DESIGN AND ANALYSIS OF ALGORITHMS

Grid Paths

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
http://www.cmi.ac.in/~madhavan

Grid Paths

* Roads arranged in a rectangular grid

* Can only go up or right
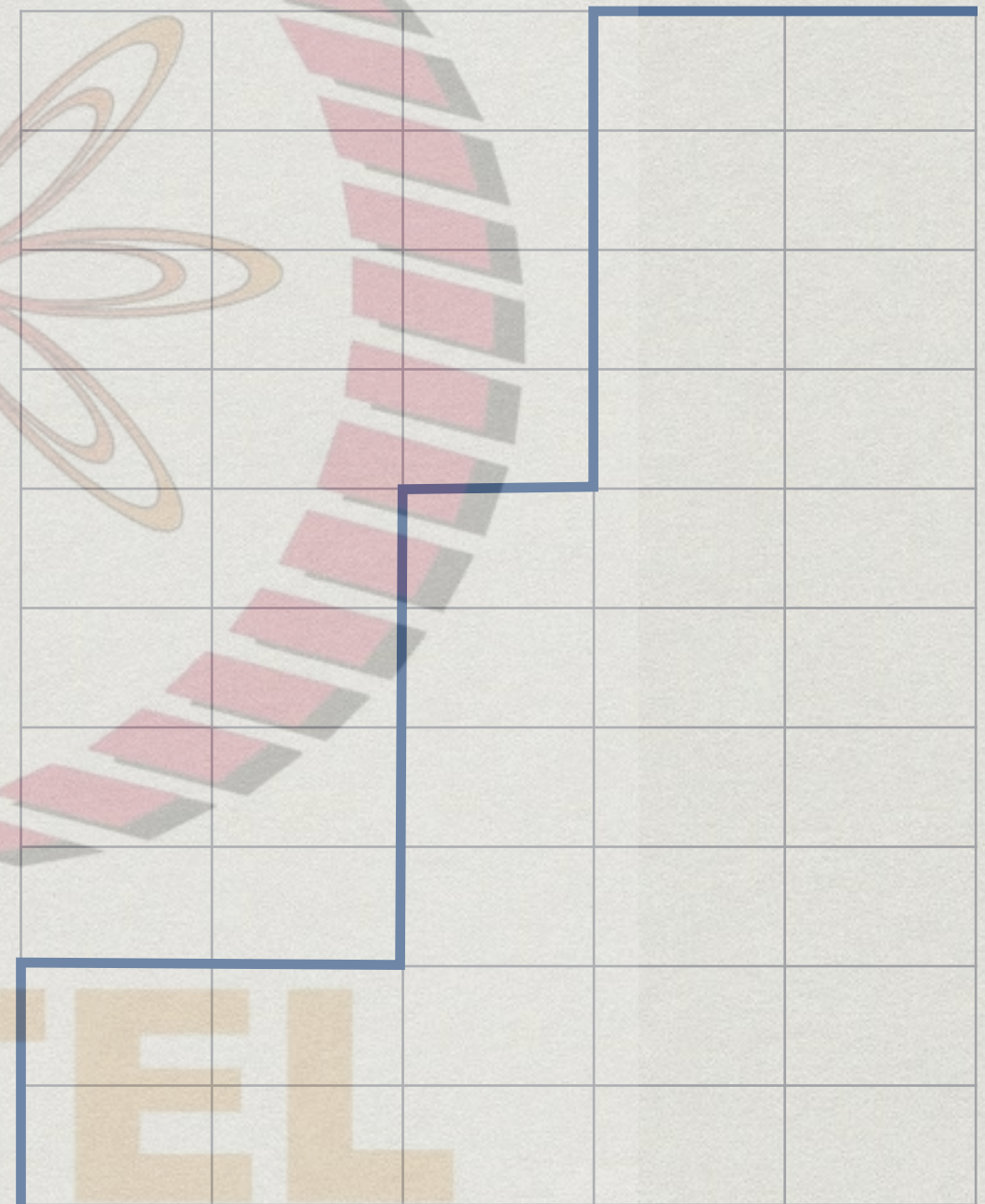
* How many different routes from (0,0) to (m,n)?

(0,0)

Grid Paths

(5,10)

* Roads arranged in a rectangular grid

* Can only go up or right

* How many different routes from (0,0) to (m,n)?

(0,0)

Grid Paths

(5,10)

* Roads arranged in a rectangular grid

* Can only go up or right
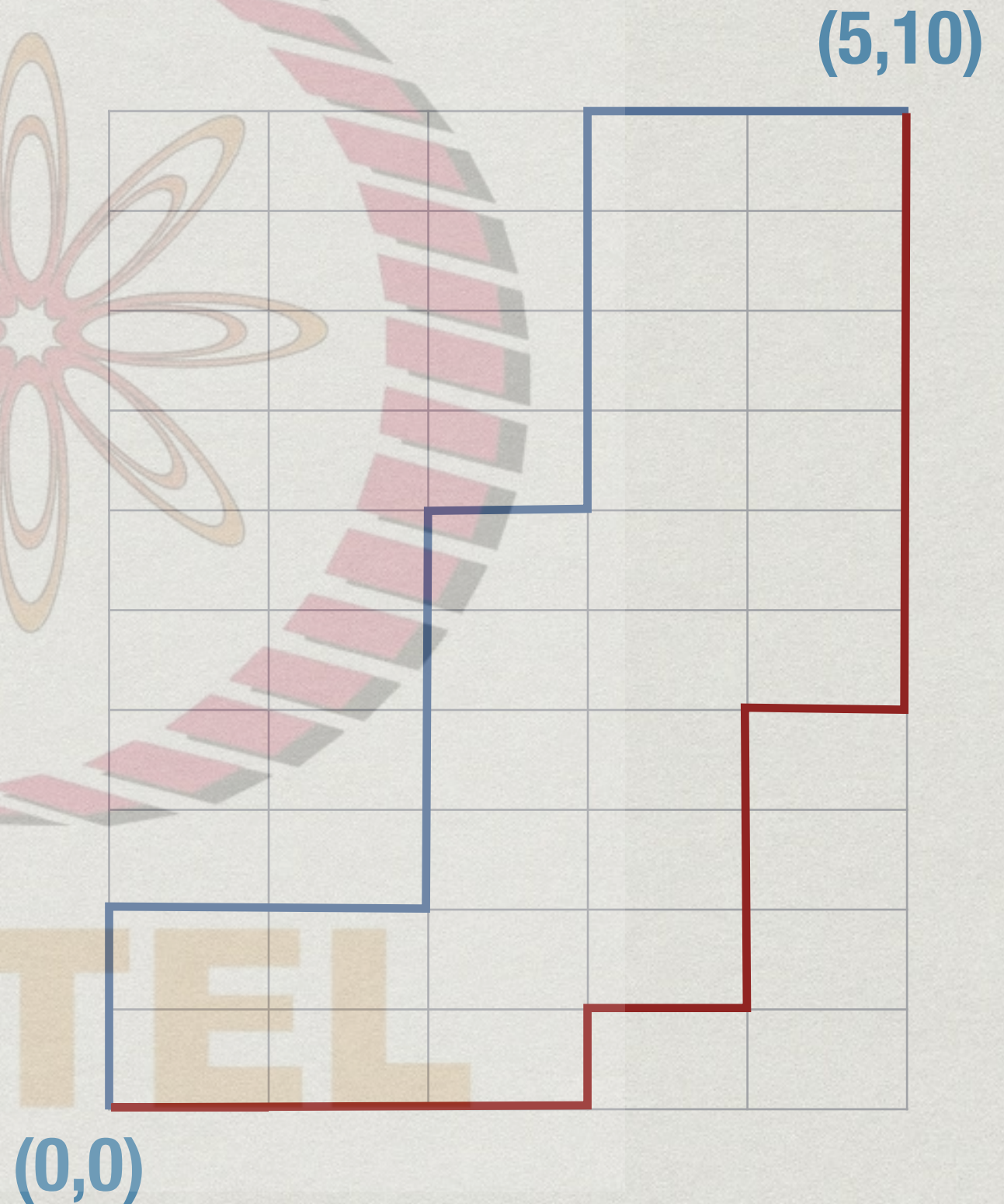
* How many different routes from (0,0) to (m,n)?

(0,0)

Grid Paths

(5,10)

* Roads arranged in a rectangular grid

* Can only go up or right
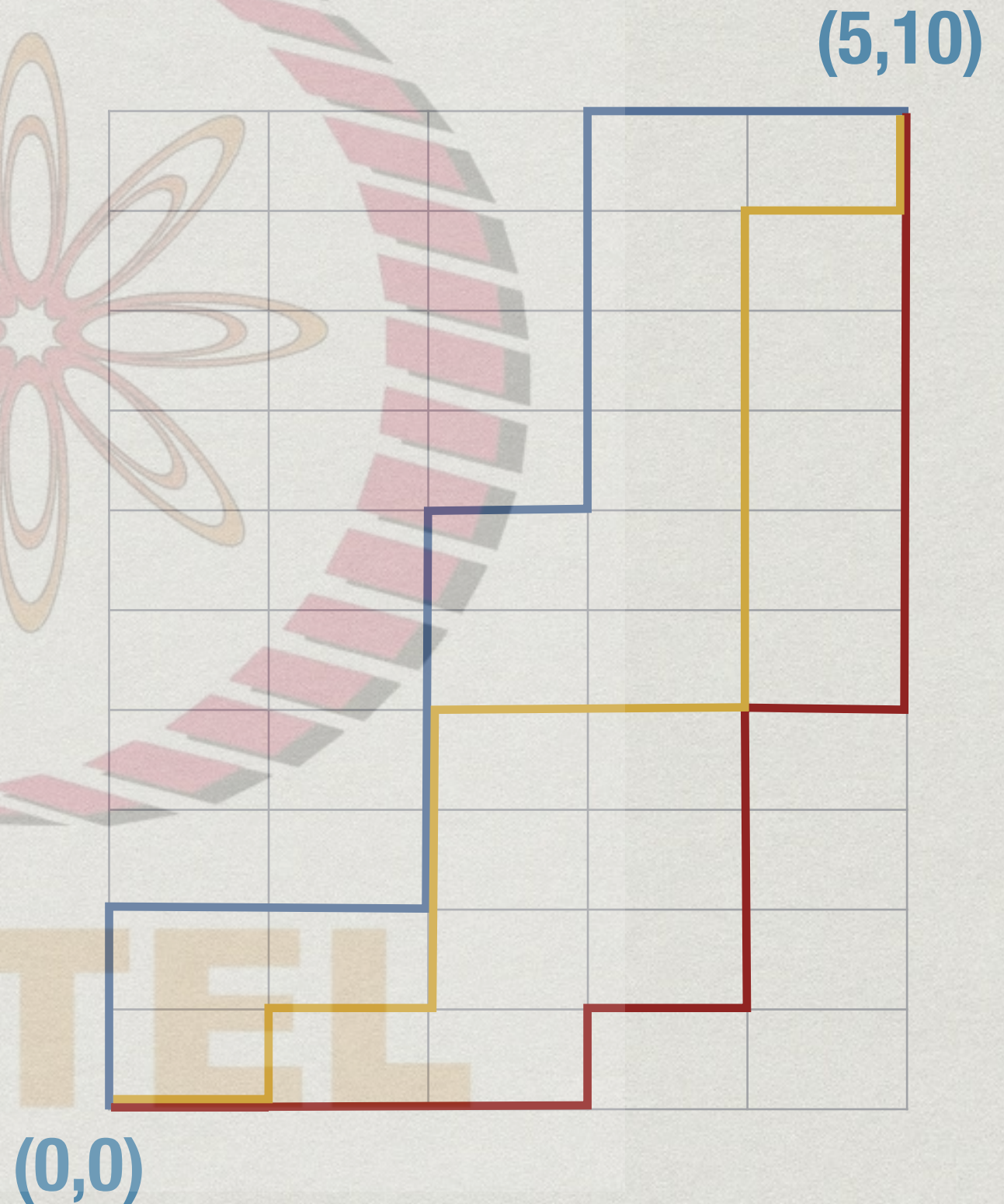
* How many different routes from (0,0) to (m,n)?

(0,0)

Combinatorial solution

* Every path from (0,0) to (5,10) has 15 segments

  * In general m+n segments from (0,0) to (m,n)

* Of these exactly 5 are right moves, 10 are up moves

* Fix the positions of the 5 right moves among the overall 15 positions

  * 15 choose 5 = (15!)/(10!)(5!) = 3003

  * Same as 15 choose 10: fix the 10 up moves

Holes

* What if an intersection is blocked?

