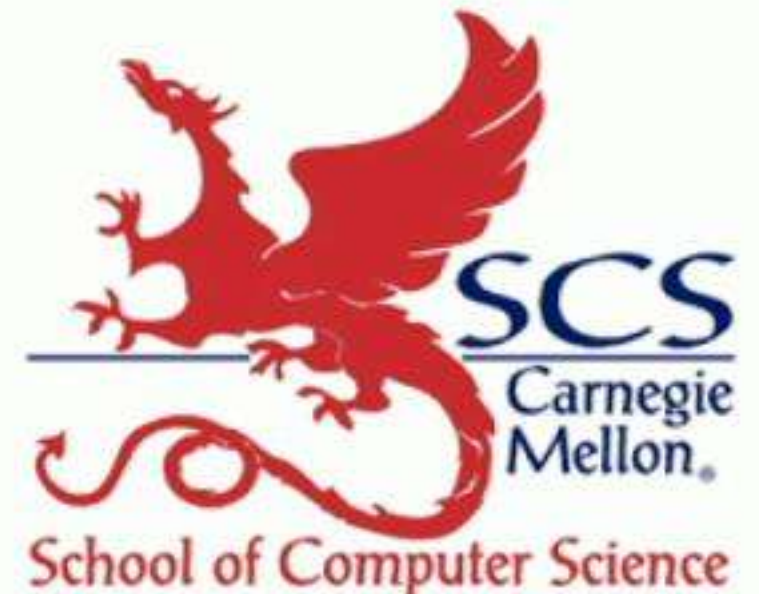


# Resource-efficient redundancy for large-scale data processing and storage systems

**Rashmi Vinayak**

Computer Science

Carnegie Mellon University



# TheSys @ CMU CS

---

**Both theory and systems research**

**Theory for  
systems**

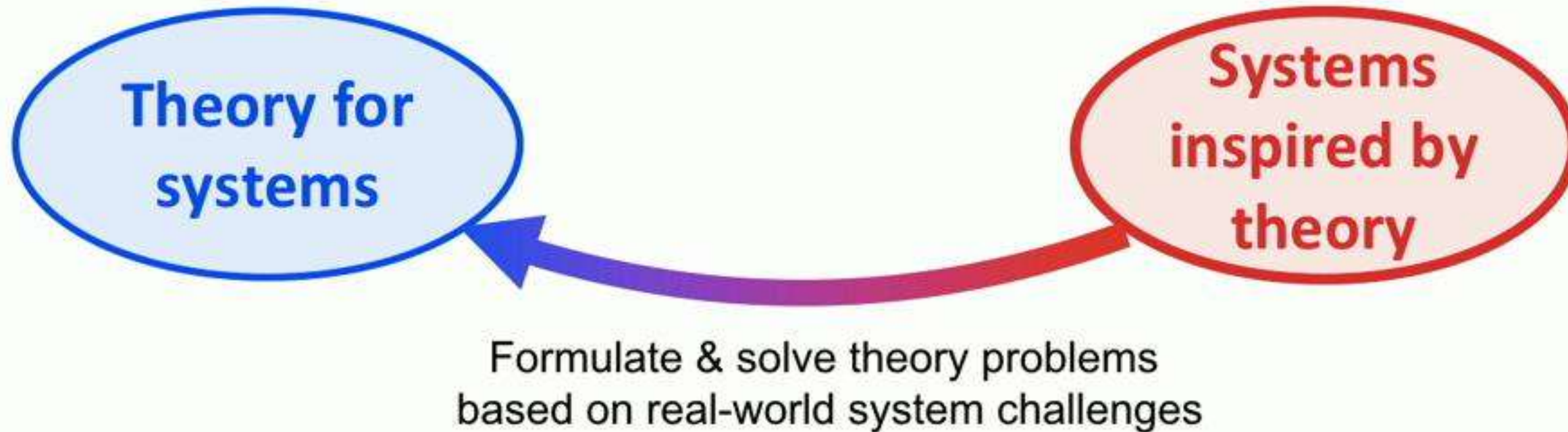
**Systems  
inspired by  
theory**



# TheSys @ CMU CS

---

**Both theory and systems research**

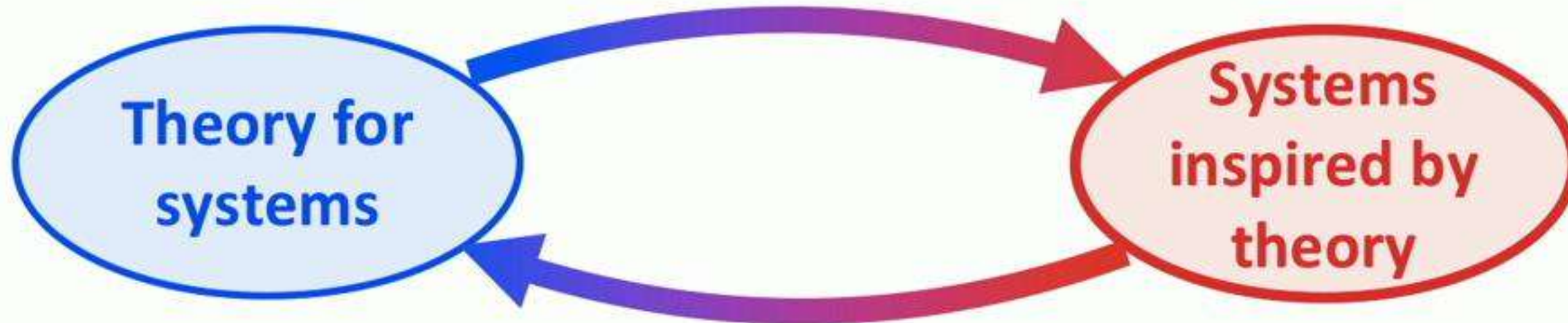


# TheSys @ CMU CS

---

## Both theory and systems research

Using insights & tools from theory  
to build better systems



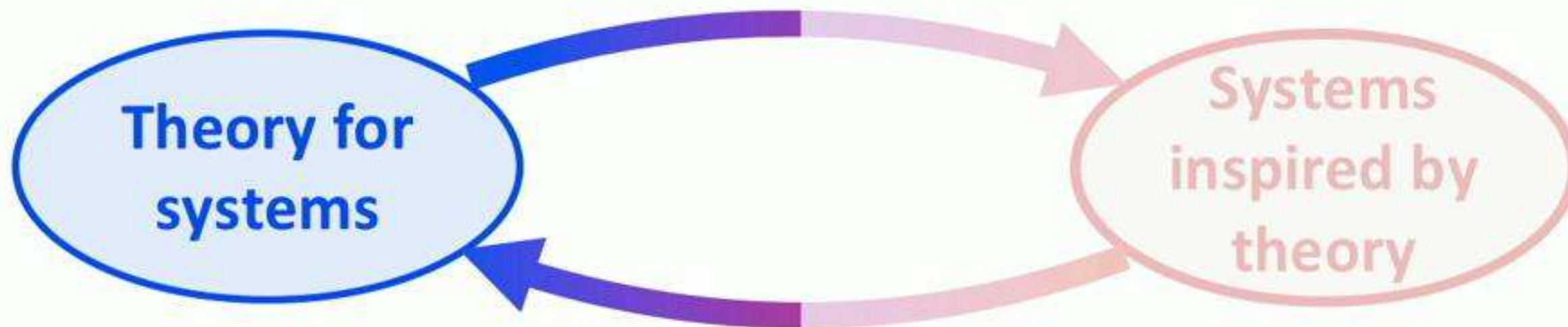
Formulate & solve theory problems  
based on real-world system challenges



# TheSys @ CMU CS

---

**Both theory and systems research**



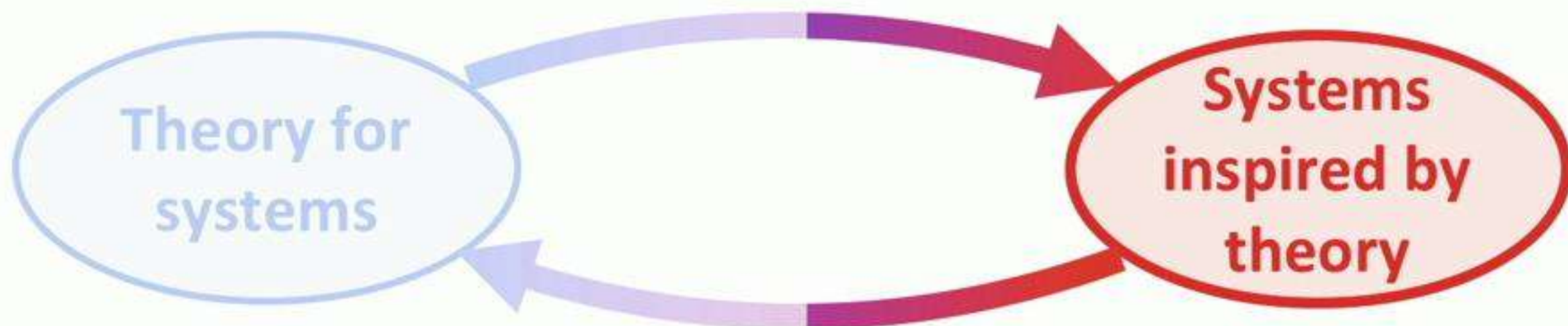
- Information & Coding Theory
  - Erasure codes (aka error-correcting-codes)
- Probability and statistics

# TheSys Lab

---

## Both theory and systems research

Using insights & tools from theory  
to build better systems



Formulate & solve theory problems  
based on real-world system challenges

### Reliability and predictable (tail) performance:

- Distributed storage systems
- Machine learning systems
- Content delivery networks
- Live streaming



# Redundancy in large scale systems

---

- Large scale systems often prone to non-ideal operating conditions
  - Failures, stragglers, load imbalance ...
- **Redundancy: a common approach for resilience**
  - Duplicating queries and/or data
- **Cost of redundancy**
  - Additional hardware resources
  - Increase in cost for equipment, energy, cooling, physical space, operations

# This talk: Resource-efficient redundancy

## Part 1. Data processing systems

### Prediction serving systems

- Serving ML inference

## Part 2. Data storage systems

### Distributed storage systems

- Large-scale cluster storage

Coding theory, machine learning, data analysis, systems insights



# Part 1.

# Resource-efficient Redundancy for Prediction Serving Systems

“Parity Models: Erasure-Coded Resilience for Prediction Serving Systems”

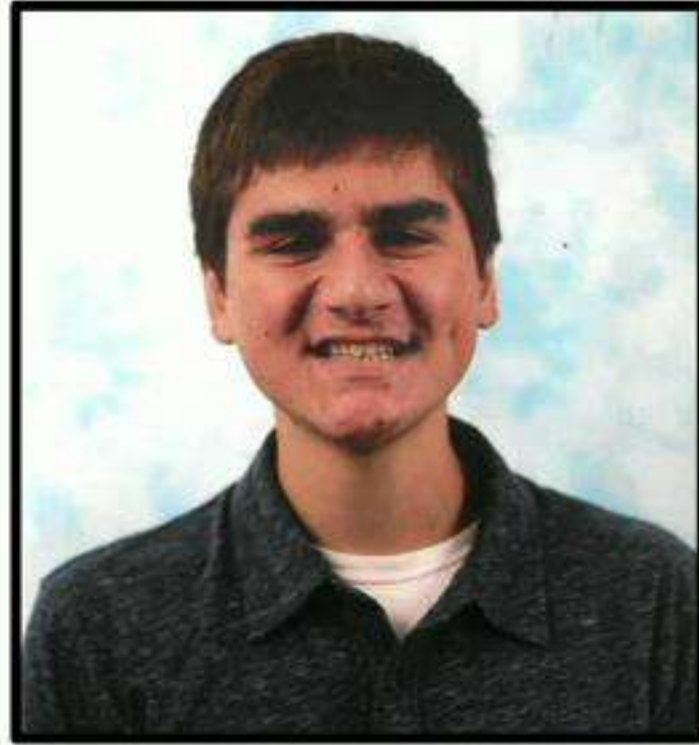
J. Kosaian, K. V. Rashmi, S. Venkataraman

To appear in *ACM SOSP 2019*.

ArXiv preprints: June 2018, May 2019

# Joint work with

---



**Jack Kosaian**

Carnegie Mellon University



**Shivaram Venkataraman**

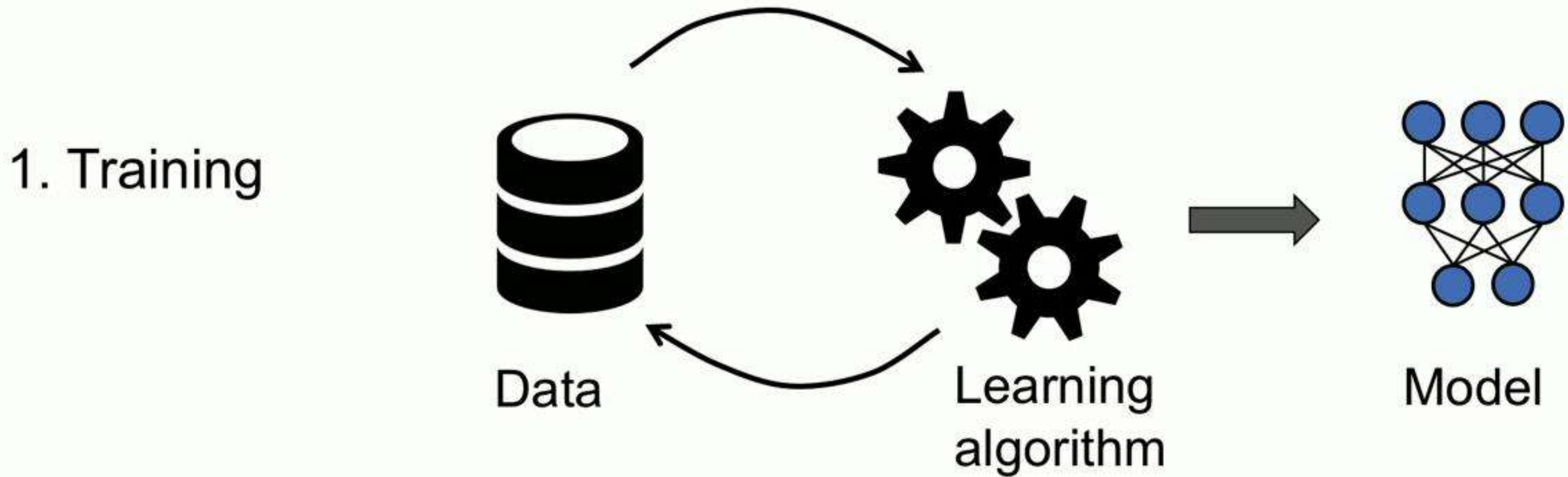
University of  
Wisconsin-Madison



# Machine learning in production systems

---

Two phases: 1. Training 2. Inference

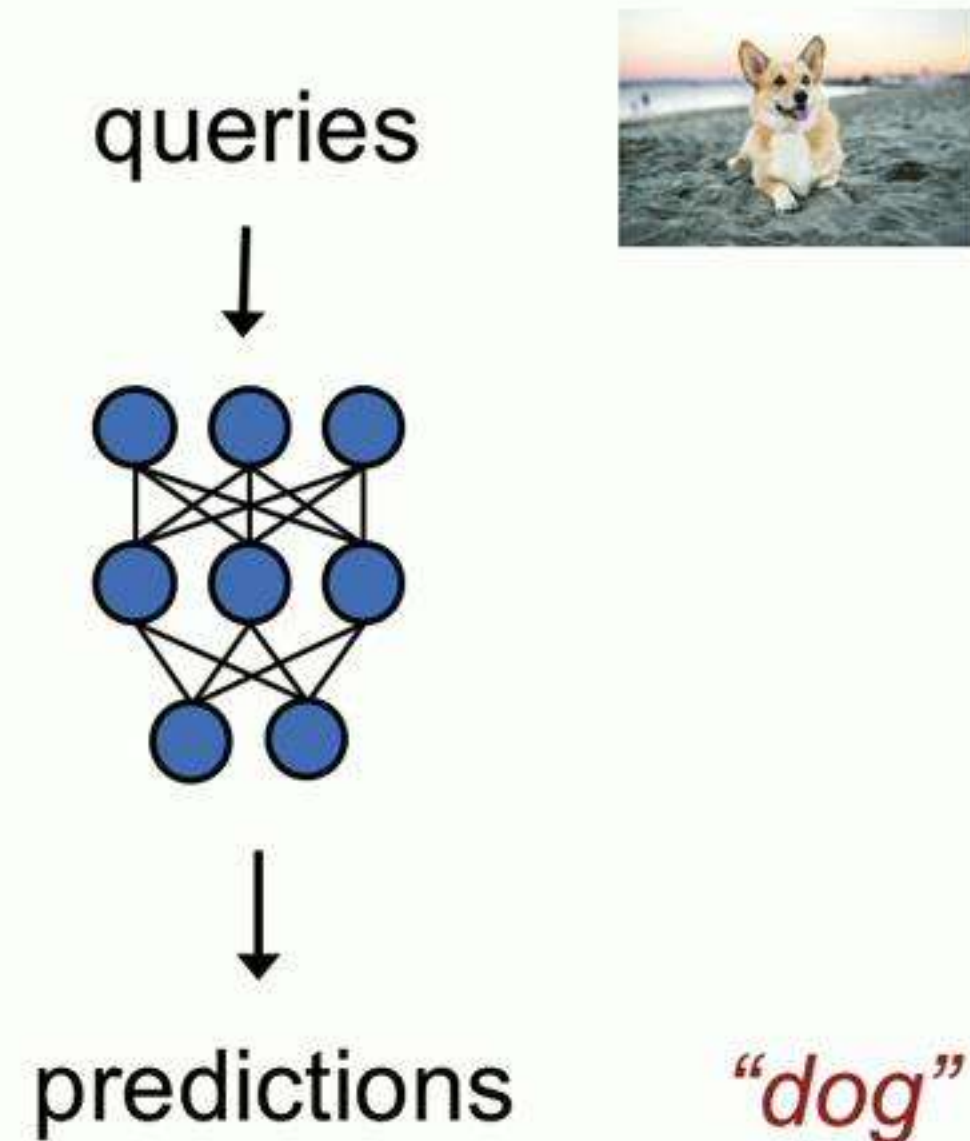


# Machine learning in production systems

---

Two phases: 1. Training 2. Inference

2. Inference





# Prediction serving systems

---

Platforms that deploy ML models and handle inference

## Open Source



Google  
TensorFlow Serving

Amazon The logo for Amazon MXNet, featuring the word "mxnet" in a blue, lowercase, sans-serif font. The 'm' is enclosed in a blue circle.

## Cloud Services



Microsoft  
Azure



Google AI



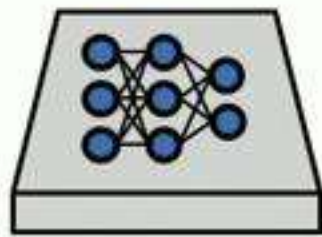
Amazon  
Sagemaker

# Prediction serving system architecture

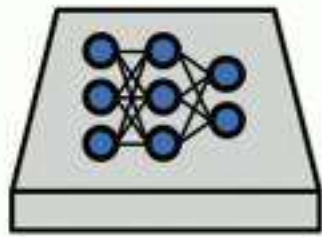
---

Multiple copies of model instances on separate nodes to support high query rates

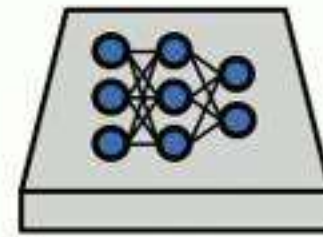
model instances



node 1



node 2



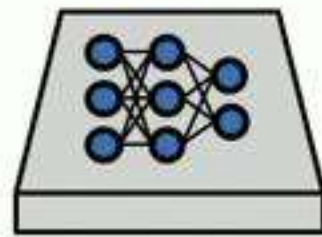


# Prediction serving system architecture

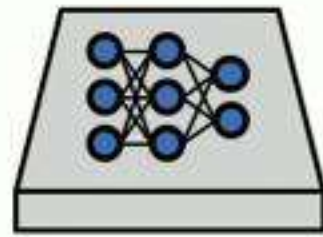
---



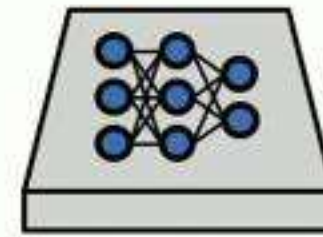
model instances



node 1

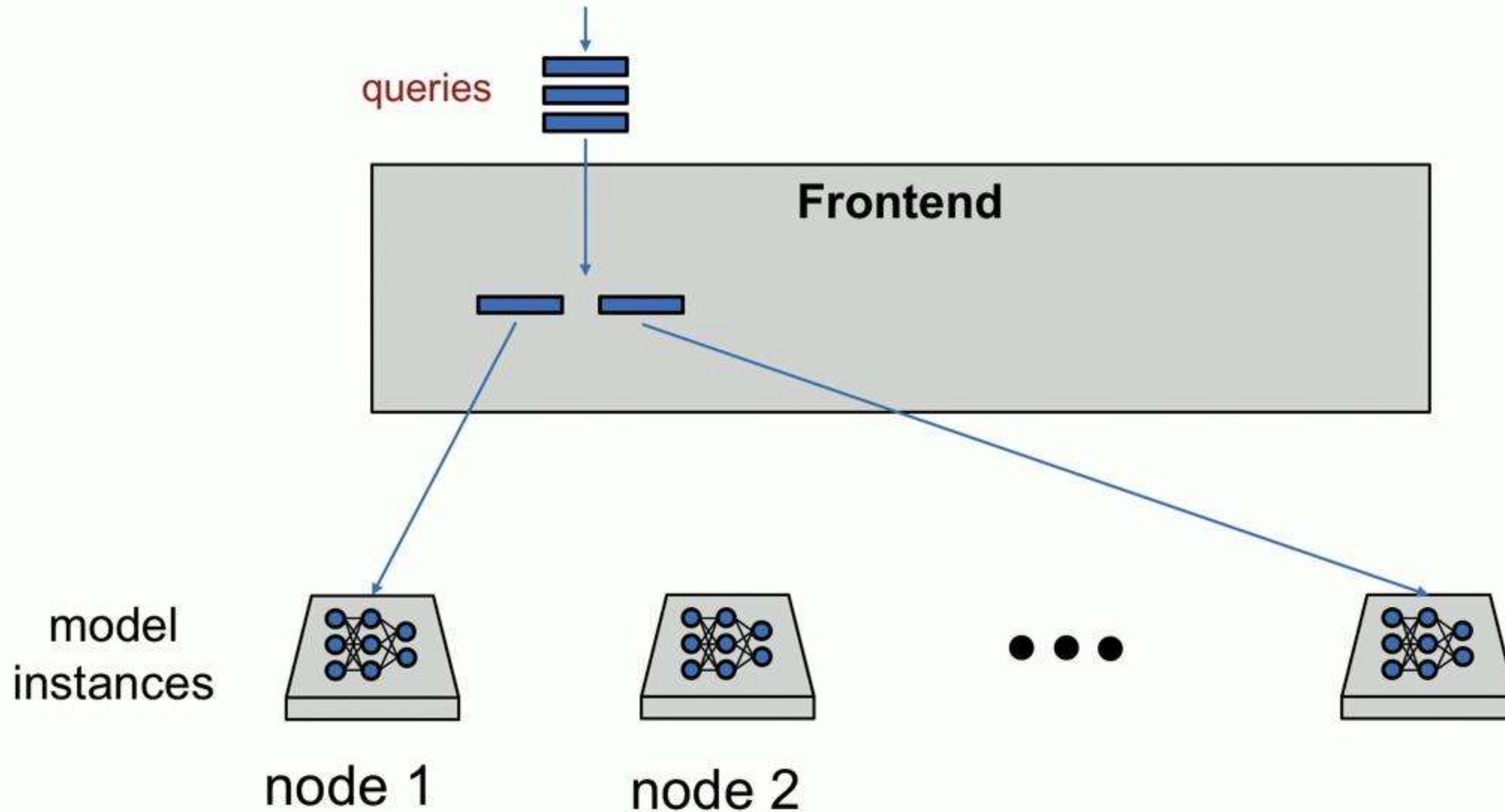


node 2



# Prediction serving system architecture

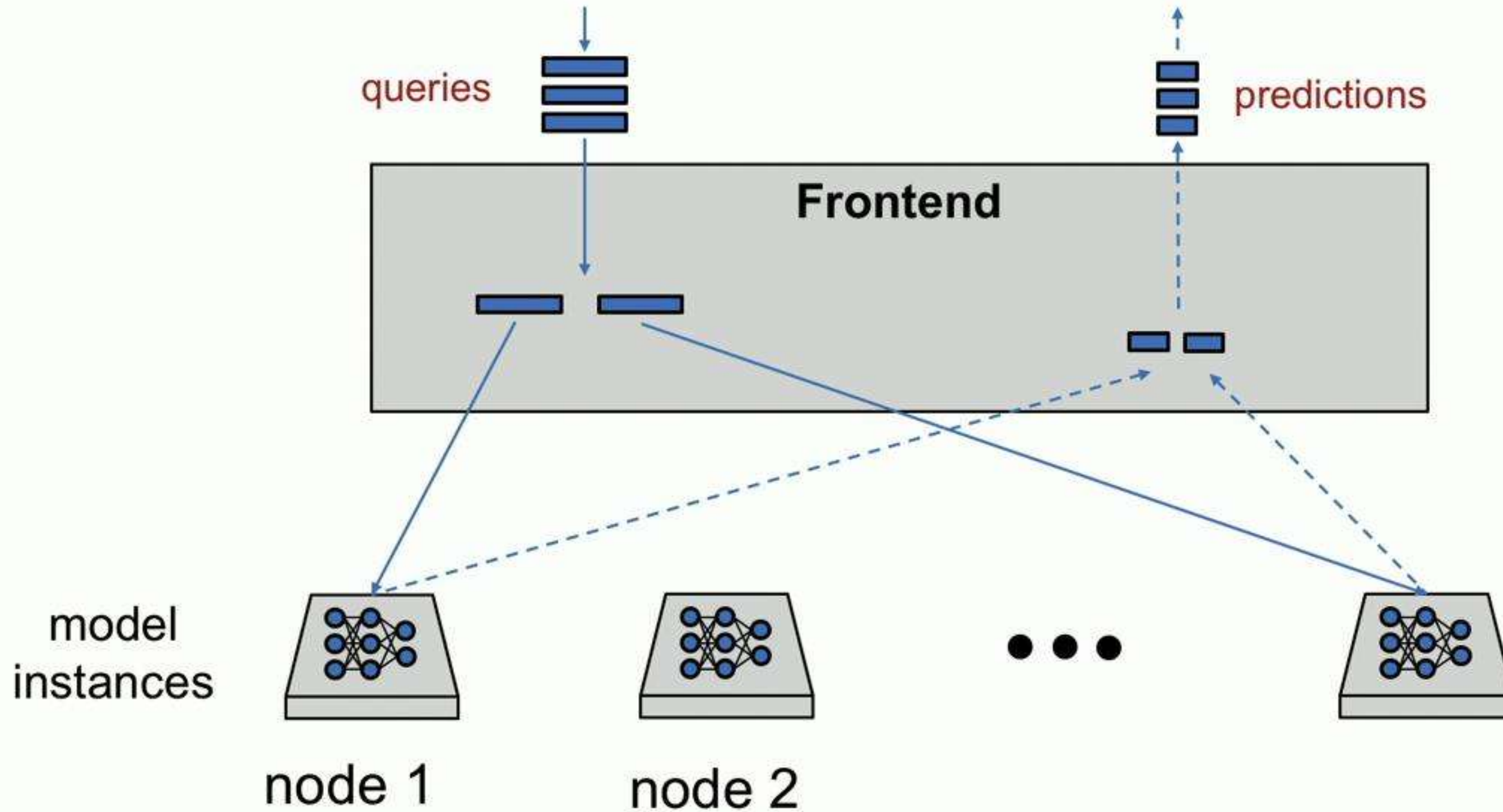
---





# Prediction serving system architecture

---

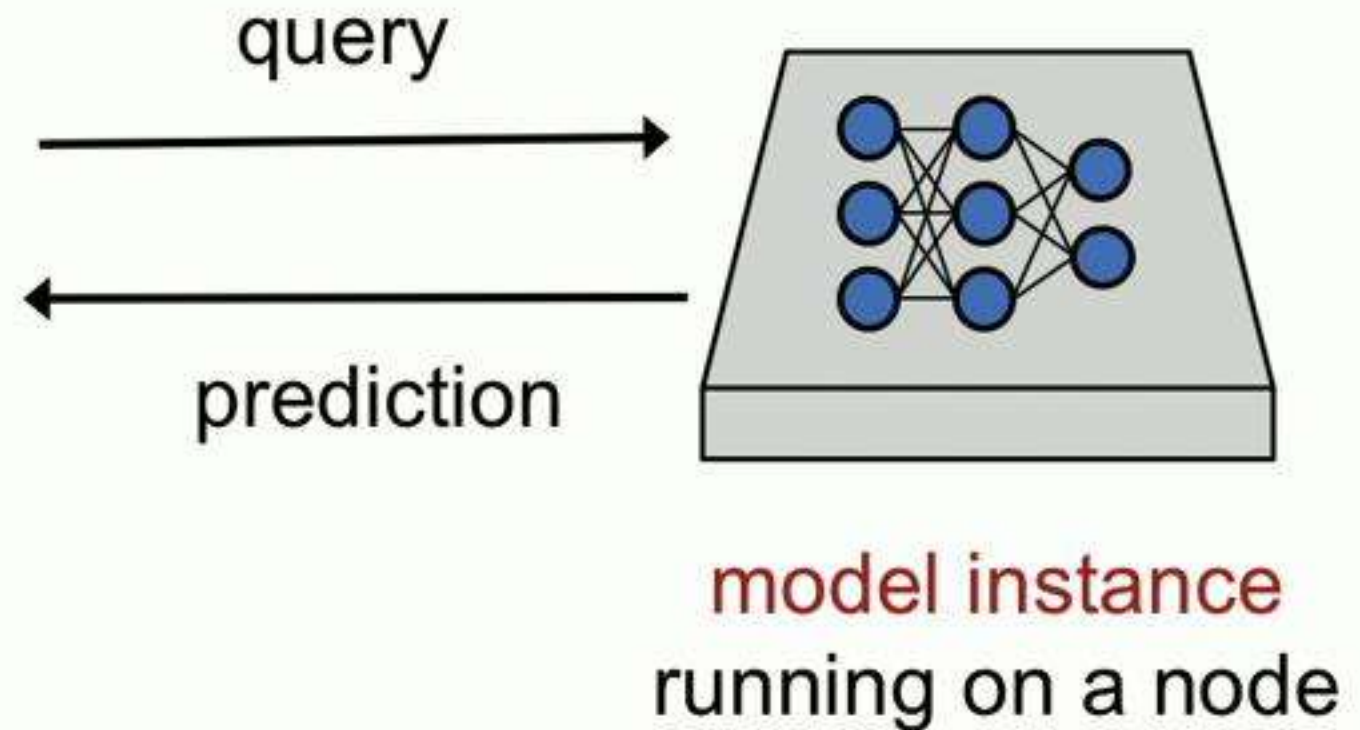


# Prediction serving systems

---

Used as a backend for:

- User-facing applications
- Production services



- Must meet strict service level objectives
- Must operate with low, predictable latency



# Unavailability in prediction serving systems

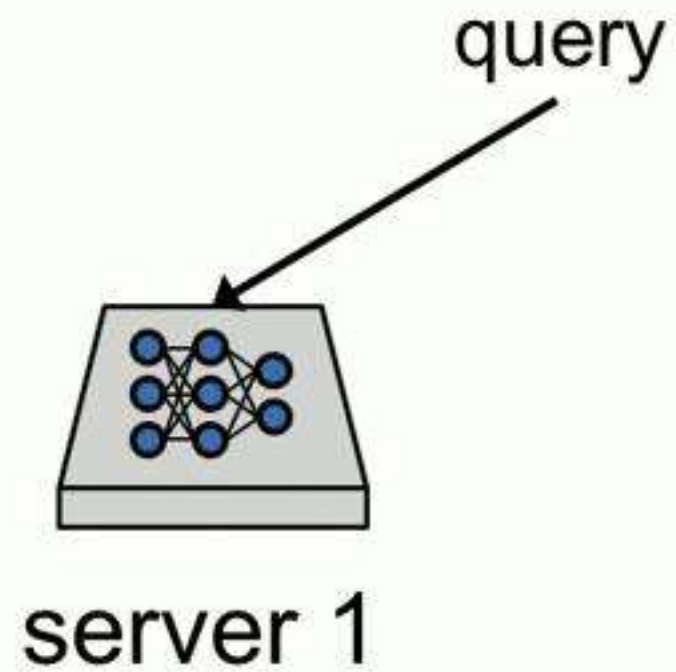
- Slowdowns and failures (“**unavailability**”)
  - Resource contention, HW/SW failures, runtime slowdowns
  - ML-specific events
- Result in **inflated tail latency**
  - Unpredictable latency
  - Miss latency SLAs

Minimizing tail latency is critical

# Solution 1: Reactive

---

- First send to one server
- Wait, retry request

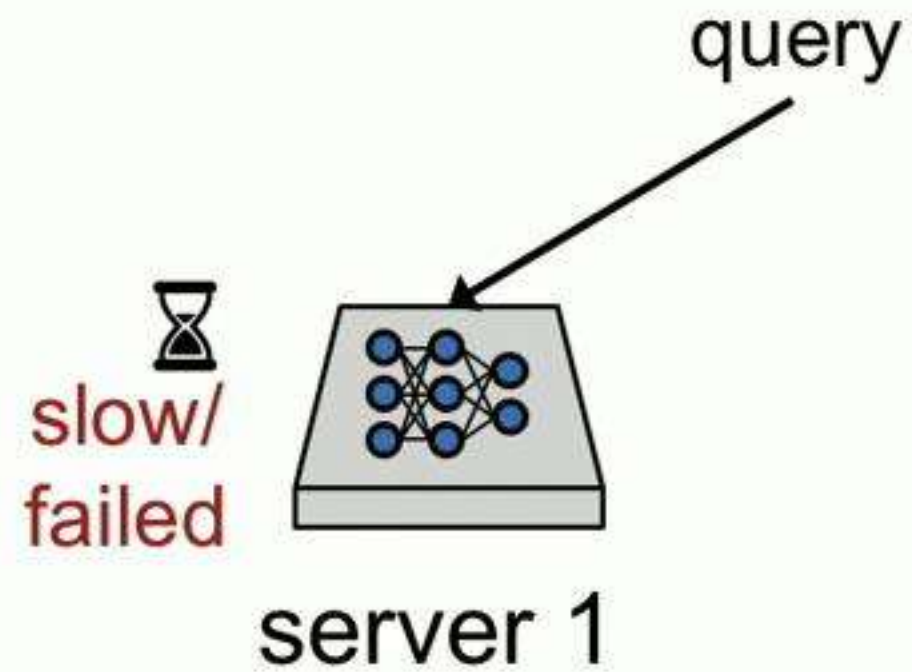




# Solution 1: Reactive

---

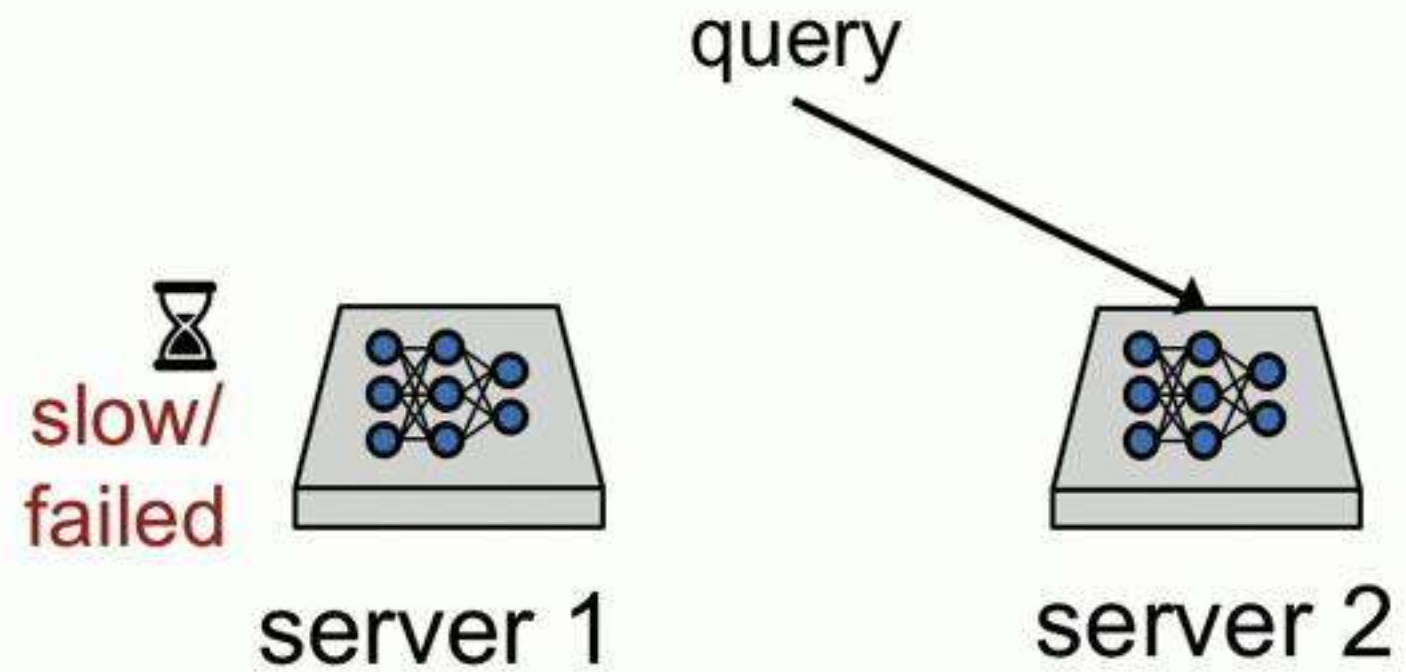
- First send to one server
- Wait, retry request



