

Building an Open-source DNA Storage Algorithm for a Space Mission

Chayapatr Archiwaranguprok



Blue Origin's New Shepard | NS-13

13 October 2020

KARMAN LINE 100 KM

CAPSULE FREE FLIGHT

SEPARATION

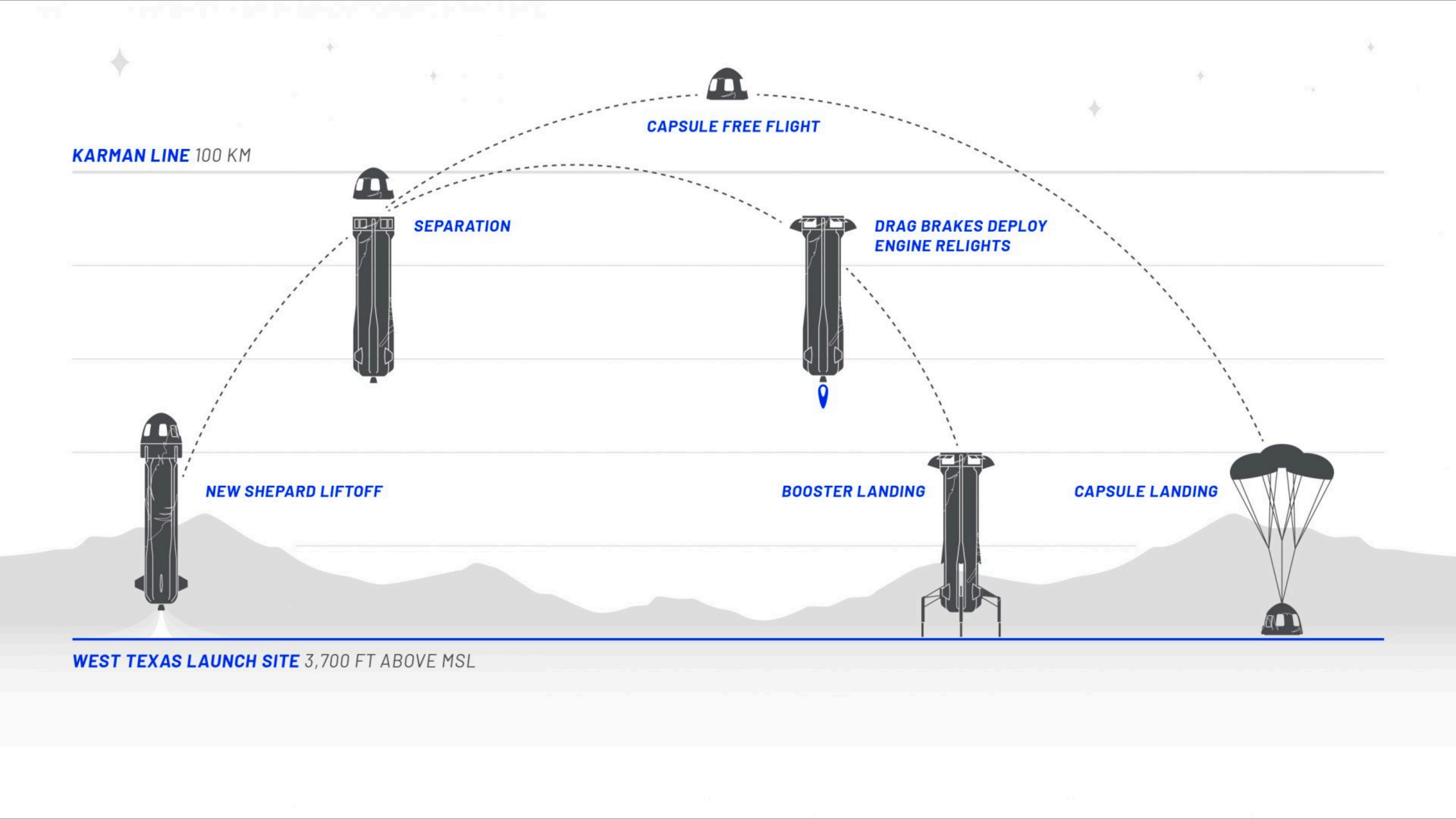
**DRAG BRAKES DEPLOY
ENGINE RELIGHTS**

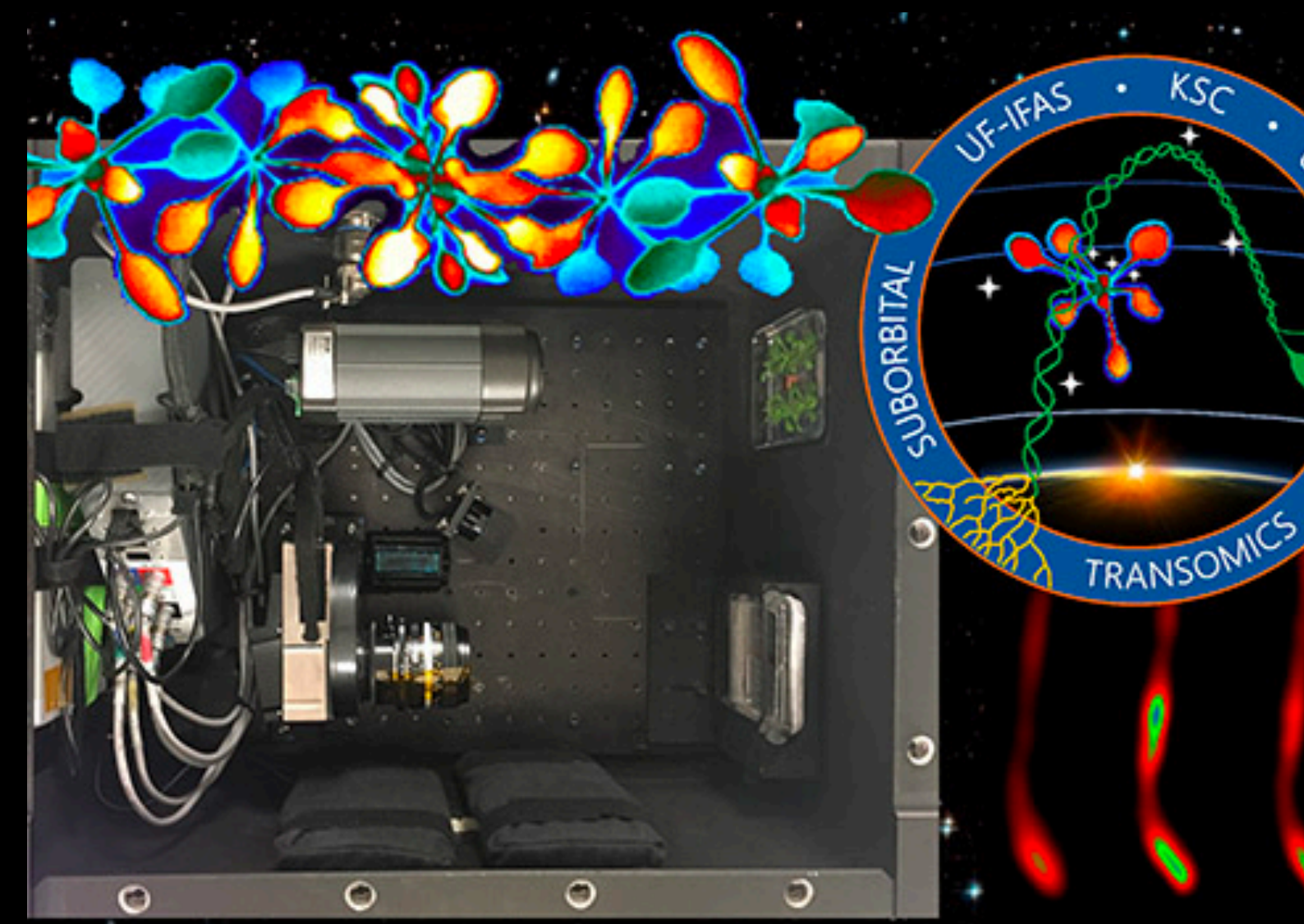
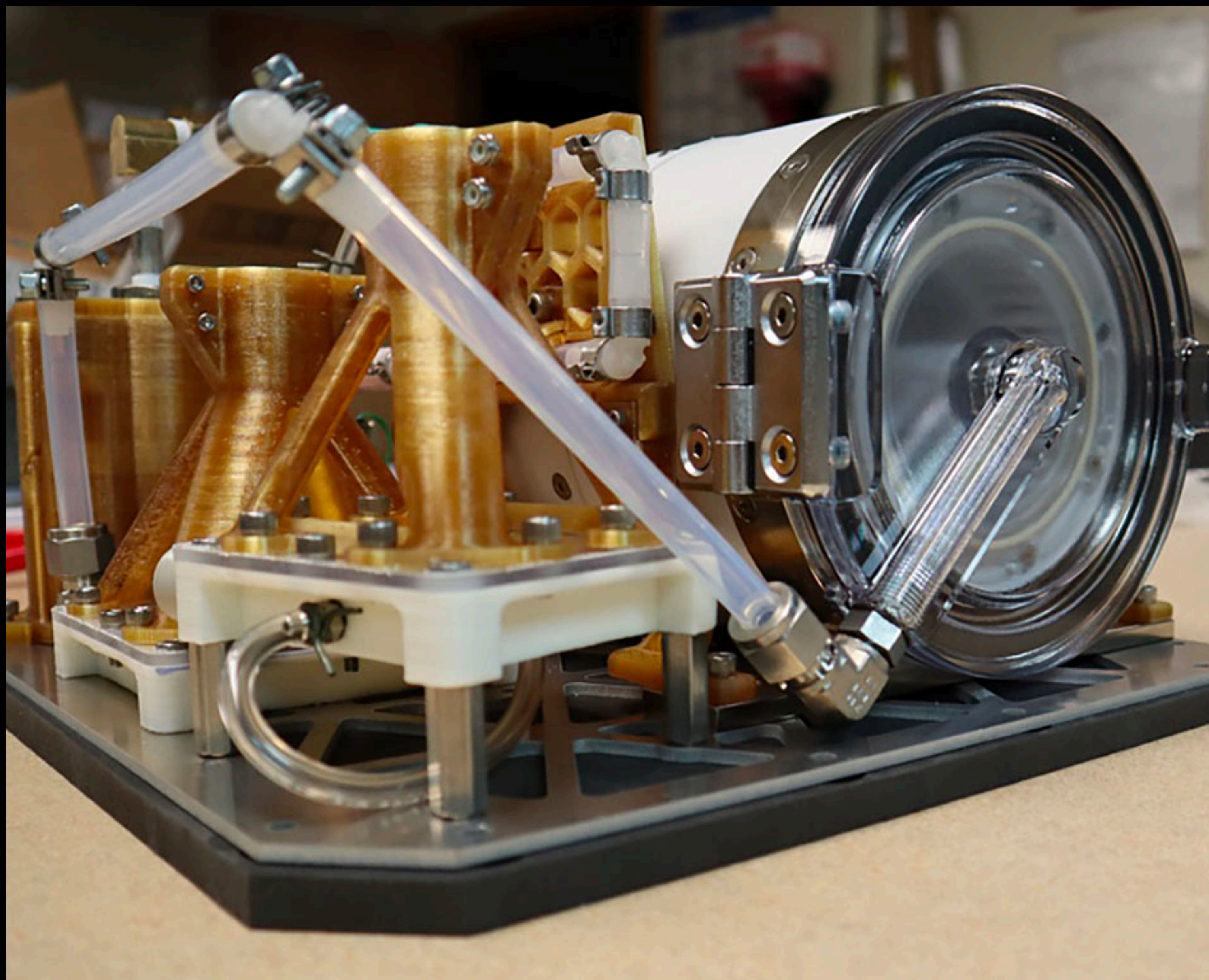
NEW SHEPARD LIFTOFF

