栈与队列

调用栈：原理与空间

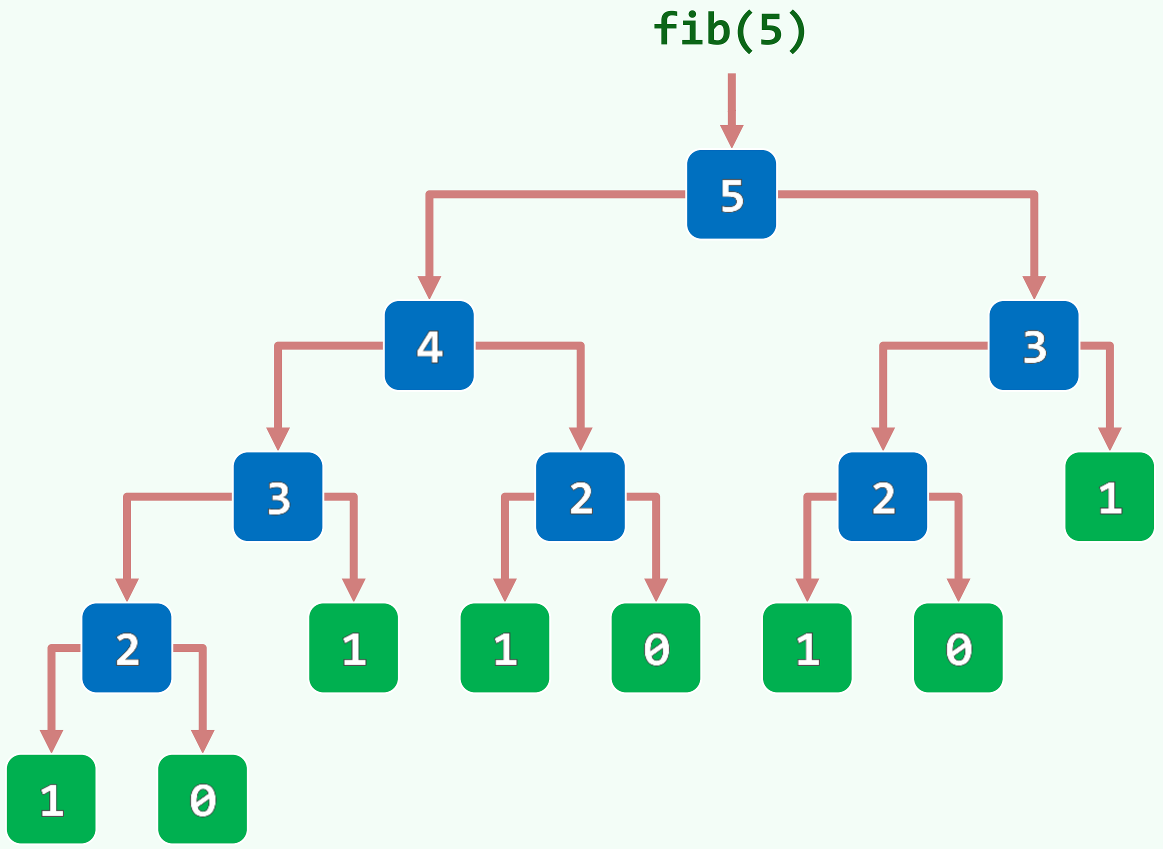
04-B1

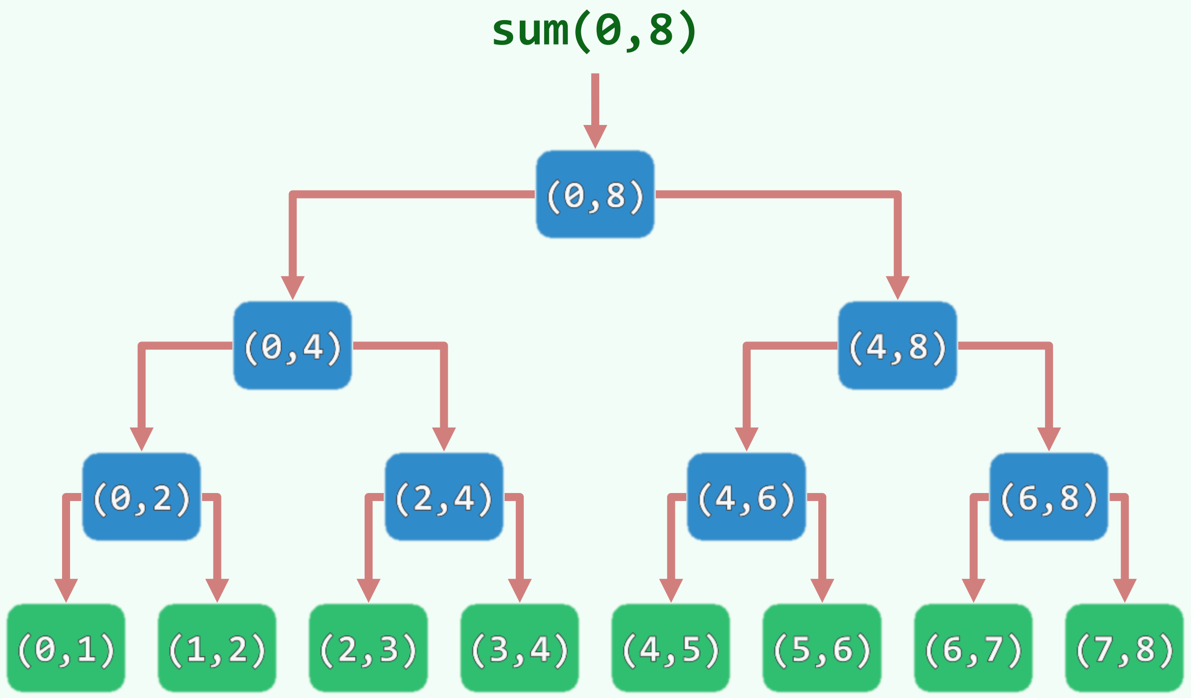
Yessiree. We do not doubt his word, an