04-B2

栈与队列

调用栈：消除递归

无垂不缩，无往不收

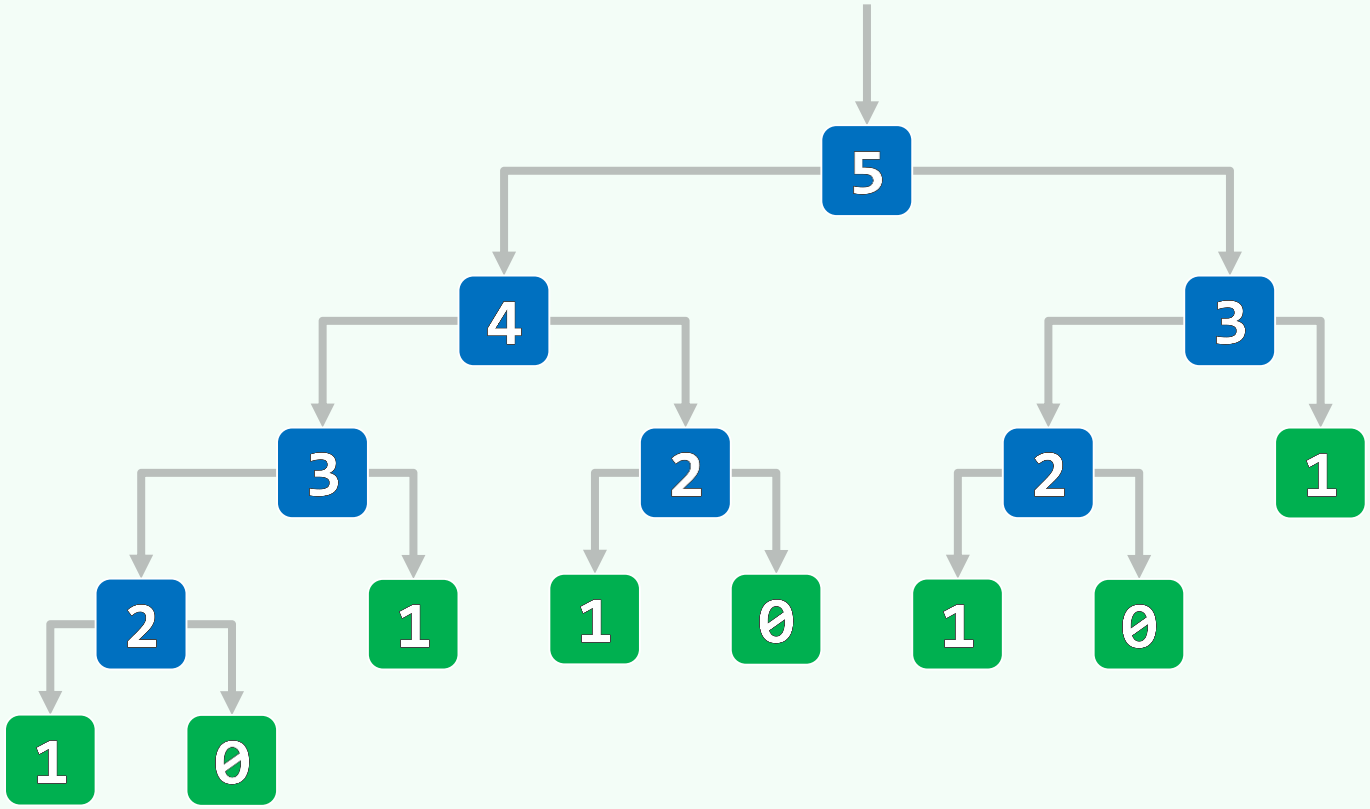
《星期评论》问我“女子解放从那里做起？”
我的答案是：“女子解放当从女子解放做起。此外更无别法。”

邓俊辉
deng@tsinghua.edu.cn

消除递归：动机 + 方法

- 递归函数的空间复杂度
  - 主要取决于最大递归深度
  - 而非递归实例总数
- 为隐式地维护调用栈

  需花费额外的时间、空间
- 为节省空间，可
  - 显式地维护调用栈
  - 将递归算法改写为迭代版本...



消除递归：实例

- 通常，消除递归只是
  在常数意义上优化空间
- 但也可能有实质改进
  ...
-
```
int fac( int n ) {
   int f = 1; //O(1)空间
   while ( n > 1 )
      f *= n--;
   return f;
}
```

-
```
void hailstone( int n ) { //O(1)空间
   while ( 1 < n )
      n = n % 2 ? 3*n + 1 : n/2;
}
```

-
```
int fib( int n ) { //O(1)空间
   int f = 0, g = 1;
   while ( 0 < n-- )
      { g += f; f = g - f; }
   return f;
}
```

