



**NANYANG**  
**TECHNOLOGICAL**  
**UNIVERSITY**

Parallel and Distributed Computing Centre

**CORE:**

**Cross-Object **RE**dundancy for  
Efficient Data Repair in Erasure-  
Coded Storage Systems**

Kyumars Sheykh Esmaili, Lluís Pamies-Juarez  
Anwitaman Datta

---

**The Problem:**  
**Replication is not**  
**Cheap**

---

# Replication: Why and How?

- ❑ Huge volumes of data maintained by storage systems
- ❑ To scale-out, it has to be distributed
- ❑ Failures are norm rather than exception
- ❑ Redundancy is used to guarantee fault tolerance
- ❑ Replication is the simplest and the most commonly used form of redundancy
  - ❖ Identical copies (de-facto standard: 3 copies)

# Replication Can Be Expensive

- ❑ According to the “new” conventional wisdom, storage is cheap and replication is always affordable
- ❑ But at the scale of petabytes and exabytes, it’s not the case anymore
  - ❖ e.g., Facebook’s has 100PB of data and it doubles every year
  - ❖ Similar for Cloud Servers
- ❑ A 50% reduction in storage would save tens of millions of dollars

# Alternatives Needed

## □ Requirements

- ❖ Lower storage
- ❖ Same or higher reliability
- ❖ No drastic performance degradation

---

---

# An Incomplete Solution: Erasure Codes

---

# Erasure Codes: History

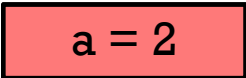
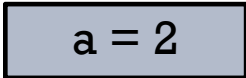
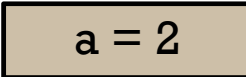
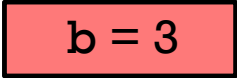
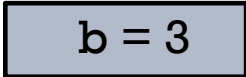
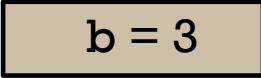
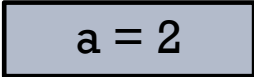
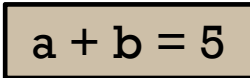
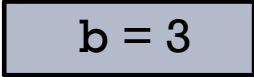
- ❑ Invented by the Coding Theory community
  - ❖ A sender; A receiver; An erasure channel in between
  - ❖ Along with the data some redundancy is introduced so that receiver can recover the message even if some parts of the data are erased
- ❑ Reed-Solomon Codes: RS (n,k)
  - ❖ One of the most well-known and widely-used instances
  - ❖ Encodes '**k**' data parts into '**n**' encoded parts ( $n > k$ )
  - ❖ Data can be recovered using **any k subset** of n
- ❑ Used in wide range of systems (e.g., satellite communications, CDs, QR codes, ...)

# Erasure Codes: Basics

- ❑ Based on linear algebra and finite fields (Galois fields)
- ❑ Mathematically complex, but well understood and with many efficient algorithms and implementations (including dedicated hardware circuits)
- ❑ Not included in this talk, but I would be happy to provide detailed information afterwards



# Erasure Codes: Simple Example

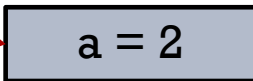
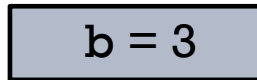
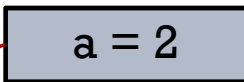
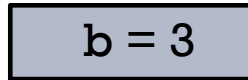
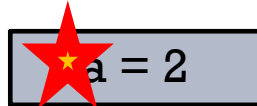
	Replication R. Factor =2	Erasure Coding: RS(3,2)	
			<b>25% saving in storage!</b>
			
failure rate $f = 0.1$			In realistic scenarios, it reduces storage by up to <b>50%</b> and guarantees <b>higher availability</b>
			
