Bit Rate Adaptation and Rateless Codes

COS 463: Wireless Networks
Lecture 10
Kyle Jamieson

[Parts adapted from H. Balakrishnan, M. Vutukuru]

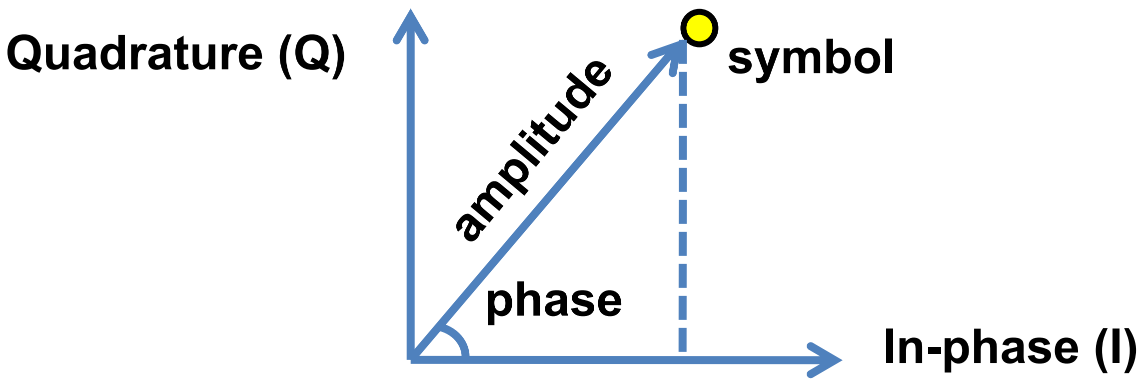
Today

1. **Bit Rate Adaptation**
   – Modulation
   – Bit error rate (BER) and signal to noise ratio (SNR)
   – Adapting modulation and error control coding

2. Rateless Codes: Spinal Codes

What is modulation?

- ***Modulate*** means **to change.** Change what?
  - The **amplitude** and ***phase* (angle)** of a radio ***carrier signal***

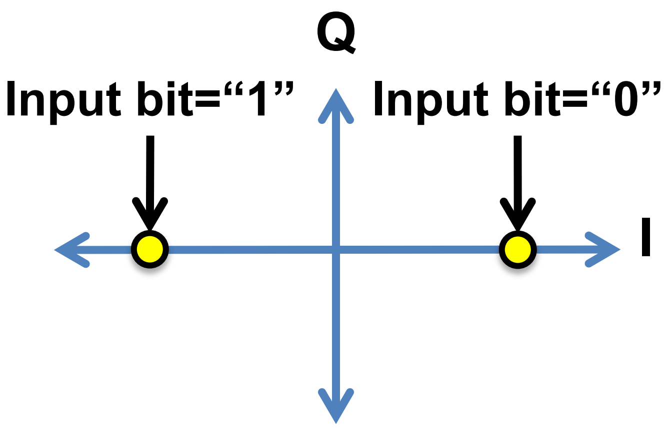


- ***Digital modulation:*** Use only a **finite set** of choices (*i.e.*, ***symbols***) for how to change the carrier and phase

From information bits to symbols…

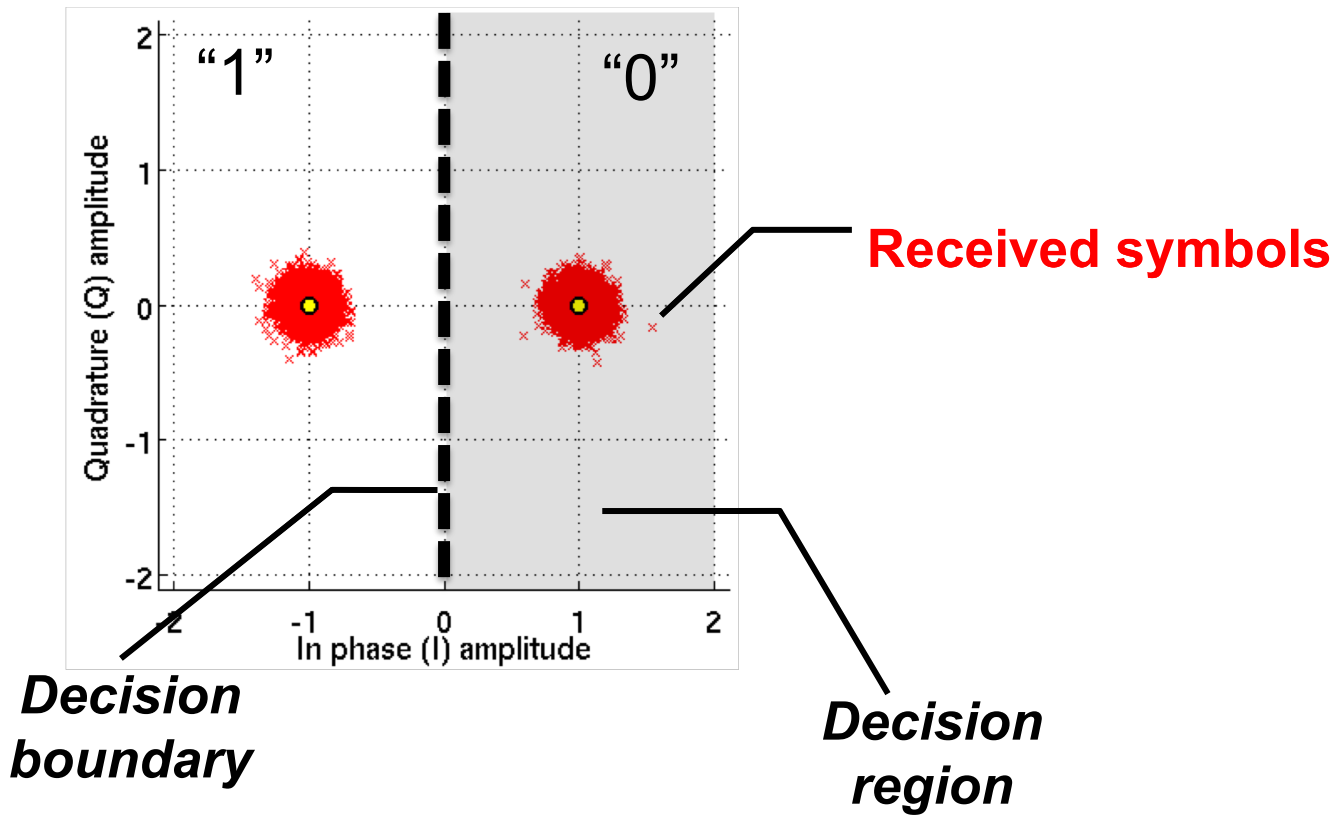
- Pick two symbols (**binary**)
  - The information bit decides which symbol you transmit

  - **Phase shift** of 180 degrees between the two symbols, so called ***binary phase shift keying***



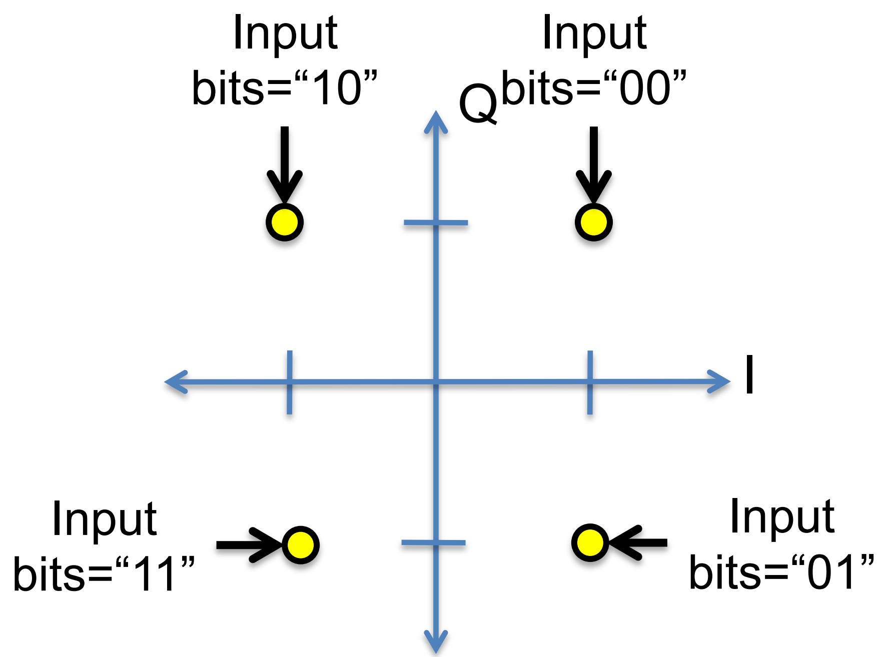
…and back to bits!

Received BPSK constellation



From bits to symbols, twice as fast

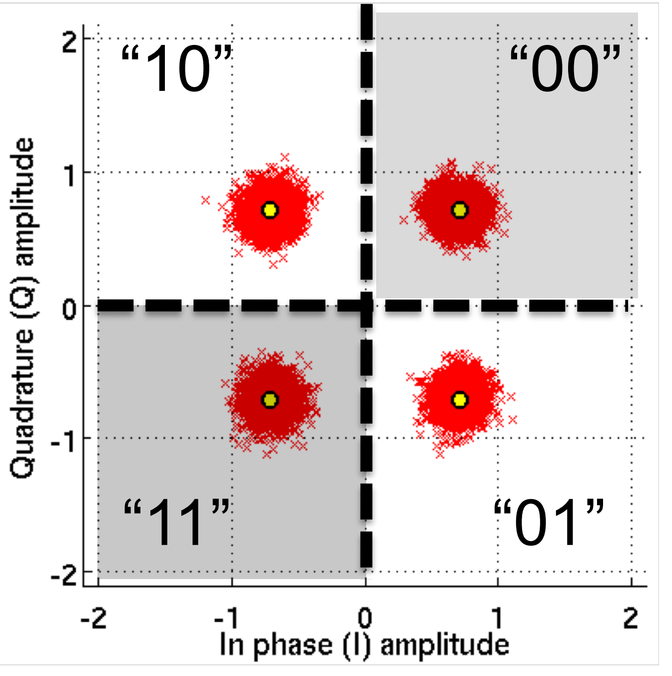
Quadrature phase shift keying (QPSK)



Sending $\log_2 4 = 2$ bits/symbol

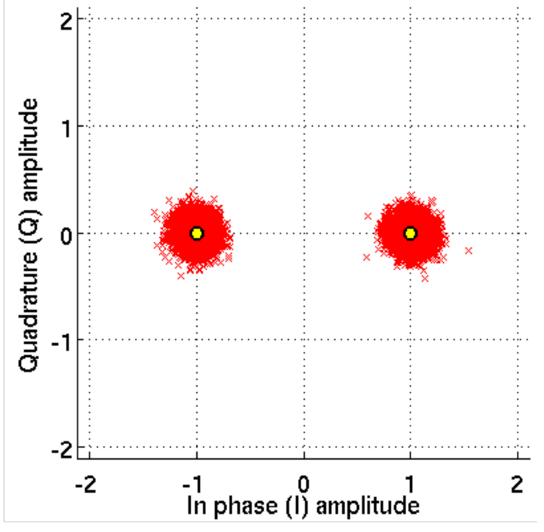
…and back to bits, twice as fast!

Received QPSK constellation

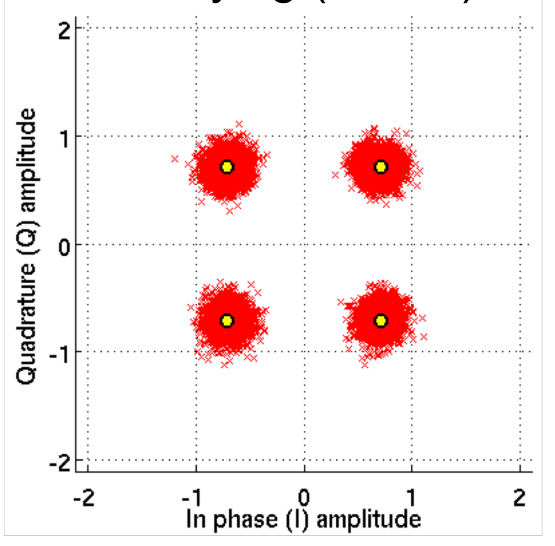


Change modulations, increase bitrate

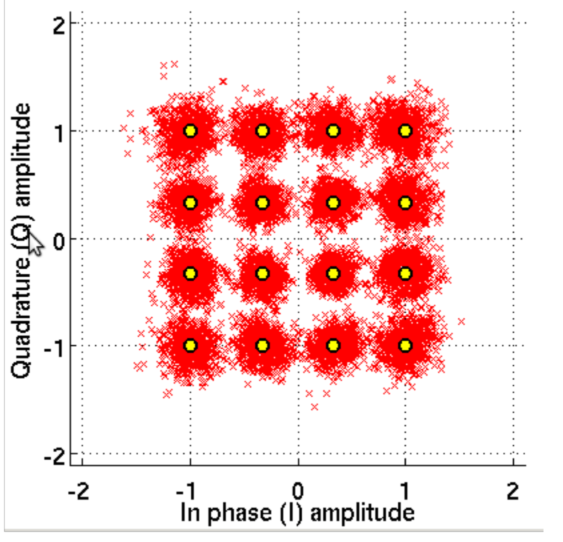
Binary Phase-Shift Keying (BPSK)



Quadrature Phase-Shift Keying (QPSK)

