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?

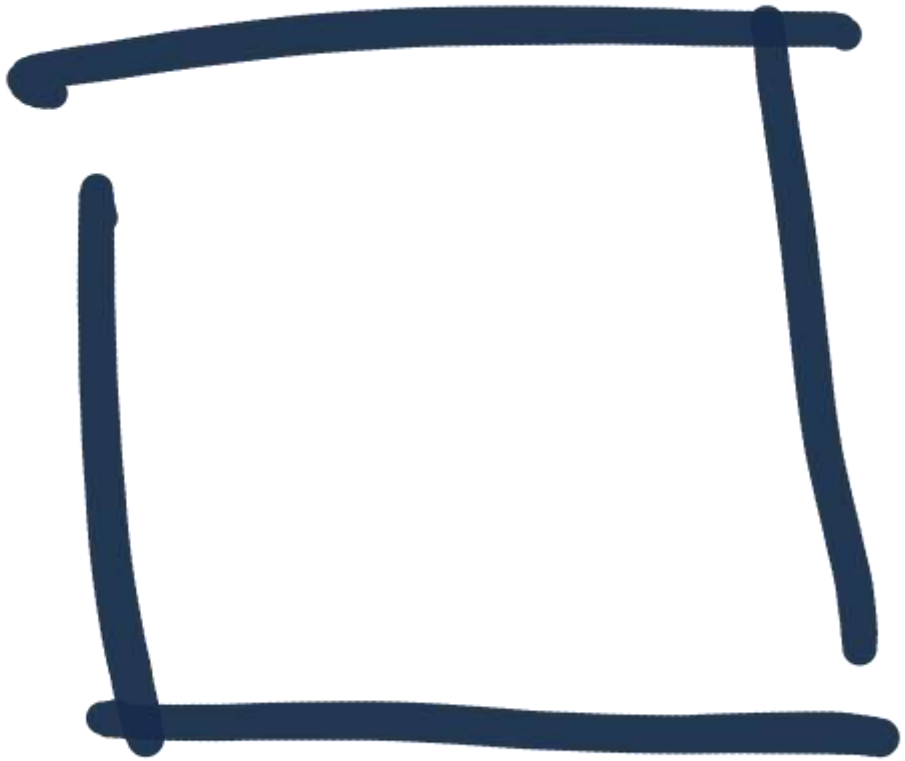
@jeremybytes

Why?