  * (2,4), for example

* Paths through (2,4) need to be discarded

  * Two of our earlier examples are invalid paths
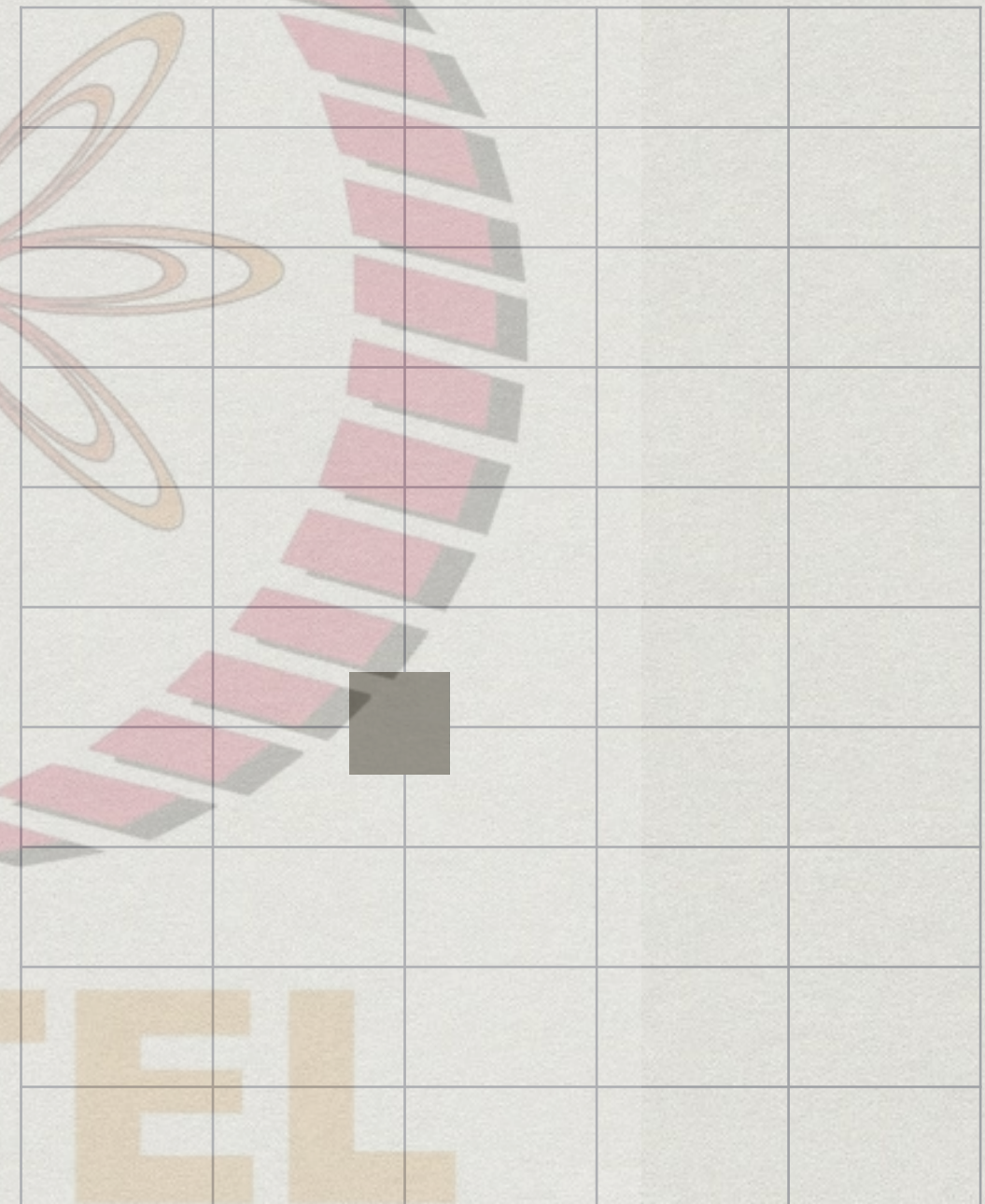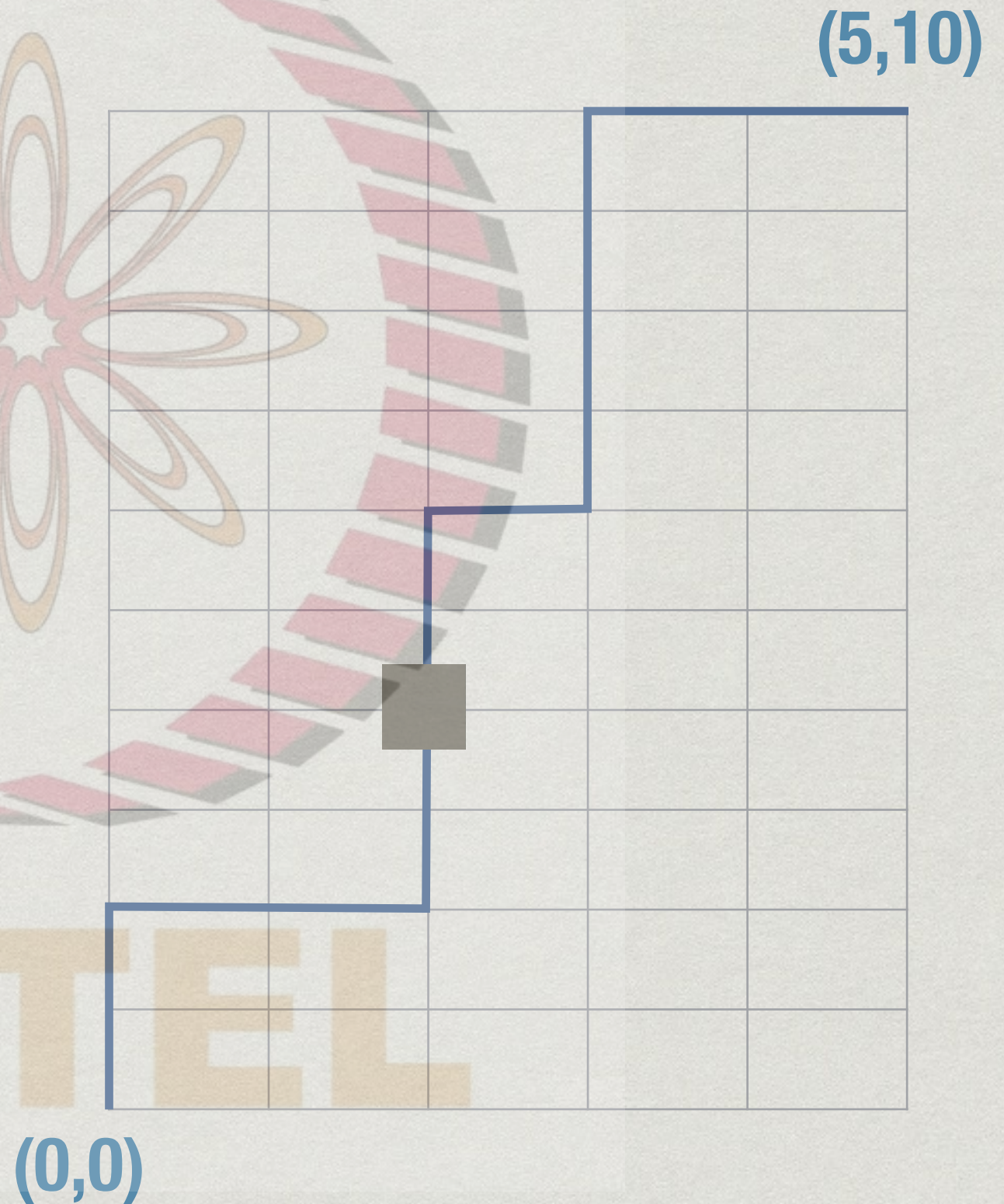
(0,0)

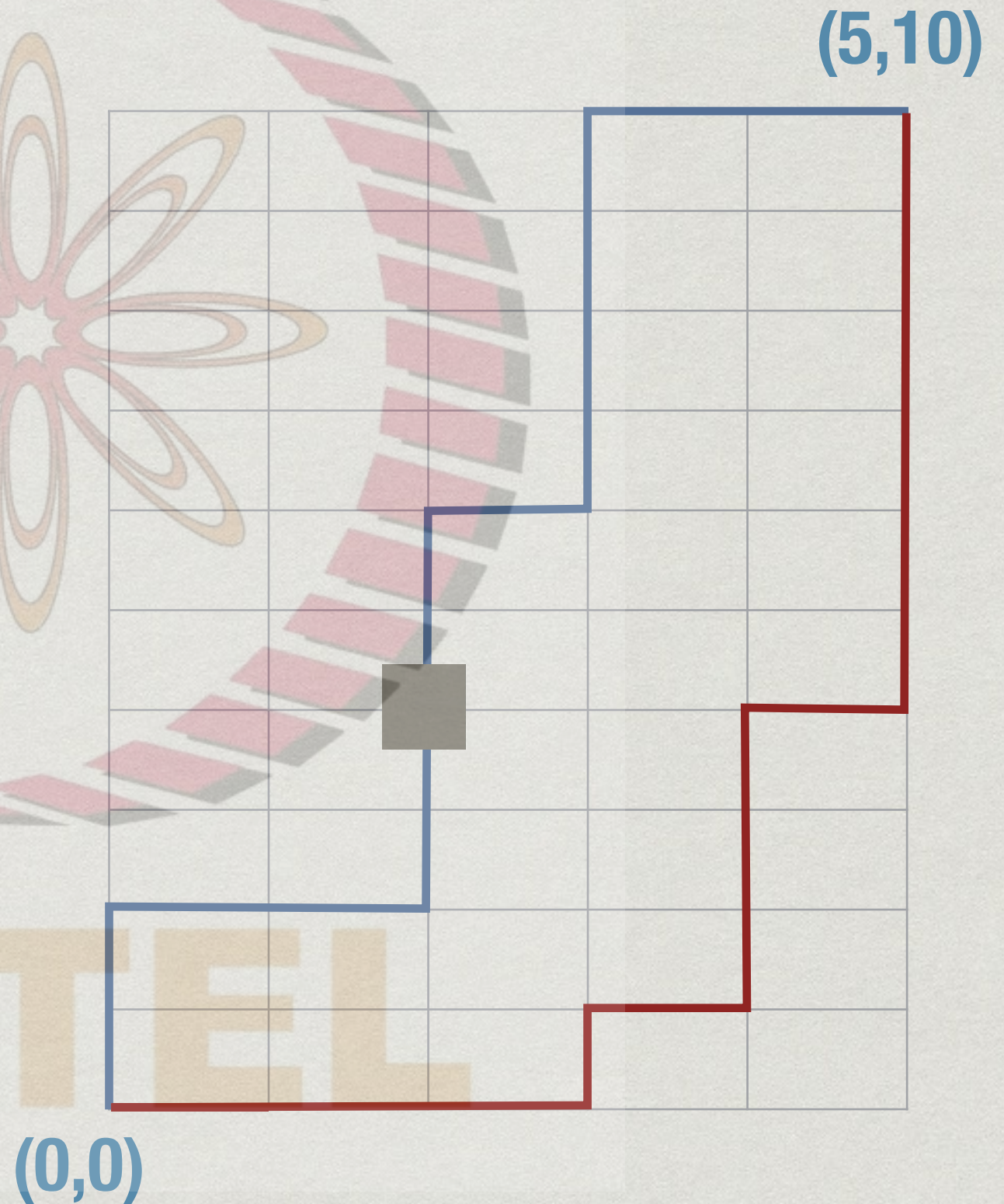# Holes

* What if an intersection is blocked?

    * (2,4), for example

* Paths through (2,4) need to be discarded

    * Two of our earlier examples are invalid paths

(5,10)

(0,0)

Holes

(5,10)

* What if an intersection is blocked?

  * (2,4), for example

* Paths through (2,4) need to be discarded

  * Two of our earlier examples are invalid paths
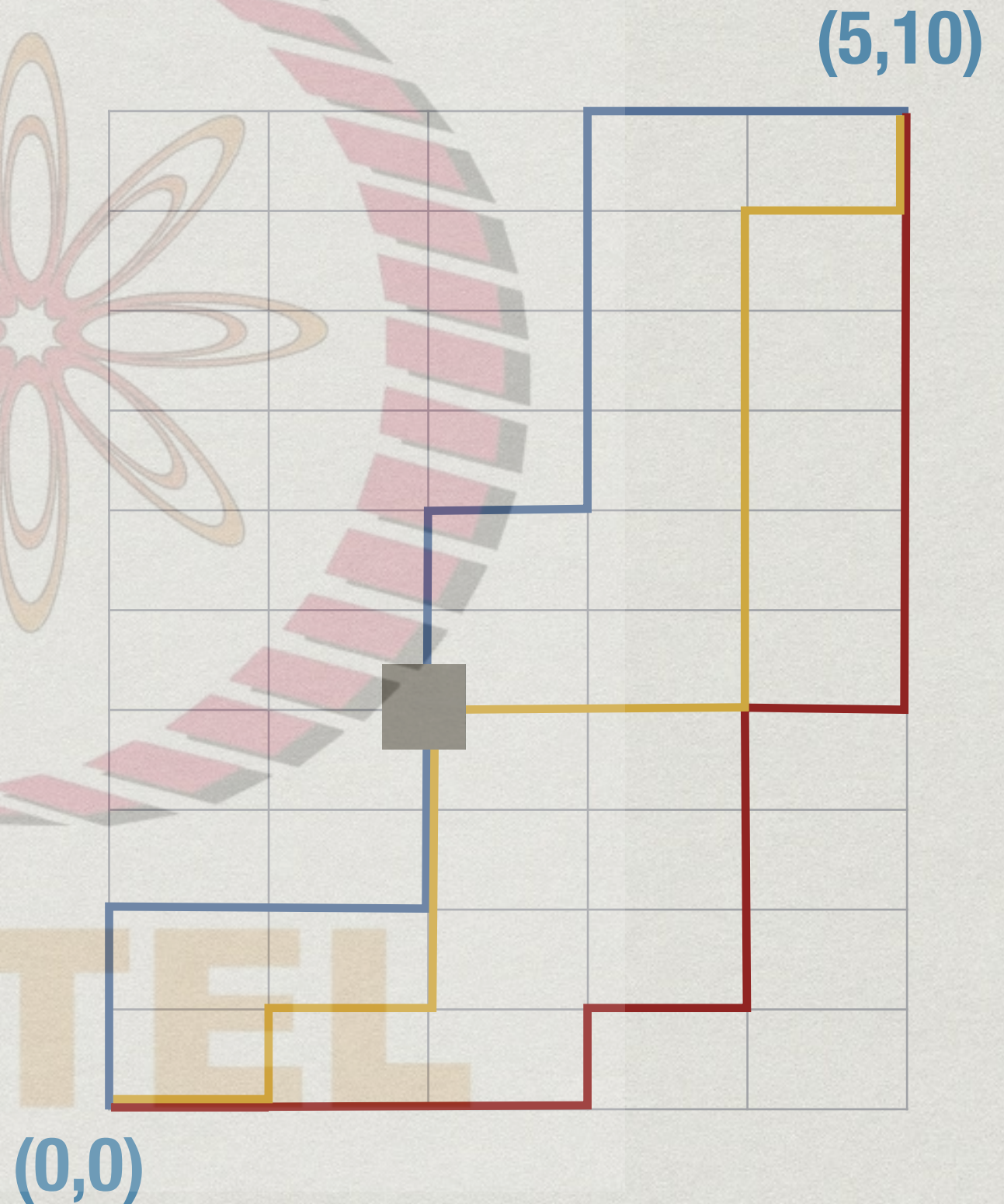
(0,0)

Holes

(5,10)

* What if an intersection is blocked?

