Recursion

Instructor: Chi Tse (Ricky)

Slides adopted from professor Kent Chin and professor Jean Lai

What is Recursion?

- **Recursion**:
  A function calling itself to solve a simpler form of the same problem.
- **Recursive Function**:

  Base Case – terminate case
  Iterative Step – What the function should do in each step

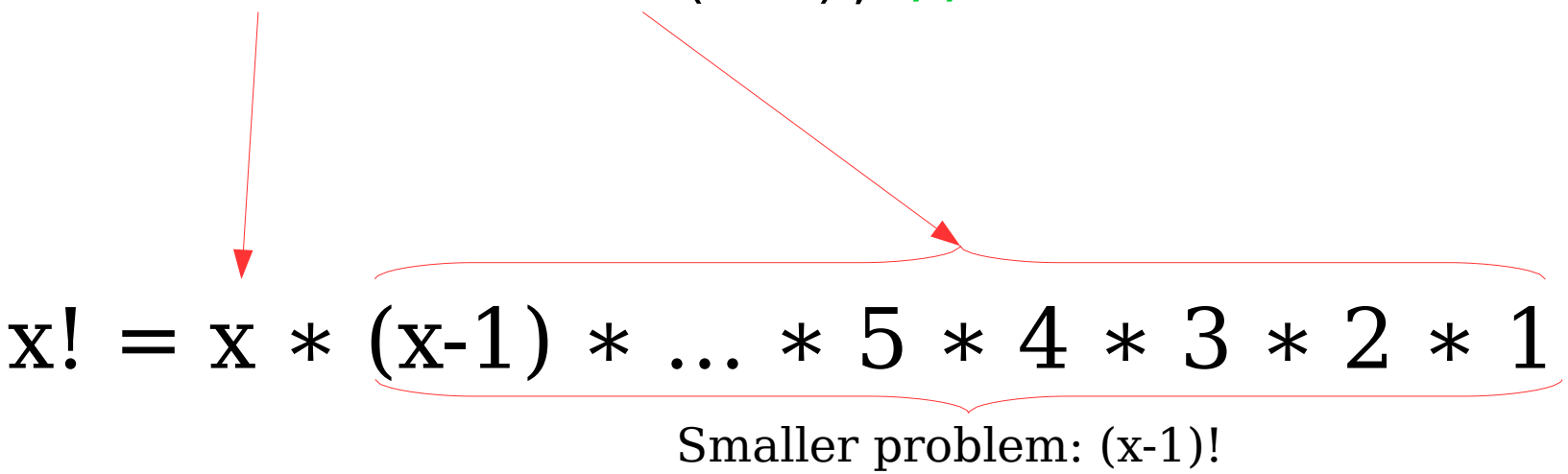
Example 1: Factorial

Recursive algorithms work as follows:

