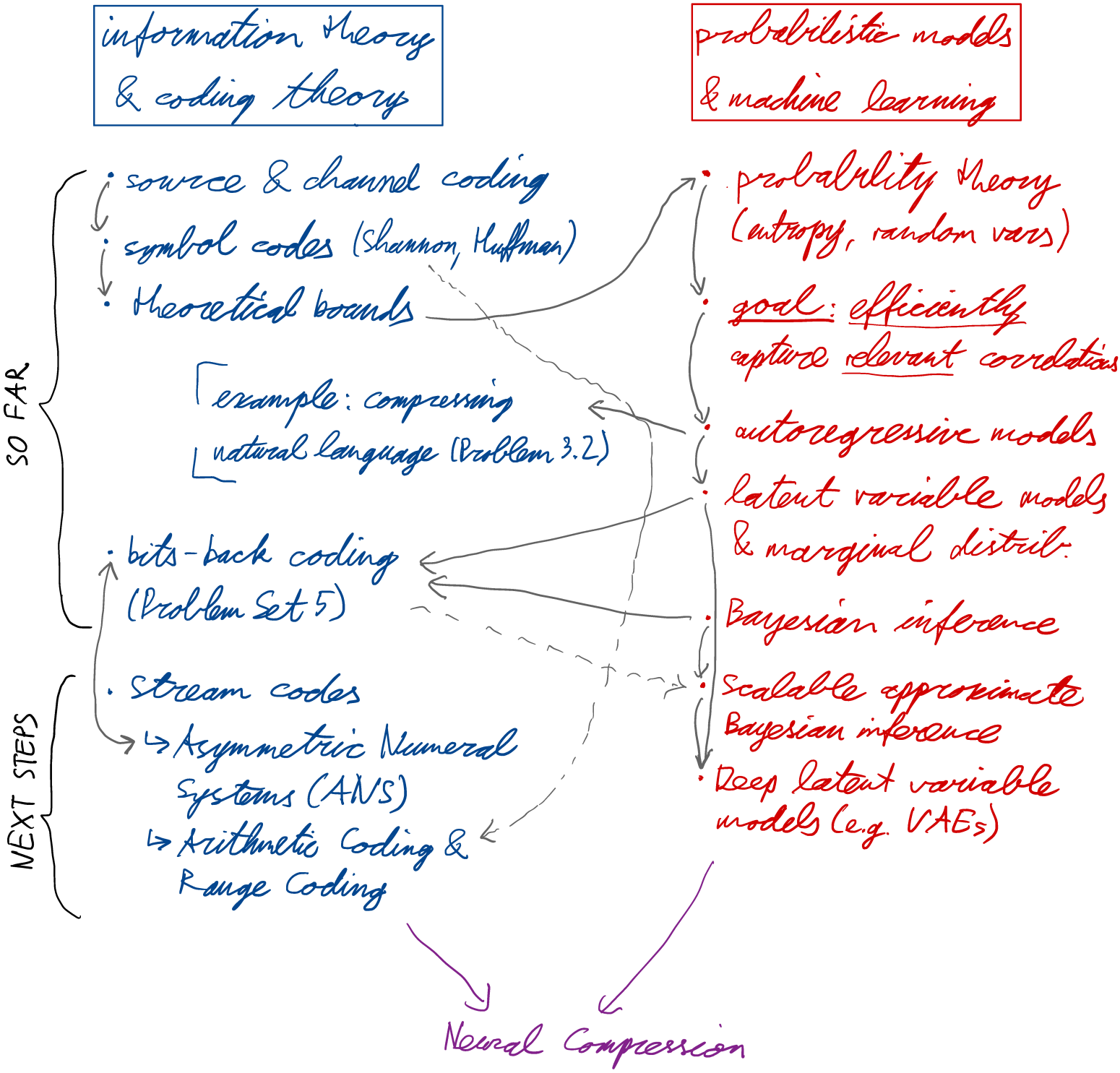
# Where Were We?



# Stream Codes

- Reminder: 2 pairs of theoret bounds for lossless comp.

