# Automated discovery of permutation patterns

Henning Úlfarsson, Reykjavík University
joint work with Anders Claesson, University of Strathclyde

*Permutation Patterns 2012, University Of Strathclyde,*

# Origin of this talk

Consequences of the Lakshmibai-Sandhya Theorem

Sara Billey
University of Washington
http://www.math.washington.edu/∼billey

AWM Anniversary Conference, September 18, 2011

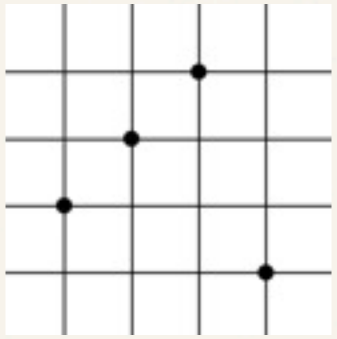
**Open Problems**

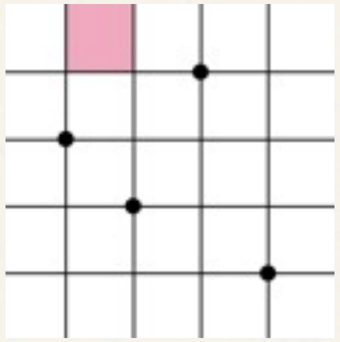
1. Give a pattern based algorithm to produce the factorial and/or Gorenstein locus of a Schubert variety.
2. Describe the maximal singular locus of a Schubert variety for other semisimple Lie groups using generalized pattern avoidance.
3. Find a method to "learn" marked mesh patterns by computer.

# What do we want to do?

- Come up with and prove conjectures such as:

*A perm is West-2-stack-sortable if and only if it avoids*





(Proved by West in his thesis, 1990)

# There is a nice algorithm...

- ... for classical patterns: What patterns does this class avoid?

  {1, 12,21, 132,213,231,312,321,
  1432,2143,2413,2431,3142,3214,3241,3412,3421,4132,4213,4231,4312, ... }

- We scan through the list and when we see the first missing perm we add it to the *base* B = {123}

- We keep on scanning and when we see another missing perm we check if it contains something from the base. If not we add it to the base. B = {123,4321}. We keep going and hope the base stops growing

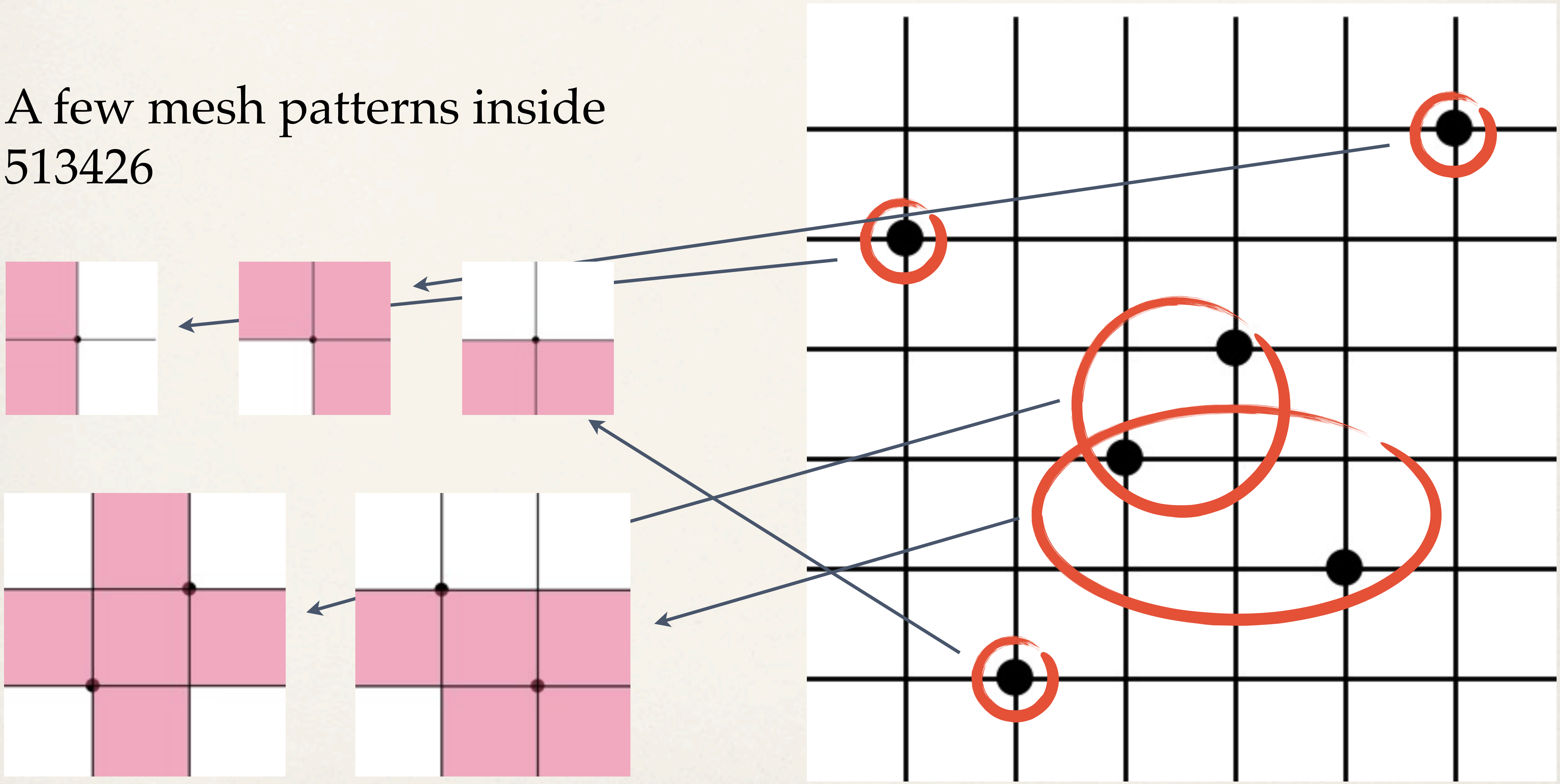
- Note that we are using the missing perms to discover the patterns

# But not everything is “classical”

- Sometimes a set of perms has an infinite or no basis
- Mesh patterns to the rescue! (Brändén & Claesson, 2010)
- Any set of permutations can be described by mesh patterns

# Mesh patterns in permutations

- A few mesh patterns inside 513426



# An algorithm for mesh patterns

- Step 1: Discover the allowed patterns
  This is easy - also parallelizes beautifully!

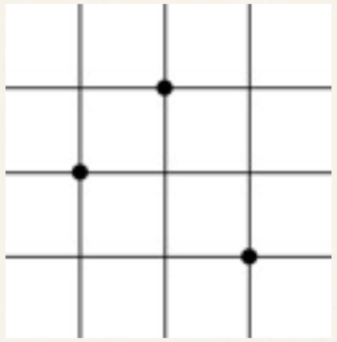
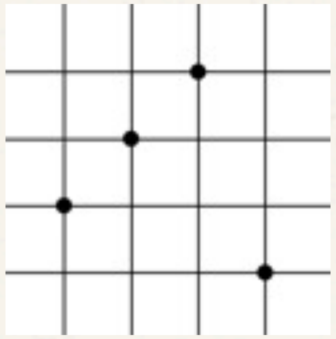
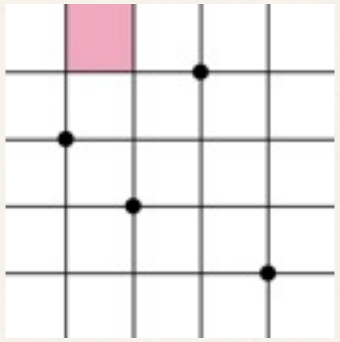
This was the first implementation - it was slow!

