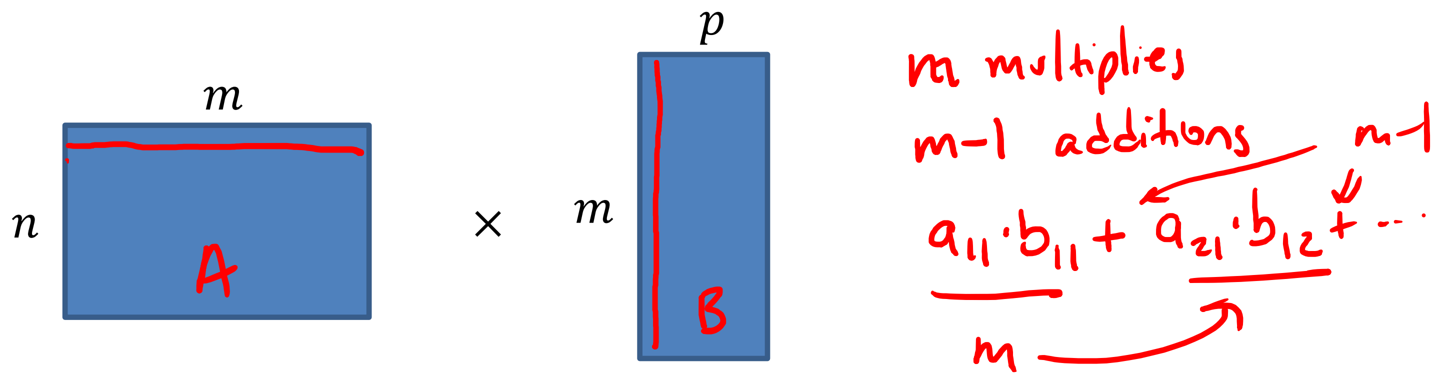
CS4102 Algorithms

Spring 2019

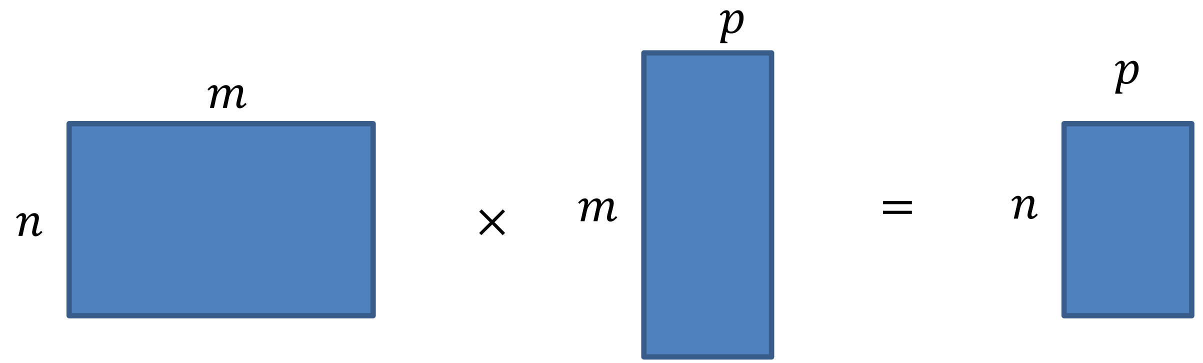
Warm up

How many arithmetic operations are required to multiply a $n \times m$ Matrix with a $m \times p$ Matrix?

(don't overthink this)



How many arithmetic operations are required to multiply a $n \times m$ Matrix with a $m \times p$ Matrix?



- m multiplications and additions per element
- $n \cdot p$ elements to compute
- Total cost: $m \cdot n \cdot p$

$n \cdot p(m + m - 1)$

Today’s Keywords

- Dynamic Programming
- Matrix Chaining
- Seam Carving
- Longest Common Subsequence

CLRS Readings

- Chapter 15

Administrativa

- HW5 out by tomorrow morning
  - Due March 27 at 11pm
  - Seam Carving!
  - Dynamic Programming (implementation)
  - Java or Python
- Midterm
  - Grading underway! Should be returned tomorrow
- HW4 grading in-progress

Dynamic Programming

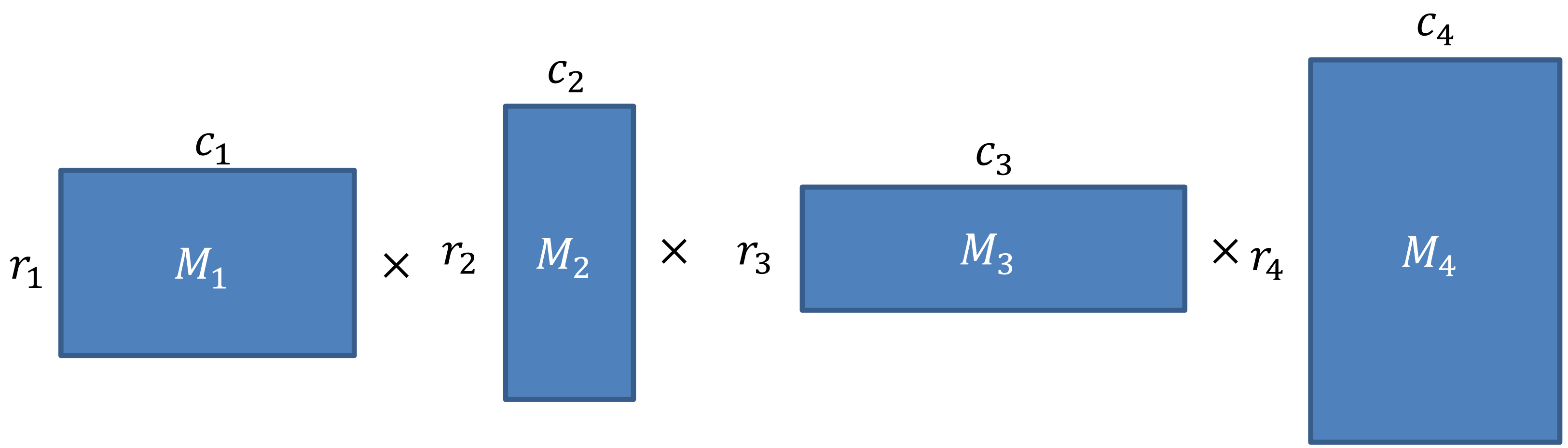
- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify recursive structure of the problem
     - What is the “last thing” done?
  2. Select a good order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory

Generic Top-Down Dynamic Programming Soln

```
mem = {}
def myDPalgo(problem):
    if mem[problem] not blank:
        return mem[problem]
    if baseCase(problem):
        solution = solve(problem)
        mem[problem] = solution
        return solution
    for subproblem of problem:
        subsolutions.append(myDPalgo(subproblem))
    solution = OptimalSubstructure(subsolutions)
    mem[problem] = solution
    return solution
```

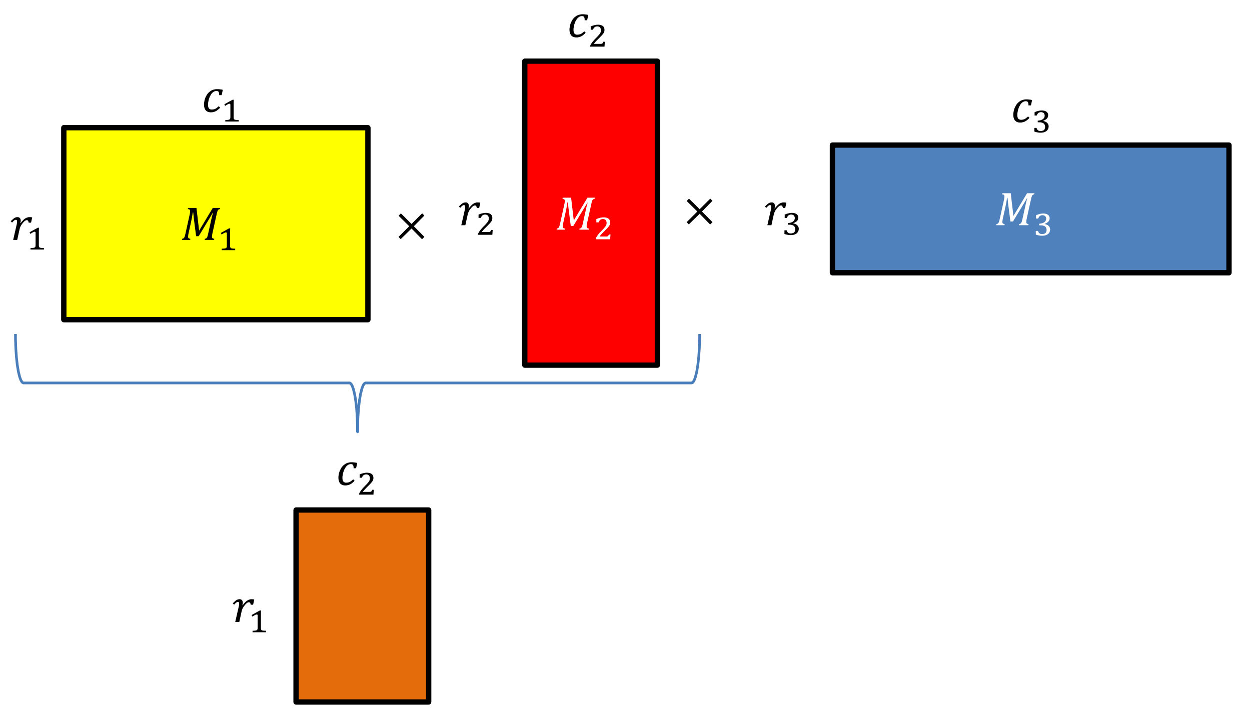
Matrix Chaining

- Given a sequence of Matrices $(M_1, \dots, M_n)$, what is the most efficient way to multiply them?