# Solution 1: Reactive

---

- First send to one server
- Wait, retry request

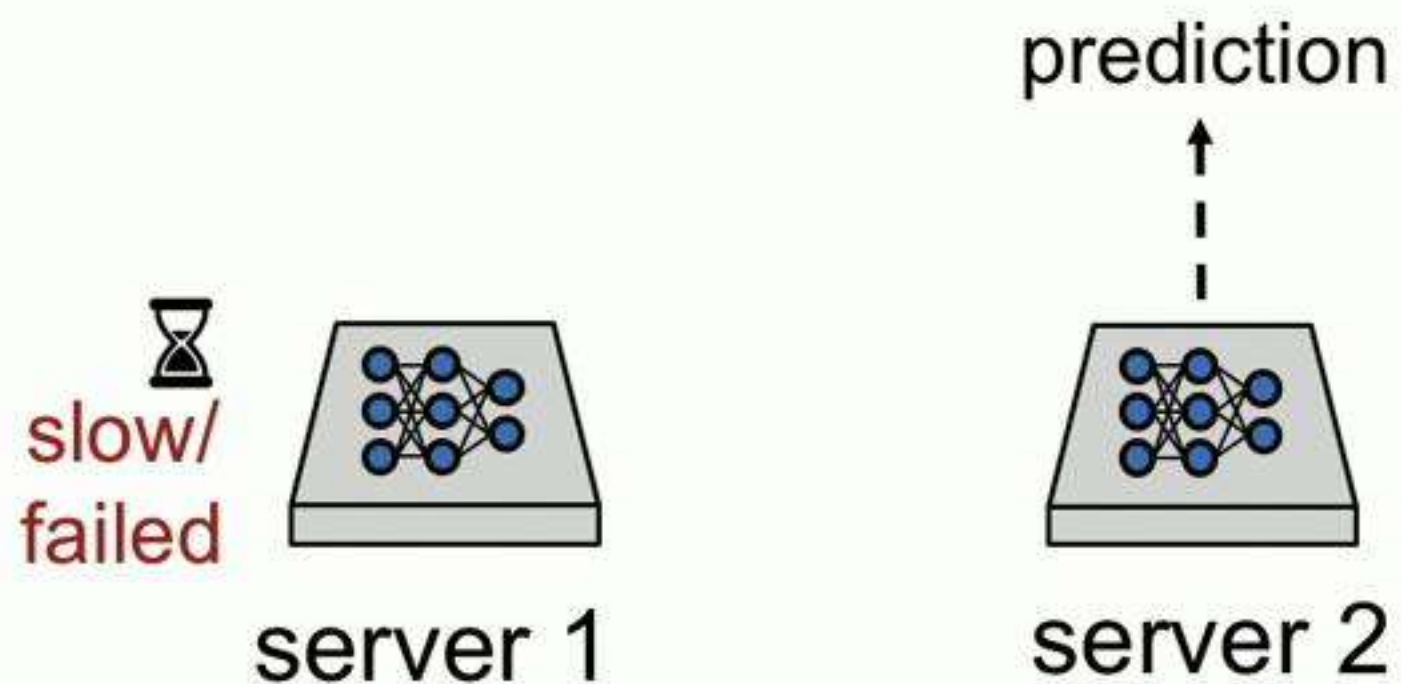




# Solution 1: Reactive

---

- First send to one server
- Wait, retry request

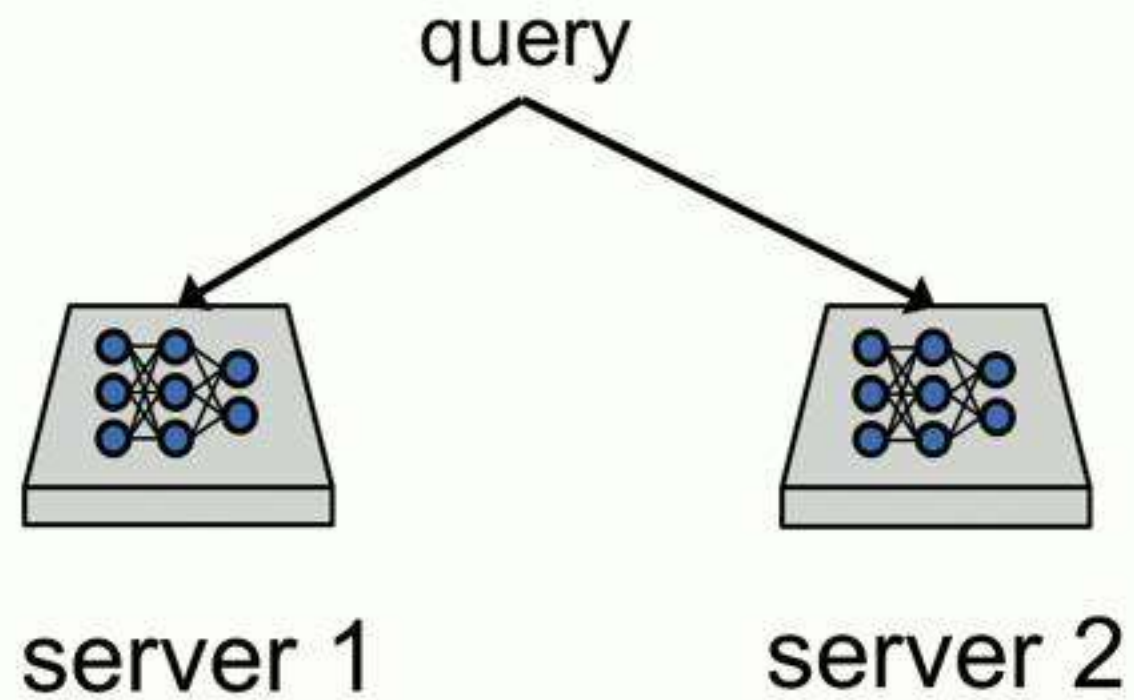


**Problem:** *delayed* mitigation

# Solution 2: Replicative

---

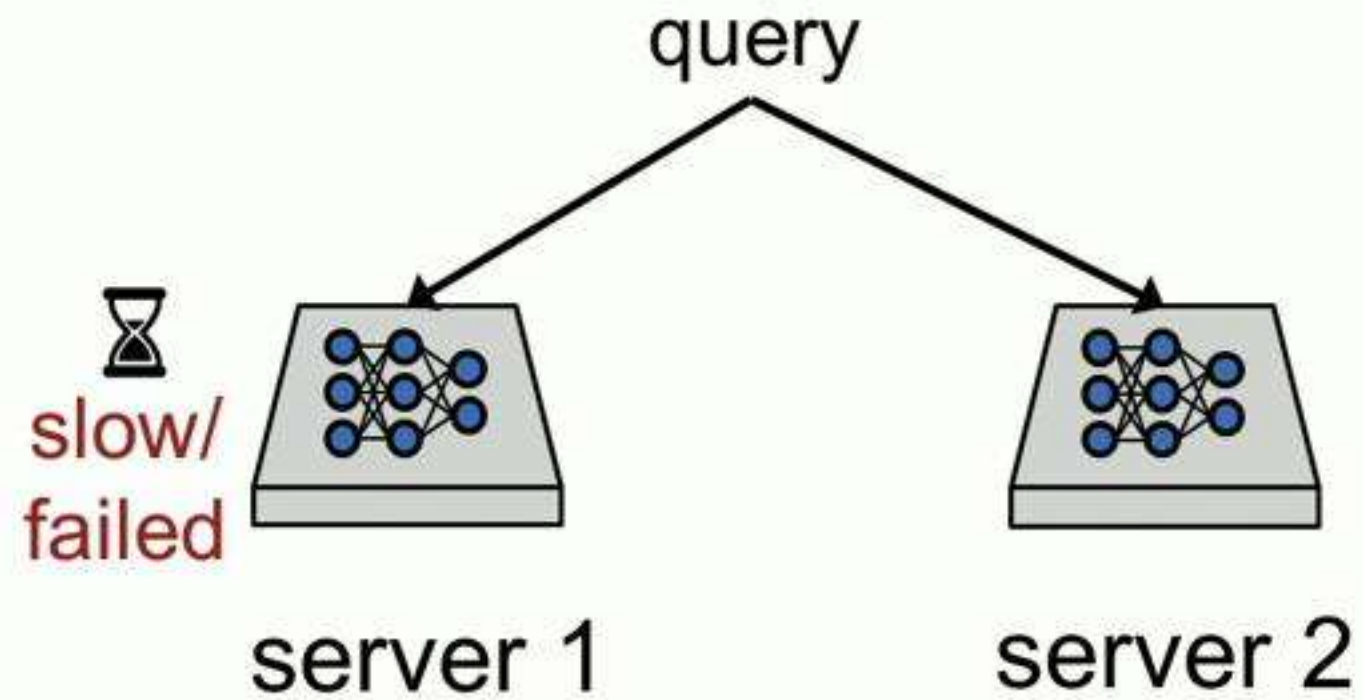
- Proactively issue duplicate query
- Wait only for first response



# Solution 2: Replicative

---

- Proactively issue duplicate query
- Wait only for first response

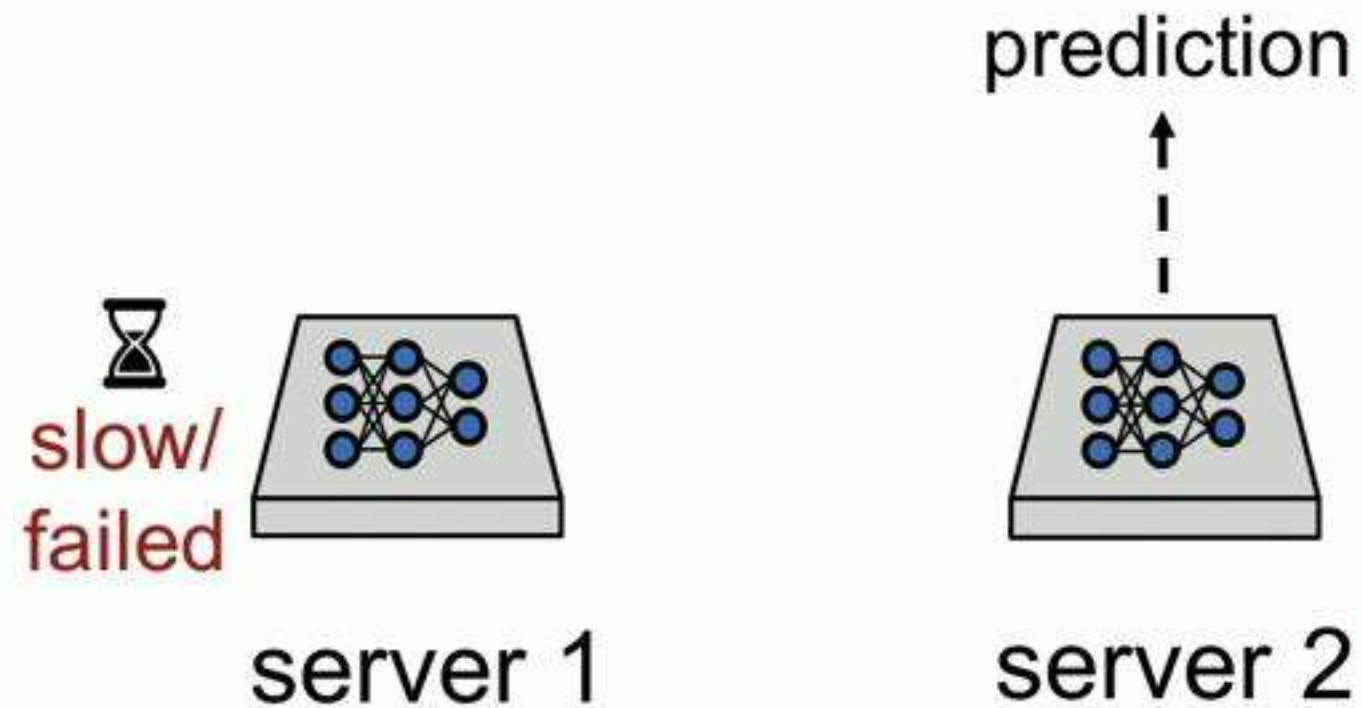




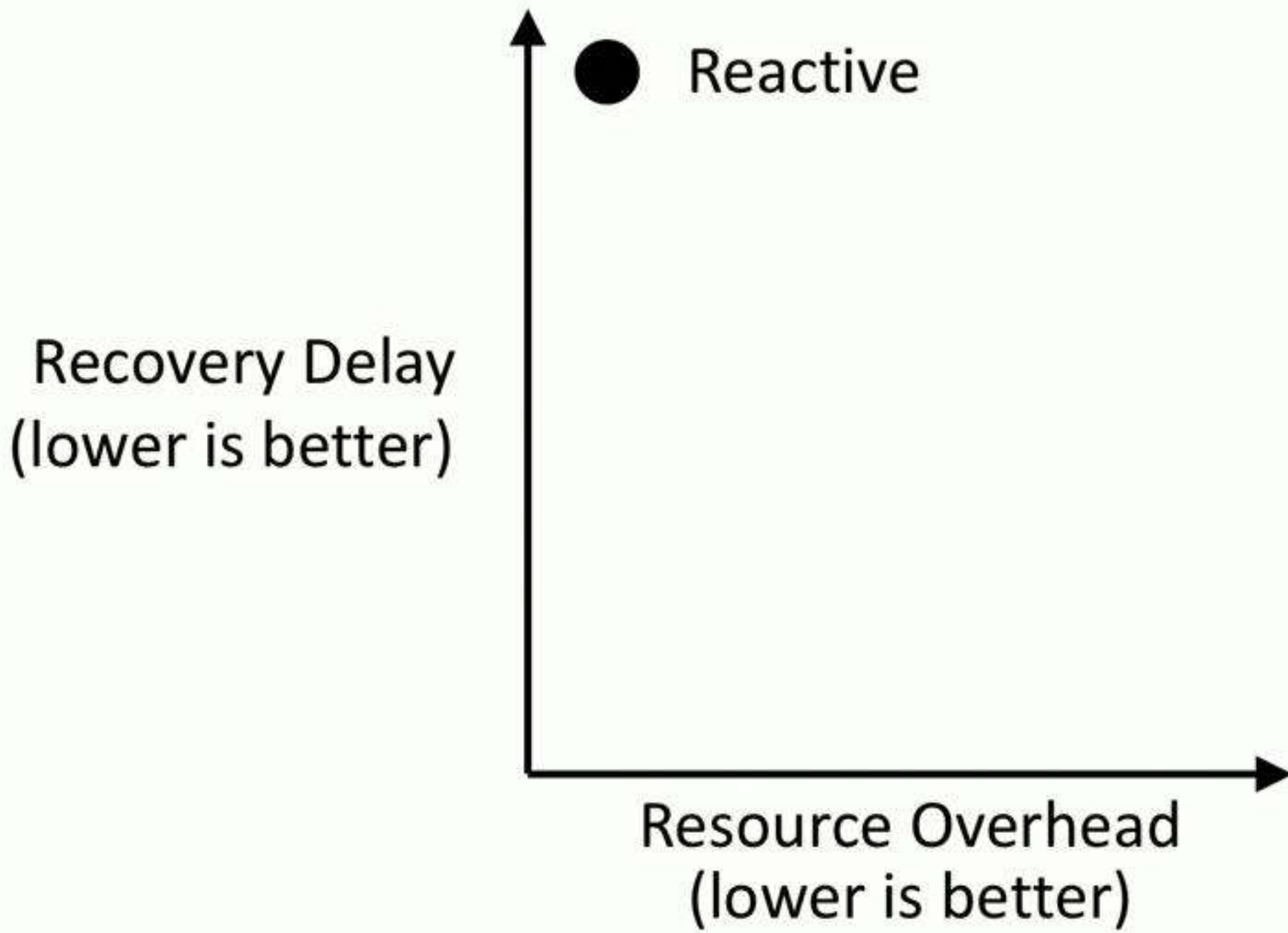
# Solution 2: Replicative

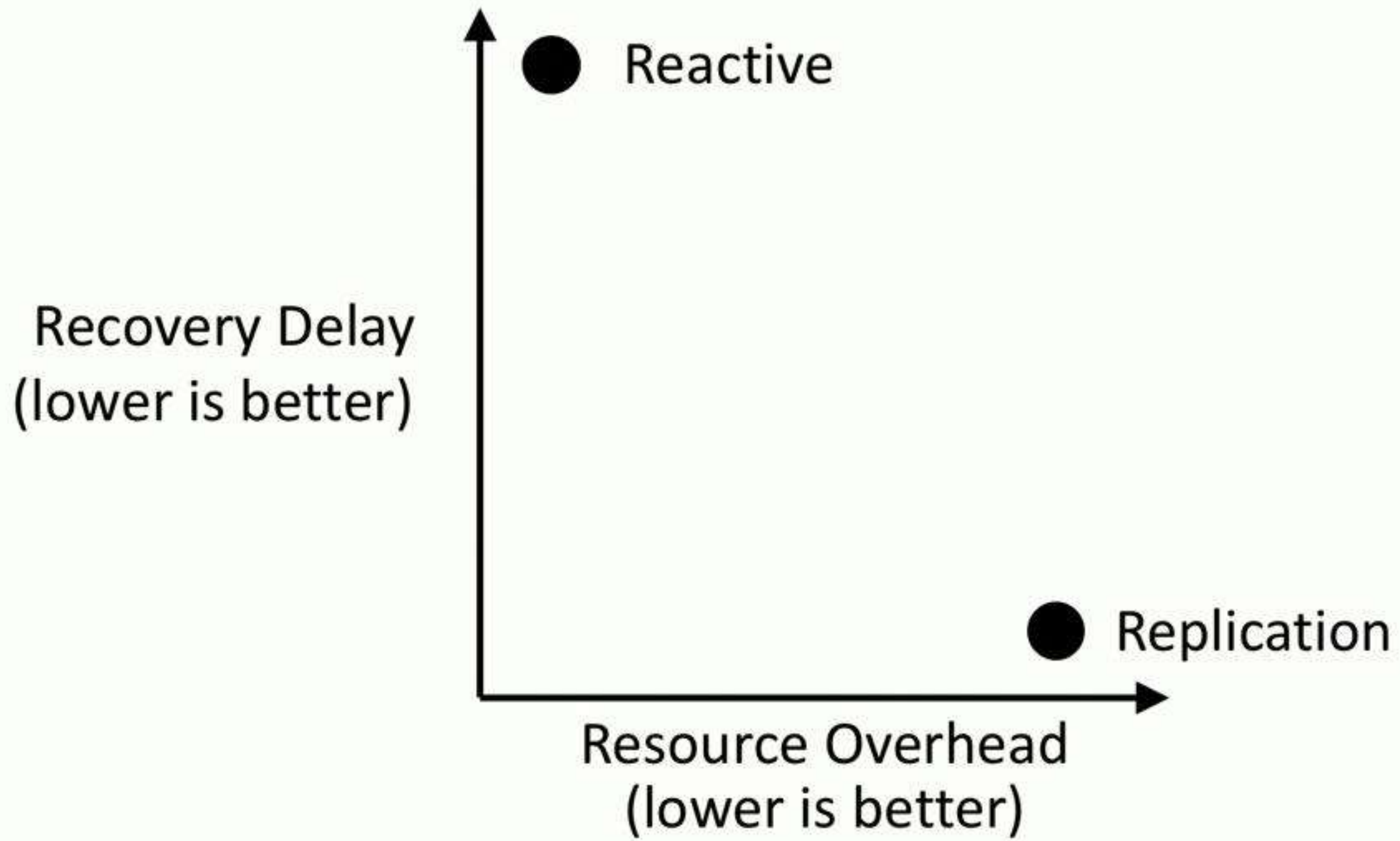
---

- Proactively issue duplicate query
- Wait only for first response

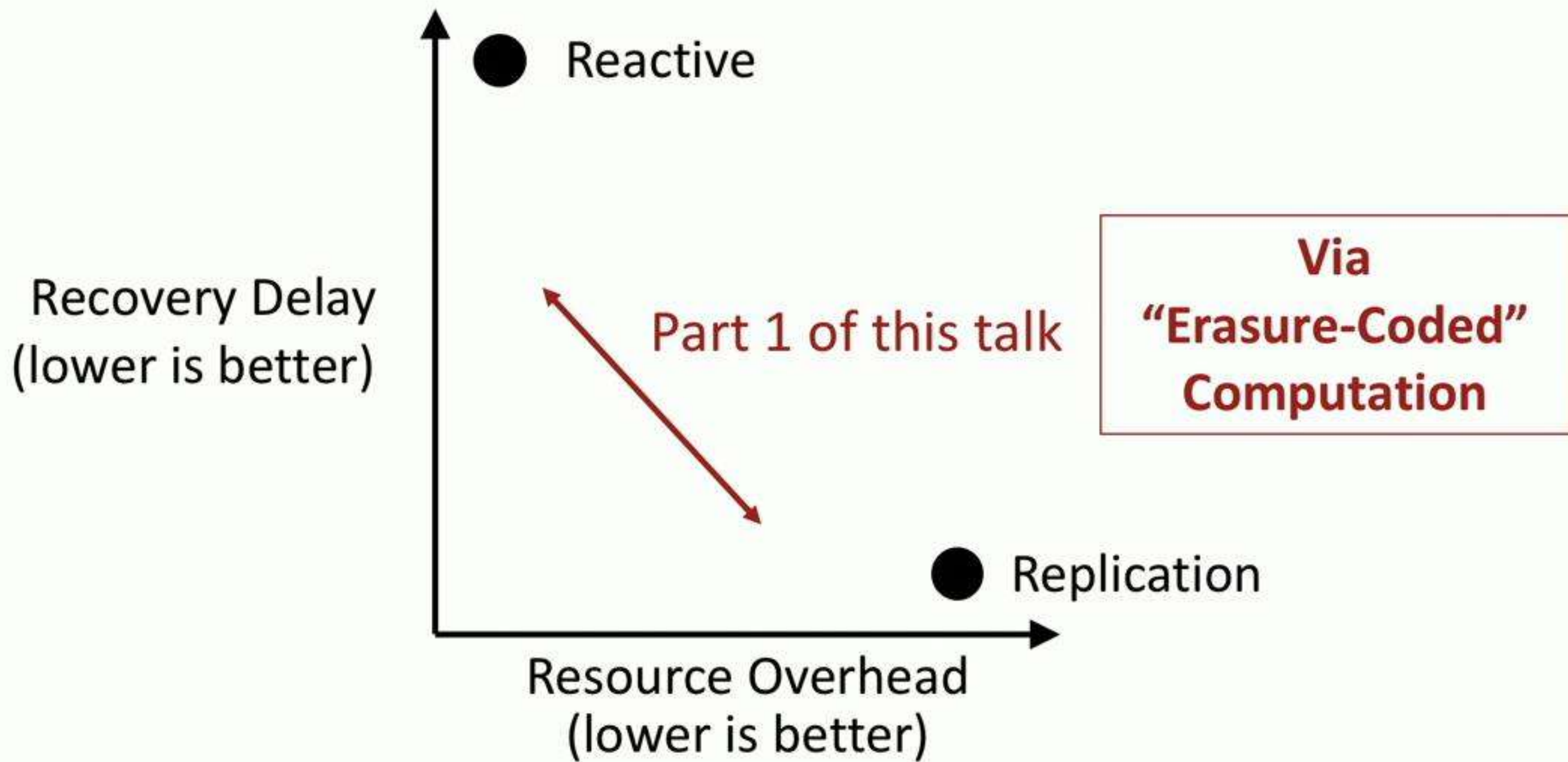


**Problem:** requires at least **2x** resources





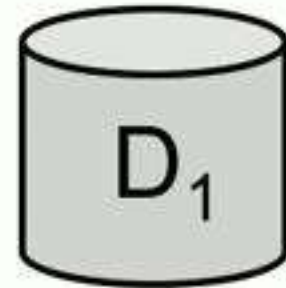




# Quick primer on erasure coding

---

- **Resource efficient redundancy** for resilience against unavailability
- Illustration using data storage application



Disk 1



Disk 2

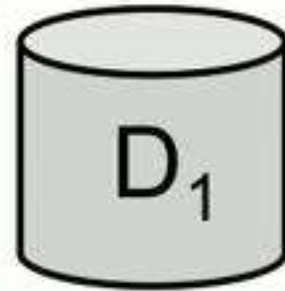
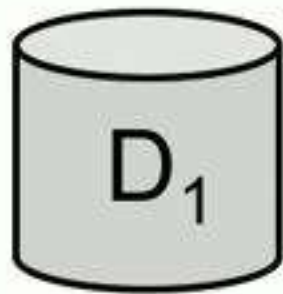
Goal: Add resilience against **single disk failure**

# Quick primer on erasure coding

---

Goal: Add resilience against single disk failure

Simplest way of adding redundancy: **replication**



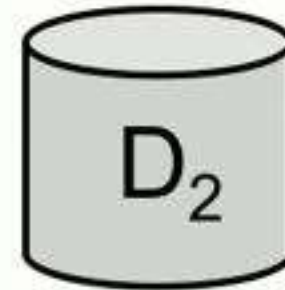
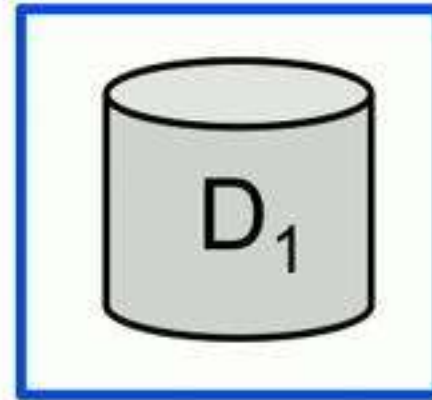


# Quick primer on erasure coding

---

Goal: Add resilience against single disk failure

Simplest way of adding redundancy: **replication**



Replication:

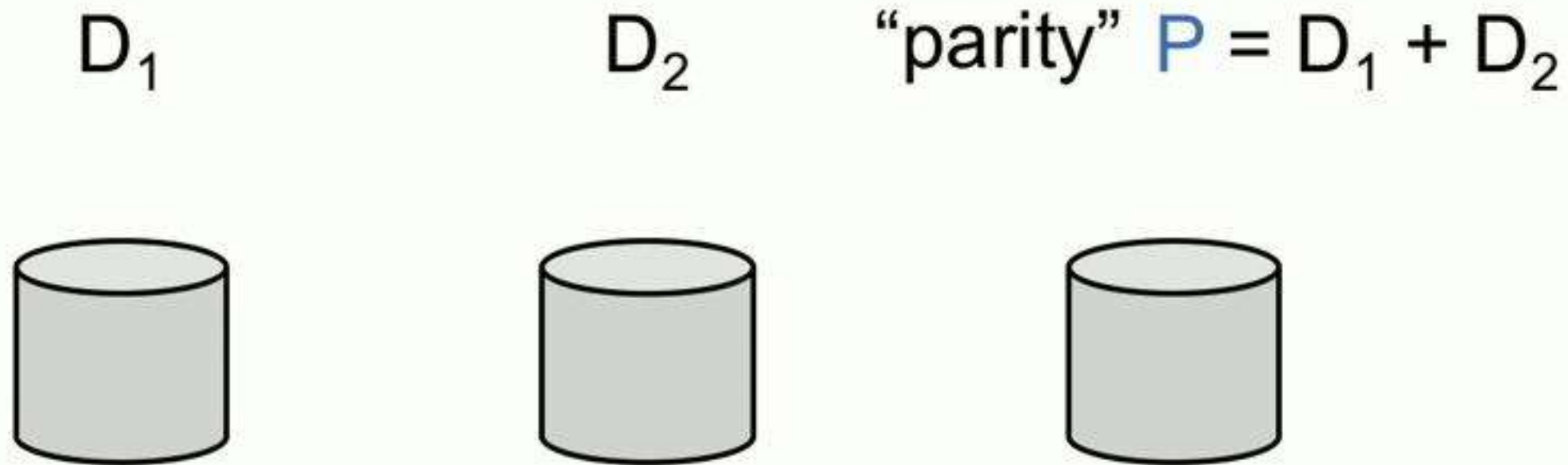
**At least 50% of the resources consumed for redundancy**

# Quick primer on erasure coding

---

Goal: Add resilience against single disk failure

Redundancy via **erasure coding**: “encode” data into “parities”

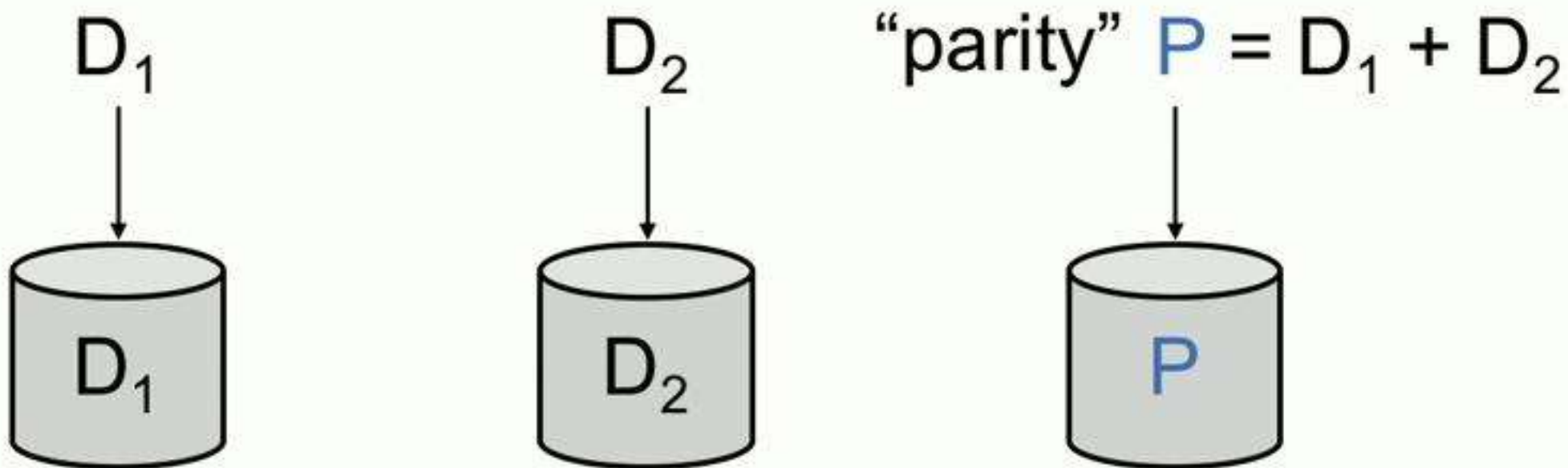


# Quick primer on erasure coding

---

Goal: Add resilience against single disk failure

Redundancy via **erasure coding**: “encode” data into “parities”



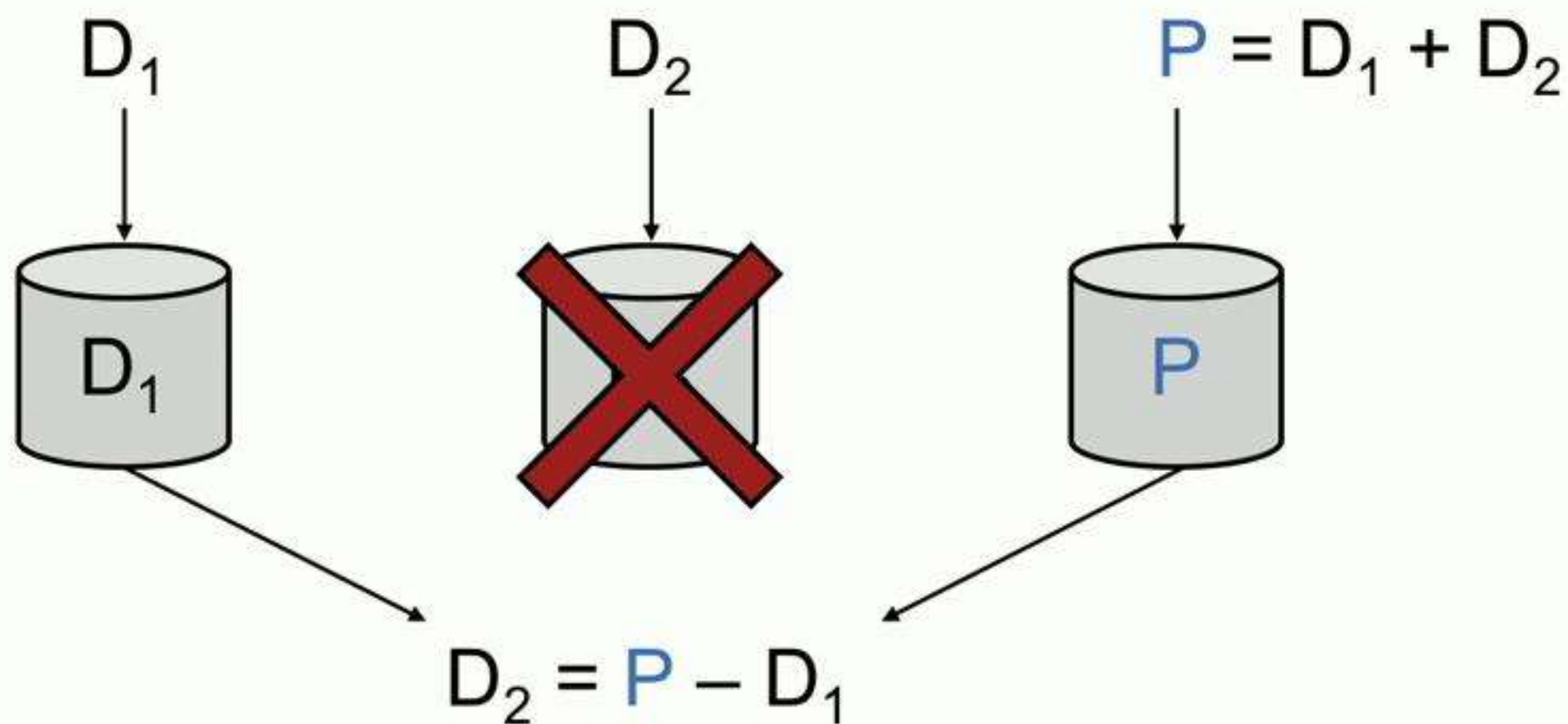


# Quick primer on erasure coding

---

Goal: Add resilience against single disk failure

Redundancy via **erasure coding**: “encode” data into “parities”



# Quick primer on erasure coding

---

Goal: Add resilience against single disk failure

Redundancy via **erasure coding**: “encode” data into “parities”

Erasure codes achieve desired resilience with much lower resource overhead than replication


$$D_2 = P - D_1$$

Only **~33%** of the resources consumed for redundancy

# Quick primer on erasure coding

---

In general:

$k$  data units  $\longrightarrow$  **encoding**  $\longrightarrow$   $r$  “parity” units

any  $k$  out of  $(k+r)$  units  $\longrightarrow$  **decoding**  $\longrightarrow$  original  $k$  data units



# Quick primer on erasure coding

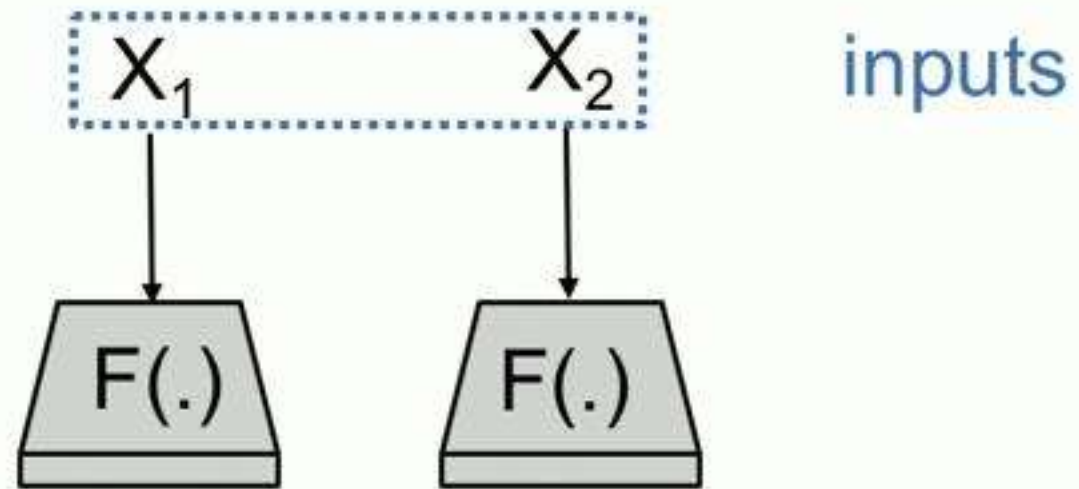
---

- Almost all erasure codes used in practice are “linear codes”
- Linear codes:
  - ⇒ Parity units **linear combinations** of the data units

# Coded computation

---

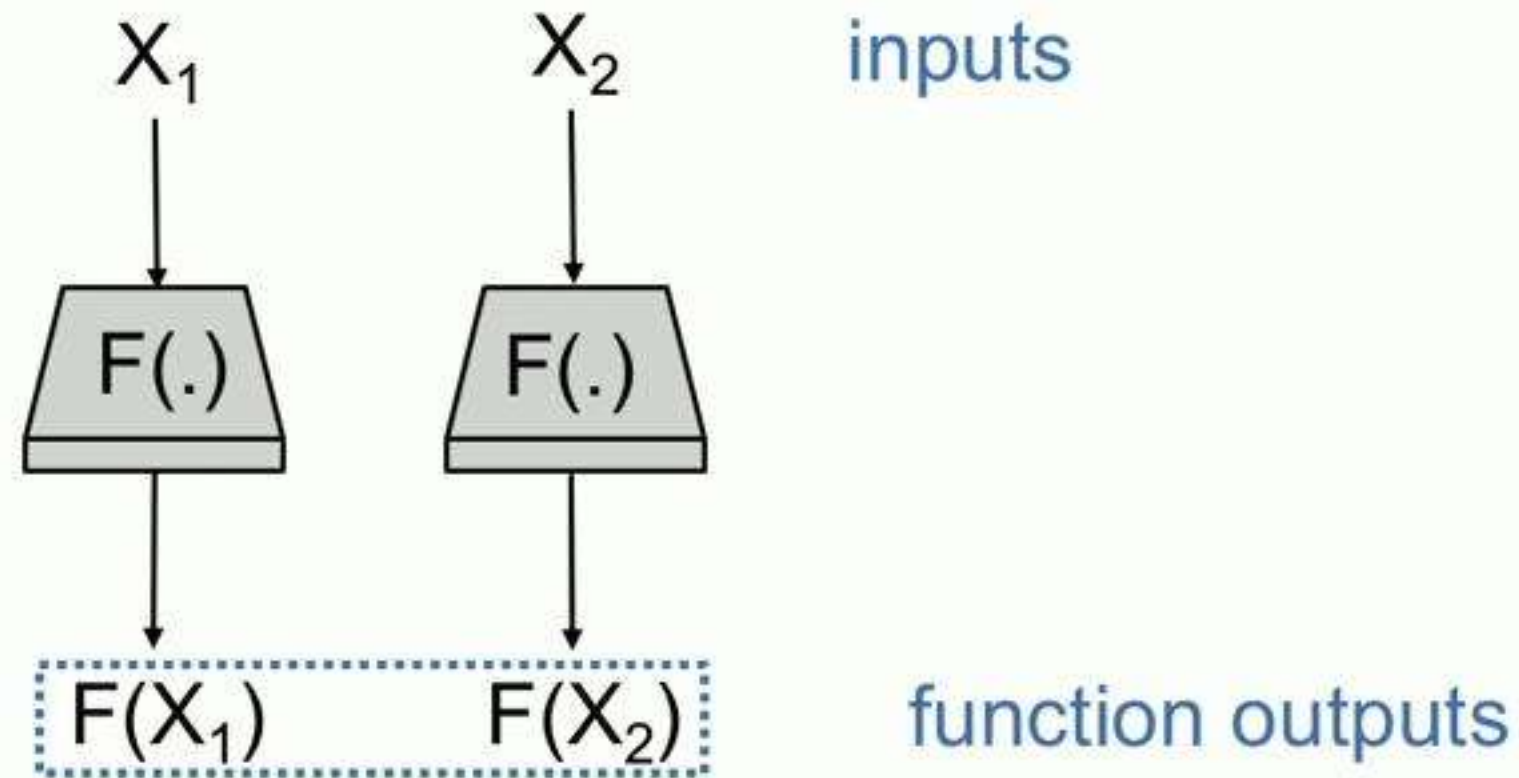
Multiple copies of a function  $F$  computed on different nodes



