# Welcome to CS 126!

# COS 126 Lecture 1: Introduction

Introductory survey course

- no prerequisites
- basic principles of computer science
- learn to use computers effectively
- check FAQs on web

Topics introduced:

- hardware and software systems
- programming in C and other languages
- algorithms and data structures
- theory of computation
- applications to solving scientific problems

```
#include <stdio.h>
main()
   {
      printf("This is a C program\n");
   }
```

Q. How did the computer scientist die in the shower?

A. The instructions on the shampoo said "Lather, Rinse, Repeat"

# Outline

- **Administrivia**
- What is "computer science"?
  - What it's not
  - Why we learn it
  - Syllabus (long answer)
- An example
  - A simple machine
  - "Science" behind it
- Conclusion
  - CS is about abstractions (short answer)

# The Usual Suspects

- Randy Wang (rywang@cs)
- Lisa Worthington (lworthin@cs)
- Spyridon Triantafyllis (strianta@cs)
- Jie Chen (jennifer@cs)
- Petru Chebeleu (chebeleu@cs)
- Ben Gum (gum@cs)
- Alexey Lvov (lvov@math)

# To Get Started

- Visit course web page:
  - **`http://www.cs.princeton.edu/courses/cs126`**
- Get course packet from Pequod (ready by 9/22?):
  - for more general information
- Go to lab tomorrow (9/17, 10-11:50, 1:30-3:30, CS101)
  - to get on-line
- Decide which precept to go to
  - visit course page for preceptor assignment
  - contact tmhill@cs to make time changes
- Go to precept on Monday (9/20)
  - to get remaining questions answered

# COS 126 Survival Guide

- Participate in precepts
  - Friday: programming assignments/review
  - Monday: quizzes/exercises

- Keep up with the course materials
  - read over handouts when you get them
  - www.CS.Princeton.EDU/courses/cs126
  - prepare for precepts

- Keep in touch
  - mail
  - office hours
  - after class

preceptors

- Use the simplest tool that gets the job done

- Understand your program
  - what would the machine do?
  - find the first bug
  - develop programs incrementally
  - plan multiple lab sessions

- Ask for help when you need it

find your niche

# Tips

- "CS126 survival guide"
- More...
  - Come to lectures and precepts
  - Do readings, exercises, as well as program assignments
  - Find a "system" that works best for you
  - Read, understand, and borrow from example code before writing your own

# Outline

- ~~Administrivia~~
- **What is "computer science"?**
- An example
- Conclusion

# What Is CS?

- (Why don't we call chemistry "test tube science"?)
- What CS is not
  - CS is not programming, just as
  - Biology is not about learning to use a microscope
  - Programming is merely a tool
- Why we learn it
  - Appreciate underlying principles and limitations
  - "Meta-learning": learning how to learn
- What is it?
  - Syllabus (long answer)

# Lecture Outline

## INTRODUCTION (1 lecture)

I1. Abstract machine example

## PROGRAMMING FUNDAMENTALS (7 lectures)

P1. C
P2. Unix
P3. Arrays/structs/lists
P4. Card game example
P5. ADTs
P6. Recursion
P7. Trees

*Take it a for a spin in the parking lot.*

*Going to traffic school.*

## ARCHITECTURE (5 lectures)

A1. TOY
A2. TOY/simulator
A3. Boolean logic
A4. Sequential circuits
A5. Machine organization

*Opening up the hood.*

*Hot-wiring a car.*

*Making your own car.*

# Lecture Outline (continued)

## THEORY OF COMPUTATION (6 lectures)

T1. REs and FSAs
T2. Turing machines
T3. Formal languages
T4. Computability
T5. Algorithms/Complexity
T6. NP-completeness

*Fundamental laws and mathematics.*

## SYSTEMS (5 lectures)

S1. Java
S2. Java/Graphics
S3. Compilers
S4. Operating systems
S5. Applications

*Driving buses-- big "systems" to provide serivices.*

## REVIEW (1 lecture)

R1. History/Course review

*Model-T's.*

# Outline

- ~~Administrivia~~
- ~~What is "computer science"?~~
- **An example**
  - How to make a simple machine
  - What we can do with it
  - "Science" behind it
- Conclusion

