Rateless Codes

COS 463: Wireless Networks
Lecture 10

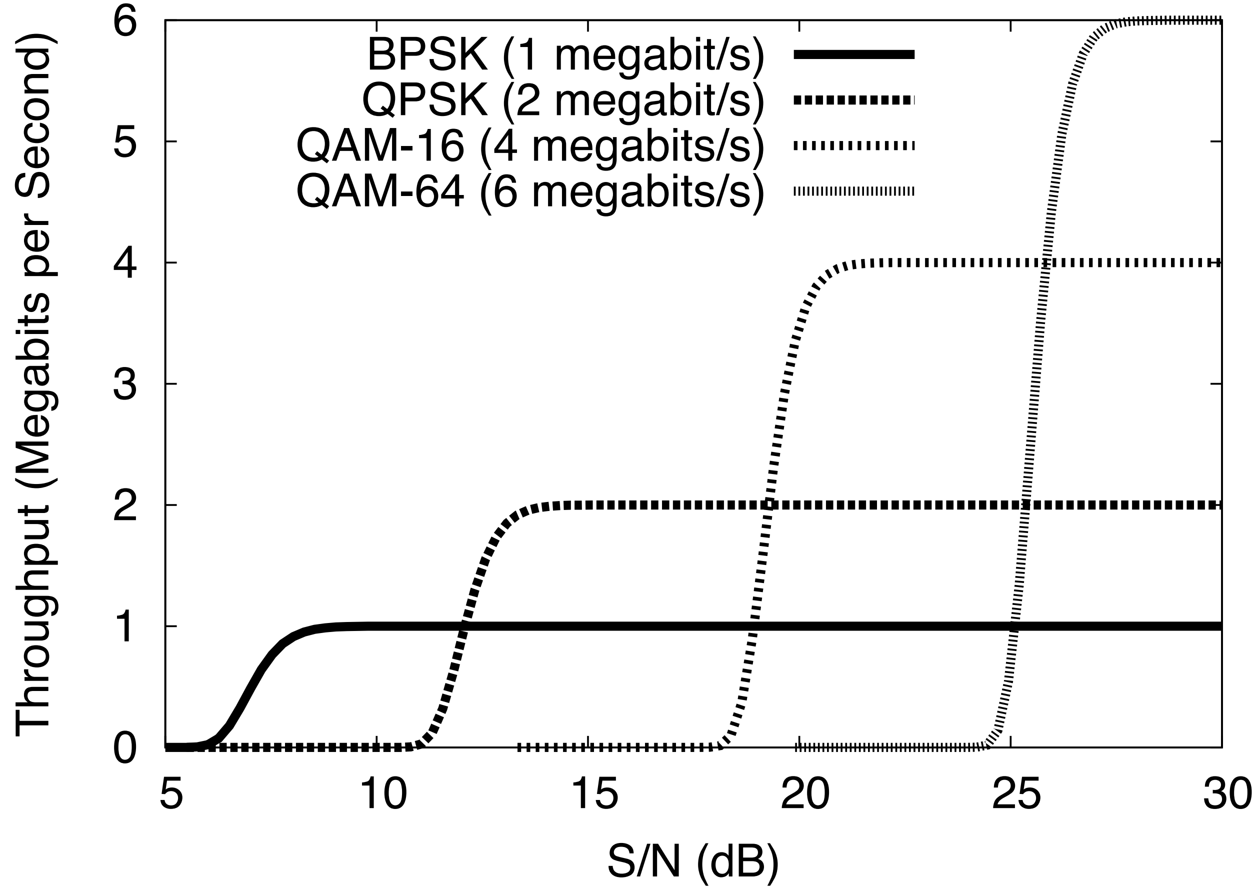
Kyle Jamieson

Today

1. **Rateless *fountain* codes**
   – Luby Transform (LT) Encoding
   – LT Decoding

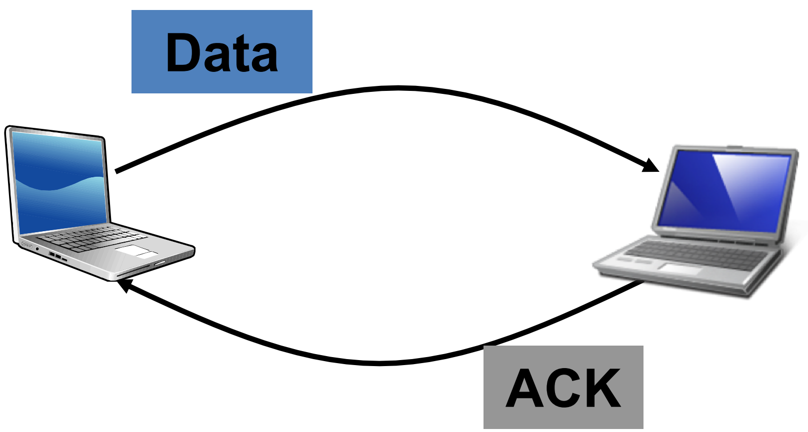
2. Rateless Spinal codes

Fixed-rate codes *require* channel adaptation



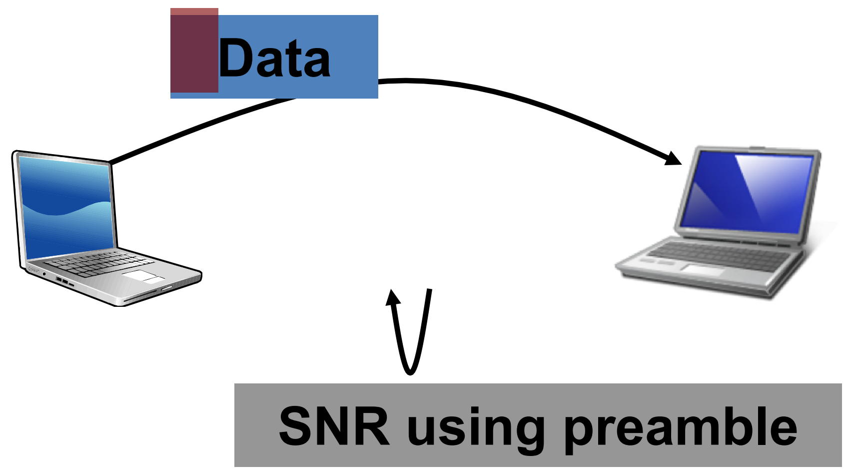
Existing rate adaptation algorithms

Frame-based



Estimate frame loss rate at each bit rate

SNR/BER-based



Lookup table:
SNR/BER → best rate

Rateless codes: Motivation (1)

- Sender transmits information at a rate **higher** than the channel can sustain
  - At first glance, this sounds disastrous!
- Receiver extracts information at the rate the channel can sustain **at that instant**
  - **No adaptation loop is needed!**

Rateless codes: Motivation (2)

- **Sender** sends a potentially limitless stream of encoded bits

- **Receiver(s)** collect bits until they are reasonably sure that they can recover the content from the received bits, then send **STOP** feedback to sender

- **Automatic adaptation**: Receivers with larger loss rate need longer to receive the required information

LT encoding

- Consider a **message** ***m*** with ***K*** bits

- LT encoding produces ***N coded bits,*** **for any *N***
  - **Variable code rate: *K / N***

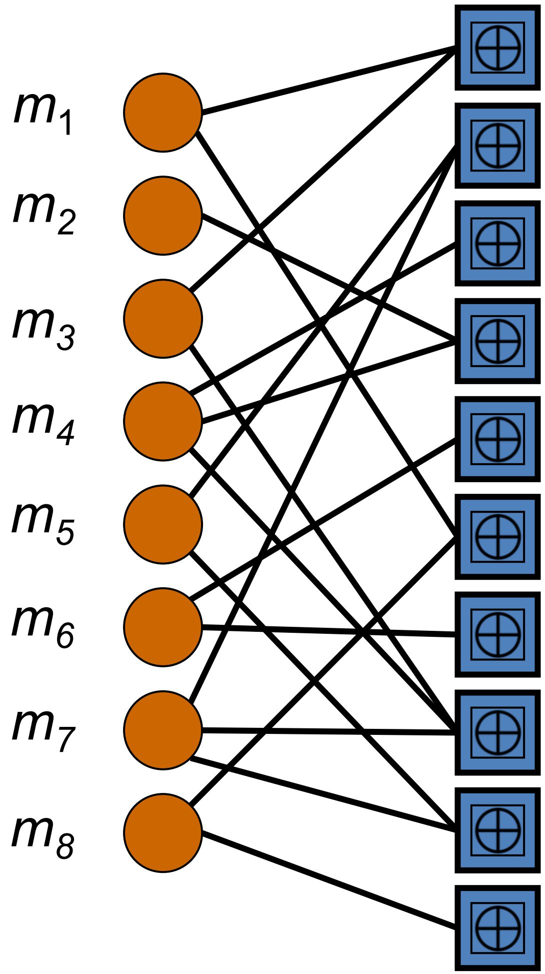
- To produce the n^{th} coded bit:
  - Choose at random the coded bit's ***degree*** d_n from a ***degree distribution***

  - Choose **d_n distinct message bits,** uniformly at random
    - Xor them together to form one ***coded bit*** $\boldsymbol{c_n}$

LT encoding

Message bits:

Coded bits:



$c_1 = m_1 \oplus m_3$

$c_2 = m_5 \oplus m_7$

$c_3 = m_4$

$c_4 = m_2 \oplus m_4$

$c_5 = m_6$

$c_6 = m_1 \oplus m_8$

$c_7 = m_6$

$c_8 = m_3 \oplus m_4 \oplus m_7$

$c_9 = m_5 \oplus m_7$

$c_{10} = m_8$

LT decoding algorithm

1. Find a coded bit c_n with **degree one**
   – If not possible, **fail**

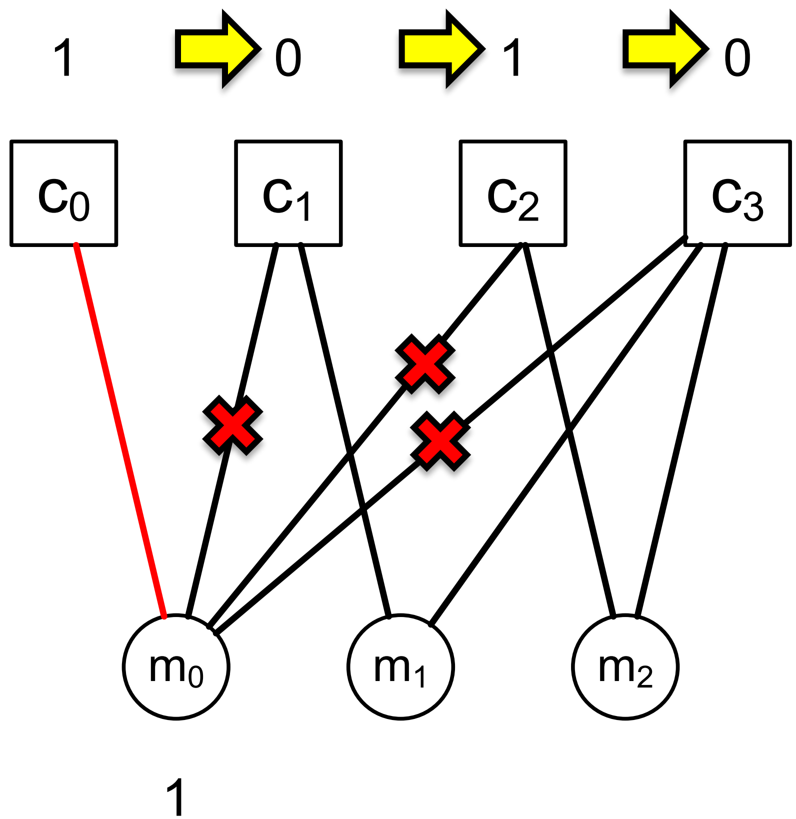
2. Decide its **incident** message bit *i:* $m_i = c_n$