CPUs are getting Bigger

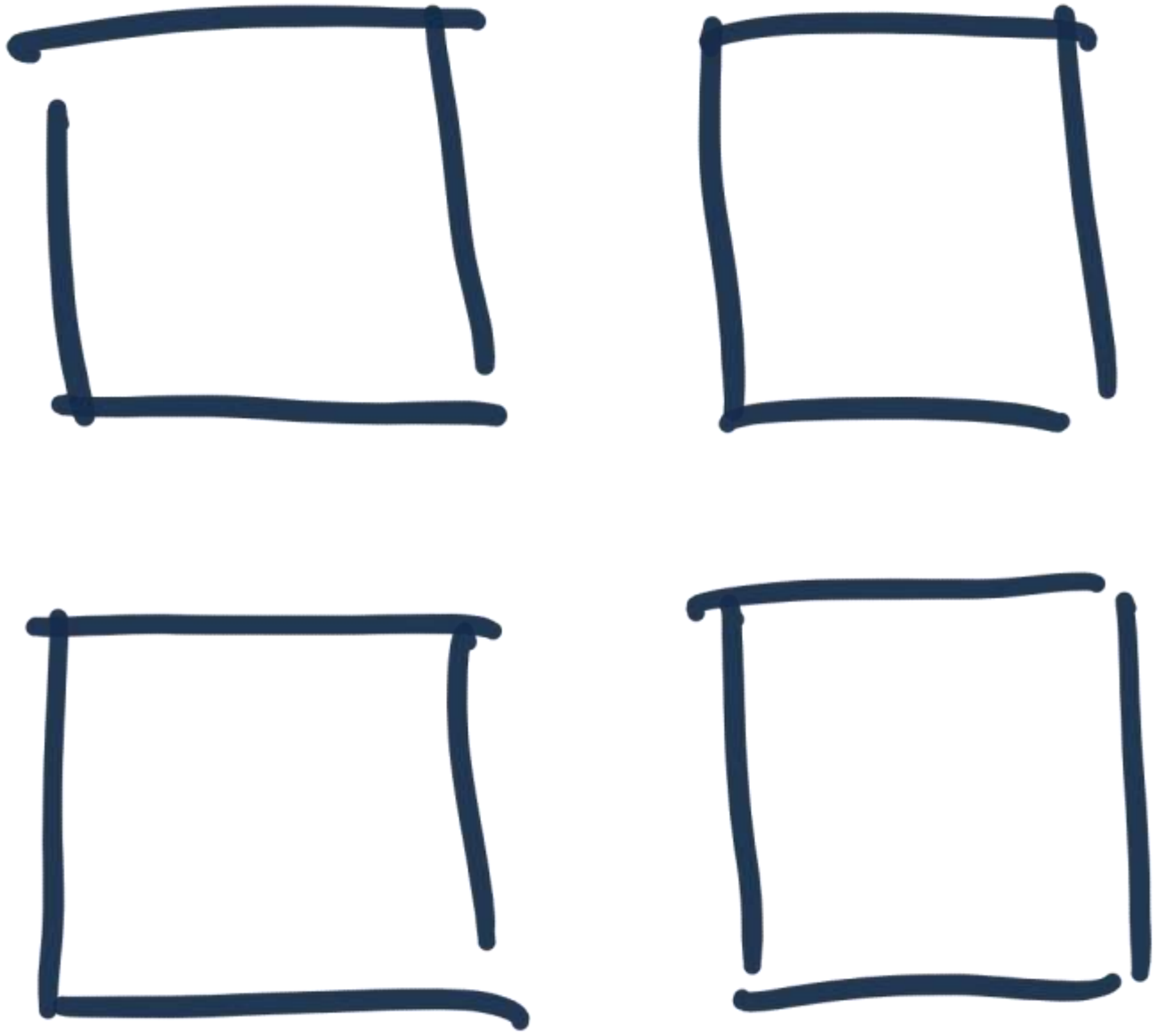
Not Faster

@jeremybytes

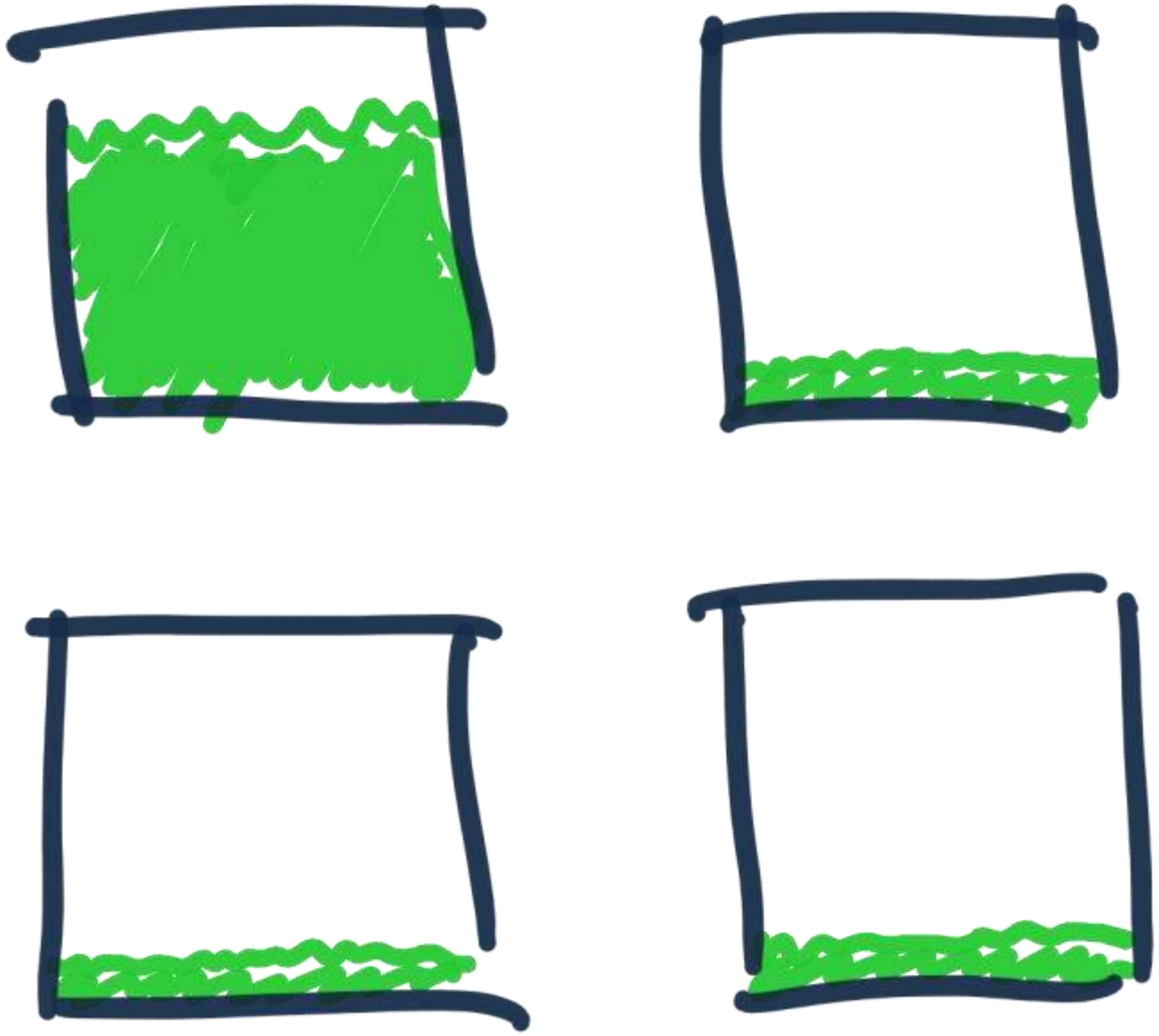


@jeremybytes

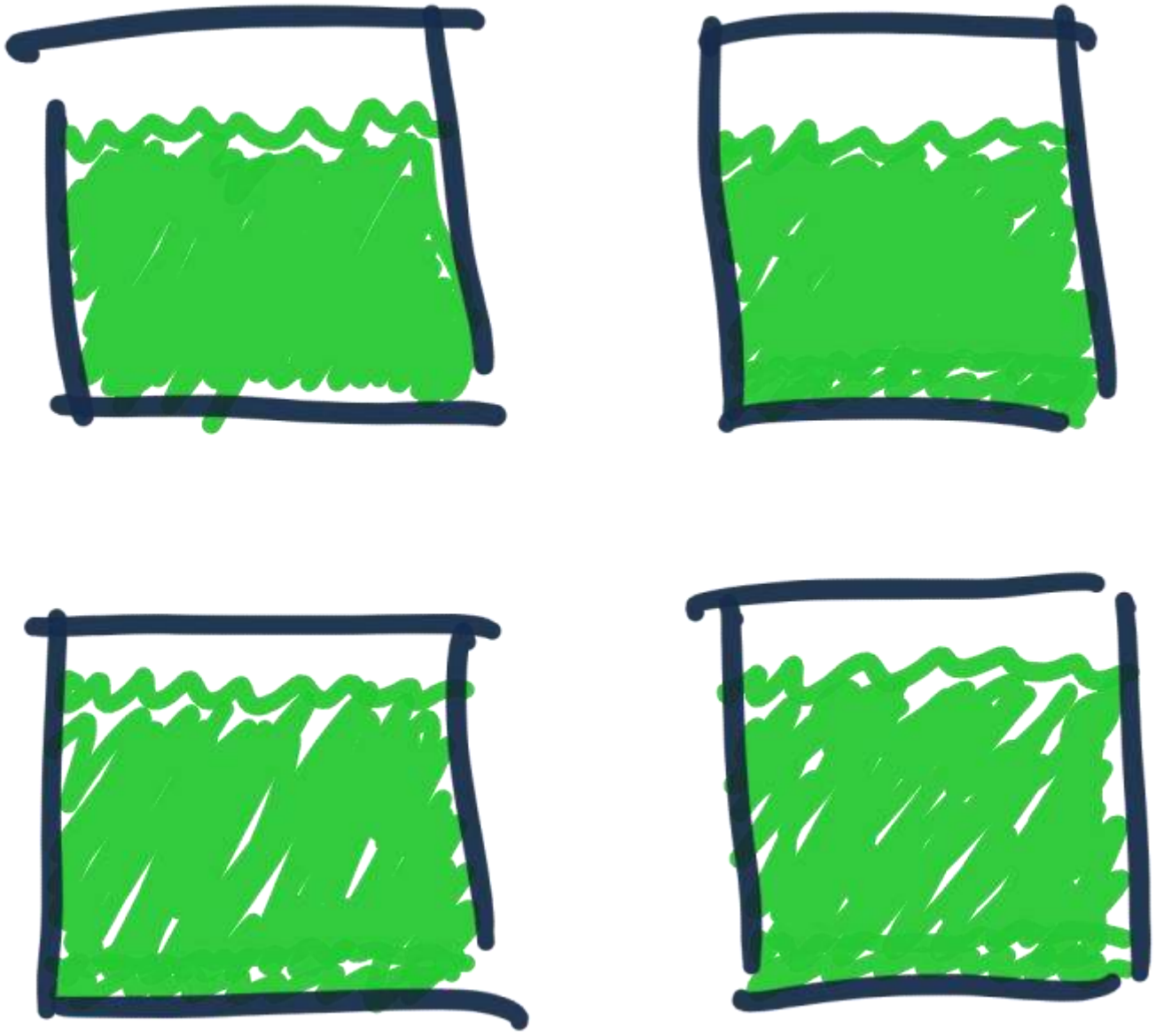
@jeremybytes



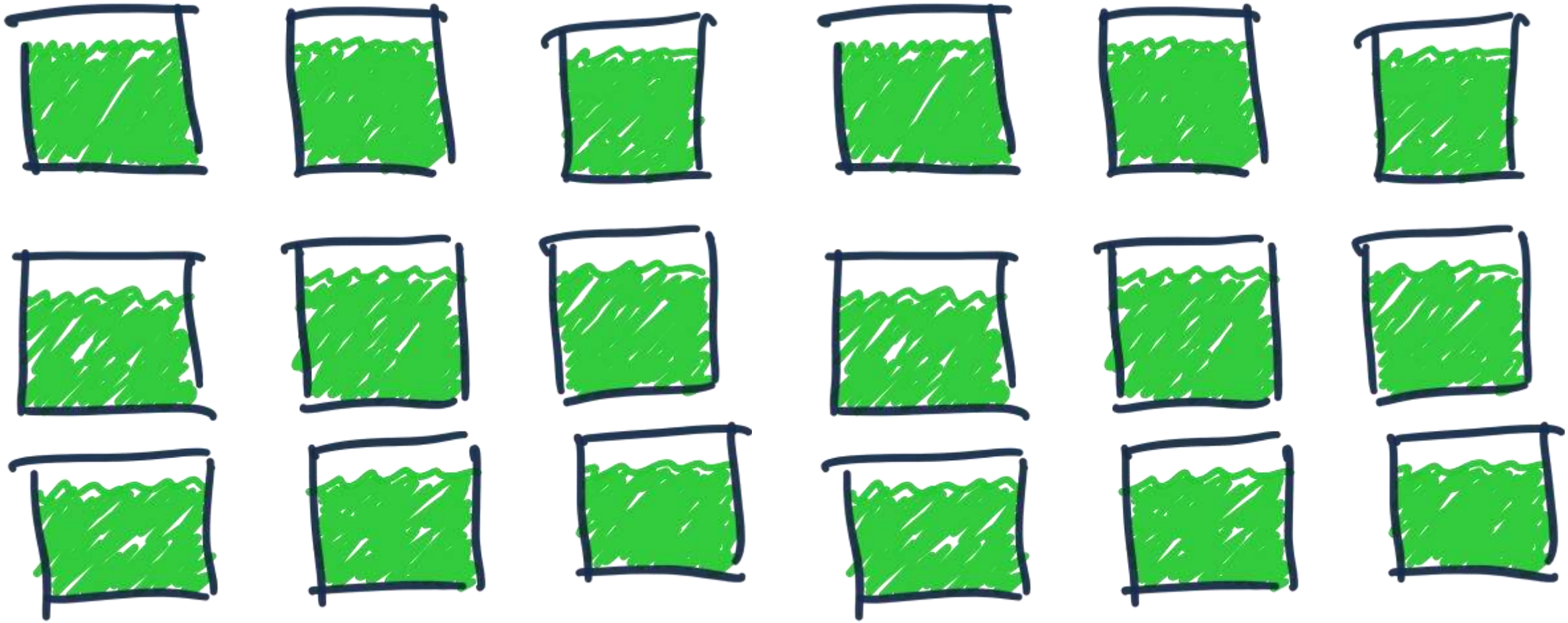
@jeremybytes



@jeremybytes



@jeremybytes



@jeremybytes

Can I Run in Parallel?

@jeremybytes

Maybe

Maybe Not

@jeremybytes

Atomic

Maybe

Deterministic

Discrete Input/Output

No Shared Data

@jeremybytes

Maybe

Maybe Not

@jeremybytes

Shared Data

Maybe Not

Shared Resources

External Dependencies

Ordered/Sequential

@jeremybytes

Parallel.For

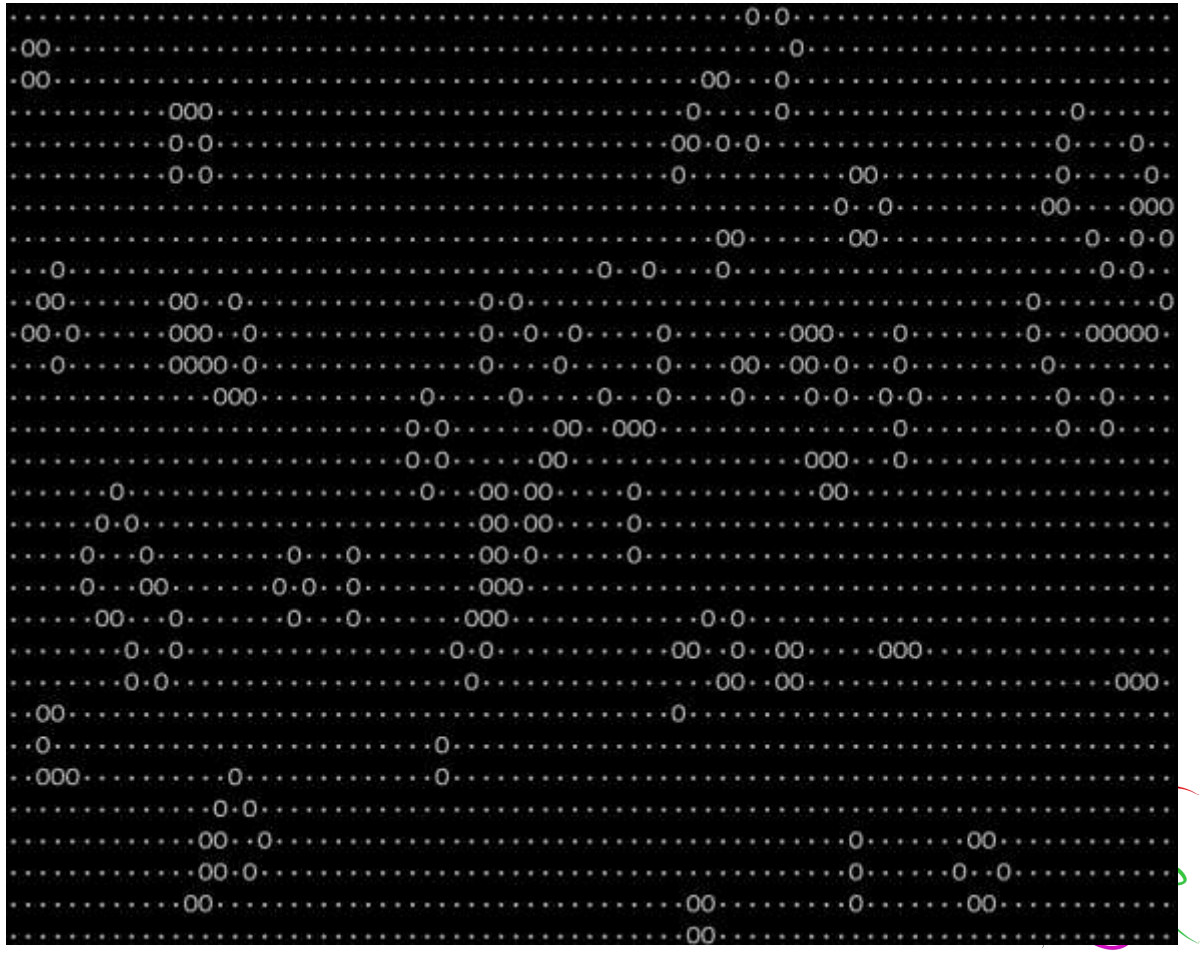
Parallel.ForEach

@jeremybytes

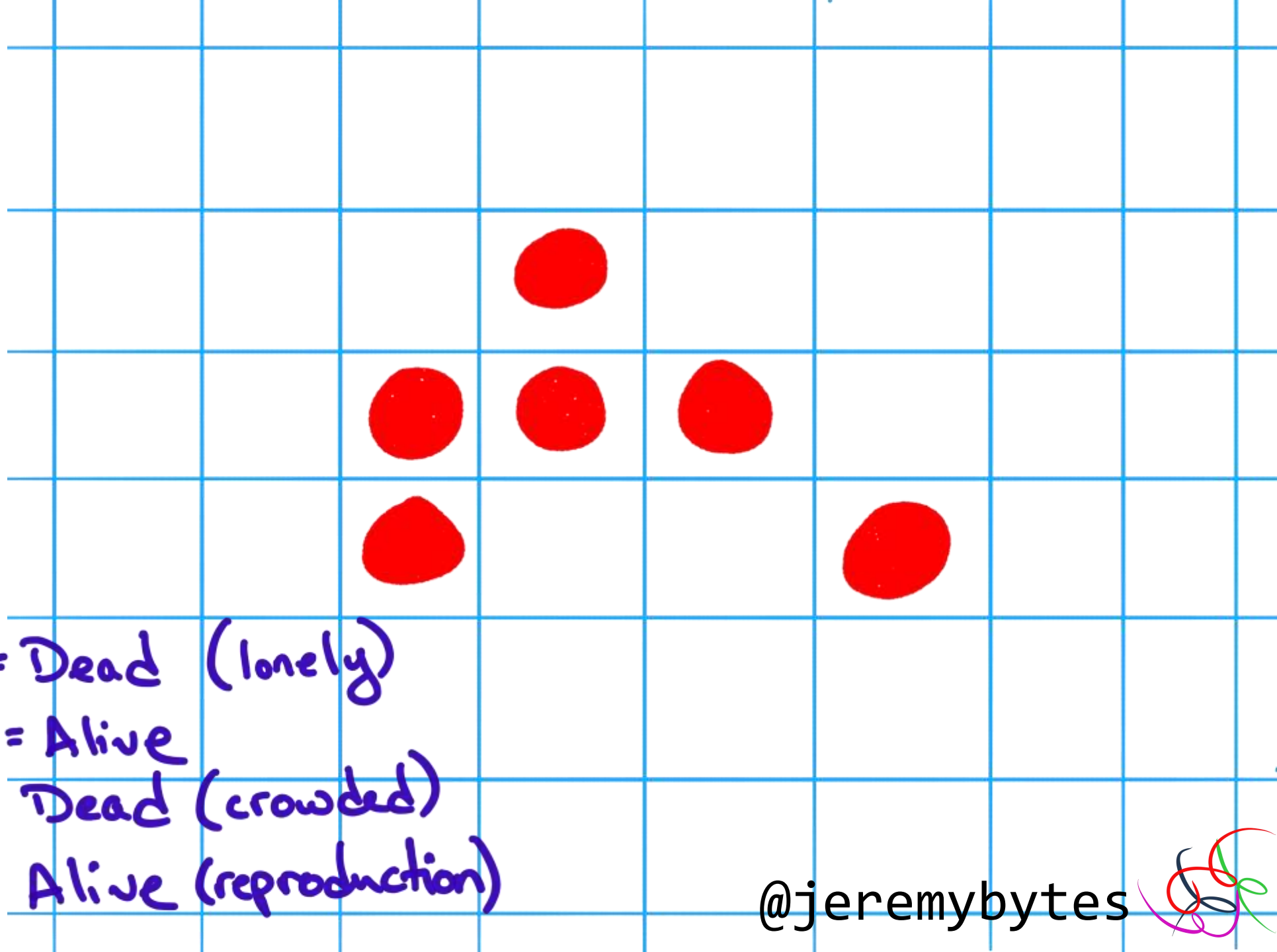
Conway's Game of Life

Parallel.For

Parallel.ForEach



Conway's Game of Life



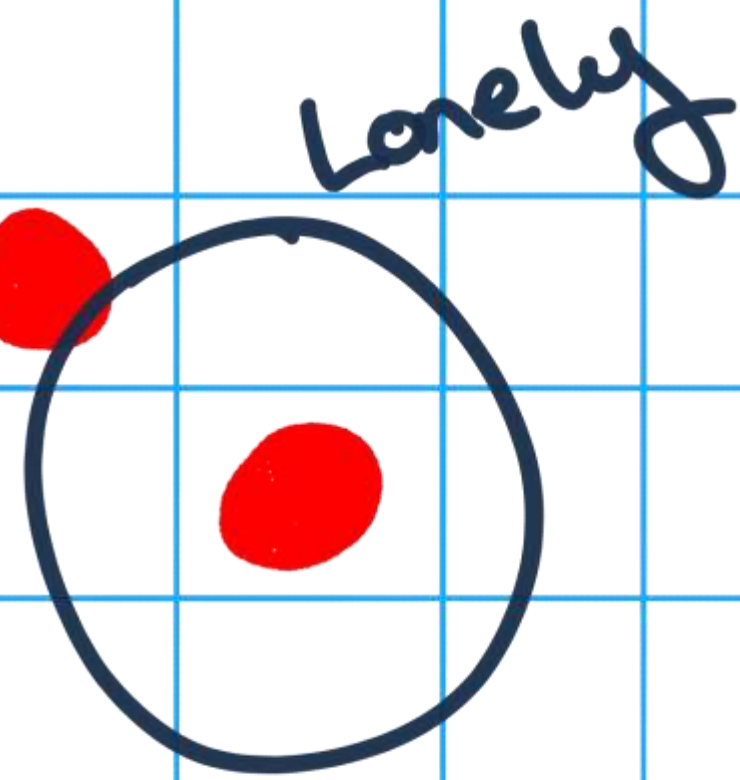
Live w/ < 2 = Dead (lonely)

