Great Theoretical Ideas In Computer Science

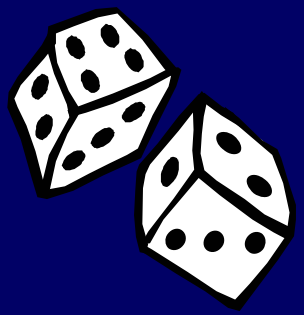
Steven Rudich

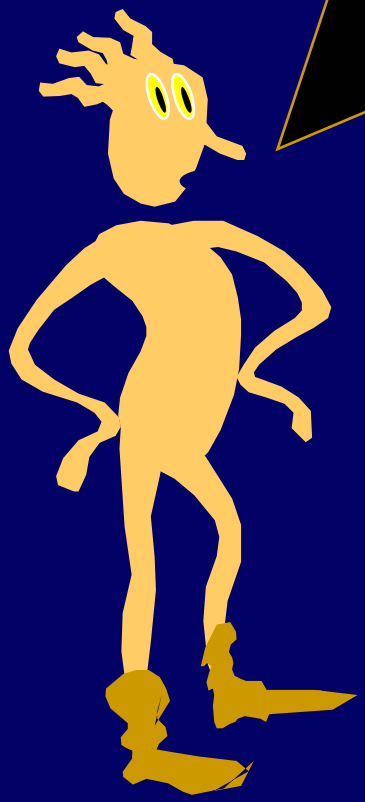
CS 15-251 Spring 2005

Lecture 29 Apr 28, 2005

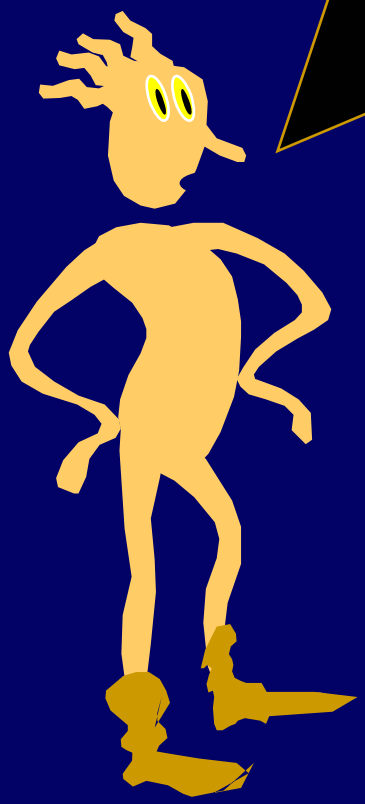
Carnegie Mellon University

Ancient Paradoxes With An Impossible Resolution.

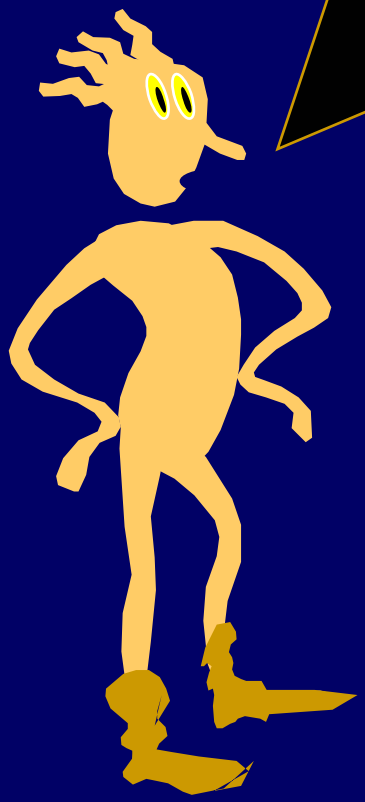




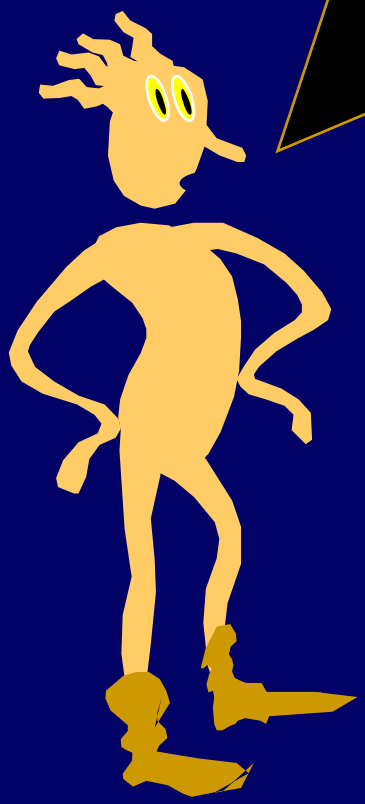
Each Java program has a unique and determined outcome [not halting, or outputting something].



Unless otherwise stated, we will be considering programs that take no input.



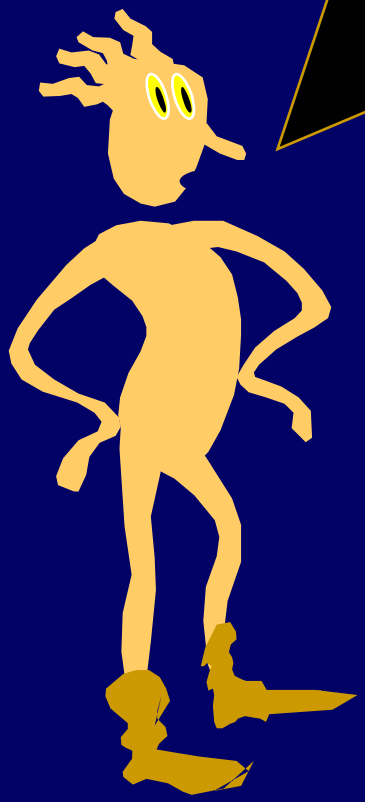
Each Java program has an unambiguous meaning.



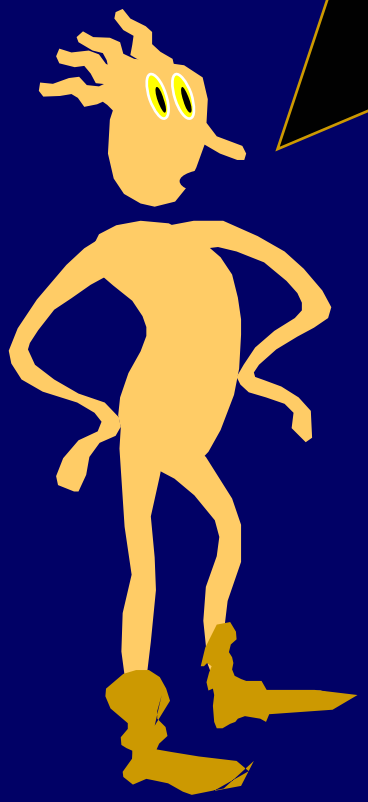
Java is a prefix free language. That is, no Java program is the prefix of any other.

Binary Java is Prefix-Free

We will represent Java in binary (using ASCII codes for each character). We will allow only java programs where all the classes are put in one big class delimited by { }.



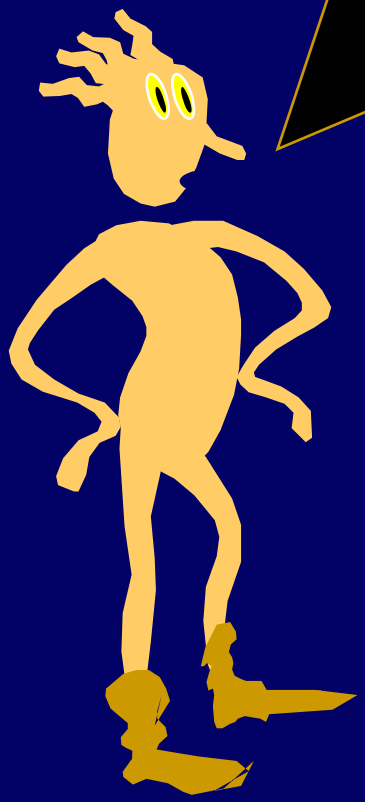
PREFIX FREE MEANS THAT THE NOTION OF A RANDOM JAVA PROGRAM IS WELL DEFINED.



Flip a fair coin to create a sequence of random bits. Stop, If the bits form a Java program P.

Each program gets picked with probability

$(1/2)^{\text{length of program P}}$



Java is an
unambiguous,
prefix-free language.

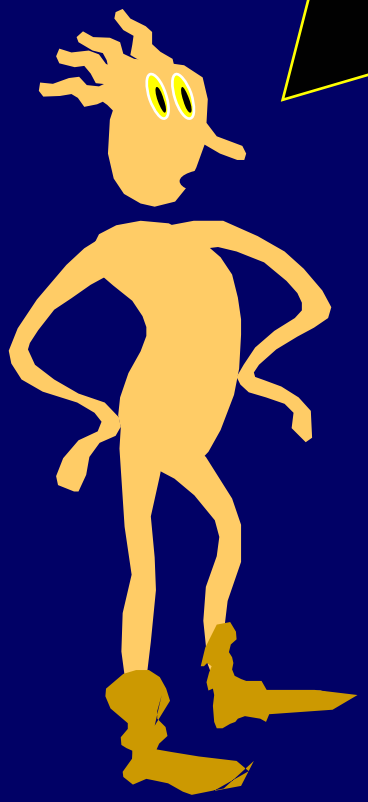
Define Ω to be the probability that a random program halts.

Ω Is the probability that a random coin sequence will describe the text of a halting program.

$$\Omega = \sum_{\text{halting programs p}} 2^{-\text{length of p}}$$

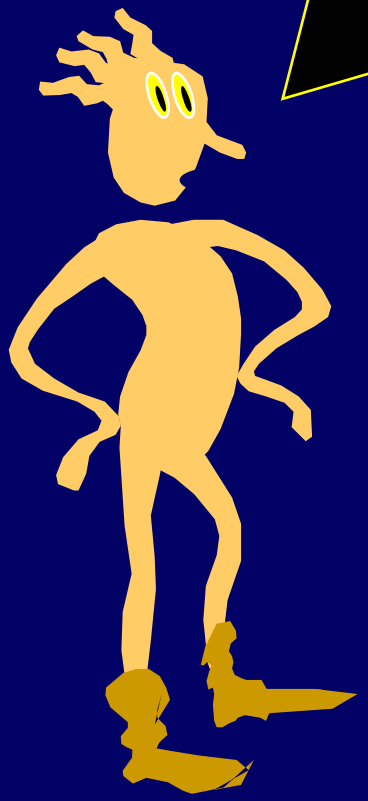
Gottfried Wilhelm von Leibniz

There is almost no paradox without utility

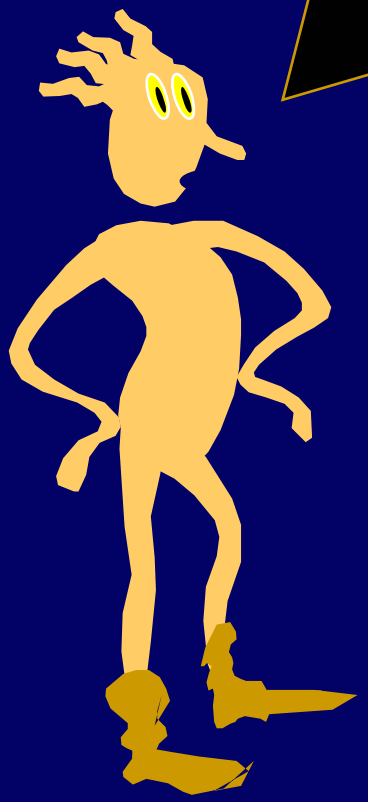


BERRY PARADOX:

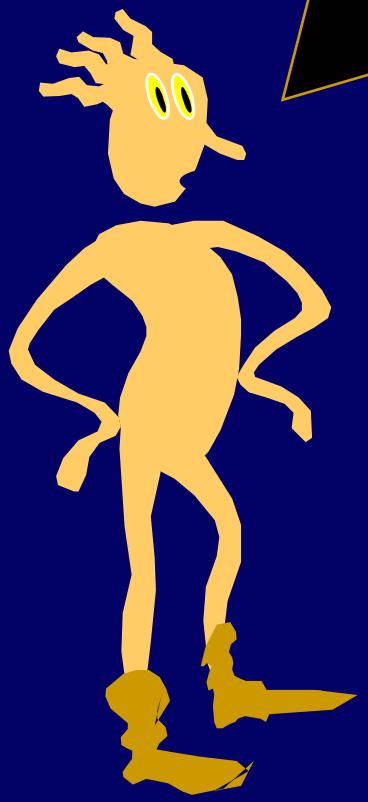
"The smallest natural number that can't be named in less than fourteen words."



