2021 OFA Virtual Workshop

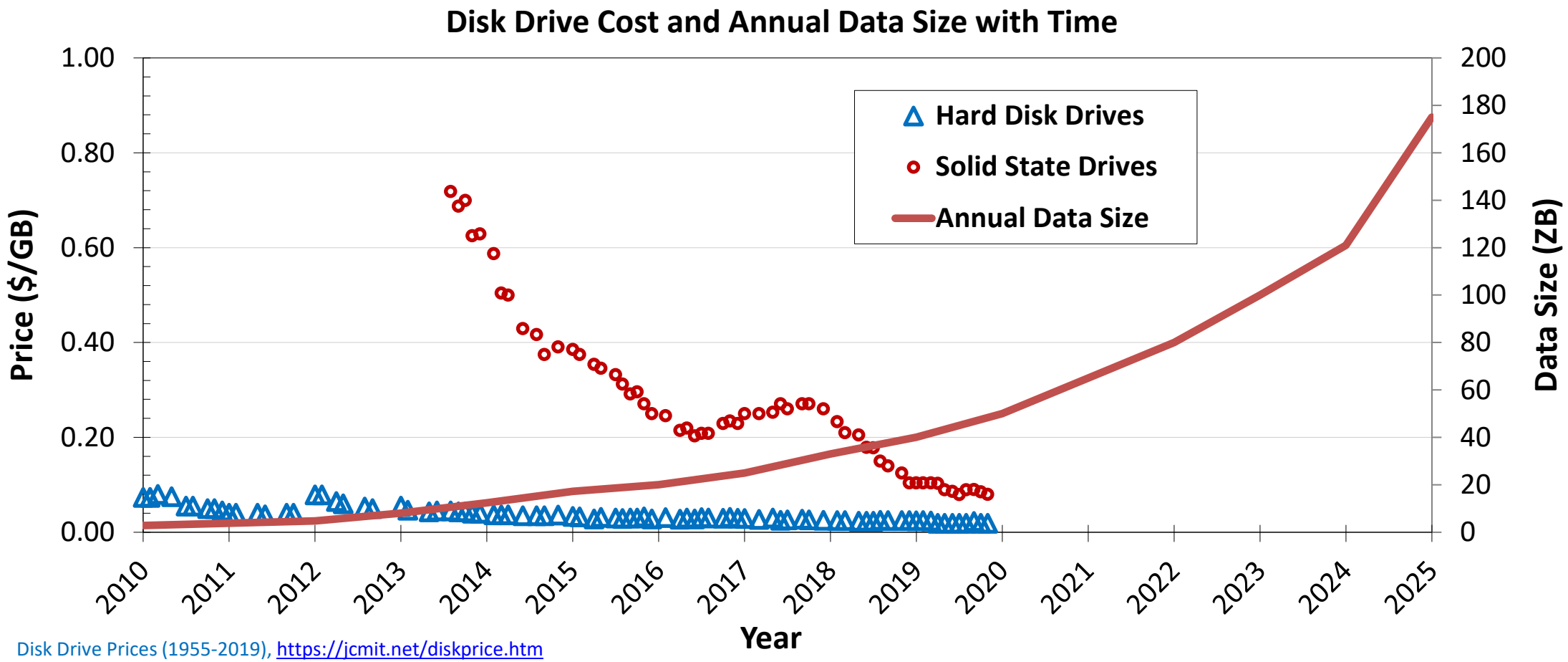
INEC: IN-NETWORK ERASURE CODING

Xiaoyi Lu

University of California, Merced

xiaoyi.lu@ucmerced.edu
http://faculty.ucmerced.edu/luxi

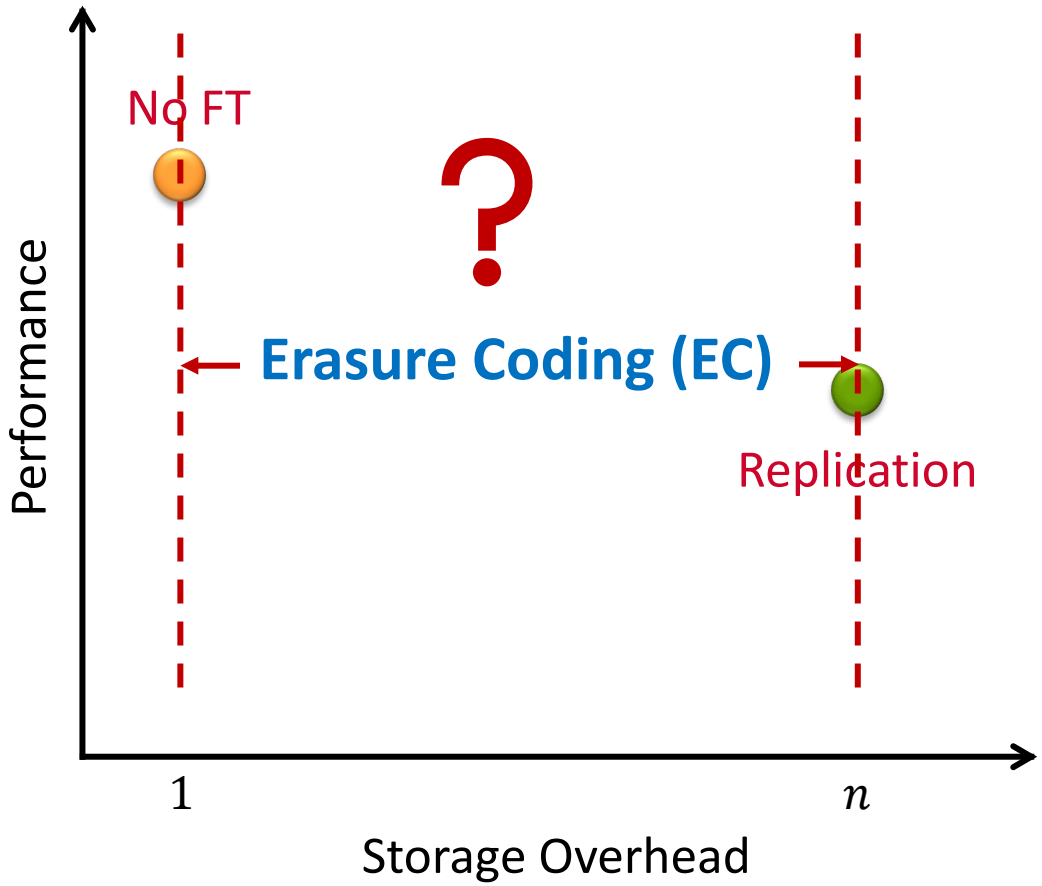
CHALLENGES OF DATA EXPLOSION



Disk Drive Prices (1955-2019), https://jcmit.net/diskprice.htm
Data Age 2025, sponsored by Seagate with data from IDC Global DataSphere, Nov 2018

RETHINKING OF DISTRIBUTED STORAGE SYSTEMS

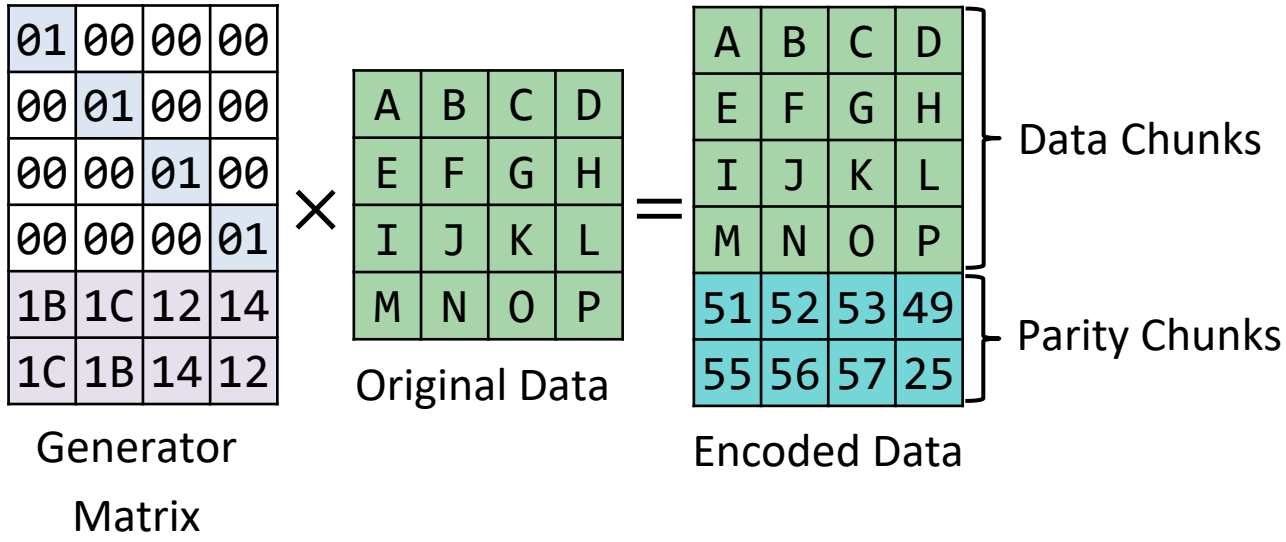
- **No Fault Tolerance**
  - ❖ High performance
  - ❖ Not reliable
- **N-way Replication**
  - Tolerate up to $n-1$ node failures
  - Performance degradation
  - Large storage overhead



