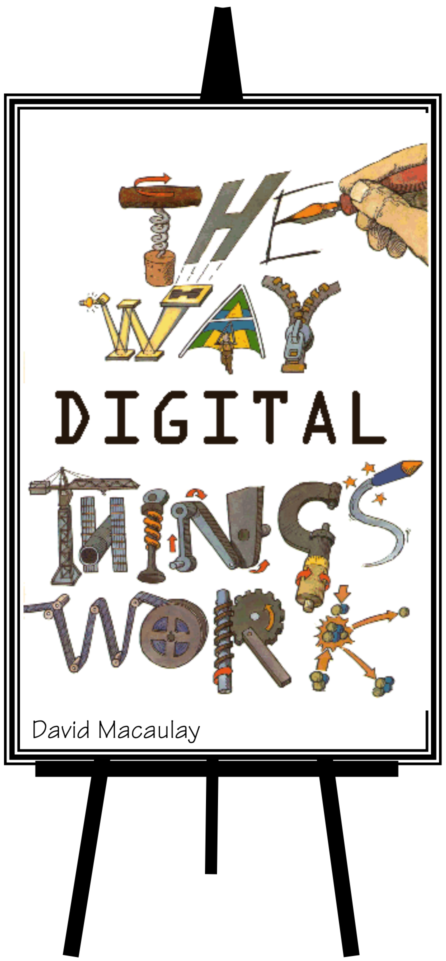
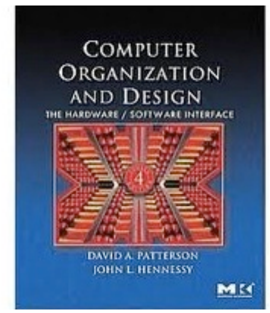
Welcome to Comp 411!



I thought this course was called
"Computer Organization"

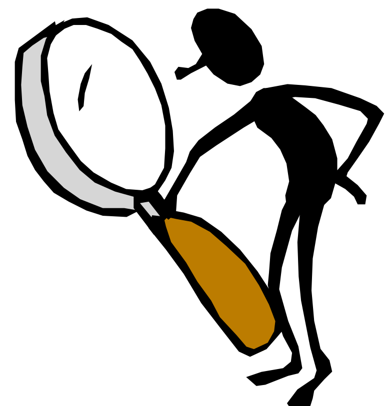
1) Course Mechanics
2) Course Objectives
3) Information

Meet the Crew…



Lectures:	Leonard McMillan (SN-311)
	Office Hours Th 2-3
TA:	David Wilkie (SN-008)
Book:	Patterson & Hennessy <u>Computer Organization & Design</u> 4rd Edition, ISBN: 1-55860-604-1
	(However, you won't need it until next week)

Course Mechanics



Grading:

Best 5 of 6 problem sets	25%
Best 9 of 10 Labs	25%
2 Quizzes	30%
Final Exam	20%

You will have at least two weeks to complete each problem set. Late problem sets will not be accepted, but the lowest problem-set score will be dropped.

Lab (COMP 590-411) is mandatory, and will meet on most Fridays, grade is based on completing a "lab check list."

Quizzes are multiple choice and will be given during the lab period.

I will attempt to make Lecture Notes, Problem Sets, and other course materials available on the web before class on the day they are given.

Comp 411: Course Website

Logged in as: *mcmillan* Log out

Home | Research | Courses | Publications | Setup

Announcements

- The first lecture will be held on January 9, 2012.

Announcements, corrections, etc.
On-line copies of all handouts
Download Problem Sets

Course Description

Comp 411, *Computer Organization*, explores the topic of how computers work, in terms of both software and hardware. It covers a wide range of topics including what a *bit* is, and why bits are the atoms in the universe of computation. We also discuss how information is represented and processed in hardware, and arrive to the conclusion that, to a computer, everything is data, including the instructions that underly software.

Comp 411 also covers the wide range of languages, and layers of translation, used for computation--

http://www.cs.unc.edu/~mcmillan/Comp411S12

Goal 1: Demystify Computers

Strangely, most people (even some computer scientists I know) are afraid of computers.

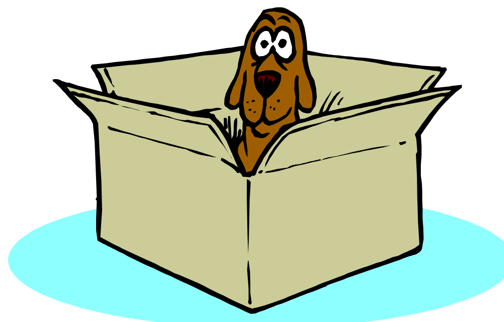
We are only afraid of things we do not understand!

I do not fear computers. I fear the lack of them.
- Isaac Asimov (1920 - 1992)

Fear is the main source of superstition, and one of the main sources of cruelty. To conquer fear is the beginning of wisdom.
- Bertrand Russell (1872 – 1970)

Goal 2: Power of Abstraction

Define a function, develop a robust implementation, and then put a box around it.

