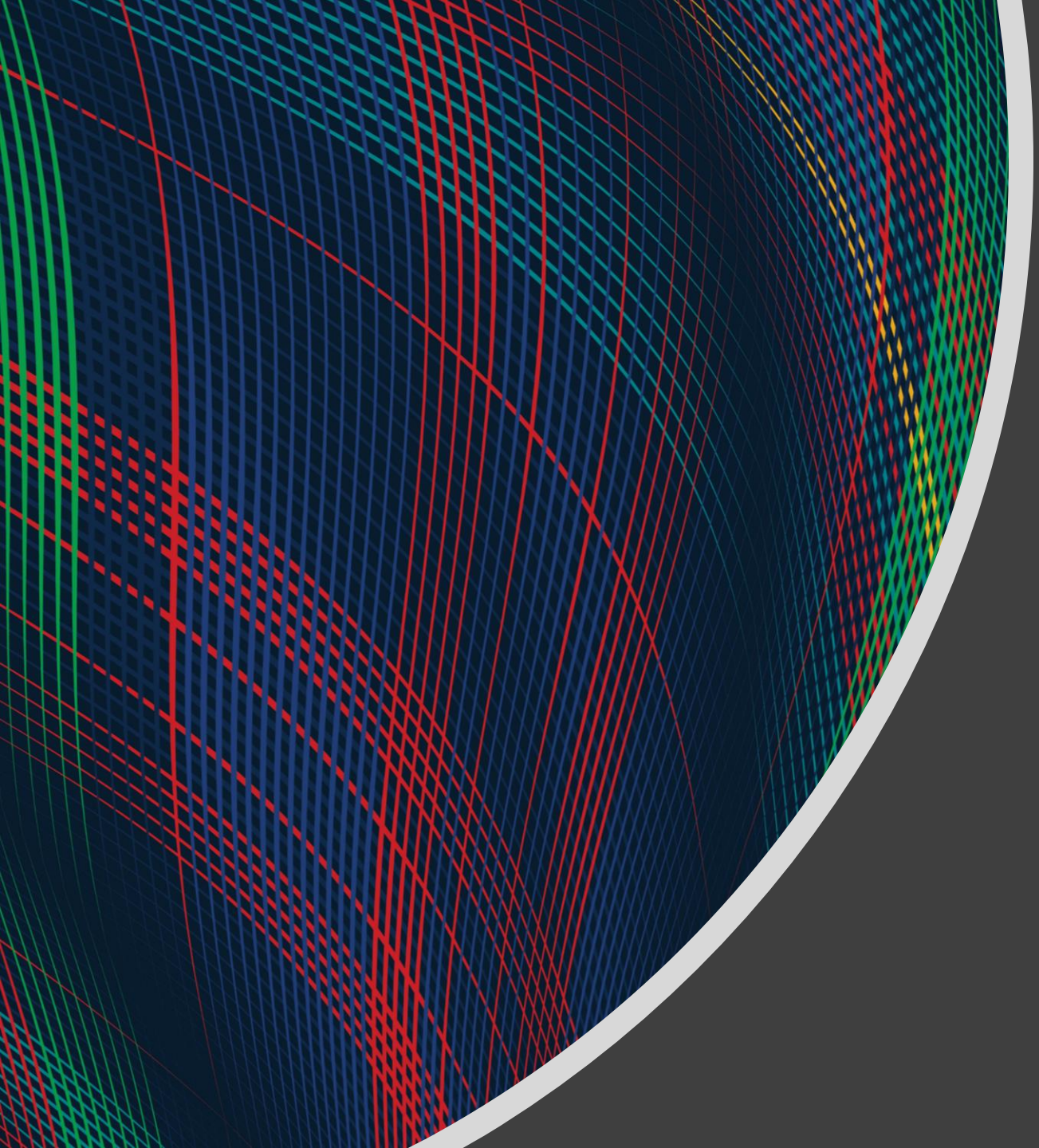


10-607 Computational Foundations for Machine Learning

Recursion, Induction, & Dynamic Programming

Instructor: Pat Virtue

Plan

Recursion

- Recursion
- Practical considerations
- Proof by induction

Divide and Conquer

- Sorting example

Dynamic Programming

- Overlapping subproblems
- Examples

Factorial

Complete the recursive function for factorial!

Note: A recursive function calls itself

```
def factorial(int n):
    if _____________:
        return _____________
    return ____________________
```

0! = 1

1! = 1

2! = 2

3! = 6

4! = 24

Recursion

https://www.cs.cmu.edu/~112/notes/notes-recursion-part1.html

Template

```
def recursiveFunction():
    if (this is the base case):
        do something non-recursive
    else:
        do something recursive
```

Recursion Examples

List sum

```
# Problem: sum all of the numbers in a given list
def listSum(L):
    # Base Case: the list is empty, so the sum is 0
    if (len(L) == 0):
        return 0
    else:
        # Recursive Case: assume we already know the sum of the entire list
        # after the first element. Add that sum to the first element.
        return L[0] + listSum(L[1:])

print(listSum([2,3,5,7,11])) # 28
```

Slide credit: CMU 15-112

Recursion Examples

Power

```
# Problem: raise the number base to the given exponent
def power(base, expt):
    # assume expt is non-negative integer
    if (expt == 0):
        return 1
    else:
        return base * power(base, expt-1)

print(power(2,5)) # 32
```

2^{10}

$2 \cdot 2^9$

Recursion Examples

Power – Multiple base cases!

```
def power(base, expt):
    # This version allows for negative exponents
    # It still assumes that expt is an integer, however.
    if (expt == 0):
        return 1
    elif (expt < 0): # new recursive case!
        return 1.0 / power(base, abs(expt))
    else:
        return base * power(base, expt-1)


print(power(2,5)) # 32
print(power(2,-5)) # 1/32 = 0.03125
```

Plan

Recursion

- Recursion
- Practical considerations
- Proof by induction

Divide and Conquer

- Sorting example

Dynamic Programming

- Overlapping subproblems
- Examples

Iterative vs Recursive

Factorial

Iterative

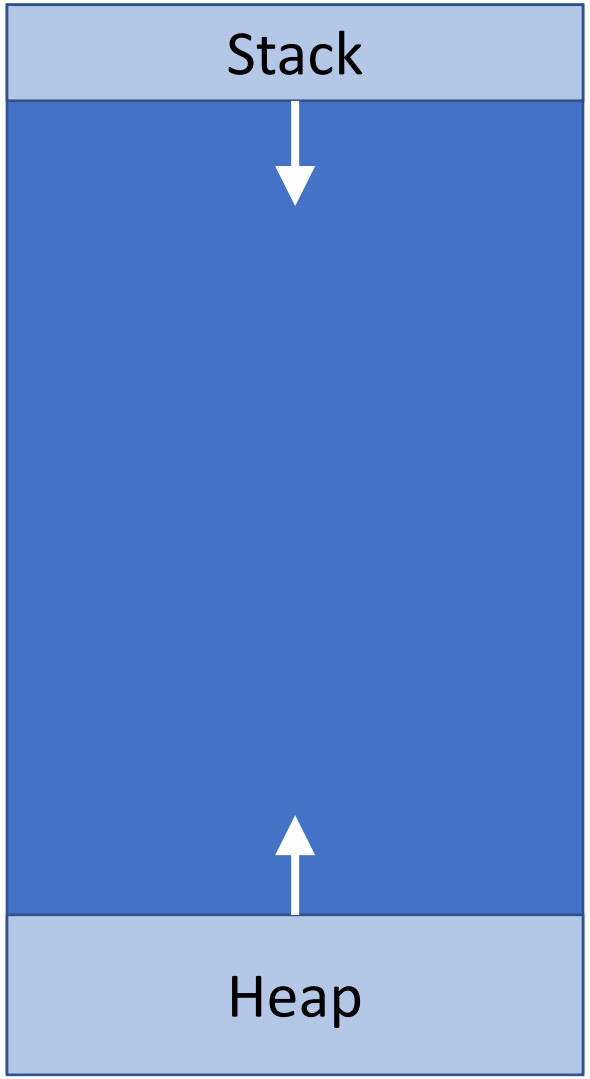
```
def factorial(n):
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result

print(factorial(5))
```

Recursive

```
def factorial(n):
    if (n < 2):
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
```

How does this look in memory?