3. Add m_i (with xor) to all coded bits c_n **incident on** m_i

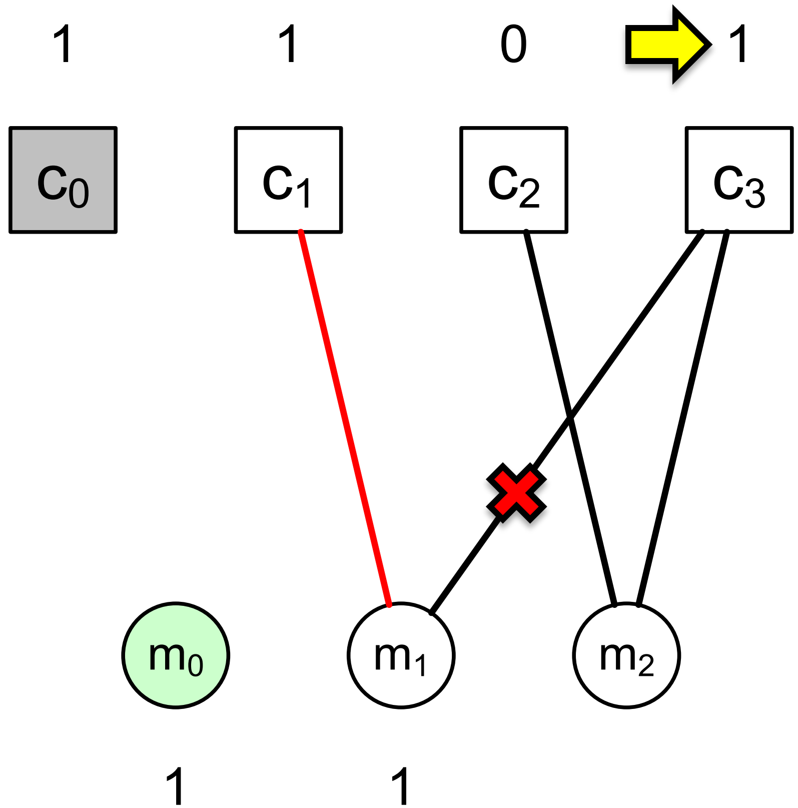
4. Remove all edges **incident on** m_i

5. Repeat Steps 1 to 4 until all m_i are found

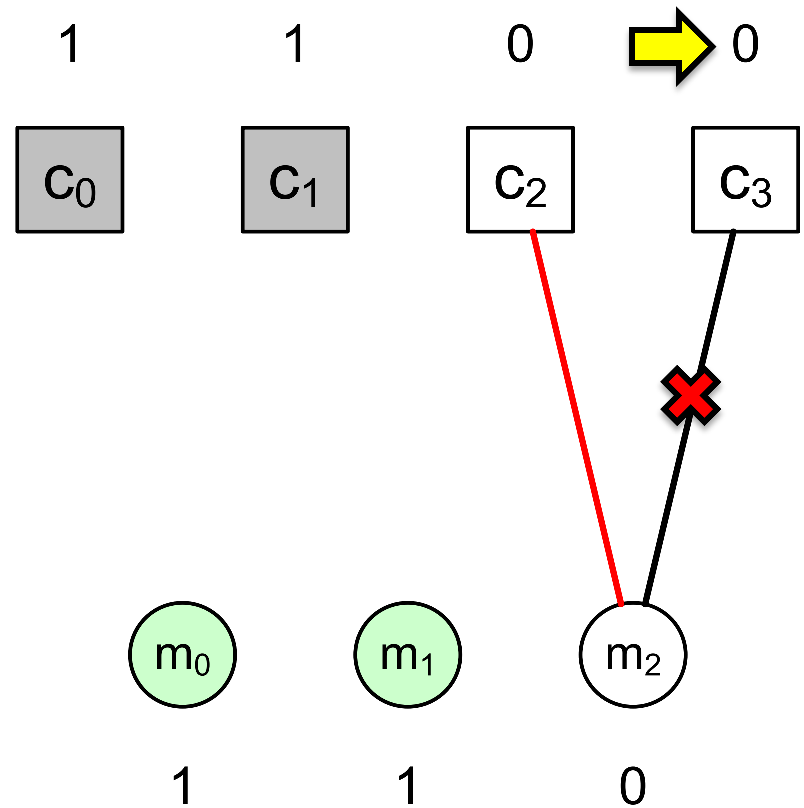
LT decoding example



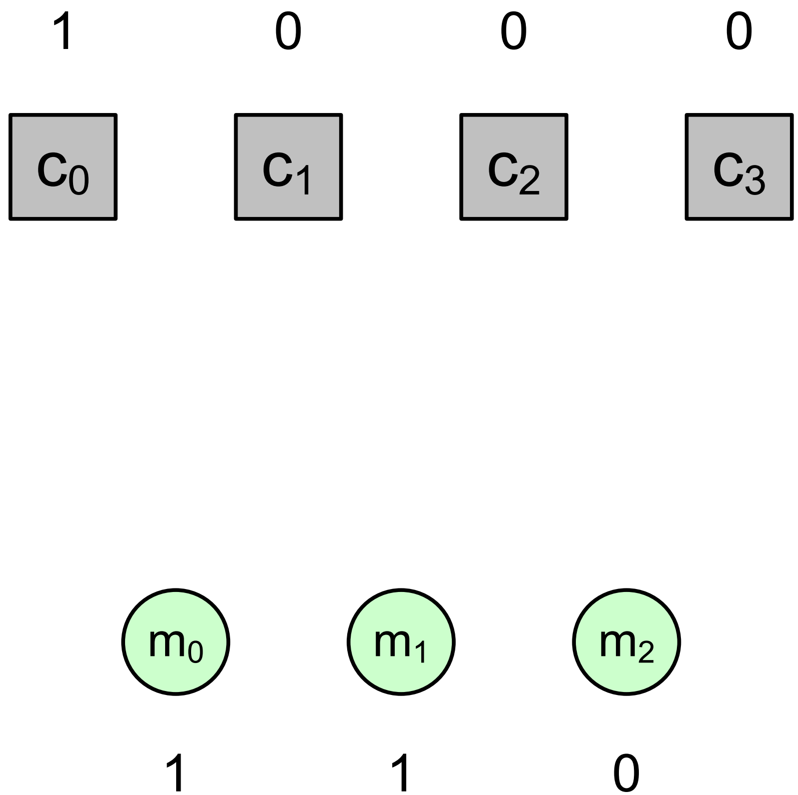
LT decoding example



LT decoding example



LT decoding example

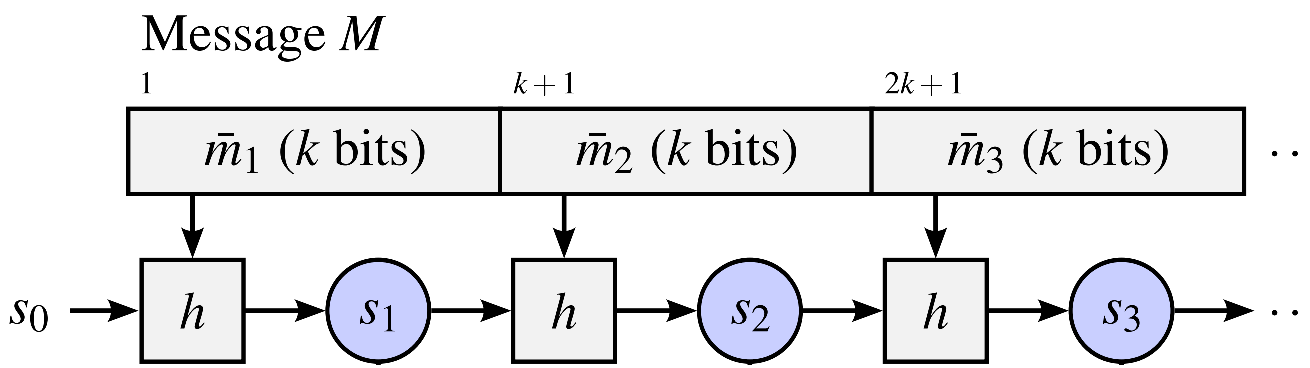


Today

1. Rateless *fountain* codes
   – Luby Transform (LT) Encoding
   – LT Decoding

2. Rateless Spinal codes
   – **Encoding Spinal Codes**
   – Decoding Spinal codes
   – Performance evaluation

Spinal encoder: Computing the spines

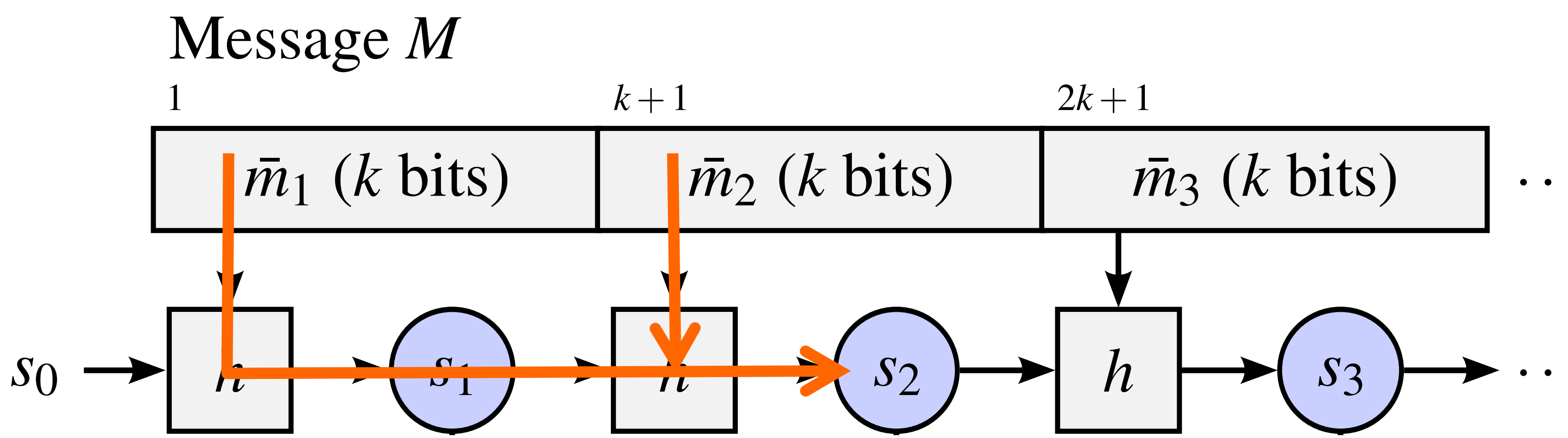


- Start with a hash function *h* and an initial random *v*-bit ***state*** s_0
  - Sender and receiver agree on *h* and s_0 *a priori*

- Sender divides its *n*-bit ***message*** *M* into *k*-bit ***chunks*** m_i

- *h* maps the state and a message chunk into a new state
  - The *v*-bit states $s_1, \ldots, s_{\lceil n/k \rceil}$ are the ***spines***

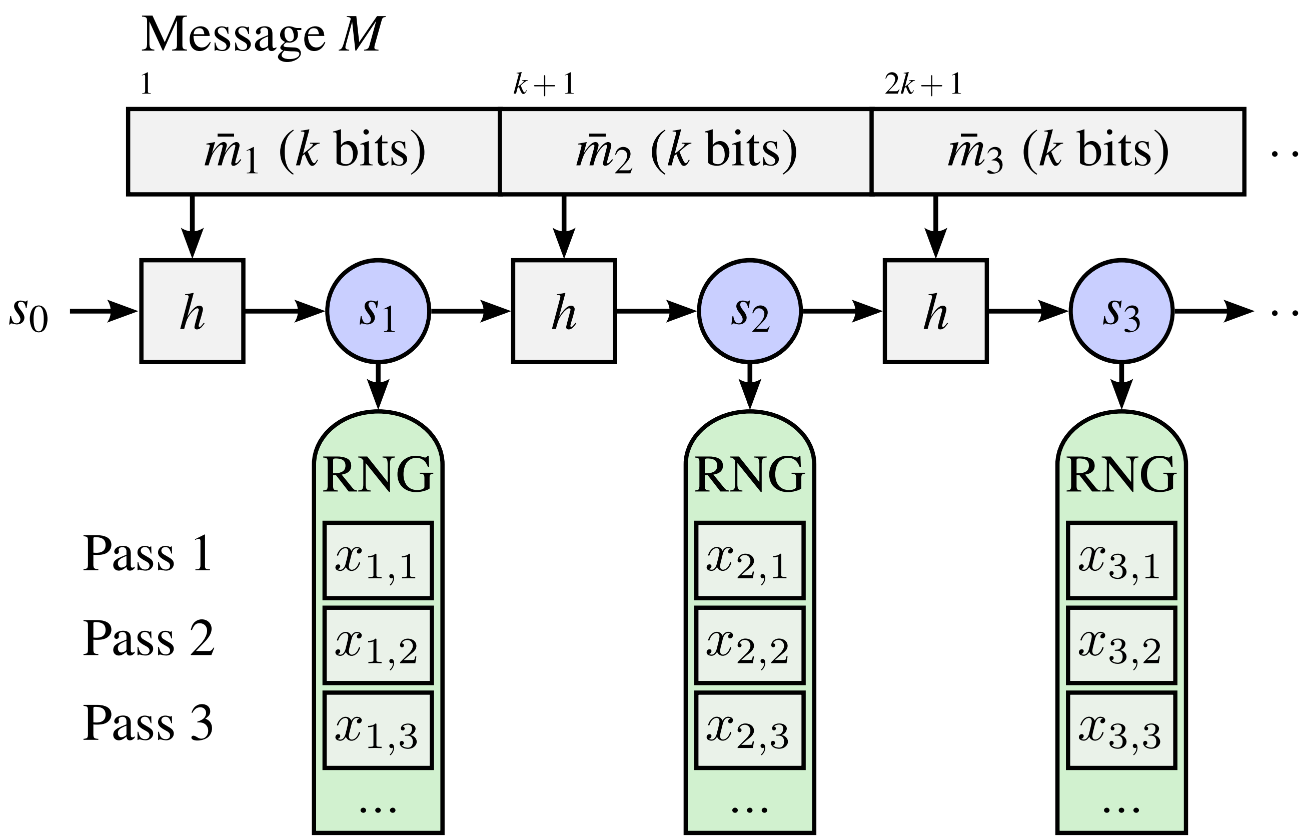
Spinal encoder: Information flow



- Observe: State s_i contains information about chunks $m_1, \ldots, m_i$
  - A stage's state depends on the message bits **up to** that stage

- So only state $s_{\lceil n/k \rceil}$ has information about entire message

Spinal encoder: Computing the spines

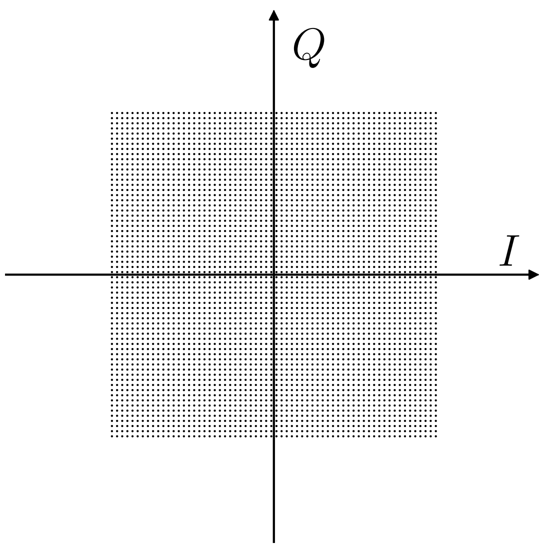


- Each spine seeds a random number generator ***RNG***
- RNG generates a sequence of *c*-bit numbers
- Encoder output is a series of ***passes*** of $\lceil n/k \rceil$ ***symbols*** $x_{i,l}$ each

Spinal encoder: RNG to symbols

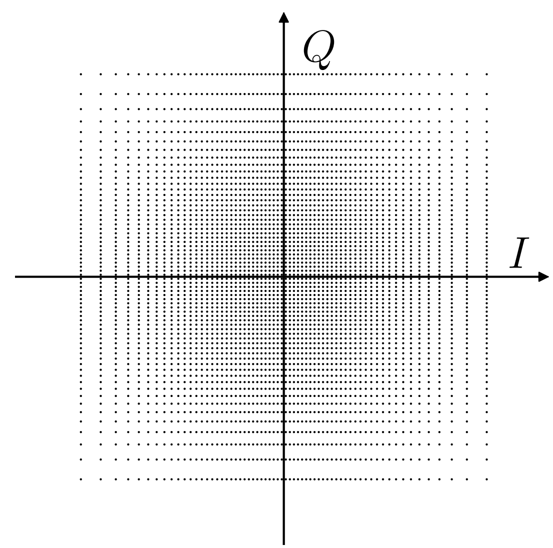
- A constellation mapping function translates *c*-bit numbers $x_{i,l}$ from the RNG to in-phase (I) and quadrature (Q)
  - Generates in-phase (I) and quadrature (Q) components **independently from two separate** $x_{i,l}$

Uniform



$$x_{i,l} \mapsto \frac{x_{i,l} + \frac{1}{2} - 2^{c-1}}{2^c \sqrt{6P}}$$

Truncated Gaussian



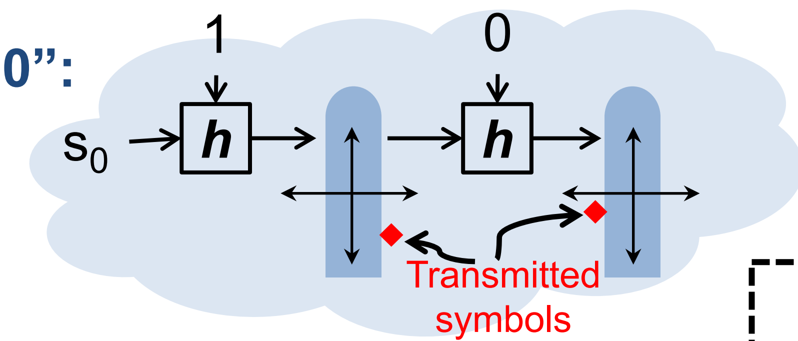
Today

1. **Rateless *fountain* codes**
   - Luby Transform (LT) Encoding
   - LT Decoding

