Lab 3

Arrays, Function Overloading, & Recursion

Presentation by *Asem Alaa*

Functions overloading

Functions overloading

C++ allows *functions with same name*, **but different parameter types**.

Functions overloading

C++ allows *functions with same name*, **but different parameter types**.

For example, this is **not allowed** in C++:

```
double area( double w , double h )
{
    return w * h;
}

double area( double base , double height )
{ // Compiler Error, redefinition of area(double,double)
    return base * height / 2;
}
```

AMBIGUOUS when calling `area(1.4,5)`

Functions overloading

This works

Functions overloading

This works

```
double area( double d )
{
    return d * d; // square area
}

double area( double w, double h)
{
    return w * h;
}
```

Functions overloading

```
struct Rectangle
{
  double w;
  double h;
};
struct RTriangle
{
  double b;
  double h;
};
double area( Rectangle rect ) // Works!
{
    return rect.w * rect.h;
}
double area( RTriangle tr ) // Works!
{
    return tr.b * tr.h / 2;
}
```

Functions overloading

```
double area( double d ); // square
double area( Rectangle rect );
double area( RTriangle tr );

int main()
{
  double s;
  std::cin >> s;
  std::cout << "square area="<< area(s) <<"\n";
  Rectangle r;
  std::cin >> r.w >> r.h;
  std::cout << "rectangle area=" << area(r) << "\n";
  RTriangle tr;
  std::cin >> tr.b >> tr.h;
  std::cout << "triangle area=" << area(tr) << "\n";
}
```

Recursion

Factorial example

```
factorial( n ):
if n == 1:
    return 1
else:
    return n * factorial(n-1):
```