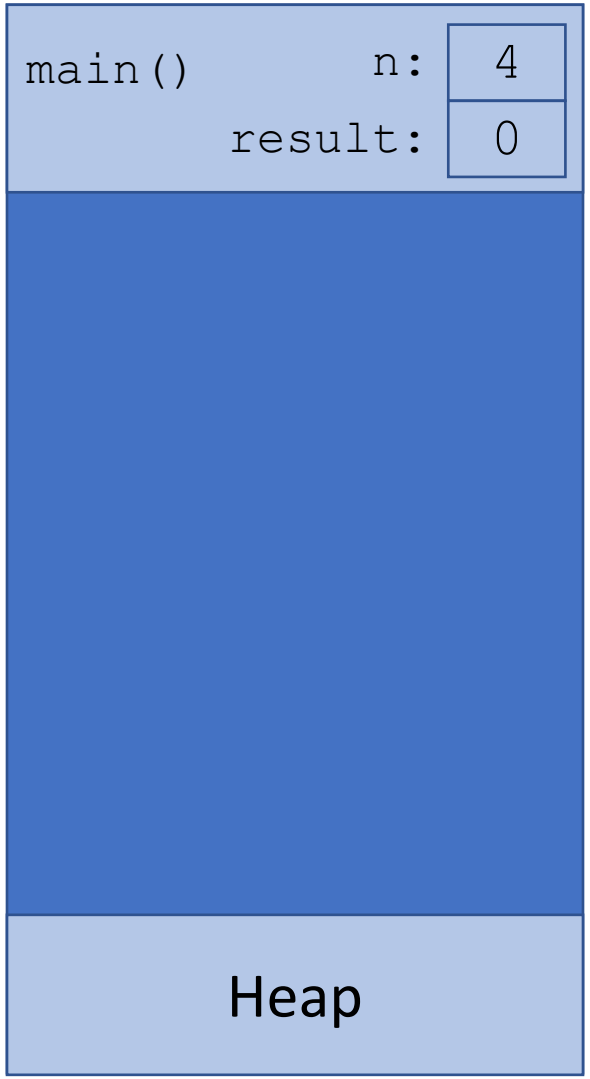
Memory




```
long factorial(int n) {
    if (n == 0) {
        return 1;
    }

    return n * factorial(n-1);
}

int main() {
    int n = 4;

    long result;
    result = factorial(n);

    cout << result << endl;
}
```

How does this look in memory?

Memory




```
long factorial(int n) {
    if (n == 0) {
        return 1;
    }

    return n * factorial(n-1);
}

int main() {
    int n = 4;

    long result = 0;
    result = factorial(n);

    cout << result << endl;
}
```

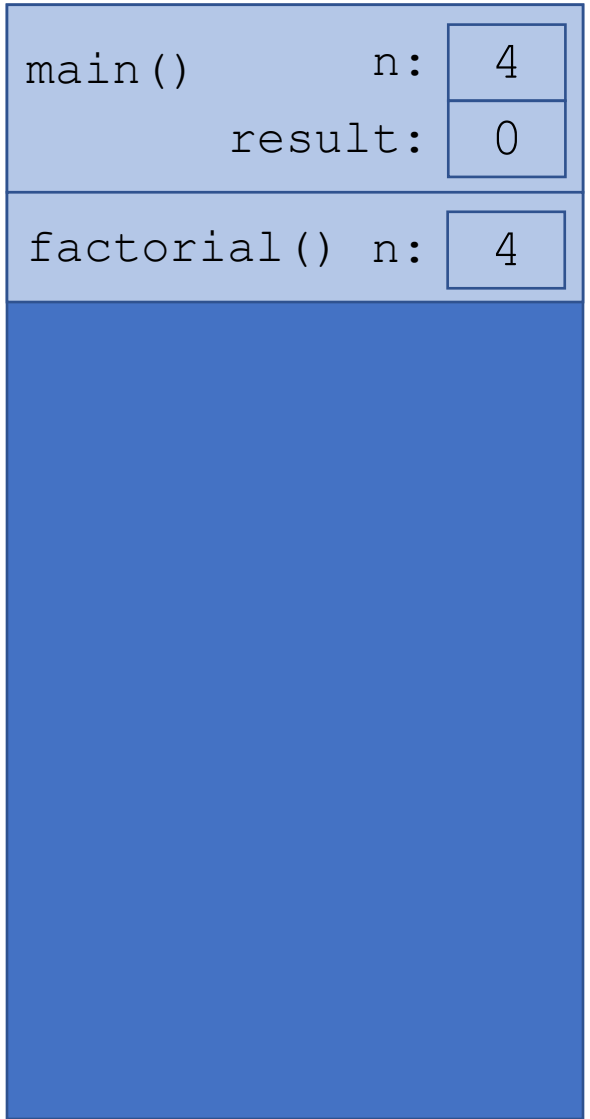
The “stack” part of memory is a stack

Function call: push

Return: pop

“Stack” part of memory is a stack

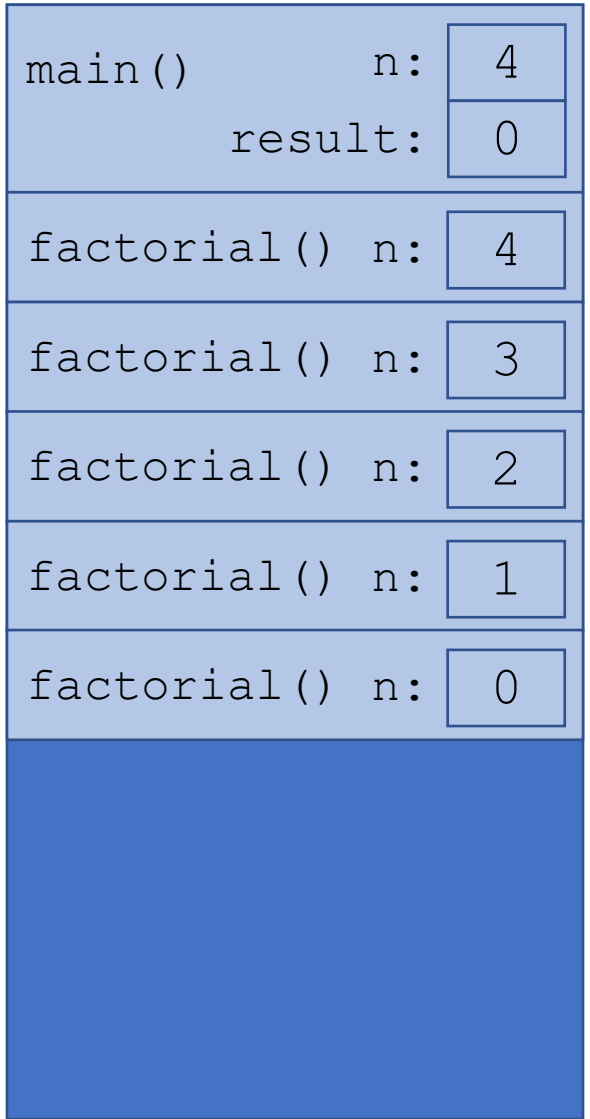
Memory




```
long factorial(int n) {
    if (n == 0) {
        return 1;
    }

    return n * factorial(n-1);
}

int main() {
    int n = 4;

    long result = 0;
    result = factorial(n);

    cout << result << endl;
}
```

“Stack” part of memory is a stack

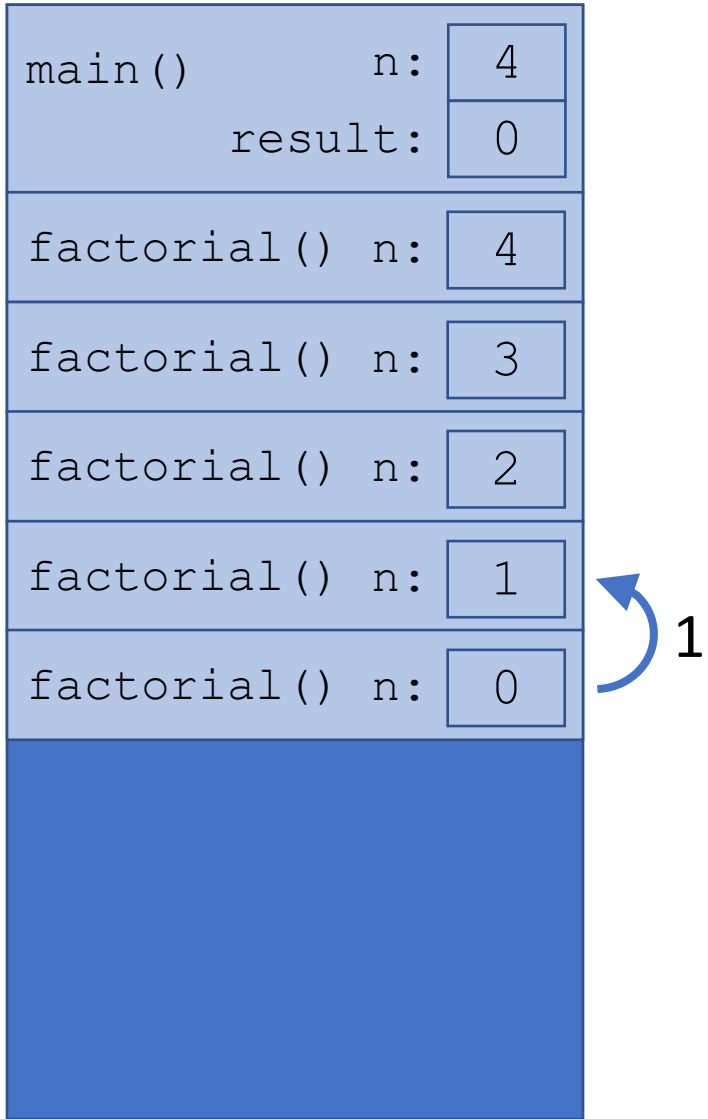
Memory




```
long factorial(int n) {
    if (n == 0) {
        return 1;
    }

    return n * factorial(n-1);
}

int main() {
    int n = 4;

    long result = 0;
    result = factorial(n);

    cout << result << endl;
}
```

