> A data structure is just a stupid programming language.
>
> — *Bill Gosper*
>
> A programming language is a very dangerous data structure.
>
> — *Guy Steele*

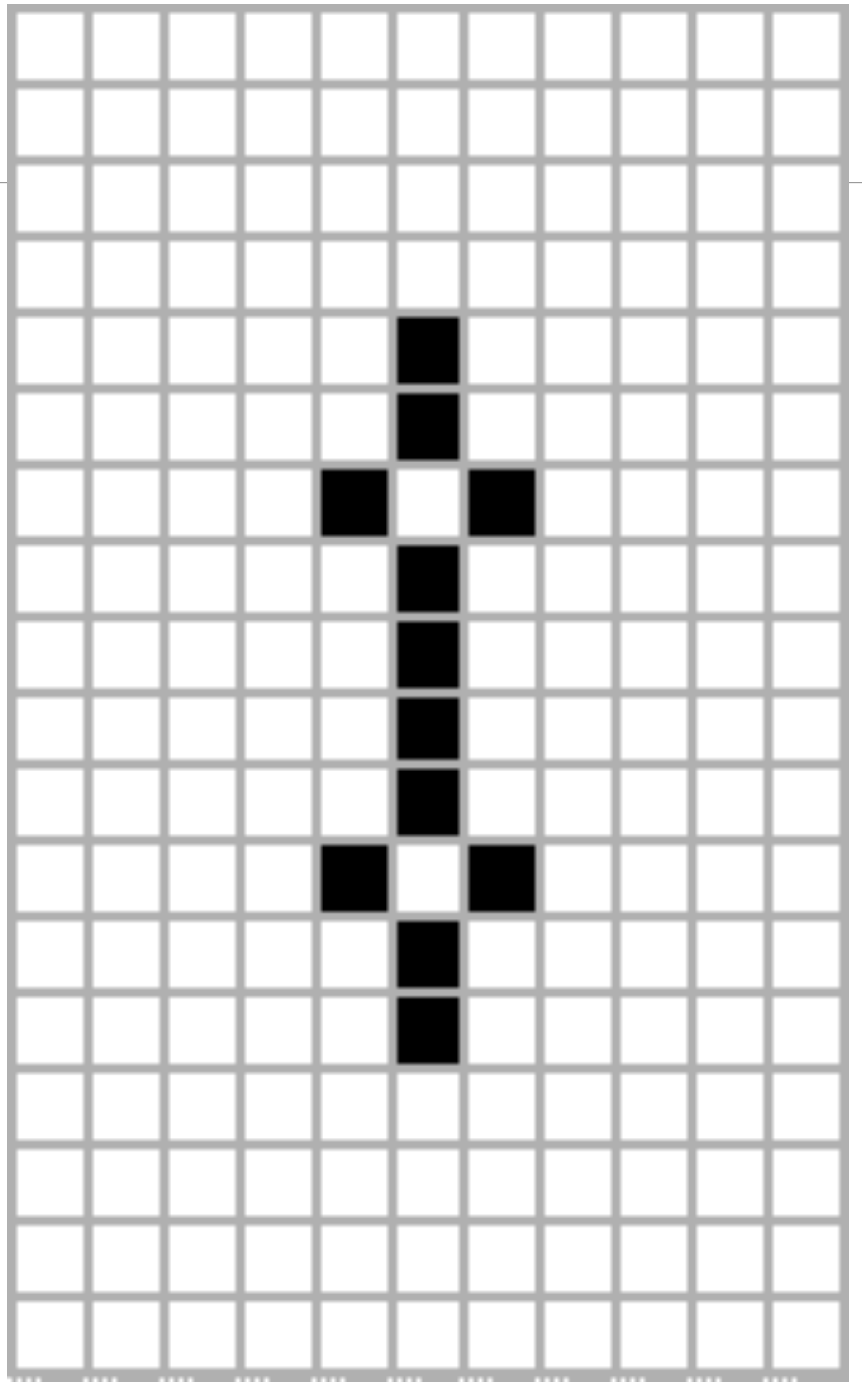
HashLife

Jeff Erickson

April 29, 2024

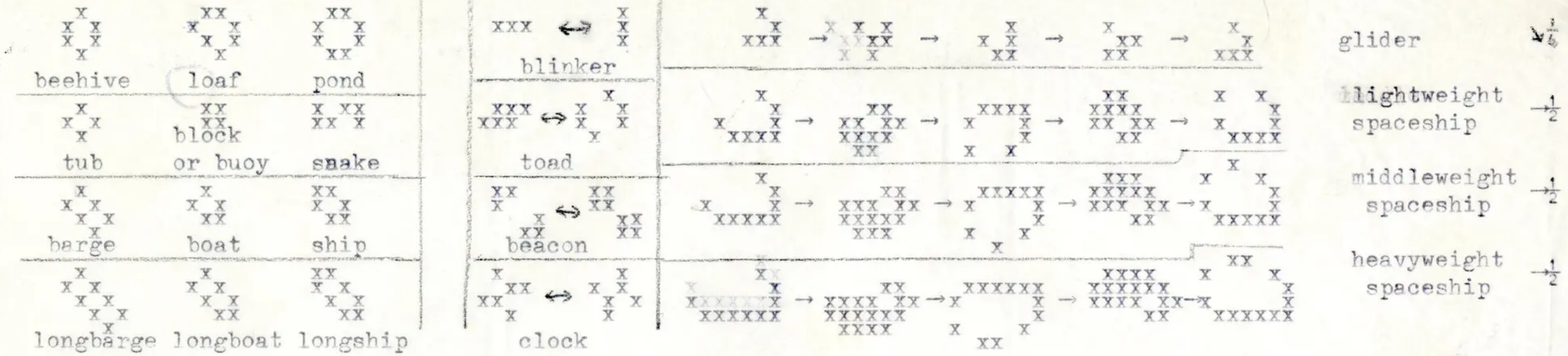
The Game of Life

- Designed by John H. Conway in 1970
- Infinite grid of ***cells***, each ***alive*** or ***dead***.
- Ruleset **23/3**: In each ***generation***...
  - **Live** cells with **2 or 3** live neighbors stay alive
  - **Dead** cells with **3** live neighbors comes to life
  - All other cells become/stay dead

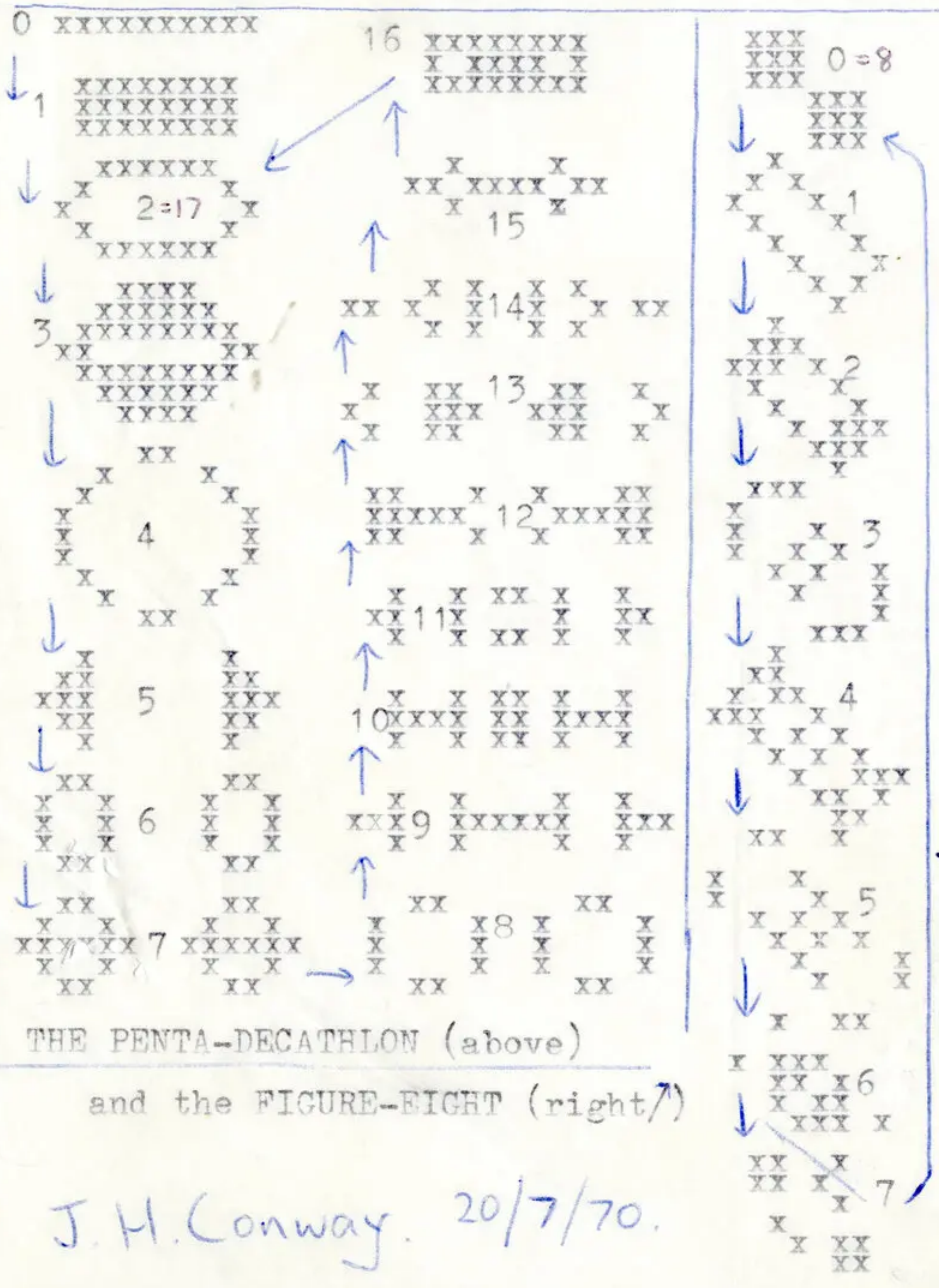


Pentadecathlon *[Conway 1970]*

A SURVEY OF LIFE-FORMS

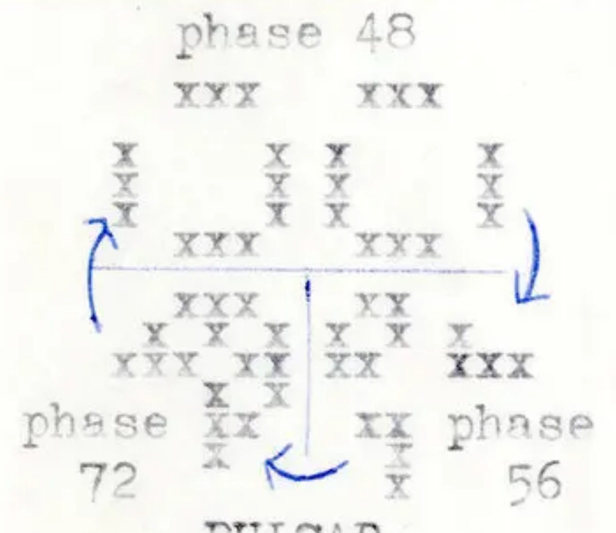


THE COMMONEST STILL-LIFE FLIP-FLOPS ALL KNOWN SPACE-SHIPS.

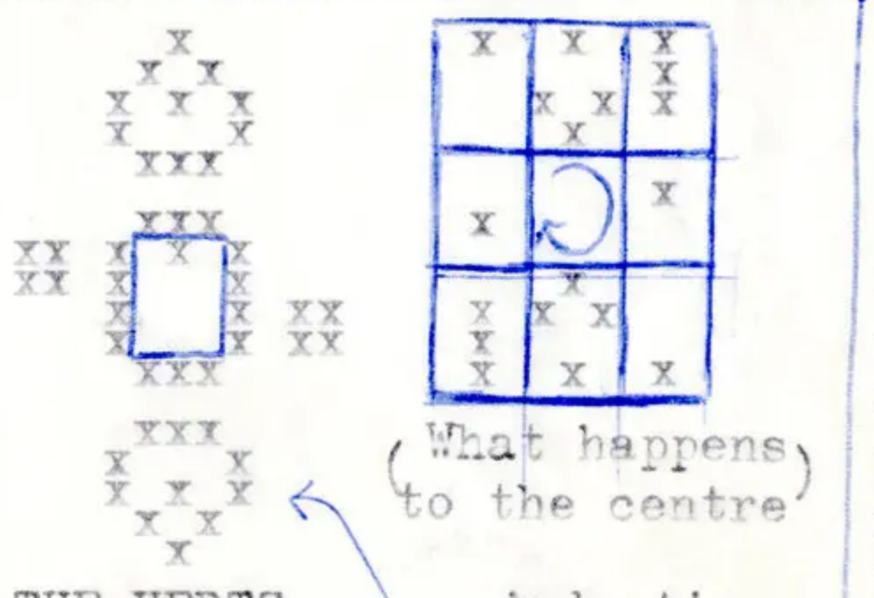


THE PENTA-DECATHLON (above)

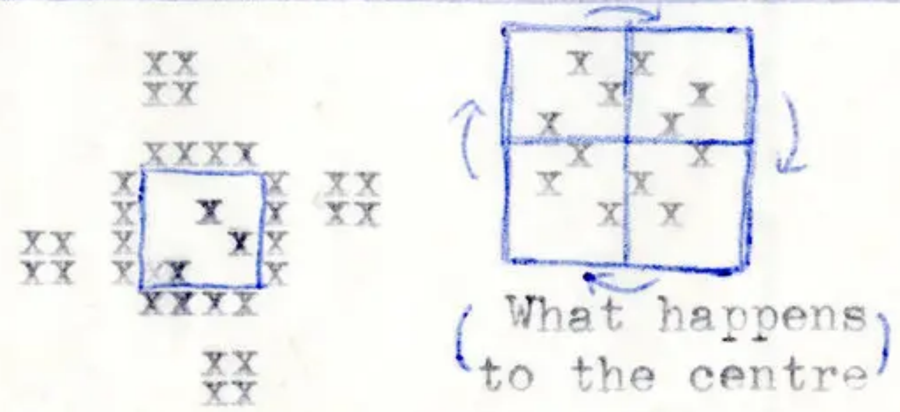
and the FIGURE-EIGHT (right)



PULSAR
CP 48-56-72
(a composite figure)



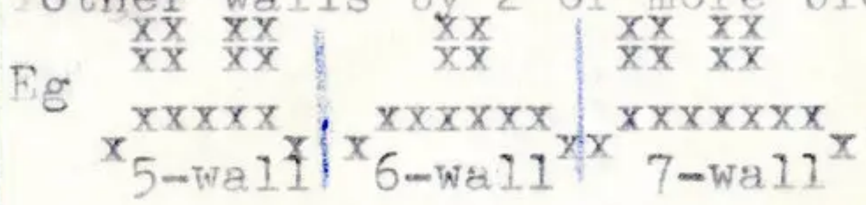
THE HERTZ OSCILLATOR



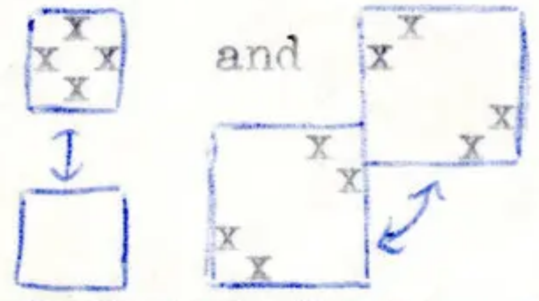
NORTON'S PINWHEEL

3-wall retained by induction-coil
4-wall or 6-wall by a block,
other walls by 2 or more blocks.