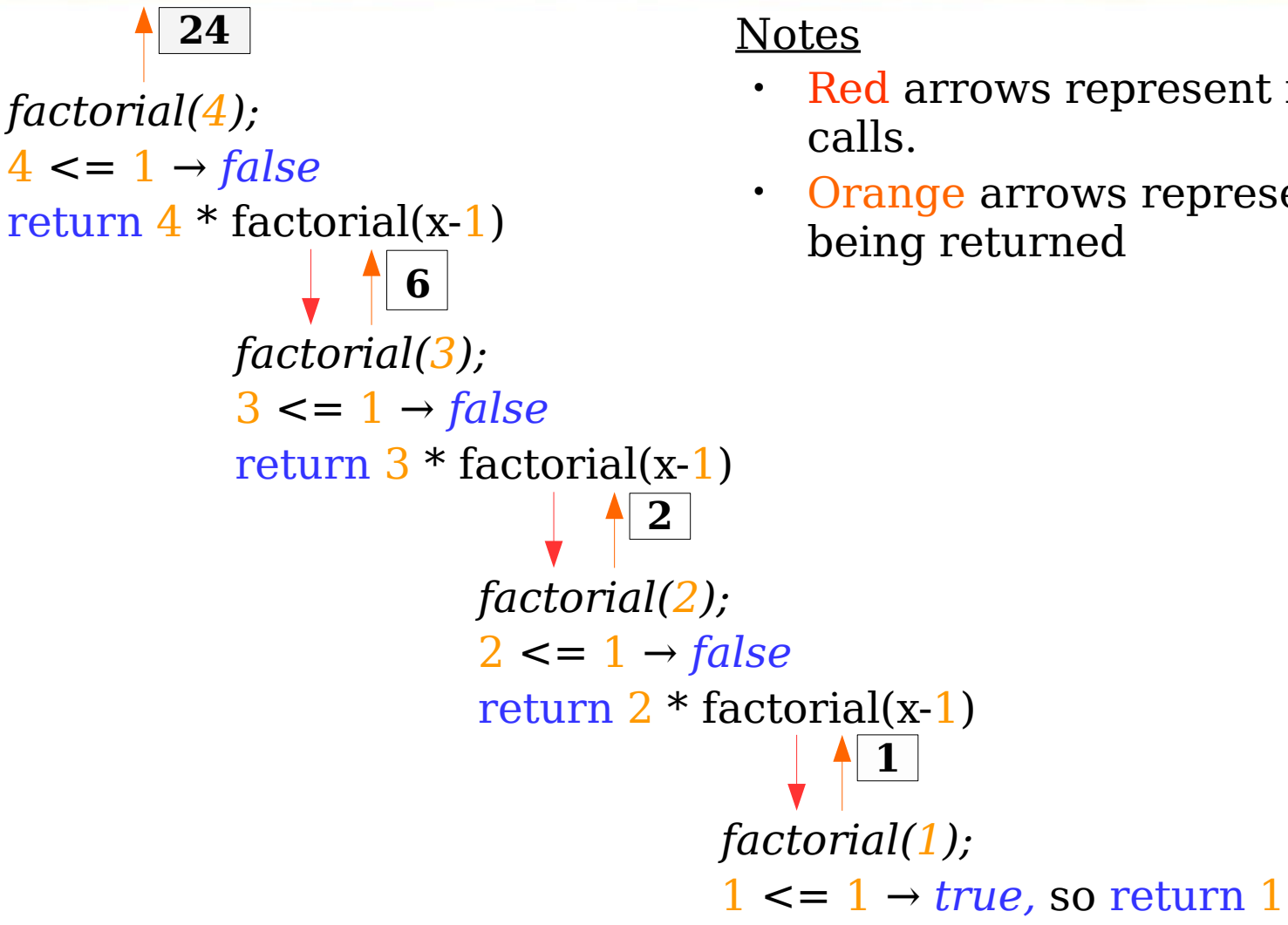
When your problem is small and easy (i.e. cannot be broken down), just return the answer! We call this the **base case** (it's possible to have more than one base case).

If your problem is too difficult, solve a smaller problem! This is where we make one or more *recursive* calls.

```
int factorial(int x) {
  if (x <= 1) return 1; //base case, 0! = 1! = 1
  return x * factorial(x-1); //recursive call
```



Trace



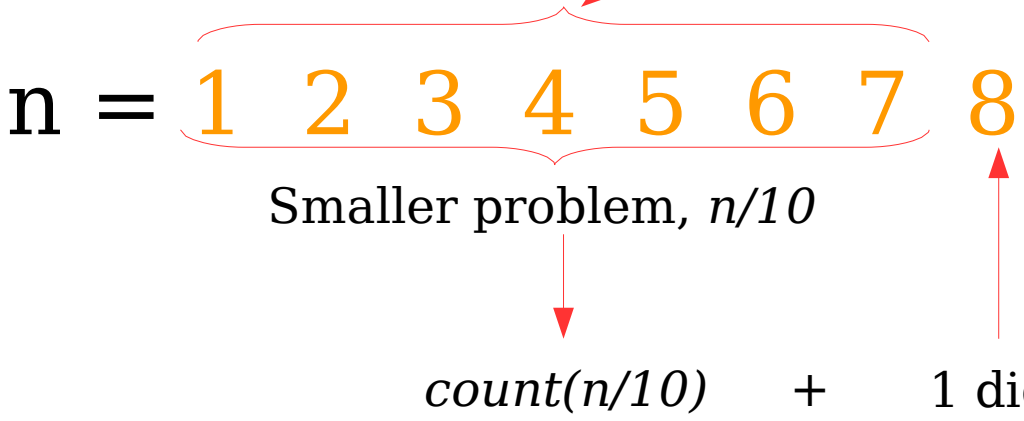
Notes

- Red arrows represent function calls.
- Orange arrows represent values being returned

