

# Coding for Distributed Information Systems

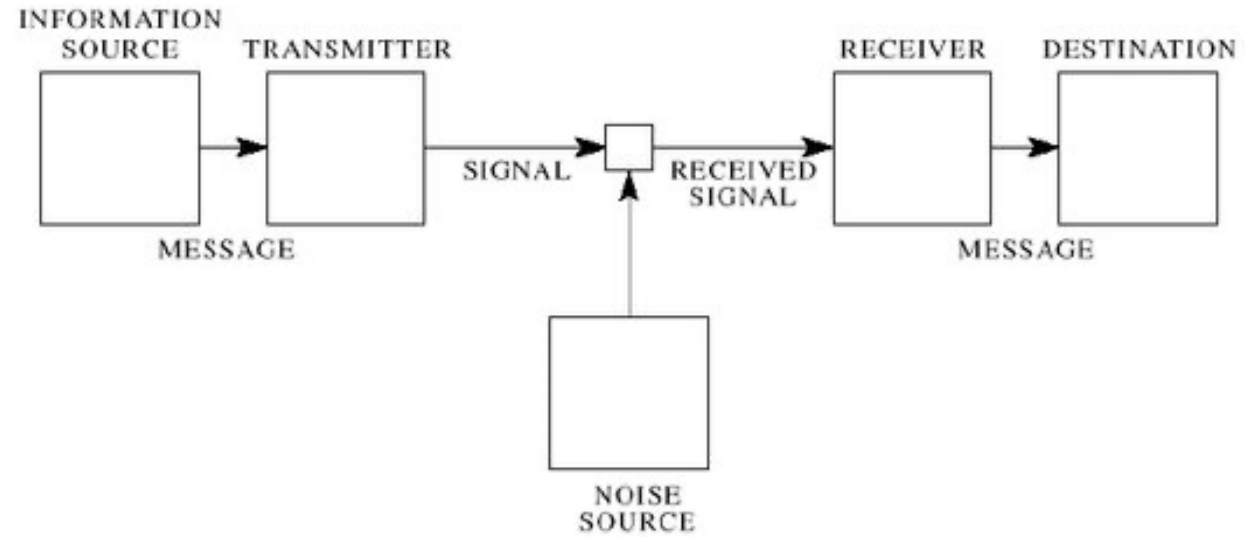
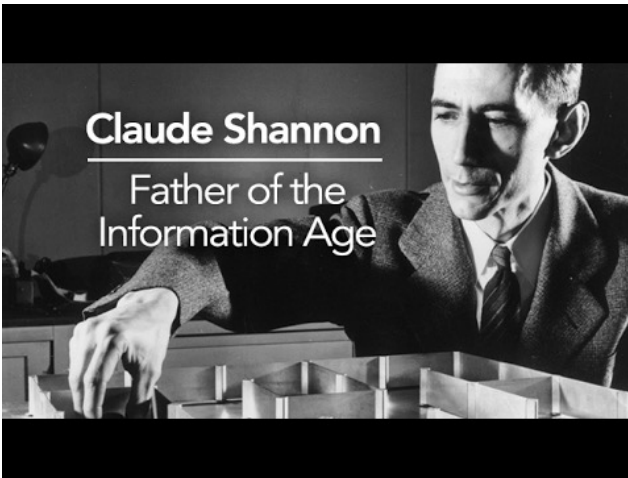
Mohammad Ali Maddah-Ali

Department of Electrical Engineering

Sharif University of Technology

**Collaboration with: Mahtab Mirmohseni, Tayyeb Jahani-Nezhad,  
Nastaran Abadi, Ali Khaled**

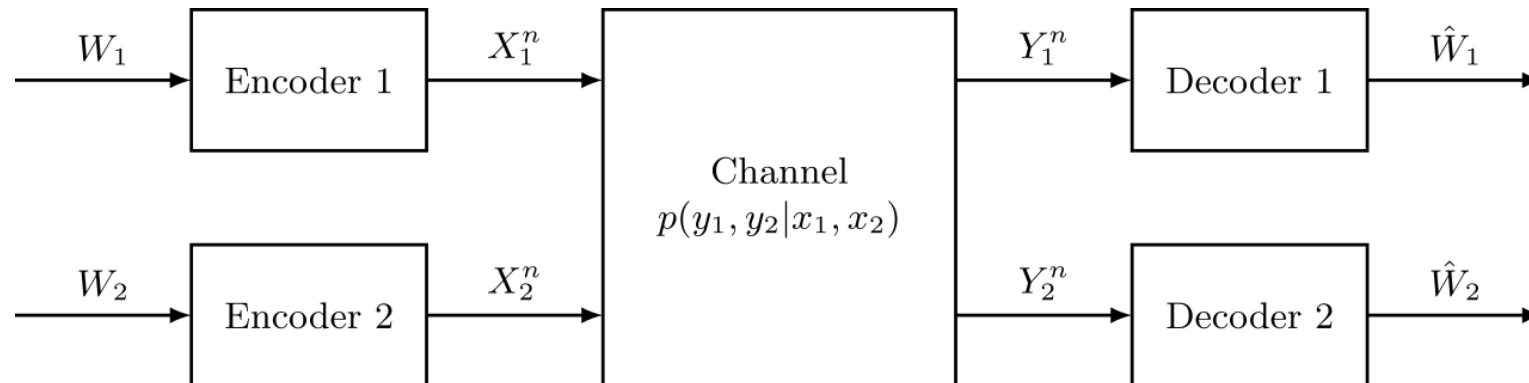
# Coding Theory



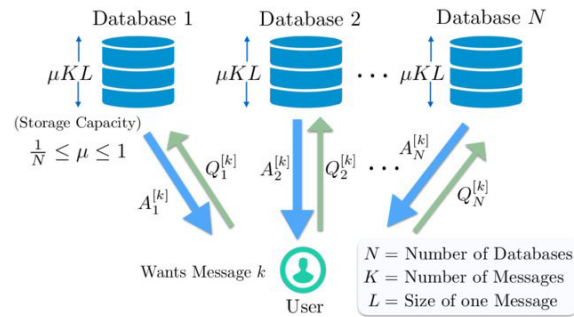
We can achieve **vanishing probability of error** with **a non-vanishing rate**.

# Multi-User Information Theory

- Multi-Access Channels
- Broadcast Channels
- Interference Channels
- Relay Networks

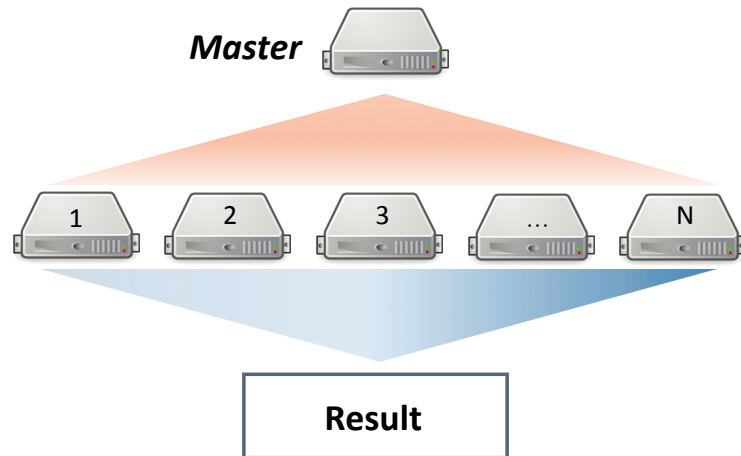


# New Applications

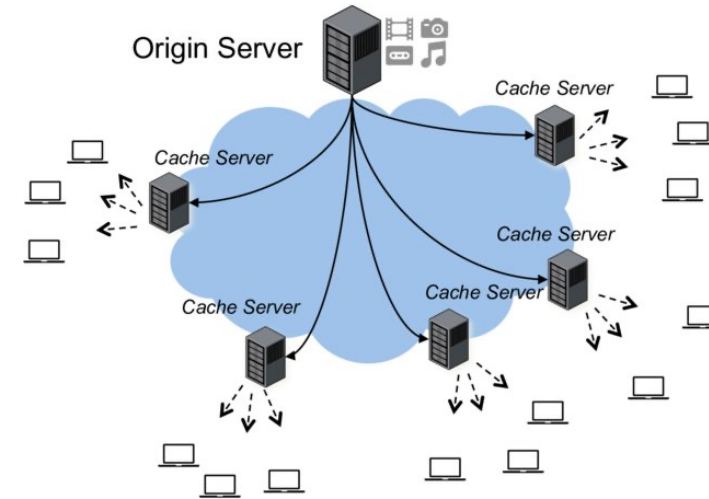


## Private Information Retrieval

Courtesy of Attia, Kumar, Tandon [2018]

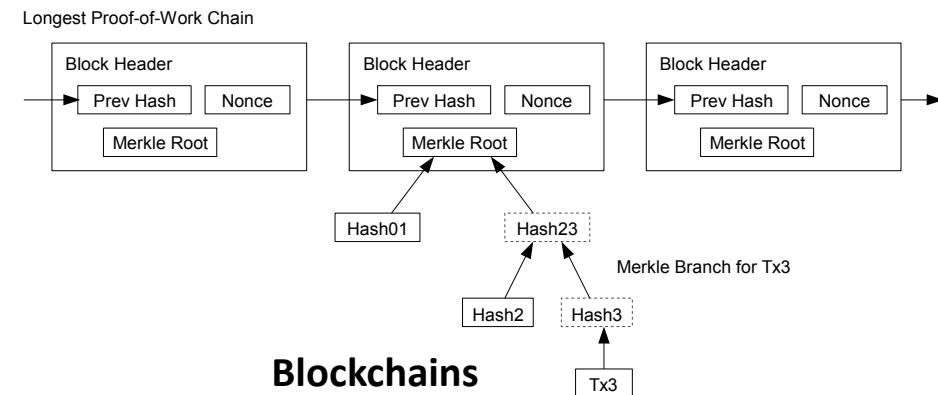


## Distributed Computing/Storage



## Distributed Cache Networks

Courtesy of Bruneau-Queyreix [2017]

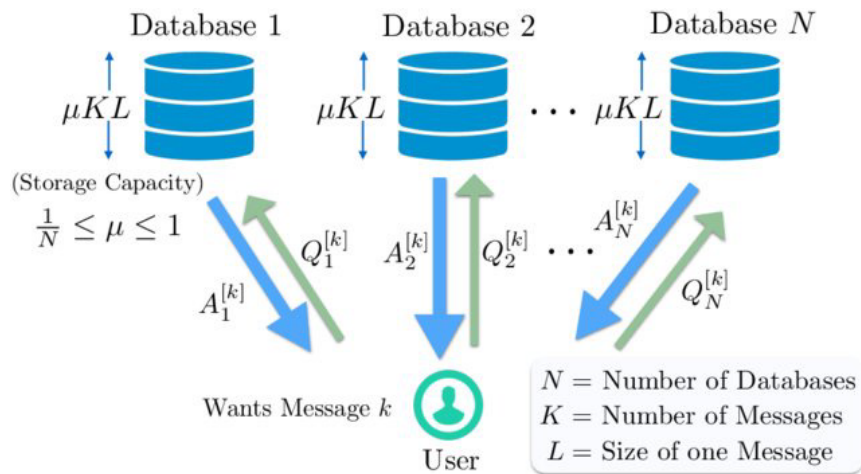


## Blockchains

Courtesy of Nakamoto [2008]



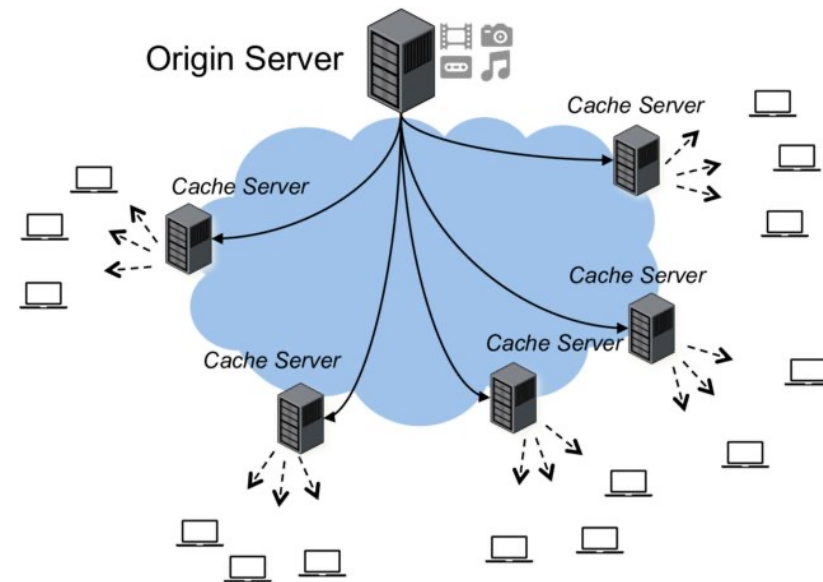
# New Applications: New Aspect



## Private Information Retrieval

Fig. Credit: of Attia, Kumar, Tandon [2018]

Receiver receives a file,  
while transmitters don't realize which!

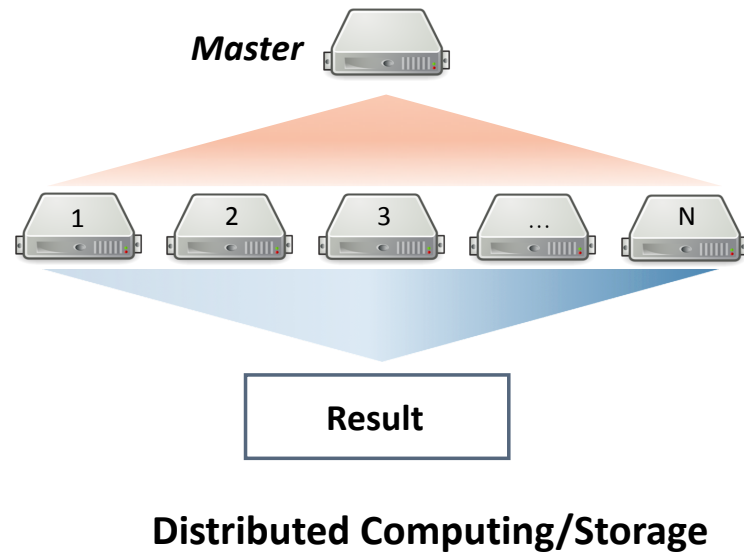


## Distributed Cache Networks

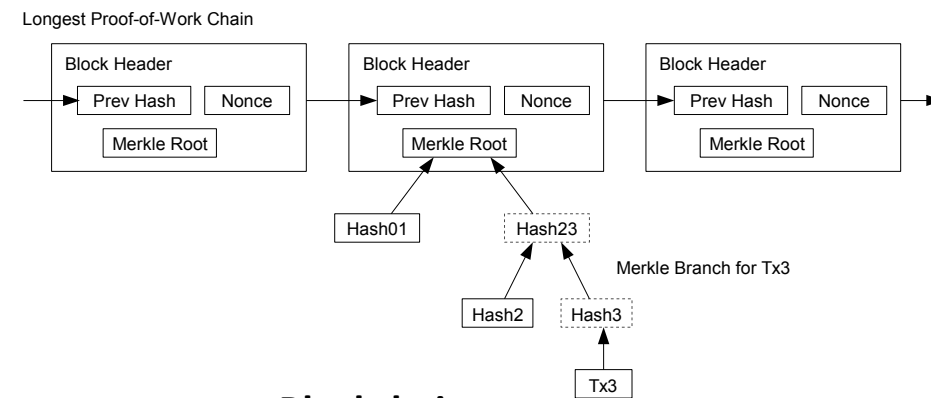
Figure Credit: Bruneau-Queyreix [2017]

Communication with limited budget for side-information

# In this talk



**Approximate Decoding**



Courtesy of Nakamoto [2008]

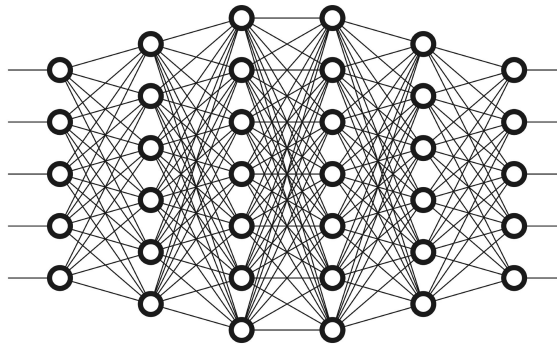
**Distributed Encoding**

# Coded Computing and Approximate Decoding

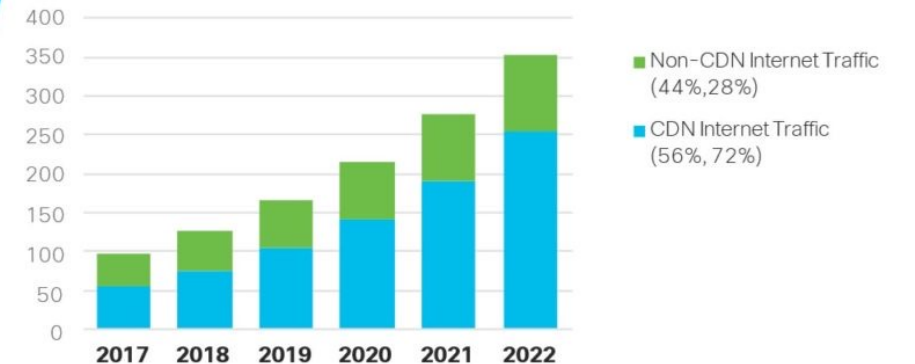
Jahani-Nezahd, Maddah-Ali [2020]

# The Size of Data is Exploding

- Big Data
  - Social Networks:  $10^9$  nodes and  $10^{12}$  edges
  - $10^{10}$  Pages in internet
- Huge Models
  - $10^{10}$  weights in Deep Learning
- Enormous Computing



