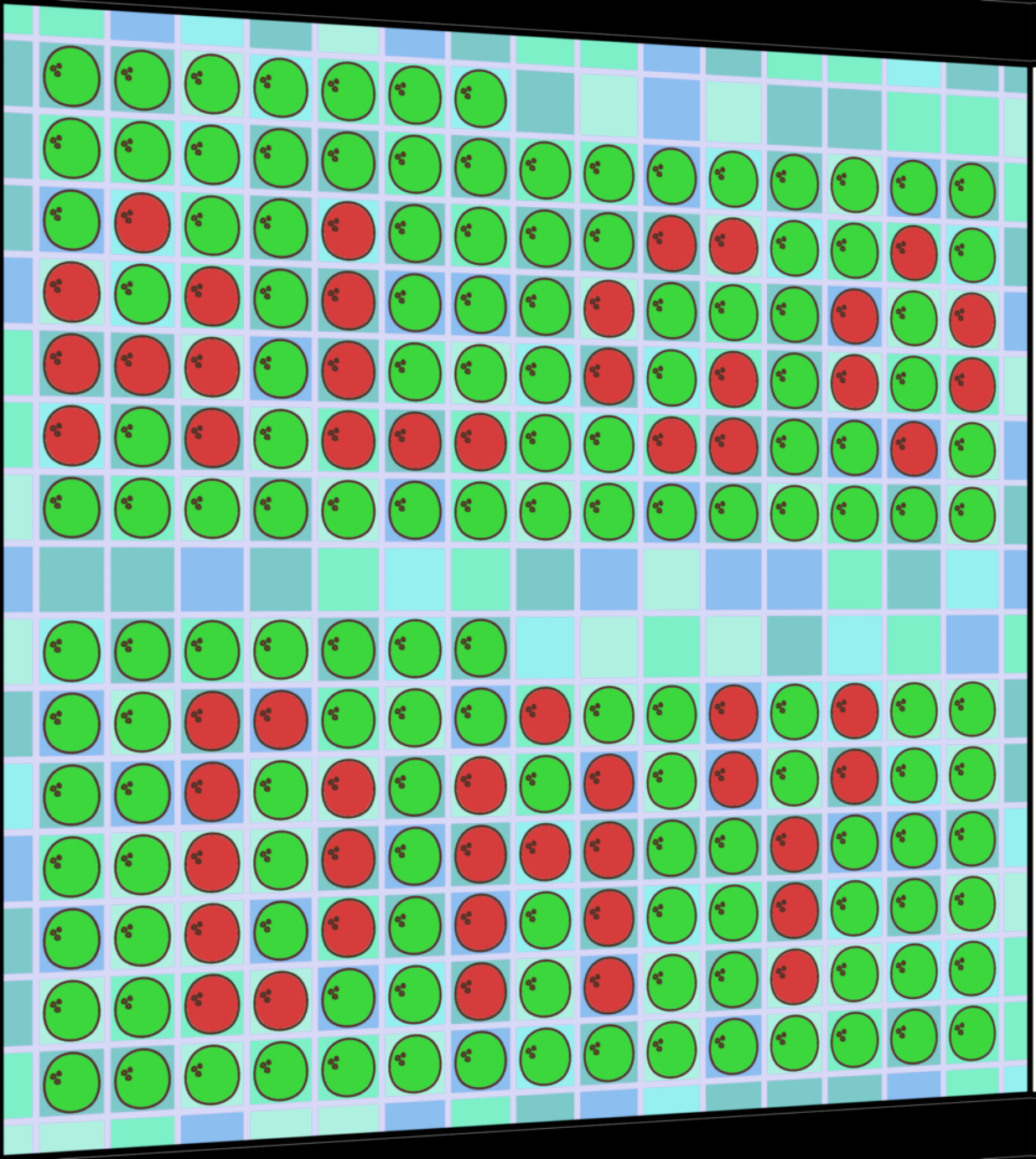
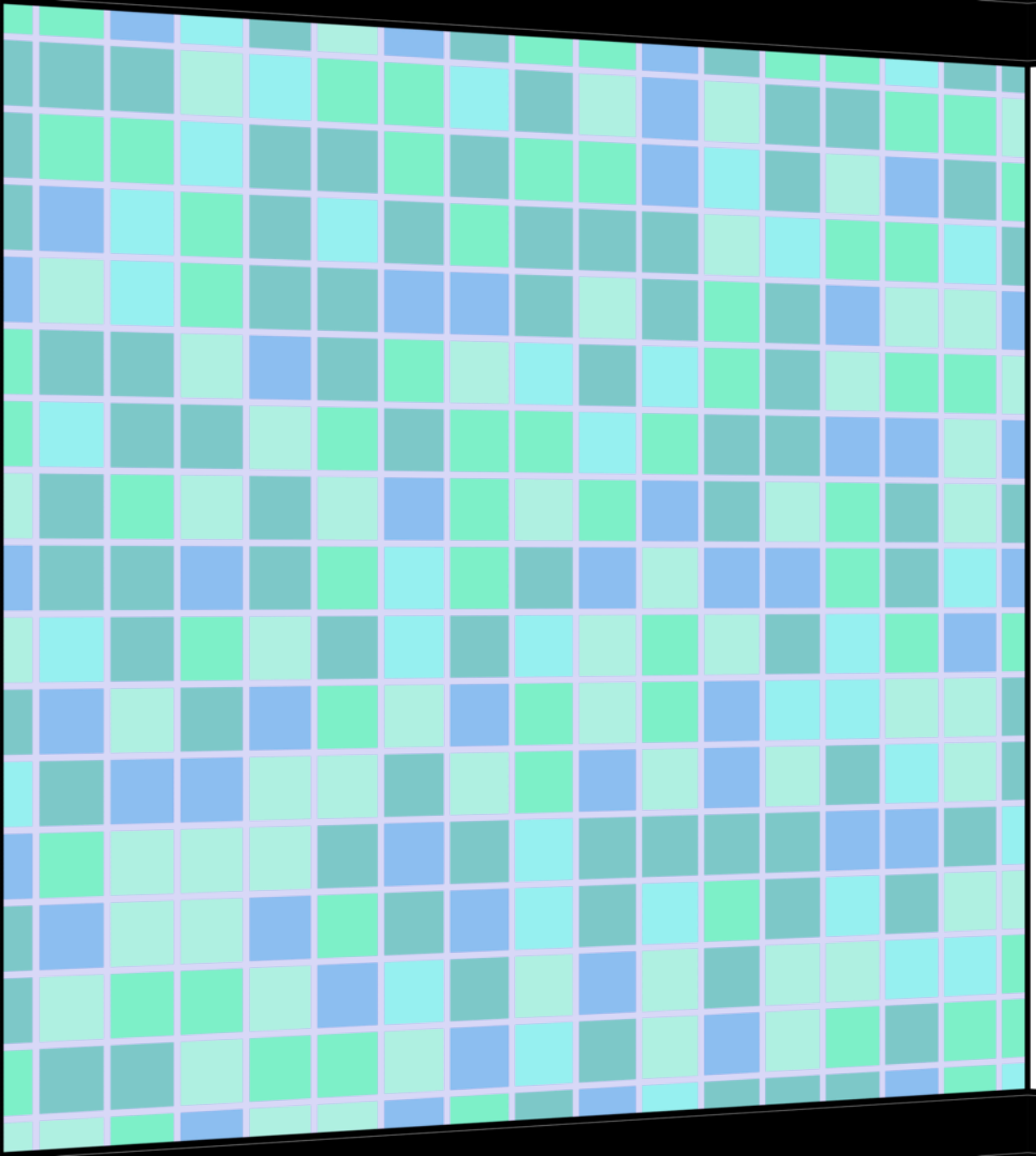


Labeled Reconfiguration by Compaction

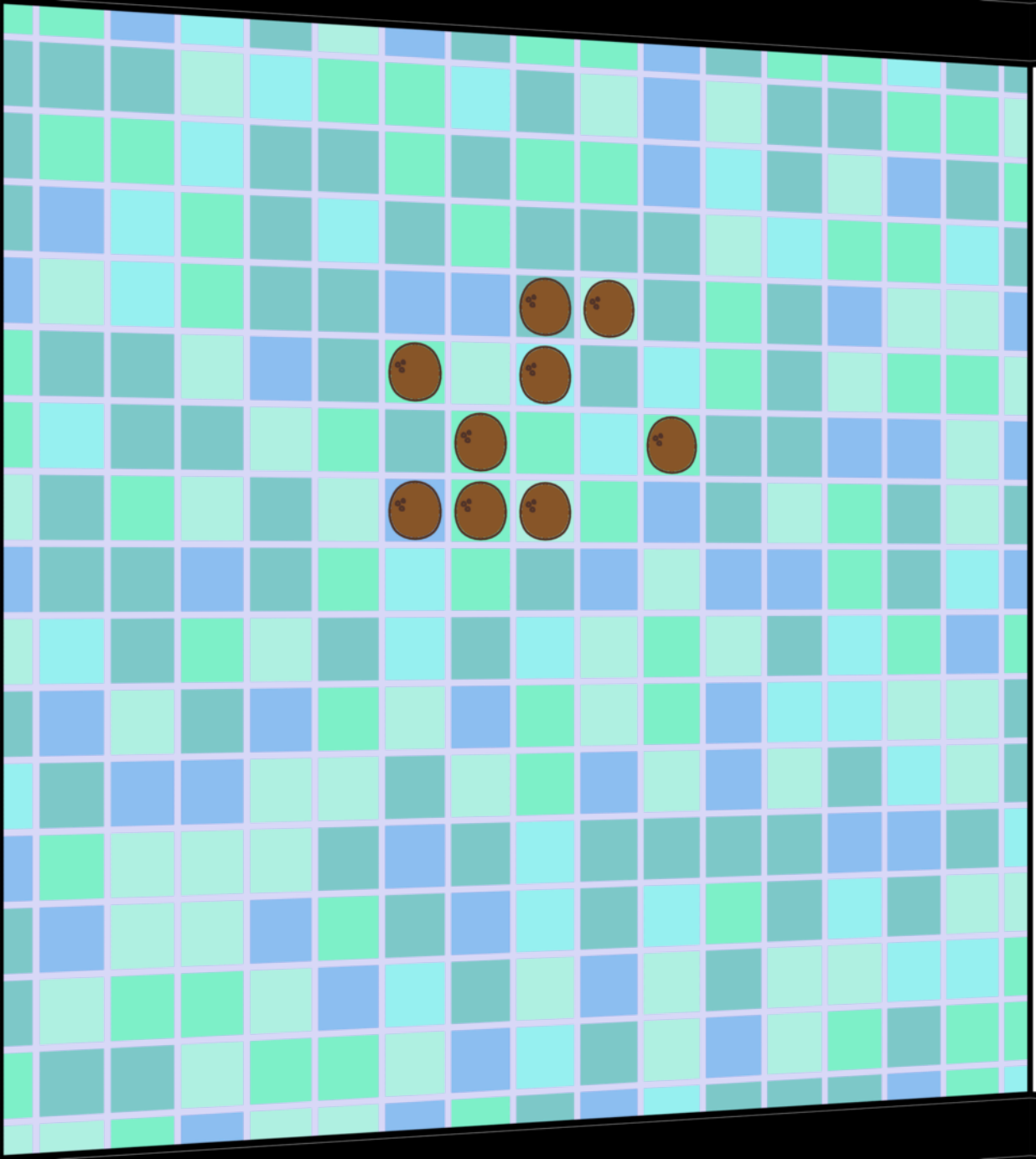
Maarten Löffler

Based on joint work with

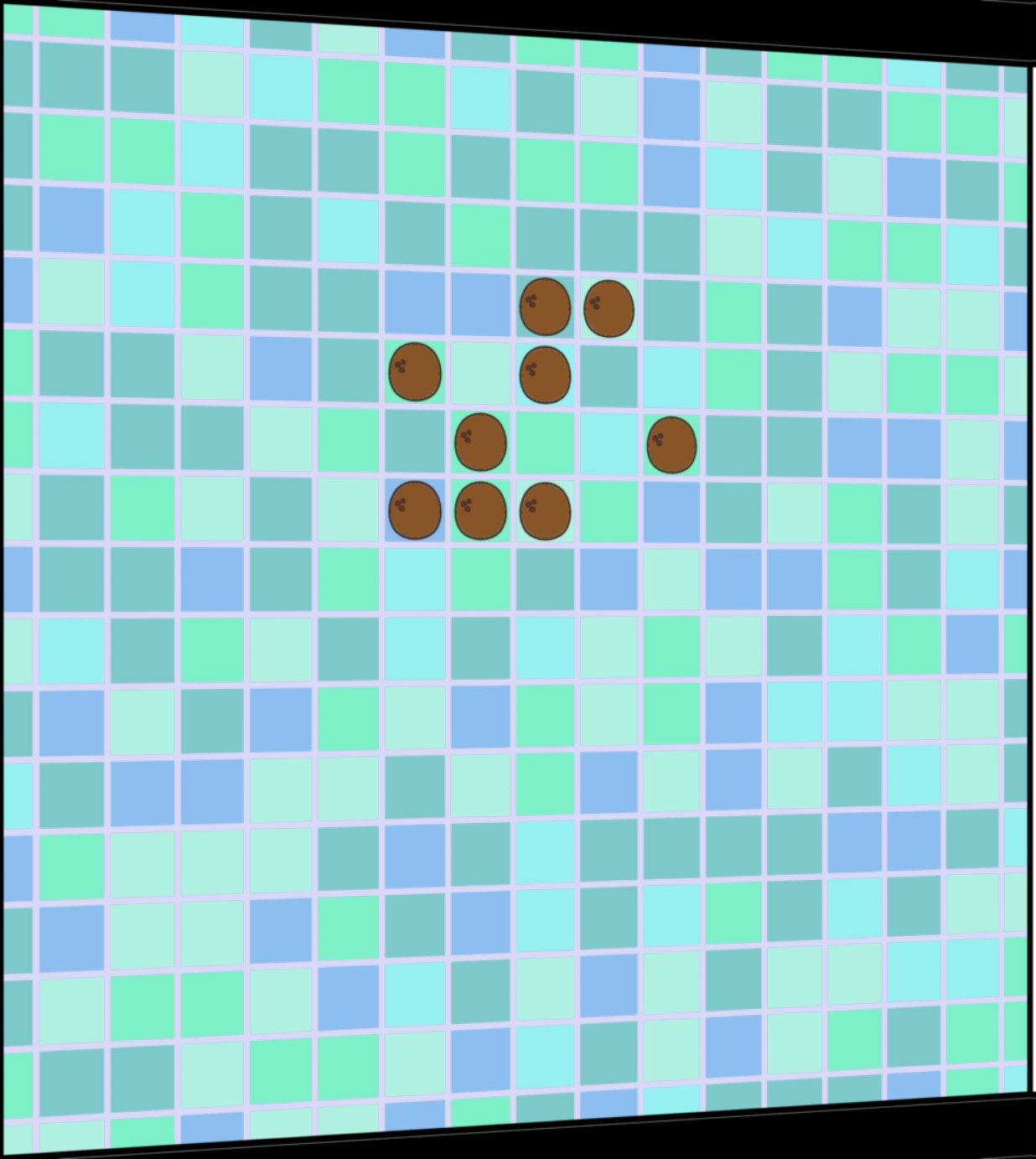
Hugo Akitaya
Greg Aloupis
Anika Rounds
Giovanni Viglietta



Let P be a set of n coconuts in a pool.

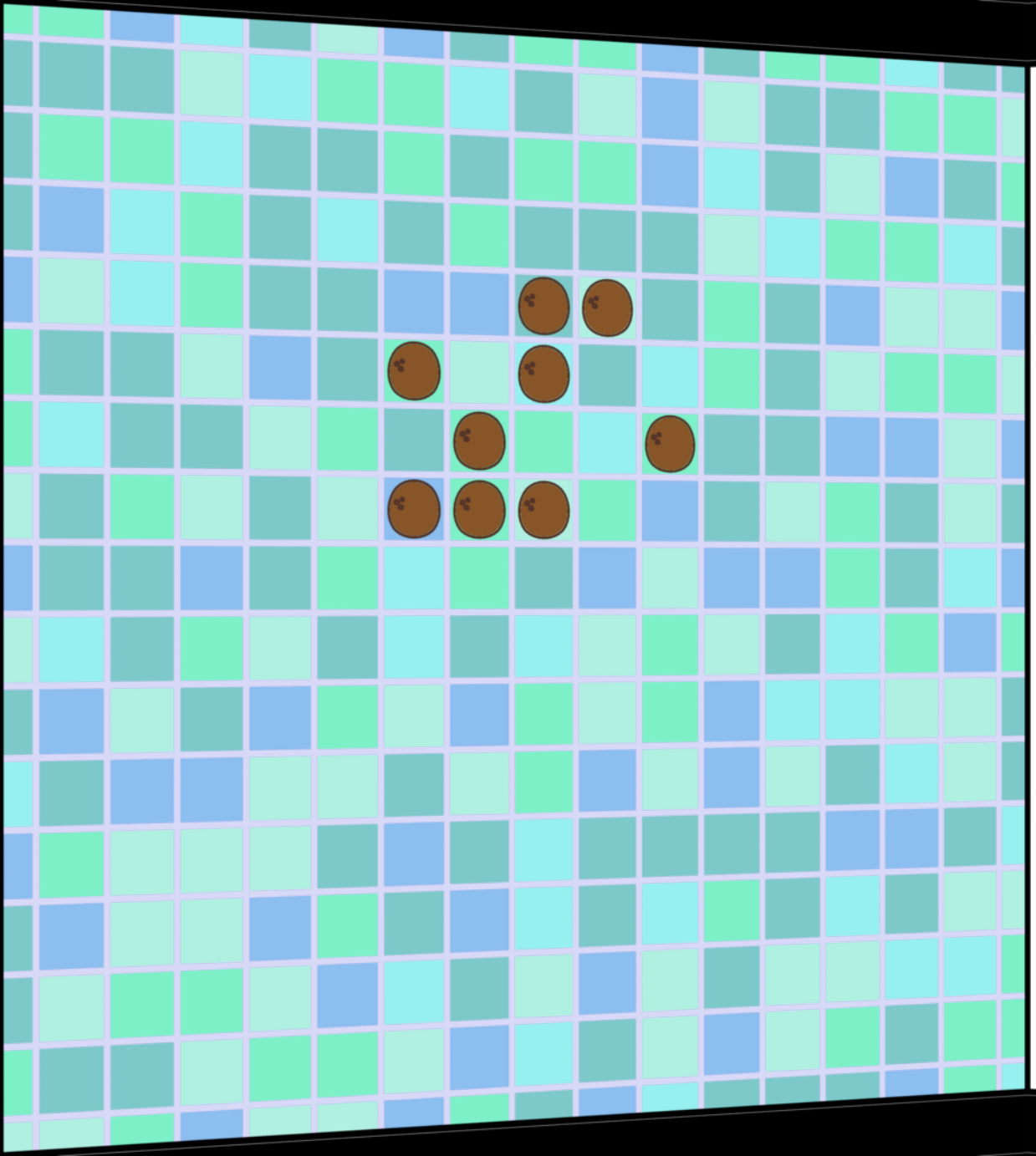


Let P be a set of n coconuts in a pool.



Let P be a set of n coconuts in a pool.

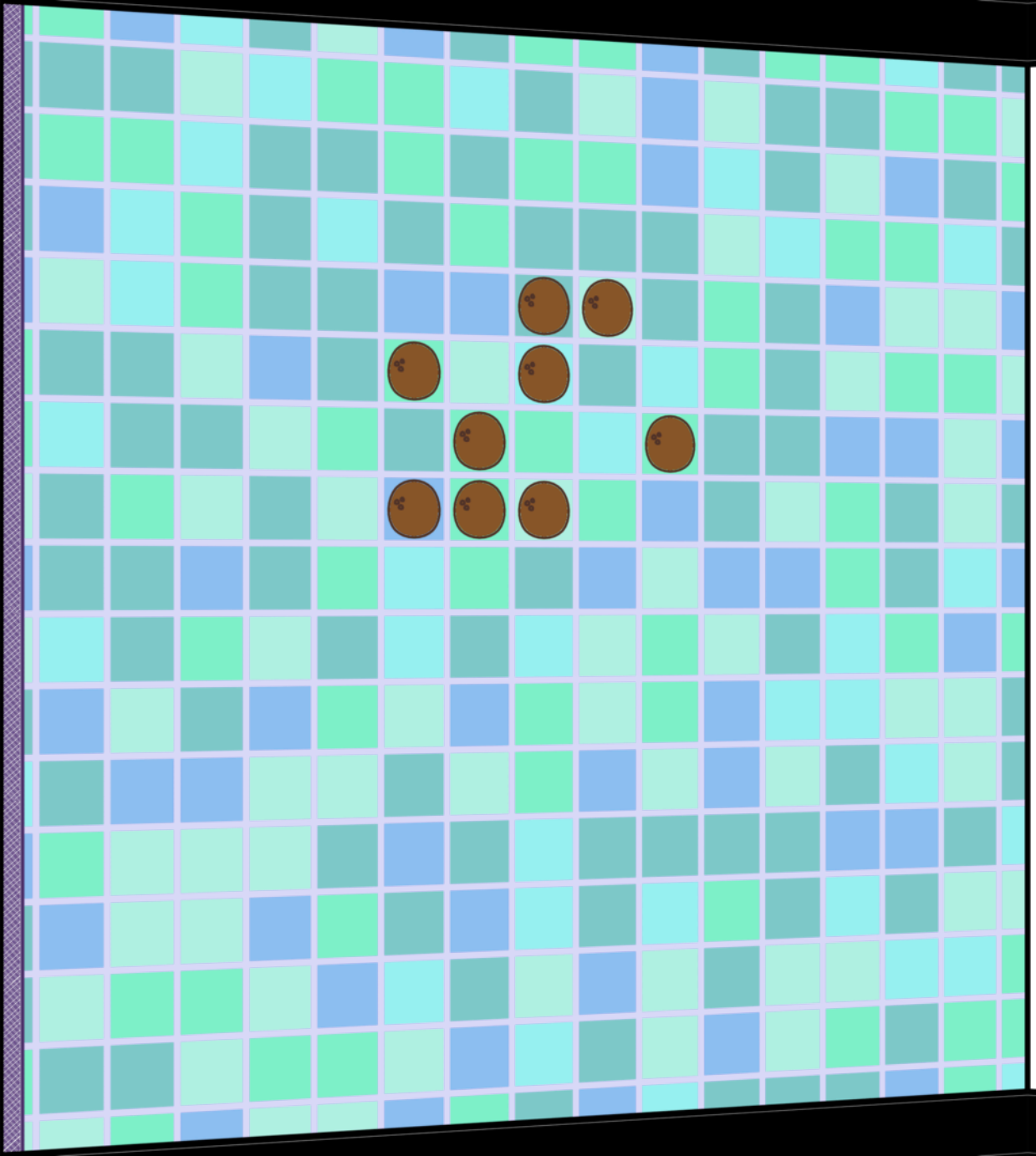
We assume the coconuts are aligned to a grid.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

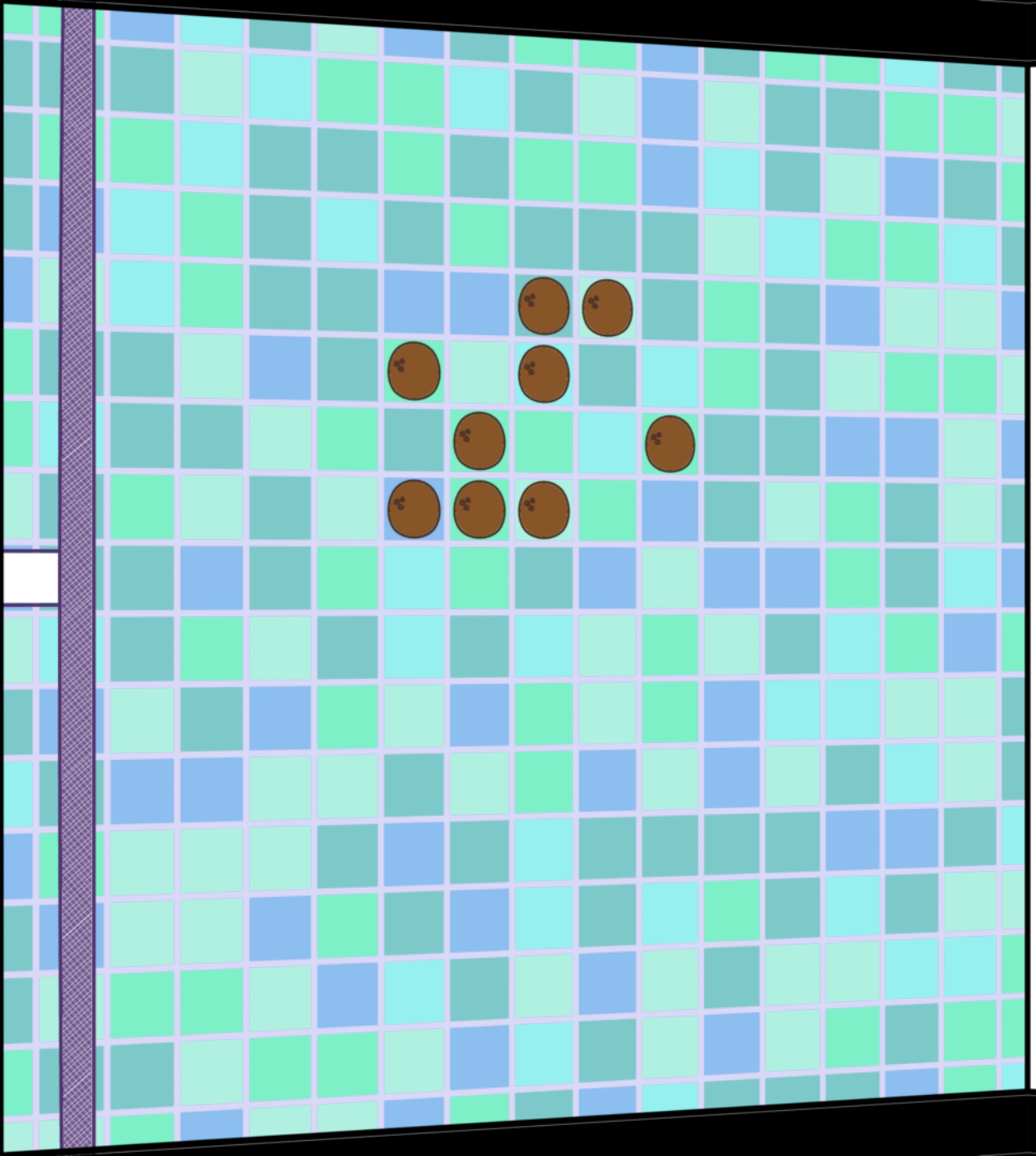
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

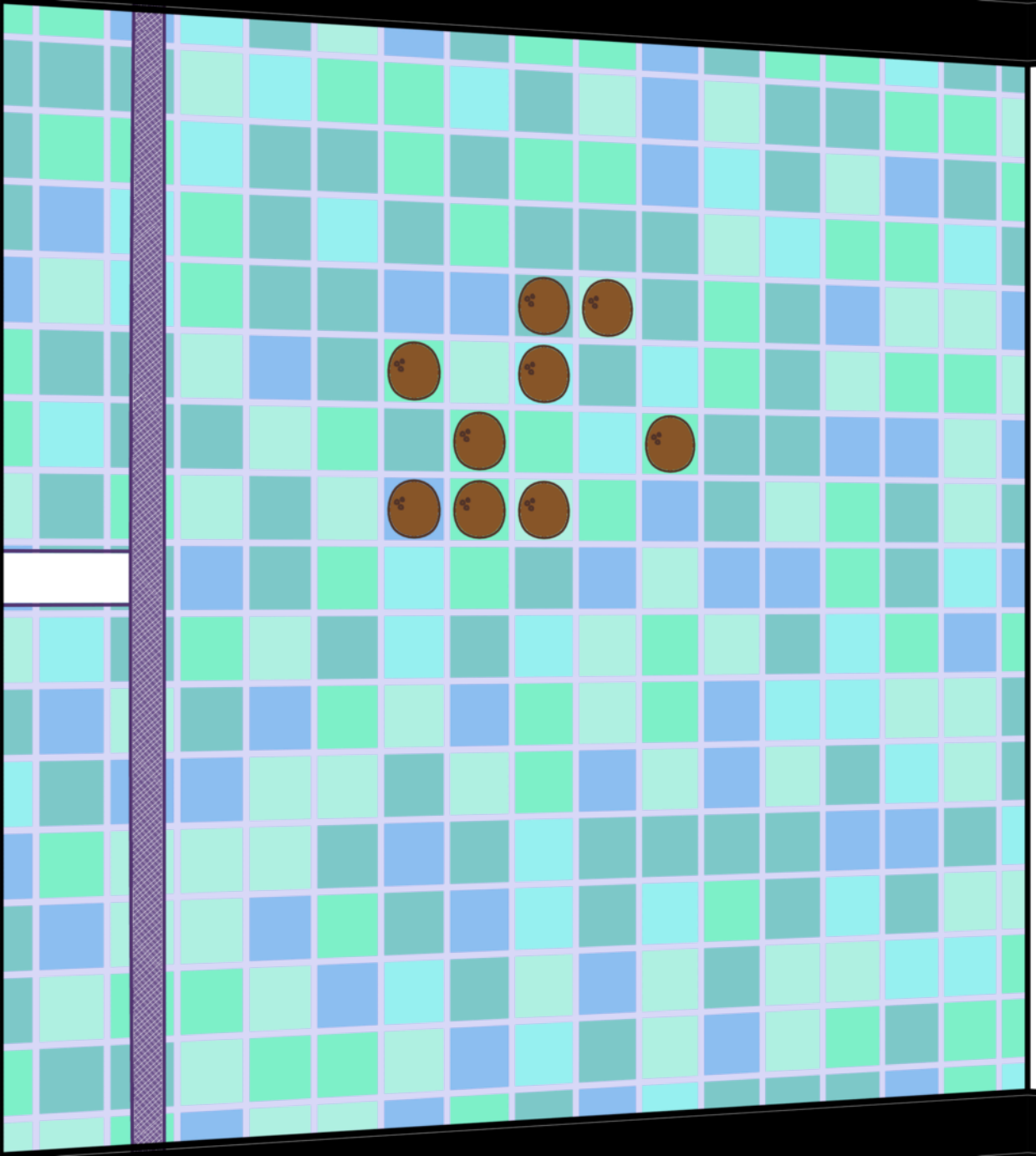
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

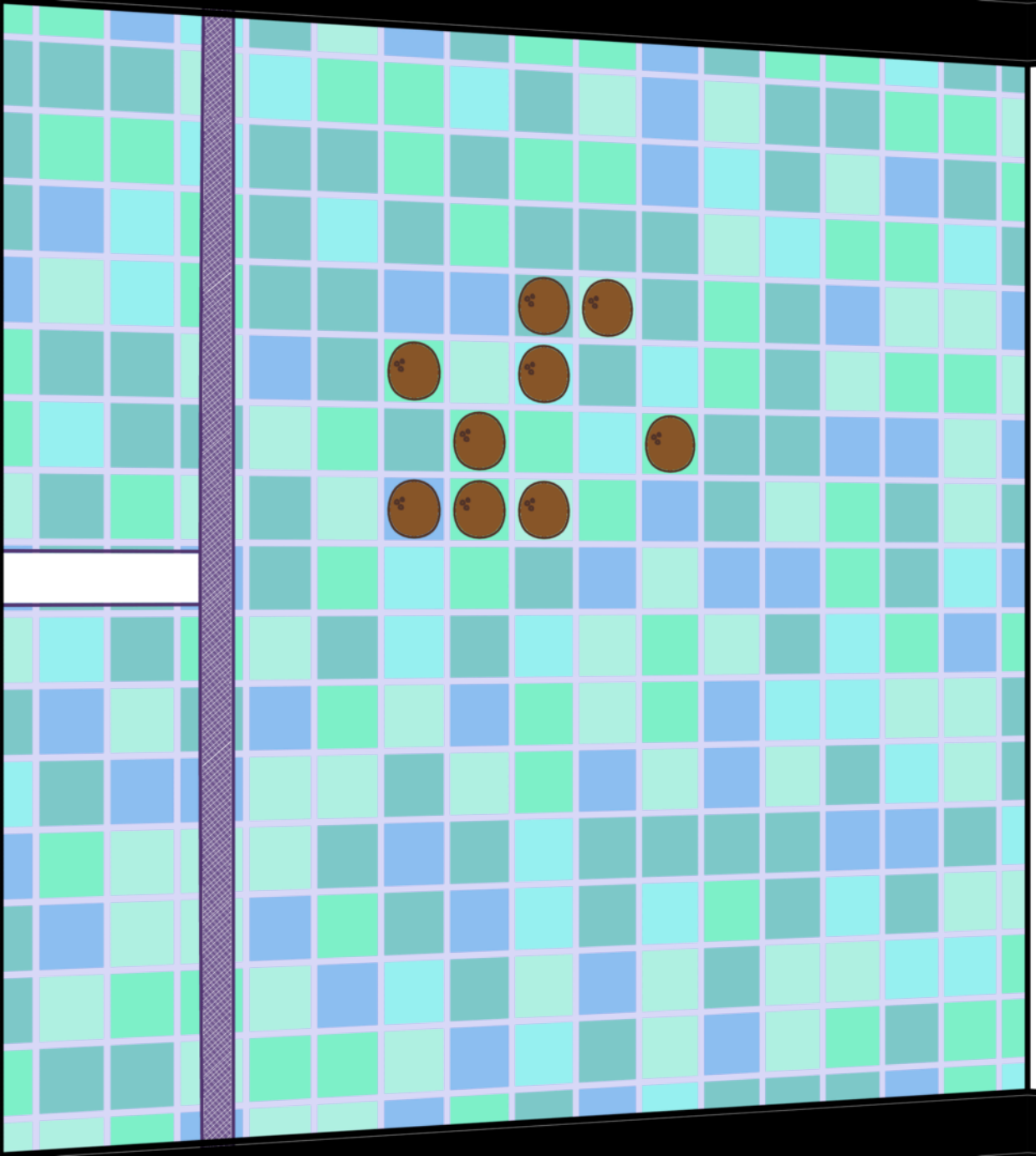
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

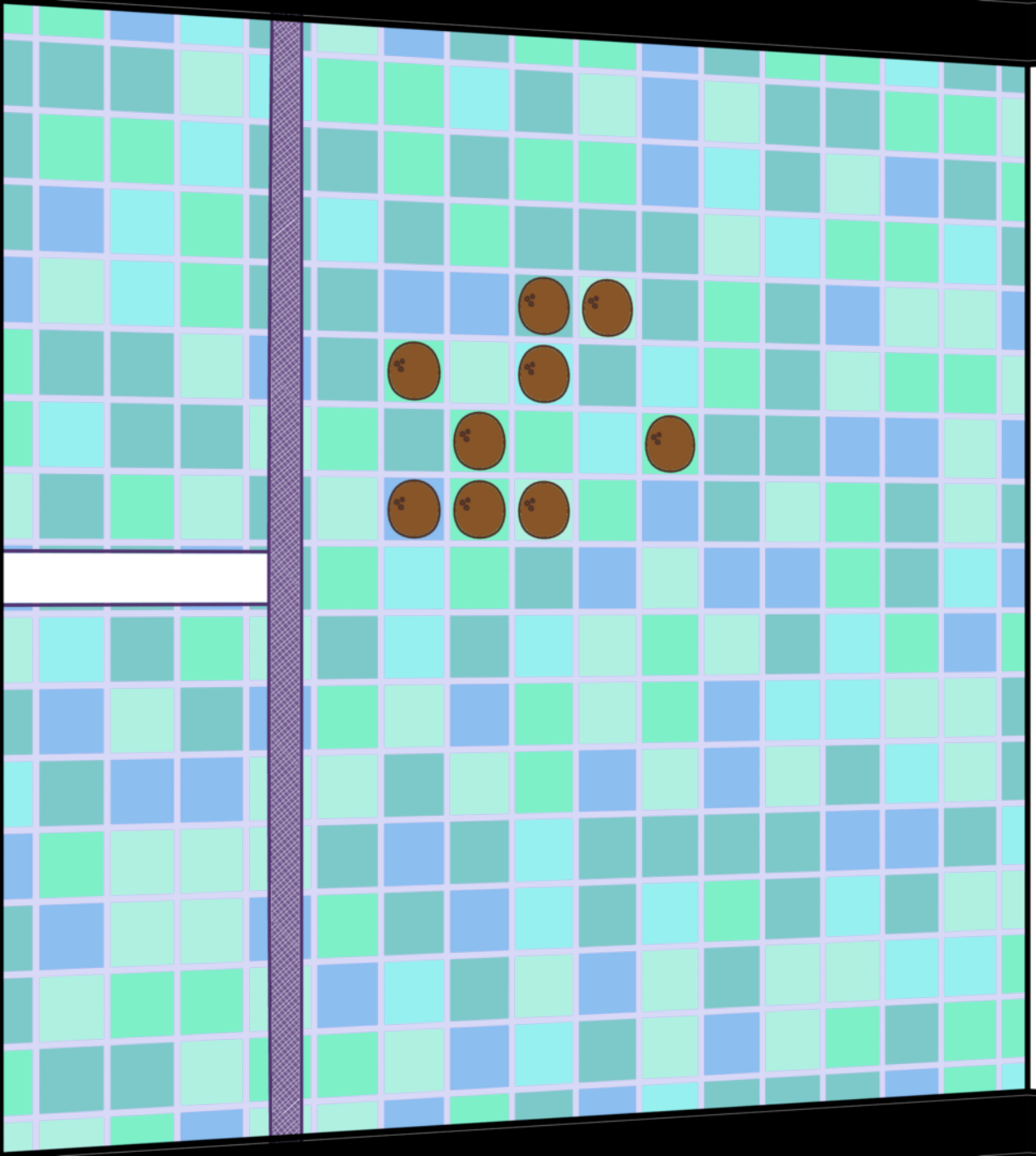
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

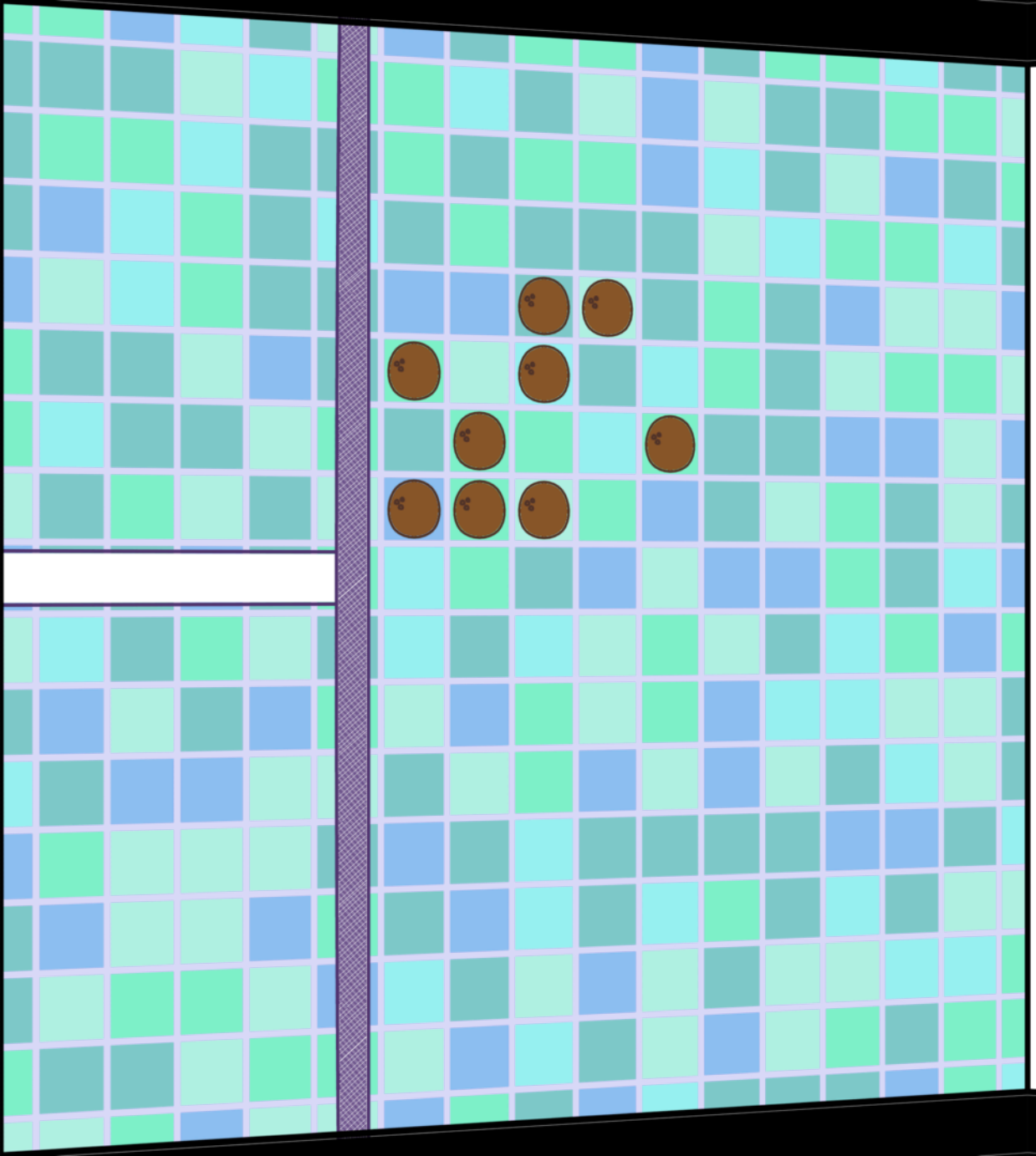
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

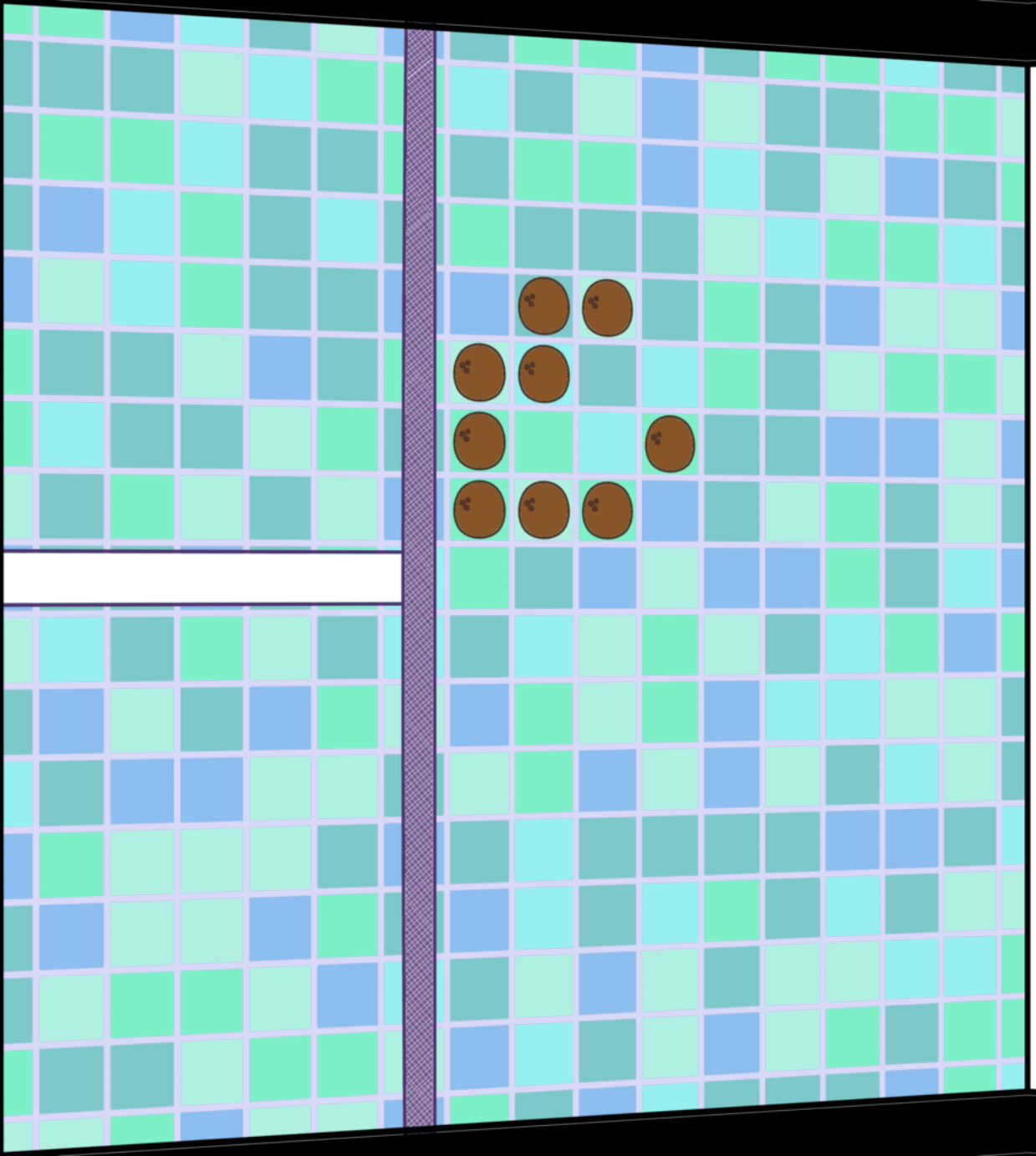
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

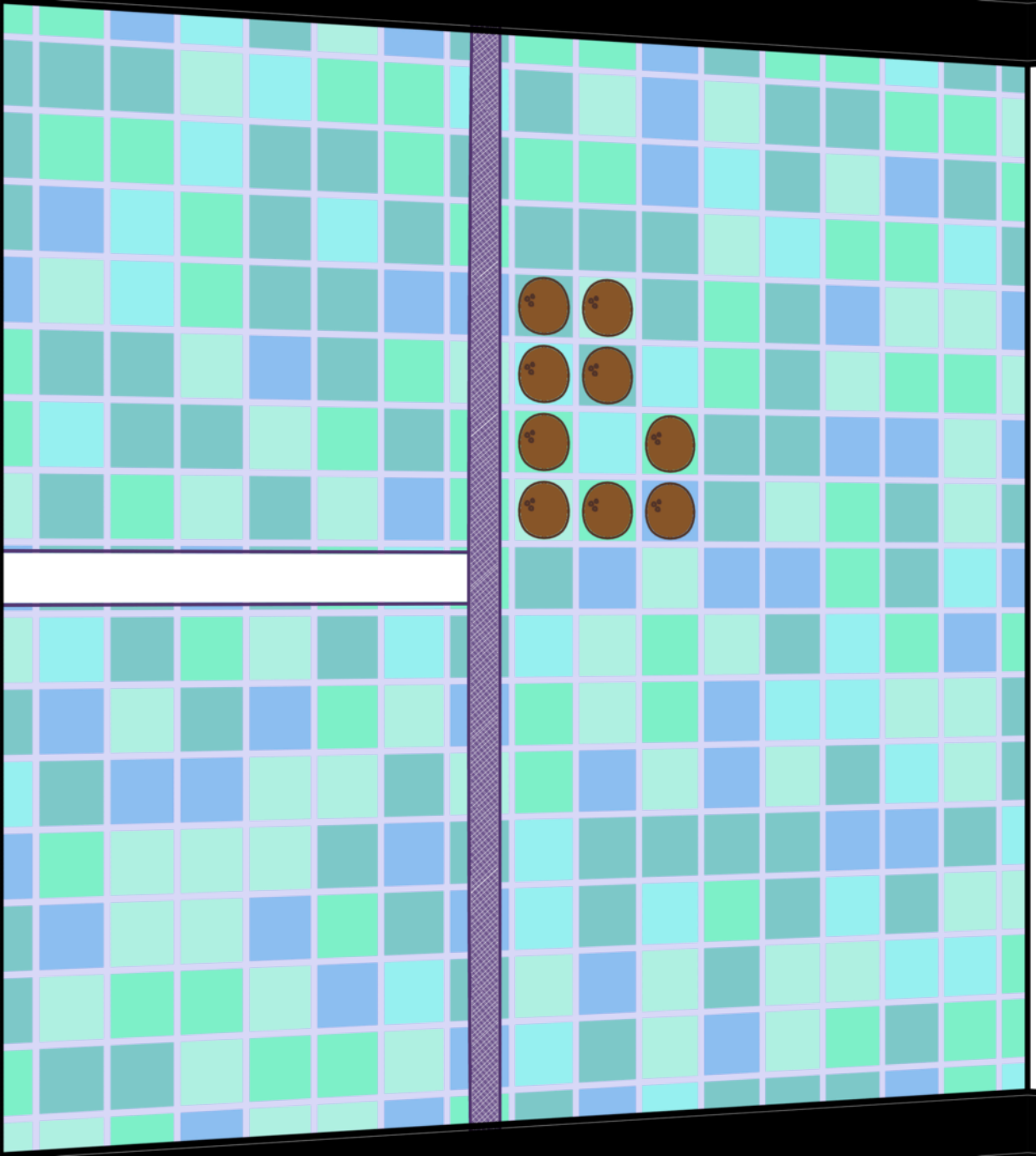
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

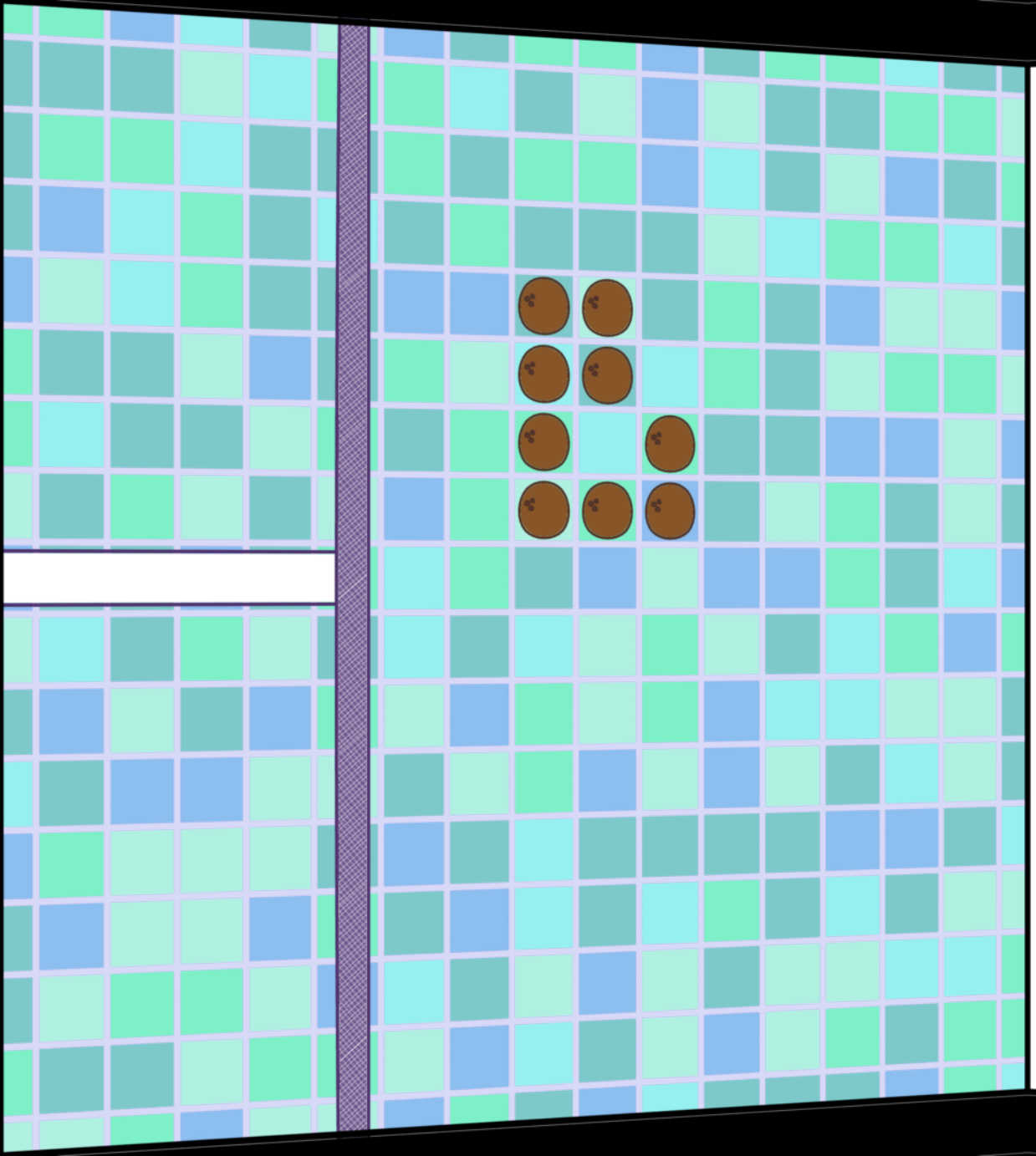
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

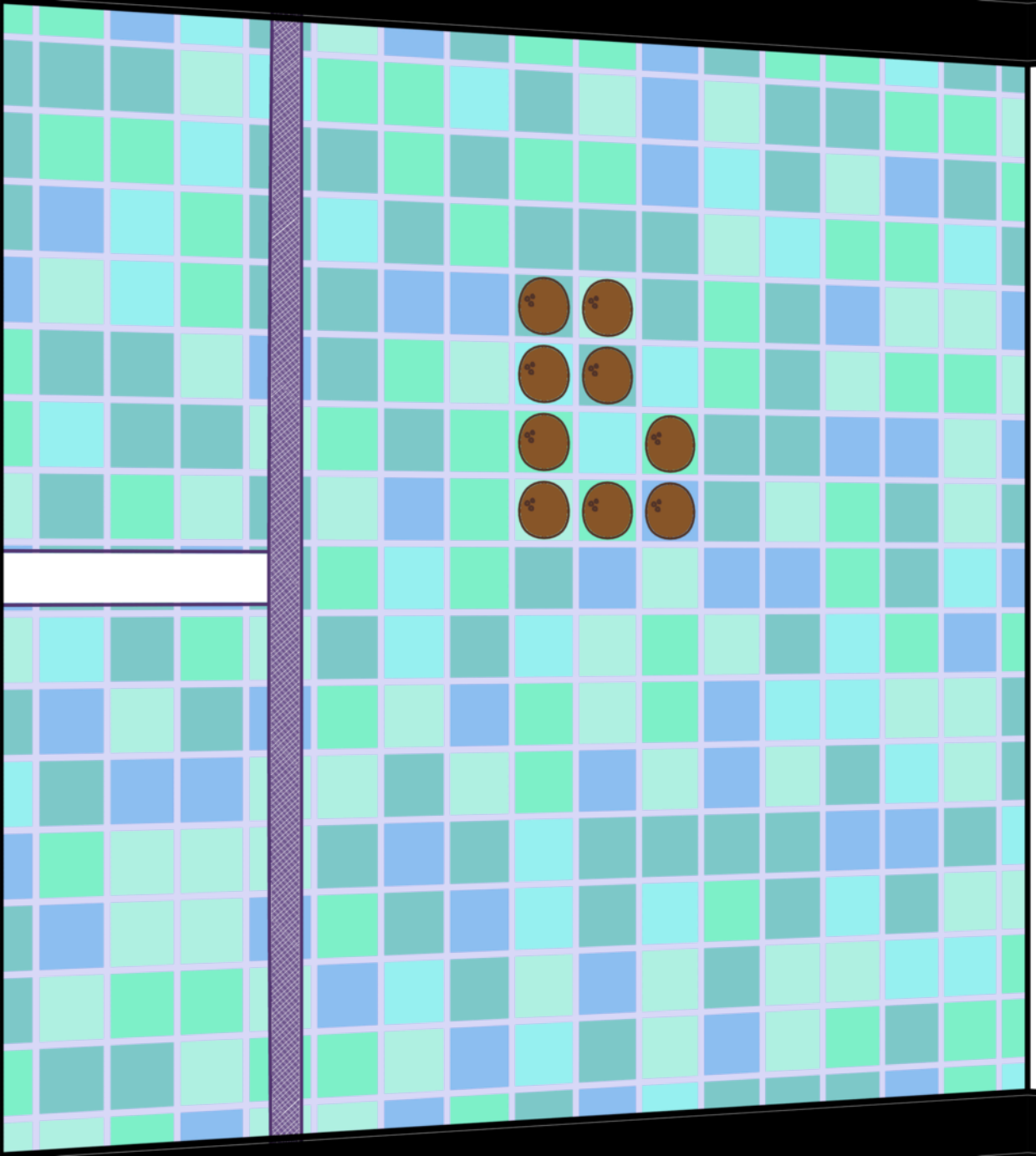
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

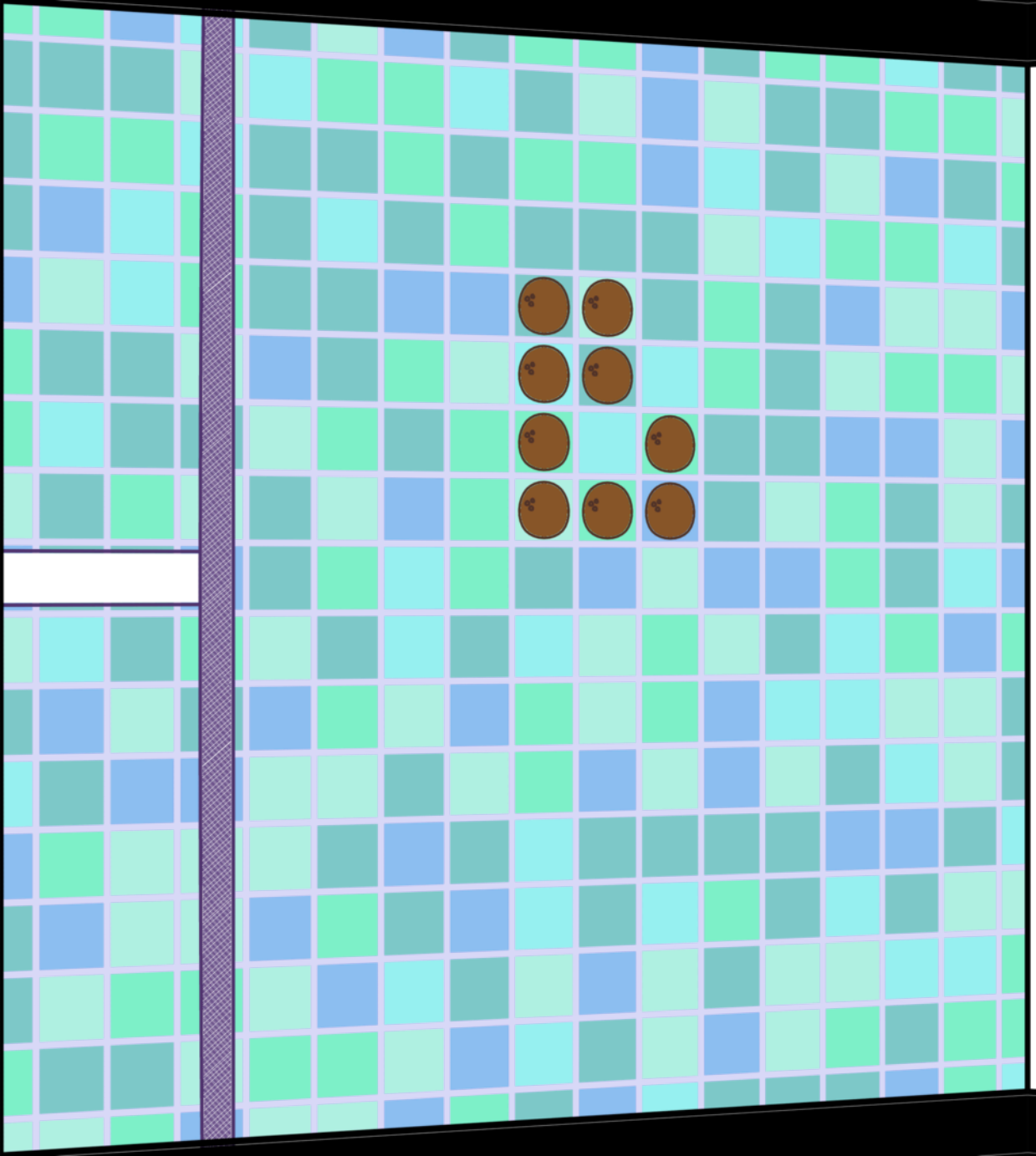
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

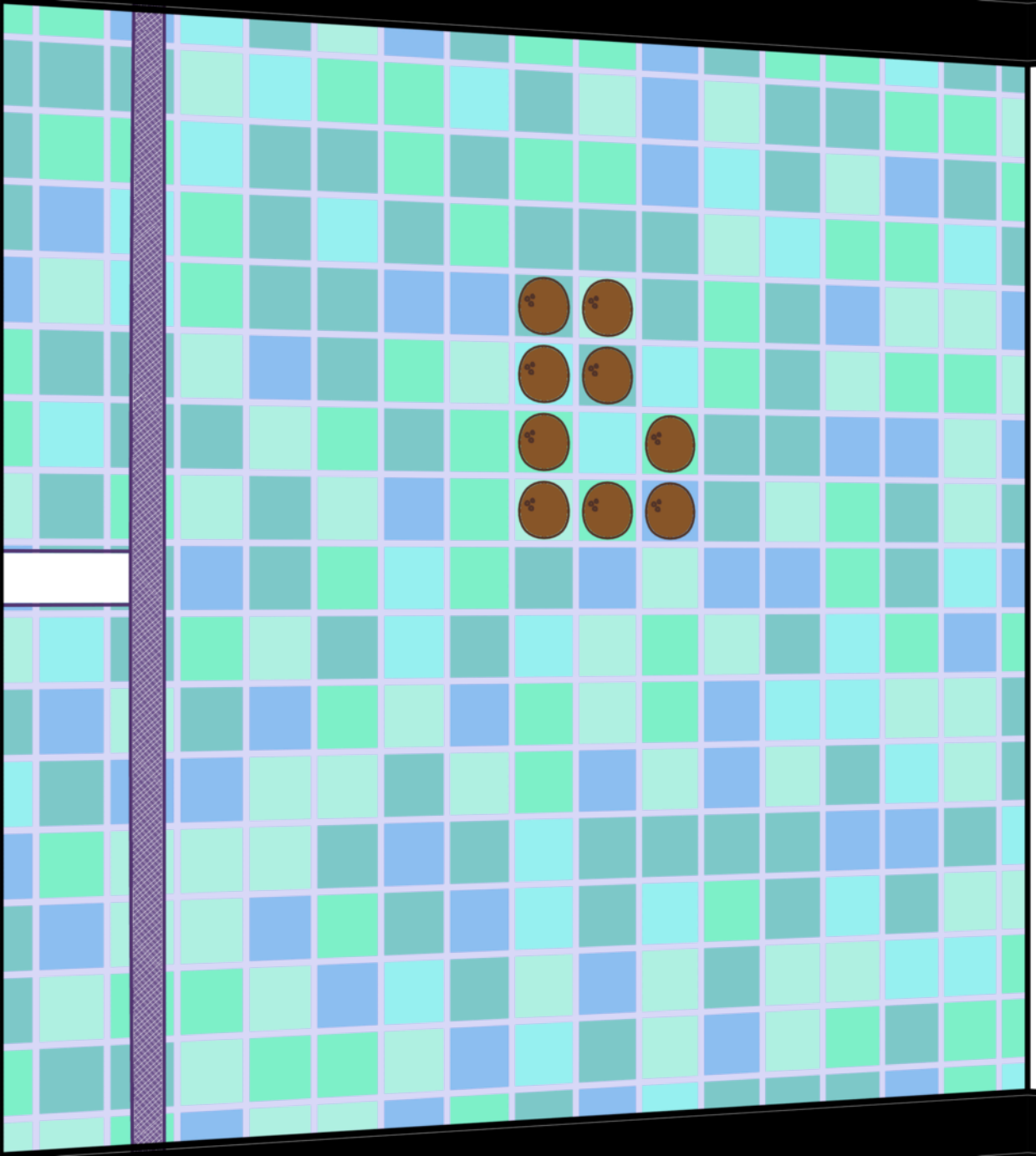
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

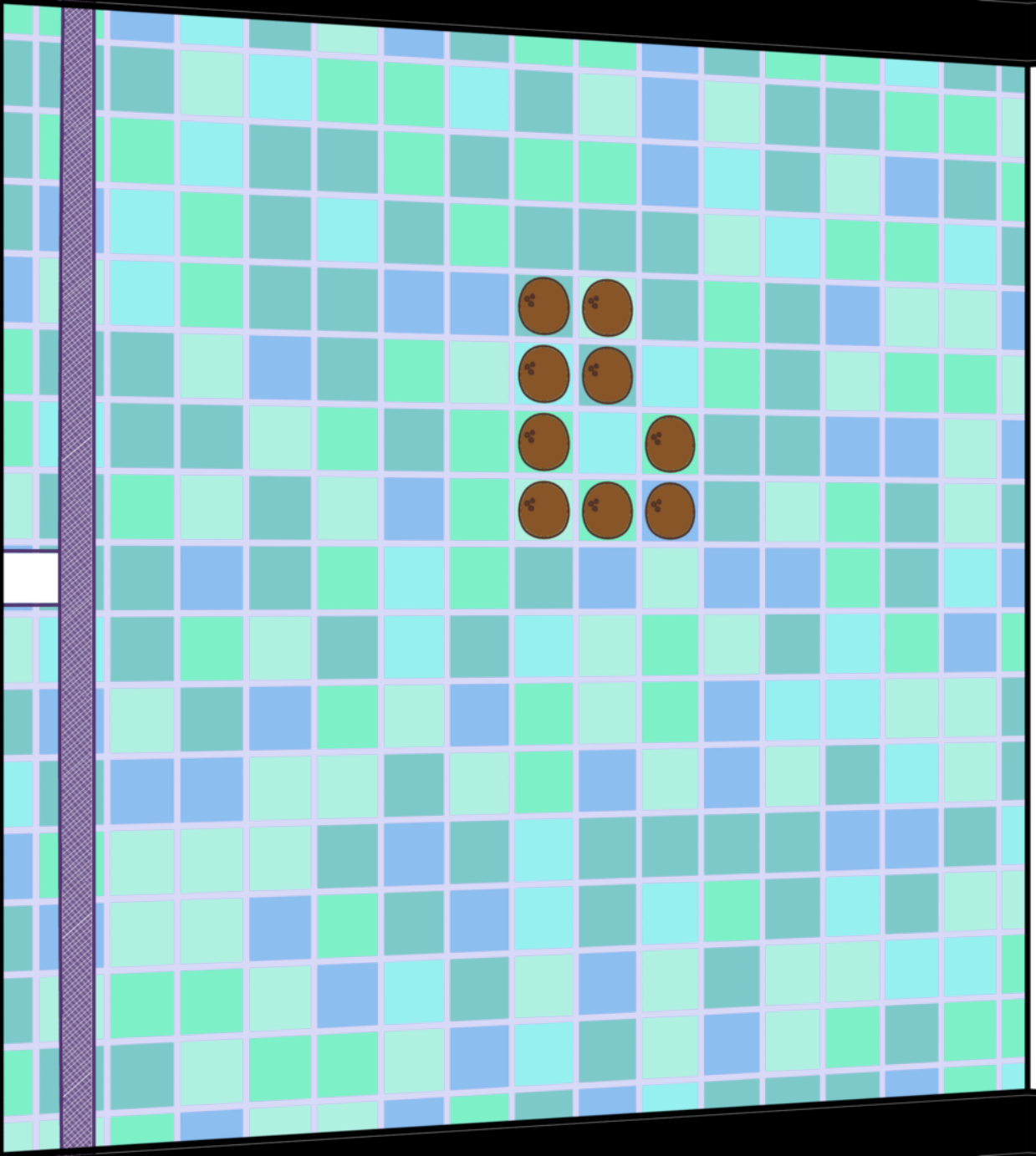
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

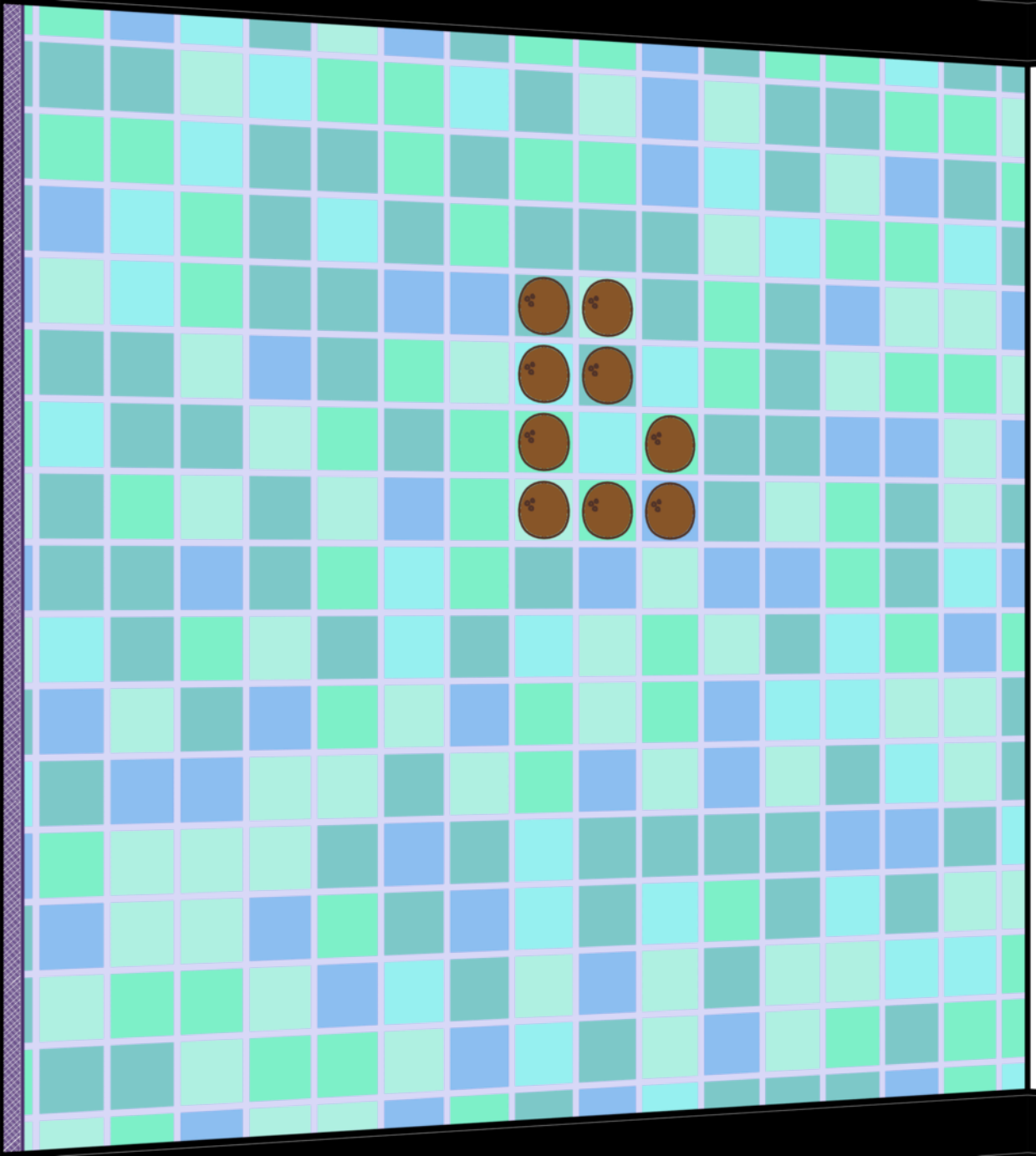
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

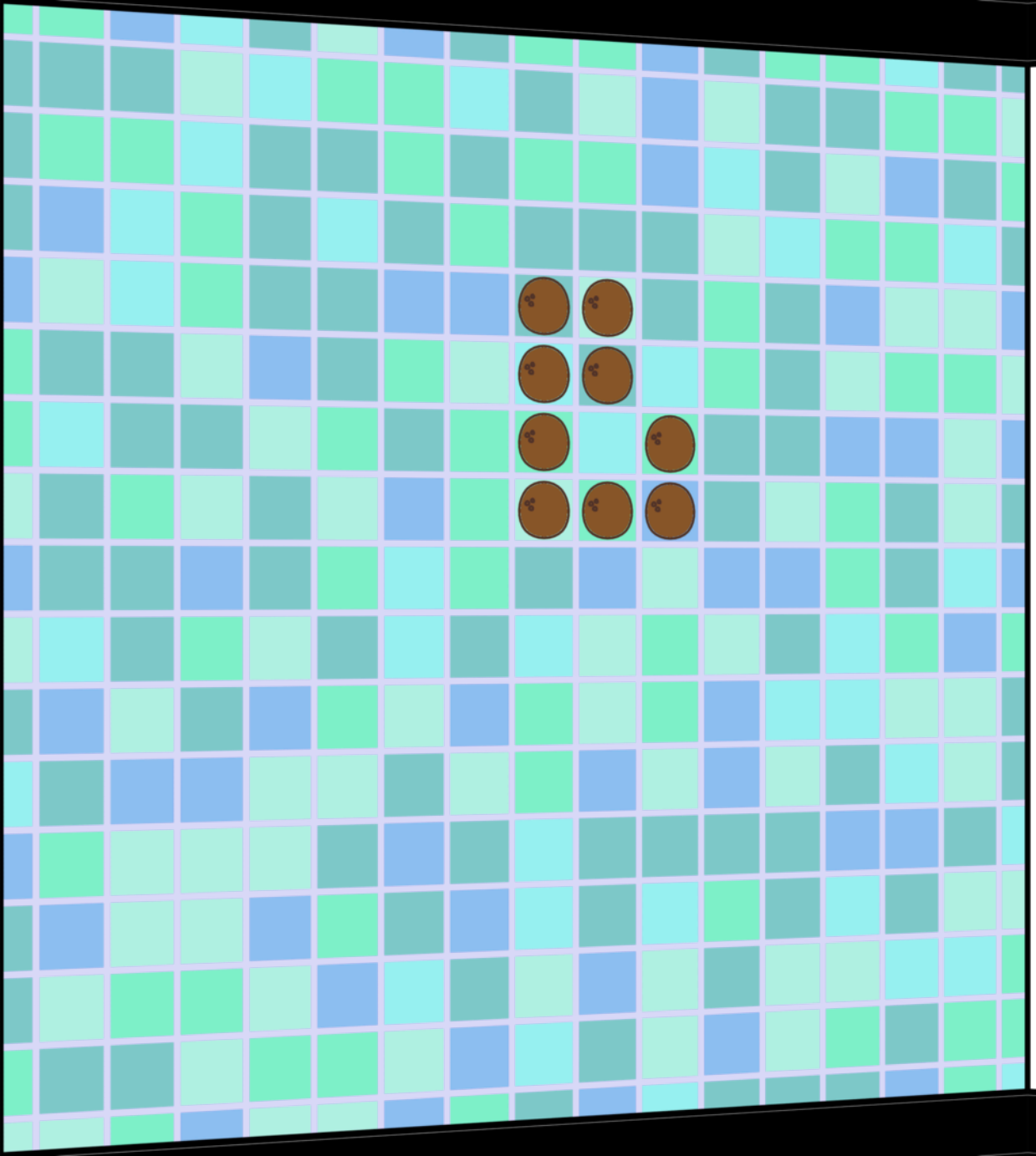
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

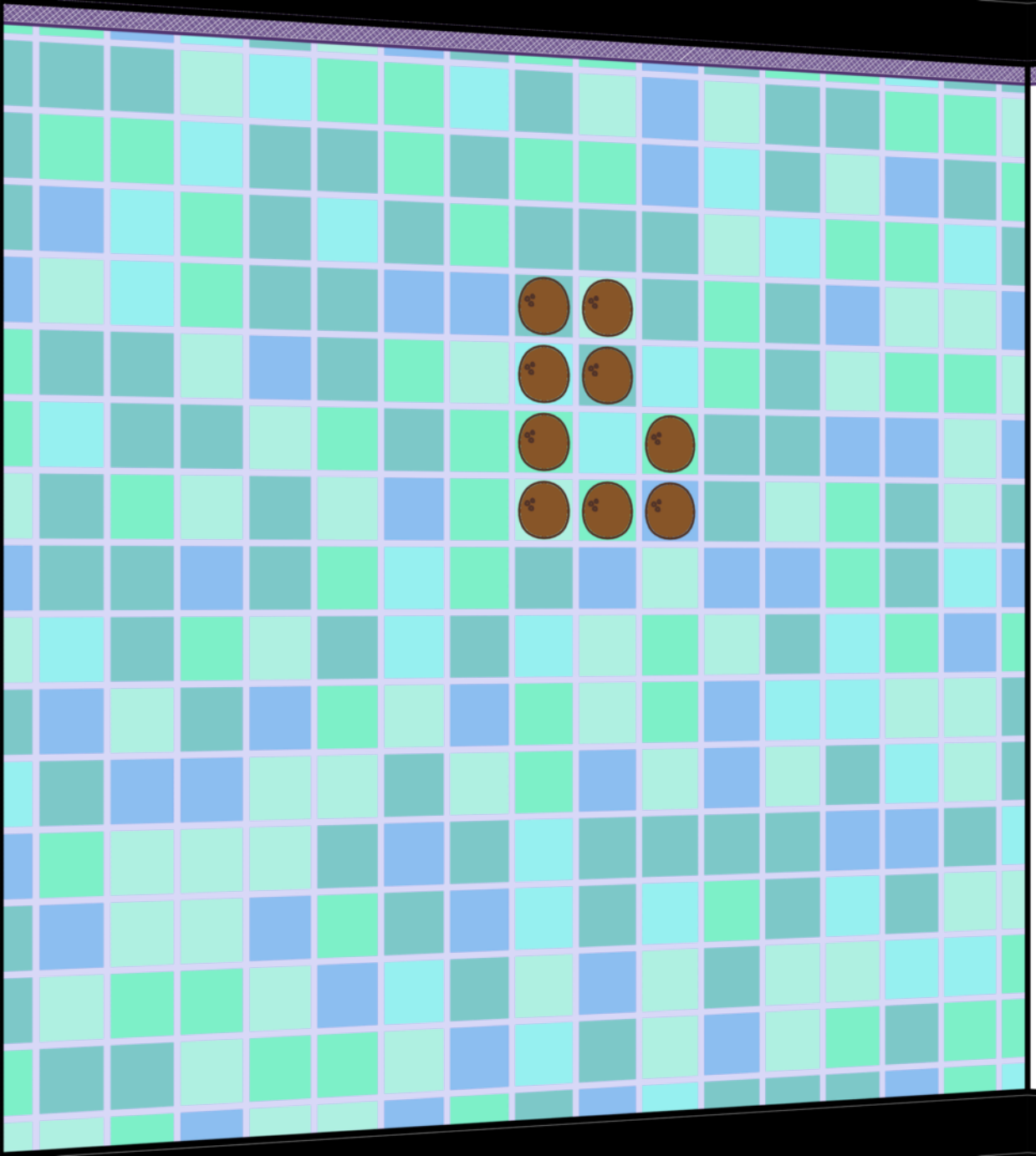
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

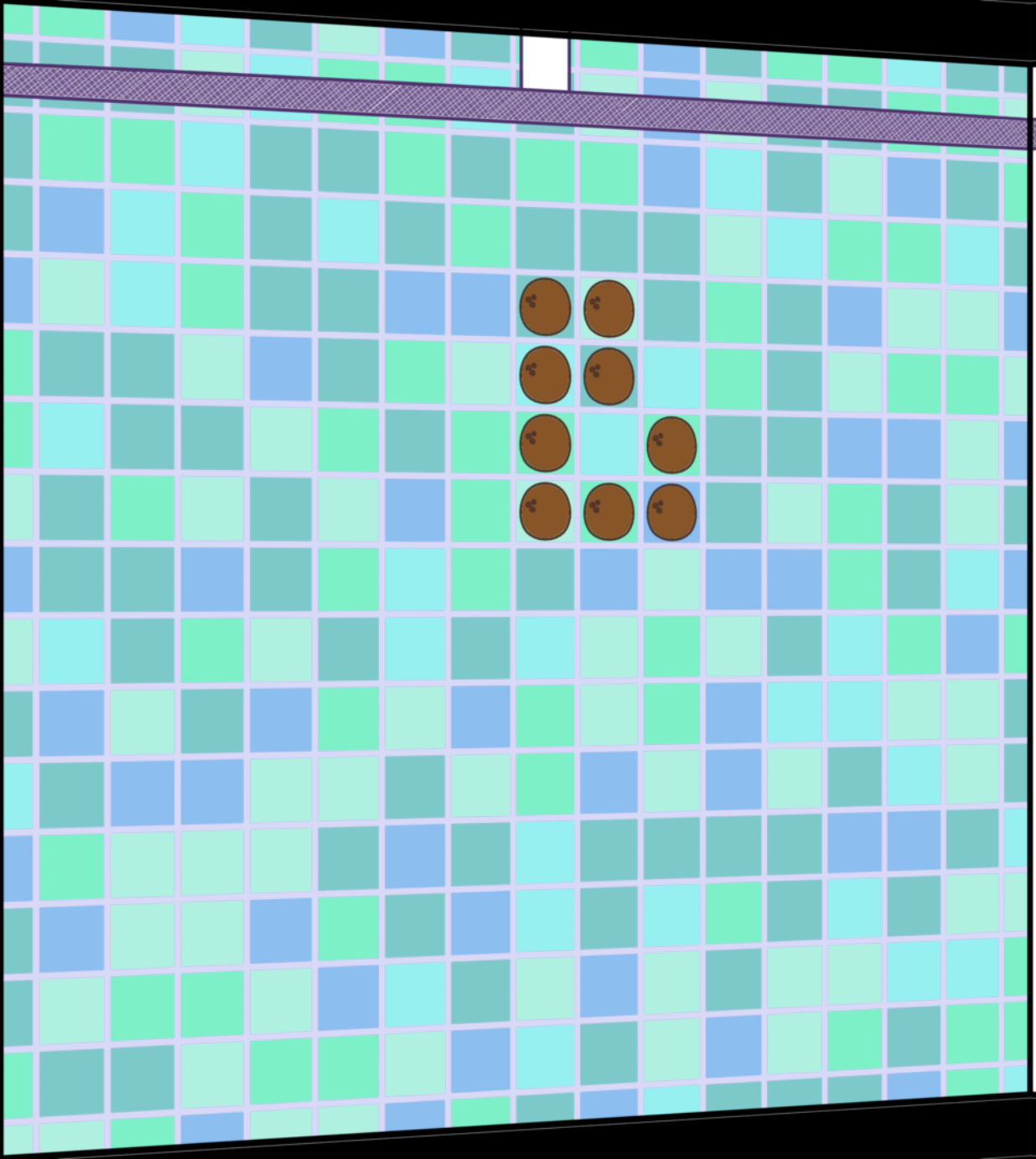
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

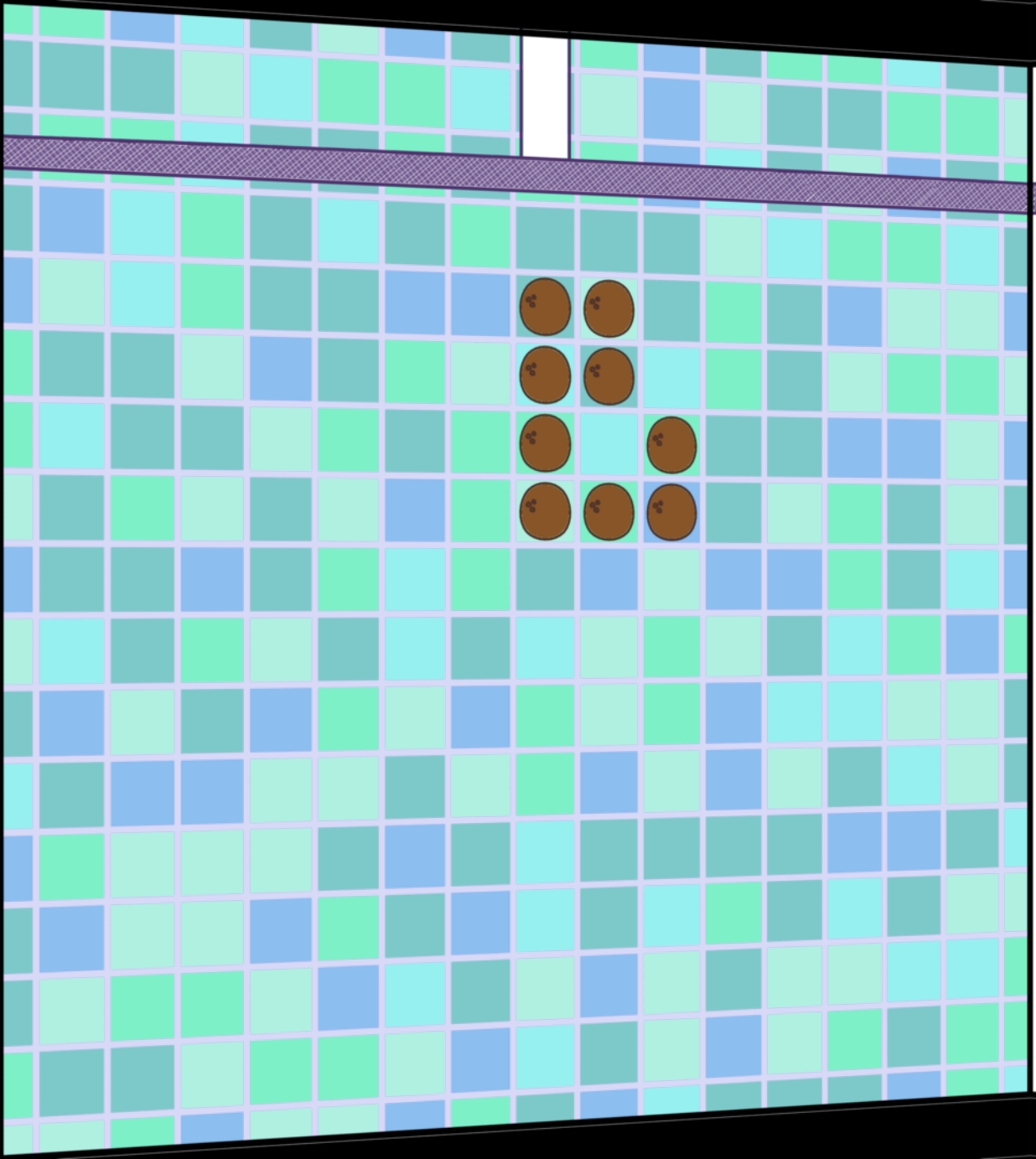
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