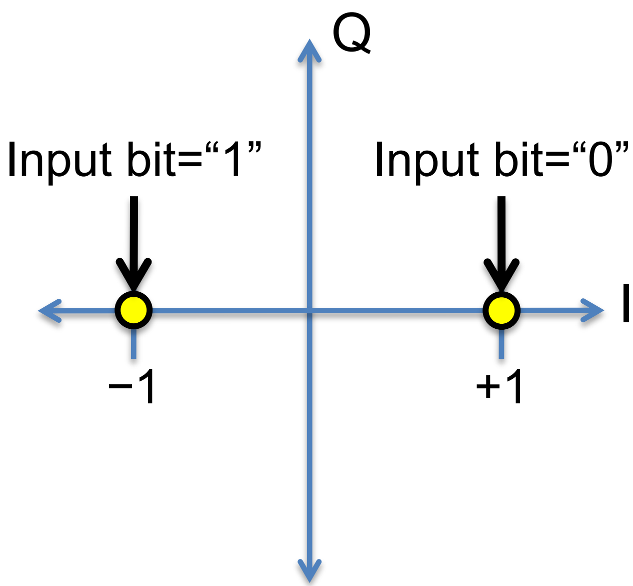


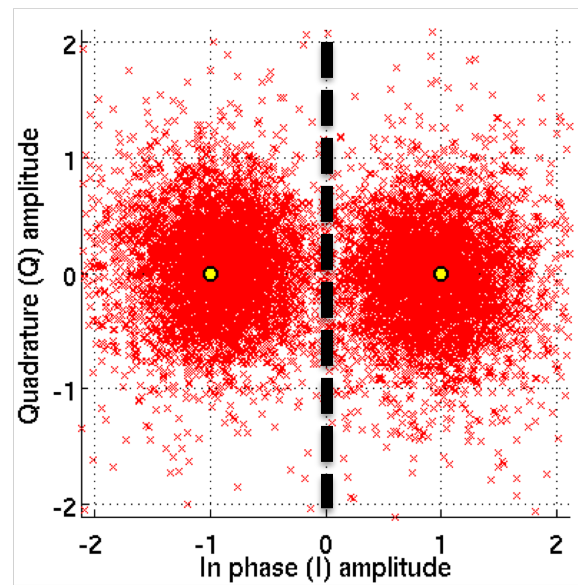
16-Quadrature Amplitude Modulation (16-QAM)



The wireless channel

Transmitted signal: $s(t)$ **Received signal:** $r(t) = s(t) + \text{noise}$





Signal to Noise Ratio (SNR)

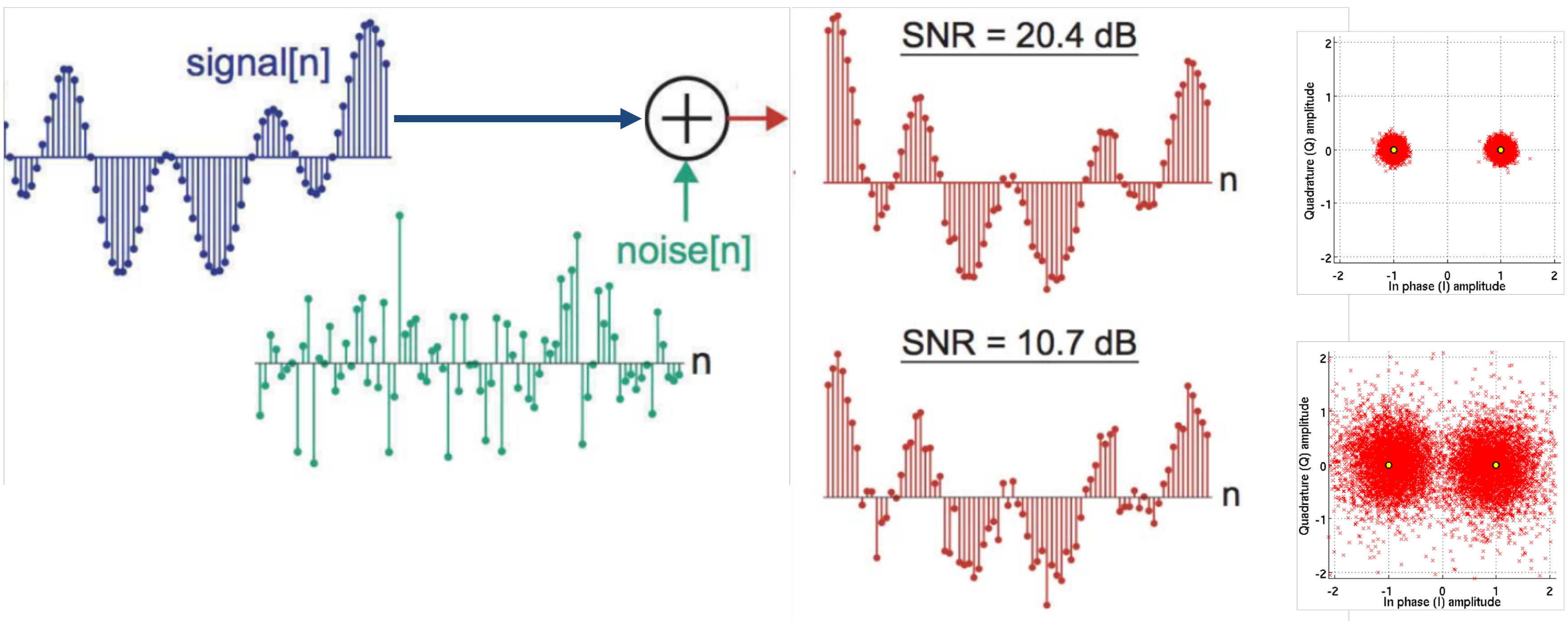
- ***Signal-to-Noise Ratio*** (SNR) measures power ratio between a signal of interest and background noise: $\mathrm{SNR} = \frac{P_{signal}}{P_{noise}}$

- SNR is often expressed in ***decibels*** (dB), 10 times the base-10 logarithm of a quantity: $\mathrm{SNR}\ (\mathrm{dB}) = \log_{10}\left(\frac{P_{signal}}{P_{noise}}\right)$

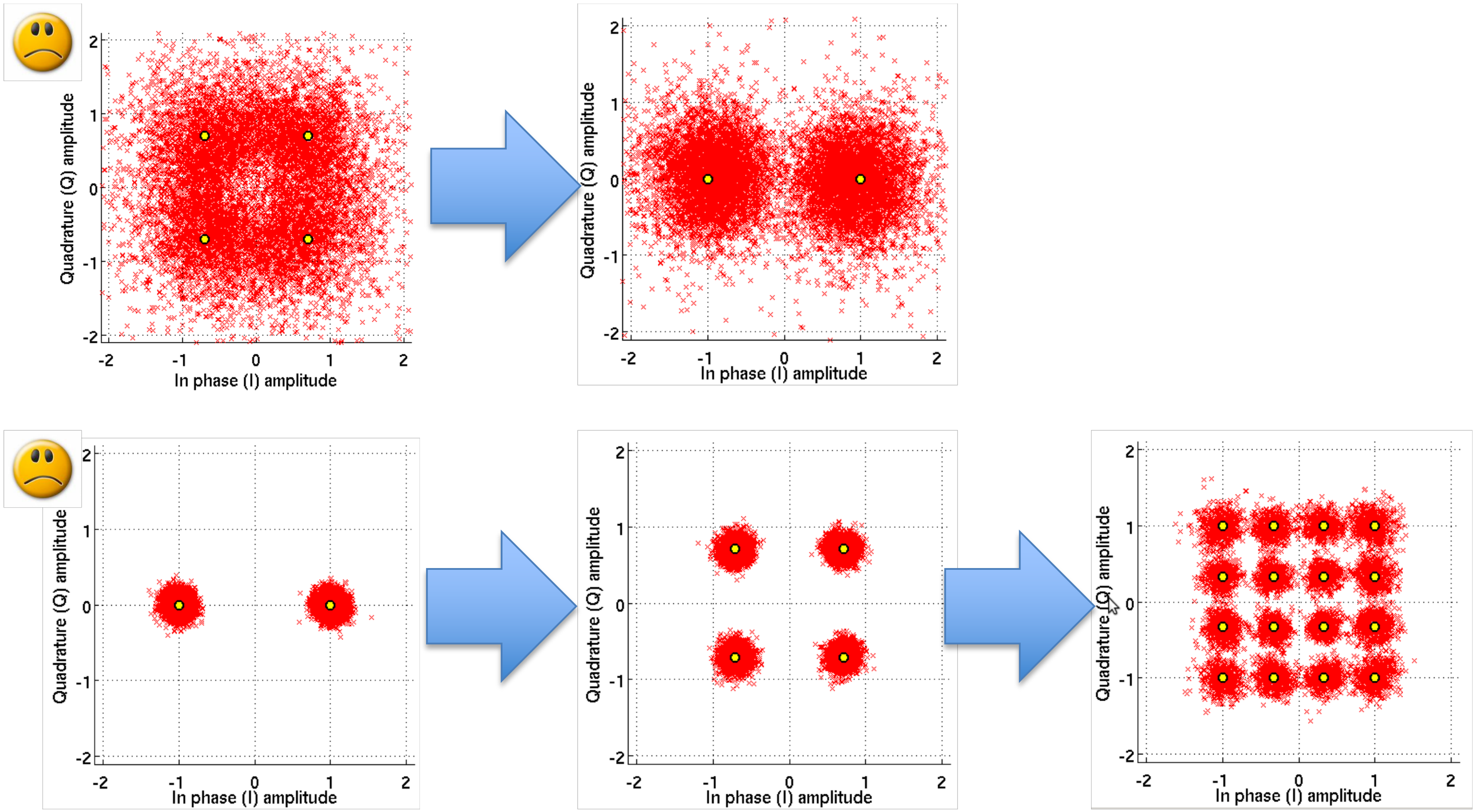
SNR (dB)	SNR
30	1,000
20	100
10	10
0	**1 (equal)**
-10	0.1
-20	0.01
-30	0.001

