Offloading Erasure Coding to CSD in Hyperledger Fabric

유정현[1], 변홍수[1], 김영재[1]
이명철[2], 최진춘[2]

[1] *서강대학교 컴퓨터공학과*
[2] *한국전자통신연구원 스마트데이터연구실*

KCS'24
2024. 1. 26.

SOGANG UNIVERSITY
DISCOS Data-Intensive Computing & Systems Laboratory
ETRI 한국전자통신연구원 Electronics and Telecommunications Research Institute

Outline

- Background
- Motivation
- Approach: Mathematical Modeling
- Use Case Study: NewportCSD (NGD System)
- Concluding Remark

Outline

- **Background**
- Motivation
- Approach: Mathematical Modeling
- Use Case Study: NewportCSD (NGD System)
- Concluding Remark

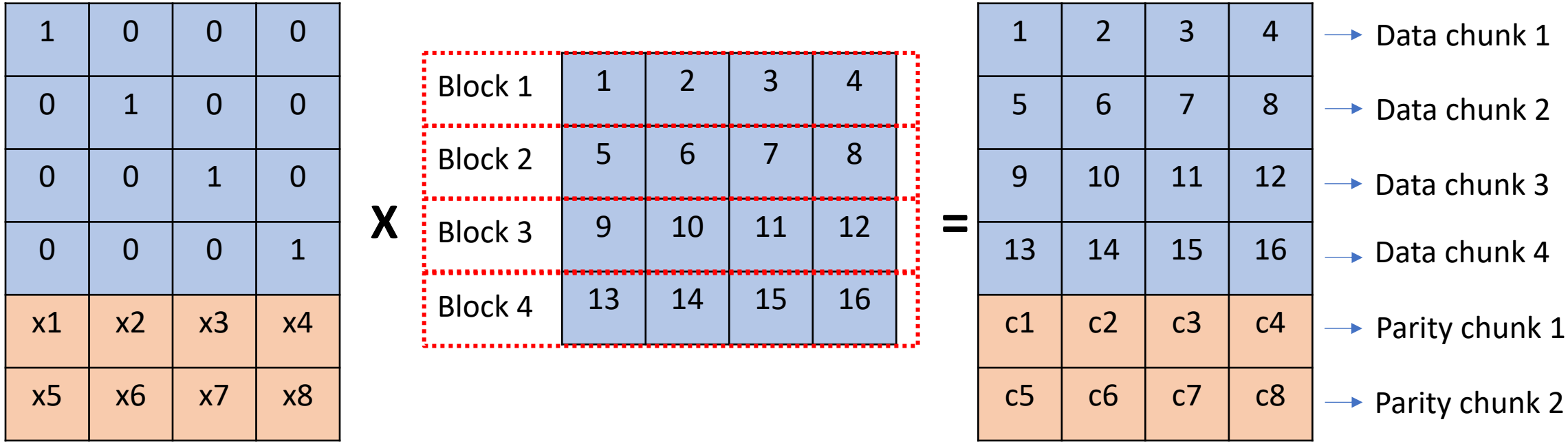
Hyperledger Fabric

- Permissioned blockchain developed by the Linux Foundation

- Applications
  - Supply chain management
  - Healthcare
  - Logistics

Erasure Coding

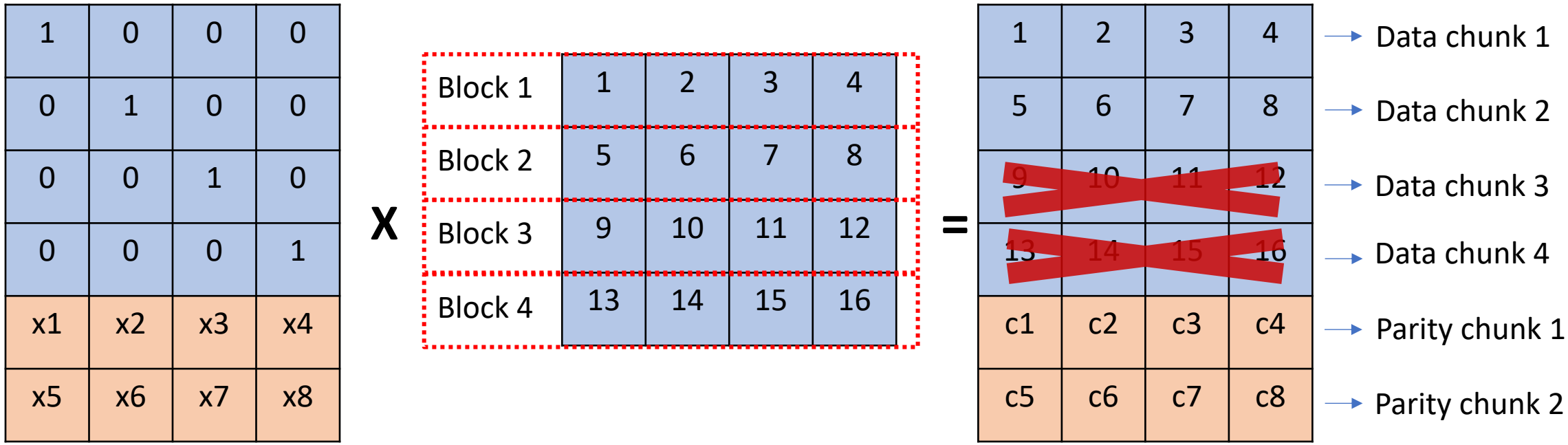
- Data storage and recovery technology for efficient distributed system
- Divide the original data into multiple chunks



< Example of (6, 4) Erasure Coding>

Erasure Coding

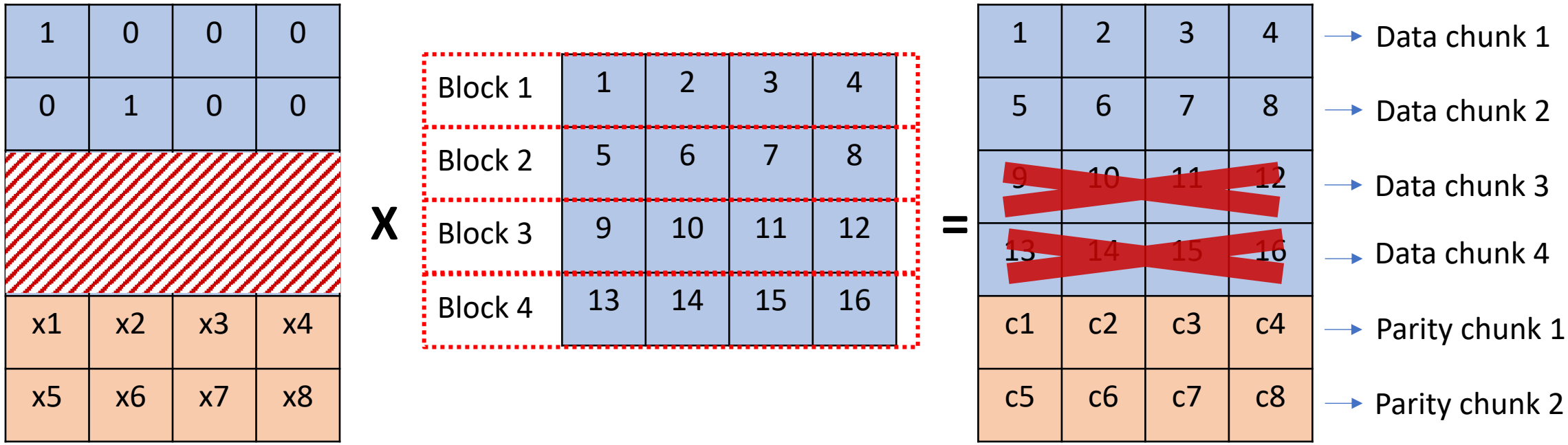
- Ensure consistency against failures up to the number of parity chunks