BOOSTER LANDING

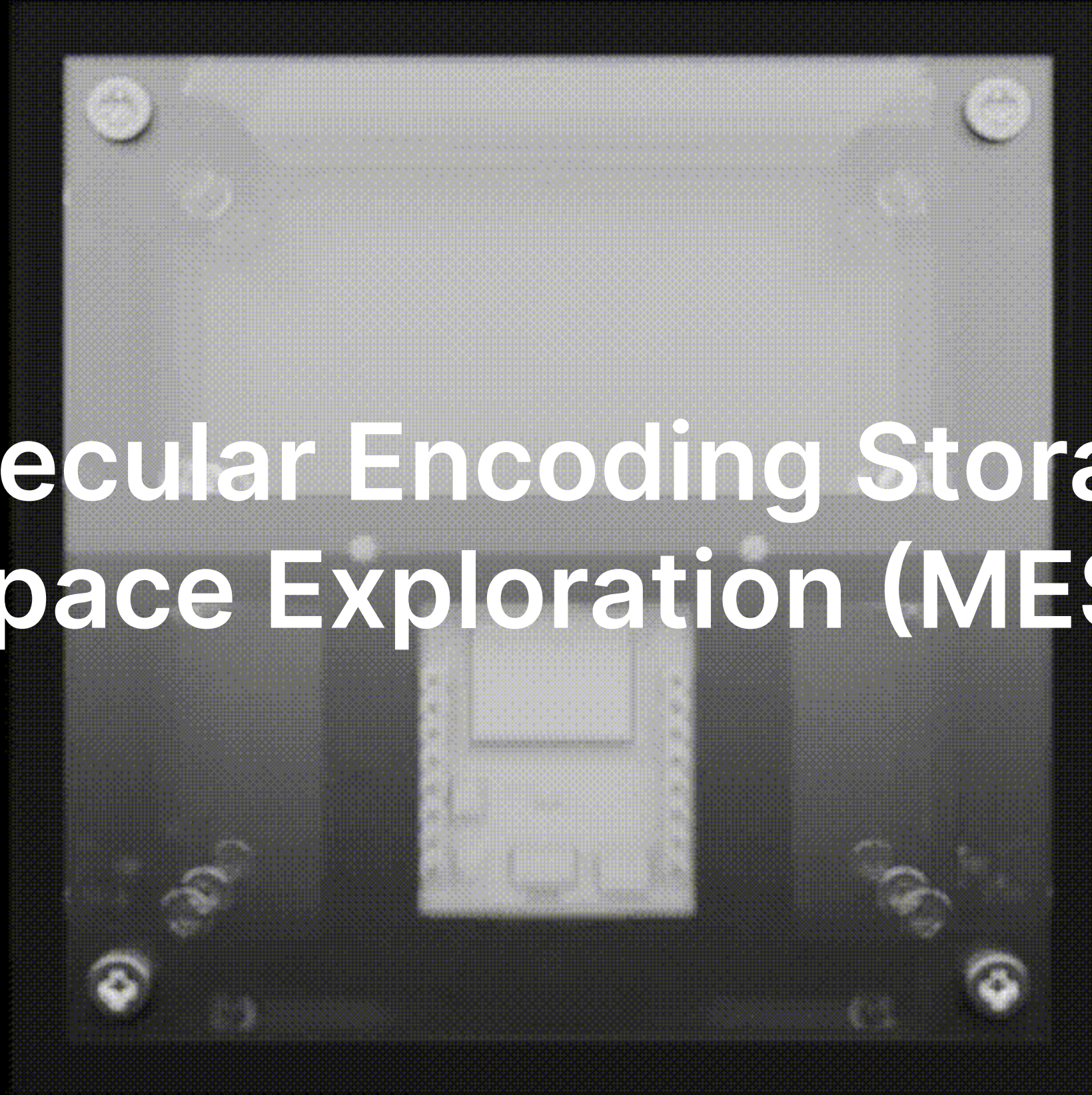
CAPSULE LANDING

WEST TEXAS LAUNCH SITE 3,700 FT ABOVE MSL



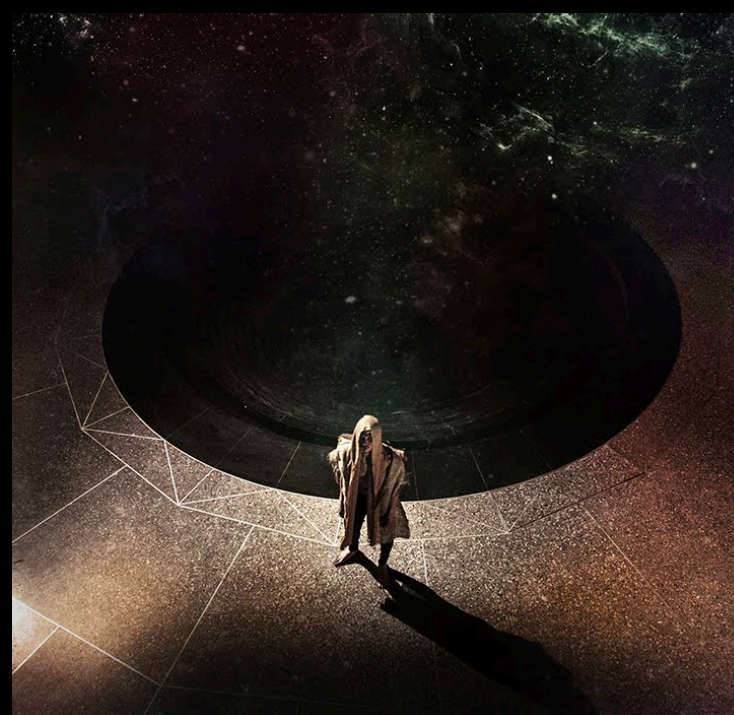


Molecular Encoding Storage for Space Exploration (MESSE)

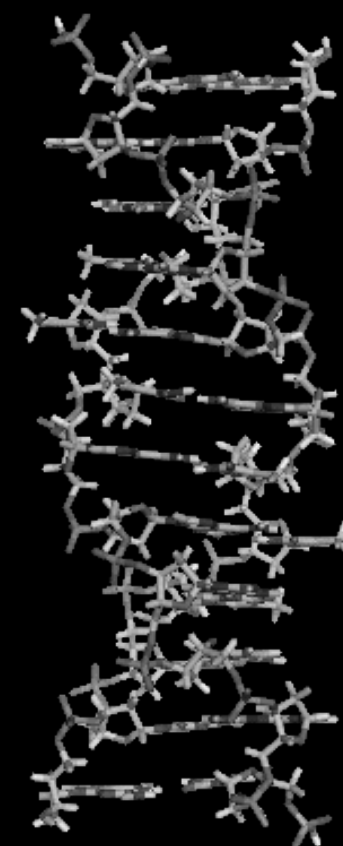




MESSE



The Dream and
the Universe

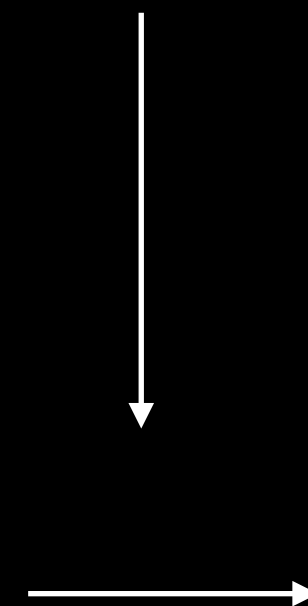


Encoded DNA



Launch to
Space

Waiting for the
Package to be
shipped back to
Thailand

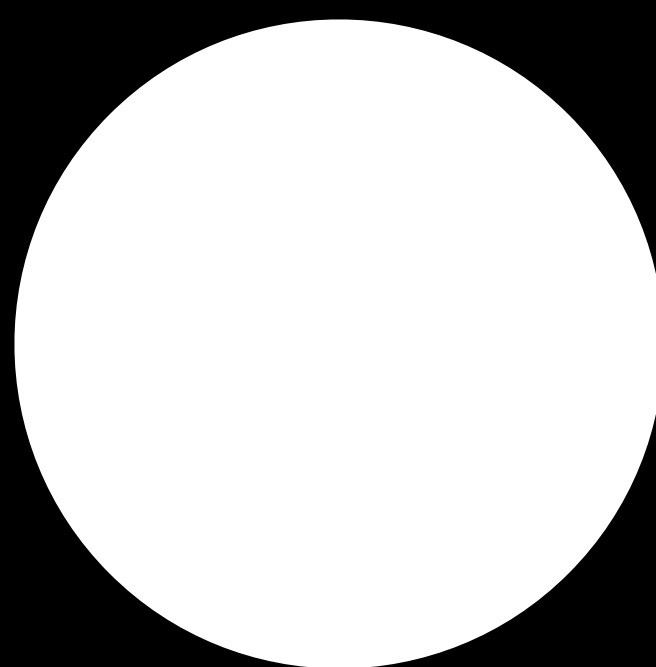


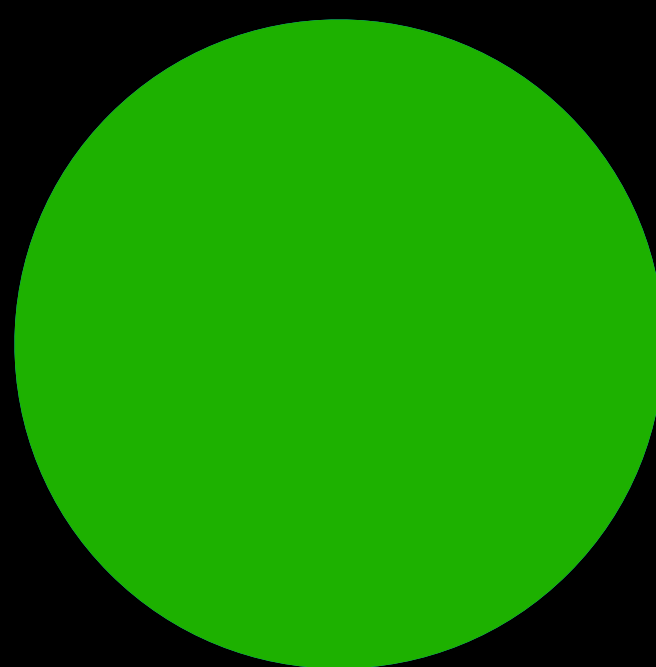
Researches and
Experiments

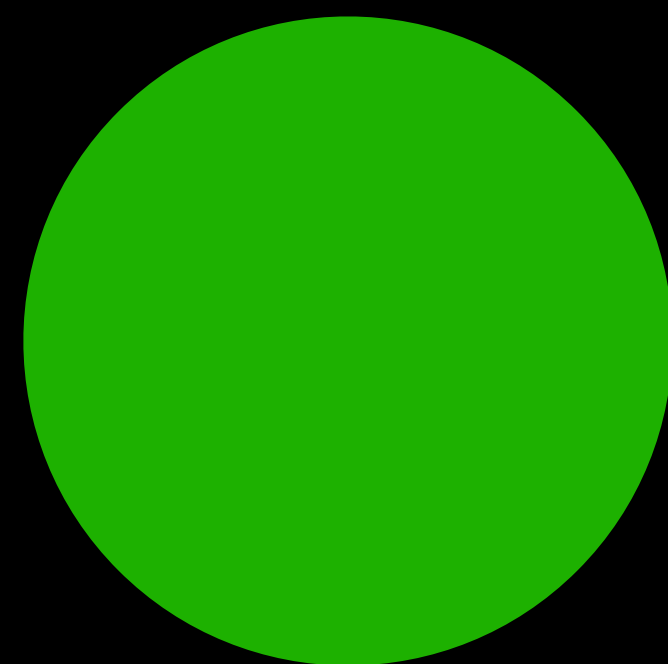
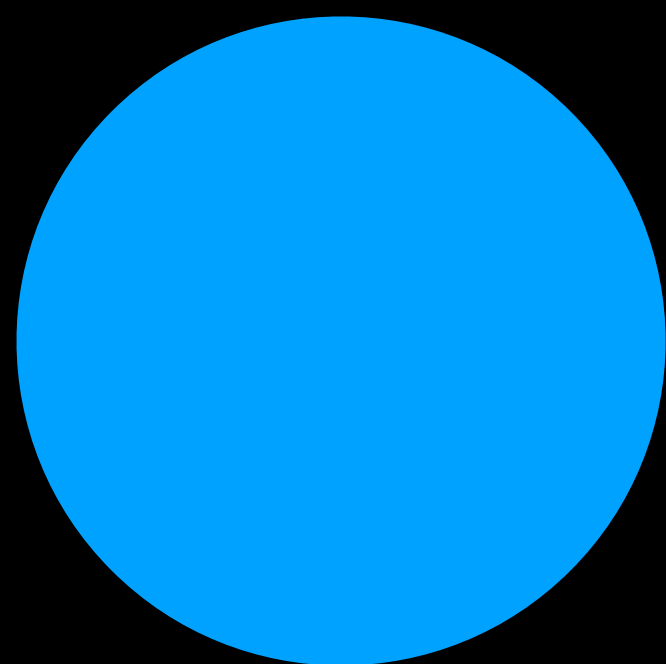


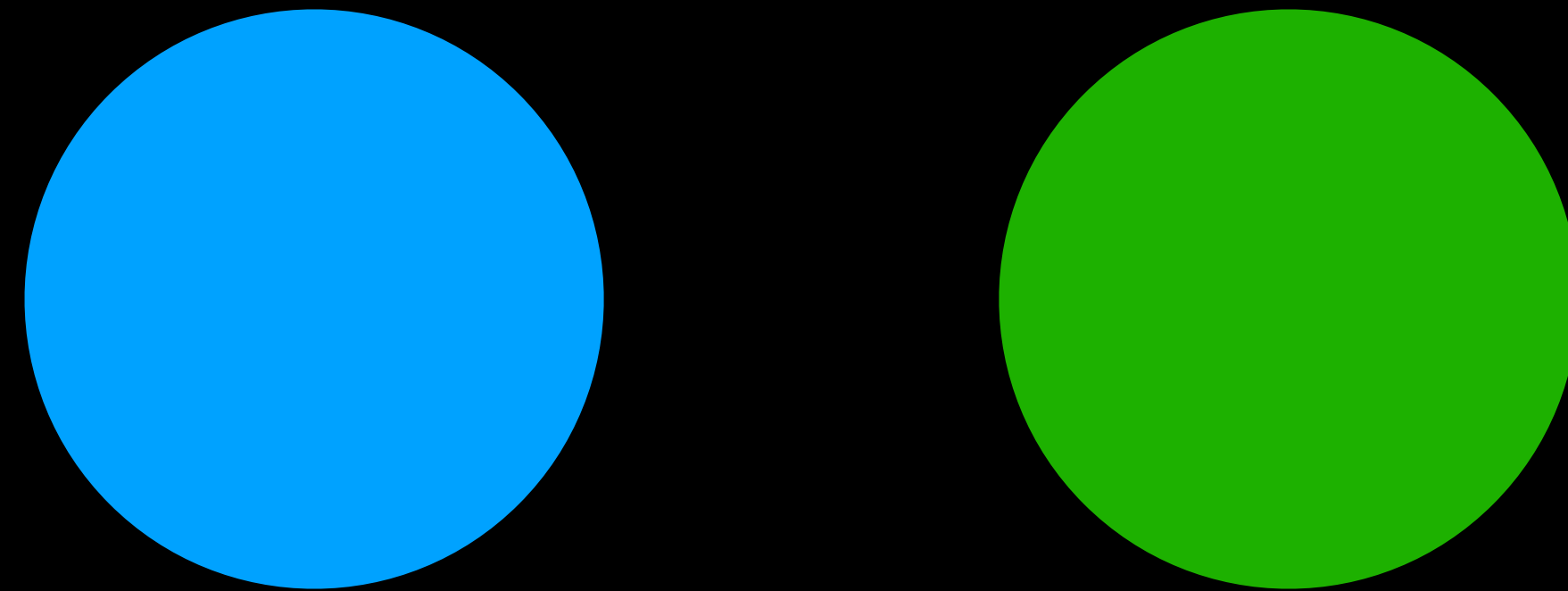
“ DNA Storage”

How to store data?