EC BASICS - REED-SOLOMON CODE

REED-SOLOMON (RS) CODE

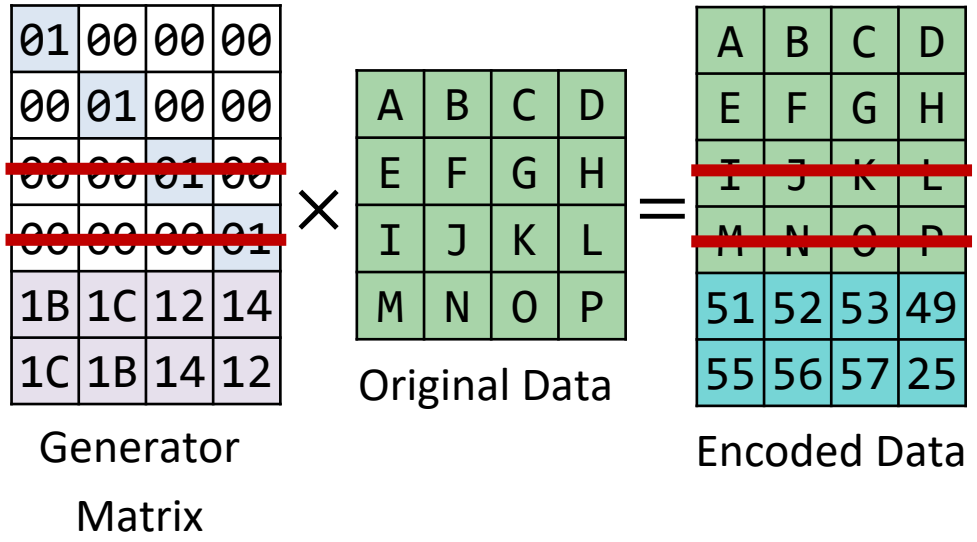
- $RS(k, m)$
  - Widely used (RAID, Microsoft Azure, HDFS 3.x)
  - Encodes on k data chunks to generate m parity chunks
  - Decodes on any k data/parity chunks to recover the original data
- *Storage Overhead of Common Codes*
  - RS(4,2) => 1.5x overhead
  - RS(6,3) => 1.5x overhead



Reed-Solomon EC for k=4 and m=2

REED-SOLOMON (RS) CODE

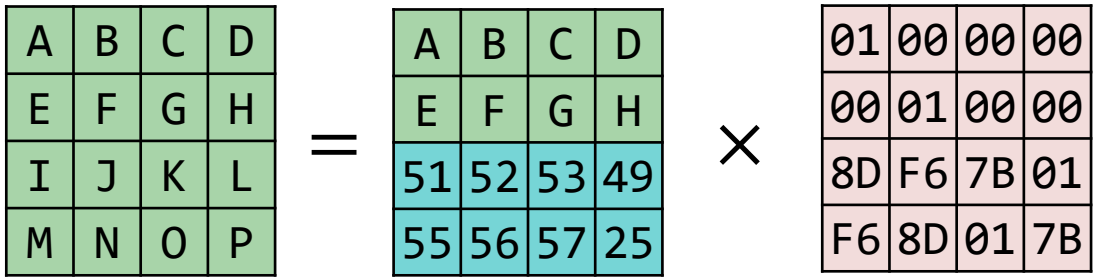
- $RS(k, m)$
  - Widely used (RAID, Microsoft Azure, HDFS 3.x)
  - Encodes on k data chunks to generate m parity chunks
  - Decodes on any k data/parity chunks to recover the original data
- *Storage Overhead of Common Codes*
  - RS(4,2) => 1.5x overhead vs. 3x in 3-way replication
  - RS(6,3) => 1.5x overhead vs. 4x in 4-way replication



Reed-Solomon EC for k=4 and m=2

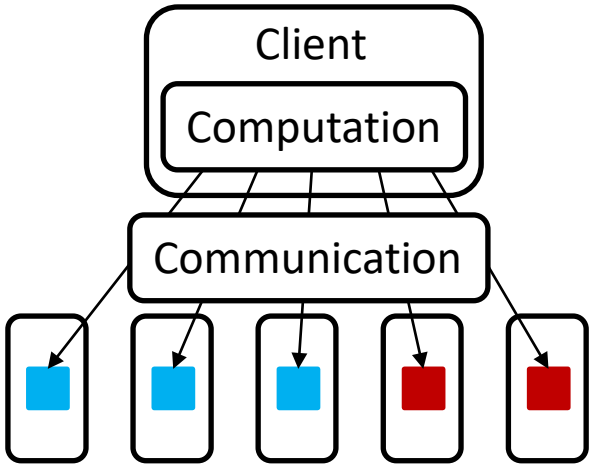
REED-SOLOMON (RS) CODE

- $RS(k, m)$
  - Widely used (RAID, Microsoft Azure, HDFS 3.x)
  - Encodes on k data chunks to generate m parity chunks
  - Decodes on any k data/parity chunks to recover the original data
- *Storage Overhead of Common Codes*
  - RS(4,2) => 1.5x overhead vs. 3x in 3-way replication
  - RS(6,3) => 1.5x overhead vs. 4x in 4-way replication

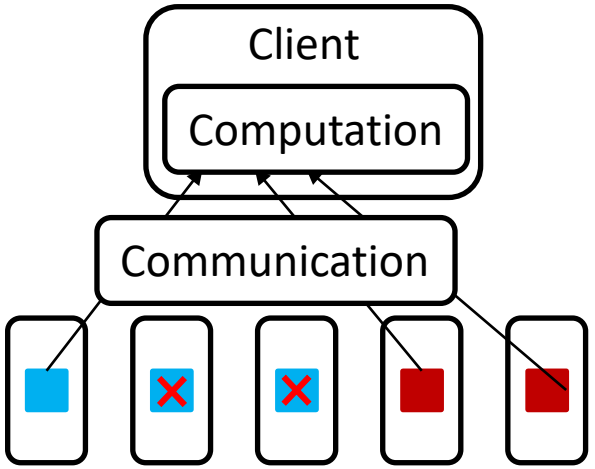


Reed-Solomon EC for k=4 and m=2

EC: THE ALTERNATIVE RESILIENCE TECHNIQUE



Write(Encoding)

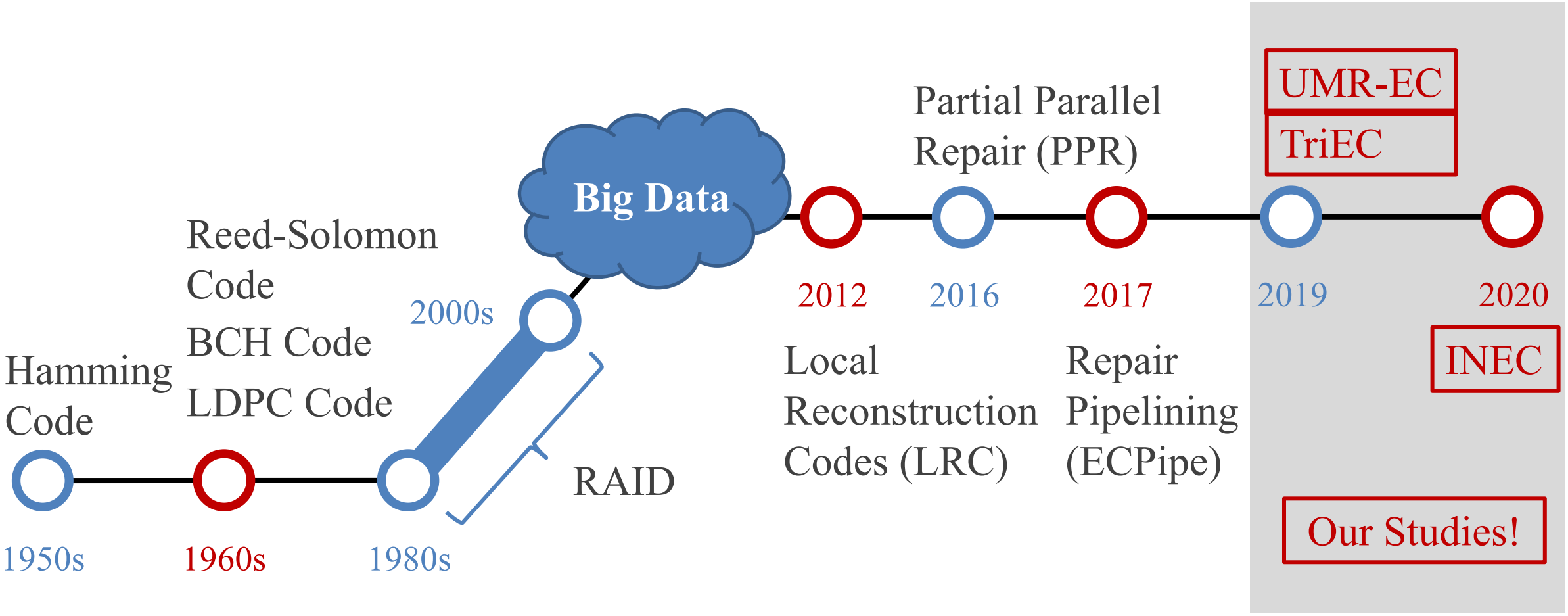


Read with Erasures (Decoding)