Availability:	98%	97%	
Storage Overhead:	100%	50%	

# Erasure Coding for Storage?

- ❑ First EC-based systems in use already
  - ❖ Azure (Microsoft), GFS (Google), HDFS (Facebook)
- ❑ Advantages
  - ❖ Lower storage overhead
  - ❖ Higher reliability
- ❑ Disadvantage
  - ❖ No notion of **repair** and **network**
    - It was designed for another set of goals
    - In networked storage systems, failed nodes can be and must be replaced to guarantee long-term availability

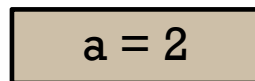
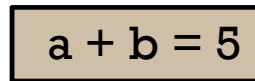
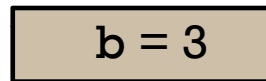
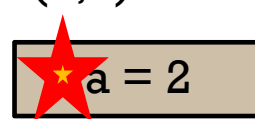
# Repair Inefficiency of EC Storages – Example

Replication  
R. Factor = 2



Cost: 1 blk  
Fan-in: 1

Erasure Coding:  
RS(3,2)



2 blks  
2

- ❑ Repair Metrics:
  - ❖ Cost (no. blks)
  - ❖ Fan-in (no. nodes)
  - ❖ Time
  - ❖ ...
- ❑ The challenge: make the repair fast

---

---

# **Towards a Complete Solution: CORE**

---

# State-of-the-Art

## □ Scientific approaches

- ❖ New Codes with inherently better repairability
  - e.g., Regenerating Codes [Dimakis et. al, 2010]
  - e.g., Self-repairing Codes [Oggier and Datta, 2011]
- ❖ Mathematically complex
- ❖ No real-system implementation

## □ Engineering approaches

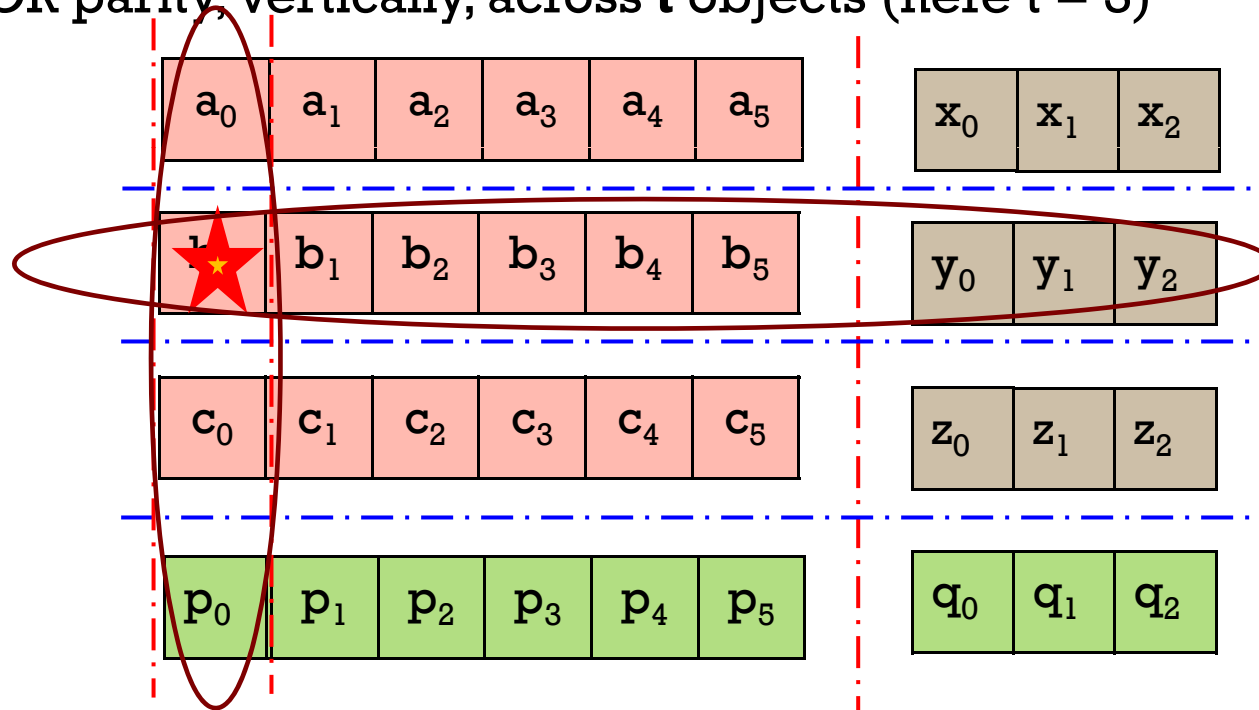
- ❖ Codes on Codes
- ❖ Easier to understand
- ❖ Possible to reuse the existing implementations and optimizations
- ❖ Examples:
  - Local Reconstruction Codes (**LRC**) in Microsoft Azure
  - **CORE**