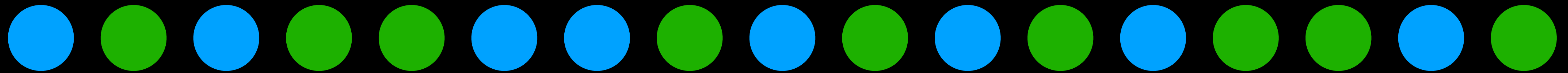






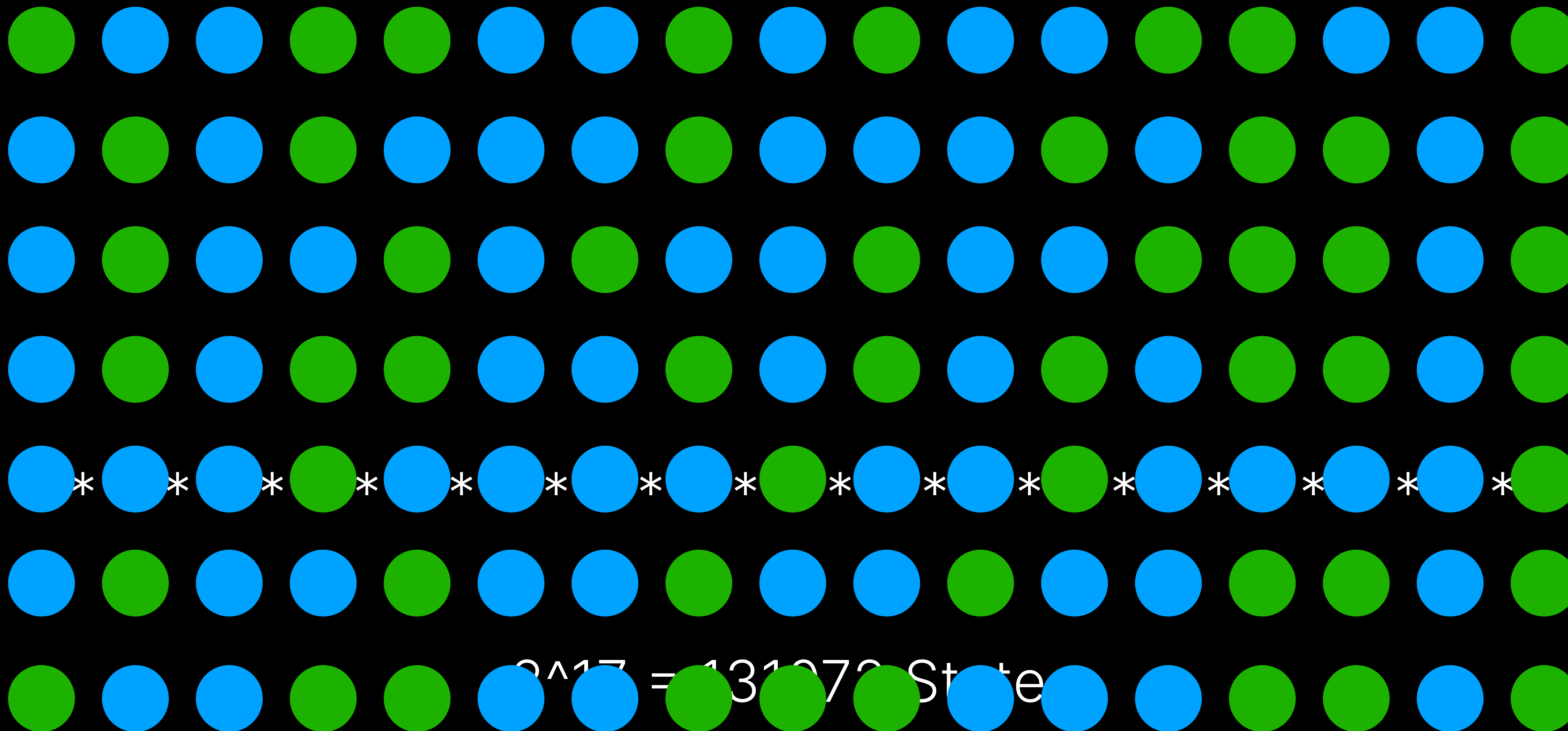


Two States Storage



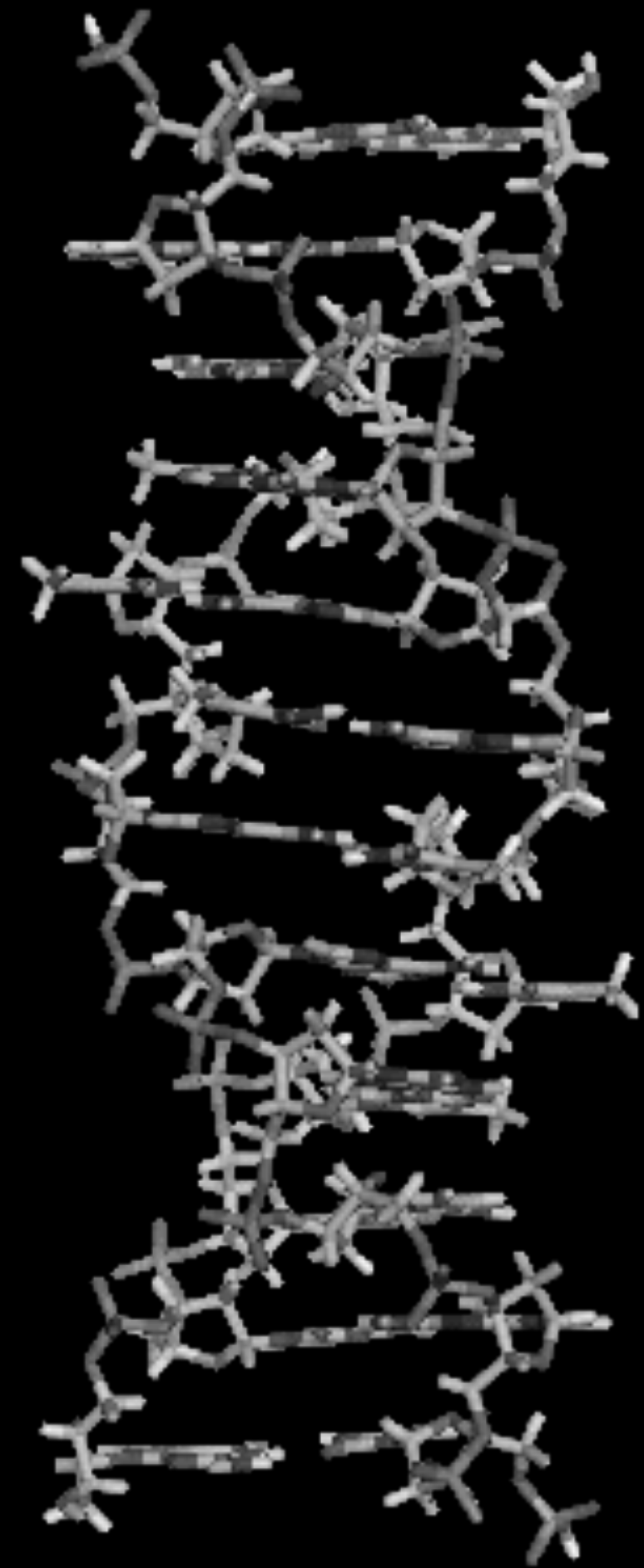
2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2

$$2^{17} = 131072 \text{ States}$$





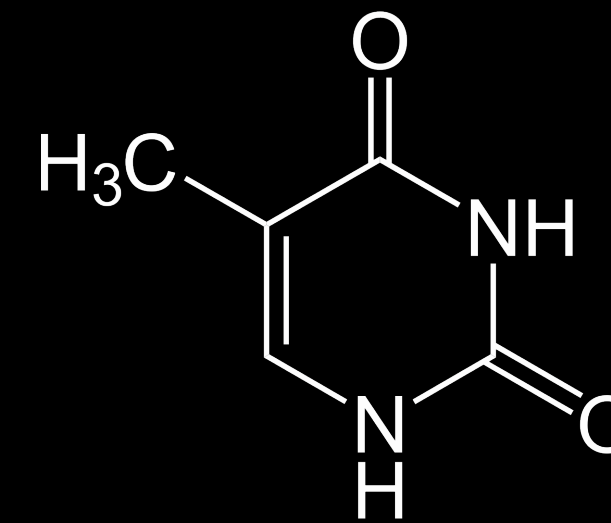
Binary



DNA



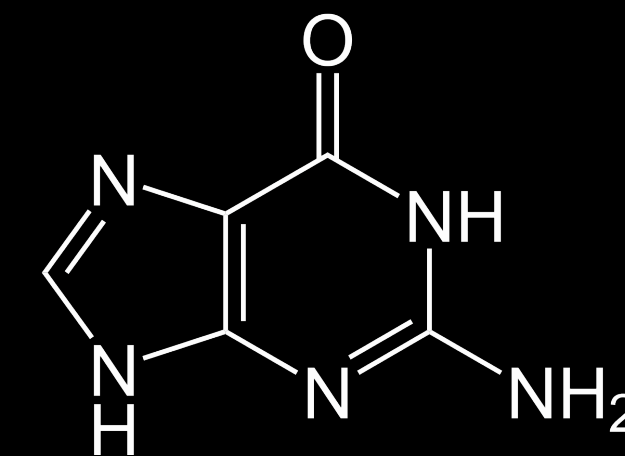
Adenine



Thymine



Cytosine

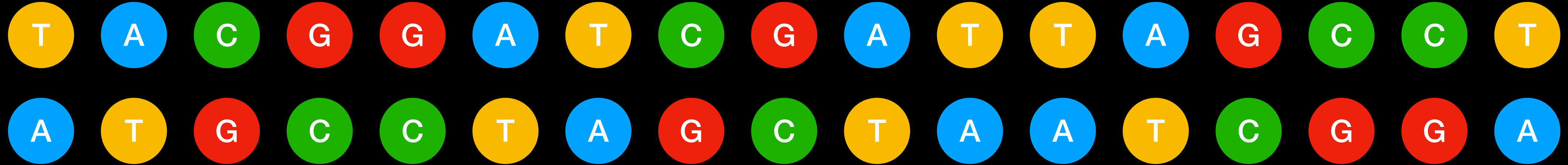


Guanine

A T G C C T A G C T A A T C G G A

4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4 * 4

$4^{17} = 17179869184$ States



“Base Pair (bp)”

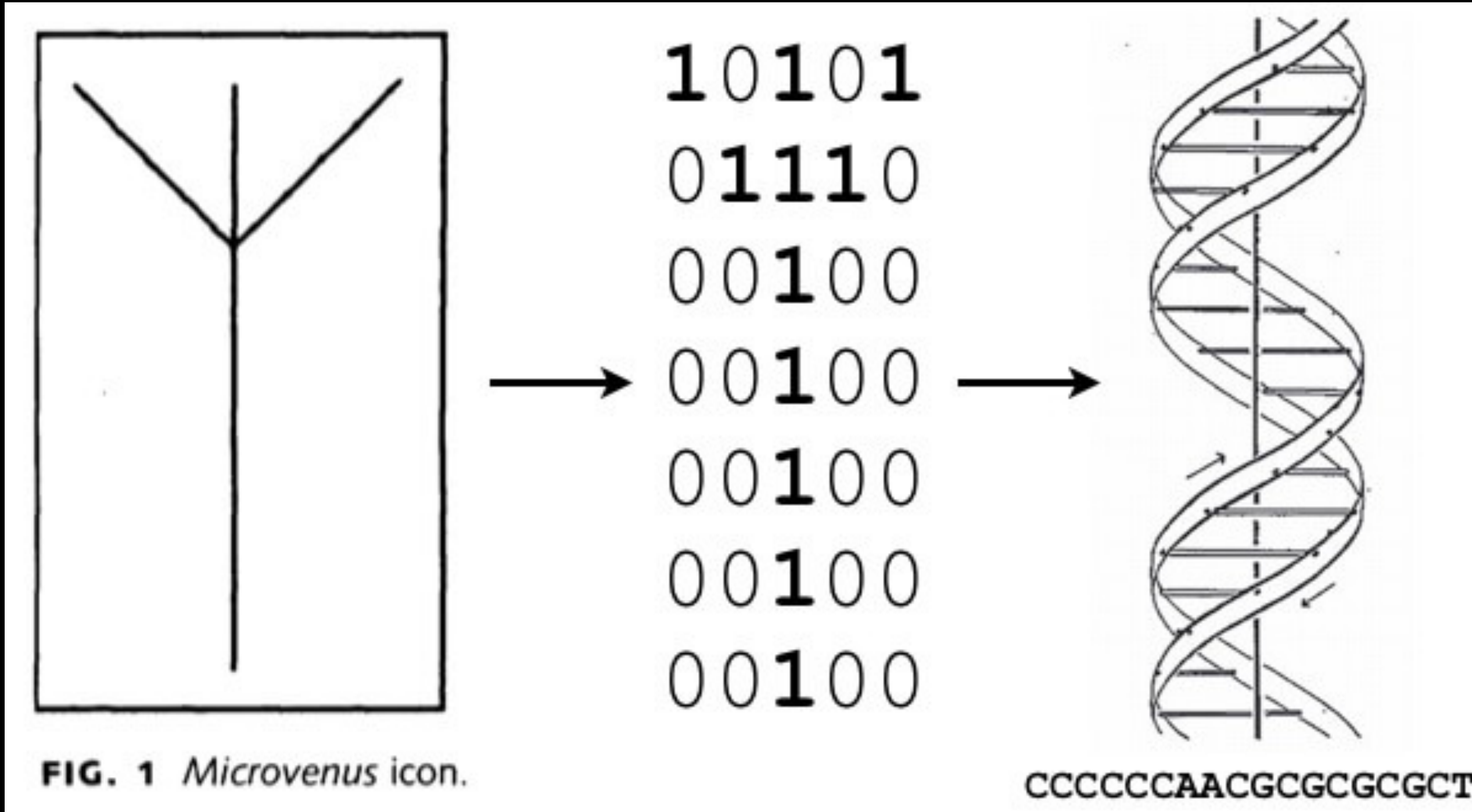


FIG. 1 *Microvenus* icon.