  * (2,4), for example

* Paths through (2,4) need to be discarded

  * Two of our earlier examples are invalid paths

(0,0)

Combinatorial solution

* Every path through (2,4) goes from (0,0) to (2,4) and then from (2,4) to (5,10)

  * Count these separately:

    * (4+2) choose 2 = 15

    * (6+3) choose 3 = 84

  * Multiply to get all paths through (2,4): 1260

 * Subtract from 15 choose 5 = 3003 to get valid paths that avoid (2,4): 1743

Holes

(5,10)

* What if two intersections are blocked?

* Subtract paths through (2,4), (4,4)

  * Some paths are counted twice!

* Add back paths through both holes

* Inclusion-exclusion: messy

(0,0)
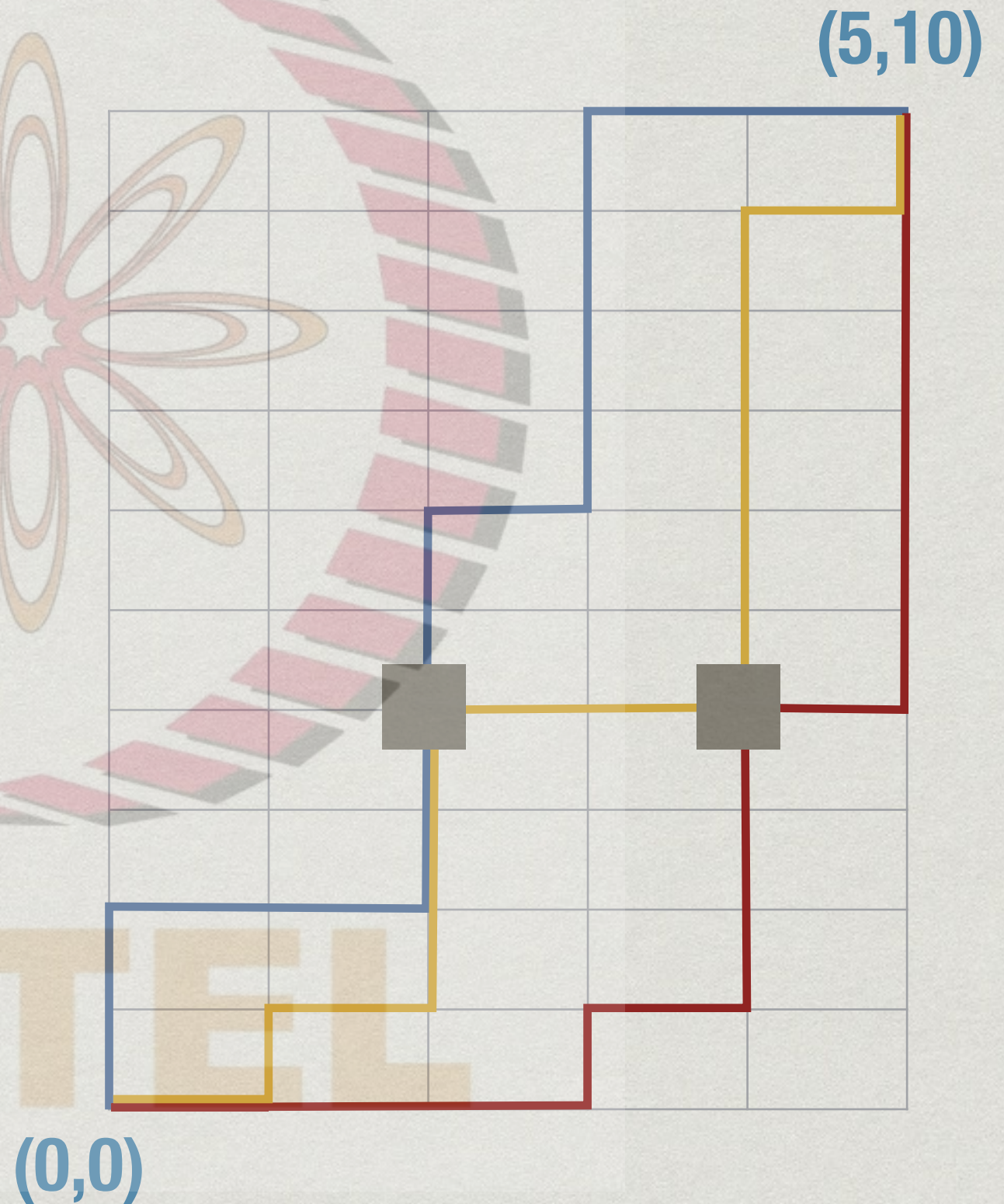
Inductive formulation

* How can a path reach (i,j)

  * Move up from (i,j-1)

  * Move right from (i-1,j)

* Every path to these neighbours extends in a unique way to (i,j)

(i-1,j) ——→ **(i,j)**

↑

(i,j-1)
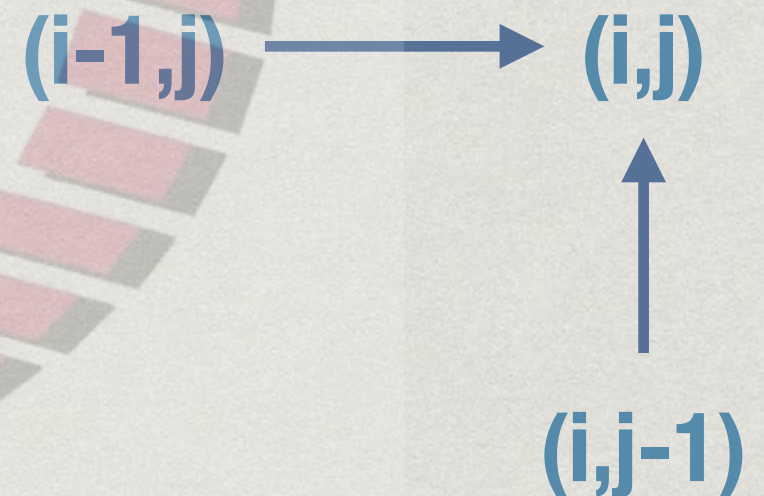
Inductive formulation

* Paths(i,j) : Number of paths from (0,0) to (i,j)

* Paths(i,j) = Paths(i-1,j) + Paths(i,j-1)

* Boundary cases

  * Paths(i,0) = Paths(i-1,0) # Bottom row

  * Paths(0,j) = Paths(0,j-1) # Left column

  * Paths(0,0) = 1 # Base case

Dealing with holes

* Paths(i,j) = 0, if there is a hole at (i,j)

* Paths(i,j) = Paths(i-1,j) + Paths(i,j-1), otherwise

* Boundary cases

    * Paths(i,0) = Paths(i-1,0) # Bottom row

    * Paths(0,j) = Paths(0,j-1) # Left column
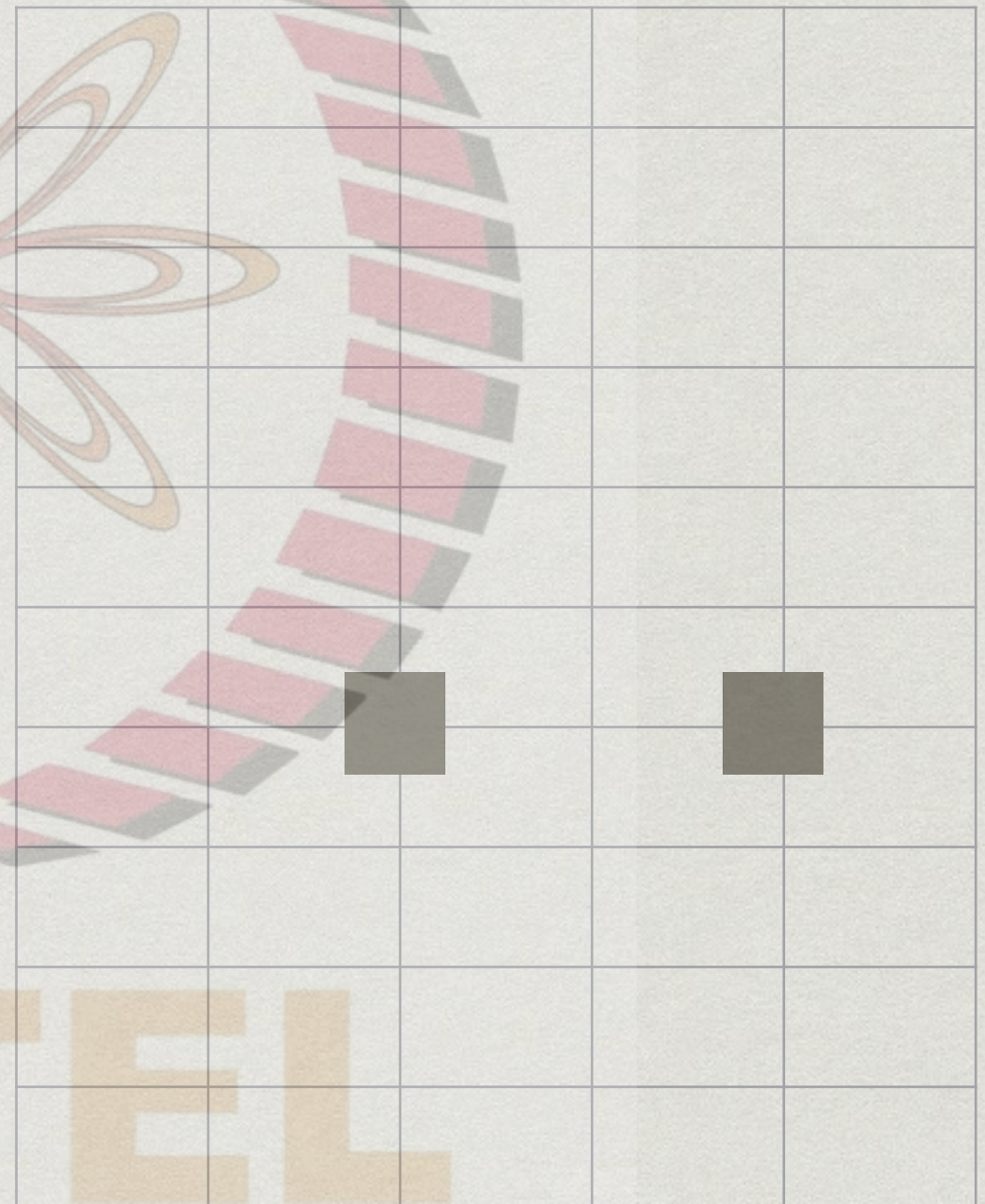
    * Paths(0,0) = 1 # Base case

Computing Paths(i,j)

* Naive recursion will recompute multiple times

    * Paths(5,10) requires Paths(4,10) and Paths(5,9)

    * Both Paths(4,10) and Paths(5,9) require Paths(4,9)

