Having Your Cake and Eating It Too: Jointly Optimal Erasure Codes for I/O, Storage and Network-bandwidth

K. V. Rashmi, Preetum Nakkiran, Jingyan Wang, Nihar B. Shah, Kannan Ramchandran

EECS, UC Berkeley



Redundancy in distributed storage

- For durability and availability of data
- Simplest approach: replication
  - store multiple copies on different machines
- Alternative approach: erasure coding



Erasure coding for distributed storage

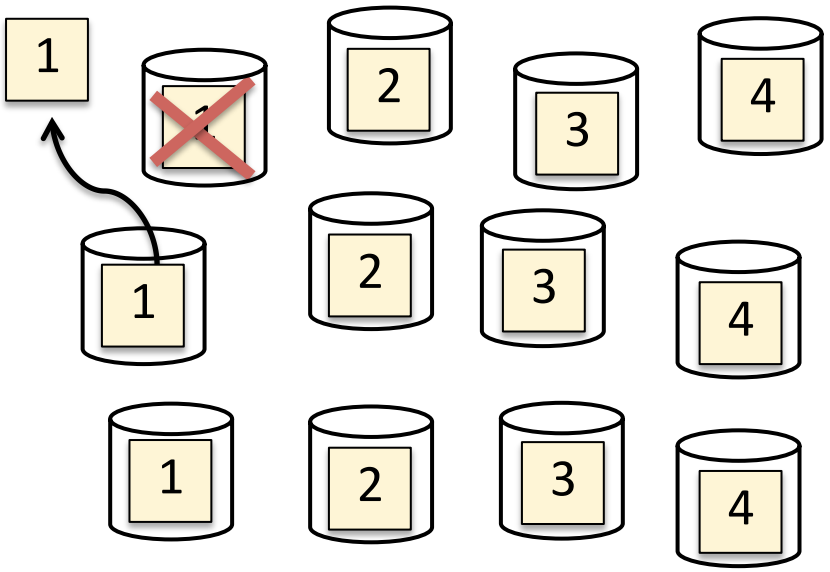
- Storage space efficient
  - traditional codes, e.g., Reed-Solomon, optimal
  - maximum fault tolerance for storage overhead
- Another important metric: "maintenance" cost
  - replacing missing/unavailable data
  - quite frequent in distributed storage
  - *network and I/O costs*
- Traditional codes highly inefficient

Birds-eye view

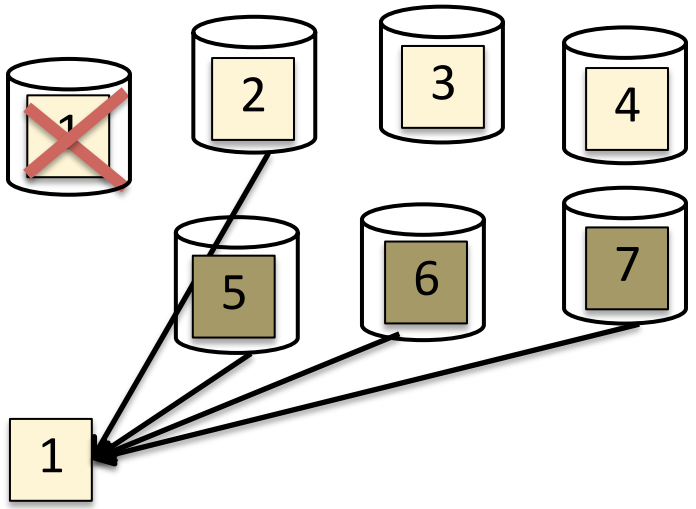
- Traditional codes highly inefficient in replacing missing/unavailable data
- Considerable amount of on-going research in theory and systems communities
- A powerful class of erasure coding framework optimizes only for storage and network-bandwidth costs

We show how to transform these codes to optimize I/O cost as well

“Reconstruction” costs



I/O & network cost = 1x



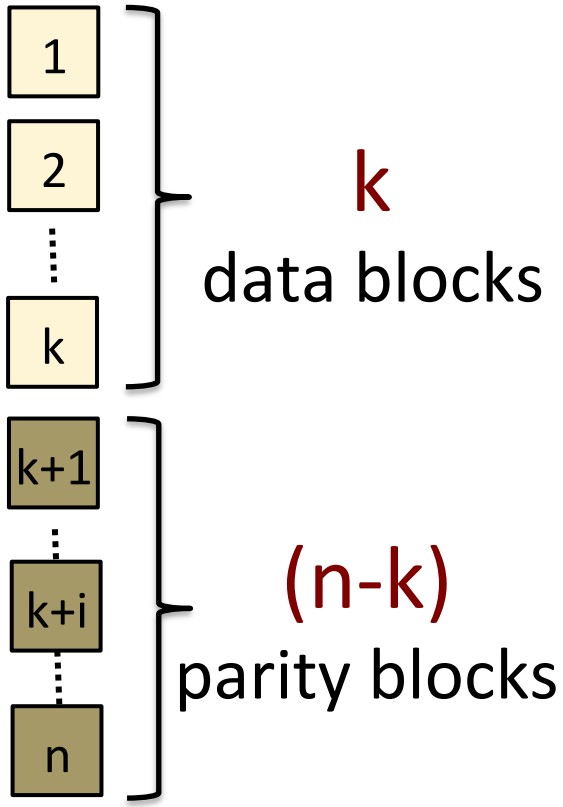
(7, 4) Reed-Solomon code

- any 4 out of 7 sufficient

I/O & network cost = 4x

Reed-Solomon Codes

- (n, k)
  - k data blocks, (n-k) parity blocks
  - any k out of n sufficient
  - optimal storage and fault tolerance
- Reconstruction
  - *any k out of the remaining (n-1) blocks*



I/O & network cost = k x (amount reconstructed)

= 10x - 20x for typical parameters!

Recent research in theory and systems

- ***Theory:*** Minimum Storage Regenerating codes (Dimakis et al, Shah et al, Rashmi et al, Tamo et al, Wang et al, Papailiopoulos et al, Cadambe et al, etc.); **Local codes** (Gopalan et al, Papailiopoulos et al, Tamo et al, Kamath et al, etc.); **Piggyback** (Rashmi et al); **Rotated-RS** (Khan et al), **etc.**

- ***Systems:*** **NCFS & NC-Cloud** (Hu et al), **Xorbas** (Sathiamoorthy et al), **Hitchhiker** (Rashmi et al), **CORE** (Li et al), Andre et al, **etc.**

Recent research in theory and systems

- ***Theory:*** **Minimum Storage Regenerating codes**

  (Dimakis et al, Shah et al, Rashmi et al, Tamo et al, Wang et al, Papailiopoulos et al, Cadambe et al, etc.); **Local codes** (Gopalan et al, Papailiopoulos et al, Tamo et al, Kamath et al, etc.); **Piggyback** (Rashmi et al); **Rotated-RS** (Khan et al), **etc.**