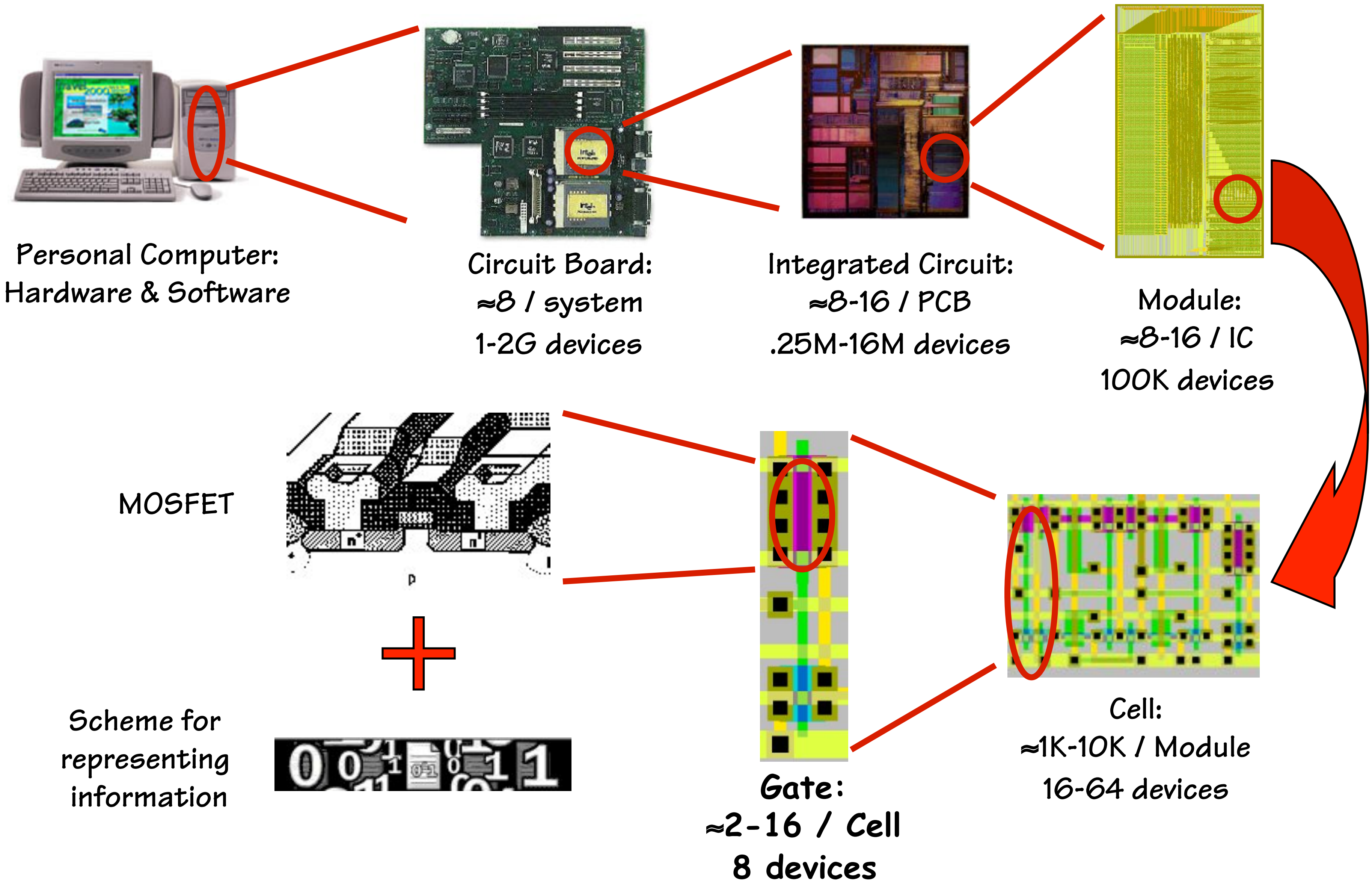


Abstraction enables us to create unfathomable systems (including computer hardware and software).

Why do we need ABSTRACTION…

Imagine a billion --- 1,000,000,000

The key to building systems with >1G components



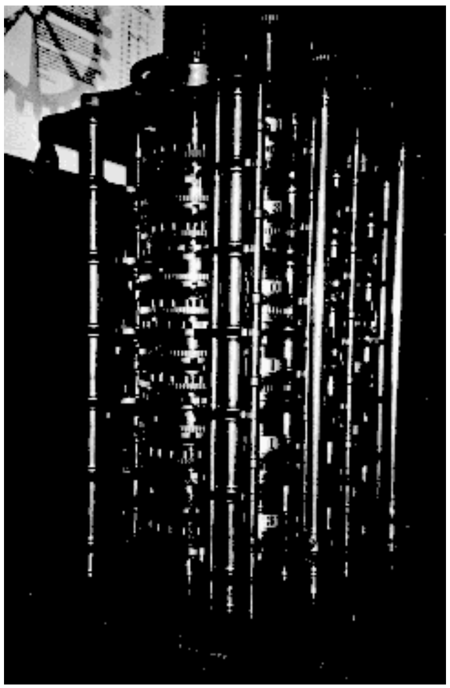
What do we See in a Computer?