Exabytes  
per Month



\* Figures (n) refer to 2017, 2022 traffic share

Source: Cisco VNI Global IP Traffic Forecast, 2017-2022

**We have to offload the computation to many servers**

# Offloading Computation

**Source**

$$\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K$$

***N Servers***



***Data Collector***

$$g(\mathbf{X}_1, \dots, \mathbf{X}_K)$$

# Offloading Computation

**Source**

$$\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K$$

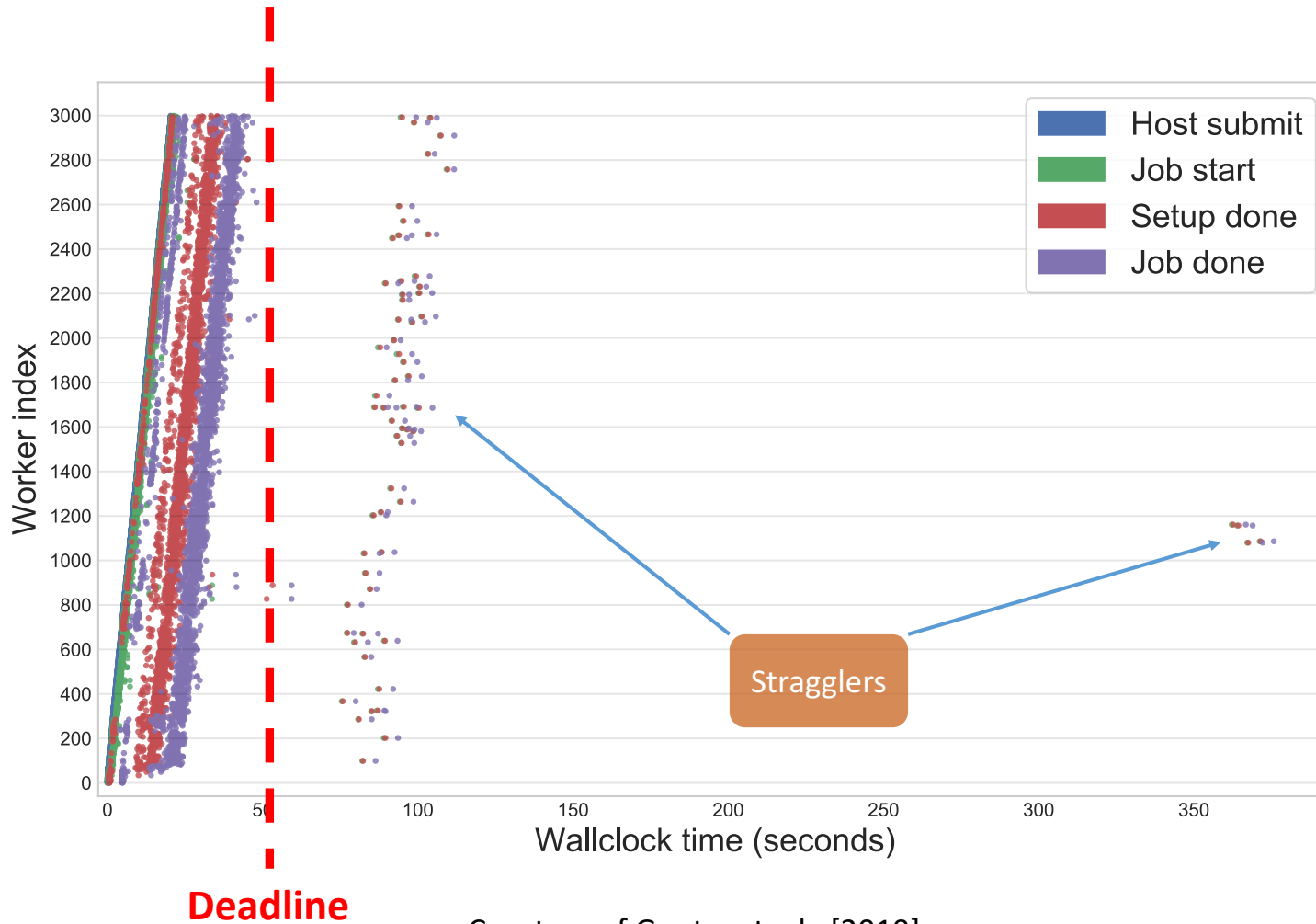
***N Servers***



***Data Collector***

$$g(\mathbf{X}_1, \dots, \mathbf{X}_K)$$

# Separating lines of research



Courtesy of Gupta, et. al. [2019]

# Offloading Computation

*Source*

$$\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K$$

*N Servers*



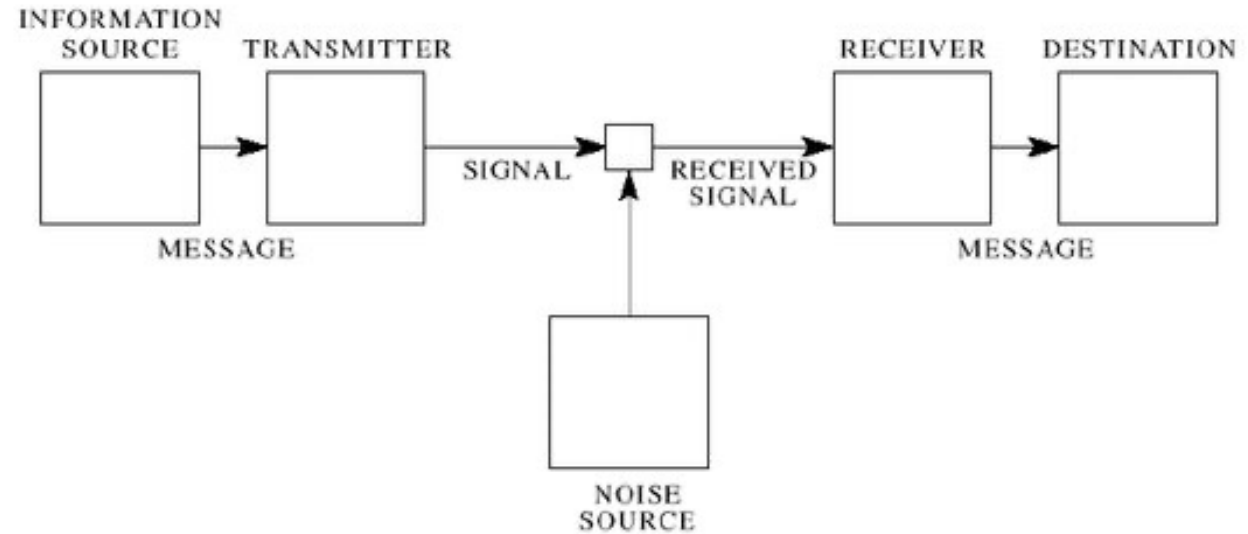
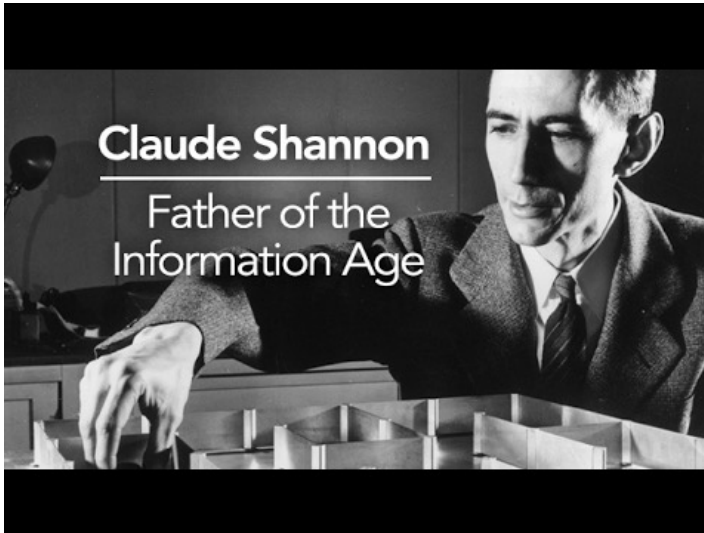
*Data Collector*

$$g(\mathbf{X}_1, \dots, \mathbf{X}_K)$$

How to deal with faulty/slow nodes?



# Communication: Approaching Shannon Capacity



# Coding for Communication

*Source*

$\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K$

*Encoder*

*N Channel Use*

**X**

1

2

**X**

N

*Decoder*

*Receiver*

$\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K$

# Reed-Solomon Codes

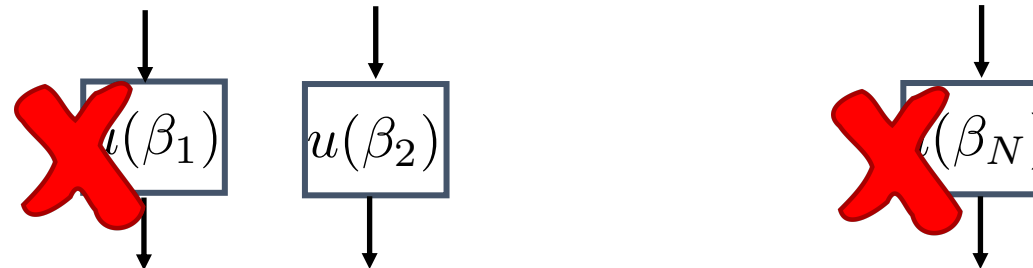
**Source**

$$\boxed{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K} \equiv u(z) \quad \text{Polynomial}$$

**Encoder**

$$\boxed{u(z) = \mathbf{X}_1 + \mathbf{X}_2 z + \dots + \mathbf{X}_K z^{K-1}}$$

**Channel**



*Computation Efficiency*

**Decoder**

**Reconstructing  $u(z)$**

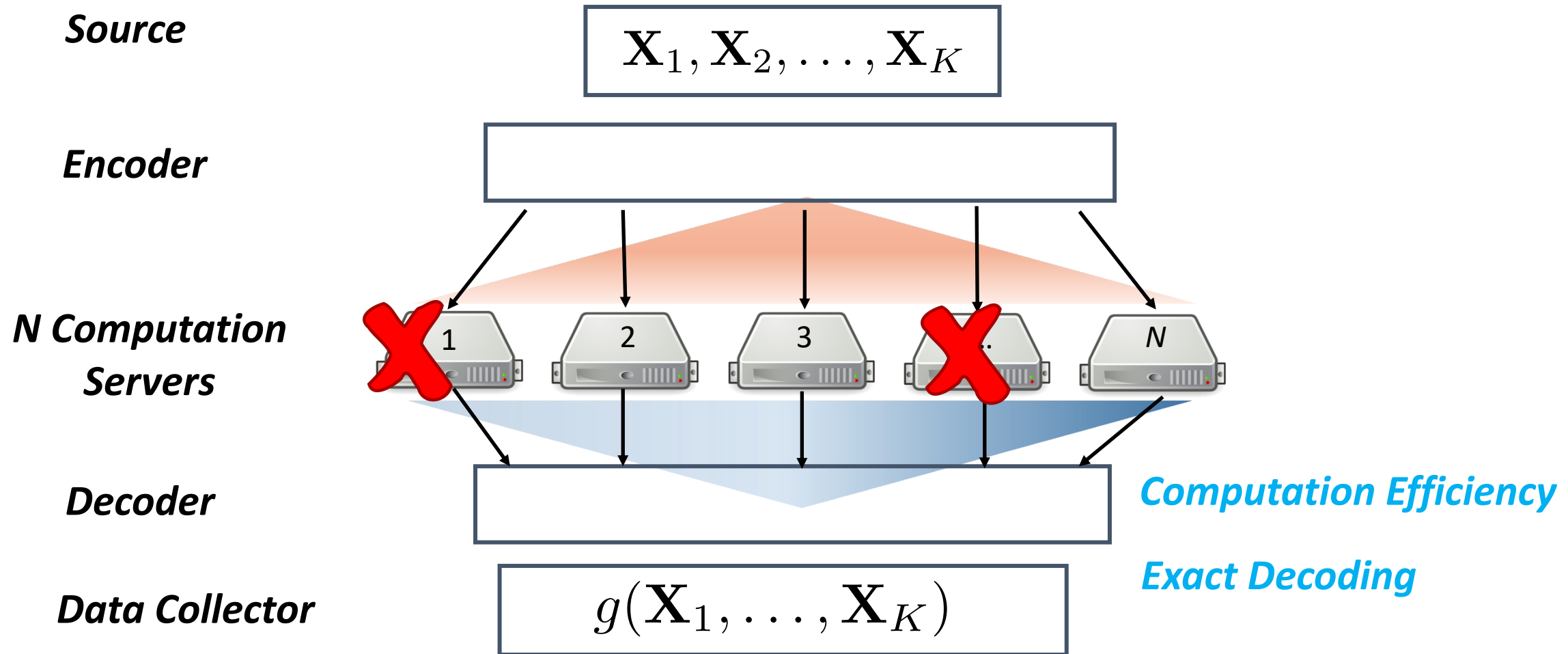
*Exact Decoding*

**Receiver**

$$\boxed{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K} \quad \text{Complexity} = K \log(K)$$

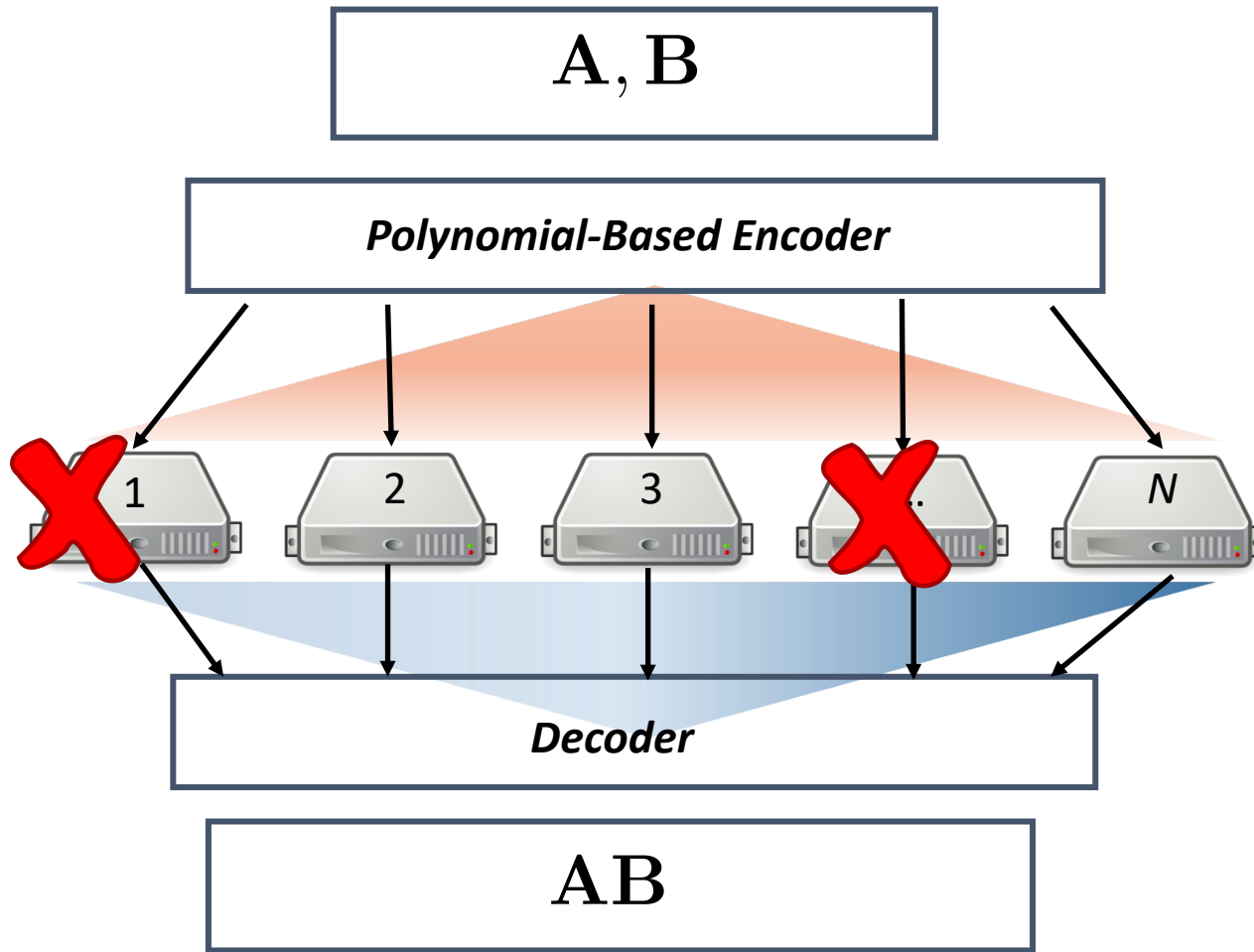
Number of Samples  $\geq \deg(u(Z)) + 1$

# Coding for Computation



**Main Challenge: How to Design a Code that Goes Through The Computation**

# Coded Computing for Matrix Multiplication

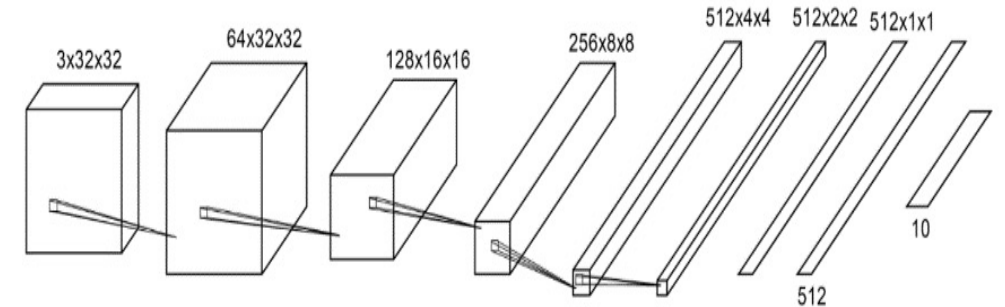


Polynomial Codes [Yu, Maddah-Ali, Avestimehr, 2016]