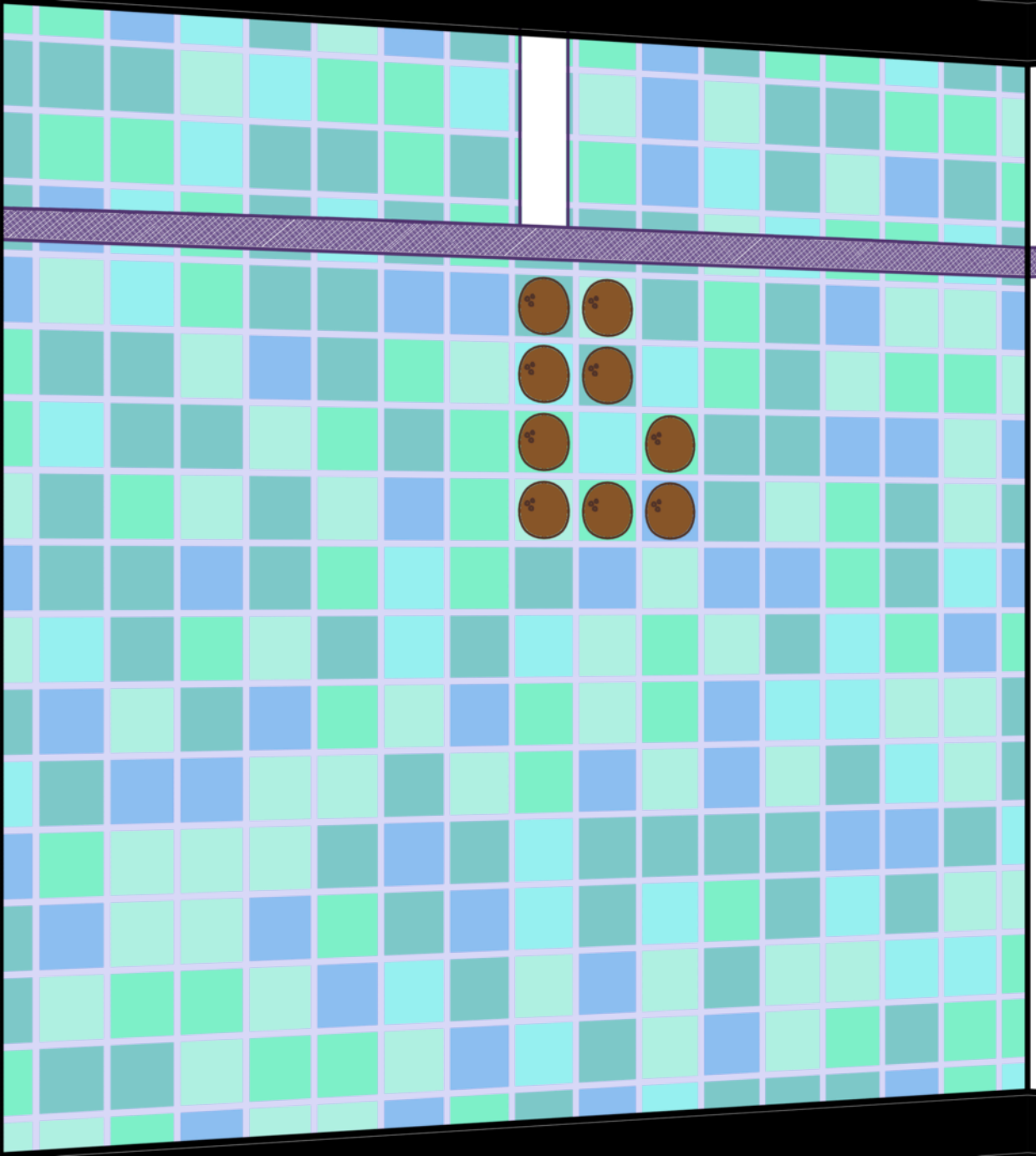
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

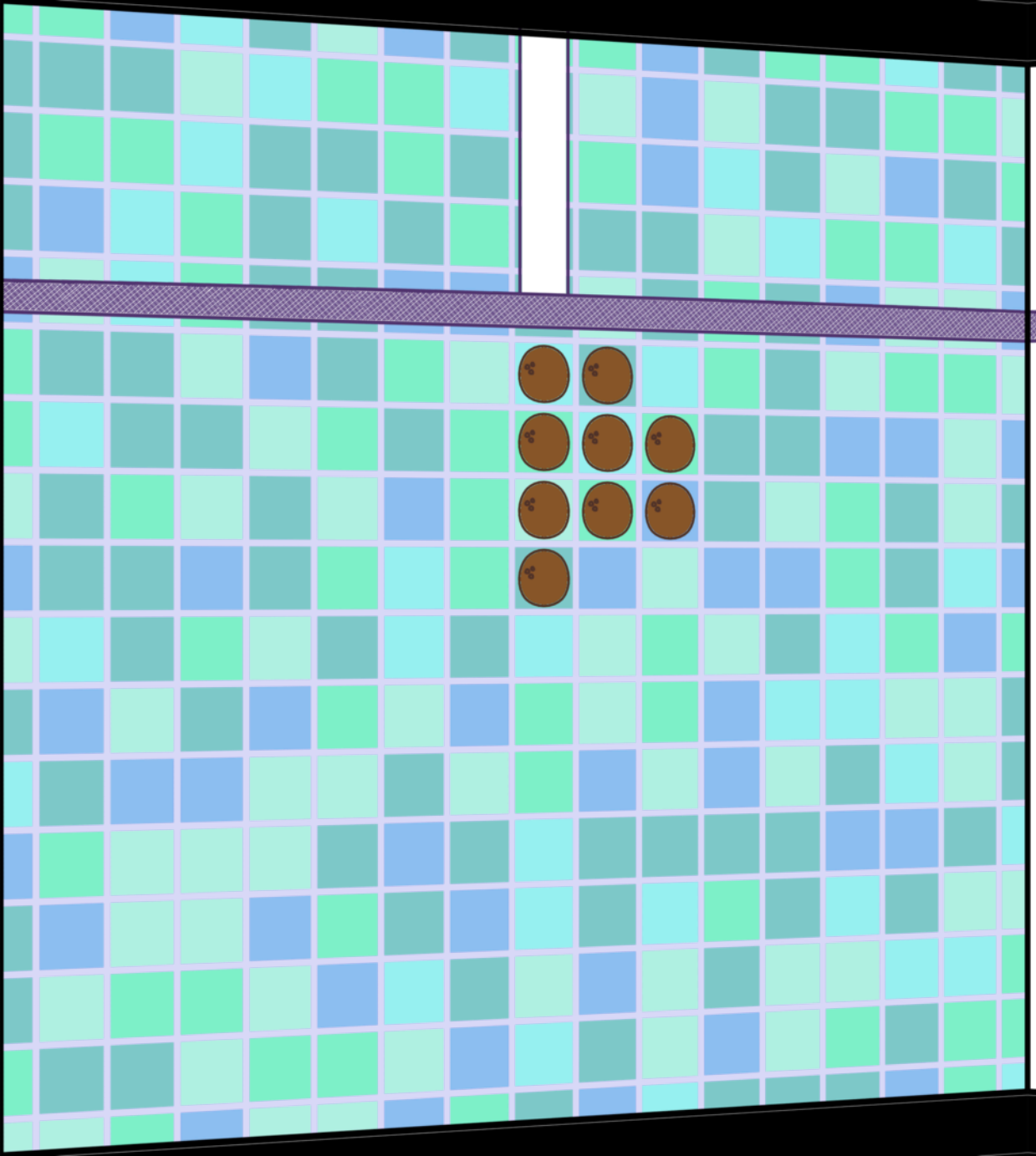
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

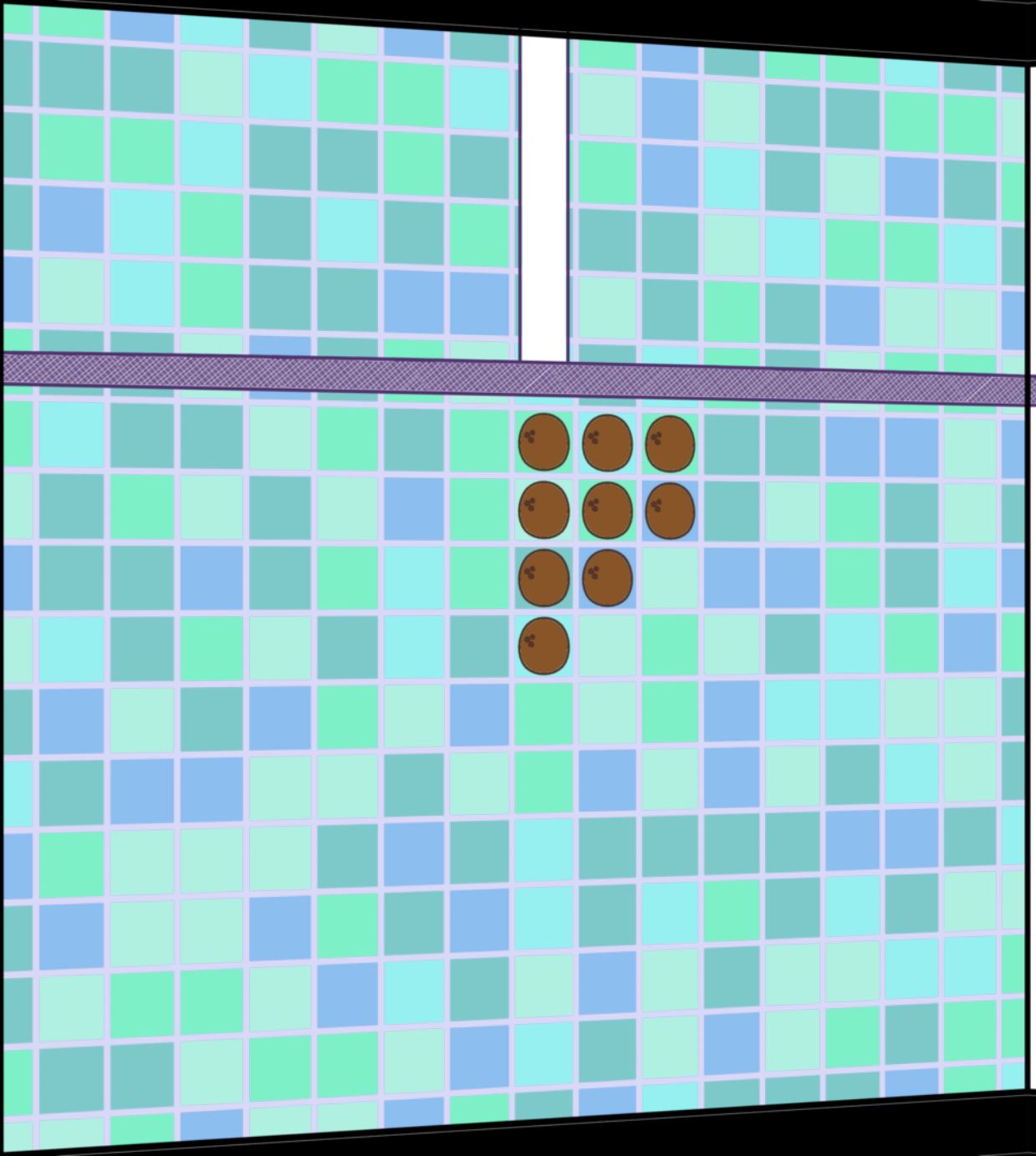
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

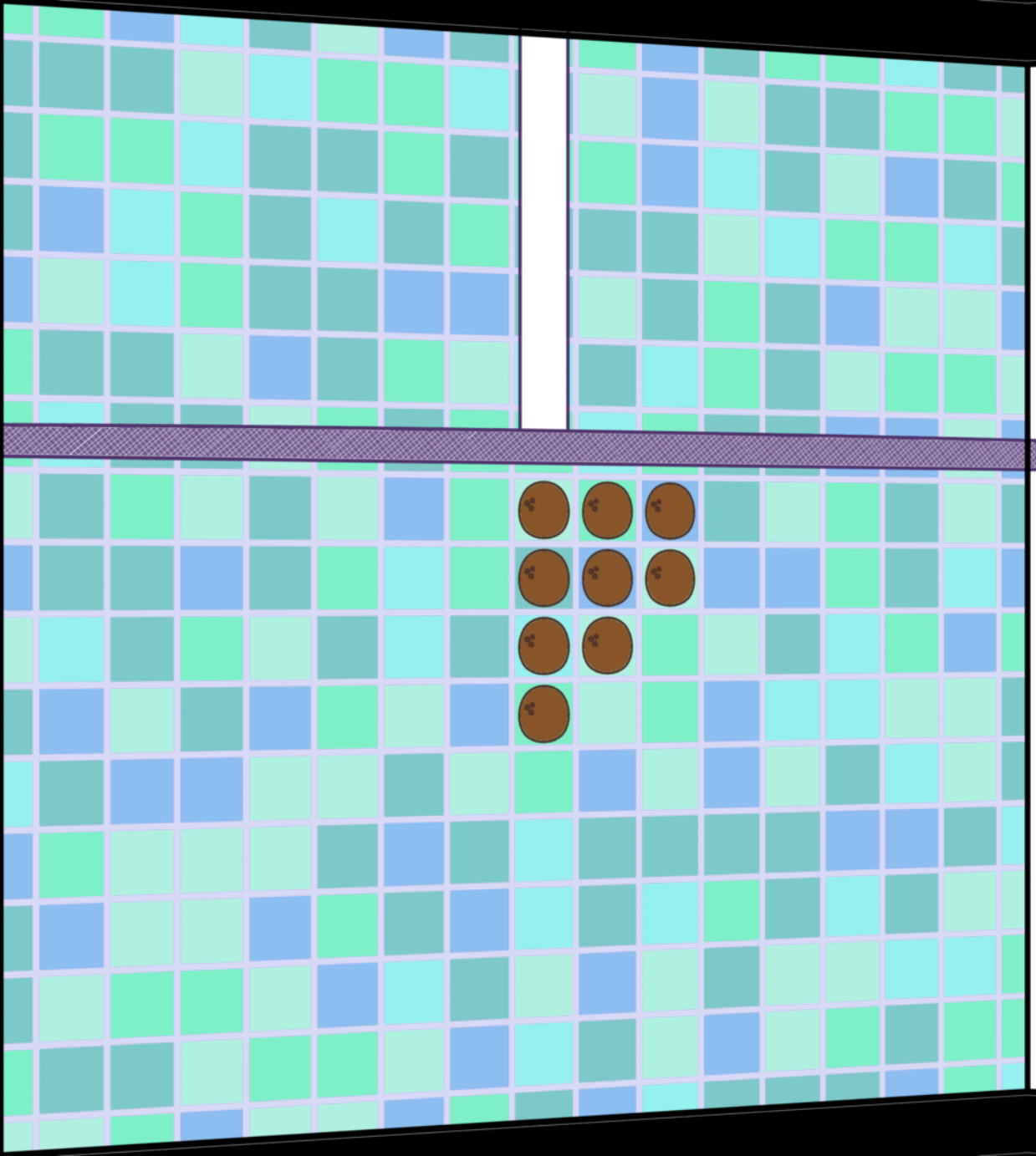
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

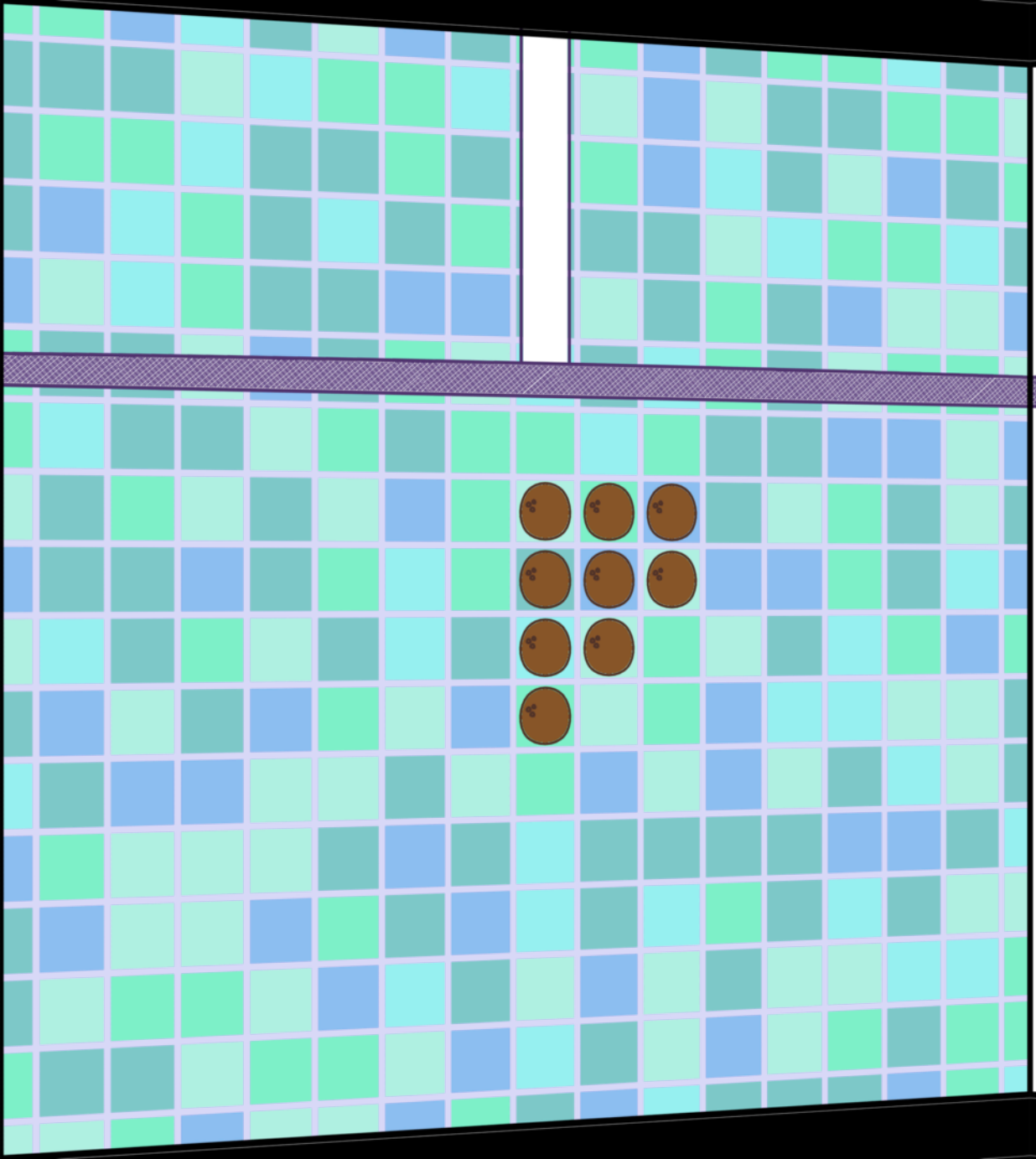
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

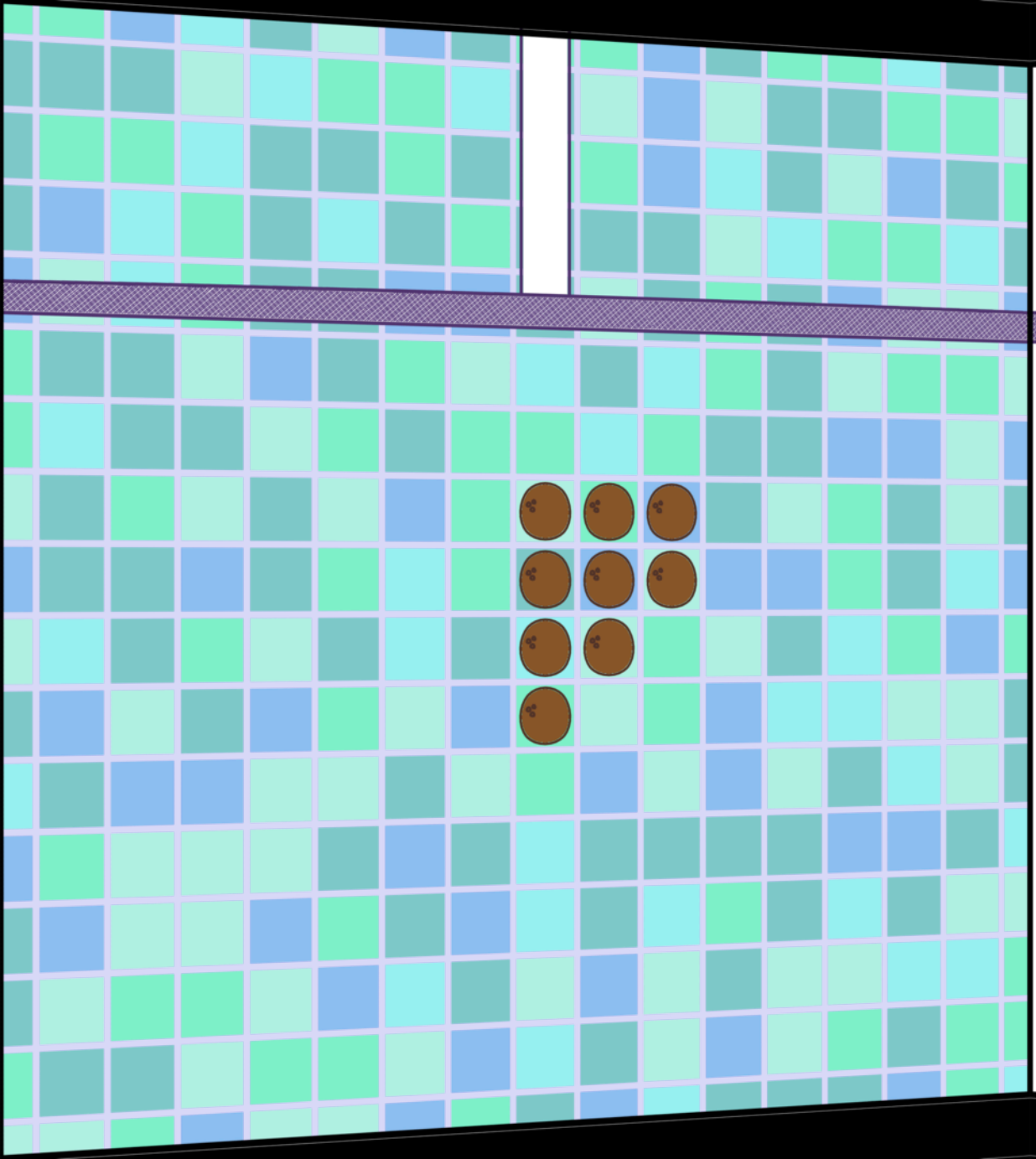
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

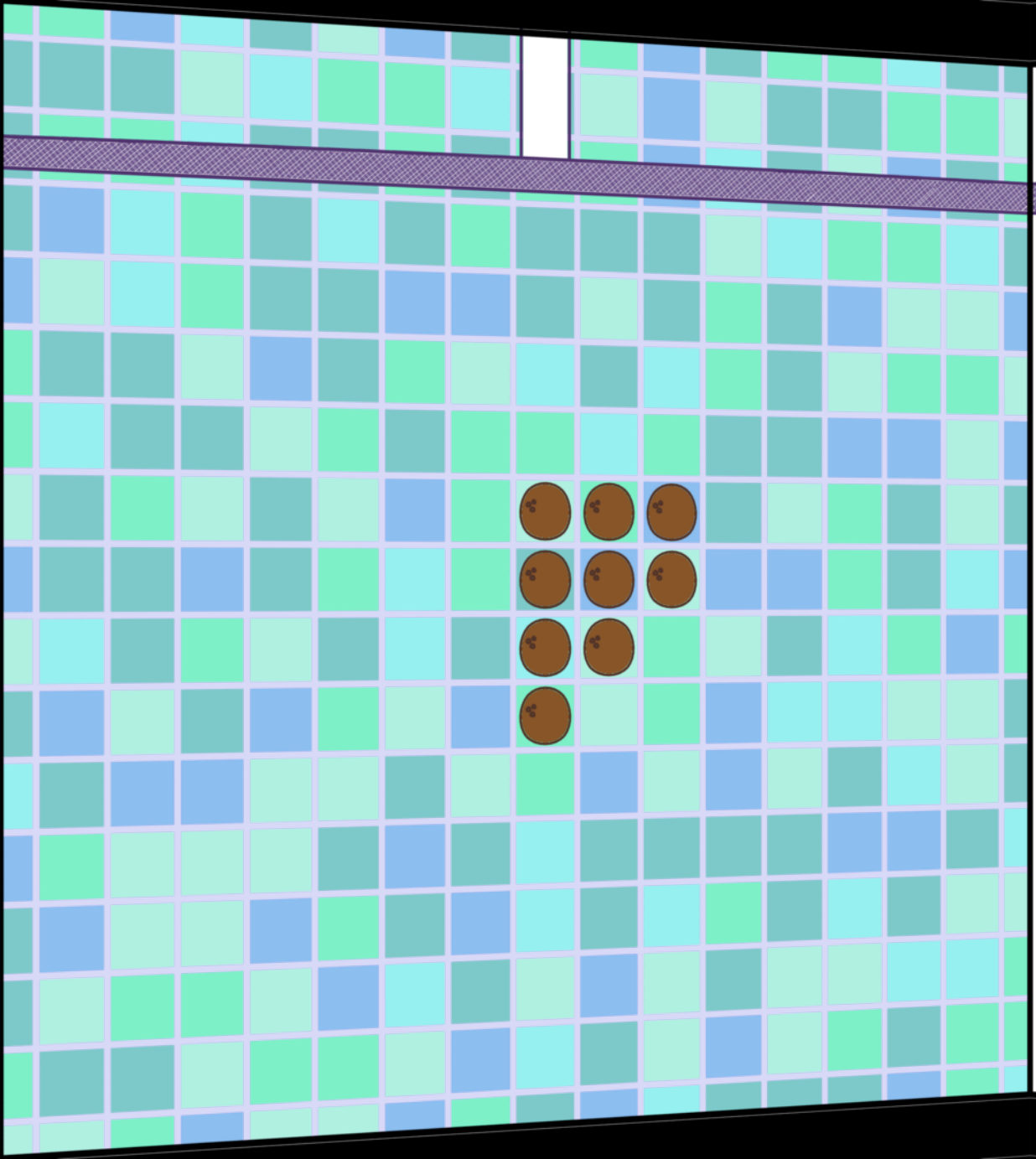
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

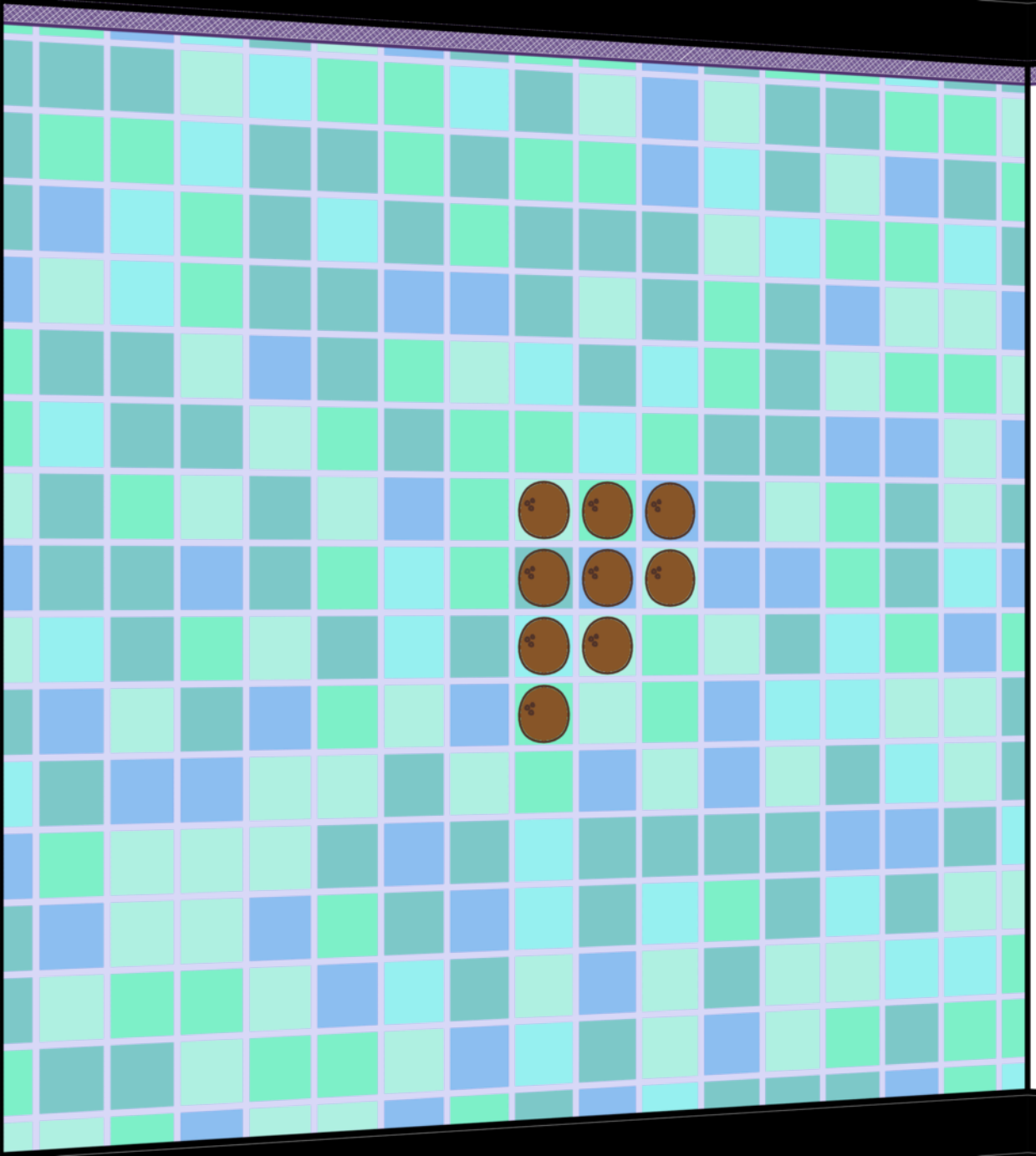
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

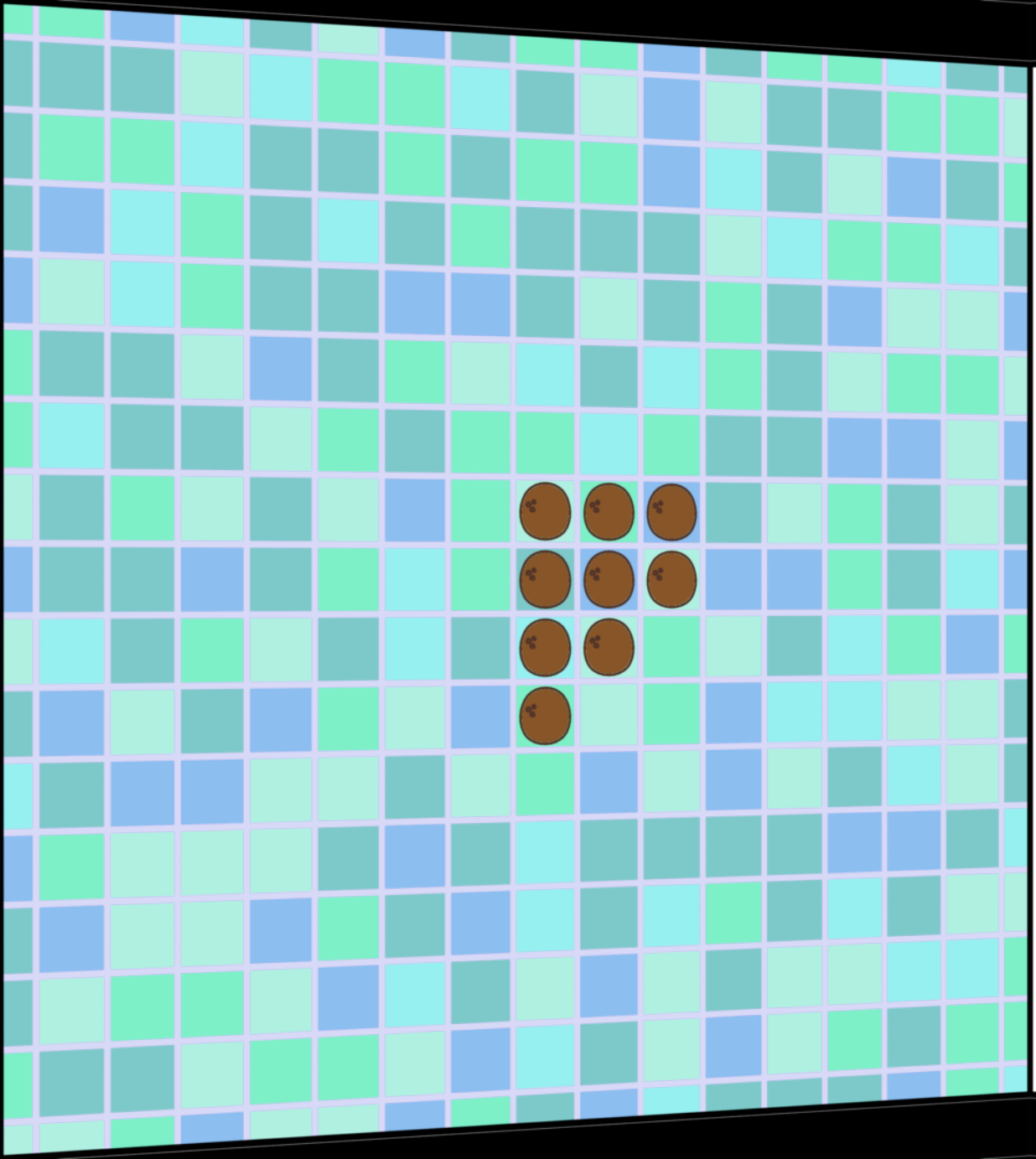
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

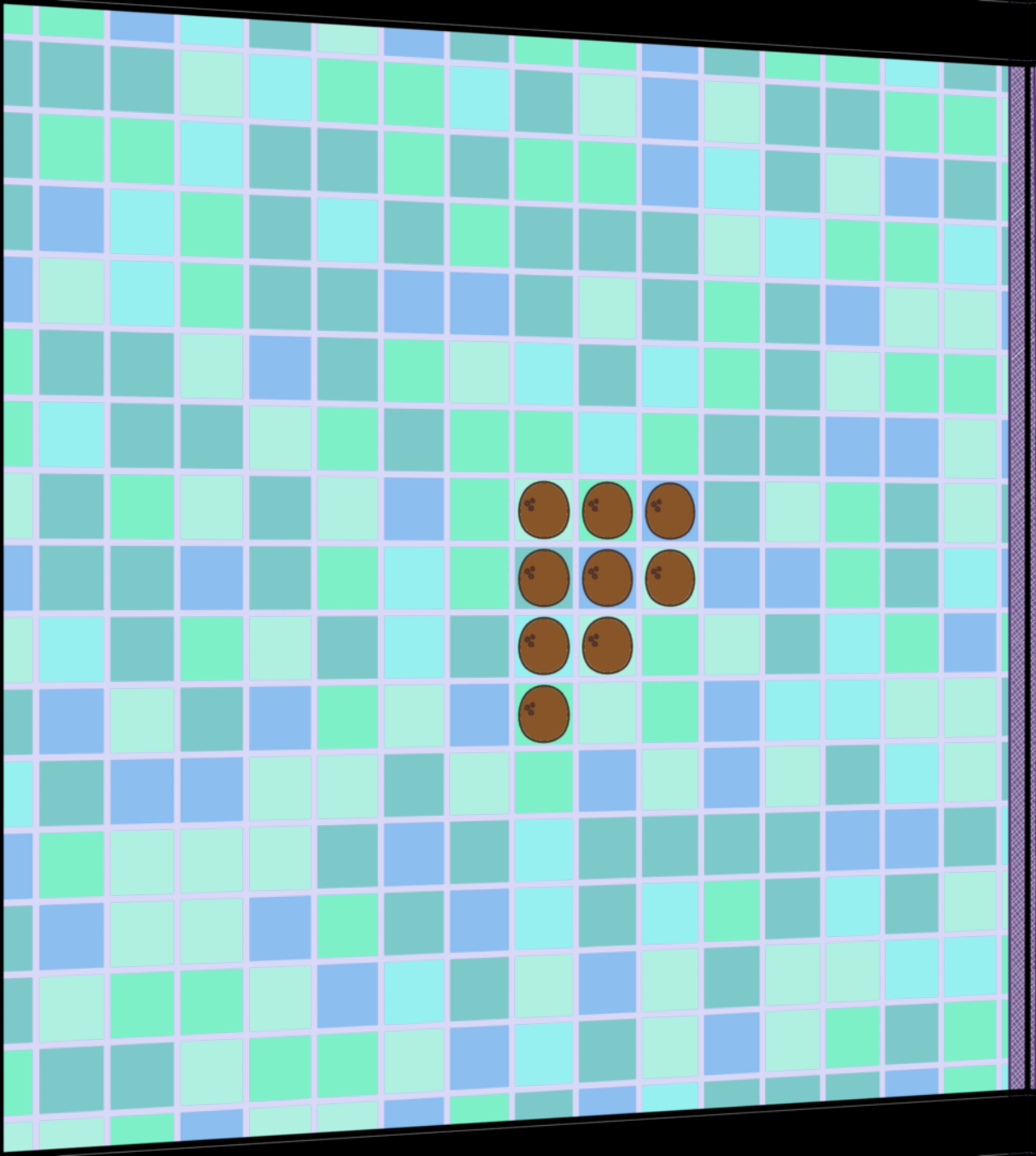
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

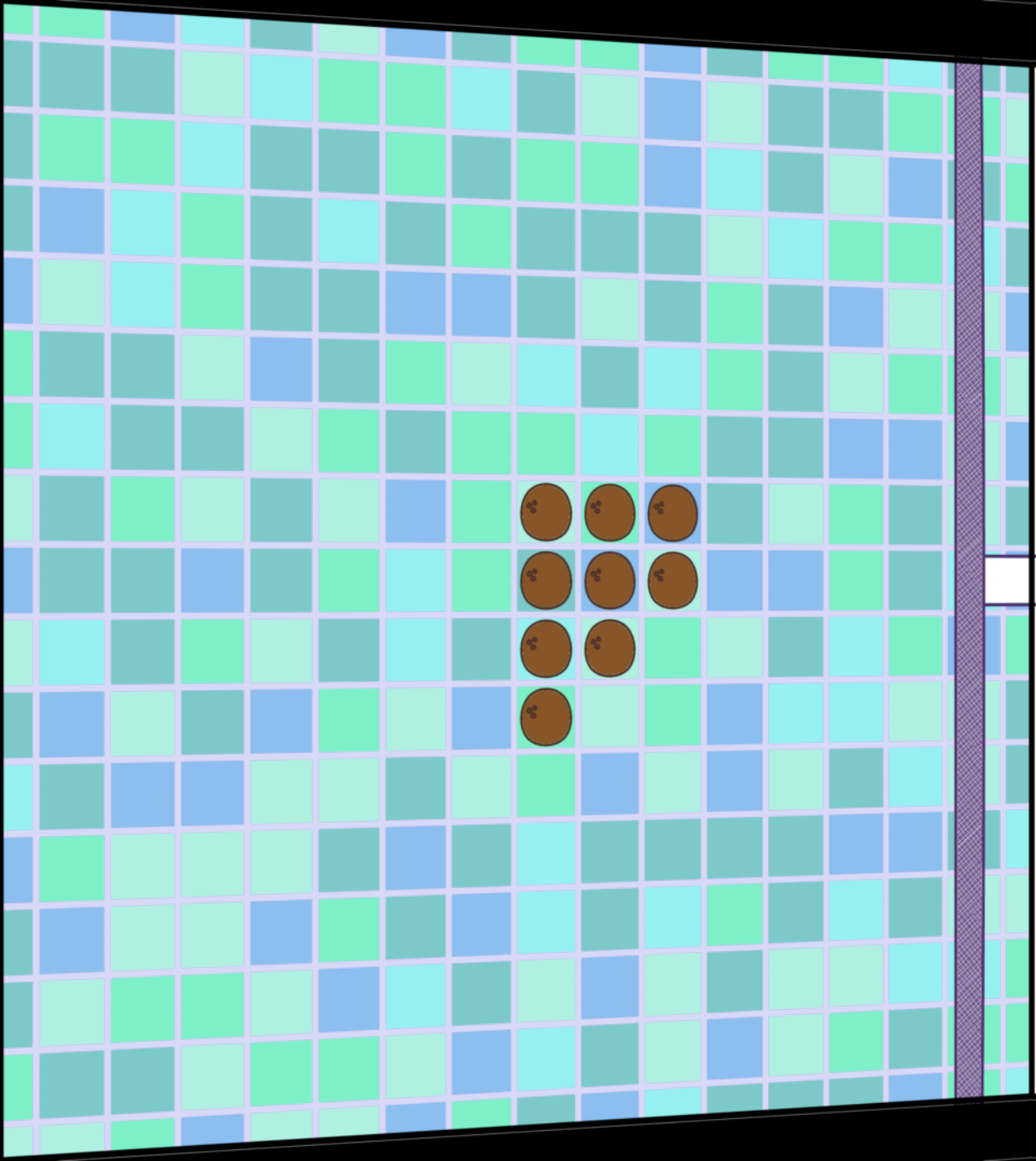
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

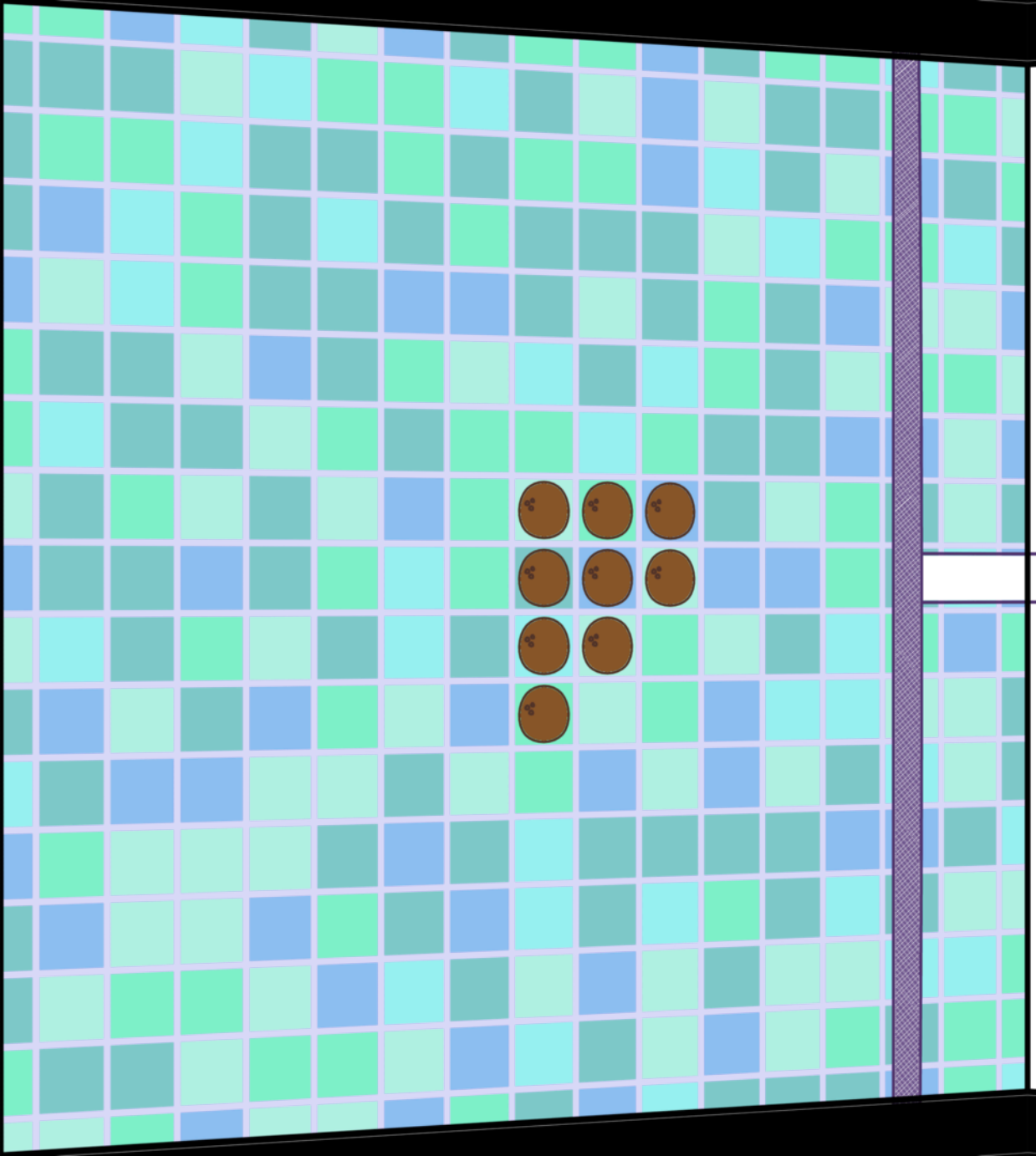
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

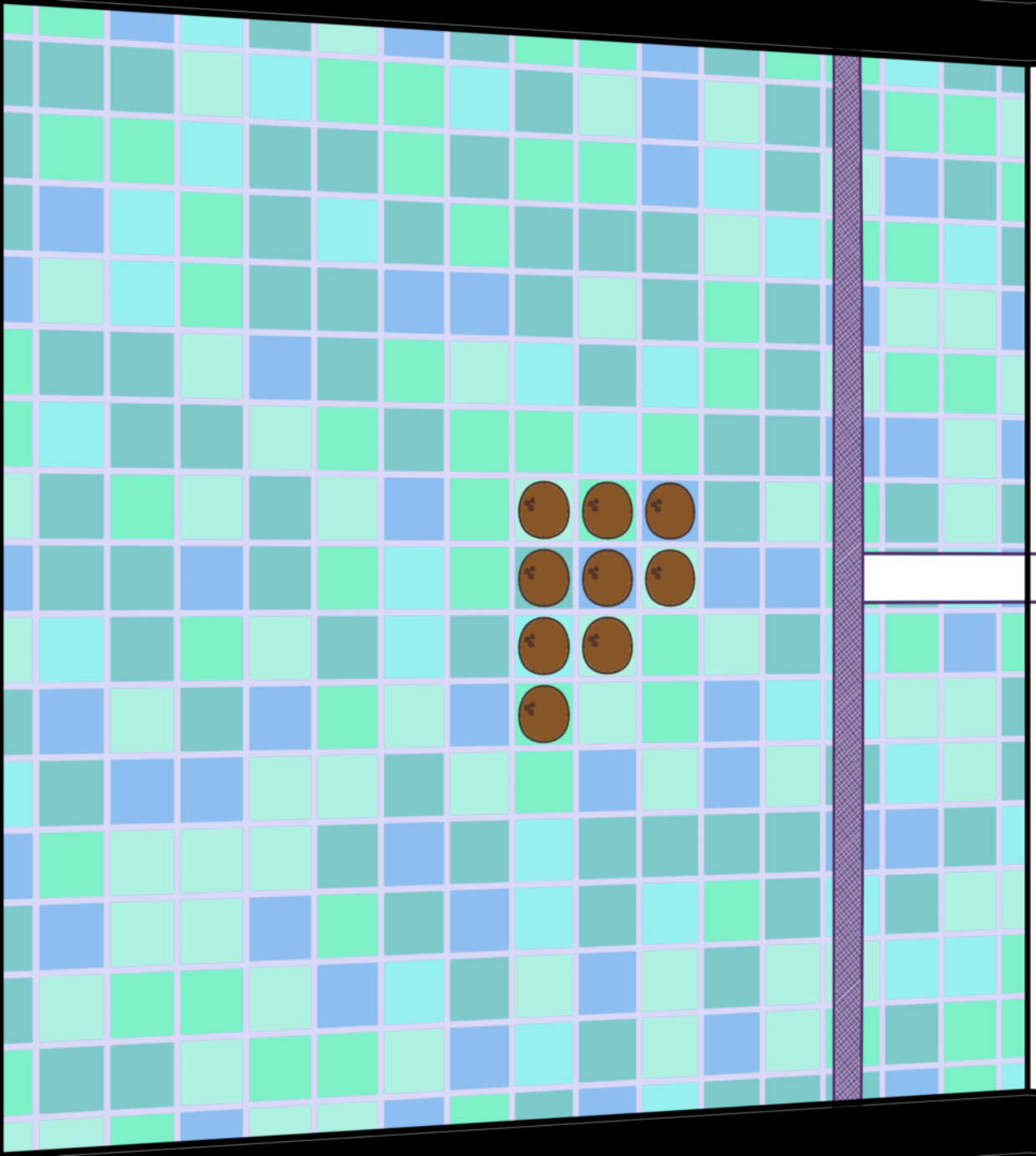
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

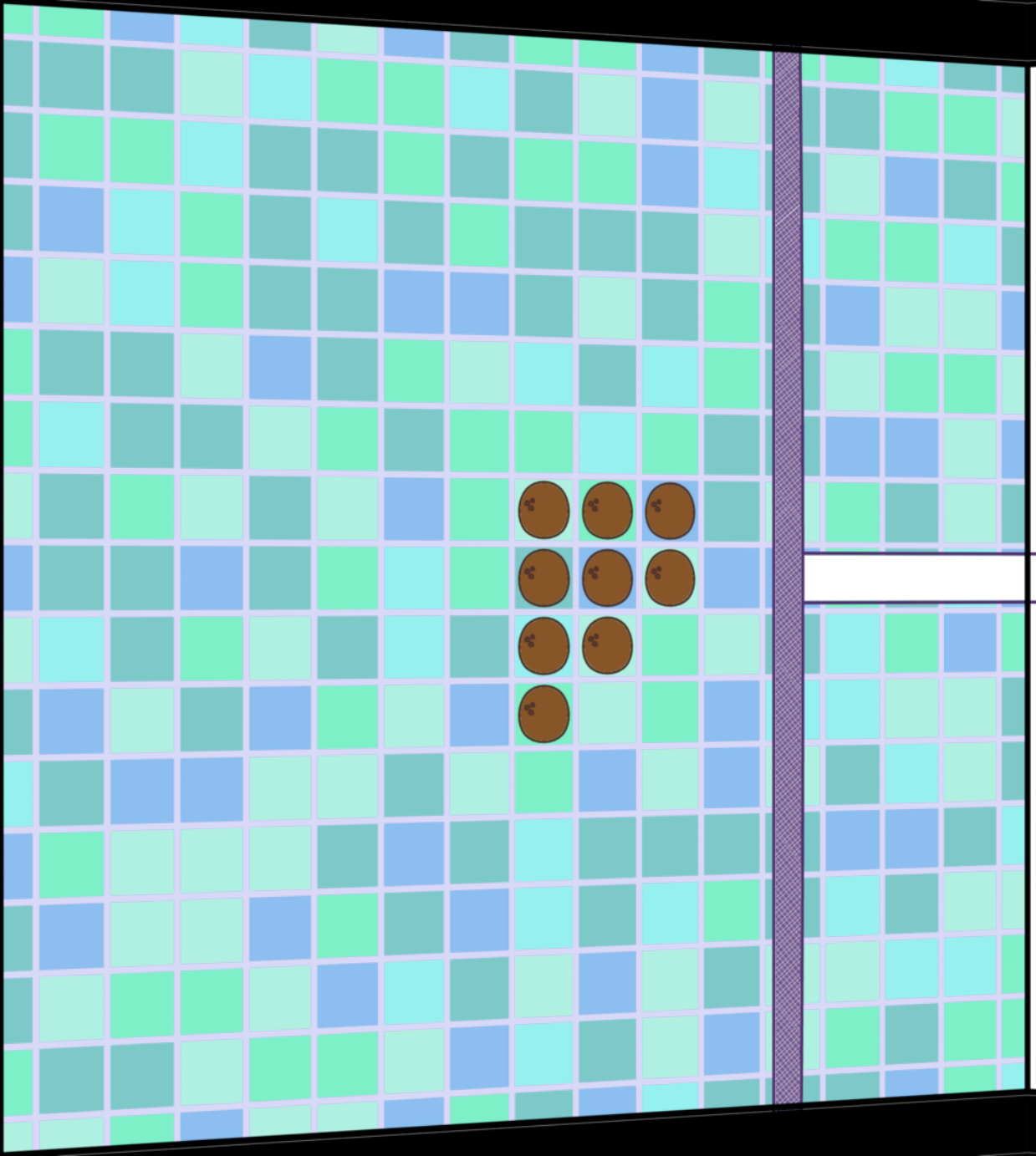
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

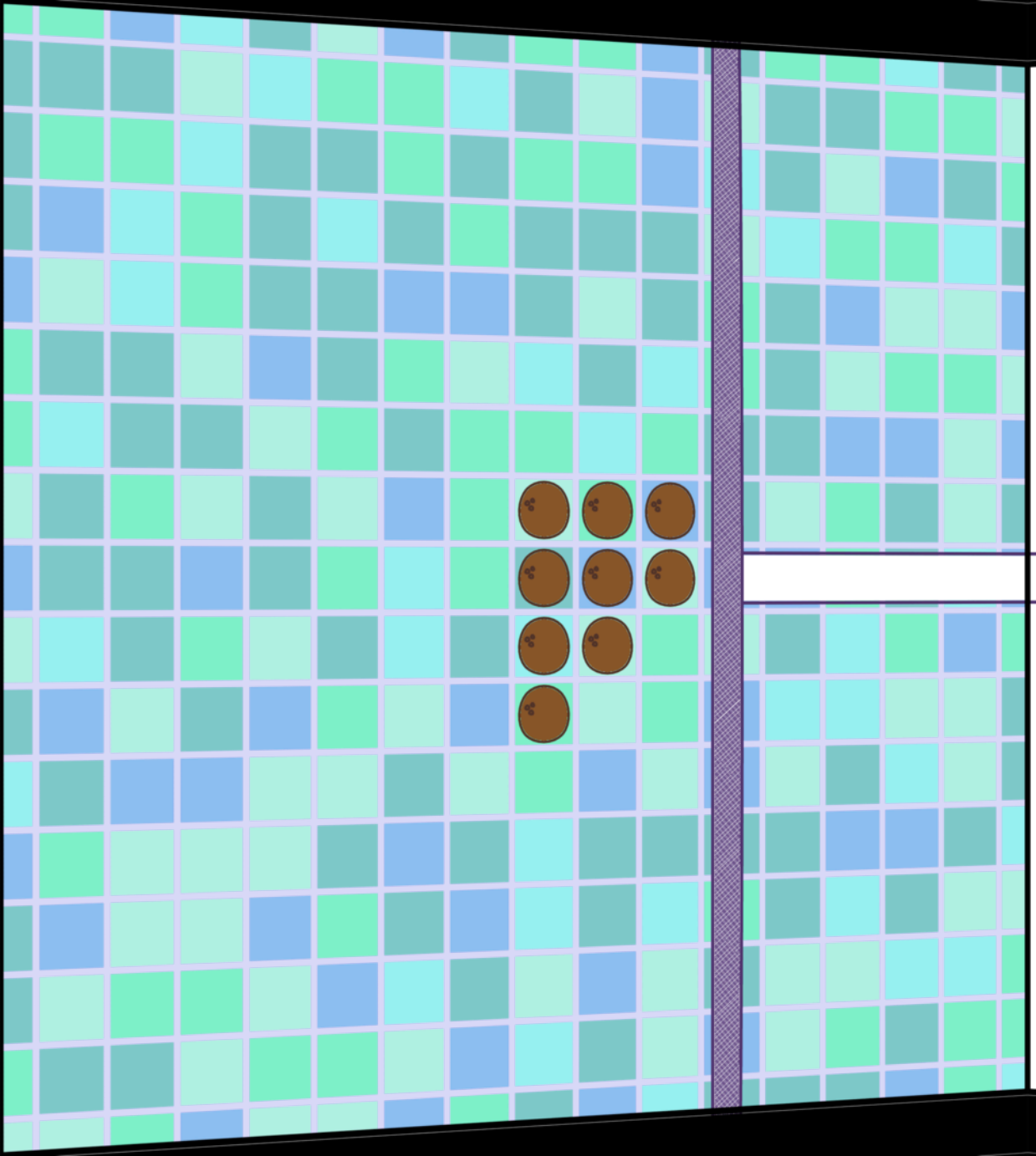
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

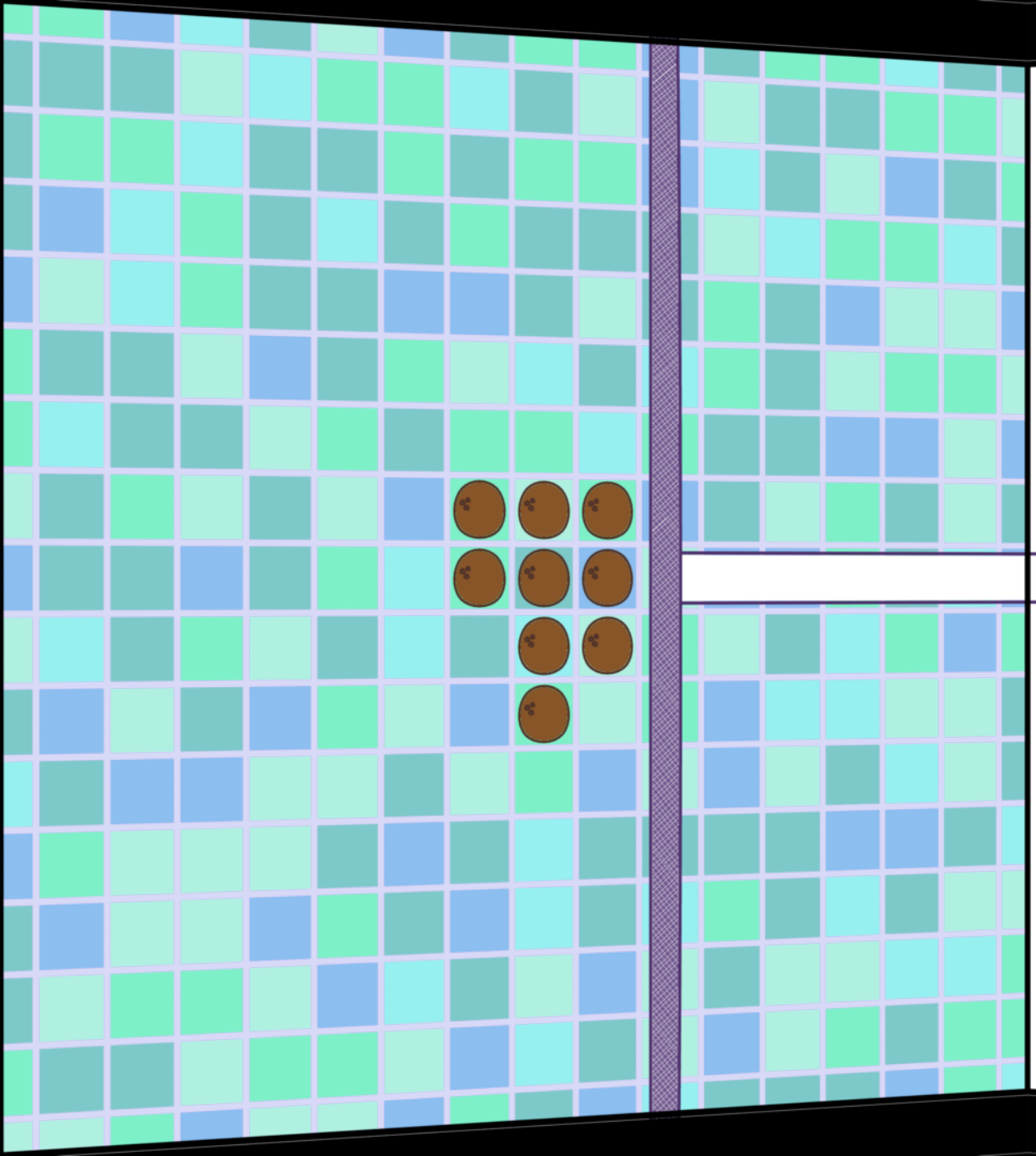
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

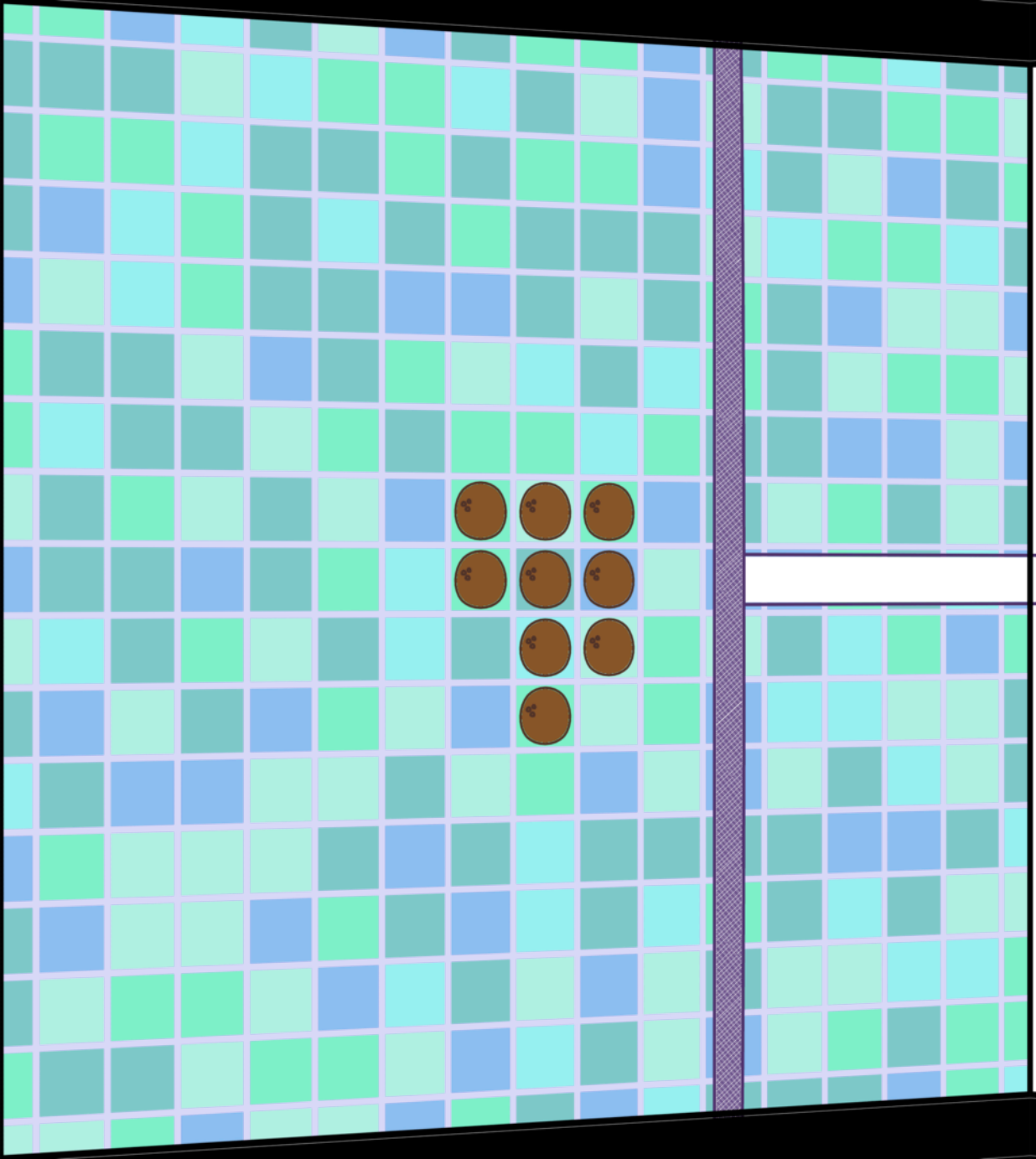
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

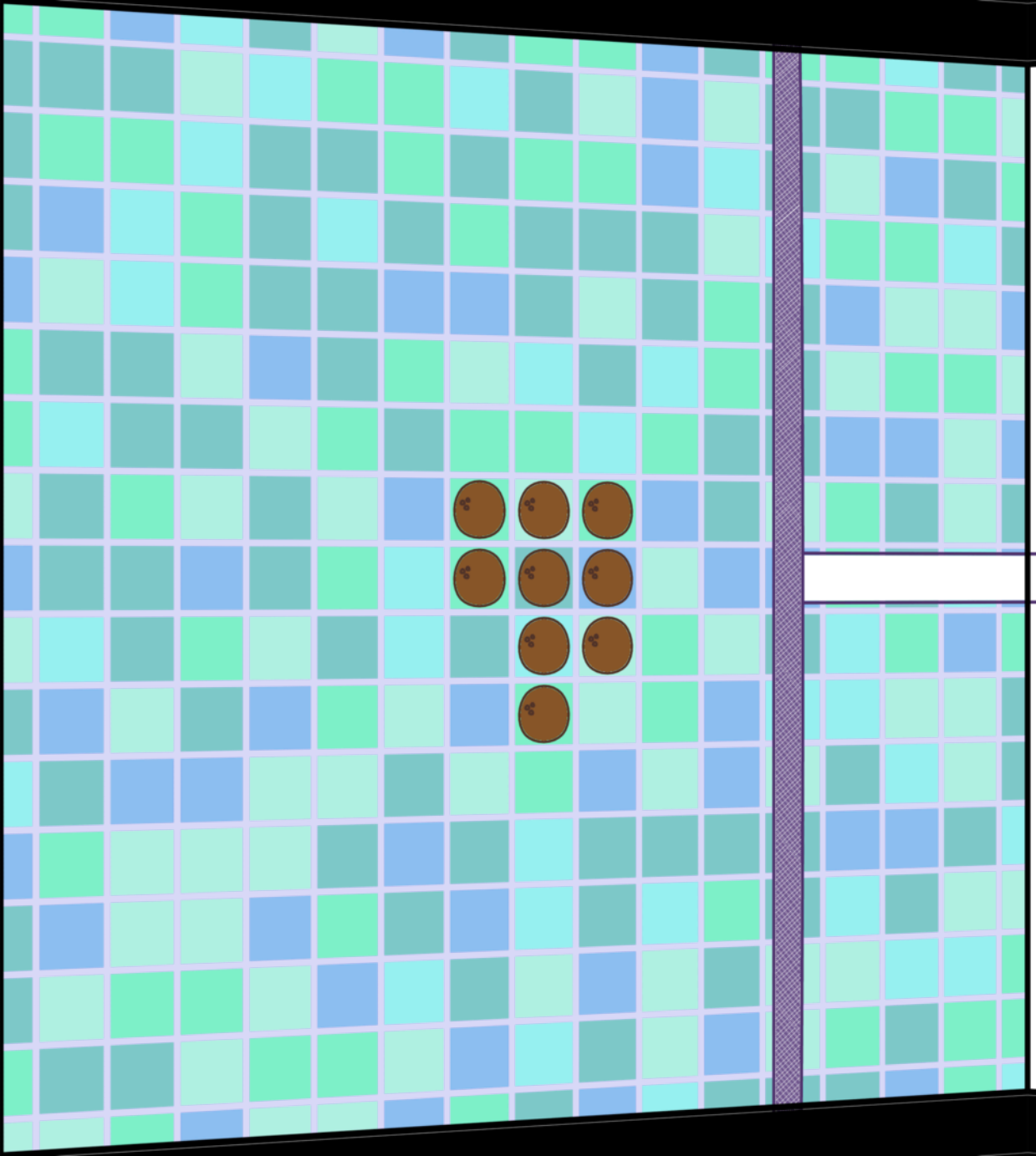
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

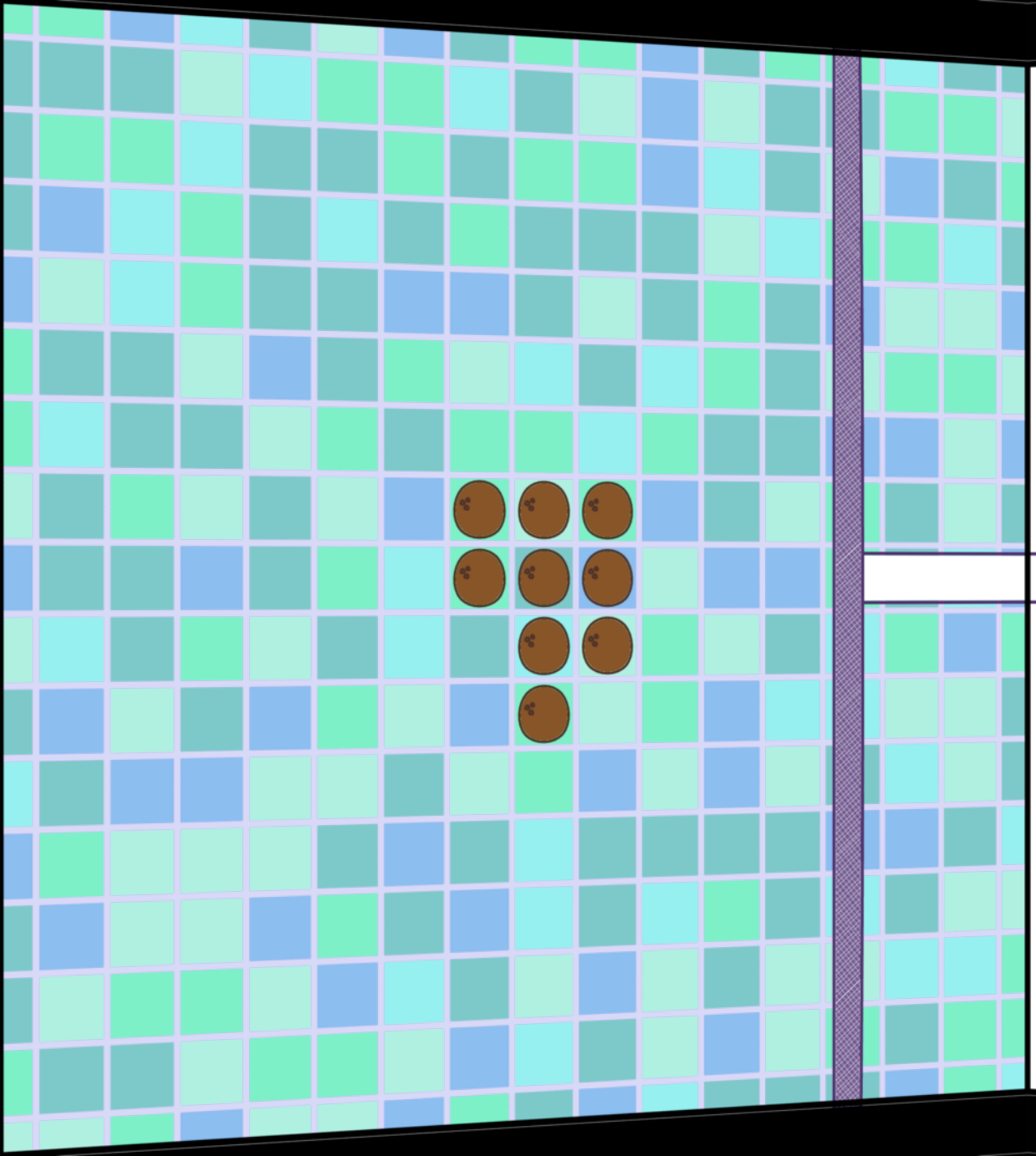
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

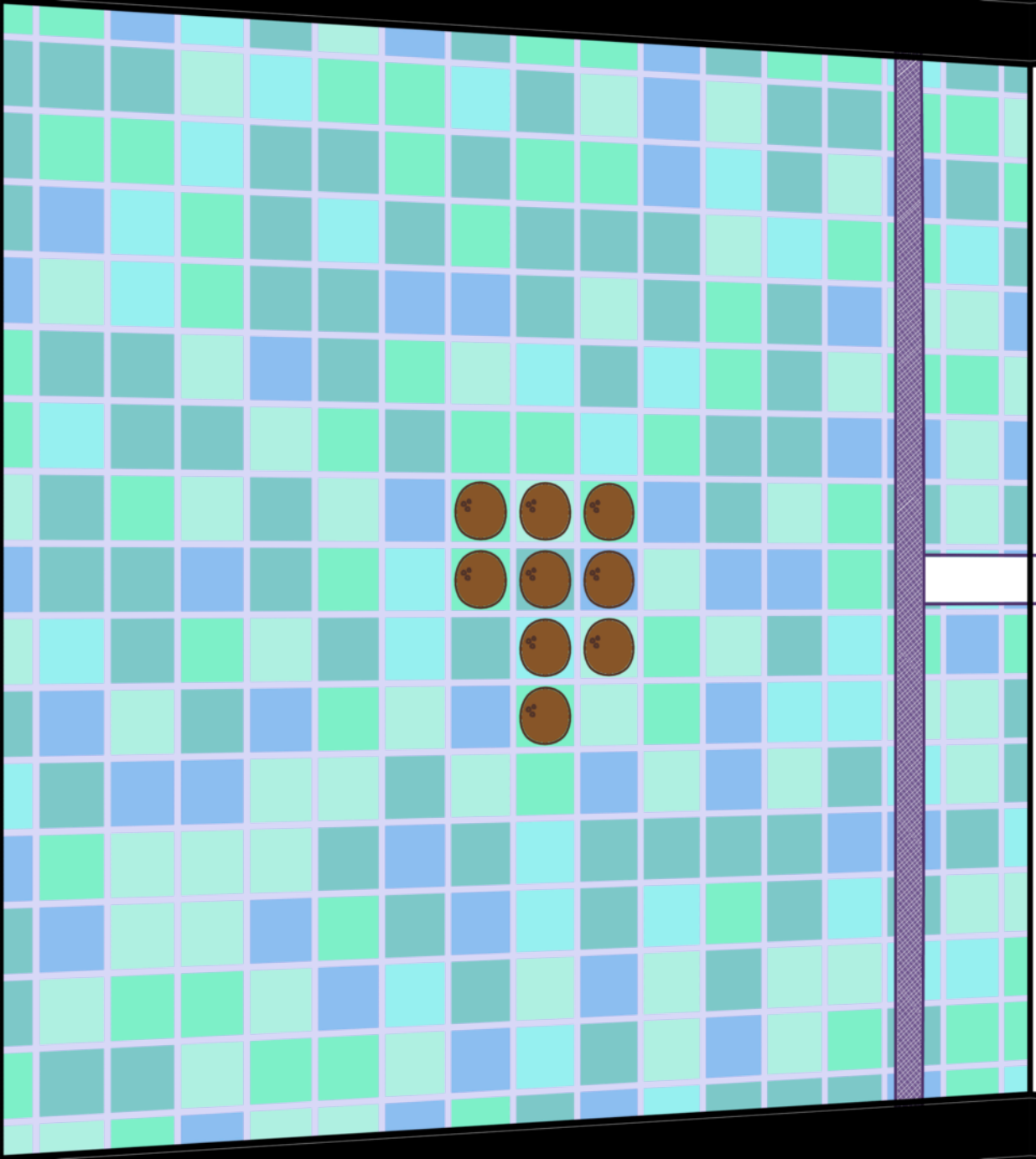
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

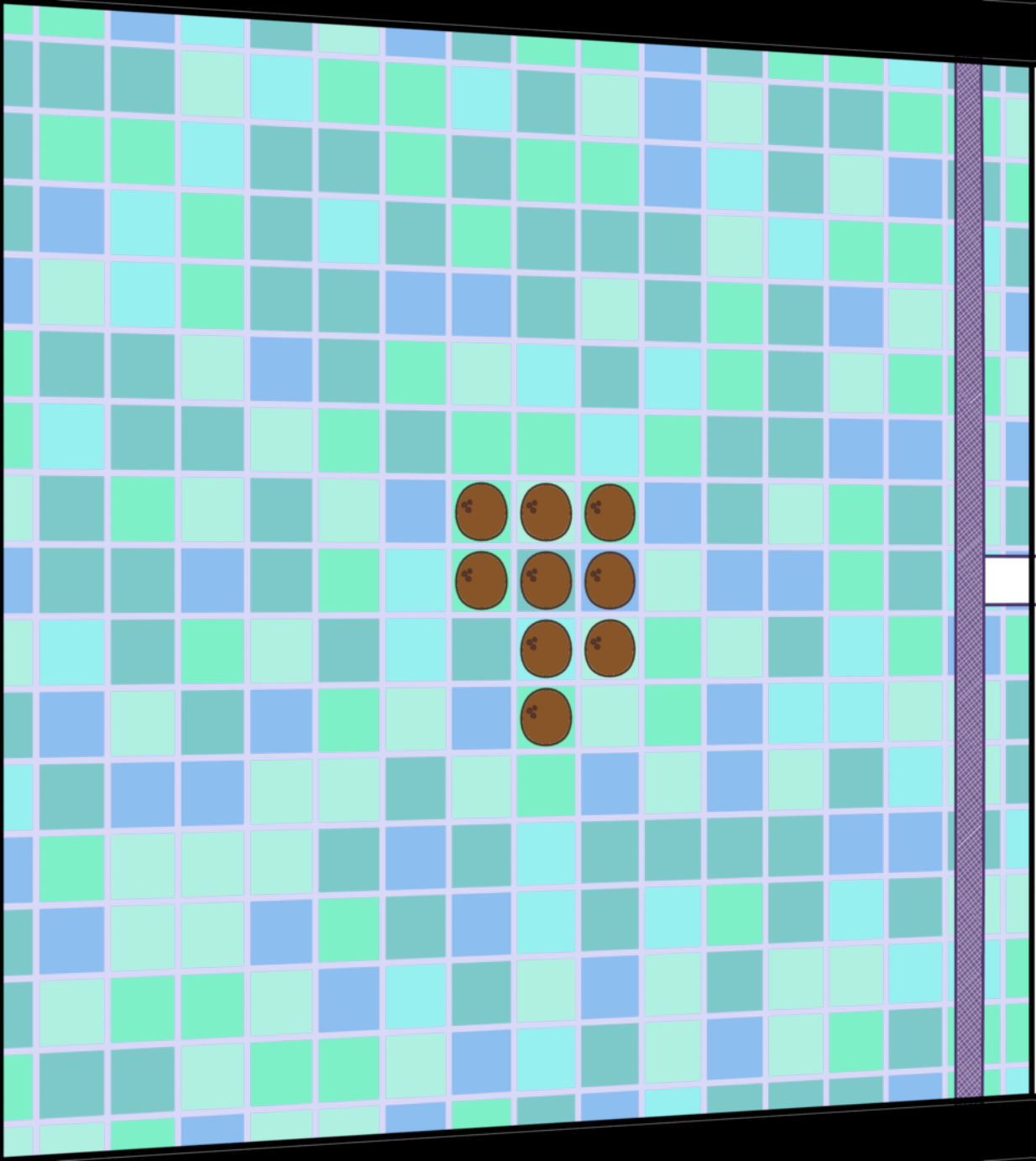
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

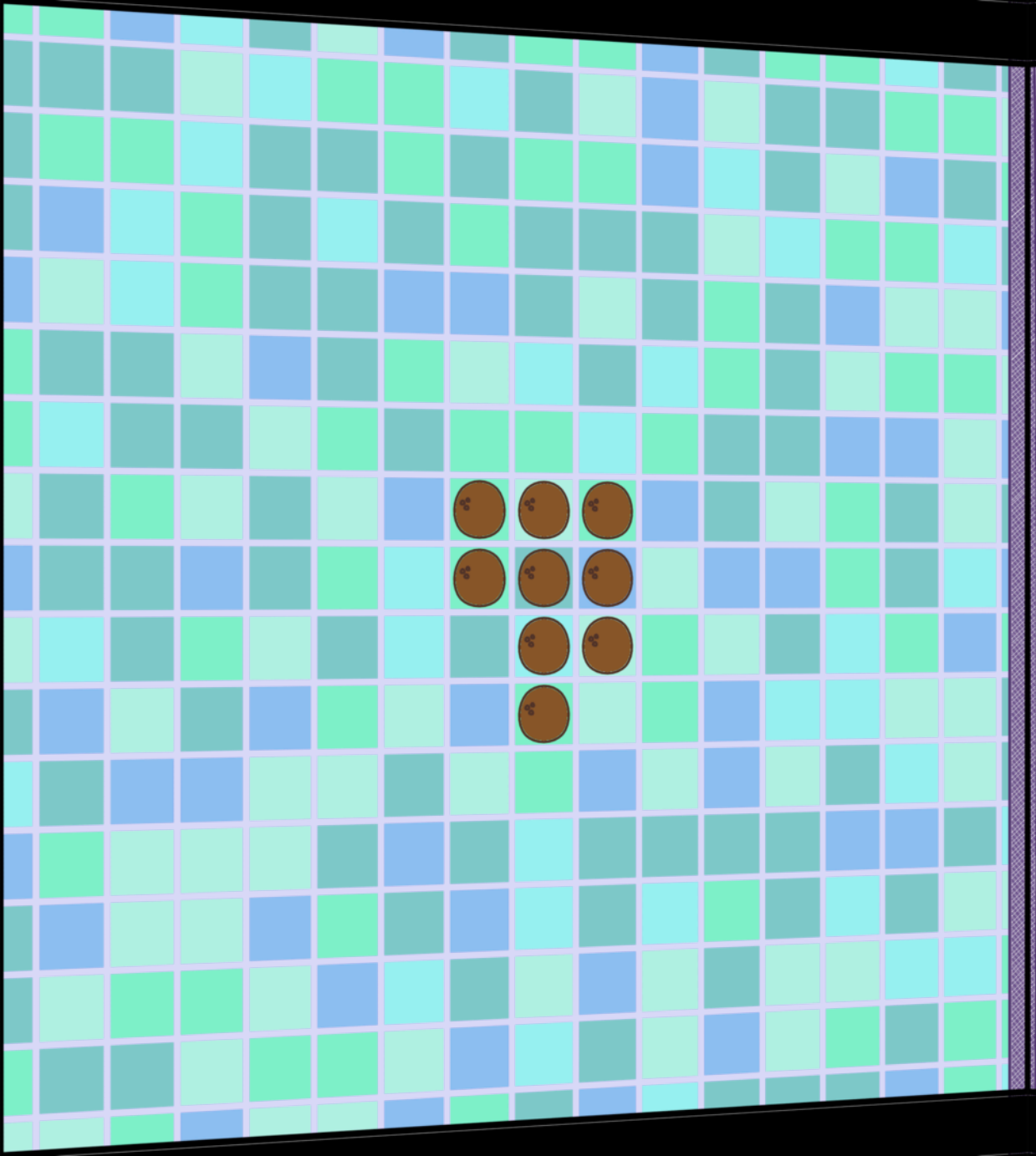
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

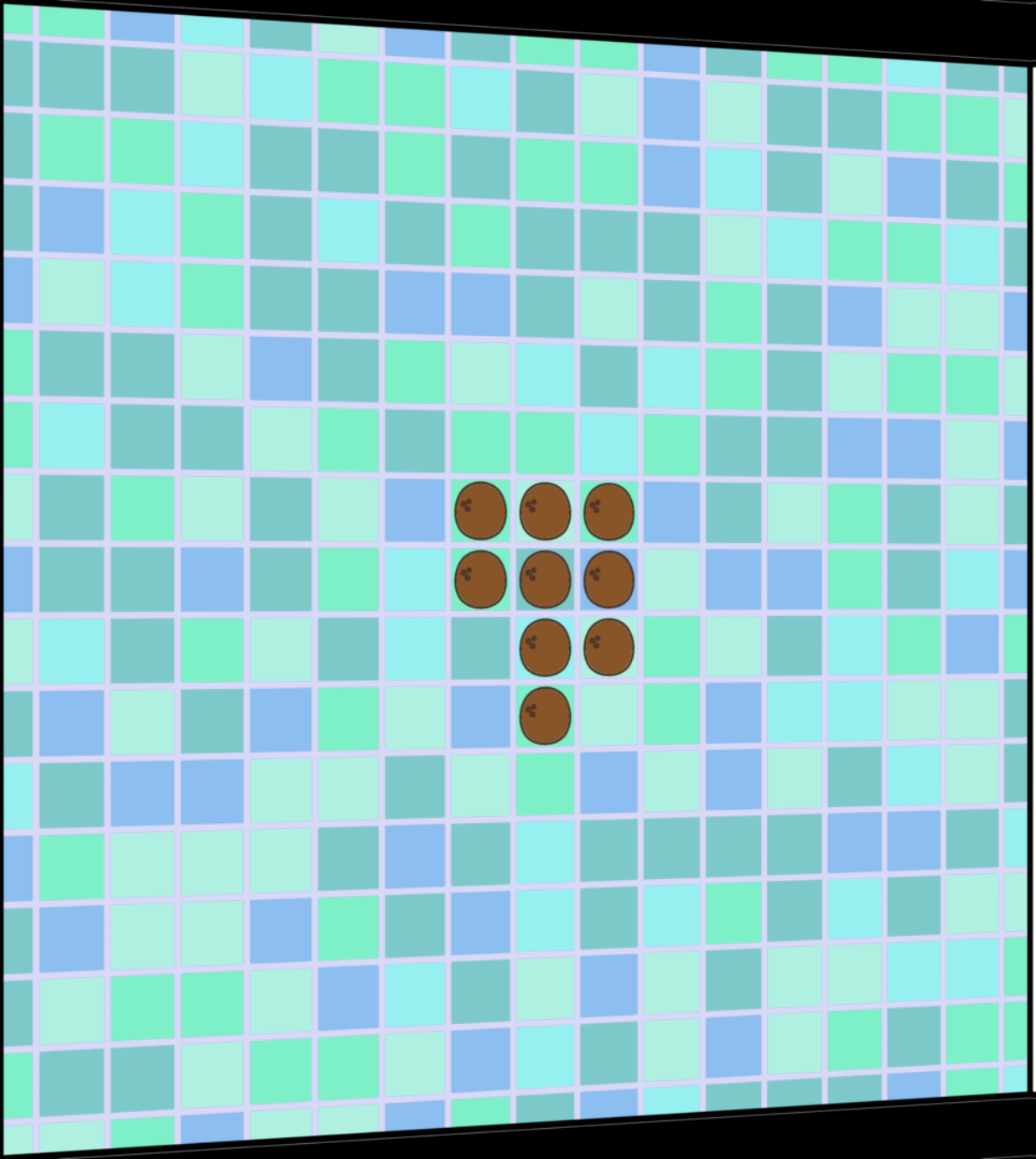
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

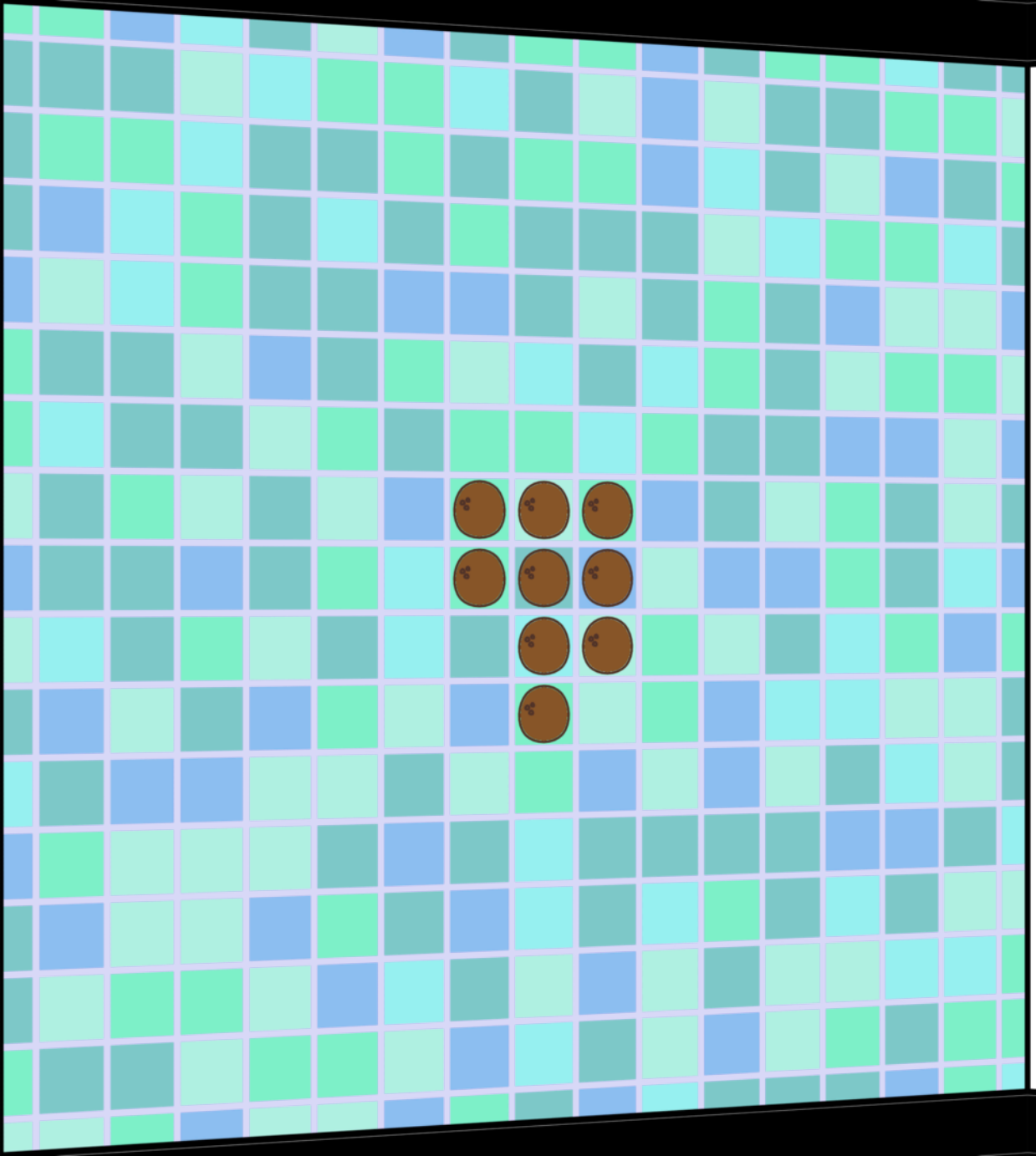
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.



Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.

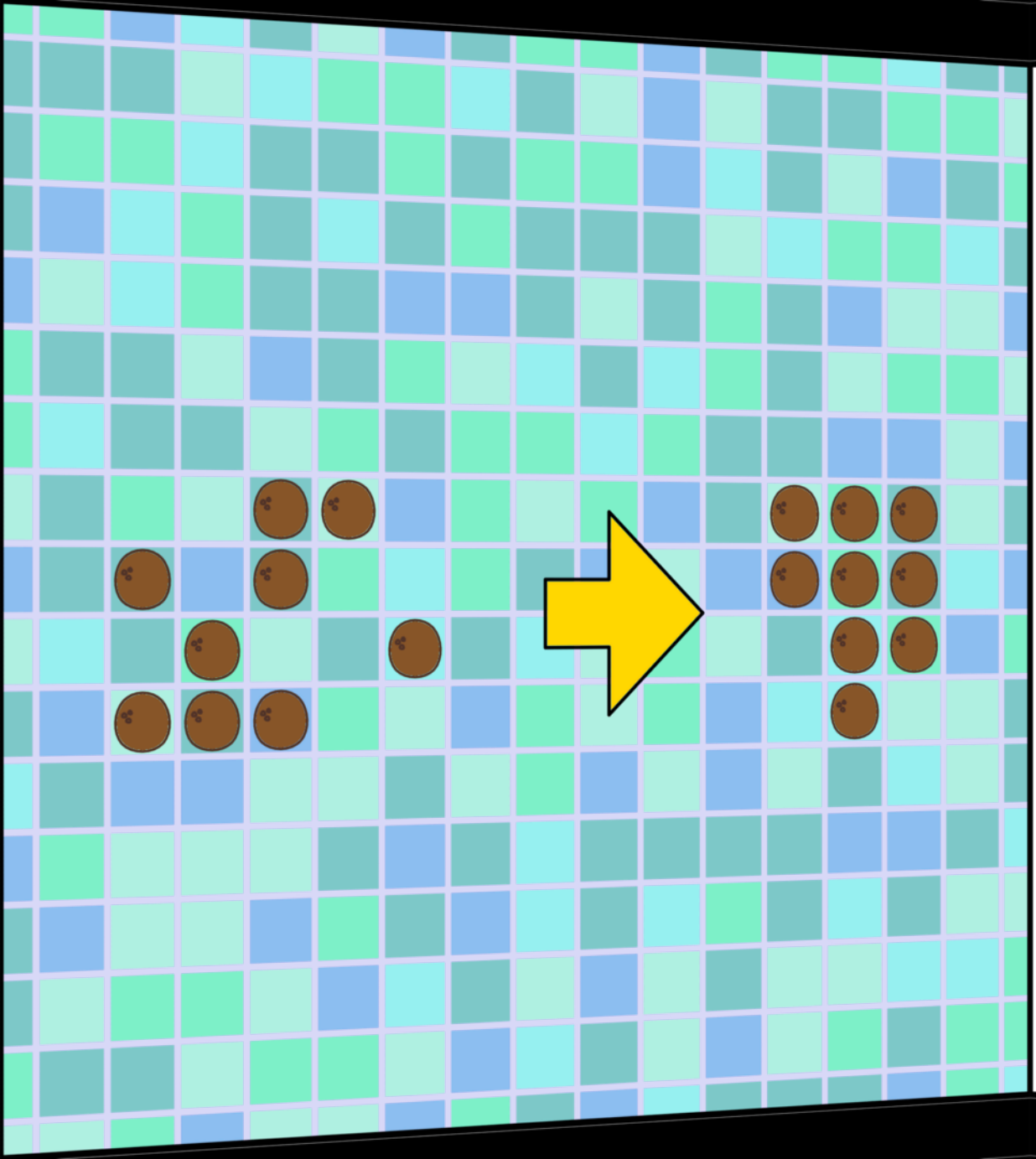


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.

Any sequence of such coconut pushes leads to a *reconfiguration* of the coconuts.

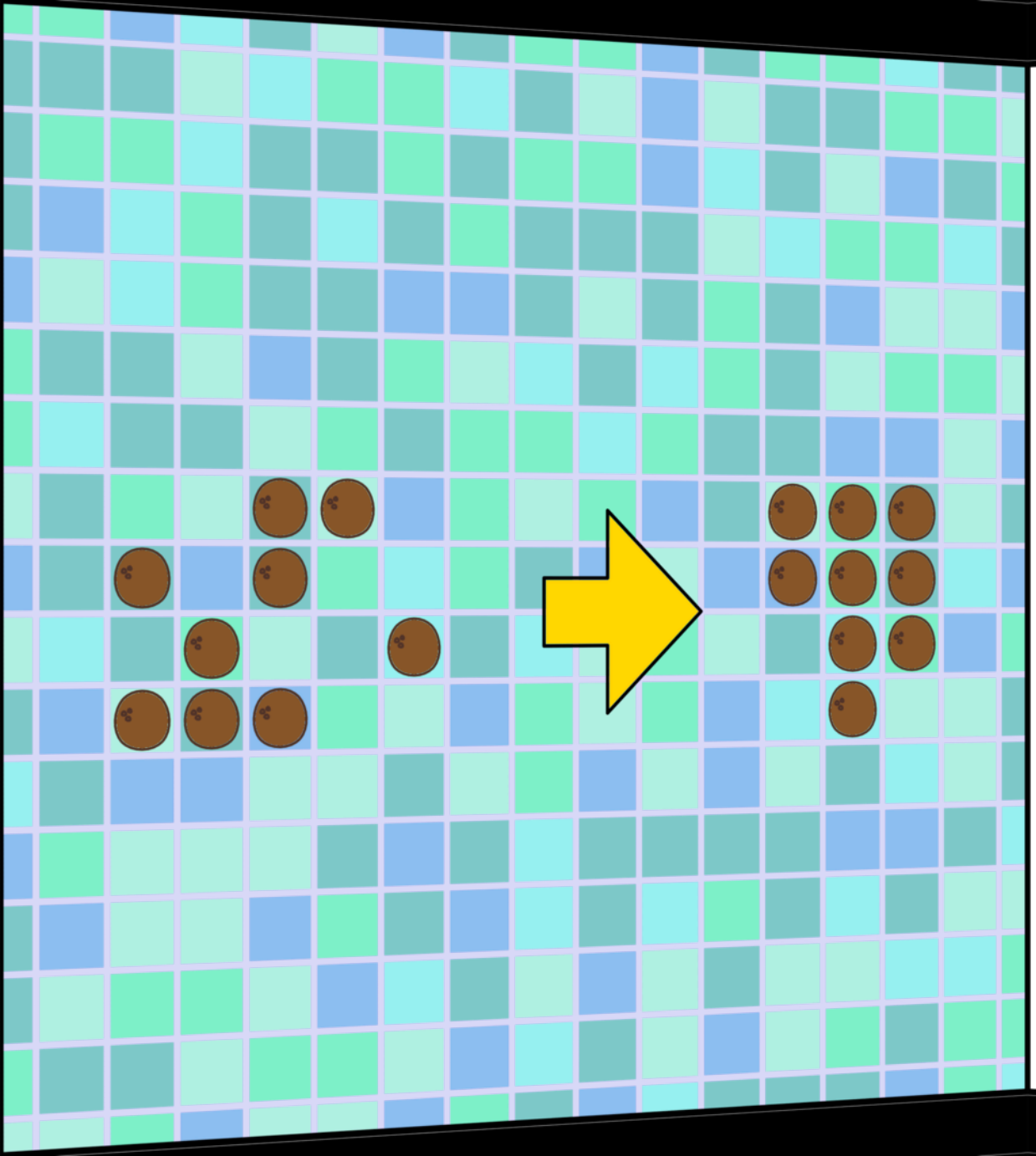


Let P be a set of n coconuts in a pool.

We assume the coconuts are aligned to a grid.

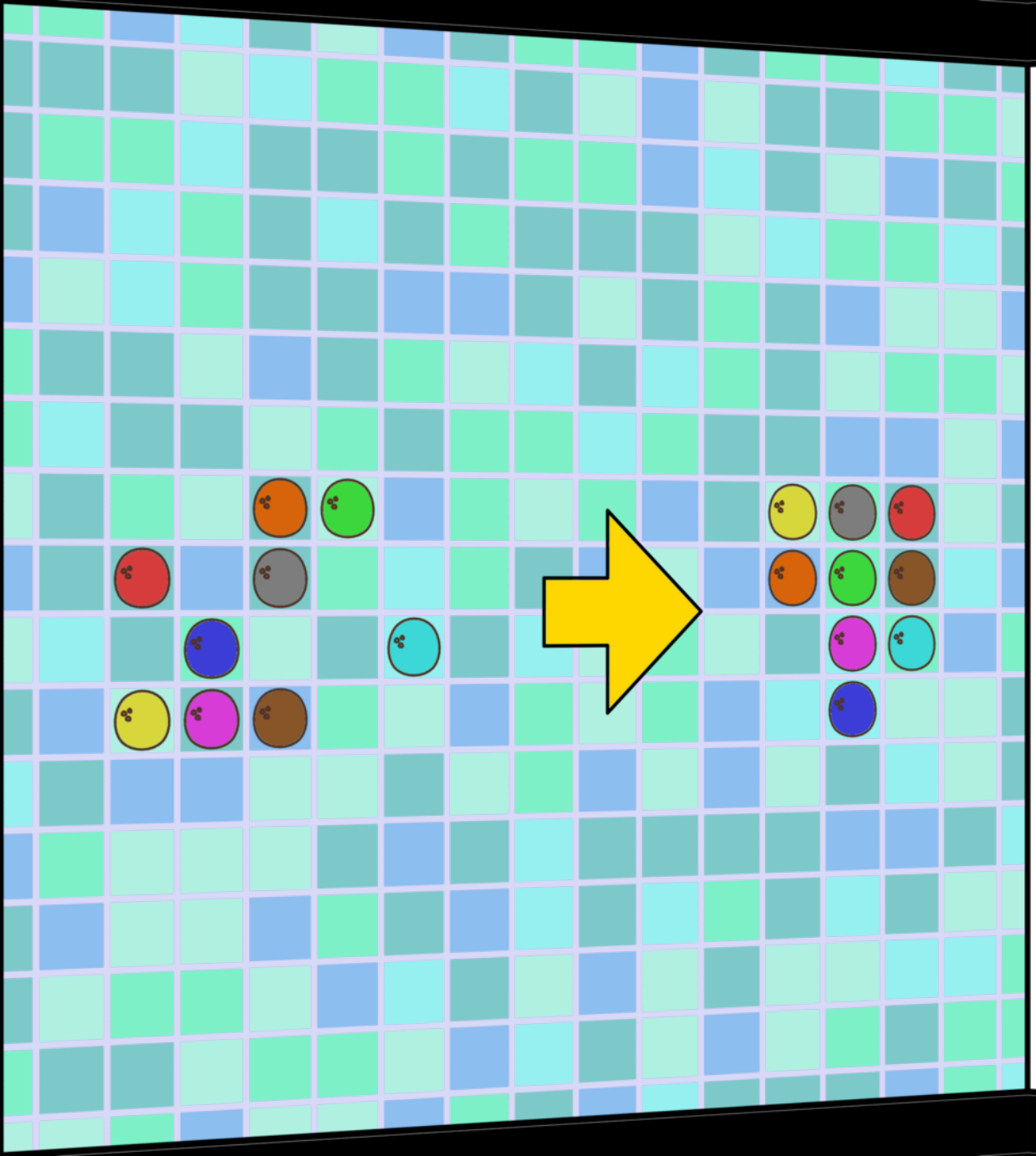
Assume further that we have a giant *coconut pusher*, which can be used to push coconuts.

Any sequence of such coconut pushes leads to a *reconfiguration* of the coconuts.



Problem 1. *(unlabeled coconut reconfiguration)*

Given a starting configuration P and a goal configuration P', is there a sequence of pushes that transforms P into P'?

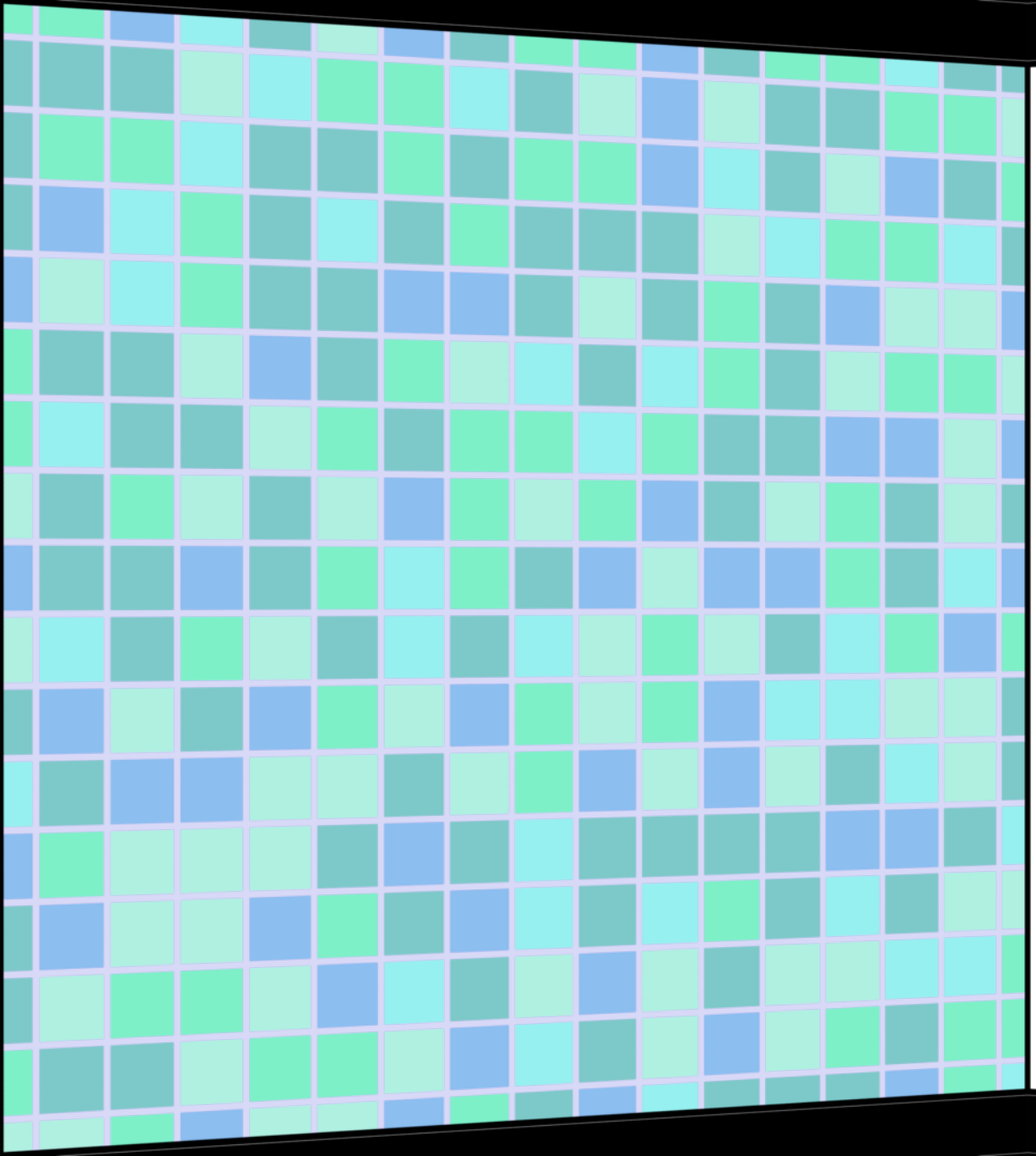


Problem 1. (*unlabeled coconut reconfiguration*)

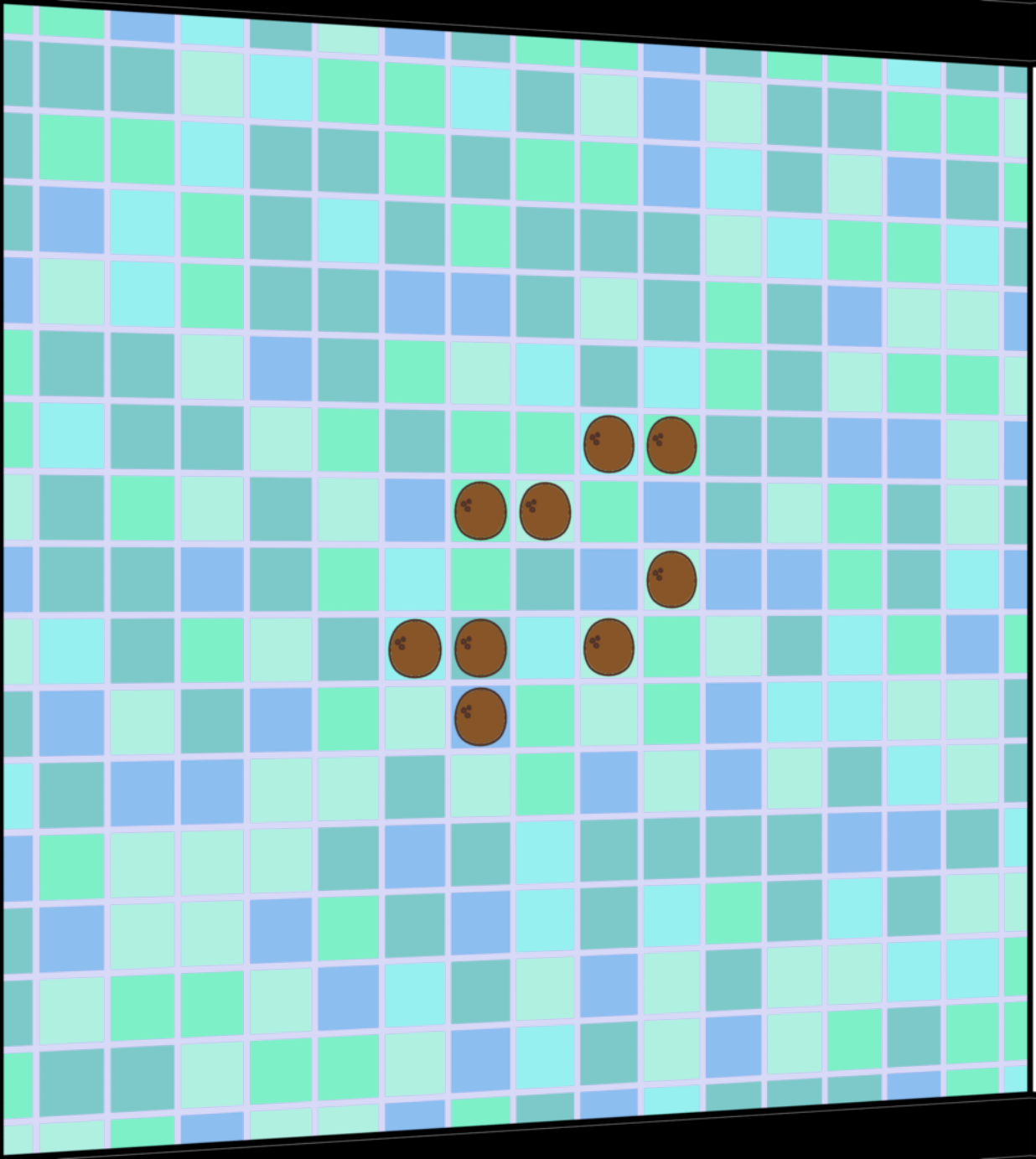
Given a starting configuration P and a goal configuration P', is there a sequence of pushes that transforms P into P'?

Problem 2. (*labeled coconut reconfiguration*)

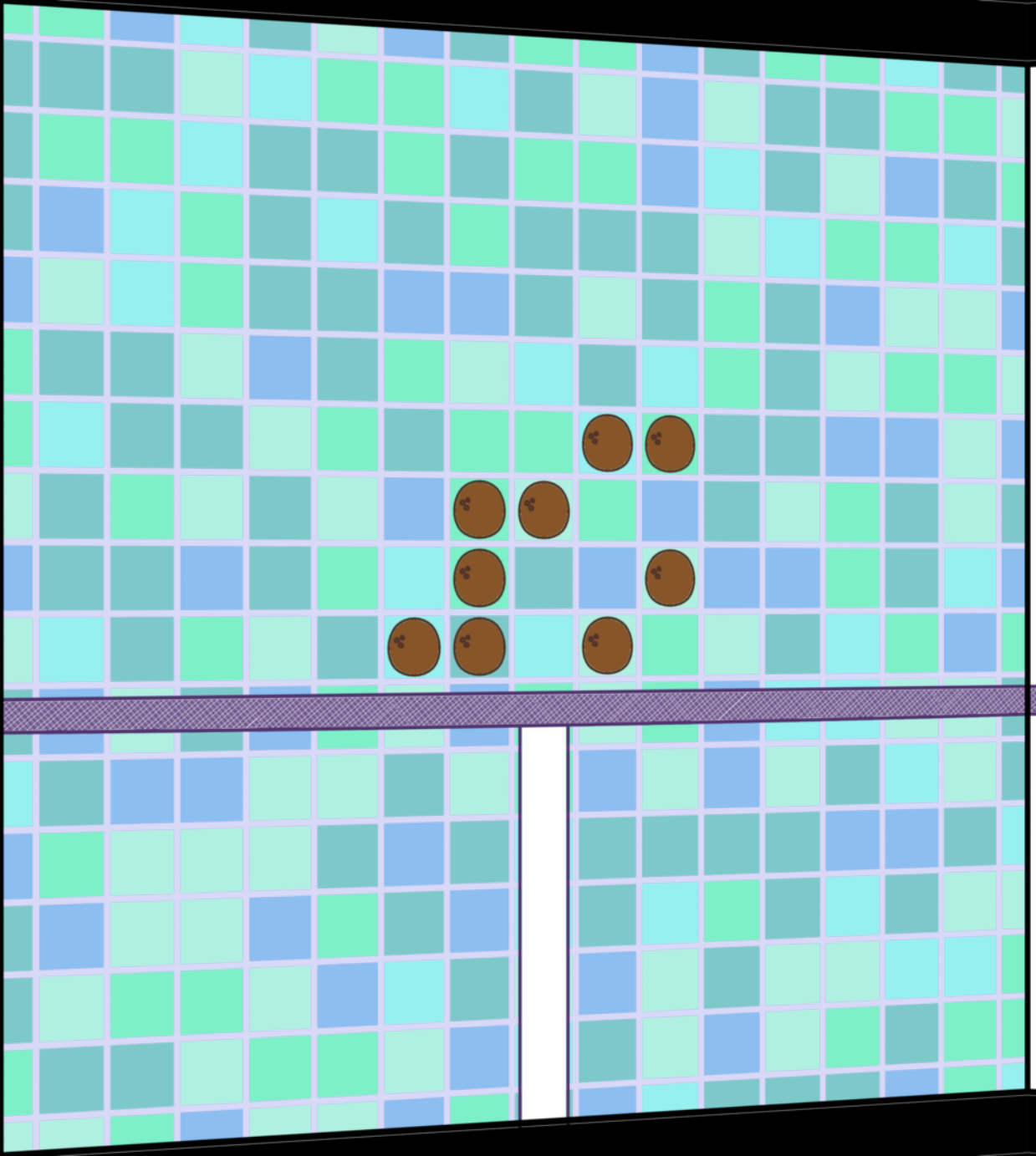
Same thing, with labels.



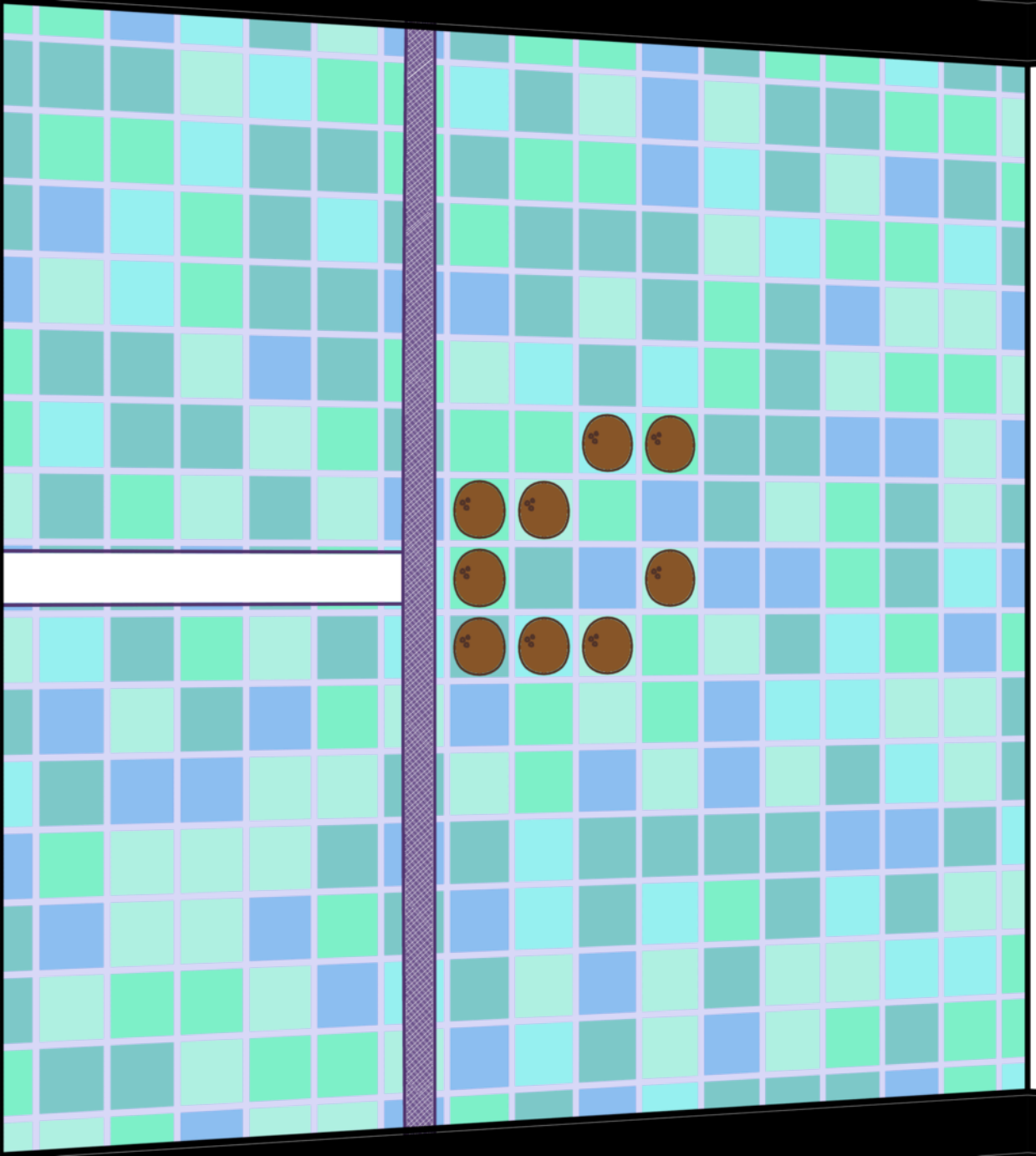
Can we always push our coconuts into tidy rectangles?



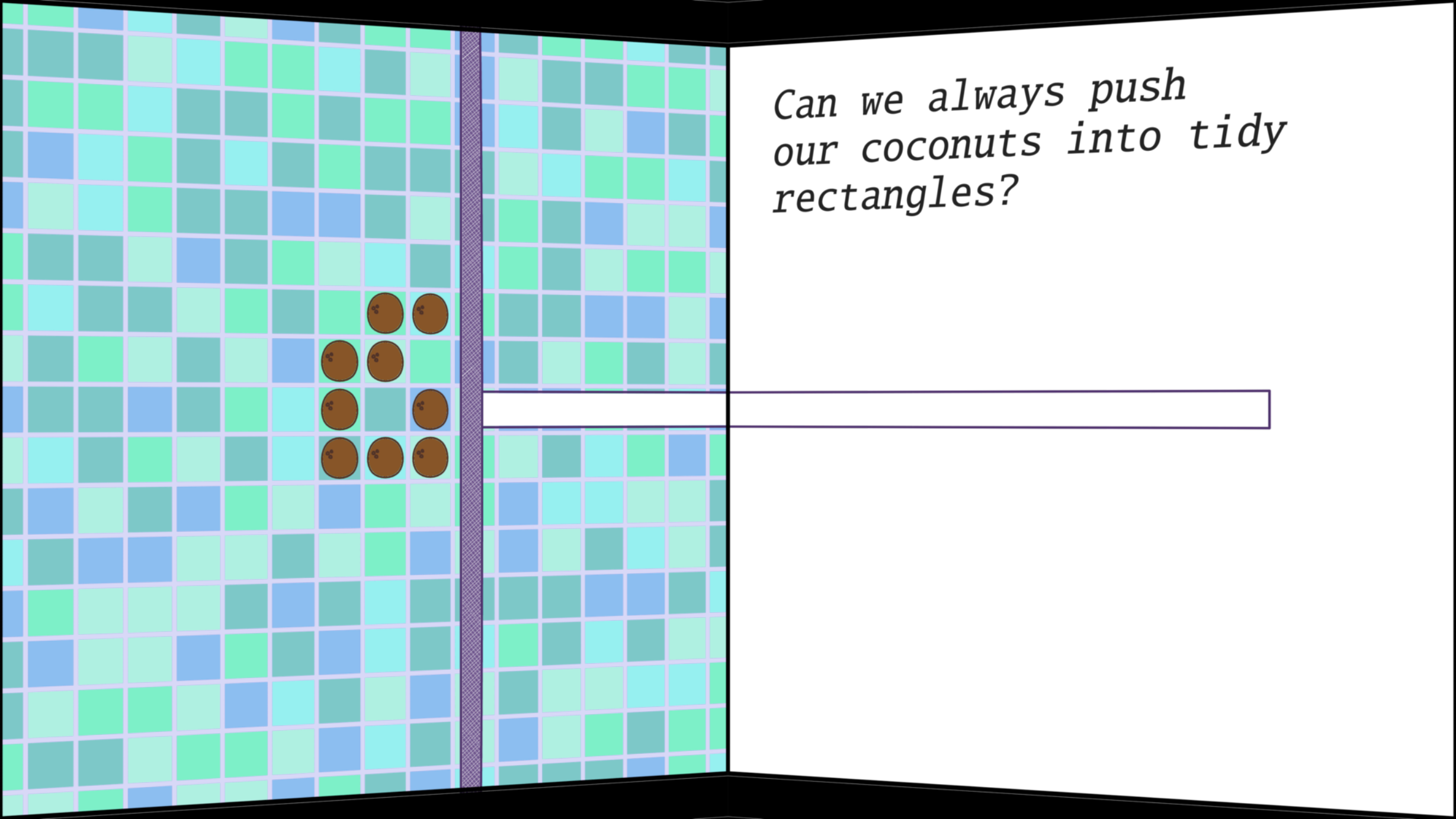
Can we always push our coconuts into tidy rectangles?

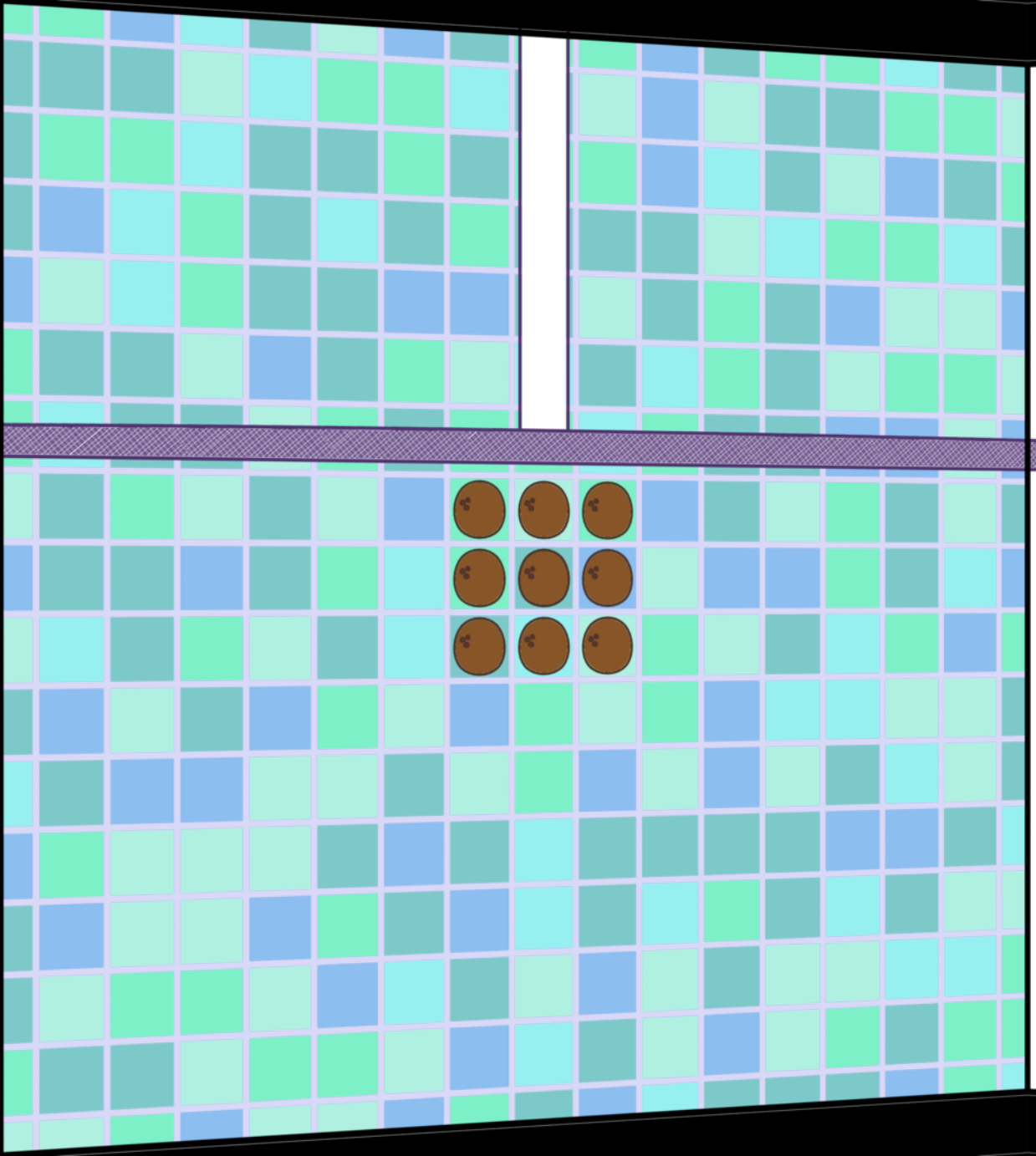


Can we always push our coconuts into tidy rectangles?

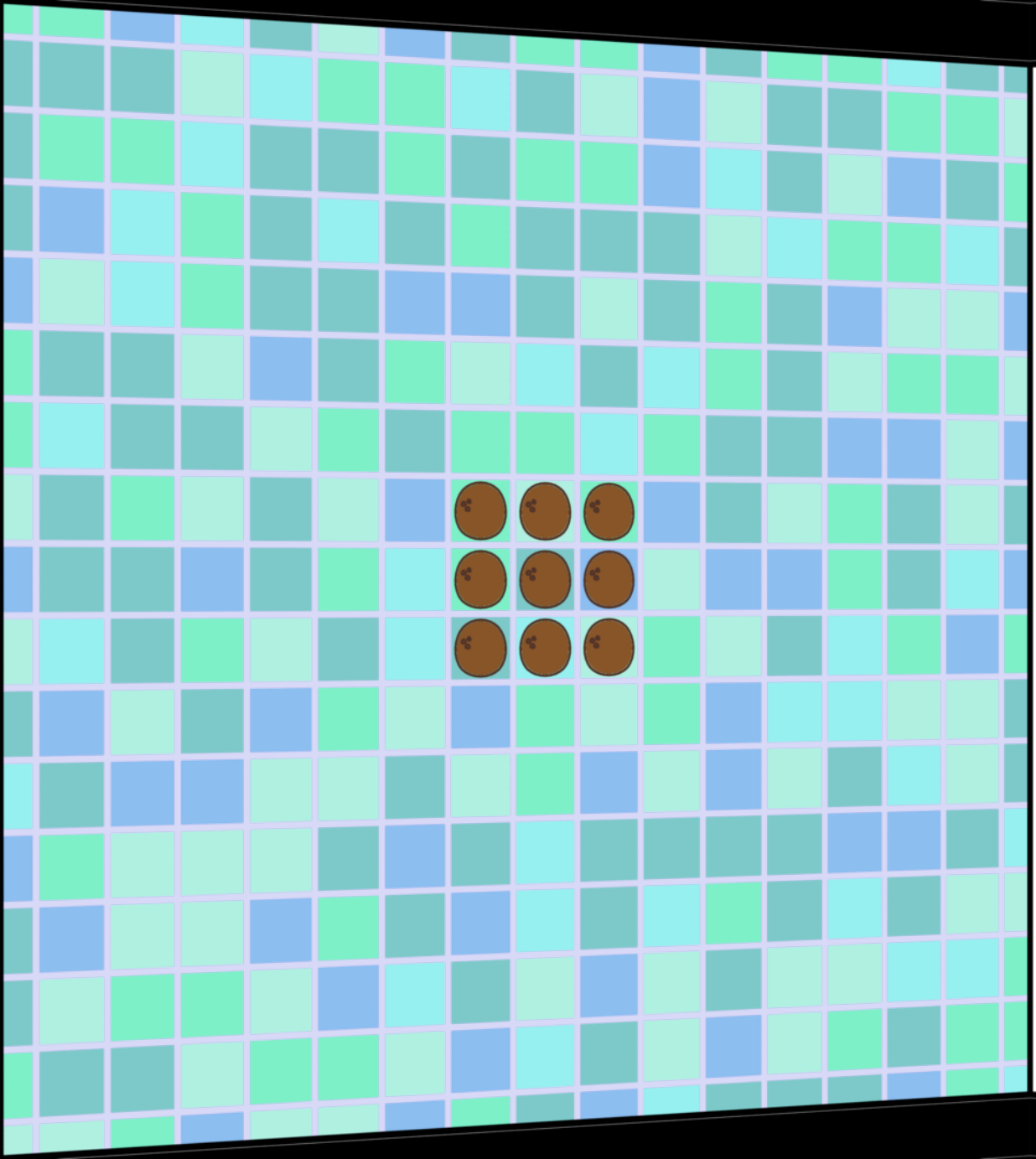


Can we always push our coconuts into tidy rectangles?

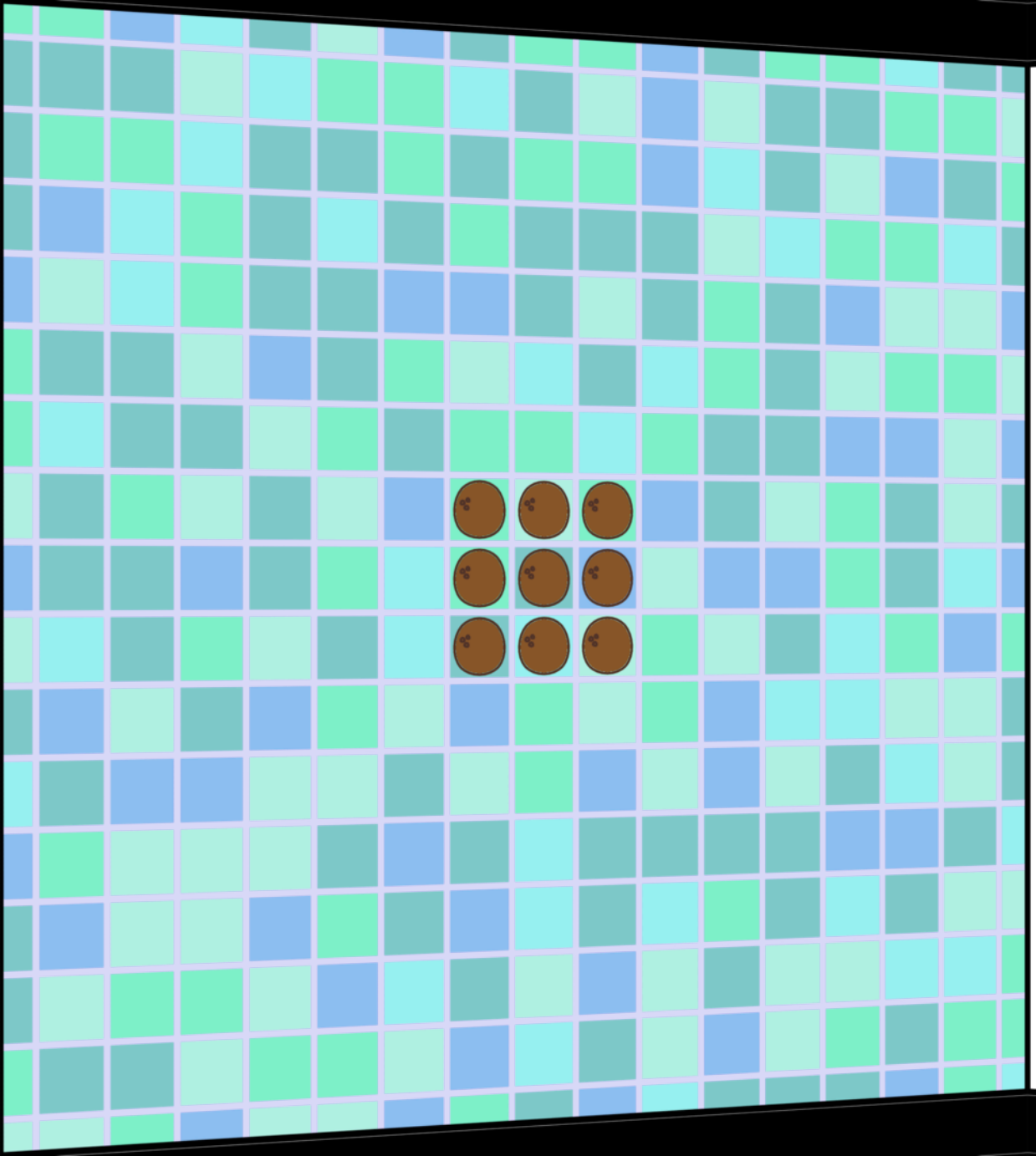

Can we always push
our coconuts into tidy
rectangles?



Can we always push our coconuts into tidy rectangles?

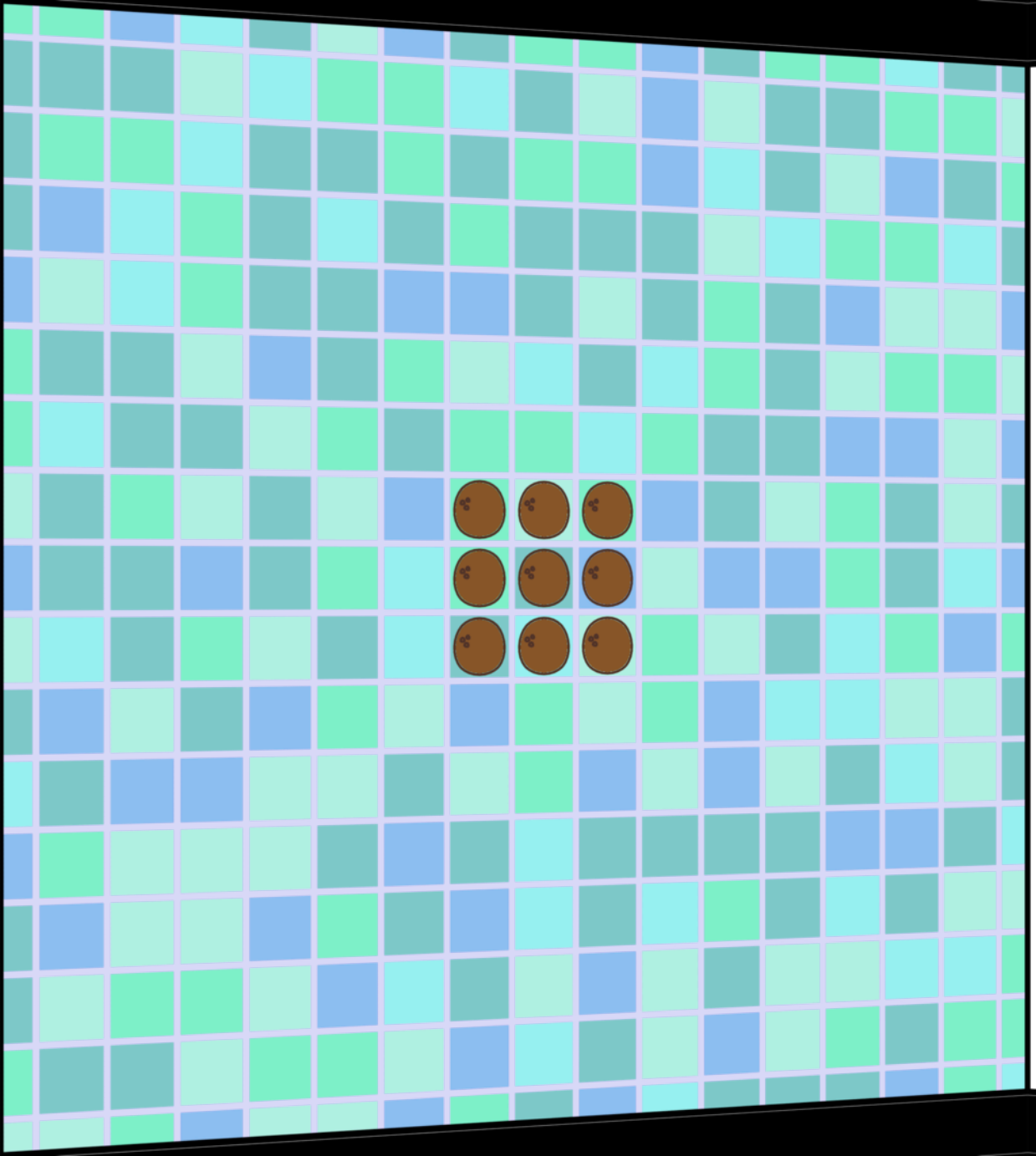


Can we always push our coconuts into tidy rectangles?



Can we always push our coconuts into tidy rectangles?

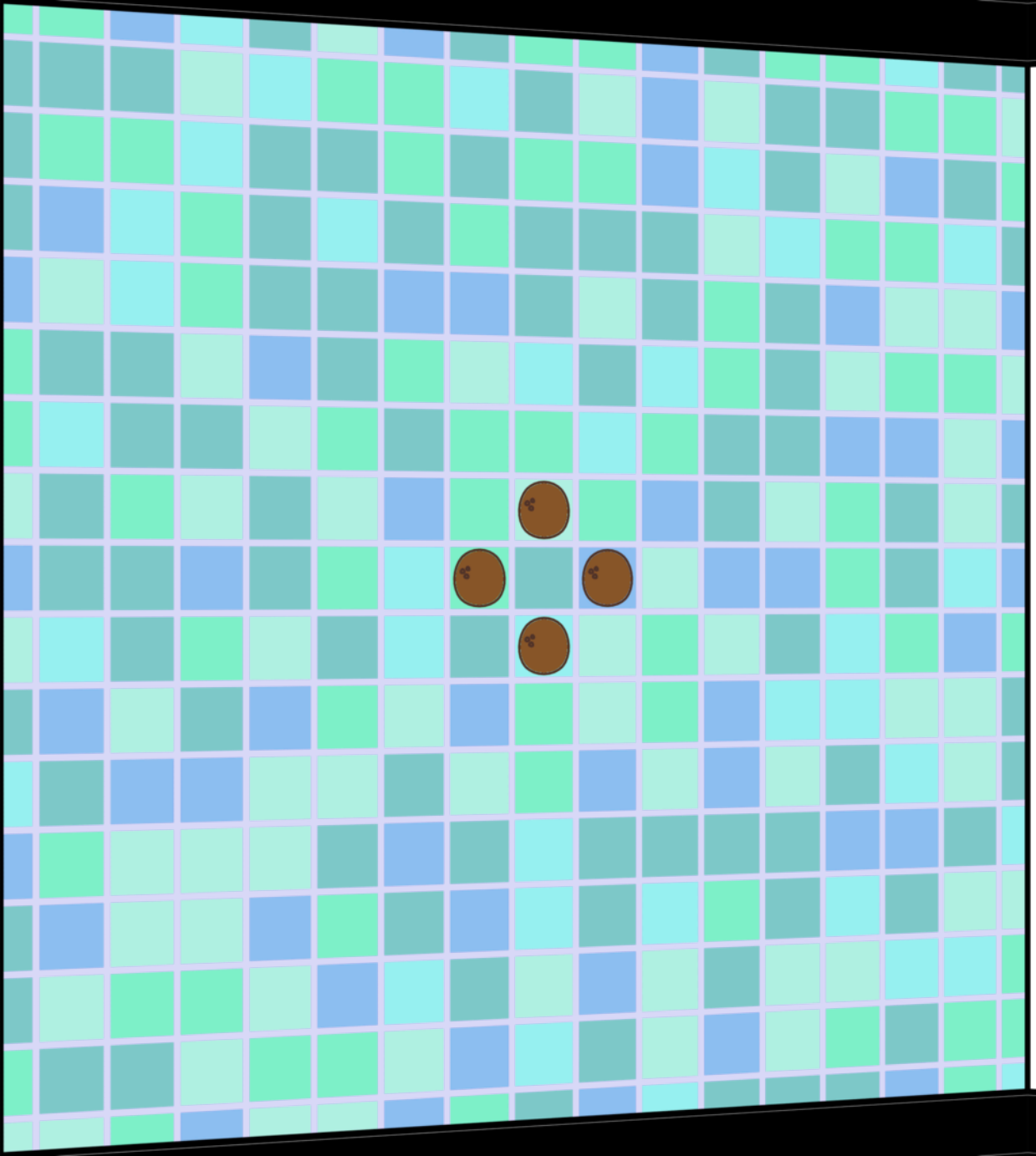
Obviously, n coconuts can only be pushed into a $a \times b$ rectangle if $ab = n$!



Can we always push our coconuts into tidy rectangles?

Obviously, n coconuts can only be pushed into a $a \times b$ rectangle if $ab = n$!

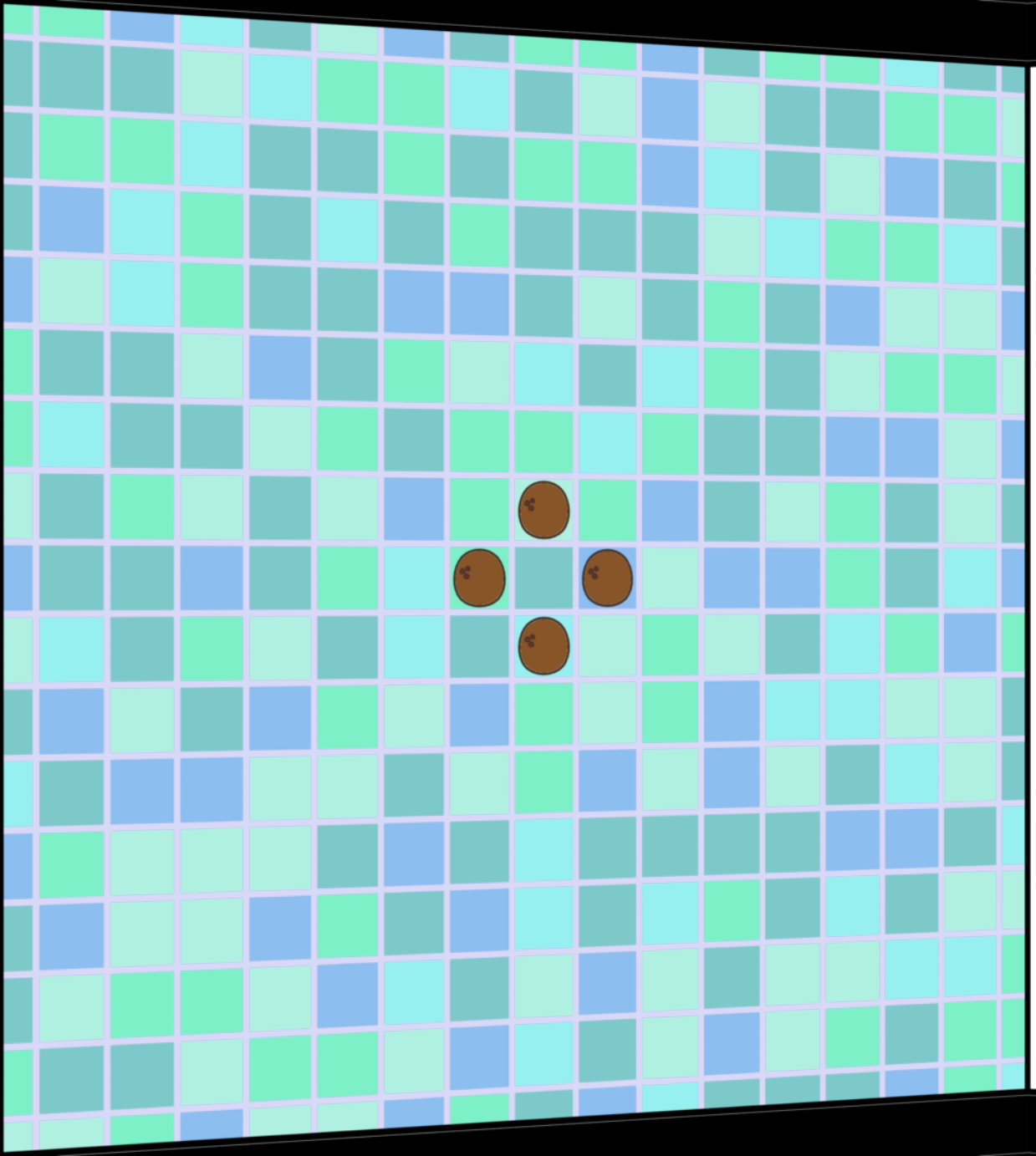
But is it always possible?



Can we always push our coconuts into tidy rectangles?

Obviously, n coconuts can only be pushed into a $a \times b$ rectangle if $ab = n$!

But is it always possible?

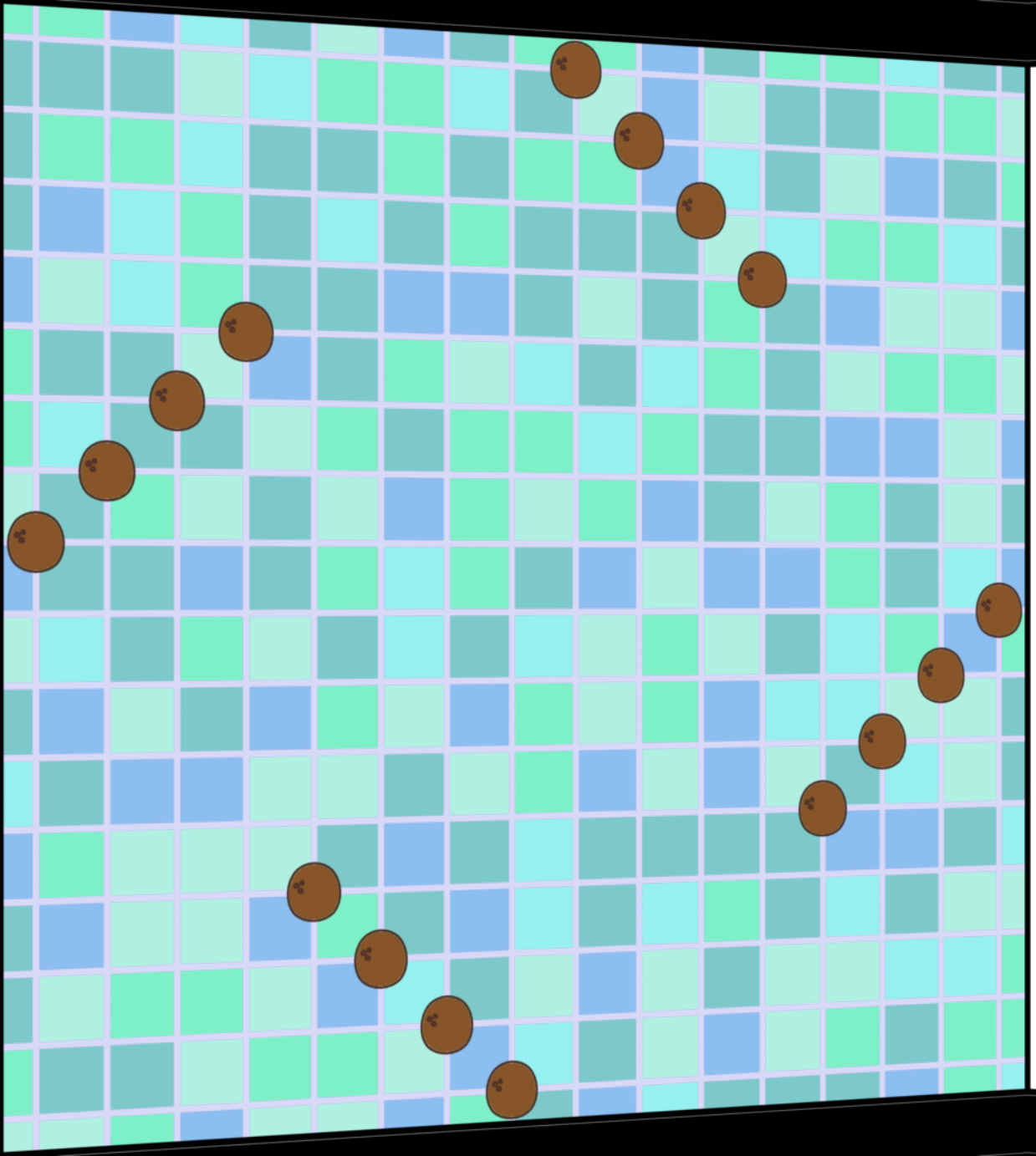


Can we always push our coconuts into tidy rectangles?

Obviously, n coconuts can only be pushed into a $a \times b$ rectangle if $ab = n$!

But is it always possible?

No!

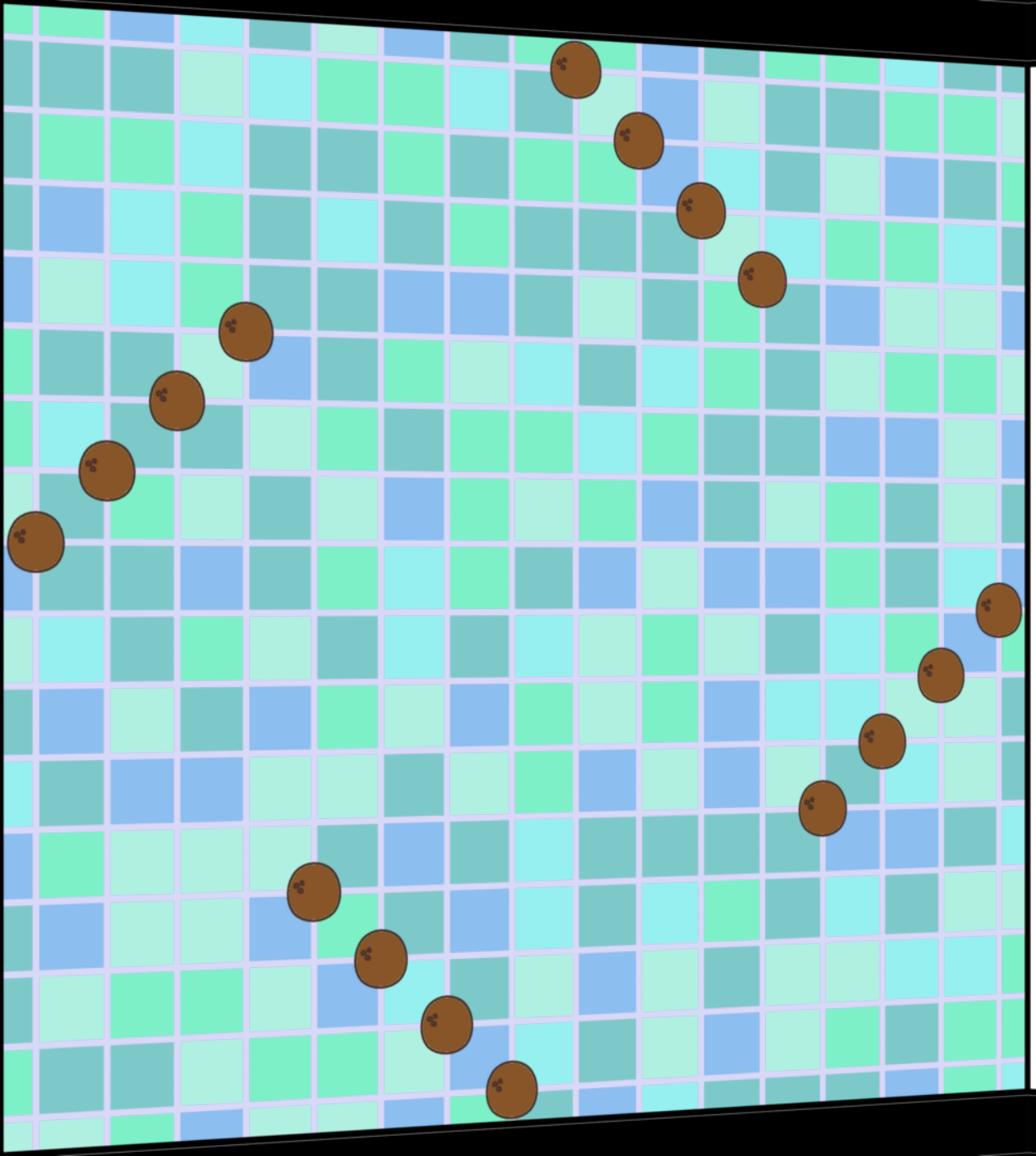


Can we always push our coconuts into tidy rectangles?

Obviously, n coconuts can only be pushed into a $a \times b$ rectangle if $ab = n$!

But is it always possible?

No!



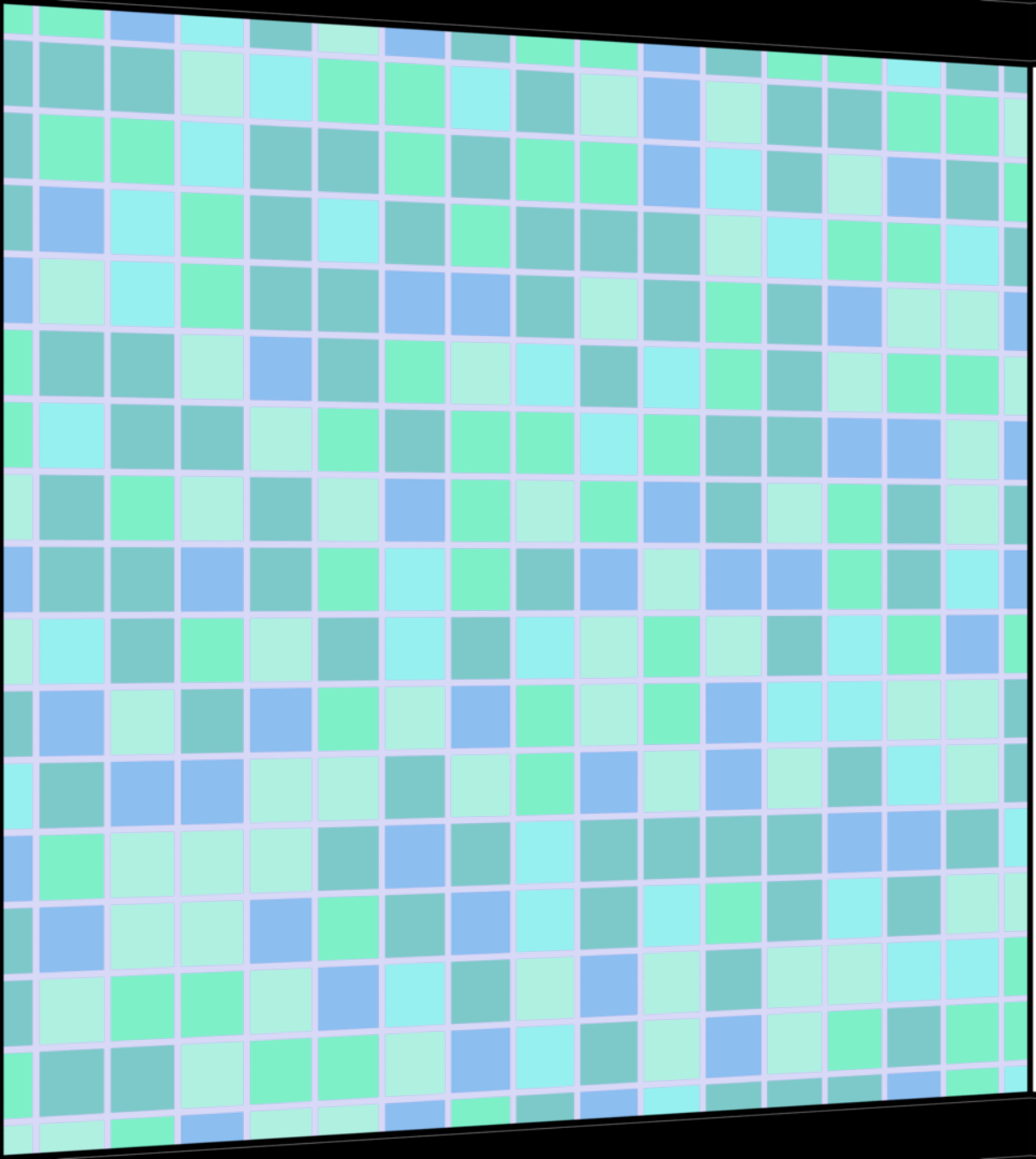
Can we always push our coconuts into tidy rectangles?

Obviously, n coconuts can only be pushed into a $a \times b$ rectangle if $ab = n$!

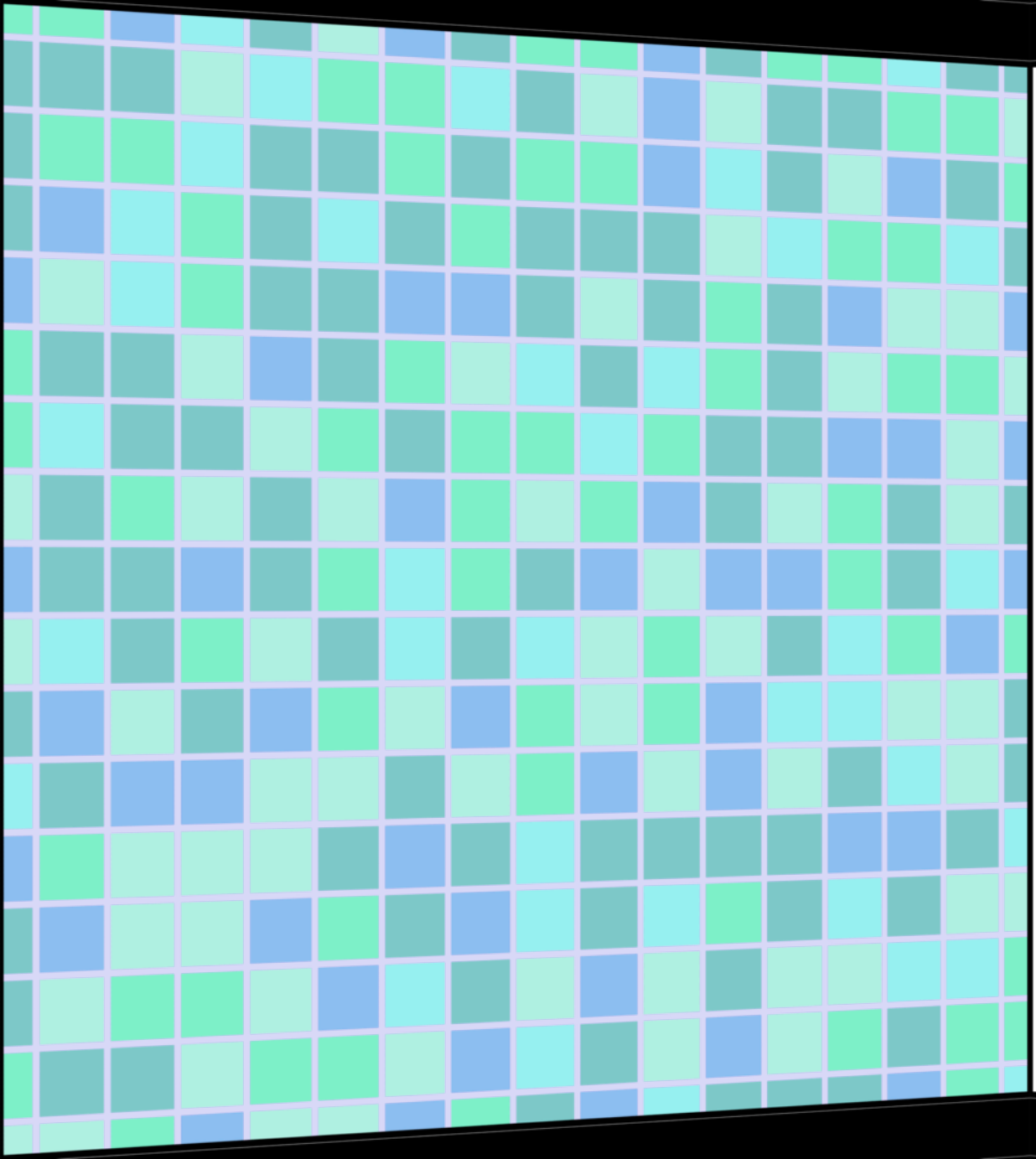
But is it always possible?

No!

...and *some* aspect ratios are not even possible if we start with at most one coconut per row and column.

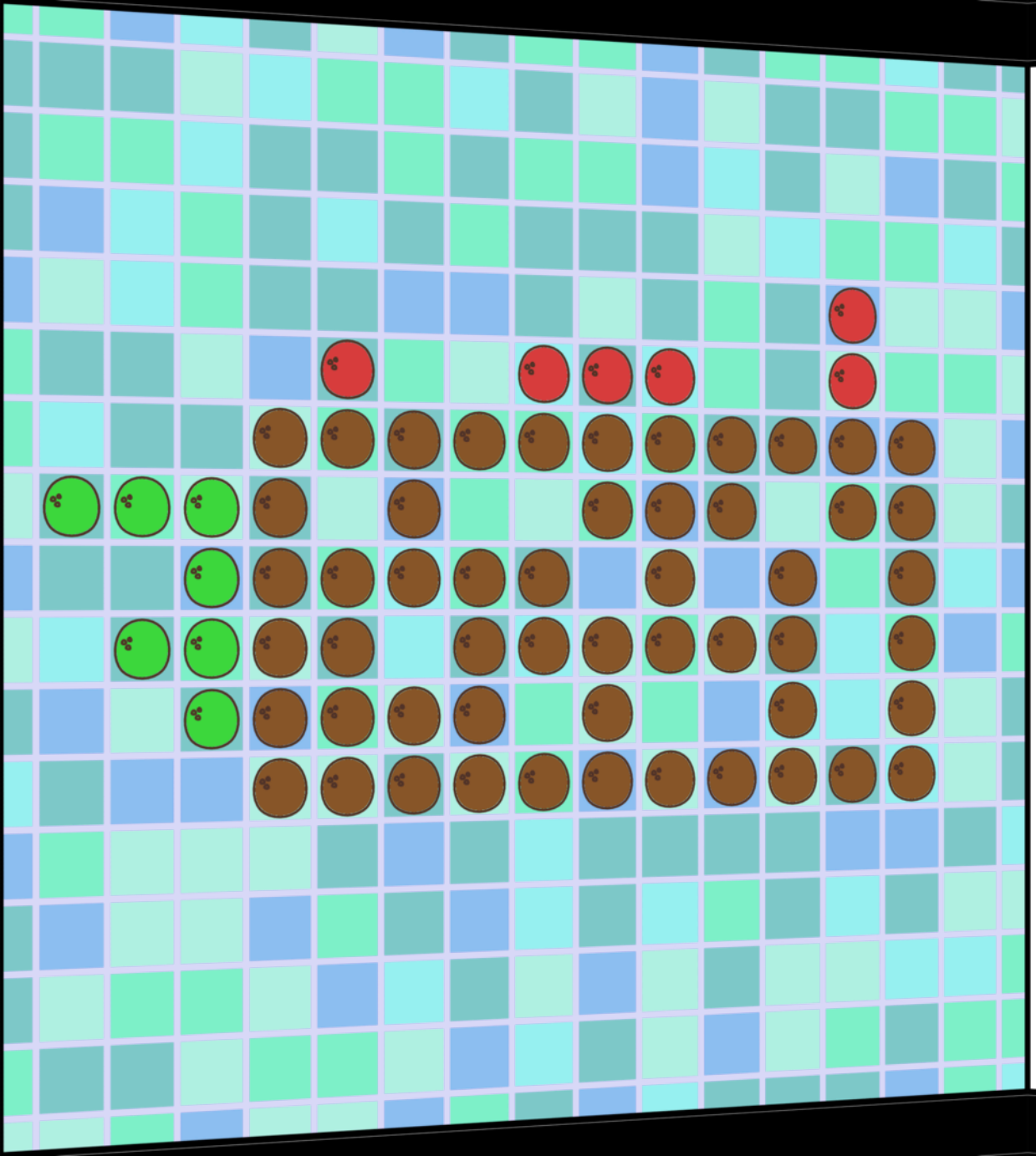


Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.