“Stack” part of memory is a stack

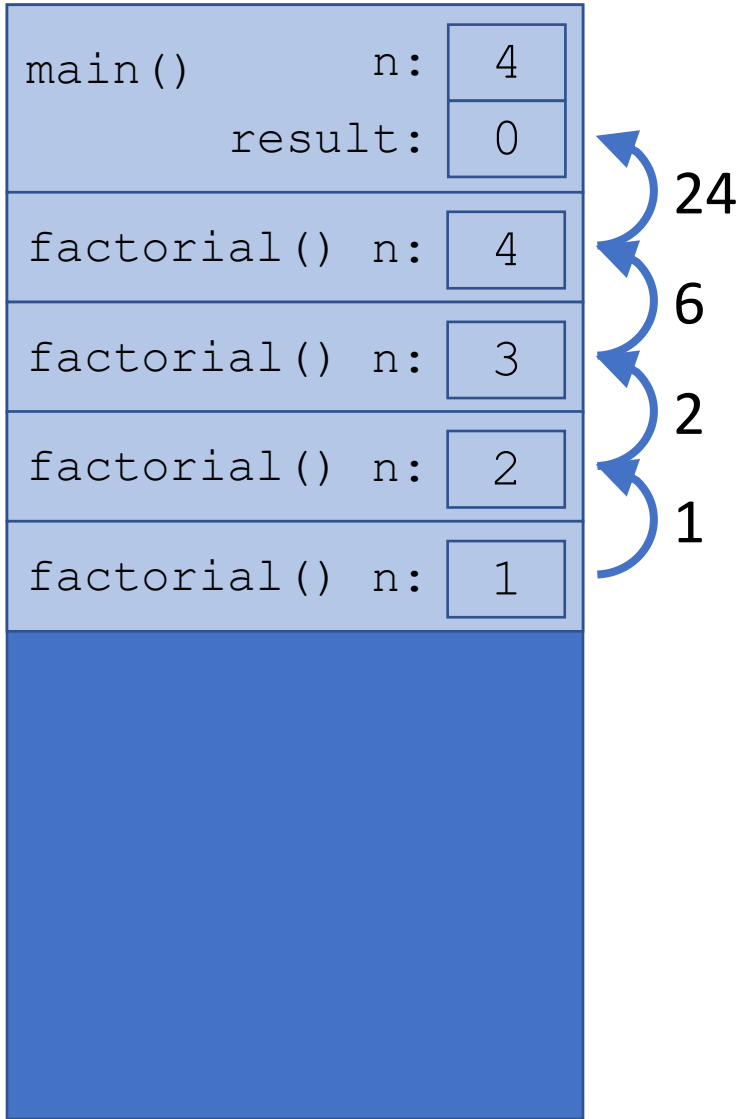
Memory




```
long factorial(int n) {
    if (n == 0) {
        return 1;
    }

    return n * factorial(n-1);
}

int main() {
    int n = 4;

    long result = 0;
    result = factorial(n);

    cout << result << endl;
}
```

“Stack” part of memory is a stack

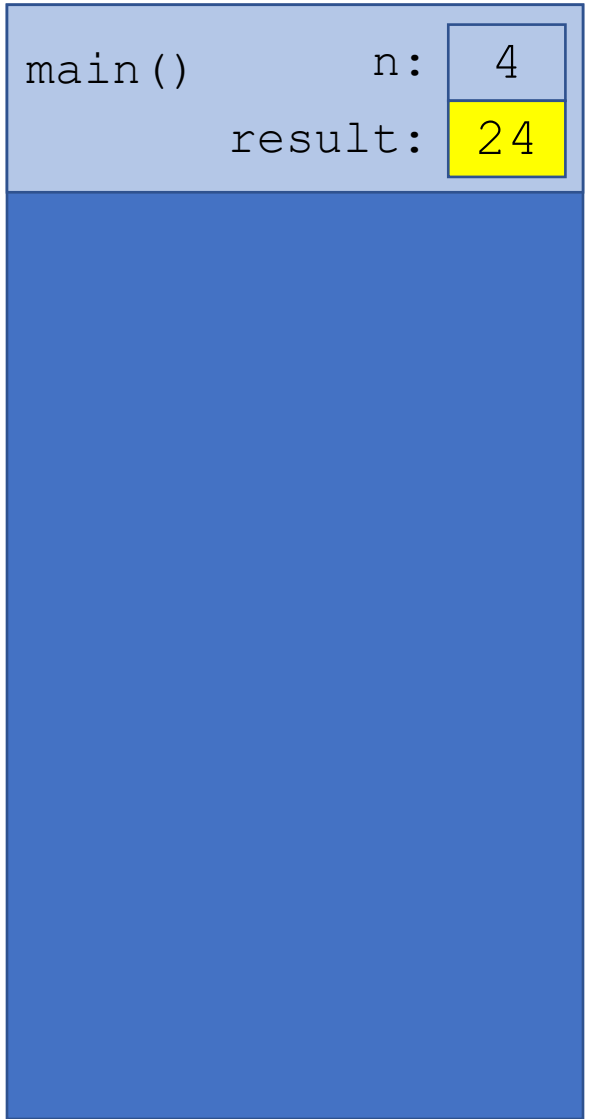
Memory




```
long factorial(int n) {
    if (n == 0) {
        return 1;
    }

    return n * factorial(n-1);
}

int main() {
    int n = 4;

    long result = 0;
    result = factorial(n);

    cout << result << endl;
}
```

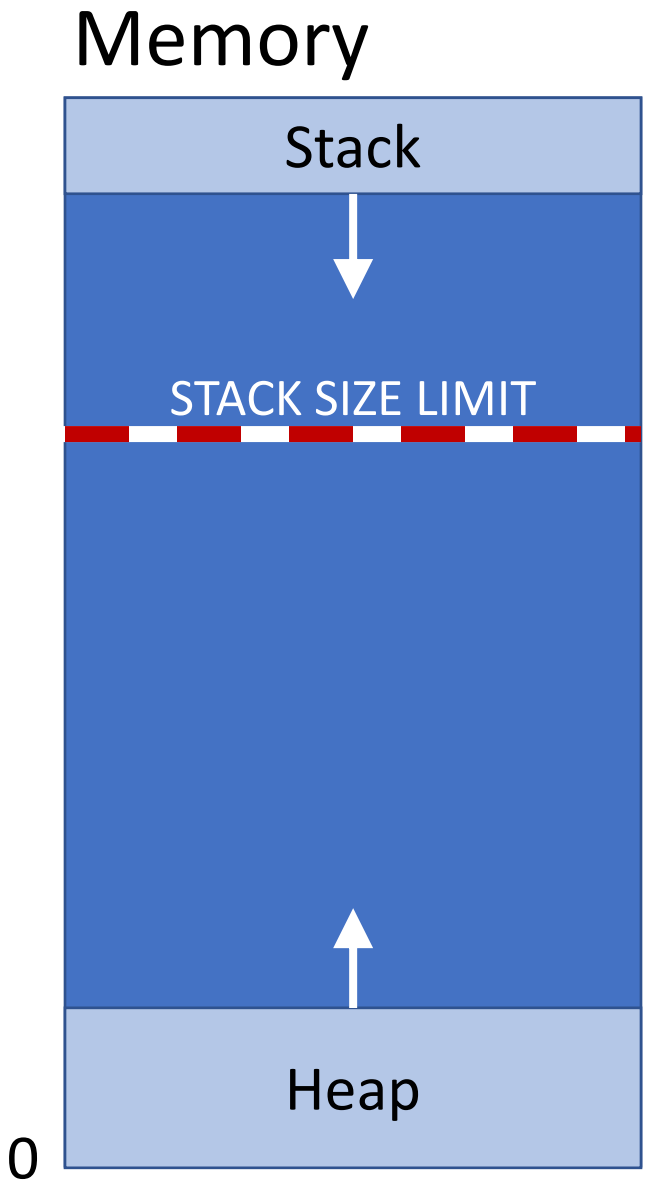
“Stack” part of memory is a stack

Memory



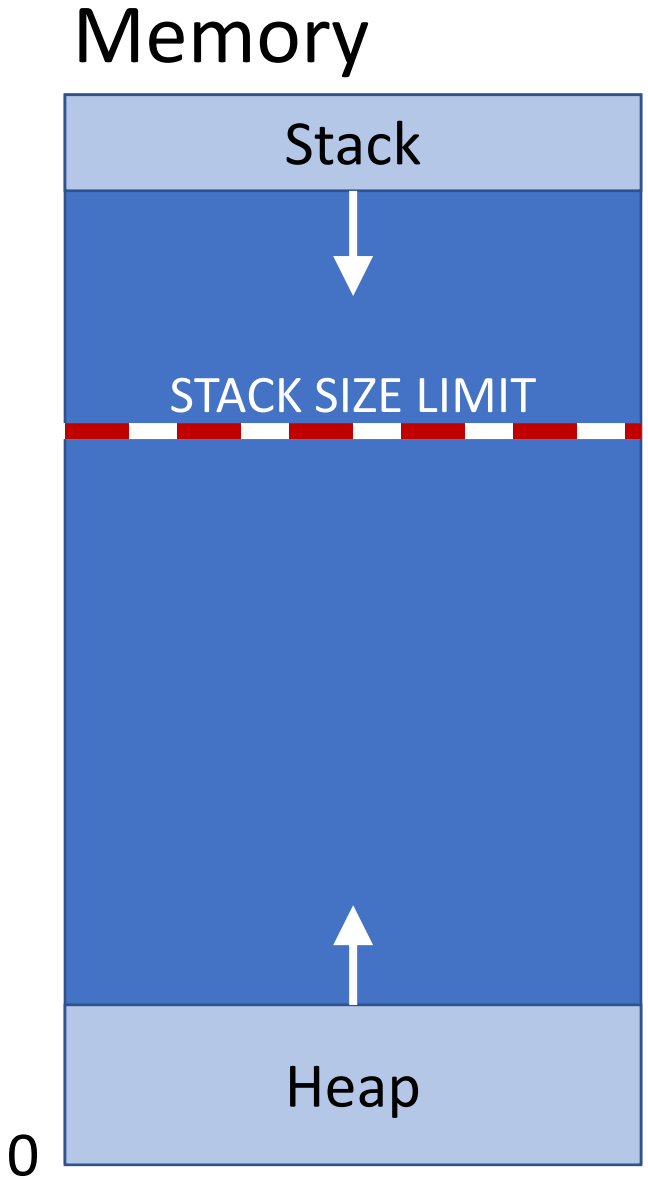

```
long factorial(int n) {
    if (n == 0) {
        return 1;
    }

    return n * factorial(n-1);
}

int main() {
    int n = 4;

    long result = 0;
    result = factorial(n);

    cout << result << endl;
}
```

What can go wrong?



