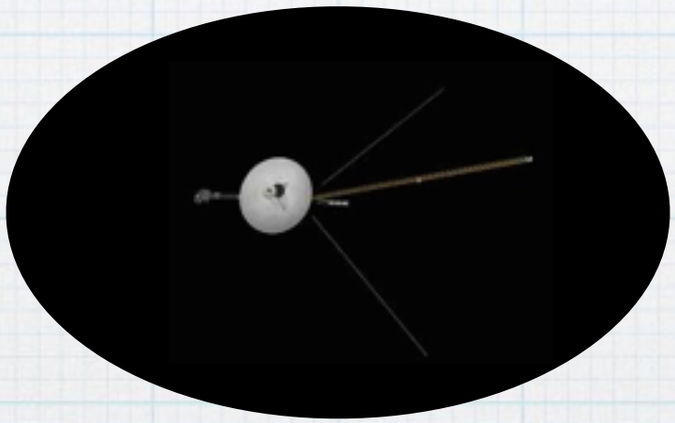
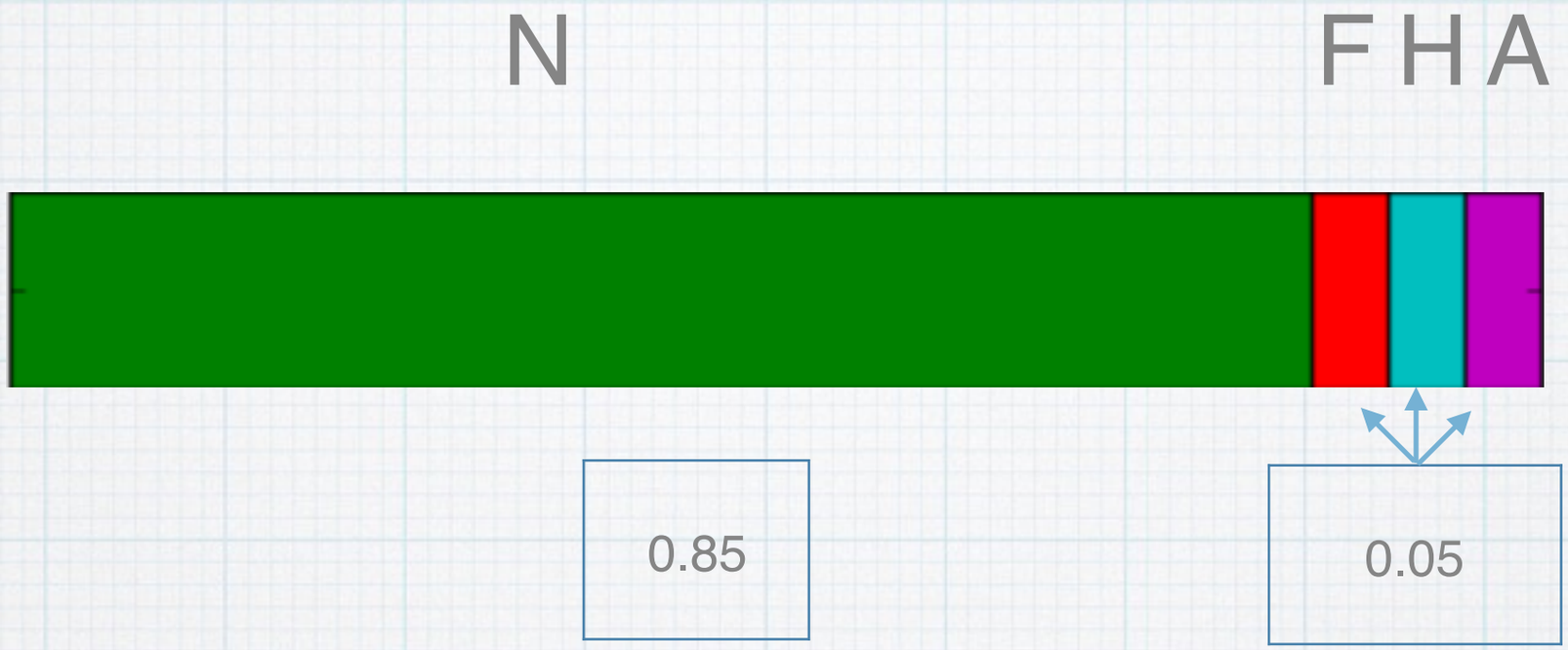
Precision Can be a Stream