Live w/ 2 or 3 = Alive

Live w/ > 3 = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes



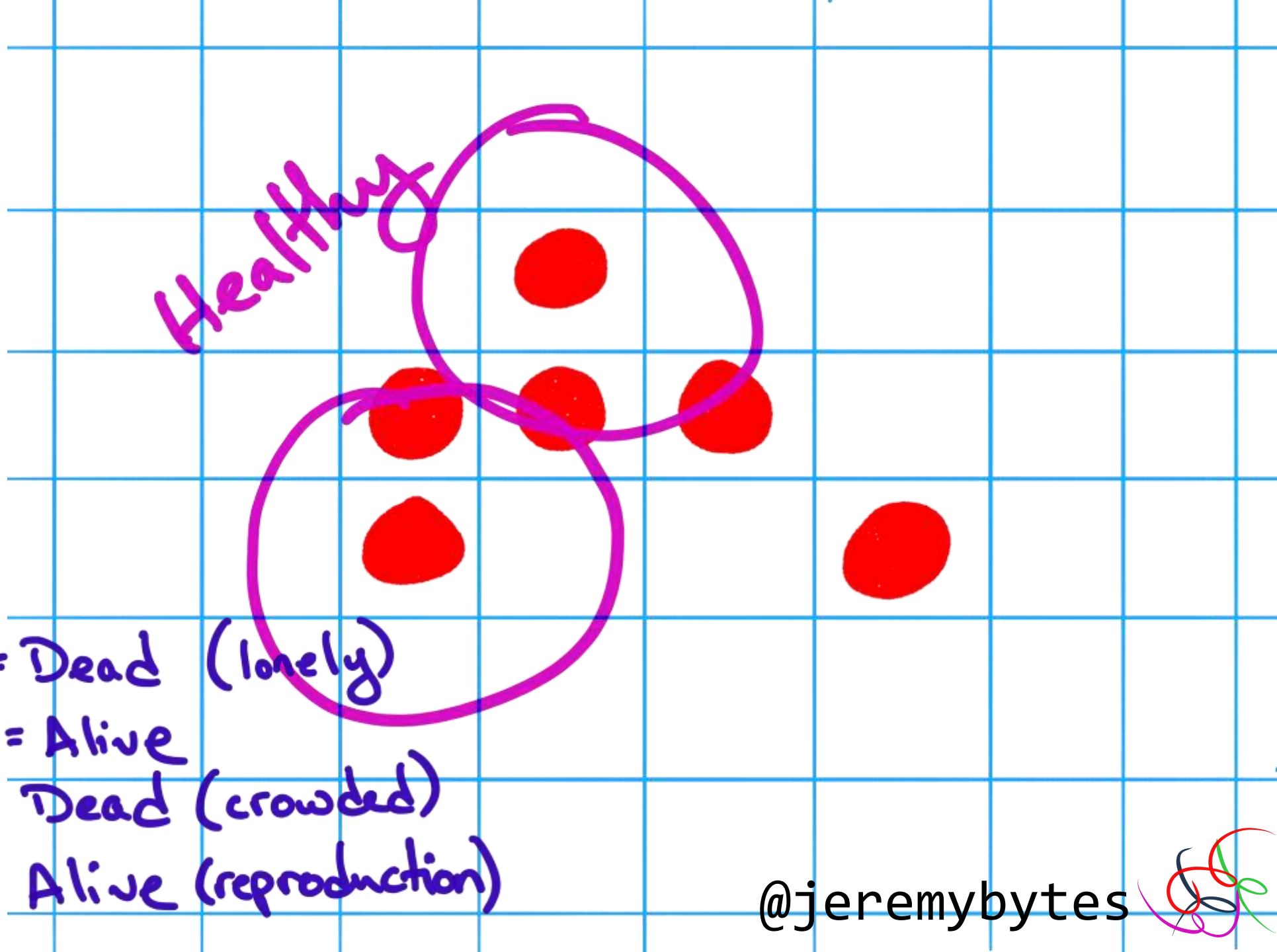
Live w/ < 2 = Dead (lonely)

Live w/ 2 or 3 = Alive

Live w/ > 3 = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes



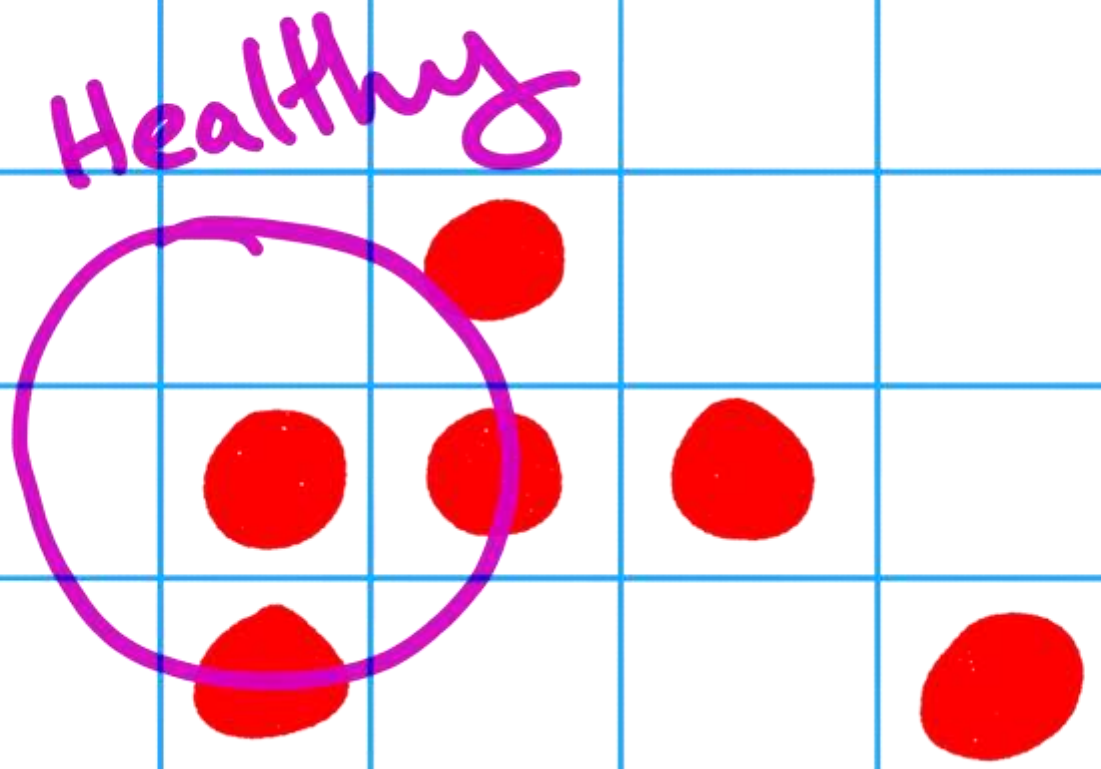
Live w/ < 2 = Dead (lonely)

Live w/ 2 or 3 = Alive

Live w/ > 3 = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes



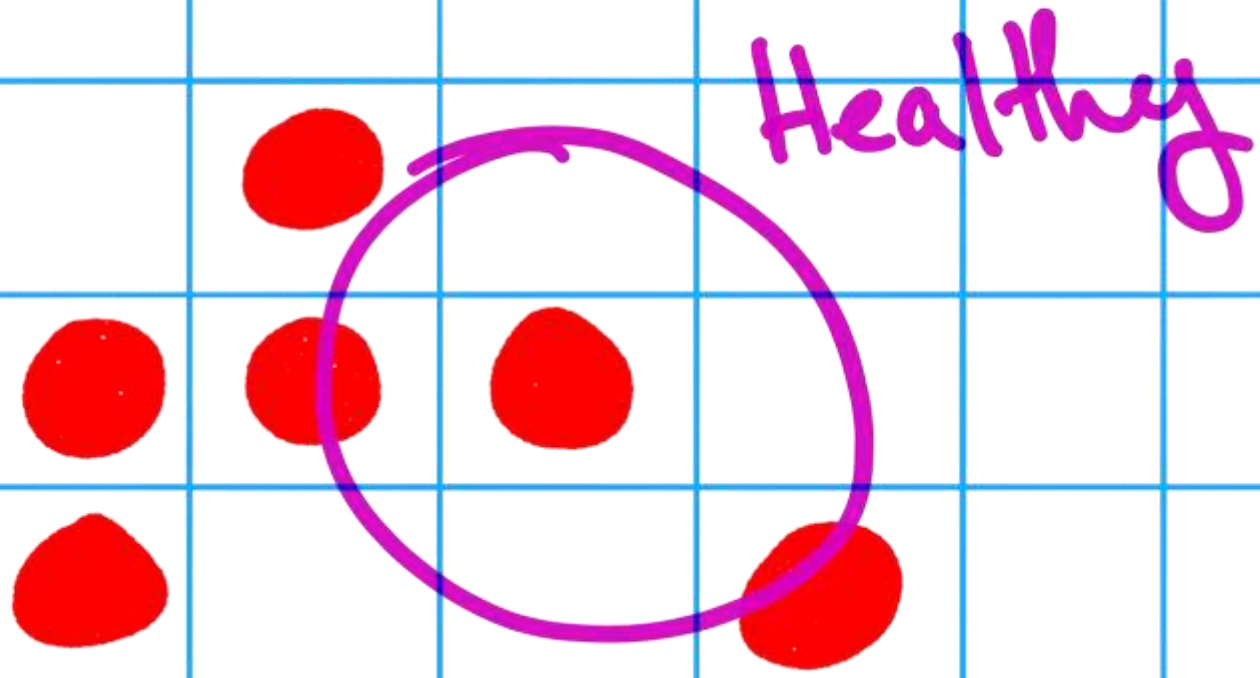
Live w/ < 2 = Dead (lonely)

Live w/ 2 or 3 = Alive

Live w/ > 3 = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes



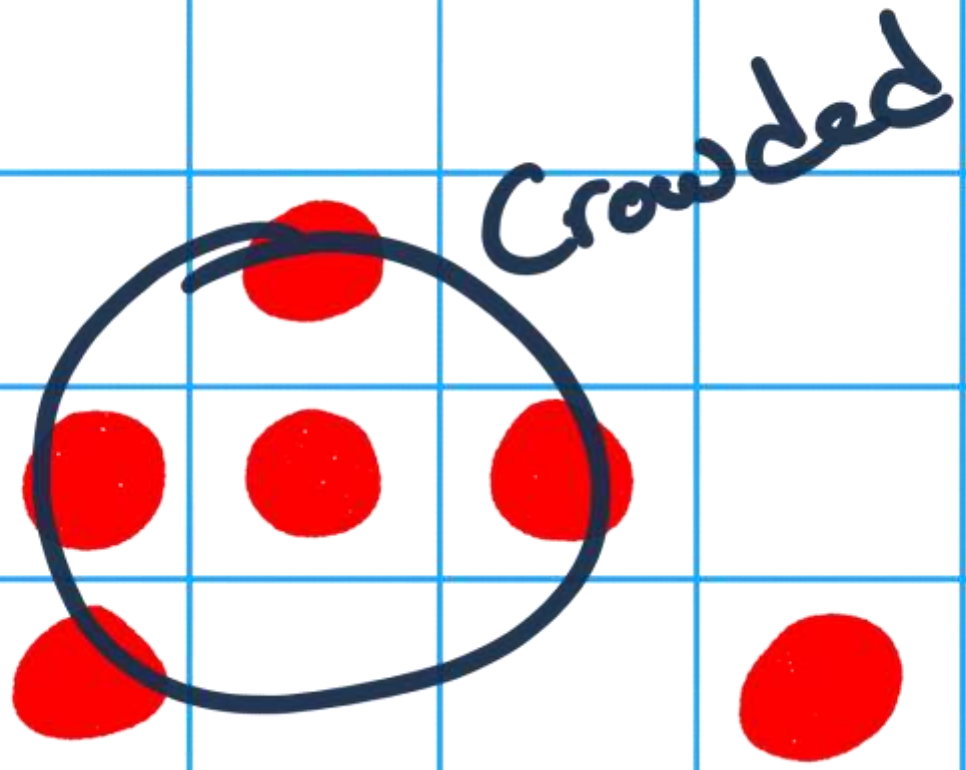
Live w/ < 2 = Dead (lonely)

Live w/ 2 or 3 = Alive

Live w/ > 3 = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes



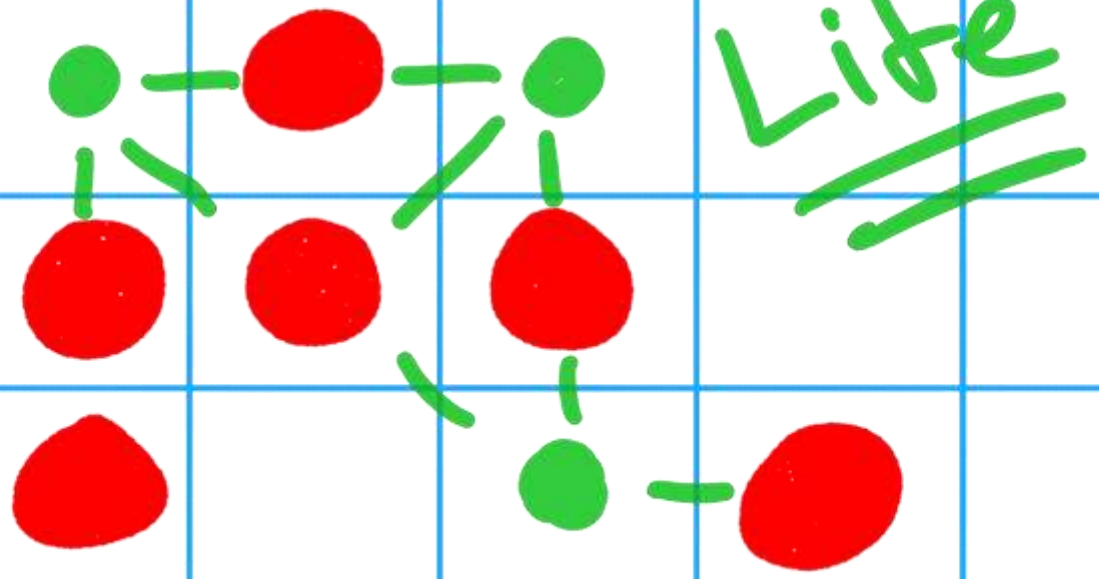
Live w/ < 2 = Dead (lonely)