List all English sentences of 13 words or less. For each one, if it names a number, cross that number off a list of natural numbers. Smallest number left is number named by the Berry Sentence?

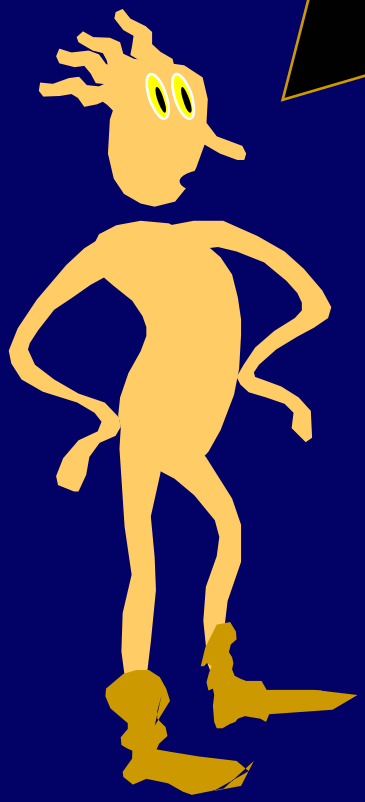


As you loop through sentences, you will meet the Berry sentence. This procedure will not have a well defined outcome.

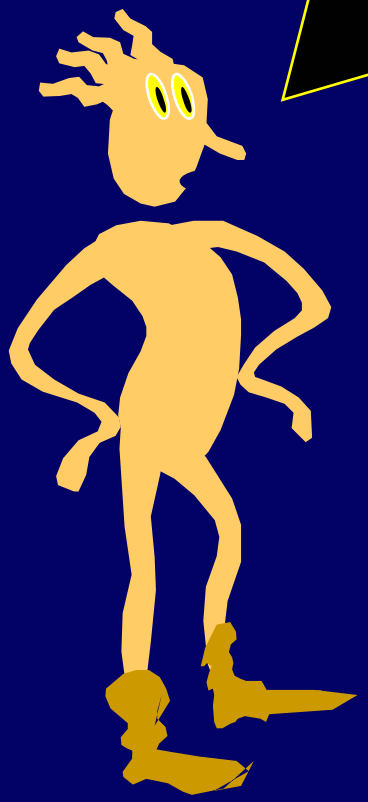


Worse:

In English, there is not always a fact of the matter about whether or not a given sentence names a number.

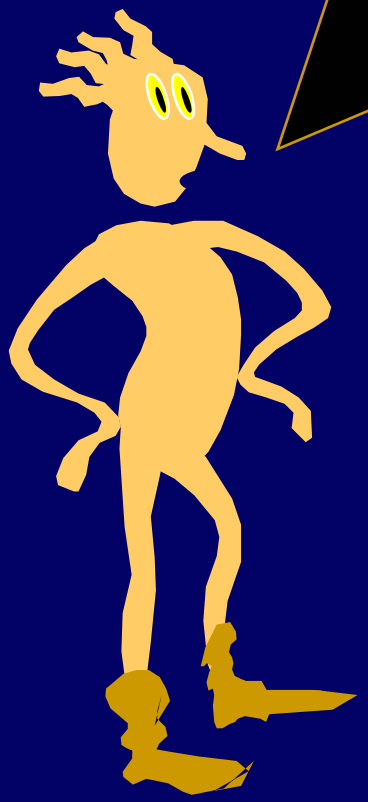


"This sentence refers to the number 7, unless the number named by this sentence is 7."



BERRY PARADOX:

"The smallest natural number that can't be named in less than fourteen words."



Java is a language where each program either produces nothing or outputs a unique string. What happens when we express the Berry paradox in Java?

Counting

A set of binary stings is "prefix-free" if no string in the set is a prefix of another string in the set

Theorem: If S is prefix-free and contains no strings longer than n, then S contains at most 2^n strings.

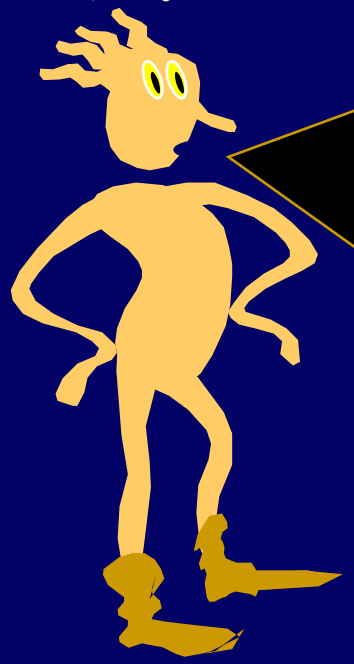
For each string x in S, let f(x) be the string x with n-|X| 0's appended to its right. Thus, f is a 1-1 map from S into $\{0,1\}^n$.

Storing Poker Hands

I want to store a 5 card poker hand using the smallest number of bits (space efficient). The naïve scheme would use 2 bits for a suit, 4 bits for a rank, and hence 6 bits per card and 30 bits per hand. How can I do better?

Order the Poker hands lexicographically

To store a hand all I need is to store its index of size $\lceil \log(2{,}598{,}960) \rceil = 22$ bits.



Let's call this the "indexing trick".

22 Bits Is OPTIMAL

$2^{21} < 2{,}598{,}560$

There are more poker hands than there are 21 bit strings. Hence, you can't have a string for each hand.

Incompressibility

We call a binary string x incompressible if the shortest Binary Java program to output x is at least as long as x.

Th: Half the strings of any given length are incompressible

Java is prefix-free so there are at most 2^{n-1} programs of length n-1 or shorter.

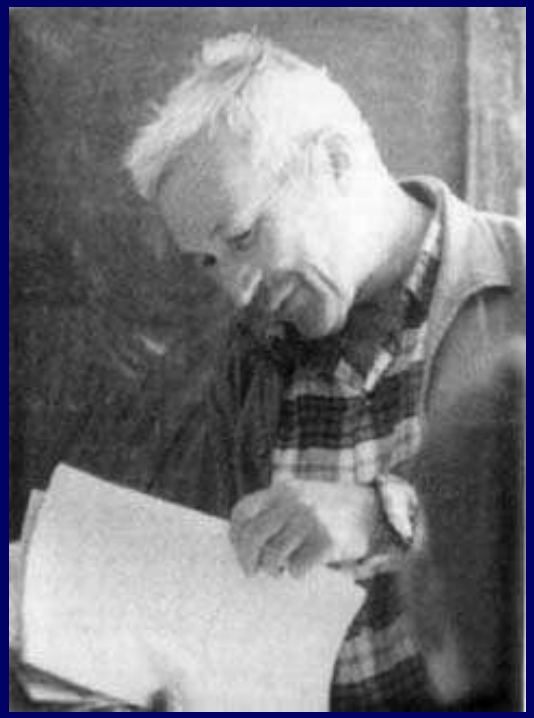
There are 2^n strings of length n, and hence at least half of them have no smaller length program that outputs them.

A compressible string

0101010101010101... a million times ..01

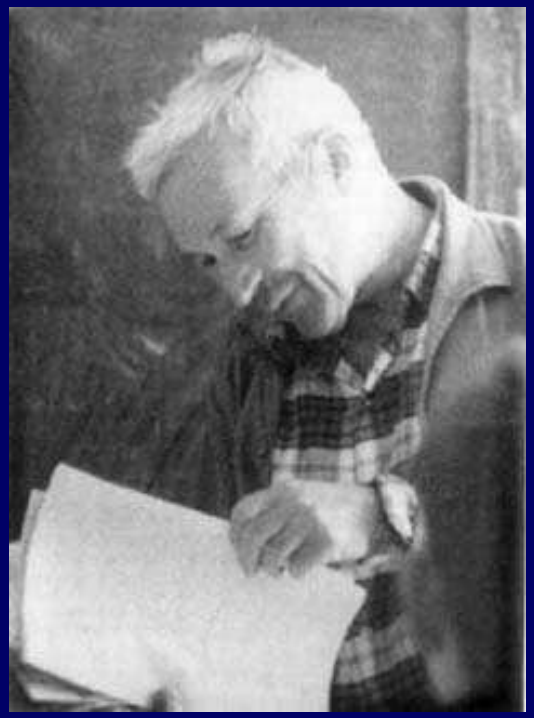
```
public class Counter
{
  public static void main(String argv[])
    {
      for (int i=0; i<1000000; i++)
        System.out.print("01");
    }

}
```

It is possible to *define* randomness in terms of incompressibility



Kolmogorov

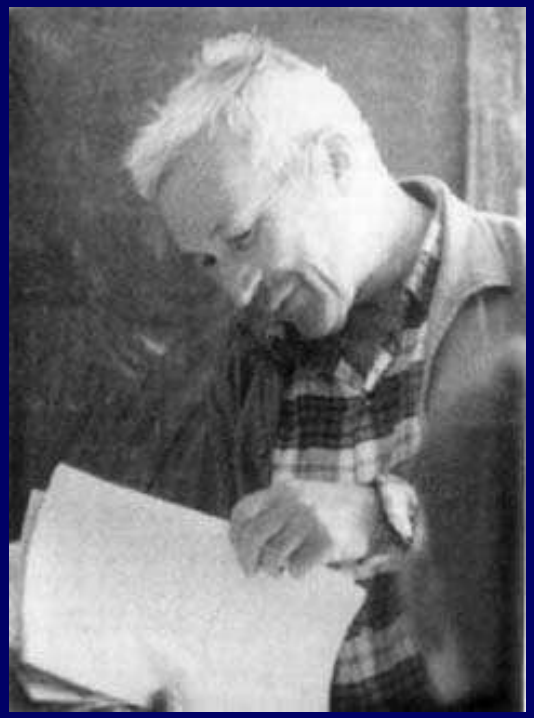
Chaitin



Kolmogorov

Chaitin

An incompressible string has no computable, atypical properties!



