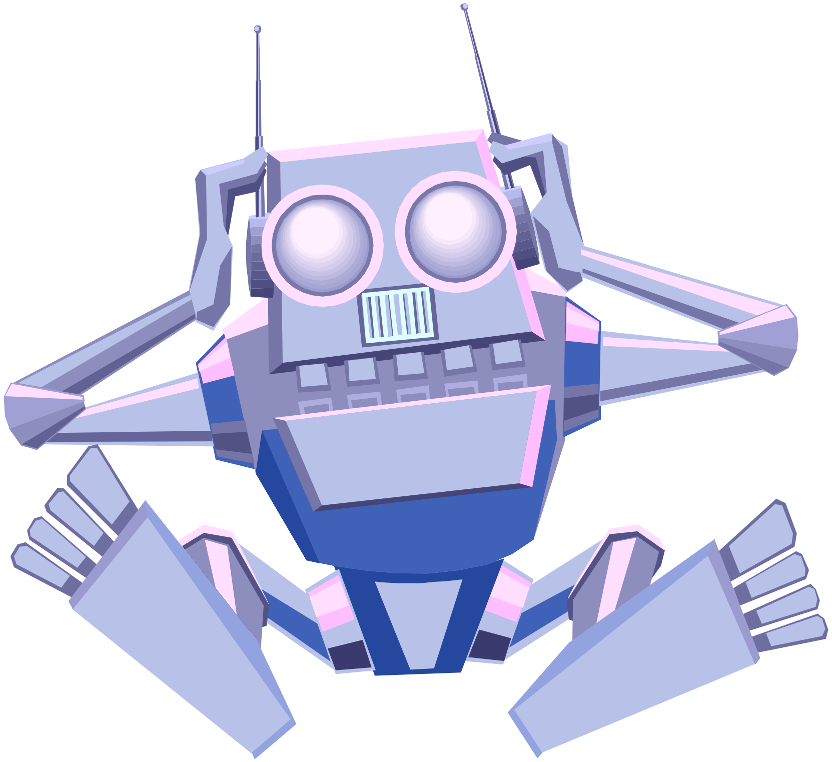
Cellular Automata III

Based mostly on lectures by Dr. Richard Spillman

OUTLINE



What an advanced class!

- Random Number Generation
- A Cellular Automata Cipher System
- Computation on Cellular Automata

Random Number Generation

- Random numbers are required for a wide range of applications
  - Cryptology
  - Testing
  - Modeling and Simulation
  - Genetic Algorithms
  - . . .

- Yet, true random numbers are very difficult to find
  - Computer based random number generators are really ***pseudo-random*** number generators because they eventually repeat

Current Random Number Generators

- There are two typical approaches to random number generation
  - Use of a mathematical relationship
  - Use of a linear feedback shift register (LFSR)

- A common mathematical relationship is of the form

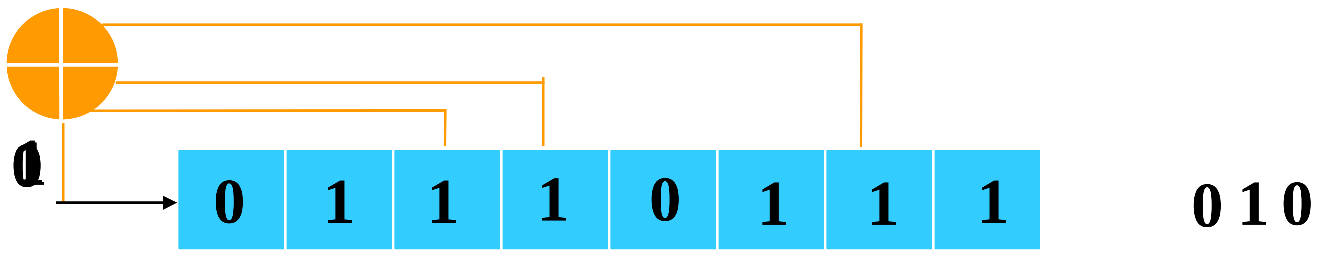
$x' = (ax + b) \bmod n$

$x' = (11x + 17) \bmod 61$

	Random Numbers
Seed	13
	38
	8
	44

Linear Feedback Shift Register

- A LFSR is a hardware random number generator
  - A shift register holds a data word and can shift it to the left or right one bit position on each clock pulse



Add feedback

Load a seed

A CA Random Generator

- The study of random bit generation in cellular automata began with the work of Steven Wolfram
  - He first suggested a 1-d binary cellular automata with a k=2 neighborhood and rule 30

$$\mathbf{q_i(t+1) = q_{i-1}(t)\ XOR\ (q_i(t)\ \ OR\ q_{i+1}(t)\ )}$$

Example

- Consider a simple 1-d CA with an initial seed:

Select one cell to provide a random binary bit

$q_i(t+1) = q_{i-1}(t) \text{ XOR } (q_i(t) \text{ OR } q_{i+1}(t))$

Questions:

When will the patterns begin to repeat?

What is the best seed?

Are there better rules?

1

0 1 1 0 0 0 1 1 0 0 0

0

1 1 0 1 0 1 1 0 1 0 0

0

1 0 0 1 0 1 0 0 1 1 1

1

0 1 1 1 0 1 1 1 1 0 0

0

1 1 0 0 0 1 0 0 0 1 0

0

1 0 1 0 1 1 1 0 1 1 0

GOAL

- The goal of a random number generator is to produce as long a sequence of random numbers (bits) as possible before the pattern begins to repeat itself
  - We are looking for the set of CA parameters that produce a maximum length cycle
  - Wolfram found that a single nonzero element seed produced the maximum length cycle for any rule
  - Wolfram also found that the longest cycles were produced by **rule 30**

Experimental Data

- Maximum cycle lengths for an n-bit CA using **rule 30** and a **single nonzero bit seed**:

N	Maximum Cycle Length
4	8
7	63
11	154
14	1428
24	185,040
32	2,002,272

Hybrid Cellular Automata (HCA)

- **Hybrid Cellular Automata** (also called ***non-uniform*** cellular automata) function the same way as *uniform cellular automata* except that the cellular rules need not be identical for each cell
  - One cell may follow rule 90 and another cell may follow rule 30
  - This results in a new level of cellular automata behavior

Random Numbers w/HCAs

- It turns out that HCAs (hybrid cellular automata) produce maximal-length binary sequences when the right rule set is used
  - One of the better choices seems to be a combination of **rule 90** and **rule 150**

RULE 150

$$\mathbf{q_i(t+1) = q_{i-1}(t) \text{ XOR } q_i(t) \text{ XOR } q_{i+1}(t)}$$

HCA Operation

- If more than one rule is active in a cellular automata then which rule is assigned to which cell?
  - This is usually solved experimentally (or it could be solved by a genetic algorithm)
  - For example, it has been found that a four element hybrid cellular automata with rules distributed as **90 150 90 150** produces a maximal length cycle
    - The notation for this rule distribution is 0 1 0 1 where rule 150 is 1 and rule 90 is 0

Comparison

- The best rule distribution has been determined for some HCAs:

	Maximum Cycle Length	
N	Uniform	HCA
4	**8**	**15**
7	**63**	**127**
11	**154**	**2,047**
14	**1428**	**16,383**
24	**185,040**	**16,777,215**
32	**2,002,272**	$\mathbf{2^{32}-1}$

CA-Based Cipher Systems

- The need for **security** and **privacy** has become a given in today's internet connected world.