< Example of (6, 4) Erasure Coding>

Erasure Coding

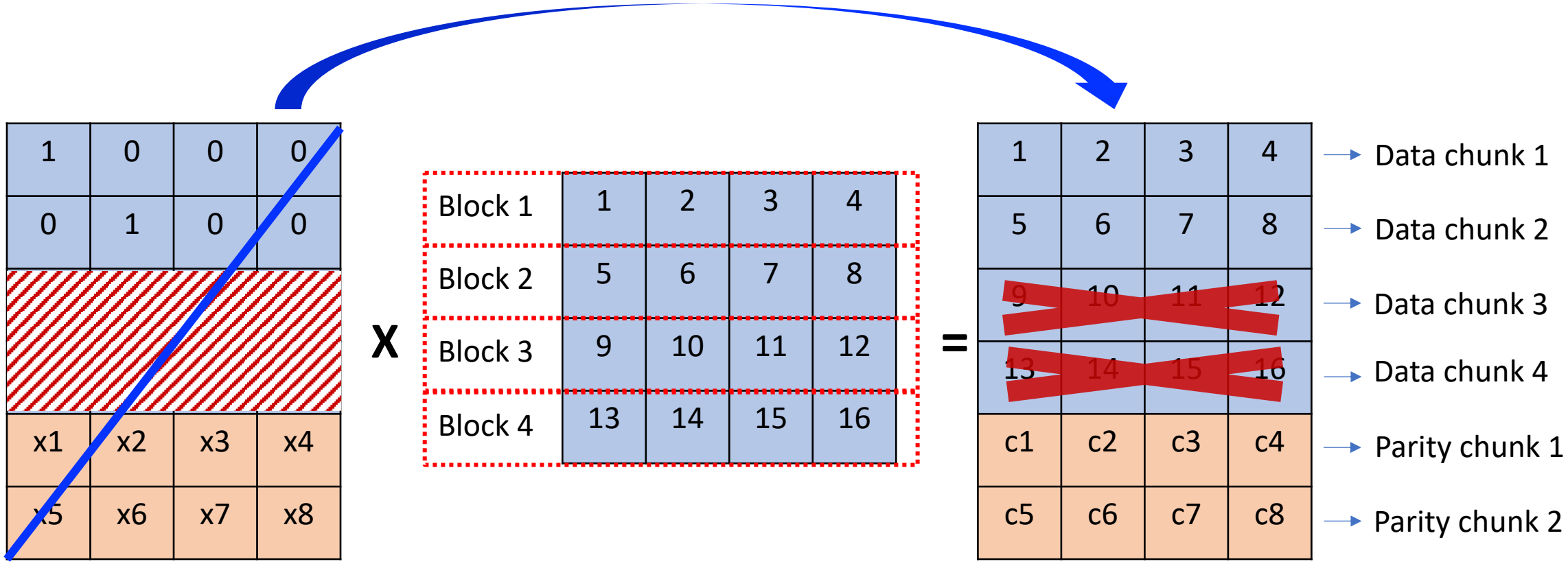
- Ensure consistency against failures up to the number of parity chunks



< Example of (6, 4) Erasure Coding>

Erasure Coding

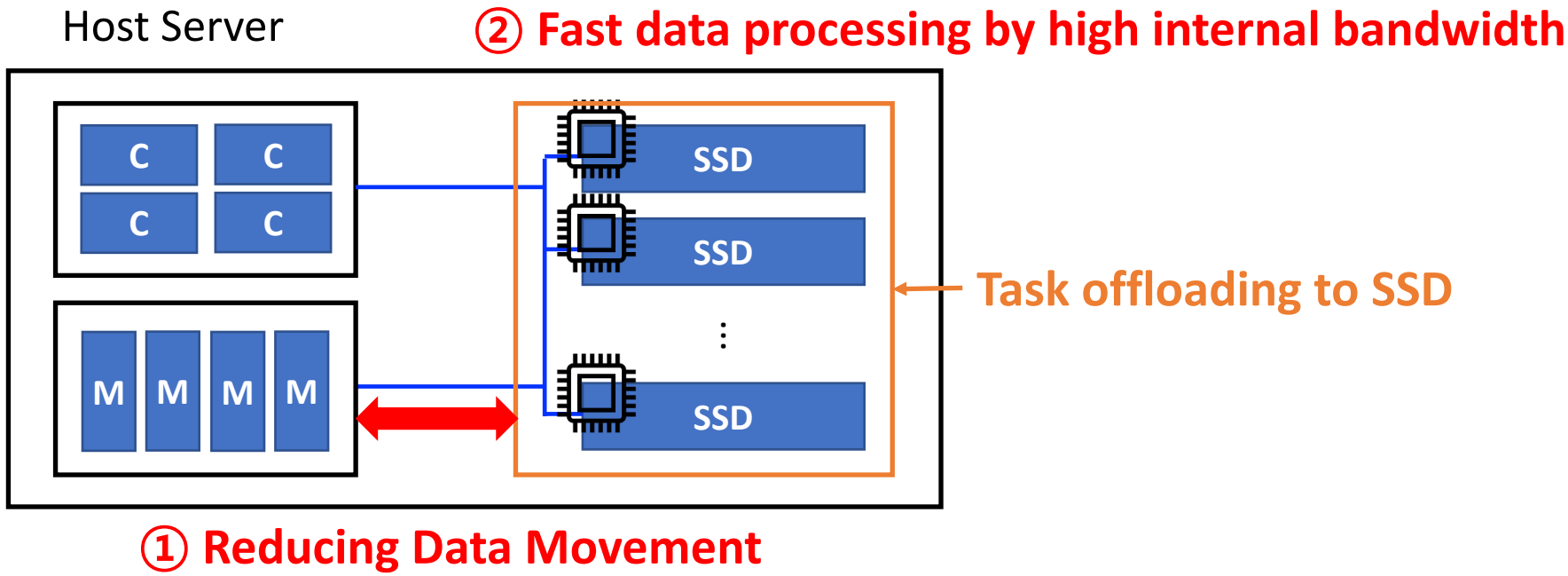
- Ensure consistency against failures up to the number of parity chunks



< Example of (6, 4) Erasure Coding>

Computational Storage Device (CSD)

- Run computational tasks inside the storage device, reducing data transfer between the host and the device

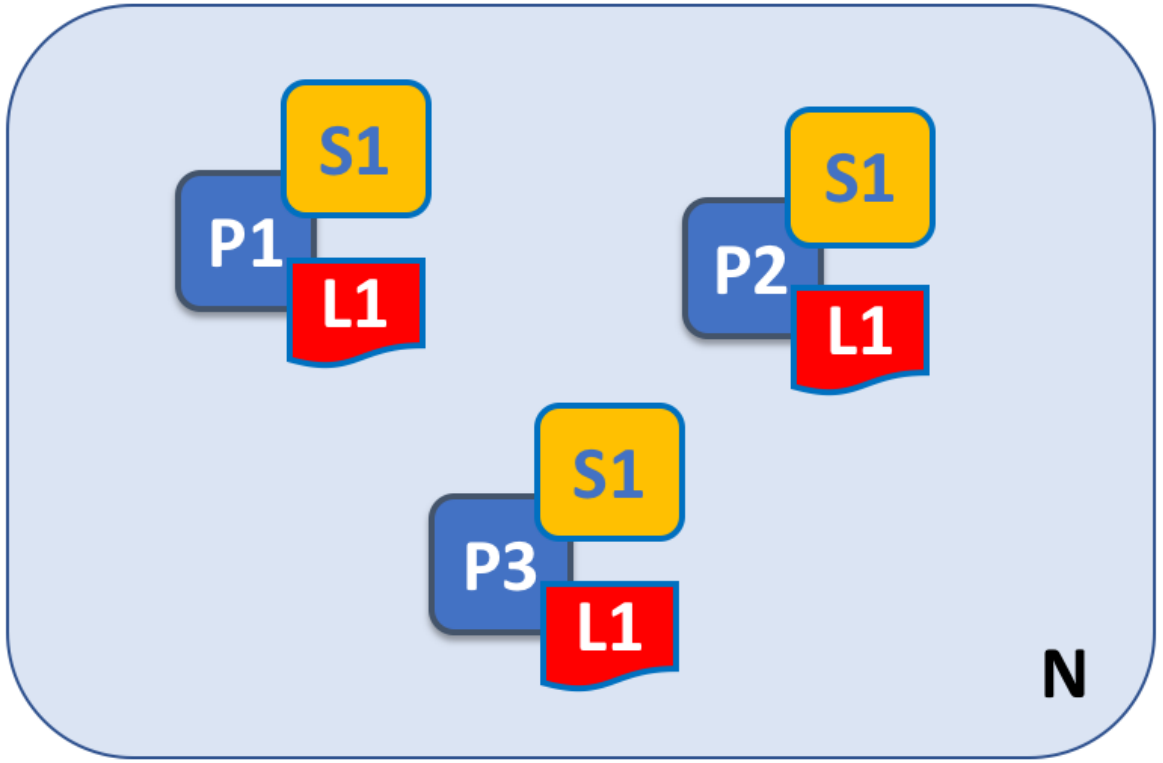


Outline

- Background
- **Motivation**
- Approach: Mathematical Modeling
- Use Case Study: NewportCSD (NGD System)
- Concluding Remark

