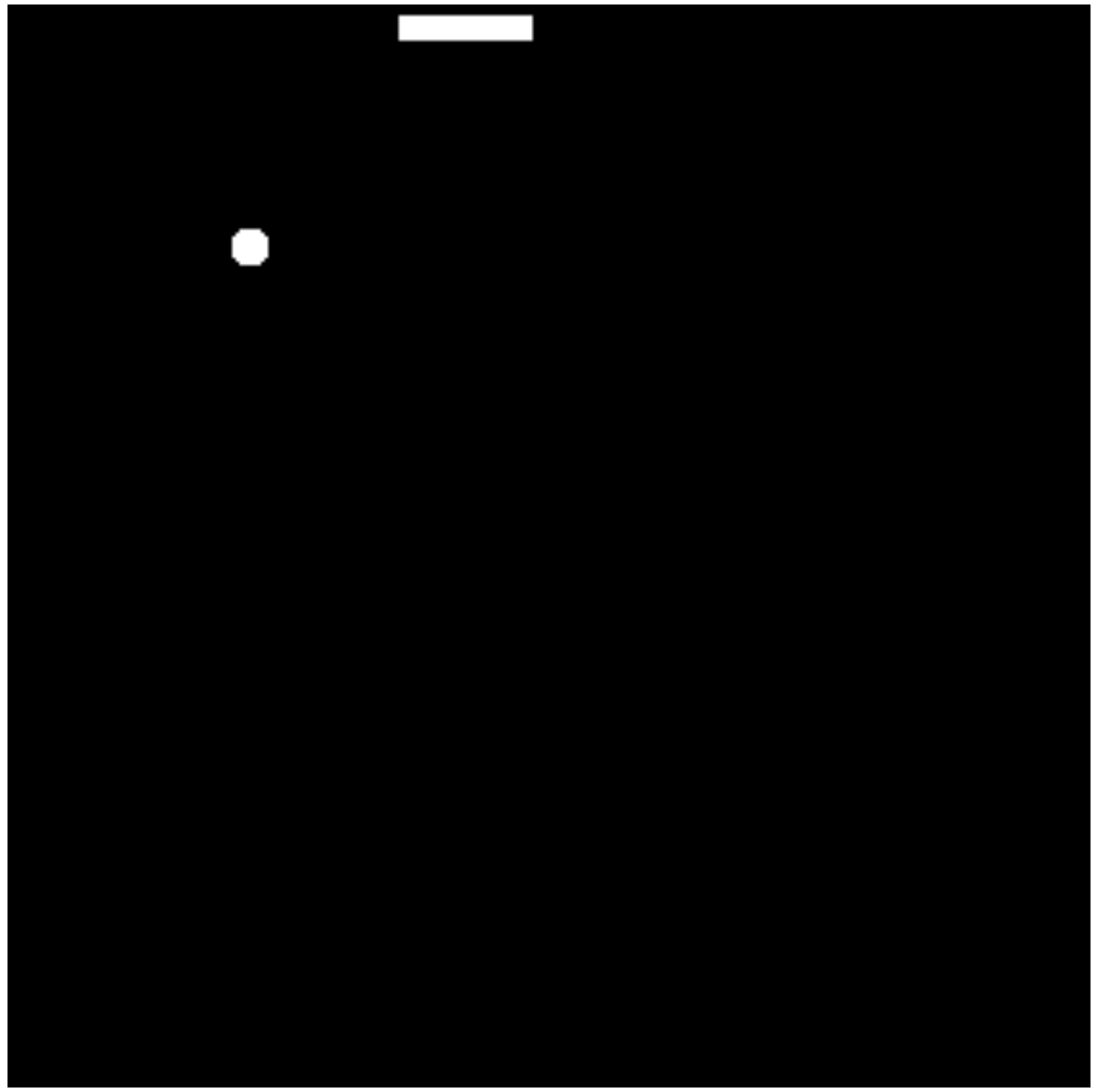
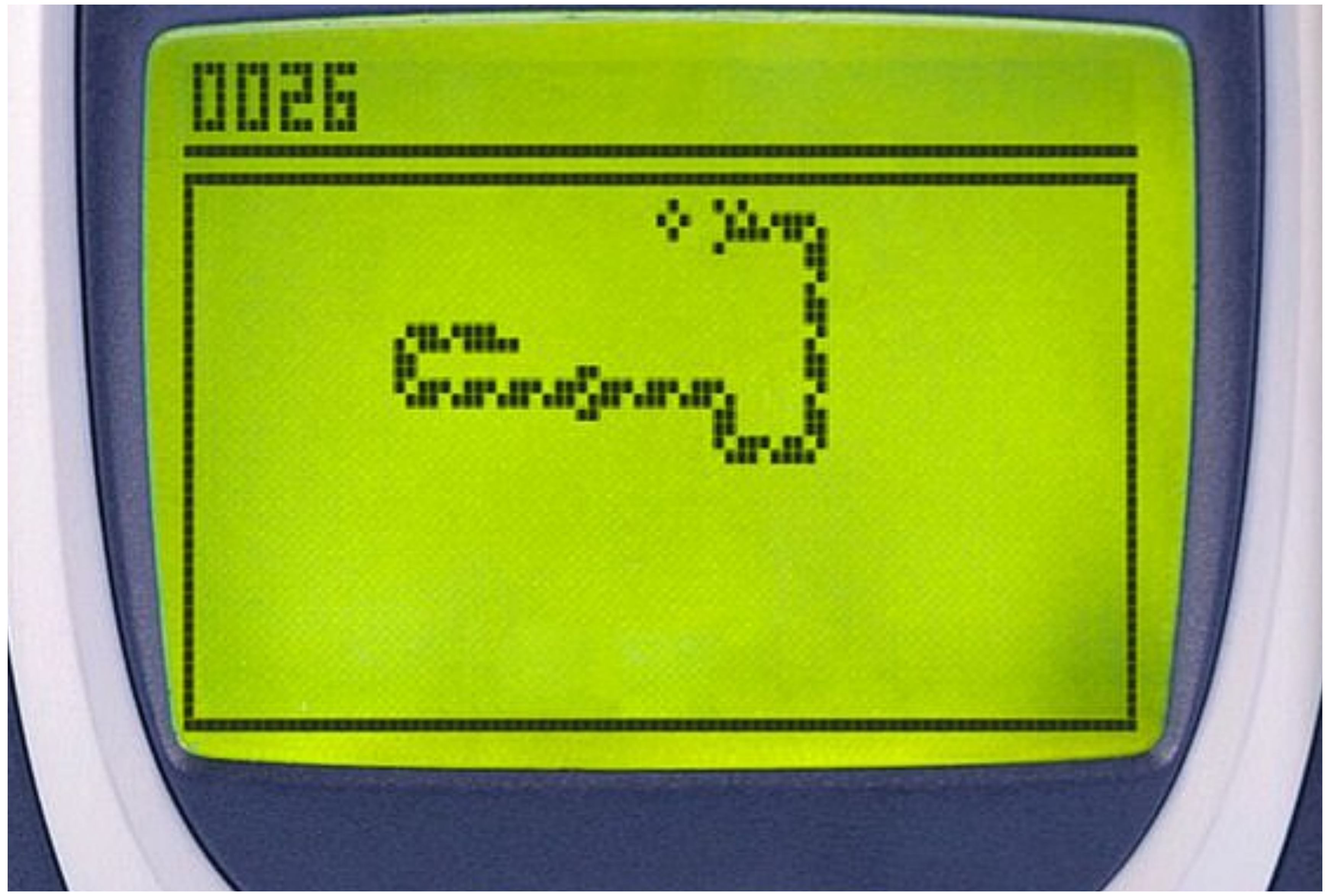
# Snake in Optimal Space and Time