Example 2: Counting Digits

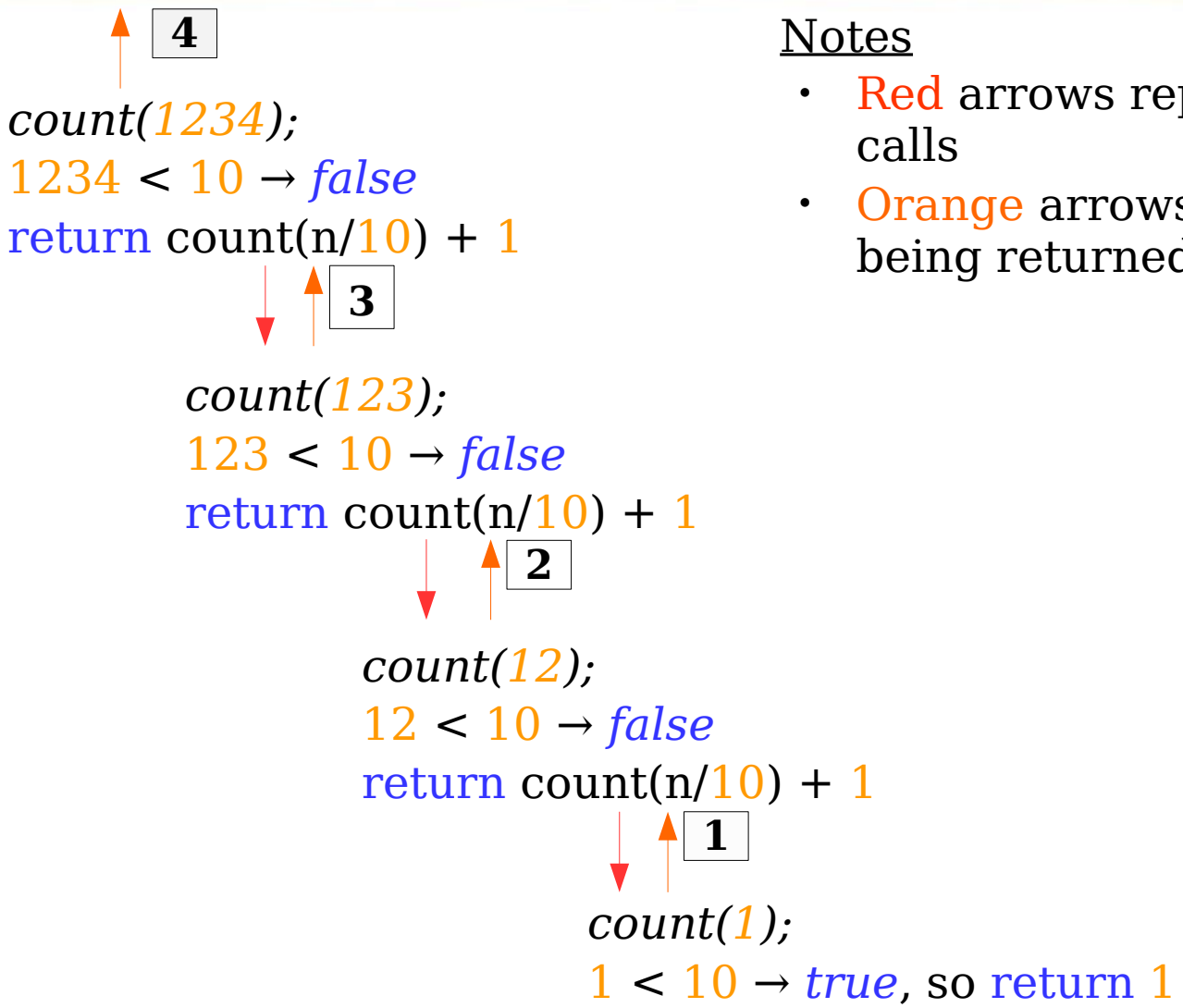
Function Prototype: `int count(int n);`
Base Case: A single digit number has 1 digit, so if *n* < *10*, return 1
Recursive Case: ***1 + # of digits in the rest of n***

```
int count(int n) {
  if (n < 10) return 1;
  return count(n/10) + 1;
}
```



12345678 is just a dummy number being used for illustration.