Hyperledger Fabric

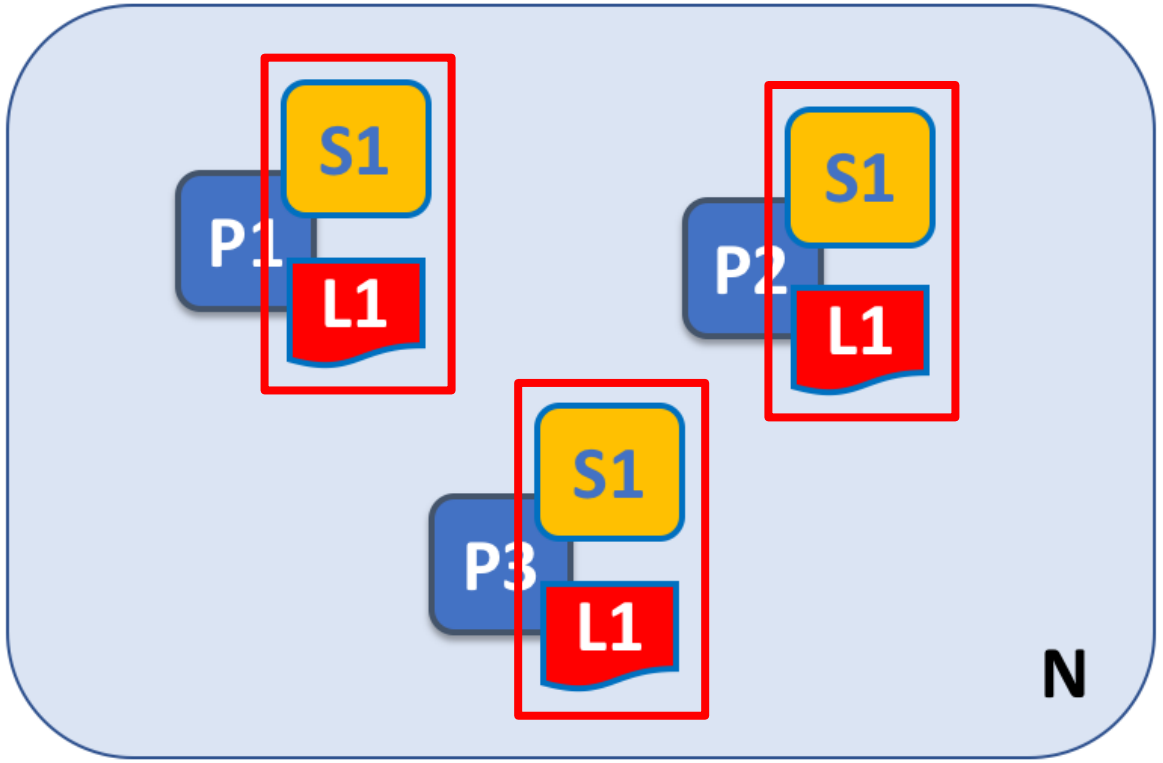
- All transactions that occur within the blockchain network are recorded in the ledger
- A copy of the original ledger must be stored redundantly across all peer node



N	Blockchain network
P	Peer node
S	Smart contract (aka chaincode)
L	Ledger

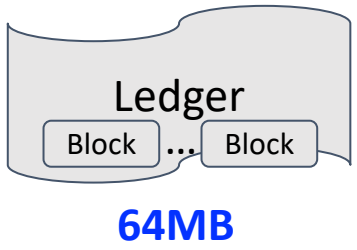
Hyperledger Fabric

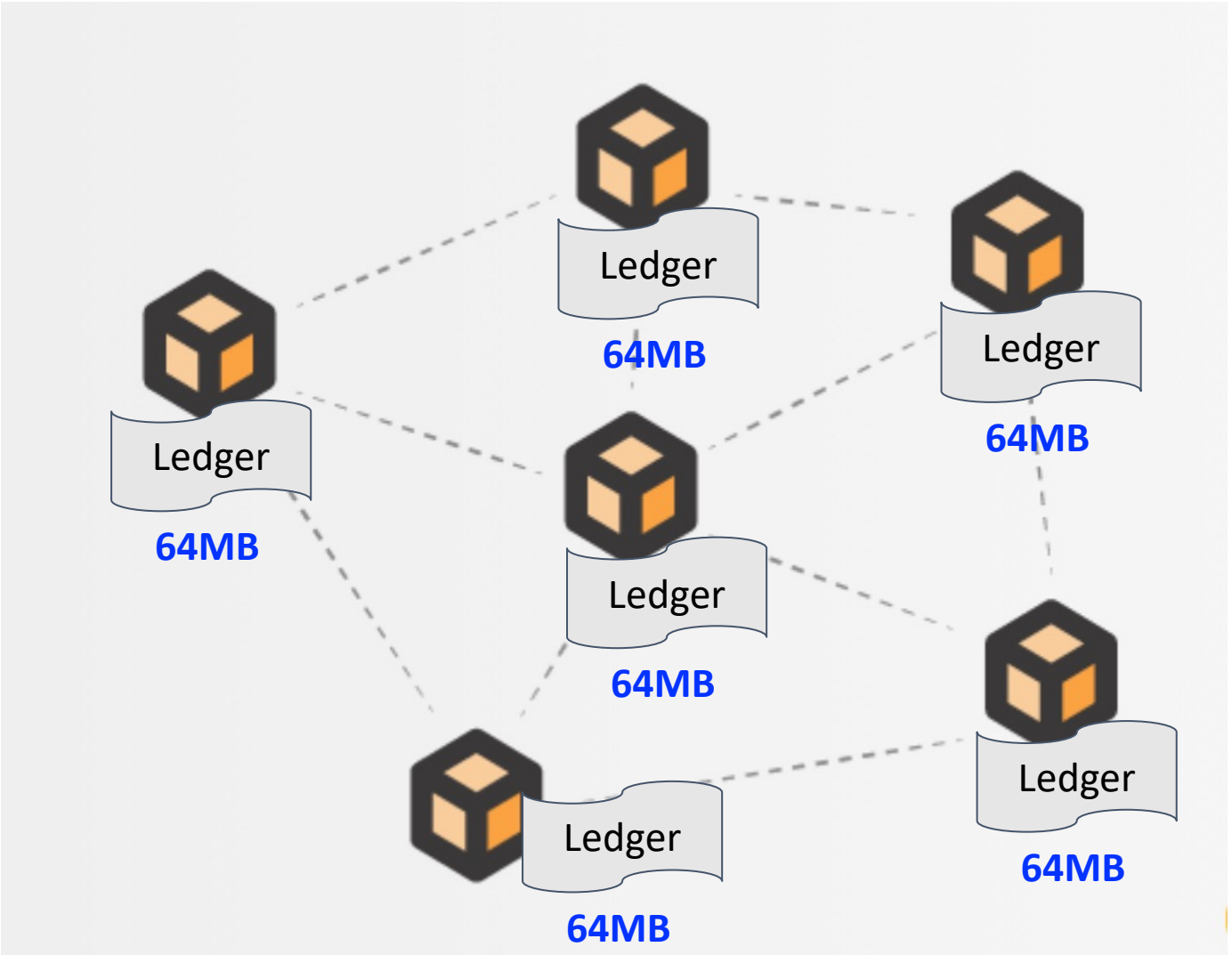
- All transactions that occur within the blockchain network are recorded in the ledger
- A copy of the original ledger must be stored redundantly across all peer node