* Use memoization …

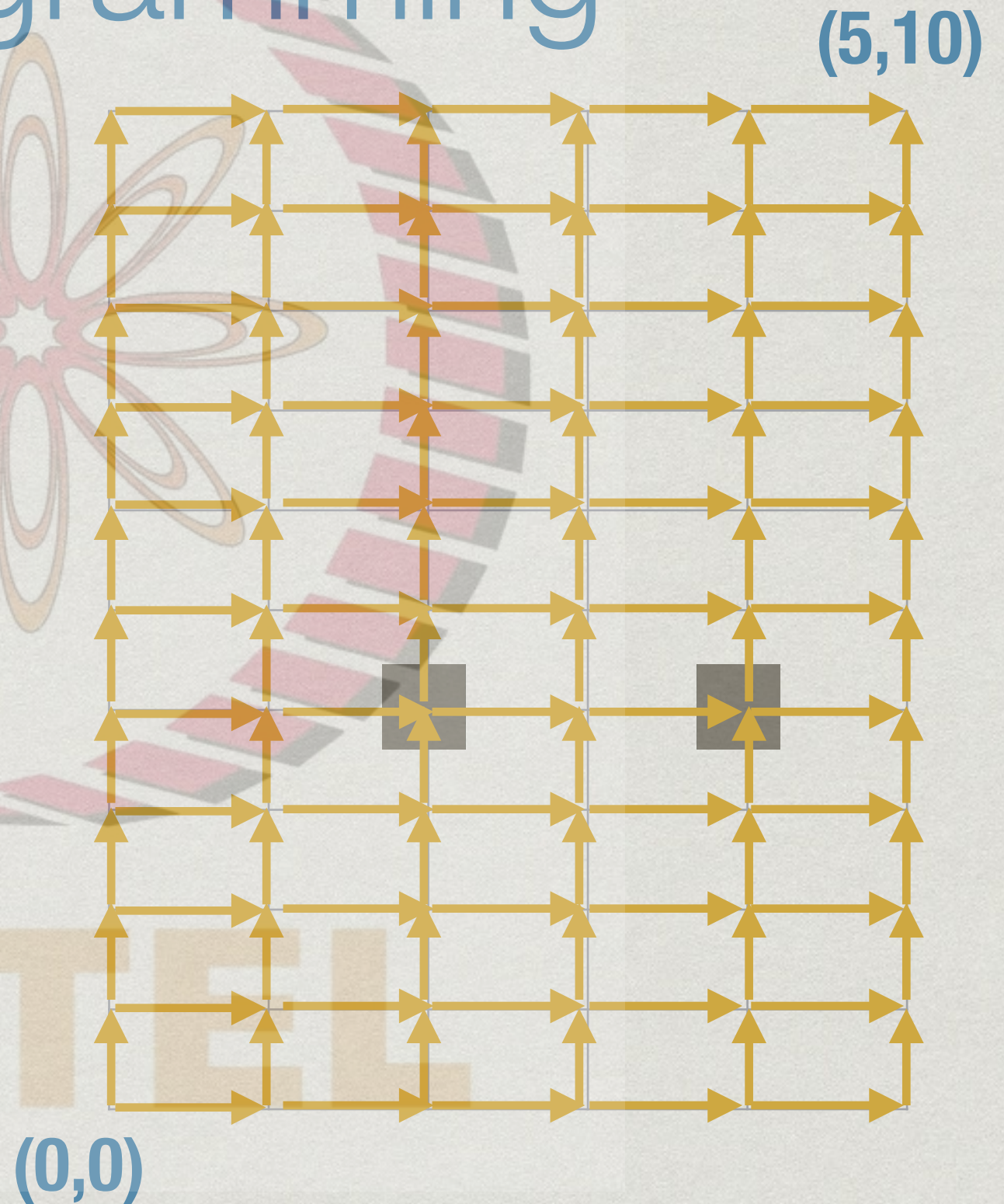* … or compute the subproblems directly in a suitable way

Dynamic programming

* Identify DAG structure

* Paths(0,0) has no dependencies

* Start at (0,0)

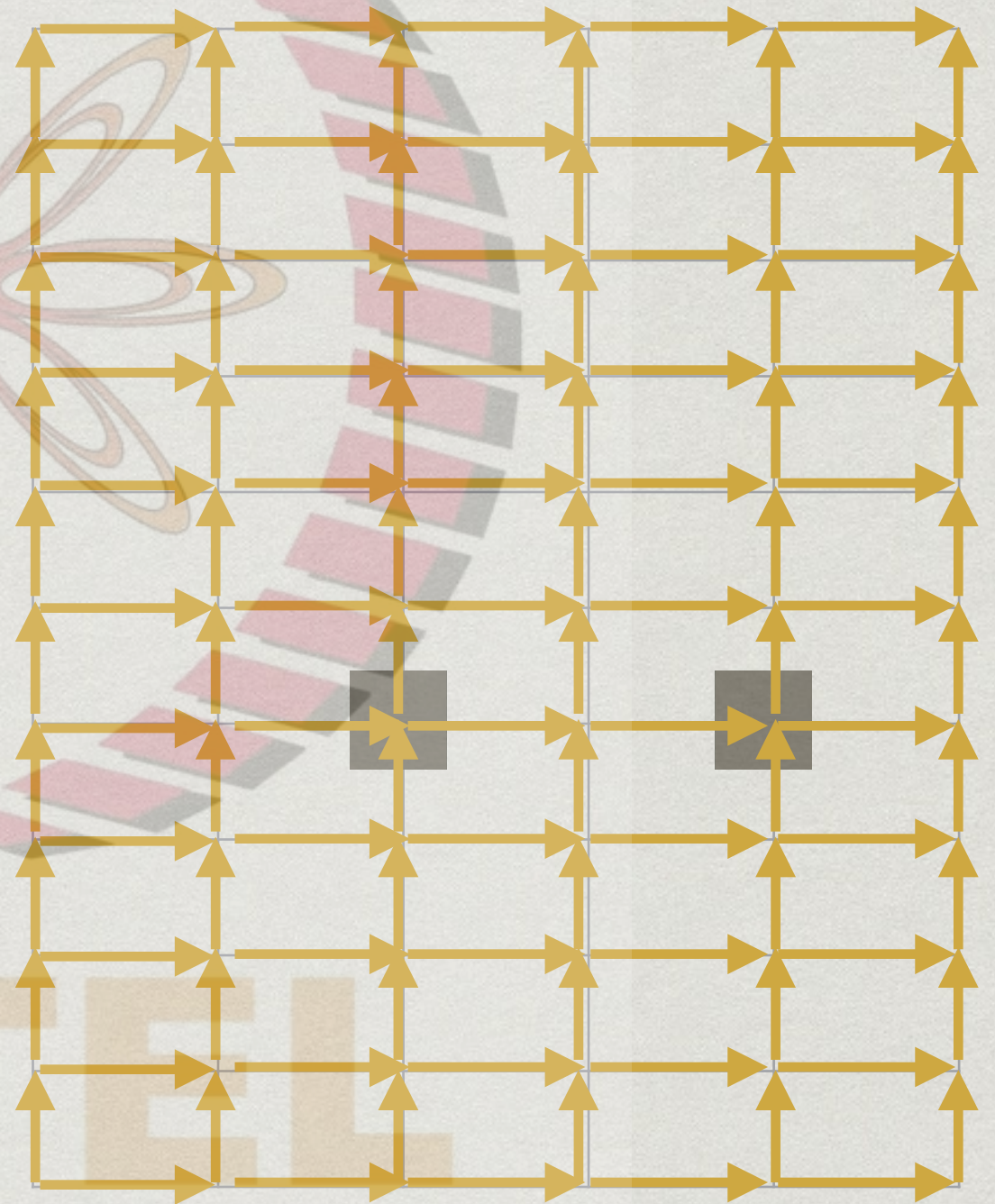# Dynamic programming

(5,10)

