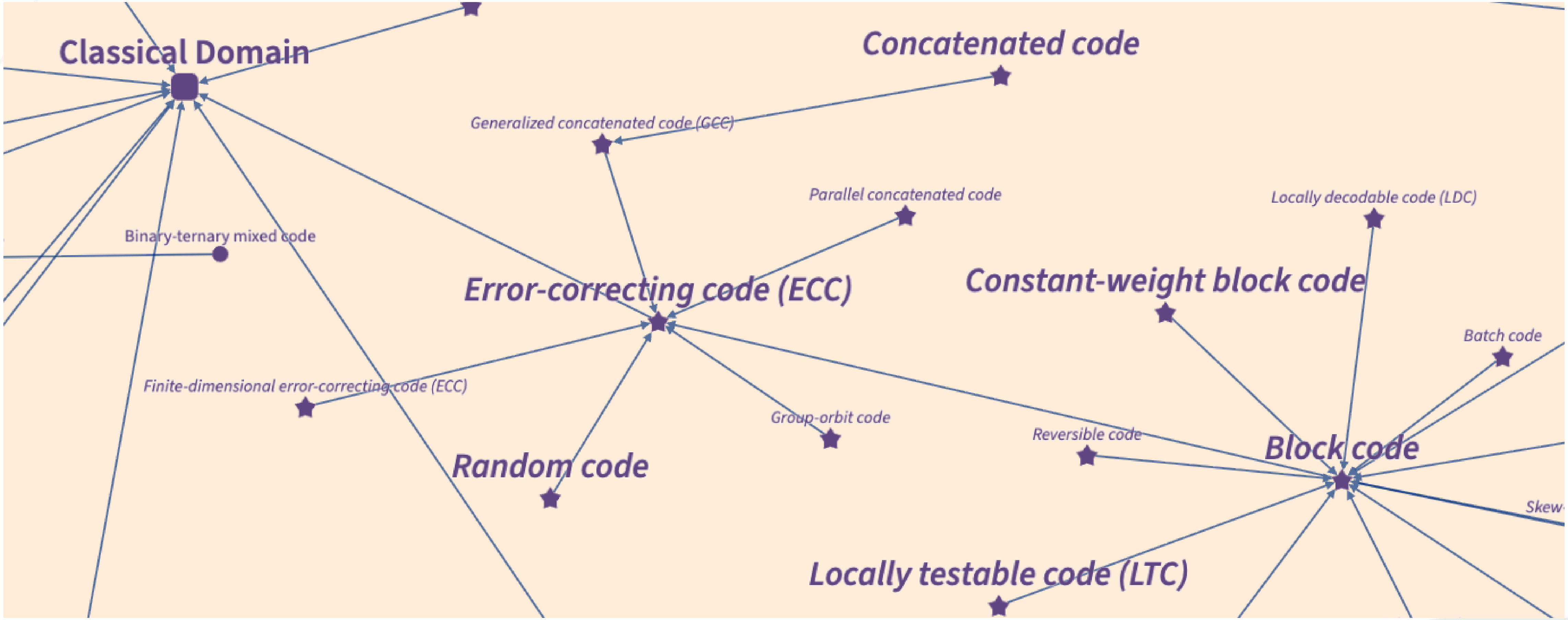
Erasure Codes: LT and Raptor Codes

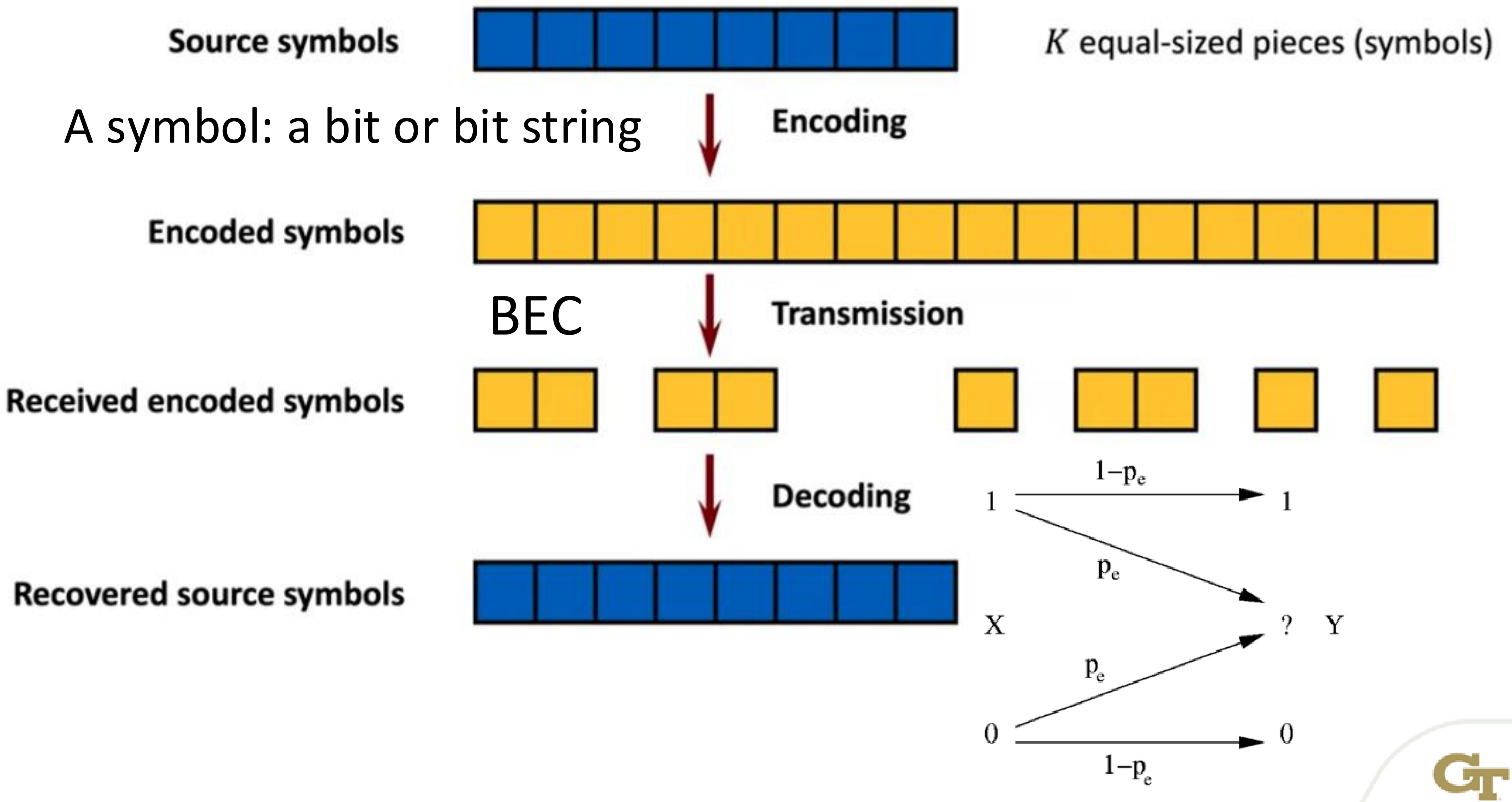
Georgia Tech

Error-correcting codes (ECC)

Main idea: add redundancy to detect and correct errors

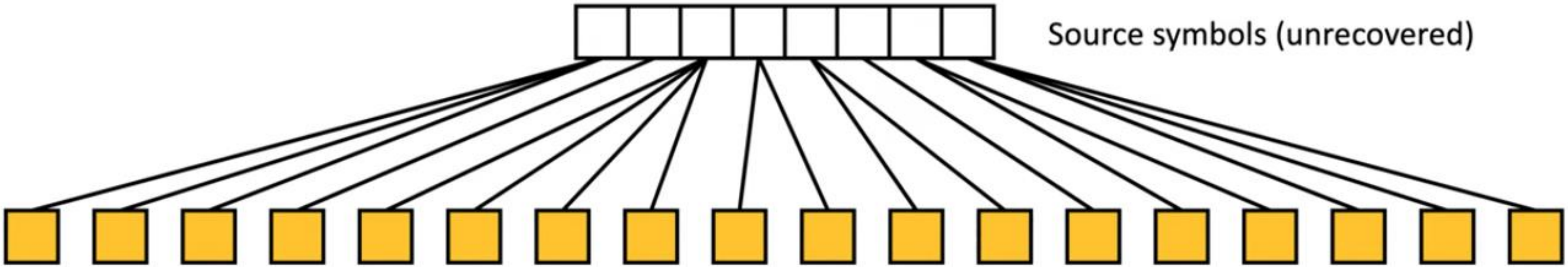


Erasure Codes: ECC under Binary Erasure Channel (BEC)



Copy Encoding

Collect encoded symbols and set up graph between encoded symbols and source symbols to be recovered



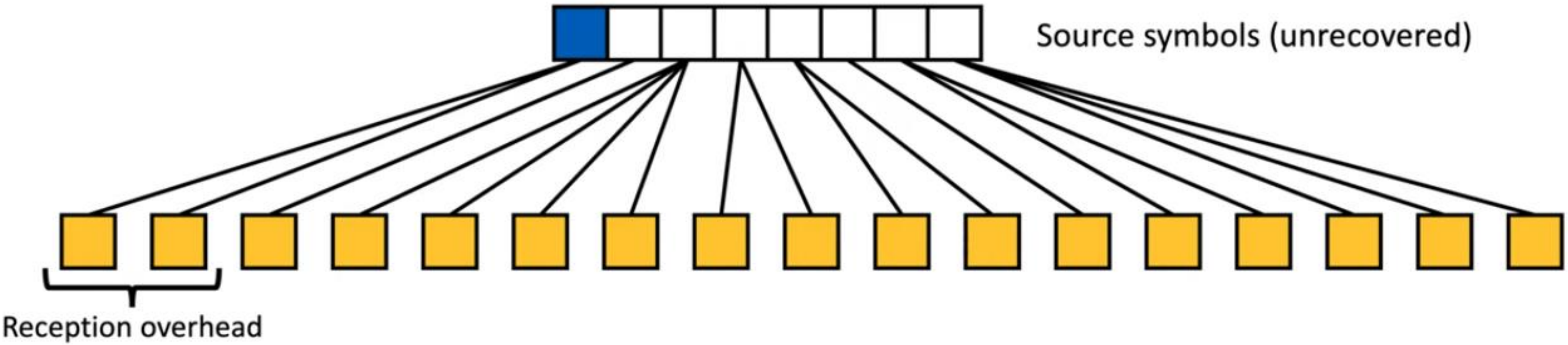
While there is an encoded symbol of degree 1, **process encoded symbol**:

- Source symbol neighbor = encoded symbol (source symbol neighbor is recovered)
- Encoded symbol neighbors of source symbol neighbor are useless (reception overhead)

If all source symbols are recovered then decoding is successful else decoding is unsuccessful

Copy Decoding

Collect encoded symbols and set up graph between encoded symbols and source symbols to be recovered