- ***Systems:*** **NCFS & NC-Cloud** (Hu et al), **Xorbas** (Sathiamoorthy et al), **Hitchhiker** (Rashmi et al), **CORE** (Li et al), Andre et al, **etc.**

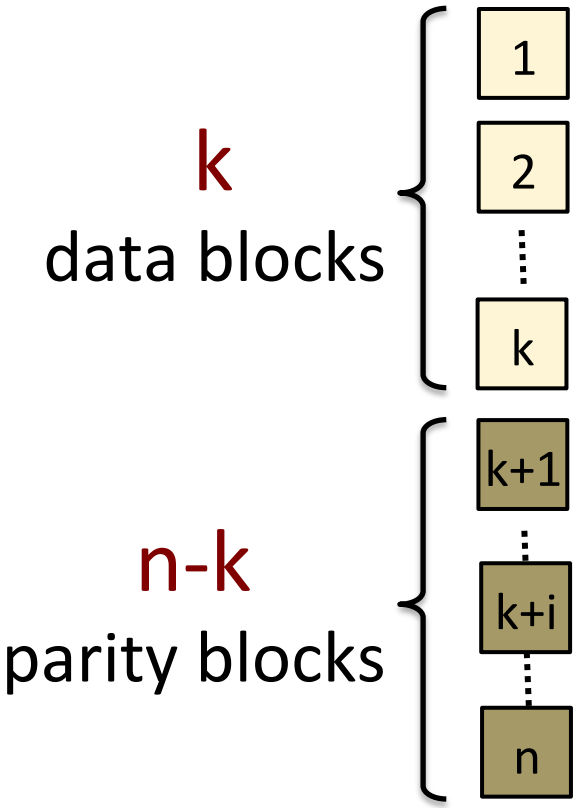
MSR framework

Minimum Storage Regenerating (MSR) framework[§]:

- Theoretical framework that allows optimizing storage and network-bandwidth costs
- Tells what is the minimum amount of transfer required for reconstruction