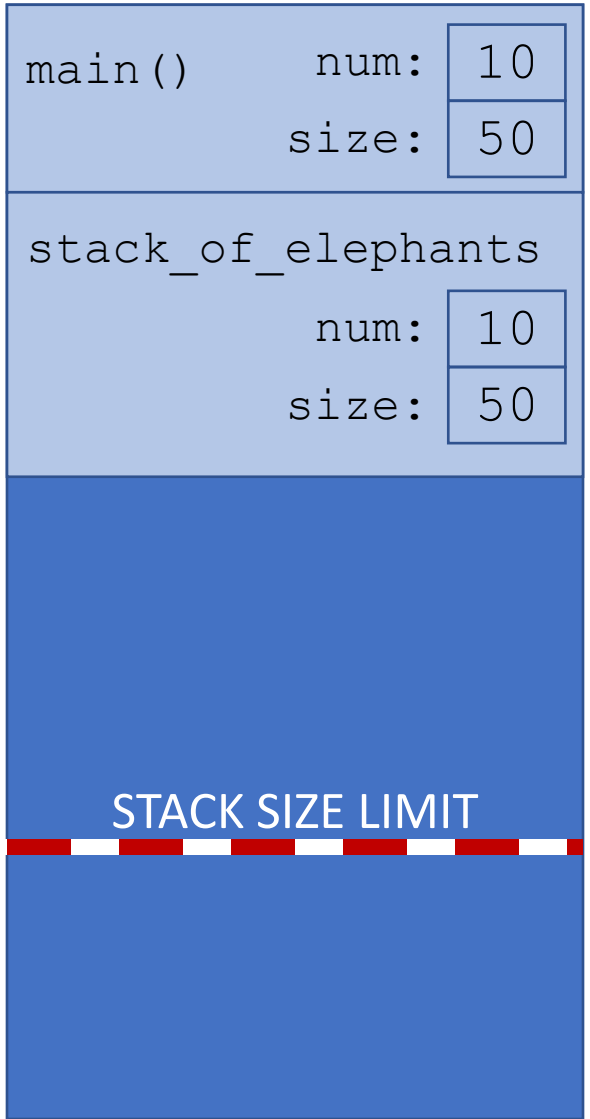

```
long factorial(int n) {
    if (n == 0) {
        return 1;
    }

    return n * factorial(n-1);
}

int main() {
    int n = 4;

    long result = 0;
    result = factorial(n);

    cout << result << endl;
}
```

Demo: Stack of elephants



Demo: Stack of elephants

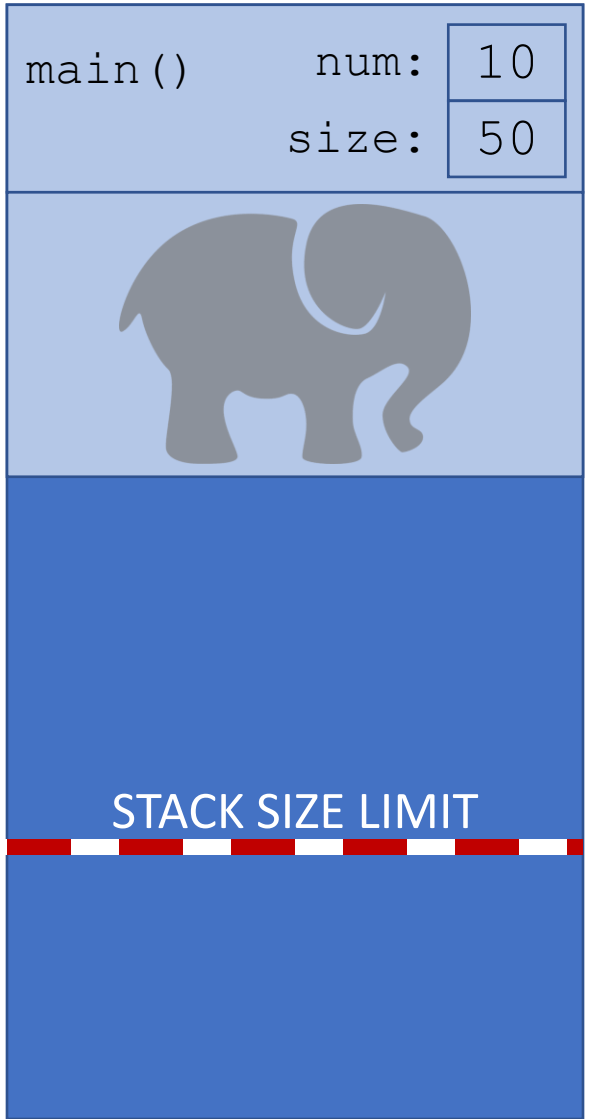
Memory




```
void stack_of_elephants(
        int num, int size) {
    if (num == 0) {
        return;
    }

    int elephant_array[size]

    stack_of_elephants(num-1, size);
}

int main() {
    int num = 10;
    int size = 50;
    stack_of_elephants(num, size)
}
```

Demo: Stack of elephants

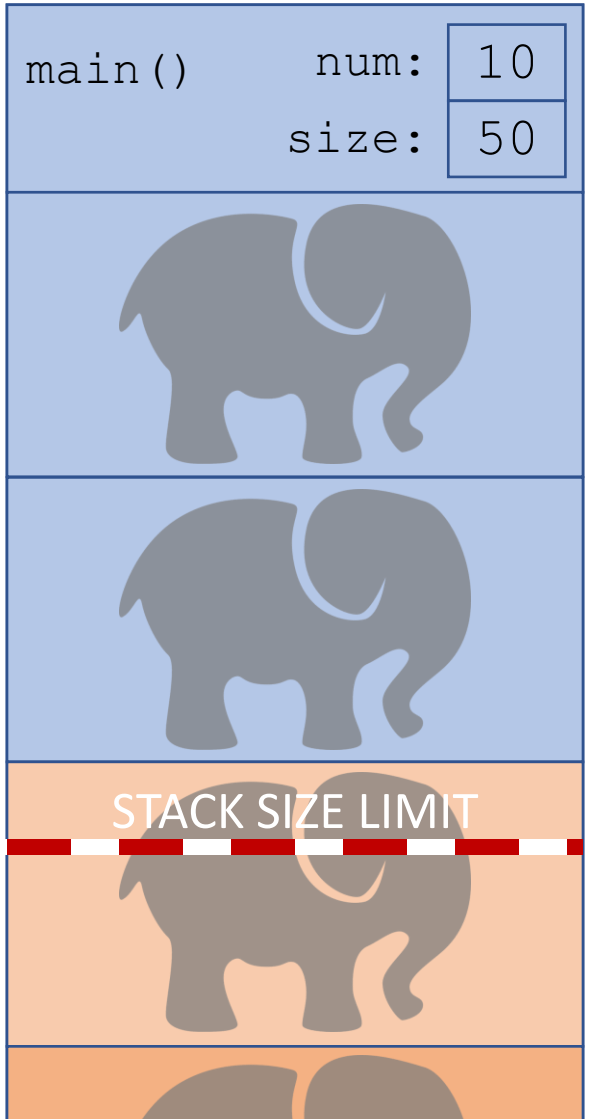
Memory




```
void stack_of_elephants(
        int num, int size) {
    if (num == 0) {
        return;
    }

    int elephant_array[size]

    stack_of_elephants(num-1, size);
}

int main() {
    int num = 10;
    int size = 50;
    stack_of_elephants(num, size)
}
```