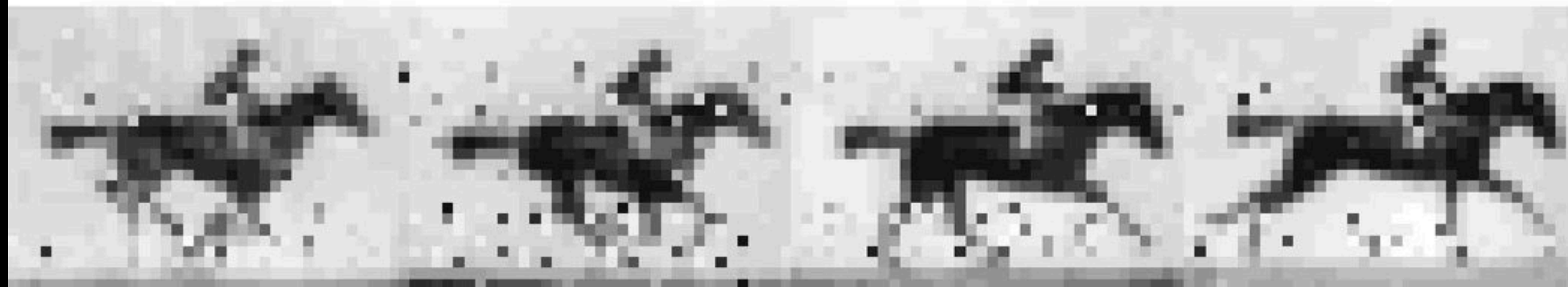
5'-CTTAAAGGGGCCCCCAACGCGCGCGCT-3'
|||||
3'-GAATTTCCCCGGGGGTTGCGCGCGCGA-5'
(COMPLEMENTARY STRAND)
Double-Stranded *Microvenus* DNA

Joe Davis' "Microvenus", the first demonstration of DNA Storage, Late 1980s

encoded GIF



recalled GIF



Shipman, S., Nivala, J., Macklis, J. et al. CRISPR–Cas encoding of a digital movie into the genomes of a population of living bacteria. *Nature* 547, 345–349 (2017). <https://doi.org/10.1038/nature23017>

How to store data?

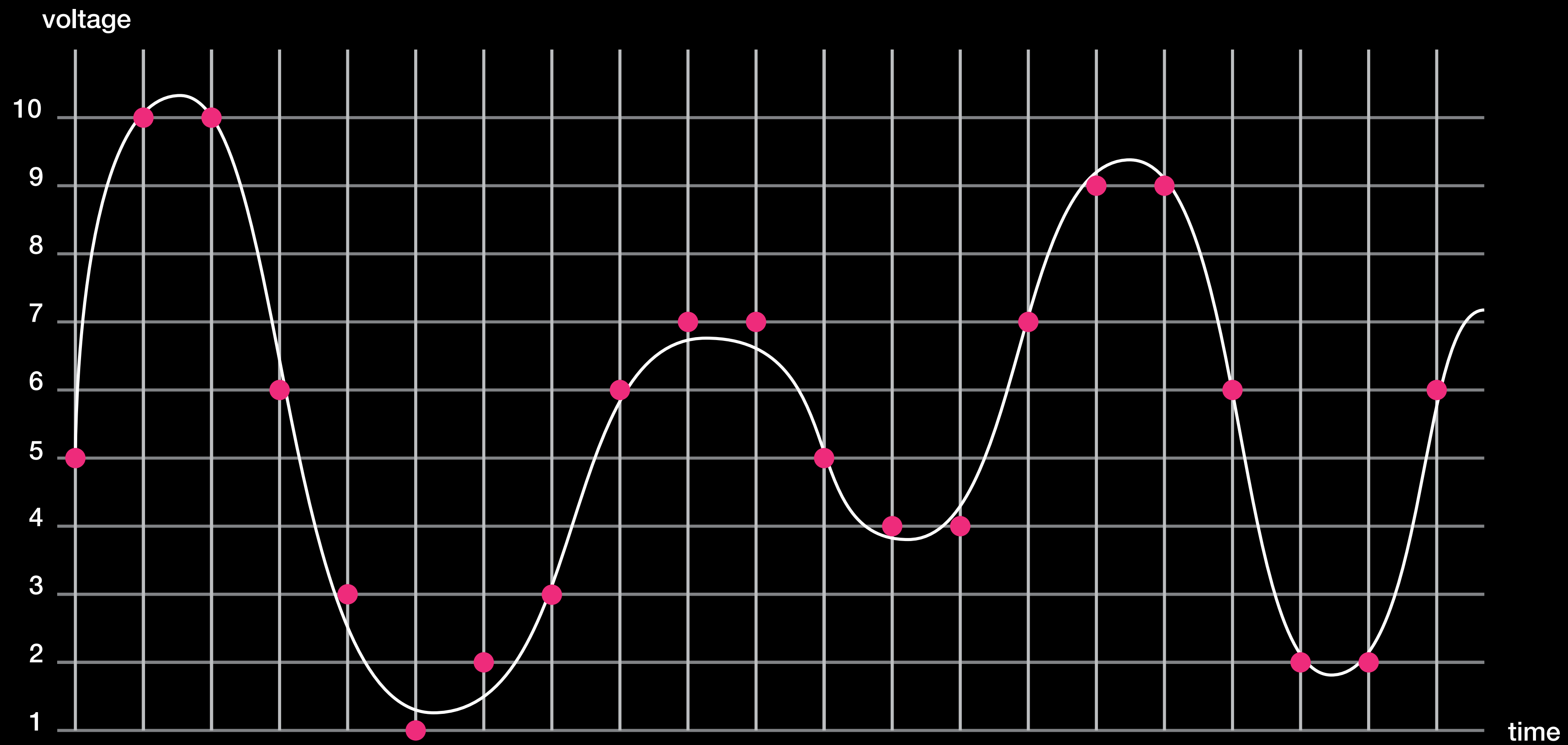
How to **a song** into DNA?

(~500 base pairs)



1st Idea

Soundwave



The First Part of The Dream and The Universe

4 (bits per dot) * 400 (sample rate per second)
* 13 (seconds) = 20800 bits

The First Part of The Dream and The Universe

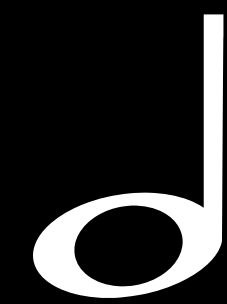
4 (bits per dot) * 400 (sample rate per second)
* 13 (seconds) = 20800 bits = (at least) 10400bp

cost = wtf



2nd Idea

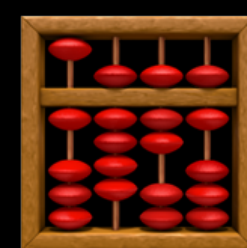
(Somehow) store a Musical Note



[A][A][G] - [T][C][G] - [A][T][C] - [G][T][A] - [C][C][C]

(one note at the time)





 **Let's do it** 

1. Song → Notes

1. Pitch (C3, D4#, F1)
2. Duration

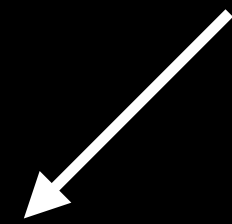
1. C2 (Black Note) → [] [] []

2. C2 + *8 → [] [] + [] []

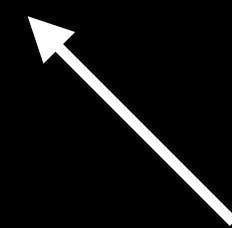
1. C2 (Black Note) → $\square\square\square$

2. C2 + *8 → $\square\square + \square\square$

*2 (H = Hold)



E2-H0-G2-F#2-H0-G2-C2-H1-G2-
D2-B2-G2-F#2-D2-E2-H0-G2-F#2-
H0-G2-C2-H1-G2-D2-B2-G2-F#2-D2



*4 (H = Hold)

E2-H0-G2-F#2-H0-G2-C2-H1-G2-D2-B2-G2-F#2-D2-E2-H0-G2-F#2-H0-G2-C2-H1-G2-D2-B2-G2-F#2-D2-E2-H0-G2-F#2-H0-G2-C2-H1-G2-D2-B2-G2-F#2-D2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-F#2-G2-H2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-D3-H1-B2-H2-E3-E3-F#3-H1-U0-D3-H0-B2-H1-U0-E3-E3-F#3-H1-U0-G3-H0-A3-H1-U0-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-F#2-G2-H2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-D3-H1-B2-H2-E3-E3-F#3-H1-U0-D3-H0-B2-H1-U0-E3-E3-F#3-H1-U0-G3-H0-A3-H1-U0-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-F#2-G2-H2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-D3-H1-B2-H2-E3-E3-F#3-H1-U0-D3-H0-B2-H1-U0-E3-E3-F#3-H1-U0-G3-H0-A3-H1-U0-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-H0-D3-H0-B2-H2-G2-H0-G2-H0-D3-H2-U0-S0-U0-E3-F#3-H0-G3-H0-F#3-H0-E3-H0-D3-H0-E3-H0-D3-H0-A2-H0-G2-H0-D3-H2-U0-B2-C3-D3-A2-H1-U0-B2-H0-G2-H1-U0-D2-H0-D3-H2-U0-E3-F#3-H0-G3-H0-F#3-H0-E3-H0-D3-H0-A3-H0-U0-B3-A3-H0-G3-H0-G3-H1-S0-G3-H0-F#3-H1-U0-D3-H1-E3-H2-S1-E3-E3-H0-F#3-H0-G3-H1-G3-H0-U0-F#3-H1-D3-H1-E3-H2-U0-S1-D3-H0-B2-H2-S1-U0-G2-G2-H0-A2-H1-G2-H2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-F#2-H0-G2-H2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-D3-H1-B2-H2-E3-E3-F#3-H1-U0-D3-H0-B2-H1-U0-E3-E3-F#3-H1-U0-G3-H0-A3-H1-U0-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-H0-D3-H0-B2-H2-G2-H0-G2-H0-D3-H2-U0-S0-U0-E3-F#3-H0-G3-H0-F#3-H0-E3-H0-D3-H0-E3-H0-D3-H0-A2-H0-G2-H0-D3-H2-U0-B2-C3-D3-A2-H1-U0-B2-H0-G2-H1-U0-D2-H0-D3-H2-U0-E3-F#3-H0-G3-H0-F#3-H0-E3-H0-D3-H0-A3-H0-U0-B3-A3-H0-G3-H0-G3-H1-S0-G3-H0-F#3-H1-U0-D3-H1-E3-H2-S1-E3-E3-H0-F#3-H0-G3-H1-G3-H0-U0-F#3-H1-D3-H1-E3-H2-U0-S1-D3-H0-B2-H2-S1-U0-D3-H0-D3-H0-D3-H0-E3-H2-U0