Visualizing Signal to Noise Ratio

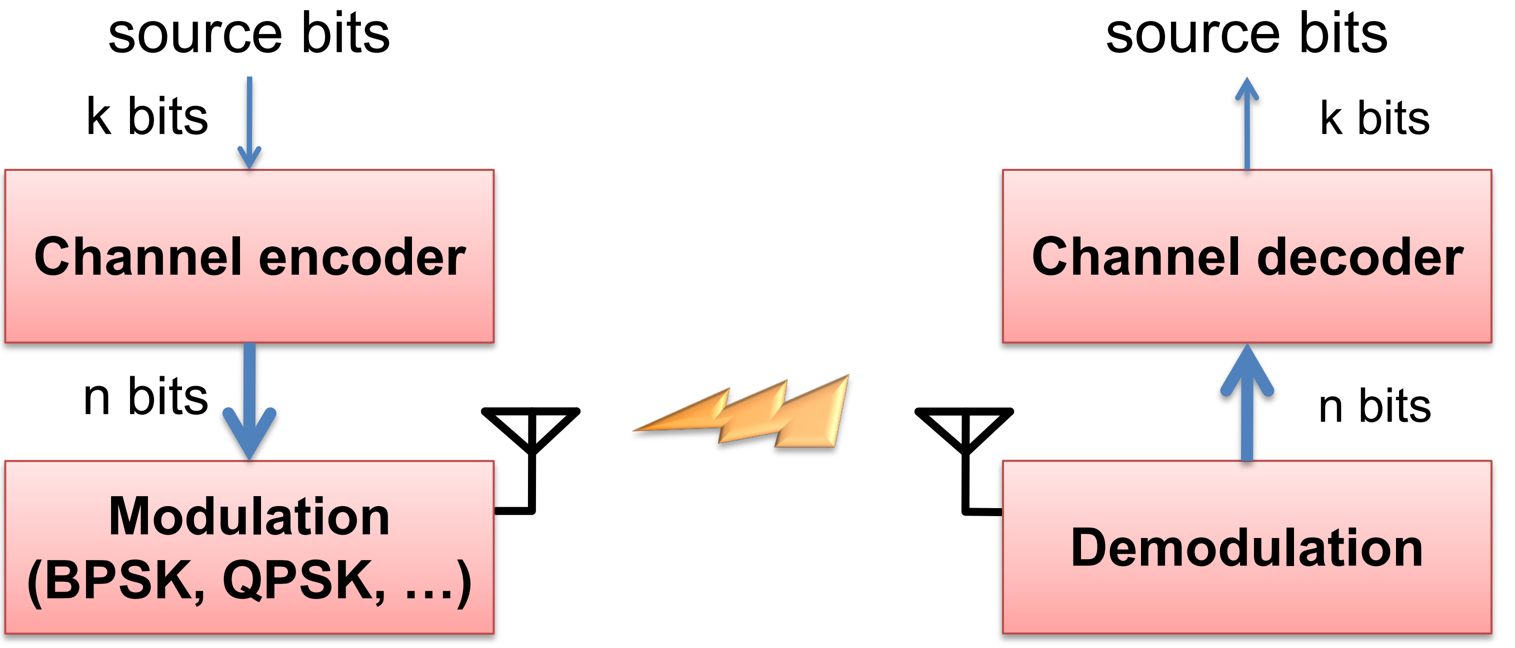
Signal view: **Constellation view:**



Modulation adaptation



Bit Rate Adaptation in the Physical Layer

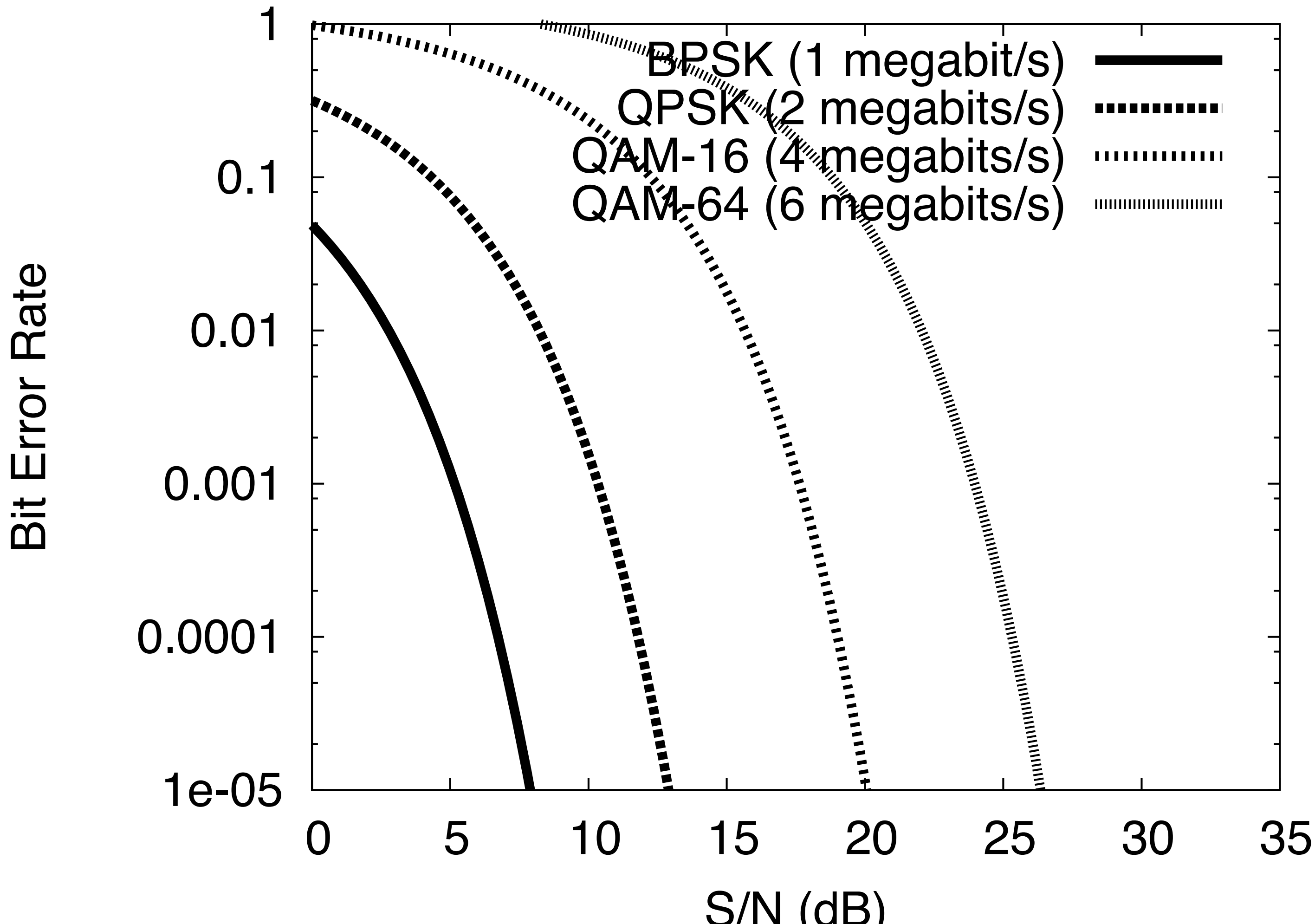


Code rate: R = k/n

802.11: adapt code rate, modulation

Bit-rate	802.11 Standards	DSSS or OFDM	Modulation	Bits per Symbol	Coding Rate	Mega-Symbols per second
1	b	DSSS	BPSK	1	1/11	11
2	b	DSSS	QPSK	2	1/11	11
5.5	b	DSSS	CCK	1	4/8	11
11	b	DSSS	CCK	2	4/8	11
6	a/g	OFDM	BPSK	1	1/2	12
9	a/g	OFDM	BPSK	1	3/4	12
12	a/g	OFDM	QPSK	2	1/2	12
18	a/g	OFDM	QPSK	2	3/4	12
24	a/g	OFDM	QAM-16	4	1/2	12
36	a/g	OFDM	QAM-16	4	3/4	12
48	a/g	OFDM	QAM-64	6	2/3	12
54	a/g	OFDM	QAM-64	6	3/4	12

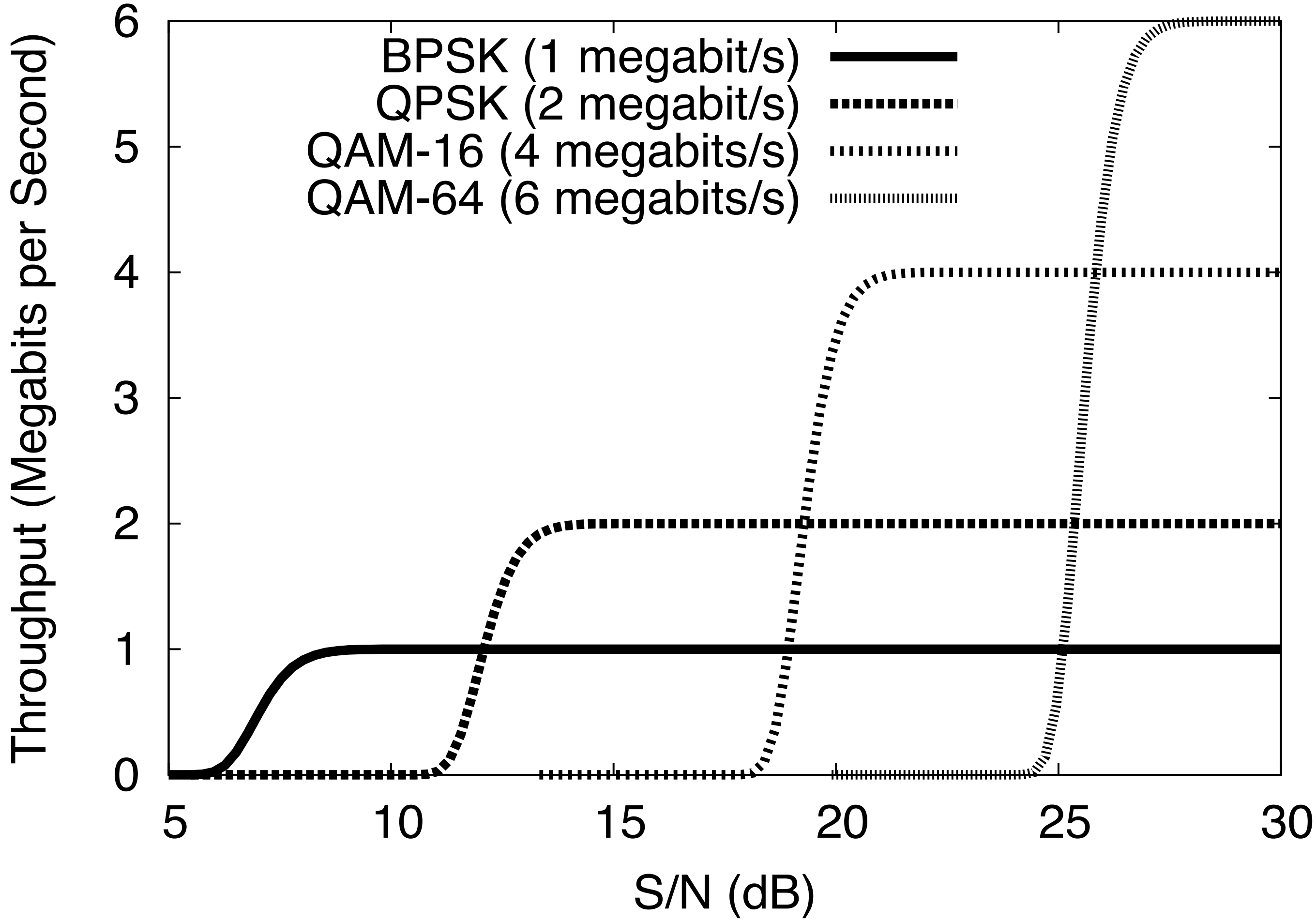
BER vs SNR



1
0.1
0.01
0.001
0.0001
1e-05
Bit Error Rate
0
5
10
15
20
25
30
35
S/N (dB)
BPSK (1 megabit/s)
QPSK (2 megabits/s)
QAM-16 (4 megabits/s)
QAM-64 (6 megabits/s)

Packetized throughput

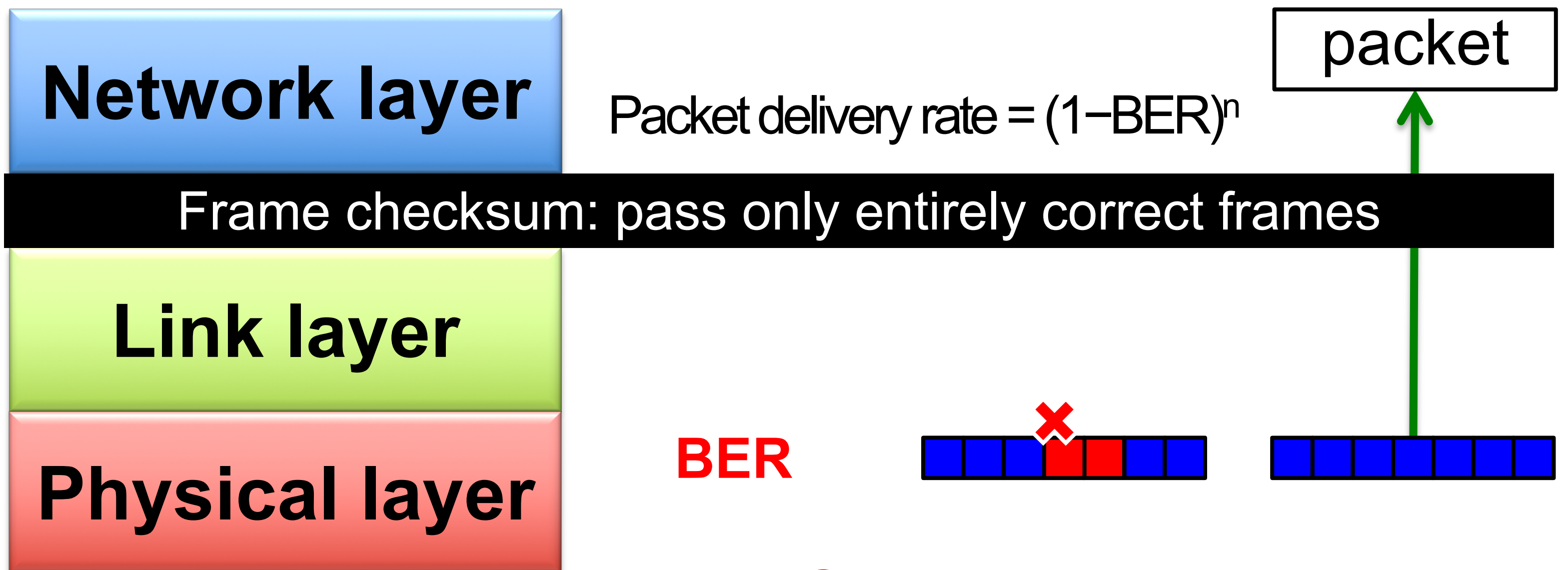
Throughput = delivery rate × bitrate = $(1 - \text{BER})^n \times$ bitrate