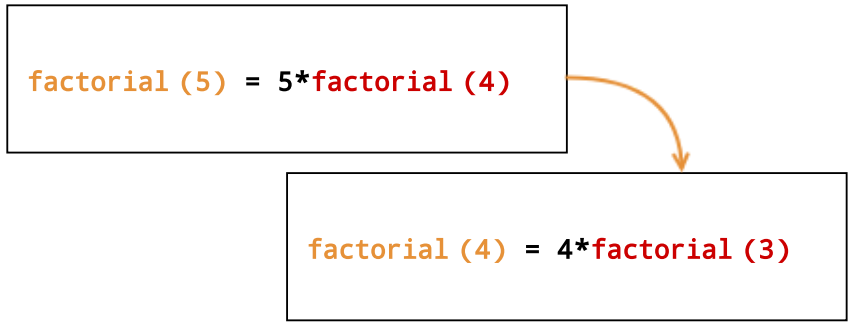
```
if n == 1:
    return 1
else:
```

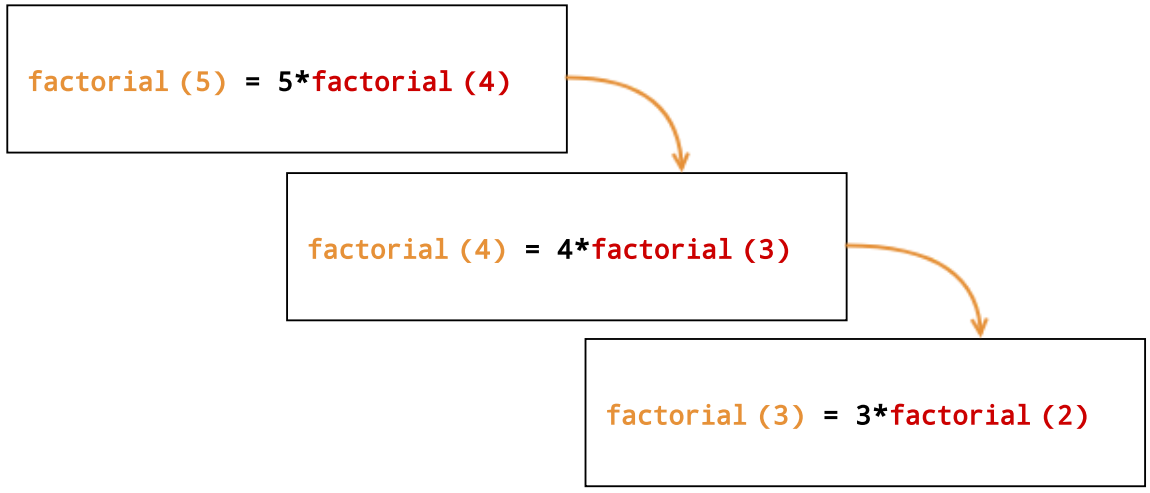
factorial(n) =

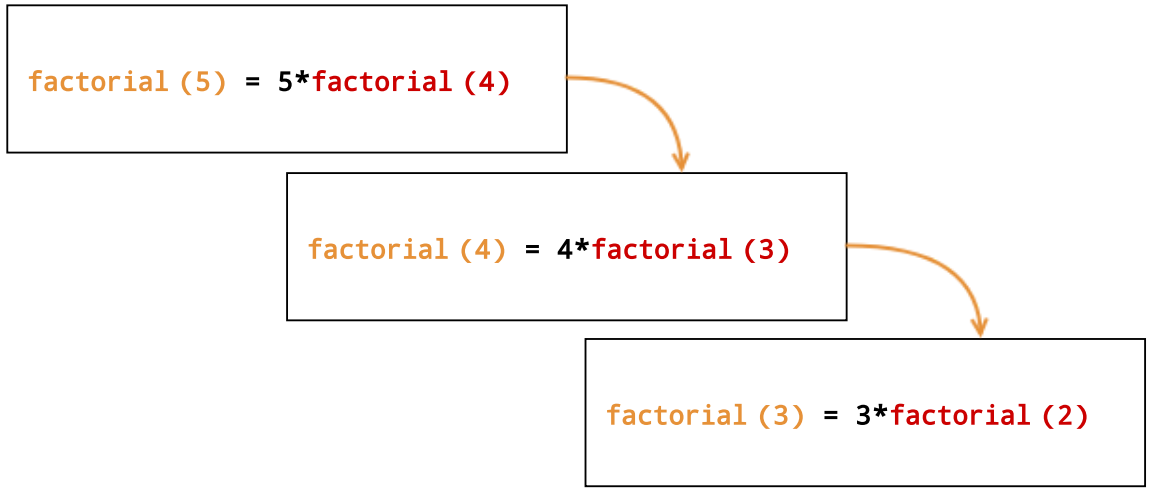
www.mathwarehouse.com

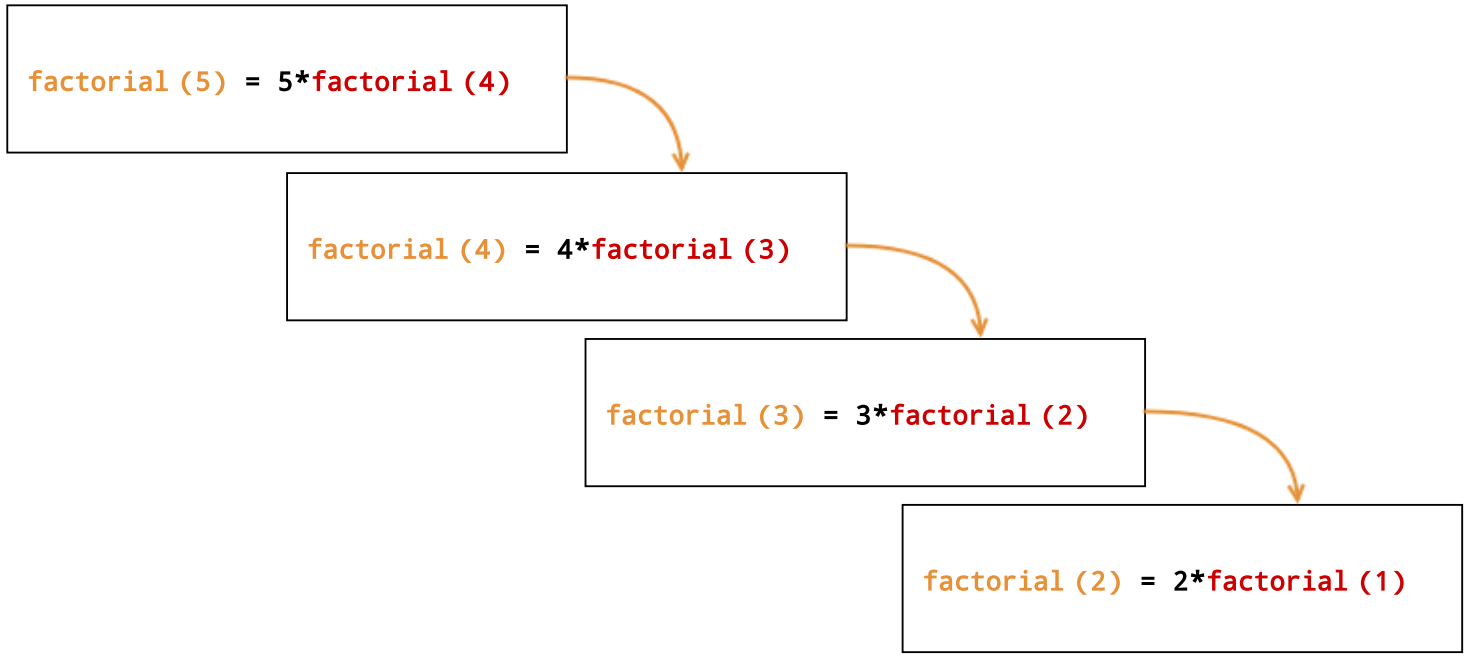
```cpp
#include <iostream>
int factorial( int n )
{
    if( n <= 1 )
        return 1;
    else
        return n * factorial( n - 1 );
}

int main()
{
    std::cout << "5!=" << factorial( 5 );
    return 0;
}
```

