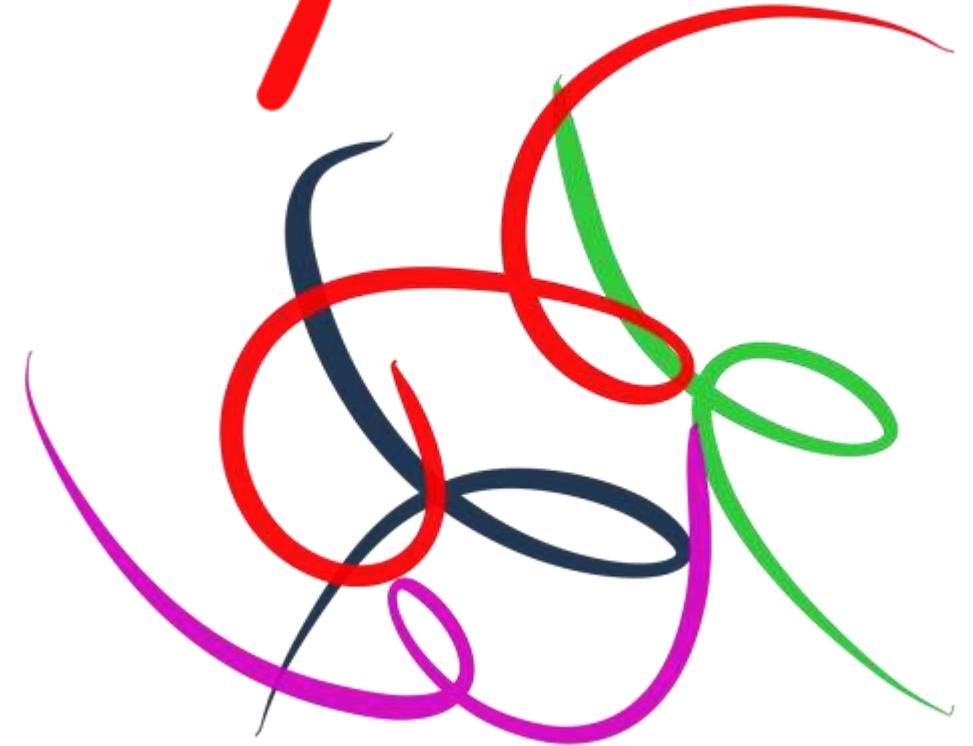


Run  
Faster

Jeremy Clark



Parallel Programming in C#

@jeremybytes

what?

@jeremybytes 

What?

CPU-bound  
Operations

@jeremybytes



# CPU-bound Operations

Data Processing

Complex Calculations

Data Manipulation

@jeremybytes



why?



@jeremybytes

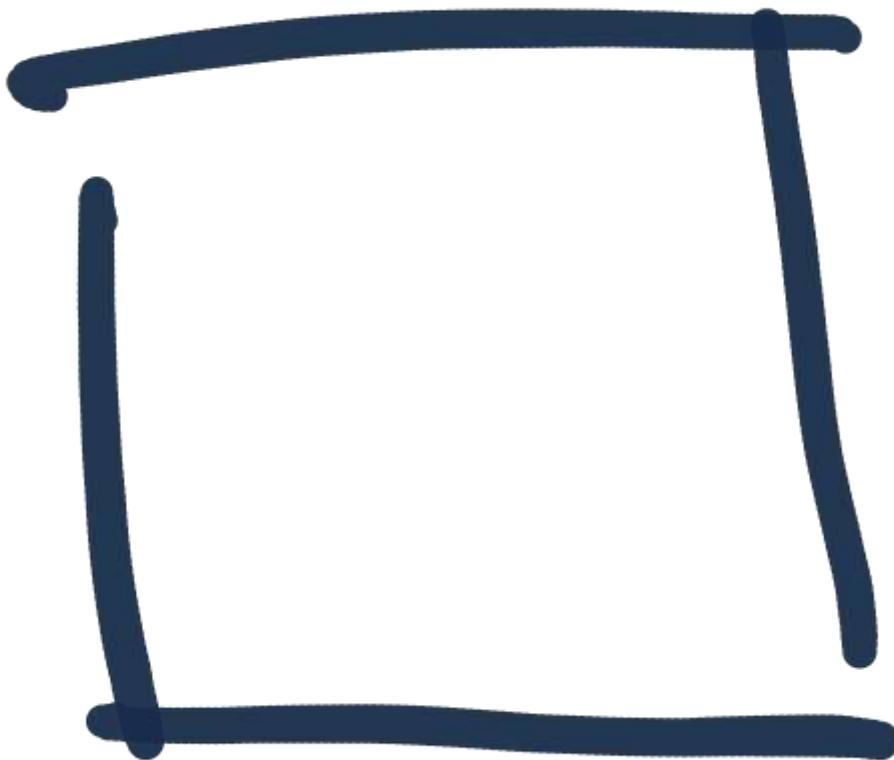


CPUs are getting  
Bigger  
Not Faster

Why?

@jeremybytes

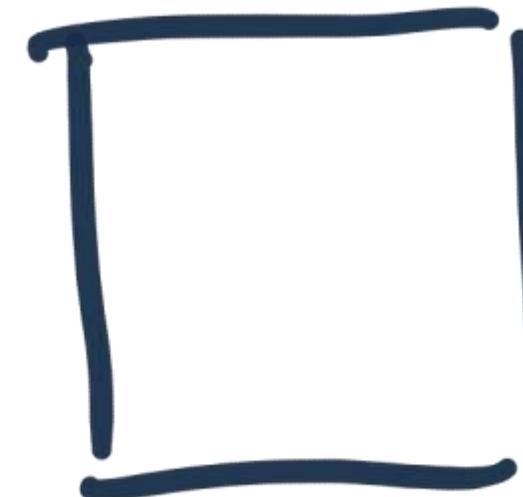
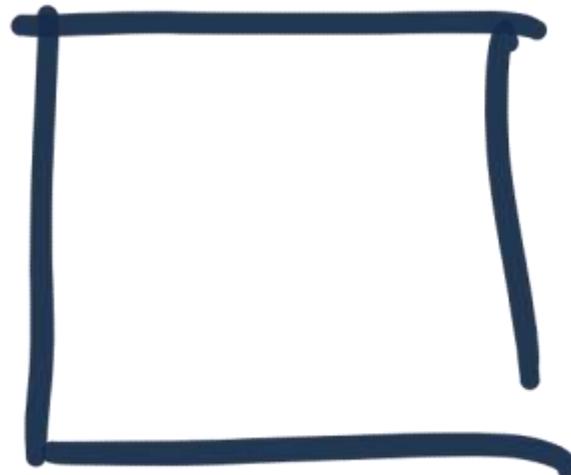
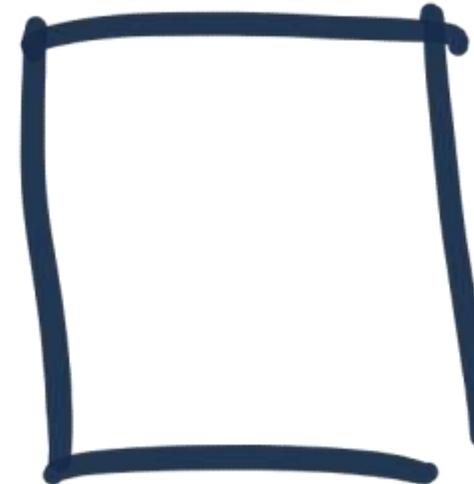
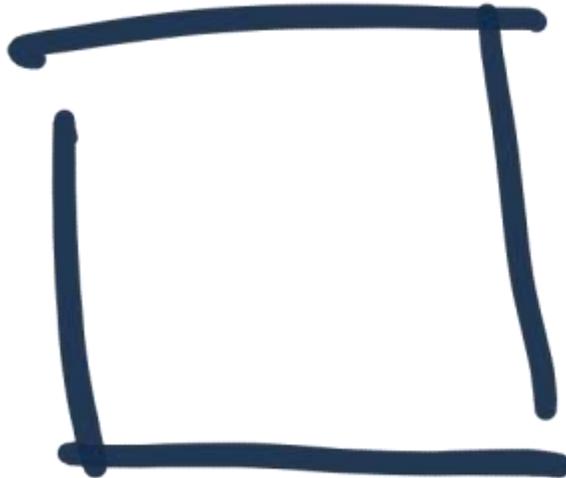




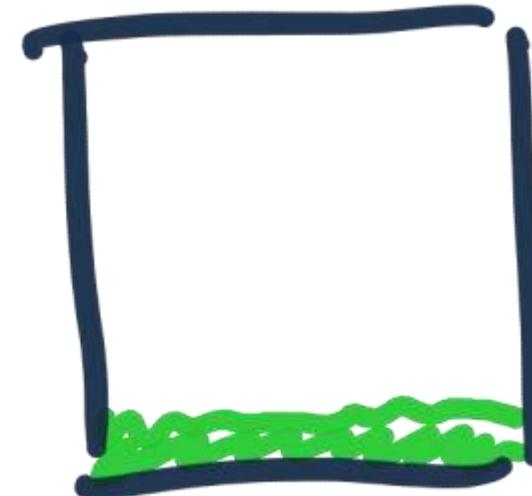
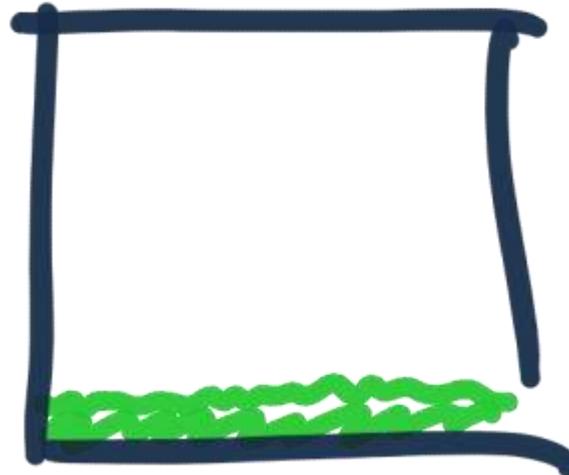
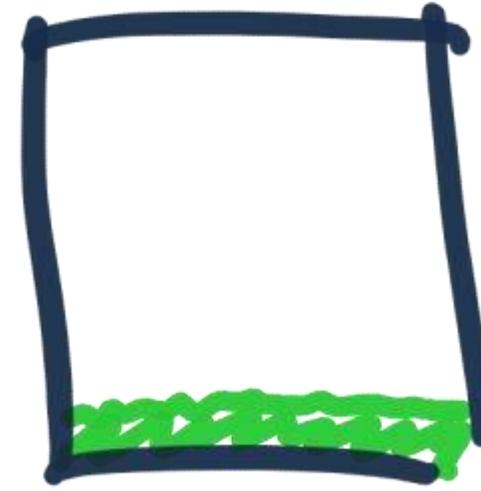
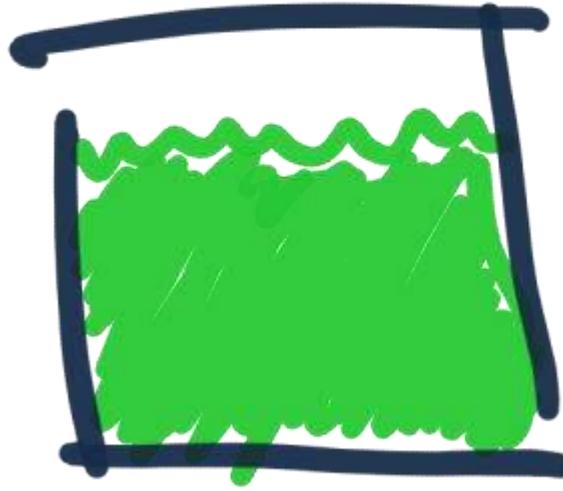
@jeremybytes



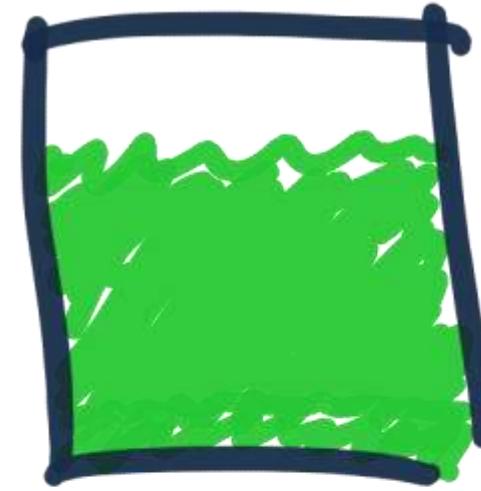
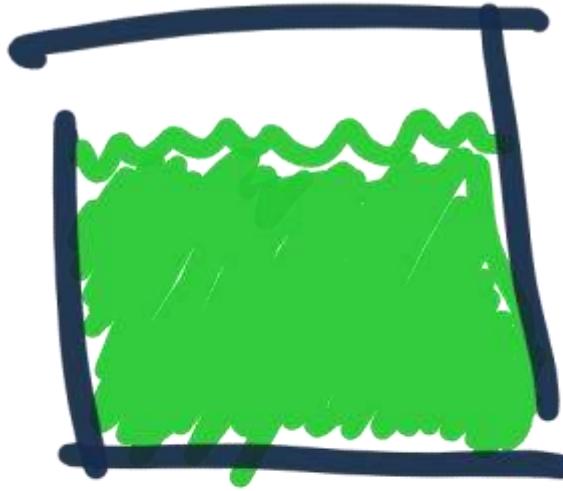
@jeremybytes