$n = 1500 \times 8$ bits

Link/PHY checks packet integrity

Throughput = delivery rate × bitrate = $(1 - BER)^n$ × bitrate

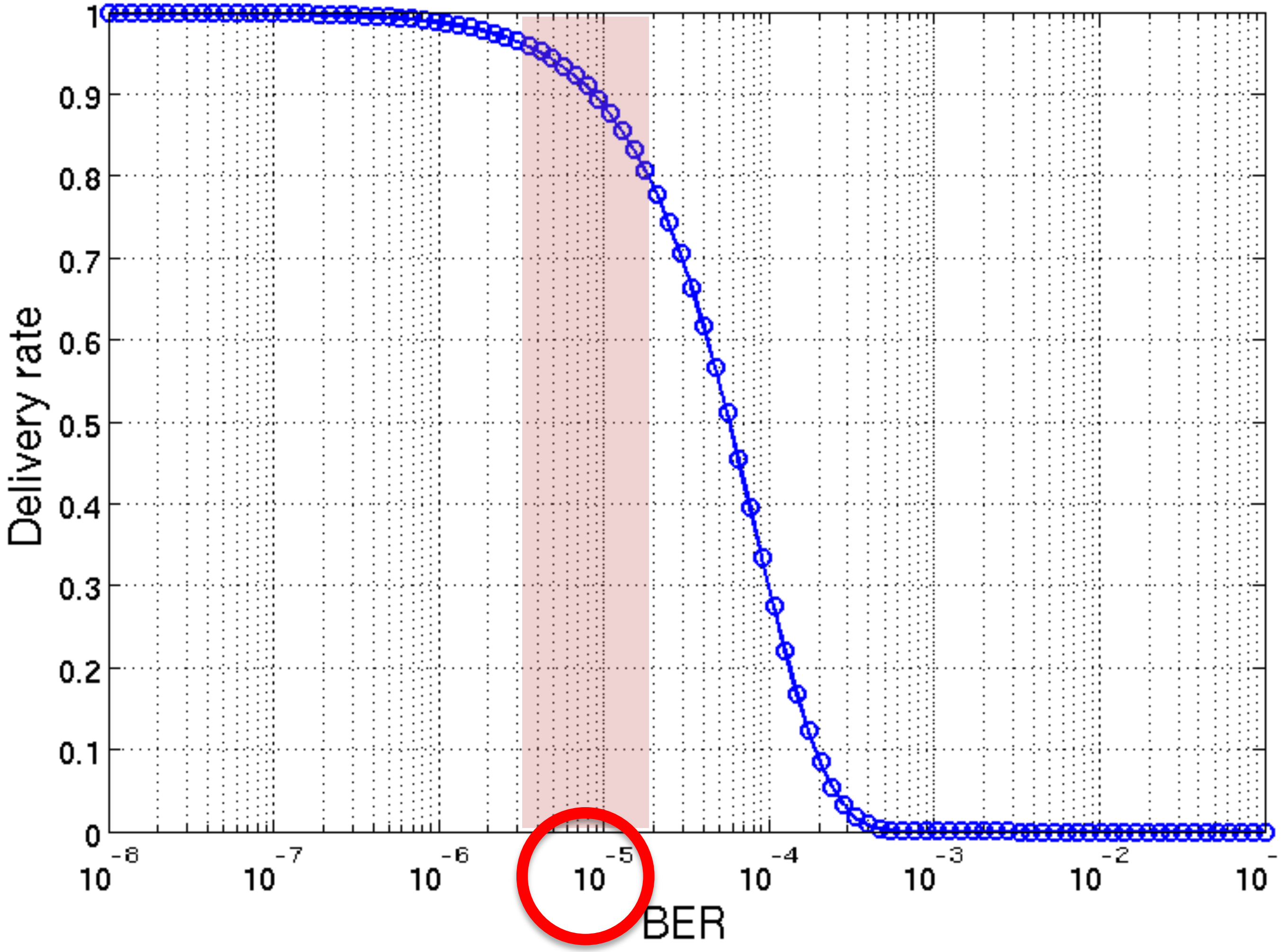


- Change modulation
- Change coding rate

$n = 1500 \times 8$ bits

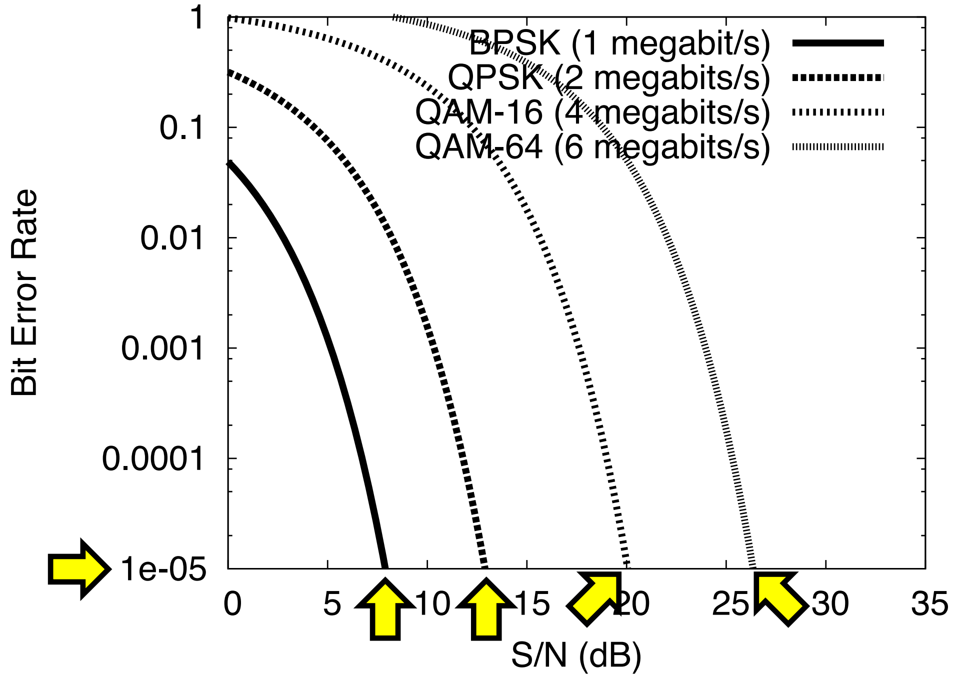
Delivery rate vs BER

Throughput = delivery rate × bitrate = $(1 - \text{BER})^n$ × bitrate



BER vs SNR

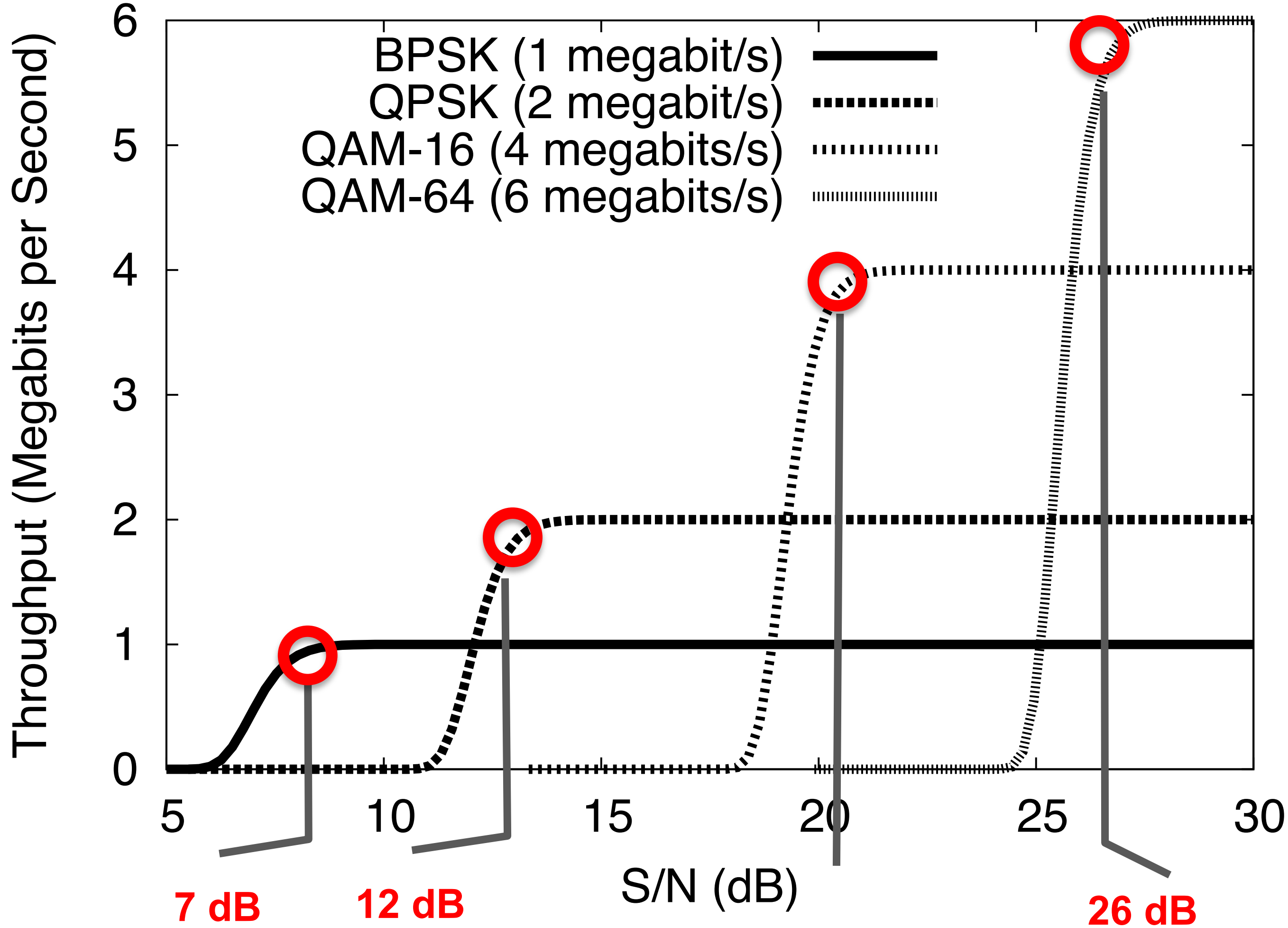
- Let's go back to the BER vs SNR graph
  - For each modulation: *What are the SNRs required for BER < 10^{-5}?*



- BPSK: 7 dB; QPSK: 12 dB; 16-QAM: 20 dB; 64-QAM: 26 dB

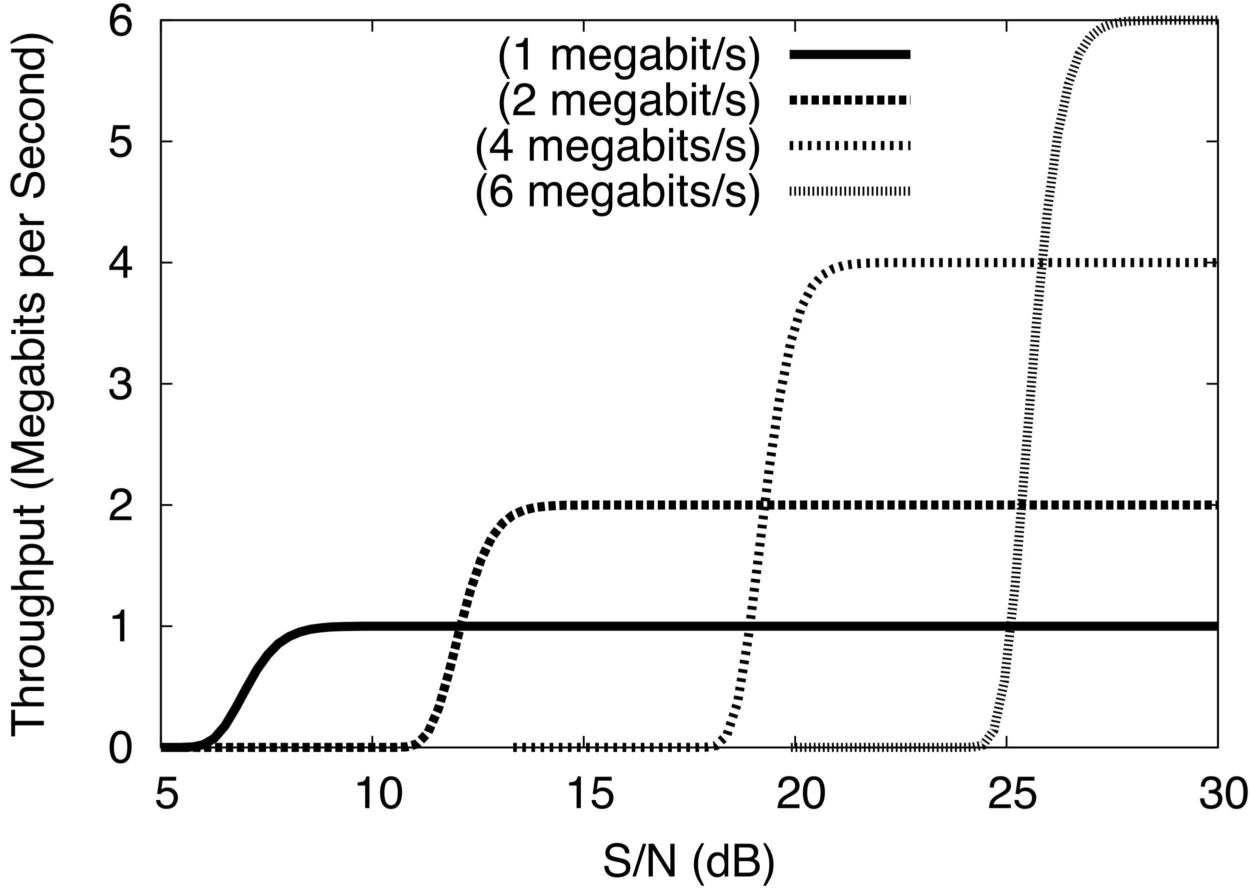
Fixed-rate codes *require* channel adaptation

Throughput = delivery rate × bitrate = $(1 - BER)^n$ × bitrate



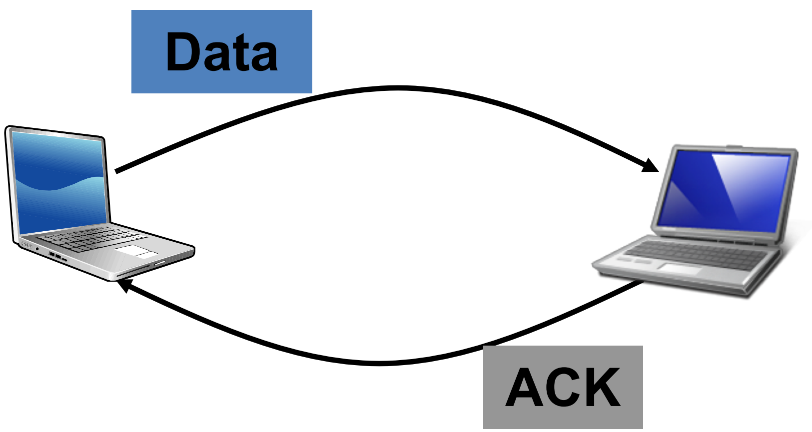
n = 1500 × 8 bits

Fixed-rate codes *require* channel adaptation



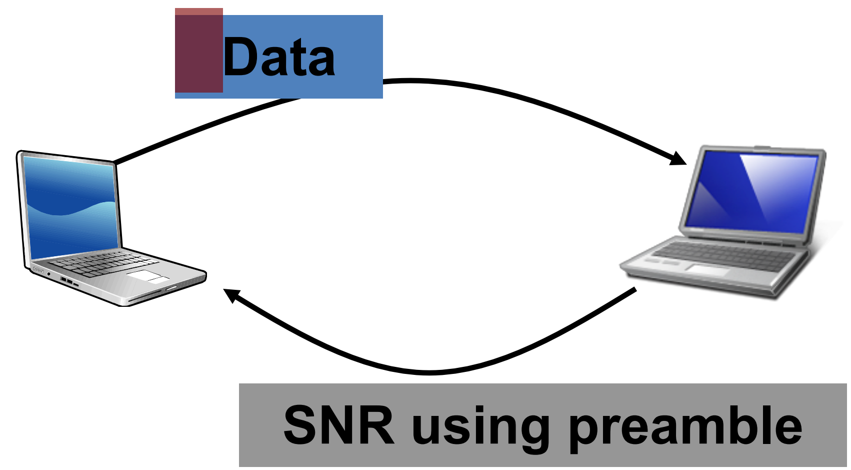
Existing rate adaptation algorithms

Frame-based



Estimate frame loss rate at each bit rate

SNR/BER-based



Lookup table:
SNR/BER → best rate

Today

1. Bit Rate Adaptation

2. **Rateless Codes: Spinal Codes**

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)