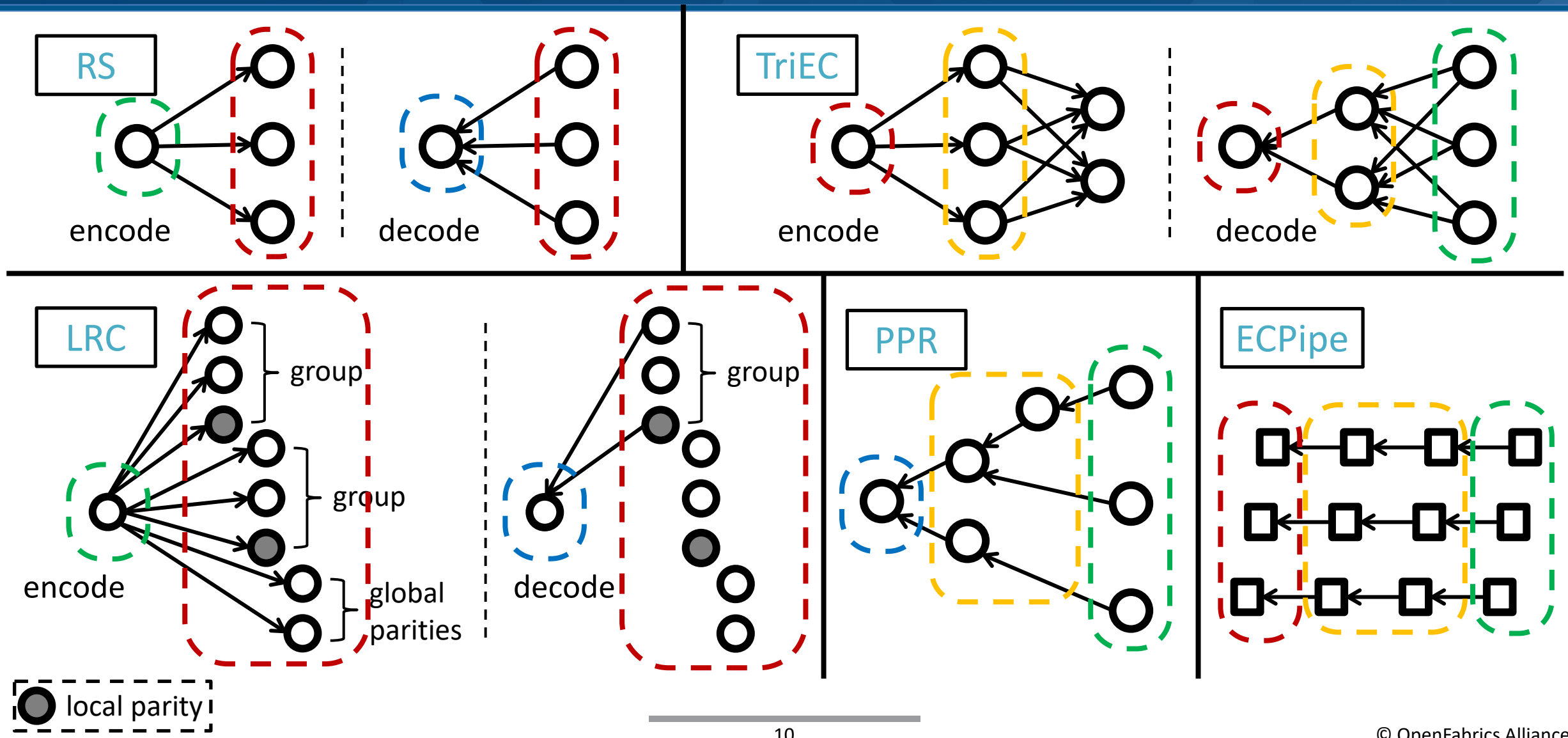
- **Exploiting EC will introduce both computation overhead and communication overhead**

Are there any good approaches to improve EC performance?

BRIEF HISTORY OF ERASURE CODING (EC)



STATE-OF-THE-ART EC SCHEMES



OPPORTUNITIES AND CHALLENGES

WHAT CAN WE LEARN?

- **EC calculations are tightly coupled with data transmission in these state-of-the-art EC schemes**
- **SmartNICs with EC capability are promising for designing next-generation distributed storage systems**

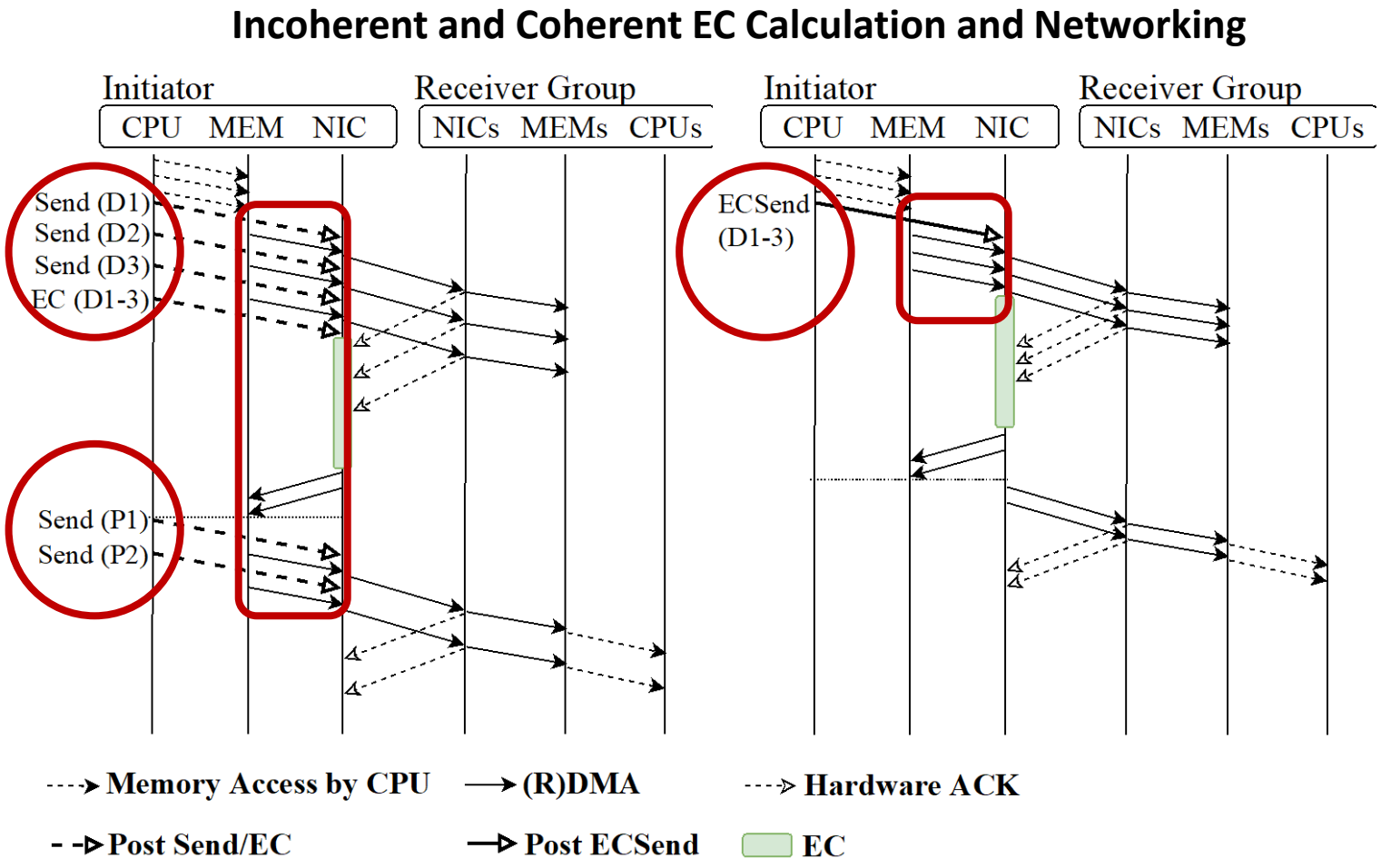
RDMA NICs


Fugaku

Summit

Google Cloud Microsoft Azure Alibaba Cloud

OVERVIEW EC CAPABILITIES ON MODERN SMARTNICS



- **Coherent EC Calculation and Networking**
  - Less CPU involvement
  - Less DMA operations

CHALLENGES

- **What should be the Coherent In-Network EC primitives to effectively support existing state-of-the-art EC schemes?**
- **How can we make EC calculations and networking functionality coherent for all the proposed Coherent In-Network EC primitives?**
- **How can we analyze and verify the effectiveness and efficiency of the proposed EC primitive and protocol designs?**

INEC: COHERENT IN-NETWORK EC PRIMITIVES

INEC APIS

- Three types of essential coherent in-network EC primitives
- Flexible APIs
  - calculator
  - memory layout
  - communication channels
  - #receives to wait for