**Philip Bille**
Martín Farach-Colton
Ivor van der Hoog
Inge Li Gørtz

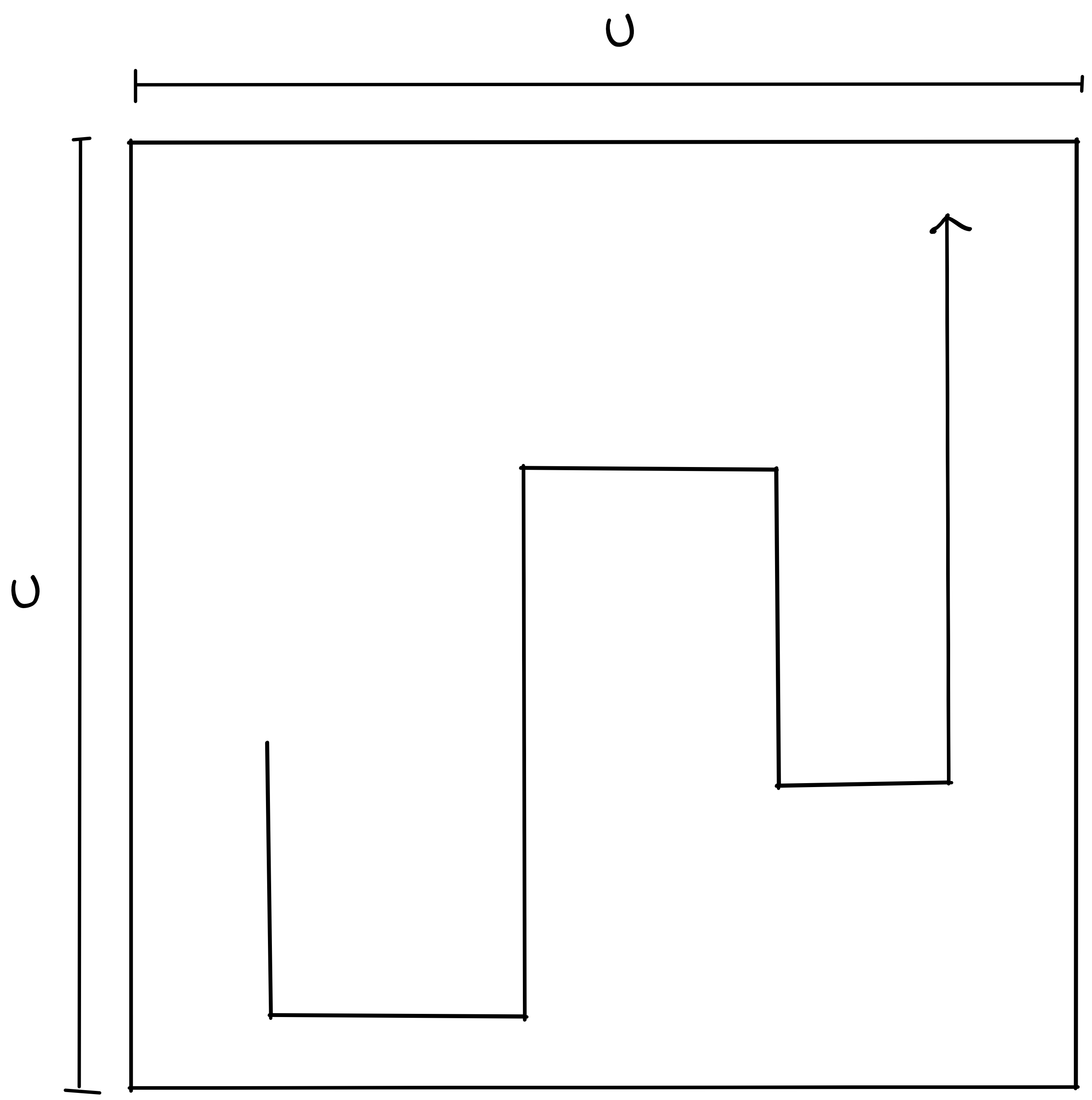
# Snake





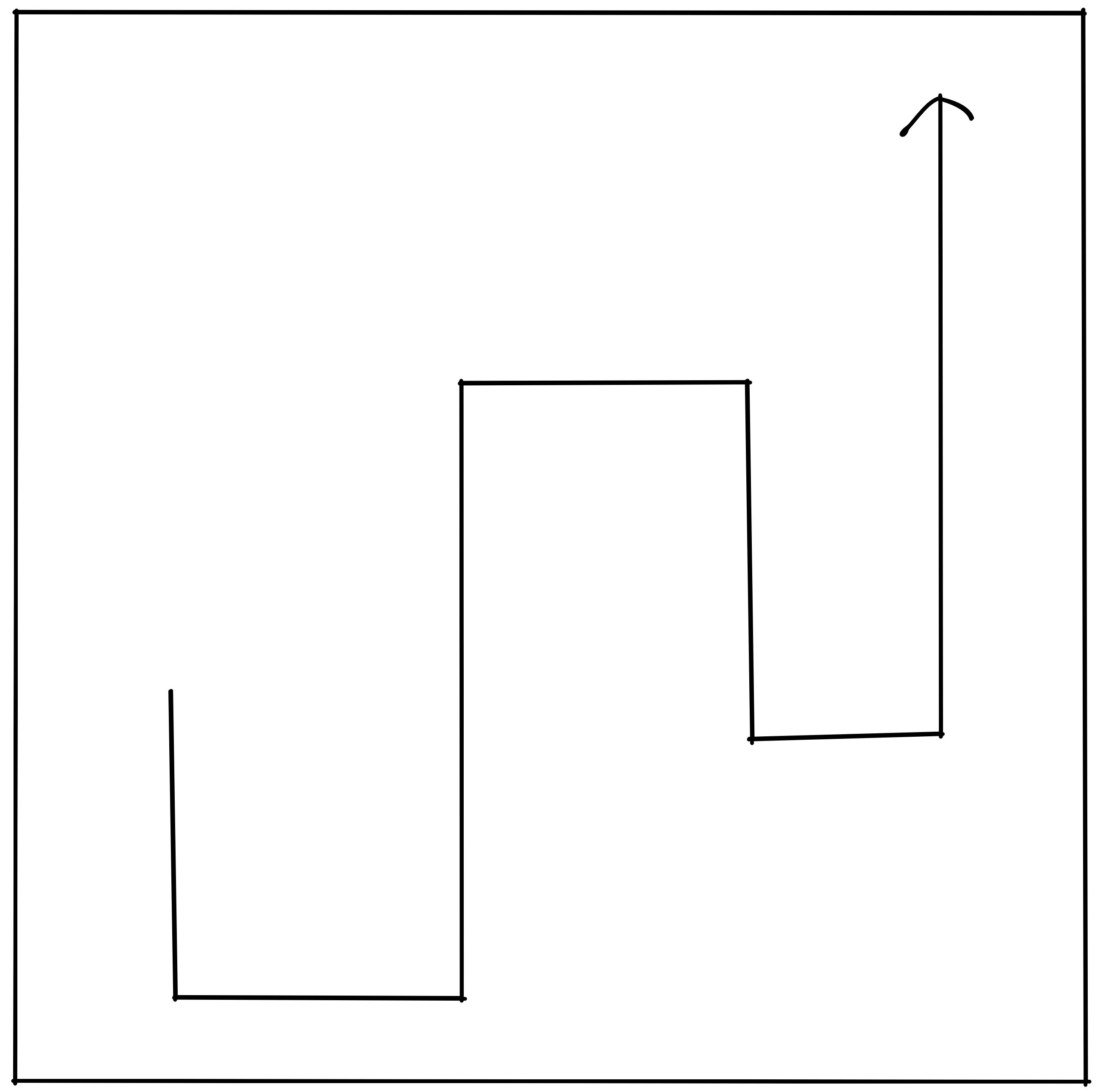

- Snake.
  - Control the head of the snake. Can change direction.
  - Avoid collision with yourself and the boundary (and maybe obstacles).
  - Snake extends when it eats an item.
- Question: how many bits do we need to represent the snake and update it in constant time?

