
# Optimize Storage Efficiency & Performance with Erasure Coding Hardware Offload

Dror Goldenberg
VP Software Architecture
Mellanox Technologies

# SNIA Legal Notice

- The material contained in this tutorial is copyrighted by the SNIA unless otherwise noted.
- Member companies and individual members may use this material in presentations and literature under the following conditions:
  - Any slide or slides used must be reproduced in their entirety without modification
  - The SNIA must be acknowledged as the source of any material used in the body of any document containing material from these presentations.
- This presentation is a project of the SNIA Education Committee.
- Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be, or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.
- The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.

# Abstract

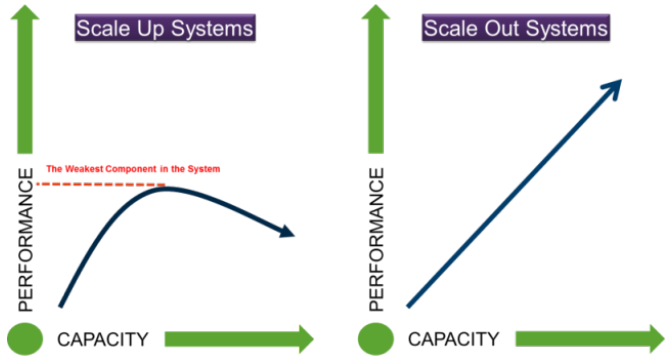

- Optimize Storage Efficiency & Performance with Erasure Coding Hardware Offload

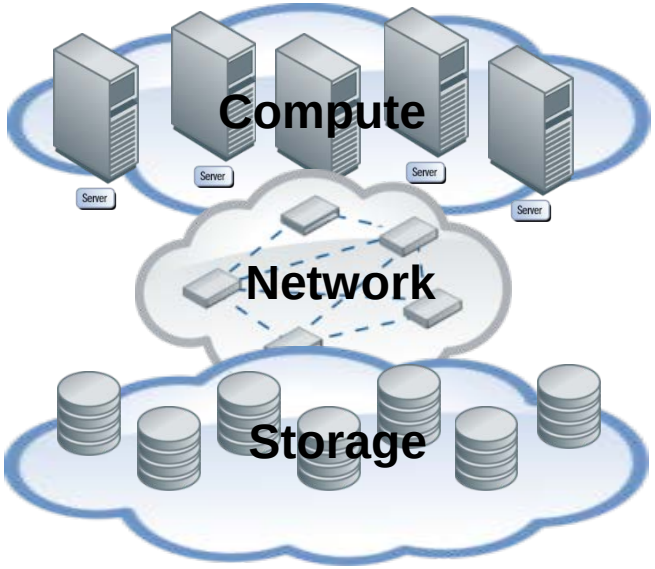
  Nearly all object storage, including Ceph and Swift, support erasure coding because it is a more efficient data protection method than simple replication or traditional RAID. However, erasure coding is very CPU intensive and typically slows down storage performance significantly. Now Ethernet network cards are available that offload erasure coding calculations to hardware for both writing and reconstructing data. This offload technology has the potential to change the storage market by allowing customers to deploy more efficient storage without sacrificing performance. Attend this presentation to learn how erasure coding hardware offloads work and how they can integrate with products such as Ceph.

- Learning Objectives
  - Learn the benefits and costs of erasure coding
  - Understand how erasure coding works in products such as Ceph
  - See how erasure coding hardware offloads accelerate storage performance

# Software Defined Storage

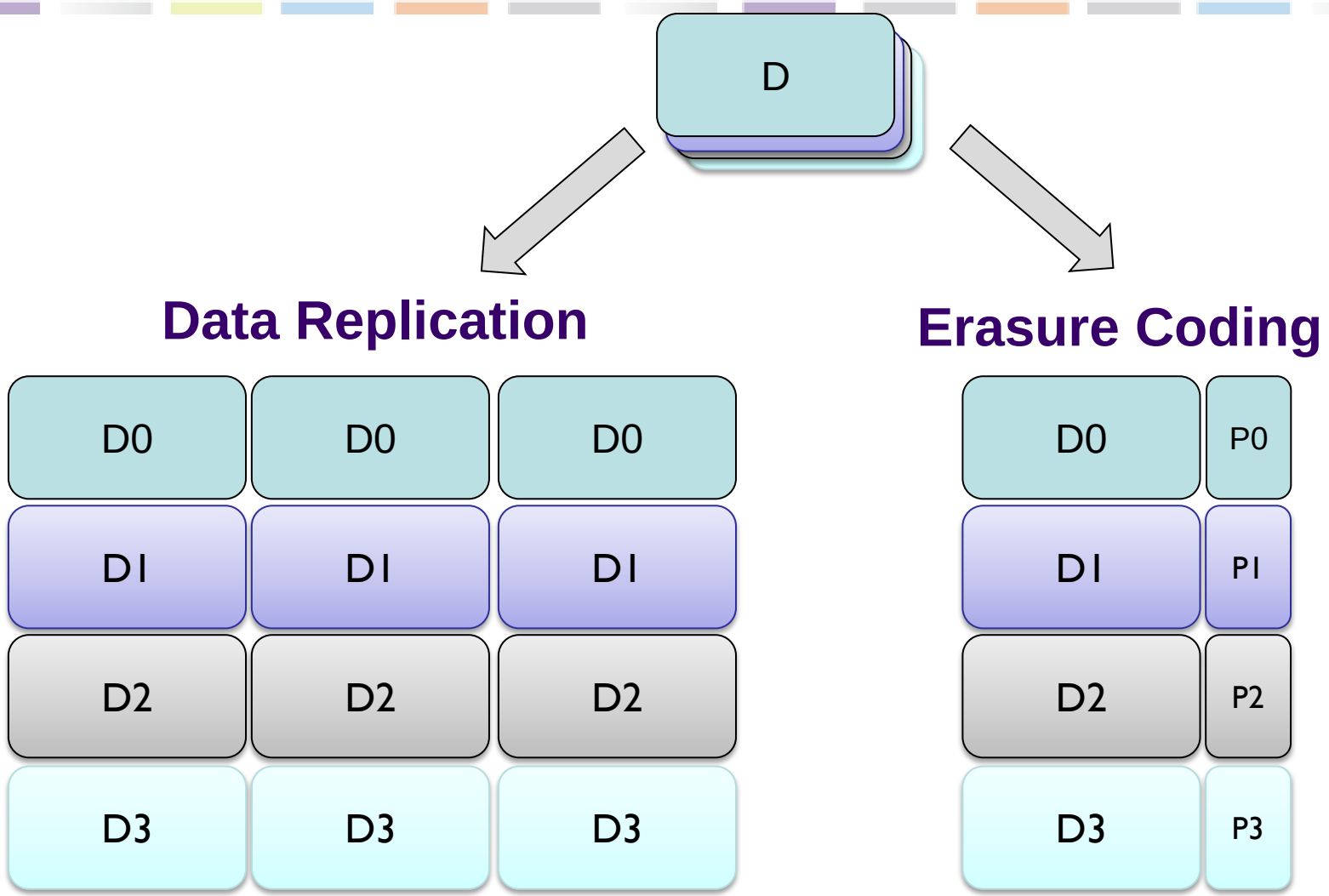
- Software Defined Datacenter
  - Software Defined Networks
  - Software Defined Storage
- Scale Out Advantage