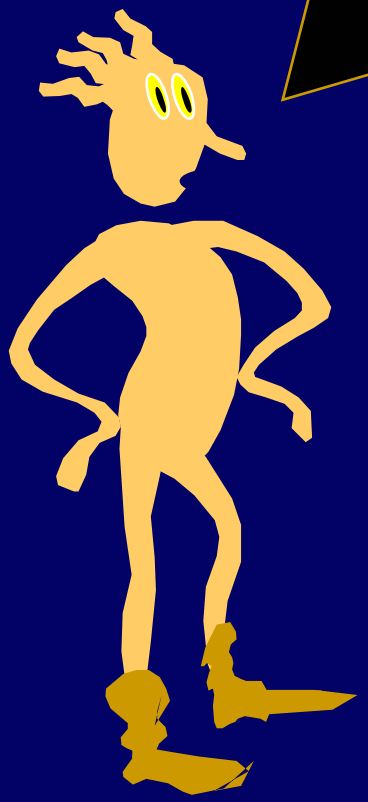
Kolmogorov

Chaitin

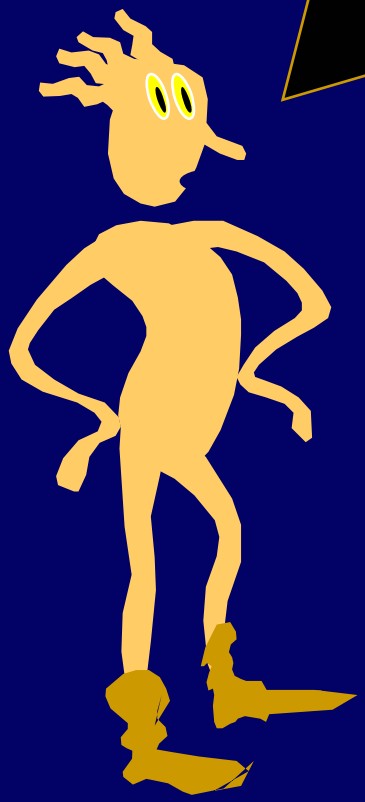
An incompressible string has no computable pattern!

If a string x is incompressible, then there is nothing atypical that you can say about it.

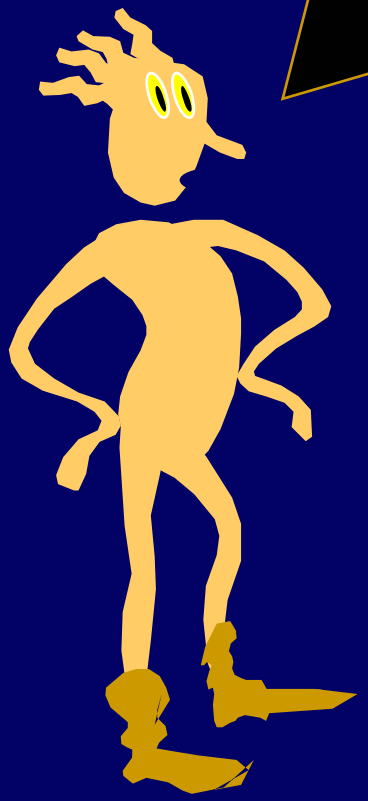
Suppose D is some atypical, computable predicate that is true of x. Since D is atypical, it is not true of many n bit strings. So compress x by referring to x by its index i in the enumeration of strings of length n that have property D. [Notice the use of the "indexing trick"



When we notice a “pattern”, we always mean something atypical.

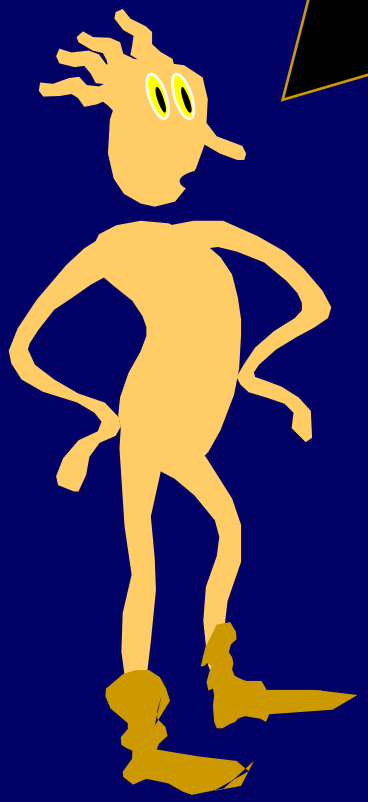


So when you see a “pattern” in a sufficiently long string it allows you to compress it. Hence, incompressible strings have no pattern.



For example, we can compress a sufficiently long Binary string with:

- 60% 1's
- 1 always following 1101
- ASCII Of English Language Text



BERRY PARADOX:

"The smallest natural number that can't be named in less than fourteen words."

Java Berry

The shortest incompressible string that is longer than this Java program

Java Berry

The shortest incompressible string that this program can certify is longer than this program

Define an Incompressibility Detector to be a program P such that:

P(x) = "yes" means x is definitely incompressible

P(x) = "not sure", otherwise

Let INCOMPRESSIBLE be a JAVA incompressibility detector whose program length is n.

INCOMPRESSIBLE(x) = "yes" means x is definitely incompressible

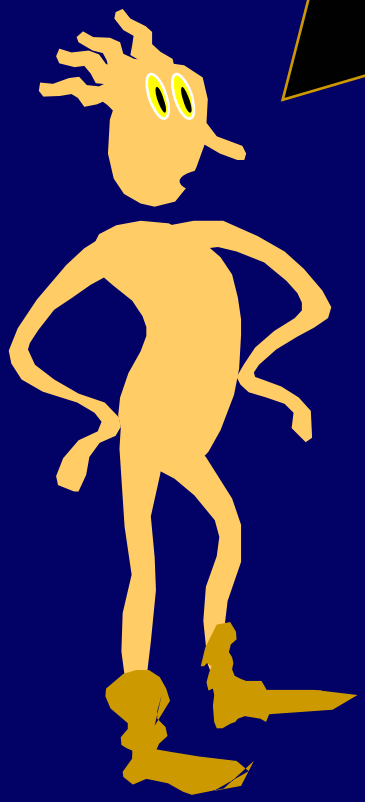
INCOMPRESSIBLE(x) = "not sure", otherwise

JAVA BERRY

{
k:= bound on length of my program text
Loop x = strings of length k+1 to infinity
{ If INCOMPRESSIBLE(X) Output X}

Text of subroutine for INCOMPRESSIBLE.
}

The shortest incompressible string that this program can certify is longer than this program



If JAVA BERRY outputs ANYTHING a real paradox would result!

JAVA BERRY

```
{
S = Text of subroutine for INCOMPRESSIBLE
k:= STRING_LENGTH(S)
Loop x = strings of length k+b to infinity
 { If EXECUTE(S, X) = "YES" Output X}

Routine for EXECUTE (S,X) which executes the Java program
is the string S on input X

Routing for STRING_LENGTH(S) returns the length of string
S
}
```

BERRY has text length b + n

Note: b is a constant, independent of n

JAVA BERRY OUTPUTS NOTHING.

Theorem: There is a constant b such that no INCOMPRESSIBLE detector of length n outputs "yes" on any string of length greater than n+b.

Proof: If so, we could use it inside Java Berry and obtain a contradiction.

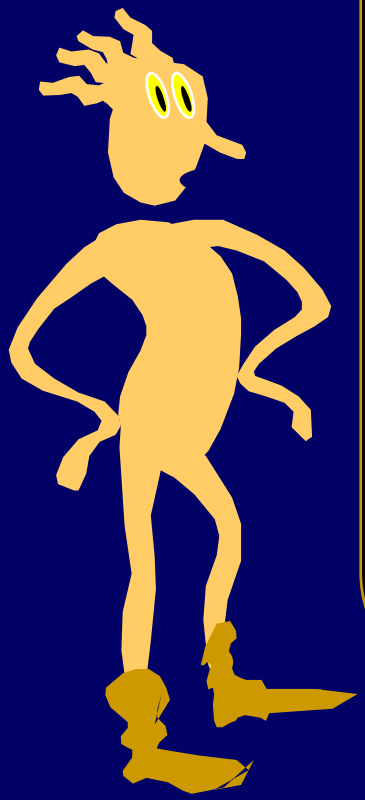
Let Π be a sound, formal system that can be presented as a n-bit program enumerating consequences of its axioms.

No statement of the form

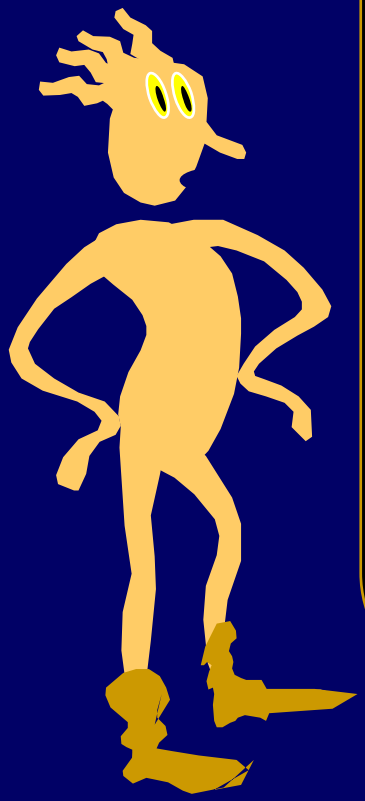
"X is incompressible"

for X of length > n+b

is a consequence of Π.



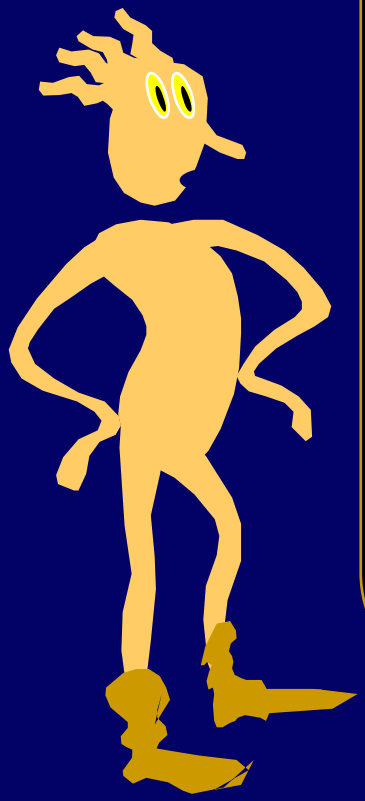
You fix any n-bit foundation for mathematics. Now consider that half of the strings of length m>n+b are incompressible. Your foundation can’t prove that any one of them is incompressible.



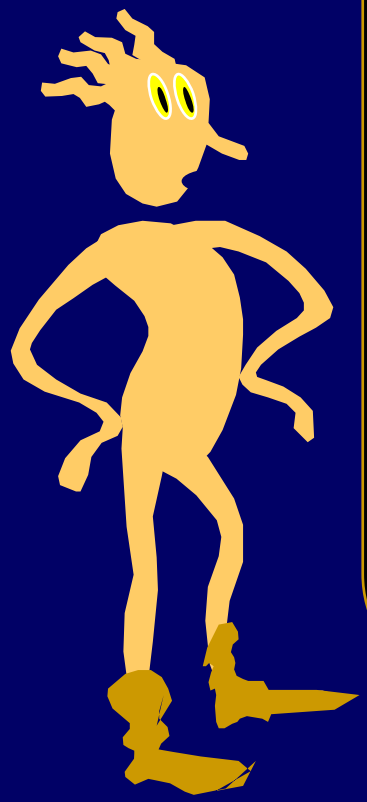
Random Unknowable Truths.

Define Ω to be the probability that a random program halts.