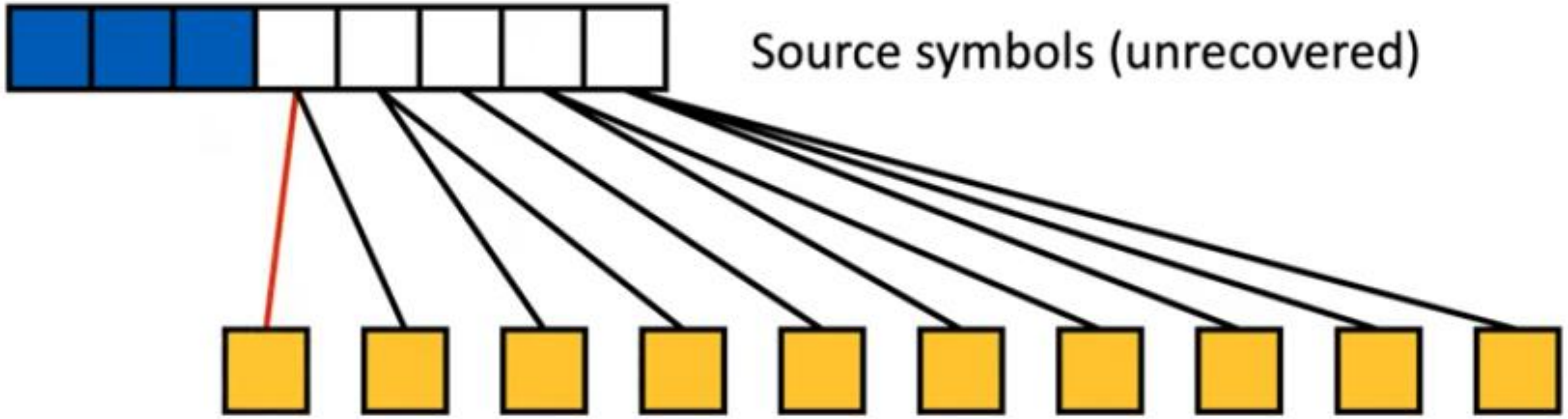
While there is an encoded symbol of degree 1, **process encoded symbol**:

- Source symbol neighbor = encoded symbol (source symbol neighbor is recovered)
- Encoded symbol neighbors of source symbol neighbor are useless (reception overhead)

If all source symbols are recovered then decoding is successful else decoding is unsuccessful

Copy Decoding

Collect encoded symbols and set up graph between encoded symbols and source symbols to be recovered



While there is an encoded symbol of degree 1, **process encoded symbol**:

- Source symbol neighbor = encoded symbol (source symbol neighbor is recovered)
- Encoded symbol neighbors of source symbol neighbor are useless (reception overhead)

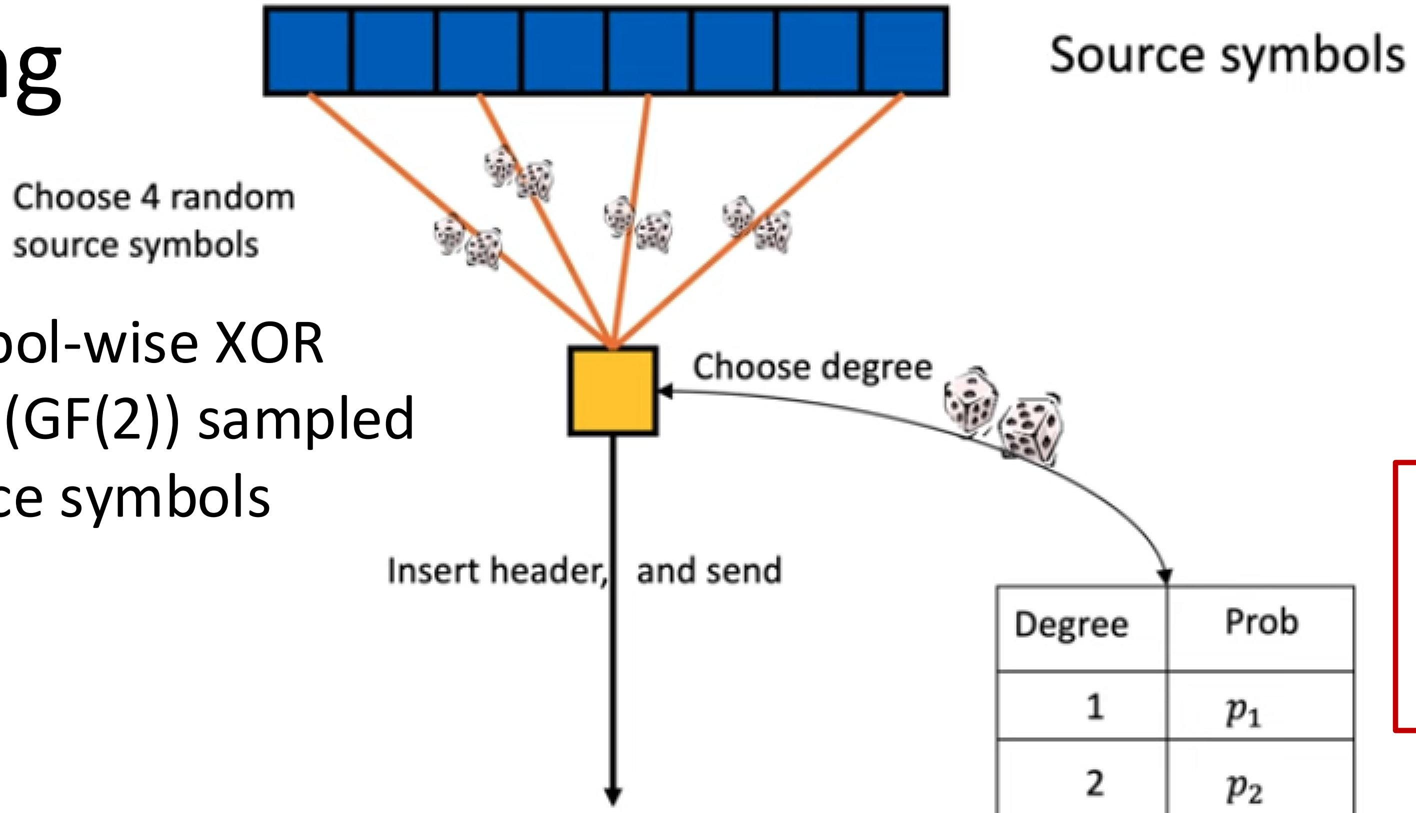
If all source symbols are recovered then decoding is successful else decoding is unsuccessful

Copy decoding analysis (coupon collector)

- "collect all coupons and win" contests
- It asks the following question: if each box of a given product contains a coupon, and there are K different types of coupons, what is the probability that more than N boxes need to be bought to collect all K coupons?
- The probability of collecting the i-th new coupon is
  - o pi=(K−i+1)/K
  - o # of draws until the first success in repeated independent trials follows a geometric distribution