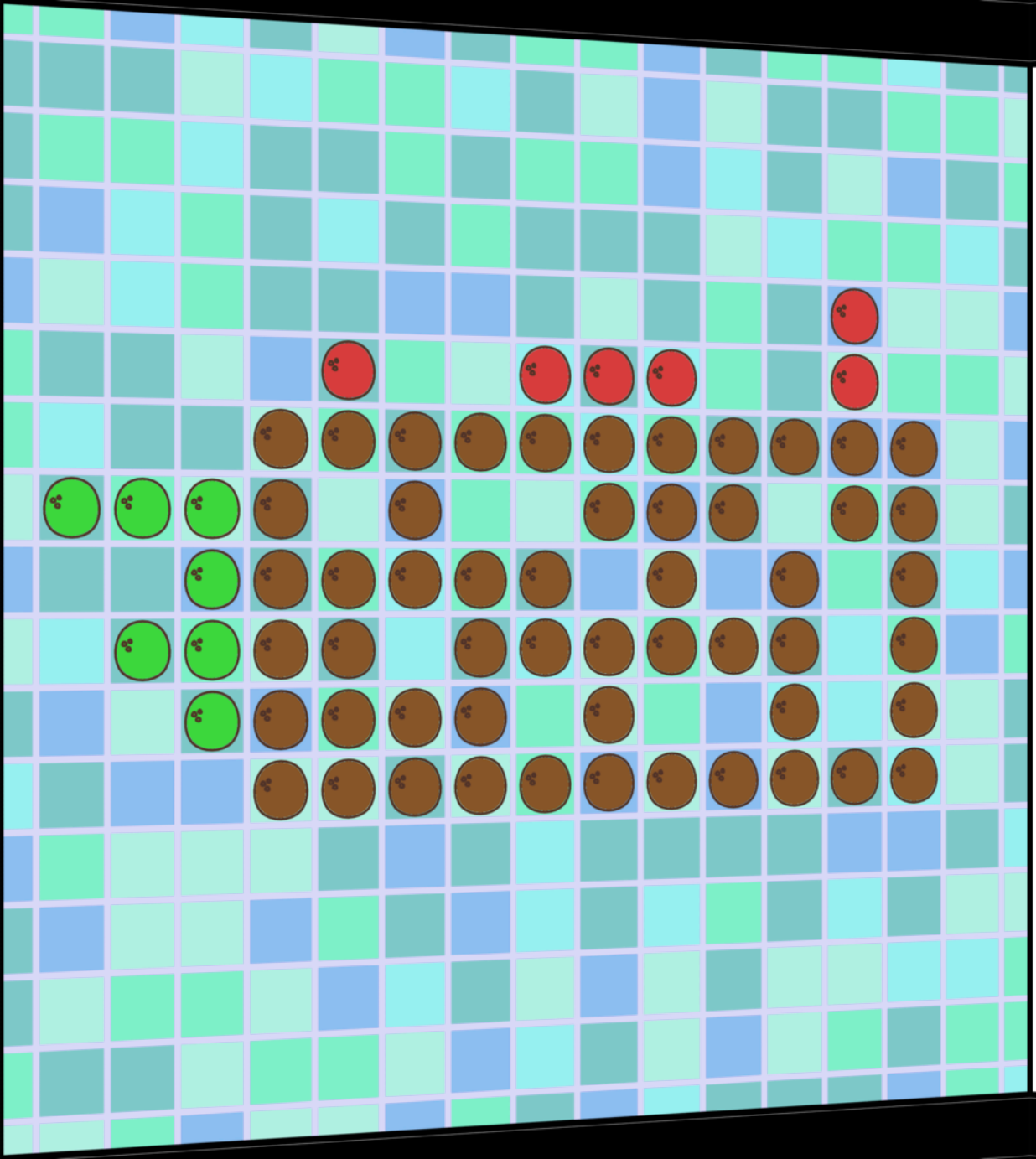
Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

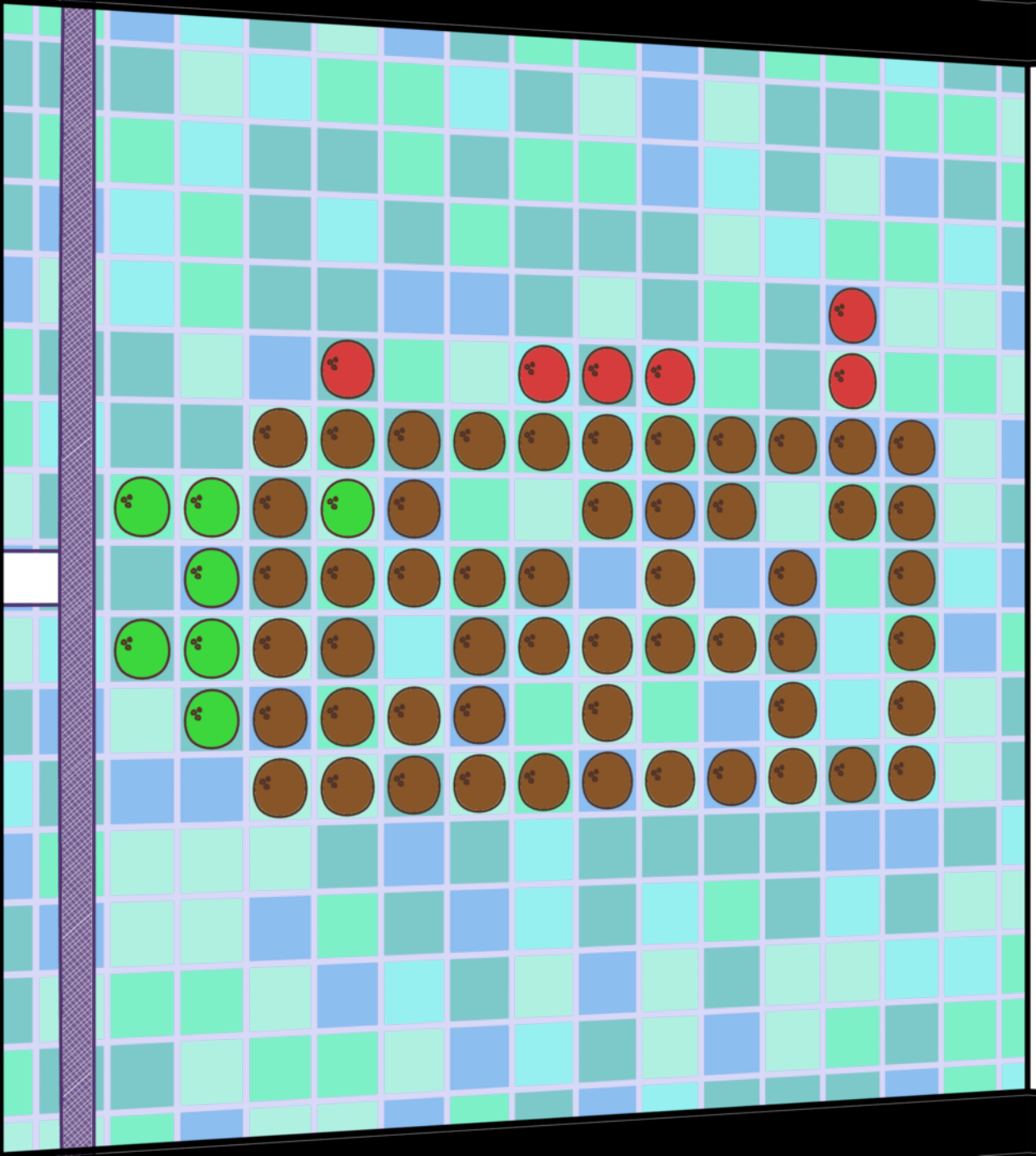
General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

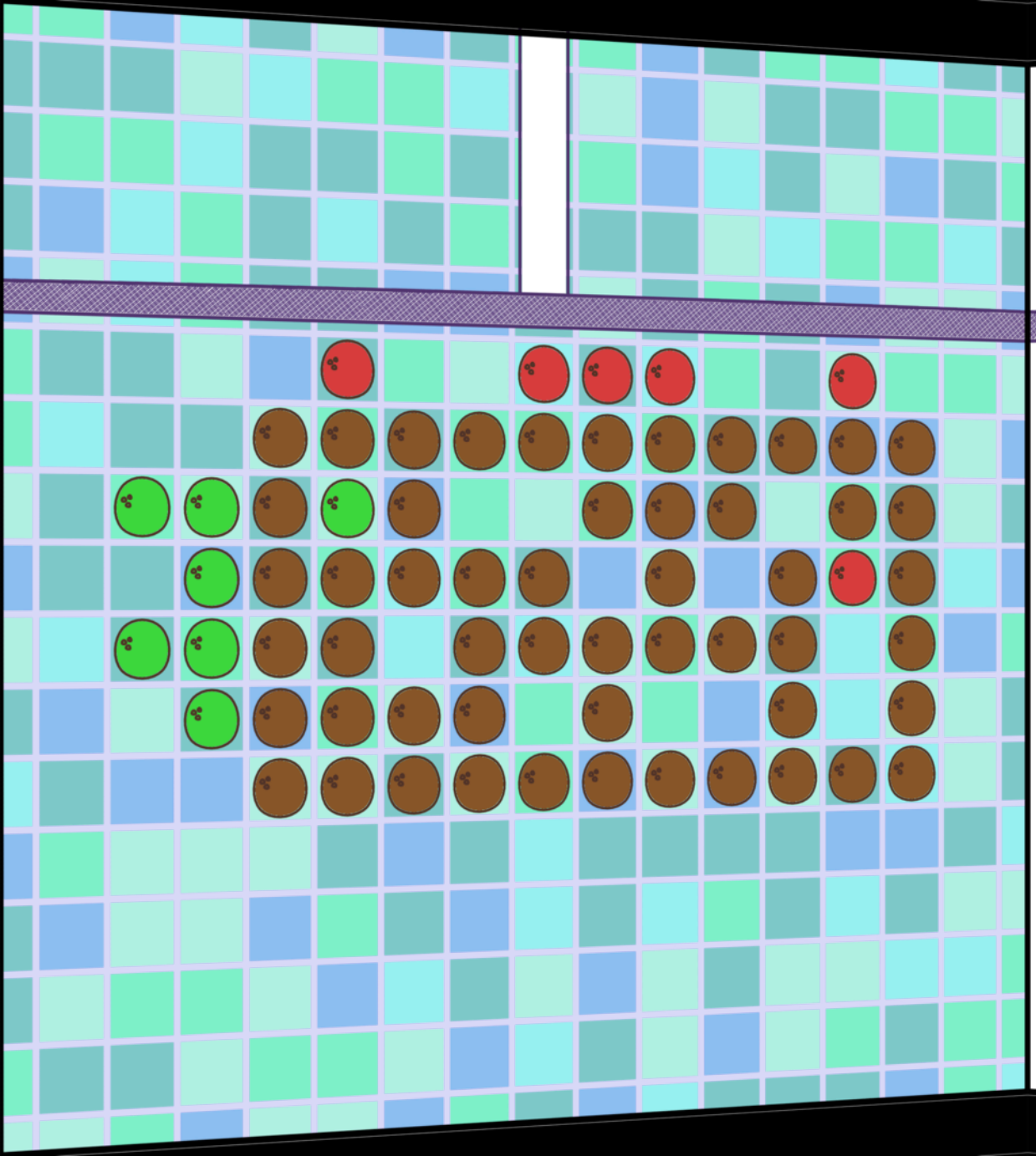
The question then becomes in which order to push to make everything fit.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

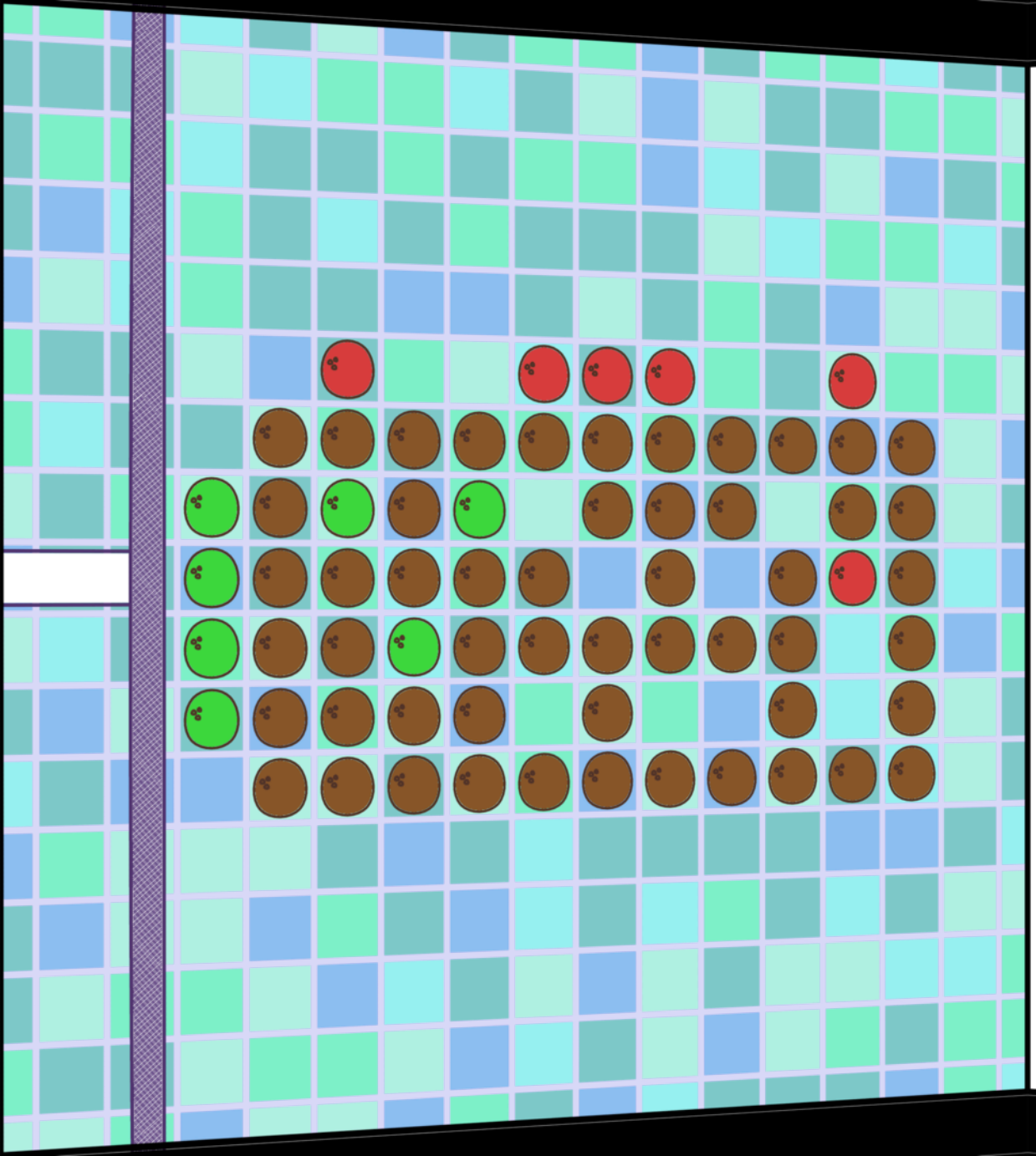
The question then becomes in which order to push to make everything fit.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

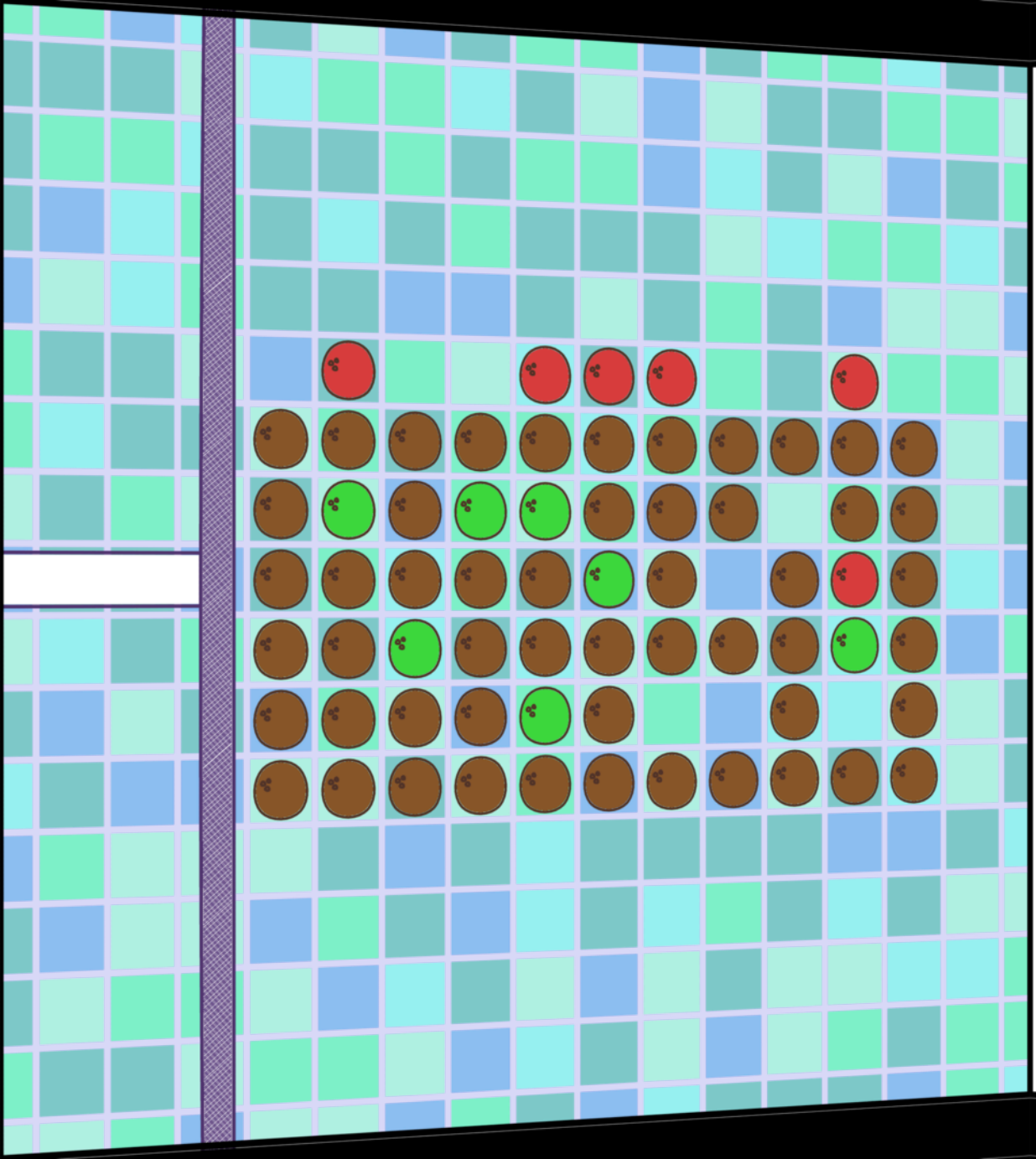
The question then becomes in which order to push to make everything fit.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

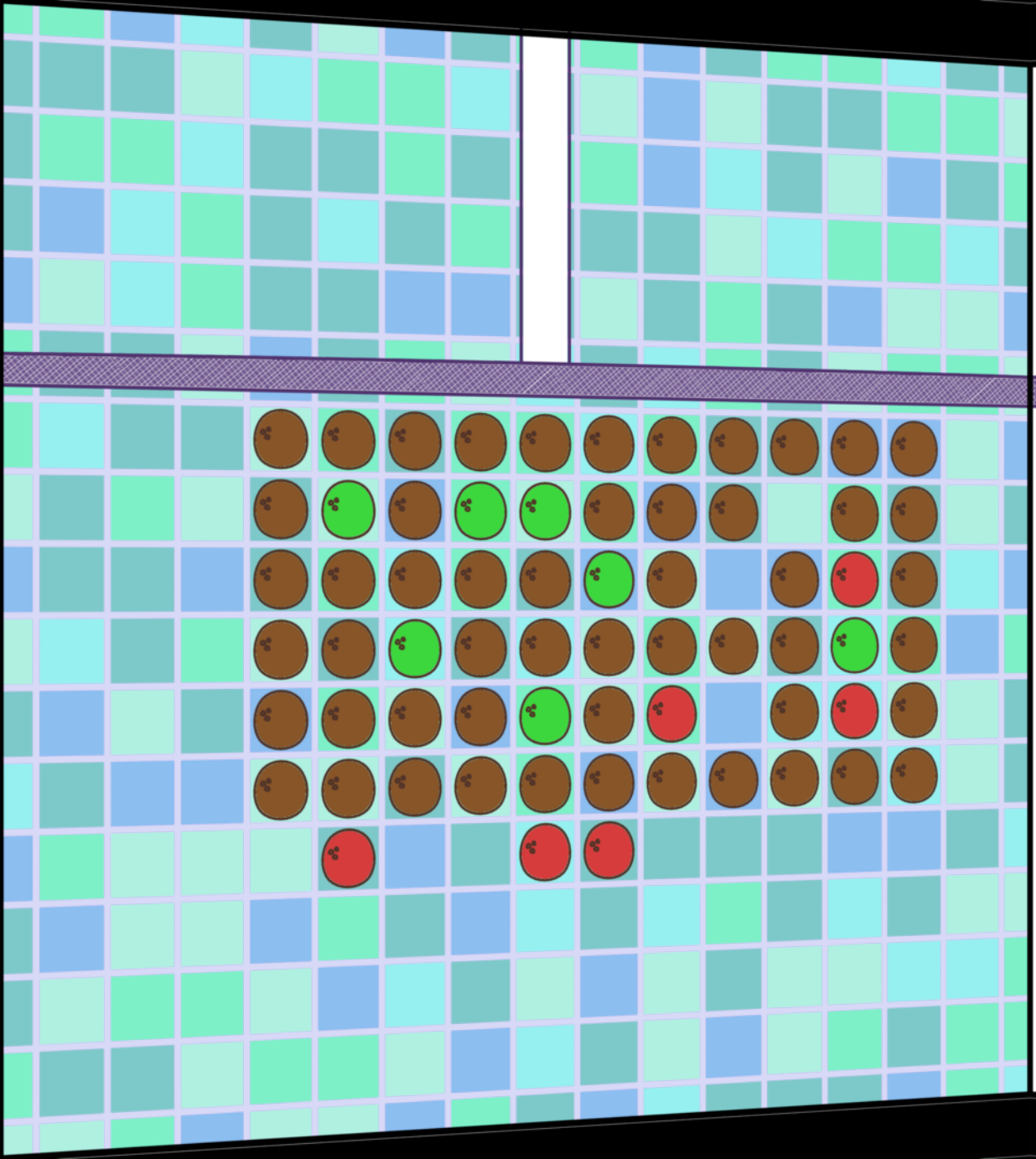
The question then becomes in which order to push to make everything fit.



Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

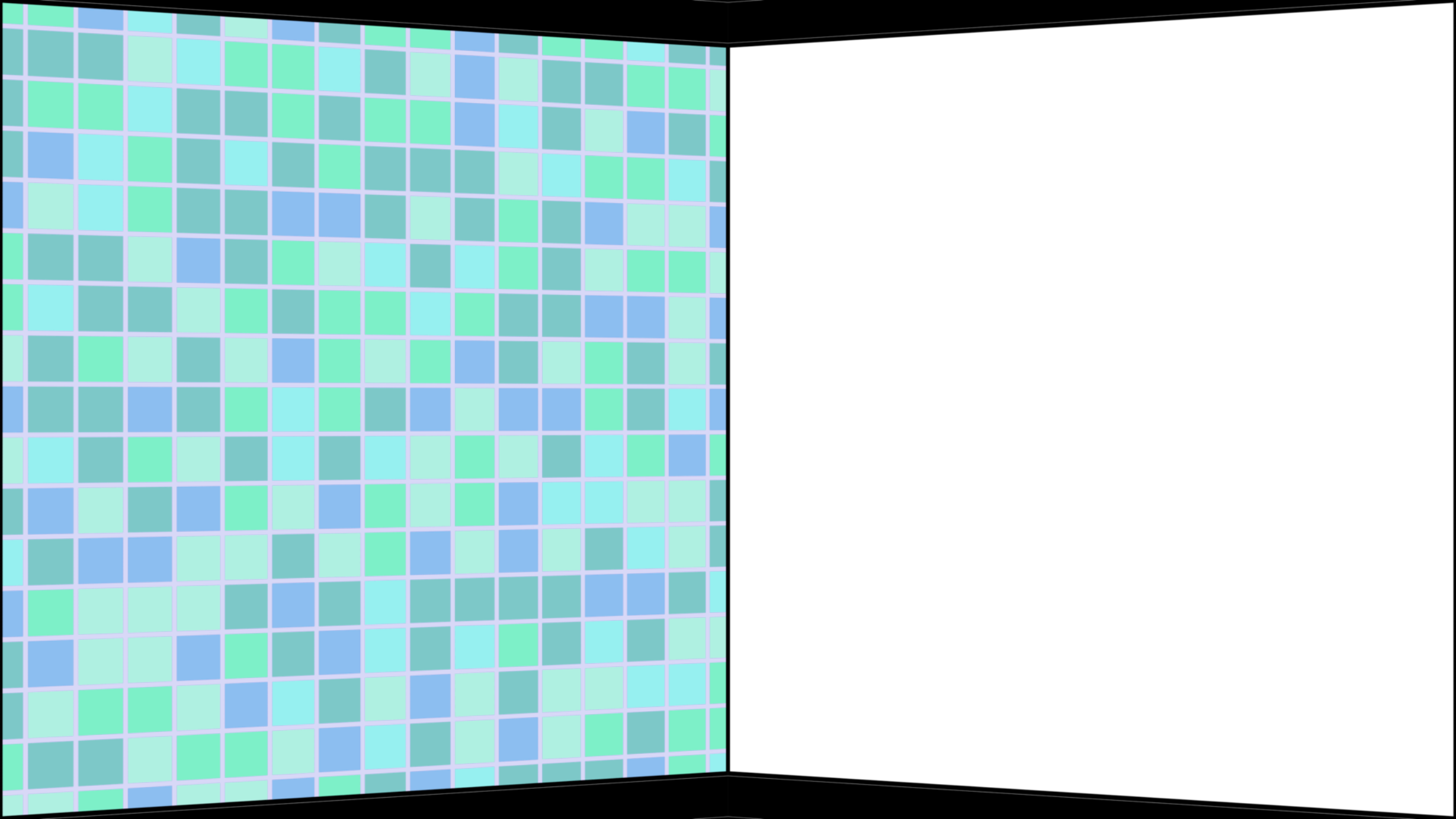
The question then becomes in which order to push to make everything fit.

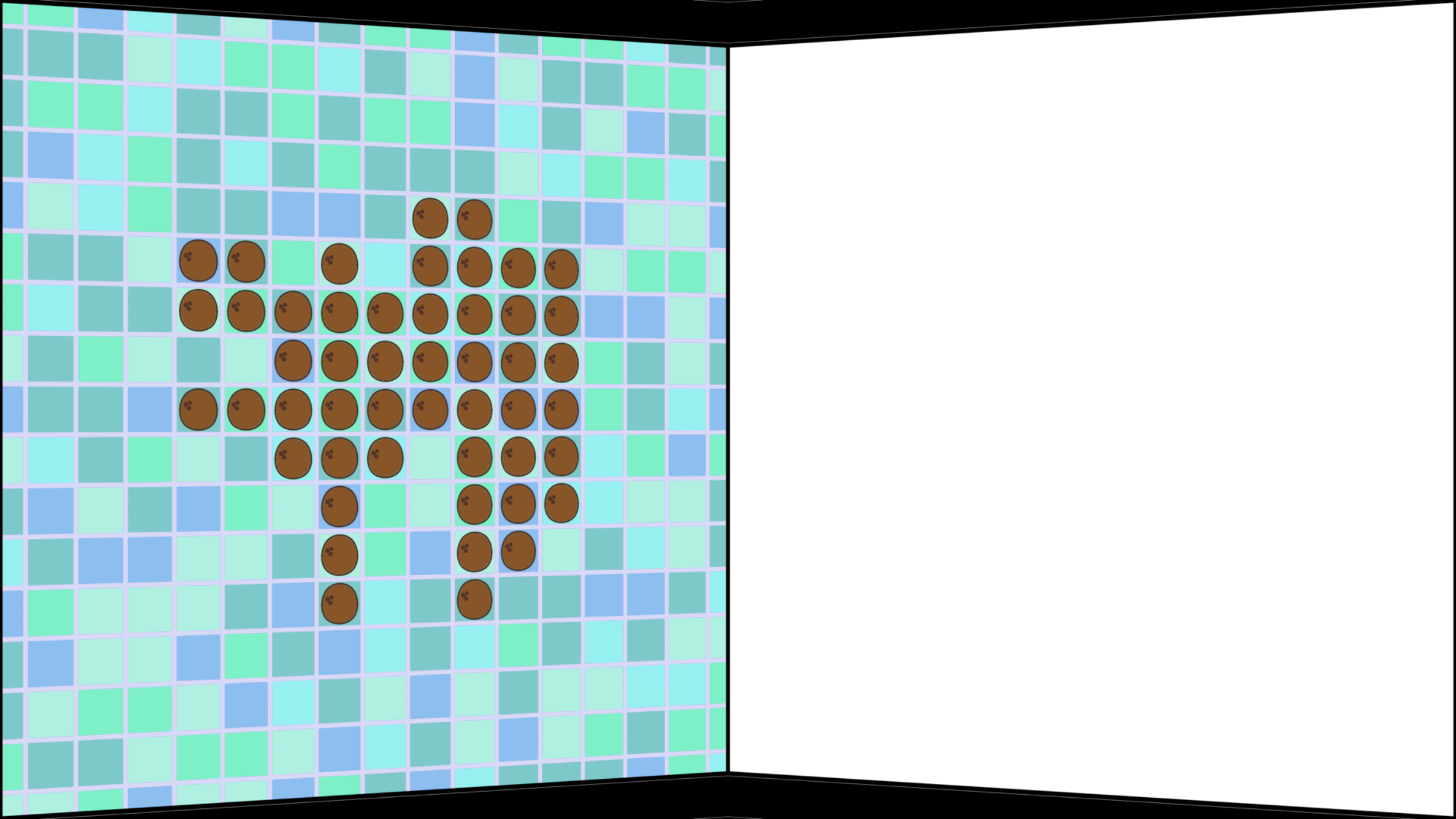


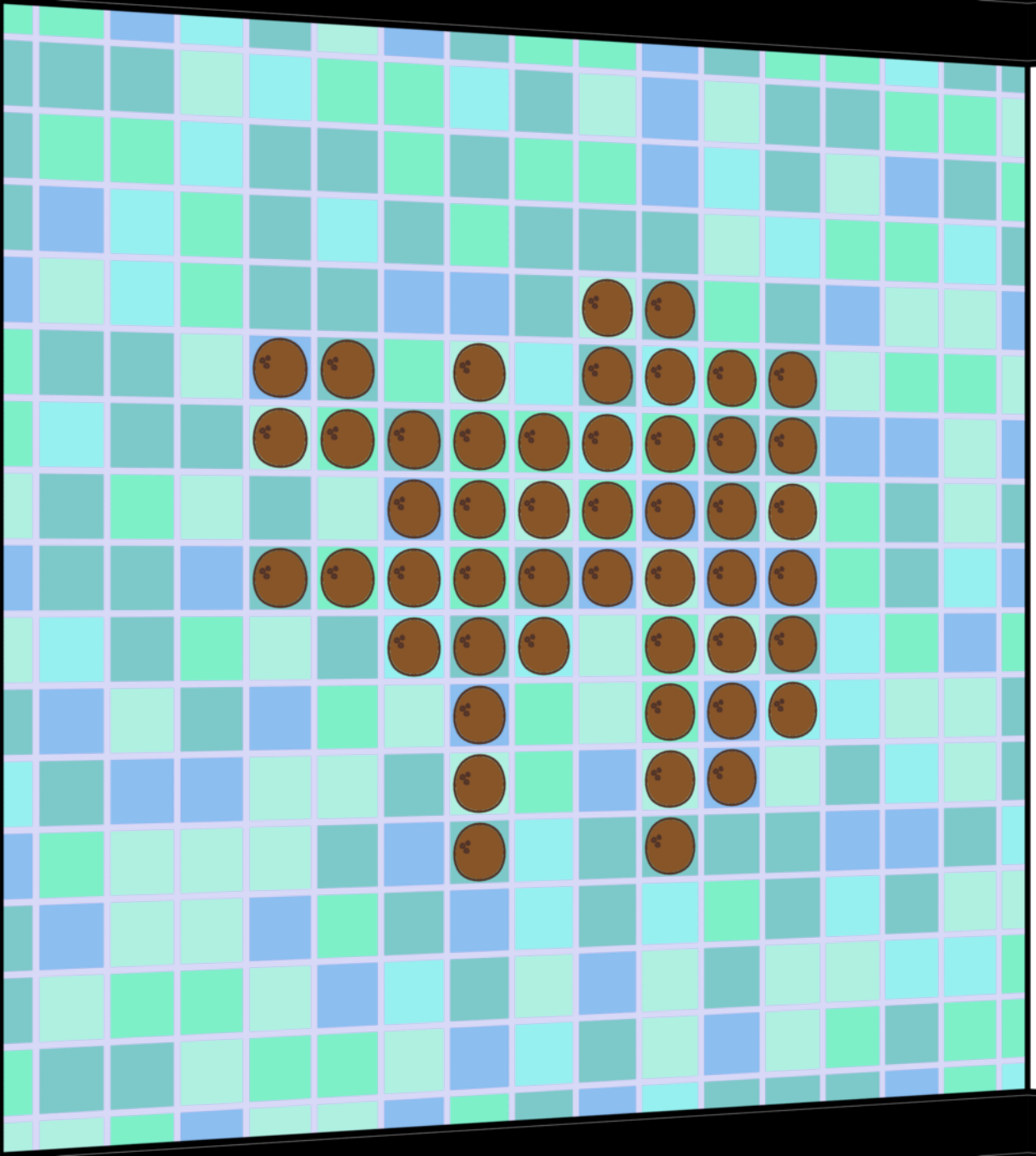
Deciding whether n coconuts can be pushed into an $a \times b$ rectangle is NP-complete.

General idea: we will create a big coconut rectangle with holes, and put some coconuts on the side to fill them.

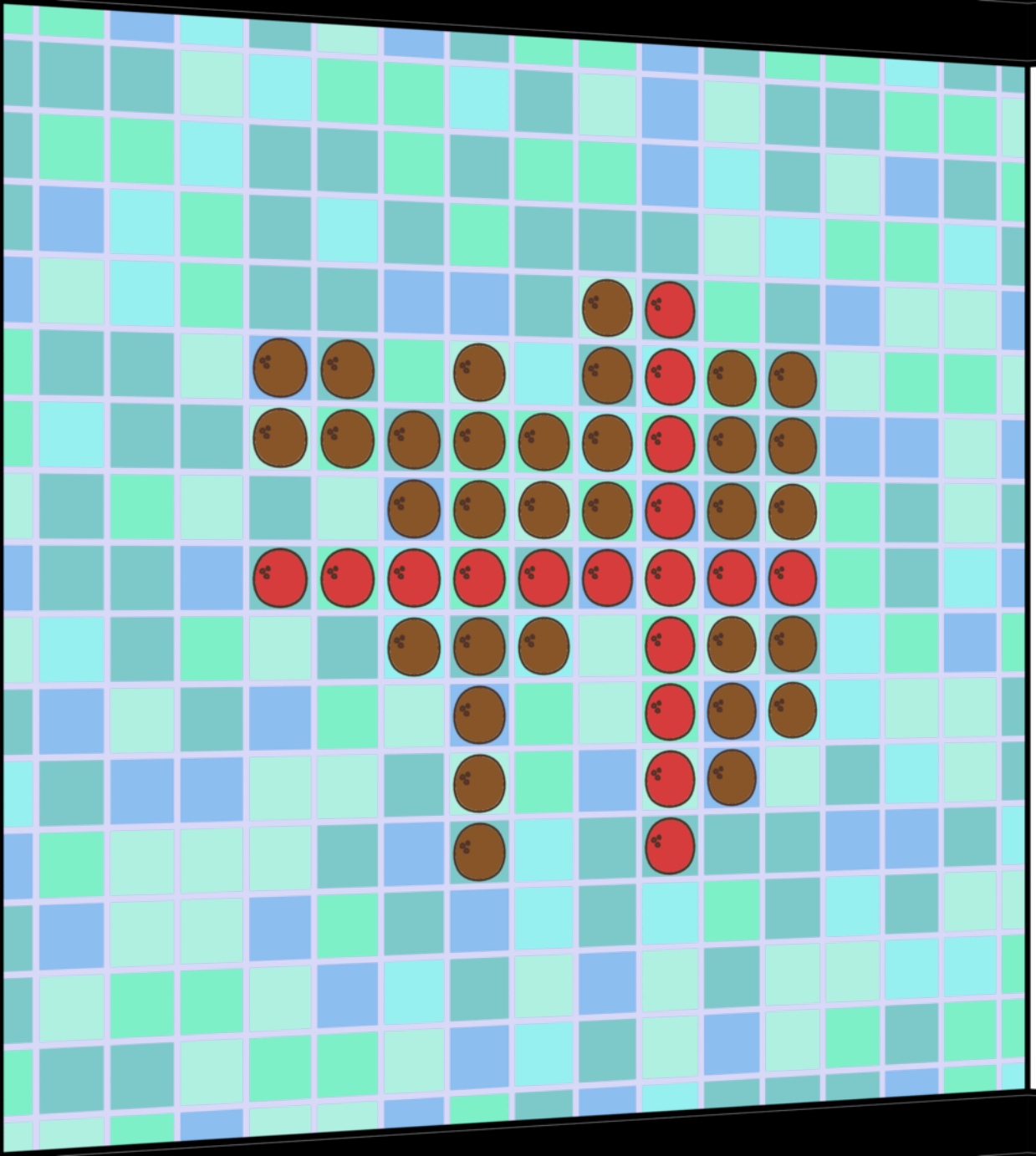
The question then becomes in which order to push to make everything fit.



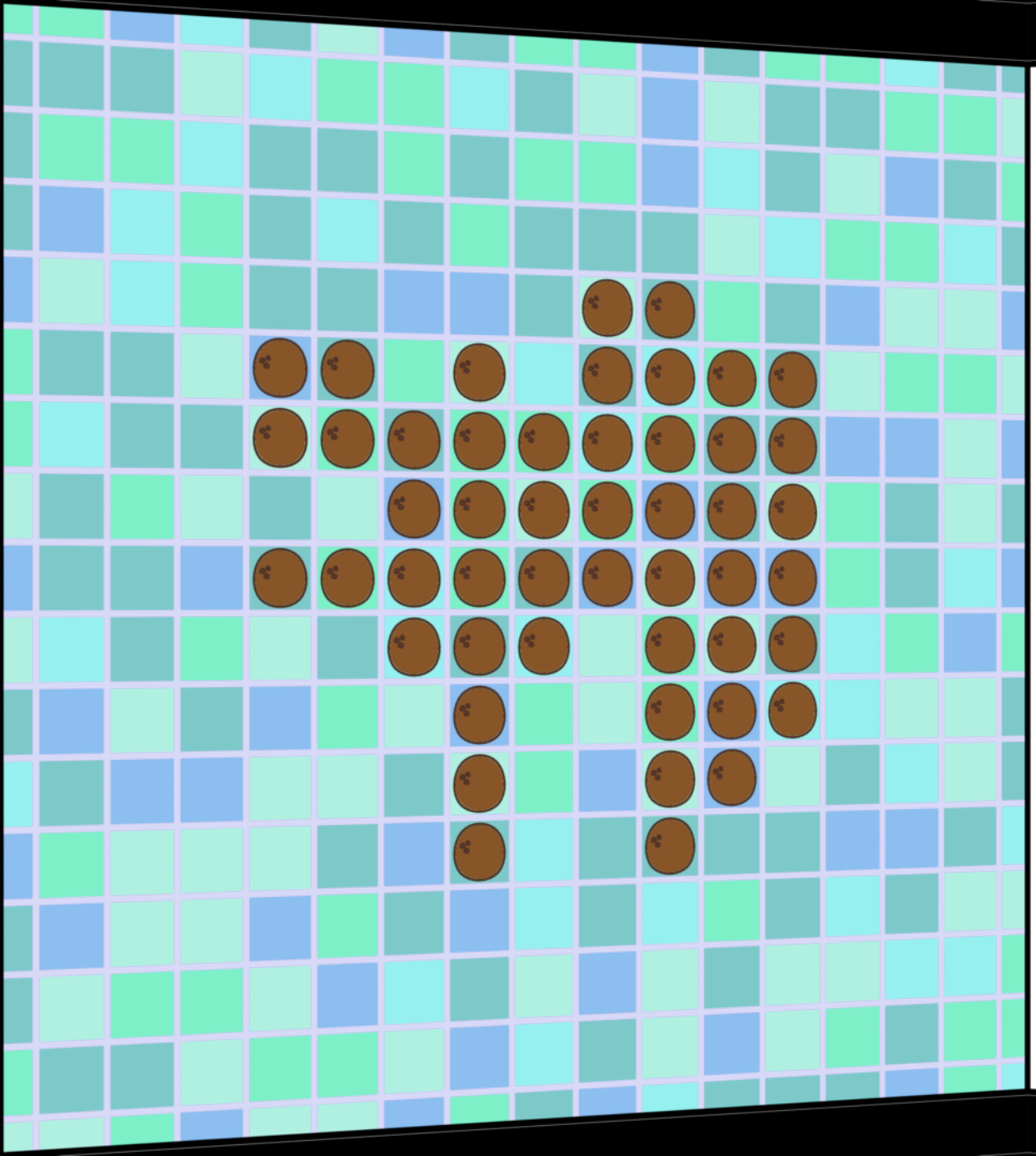




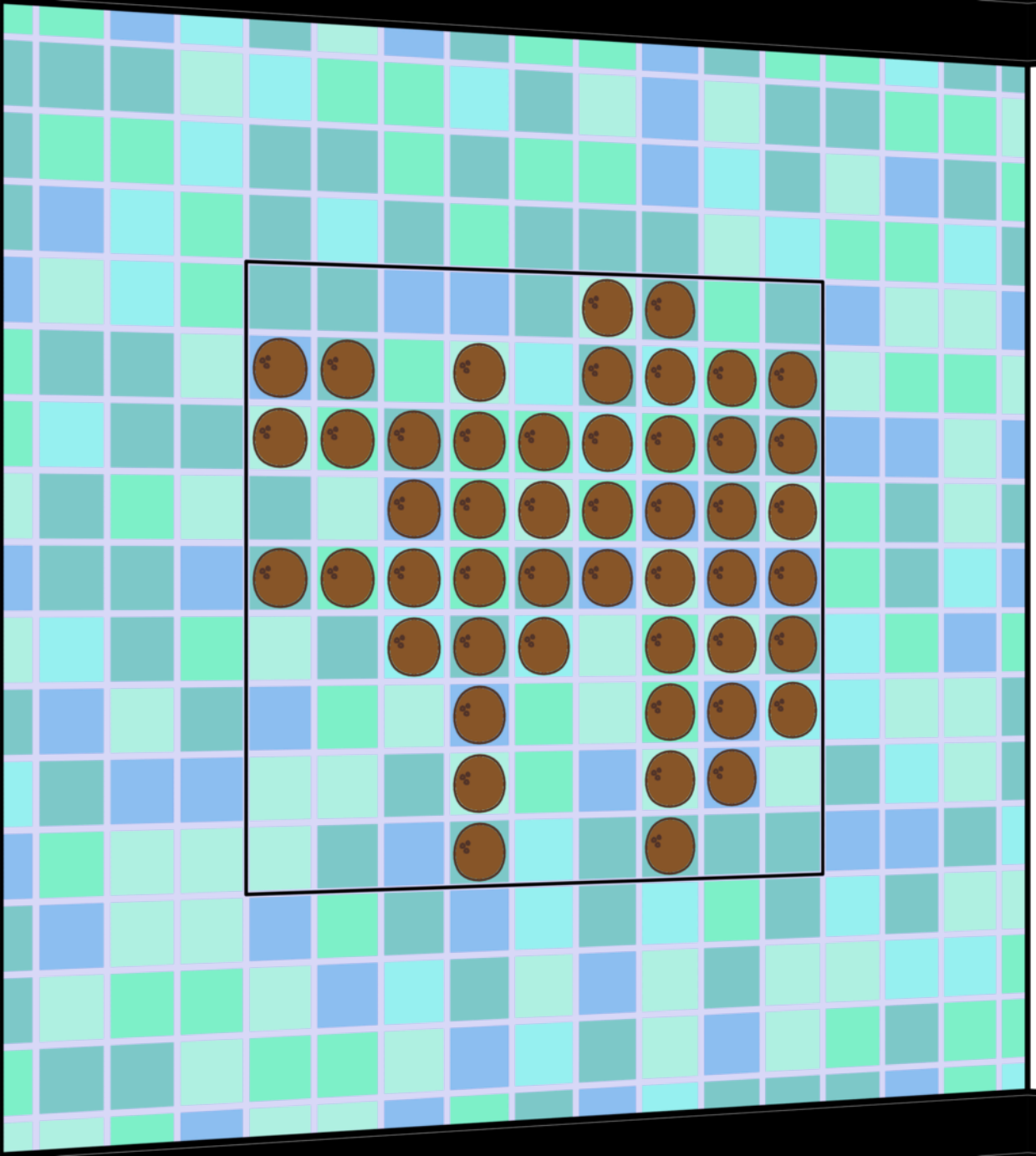
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.



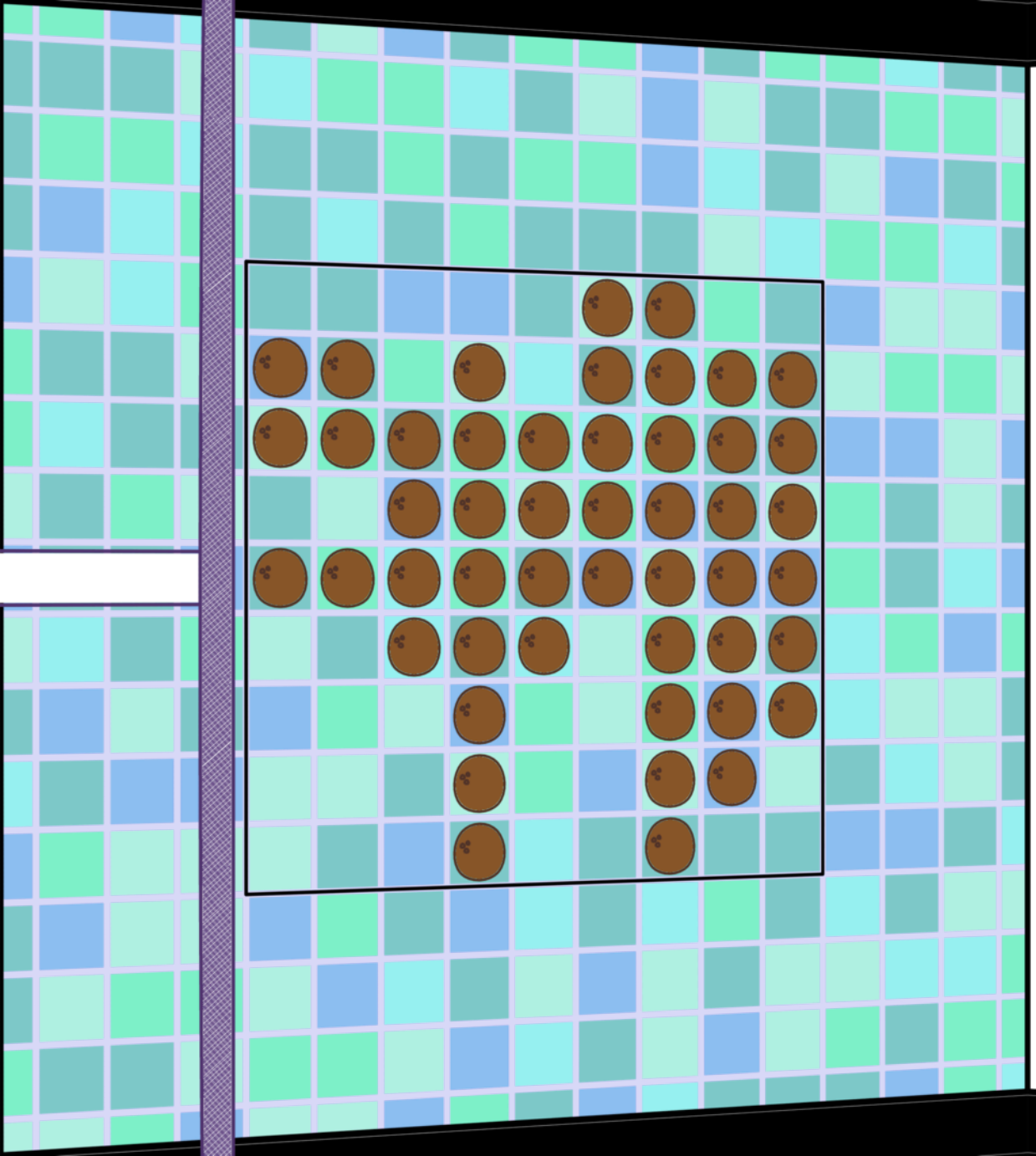
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.



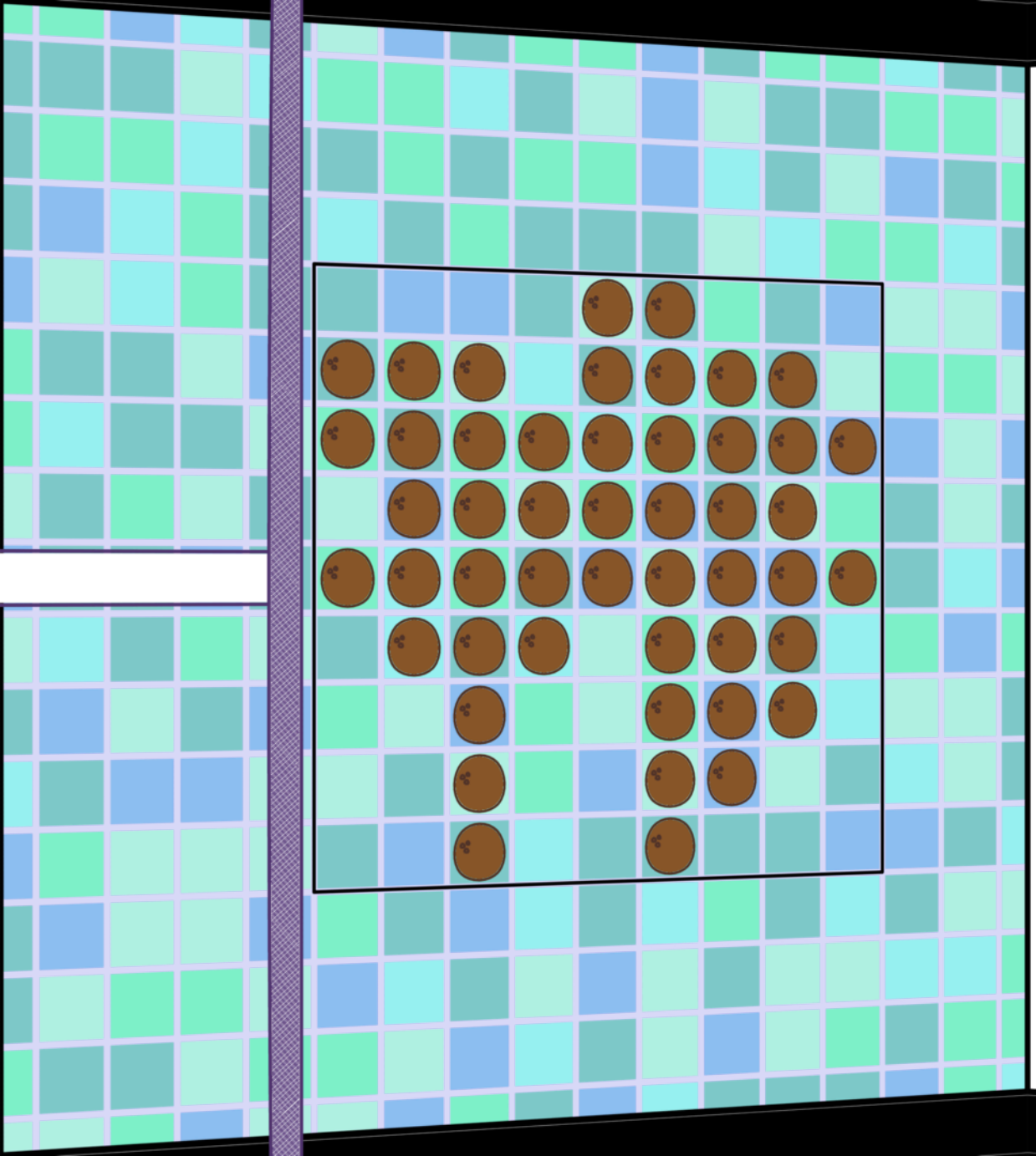
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.



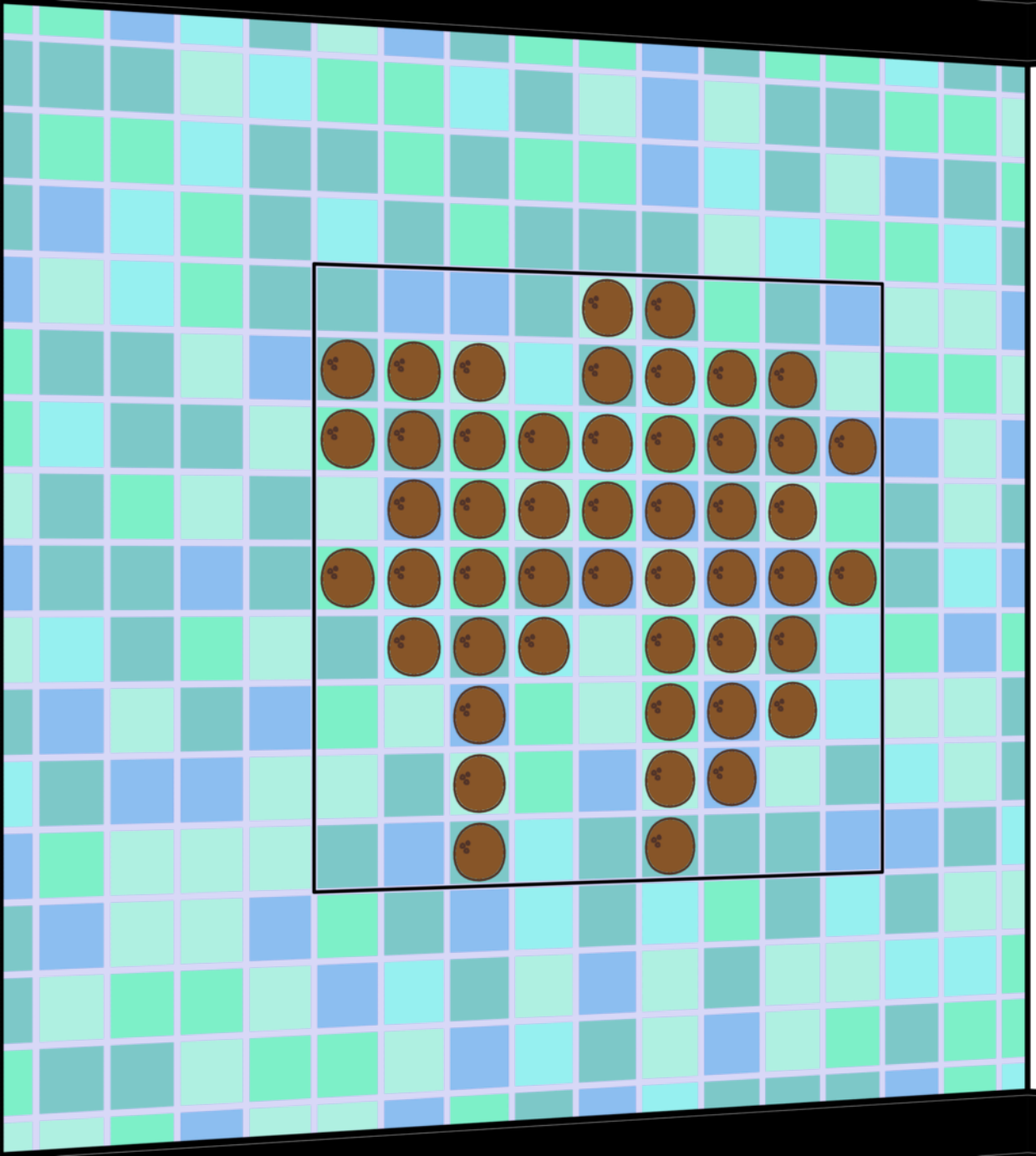
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.



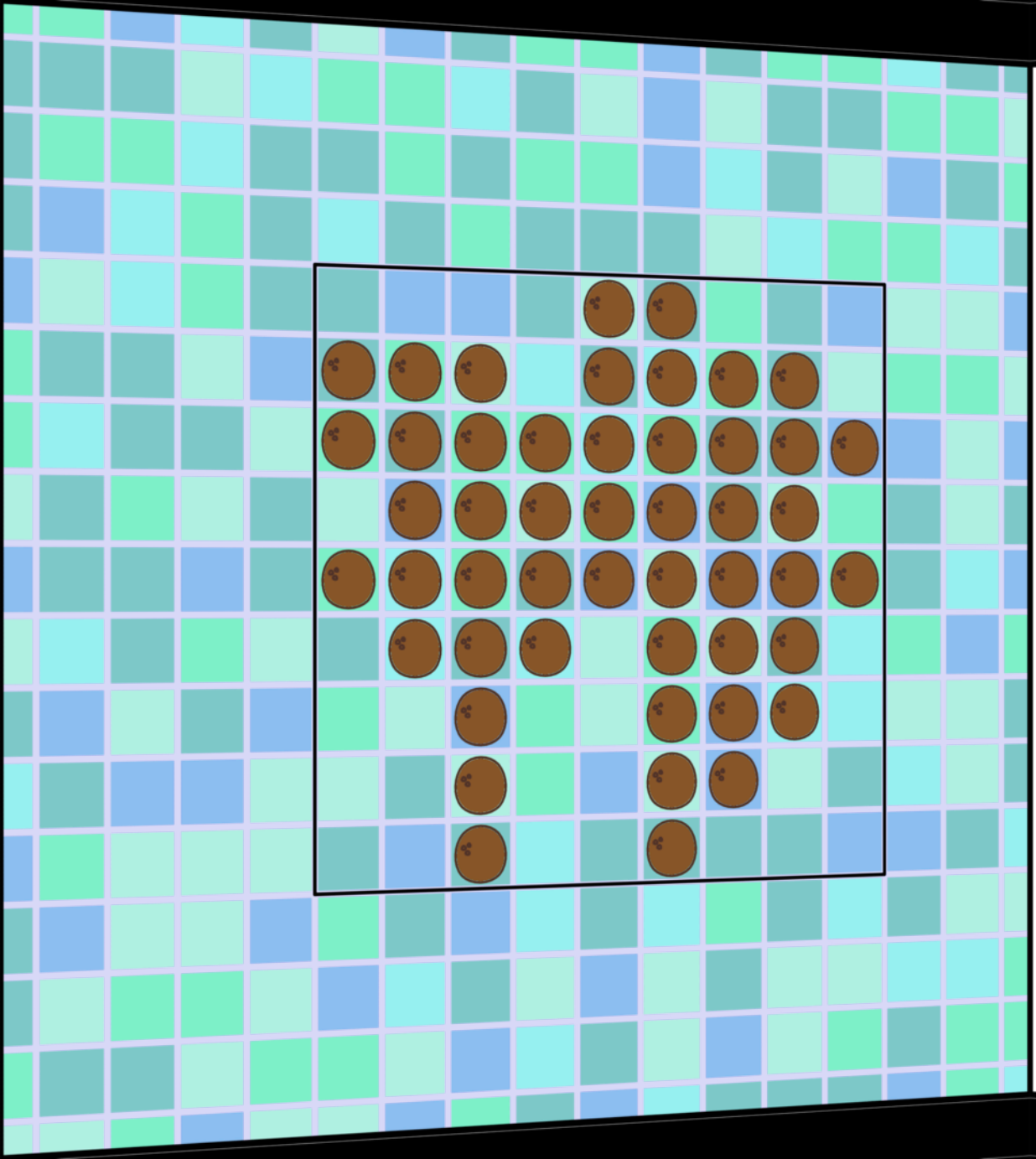
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

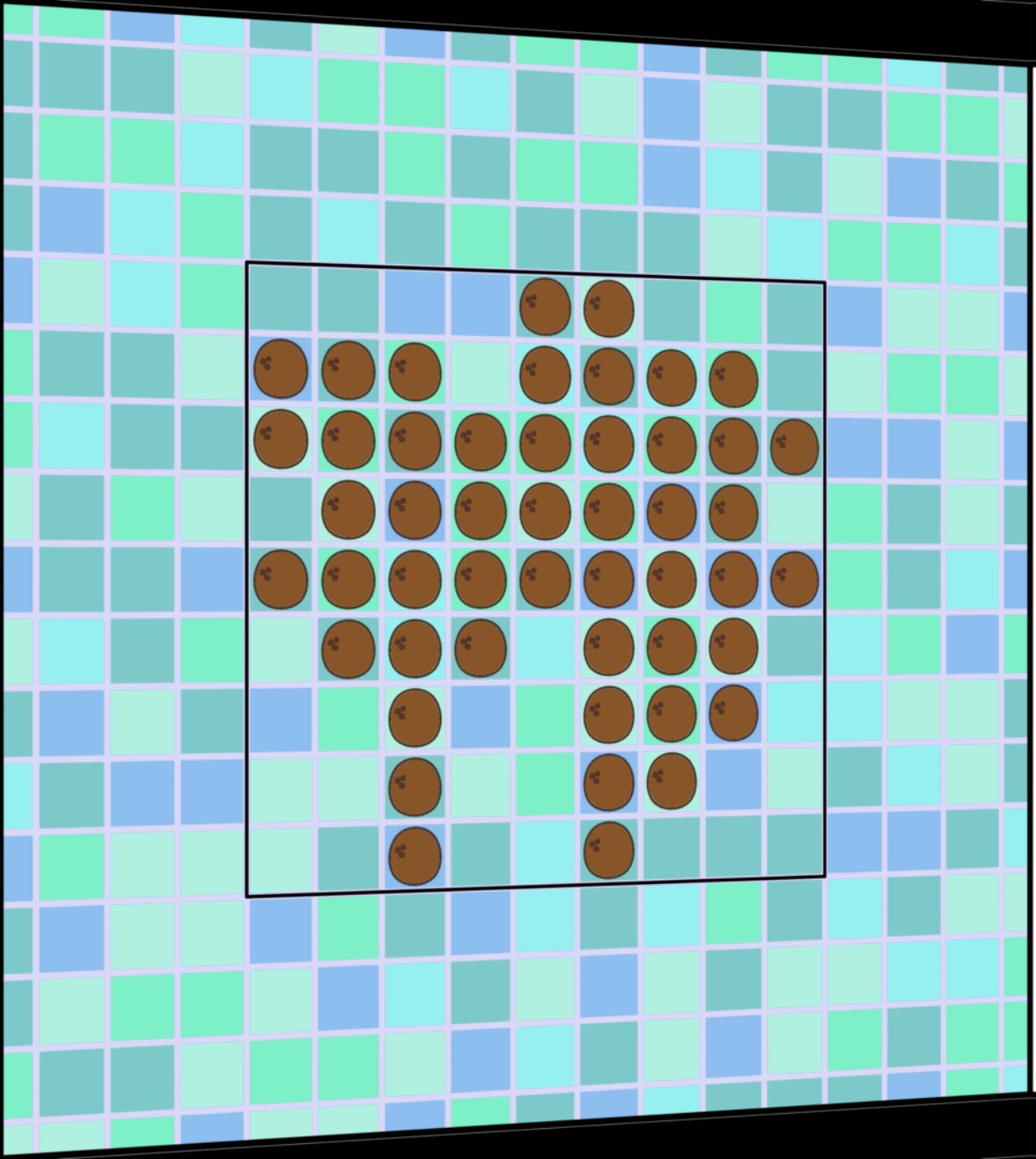


When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.



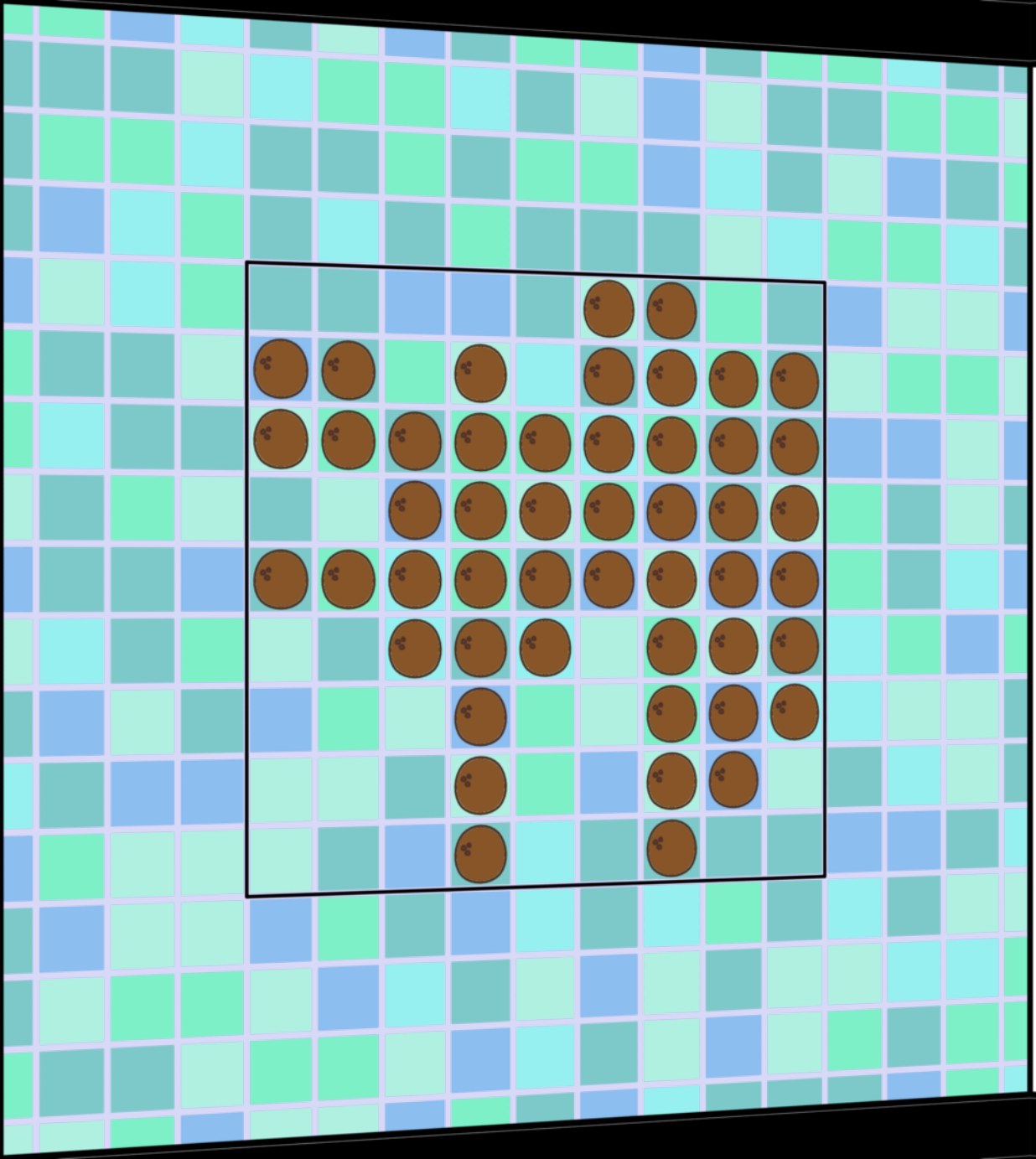
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.



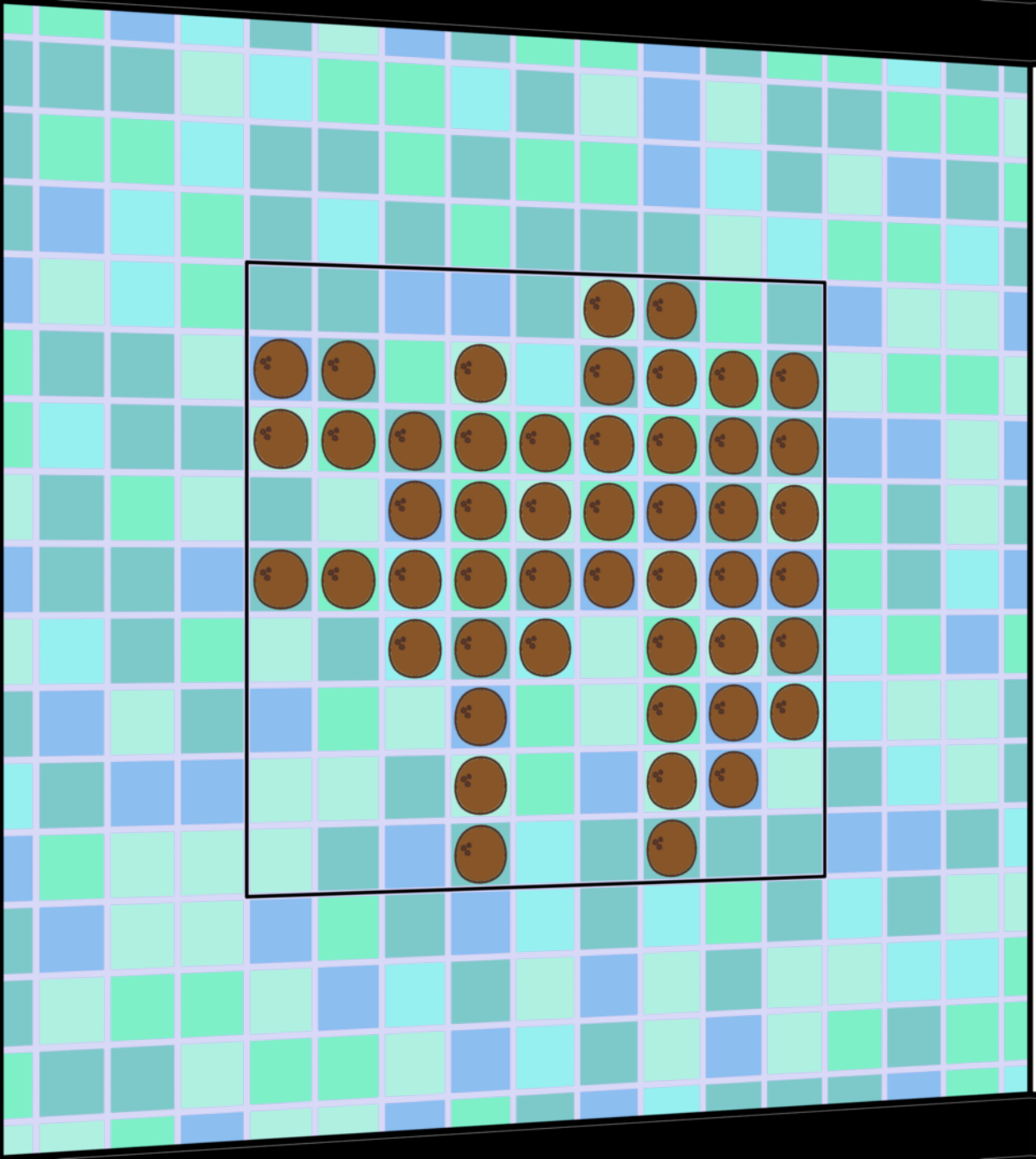
When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

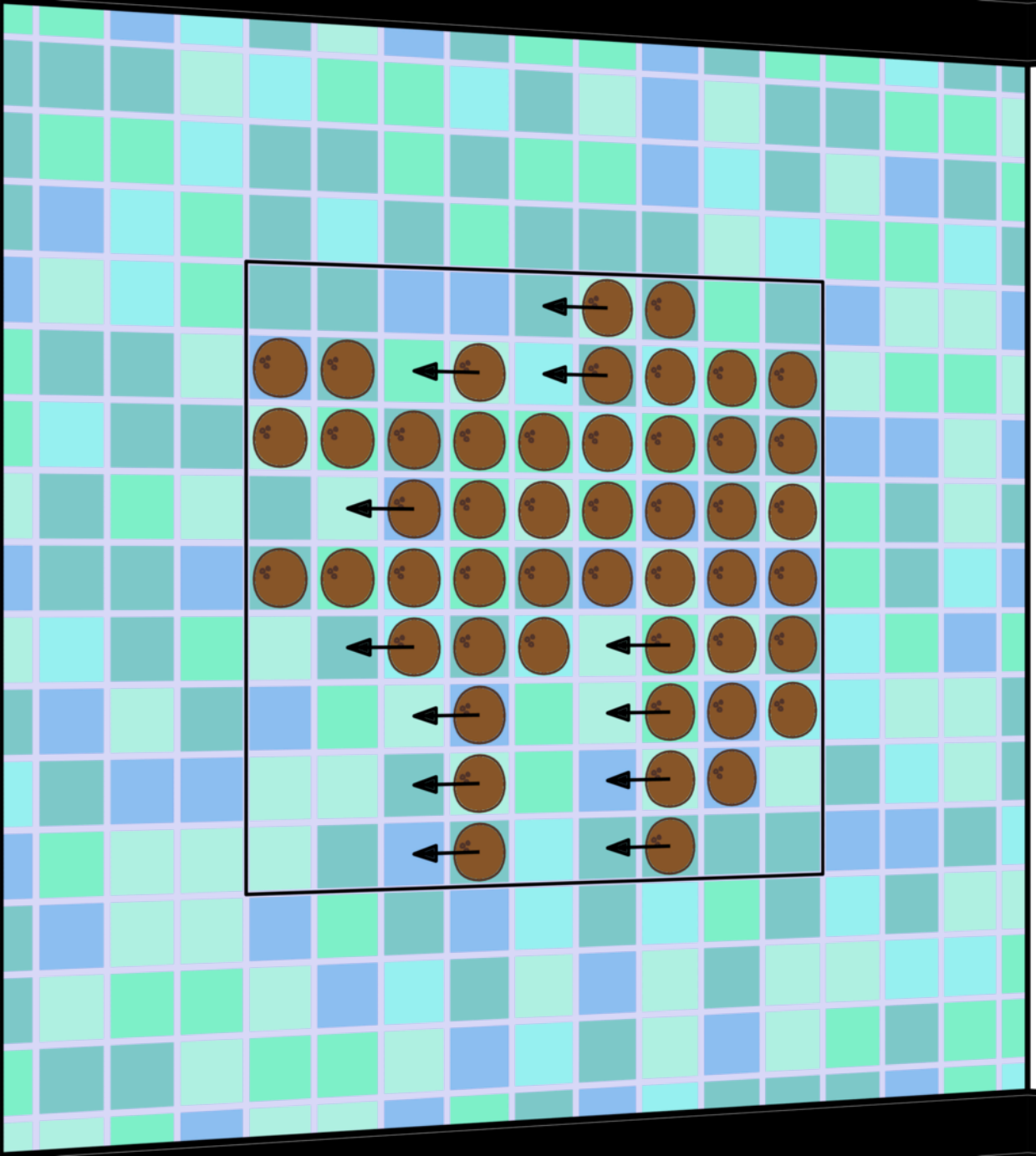
Now, suppose we keep the view centered on the box.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

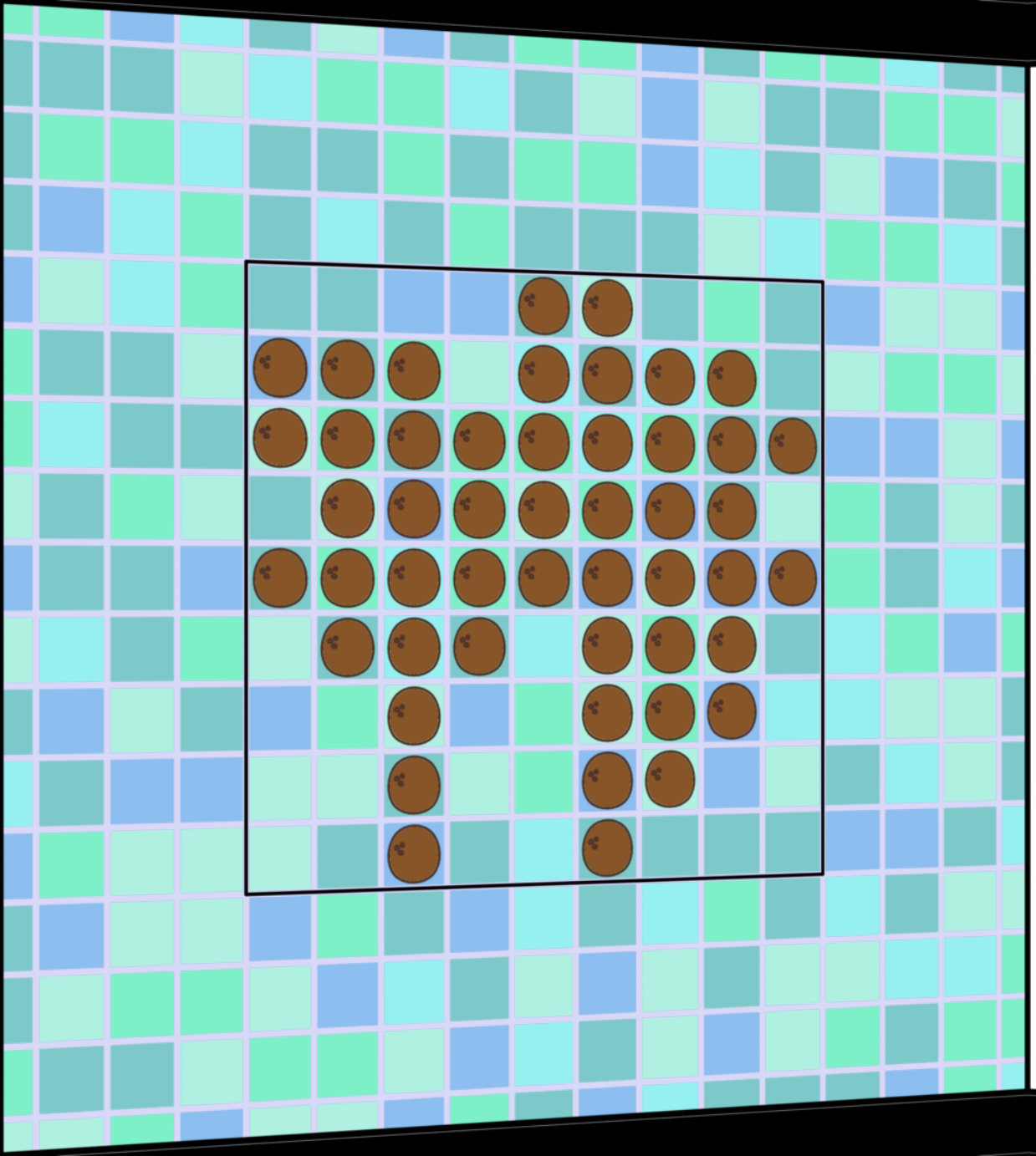
By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

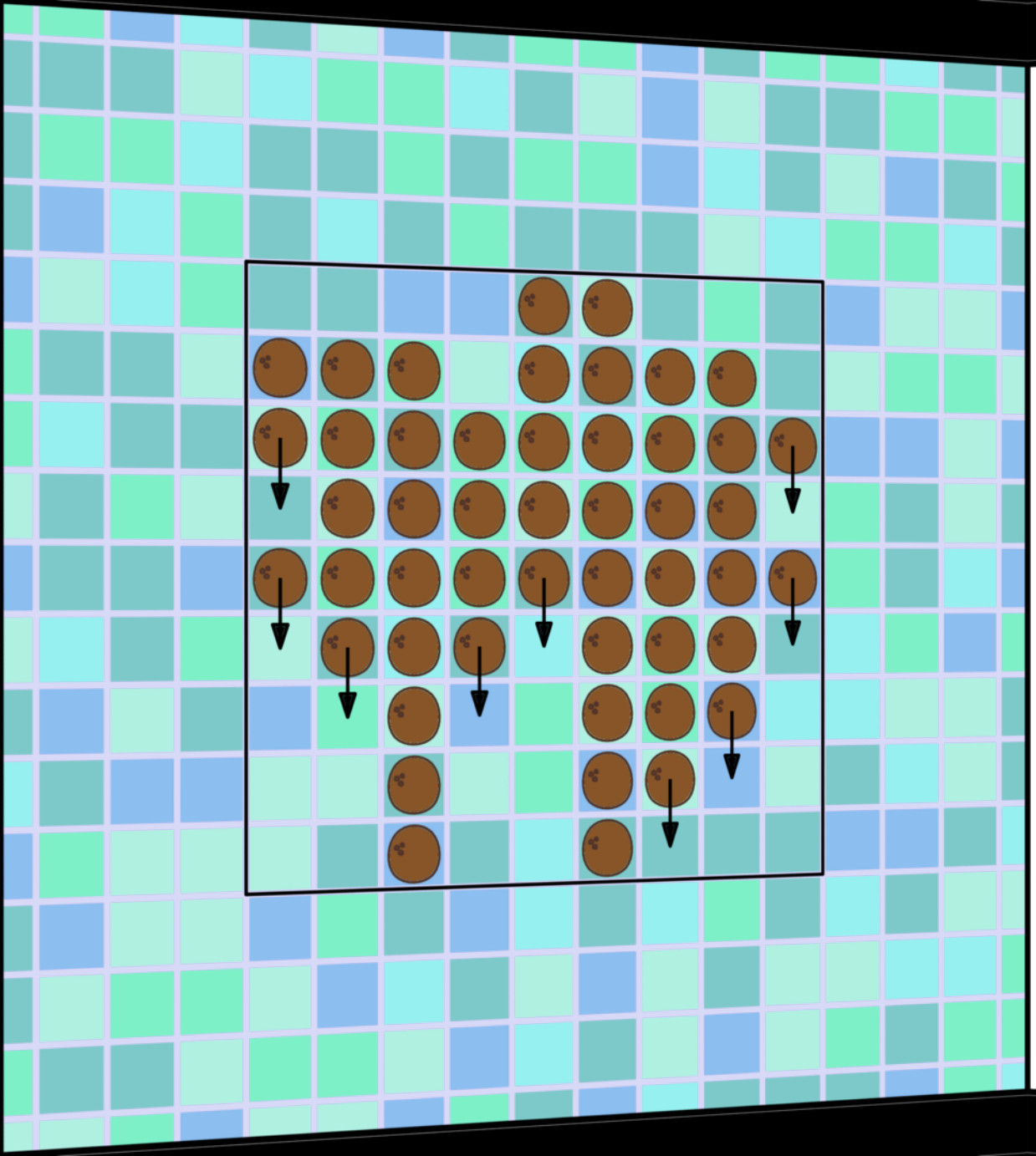
By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

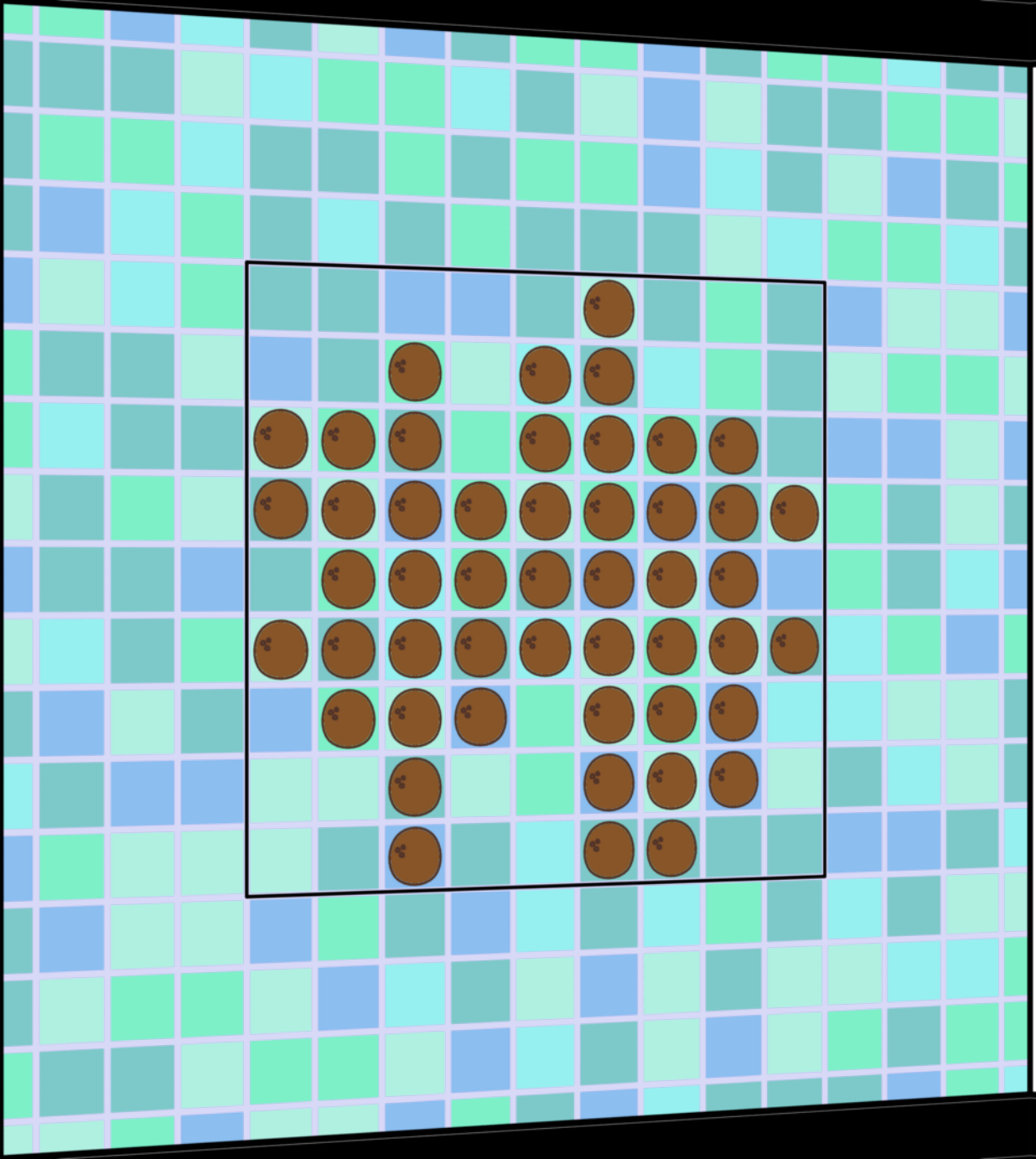
By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

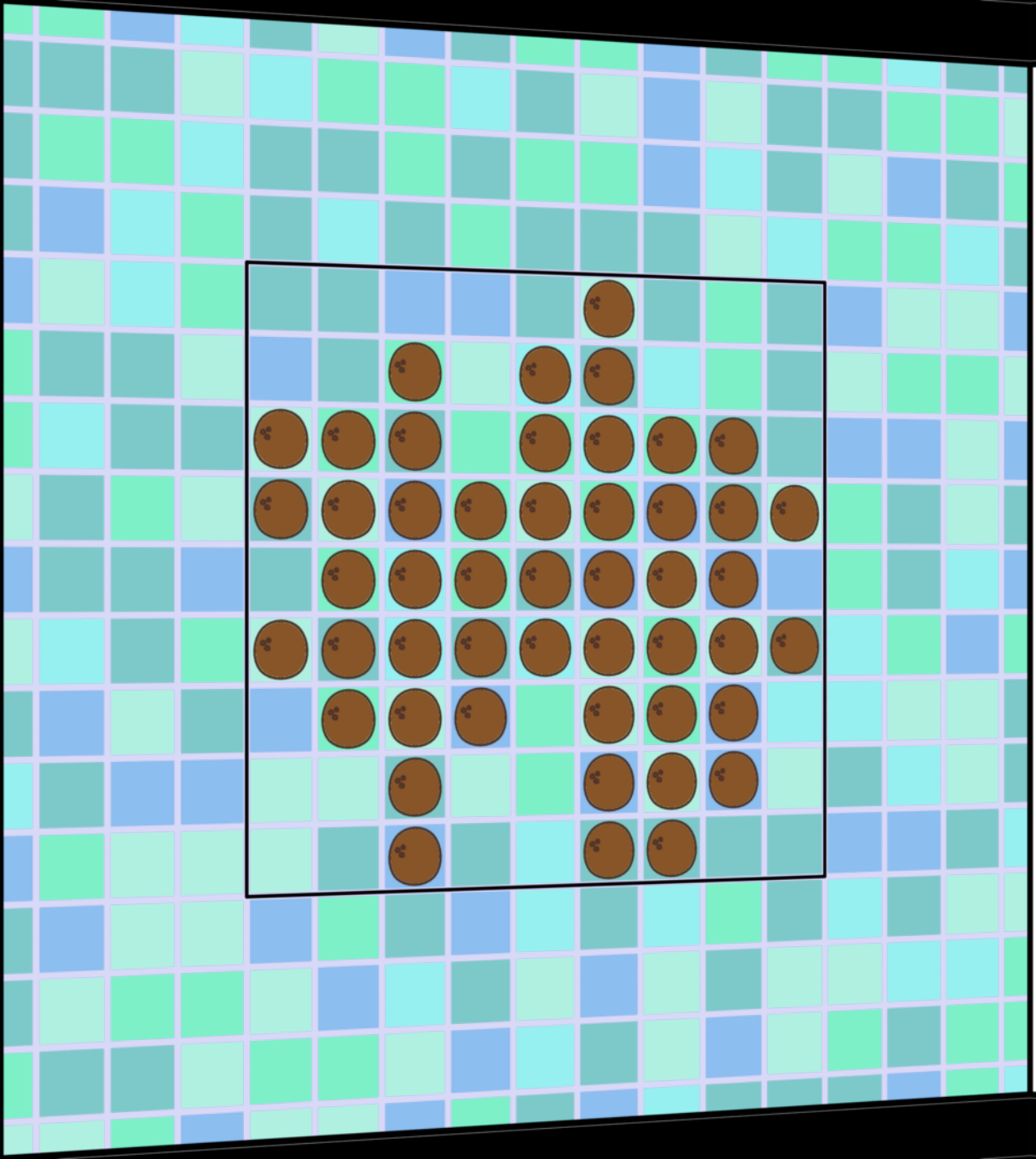
By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.



When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

Now, suppose we keep the view centered on the box.

By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.

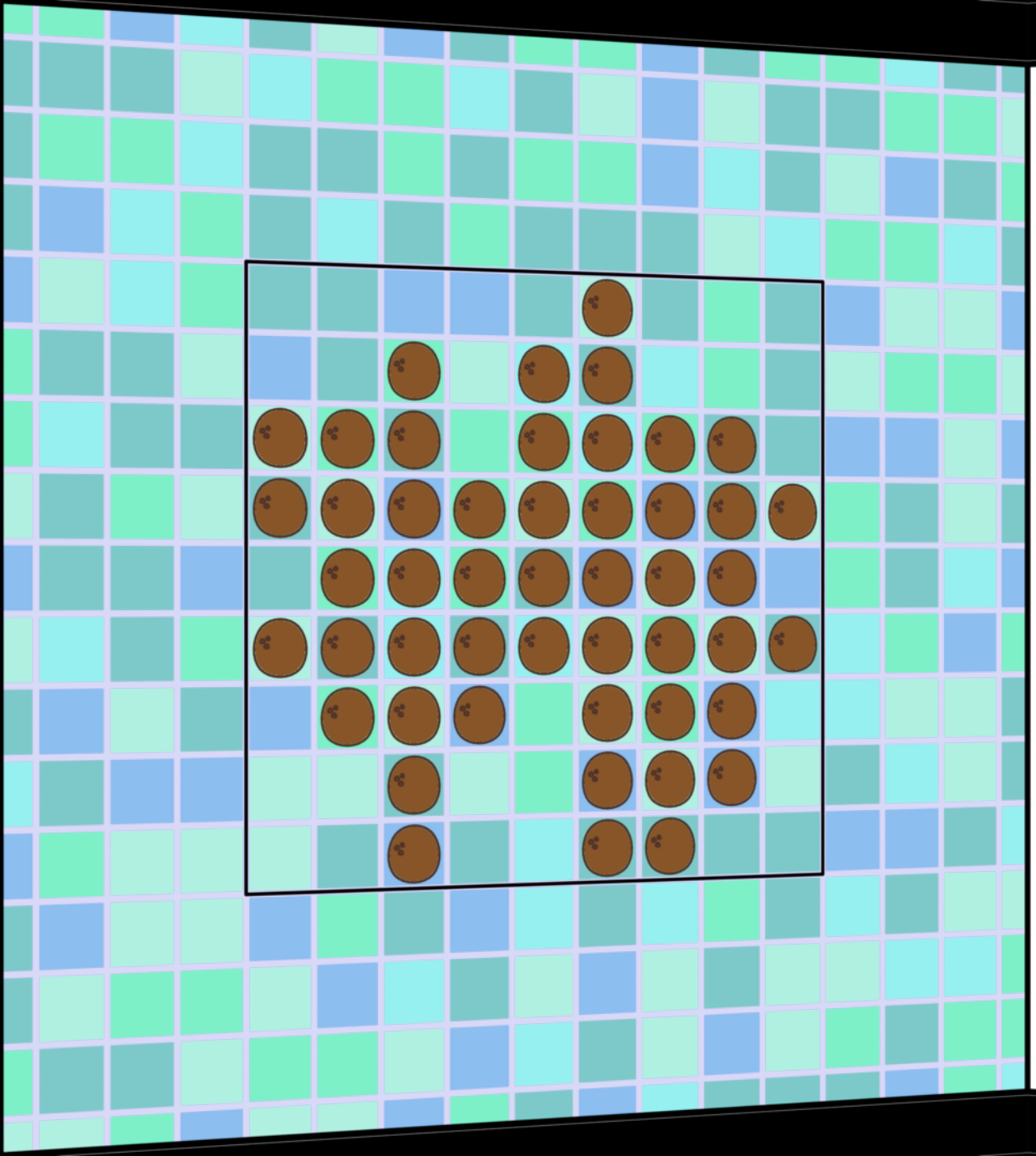


When at least one column and one row are *full*, the size of the bounding box doesn't change anymore.