Entangle Codes [Yu, Maddah-Ali, Avestimehr, 2017]

PolyDot Codes [Dutta, Fahim, Haddadpour, Jeong, Cadambe, Grover [2017]

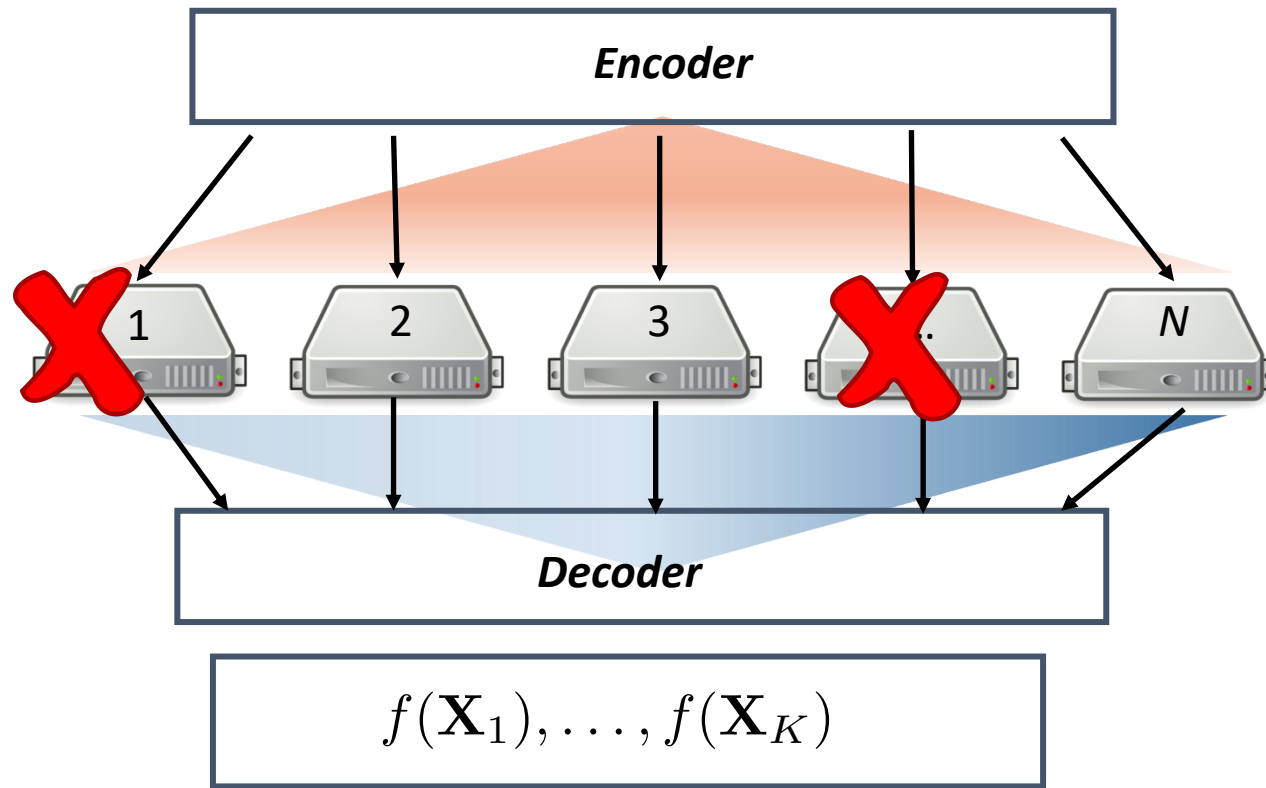
Deep Neural Network



# Coded Computing for Polynomial Computation

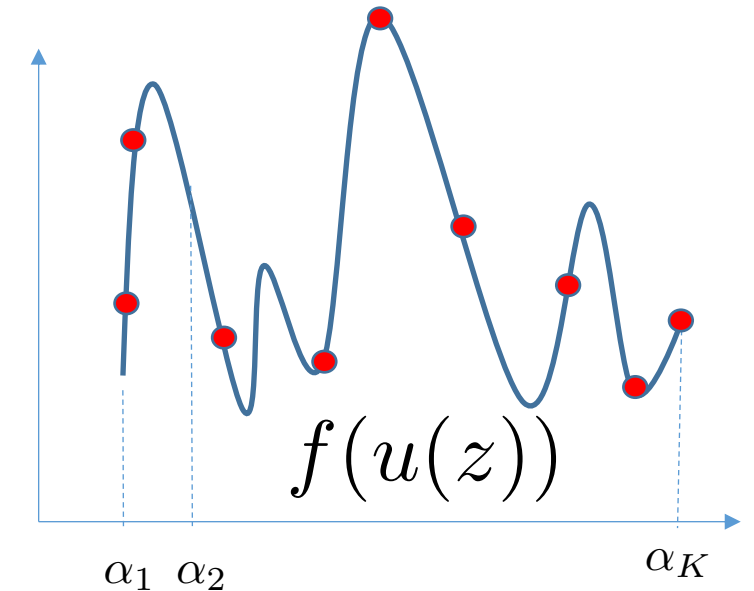
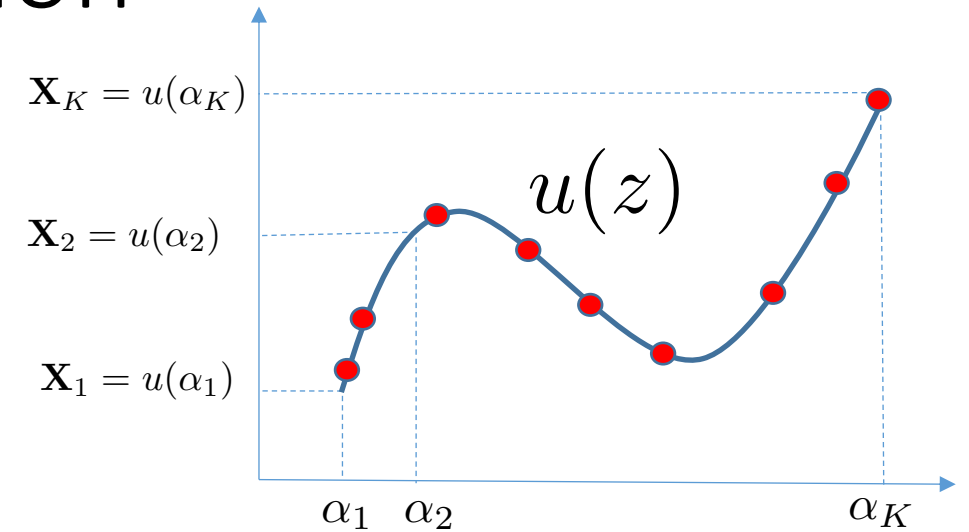
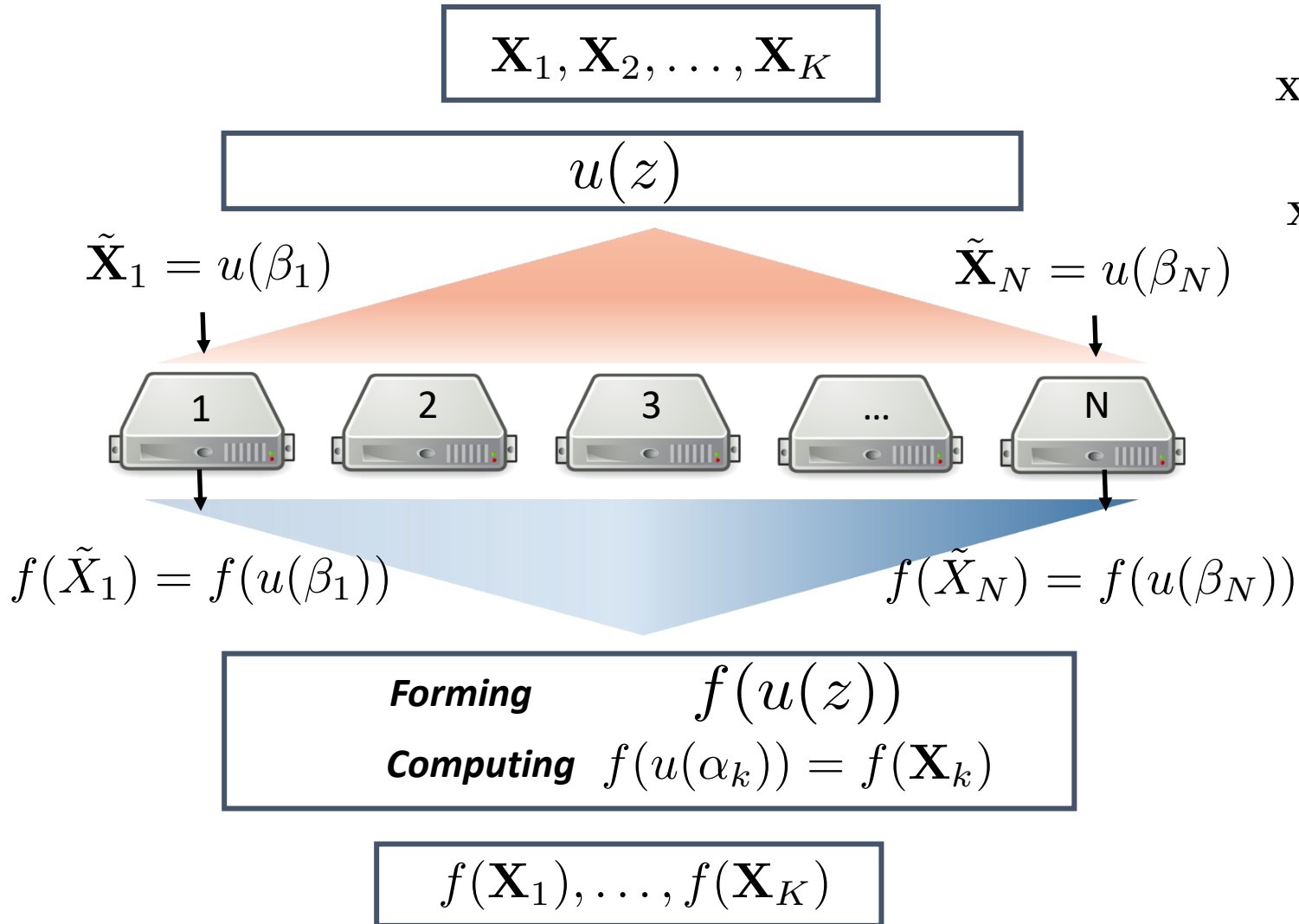
$\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K$

Yu, Li, Raviv, Mousavi Kalan, Soltanolkotabi, Avestimehr [2017]



$f(x)$ : Any general polynomial

# Lagrange Coded Computation



# Lagrange Coded Computation

$$\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K$$

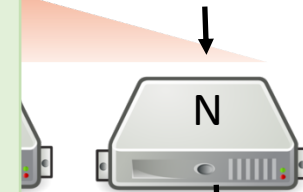
$$u(z)$$

$u(z)$  is a polynomial

$f(z)$  is a polynomial

$f(u(z))$  is a polynomial as well

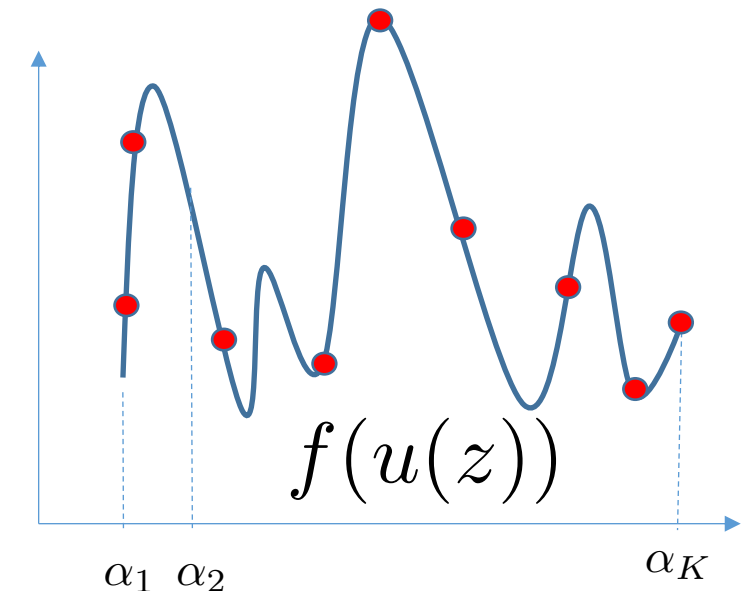
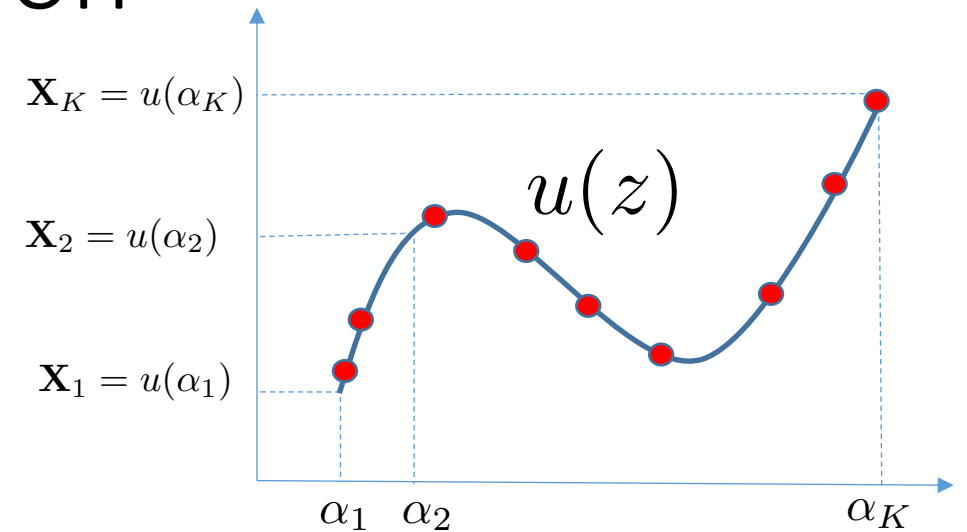
$$\tilde{\mathbf{X}}_N = u(\beta_N)$$



$$\tilde{\mathbf{X}}_N = f(u(\beta_N))$$

Forming  $f(u(z))$   
Computing  $f(u(\alpha_k)) = f(\mathbf{X}_k)$

$$f(\mathbf{X}_1), \dots, f(\mathbf{X}_K)$$





# Lagrange Coded Computation

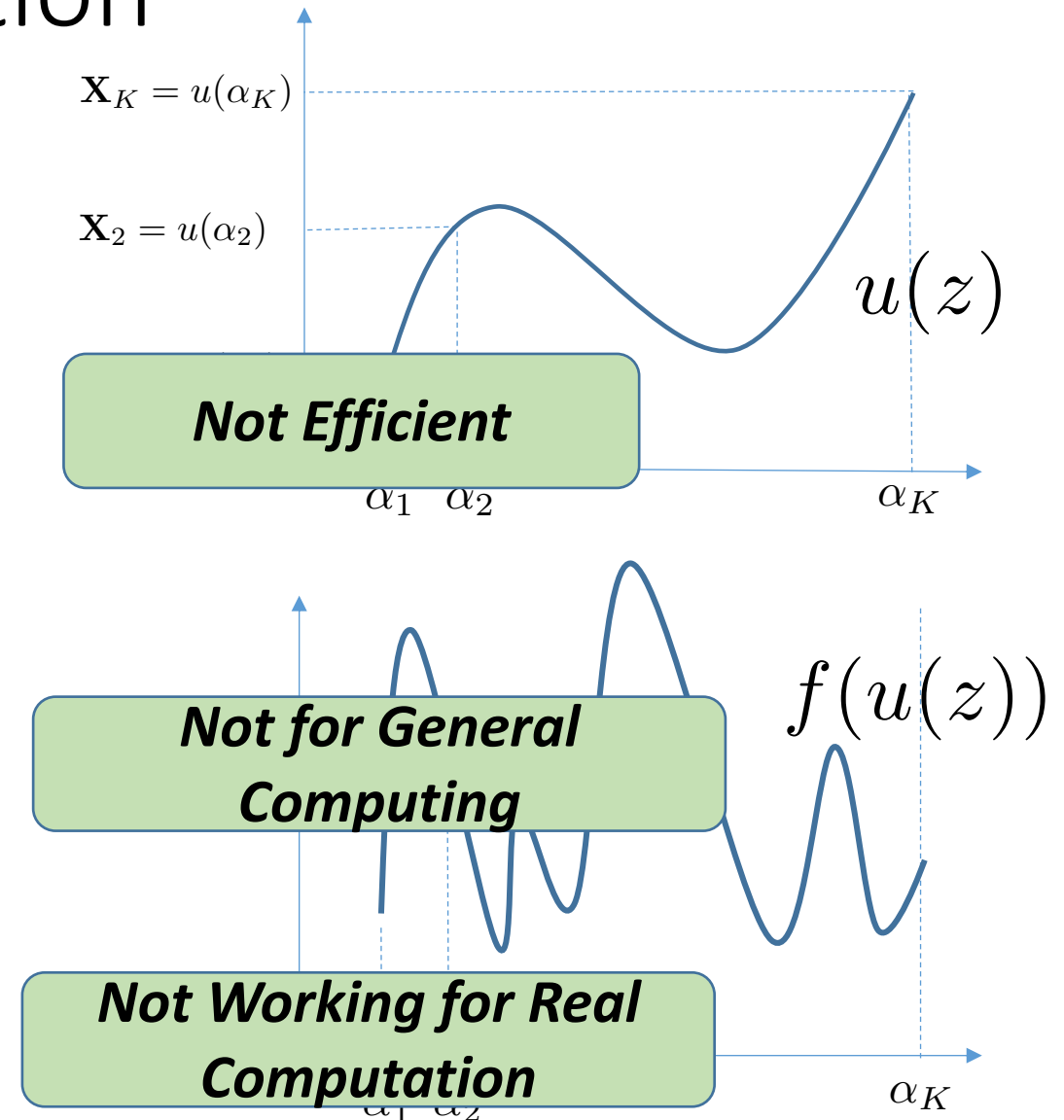
Challenges:

1. Number of samples needed  $K \cdot \deg(f)$

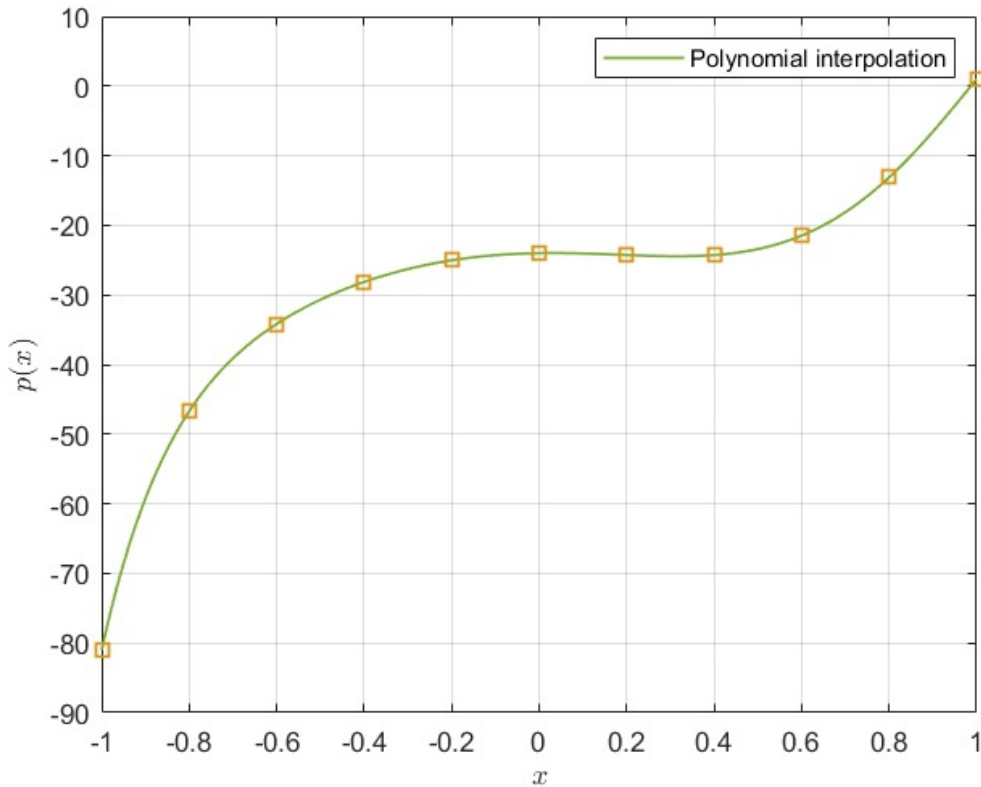
$$\begin{aligned}\# \text{ of Samples} &= \deg(f(u(z))) + 1 \\ &= (K - 1) \deg(f) + 1\end{aligned}$$

2. Only works for polynomial functions

3. It is not numerically stable, for real computation



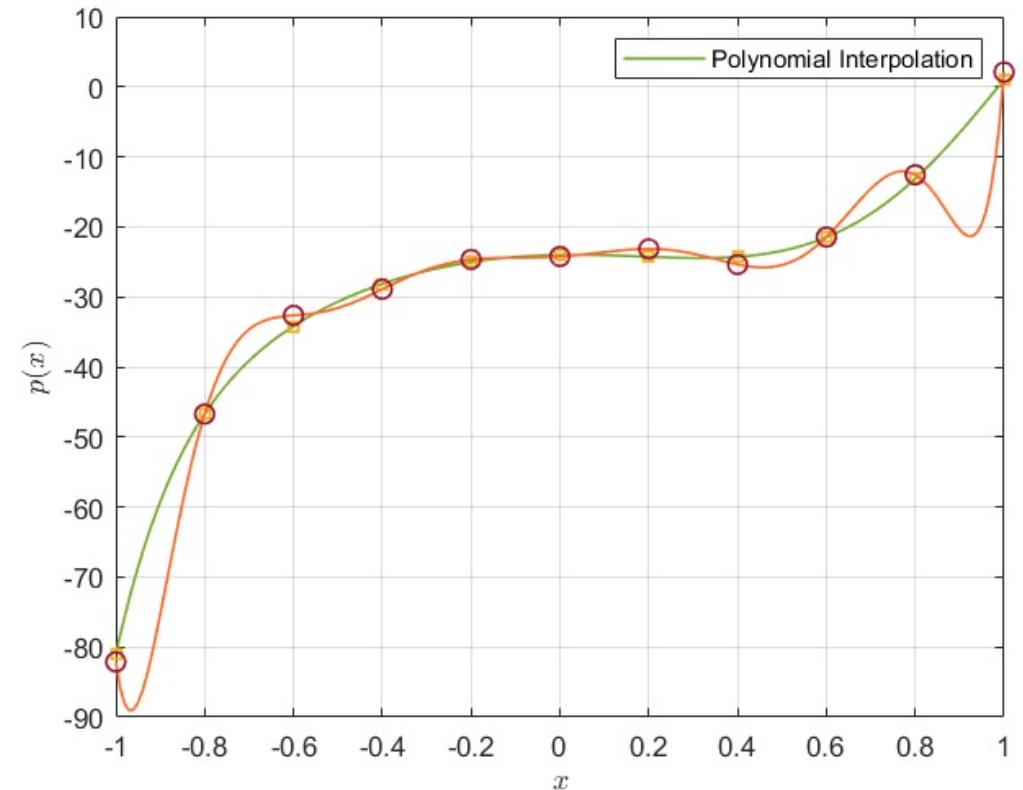
# On Numerical Instability



Coefficient = [6, -17, 0, -3, 12, 21, 22, -17, 1, -24]



The error  
caused by  
floating-  
point  
arithmetic



Coefficient=[1769.4, 16.5, -3721.2, -54.1, 2513.7, 855, -606.2, -9.8, 285, 4, -24.2]

# On Numerical Instability

$$\begin{bmatrix} \text{Vandermonde} \end{bmatrix} \begin{bmatrix} \text{Results of non-straggling nodes} \end{bmatrix} = \begin{bmatrix} -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_m & \alpha_m^2 & \dots & \alpha_m^{n-1} \end{bmatrix}$$

Inverting Vandermonde matrix is **numerically unstable** for  $n \geq 25$ .



Exact Computing



Polynomial Encoding

# Berrut Approximated Coded Computing

Jahani-Nezahd, Maddah-Ali [2020]

# Berrut Approximated Coded Computation

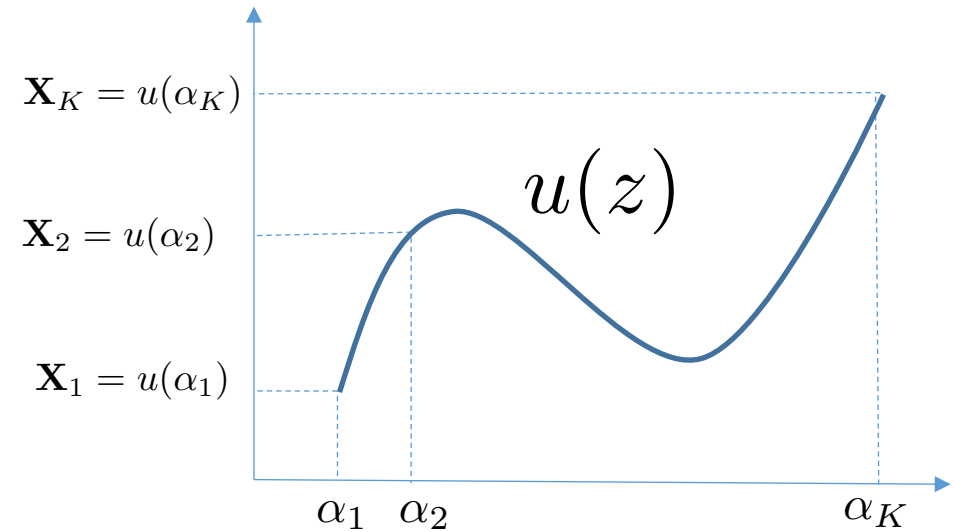
- ✓ Based on Approximation Theory and Numerical Analysis
- ✓ is **not limited** to polynomial function computation (Model Agnostic)
- ✓ **works for any number** of non-straggling worker nodes
- ✓ is **not limited** to computation over finite fields
- ✓ is **numerically stable**
- ✓ **has low computational complexity**

# Berrut Coded Computation

$$u(z) = \sum_{k=0}^{K-1} \frac{\frac{(-1)^k}{(z - \alpha_k)}}{\sum_{j=0}^{K-1} \frac{(-1)^j}{(z - \alpha_j)}} \mathbf{X}_k$$

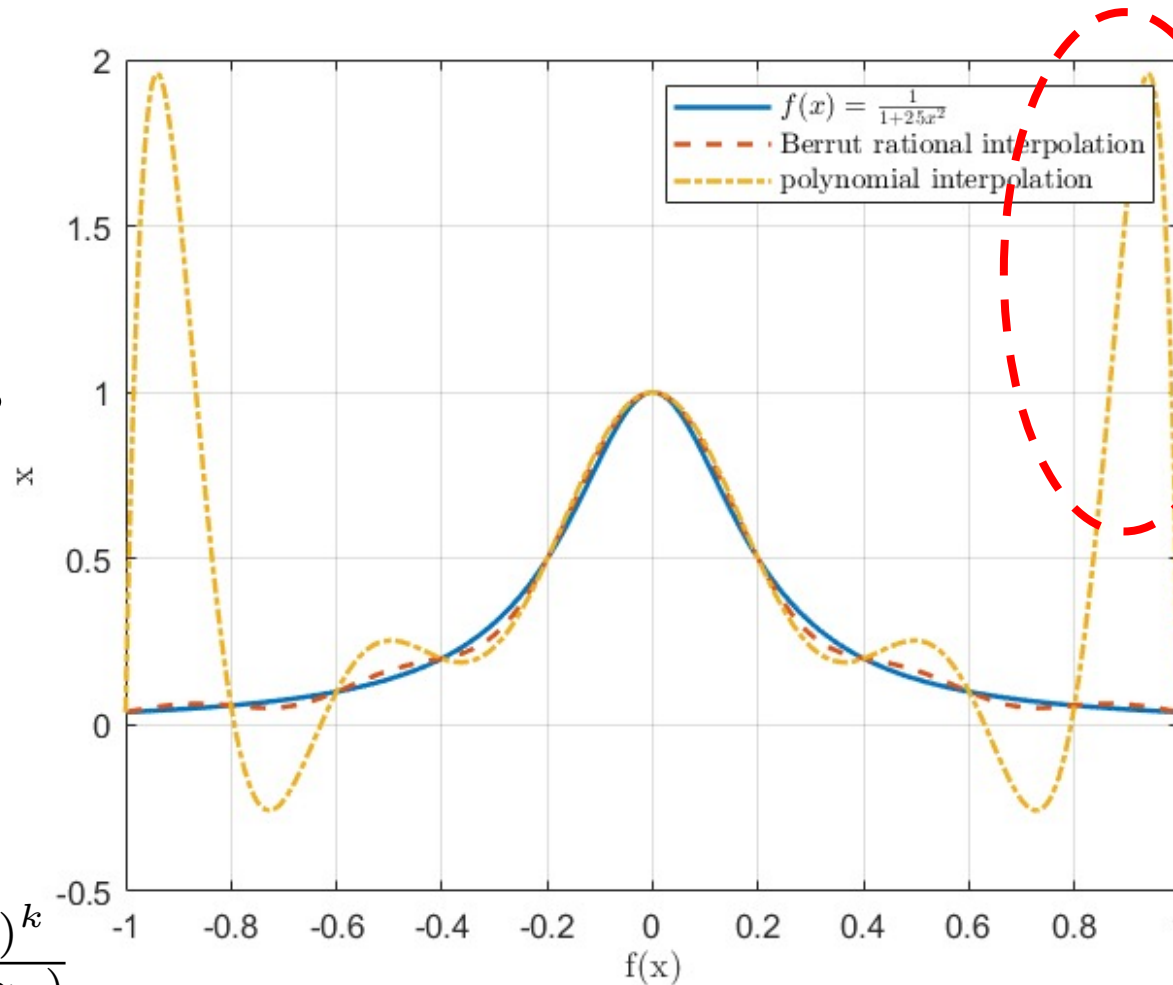
Chebyshev points of the first kind:

$$\alpha_j = \cos\left(\frac{(2j+1)\pi}{2K}\right), \quad j \in [K-1]$$



**Why Rational?**

# Why Rational Interpolation?



Runge's Phenomenon

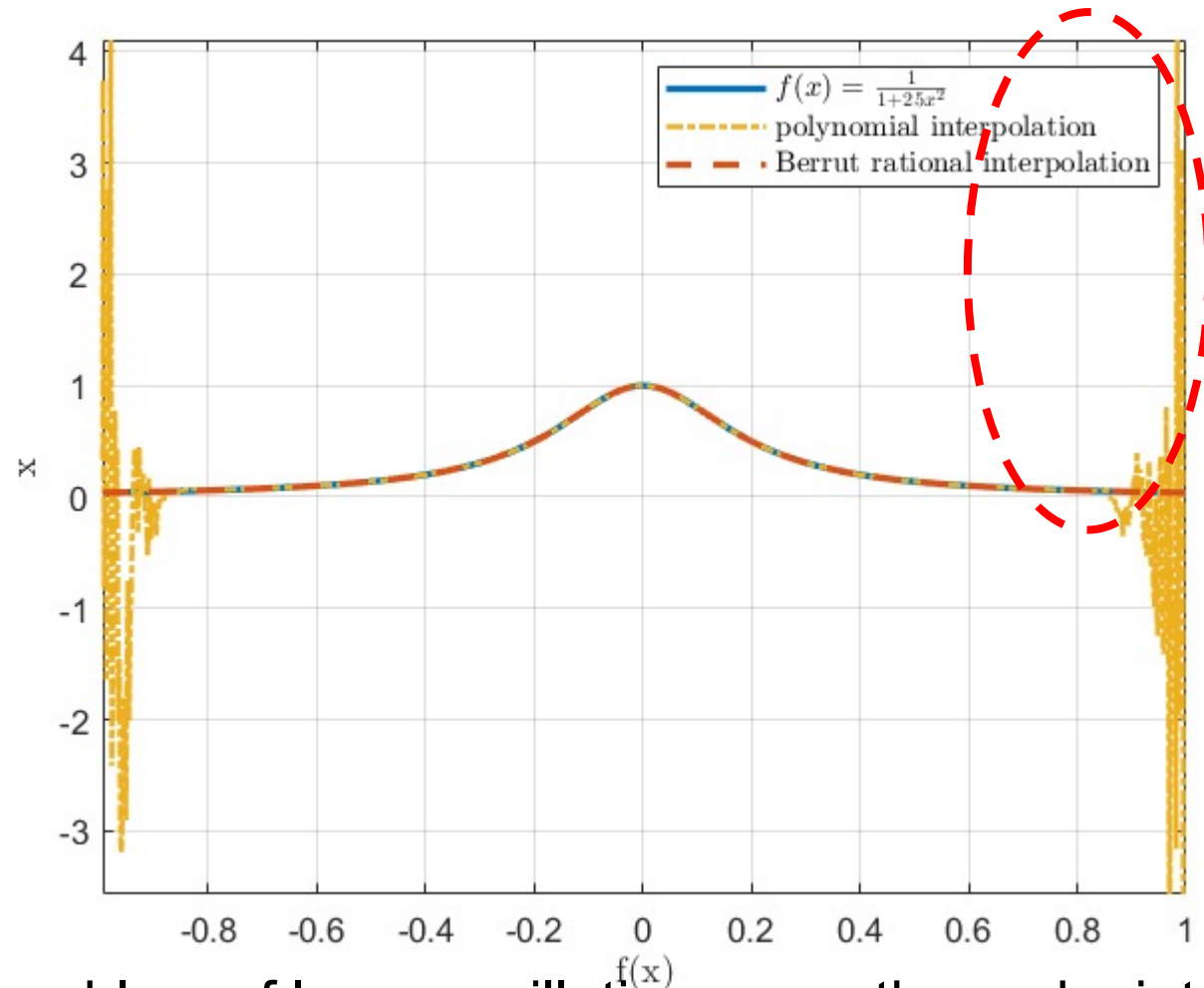
$$f(x) = \frac{1}{1 + 2.5x^2}$$

10 Equidistance Samples

$$u(z) = \sum_{k=0}^{K-1} \frac{(-1)^k}{(z - \alpha_k)} \mathbf{x}_k$$

# Why Rational Interpolation?

**60 Chebyshev Samples**



- ✓ Prevents the problem of large oscillations near the endpoints (Runge phenomenon)
- ✓ Berrut's rational interpolation is **more numerically stable**