- Expectation of N: Exp(reception overhead) + K = KlnK
- Decoding complexity: # edges KlnK

LT Encoding



Degree	Prob
1	p_1
2	p_2
3	p_3
4	p_4
⋮	⋮

Symbol-wise XOR over (GF(2)) sampled source symbols

Degree Distribution

$$\sum p_i = 1$$

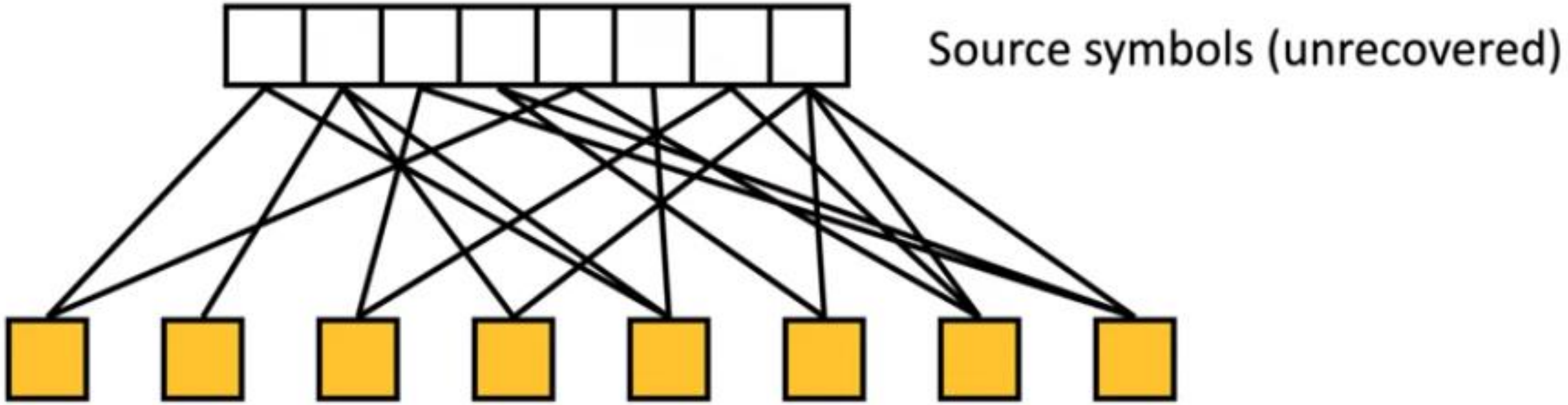
Rateless codes (fountain codes): Generating an infinite stream of encoded symbols

M. Luby, "LT codes," *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, Vancouver, BC, Canada, 2002, pp. 271-280

Georgia Tech

LT Decoding

Collect encoded symbols and set up graph between encoded symbols and source symbols to be recovered



While there is an encoded symbol of degree 1, **process encoded symbol**:

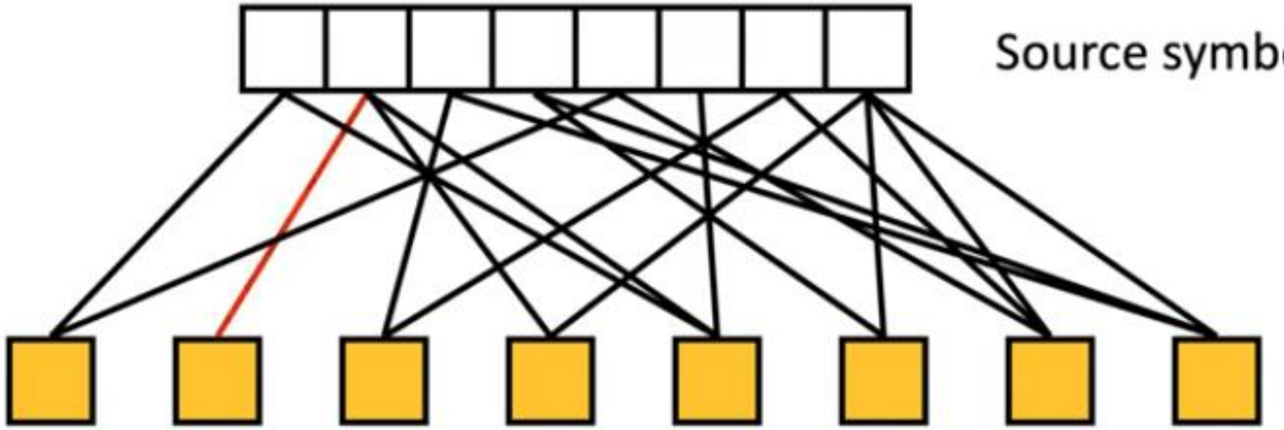
- Source symbol neighbor = encoded symbol (source symbol neighbor is recovered)
- XOR source symbol neighbor into all encoded symbol neighbors (reduces their degree by 1)

If all source symbols are recovered then decoding is successful else decoding is unsuccessful

M. Luby, "LT codes," *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, Vancouver, BC, Canada, 2002, pp. 271-280

LT Decoding

Collect encoded symbols and set up graph between encoded symbols and source symbols to be recovered



While there is an encoded symbol of degree 1, **process encoded symbol**:

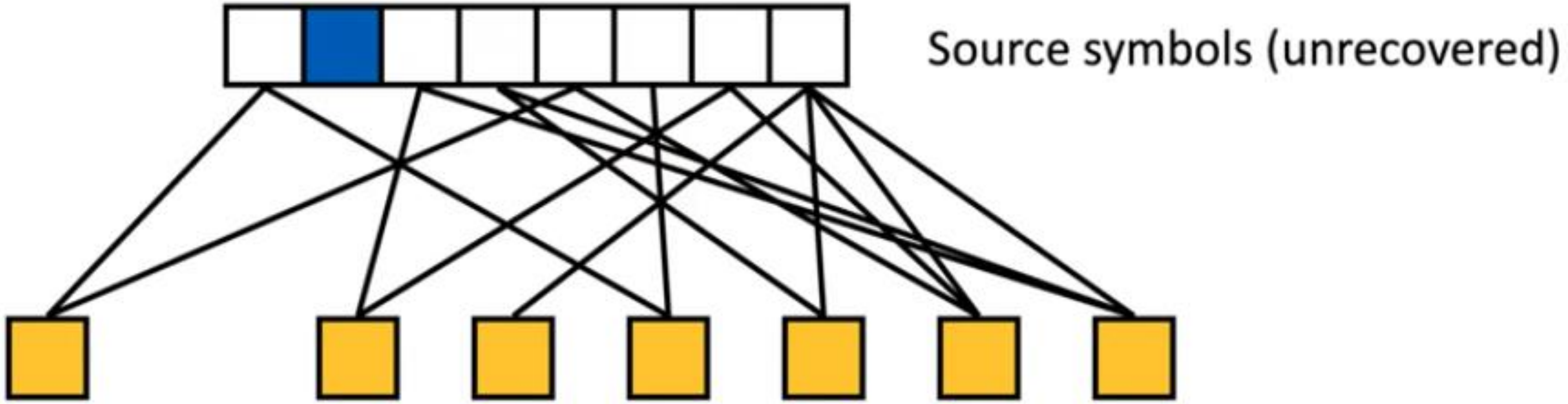
- Source symbol neighbor = encoded symbol (source symbol neighbor is recovered)
- XOR source symbol neighbor into all encoded symbol neighbors (reduces their degree by 1)

If all source symbols are recovered then decoding is successful else decoding is unsuccessful

M. Luby, "LT codes," *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, Vancouver, BC, Canada, 2002, pp. 271-280

LT Decoding

Collect encoded symbols and set up graph between encoded symbols and source symbols to be recovered



While there is an encoded symbol of degree 1, **process encoded symbol**:

- Source symbol neighbor = encoded symbol (source symbol neighbor is recovered)
- XOR source symbol neighbor into all encoded symbol neighbors (reduces their degree by 1)

If all source symbols are recovered then decoding is successful else decoding is unsuccessful

M. Luby, "LT codes," *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, Vancouver, BC, Canada, 2002, pp. 271-280

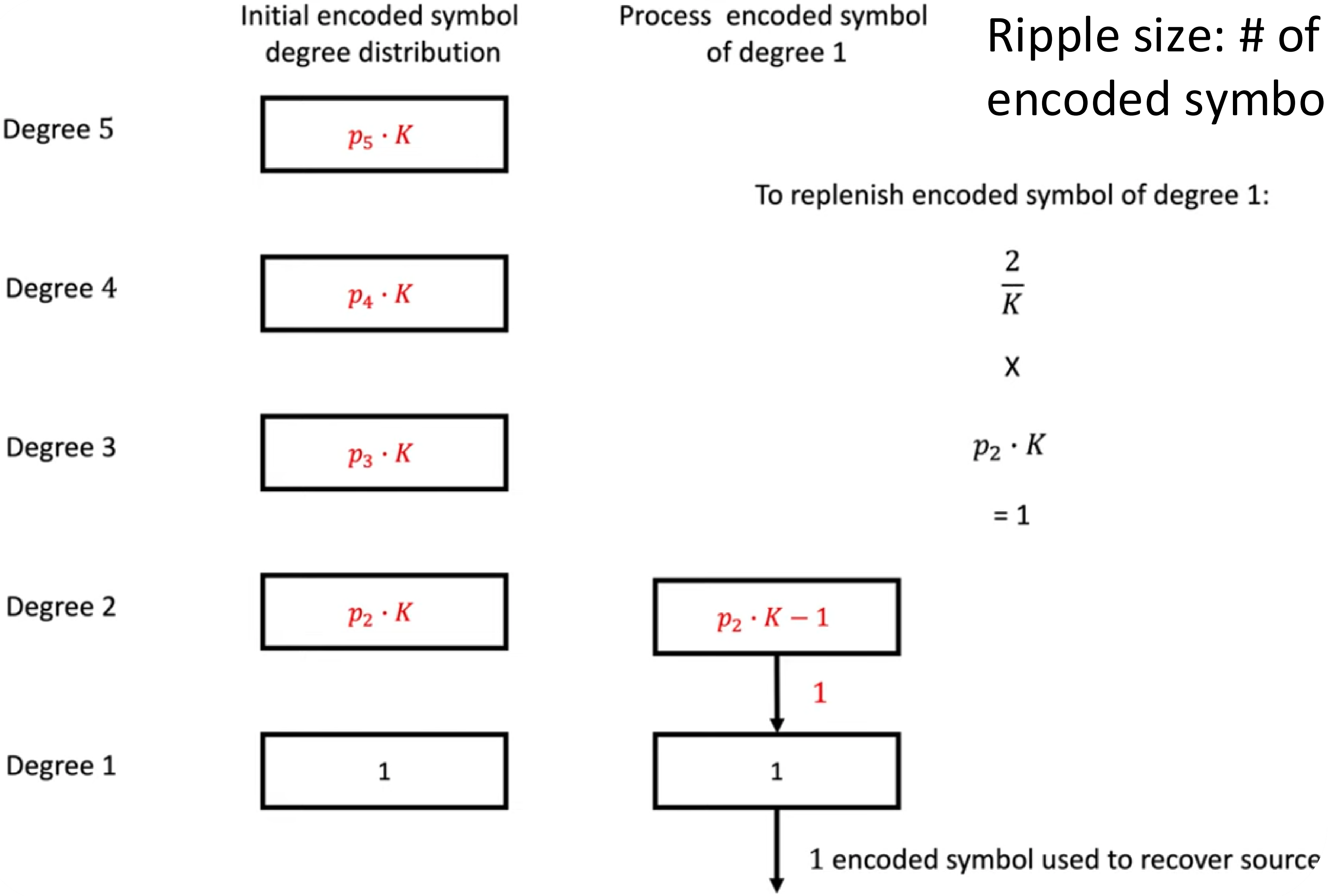
And-or Tree Analysis (Density Evolution)

- Goal: to analyze the asymptotic behavior of iterative peeling decoding
- For fixed iteration t, how many source and encoded symbols survive (density)?
- Randomly choose one edge from the graph to start
- This iterative process ≈ a tree recursion of “and–or” type:
- Source symbols act like OR (recover if any neighbor is known),
- Encoded symbols act like AND (known only if all its contributing sources are resolved).