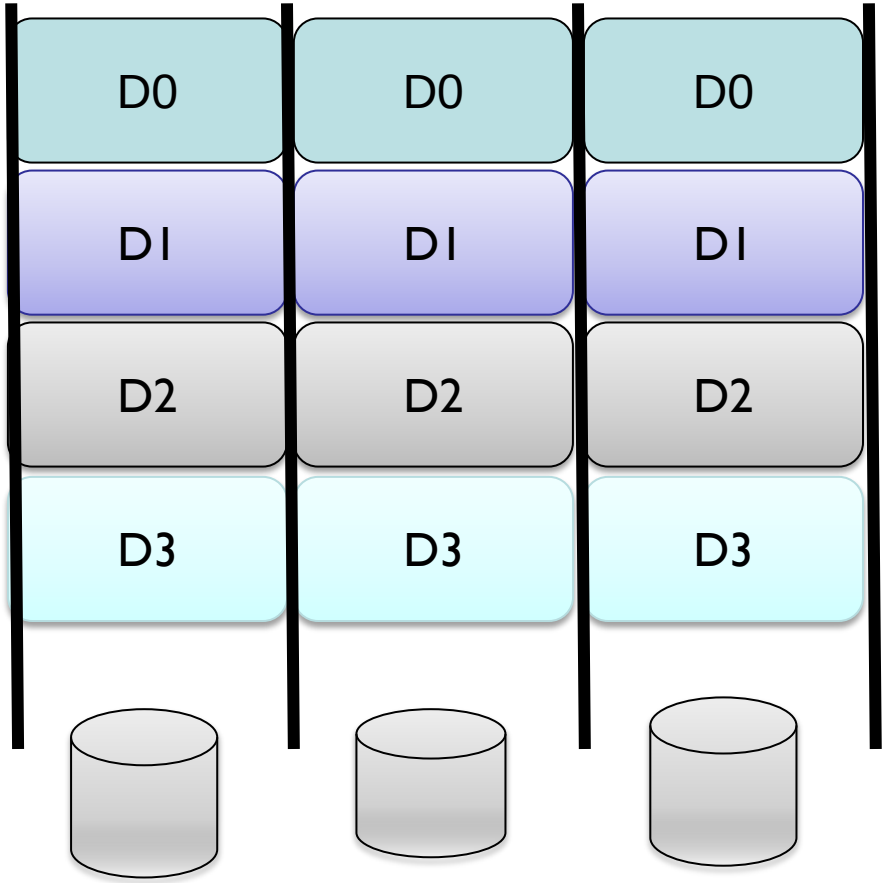

- Scale-out Storage Examples
  - Ceph, Swift, Gluster and many more

# Ensuring Data Availability

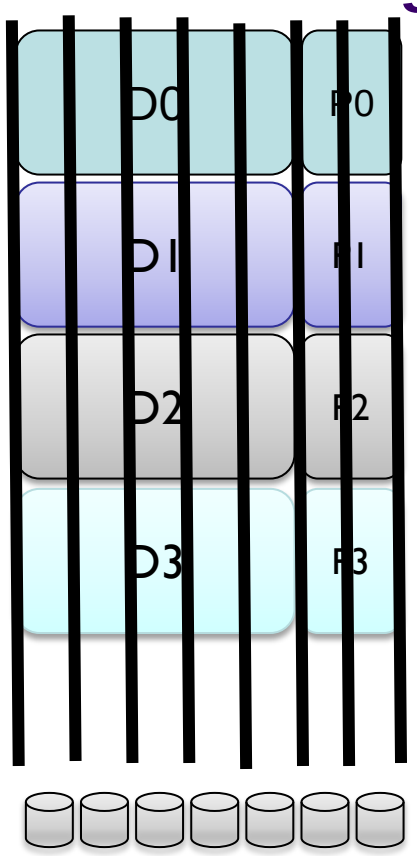


# Ensuring Data Availability

## Data Replication

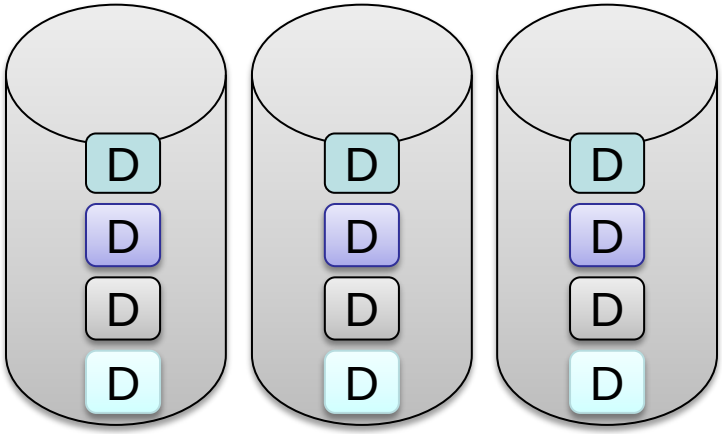



## Erasure Coding



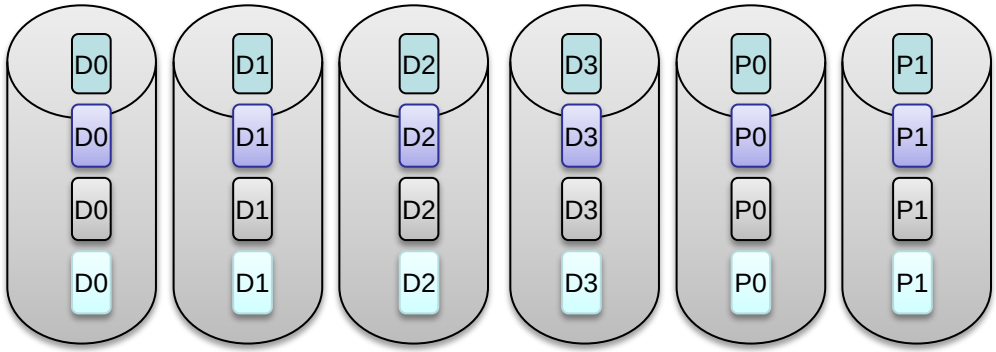
# Ensuring Data Availability - Summary

## Data Replication




- Capacity: 3x data typical
- Resilient to 2 failures

## Erasure Coding



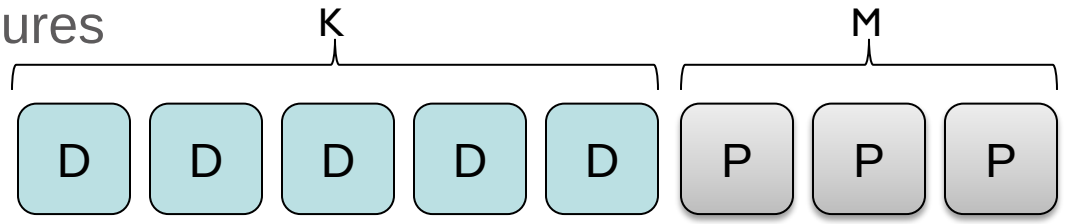

- Capacity: 1.4x data typical
- Better failure resilience
  - e.g. 10+4: 1.4x capacity, 4 failures
- CPU & network hungry
- Longer & data intensive rebuild
- Partial update requires read

# The Value of Erasure Coding Offload

- System cost
  - Reduce overall system cost – use cheaper CPU
- Enable hyperconverged systems
  - Efficient compute - reduce overall storage overhead
- Enable value add storage disaggregation
  - Lightweight clients can efficiently add data availability
- Efficient data recovery/rebuild
  - Systems spend long time in degraded state during rebuild
  - Ensure rebuild has minimal impact on operational characteristics
- Network efficiency with EC calculation and I/O
  - Avoid extra interrupts and context switches
- Better parallelism

# Erasure Codes Theory

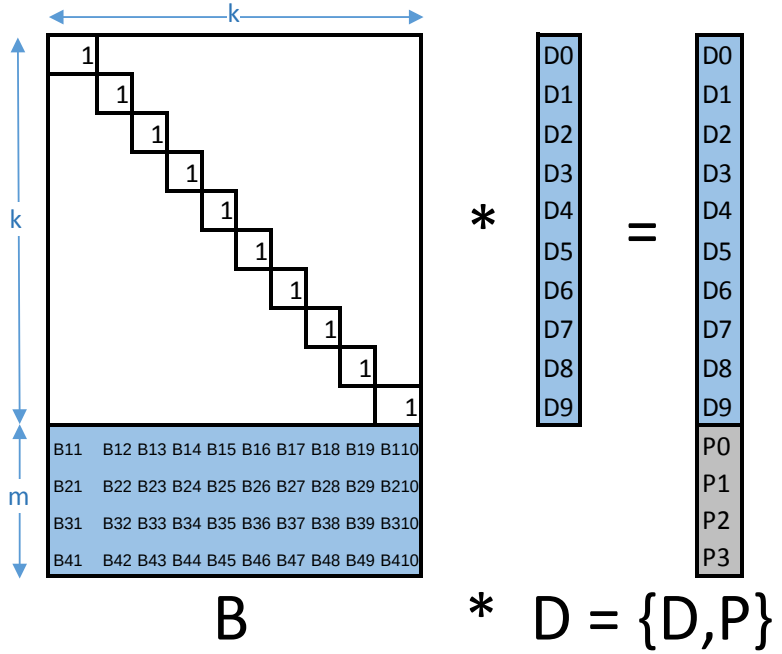
- K data + M parity = N total
  - Tolerates up to M failures
  - Total overhead N/K