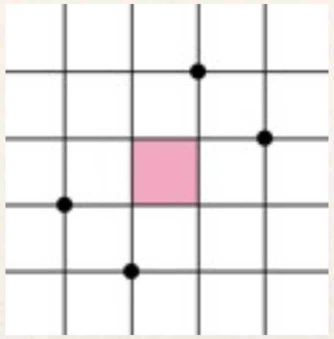
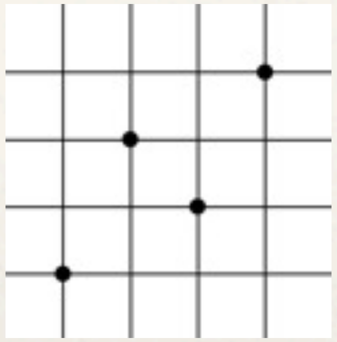
- Step 2: Find forbidden patterns
  This is the hard part. One way is to search through all possibilities

The search space will grow like $n!*2^{(n+1)^2}$

- Step 2′: Generate (in a smart way) the forbidden patterns.
  For $n = 4$ this reduces the run-time from 18 hours to 0.18 seconds!

# Testing, testing (sagenb.org)

- Stack-sortable permutations avoid  (Knuth ~1960)
- West-2-stack-sortable perms avoid   (West 1990)
- Factorial Schubert varieties correspond to permutations avoiding

(Bousquet-Mélou & Butler 2006)

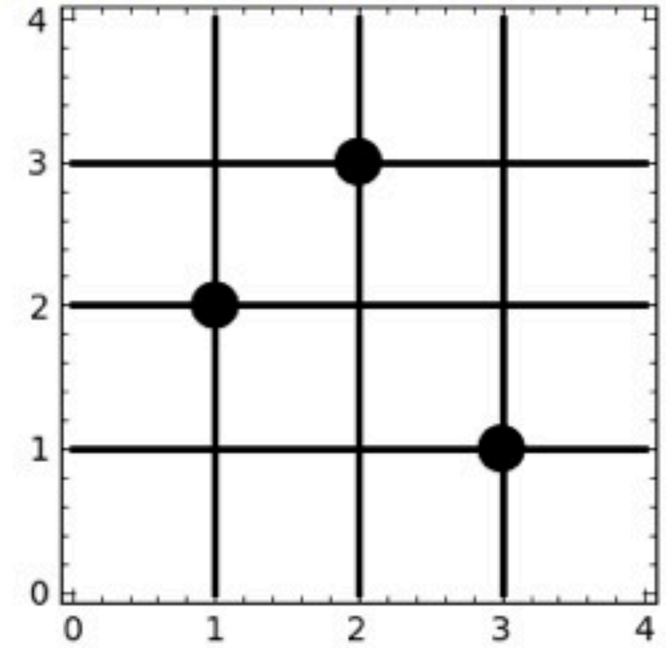
# Stack-sortable perms

```
C = 5

prop = lambda x: is_stack_sortable(x)


A = para_perms_sat_prop(C,prop)
size_of_subdicts(A)
```

```
Perms of length 1 with this property are 1
Perms of length 2 with this property are 2
Perms of length 3 with this property are 5
Perms of length 4 with this property are 14
Perms of length 5 with this property are 42
```

```
%time

# Maximum length of patterns to search for
M = 4

# Maximum length of permutations from A to consider.
N = C

# Initializing a dictionary of good patterns that will
# be learned from A
goodpatts = dict()

SG = parallel_guess(M,N,report=False,run_parallel=False)
visualize_patts(SG,6)
```

```
CPU time: 0.63 s,  Wall time: 0.63 s
```



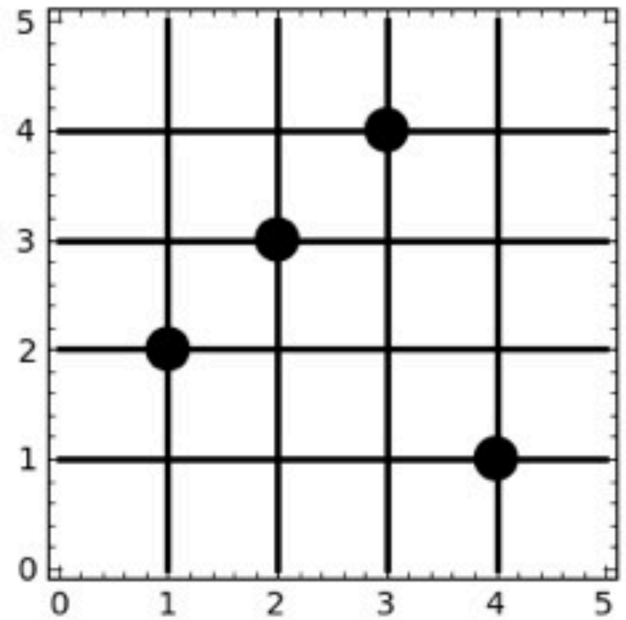
```
C = 5

prop = lambda x: is_West_2_stack_sortable(x)


A = para_perms_sat_prop(C,prop)
size_of_subdicts(A)
```

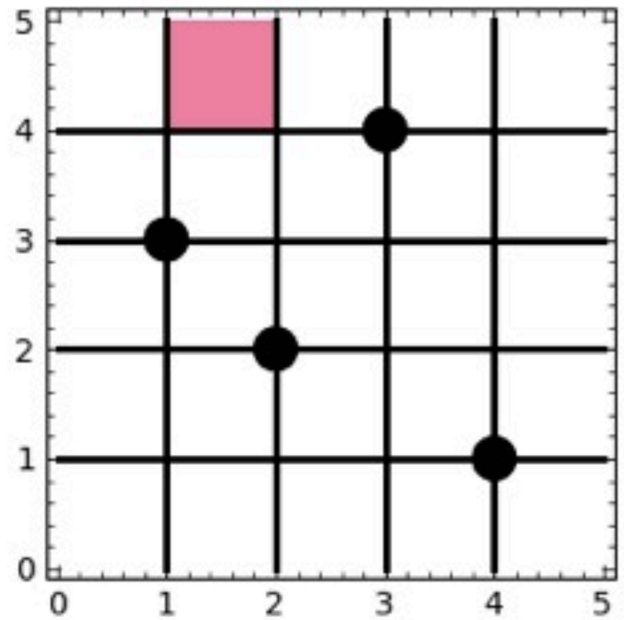
# West-2-stack-sortable perms

```
Perms of length 1 with this property are 1
Perms of length 2 with this property are 2
Perms of length 3 with this property are 6
Perms of length 4 with this property are 22
Perms of length 5 with this property are 91
```

```
%time

# Maximum length of patterns to search for
M = 4

# Maximum length of permutations from A to consider.
N = C

# Initializing a dictionary of good patterns that will
# be learned from A
goodpatts = dict()

SG = parallel_guess(M,N,report=False,run_parallel=False)
visualize_patts(SG,6)
```

```
CPU time: 0.98 s,  Wall time: 0.98 s
```