$c_1 = r_2$
$c_2 = r_3$

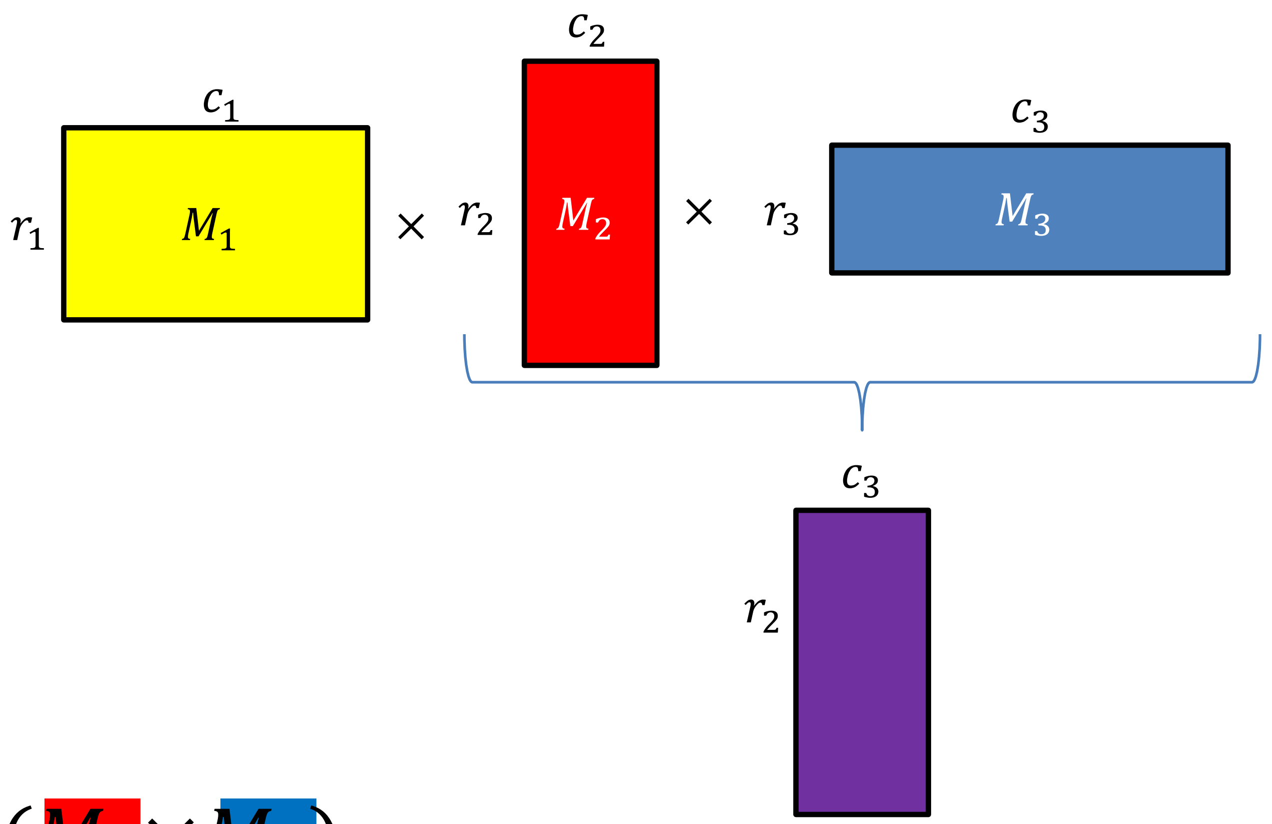
Order Matters!



- $(M_1 \times M_2) \times M_3$
  - uses $(c_1 \cdot r_1 \cdot c_2) + c_2 \cdot r_1 \cdot c_3$ operations

$c_1 = r_2$
$c_2 = r_3$

Order Matters!



- $M_1 \times (M_2 \times M_3)$
  - uses $c_1 \cdot r_1 \cdot c_3 + (c_2 \cdot r_2 \cdot c_3)$ operations

$c_1 = r_2$
$c_2 = r_3$

Order Matters!

- $(M_1 \times M_2) \times M_3$
  - uses $(c_1 \cdot r_1 \cdot c_2) + c_2 \cdot r_1 \cdot c_3$ operations
  - $(10 \cdot 7 \cdot 20) + 20 \cdot 7 \cdot 8 = 2520$
- $M_1 \times (M_2 \times M_3)$
  - uses $c_1 \cdot r_1 \cdot c_3 + (c_2 \cdot r_2 \cdot c_3)$ operations
  - $10 \cdot 7 \cdot 8 + (20 \cdot 10 \cdot 8) = 2160$

$M_1 = 7 \times 10$
$M_2 = 10 \times 20$
$M_3 = 20 \times 8$

$c_1 = 10$
$c_2 = 20$
$c_3 = 8$
$r_1 = 7$
$r_2 = 10$
$r_3 = 20$

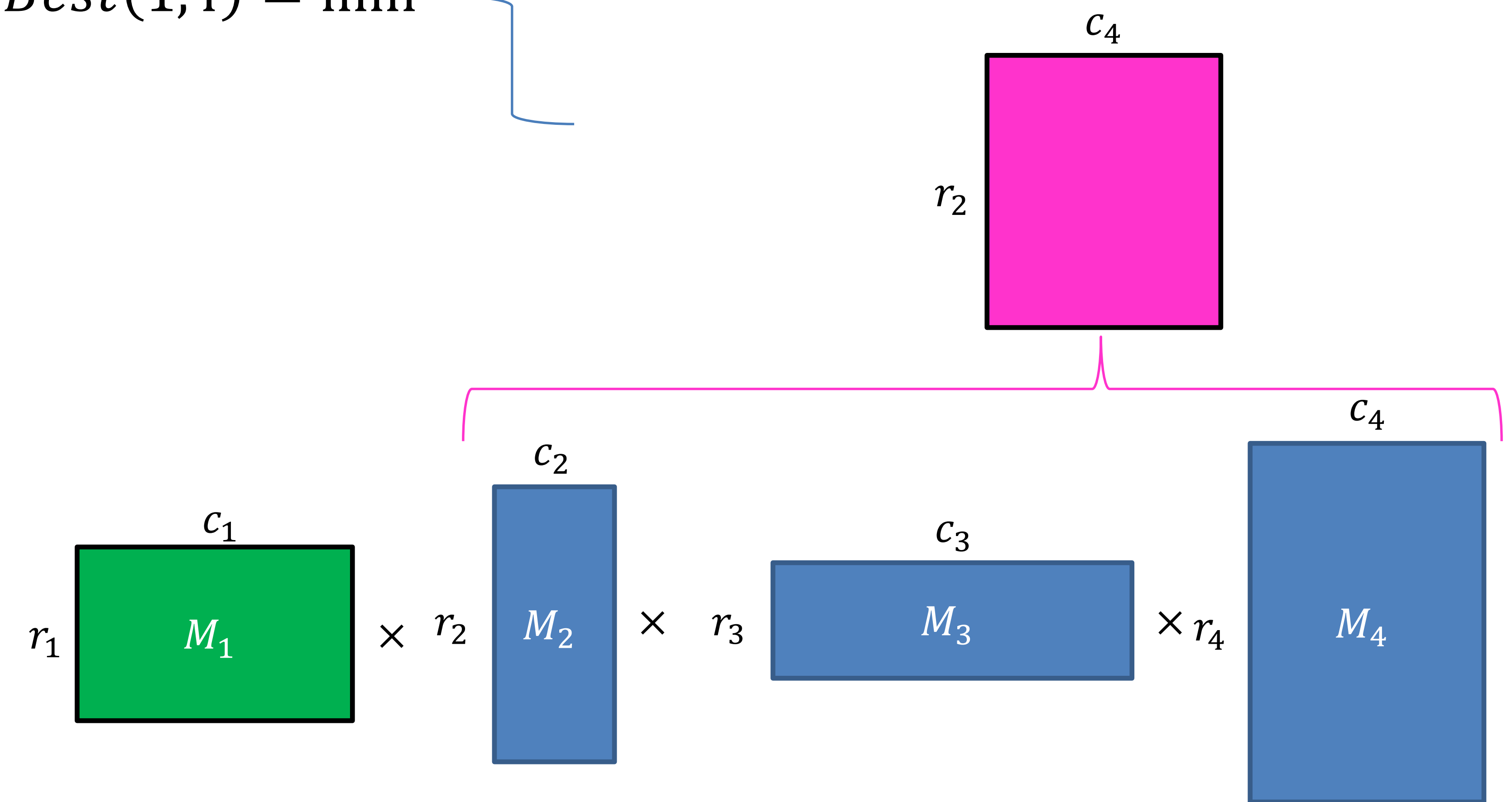
Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify recursive structure of the problem
     - What is the "last thing" done?
  2. Select a good order for solving subproblems
     - "Top Down": Solve each recursively
     - "Bottom Up": Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory

1. Identify the Recursive Structure of the Problem

$Best(1, n)$ = cheapest way to multiply together M_1 through M_n

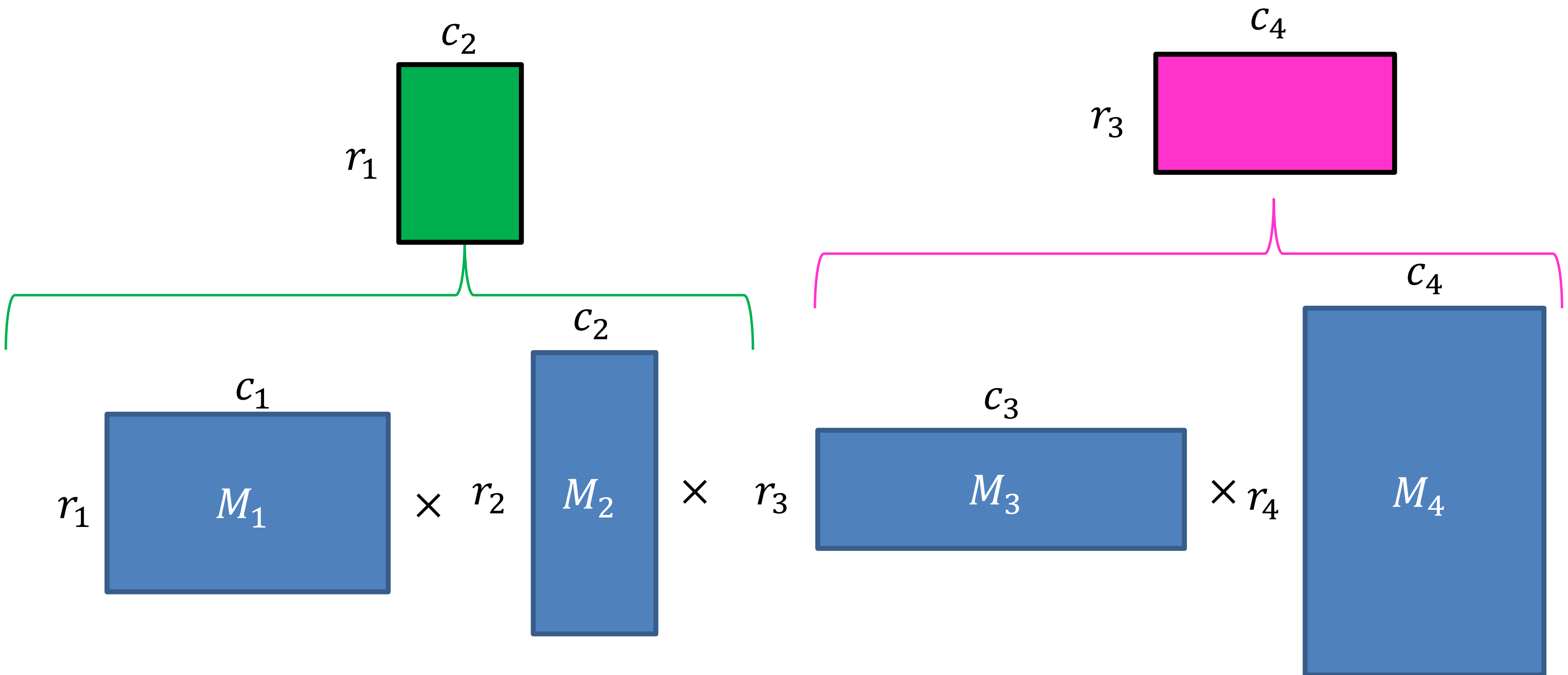
$$Best(1,4) = \min \begin{cases} Best(2,4) + r_1 r_2 c_4 \\ \\ \end{cases}$$



1. Identify the Recursive Structure of the Problem

$Best(1, n)$ = cheapest way to multiply together M_1 through M_n

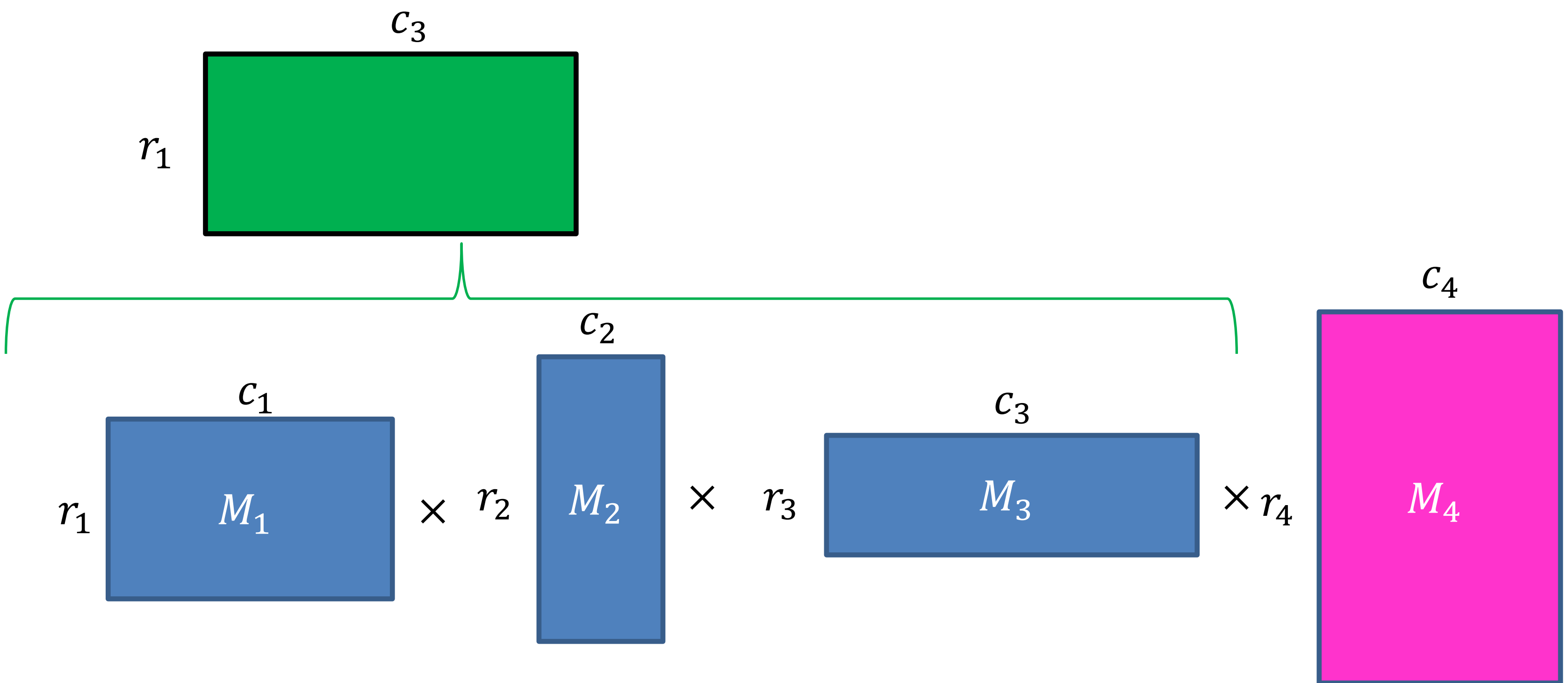
$$Best(1,4) = \min \begin{cases} Best(2,4) + r_1 r_2 c_4 \\ Best(1,2) + Best(3,4) + r_1 r_3 c_4 \\ \\ \end{cases}$$



1. Identify the Recursive Structure of the Problem

$Best(1,n)$ = cheapest way to multiply together M_1 through M_n

$$Best(1,4) = \min \begin{cases} Best(2,4) + r_1 r_2 c_4 \\ Best(1,2) + Best(3,4) + r_1 r_3 c_4 \\ Best(1,3) + r_1 r_4 c_4 \end{cases}$$