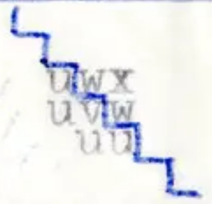
Eg



The PINWHEEL and OSCILLATOR are examples of Norton's 'BILLIARD-TABLE' configurations. Other centres are



Speeds measured in terms of lightspeed=kingspeed.

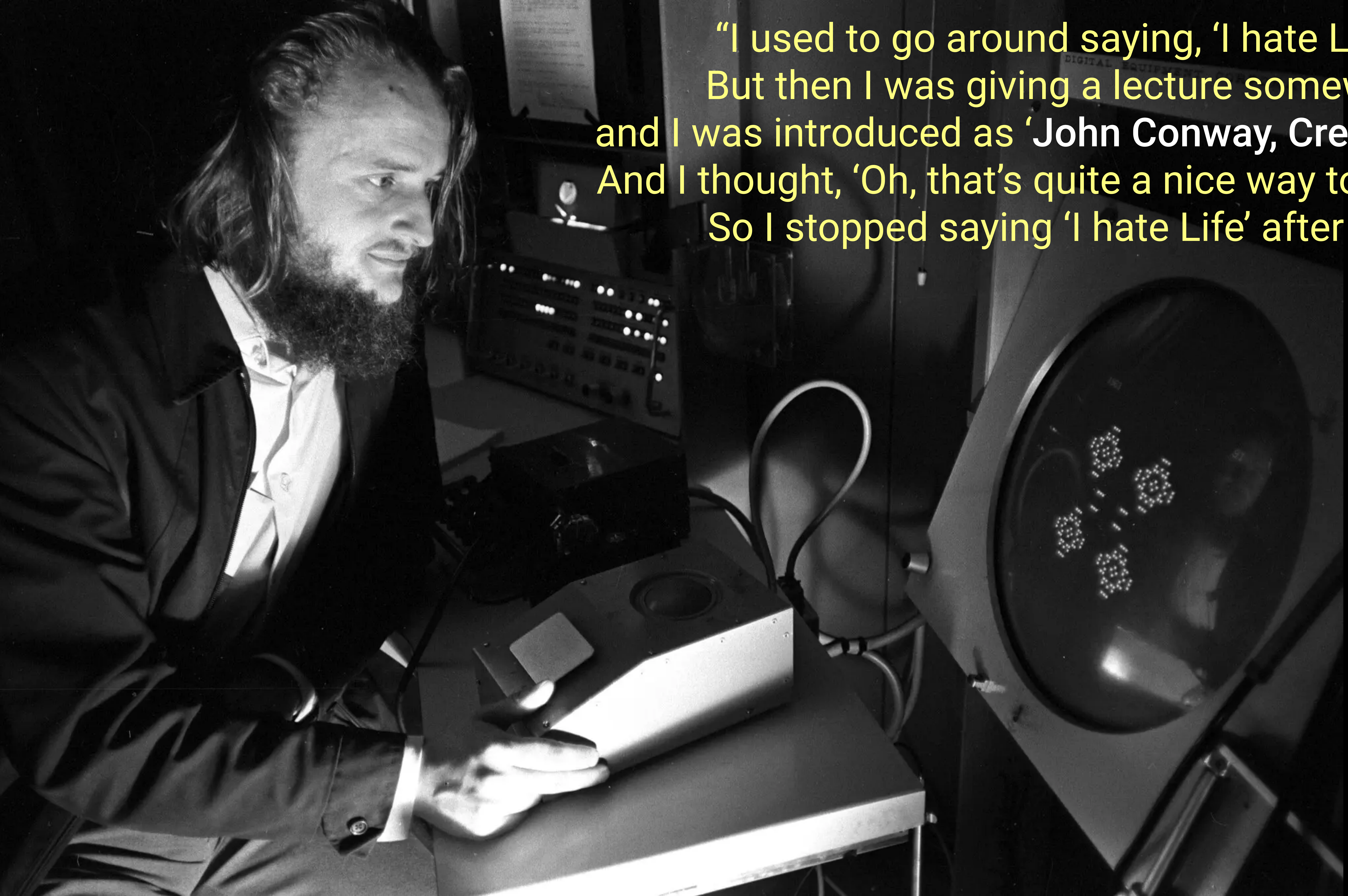


If a population is below the indicated diagonal at time 0, then it can't include x at time 2. If it did, then all the squares u & v would be there at time 1, and so (to get w) all squares u and v would be there at time 0. But then v would be killed off by its 4 neighbours u, a contradiction.

This proves diagonal speed into empty space is at most 1/4.

As a corollary, an object to the left of the 'V' at time 0 can be at most one place beyond it at time 2. So horizontal speed of finite object ≤ ½.

J. H. Conway. 20/7/70.


"I used to go around saying, 'I hate Life,'...
But then I was giving a lecture somewhere,
and I was introduced as 'John Conway, Creator of Life.'
And I thought, 'Oh, that's quite a nice way to be known.'
So I stopped saying 'I hate Life' after that."

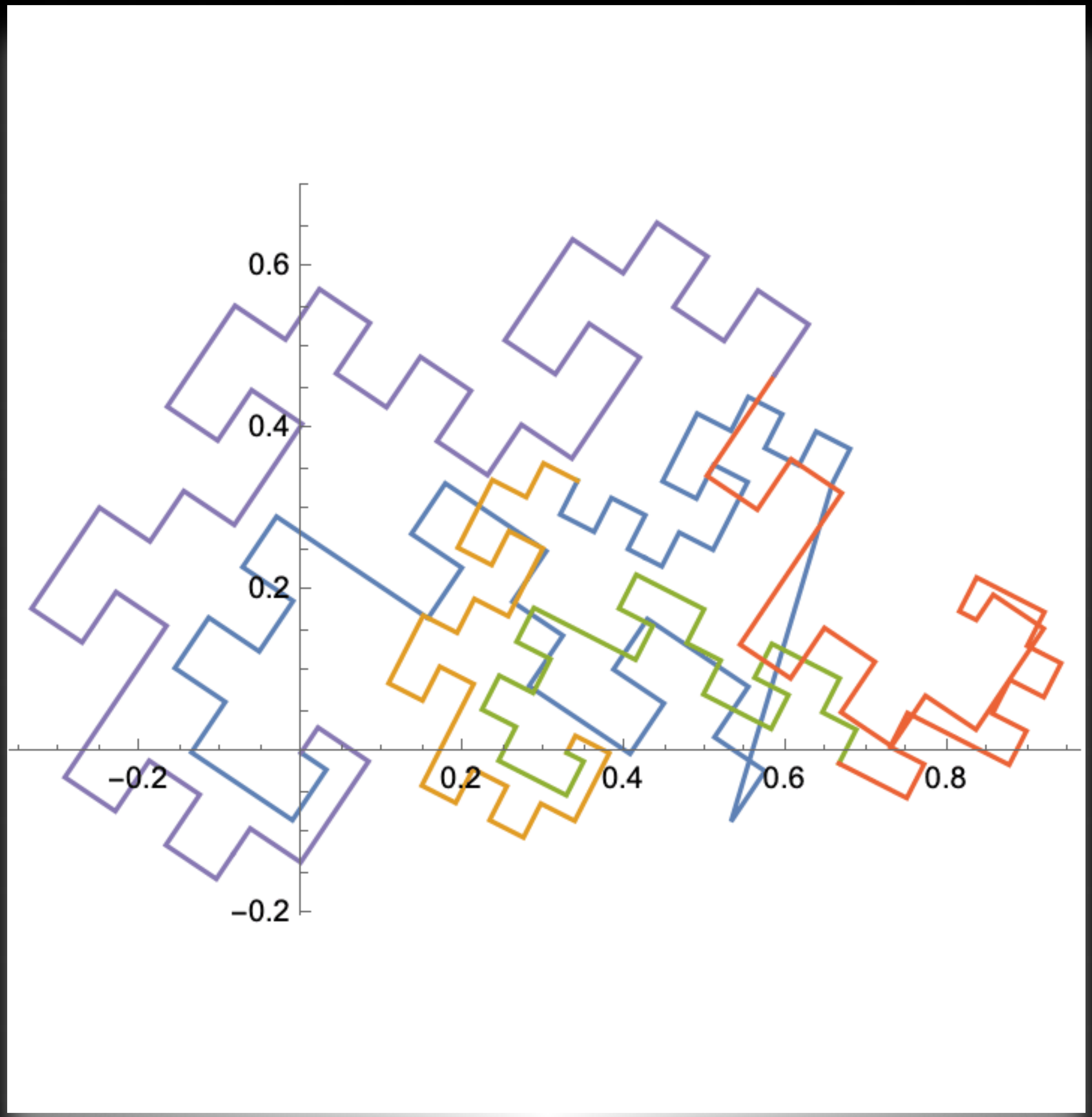
HAKMEM

by

M. Beeler

R.W. Gosper

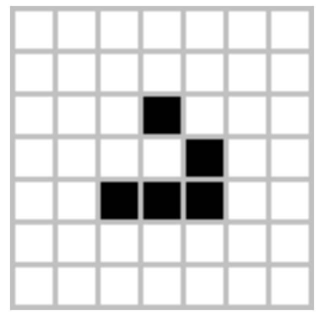
R. Schroeppel



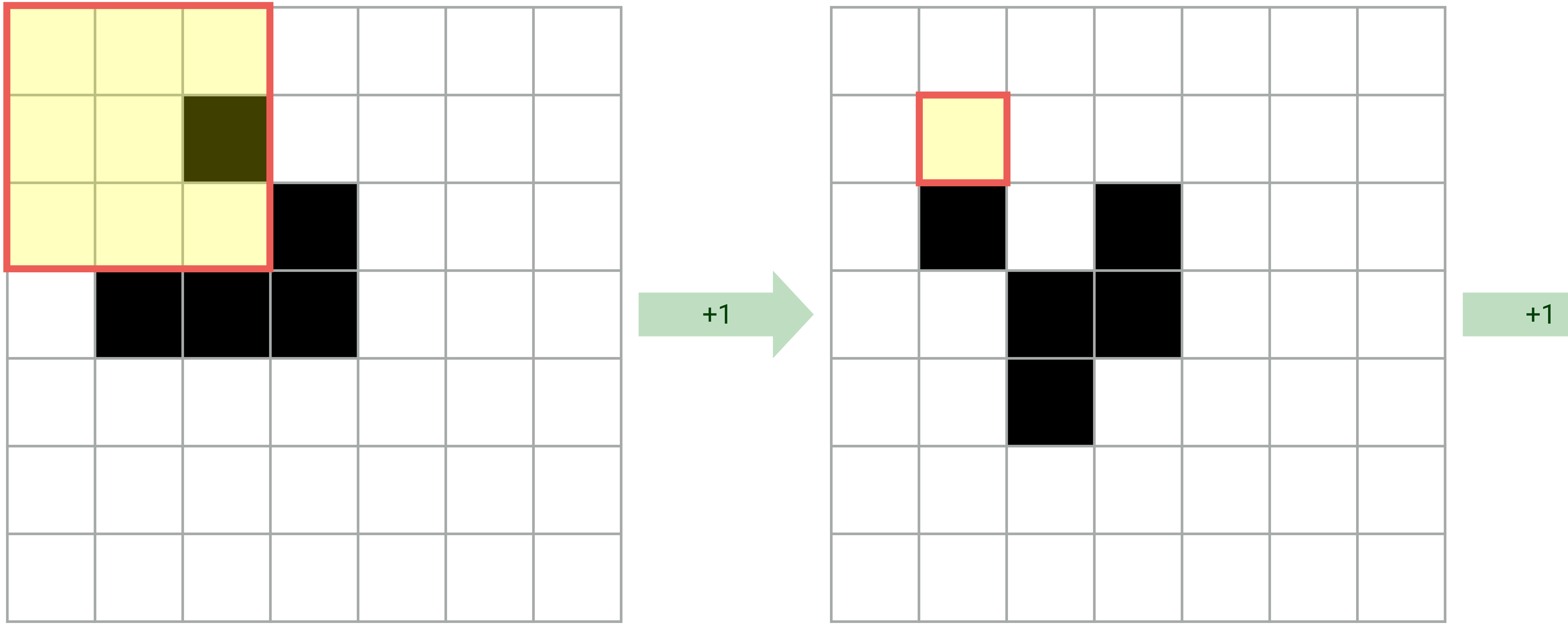
If arithmetic overflow is a fatal error, some fascist pig with a read-only mind is trying to enforce machine independence.

Abstract: Contrary to everybody, this self contained paper will show that continued fractions are not only perfectly amenable to arithmetic, they are amenable to perfect arithmetic.