# Coded computation

---

Multiple copies of a function  $F$  computed on different nodes

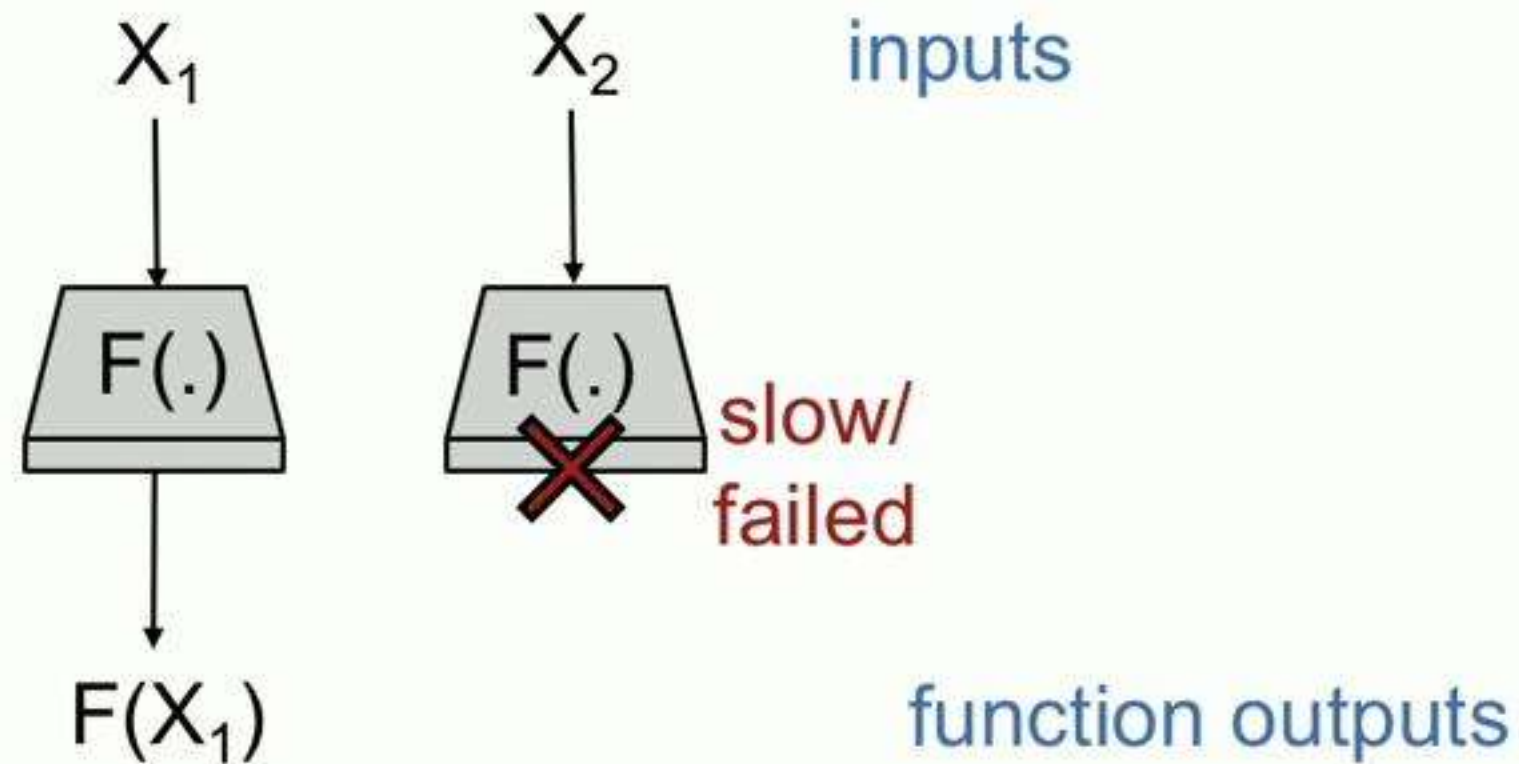




# Coded computation

---

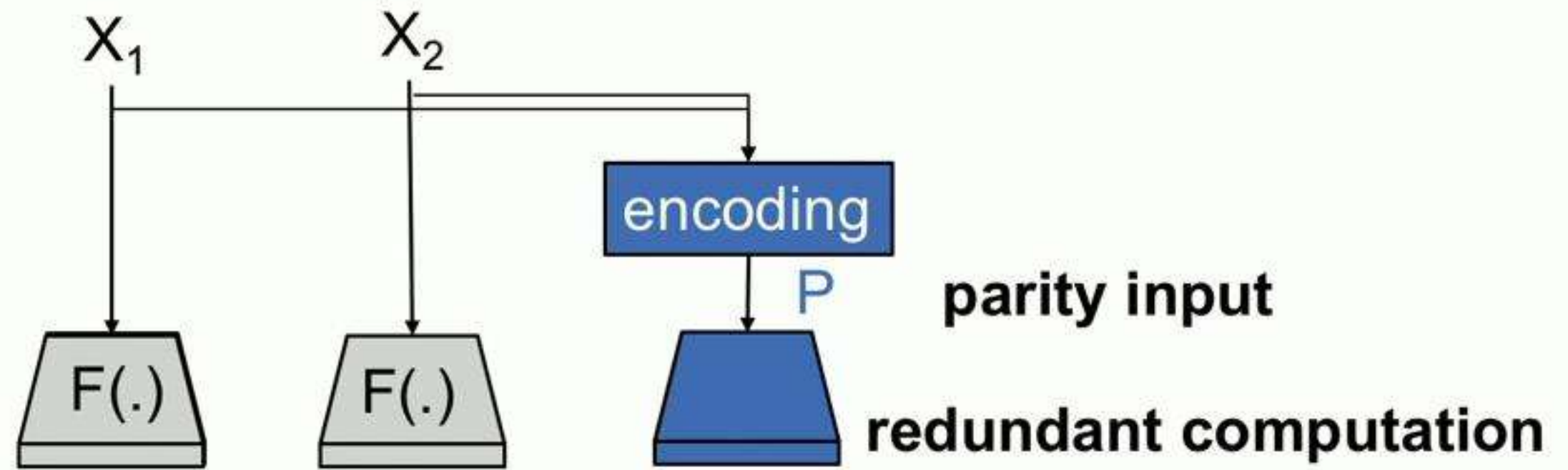
Multiple copies of a function  $F$  computed on different nodes



# Coded computation

---

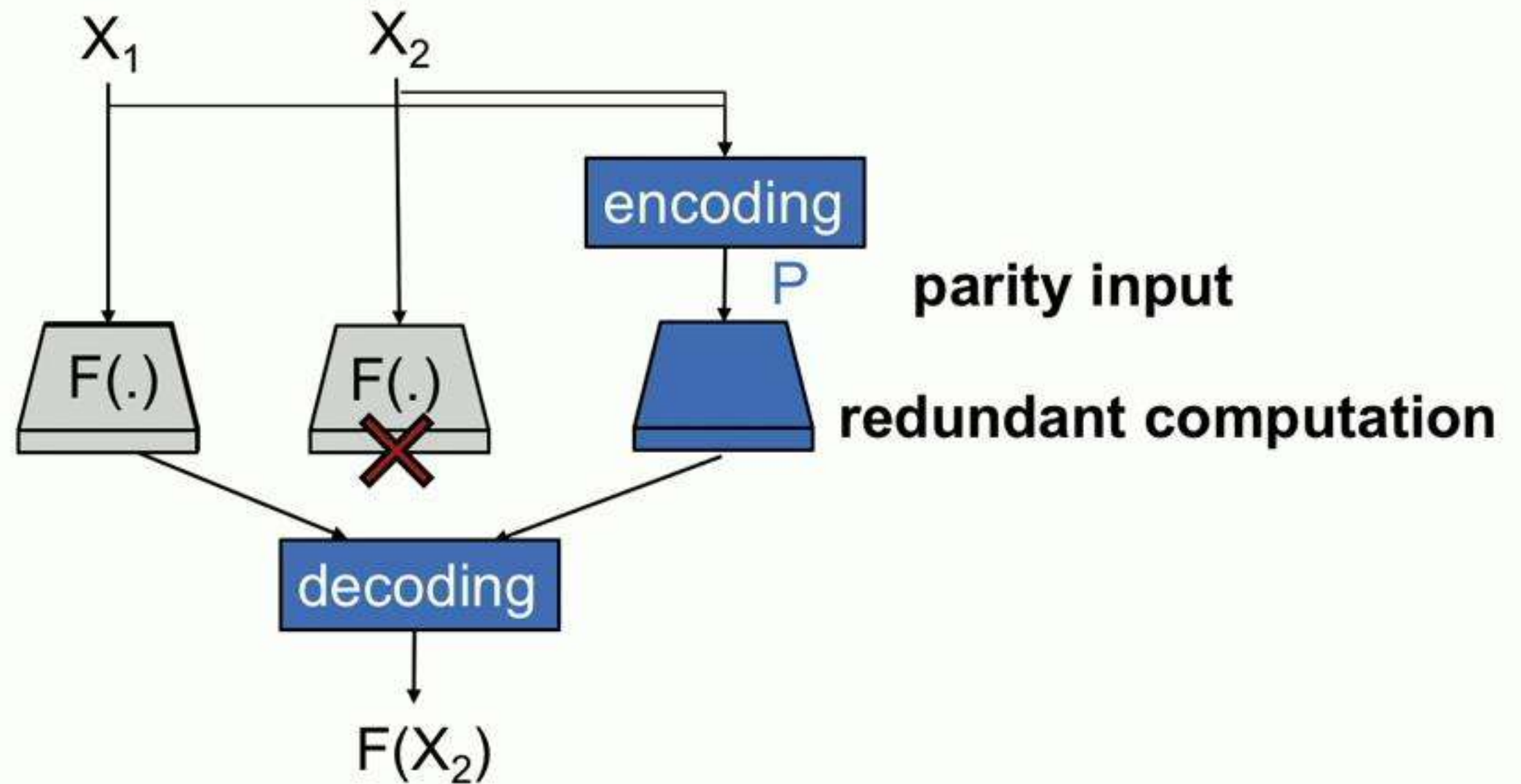
Multiple copies of a function  $F$  computed on different nodes



# Coded computation

---

Multiple copies of a function  $F$  computed on different nodes

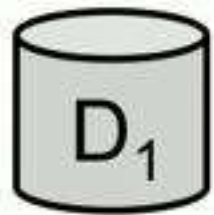




# Resilient computation via erasure coding

---

Codes for storage



slow/  
failed

Goal: recover **data**

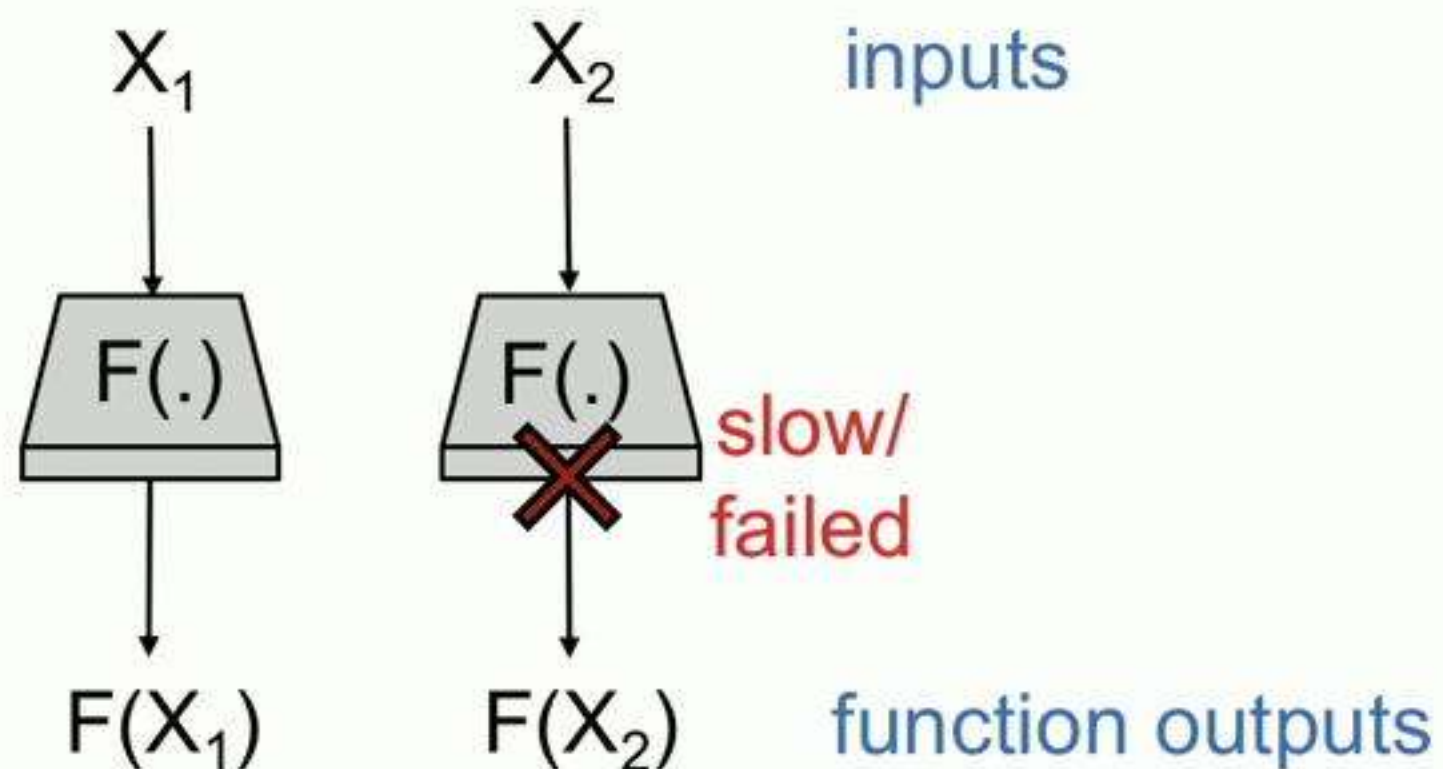
# Resilient computation via erasure coding

## Codes for storage



Goal: recover **data**

## Codes for computation



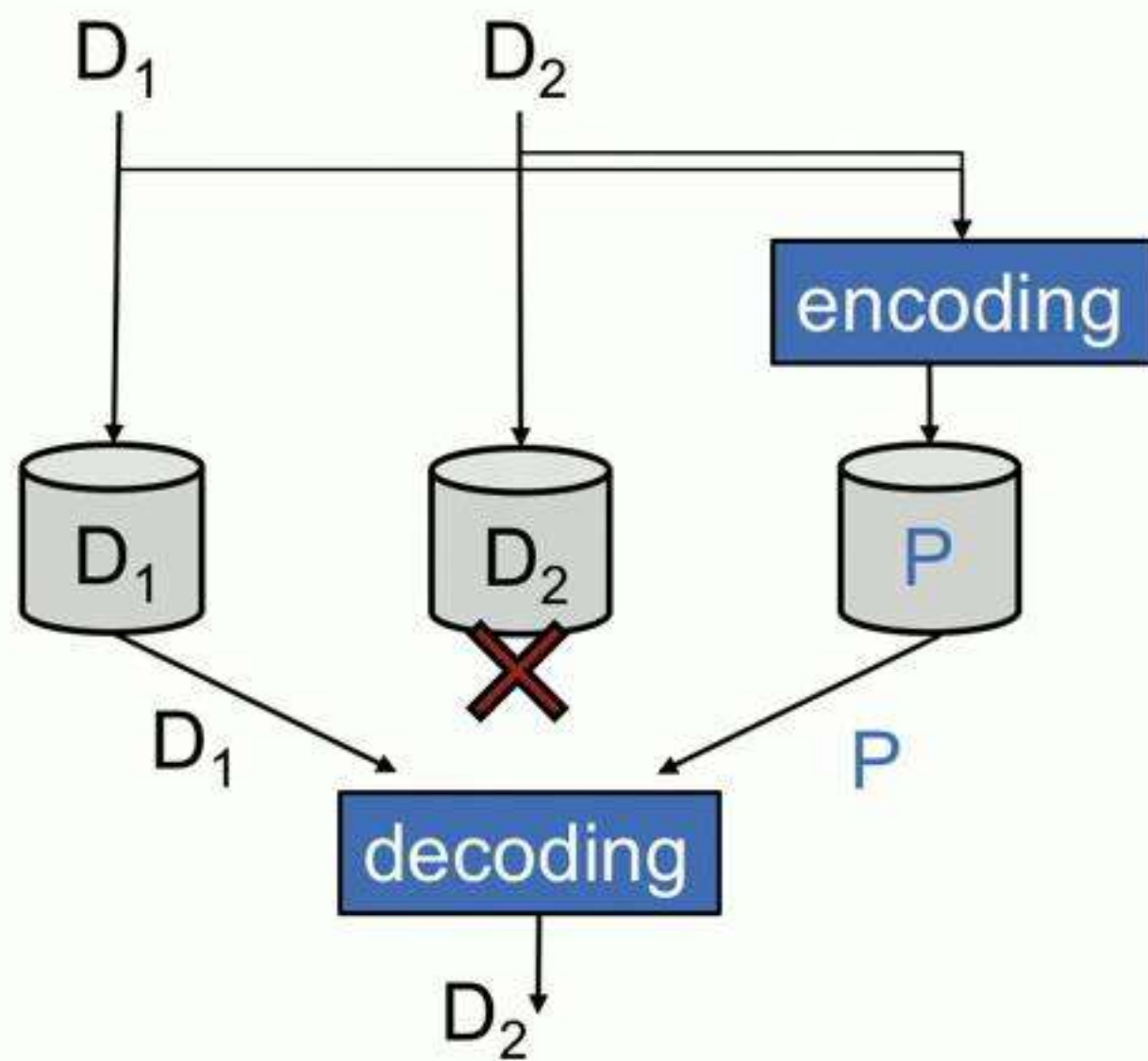
Goal: recover **function outputs on data**

# Resilient computation via erasure coding

---

## Codes for storage

Goal: recover **data**

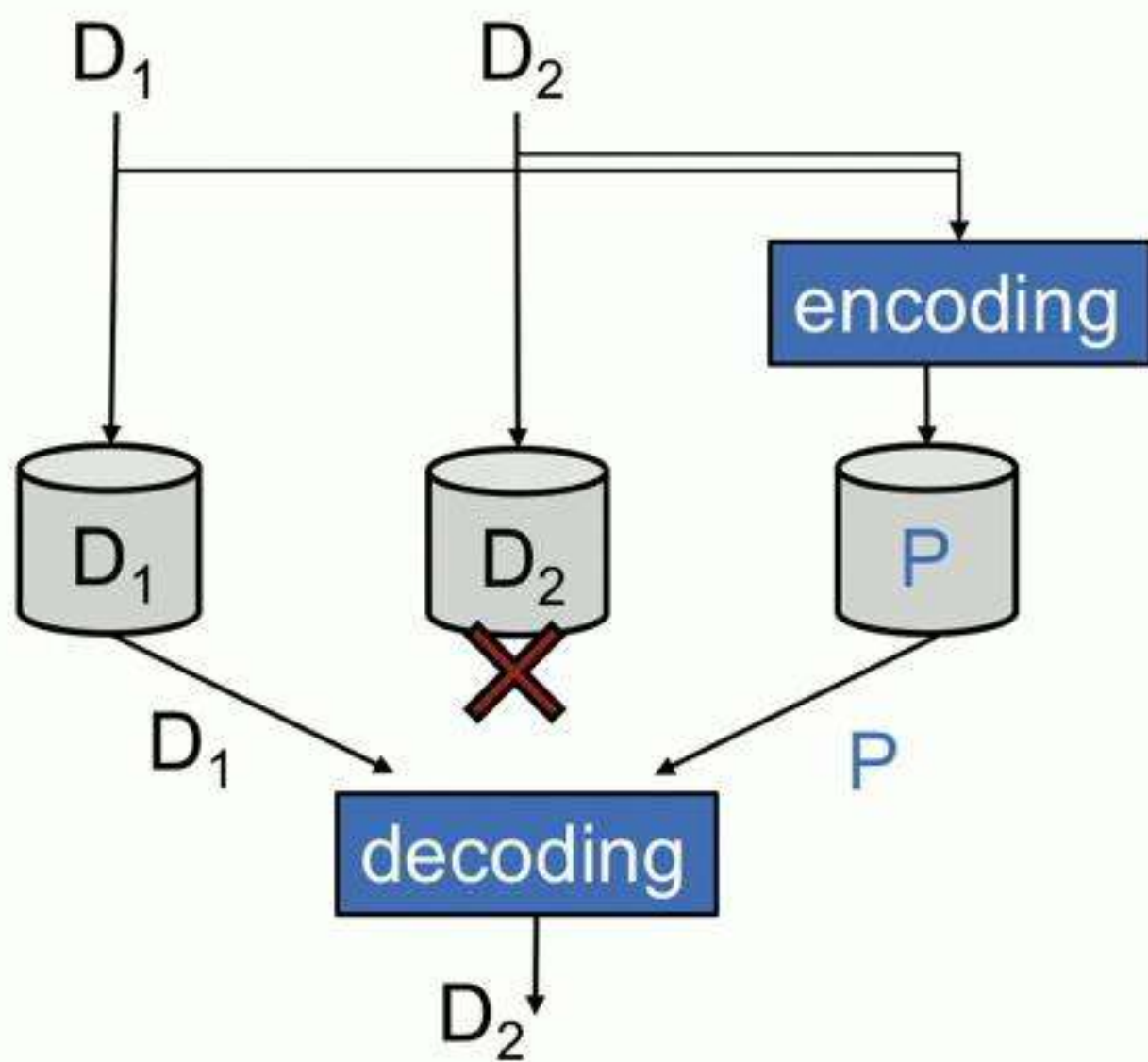




# Resilient computation via erasure coding

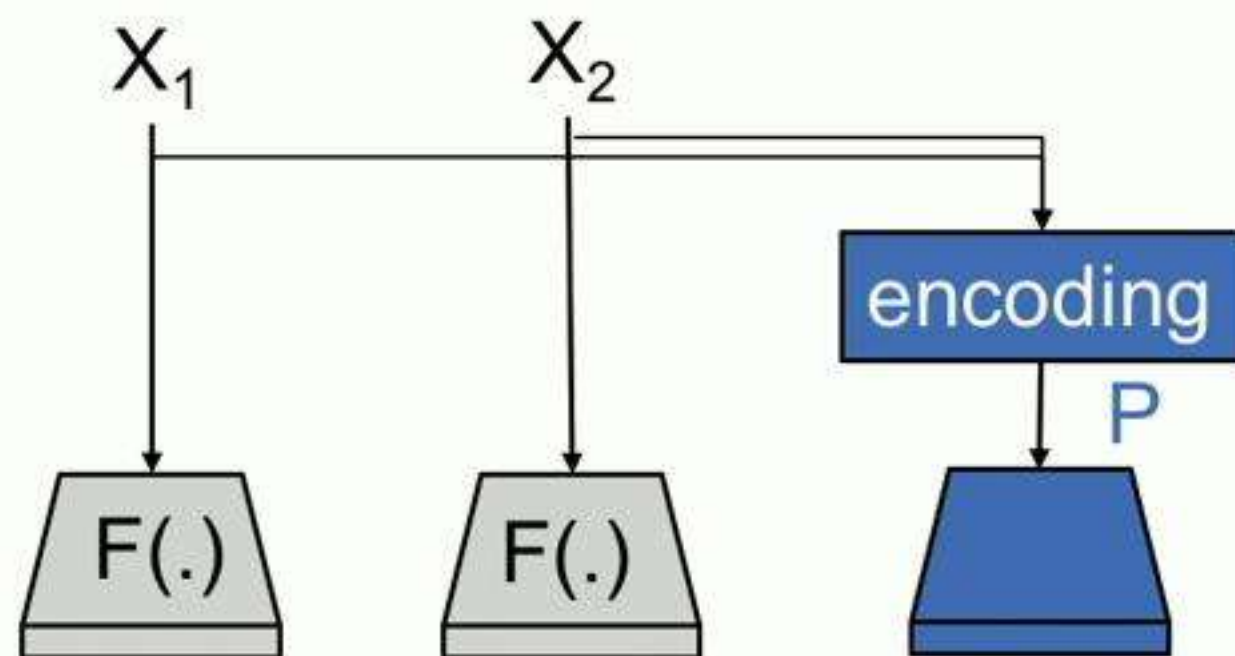
## Codes for storage

Goal: recover **data**



## Codes for computation

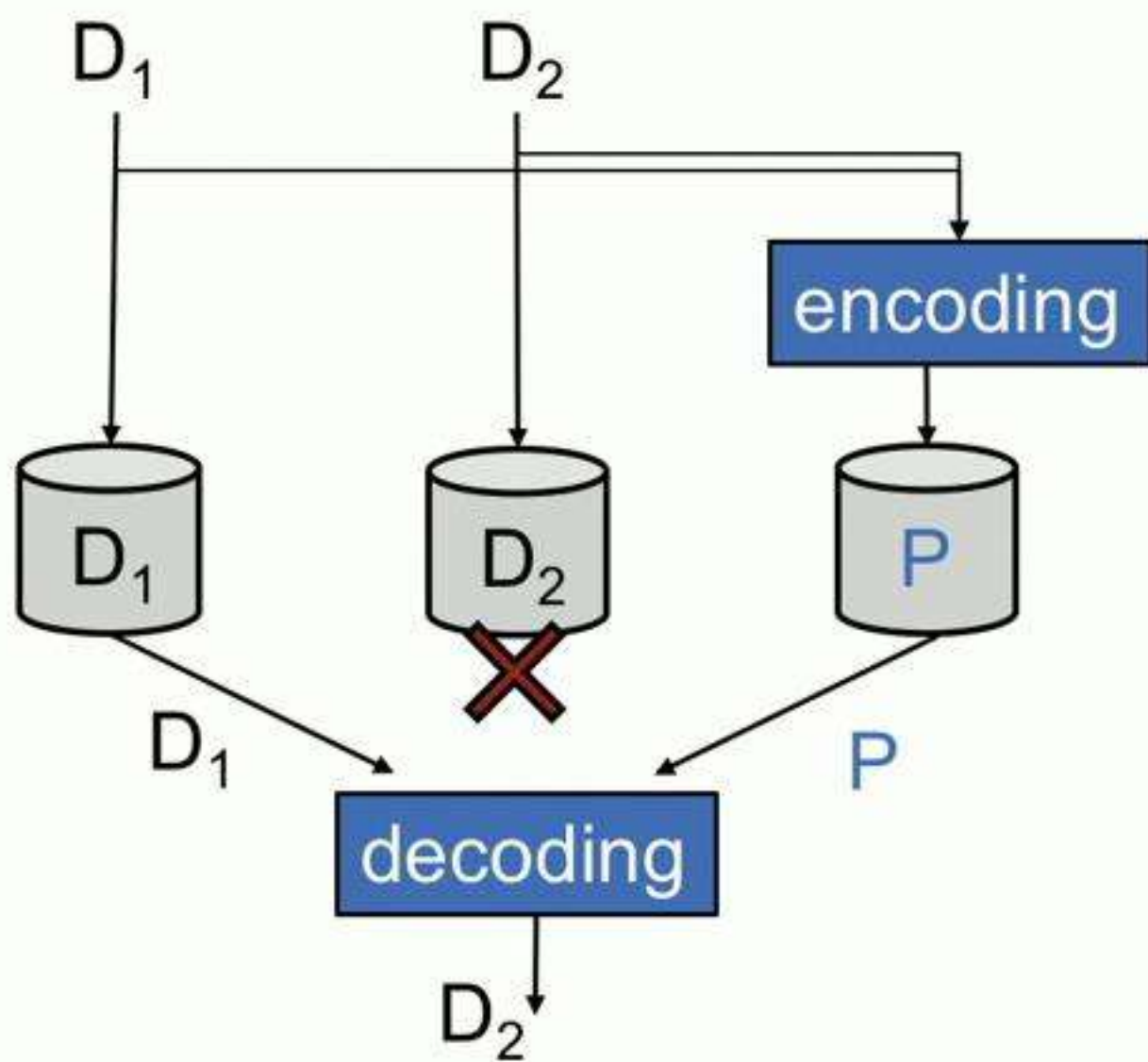
Goal: recover **function outputs on data**



# Resilient computation via erasure coding

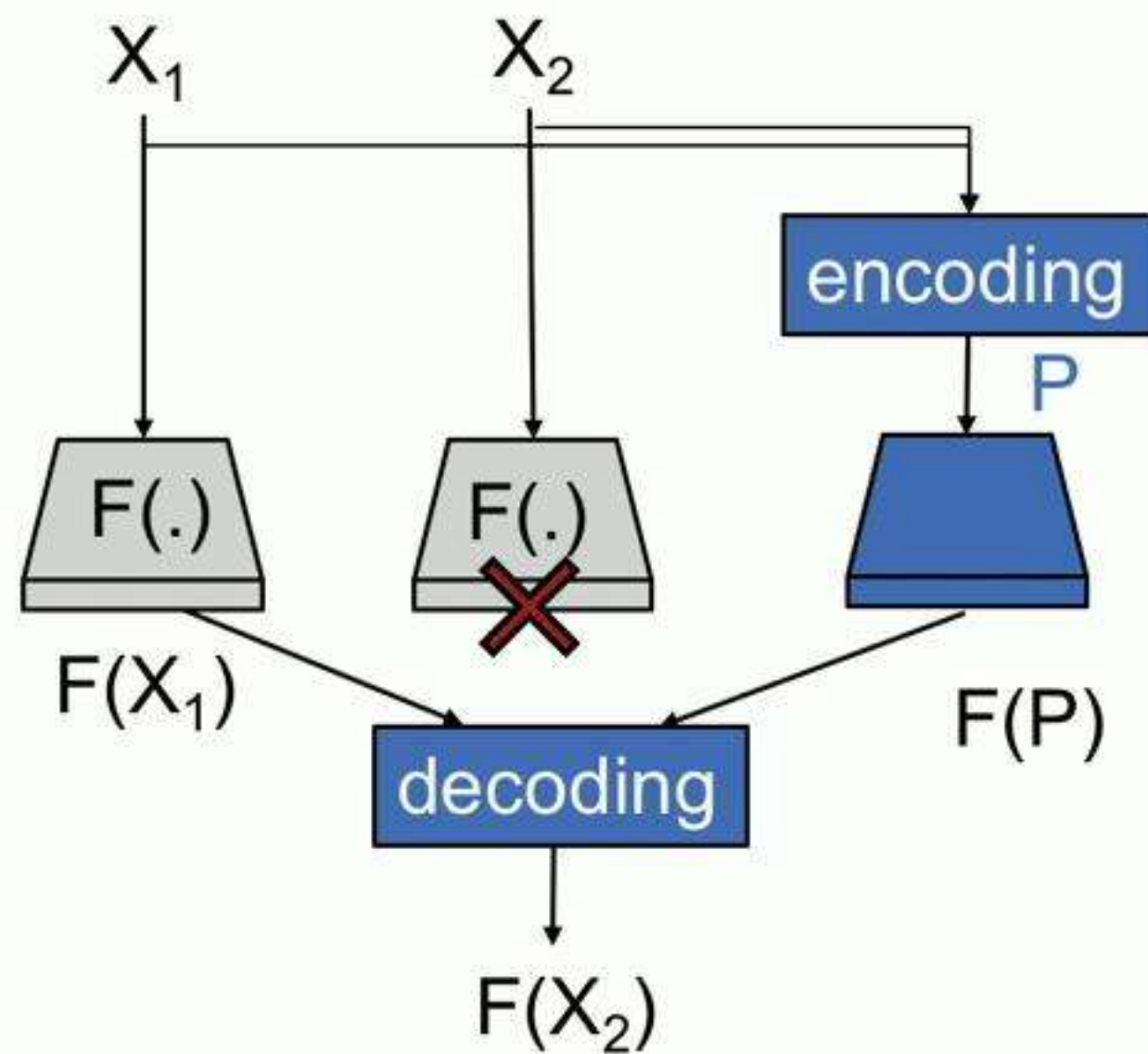
## Codes for storage

Goal: recover **data**



## Codes for computation

Goal: recover **function outputs on data**





# Resilient computation via erasure coding

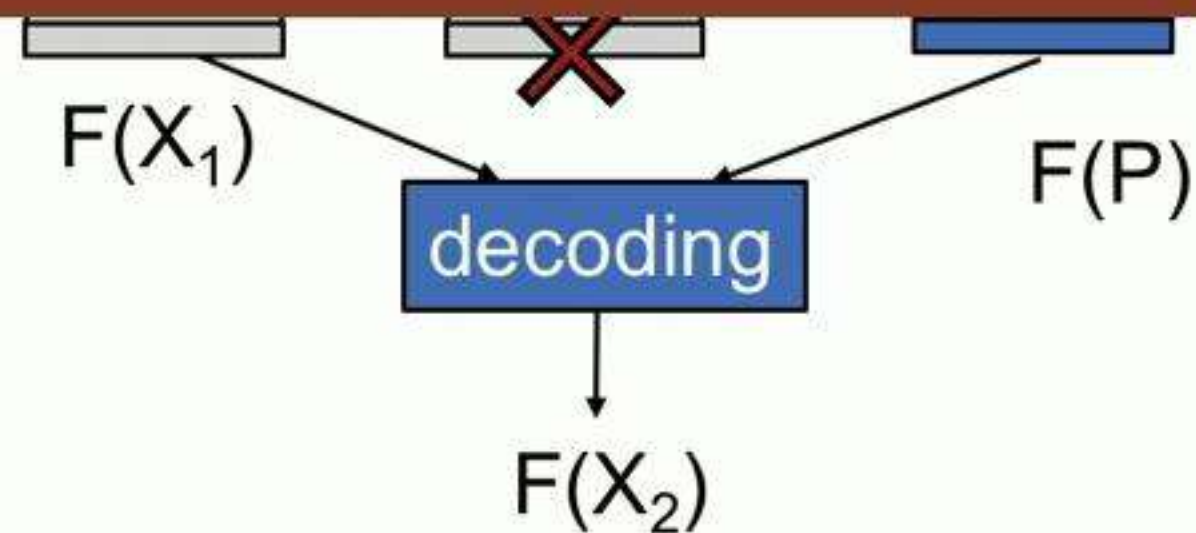
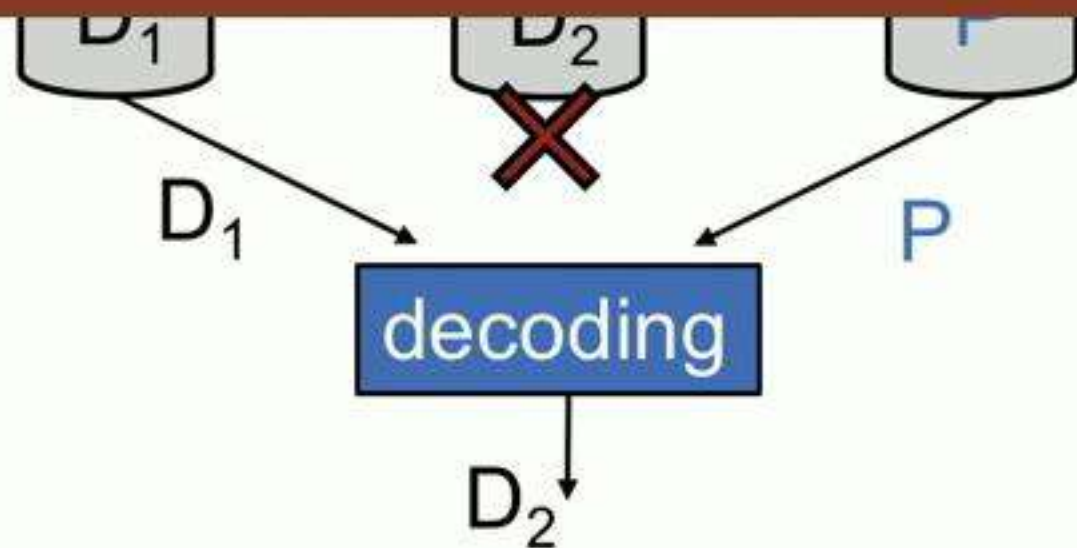
## Codes for storage

Goal: recover **data**

## Codes for computation

Goal: recover **function outputs on data**

When will such a reconstruction using the *computation on parity query* work?



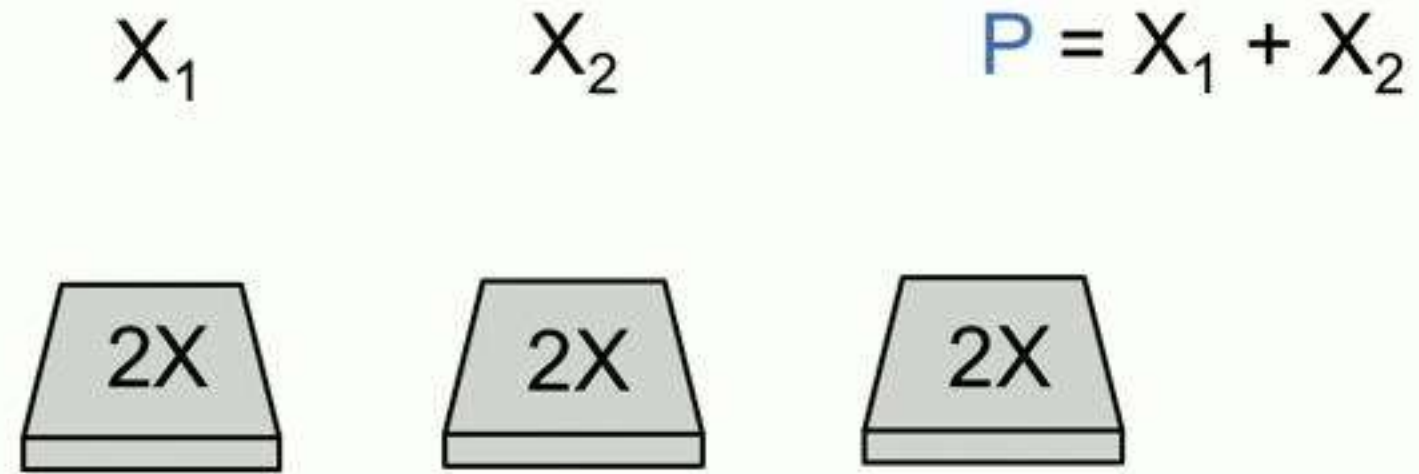


# Linear functions work with linear codes

---

Linear functions **commute with encoding and decoding** operations of linear codes

Example:  $F(X) = 2X$

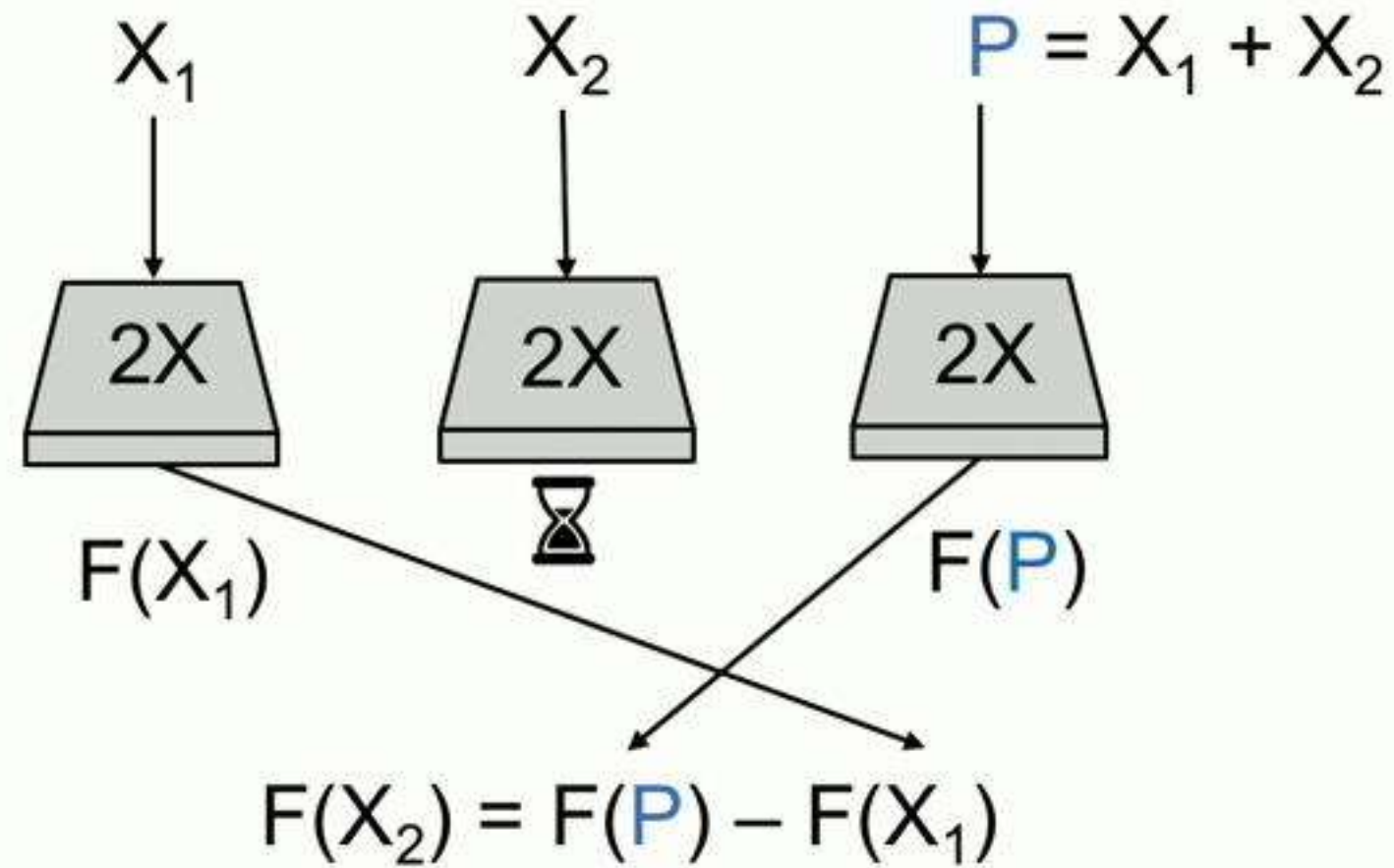


# Linear functions work with linear codes

---

Linear functions **commute with encoding and decoding** operations of linear codes