# CORE - The Basic Idea

- Introduce **two different kinds of redundancy**
  - ❖ Erasure Code for individual objects
    - for availability
  - ❖ RAID-4 like parity across encoded pieces of different objects
    - for repairability

# CORE - Example

- ❑ Three objects
- ❑ Reed Solomon ( $n=9, k=6$ ) horizontally and individually
- ❑ XOR parity, vertically, across  $t$  objects (here  $t = 3$ )



Bandwidth Usage: 3 blks (instead of 6 blks)

# CORE – Essential Properties

---

- ❑ Reduces the repair cost (no. blocks)
- ❑ Reduces repair fan-in
- ❑ Requires a much cheaper computation in most of the cases (XOR vs. RS)
  - ❖ Can be further optimized by pipelining
- ❑ Retains comparable storage overhead



# CORE: Performance Analysis

## □ Analytically

- ❖ Versus Reed-Solomon coding and Microsoft's LRC codes
- ❖ CORE and LRC much more efficient in repairs
- ❖ For same storage overhead, CORE provides higher availability than LRC
- ❖ Details in the paper

## □ Experimentally

- ❖ Versus Reed-Solomon coding

# Implementation

- ❑ HDFS-RAID
  - ❖ Developed at Facebook
  - ❖ Wraps around Apache HDFS
- ❑ Optimized HDFS-RAID
  - ❖ **Opt1**: Repair using exactly “k” blocks
    - Beneficial in network-scarce setups
  - ❖ **Opt2**: Simultaneous repairs (avoiding “doomed” recoveries)
- ❑ Integrated into HDFS-RAID
  - ❖ Named StorageCORE
  - ❖ Available at:  
<http://sands.sce.ntu.edu.sg/StorageCORE/>

# Experiments - Setup

- Two different clusters, each of 20 nodes
  - ❖ A network-critical cluster (**NTU**)
  - ❖ A computation-critical cluster (**EC2**)
- Coding parameters
  - ❖  $(n=14, k=12, t=5)$ 
    - Similar to Microsoft's
  - ❖  $(n=9, k=6, t=3)$ 
    - Similar to Google's