Trace



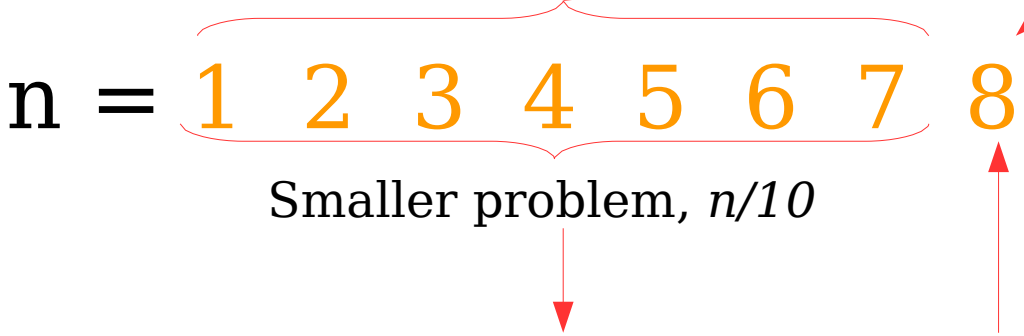
Notes

- Red arrows represent function calls
- Orange arrows represent values being returned.

Example 3: Summing Digits

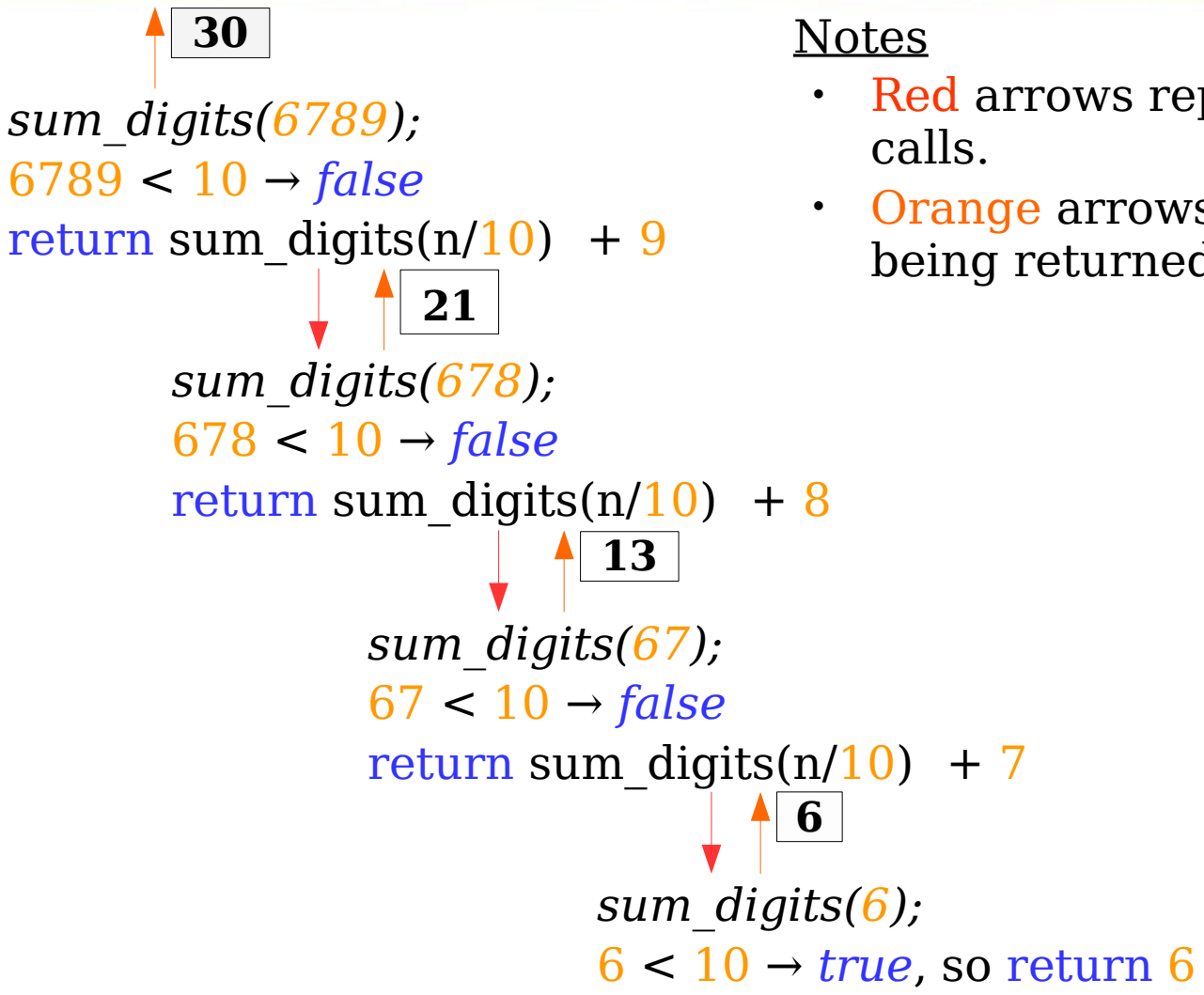
Function Prototype: `int sum_digits(int n);`
Base Case: If single digit's sum is itself, so if *n* < *10*, return *n*.
Recursive Case: ***last digit*** + ***sum of rest of the digits of n***

```
int sum_digits(int n) {
  if (n < 10) return n;
  return sum_digits(n/10) + (n%10);
```