# Berrut Coded Computation

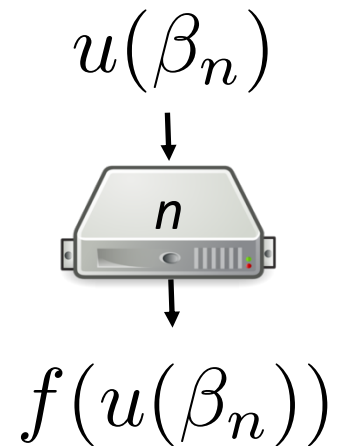
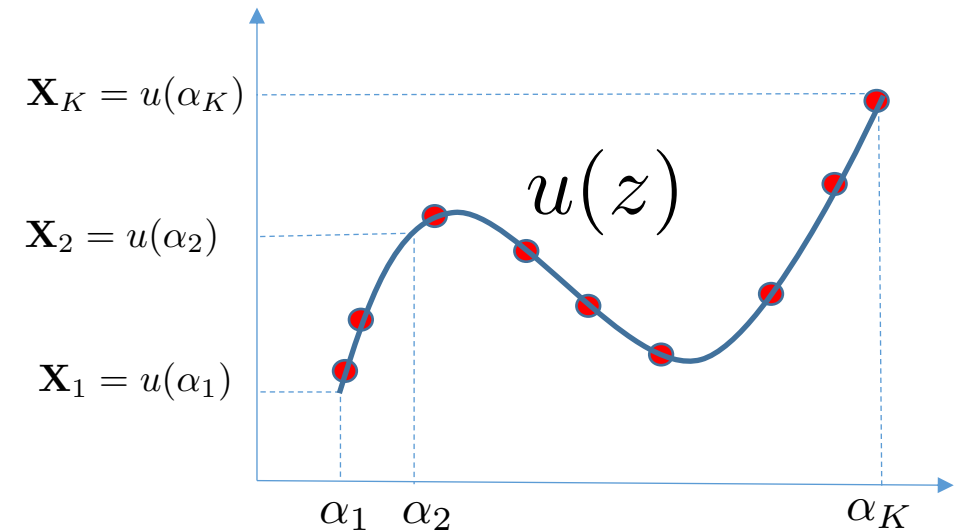
$$u(z) = \sum_{k=0}^{K-1} \frac{\frac{(-1)^k}{(z-\alpha_k)}}{\sum_{j=0}^{K-1} \frac{(-1)^j}{(z-\alpha_j)}} \mathbf{X}_k$$

Chebyshev points of the first kind:

$$\alpha_j = \cos\left(\frac{(2j+1)\pi}{2K}\right), \quad j \in [K-1]$$

Chebyshev points of the second kind:

$$\beta_j = \cos \frac{j\pi}{N}, \quad j \in [N]$$



# Berrut Coded Computation

$$u(z) = \sum_{k=0}^{K-1} \frac{\frac{(-1)^k}{(z-\alpha_k)}}{\sum_{j=0}^{K-1} \frac{(-1)^j}{(z-\alpha_j)}} \mathbf{X}_k$$

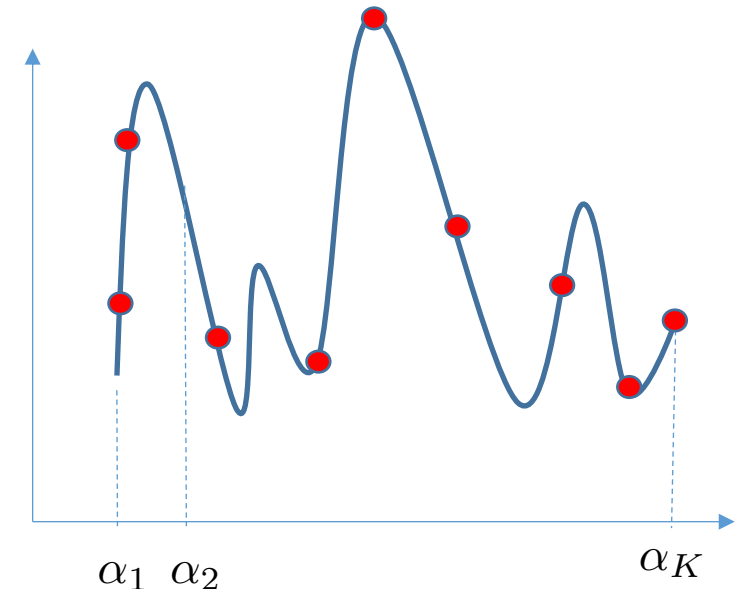
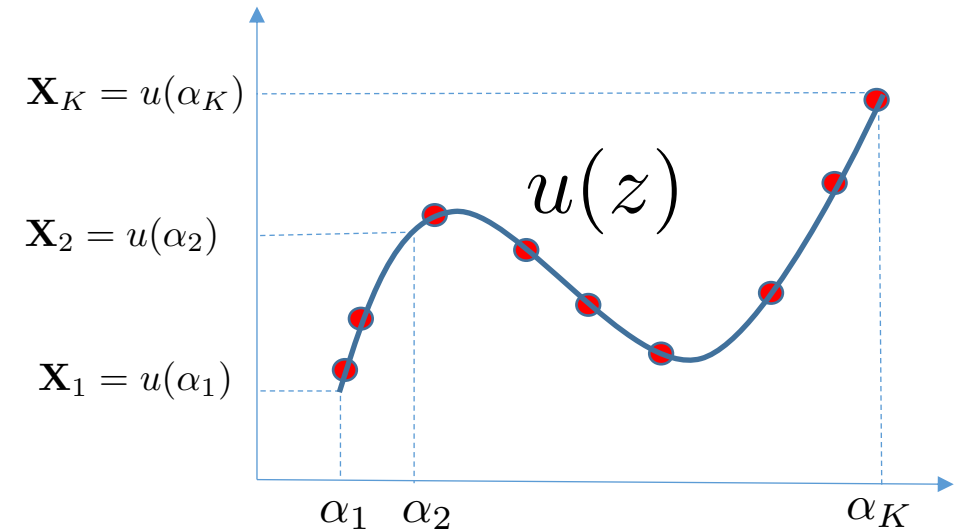
$$\alpha_j = \cos\left(\frac{(2j+1)\pi}{2K}\right), \quad j \in [K-1]$$

$$\beta_j = \cos \frac{j\pi}{N}, \quad j \in [N]$$

$$\mathcal{F} = \{\hat{\beta}_1, \hat{\beta}_2, \dots, \hat{\beta}_{|\mathcal{F}|}\}$$

$$g(z) = \sum_{n=0}^{|\mathcal{F}|-1} \frac{\frac{(-1)^n}{(z-\hat{\beta}_n)}}{\sum_{j=0}^{|\mathcal{F}|-1} \frac{(-1)^j}{(z-\hat{\beta}_j)}} f(u(\hat{\beta}_n))$$

$$f(\mathbf{X}_k) \approx g(\alpha_k)$$

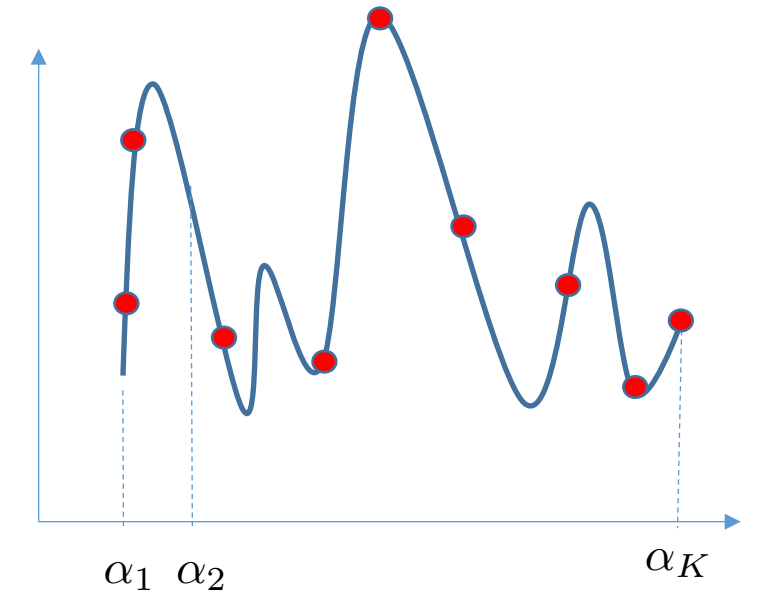
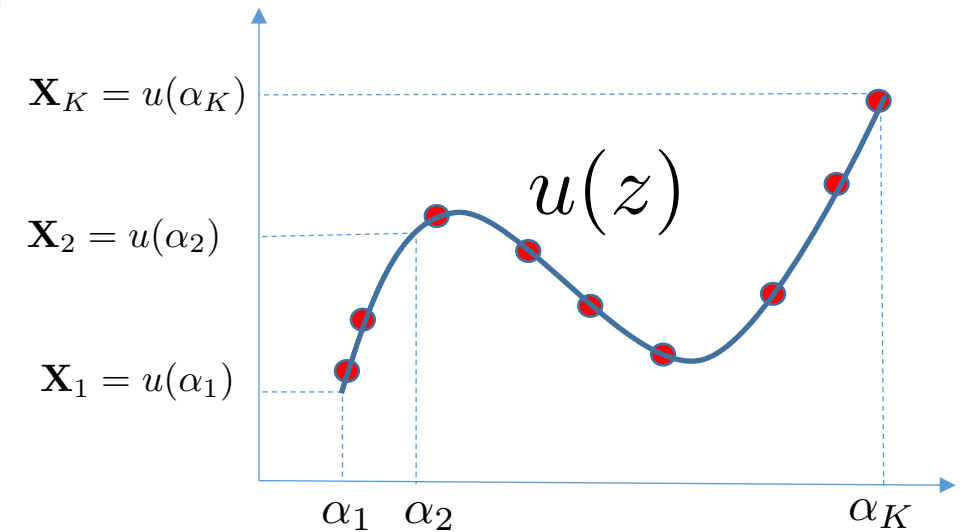


# Berrut Coded Computation

$$\mathcal{F} = \{\hat{\beta}_1, \hat{\beta}_2, \dots, \hat{\beta}_{|\mathcal{F}|}\}$$

$$g(z) = \sum_{n=0}^{|\mathcal{F}|-1} \frac{\frac{(-1)^n}{(z-\hat{\beta}_n)}}{\sum_{j=0}^{|\mathcal{F}|-1} \frac{(-1)^j}{(z-\hat{\beta}_j)}} f(u(\hat{\beta}_n))$$

- ✓ Complexity of Decoding  $\mathcal{O}(|\mathcal{F}|)$
- ✓ The computation is approximate.
- ✓ Works for any number of samples!
- ✓ Works for general functions



# Theoretical Guarantees

## Theorem

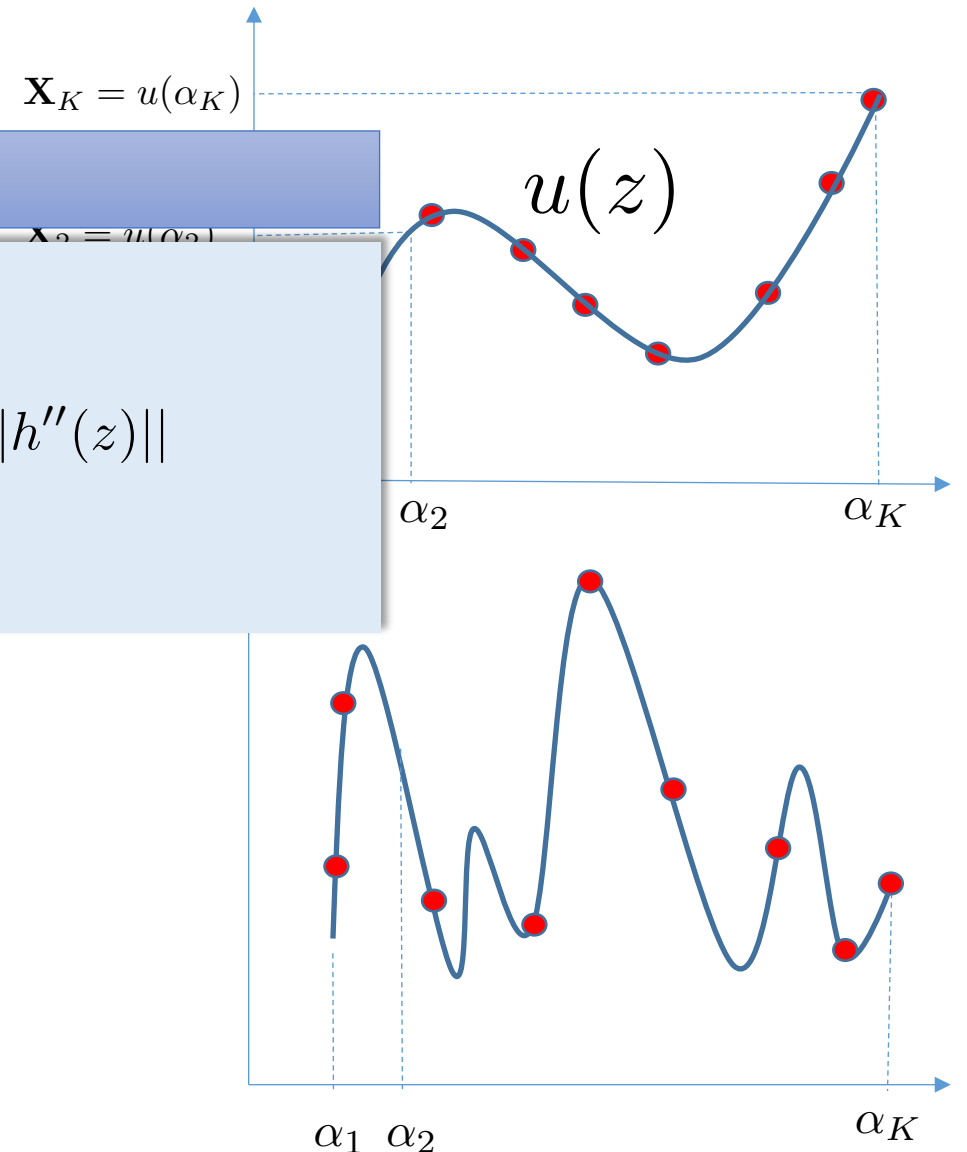
For a system with  $s$  stragglers, we have

$$\|r_{\text{Berrut}}(z) - h(z)\| \leq 2(1 + R) \sin\left(\frac{(s+1)\pi}{2N}\right) \|h''(z)\|$$

where  $R = \frac{(s+1)(s+3)\pi^2}{4}$ .

$$u(z) = \sum_{k=0}^{K-1} \frac{(-1)^k}{\sum_{j=0}^{K-1} \frac{(-1)^j}{(z-\alpha_j)}} \mathbf{X}_k$$

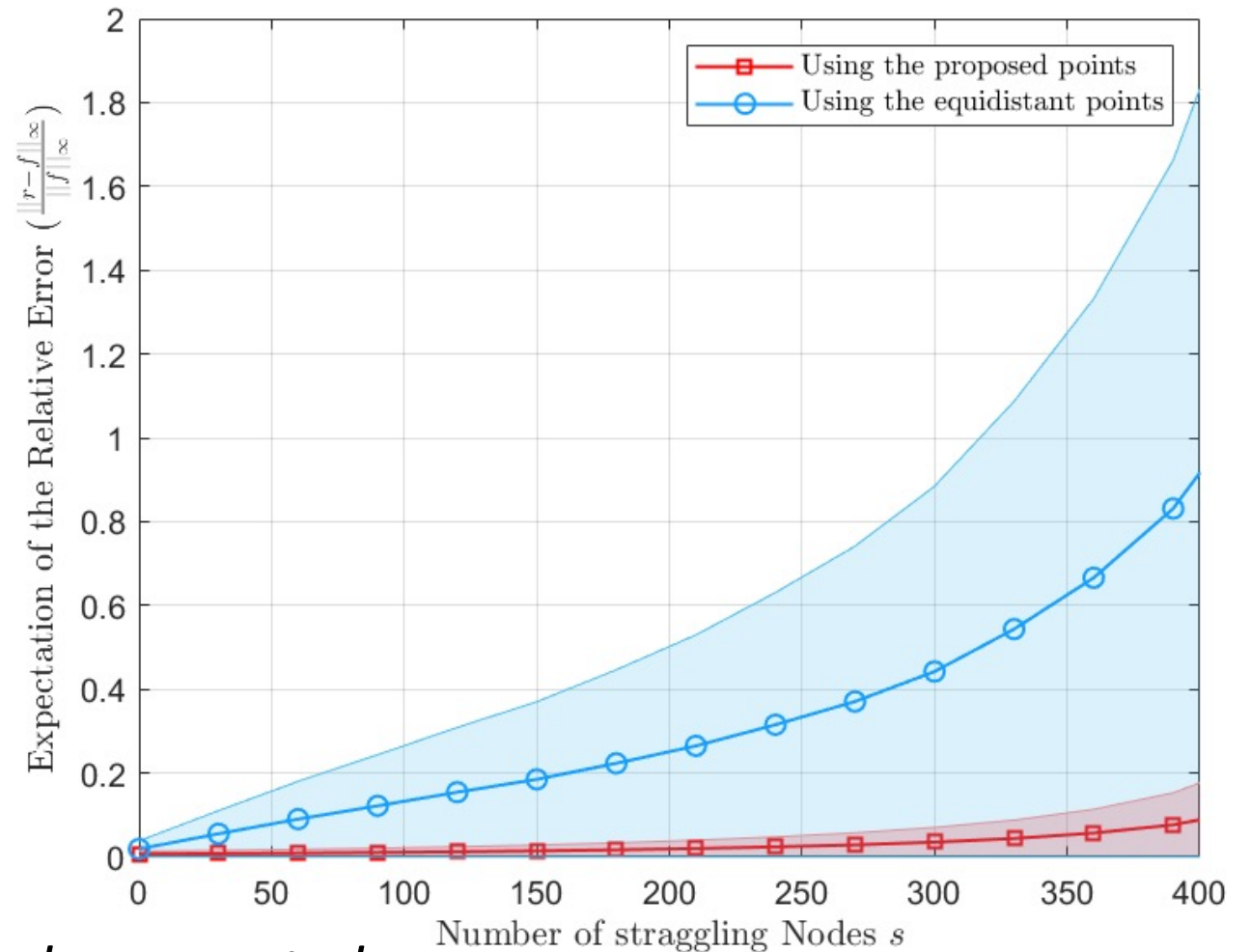
$$h(z) = f(u(z))$$



# Performance

$\deg f = 25, N = 500, K = 20$

Lagrange Coded Computing  
needs 476 non-stragglers  
(Tolerates 24 Stragglers )