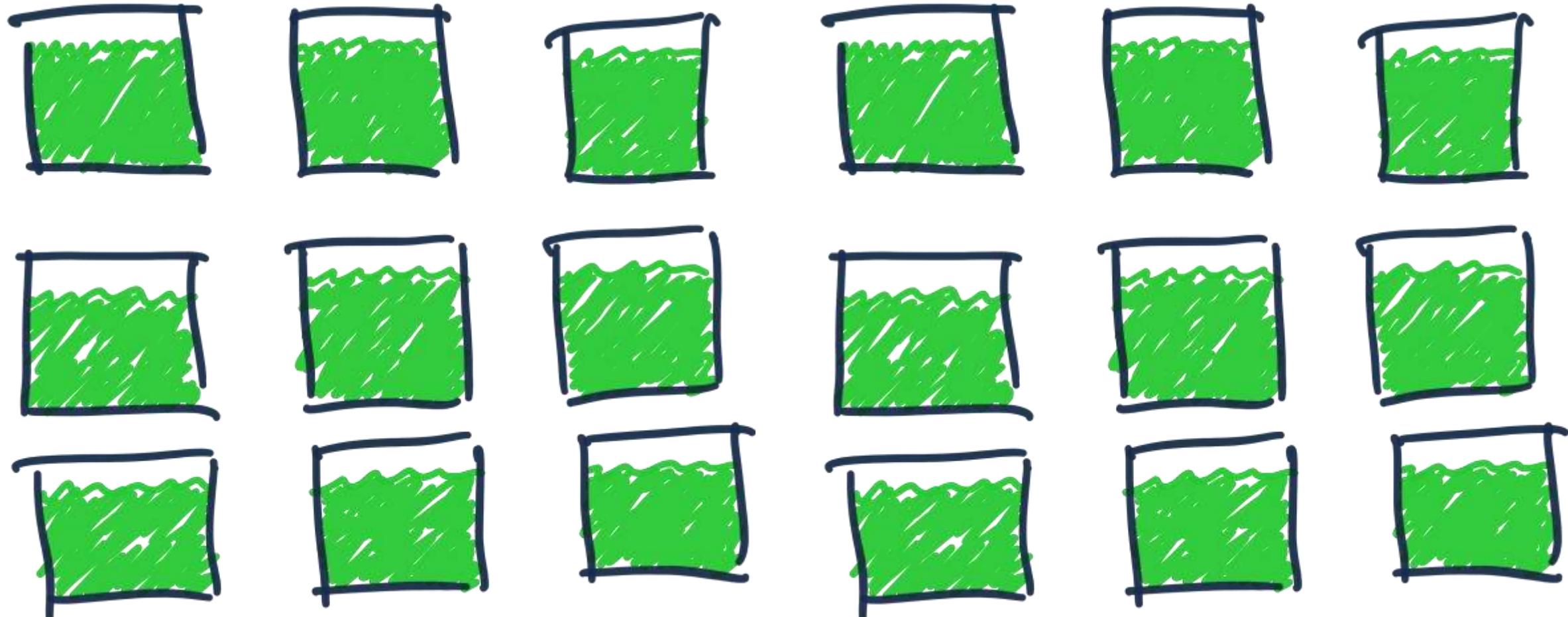
@jeremybytes



@jeremybytes



@jeremybytes



@jeremybytes

Can I Run  
in Parallel?

@jeremybytes



Maybe

Maybe Not

@jeremybytes



Atomic

Maybe

Deterministic

Discrete Input / Output

No Shared Data



Maybe

Maybe Not

@jeremybytes



Shared Data

Maybe Not

Shared Resources

External Dependencies

Ordered/Sequential

@jeremybytes



Parallel.For

Parallel.ForEach

@jeremybytes

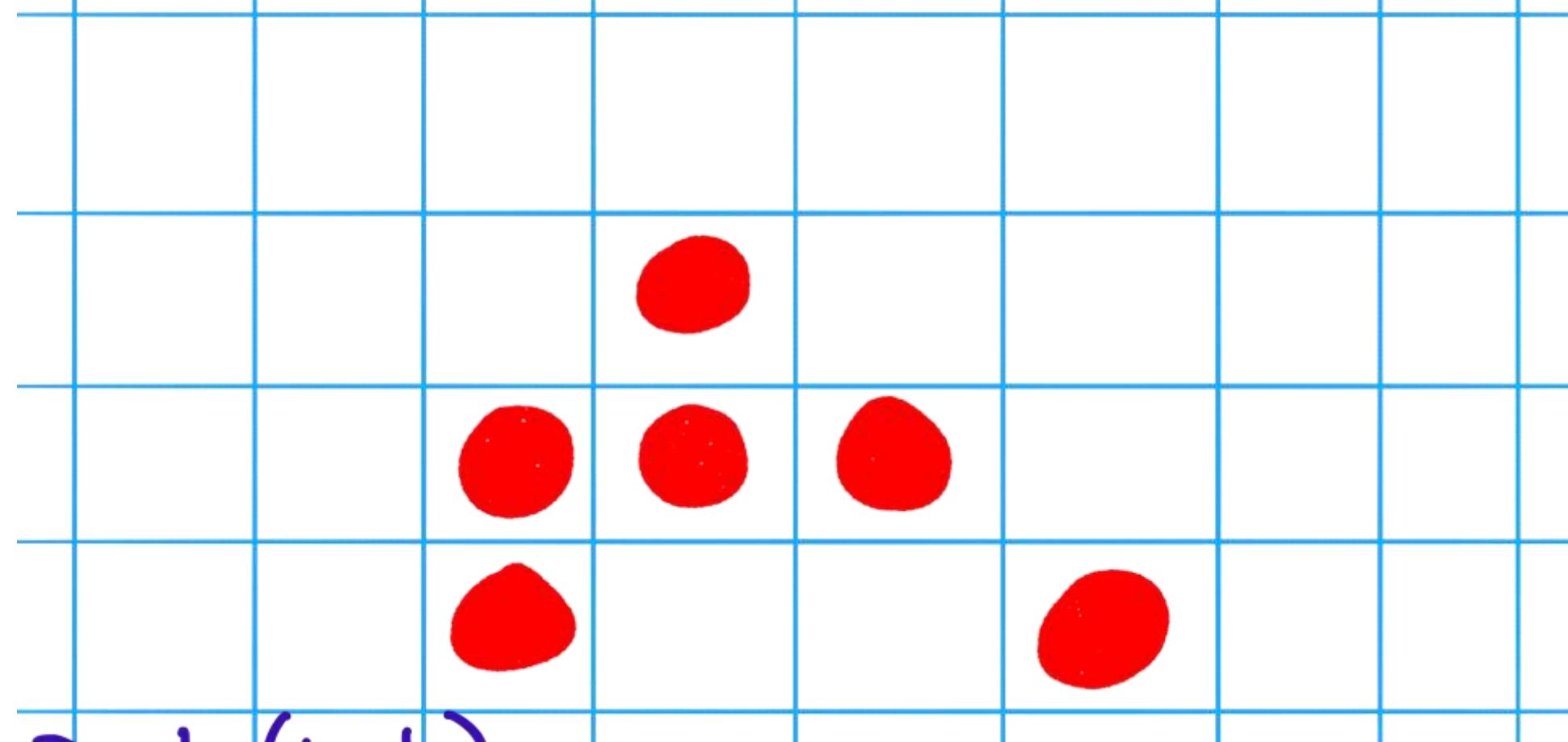


# Conway's Game of Life

# Parallel.FOR

# Parallel.ForEach

# Conway's Game of Life



Live w/  $< 2 = \text{Dead}$  (lonely)

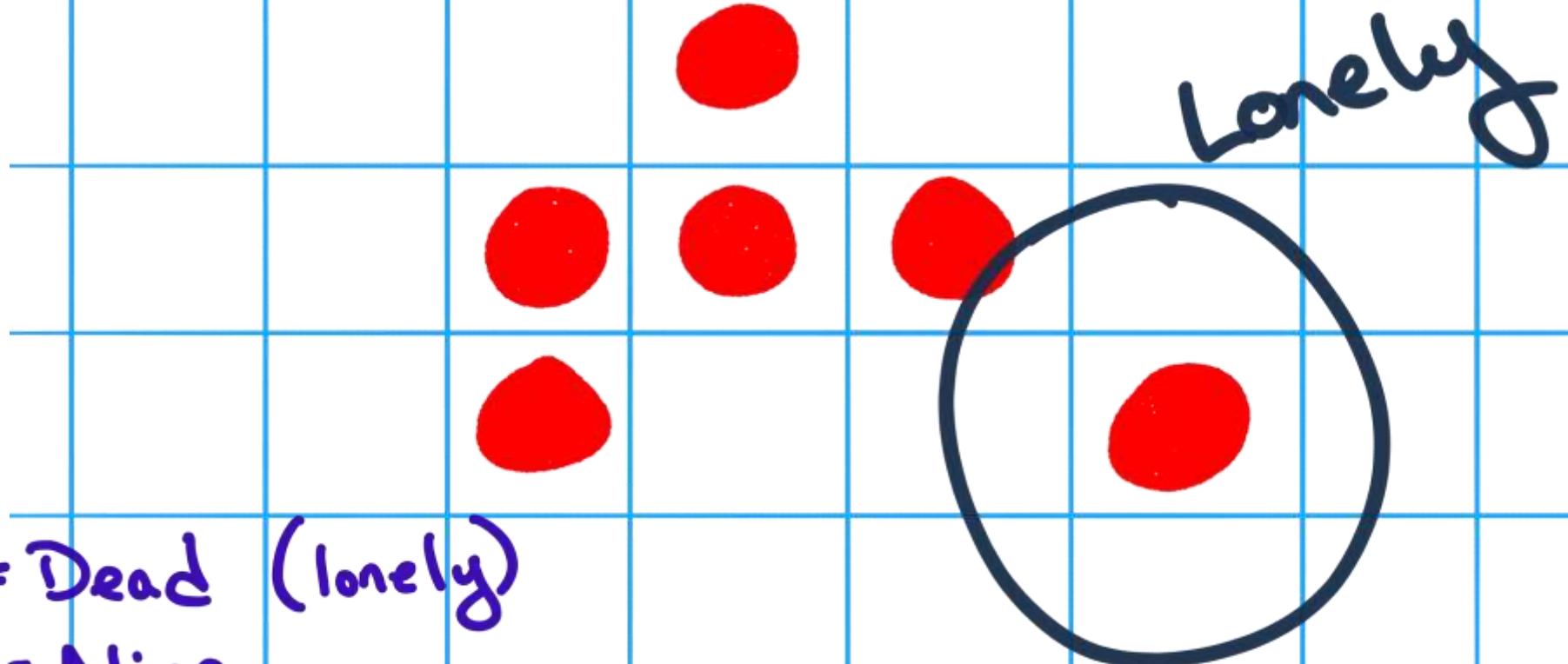
Live w/ 2 or 3 = Alive

Live w/  $> 3 = \text{Dead}$  (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes





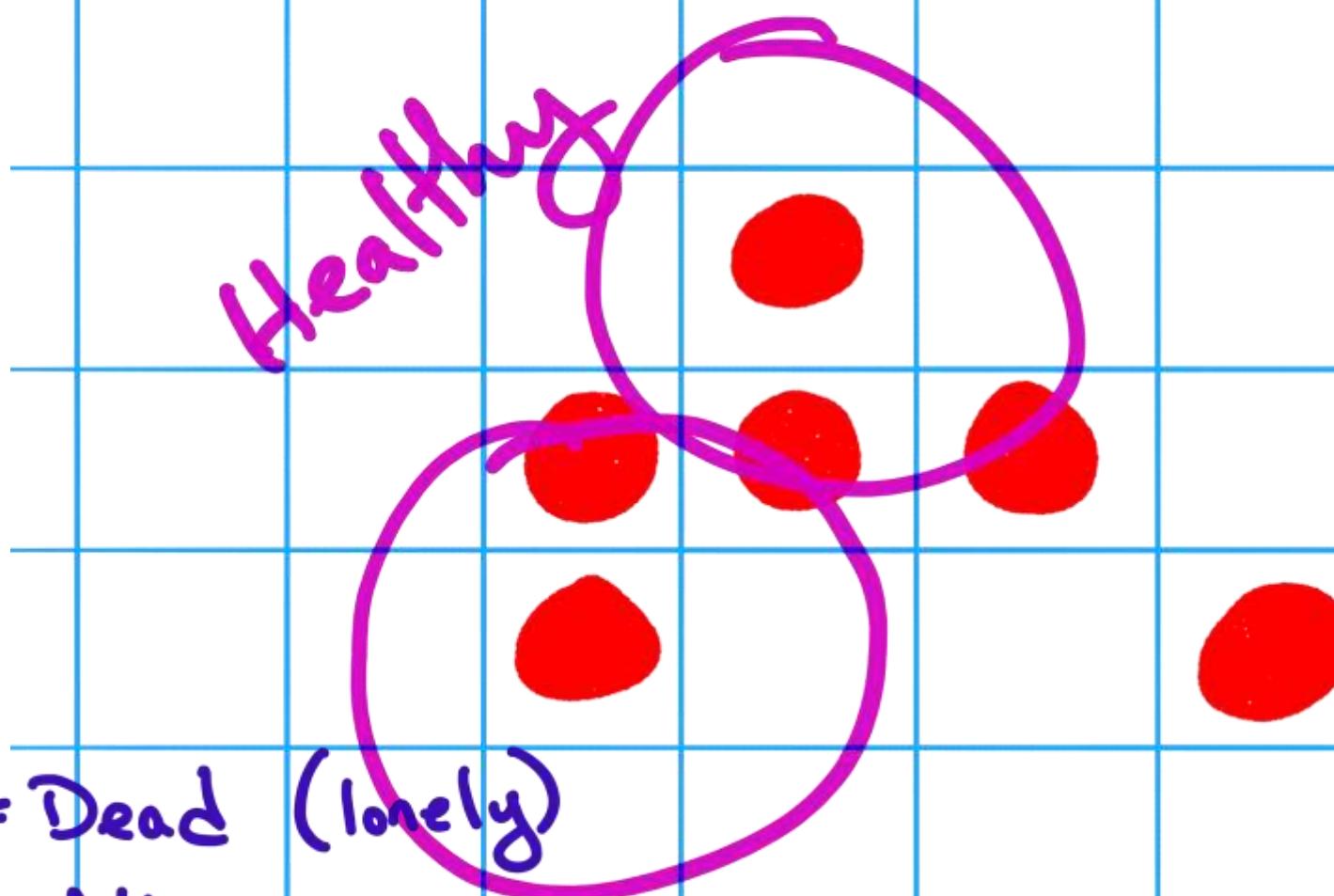
Live w/  $< 2$  = Dead (lonely)

Live w/ 2 or 3 = Alive

Live w/  $> 3$  = Dead (crowded)

Dead w/ 3 = Alive (reproduction)