[§]Dimakis et al, IEEE Transactions on Information Theory, 2010

MSR framework

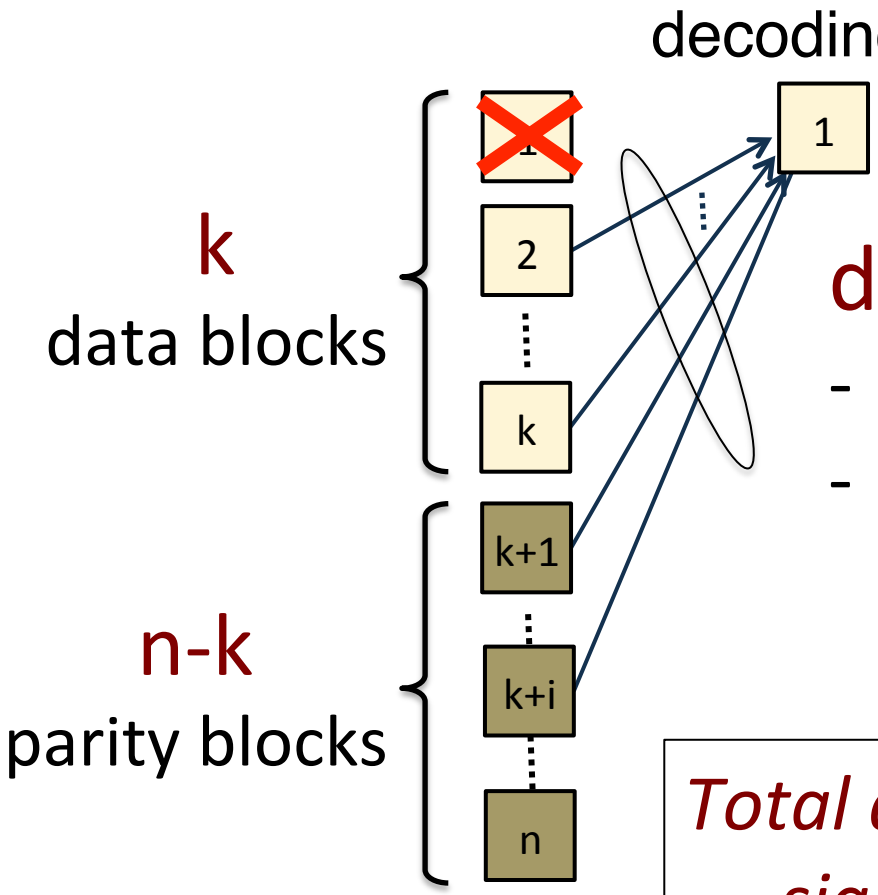


Just as in RS,

any k out of n blocks sufficient to recover all the data

=> Optimal storage & fault tolerance

MSR framework



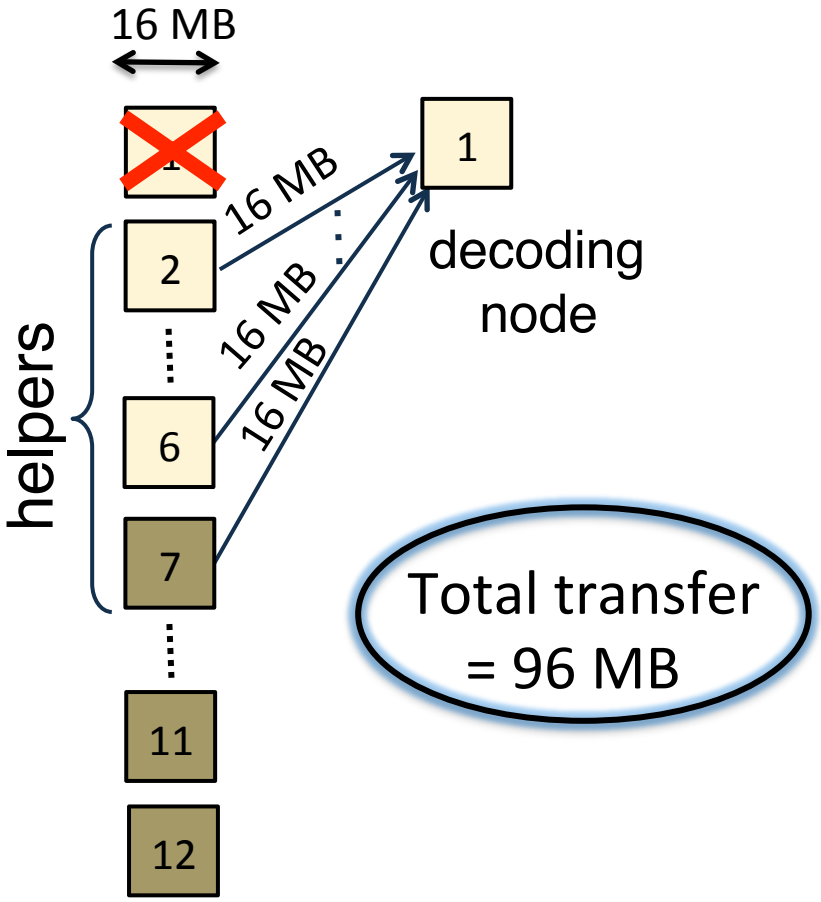
d helpers

- *any d of remaining (n-1)*
- *small amount transferred from each*

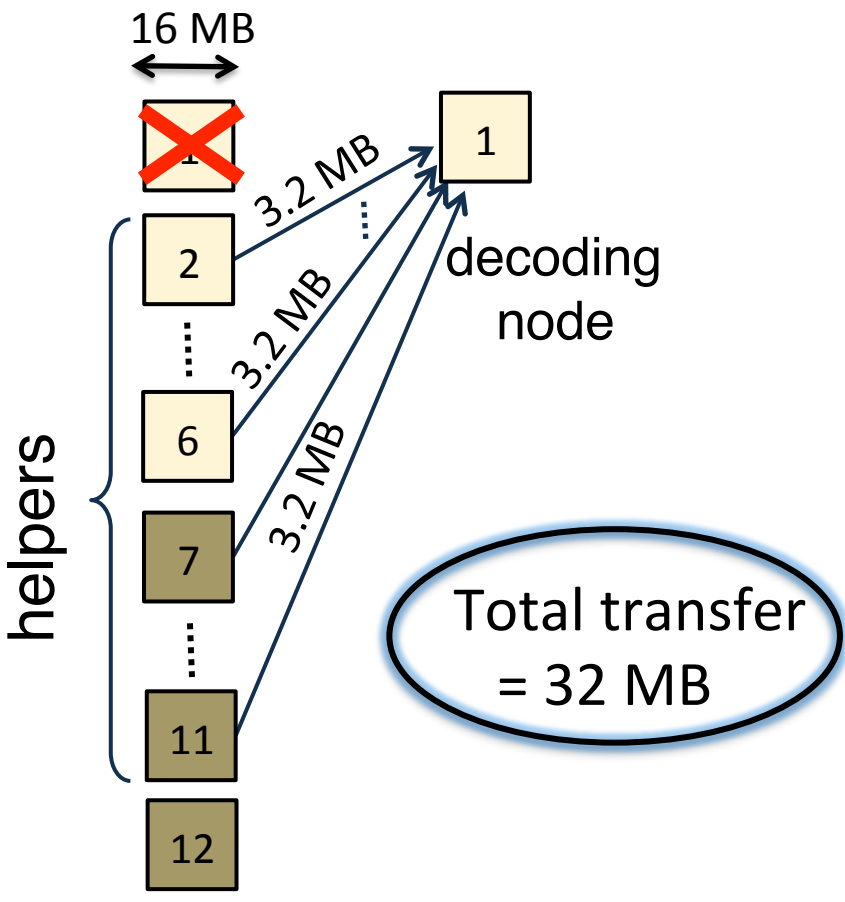
Total amount of data transferred:

- *significantly smaller than RS*
- *minimum possible*

Example: RS vs. MSR



Reed-Solomon (12, 6)

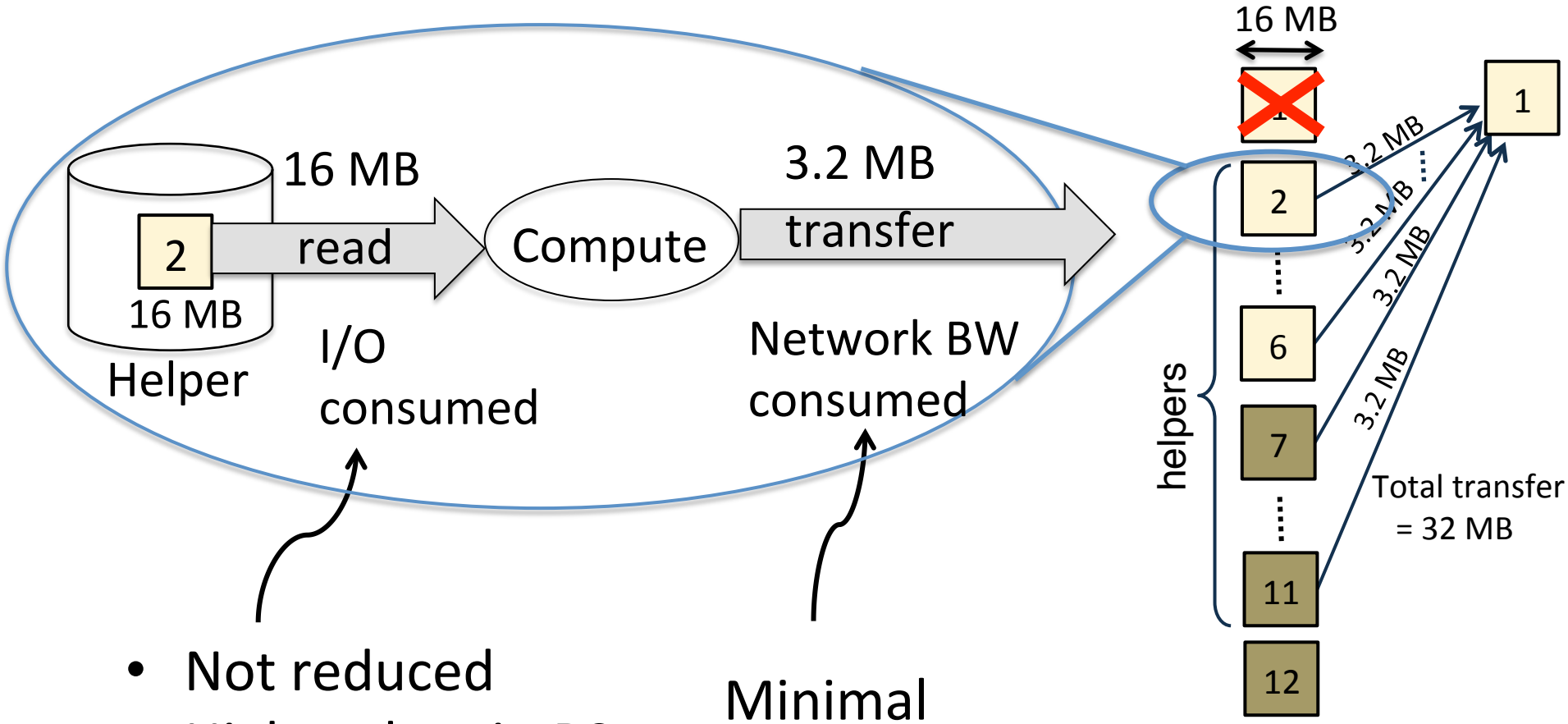


MSR framework with d = 10

MSR framework:

- optimizes storage and amount of data transfer
- *but not I/O consumed at helpers*

I/O in MSR Framework



- Not reduced
- Higher than in RS
  - 6 helpers in RS
  - 10 helpers in MSR

Minimal

MSR framework
with d = 10

Having your cake and eating it too..

In general, codes under MSR:

- optimal storage and fault tolerance 
- optimal network bandwidth 

but..

