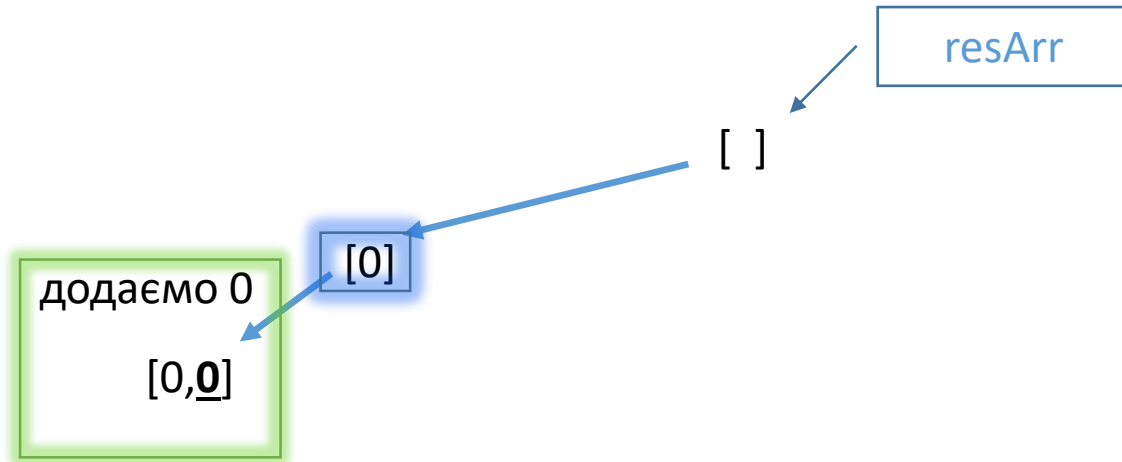
step=0

```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

step=1

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

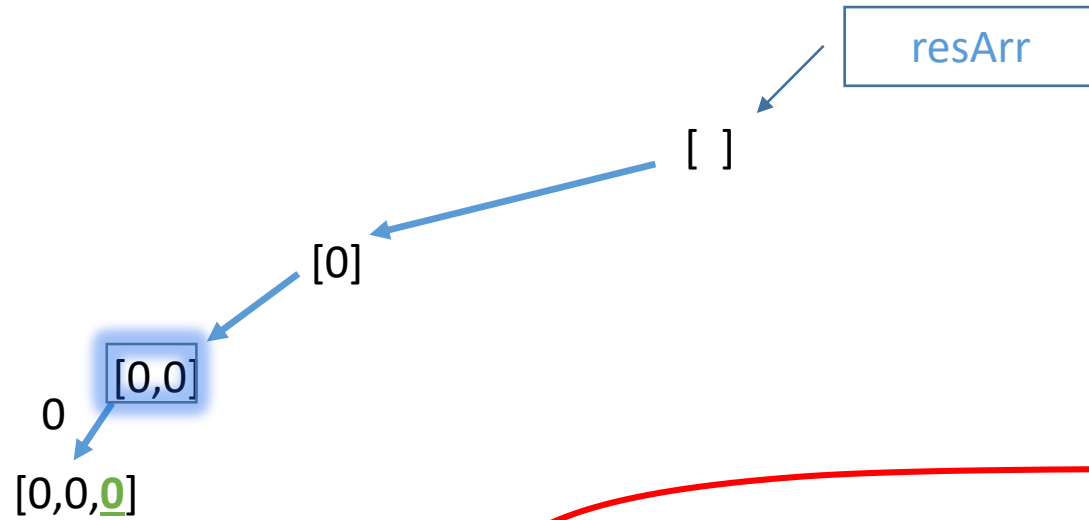
step=0

```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

step=1

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=0

```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

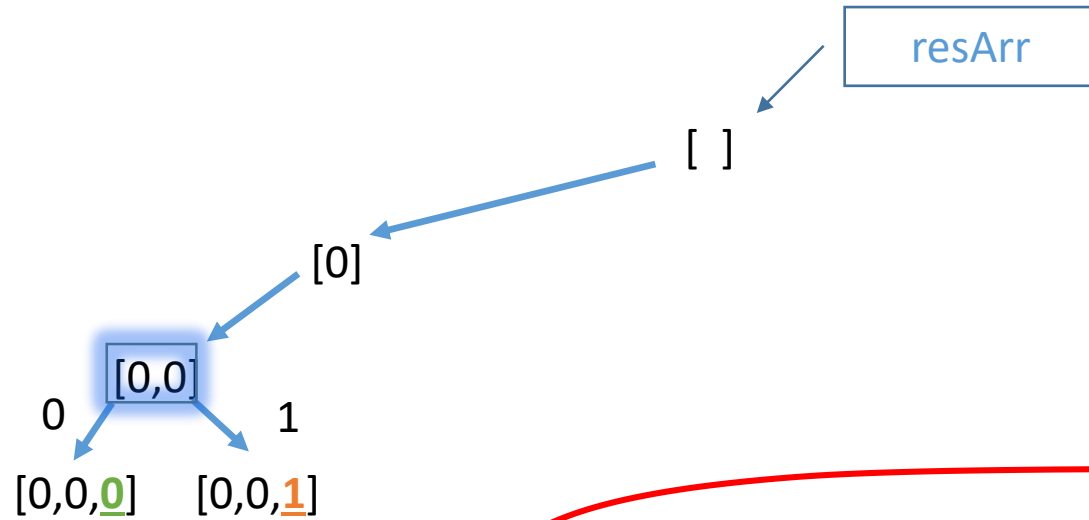
step=1

```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=2

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



`function nextB(step, resArr) {`
 `if (step === 3) document.write(`${resArr}
`)`
 `else {`
 `nextB(step + 1, [...resArr, 0])` додаємо 0
 `nextB(step + 1, [...resArr, 1])`
 `}`
`}`

step=0

`function nextB(step, resArr) {`
 `if (step === 3) document.write(`${resArr}
`)`
 `else {`
 `nextB(step + 1, [...resArr, 0])` додаємо 0
 `nextB(step + 1, [...resArr, 1])`
 `}`
`}`

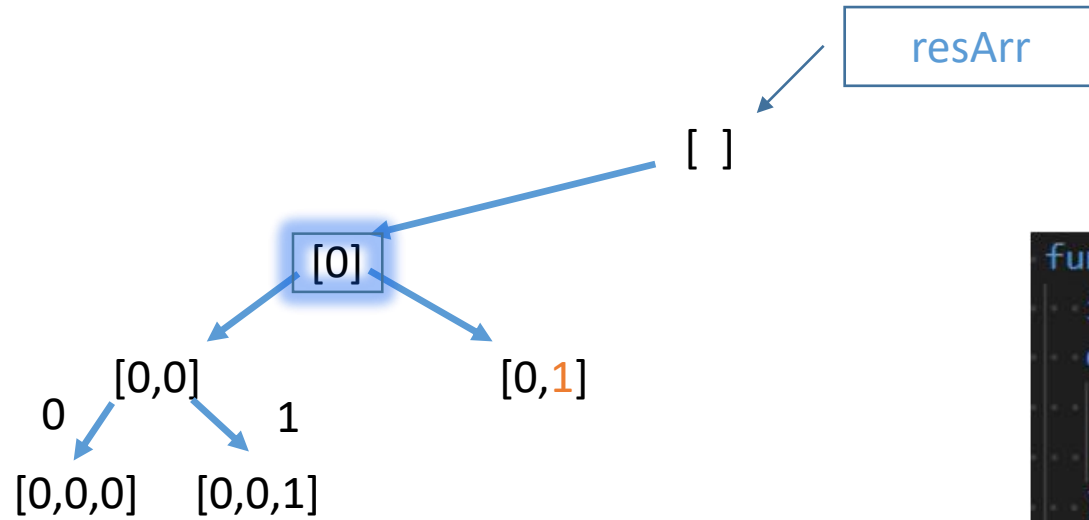
step=1

`function nextB(step, resArr) {`
 `if (step === 3) document.write(`${resArr}
`)`
 `else {`
 `nextB(step + 1, [...resArr, 0])`
 `nextB(step + 1, [...resArr, 1])` додаємо 1
 `}`
`}`

step=2

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

додаємо 0
[0]

step=0

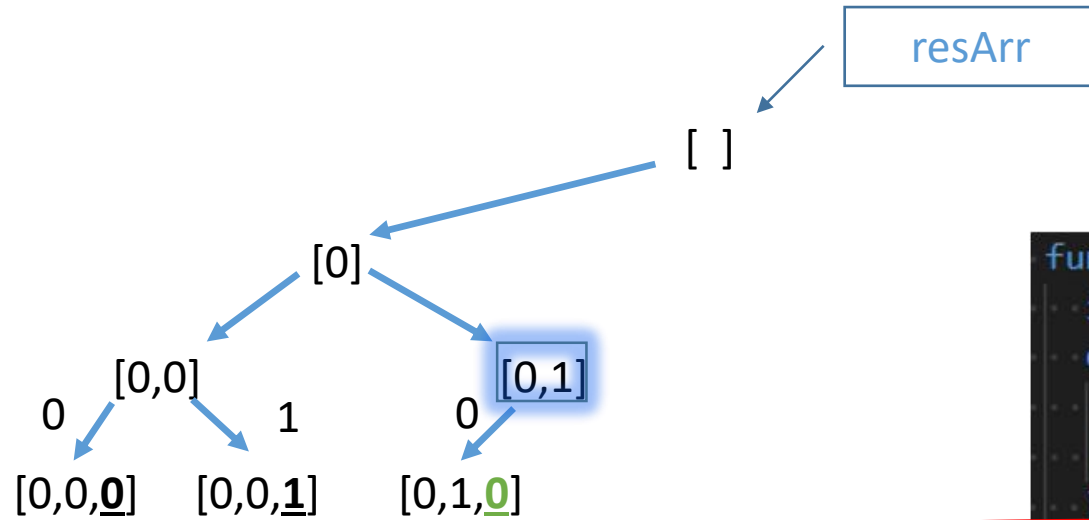
```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

додаємо 1
[0, 1]

step=1

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

step=0

```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

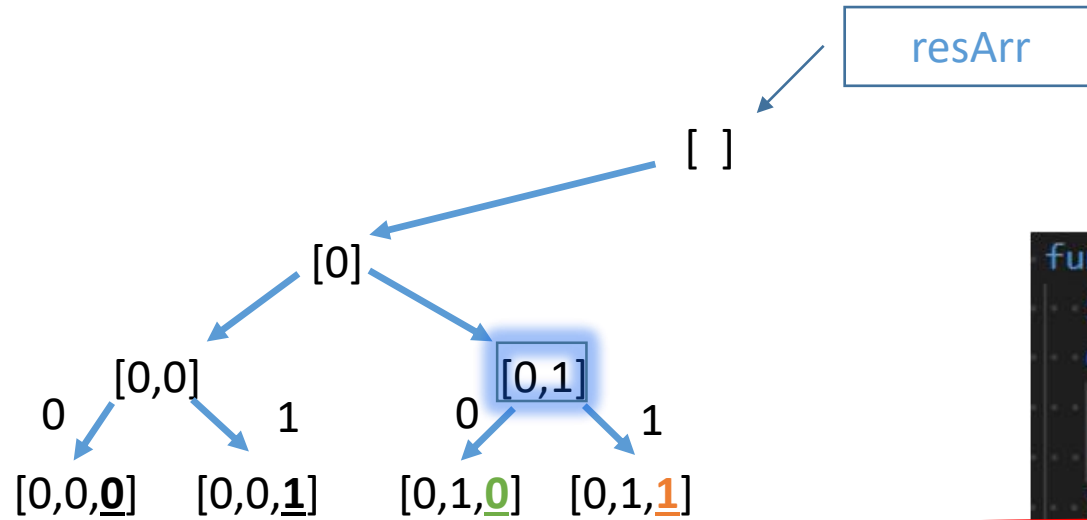
step=1

```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

step=2

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=0

Annotations: `[]` points to the initial state. The recursive calls are highlighted with a green box and labeled "додаємо 0" (adding 0). The array `[0]` is shown as the result of the first step.

```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=1

Annotations: The recursive calls are highlighted with a blue box. The array `[0, 1]` is shown as the result of the second step.

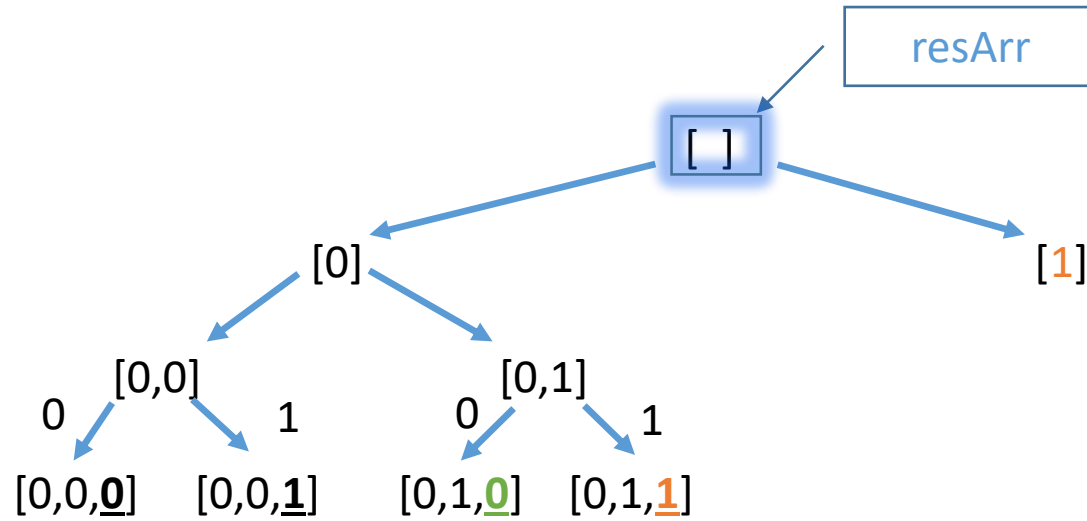
```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=2

Annotations: The recursive calls are highlighted with a blue box. The array `[0, 1, 1]` is shown as the result of the third step.

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

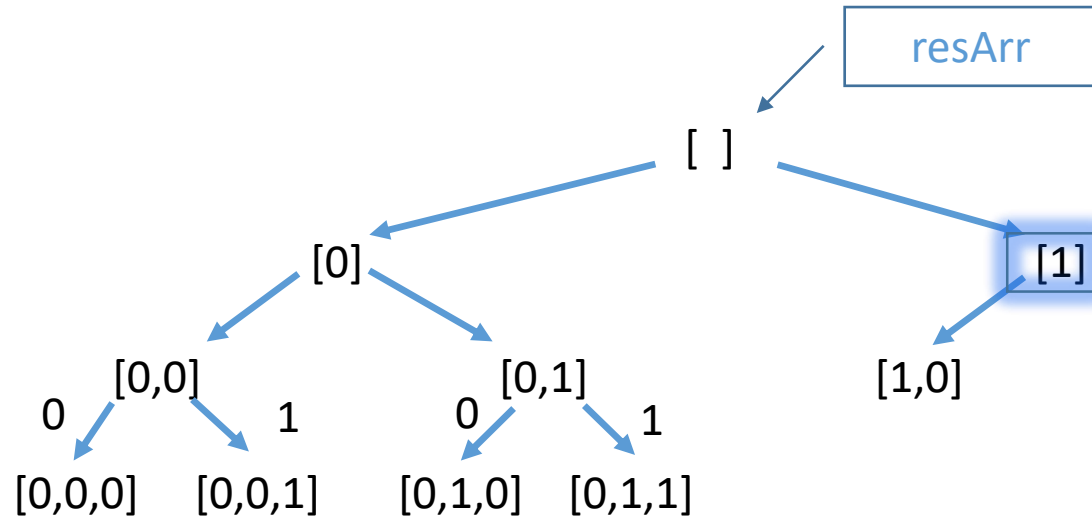
step=0

додаємо 1

[1]

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



[]

```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

додаємо 1

step=0

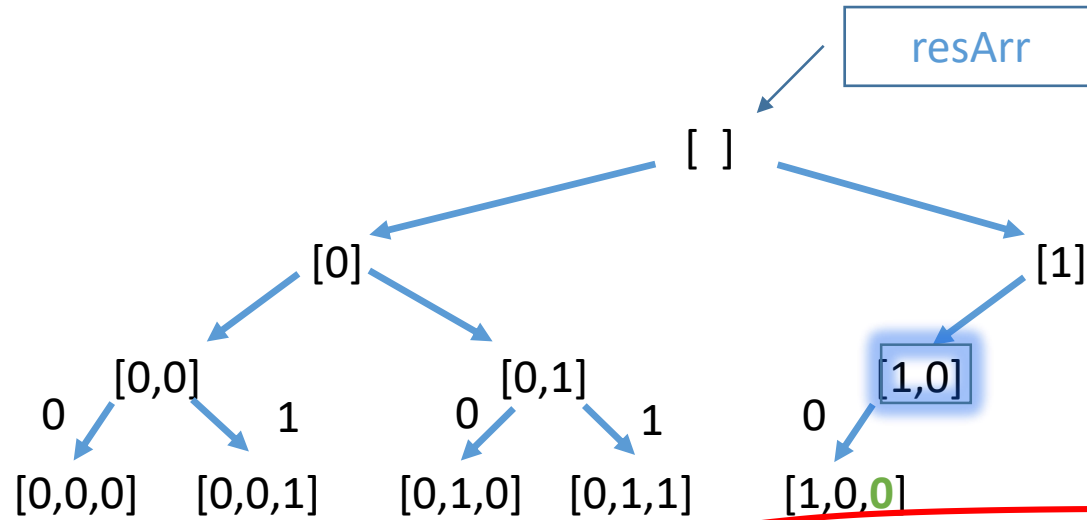
function nextB(step, resArr) {
 if (step === 3) document.write(`\${resArr}
`)
 else {
 nextB(step + 1, [...resArr, 0])
 nextB(step + 1, [...resArr, 1])
 }
}

додаємо 0

step=1

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

додаємо 1

step=0

```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

додаємо 0

step=1

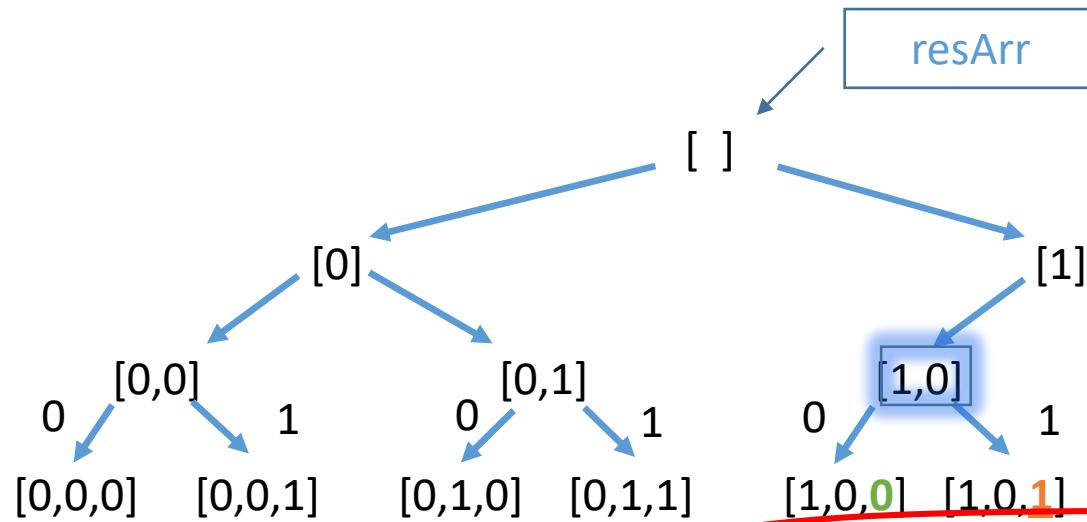
```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

додаємо 0

step=2

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

додаємо 1

step=0

```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

додаємо 0

step=1

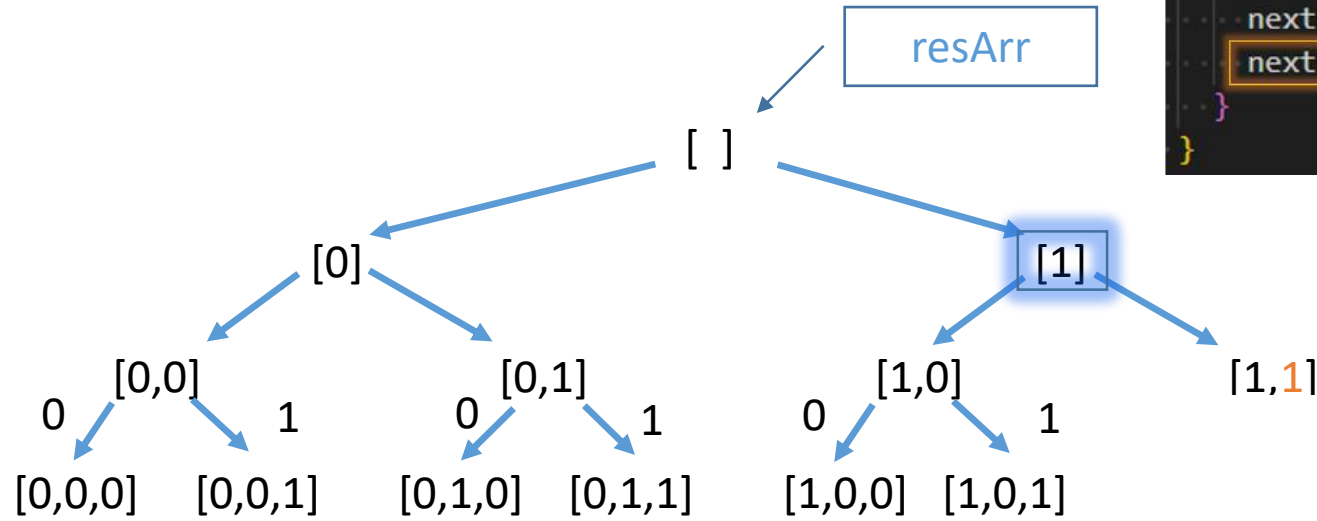
```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

додаємо 1

step=2

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

step=0

додаємо 1

[1]

```
function nextB(step, resArr) {  
  if (step === 3) document.write(`${resArr}<br>`)  
  else {  
    nextB(step + 1, [...resArr, 0])  
    nextB(step + 1, [...resArr, 1])  
  }  
}
```

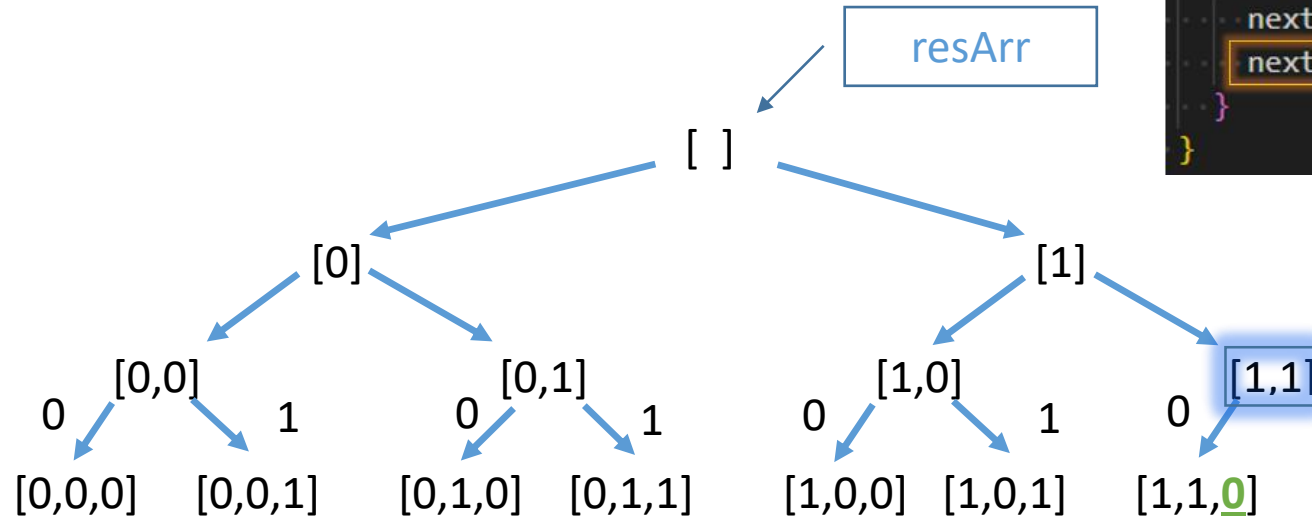
step=1

додаємо 1

[1,1]

Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=0

```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=1

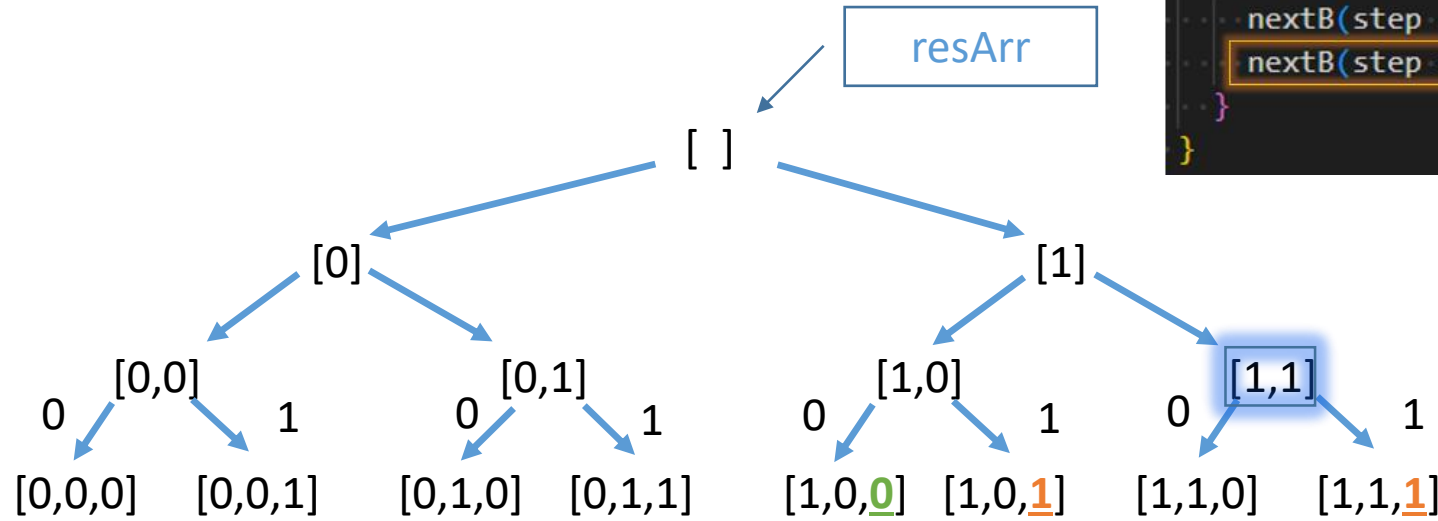
```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=2

```
nextB(0, [])
```


Відобразити усі комбінації з нулів і одиниць у 3 позиціях

0,0,0
0,0,1
0,1,0
0,1,1
1,0,0
1,0,1
1,1,0
1,1,1



```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=0

```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=1

```
function nextB(step, resArr) {
  if (step === 3) document.write(`${resArr}<br>`)
  else {
    nextB(step + 1, [...resArr, 0])
    nextB(step + 1, [...resArr, 1])
  }
}
```

step=2

nextB(0, [])

Знаходження чисел Фібоначчі

Послідовність Фібоначчі, числа Фібоначчі — у математиці числова послідовність F_n , задана рекурентним співвідношенням другого порядку

$$F_1 = 1, F_2 = 1, F_{n+2} = F_n + F_{n+1}, n = 1, 2, 3, \dots,$$

$$F_1 = 1, F_2 = 1, F_3 = 2, F_4 = 3, F_5 = 5, F_6 = 8, F_7 = 13, F_8 = 21, \dots$$

