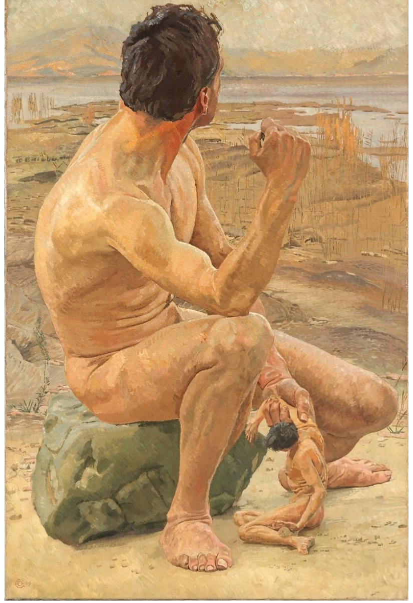


CS61B

Lecture 39: Compression, Complexity, and P = NP

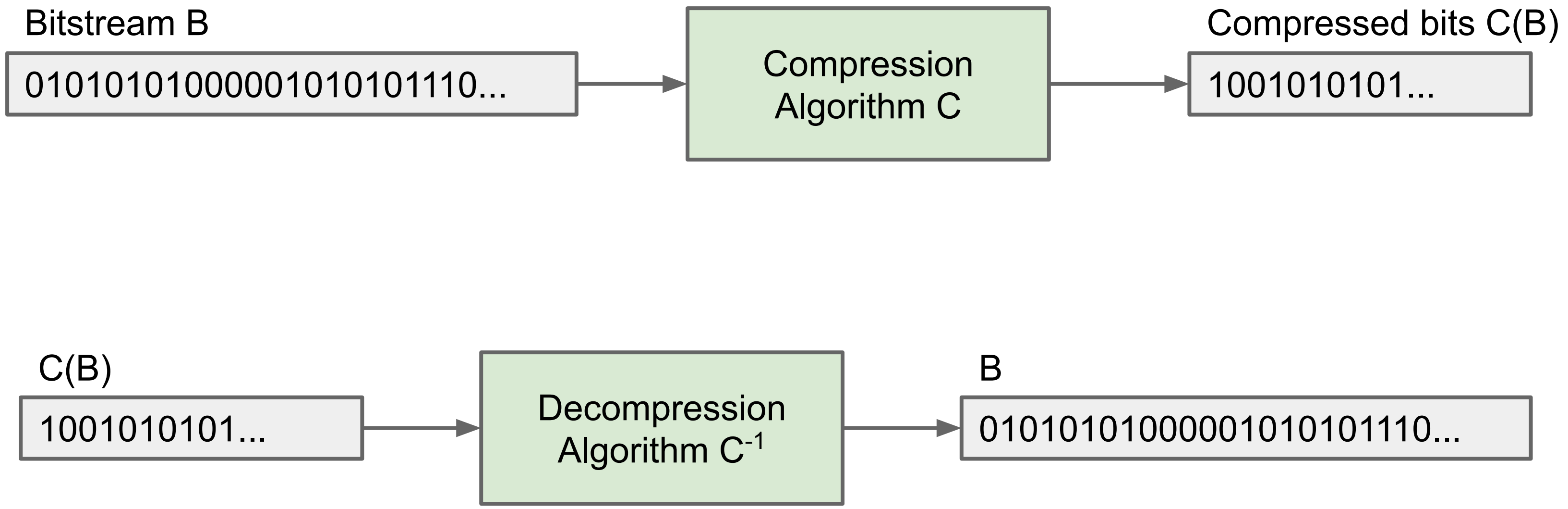
- Models of Compression
- Kolmogorov Complexity (extra)
- Space / Time Compression (extra)
- P=NP? (Extra)
- Is Short = Comprehensible? (Extra)

datastructur.es

Another Look at Model 1 vs. Model 2 for Compression

datastructur.es

Last Time: Compression



We saw one technique for compression called Huffman Coding.

- There are many other approaches to compression.
- Interesting question: What is the best way to compress a given bitstream?

Comparing Compression Algorithms

Example: What is the best way to compress mobydick.txt?

- One way to approach this problem: Try a bunch of different standard tools and see which yields the smallest size.

Algorithm	Uncompressed size (bits)	Compressed size (bits)
zip	5145656	2091000
huffman	5145656	3412896
bzip2	5145656	1805288

Comparing Compression Algorithms

Example: What is the best way to compress mobydick.txt?

- One way to approach this problem: Try a bunch of different standard tools and see which yields the smallest size.

Algorithm	Uncompressed size (bits)	Compressed size (bits)
zip	5145656	2091000
huffman	5145656	3412896
bzip2	5145656	1805288

One problem: What if someone writes a custom compression algorithm?

- If input is 0, return the entire text of Moby Dick.
- If input is 1, return the entire text of Great Expectations.
- For all other inputs, it just ignores the top bit and returns the rest.

Ultra Compression

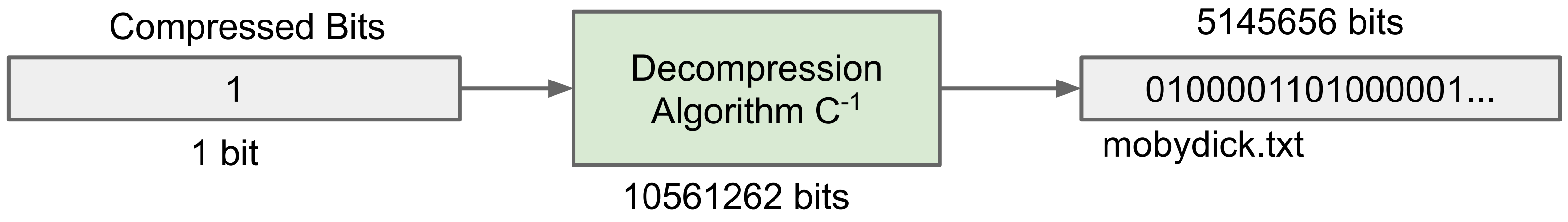
```
public static String greatmobydecompress(Bitstream bs) {
    if (bs.toInt() == 0) {
        return "CALL me Ishmael. Some years ago never mind how...";
    }
    if (bs.toInt() == 1) {
        return "It was the best of times, it was the worst of times...";
    }
    return bs.dropFirstBit().toString();
}
```

Algorithm	Uncompressed size (bits)	Compressed size (bits)
zip	5145656	2091000
huffman	5145656	3412896
bzip2	5145656	1805288
greatmobydecompress	5145656	1

A Flaw in Compression Model #1

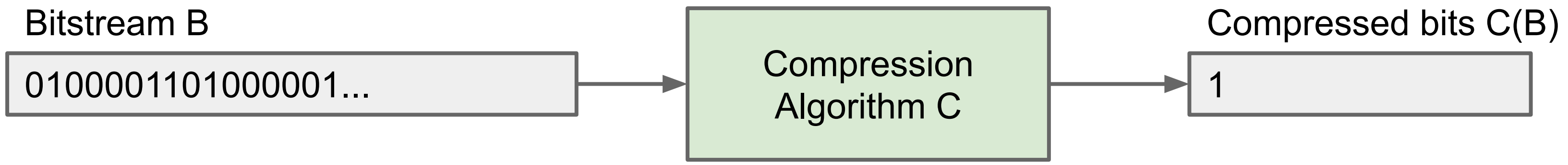
Source code for decompression algorithm itself might be highly complex.

- To avoid this issue, we can include the number of bits for the source code of the algorithm itself.

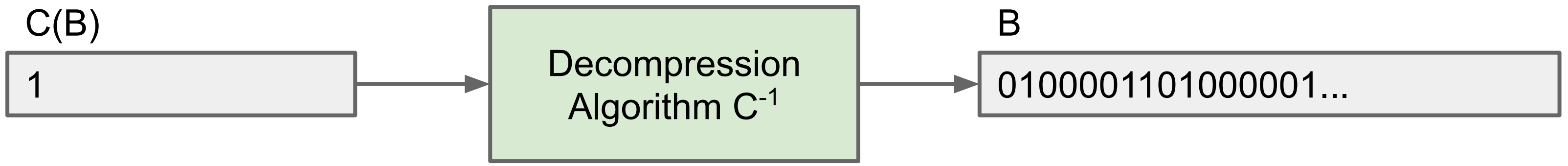


Compression Models #1 and #2

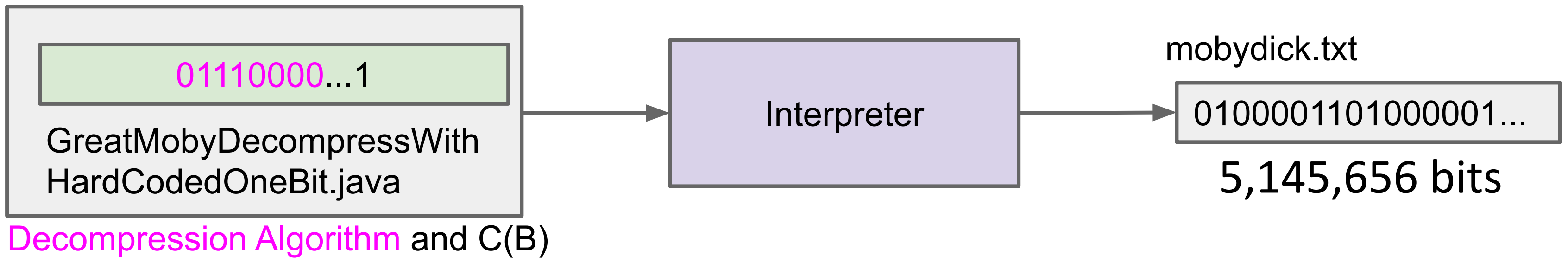
Compression



Decompression Model #1



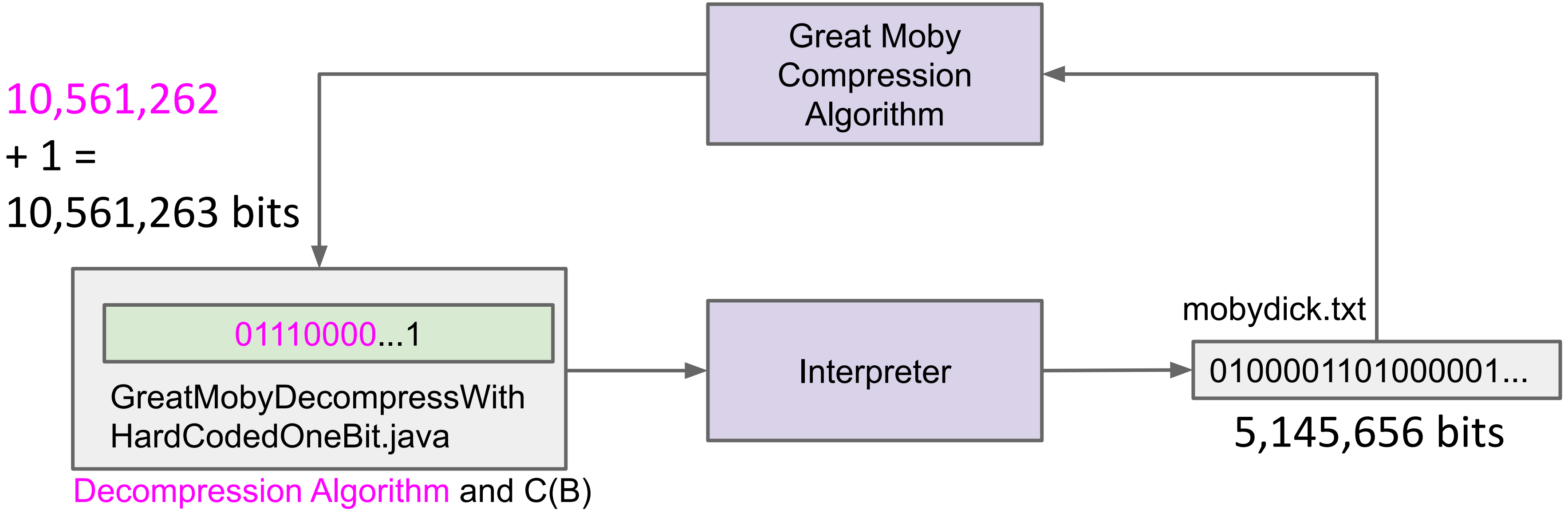
Decompression Model #2: Include the code for the decompression algorithm as part of the compressed output.