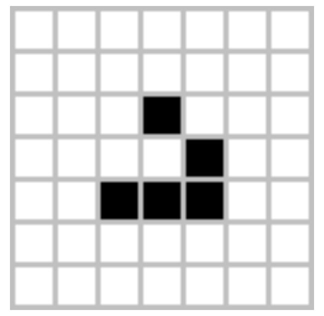
Rule 23/3



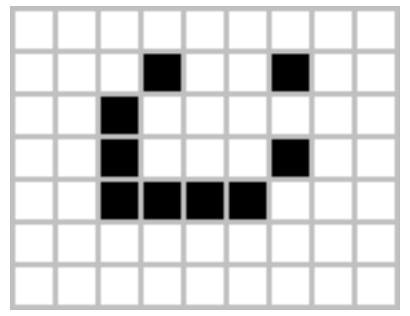
Glider [Guy 1969]



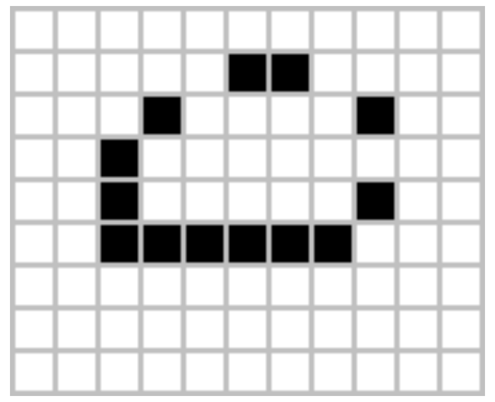
Gallery: Spaceships



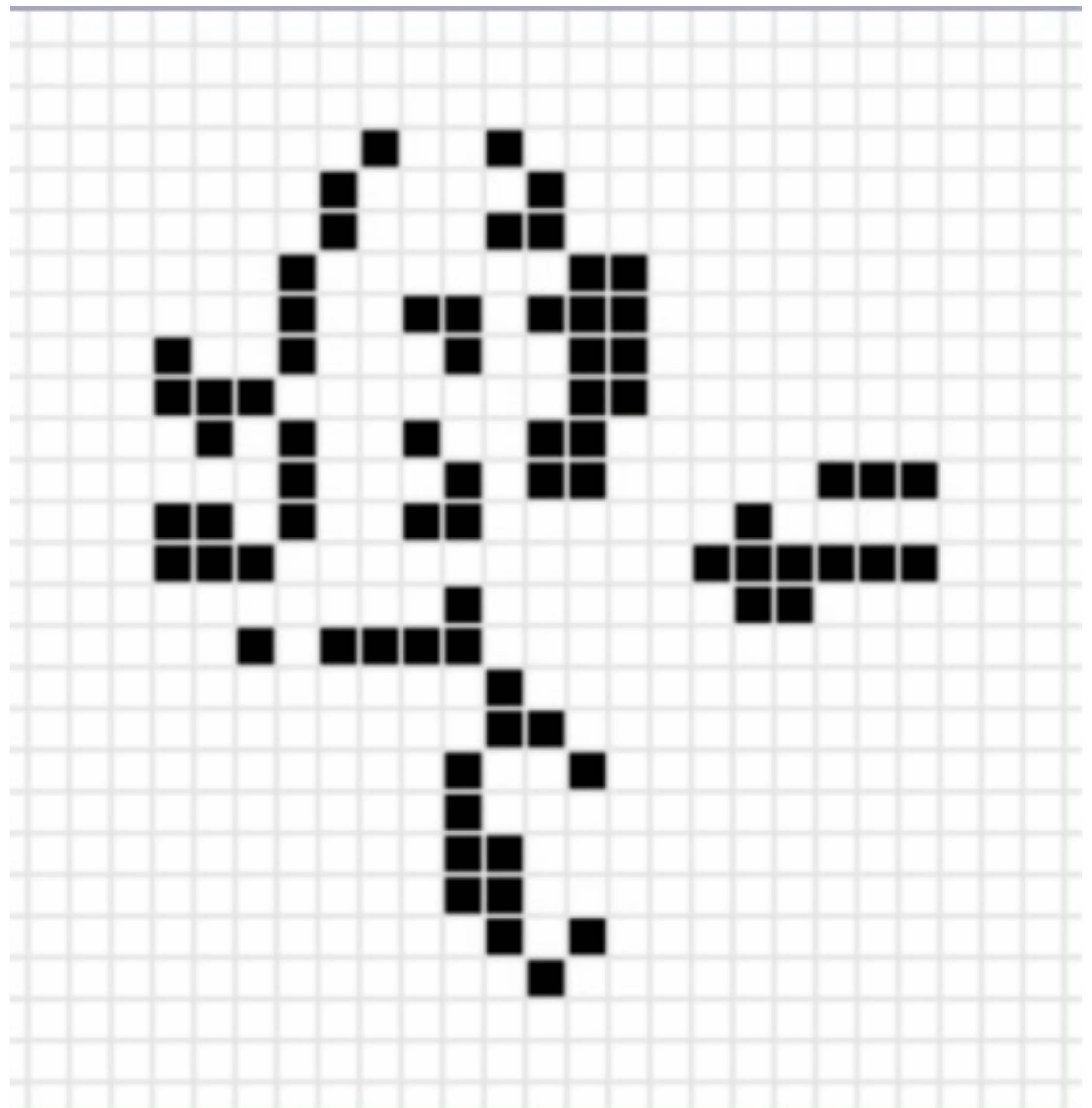
Glider *[Guy 1969]*



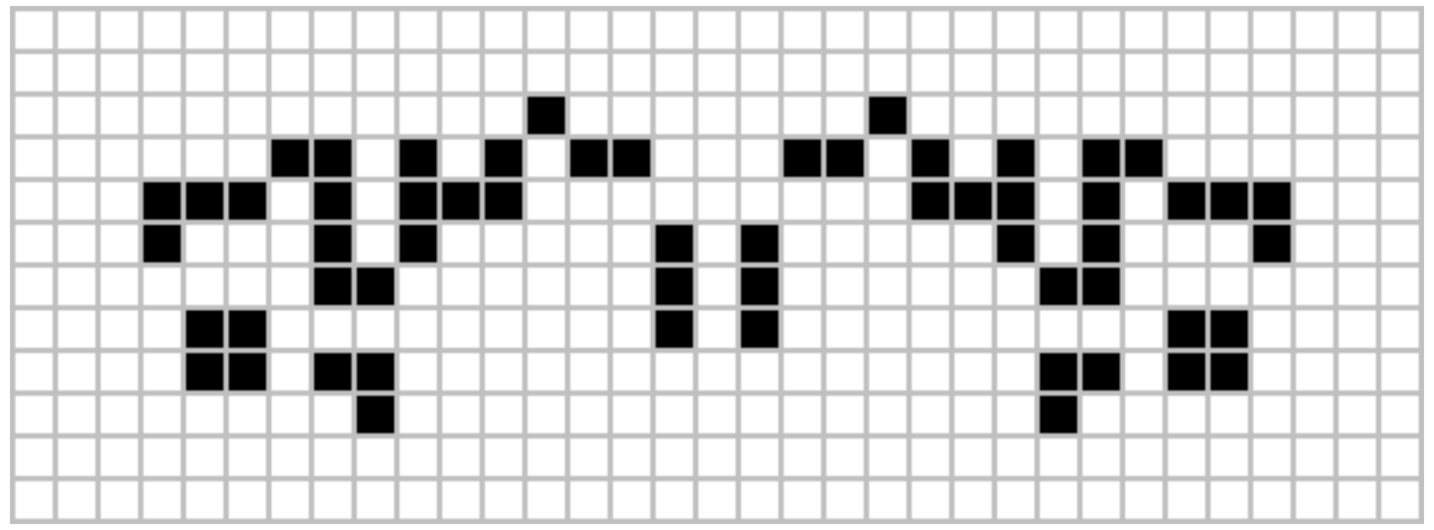
Lightweight *[Conway 1970]*



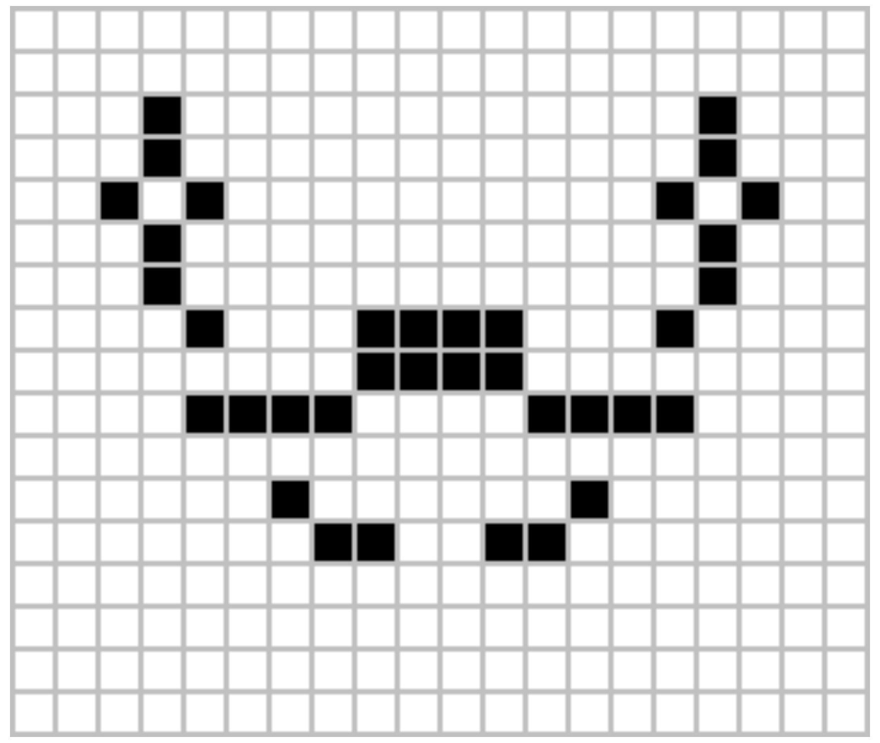
Heavyweight *[Conway 1970]*



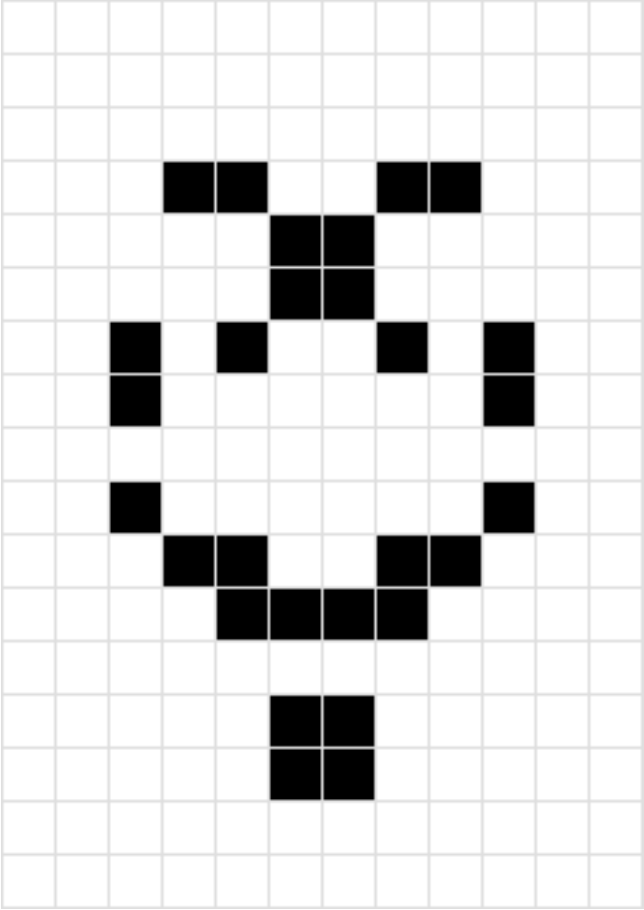
Walrus *[Gaucher 2023]*



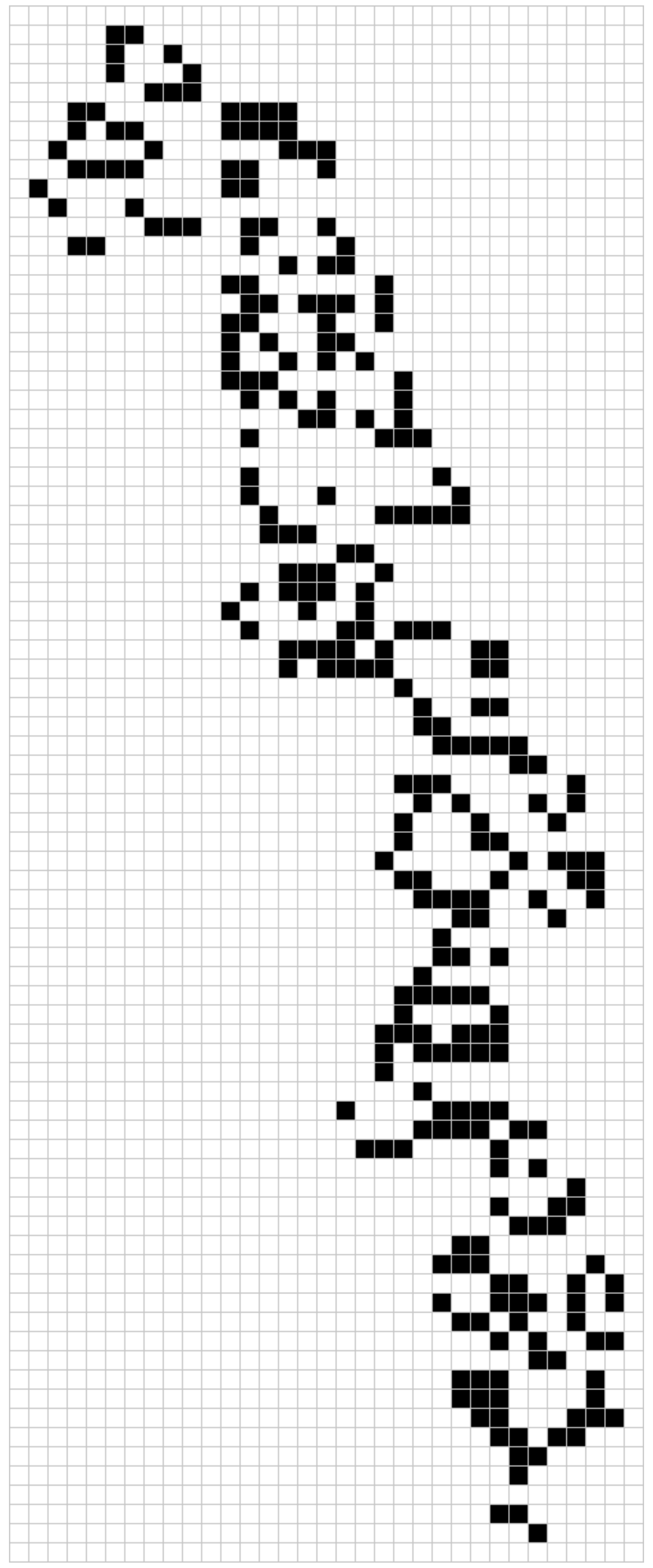