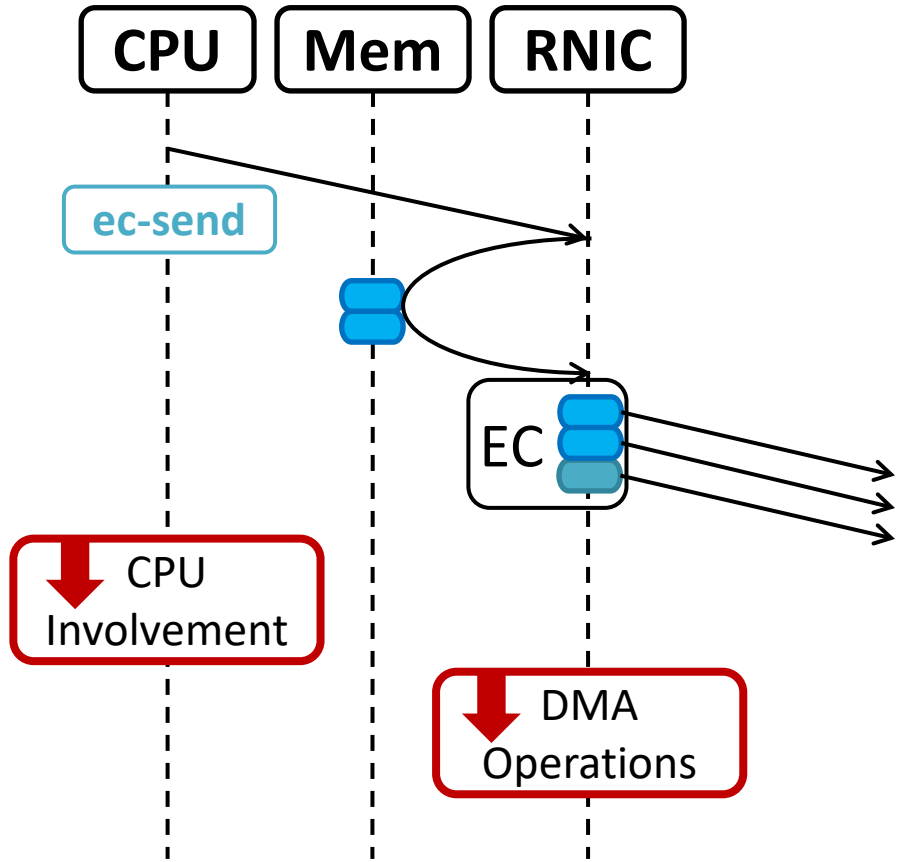
```
int inec_ec_send(struct inec_calc_desc *calc, struct
     inec_mem_desc *mem, struct inec_stripe_desc *stripe);
int inec_recv_ec_send(struct inec_wait_desc *wait, struct
     inec_calc_desc *calc, struct inec_mem_desc *mem,
     struct inec_stripe_desc *stripe);
int inec_recv_ec(struct inec_wait_desc *wait, struct
     inec_calc_desc *calc, struct inec_mem_desc *mem);
```

INEC APIs

PRIMITIVE DESIGN - EC-SEND

ec-send

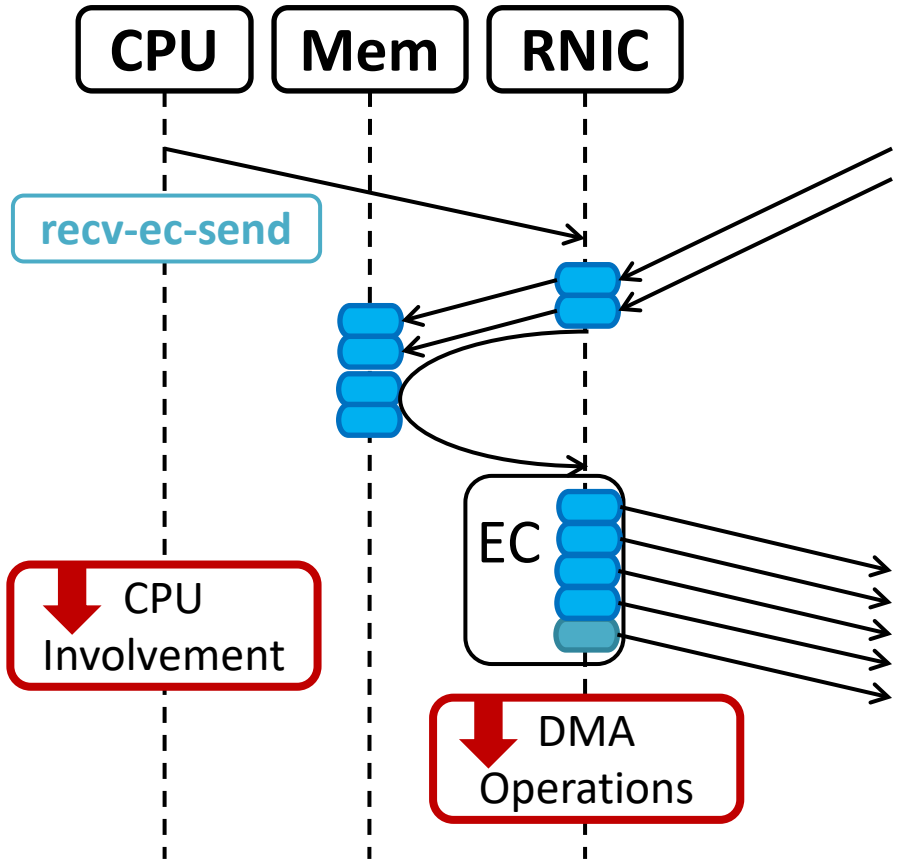
- Fetches data chunks from host memory via DMA
- Puts data chunks into on-NIC buffer
- Performs EC/XOR on data chunks
- Sends out the computation result



PRIMITIVE DESIGN - RECV-EC-SEND

recv-ec-send

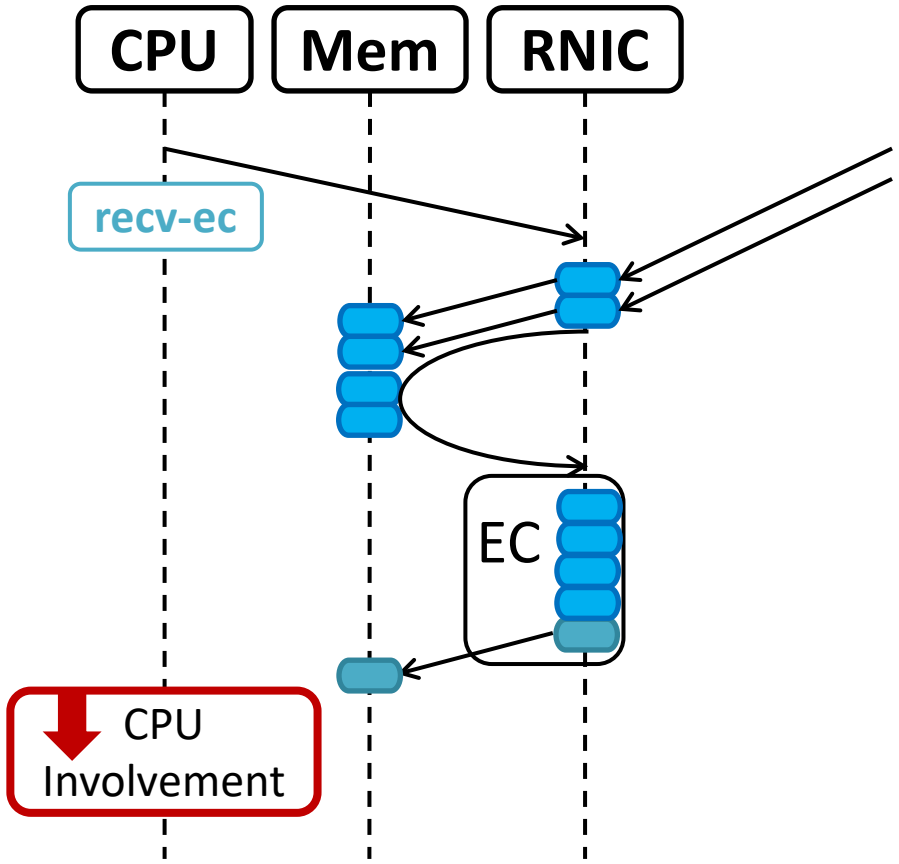
- Waits for remote chunks to arrive
- Fetches data chunks from host memory via DMA
- Puts data chunks into on-NIC buffer
- Performs EC/XOR on local chunks and received chunks
- Sends out the computation result