2. Rateless Spinal codes
   - Encoding Spinal Codes
   - **Decoding Spinal codes**
     - **“Maximum-likelihood” decoding**
     - The Bubble Decoder
     - Puncturing for higher rate
   - Performance evaluation

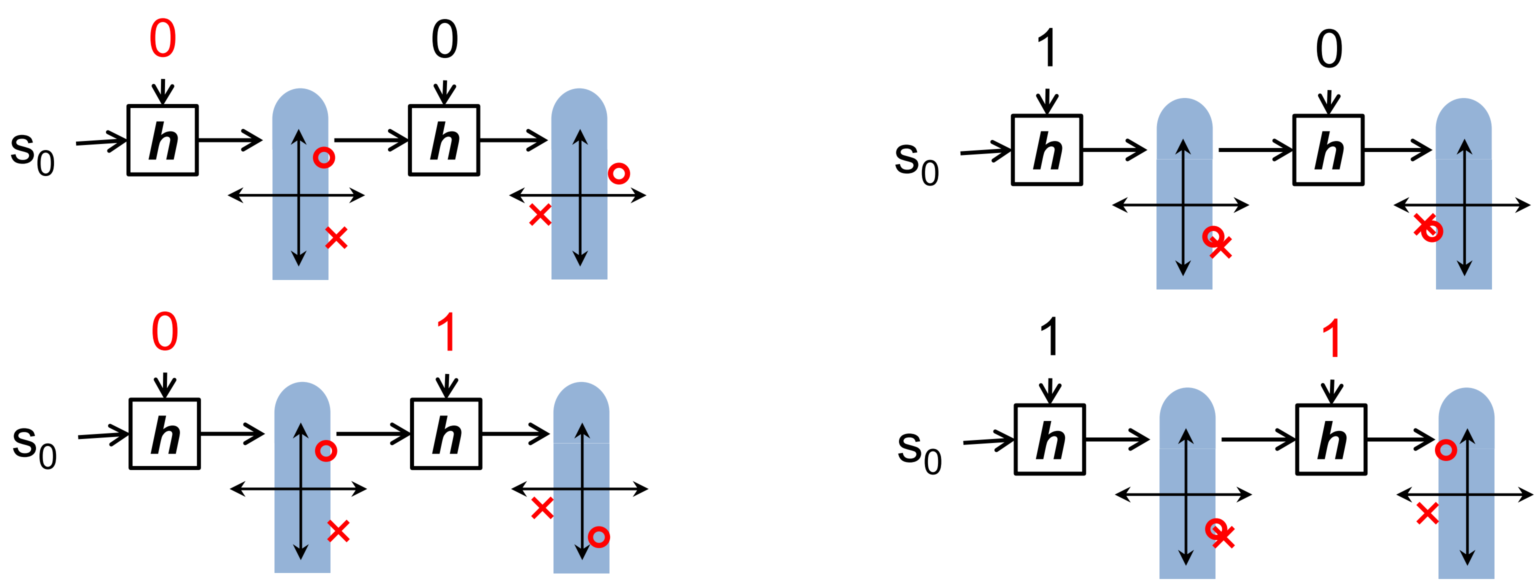
Decode by replaying the encoder

Sender transmits “1”, “0”:



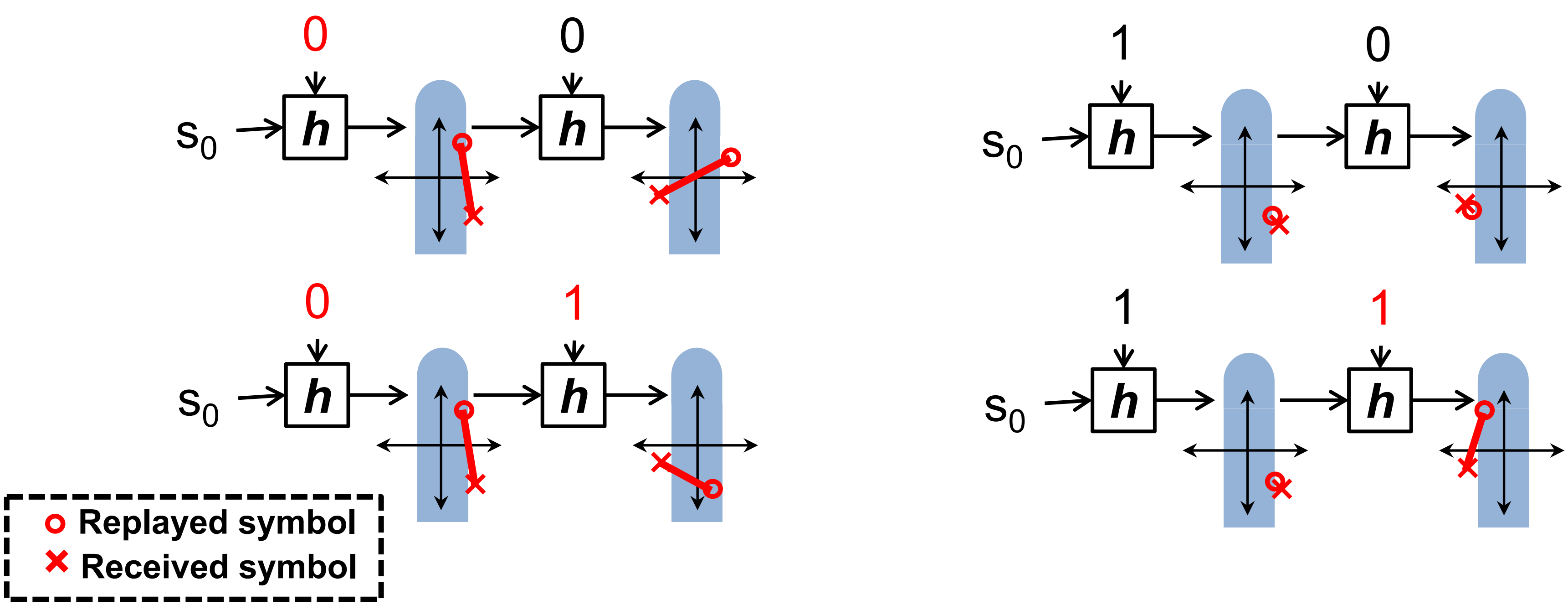
- o **Replayed symbol**
- × **Received symbol**

Instead of inverting the hash function, the decoder *replays* all four possibilities:

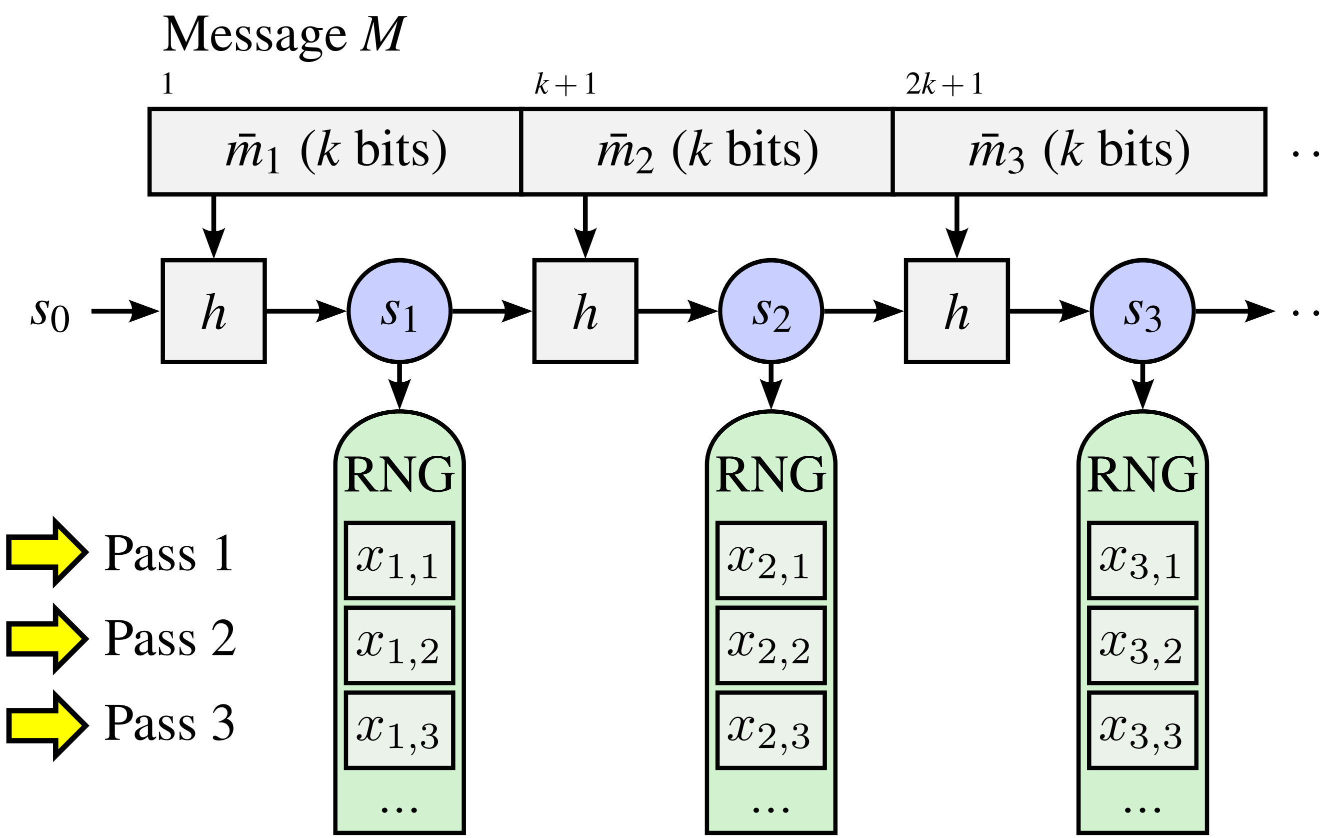


Decode by measuring distance

- ***How to decide between the four possible messages?***
- **Measure total distance** between:
  - Received symbols, corrupted by noise (×), and
  - Replayed symbols (○)
- Sum across stages: the distance increases at **first incorrect symbol**