12345678 is just a dummy number being used for illustration.

Trace



Notes

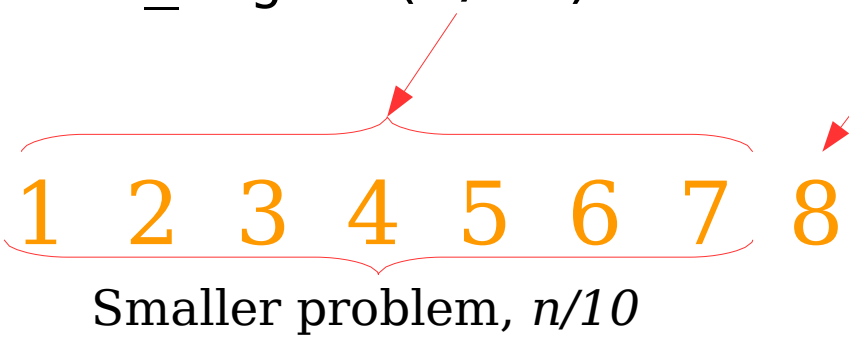
- Red arrows represent function calls.
- Orange arrows represent values being returned

Example 4: Doubling All Digits

```
cout << double_digits(1) << endl;   //11
cout << double_digits(21) << endl;  //2211
cout << double_digits(101) << endl; //110011
```

Function Prototype: `int double_digits(int n);`
Base Case: If *n < 10*, return *n*11* (this "doubles" a single digit).
Recursive Case: ***the last digit doubled*** + (***the rest of the digits doubled)*100*** (to make "room" for the doubled last digit)

```
int double_digits(int n)
  if (n < 10) return n*11;
  return double_digits(n/10)*100 + (n%10)*11;
```



12345678 is just a dummy number being used for illustration.

Example 5: Summing Odd Digits, only

```
cout << sum_odd(123) << endl; //4
cout << sum_odd(211) << endl; //2
```

Function Prototype: `int sum_odd(int n);`

Base Case: If *n == 0*, the sum is 0, so return *0.*

Recursive Cases:

- If last digit is odd, add the last digit with the sum of the *rest of the odd digits of n (recursive call!).*
- Otherwise, just return the sum of the *rest of the odd digits of n (recursive call!).*

```
int sum_odd(int n) {
  if (n == 0) return 0;
  if (n % 2 == 1) //same as (n%10) % 2 == 1
    return sum_odd(n/10) + (n%10);
  else
    return sum_odd(n/10);
}
```

Example 6: Fibonacci

```
int Fibonacci (int n)
{
        if (n <= 1)
                return n;
        else
                return Fibonacci(n-1) + Fibonacci(n-2);
}
```

Base case
No recursion calls!

Smaller problem
with smaller input size.

Example 6: Fibonacci