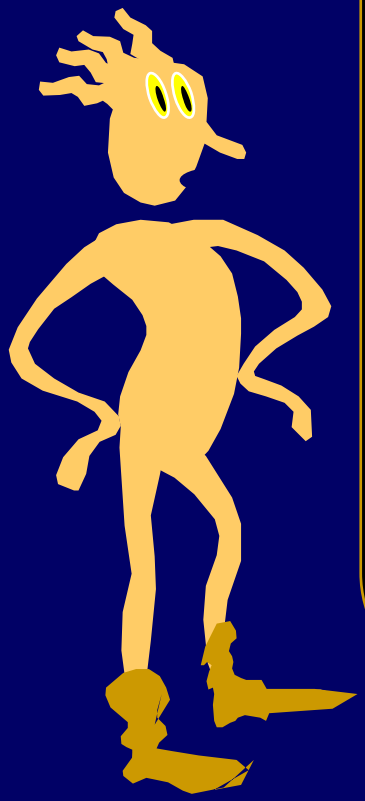
$$\Omega = \sum_{\text{halting programs p}} 2^{-\text{length of p}}$$



Ω is a maximally unknowable number



Ω is the optimally compressed form of the halting oracle.



Let Ω_n be the first n-bits of Ω. By the properties of binary representation:

$$\Omega - \Omega_n < \tfrac{1}{2}^n$$

The first n bits of Ω give enough information to solve the halting problem for any program with length bounded by n.

Let Ω_n be the first n bits of Ω. Let P be a program of length n, of weight 2^{-n}.

Start with a balance with Ω_n on the left side and nothing on the other:

Ω_n

Notice that $\Omega_n + \frac{1}{2}^n$ is greater than Ω

The first n bits of Ω give enough information to solve the halting problem for any program with length bounded by n.

Now start time sharing to run every program except P for an infinite number of steps each. If a program M halts, put weight $\frac{1}{2}^{|M|}$ on the right side:

$[\frac{1}{2}^{|M|} \ldots.]$

Converging to Ω or to $\Omega - (1/2)^n$

Ω_n

The first n bits of Ω give enough information to solve the halting problem for any program with length bounded by n.

If P halts, then $W < \Omega - \frac{1}{2}^n < \Omega_n$. Hence, the balance will never tip.

If P does not halt, W converges to Ω, and hence the balance must tip.

$$W = \frac{1}{2}^n \left[\frac{1}{2}^{|M|} \ldots.\right]$$

Ω_n

The first n bits of Ω give enough information to solve the halting problem for any program with length bounded by n.

L:=0

Timeshare each program M, except P

When a program of length a halts, add 2^{-a} to L.

When the first n bits of L equals the first n-bits of Ω, any length <= n program that is going to halt will have halted.

Busy Beaver Function

BusyBeaver(n) = max running time of any halting program of length n.

In BusyBeaver(n) we can unpack the first n bits of information encoded in Omega.

From n-bits of Ω we can find all incompressible strings of length n+1

Determine all the programs of length n that halt. Run them and cross off any (n+1)-bit strings they output. The strings that are left are incompressible.

n bits of axioms can only help you know n + b bits of Ω

Or else you could prove that strings longer than your axiom system were incompressible

Ω Is not compressible by more than b.

Suppose you could compress n bits of Ω by more b to get a string X. Decompress X and use it to find an incompressible strings of length n+1 and output it. This method has the length of X plus b which is still less than n+1. Contradiction.

Busy Beaver Function

BusyBeaver(n) = max running time of any halting program of length n.

In BusyBeaver(n) we can unpack the first n bits of information encoded in Omega.

Busy Beaver Function

BusyBeaver(n) = max running time of any halting program of length n.

What is the growth rate of BusyBeaver?

Grows faster than any computable function!

Suppose a computable f(n) > BusyBeaver(n)

BusyBeaver(n) = max running time of any halting program of length n.

Run all n-bit programs for f(n) time.

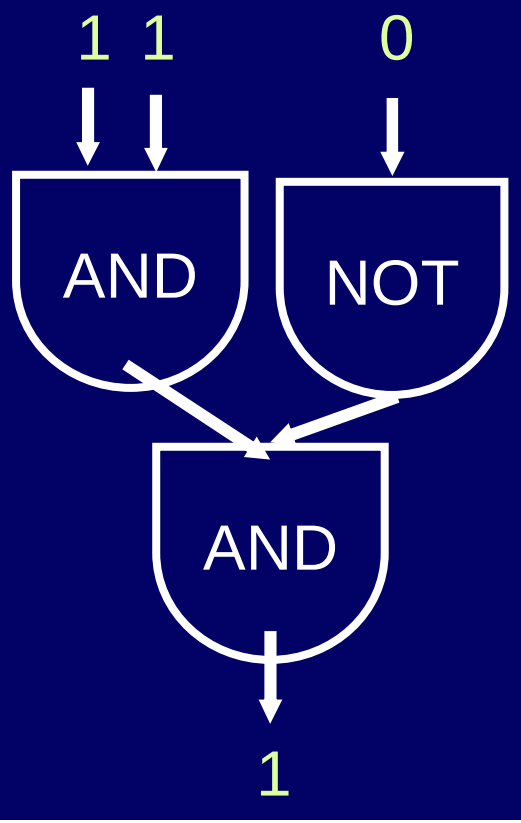
The ones that have not halted will never halt.

Reason is our most powerful tool, but some truths of the mathematical world have no pattern, or representation that can be reasoned about.

We can make a Diophantine polynomial U in 16 variables such that when X_1 is fixed to k, the resulting polynomial has a root iff the kth bit of Omega is 1.

CIRCUIT-SATISFIABILITY

Given a circuit with n-inputs and one output, is there a way to assign 0-1 values to the input wires so that the output value is 1 (true)?



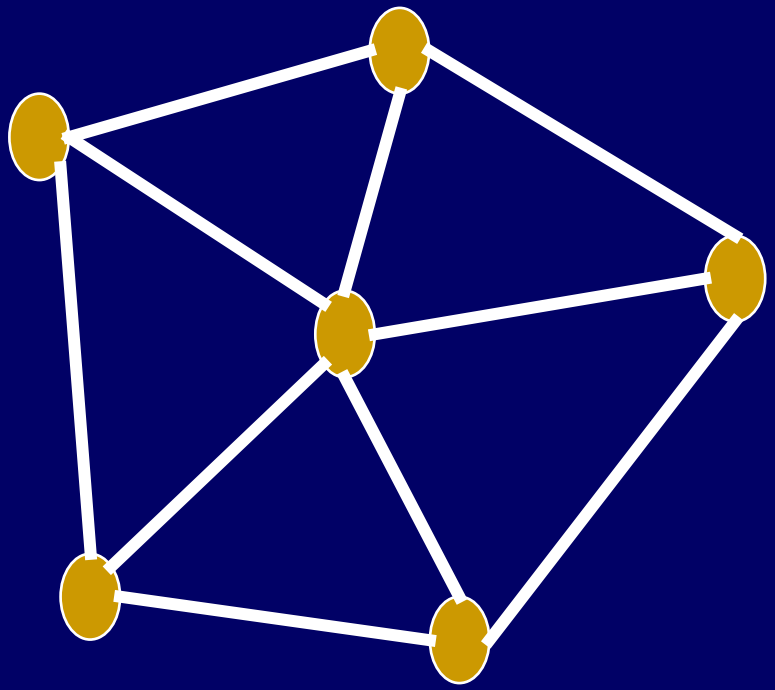
Yes, this circuit is satisfiable. It has satisfying assignment 110.

CIRCUIT-SATISFIABILITY

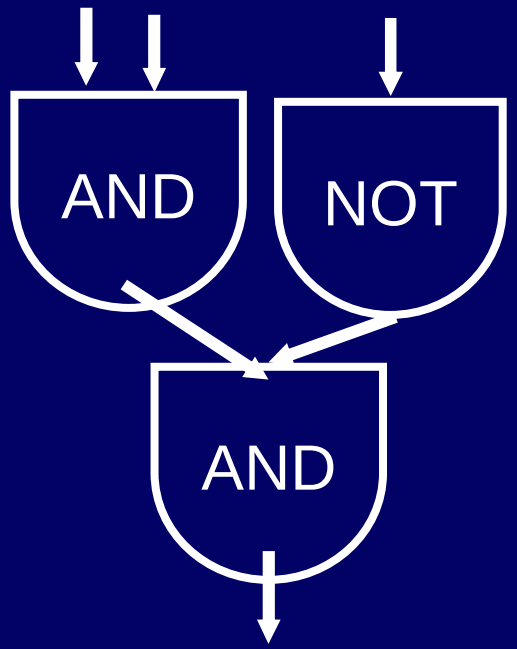
Given: A circuit with n-inputs and one output, is there a way to assign 0-1 values to the input wires so that the output value is 1 (true)?