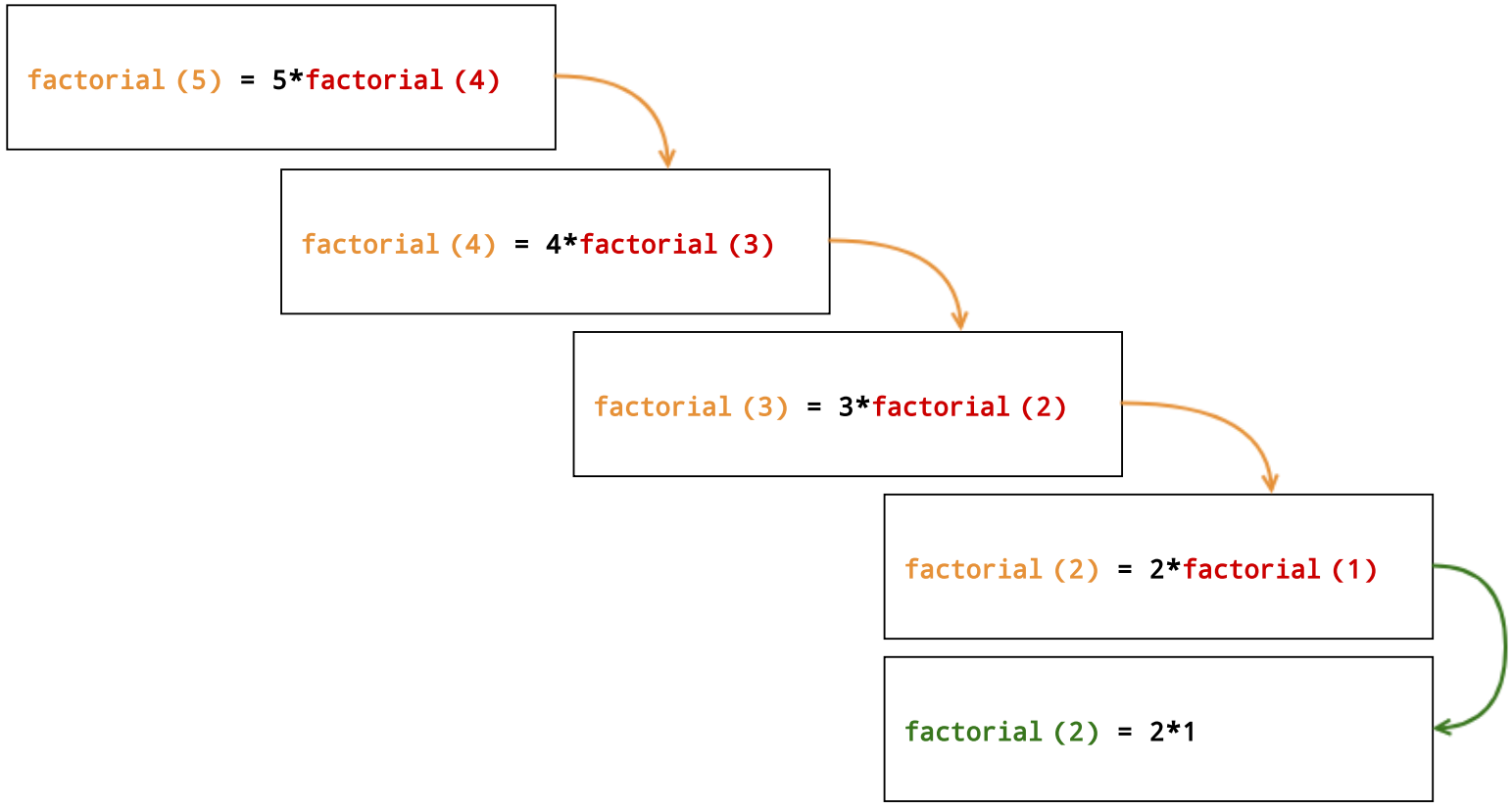
```
factorial(5) = 5*factorial(4)
```