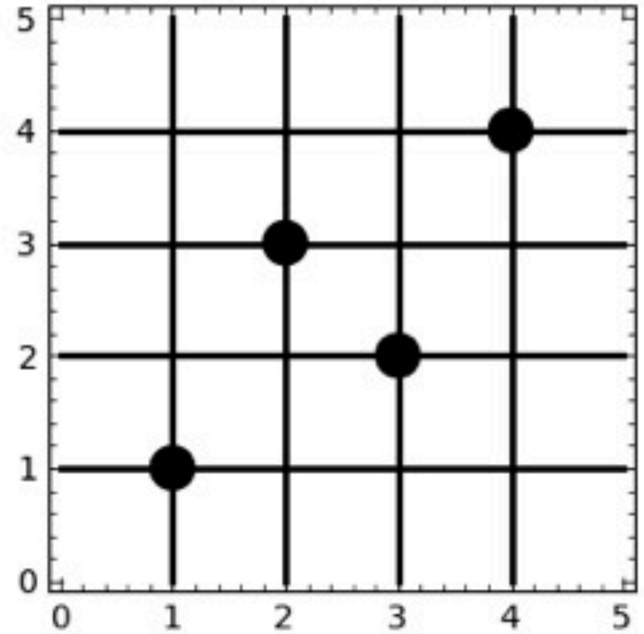
# Factorial Schubert varieties

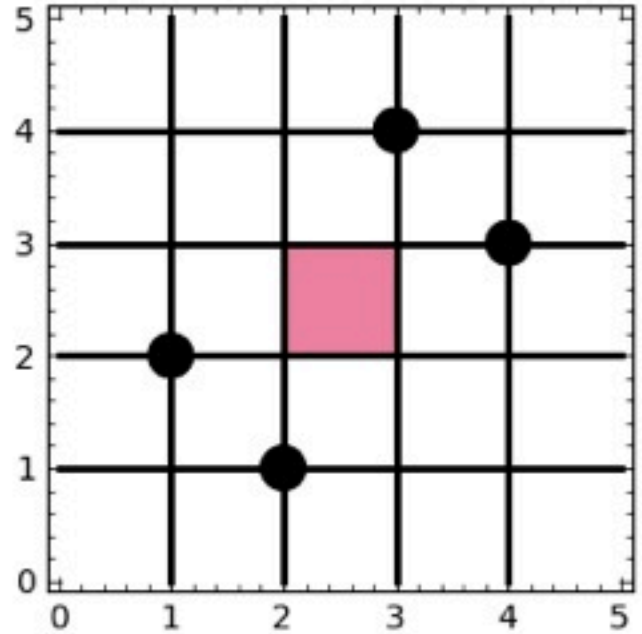
```
C = 5

prop = lambda x: is_factorial(x)


A = para_perms_sat_prop(C,prop)
size_of_subdicts(A)
```

```
Perms of length 1 with this property are 1
Perms of length 2 with this property are 2
Perms of length 3 with this property are 6
Perms of length 4 with this property are 22
Perms of length 5 with this property are 89
```

```
%time

# Maximum length of patterns to search for
M = 4

# Maximum length of permutations from A to consider.
N = C

# Initializing a dictionary of good patterns that will
# be learned from A
goodpatts = dict()

SG = parallel_guess(M,N,report=False,run_parallel=False)
visualize_patts(SG,6)
```

```
CPU time: 0.98 s,  Wall time: 0.98 s
```

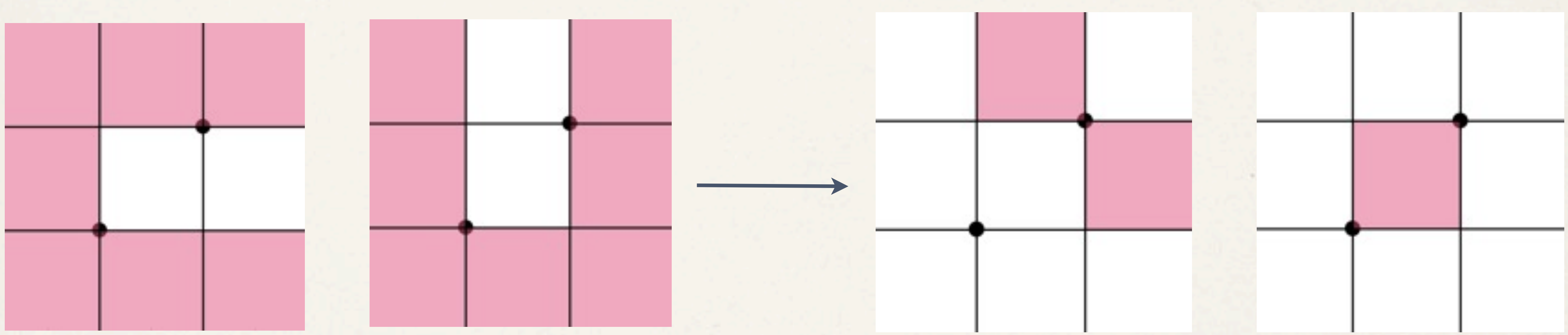


# You can play with this yourself

# How do we generate the forbidden patterns?

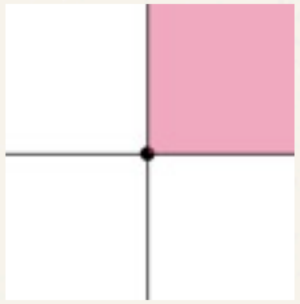
- Given the allowed patterns from step 1) we generate the minimal forbidden patterns (can't search through all possibilities: $n!*2^{(n+1)^2}$)
- What does that mean? If the following patterns are allowed

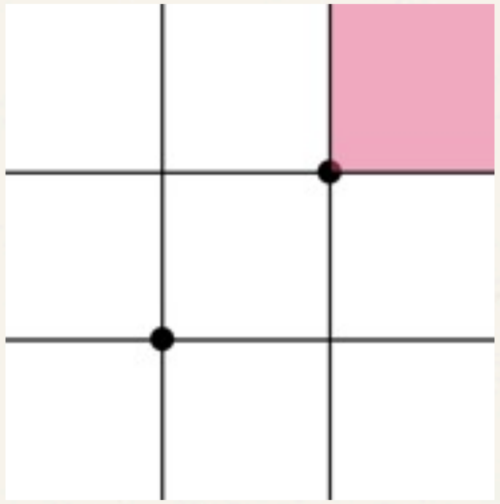


then step 2') will generate two forbidden patterns.

# Redundancy

- We generate the forbidden patterns for each length individually so there is redundancy between different lengths





- We only need to remember that the smaller one is forbidden

- There is still some redundancy in the output which we have ideas on how to remove (based on the shading lemma, upgrading, minimal permutations, but current implementations are too slow)

# New conjectures

- As shown above the algorithm can find old theorems
- Discovered some new conjectures: Tableaux conjectures!
- But first: what is a tableau

# Young tableaux

- There is a beautiful bijection between permutations and pairs of YT's, e.g. 581279643 gives us (using Sage!)