- One of the major tools for security is the use of **encryption algorithms** to hide data and messages

- Cellular automata can play a significant role in the construction of **new fast, secure and efficient ciphers**

Cipher Systems

- The process of disguising a message in such a way as to hide its substance is called ***encryption***
  - **a message is called** ***plaintext***
  - the encrypted message is called ***ciphertext***
  - the process of turning ciphertext back into plaintext is called ***decryption***

- The art and science of keeping messages secure is called ***cryptography***
  - ***cryptanalysis*** is the *art and science* of <u>breaking ciphertext</u>

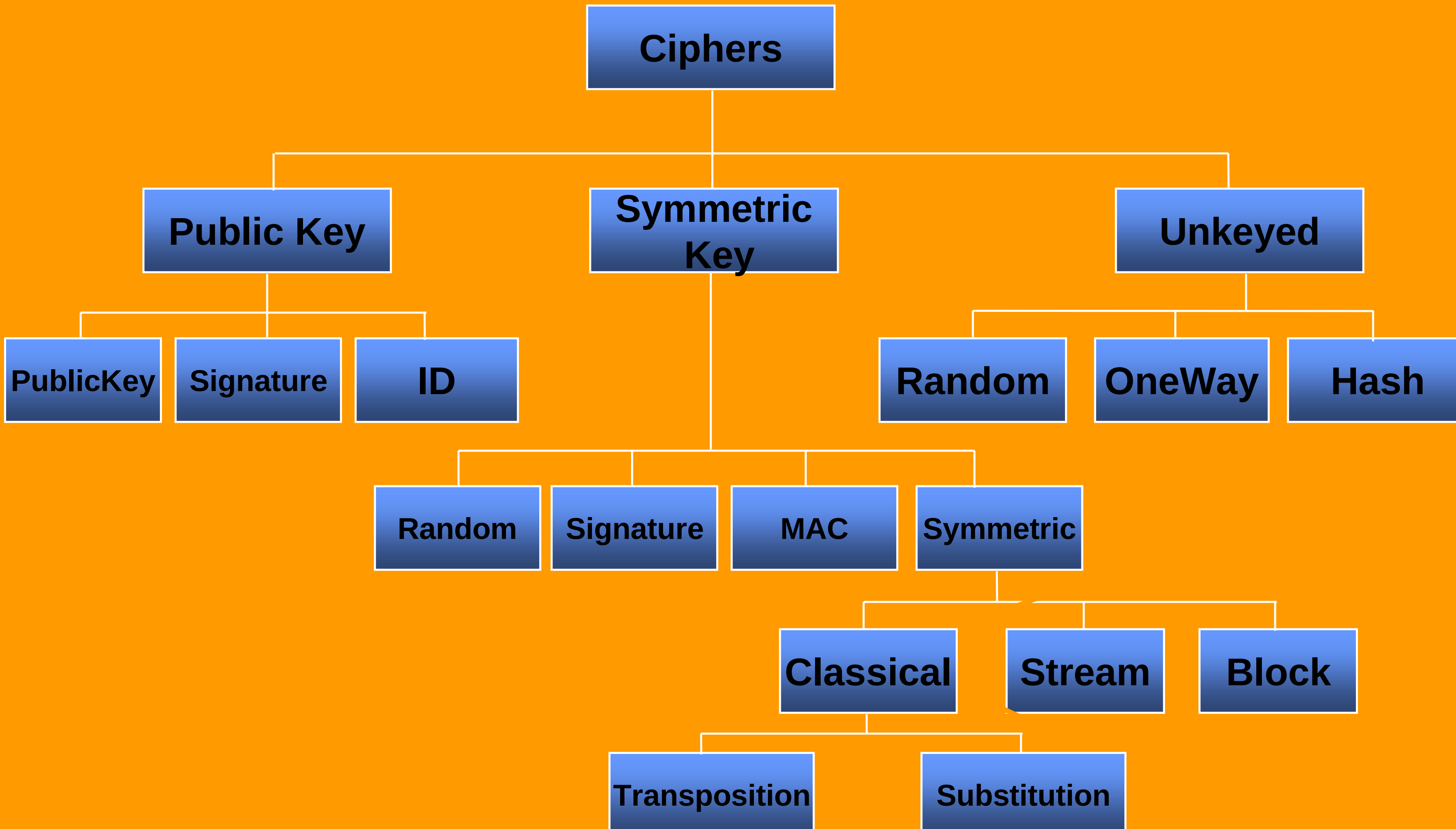
A Good Cipher

- Enciphering and deciphering should be efficient for all keys - it should not take forever to get message.

- Easy to use. The problem with hard to use cryptosystems is that mistakes tend to be made

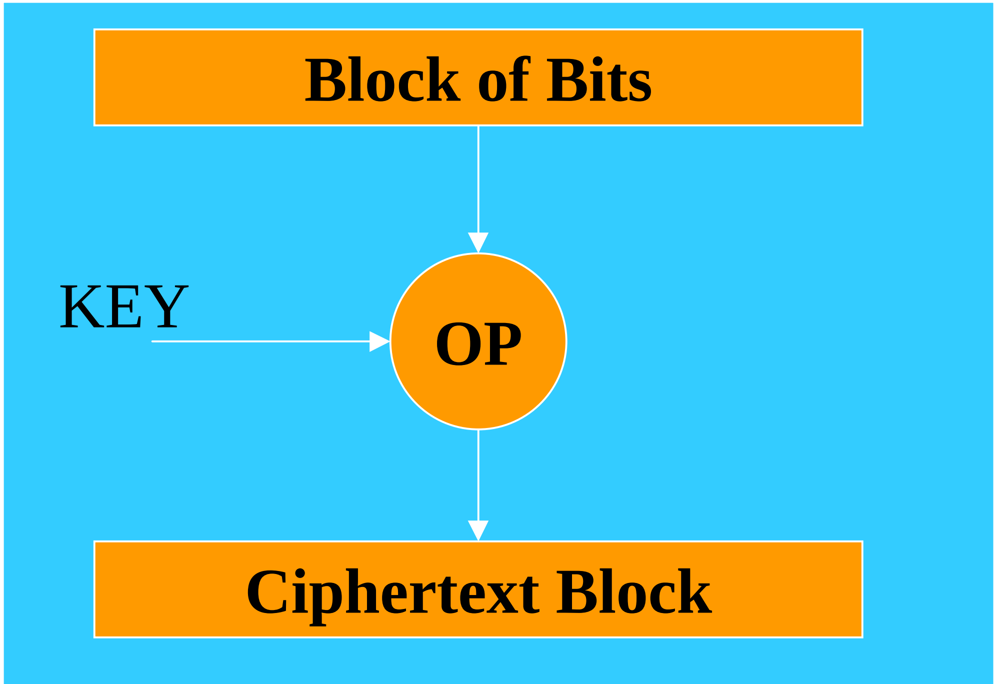
- The strength of the system should not lie in the secrecy of your algorithms.
  - The strength of the system should depend the **secrecy of your key.**

Cipher Classification



Block Ciphers

- Today's most widely used ciphers are in the class of Block Ciphers
  - Define a block of computer bits which represent several characters
  - Encipher the complete block at one time