Простіше кажучи, перші два члени послідовності — одиниці, а кожний наступний — сума значень двох попередніх чисел:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \dots$$

Знаходження чисел Фібоначчі

1) Рекурсивний алгоритм

$$F_n = \begin{cases} 1, & n = 1 \\ 1, & n = 2 \\ F_{n-1} + F_{n-2}, & n > 2 \end{cases}$$

```
function fibonacci(n) {  
  if(n===1 || n===2)  
    return 1  
  else  
    return fibonacci(n-1)+fibonacci(n-2)  
}  
  
fib7 = fibonacci(7)
```

2) Нерекурсивний алгоритм

```
function fibonacci(n) {  
  if(n===1 || n===2)  
    return 1  
  else {  
    let f_n_1 = f_n_2 = 1  
    for (let i = 3; i <= n; i++) {  
      f_n = f_n_1 + f_n_2  
  
      f_n_2 = f_n_1  
      f_n_1 = f_n  
    }  
    return f_n  
  }  
}  
  
fib7 = fibonacci(7)
```

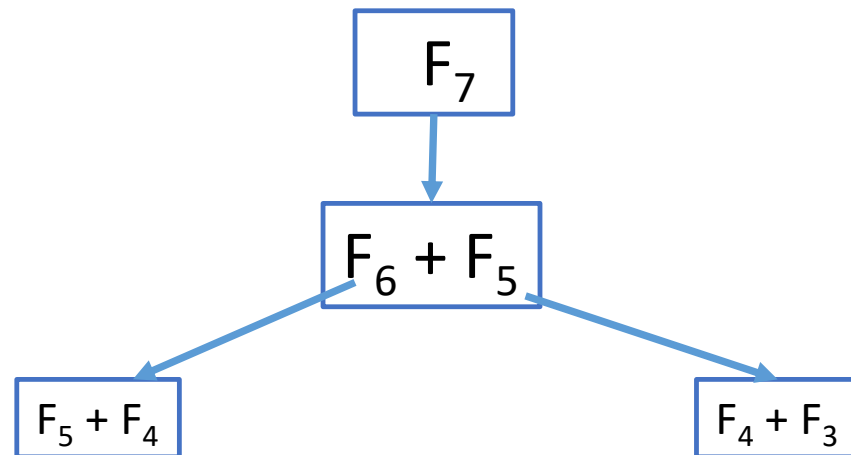
$$F_n = F_{n-1} + F_{n-2}$$

$$F_7$$

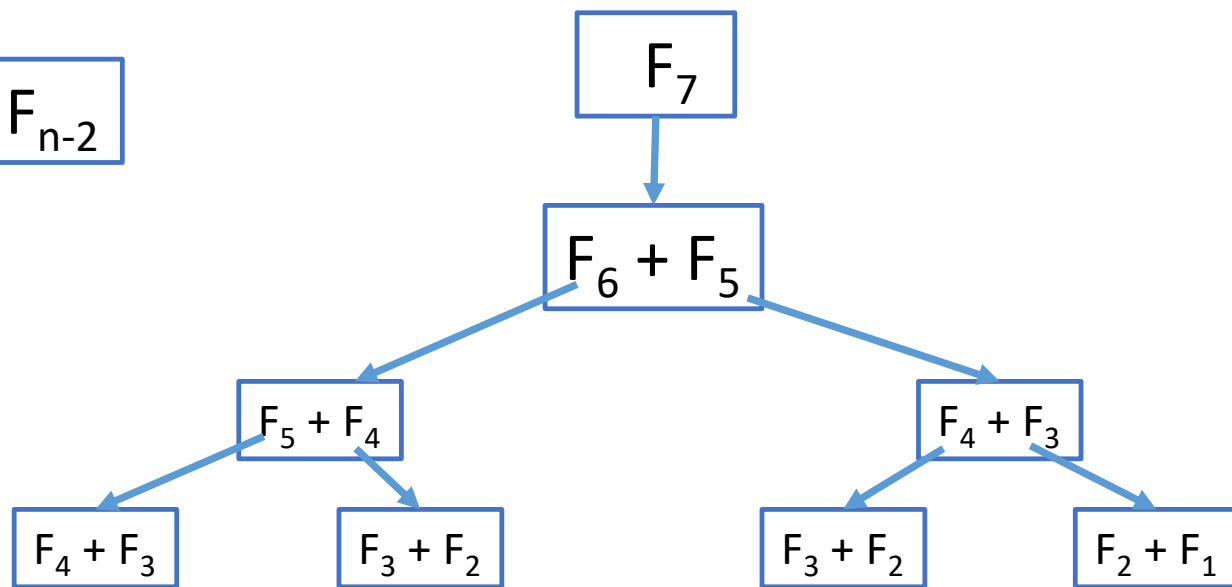


$$F_6 + F_5$$

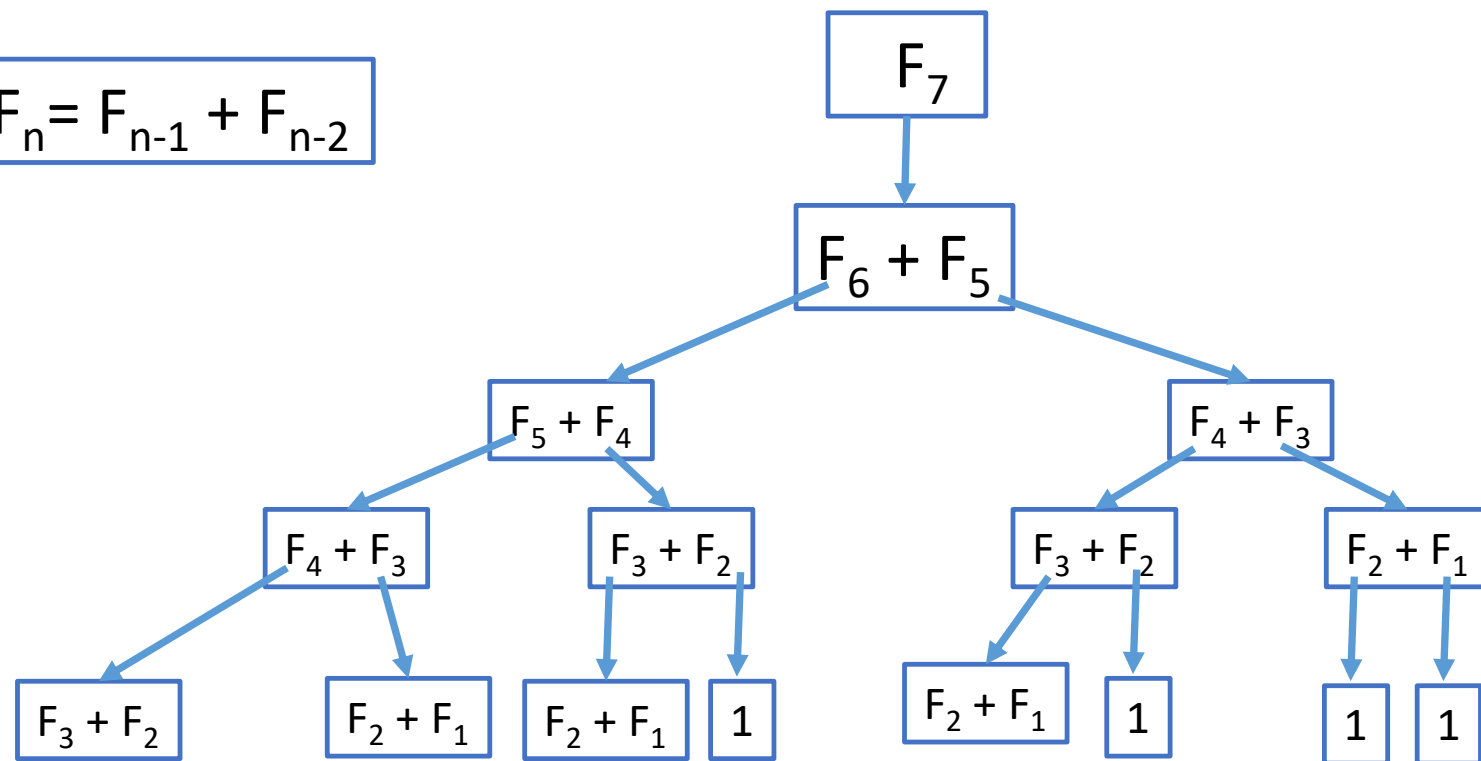
$$F_n = F_{n-1} + F_{n-2}$$



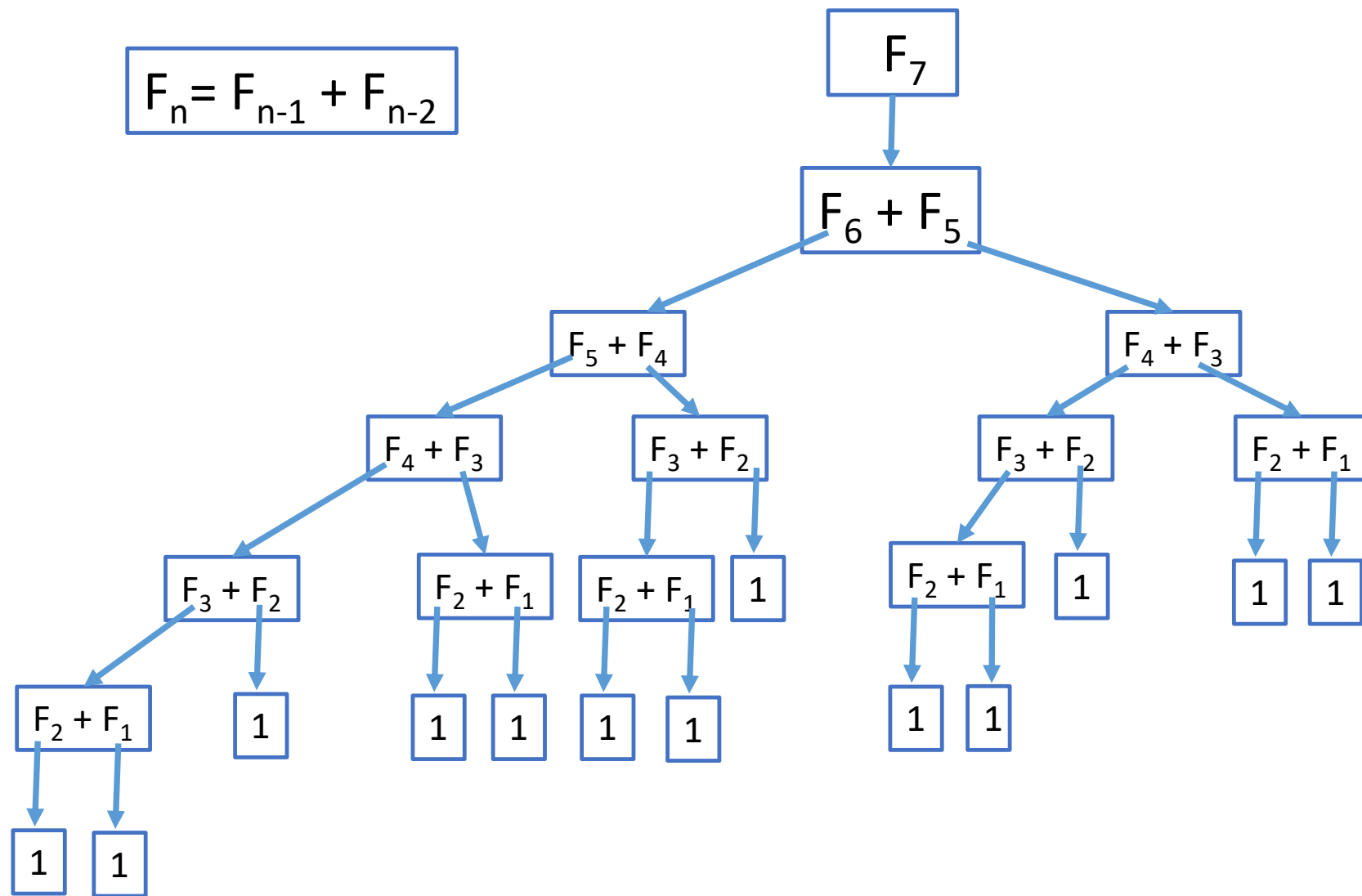
$$F_n = F_{n-1} + F_{n-2}$$



$$F_n = F_{n-1} + F_{n-2}$$



$$F_n = F_{n-1} + F_{n-2}$$



$$F_1=1, F_2=1$$

$$F_n = F_{n-1} + F_{n-2}, n=3,4,\dots$$

$$F_1=1, F_2=1$$

$$F_1=1, F_2=1$$

$$F_n = F_{n-1} + F_{n-2}, n=3,4,\dots$$

$$F_1=1, F_2=1$$

$$F_3 = F_2 + F_1$$

$$F_1=1, F_2=1$$

$$F_n = F_{n-1} + F_{n-2}, n=3,4,\dots$$

$$F_1=1, F_2=1$$

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$F_1=1, F_2=1$$

$$F_n = F_{n-1} + F_{n-2}, n=3,4,\dots$$

$$F_1=1, F_2=1$$

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$F_5 = F_4 + F_3$$

$$F_1=1, F_2=1$$

$$F_n = F_{n-1} + F_{n-2}, n=3,4,\dots$$

$$F_1=1, F_2=1$$

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$F_5 = F_4 + F_3$$

$$F_6 = F_5 + F_4$$

$$F_1=1, F_2=1$$

$$F_n = F_{n-1} + F_{n-2}, n=3,4,\dots$$

$$F_1=1, F_2=1$$

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$F_5 = F_4 + F_3$$

$$F_6 = F_5 + F_4$$

$$F_7 = F_6 + F_5$$

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

```
function fibonacci(n) {  
  if (n === 1 || n === 2)  
    return 1  
  else {  
    let f_n_1 = f_n_2 = 1  
    for (let i = 3; i <= n; i++) {  
      f_n = f_n_1 + f_n_2  
  
      f_n_2 = f_n_1  
      f_n_1 = f_n  
    }  
    return f_n  
  }  
}
```

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

```
function fibonacci(n) {
  if (n === 1 || n === 2)
    return 1
  else {
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++) {
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_2=1, \quad F_1=1$$

$$f_{n-1} = 1$$

$$f_{n-2} = 1$$

$$f_n = f_{n-1} + f_{n-2}$$

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$F_5 = F_4 + F_3$$

$$F_6 = F_5 + F_4$$

$$F_7 = F_6 + F_5$$

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

```
function fibonacci(n) {
  if (n === 1 || n === 2)
    return 1
  else {
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++) {
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_2=1, F_1=1$$

$$f_{n-1} = 1$$

$$f_{n-2} = 1$$

$$f_n = f_{n-1} + f_{n-2}$$

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$F_5 = F_4 + F_3$$

$$F_6 = F_5 + F_4$$

$$F_7 = F_6 + F_5$$

$$f_n = f_{n-1} + f_{n-2}$$

$$f_n = f_{n-1} + f_{n-2}$$

$$f_n = f_{n-1} + f_{n-2}$$

$$f_n = f_{n-1} + f_{n-2}$$

$$f_n = f_{n-1} + f_{n-2}$$

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

$$F_2=1, F_1=1$$

$$f_{n-1} = 1$$

$$f_{n-2} = 1$$

$$f_{n-2} = 1$$

$$f_{n-1} = 1$$

```
function fibonacci(n) {  
  if (n === 1 || n === 2) {  
    return 1  
  }  
  else {  
    let f_n_1 = f_n_2 = 1  
    for (let i = 3; i <= n; i++) {  
      f_n = f_n_1 + f_n_2  
      f_n_2 = f_n_1  
      f_n_1 = f_n  
    }  
    return f_n  
  }  
}
```

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

$$F_2=1, F_1=1$$

$$f_{n-1} = 1$$

$$f_{n-2} = 1$$

$$f_n$$

$$f_{n-1}$$

$$f_{n-2}$$

$$F_3 = F_2 + F_1$$

$$f_{n-2} = 1$$

$$f_{n-1} = 1$$

$$f_n = f_{n-1} + f_{n-2}$$

```
function fibonacci(n) {
  if (n === 1 || n === 2)
    return 1
  else {
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++) {
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_1=1, F_2=1$$

$$\boxed{f_n} = \boxed{f_{n-1}} + \boxed{f_{n-2}}, n=3,4,\dots$$

$$F_2=1, \quad F_1=1$$

$$\boxed{f_{n-1}} = 1 \quad \boxed{f_{n-2}} = 1$$

$$\boxed{f_n} \quad \boxed{f_{n-1}} \quad \boxed{f_{n-2}}$$

$$F_3 = F_2 + F_1 \quad f_n = f_{n-1} + f_{n-2}$$

$$F_4 = F_3 + F_2$$

```
function fibonacci(n) {
  if (n === 1 || n === 2)
    return 1
  else {
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++) {
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

$$F_2=1, F_1=1$$

$$f_{n-1} = 1$$

$$f_{n-2} = 1$$

$$f_n = f_{n-1} + f_{n-2}$$

$$F_3 = F_2 + F_1$$

$$f_n = f_{n-1} + f_{n-2}$$

$$f_{n-2} = f_{n-1}$$

$$F_4 = F_3 + F_2$$

```
function fibonacci(n){
  if(n===1 || n===2)
    return 1
  else{
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++){
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

```
function fibonacci(n){
  if(n===1 || n===2)
    return 1
  else{
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++){
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_2=1, F_1=1$$

$$f_{n-1} = 1$$

$$f_{n-2} = 1$$

$$f_n = f_{n-1} + f_{n-2}$$

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$f_n = f_{n-1} + f_{n-2}$$

$$f_{n-2} = f_{n-1}$$

$$f_{n-1} = f_n$$

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

```
function fibonacci(n){
  if(n===1 || n===2)
    return 1
  else{
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++){
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_2=1, F_1=1$$

$$f_{n-1} = 1$$

$$f_{n-2} = 1$$

$$f_n$$

$$f_{n-1}$$

$$f_{n-2}$$

$$F_3 = F_2 + F_1$$

$$f_{n-2} = f_{n-1}$$

$$f_{n-1} = f_n$$

$$F_4 = F_3 + F_2$$

$$f_n = f_{n-1} + f_{n-2}$$

$$F_1=1, F_2=1$$

$$\boxed{f_n} = \boxed{f_{n-1}} + \boxed{f_{n-2}}, n=3,4,\dots$$

$$F_2=1, \quad F_1=1$$

$$\boxed{f_{n-1}} = 1 \quad \boxed{f_{n-2}} = 1$$

$$\boxed{f_n} \quad \boxed{f_{n-1}} \quad \boxed{f_{n-2}}$$

$$F_3 = F_2 + F_1 \quad \begin{array}{l} f_{n-2} = f_{n-1} \\ f_{n-1} = f_n \end{array}$$

$$F_4 = F_3 + F_2 \quad f_n = f_{n-1} + f_{n-2}$$

$$F_5 = F_4 + F_3$$

```
function fibonacci(n) {
  if (n === 1 || n === 2)
    return 1
  else {
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++) {
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

```
function fibonacci(n){
  if(n===1 || n===2)
    return 1
  else{
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++){
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_2=1, F_1=1$$

$$f_{n-1} = 1, f_{n-2} = 1$$

$$f_n = f_{n-1} + f_{n-2}$$

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$F_5 = F_4 + F_3$$

$$f_{n-2} = f_{n-1}$$

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

```
function fibonacci(n){
  if(n===1 || n===2)
    return 1
  else{
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++){
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_2=1, F_1=1$$

$$f_{n-1} = 1$$

$$f_{n-2} = 1$$

$$f_n = f_{n-1} + f_{n-2}$$

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$F_5 = F_4 + F_3$$

$$f_{n-2} = f_{n-1}$$

$$f_{n-1} = f_n$$

$$F_1=1, F_2=1$$

$$\boxed{f_n} = \boxed{f_{n-1}} + \boxed{f_{n-2}}, n=3,4,\dots$$

```
function fibonacci(n) {
  if(n===1 || n===2)
    return 1
  else{
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++) {
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

$$F_2=1, \quad F_1=1$$

$$\boxed{f_{n-1}} = 1 \quad \boxed{f_{n-2}} = 1$$

$$\boxed{f_n} \quad \boxed{f_{n-1}} \quad \boxed{f_{n-2}}$$

$$F_3 = F_2 + F_1$$

$$\boxed{F_4} = \boxed{F_3} + F_2$$

$$\boxed{F_5} = \boxed{F_4} + \boxed{F_3}$$

$$f_{n-2} = f_{n-1}$$

$$f_{n-1} = f_n$$

$$\boxed{f_n = f_{n-1} + f_{n-2}}$$

$$F_1=1, F_2=1$$

$$f_n = f_{n-1} + f_{n-2}, n=3,4,\dots$$

$$F_2=1, F_1=1$$

$$f_{n-1} = 1$$

$$f_{n-2} = 1$$

$$f_n = f_{n-1} + f_{n-2}$$

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$F_5 = F_4 + F_3$$

$$F_6 = F_5 + F_4$$

$$F_7 = F_6 + F_5$$


```
function fibonacci(n){
  if(n===1 || n===2)
    return 1
  else{
    let f_n_1 = f_n_2 = 1
    for (let i = 3; i <= n; i++){
      f_n = f_n_1 + f_n_2

      f_n_2 = f_n_1
      f_n_1 = f_n
    }
    return f_n
  }
}
```

Випадок вкладених функцій

Вкладені функції – функції, що описані всередині іншої функції

Випадок вкладених функцій (функція описана всередині іншої функції)

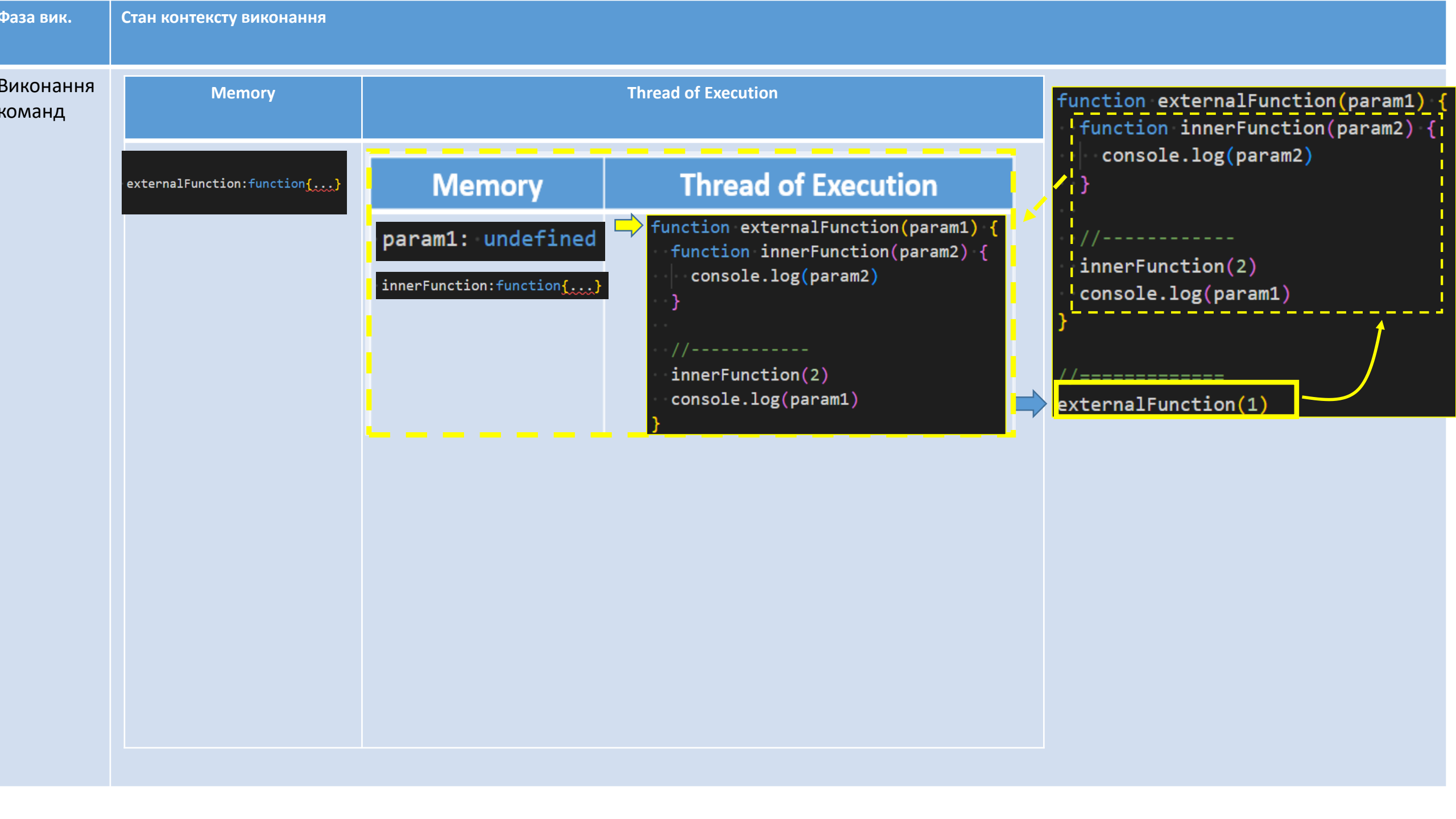


```
function externalFunction(param1) {  
    function innerFunction(param2) {  
        console.log(param2)  
    }  
  