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 9 |
| 4 | 6 | | |
| 5 | | | |
| 7 | | | |
| 8 | | | |

| | | | |
|---|---|---|---|
| 1 | 2 | 5 | 6 |
| 3 | 4 | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |

- The bijection has several nice properties. But it has been hard to connect patterns in permutations to something in the tableaux

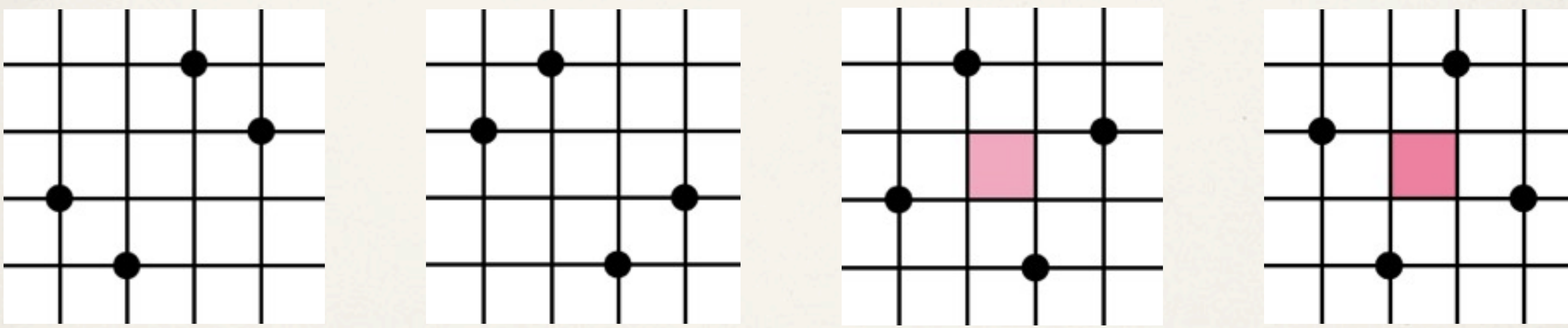
- Only special cases are known, e.g. separable patterns (Crites, Panova & Warrington 2011)

- We have some new conjectures and results

# Perms with hook-shaped tableaux

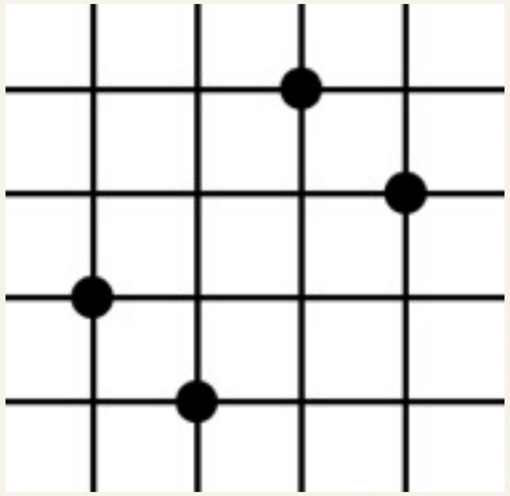
As pointed out by Vince Vatter, this actually follows from a paper by Atkinson on skew-merge perms

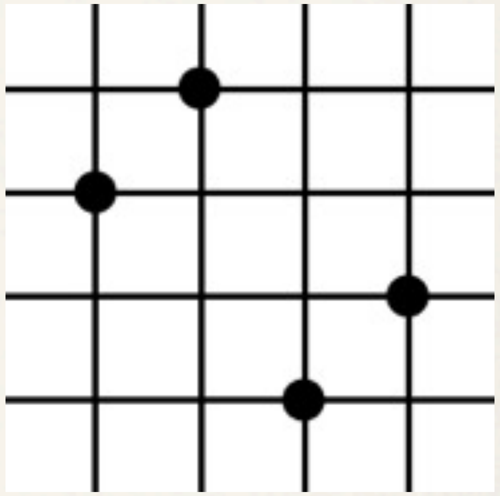
- Theorem (~~HÚ & Claesson~~, Atkinson)
  A perm has hook-shaped tableaux if and only if it avoids



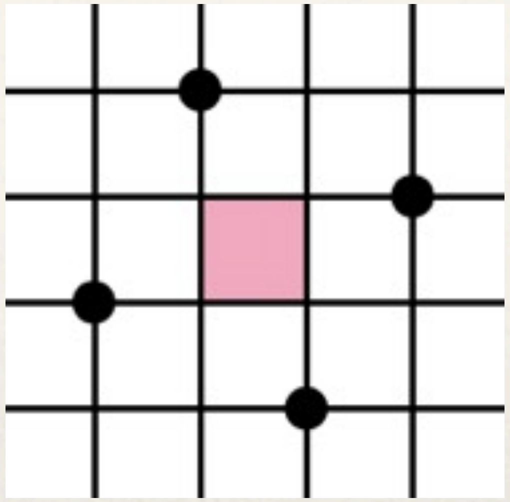
# Sketch of proof

- If a perm contains the patterns then it is not hook-shaped by a theorem of Crites, Panova & Warrington, 2011