PRIMITIVE DESIGN - RECV-EC

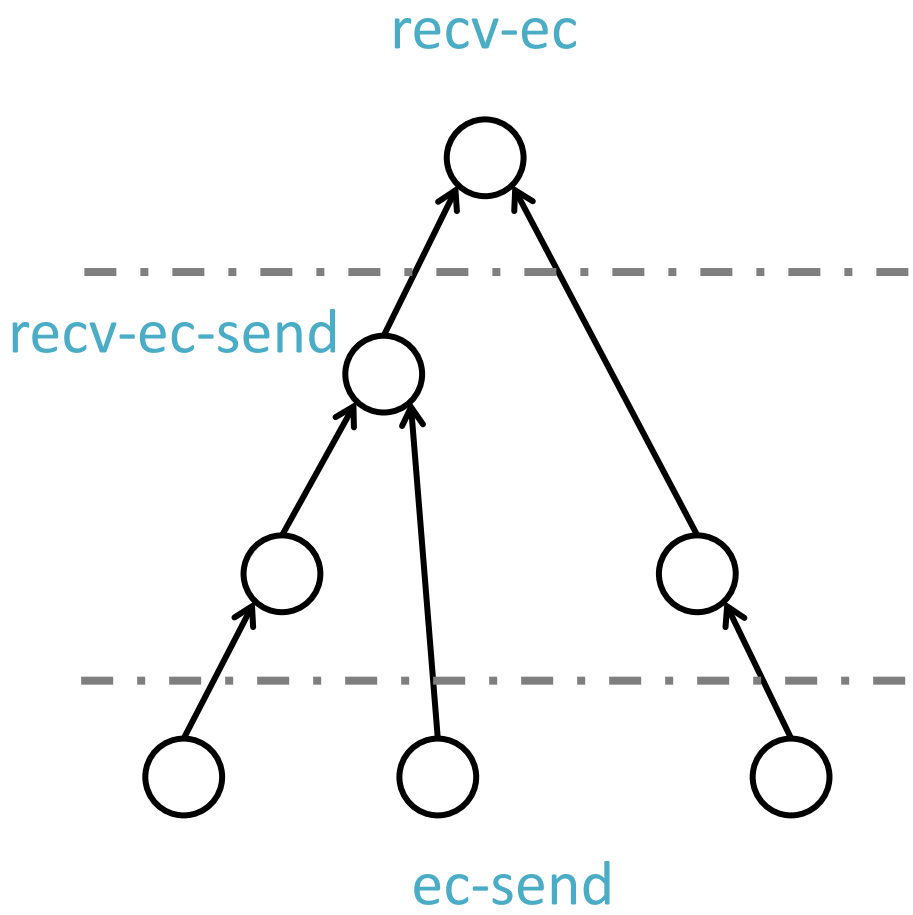
recv-ec

- Waits for remote data to arrive
- Fetches data chunks from host memory via DMA
- Puts data chunks into on-NIC buffer
- Performs EC/XOR on its local data and the received data
- Puts result into host memory



DYNAMIC EC GRAPH (DEG)

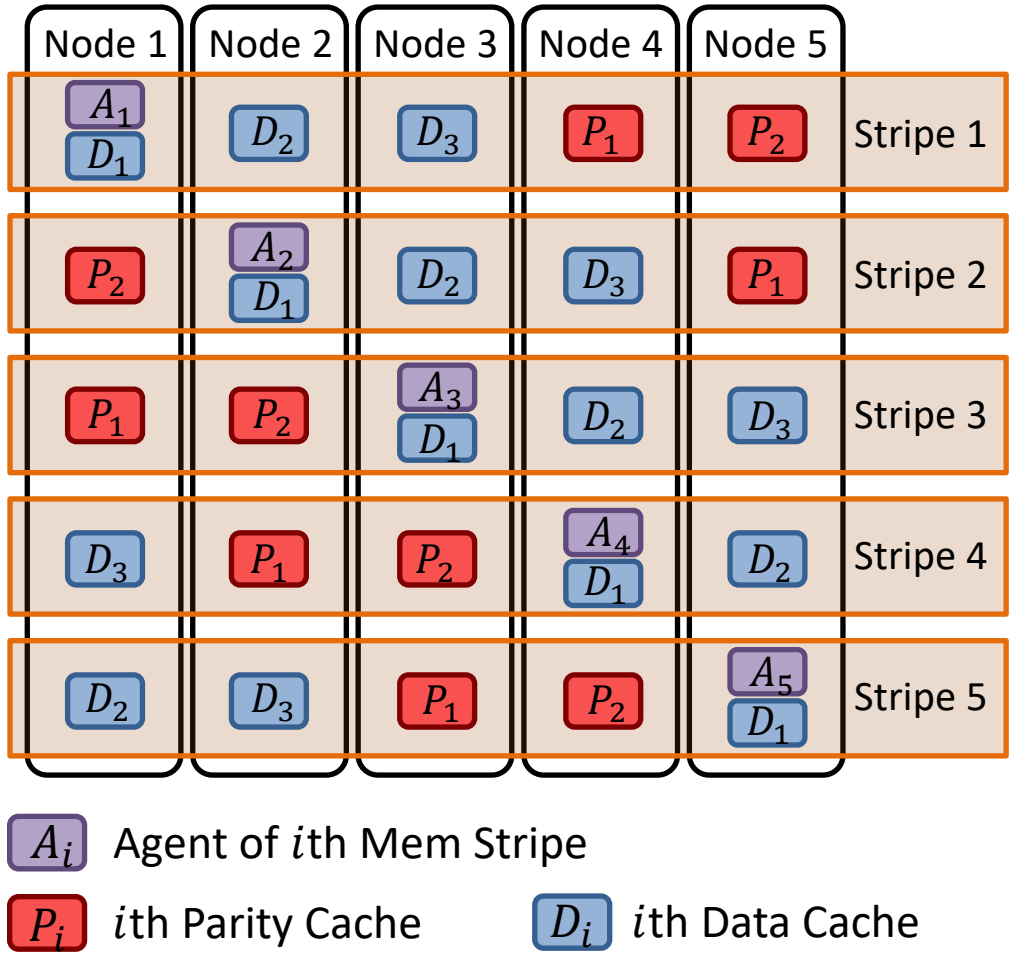
- **A dynamic acyclic graph for representing EC schemes' node layouts and automatically applying INEC primitives**
- **Rules:**
  - vertices with zero indegree choose ec-send (or send if there is no computation)
  - vertices with zero outdegree leverage recv-ec (or recv if there is no computation)
  - other vertices utilize recv-ec-send
  - explicitly specifying computation type for each vertex



MORE DETAILS

- **Primitive Analysis and Performance Model**
- **Implemented within Mellanox OFED driver 4.7-3.2.9.0**
- **Co-Design: INEC-Cache**
  - Based on *memcached* (v1.5.12)
  - Interleaved Architecture
    - Interleave Mem Stripes into the cluster to balance workload and resource utilizations

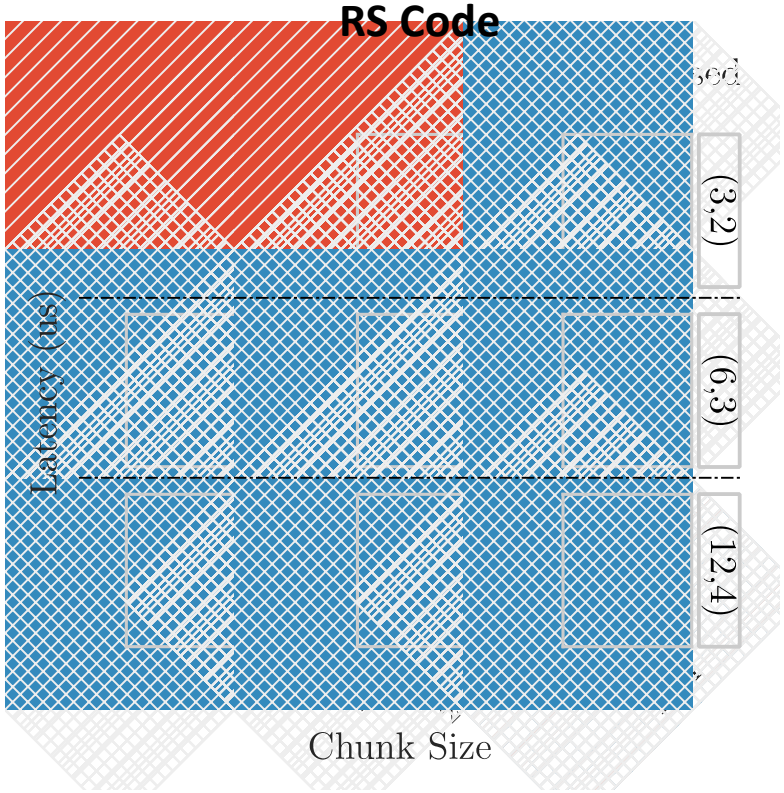
- **H. Shi, and X. Lu, *INEC: Fast and Coherent In-Network Erasure Coding*, The 32nd International Conference for High Performance Computing, Networking, Storage and Analysis (SC'20)**