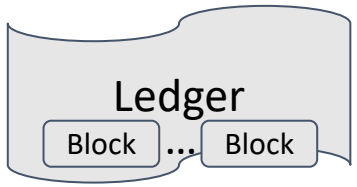
N	Blockchain network
P	Peer node
S	Smart contract (aka chaincode)
L	Ledger

Hyperledger Fabric with Erasure Coding



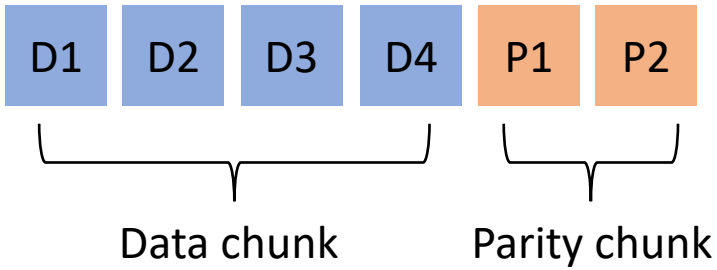


Hyperledger Fabric with Erasure Coding



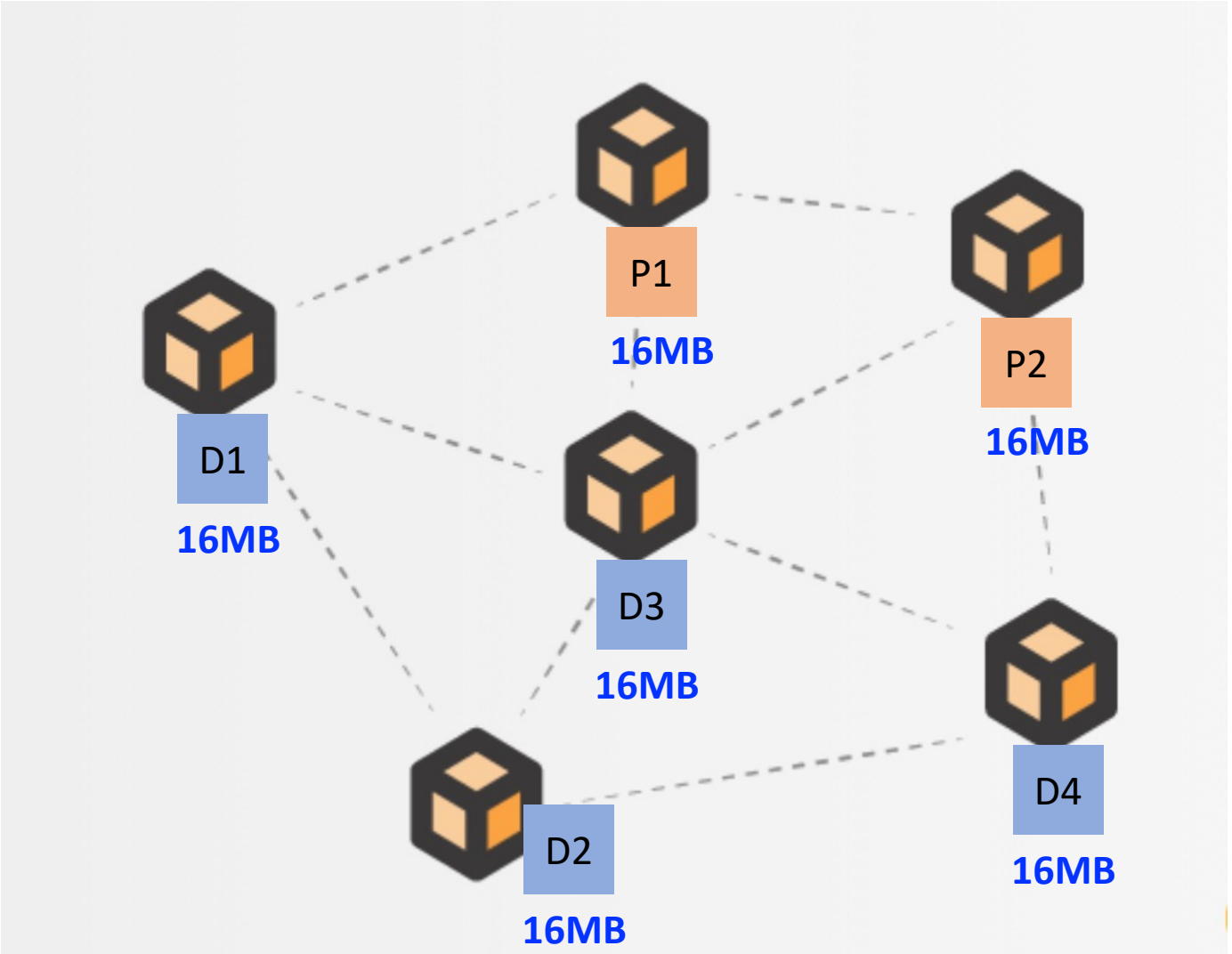
64MB

(6, 4) Erasure Coding

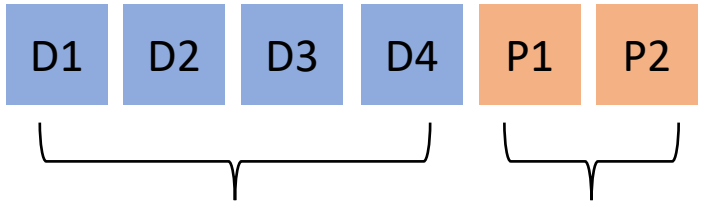


Each chunk's size = 64MB / 4 = 16MB

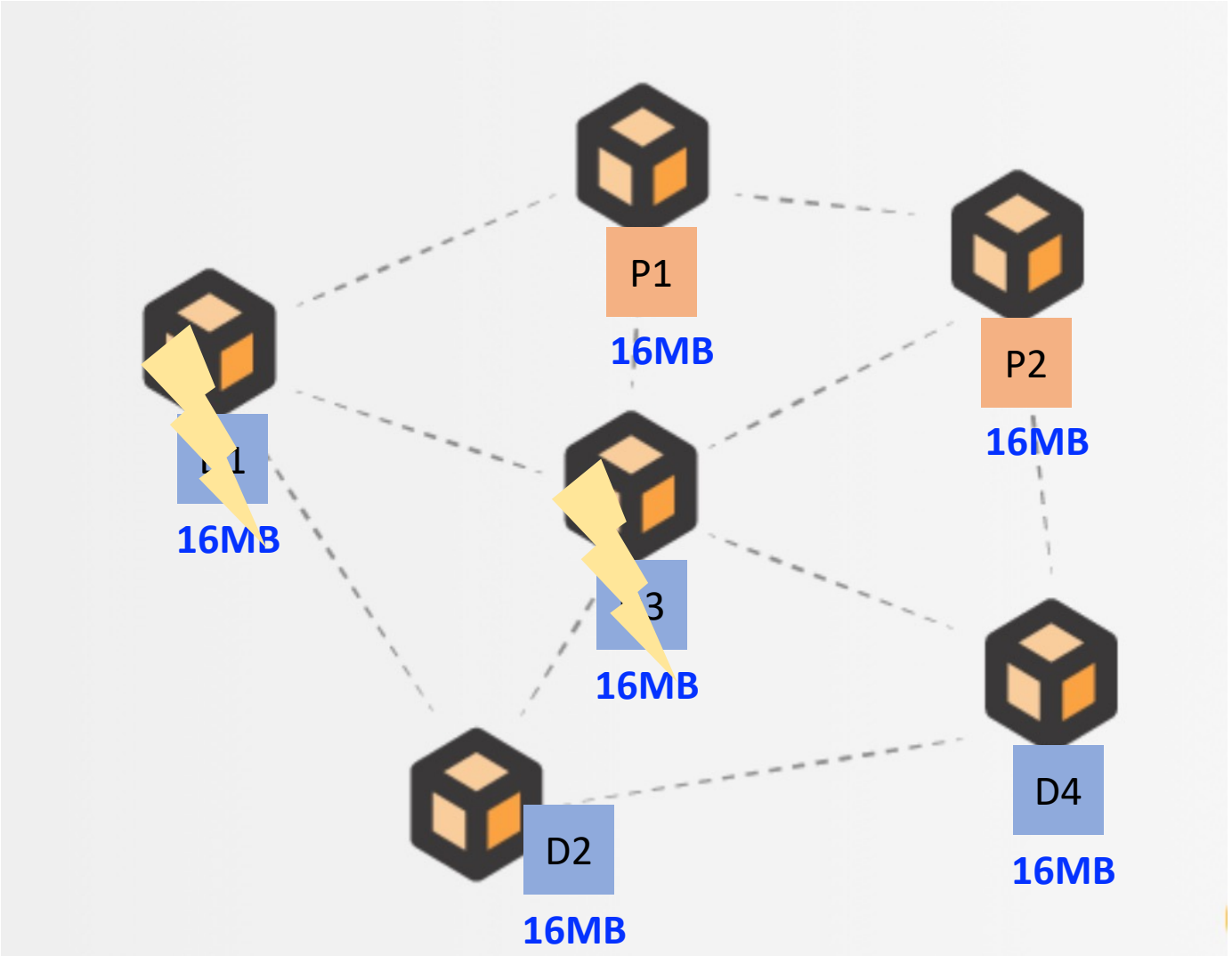
64MB → 16MB
Achieved **75%** space efficiency