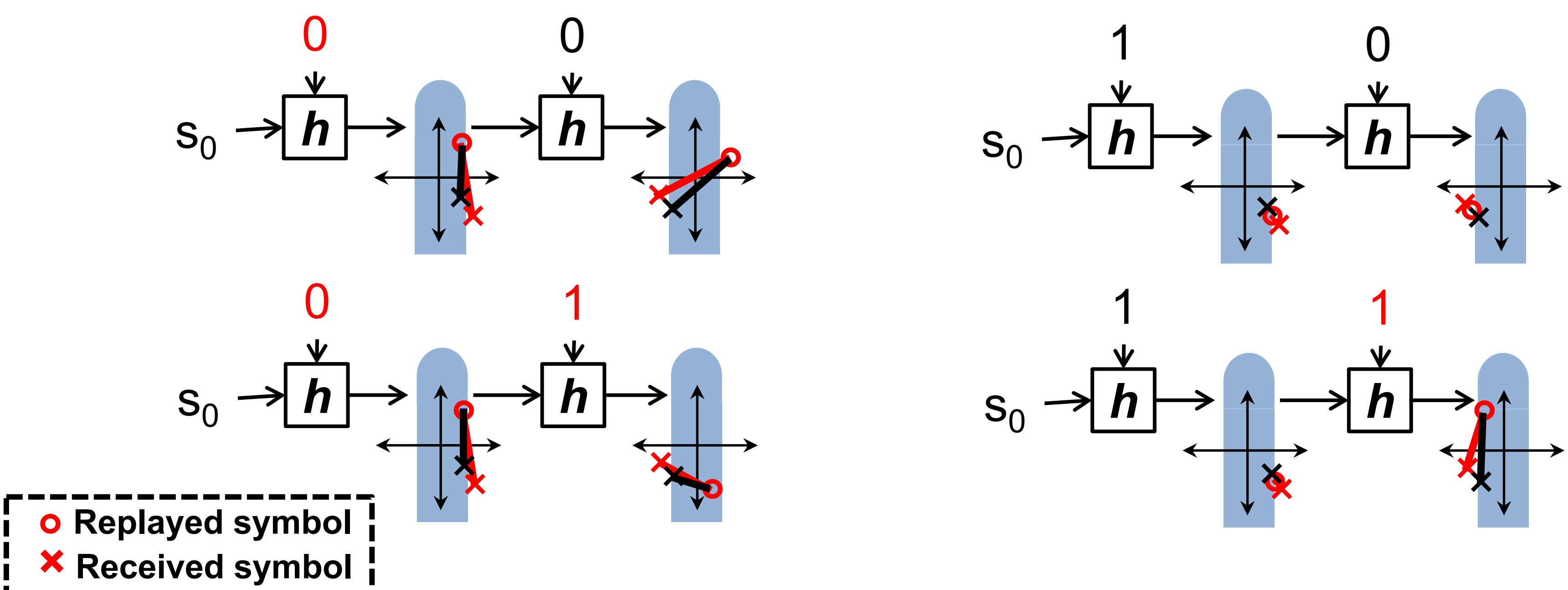
Adding additional passes



- **Recall:** The encoder sends **multiple passes** over the same message blocks

Adding additional passes

- What's a reasonable strategy for decoding now?
- Take the **average distance** from the replayed symbol (o), **across all received symbols** (×, ×)
  - Intuition: As number of passes increases, noise and bursts of interference average out and impact the metric less



The Maximum Likelihood (ML) decoder

- Consider all 2^n possible messages that could have been sent
  - The ML decoder **minimizes probability of error**
- Pick the message M' that minimizes the vector distance between:
  - The vector of all received constellation points $\mathbf{y}$
  - The vector of constellation points sent **if M' were the message,** $\mathbf{x}(M')$

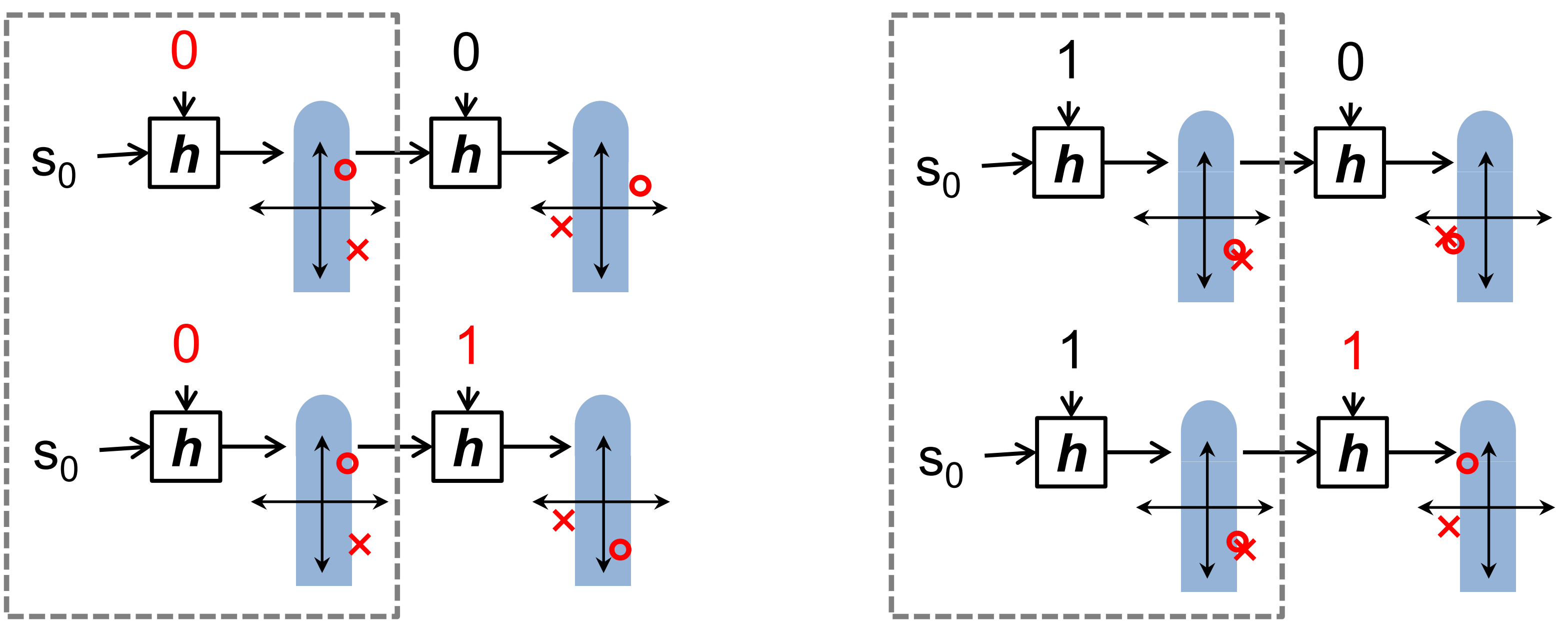
$$\hat{M} = \arg \min_{M' \in \{0,1\}^n} \left\| \mathbf{y} - \mathbf{x}\left(M'\right) \right\|^2$$

- In further detail:
  1. $x_{t,l}(M')$: t^{th} constellation point **sent** in the l^{th} pass for M'
  2. $y_{t,l}$: t^{th} constellation point **received** in the l^{th} pass

$$\hat{M} = \arg \min_{M' \in \{0,1\}^n} \sum_{\text{all } t,l} \left| y_{t,l} - x_{t,l}\left(M'\right) \right|^2$$

ML decoding over a tree

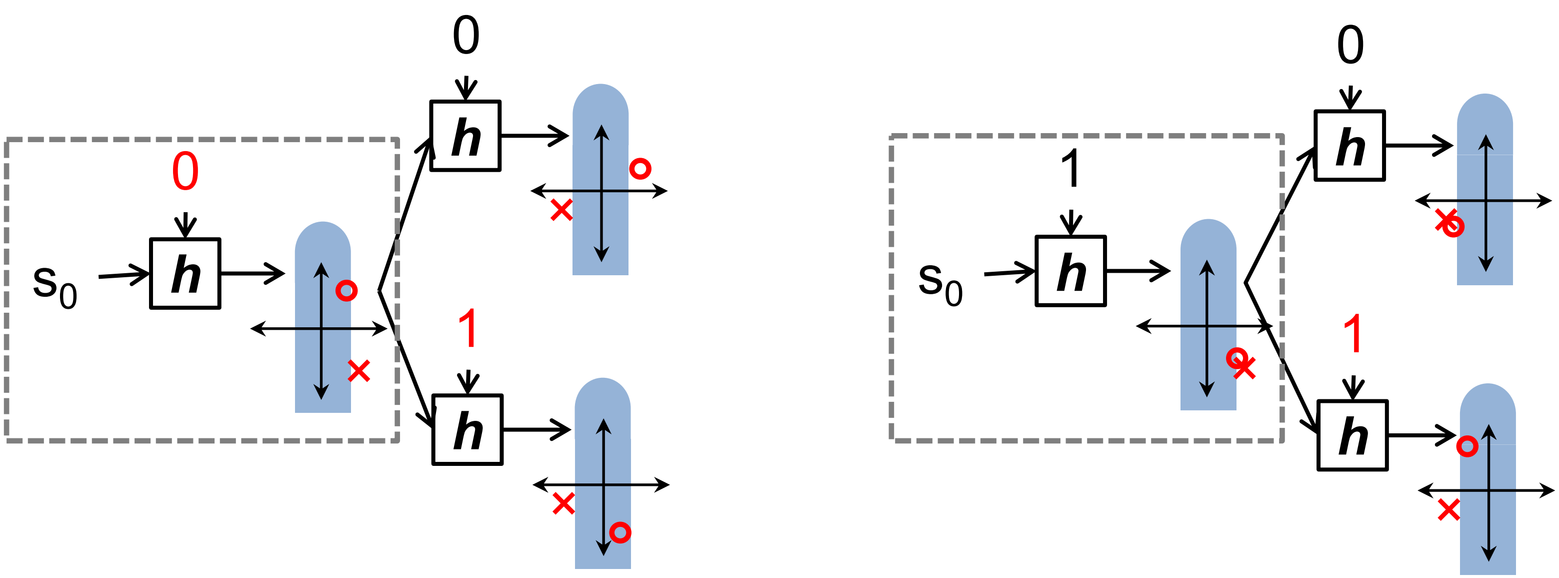
- Observe: Hypotheses whose initial stages share the same symbol guesses are **identical** in those stages