01110000...1
GreatMobyDecompressWithHardCodedOneBit.java
Decompression Algorithm and C(B)
Interpreter
mobydick.txt
0100001101000001...
5,145,656 bits

Compression Model 2

The goal of a compression algorithm is to find short sequences of bits that generate desired longer sequences of bits.

- Given a sequence of bits B, find a shorter sequence DA+C(B) that produces B when fed into an interpreter.



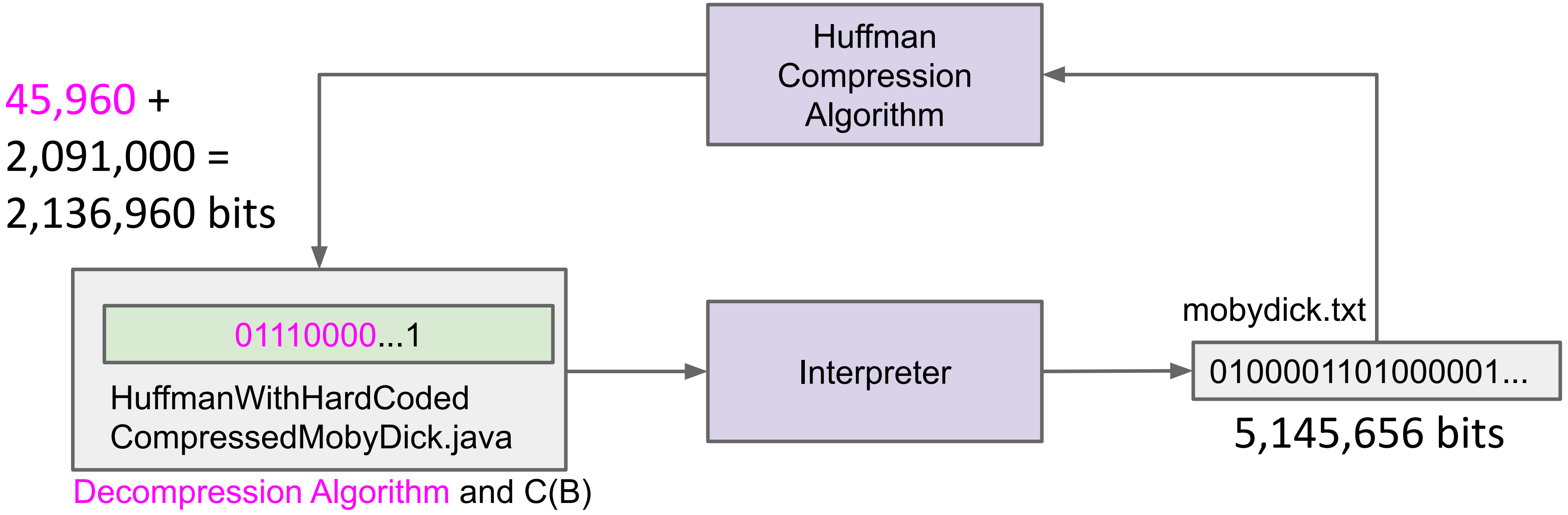
10,561,262
+ 1 =
10,561,263 bits
Great Moby Compression Algorithm
01110000...1
GreatMobyDecompressWithHardCodedOneBit.java
Decompression Algorithm and C(B)
Interpreter
mobydick.txt
0100001101000001...
5,145,656 bits

datastructur.es

Compression Model 2

The goal of a compression algorithm is to find short sequences of bits that generate desired longer sequences of bits.

- Given a sequence of bits B, find a shorter sequence DA+C(B) that produces B when fed into an interpreter.



45,960 +
2,091,000 =
2,136,960 bits
Huffman Compression Algorithm
01110000...1
HuffmanWithHardCoded CompressedMobyDick.java
Decompression Algorithm and C(B)
Interpreter
mobydick.txt
0100001101000001...
5,145,656 bits

datastructur.es

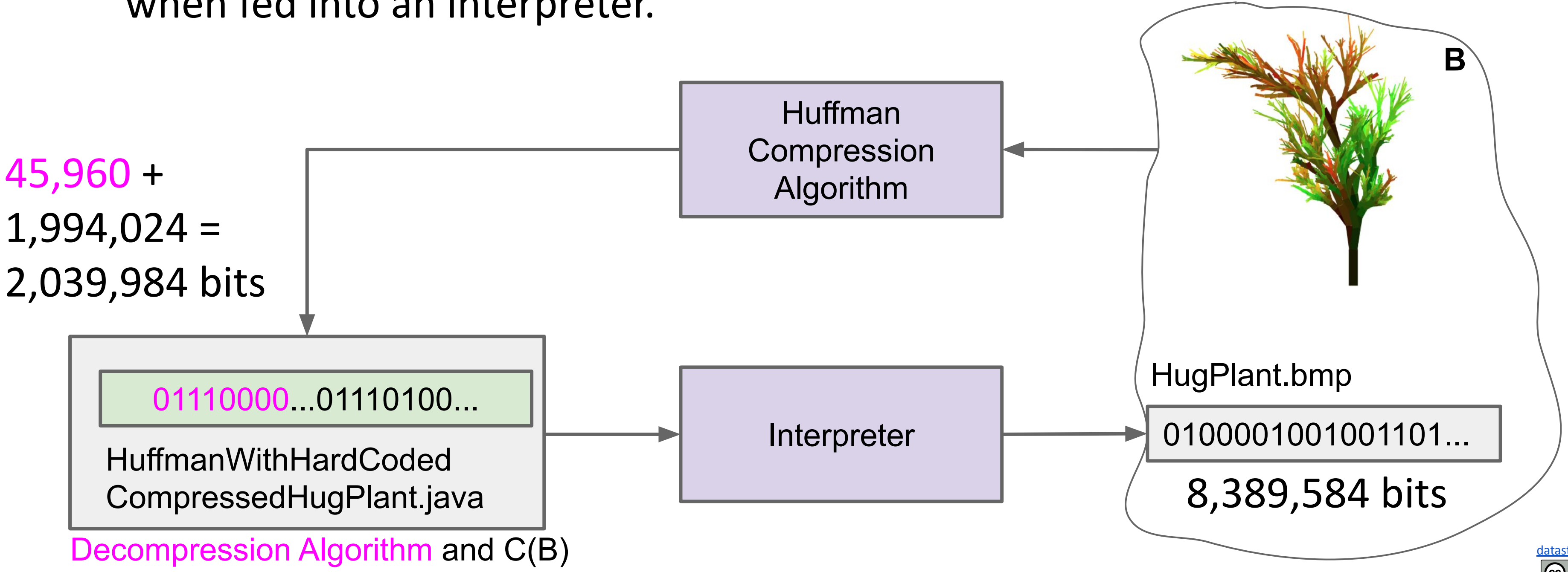
Model 1 vs. Model 2 Compression

Algorithm	Uncompressed size (bits)	Compressed size (bits)	Compressed size using model 2 (bits)
zip	5145656	2,091,000	2,091,000 + 67,160
huffman	5145656	3,412,896	2,091,000 + 45,960
bzip2	5145656	1,805,288	2,091,000 + 74,984
greatmobydecompress	5145656	1	1 + 10,561,262

Compression Model 2

The goal of a compression algorithm is to find short sequences of bits that generate desired longer sequences of bits.

- Given a sequence of bits B, find a shorter sequence DA+C(B) that produces B when fed into an interpreter.

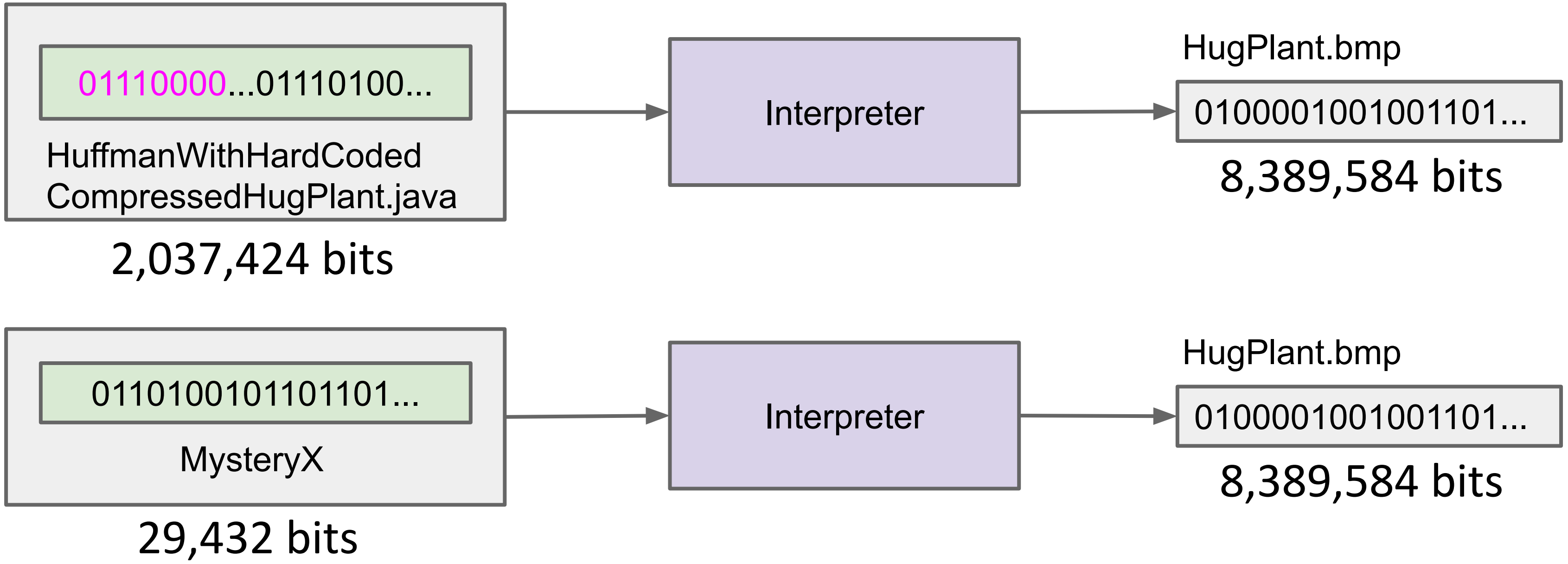


datastructur.es

Even Better Compression

Compression ratio of 25% is certainly very impressive, but we can do much better. MysteryX achieves a 0.35% compression ratio!