Live w/  $< 2$  = Dead

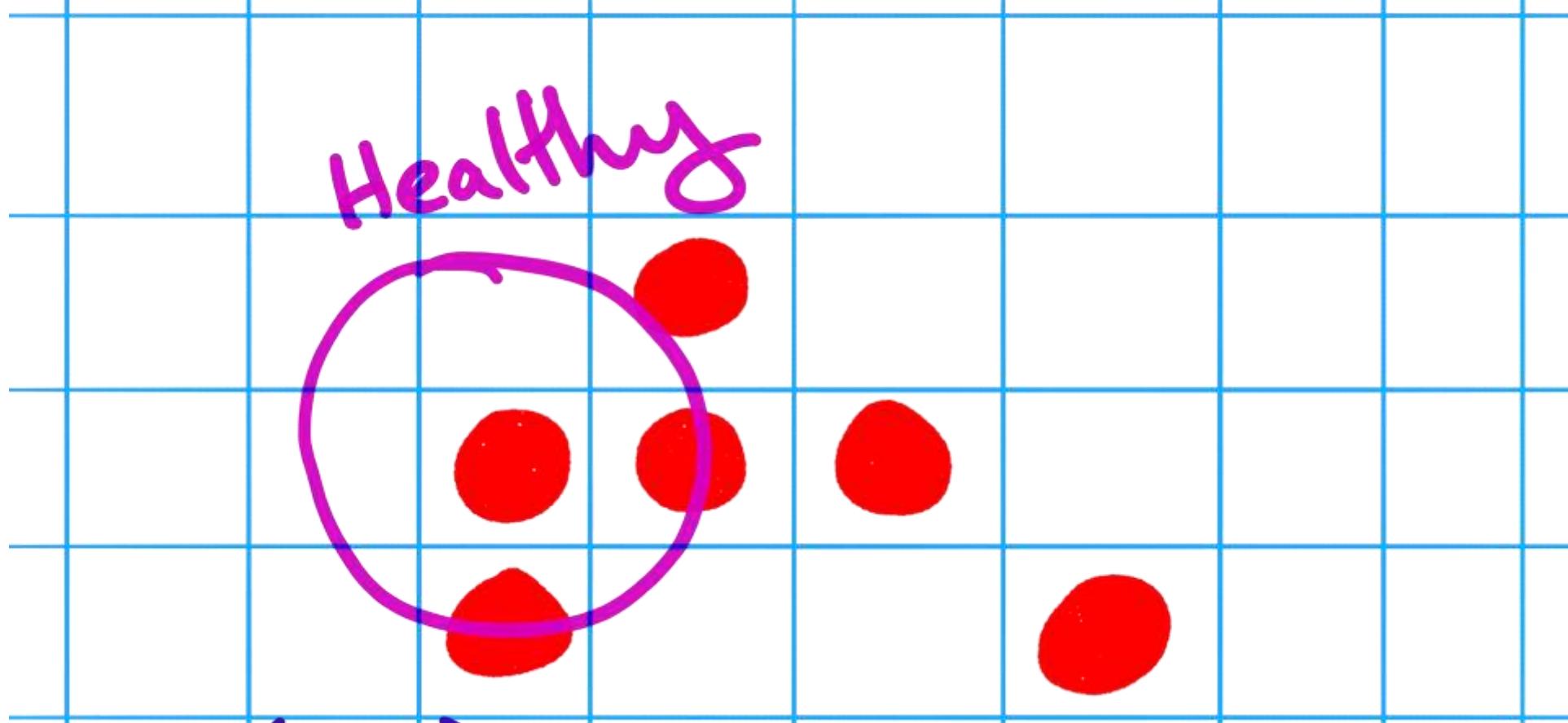
Live w/ 2 or 3 = Alive

Live w/  $> 3$  = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes





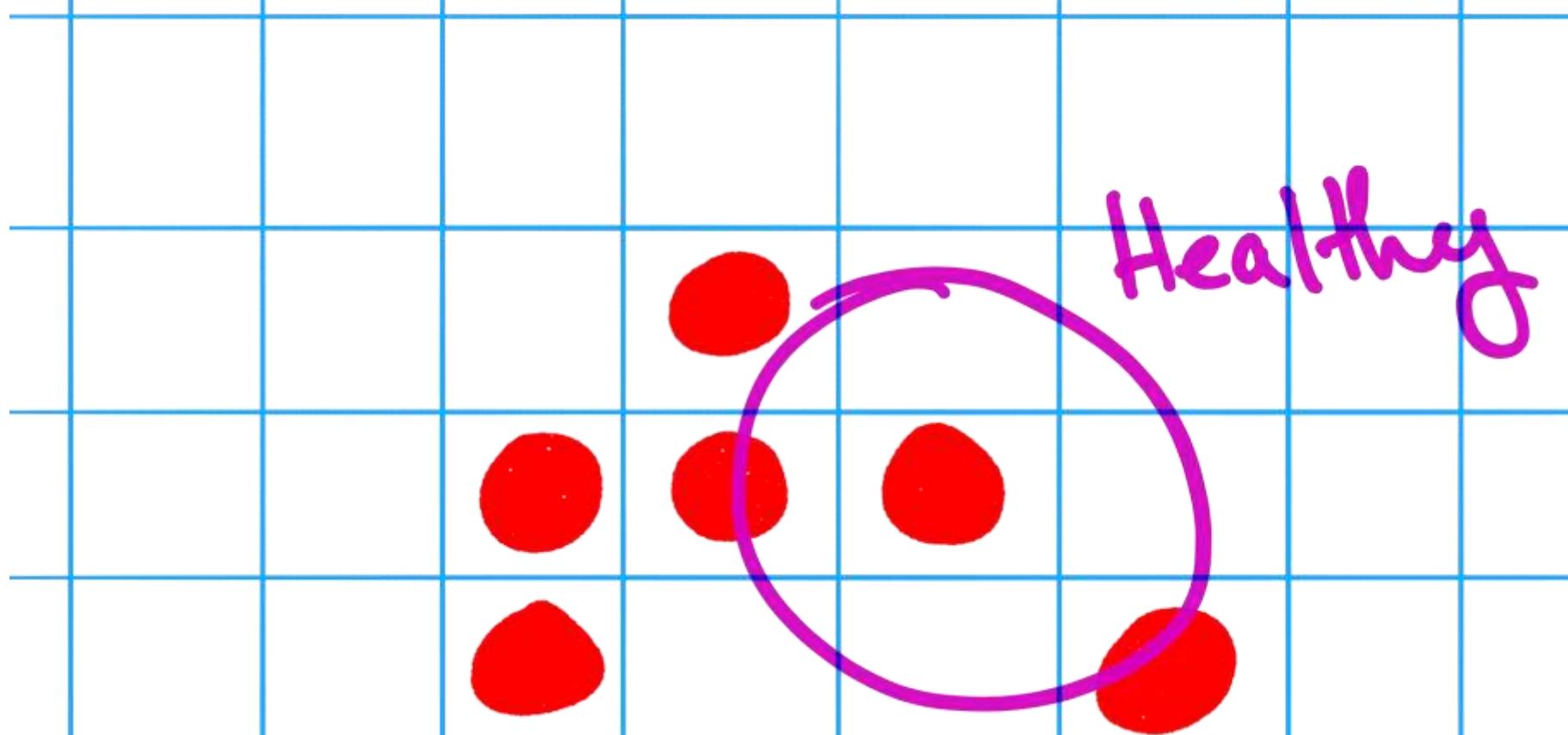
Live w/  $< 2$  = Dead (lonely)

Live w/ 2 or 3 = Alive

Live w/  $> 3$  = Dead (crowded)

Dead w/ 3 = Alive (reproduction)





Live w/  $< 2$  = Dead (lonely)

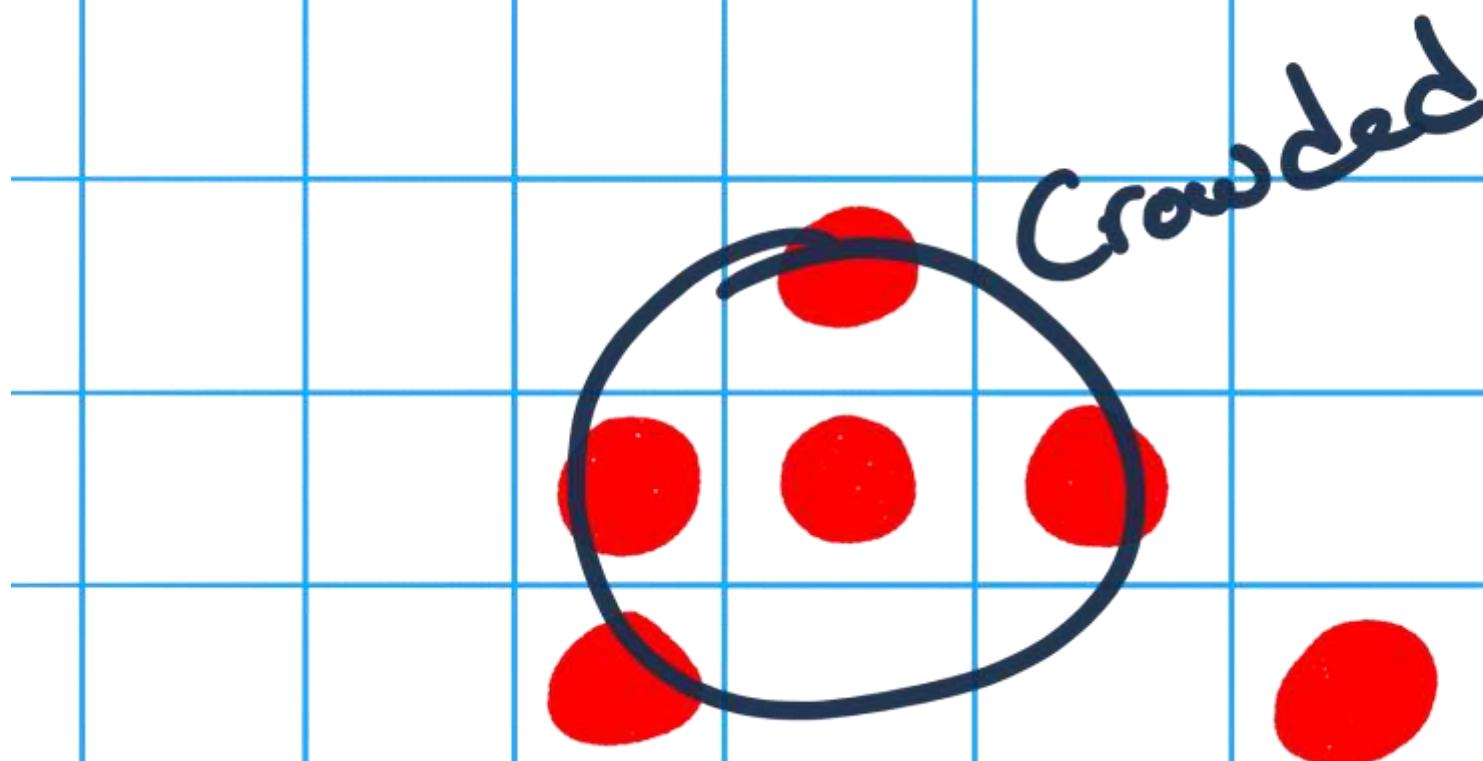
Live w/ 2 or 3 = Alive

Live w/  $> 3$  = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes





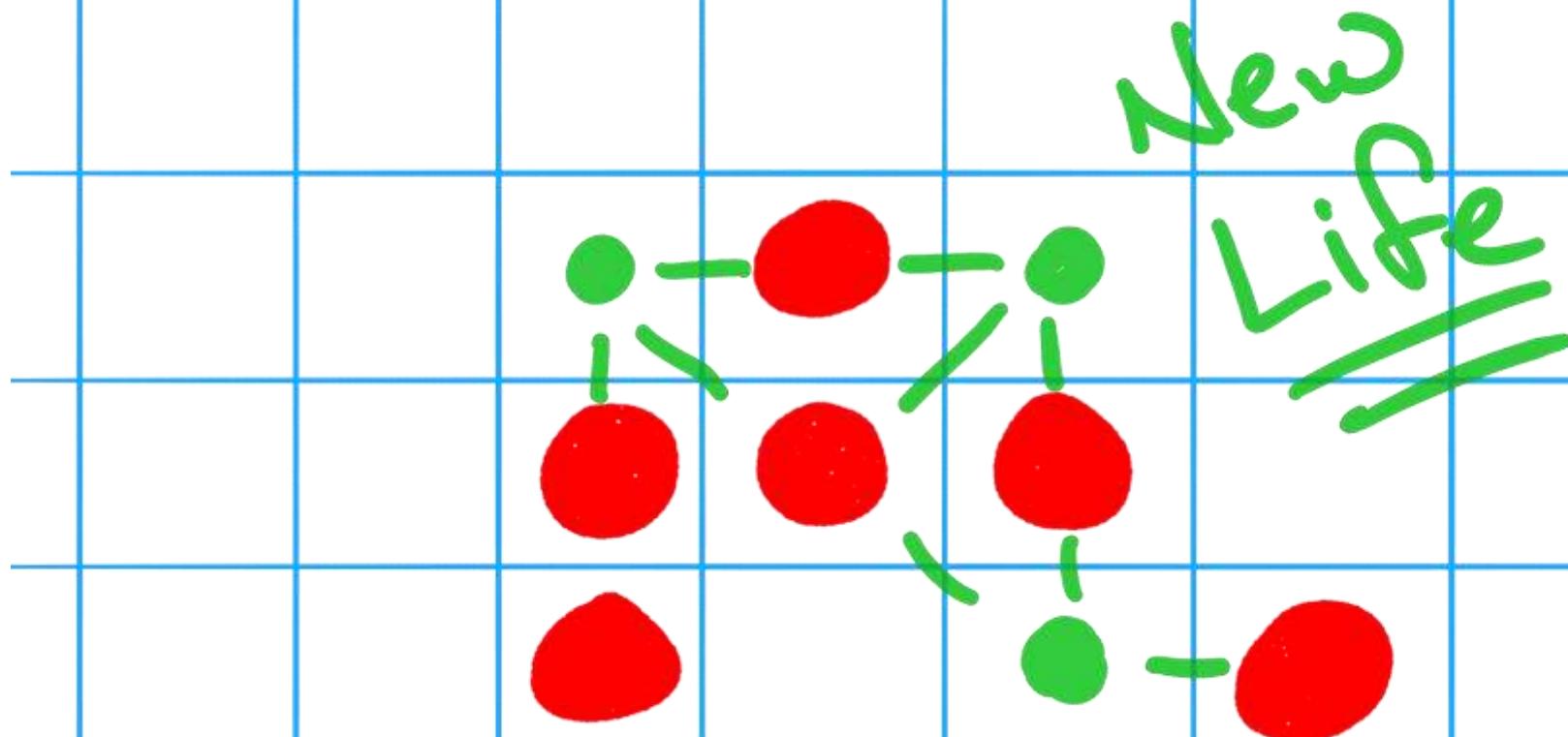
Live w/  $< 2$  = Dead (lonely)