(i) optimal symbol code:

$$H_p(X_i) \leq L_{opt} < H_p(X_i) + 1 \text{ bit } \underline{\text{per symbol}}$$

$L_{opt}$: expected code word length = expected bit rate per symbol

(ii) for any lossless compression:

$$H_p(\underline{X}) \leq \mathbb{E}_p[R_{opt}(\underline{X})] \leq H_p(\underline{X}) + 1 \text{ bit } \underline{\text{per message}}$$

$R_{opt}(\underline{X})$: bit rate of entire message $\underline{X}$ (with symbol/block codes)

→ Problem: satisfying upper bound is prohibitively expensive ☹

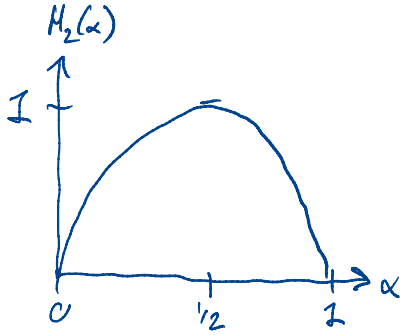
↳ in ML-based compression, $H_p(X_i)$ is often $\ll 1$ bit

⇒ symbol codes would have high overhead

Toy example: bent coin: $P(X_i = \text{"heads"}) = \alpha \quad (\alpha \in (0,1))$

$P(X_i = \text{"tails"}) = 1-\alpha$

$$\Rightarrow H_p(X_i) = -\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha) =: H_2(\alpha)$$




e.g. for $\alpha = 0.1$: $H_2(\alpha) \approx 0.47$ bits

→ symbol code:

> factor of 2 overhead

Goal: • alleviate overhead of symbol codes
• while maintaining $O(k)$ computational cost
} → stream codes

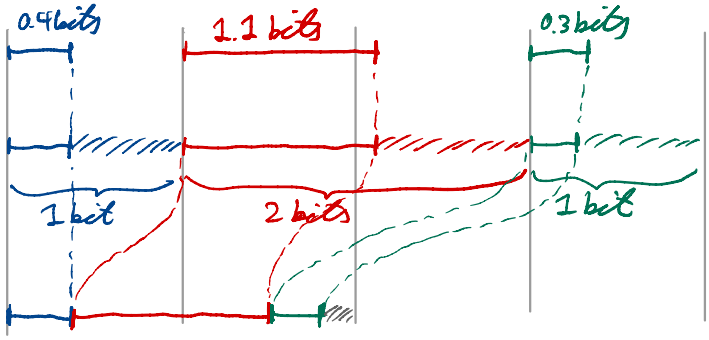
Strategy: amortize bit rate over symbols → # code words

Intuition: message: $\underline{x} = (x_1, x_2, x_3)$

inf. content:

symbol code: (Shannon coding)

stream codes




total: 1.8 bits

total: 4 bits

total: 2 bits

# symbol that corresponds to this bit
→ "amortization"

## Important Stream Codes

- Asymmetric Numeral Systems (ANS) (Duda et al., 2015)
  - stack ("last in first out")
  - → useful for bits-back coding with latent variable models
- Arithmetic Coding (Rissanen 1976; Pasco 1976)
  - ↳ variant: Range Coding
  - queue ("first in first out")
  - → useful for autoreg. model

# Asymmetric Numeral Systems (ANS)

→ generalizes positional numeral systems (e.g., decimal systems)

## Exercise 1: Decimal System

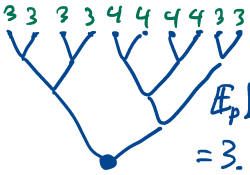
Consider the task of compressing a (variable-length) string of statistically independent and uniformly distributed symbols from the alphabet $\mathfrak{X} = \{0, \dots, 9\}$. Here, "uniformly distributed" means that all symbols from the alphabet occur with equal probability.

$H_p(X_i) = \mathbb{E}_p[-\log_2 P(X_i)] = \mathbb{E}_p[-\log_2 \frac{1}{10}] = \log_2(10) \approx 3.32$ bits

apply what you've learned in this course

(a) What is the entropy per symbol?

(b) If you were to compress this sequence of symbols with an *optimal symbol code*, what would be the expected code word length (i.e., the expected bit rate per symbol)?




(c) Can you do better than an optimal symbol code? Describe your approach first in words, then implement it in Python or in pseudo code (about 4 lines of code for encoding and 4 lines of code for decoding).