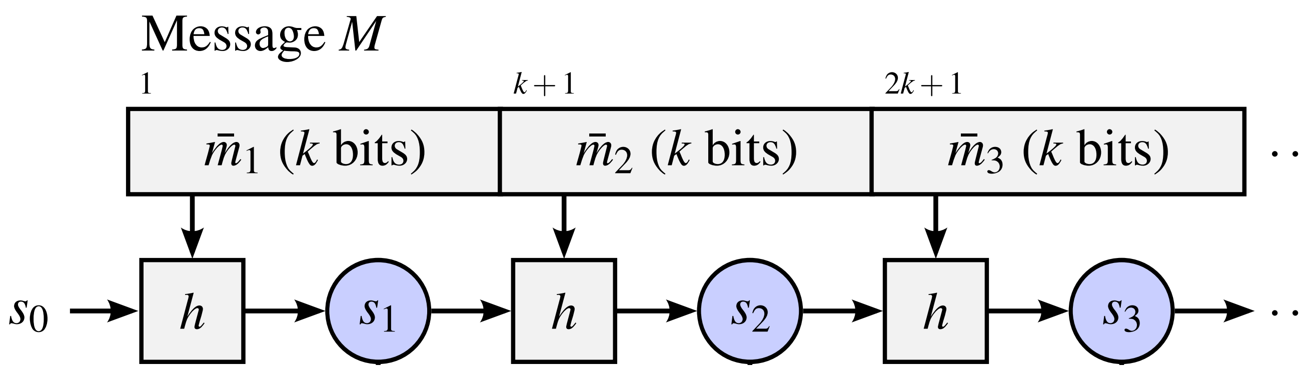
- **Setting:** Multicast or unicast links
- **Sender** sends a potentially limitless stream of encoded bits
- **Receiver(s)** collect bits until they are reasonably sure that they can recover the content from the received bits, then send **STOP** feedback to sender
- **Automatic adaptation**: Receivers with larger loss rate need longer to receive the required information

Today

1. Bit Rate Adaptation

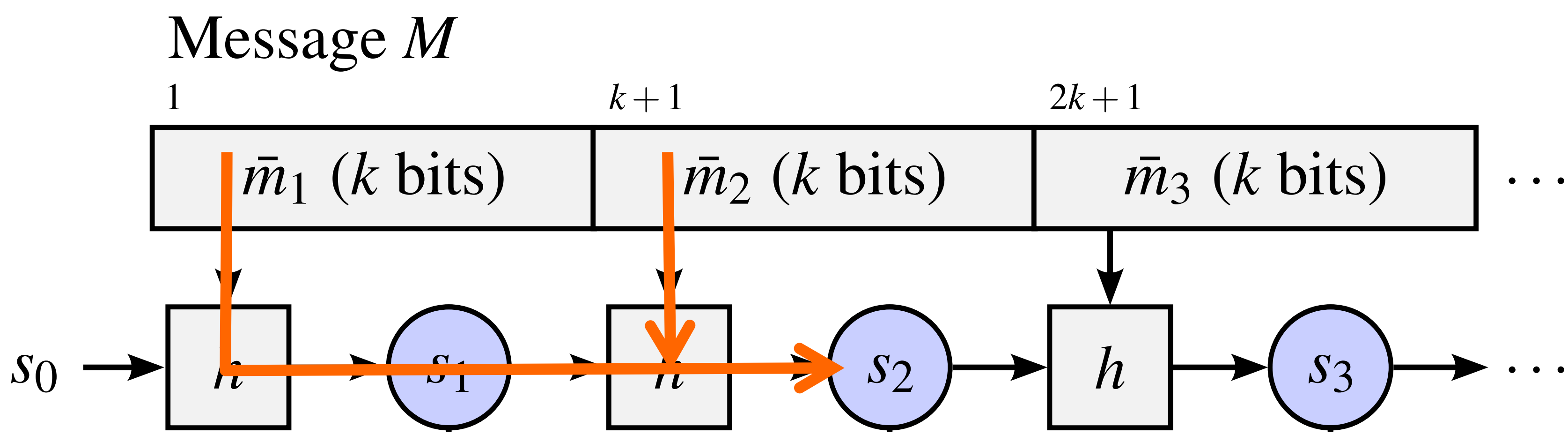
2. **Rateless Codes: Spinal Codes**
   - **Encoder structure**
   - Decoder structure

Spinal encoder: Computing the spines



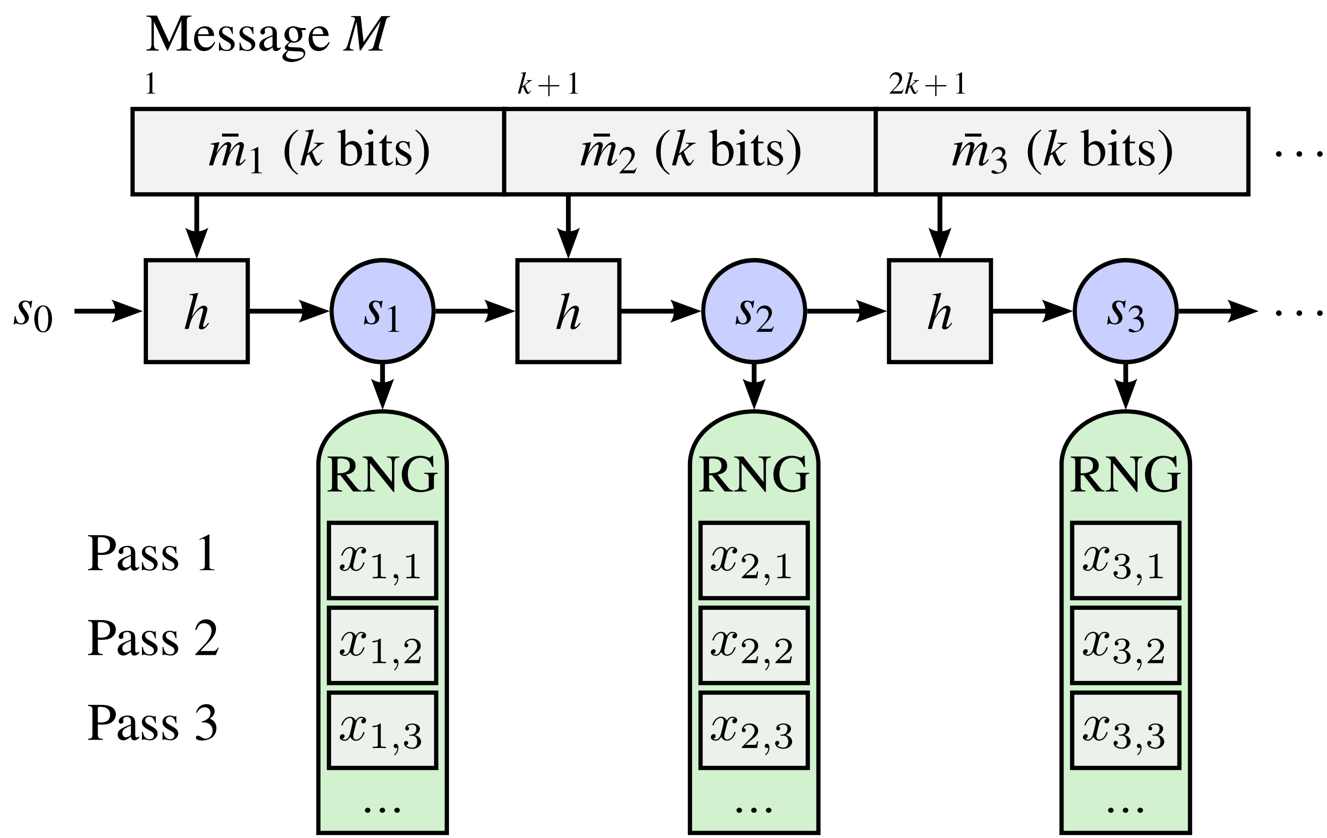
- Start with a hash function *h* and an initial random *v*-bit ***state*** s_0
  - Sender and receiver agree on *h* and s_0 *a priori*
- Sender divides its *n*-bit ***message*** *M* into *k*-bit ***chunks*** m_i
- *h* maps the state and a message chunk into a new state
  - The *v*-bit states $s_1, \ldots, s_{\lceil n/k \rceil}$ 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

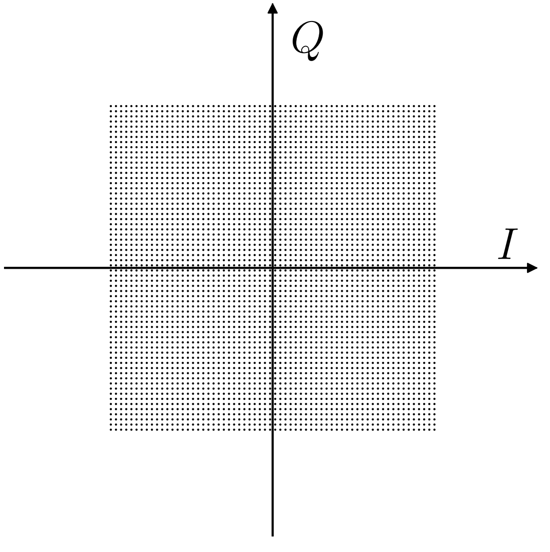


- Each spine seeds a pseudorandom number generator ***RNG***
- RNG generates a sequence of c-bit numbers $x_{i,l}$ called ***symbols***
- Encoder output is a series of ***passes,*** each of ⌈n/k⌉ symbols

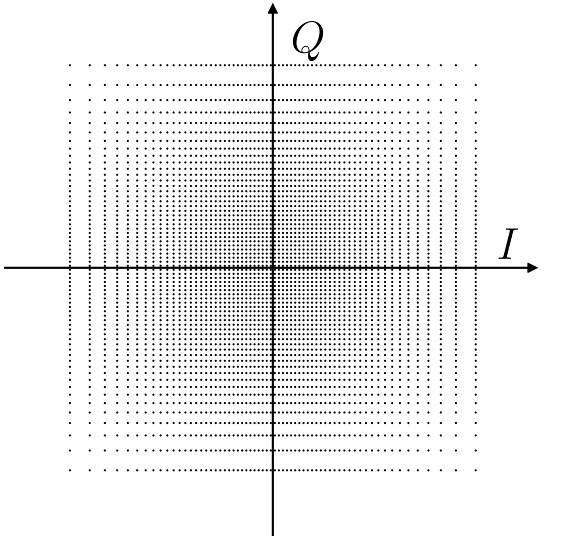
Spinal encoder: RNG to symbols

- 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



Truncated Gaussian



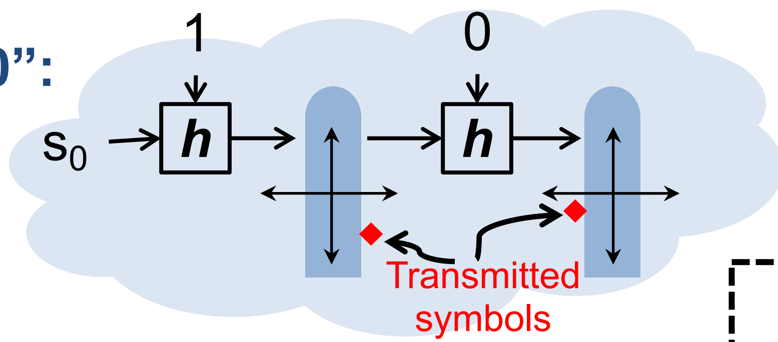
Today

1. Bit Rate Adaptation

2. **Rateless Codes: Spinal Codes**
   - Encoder structure
   - **Decoder structure**
     - **“Maximum-likelihood” decoding**
     - Practical “Bubble” Decoder
     - Puncturing for higher rate
     - Performance

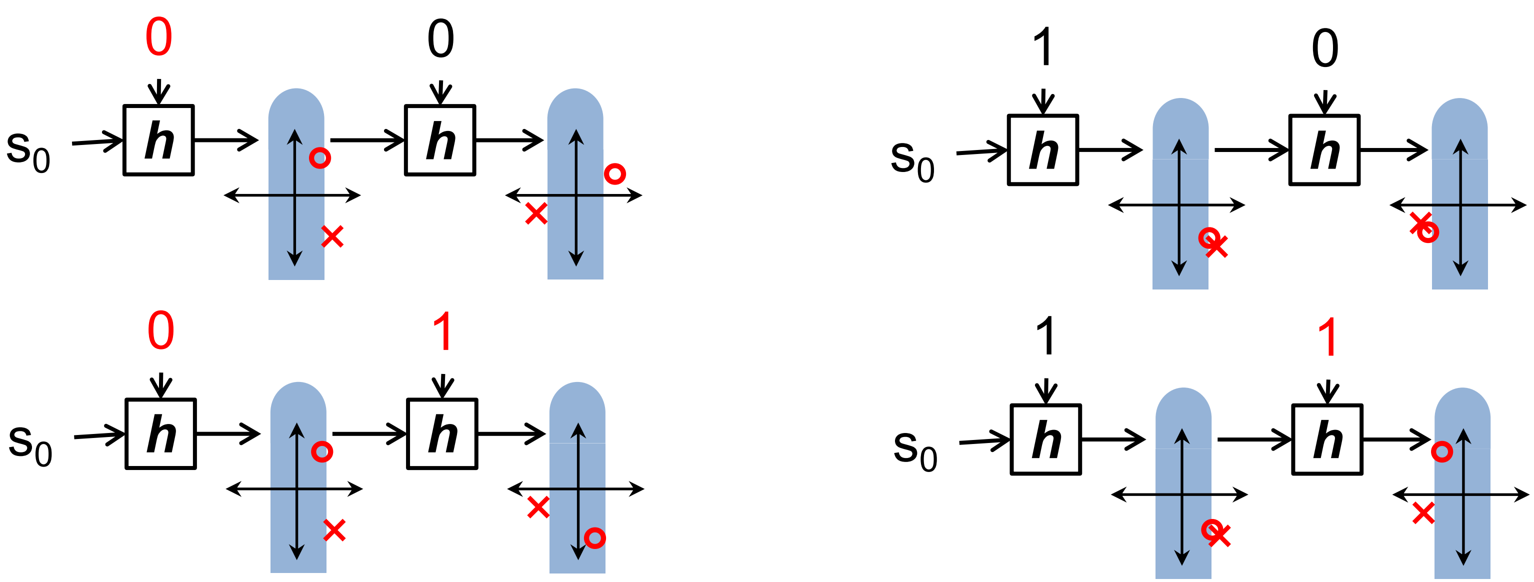
Decode by replaying the encoder

Sender transmits "1", "0":