BRUTE FORCE: Try out all 2^n assignments

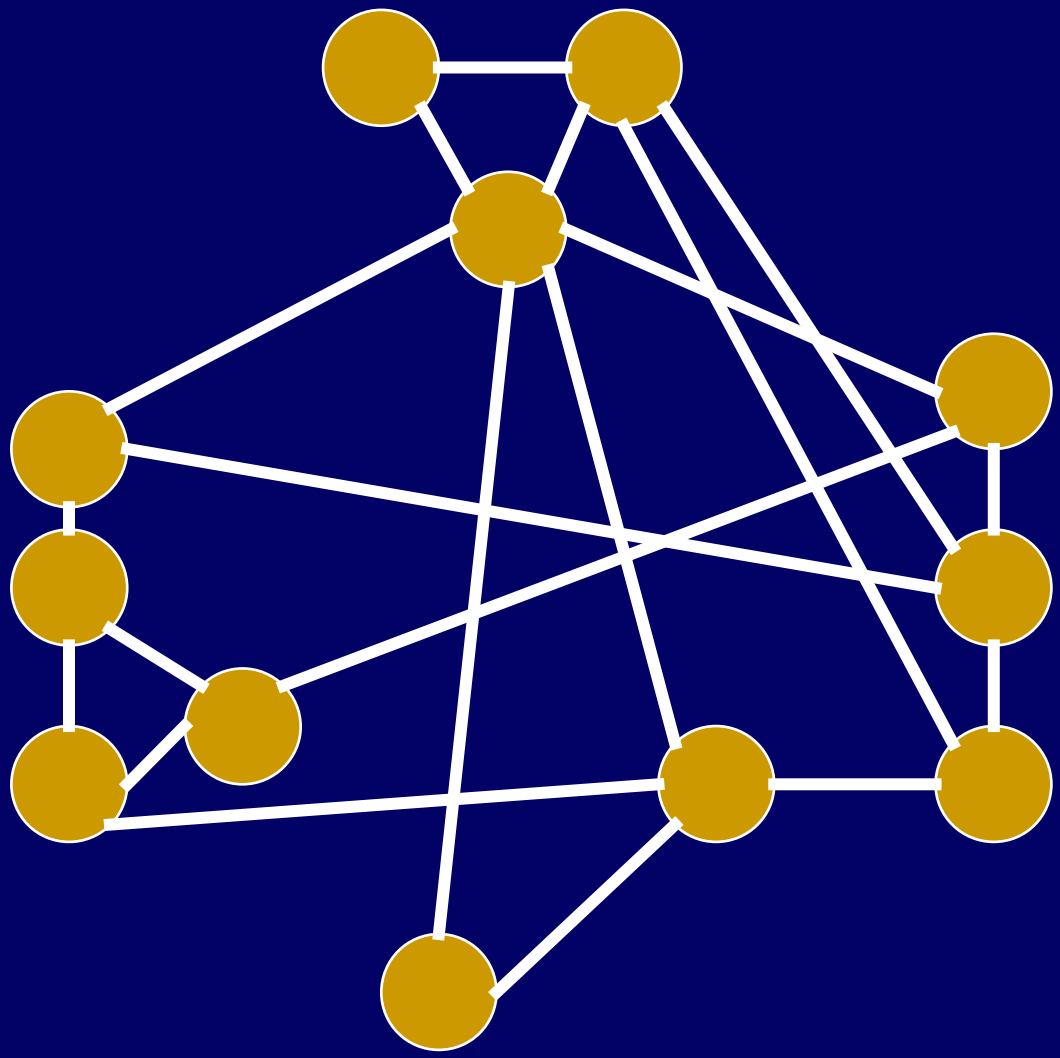
3-colorability

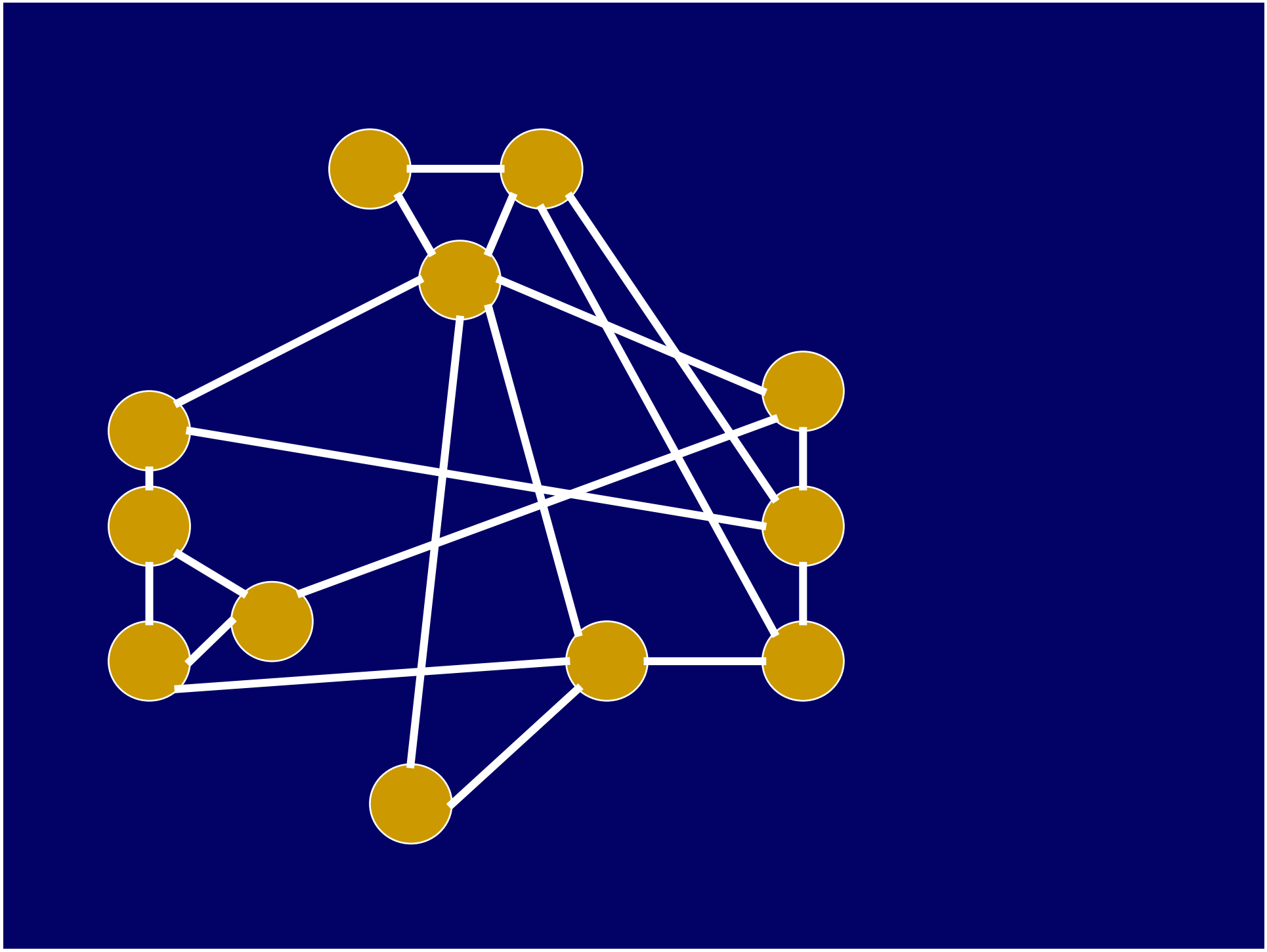


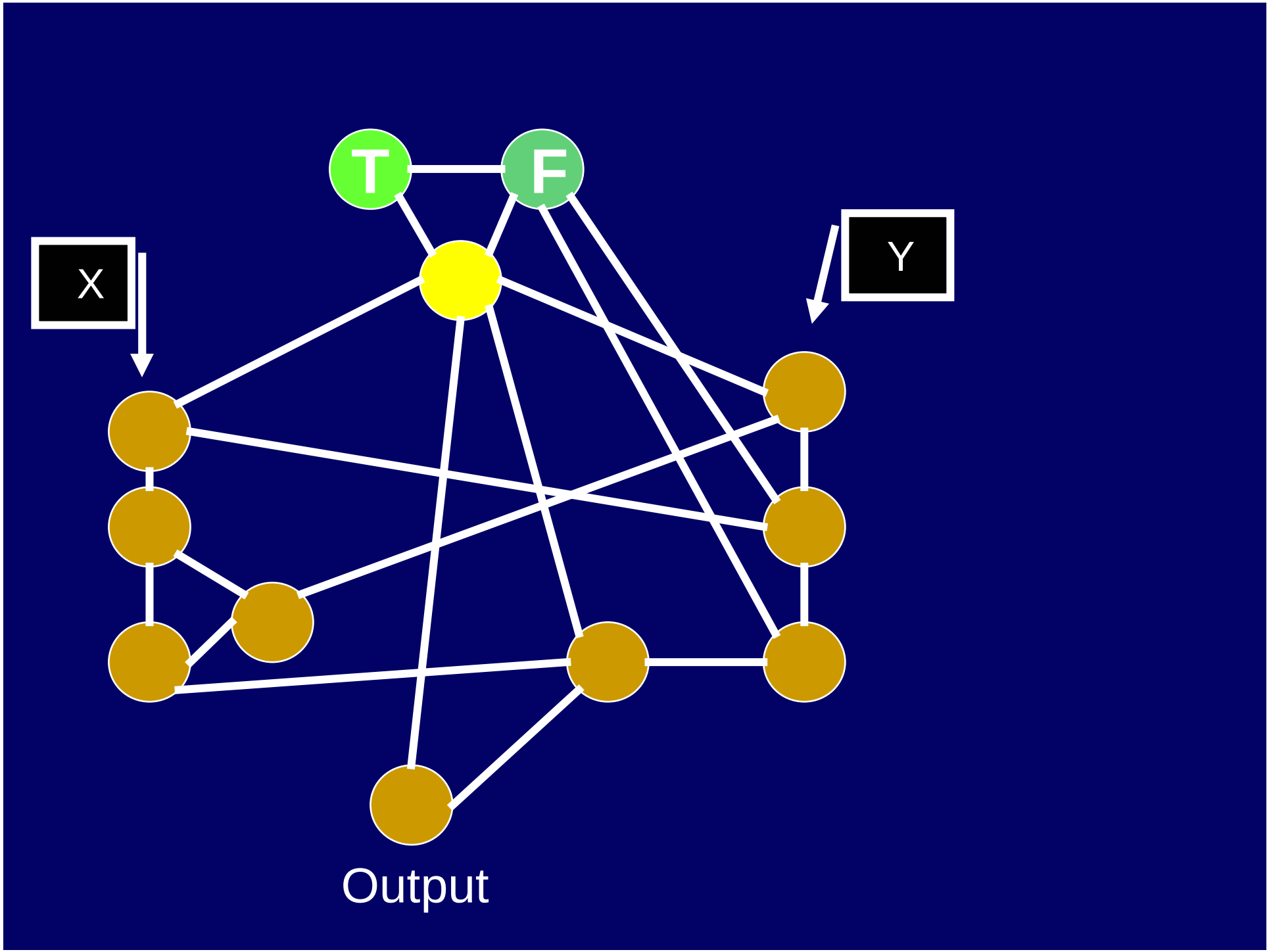
Circuit Satisfiability

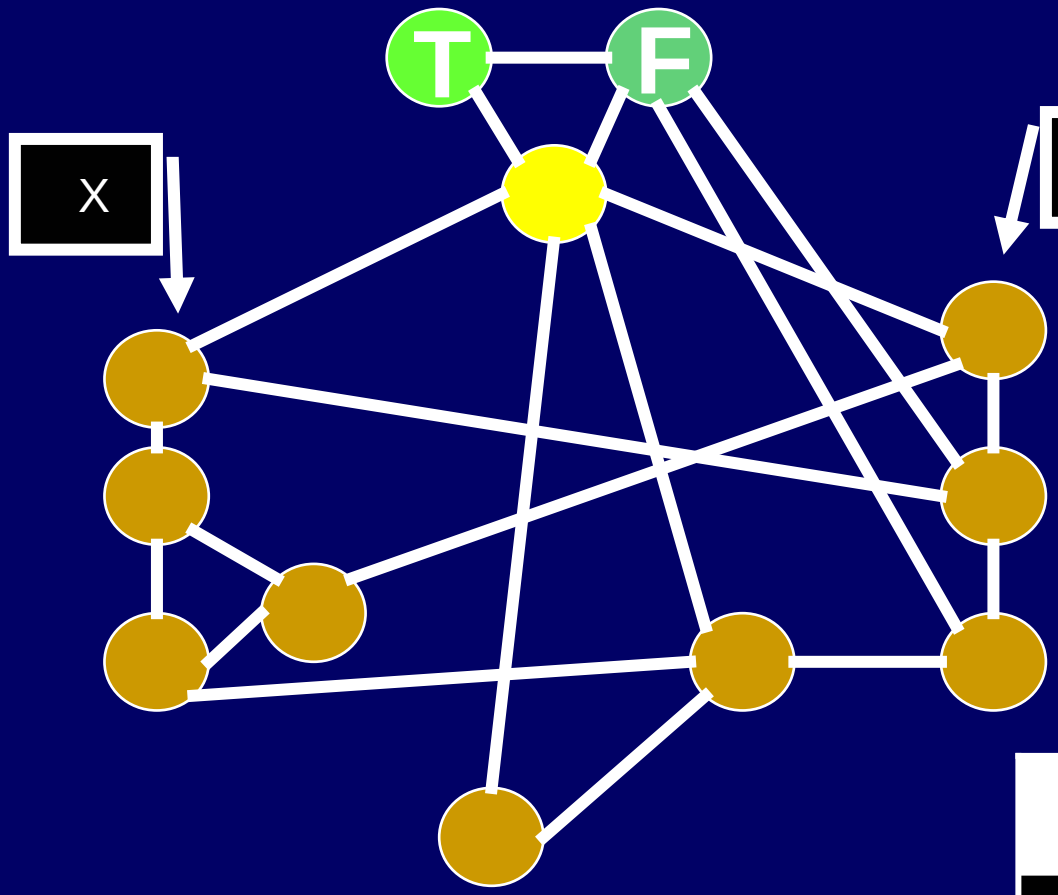


A Graph Named "Gadget"