- *do not optimize I/O* 

Optimize I/O as well while retaining storage and bandwidth optimality

In this talk

- Two algorithms that together *transform MSR codes into codes that are I/O efficient as well*
  - while retaining storage and bandwidth optimality
  - Algorithm 1: transforms to minimizes I/O cost "locally" at each helper block
  - Algorithm 2: builds on top of Algorithm 1 to minimize I/O cost "globally" across all blocks

Have your cake and eat it too…

- Apply to *Product-Matrix MSR (PM)* [§] codes: a class of *practical MSR codes*
  - *transformed code = "PM_RBT"*
- PM codes exist for parameters:
  - storage overhead ~2x (or higher)
  - provides high (optimal) fault tolerance
  - useful in applications which need high fault tolerance, e.g., peer-to-peer storage

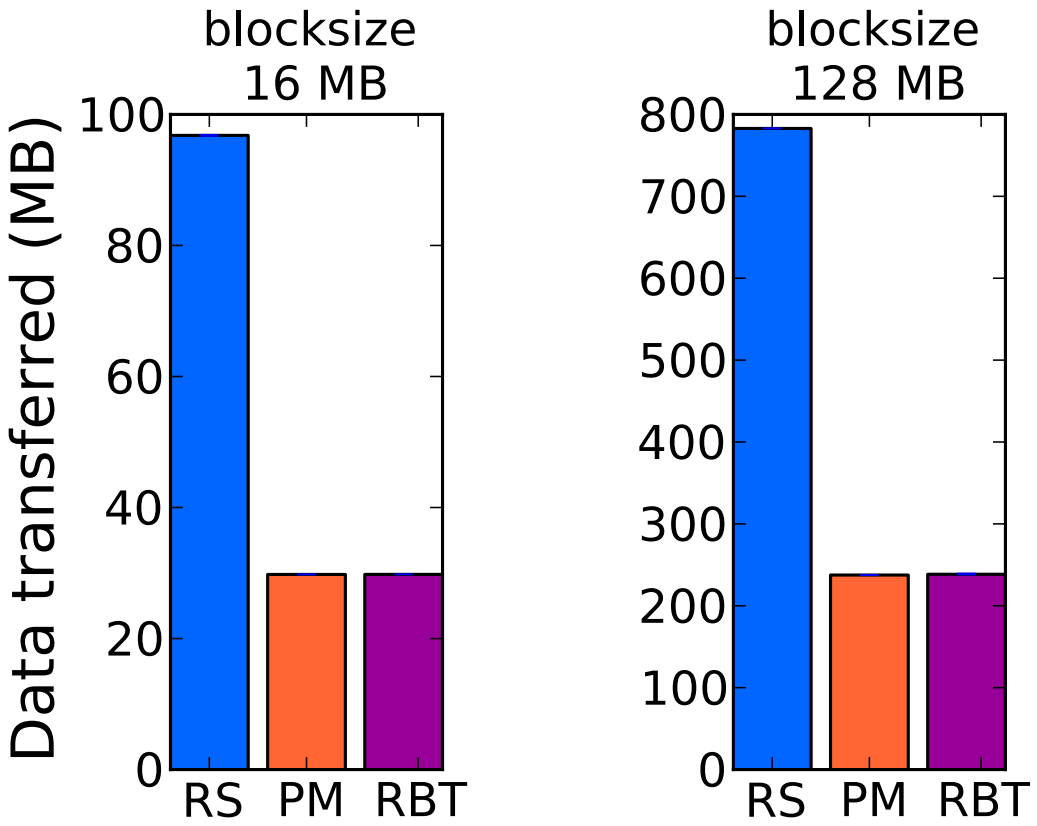
[§] Rashmi et al, IEEE Transactions on Information Theory, 2011

Implementation & Evaluation

- PM and PM_RBT in C
- Evaluation on Amazon EC2 instances
- Use Jerasure2[§] and GF-Complete[§] for finite-field arithmetic and RS