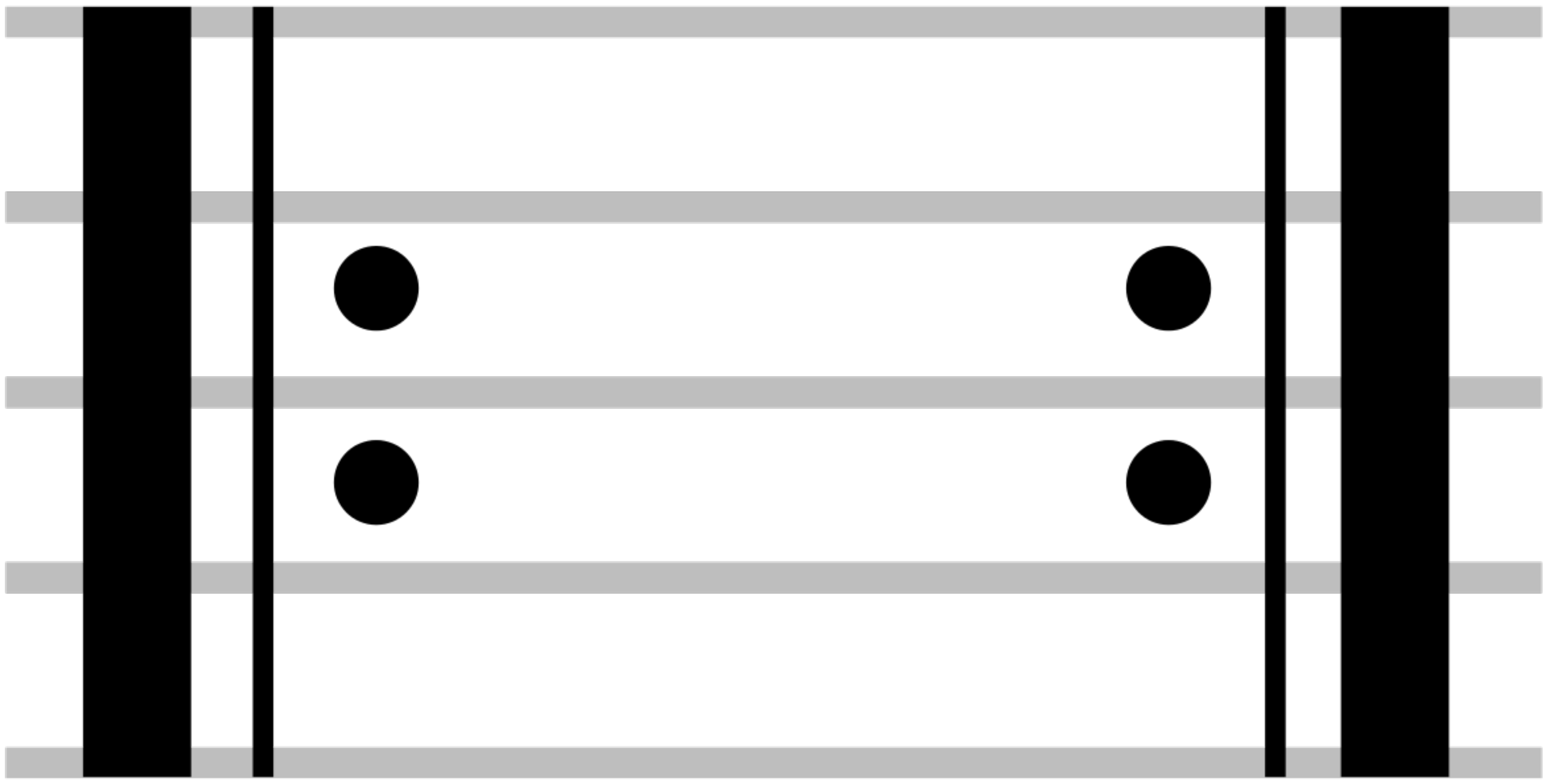
Let's listen 👂

1. Song → Notes

2. Optimize Notes

Let's listen (again) 🦻

there are repeat patterns!



A1-C2-D3-A1-C2-D3-A1-C2-D3

A1-C2-D3-A1-C2-D3-A1-C2-D3

A1-C2-D3-R-R-R

Greedy + Brute force



```
const findSequence = (data: string): string[] => {
  let i: number;
  let j: number;
  let same: any = [];

  if (data[data.length - 1] !== '-') data = data + '-';
  //console.log(data, data.length);

  let dataCalc =
    data.length % 2 === 0 ? data.length : data.length - 3;

  for (i = dataCalc / 2; i >= 9; i -= 3) {
    let max: number = 0;
    let strMap: Map<string, Array<number>> = new Map();
    let numMap: Map<number, Array<string>> = new Map();
    for (j = 0; j < data.length - i + 1; j += 3) {
      let str: string = data.substr(j, i);
      if (!strMap.has(str)) {
        strMap.set(str, [1, j]);
        // console.log('new key', str, strMap.get(str));
      } else {
        let [count, loc] = strMap.get(str);
        if (j >= loc + i) {
          strMap.set(str, [++count, j]);
          //console.log('update key', str, j, strMap.get(str));
        }
      }
    }
  }

  strMap.forEach((valueArr: number[], key: string) => {
    const value: number = valueArr[0];
    if (value >= max) {
      max = value;
      if (!numMap.has(value)) {
        numMap.set(value, [key]);
      } else {
        let keyArr = numMap.get(value);
        keyArr.push(key);
        numMap.set(value, keyArr);
      }
    }
  });

  if (max > 1) {
    //console.log(i, max, numMap.get(max));
    same.push([
      data.length - i * (max - 1) + (max + 1) * 3,
      numMap.get(max),
    ]);
  }
}

const compressLength: number[] = same.map((x) => x[0]);
const indexOfMin: number = compressLength.indexOf(
  Math.min(...compressLength),
);
const minString: string[] = same.map((x) => x[1])[indexOfMin];
//console.log(same.map(x => x[1]));

return minString;
};
```

E2-H0-G2-F#2-H0-G2-C2-H1-G2-D2-B2-G2-F#2-D2-E2-H0-G2-F#2-H0-G2-C2-H1-G2-D2-B2-G2-F#2-D2-E2-H0-G2-F#2-H0-G2-C2-H1-G2-D2-B2-G2-F#2-D2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-F#2-G2-H2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-D3-H1-B2-H2-E3-E3-F#3-H1-U0-D3-H0-B2-H1-U0-E3-E3-F#3-H1-U0-G3-H0-A3-H1-U0-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-F#2-G2-H2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-D3-H1-B2-H2-E3-E3-F#3-H1-U0-D3-H0-B2-H1-U0-E3-E3-F#3-H1-U0-G3-H0-A3-H1-U0-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-F#2-G2-H2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-D3-H1-B2-H2-E3-E3-F#3-H1-U0-D3-H0-B2-H1-U0-E3-E3-F#3-H1-U0-G3-H0-A3-H1-U0-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-H0-D3-H0-B2-H2-G2-H0-G2-H0-D3-H2-U0-S0-U0-E3-F#3-H0-G3-H0-F#3-H0-E3-H0-D3-H0-E3-H0-D3-H0-A2-H0-G2-H0-D3-H2-U0-B2-C3-D3-A2-H1-U0-B2-H0-G2-H1-U0-D2-H0-D3-H2-U0-E3-F#3-H0-G3-H0-F#3-H0-E3-H0-D3-H0-A3-H0-U0-B3-A3-H0-G3-H0-G3-H1-S0-G3-H0-F#3-H1-U0-D3-H1-E3-H2-S1-E3-E3-H0-F#3-H0-G3-H1-G3-H0-U0-F#3-H1-D3-H1-E3-H2-U0-S1-D3-H0-B2-H2-S1-U0-G2-G2-H0-A2-H1-G2-H2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-F#2-H0-G2-H2-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-D3-H1-B2-H2-E3-E3-F#3-H1-U0-D3-H0-B2-H1-U0-E3-E3-F#3-H1-U0-G3-H0-A3-H1-U0-S0-E3-D3-H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-G2-H0-D3-H0-B2-H2-G2-H0-G2-H0-D3-H2-U0-S0-U0-E3-F#3-H0-G3-H0-F#3-H0-E3-H0-D3-H0-E3-H0-D3-H0-A2-H0-G2-H0-D3-H2-U0-B2-C3-D3-A2-H1-U0-B2-H0-G2-H1-U0-D2-H0-D3-H2-U0-E3-F#3-H0-G3-H0-F#3-H0-E3-H0-D3-H0-A3-H0-U0-B3-A3-H0-G3-H0-G3-H1-S0-G3-H0-F#3-H1-U0-D3-H1-E3-H2-S1-E3-E3-H0-F#3-H0-G3-H1-G3-H0-U0-F#3-H1-D3-H1-E3-H2-U0-S1-D3-H0-B2-H2-S1-U0-D3-H0-D3-H0-D3-H0-E3-H2-U0

E3-E3-F#3-H1-U0-R7-H0-G2-H0-D3-H2-U0-R6-H0-D3-H0-R5-U0-E3-F#3-H0-G3-
H0-F#3-H0-E3-R5-R4-G2-H2-R1-D3-H1-B2-H2-R7-D3-H0-B2-H1-U0-R7-G3-H0-A3-
H1-U0-R1-G2-R3-E2-H0-G2-F#2-H0-G2-C2-H1-G2-D2-B2-G2-F#2-D2-R2-S0-E3-D3-
H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-R1-R3-R5-B2-H2-G2-R6-S0-R4-
E3-R5-A2-R6-B2-C3-D3-A2-H1-U0-B2-H0-G2-H1-U0-D2-H0-D3-H2-R4-A3-H0-U0-
B3-A3-H0-G3-H0-G3-H1-S0-G3-H0-F#3-H1-U0-D3-H1-E3-H2-S1-E3-E3-H0-F#3-
H0-G3-H1-G3-H0-U0-F#3-H1-D3-H1-E3-H2-U0-S1-D3-H0-B2-H2-S1-U0-R0-R2-R2-
R2-R2-R1-G2-F#2-R3-F#2-G2-H2-R1-G2-F#2-R0-G2-G2-H0-A2-H1-G2-H2-R1-G2-
F#2-H0-R0-D3-R5-D3-H0-E3-H2-U0

1. Song → Notes
2. Optimize Notes
- 3. Notes to Binary String**

12 Notes/Octave $\rightarrow$ 2bp ($4*4 = 16$)
Octaves $\rightarrow$ 1bp (4)

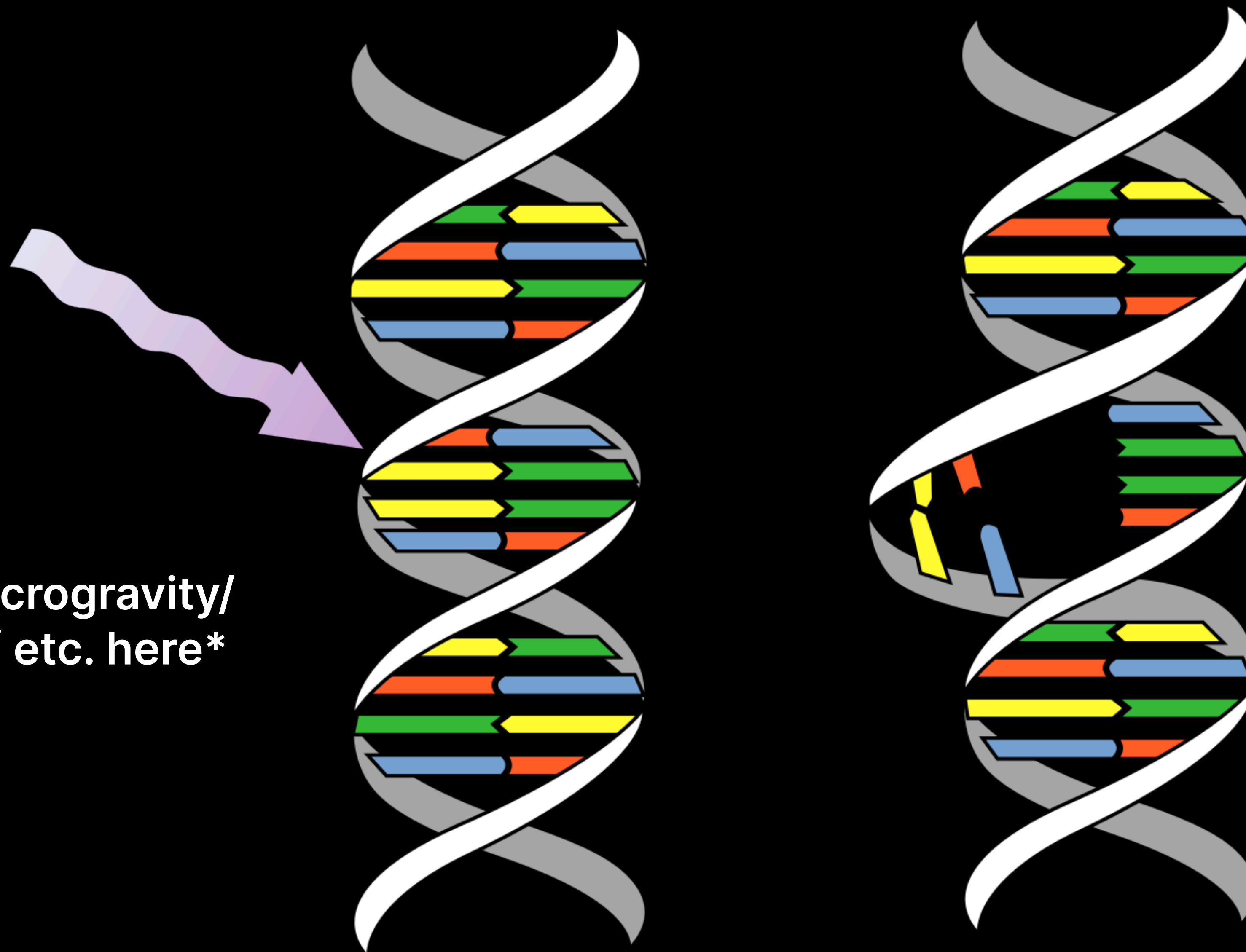
E3-E3-F#3-H1-U0-R7-H0-G2-H0-D3-H2-U0-R6-H0-D3-H0-R5-U0-E3-F#3-H0-G3-
H0-F#3-H0-E3-R5-R4-G2-H2-R1-D3-H1-B2-H2-R7-D3-H0-B2-H1-U0-R7-G3-H0-A3-
H1-U0-R1-G2-R3-E2-H0-G2-F#2-H0-G2-C2-H1-G2-D2-B2-G2-F#2-D2-R2-S0-E3-D3-
H0-B2-H0-C3-H0-B2-H0-A2-H1-G2-G2-H0-A2-H0-R1-R3-R5-B2-H2-G2-R6-S0-R4-
E3-R5-A2-R6-B2-C3-D3-A2-H1-U0-B2-H0-G2-H1-U0-D2-H0-D3-H2-R4-A3-H0-U0-
B3-A3-H0-G3-H0-G3-H1-S0-G3-H0-F#3-H1-U0-D3-H1-E3-H2-S1-E3-E3-H0-F#3-
H0-G3-H1-G3-H0-U0-F#3-H1-D3-H1-E3-H2-U0-S1-D3-H0-B2-H2-S1-U0-R0-R2-R2-
R2-R2-R1-G2-F#2-R3-F#2-G2-H2-R1-G2-F#2-R0-G2-G2-H0-A2-H1-G2-H2-R1-G2-
F#2-H0-R0-D3-R5-D3-H0-E3-H2-U0

