DAA PROJECT

ELEMENTARY CELLULAR AUTOMATON SIMULATOR AND PREDICTOR

GitHub Link:
https://github.com/Prathamesh111-netizen/DAA-project-sem-4

A presentation by :
Prathamesh Rajan Pawar (2020400040)
Harsh Ravindra Patil (2020400037)
Yash Kishorbhai Pabari (2020400030)

Today's Discussion

OUTLINE OF TOPICS

Walkthrough of History / Problem Statement

1 Dimensional Cellular Automata

1. Rules
2. First solution that came to our Mind
3. Most Optimised Possible

2 Dimensional Cellular Automata

1. Game of Life
2. First solution that came to our Mind
3. HashLife

The Westmire Way

A BRIEF HISTORY

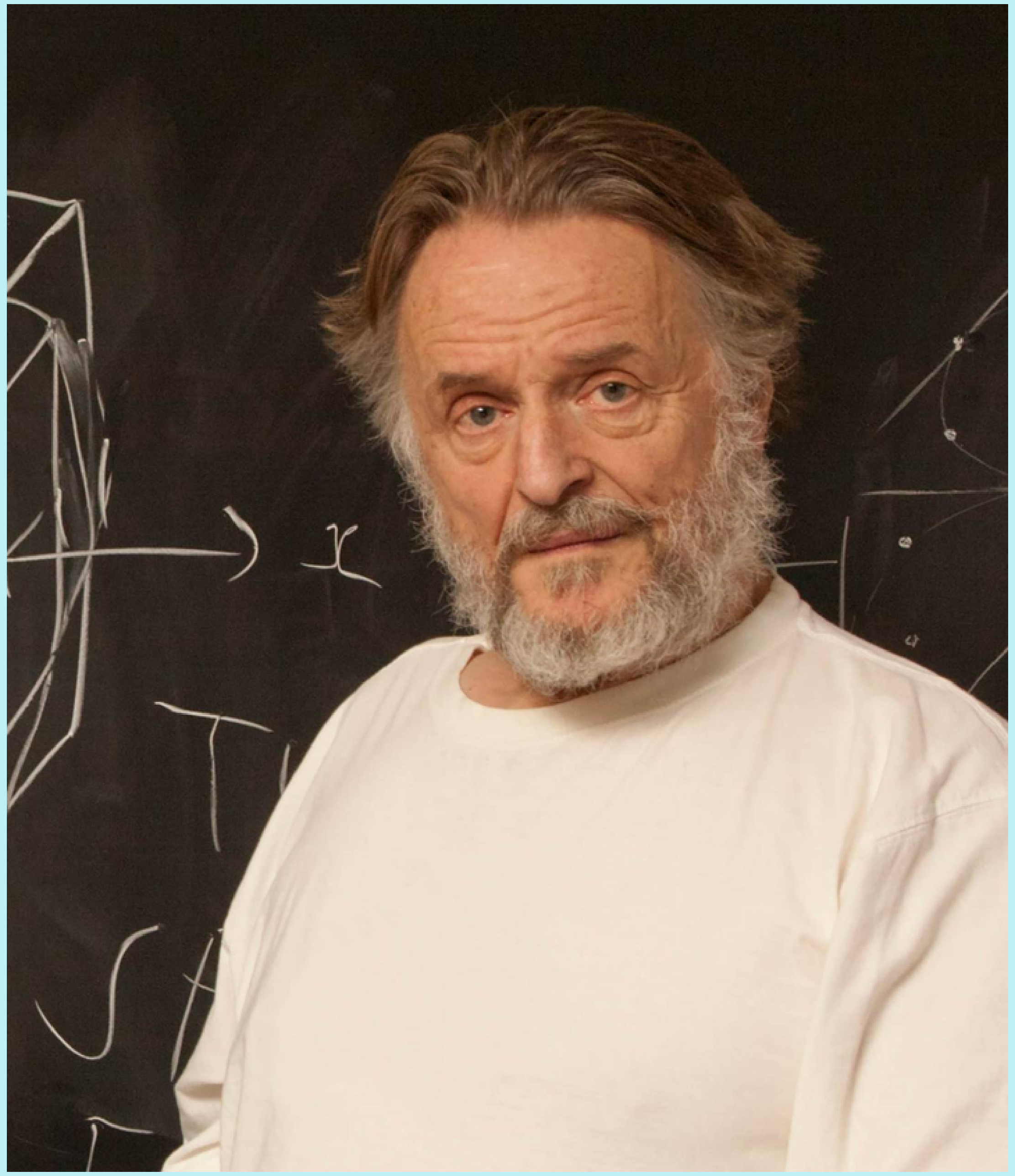
The game made its first public appearance in the October 1970 issue of Scientific American, in Martin Gardner's "Mathematical Games" column, which was based on personal conversations with Conway.

Theoretically, the Game of Life has the power of a universal Turing machine: anything that can be computed algorithmically can be computed within the Game of Life.

Gardner wrote, "Because of Life's analogies with the rise, fall and alterations of a society of living organisms, it belongs to a growing class of what are called 'simulation games' (games that resemble real-life processes)

JOHN VON NEUMANN

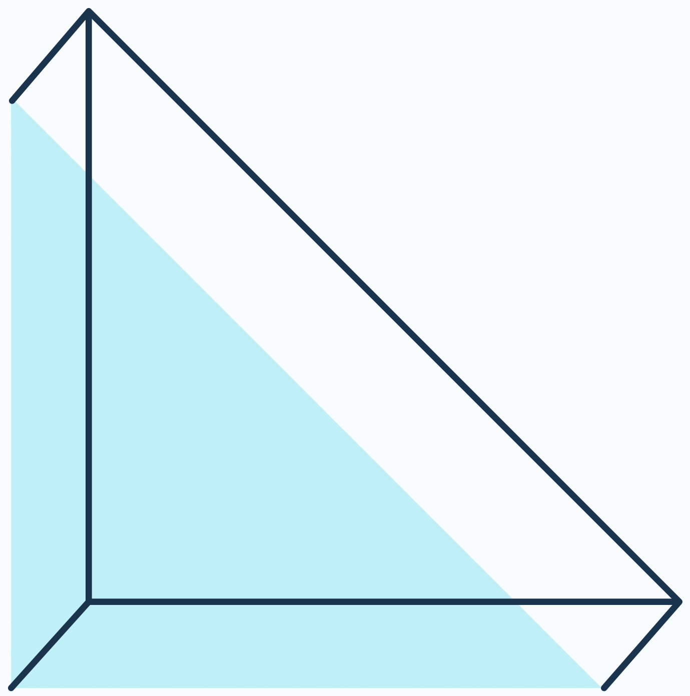
Von Neumann's machine consisted of an infinite, two-dimensional grid of cells that could be in up to twenty-nine states and followed a large number of complex rules. It contained several sub-organisms that gathered materials from the environment, read the instructions and copied them, then performed the computation.