ML decoding over a tree

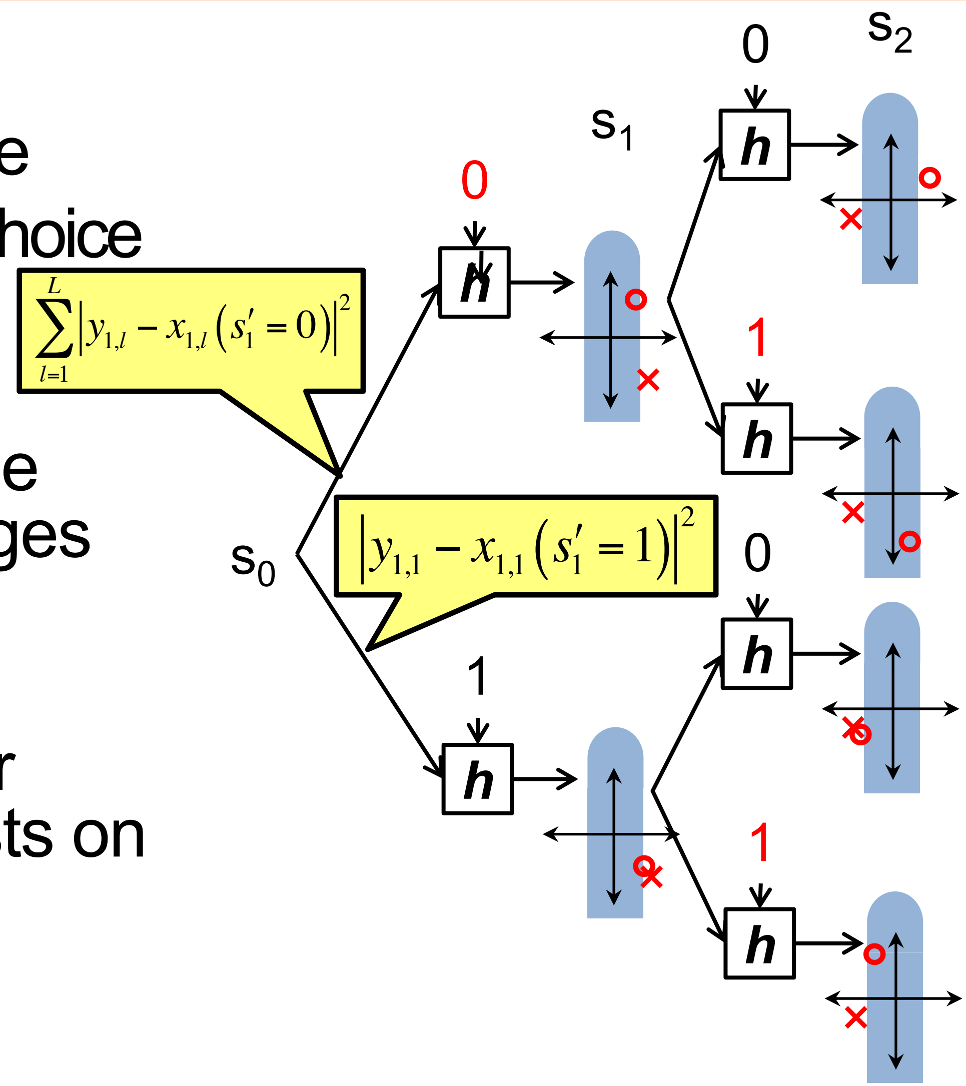
- Observe: Hypotheses whose initial stages share the same symbol guesses are **identical** in those stages

- Therefore we can **merge** these initial identical stages:



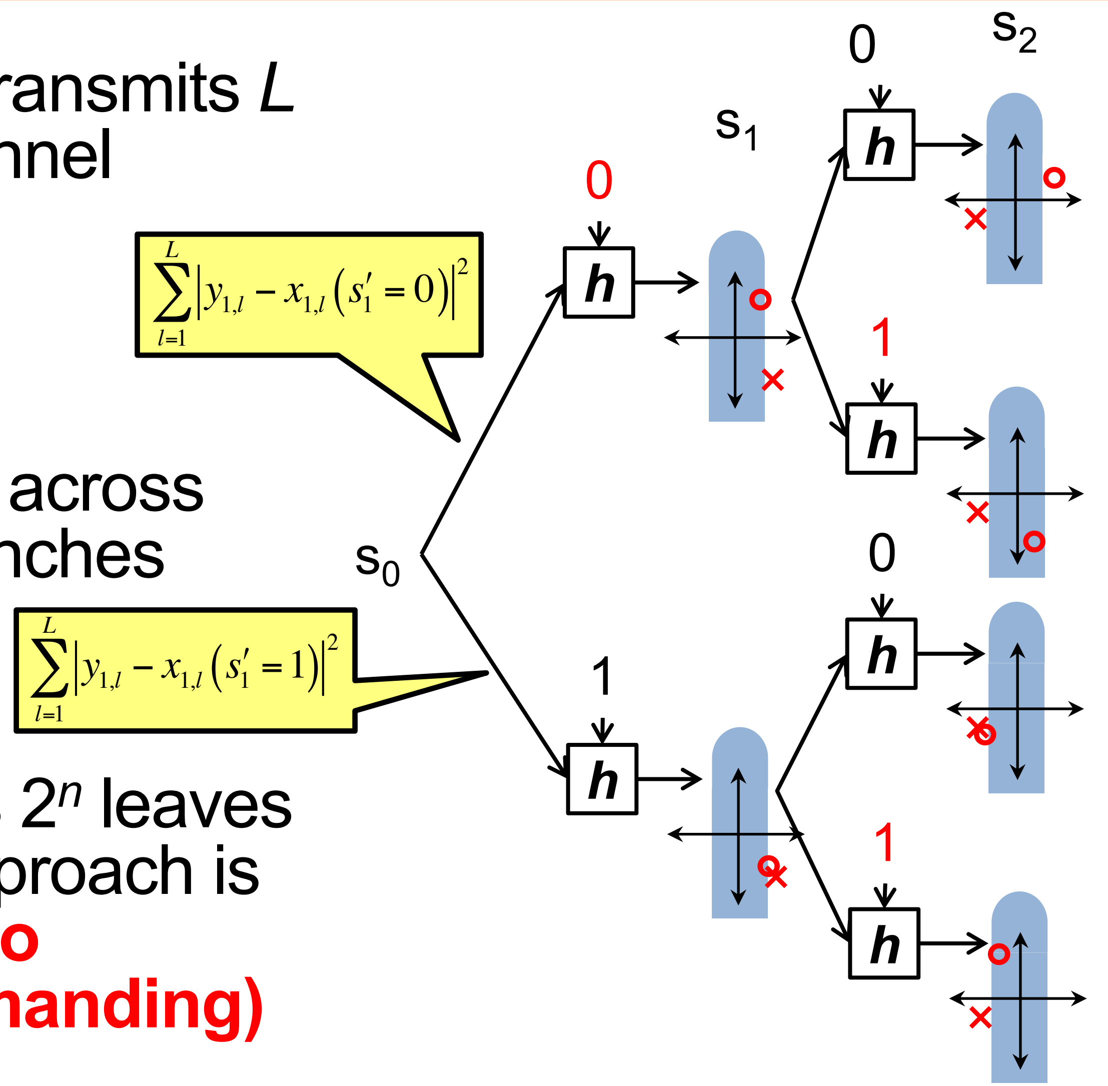
ML decoding over a tree

- General tree properties:
  - n/k levels, one per spine
  - Branching factor 2^k (per choice of k-bit message chunk)
- Let $\mathbf{s'}_t$ be the t^{th} spine value associated with all messages that share s'_t
- We find cost of a particular message by summing costs on path from root to leaf



ML decoding over a tree: Multiple passes

- Suppose the sender transmits *L* passes, in a poor channel
- Average (sum) metric across passes, and label branches
- However, the tree has 2^n leaves to compare so this approach is still **impracticable (too computationally demanding)**