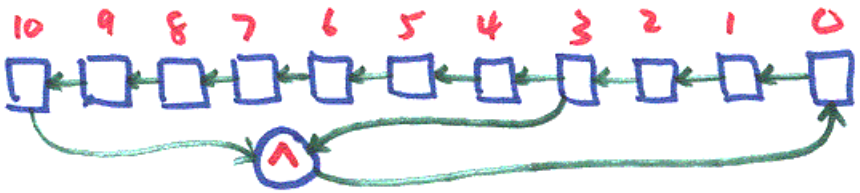
# A Simple Machine

- Want
  - a machine that outputs a random sequence of 0s and 1s
- Some basic terms
  - a bit: a student who's either male or female
  - a storage element (cell): a seat that can hold one student
  - a register: a whole row of seats
  - a shift register: when clock strikes, stand up and take the seat to your right
  - a "linear feedback shift register": ...

# An Abstract Machine

Simple abstract computional device

Linear feedback shift register (LFBSR)




Machine consists of 11 BITS, or 0-1 values
Bit values change at discrete time points
Bit values at time T+1 completely determined
by values at time T

| T | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 10^3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T+1 | ↓ | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

"XOR" of two bits (addition mod 2)
1 if different; 0 if same

| a | b | a^b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$a \wedge b = (a+b) \bmod 2$

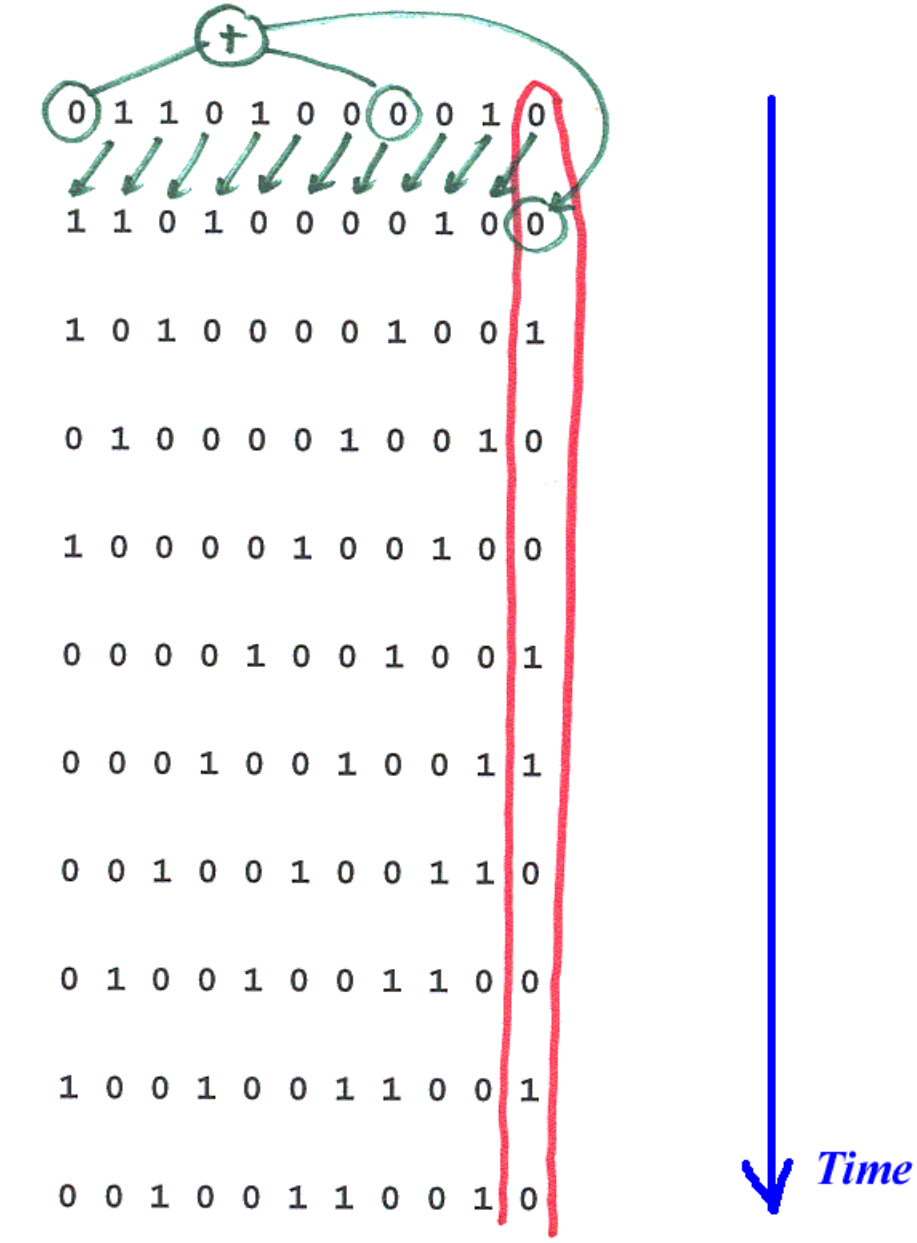
Magic properties:

b^b = 0

a^0 = a

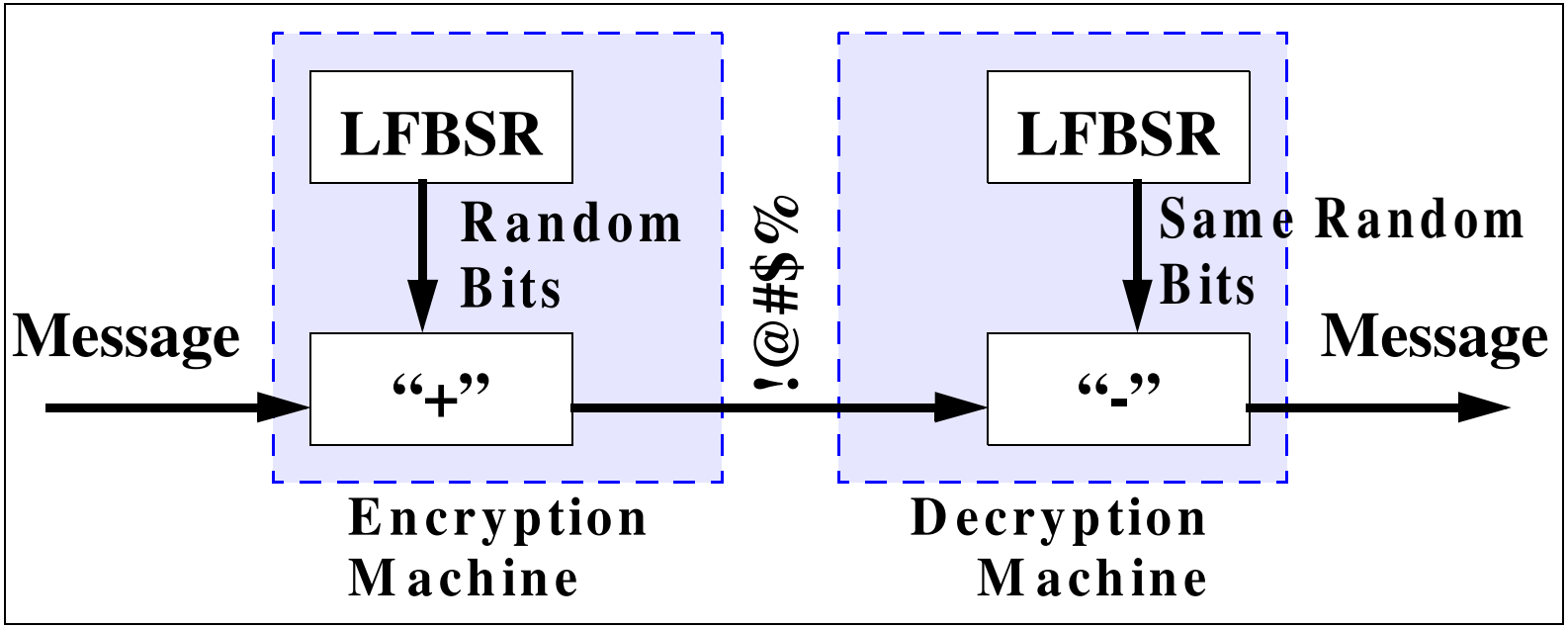
(a^b)^b = a^(b^b) = a^0 = a

# LFBSR example




Bits "look" random (but aren't!)