stack ourselfs into the bus like flapjacks.

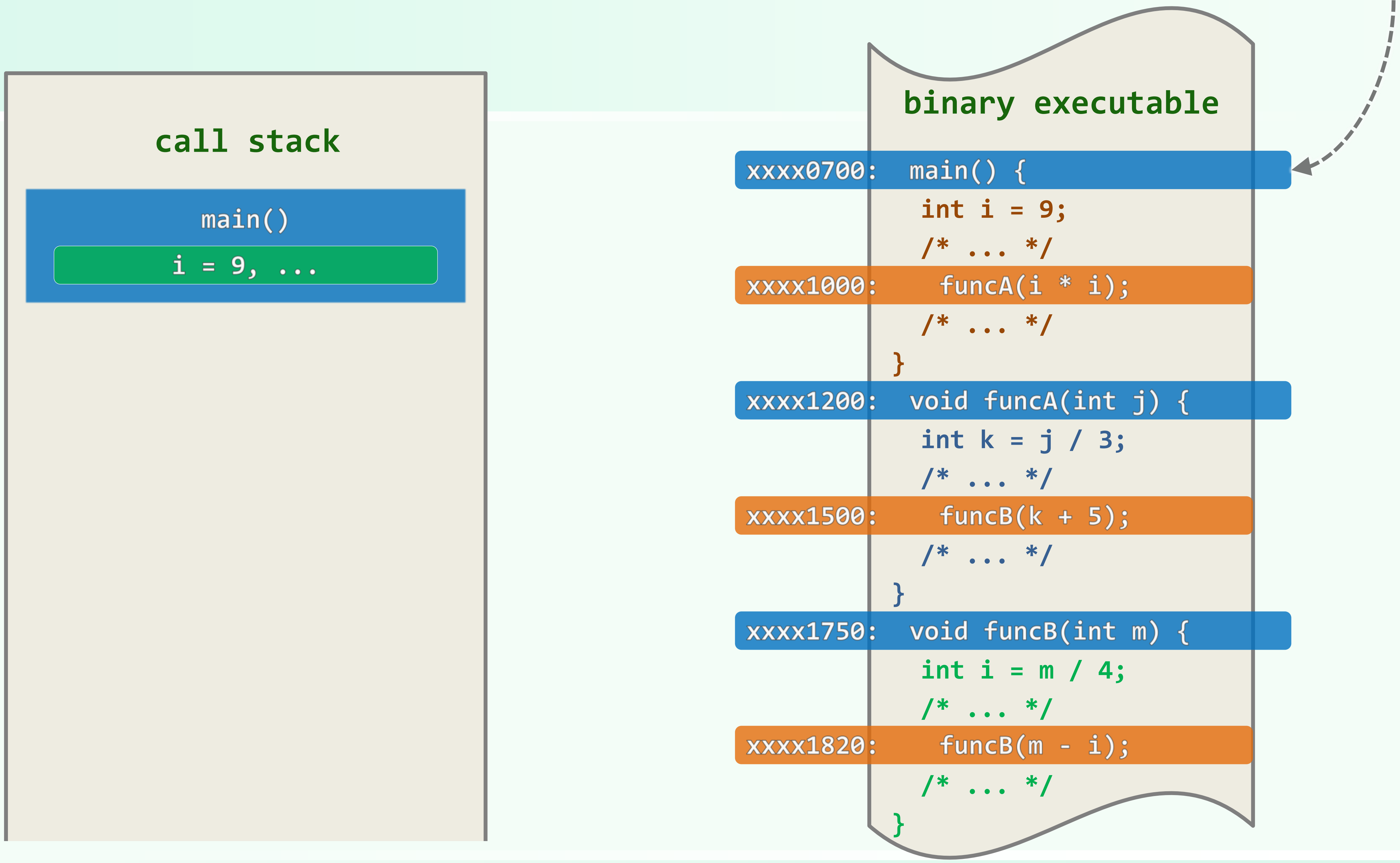
命运把我们的大脑当作一个容器，不停地把各种见解装进去、取出来，但总是现在的和最后的那个见解是可靠没错的