- Systematic Codes – encoded data contains original data
- Maximal Distance Separable (MDS)
  - Can survive any M erasures
- Reed Solomon Coding – this session focus
  - Many other codes exist: RAID 6, XOR, Pyramid, LRC, …

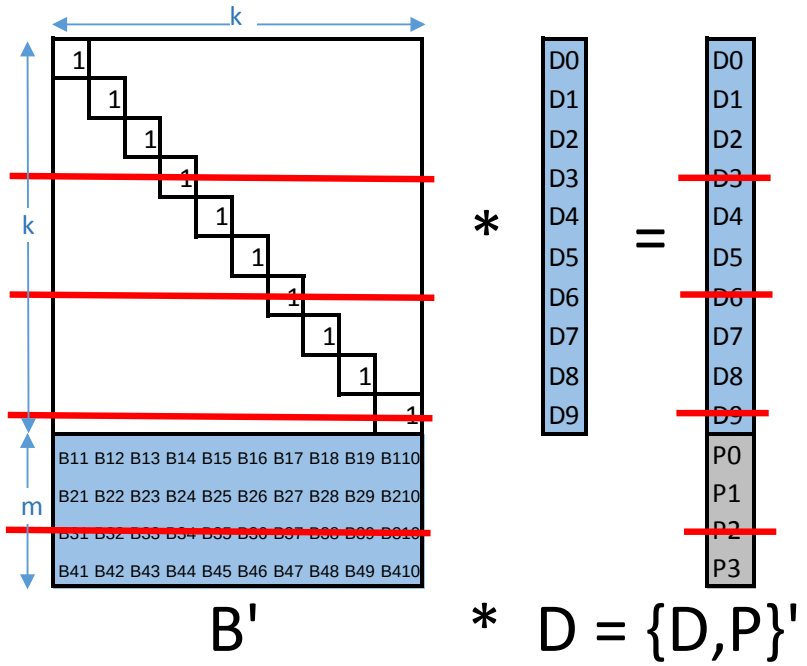
# Reed Solomon Encoding

- Generating parity is matrix multiplication



# Reed Solomon Decoding (1/2)

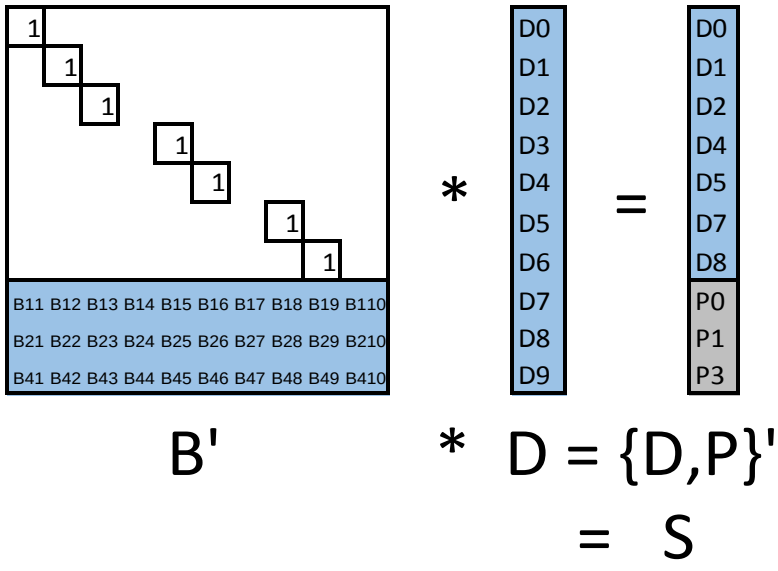
- In case of failures
  - Matrix is reduced
  - Calculate Inverse matrix
  - Recover Data
  - Recover Parity



# Reed Solomon Decoding (2/2)

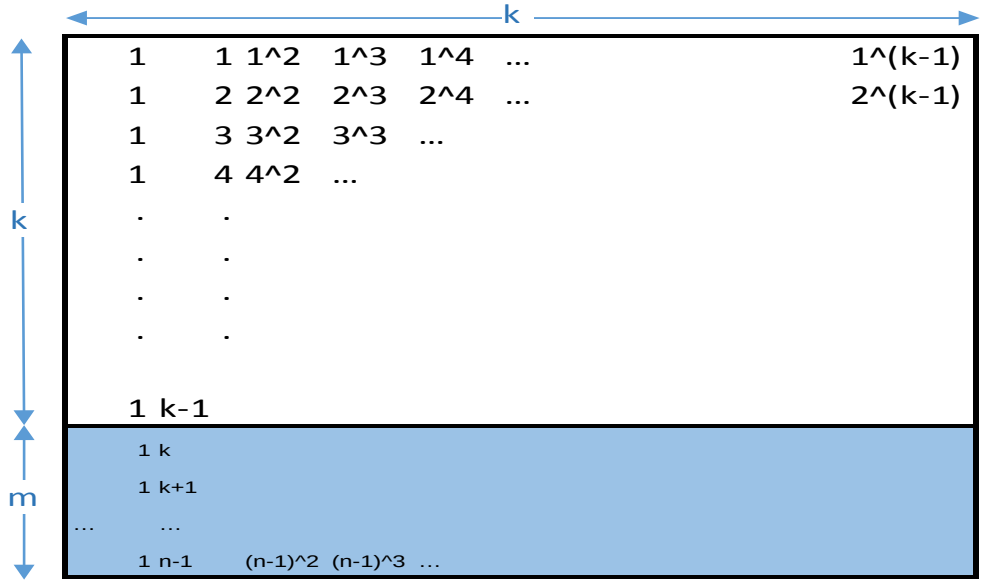
- Recovery - how?
  - B’ * D = S
  - B’$^{-1}$ * B’ * D = B’ $^{-1}$ * S
  - D=B’ $^{-1}$ * S
  - {D,P}=B * D

S = Survivors vector

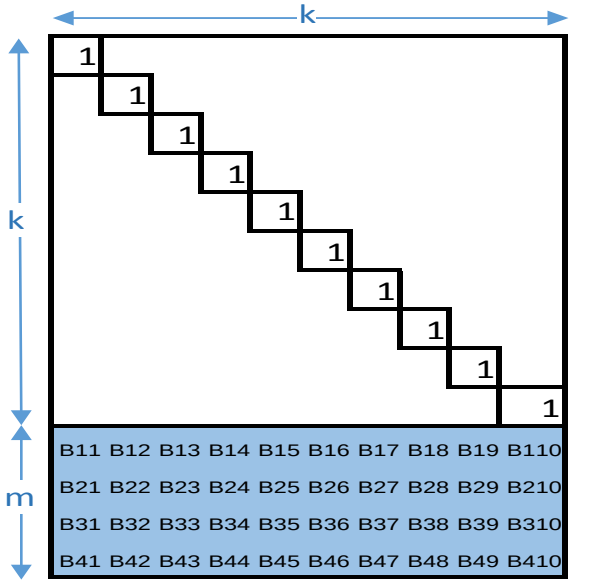


# Algebra Magic (1/2) – Matrix Generation

- All submatrices must be invertible
- Vandermonde matrix is used as baseline
- Derived matrix through elementary operations




Vandermonde Matrix




Derived Matrix

# Algebra Magic (2/2) – Arithmetic OPs

- Needed: finite field, multiplicative inverse
- Operation done over Galois Field – GF($2^w$)
  - Sum is XOR operation
  - Multiplication is more complicated…
    - Numbers are multiplied and then divided by an irreducible polynomial
- n must be ≤ $2^w$