Example:  $F(X) = 2X$

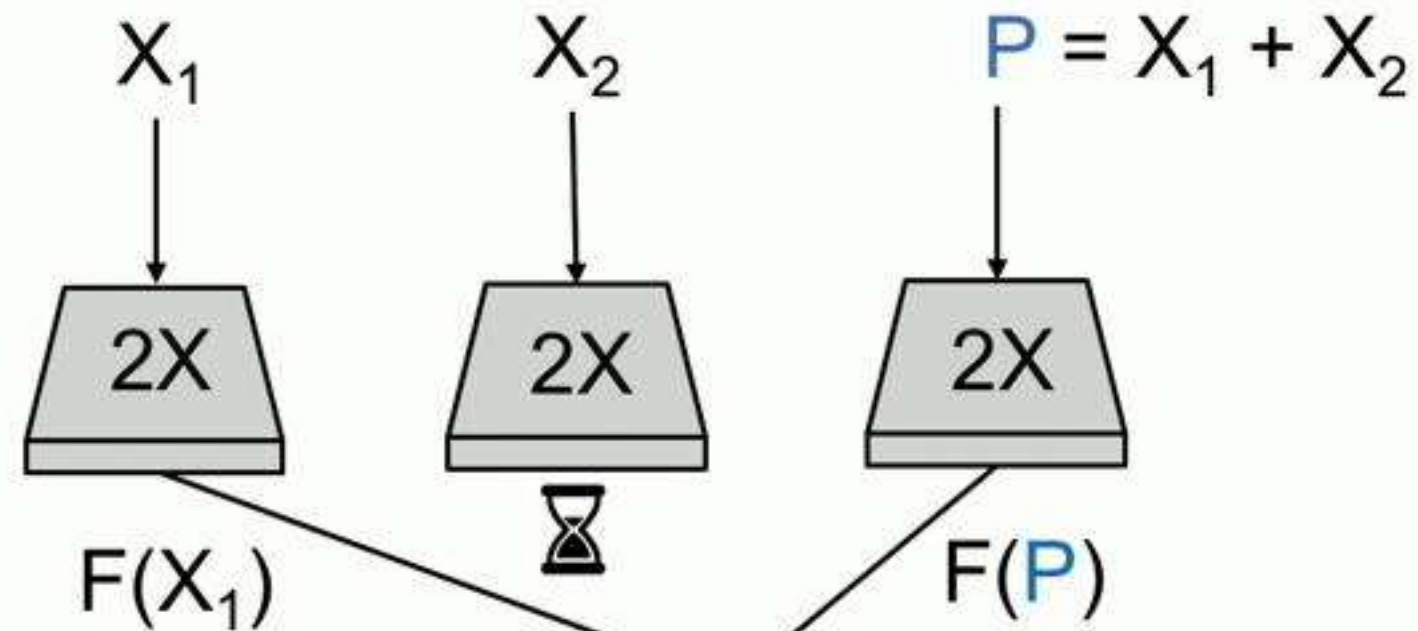


# Linear functions work with linear codes

---

Linear functions **commute with encoding and decoding** operations of linear codes

Example:  $F(X) = 2X$



$$\begin{aligned} F(X_2) &= F(P) - F(X_1) \\ &= 2(X_1 + X_2) - X_1 \\ &= 2X_2 \end{aligned}$$



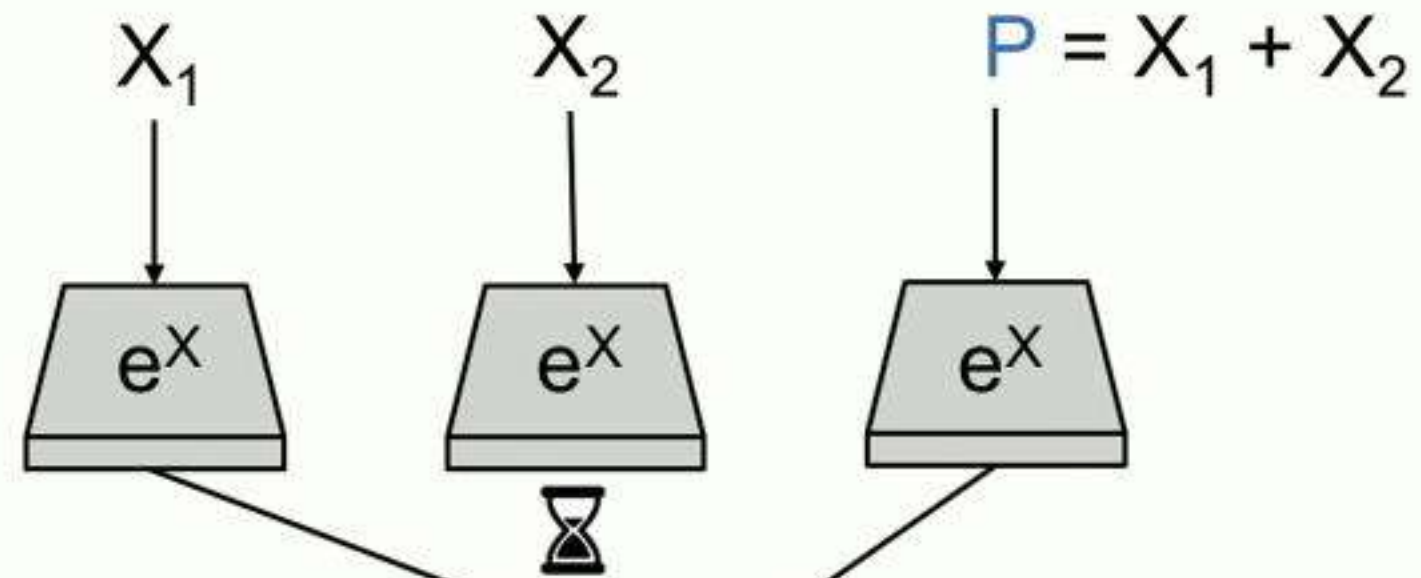


Key challenge:

Handling general **non-linear** functions

# Challenge in handling **non-linear** functions

Example:  $F(X) = e^X$



$$\begin{aligned} F(X_2) &= F(P) - F(X_1) \\ &= e^{(X_1+X_2)} - e^{X_1} \\ &= e^{X_1}(e^{X_2} - 1) \end{aligned}$$

**Incorrect**  
**Needed  $e^{X_2}$**



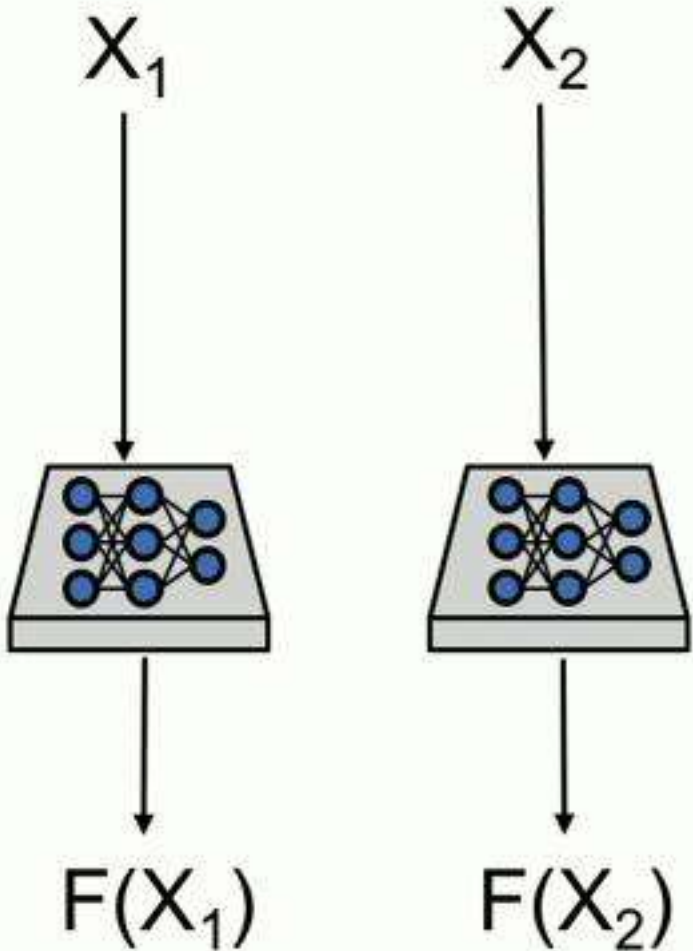
# Related works

---

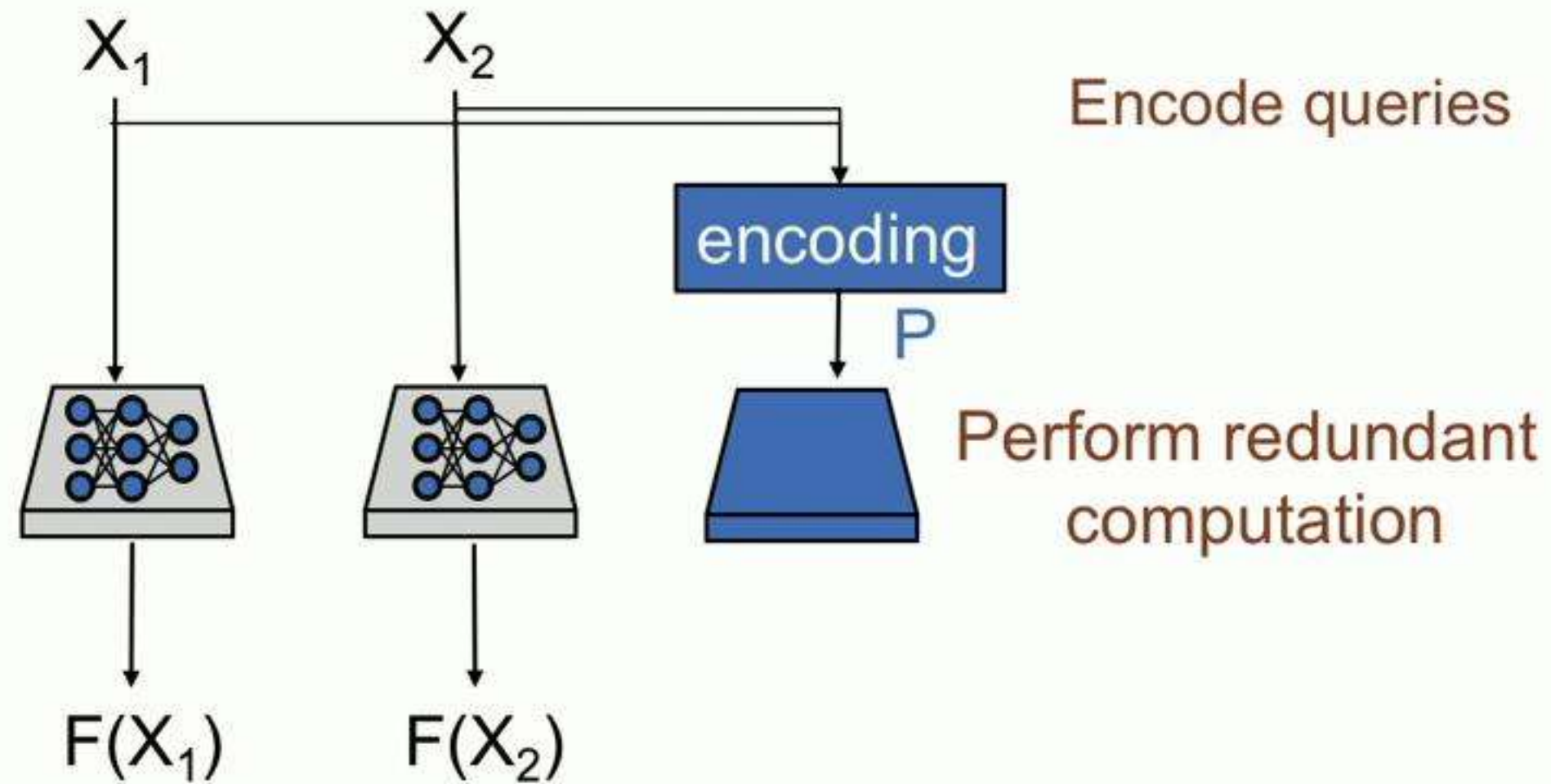
- HW-fault tolerant computation: Huang et al. 1984, Jou et al. 1986,...
- Lee et al., 2016 introduced coded computation in distributed setting for tolerating stragglers/failures
  - Distributed matrix-vector multiplication
- Large body of recent work
  - Dutta et al., 2016, Dutta et al., 2017, Karakus et al., 2017, Fahim et al., 2017, Yu et al., 2017, Reisizadeh and Pedarsani, 2017, Charles et al., 2017, Reisizadeh et al., 2017, Raviv et al., 2017, Yang et al., 2017, Yu et al., 2018, Baharav et al., 2018, Charles and Papailiopoulos, 2018, Chen et al., 2018, Dutta et al., 2018, Halbawi et al., 2018, Haddadpour and Cadambe, 2018, Haddadpour et al., 2018, Jeong et al., 2018, Kiani et al., 2018, Li et al., 2018, Mallick et al., 2018, Maity et al., 2018, Park et al., 2018, Sheth et al., 2018, Wang et al., 2018, Yang et al., 2018, Dutta et al., 2019, Fahim and Cadambe, 2019, Gupta et al., 2019, Kadhe et al., 2019, Yang et al., 2019, ...



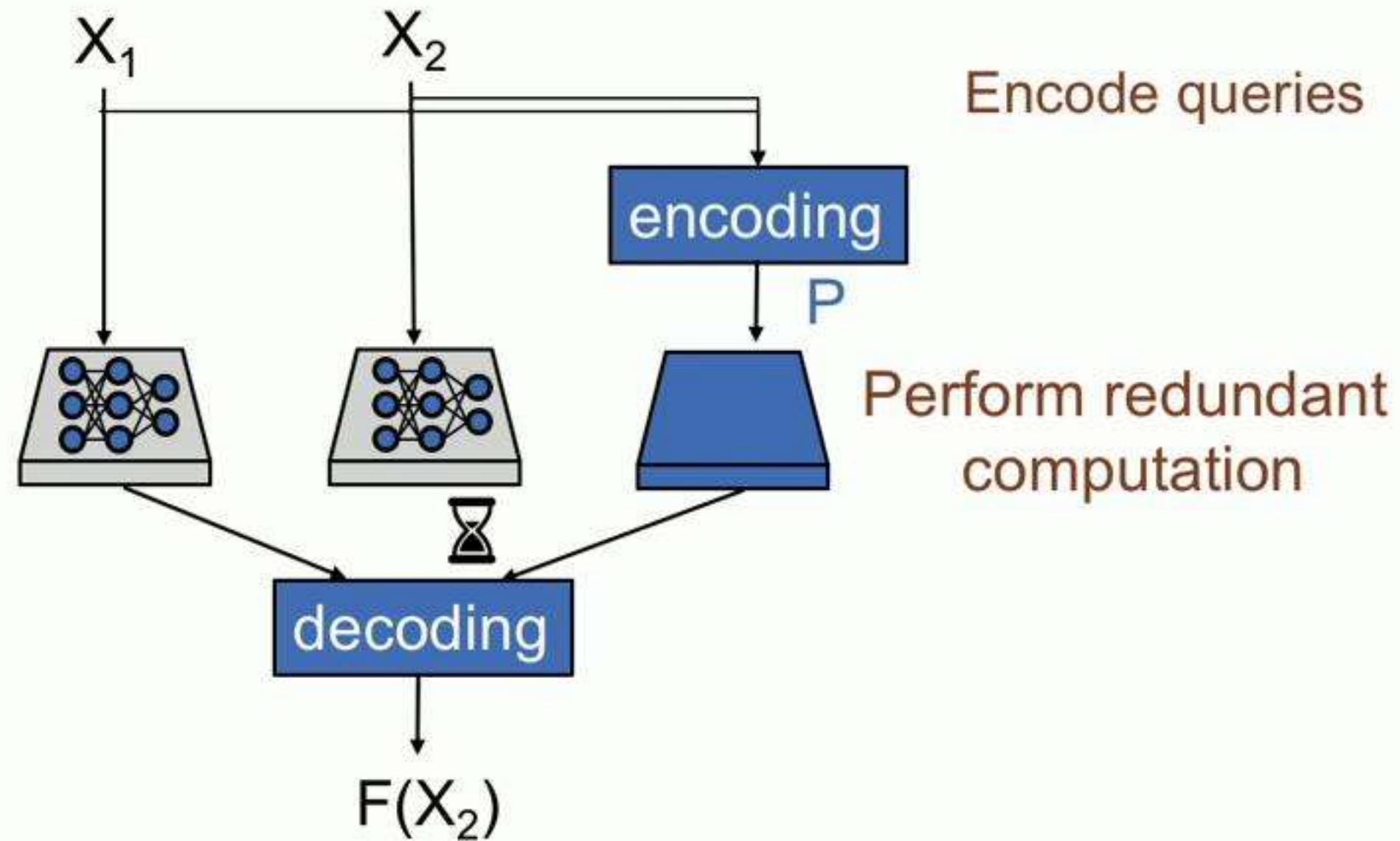
# Coded computation in prediction serving systems



# Coded computation in prediction serving systems



# Coded computation in prediction serving systems



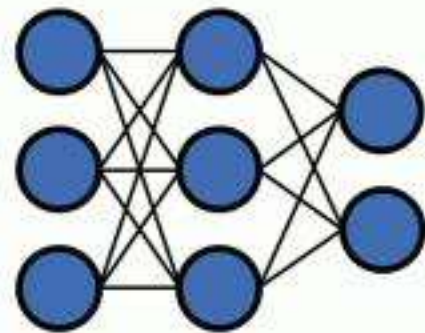


# Challenge

---

- State-of-the-art models for a variety of tasks are **neural networks**

Neural network model



=

Complex  
non-linear function

Coded computation for  
general non-linear functions is challenging

# Challenge

---

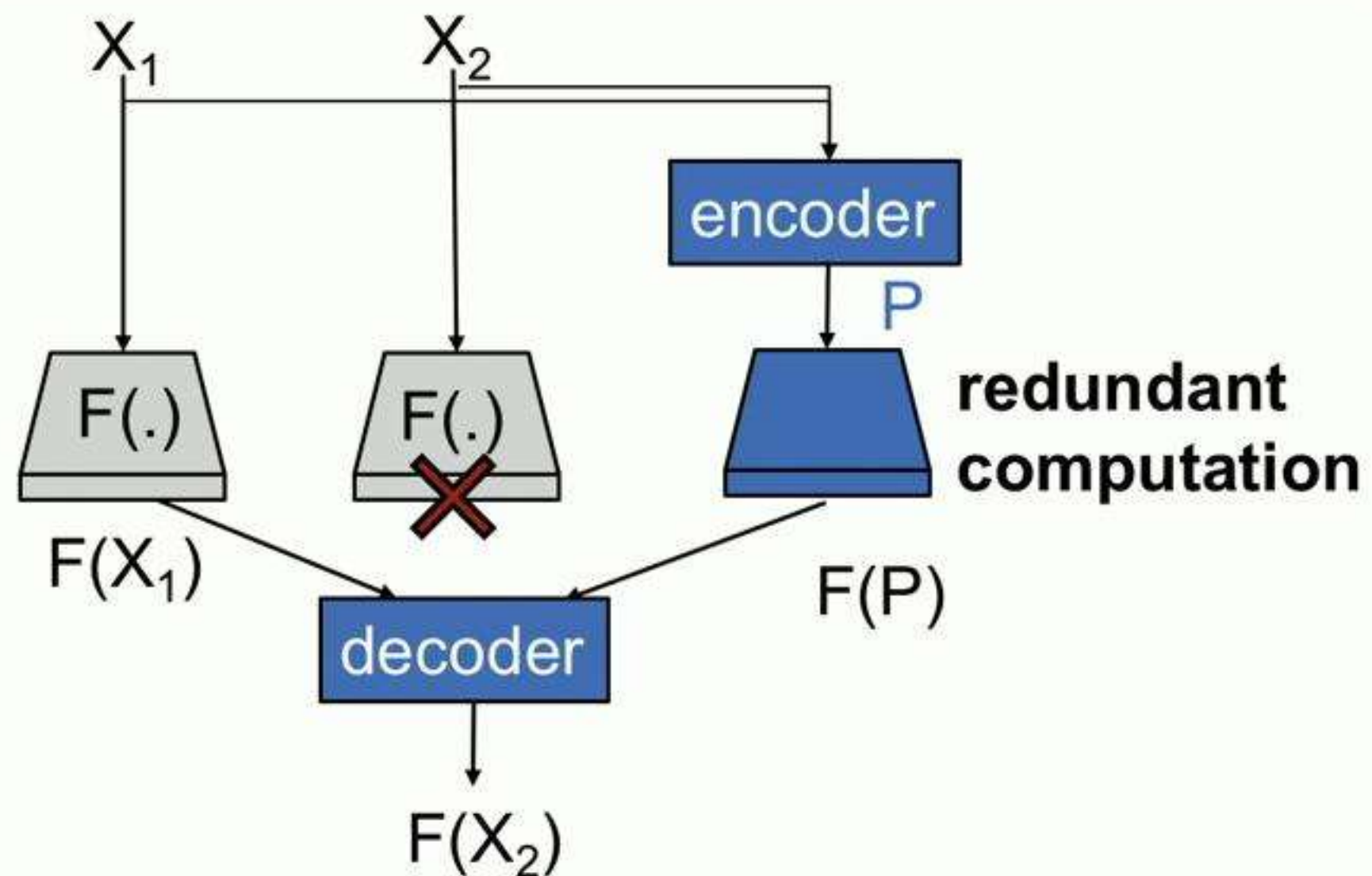
- Existing coded computation schemes applicable only to **limited class of functions:**
  - Linear functions and Polynomials**
- **Efficient schemes known only for linear functions**
  - Even polynomials of degree 2 require overhead more than full replication



# Our solution: **Use ML + Coding**

## Learning-based approach to coded computation

**Learn** components of the coded computation framework



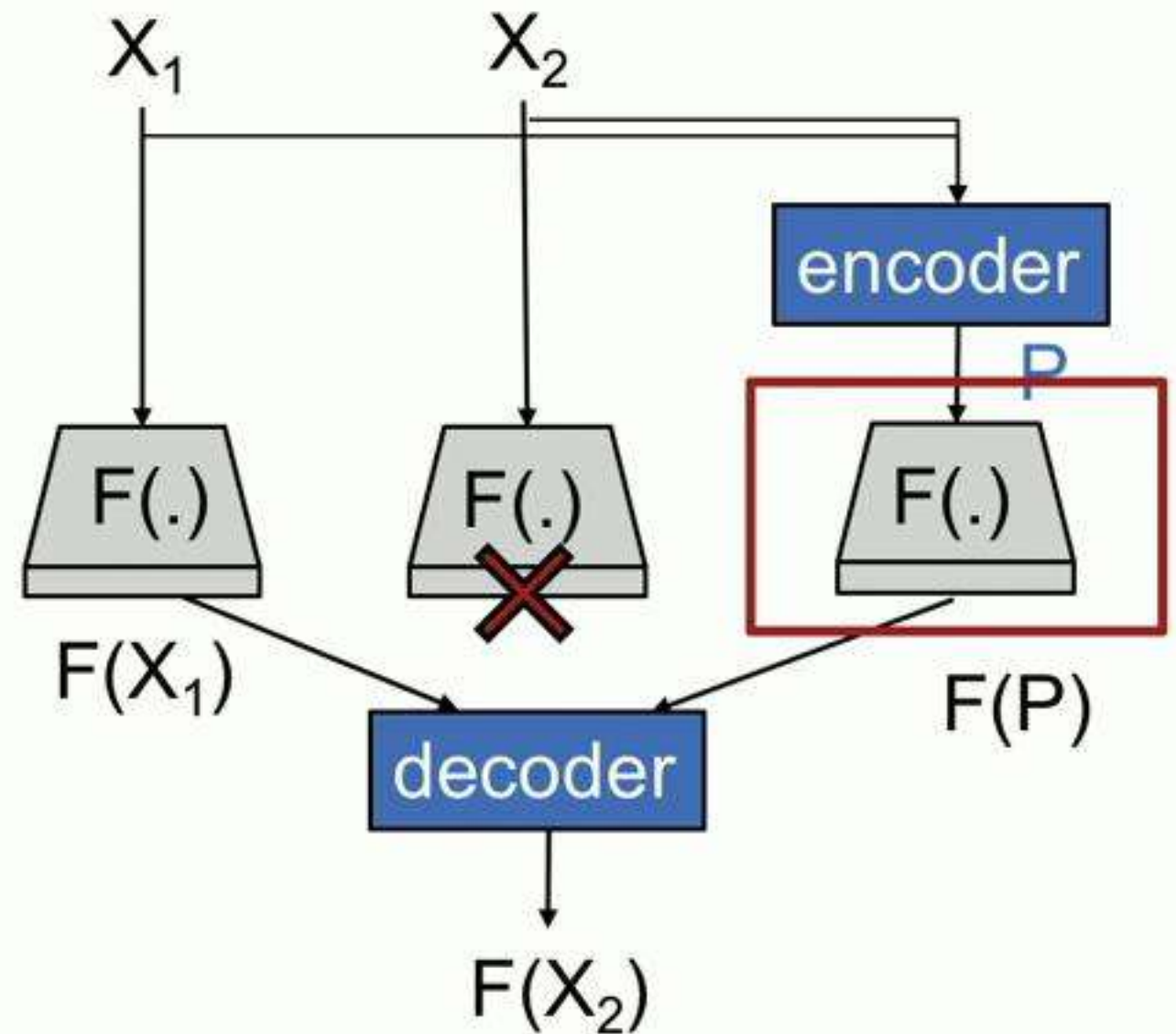


# Two learning-based approaches

---

## 1. Learning a code

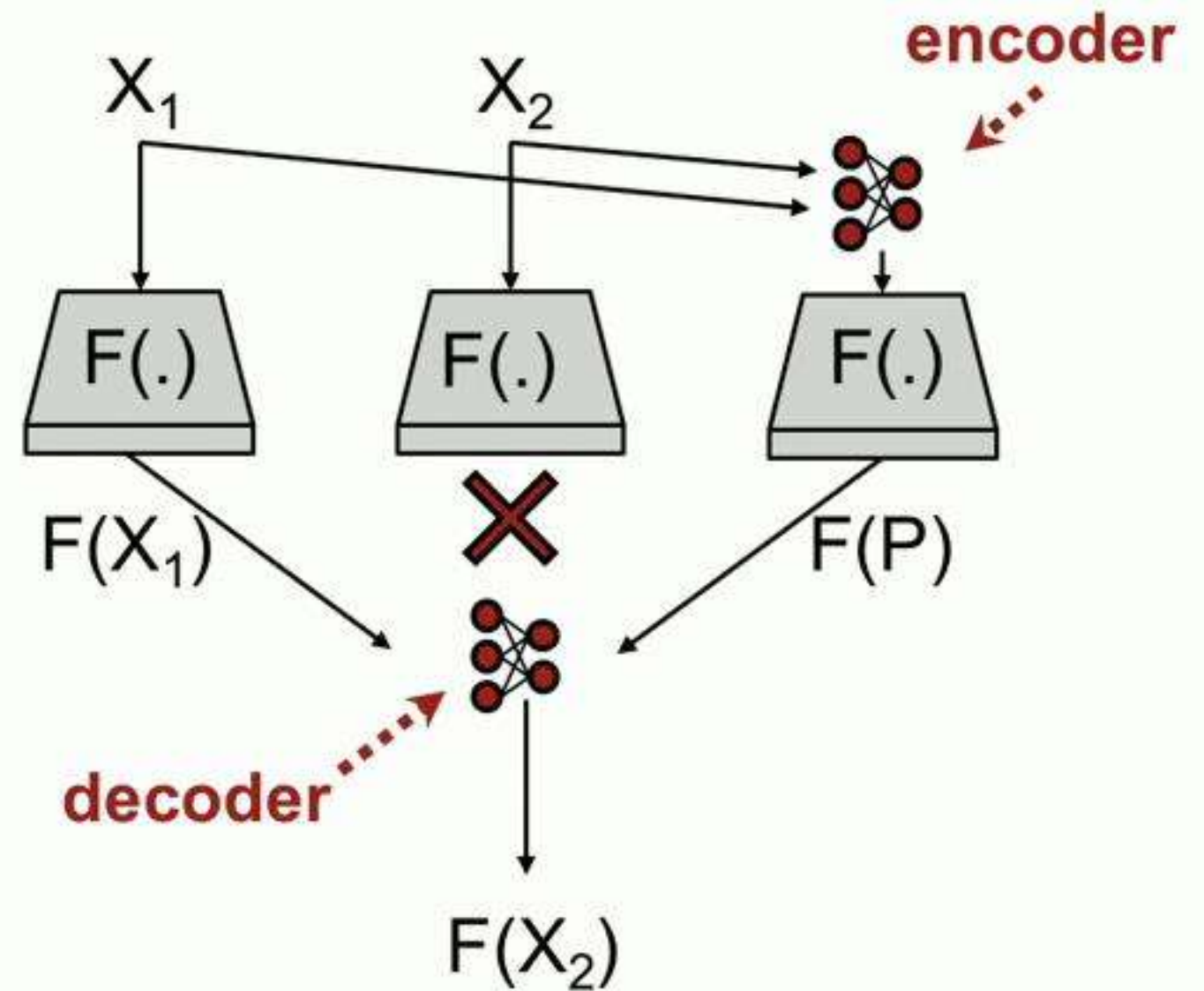
- Keep redundant computation identical to original
- Learn encoder and decoder using neural networks



"Learning a Code: Machine Learning for Approximate Non-Linear Coded Computation",  
J. Kosaian, K. V. Rashmi, S. Venkataraman, ArXiv June 2018.

# Learning a code

- Redundant computation same as original model
- “Learn” the encoder and decoder
- Using **neural networks** for enc/dec
- Encoder:
  - Input:  $k$  queries
  - Output:  $r$  “parity queries”
- Decoder:
  - Input:  $k$  predictions
  - Output:  $r$  unavailable predictions





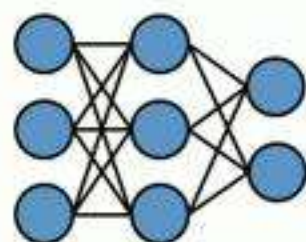
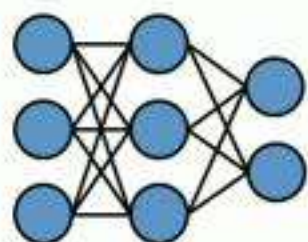
# Training encoder and decoder neural networks

- Training **dataset** same as the original model
- Mimic stragglers/failures by **artificially erasing** outputs from the model instances
- **Back-propagate through the original model**

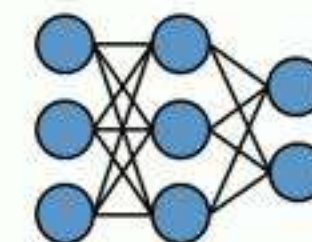
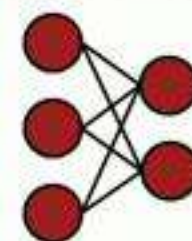
Example:

- $F(\cdot)$  = image classifier neural network
- $k = 2, r = 1$  (single parity query)



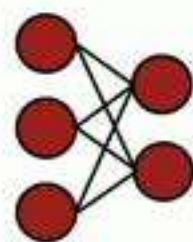


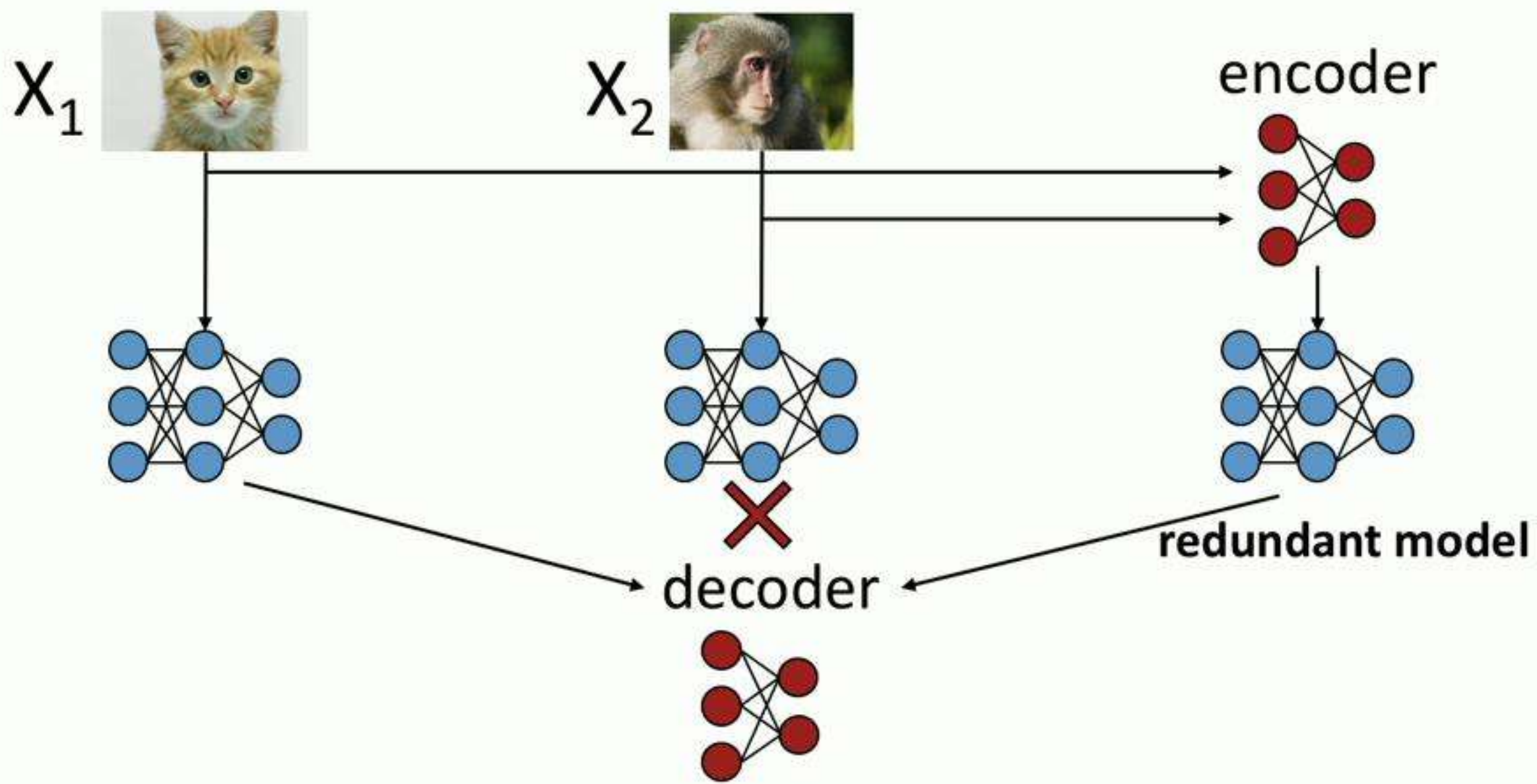
encoder

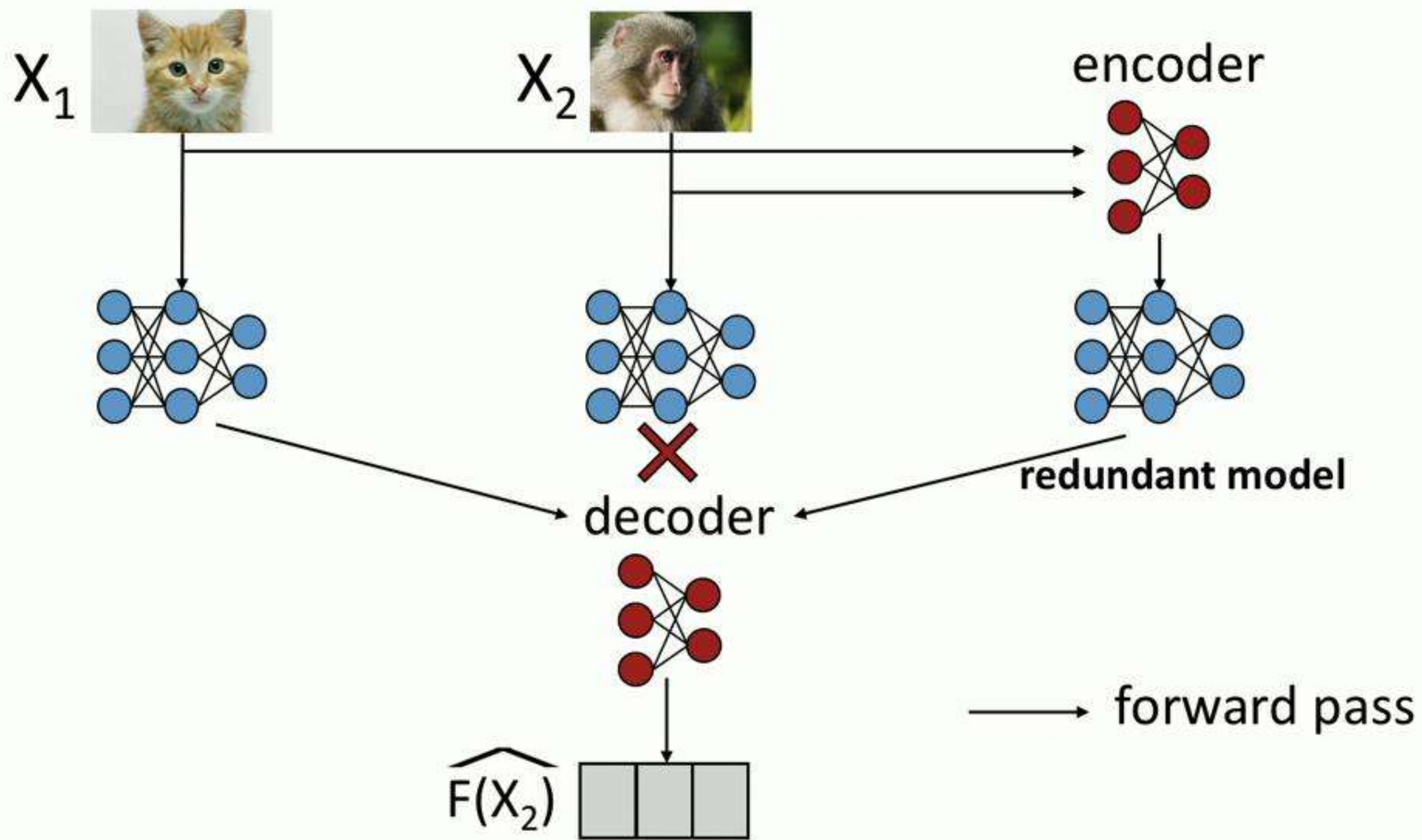


redundant model

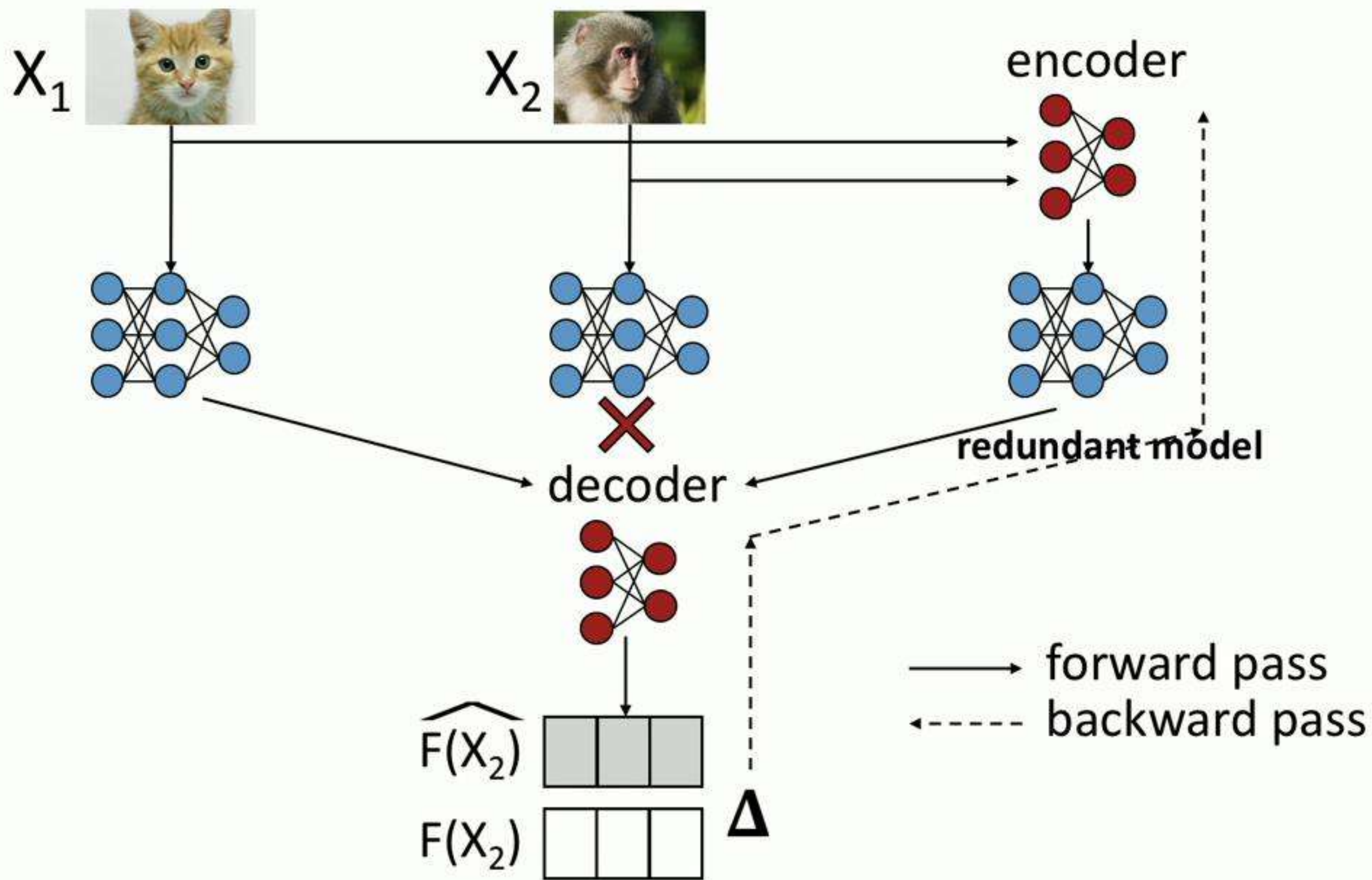
decoder











# Training encoder and decoder neural networks

- Training **dataset** same as the original model
- Mimic stragglers/failures by **artificially erasing** outputs from the model instances
- **Back-propagate** through the original model