尾递归：定义 + 实例

- 递归发生在函数末尾

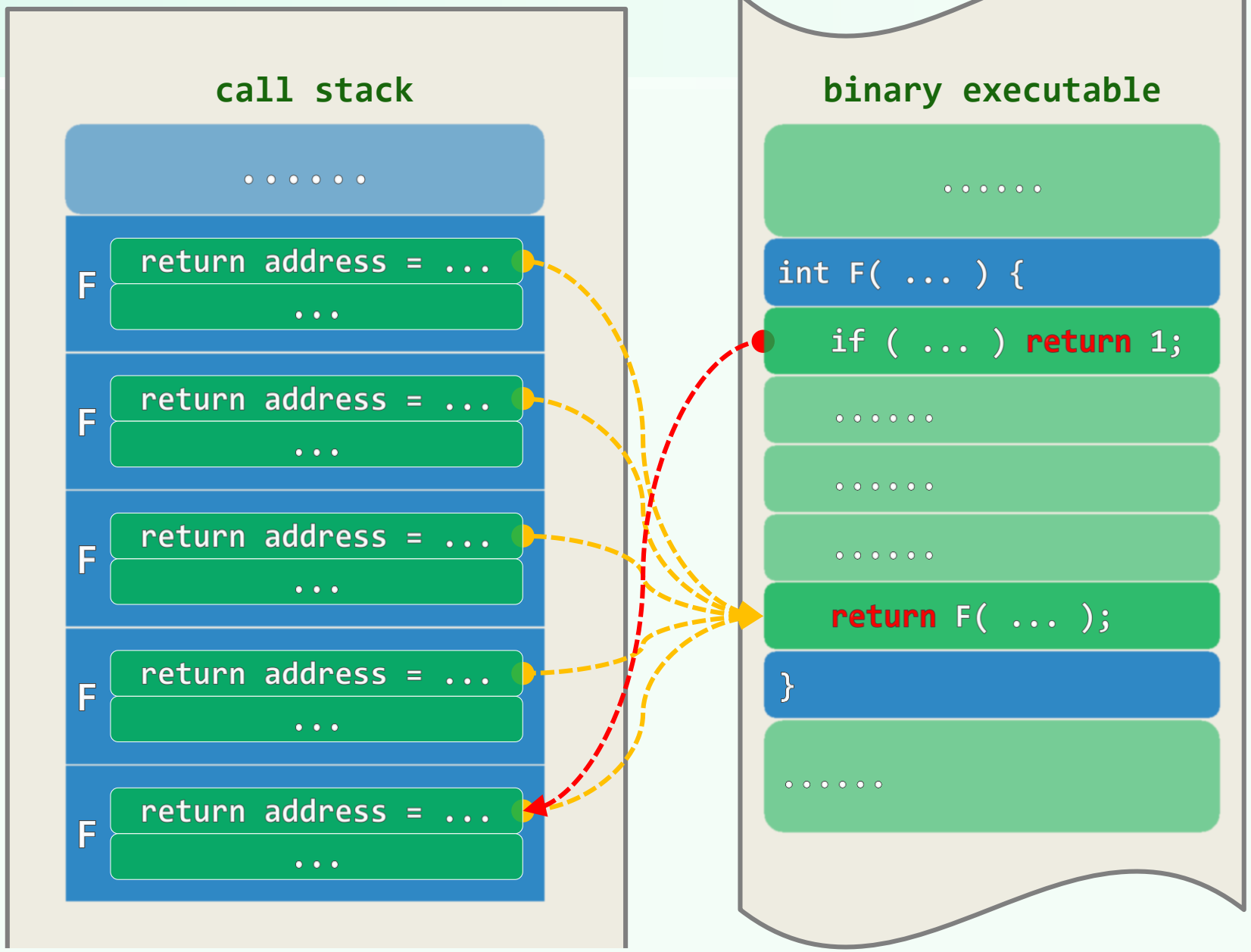
  就在return之前

- 比如，此前学过的...

```
void reverse( int * A, int n ) {
    if ( n < 2 ) return;
    swap( A[0], A[n - 1] );
    reverse( A + 1, n - 2 );
}
```

尾递归：性质

- 系最简单的递归模式
- 一旦抵达递归基，便会
  - 引发一连串的return
    （且返回地址相同）
  - 调用栈相应地连续pop
- 故不难改写为迭代形式
- 越来越多的编译器可以
  自动识别并代为改写
- 时间复杂度有常系数改进
  空间复杂度或有渐近改进



尾递归：消除

```
//尾递归
reverse(int* A, int n) {

   if (n < 2) return;
   swap(A[0], A[n-1]);
   A++; n -= 2;
   reverse(A, n);
} //O(n)时间 + O(n)空间
```

```
//统一转换为迭代
reverse(int* A, int n) {
next: //转向标志
   if (n < 2) return;
   swap(A[0], A[n-1]);
   A++; n -= 2;
   goto next; //模拟递归返回
} //O(n)时间 + O(1)空间
```

```
//进一步规整化
reverse(int* A, int n) {

   while (1 < n) {
      swap(A[0], A[n-1]);
      A++; n -= 2;
   }
} //O(n)时间 + O(1)空间
```