1. Identify the Recursive Structure of the Problem

- In general:

$Best(i,j)$ = cheapest way to multiply together M_i through M_j

$r_i \times c_k$ $\quad$ $r_{k+1} \times c_j$ $\quad$ $c_k = r_{k+1}$

$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$$Best(1,n) = \min \begin{cases} Best(2,n) + r_1 r_2 c_n \\ Best(1,2) + Best(3,n) + r_1 r_3 c_n \\ Best(1,3) + Best(4,n) + r_1 r_4 c_n \\ Best(1,4) + Best(5,n) + r_1 r_5 c_n \\ \quad \ldots \\ Best(1,n-1) + r_1 r_n c_n \end{cases}$$

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify recursive structure of the problem
     - What is the "last thing" done?
  2. Select a good order for solving subproblems
     - "Top Down": Solve each recursively
     - "Bottom Up": Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory

1. Identify the Recursive Structure of the Problem

- In general:

$Best(i,j) =$ cheapest way to multiply together M_i through M_j

$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$O(4^n)$

$$Best(i,i) = 0$$

Read from M[n] if present

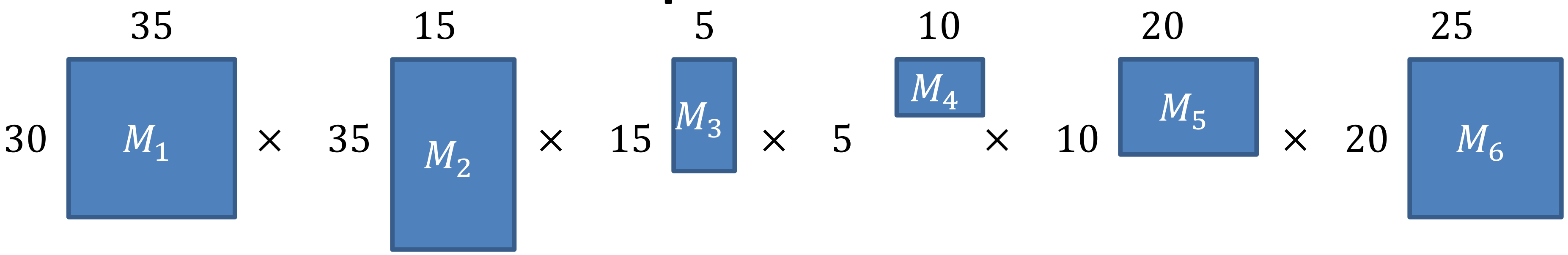
Save to M[n]

$$Best(1,n) = \min \begin{cases} Best(2,n) + r_1 r_2 c_n \\ Best(1,2) + Best(3,n) + r_1 r_3 c_n \\ Best(1,3) + Best(4,n) + r_1 r_4 c_n \\ Best(1,4) + Best(5,n) + r_1 r_5 c_n \\ \dots \\ Best(1,n-1) + r_1 r_n c_n \end{cases}$$

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify recursive structure of the problem
     - What is the "last thing" done?
  2. Select a good order for solving subproblems
     - "Top Down": Solve each recursively
     - "Bottom Up": Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory

2. Select a good order for solving subproblems

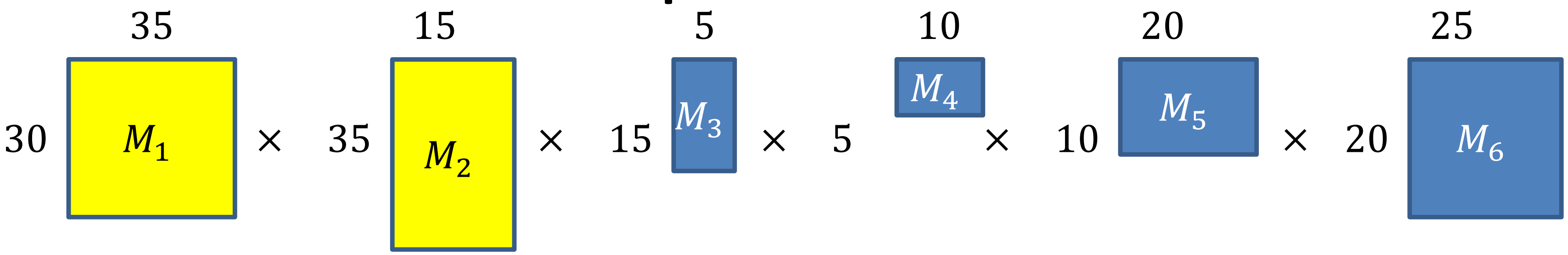


$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$	1	2	3	4	5	6	$i =$
	0						1
		0					2
			0				3
				0			4
					0		5
						0	6

2. Select a good order for solving subproblems



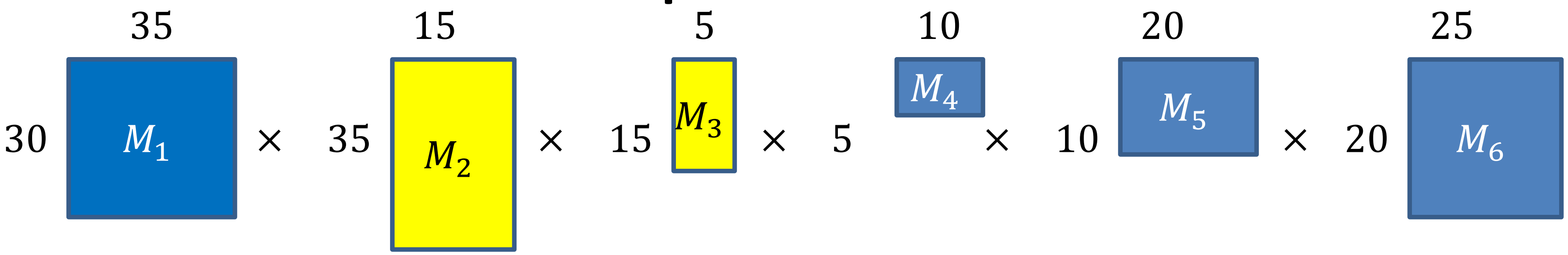
$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$ 1	2	3	4	5	6	$i =$
0	15750					1
	0					2
		0				3
			0			4
				0		5
					0	6

$$Best(1,2) = \min \begin{cases} Best(1,1) + Best(2,2) + r_1 r_2 c_2 \end{cases}$$

2. Select a good order for solving subproblems



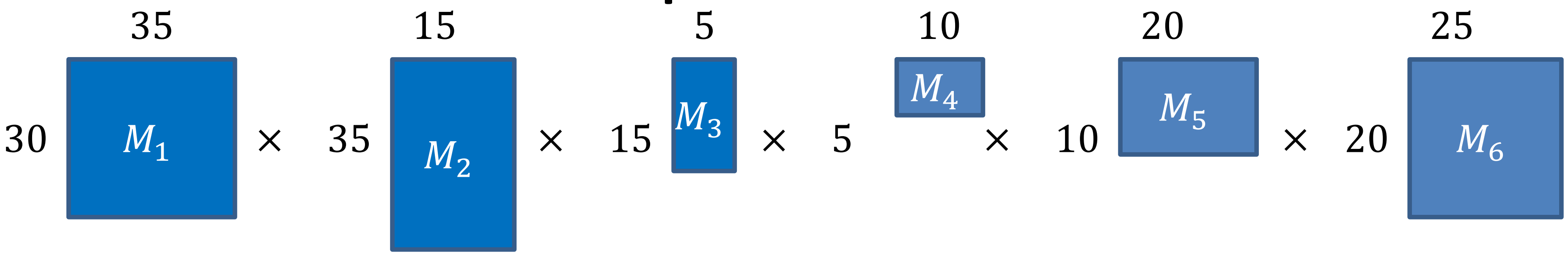
$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

i \ $j =$	1	2	3	4	5	6
1	0	15750				
2		0	2625			
3			0			
4				0		
5					0	
6						0

$$Best(2,3) = \min\left\{ Best(2,2) + Best(3,3) + r_2 r_3 c_3 \right.$$

2. Select a good order for solving subproblems

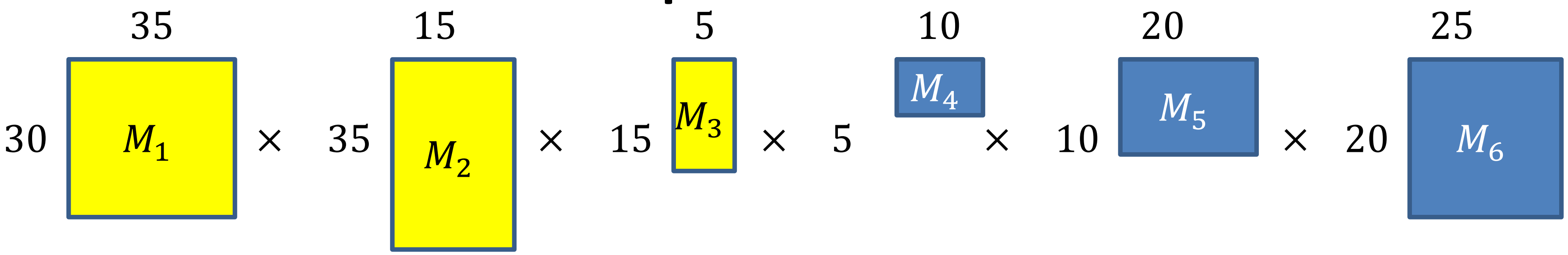


$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$	1	2	3	4	5	6	$i =$
	0	15750					1
		0	2625				2
			0	750			3
				0	1000		4
					0	5000	5
						0	6