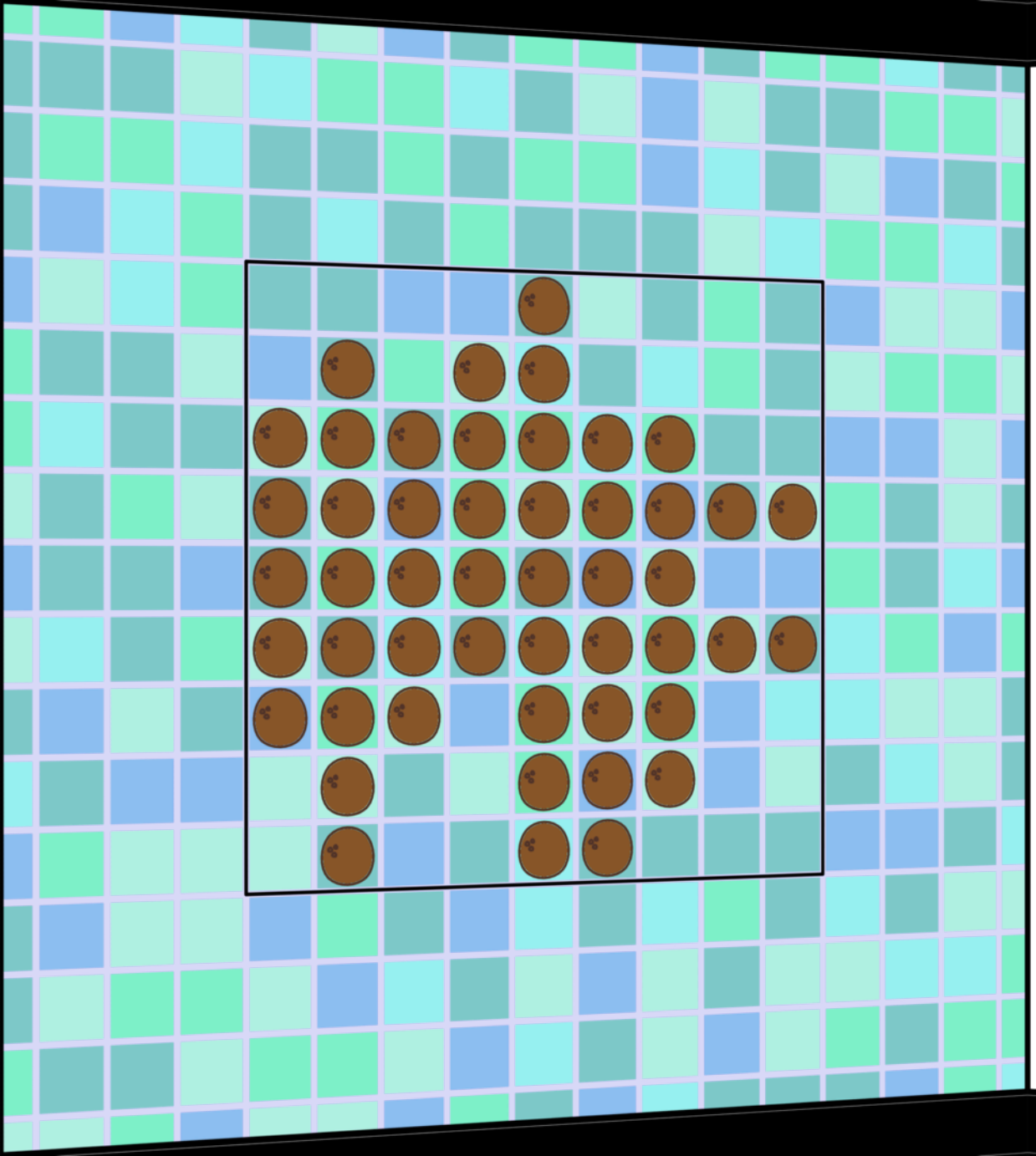
Now, suppose we keep the view centered on the box.

By pushing *from* the left, all coconuts move one step *to* the left, unless blocked.

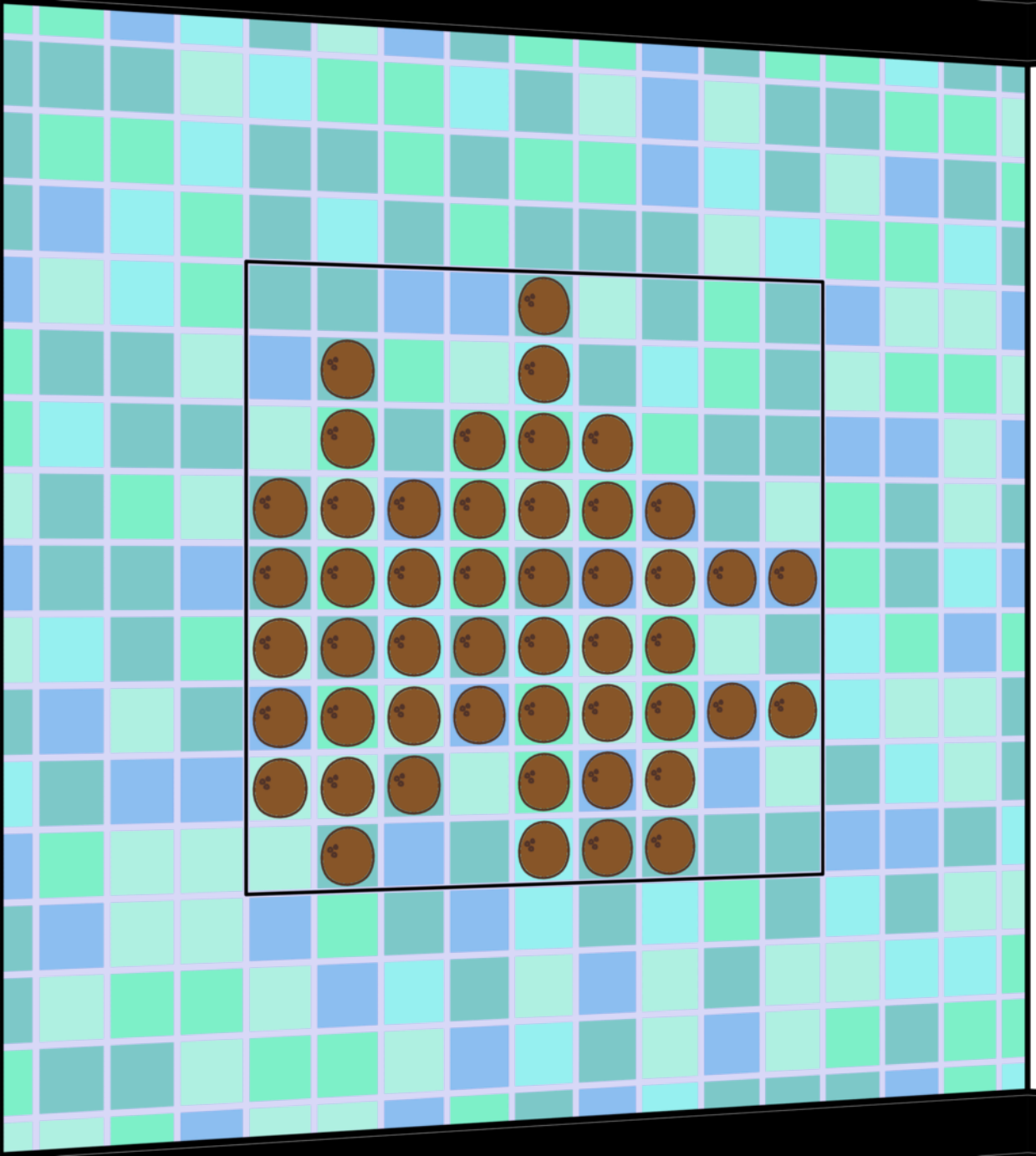
Same game, different viewpoint.



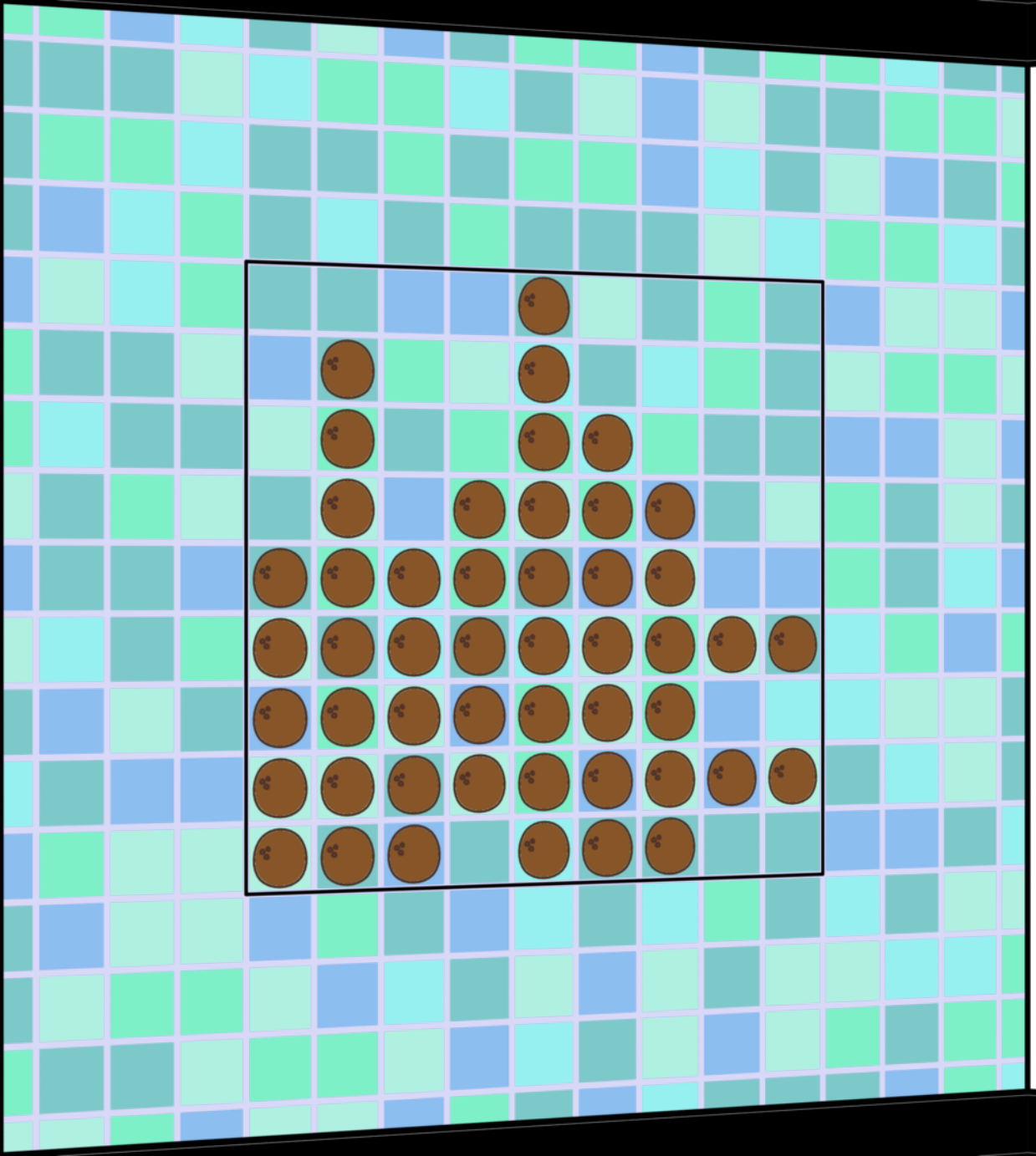
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



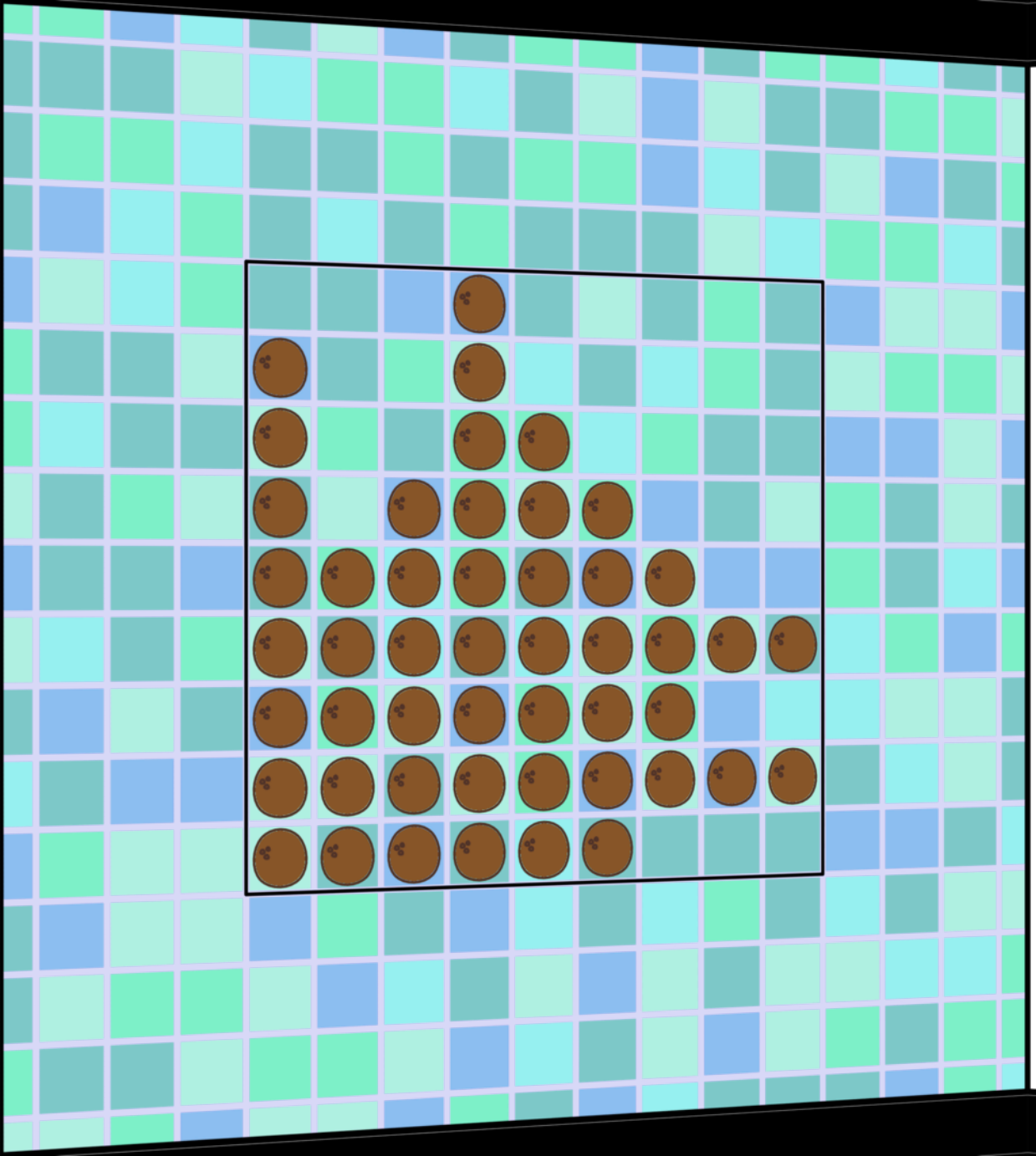
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



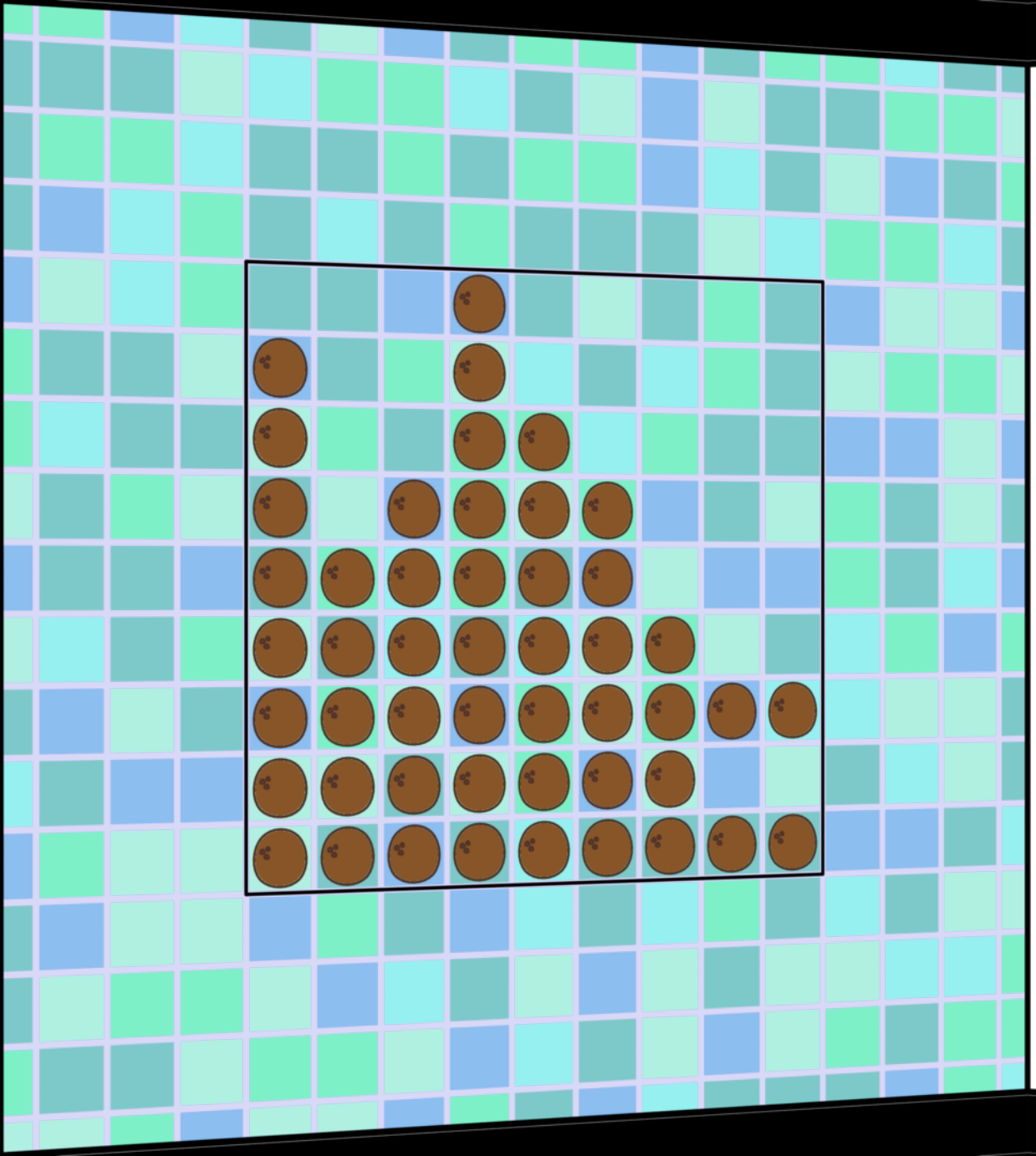
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



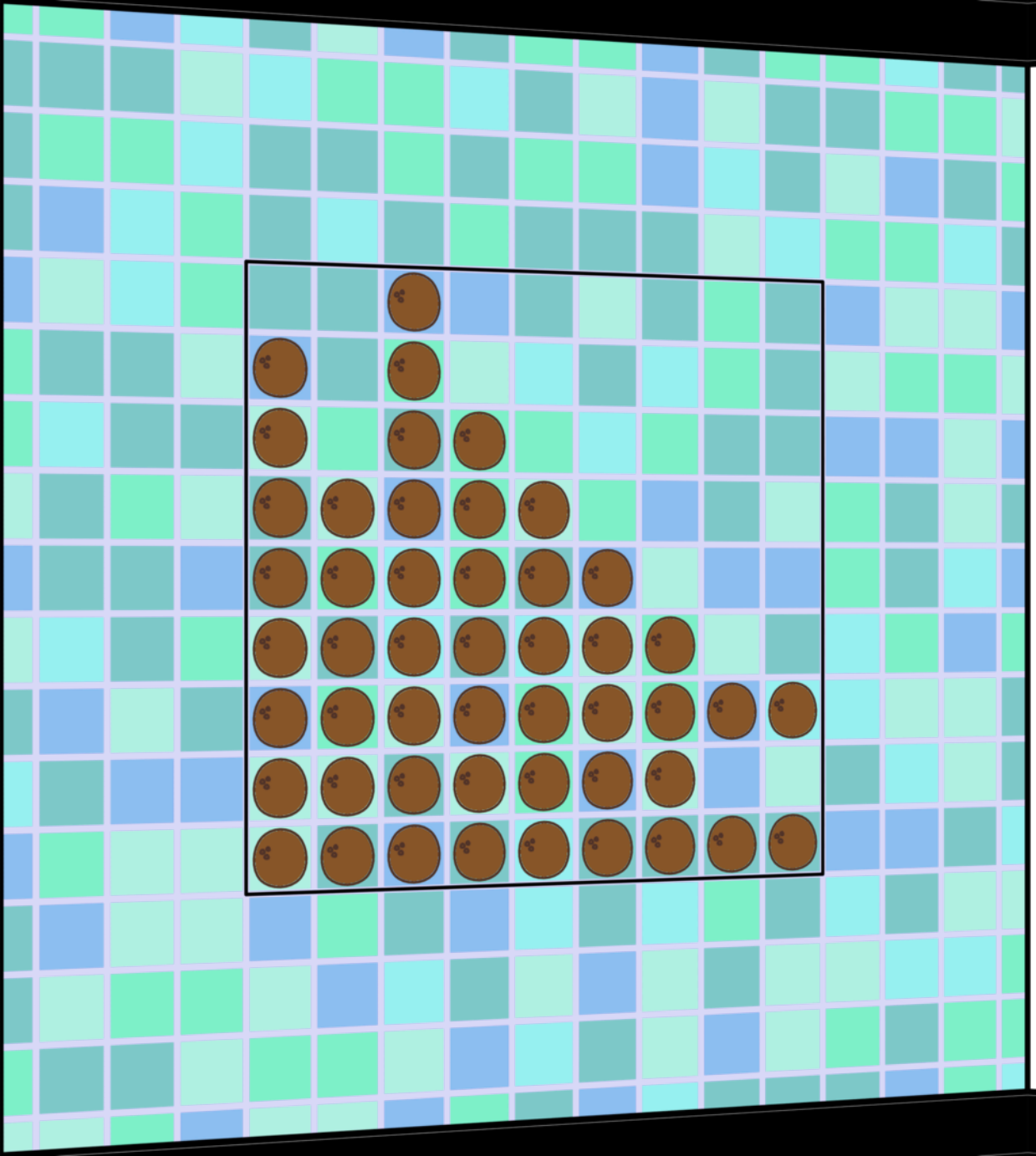
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



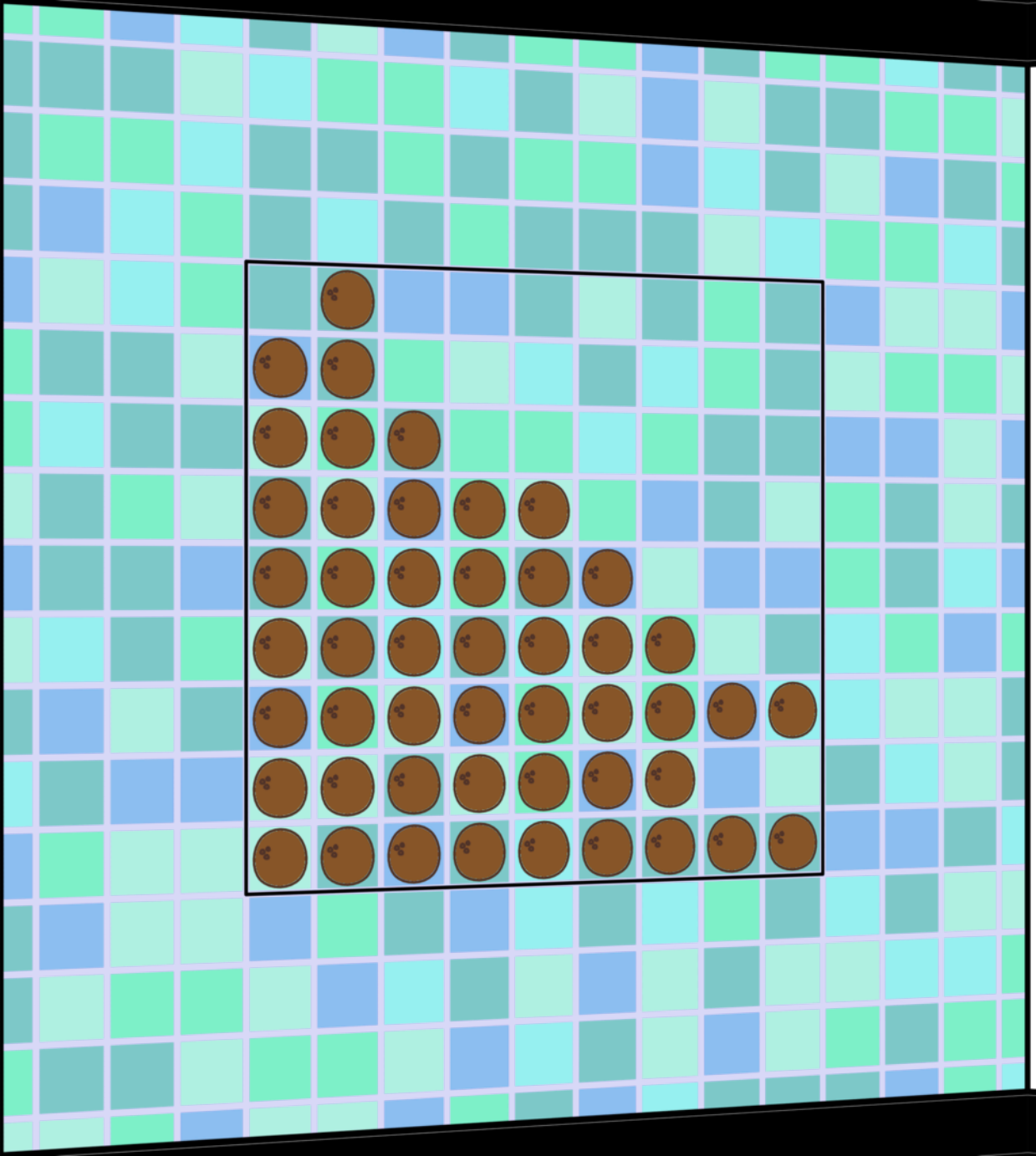
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



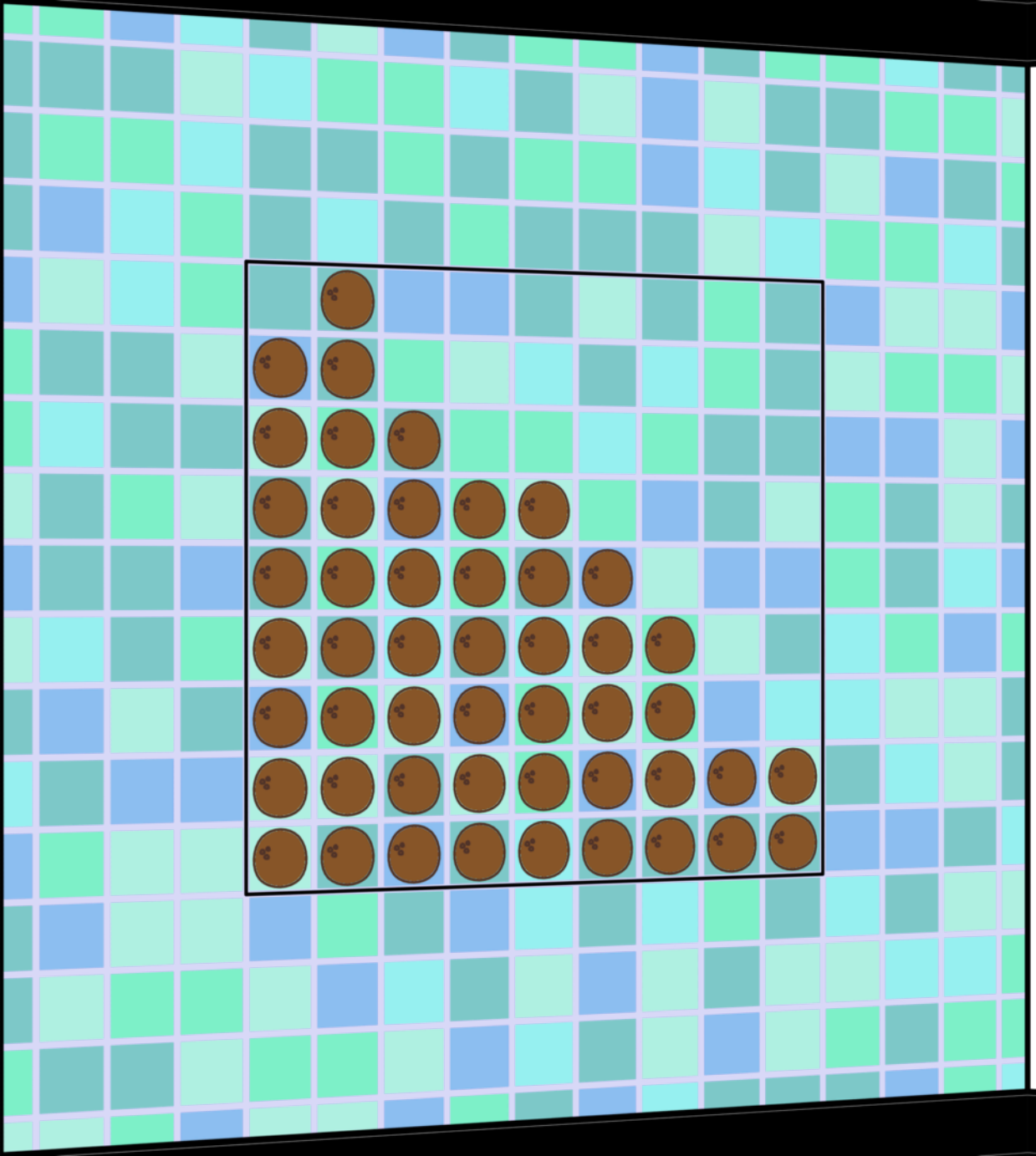
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



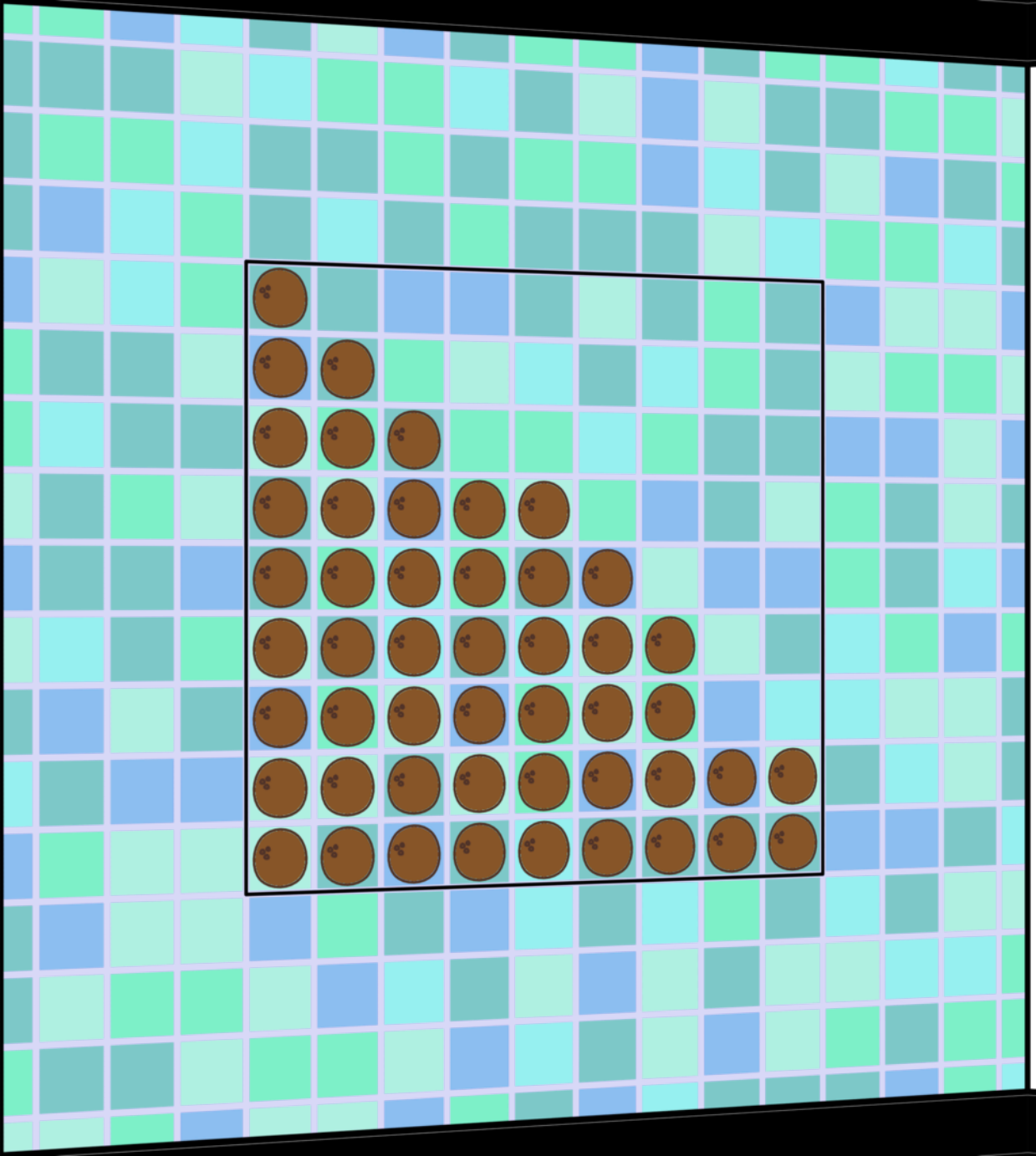
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



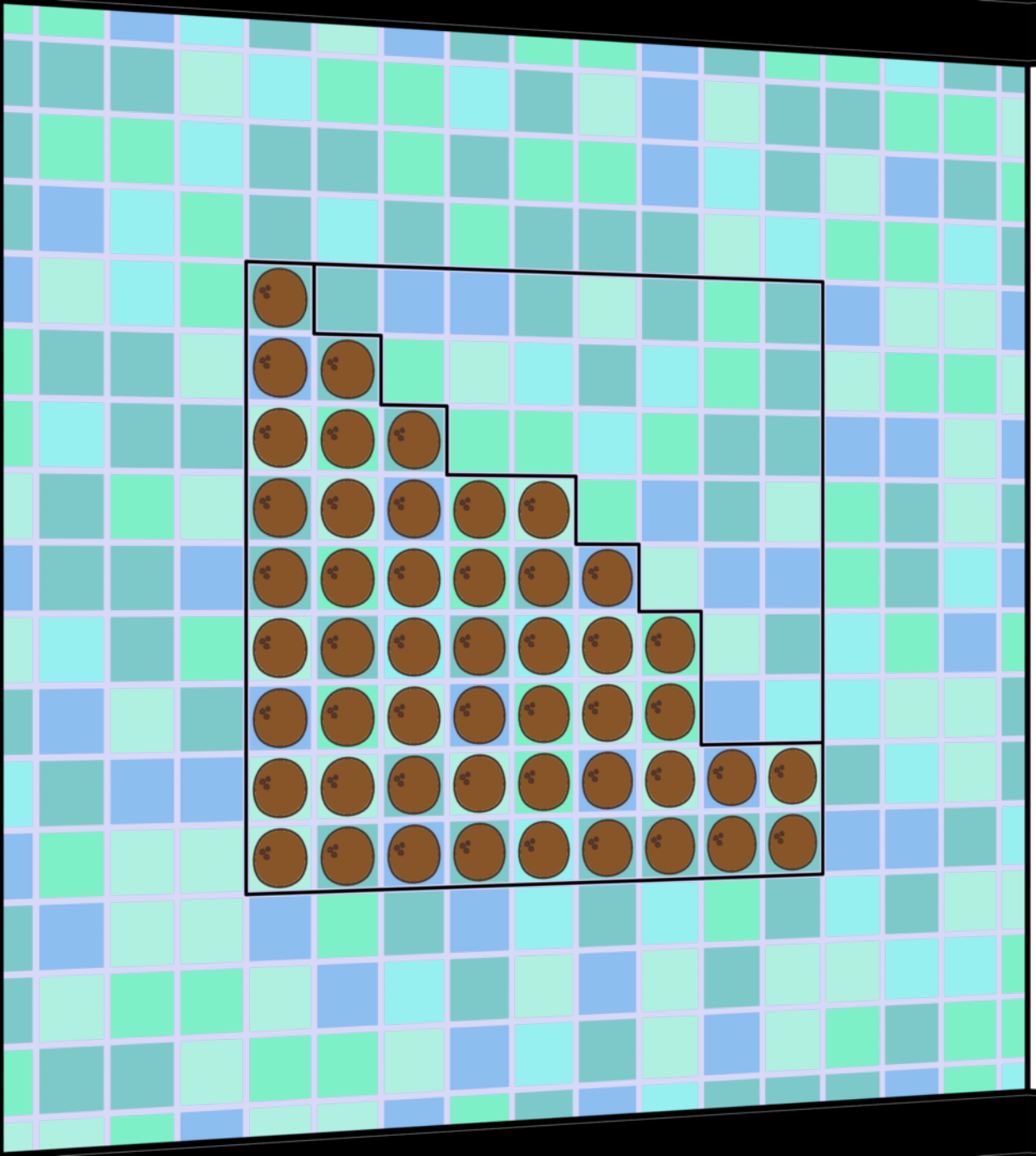
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



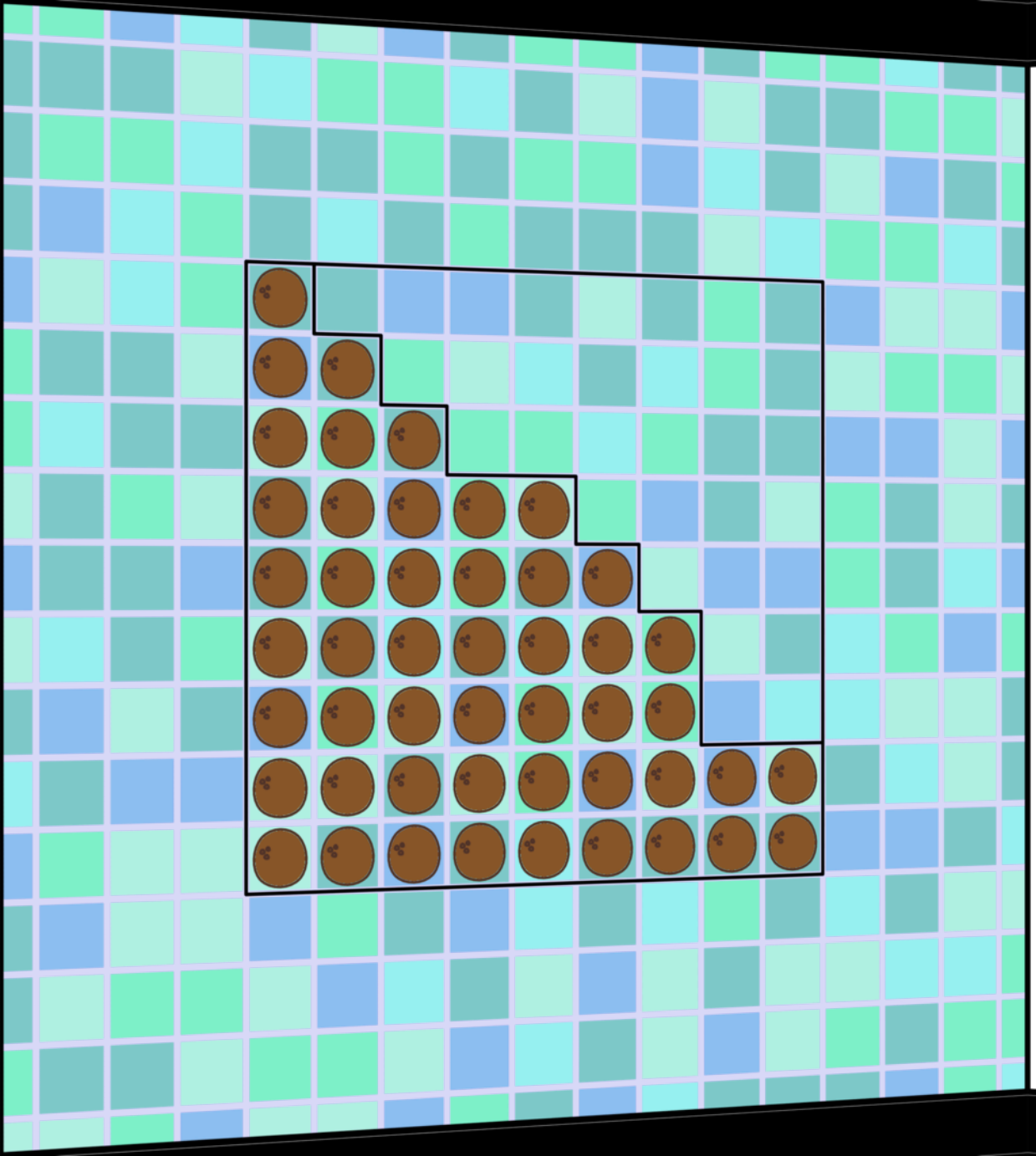
After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.



After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.

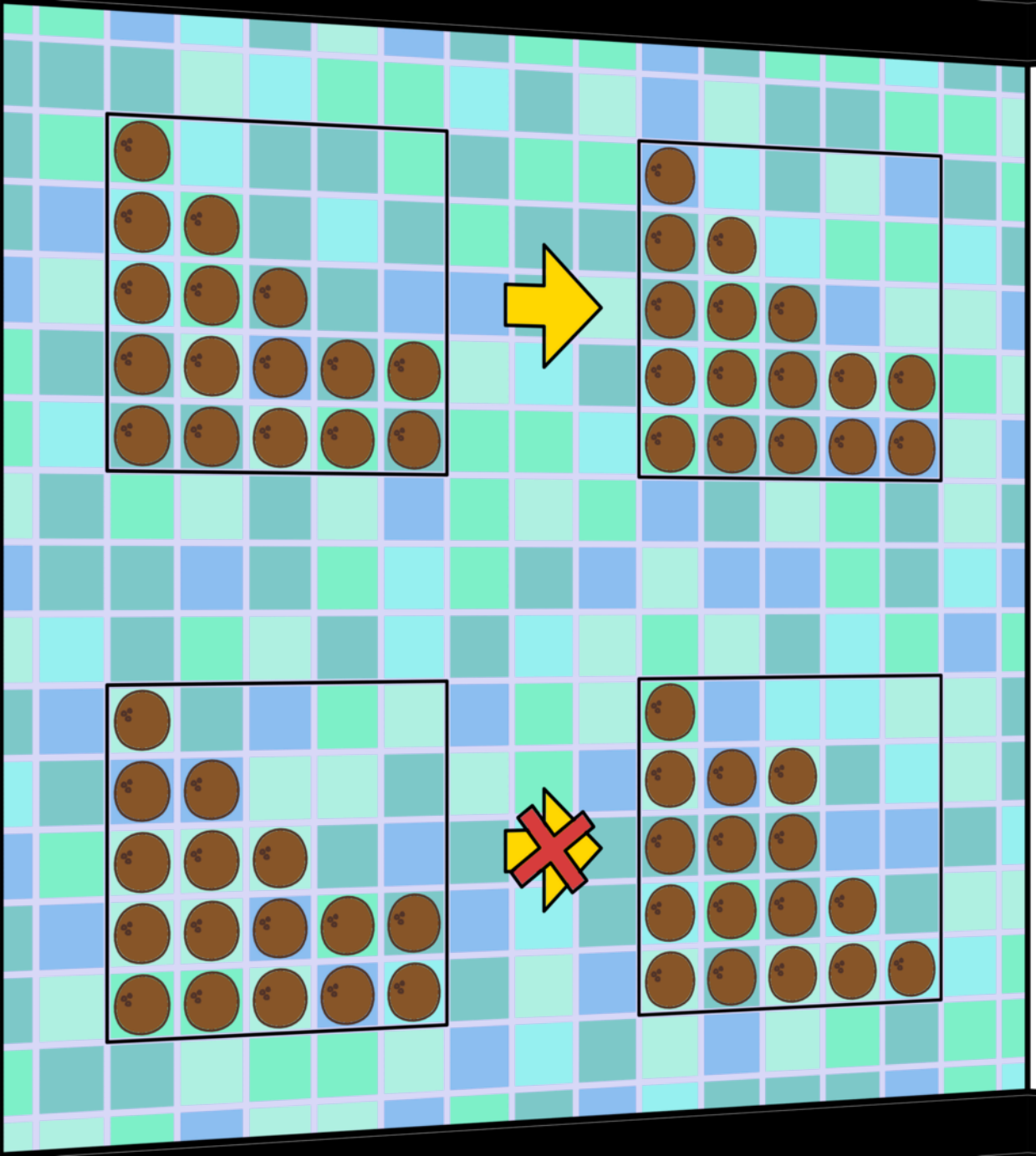


After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.

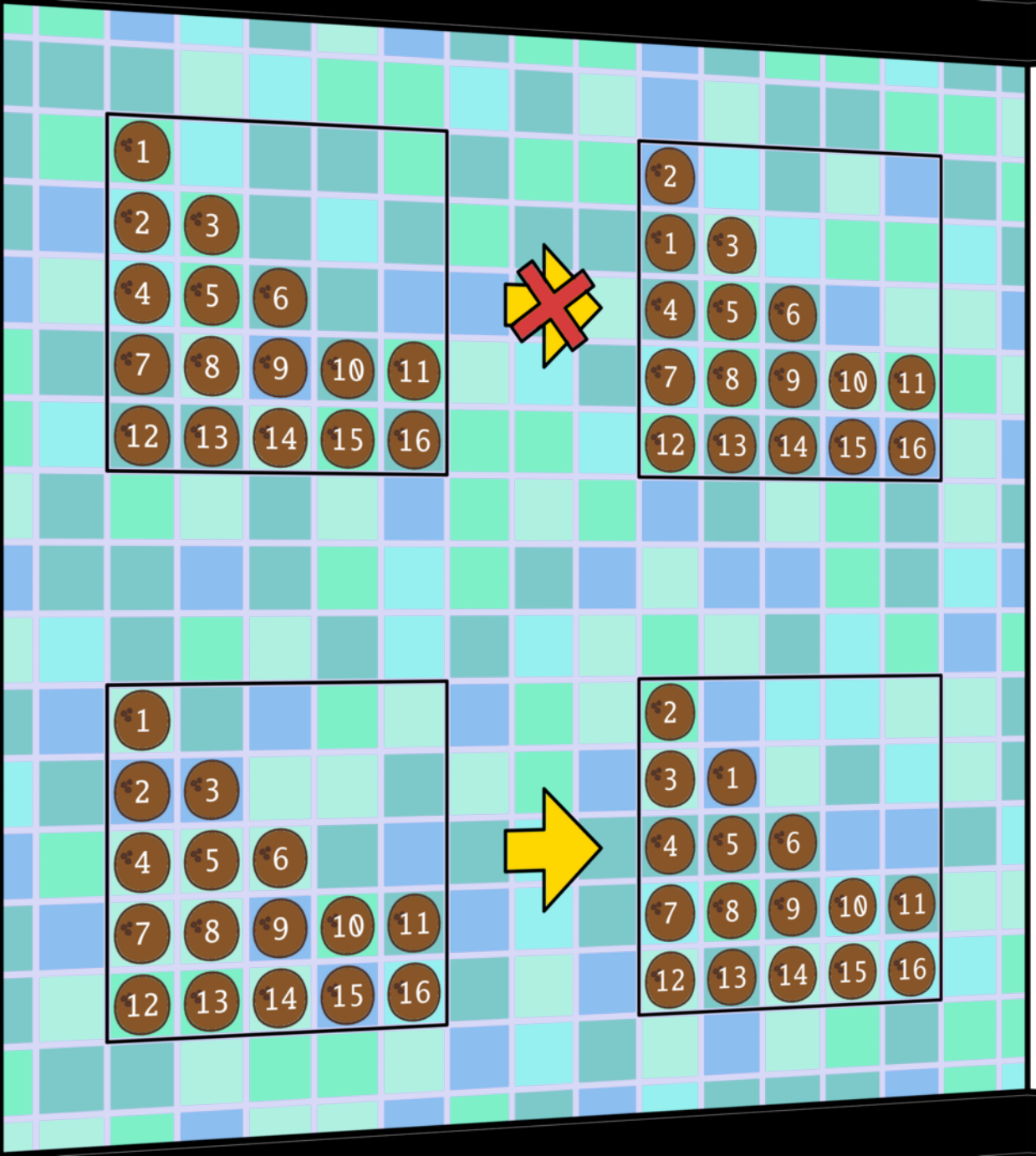


After pushing from the left and from below sufficiently often, we get a *canonical* (staircase) configuration.

What can we say about reconfiguring canonical configurations?



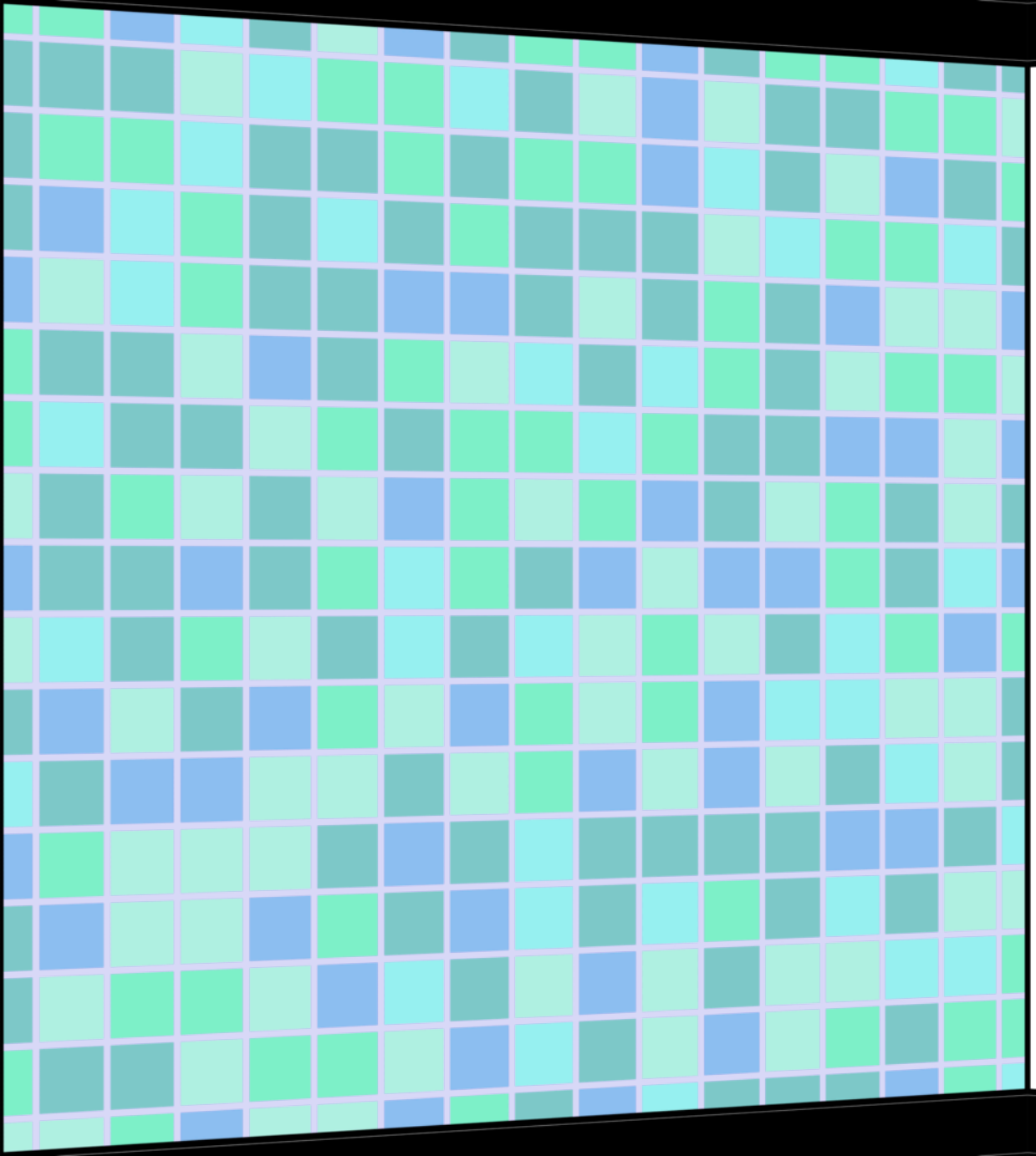
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.



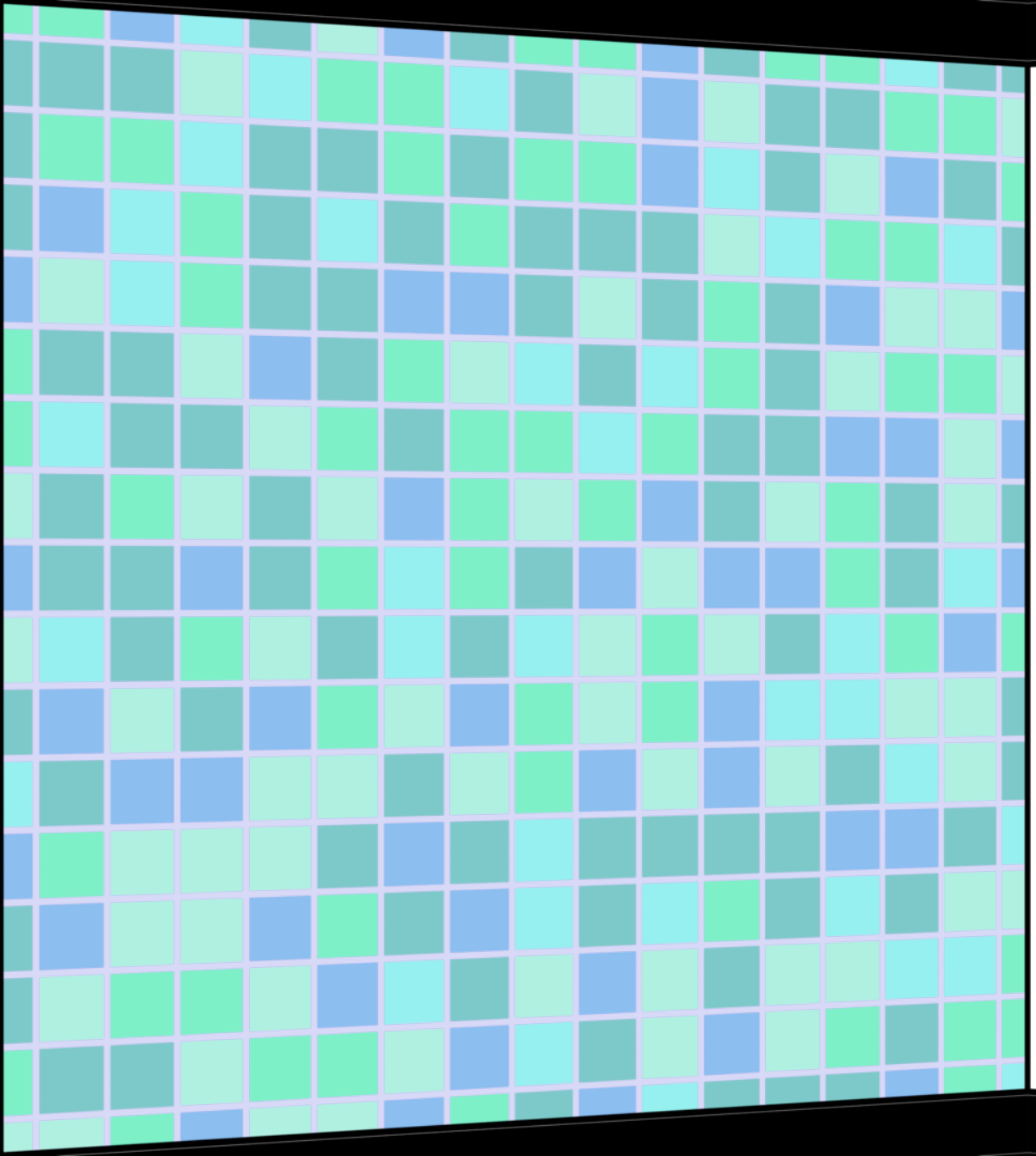
Theorem 2. Every realizable permutation of a canonical configuration is *even*.

Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

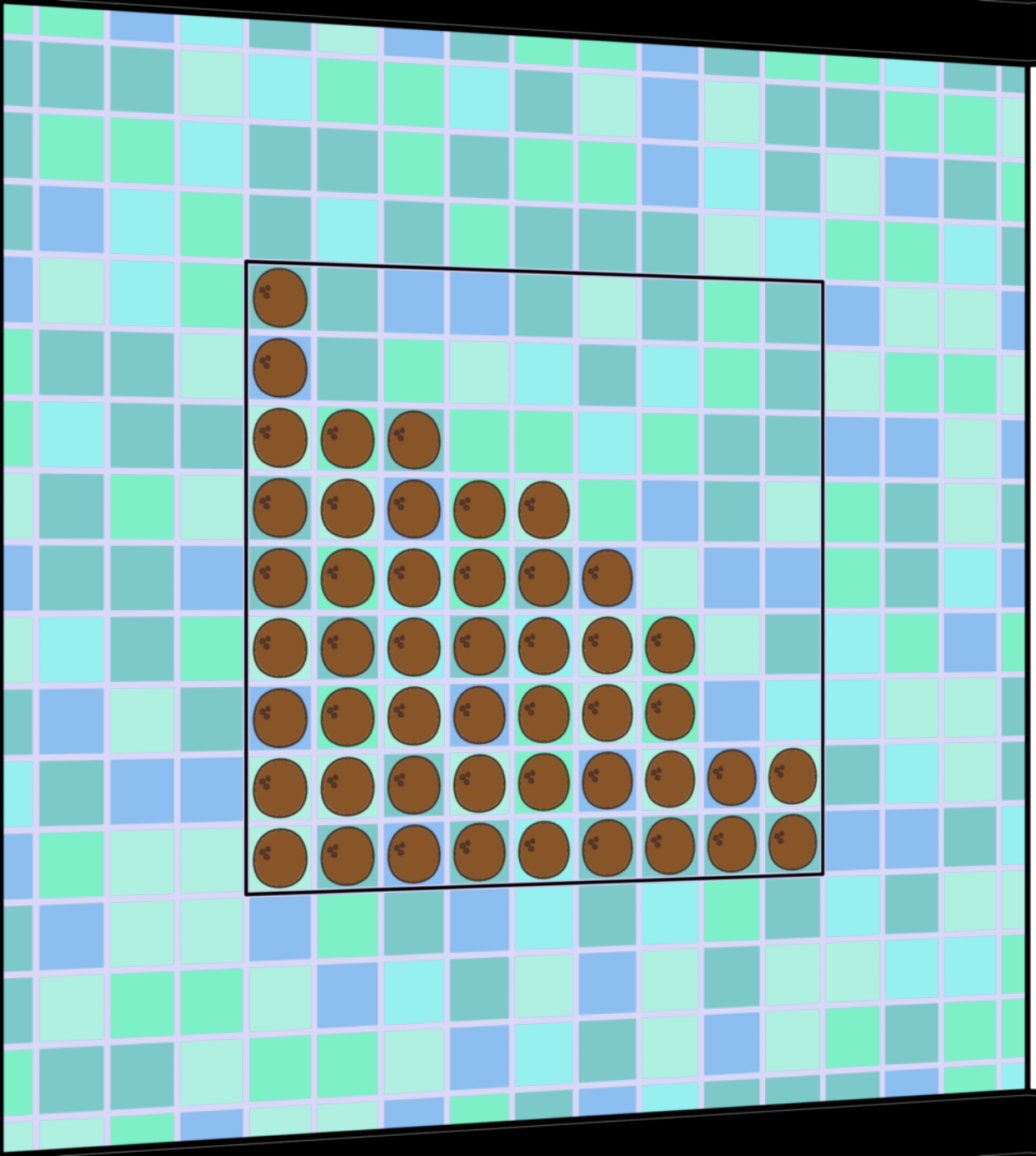


Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.



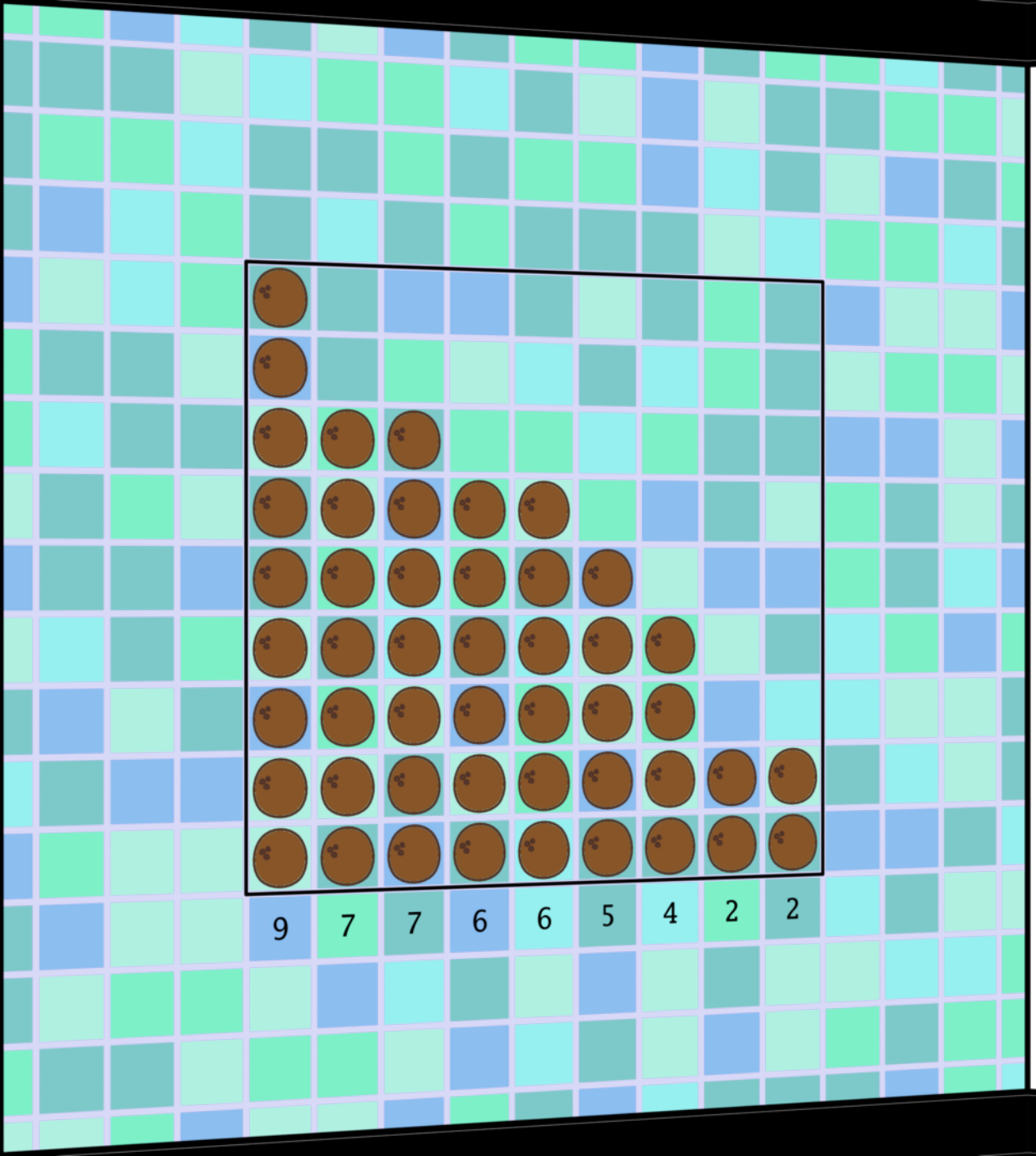
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



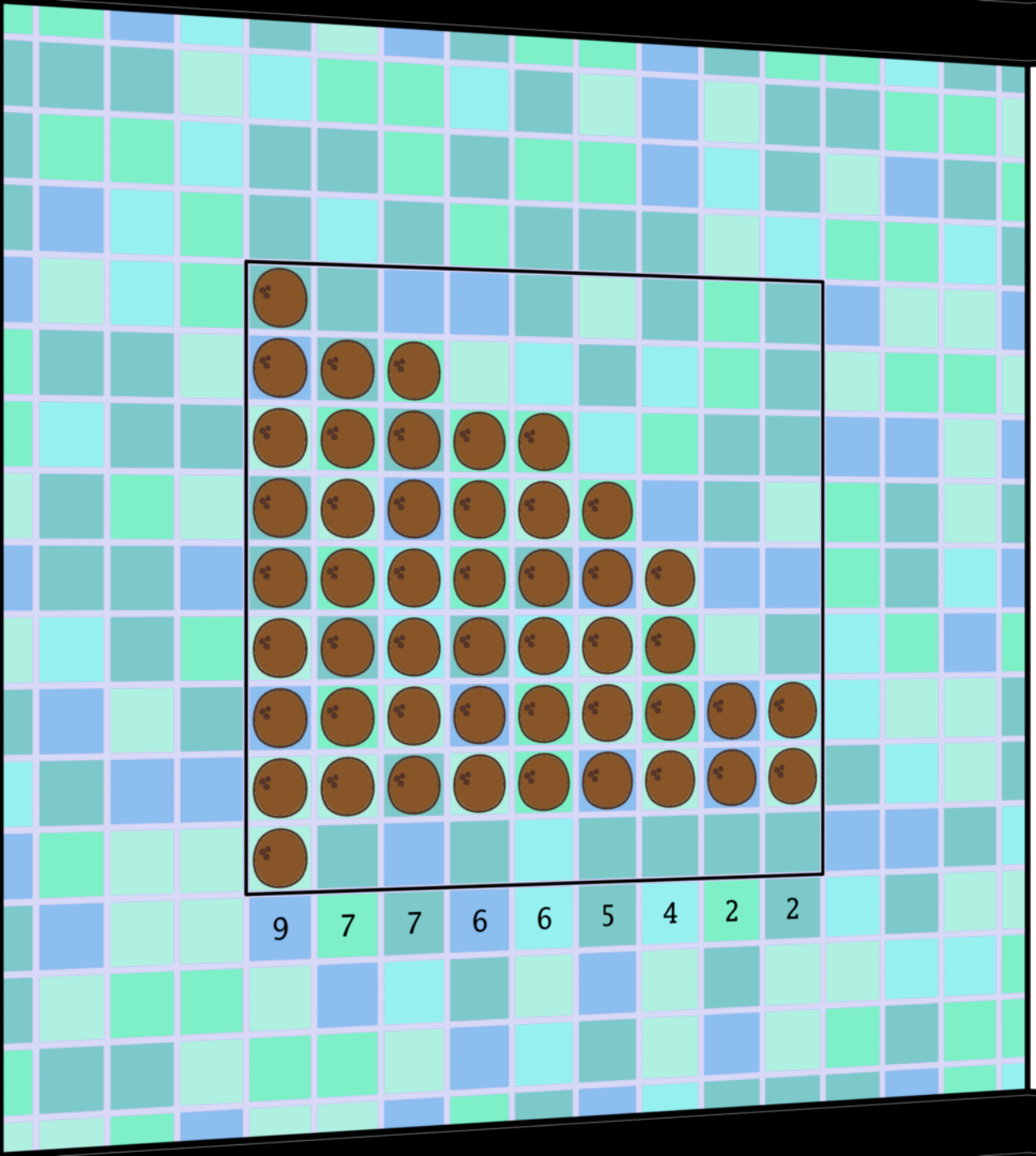
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



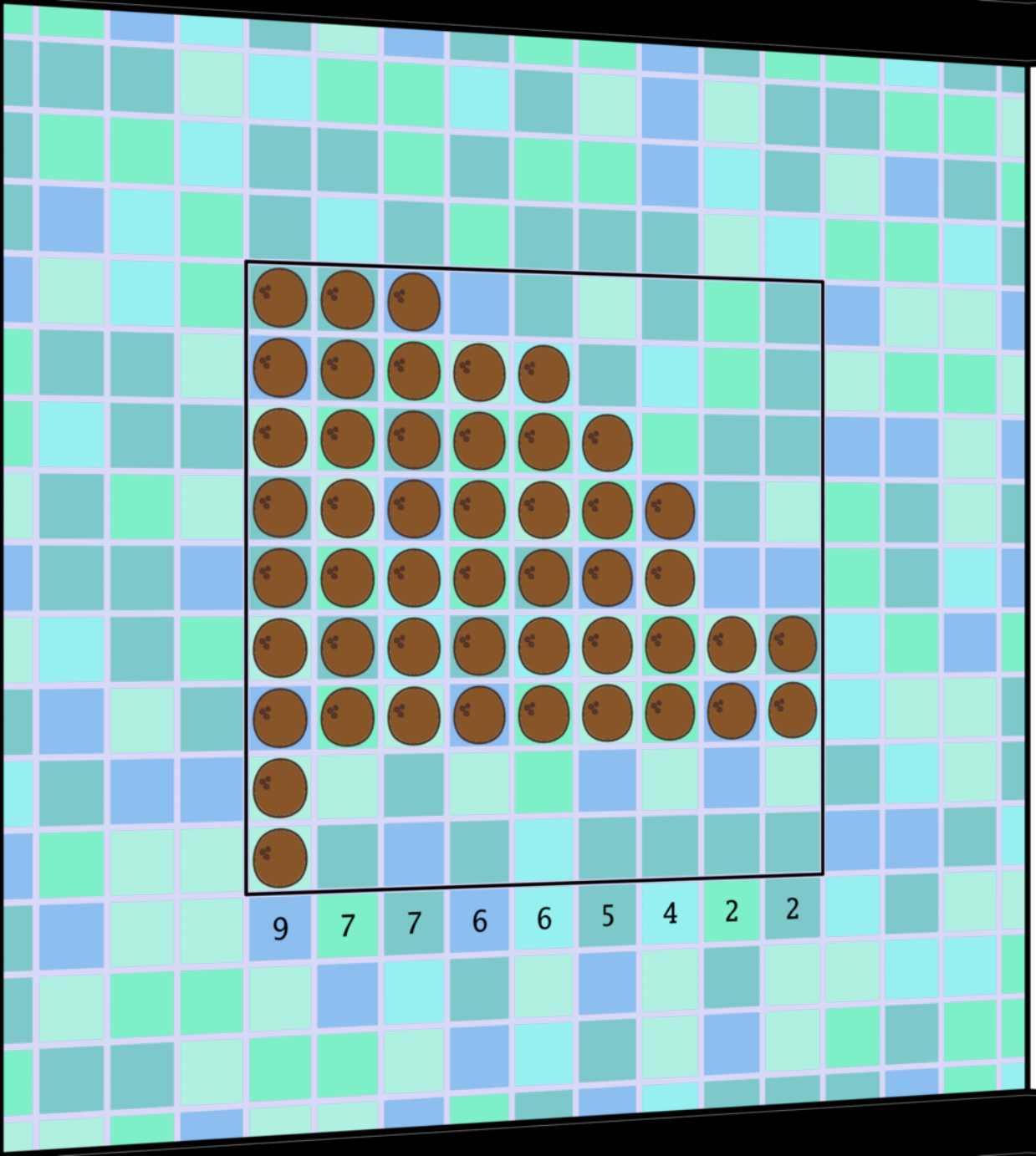
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



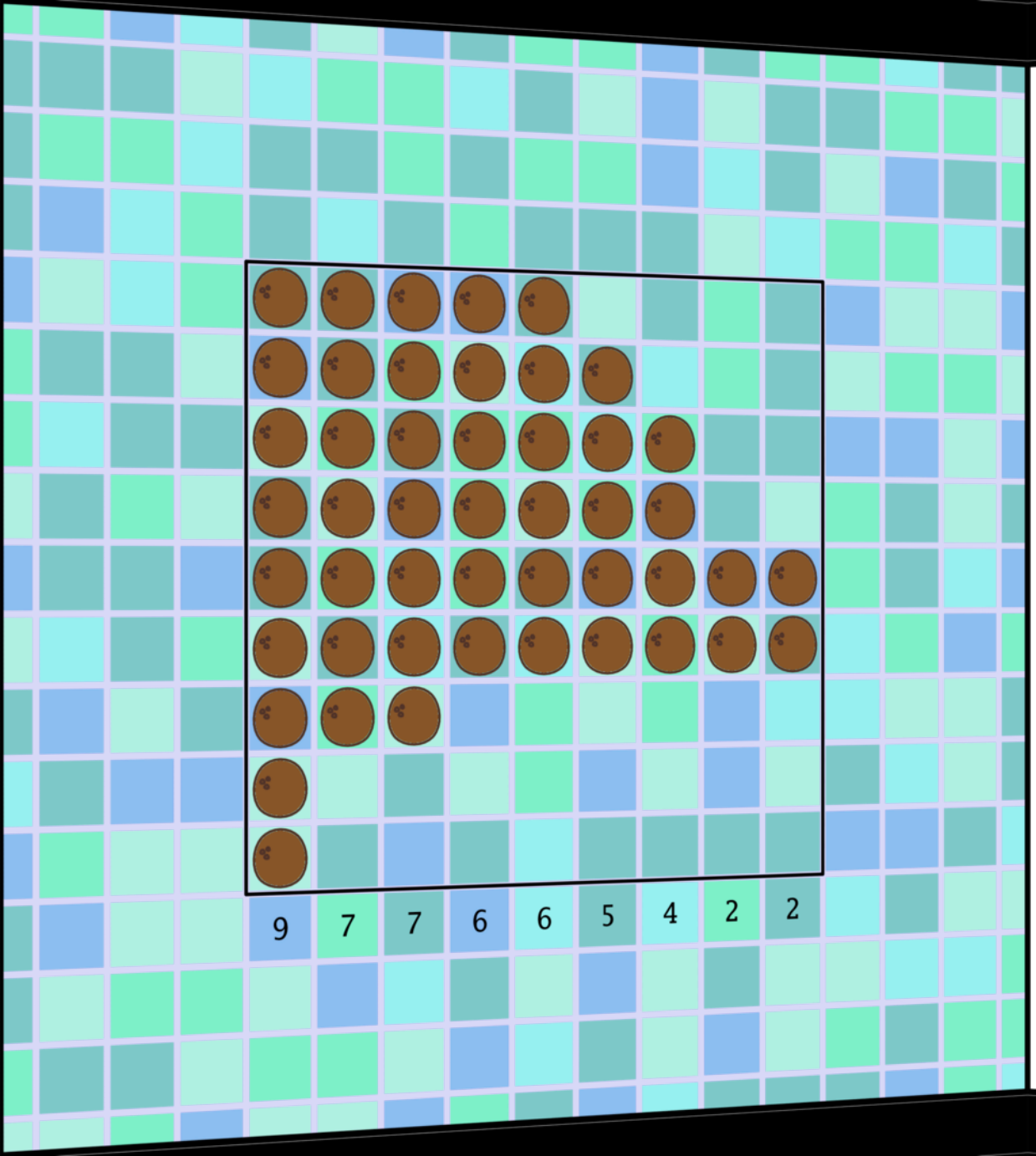
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



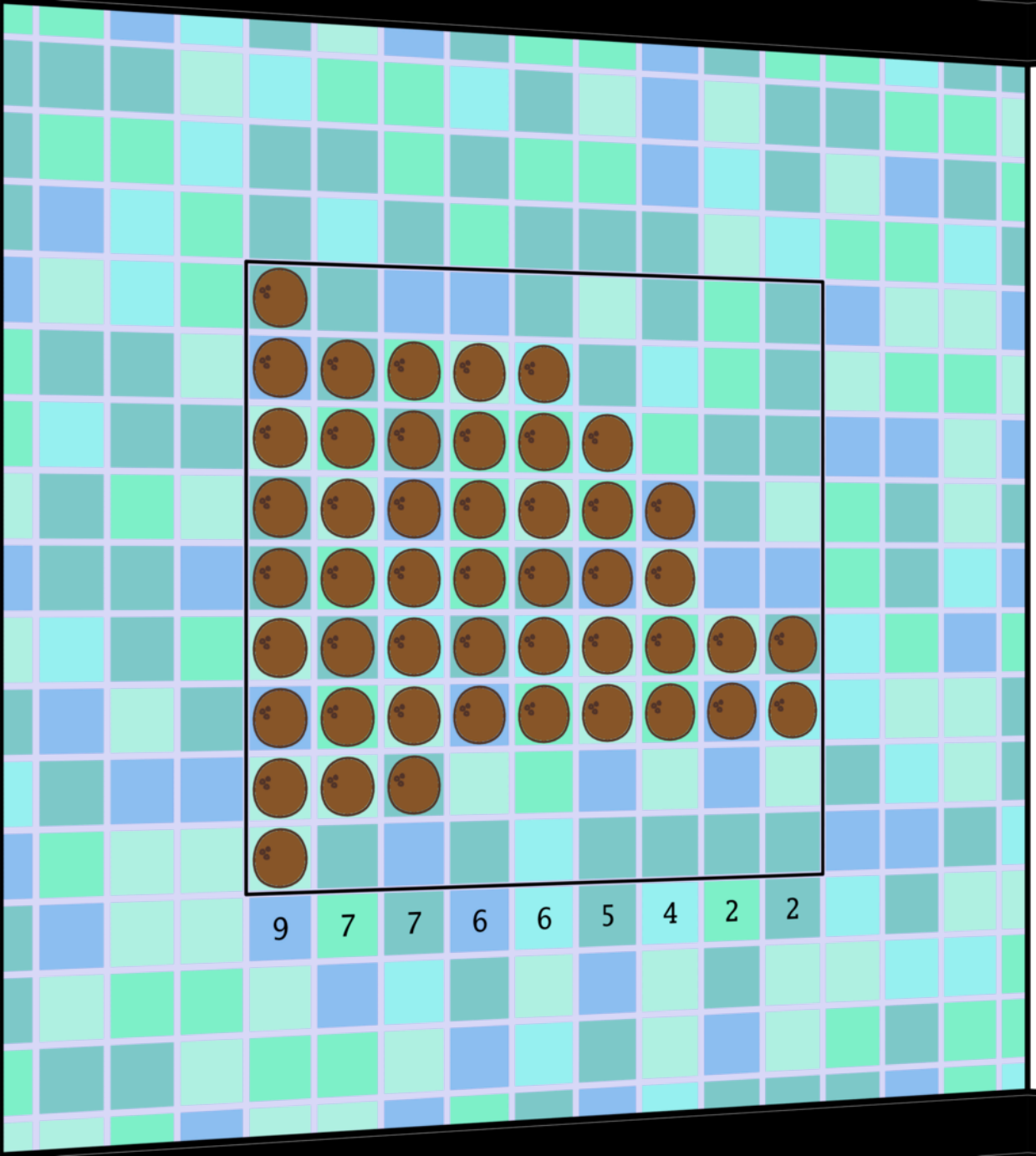
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



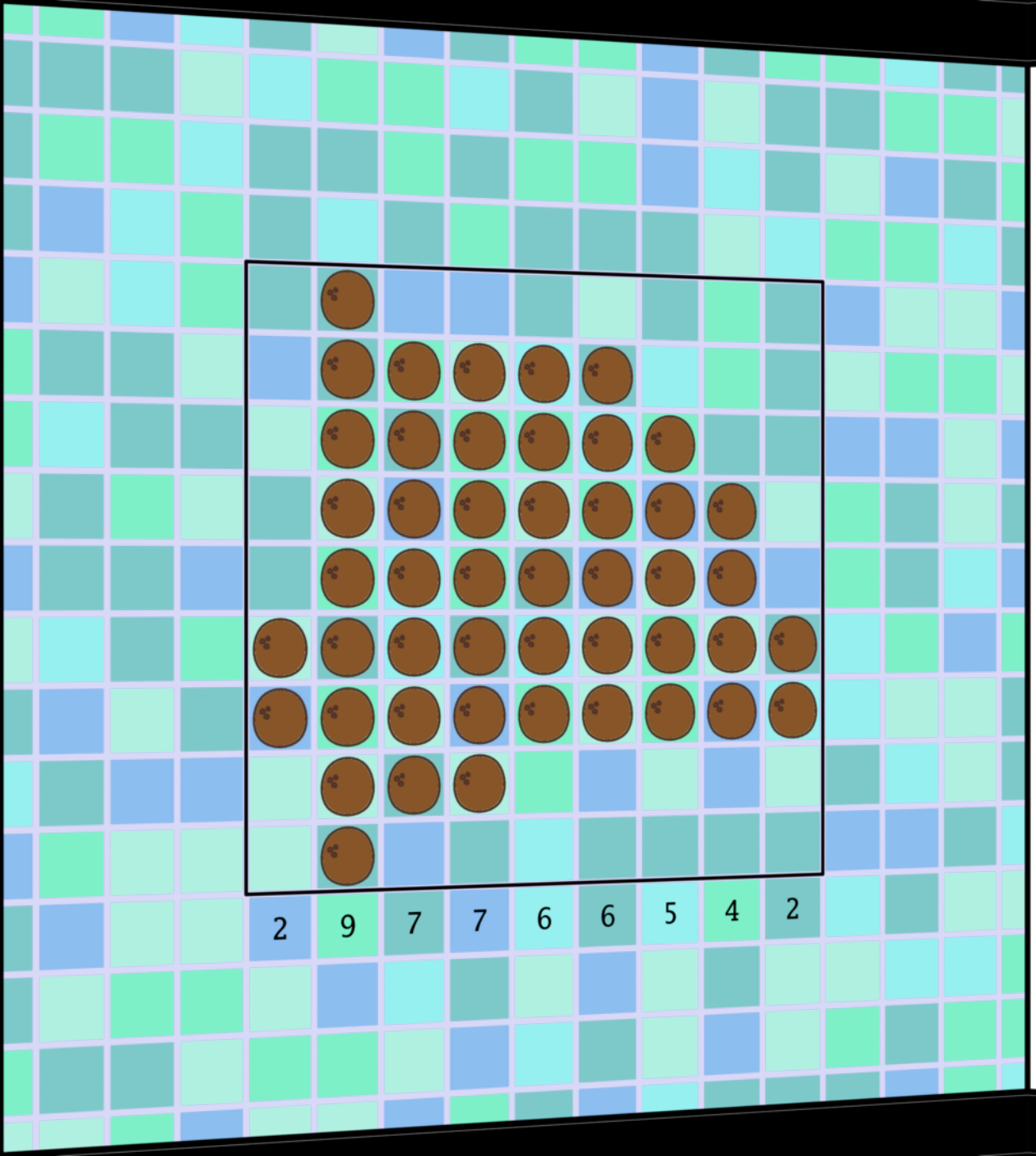
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



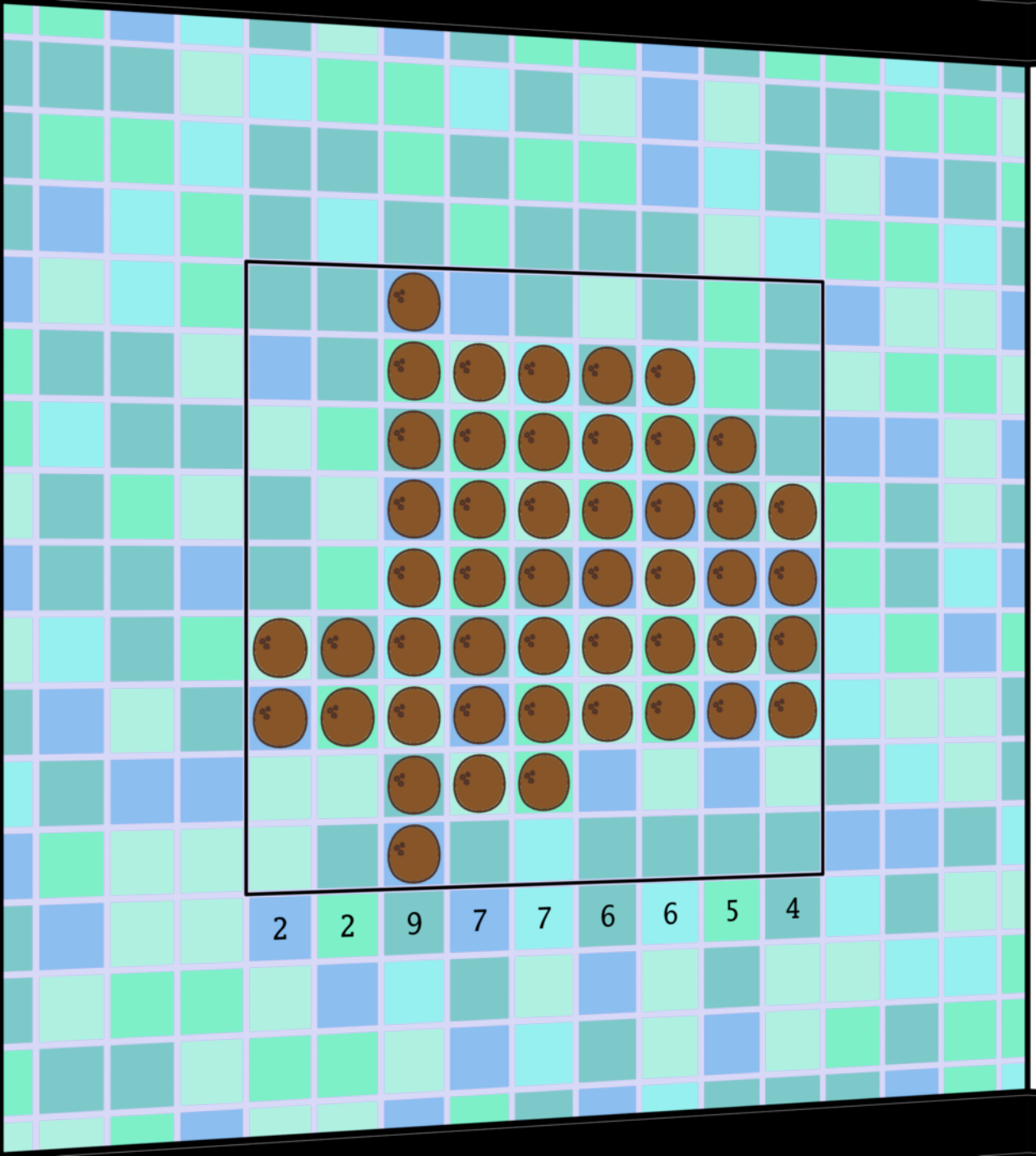
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