JOHN HORTON CONWAY

Dr. Conway, called Life a “no-player, never-ending game.”Whenever the subject came up, he would bellow, “I hate Life!” But in his final years he learned to love Life again.

1 Dimensional



Rules

The simplest class of one-dimensional cellular automata.

Elementary cellular automata have two possible values for each cell (0 or 1), and rules that depend only on nearest neighbor values.

As a result, the evolution of an elementary cellular automaton can completely be described by a table specifying the state a given cell will have in the next generation based on the value of the cell to its left, the value the cell itself, and the value of the cell to its right

Explanation

101 : TRUE

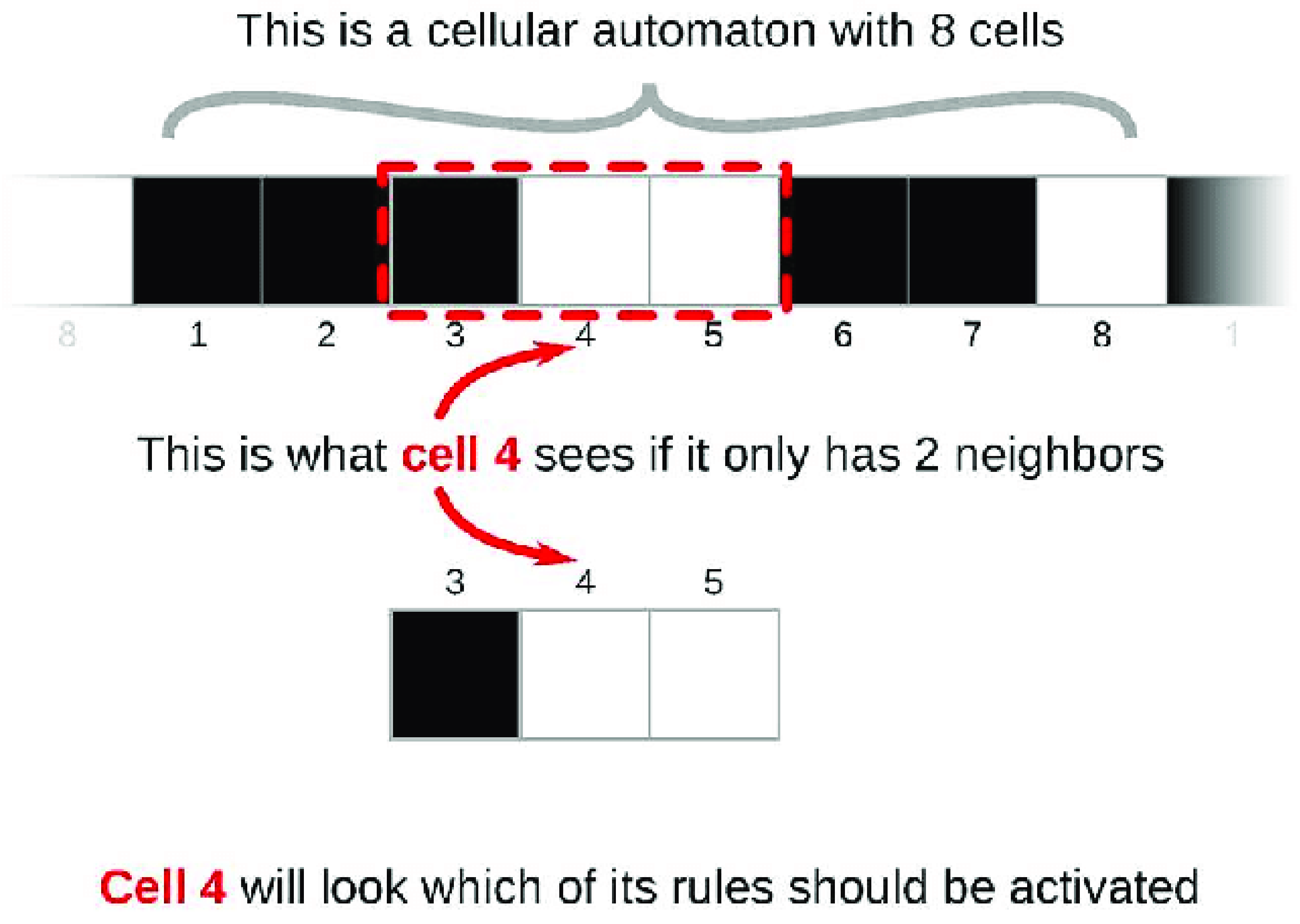
Element with 0 state and both neighbours 1,
will become 1 in next generation