Live w/ 2 or 3 = Alive

Live w/  $> 3$  = Dead (crowded)

Dead w/ 3 = Alive (reproduction)





Live w/  $< 2$  = Dead (lonely)

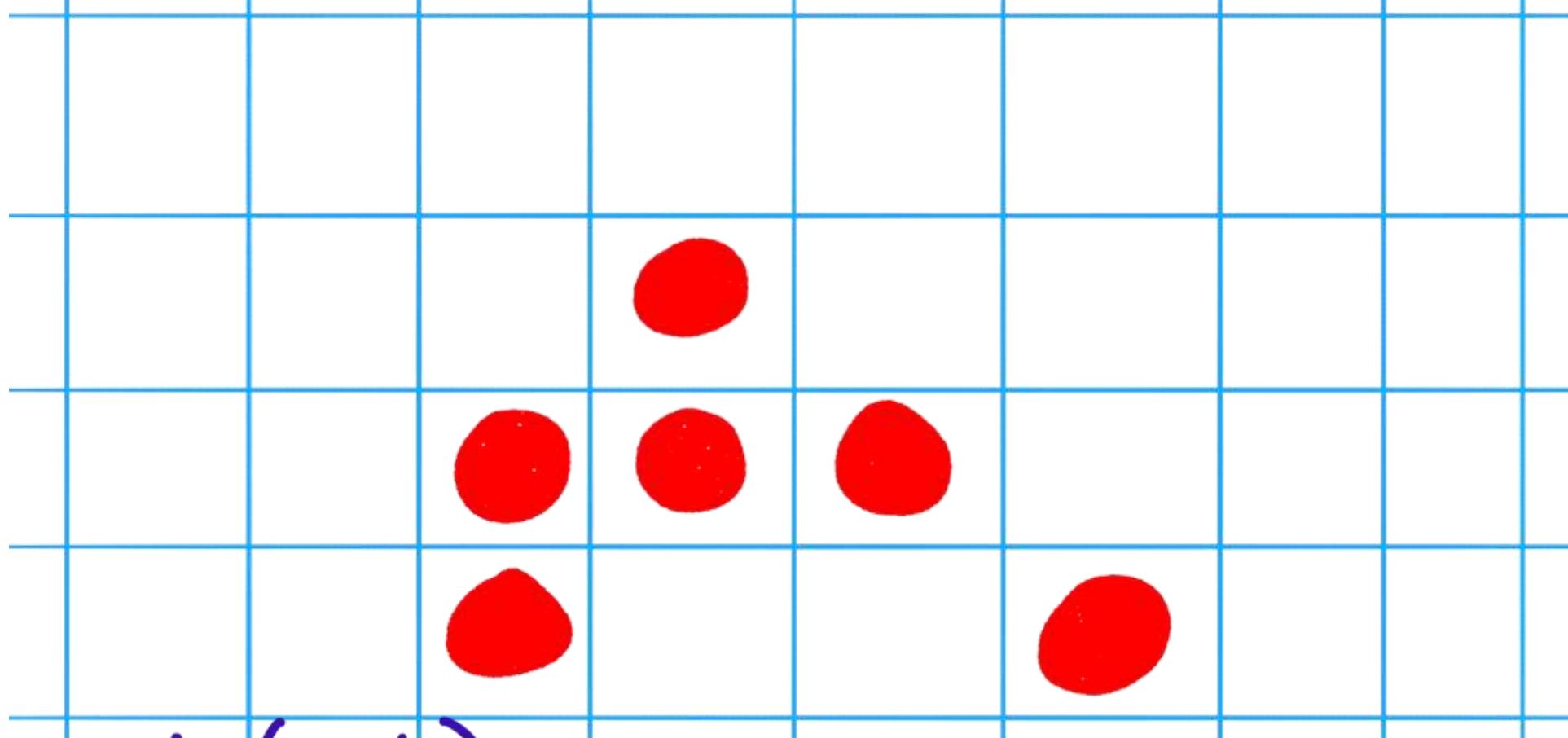
Live w/ 2 or 3 = Alive

Live w/  $> 3$  = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes





Live w/  $< 2$  = Dead (lonely)

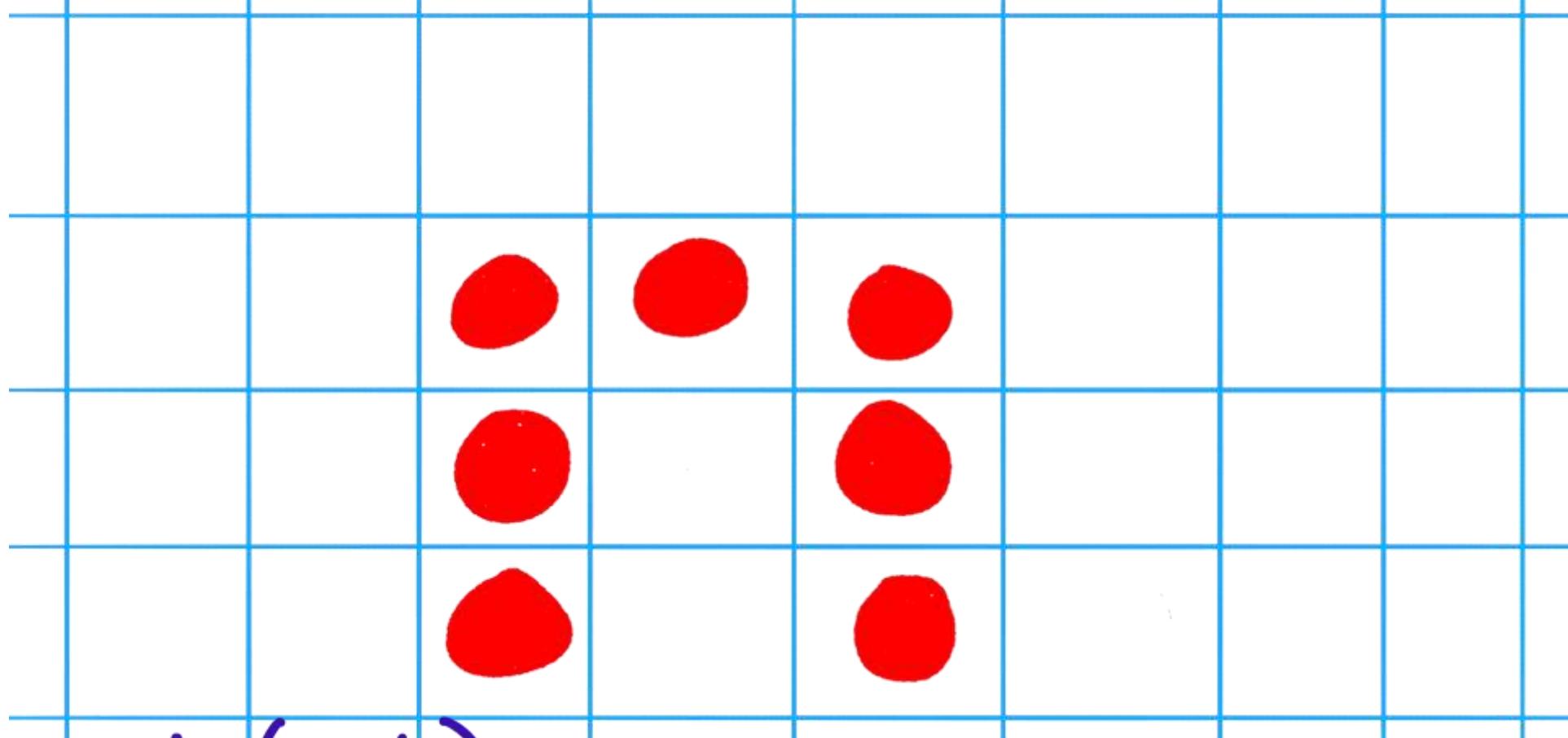
Live w/ 2 or 3 = Alive

Live w/  $> 3$  = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes





Live w/  $< 2$  = Dead (lonely)

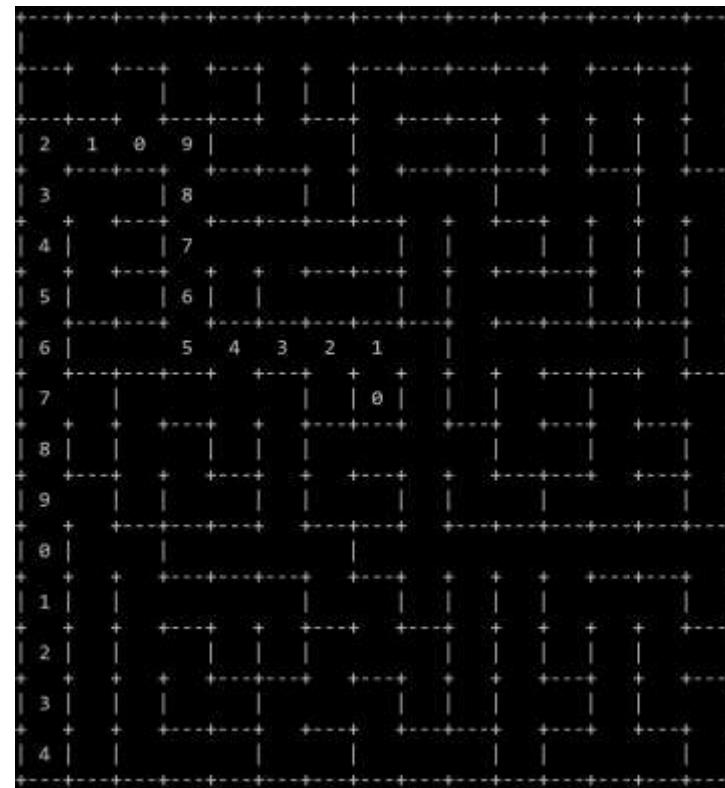
Live w/ 2 or 3 = Alive

Live w/  $> 3$  = Dead (crowded)

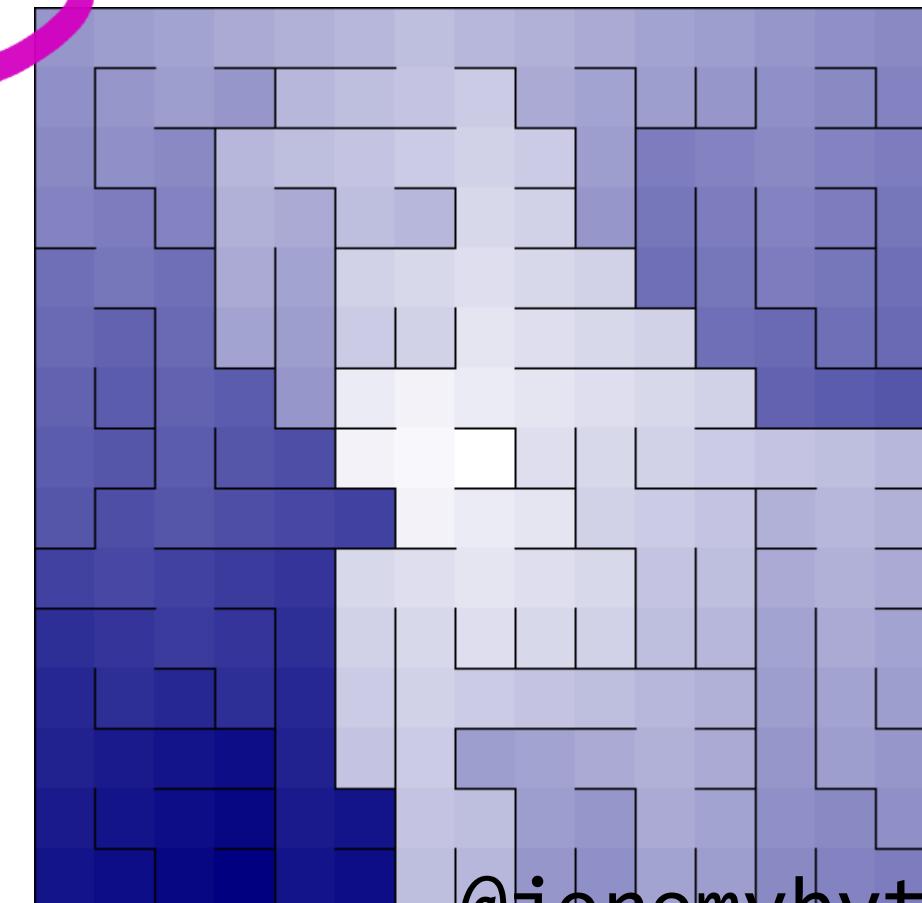
Dead w/ 3 = Alive (reproduction)