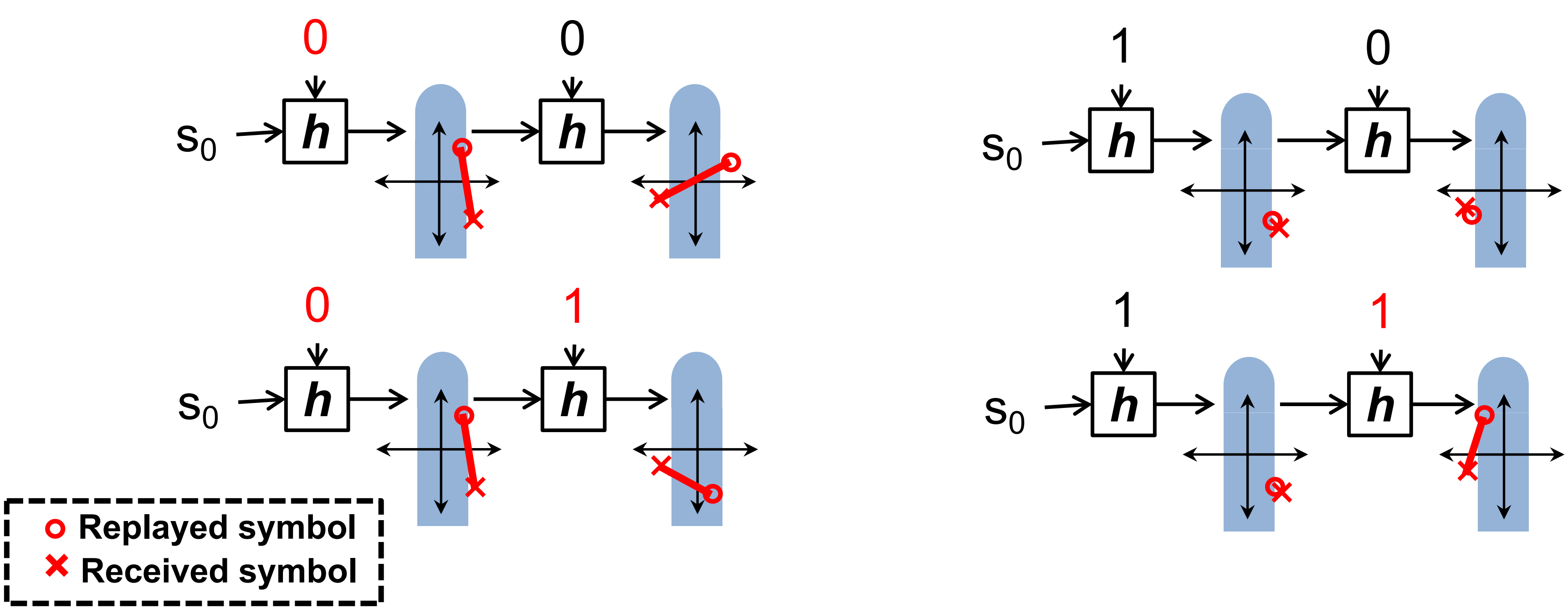
○ **Replayed symbol**
× **Received symbol**

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

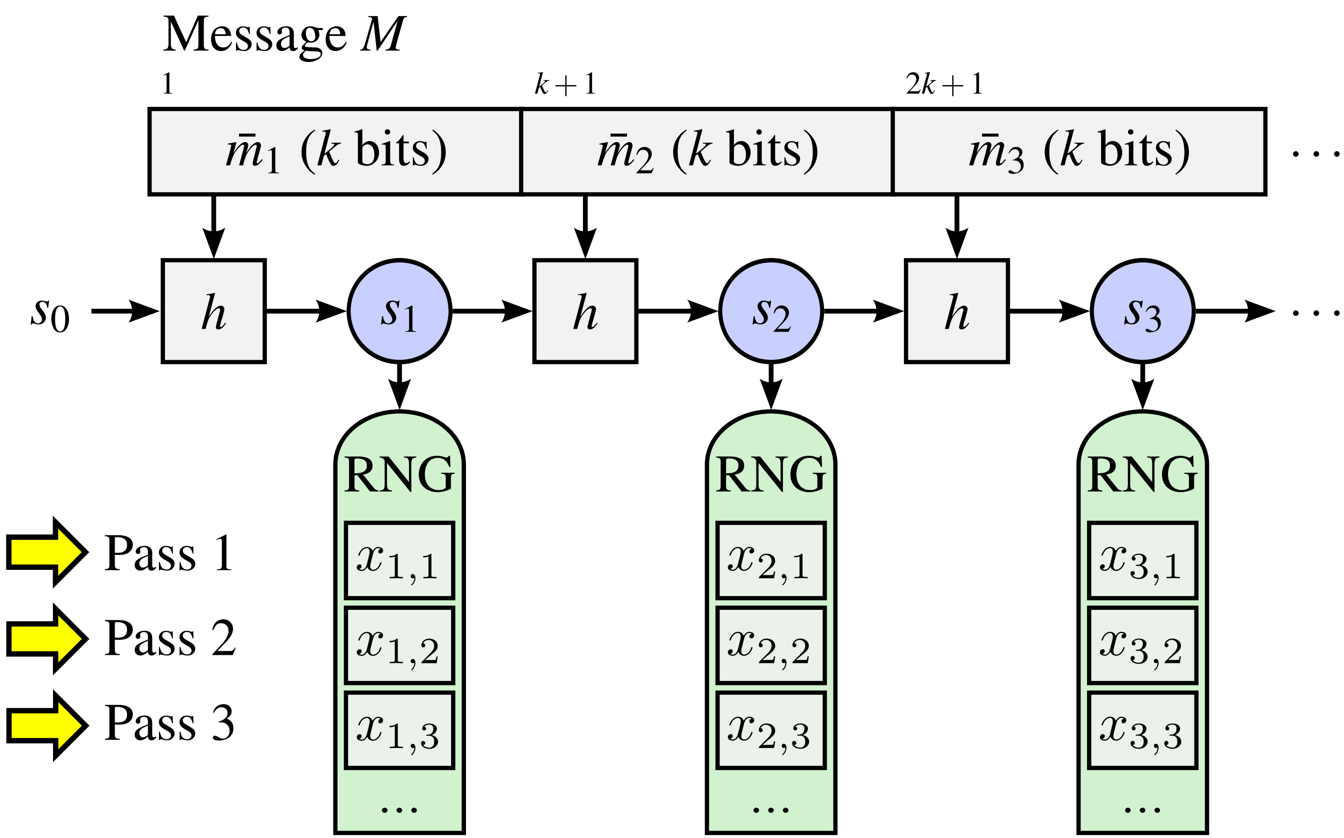


Decode by measuring distance

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



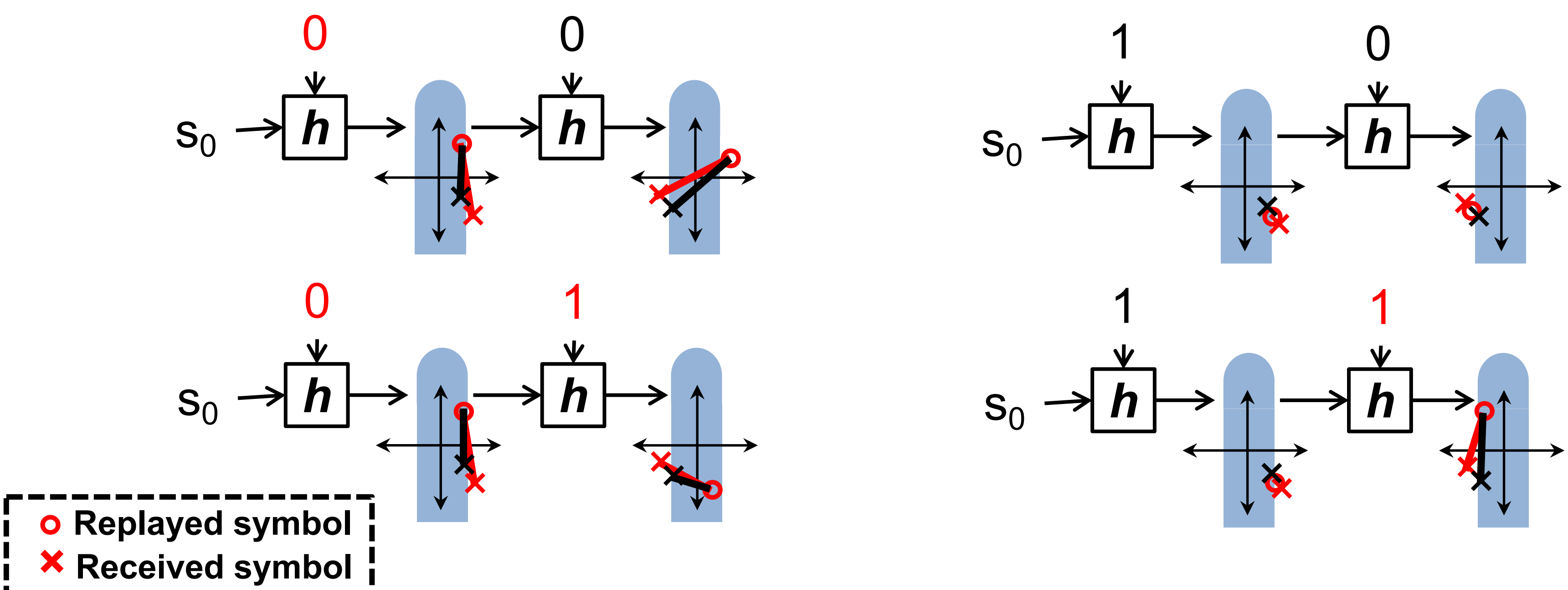
Adding additional passes



- **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 (○), **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 **y**
  - The vector of constellation points sent **if *M'* were the message,** $\mathbf{x}(M')$

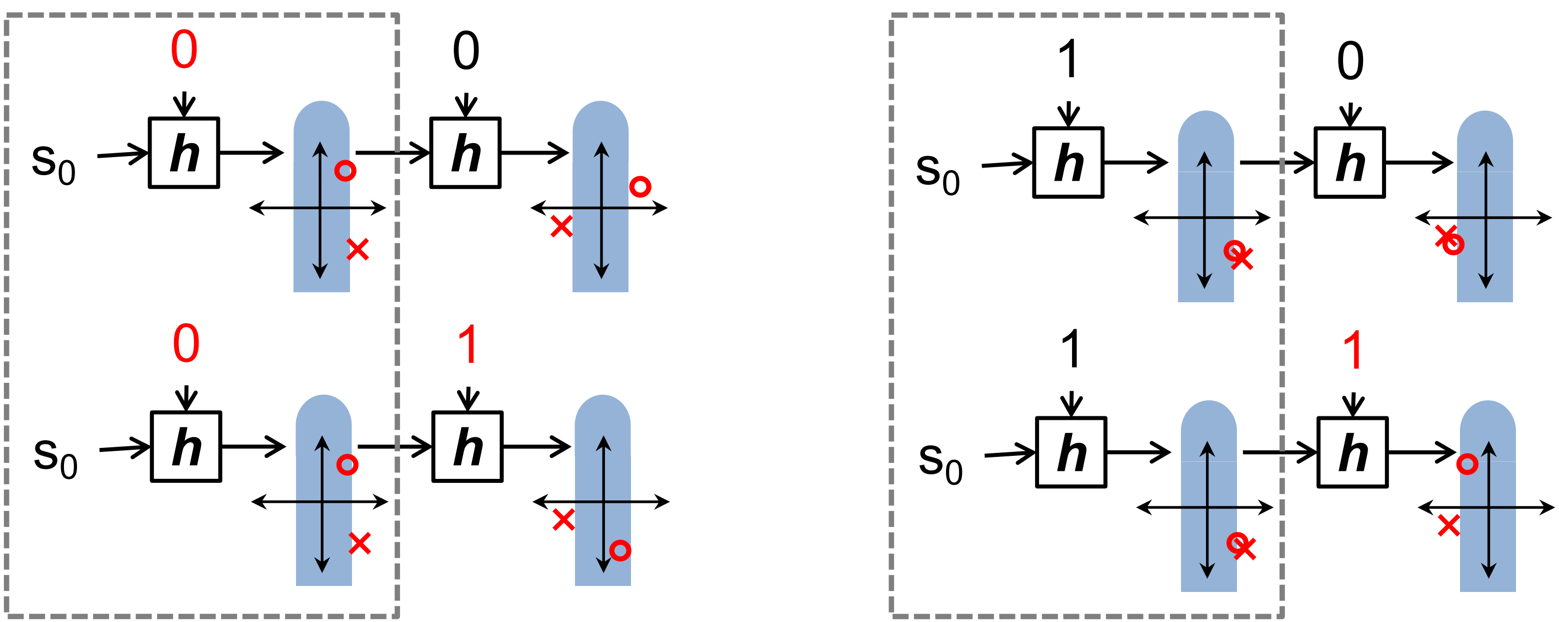
$$\hat{M} = \arg \min_{M' \in \{0,1\}^n} \left\| \mathbf{y} - \mathbf{x}\left(M'\right) \right\|^2$$