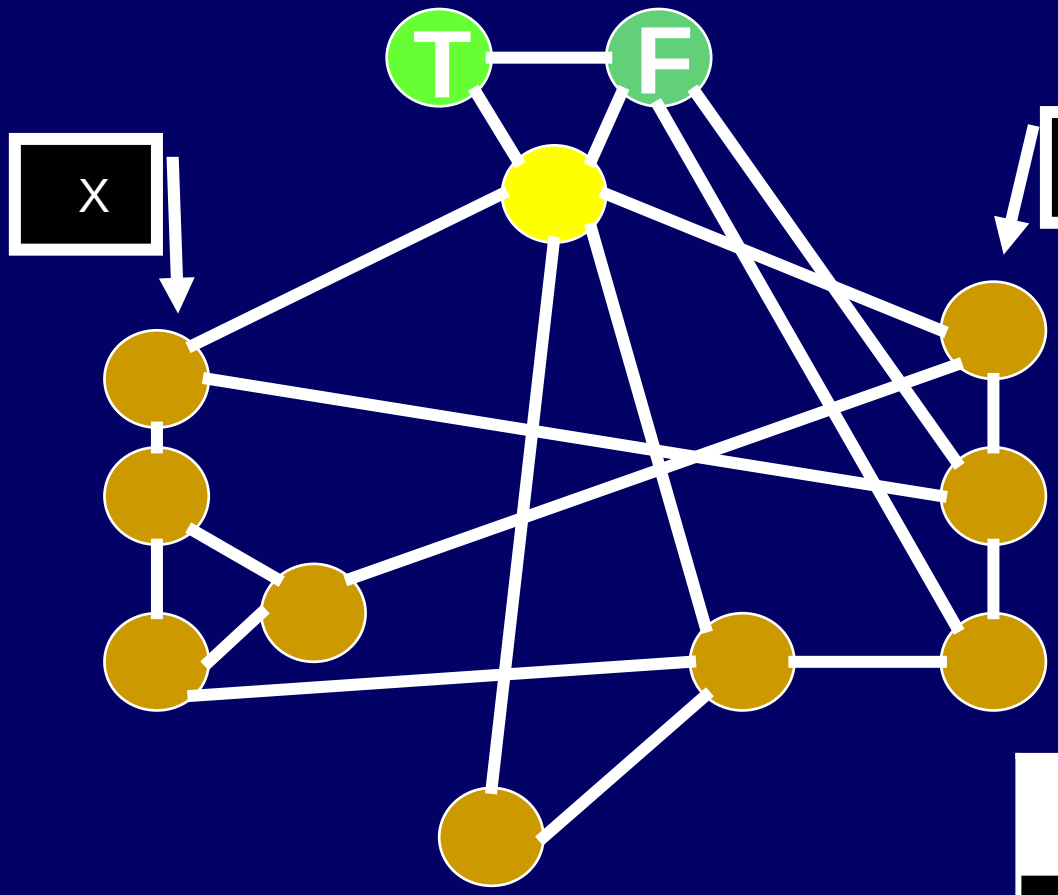




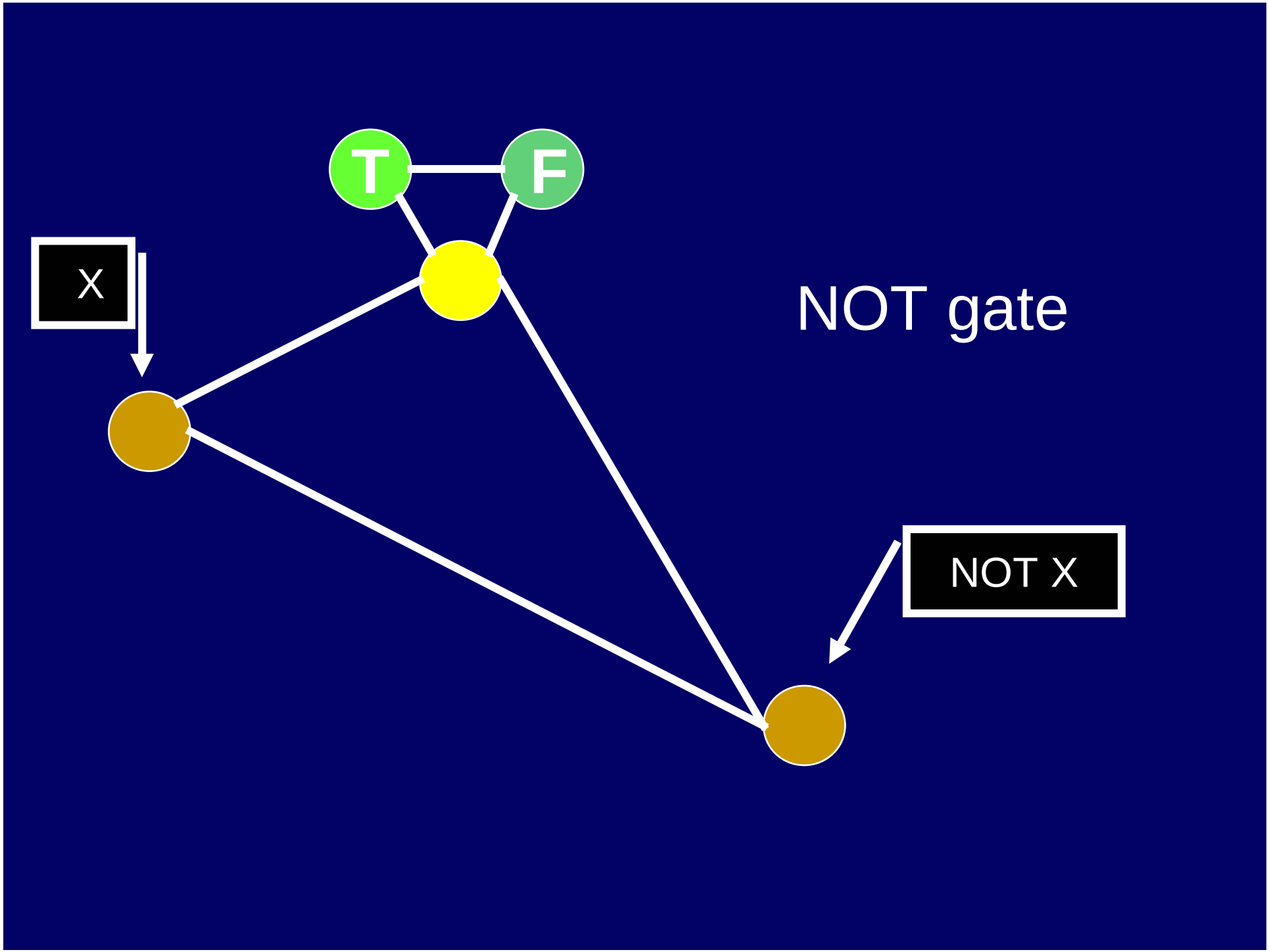

T
F
X
Y
Output

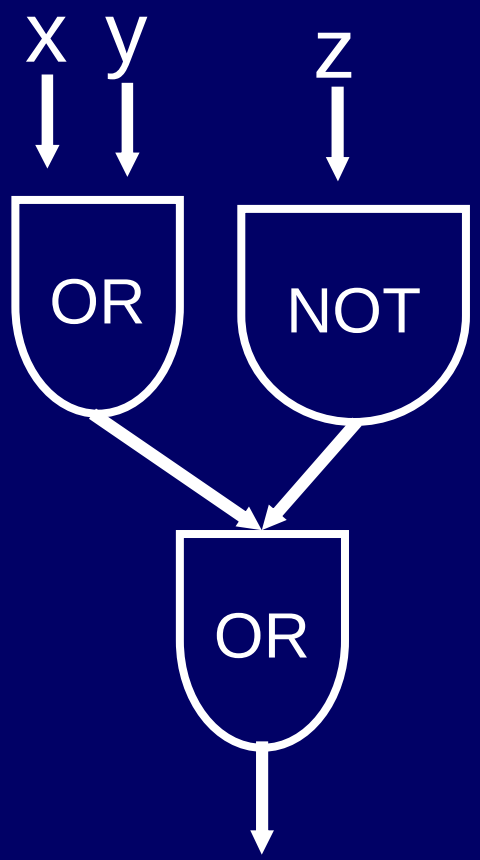


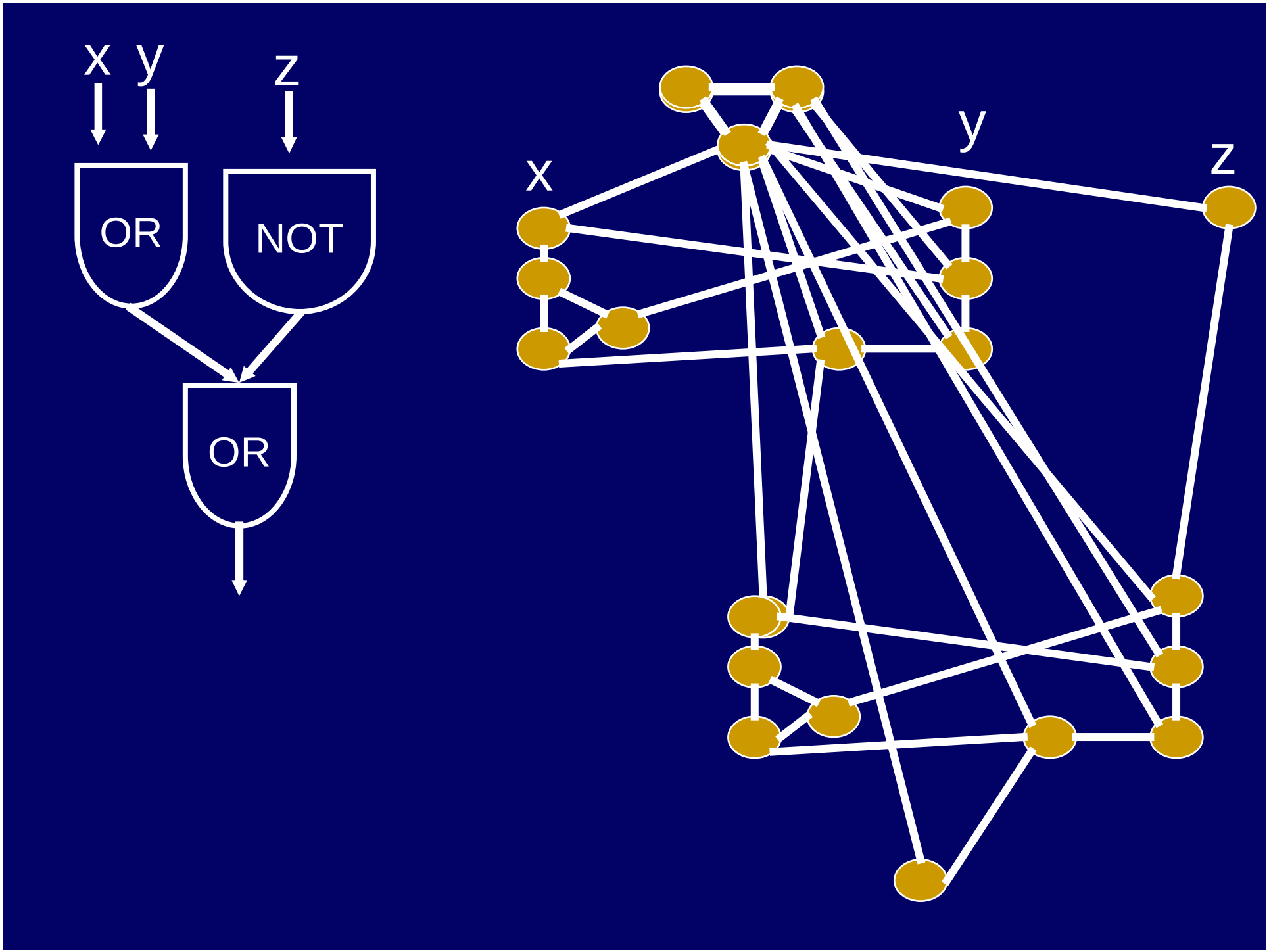
X	Y	
F	F	F
F	T	T
T	F	T
T	T	T

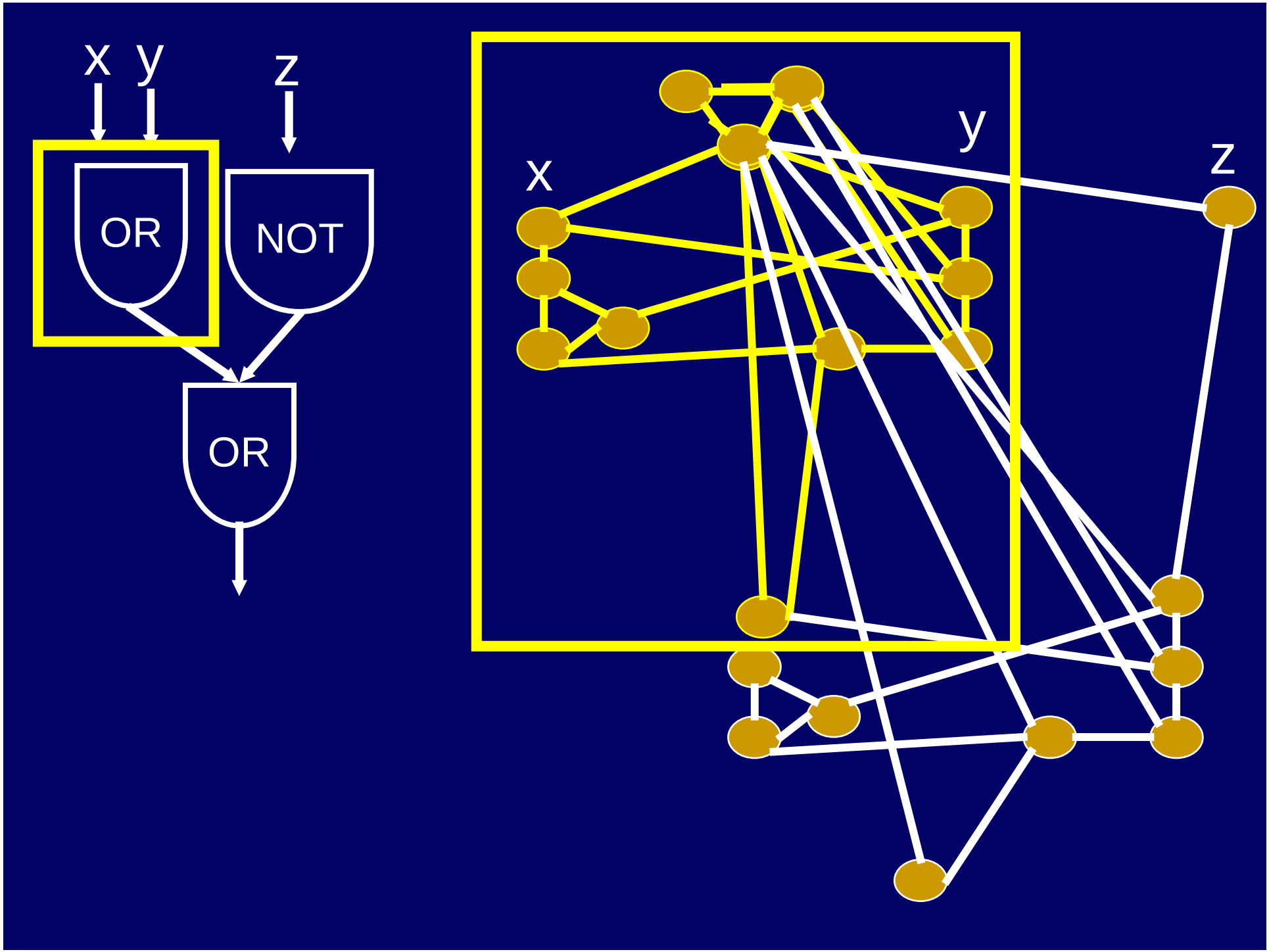


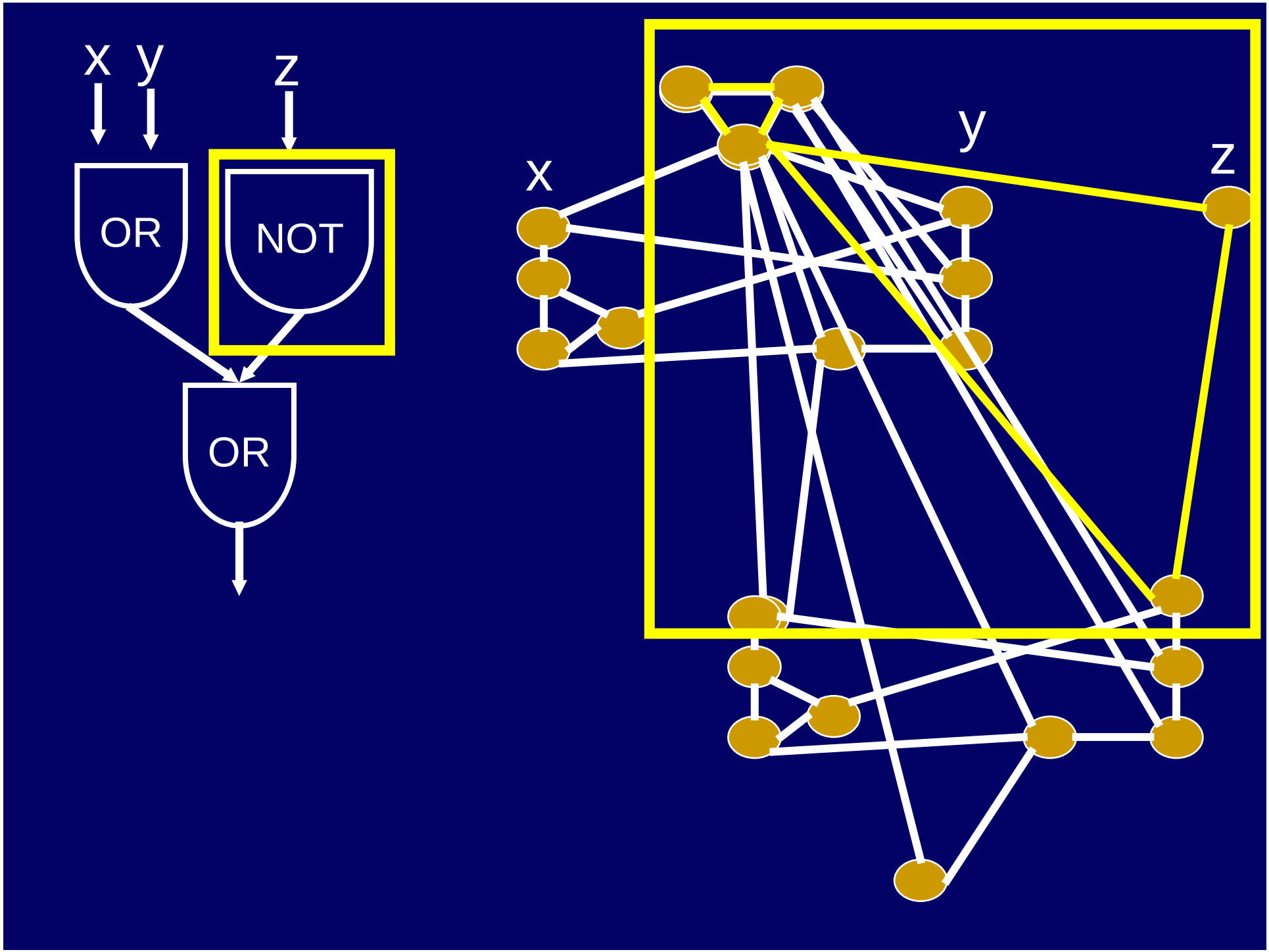
X	Y	OR