Hard to  
Run Parallel

# Mazes

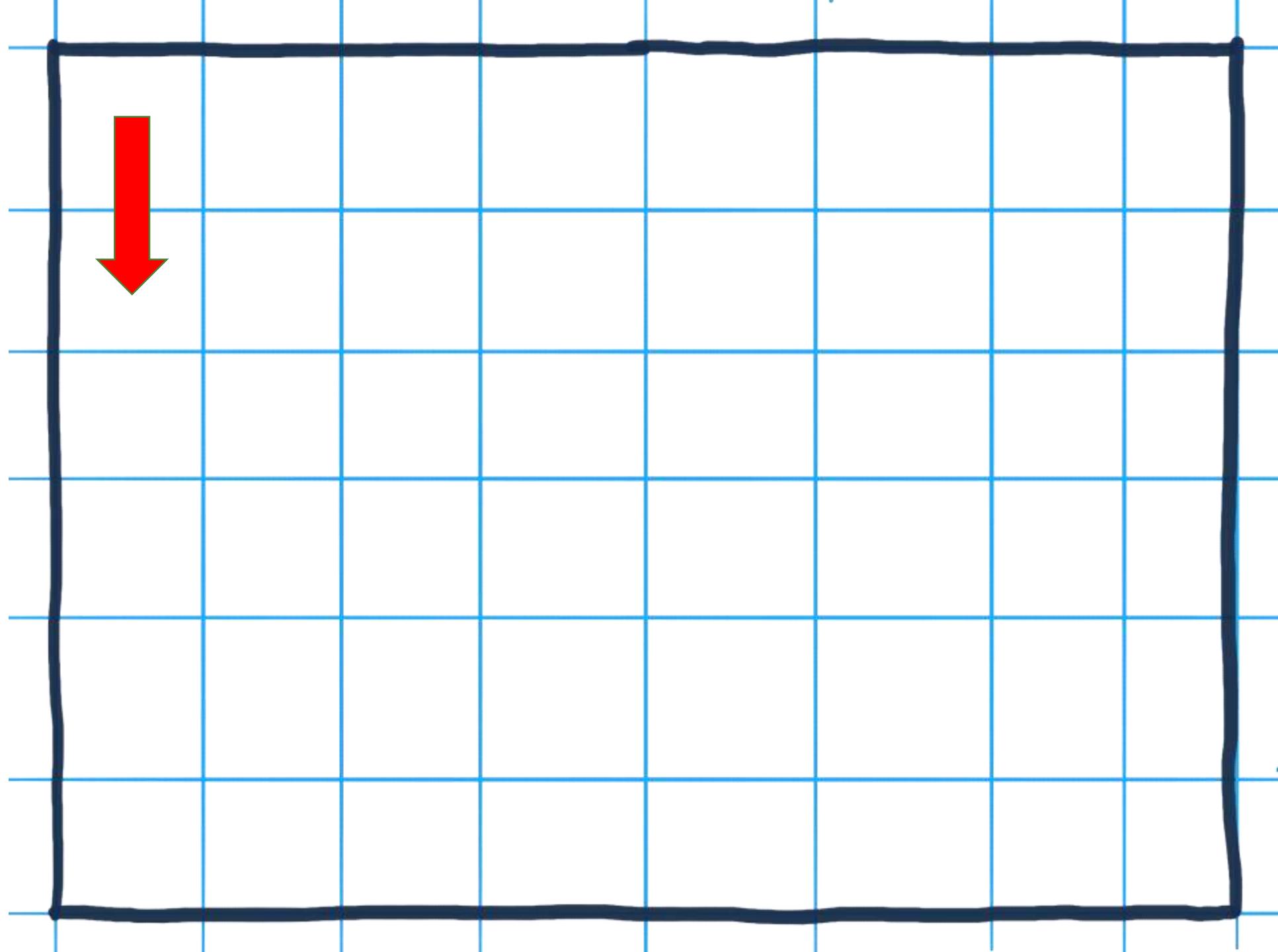


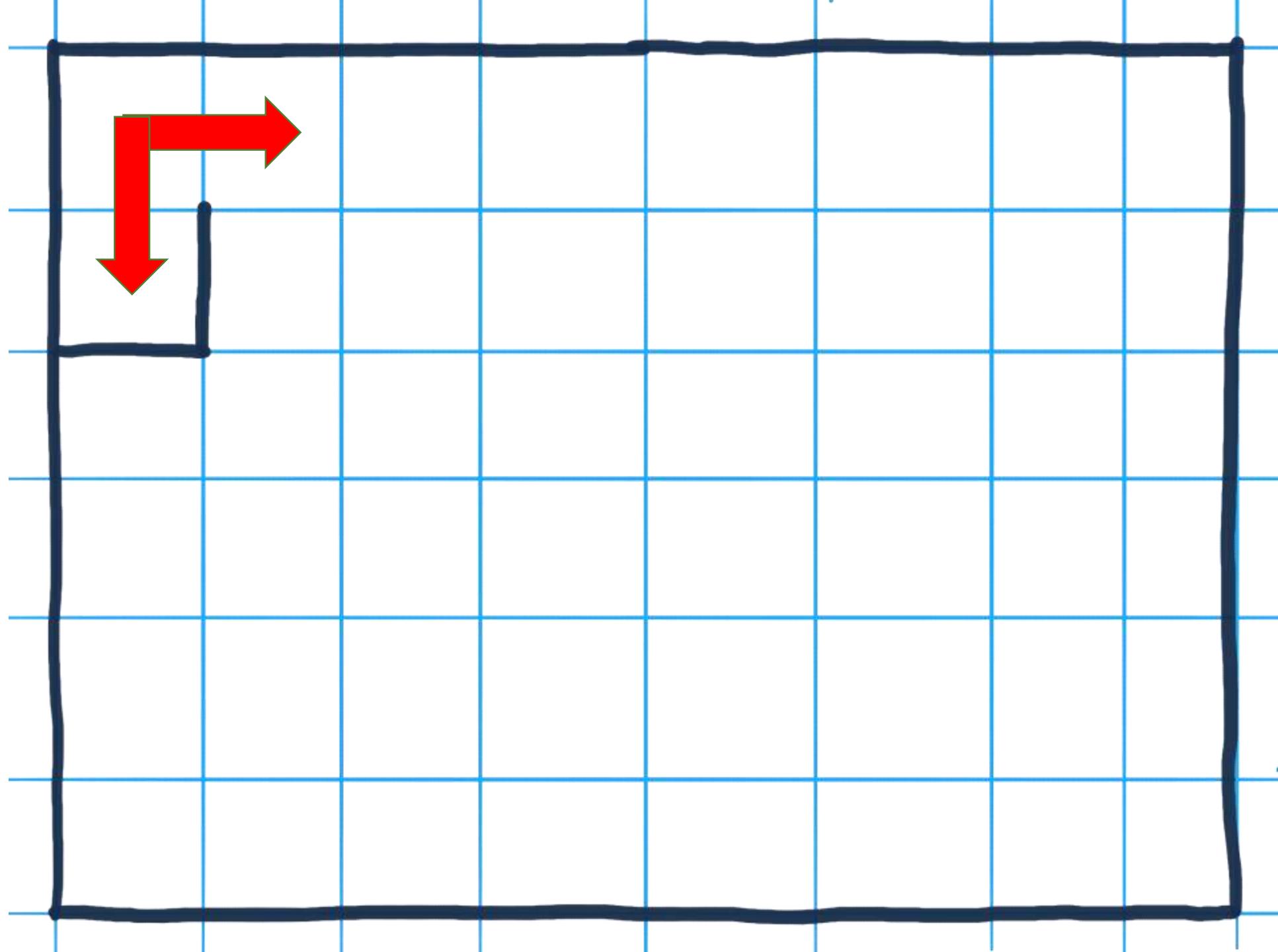
Hard to Run Parallel

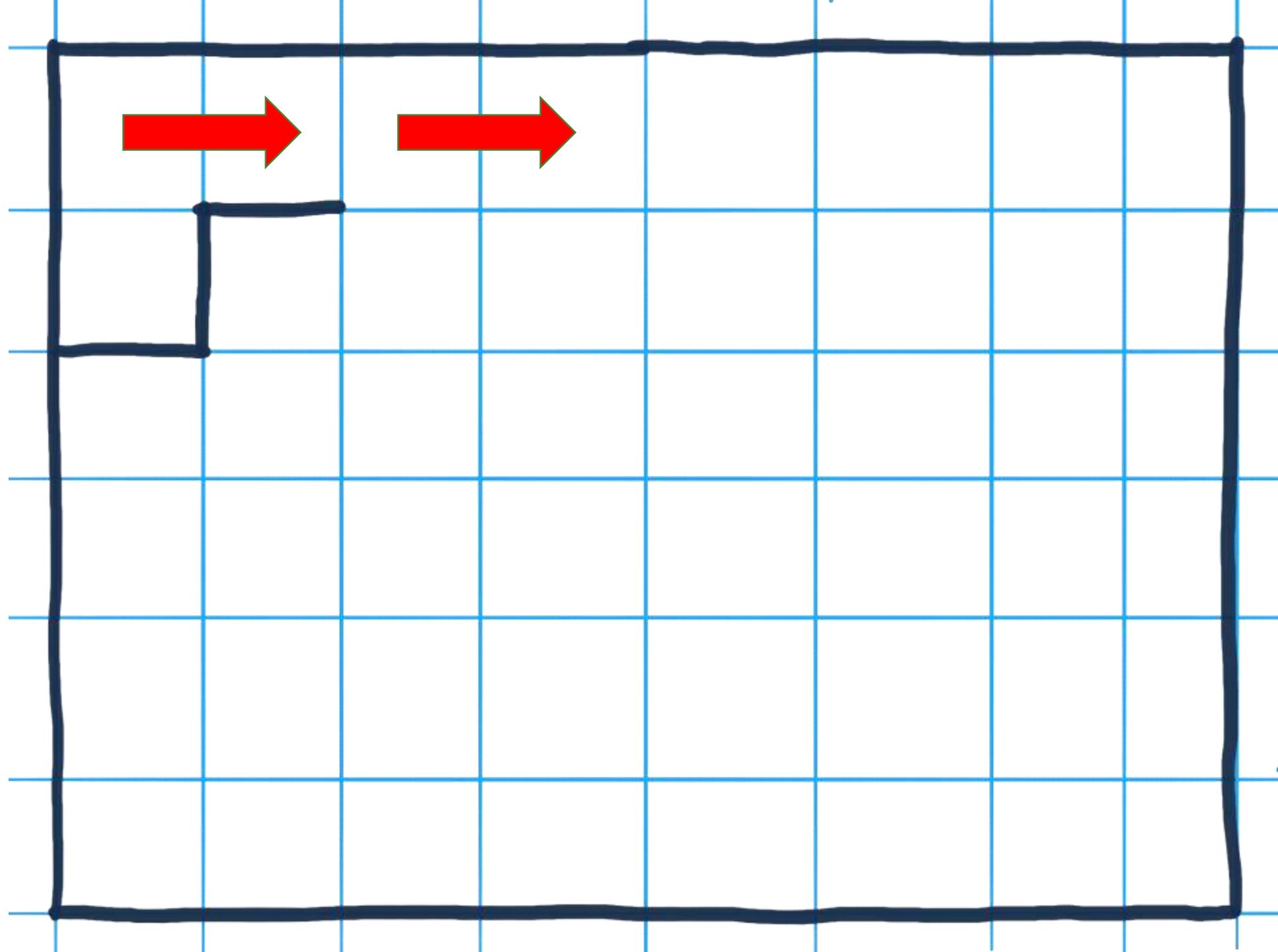