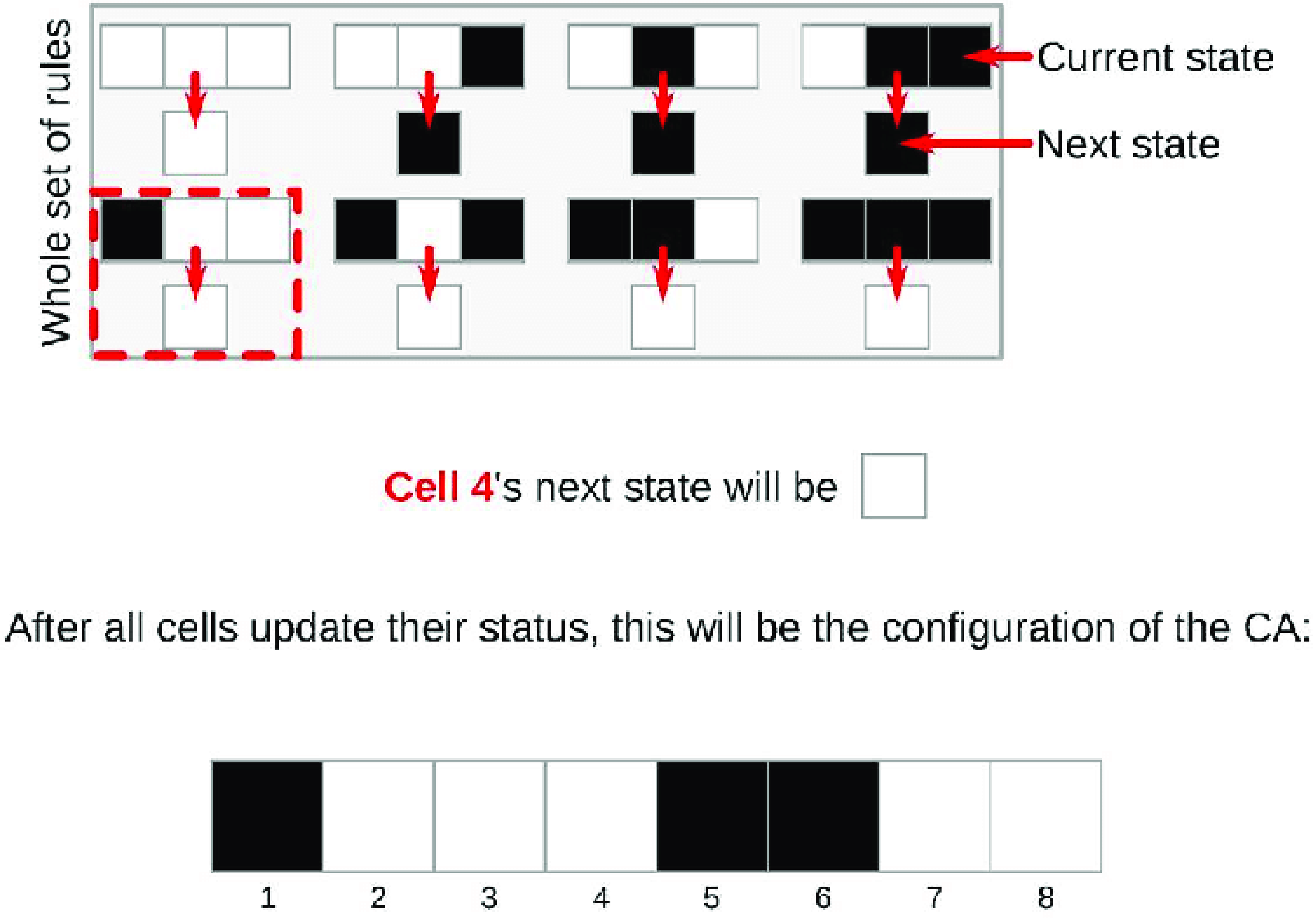
100 : FALSE

Element with 0 state and left neighbour 1, right neighbour 0,
will become 0 in next generation

RULE 30

Binary Representation of 30 is considered


This is a cellular automaton with 8 cells
8
1
2
3
4
5
6
7
8
1
This is what cell 4 sees if it only has 2 neighbors
3
4
5
Cell 4 will look which of its rules should be activated

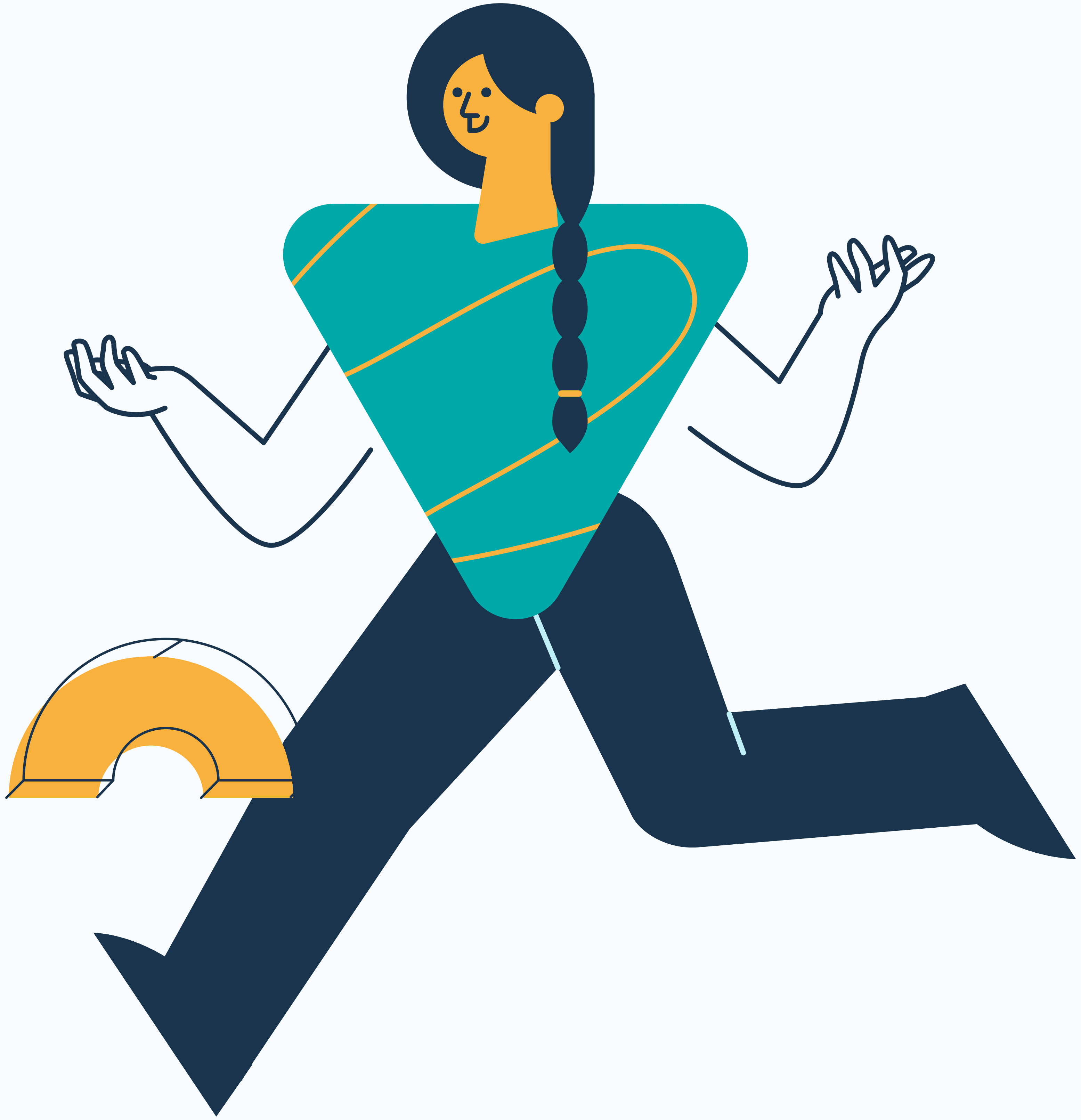

Whole set of rules
Current state
Next state
Cell 4's next state will be
After all cells update their status, this will be the configuration of the CA:
1
2
3
4
5
6
7
8