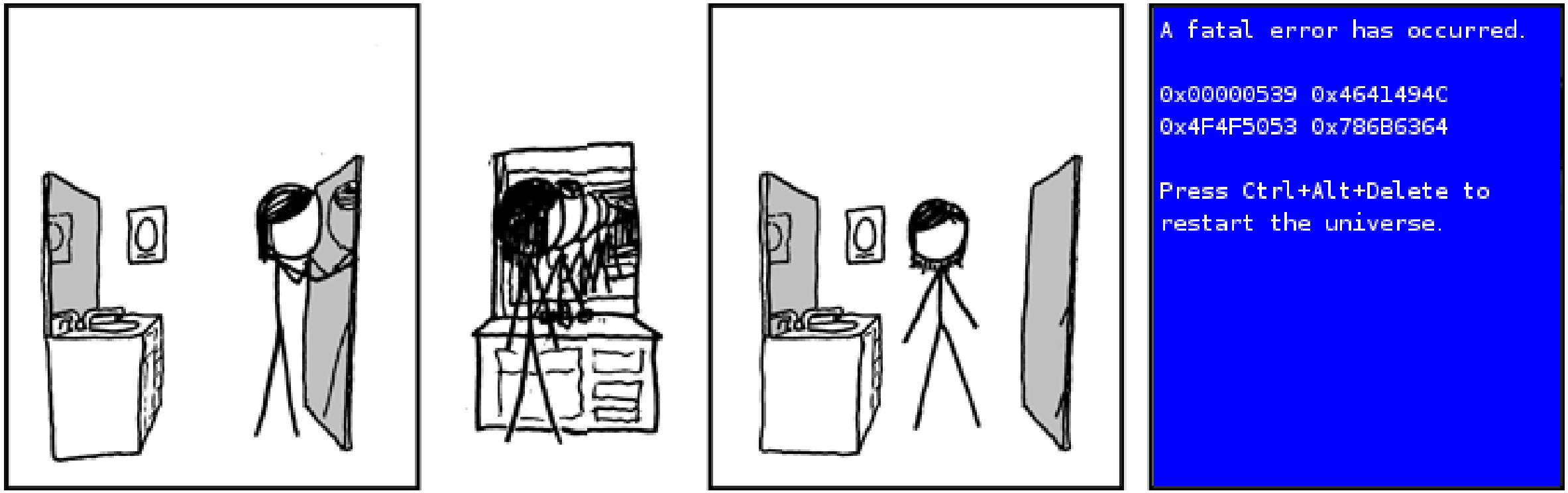
Demo: Stack of elephants

Memory




```
void stack_of_elephants(
        int num, int size) {
    if (num == 0) {
        return;
    }

    int elephant_array[size]

    stack_of_elephants(num-1, size);
}

int main() {
    int num = 10;
    int size = 50;
    stack_of_elephants(num, size)
}
```

500000



Images: http://xkcdsw.com/1361 (https://xkcd.com/555/)

Plan

Recursion

- Recursion
- Practical considerations
- Proof by induction

Divide and Conquer

- Sorting example

Dynamic Programming

- Overlapping subproblems
- Examples

Proof by Induction

Template

- Prove proposition for base case, e.g., n=0
- Inductive step
  - Assume proposition holds for n=k
  - Prove proposition holds for n=k+1

Proof by Induction

$f(n) = n(n+1)/2$

$k+1 \hookrightarrow (k+1)(k+1+1)/2$

Example: Prove sum of integers 1 to n is n(n+1)/2

Base $n = 1$

$f(1) = 1(1+1)/2 = 1$ ✓

Assume $n = k$ hold $f(k)$

Prove $n = k+1$ is correct

$$f(k+1) = f(k) + k + 1$$

$$= k(k+1)/2 + k + 1$$

$$= (k(k+1) + 2(k+1))/2$$

$$= (k+1)(k+2)/2$$

✓

Exercise 1

Use induction to prove that for $f(n) = n^2$ in $O(2^n)$

$n^2 \leq 1 \cdot 2^n$

$c = 1$

$n_0 = 5$

Base $n=5$: $5^2 \leq 2^5$

$25 \leq 32$ ✓

Assume $n=k$: $k^2 \leq 2^k$

Prove $n=k+1$:

$(k+1)^2 \leq 2^{(k+1)}$

Plan

Recursion

- Recursion
- Practical considerations
- Proof by induction

Divide and Conquer

- Sorting example

Dynamic Programming

- Overlapping subproblems
- Examples

Divide and Conquer

Merge sort

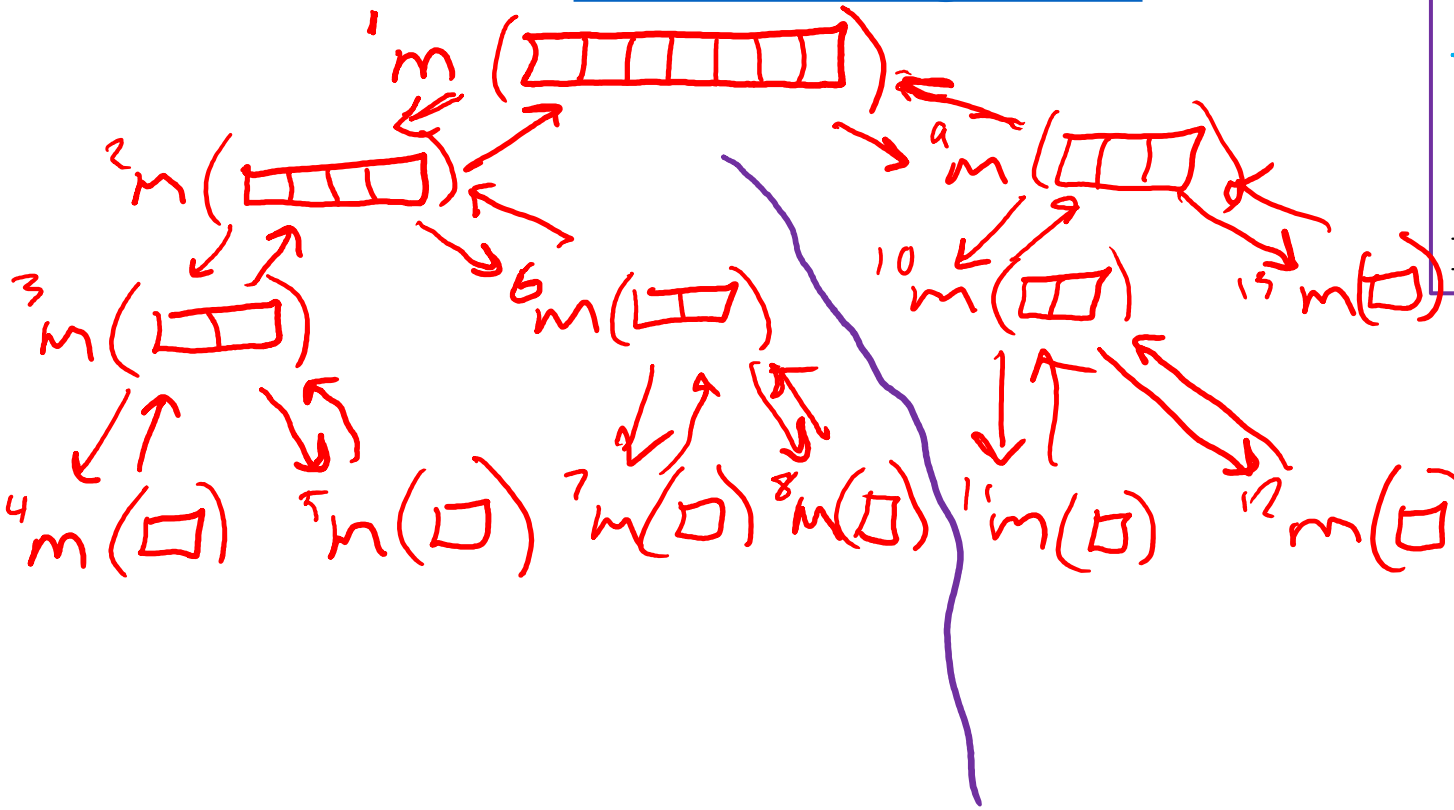
Detailed code: [15-112 Merge Sort]

```
def mergesort(L):
  if (len(L) < 2):
    return L
  else:
    mid = len(L)//2
    left = mergesort(L[:mid])
    right = mergesort(L[mid:])
    return merge(left, right)

print(mergesort([1,5,3,4,2,0]))
```

Divide and Conquer

Merge sort

Detailed code: [15-112 Merge Sort]



```
def mergesort(L):
  if (len(L) < 2):
    return L
  else:
    mid = len(L)//2
    left = mergesort(L[:mid])
    right = mergesort(L[mid:])
    return merge(left, right)

print(mergesort([1,5,3,4,2,0]))
```