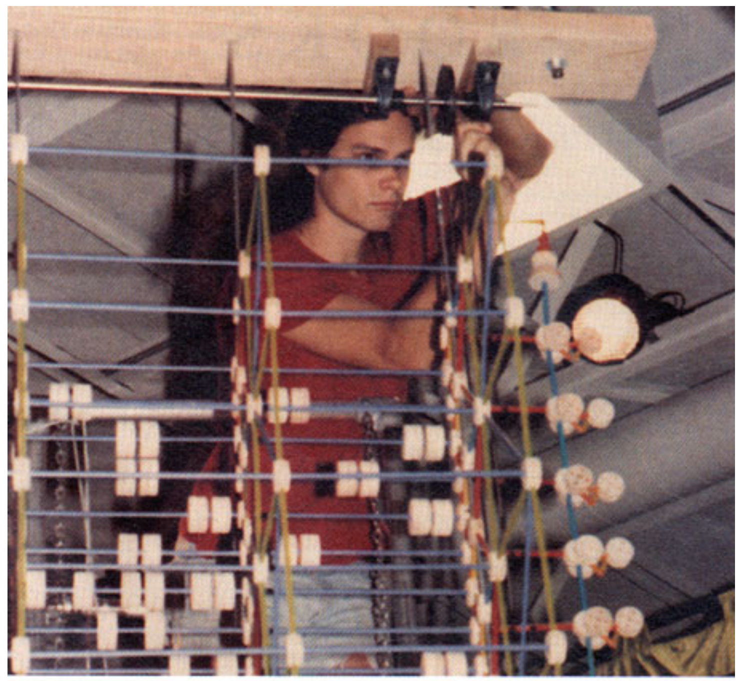
- Structure
  - hierarchical design:
    - limited complexity at each level
    - reusable building blocks
- Interfaces
  - Key elements of system engineering; typically outlive the technologies they interface
  - Isolate technologies, allow evolution
  - Major abstraction mechanism
- What makes a good system design?
  - "Bang for the buck": minimal mechanism, maximal function
  - reliable in a wide range of environments
  - accommodates future technical improvements

Computational Structures

What are the fundamental elements of computation?

Can we define computation independent of implementation or the substrate that it is built upon)?

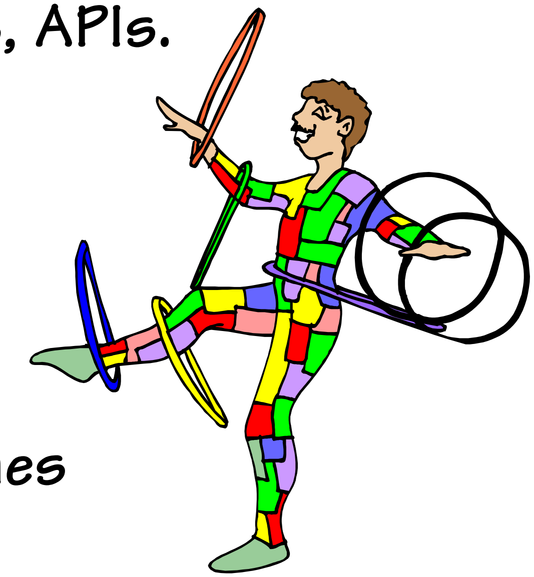




Edward Hardebeck helps to assemble the Tinkertoy computer

Our Plan of Attack…

- Understand how things work, by alternating between low-level (*bottom-up*) and high level (*top-down*) concepts
- Encapsulate our understanding using appropriate abstractions
- Study organizational principles: hierarchy, interfaces, APIs.



- Roll up our sleeves and design at each level of hierarchy
- Learn engineering tricks
  - from a historical perspective
  - using systematic design approaches
  - diagnose, fix, and avoid bugs

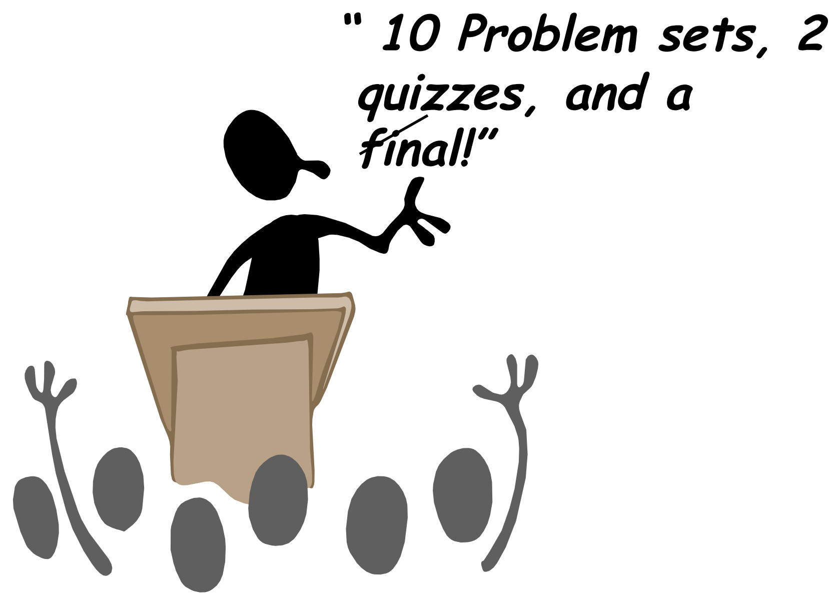
What is "Computation"?

Computation is about "processing information"

- Transforming information from one form to another
- Deriving new information from old
- Finding information associated with a given input
- "**Computation**" describes the motion of information through time
- "**Communication**" describes the motion of information through space

What is "Information"?

information, *n.* Knowledge communicated or received concerning a particular fact or circumstance.