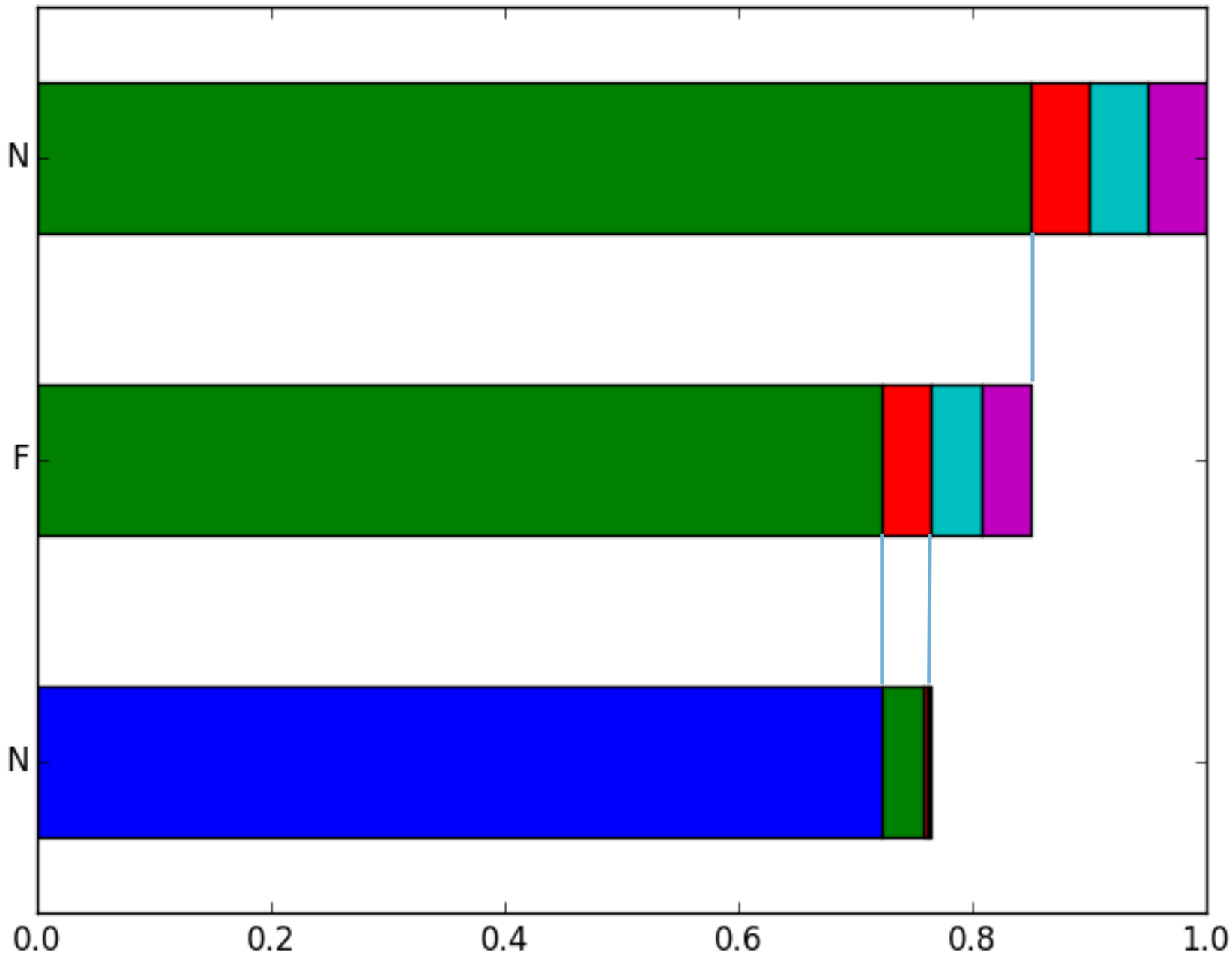
Message	Probability	Naive Code	Better Code	P(x) * Bits	$- \Sigma P(x_i)Log_2P(x_i)$
No Aliens	0.85	00	0	0.85	0.20
Friendly Aliens	0.05	01	100	0.15	0.22
Hostile Aliens	0.05	10	101	0.15	0.22
Aloof Aliens	0.05	11	110	0.15	0.22
Bits / Message		2		1.3	0.85

$$- \Sigma P(x_i)Log_2P(x_i)$$

* Arithmetic Encoding sends a series of messages as an arbitrary precision fraction.
* Decode using an agreed frequency distribution.
* Each symbol refines the range the number can be in



Suppose our message starts NFN….


N
F
N
0.0
0.2
0.4
0.6
0.8
1.0

```
probAliens = {
    'N': (0, 0.85),
    'F': (0.85, 0.05),
    'H': (0.9, 0.05),
    'A': (0.95, 0.05)
}

def encode(string, prob):
    start = 0
    width = 1
    for ch in string:
        d_start, d_width = prob[ch]
        start += d_start*width
        width *= d_width
    return random.uniform(start, start+width)

def decode(num, prob, message_length):
    string = []
    for x in range(0,message_length):
        for symbol, (start, width) in prob.iteritems():
            if 0 <= num - start < width:
                num = (num - start) / width
                string.append(symbol)
                break
    return ''.join(string)

original='NFN'
encoded_number = encode(original, probAliens)
print "encoded_number", encoded_number

decoded_string = decode(encoded_number, probAliens, len(original))
print "decoded_string", decoded_string
```

See Program

* Like Huffman encoding compression depends on the probability being accurate
* Will approach theoretical compression rate from above
* Just pick any number in calculated range

* Typically some end symbol agreed - takes part of the probability range.
* Updating probability - agreed on code or period.
* Integer implementation (Witten) which avoids the need for infinite precision floating point values