Forget everything you've learned in this course so far

*Hint:* you may assume that integers in your programming language can be of arbitrary size (as is the case in Python), that they support basic arithmetic operations (addition, subtraction, multiplication, and integer division with remainder) and that they are represented internally as bit strings. In Python 3, integer division is notated as double-slash "//".

1 3 2 7 5 6 → binary
1 0 3 2 7 5 6 → binary

(d) What is the expected bit rate per symbol of your method from part (c), in the limit long messages? Compare your result to the results from parts (a) and (b).

## Exercise 2: Ternary System

All parts (a)-(d) exactly as in Exercise 1 above, but this time the alphabet is only of size three. Thus, for each $i \in \{1, \dots, k\}$, you have a symbol $X_i \in \{0, 1, 2\}$ with probabilities $P(X_i = 0) = P(X_i = 1) = P(X_i = 2) = \frac{1}{3}$.

(d) upper & lower bound on $R(\underline{x})$

$$R(\underline{x}) \geq \log_2(10^k) + \underbrace{\varepsilon}_{<1} = k \log_2(10) + \varepsilon$$

$$\Rightarrow \frac{R(\underline{x})}{k} \geq \log_2(10) + \underbrace{\frac{\varepsilon}{k}}_{\to 0} \xrightarrow{k \to \infty} \log_2(10) = H_p(X_i)$$

$$R(\underline{x}) \leq \log_2(2 \times 10^k - 1) + \varepsilon < \log_2(2 \times 10^k) + \varepsilon$$
$$= 1 + k \log_2(10) + \varepsilon$$

$$\Rightarrow \frac{R(\underline{x})}{k} \leq \log_2(10) + \frac{1+\varepsilon}{k} \xrightarrow{k \to \infty} \log_2(10) = H_p(X_i)$$

$$\Rightarrow \frac{R(\underline{x})}{k} \xrightarrow{k \to \infty} H_p(X_i) \Rightarrow \text{optimal}$$

Observations: Conversion between different positional numeral systems

↳ operates as a stack

↳ amortizes compressed bits over symbols

↳ optimally compresses a sequence of symbols if these symbols

(i) all are from same alphabet

(ii) are uniformly distributed over their alphabet

(iii) (therefore) are stat. independent

→ Let's remove these constraints !

Constraint (i): → mixing symbols with different alphabets just works ☺

Constraint (ii): consider non-uniform $P(x_i)$

↳ use bits-back trick

↳ approximate $P$ by $P_{ANS}$, which represents all probabilities in fixed-point arithmetic

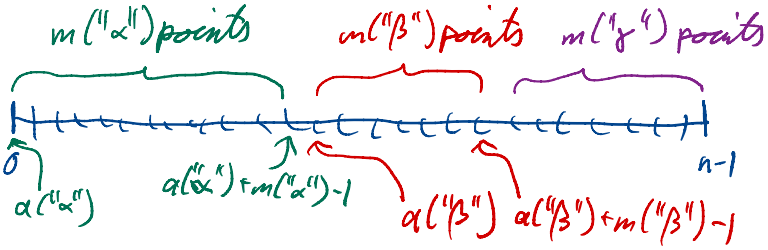
→ parameter $n \in \mathbb{N}$ (typically a power of 2)

$$P_{ANS}(X_i = x_i) = \frac{m(x_i)}{n}$$ ← integer

→ partitions the range $\{0, 1, \dots, n-1\}$ into disjoint subranges for each $x_i \in \mathcal{X}$.

$$\{a(x_i), \dots, a(x_i) + m(x_i) - 1\}$$

e.g.: $\mathcal{X} = \{\text{"}\alpha\text{"}, \text{"}\beta\text{"}, \text{"}\gamma\text{"}\}$




↳ interpret $P_{ANS}(X_i)$ as the marginal dist. of the latent variable model

$$P_{ANS}(X_i, Z_i) = \underbrace{P_{ANS}(Z_i)}_{=\frac{1}{n}} \; \underbrace{P_{ANS}(X_i \mid Z_i)}_{=1 \text{ for exactly } m(x_i) \text{ cases}}$$

where $z_i \in \{0, 1, \dots, n-1\}$

with uniform prior, i.e., $P_{ANS}(Z_i = z_i) = \frac{1}{n} \; \forall z$

and likelihood

$$P(X_i = x_i \mid Z_i = z_i) = \begin{cases} 1 \text{ if } z_i \in \{a(x_i), \dots, a(x_i) + m(x_i) - 1\} \\ 0 \text{ else} \end{cases}$$