邓俊辉
deng@tsinghua.edu.cn

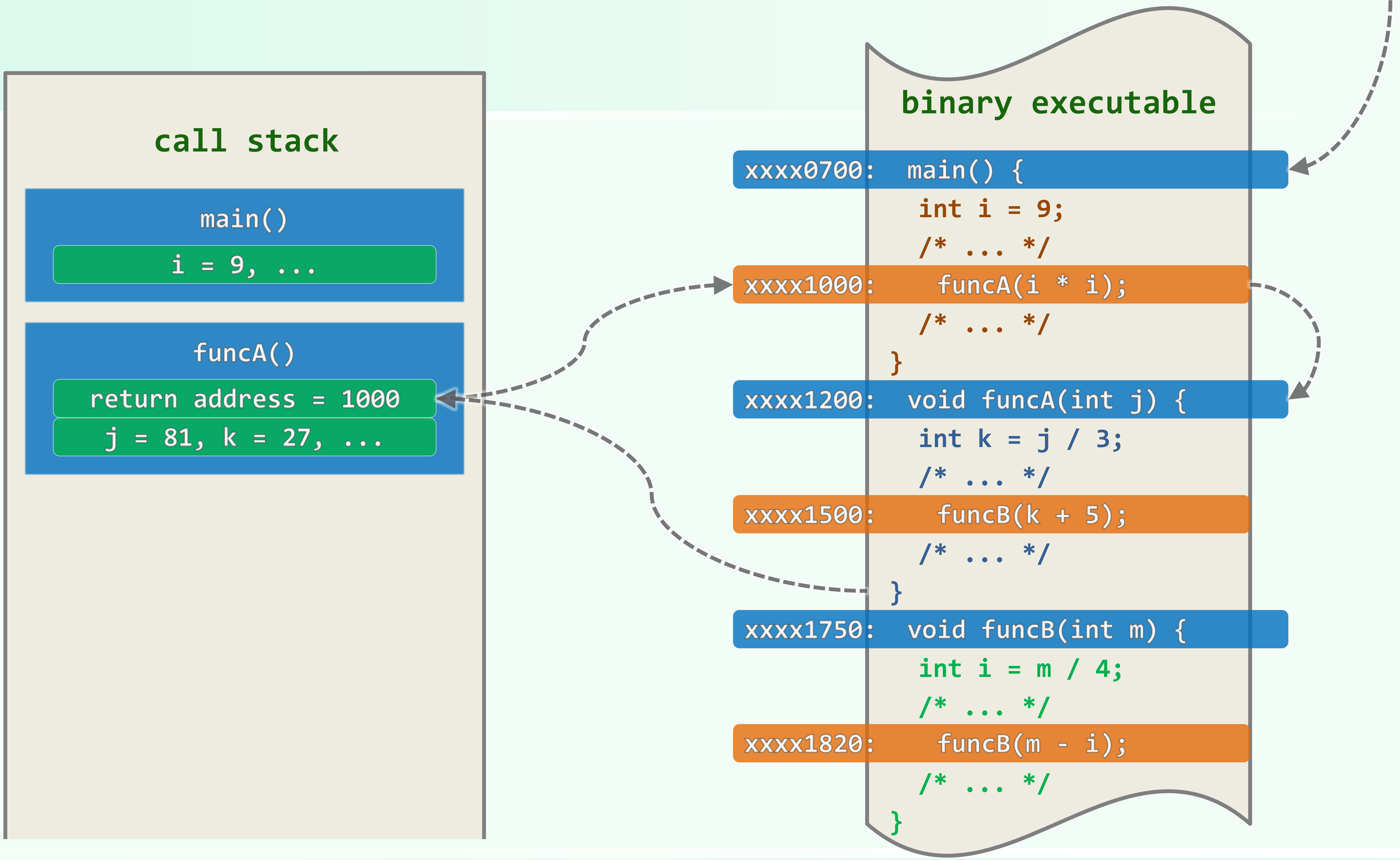
函数调用树：如何实现？Theseus的线团 + 粉笔



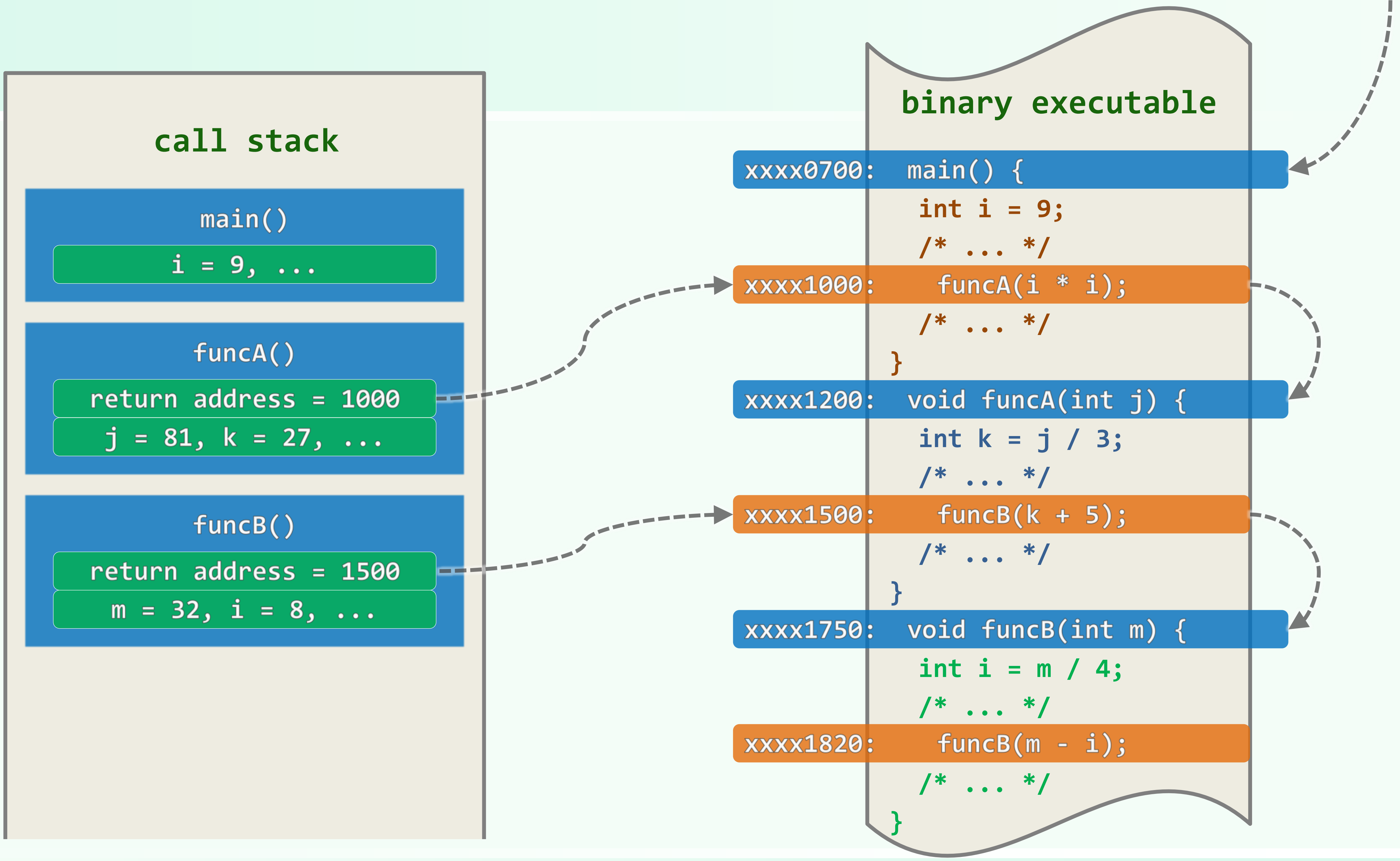




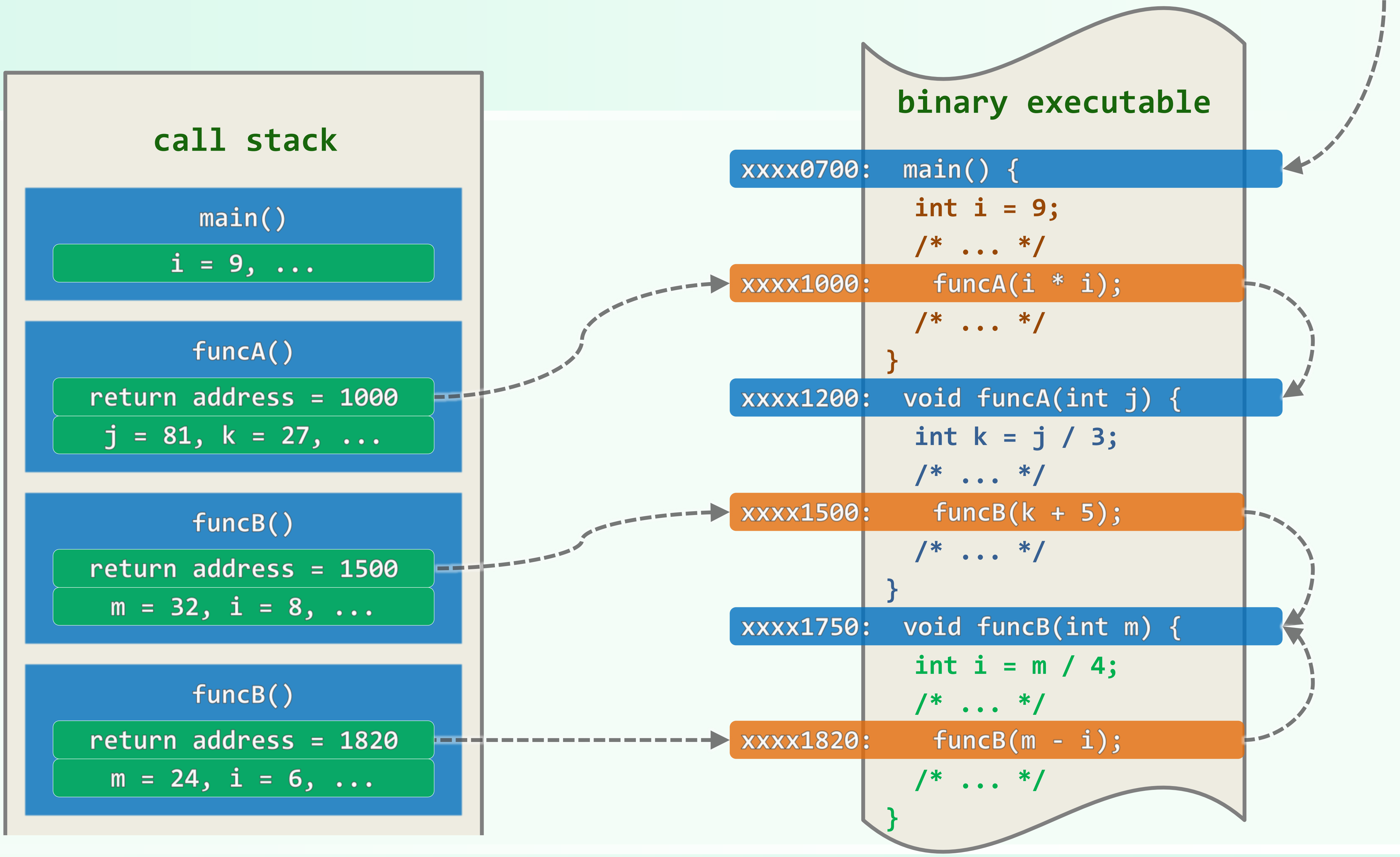
```
xxxx0700:  main() {
            int i = 9;
            /* ... */
xxxx1000:     funcA(i * i);
            /* ... */
           }
xxxx1200:  void funcA(int j) {
            int k = j / 3;
            /* ... */
xxxx1500:     funcB(k + 5);
            /* ... */
           }
xxxx1750:  void funcB(int m) {
            int i = m / 4;
            /* ... */
xxxx1820:     funcB(m - i);
            /* ... */
           }
```



```
xxxx0700:  main() {
             int i = 9;
             /* ... */
xxxx1000:     funcA(i * i);
             /* ... */
           }
xxxx1200:  void funcA(int j) {
             int k = j / 3;
             /* ... */
xxxx1500:     funcB(k + 5);
             /* ... */
           }
xxxx1750:  void funcB(int m) {
             int i = m / 4;
             /* ... */
xxxx1820:     funcB(m - i);
             /* ... */
           }
```