    // -----  
    innerFunction(2)  
    console.log(param1)  
}  
  
// =====  
externalFunction(1)
```

Приклад. Приклад виконання вкладених функцій

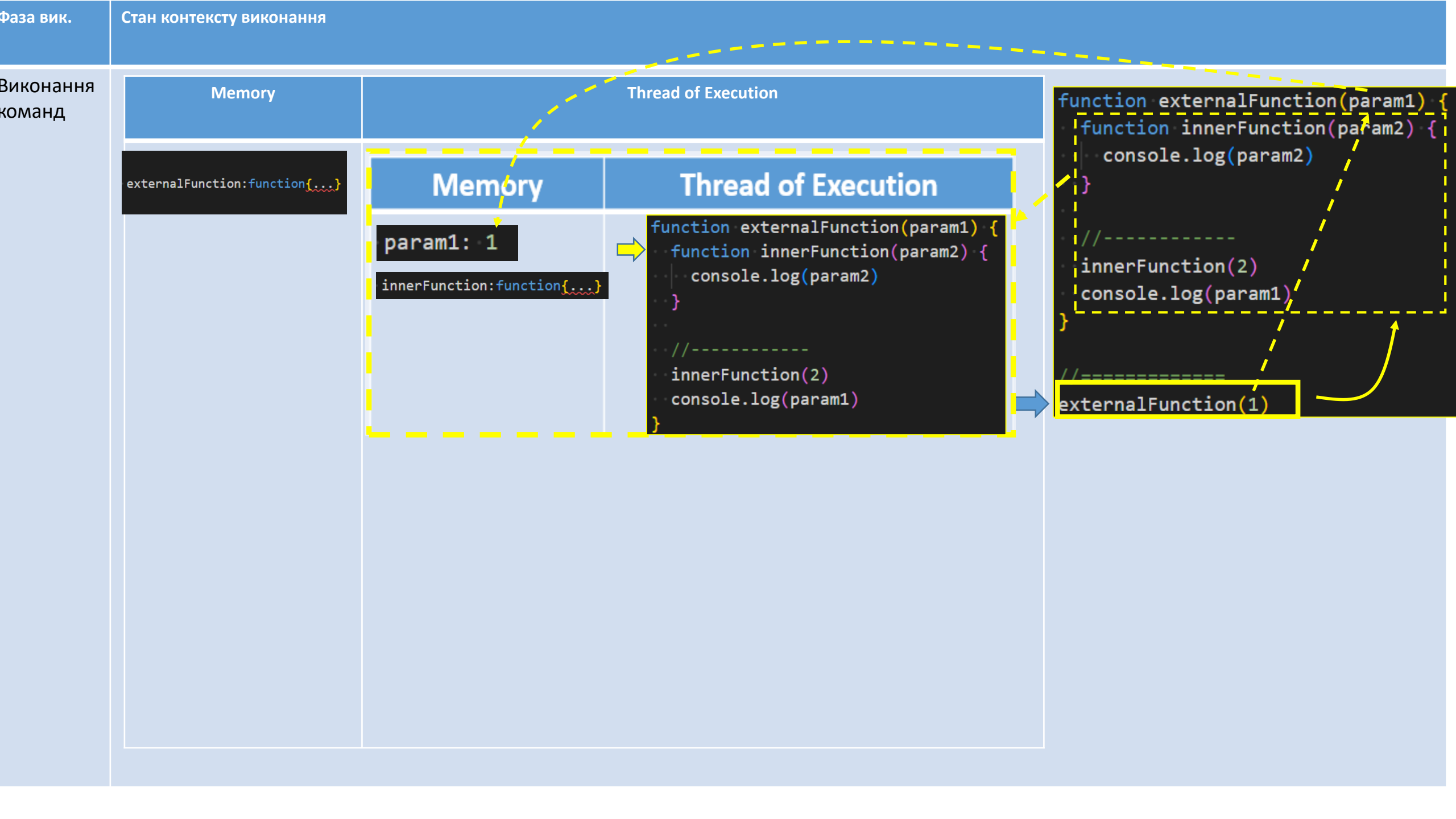
| Фаза вик. | Стан контексту виконання | |
|---|---|---|
| 1. Виділення пам'яті для: 1)змінних(змінні мають початкове значення undefined) 2)функцій (function-declaration) | Memory | Thread of Execution |
| | <pre>externalFunction:function{...}</pre> | |
| 2) Виконання команд | Memory | Thread of Execution |
| | <pre>externalFunction:function{...}</pre> | <pre>function externalFunction(param1) { · function innerFunction(param2) { · · console.log(param2) · } · · //----- · innerFunction(2) · console.log(param1) } //----- externalFunction(1)</pre> |

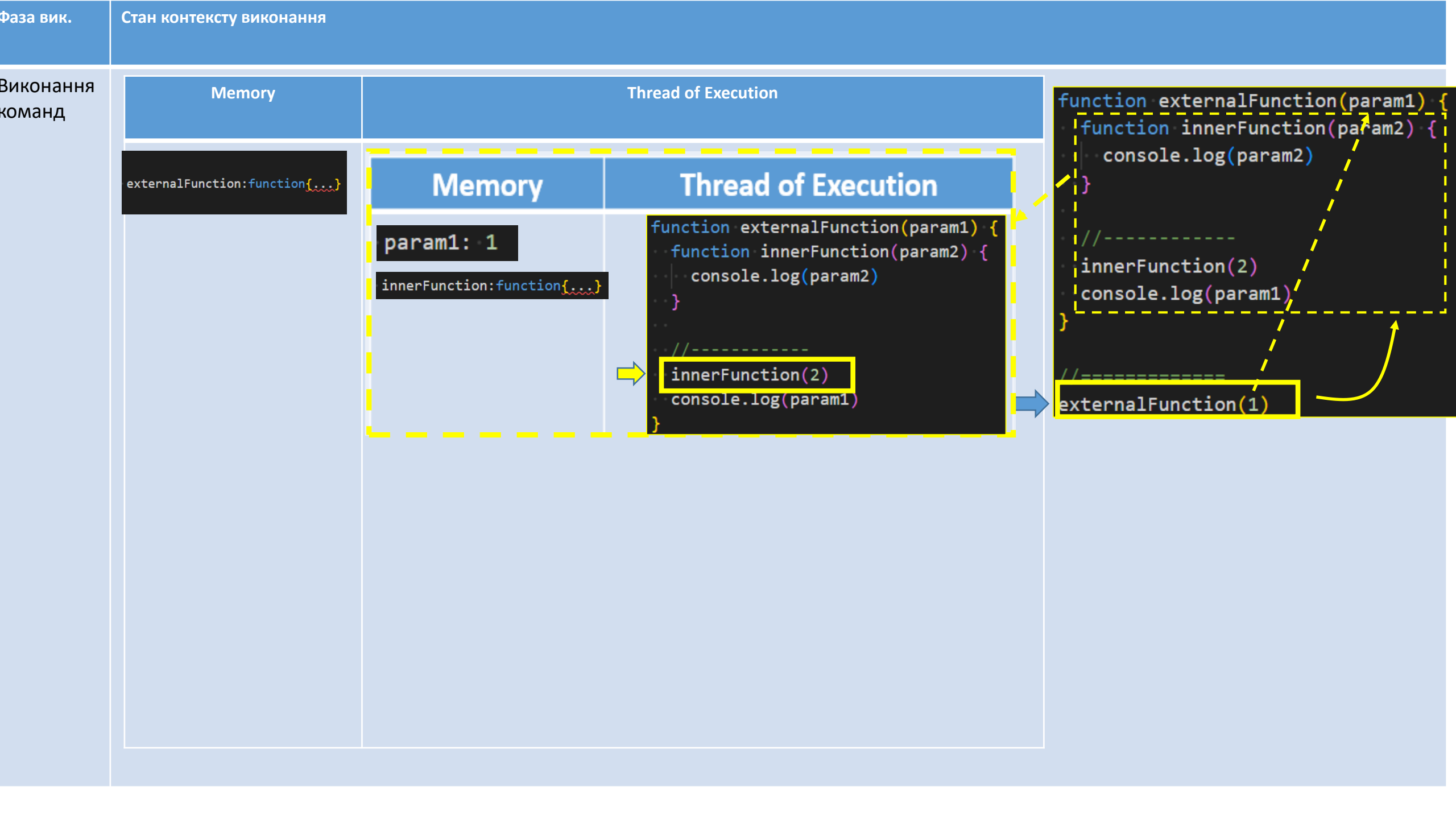
| Фаза вик. | Стан контексту виконання | | |
|------------------|---|---|--|
| Виконання команд | Memory | Thread of Execution | |
| | <pre>externalFunction: function {...}</pre> | <div><pre>function externalFunction(param1) { function innerFunction(param2) { console.log(param2) } // ----- innerFunction(2) console.log(param1) } // ----- externalFunction(1)</pre></div> | |

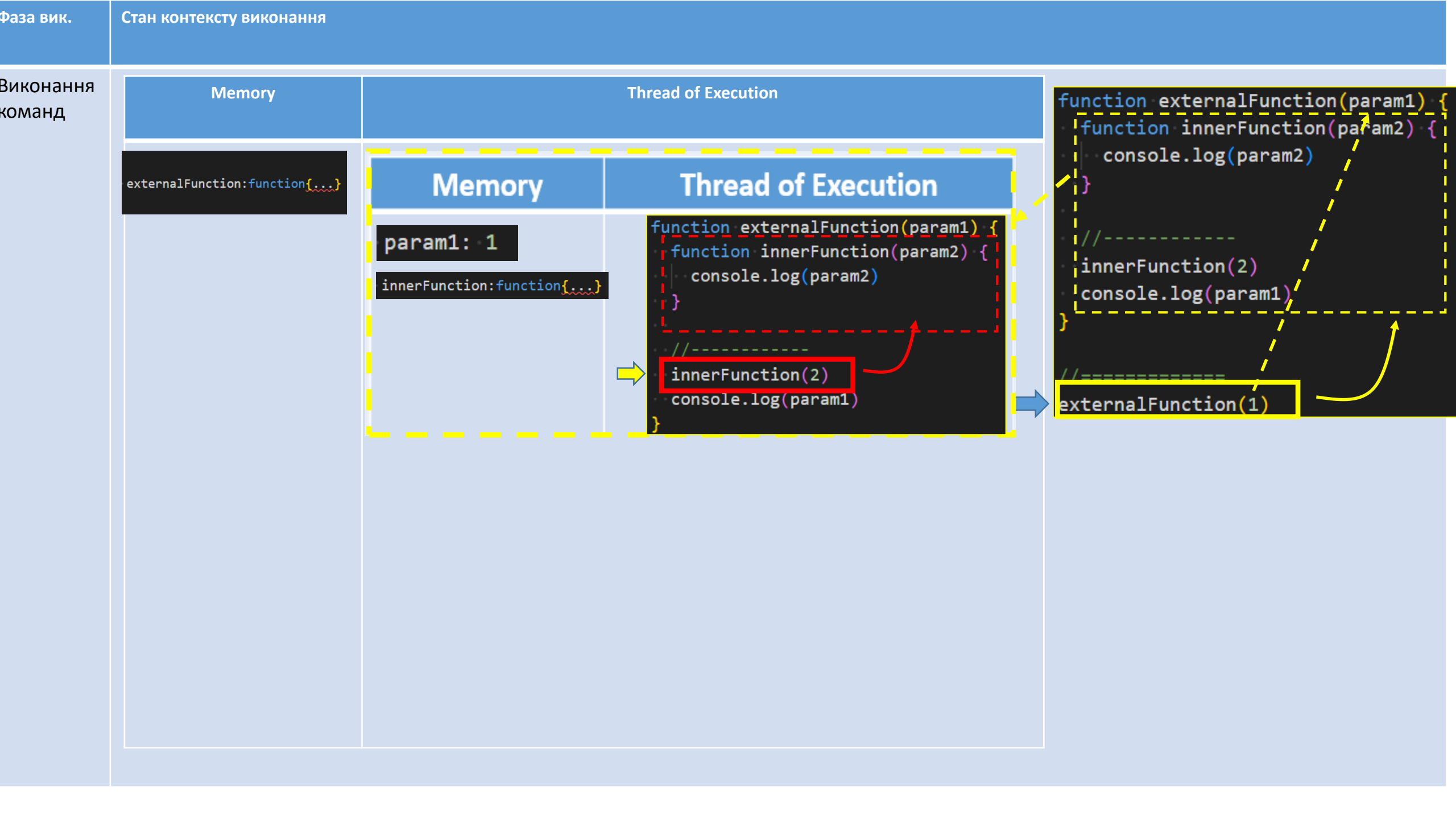


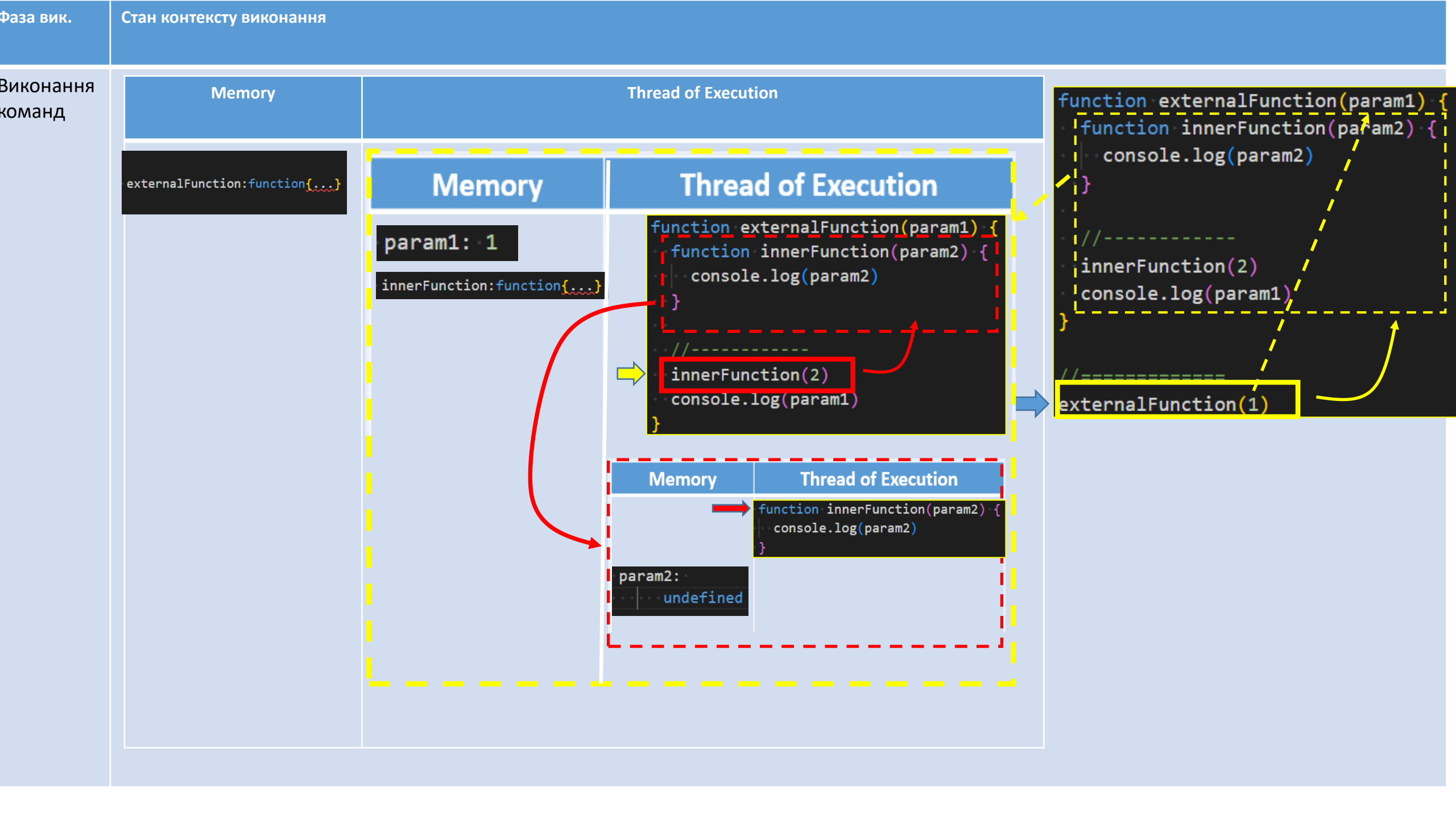
function externalFunction(param1) {
 function innerFunction(param2) {
 console.log(param2)
 }
 //-----
 innerFunction(2)
 console.log(param1)
}

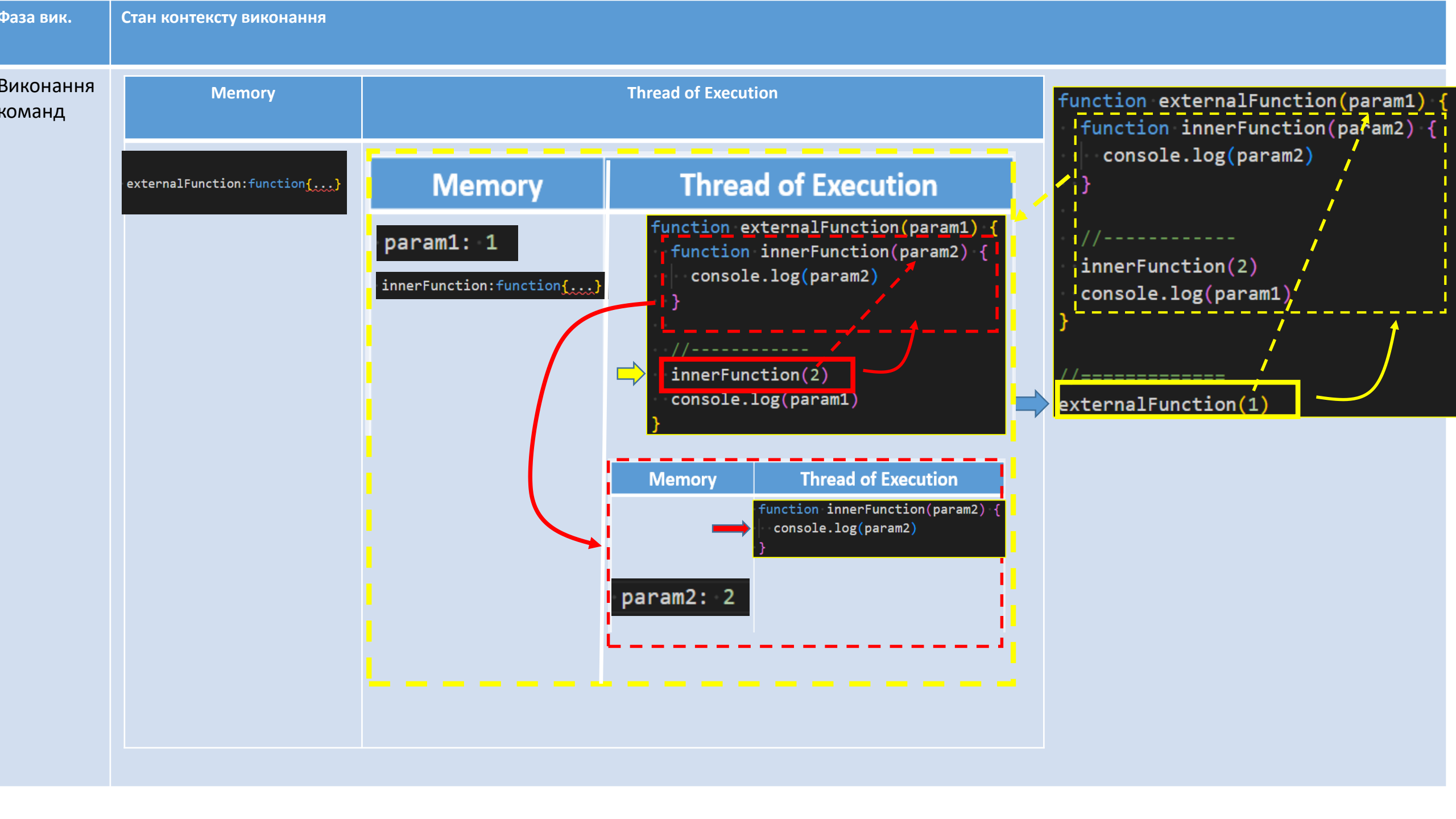
externalFunction(1)

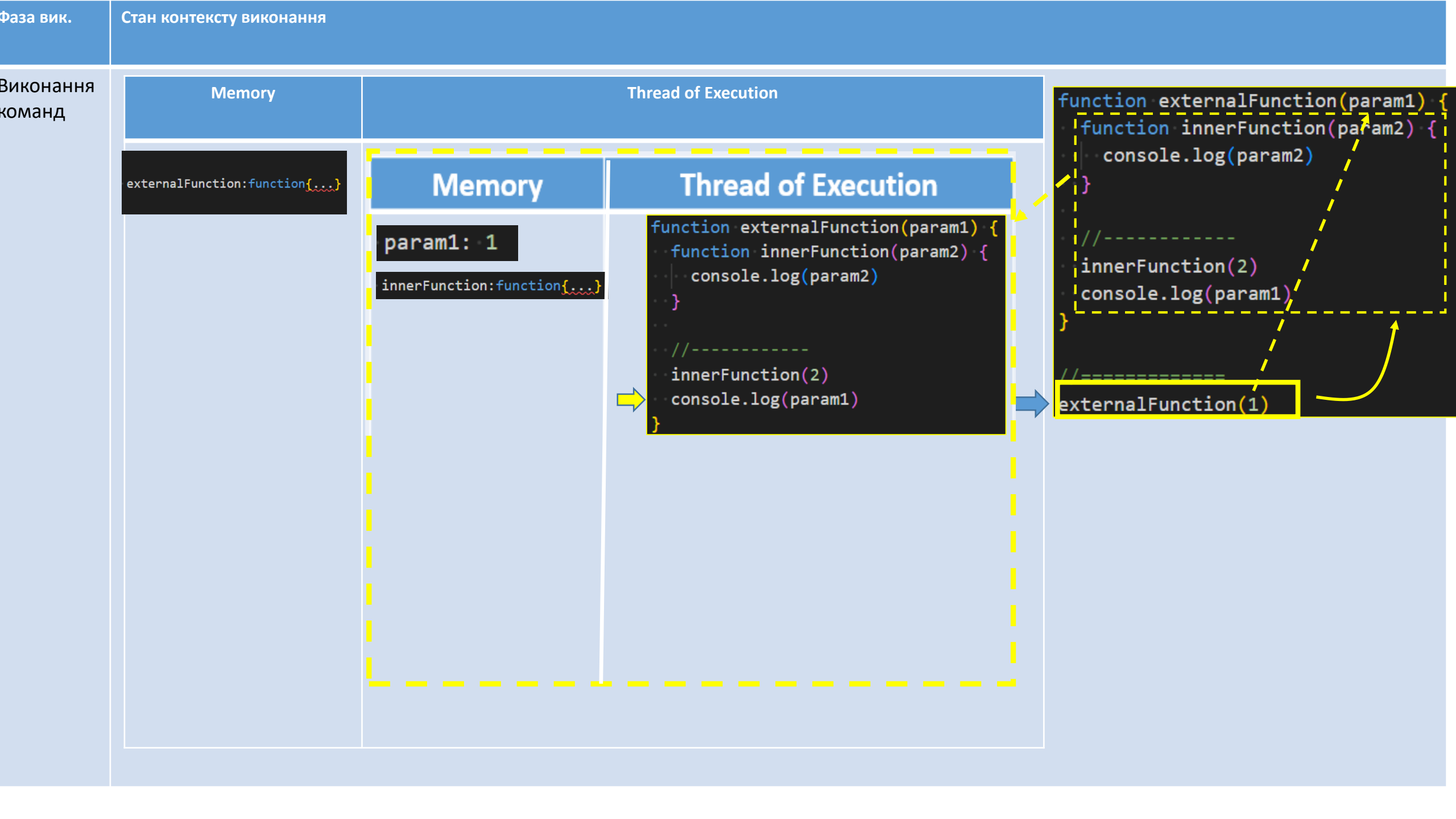












| Фаза вик. | Стан контексту виконання | | |
|------------------|--|--|--|
| Виконання команд | Memory | Thread of Execution | |
| | <pre>externalFunction: function(...) {</pre> | <pre>function externalFunction(param1) { function innerFunction(param2) { console.log(param2) } //----- innerFunction(2) console.log(param1) } //===== externalFunction(1)</pre> | <pre>function externalFunction(param1) { function innerFunction(param2) { console.log(param2) } //----- innerFunction(2) console.log(param1) } //===== externalFunction(1)</pre> |

1. Без замикання (аналогія з кавою)

Без замикання (кава): Баба Галя отримала чашку кави й пішла. Результат — незалежний, одноразовий, зв'язок із кав'ярнею розірвано.

```
function makeCoffee() {  
  let coffee = "Еспресо";  
  return coffee;  
}  
  
let myCoffee = makeCoffee();  
console.log(myCoffee); // "Еспресо"
```

2. Із замиканням (аналогія з кавовим абонементом)

Із замиканням (абонемент): Баба Галя отримала картку, яка пов'язана з кав'ярнею. Кожен раз, коли вона її використовує, картка "знає" попередній стан і працює з ним.

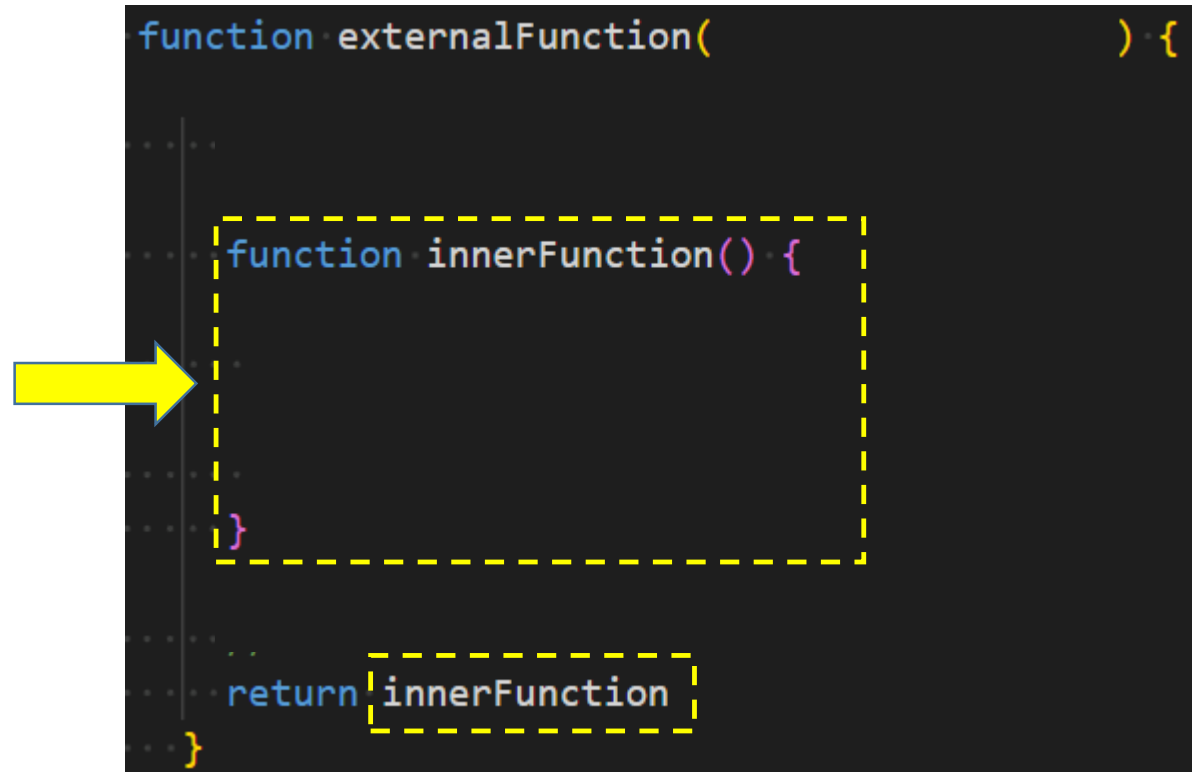
У програмуванні це **замикання** — функція повертає щось, що зберігає зв'язок із внутрішнім контекстом:

```
function createCoffeeSubscription() {  
  let coffeeCount = 0; // внутрішній стан  
  
  return function() { // функція, що повертається  
    coffeeCount += 1;  
    return `Твоя ${coffeeCount}-а кава готова!`;  
  };  
}  
  
let mySubscription = createCoffeeSubscription();  
console.log(mySubscription()); // "Твоя 1-а кава готова!"  
console.log(mySubscription()); // "Твоя 2-а кава готова!"
```


Замикання

У програмуванні **замиканням** (англ. *closure*) називають підпрограму, що виконується в середовищі, яке містить одну або більше зв'язаних змінних. Підпрограма має доступ до цих змінних під час виконання.

У програмуванні **замикання** — функція повертає щось, що зберігає зв'язок із внутрішнім контекстом

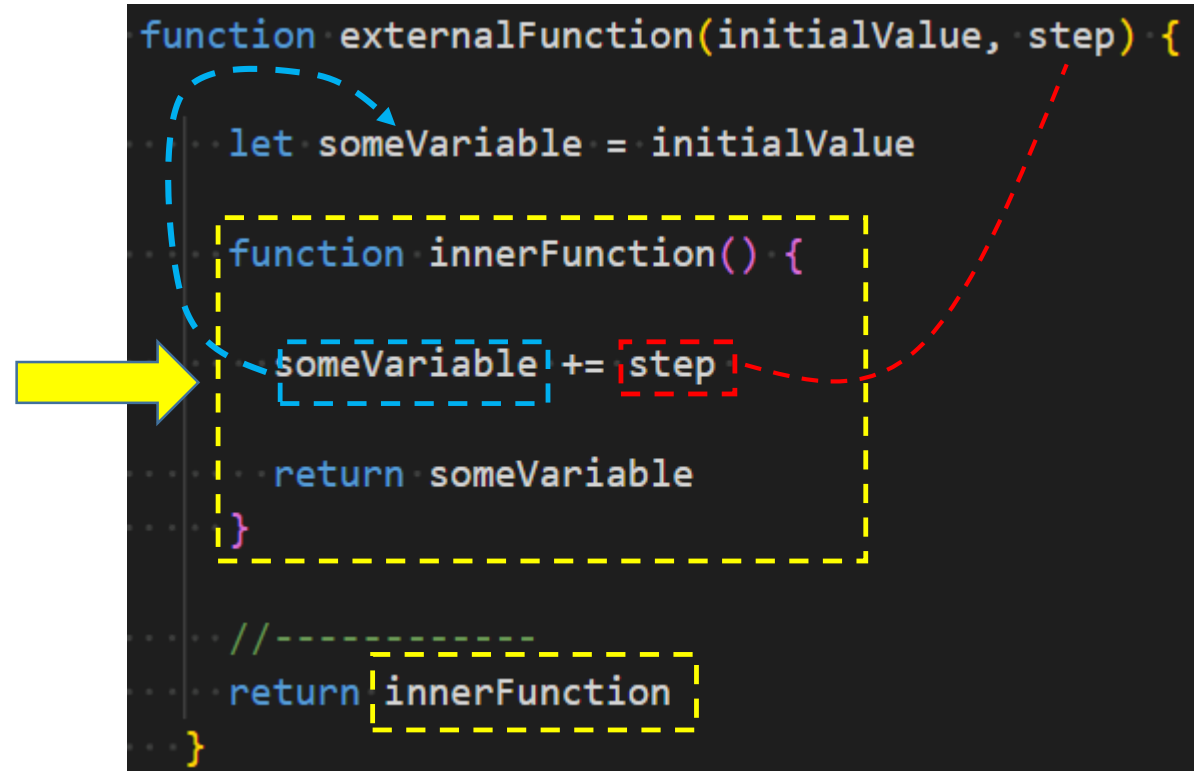