A Computer Scientist's Definition:

Information resolves uncertainty.

Information is simply that which cannot be predicted. *The less predictable a message is, the more information it conveys!*

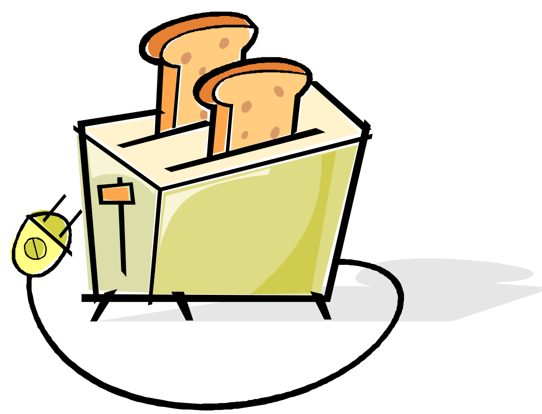
Real-World Information

Why do *unexpected* messages get allocated the biggest headlines?

... because they carry the most information.

What Does A Computer Process?

- Toasters processes bread and bagels
- Blenders processes smoothies and margaritas
- What does a computer process?
- 2 allowable answers:
  - Information
  - Bits
- How does information relate to bits?

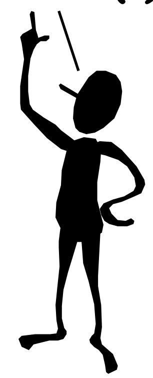


Quantifying Information
(Claude Shannon, 1948)

Suppose you're faced with N equally probable choices, and I give you a fact that narrows it down to M choices. Then you've been given:

$\log_2(N/M)$ bits of information

Information is measured in bits (binary digits) = number of 0/1's required to encode choice(s)

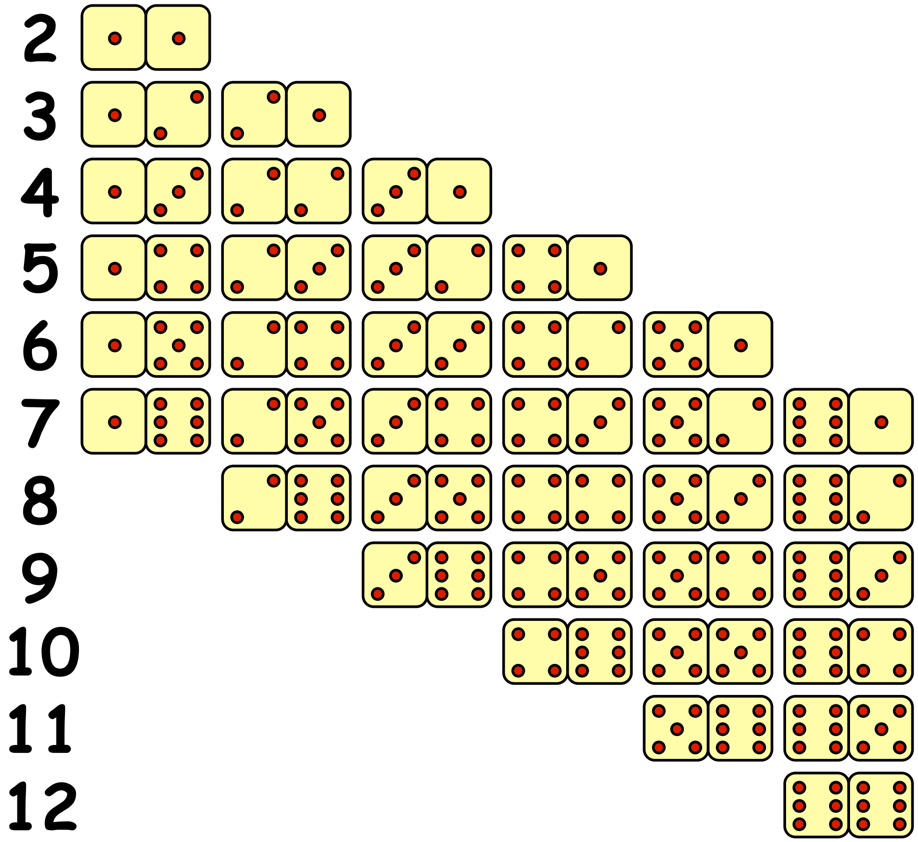


Examples:

- information in one coin flip: $\log_2(2/1) = 1$ bit
- roll of a single die: $\log_2(6/1) = \sim 2.6$ bits
- outcome of a Football game: 1 bit

(well, actually, "they won" may convey more information if they were "expected" to lose…)

Example: Sum of 2 dice



$i_2 = \log_2(36/1) = 5.170$ bits

$i_3 = \log_2(36/2) = 4.170$ bits

$i_4 = \log_2(36/3) = 3.585$ bits

$i_5 = \log_2(36/4) = 3.170$ bits

$i_6 = \log_2(36/5) = 2.848$ bits

$i_7 = \log_2(36/6) = 2.585$ bits

$i_8 = \log_2(36/5) = 2.848$ bits

$i_9 = \log_2(36/4) = 3.170$ bits

$i_{10} = \log_2(36/3) = 3.585$ bits

$i_{11} = \log_2(36/2) = 4.170$ bits

$i_{12} = \log_2(36/1) = 5.170$ bits

The **average** information provided by the sum of 2 dice:

$$i_{ave} = \sum_{i=2}^{12} \frac{M_i}{N} \log_2\left(\frac{N}{M_i}\right) = -\sum_i p_i \log_2(p_i) = 3.274 \text{ bits}$$

Show Me the Bits!