Applicable for  
**any (numerically) differentiable function  $F$**



# “Learning a code” approach

---

- First coded-computation approach to **overcome the challenging barrier of non-linearity**
  - can handle general non-linear functions (any numerically differentiable function)
- First work to propose and enable use of coded-computation for **prediction serving systems**
  - prior works on coded-computation for ML primarily on training

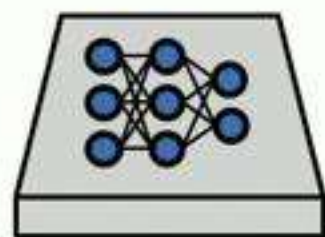


# Approach 1. “Learning a code” in action

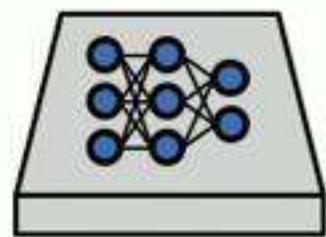
---



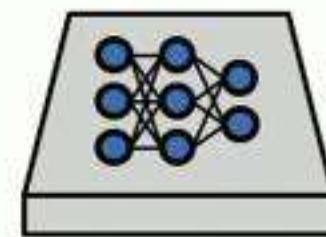
model  
instances



server 1

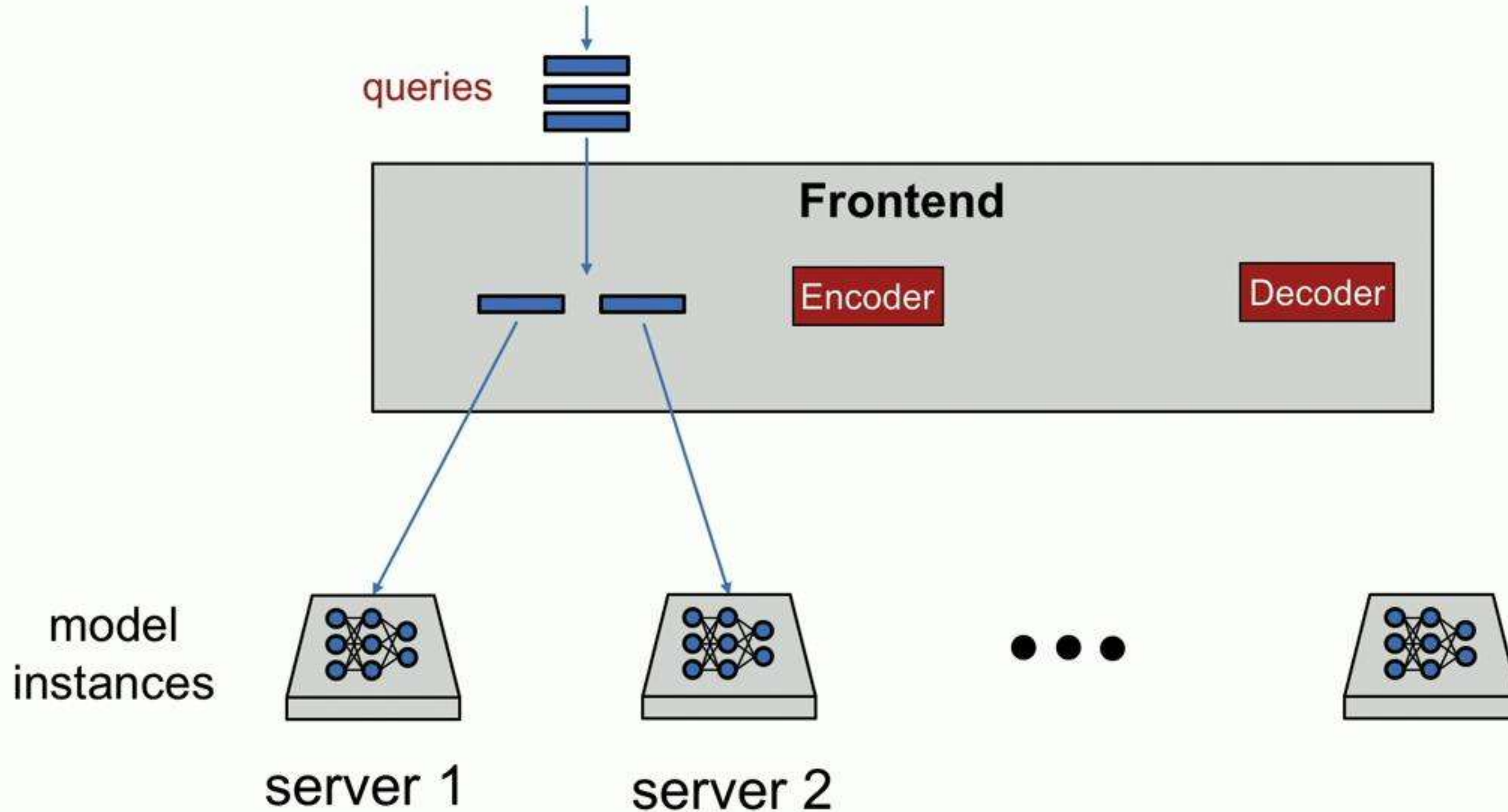


server 2



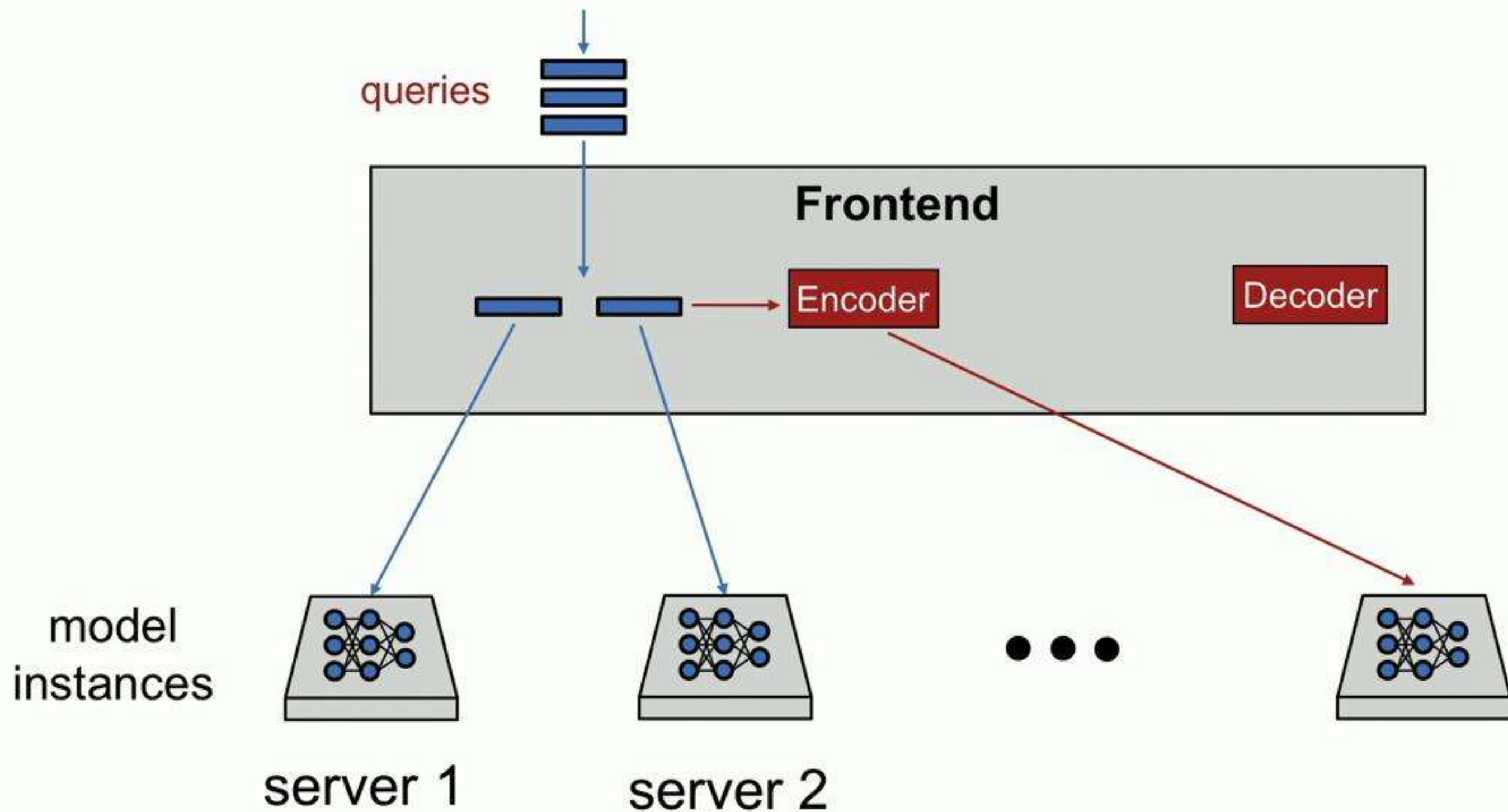
# Approach 1. “Learning a code” in action

---



# Approach 1. “Learning a code” in action

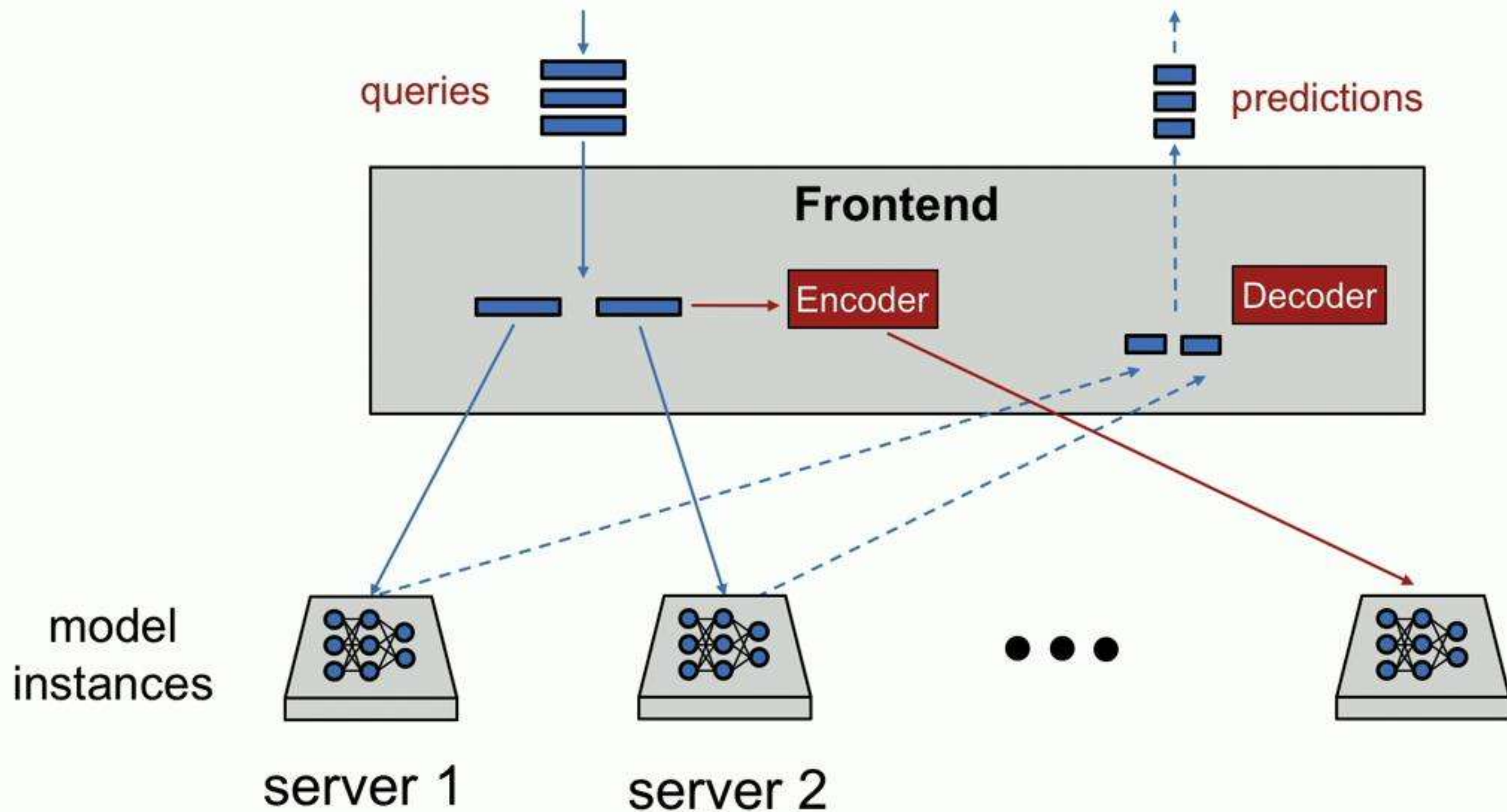
---



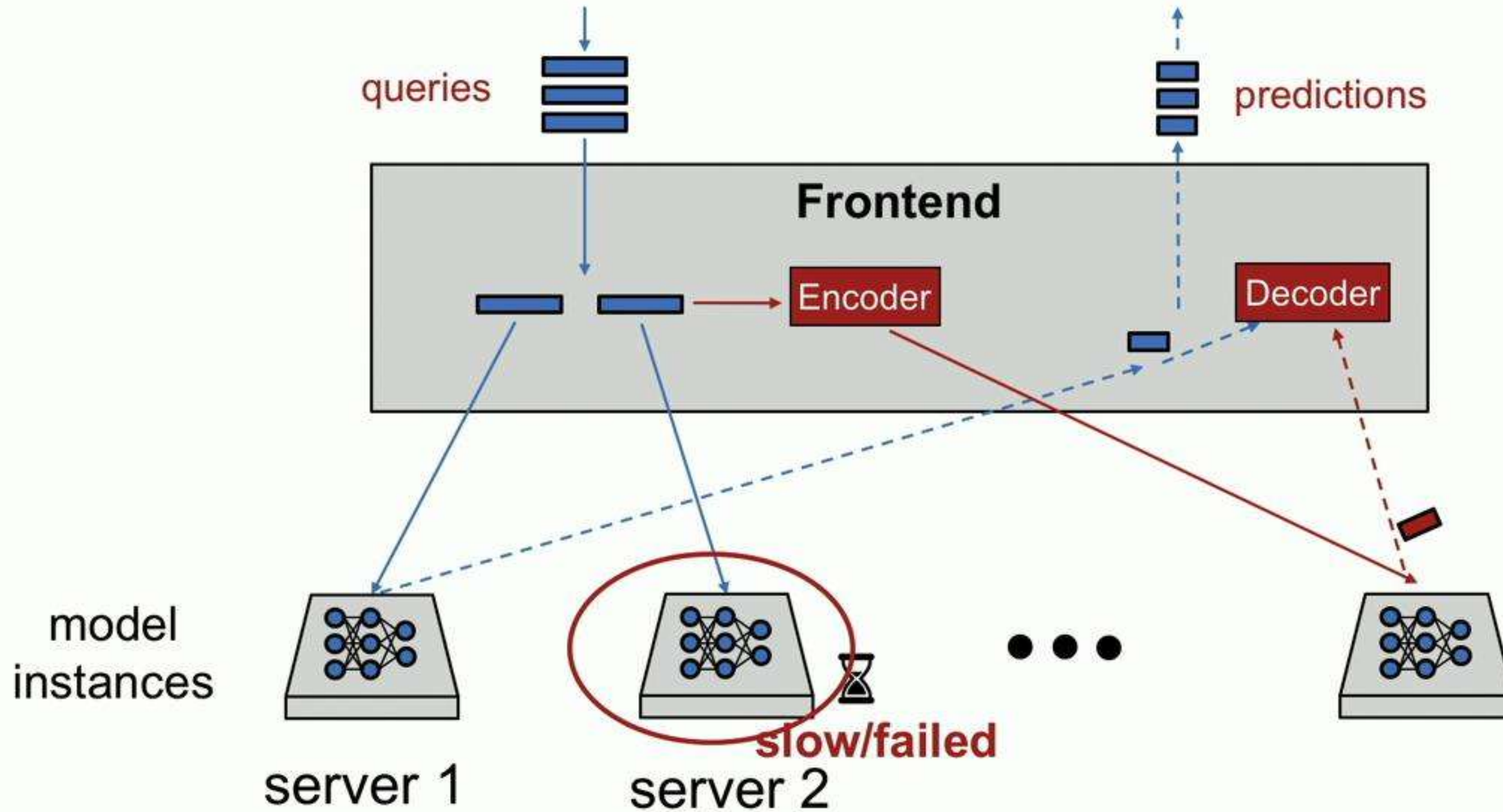


# Approach 1. “Learning a code” in action

---



# Approach 1. "Learning a code" in action



# Downsides of Approach 1

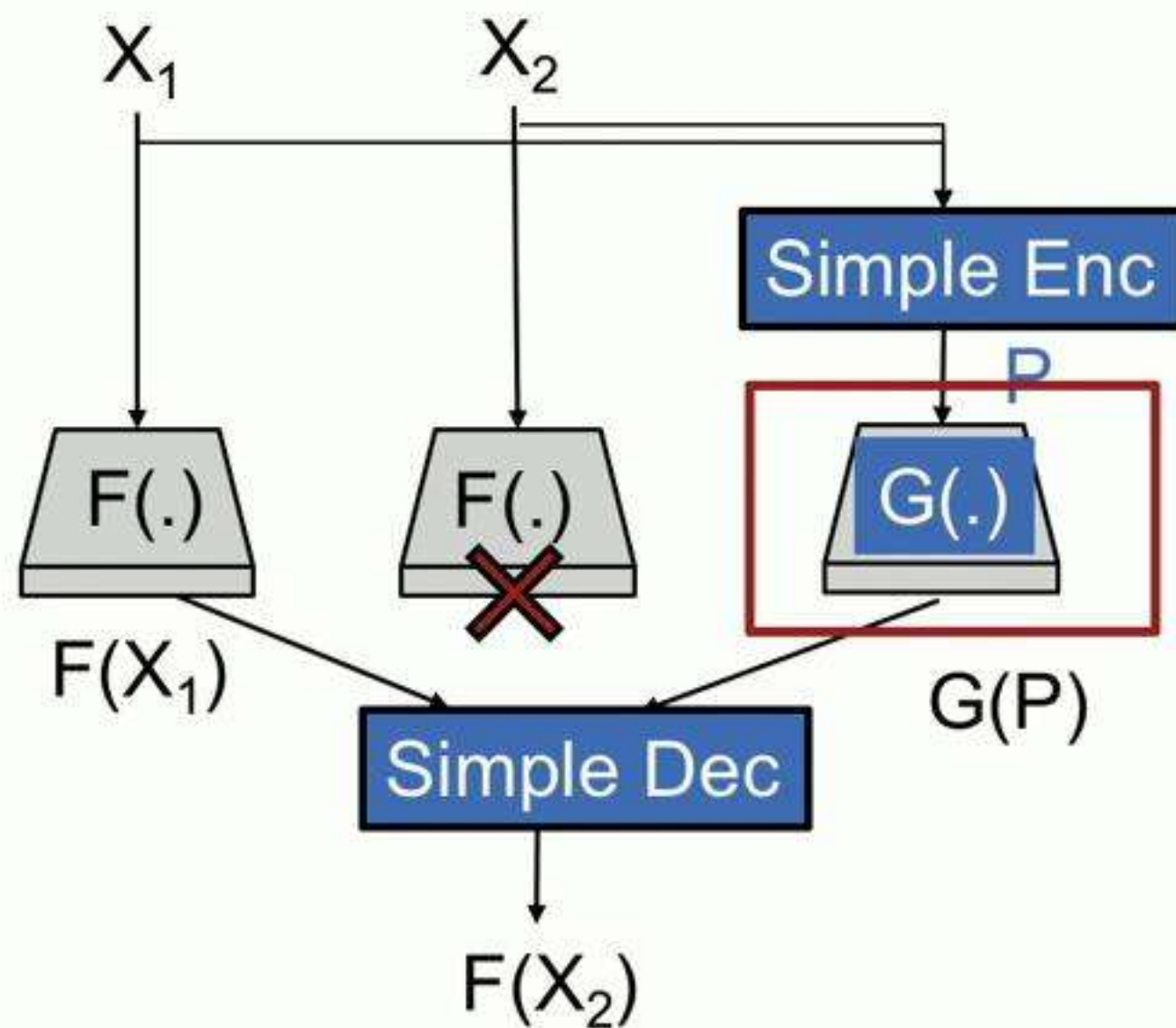
---

- Need a bulkier frontend
  - Neural network computation for encoding/decoding
- Reduced opportunity for improving tail latency
  - Redundant path takes longer time
  - Neural network encoding and decoding in addition to original model



# Approach 2. Learning “parity models”

- Simple encoder and decoder
  - **Generic:** E.g., Add/Subtract
  - **Task-specific:** E.g., Downsize & concatenate for image classification
- Learn a new redundant computation: “**parity model**”



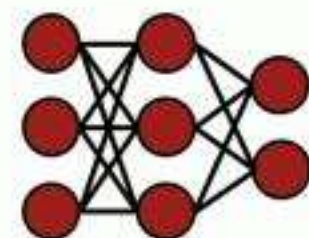
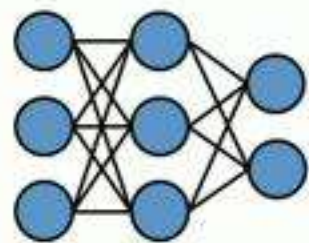
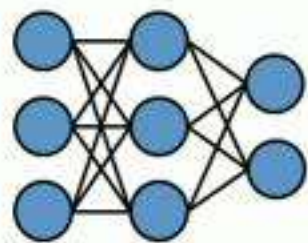
# Training parity models

---

- Training **dataset** same as the original model
- Mimic stragglers/failures by **artificially erasing** outputs from the model instances

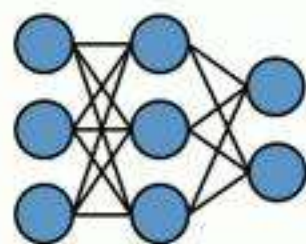
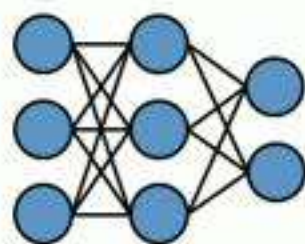
Example:

- $F(\cdot)$  = image classifier neural network
- $k = 2, r = 1$  (single parity query)
- Generic encoder/decoder: addition/subtraction

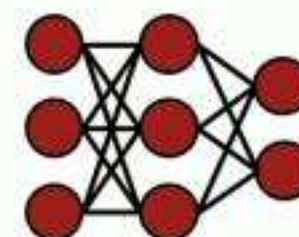


**parity model**



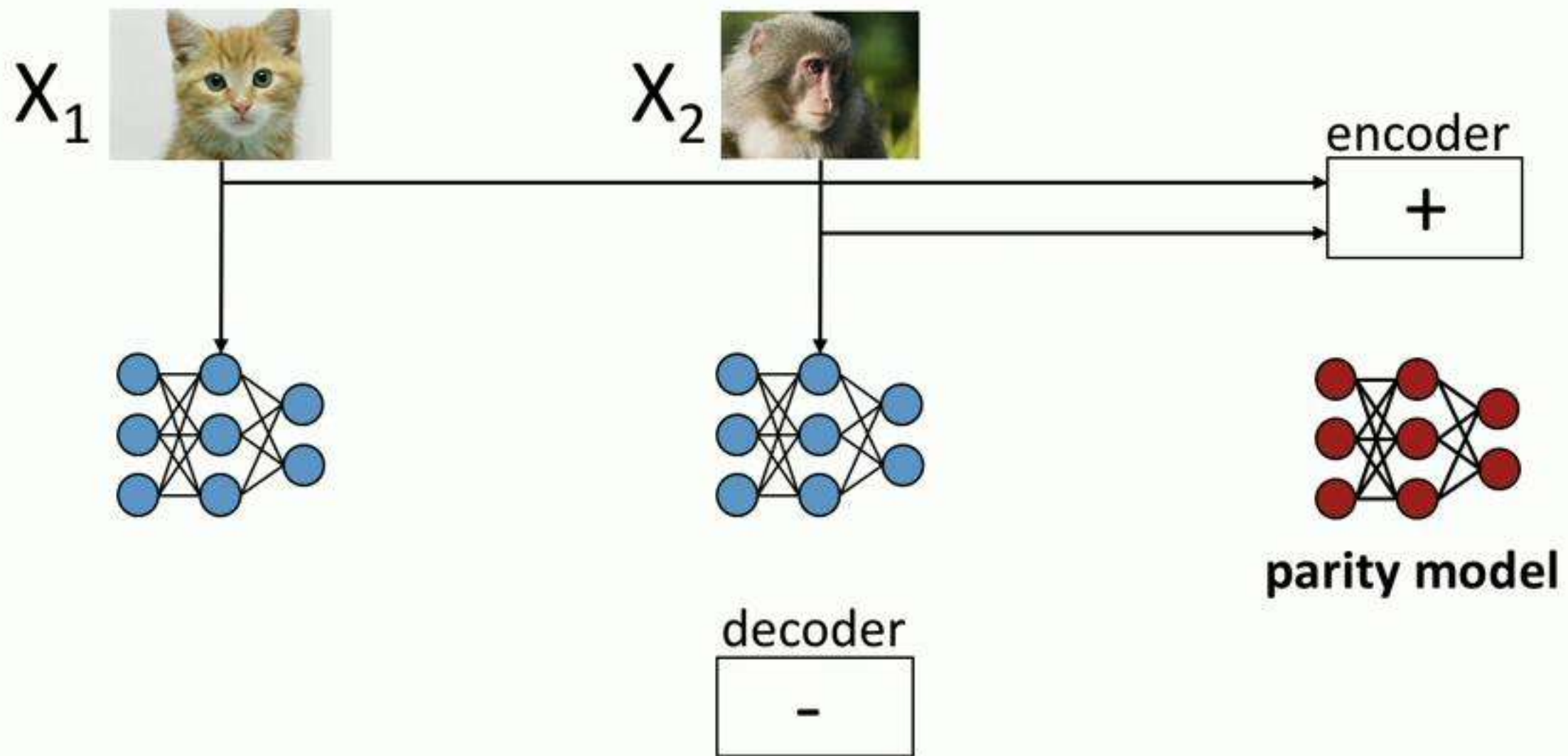


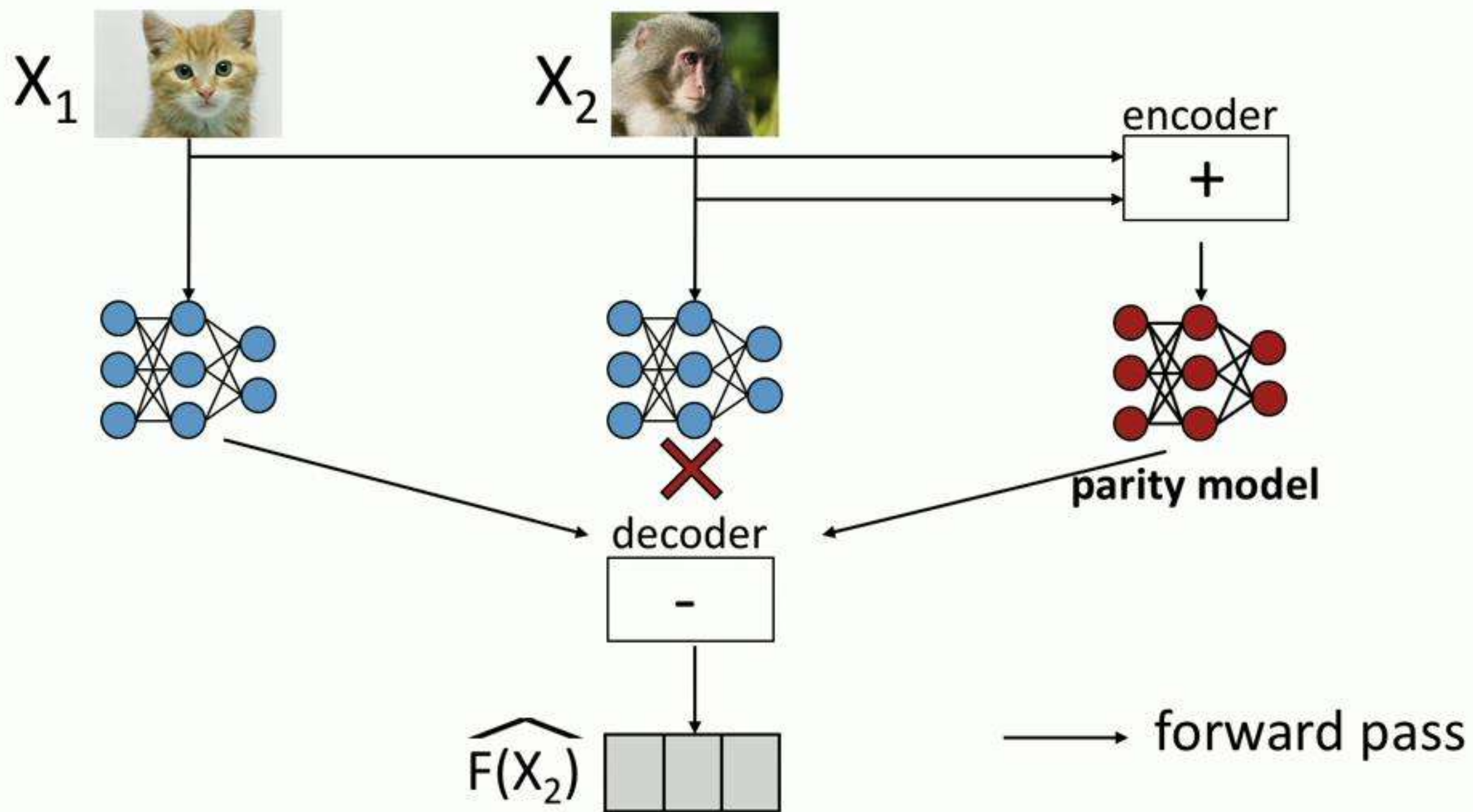
encoder  
+



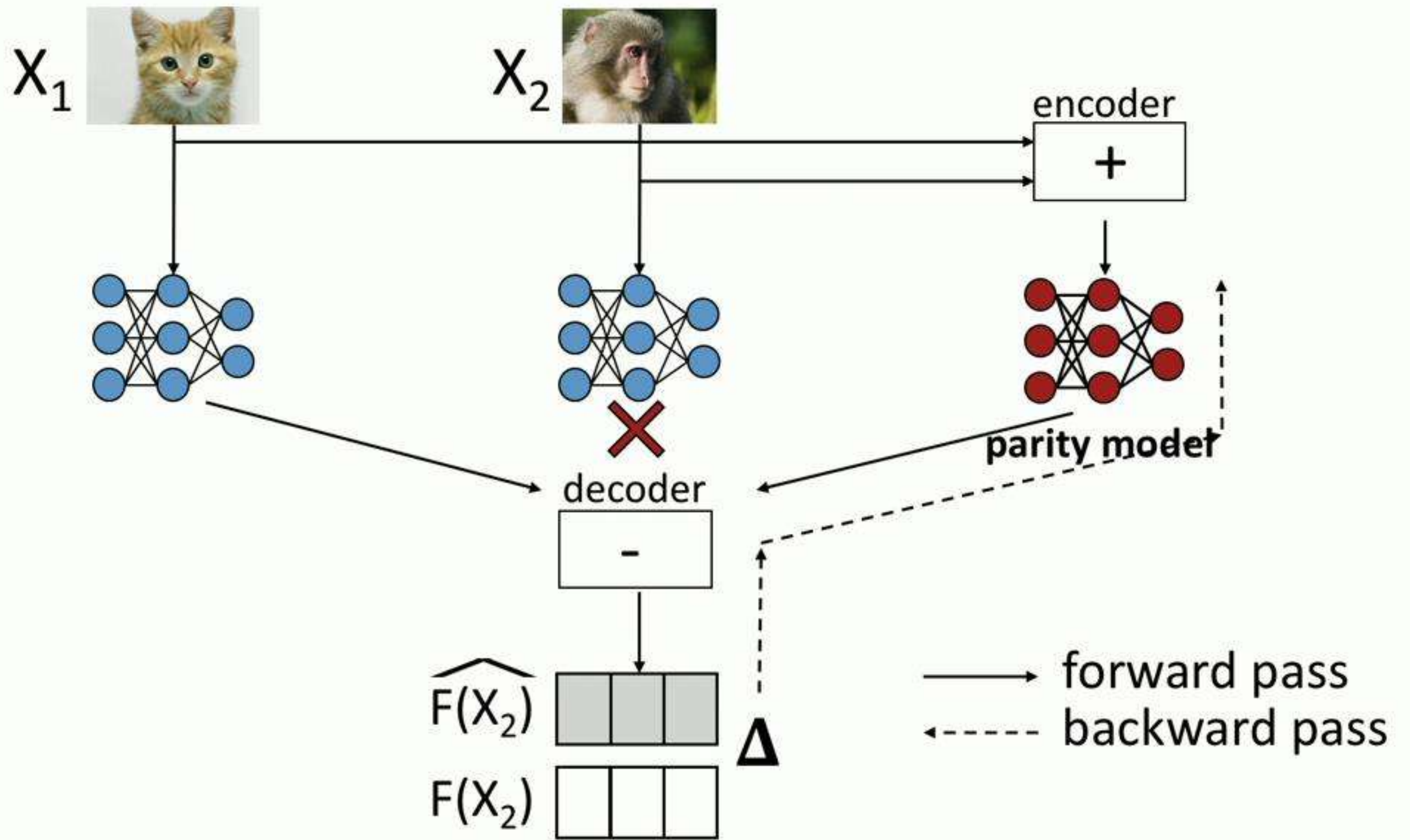
**parity model**

decoder  
-









# Implementation

---

- Approach 2 “Parity Models” (ParM) on top of Clipper



- Encoder and decoder in Clipper’s frontend
- Original and parity models:  
PyTorch models in Docker containers

# Evaluation of Accuracy



# Evaluation of Accuracy

---

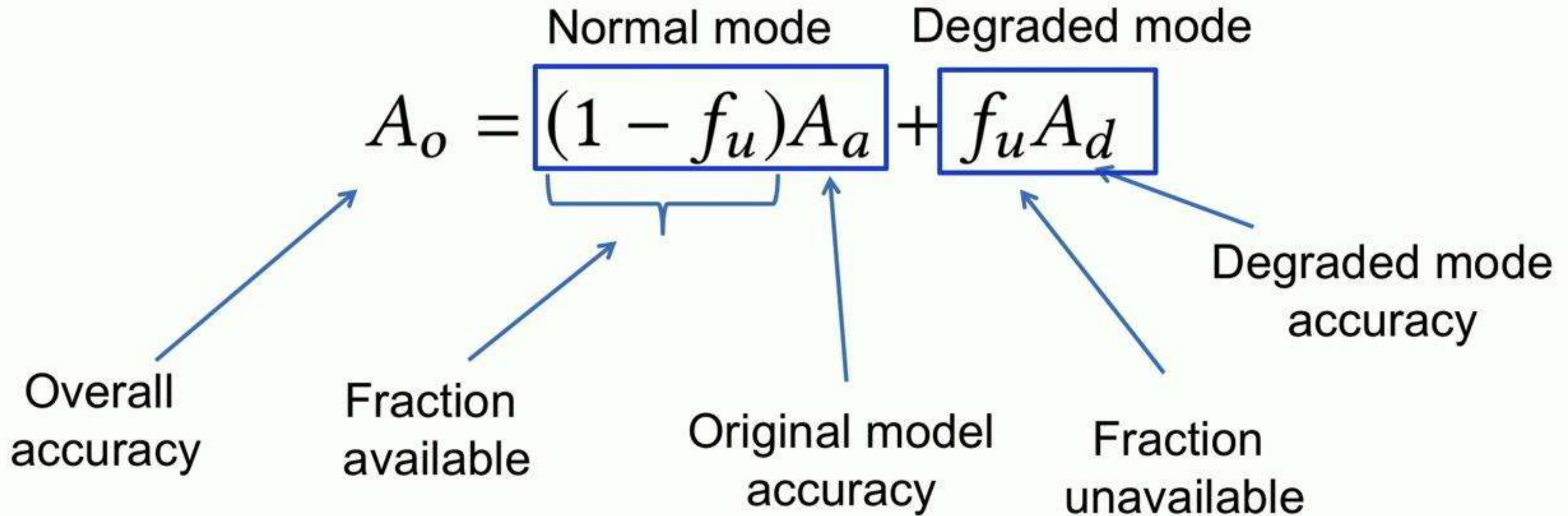
- Variety of popular inference tasks
  - Image classification, speech recognition, object localization



- Variety of popular neural network models
  - MLP, LeNet-5, VGG-11, ResNet-18, ResNet-152
- Encoder/Decoder:
  - **Generic:** Add/Sub  $\Rightarrow$  showing applicability to variety of inference tasks
  - **Inference-task specific:** Downsize and concatenate for image classification

# Evaluation of Accuracy

---



## Metrics:

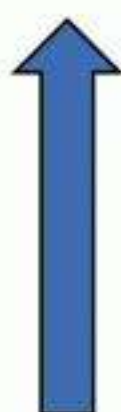
- Available vs Degraded-mode accuracy
- Overall accuracy