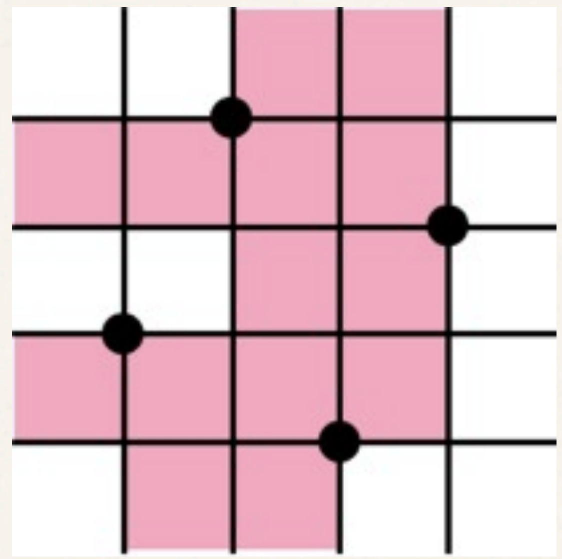


- Assume it contains

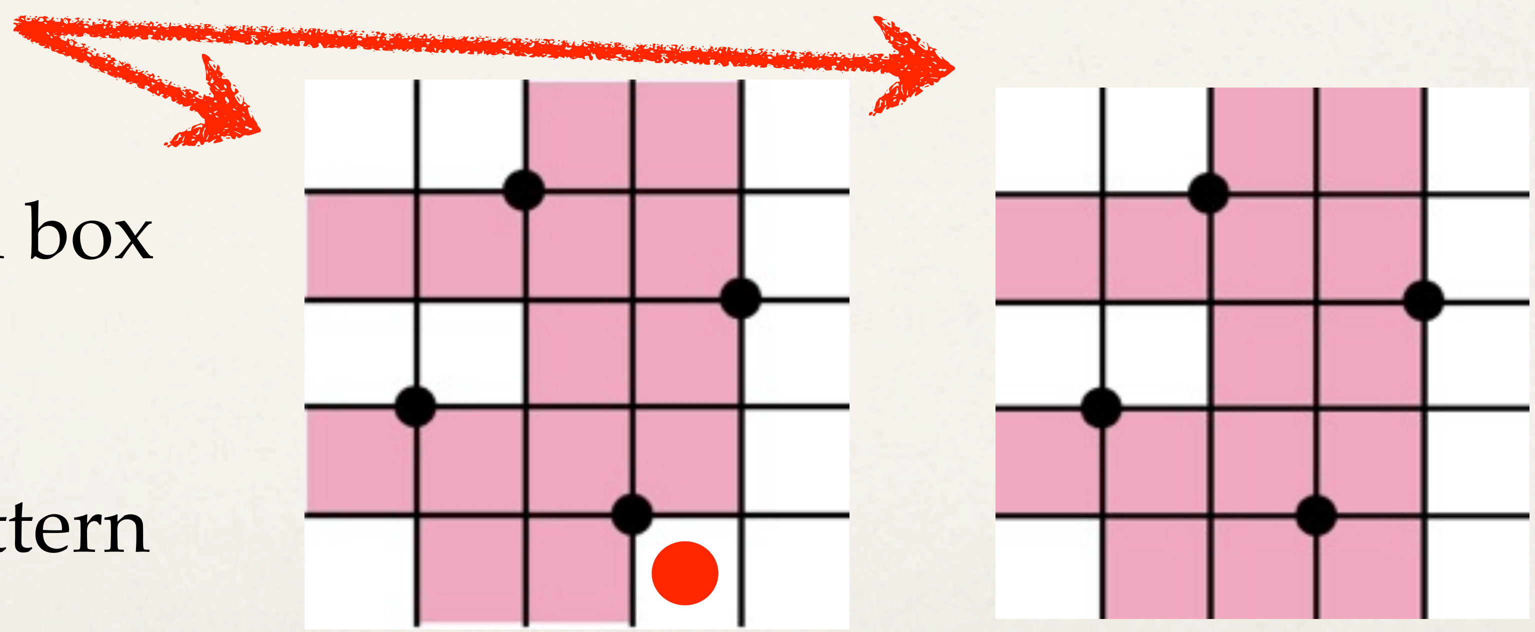


# Sketch of proof, cont.

- We can assume that we have this pattern



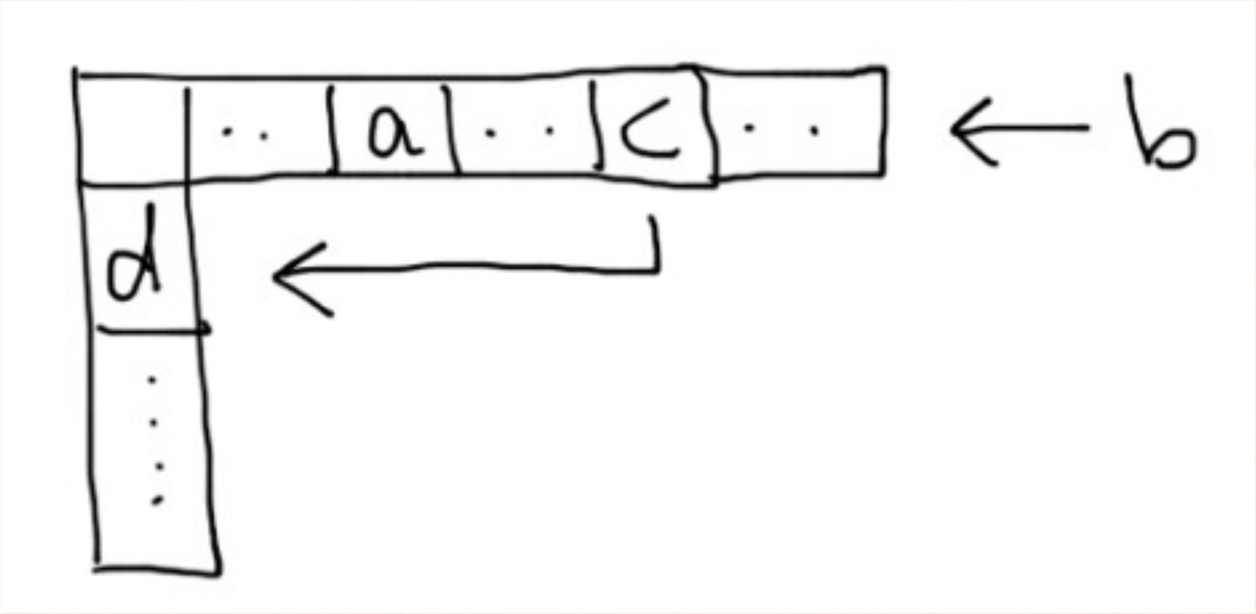
- So we have either



- This will produce a box in the tableaux.
- The other mesh pattern is similar

# Sketch of proof, cont.

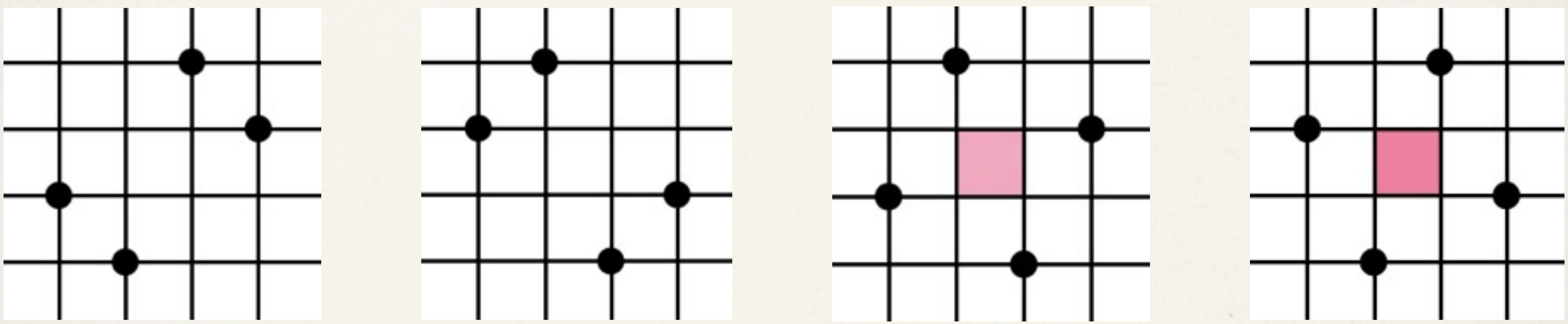
- If there is a box in the tableaux, we let c be the element that first creates it, b the element that bumps it, d the element in (2,1) and a the element that bumped d



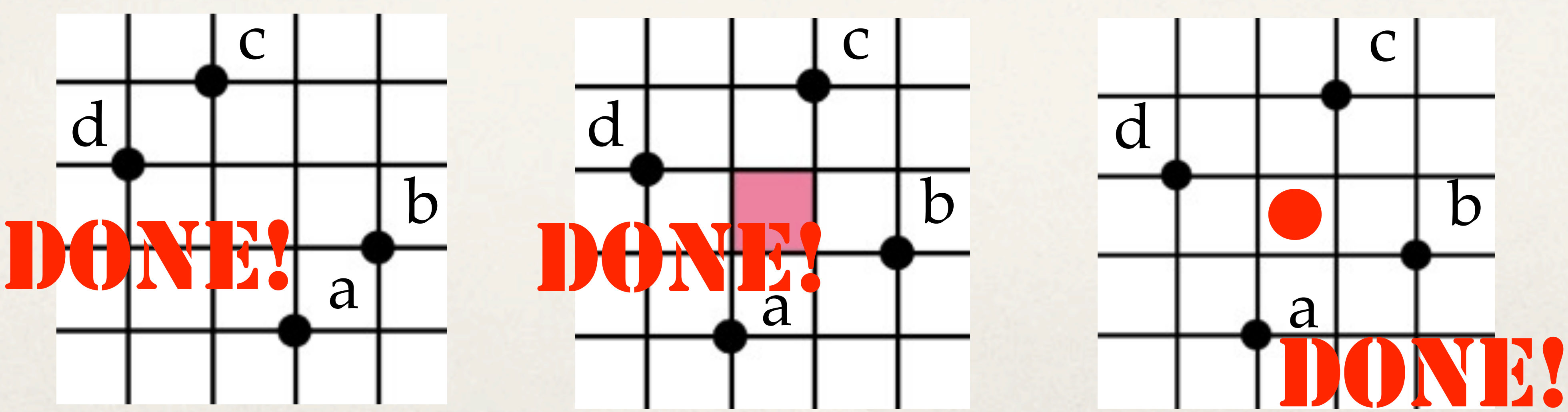

- Then we know that a < b,c,d and b,d < c, and that d appeared first in the perm, and b appeared last