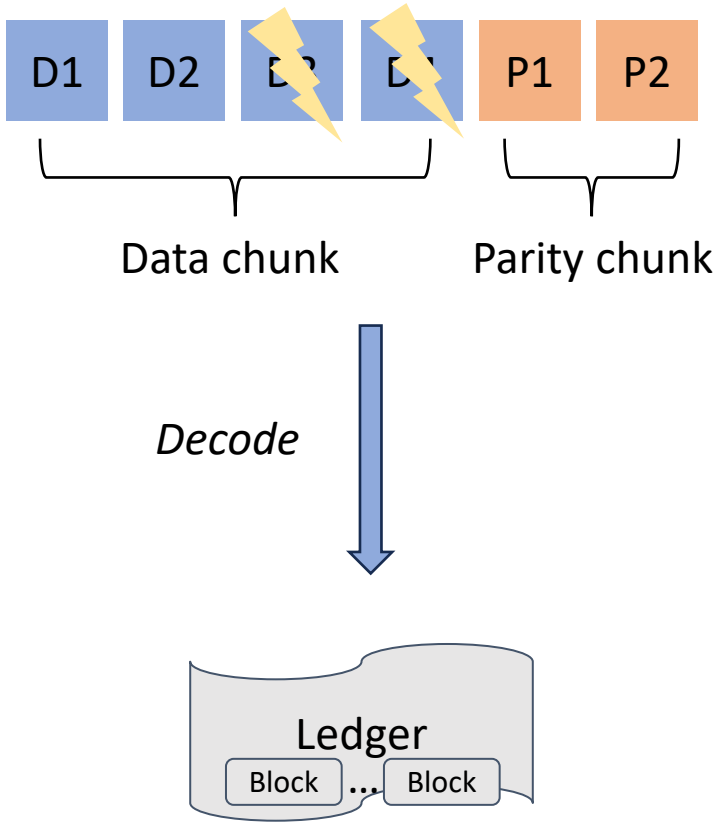
Hyperledger Fabric with Erasure Coding

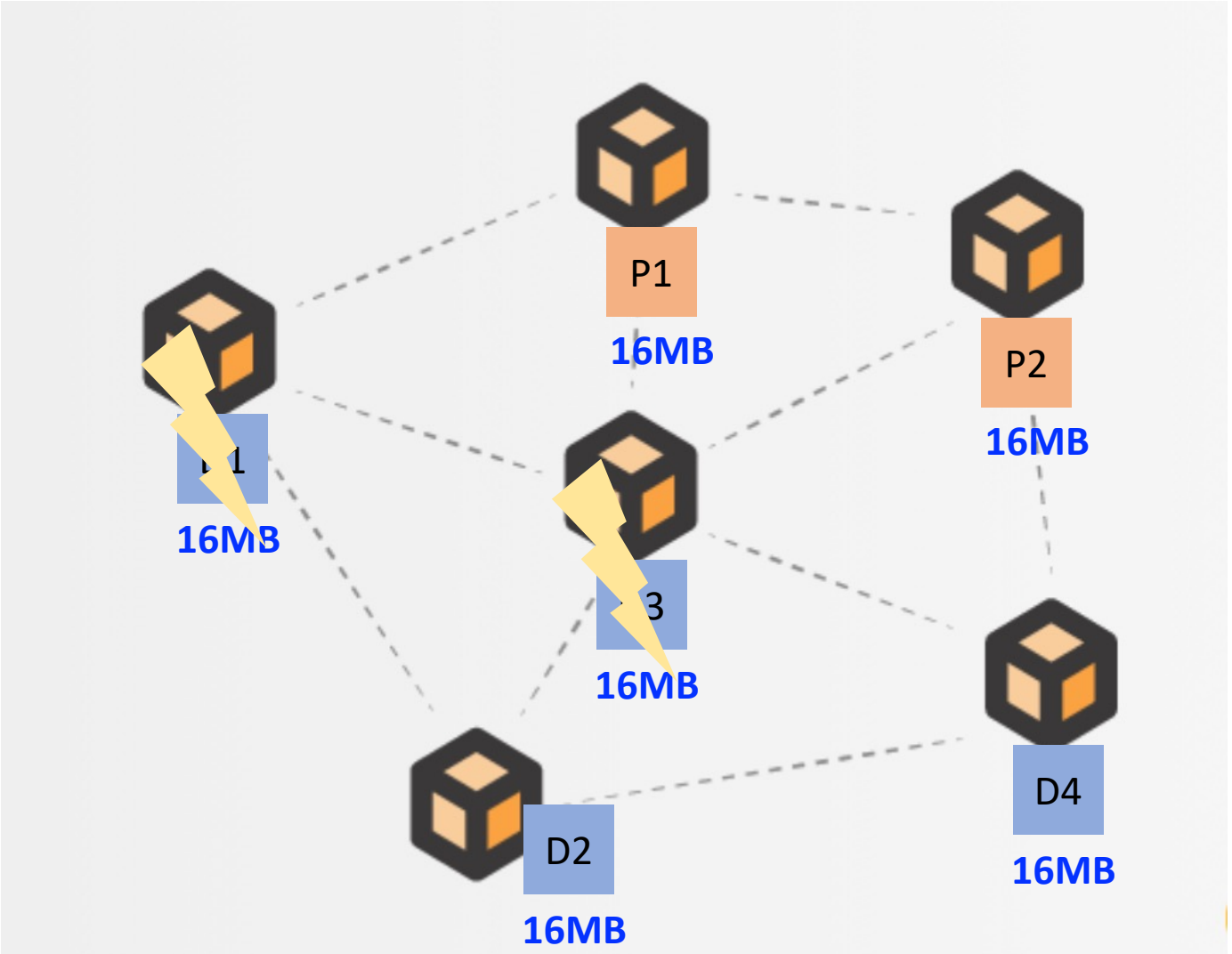


Each chunk's size = 64MB / 4 = 16MB

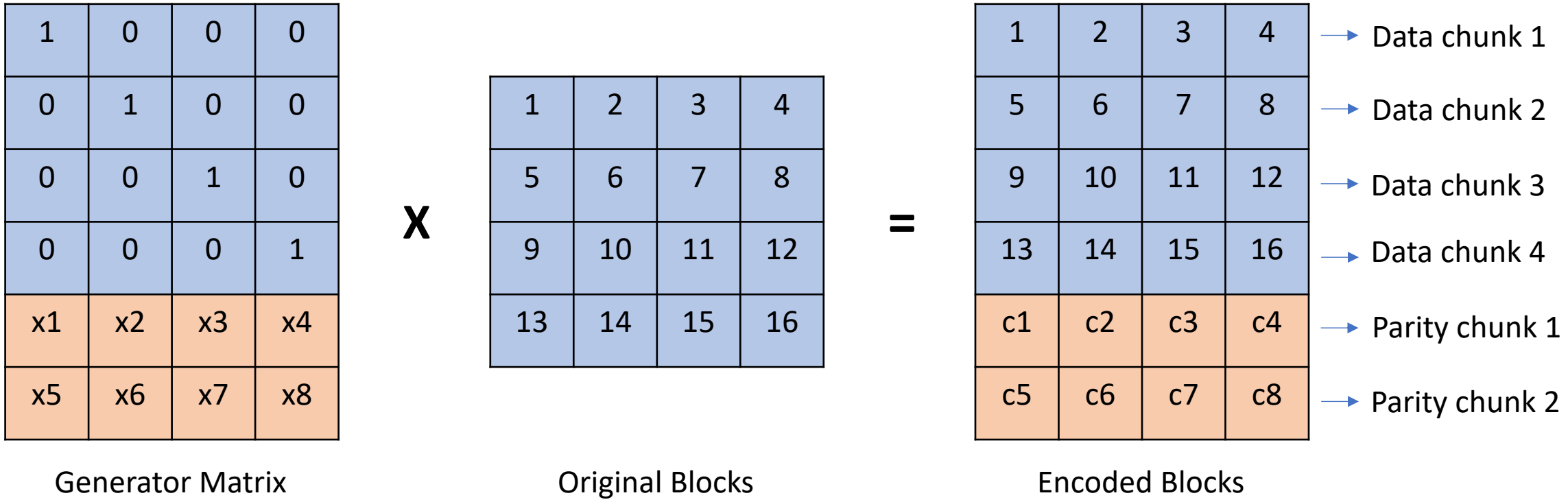


Hyperledger Fabric with Erasure Coding





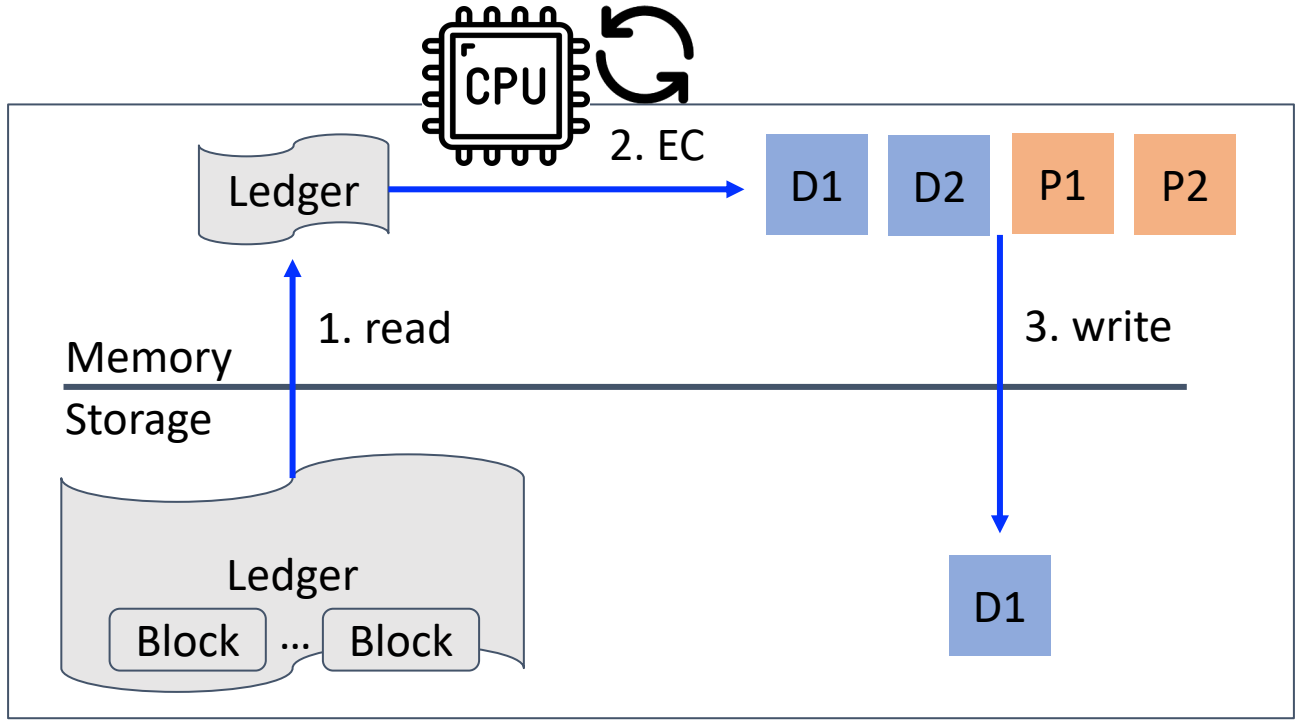
Problem #1: CPU Contention



Erasure Coding involves matrix multiplication, making it a CPU-intensive task

→ **Causes CPU Cycle competition with Fabric internal tasks and host's application**

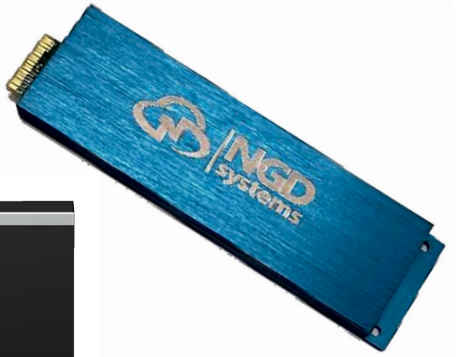