Spider *[Bell 1997]*



Weekender *[Eppstein 2000]*



Copperhead *[zfind 2016]*

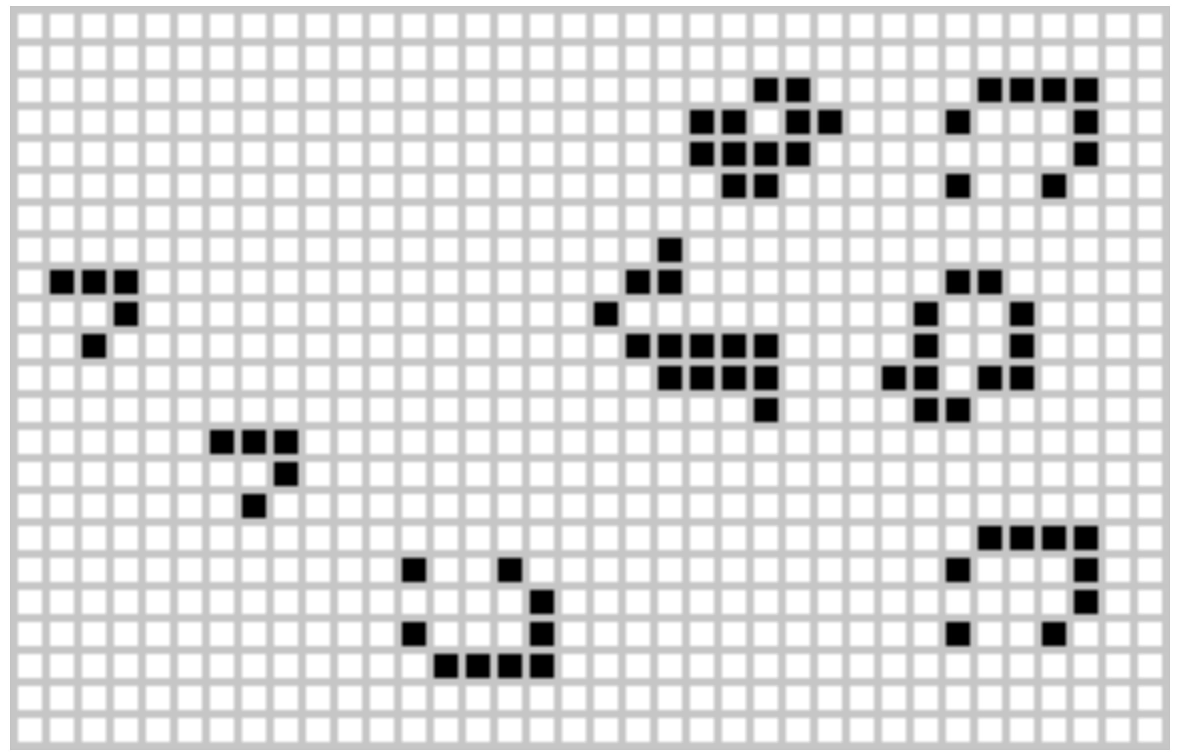


Sir Robin *[Goucher 2018]*

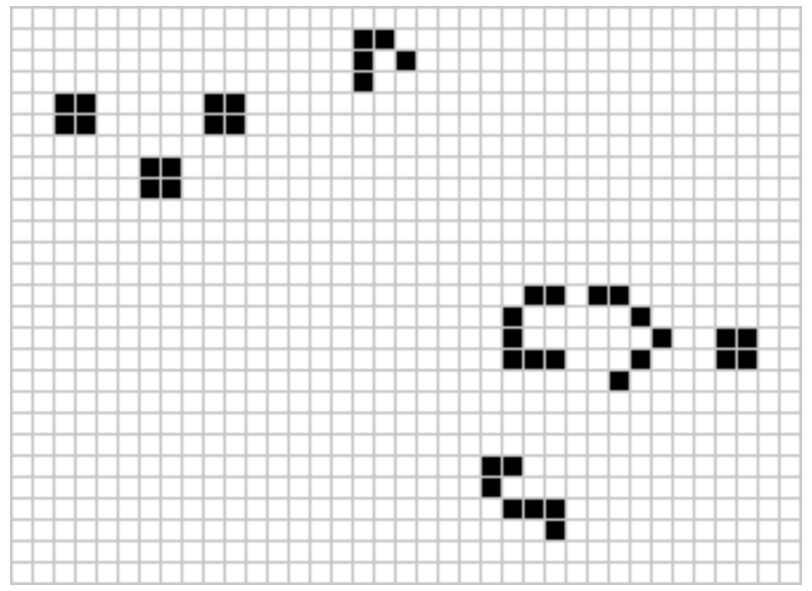
Gallery: Guns and Rakes

Glider Gun *[Gosper 1970]*

Gliderless LWSS gun
[Merzinich 2023]

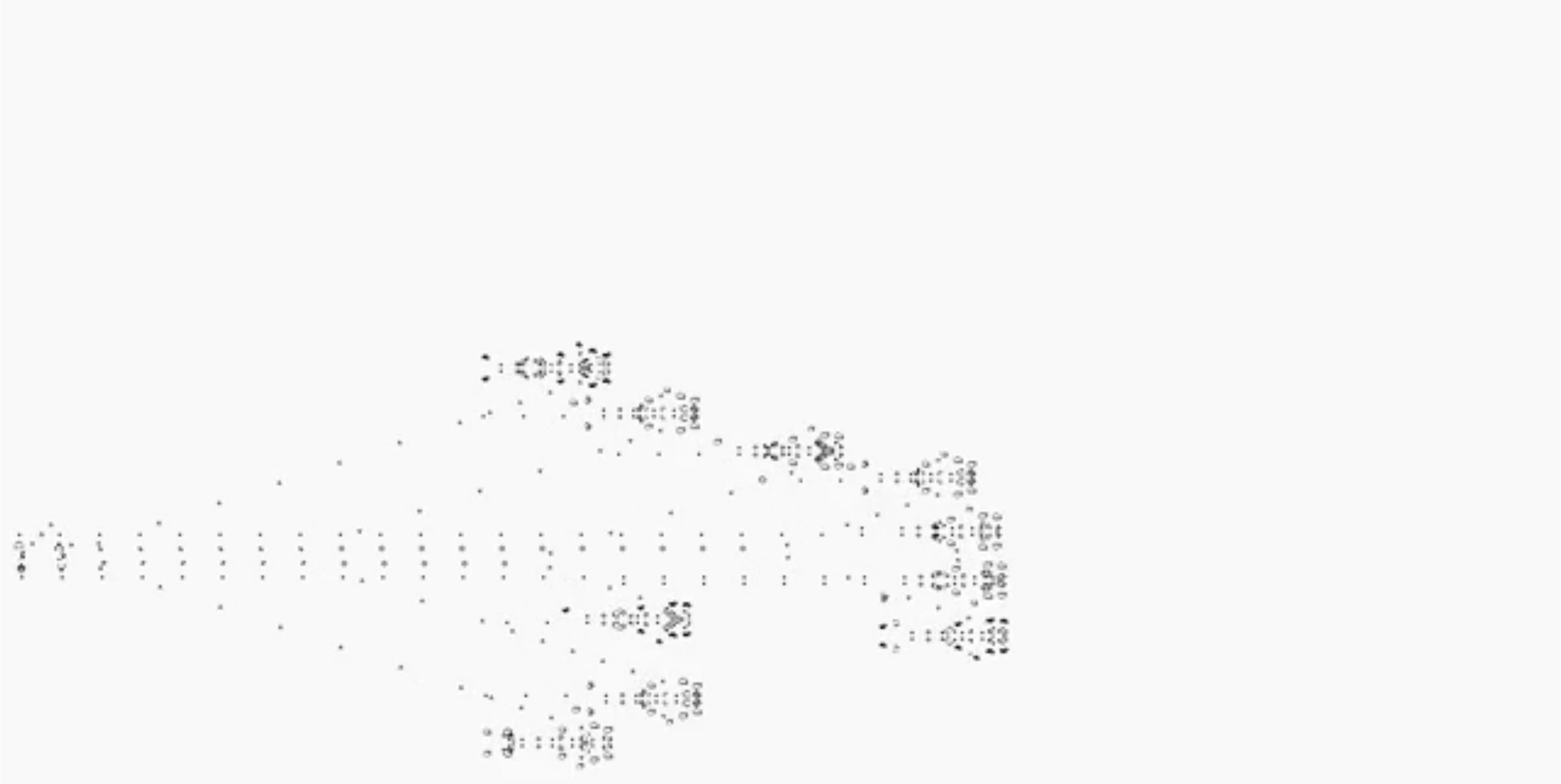


Spacerake
[Gosper(?) 1971]



Glider Gun *[Simkin 2015]*

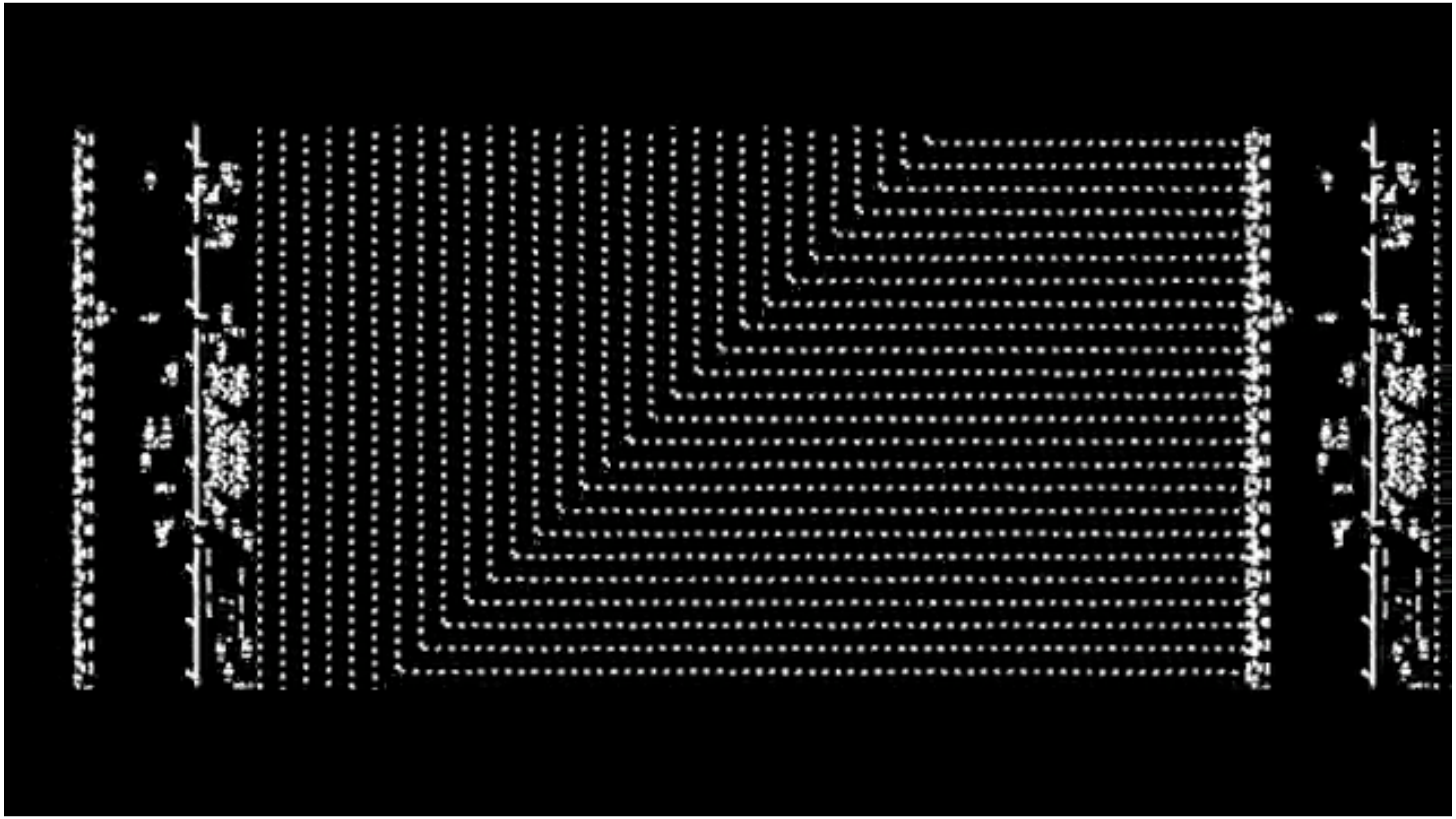
Gallery: Breeders



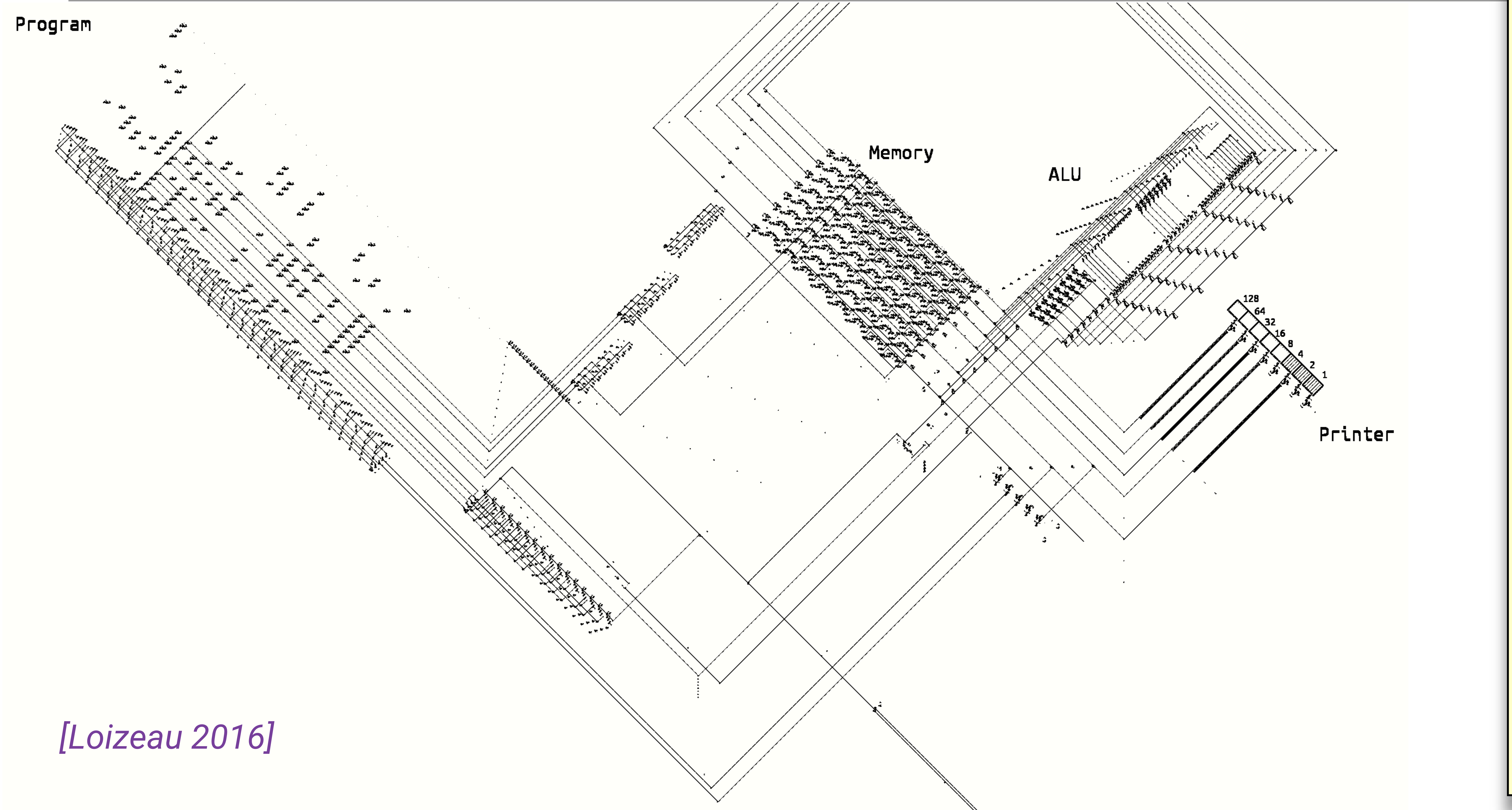
Breeder 1
[Gosper ≤1971]