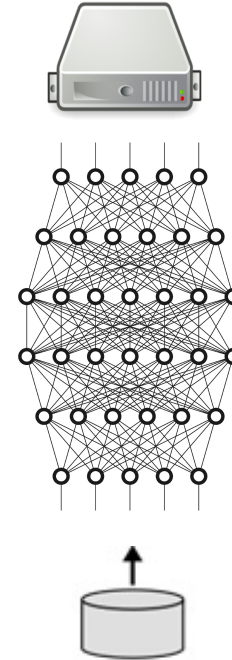


100 polynomial functions  $f$ , *Randomly generated*

# Training a Deep Neural Network: Single Server

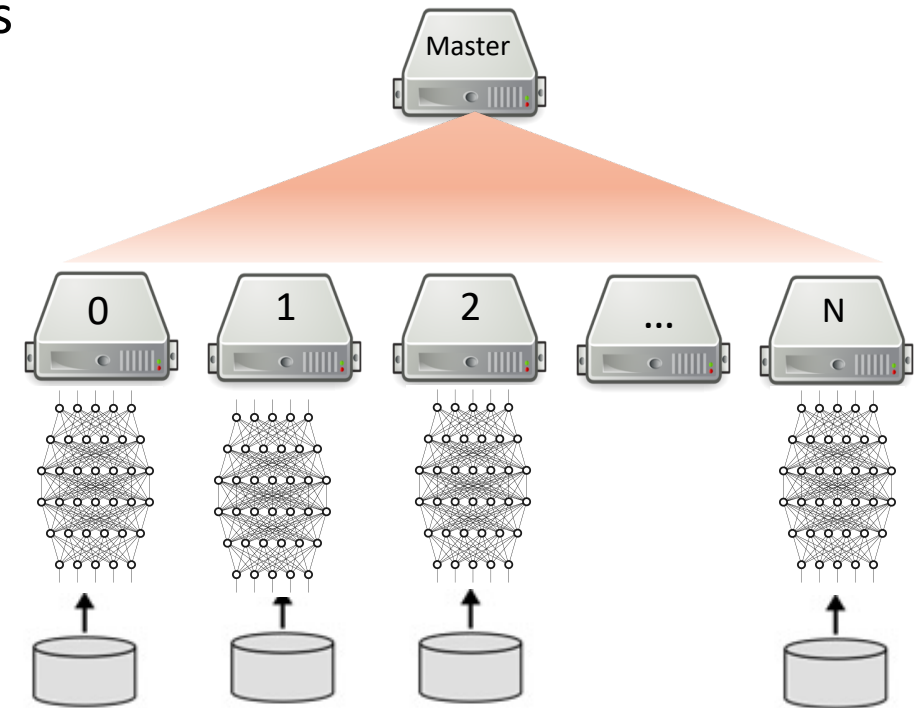
- I. The master node sends mini-batches of the data to the worker node
- II. Worker node computes the gradient based on the parameter of the model with its data samples.
- III. It updates the model using the gradients

**This Computation is heavy!**  
**Let us do parallel processing**



# Training a Deep Neural Network: Parallel Processing

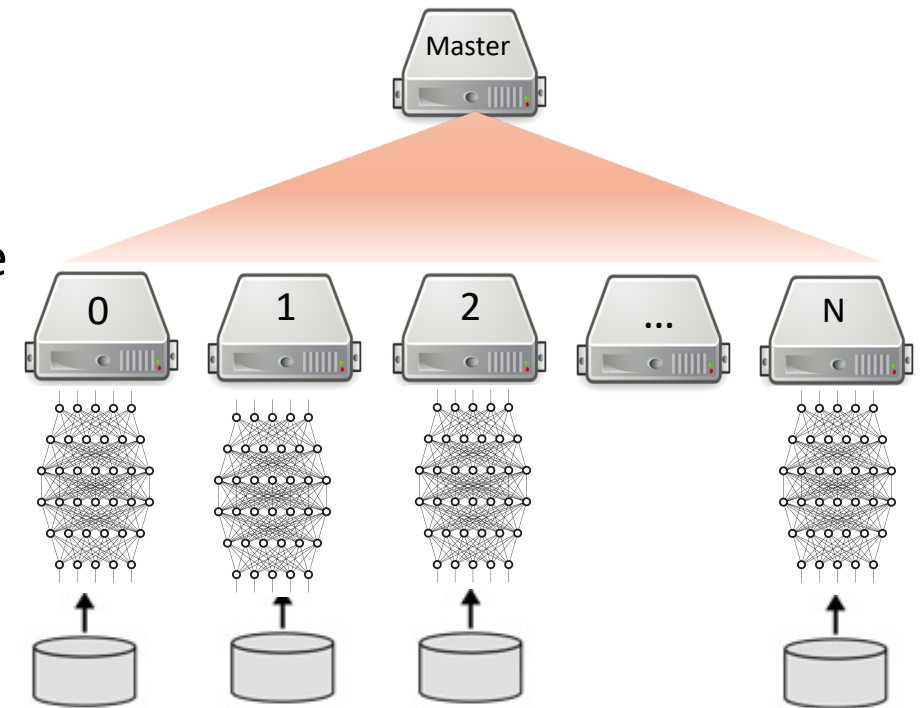
- I. The master node sends independent mini-batches of the data to each worker node
- II. Each worker node computes the gradient based on the parameter of the model with its data samples.
- III. All server nodes the model using the gradients



**Challenge: Stragglers**

# Training a Deep Neural Network: BACC

- I. The master node **encodes mini-batches of the data samples using encoding step of BACC.**
- II. Worker nodes compute the gradient based on the shared parameter with their local coded data samples.
- III. Having received the results from a set of non-straggling worker nodes, **the master node can approximately recover gradients using Berrut's rational interpolant.**



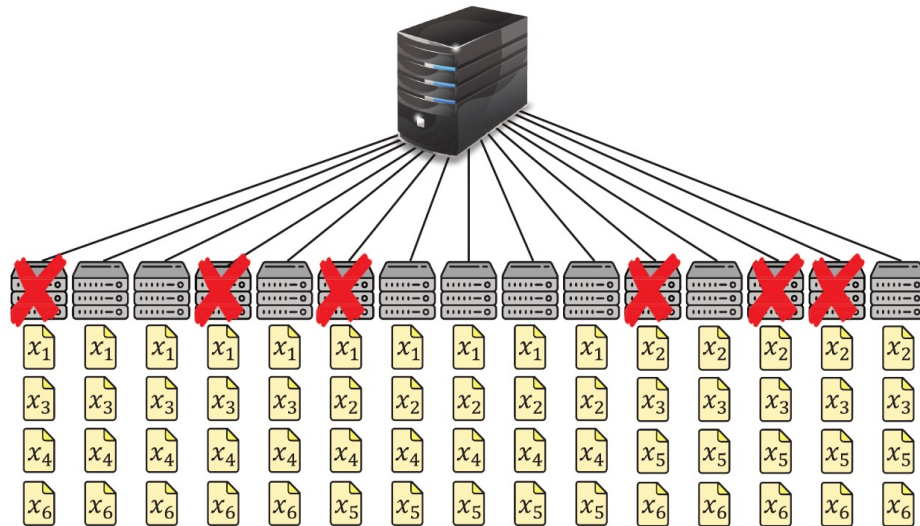


# Training a Deep Neural Network

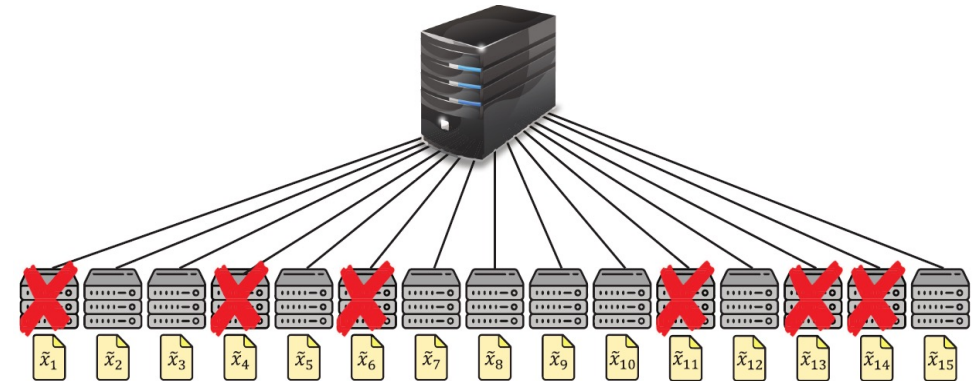
Comparison between two distributed learning approaches:

(a) data replication as a baseline scheme

(b) the proposed scheme

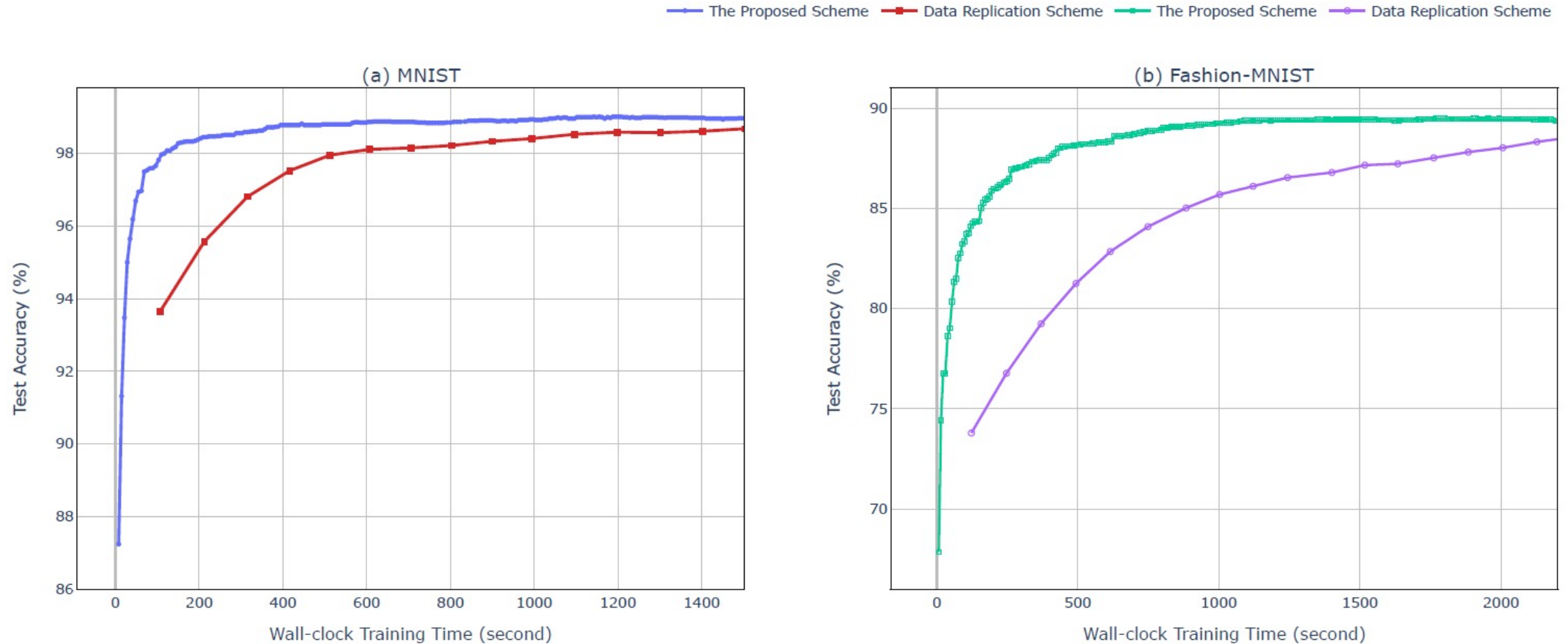


(a)



(b)

# Training a Deep Neural Network



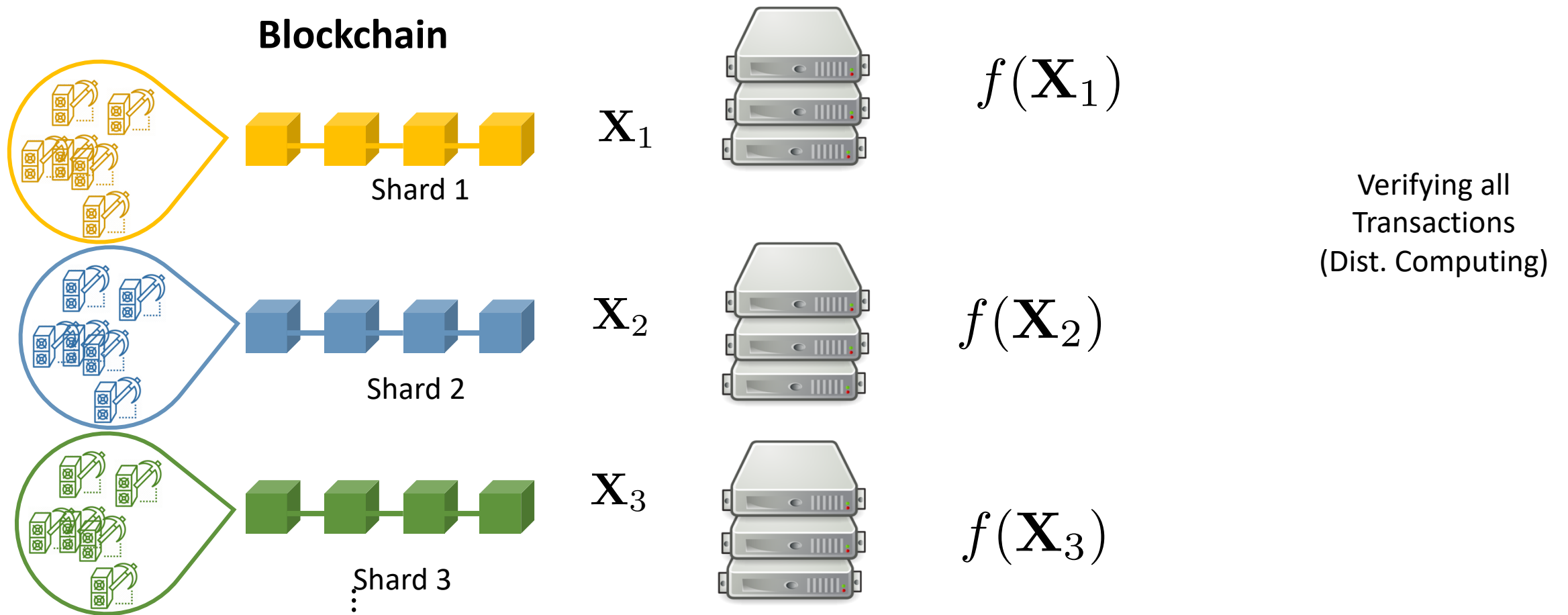
# Open Problems

- Using Approximation Technique for Coded Computing is a Wide Open Area
  - Jahani-Nezhad, Maddah-Ali [2018]
  - Fahim, Cadambe [2019]
  - Jeong, Devulapalli, Cadambe, Calmon [2021]
  - Soleymani, Mahdavifar, Avestimehr [2021]
- Connection with Joint Source-Channel Coding
  - Uncoded Coded!
- Security/Privacy
- Federated Learning

# Distributed Encoding

Abadi, Maddah-Ali [2021]