Plan

Recursion

- Recursion
- Practical considerations
- Proof by induction

Divide and Conquer

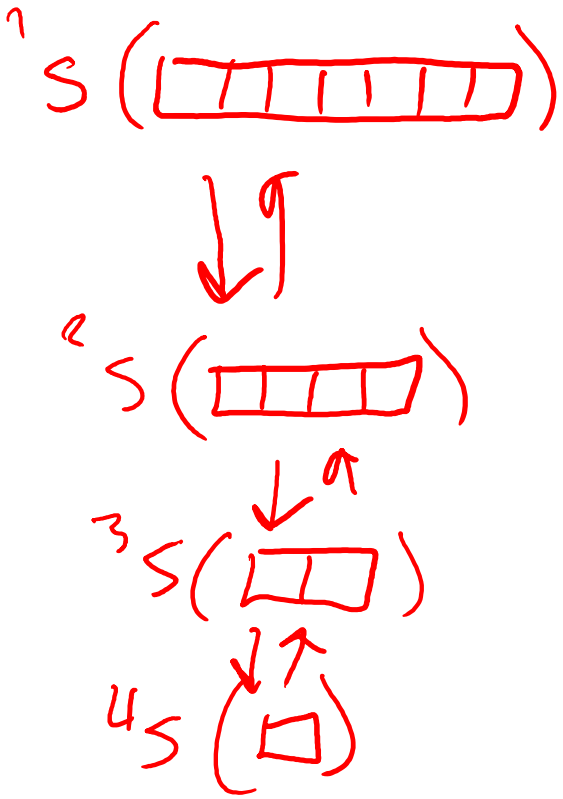
- Sorting example

Dynamic Programming

- Overlapping subproblems
- Examples

Reminder Binary Search

Graph of recursive calls




```
binarySearch2(data, key, first, last) {
  // TODO base case
  int mid = (first + last) // 2;
  if (data[mid] == key)
    return true;
  else if (data[mid] > key)
    return binarySearch2(data, key, first, mid-1);
  else
    return binarySearch2(data, key, mid+1, last)
}
```

Exercise 2: Fibonacci Naïve

Notebook

Fibonacci

```
N Value
0 1
1 1
2 2 = 1+1
3 3 = 1+2
4 5 = 2+3
5 8 = 3+5
6 13 = 5+8
7 21 = 8+13
```