2. Select a good order for solving subproblems



$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

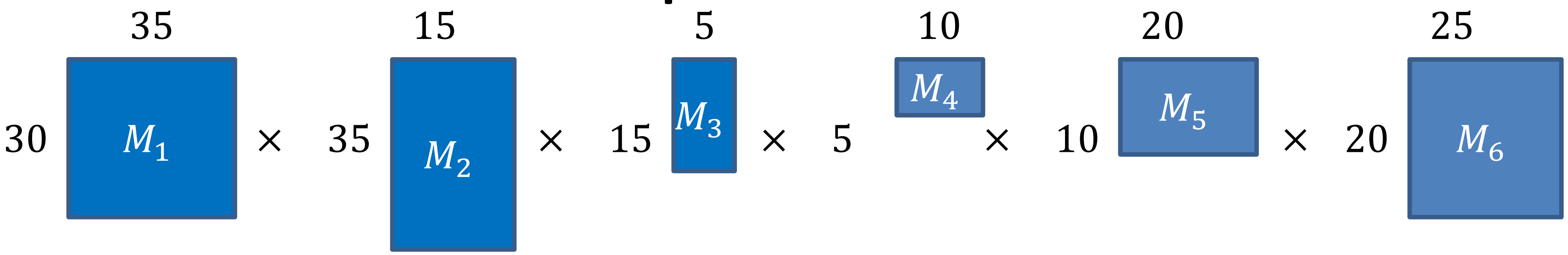
$$Best(i,i) = 0$$

$$r_1 r_2 c_3 = 30 \cdot 35 \cdot 5 = 5250$$
$$r_1 r_3 c_3 = 30 \cdot 15 \cdot 5 = 2250$$

$j =$ 1	2	3	4	5	6	$i =$
0	15750	7875				1
	0	2625				2
		0	750			3
			0	1000		4
				0	5000	5
					0	6

$$Best(1,3) = \min \begin{cases} \overbrace{Best(1,1)}^{0} + \overbrace{Best(2,3)}^{2625} + r_1 r_2 c_3 \\ \underbrace{Best(1,2)}_{15750} + \underbrace{Best(3,3)}_{0} + r_1 r_3 c_3 \end{cases}$$

2. Select a good order for solving subproblems



$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

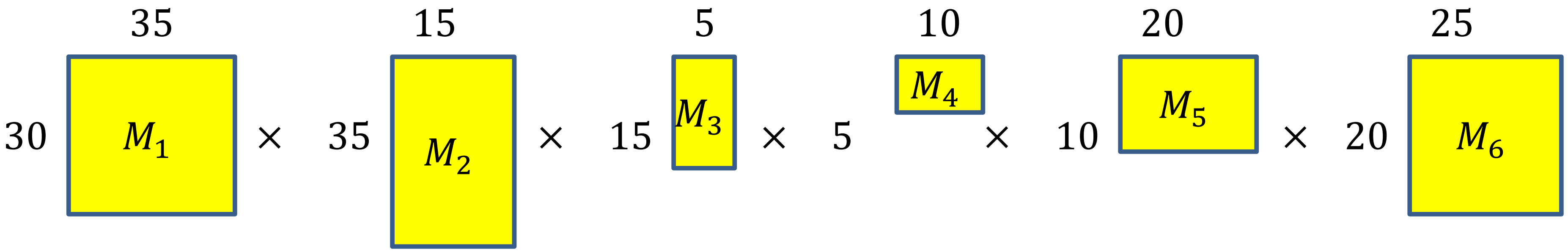
$$Best(i,i) = 0$$

i \ $j =$	1	2	3	4	5	6
1	0	15750	7875			
2		0	2625			
3			0	750		
4				0	1000	
5					0	5000
6						0

To find $Best(i,j)$: Need all preceding terms of row i and column j

Conclusion: solve in order of diagonal

Longest Common Subsequence



$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

i \ $j =$	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

$$Best(1,6) = \min \begin{cases} Best(1,1) + Best(2,6) + r_1 r_2 c_6 \\ Best(1,2) + Best(3,6) + r_1 r_3 c_6 \\ Best(1,3) + Best(4,6) + r_1 r_4 c_6 \\ Best(1,4) + Best(5,6) + r_1 r_5 c_6 \\ Best(1,5) + Best(6,6) + r_1 r_6 c_6 \end{cases}$$

Run Time

1. Initialize $Best[i,i]$ to be all 0s
2. Starting at the main diagonal, working to the upper-right, fill in each cell using:
   1. $Best[i,i] = 0$
   2. $Best[i,j] = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$

$\Theta(n^2)$ cells in the Array

$\Theta(n)$ options for each cell

$\Theta(n^3)$ overall run time

Backtrack to find the best order

"remember" which choice of k was the minimum at each cell

$$Best(i,j) = \min_{k=i}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

i \ $j =$	1	2	3	4	5	6
1	0	15750	7875 (1)	9375	11875	**15125** (3)
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500 (5)
5					0	5000
6						0

$$Best(1,6) = \min \begin{cases} Best(1,1) + Best(2,6) + r_1 r_2 c_6 \\ Best(1,2) + Best(3,6) + r_1 r_3 c_6 \\ Best(1,3) + Best(4,6) + r_1 r_4 c_6 \\ Best(1,4) + Best(5,6) + r_1 r_5 c_6 \\ Best(1,5) + Best(6,6) + r_1 r_6 c_6 \end{cases}$$

28

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify recursive structure of the problem
     - What is the “last thing” done?
  2. Select a good order for solving subproblems
     - “Top Down”: Solve each recursively
     - “Bottom Up”: Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory

Movie Time!

In Season 9 Episode 7 “The Slicer” of the hit 90s TV show *Seinfeld*, George discovers that, years prior, he had a heated argument with his new boss, Mr. Kruger. This argument ended in George throwing Mr. Kruger’s boombox into the ocean. How did George make this discovery?

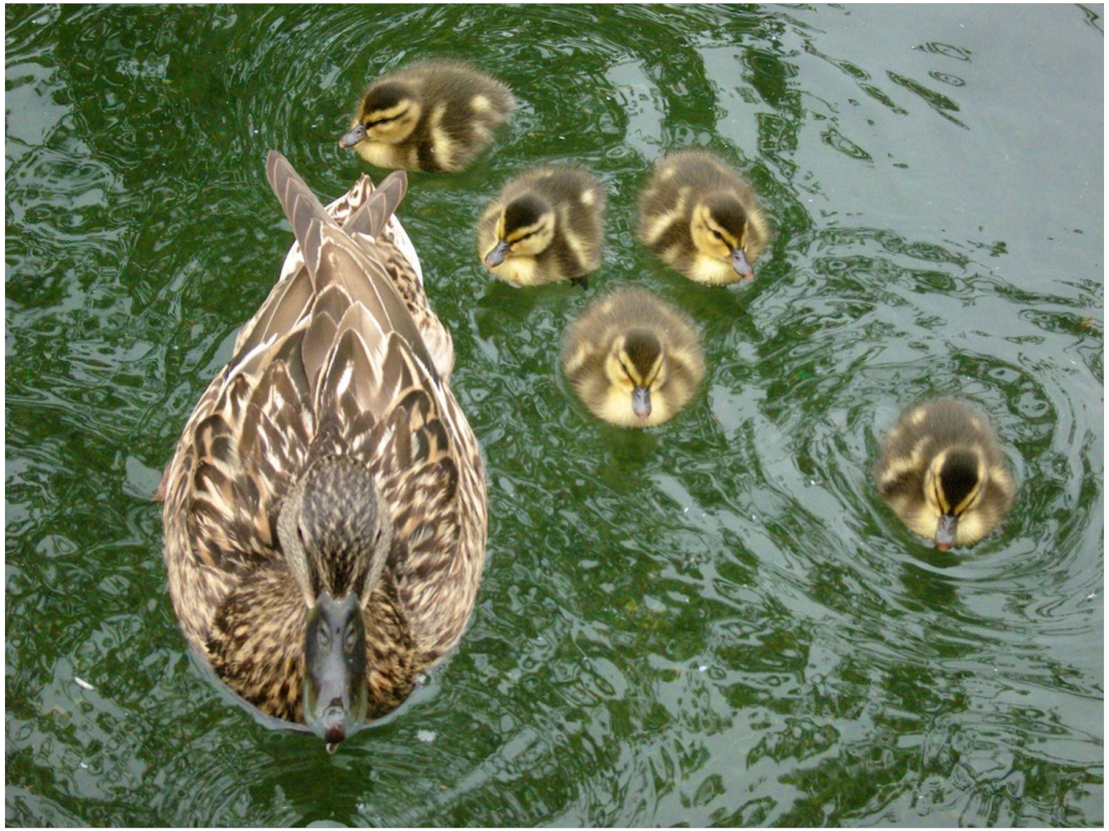
https://www.youtube.com/watch?v=pSB3HdmLcY4