- Of the $2^{8389584}$ possible bit streams of length 8389584, only one in $2^{8360151}$ can be generated by our interpreter using an input of length 29,432 bits.

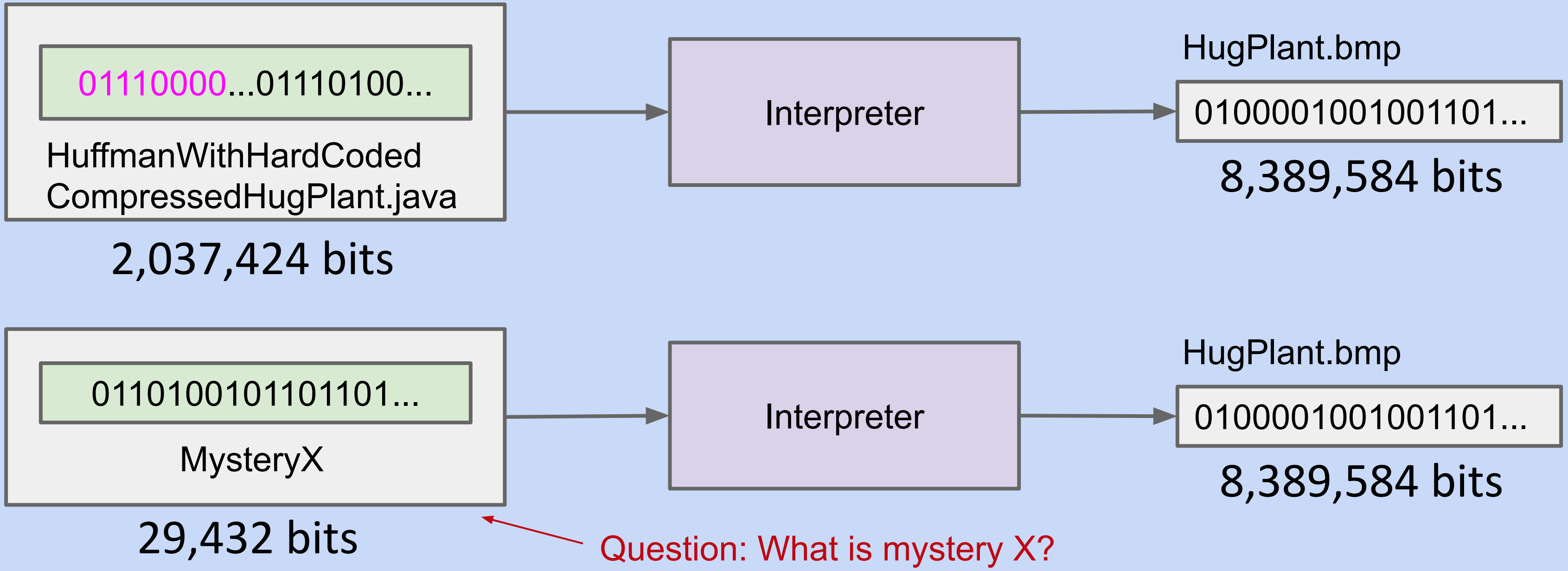


datastructur.es

Even Better Compression

Compression ratio of 25% is certainly very impressive, but we can do much better. MysteryX achieves a 0.35% compression ratio!

- Of the $2^{8389584}$ possible bit streams of length 8389584, only one in $2^{8360151}$ can be generated by our interpreter using an input of length 29,432 bits.

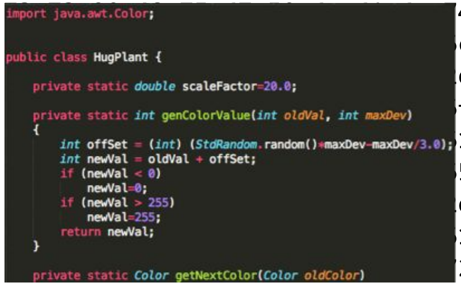


datastructur.es

MysteryX: HugPlant.java

MysteryX is just HugPlant.java, the piece of code that I used to generate the .bmp file originally.

```
69 6d 70 6f 72 74 20 6a 61 76 61 2e 61 77 74 2e
43 6f 6c 6f 72 3b 0a 0a 0a 70 75 62 6c 69 63 20
63 6c 61                                  4 20 7b
0a 09 2f                                  c 20 74
6f 20 70                                  e 75 67
20 74 6f                                  f 75 72
20 70 72                                  1 74 65
20 73 74                                  5 20 73
63 61 6c                                  e 30 3b
0a 0a 09                                  1 74 69
63 20 69                                  2 56 61
6c 75 65 28 69 6e 74 20 6f 6c 64 56 61 6c 2c
```



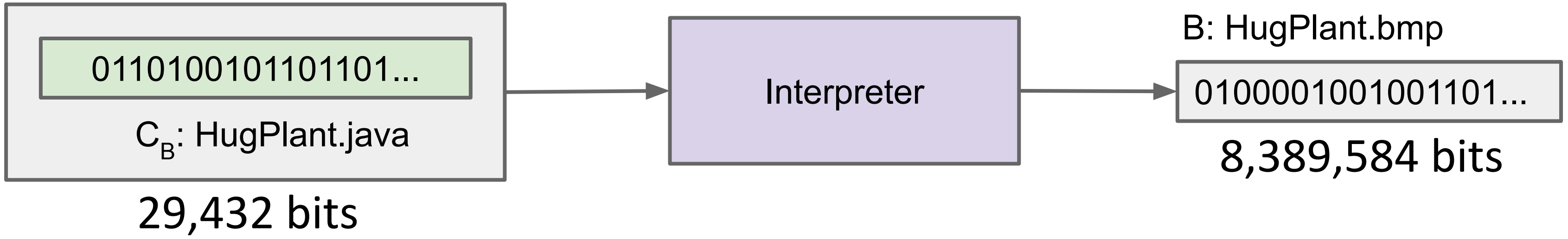
```java
import java.awt.Color;

public class HugPlant {

    private static double scaleFactor=20.0;

    private static int genColorValue(int oldVal, int maxDev)
    {
        int offSet = (int) (StdRandom.random()*maxDev-maxDev/3.0);
        int newVal = oldVal + offSet;
        if (newVal < 0)
            newVal=0;
        if (newVal > 255)
            newVal=255;
        return newVal;
    }

    private static Color getNextColor(Color oldColor)
```

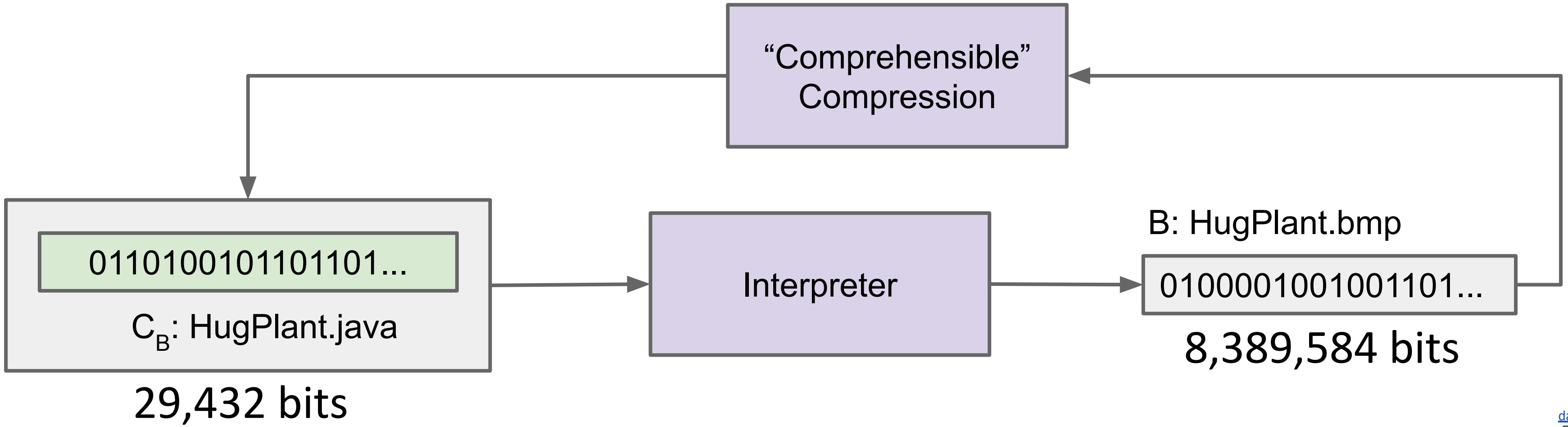


datastructur.es

Question #1: Comprehensible Compression

Interesting question #1:

- Can we create a “comprehensible” compression algorithm that takes as input a target bitstream B, and outputs useful, readable Java code?



datastructur.es

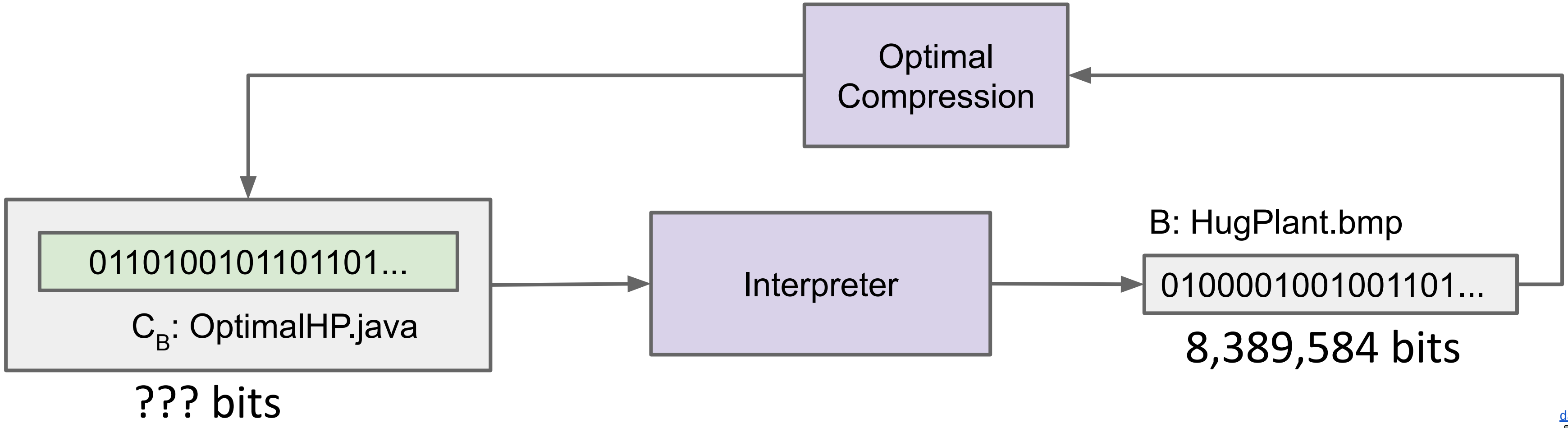
Question #2: Optimal Compression

Interesting question #2:

- Can we create an optimal compression takes as input a target bitstream B, and outputs the shortest possible Java program that outputs this bitstream?