# What Is It Good For?
# Message Encryption



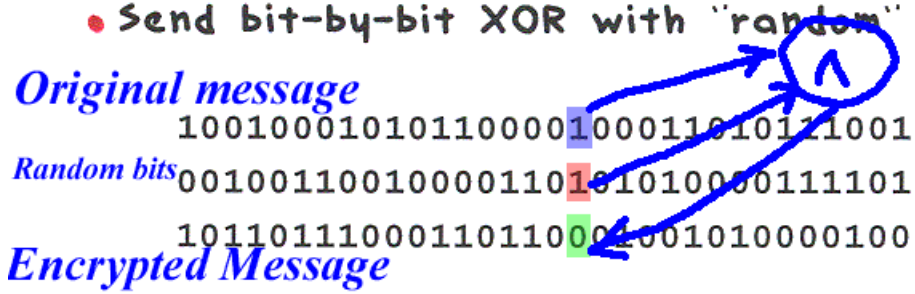

- Use LFBSR as a component in an encryption/decryption machine
- Cool detail: "+" and "-" can be xor; so same machines!

## Using "Random" Bits for Encryption

- Convert message to bitstream

```
  S     E     N     D     M     O     N     E     Y
01010011|01000101|01001110|01000100|01001101|01001111|01001110|01000101|01011001
```

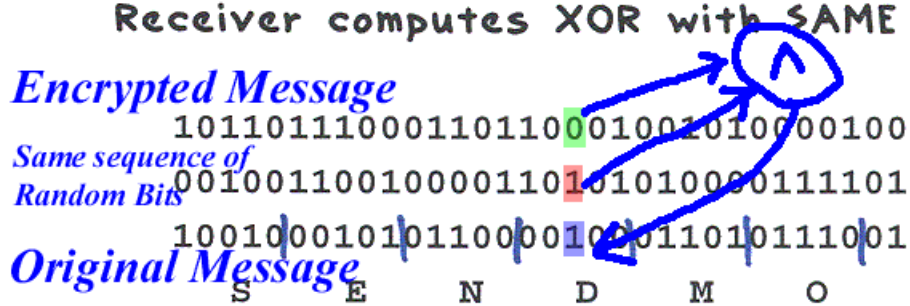
- Send bit-by-bit XOR with "random" bitstream




- Message looks random to anyone reading it

```
01110111000110111000100101000010011000010111100
  W     ?     M     R     E     A     F     B     Z
```

- Receiver has identical machine
  (Secretly) provide receiver with initial fill
  Receiver computes XOR with SAME "random" bits1




- Works because (a ^ b) ^ b = a ^ (b ^ b) = a

# Now the "Science" Behind It

- Are the bits really "random"?
- How long would it take before the bit pattern repeat itself?
- Will the machine work equally well if I xor the 10th and ***4th*** bits?
- How many cells do I need for my LFBSR if I want to guarantee a certain degree of security?

## Properties of shift register "machine"

- Clocked

  Control: start, stop, or "load"
  Data: initial values of bits (fill)

- Built from very simple components
  - "clock" (regular electrical pulse)
  - electrically controlled shift register cell
    remembers value until clock "ticks"
  - some wires "input", some "output"

- Scales to handle huge problems
  10 cells yields 1 thousand random bits
  20 cells yields 1 million random bits
  30 cells yields 1 billion random bits