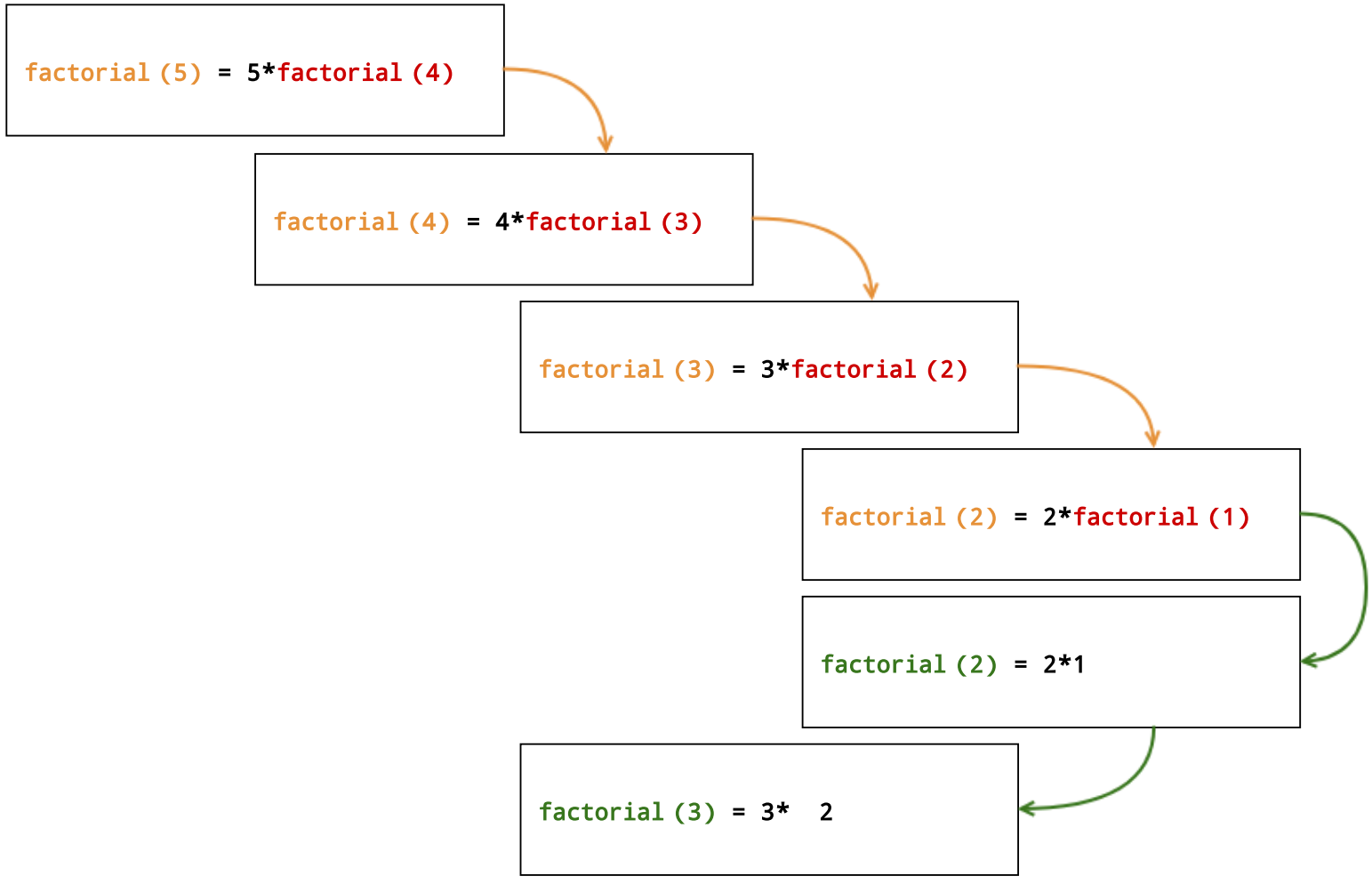

factorial (5) = 5*factorial (4)
factorial (4) = 4*factorial (3)