Problem #2: I/O Amplification



Reading the blockfile(ledger) into memory and then writing it back to storage is required, leading to **additional disk I/O**

Offloading EC to CSD in Hyperledger Fabric

- Each CSD has unstandardized performance for each manufacturer.[1]
- **What we do:**
  Explore opportunities for Host CPU usage reduction and Near-Data-Processing.

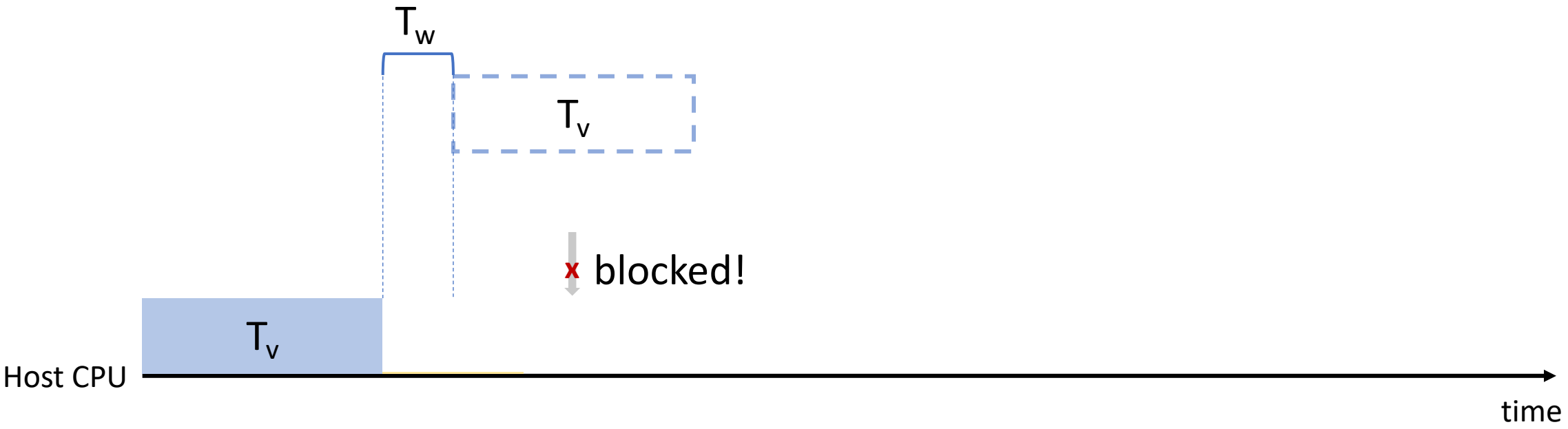


[1] Hongsu Byun, et al. An Analytical Model-based Capacity Planning Approach for Building CSD-based Storage Systems. ACM TECS 2023.