# High CPU Demand

- Matrix multiplication compute needed
  - O(k*m) multiply-add operations
- Cache & TLB intensive (large data sets)

# Erasure Code History

- Évariste Galois 1811-1832

http://selunec.deviantart.com/art/Evariste-Galois-189586719

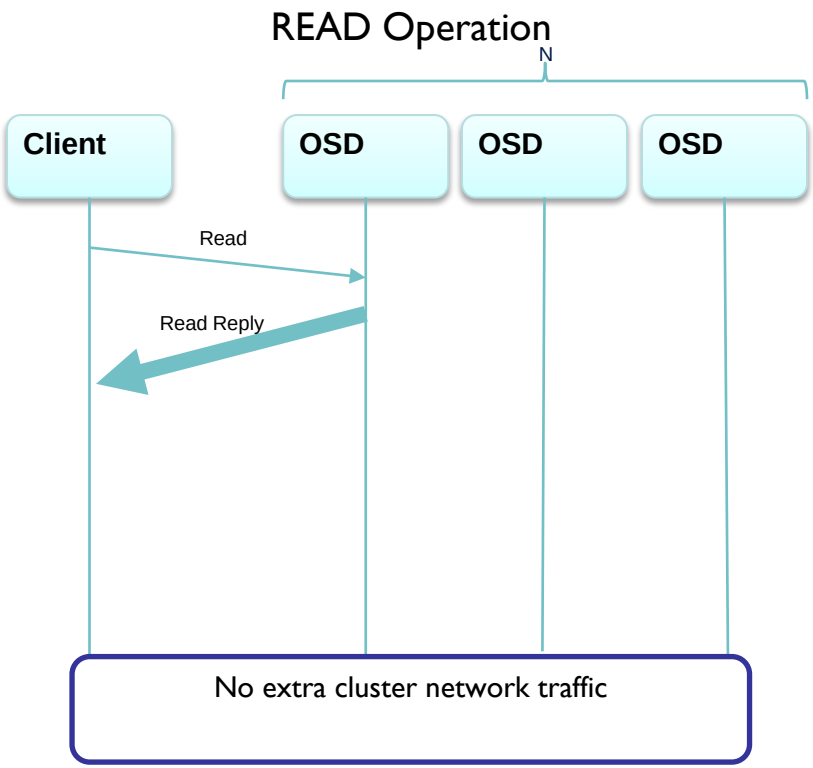
- Alexandre-Théophile Vandermonde 1735 – 1796

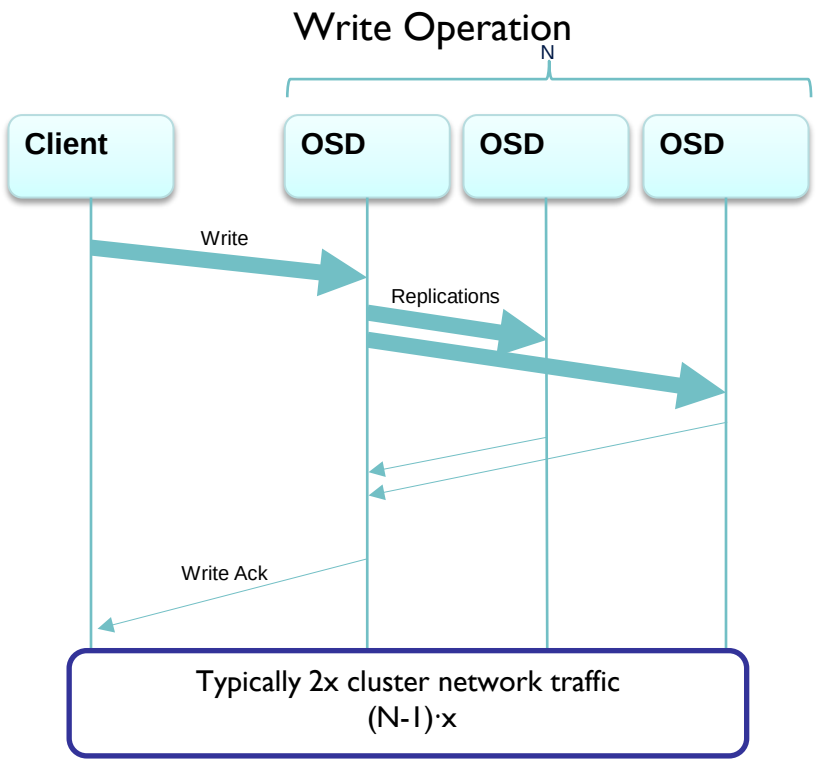
http://www.giancarlodurso.altervista.org/blog/interpolazione-polinomiale-con-matrice-di-vandermonde/