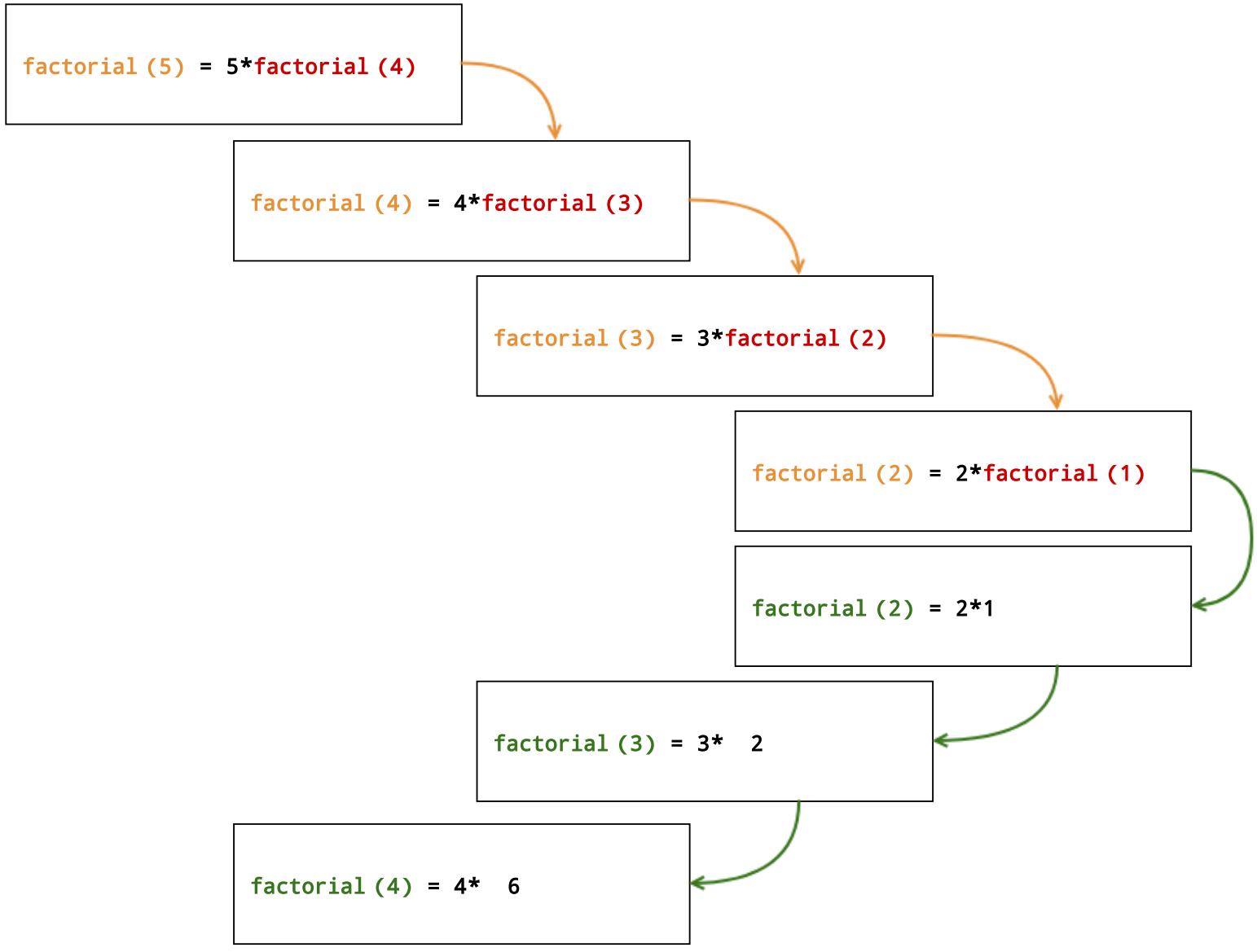

factorial (5) = 5*factorial (4)
factorial (4) = 4*factorial (3)
factorial (3) = 3*factorial (2)