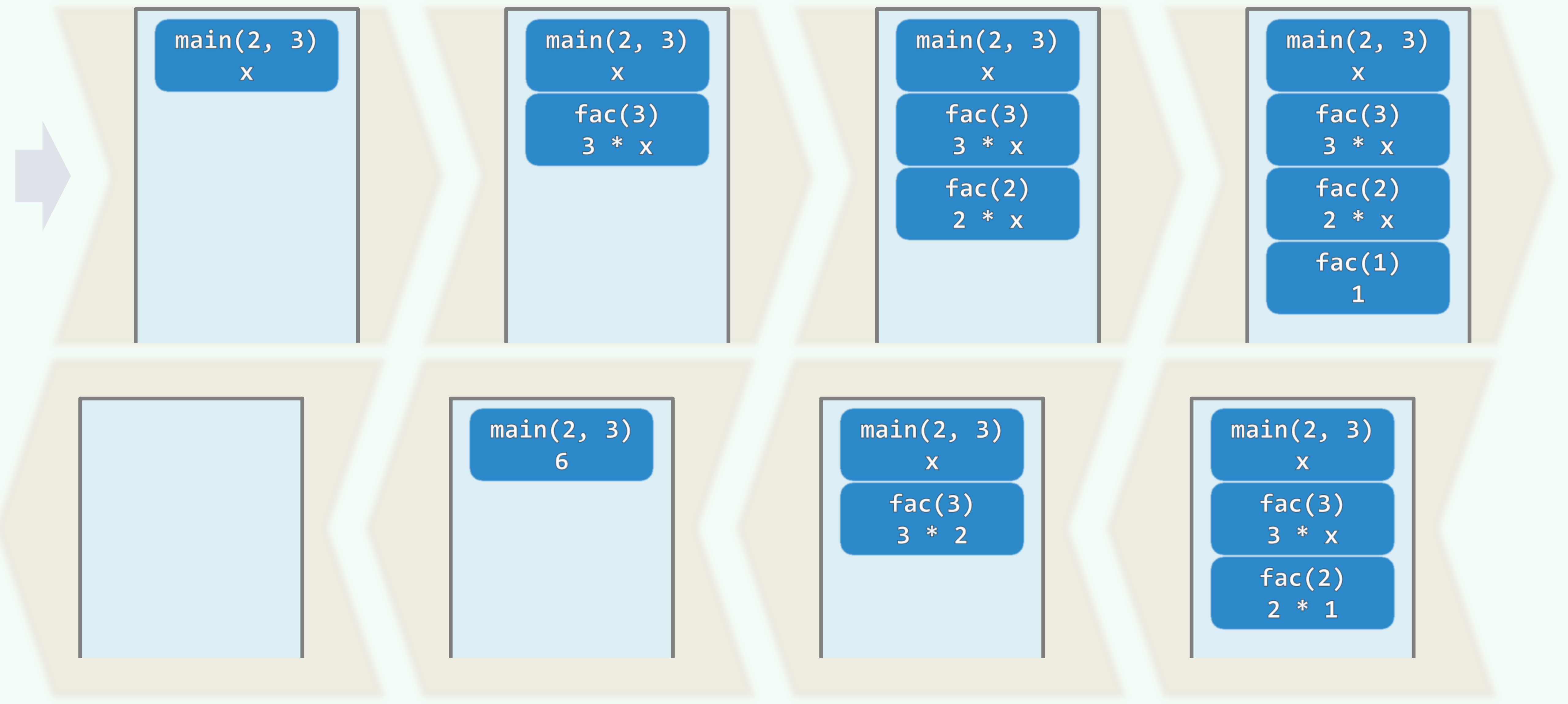


```
xxxx0700:  main() {
             int i = 9;
             /* ... */
xxxx1000:     funcA(i * i);
             /* ... */
           }
xxxx1200:  void funcA(int j) {
             int k = j / 3;
             /* ... */
xxxx1500:     funcB(k + 5);
             /* ... */
           }
xxxx1750:  void funcB(int m) {
             int i = m / 4;
             /* ... */
xxxx1820:     funcB(m - i);
             /* ... */
           }
```