Can the sum of two dice REALLY be *represented* using 3.274 bits? If so, how?

The fact is, the average information content is a strict **lower-bound** on how small of a representation that we can achieve.

In practice, it is difficult to reach this bound. But, we can come very close.

Variable-Length Encoding

- Of course we can use differing numbers of “bits” to represent each item of data
- This is particularly useful if all items are *not* equally likely
- Equally likely items lead to fixed length encodings:
  - Ex: Encode a “particular” roll of 5?
  - {(1,4), (2,3), (3,2), (4,1)} which are equally likely if we use fair dice
  - Entropy = $-\sum_{i=1}^{4} p(roll_i | roll = 5)\log_2(p(roll_i | roll = 5)) = -\sum_{i=1}^{4} \frac{1}{4}\log_2(\frac{1}{4}) = 2$ bits
  - 00 – (1,4), 01 – (2,3), 10 – (3,2), 11 – (4,1)
- Back to the original problem. Let’s use this encoding:

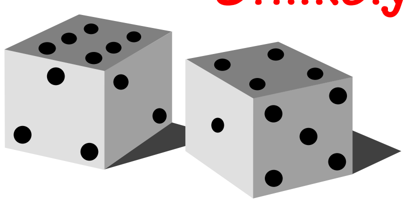
2 - 10011	3 - 0101	4 - 011	5 - 001
6 - 111	7 - 101	8 - 110	9 - 000
10 - 1000	11 - 0100	12 - 10010	

Variable-Length Encoding

- Taking a closer look

2 - 10011   3 - 0101   4 - 011   5 - 001
6 - 111   7 - 101   8 - 110   9 - 000
10 - 1000   11 - 0100   12 - 10010

Unlikely rolls are encoded using more bits



More likely rolls use fewer bits

- Decoding

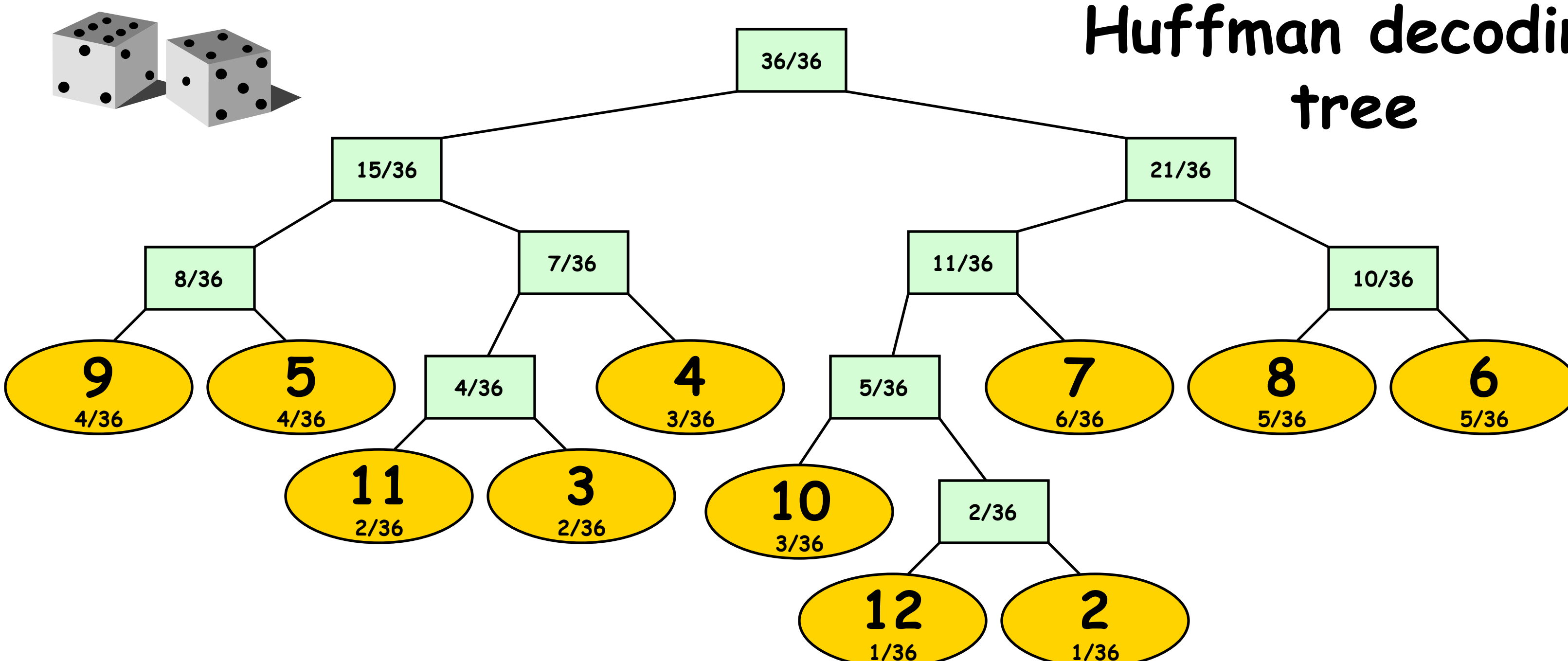
Example Stream:

2	5	3	6	5	8	3
10011	001	0101	111	001	110	0101

Huffman Coding

- A simple *greedy* algorithm for approximating an entropy efficient encoding
  1. Find the 2 items with the smallest probabilities
  2. Join them into a new *meta* item whose probability is their sum
  3. Remove the two items and insert the new meta item
  4. Repeat from step 1 until there is only one item

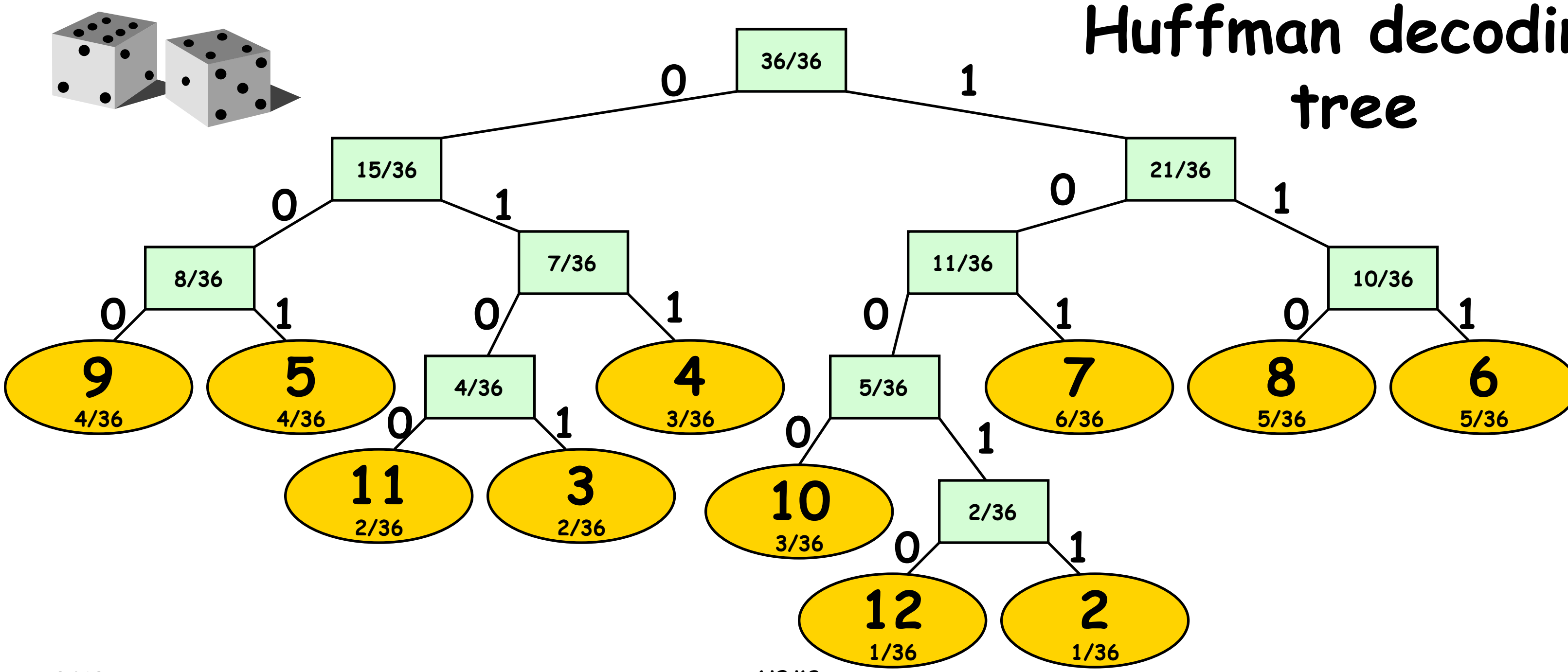
Huffman decoding tree



Converting Tree to Encoding

Once the *tree* is constructed, label its edges consistently and follow the paths from the largest *meta* item to each of the real item to find the encoding.

2 - 10011	3 - 0101	4 - 011	5 - 001
6 – 111	7 – 101	8 - 110	9 - 000
10 - 1000	11 - 0100	12 - 10010	



Huffman decoding tree

Encoding Efficiency

How does this encoding strategy compare to the information content of the roll?

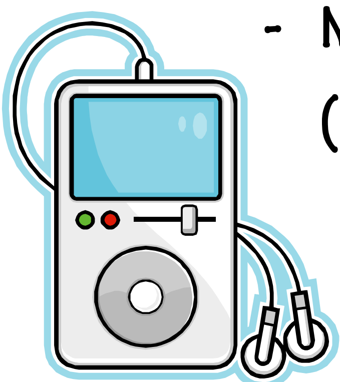
$$b_{ave} = \frac{1}{36}5 + \frac{2}{36}4 + \frac{3}{36}3 + \frac{4}{36}3 + \frac{5}{36}3 + \frac{6}{36}3$$
$$+ \frac{5}{36}3 + \frac{4}{36}3 + \frac{3}{36}4 + \frac{2}{36}4 + \frac{1}{36}5$$

$$b_{ave} = 3.306$$

Pretty close. Recall that the lower bound was 3.274 bits. However, an efficient encoding (as defined by having an average code size close to the information content) is not always what we want!