EVALUATION

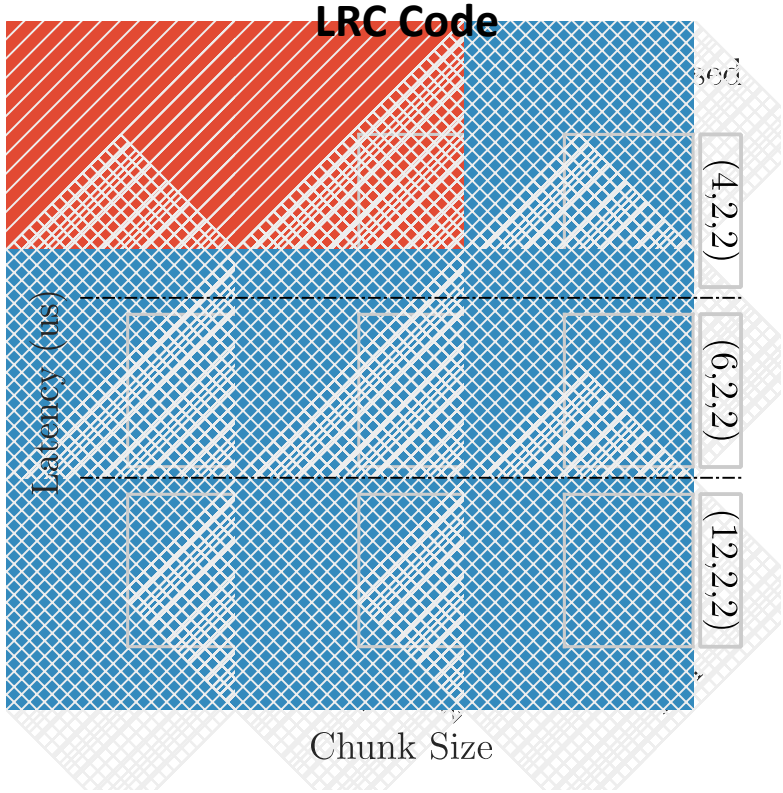
MICROBENCHMARK – ENCODING LATENCY

RS Code



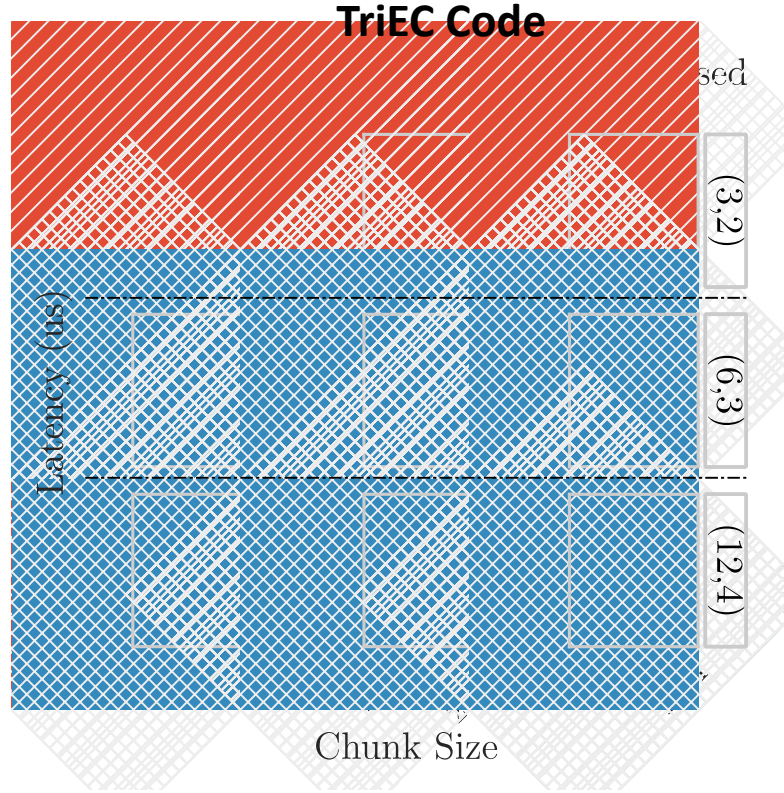
INEC improves RS by up to 2.28x

LRC Code



INEC improves LRC by up to 1.58x

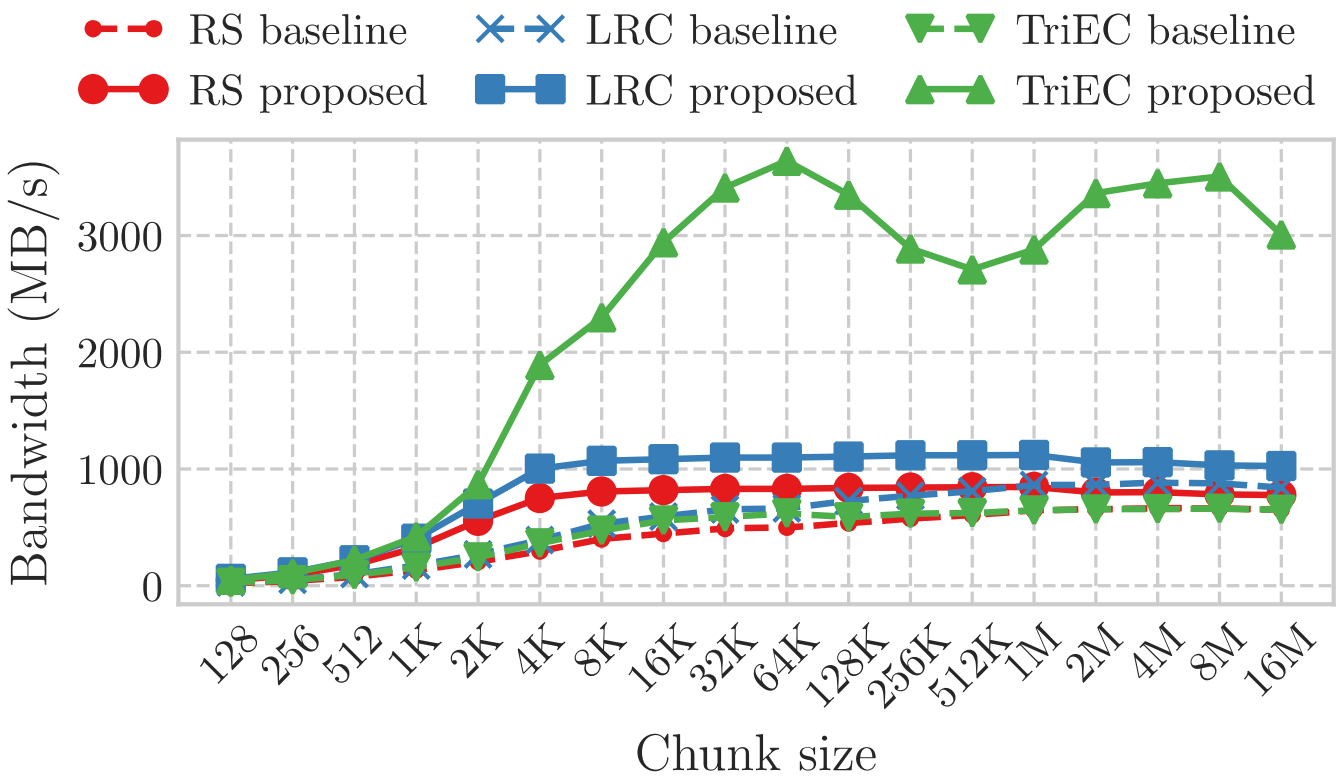
TriEC Code



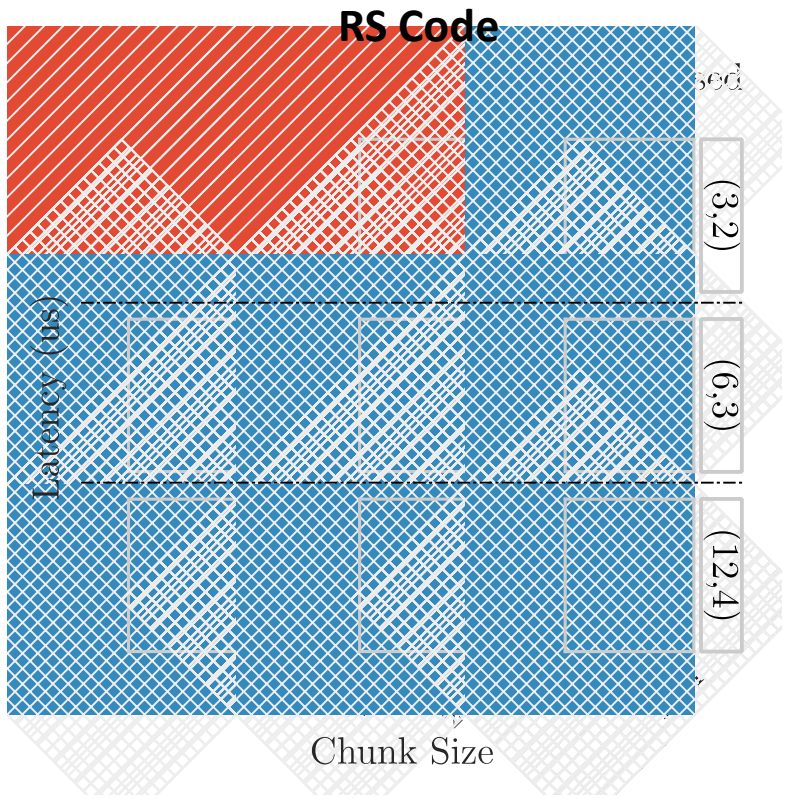
INEC improves TriEC by up to 3.36x

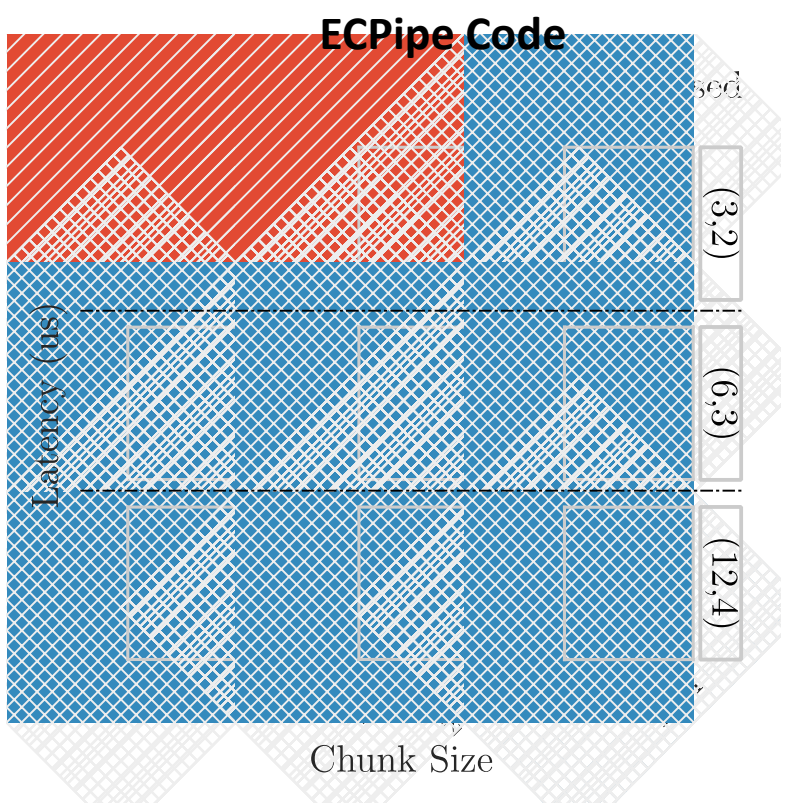
MICROBENCHMARK – ENCODING BANDWIDTH

- bandwidth = (size of generated data)/(elapsed time)