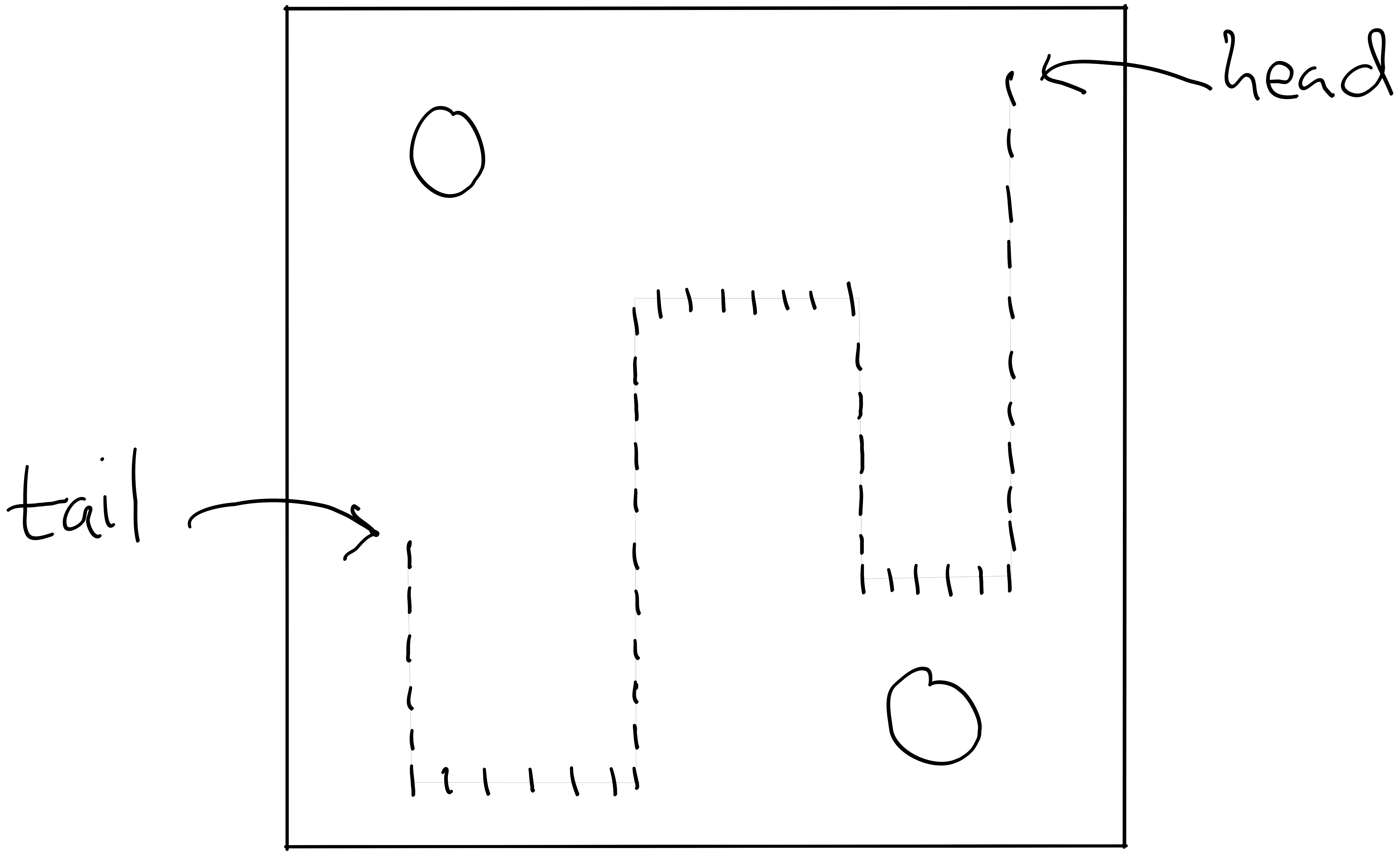
# Snake




- Snake problem. Maintain snake S of length n on a u by u grid subject to the operations:
  - EXTEND(d): add new point to S adjacent to the head in direction d (up, down, left, right). If collision with itself or boundary terminate and report collision.
  - REDUCE(): remove the tail from S.
- Goal. O(n + log u) bits and O(1) time operations.