Block Cipher Methods

- Plaintext is divided into fixed length blocks M_1, M_2, . . . , M_m and <u>each block is transformed into</u> ciphertext so the entire ciphertext is given by C_1, C_2, …, C_m

- Block size should be large
  - usually it is 64 bits

CA Block Cipher

- The problem with any random number generator including a CA-based random number generator is that eventually it will cycle back to the beginning
  - Hence, the design of a random number generator required finding a rule which made the cycle as long as possible
  - For a cipher, we want to take advantage of the cycle feature

CA Cycles

- For a cipher, find a CA rule that:
  - Produces a short cycle of length 2r
  - The rule becomes the key
- There are three rules called **fundamental transformations** that meet the necessary requirements for a block cipher – rules **51, 153, and 195**

195: $q_i(t+1) = \mathrm{NOT}(q_{i-1}(t) \text{ XOR } q_i(t))$

153: $q_i(t+1) = \mathrm{NOT}(q_{i+1}(t) \text{ XOR } q_i(t))$

51: $q_i(t+1) = \mathrm{NOT}(q_i(t))$

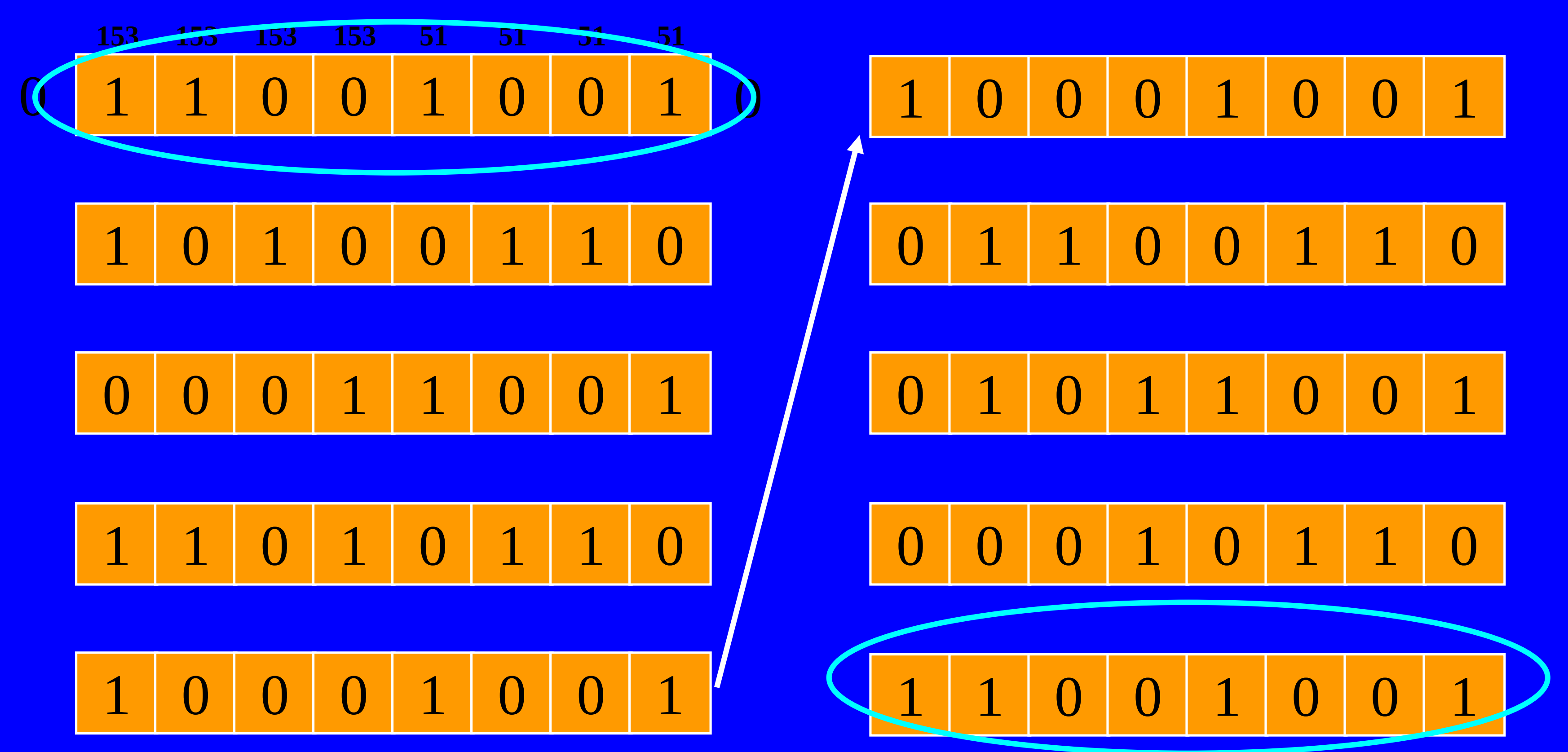
Hybrid CA

- A cellular automata cipher requires a hybrid system with some combination of the 3 fundamental rules.

- For example, consider a null boundary, 8-bit CA with rules applied to the cells in this order:
  - (153,153,153,153,51,51,51,51)
  - This has a cycle of length 8, so run it for 4 clock ticks and send the result
  - The result is decoded by completing the cycle by running the ciphertext for an additional 4 clock ticks

Exampleof a hybrid system with some combination of the 3 fundamental rules.

- Start with an 8-cell CA with null boundaries
  - That is, each end is connected to 0 instead of each other



Key Space

- For a CA Block Cipher, the key space is the space of fundamental transformations

- For the example on the prior slide other possible transformation patterns include:
  - (195, 195, 195, 195, 51, 51, 51, 51)
  - (51, 51, 153, 153, 153, 153, 51, 51)
  - (51, 153, 153, 153, 153, 153, 153, 51)

Stream Cipher

- Consider the plaintext as a sequence of bits
  - Generate a random sequence of bits for the key
  - Form the ciphertext by a bit by bit XOR of the *plaintext with the ciphertext*

plaintext:	1	0	0	1	0	1	1	0	0	1	1	1	0	1	0	1	0	0	1
key:	0	0	1	1	1	0	1	0	1	0	0	1	1	0	0	1	1	0	0
ciphertext:	1	0	1	0	1	1	0	0	1	1	1	0	1	1	0	0	1	0	1
key:	0	0	1	1	1	0	1	0	1	0	0	1	1	0	0	1	1	0	0
plaintext:	1	0	0	1	0	1	1	0	0	1	1	1	0	1	0	1	0	0	1

Problem: **How do we recover the plaintext from knowledge of the ciphertext and key?**

Problem

- A short sequence of key bits would be easy to remember but not very secure

- A long sequence of key bits would be secure but hard to remember

- **PROBLEM:** How can we generate a long random-appearing sequence of 0's and 1's in way that will insure that everyone who should have access to the plaintext are able to generate the key when needed?