[§]Plank et al, 2013, 2014

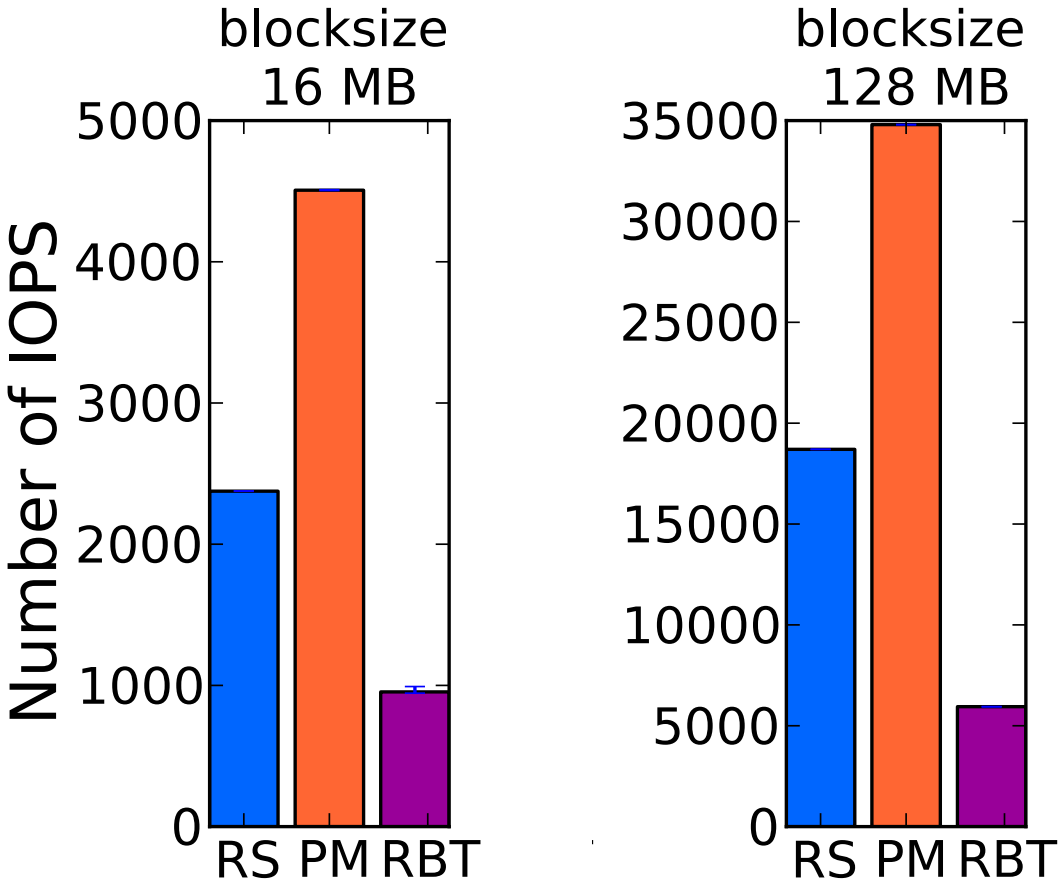
Amount of data transfer



- k = 6, d = 11
- PM and RBT both have *3.27x lower transfers*

PM-RBT retains optimality in network transfers during reconstruction as in PM

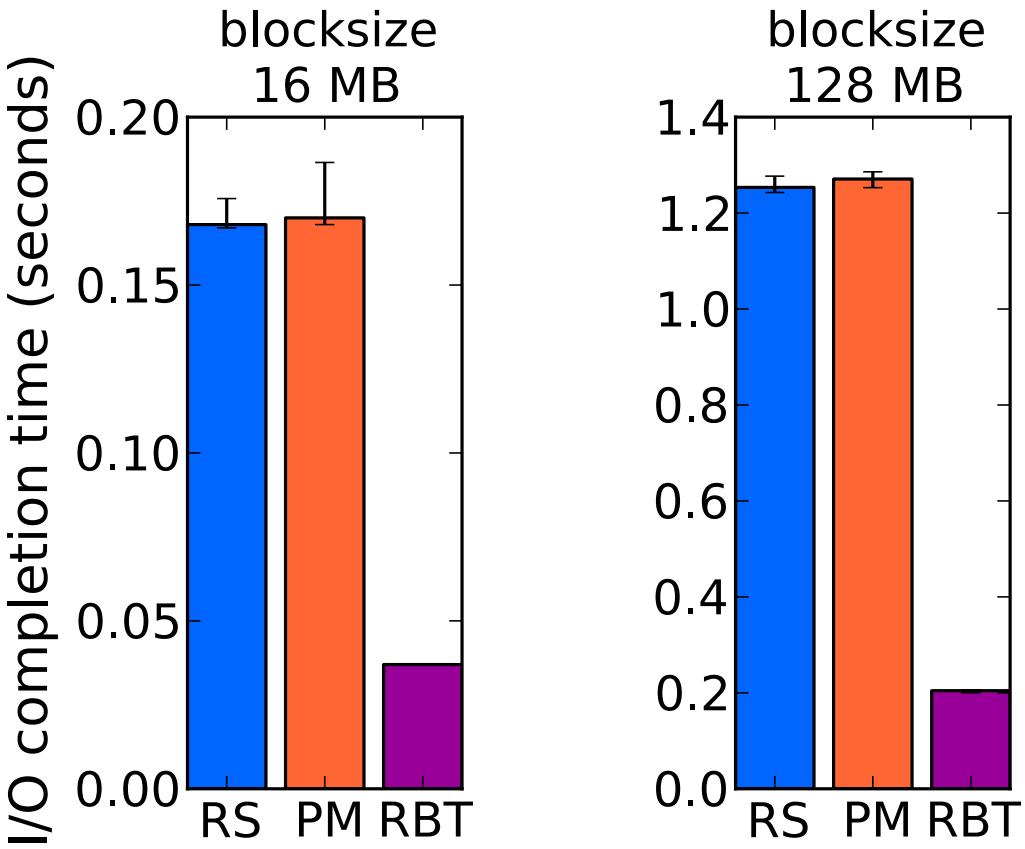
Number of IOPS consumed



- k = 6, d = 11
- RBT:
  - *5x lesser IOPS* as PM
  - *3x lesser IOPS* as RS

PM-RBT results in significant savings in IOPS

I/O completion time

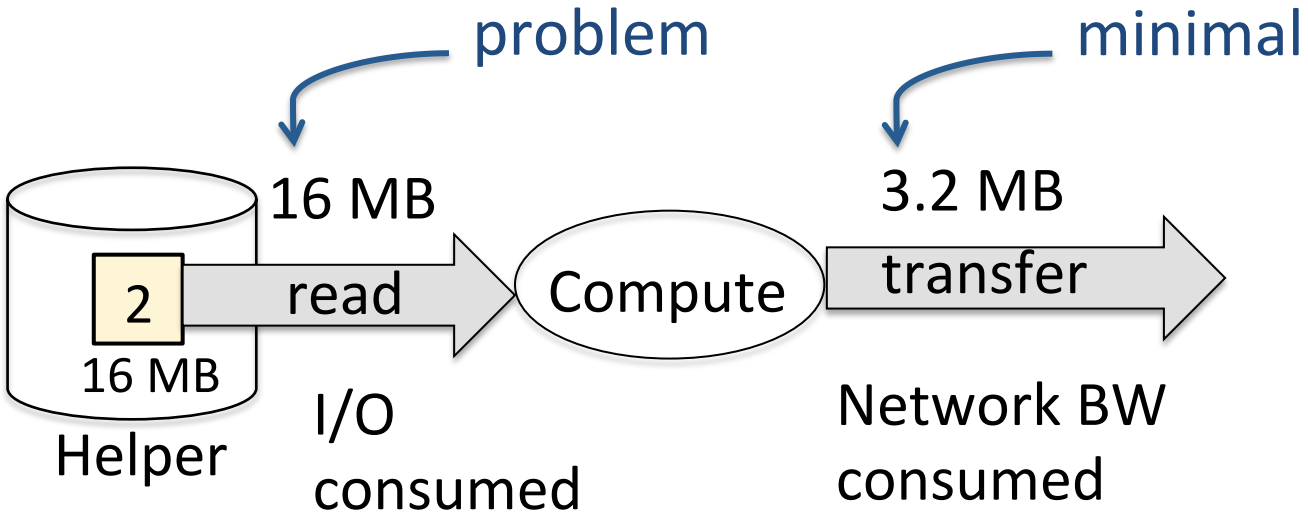


- k = 6, d = 11
- RBT:
  - *5x - 6x faster I/O*

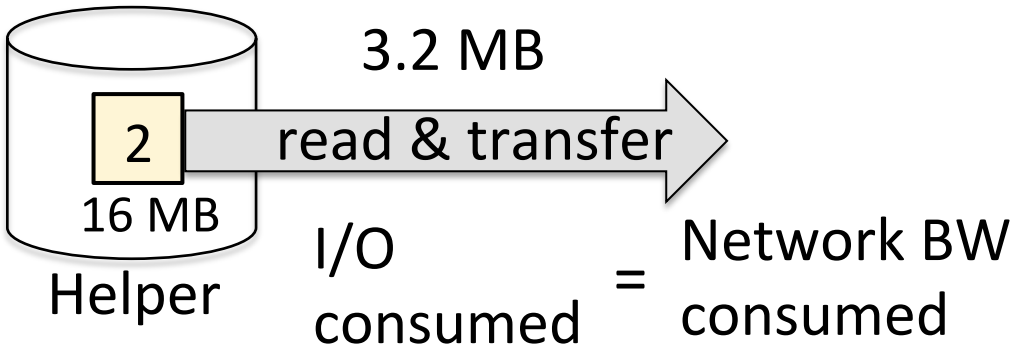
PM-RBT results in significantly faster completion of I/O

Algorithm 1

MSR framework:

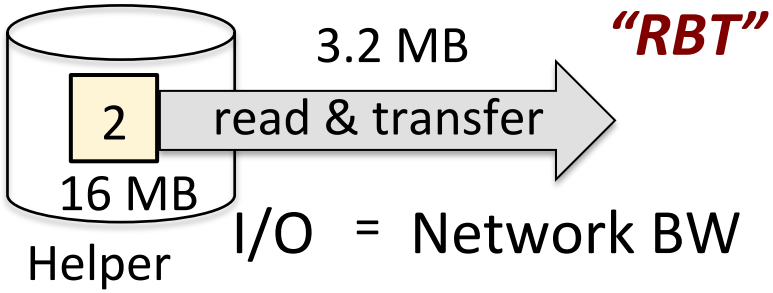


We would like:



"Reconstruct-by-transfer (RBT)"

Algorithm 1

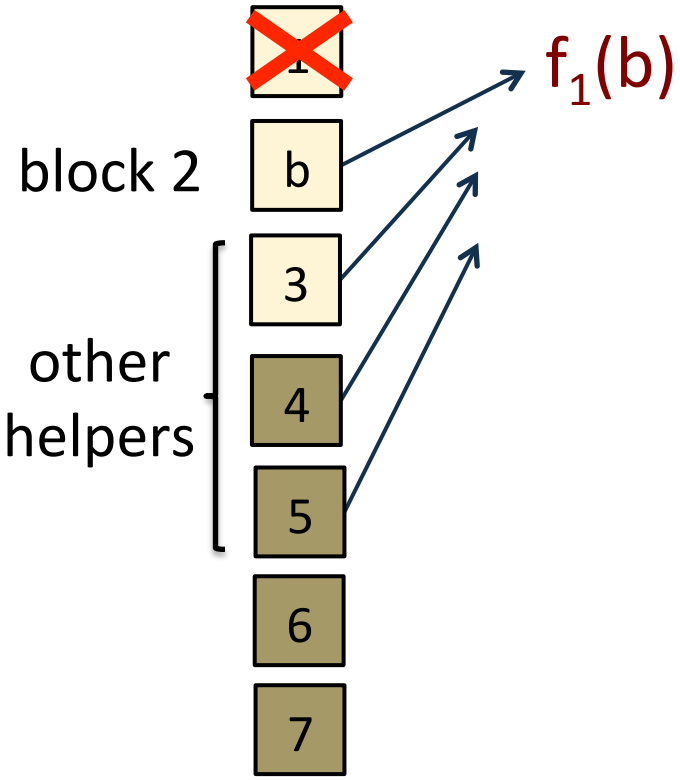


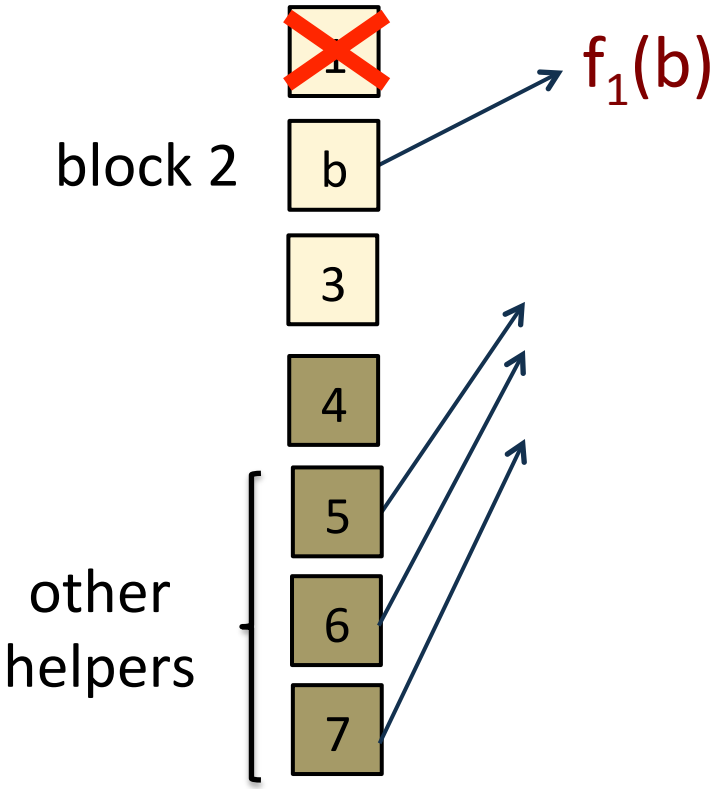
- Transforms MSR codes to achieve RBT
- Applicable to all MSR codes that satisfy two properties

Property 1: Independence between helpers

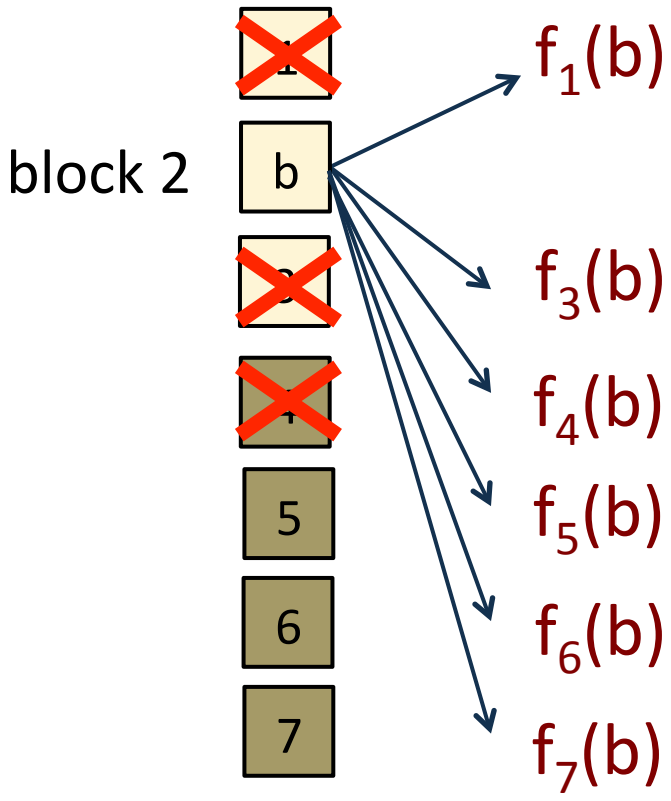
Function computed at a helper is *not dependent* on which other blocks are helping