# Snake

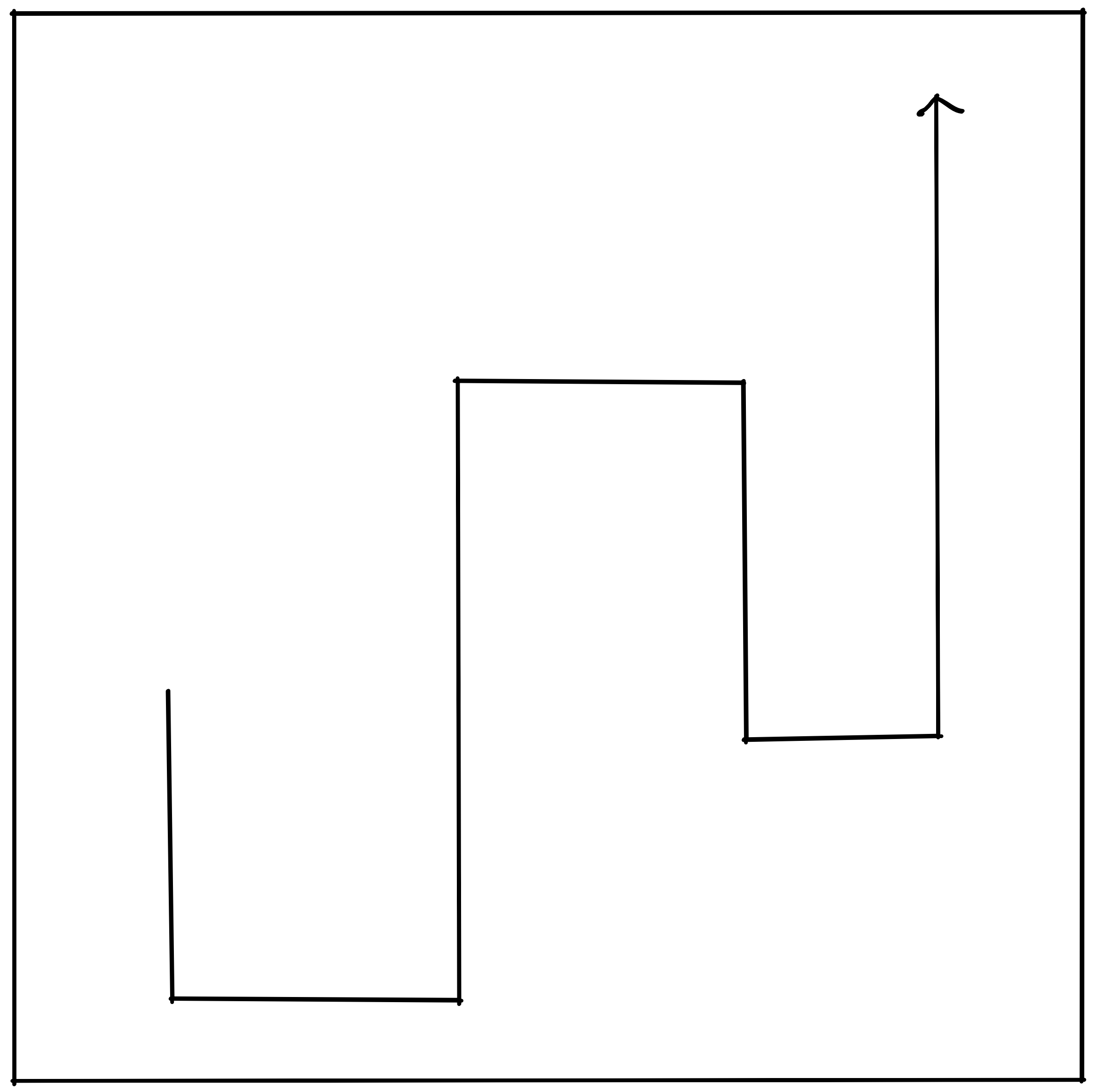


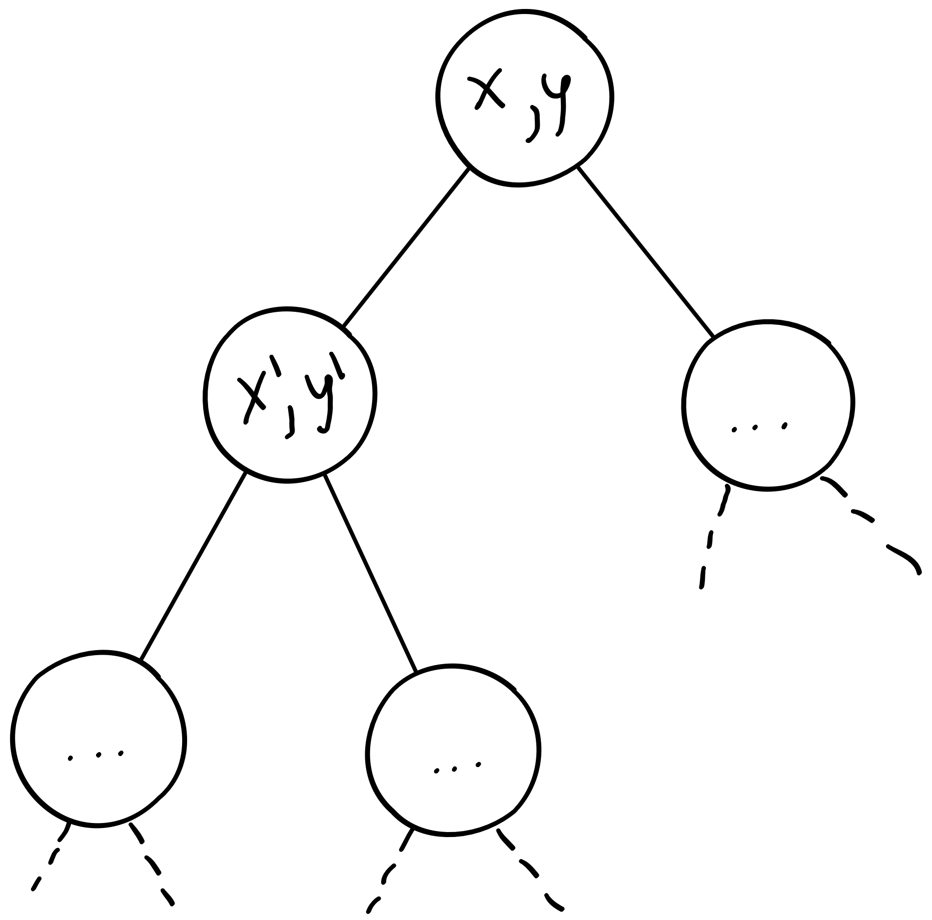



- Bit matrix solution.
  - Maintain complete bit matrix representing snake + head and tail pointers.

- ⇒ $O(u^2)$ bits of space and $O(1)$ time EXTEND and REDUCE.