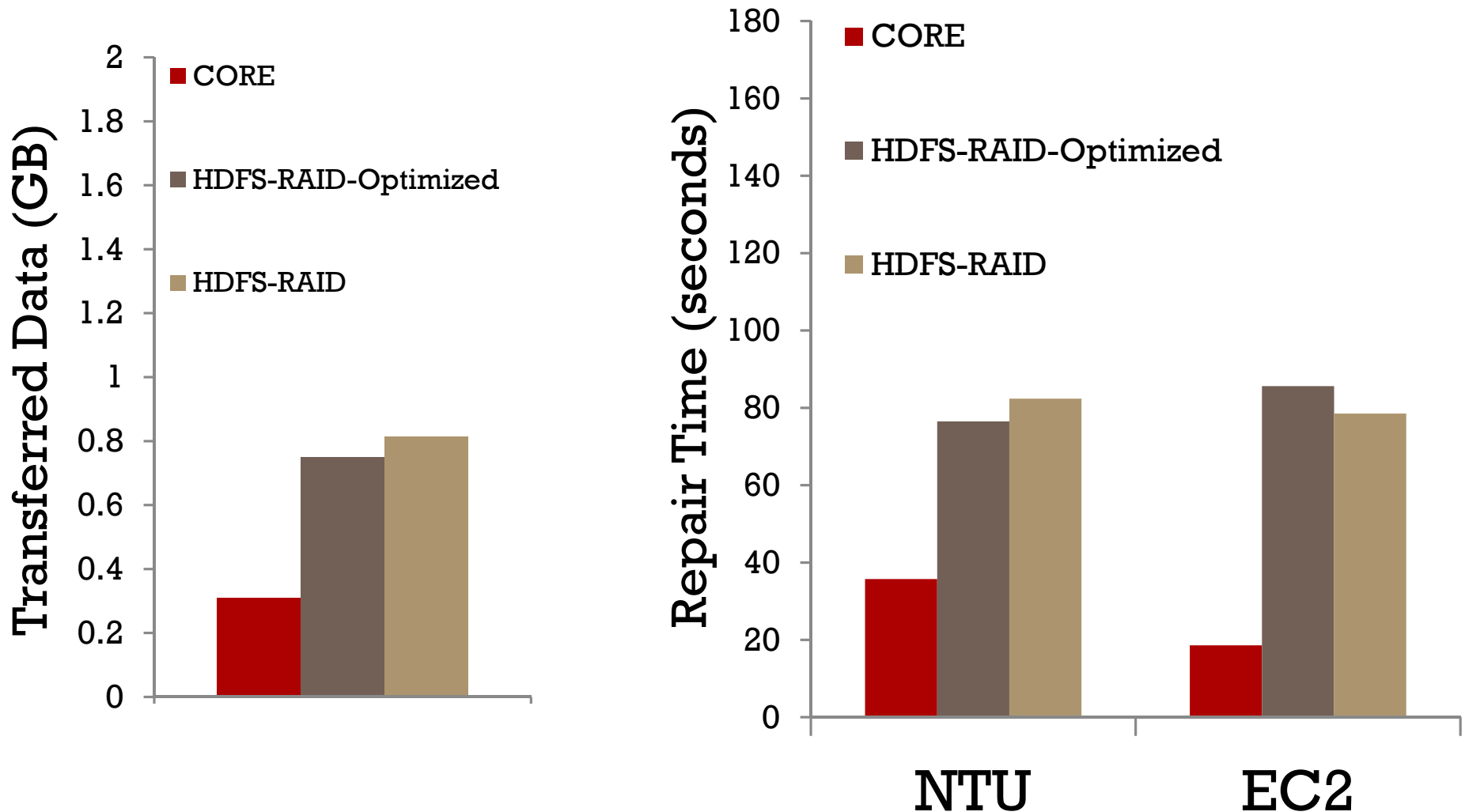
# Experiments - Setup

- Alternative approaches:
  - ❖ HDFS-RAID
  - ❖ Optimized HDFS-RAID
  - ❖ CORE
- Number of failures
  - ❖ One failure (**X**)
  - ❖ Two failures, both from the same object (**XX**)
    - Best case scenario for HDFS-RAID