- **ANSWER:** Construct a random bit generator

Stream Cipher Key

- For a CA-based stream cipher, the two parties must have the same CA with the same initial condition.
  - Hence, the key is the initial state of the CA
  - With the key, the person who receives the ciphertext can start the CA random number generator and produce the correct key stream

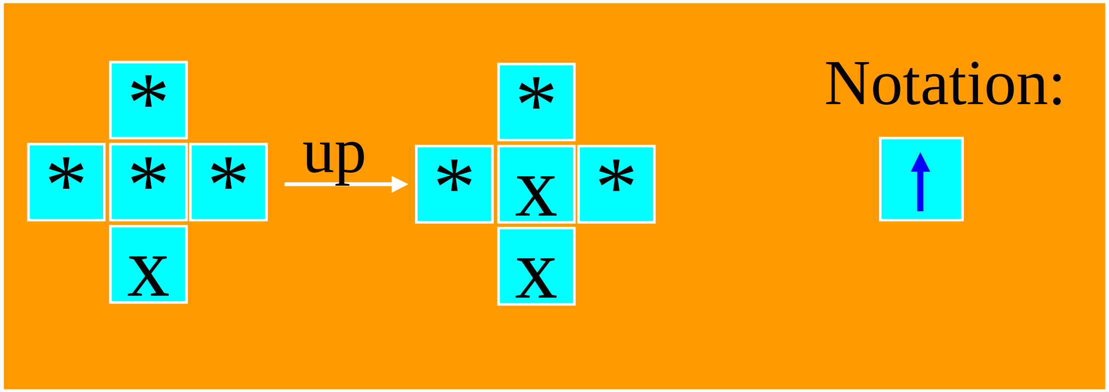
Computation with CAs

- It is possible to design a complete circuit on a cellular automata
  - The equivalent of a digital circuit embedded in the cellular automata in the form of rules

- Create a large 2-d binary, von Neumann neighborhood hybrid cellular automata
  - It will have pathways for signal transmission (wires)
  - It will have functional cells that perform NANDs, XORs, …

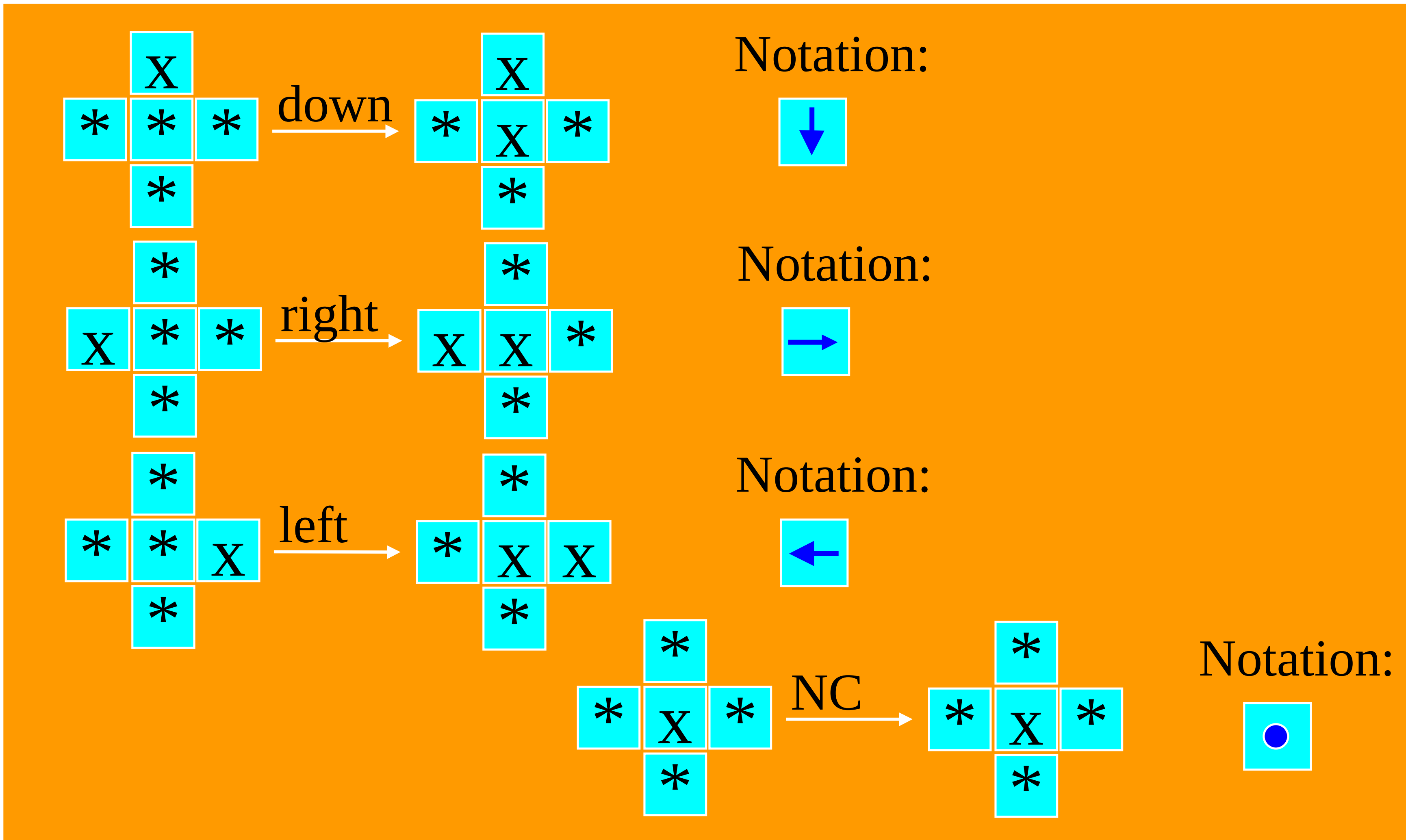
Signal Pathways

- A "wire" on our cellular automata consists of a path of connected **propagation cells**
  - Each of these cells implements a propagation rule
  - There are four propagation rules: right, left, up, down
  - For cells which are are not part of the circuit there is a NC (no change) cell