# Snake





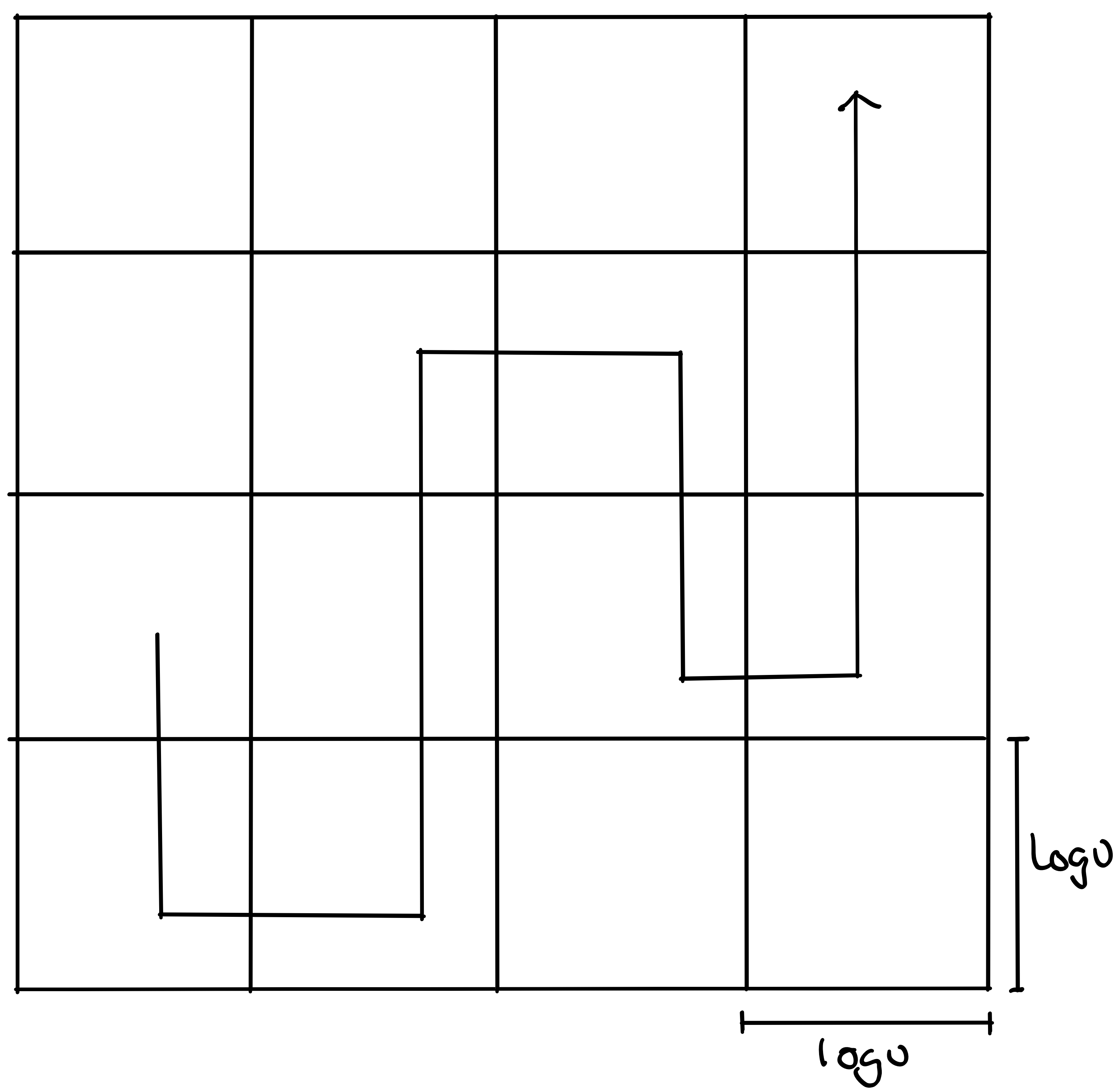

- Dictionary solution.
  - Maintain a dictionary (eg. search tree) on the position of the snake.

- ⇒ O(n log u) bits of space and O(log n) time EXTEND and REDUCE.
- (with faster dictionaries ⇒ O(1) randomized time or $O\left(\sqrt{\log n / \log\log n}\right)$ deterministic time .

# Snake

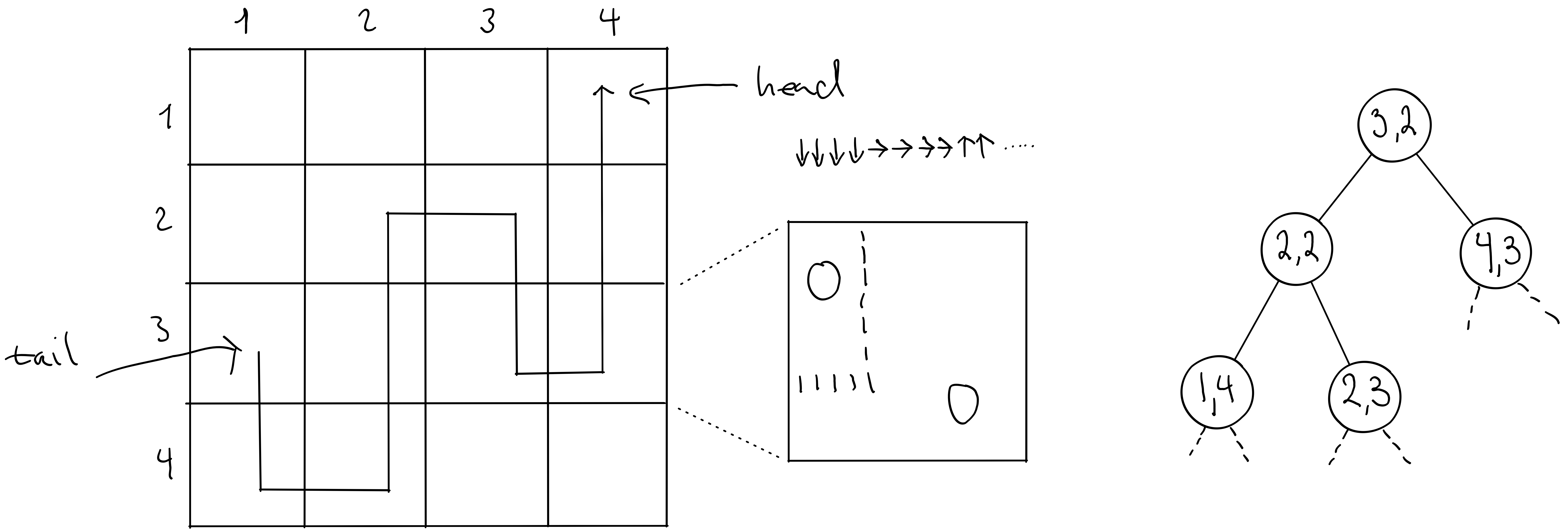
| | Space | Time |
|---|---|---|
| Bit matrix | $O(u^2)$ | $O(1)$ |
| Dictionary | $O(n \log u)$ | $O(\log n)$ |
| New | $O(n + \log u)$ | $O(1)$ |