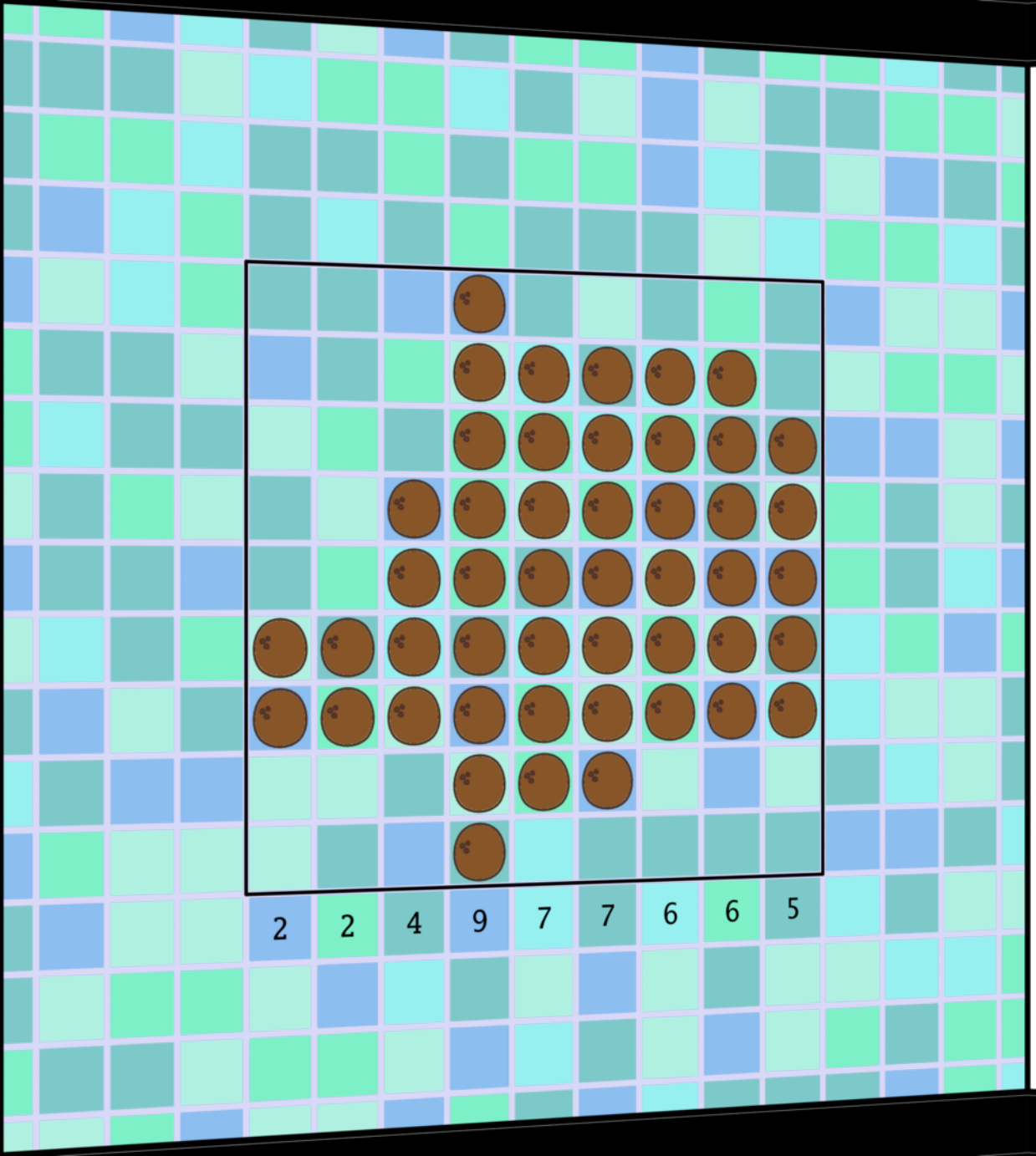
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



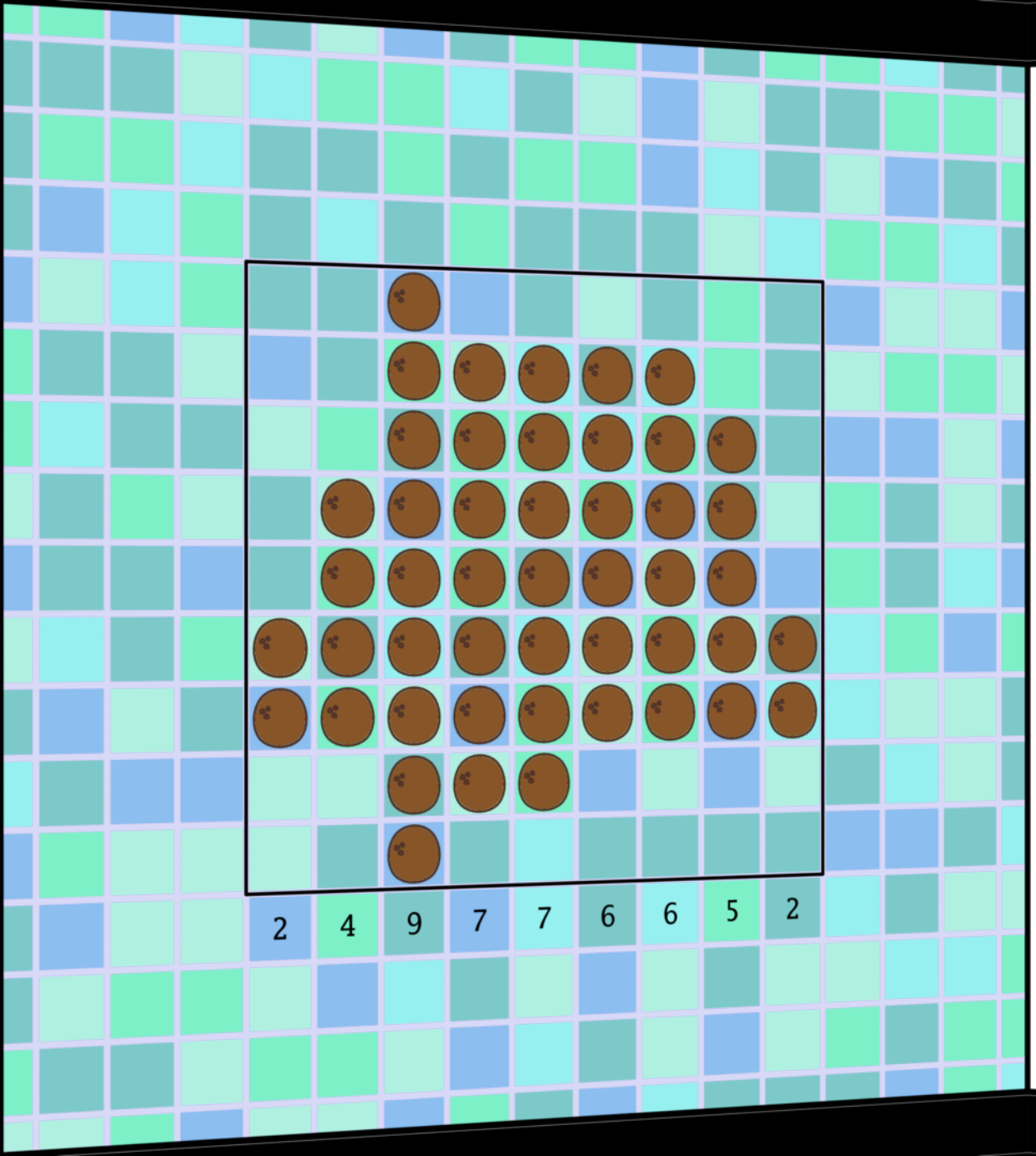
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



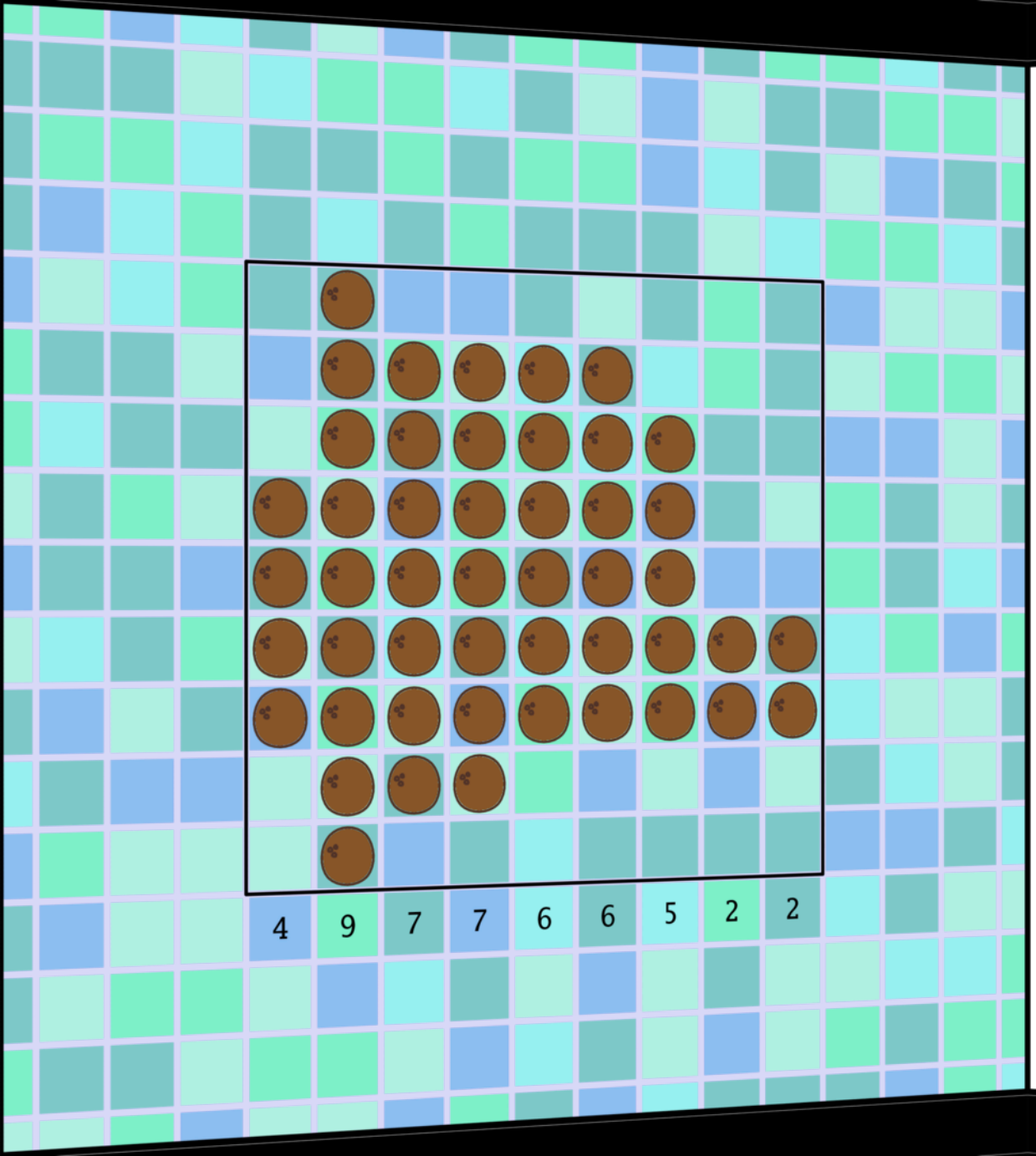
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



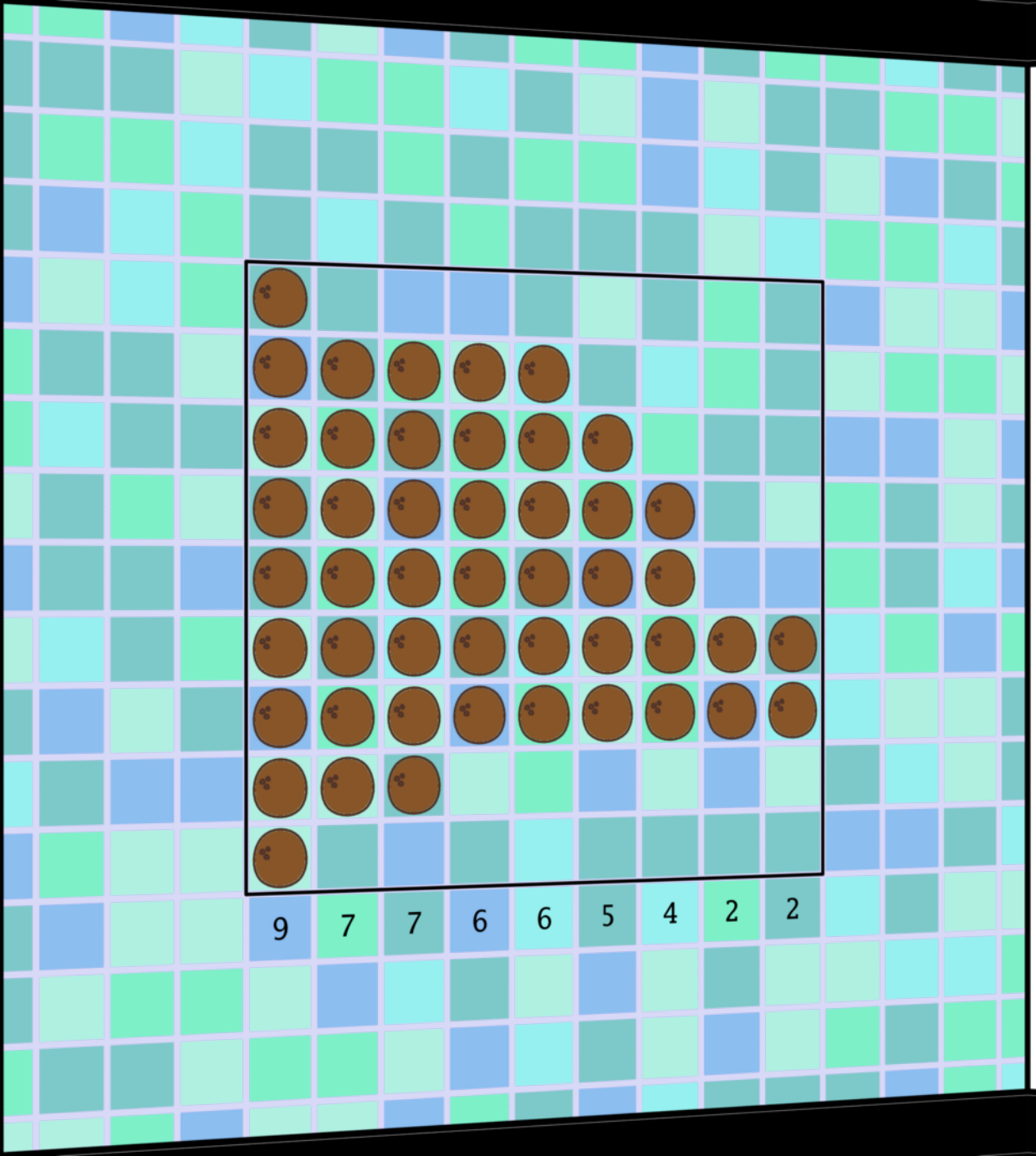
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



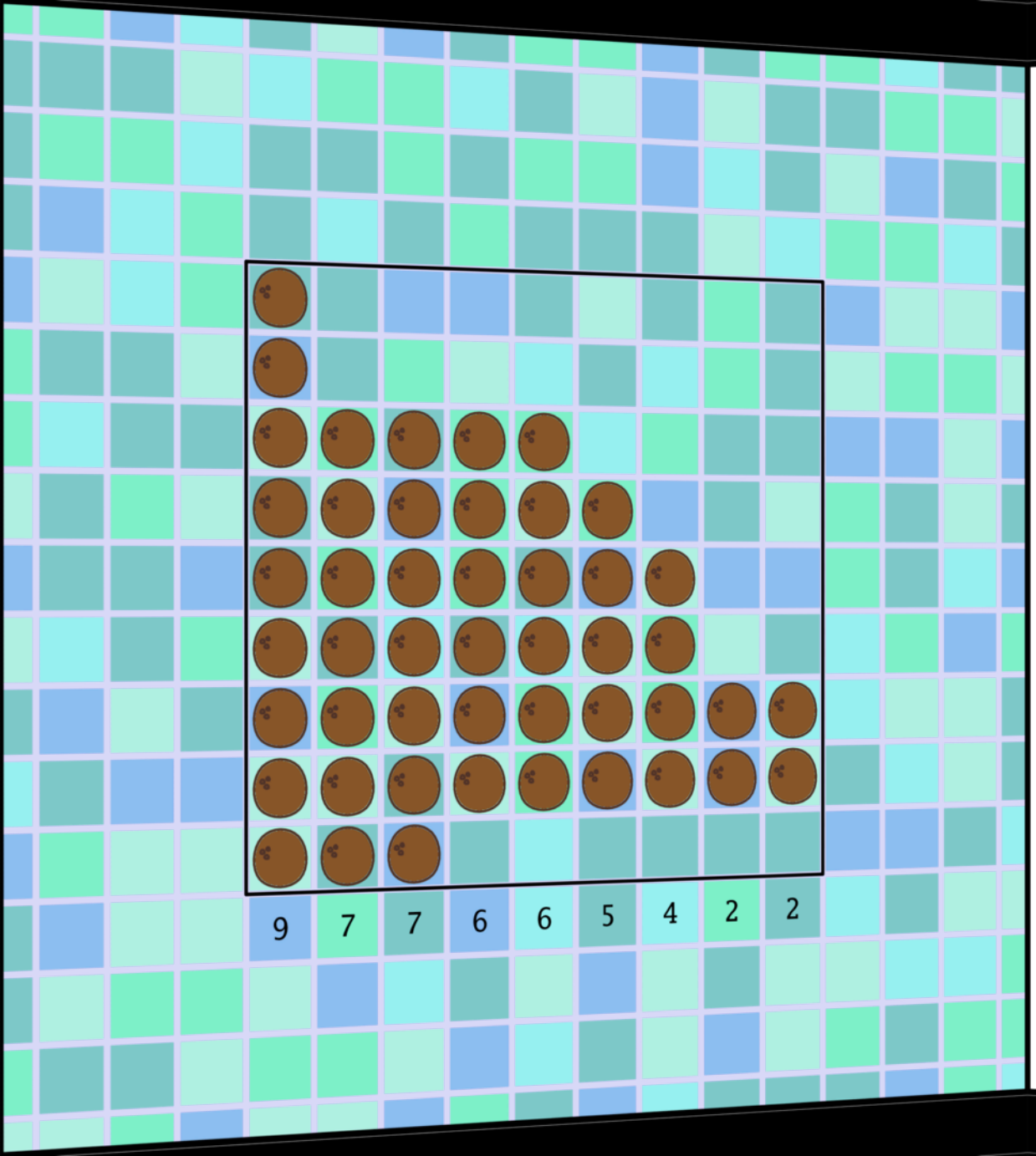
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



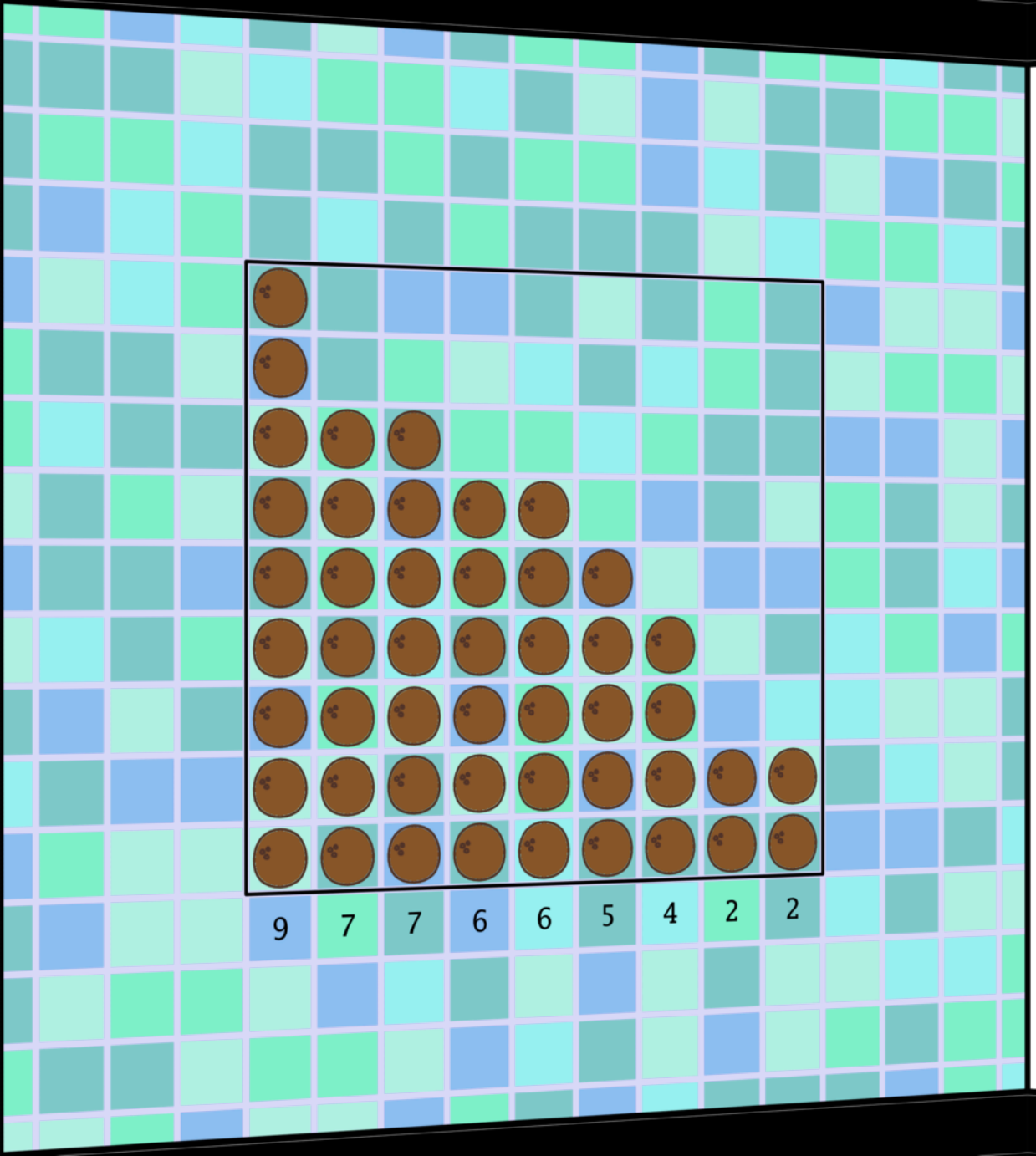
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



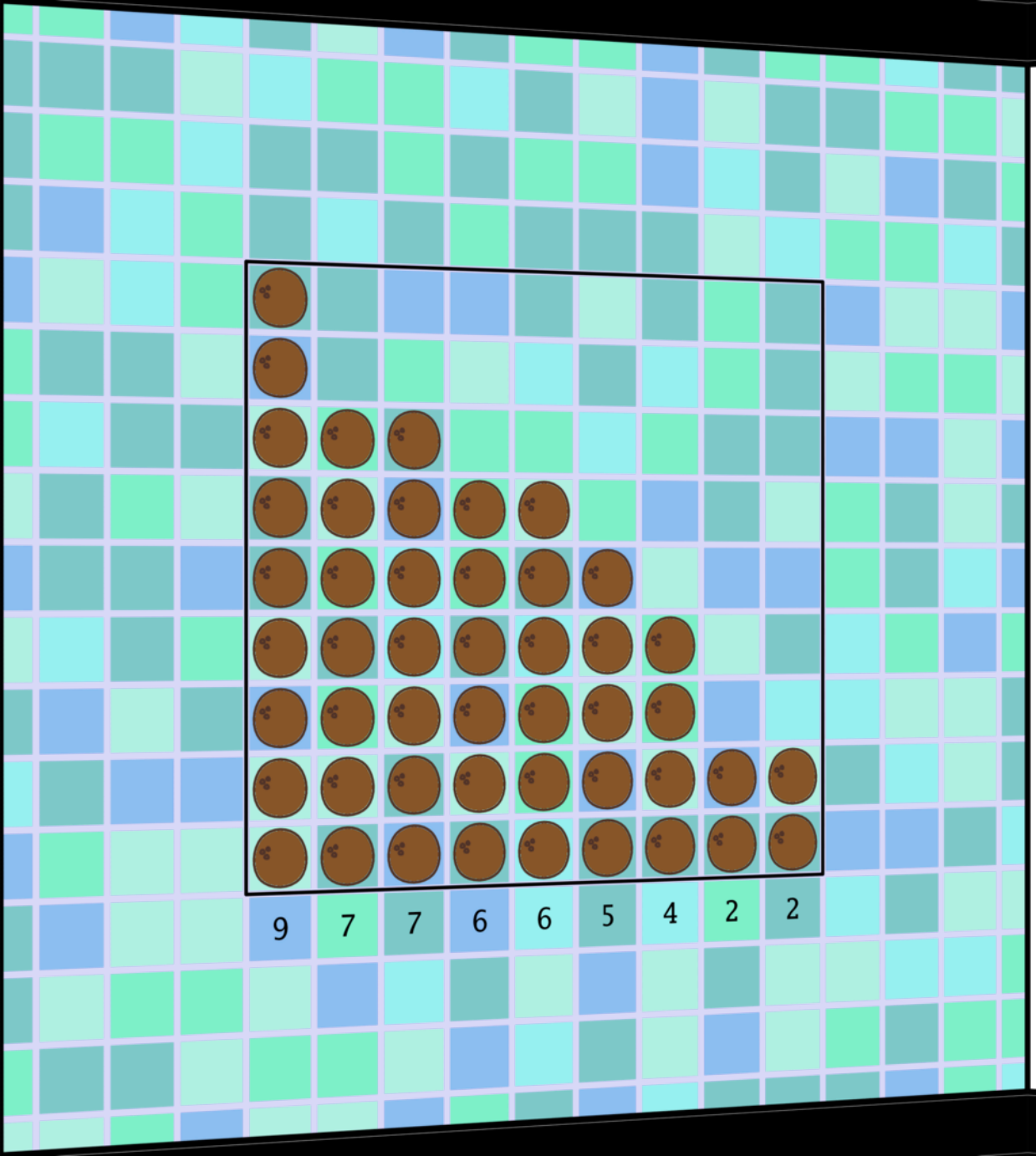
Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.



Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

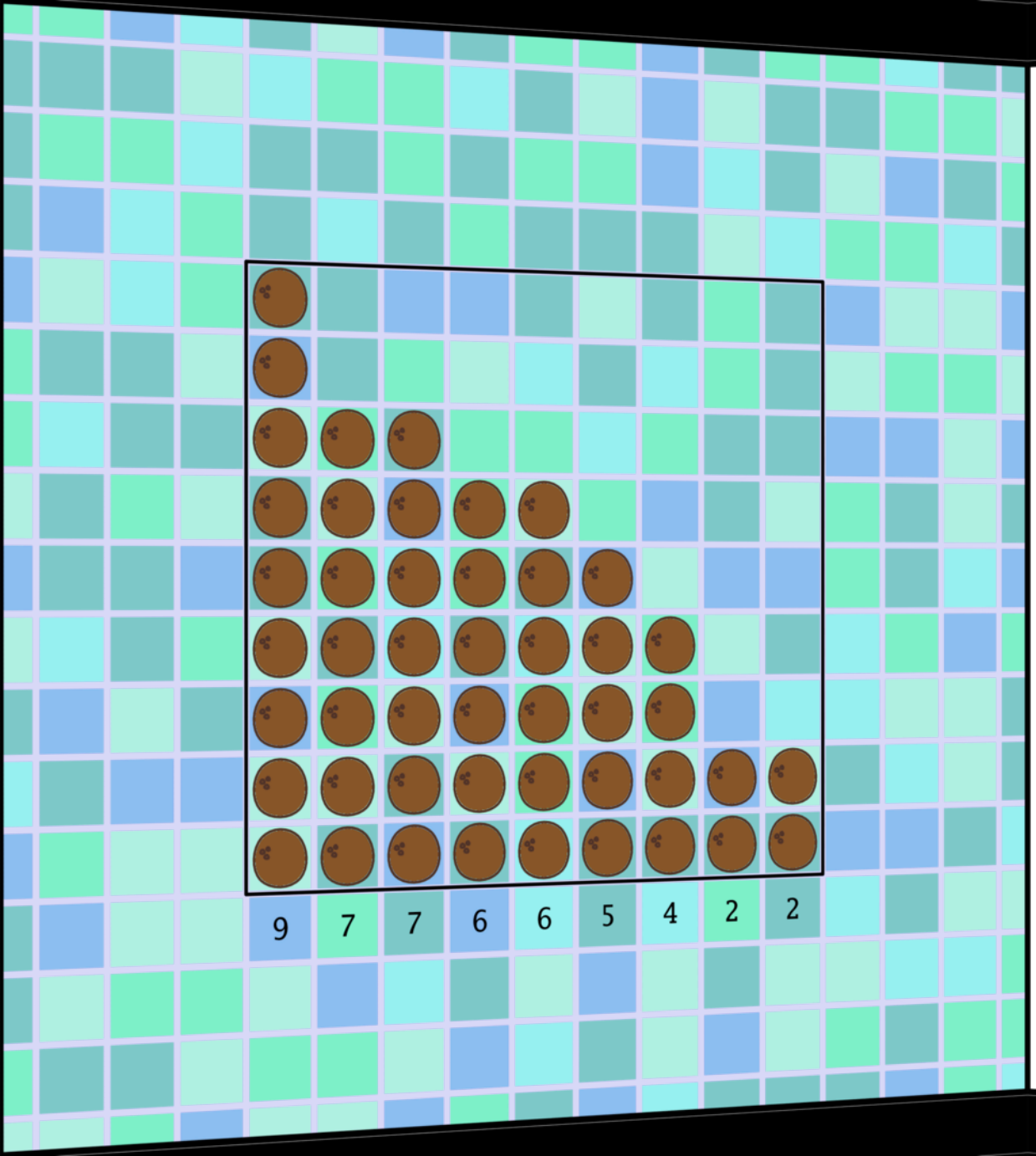
Proof idea. The number of rows and columns of given lengths are invariant.



Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

Proof idea. The number of rows and columns of given lengths are invariant.

Lemma. The number of moves is always even.

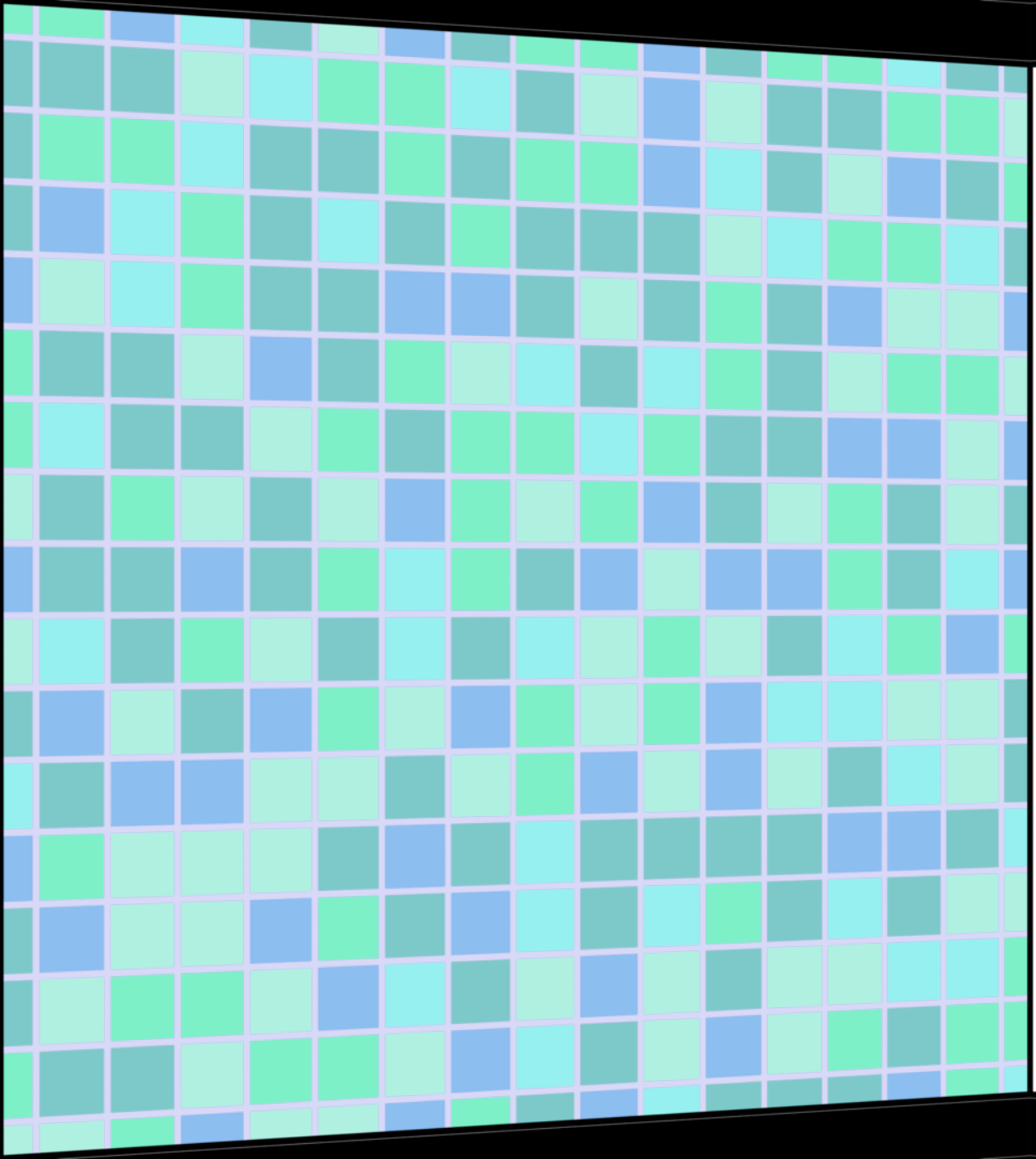


Theorem 1. Two unlabeled canonical configurations P and P' can be reached if and only if $P = P'$.

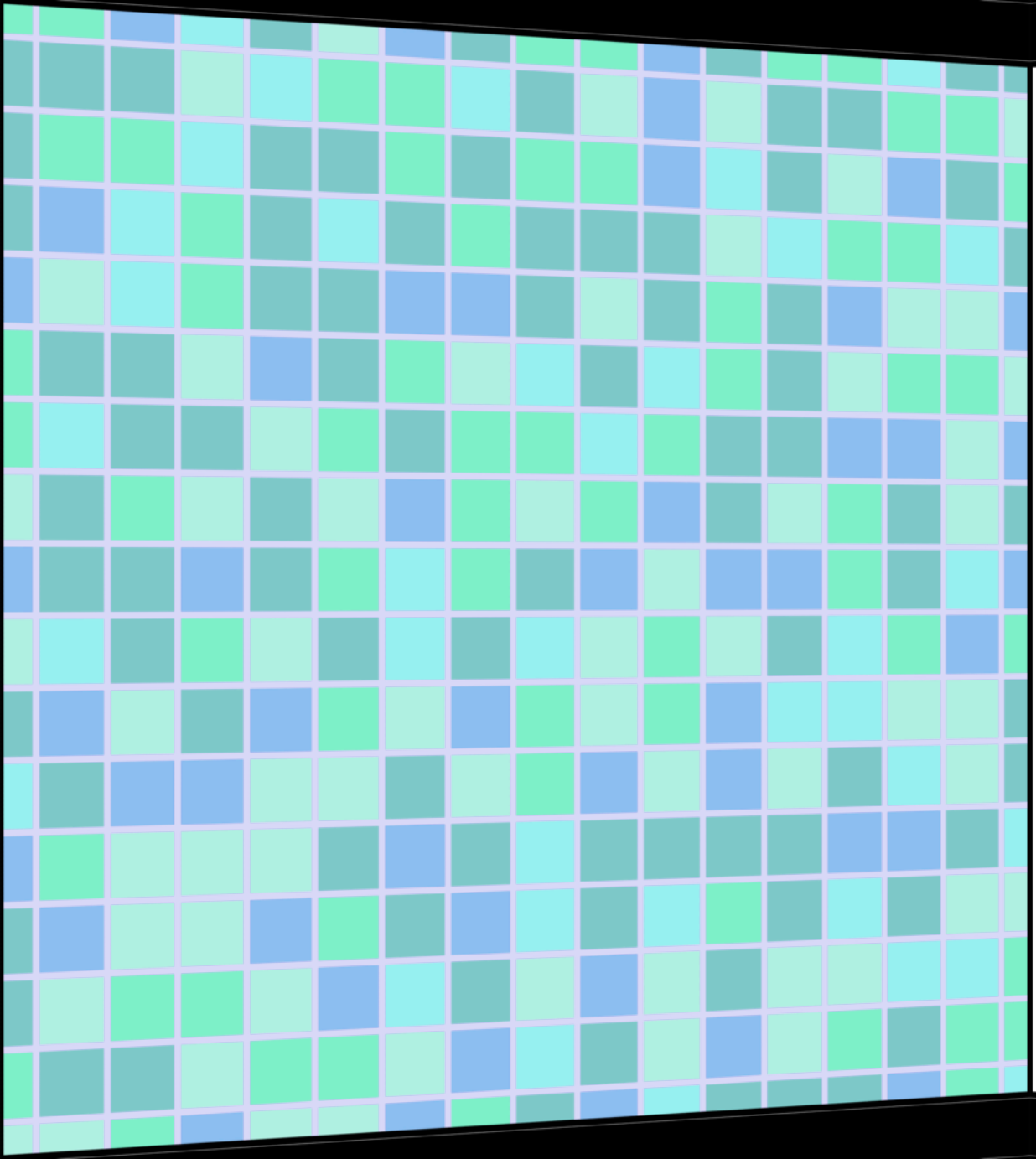
Proof idea. The number of rows and columns of given lengths are invariant.

Lemma. The number of moves is always even.

Proof. Watch the 9s.

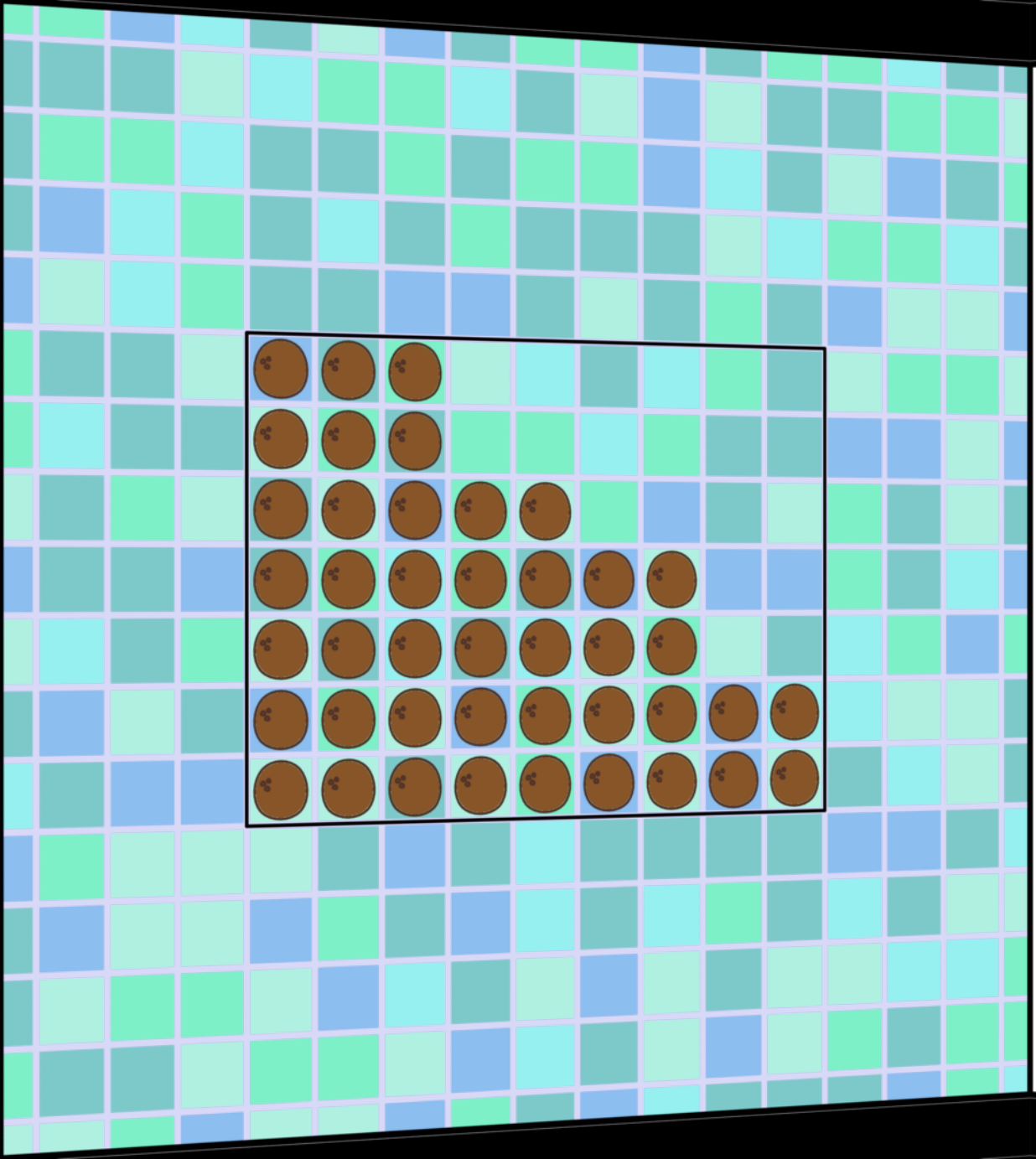


Theorem 2. Every realizable permutation of a canonical configuration is *even*.



Theorem 2. Every realizable permutation of a canonical configuration is *even*.

Proof idea. First reason about permutations on all positions, including *empty* ones.



Theorem 2. Every realizable permutation of a canonical configuration is *even*.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Theorem 2. Every realizable permutation of a canonical configuration is *even*.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Theorem 2. Every realizable permutation of a canonical configuration is *even*.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.

Theorem 2. Every realizable permutation of a canonical configuration is *even*.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.

Theorem 2. Every realizable permutation of a canonical configuration is *even*.

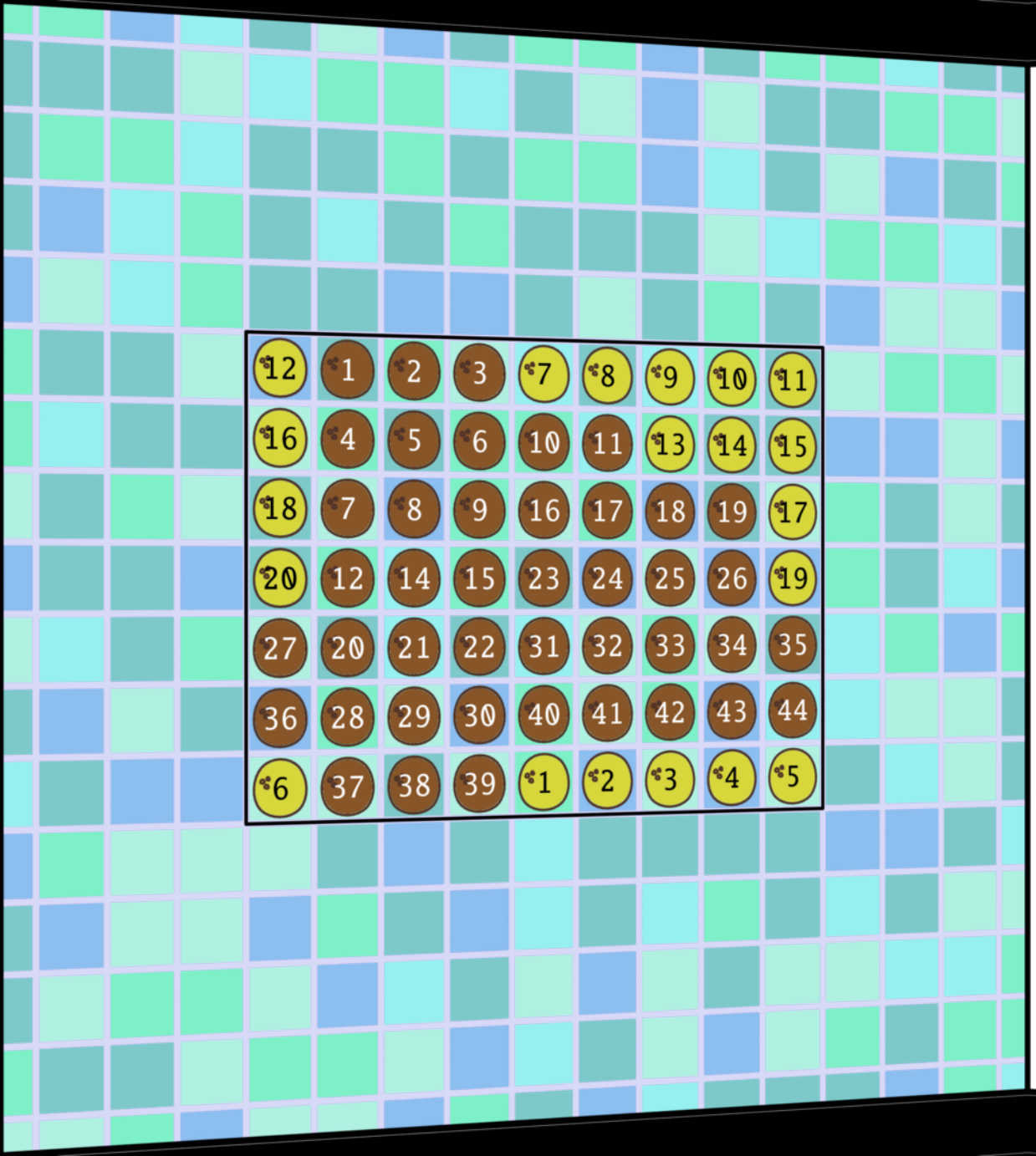
Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.

Theorem 2. Every realizable permutation of a canonical configuration is *even*.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.



Theorem 2. Every realizable permutation of a canonical configuration is *even*.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.

Theorem 2. Every realizable permutation of a canonical configuration is *even*.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.

Theorem 2. Every realizable permutation of a canonical configuration is *even*.

Proof idea. First reason about permutations on all positions, including *empty* ones.

Each empty position inside the bounding box contains an *antinut*.

Each move is a *cyclic permutation* on some rows or columns.

Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

The final permutation on *all coconuts and antinuts* is even.

Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

The final permutation on *all coconuts and antinuts* is even.

It could still be that the permutation of *only the coconuts* is odd!

Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

The final permutation on *all coconuts and antinuts* is even.

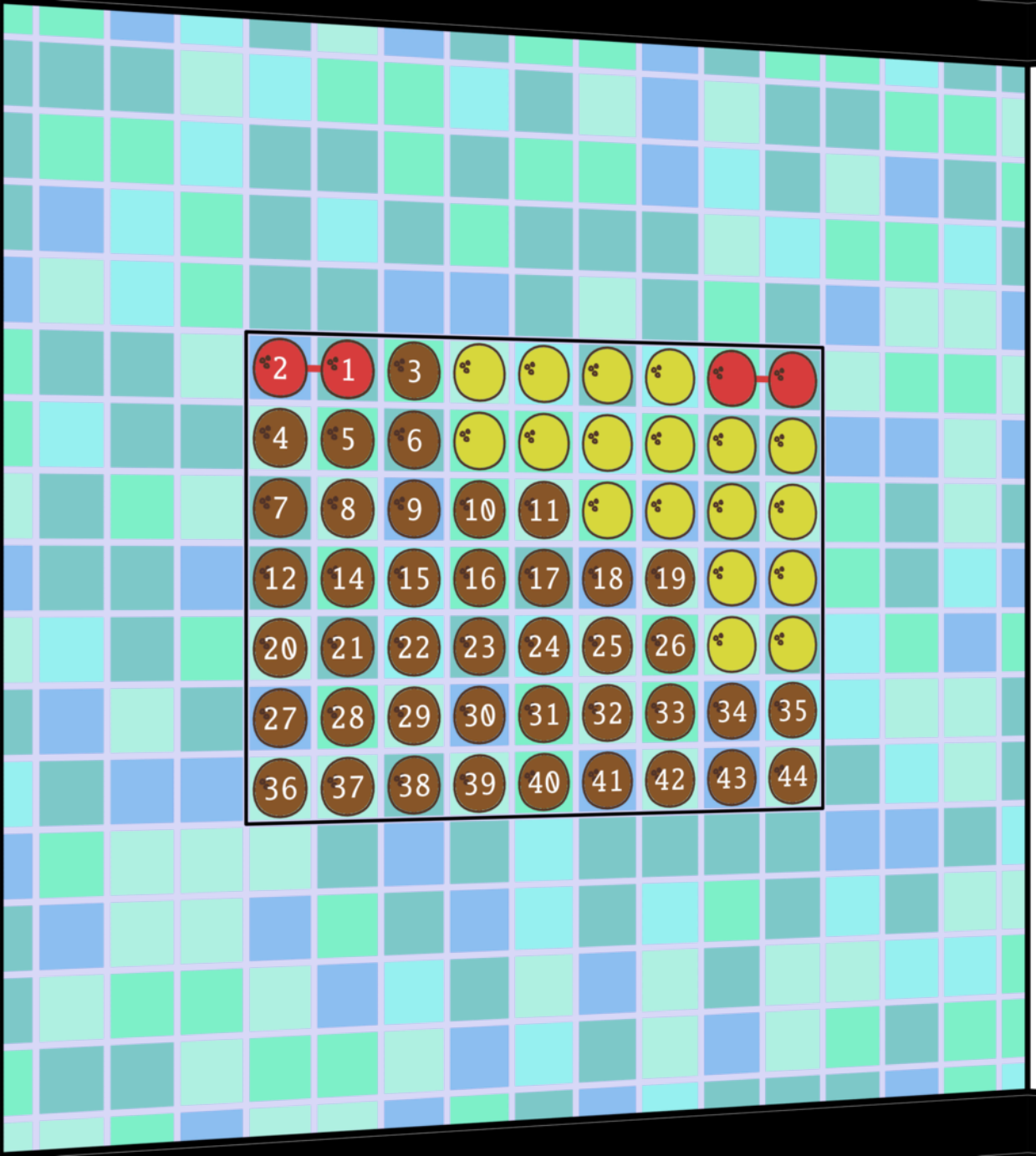
It could still be that the permutation of *only the coconuts* is odd!

Each move is a *cyclic permutation* on some rows or columns.

By the previous lemma, the number of moves is even.

The final permutation on *all coconuts and antinuts* is even.

It could still be that the permutation of *only the coconuts* is odd!

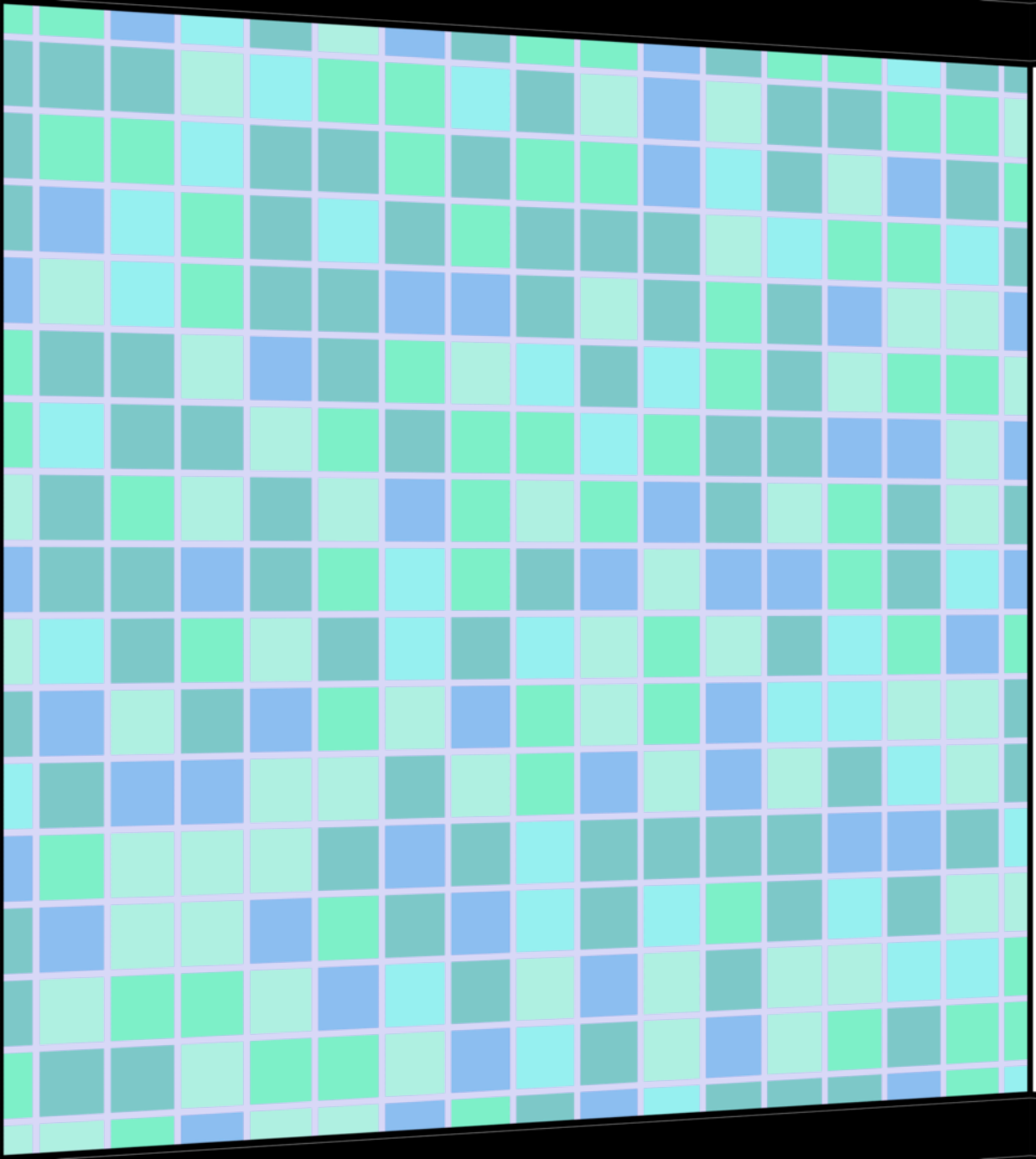


Each move is a *cyclic permutation* on some rows or columns.

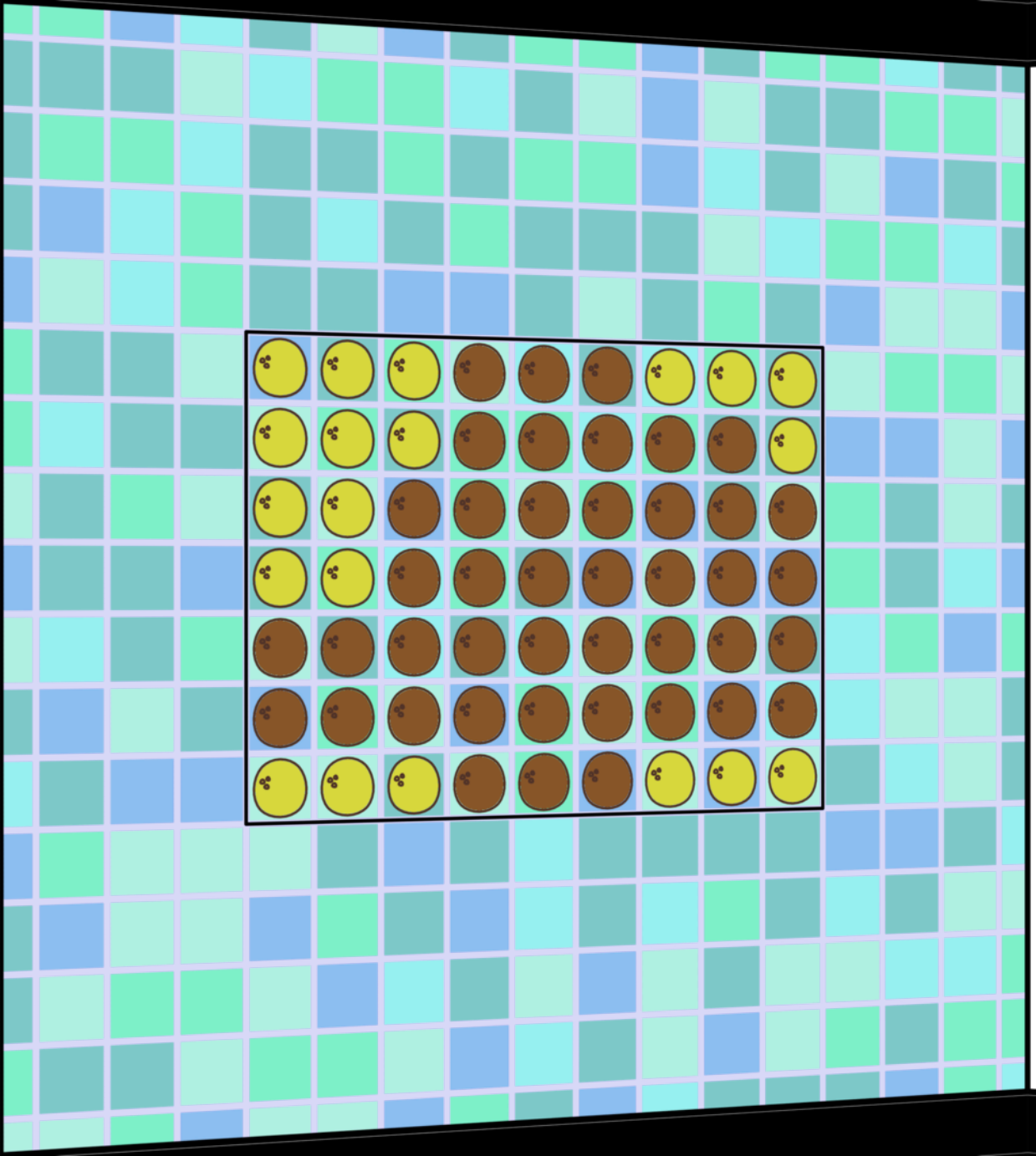
By the previous lemma, the number of moves is even.

The final permutation on *all coconuts and antinuts* is even.

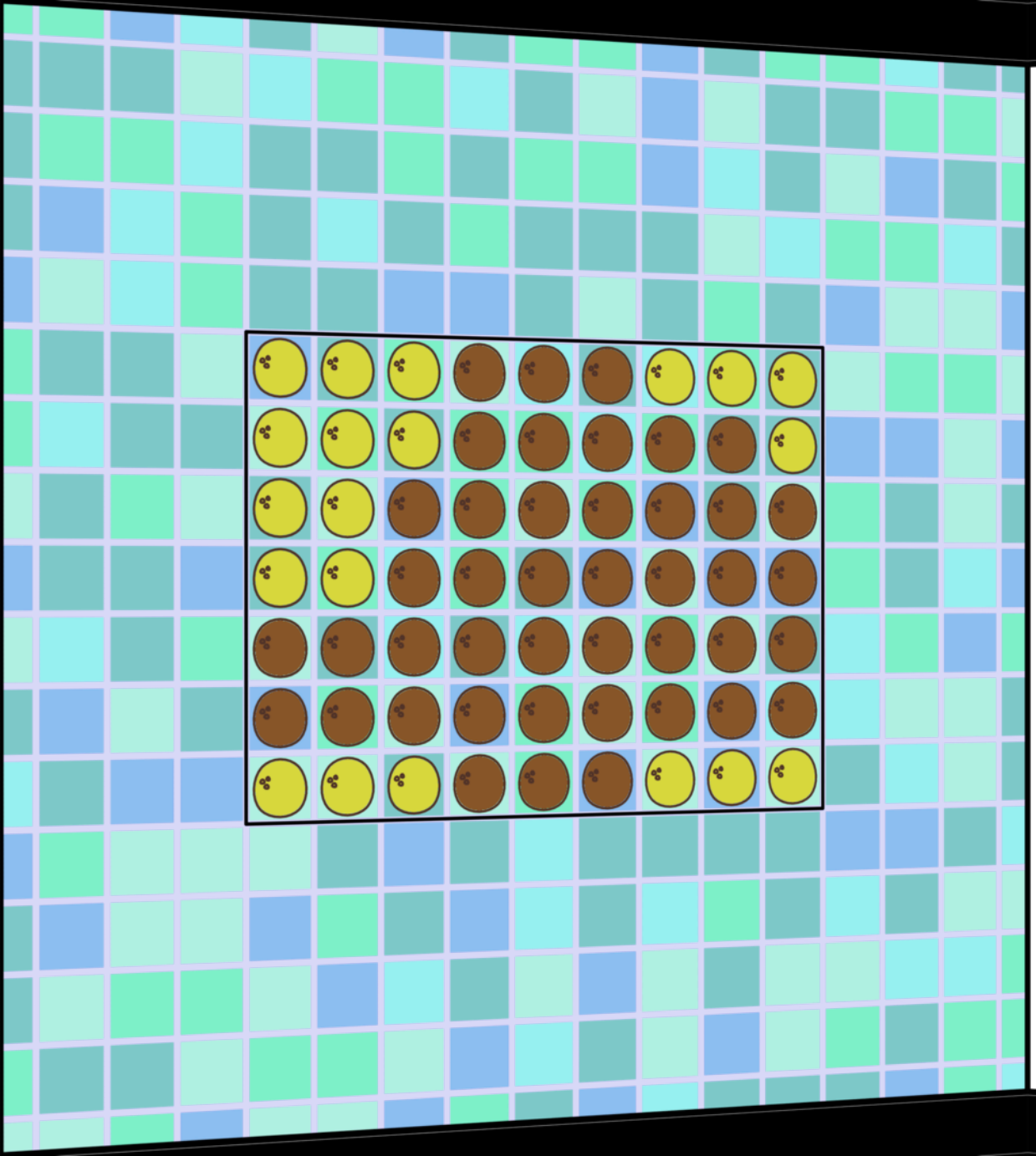
It could still be that the permutation of *only the coconuts* is odd!



Take a closer look at the antinuts.

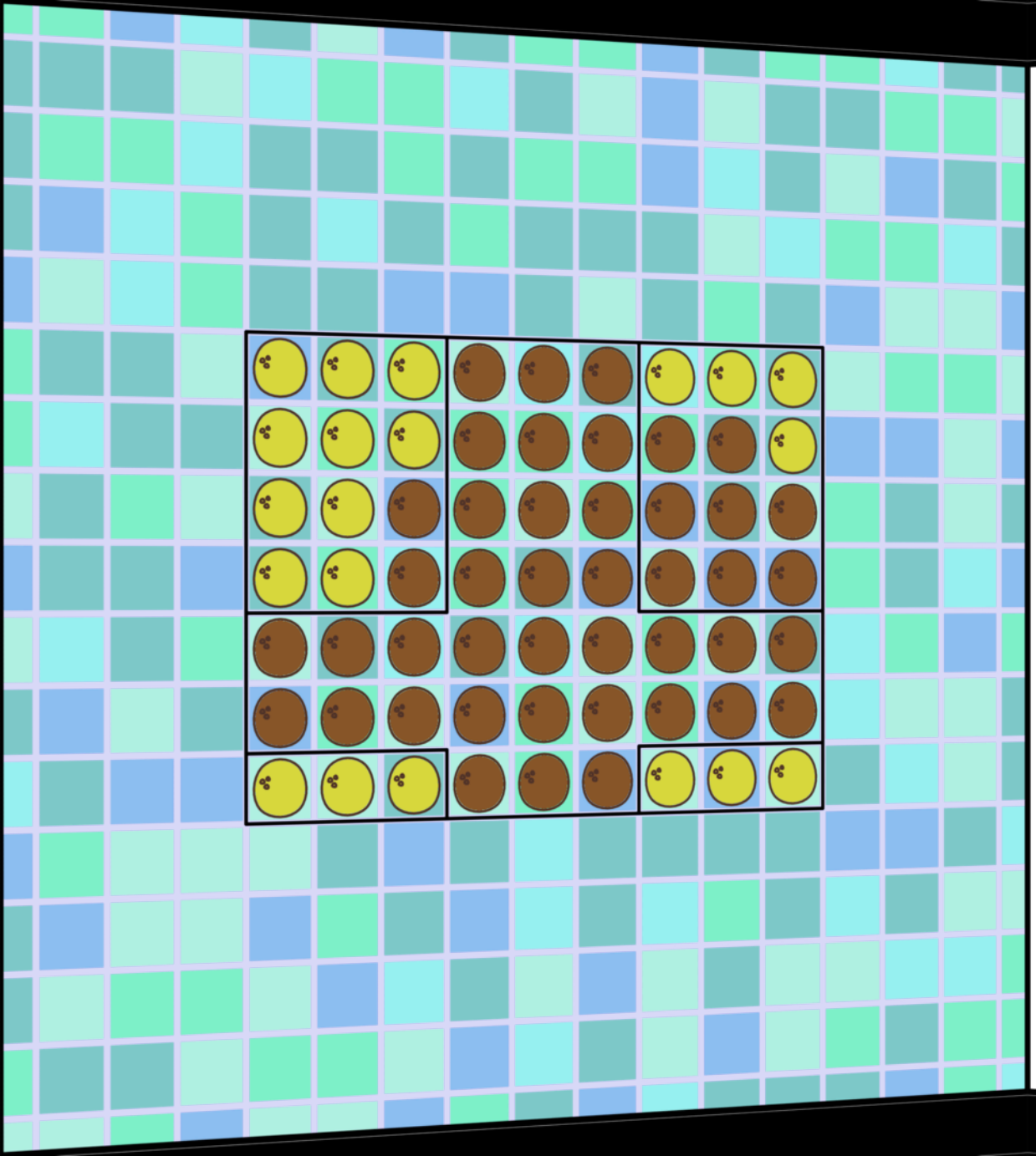


Take a closer look at the antinuts.



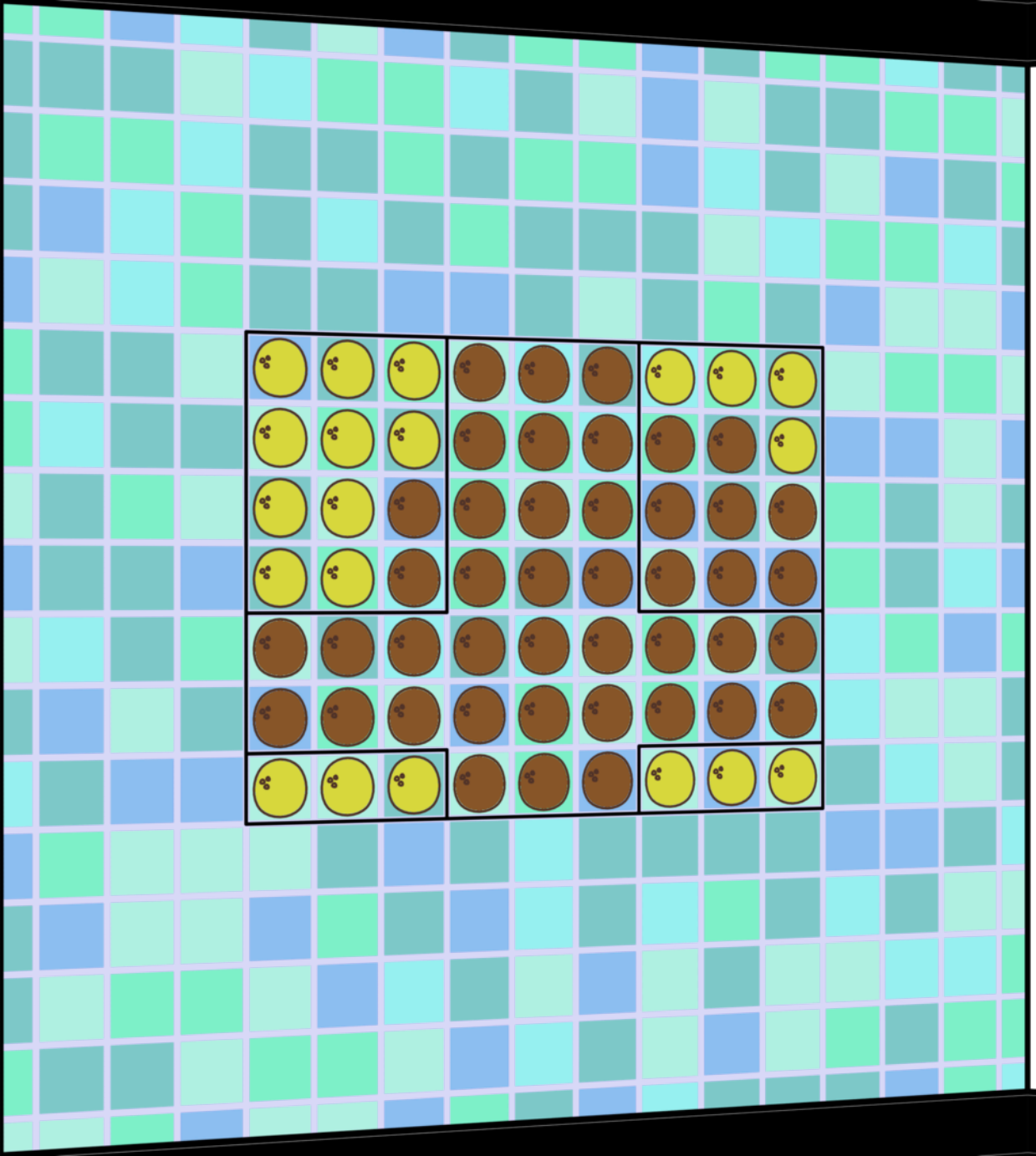
Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.



Take a closer look at the antinuts.

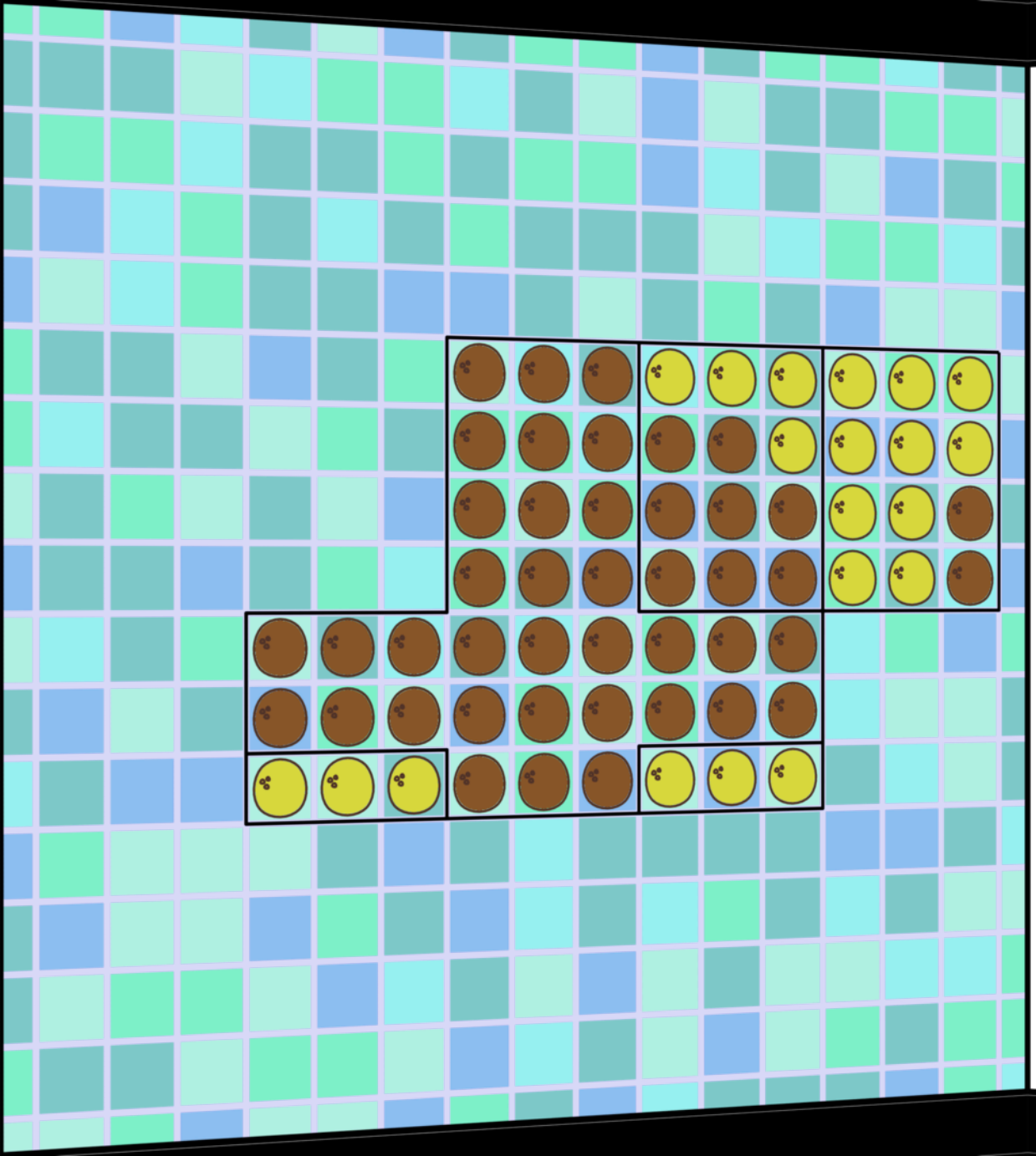
The full rows and columns separate them into four quadrants.



Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

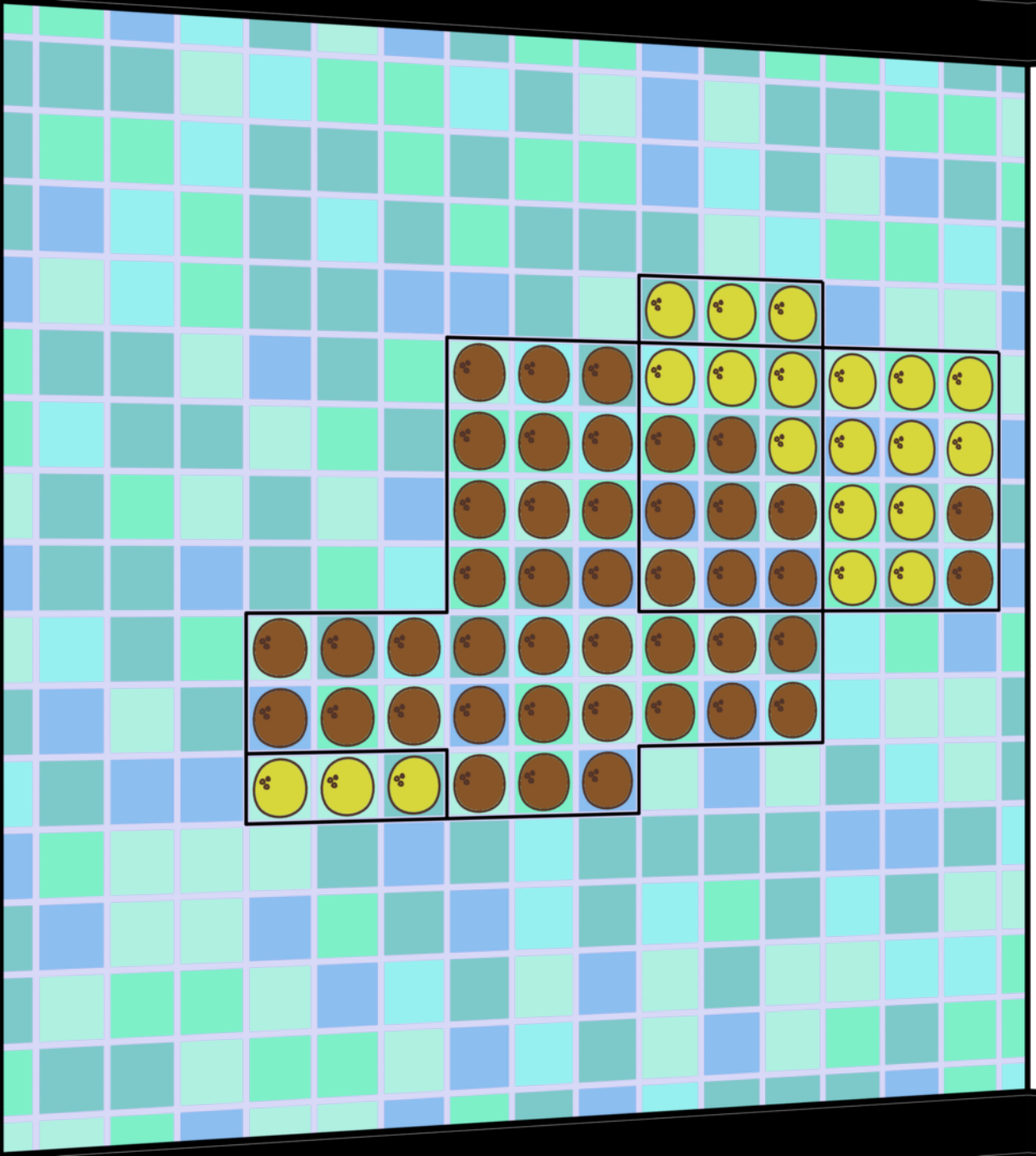
Put them together.



Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.



Take a closer look at the antinuts.

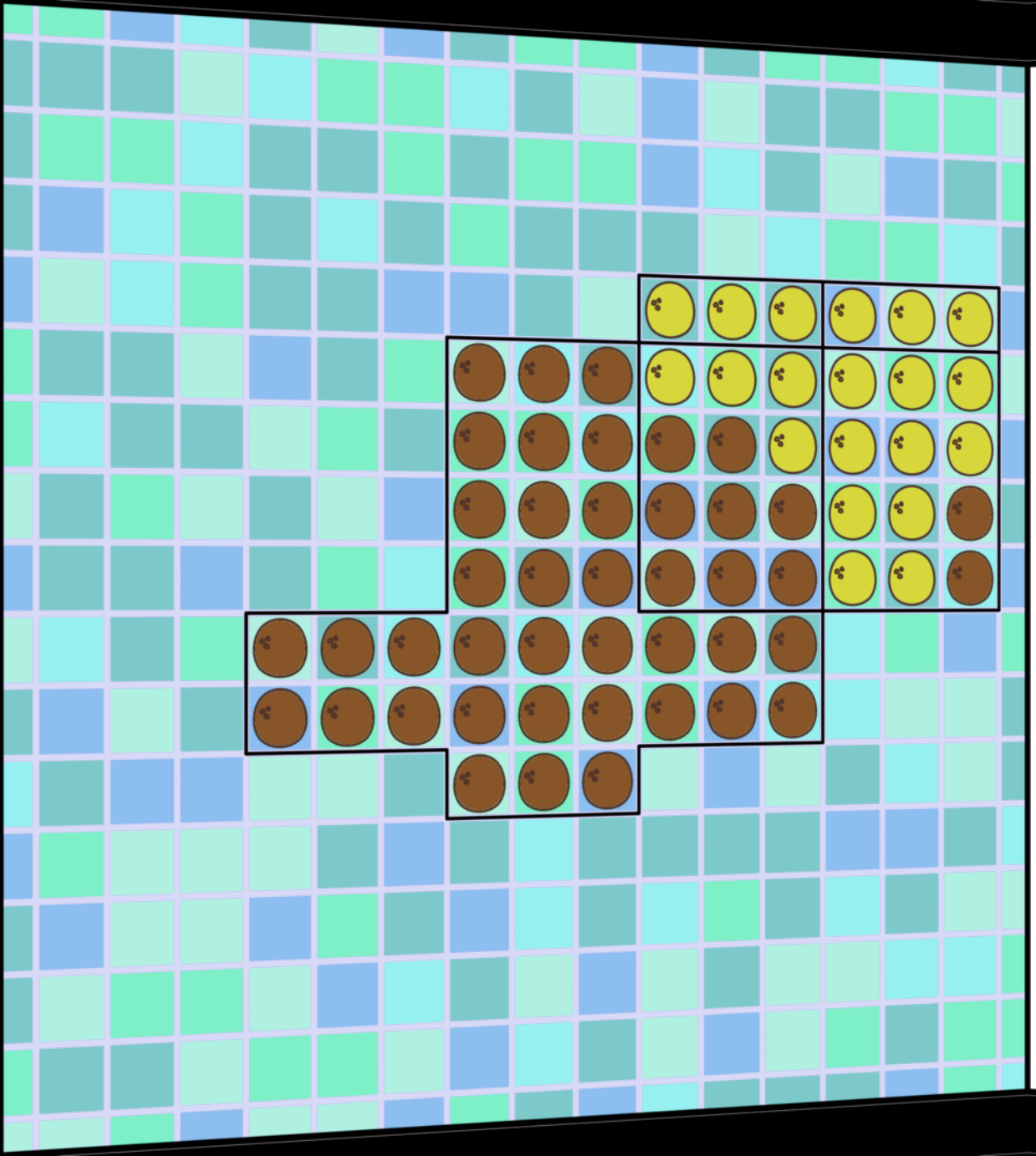
The full rows and columns separate them into four quadrants.

Put them together.

Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.

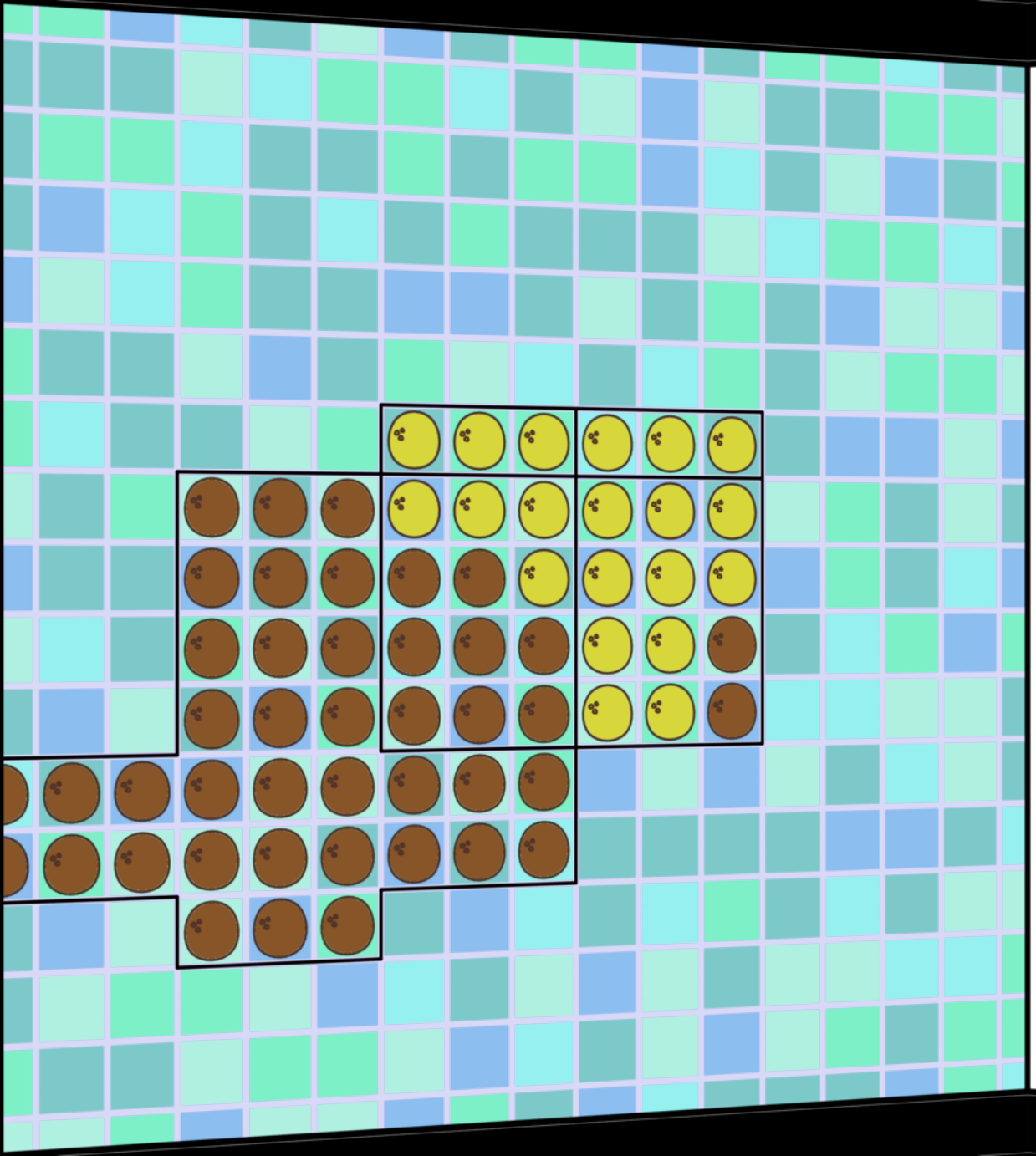


Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.

Center the view and forget about the full rows and columns.

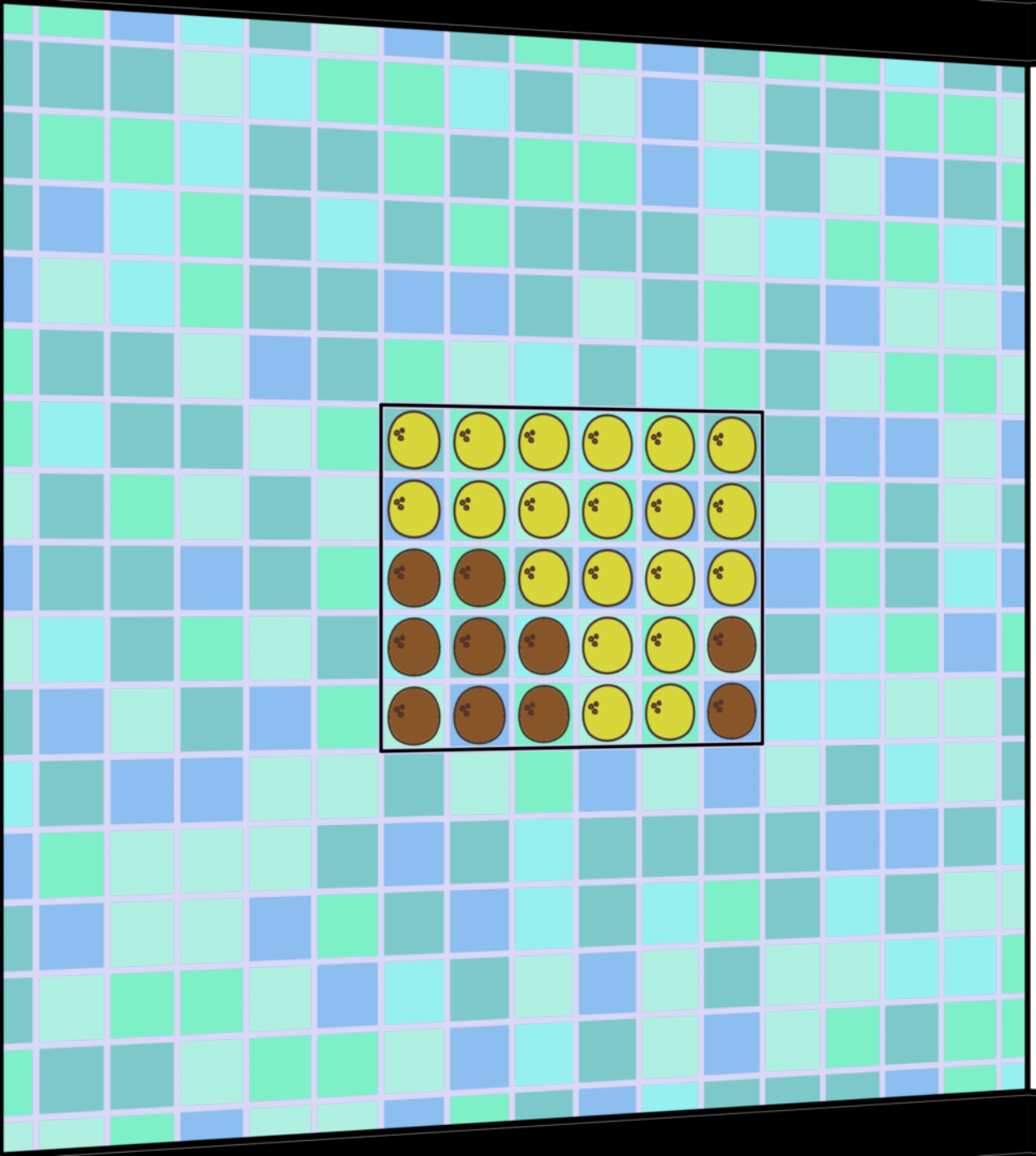


Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.