```
function externalFunction() {  
    function innerFunction() {  
    }  
    return innerFunction  
}
```

Замикання

У програмуванні замиканням (англ. *closure*) називають підпрограму, що виконується в середовищі, яке містить одну або більше зв'язаних змінних. Підпрограма має доступ до цих змінних під час виконання.

У програмуванні замикання — функція повертає щось, що зберігає зв'язок із внутрішнім контекстом

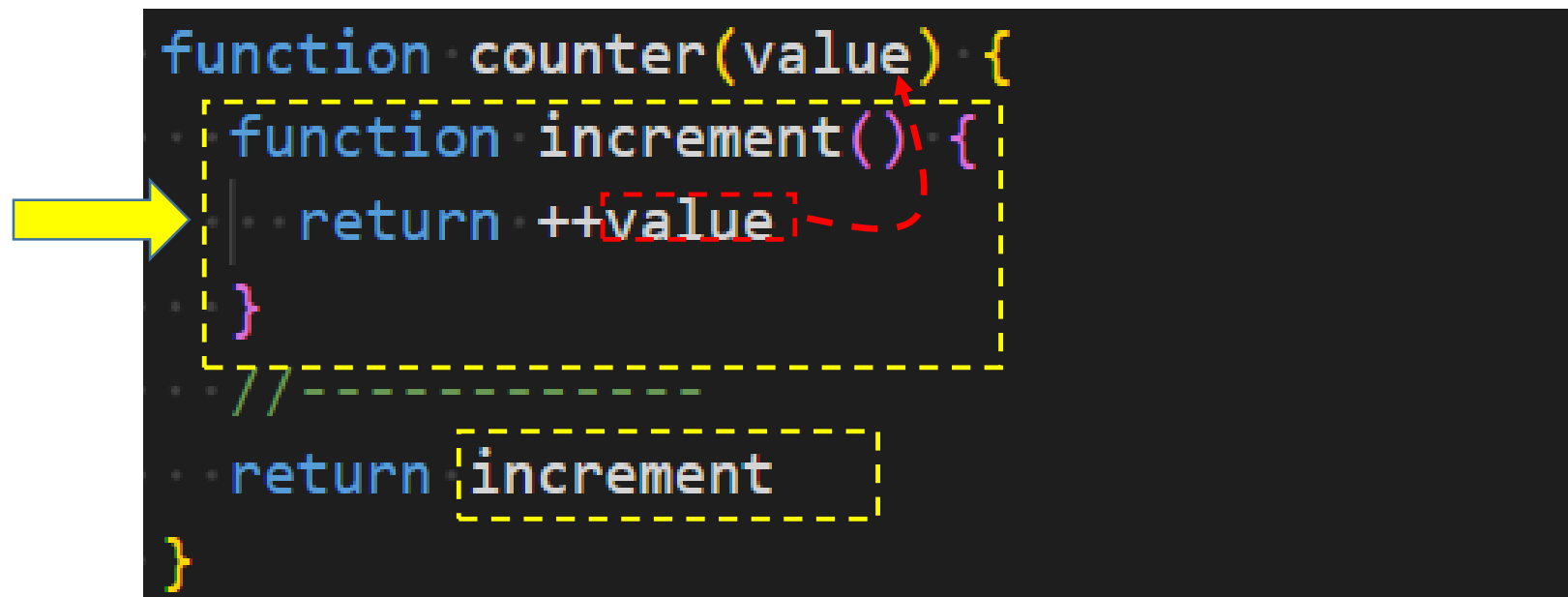


```
function externalFunction(initialValue, step) {  
  let someVariable = initialValue  
  
  function innerFunction() {  
    someVariable += step  
    return someVariable  
  }  
  
  // -----  
  return innerFunction  
}
```

Приклад використання замикання

```
function counter(value) {  
  function increment() {  
    return ++value  
  }  
  //-----  
  return increment  
}  
//=====  
let cntr1 = counter(0)  
  
document.write(`Cntr1 = ${cntr1()} <br>`)  
document.write(`Cntr1 = ${cntr1()} <br>`)
```

Приклад використання замикання



```
function counter(value) {  
  function increment() {  
    return ++value  
  }  
  // -----  
  return increment  
}
```

```
let cntr1 = counter(0)  
  