E.g.: #helpers = 4



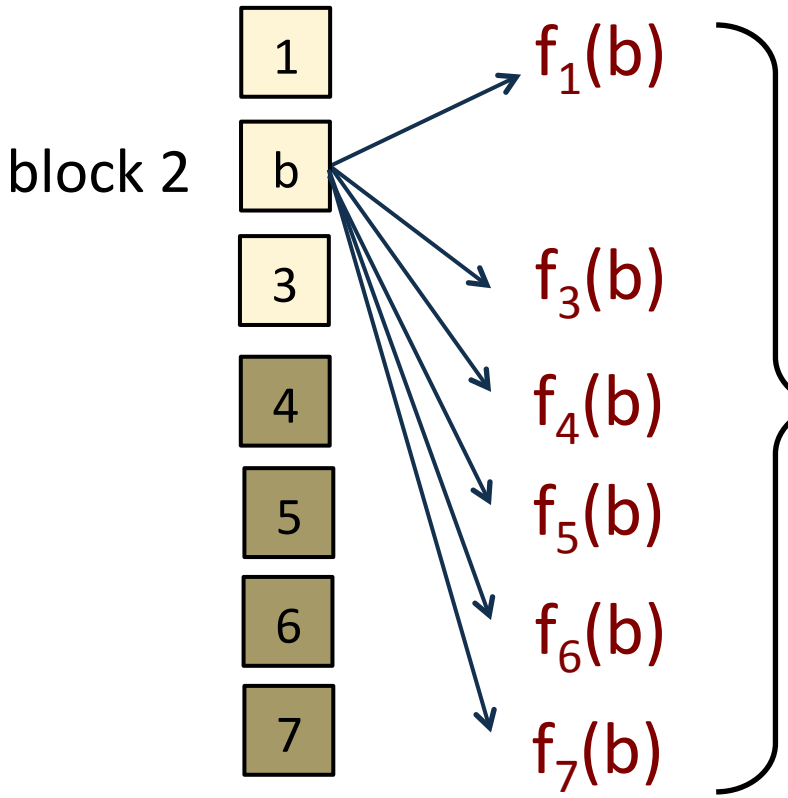


When property 1 is satisfied..



A block computes *pre-determined functions* to aid in reconstruction of each of the other blocks

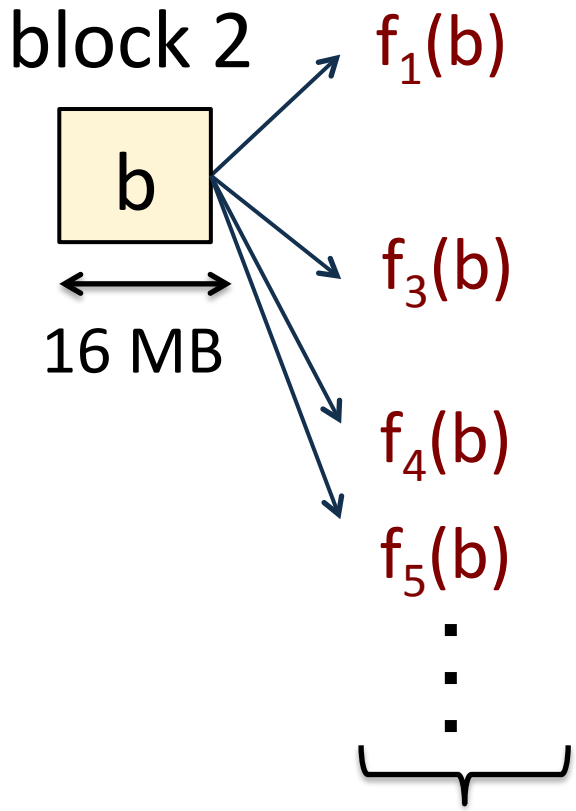
Property 2: Independence between functions computed at helper block



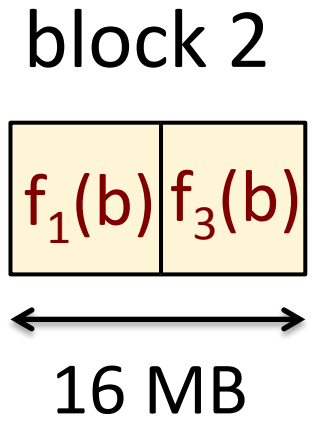
Any subset of size equal to block size are independent

- for k =3, d=4 as in example, *each function is half the block size*
- any two of these functions must be independent

Algorithm 1: Precompute and Store



transform



block 2 can help reconstruct blocks 1 & 3 in RBT-fashion

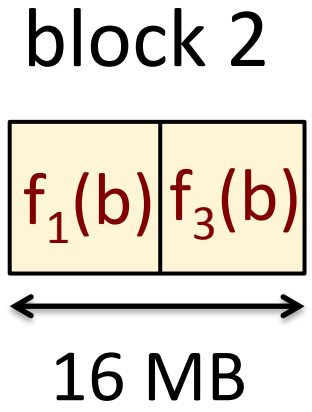
- Half block size
- Any 2 independent

Optimal I/O at each RBT-helper

- Under MSR, a block does minimum I/O when helping in RBT-fashion

  amount read = amount transferred

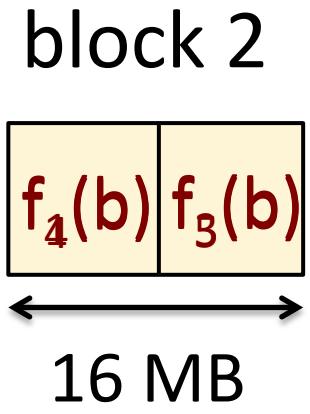
  = minimum possible (due to MSR)



- Minimum I/O when helping blocks 1 & 3
- For other blocks, reads full block and does computation

How to choose RBT-helper assignment?

- Algorithm 1 takes the assignment of ‘who acts as RBT helper to whom’ as input



How to choose this assignment?

Algorithm 2

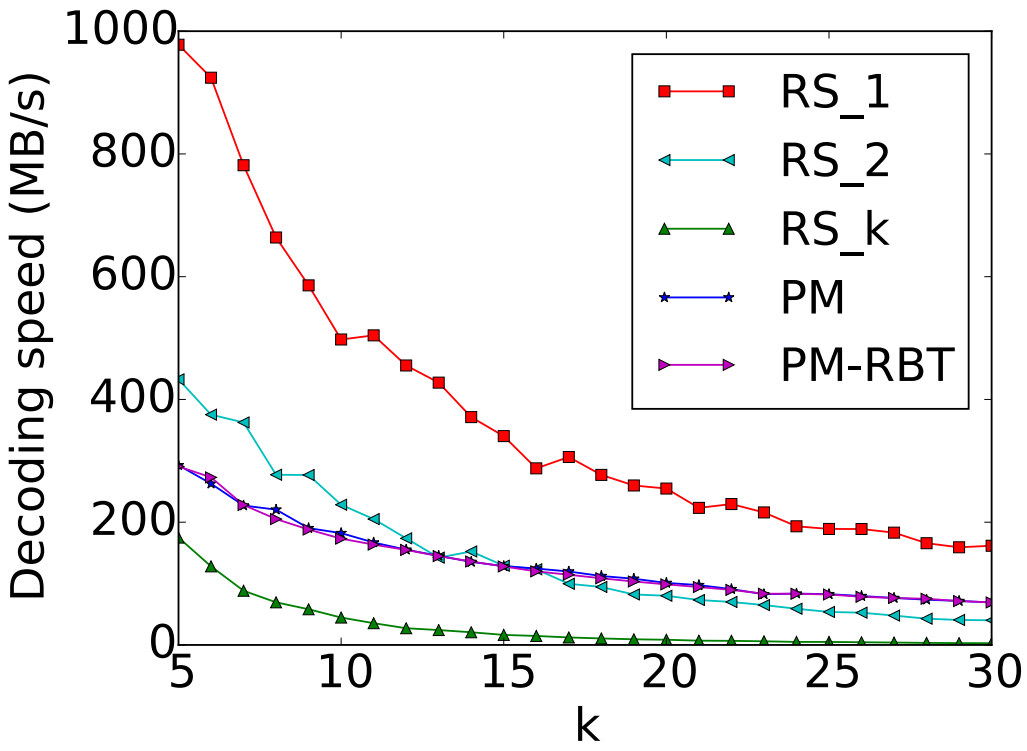
- Chooses RBT-helper assignment to *minimize I/O cost globally across all blocks*
- Greedy algorithm
- Optimal!