Seam Carving

- Method for image resizing that doesn't scale/crop the image

Seam Carving

- Method for image resizing that doesn't scale/crop the image

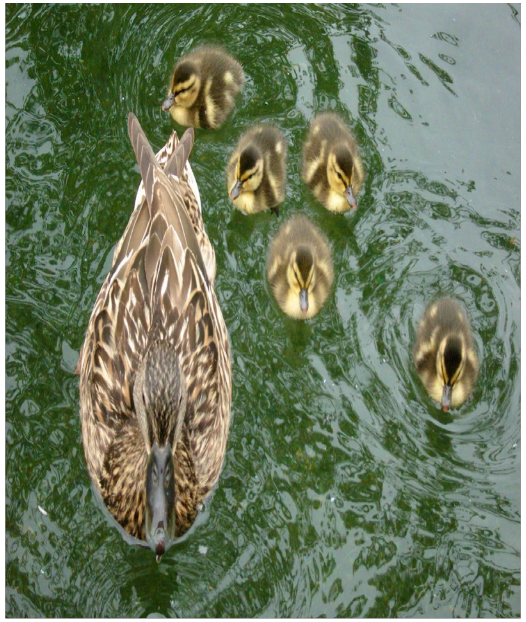


Seam Carving

- Method for image resizing that doesn't scale/crop the image

Cropped

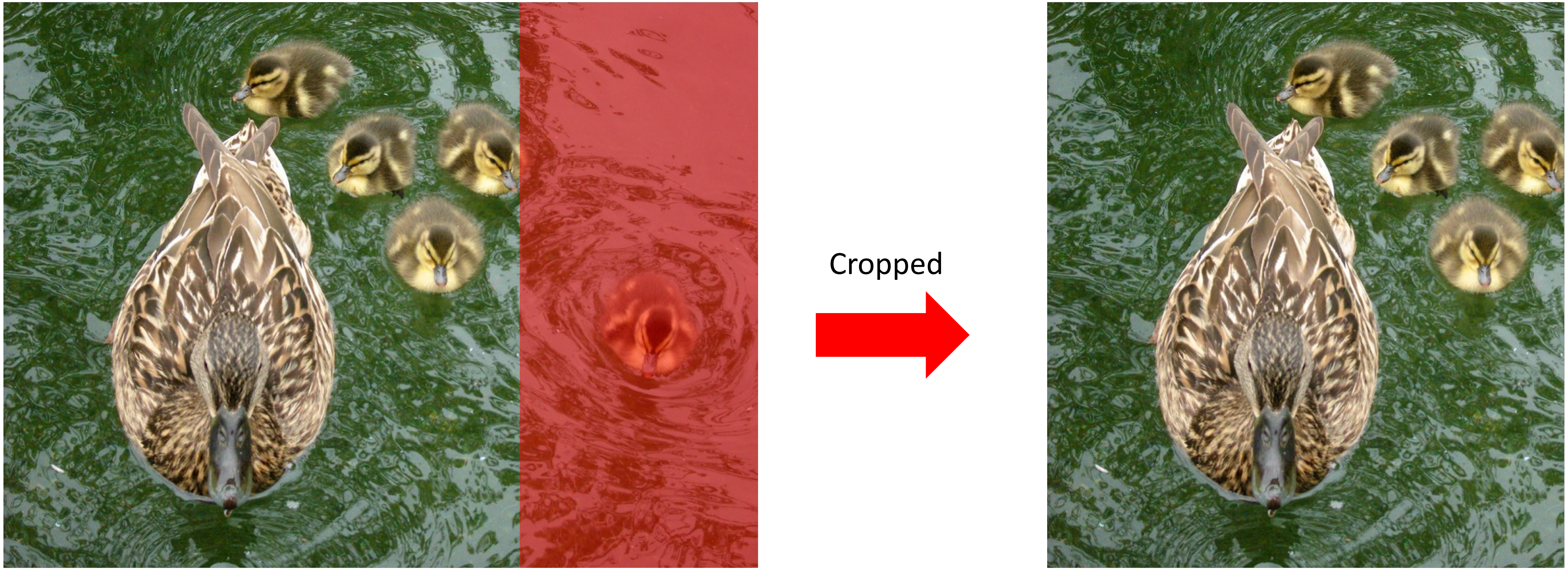
Scaled



Carved

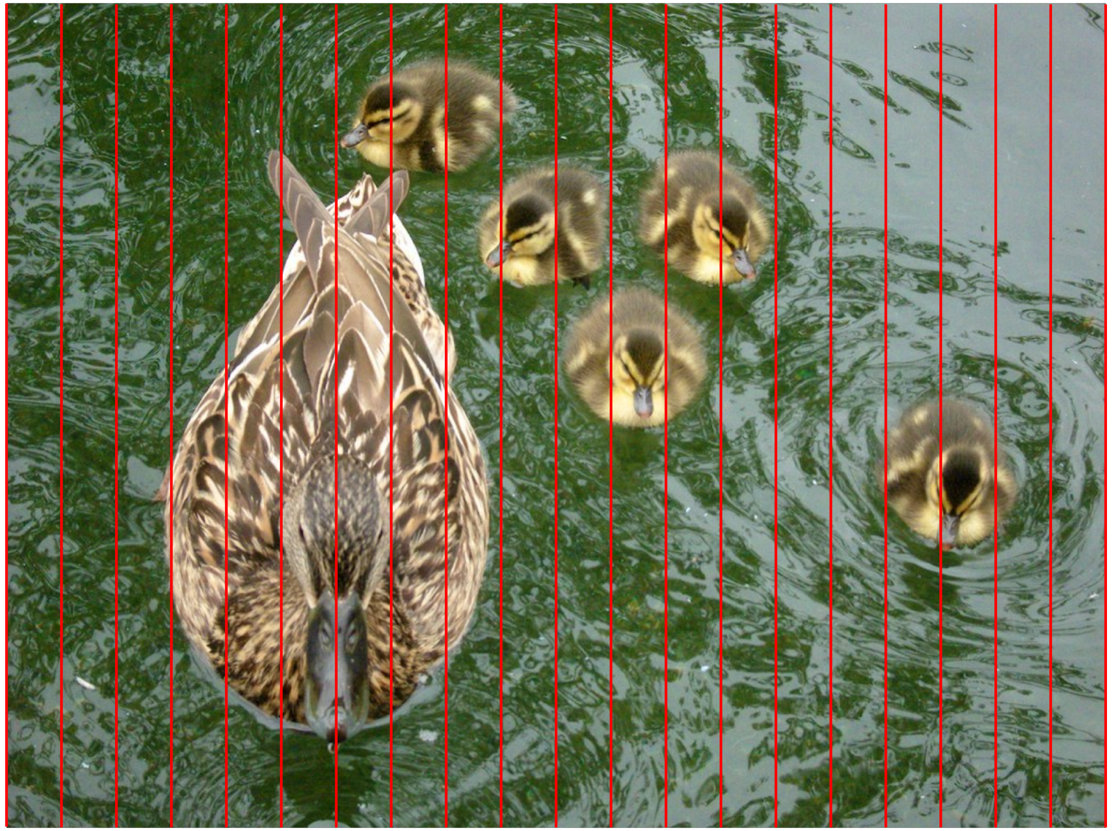
Cropping

- Removes a “block” of pixels

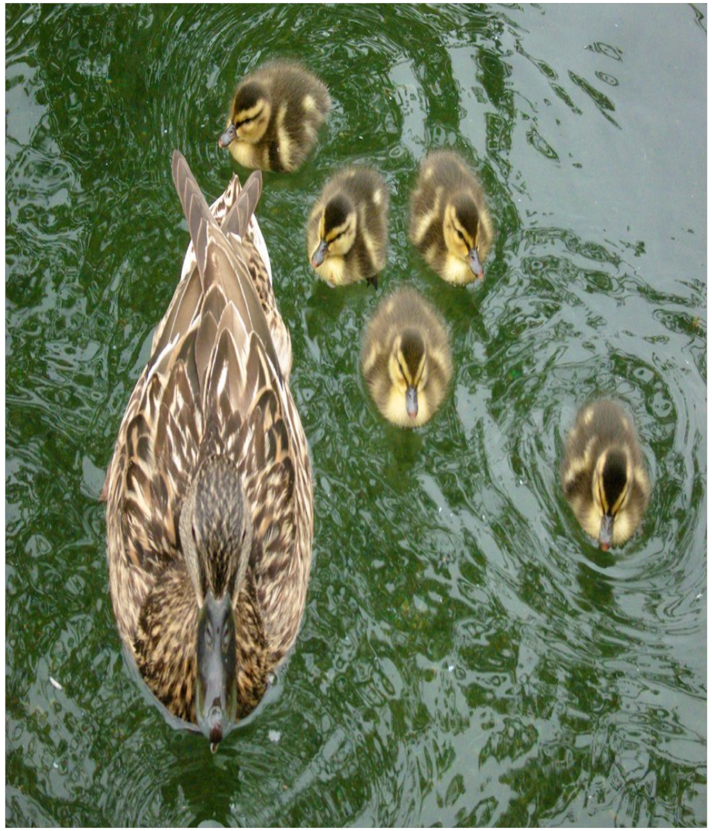


Scaling