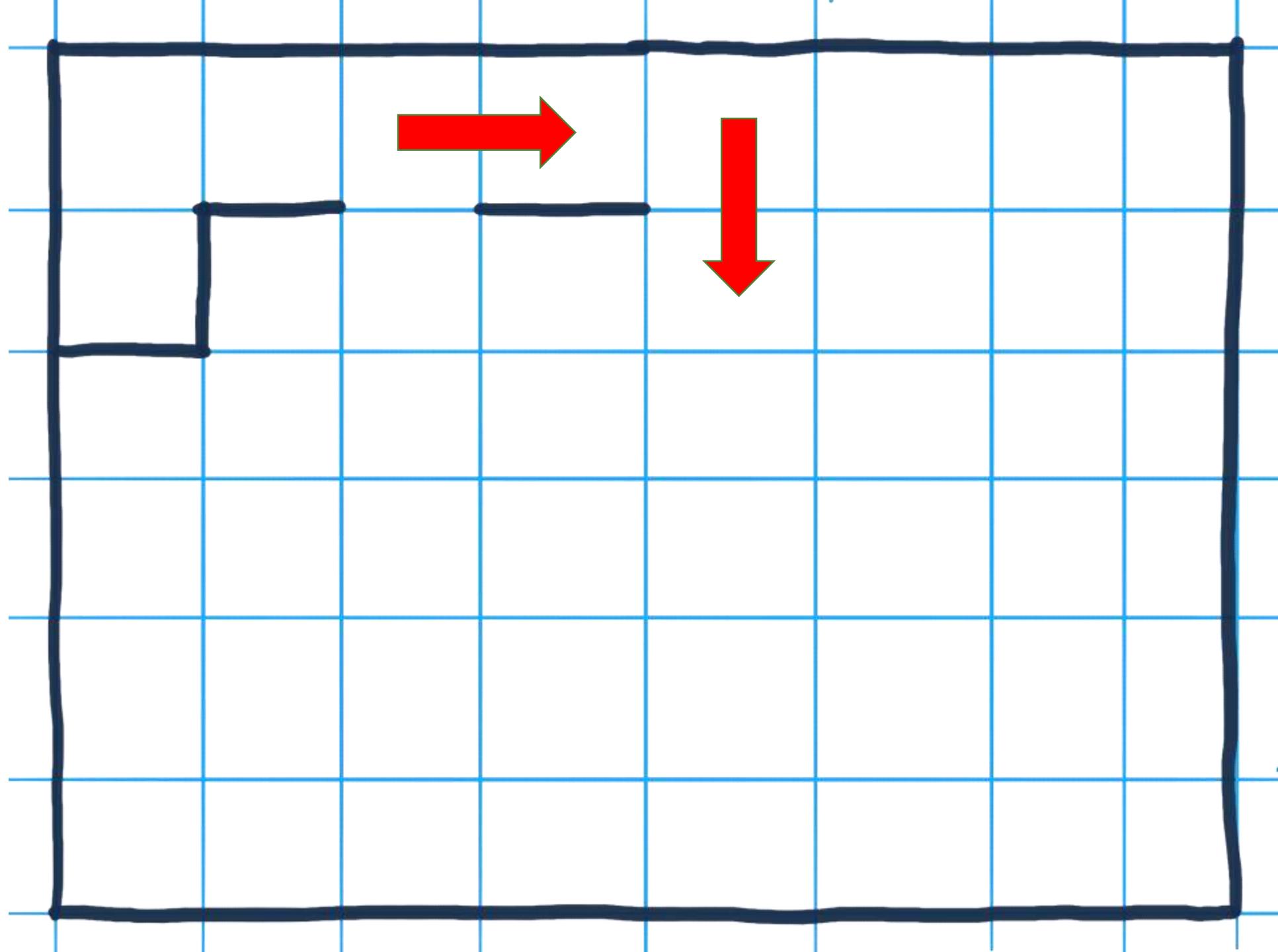
@jeremybytes

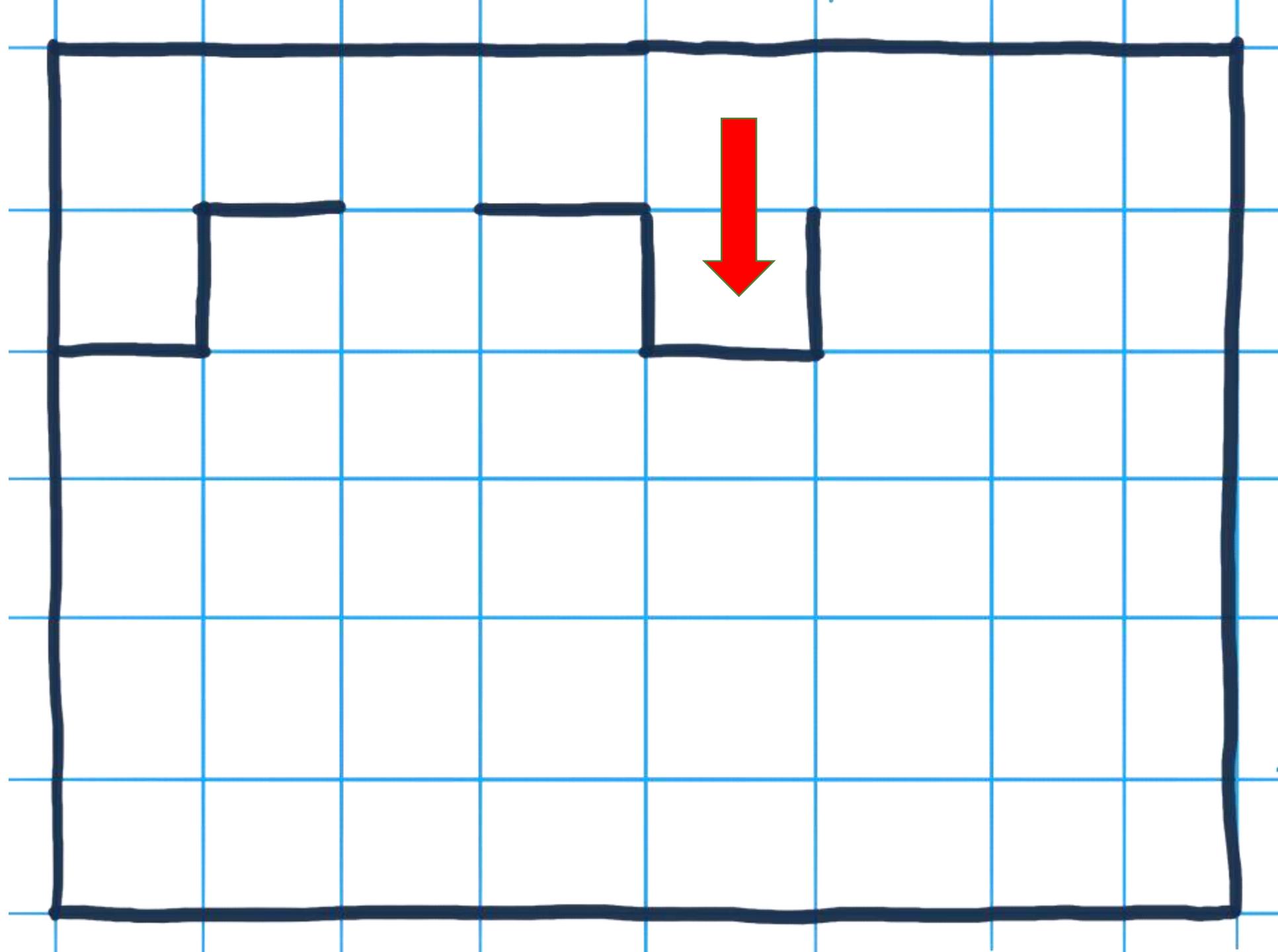


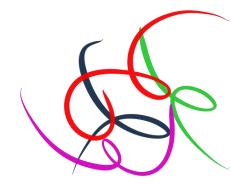
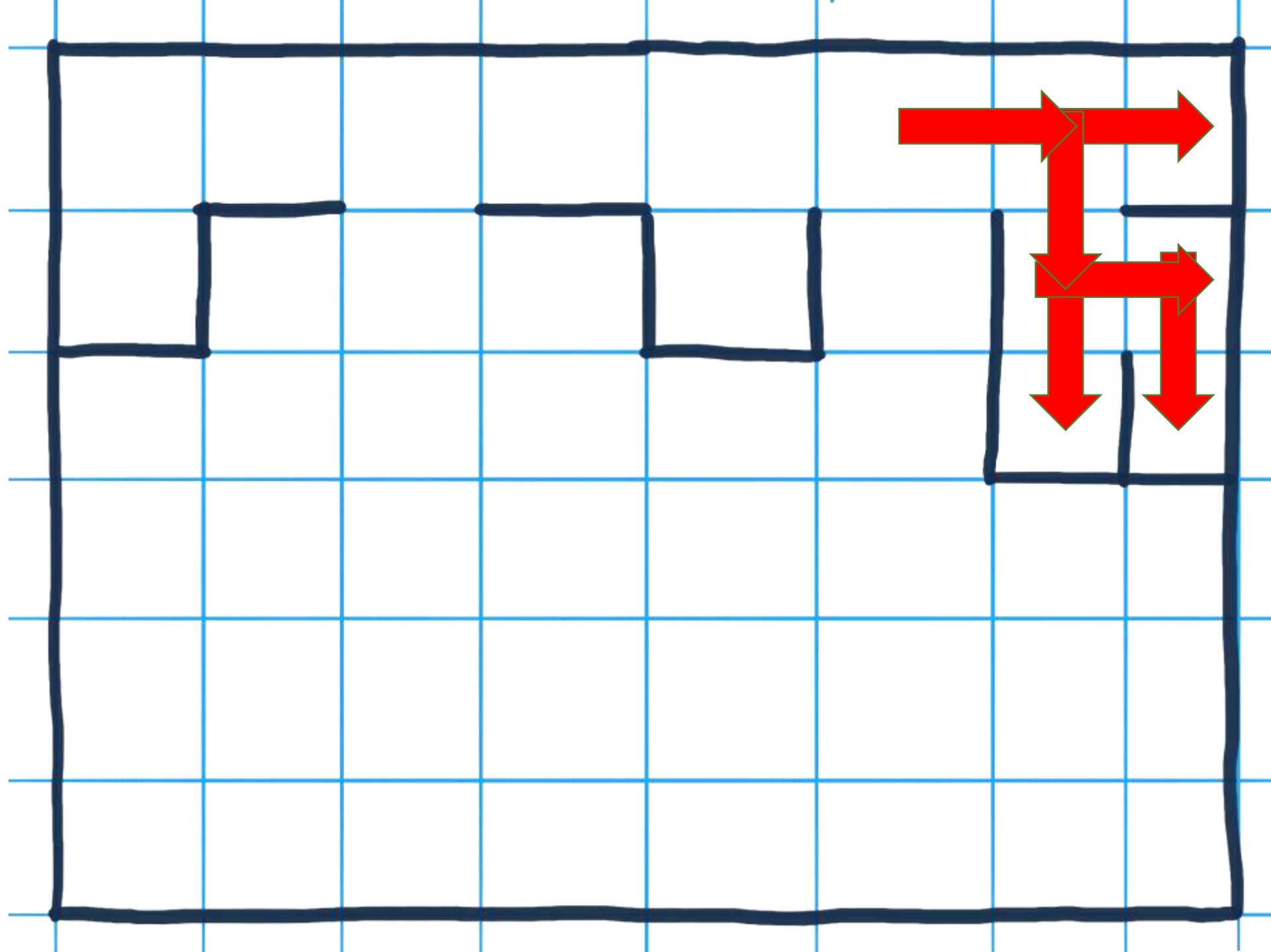