- 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

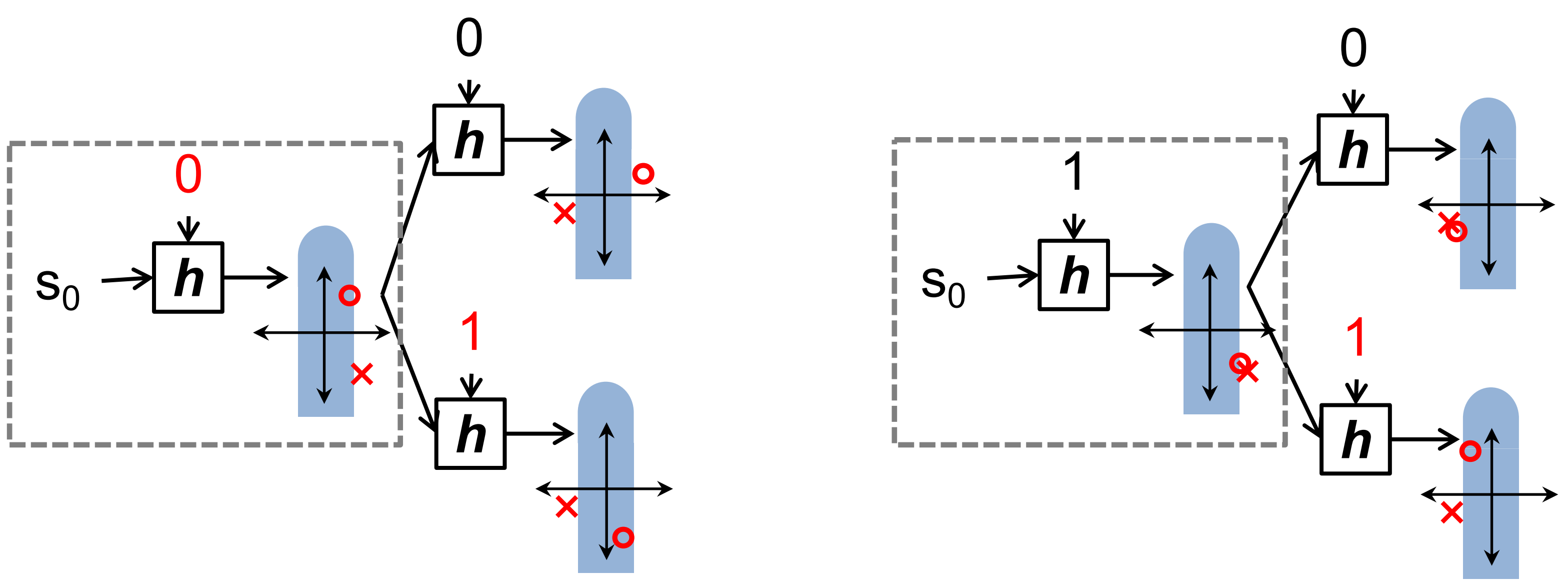
- Observe: Hypotheses whose initial stages share the same symbol guesses are **identical** in those stages



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:

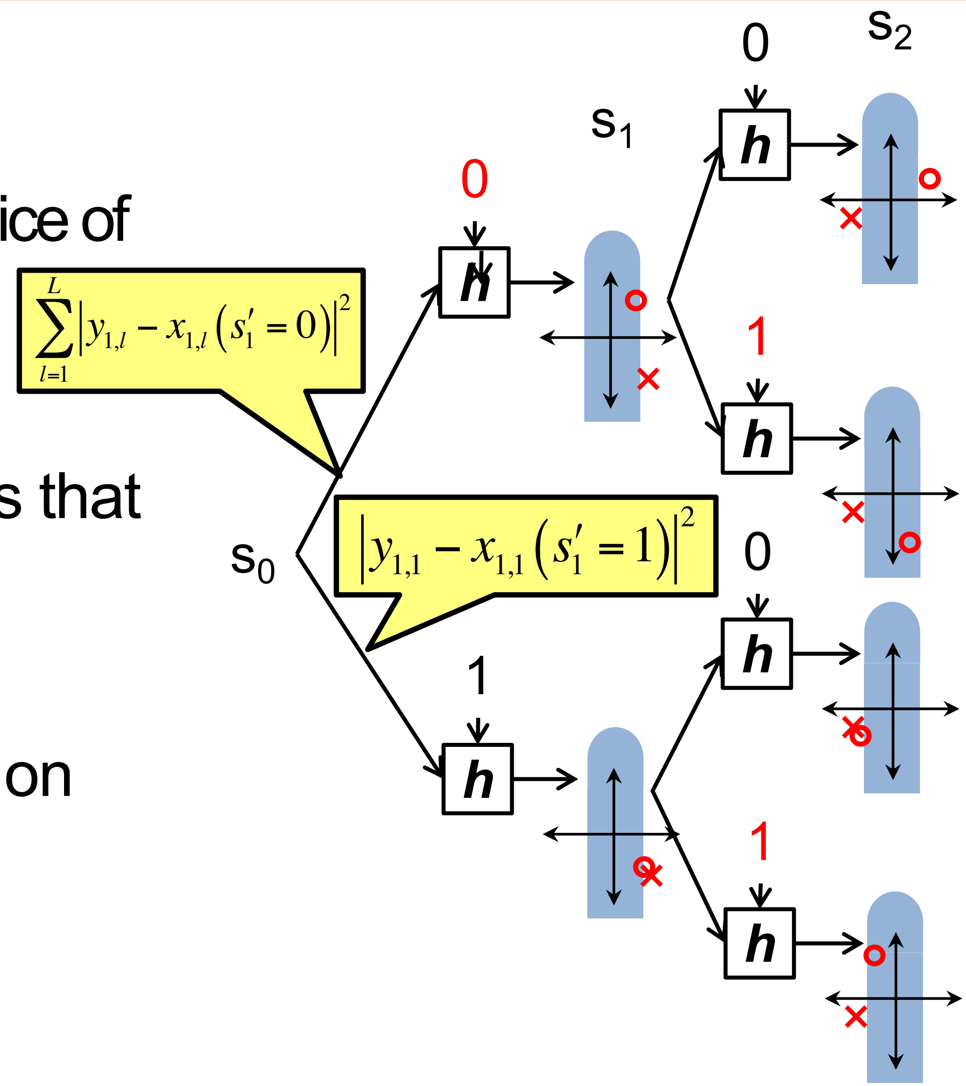


ML decoding over a tree

- General tree properties:
  - n/k levels, one per spine
  - Branching factor 2^k (per choice of k-bit message chunk)

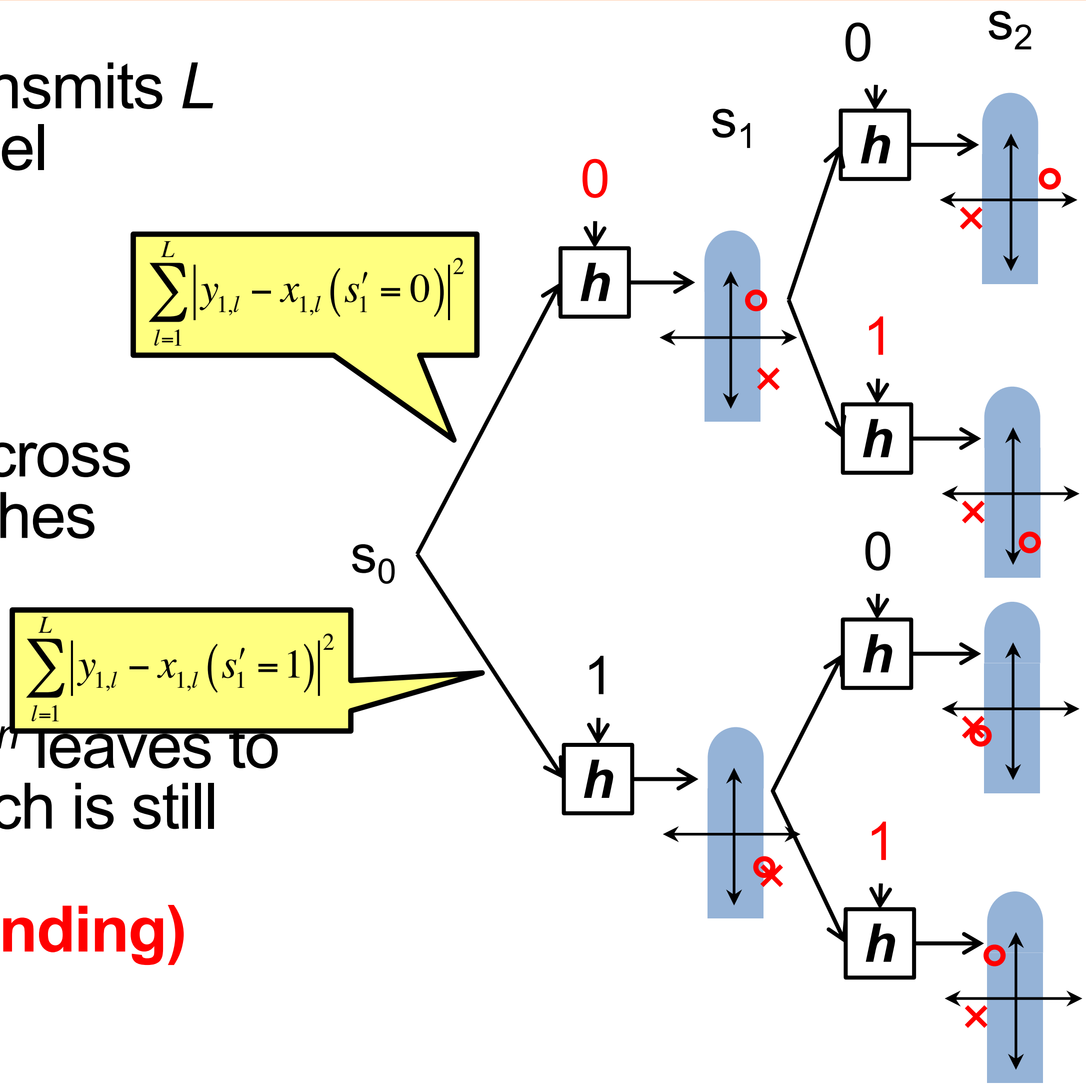
- Let s'_t be the t^{th} spine value associated with all messages that share s'_t

- 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

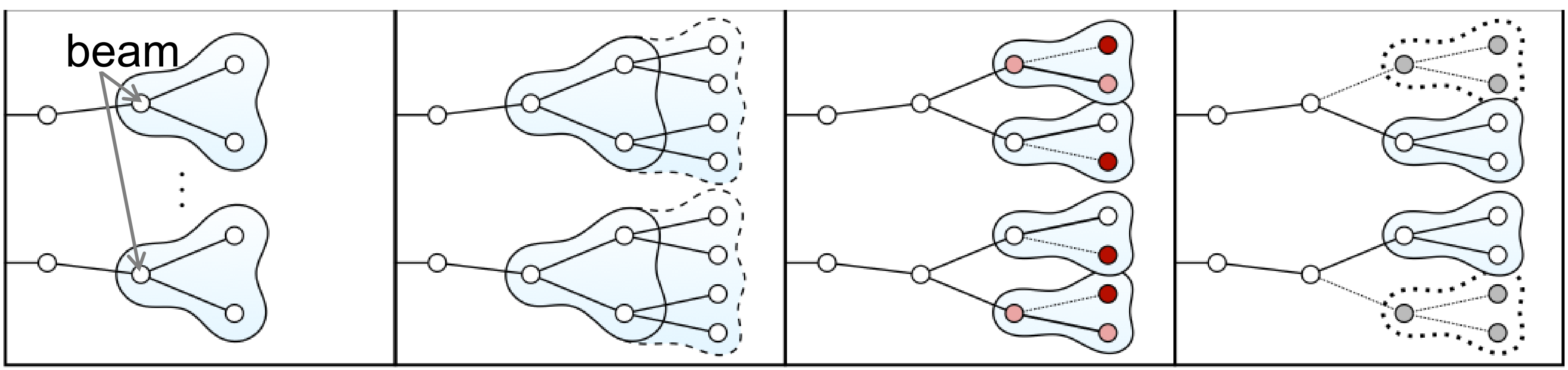
- Consider the **best 100 leaves** in the ML tree:
  - Tracing back through the tree, they will have a **common ancestor** with *M** in O(log *n*) steps
  - This suggests a strategy in which we only **keep a limited number of ancestors**

“Bubble” decoder

- Maintain a ***beam*** of ***B*** **tree node possibilities to explore at each stage,** 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