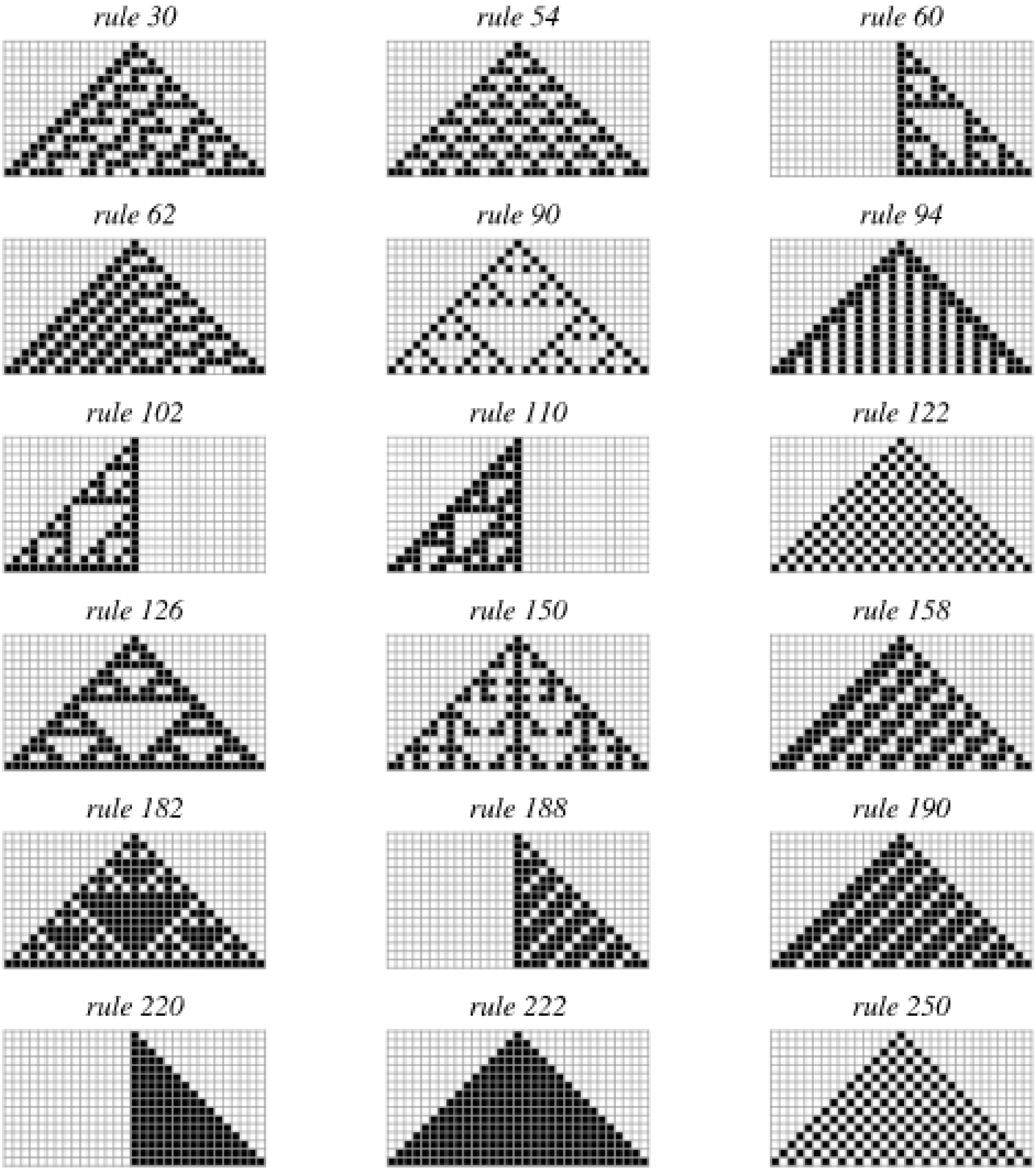
3Layer/sec

Naive Way

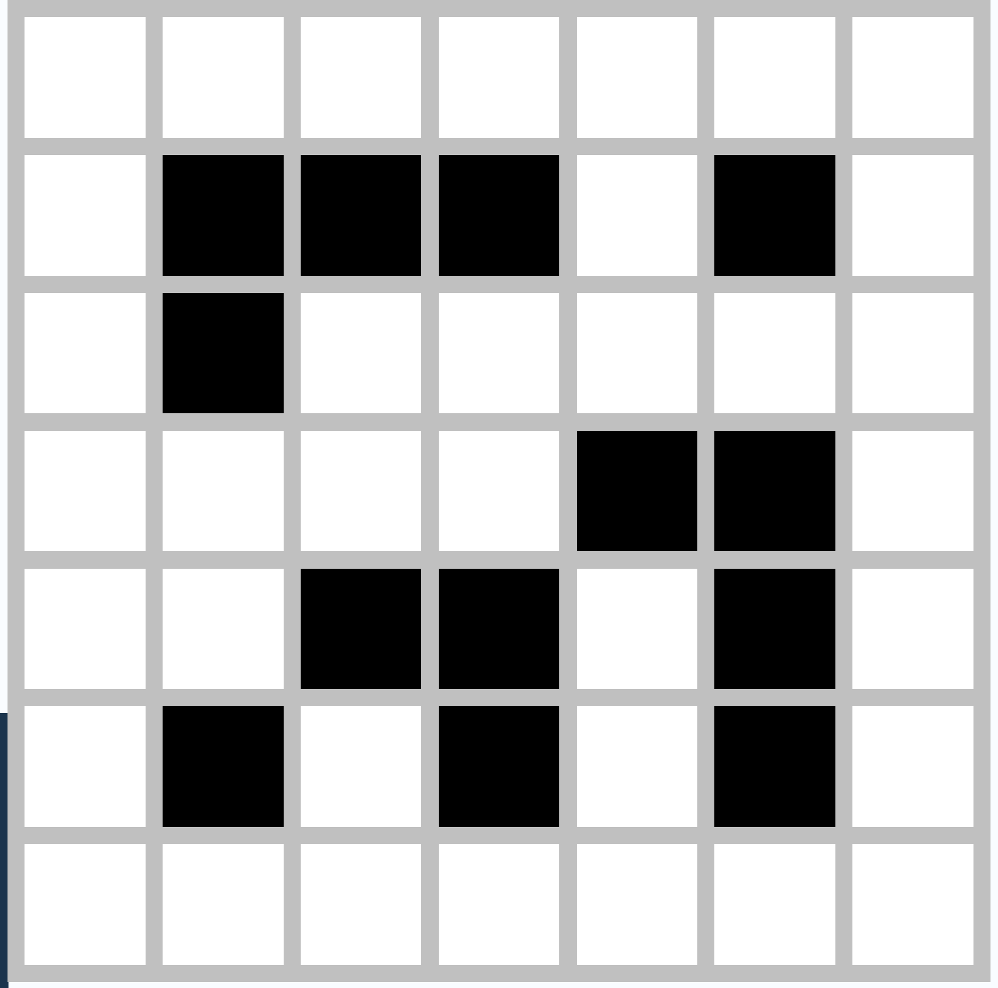
150Layer/sec

Optimised Way




rule 30
rule 54
rule 60
rule 62
rule 90
rule 94
rule 102
rule 110
rule 122
rule 126
rule 150
rule 158
rule 182
rule 188
rule 190
rule 220
rule 222
rule 250

2D : Game of Life



Birth

Any dead cell with three live neighbours becomes a live cell.

Survival

Any live cell with two or three live neighbours survives.