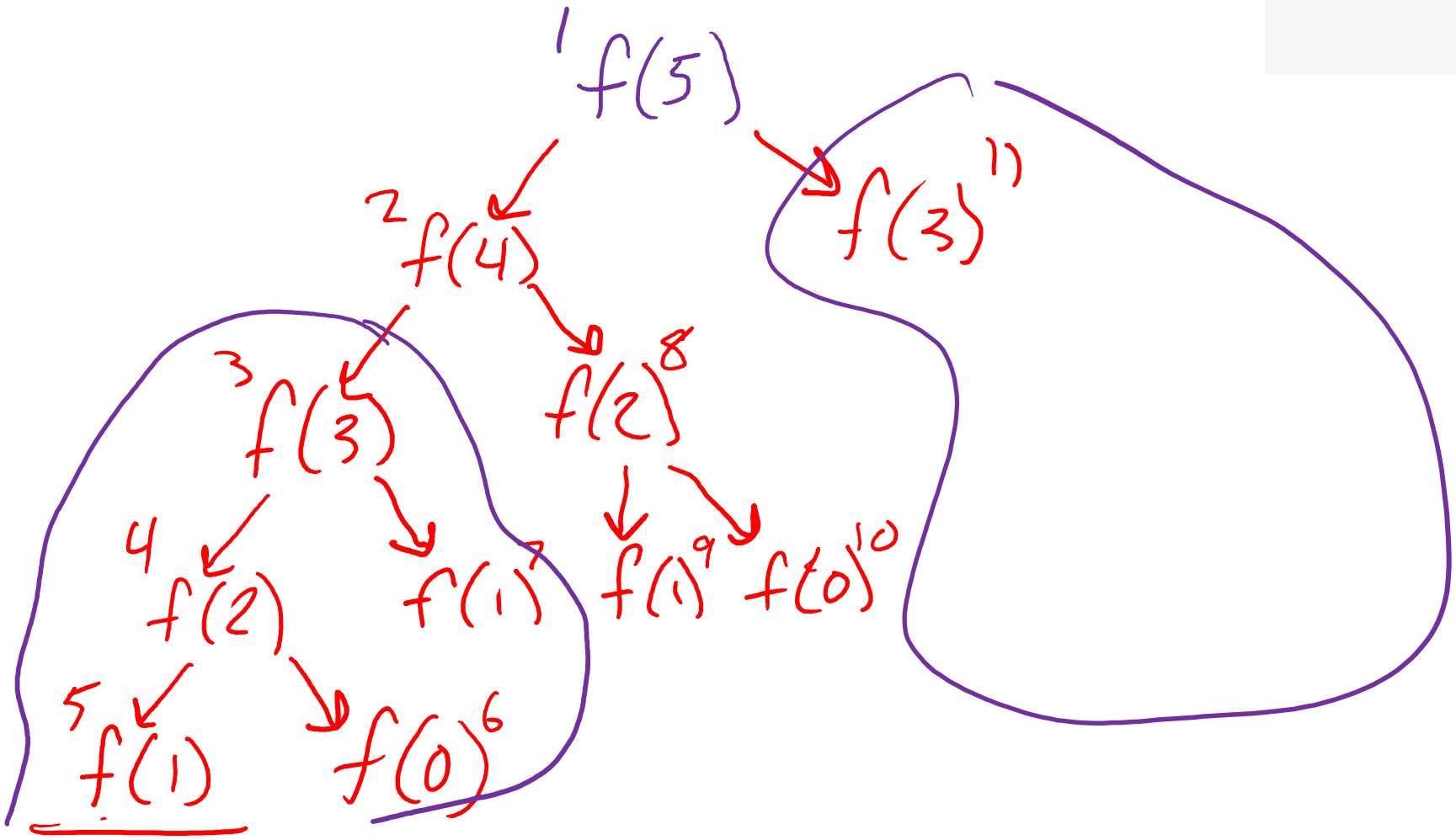
$f(n) = f(n-2) + f(n-1)$

https://www.cs.cmu.edu/~112/notes/notes-recursion-part2.html#memoization

Exercise 3: Fibonacci Naïve

```
def fib(n):
    if n < 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

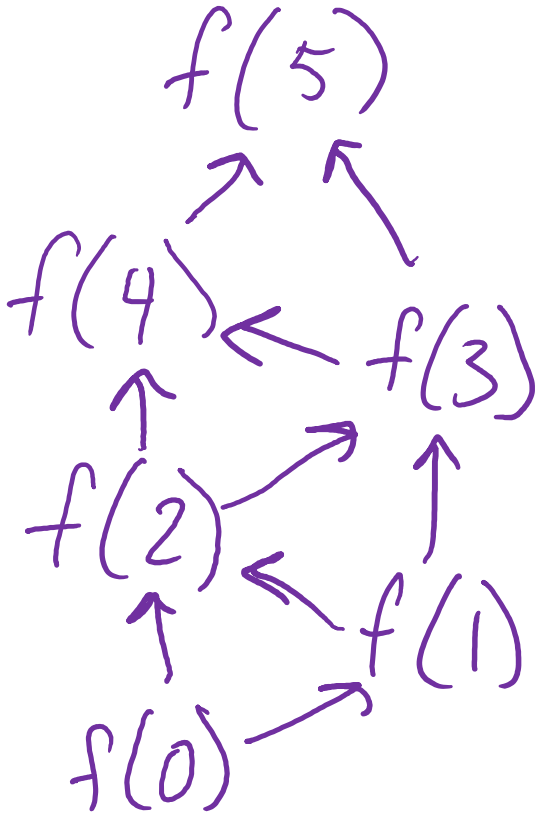
Graph of recursive calls



https://www.cs.cmu.edu/~112/notes/notes-recursion-part2html#memoization

Fibonacci Dynamic Programming

Graph with overlapping subproblems



https://www.cs.cmu.edu/~112/notes/notes-recursion-part2.html#memoization

Fibonacci Dynamic Programming

Notebook

https://www.cs.cmu.edu/~112/notes/notes-recursion-part2.html#memoization

Dynamic Programming

Efficient way to solve problems that can take a long time

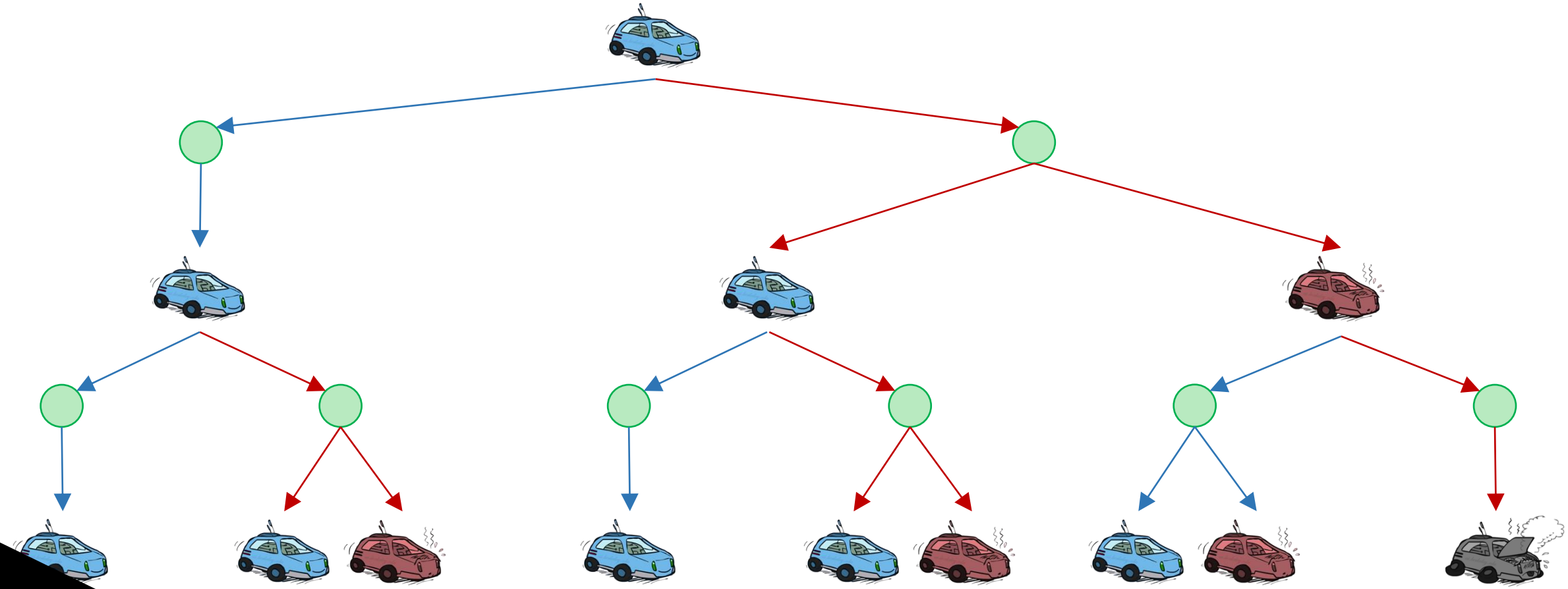
- Dividing them into a set of smaller problems
- Optimal solutions of each subproblem are stored and used to find the optimal solution for the larger problem
- Unlike divide and conquer, these problems can overlap.

Slide credit: Avrim Blum and Anupam Gupta

Examples from ML

Dynamic Programming

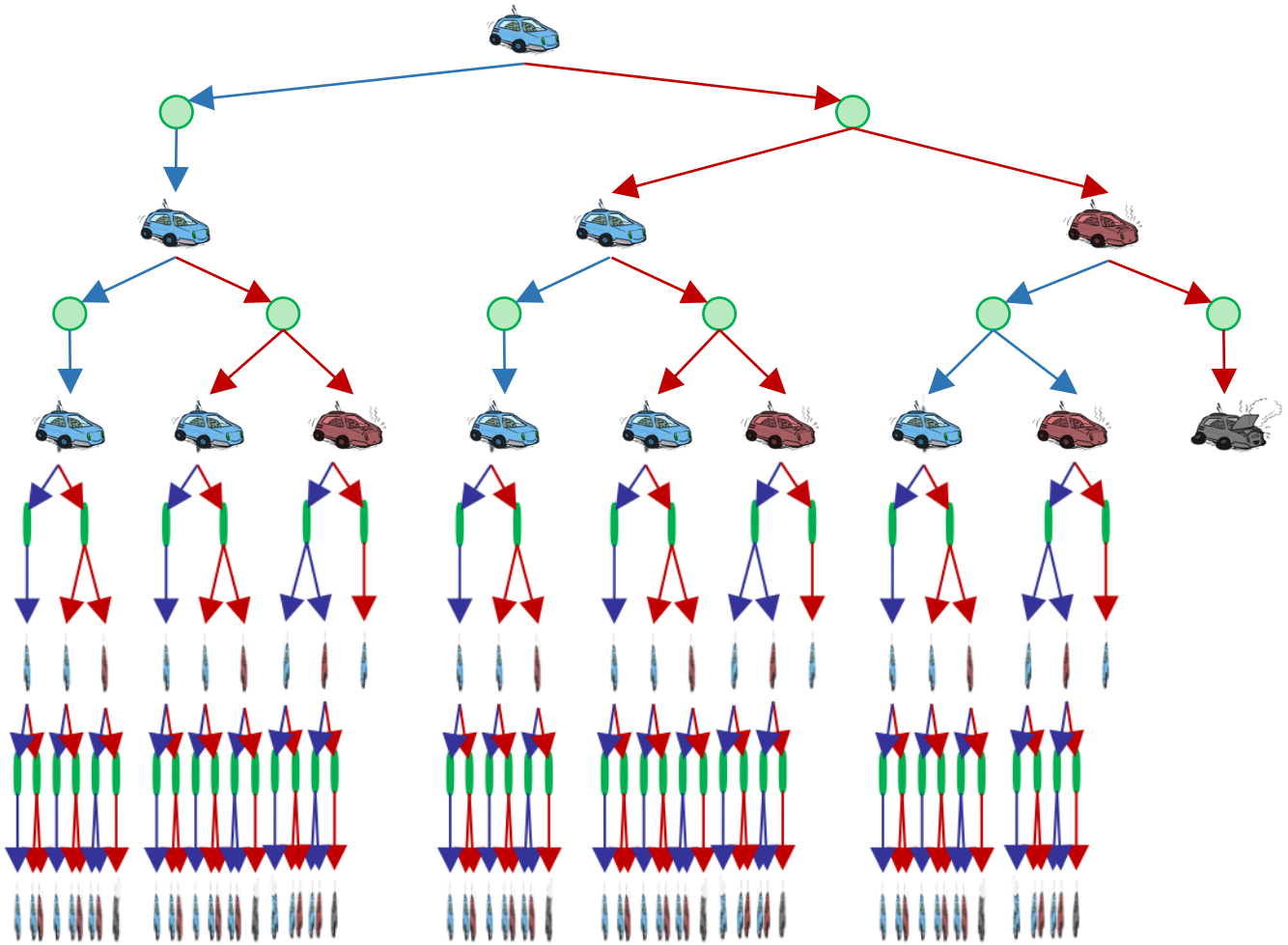