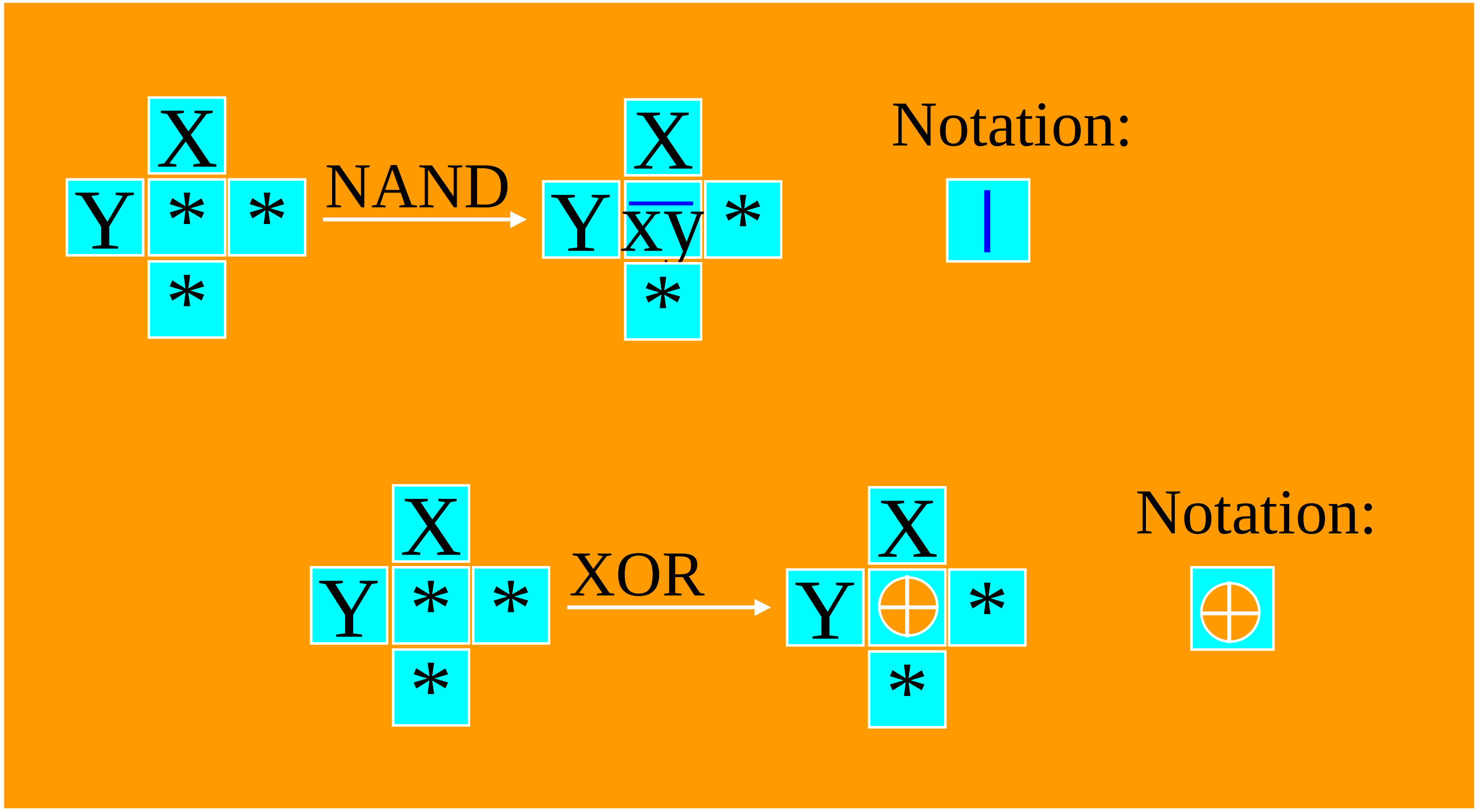
Signal Propagation Rules

- The other propagation rules are given by:



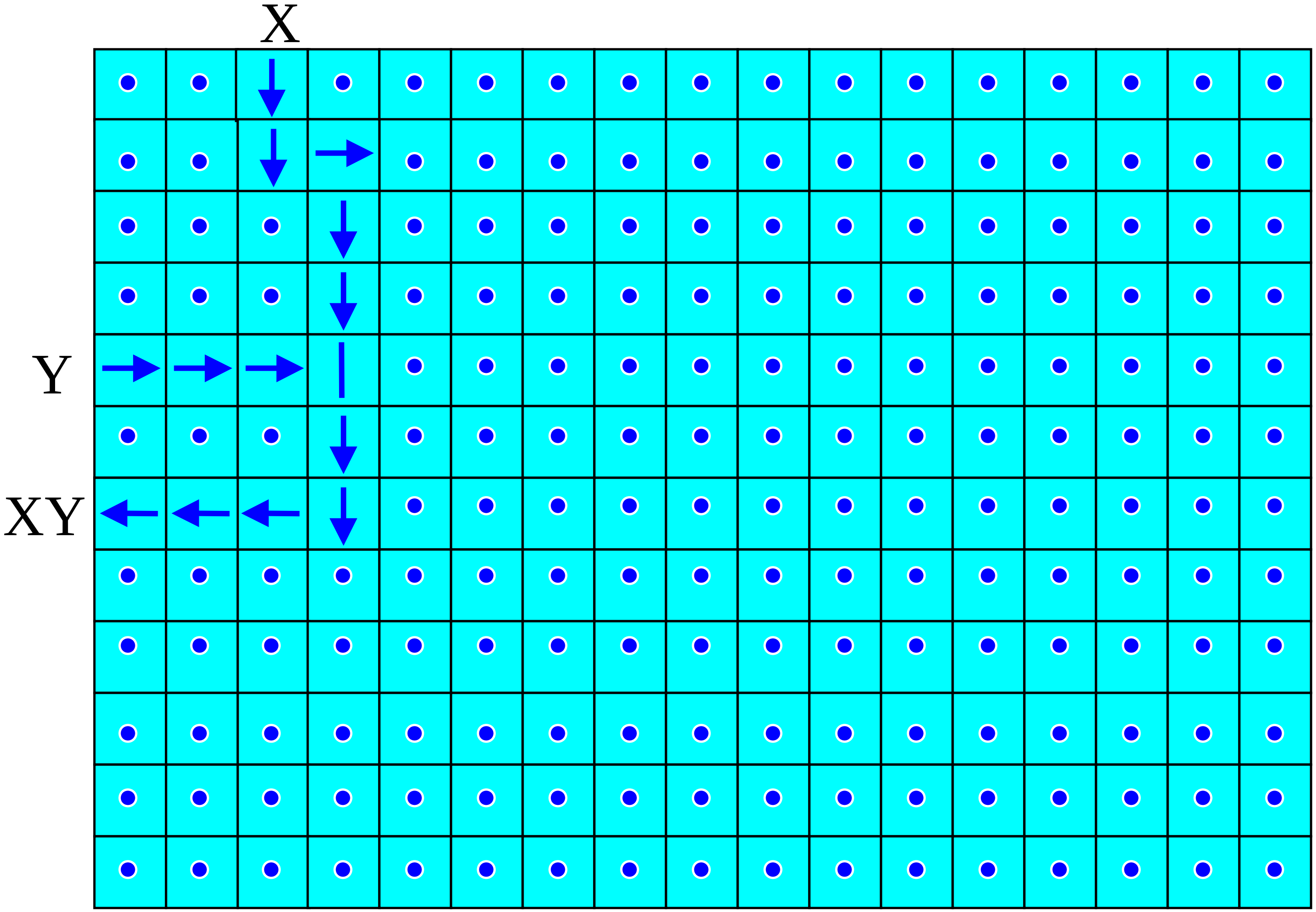
Logic Operations Rules

- Rules for both **NAND** and **XOR** are given by:

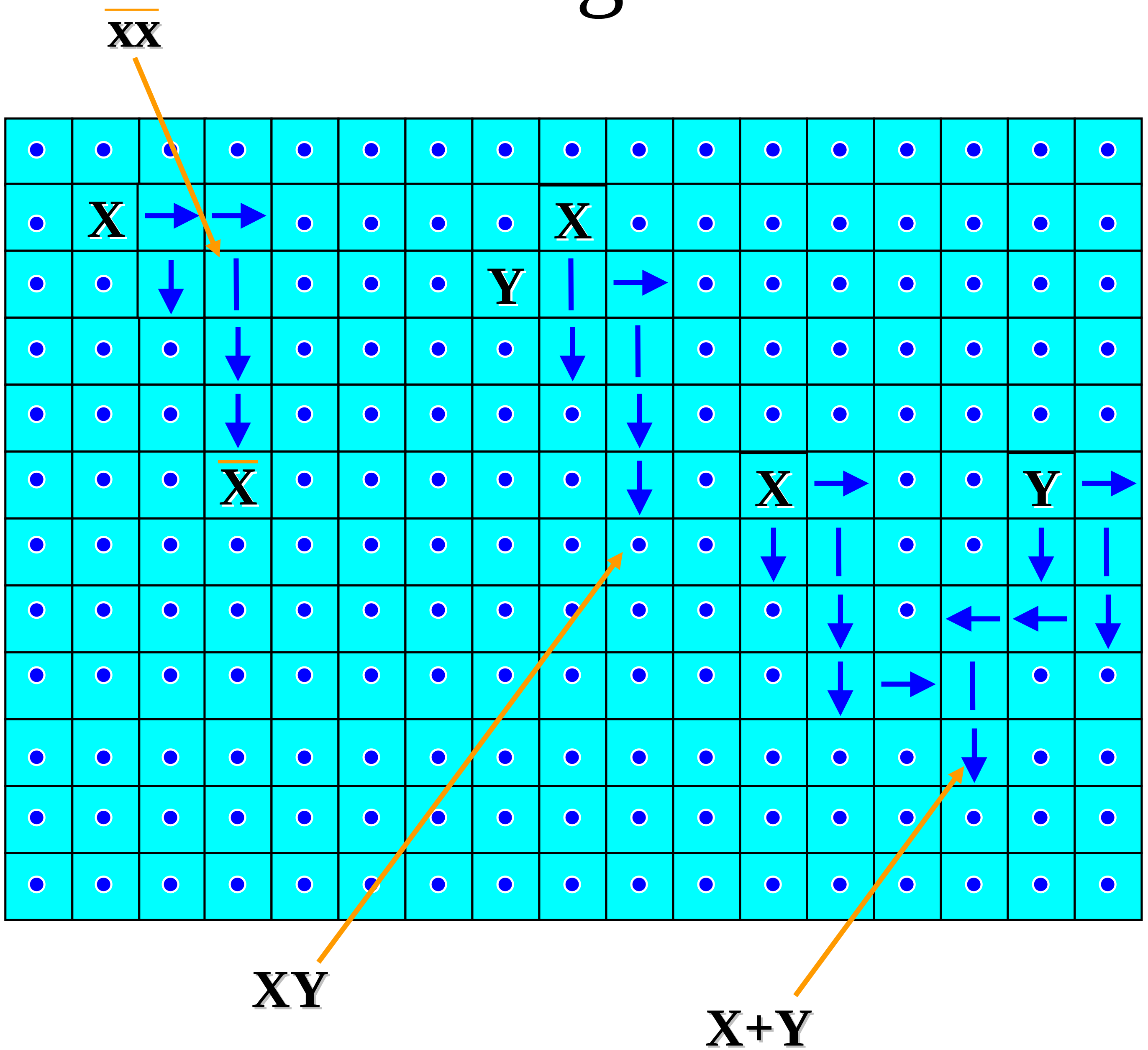


Example

- Given a 2-d cellular automata:



Other Logic Functions



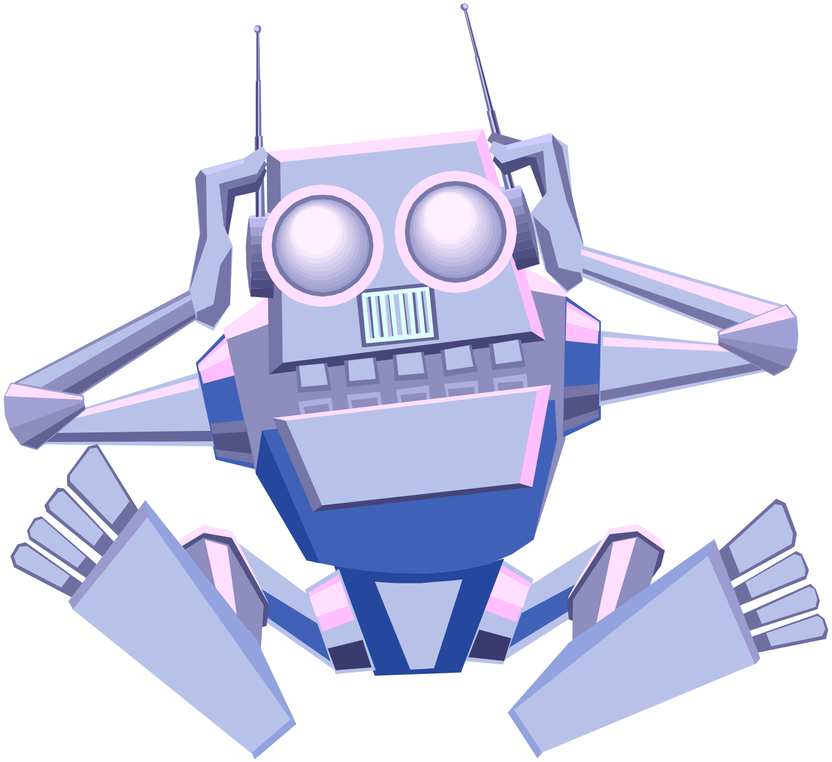
Possible Homeworks

- You might want to look at the application of CAs to random number generation
  - For a possible senior project, consider other pairs of rules or even 3 or more rules in an HCA
  - For a possible advanced homework project, build a genetic algorithm to create a 2 or 3 rule HCA
- Construct a simple CA cipher system
- Design a simple digital circuit on a CA

Reversible Cellular Automata

Dr. Richard Spillman

OUTLINE



What an advanced class!

- Reversible Computing
- Reversible Cellular Automata
- Billiard Ball Model of Computation

Reversible Computing

- A reversible computing is a backward deterministic system
  - Each state of the system has at most one predecessor
  - The result is that its computation path can always be retraced

- It turns out that reversible computing does not consume energy or produce heat
- **Reversible Computing** is a general Computer Science Idea that is more general than Reversible Logic Circuits that we discussed in the past

Reversible Cellular Automata

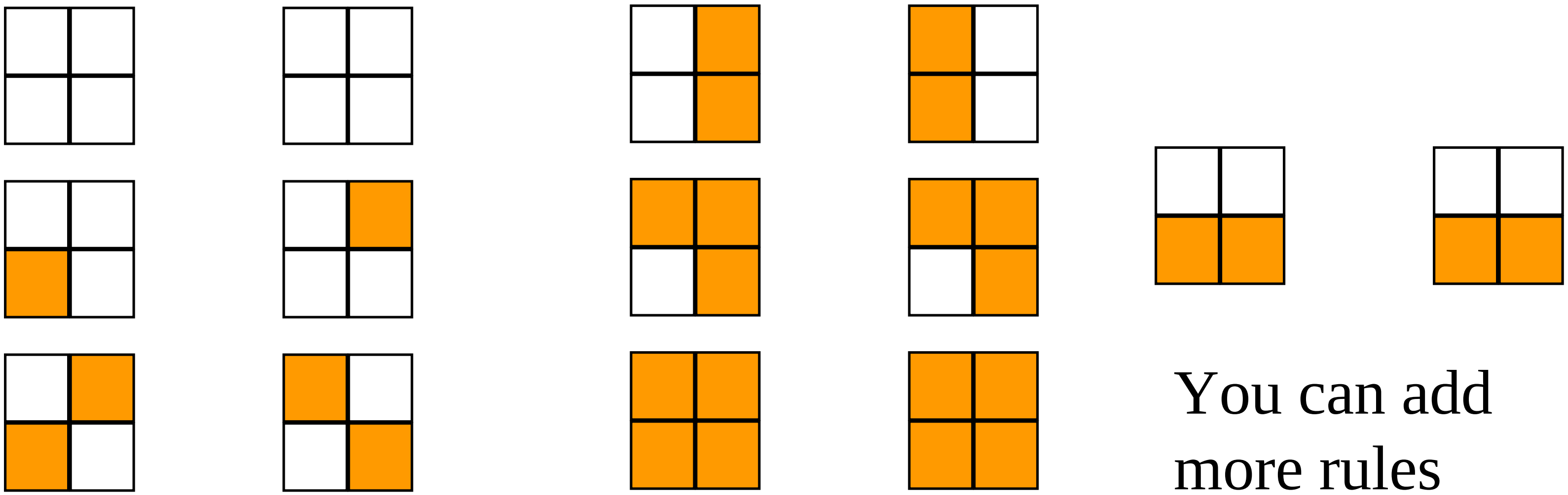
- A reversible cellular automata (**RCA**) is a cellular automata for which each state has ***at most one predecessor***
  - Given any current state it is possible to trace it back to its initial state