Gallery: Life in Life

OTCA metapixel *[Due 2006]*
Video: *[Pincombe 2012]*



Gallery: Programmable computer



[Loizeau 2016]

print primes starting at 5

```
write f 224
write g 2
write a 5
write b 3
write c 0
not a e
increment e
add b e d
increment c
add d b d
and d f e
jumpif e
goto 14
goto 8
jumpif d
goto 24
not c e
add b e e
sign e e
jumpif e
goto 23
add b g b
goto 4
print a
add a g a
goto 3
```

CODE GOLF & coding challenges

Home

Questions

Tags

Users

Companies

Unanswered

TEAMS

Ask questions, find answers and collaborate at work with Stack Overflow for Teams.

Explore Teams

Build a working game of Tetris in Conway's Game of Life

Asked 10 years, 10 months ago Modified 13 days ago Viewed 303k times

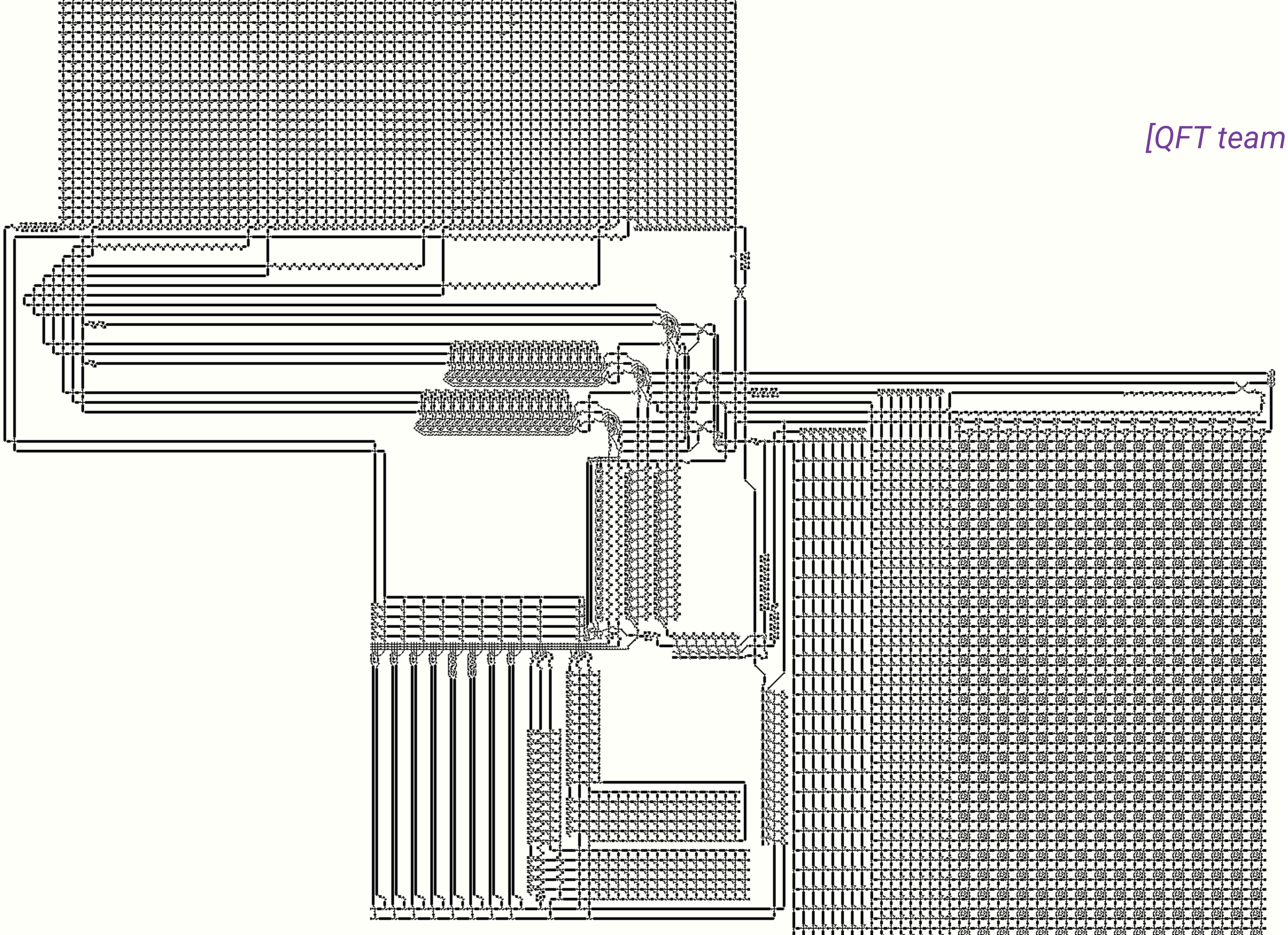
1234

Here is a theoretical question - one that doesn't afford an easy answer in any case, not even the trivial one.

In Conway's Game of Life, there exist constructs such as the metapixel which allow the Game of Life to simulate any other Game-of-Life rule system as well. In addition, it is known that the Game of Life is Turing-complete.

Your task is to build a cellular automaton using the rules of Conway's game of life that will allow for the playing of a game of Tetris.

Your program will receive input by manually changing the state of the automaton at a specific generation to represent an interrupt (e.g. moving a piece left or right, dropping it, rotating it, or randomly generating a new piece to place onto the grid), counting a specific number of generations as waiting time, and displaying the result somewhere on the automaton. The displayed result must visibly resemble an actual Tetris grid.



[QFT team 2021]

Tetris in Life

[QFT team 2021]

- Use metapixel to simulate “wired” variant of Life
- Build logic gates in VarLife
- Build a RISC processor from logic gates
- Define assembly language QFTASM
- Define higher-level language Cogol
- Write Cogol-to-QFTASM compiler
- Implement Tetris in Cogol

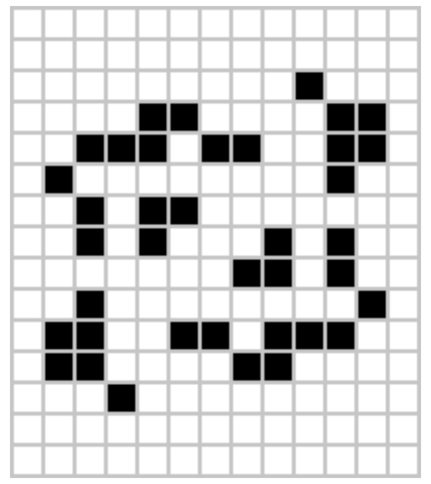
> **A data structure is just a stupid programming language.**
>
> — *Bill Gosper*
>
> **A programming language is a very dangerous data structure.**
>
> — *Guy Steele*

➔ 2,940,928 × 10,295,296 bounding box, >29 billion live cells!

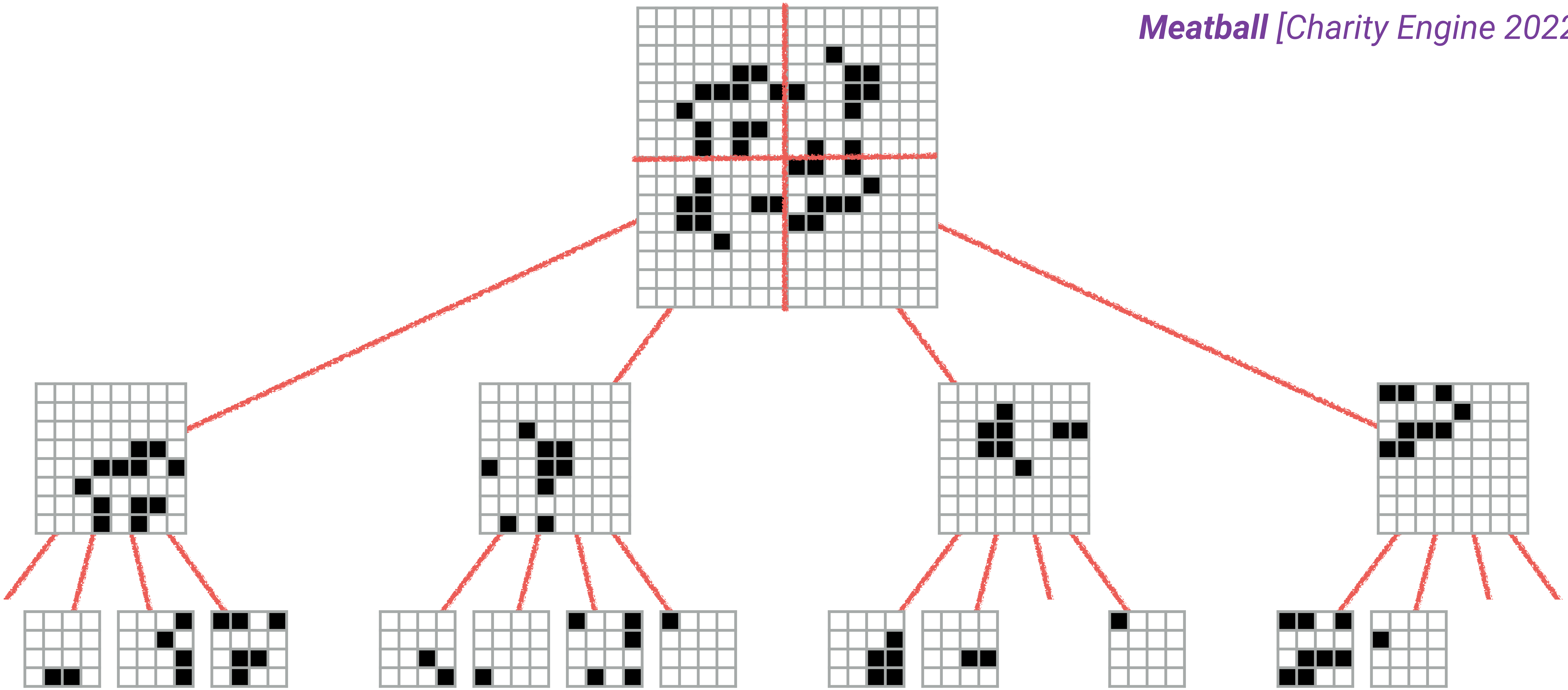
Data Structures

- 2D array large enough to contain every live cell — *Adjacency matrix*
  - O(*HW*) space, O(*HW*) time per generation
- List of live cells — *Edge list*
  - O(*n*) space, O(*n*) time per generation (via hashing, ugly)
- Sorted list of live rows, sorted list of live cells in each live row — *Adjacency list*
  - O(*n*) space, O(*n*) time per generation (ugly)
- Quadtree!!