Live w/ 2 or 3 = Alive

Live w/ > 3 = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes



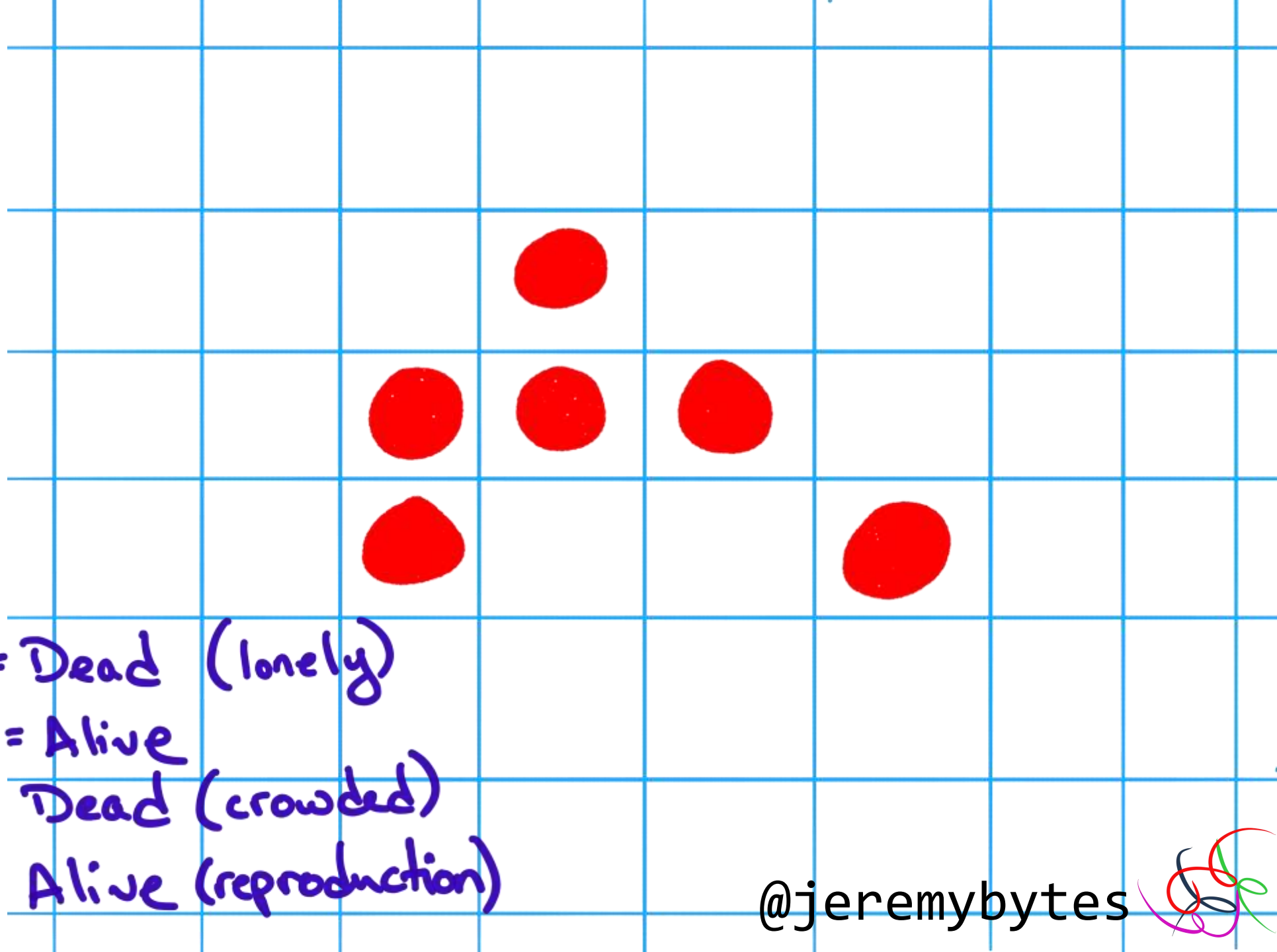
Live w/ < 2 = Dead (lonely)

Live w/ 2 or 3 = Alive

Live w/ > 3 = Dead (crowded)

Dead w/ 3 = Alive (reproduction)

@jeremybytes



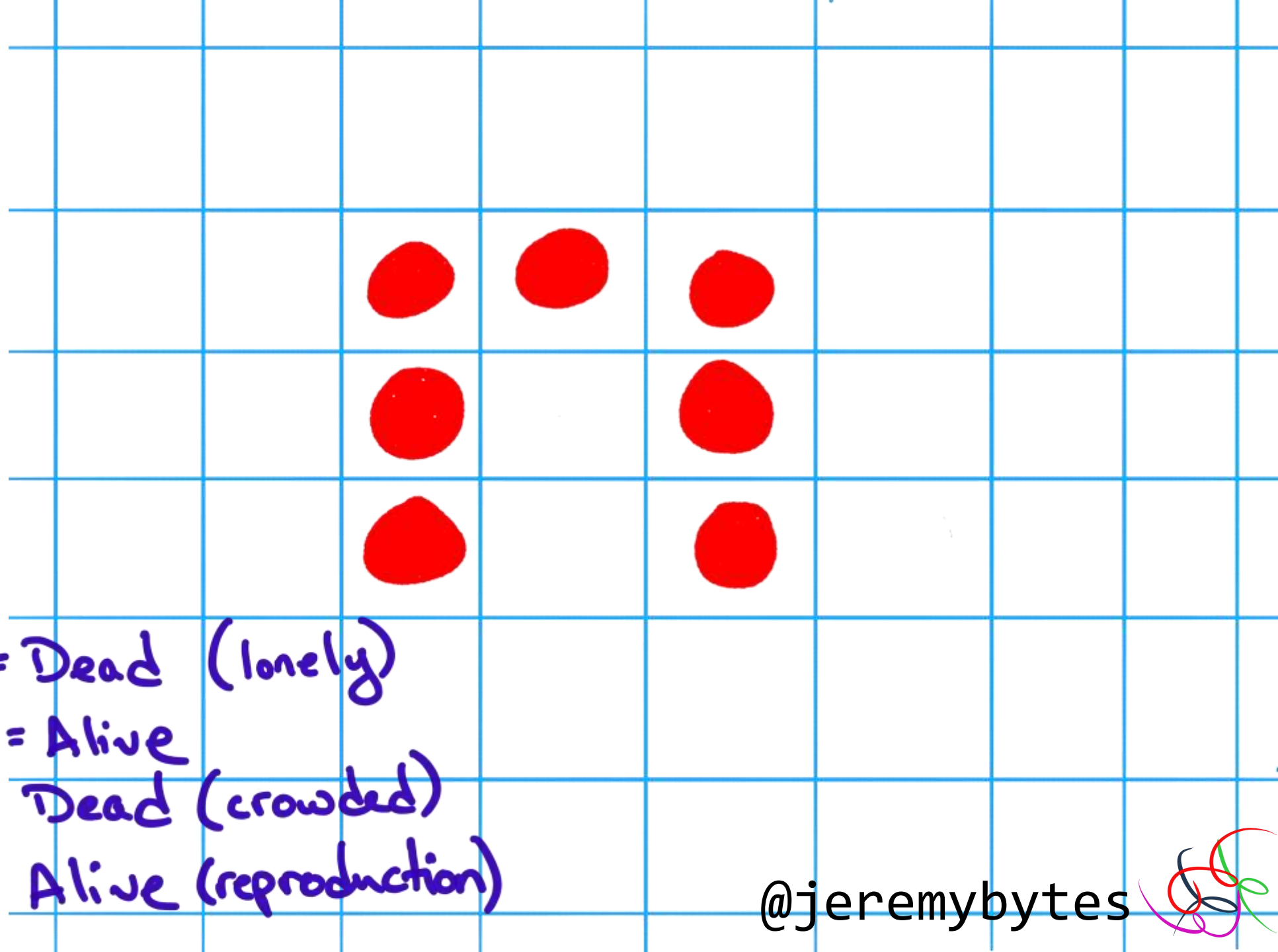
Live w/ < 2 = Dead (lonely)

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)