* Identify DAG structure

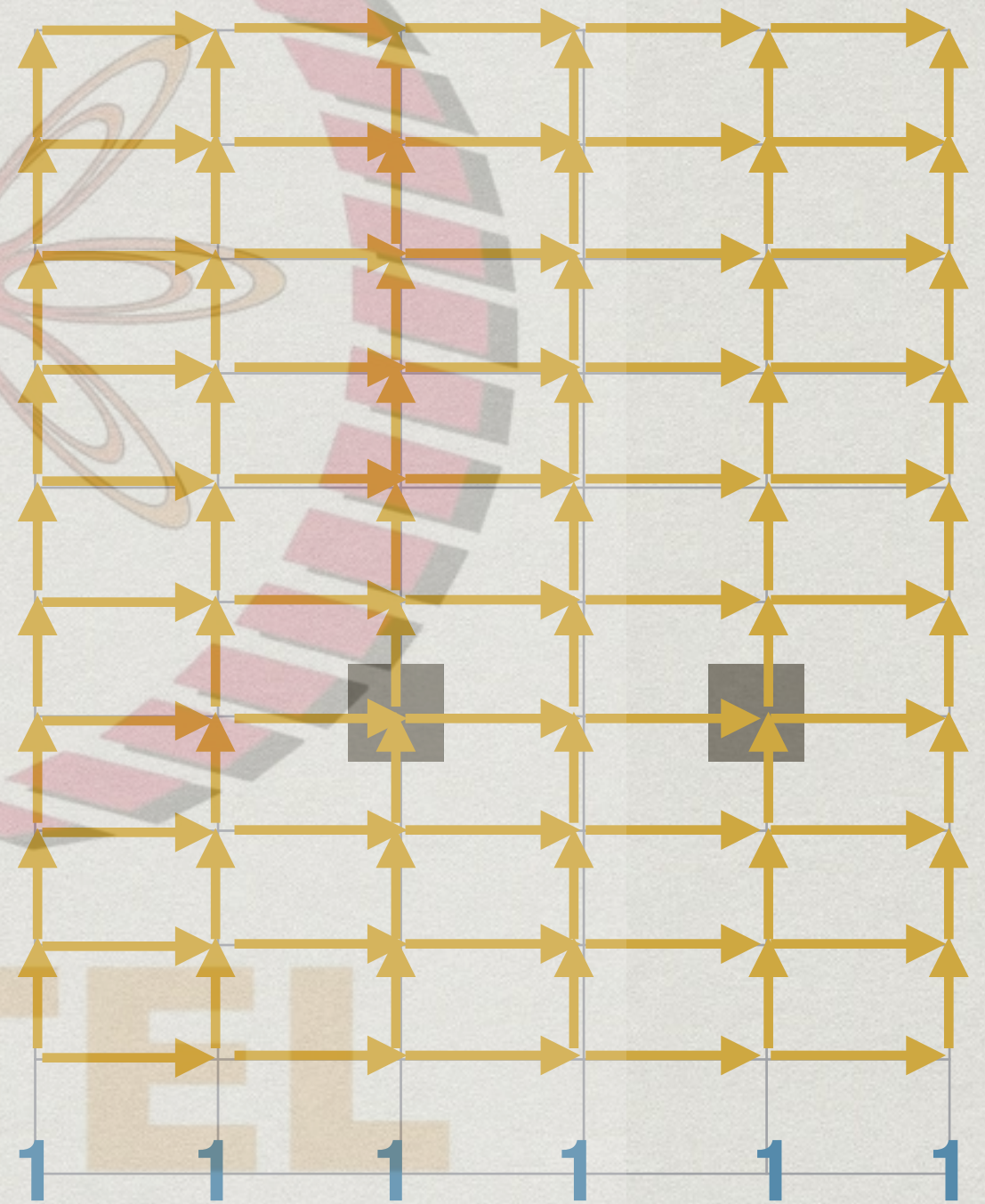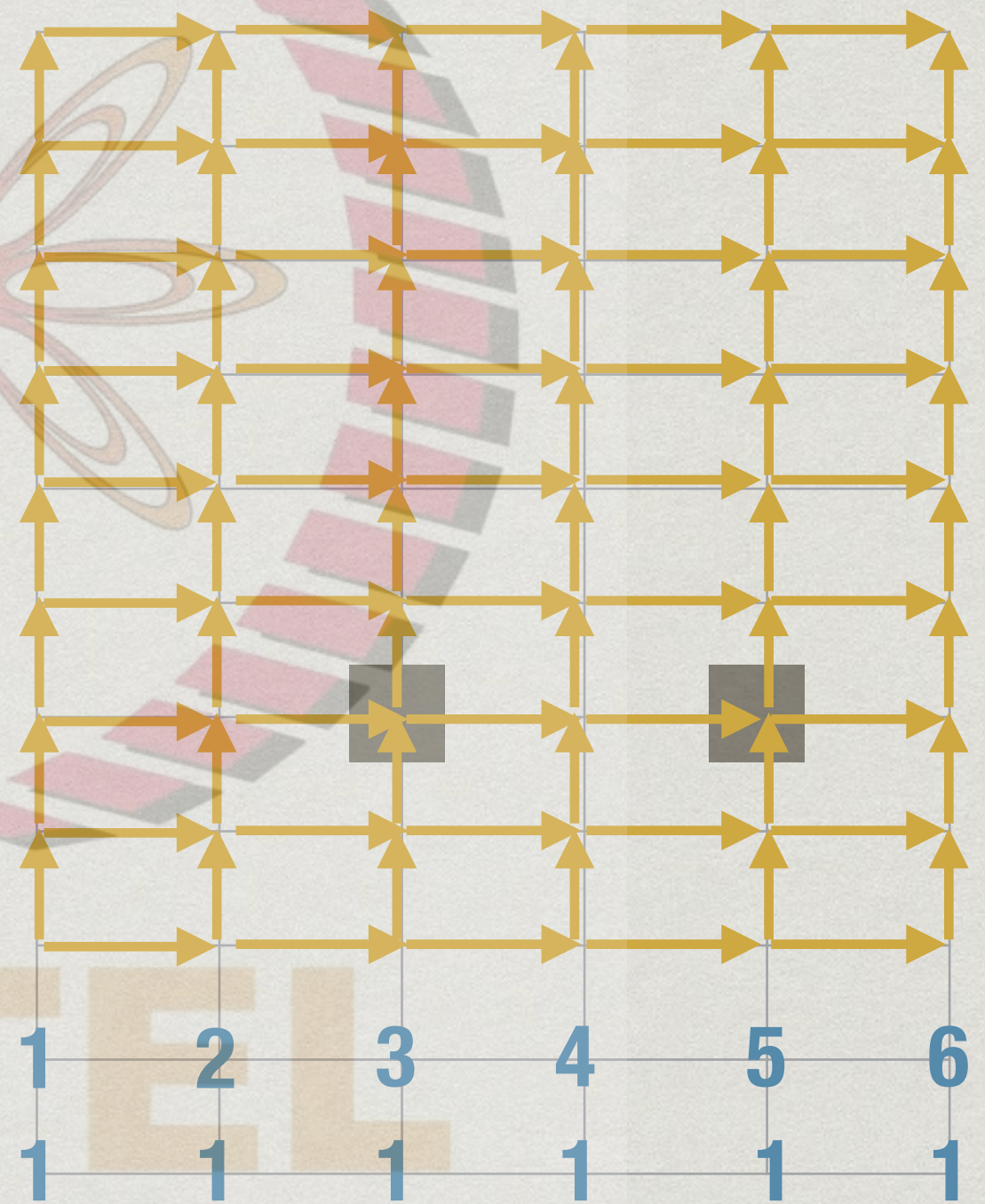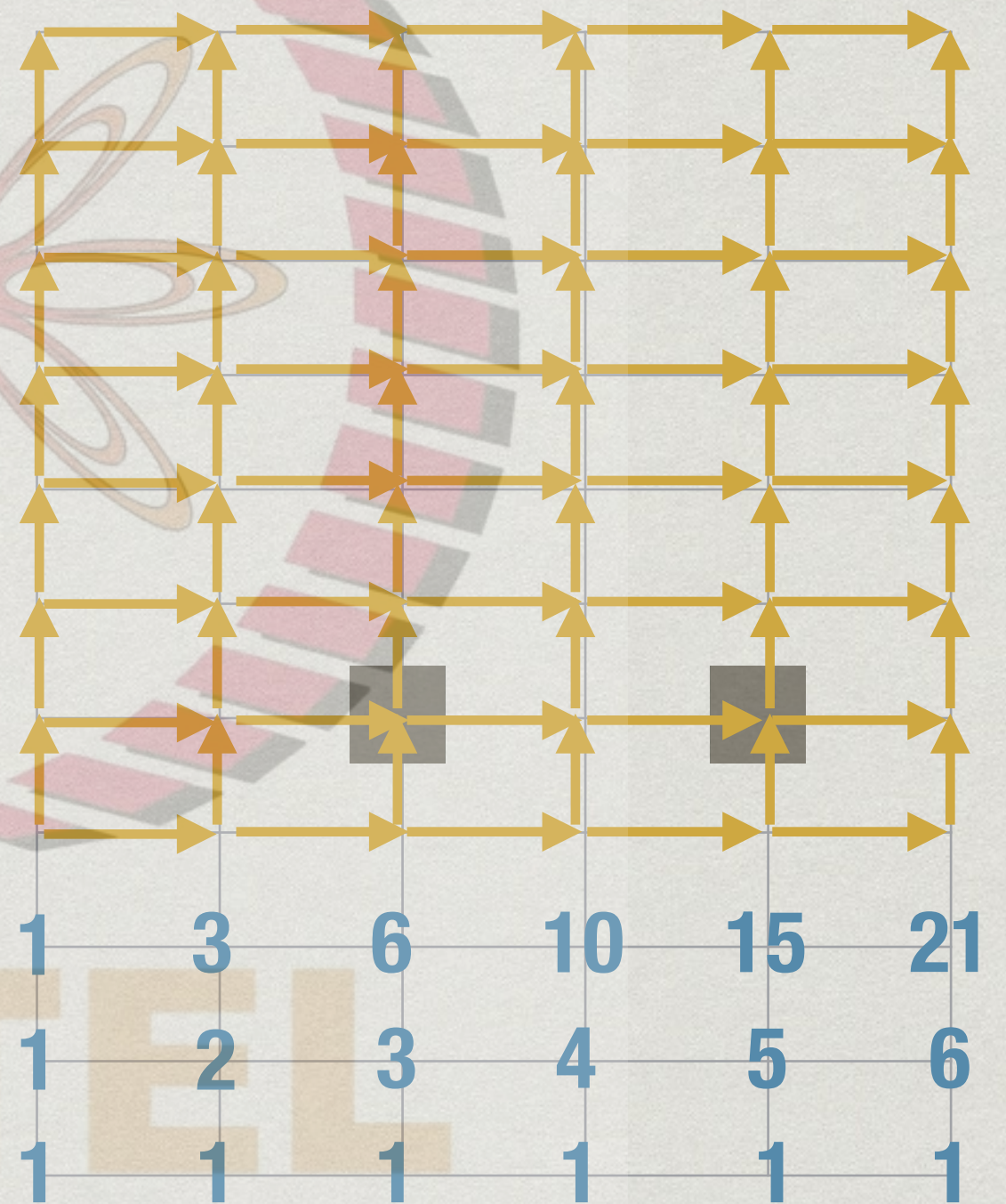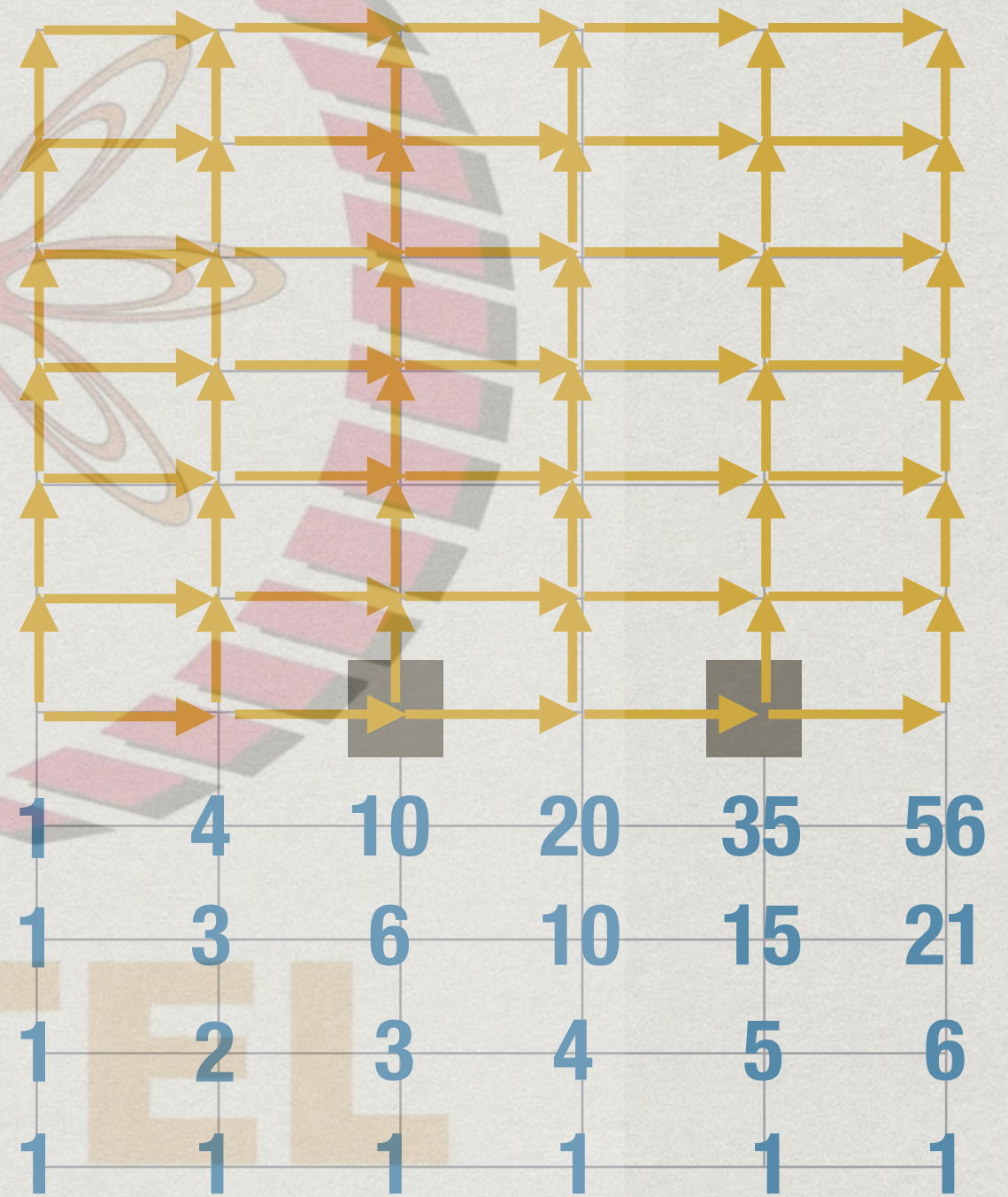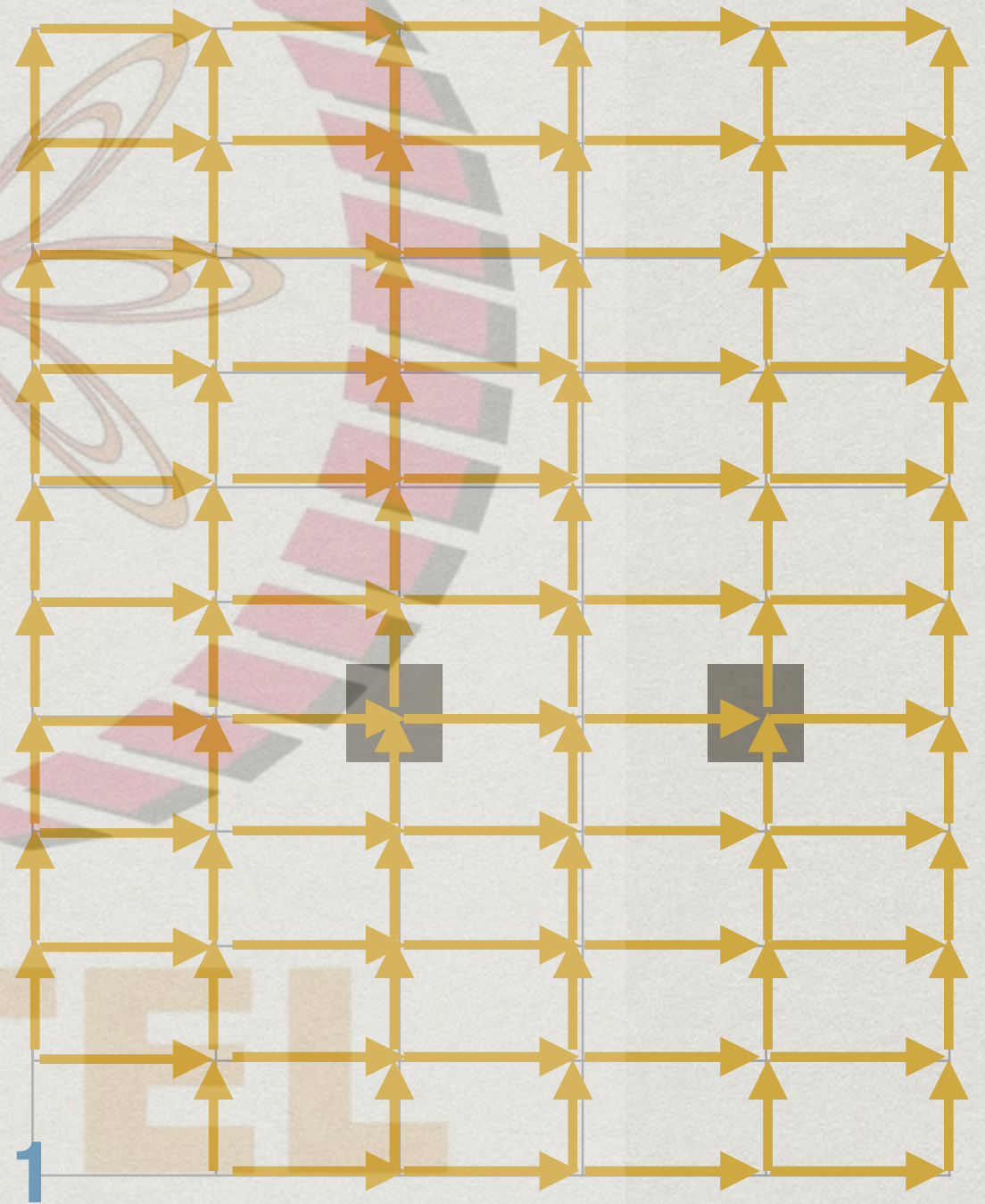* Paths(0,0) has no dependencies

* Start at (0,0)

(0,0)

Dynamic programming

* Start at (0,0)

* Fill row by row

Dynamic programming

* Start at (0,0)

* Fill row by row

1

Dynamic programming

* Start at (0,0)