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*

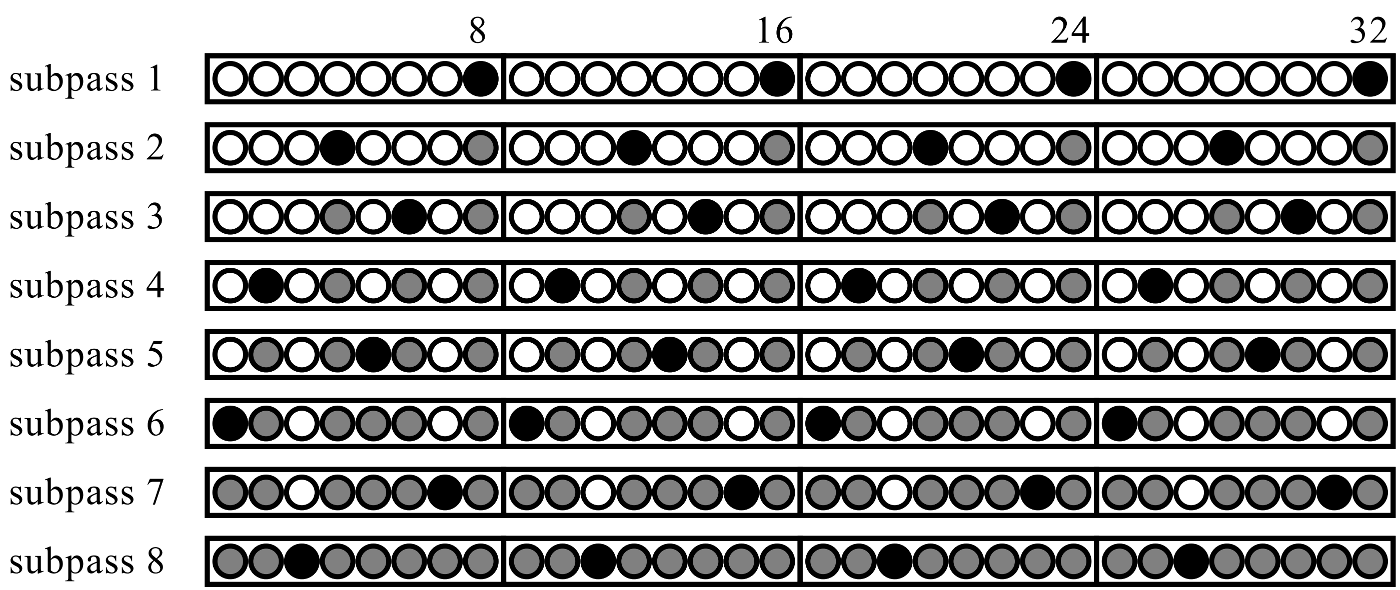
1. Sending *L* passes reduces rate to *k*/*L*—**abrupt drop**
   - Introduces plateaus in the rate versus SNR curve

Puncturing for higher and finer-grained 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. Bit Rate Adaptation

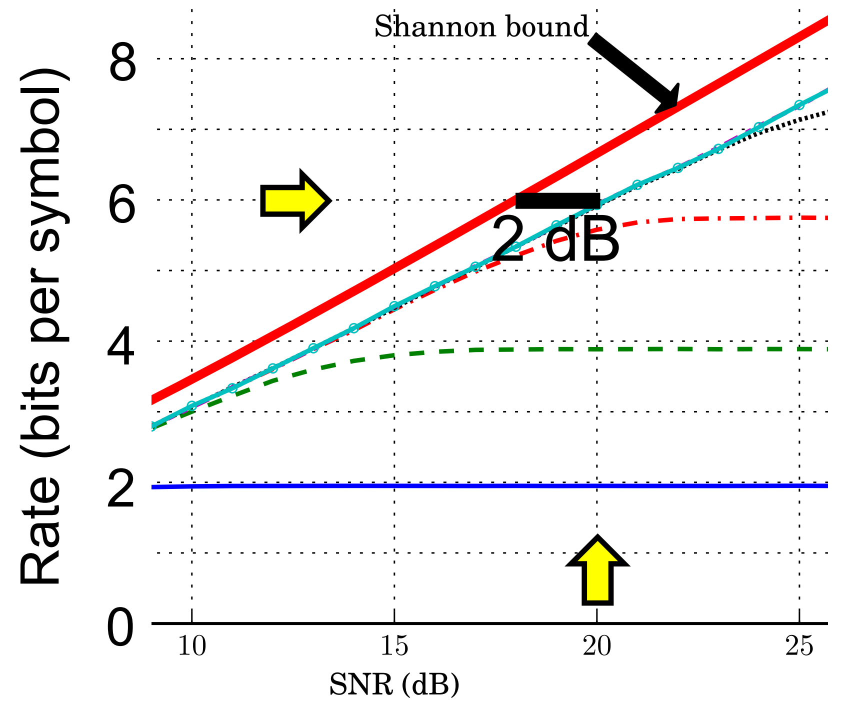
2. **Rateless Codes: Spinal Codes**
   - Encoder structure
   - Decoder structure
   - **Performance**

Methodology

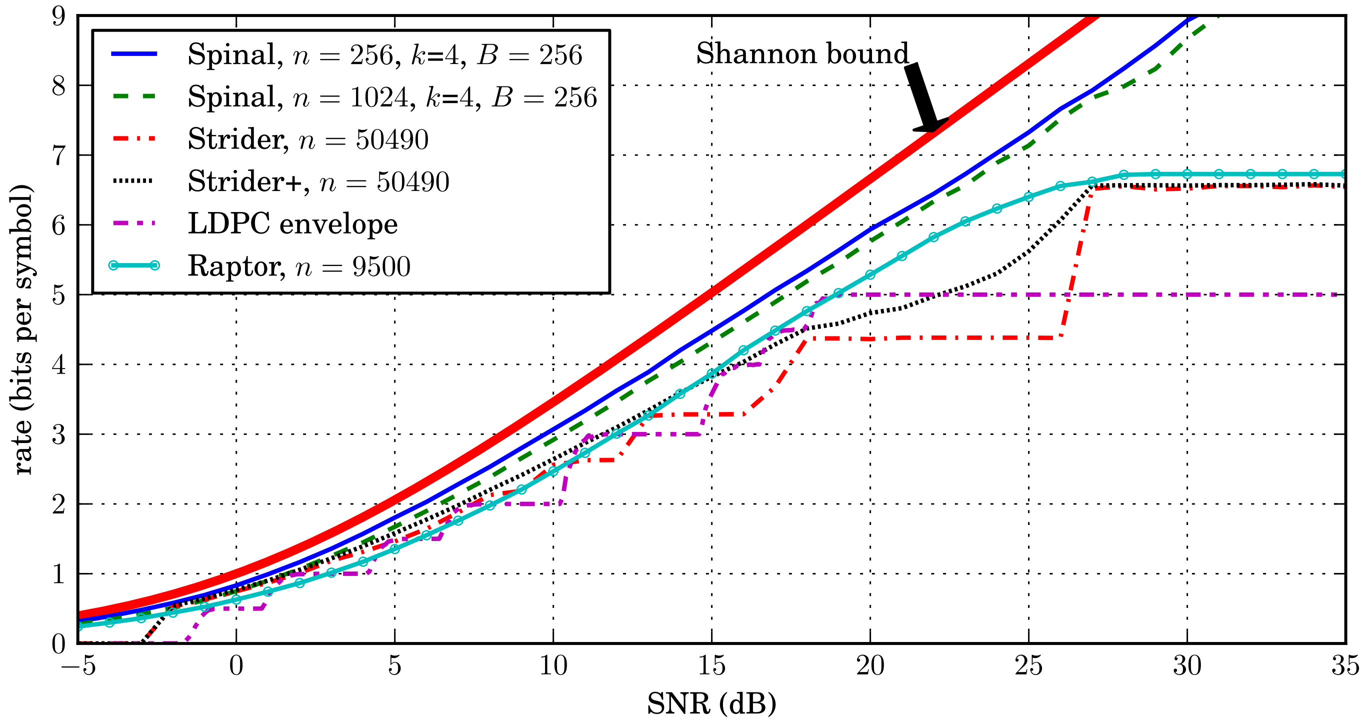
- Software simulation: Simulated wireless channel (additive white Gaussian noise and Rayleigh fading)

- 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**



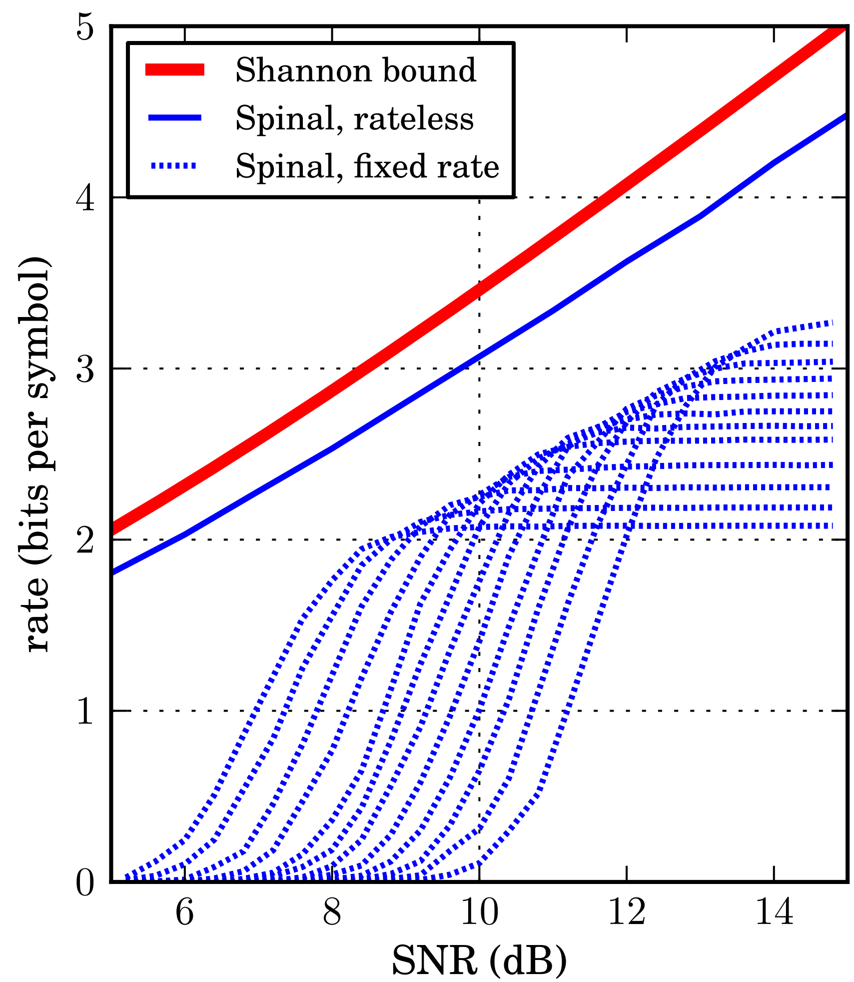
Spinal codes: Higher rate on AWGN channel



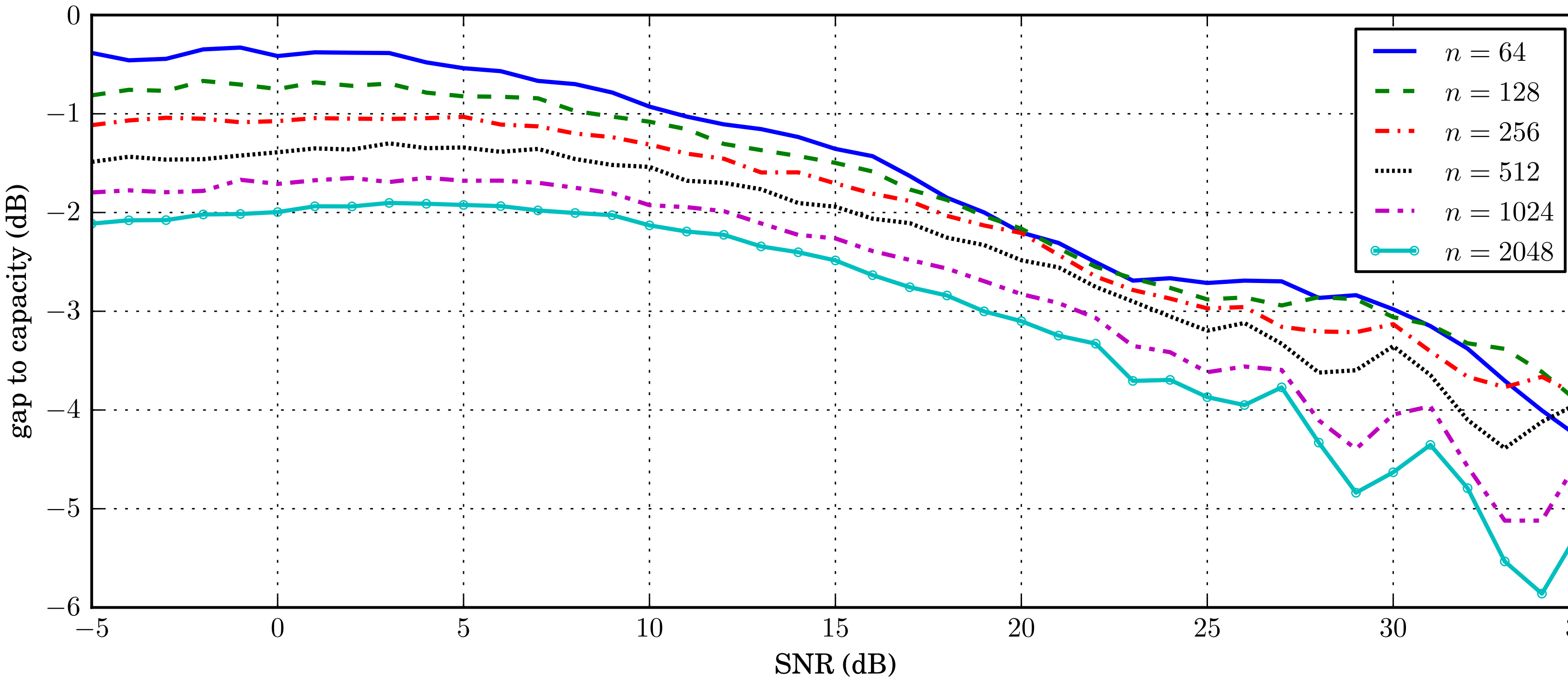
- Simulated AWGN channel: no link-layer performance effects here
- **LDPC envelope:** Choose best-performing rated LDPC code at each SNR to mimic the best a rate adaptation strategy could do
- **Strider+:** Strider + puncturing: finer rate control, but significant gap to capacity

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
- Exam is closed book, closed Internet, closed electronic devices; calculators not permitted

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

Next week’s precepts:
Lab 2

Tuesday, March 12, 11:00 AM:
In-Class Midterm

Next Thursday:
[Part III: Wireless from the PHY Upwards]
Signals and Systems Preliminaries