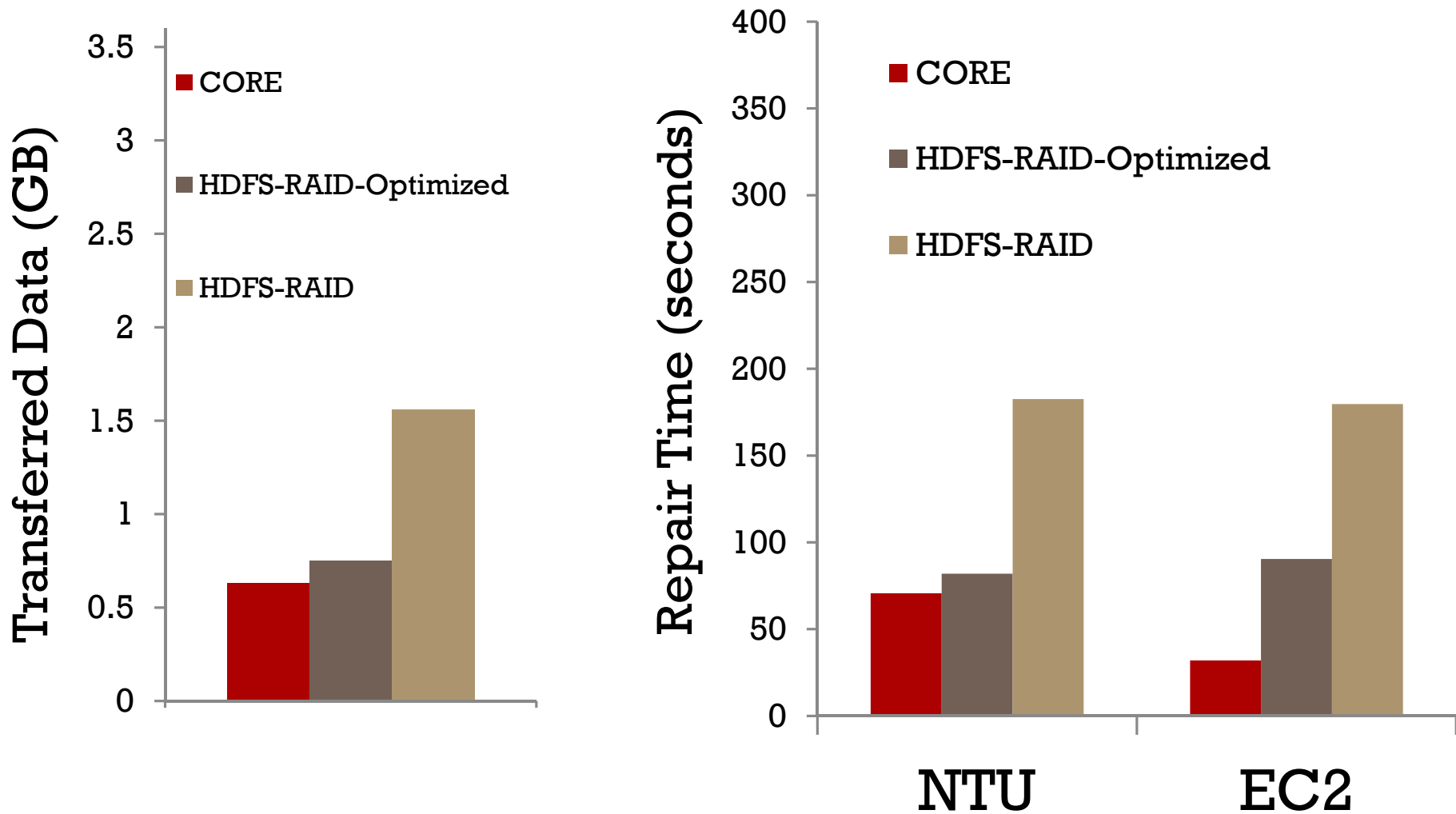
# Experiments - Measurements

- Repair cost (transferred data)
  - ❖ Statically computable and cluster-independent
  - ❖ Used for sanity check and comparison purposes
- Repair time
  - ❖ Varies with cluster type