document.write(`Cntr1 = ${cntr1()} <br>`)  
document.write(`Cntr1 = ${cntr1()} <br>`)
```

Фази виконання програм JavaScript

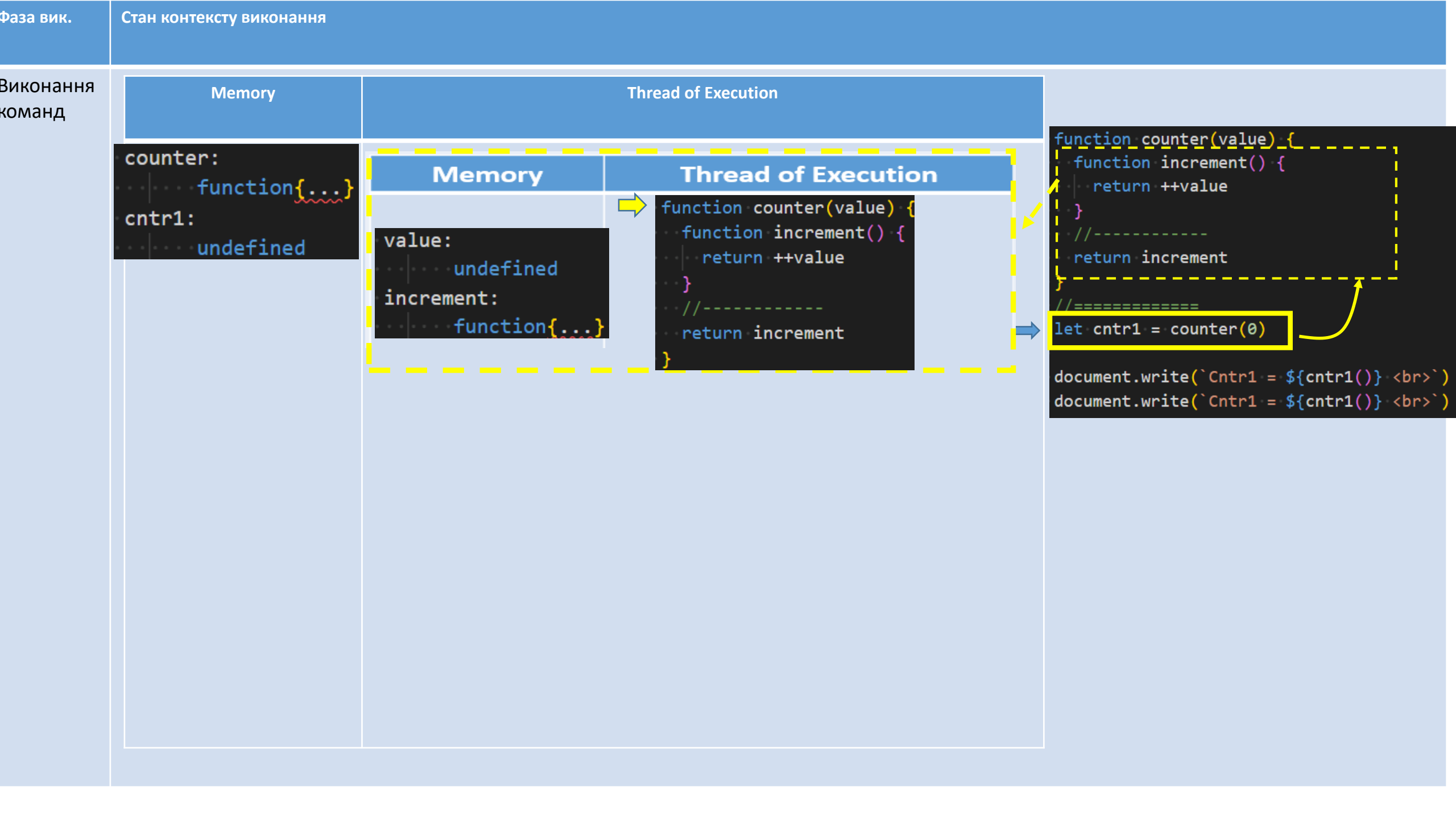
1) **Фаза виділення пам'яті** (виділення пам'яті для змінних і функцій)

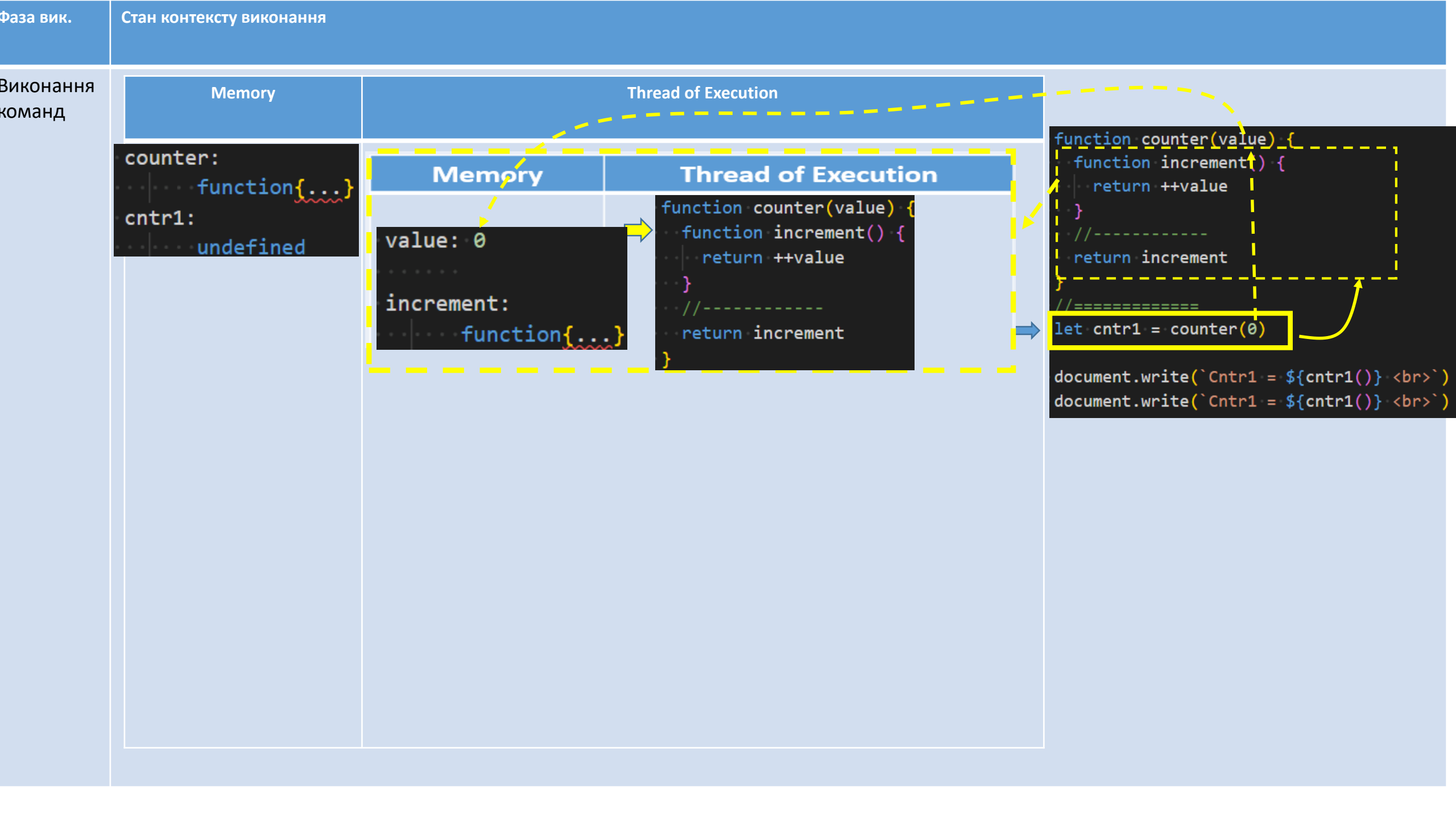
2) **Фаза виконання команд** (поступове виконання команд по одній за раз)

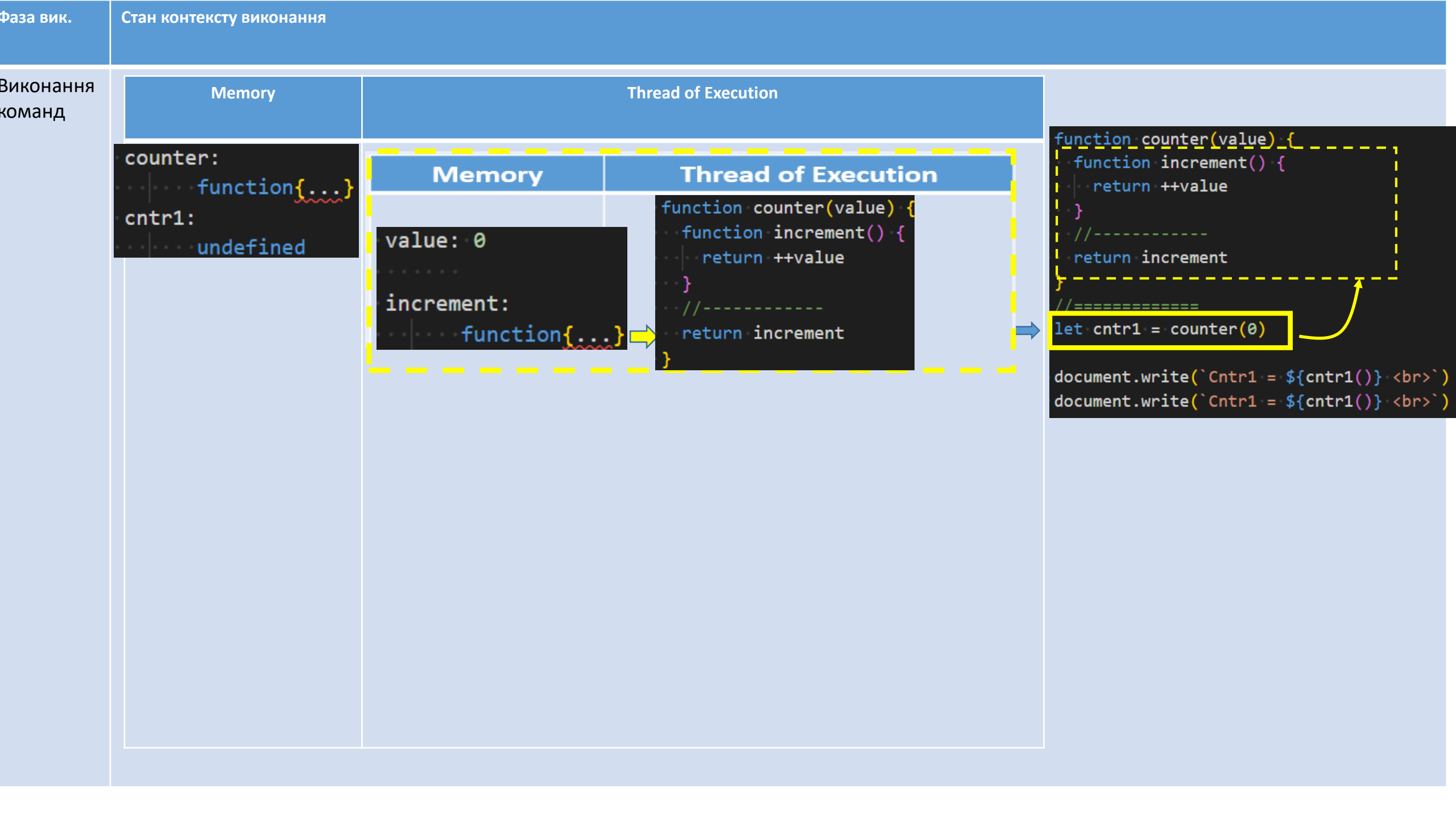
Приклад.

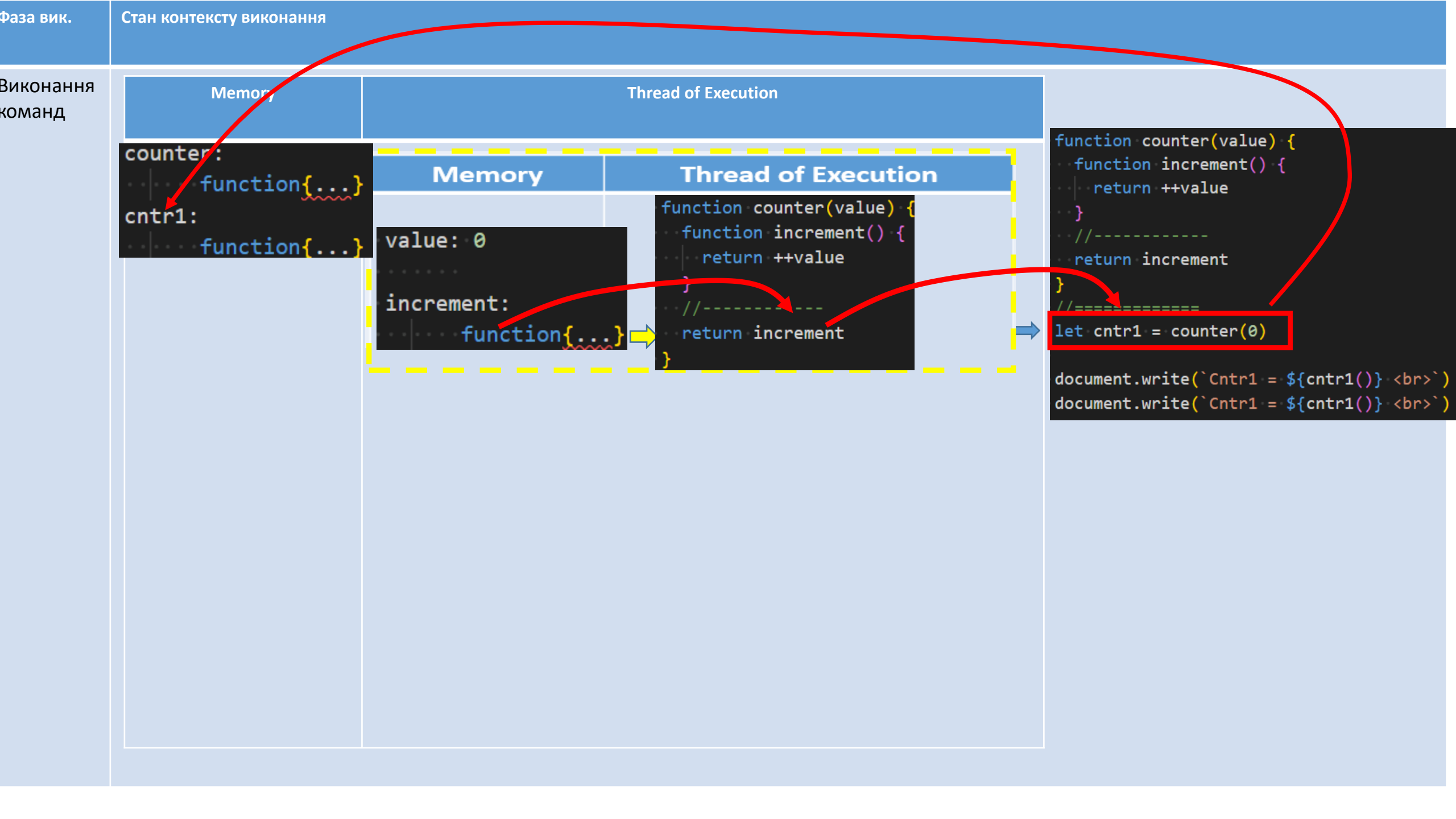
| Фаза вик. | Стан контексту виконання | |
|--|--|---|
| 1. Виділення пам'яті для: 1) змінних(змінні і мають початкове значення undefined) 2)функцій (function-declaration) | Memory | Thread of Execution |
| | <pre>counter: ... function{...} cntr1: ... undefined</pre> | |
| 2) Виконання команд | Memory | Thread of Execution |
| | <pre>counter: ... function{...} cntr1: ... undefined</pre> | <pre>function counter(value) { ... function increment() { ... return ++value ... } ... //----- ... return increment } //----- let cntr1 = counter(0) document.write(`Cntr1 = \${cntr1()} `) document.write(`Cntr1 = \${cntr1()} `)</pre> |

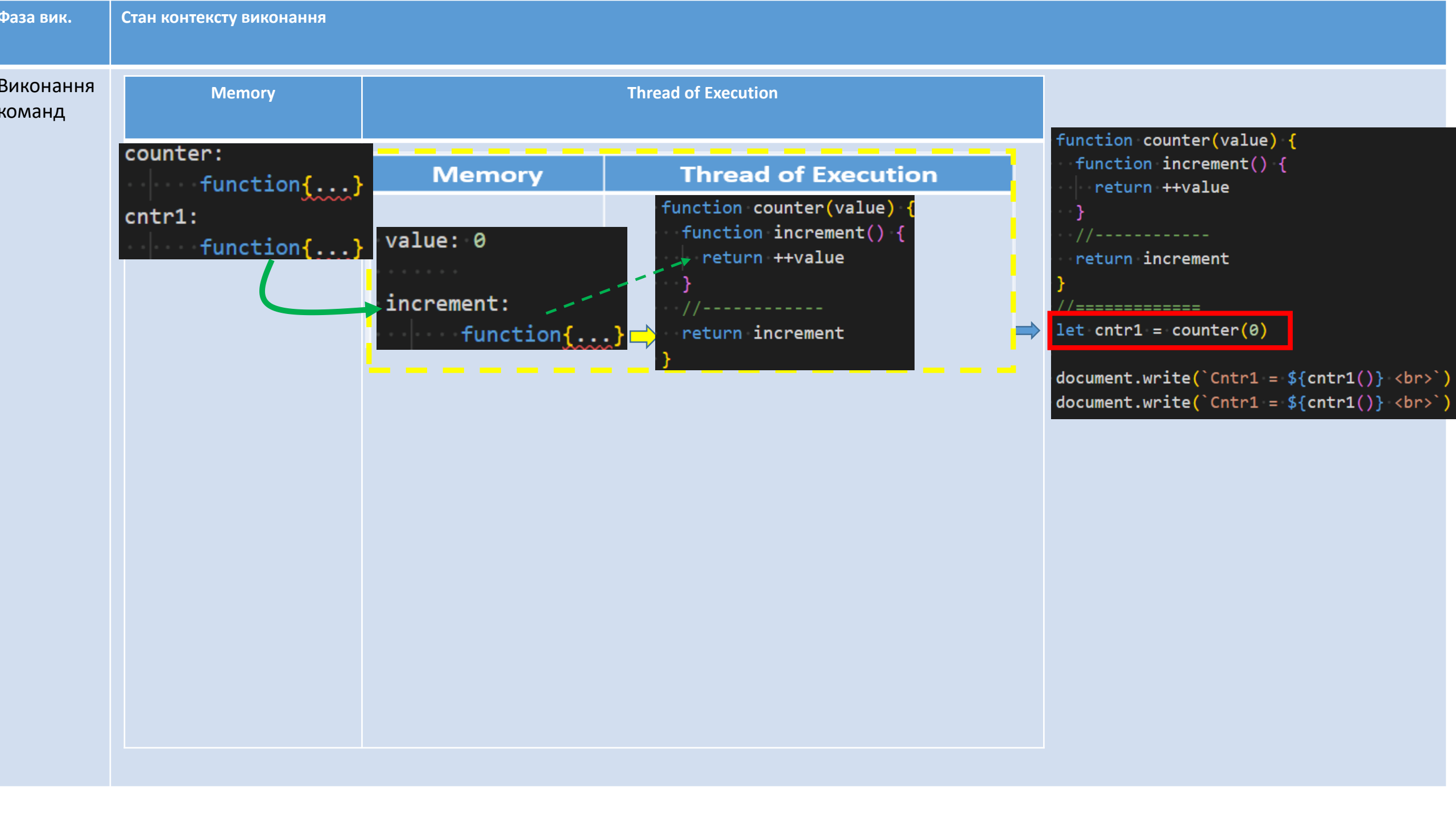
| Фаза вик. | Стан контексту виконання | | |
|------------------|--|--|--|
| Виконання команд | Memory | Thread of Execution | |
| | <pre>counter: ... function{...} cntr1: ... undefined</pre> | <div><div></div><div><div></div><div></div></div><div></div></div> <div><div></div><div></div></div> <div></div> | <pre>function counter(value){ function increment(){ return ++value } //----- return increment } //===== let cntr1 = counter(0) document.write(`Cntr1 = \${cntr1()} `) document.write(`Cntr1 = \${cntr1()} `)</pre> |

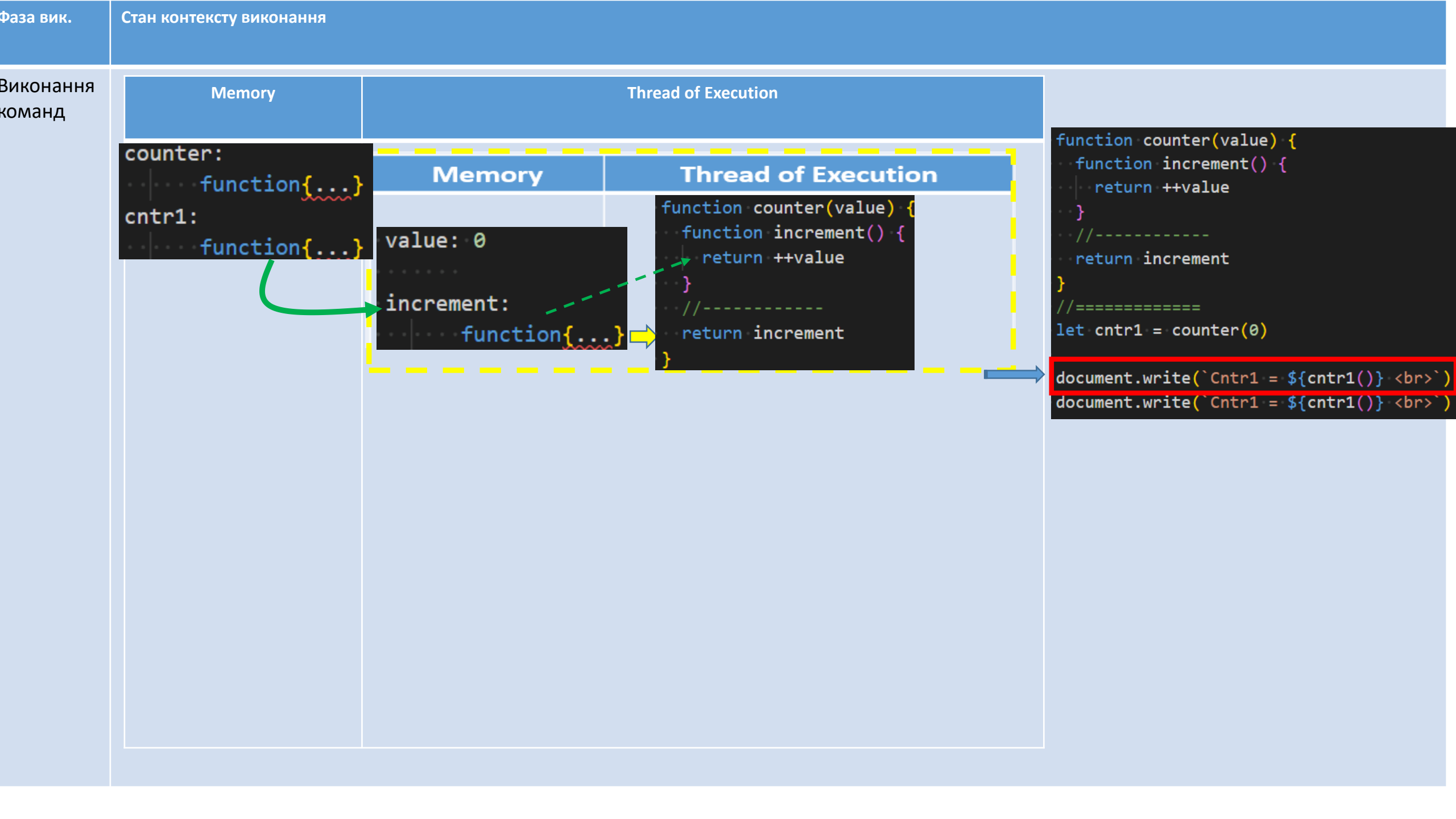


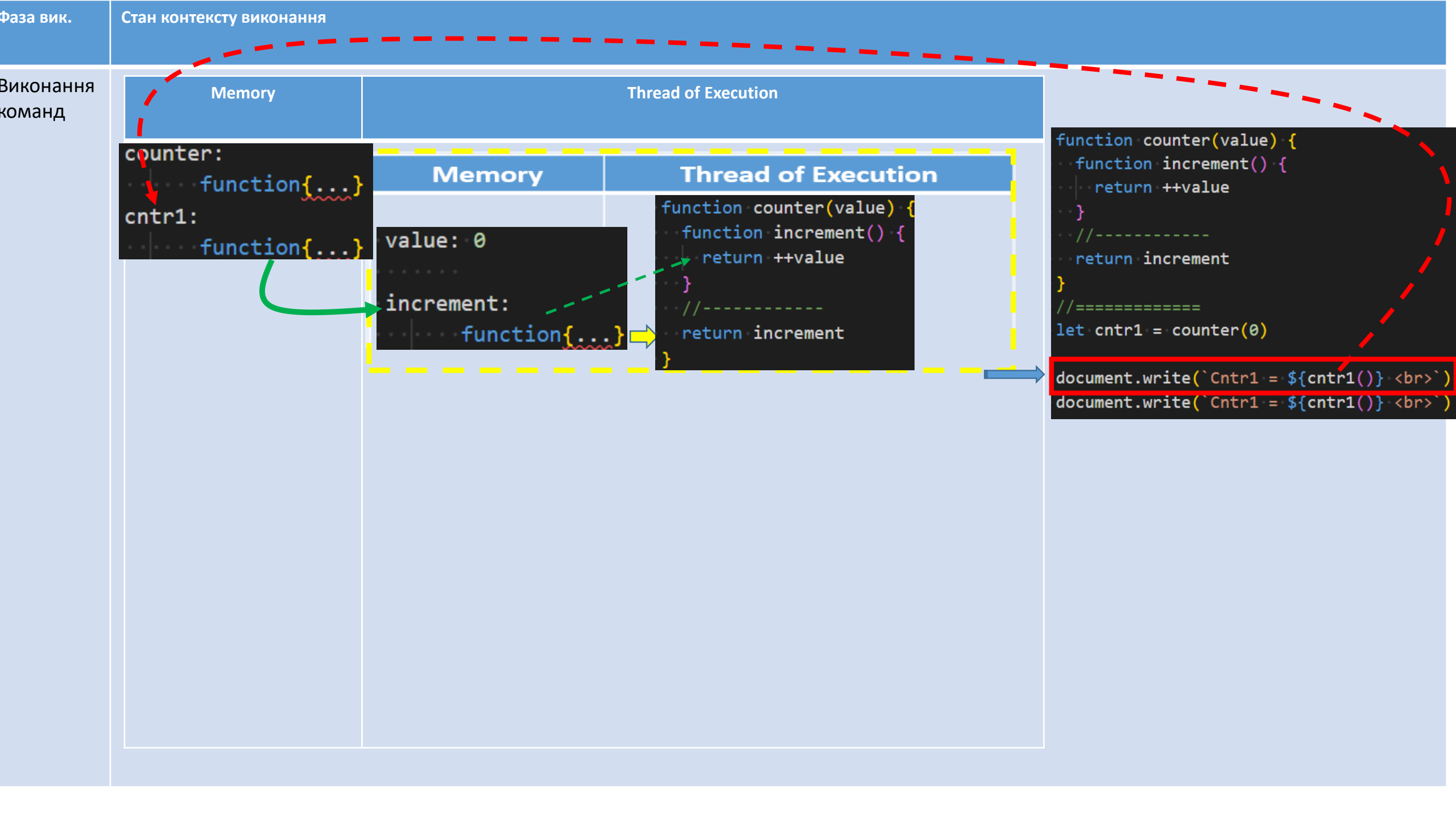


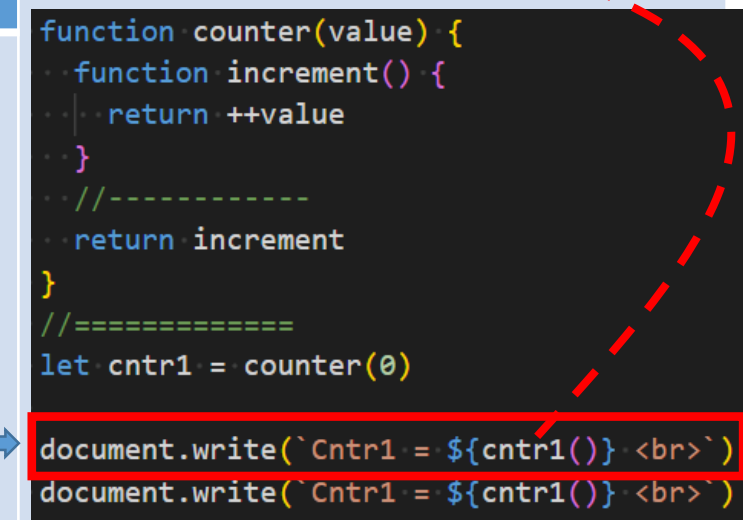


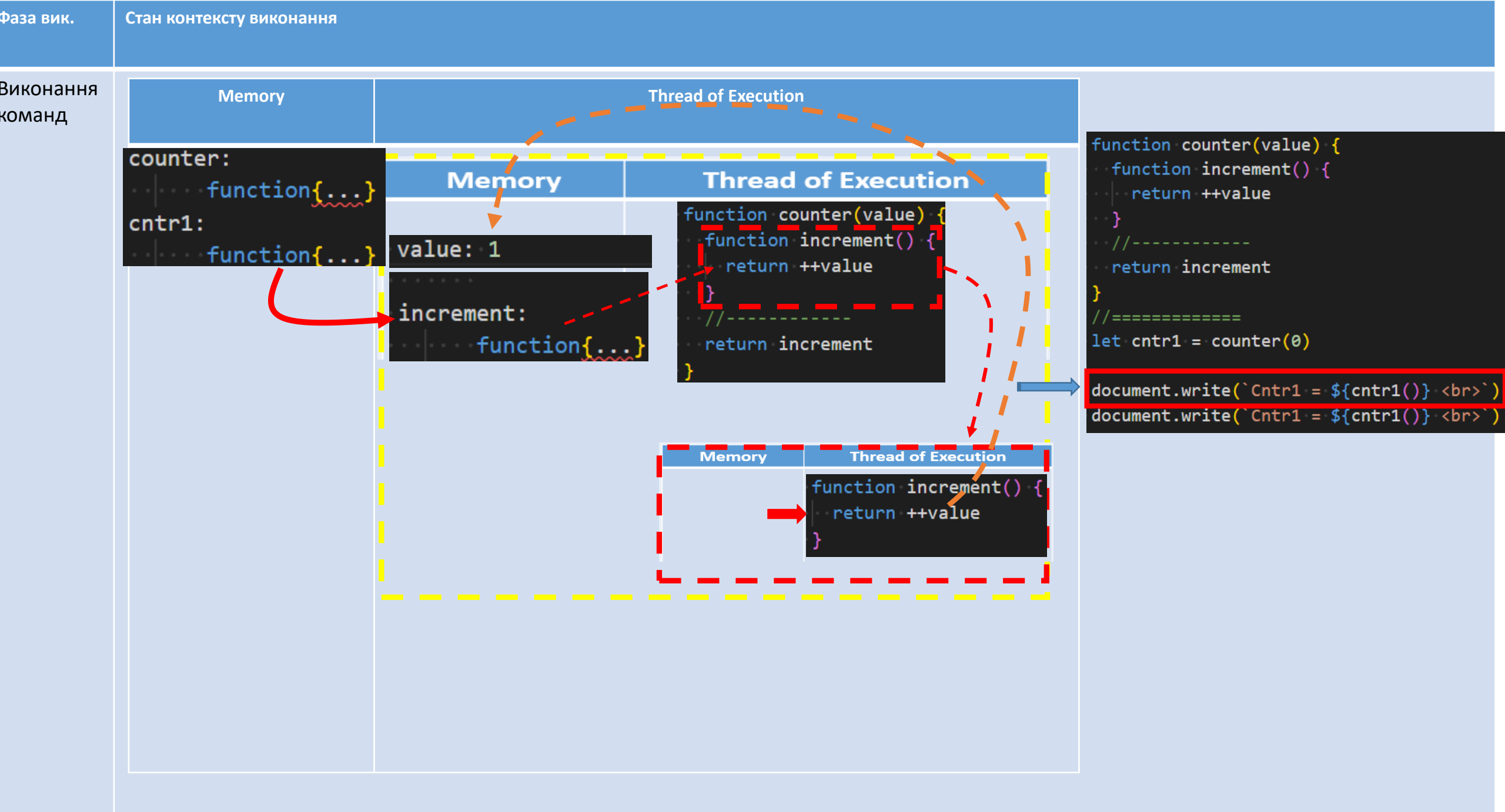


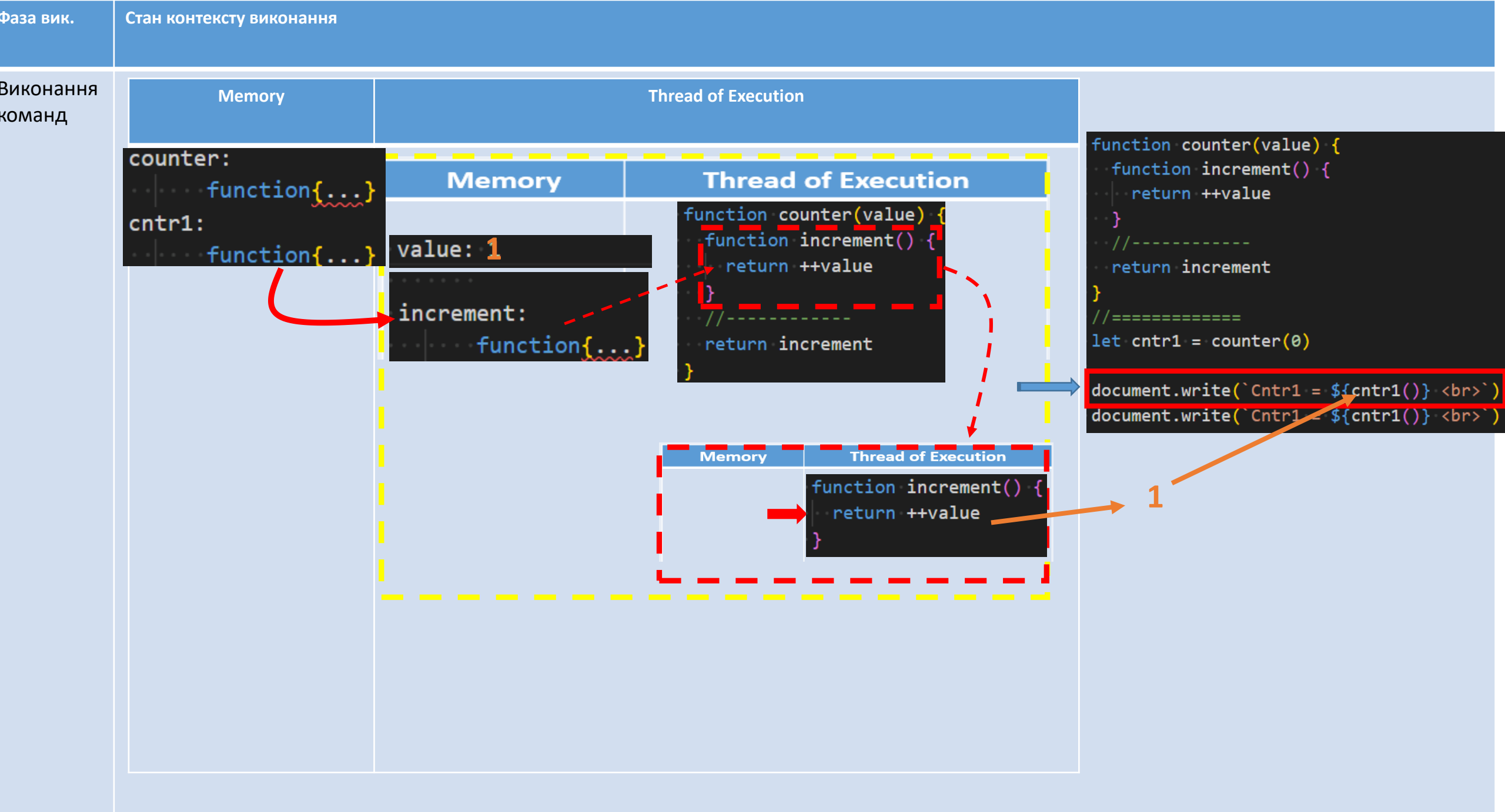


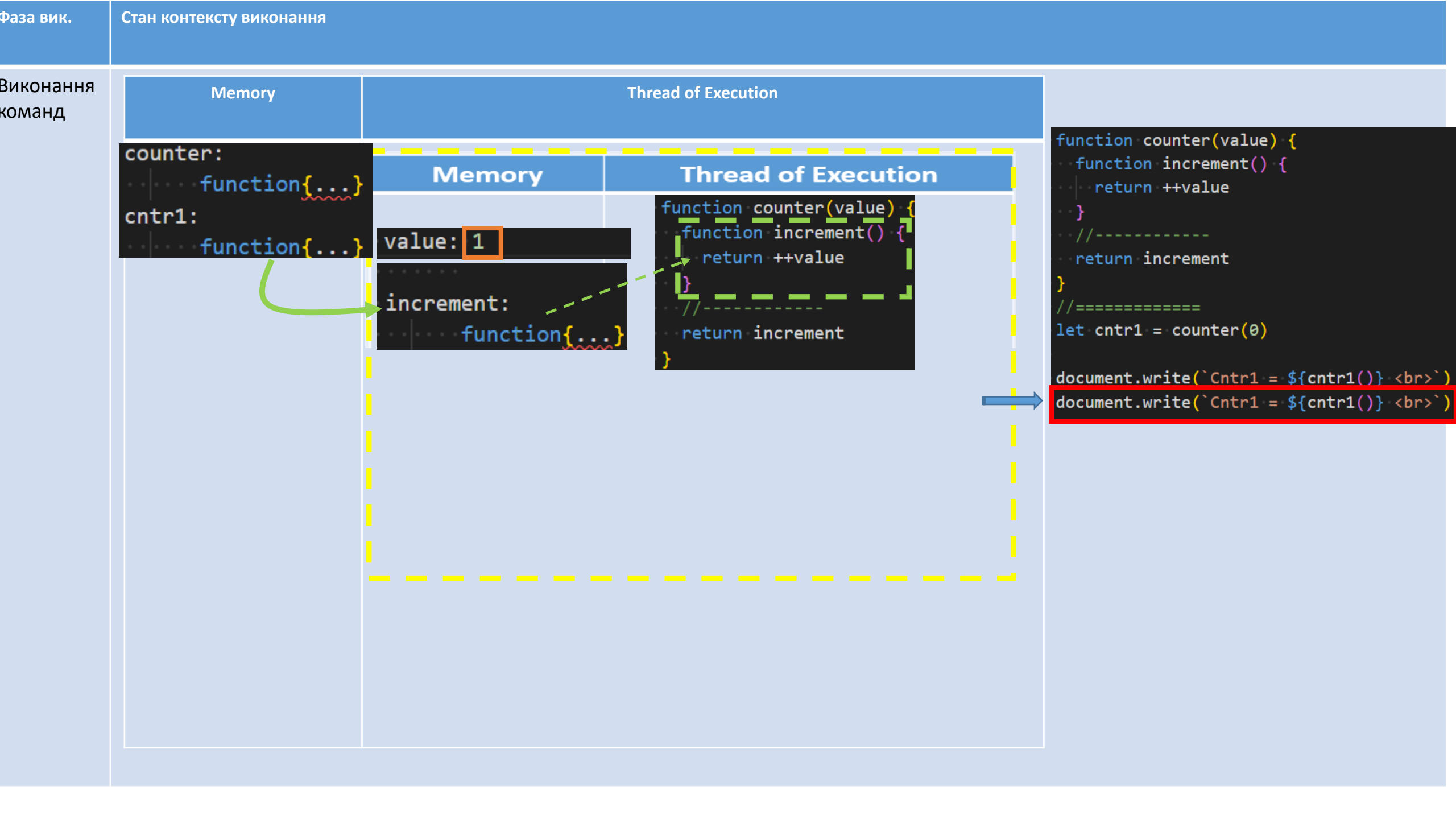


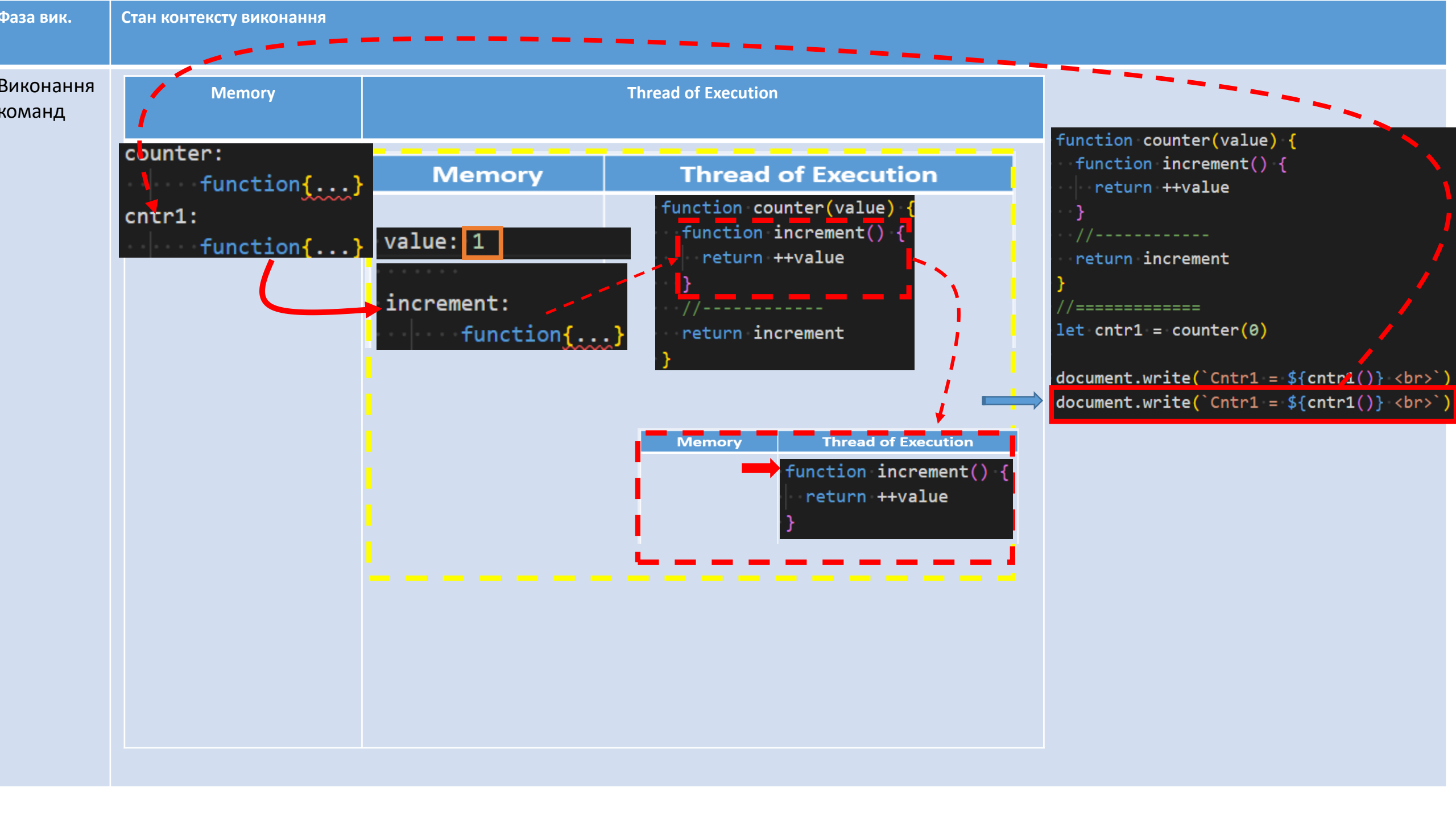


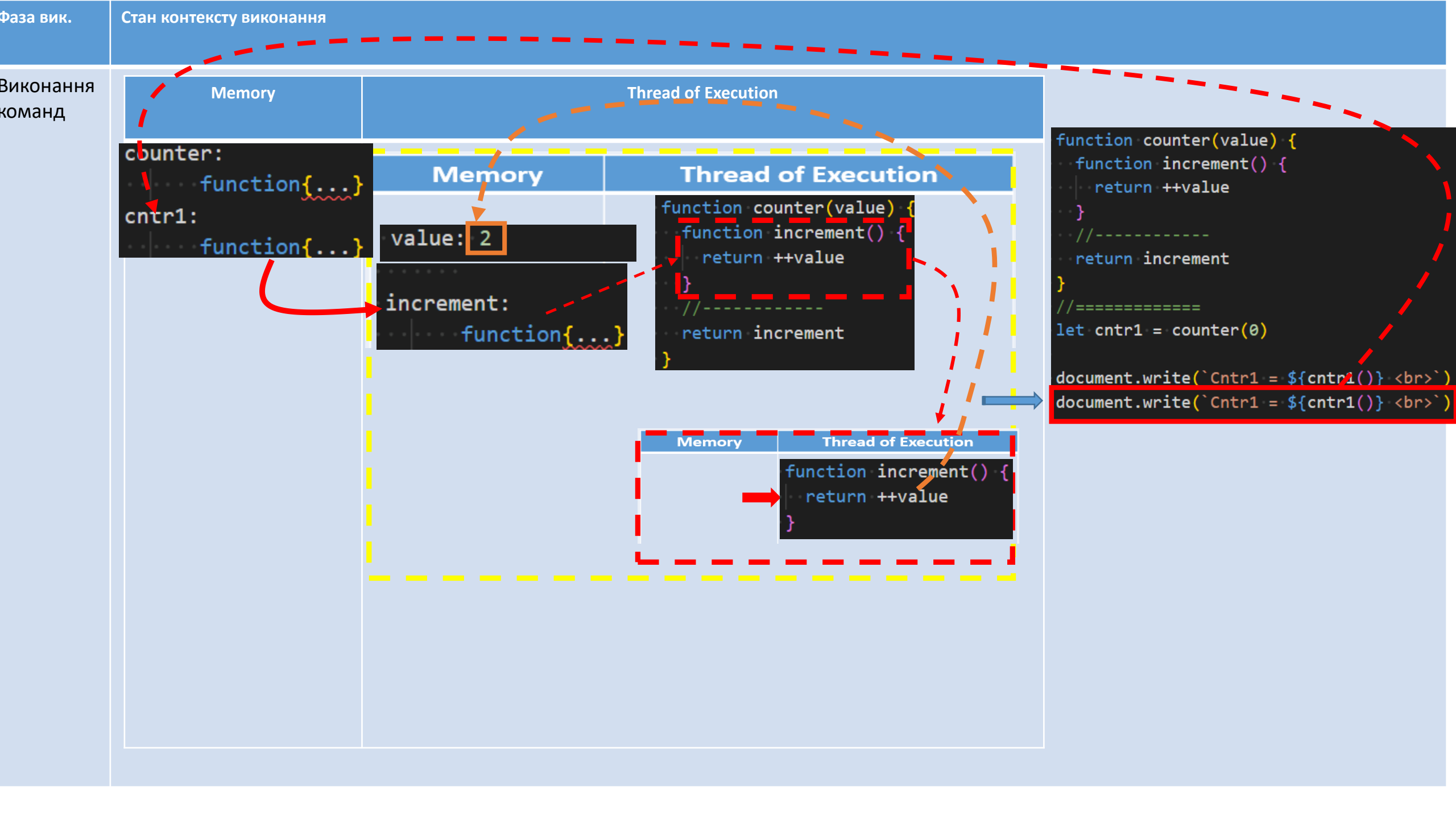


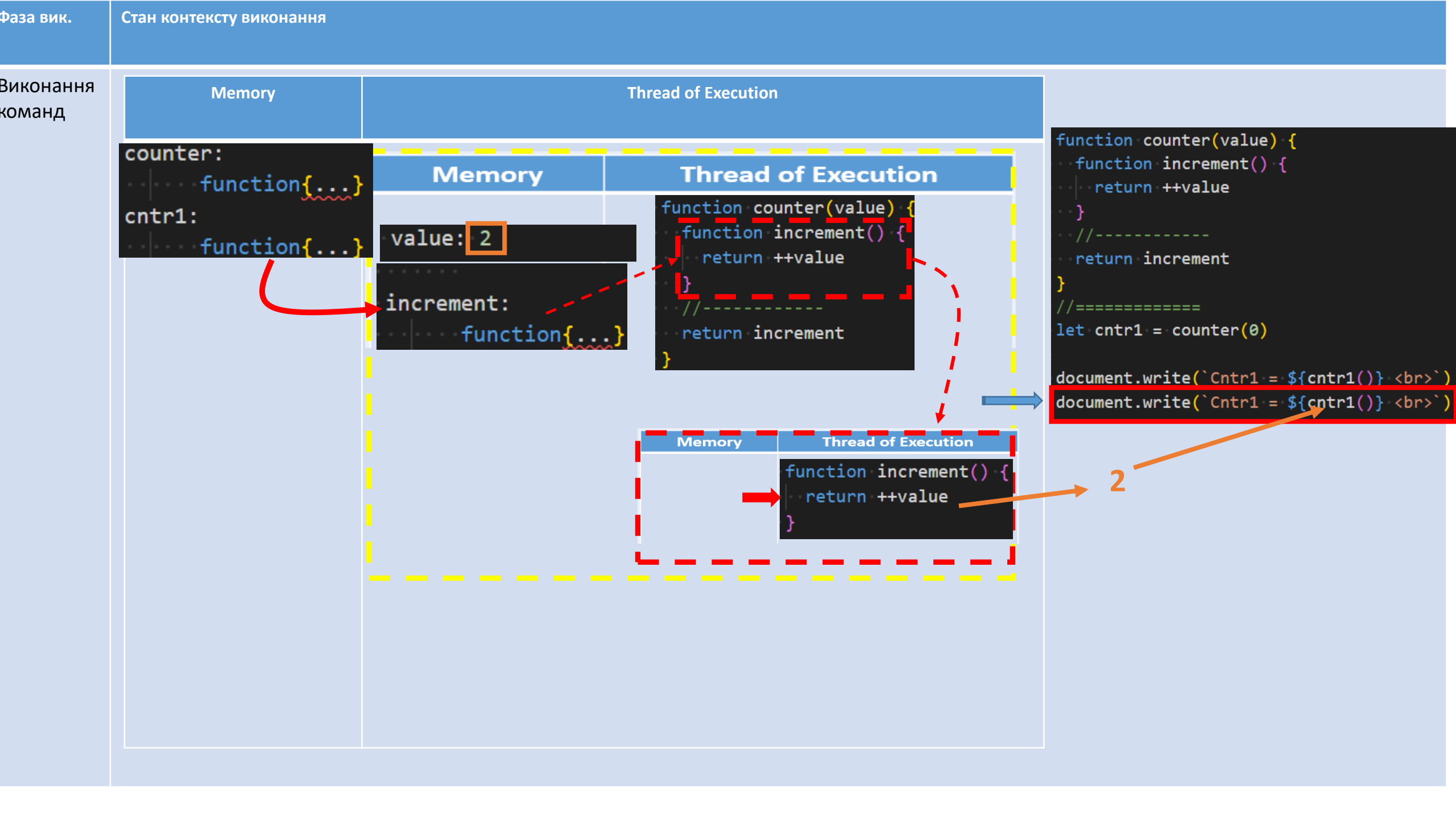


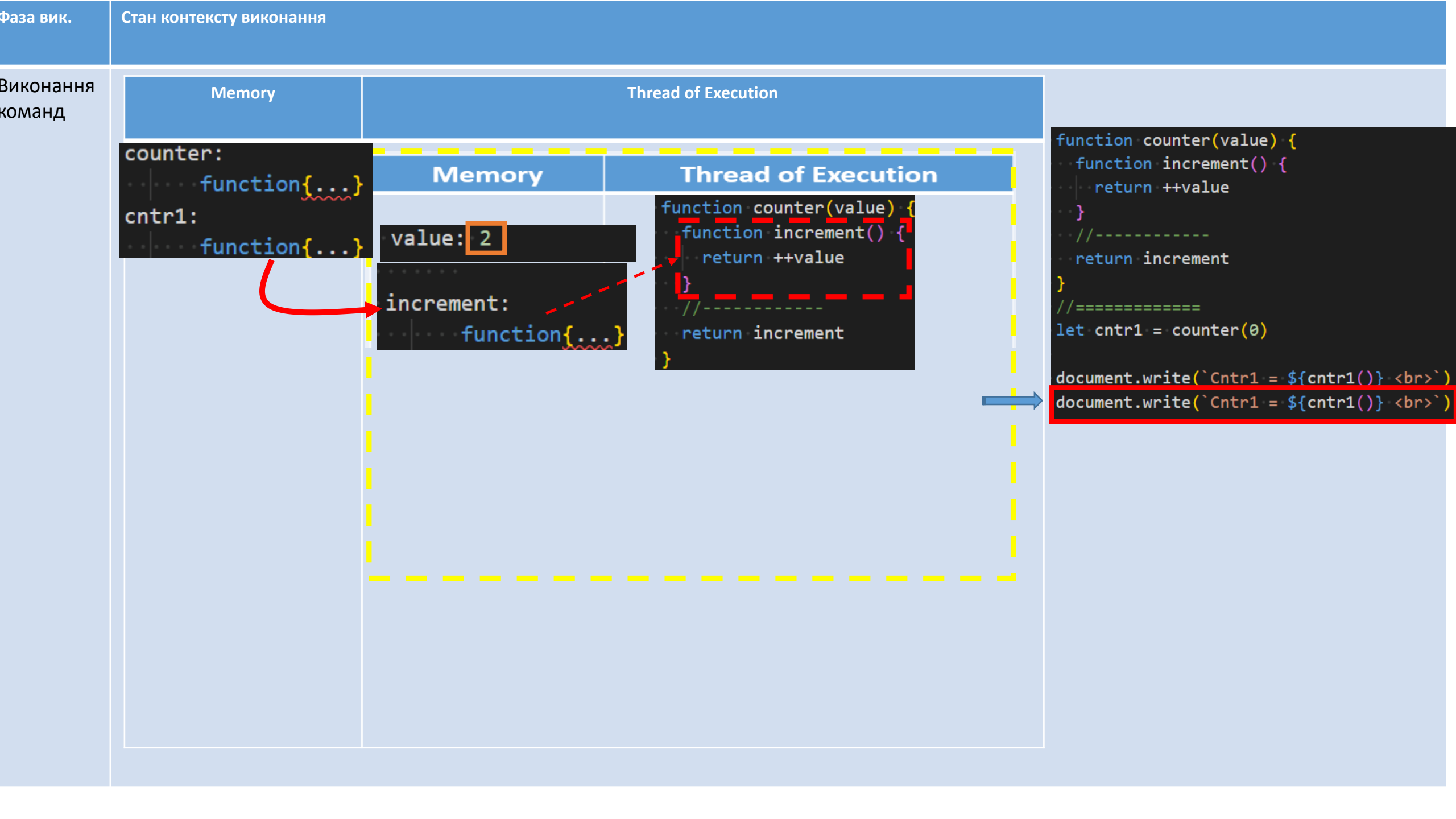








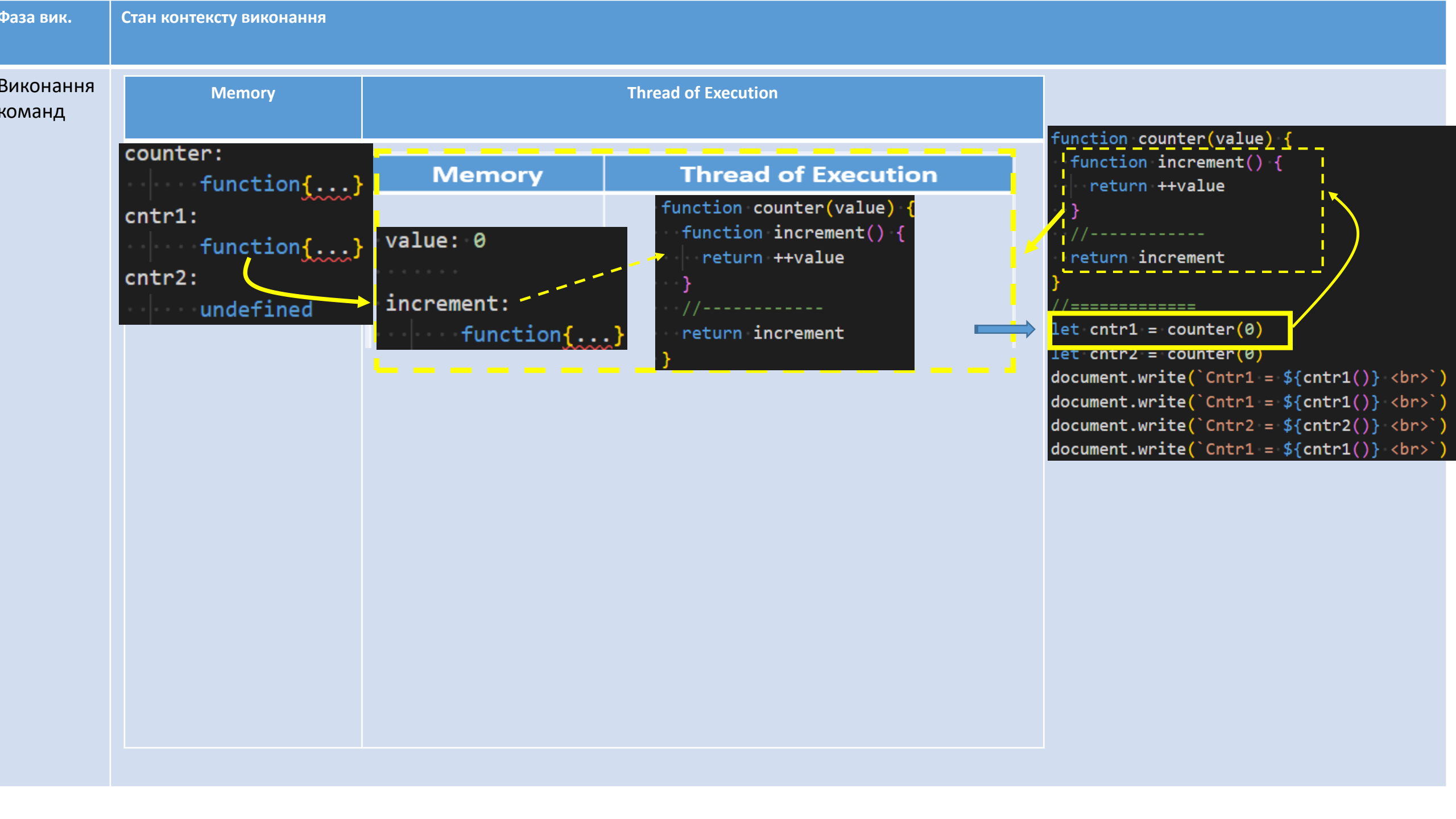


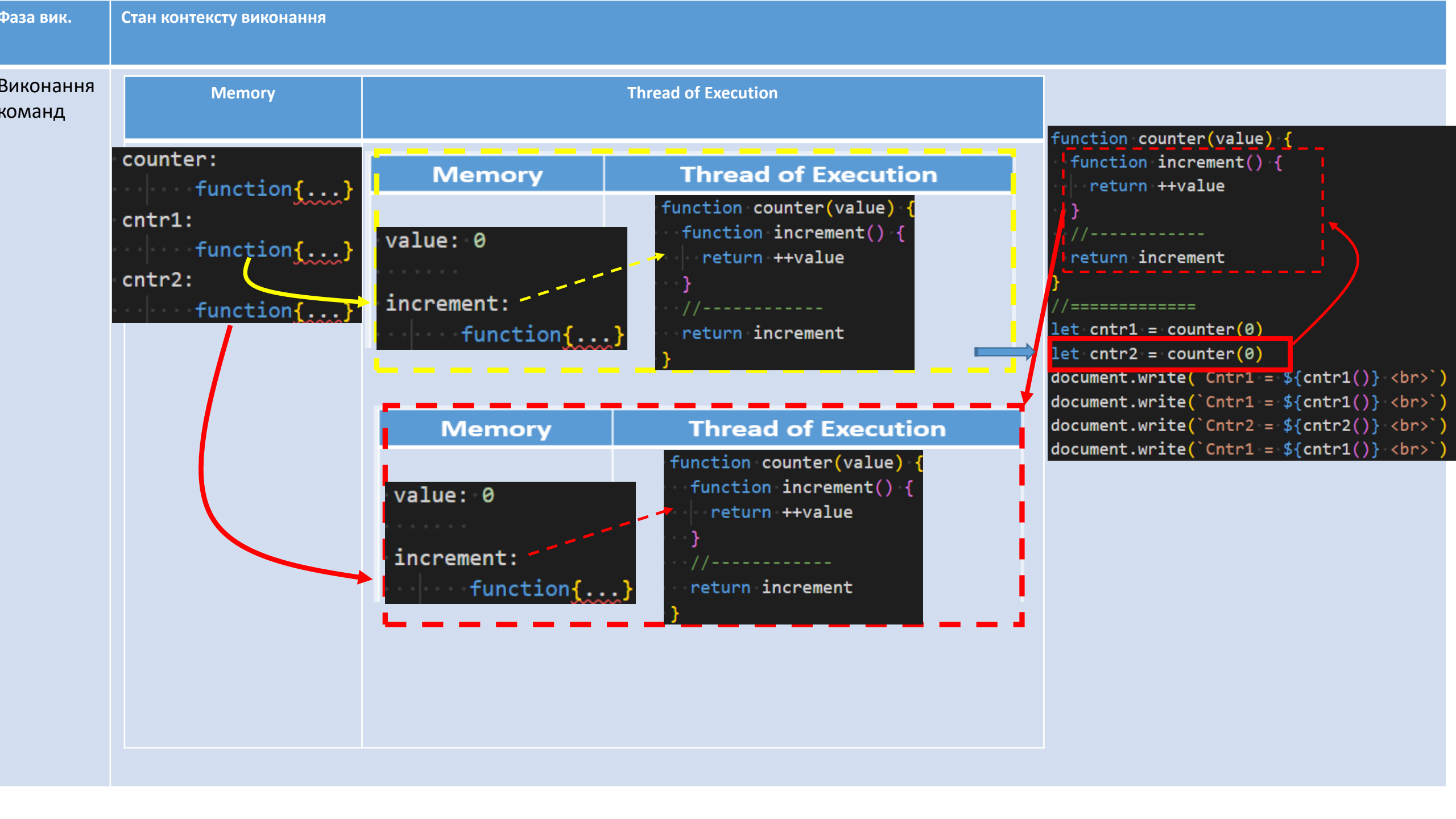


Приклад з двома лічильниками