# Simple Snake



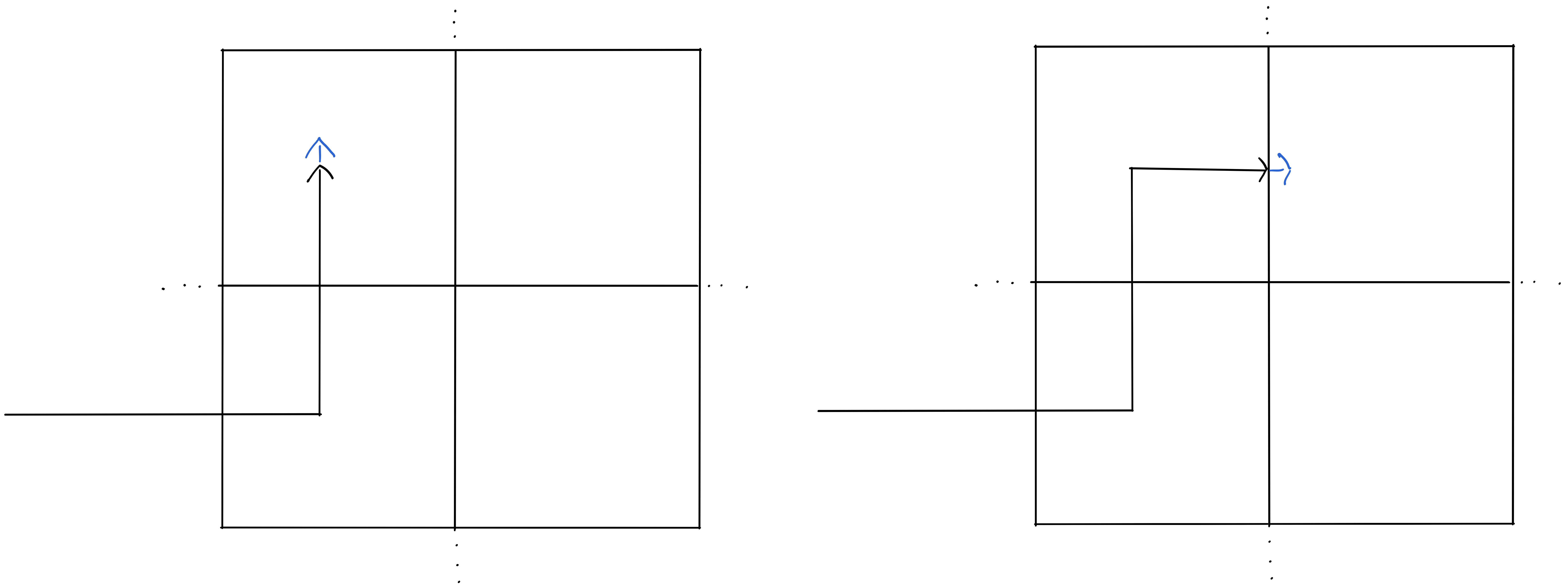

- Tiling.
  - Partitioning grid into tiles of log u by log u.
  - Key property: The number of non-empty tiles is $\leq 4\frac{n}{\log u}$.

# Simple Snake




- Data structure.
  - Direction string and head and tail positions.
  - Search tree on non-empty tiles.
  - Bitmatrix for each non-empty tile.
- ⇒ Space: $O\left(n + \frac{n}{\log u}\log u + \frac{n}{\log u}\log^2 u\right) = O(n \log u)$ bits.

# Simple Snake



- EXTEND(d).
  - Collision check + data structure update.
  - Same tile: $O(1)$ time.
  - New tile: $O(\log n)$ time.
- Tiling property ⇒ $O(1)$ amortized time.
- Deamortize to $O(1)$ worst-case time with buffers + scope.

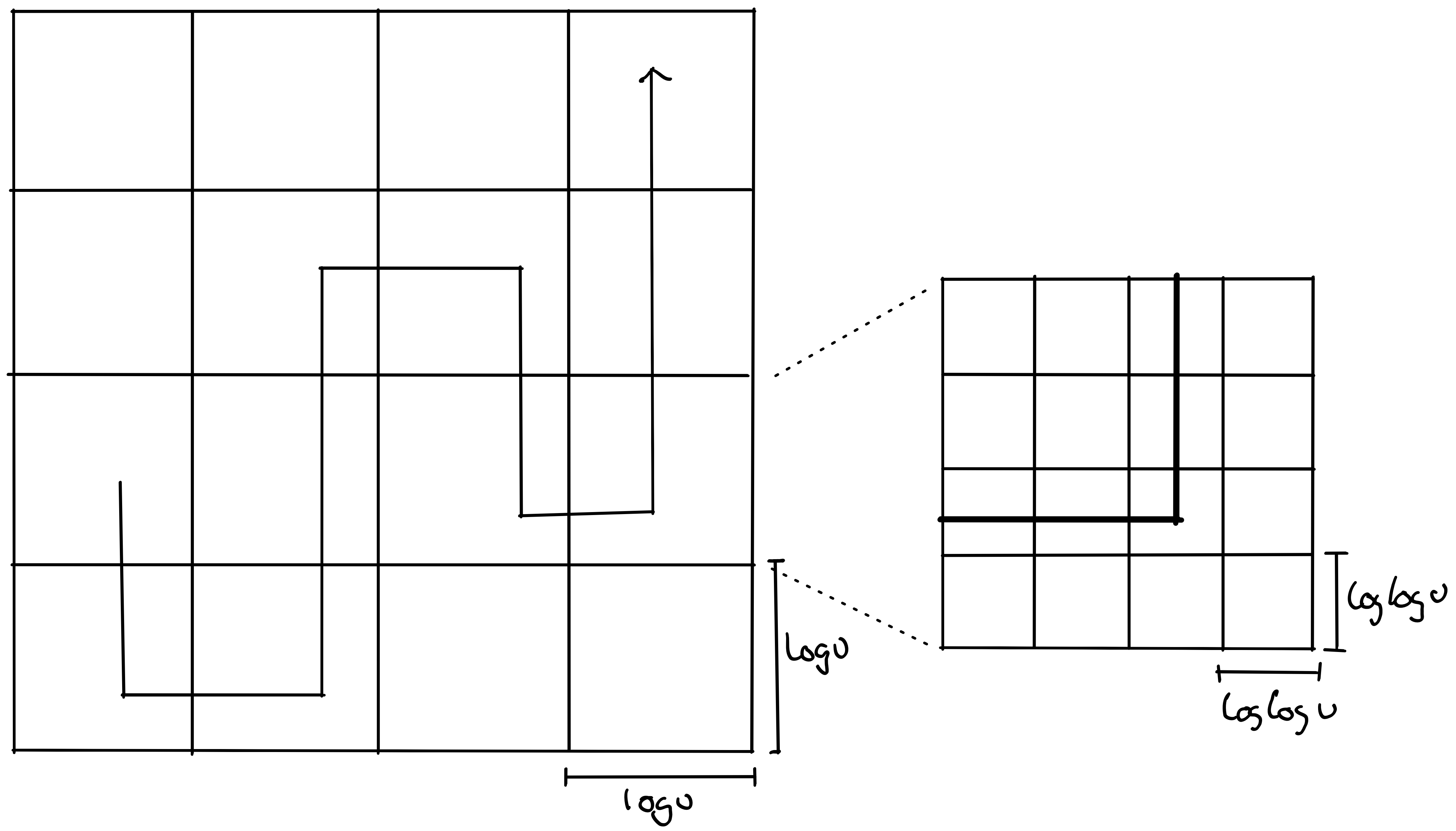
# Simple Snake

- Simple snake
  - REDUCE in O(1) time with similar (but easier) ideas as EXTEND.
  - ⇒ O(n log u) bits and O(1) time for REDUCE and EXTEND.