- Irving S. Reed 1923-2012
- Gustave Solomon 1930-1996

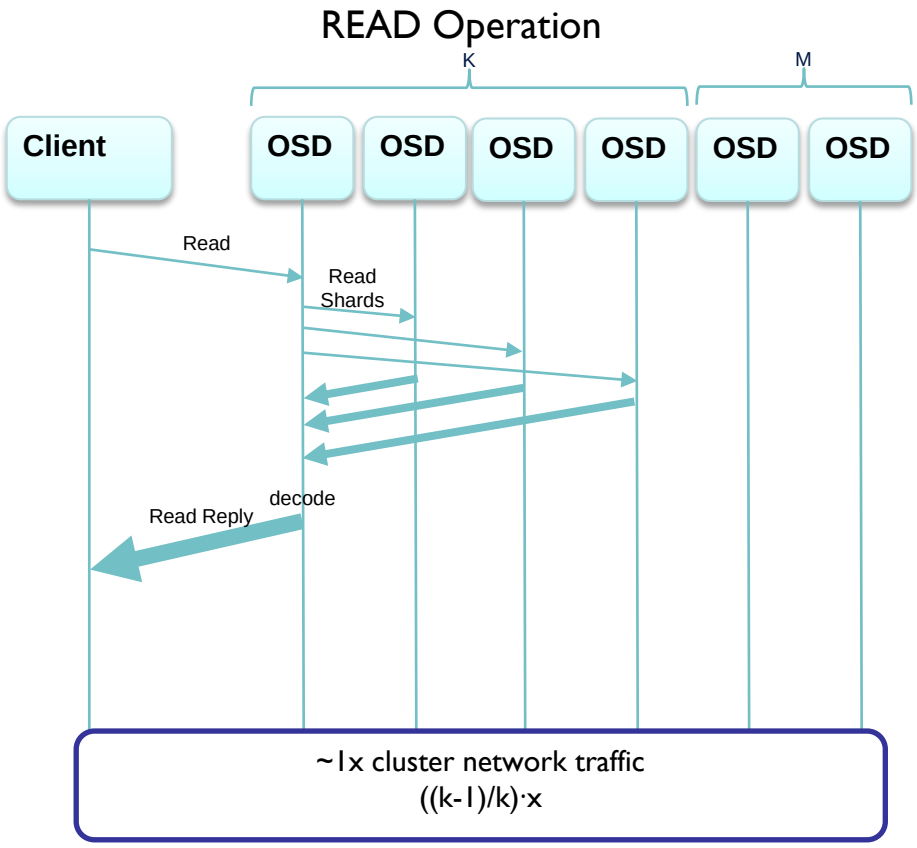
http://reedsolomon.tripod.com/

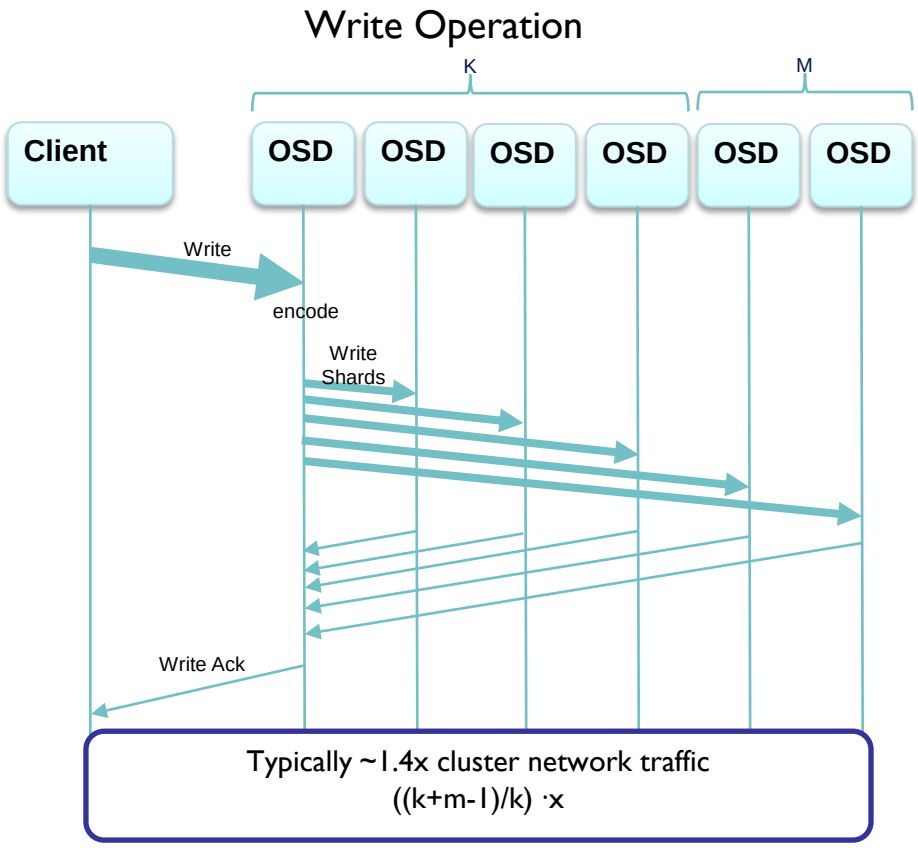
# Network Traffic – Sunny Day Scenario (Replication)



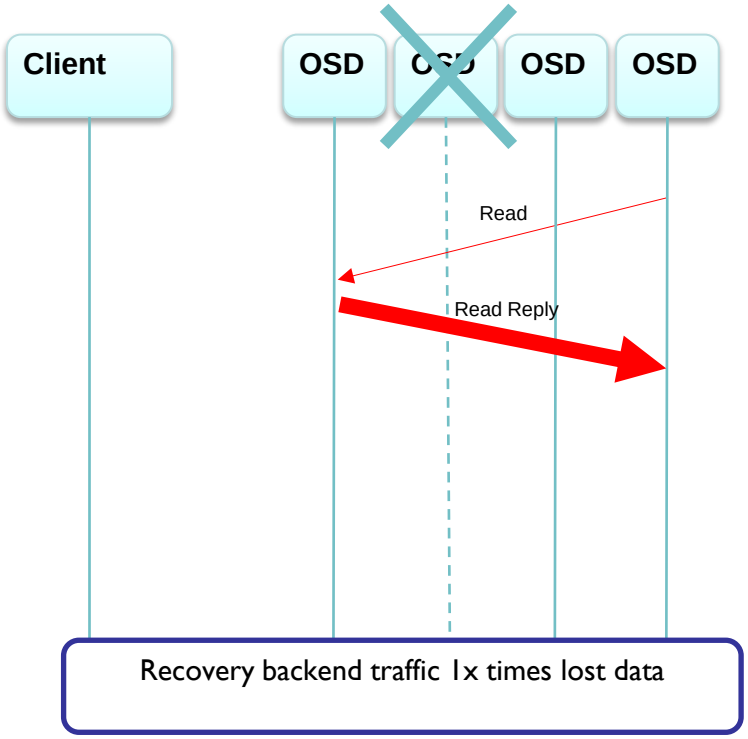



# Network Traffic – Sunny Day Scenario (Erasure Coding)



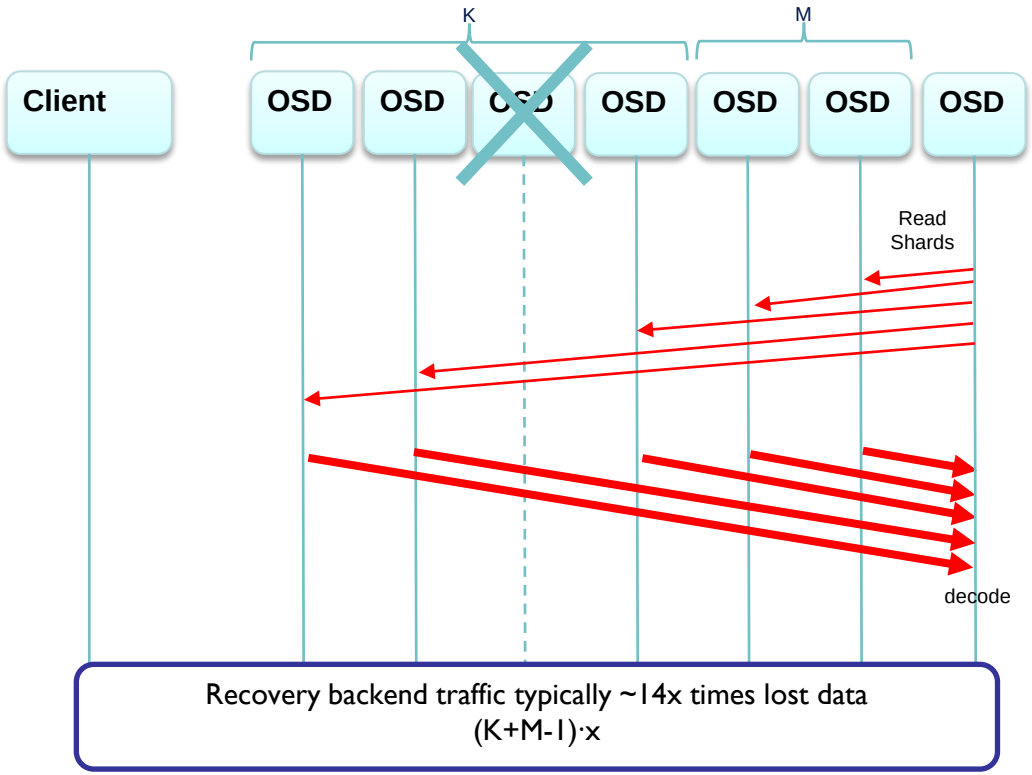



# Network Traffic – Recovery (Replication)




- Example - Time to recover
  - Net networking time to move data
  - 20TB system @40GE 1.1hrs
  - 200TB system @40GE 11.1hrs
- Similar flows for scrubbing

# Network Traffic – Recovery (Erasure Coding)




- Example - Time to recover (10+4)
  - Net networking time to move data
  - 20TB system @40GE 14.4hrs
  - 200TB system @40GE 144.4hrs
- Similar flows for scrubbing