* Fill row by row
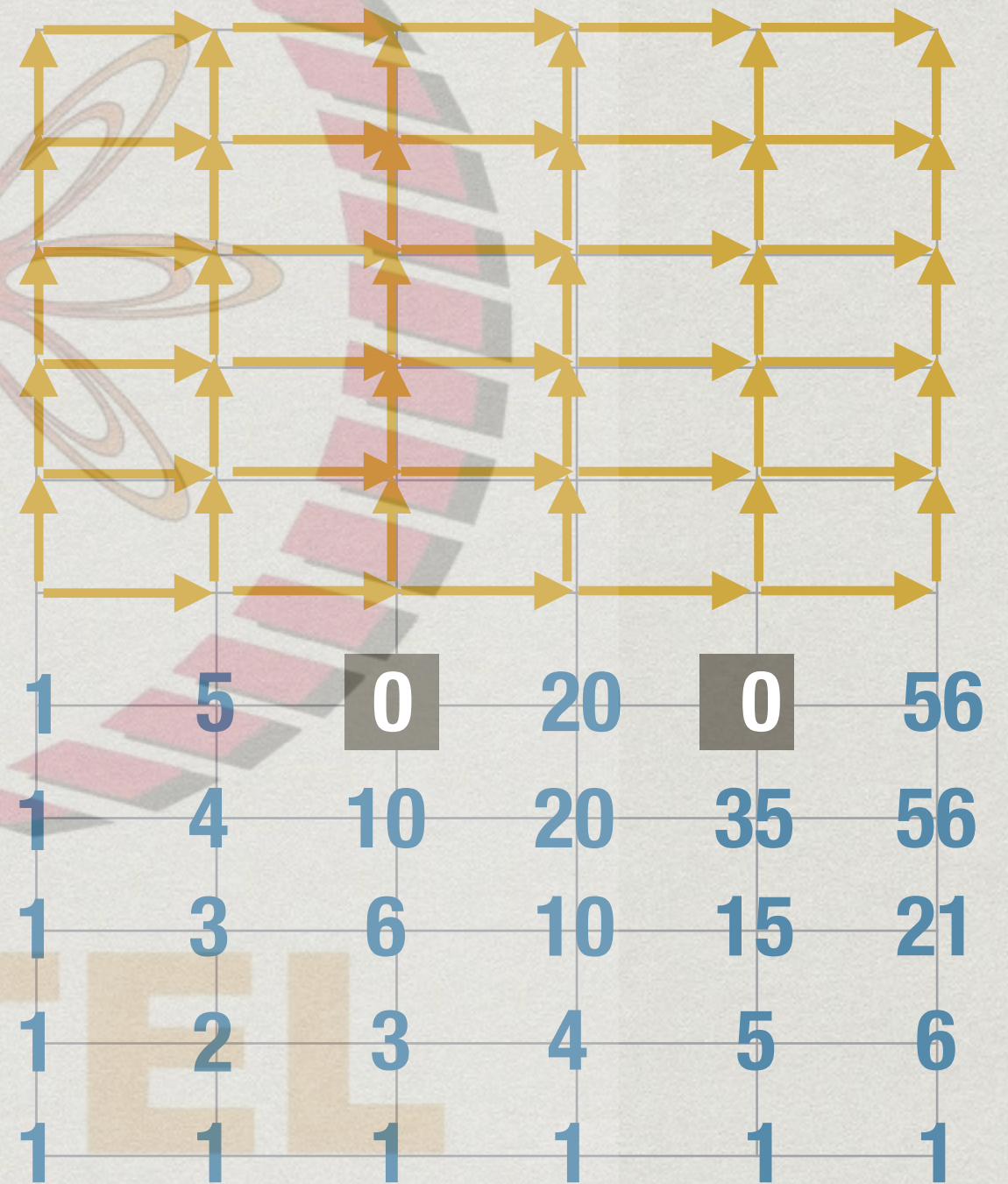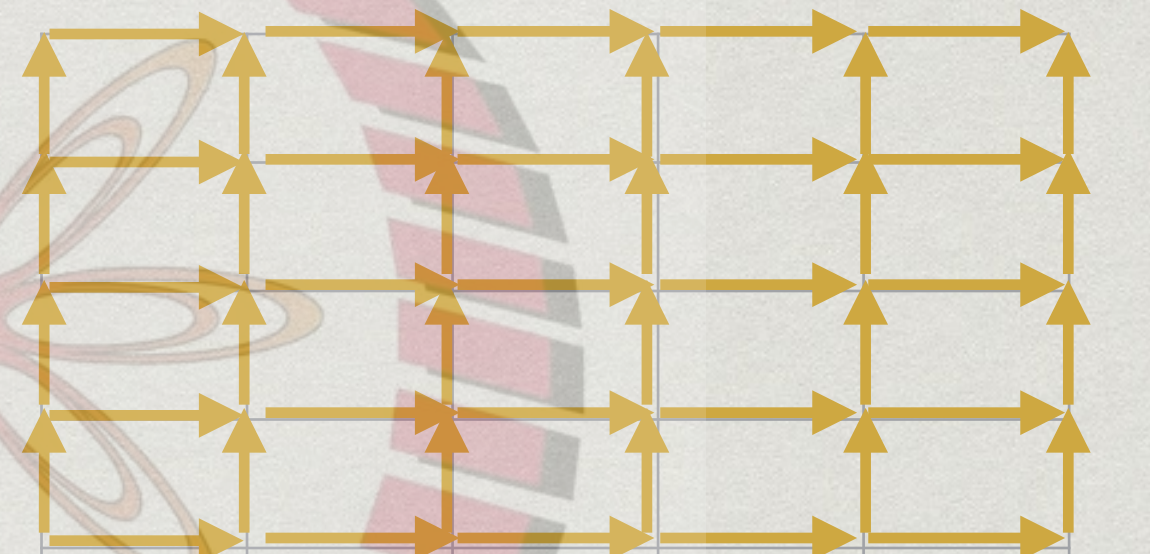
Dynamic programming

* Start at (0,0)

* Fill row by row

Dynamic programming

* Start at (0,0)

* Fill row by row

1	3	6	10	15	21
1	2	3	4	5	6
1	1	1	1	1	1

Dynamic programming

* Start at (0,0)

* Fill row by row

1	4	10	20	35	56
1	3	6	10	15	21
1	2	3	4	5	6
1	1	1	1	1	1

Dynamic programming

* Start at (0,0)

* Fill row by row

1	5	0	20	0	56
1	4	10	20	35	56
1	3	6	10	15	21
1	2	3	4	5	6
1	1	1	1	1	1

Dynamic programming

* Start at (0,0)

* Fill row by row

1	6	6	26	26	82
1	5	**0**	20	**0**	56
1	4	10	20	35	56
1	3	6	10	15	21
1	2	3	4	5	6
1	1	1	1	1	1

Dynamic programming

1	11	51	181	526	1363
1	10	40	130	345	837
1	9	30	90	215	492
1	8	21	60	125	272
1	7	13	39	65	147
1	6	6	26	26	82
1	5	**0**	20	**0**	56
1	4	10	20	35	56
1	3	6	10	15	21
1	2	3	4	5	6
1	1	1	1	1	1

* Start at (0,0)

* Fill row by row

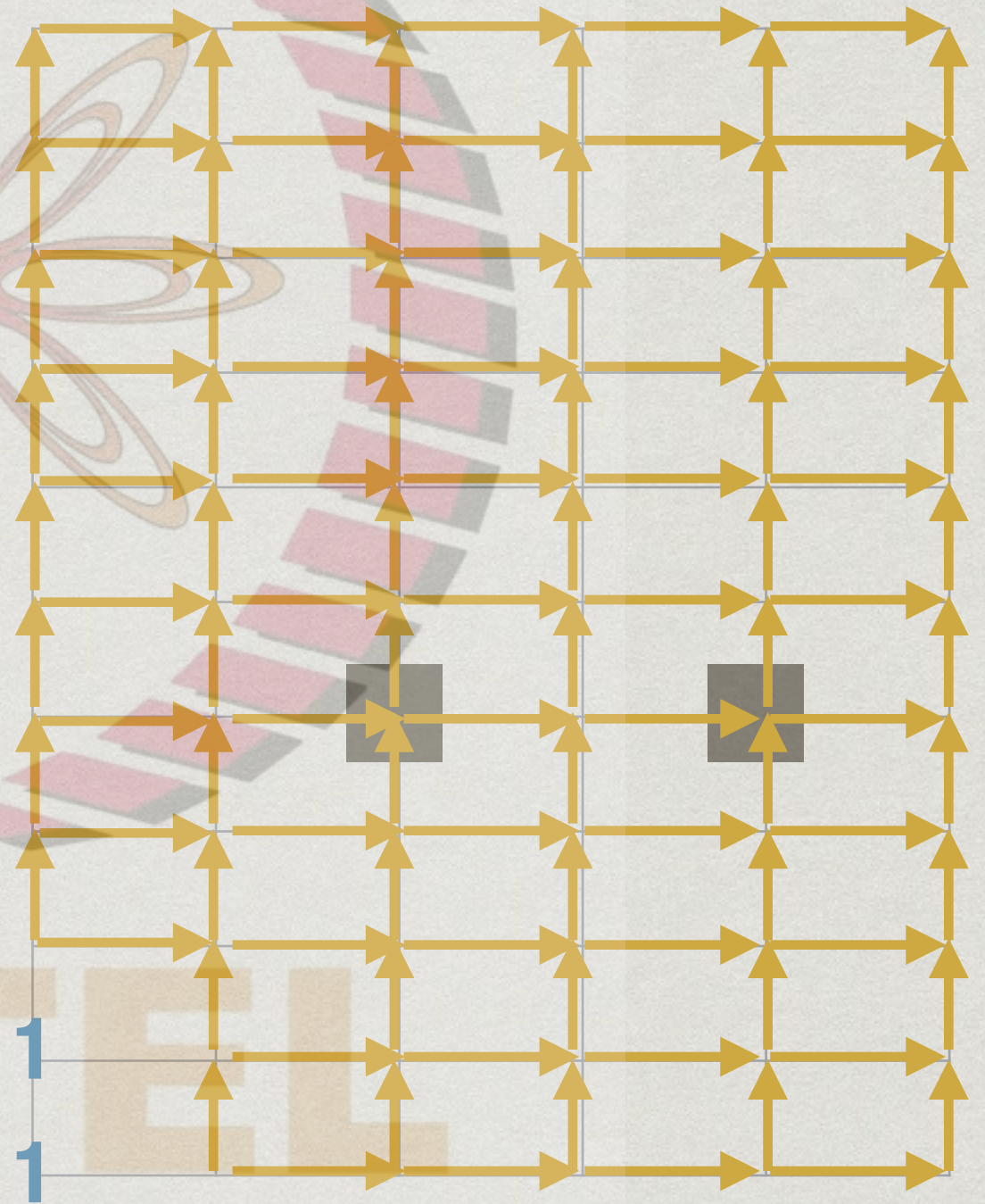# Dynamic programming

* Start at (0,0)

* Fill by column

Dynamic programming

* Start at (0,0)

* Fill by column

1

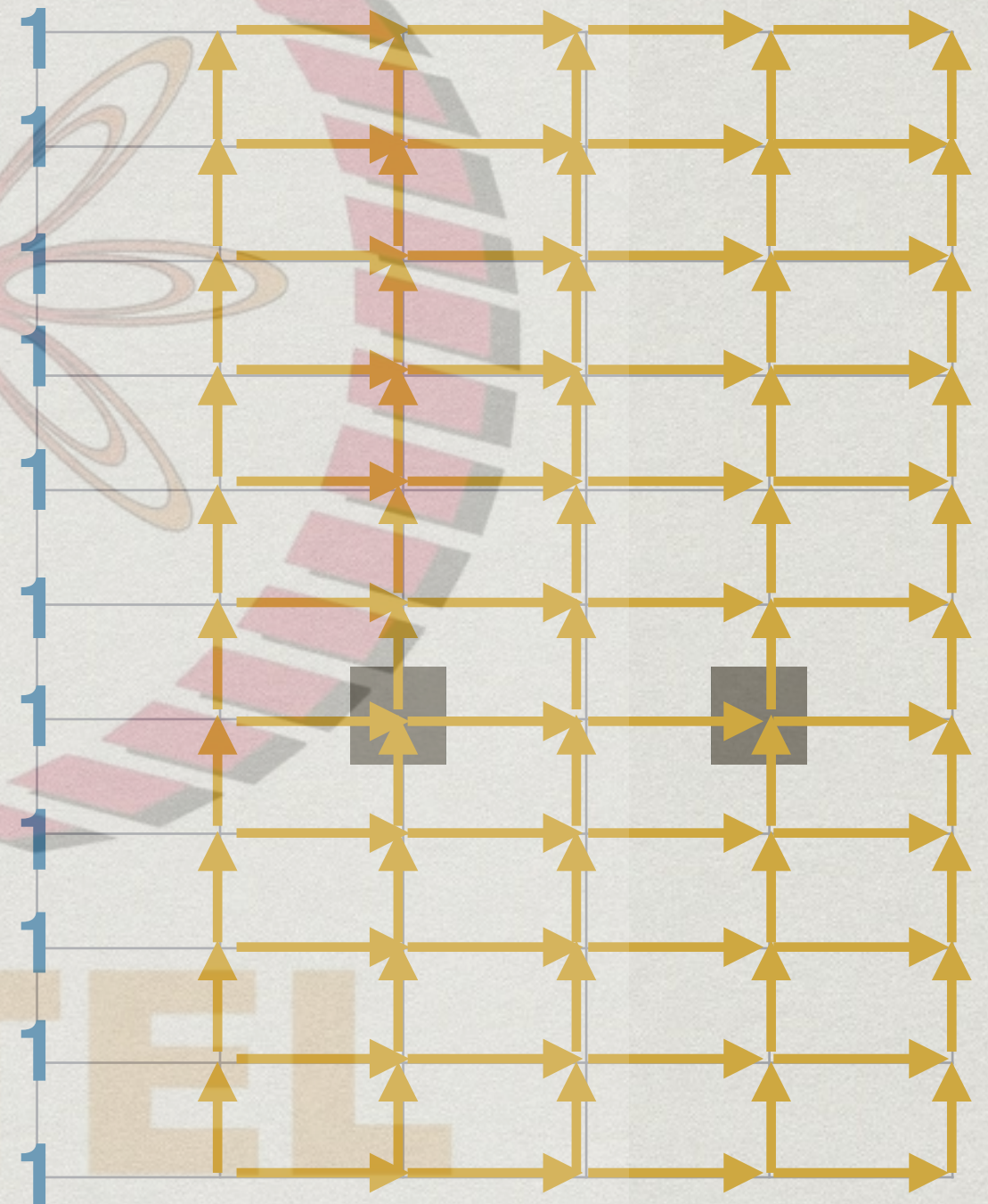# Dynamic programming

* Start at (0,0)

* Fill by column

1
1

Dynamic programming

* Start at (0,0)

* Fill by column

Dynamic programming

* Start at (0,0)

* Fill by column

Dynamic programming

* Start at (0,0)