01000101000101100111000111011011111110000011100110000001001110010110110111
1101100000001001110000111101110110010001011001110000011101110000001100111000
0010001111101111100011100110010111001001001110001101100110010111111001001110
00010110011000111011011111101110111000010010111000111011011100101110011101101
000011000001110001100011000001110000000001100010111000010001011000111000
1100000100011101011010001000100100111000010110011000000000011100001011001
10000100100110001011100011100110000100100110000111001111011111101101100110
010011100111110110100111100010001111101100100111110101100000001001001100100
11000111011010110011000001110011000111011000100011000000100111001011110010
010111000011011010110110010111000001110111000001110111000111010001110111000
00110011100011101100010011100010100011100101101010100010100011100000110011
10000011101110001011101110000110110011001110001001001110001010001110010110
1101101010010011100001011001100101101011101101110001110101110101110101101011
100101110001100011101101100001110011001011100101110001100011100001110001110
01100001001001100010111001100101110010111000110001100001110000010011111010
01001110000010001110010110110

1. Song → Notes
2. Optimize Notes
3. Notes to Binary String
- 4. Error Correction**

Mutation

***insert microgravity/
radiation/ etc. here***



Error Correction Algorithm



Reed-Solomon Algorithm

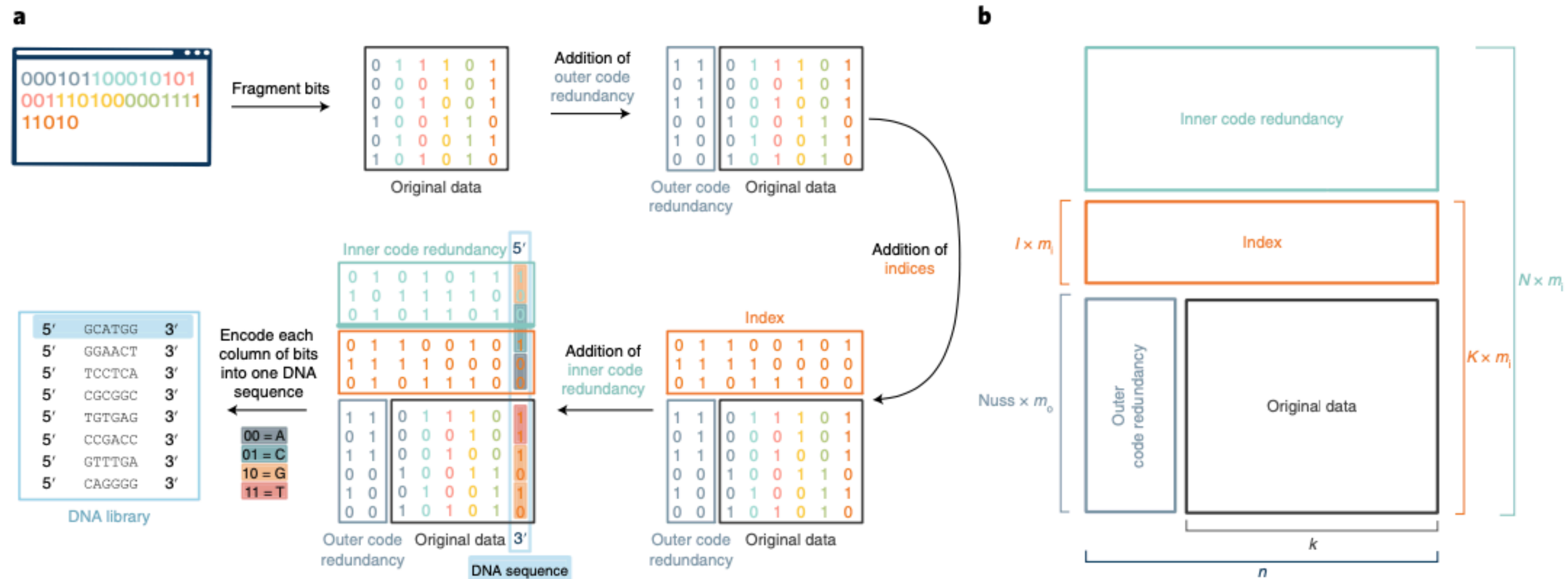


Fig. 2 | Reed-Solomon-based error-correcting coding scheme from file to DNA. a, The digital file is fragmented and appended with outer code redundancy to enable correction of missing sequences. An index is then added to every sequence to later be able to sort the individual fragments back into their original order. Subsequently, inner code redundancy is added to each sequence to enable error correction for erasures and substitutions within a sequence. The bit fragment outlined in blue corresponds to its encoded form in the last step, highlighted in blue. In the last step, each fragment of bits in the scheme is converted into a sequence of DNA, by mapping two bits to one nucleotide. **b**, Sequences of bits are aligned vertically within the scheme. Parameters shown are used by the code and offer constraints to the code design problem. Constraints include sufficient length of index and the criterion that $K \times m_i = nuss \times m_o + l \times m_i$ to ensure that the inner and outer codes fit together. All parameters can be calculated using the Excel sheet supplied in Supplementary Data 1. If the size of the data is larger than what can be fitted into one block, the encoder generates several equal-sized blocks (Supplementary Data 1). k , length of outer code message (without outer code redundancy); K , length of inner code message (without redundancy); l , length of index symbols; m_i , number of bits per inner code symbol; m_o , number of bits per outer code symbol; N , length of inner code (including redundancy); n , length of outer code (with outer redundancy); $nuss$, number of symbols of outer code per segment.

1. Song $\rightarrow$ Notes
2. Optimize Notes
3. Notes to Binary String
4. Error Correction
- 5. Randomization**

11 11 11 11 11 → GGGGGG

💡 : Let's XOR it with some other binary string to lowering the chance of this being happened

?



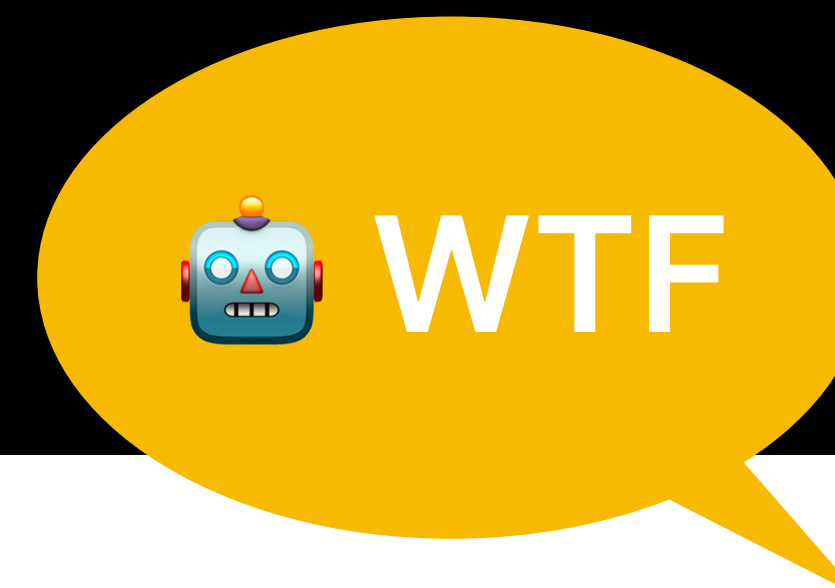
PRNGS

Pseudorandom Number Generators

```
Computer = (input) ⇒ {  
    /*do something*/  
    return output  
}
```




Hey computer, give
me a random number



PRNGS(seed) → *some (seem like) random data*

XORSHIFT



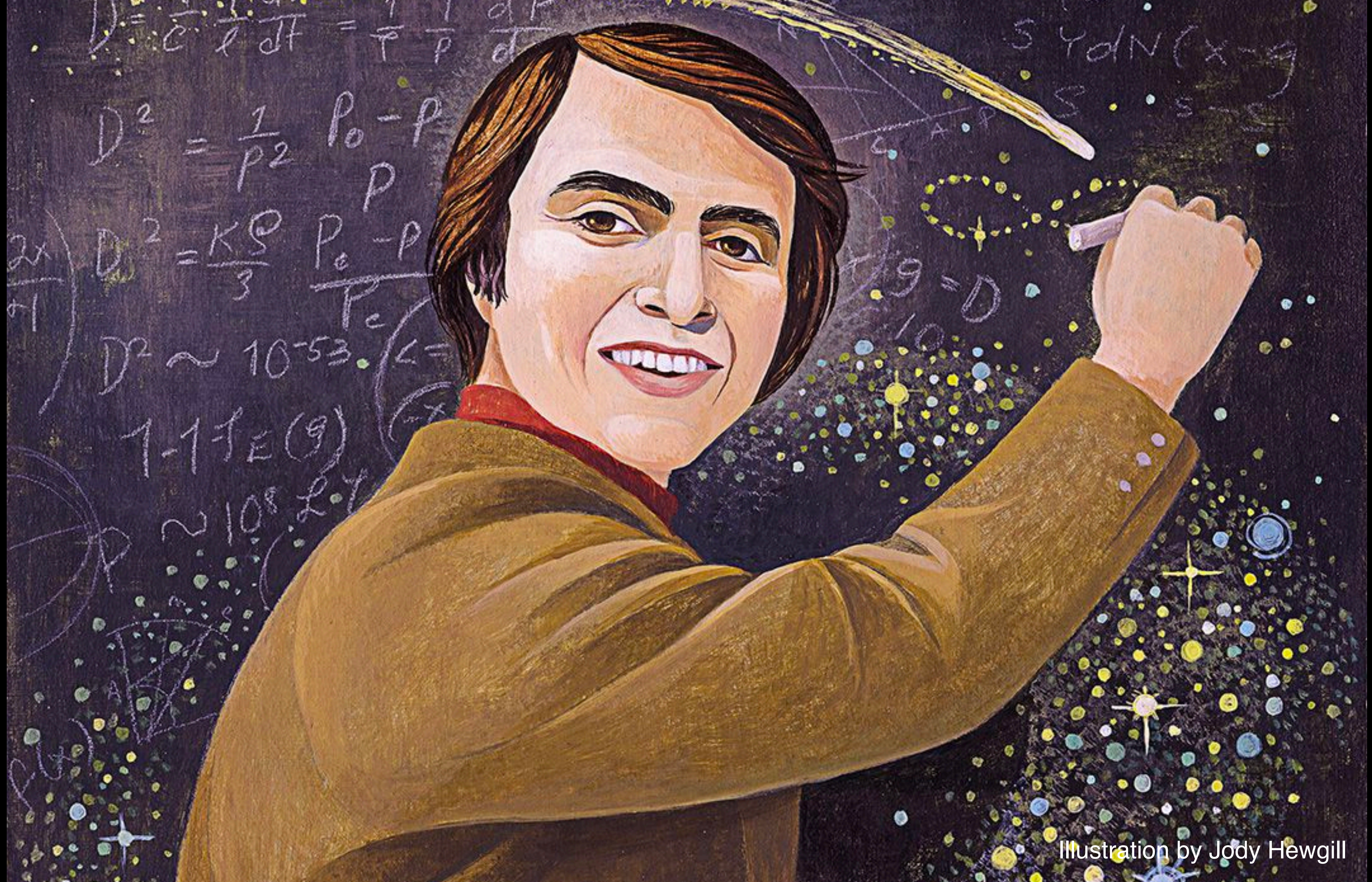
```
const xorshift = (number: number, [s1 = 1, s2 = 2]) => {
  let state0: number = s1;
  let state1: number = s2;
  let randomString: string = '';
  for (let i = 0; i < number; i++) {
    let s1: number = state0;
    let s0: number = state1;
    state0 = s0;
    s1 ^= s1 << 23;
    s1 ^= s1 >> 17;
    s1 ^= s0;
    s1 ^= s0 >> 26;
    state1 = s1;
    let randomNumber: number = Math.round(
      (state0 + state1) / 4294967296,
    );
    randomString += randomNumber.toString();
  }
  return randomString;
};
```



```
const xorshift = (number: number, [s1 = 1, s2 = 2]) => {  
  let state0: number = s1;  
  let state1: number = s2;  
  let randomString: string = '';  
  for (let i = 0; i < number; i++) {  
    let s1: number = state0;  
    let s0: number = state1;  
    state0 = s0;  
    s1 ^= s1 << 23;  
    s1 ^= s1 >> 17;  
    s1 ^= s0;  
    s1 ^= s0 >> 26;  
    state1 = s1;  
    let randomNumber: number = Math.round(  
      (state0 + state1) / 4294967296,  
    );  
    randomString += randomNumber.toString();  
  }  
  return randomString;  
};
```

seed = s1, s2

?