**Tradeoff:**
recovery time
vs
storage efficiency

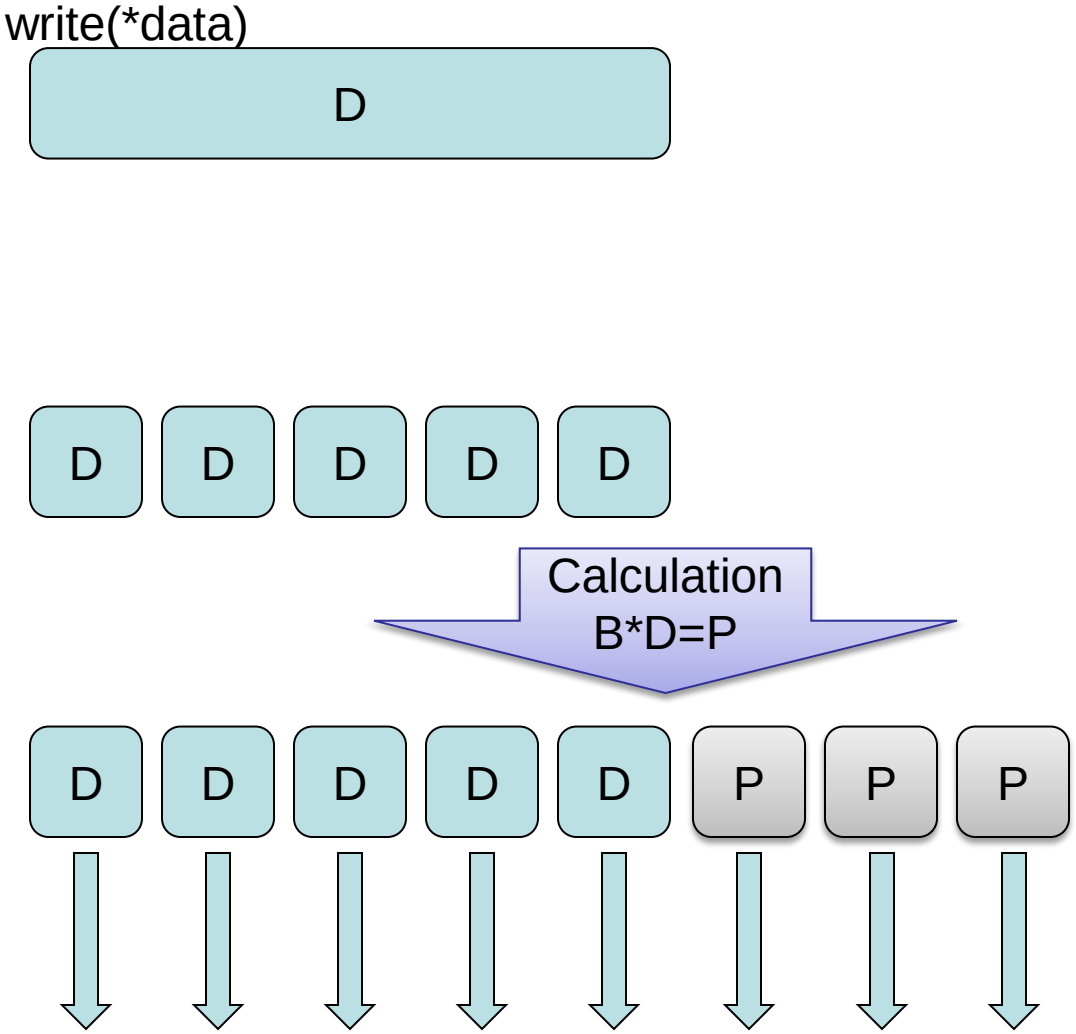
# Erasure Coding Offload Primitives

- Initialization

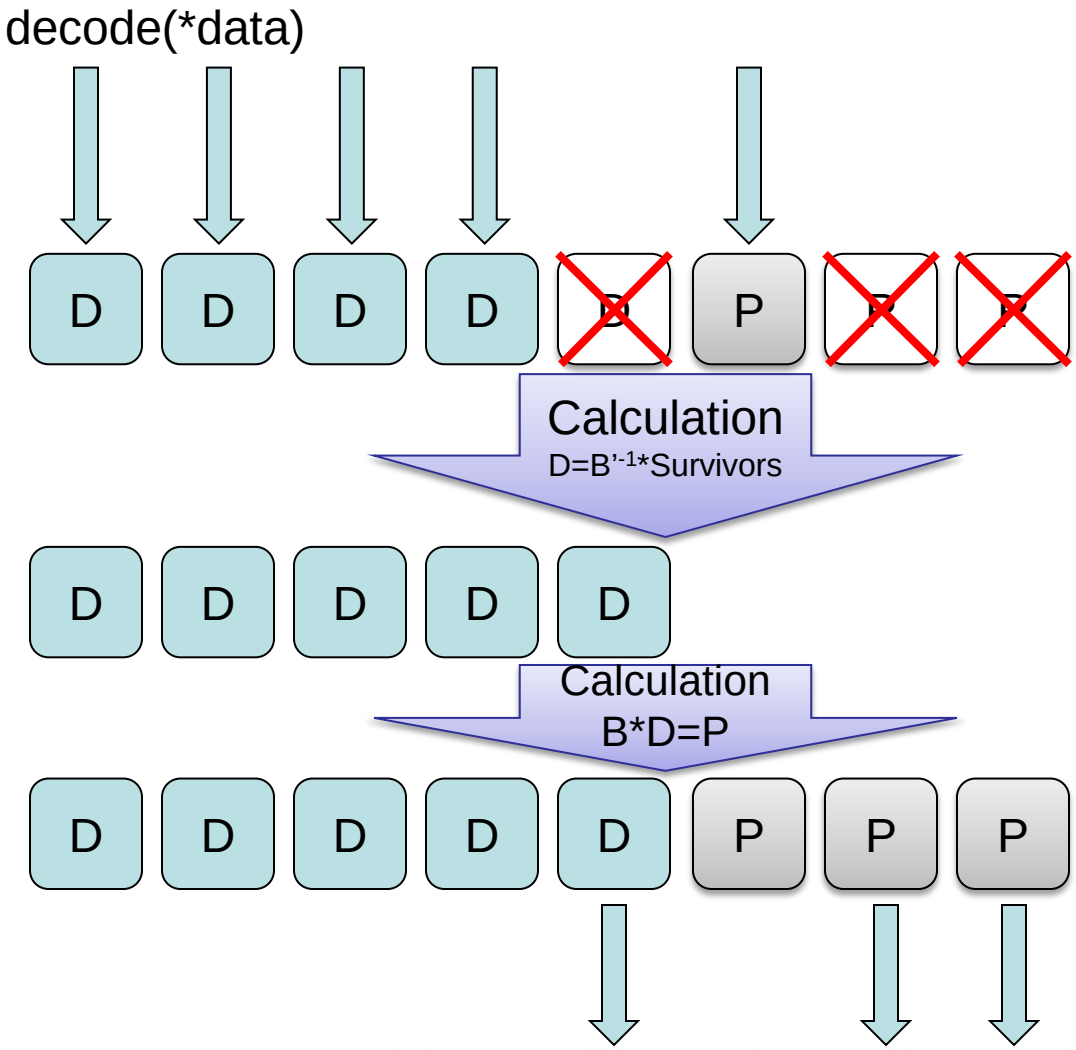
- Encode
- Decode
- Update

- Send

# Typical Workflow
# Long Write (Encode)



# Typical Workflow Reconstruct (Decode)



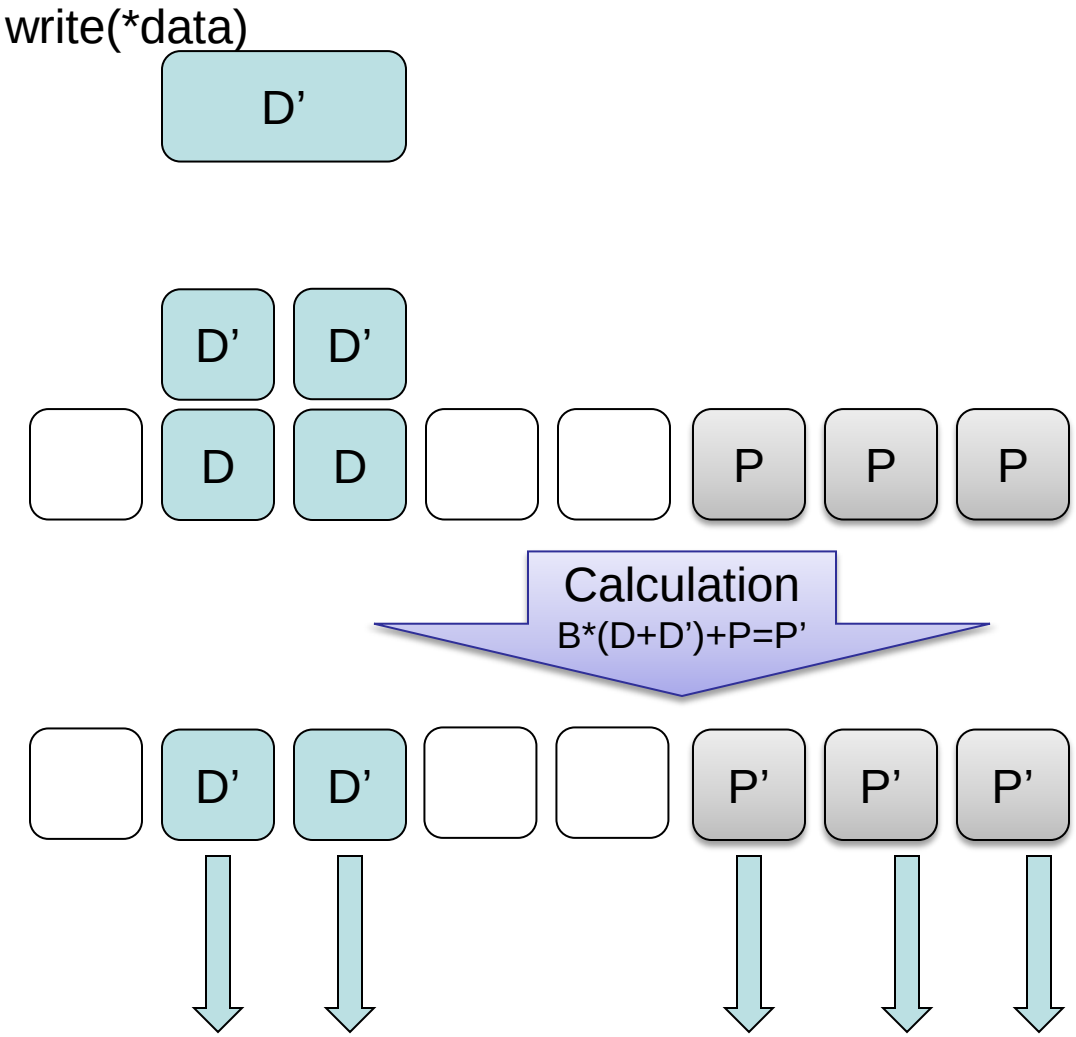

Retrieve Available Elements

Decode (recover lost data)

Encode (calculate parity)

Send

# Typical Workflow
# Short Write (Update)




Short Write

Data stripe
Retrieve parity and modified elements

Encode

Send

# Synchronous vs Asynchronous

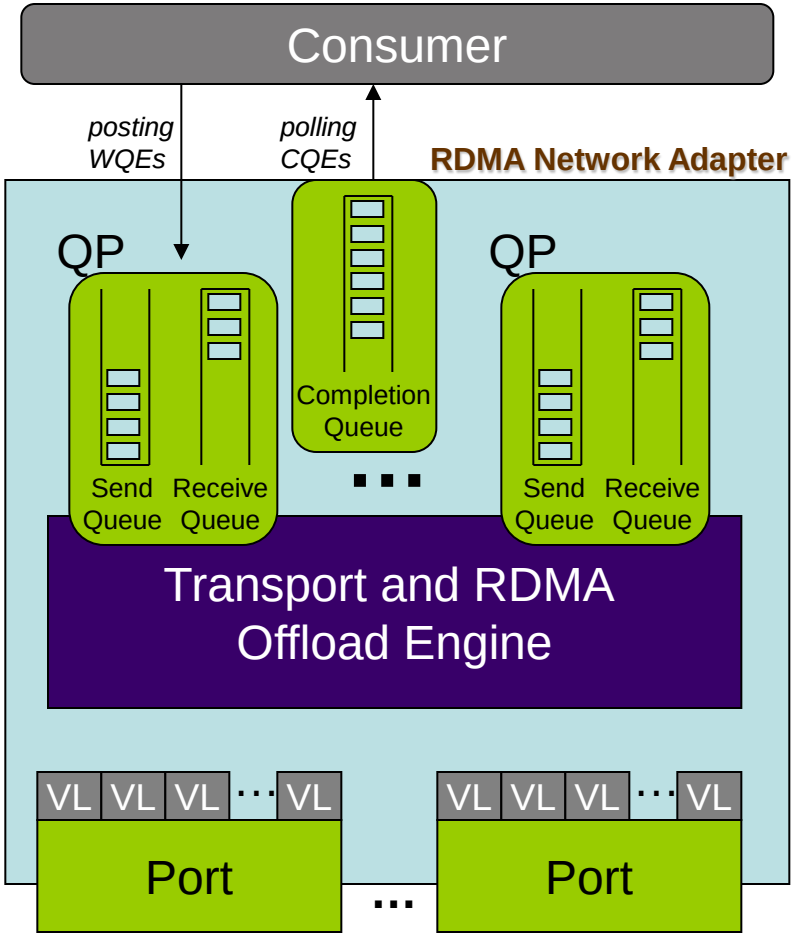
- Synchronous APIs block until operation completes
  - Suitable to the current common API semantics
  - CPU computes – runs to completion
  - Onload operation – faster ISA, but still 100% CPU utilization
- Asynchronous enables CPU to focus on computation
  - Suitable to offload semantics