# Repairing One Failure (X)



# Repairing Two Failures (XX)



---

---

# Algorithmic Aspects of CORE

---



# CORE – Algorithmic Aspects

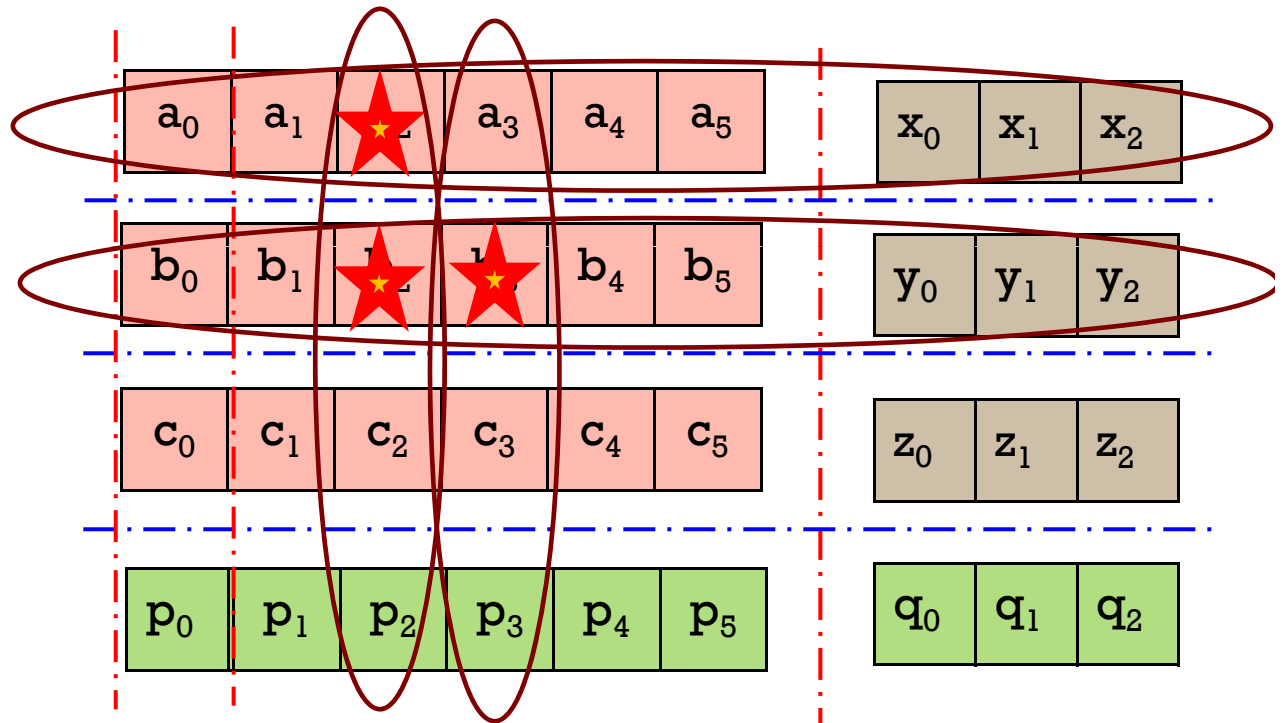
- ❑ Failure clustering
- ❑ Recoverability checking
- ❑ Repair Scheduling

# Repair Scheduling – the “Step” Pattern

Row-first

Column-first

RGS



# Recursively Generated Schedules

- Two simple functions

$$\min H(\text{Col}_j) = \begin{cases} 0 & \text{if } |\#X| \leq 1 \\ |\#X| - 1 & \text{otherwise} \end{cases}$$

$$\min V(\text{Row}_i) = \begin{cases} 0 & \text{if } |\#X| \leq (n - k) \\ |\#X| - (n - k) & \text{otherwise} \end{cases}$$

- Accumulative values

$$v = \sum_{i=1}^t \min V(\text{Row}_i) \quad ; \quad h = \sum_{j=1}^k \min H(\text{Col}_j)$$

# The RGS Algorithm

- The schedule is generated along the call chain of the recursive function  $c(h, v)$

$$c(h, v) = \begin{cases} c(h, dec(v)) + m & \text{if } v > 0 \\ c(dec(h), v) + k & \text{if } v = 0 \\ & \text{or } dec(v) \text{ is not applicable} \end{cases}$$