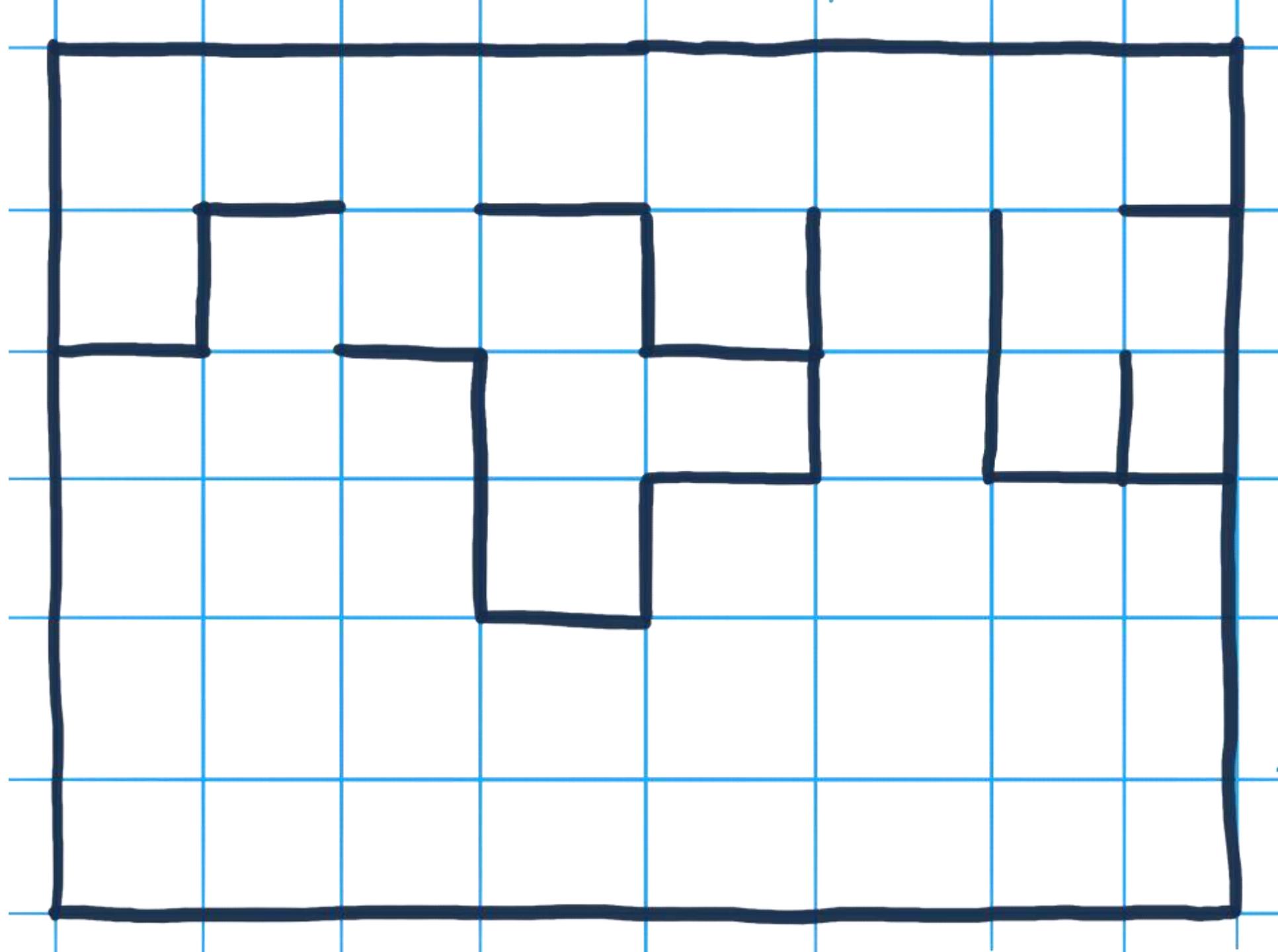


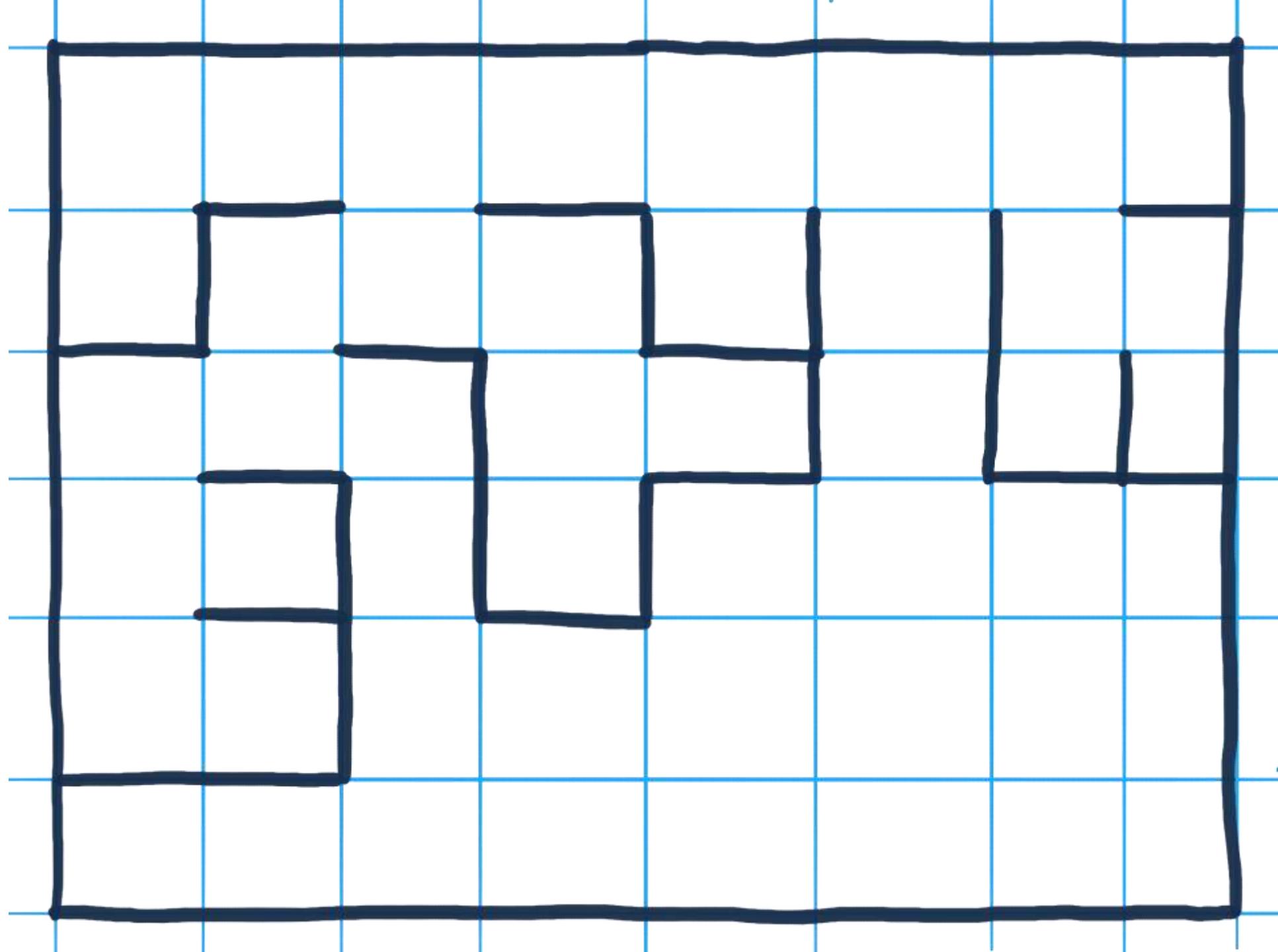


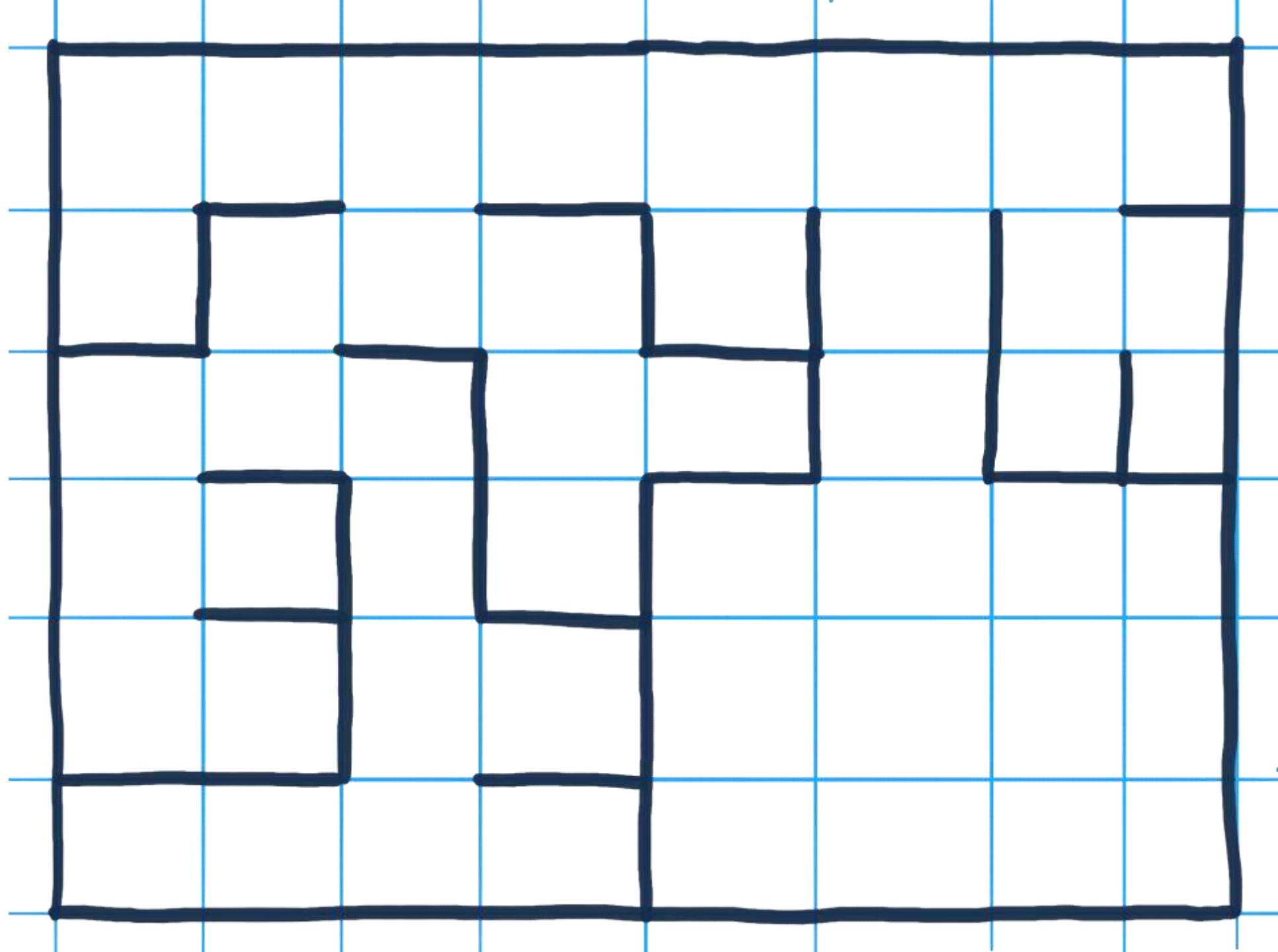


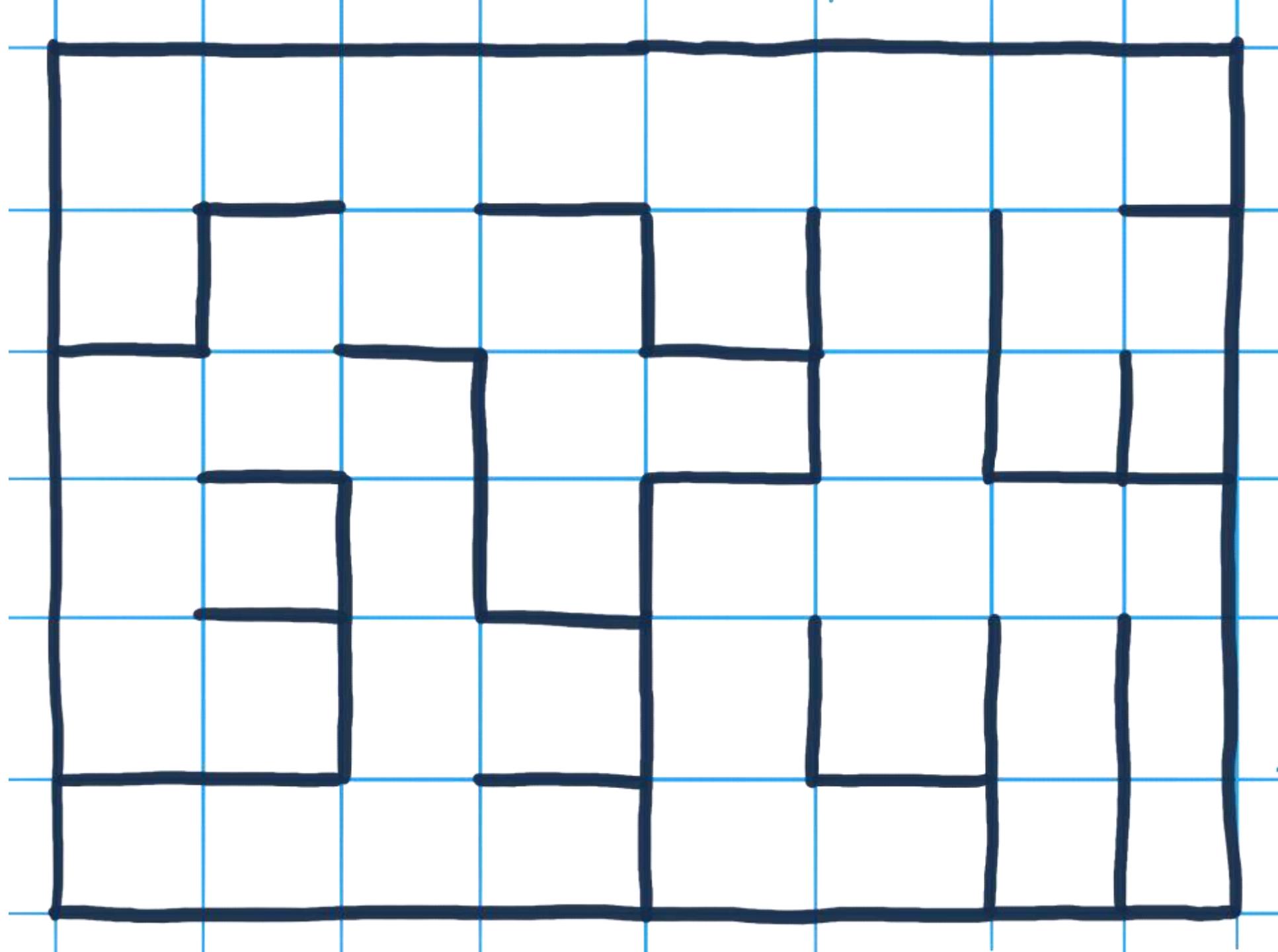












# Parallel Tasks

# Digit Recognition

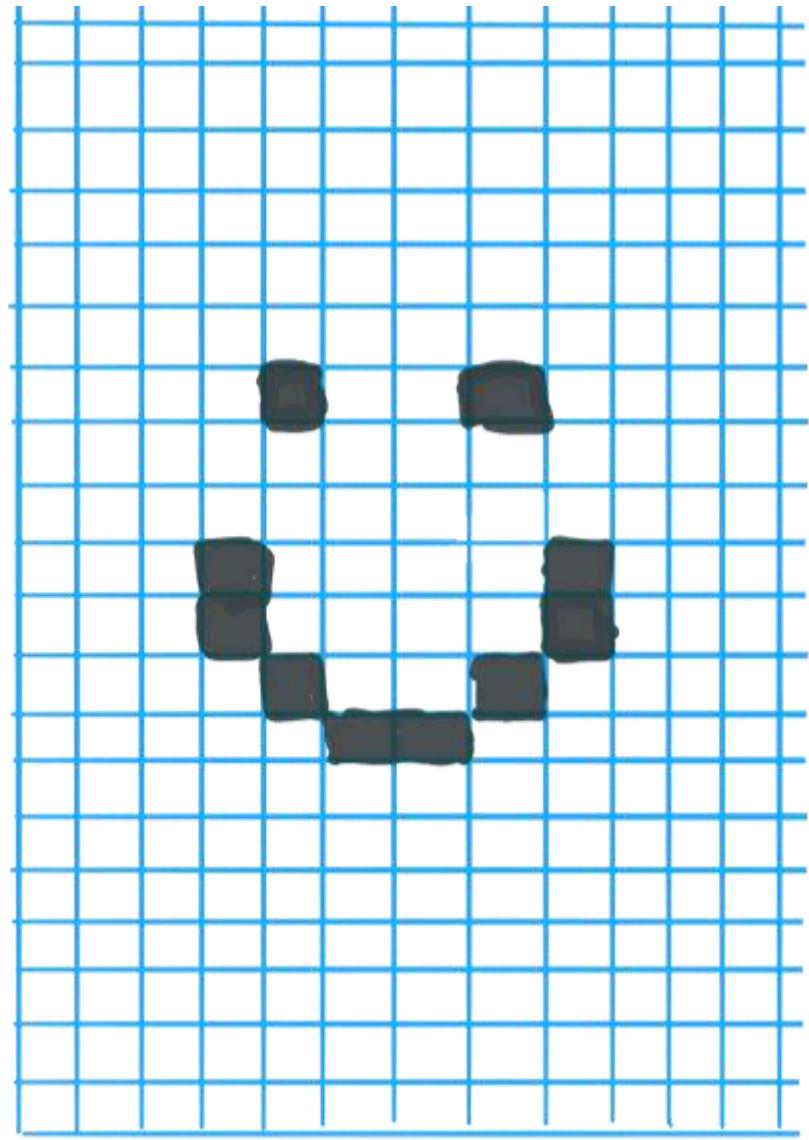
Actual: 9 - Prediction: 4

Single-Threaded Manhattan Classifier										Parallel Manhattan Classifier									
Duration (seconds): 47 Errors: 7					Offset: 8000					Duration (seconds): 7 Errors: 3					Go				
Record Count: 242										Duration (seconds): 7 Errors: 3									
8	6	6	4	4	8	4	2	2	3	6	6	9	9	9	8	6	4	9	9
5	6	1	1	6	6	2	7	8	3	3	9	2	3	4	4	2	3	3	4
6	6	7	7	2	2	4	4	1	1	1	2	2	2	1	1	2	2	1	2
5	3	0	0	1	1	2	2	8	8	3	3	2	2	5	5	5	5	4	4
7	7	6	6	7	7	7	9	9	1	1	9	9	1	1	5	5	2	2	2
0	0	7	7	3	3	0	0	2	6	6	7	0	0	8	6	6	4	4	4
0	0	8	3	2	2	4	4	8	8	0	0	5	4	1	8	9	9	1	1
3	3	5	5	8	8	4	4	1	9	3	3	3	3	2	2	6	6	7	7
7	7	0	0	2	2	5	5	2	2	0	0	1	2	5	5	4	4	4	4
2	2	0	0	2	2	6	6	7	7	3	3	4	4	8	8	4	4	4	4
2	2	4	4	9	9	2	2	1	1	6	6	5	5	7	7	1	1	1	1
3	3	3	3	9	9	7	7	0	0	7	7	3	3	3	3	3	2	2	2
4	4	9	9	3	3	4	4	6	5	8	8	0	0	3	3	5	6	8	8
1	1	0	0	6	5	8	8	1	1	1	7	7	9	9	9	3	3	6	6
0	0	7	7	4	4	3	3	6	6	8	8	7	7	8	8	9	9	0	2
4	4	3	3	2	2	2	2	8	8	9	9	5	5	9	9	4	4	5	5
3	3	1	1	5	5	1	1	9	9	4	4	7	7	9	9	6	6	4	2
1	1	1	8	4	4	9	3	2	2	6	6	6	1	7	4	4	2	2	5
8	8	5	5	0	0	8	8	3	3	0	0	9	9	3	3	7	7	7	7
4	4	0	0	9	9	8	8	5	5	5	5	1	1	7	7	6	6	3	3
9	9	1	1	5	5	0	0	9	9	9	0	0	6	6	6	6	7	9	6
4	4	9	9	0	0	4	4	0	0	6	6	9	9	7	7	1	1	8	0