F	F	F
F	T	T
T	F	T
T	T	T

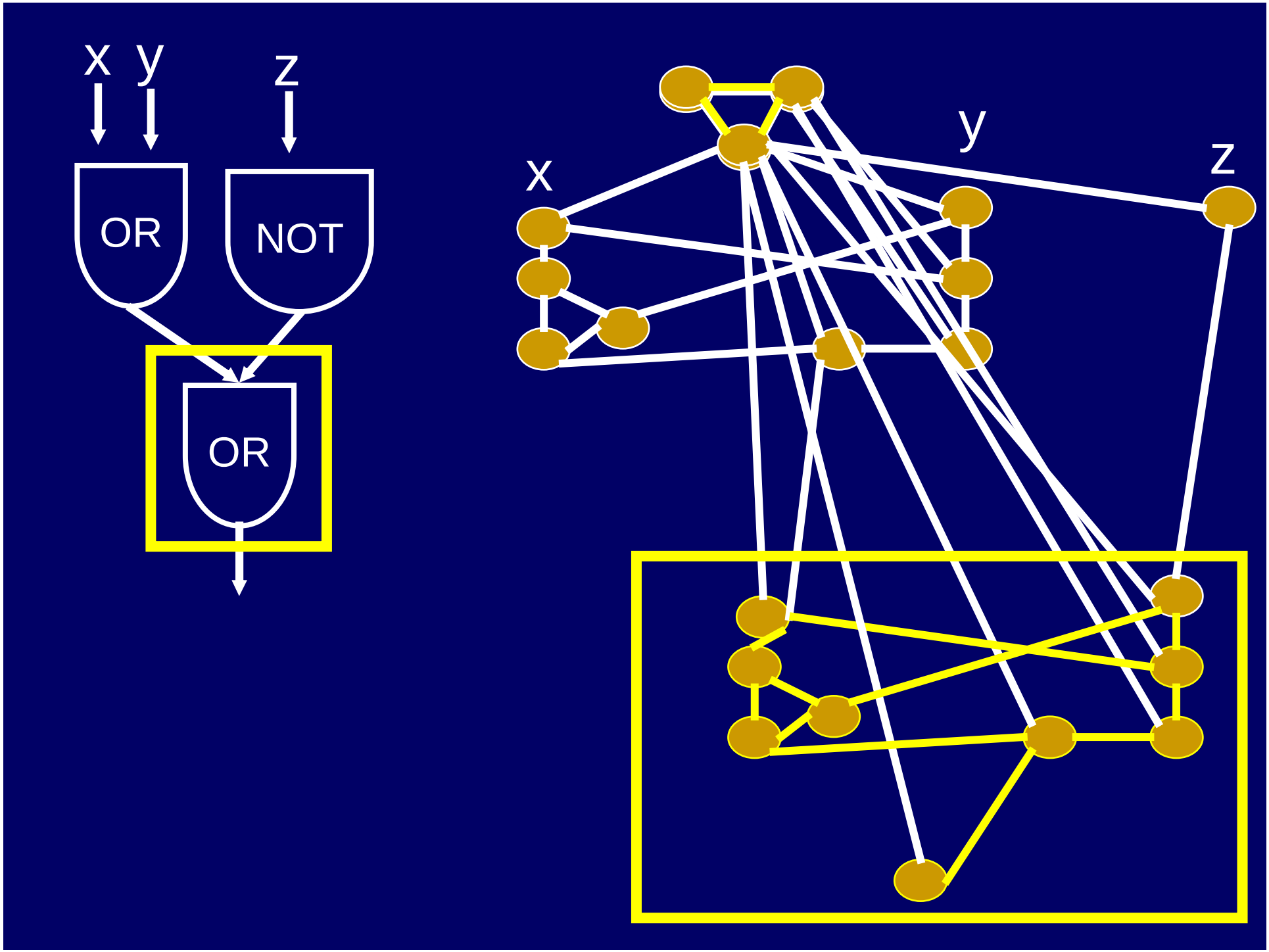

T
F
X
NOT gate
NOT X

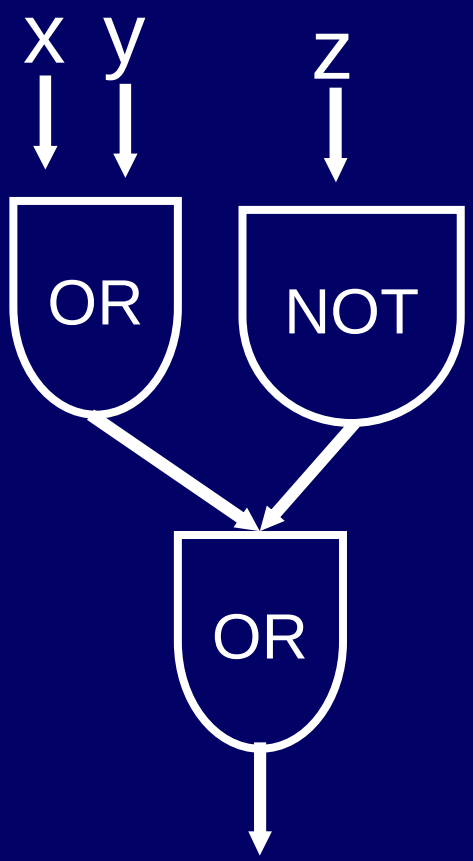

x
y
z
OR
NOT
OR

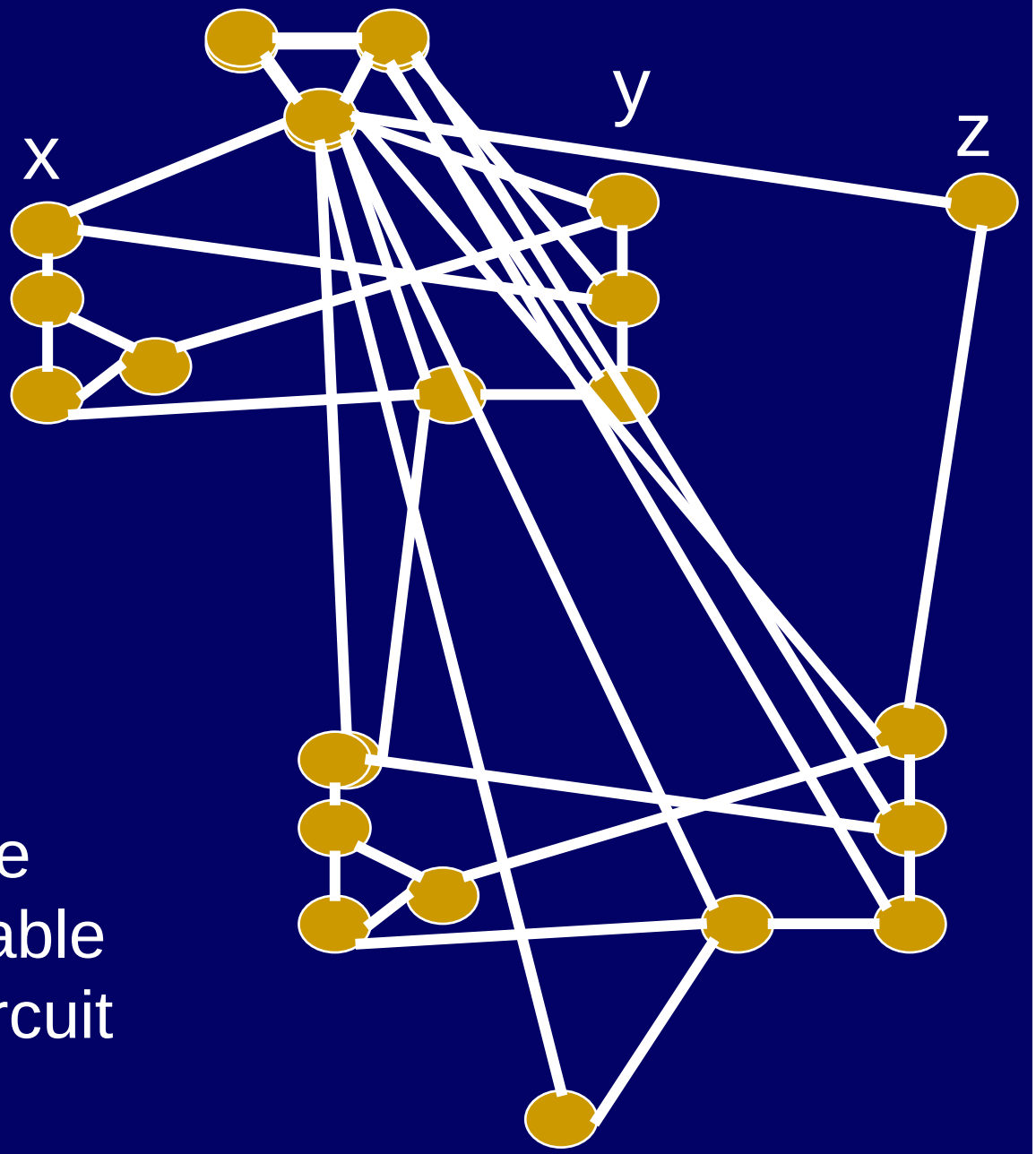

x
y
z
OR
NOT
OR
x
y
z


x
y
z
OR
NOT
OR
x
y
z

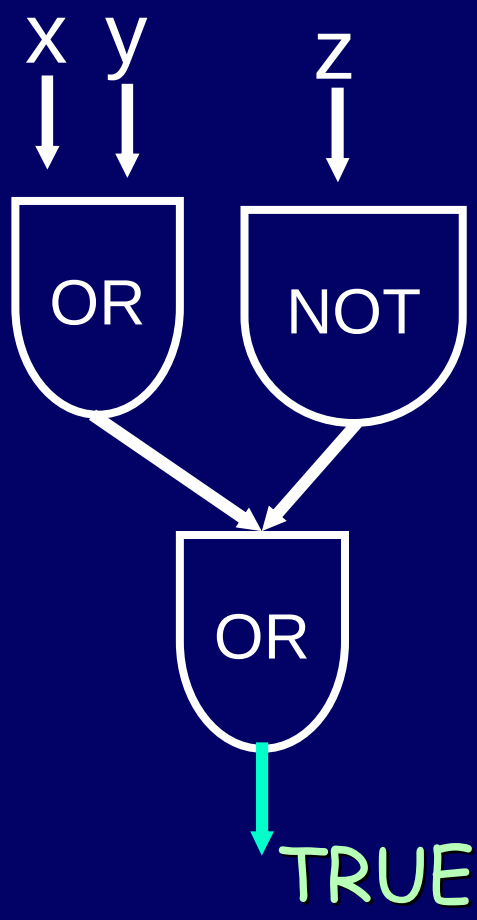

x
y
z
OR
NOT
OR
x
y
z