Algorithm 2: Two extreme cases

- Complete preferential treatment for data blocks
  - *Each block RBT-helps data blocks*
  - "SYS" pattern
- Equality for all: no preferential treatment
  - *Each block RBT-helps following blocks*
  - "CYC" pattern

Impact of the transformation on encoding and decoding speeds

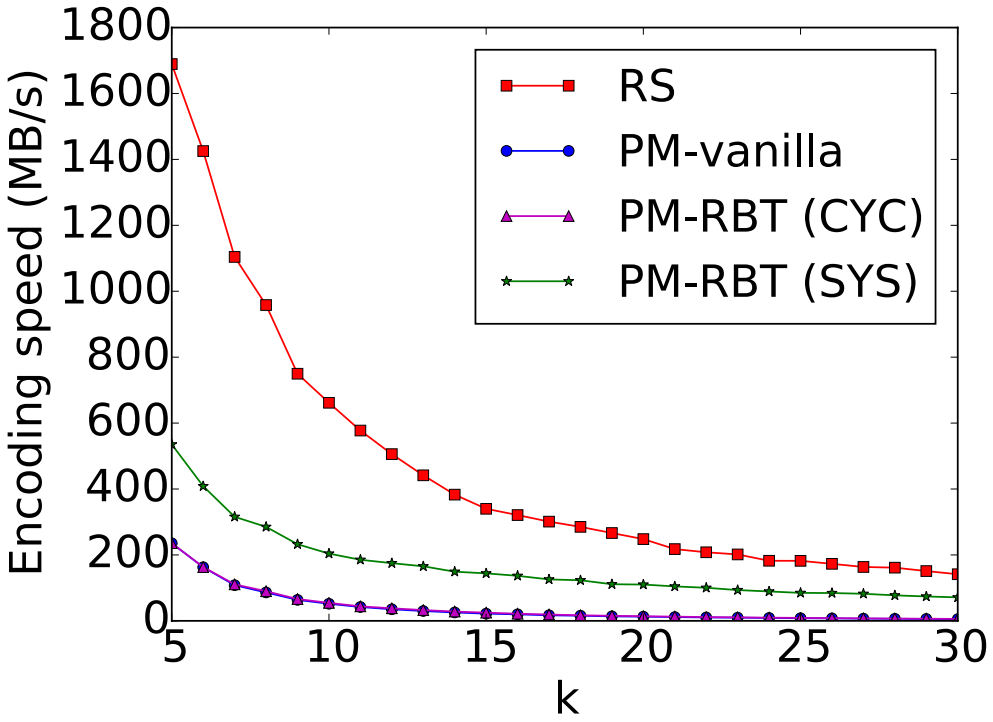
Decoding speed (computation for reconstruction)



- n=2k, d = 2k-1
- Single thread
- RS_m: m parity blocks and remaining (k-m) data blocks helping

- RBT does not affect the decoding speed of PM
- Similar to RS decoding with two parities

Encoding speed (parity generation)

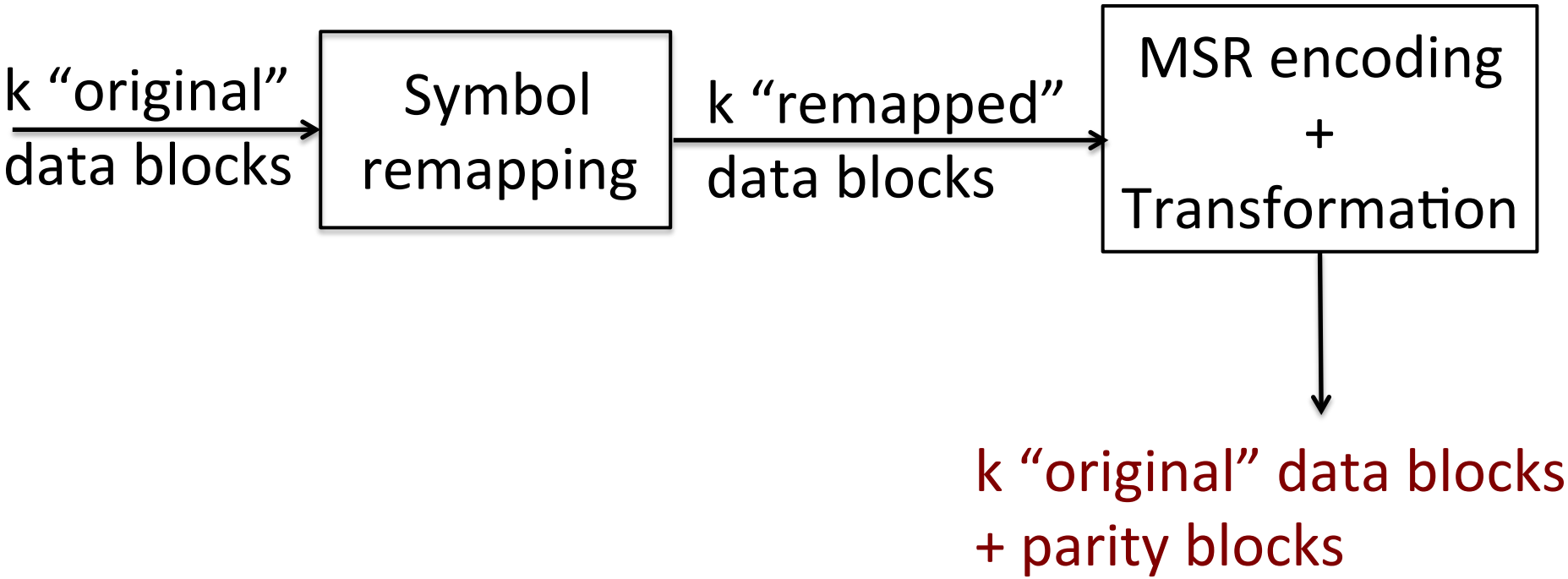


- Single thread
- n=2k, d = 2k-1

- Slower than RS but still practical
- RBT-SYS has higher encoding speed than PM

No “original” data blocks left?

- Use *“symbol-remapping”* to retain original data blocks



Summary

- Algorithms to transform MSR codes
  - optimize I/O retaining storage and network optimality
- Implemented and evaluated application onto Product-Matrix MSR codes
  - significant reduction in I/O costs
- Analytical results on optimality



Thanks!