Death

All other live cells die in the next generation. Similarly, all other dead cells stay dead

THINK !

Brute Force

Implemented a naive solution, where we traverse through each box and check his neighbours each time

Not better

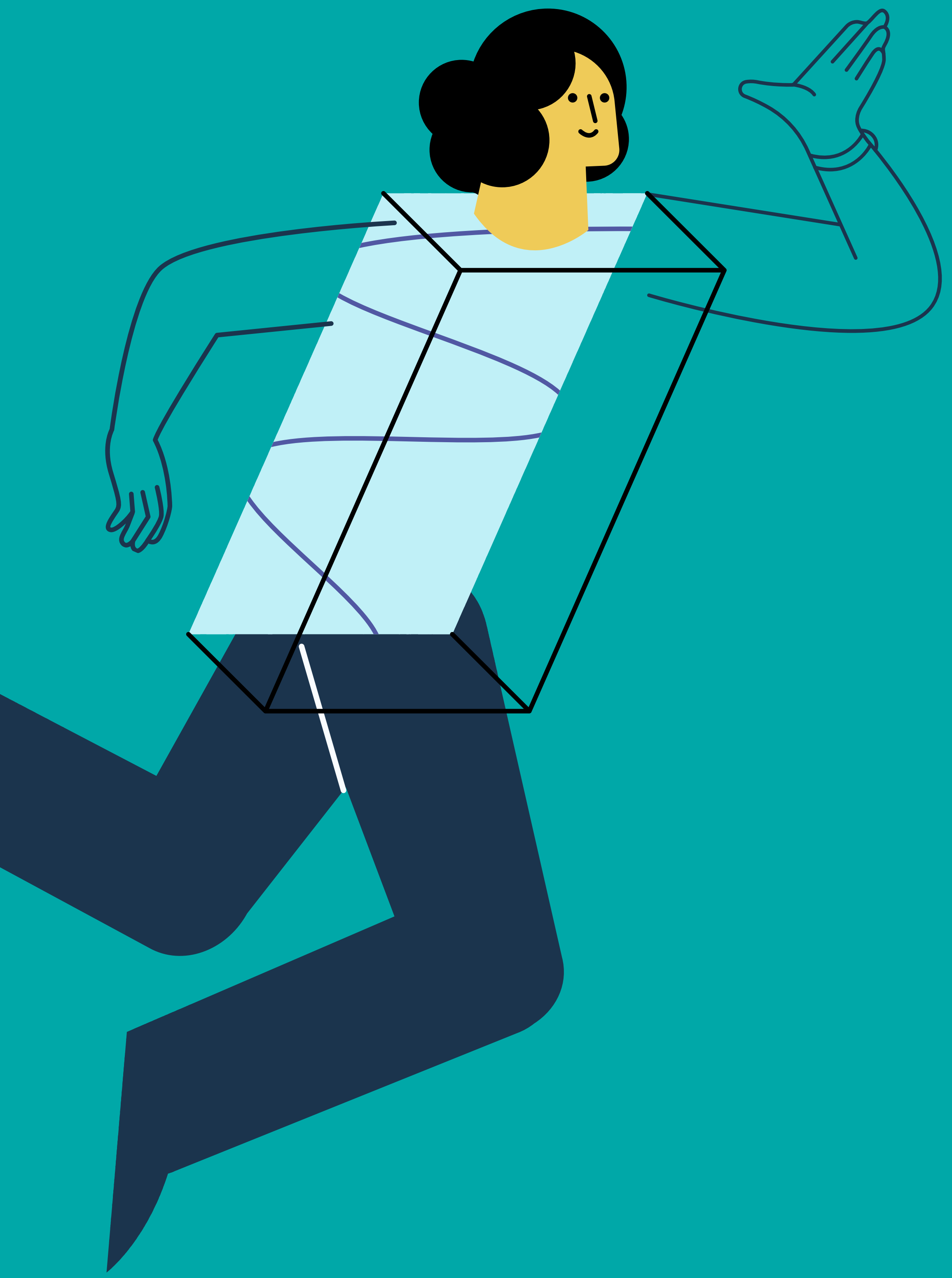
Put constraints on the traversal, Visit only live nodes and their neighbours, etc.

Bits

Use Bit manipulation to reduce calculations time complexity

Research !