Seems plausible that this optimal program would also have structure.

- The answer turns out to be deep!



datastructur.es

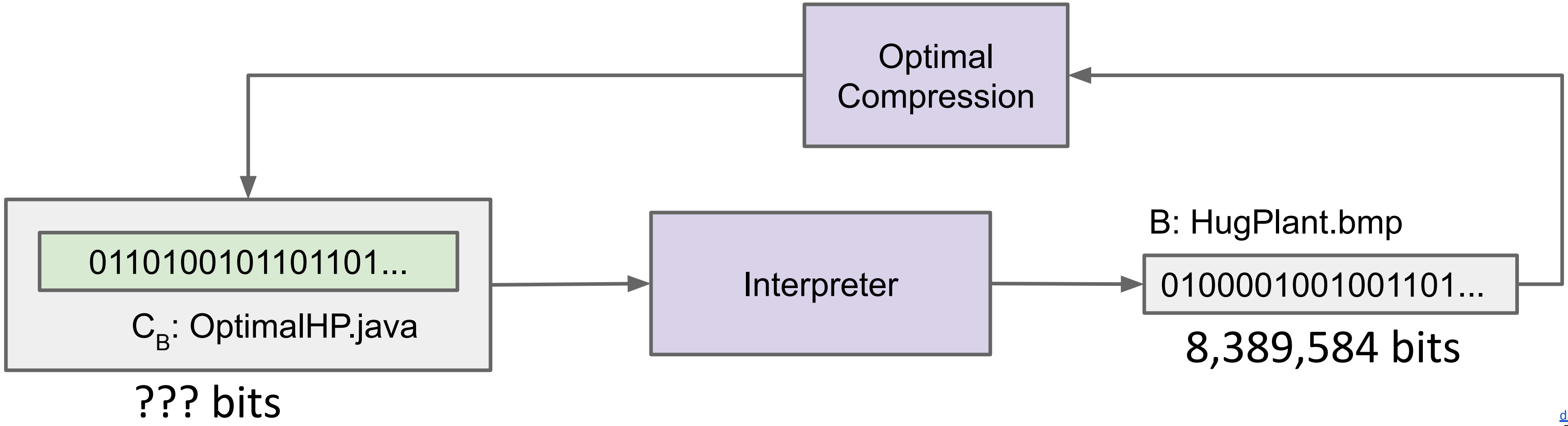
Optimal Compression and Kolmogorov Complexity (Extra - CS172 Preview)

datastructur.es

Kolmogorov Complexity

Given a target bitstream B, what is the shortest bitstream C_B that outputs B.

- Definition: The Java-Kolmogorov complexity $K_J(B)$ is the length of the shortest Java program (in bytes) that generates B.
  - Example: K_J(HugPlant.bmp) would be the length of OptimalHP.java.
  - There IS an answer. It just might be very hard to find.



datastructur.es

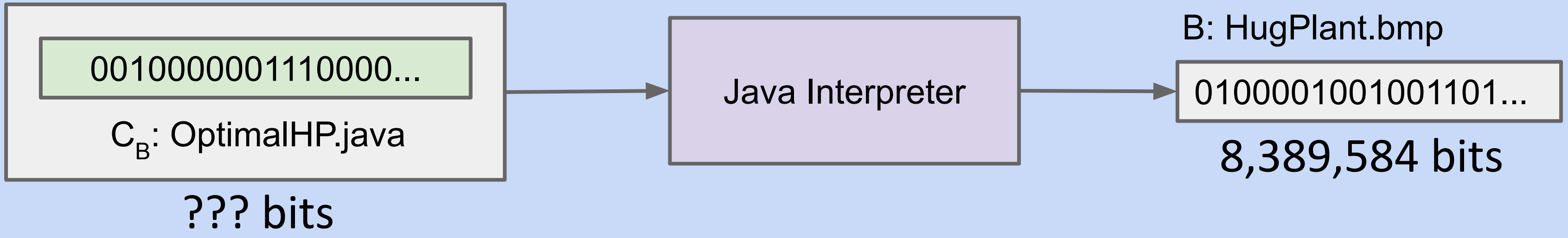
Kolmogorov Complexity

Given a target bitstream B, what is the shortest bitstream C_B that outputs B.

- Definition: The Java-Kolmogorov complexity $K_J(B)$ is the length of the shortest Java program (in bytes) that generates B.
  - There IS an answer. It just might be very hard to find.

Fact #1: Kolmogorov Complexity is effectively independent of language.

- For any bit stream, the Java-Kolmogorov Complexity is no more than a constant factor larger than the Python-Kolmogorov Complexity.
  - Why?



datastructur.es

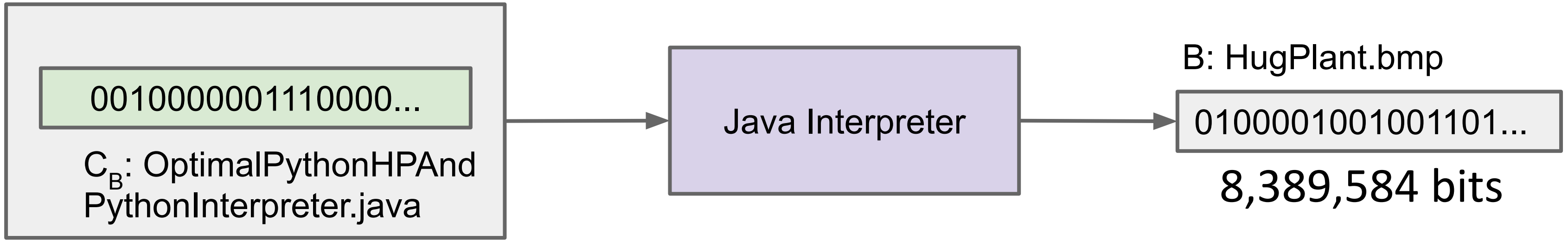
Kolmogorov Complexity (Language Independence)

Fact #1: Kolmogorov Complexity is effectively **independent** of language.

- For any bit stream, the Java-Kolmogorov Complexity is no more than a constant factor larger than the Python-Kolmogorov Complexity.
  - Why?

Paul Hilfinger writes a Python program that is very short and uses weird Python features and I am stumped and cannot think of a similar thing in Java.

- I could just write a Python interpreter in Java and then run Paul's program.
  - $K_J(B) \leq K_P(B)$ + size(python interpreter)



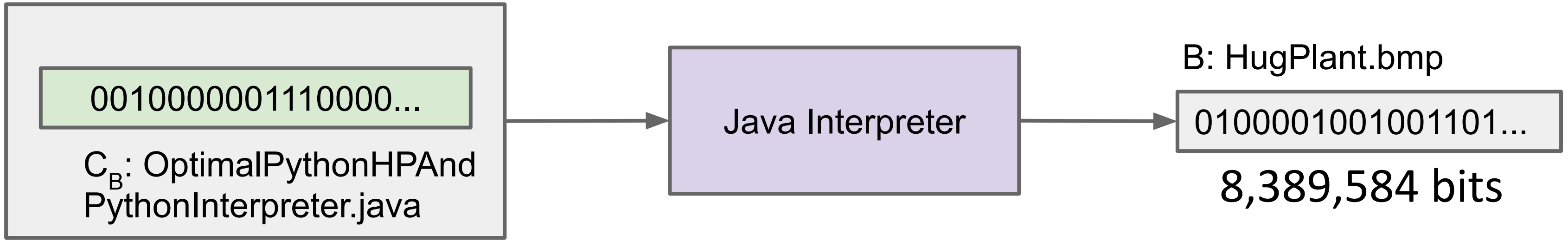
datastructur.es

Kolmogorov Complexity (Language Independence)

Fact #1: Kolmogorov Complexity is effectively **independent** of language.

This is a deep fact!

- It means that most bitstreams are fundamentally incompressible no matter what programming language we use for our compression algorithm.
- Example: For all possible compression algorithms in all possible programming languages, a completely random sequence of 1,000,000 bits has at best, a 1 in 2^{499999} chance of being compressed by 50%.

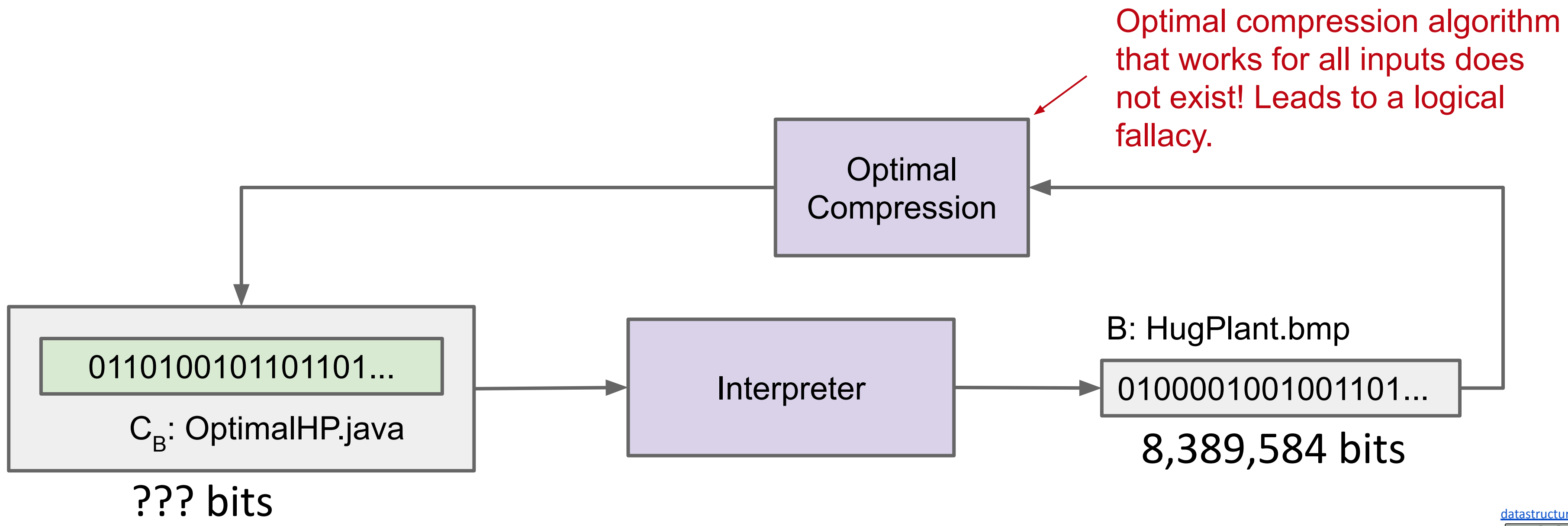


datastructur.es

Kolmogorov Complexity (Uncomputability)

Fact #2: It is **impossible** to write a program that even calculates the Kolmogorov Complexity of any bitstream. Proof available here.