@jeremybytes

# Digit Recognition

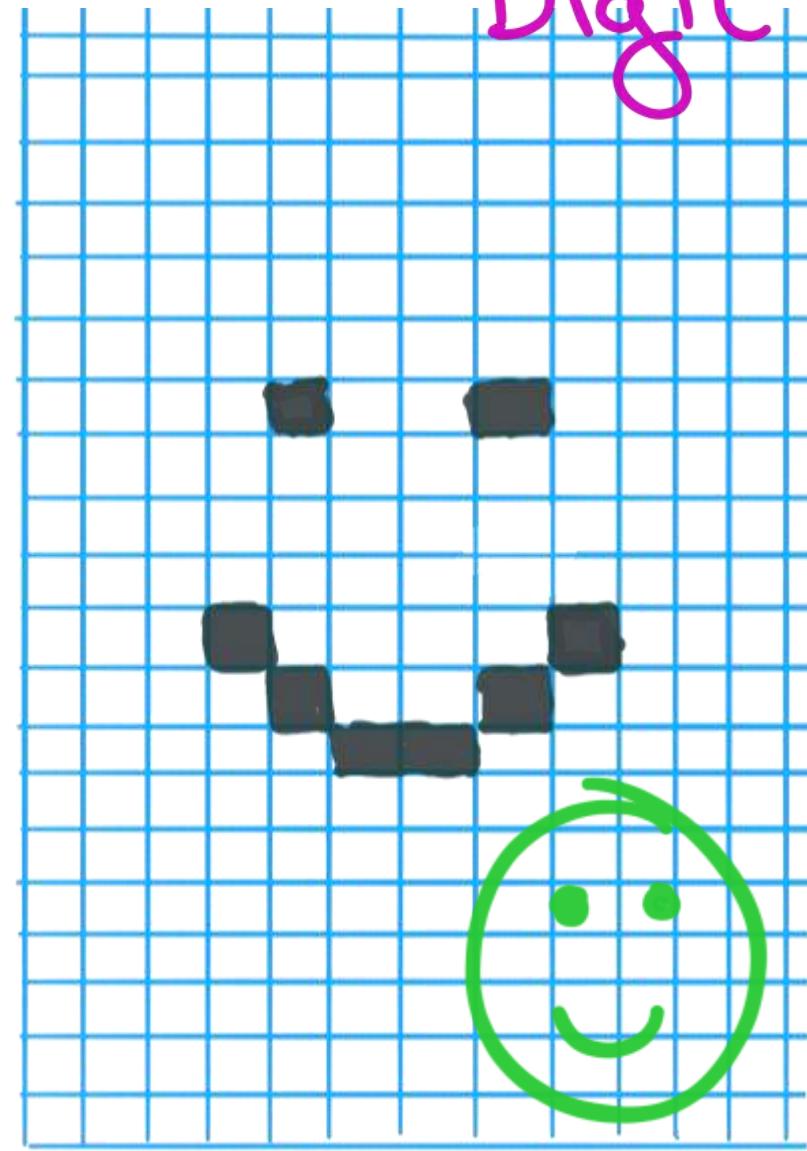
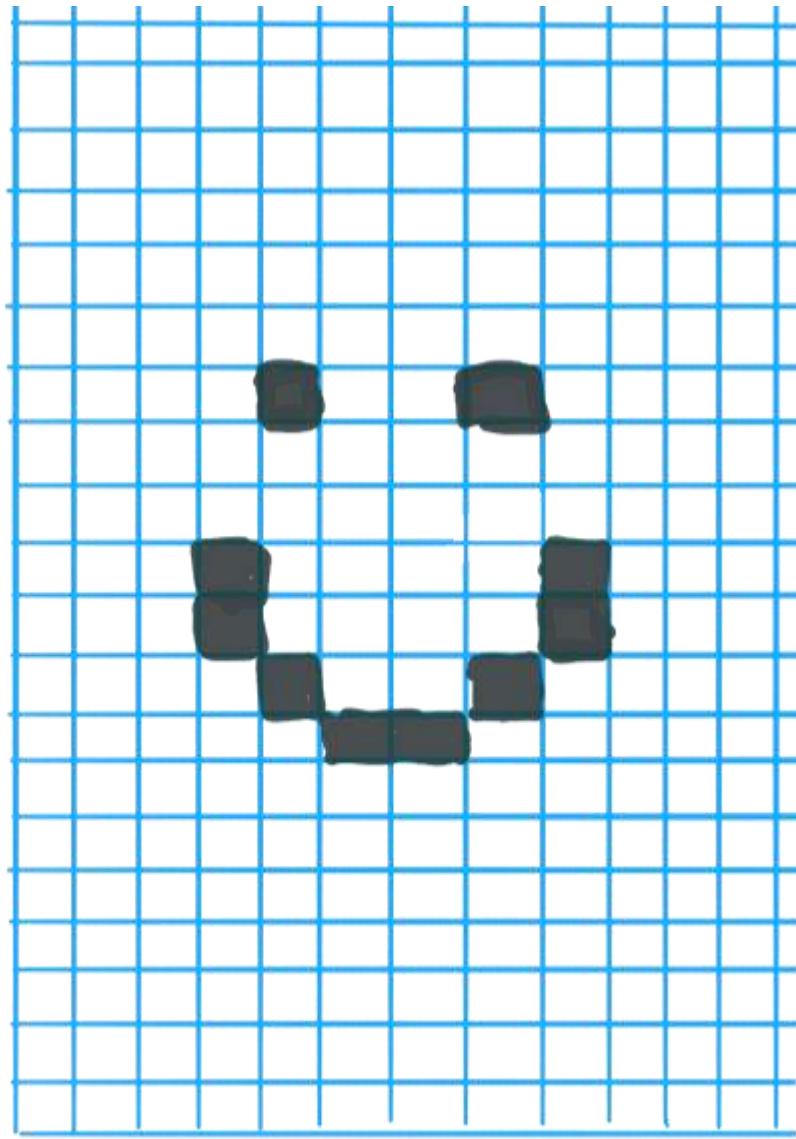
@jeremybytes



# Digit Recognition

@jeremybytes

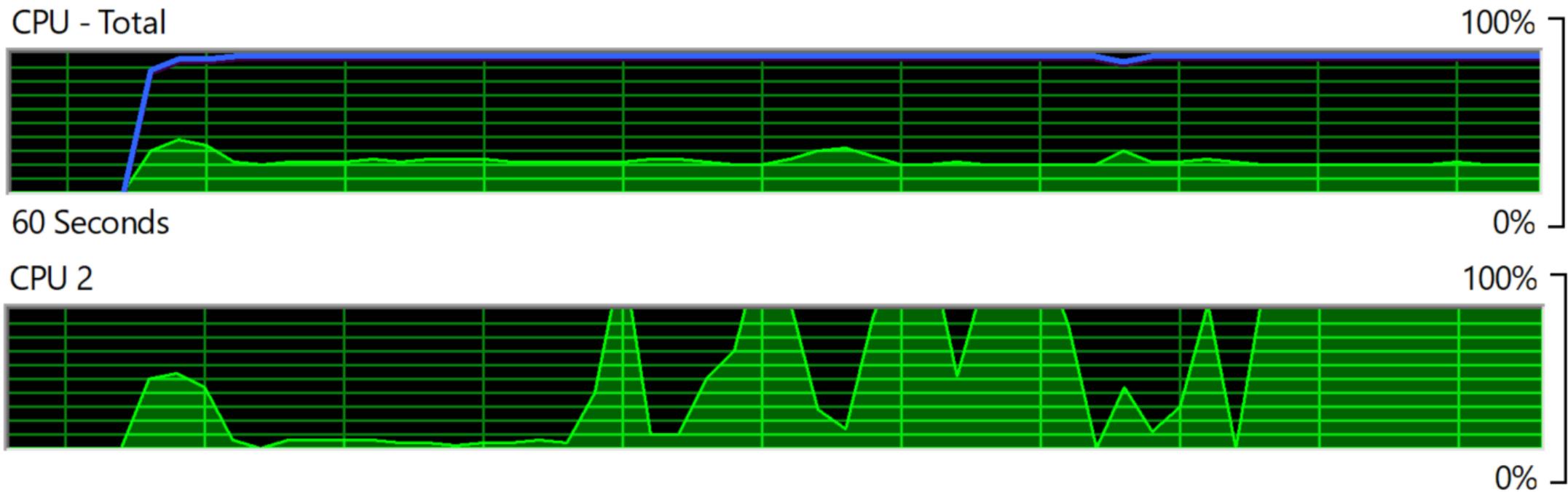




Digit Recognition

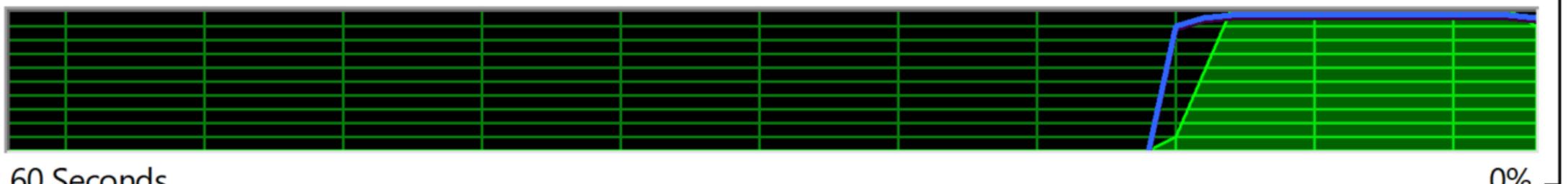
@jeremybytes



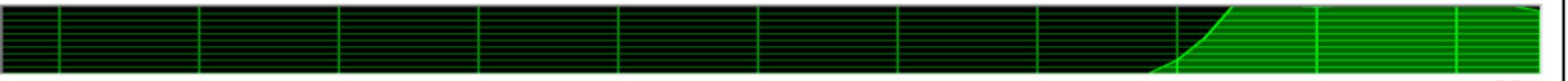


@jeremybytes

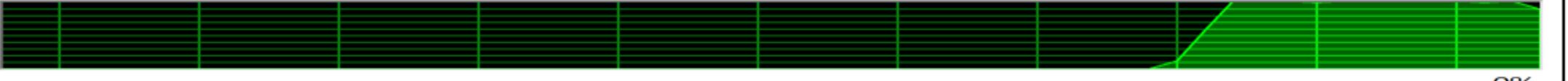
CPU - Total



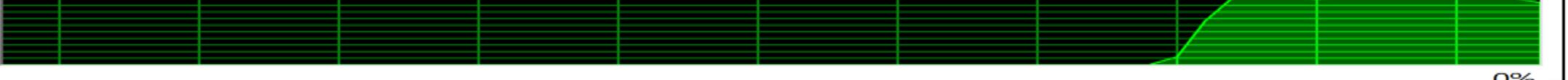
CPU 0



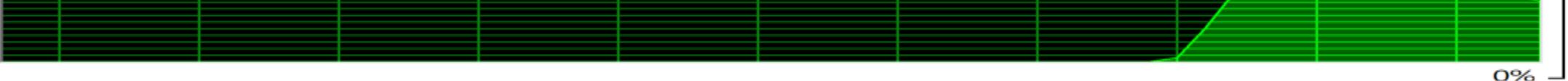
CPU 1



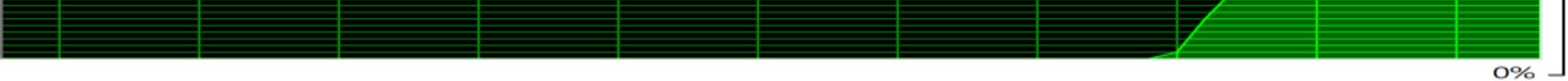
CPU 2



CPU 3



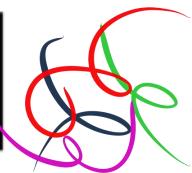
CPU 4



CPU 5



@jeremybytes



Measure

Measure

Measure

@jeremybytes



# Resources

@jeremybytes 

Resources

# Parallel Programming with Microsoft .NET

@jeremybytes



# Parallel Programming with Microsoft .NET

## Selecting the Right Pattern

To select the relevant pattern, use the following table.

Application characteristic	Relevant pattern
Do you have sequential loops where there's no communication among the steps of each iteration?	The Parallel Loop pattern (Chapter 2). Parallel loops apply an independent operation to multiple inputs simultaneously.
Do you have distinct operations with well-defined control dependencies? Are these operations largely free of serializing dependencies?	The Parallel Task pattern (Chapter 3) Parallel tasks allow you to establish parallel control flow in the style of fork and join.
Do you need to summarize data by applying some kind of combination operator? Do you have loops with steps that are not fully independent?	The Parallel Aggregation pattern (Chapter 4) Parallel aggregation introduces special steps in the algorithm for merging partial results. This pattern expresses a reduction operation and includes map/reduce as one of its variations.
Does the ordering of steps in your algorithm depend on data flow constraints?	The Futures pattern (Chapter 5) Futures make the data flow dependencies between tasks explicit. This pattern is also referred to as the Task Graph pattern.
Does your algorithm divide the problem domain dynamically during the run? Do you operate on recursive data structures such as graphs?	The Dynamic Task Parallelism pattern (Chapter 6) This pattern takes a divide-and-conquer approach and spawns new tasks on demand.
Does your application perform a sequence of operations repetitively? Does the input data have streaming characteristics? Does the order of processing matter?	The Pipelines pattern (Chapter 7) Pipelines consist of components that are connected by queues, in the style of producers and consumers. All the components run in parallel even though the order of inputs is respected.

# Resources

- GitHub

<https://github.com/jeremybytes/parallel-programming>

- Parallel Programming w/ Microsoft .NET

[https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff963553\(v=pandp.10\)](https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff963553(v=pandp.10))

- Task & Await

<http://www.jeremybytes.com/Demos.aspx#TaskAndAwait>

- Presentation Links

<http://www.jeremybytes.com/Demos.aspx#RunFaster>



Thank You



*jeremybytes.com*

@jeremybytes