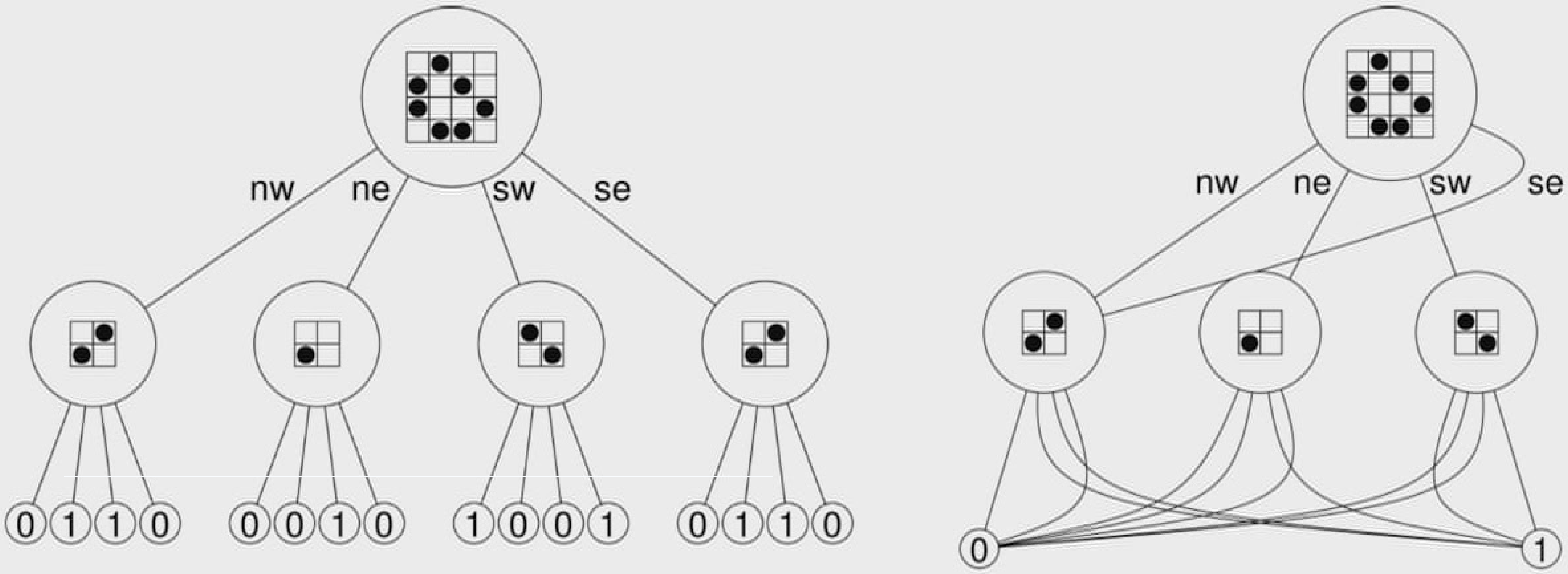
SuperSpeed Method Found



What Plan did we had in mind?

The Ins and Outs

Canonicalized Quadtrees



Each node has a **level** k, where the size of the node is $2^k \times 2^k$ cells.

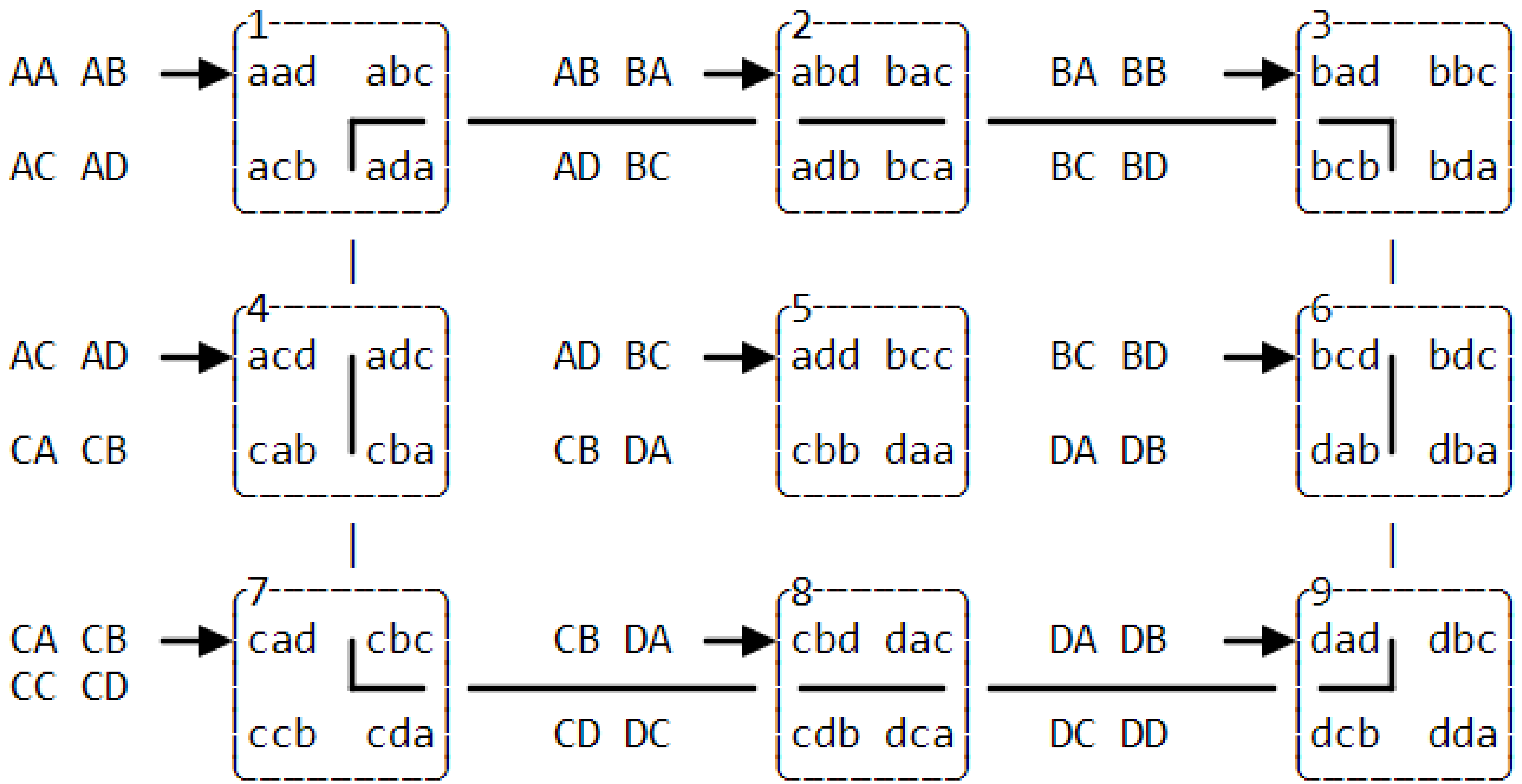
- level 0 is a 1×1 block (leaf node) which are just binary values, on or off.
- level 1 is a 2×2 block whose children are level 0 blocks
- level 2 is 4×4 block whose children are level 1 blocks
- level 3 is an 8×8 block whose children are level 2 blocks