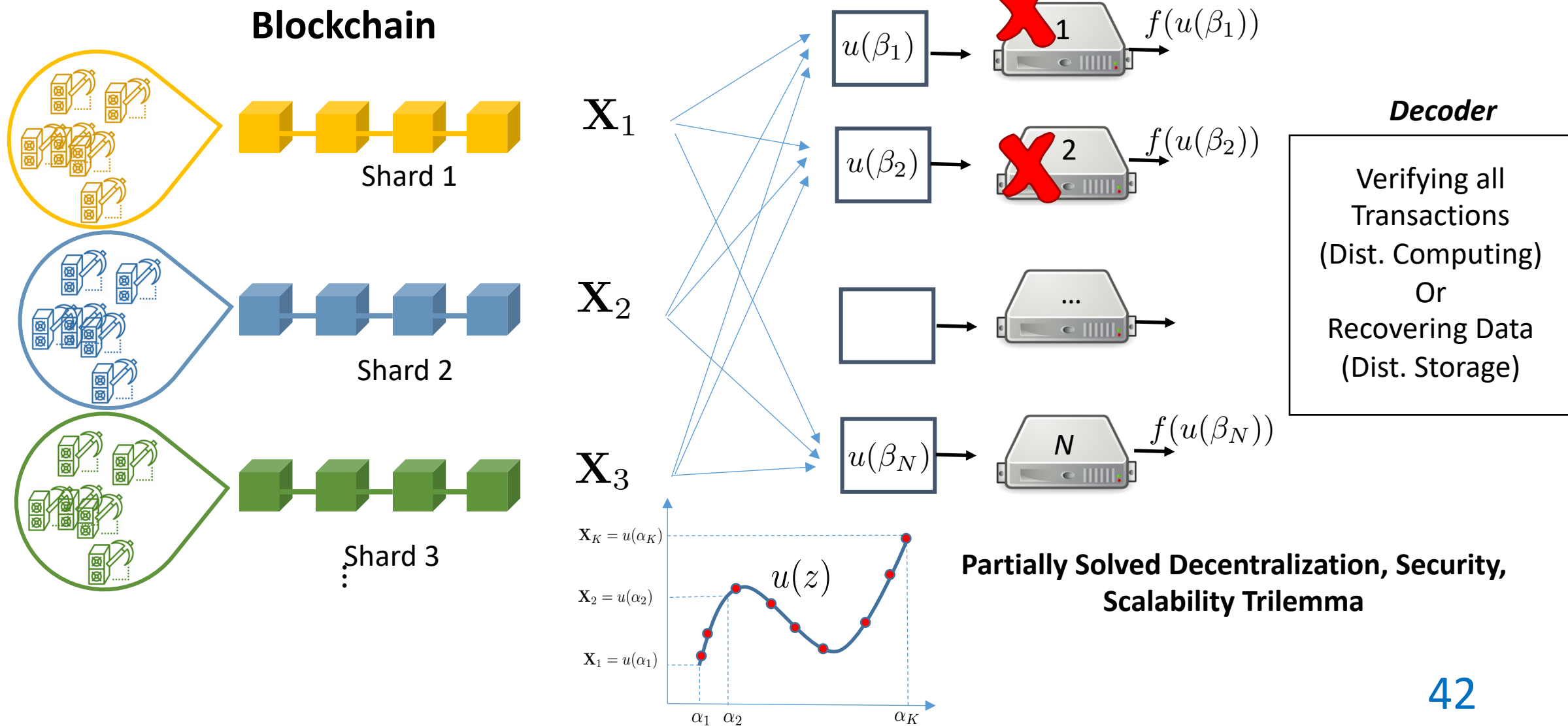
# Sharding in Blockchain



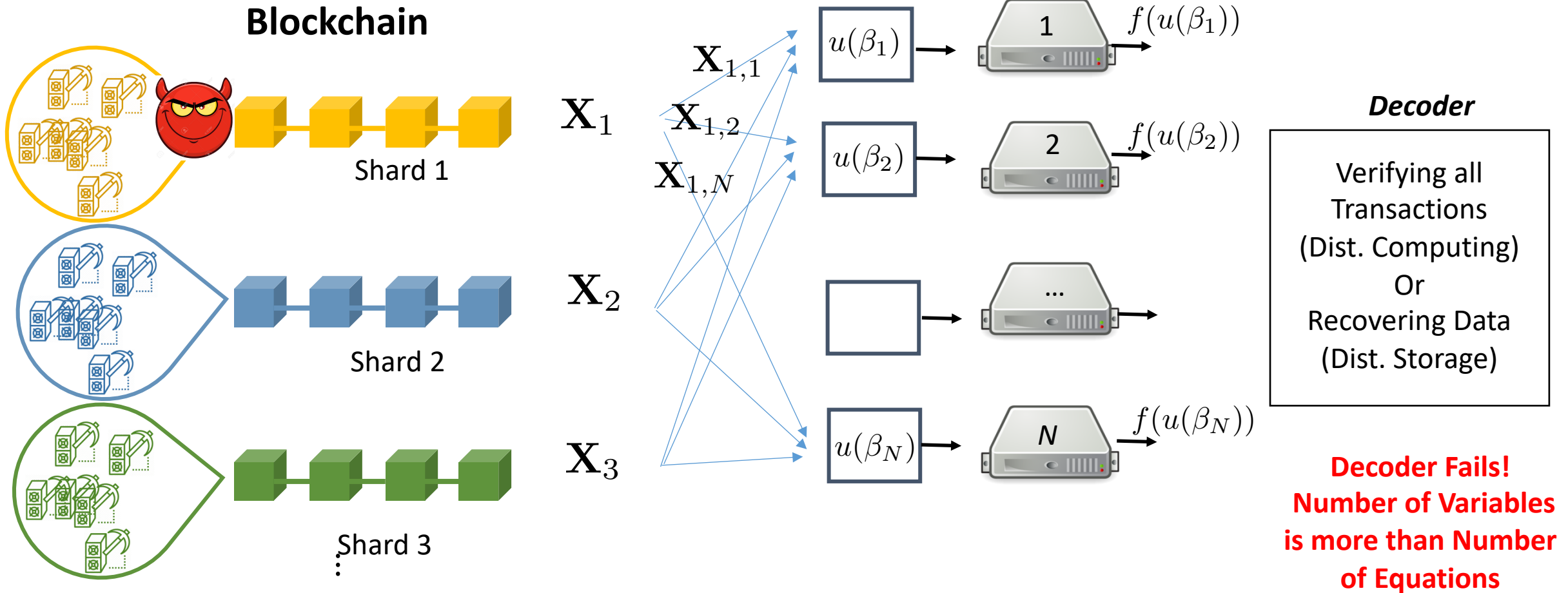
**Not Very Secure!**

# PolyShard in Blockchain

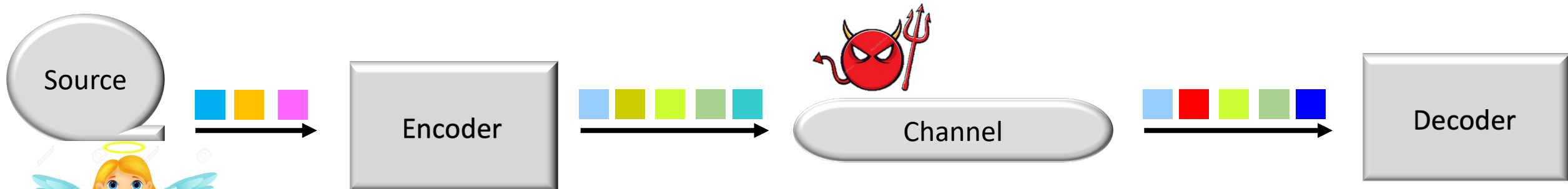
Li, Yu, Yang, Avestimehr, Kannan and Viswanath [2020]



# Challenge of Distributed Encoding



# Classical Coding



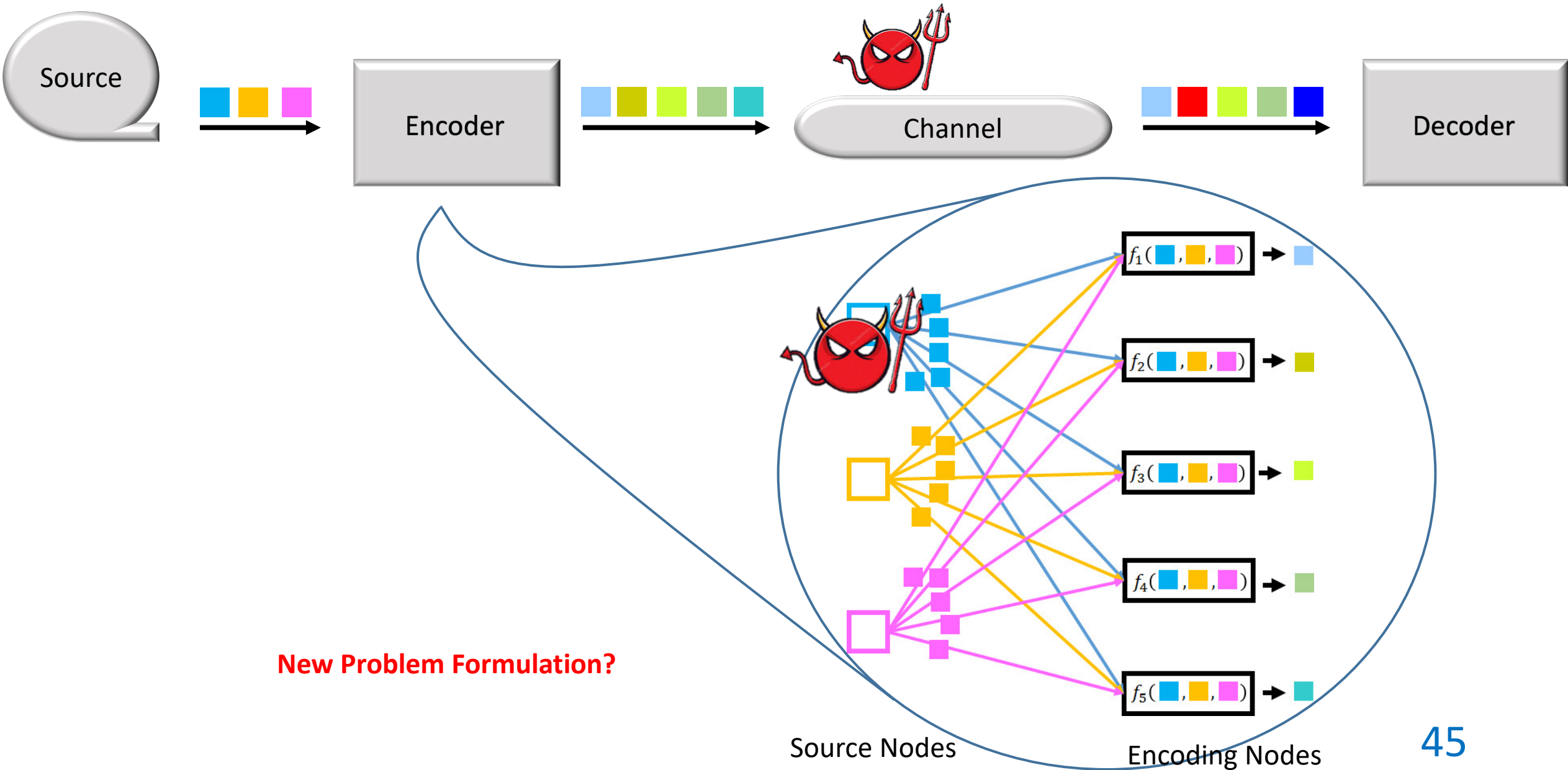
**Shannon approach**  
Probabilistic errors



**Hamming approach**  
Adversarial errors





# Distributed Encoding



# Distributed Encoding: Problem Formulation

1- Adversaries Collaborates

2- No information about  and .

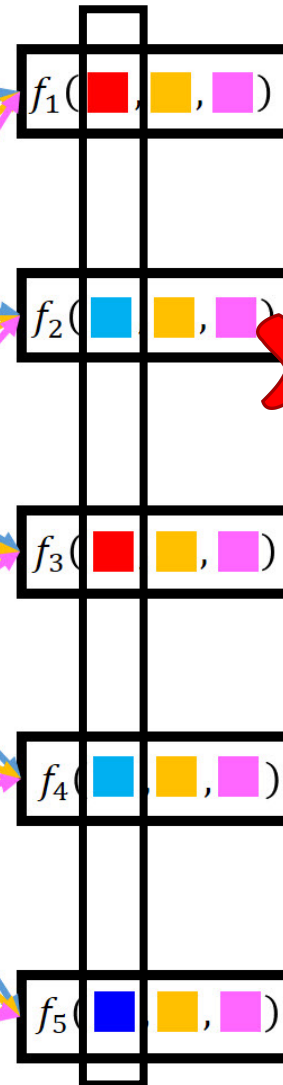
3- Adversary can produce a limited number of messages

Source node 1

Source node 2

Source node 3

Encoding Nodes



3- No information about the adversaries and their behavior.

Decoder

4- The decoder wants to decode the messages of **the honest nodes** correctly.

5- Decoder doesn't care to **recover the messages of adversaries or even detect the adversaries.**

**Minimize the number of encoding nodes needed by the encoder to do its job** 46

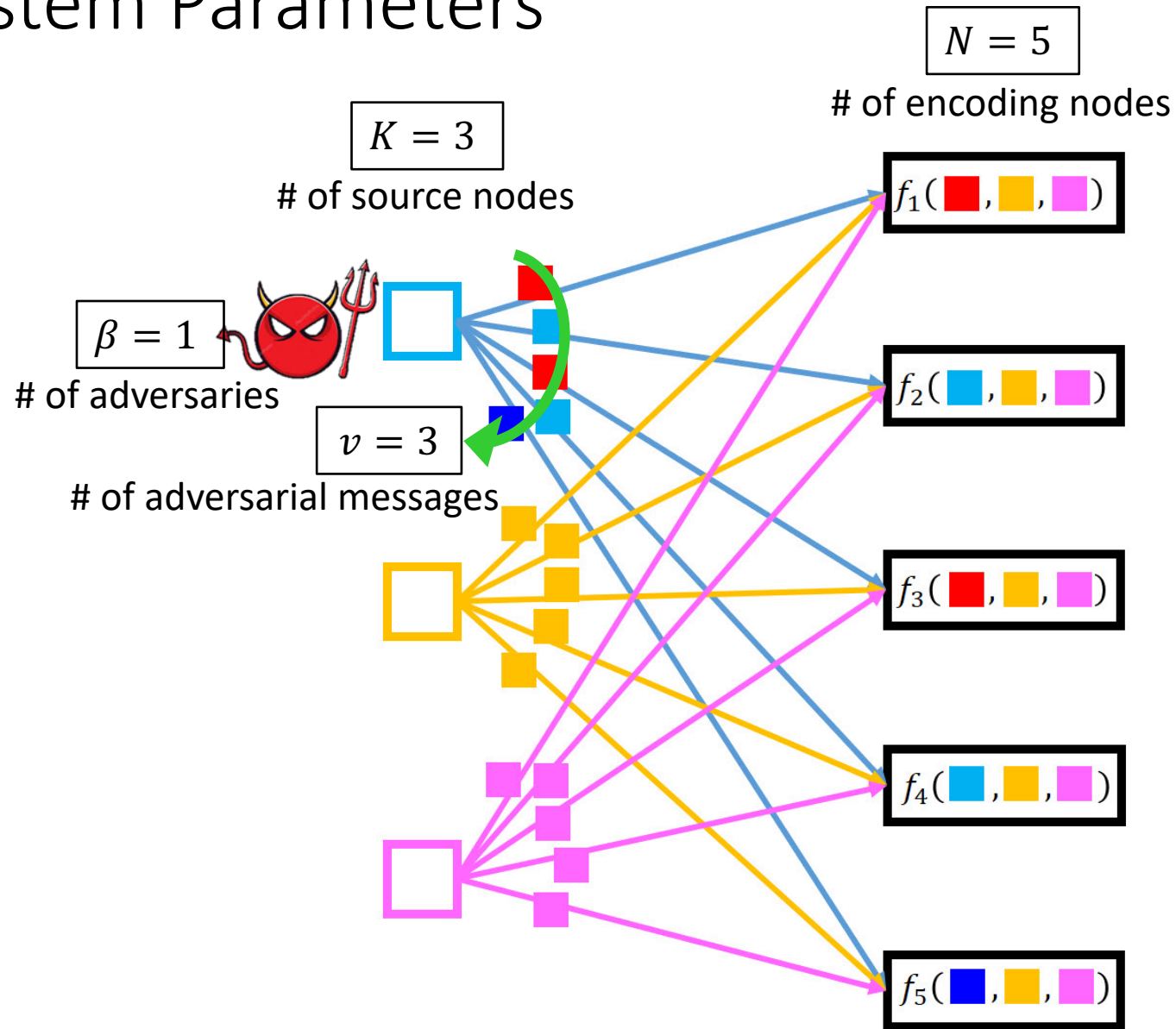


I'm not a robot



reCAPTCHA  
Privacy - Terms

# System Parameters



$K$ : the number of source nodes

$N$ : the number of encoding nodes

$\beta$ : the number of adversaries

$v$ : the maximum number of the messages of one adversarial source node

$t$ : the number of encoding nodes that decoder needs to connect to.

# The problem

To characterize  $t^*$ , as the minimum of  $t$  in an  $(N, K, \beta, v)$  distributed encoding system.

(Informally, at least how many encoding nodes does the decoder need?)

Design Parameters:

1. Encoding Functions
2. Decoding Algorithm

# Fundamental limit of $t$

## Theorem

Abadi, Maddah-Ali [2020]

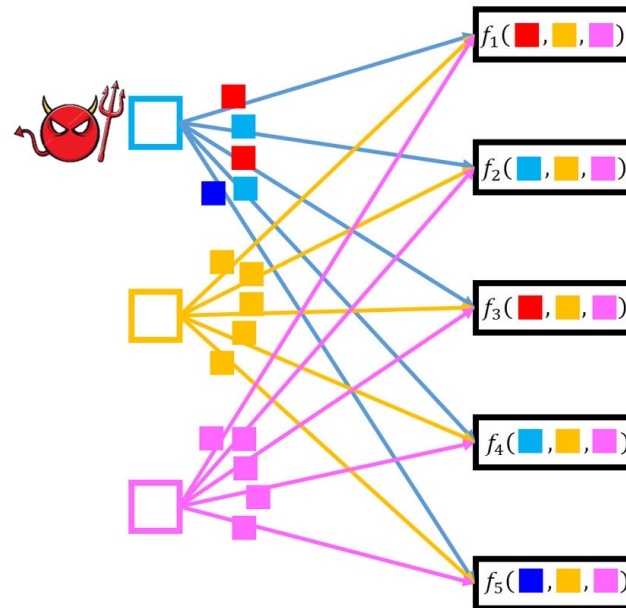
In an  $(N, K, \beta, v)$  distributed encoding system, with linear encoding function

- if  $N \geq K + 2\beta(v - 1) + 1$

$$t_{\text{linear}}^* = K + 2\beta(v - 1)$$

- if  $K \leq N \leq K + 2\beta(v - 1)$

$$t_{\text{linear}}^* = N$$



$\beta$ : the number of adversaries

$v$ : the maximum number of the messages of one adversarial source node

$K$ : the number of source nodes

$N$ : the number of encoding nodes

$t$ : the number of encoding nodes that decoder needs to connect to.

# Achievable Scheme

**Case:**  $N \geq K + 2\beta(v - 1) + 1$

$x_{nk}$  sent by source  $k$  to encoding node  $n$

**Encoding function:**

$$f_n(x_{n1}, \dots, x_{nK}) = \alpha_{n1}x_{n1} + \dots + \alpha_{nK}x_{nK}, 1 \leq n \leq N$$

$\alpha_{n1}$  are generated randomly from the field

**Decoding Procedure:**

1. Consider every possible scenario for the set of adversaries and how they use their options.
2. Form the corresponding set of linear equations and try solve it.
3. If a case offers a solution, announce it.

# Achievable Scheme

## Lemma

In any feasible solution, formed with  $t^* = K + 2\beta(v - 1)$  coded symbols, the symbols of honest nodes have been recovered correctly.