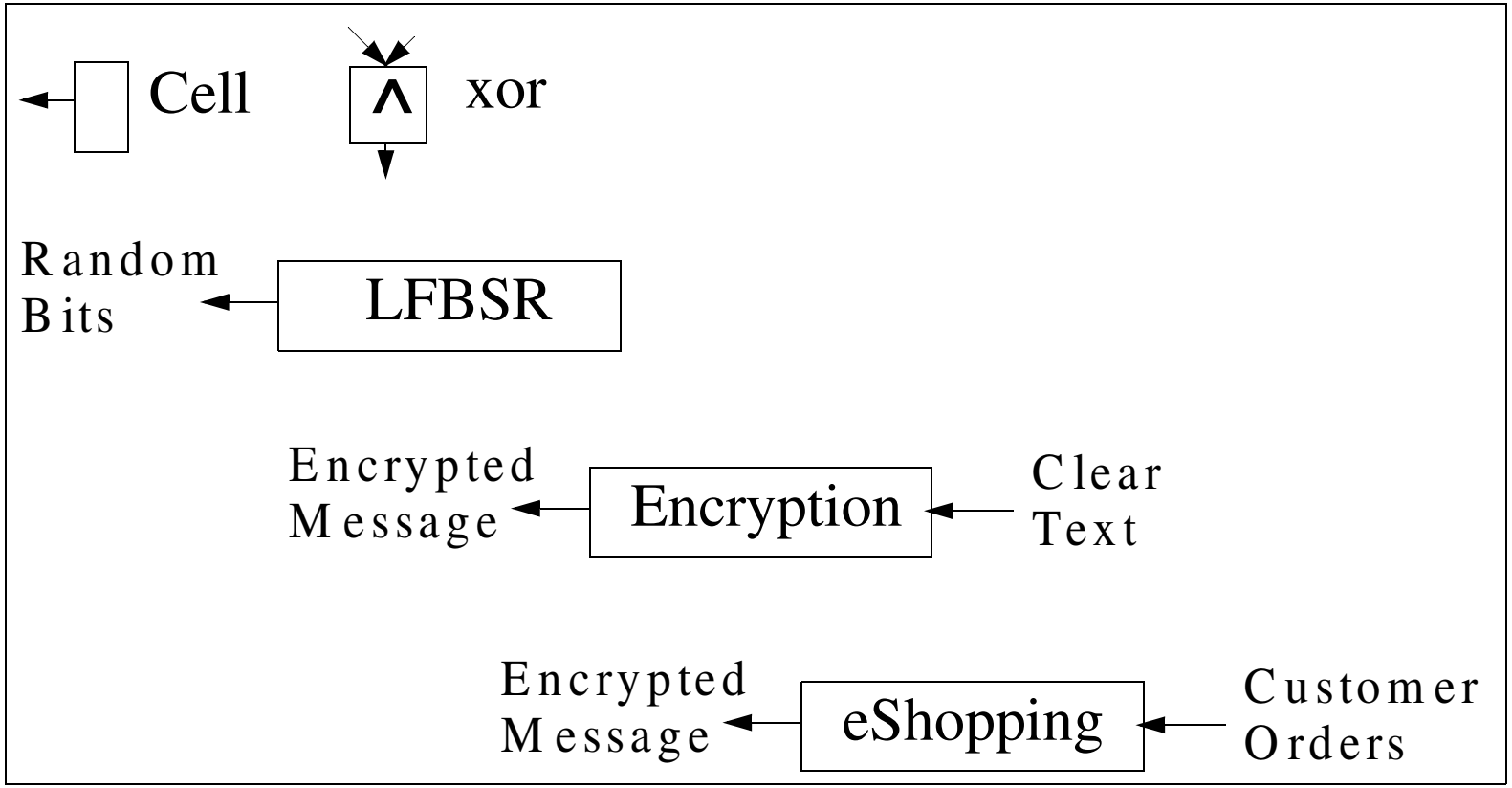
- BUT, need to understand abstract machine!
  (higher math needed to know XOR taps)

Same basic principles used for computer
- clocked
- all built from switches with feedback
- control, data
- abstraction aids understanding

# Outline

- ~~Administrivia~~
- ~~What is "computer science"?~~
- ~~An example~~
- **Conclusion**
  - CS is about abstractions (short answer)

# Abstractions Involving LFBSR




- Bigger boxes made of smaller ones, hide details behind interfaces
- “Science” at each step for design decisions

# Computer Systems and Abstract Machines

- Layers of abstraction
  - precisely define a simple machine
    use it to build a more complex one
  - develop complex systems by building
    increasingly more complicated machines
  - improve systems by substituting
    new (better) implementations
    of abstract machines at any level

- LFBSR layers of abstraction
  - simple piece of hardware
  - converts fill to "random" bits
  - can use "random" bits for encryption
  - can use encryption for internet commerce

- "Computer" layers of abstraction
  - complex piece of hardware
    CPU, keyboard, printer, storage device
  - machine language programming
  - software systems
    editor (emacs): create, modify files
    compiler (cc): transform program
      to machine instructions
    operating system (Unix): invoke programs
  - windowing system (X):
    illusion of multiple computer systems ...