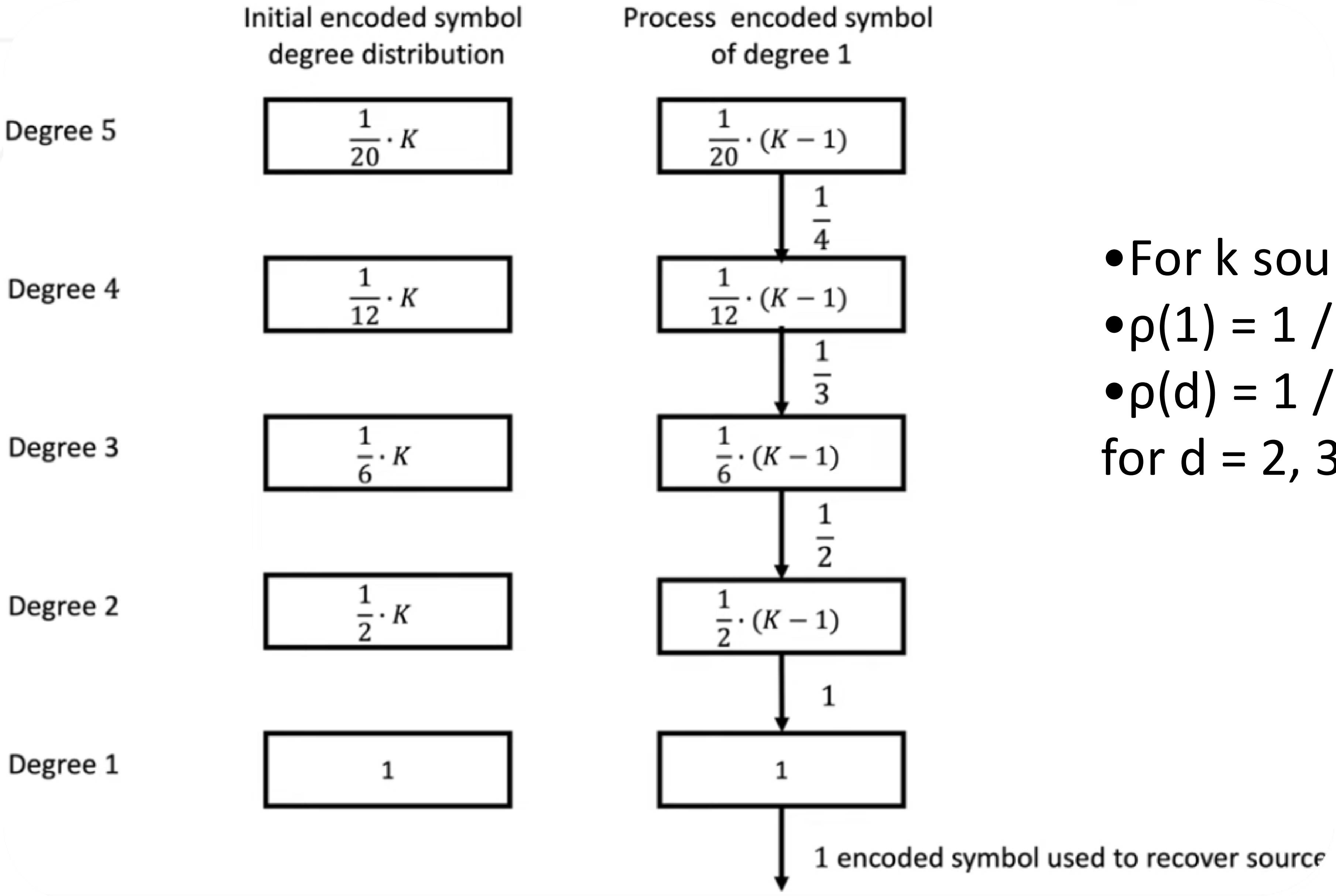
Michael G. Luby, Michael Mitzenmacher, and M. Amin Shokrollahi. 1998. Analysis of random processes via And-Or tree evaluation. SODA '98. Society for Industrial and Applied Mathematics, USA, 364–373.

Degree Distribution: Ideal Soliton

Ripple size: # of degree-1 encoded symbols



Degree Distribution: Ideal Soliton



- For k source symbols:
- $\rho(1) = 1 / k$
- $\rho(d) = 1 / [d(d-1)]$, for $d = 2, 3, \ldots, k$

Ideal LT decoding analysis

- Expectation of # of encoded symbols to recover K source symbols = K
- Decoding complexity: # edges KlnK

- Fragile in practice: ripple size can easily become 0 due to the random fluctuations -> decoding stalls

- To make decoding reliable under randomness, a small modification leads to the Robust Soliton Distribution

M. Luby, "LT codes," *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, Vancouver, BC, Canada, 2002, pp. 271-280

Degree Distribution: Robust Soliton

Definition 11 *(Robust Soliton distribution): The Robust Soliton distribution is* $\mu(\cdot)$ *defined as follows. Let* $R = c \cdot \ln(k/\delta)\sqrt{k}$ *for some suitable constant* $c > 0$. *Define*

$$\tau(i) = \begin{cases} R/ik & \texttt{for } i = 1, \ldots, k/R - 1 \\ R\ln(R/\delta)/k & \texttt{for } i = k/R \\ 0 & \texttt{for } i = k/R + 1, \ldots, k \end{cases}$$

Add the Ideal Soliton distribution $\rho(\cdot)$ *to* $\tau(\cdot)$ *and normalize to obtain* $\mu(\cdot)$:

- $\beta = \sum_{i=1}^{k} \rho(i) + \tau(i)$.
- *For all* $i = 1, \ldots, k$, $\mu(i) = (\rho(i) + \tau(i))/\beta$.