Quadtree



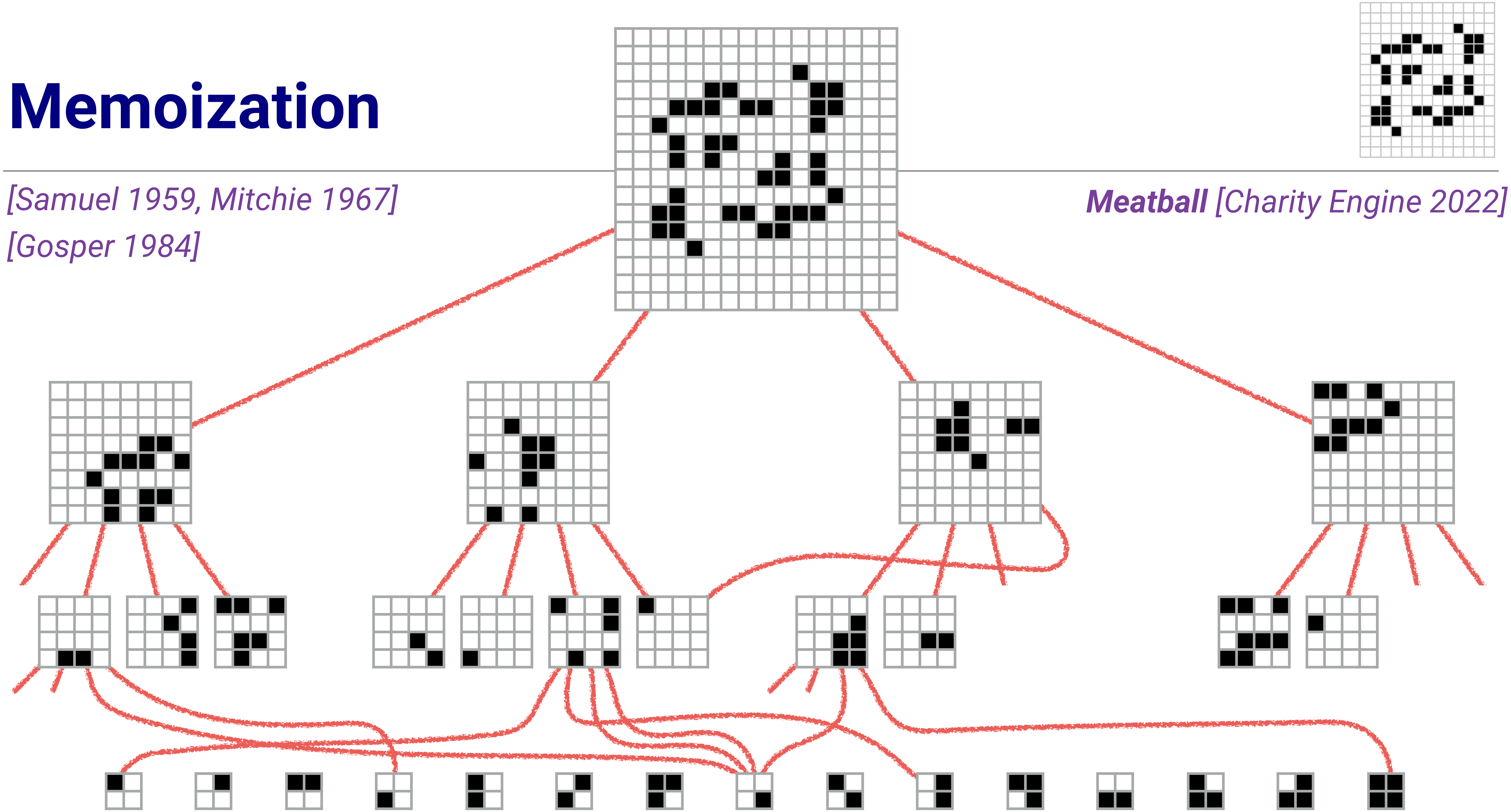
Meatball *[Charity Engine 2022]*



Memoization

[Samuel 1959, Mitchie 1967]
[Gosper 1984]

Meatball *[Charity Engine 2022]*

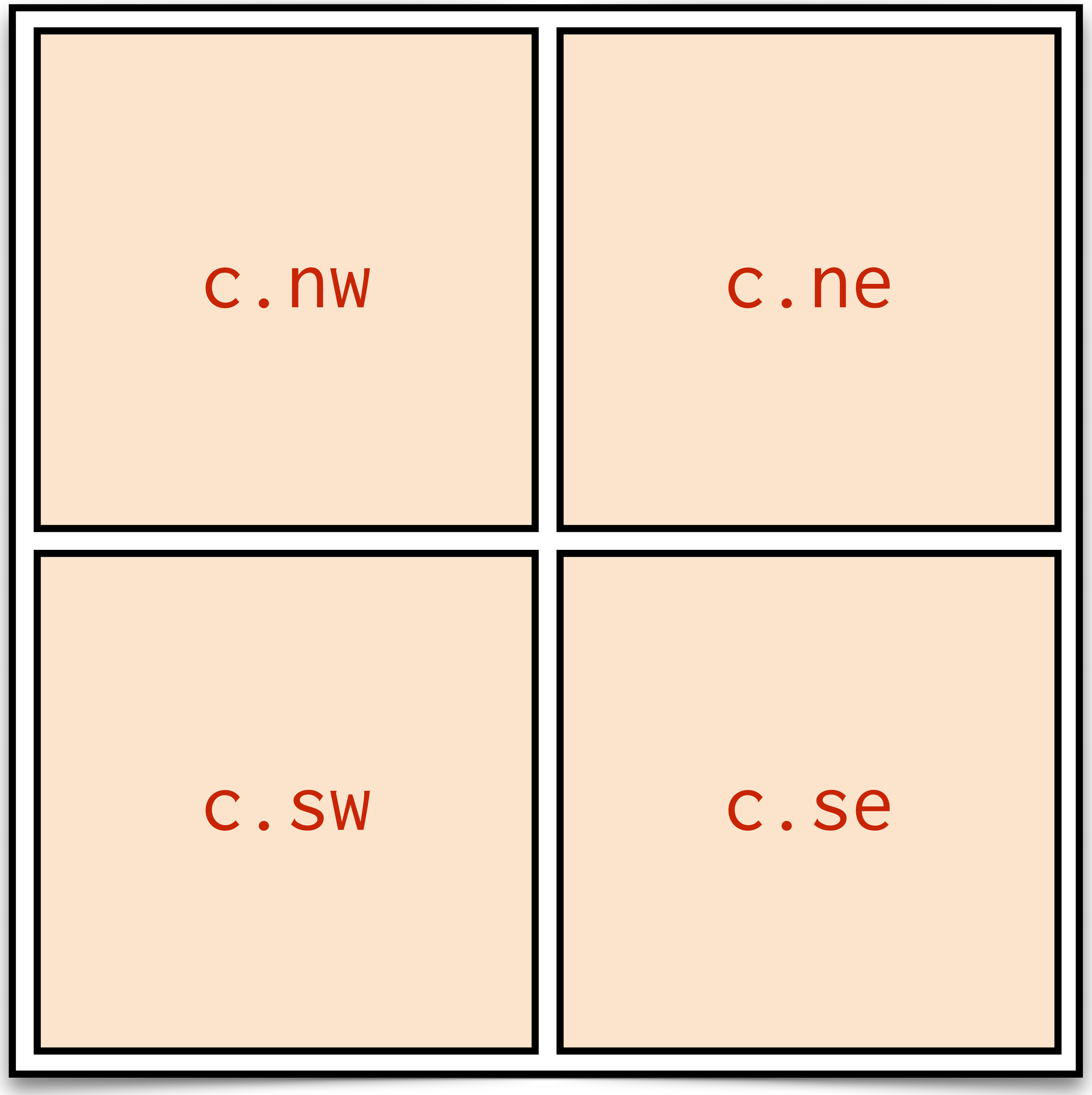


Macro-cells

[Gosper 1984]

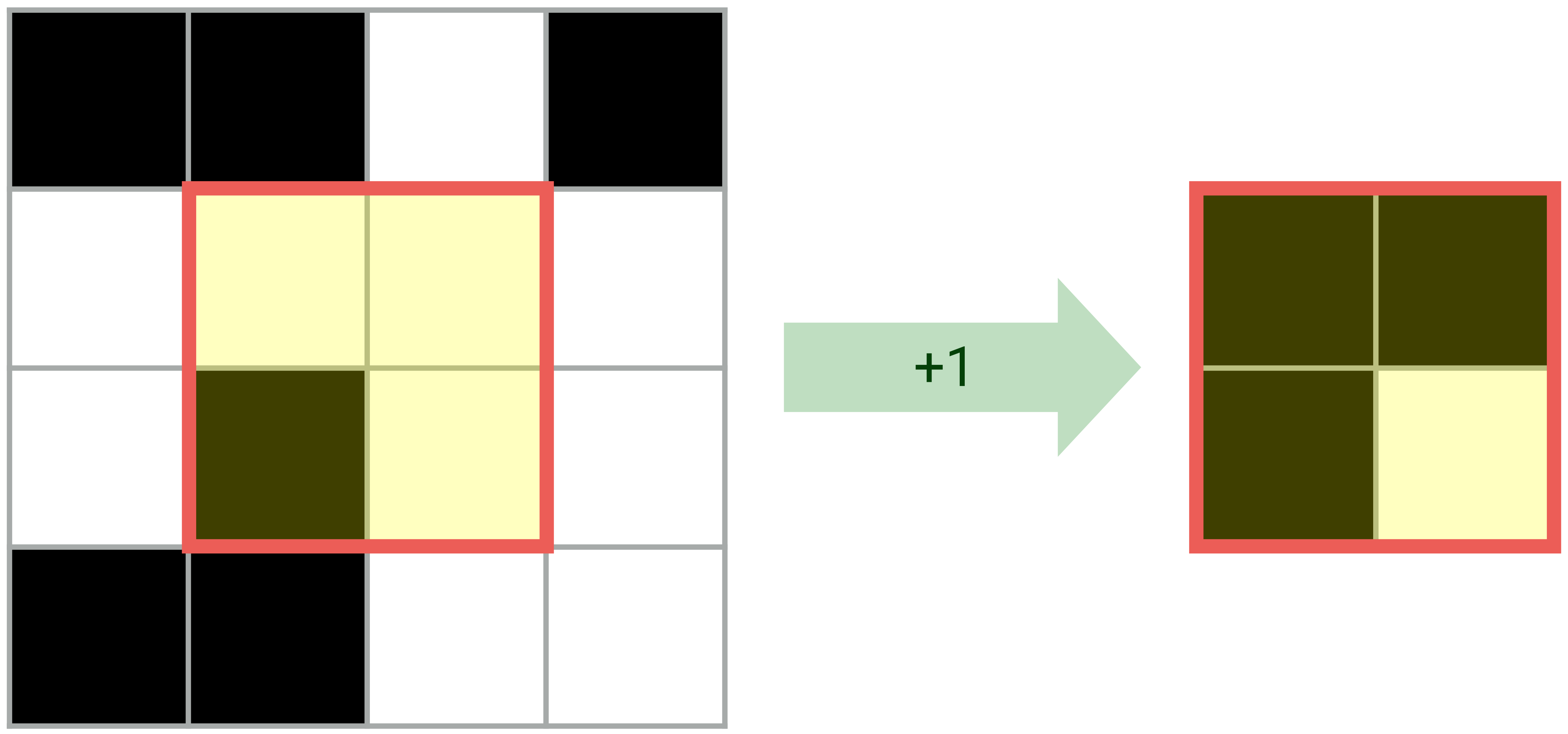
- Order-k macro-cell represents a $2^k \times 2^k$ square of cells
- If $k \geq 3$, store "pointers" to four order-$(k-1)$ macro-cells = children
- *Use hashing to avoid duplication*

```
c.hash = Hash(c.nw.hash, c.ne.hash,
              c.sw.hash, c.se.hash)
```



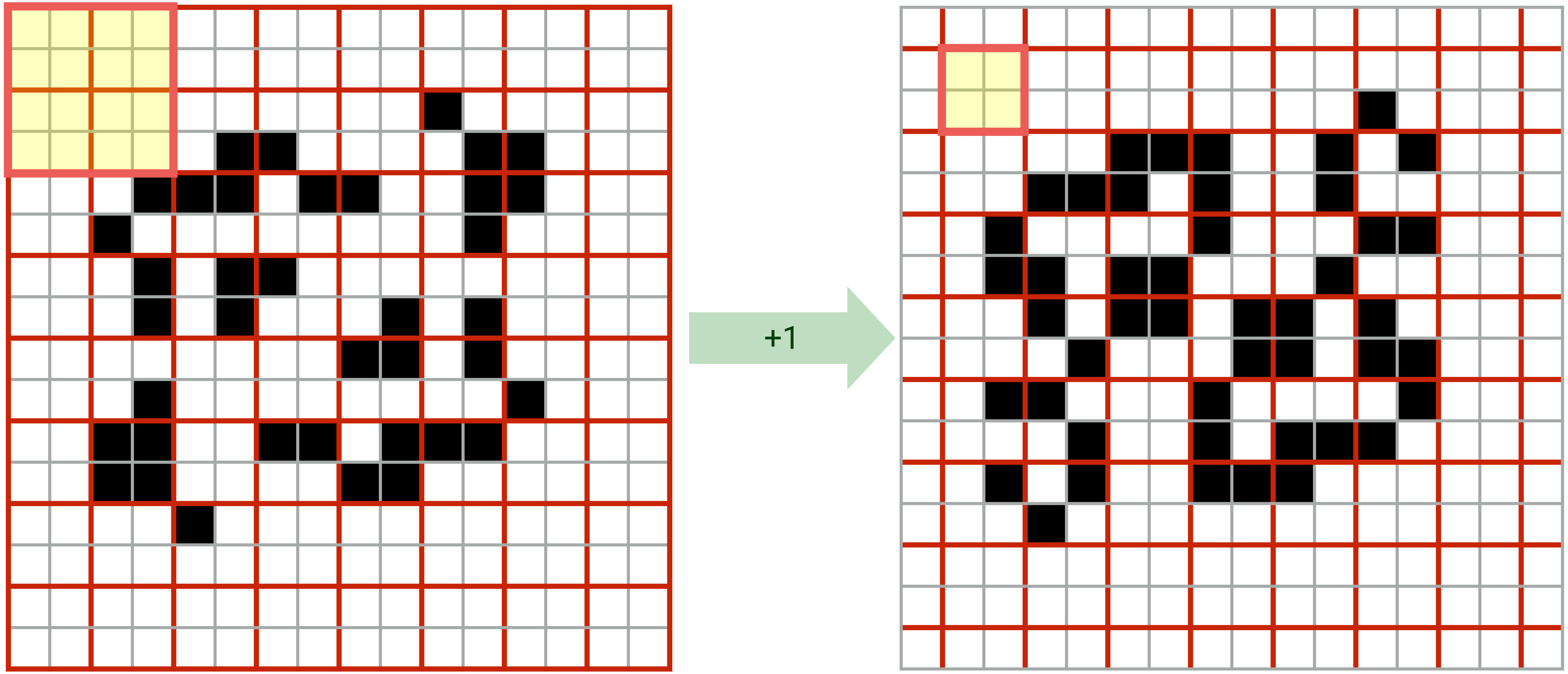
One generation

- 4×4 block at time *T* determines central 2×2 block at time *T*+1
- Store lookup table of all 2^{16} = 65536 possible 4×4 blocks