```
function counter(value) {  
  function increment() {  
    return ++value  
  }  
  //-----  
  return increment  
}  
//=====  
let cntr1 = counter(0)  
let cntr2 = counter(0)  
document.write(`Cntr1 = ${cntr1()} <br>`)  
document.write(`Cntr1 = ${cntr1()} <br>`)  
document.write(`Cntr2 = ${cntr2()} <br>`)  
document.write(`Cntr1 = ${cntr1()} <br>`)
```

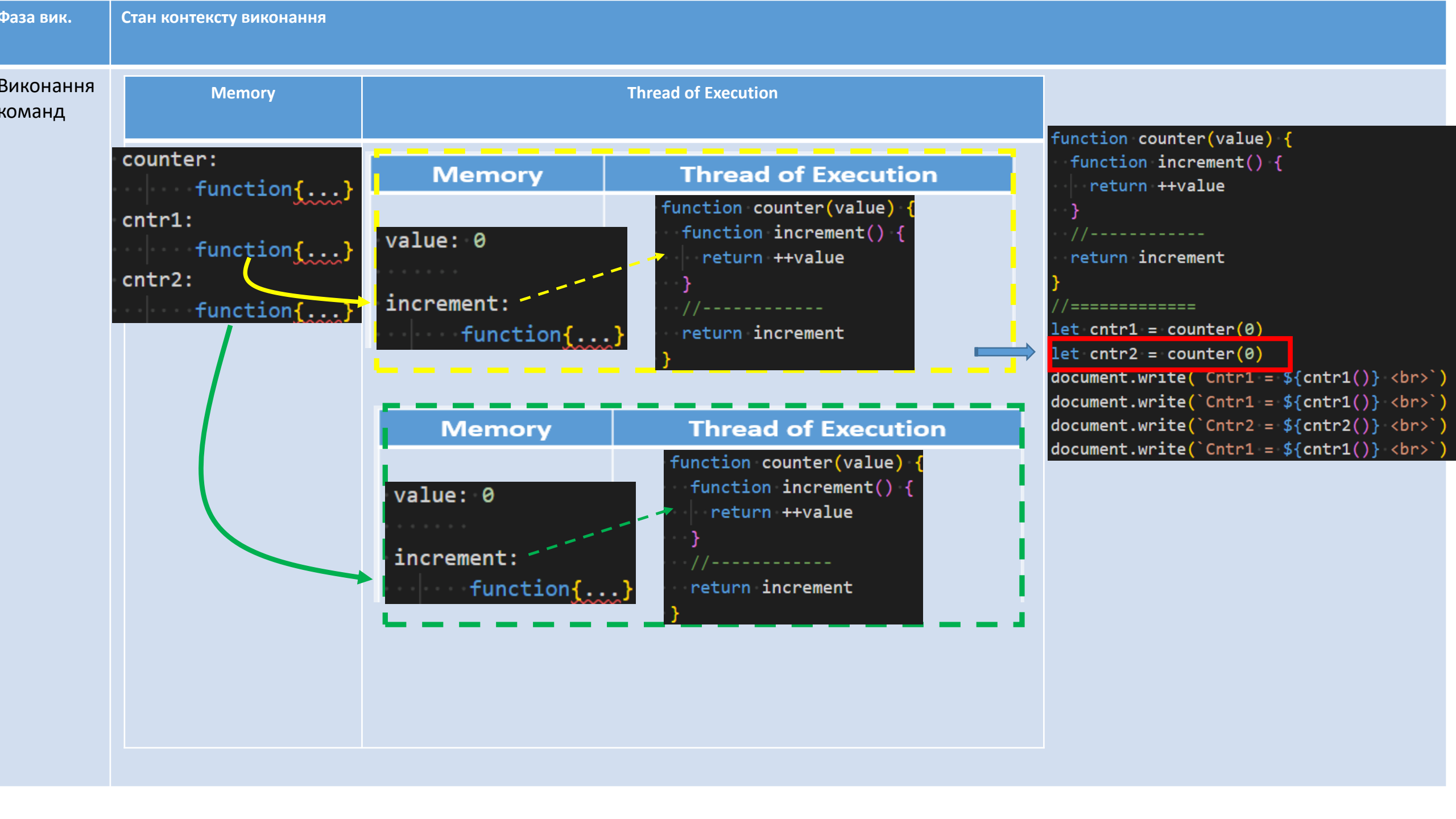
| Фаза вик. | Стан контексту виконання | | |
|------------------|---|---------------------|---|
| Виконання команд | Memory | Thread of Execution | |
| | <pre>counter: . . . function{...} cntr1: . . . undefined cntr2: . . . undefined</pre> | | <pre>function counter(value){ function increment(){ return ++value } //----- return increment } //===== let cntr1 = counter(0) let cntr2 = counter(0) document.write(`Cntr1 = \${cntr1()} `) document.write(`Cntr1 = \${cntr1()} `) document.write(`Cntr2 = \${cntr2()} `) document.write(`Cntr1 = \${cntr1()} `)</pre> |

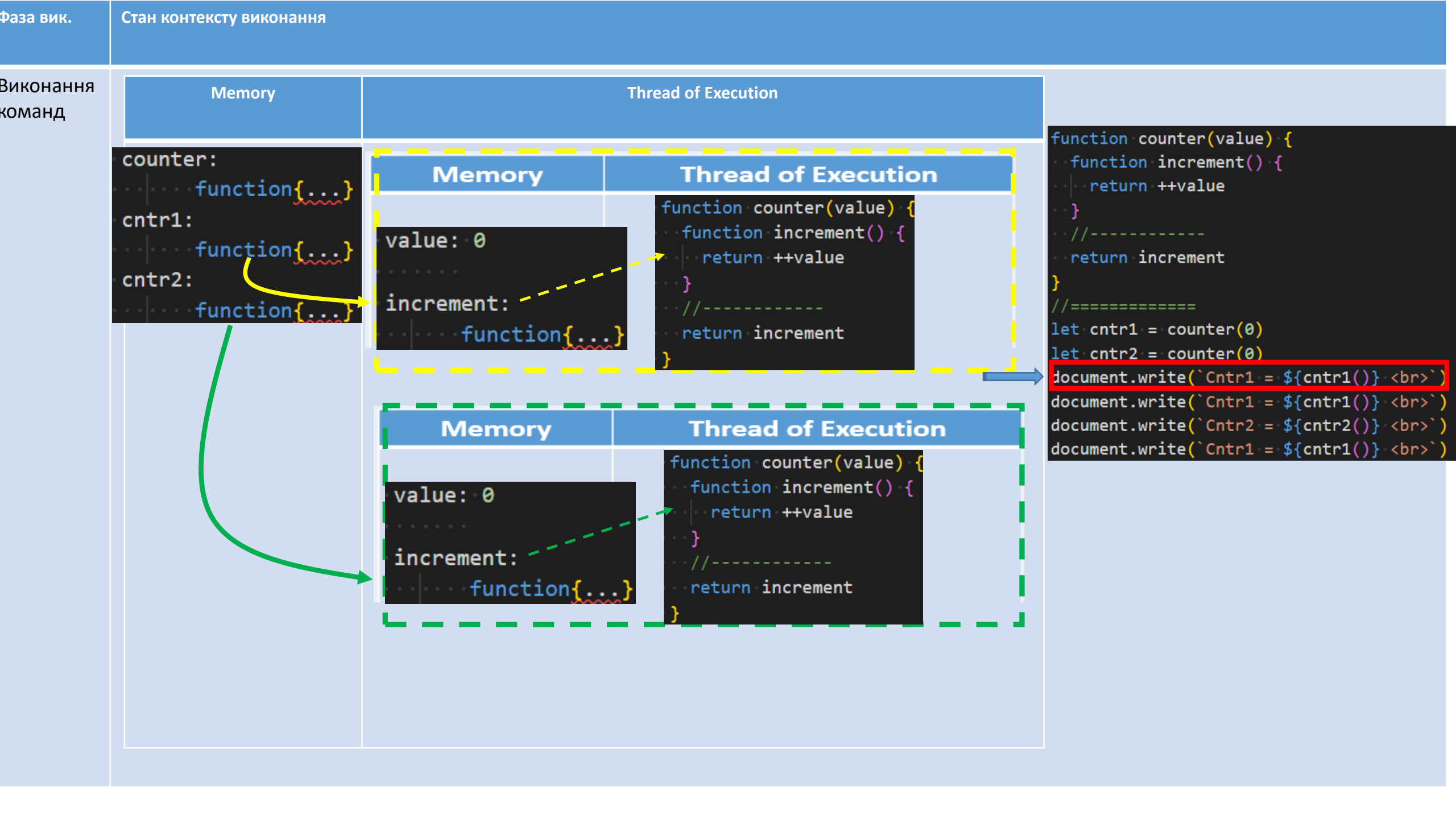




```
function counter(value) {
  function increment() {
    return ++value
  }
  //-----
  return increment
}