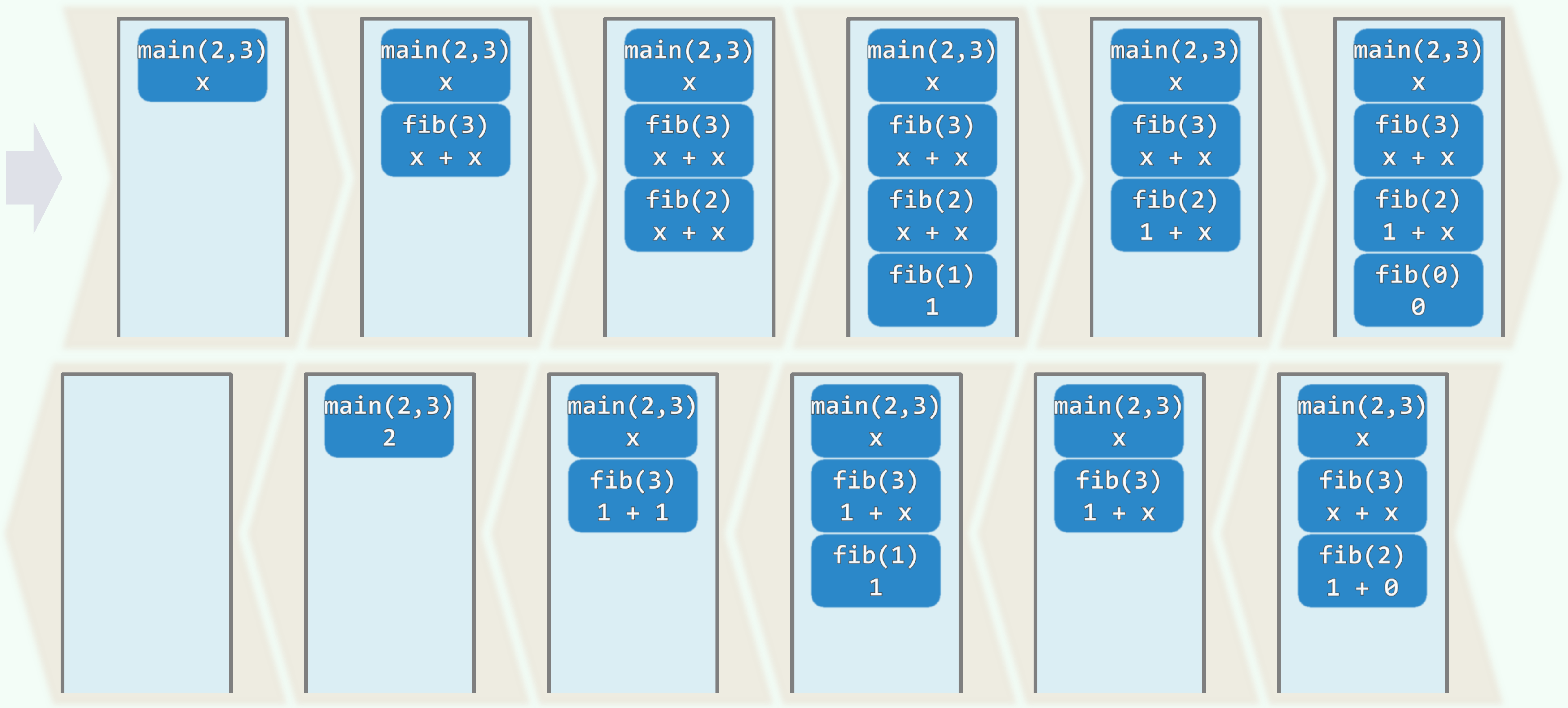


```
xxxx0700:  main() {
             int i = 9;
             /* ... */
xxxx1000:     funcA(i * i);
             /* ... */
           }
xxxx1200:  void funcA(int j) {
             int k = j / 3;
             /* ... */
xxxx1500:     funcB(k + 5);
             /* ... */
           }
xxxx1750:  void funcB(int m) {
             int i = m / 4;
             /* ... */
xxxx1820:     funcB(m - i);
             /* ... */
           }
```

```
int fac(int n) { return (n < 2) ? 1 : n * fac(n - 1); }
```



```
int fib( int n ) { return (n < 2) ? n : fib(n - 1) + fib(n - 2); }
```