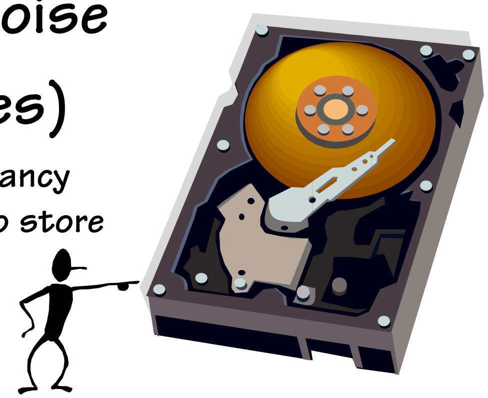
Encoding Considerations

- Encoding schemes that attempt to match the information content of a data stream remove redundancy. They are *data compression* techniques.
- However, sometimes our goal in encoding information is *increase redundancy*, rather than remove it. Why?
  - Make the information easier to manipulate (fixed-sized encodings)
  - Make the data stream resilient to noise (error detecting and correcting codes)



-Data compression allows us to store our entire music and video collections in a pocketable device

-Data redundancy enables us to store that *same* information *reliably* on a hard drive



Error detection using parity

Sometimes we add extra redundancy so that we can detect errors. For instance, this encoding detects any single-bit error:

2-1111000
3-1111101
4-0011
5-0101
6-0110
7-0000
8-1001
9-1010
10-1100
11-1111110
12-1111011

ex: 0011 11100000101 (4)

1: 0011 1110 0000 101 (4, 6*, 7)

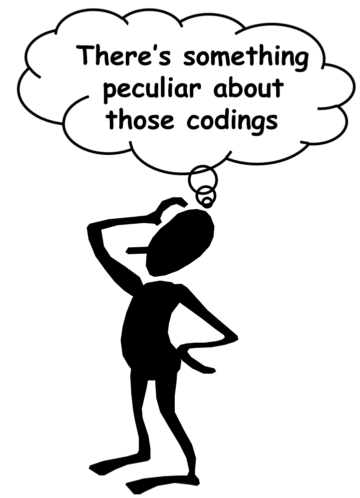
2: 0011 1110 0000 101 (4, 9*, 7)

3: 0011 1110 0000 101 (4, 10*, 7)

4: 0011 1110000 0101 (4, 2*, 5)

Same bitstream – w/4 possible interpretations if we allow for only one error

-
0
1
0
1
1



Property 1: Parity

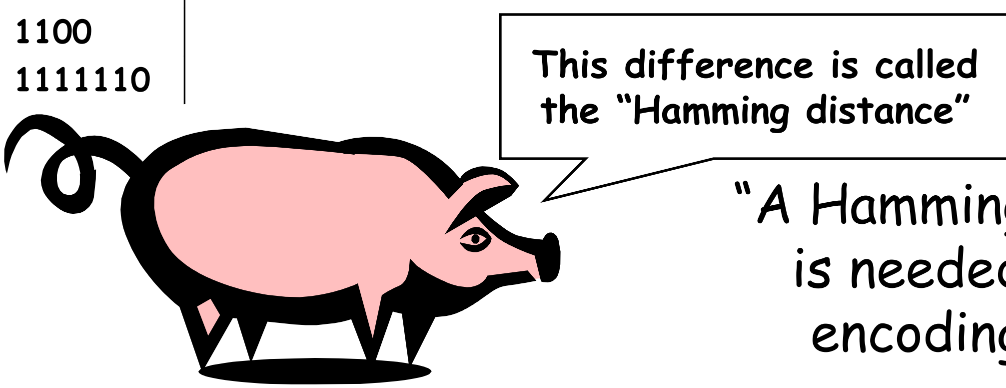
The sum of the bits in each symbol is even.
(this is how errors are detected)

2-1111000 = 1 + 1 + 1+ 1 + 0 + 0 + 0 = 4
3-1111101 = 1 + 1 + 1 + 1 + 1 + 0 + 1 = 6
4-0011 = 0 + 0 + 1 + 1 = 2
5-0101 = 0 + 1 + 0 + 1 = 2
6-0110 = 0 + 1 + 1 + 0 = 2
7-0000 = 0 + 0 + 0 + 0 = 0
8-1001 = 1 + 0 + 0 + 1 = 2
9-1010 = 1 + 0 + 1 + 0 = 2
10-1100 = 1 + 1 + 0 + 0 = 2
11-1111110 = 1 + 1 + 1 + 1 + 1 + 1 + 0 = 6
12-1111011 = 1 + 1 + 1 + 1 + 0 + 1 + 1 = 6

Property 2: Separation

Each encoding differs from all others by at least **two bits** in their overlapping parts