# Evaluation of accuracy: available vs degraded

Parameters:  $k = 2, r = 1 \Rightarrow 33\%$  resources for redundancy

Available ParM Degraded

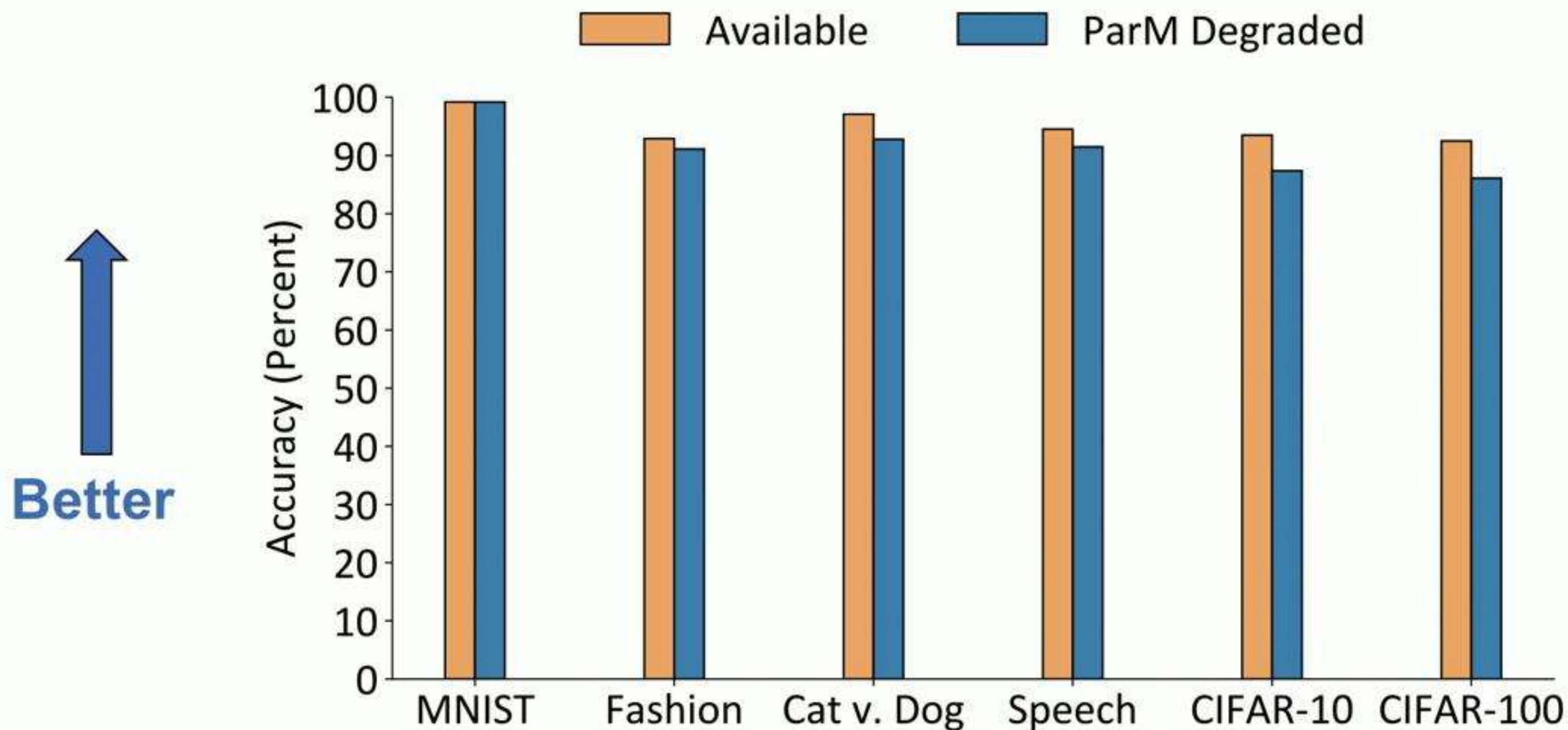
  
Better





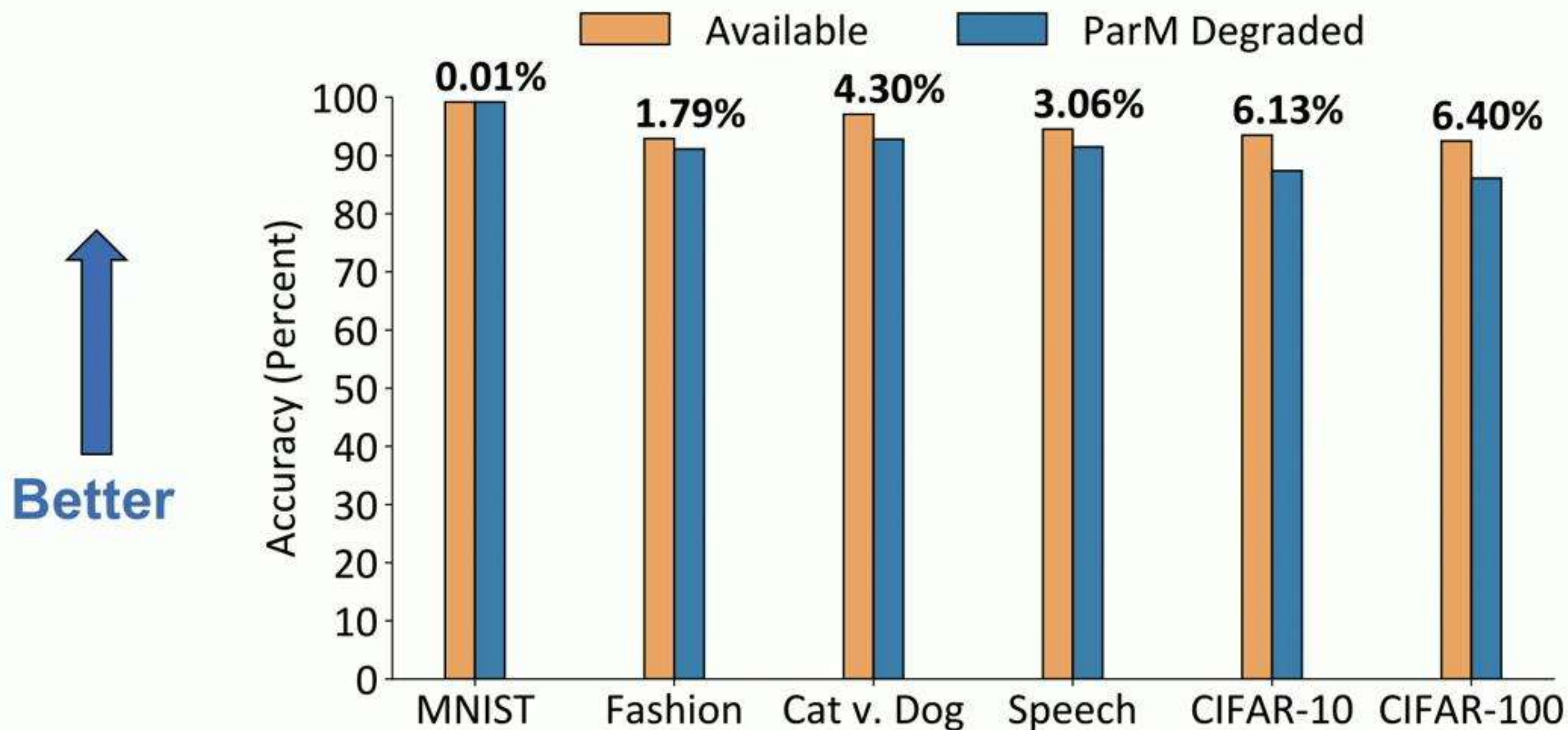
# Evaluation of accuracy: available vs degraded

Parameters:  $k = 2, r = 1 \Rightarrow 33\%$  resources for redundancy



# Evaluation of accuracy: available vs degraded

Parameters:  $k = 2, r = 1 \Rightarrow 33\%$  resources for redundancy



# Accuracy on object localization task

---





# Accuracy on object localization task

---

— Ground Truth

— Available

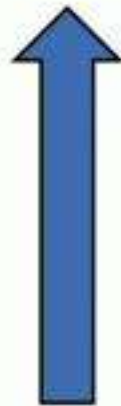
— ParM Degraded

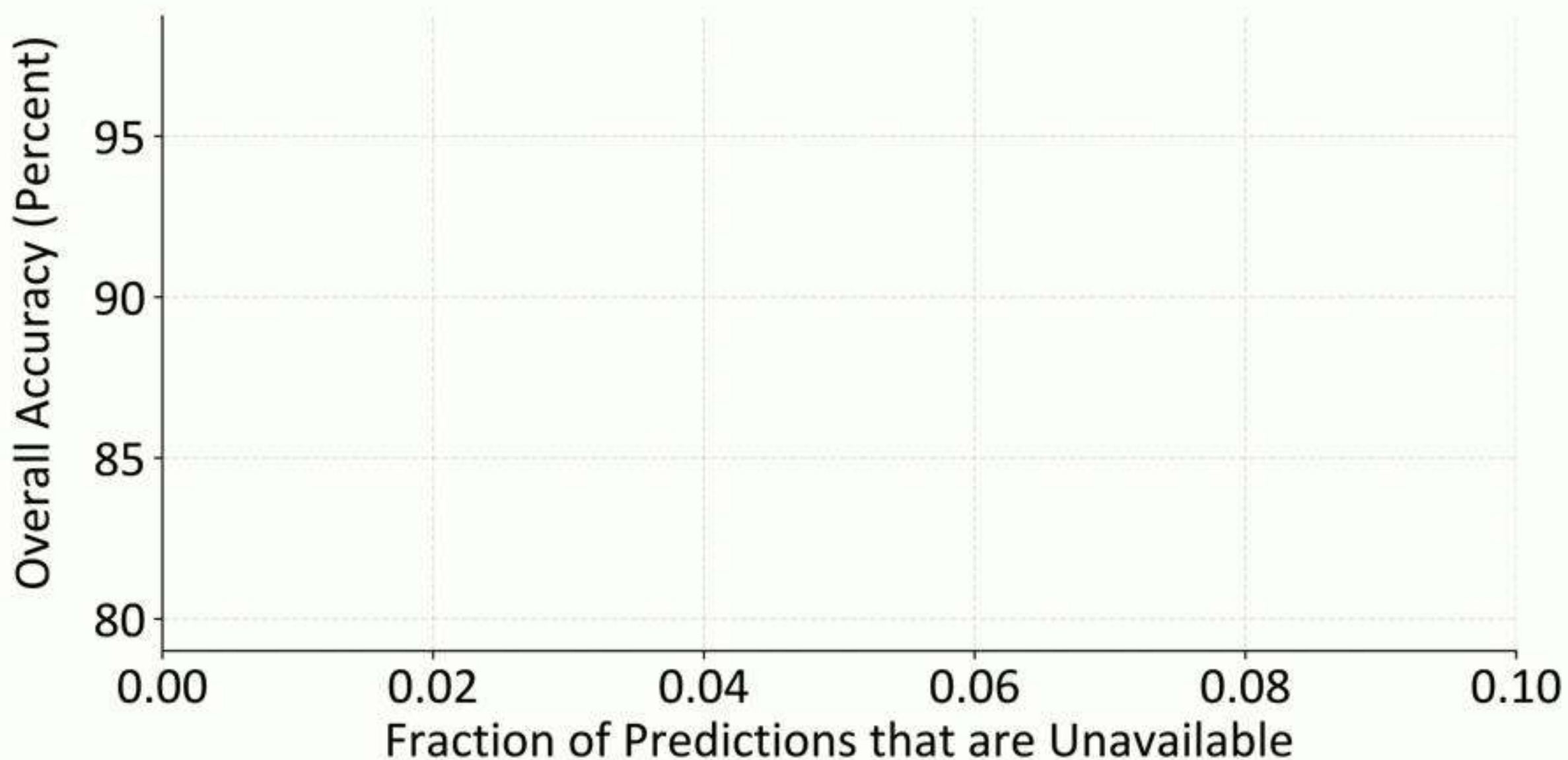


# Evaluation of Accuracy: Overall accuracy

---

- Inference task: Image classification on CIFAR 10

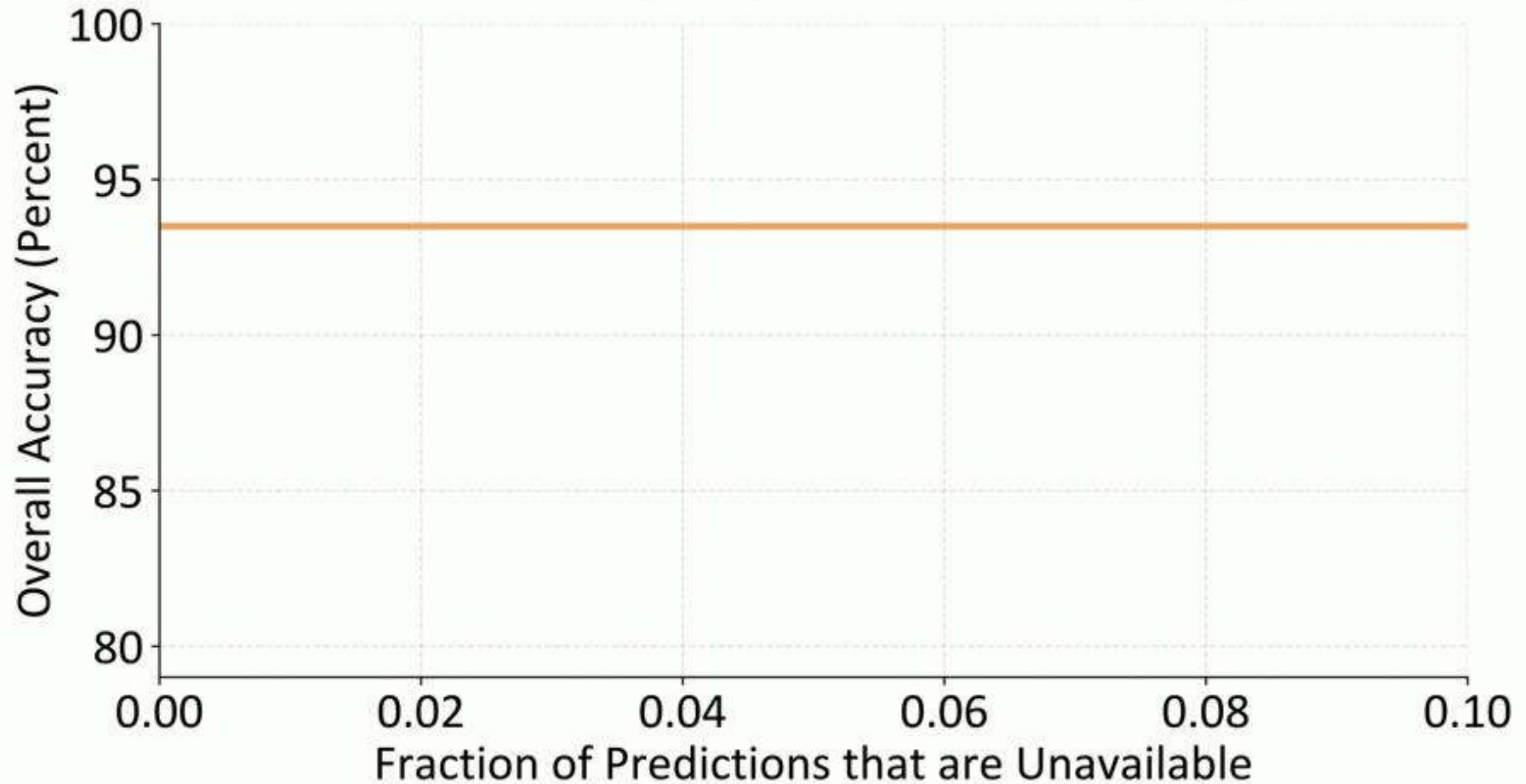
  
**Better**



# Evaluation of Accuracy: Overall accuracy

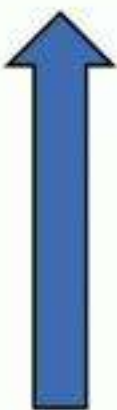
---

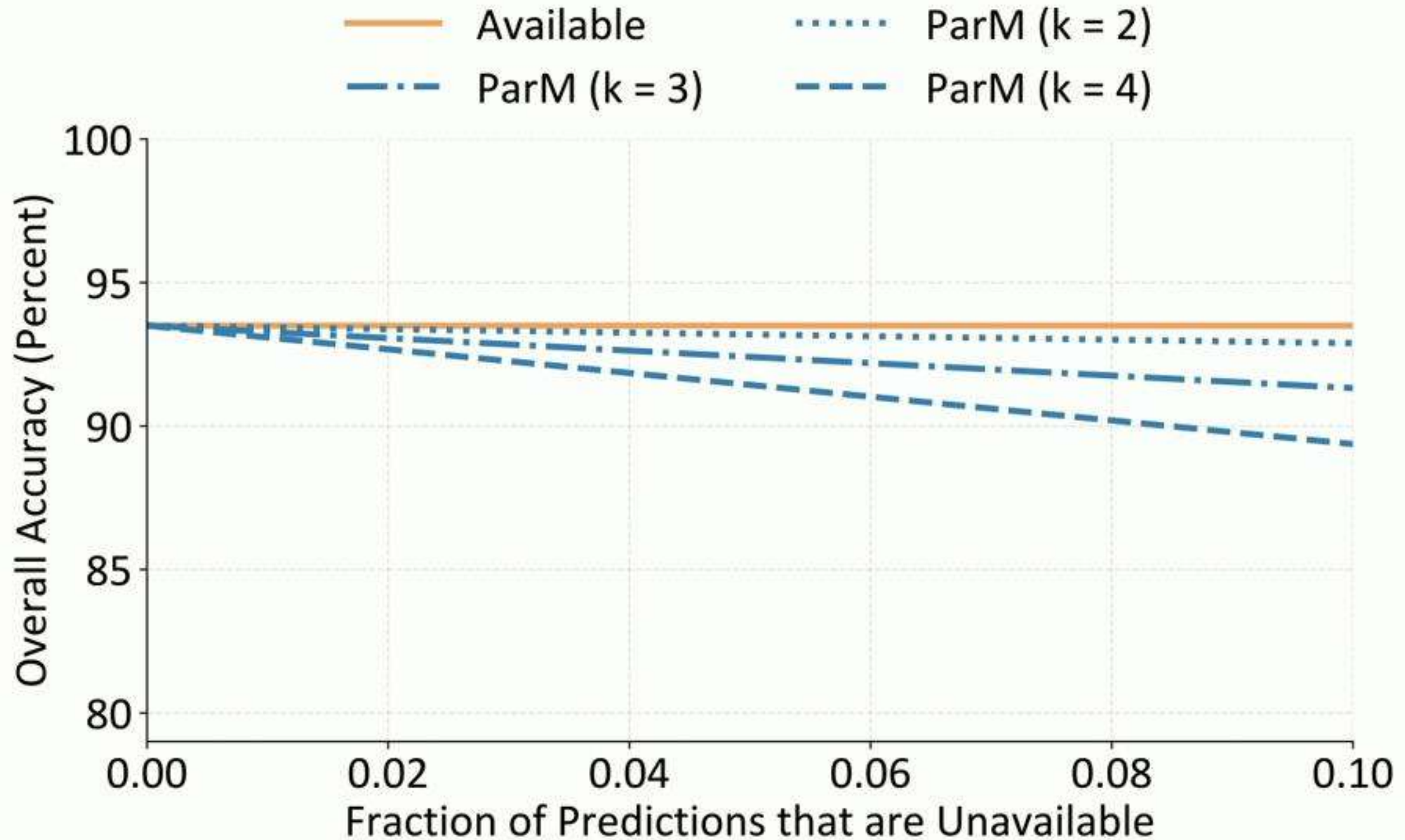
— Available      ····· ParM (k = 2)  
- · - ParM (k = 3)      - - - ParM (k = 4)





# Evaluation of Accuracy: Overall accuracy

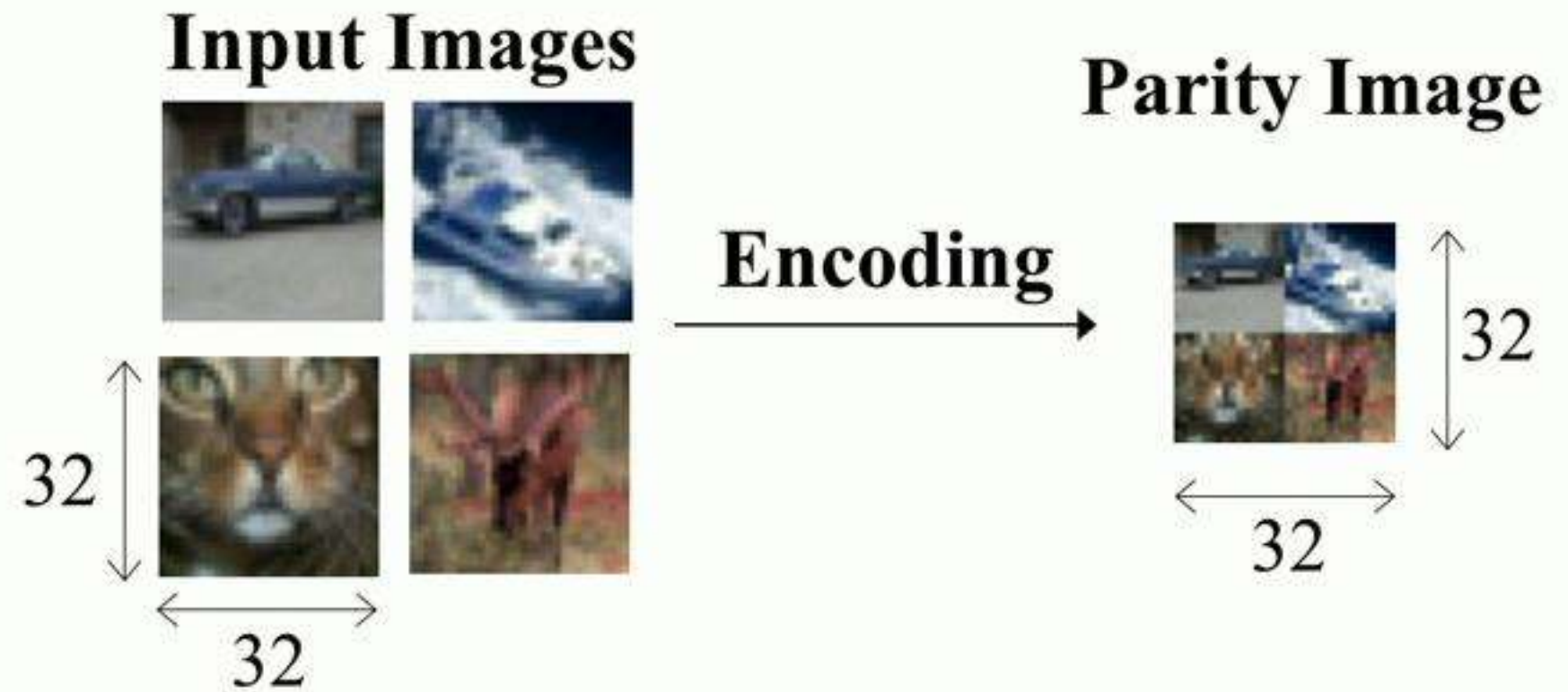
  
**Better**



# Inference task specific encoder & decoder

---

- Image classification task:
  - Downsize and concatenate
- Example:  $k = 4$ , CIFAR-10
- Helps improve accuracy



- Details in: "Parity Models: A General Framework for Coding-Based Resilience in ML Inference", J. Kosaian, K. V. Rashmi, S. Venkataraman, ArXiv May 2019.
- Concurrent work: "Collage inference: Tolerating stragglers in distributed neural network inference using coding", Narra et. al., ArXiv May 2019.
  - Builds on top of learning-based coded computation
  - Focusing on image classification



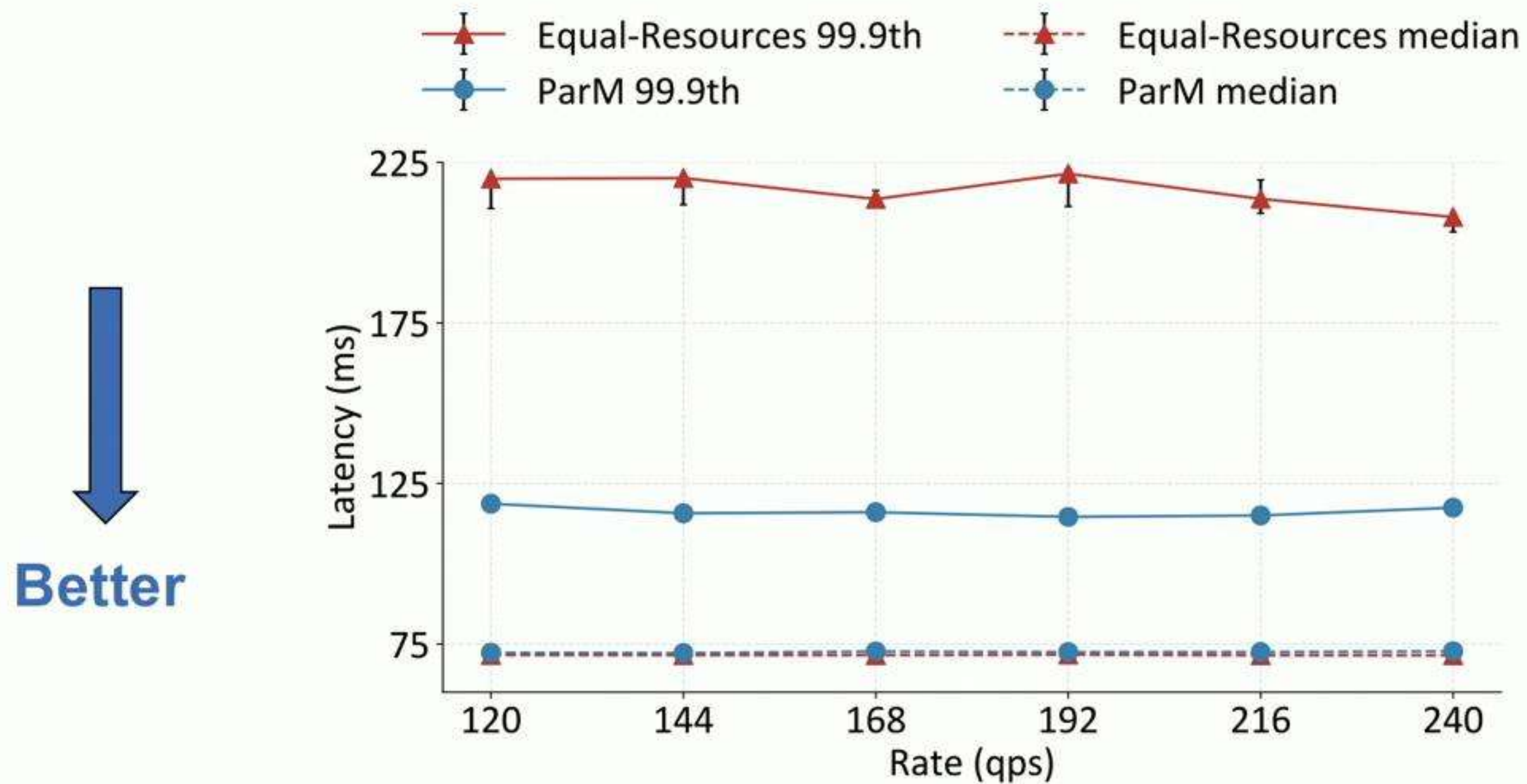
# Evaluation of Latency

---

- CPU and GPU clusters on EC2
  - CPU: 24 model instances
  - GPU: 12 model instances
- Baseline: “Equal-resources”
  - Same number of instances as ParM
  - Uses to deploy additional original models
- Varying query rates
- Varying background traffic
  - light inference multi-tenancy to few background transfers



# CPU cluster evaluation



# CPU cluster evaluation

---



- ~50% reduction in tail latency maintaining same median
- Brings 99.9<sup>th</sup> percentile 3.5x closer to median





# CPU cluster evaluation

---



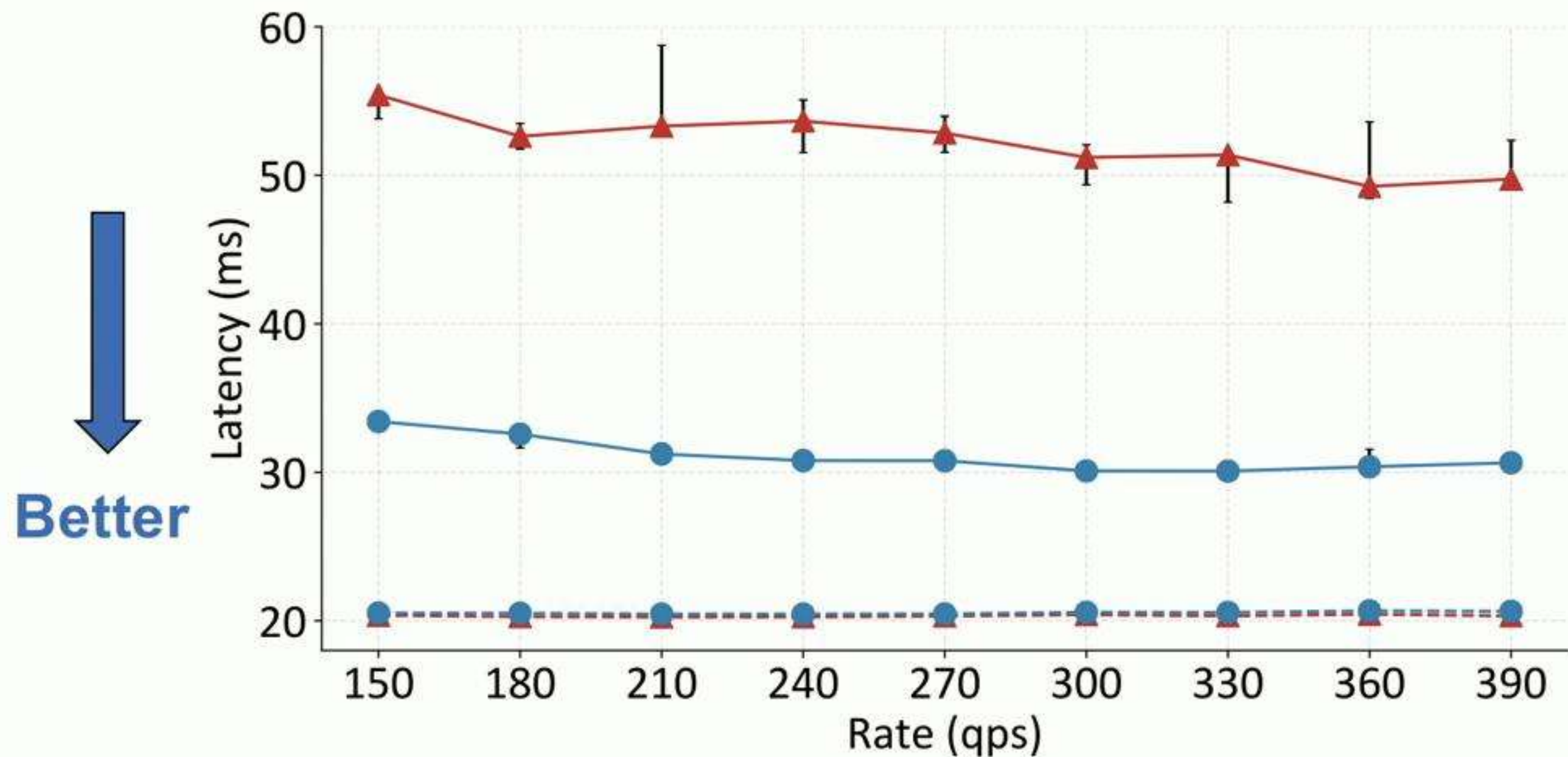
- ~50% reduction in tail latency maintaining same median
- Brings 99.9<sup>th</sup> percentile 3.5x closer to median

More predictable latency





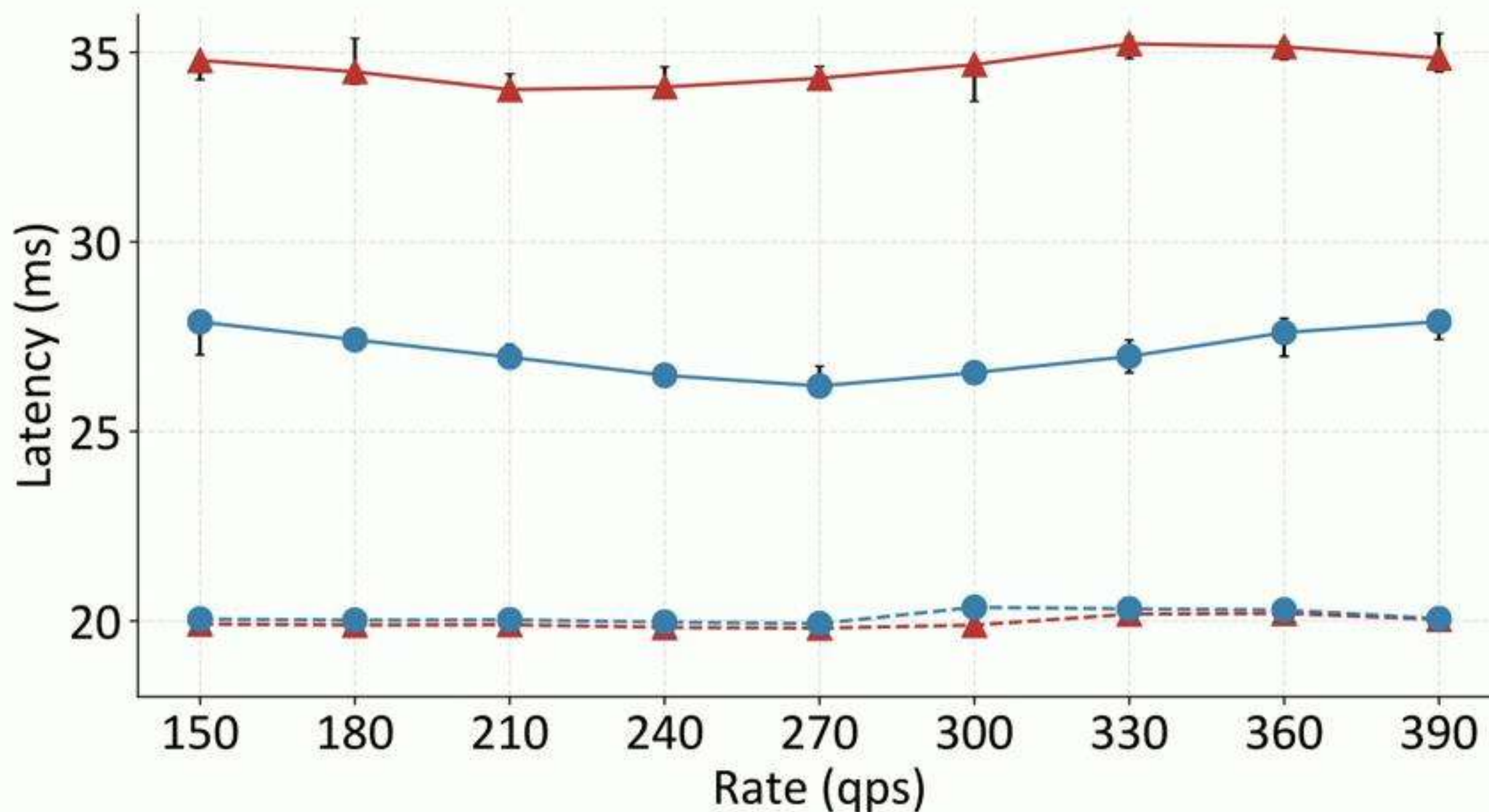
# GPU cluster evaluation



# Light multi-tenancy background



↓  
**Better**





# Summary (Part 1)

---

- Resilient ML **inference** via coded computation
- Challenge: handling **non-linearity of neural networks**
- Our solution: **learning-based approach** for coded computation
- Rich design space: encoder, decoder, parity models

- Applicable to variety of inference tasks



- Implementation on Clipper & evaluation
  - Significantly better degraded-mode accuracy; no loss in normal-mode
  - More predictable latency: 99.9<sup>th</sup> tail latency 3.5x closer to median



# Future work (Part 1)

---

- Improve **training strategy** for learning-based coded computation
  - New learning tasks
  - Current: random choice of data samples
  - Intelligent choice of samples to combine?
- Explore **rich design space of parity models** framework
  - Design better generic encoder-decoder
  - Design task-specific encoder-decoder for various popular tasks
  - Explore better choices for parity models
- Explore application to **other workloads/systems**
  - Fundamentally a new approach for redundant computation

# Part 2.

# Resource-efficient Redundancy in Cluster Storage Systems

“Cluster storage systems gotta have HeART: improving storage efficiency by exploiting disk-reliability heterogeneity”

Saurabh Kadekodi, K. V. Rashmi, and Greg Ganger

*USENIX FAST 2019*



# Joint work with

---



**Saurabh Kadekodi**



**Greg Ganger**

Carnegie Mellon University



# Cluster storage systems

---

- Storage subsystem of distributed systems



- Thousands to millions of disks
- Built incrementally according to demand

# Cluster storage system reliability

---

- Failures are common
  - Disk failures measured as **annualized failure rates (AFR)**
  - AFR => expected % of disk failures in a year
- Popular fault tolerance mechanism: redundancy
  - Full data replication
  - **Erasure coding**



# Redundancy in storage systems

---

Erasure coding example:  $(n=14, k=10)$  code

a b c d e f g h i j



a b c d e f g h i j P1 P2 P3 P4

data blocks

parity blocks

distributed on disks  
across servers (across failure domains)





# Redundancy configuration in storage systems

---

- Amount of redundancy
  - Function of the erasure code parameters, “n” and “k”
  - Example (n=14, k=10): 1.4x redundancy
- Chosen to meet **durability** and **availability** requirements
  - Mean Time To Data Loss (MTTDL) & reconstruction constraints
  - **Based on (average) failure rate across disk fleet**
- Chosen at the time when data is erasure coded
  - Not modified thereafter

# Redundancy configuration in storage systems

---

- Amount of redundancy
  - Function of the erasure code parameters, “n” and “k”
  - Example (n=14, k=10): 1.4x redundancy

Current redundancy configuration approaches are  
**“Static”**

- Chosen at the time when data is erasure coded
  - Not modified thereafter

**However...**

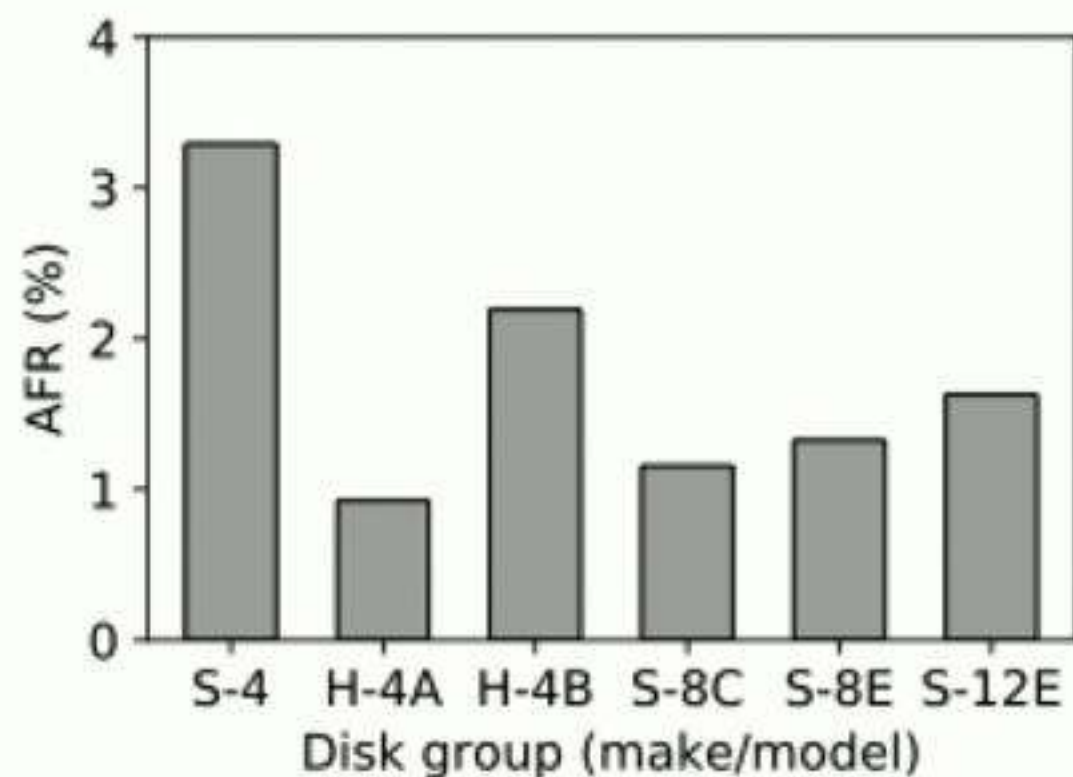
**The failure rates of disks are *\*not\** static**



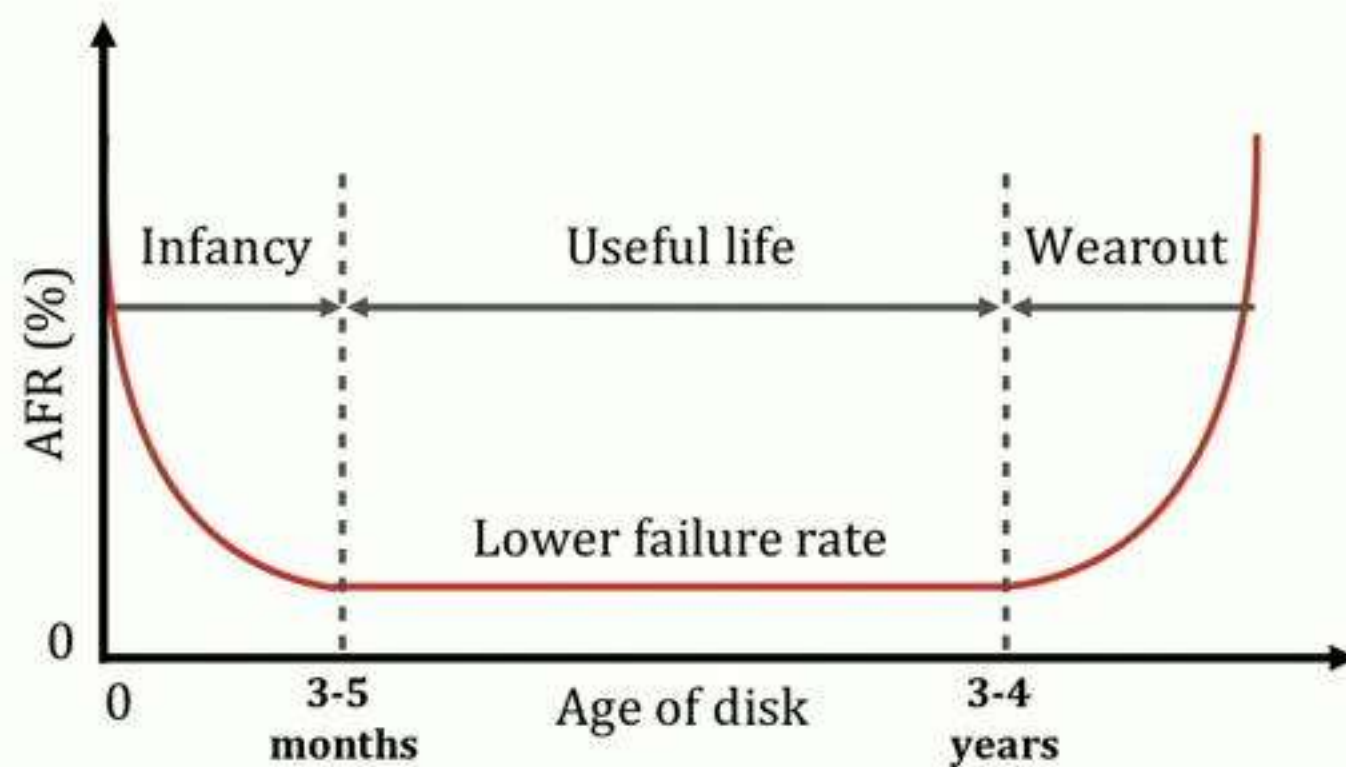
# Disk failure rates are highly variable