- Removes “stripes” of pixels

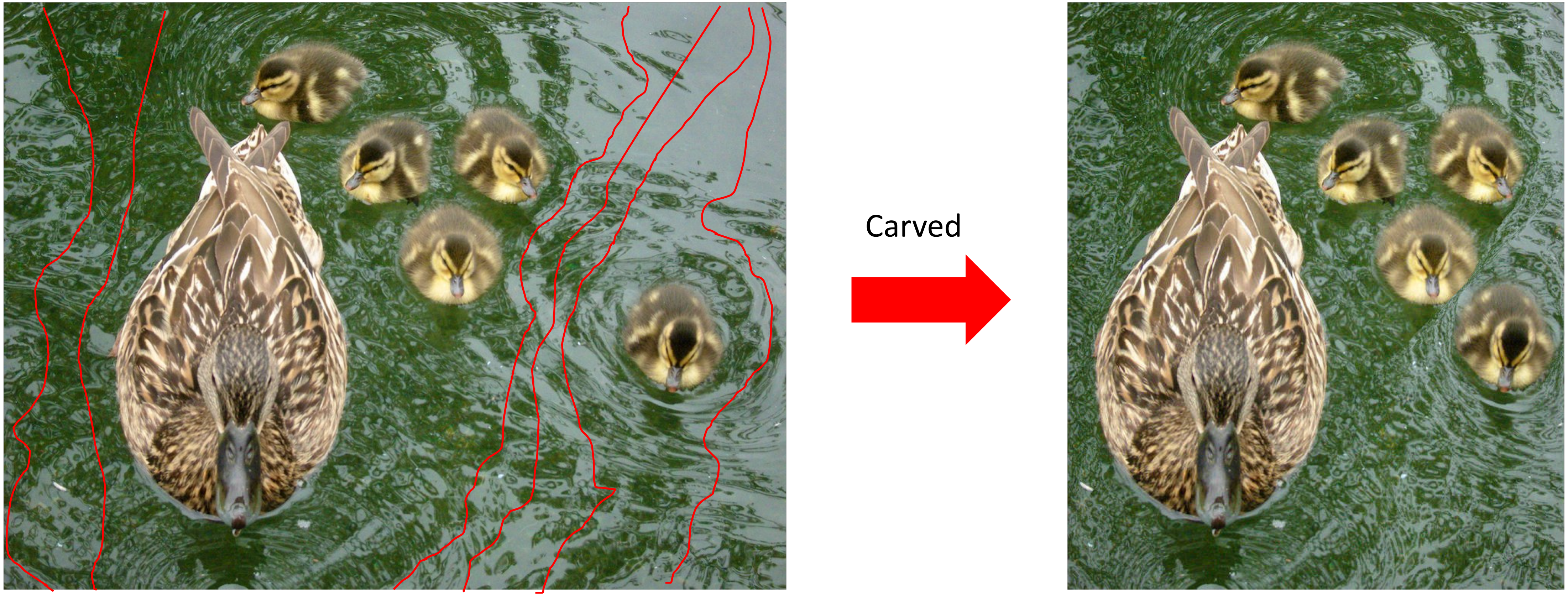


Scaled



Seam Carving

- Removes "least energy seam" of pixels
- http://rsizr.com/



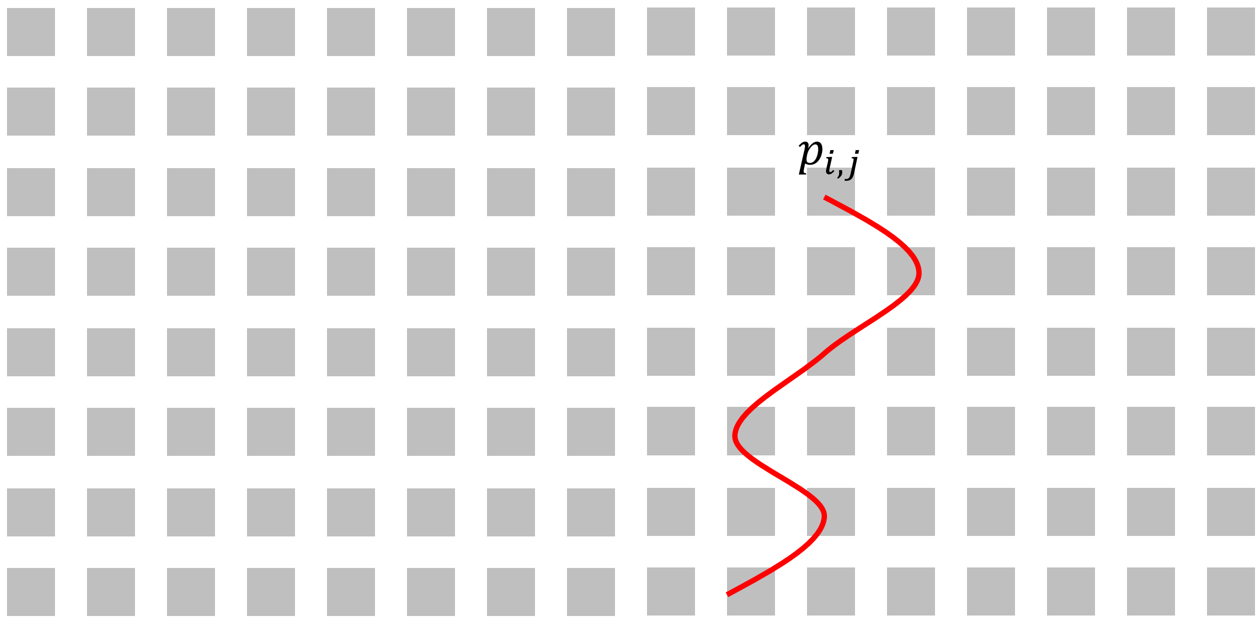
Seattle Skyline

Energy of a Seam

- Sum of the energies of each pixel
  - $e(p) =$ energy of pixel p
- Many choices
  - E.g.: change of gradient (how much the color of this pixel differs from its neighbors)
  - Particular choice doesn't matter, we use it as a "black box"