* Fill by column
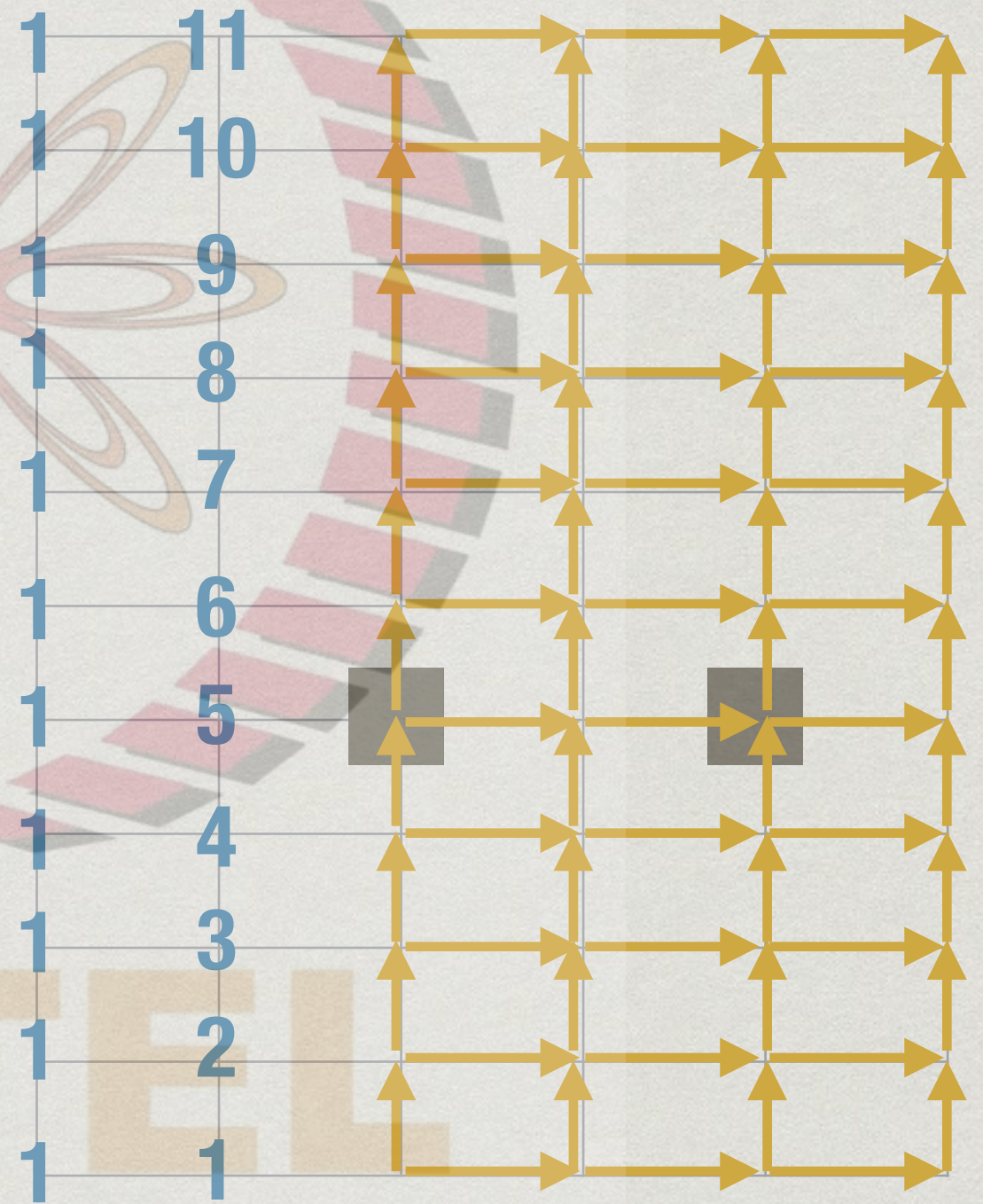
1	11	51	181	526	1363
1	10	40	130	345	837
1	9	30	90	215	492
1	8	21	60	125	272
1	7	13	39	65	147
1	6	6	26	26	82
1	5	0	20	0	56
1	4	10	20	35	56
1	3	6	10	15	21
1	2	3	4	5	6
1	1	1	1	1	1

Dynamic programming

* Start at (0,0)

* Fill by diagonal

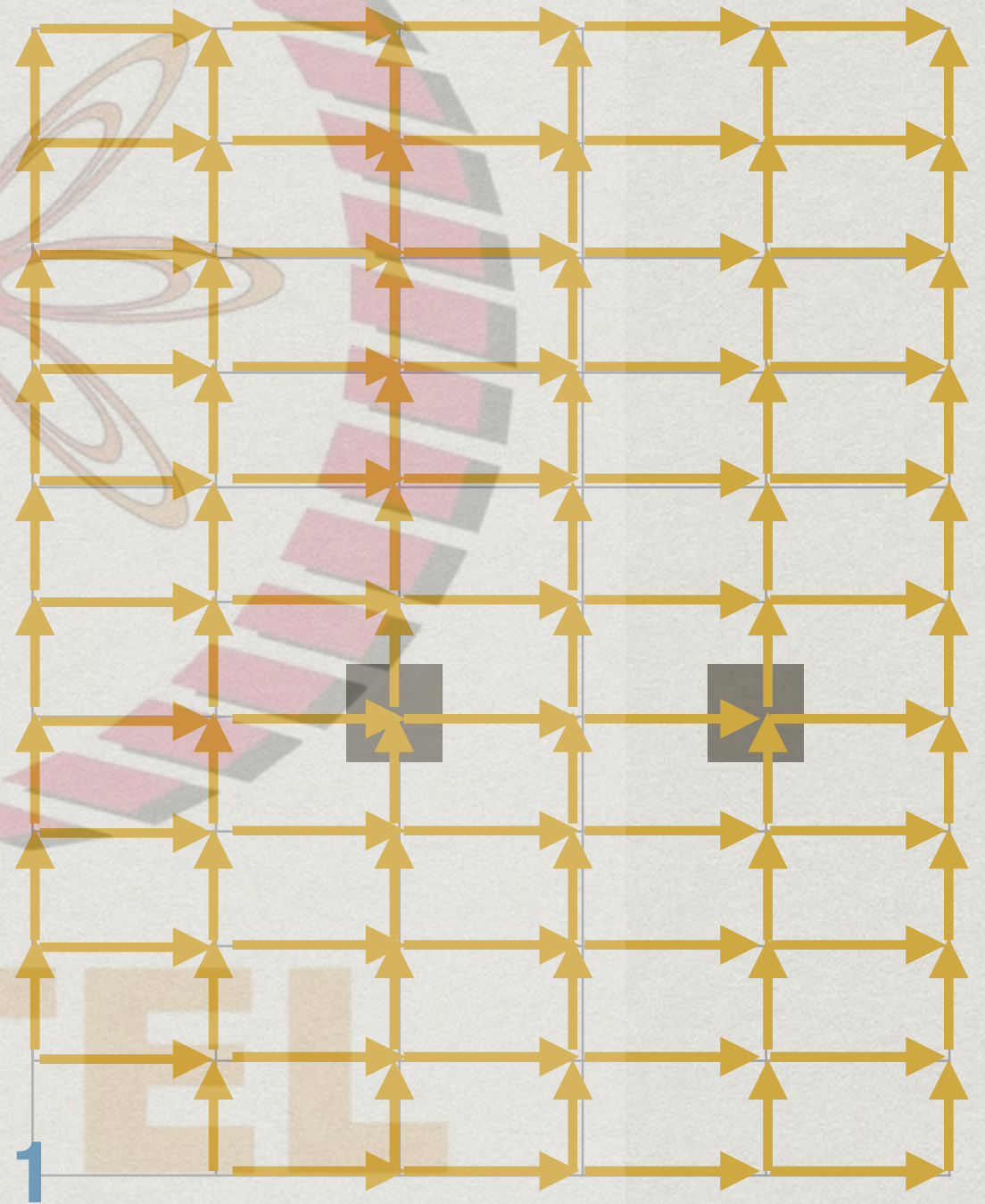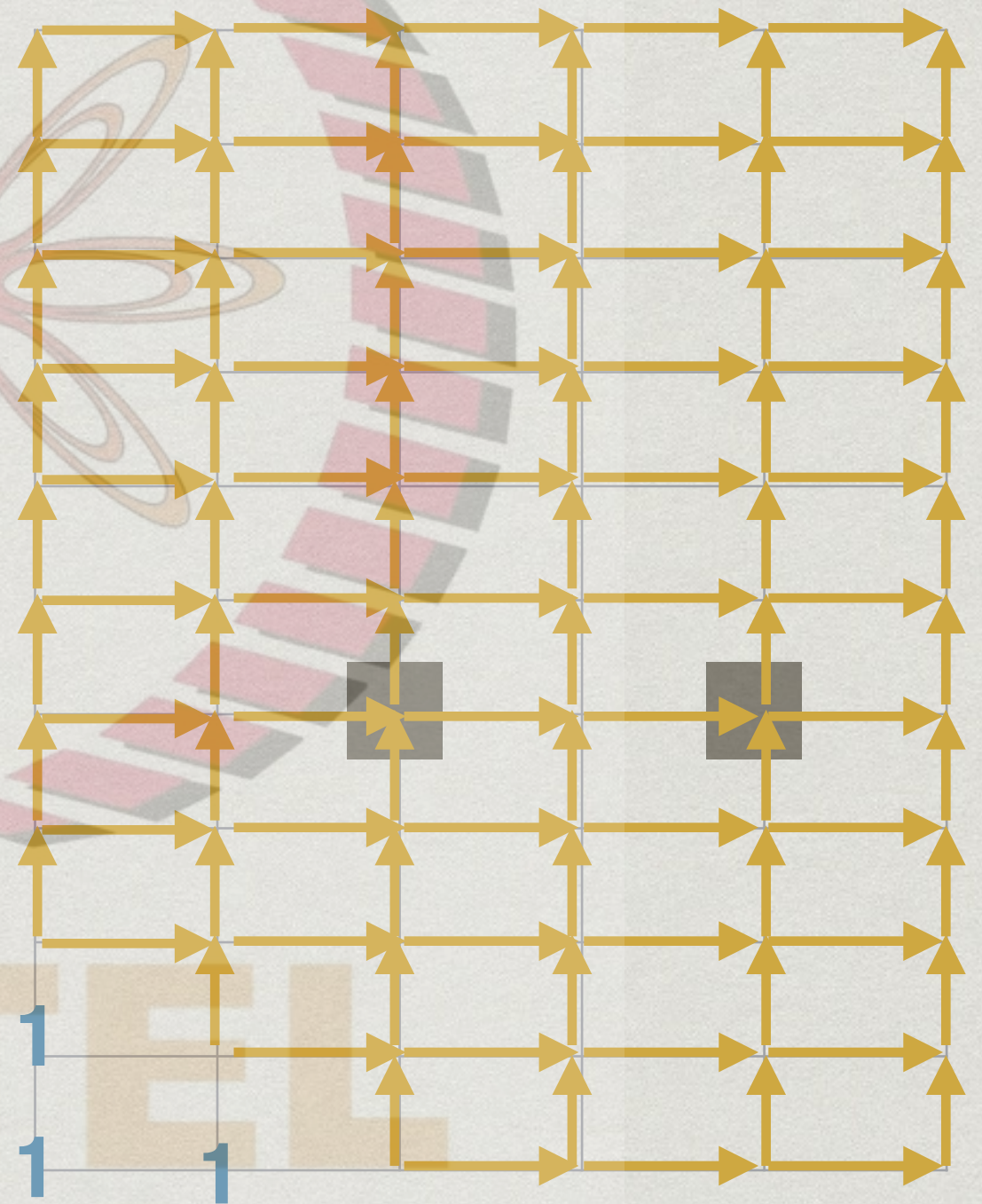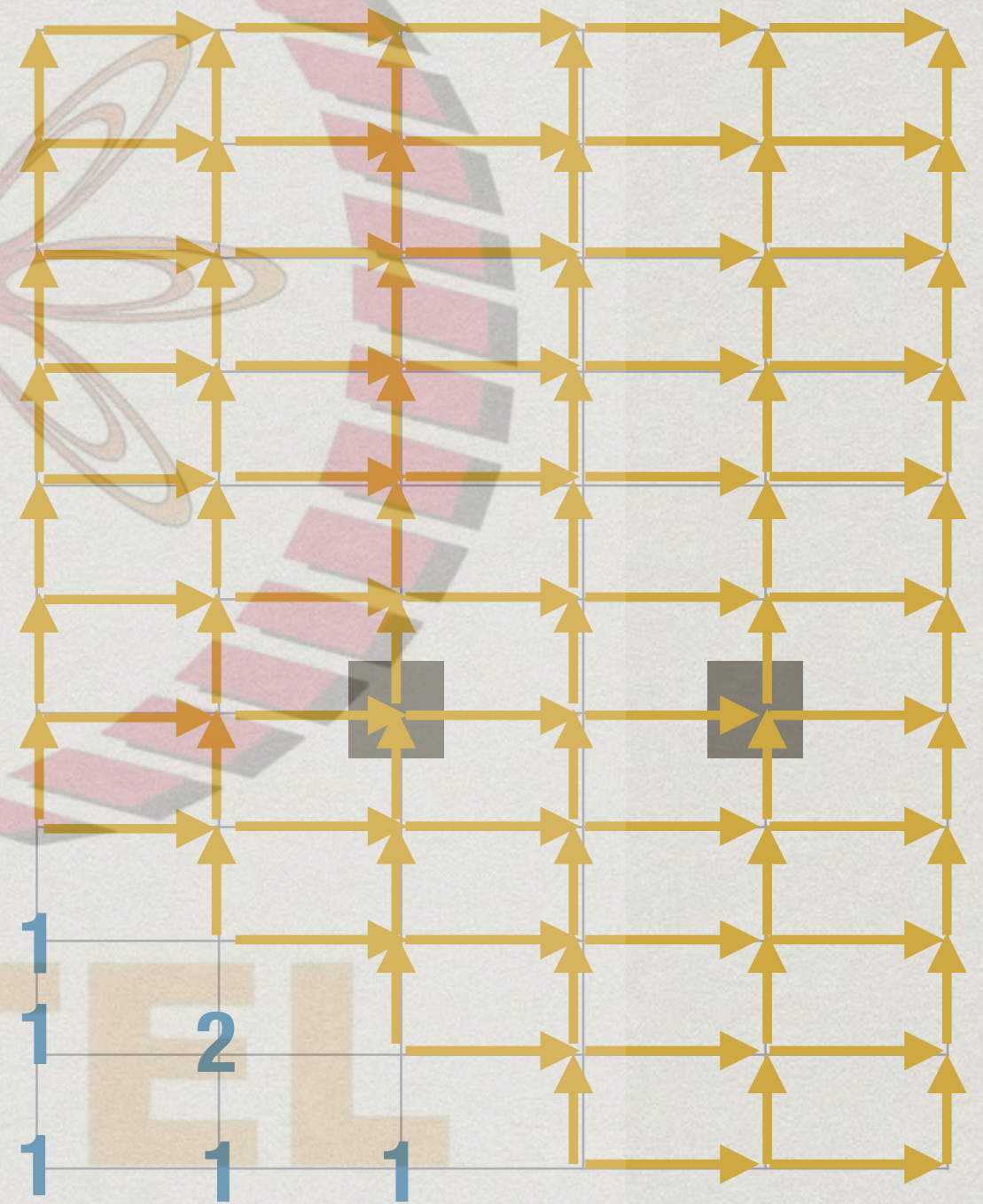# Dynamic programming