- INEC speeds up the bandwidth of RS by up to 2.71x, LRC by up to 2.63x, and TriEC by up to 5.87x
- TriEC with INEC primitives achieves the best bandwidth performance



MICROBENCHMARK – DECODING LATENCY

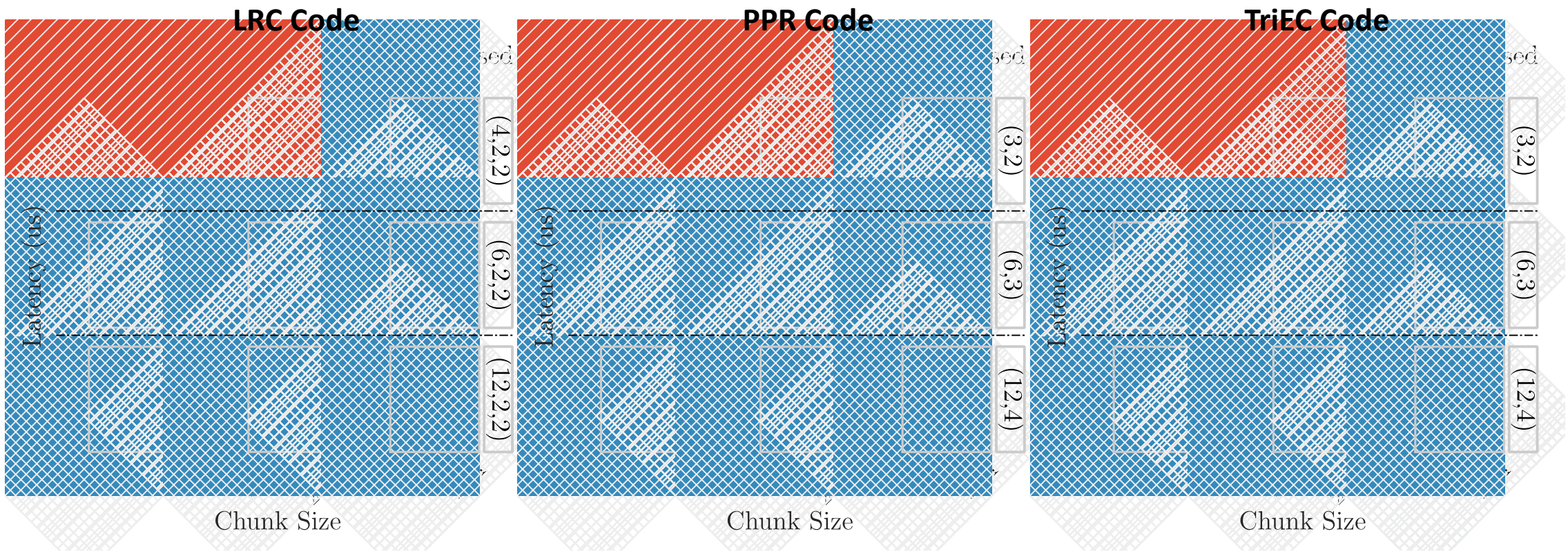




INEC reduces latencies of RS by up to 54.01%

ECPipe is accelerated by up to 72.25%

MICROBENCHMARK – DECODING LATENCY

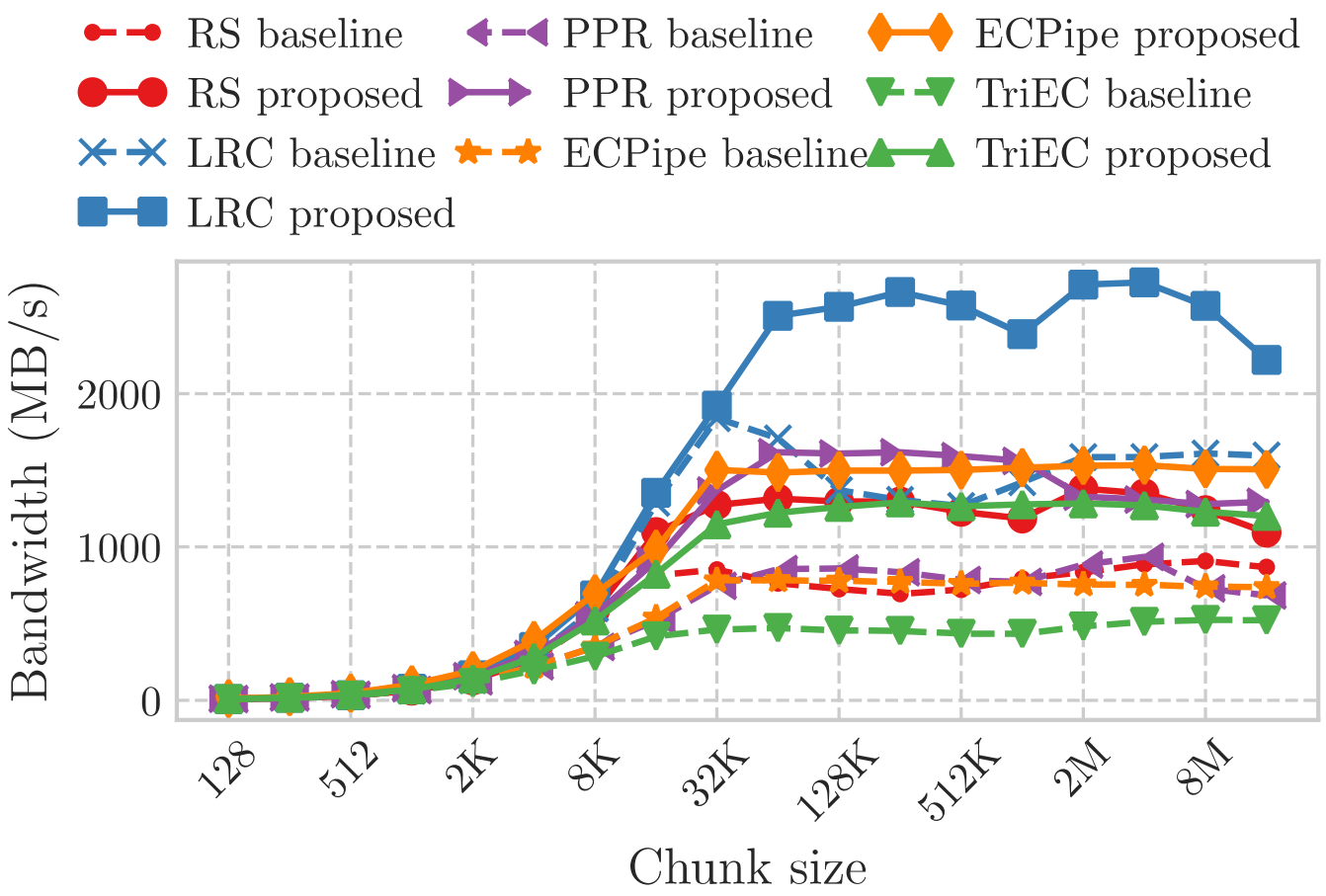


- For small chunk sizes, INEC in LRC incurs performance degradation
- INEC reduces latencies of LRC, PPR, and TriEC by up to 57.6%, 51.2%, and 64.3%, respectively