- Corollary: If we can't even compute the length of the shortest program, it is also **impossible** to write the "perfect" compression algorithm.



datastructur.es

Space/Time Bounded Compression (Extra)

datastructur.es

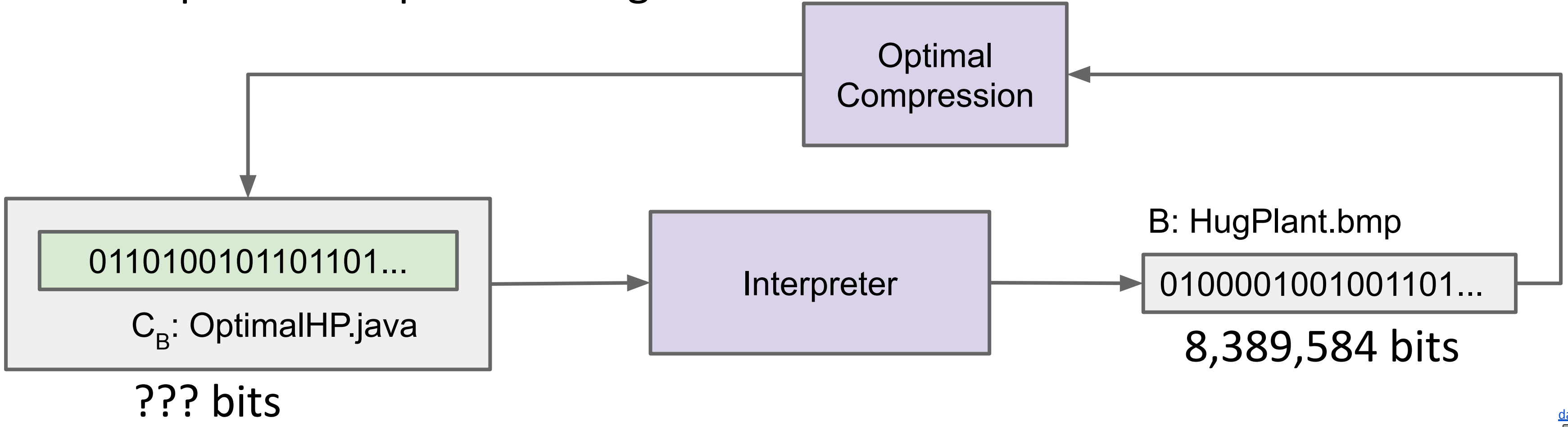
Question #2: Optimal Compression

Interesting question #2:

- Can we create an optimal compression algorithm takes as input a target bitstream B, and outputs the shortest possible Java program that outputs this bitstream?

Unfortunately the answer is no. This is not possible, even theoretically.

- No "optimal compression" algorithm exists.



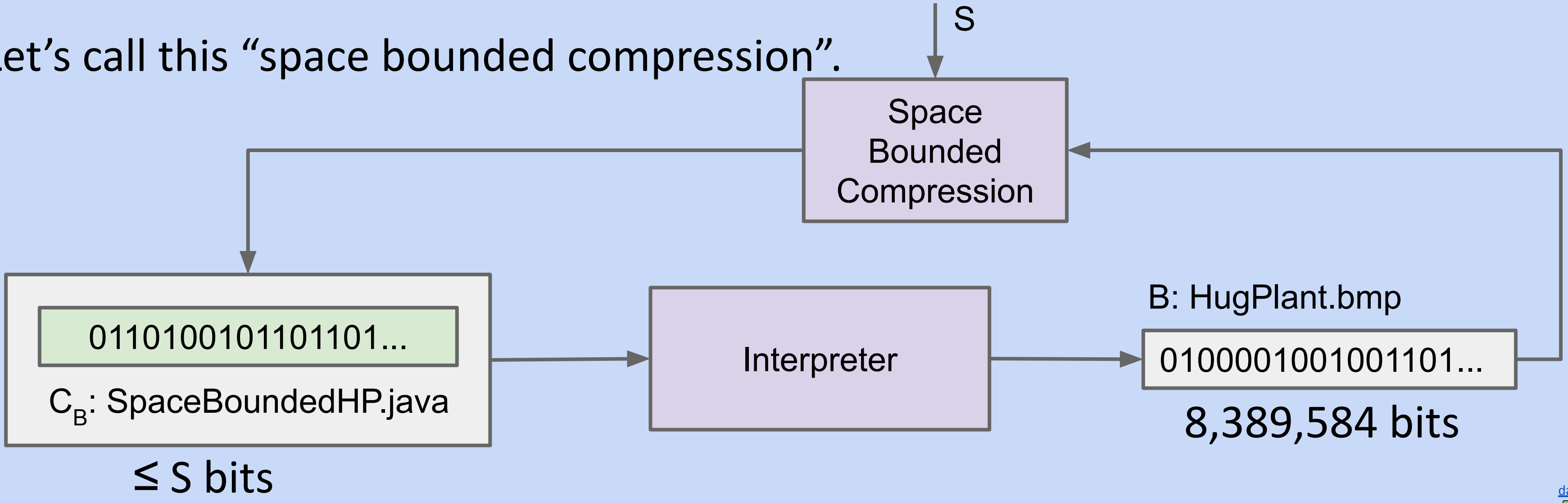
datastructur.es

Question #2S: Space Bounded Compression

Interesting question #2S: Can we create a compression algorithm that:

- Takes two inputs:
  - A target bitstream B.
  - A size S.
- and outputs a Java program of length ≤ S that outputs B?

Let's call this "space bounded compression".



datastructur.es

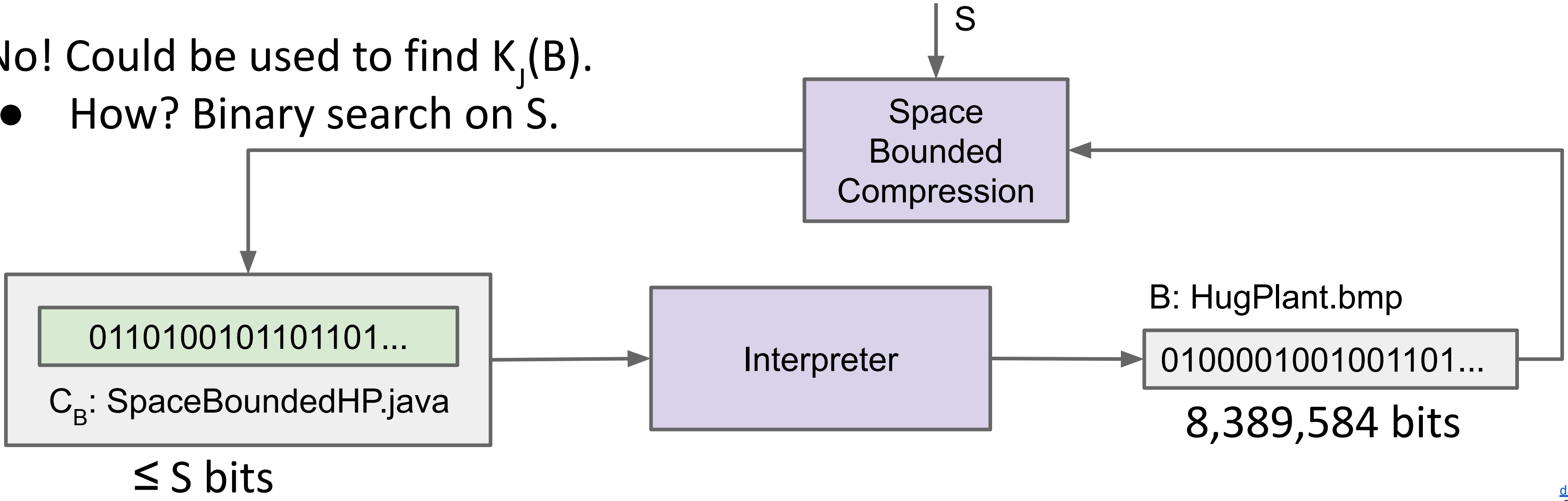
Question #2S: Space Bounded Compression

Interesting question #2S: Can we create a compression algorithm that:

- Takes two inputs:
  - A target bitstream B.
  - A size S.
- and outputs a Java program of length ≤ S that outputs B?

No! Could be used to find $K_J(B)$.

- How? Binary search on S.

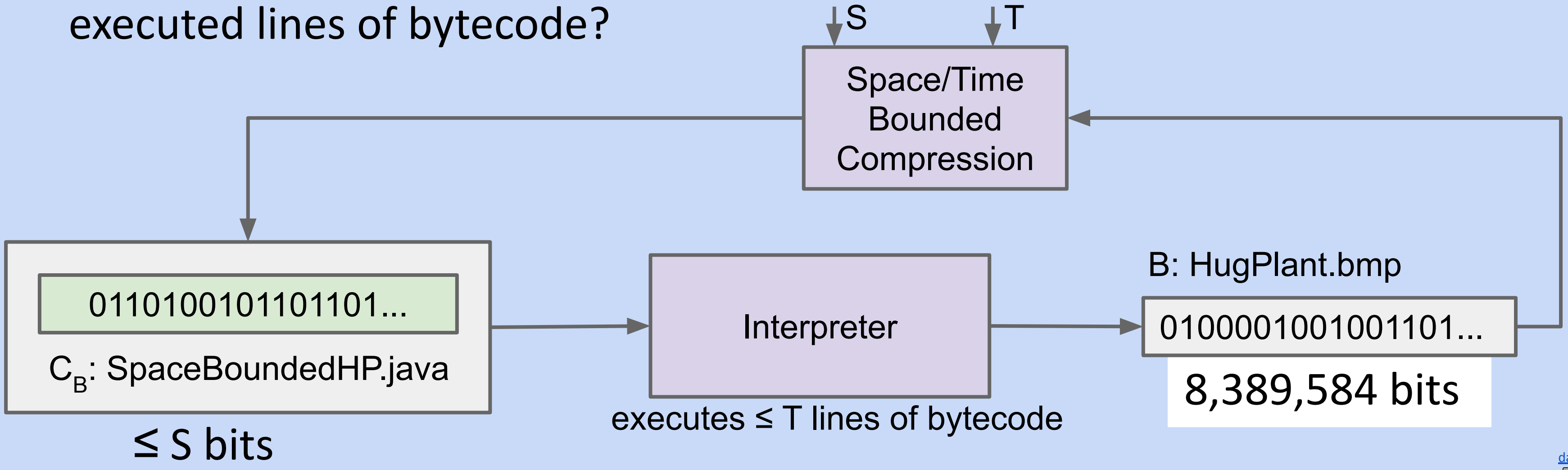


datastructur.es

Question #2ST: Space/Time Bounded Compression

Interesting question #2ST: Can we create a compression algorithm that:

- Takes three inputs:
    - A target bitstream B.
    - A size S.
    - A maximum number of lines of bytecode executed T.
- and outputs a Java program of length ≤ S that outputs B in fewer than T executed lines of bytecode?