Claim: $P_{ANS}(X_i = x_i) = \sum_{z=0}^{n-1} P_{ANS}(X_i = x_i, Z_i = z_i)$

$P_{ANS}(X_i = x_i) = \frac{m(x_i)}{n}$

⇒ Apply bits-back coding:

1) find posterior distrib.

$$P(Z_i = z_i \mid X_i = x_i) = \frac{\overbrace{P(Z_i = z_i)}^{=1/n} \overbrace{P(X_i = x_i \mid Z_i = z_i)}^{\in\{0,1\}}}{\underbrace{P(X_i = x_i)}_{= \frac{m(x_i)}{n}}}$$

$$= \begin{cases} \frac{1}{m(x_i)} \text{ if } z_i \in \{a(x_i), \dots, a(x_i) + m(x_i) - 1\} \\ 0 \quad \text{else} \end{cases}$$

Thus, both prior & posterior are uniform distributions, just with different alphabet sizes.

⇒ Bits-back coding:

↳ encode($x_i$):

- decode $z_i$ using $P(z|x_i=x_i)$
  uniform over $\{a(x_i), \dots, a(x_i)+m(x_i)-1\}$
- ~~encode $x_i$ using $P(x_i|z_i=z_i)$~~
  $=1$ for $z \in \{a(x_i), \dots, a(x_i)+m(x_i)-1\}$
  → inf content $=0$
- encode $z_i$ using $P(z_i)$
  uniform over $\{0, \dots, n-1\}$

⇒ net bit rate for 1 symbol

$-\log_2 m(x_i) + \log_2 n = -\log_2 \frac{m(x_i)}{n}$

$= -\log_2 P_{ANS}(x_i = x_i)$ ✓

↳ decoder:

- decode $z_i$ using $P(z_i)$ → identify $x_i$
- encode $z_i$ using $P(z_i|x_i=x_i)$

Constraint (iii): statistical independence of encoded symbols

→ now that we can encode each symbol with its individual probabilistic model (see experiment in video recording) we can model correlations using either a latent variable model (with another layer of bits-back coding) or an autoregressive model (as we've done before).

# (Naive) implementation in Python:

```python
class AnsCoder:
    def __init__(self, precision):
        self.precision = precision
        self.mask = (1 << precision) - 1
        self.compressed = 1

    def encode(self, symbol, scaled_probabilities):
        z = self.compressed % scaled_probabilities[symbol]
        self.compressed //= scaled_probabilities[symbol]
        for prob in scaled_probabilities[:symbol]:
            z += prob
        self.compressed = (self.compressed << self.precision) + z

    def decode(self, scaled_probabilities):
        z = self.compressed & self.mask
        self.compressed >>= self.precision
        for i, prob in enumerate(scaled_probabilities):
            if prob > z:
                symbol = i
                break
            else:
                z -= prob

        self.compressed = (
            self.compressed * scaled_probabilities[symbol] + z)
        return symbol
```

O(k)

Those parts are just for demonstration purpose. In a production setup, you should do something more efficient here (e.g., binary search, lookup tables, ...)

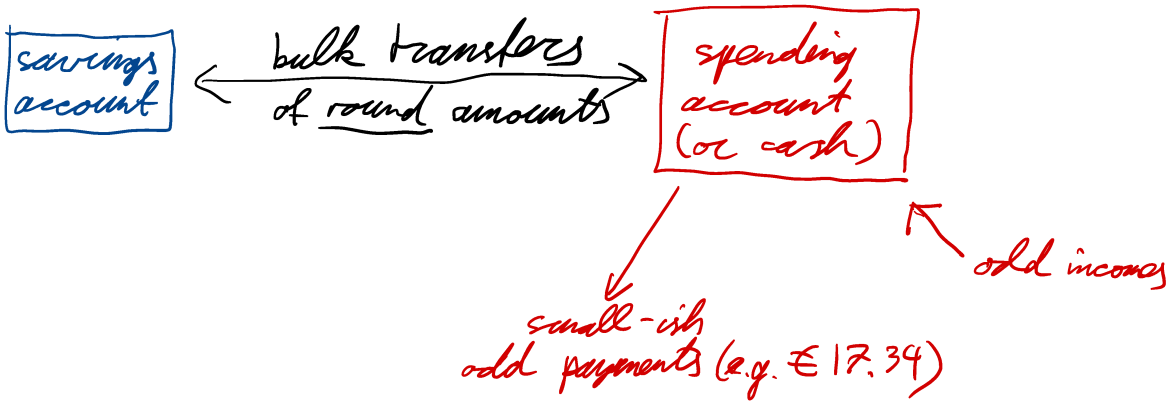
## Reminder: Goals

- reduce compression overhead compared to symbol codes ✓
- maintain O(k) run-time cost
  → not yet achieved!
  we have $O(k^2)$