* Start at (0,0)

* Fill by diagonal

1

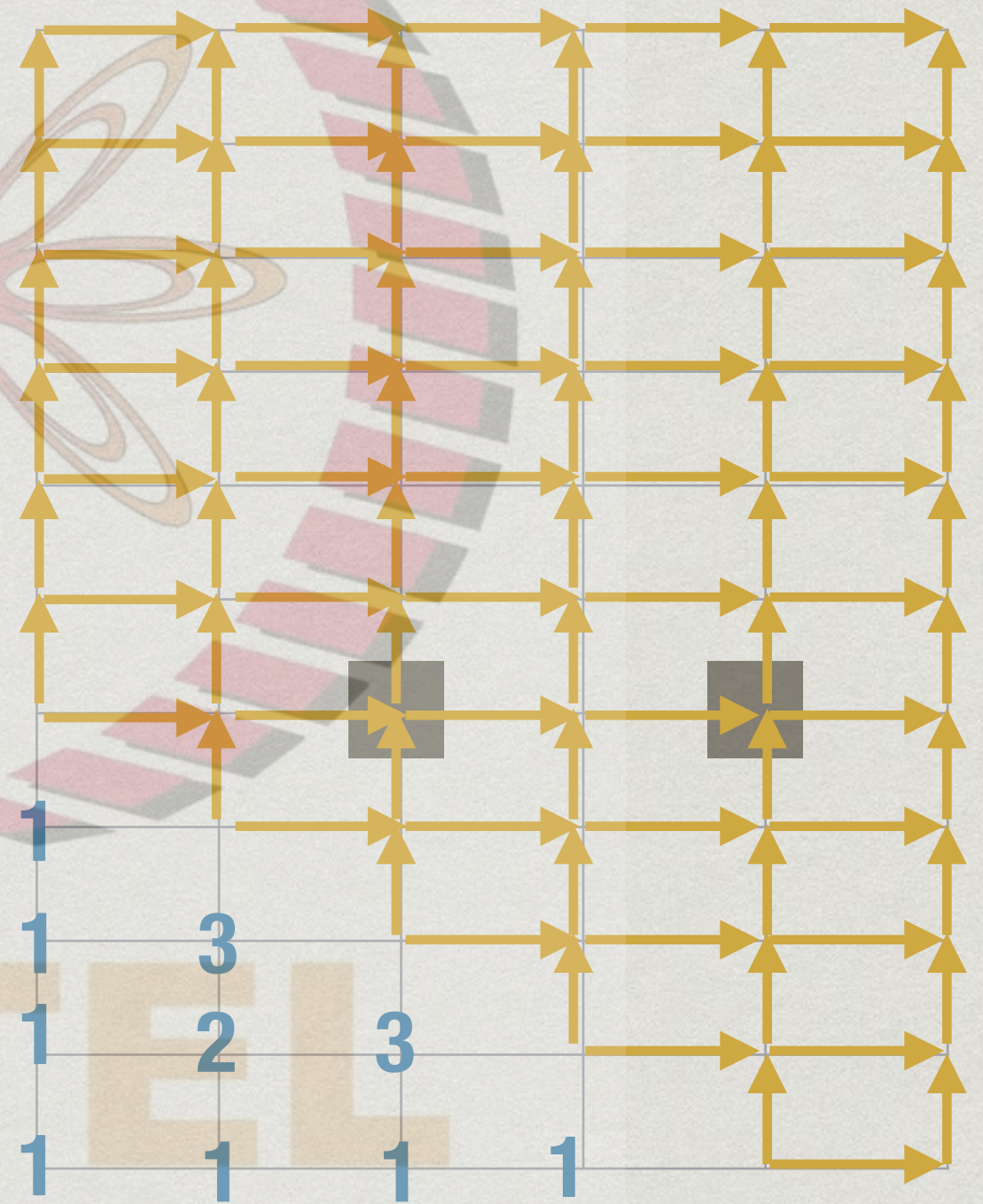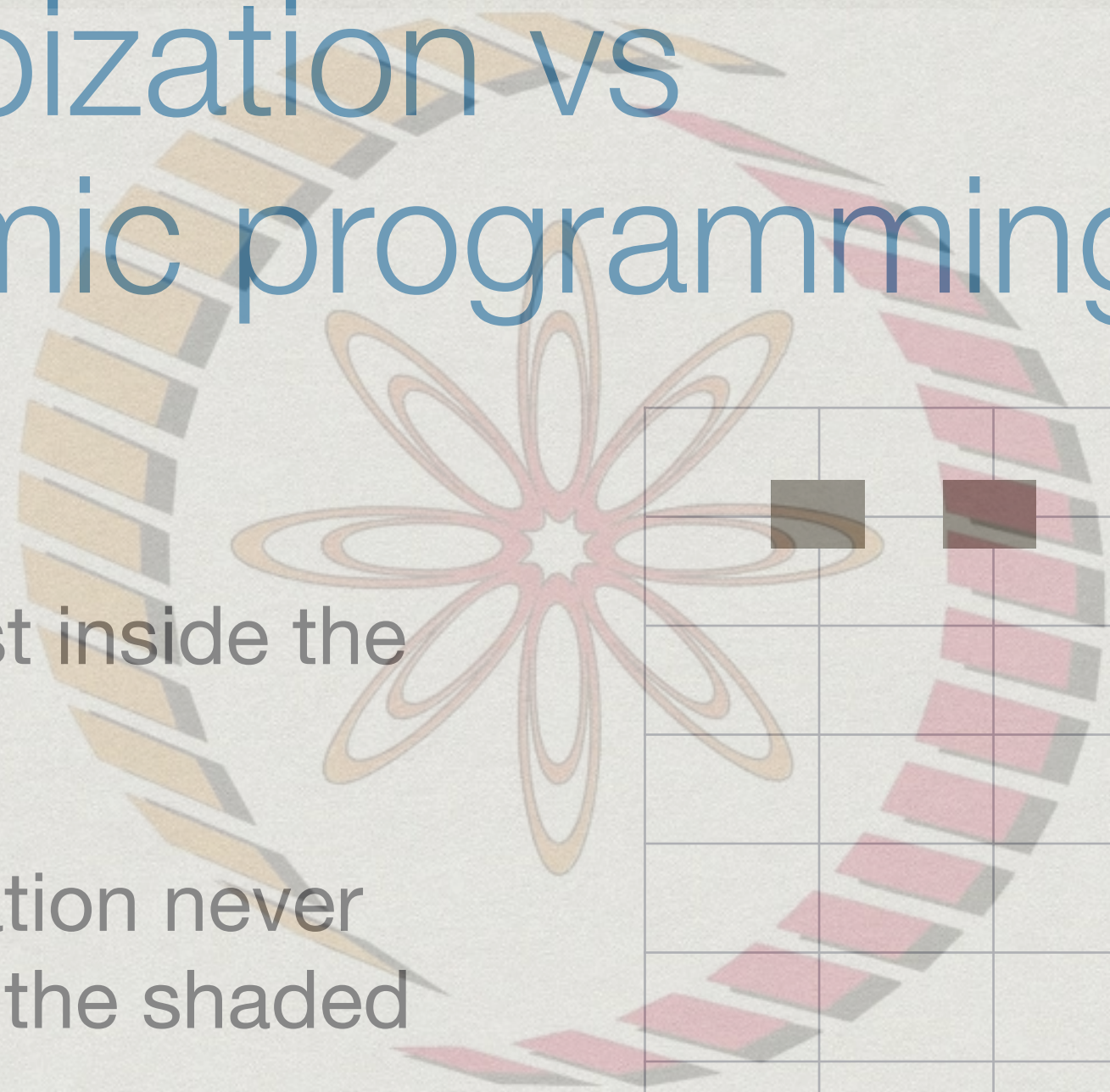# Dynamic programming

* Start at (0,0)

* Fill by diagonal

Dynamic programming

* Start at (0,0)

* Fill by diagonal

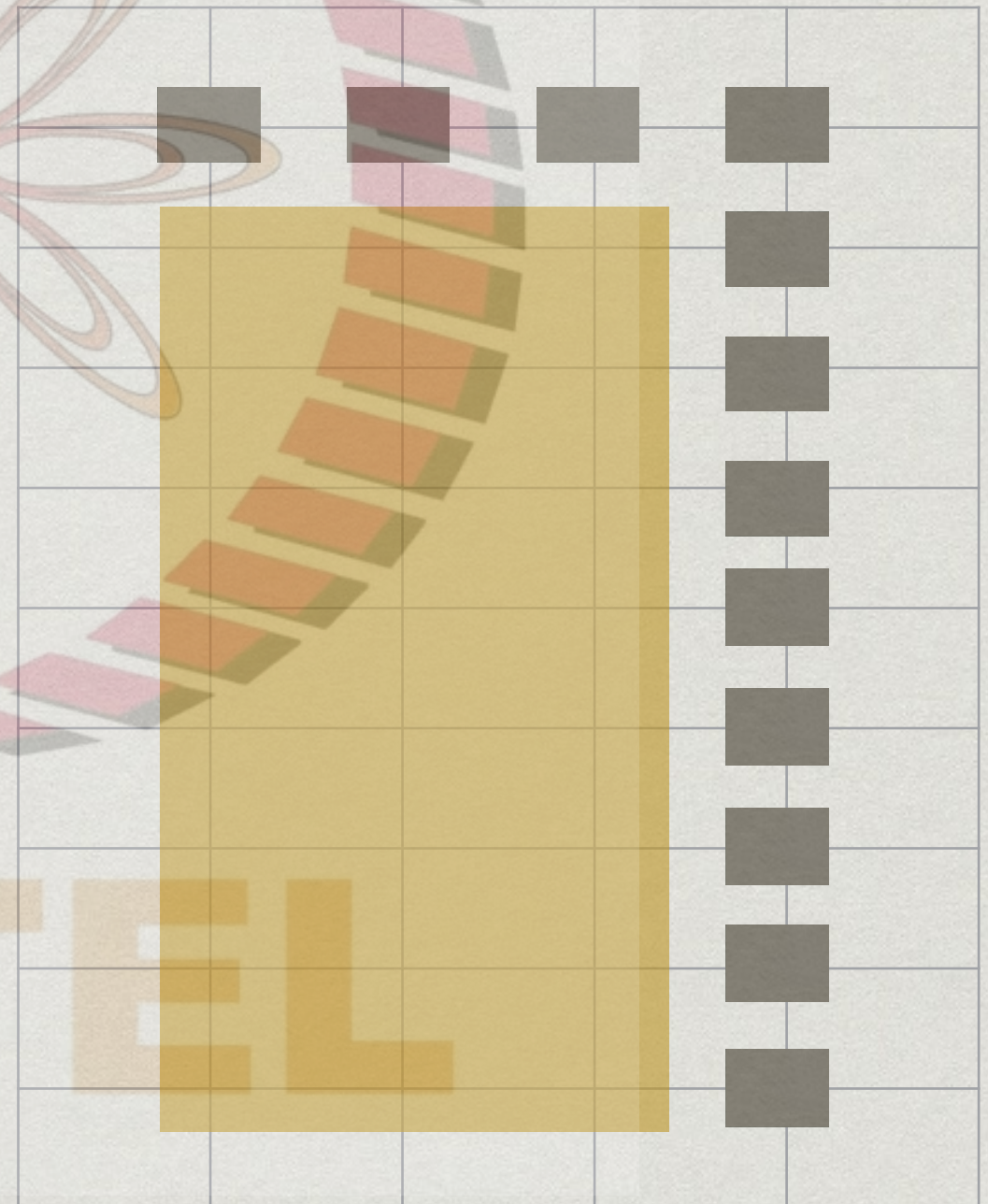# Dynamic programming

* Start at (0,0)

* Fill by diagonal

Memoization vs dynamic programming

* Holes just inside the border

* Memoization never explores the shaded region

Memoization vs dynamic programming

* Holes just inside the border

* Memoization never explores the shaded region

Memoization vs dynamic programming

* Memo table has O(m+n) entries

* Dynamic programming blindly fills all O(mn) entries

* Iteration vs recursion —"wasteful" dynamic programming is still better, in general