One generation

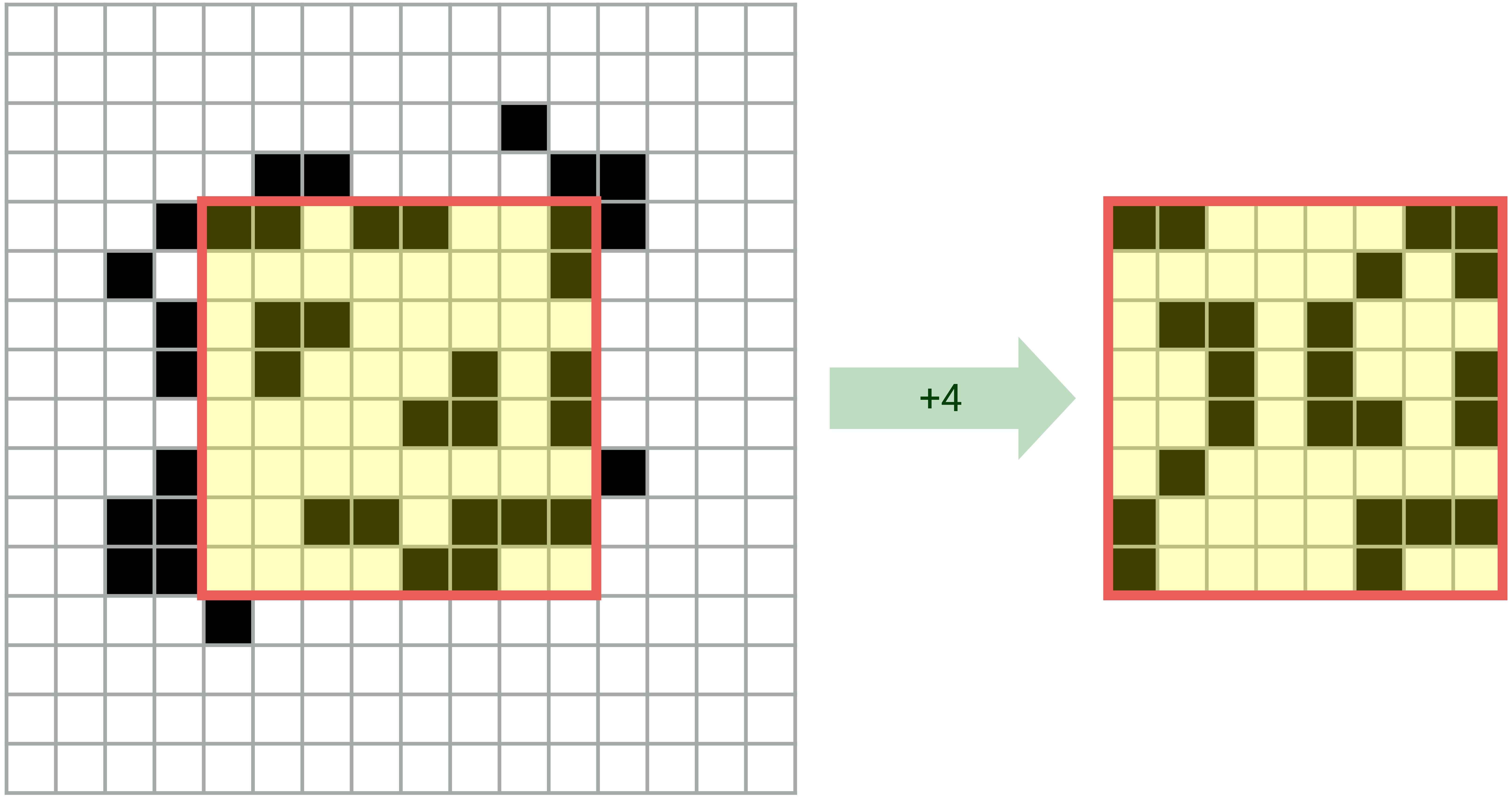
Meatball [Charity Engine 2022]



Macro-cell evolution

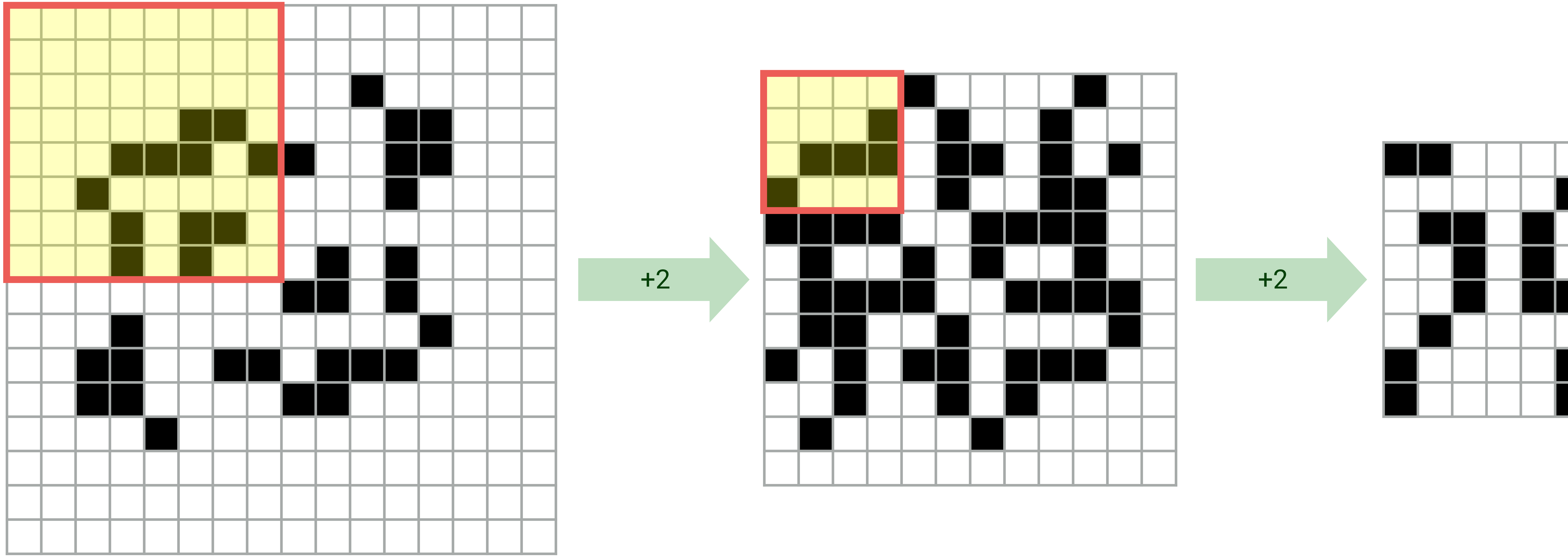
[Gosper 1984]

- Compute central order-(k−1) macrocell, 2^{k-2} steps in the future



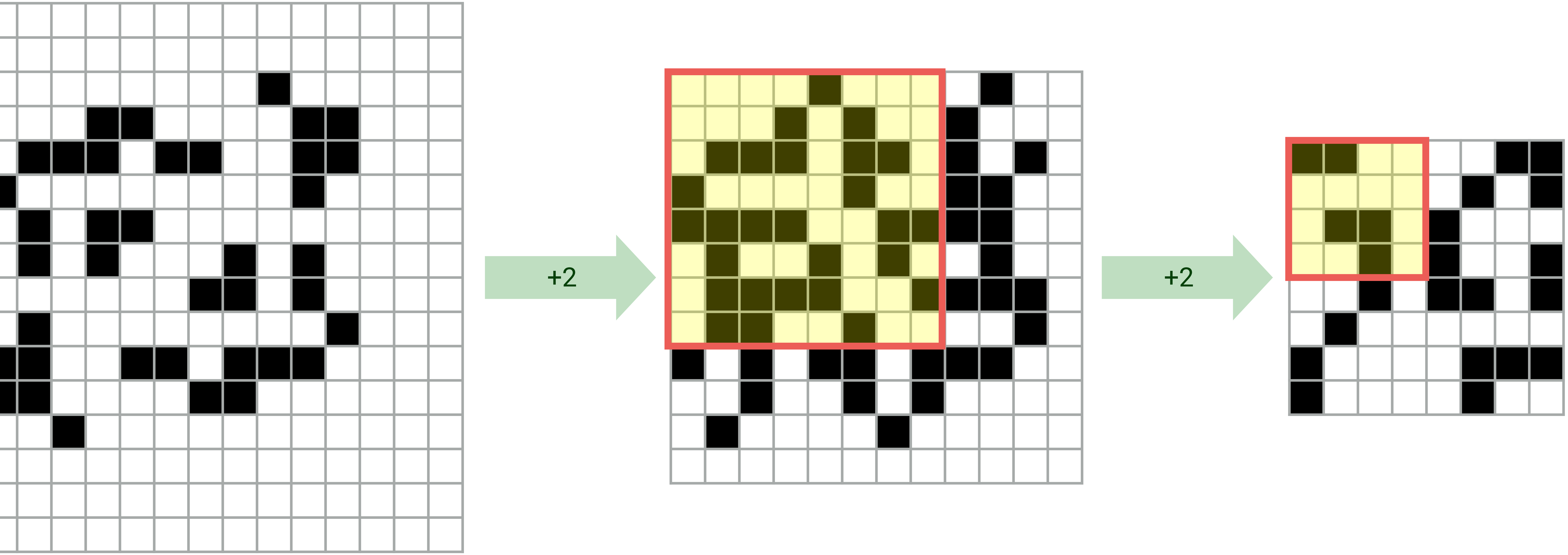
13 recursive calls

[Gosper 1984]



13 recursive calls

[Gosper 1984]

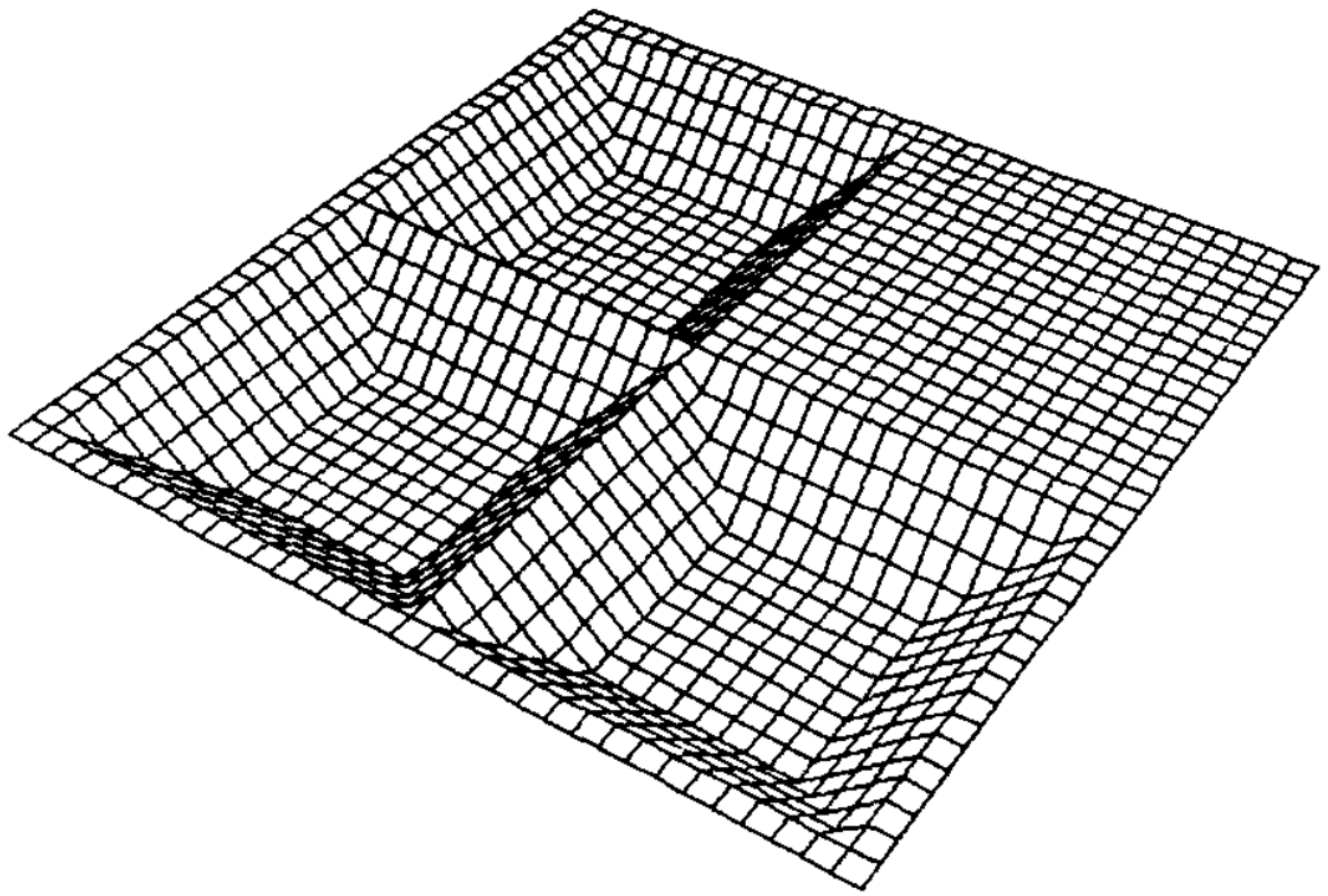


Macro-cell evolution

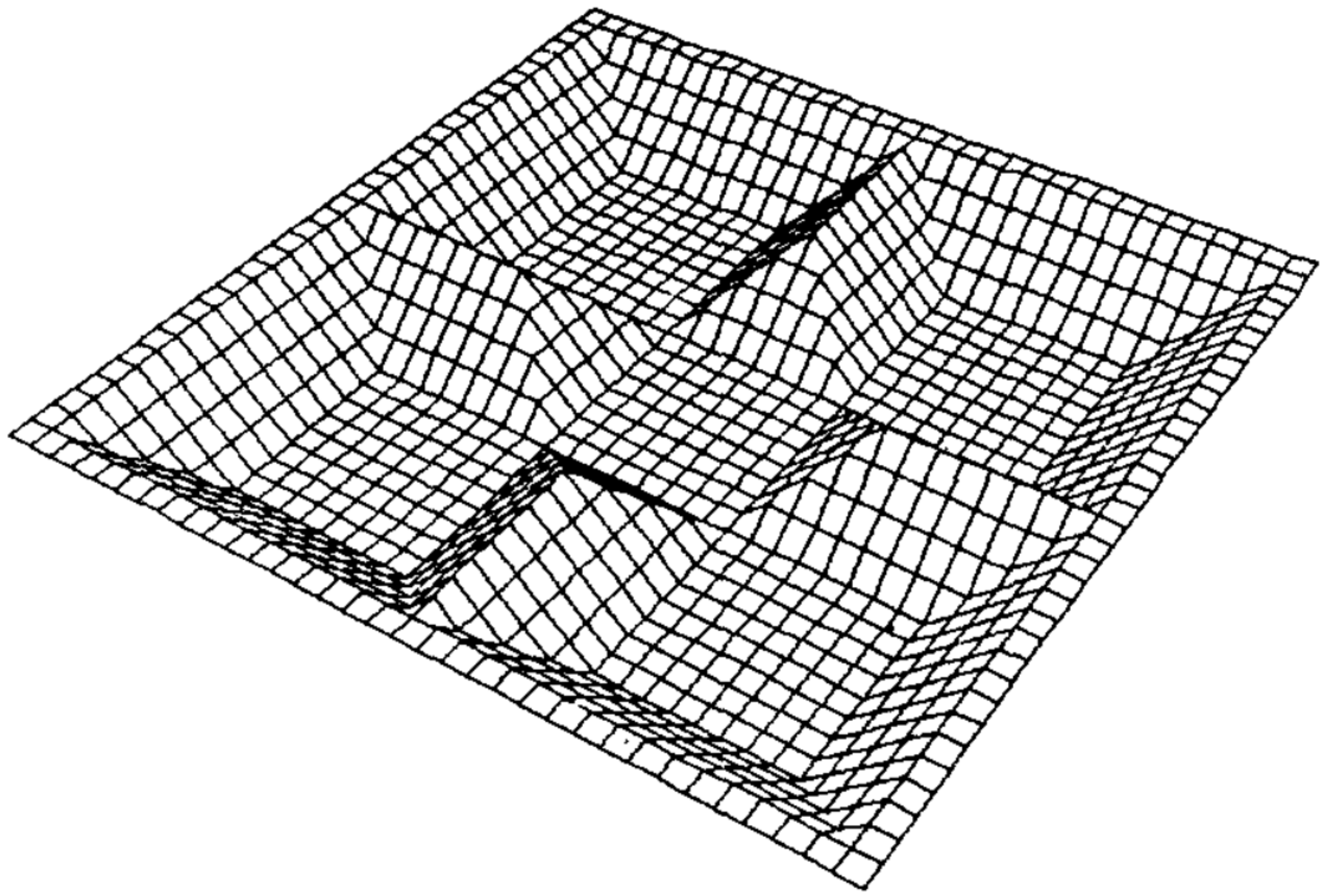
[Samuel 1959, Mitchie 1967]
[Gosper 1984]