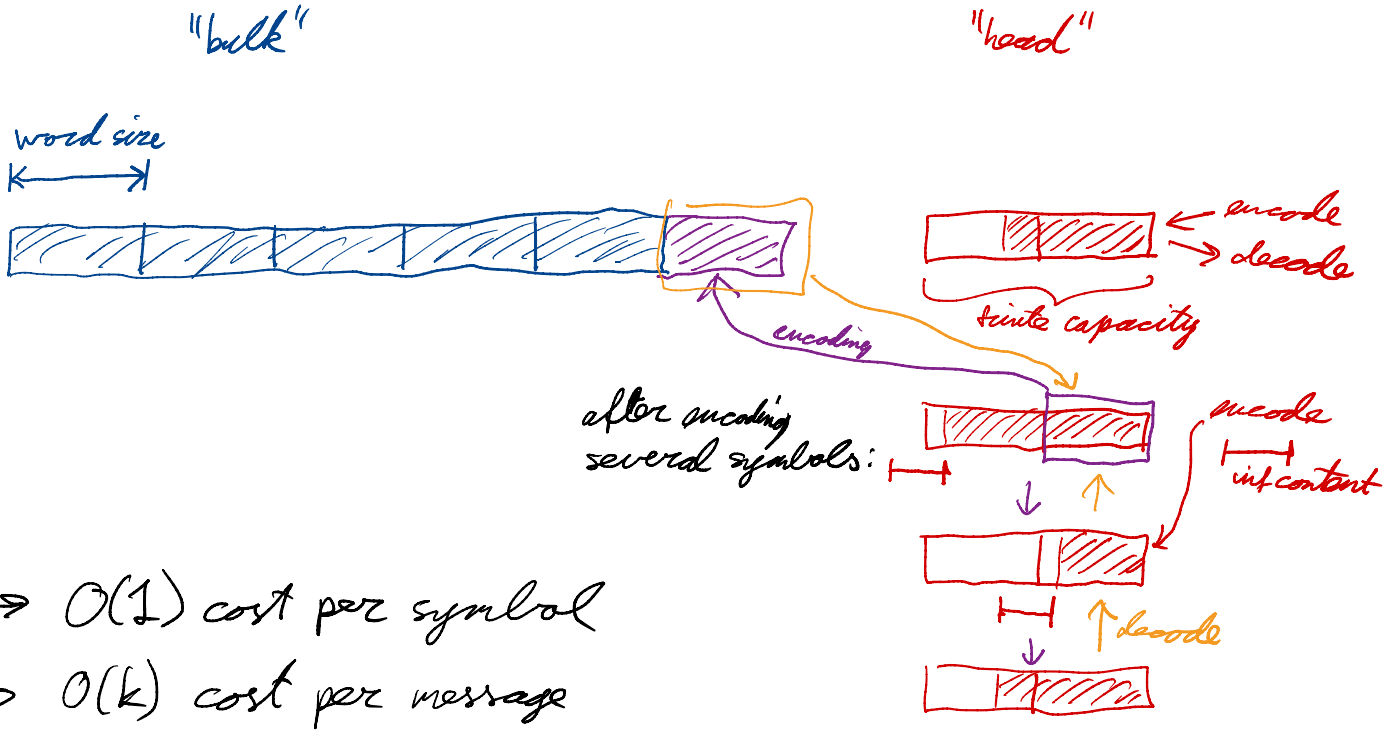


## Improving Run-Time Cost of ANS (to O(k))

→ metaphor: money



→ similar for (streaming) ANS:




→ O(1) cost per symbol

→ O(k) cost per message

→ Implement on Problem Set 7

On Problem Set 6: Optimality of positional numeral systems coder