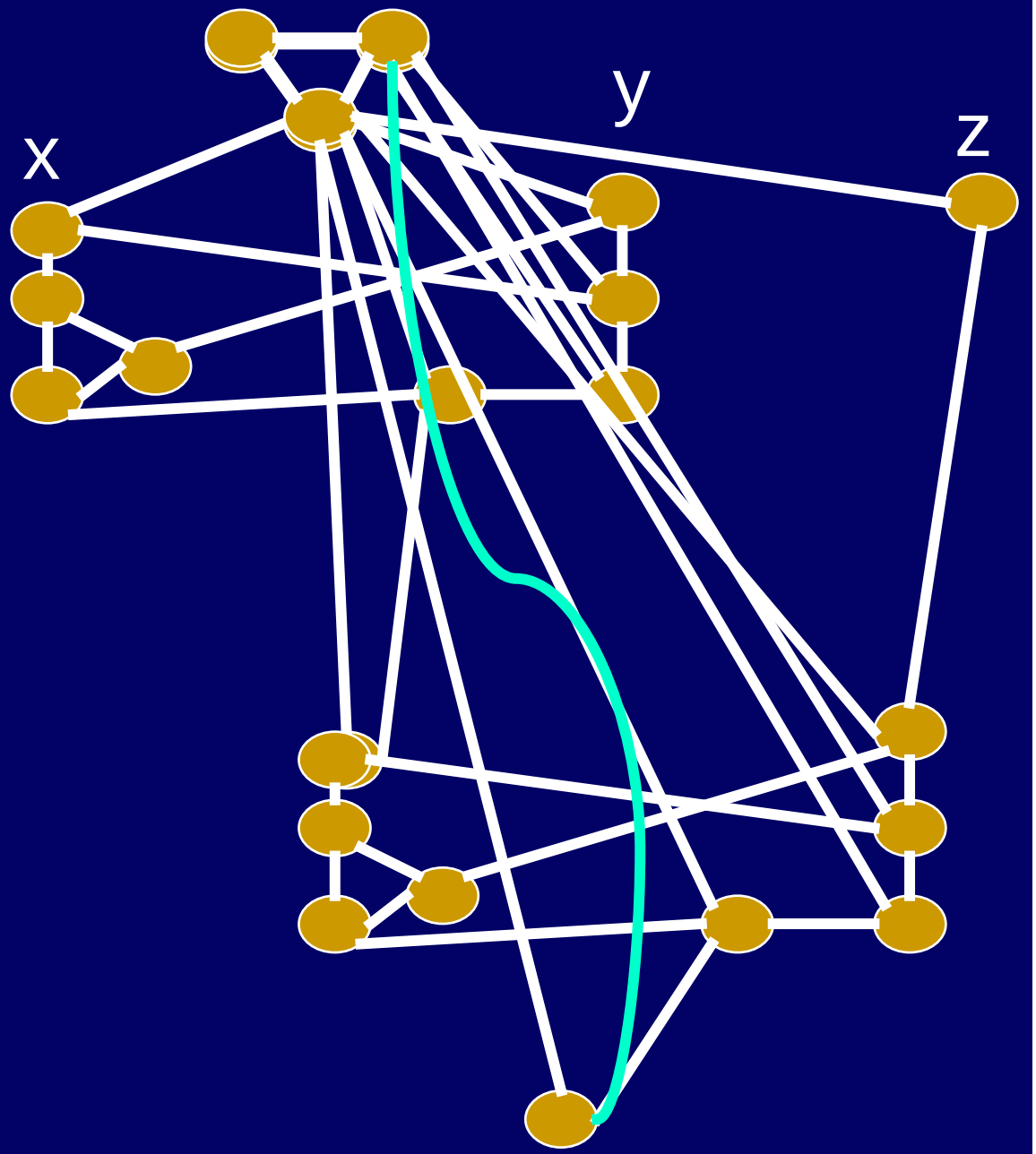

x
y
z
OR
NOT
OR
x
y
z





How do we force the graph to be 3 colorable exactly when the circuit is satifiable?





Satisfiability of this circuit = 3-colorability of this graph

You can quickly transform a method to decide 3-coloring into a method to decide circuit satifiability!