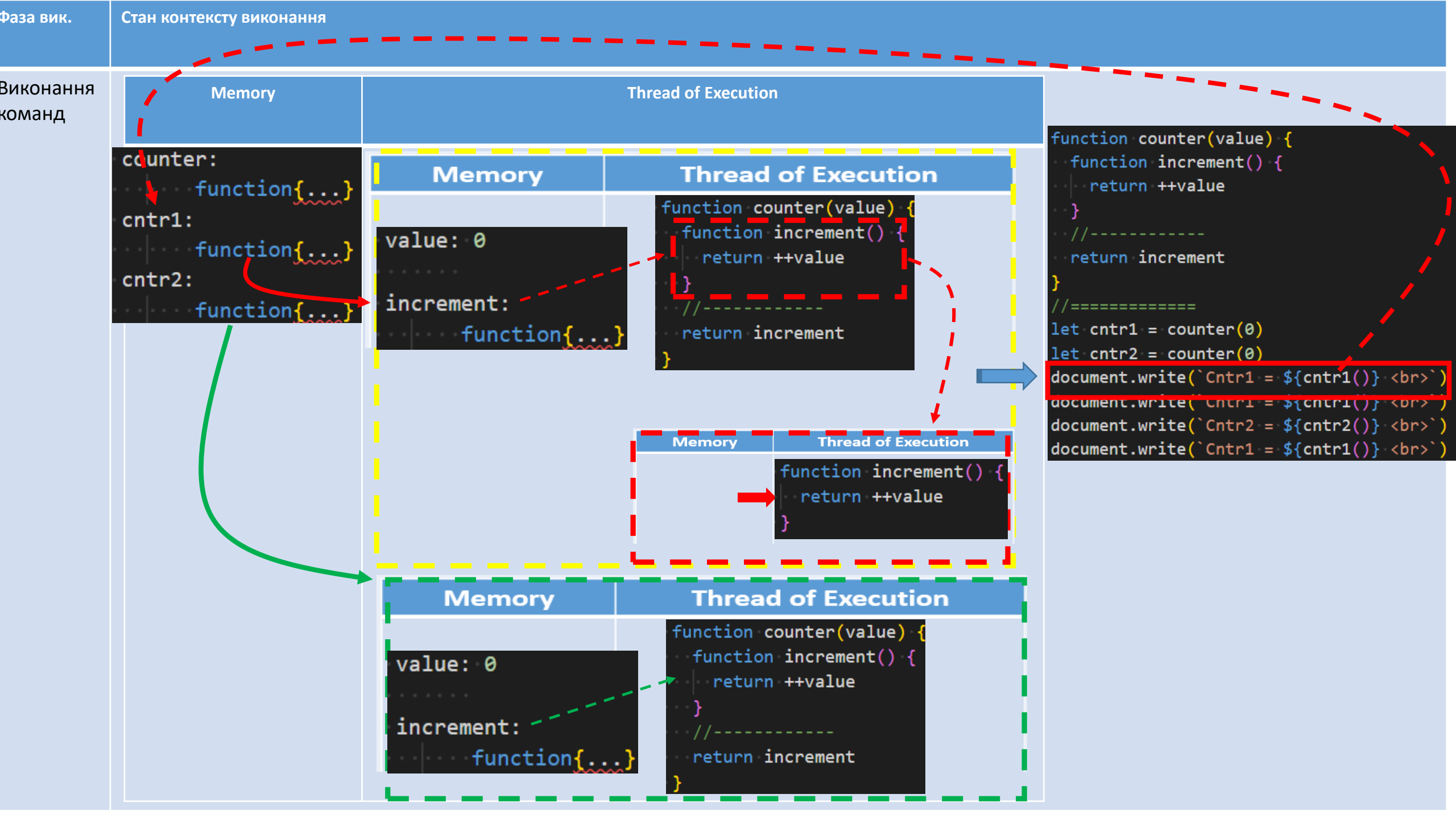
//=====
let cntr1 = counter(0)
let cntr2 = counter(0)
document.write( `Cntr1 = ${cntr1()} <br>` )
document.write( `Cntr1 = ${cntr1()} <br>` )
document.write( `Cntr2 = ${cntr2()} <br>` )
document.write( `Cntr1 = ${cntr1()} <br>` )
```

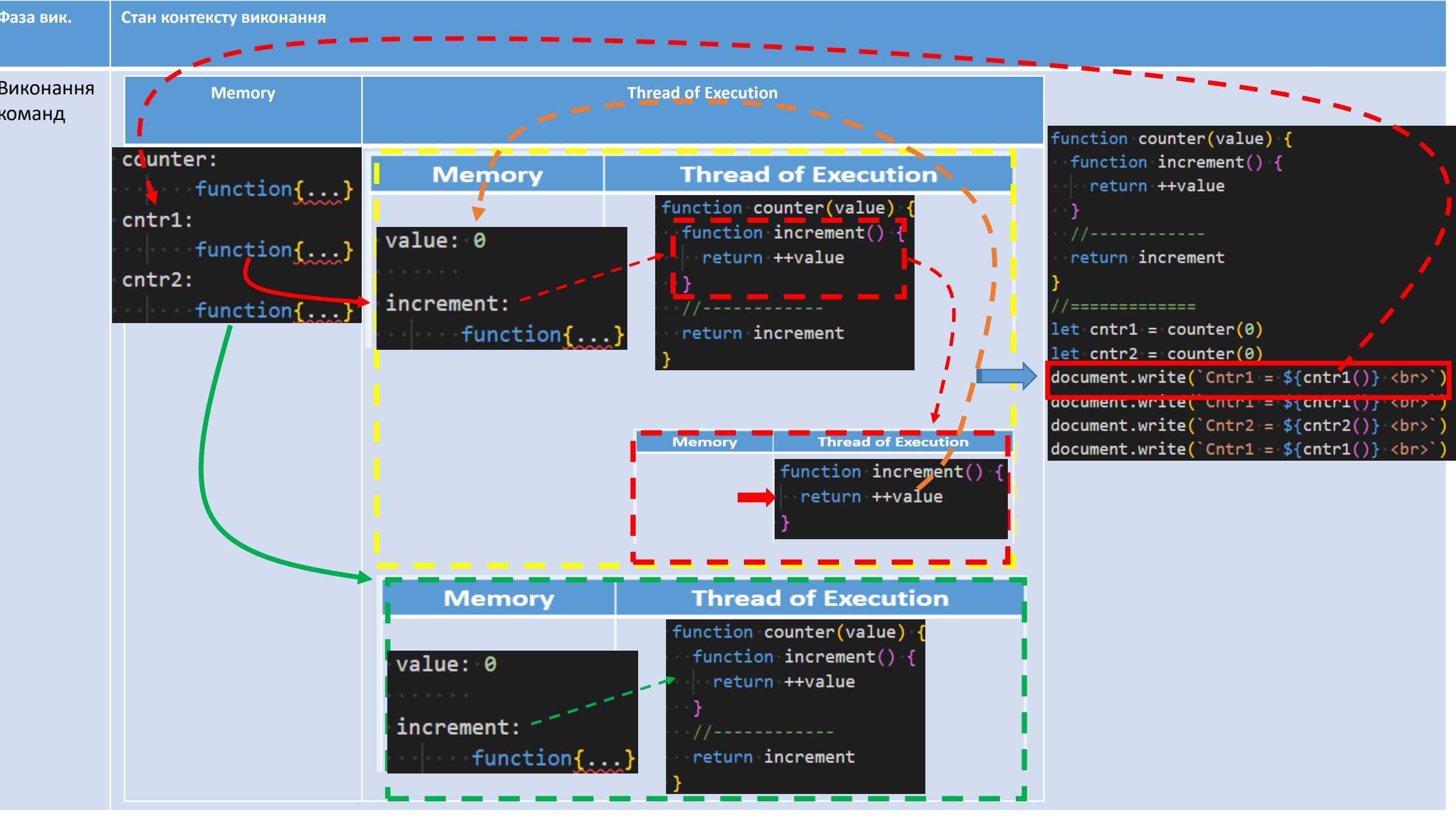


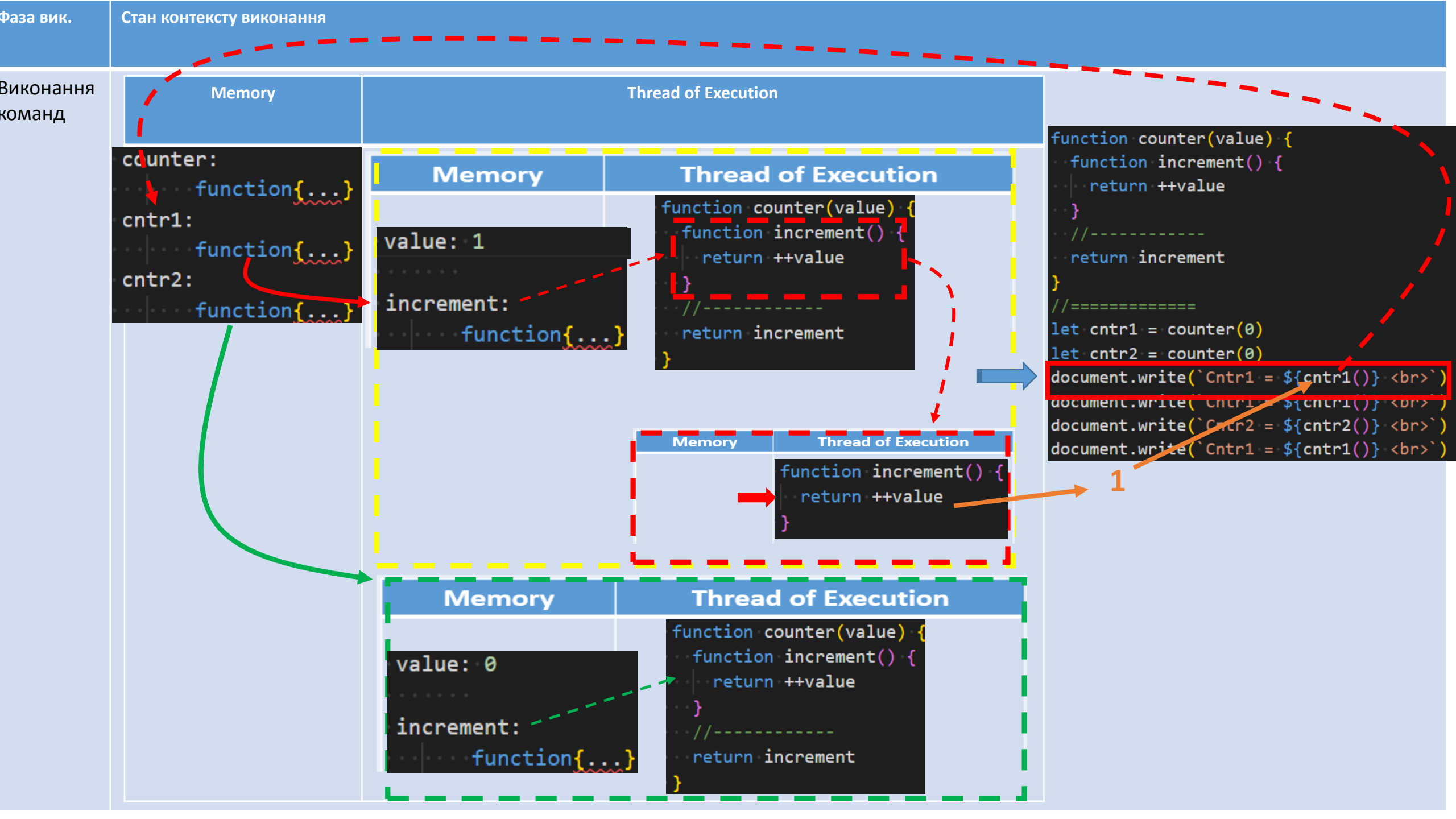


```
function counter(value) {
  function increment() {
    return ++value
  }
  //-----
  return increment
}

//=====
let cntr1 = counter(0)
let cntr2 = counter(0)
document.write(`Cntr1 = ${cntr1()}<br>`)
document.write(`Cntr1 = ${cntr1()}<br>`)
document.write(`Cntr2 = ${cntr2()}<br>`)
document.write(`Cntr1 = ${cntr1()}<br>`)
```

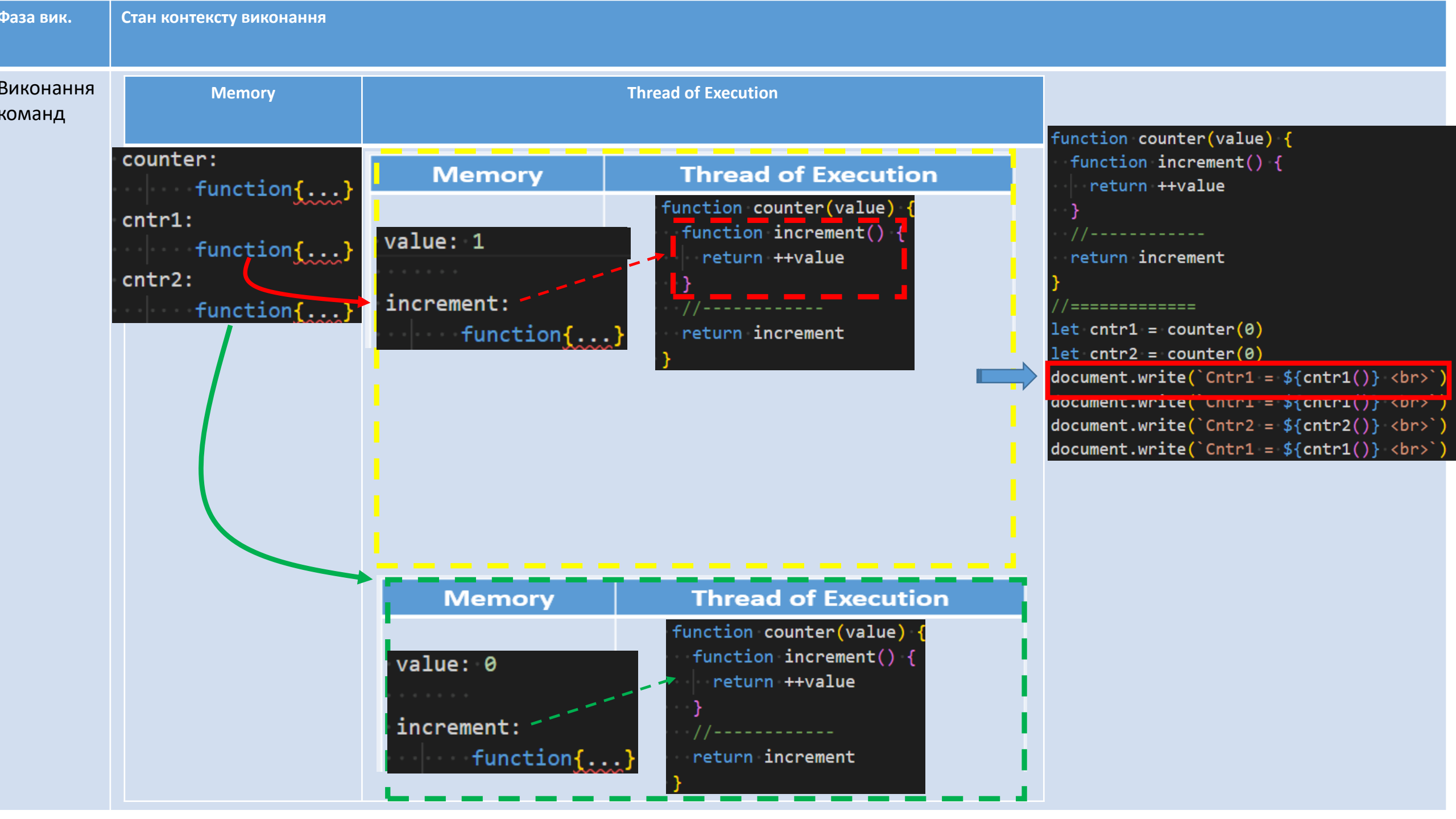


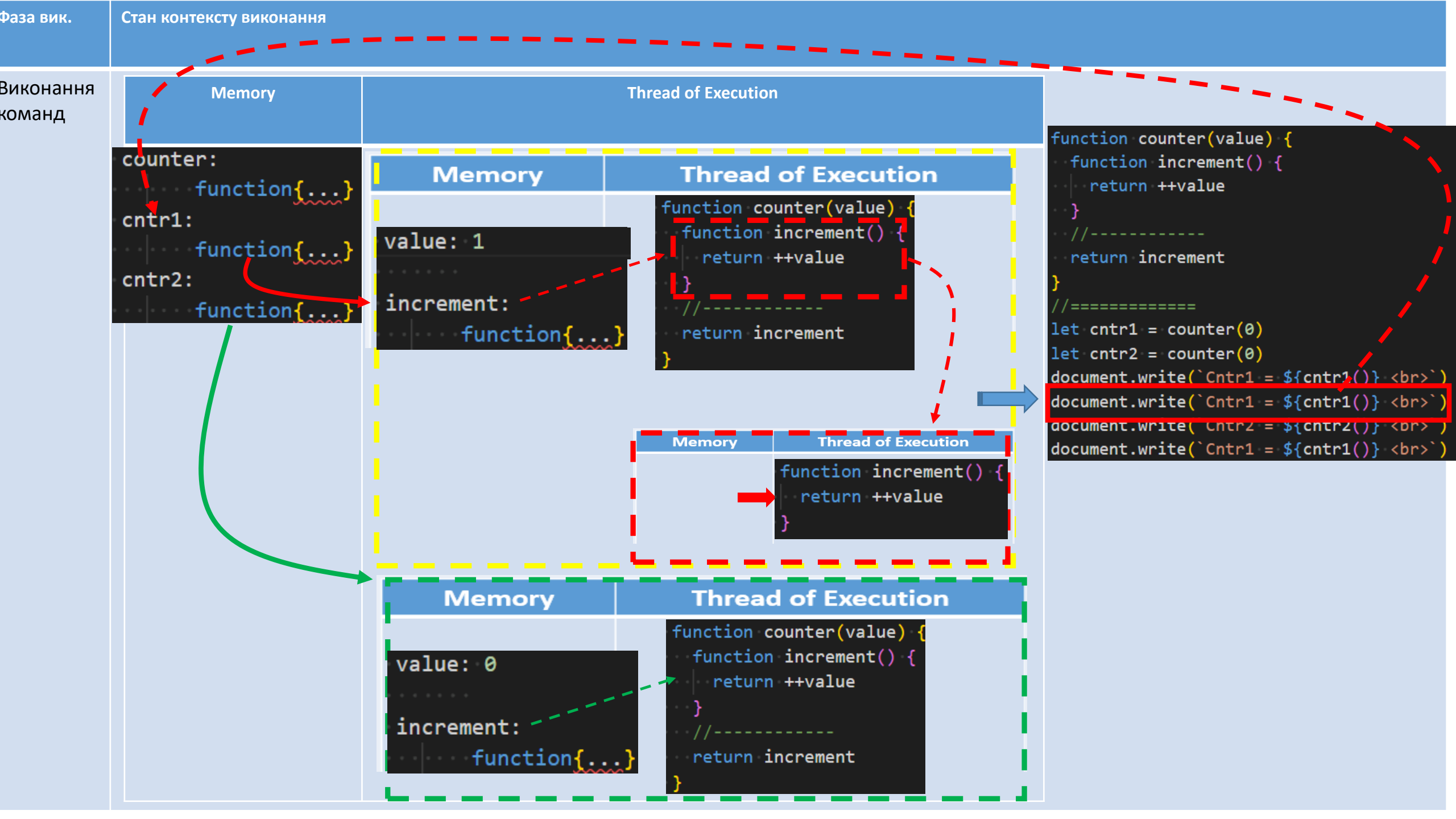


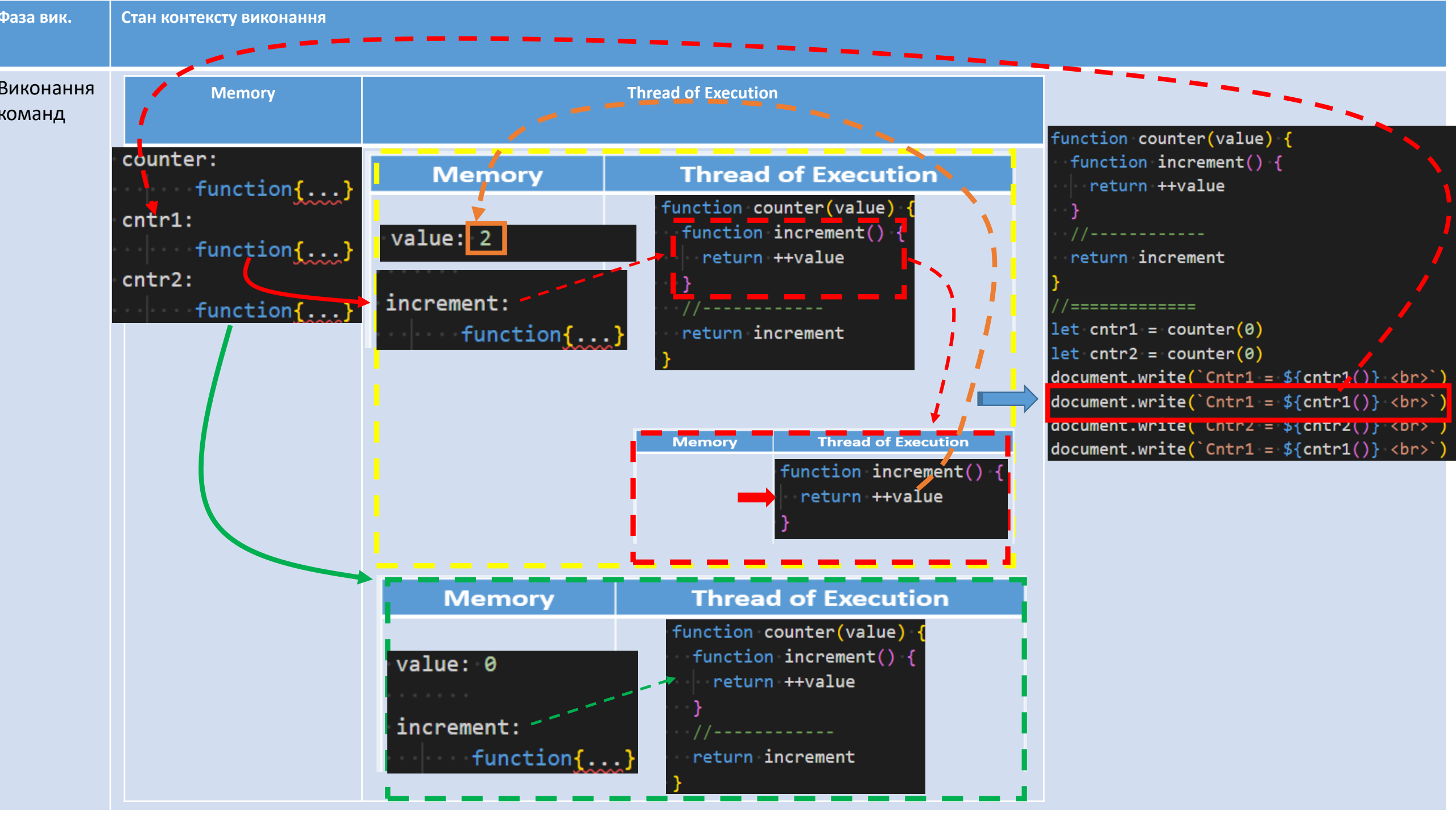


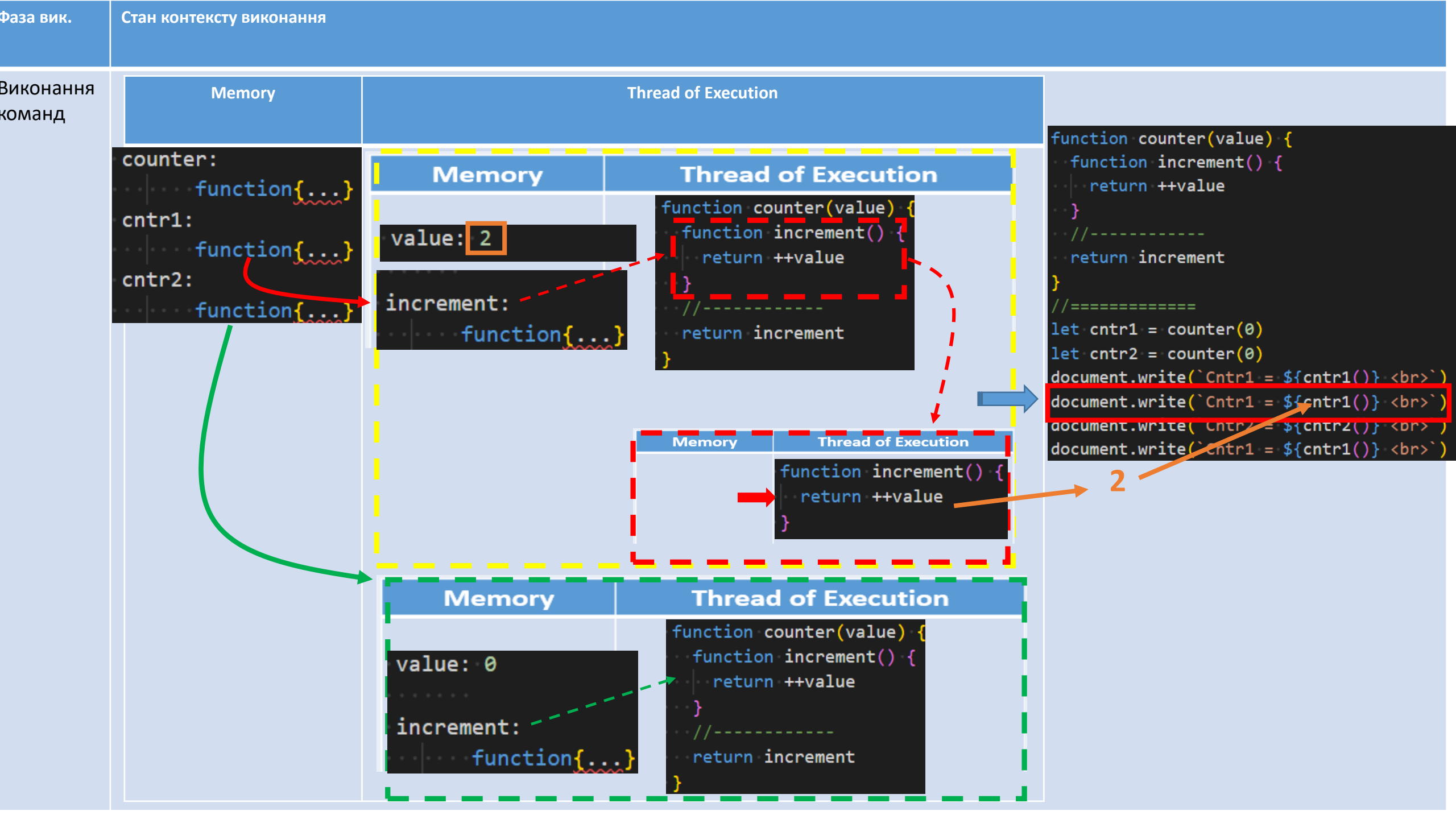
```
function counter(value) {
  function increment() {
    return ++value
  }
  //-----
  return increment
}

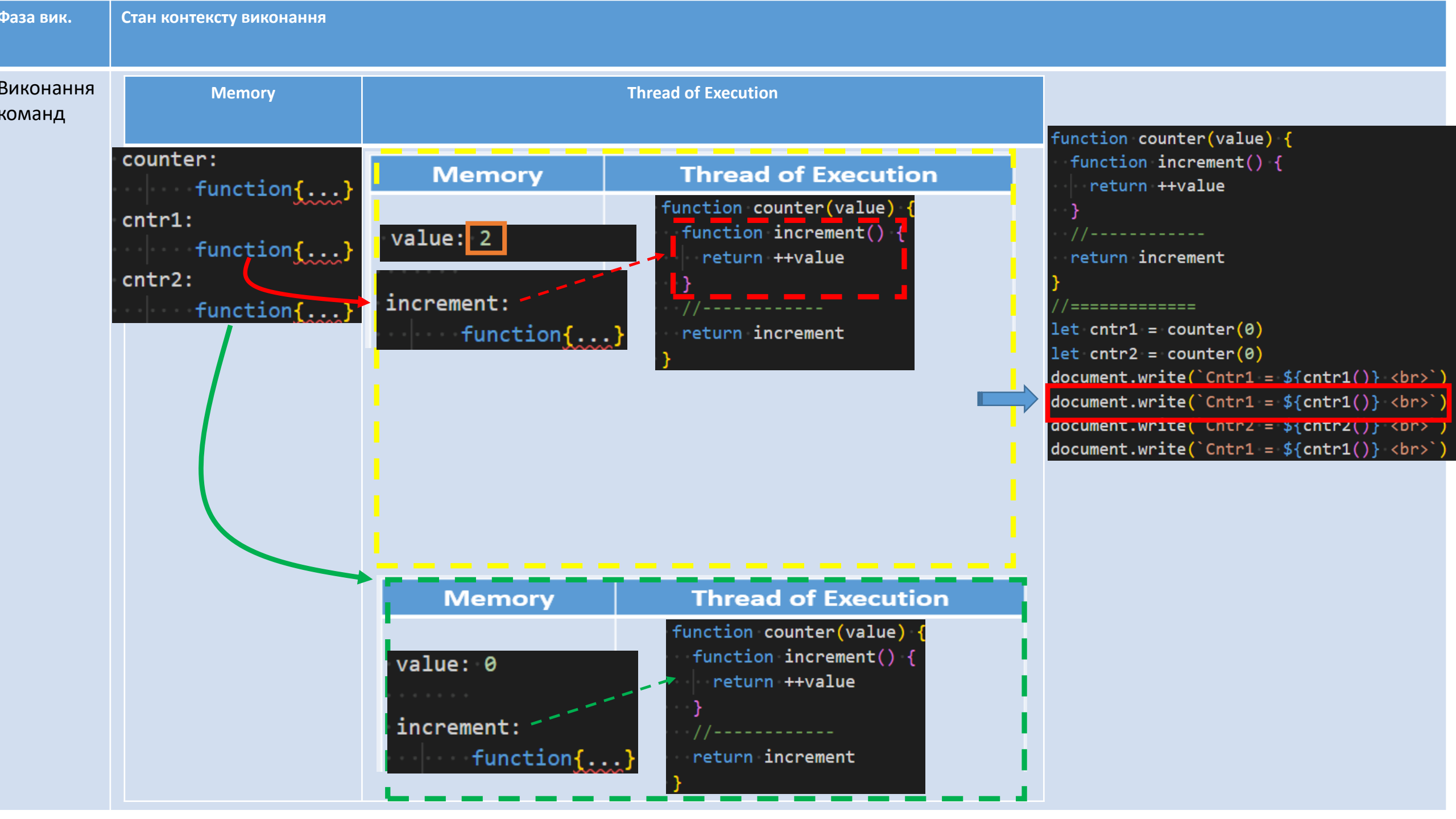
//=====
let cntr1 = counter(0)
let cntr2 = counter(0)
document.write(`Cntr1 = ${cntr1()}<br>`)
document.write(`Cntr1 = ${cntr1()}<br>`)
document.write(`Cntr2 = ${cntr2()}<br>`)
document.write(`Cntr1 = ${cntr1()}<br>`)
```

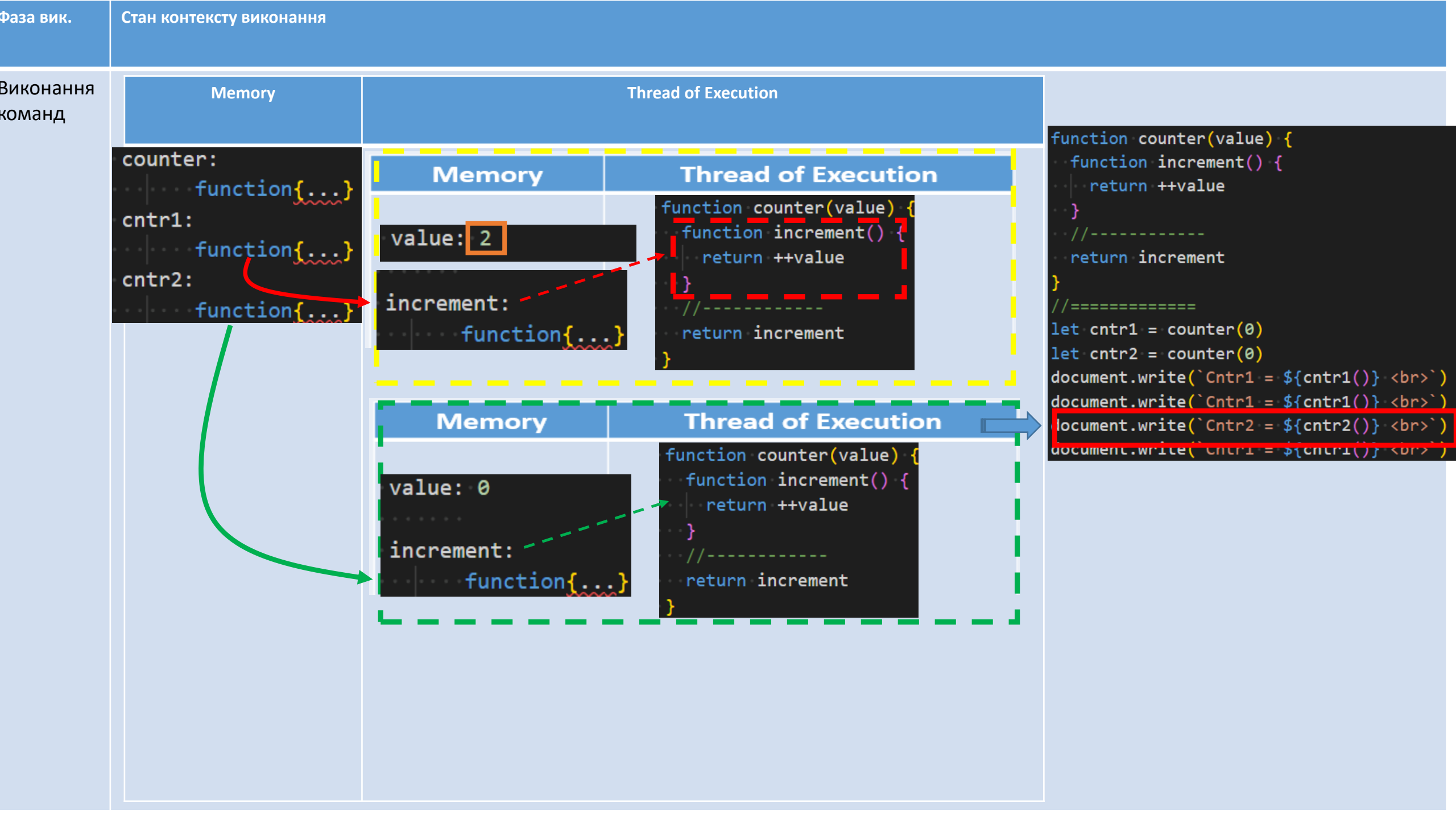


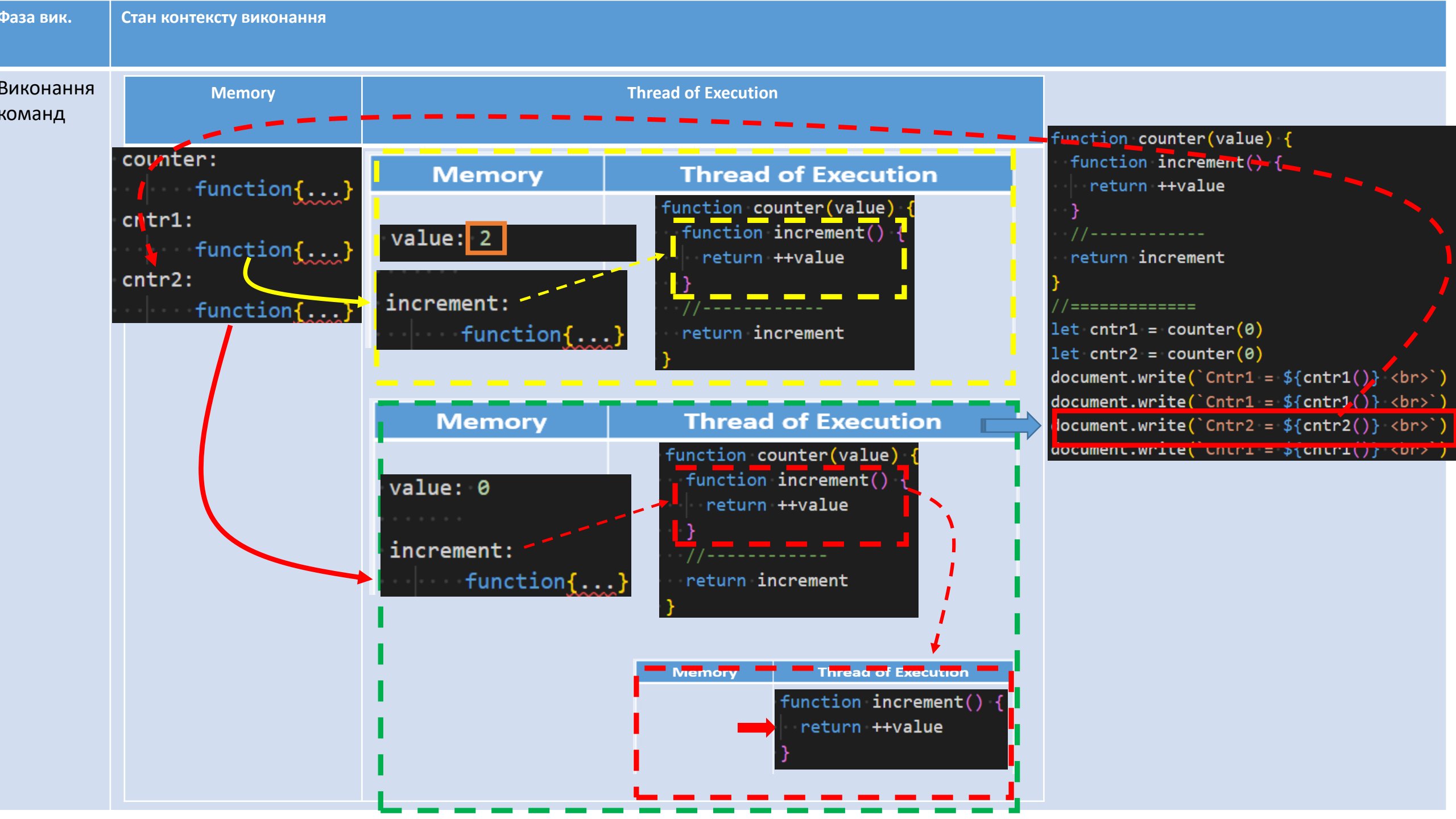