# Optimal Snake

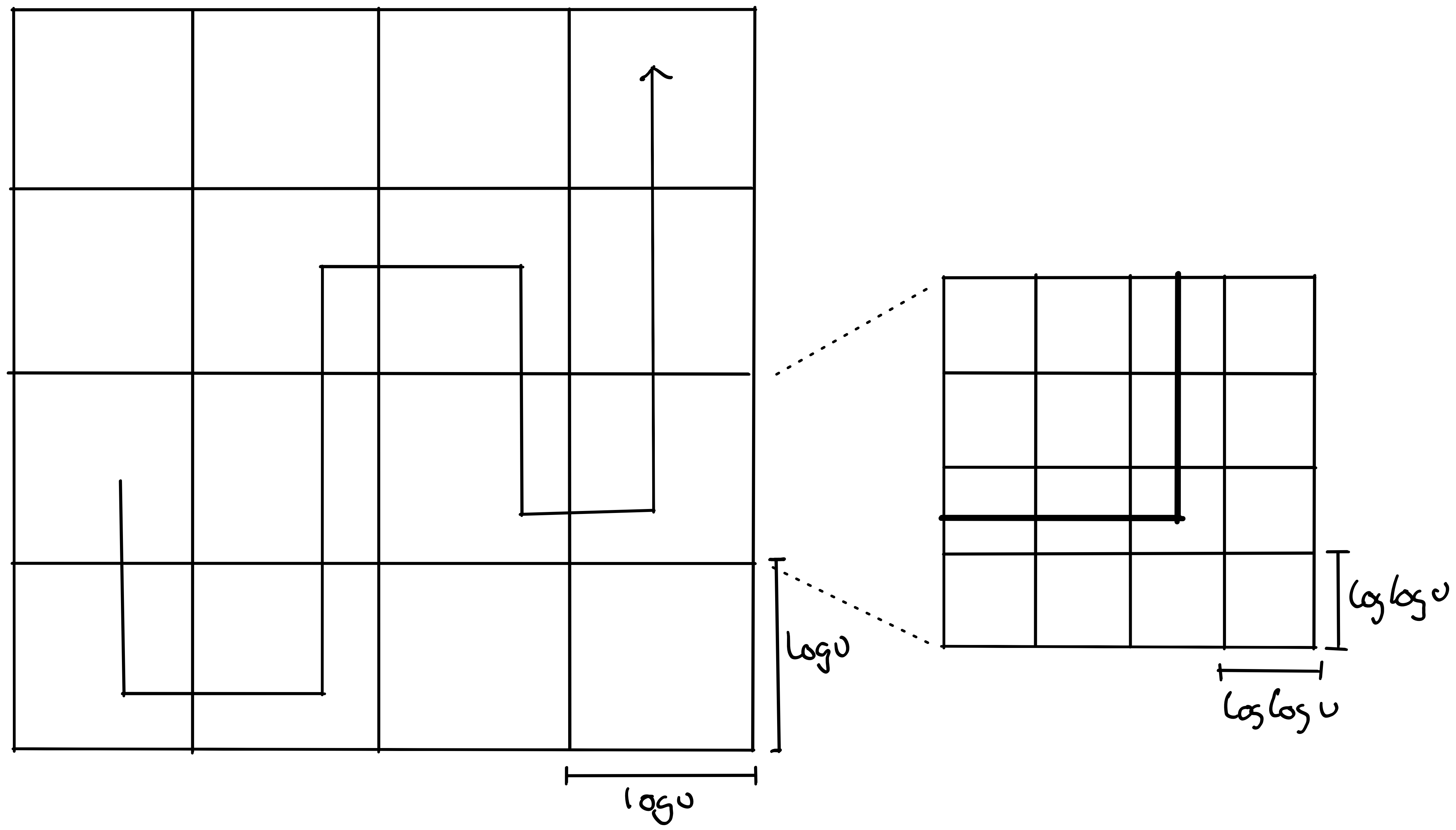
- Overview.
  - 2-level tiling.
  - Compact dynamic allocation scheme.
  - Tabulation.