$$D = \frac{1}{c} \frac{dF}{dt} = \frac{1}{F} \frac{dF}{dt}$$

$$D^2 = \frac{1}{P^2} P_0 - P$$

$$D^2 = \frac{K^2}{3} \frac{P_0 - P}{P_c}$$

$$D^2 \sim 10^{-53} \cdot C =$$

$$1 - 1 \cdot E(g)$$

$$\sim 10^8 \cdot L \cdot Y$$

$s1 = 9111934 \ (9 \ 11 \ 1934)$

$s2 = 20121996 \ (20 \ 12 \ 1996)$

0 - 4294967296

random / 4294967296


```
if(random>0.5) return 1  
return 0
```

01000101000101100111000111011011111110000011100110000001001110010110110111
11011000000100111000011110111011001000101100111000001110111000001100111000
0010001111101111100011100110010111001001001110001101100110010111111001001110
00010110011000111011011111101110111000010010111000111011011100101110011101101
00001100000111000110001100000111000000001100010111000010001011000111000
110000010001110101101000100010010011100001011001100000000011100001011001
10000100100110001011100011100110000100100110000111001111011111101101100110
010011100111110110100111100010001111101100100111110101100000001001001100100
11000111011010110011000001110011000111011000100011000000100111001011110010
010111000011011010110110010111000001110111000001110111000111010001110111000
00110011100011101100010011100010100011100101101010100010100011100000110011
10000011101110001011101110000110110011001110001001001110001010001110010110
11011010100100111000010110011001011010111011011100011101011101011101011
100101110001100011101101100001110011001011100101110001100011100001110001110
01100001001001100010111001100101110010111000110001100001110000010011111010
01001110000010001110010110110111111000010101100110101001011000000110111111
111111111010001101111101111000000000011111011110101000111110000111001000000
100010010100

00111101110100011100111100011011111101001111101010111000011111001000000001100
100011110010010101110100100111100100011010011100011011000100011101011111001
000100111111001010101010011101101101100000011111000010000100000000110101101
0110111111010000001110100001001011100111010011100011101101001110111111100110
0000101111000010001110011000010001010000111011010111000110001001110010001
10011001101101101111000111010110101101101101010010111111011100110011110011001
11110101101000000100010001110000111011100001111000100111110100010000000110
01110110011101000011010100110000100100110011100101111000001100010101011101
000001110100110000000000010110000010001001001001110000111000010011010101
1100111111101010000010110001111111000101110000111011010011101000000111110111
11010101101110010011010001100100010001100110101001110010000101001101010001
101101110110001001010101111110000010001110001101111101001101000001101001011
001010001110000110101100111101000110101111000001111001001010000101111001111
0101100111111111011100101001000110110101111111101100110001001011100111111101110
00010001011100100110010101011011101001000010101011110011111111000001011100
00011001010100010110010110000001000101000000101010101010101111100011110000
01101001100000110000010110011111111110011010001010111011101010100101100000
0011011100100

2. Optimize Notes

3. Notes to Binary String

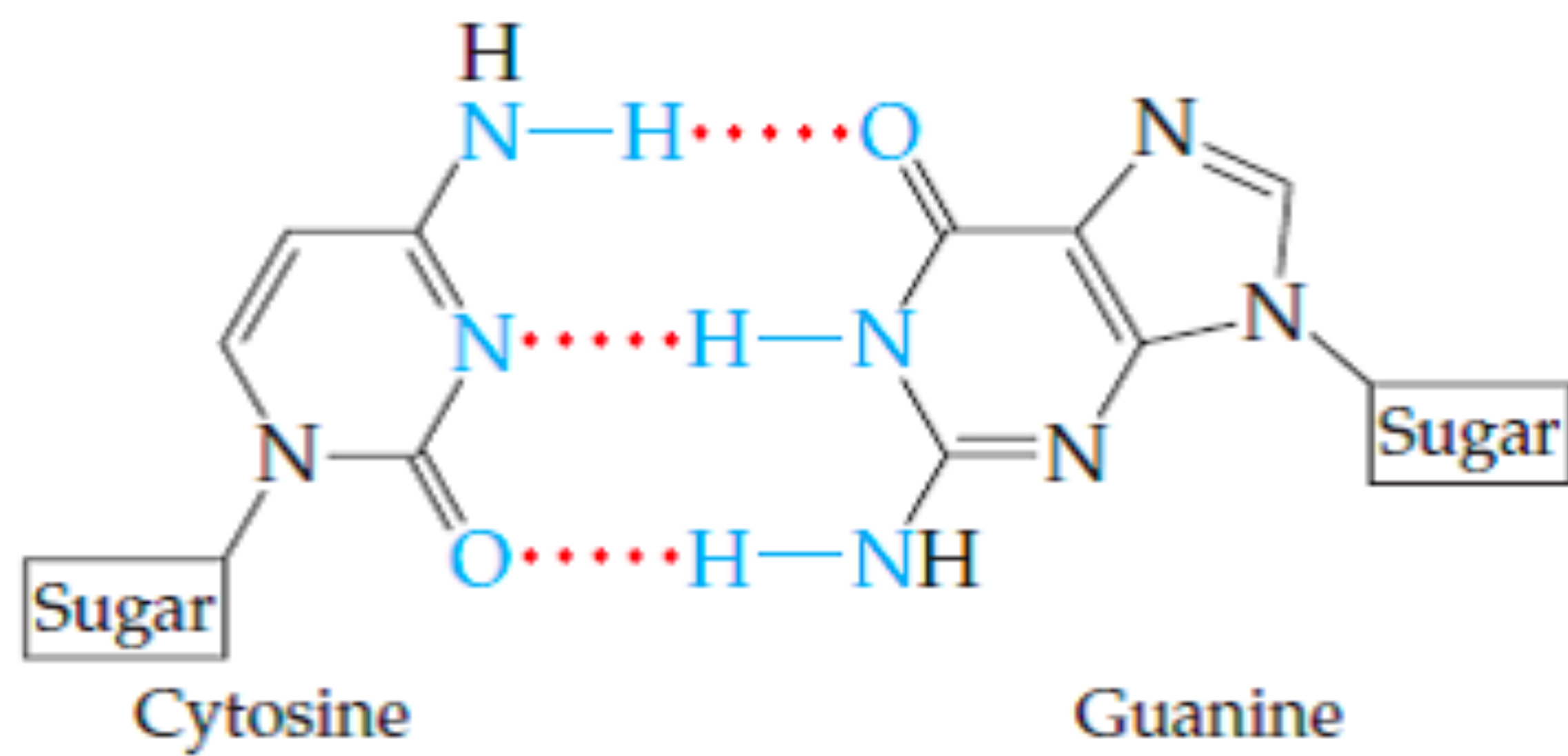
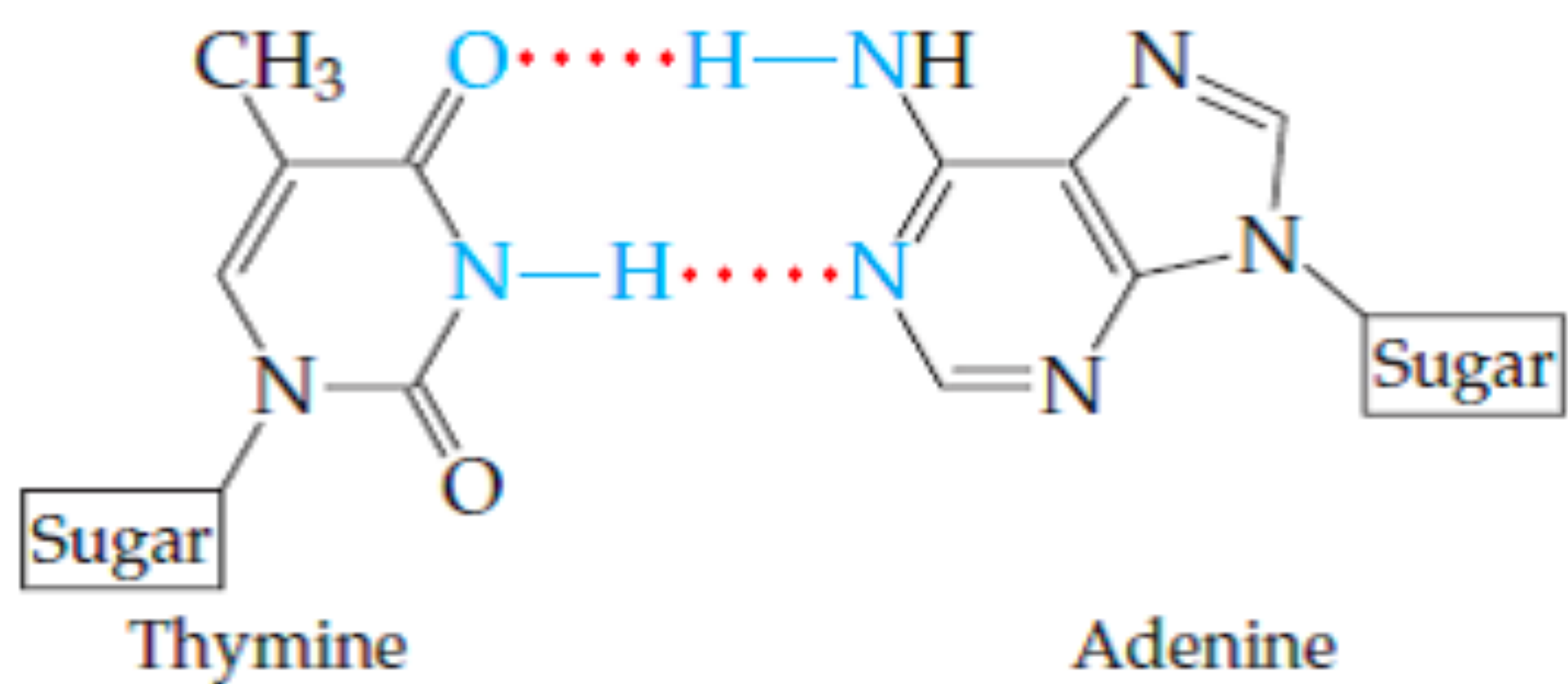
4. Error Correction

5. Randomization

6. Binary String →
Nitrogenous Bases

(Finally)

GC Content



101000011000100100101100

10 10 00 10 01 00 10 01 00 10 11 00

lets define 10 as G!

10 10 00 10 01 00 10 01 00 10 11 00

& define 11 as C!

?

GGCGACGACGTC

header

(00) C

(01) A

(10) G

(11) T

what if a header mutate?

We're f**ked!

00 → A

01 → T

10 → C

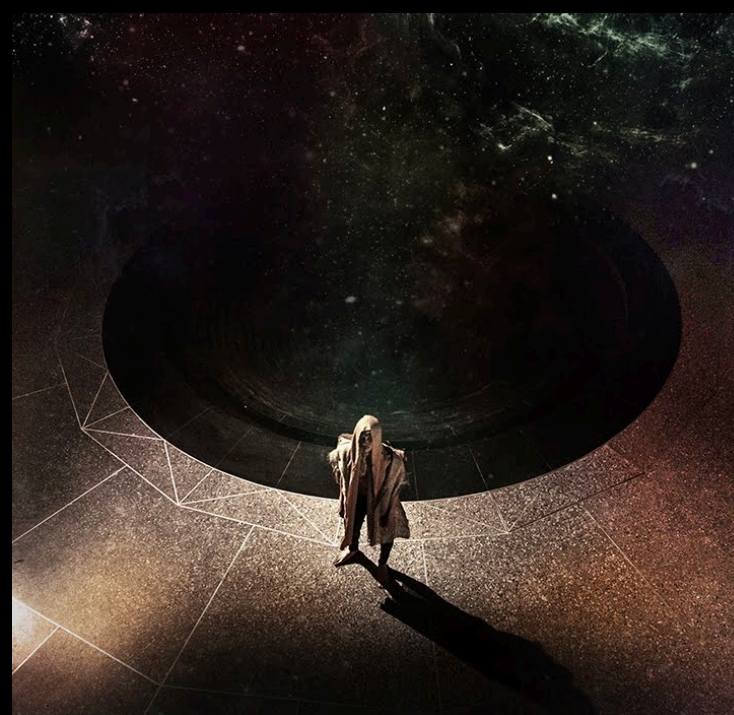
11 → G

00111101110100011100111100011011111101001111101010111000011111001000000001100
100011110010010101110100100111100100011010011100011011000100011101011111001
000100111111001010101010011101101101100000011111000010000100000000110101101
0110111111010000001110100001001011100111010011100011101101001110111111100110
0000101111000010001110011000010001010000111011010111000110001001110010001
10011001101101101111000111010110101101101101010010111111011100110011110011001
11110101101000000100010001110000111011100001111000100111110100010000000110
01110110011101000011010100110000100100110011100101111000001100010101011101
00000111010011000000000010110000010001001001001110000111000010011010101
1100111111101010000010110001111111000101110000111011010011101000000111110111
11010101101110010011010001100100010001100110101001110010000101001101010001
101101110110001001010101111110000010001110001101111101001101000001101001011
001010001110000110101100111101000110101111000001111001001010000101111001111
0101100111111111011100101001000110110101111111101100110001001011100111111101110
00010001011100100110010101011011101001000010101011110011111111000001011100
000110010101000101100101100000010001010000001010101010101111100011110000
01101001100000110000010110011111111110011010001010111011101010100101100000
0011011100100