```
function counter(value) {  
  function increment() {  
    return ++value  
  }  
  //-----  
  return increment  
}  
  
//=====  
let cntr1 = counter(0)  
let cntr2 = counter(0)  
document.write(`Cntr1 = ${cntr1()}<br>`)  
document.write(`Cntr1 = ${cntr1()}<br>`)  
document.write(`Cntr2 = ${cntr2()}<br>`)  
document.write(`Cntr1 = ${cntr1()}<br>`)
```

Фаза вик.

Виконання команд

The diagram consists of a solid blue rectangle. A red dashed line starts at the left edge of the rectangle, slightly below the top, and slopes downwards to the right, ending near the bottom right corner. The text 'Thread of Execution' is centered in white above the rectangle.

```
counter:
  ... function{...}
ctr1:
  ... function{...}
ctr2:
  ... function{...}
```

Memory

Thread of Execution

```
value: 2
```

```
function counter(value) {  
  function increment() {  
    return ++value  
  }  
  // -----  
  return increment  
}
```

```
increment: function{...}
```



Memory

Thread of Execution

```
value: 1
```

```
function counter(value) {
  function increment() {
    return ++value
  }
  // -----
  return increment
}
```

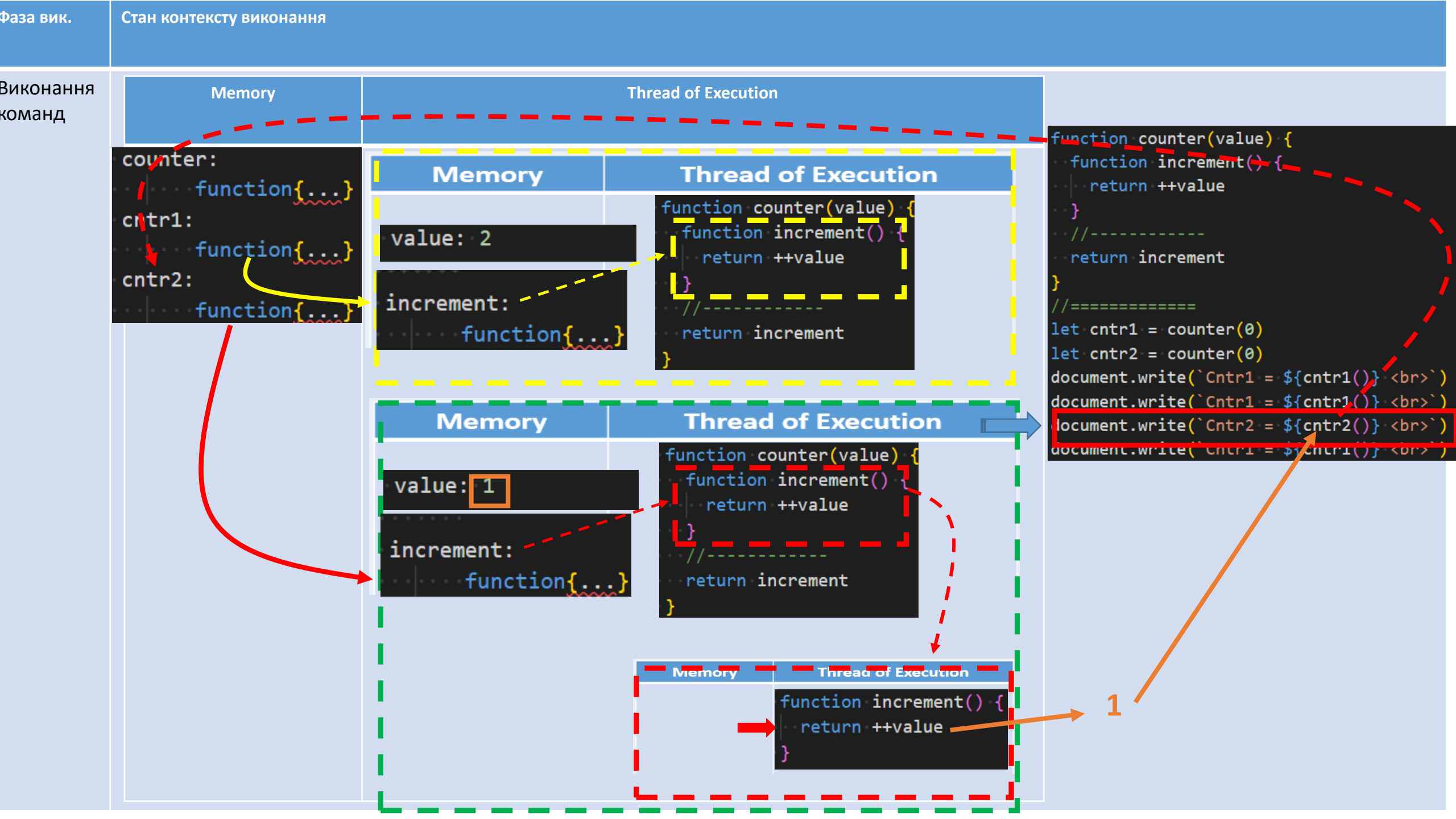
```
increment:
  function{...}
```

Memory

Thread of Execution

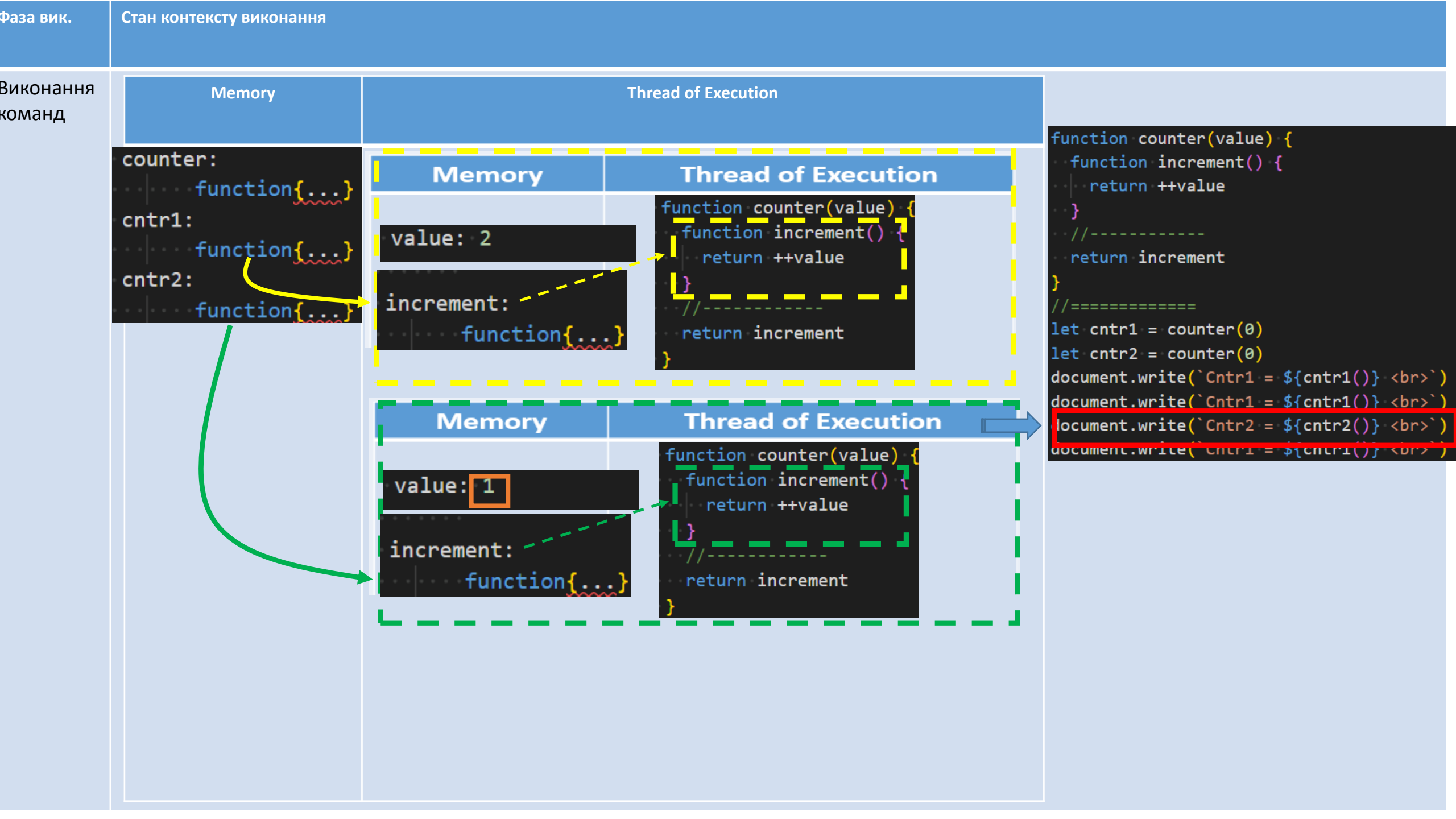
```
function increment() {  
  return ++value  
}
```

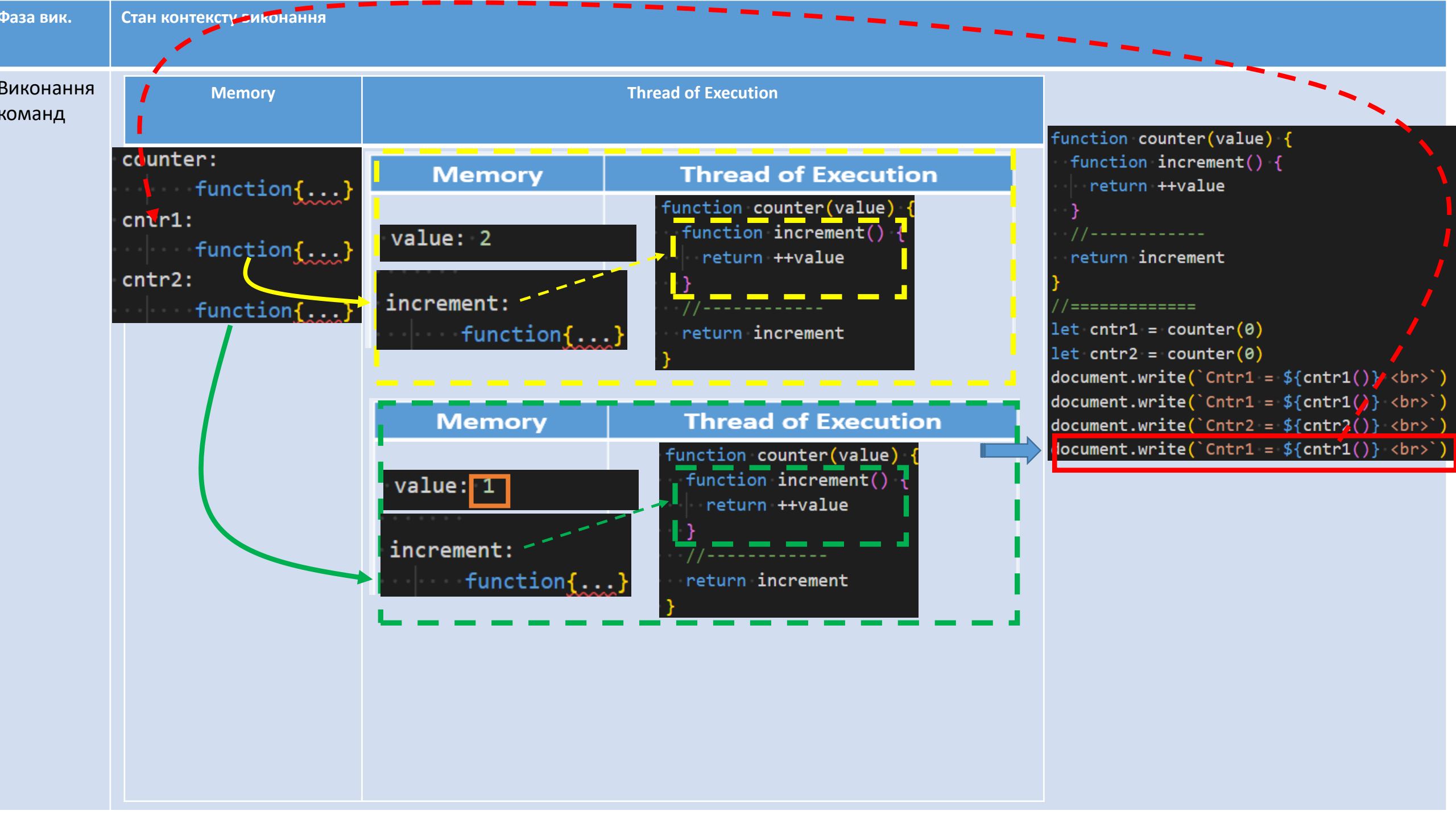
```
function counter(value) {  
  function increment() {  
    return ++value  
  }  
  //-----  
  return increment  
}  
  
//=====  
let cnt1 = counter(0)  
let cnt2 = counter(0)  
document.write(`Cnt1 = ${cnt1()} <br>`)  
document.write(`Cnt1 = ${cnt1()} <br>`)  
document.write(`Cnt2 = ${cnt2()} <br>`)  
document.write(`Cnt1 = ${cnt1()} <br>`)
```

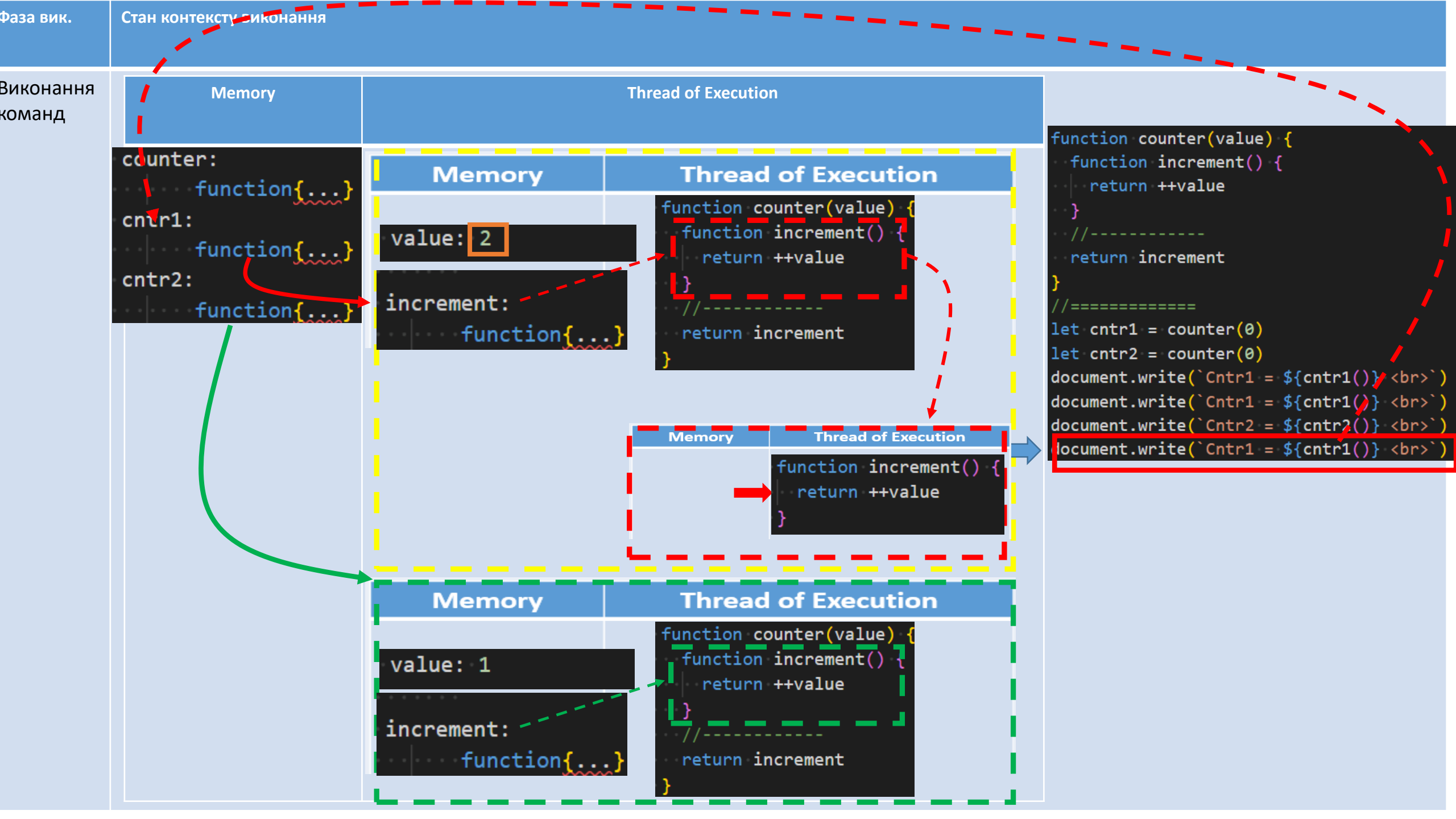


```
function counter(value) {  
  function increment() {  
    return ++value  
  }  
  //-----  
  return increment  
}  
  
//=====  
let cntr1 = counter(0)  
let cntr2 = counter(0)  
document.write(`Cntr1 = ${cntr1()}<br>`)  
document.write(`Cntr1 = ${cntr1()}<br>`)  
document.write(`Cntr2 = ${cntr2()}<br>`)  
document.write(`Cntr1 = ${cntr1()}<br>`)
```

1

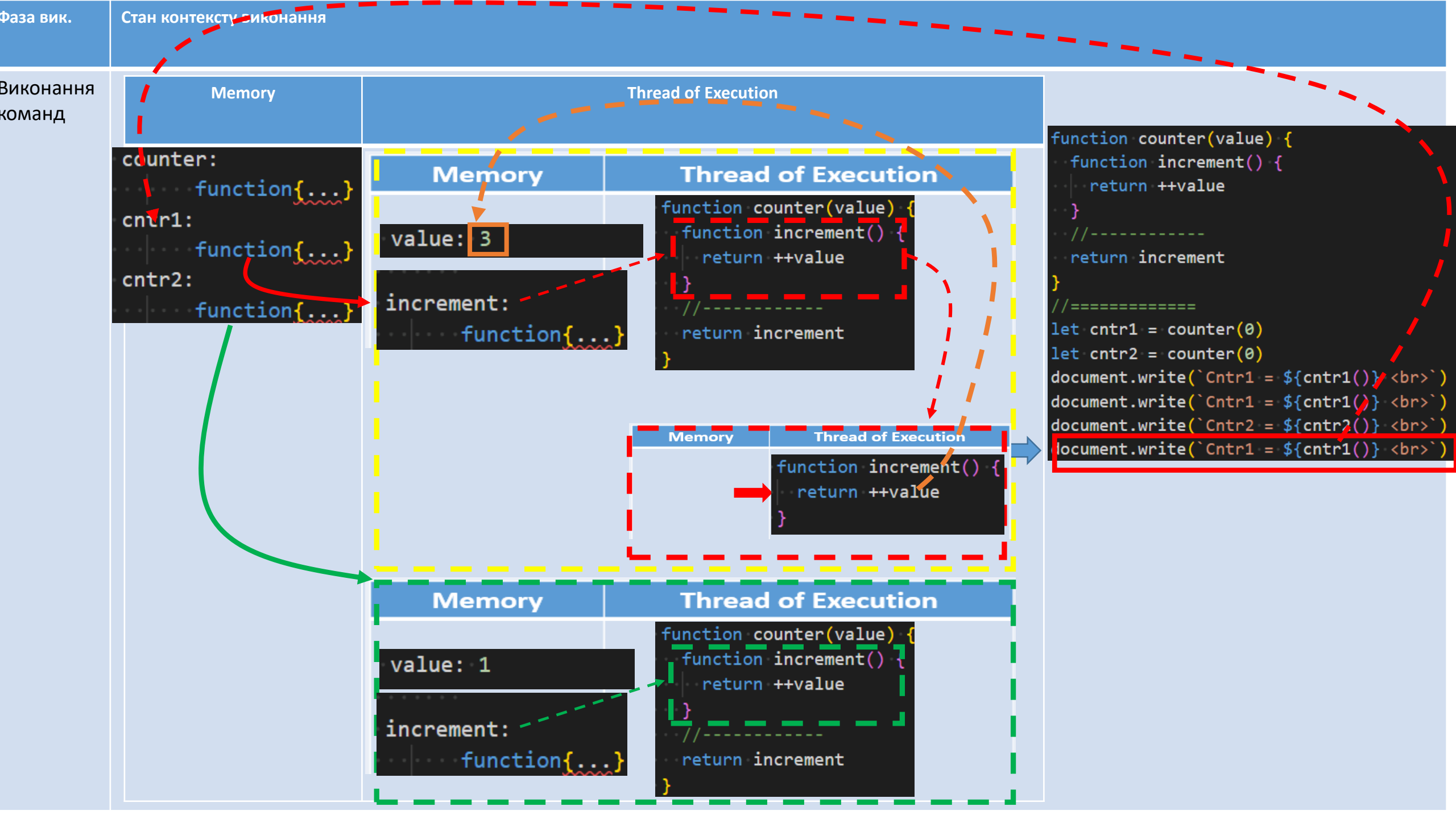






```
function counter(value) {
  function increment() {
    return ++value
  }
  //-----
  return increment
}

//=====
let cntr1 = counter(0)
let cntr2 = counter(0)
document.write(`Cntr1 = ${cntr1()}<br>`)
document.write(`Cntr1 = ${cntr1()}<br>`)
document.write(`Cntr2 = ${cntr2()}<br>`)
document.write(`Cntr1 = ${cntr1()}<br>`)
```



```
function counter(value) {
  function increment() {
    return ++value
  }
  //-----
  return increment
}

//=====
let cntr1 = counter(0)
let cntr2 = counter(0)
document.write(`Cntr1 = ${cntr1()}<br>`)
document.write(`Cntr1 = ${cntr1()}<br>`)
document.write(`Cntr2 = ${cntr2()}<br>`)
document.write(`Cntr1 = ${cntr1()}<br>`)
```