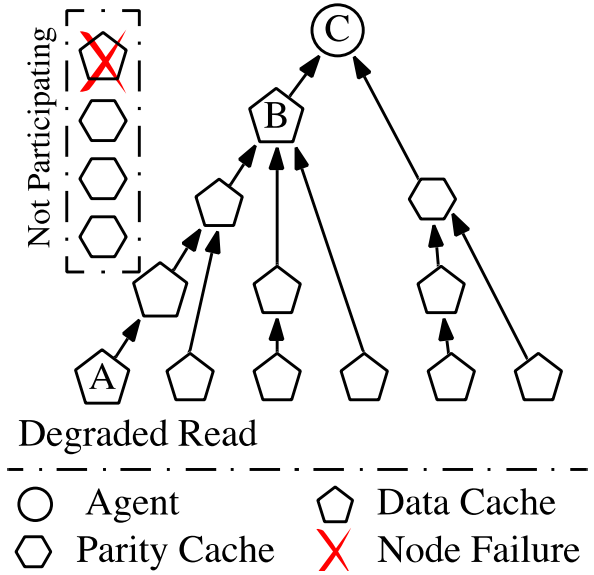
MICROBENCHMARK – DECODING BANDWIDTH

- bandwidth = (size of generated data)/(elapsed time)

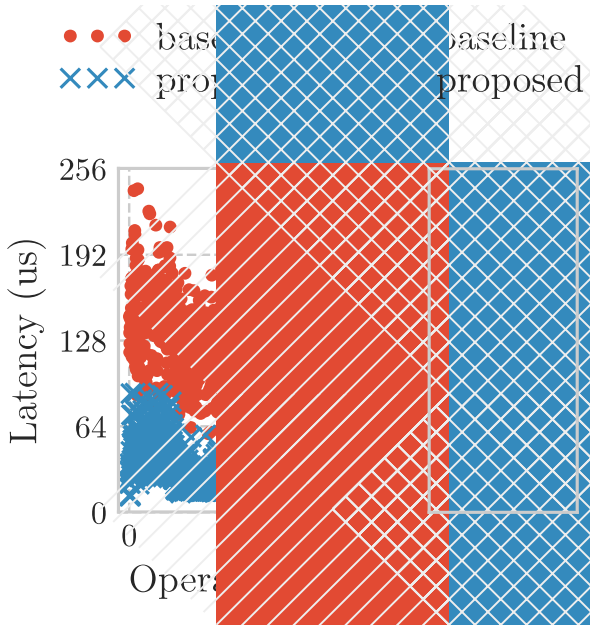
- INEC outperforms the baseline in terms of bandwidth by up to 1.86x, 2.04x, 2.03x, 2.04x, and 2.94x for RS, LRC, PPR, ECPipe, and TriEC, respectively



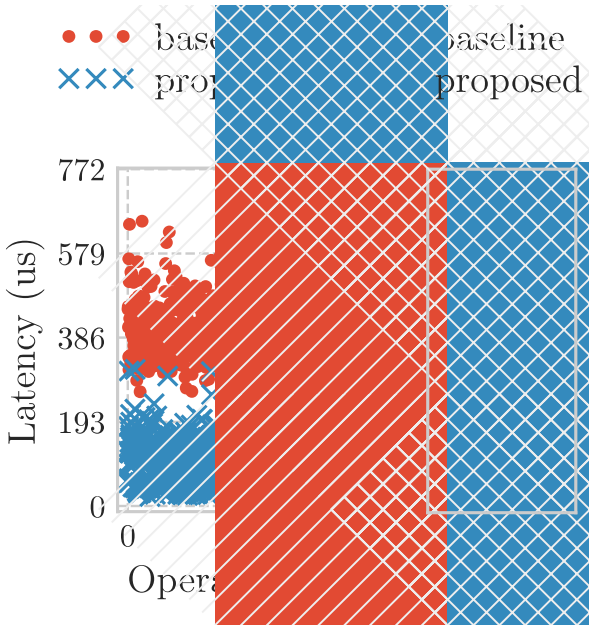
IMPACT ON PERCENTILE LATENCIES OF INEC-CACHE



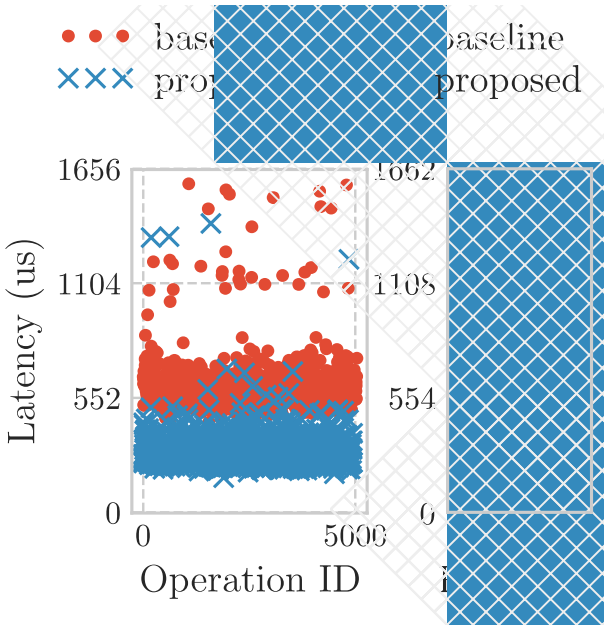
(a) PPR(12,4) Repairing DEG



(b) ec-send of Node A



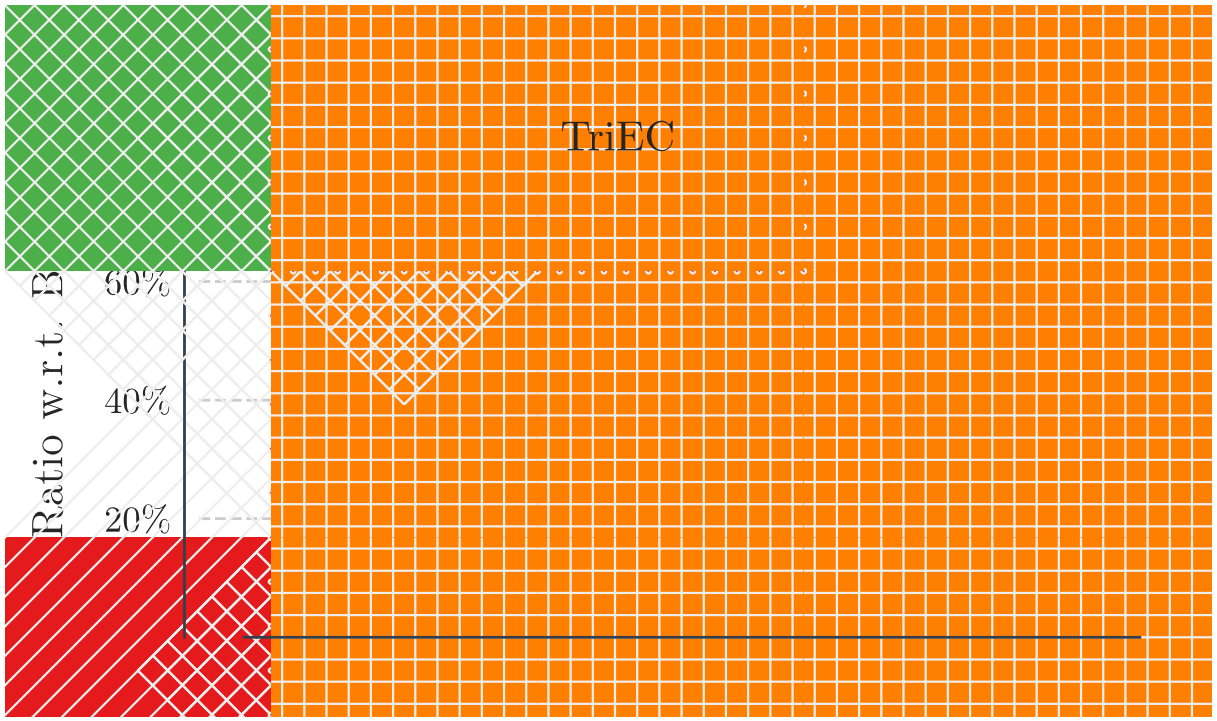
(c) recv-ec-send of Node B



(d) recv-xor of Node C

- The latency distributions demonstrate that using INEC can alleviate the performance fluctuation introduced by frequent context switches
- INEC reduces 50th, 95th, and 99th percentile latencies by up to 77.16%, 66.79%, and 62.40%, respectively

THROUGHPUT IMPROVEMENT OF INEC-CACHE



- Improves RS, LRC, PPR, ECPipe, and TriEC by up to 14.36%, 16.99%, 68.33%, 99.57%, and 92.26%, respectively
- An average performance boost of up to 57% with read-dominated workloads and an average speedup of 28% with write-heavy workloads

CONCLUSION

CONCLUSION AND IMPACT

Proposed designs in INEC are abstract and generic

- Can be used to co-design different distributed storage systems and EC schemes

Improved efficiency, parallelism, and overlapping

- Deliver performance benefits to upper-layer applications
- Make EC more viable for designing distributed storage systems

Fast and coherent in-network EC primitives (INEC) and TriEC paradigm

- Could be integrated into next-generation SmartNICs' drivers

H. Shi, and X. Lu, *INEC: Fast and Coherent In-Network Erasure Coding*, The 32nd International Conference for High Performance Computing, Networking, Storage and Analysis (SC'20)

2021 OFA Virtual Workshop

THANK YOU

Xiaoyi Lu, Assistant Professor

University of California, Merced

UNIVERSITY OF CALIFORNIA
MERCED