- Our study on failure data from production clusters at Backblaze
- Failures rates vary significantly

## Variation across disk families



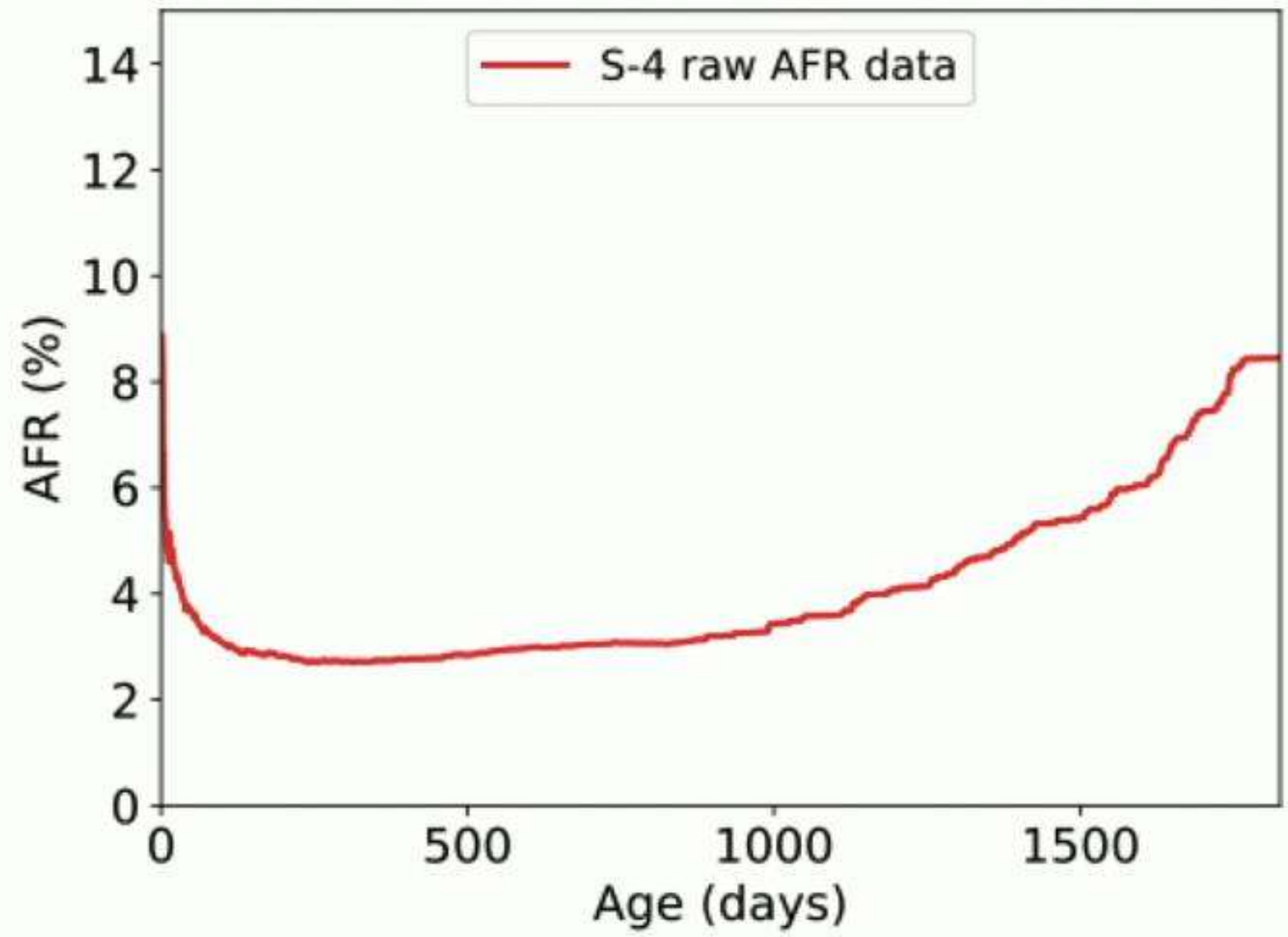
## Variation over time as disk ages



“Cluster storage systems gotta have HeART: improving storage efficiency by exploiting disk-reliability heterogeneity”, Saurabh Kadekodi, K. V. Rashmi, and Greg Ganger, USENIX FAST 2019

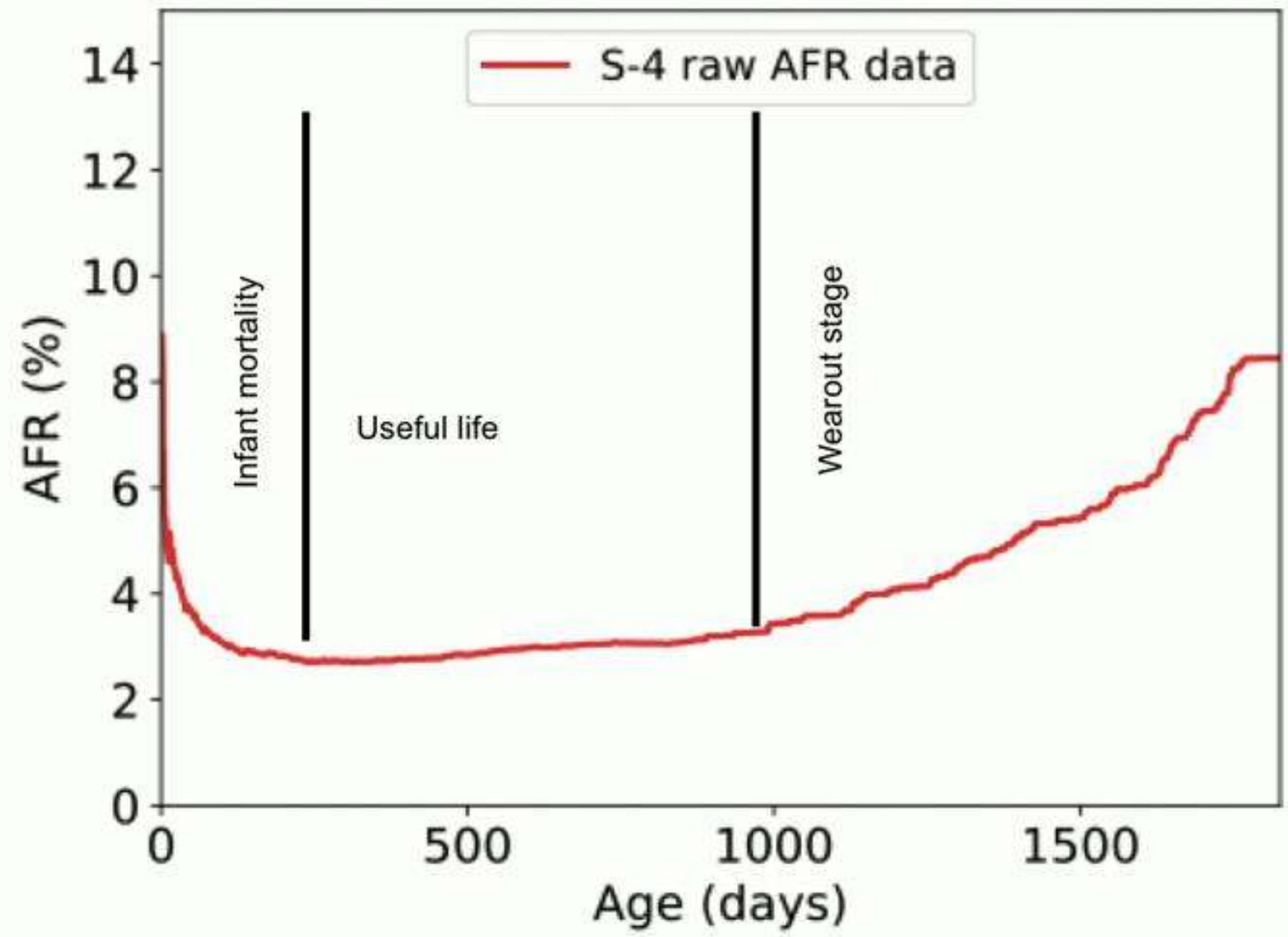
# Failure rate variation over time

---



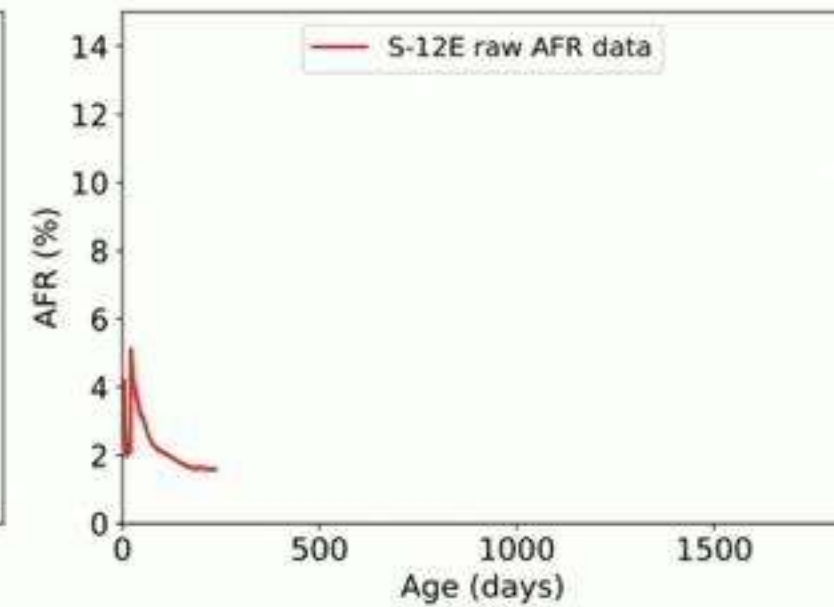
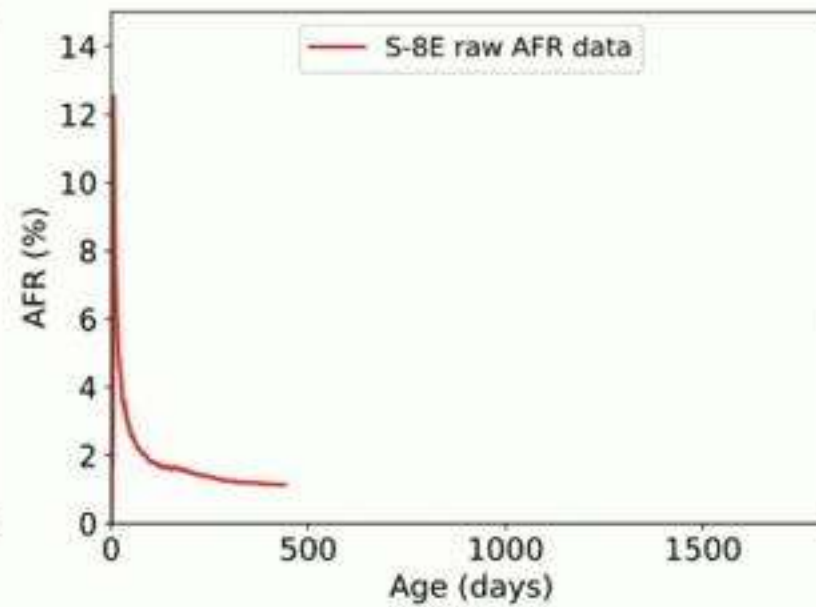
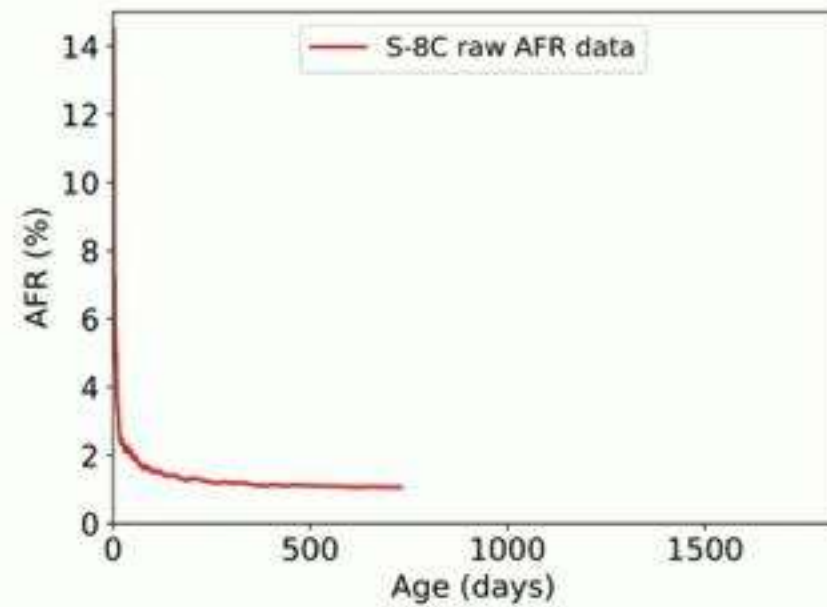
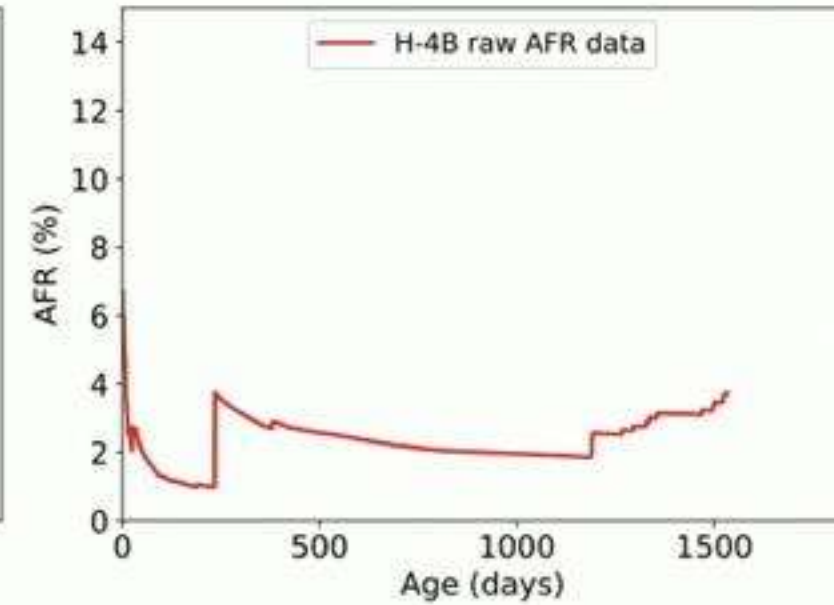
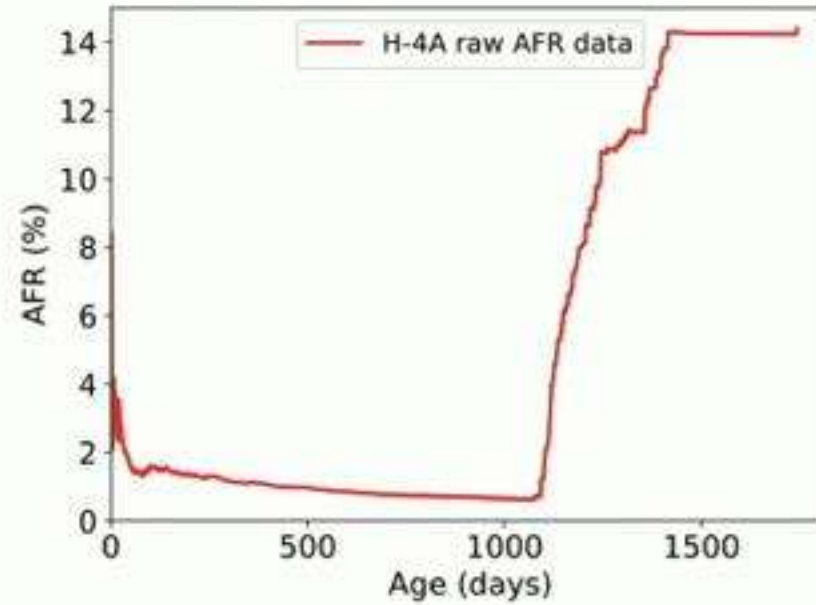
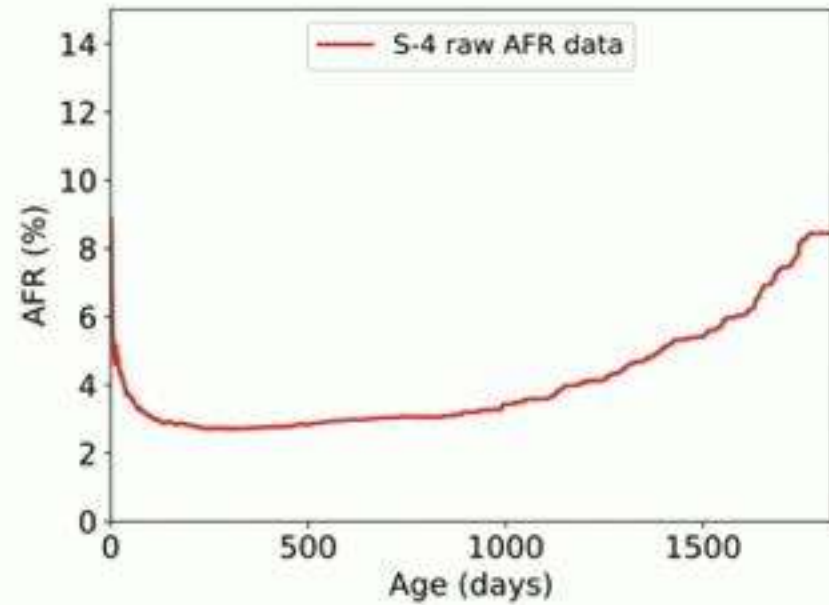
# Failure rate variation over time

---

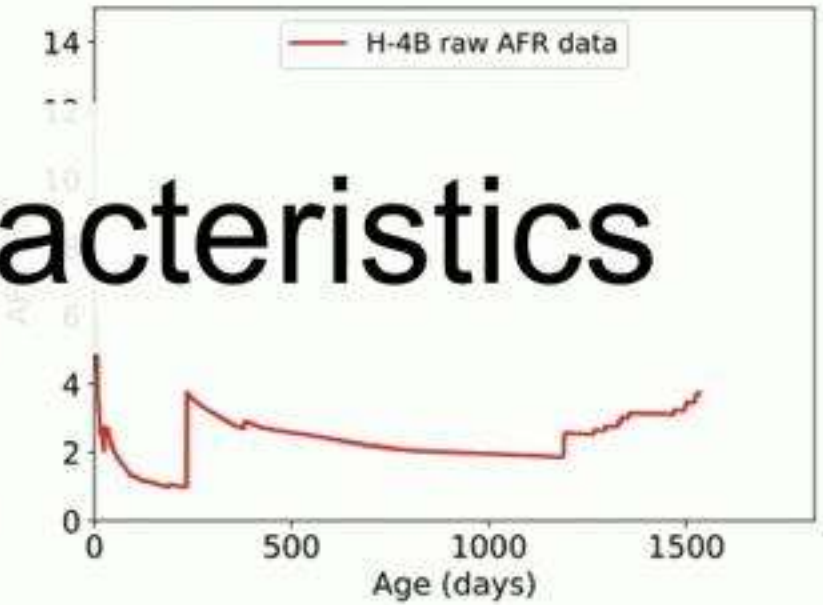
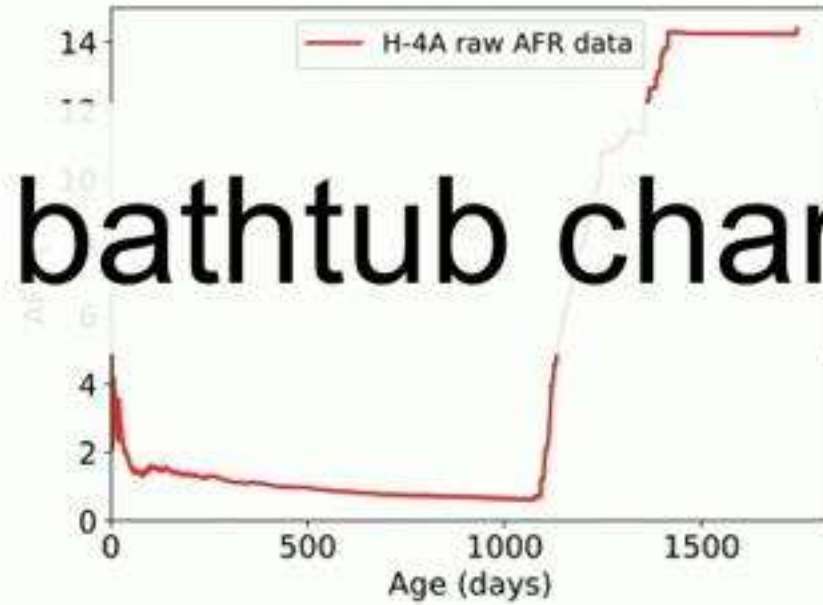
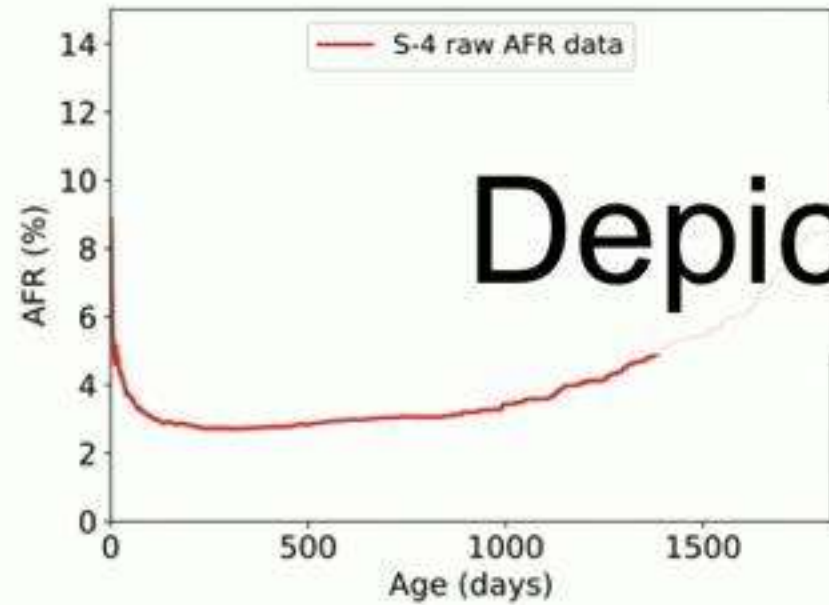




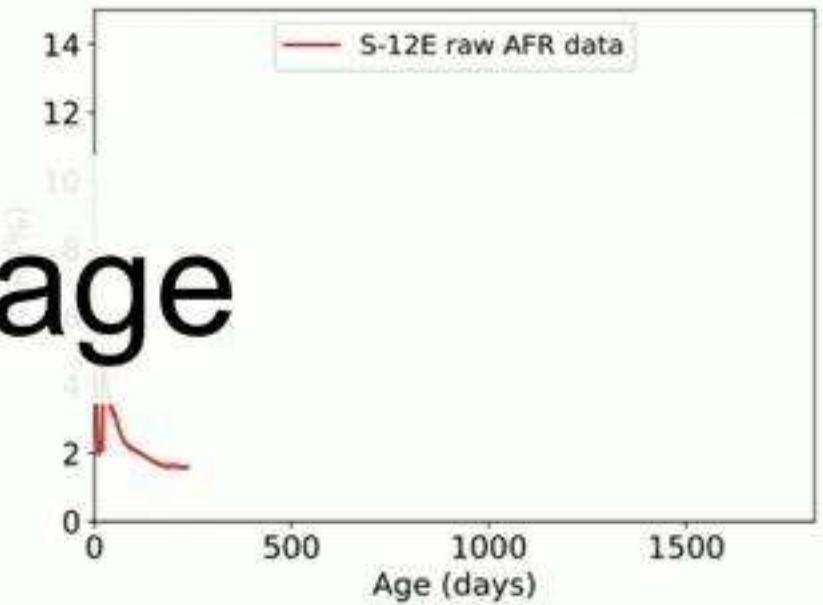
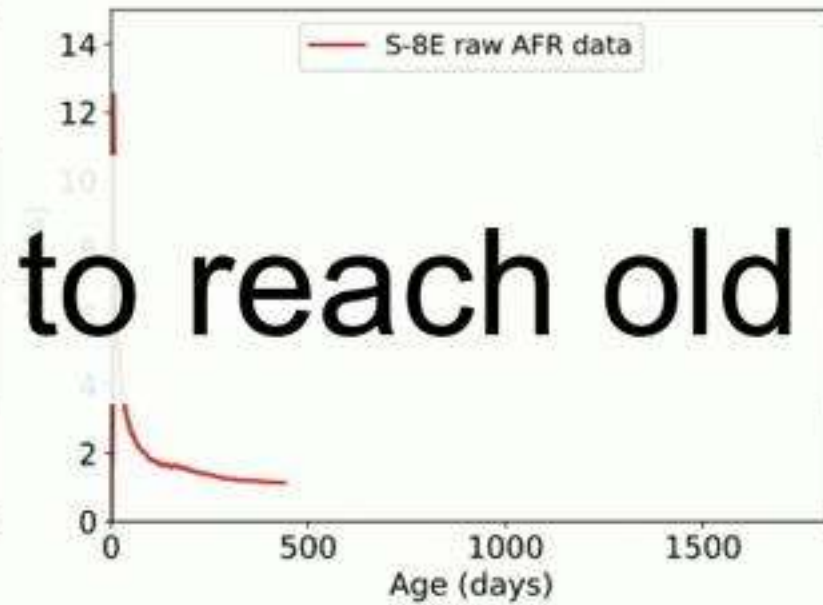
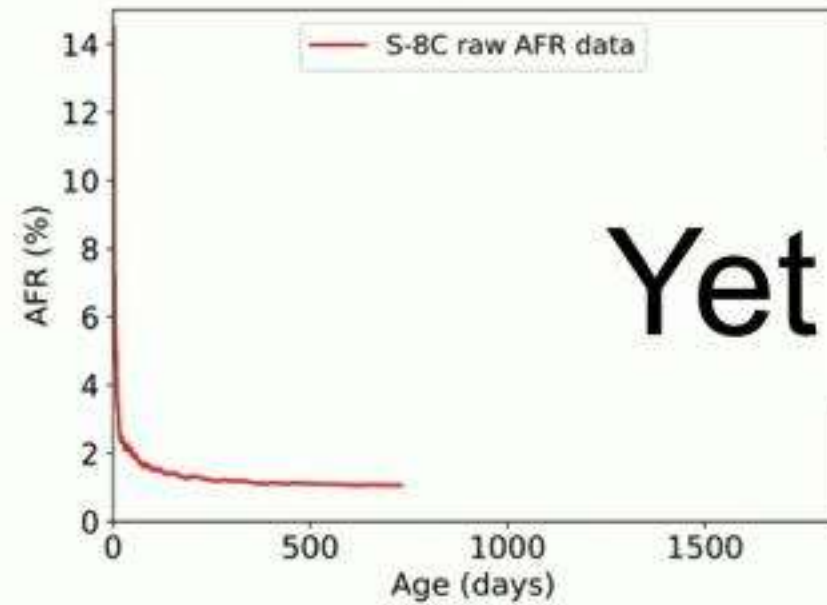
# Failure rate variation over time



# Failure rate variation over time



Depicting bathtub characteristics



Yet to reach old age

We propose a  
**dynamic approach to redundancy configuration**  
in cluster storage systems

Key Idea:

exploit **reliability heterogeneity** for cost savings  
by **tailoring redundancy levels** to observed failure rates  
of different **disk groups over time**



# A disk group over time

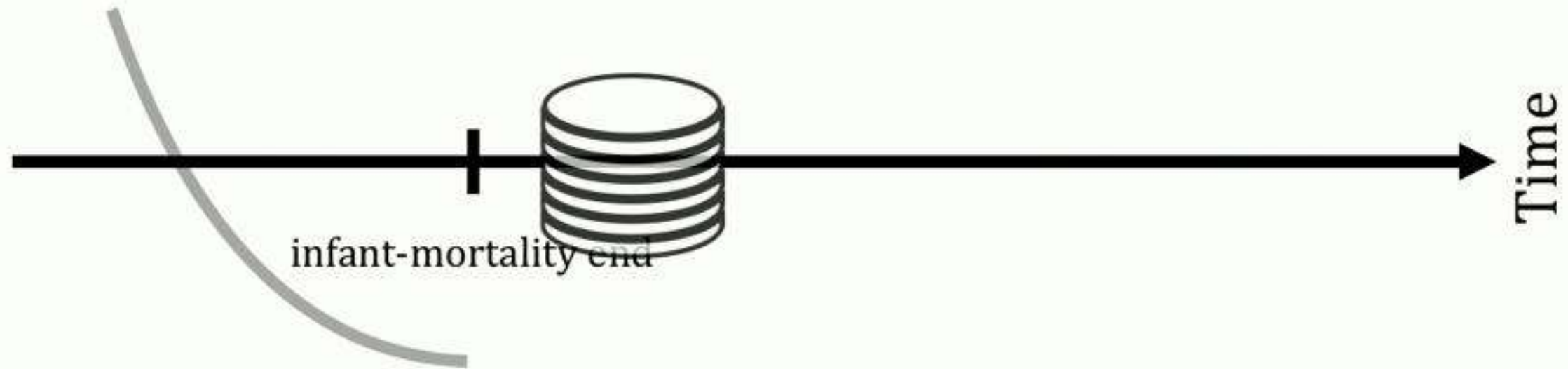
---



# A disk group over time

---

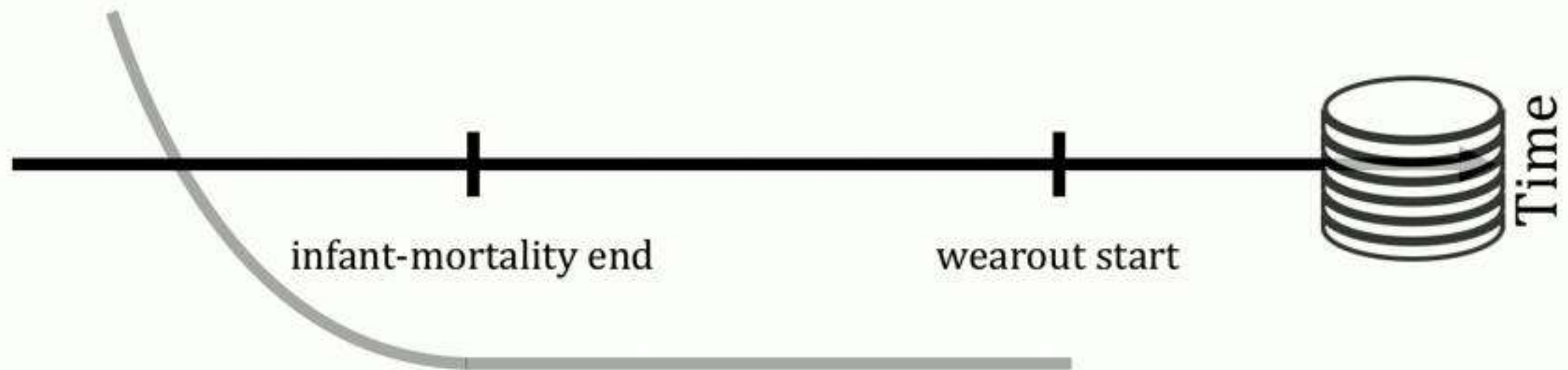
Deployment  
(start monitoring)



# A disk group over time

---

Deployment  
(start monitoring)

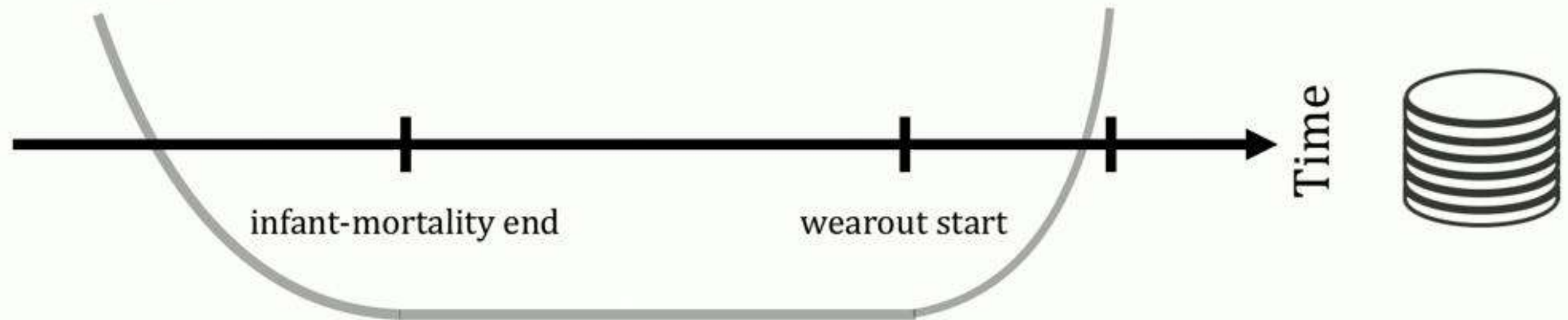




# A disk group over time

---

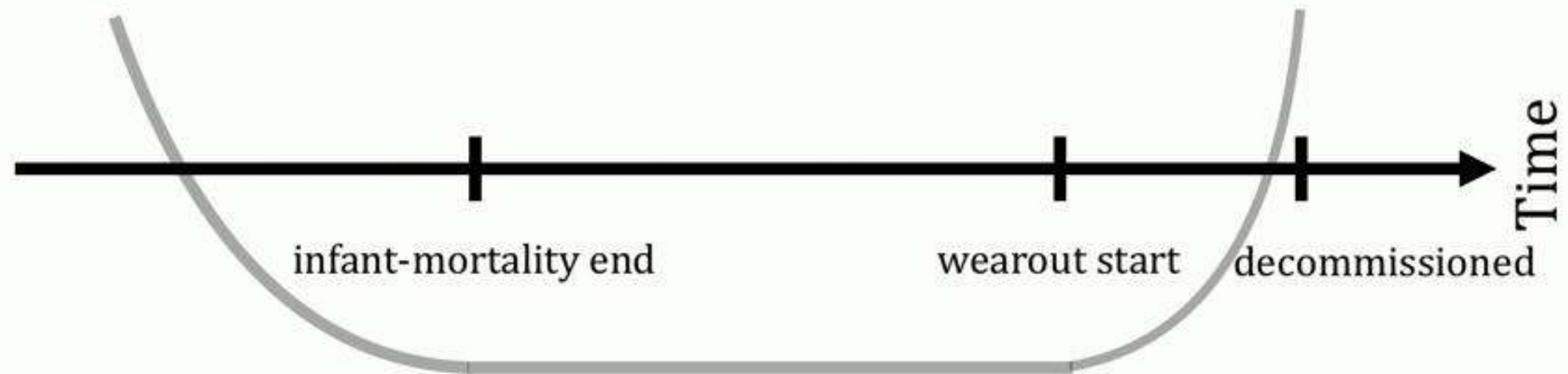
Deployment  
(start monitoring)



# A disk group over time

---

Deployment  
(start monitoring)

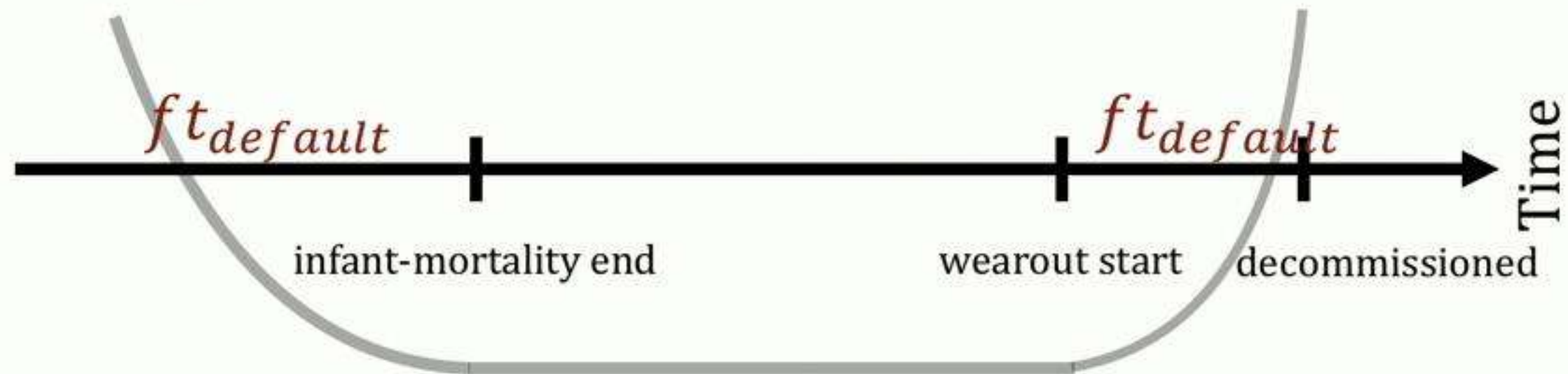


# A disk group over time

---

$ft_{default}$  = default fault tolerance scheme

Deployment  
(start monitoring)





# A disk group over time

---

$f^t_{default}$  = default fault tolerance scheme

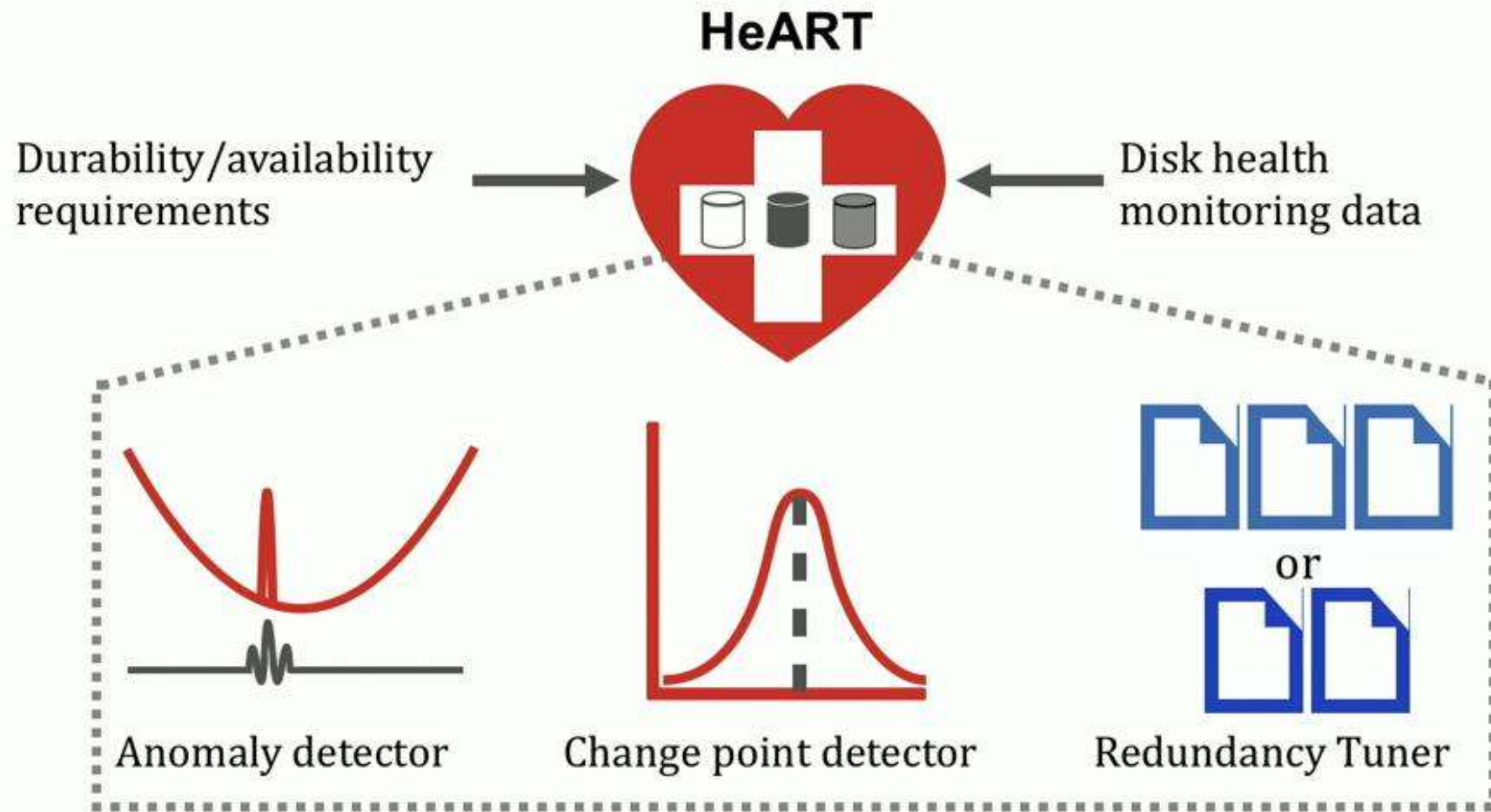
$f^t_{disk-group}$  = disk group specific fault tolerance scheme