- Compute central order-(k–1) macrocell, 2^{k-2} steps in the future
- 13 recursive calls on order-(k–1) macrocells, either children or assembled from grandchildren
  - Naïvely: $T(k) = 13 \cdot T(k-1) + O(1) \Rightarrow T(k) = O(13^k) = O(n^{3.7005})$
- *Use hashing (memoization) to avoid duplication*
  - $\Rightarrow T(k) \leq O(4^k) = O(n^2)$
  - **In practice, *significantly* faster**

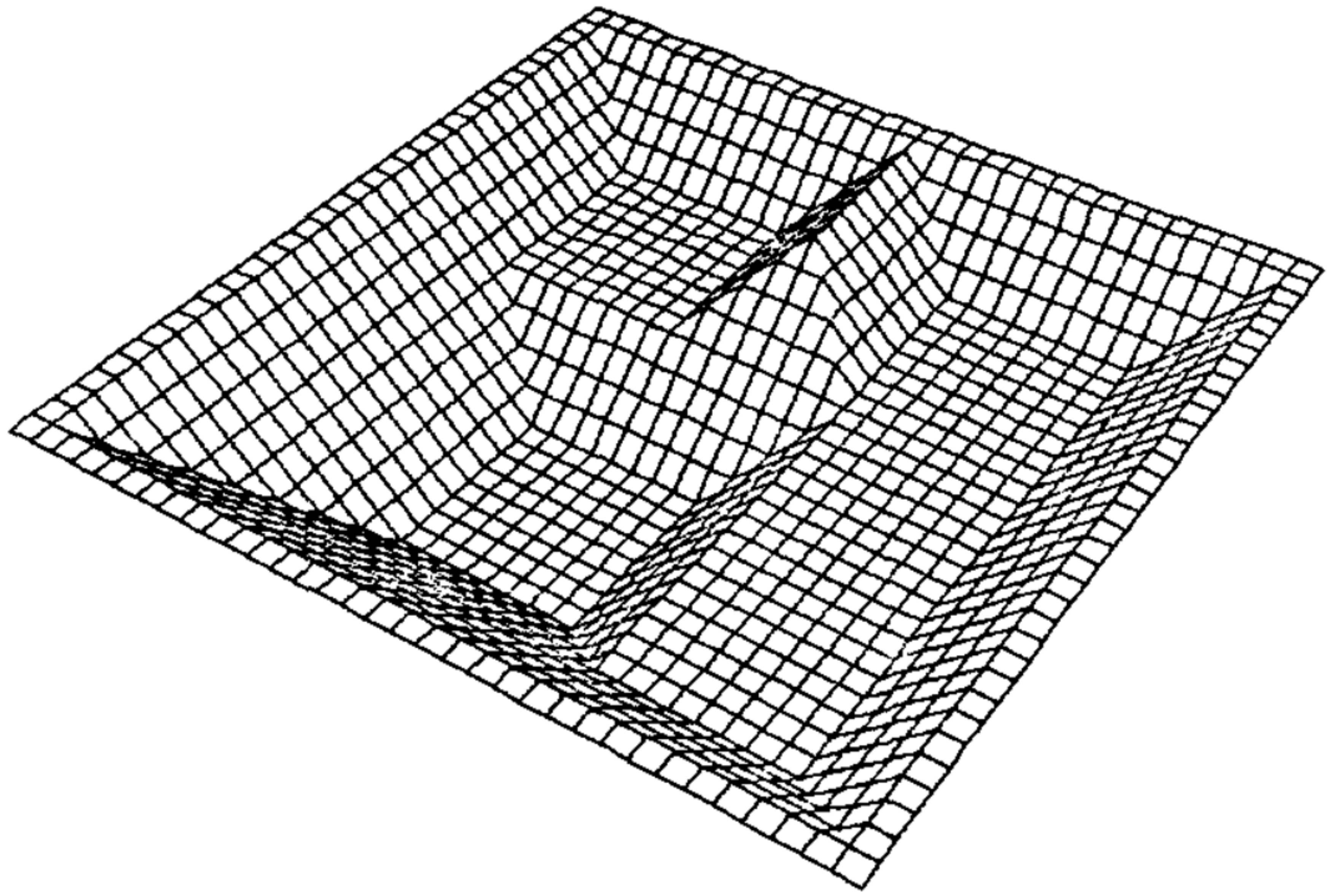
After three scoops



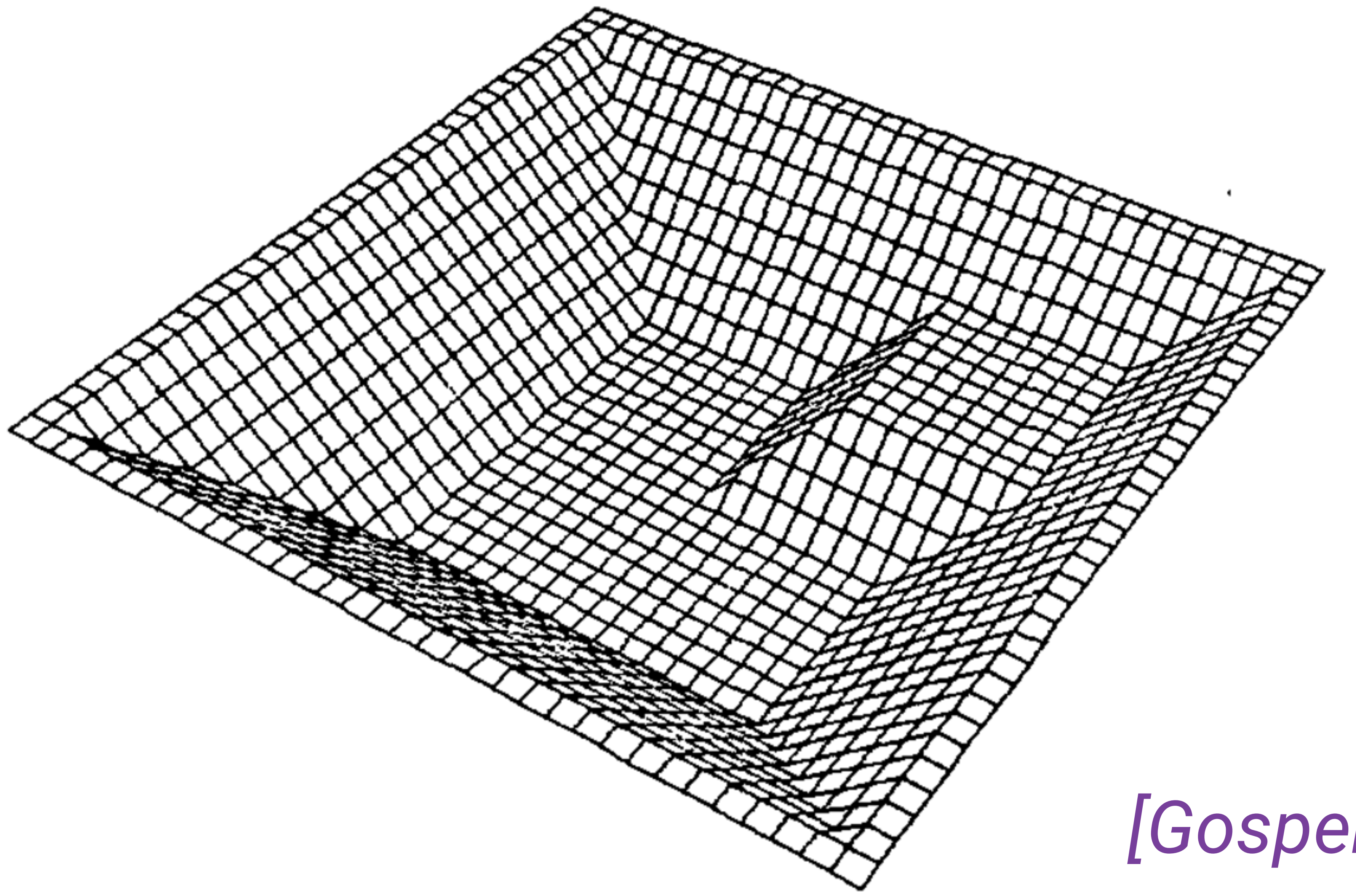
After five scoops



After nine unmethodical scoops



Twelve down, one to go



[Gosper 1984]

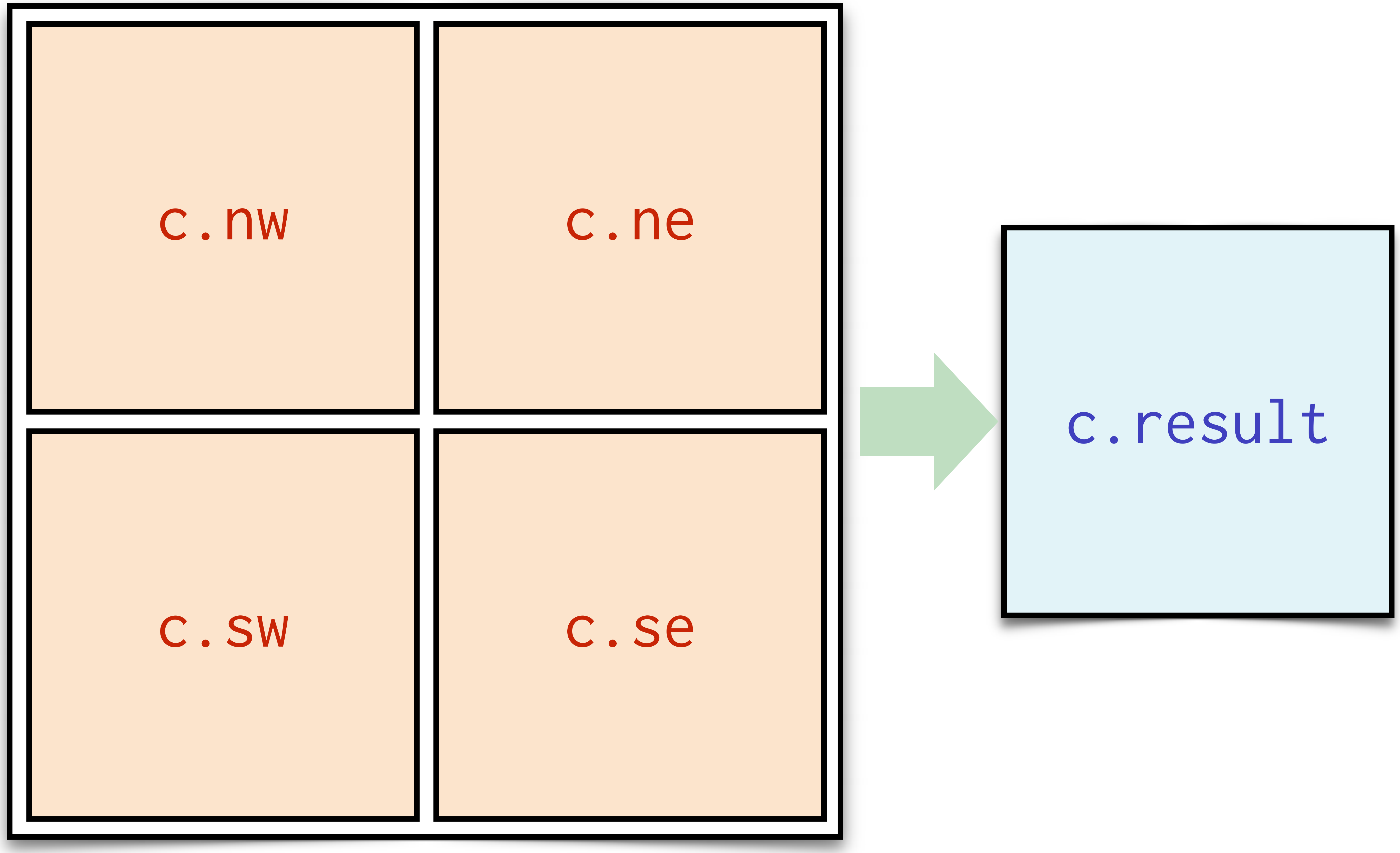
Thus, proliferations of cells is limited by indistinguishability when they are small, and by infrequency of creation when they are large.

[Gosper 1984]

Gosper's "stupid programming language"

[Gosper 1984]

- Store "pointers" to *five* order-(k–1) macro-cells:
  - nw child
  - ne child
  - sw child
  - sw child
  - *evolution result*
- *Use hashing to avoid duplication*



Summary

- Intuitively, the running time of level-k Hashlife is proportional to the "amount of novelty" in the next 2^{k-2} generations.
  - Can this intuition be formalized?

- ***Bottleneck:*** Translations of the same pattern by a non-power-of-2 are distinct.
  - Can HashLife be improved to avoid this bottleneck?
  - Is HashLife "optimal" in any formal sense?

Thank you!

3. Conclusion

Even in simulating such an unpredictable and irreversible automaton as Life, considerable economies are possible. By attributing similar thriftiness to whatever implements our own reality, our (simulated) imaginations may be stimulated.