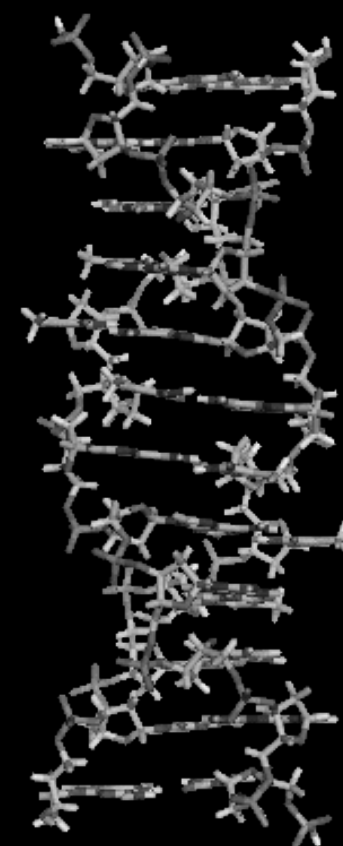
AGGTGTATGAGGATCGGGTAGGGCCCGCATGGACAAAGACAGGACTTTGTACTGCTATCCTG
ATCGATATGTTGGACACTGGCTTTTTTAGCGTCGAAAGGCATAACAAAGTTCCCGTGGCCAAT
GTAAC TTGAGCCTGATGTCCTGTGGGAGAATTGCATATGAGAACACCATGTCCGCAGATAG
CTATCTCTCGTCGGATGTTCCGTCGT TACGGGTGAGAGGAGAGGCCGTAAACACAGCATGT
GAAGGATAGGCCACAATCTGTCTGTAAGTTAGAACTAGAGCTTGCAAGATTTTGTAAATGTAG
AAAAACGAATATACTAGCATGAACTCCCGCTGGGTTAATTCAGGGGCACGCATGTCCTGTAA
AGGCGGGCCCGTGACTCCAGACACAGAGTTAGCTAACCTCCCAGTCGCGGATACCCGGGAAT
ATGATCGGGCCTCCAAGTACGACCAGCATCCGAGGTATCCGGAATGCTACCATTGCTGCCG
AGGGGTGACCTATCGTTGGGTCTCACTTGAGGGGTGAACACGCTAGACCCGTGTACATTTT
GCTGGGCAACGCAAGACCCACGACGAAACACCAATTTTTTTTGGATGCAAGTAGAATCAAC
GAGGGGGAGTATTGTGTTTACGAAATCGCTA



Final Product: 640 Base Pairs



The Dream and
the Universe



Encoded DNA

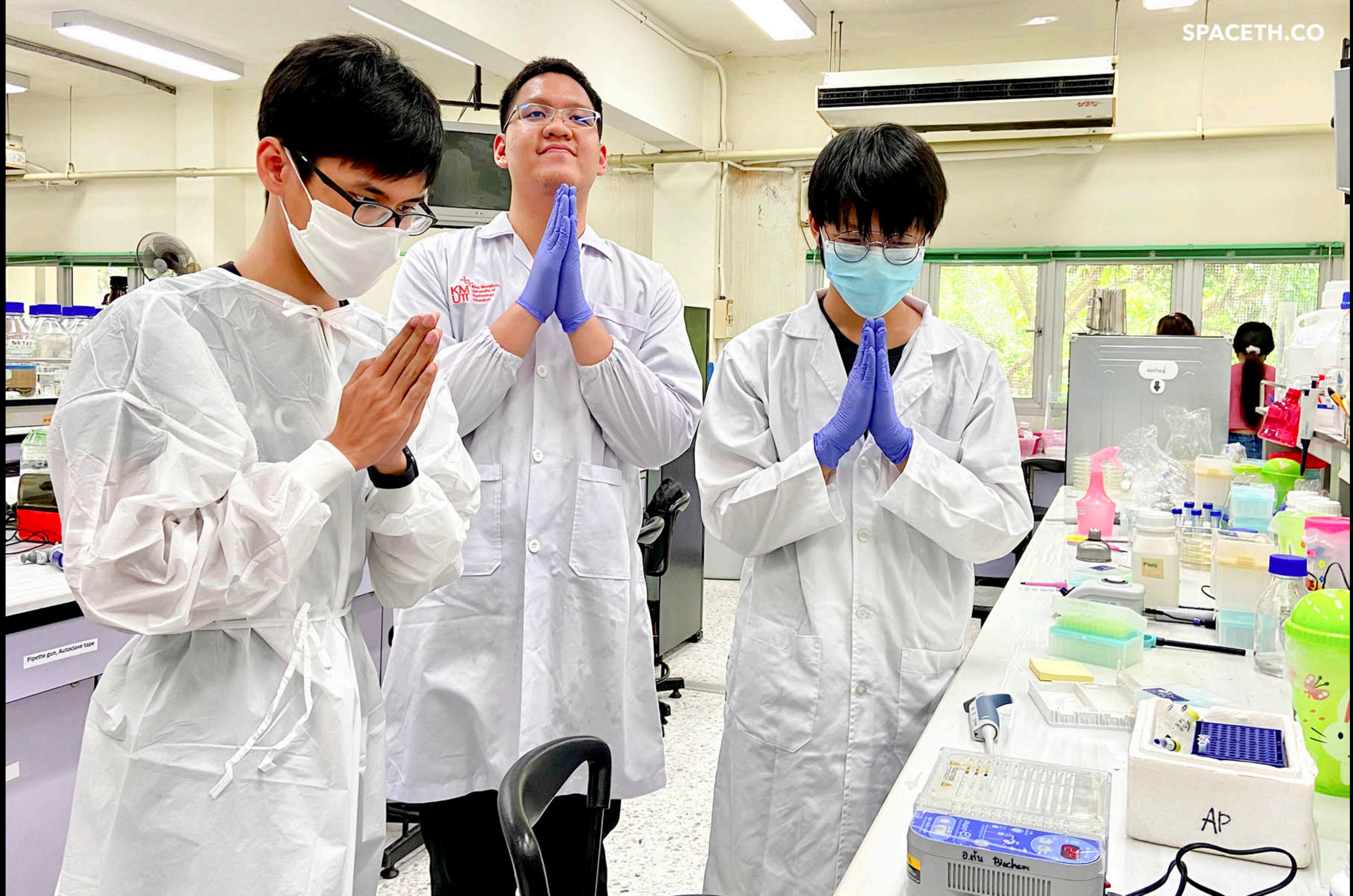


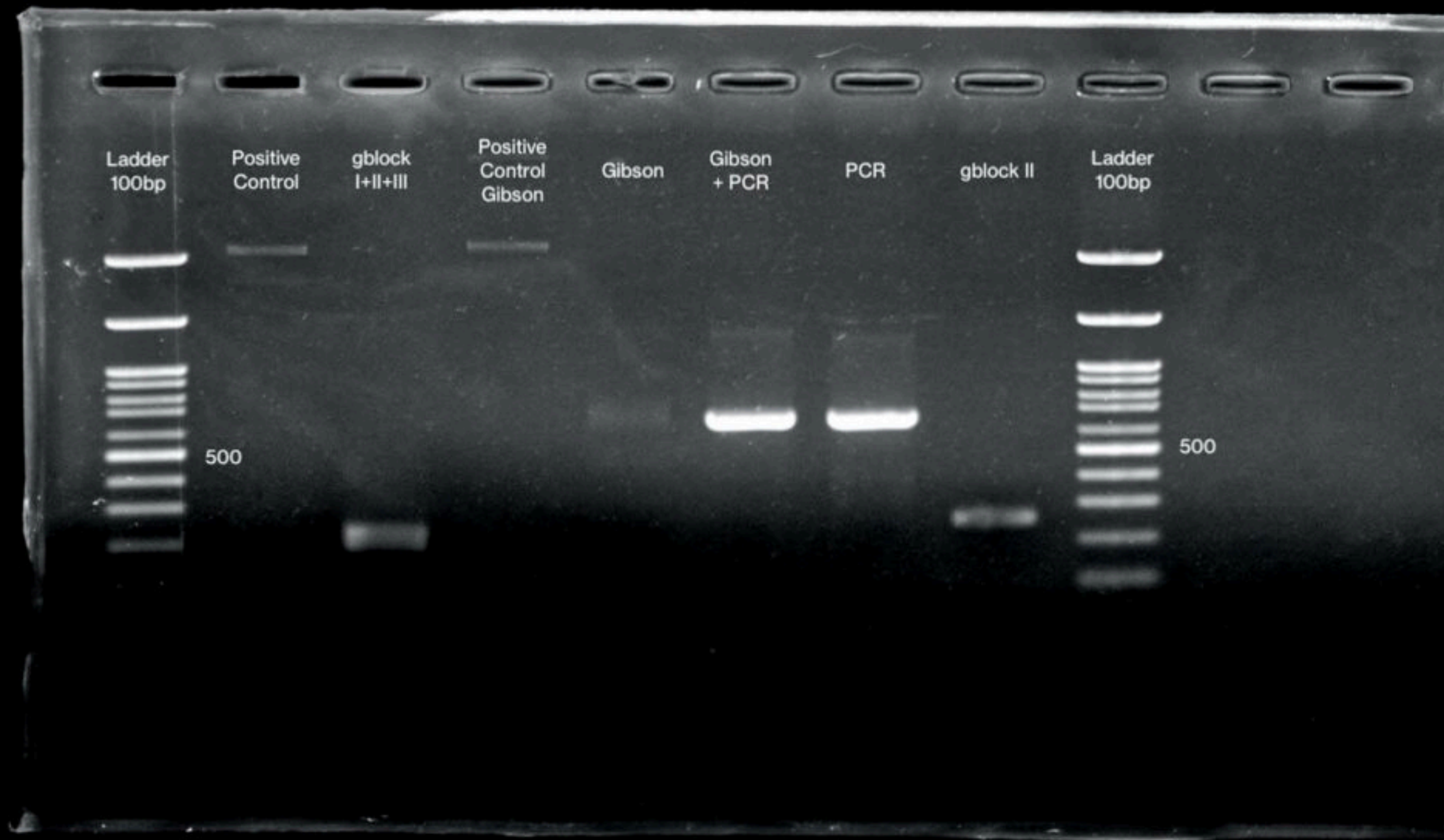
Launch to
Space

Waiting for the
Package to be
shipped back to
Thailand



Researches and
Experiments

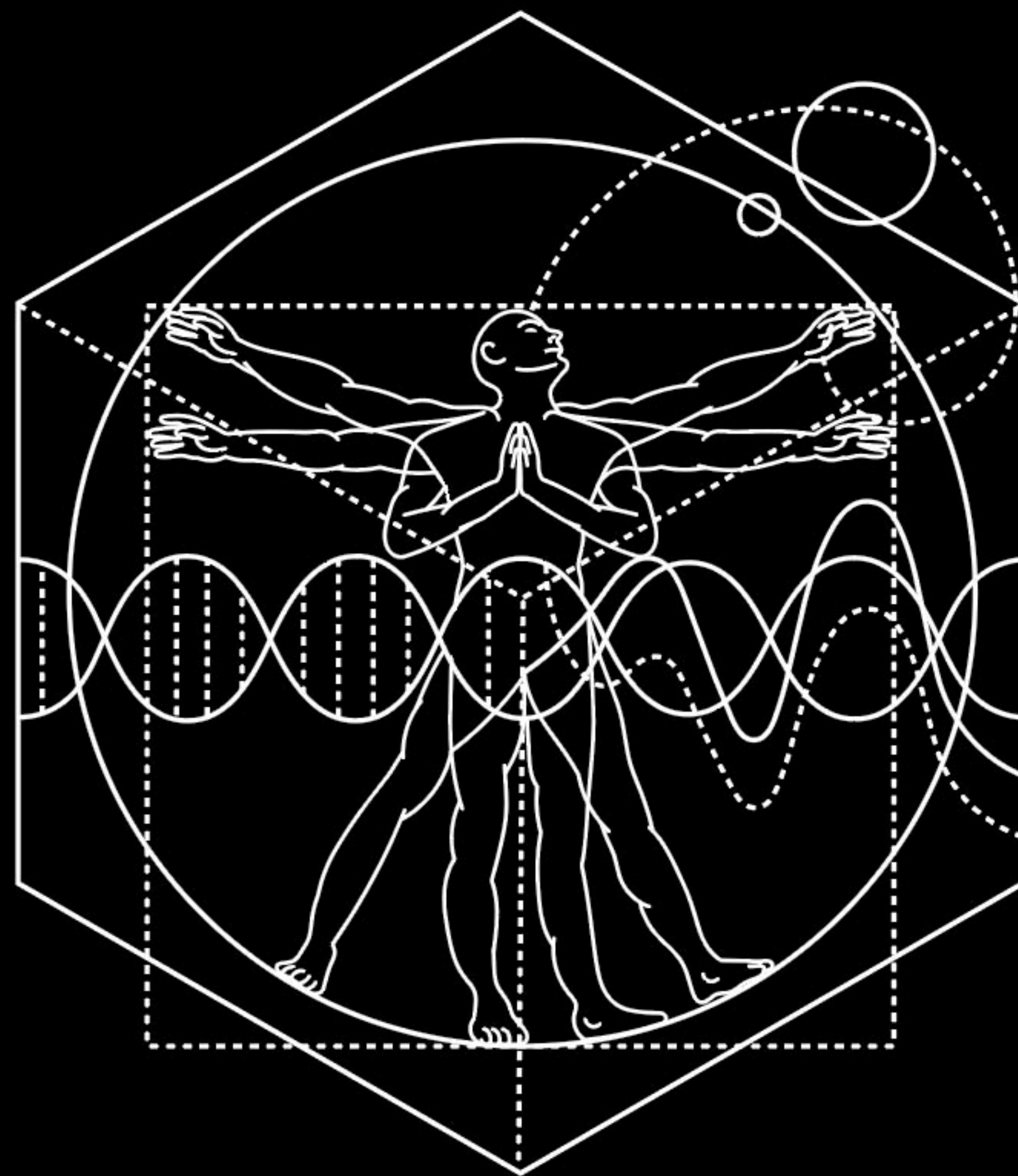






What's now





Source: github.com/spaceth/goldenrecord

Thank you very much