# Sketch of proof, cont.

- Recall, trying to produce



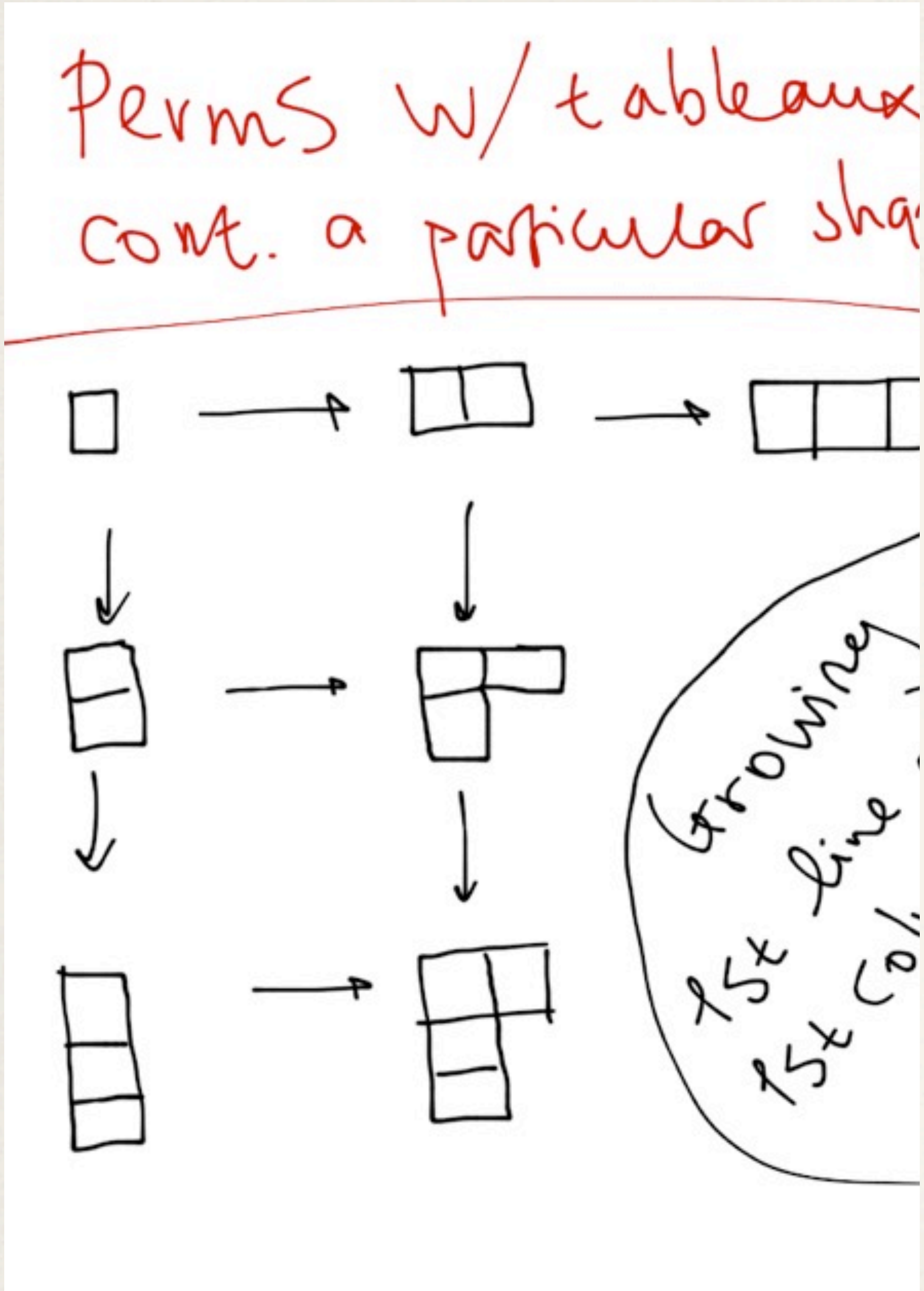
- Now assume b < d (other case is similar). Then we have one of:



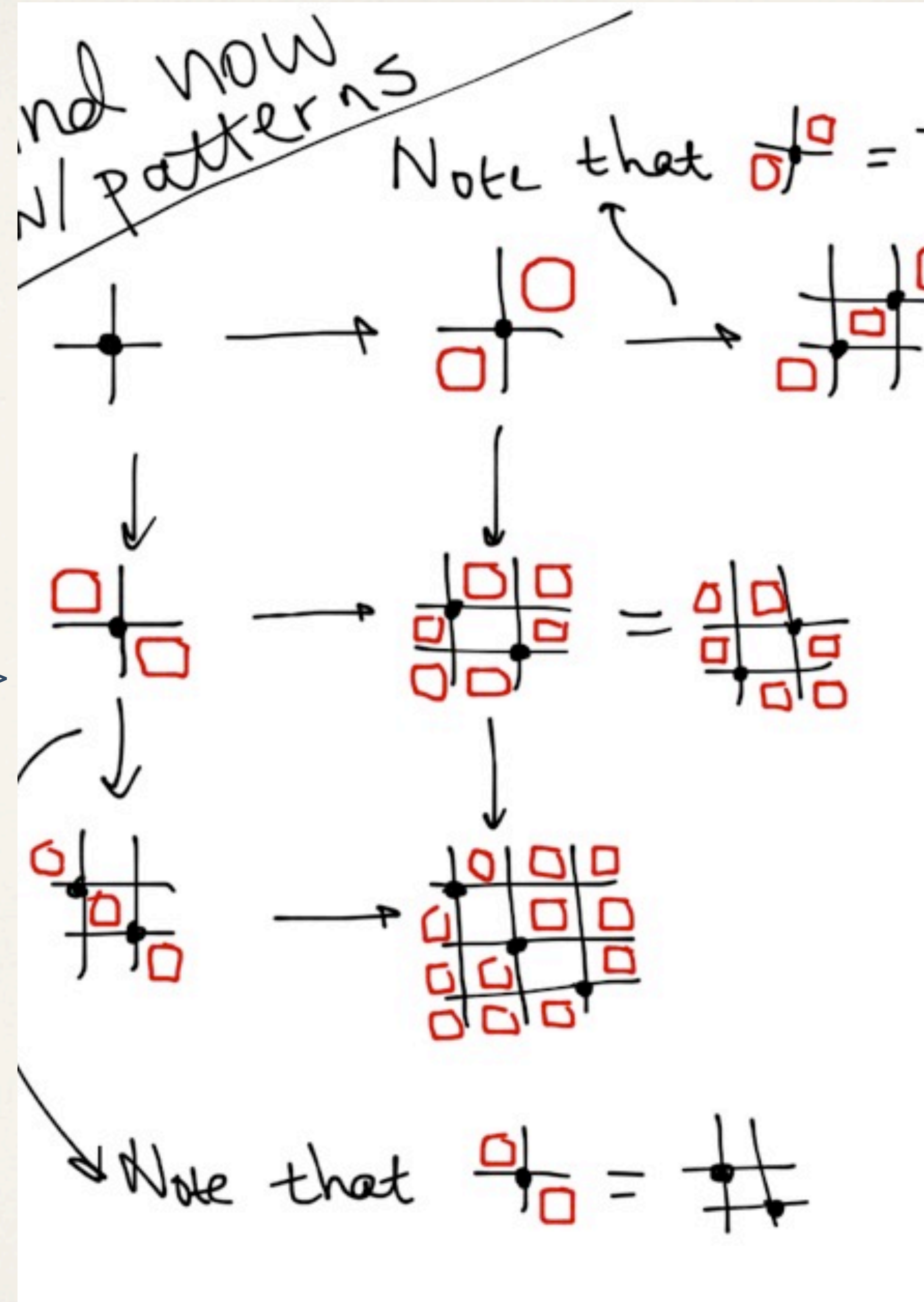

Dots must increase from red dot

# There seems to be more...

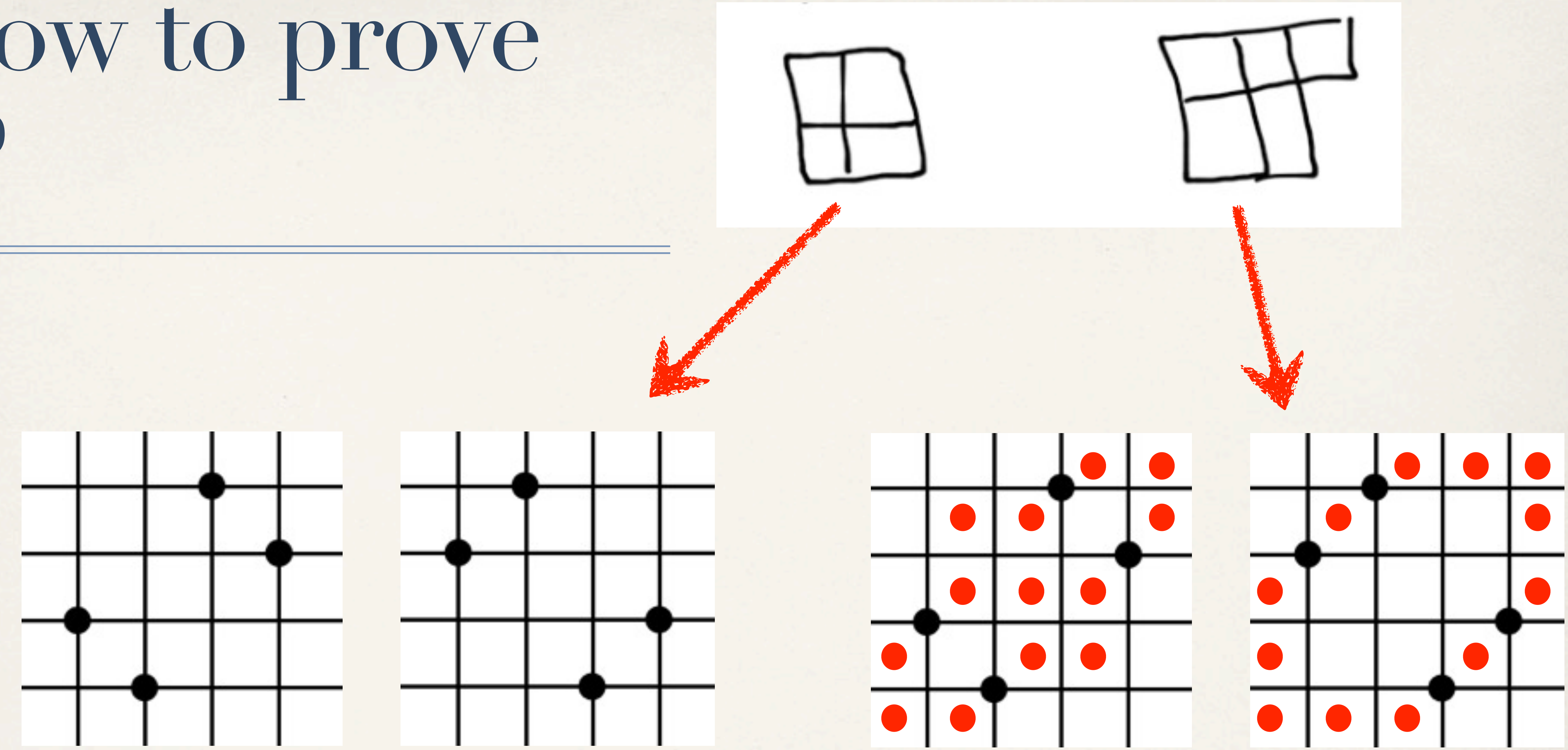
- If we create a lattice of shapes, it seems to correspond to a lattice of patterns


Perms w/ tableaux
cont. a particular sha
Growing
1st line
1st col.