Identify Recursive Structure

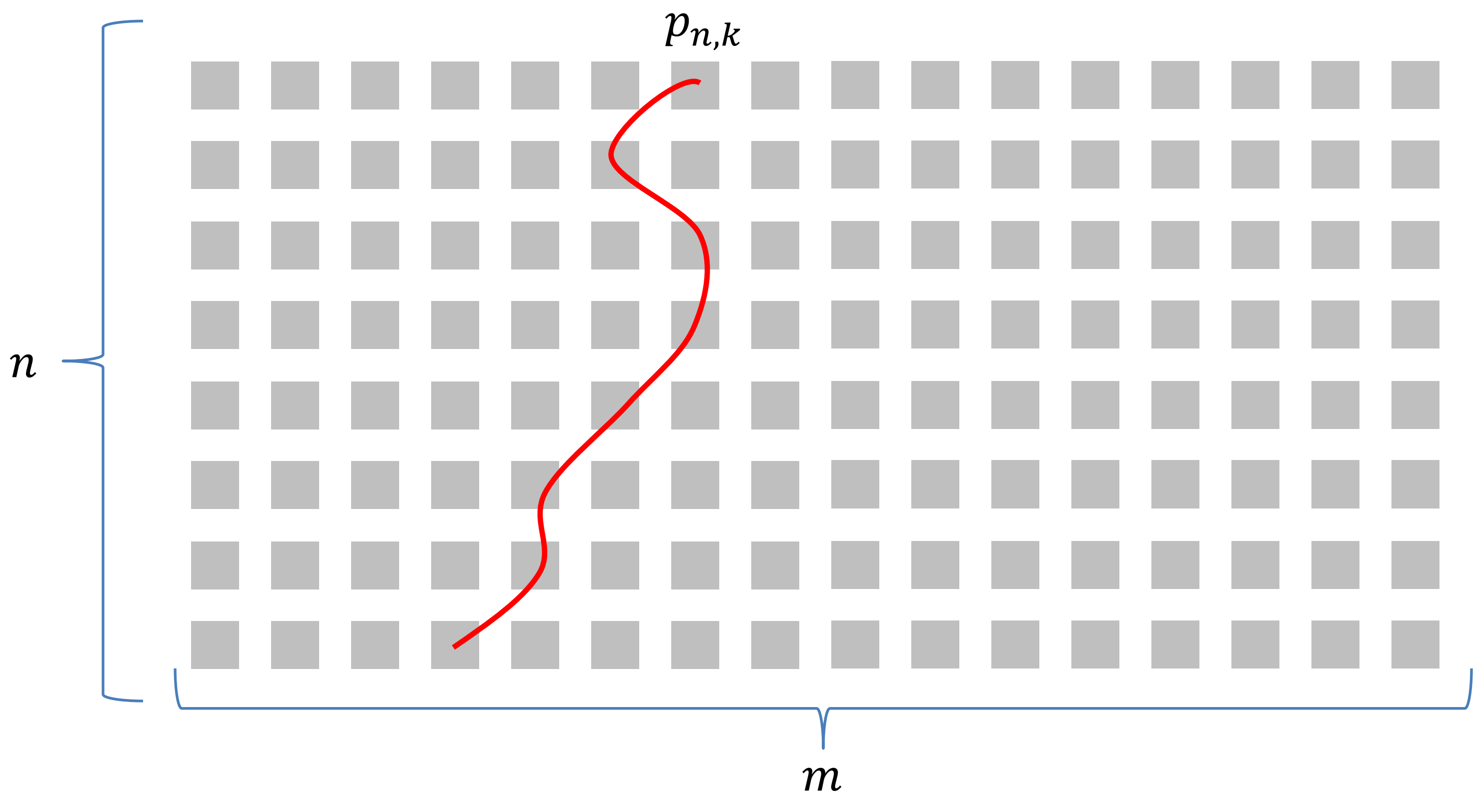
Let $S(i, j)$ = least energy seam from the bottom of the image up to pixel $p_{i,j}$



Finding the Least Energy Seam

Want the least energy seam going from bottom to top, so delete:

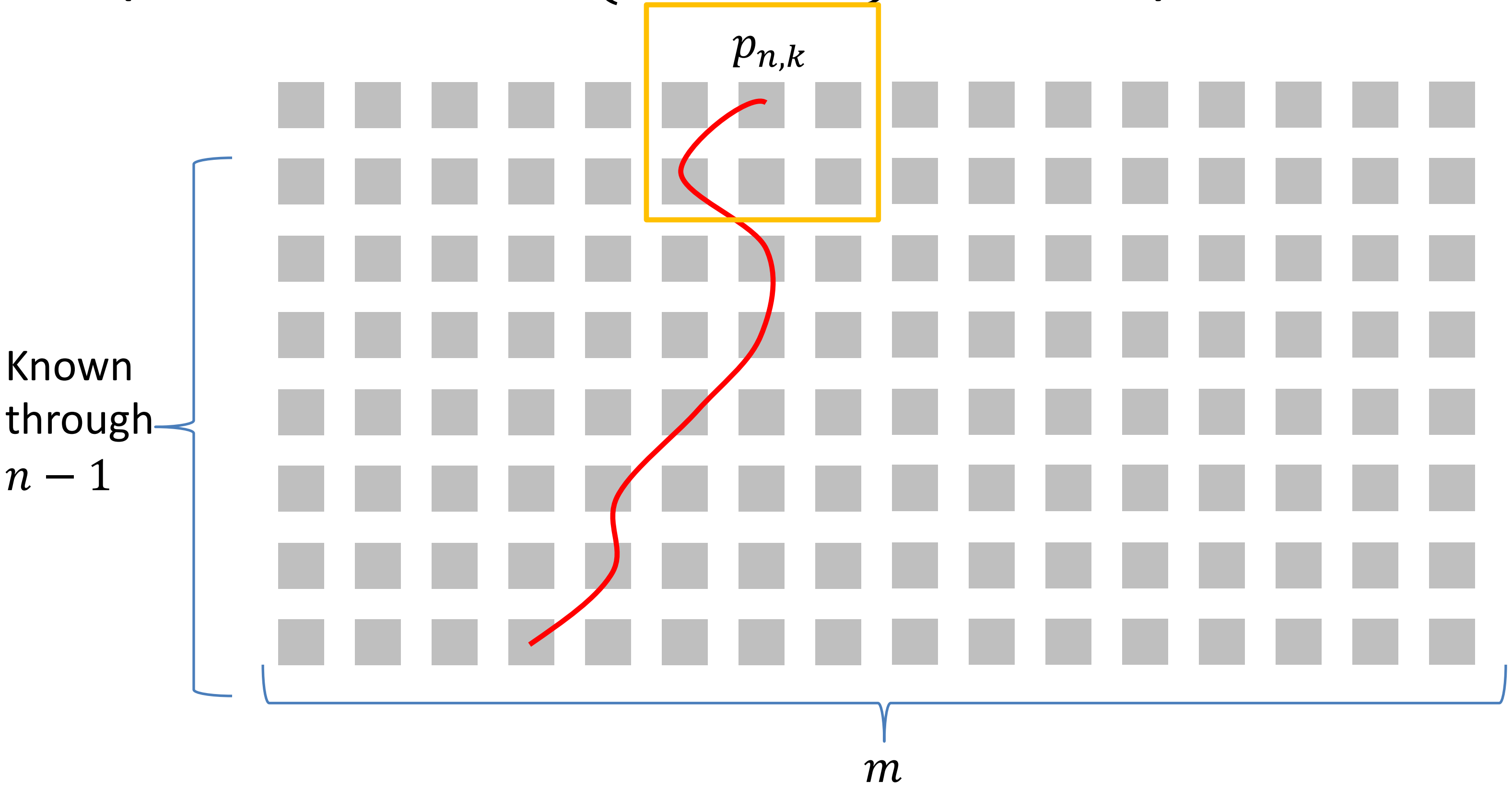
$$\min_{k=1}^{m}(S(n,k))$$



Computing $S(n, k)$

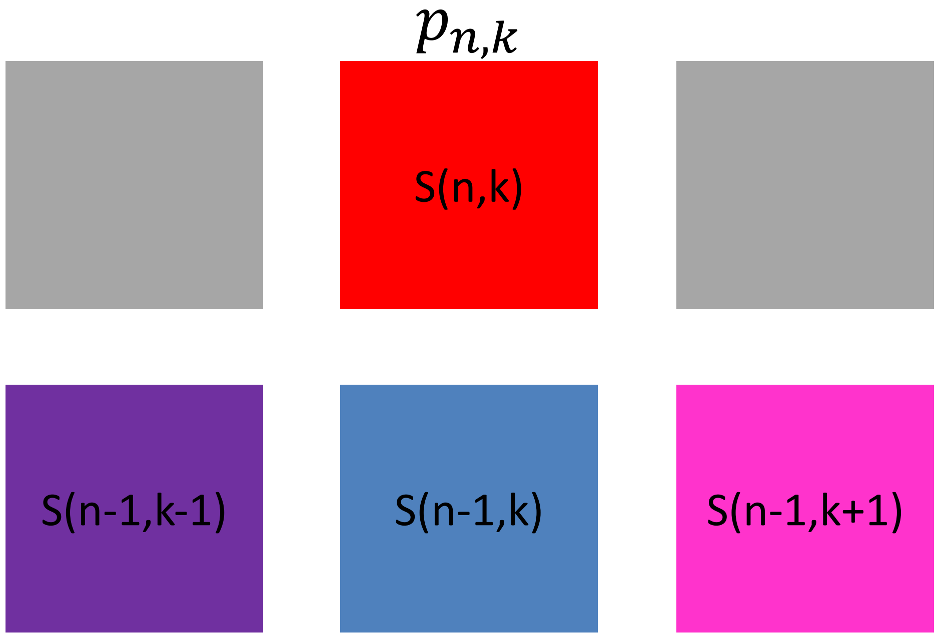
Assume we know the least energy seams for all of row $n-1$

(i.e. we know $S(n-1, \ell)$ for all ℓ)



Computing $S(n, k)$

Assume we know the least energy seams for all of row $n - 1$ (i.e. we know $S(n - 1, \ell)$ for all ℓ)



Repeated Seam Removal

Only need to update pixels dependent on the removed seam

$2n$ pixels change

$\Theta(2n)$ time to update pixels

$\Theta(n+m)$ time to find min+backtrack