- At the base case  $c(0, 0)$  use the cost function  $c'$

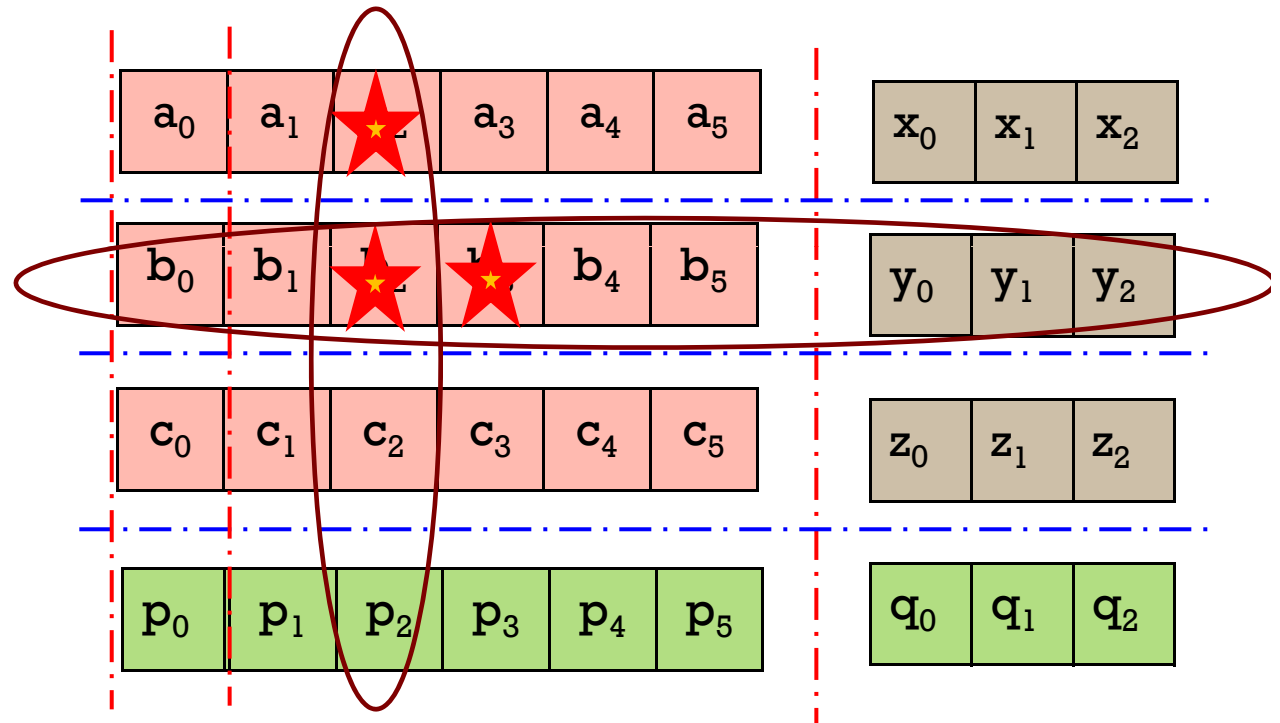
$$c'(r) = \begin{cases} k & \text{if repaired horizontally} \\ r \times m & \text{if repaired vertically} \end{cases}$$