Deployment  
(start monitoring)



$f^t_{disk-group}$  lower redundancy than  $f^t_{default}$  while meeting constraints

# Heterogeneity-Aware Redundancy Tuner





# AFR in useful life: stability & anomalies

---

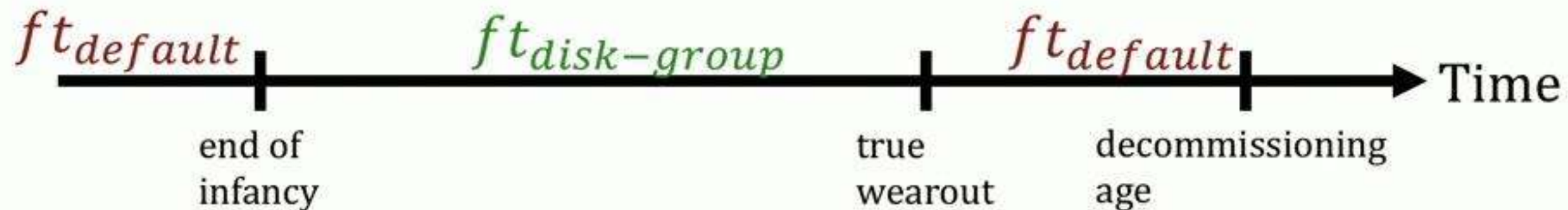
- Useful life AFR is **typically stable** (within reasonable bounds)
- External factors can cause **simultaneous bulk failures**
  - Rack power failure, accidents, human error, etc.
- Such “**anomalies**” appear like (premature) wearout
  - Bulk failures typically don't reflect true HDD failure rate
    - Need to handle with other techniques such as placement
  - Benefits proportional to length of useful life



# AFR in useful life: stability & anomalies

---

- Useful life AFR is **typically stable** (within reasonable bounds)
- External factors can cause **simultaneous bulk failures**
  - Rack power failure, accidents, human error, etc.
- Such **“anomalies”** appear like (premature) wearout
  - Bulk failures typically don't reflect true HDD failure rate
    - Need to handle with other techniques such as placement
  - Benefits proportional to length of useful life

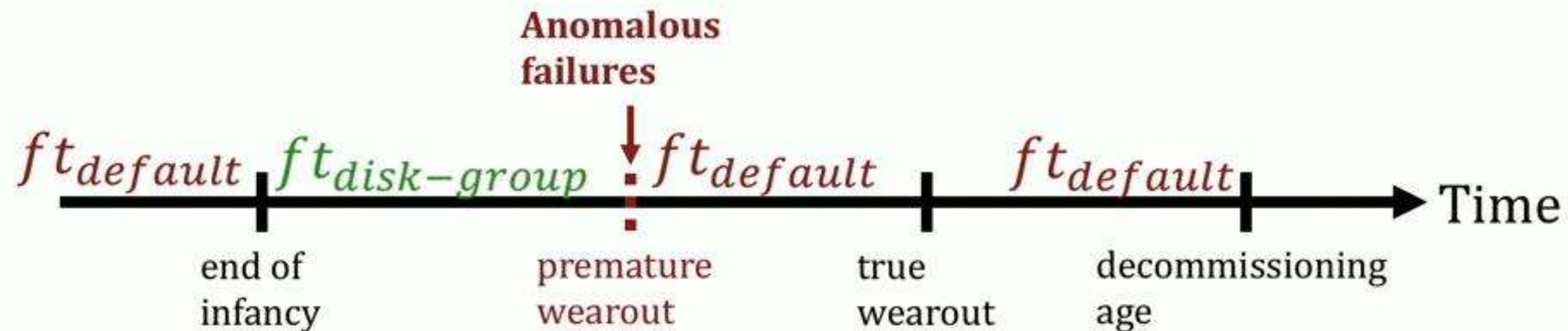




# AFR in useful life: stability & anomalies

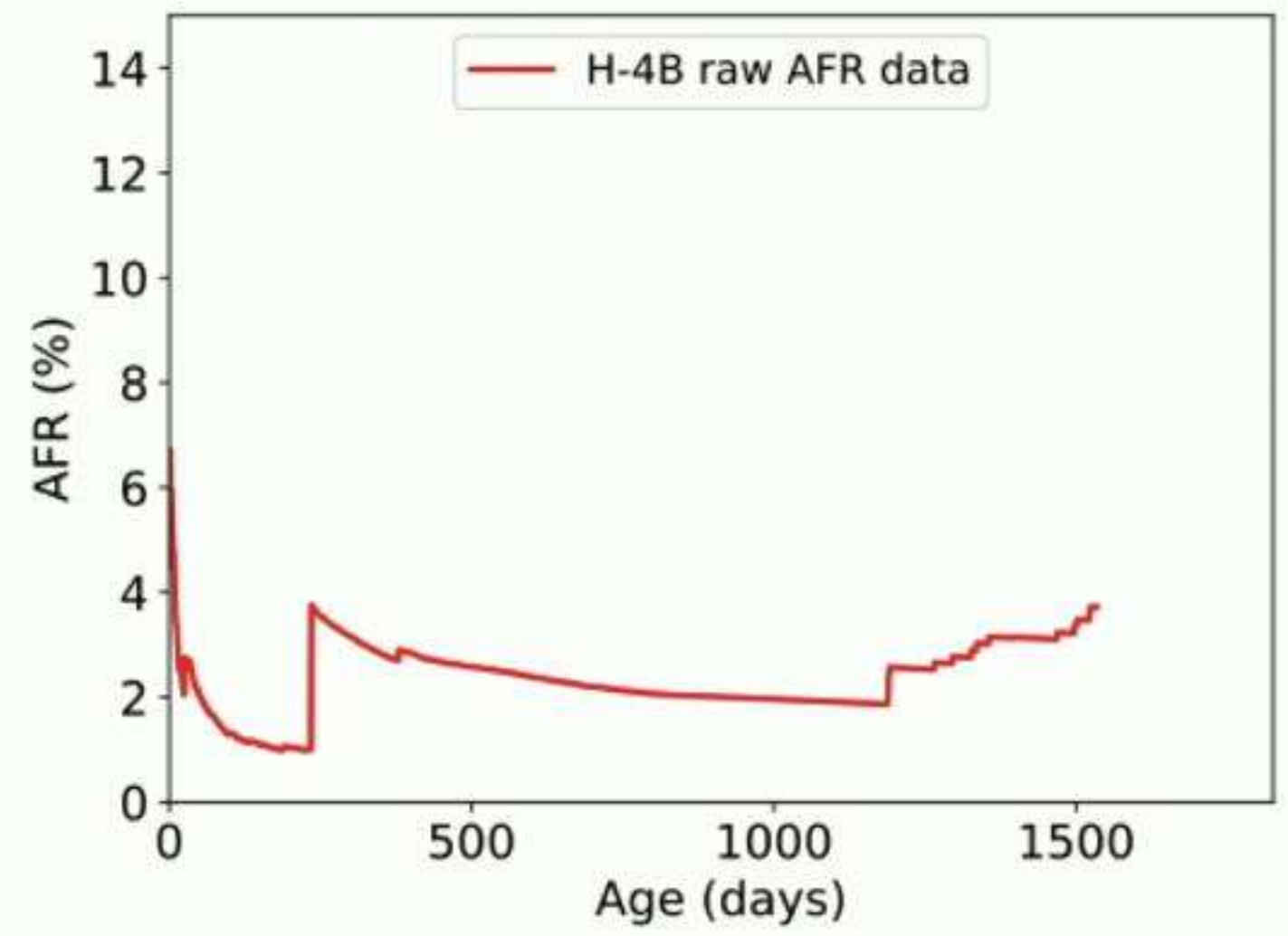
---

- Useful life AFR is **typically stable** (within reasonable bounds)
- External factors can cause **simultaneous bulk failures**
  - Rack power failure, accidents, human error, etc.
- Such **“anomalies”** appear like (premature) wearout
  - Bulk failures typically don't reflect true HDD failure rate
    - Need to handle with other techniques such as placement
  - Benefits proportional to length of useful life



# AFR in useful life: stability & anomalies

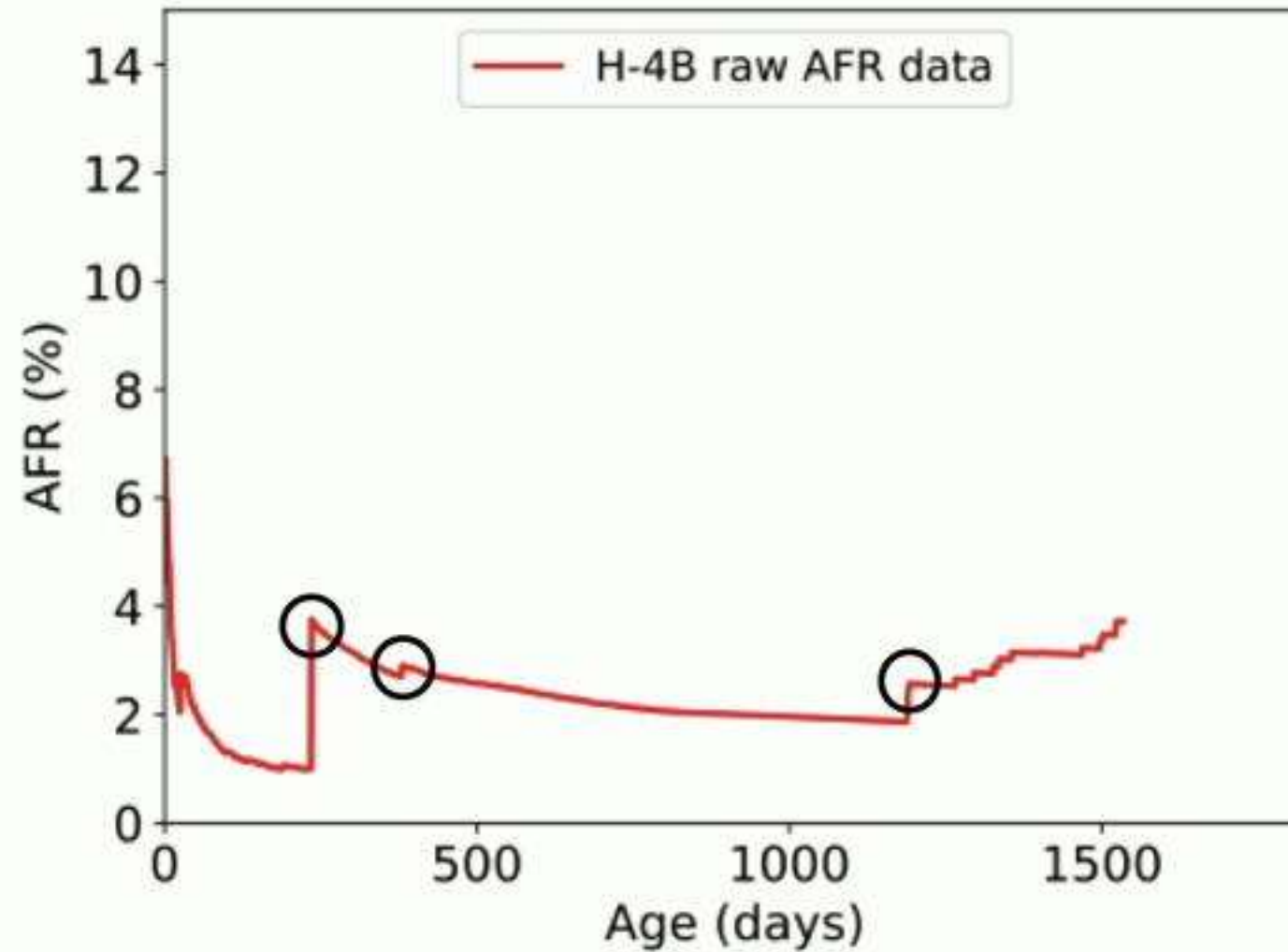
---





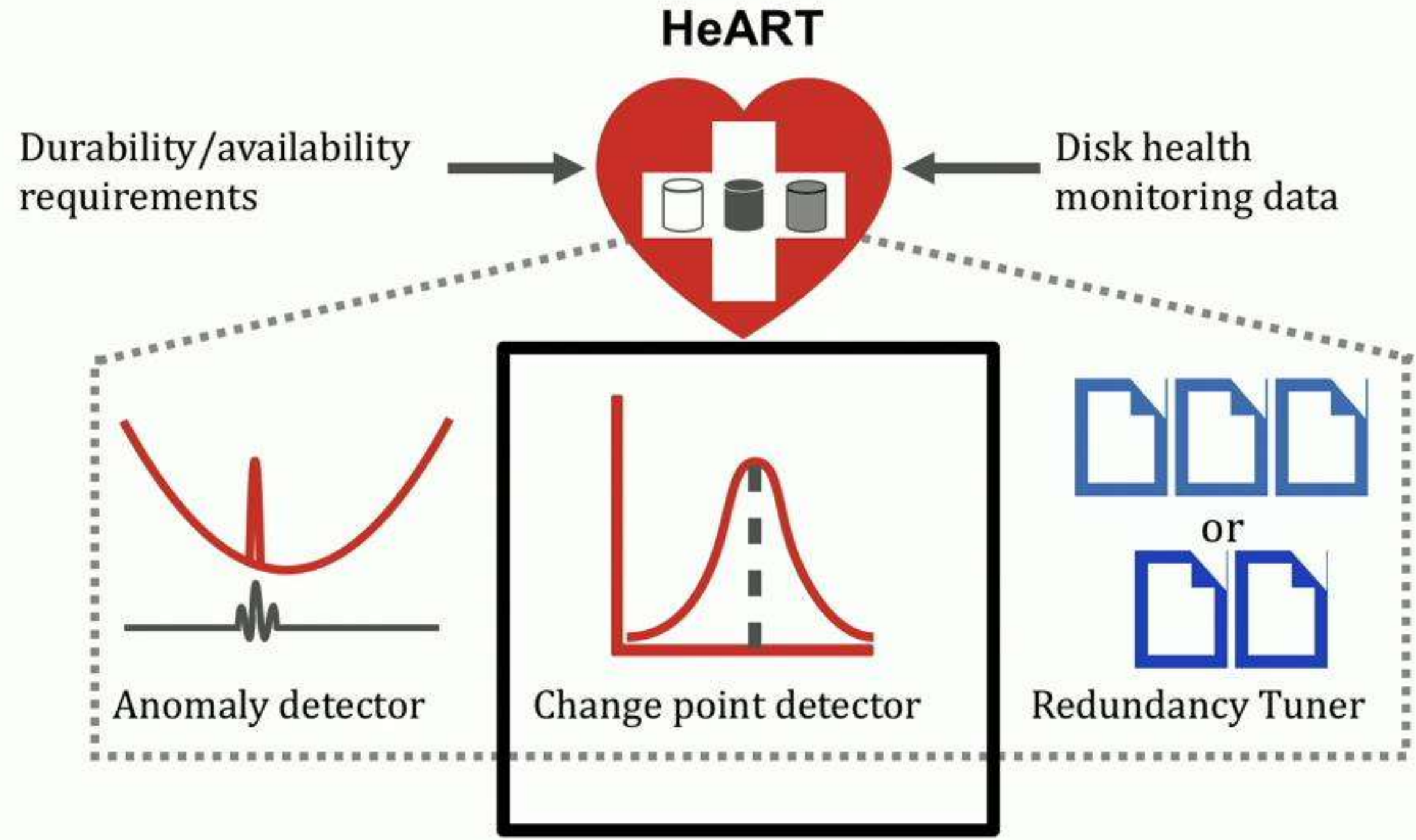
# AFR in useful life: stability & anomalies

---



Spikes due to **simultaneous** bulk failures

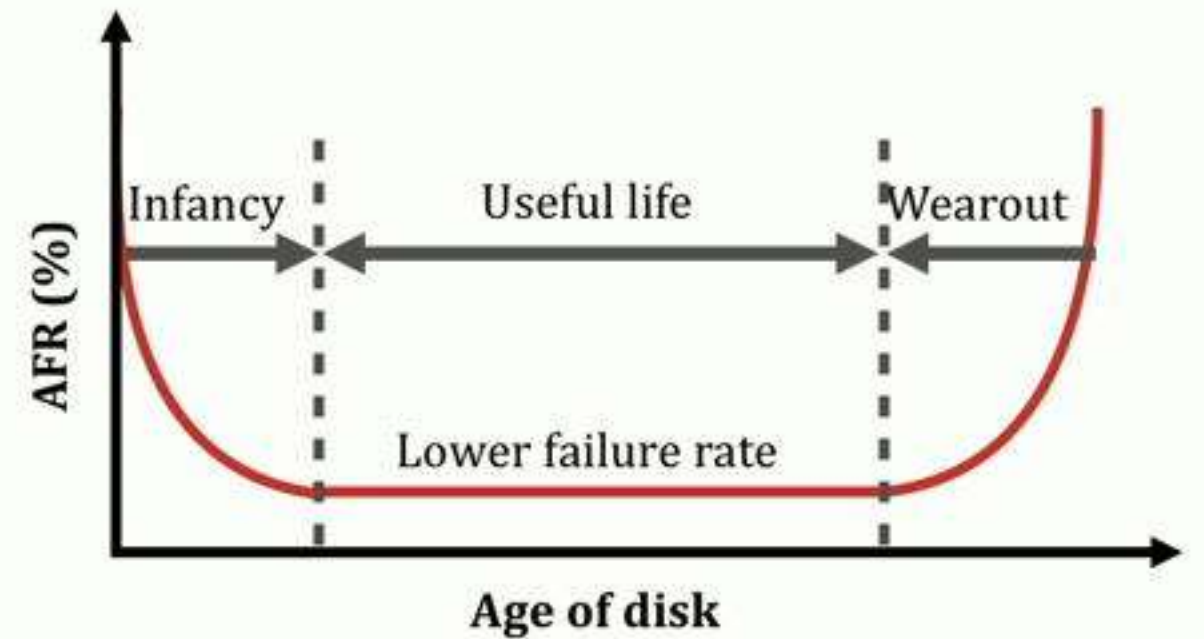
# Heterogeneity-Aware Redundancy Tuner



# Change point detection

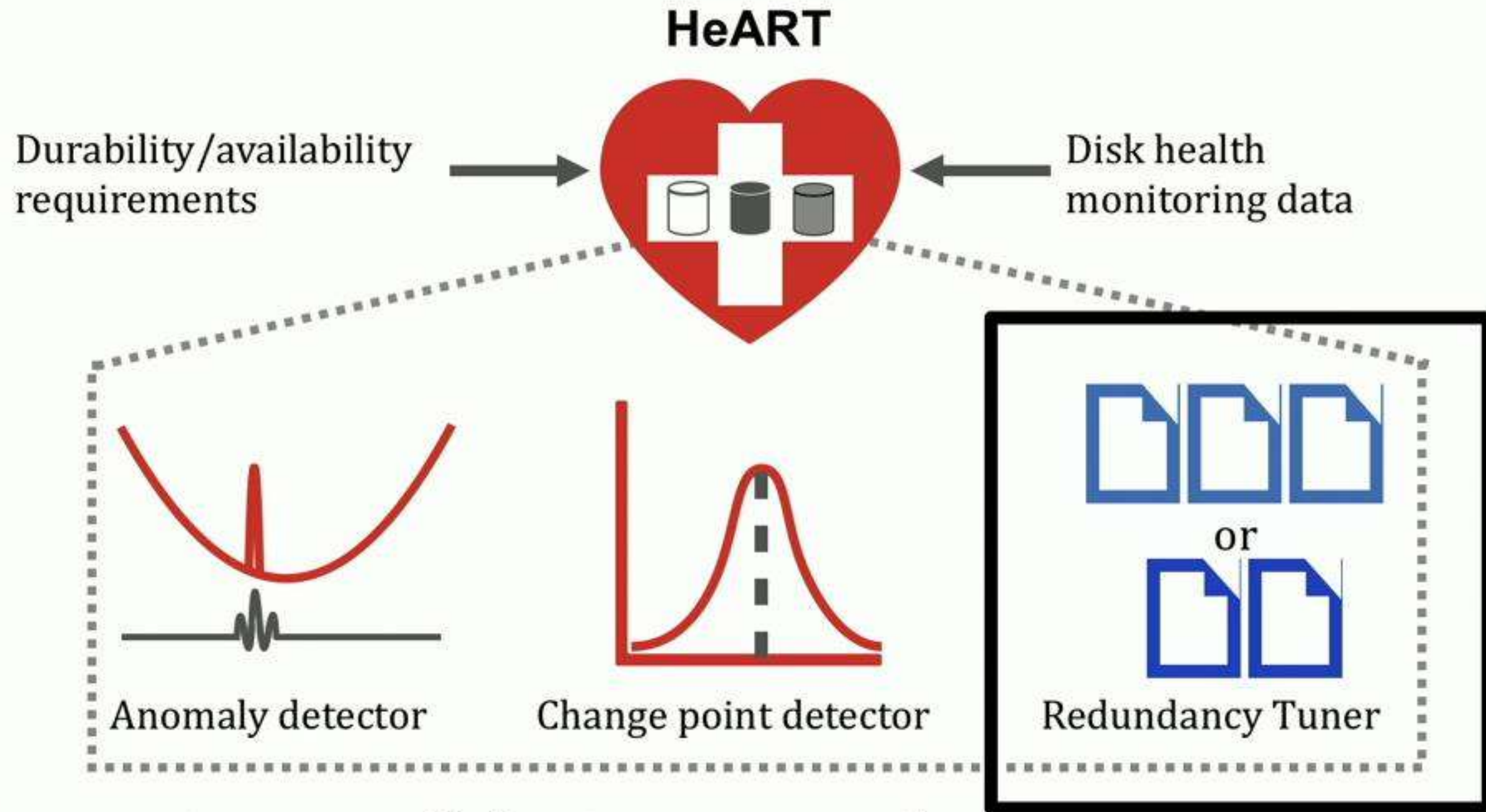
---

- Use online change point detectors to identify change points
- Reliability target can be missed if:
  - Hasty declaration of end of infancy
  - Delayed declaration of onset of wearout
- Tradeoff between benefits and safety
  - Buffer added to estimated AFR as an additional safety measure





# Heterogeneity-Aware Redundancy Tuner



Chooses the most space efficient erasure code for the observed useful-life AFR

# Evaluation on Backblaze dataset

---

- 100K+ HDDs belonging to Backblaze
  - 6 drive makes/models with significant number of disks to test
  - More than 5 years of failure data
- Methodology
  - Leverage off-the-shelf anomaly and change point detectors
  - Reliability target decided by disk group with highest AFR
  - $f t_{disk-group}$  is decided with the following constraints:
    - Tolerate at least **as many failures** as the default
    - Have an **upper bound on stripe width** (“k”)



# Evaluation on Backblaze dataset

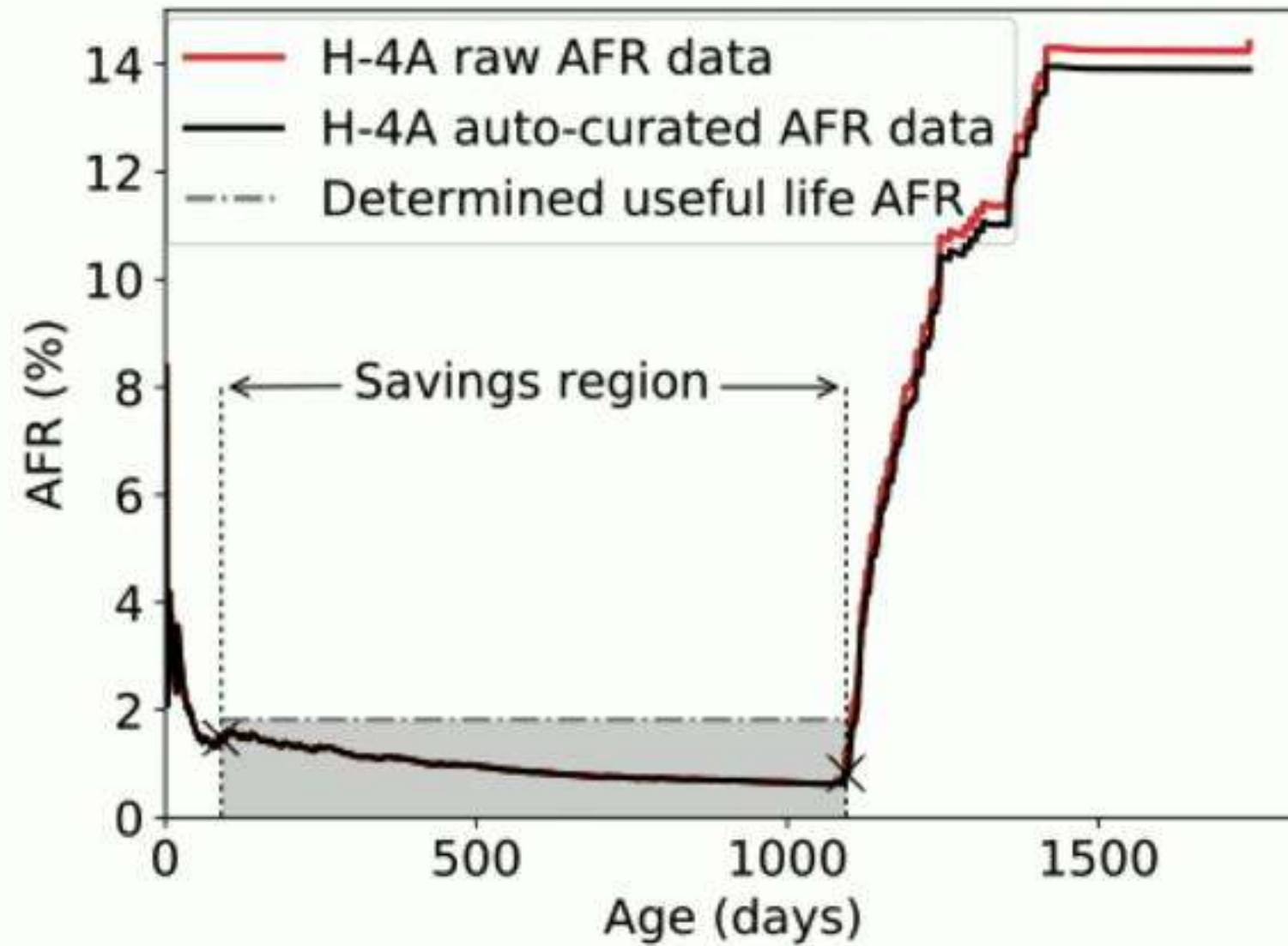
---

- S-4 disks have the highest AFR (4.01%) in Backblaze
  - Reliability target is MTDDL of  $f^{t_{default}}$  on S-4 HDDs
- Upper bound on stripe width =  $2x f^{t_{default}}$
- $f^{t_{default}}$  options evaluated:
  - (n = 9, k = 6) erasure code
  - (n = 14, k = 10) erasure code

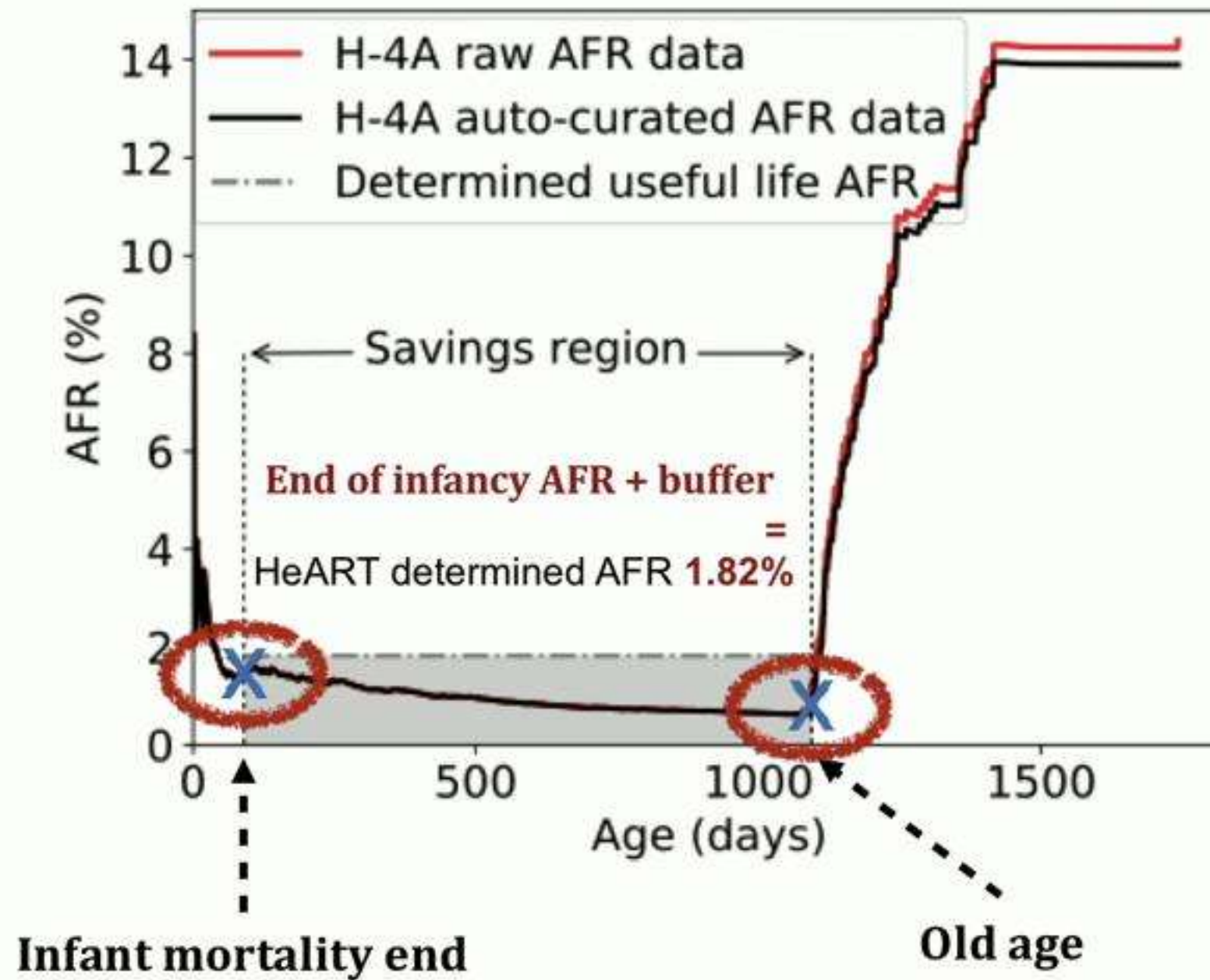


# HeART in action: H-4A HDDs

---

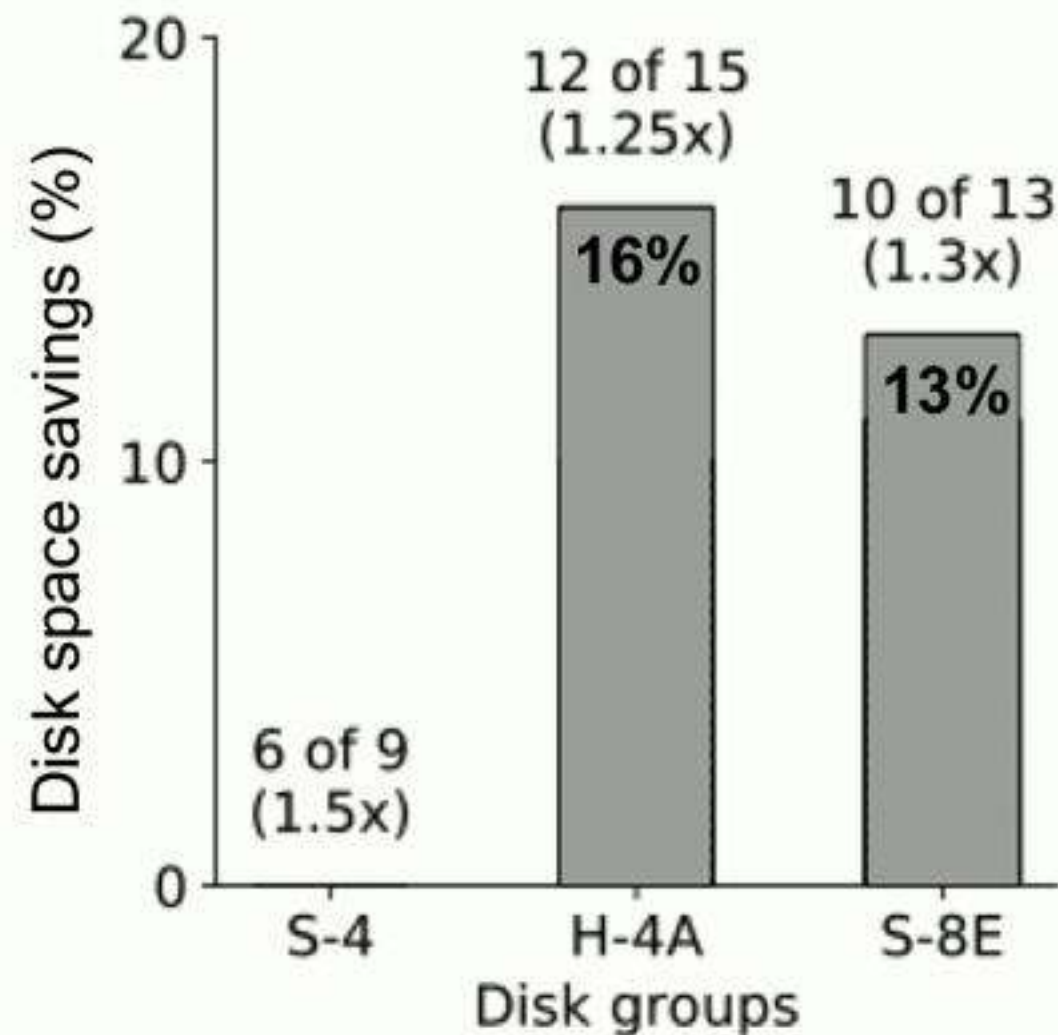


# HeART in action: H-4A HDDs



# Storage savings: (n = 9, k = 6) default scheme

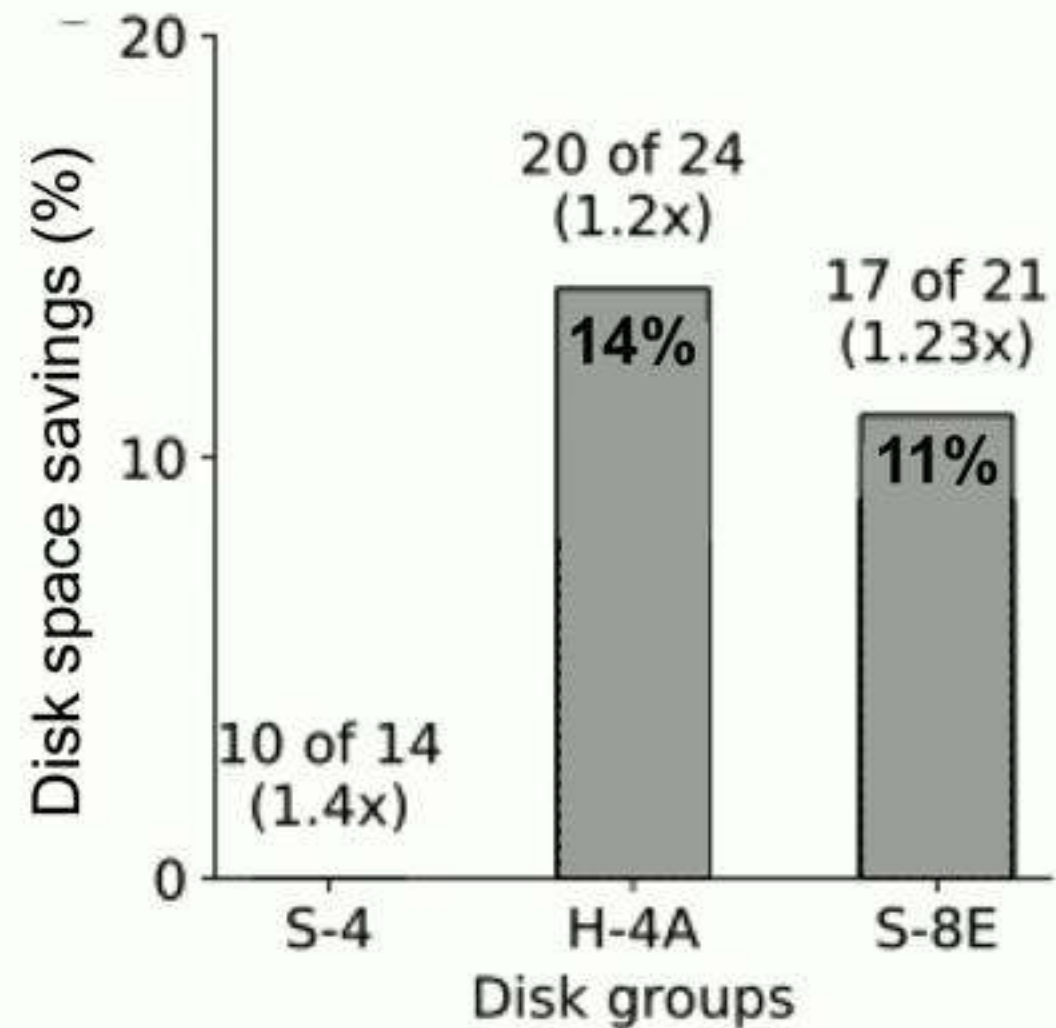
- Small storage overhead of only 1.5x





# Storage savings: (n = 14, k = 10) default scheme

- Even smaller storage overhead of only 1.4x



# Storage savings: (n = 14, k = 10) default scheme

- Even smaller storage overhead of only 1.4x

- 11% – 16% space savings even in space optimized storage systems
- Translates to significant cost savings in large scale systems





# Summary (Part 2)

---

- Proposed a **dynamic approach for redundancy configuration** in cluster storage systems
  - Exploiting **disk reliability heterogeneity** for cost savings
  - By tailoring redundancy levels to observed failure rates
  - Evaluation on production dataset from Backblaze
    - 11% - 16% savings even in space optimized storage systems
- Established the potential of HeART
- Interest from industry
  - **NetApp** and **Google** have shared data



# Future work (Part 2)

---

- Statistical study of **AFR curve estimation and bounds**
  - Statistical analysis for tight bounds on AFR estimation
- Reducing the **overhead of redundancy scheme conversions**
  - Systems: Design of efficient redundancy management
  - Theory: A new class of storage codes enabling efficient conversions
    - “Convertible codes”<sup>1</sup>
  - Lots of open questions to explore

<sup>1</sup>“Convertible Codes: Efficient Conversion of Coded Data in Distributed Storage”, Francisco Maturana and K.V. Rashmi, Available on arXiv, July 2019.

**Thanks!**