Outline

- Background
- Motivation
- **Approach: Mathematical Modeling**
- Use Case Study: NewportCSD (NGD System)
- Concluding Remark

CPU Schedule - baseline (no CSD)

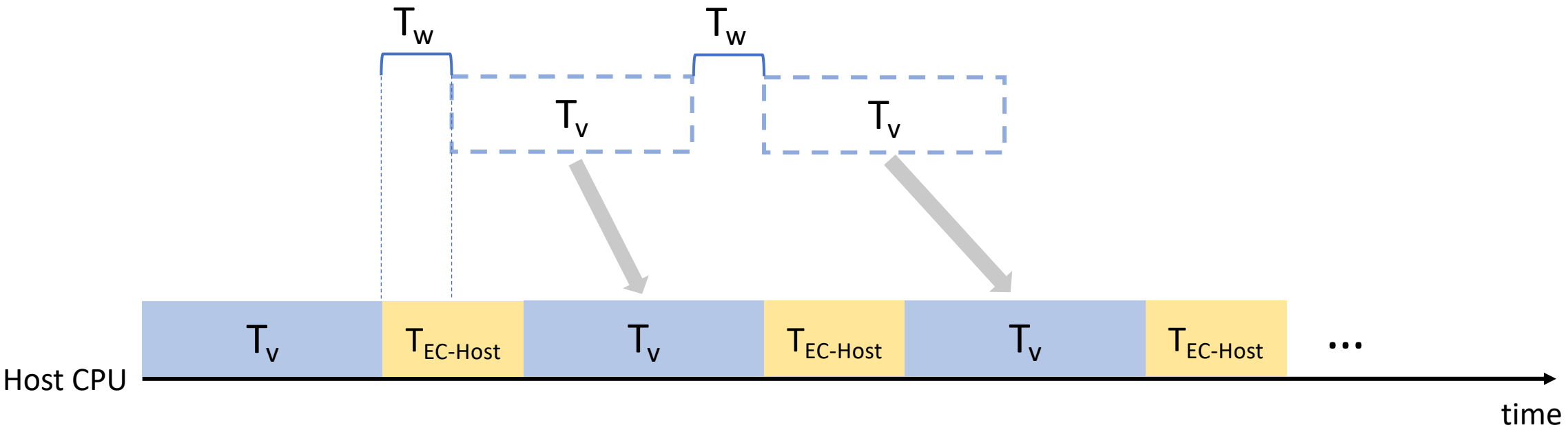


T_v = 블록 검증 및 트랜잭션 처리 작업 (validation, MVCC, …)

$T_{EC\text{-}Host}$ = Erasure Coding 실행

T_w = 다음 블록이 도착하는데 걸리는 시간

CPU Schedule - baseline (no CSD)

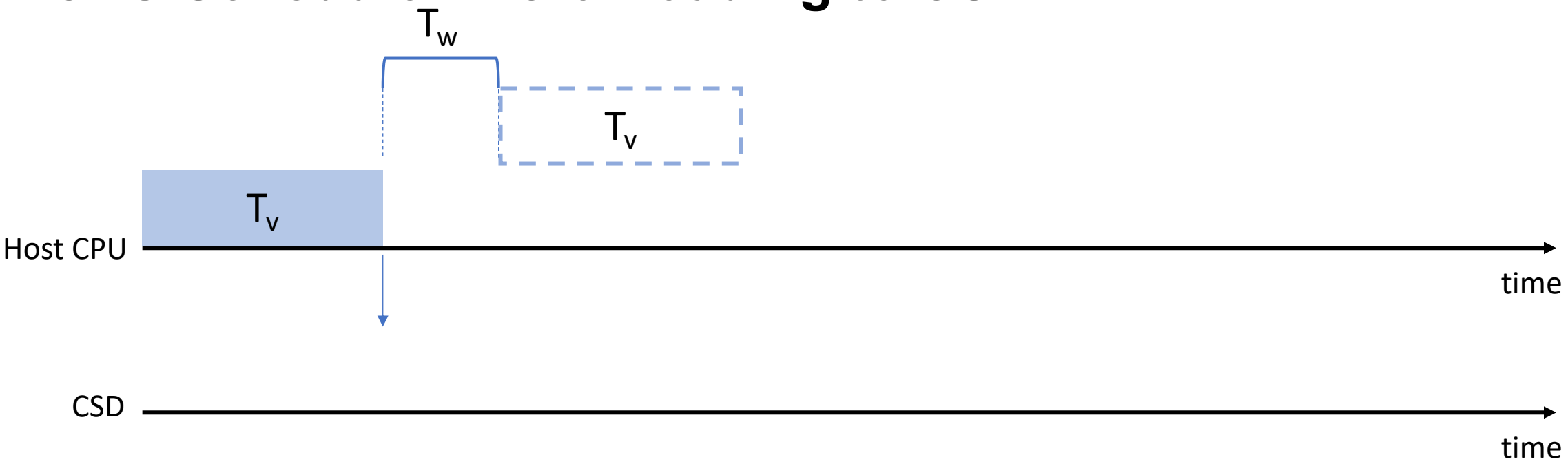


T_v = 블록 검증 및 트랜잭션 처리 작업 (validation, MVCC, …)

$T_{EC\text{-}Host}$ = Erasure Coding 실행

T_w = 다음 블록이 도착하는데 걸리는 시간

CPU Schedule - EC offloading to CSD

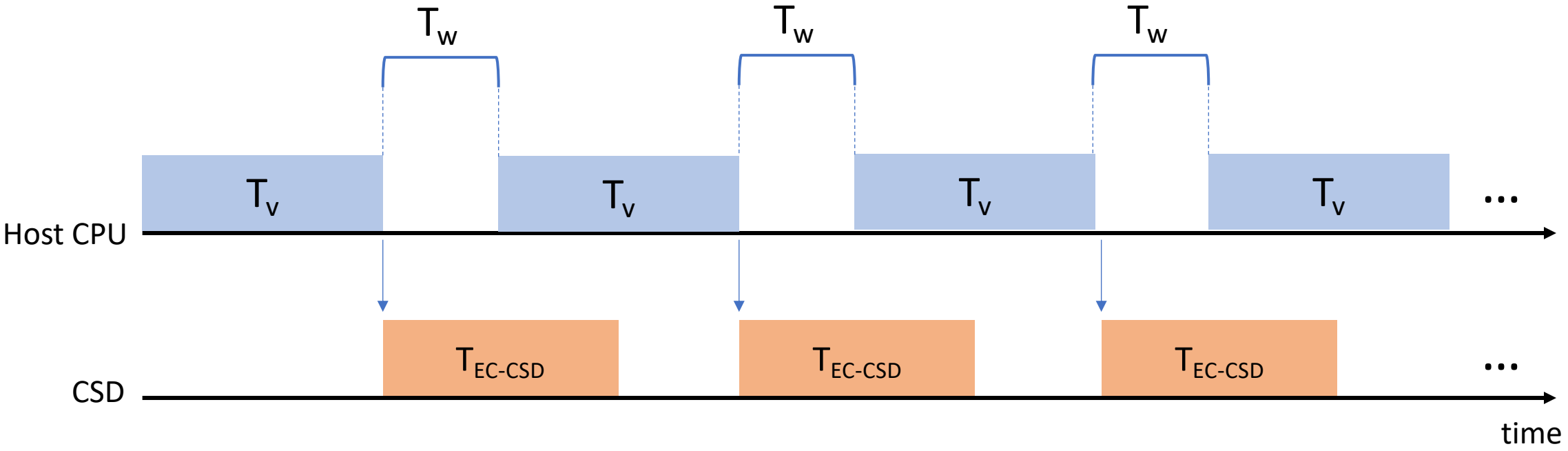


T_v = 블록 검증 및 트랜잭션 처리 작업 (validation, MVCC, …)