  - Operation starts, completion reported upon callback
  - Can implement Synchronous using Asynchronous calls
    - Easy fit for today's integrations

# Onload vs Offload

- Onload
  - Computation all done on CPU
  - CPU at 100% during calculation
  - Cache/TLB pollution
  - Example: ISA-L
- Offload
  - Computation all done in accelerator
  - CPU at 0% during calculation
  - Cache/TLB unaffected
  - Example: ec_offload APIs

# Networking Adapter and Verbs - Briefing

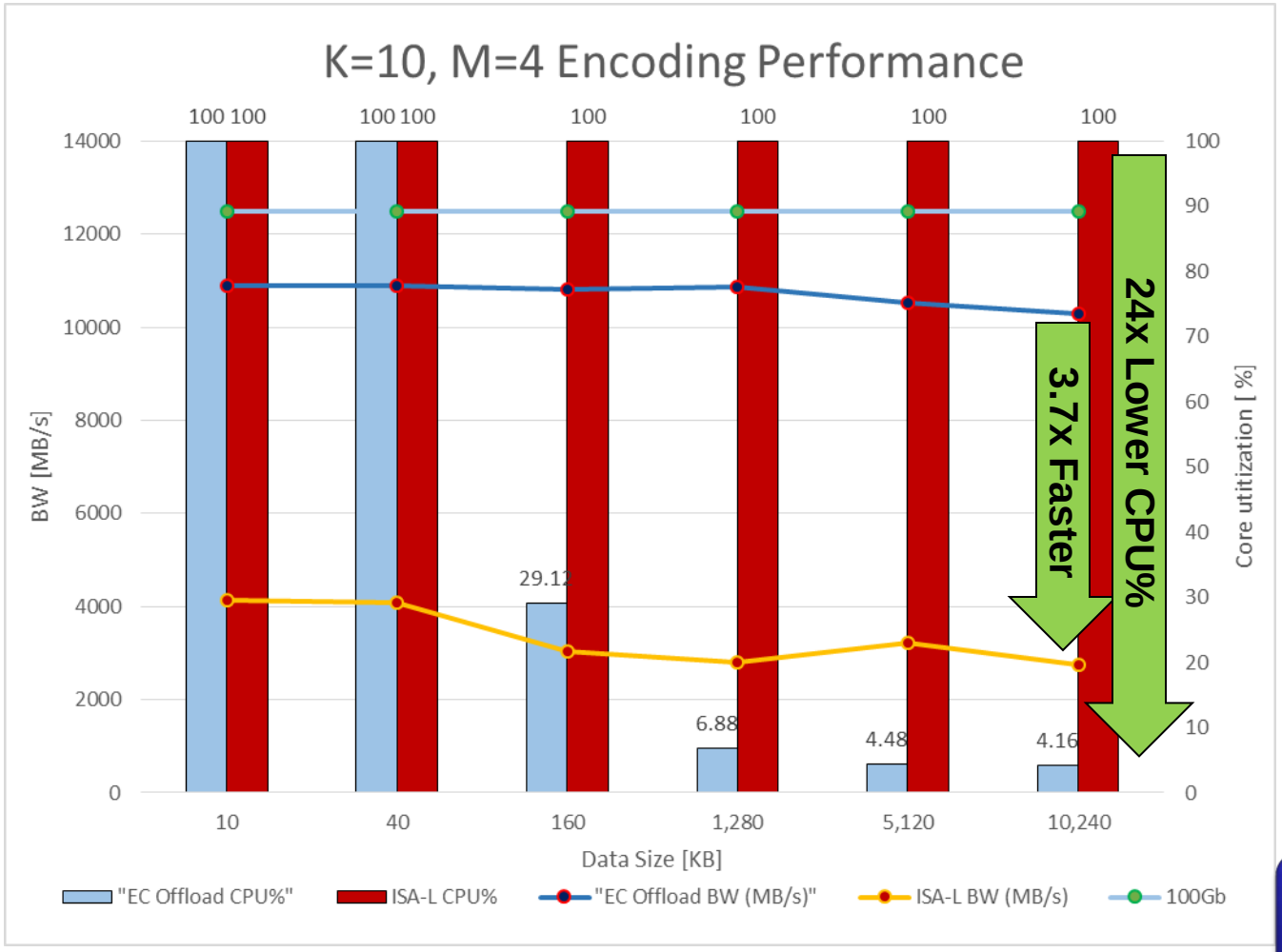
- HCA – Host Card Adapter
- Asynchronous interface
  - Consumer posts work requests
  - HCA processes
  - Consumer polls completions
- I/O channel exposed usermode apps
- Transport services
  - Reliable / Unreliable
  - Connected / Datagram
  - Send/Receive, RDMA, Atomic operations
  - **Data calculations**
- Offloading
  - Transport executed by HCA
  - Kernel bypass
  - RDMA