# Repair Scheduling – the “Step” Pattern

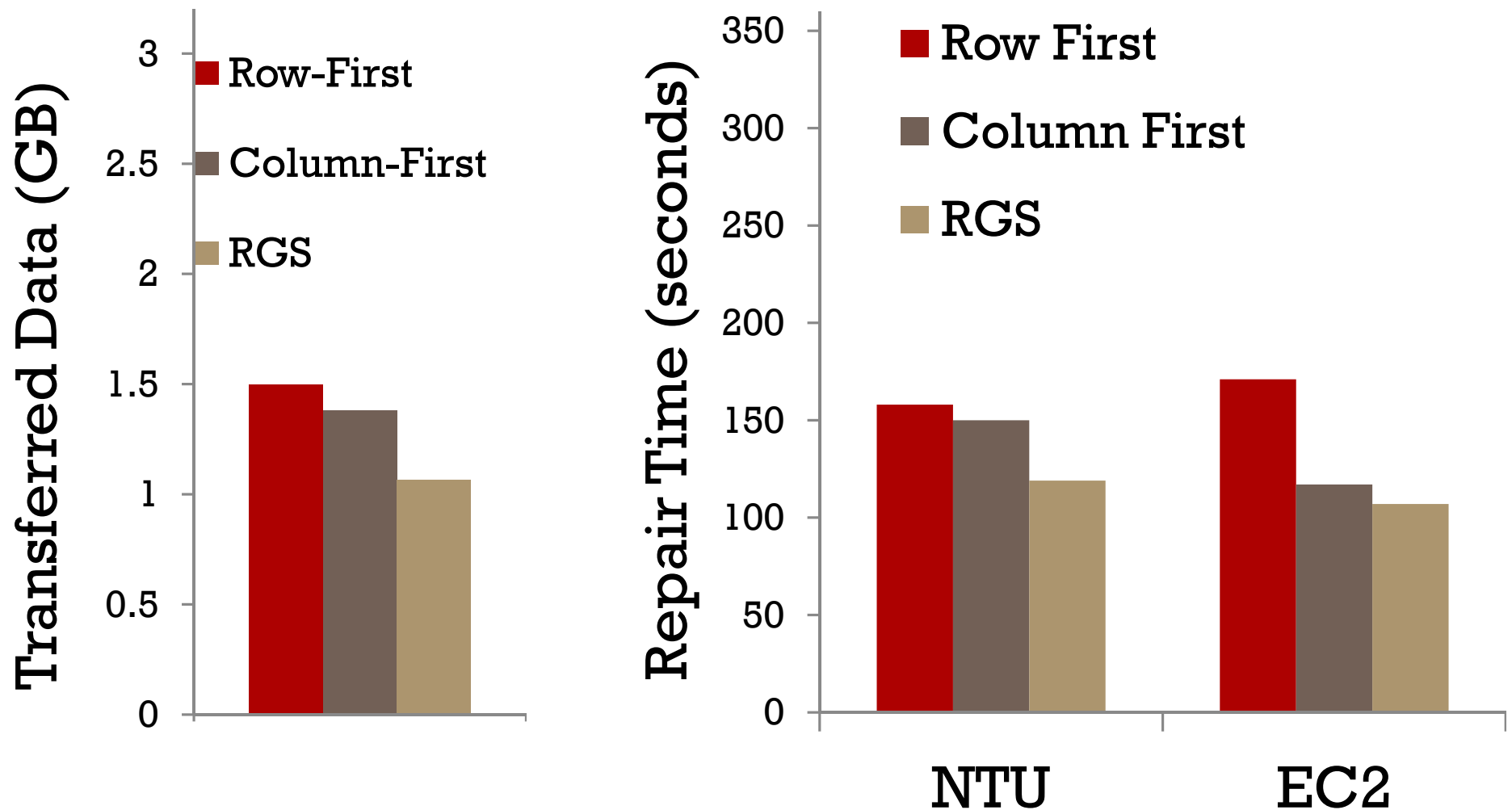
Row-first

Column-first

RGS



# The “Step” Failure Pattern



# Summary

---

- ❑ At petabytes-scale, replication is prohibitive
- ❑ Erasure Coding is an alternative, but it has some shortcomings, most notably, with repairs
- ❑ We proposed CORE, a simple and effective solution
- ❑ The benefits of CORE has been shown through experimental and analytical studies
- ❑ CORE also poses a range of novel challenges, most notably, efficient scheduling of multiple repairs
  - ❖ Our proposed algorithm, RGS, generates efficient schedules

<http://sands.sce.ntu.edu.sg/CodingForNetworkedStorage>