空间复杂度

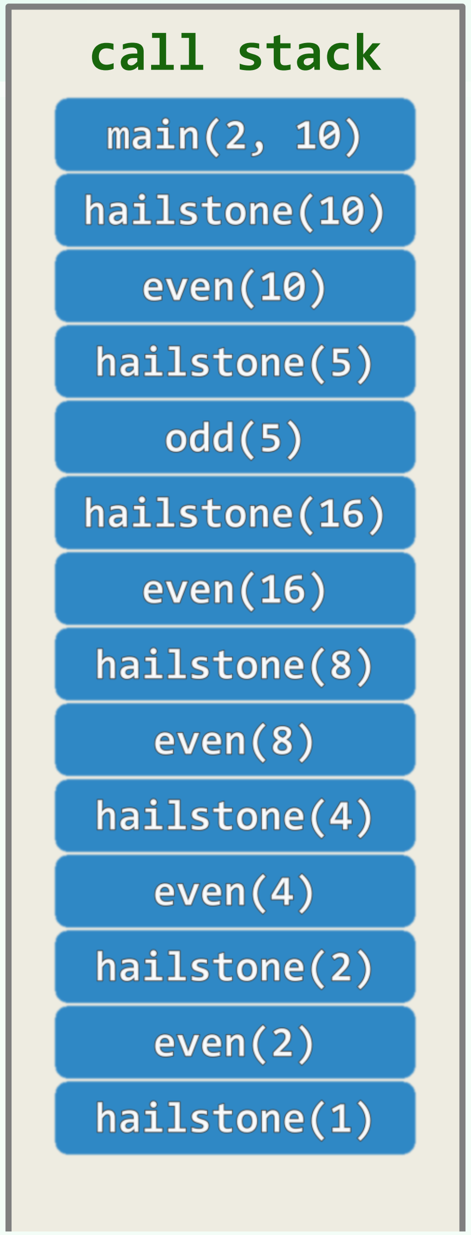
❖
```
hailstone(int n) {
   if ( 1 < n )
      n % 2  ?  odd( n ) : even( n );
}
```

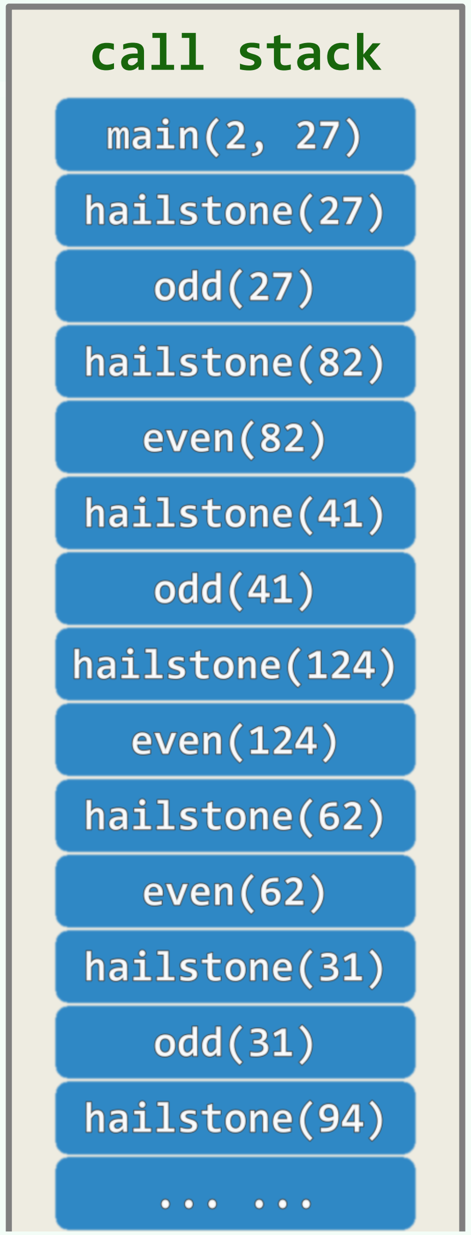
❖
```
even( int n ) { hailstone( n / 2 ); }
odd( int n ) { hailstone( 3*n + 1 ); }
```

❖
```
main( int argc, char* argv[] )
{ hailstone( atoi( argv[1] ) ); }
```

❖ 可见，递归算法所需的**空间**

主要取决于递归**深度**，而非递归实例总数


call stack
main(2, 10)
hailstone(10)
even(10)
hailstone(5)
odd(5)
hailstone(16)
even(16)
hailstone(8)
even(8)
hailstone(4)
even(4)
hailstone(2)
even(2)
hailstone(1)


call stack
main(2, 27)
hailstone(27)
odd(27)
hailstone(82)
even(82)
hailstone(41)
odd(41)
hailstone(124)
even(124)
hailstone(62)
even(62)
hailstone(31)
odd(31)
hailstone(94)
... ...