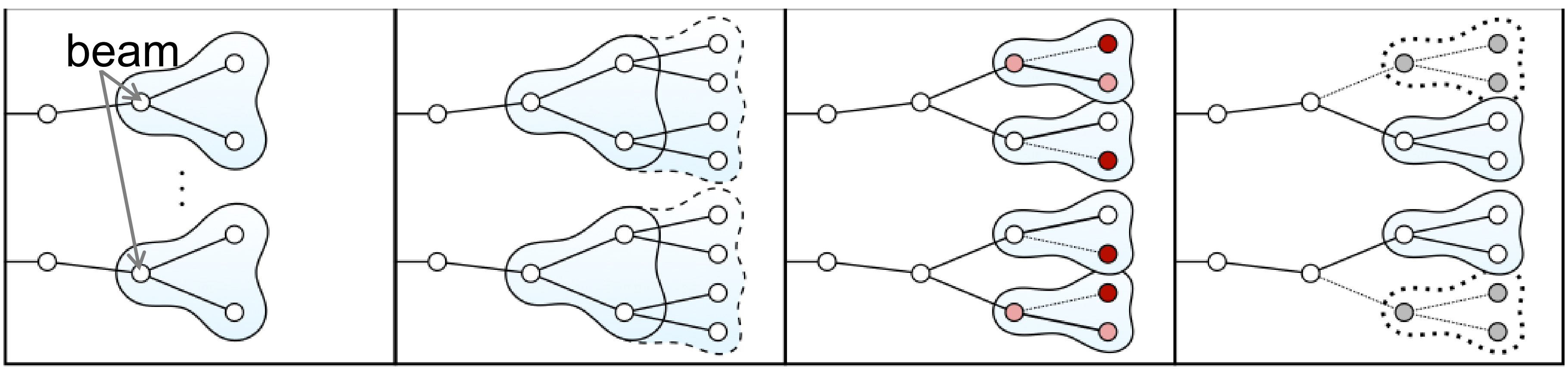
Efficiently exploring the tree

- Observation: Suppose the **ML message *M**** and some other message M' differ only in the i^{th} bit
  - Only symbols including and after index $\lceil i/k \rceil$ will disagree
  - So the **earlier** the error in M', the **larger** the cost
  - Can show that the "runners-up" to M^* differ only in the last O(log n) bits

- Consider the **best 100 leaves** in the ML tree:
  - Tracing back through the tree, they will have a **common ancestor** with M^* in O(log n) steps
  - This suggests a strategy in which we only **keep a limited number of ancestors**

Bubble decoder

- Maintain a ***beam*** of ***B*** tree node ancestors to explore, each to a certain depth ***d***
- Expand each ancestor, score every child, propagate best child score for each ancestor, pick B best survivors
- Example: $B = d = 2$, $k = 1$ (lighter color = better score)



(a) After step $i-1$ (b) Expand (c) Score (d) Pop

Decoding complexity

- The bubble decoder operates in $n/k - d$ **steps**
  - Each step explores $B{\cdot}2^{kd}$ nodes, evaluating the RNG L times
  - Selecting the best B candidates takes $B{\cdot}2^{k}$ comparisons

- **Overall cost:** $O((n/k)BL{\cdot}2^{kd})$ hashes, $O((n/k)B{\cdot}2^{k})$ comparisons

- Comparison with LDPC **belief propagation** algorithms
  - These operate in iterations, too, involve all message bits
  - But, these are also quite parallelizable
  - Hard to give exact head-to-head comparison

Adjusting the rate

- Spinal codes as described so far uses different numbers of passes to adjust the rate

- Two problems in Spinal codes **as described so far:**

1. Must transmit one full pass, so max out at *k* bits/symbol
   - Increase *k*? No: Decoding cost is **exponential** in *k*

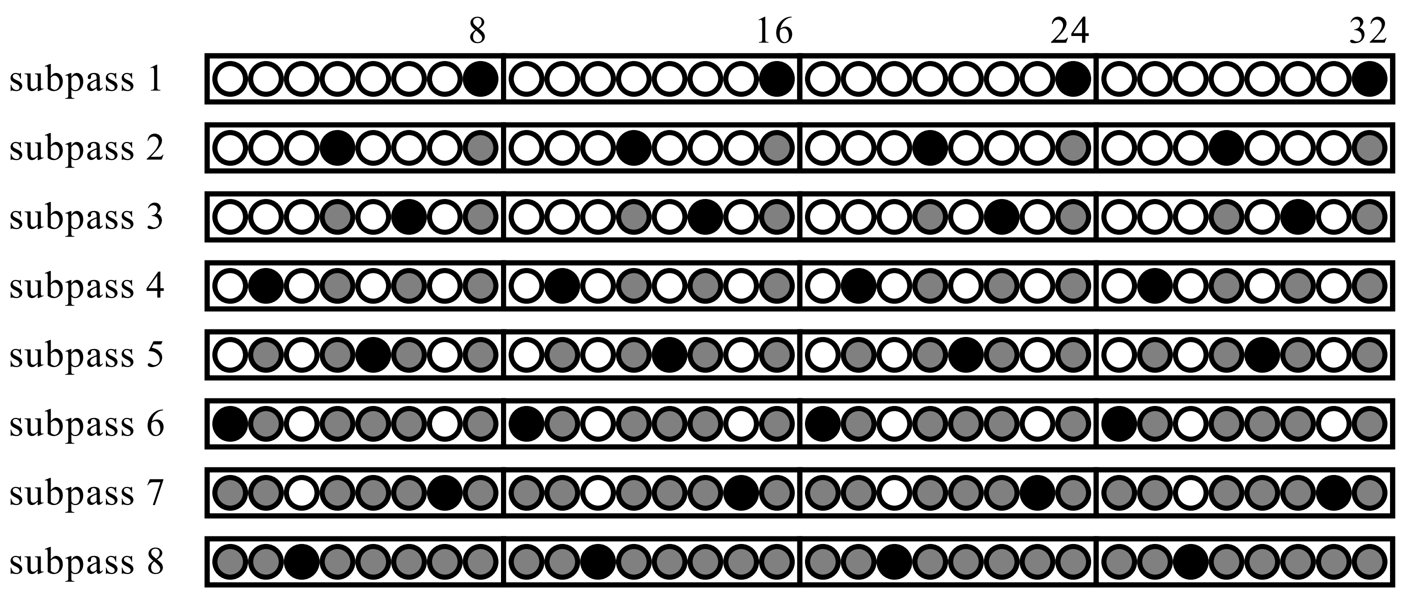
1. Sending *L* passes reduces rate to *k*/*L*—**abrupt drop**
   - Introduces plateaus in the rate versus SNR curve

Puncturing for higher and finer-controlled rates

- Idea: Systematically skip some spines
  - Sender and receiver agree on the pattern beforehand
  - Receiver can now attempt a decode before a pass concludes

- Decoder algorithm unchanged, missing symbols get zero score

- Max rate of this puncturing: 8·*k* bits/symbol



Framing at the link layer

- Sender and receiver need to maintain synchronization
  - Sender uses a short sequence number protected by a highly redundant code
- Unusual property of Spinal codes: **Shorter** message length *n* is **more** efficient
  - This is in opposition to the trend most codes follow
  - Divide the link-layer frame into shorter checksum-protected ***code blocks***
- If half-duplex radio, when should sender wait for feedback?
  - For more information, see *RateMore* (MobiCom '12)

Today

1. **Rateless *fountain* codes**
   – Luby Transform (LT) Encoding
   – LT Decoding

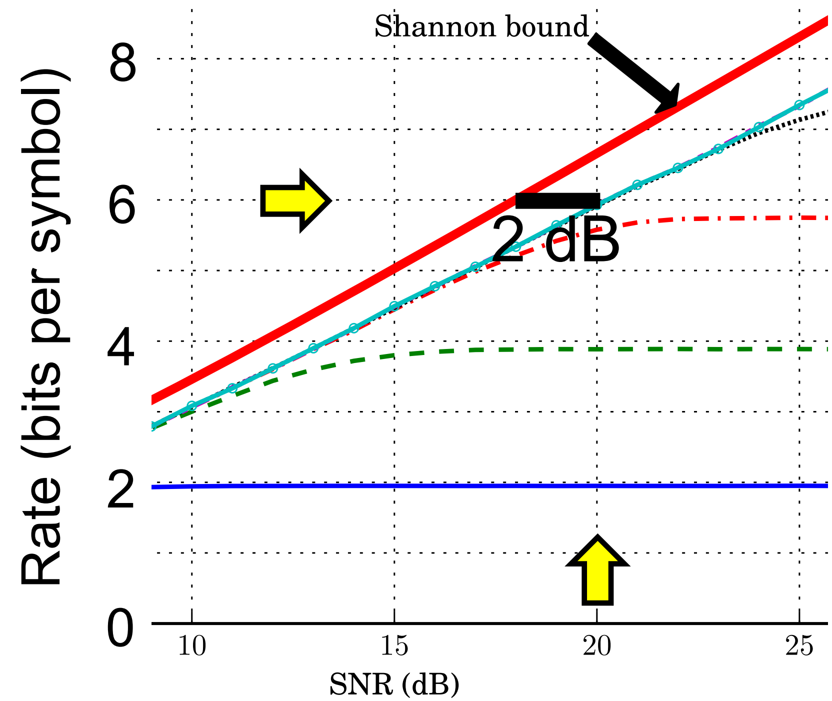
2. Rateless Spinal codes
   – Encoding Spinal Codes
   – Decoding Spinal codes
   – **Performance evaluation**