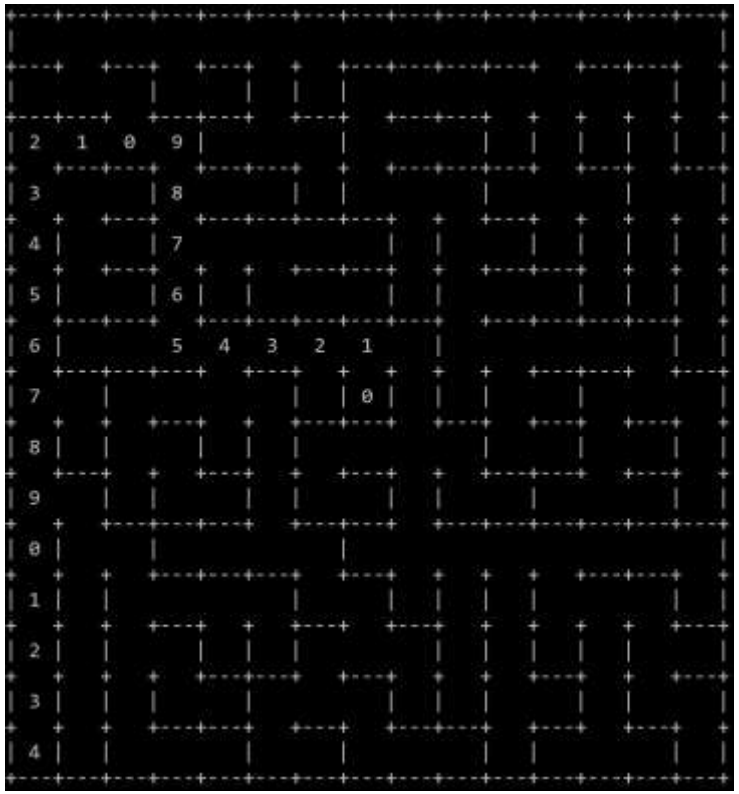
@jeremybytes

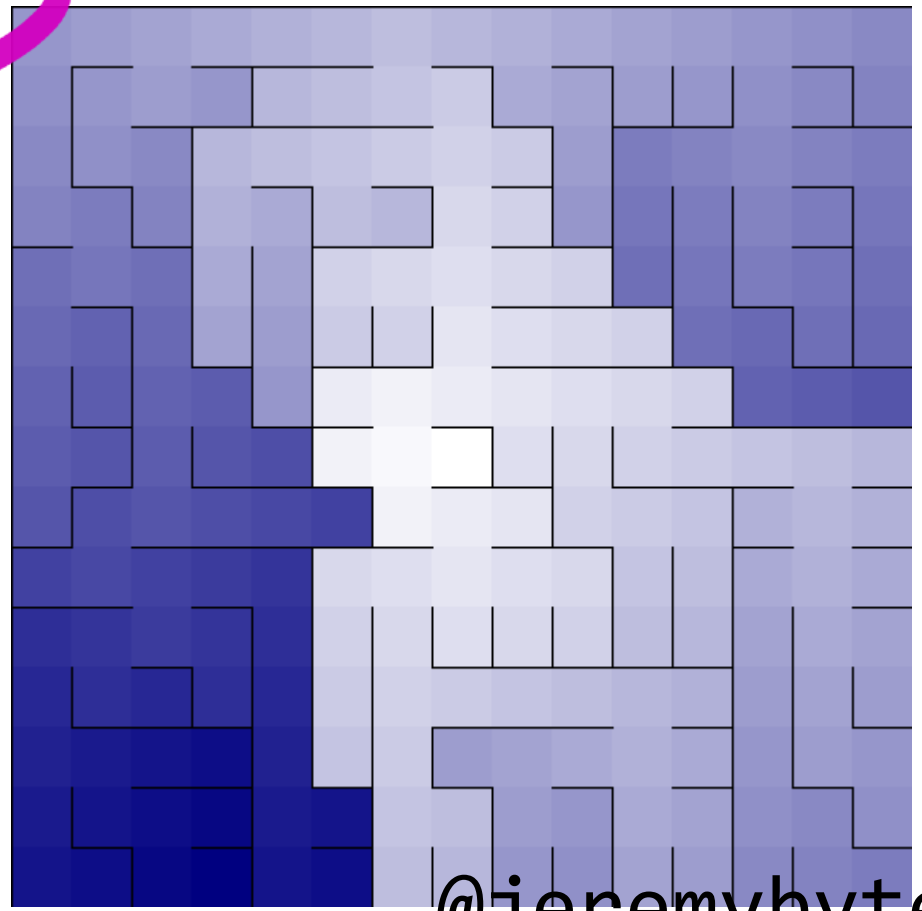
Hard to Run Parallel

@jeremybytes

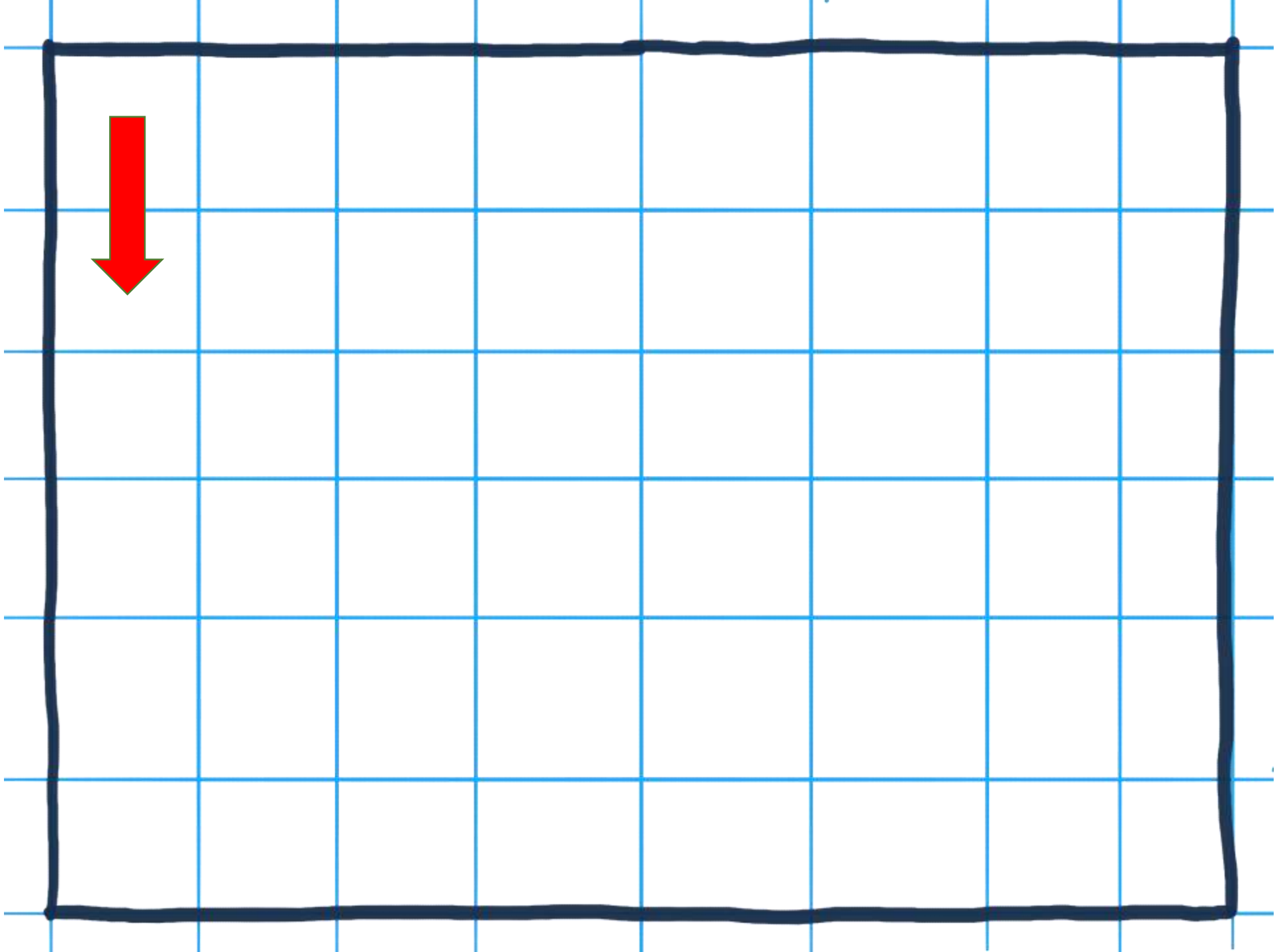
Mazes

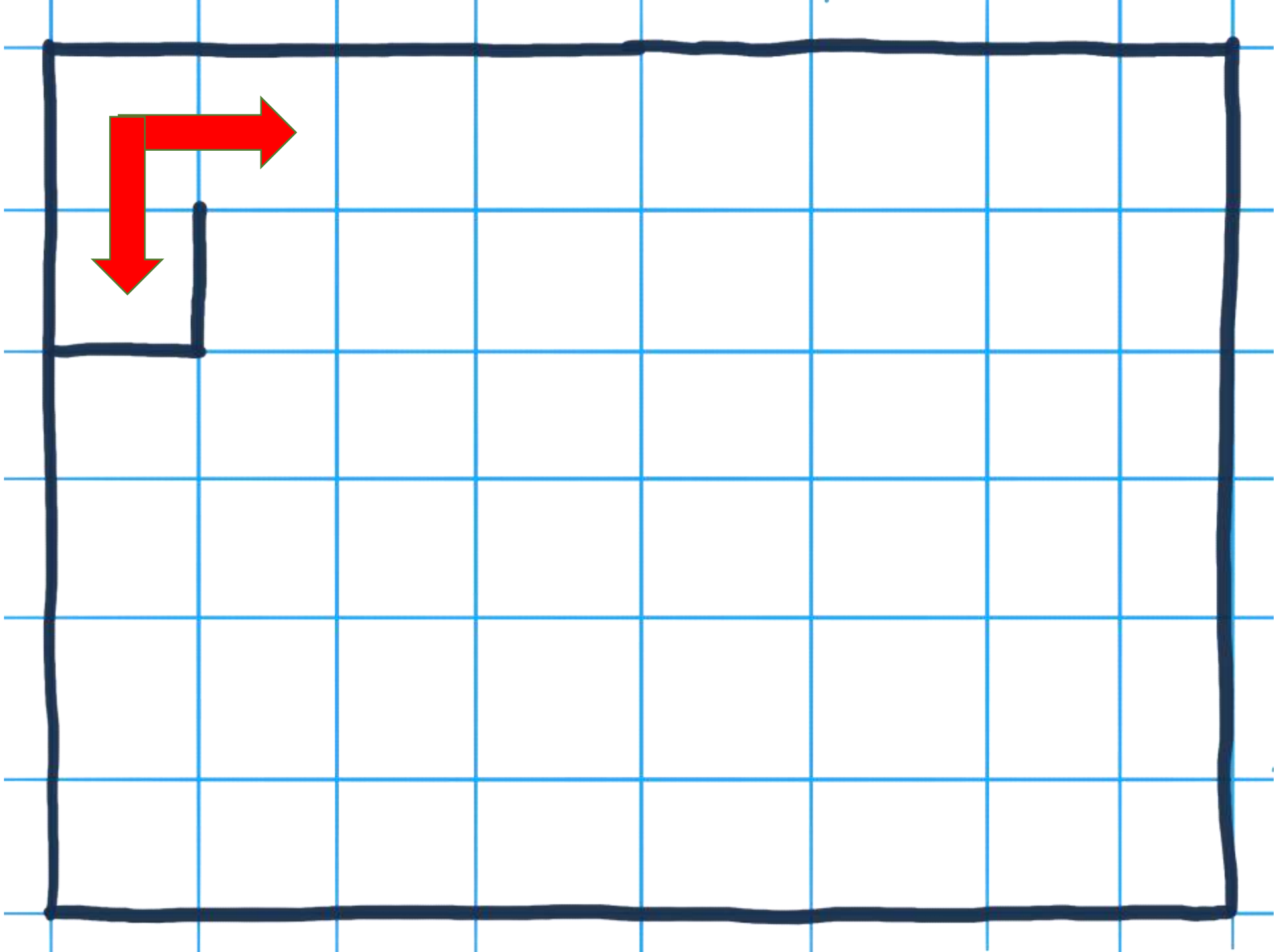
Hard to Run Parallel

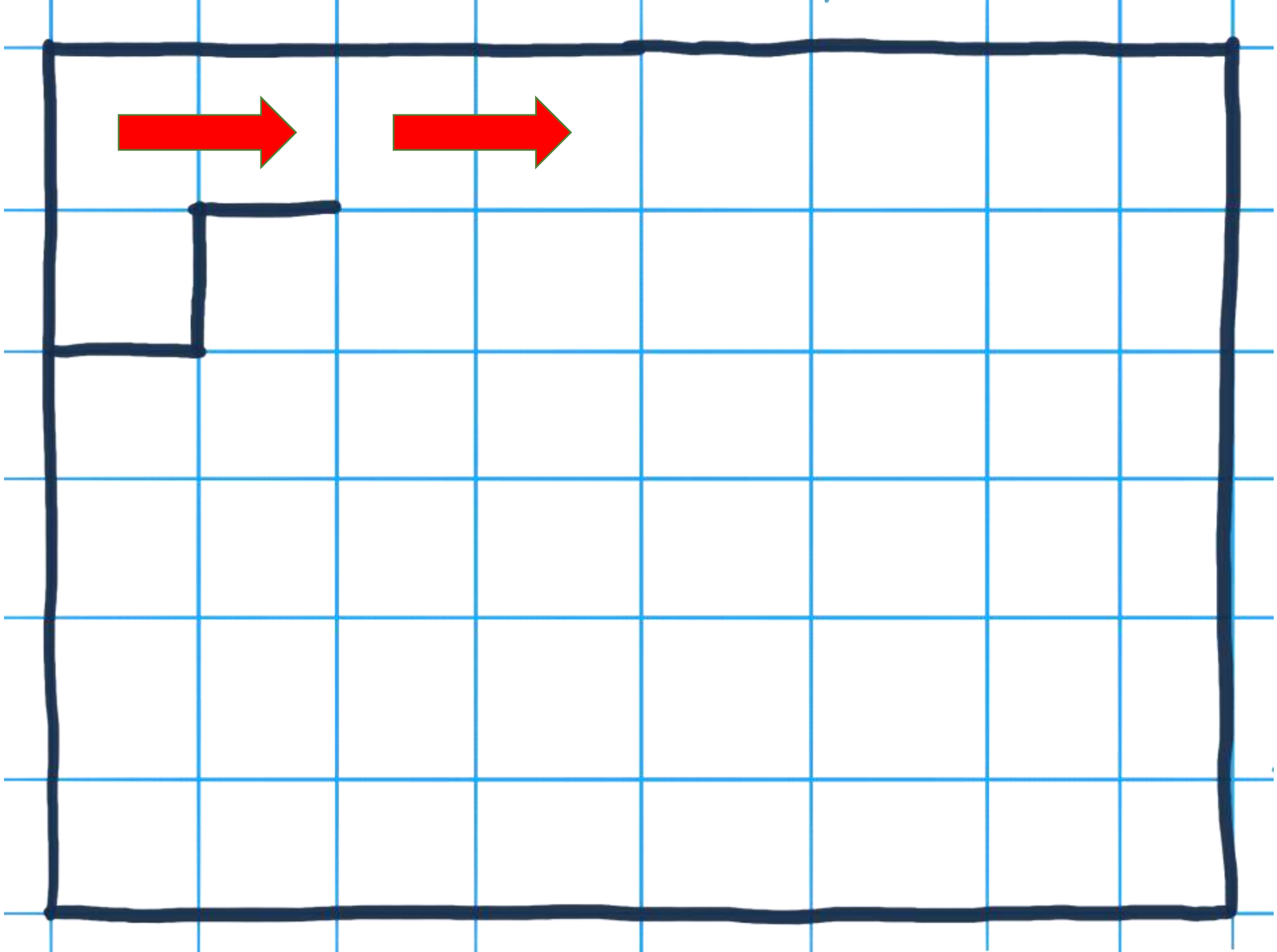


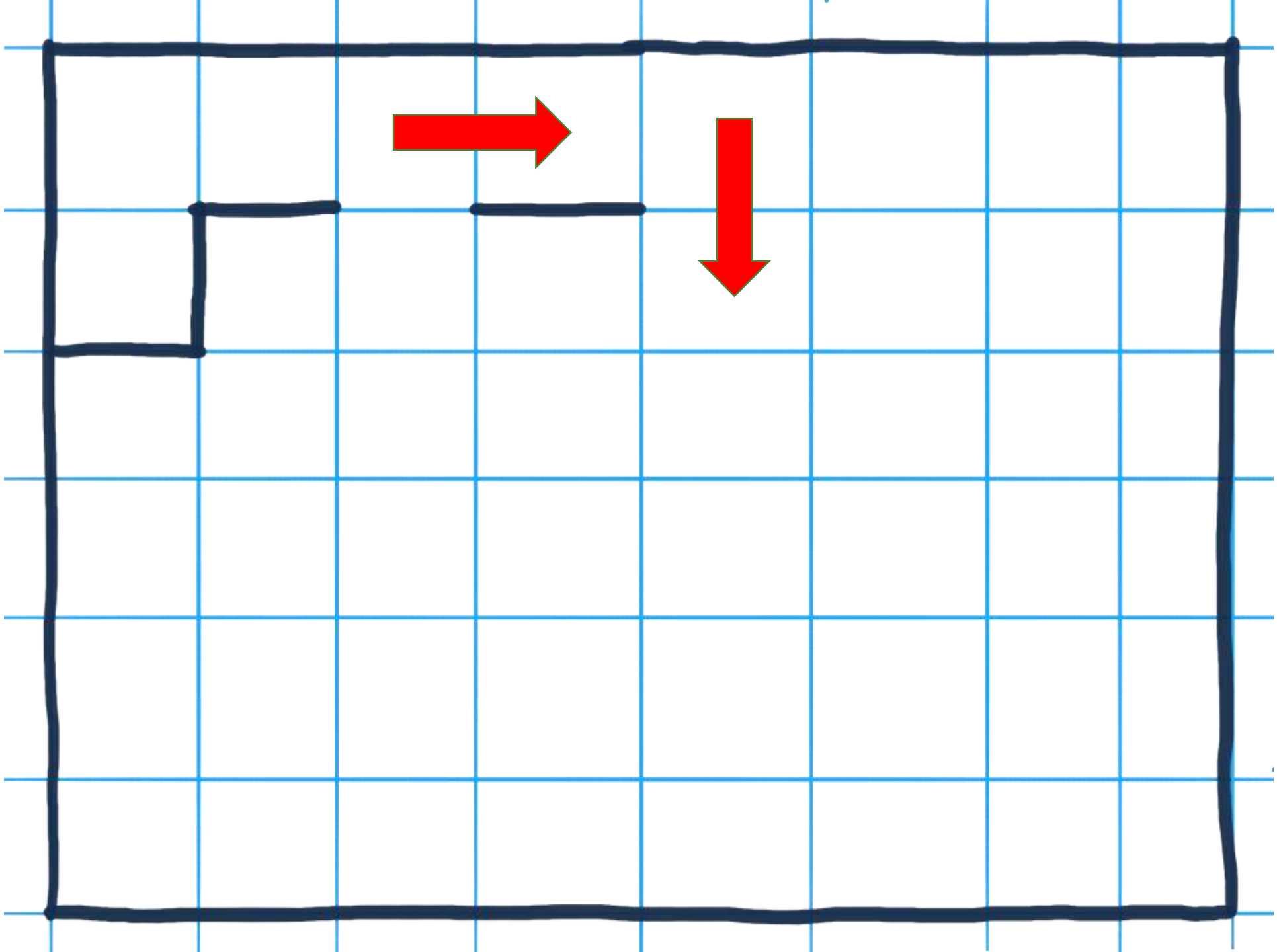


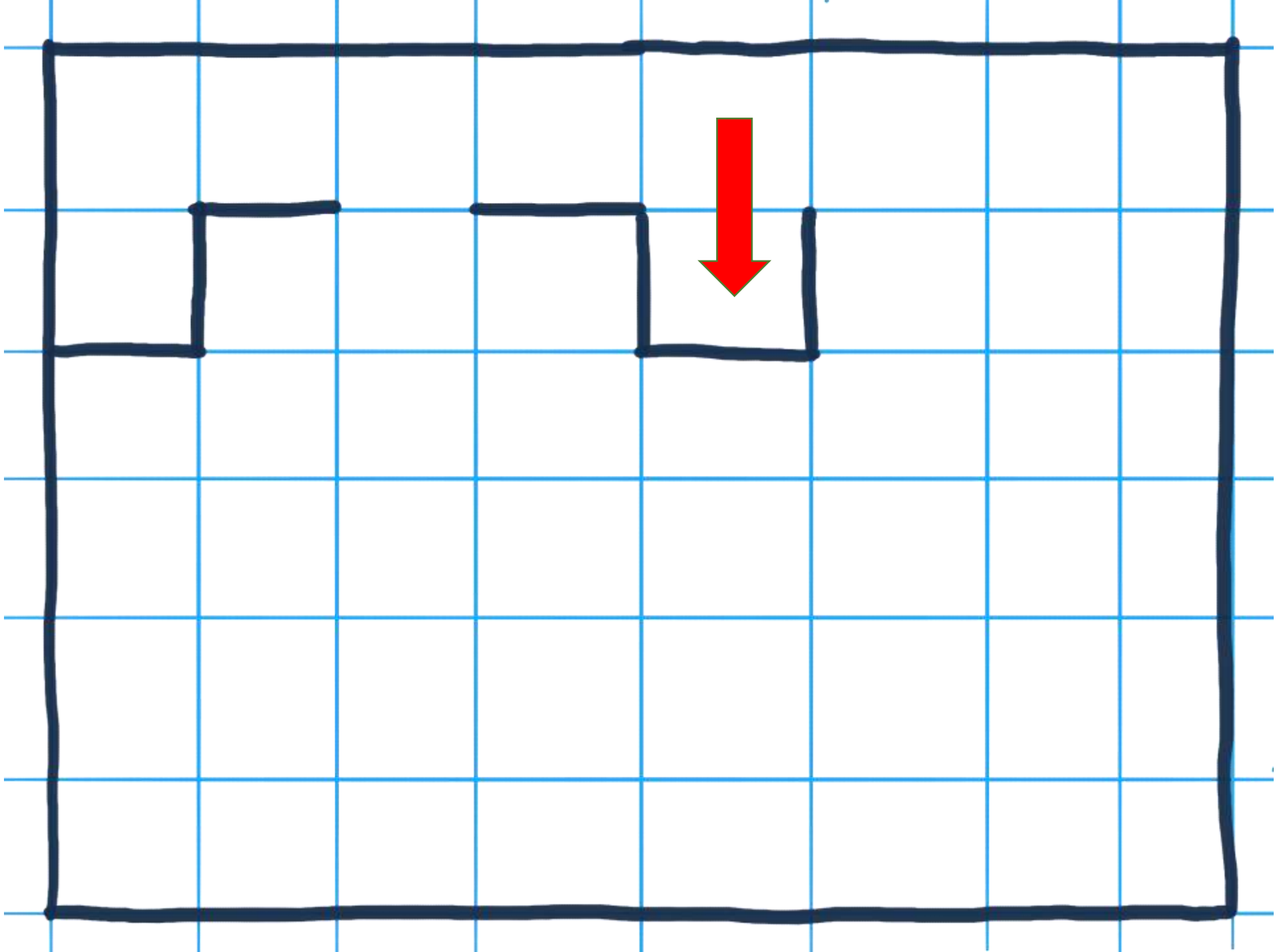
@jeremybytes

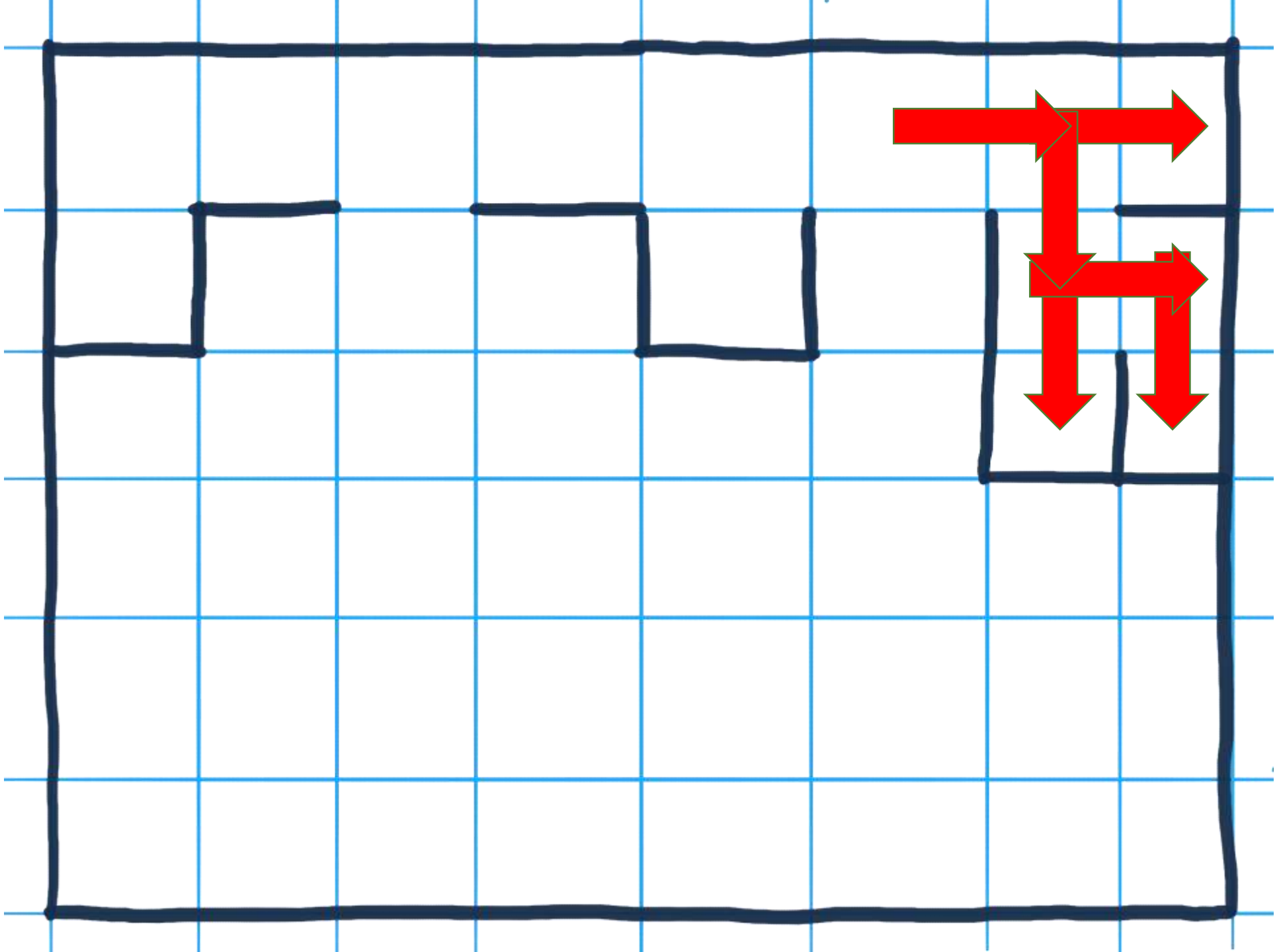