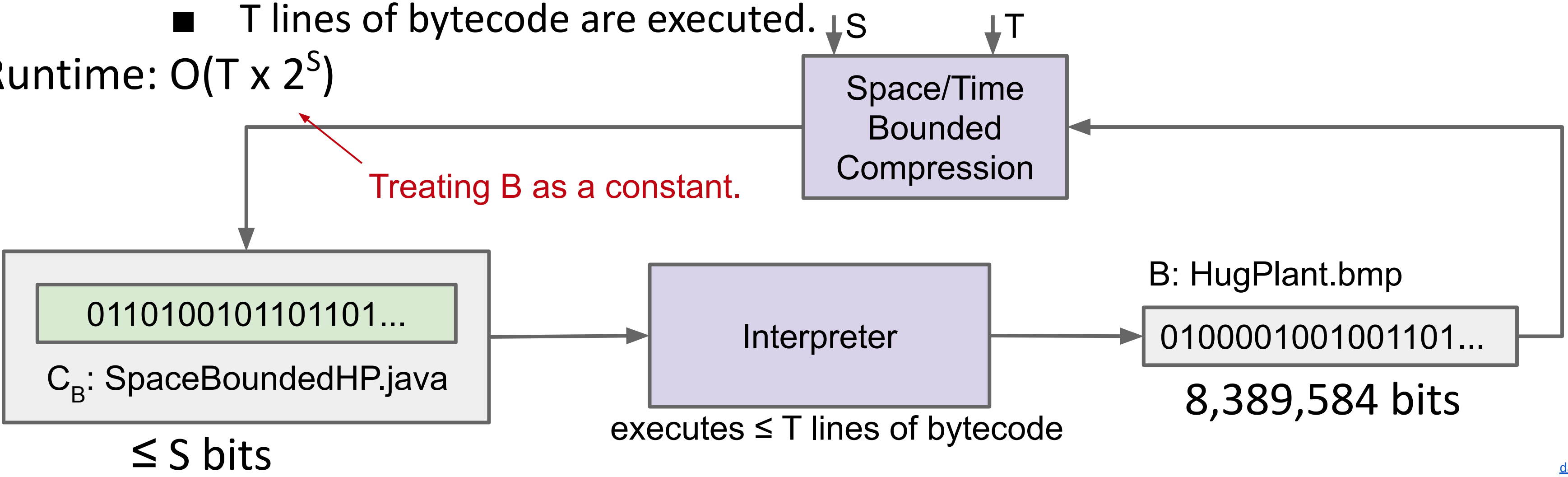
datastructur.es

Question #2ST: Space/Time Bounded Compression

Interesting question #2ST: Can we create a space/time bounded compression algorithm? Yes! And here's an algorithm:

- For each possible program p of length S or less:
    - If p compiles, run program p until either:
        - p terminates.
        - We output B.
        - T lines of bytecode are executed.

Runtime: $O(T \times 2^S)$

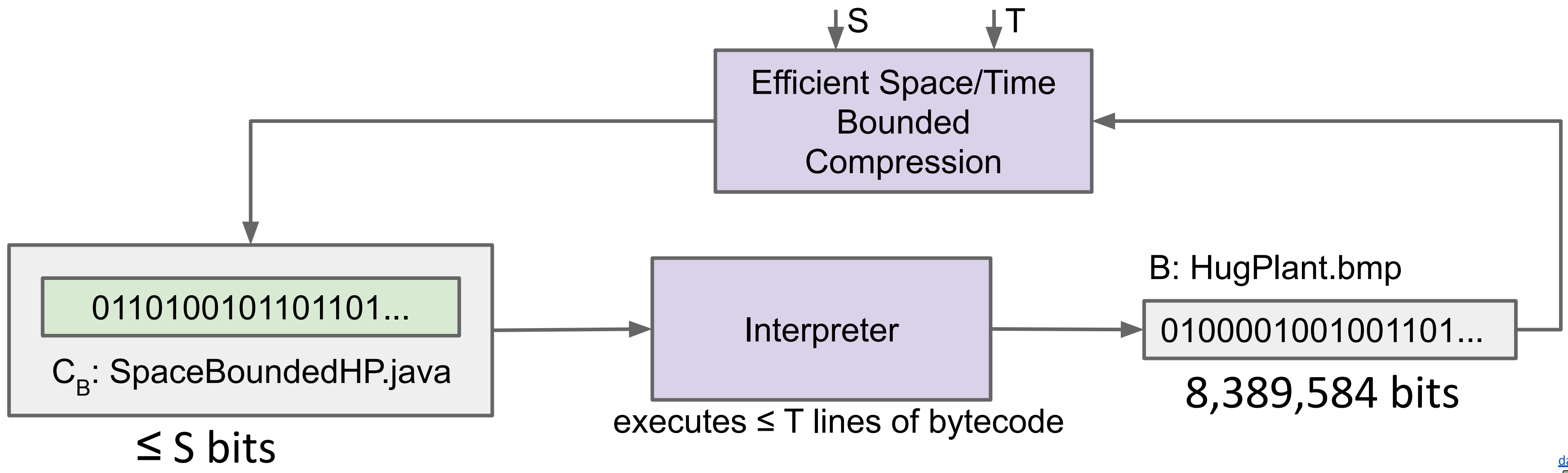

S
T
Space/Time Bounded Compression
Treating B as a constant.
0110100101101101...
C_B: SpaceBoundedHP.java
≤ S bits
Interpreter
executes ≤ T lines of bytecode
B: HugPlant.bmp
0100001001001101...
8,389,584 bits

datastructur.es

Question #2ST-E: Space/Time Bounded Compression

Interesting question #2ST-E: Can we create an **efficient** space/time bounded compression algorithm?

- Need to make a more precise definition of what we mean by "efficient".
- Turns out to be closely related to an important puzzle in computer science: Does P = NP?
  - Will study carefully in 170. But let's take a quick look.



datastructur.es

P = NP? (Extra)

datastructur.es

Surprising Fact

An efficient solution to any of these three problems:

- 3SAT
- Independent Set, a.k.a. INDSET
- Longest Path, a.k.a. LONGEST_PATH

Would also give an efficient space/time bounded compression algorithm.

Why?

- Space/time bounded compression reduces to 3SAT, INDSET, and LONGEST_PATH*.

*: And also tens of thousands of other related problems.

datastructur.es

3SAT and Space/Time Bounded Compression

Example:

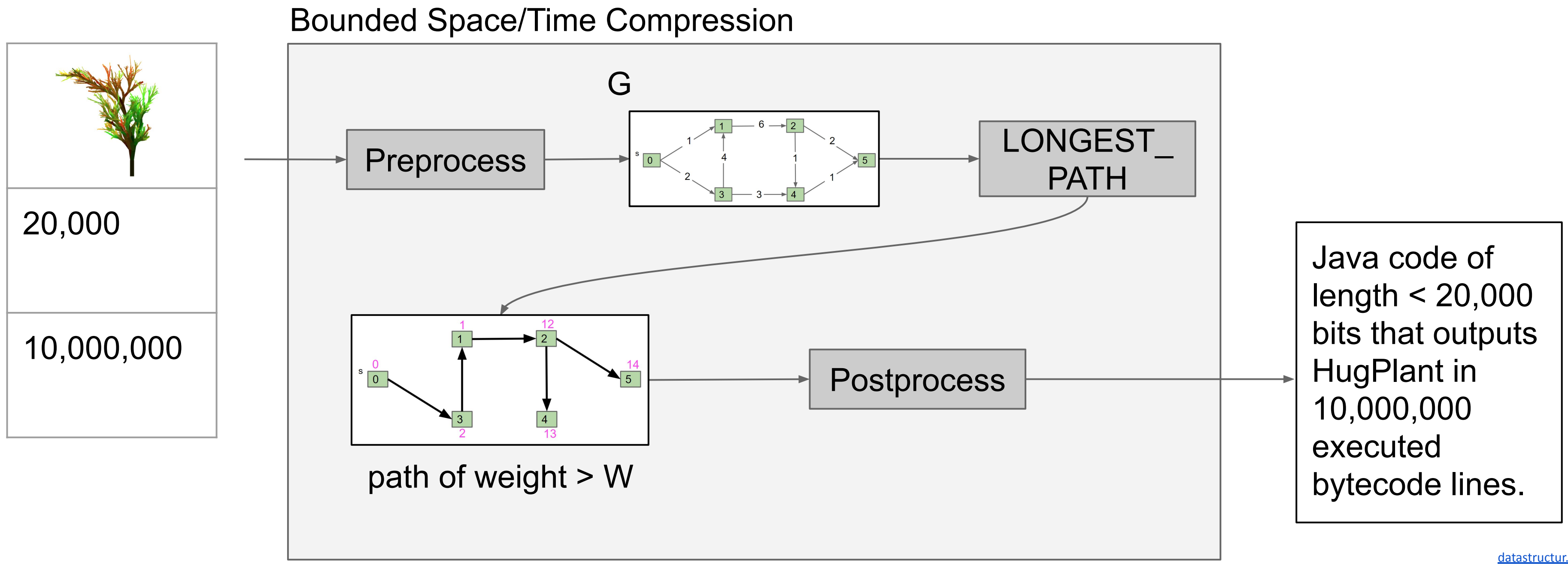
- Let X = “Does there exist a Java program that outputs , and:
    - is of of length 20,000 or less.
    - produces this output in fewer than 1,000,000 executed lines of bytecode.”
- X can be transformed into a longest paths problem (or a 3SAT problem or an independent set problem).

datastructur.es

Visual Reduction

LONGEST_PATH can be used to solve Bounded Space/Time Compression.

- The actual graphs to represent our problem will be phenomenally complex. See 170 if you're curious how this reduction works.

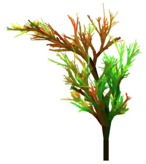


Java code of length < 20,000 bits that outputs HugPlant in 10,000,000 executed bytecode lines.

datastructur.es

3SAT and Space/Time Bounded Compression

Example:

- Let X = “Does there exist a Java program that outputs  , and:
  - is of of length 20,000 or less.
  - produces this output in fewer than 1,000,000 executed lines of bytecode.”
- X can be transformed into a longest paths problem (or a 3SAT problem or an independent set problem).

How do we know X can be turned into a longest paths problem?