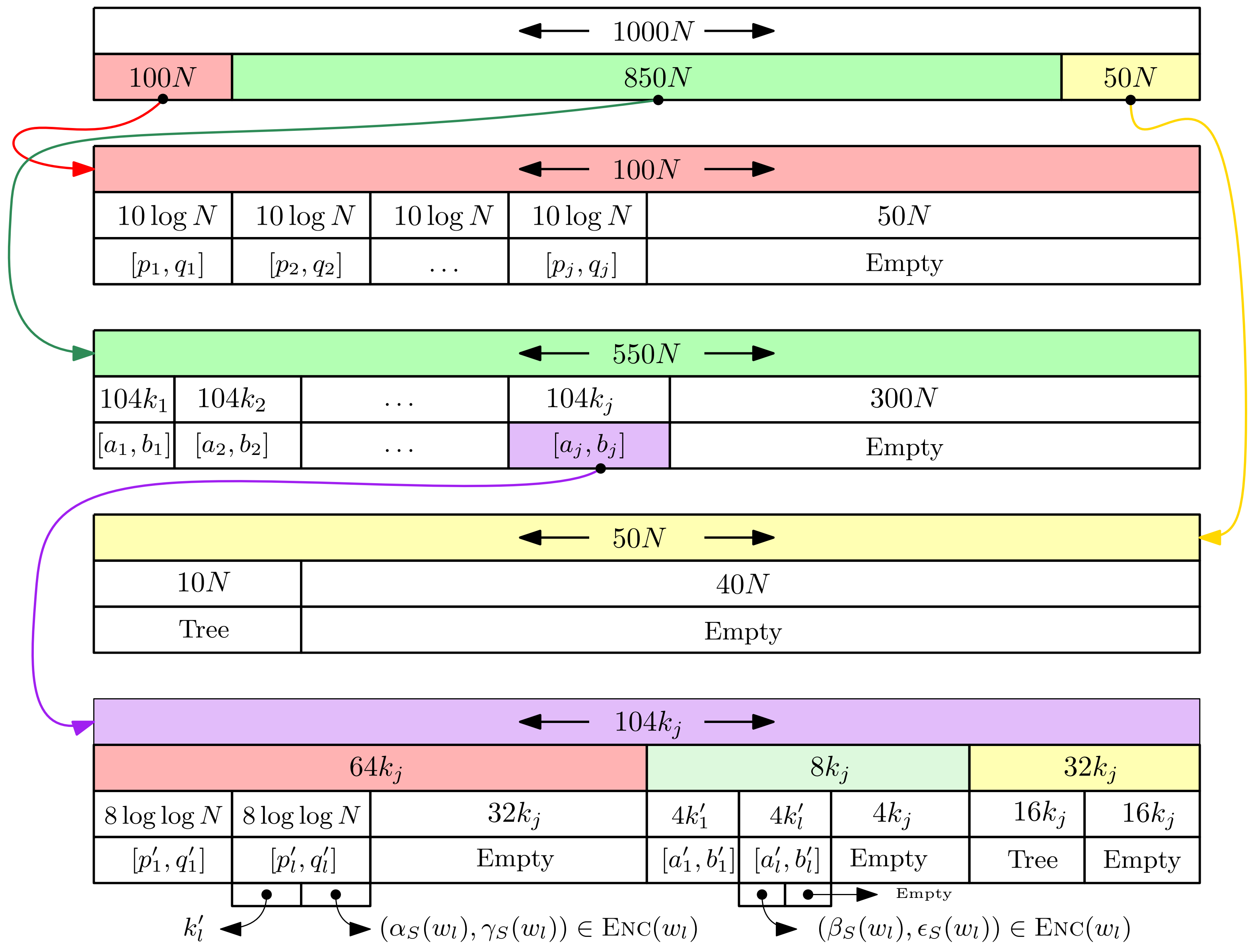
# Optimal Snake




- 2-level tiling.
    - Tiles and micro tiles.

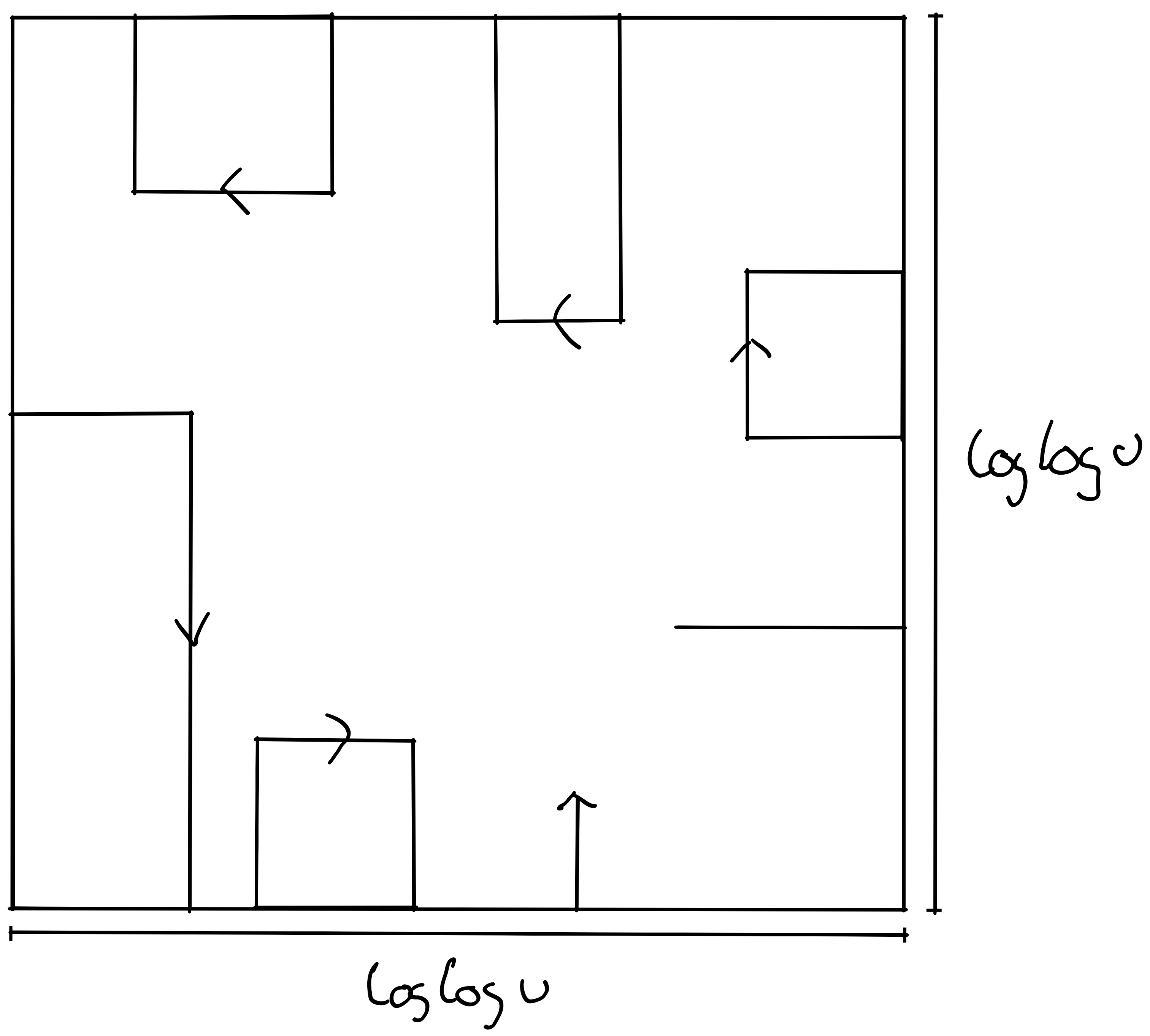
# Optimal Snake




- Dynamic allocation.
  - We have Ω(n/log log u) non-empty micro tile data structures.
  - ⇒ standard log n bit pointers to these require ω(n) bits.
  - Key idea: Compact dynamic layout of data structures to instead use local pointers.

# Optimal Snake




- Collision detection on micro tiles.
  - Encode boundary snake positions in O(log log u) bits.
  - Encode subsnakes in $O((\log \log u)^2)$ bits.
  - Table: encoding + new position -> is there a collision?
  - Table size: $2^{O((\log \log u)^2)} = O(n)$ bits.
- Similar tables for updating the head and tail.

# Snake

- Conclusion. O(n + log u) bits of space and constant time.

- Reviewer 3: “Given that the provided data structure achieves the stated bounds at the cost of large hidden constants, humongous instances are needed in order to appreciate its benefits over a naive implementation. Still, this might prove beneficial for computer scientists owning cellphones with asymptotically large screen sizes... “

- Open problems.
  - First non-trivial data structure result for a classic game? Other interesting games?
  - Interesting ways to include item to pick up or obstacles?