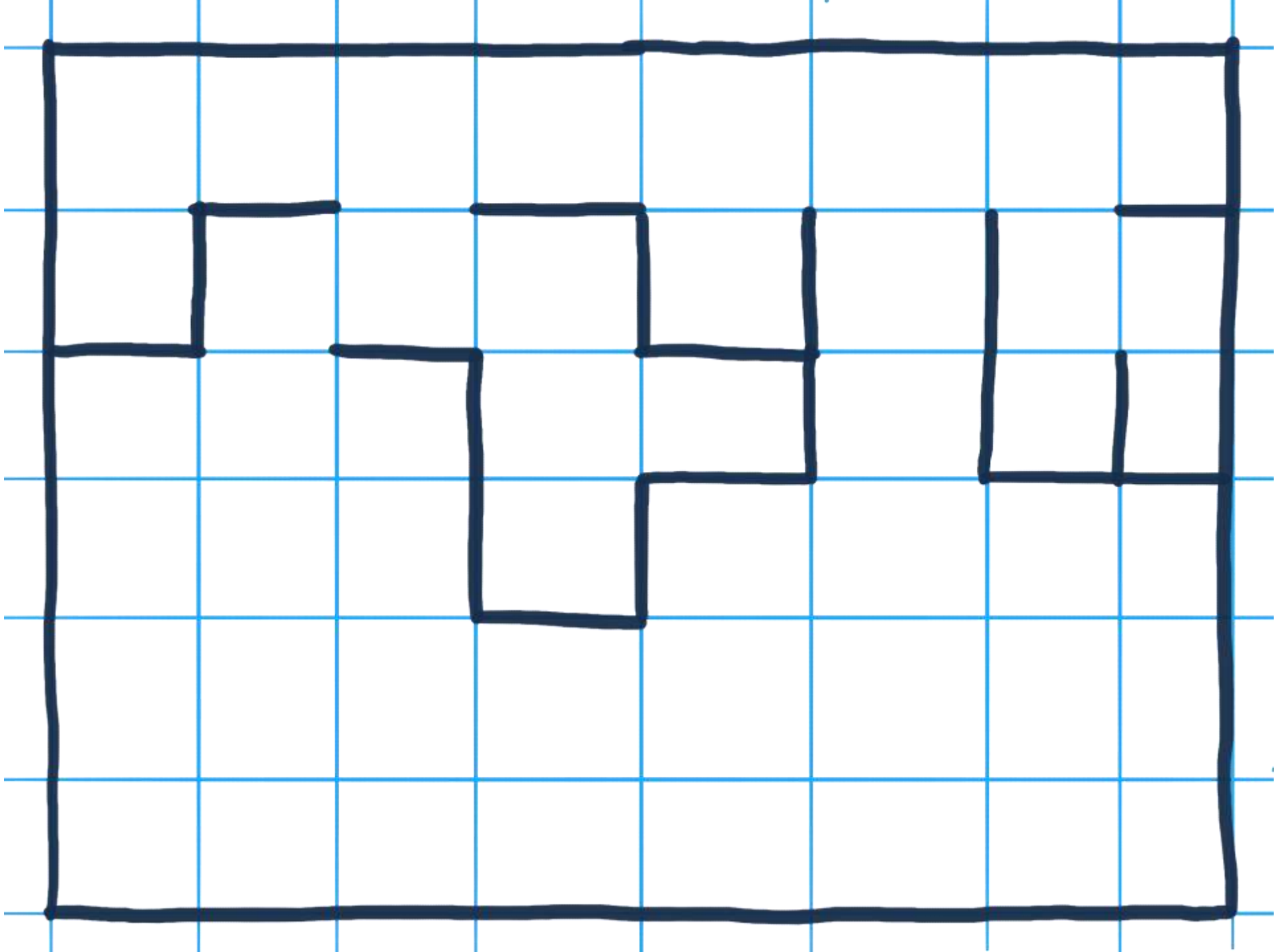


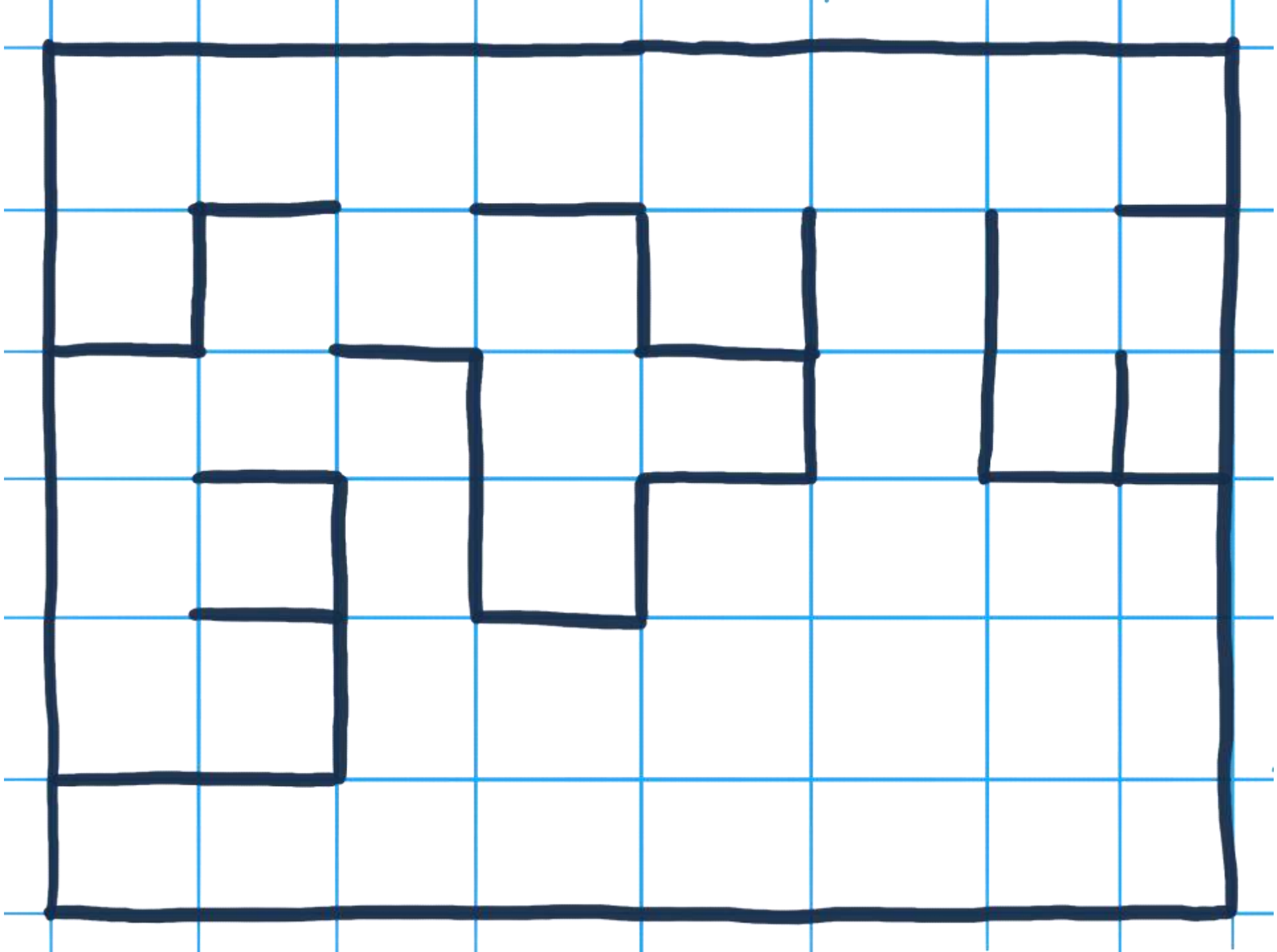


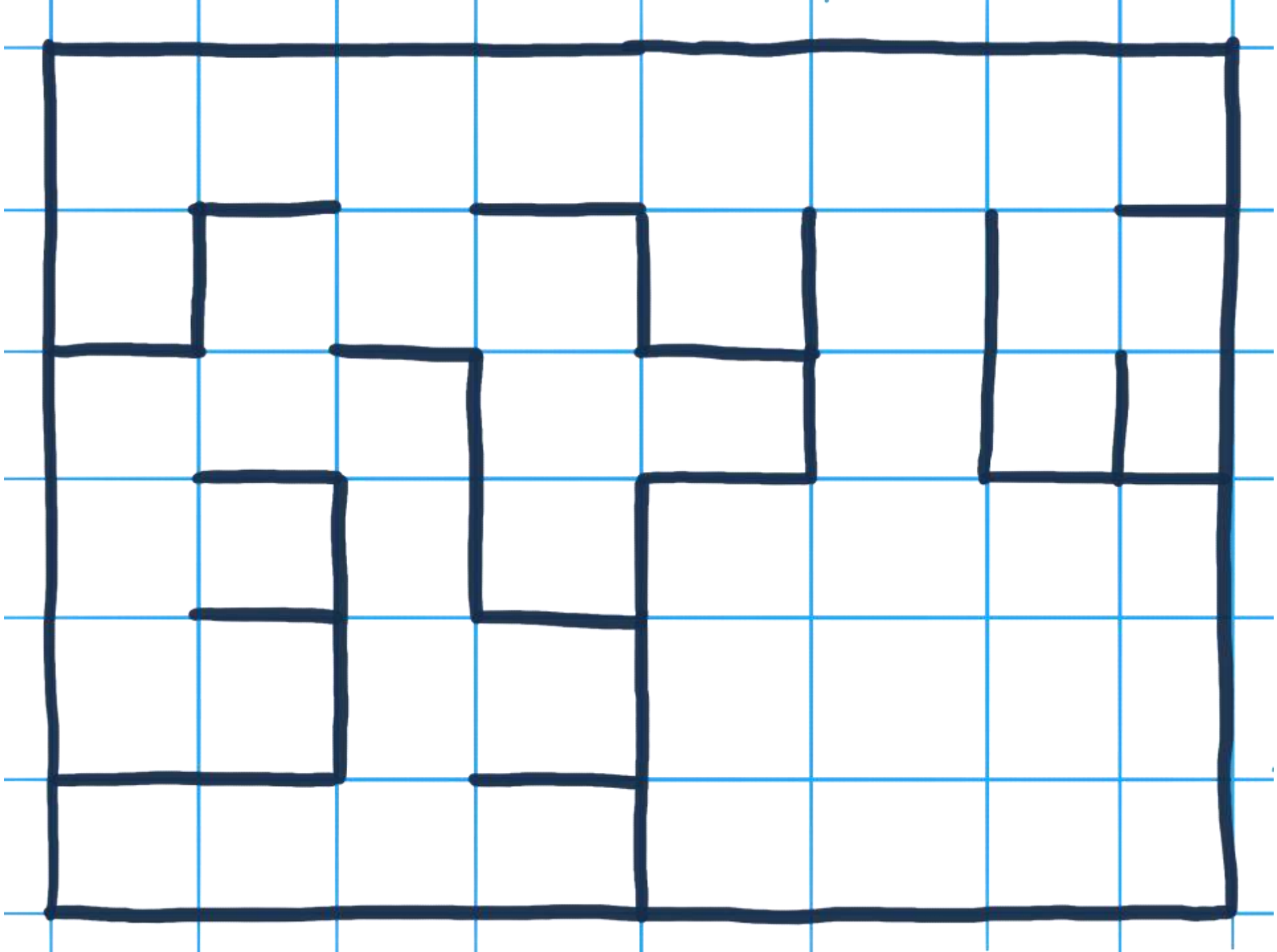


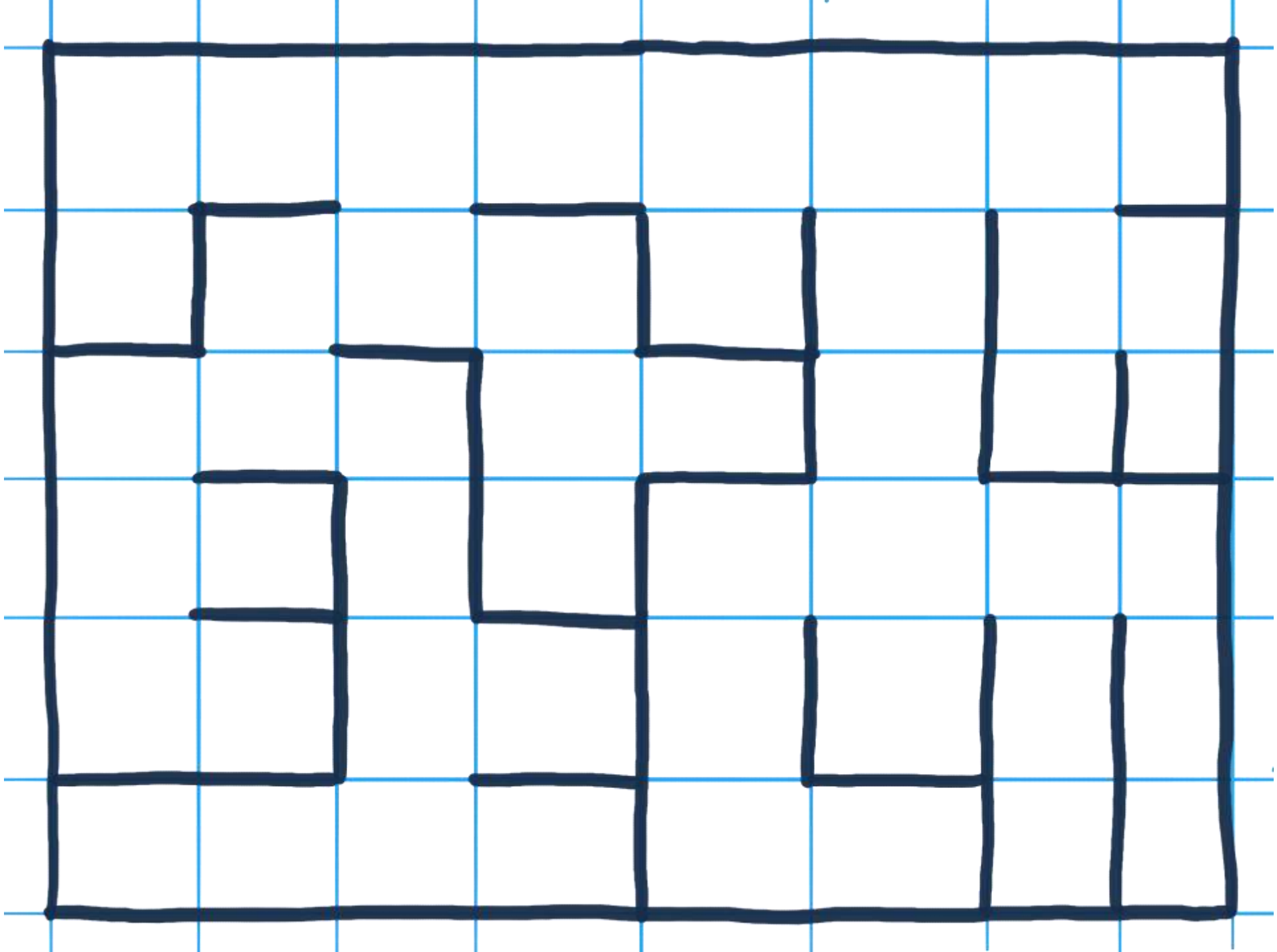










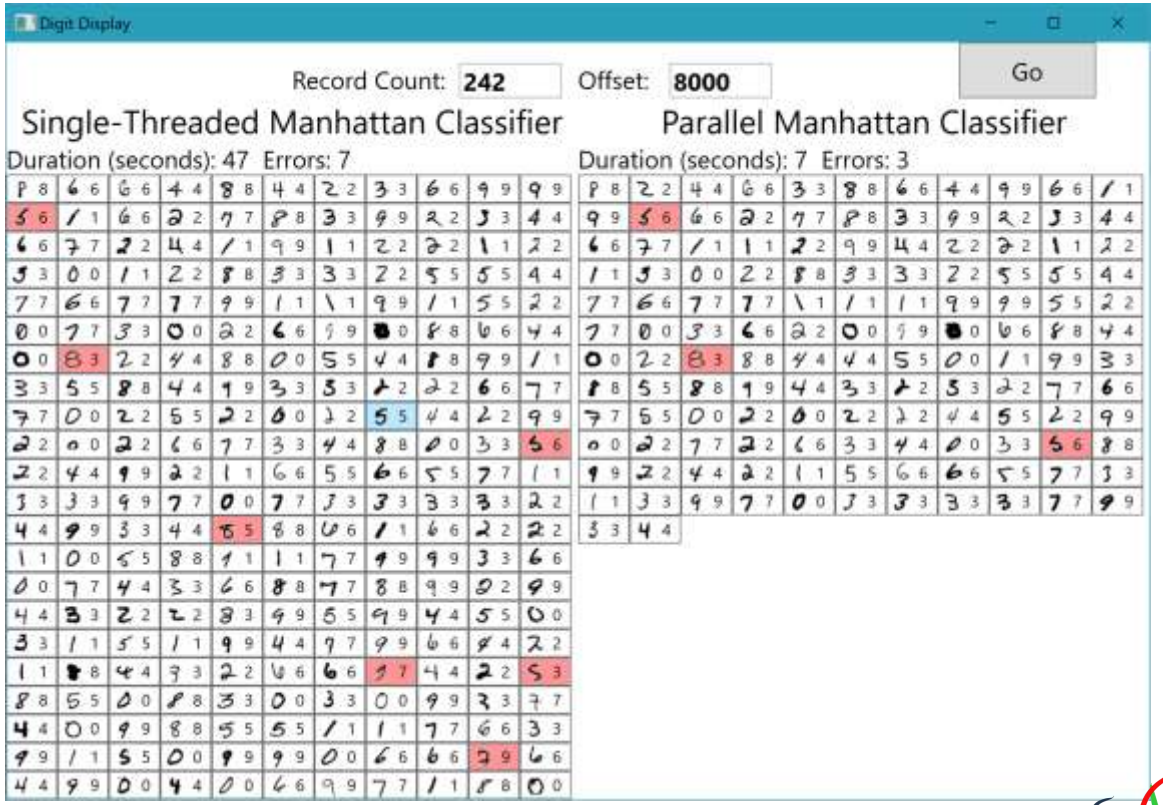


Parallel Tasks

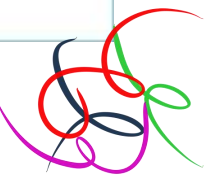
@jeremybytes

Parallel Tasks

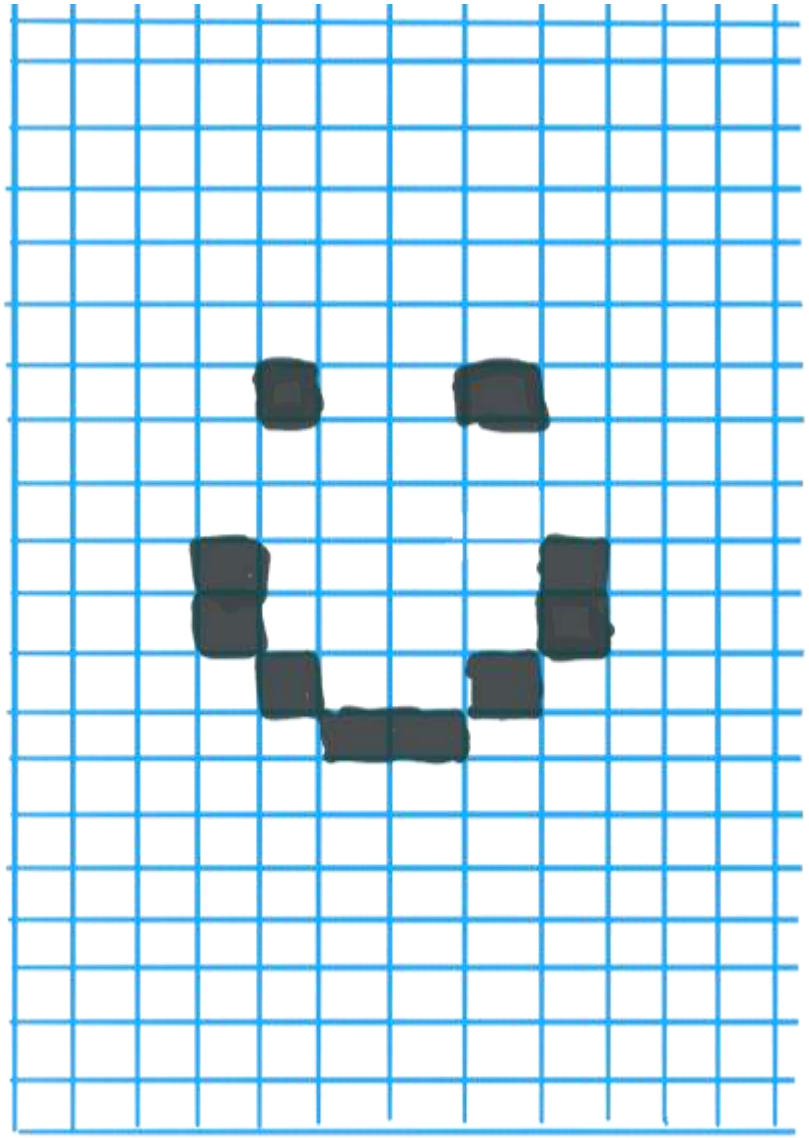
Digit Recognition



@jeremybytes



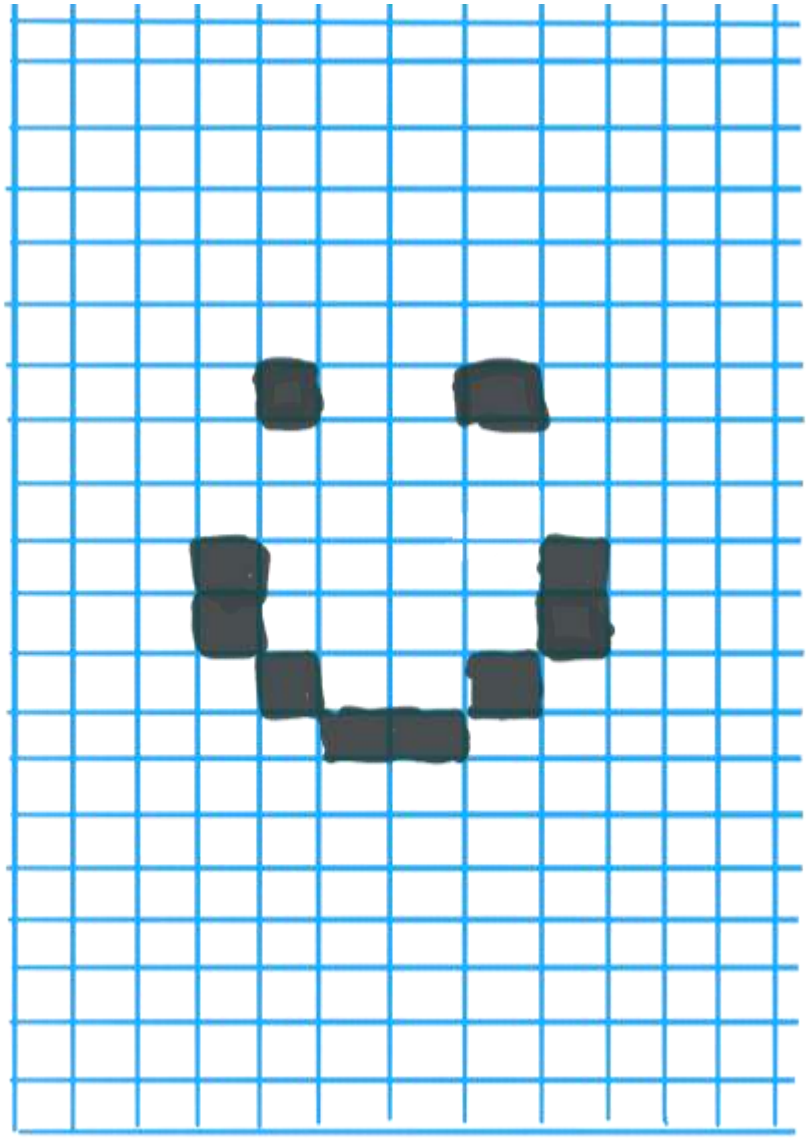