- Short answer: “It’s a problem in the complexity class NP and therefore can be reduced to any NP complete problem, including longest paths”.
  - I haven’t introduced many of the terms in this statement. We will very briefly go over them, but too quickly to make complete sense.
- Longer answer: See CS170.

datastructur.es

P = NP?

Two important classes of yes/no problems:

- P: Efficiently solvable problems.
- NP: Problems with solutions that are efficiently verifiable.*

Examples of problems in P:

- Is this array sorted?
- Does this array have duplicates?

Examples of problems in NP:

- Is there a solution to this 3SAT problem?
- In graph G, does there exist a path from s to t of weight > k?

*: Technically it's problems for a which a "yes" answer is efficiently verifiable.

datastructur.es

P = NP?

Two important classes of yes/no problems:

- P: Efficiently solvable problems.
- NP: Problems with solutions that are efficiently verifiable.*

Examples of problems not in NP:

- Is this the best chess move I can make next?
  - Hard to verify.
- What is the longest path?
  - Not a yes/no question.

*: Technically it's problems for a which a "yes" answer is efficiently verifiable.

datastructur.es

Totally Shocking Fact

Every single NP problem reduces to 3SAT.

- This includes Bounded Space/Time Compression.

In other words, **any decision problem for which a yes answer can be efficiently verified** can be transformed into a 3SAT problem.

- This transformation is also “efficient” (polynomial time).

This result is by Cook (1971) and Levin (1973). See Cook-Levin Theorem for more.

Open Question in Computer Science

Question posed Stephen Cook in 1971: Are all NP problems also P problems?

- In other words, are all problems with efficiently verifiable solutions also efficiently solvable?
- Often stated as “Does P = NP?”

One reason to think yes:

- Easy to check any given answer.
  - Maybe with the right pruning rules you can zero in on the answer?

See CS170 for a much more thorough and formal treatment of this problem.

datastructur.es

Almost Like Gods (Extra)

datastructur.es

P = NP?

Consensus Opinion (Bill Gasarch Poll, 2012 poll)

- 83%: P ≠ NP (126 respondents)
- 9%: P = NP (12 respondents)
- 9%: Other (13 respondents)

Why is opinion generally negative?

- Someone would have proved it by now.
  - "The only supporting arguments I can offer are the failure of all efforts to place specific NP-complete problems in P by constructing polynomial-time algorithms." - Dick Karp
- Creation of solutions seems philosophically more difficult than verification.

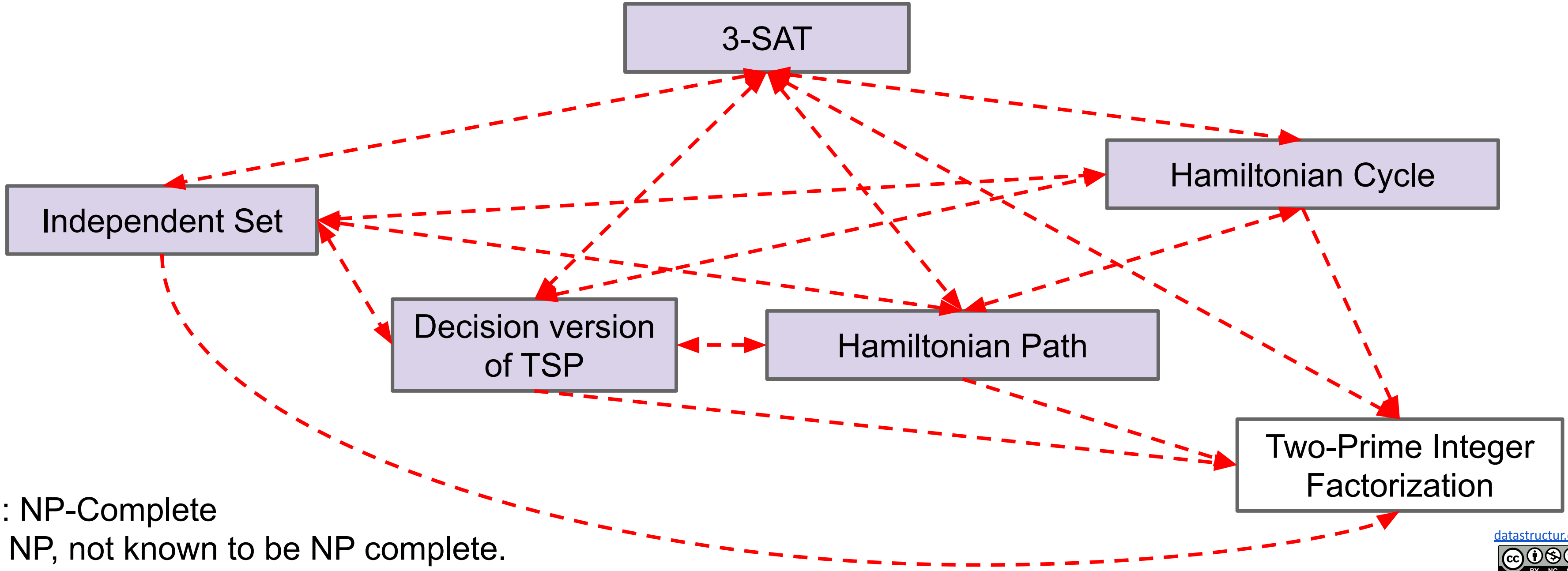
What is that?

datastructur.es

NP Complete Problems

It turns out that there are tons of NP problems that all reduce to each other.

- Solving any of these problems means that you have solved all of them.
- These problems are known as “NP Complete” problems.
    - There are tens of thousands of them, and none have been solved.



Purple: NP-Complete
White: NP, not known to be NP complete.

datastructur.es

Fun Fact: Mathematical Proofs Are in NP!

Example of NP problem: Is there a proof that the Riemann Hypothesis is true?

- A yes answer can be easily verified (we just need to check the proof).

If P=NP, then mathematical proof can be automated!

- P=NP means checking a proof is roughly as easy as creating the proof.
- First observed informally by Kurt Gödel himself.

"[A linear or quadratic-time procedure for what we now call NP complete problems would have] consequences of the greatest magnitude. [For such a procedure] would clearly indicate that, despite the unsolvability of the Entscheidungsproblem, the mental effort of the mathematician in the case of yes-or-no questions could be completely replaced by machines." - *Kurt Gödel*

es

One of These Things, Is Not Like The Others

In 2000, the Clay Mathematics Institute set up $1,000,000 prizes for the solution of each of seven problems.

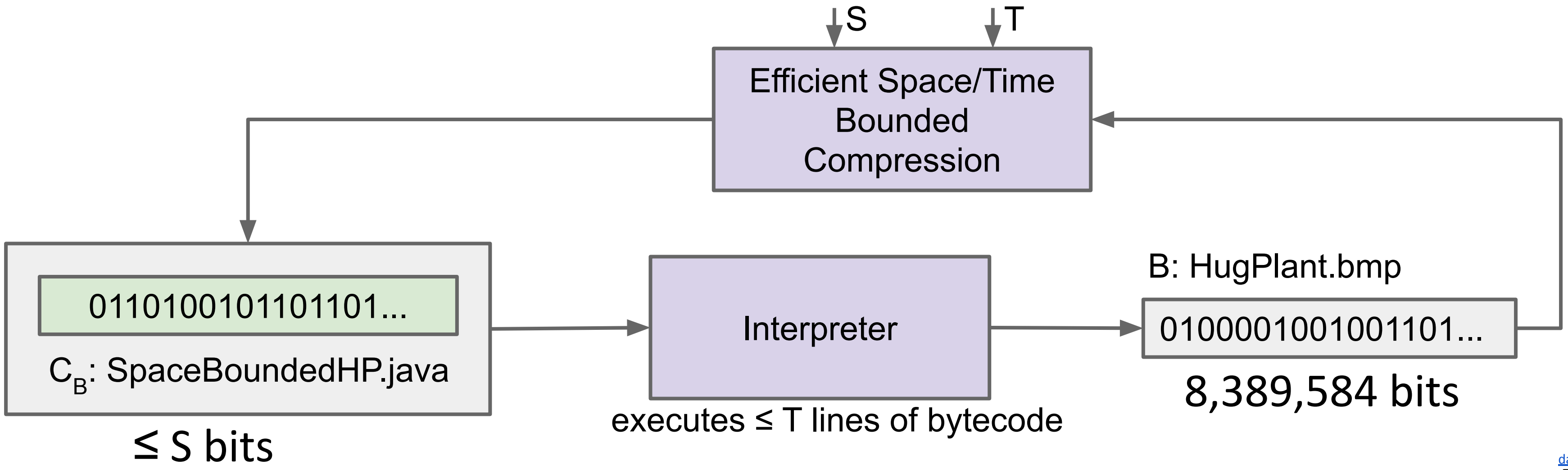
Millenium Prize Problems.

- Hodge conjecture
- Poincare conjecture (solved!)
- Riemann hypothesis
- Yang-Mills existence and mass gap
- Navier-Stokes existence and smoothness
- Birch and Swinnerton-dyer conjecture
- P=NP
  - If true, proof might allow you to trivially solve all of these problems.

Question #2ST-E: Space and Time Bounded Compression

Back to our earlier quest:

- Since Space/Time Bounded Compression can be reduced to 3SAT (or LONGEST_PATH or INDSET or any other NP complete problem), an efficient solution to any NP complete problem would act as a compression algorithm.


S
T
Efficient Space/Time Bounded Compression
0110100101101101...
C_B: SpaceBoundedHP.java
≤ S bits
Interpreter
executes ≤ T lines of bytecode
B: HugPlant.bmp
0100001001001101...
8,389,584 bits

datastructur.es

Even More Impressive Consequences

"I have heard it said, with a straight face, that a proof of P = NP would be important because it would let airlines schedule their flights better, or shipping companies pack more boxes in their trucks!

If [P = NP], then we could quickly find the smallest Boolean circuits that output (say) a table of historical stock market data, or the human genome.... It seems entirely conceivable that, by analyzing these circuits, we could make an easy fortune on Wall Street, or retrace evolution... For broadly speaking, that which we can compress we can understand, and that which we can understand we can predict.

So if we could solve the general case—if knowing something was tantamount to knowing the shortest efficient description of it—then we would be almost like gods. [Assuming P ≠ NP] is the belief that such power will be forever beyond our reach."