Center the view and forget about the full rows and columns.

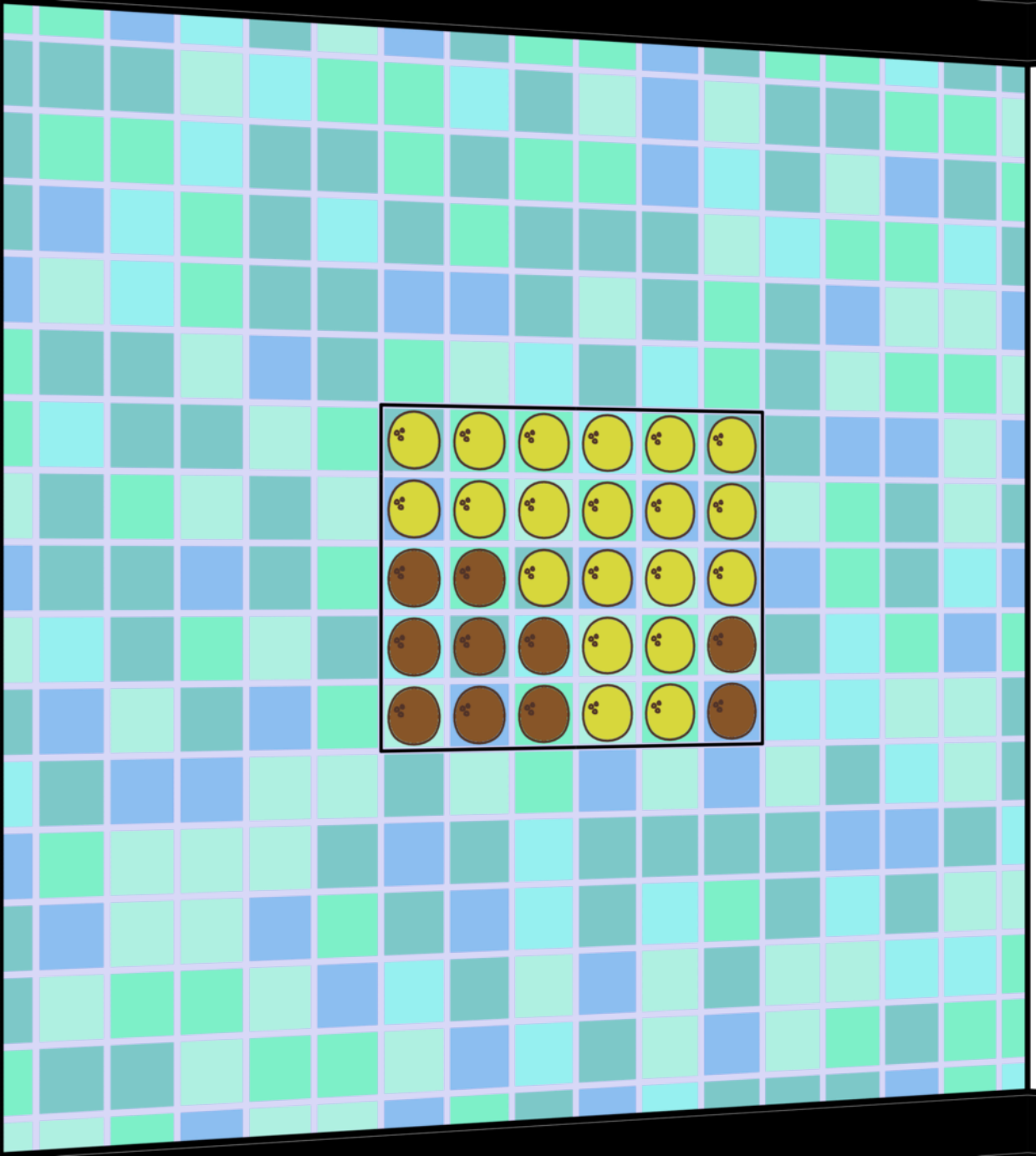


Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

Put them together.

Center the view and forget about the full rows and columns.



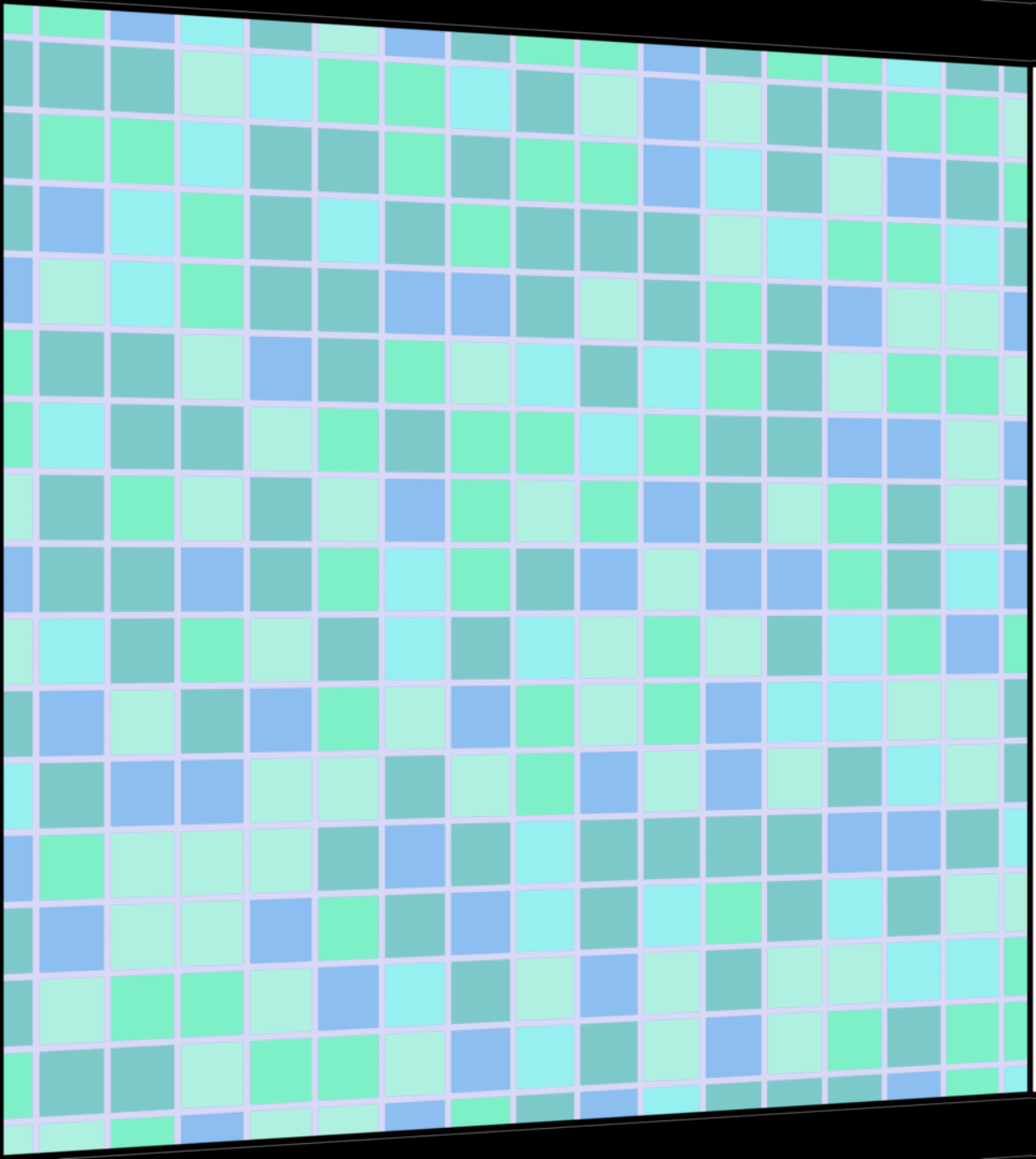
Take a closer look at the antinuts.

The full rows and columns separate them into four quadrants.

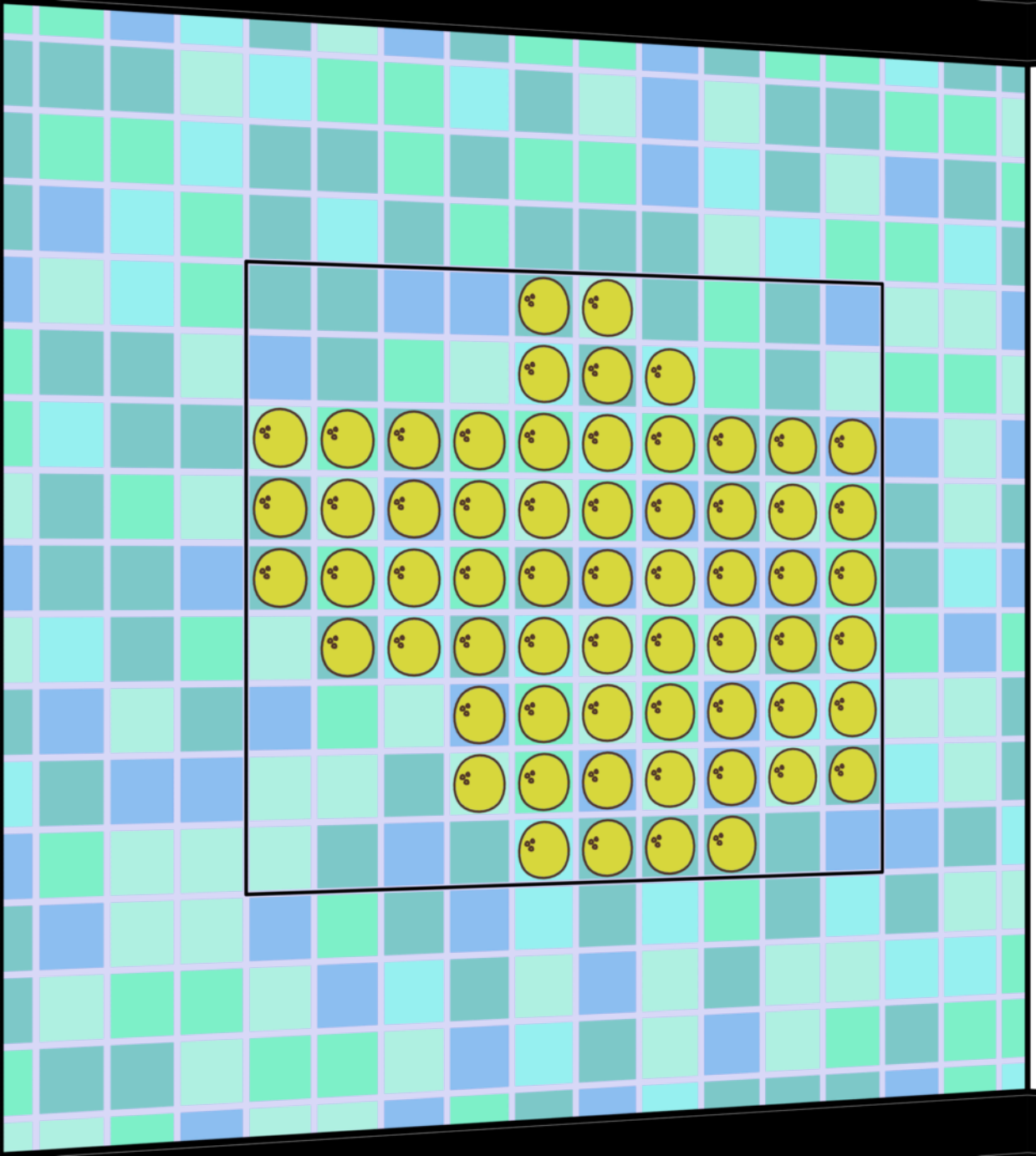
Put them together.

Center the view and forget about the full rows and columns.

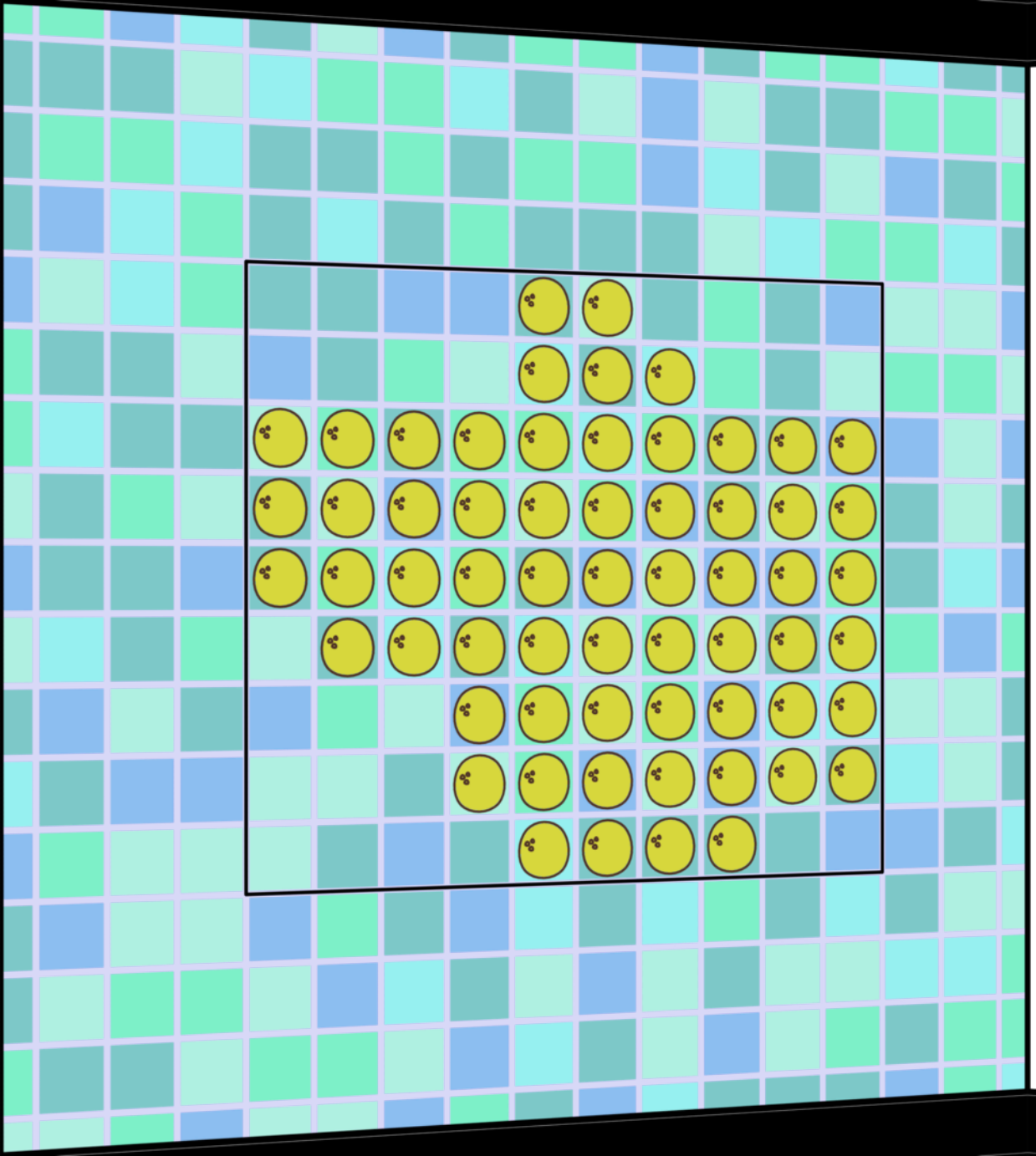
It's the *antinut-centric* view of a configuration.



We can play a dual game on the antinuts!

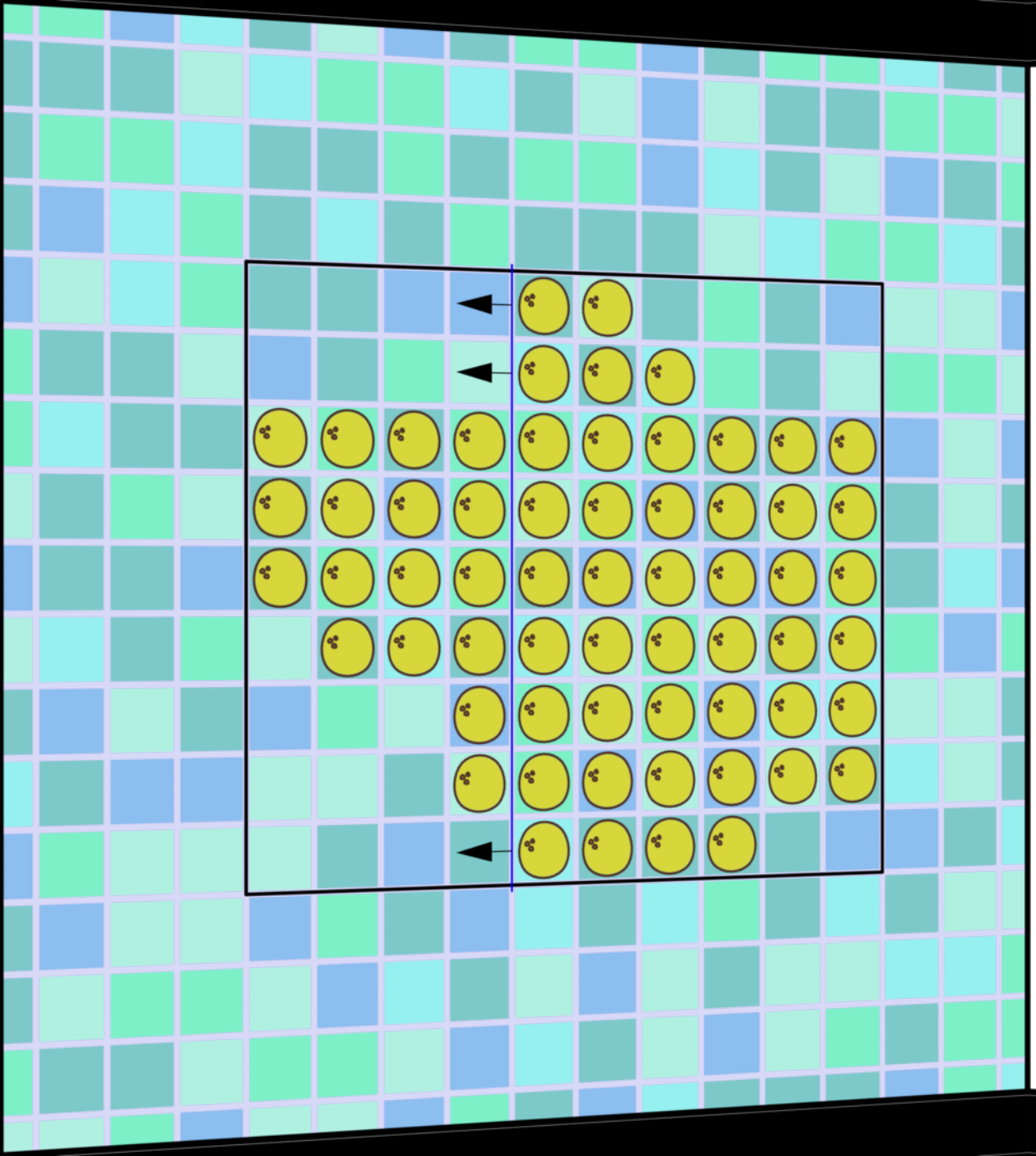


We can play a dual game on the antinuts!



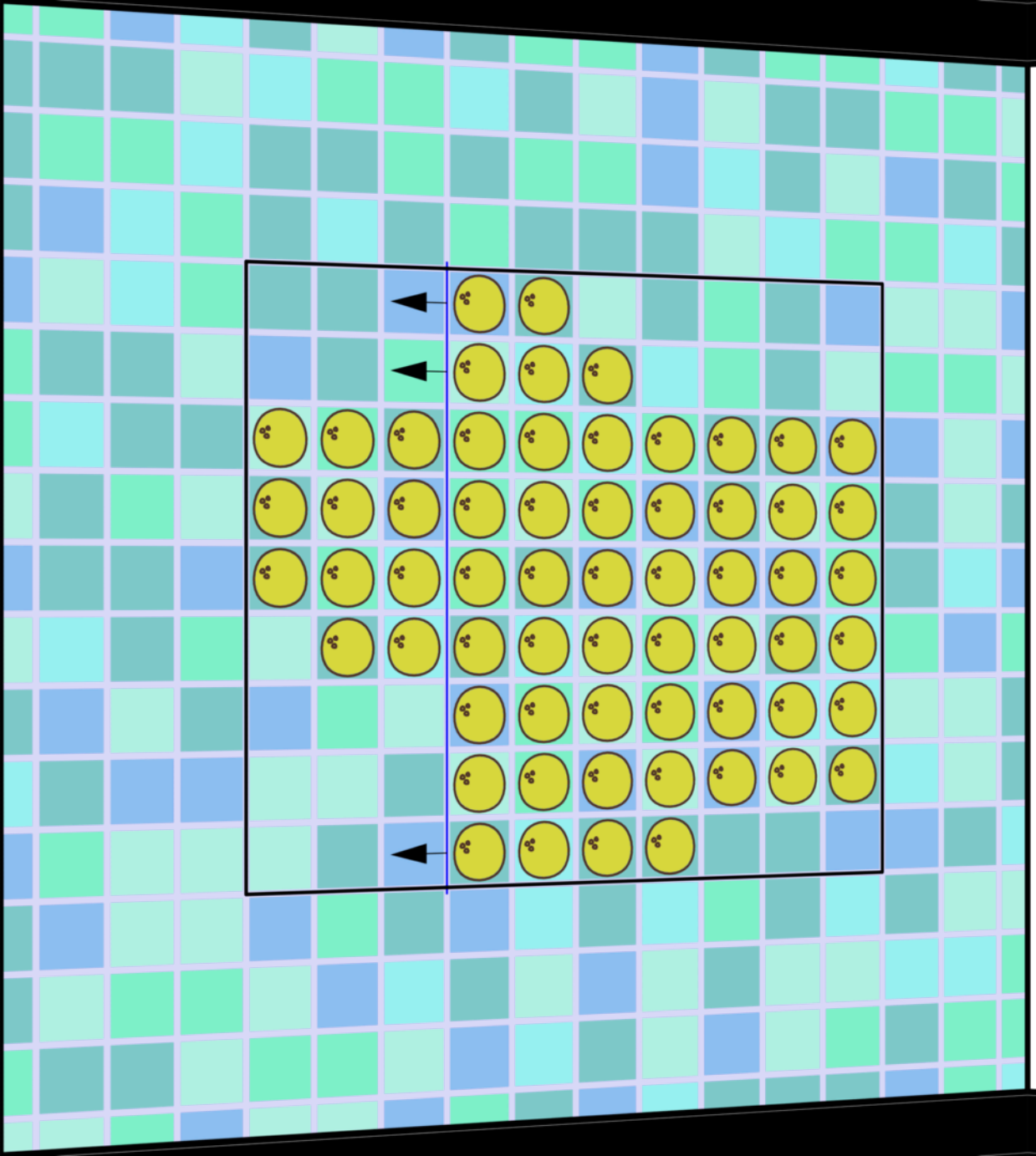
We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.



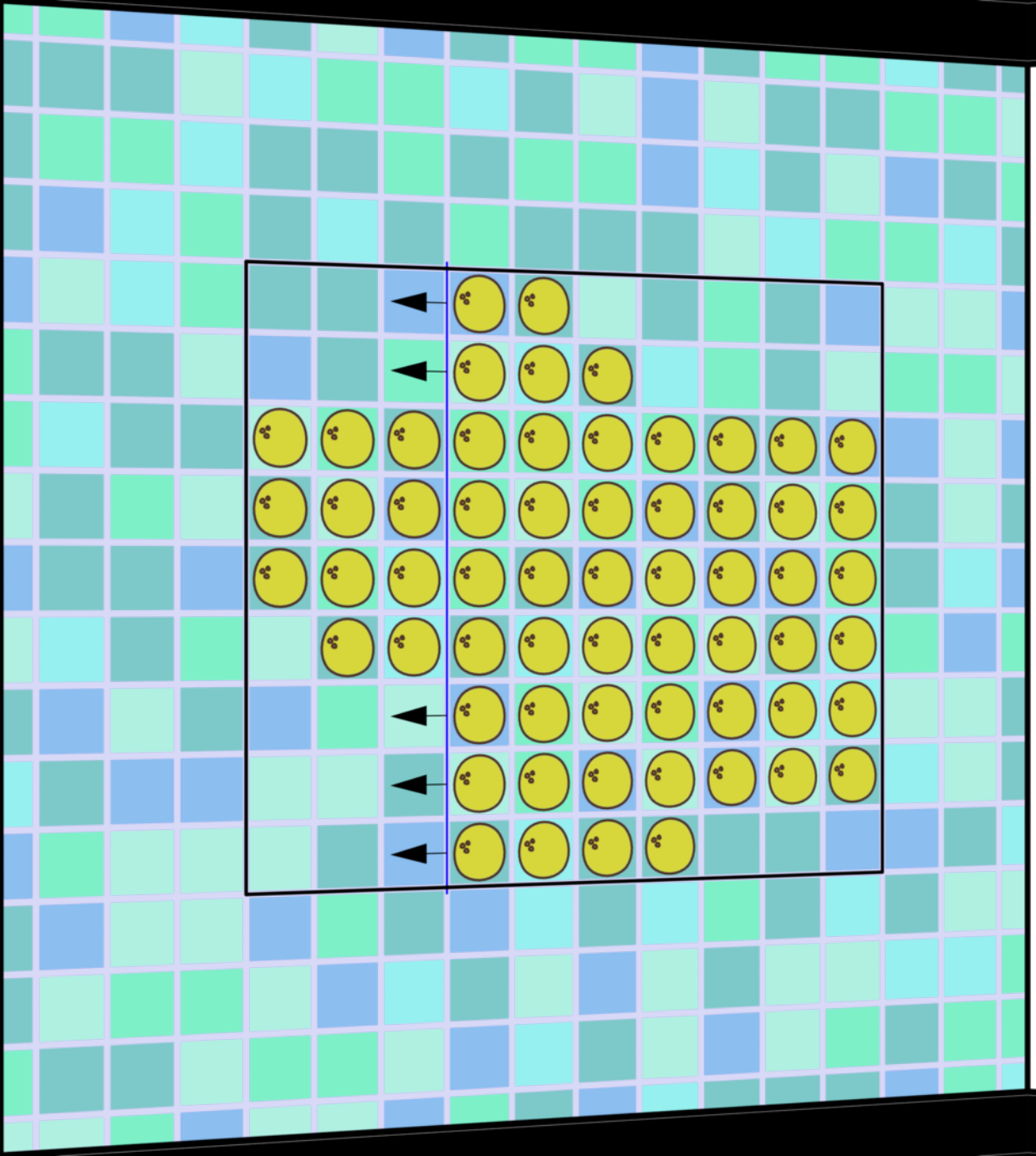
We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.



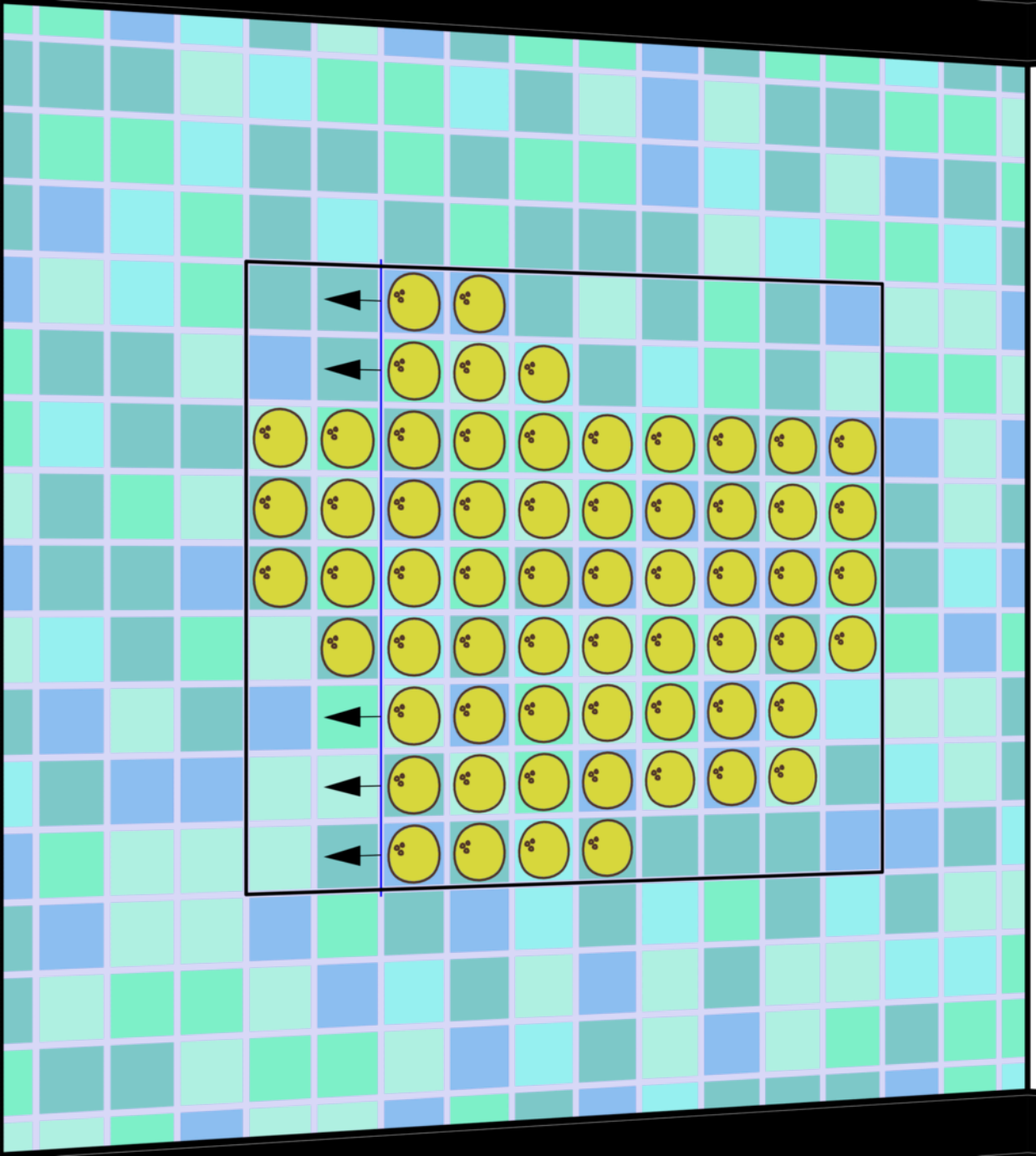
We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.



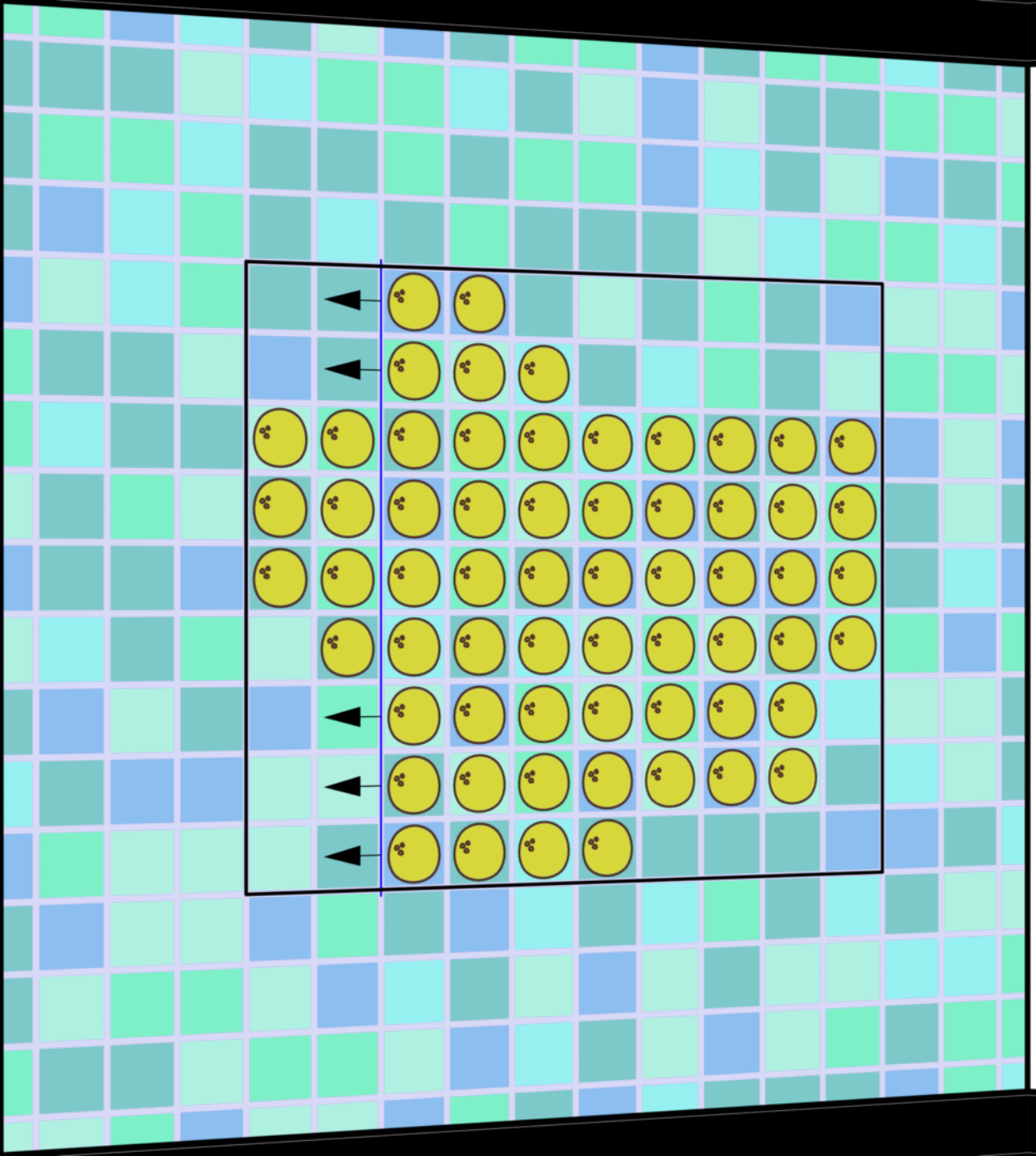
We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.



We can play a dual game on the antinuts!

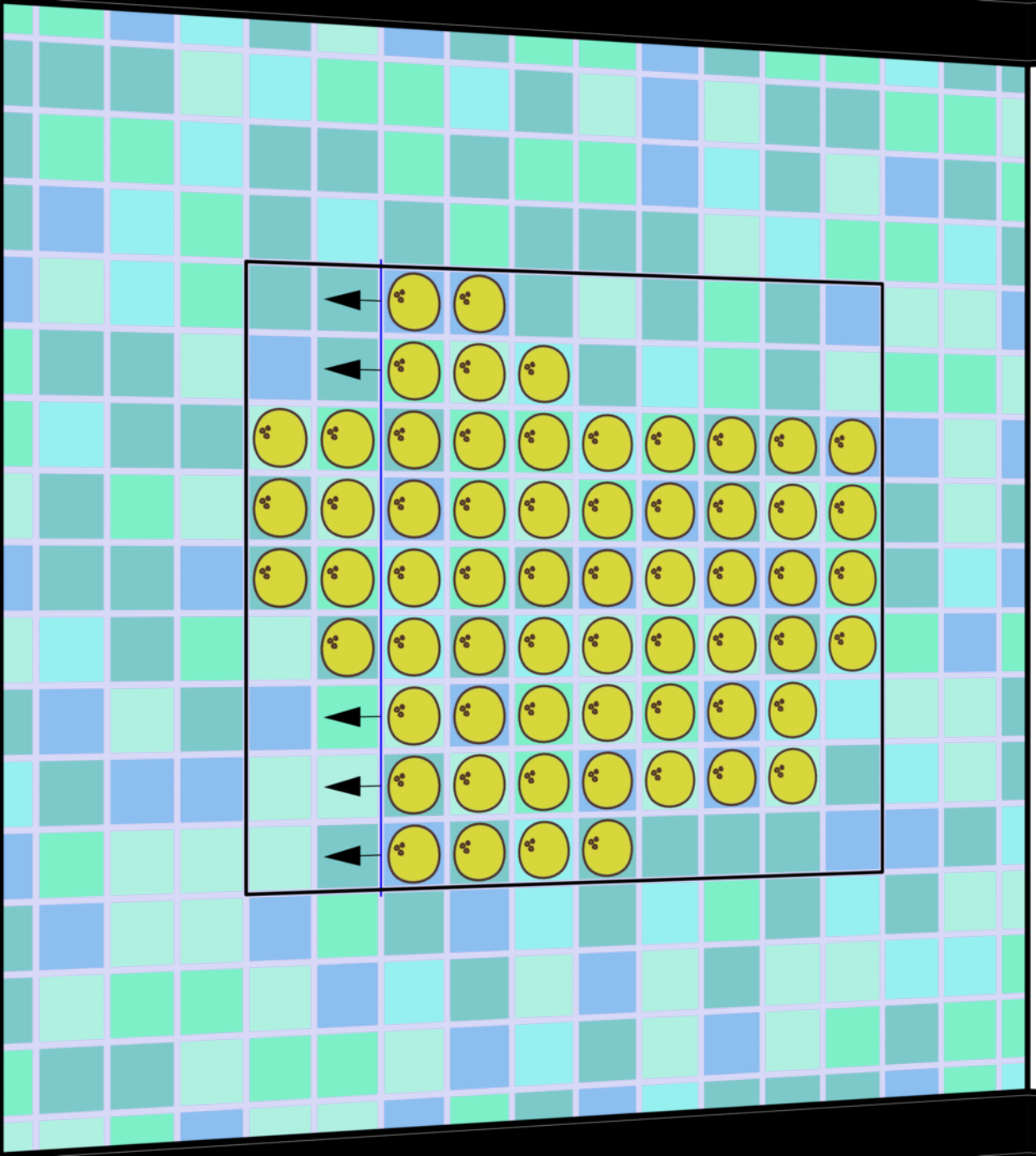
Pushes in the primal corresponds to *pulls* in the dual.



We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.

Most statements about the primal game have dual analogues.

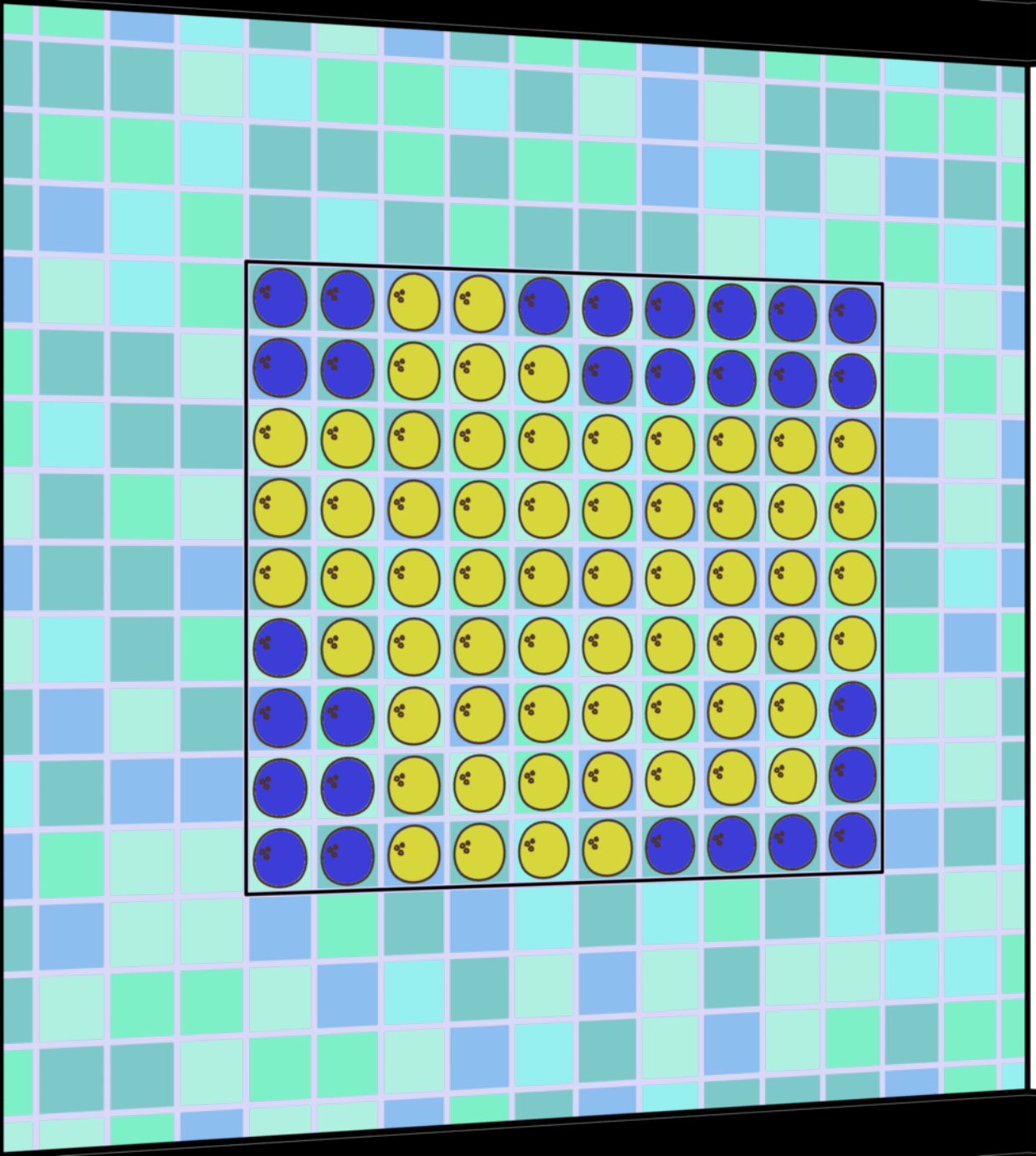


We can play a dual game on the antinuts!

Pushes in the primal corresponds to *pulls* in the dual.

Most statements about the primal game have dual analogues.

In particular, every achievable permutation on antinuts and *anti-antinuts* is *even*.

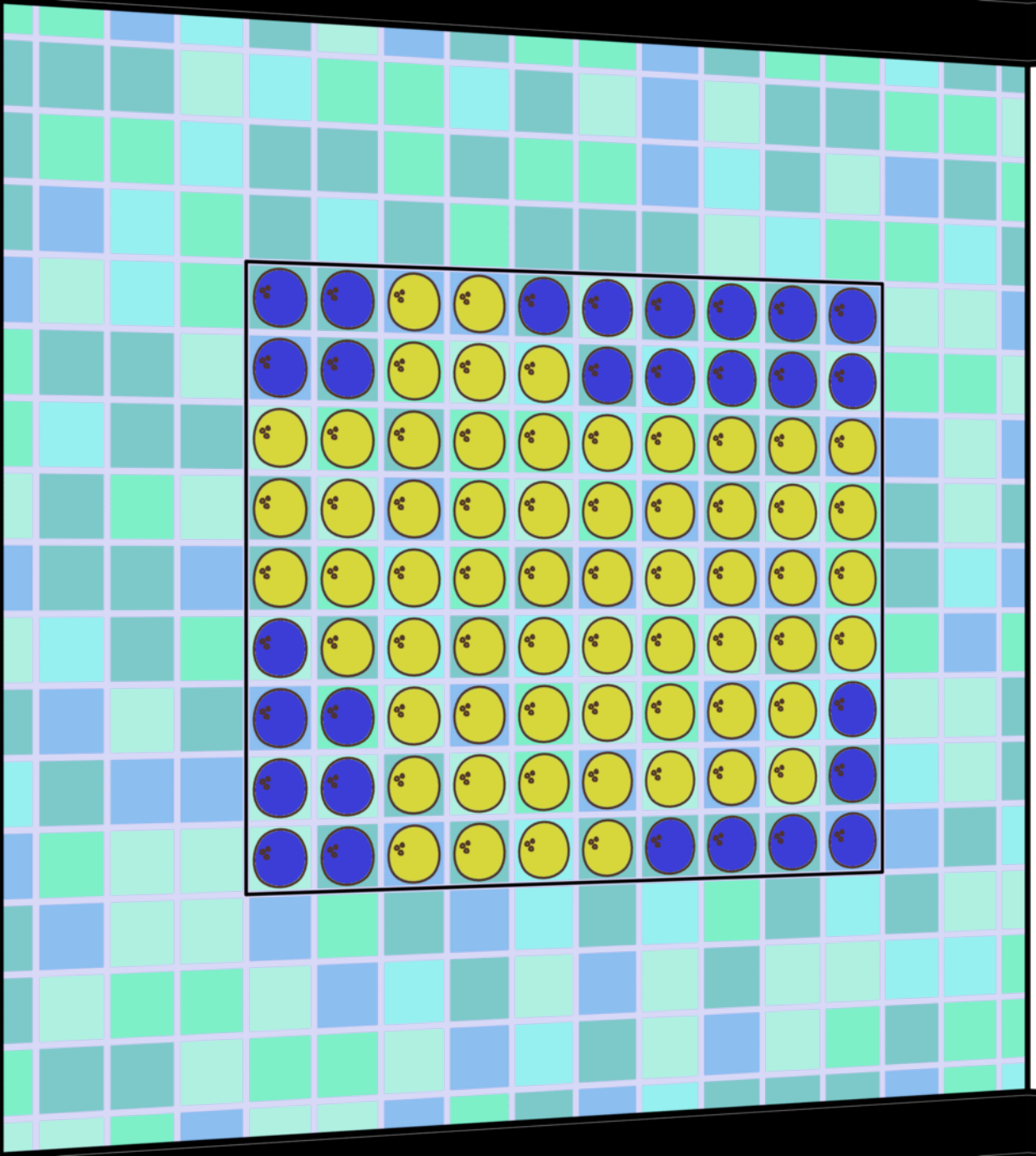


We can play a dual game on the antinuts!

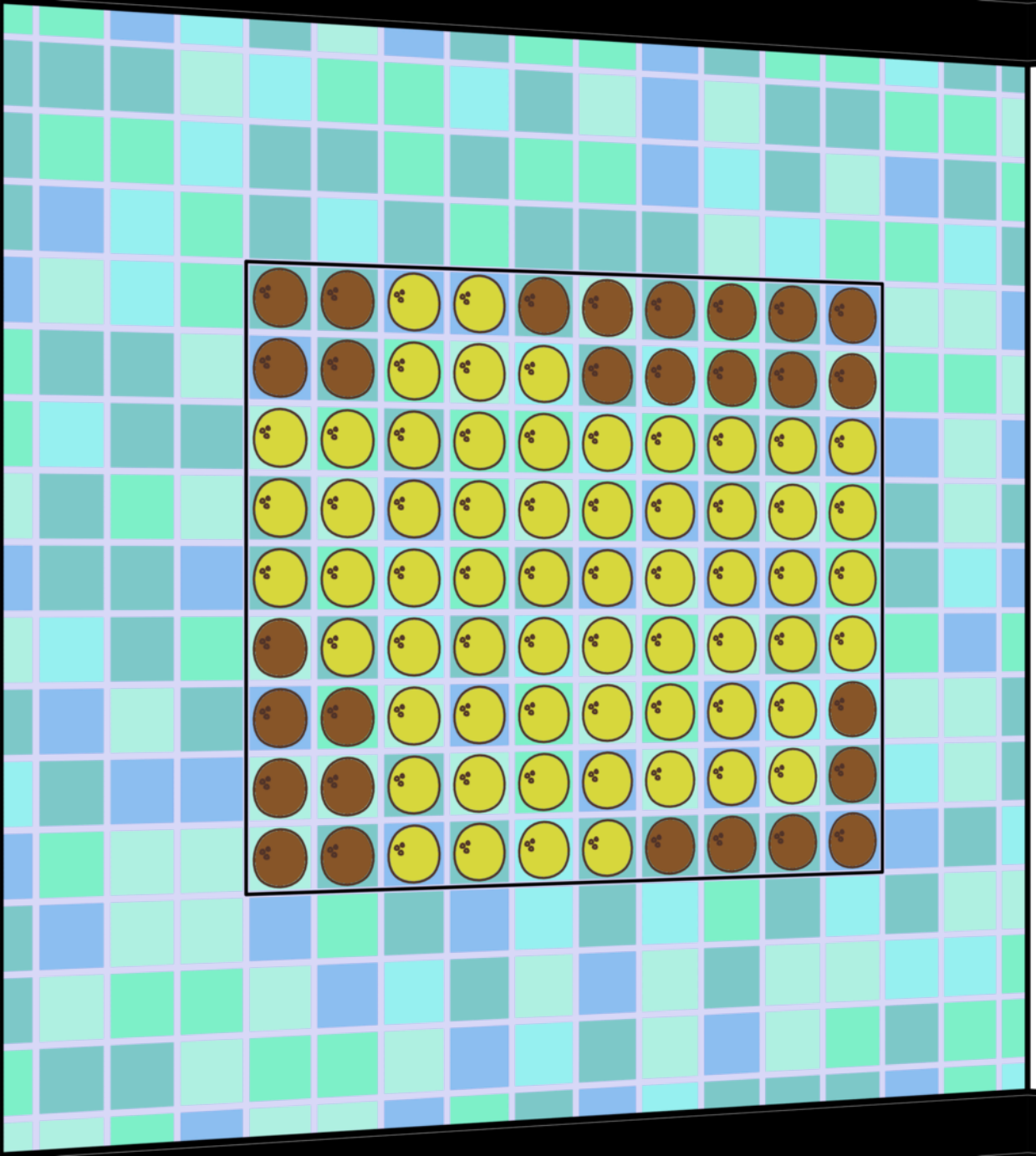
Pushes in the primal corresponds to *pulls* in the dual.

Most statements about the primal game have dual analogues.

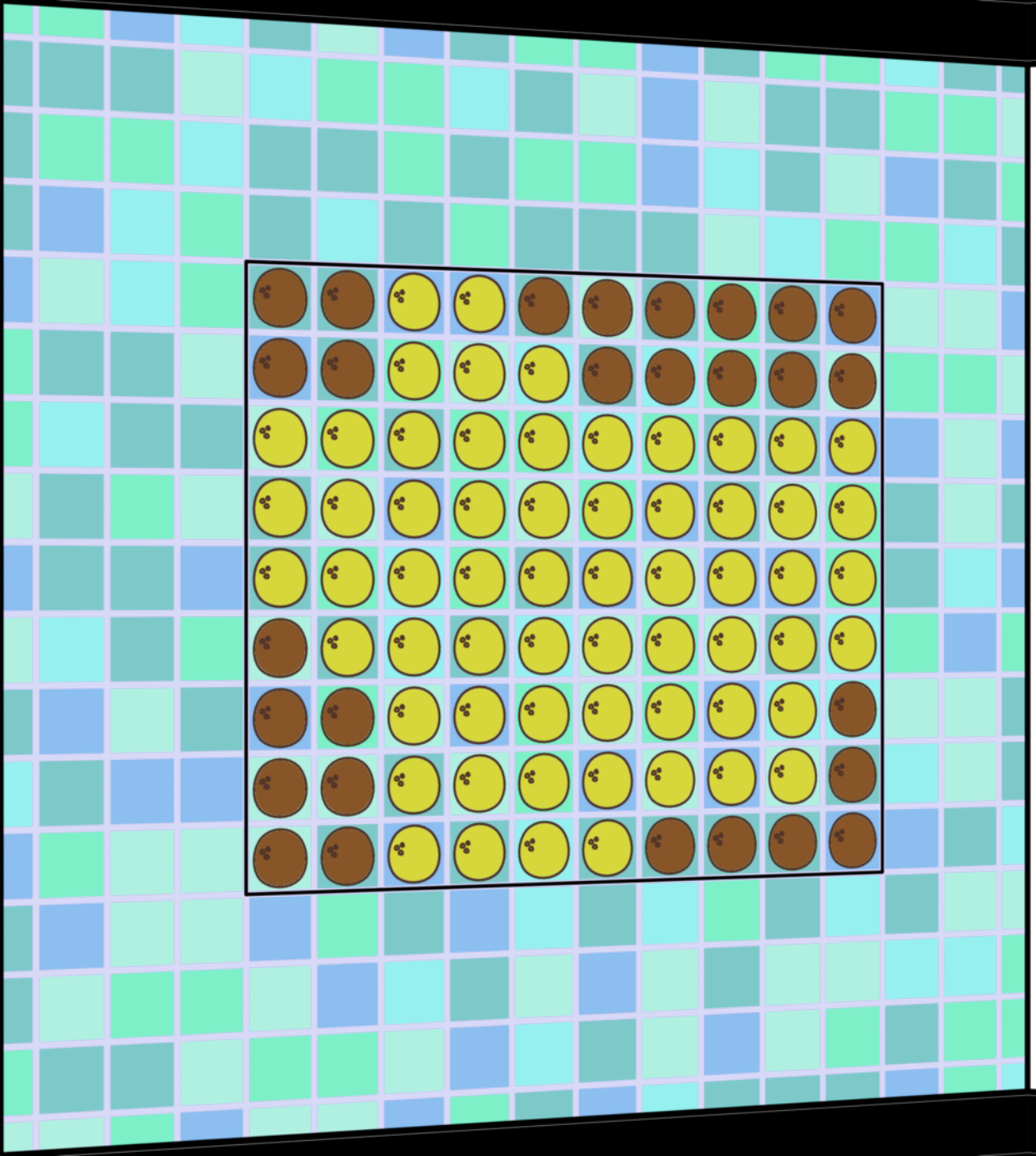
In particular, every achievable permutation on antinuts and *anti-antinuts* is *even*.



Of course, anti-antinuts are just our original coconuts...

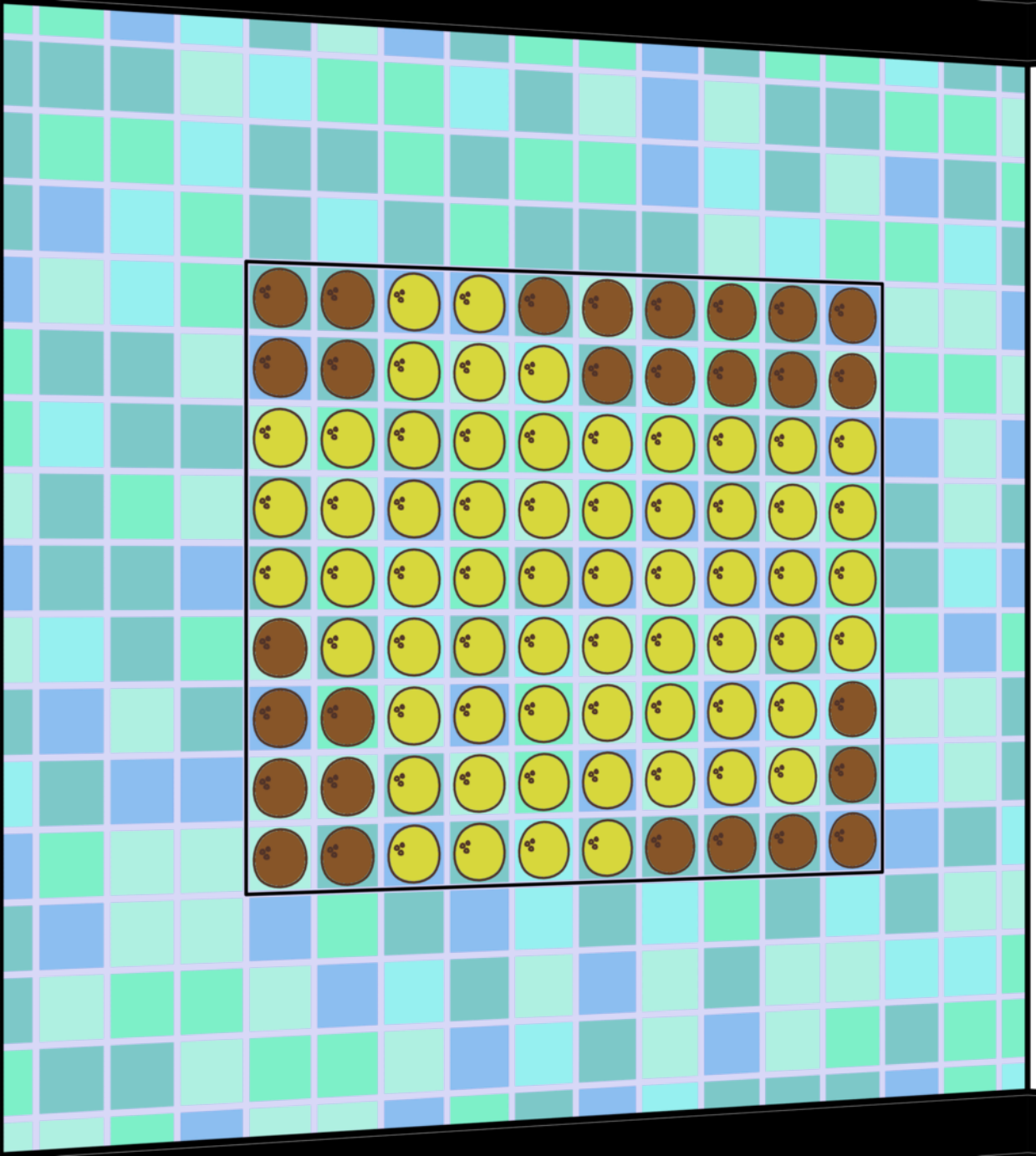


Of course, anti-antinuts are just our original coconuts...



Of course, anti-antinuts are just our original coconuts...

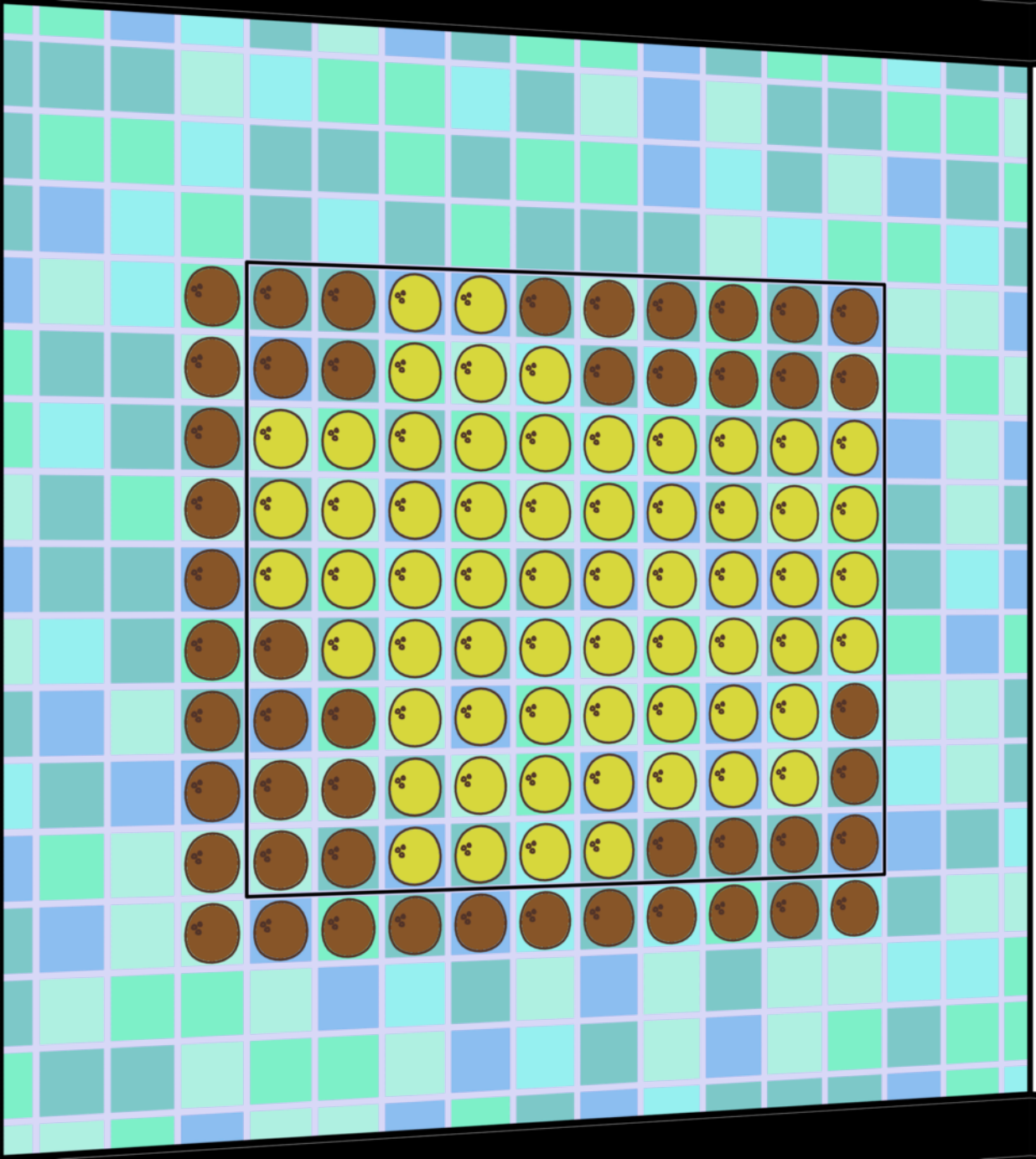
...but not *all* of our original coconuts!



Of course, anti-antinuts are just our original coconuts...

...but not *all* of our original coconuts!

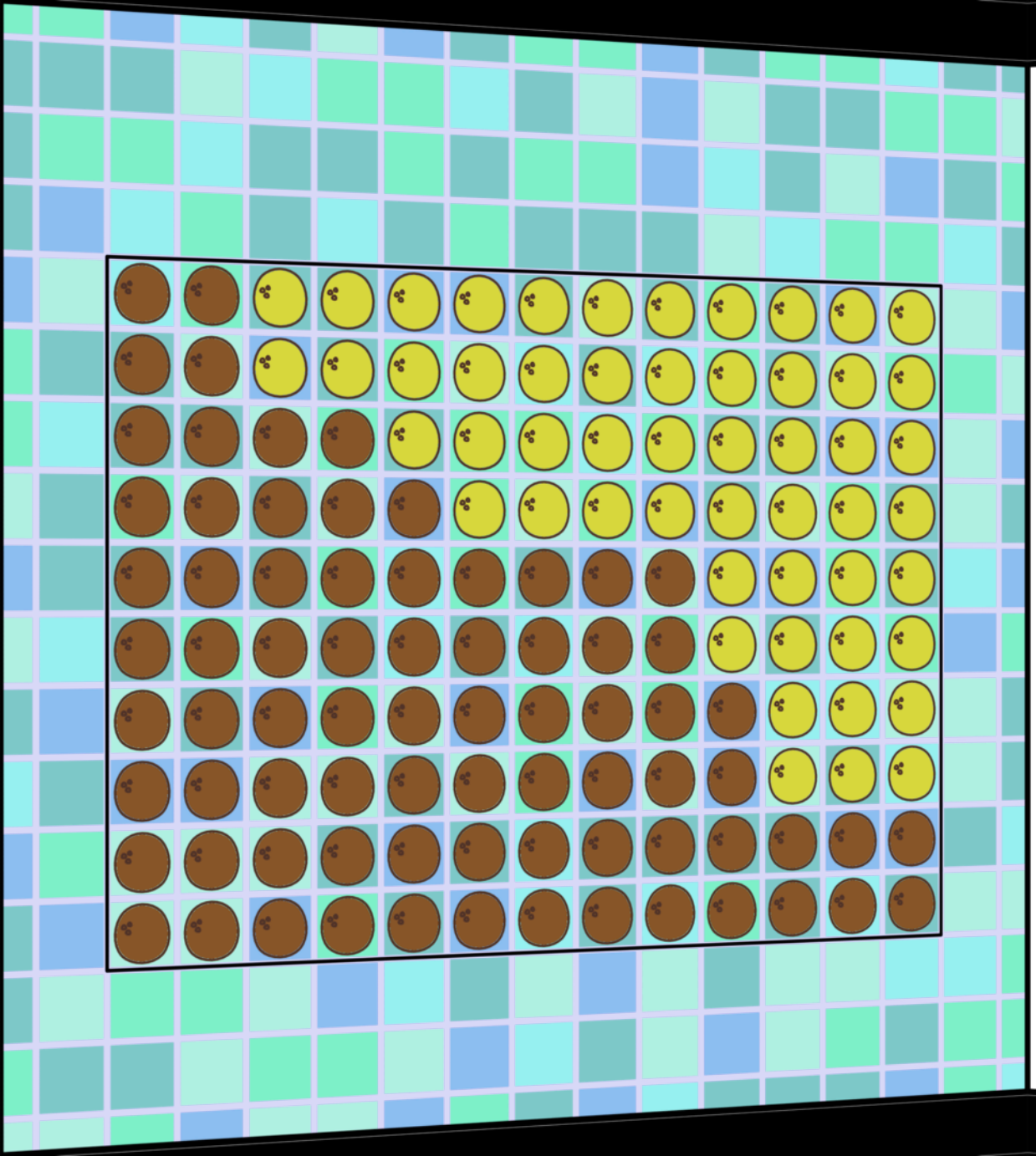
There must have been at least one full row and column outside the bounding box of the dual game.



Of course, anti-antinuts are just our original coconuts...

...but not *all* of our original coconuts!

There must have been at least one full row and column outside the bounding box of the dual game.



Proof by induction.

Proof by induction.

Proof by induction.

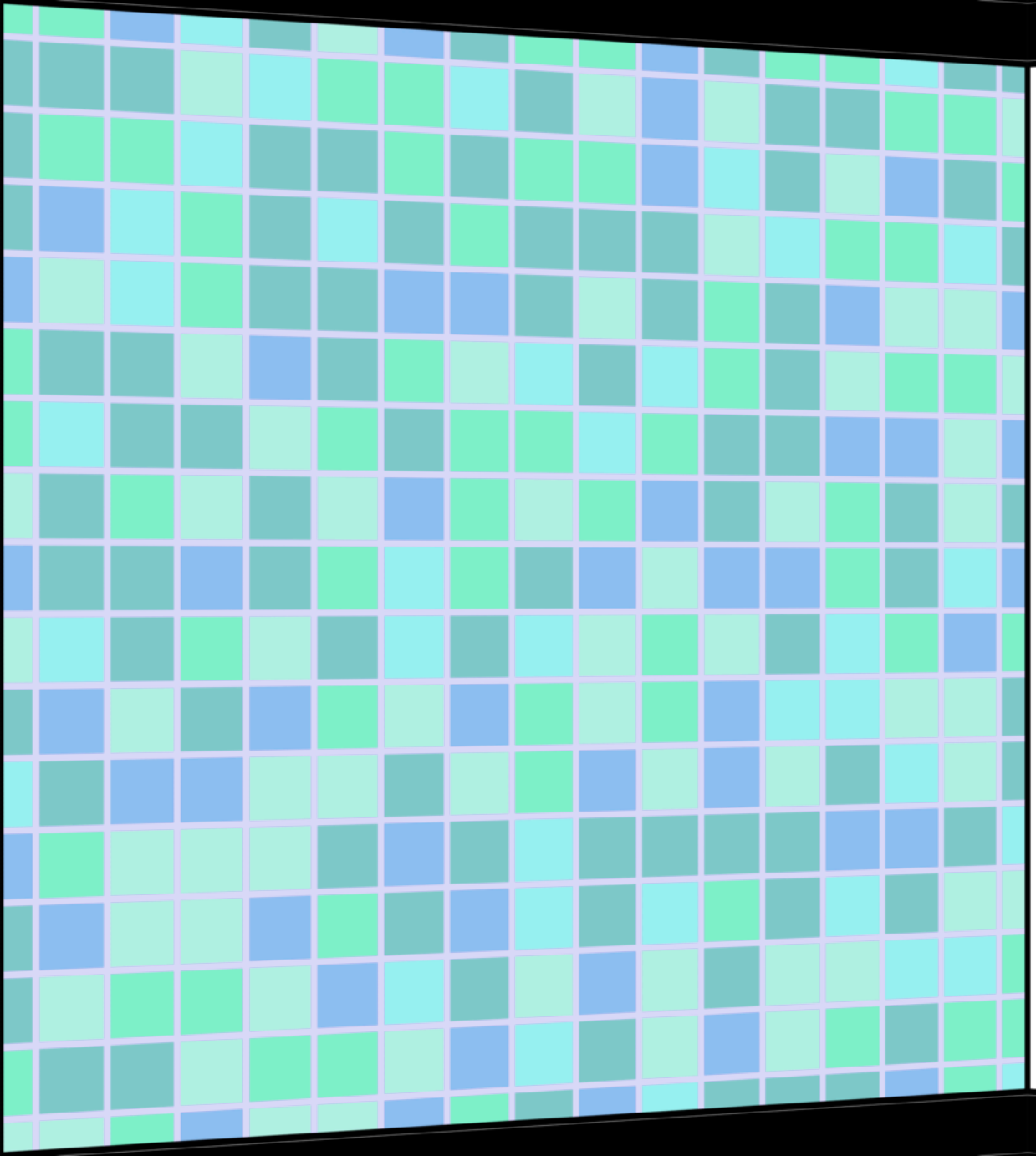
Proof by induction.

Proof by induction.

Proof by induction.

Theorem 3. Every even permutation can be realized unless:

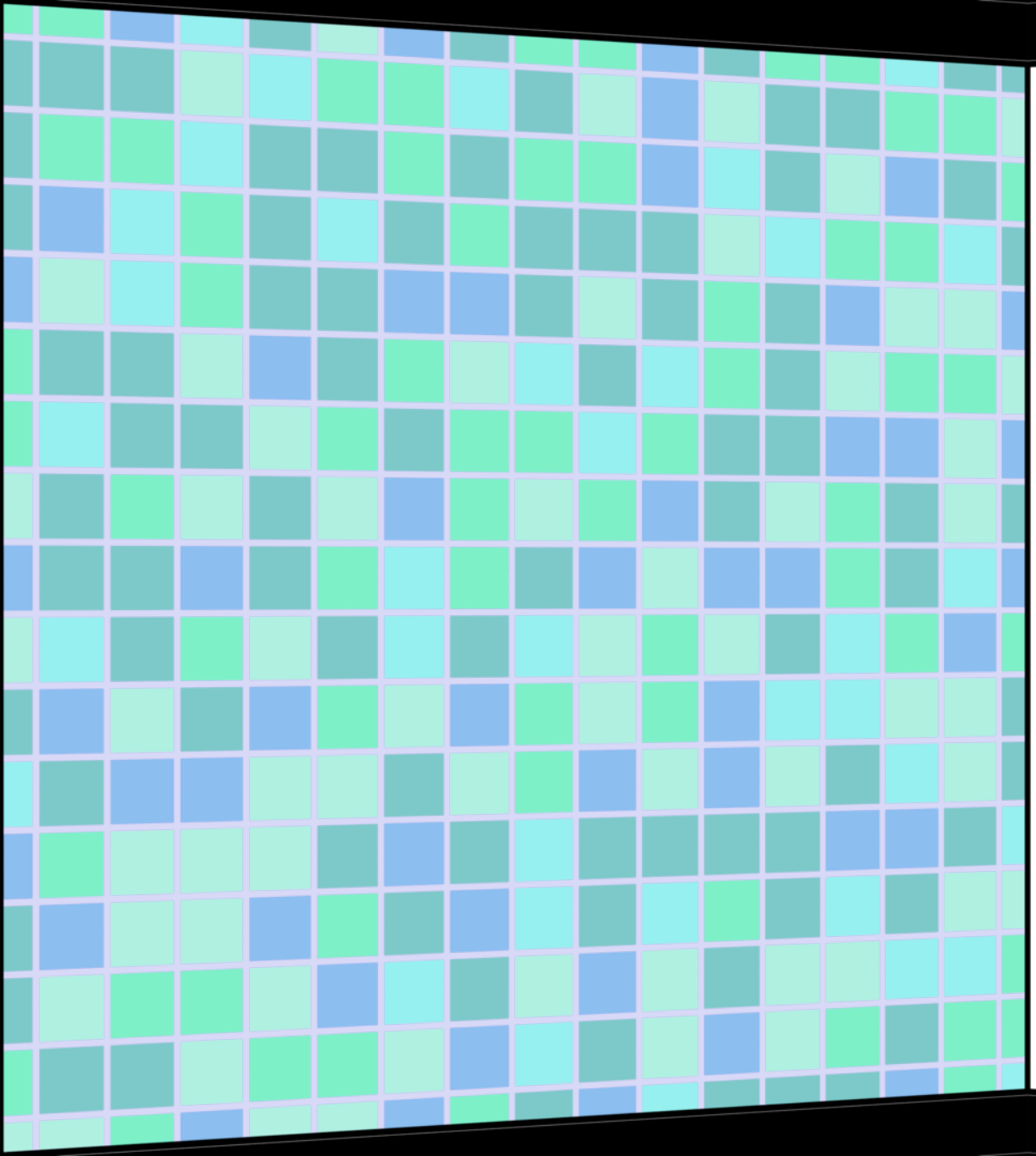
- there is a *core*; or
- it has this shape: 



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.



Theorem 3. Every even permutation can be realized unless:

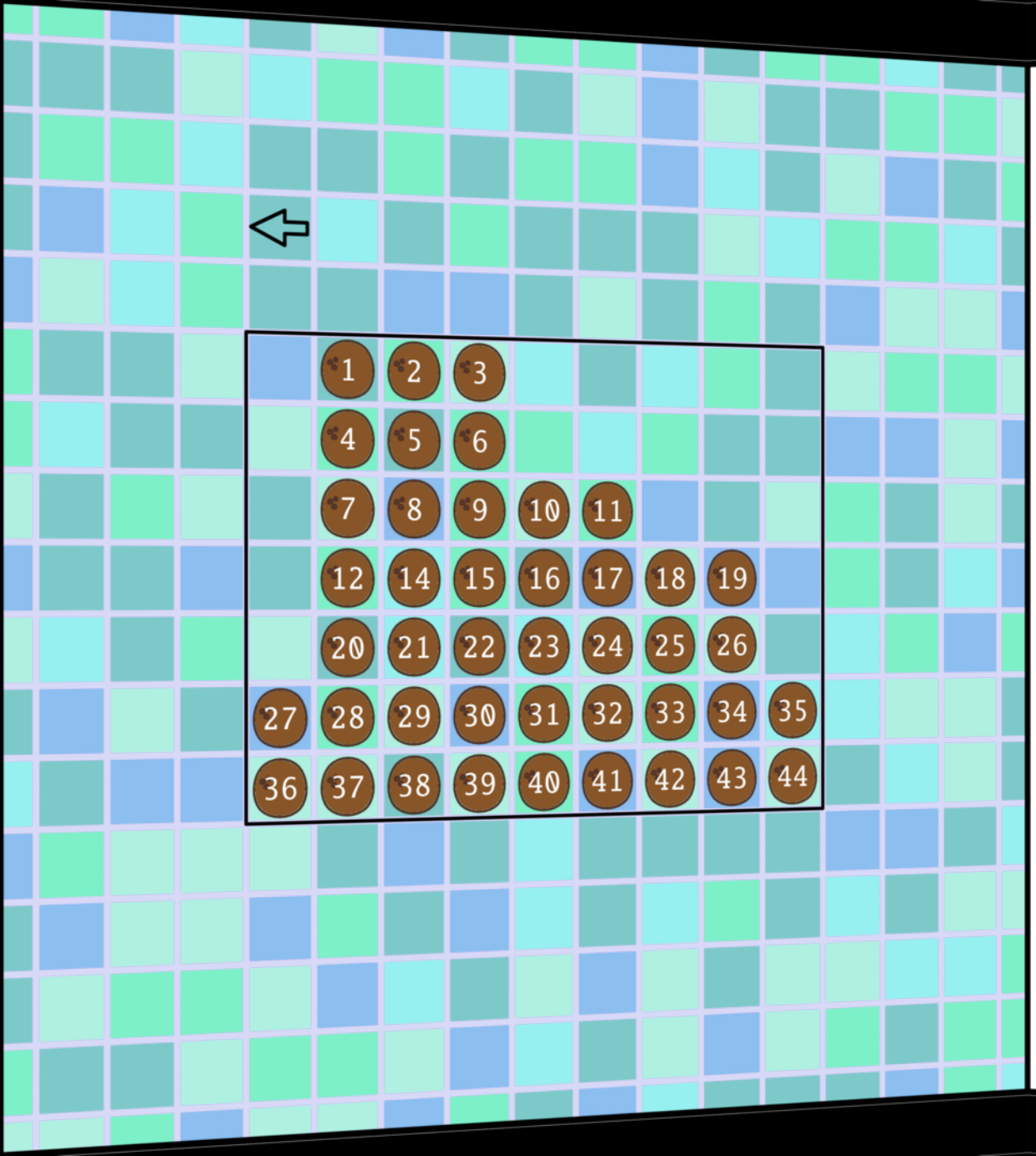
- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

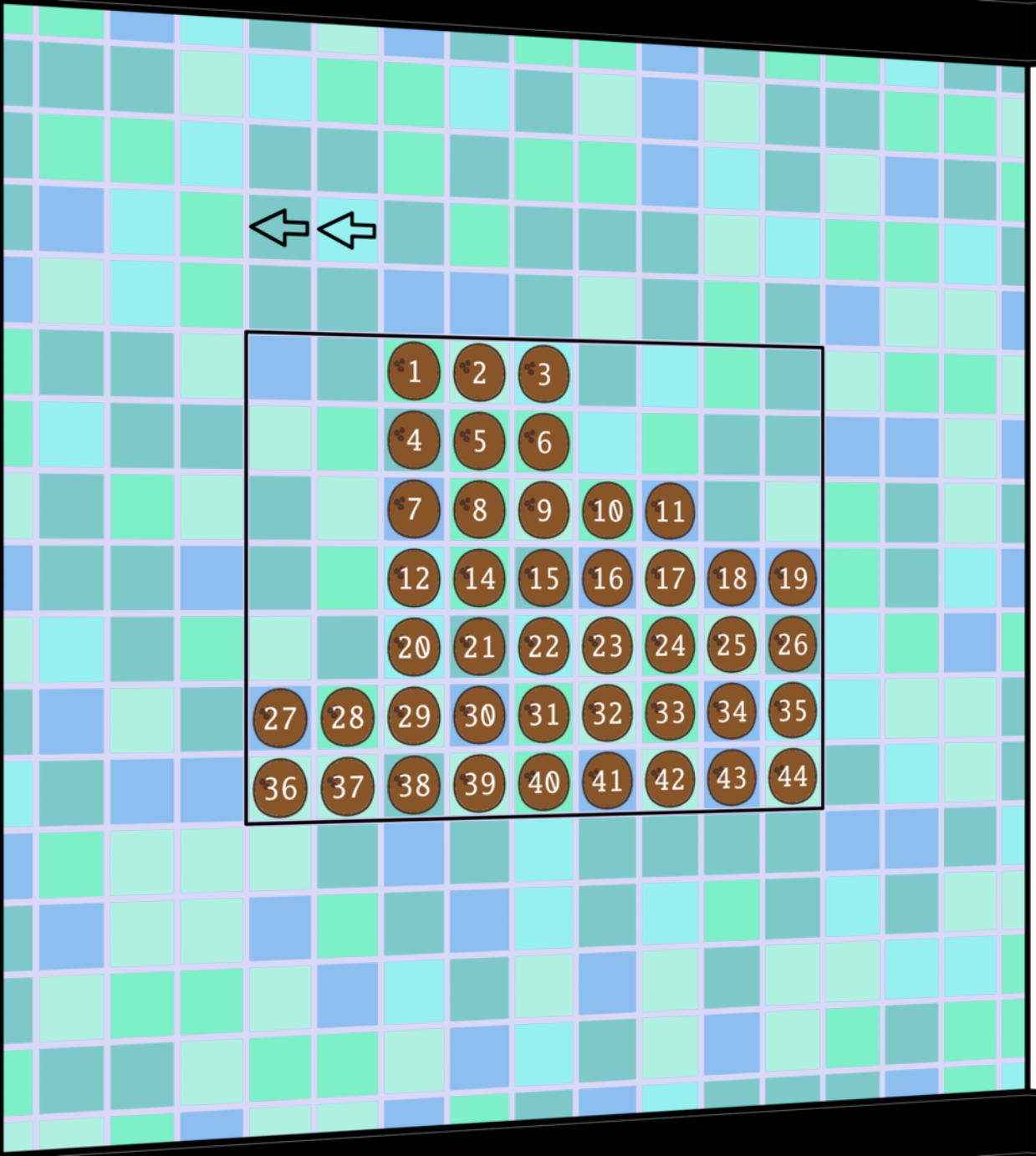
Proof idea. Cover by cycles that reach all coconuts.



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

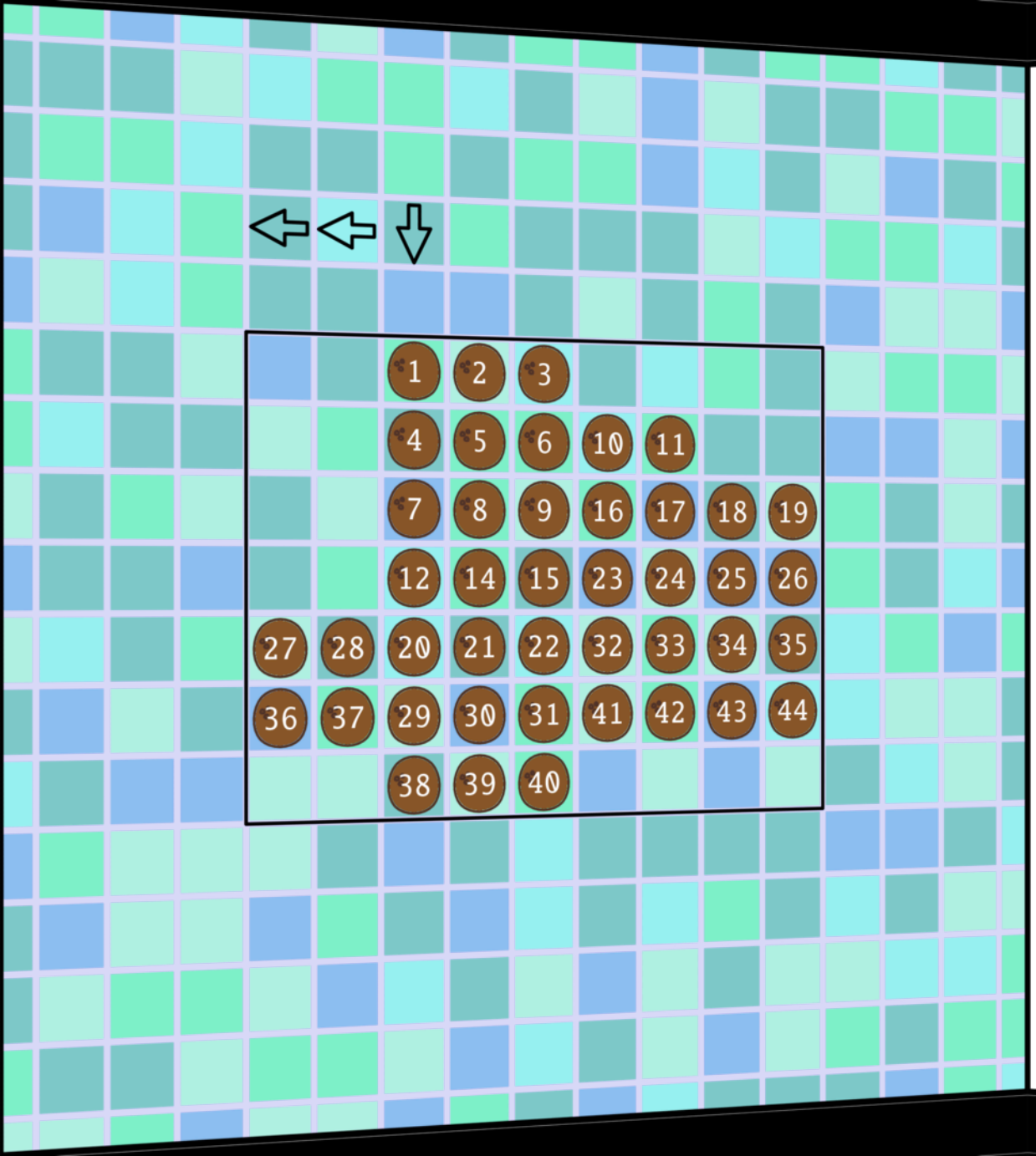
Proof idea. Cover by cycles that reach all coconuts.