# EC Offload APIs Cheatsheet

| | Synchronous | Asynchronous |
|---|---|---|
| Initialization | ibv_exp_alloc_ec_calc()<br>ibv_exp_dealloc_ec_calc() | - |
| Encode | ibv_exp_ec_encode_sync() | ibv_exp_ec_encode_async() |
| Decode | ibv_exp_ec_decode_sync() | ibv_exp_ec_decode_async() |
| Update | ibv_exp_ec_update_sync() | ibv_exp_ec_update_async() |
| Encode & Send | - | ibv_exp_ec_encode_send() |
| Bookeeping | - | ibv_exp_ec_poll() |

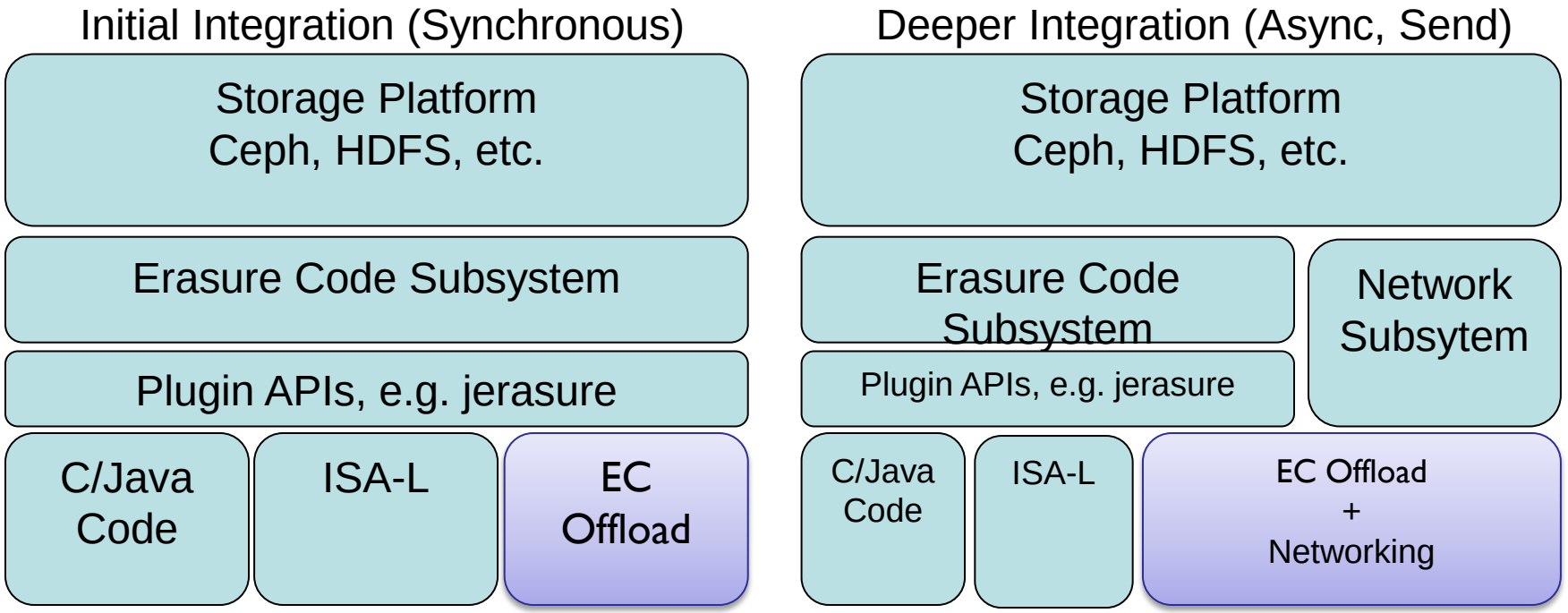
# Encoding Performance (Single Core)




Preliminary Numbers
Optimizations ongoing

# Integration into Storage Platforms

- Currently Work in Progress
  - Leverage community work



# EC Offload APIs - Current Status

- Opensource
  - BSD/GPL license
  - Supported on ConnectX-4 & ConnectX-4 LX
  - Code available today  (MLNX_OFED 3.3 and up)

- Looking at integrations into opensource and commercial applications
  - Ceph, HDFS, …

# Summary

- Software Defined Storage drives scale out storage
- Erasure Codes enable data availability at lower capacity
  - Tradeoff: CPU & Network intensive, complexity
- Erasure Codes offload offers
  - Better performance
  - Lower cost
- Library available today
  - Integration to storage systems underway
- Challenges
  - Efficient use of asynchronous acceleration
  - Combining Erasure Codes offload and networking


# Thank You !

# Attribution & Feedback

The SNIA Education Committee thanks the following Individuals for their contributions to this Tutorial.

## Authorship History

**09/2016 Dror Goldenberg**

## Additional Contributors

**Joseph L White**
**Marty Foltyn**

*Please send any questions or comments regarding this SNIA Tutorial to* ***tracktutorials@snia.org***

# Resources – Networking & Erasure Codes

Erasure codes
http://web.eecs.utk.edu/~plank/plank/papers/FAST-2005.pdf
Swift Object Storage: Adding Erasure Codes - SNIA Tutorial
http://www.snia.org/sites/default/files/Luse_Kevin_SNIATutorialSwift_Object_Storage2014_final.pdf

Linux RDMA mailing list
http://www.mail-archive.com/linux-rdma@vger.kernel.org/

InfiniBand® Trade Association http://ww[illegible]bandta.org/home

# API - Details

ibv_exp_alloc_ec_calc(*pd, *attr)
ibv_exp_dealloc_ec_calc(*calc)

ibv_exp_ec_encode_async(*calc, *ec_mem, *ec_comp)
ibv_exp_ec_encode_sync(*calc, *ec_mem)
ibv_exp_ec_decode_async(*calc, *ec_mem, *erasures, *decode_matrix, *ec_comp)
ibv_exp_ec_decode_sync(*calc,*ec_mem,*erasures,*decode_matrix)
ibv_exp_ec_update_async(*calc,*ec_mem,*data_updates,*code_updates,*ec_comp)
ibv_exp_ec_update_sync(*calc,*ec_mem,*data_updates,*code_updates)

ibv_exp_ec_poll(*calc,  n)
ibv_exp_ec_encode_send(*calc,*ec_mem,*data_stripes,*code_stripes)

# API - Initialization

```
struct ibv_exp_ec_calc_init_attr {
        uint32_t        comp_mask;
        uint32_t        max_inflight_calcs;
        int                     k;
        int                     m;
        int                     w;
        int                     max_data_sge;
        int                     max_code_sge;
        uint8_t                 *encode_matrix;
        int                     affinity_hint;
        int                     polling;
};
```