- Scott Aaronson *http://www.scottaaronson.com/papers/npcomplete.pdf*

datastructur.es

Does Short = Comprehensible? (Extra)

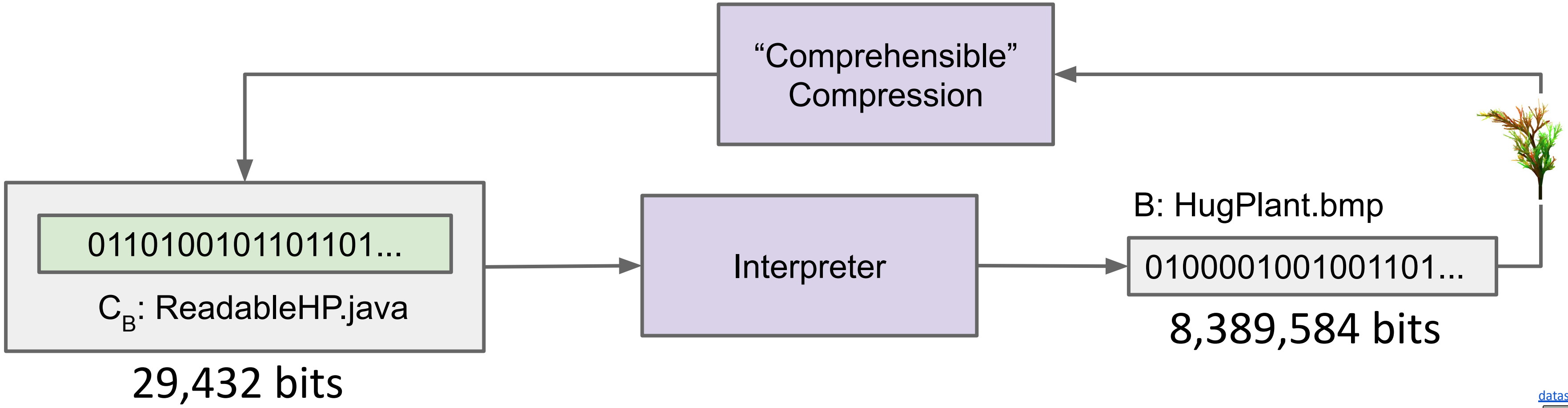
datastructur.es

Back to Question #1: Comprehensible Compression

Earlier, we'd hoped for "comprehensible compression".

Scott Aaronson earlier made an implicit conjecture that a short program will also be comprehensible.

- This seems feasible (but not obvious) to me.
- A short program will probably exhibit hierarchy and structure.
    - Example might even look like HugPlant.java.

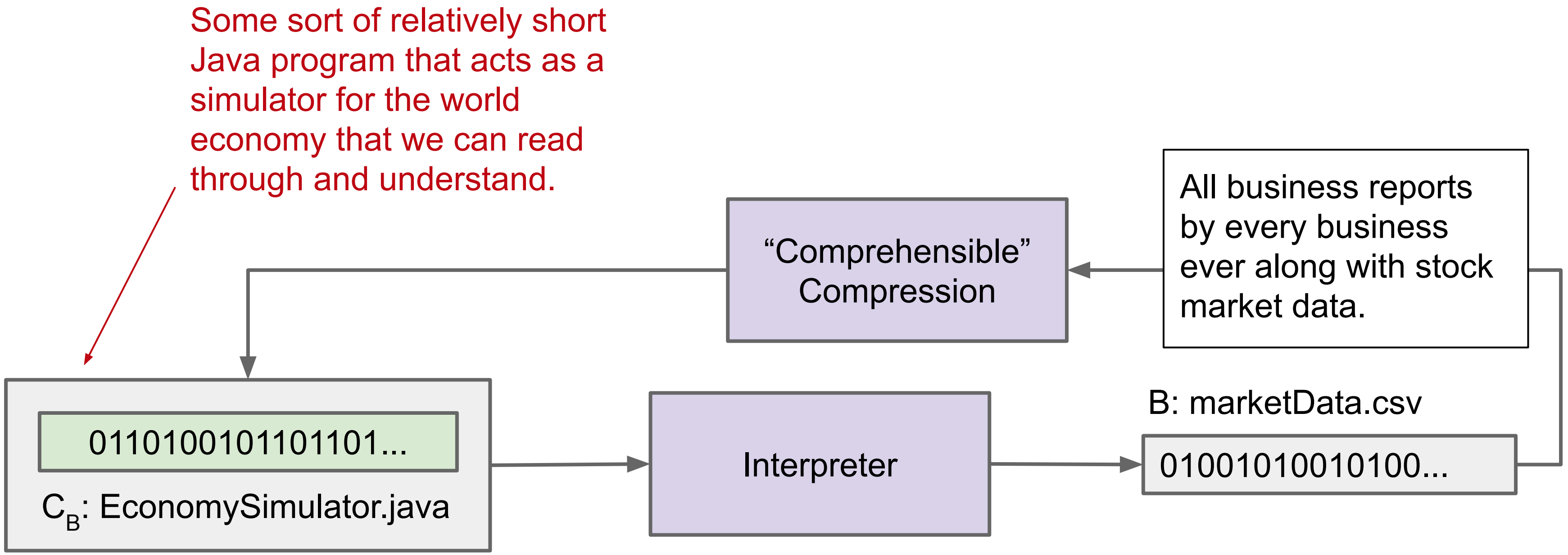


datastructur.es

Comprehensible Compression of Other Data

Scott also conjectures that if we fed in useful data about the stock market, we'd effectively get back a useful economy simulator.

- Could read the source code to understand how the world works.


Some sort of relatively short Java program that acts as a simulator for the world economy that we can read through and understand.
"Comprehensible" Compression
All business reports by every business ever along with stock market data.
0110100101101101...
C_B: EconomySimulator.java
Interpreter
B: marketData.csv
01001010010100...

datastructur.es

Complexity from Simple Rules

However, there are also reasons to suspect that simplicity might not indicate comprehensibility.

Earlier, we said that compressibility was a rare thing.

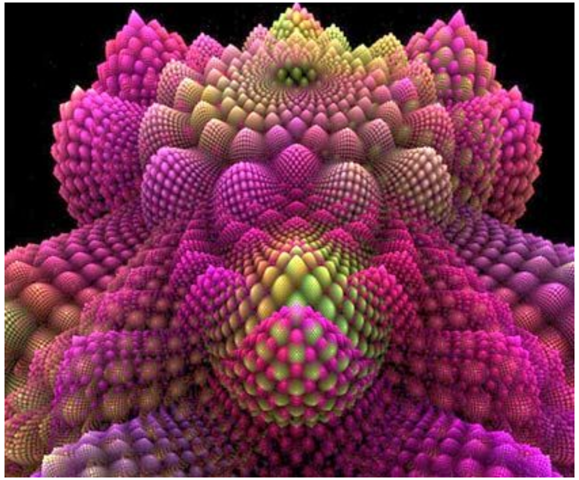
- And yet, short programs can produce surprisingly complex output.

datastructur.es

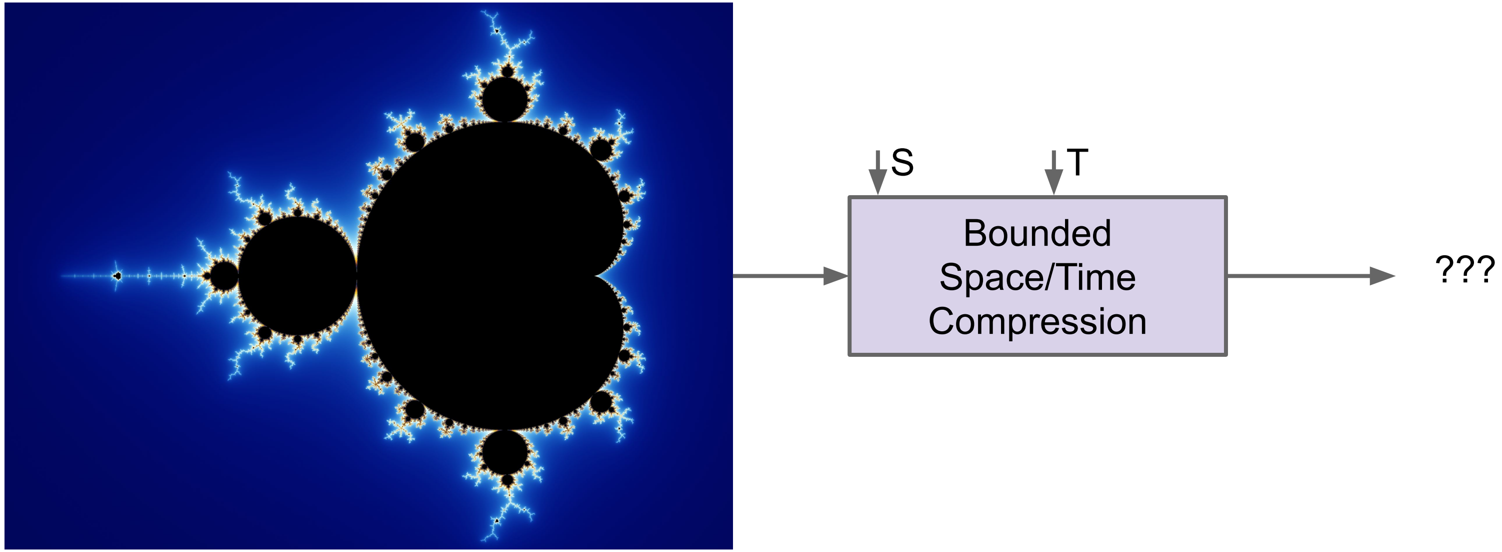
Fractals

In the 1970s, Mandelbrot built demonstrations that very short programs could generate highly complex visual patterns.

- Biological processes exploit these same ideas. Short sequences of DNA give rise to interesting spatial (and other) patterns.



The Mandelbrot Set



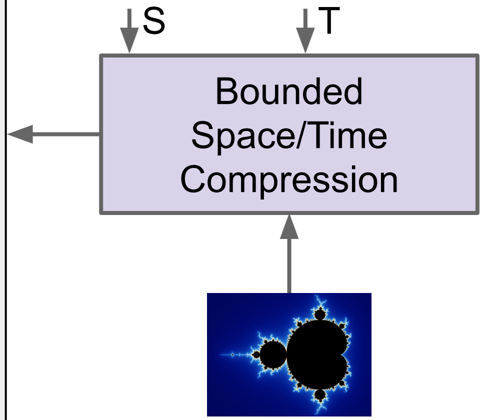
datastructur.es

Short (But Simple)?

```
...
public class Mandelbrot extends JFrame {
    ...
    for (int y = 0; y < getHeight(); y++) {
        for (int x = 0; x < getWidth(); x++) {
            zx = zy = 0;
            cX = (x - 400) / ZOOM;
            cY = (y - 300) / ZOOM;
            int iter = MAX_ITER;
            while (zx * zx + zy * zy < 4 && iter > 0) {
                tmp = zx * zx - zy * zy + cX;
                zy = 2.0 * zx * zy + cY;
                zx = tmp;
                iter--;
            }
        I.setRGB(x, y, iter | (iter << 8));
        }
    }
...
}
```

Hard to say what we learn by looking at this code.

- Very basic rules yield such a highly complex object.
- … but I'm not the complexity theorist!



datastructur.es

Music from Simple Rules

This notion of complexity from simple rules is arguably more interesting (and alarming) when applied towards sound generation.

Music from very short programs (3rd iteration): Youtube

- See this link if you want to try writing your own fractal sound generator.
- Or you can try playing around with this javascript version that I used in my freshman seminar on generative art.
  - Fun program: return 100 * (t & (t >> 3) & (t >> 8)) % 16384;
  - See these slides for examples of interesting inputs.

datastructur.es