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

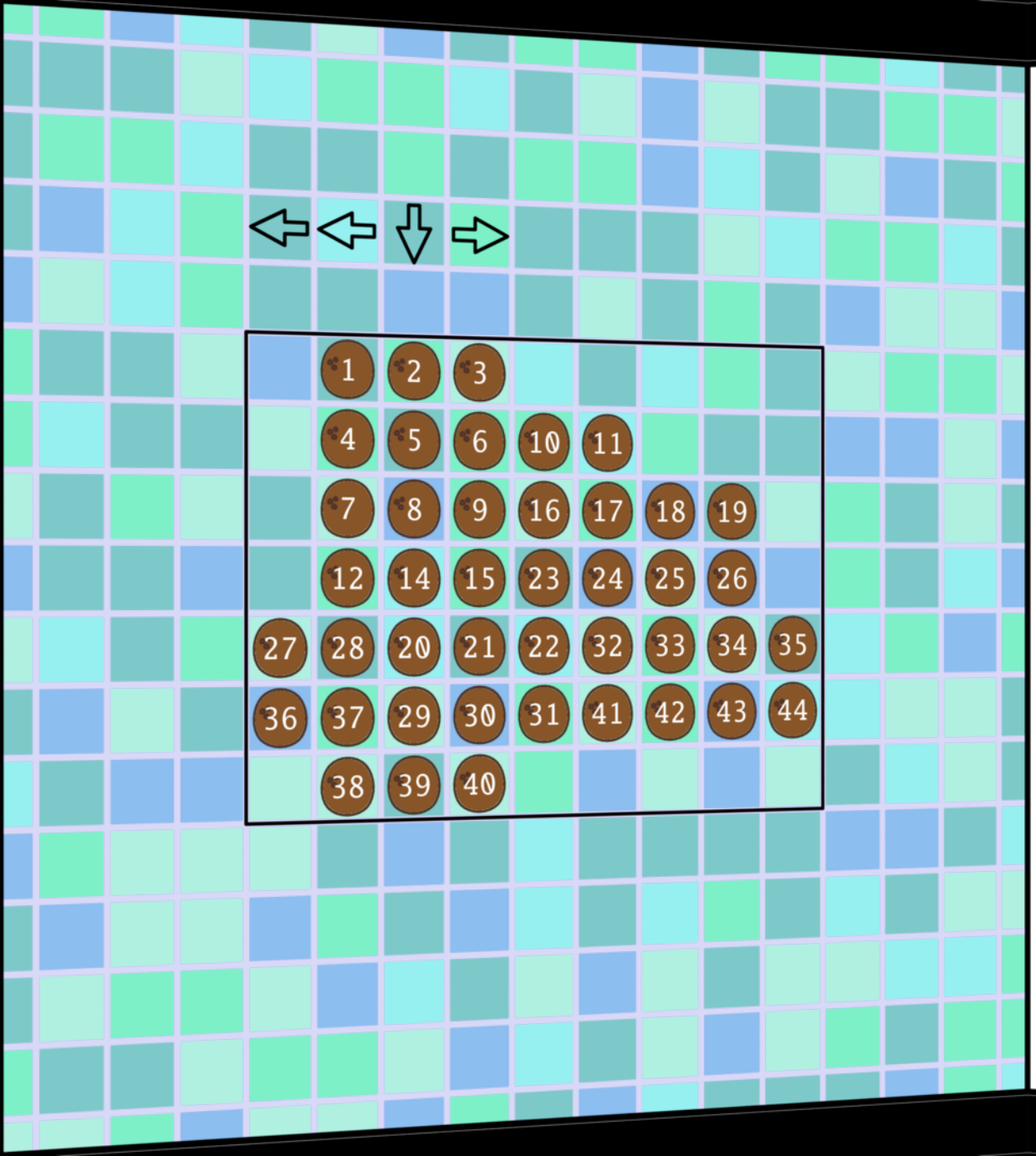
Proof idea. Cover by cycles that reach all coconuts.



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

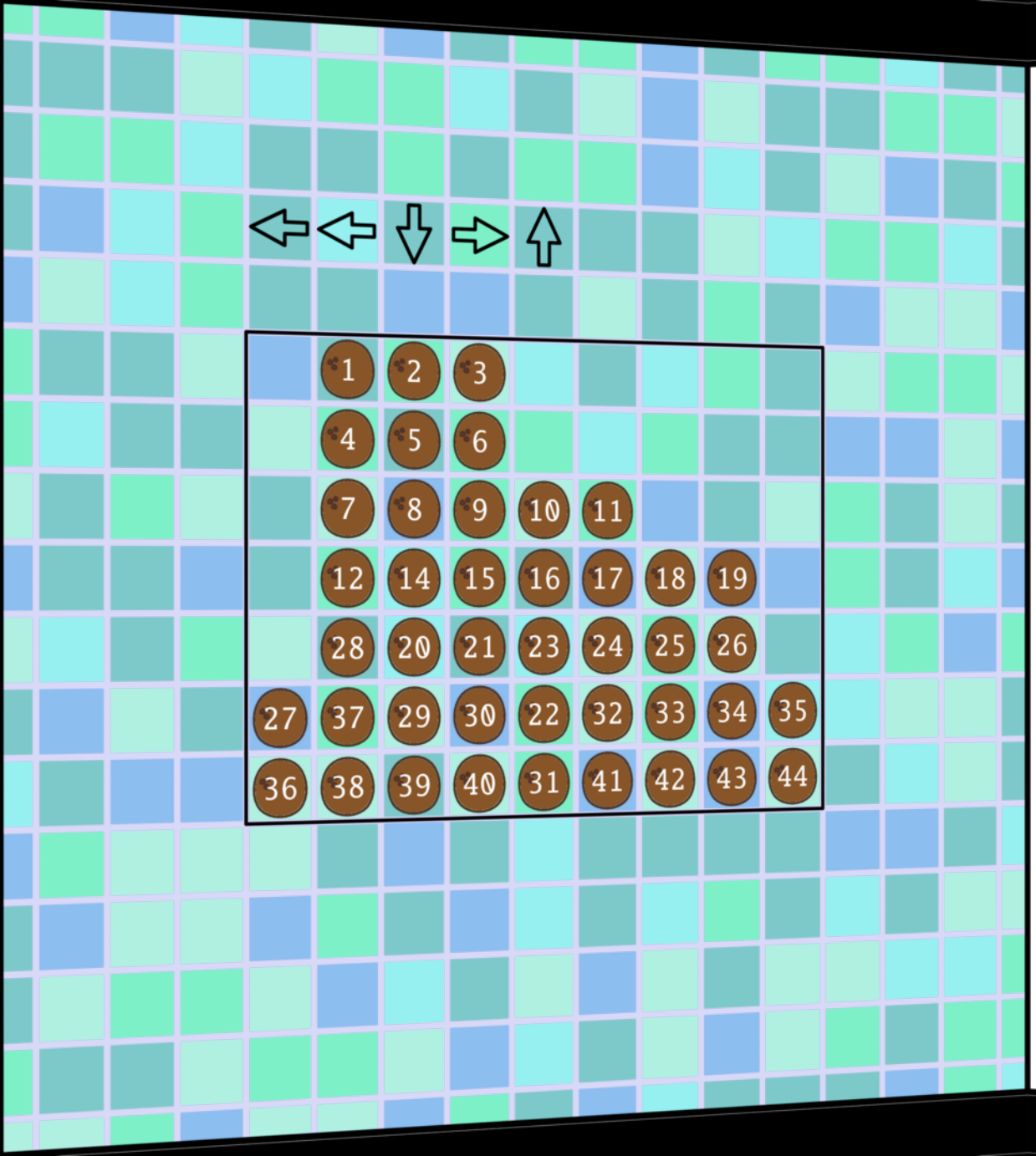
Proof idea. Cover by cycles that reach all coconuts.



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

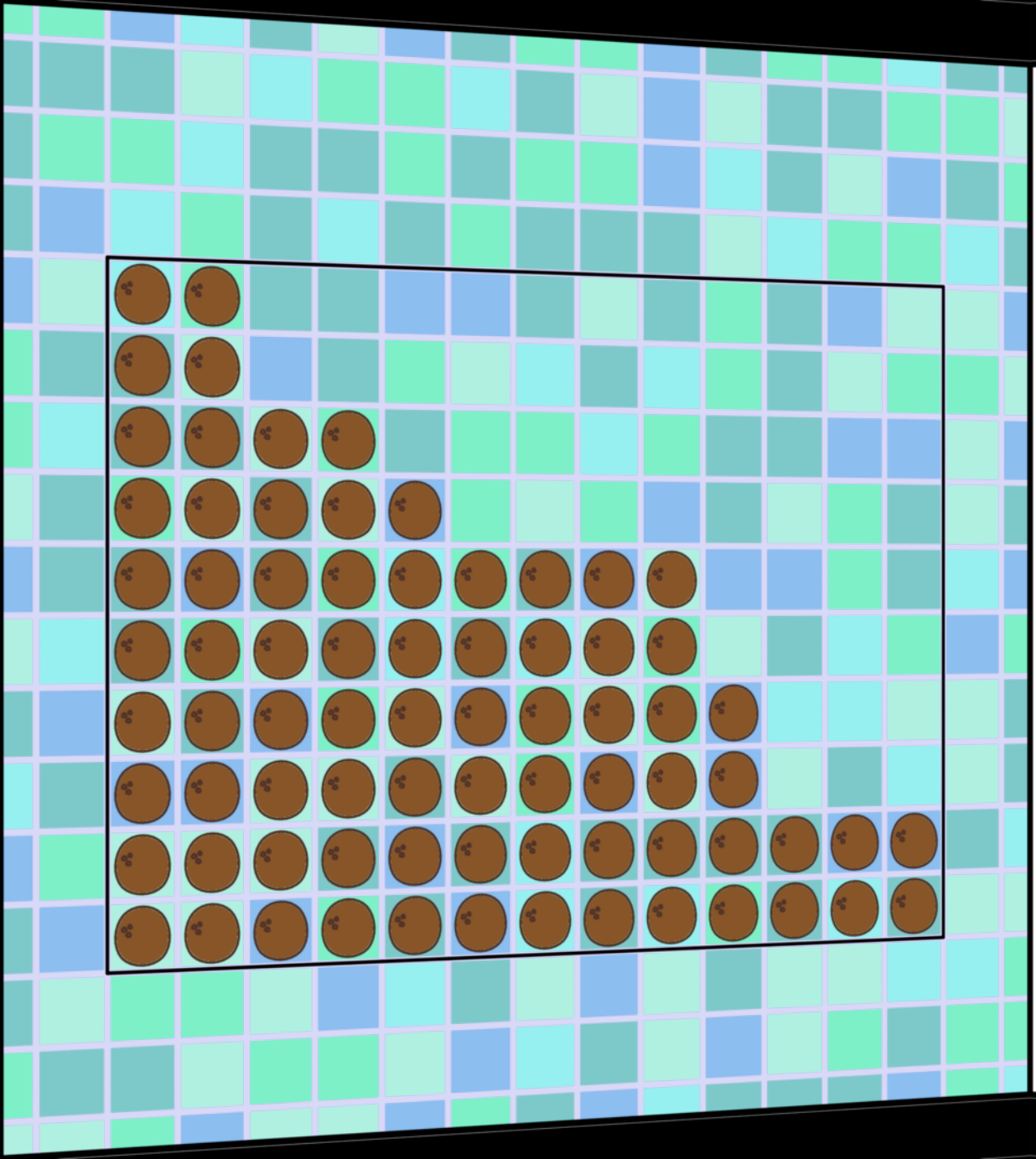
Proof idea. Cover by cycles that reach all coconuts.

Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

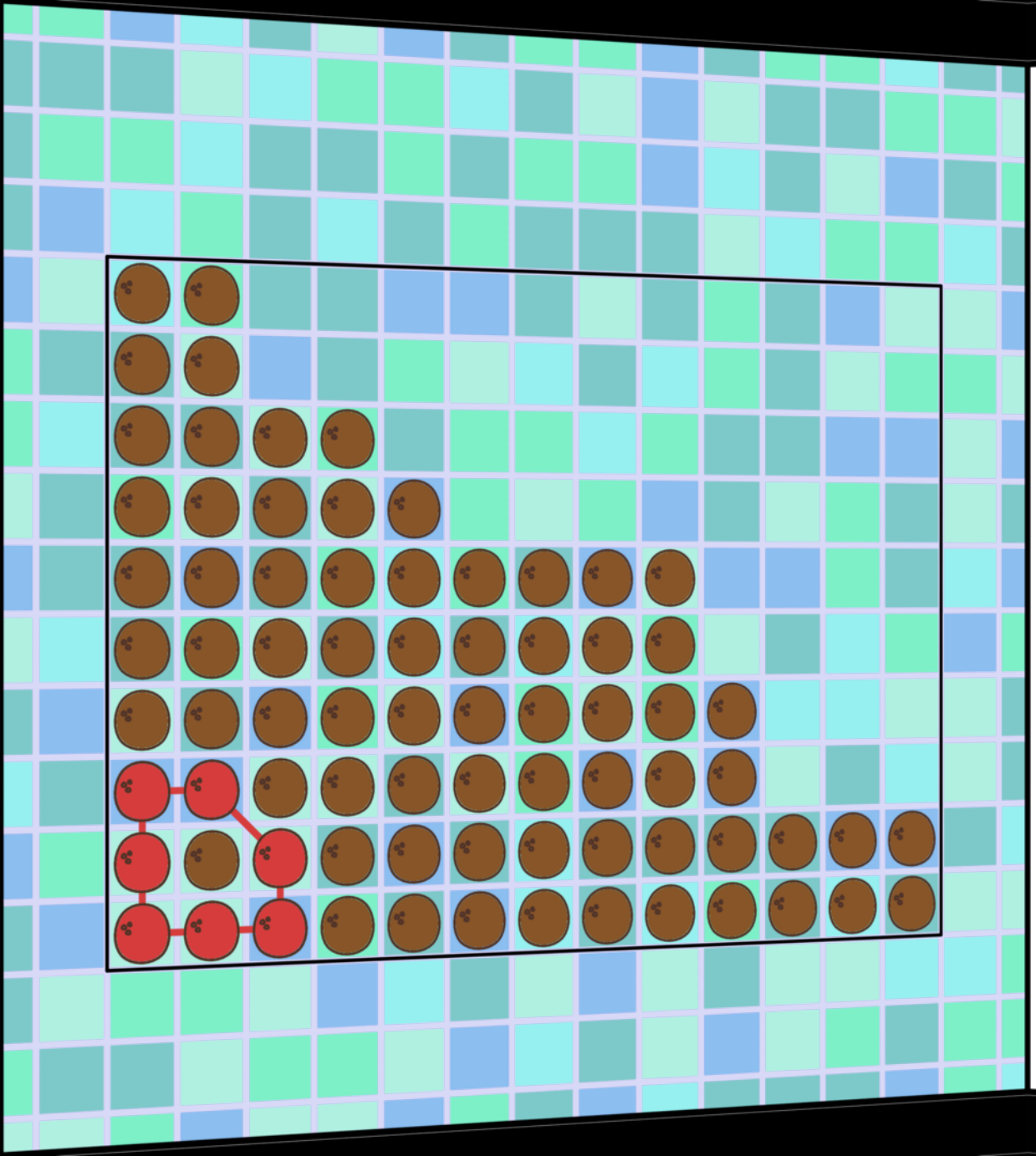


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

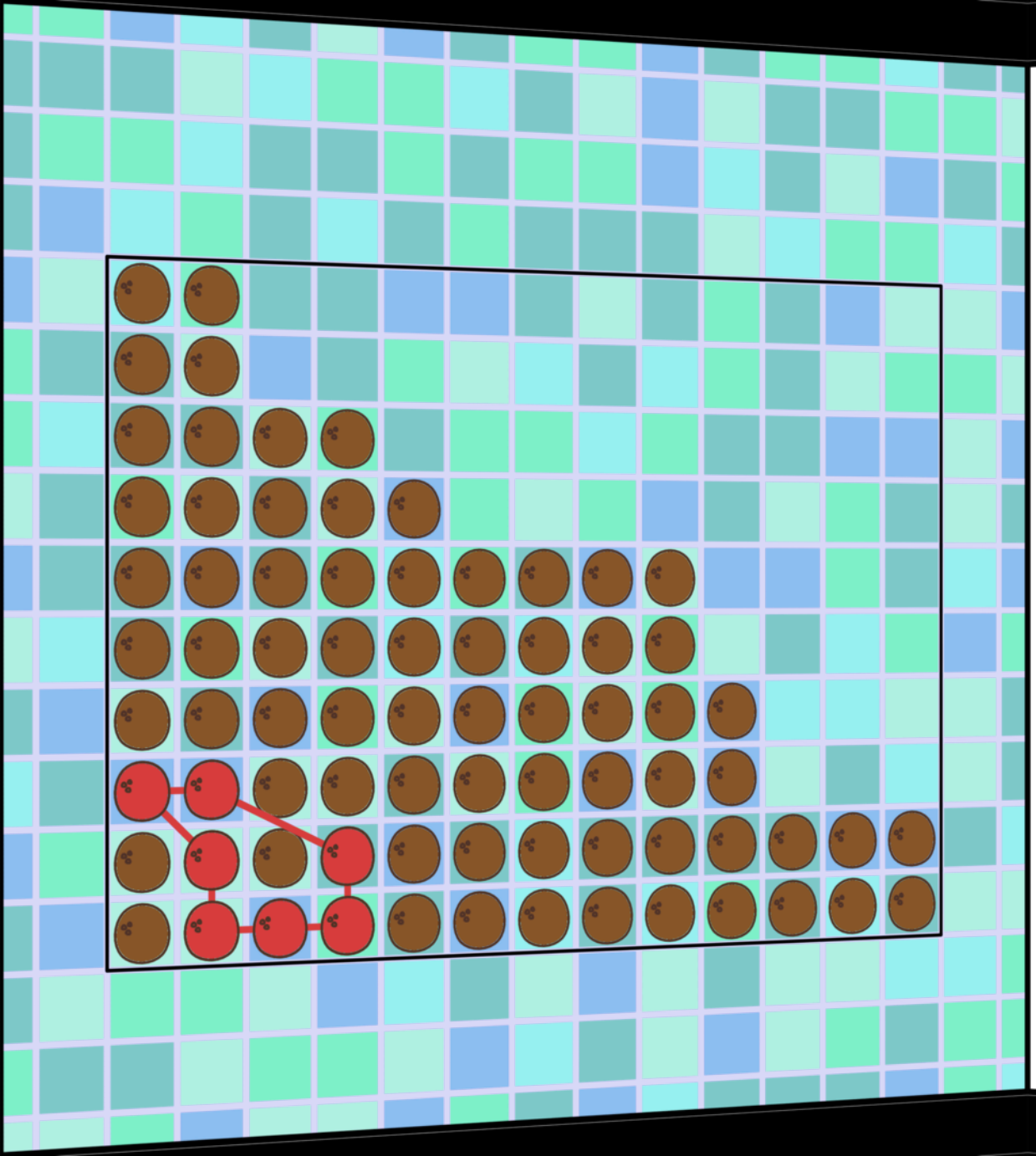


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

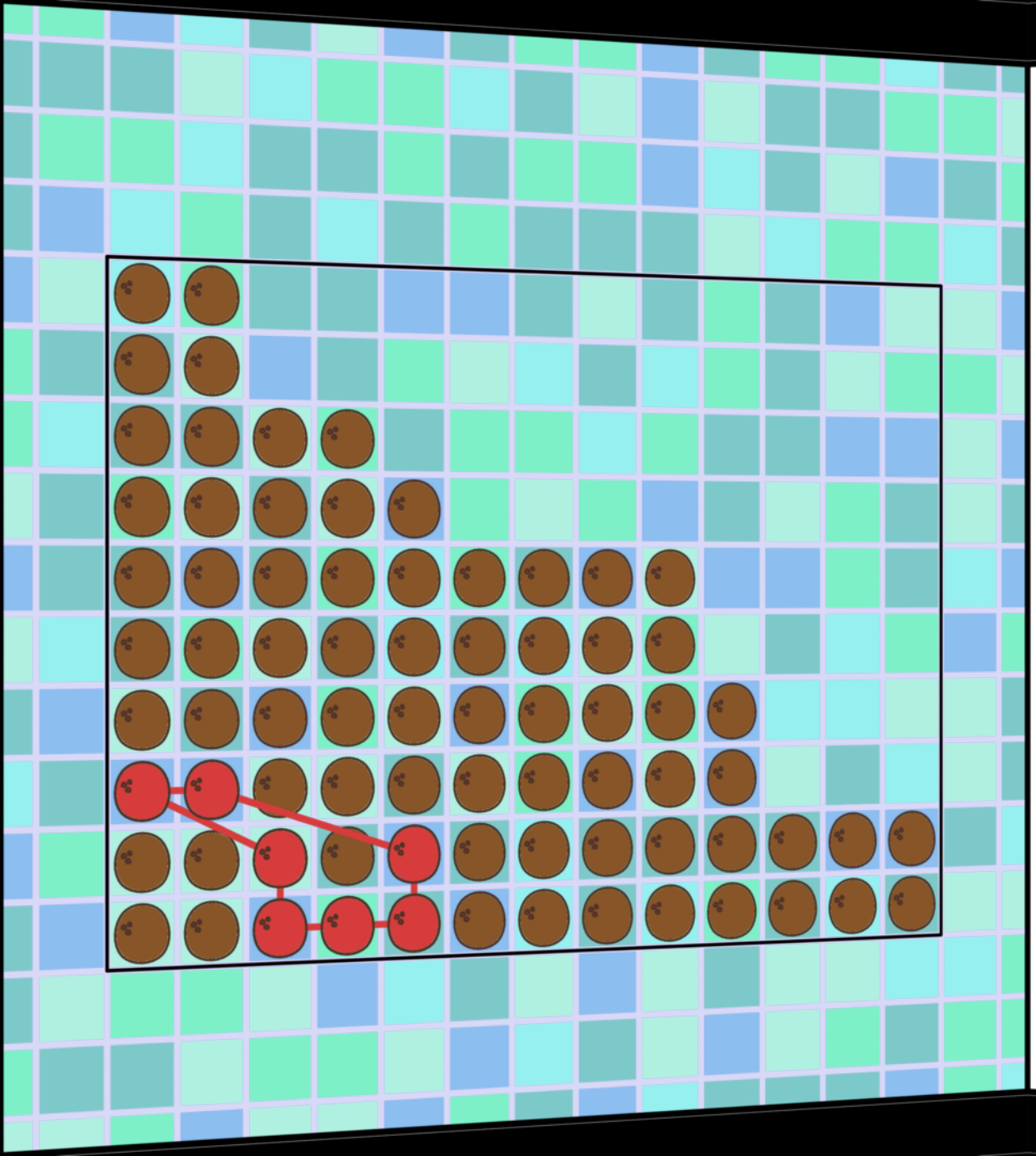


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

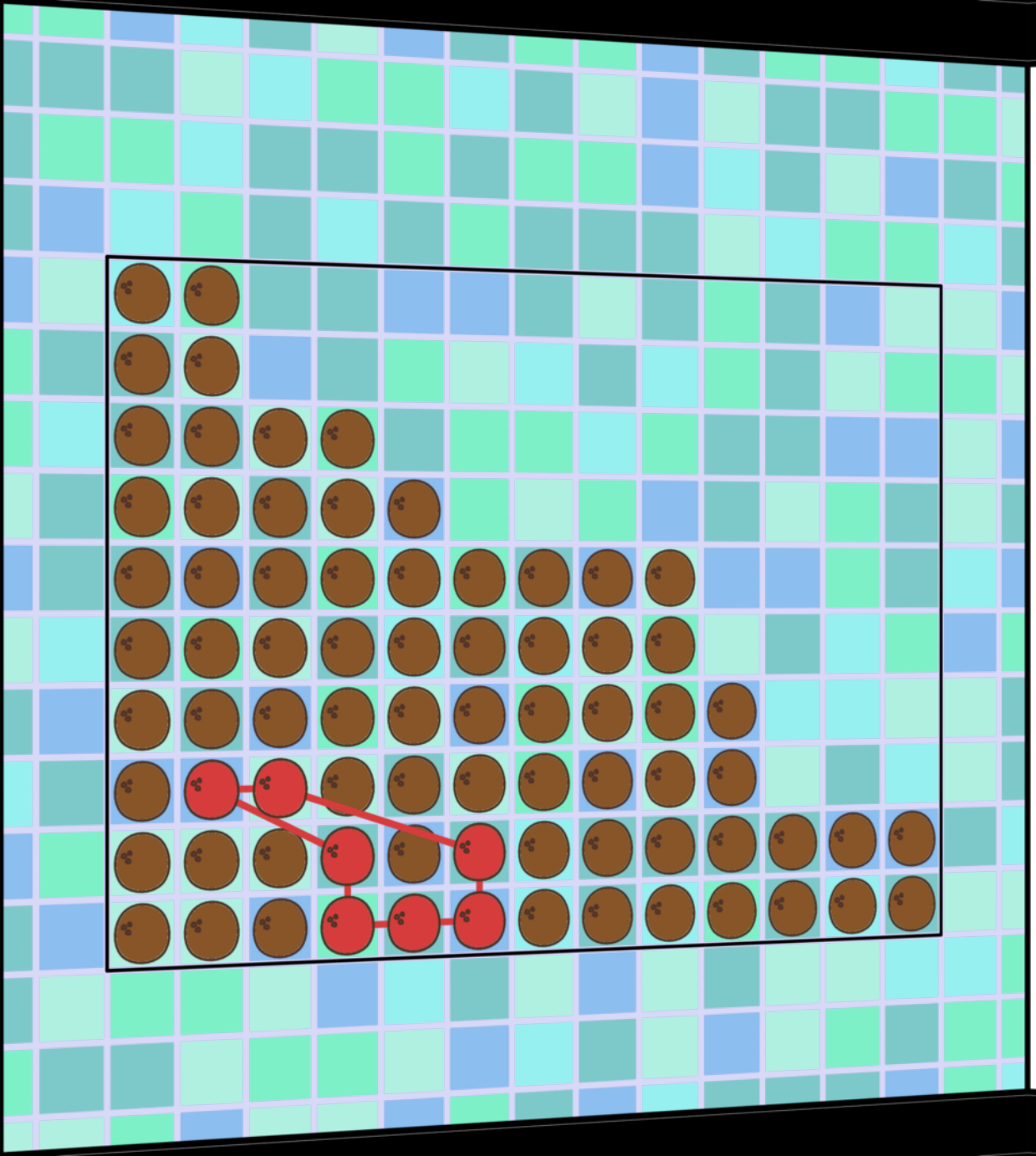


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

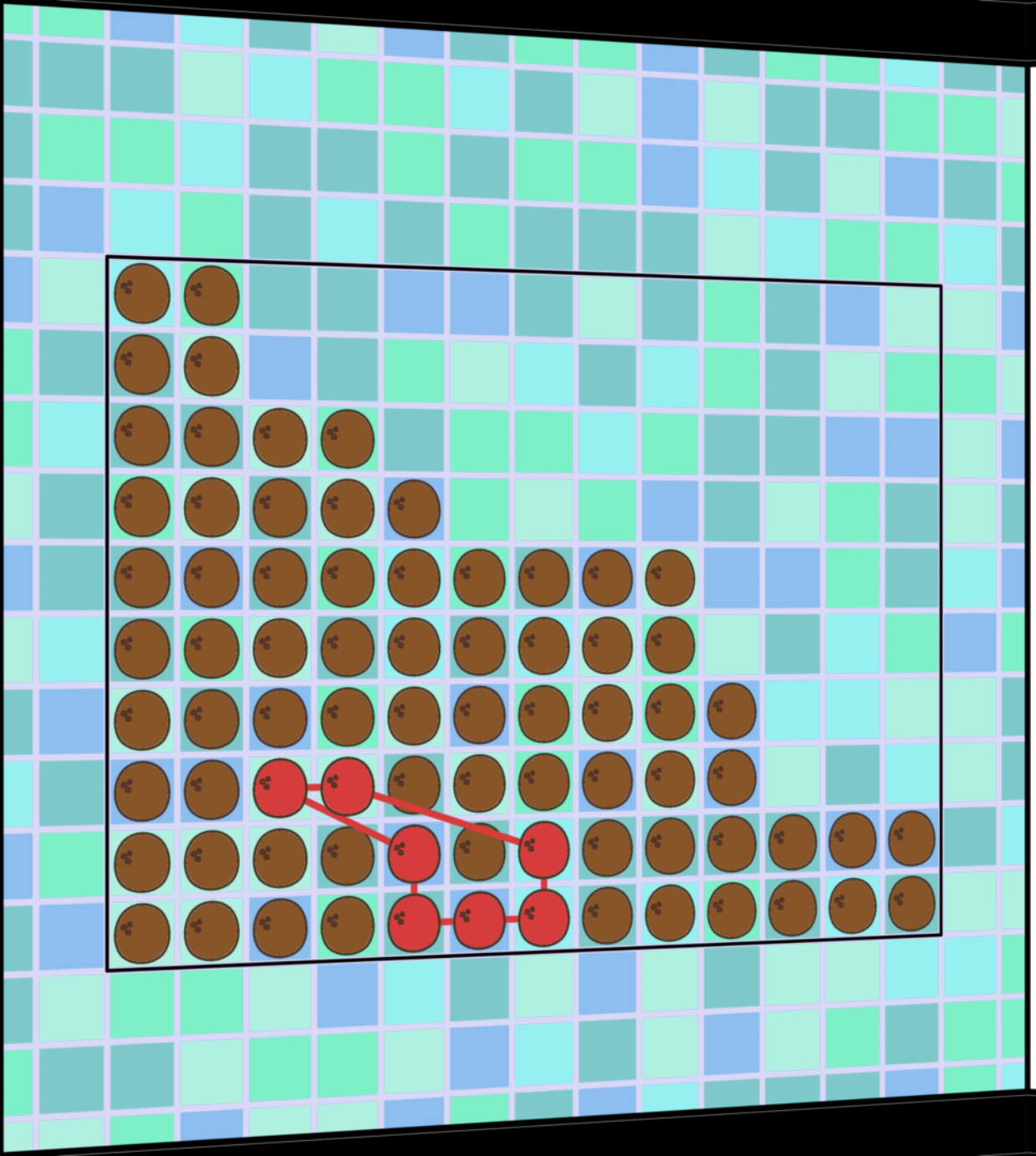


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

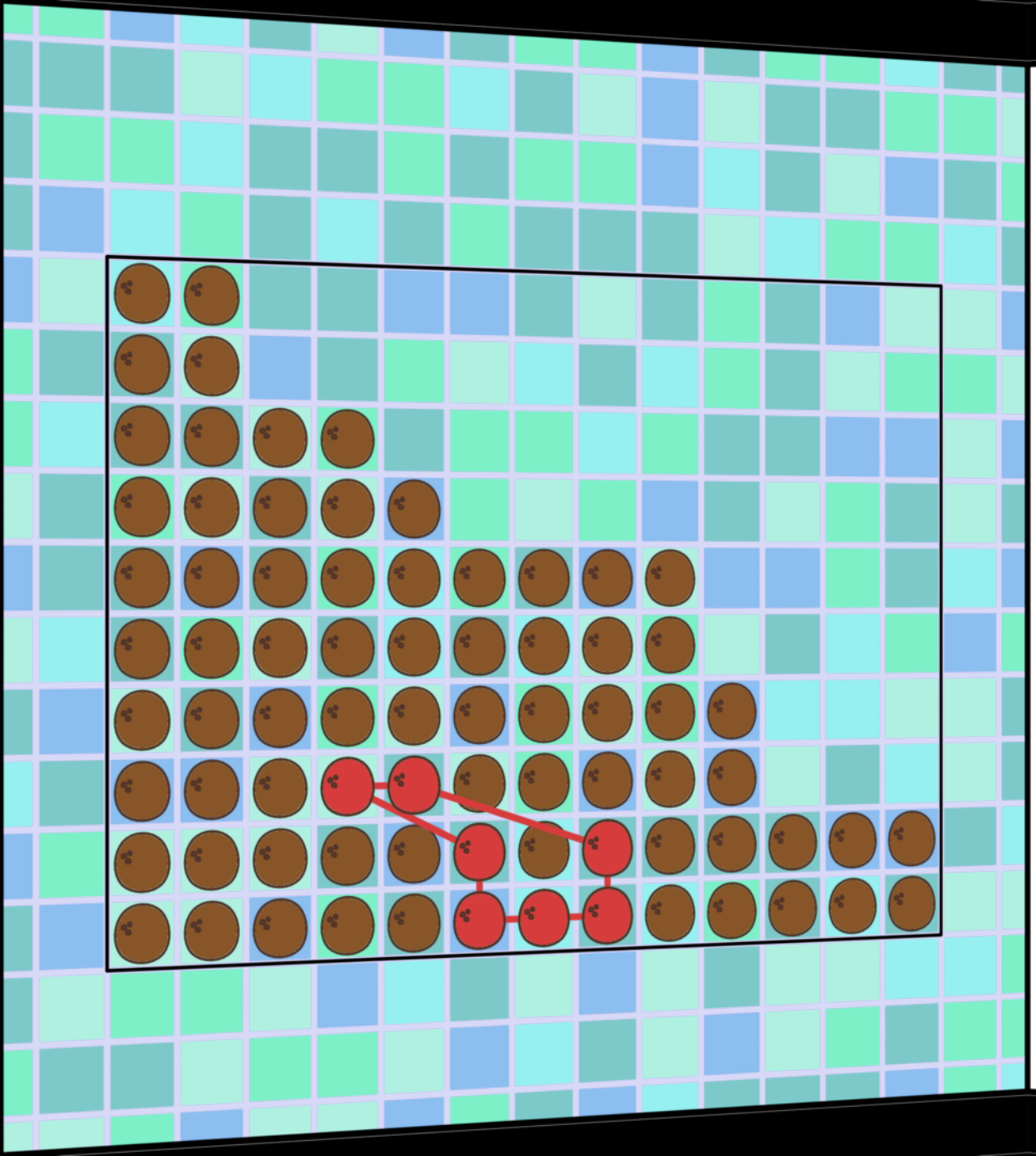


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

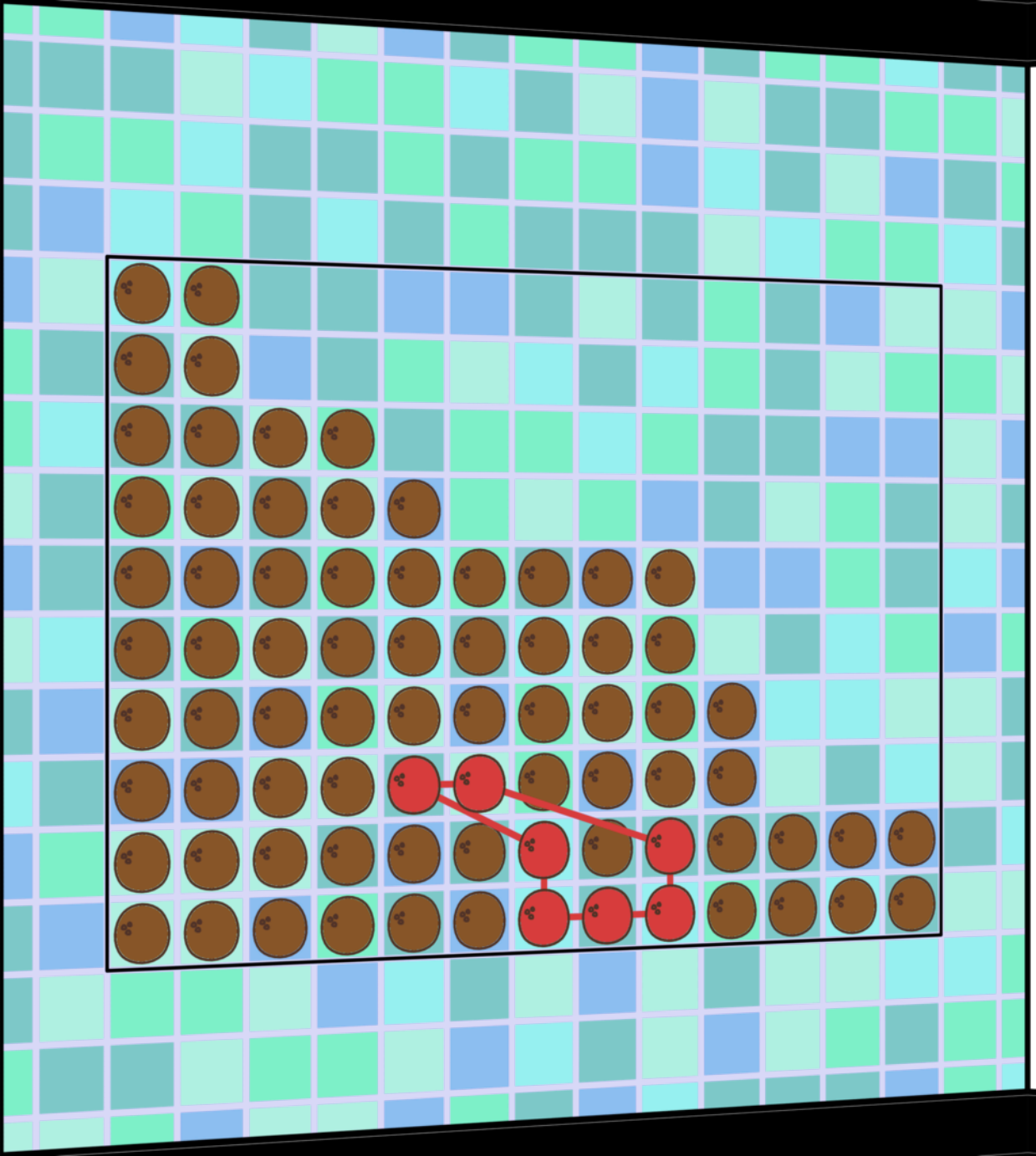


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

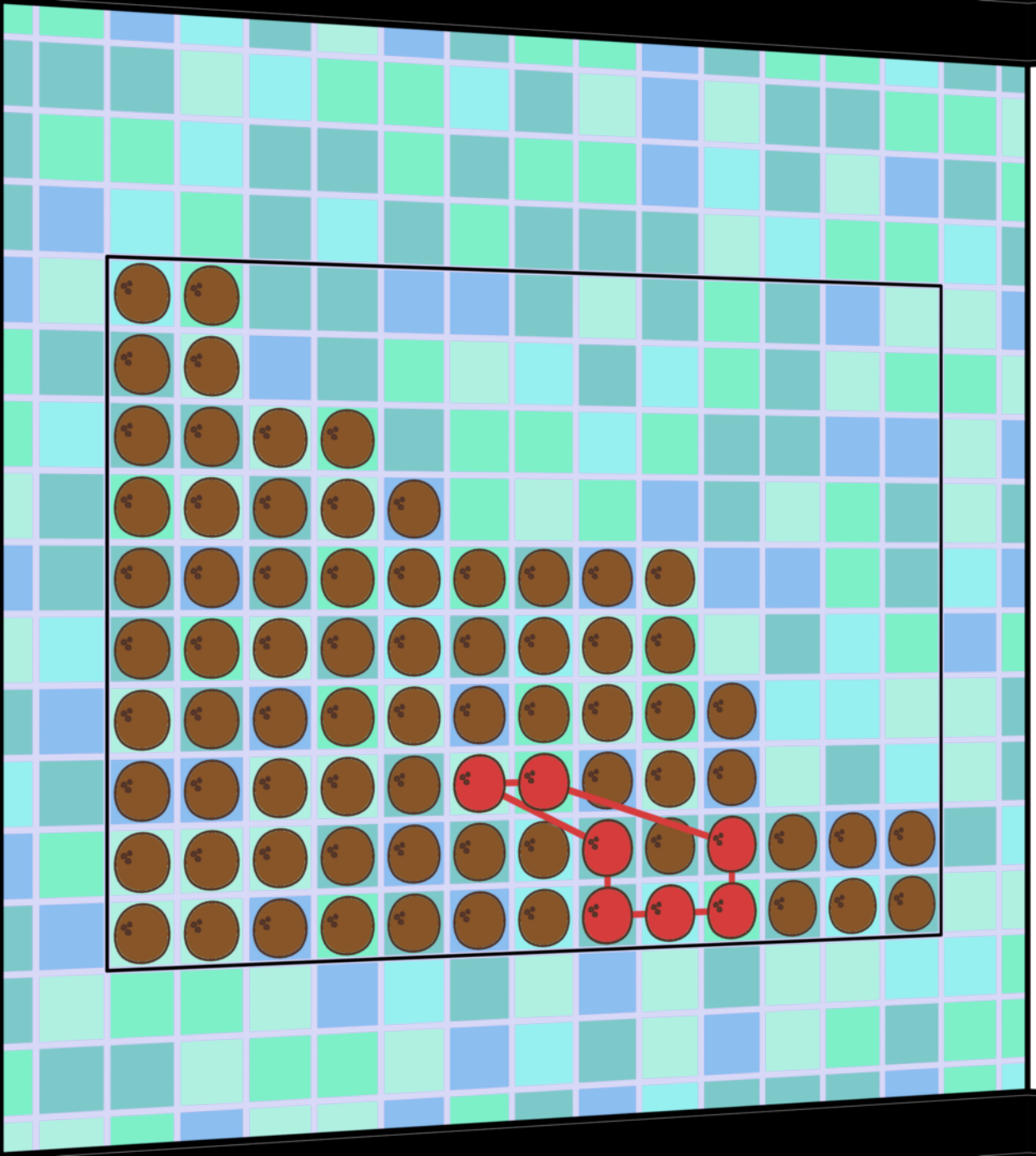


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

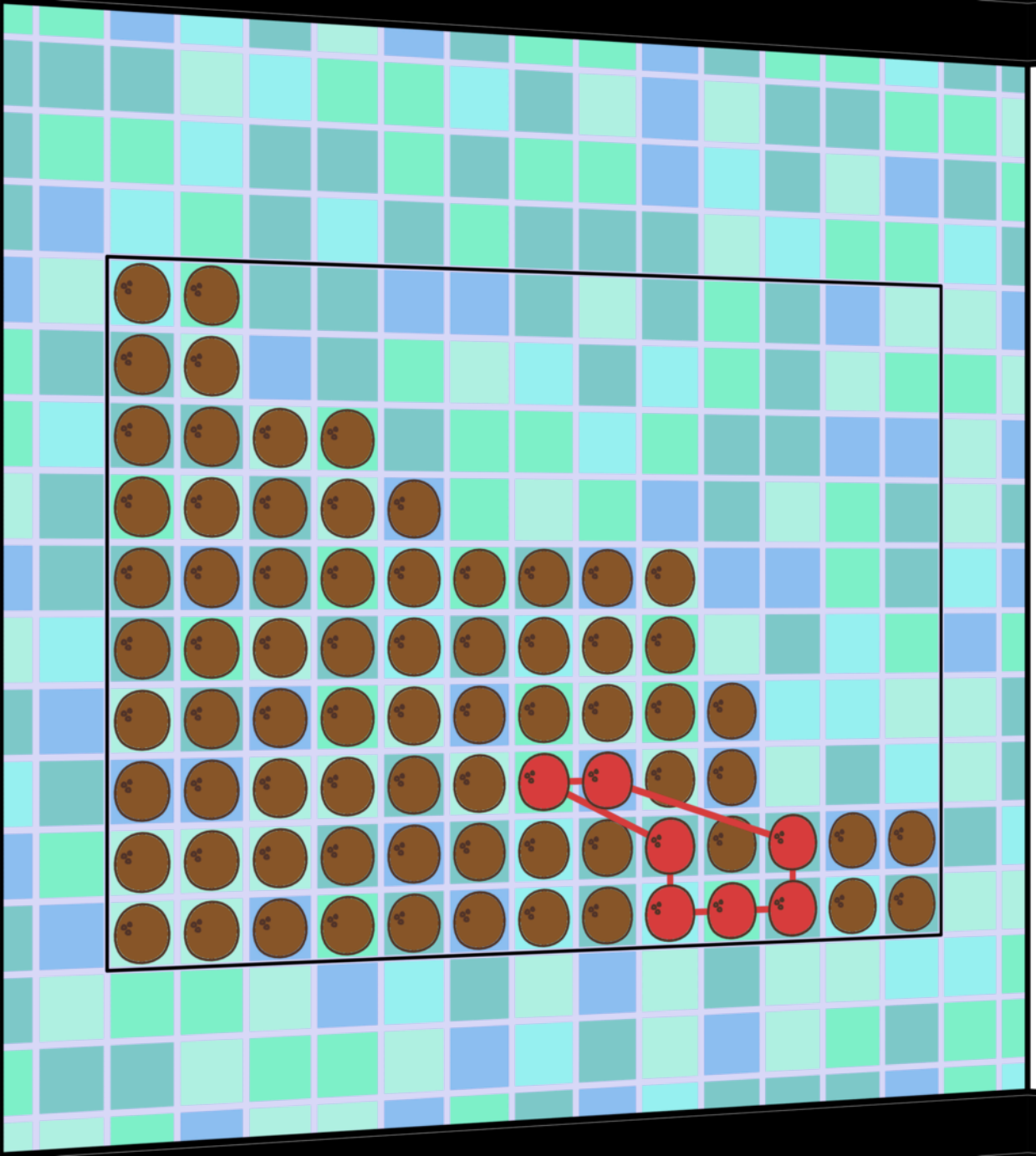


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

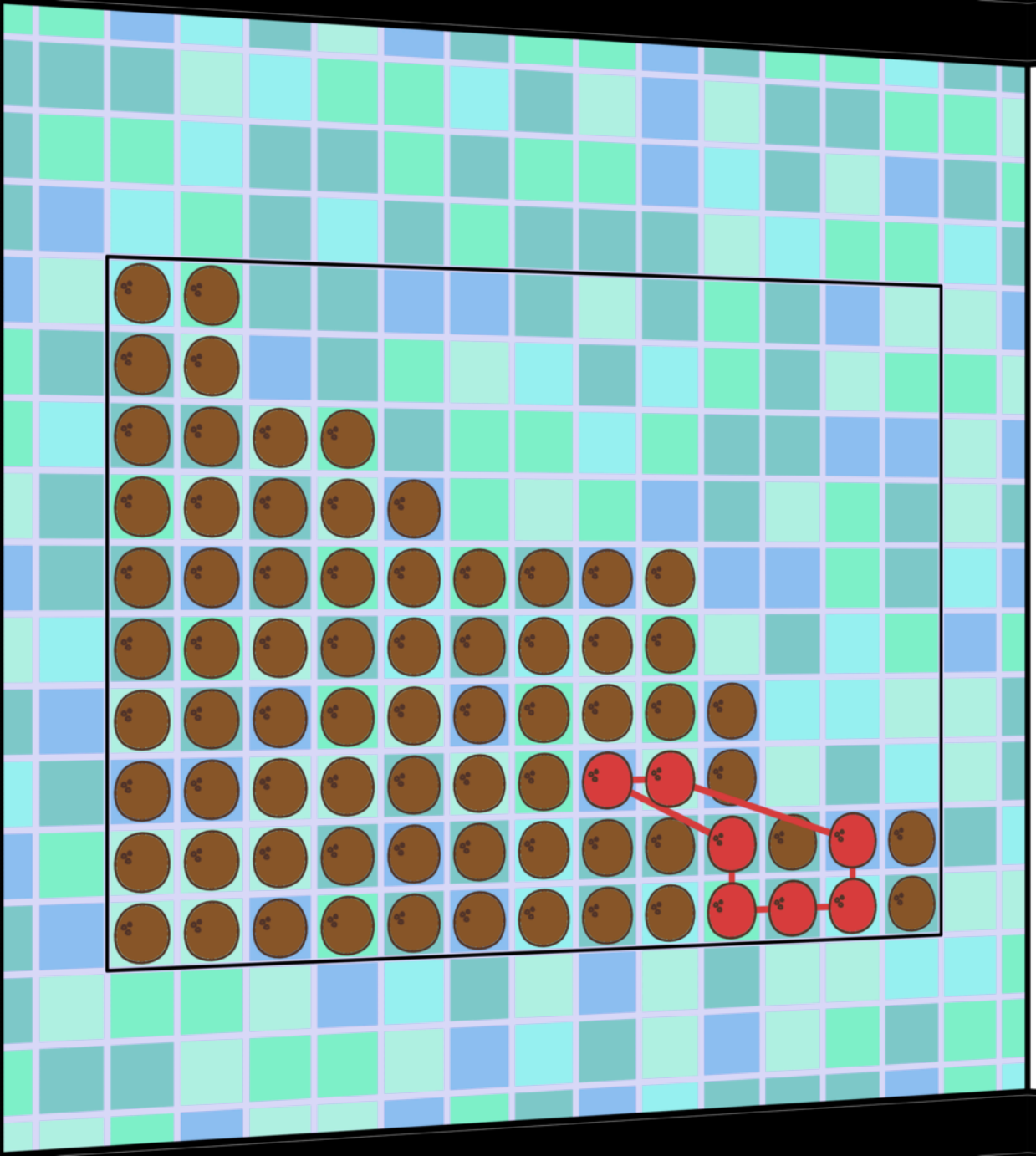


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

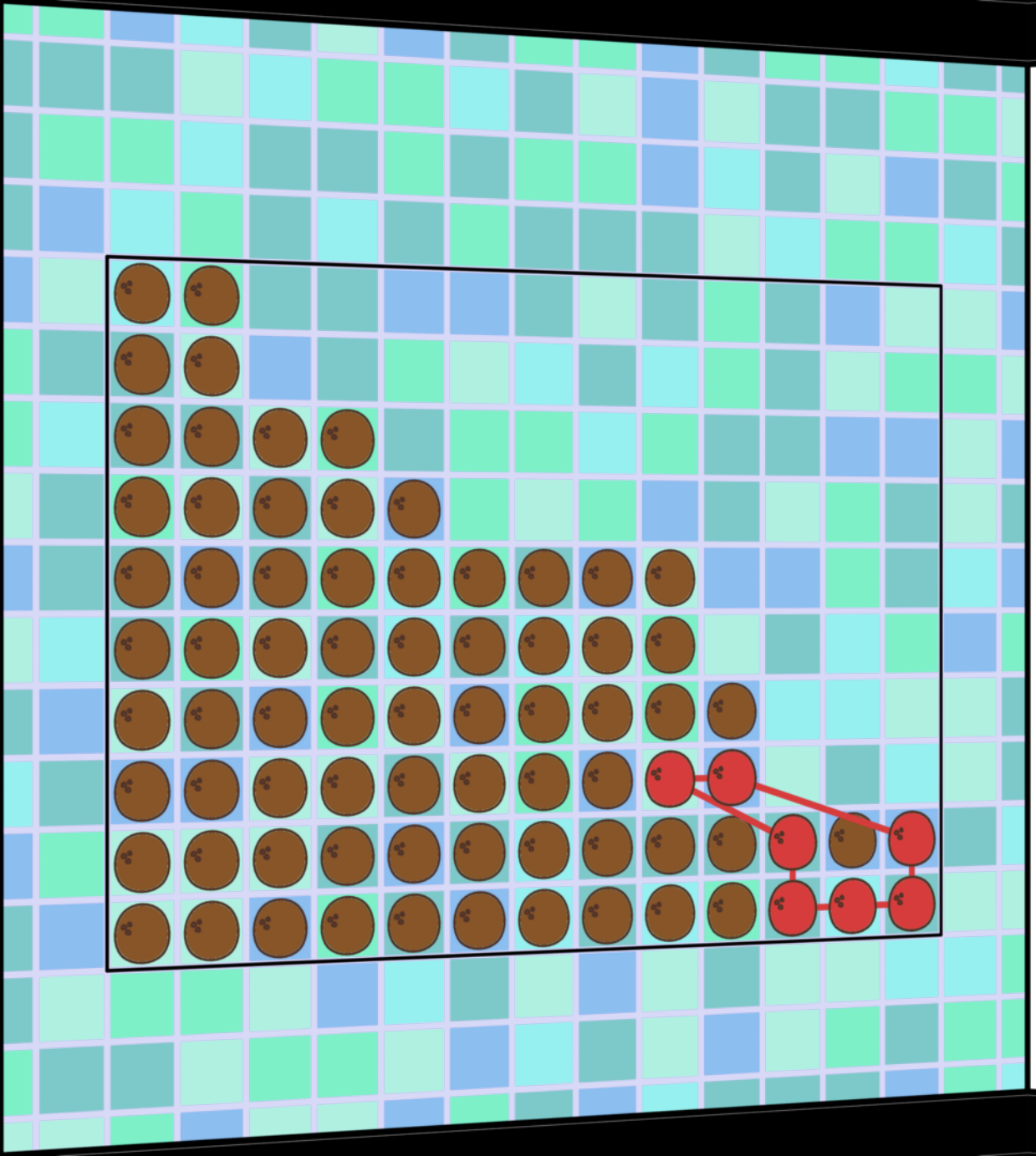


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

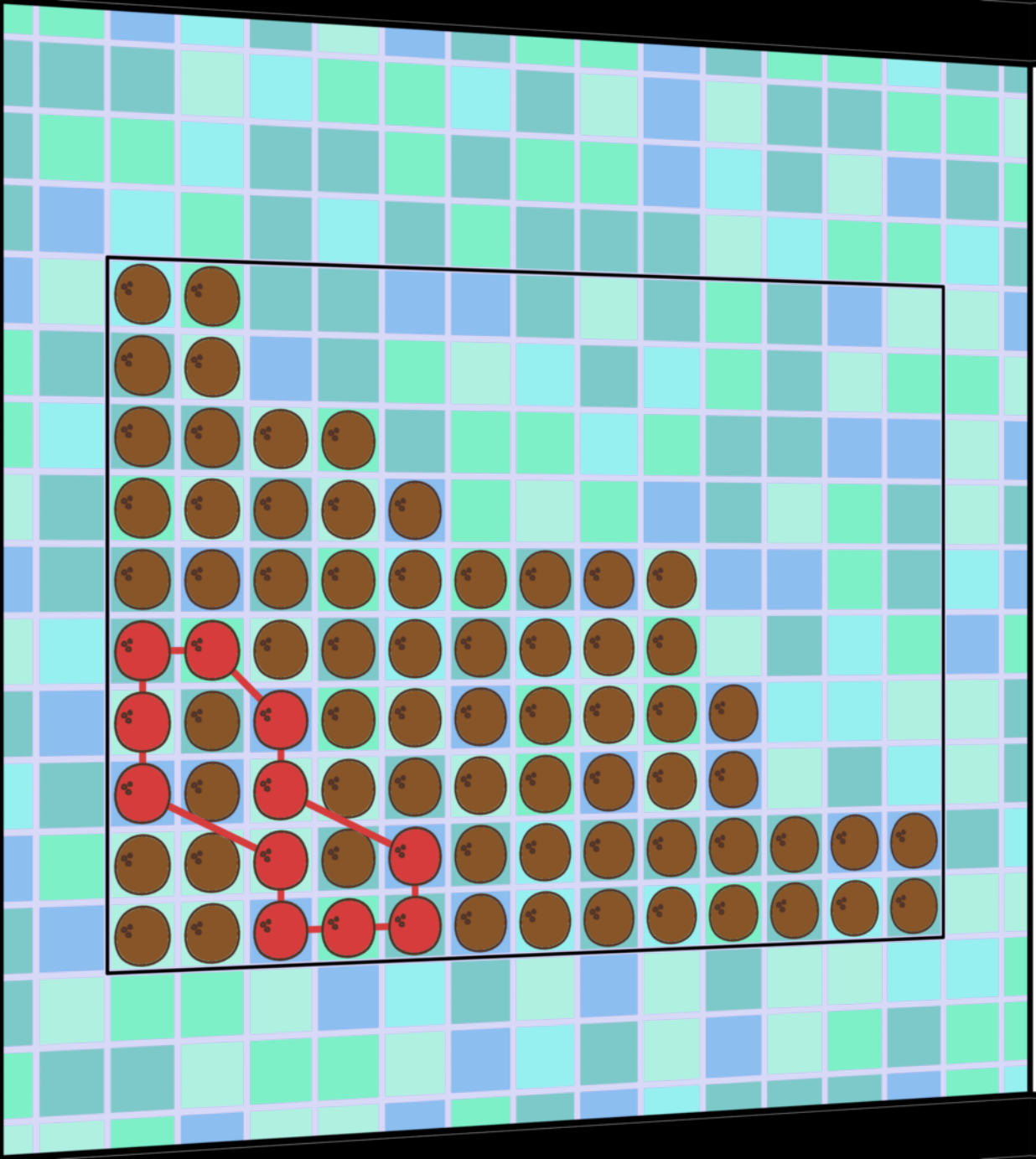


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

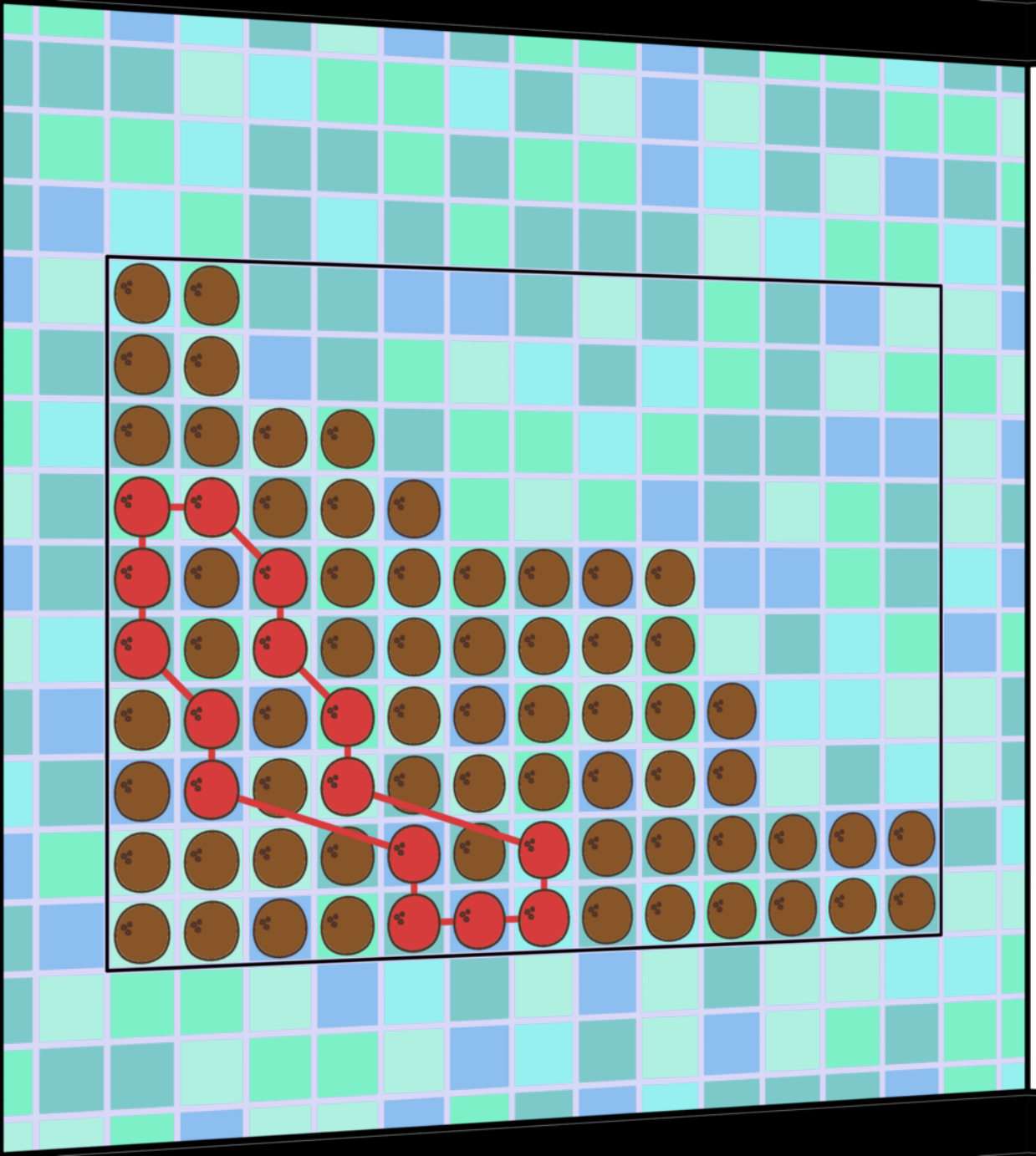


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

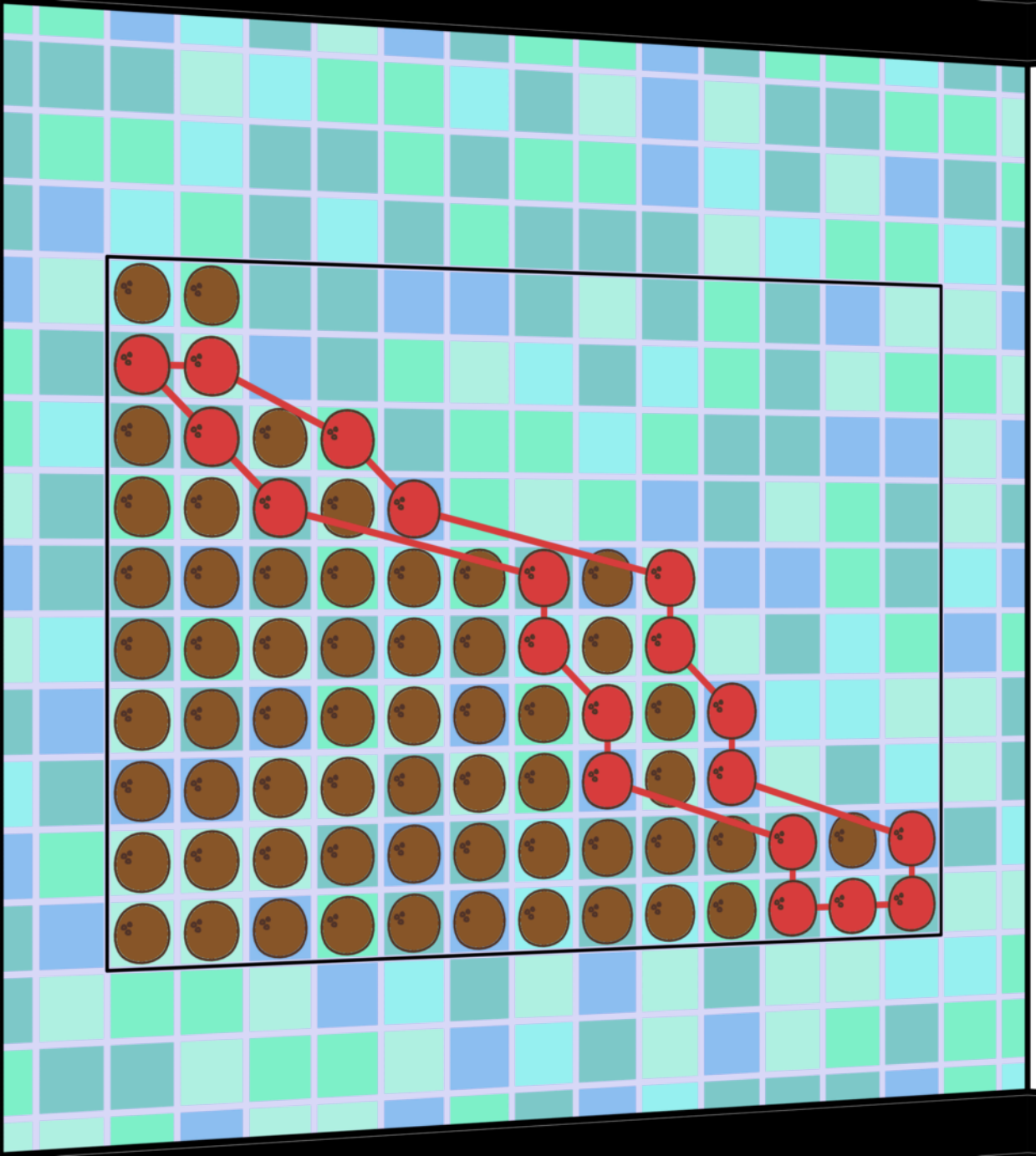


Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.



Theorem 3. Every even permutation can be realized unless:

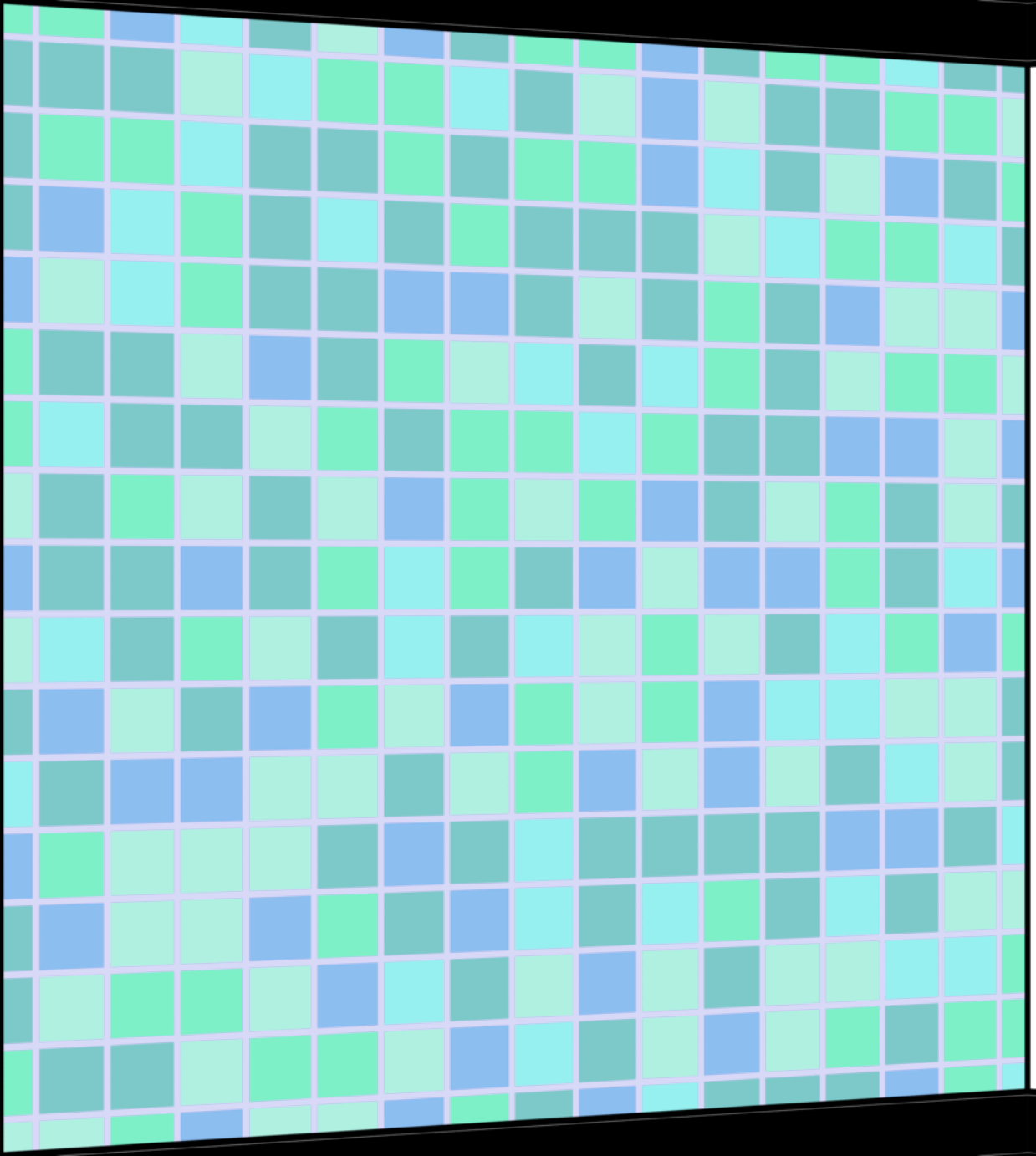
- there is a *core*; or
- it has this shape: 

Proof idea. Cover by cycles that reach all coconuts.

Theorem [Jones]. A good set of cycles implies we get all even permutations.

Theorem 3. Every even permutation can be realized unless:

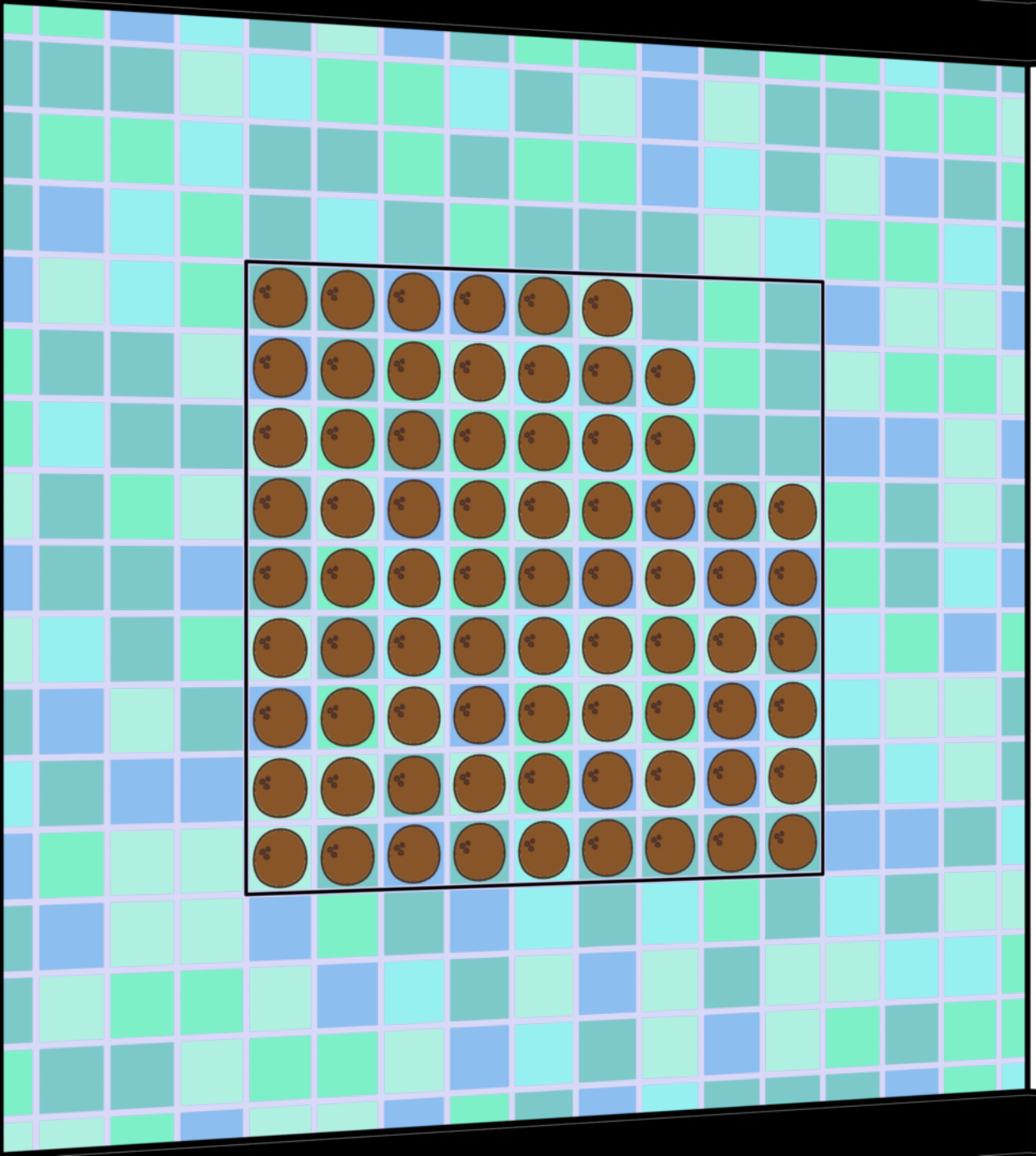
- there is a *core*; or
- it has this shape: 



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

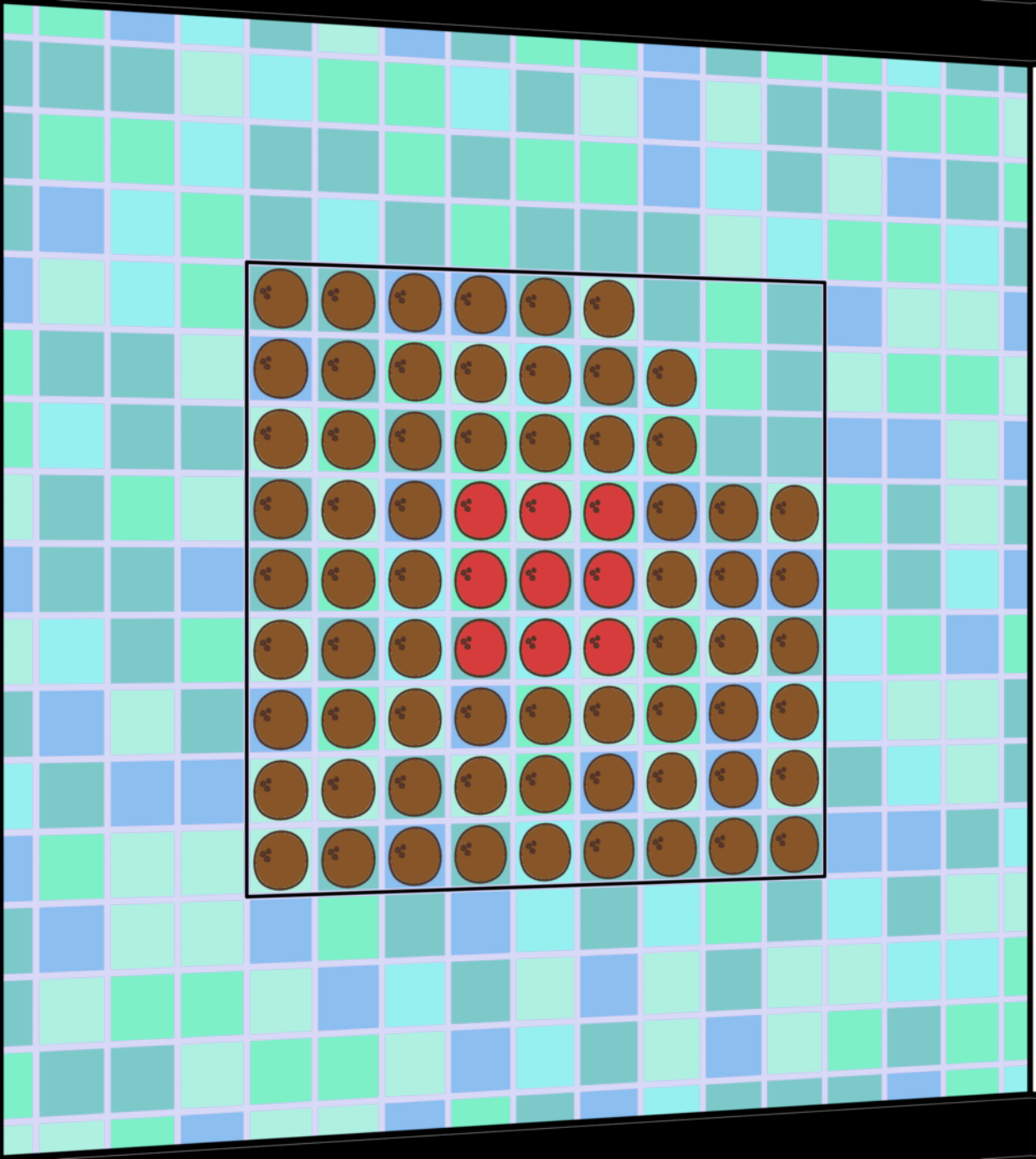
A configuration has an unmovable *core* if at least half of the columns are full and at least half of the rows are full.



Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

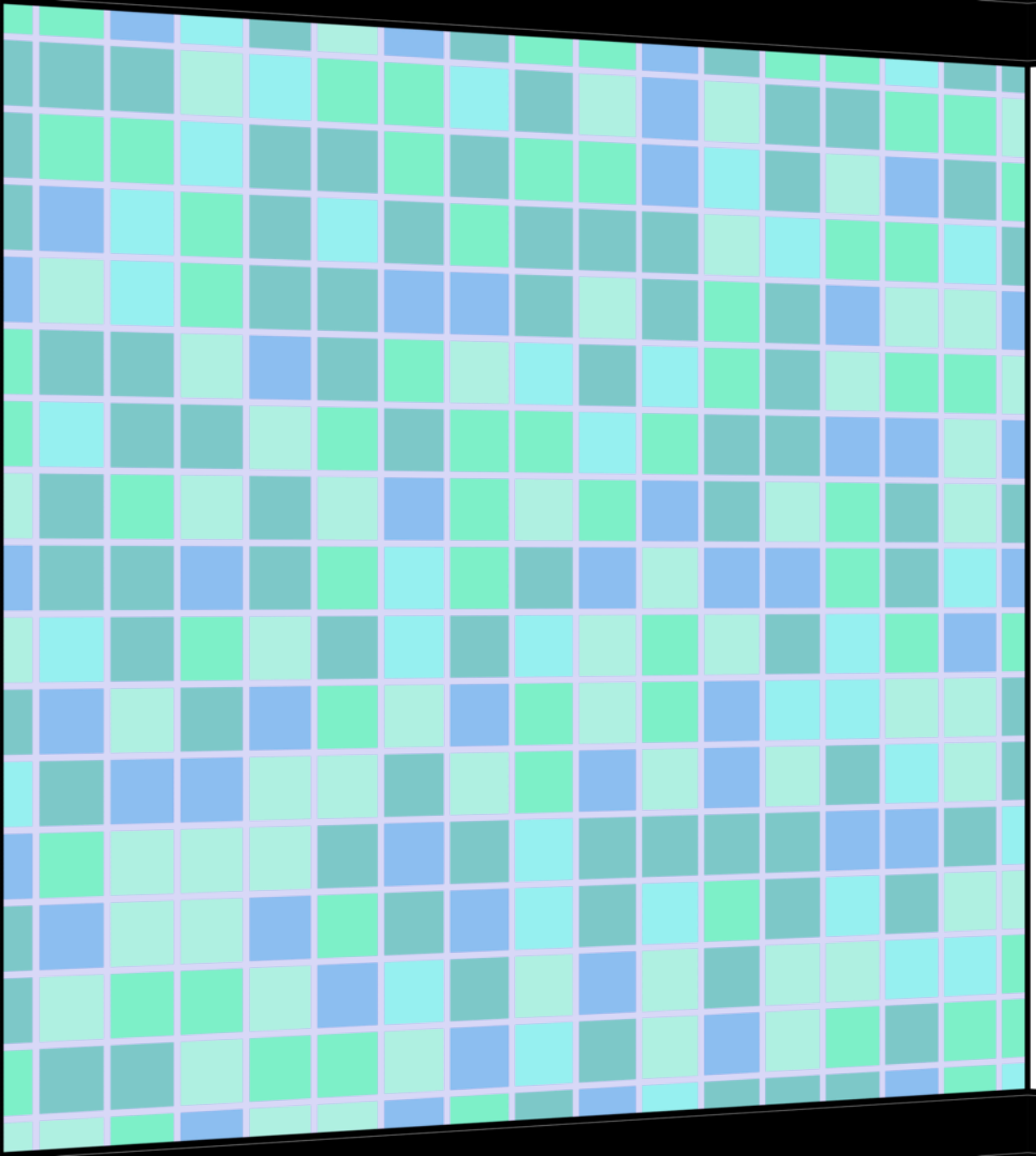
A configuration has an unmovable *core* if at least half of the columns are full and at least half of the rows are full.



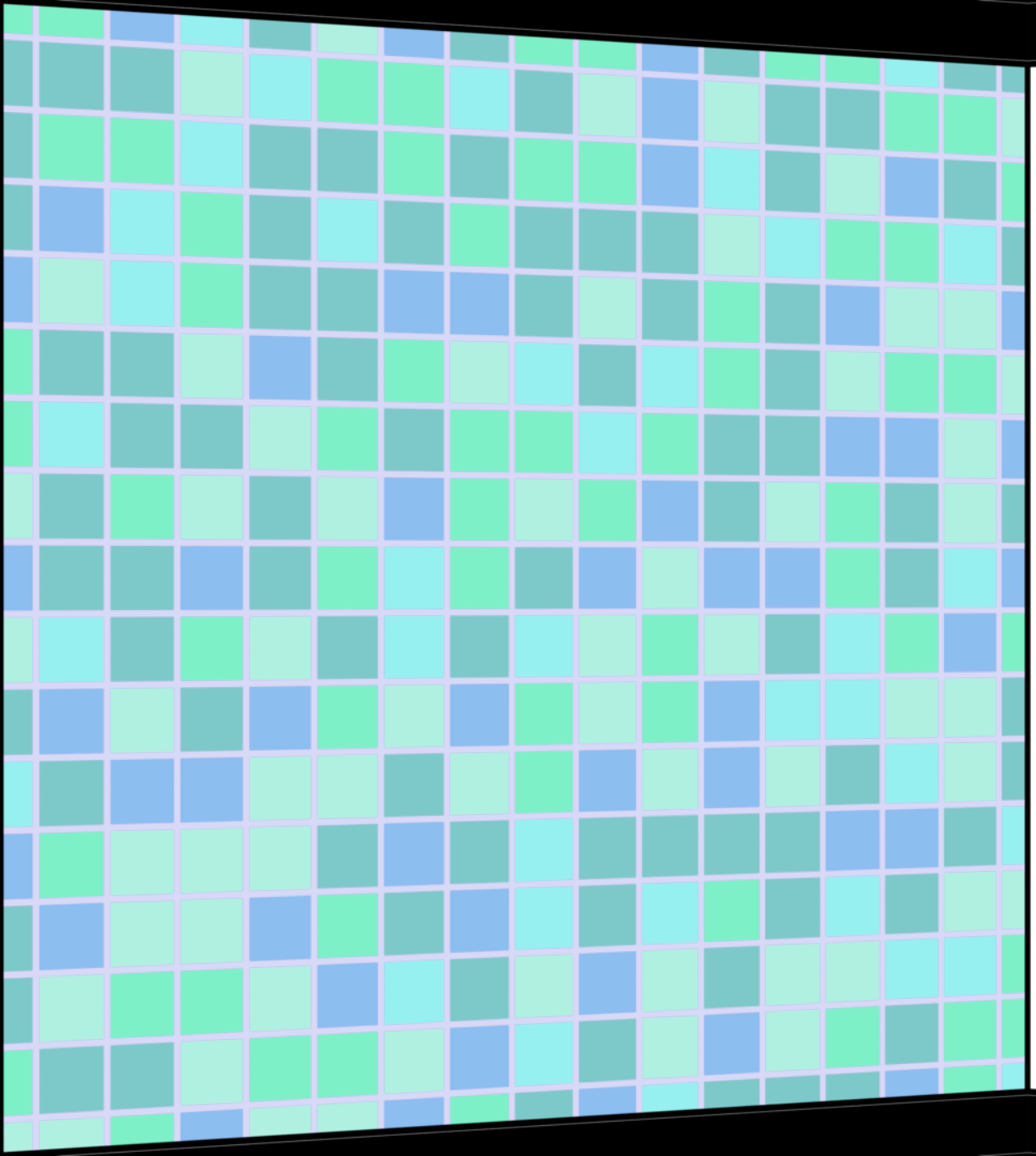
Theorem 3. Every even permutation can be realized unless:

- there is a *core*; or
- it has this shape: 

A configuration has an unmovable *core* if at least half of the columns are full and at least half of the rows are full.

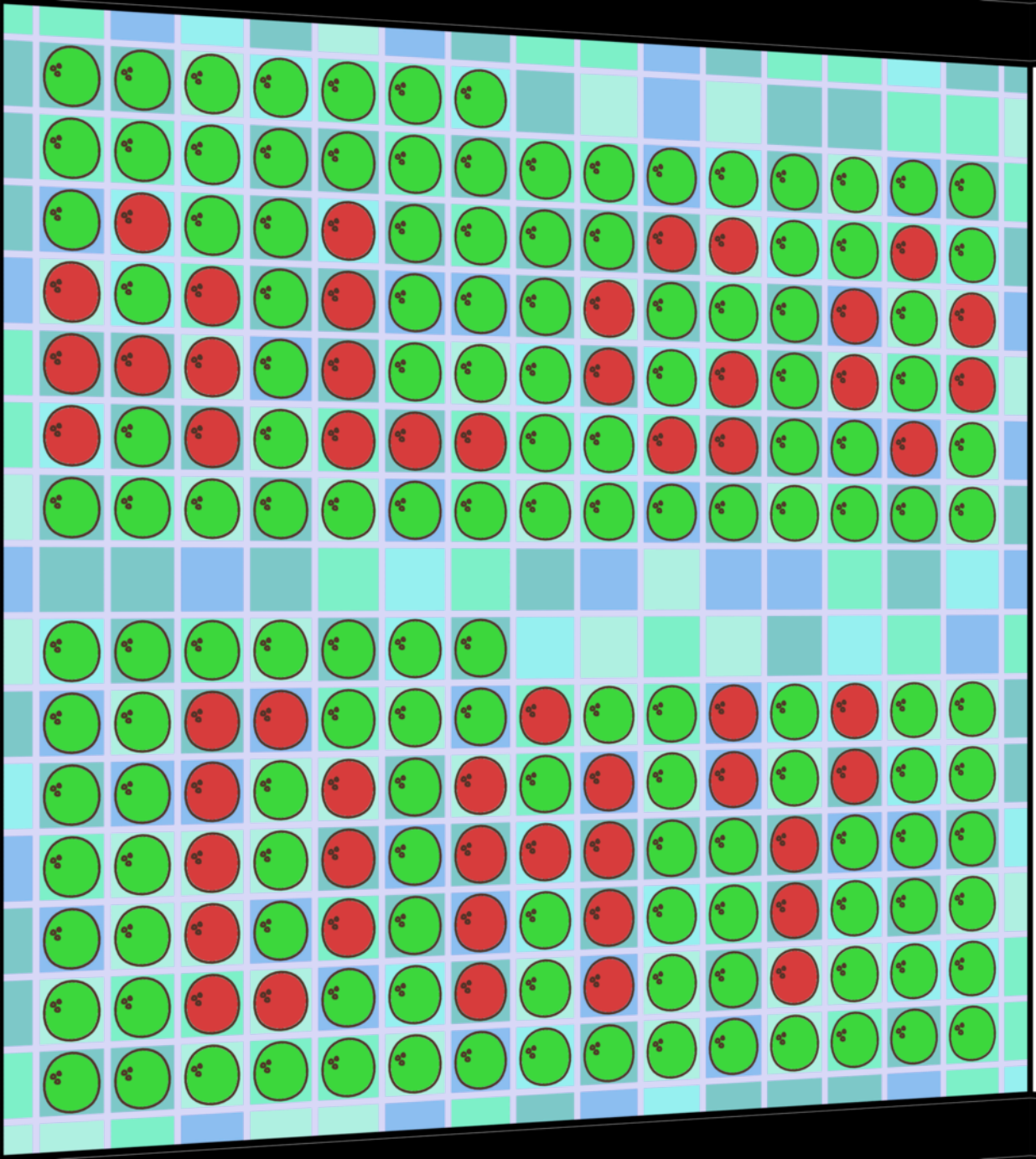


Open problem. Algorithms?



Open problem. Algorithms?

The theorem by Jones is not constructive.



Labeled Reconfiguration by Compaction

Maarten Löffler

Based on joint work with

Hugo Akitaya
Greg Aloupis
Anika Rounds
Giovanni Viglietta