Digit Recognition
@jeremybytes

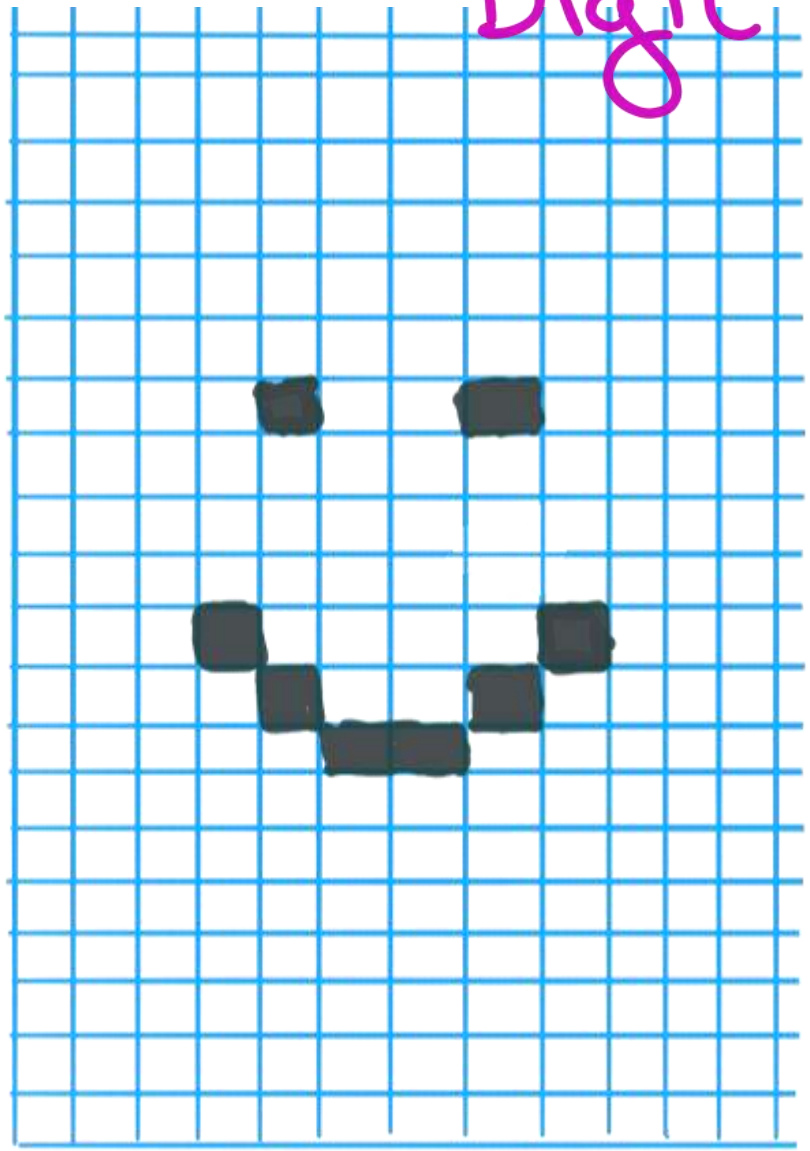


Digit Recognition

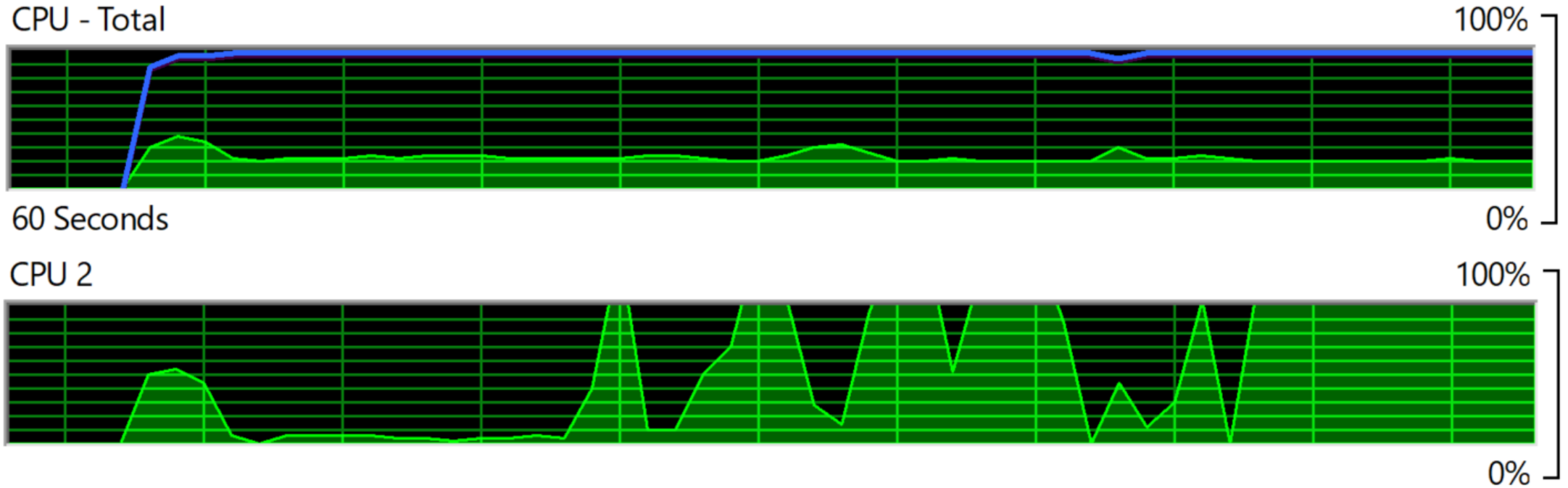
@jeremybytes

Digit Recognition

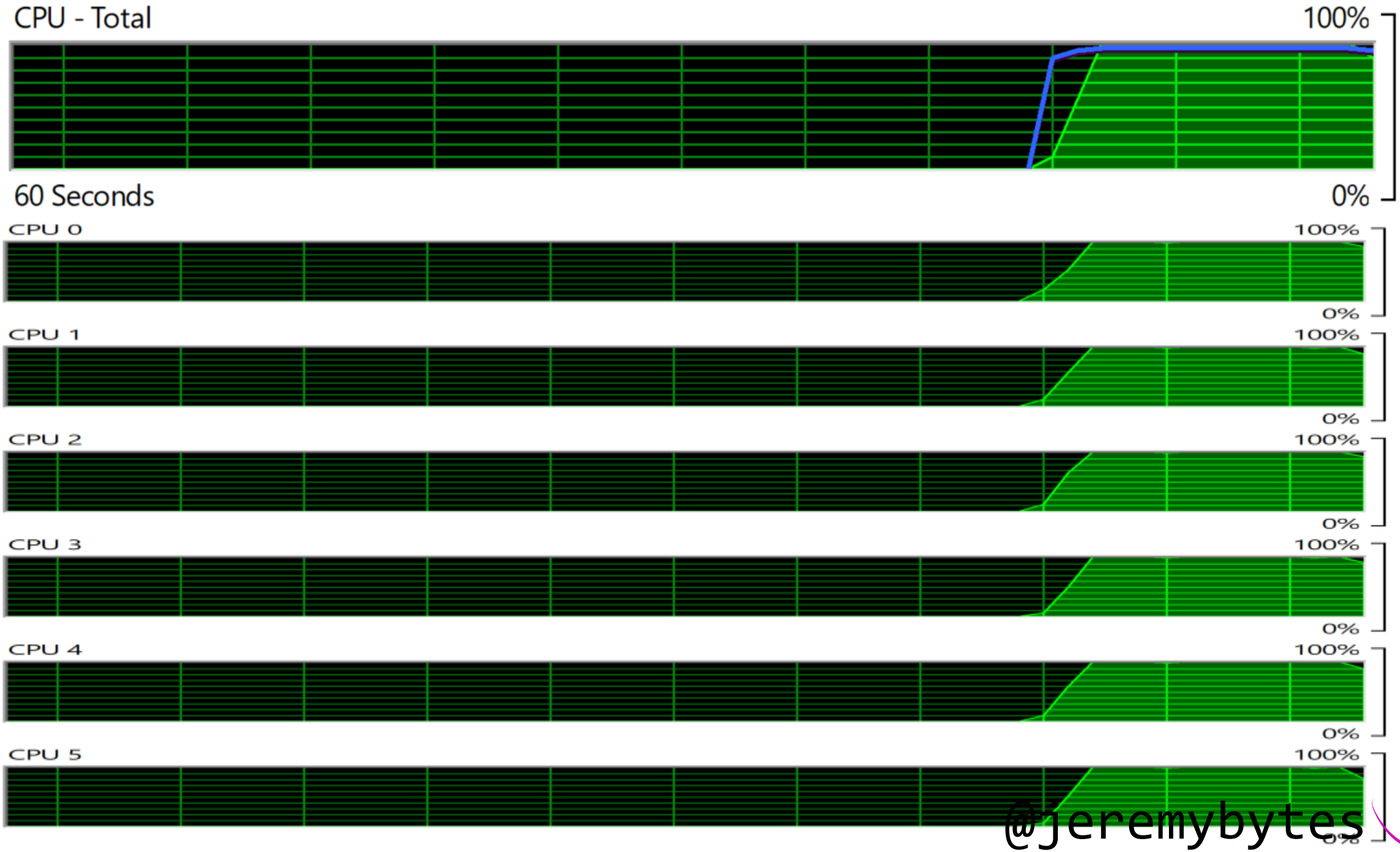




@jeremybytes



@jeremybytes


CPU - Total
100%
60 Seconds
0%
CPU 0
100%
0%
CPU 1
100%
0%
CPU 2
100%
0%
CPU 3
100%
0%
CPU 4
100%
0%
CPU 5
100%
@jeremybytes

Measure
Measure
Measure

Resources

@jeremybytes

Resources

Parallel Programming with Microsoft .NET

@jeremybytes

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.

Parallel Programming with Microsoft .NET

@jeremybytes

Resources

- Parallel Programming w/ Microsoft .NET
  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
- GitHub
  https://github.com/jeremybytes/parallel-programming

@jeremybytes

Thank You

jeremybytes.com

@jeremybytes