- Idea: add a safety buffer, making sure the ripple size is large enough
- Intuition:
  - ripple starts at a reasonable size R
  - tau(i) boost probabilities of lower-degree symbols: ripple less likely to vanish

Degree Distribution: Robust Soliton

Definition 11 *(Robust Soliton distribution): The Robust Soliton distribution is* $\mu(\cdot)$ *defined as follows. Let* $R = c \cdot \ln(k/\delta)\sqrt{k}$ *for some suitable constant* $c > 0$. *Define*

$$\tau(i) = \begin{cases} R/ik & \texttt{for } i = 1, \ldots, k/R - 1 \\ R\ln(R/\delta)/k & \texttt{for } i = k/R \\ 0 & \texttt{for } i = k/R + 1, \ldots, k \end{cases}$$

Add the Ideal Soliton distribution $\rho(\cdot)$ *to* $\tau(\cdot)$ *and normalize to obtain* $\mu(\cdot)$*:*

- $\beta = \sum_{i=1}^{k} \rho(i) + \tau(i)$.
- *For all* $i = 1, \ldots, k$, $\mu(i) = (\rho(i) + \tau(i))/\beta$.

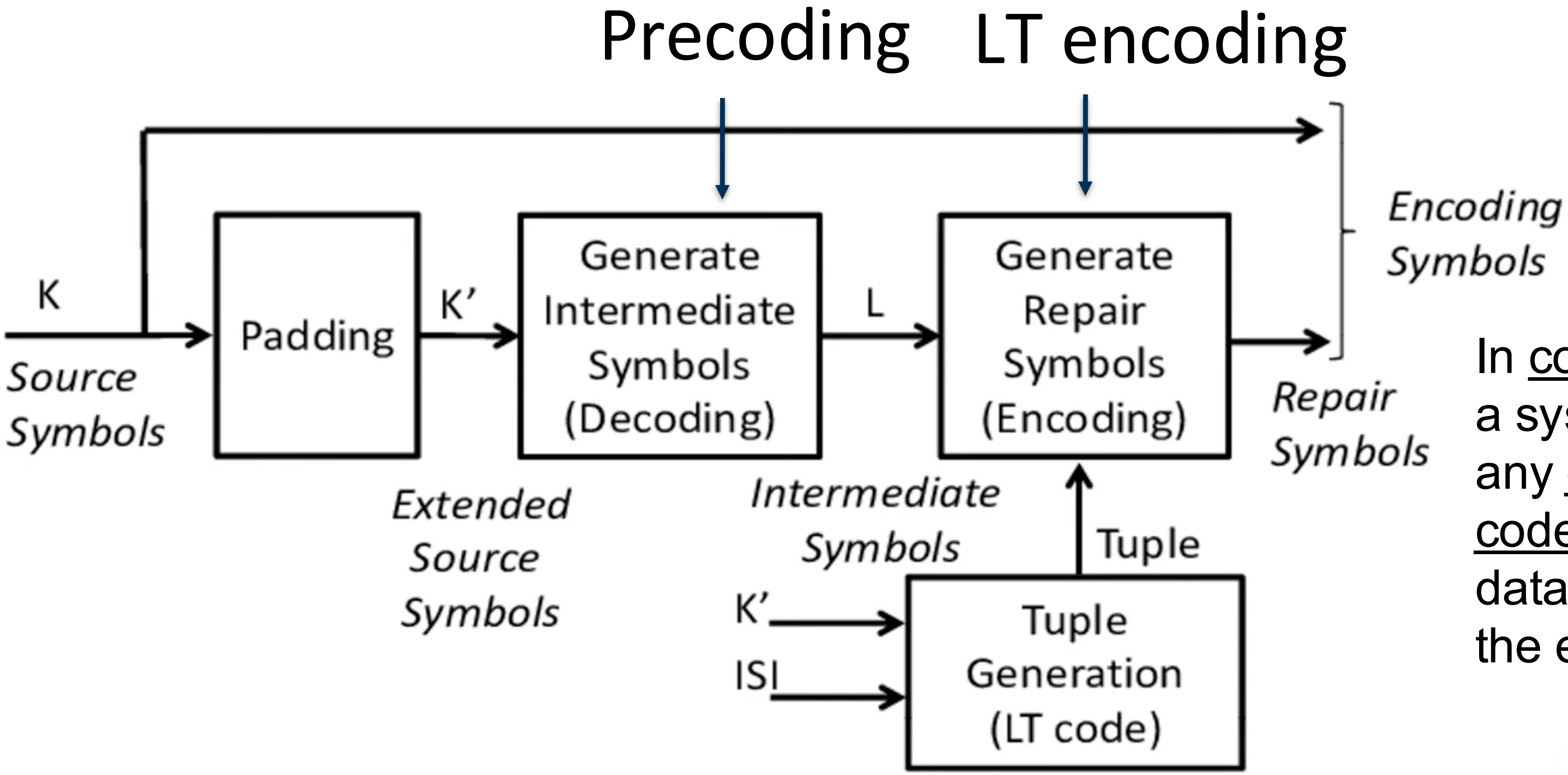
- Analysis Highlights:
  - Keeps ripple size = R w.h.p. -> avoids early stall.
  - Expected overhead ≈ $O(\sqrt{k} \ln^2(k/\delta))$
  - Still $O(k \log(k/\delta))$ encoding / decoding time.
- Practical LT and Raptor codes use this as the robust degree distribution for reliable decoding in noisy or finite-length settings.

Raptor Codes: pre-code + LT codes

LT codes are fundamental to the design of more advanced fountain codes.

Raptor codes is the most advanced fountain code, which is specified in IETF RFC 6330

RaptorQ Encoder



In coding theory, a systematic code is any error-correcting code in which the input data are embedded in the encoded output.