The Hashlife algorithm defines an recursive process that:

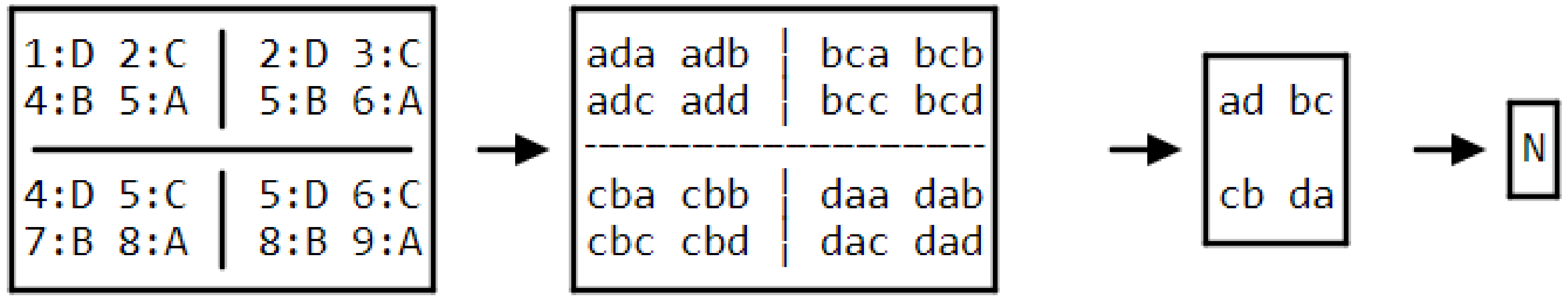
- takes a level k node, size $2^k \times 2^k$
- returns a level k-1 node, size $2^{k-1} \times 2^{k-1}$
- advanced 2^{k-2} generations in time.

Eventually, we end up processing blocks of size 4×4 ($k = 2$), where we can use basic brute-force, computing the 2×2 successor of a 4×4 cell by straightforward computation. The clever part is that by memoizing the recursion to cache the intermediate products, we can *dramatically* reduce computation requirements, as most CA patterns are very repetitive in space and in time.

AAA	AAB	ABA	ABB	BAA	BAB	BBA	BBB
AAC	AAD	ABC	ABD	BAC	BAD	BBC	BBD
ACA	ACB	ADA	ADB	BCA	BCB	BDA	BDB
ACC	ACD	ADC	ADD	BCC	BCD	BDC	BDD
CAA	CAB	CBA	CBB	DAA	DAB	DBA	DBB
CAC	CAD	CBC	CBD	DAC	DAD	DBC	DBD
CCA	CCB	CDA	CDB	DCA	DCB	DDA	DDB
CCC	CCD	CDC	CDD	DCC	DCD	DDC	DDD

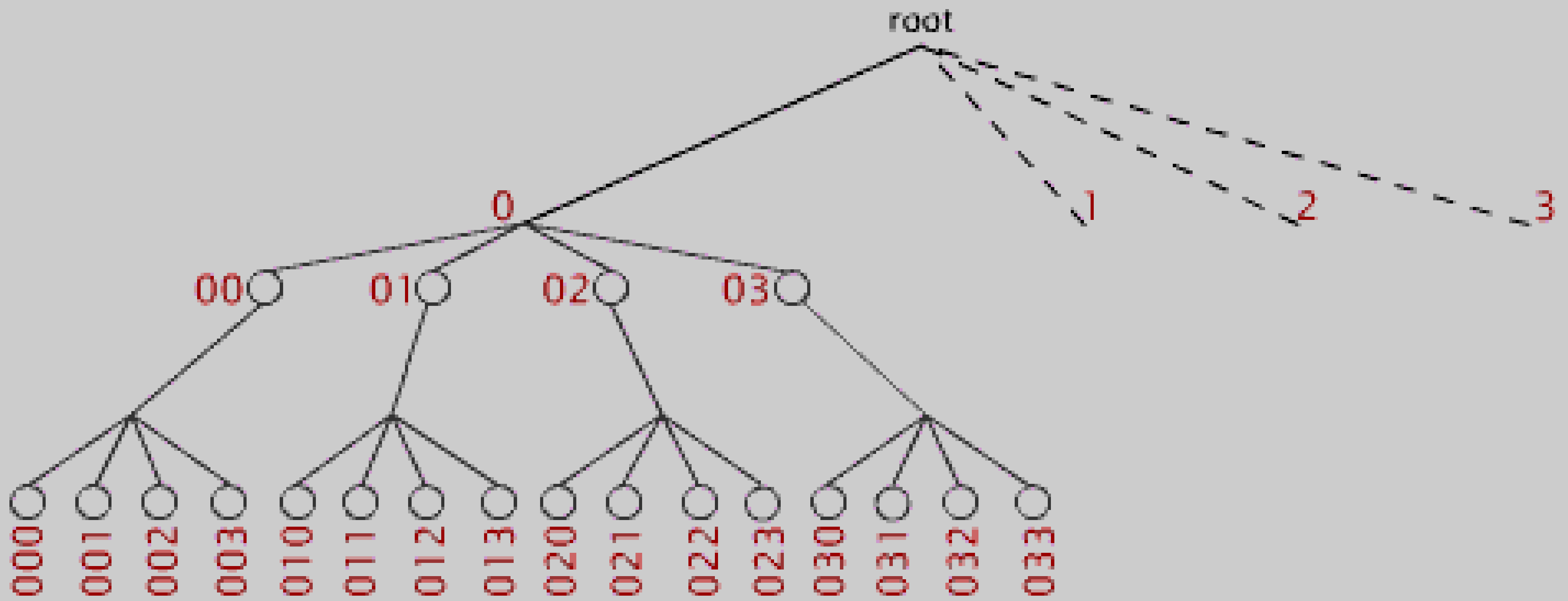


Finally, we can select from those great-grandchildren sized successor blocks the "inner" parts to make up one full child-sized successor (a 4×4 block of great-grandchild successors)