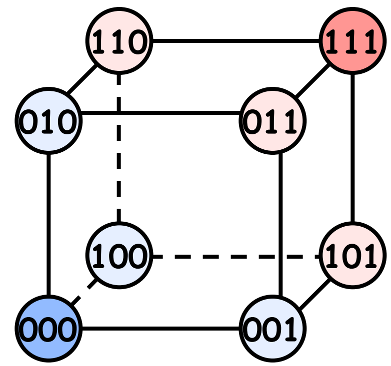
		3 1111101	4 0011	5 0101	6 0110	7 0000	8 1001	9 1010	10 1100	11 1111110	12 1111011
2	1111000	1111x0x	xx11	x1x1	x11x	xxxx	1xx1	1x1x	11xx	1111xx0	11110xx
3	1111101		xx11	x1x1	x11x	xxxx	1xx1	1x1x	11xx	11111xx	1111xx1
4	0011			0xx1	0x1x	00xx	x0x1	x01x	xxxx	xx11	xx11
5	0101				01xx	0x0x	xx01	xxxx	x10x	x1x1	x1x1
6	0110					0xx0	xxxx	xx10	x1x0	x11x	x11x
7	0000						x00x	x0x0	xx00	xxxx	xxxx
8	1001							10xx	1x0x	1xx1	1xx1
9	1010								1xx0	1x1x	1x1x
10	1100									11xx	11xx
11	1111110										1111x1x



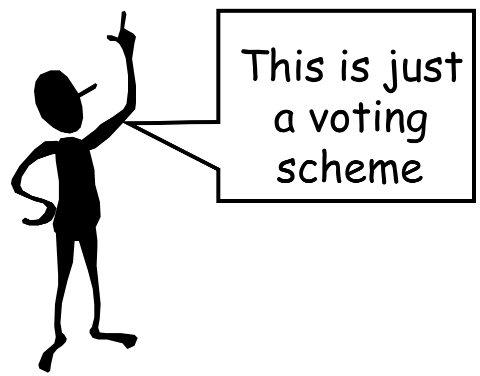
"A Hamming distance of one-bit is needed to provide unique encodings for every item"

Error correcting codes

We can actually **correct** 1-bit errors in encodings separated by a Hamming distance of three. This is possible because the sets of bit patterns located a Hamming distance of 1 from our encodings are distinct.



However, **attempting error correction with such a small separation is dangerous**. Suppose, we have a 2-bit error. Our error correction scheme will then misinterpret the encoding. Misinterpretations also occurred when we had 2-bit errors in our 1-bit-error-detection (parity) schemes.

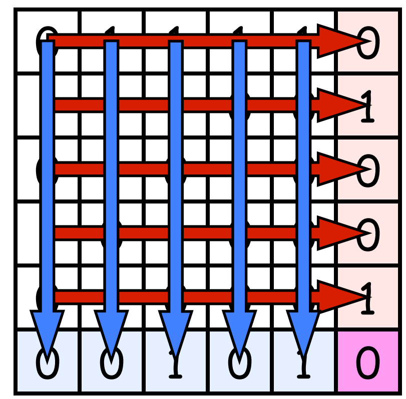


A safe 1-bit error correction scheme would correct all 1-bit errors and detect all 2-bit errors. What Hamming distance is needed between encodings to accomplish this?

An alternate error correcting code

We can generalize the notion of parity in order to construct error correcting codes. Instead of computing a single parity bit for an entire encoding we can allow multiple parity bits over different subsets of bits. Consider the following technique for encoding 25 bits.

0	1	1	1	1	0
1	1	1	0	0	1
0	1	0	1	0	0
1	0	1	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0



Include a parity bit for each row and column. This approach is easy to implement, but it is not optimal in terms of the number of bits used. Note that we also include one more parity bit for the row and column parity bits (shown in purple). Thus, to detect and correct a single bit error among 25 bits we have added 5 + 5 + 1 = 11 bits of redundancy.

An alternate error correcting code

We can generalize the notion of parity in order to construct error correcting codes. Instead of computing a single parity bit for an entire encoding we can allow multiple parity bits over different subsets of bits. Consider the following technique for encoding 25 bits.

0	1	1	1	1	0
1	1	1	0	0	1
0	1	0	0	0	0
1	0	1	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0

0	1	1	1	1	0
1	1	1	0	0	1
0	1	0	1	0	0
1	0	1	0	0	0
0	1	0	0	0	1
0	0	0	0	1	0

1-bit errors will cause both a row and column parity error, whose intersection uniquely identifying the errant bit for correction. The errant bit is shown in red above, and the associated, and now incorrect, parity bits are shown in gray. This even works for errors in the parity bits! as shown in the example on the right, where the intersection of errant parity bits is itself a parity bit.

An alternate error correcting code

We can generalize the notion of parity in order to construct error correcting codes. Instead of computing a single parity bit for an entire encoding we can allow multiple parity bits over different subsets of bits. Consider the following technique for encoding 25 bits.

0	1	1	1	1	0
1	0	1	0	0	1
0	1	0	1	0	0
1	0	1	1	0	0
0	1	0	0	0	1
0	0	1	0	1	0

0	1	1	1	1	0
1	1	1	1	0	1
0	1	0	1	0	0
1	1	1	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0

However, 2-bit errors are generally ambiguous. It they happen in the same row/column, then you can't even figure out which row/column the errors occurred in. As a result, we consider such a block parity systems to be "1-bit error correcting and 2-bit error detecting".

Summary

Information resolves uncertainty

- Choices equally probable:
  - N choices narrowed down to M $\rightarrow$
    $\log_2(N/M)$ bits of information
- Choices not equally probable:
  - $choice_i$ with probability $p_i \rightarrow$
    $\log_2(1/p_i)$ bits of information
  - average number of bits = $\Sigma p_i \log_2(1/p_i)$
  - variable-length encodings

Next time:

- How to encode thing we care about using bits, such as numbers, characters, etc…
- Bit's cousins, bytes, nibbles, and words