- An RCA can be implemented that does not require any cooling or energy (in theory)
  - The rules could be implemented using Fredkin gates

Example of RCA Rules

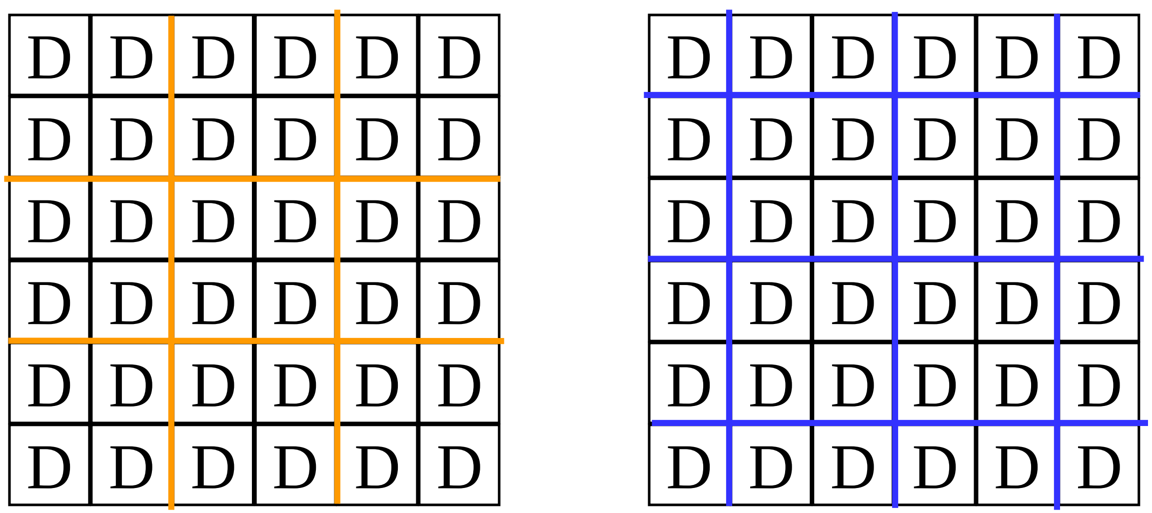
- For our example class of RCA, the **neighborhood** will consist of 2x2 blocks of cells in a 2-d cellular automata

- The transformation rules are:



Blocking

- With this new neighborhood definition comes another alteration in standard cellular automata structure
  - The 2x2 cells are blocked into two different sets
  - The rules are applied in an alternating cycle to the two different blocks