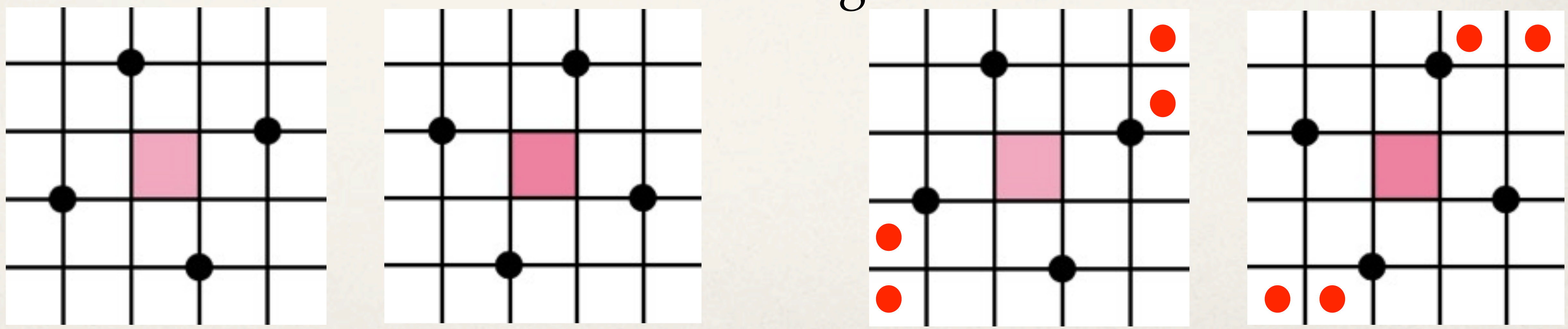
<--->


nd now
w/ patterns
Note that
Note that

# How to prove it?

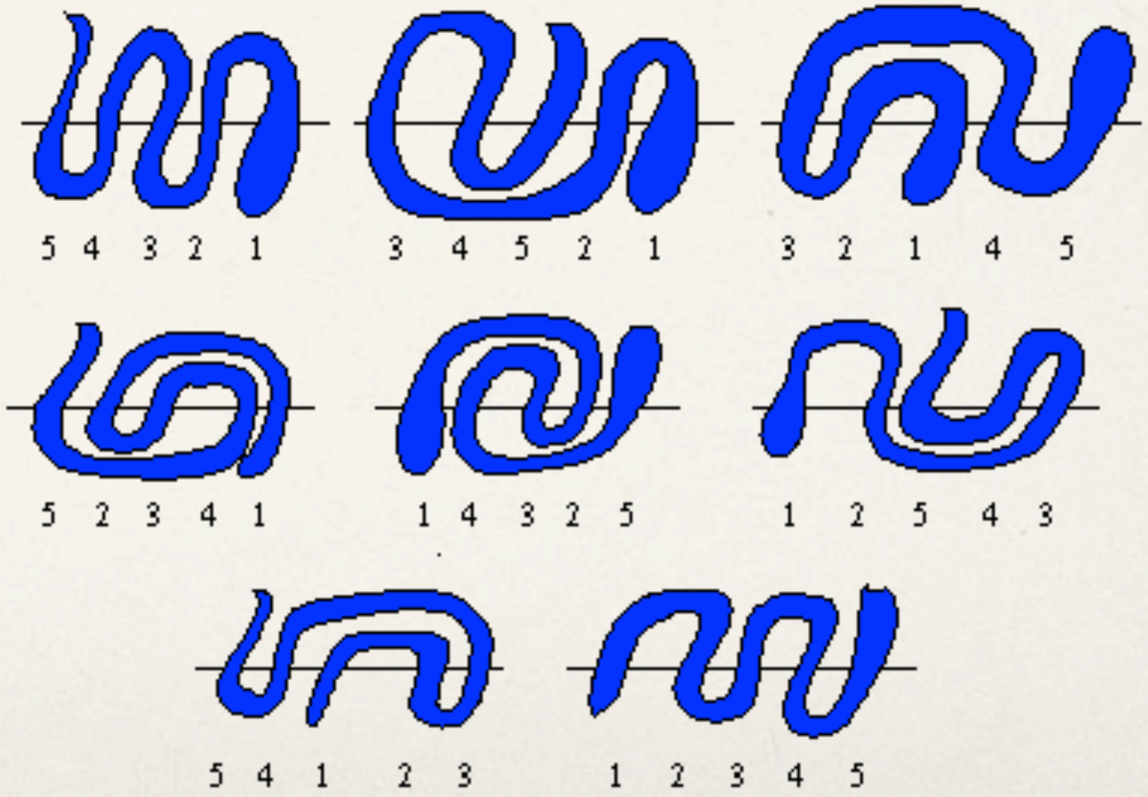


... have some ideas but an algorithm would be better



# Negativity results

- Some types of permutations are notoriously hard to describe and even to count
- Meanders are one example. These are encodings of flowing rivers

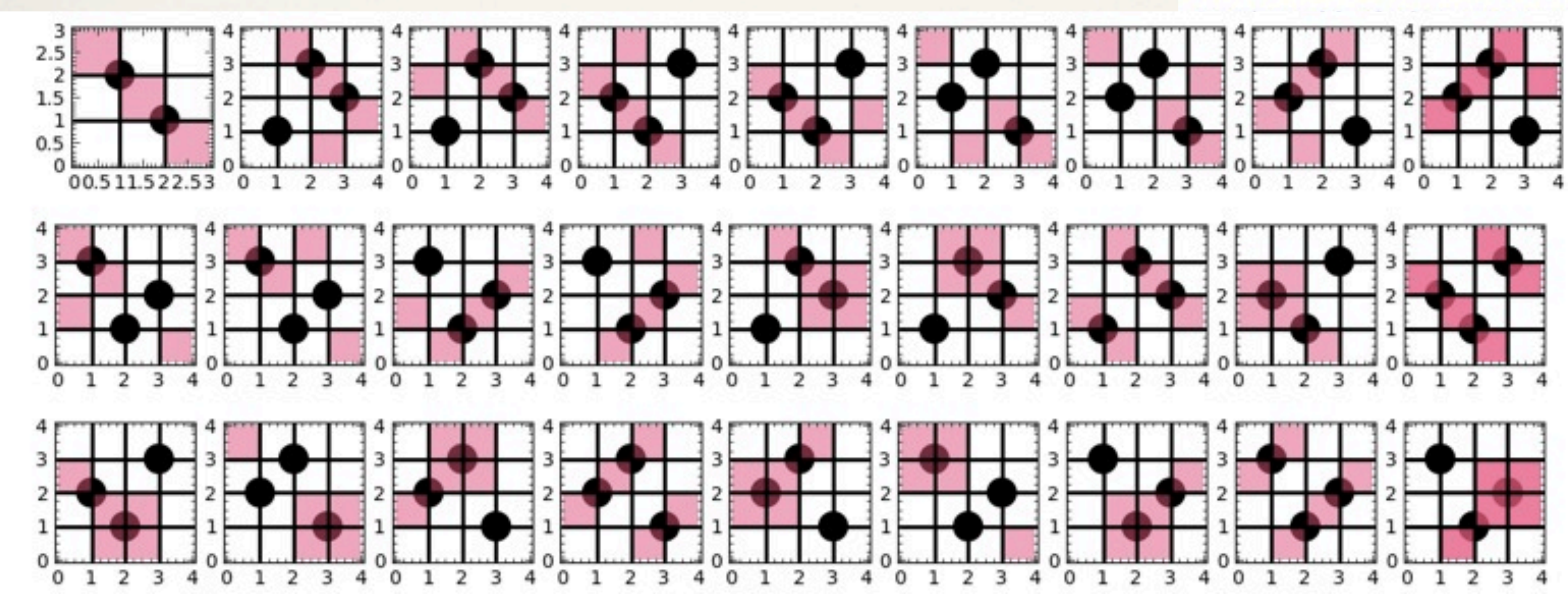


# Output from algorithm

And this does NOT describe meanders of length 4 or more!

```
%time

# Maximum length of patterns to search for
M = 3

# Maximum length of permutations from A to consider.
N = 10

# Initializing a dictionary of good patterns that will
# be learned from A
goodpatts = dict()

SG = parallel_guess(M,N,report=True,run_parallel=False)
```

```
Starting search for allowed patterns of lengths 1...3

Only need to consider patterns of lengths 2...3

Now looking at permutations of length

        1
        2
        3
        4
        5
        6
        7
        8
        9
        10

Done

The number of allowed patterns of length 2 is 4
The number of allowed patterns of length 3 is 74

 allowed patterns

 of length 2 is now 4
 of length 3 is now 74

 patterns of lengths 1...3

 length 1 is 0
 length 2 is 1
 length 3 is 207

rbidden patterns

), (1, 1), (0, 2)})
 0), (1, 3), (3, 1), (2, 2)})
 3), (3, 1), (0, 2), (2, 2)})
 0), (1, 3), (1, 1), (0, 2)})
 0), (3, 1), (1, 1), (0, 2)})
 0), (0, 3), (1, 0), (2, 1)})
 0), (0, 3), (3, 2), (2, 1)})
 2), (0, 1), (1, 0), (2, 3)})
 2), (0, 1), (3, 2), (2, 3)})
 0), (1, 2), (0, 3), (0, 1)})
 0), (1, 2), (0, 3), (2, 3)})
```



# Questions?

- Input more datasets of permutations, generate more conjectures
- Try it for other properties of permutations (instead of patterns)
- Try it for other types of data (instead of permutations)
(Can it discover Kuratowski's thm for graphs?)
- Can it be made into a theorem prover, instead of just conjecturer?
Have another algorithm that can prove special cases (also joint with Anders)