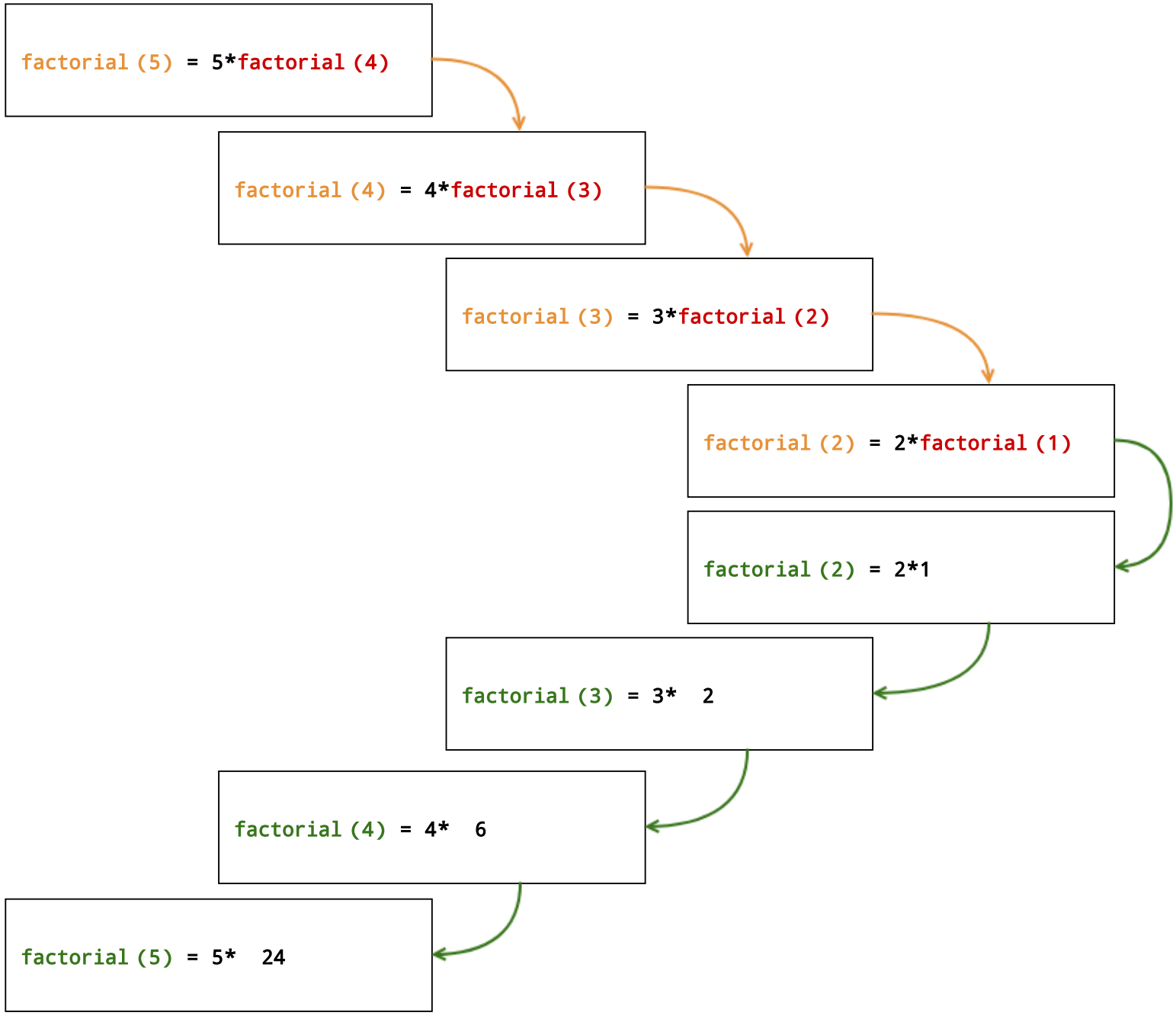

factorial (5) = 5*factorial (4)
factorial (4) = 4*factorial (3)
factorial (3) = 3*factorial (2)


factorial (5) = 5*factorial (4)
factorial (4) = 4*factorial (3)
factorial (3) = 3*factorial (2)
factorial (2) = 2*factorial (1)


factorial (5) = 5*factorial (4)
factorial (4) = 4*factorial (3)
factorial (3) = 3*factorial (2)
factorial (2) = 2*factorial (1)
factorial (2) = 2*1


factorial (5) = 5*factorial (4)
factorial (4) = 4*factorial (3)
factorial (3) = 3*factorial (2)
factorial (2) = 2*factorial (1)
factorial (2) = 2*1
factorial (3) = 3* 2


factorial (5) = 5*factorial (4)
factorial (4) = 4*factorial (3)
factorial (3) = 3*factorial (2)
factorial (2) = 2*factorial (1)
factorial (2) = 2*1
factorial (3) = 3* 2
factorial (4) = 4* 6


factorial (5) = 5*factorial (4)
factorial (4) = 4*factorial (3)
factorial (3) = 3*factorial (2)
factorial (2) = 2*factorial (1)
factorial (2) = 2*1
factorial (3) = 3* 2
factorial (4) = 4* 6
factorial (5) = 5* 24

How recursion work in stack memory

{Demo: How Recursive Factorial Work in Memory}

Recursion is not Function Overloading

Recursion is not Function Overloading

The following is not recursion

```
struct Rectangle
{
    double a = 0;
    double b = 0;
};

double area( double a , double b )
{
    return a * b;
}

double area( Rectangle rect )
{
    return area( rect.a , rect.b ); // This is not recursion.
}
```

However, the following calling `area` is recursive, it calls itself

```
struct Rectangle
{
    double a = 0;
    double b = 0;
};

double area( Rectangle rect )
{
    return area( rect ); // This is a recursion. But a buggy function!
}
```

- infinite recurions,
- until stack memory overflow happens,
- and finally the program crashes.

Exercise: Power Function

Exercise: Power Function

Implement a function `power` that uses recursion to compute the power of the input number.