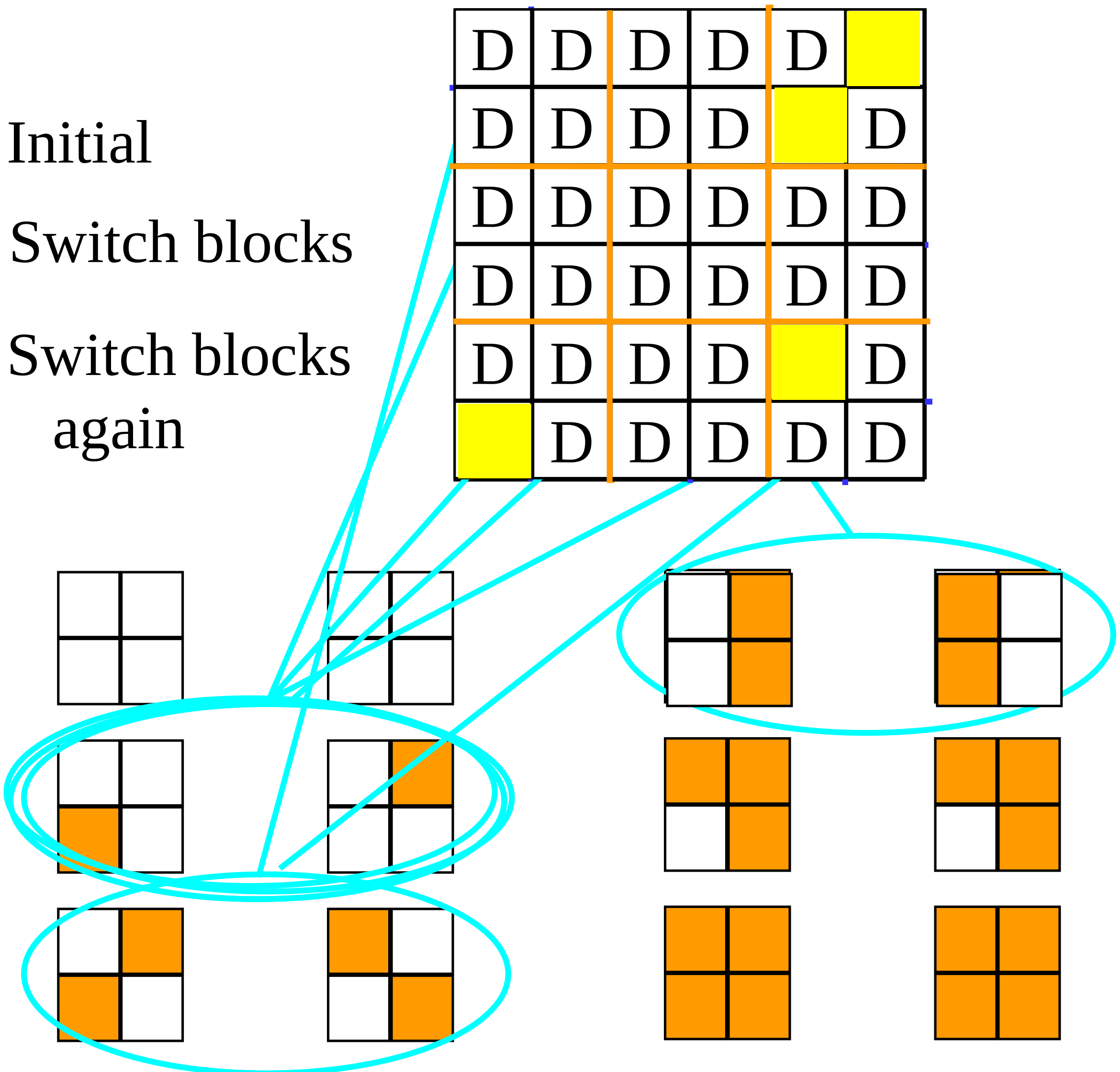
Margolus Neighborhood

Example

Initial

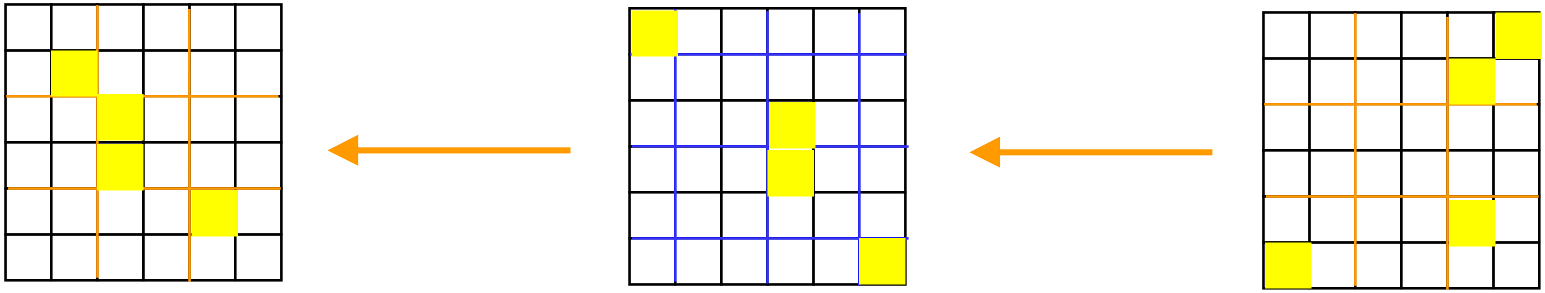
Switch blocks

Switch blocks again



Reverse

- Given the beginning and ending positions of the prior slide show that it run backwards

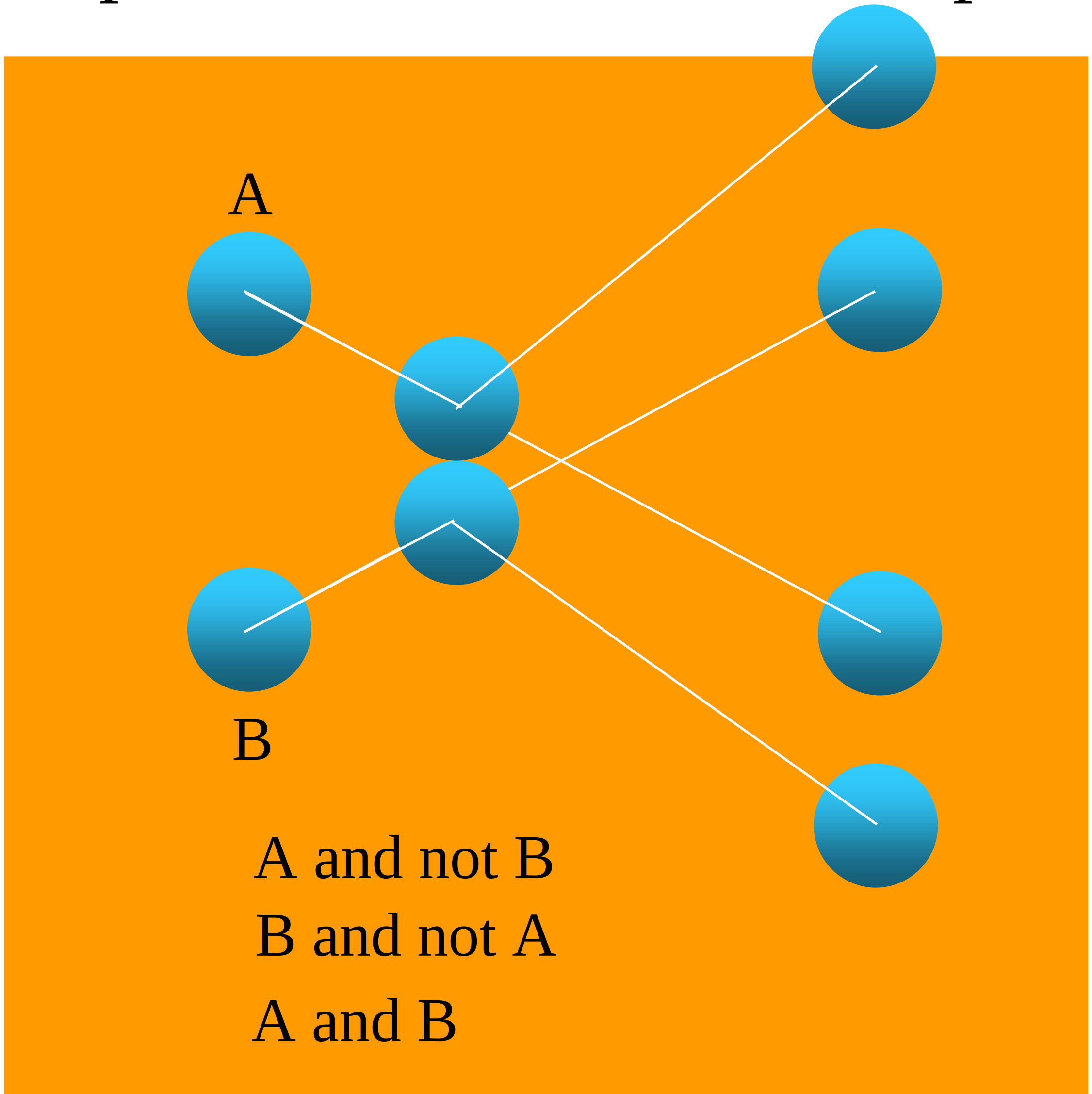


Reminder: A Billiard Ball Model of Computation

- Create a model of computation based on the motion of particles
  - Billiard Ball Model (BBM)
  - Run on a reversible cellular automata
  - The position of a billiard model is indicated by a 1 in a cell
  - Every place where a collision of finite-diameter hard spheres might occur can be viewed as a logic gate

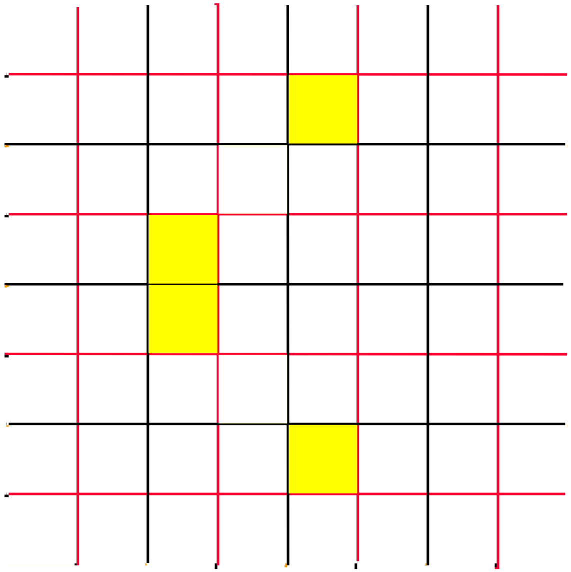
Example

- We can put billiard balls at either points A, B, or both



Cellular Automata Behavior

- A collision looks like:



This type of system is used to model the classical behavior of many particle systems in physics

Possible Homeworks with CA Focus

- Explore the nature of reversible computing
  - Why it is necessary
  - How is it implemented
- Build a BBM and explore the different behavior patterns