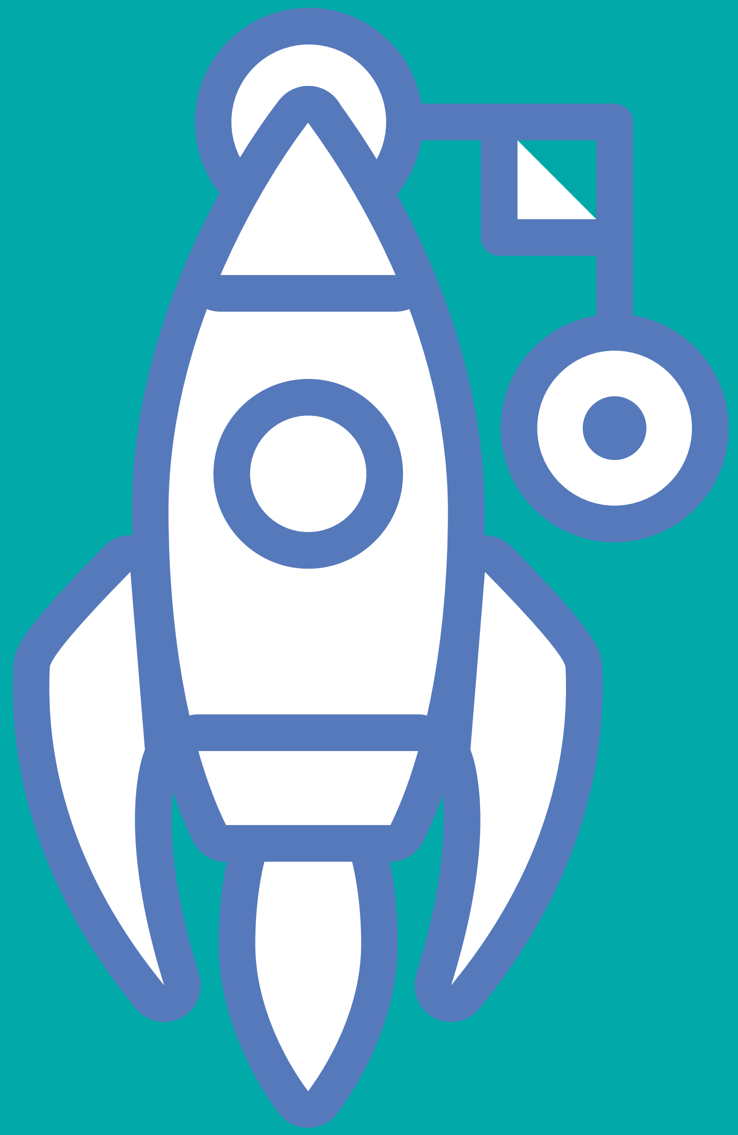


Hashlife is a memoized algorithm for computing the long-term fate of a given starting configuration in Conway's Game of Life and related cellular automata, much more quickly than would be possible using alternative algorithms that simulate each time step of each cell of the automaton.

The algorithm was first described by Bill Gosper in the early 1980s while he was engaged in research at the Xerox Palo Alto Research Center.



000	001	010	011	100	101	110	111
002	003	012	013	102	103	112	113
020	021	030	031	120	121	130	131
022	023	032	033	122	123	132	133
200	201	210	211	300	301	310	311
202	203	212	213	302	303	312	313
220	221	230	231	320	321	330	331
222	223	232	233	322	323	332	333



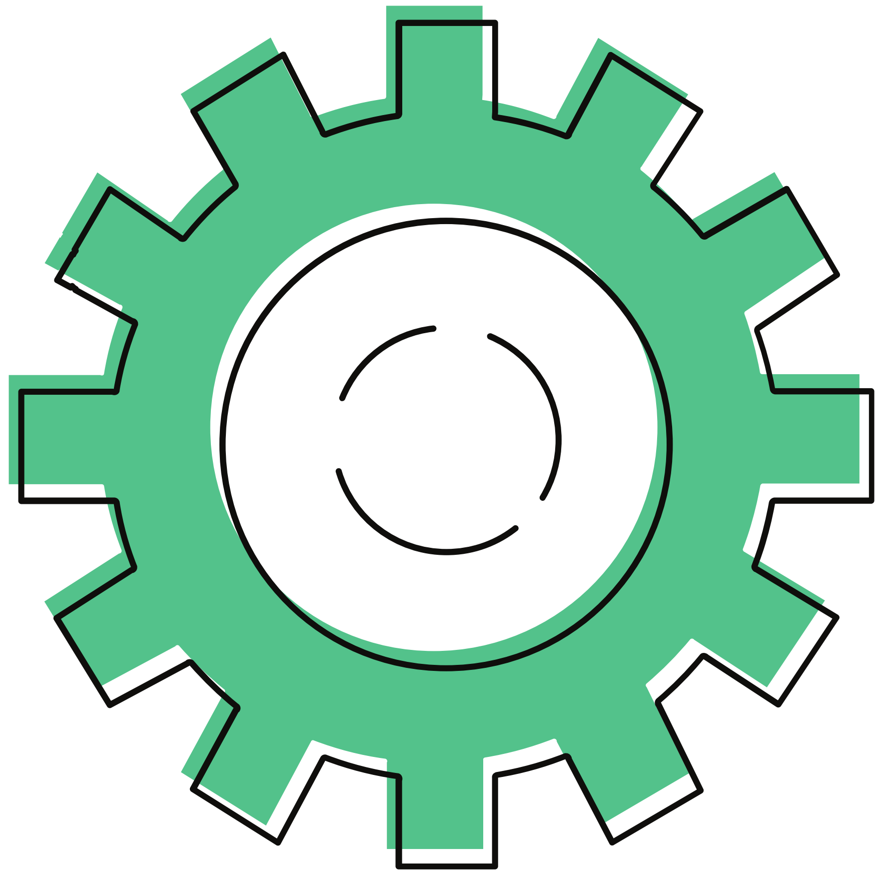
Superspeed and caching

One could compute twice the number of generations forward for a node at the (k+1)-th level compared to one at the kth.

To take full advantage of this feature, subpatterns from past generations should be saved as well.

The typical behaviour of a Hashlife program on a conducive pattern is as follows: first the algorithm runs slower compared to other algorithms because of the constant overhead associated with hashing and building the tree; but later, enough data will be gathered and its speed will increase tremendously – the rapid increase in speed is often described as "exploding".

Applications



- Traffic
- Game theory (Firing squad synchronization problem, Majority problem)
- Pseudo-randomness
- Computing with particle colliding
- Quantum gravity: Fredkin and Wolfram are strong proponents of CA-based physics and in 2016, Gerard 't Hooft published a book-length development of the idea to rebuild quantum mechanics using cellular automata.

- Biology
- Chemical Reactions
- Artifical Intelligence

THANK YOU