MDP: Racing Search Tree



10-601 Example

Slide: ai.berkeley

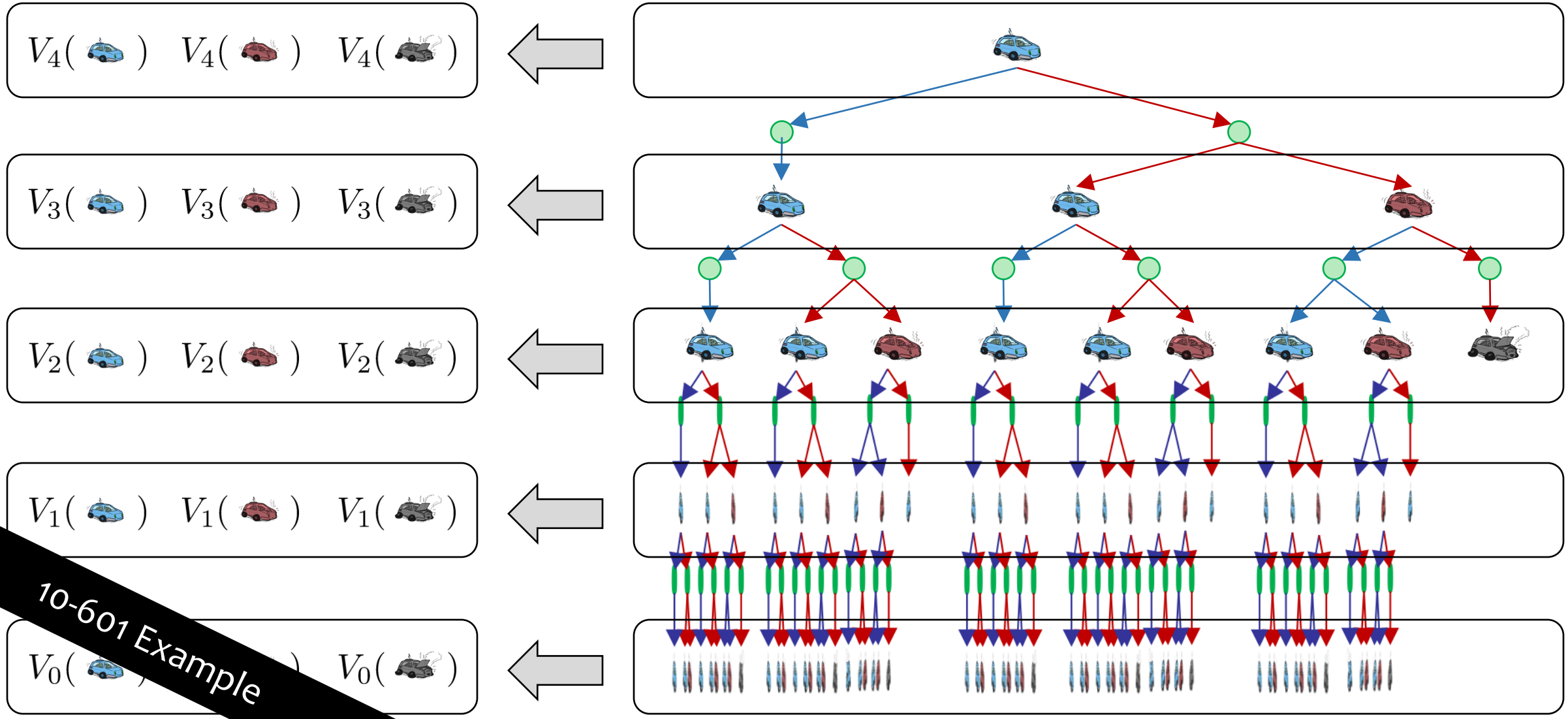
MDP: Racing Search Tree



10-601 Example

Slide: ai.berkeley

MDP: Racing Tree Example



10-601 Example

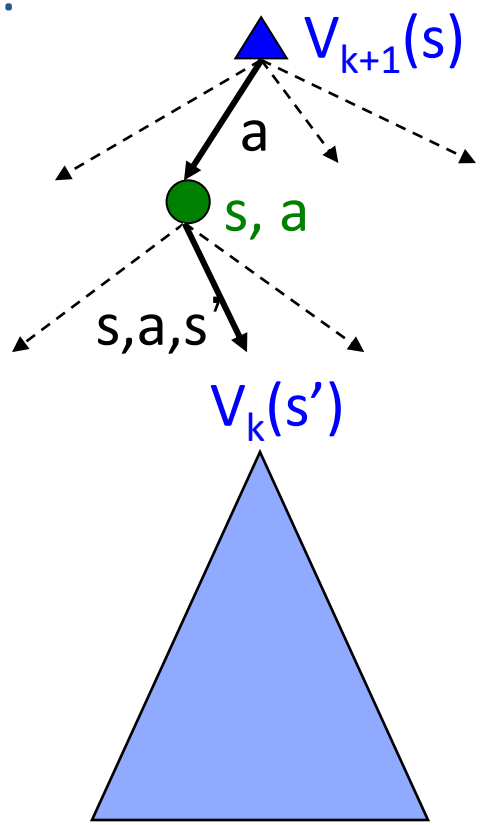
Slide: ai.berkeley.edu

MDP: Value Iteration

Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero

Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$



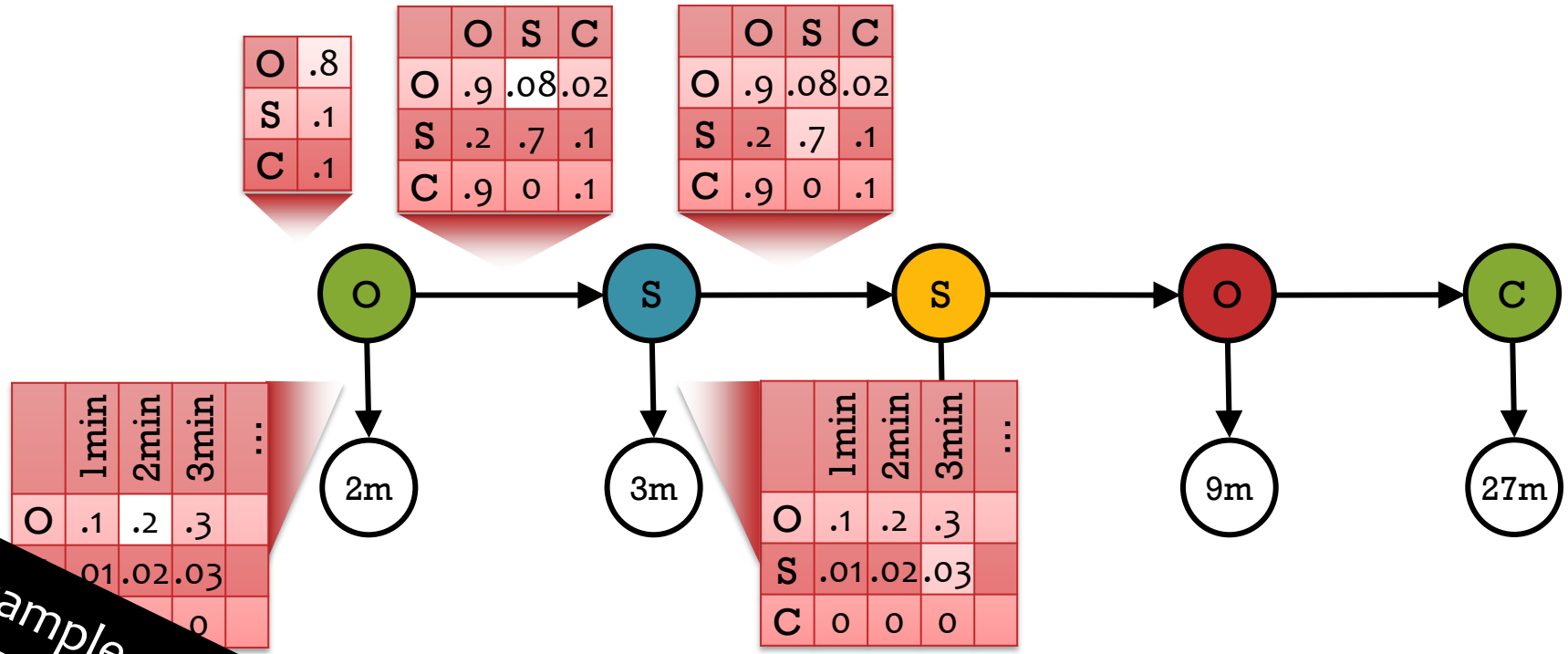
Repeat until convergence

10-601 Example

Hidden Markov Model

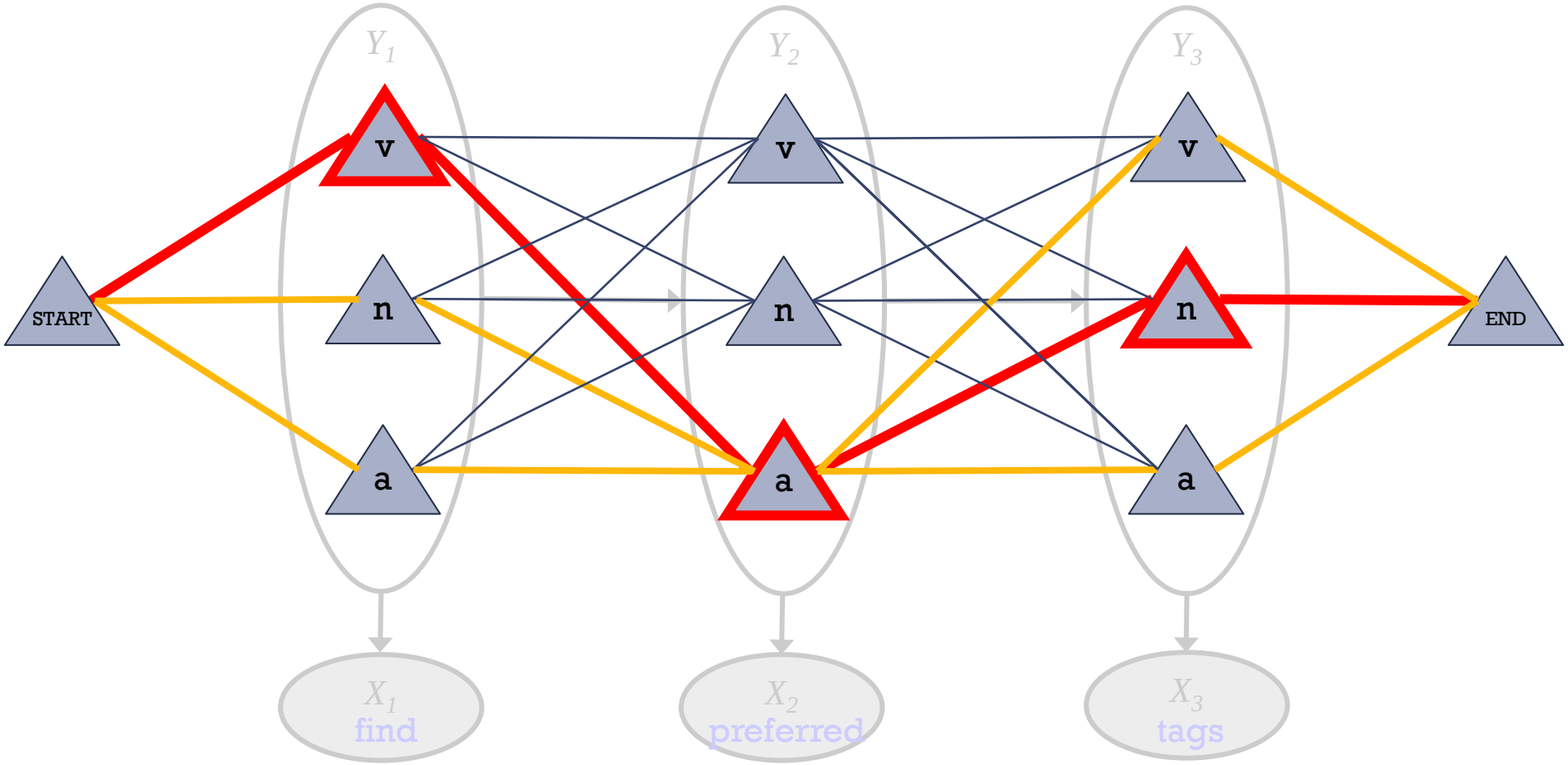
A Hidden Markov Model (HMM) provides a joint distribution over the the tunnel states / travel times with an assumption of dependence between adjacent tunnel states.

$$p(\text{O}, \text{S}, \text{S}, \text{O}, \text{C}, 2\text{m}, 3\text{m}, 18\text{m}, 9\text{m}, 27\text{m}) = (.8 * .08 * .2 * .7 * .03 * \dots)$$



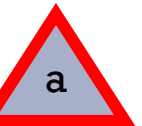
10-601 Example

Forward-Backward Algorithm: Finds Marginals



- So p(**v a n**) = (1/Z) * product weight of one path

Marginal probability p(Y_2 = a)

= (1/Z) * total weight of *all* paths through 

10-601 Example