Methodology

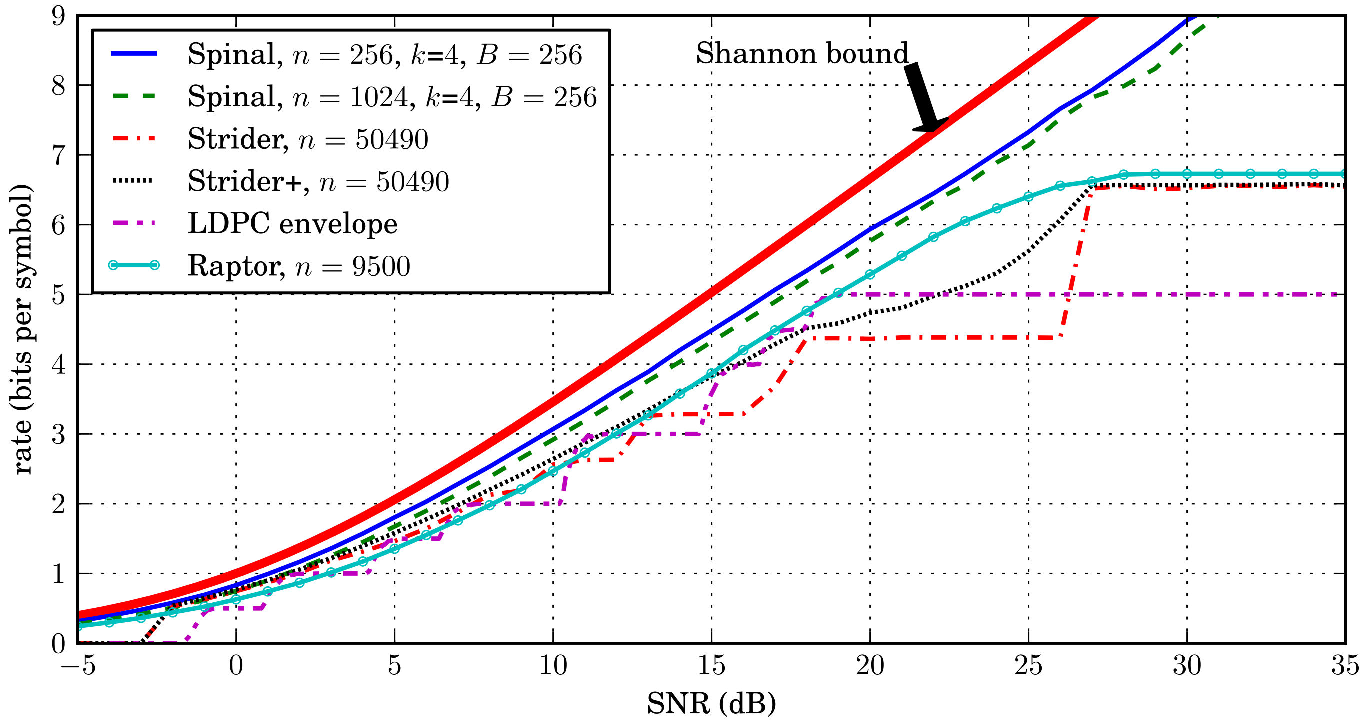
- Software simulation: Simulated wireless channel (additive white Gaussian noise and Rayleigh fading)

- Hardware platform: **Airblue** (FPGA based platform)
  - Real 10, 20 MHz bandwidth channels in 2.4 GHz ISM band

- ***Gap to capacity:*** How much more noise could a capacity-achieving code tolerate at same rate?
  - Smaller gap is better
  - *e.g.*: This code achieves six bits/symbol at 20 dB SNR, for a **2 dB gap to capacity**



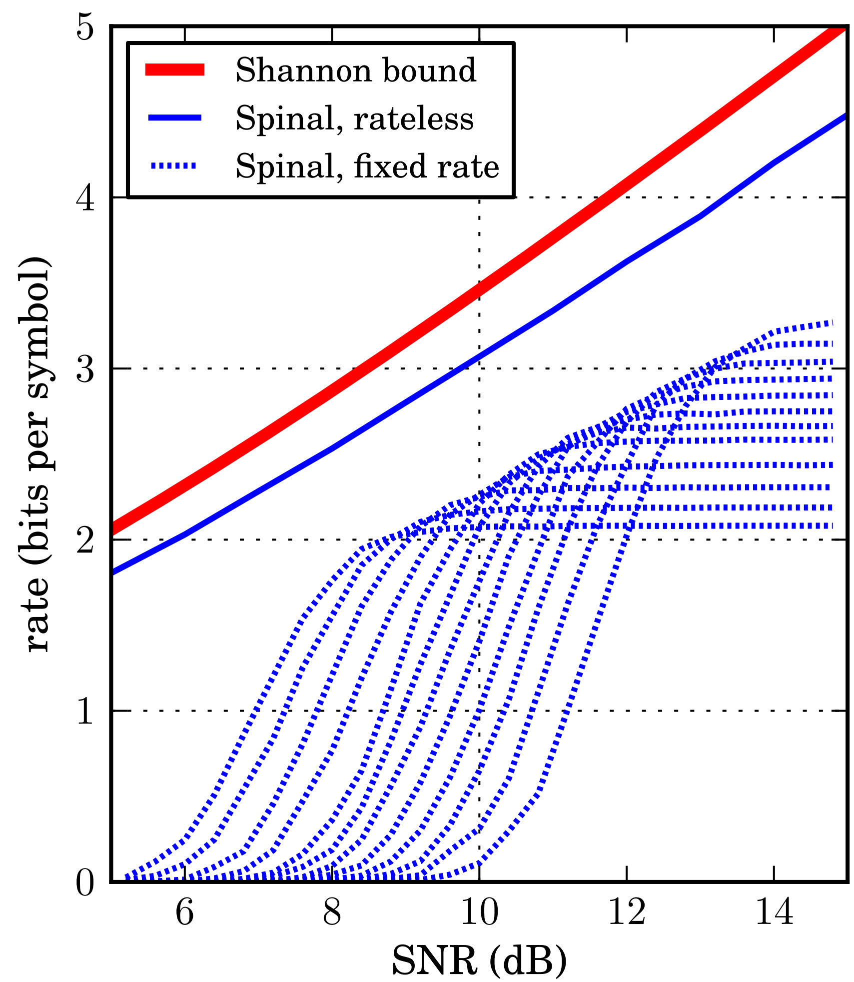
Spinal codes: Higher rate on AWGN channel



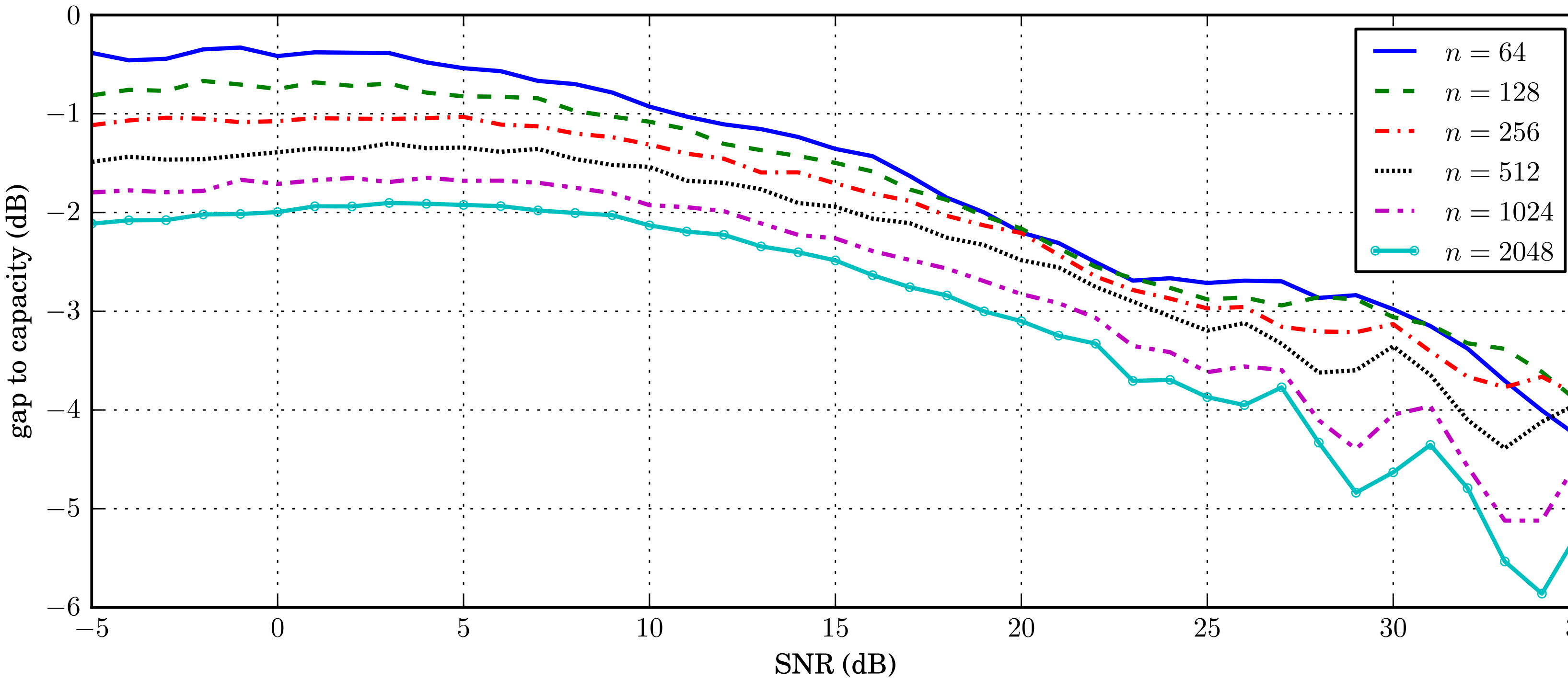
- Simulated AWGN channel: no link-layer performance effects here
- **LDPC envelope:** Choose best-performing rated LDPC code at each SNR to mimic the best a rate adaptation strategy could do
- **Strider+:** Strider + puncturing: finer rate control, but significant gap to capacity

Rateless codes can “hedge their bets”

- Constant SNR means constant **average** noise power
  - But, noise impacting **any particular symbol(s)** may be higher or lower
- Rated codes must be risk averse (send at lower rate)
- Rateless codes can decode with **fewer symbols** when noise is **momentarily lower**
- But this result **requires perfect and instantaneous feedback** so the rateless code knows when to stop



Spinal codes: Better at sending short messages



- **Longer** code block means more opportunities to **prune correct path**
  - So Spinal codes achieves **better** performance (smaller gap to capacity) with **smaller** code block length *n*

- We can see artifacts due to puncturing at higher SNRs

Spinal Codes: Conclusion

- Spinal Codes give performance **close to Shannon capacity**
- Eliminate the need to run a bit rate adaptation algorithm
- Simpler design and better performance
- Link layer design more open, **incurs overhead between transmissions**

Midterm format

- **Timing: 60 minutes** in a **90 minute** timeslot

1. **True/False/Don't Know** questions
   - One point for a **correct** T/F response
   - No effect for a don't know response or no response
   - Minus one point for an **incorrect** T/F response

   - Rescaled as a section with a zero floor

2. **Short answer** questions
   - One to two, each on a theme

Friday Precept:
Midterm Review

Tuesday Topic:
Signals and Systems Preliminaries

Next Thursday:
In-Class Midterm