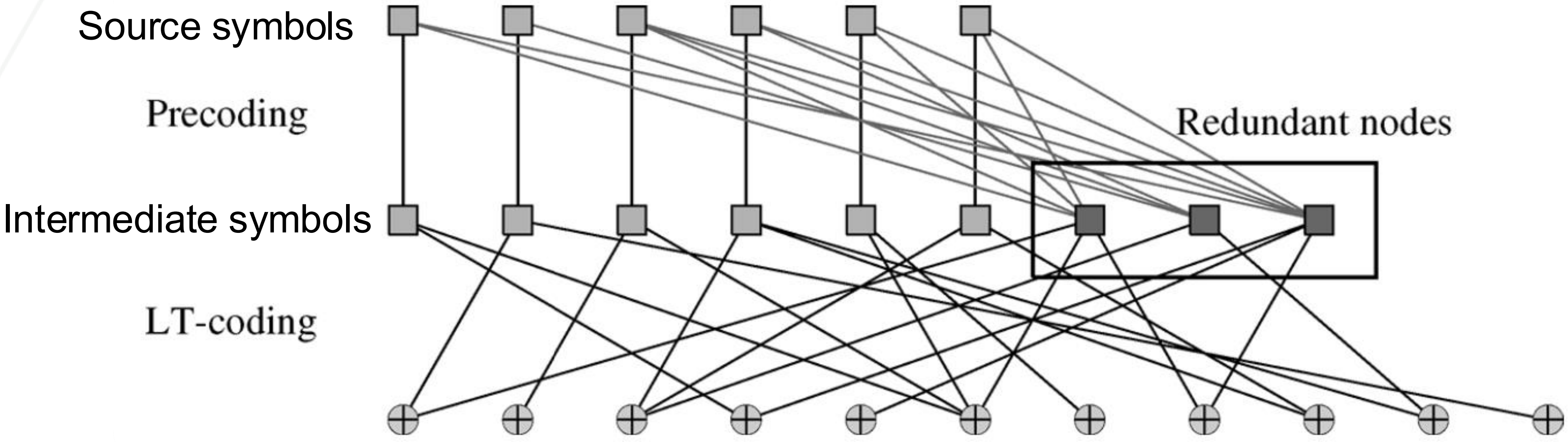
Georgia Tech

Raptor Codes: pre-code + LT codes

LT codes are fundamental to the design of more advanced fountain codes.

Raptor codes is the most advanced fountain code, which is specified in IETF RFC 6330

LT on top of a high-rate pre-code (R=0.95-0.99), so the LT layer only needs to recover most symbols; the pre-code fixes the rest.



Raptor Codes: optimized degree distribution

TABLE I
DEGREE DISTRIBUTIONS FOR VARIOUS VALUES OF k; ε IS THE OVERHEAD, AND a IS THE AVERAGE DEGREE OF AN OUTPUT SYMBOL

k	65536	80000	100000	120000
Ω_1	0.007969	0.007544	0.006495	0.004807
Ω_2	0.493570	0.493610	0.495044	0.496472
Ω_3	0.166220	0.166458	0.168010	0.166912
Ω_4	0.072646	0.071243	0.067900	0.073374
Ω_5	0.082558	0.084913	0.089209	0.082206
Ω_8	0.056058		0.041731	0.057471
Ω_9	0.037229	0.043365	0.050162	0.035951
Ω_{18}				0.001167
Ω_{19}	0.055590	0.045231	0.038837	0.054305
Ω_{20}		0.010157	0.015537	
Ω_{65}	0.025023			0.018235
Ω_{66}	0.003135	0.010479	0.016298	0.009100
Ω_{67}		0.017365	0.010777	
ε	0.038	0.035	0.028	0.02
a	5.87	5.91	5.85	5.83

- The LT layer can use a lower mean degree (3–6) than robust soliton
- Higher degree are sparser, skipping some high degree
- Expectation of # of encoded symbols to recover K sources: K(1+epsilon)
- Decoding complexity: near linear O(K)

Raptor Codes: inactivation decoding

- Idea: keep peeling, but when the ripple stalls, temporarily "inactivate" a few sources to break cycles; solve the tiny dense core at the end using Gaussian Elimination (GE)

- First, decoding seeks out the intermediate symbols that could be solved by peeling. Whenever the process get stuck, inactivate an intermediate symbol to make peeling continue.
- Second, use GE to solve a dense core for the inactivated symbols
- Third, plug in the result from 2nd phase to fully recover all the symbols

- How to select which one to inactivate
  - the decoder maintains a list of candidate sources sorted by degree
  - each time peeling stalls, it inactivates the lowest-degree source

Georgia Tech

Raptor Codes: inactivation decoding

- Decoding Complexity:
  - Peeling: O(K).
  - Dense core size s: O(logK) in analysis/design; dense solve O(s^3) → negligible vs K.
  - Total: near-linear O(K).
- With proper degree + pre-code, decoding succeeds w.h.p. once received symbols >= K(1+epsilon)

Solutions for Networking

- Legacy reliable transport: retransmission
- Multicast: limited feedback channel
- Multipath solution; Distributed matrix multiplication
- Video streaming and cloud gaming

Adaptive coding

Dynamically tuned resiliency coding ensures integrity and high throughput.

Multipath

Simultaneously use satellite, LTE, or optical for faster and more resilient delivery.

Low SWaP

Runs on lightweight embedded platforms and constrained compute.

One-Way Compatible

Send large files or streams with no return path required.

Connected Vehicles

Enhanced Communications. Evolved Driving Experiences.

Defense Communications

Stealth, Fast, and Reliable for Mission-Critical Communications.

Satellite & Aerospace

High Altitude. Total Resilience.

Media Live Contribution

Broadcast Video from Field Contribution to Global Distribution.

IoT Data Acceleration

Real-Time IoT, from "ANY"-Range Wireless.

Cloud Gaming & Interactive Apps

Flawless Gaming. Any Location. Any Network.

Conclusion

- We explored how LT and Raptor codes make reliable data delivery possible even when packets are lost.
  - LT codes is the first practical implementation of fountain codes: generate unlimited encoded packets and let the receiver collect any subset slightly larger than the original data.
  - The ideal soliton distribution kept decoding smooth in theory, while the robust soliton added extra low-degree symbols to stabilize the ripple for real systems.
  - Raptor codes built on LT by adding a light precode and optimizing the LT degree distribution, achieving near-linear complexity and very small overhead.