Proof:

- Consider (1) an arbitrary feasible solution, and (2) the real solution.
- Form two sets of equations by these two and merge them.
- Prove that the gaps of honest symbols are zero.

# Converse

## Lemma

For any choice of coefficients, for  $t < K + 2\beta(v - 1)$  coded symbols, there is a scenario, in which the decoder cannot decode the message of honest nodes correctly and uniquely.

Basically, with techniques such as interference alignment, and carefully choosing the coefficients of the linear codes, we cannot avoid confusing the decoder.



# Open Problems

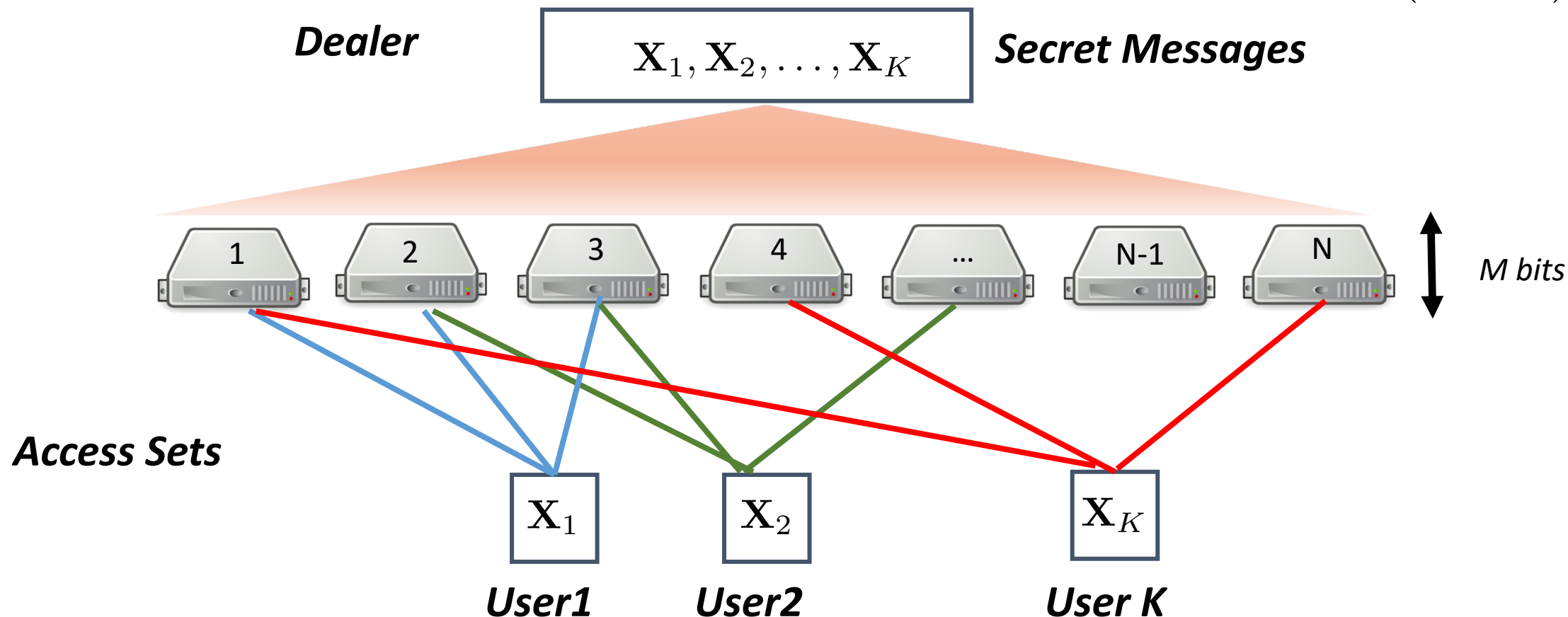
- Can nonlinear code offer gain?
  - If yes, what is the fundamental limits of the distributed encoding?
- For linear codes, can we reduce the complexity of decoding?
- Other types of errors?
- Fundamental limits of distributed coded computing?

# Multuser Secret Sharing

Khalesi, Mirmohseni, Maddah-Ali [2021]

# Multi-User Secret Sharing

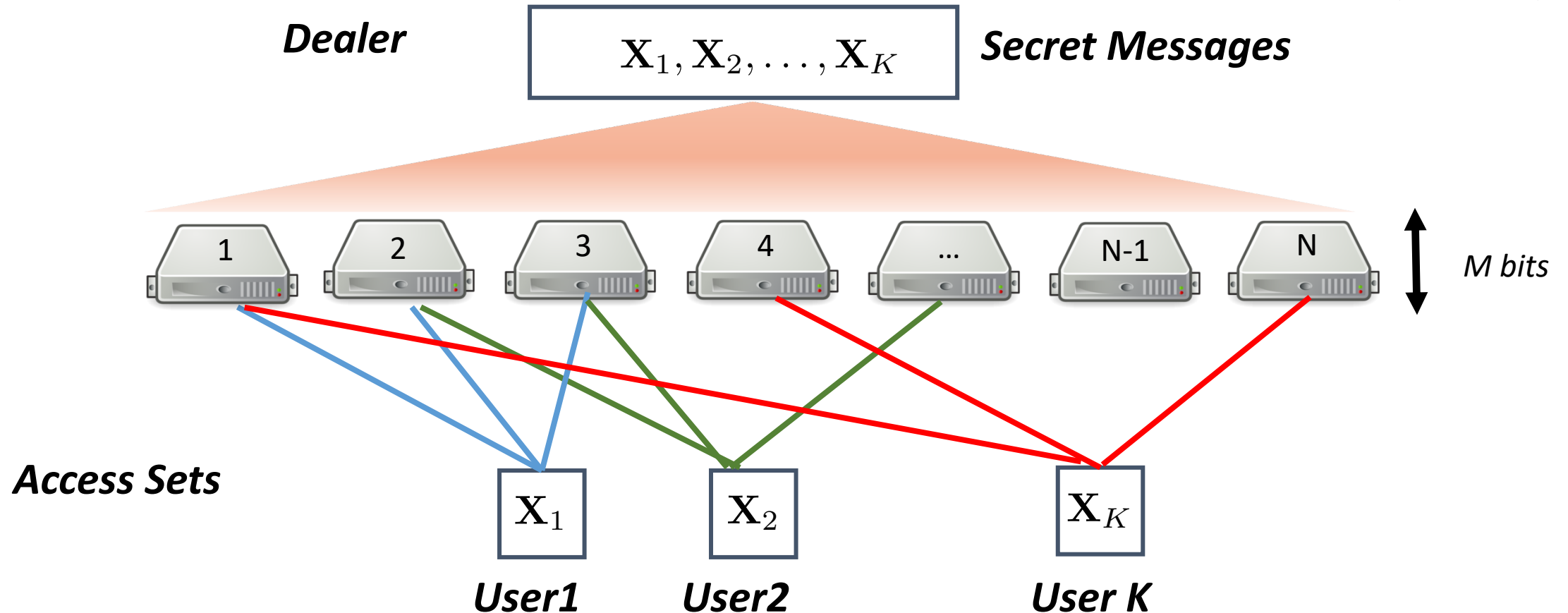
$$\mathbf{X}_k \in GF(2^{MR_k})$$



1. **Correctness:** Each user can recover its own message
2. **Privacy:** Each user learns nothing about the message of other users

# Multi-User Secret Sharing

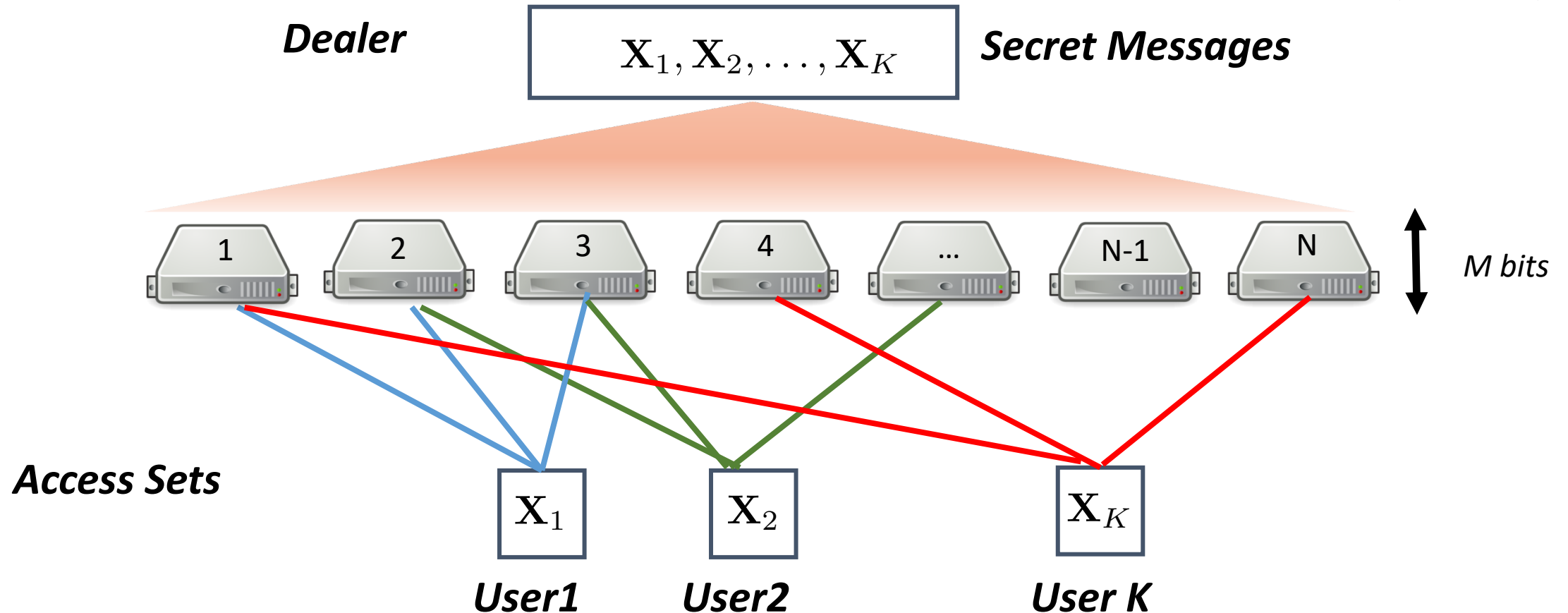
$$\mathbf{X}_k \in GF(2^{MR_k})$$



Objective: To characterize the capacity region of multi-user secret sharing as the set of all possible  $(R_1, R_2, \dots, R_K)$ , subject to correctness and privacy

# Multi-User Secret Sharing

$$\mathbf{X}_k \in GF(2^{MR_k})$$



Introduced by Soleymani and Mahdavifar [2019]

- $R_1 = R_2 = \dots = R_K$
- Specific (Structured) Access Sets

# Main Result

## Theorem

Khalesi, Mirmohseni, Maddah-Ali [2021]

The Capacity Region of Multi-User Secret Sharing Problem is the convex hull of all rate tuples satisfying:

$$R_k \leq \min_{k \neq \tilde{k}} |\mathcal{A}_k \setminus \mathcal{A}_{\tilde{k}}|, \forall k, \tilde{k} \in [K],$$

$$\sum_{i \in \mathcal{S}} R_i \leq |\cup_{i \in \mathcal{S}} \mathcal{A}_i|, \mathcal{S} \subseteq [K].$$

# Multi-User Secret Sharing: Naïve Solution

**Secret Messages**

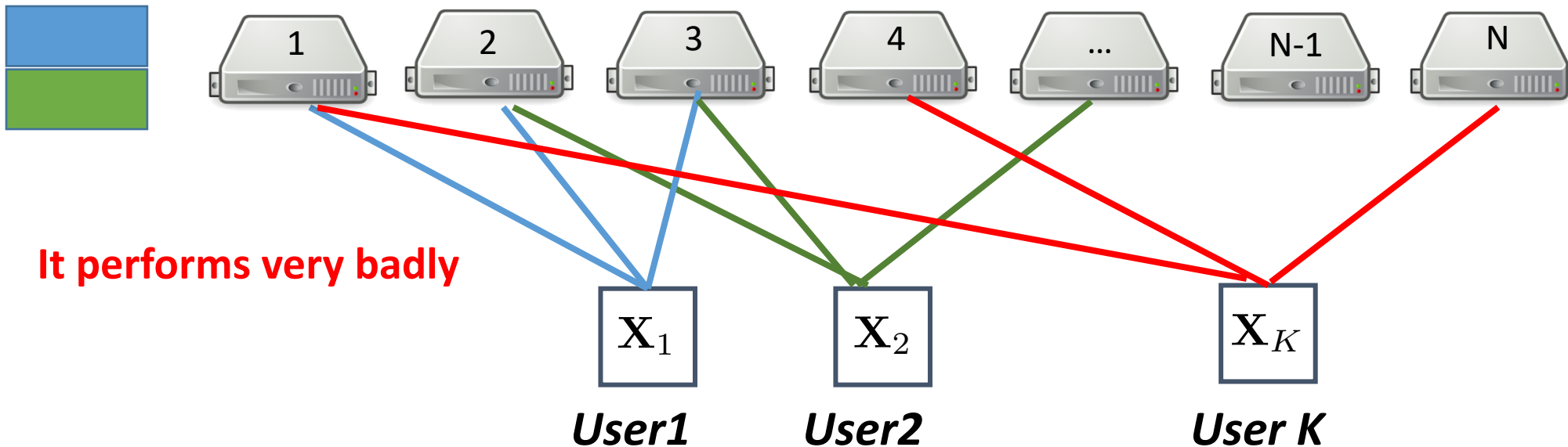
$$\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K$$

Divide each message into  $\mathbf{X}_k = (\mathbf{X}_{k,0}, \dots, \mathbf{X}_{k,r_k-1})$  **Chosen i.i.d**

Form the polynomial  $u_k(z) = \mathbf{X}_{k,0} + \dots + \mathbf{X}_{k,r_k-1}z^{r_k-1} + \mathbf{U}_{k,1}z^{r_k} + \dots + \mathbf{U}_{k,T_k}z^{r_k+T_k}$

If  $n \in \mathcal{A}_k$ , store  $u_k(\beta_n)$ , at server  $n$

$$T_k = \max |\mathcal{A}_k \cap \mathcal{A}_{\tilde{k}}|, \tilde{k} \neq k$$



# Multi-User Secret Sharing: Naïve Solution

**Secret Messages**

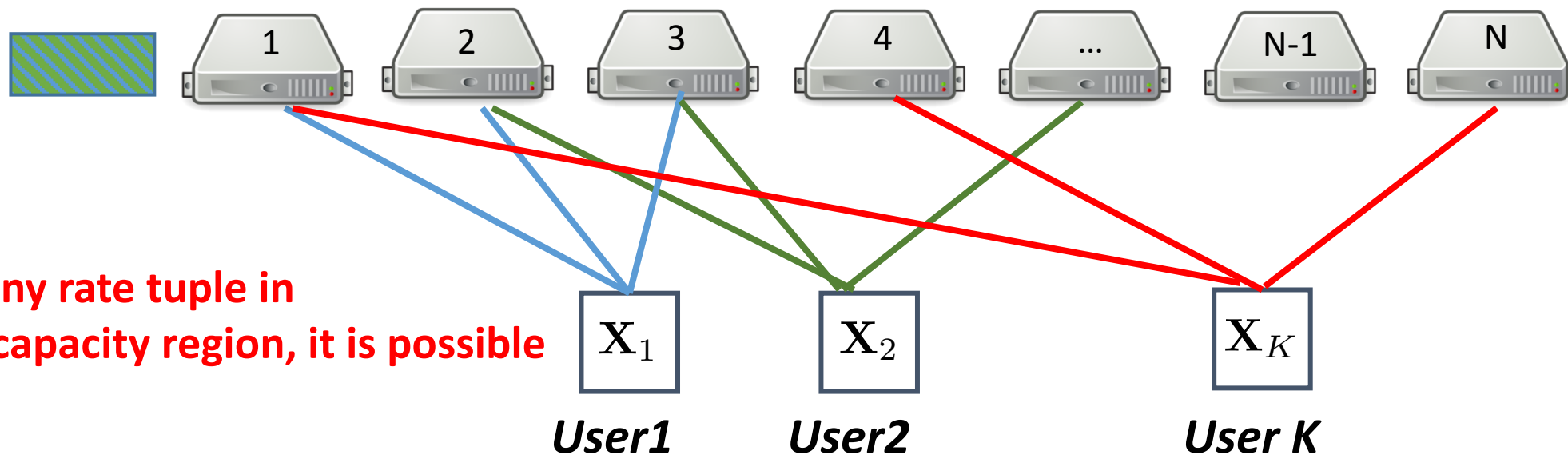
$$\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_K$$

Divide each message into  $\mathbf{X}_k = (\mathbf{X}_{k,0}, \dots, \mathbf{X}_{k,r_k-1})$   $\mathbf{U}_{k,n}$  are solved for such that  $u_i(\beta_n) = u_j(\beta_n)$  for any two users  $i$  and  $j$  connected to server  $n$

Form the polynomial  $u_k(z) = \mathbf{X}_{k,0} + \dots + \mathbf{X}_{k,r_k-1}z^{r_k-1} + \mathbf{U}_{k,1}z^{r_k} + \dots + \mathbf{U}_{k,T_k}z^{r_k+T_k}$

If  $n \in \mathcal{A}_k$ , store  $u_k(\beta_n)$ , at server  $n$

$$T_k = \max |\mathcal{A}_k \cap \mathcal{A}_{\tilde{k}}|, \tilde{k} \neq k$$





# Open Problems

- Connection to Caching, Computing, Storage?
- Fault Tolerance? Adversarial Settings?
- ...

# Conclusion

- Some of the results that we have in distributed systems are motivated and designed based on our conventional approach
- However, considering the new requirements necessitates new techniques which offers significant gains