$T_{EC\text{-}CSD}$ = Erasure Coding 실행 (CSD)

T_w = 다음 블록이 도착하는데 걸리는 시간

CPU Schedule - EC offloading to CSD



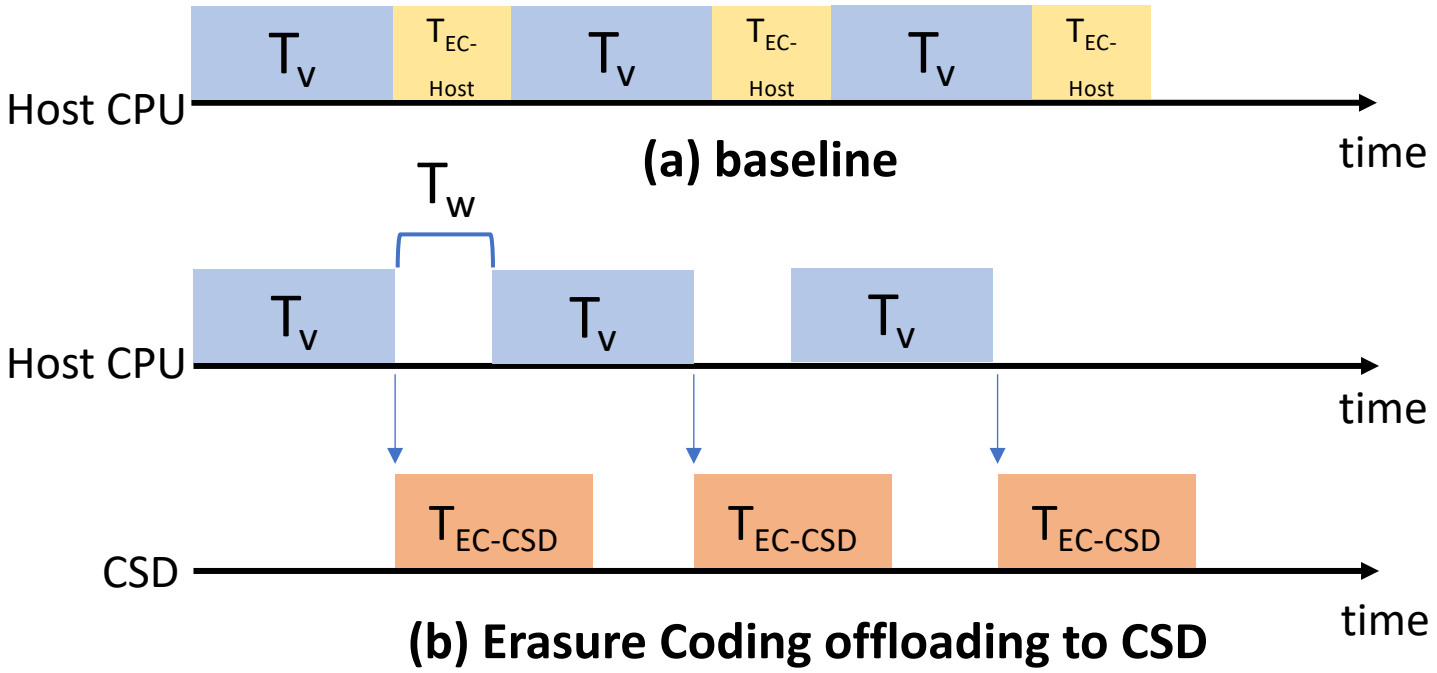
T_v = 블록 검증 및 트랜잭션 처리 작업 (validation, MVCC, …)

T_{EC-CSD} = Erasure Coding 실행 (CSD)

T_w = 다음 블록이 도착하는데 걸리는 시간

Approach: Executing EC on CSD

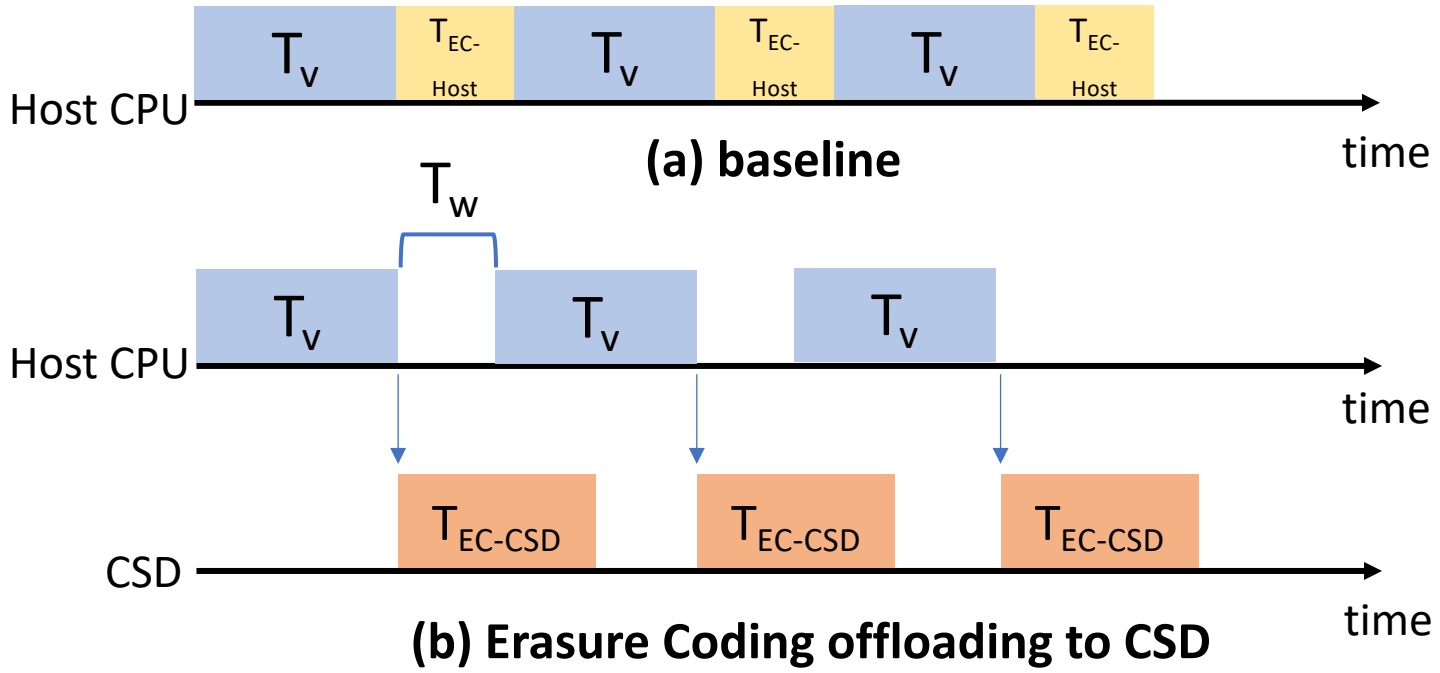
1. $T_v + T_w > T_{EC-CSD}$
2. $\sum T_{EC-Host} > \sum T_w + T_{EC-CSD}$



T_v = Validation $T_{EC-Host}$ = EC (Host CPU) T_{EC-CSD} = EC (CSD) T_w = waiting time

Approach: Executing EC on CSD

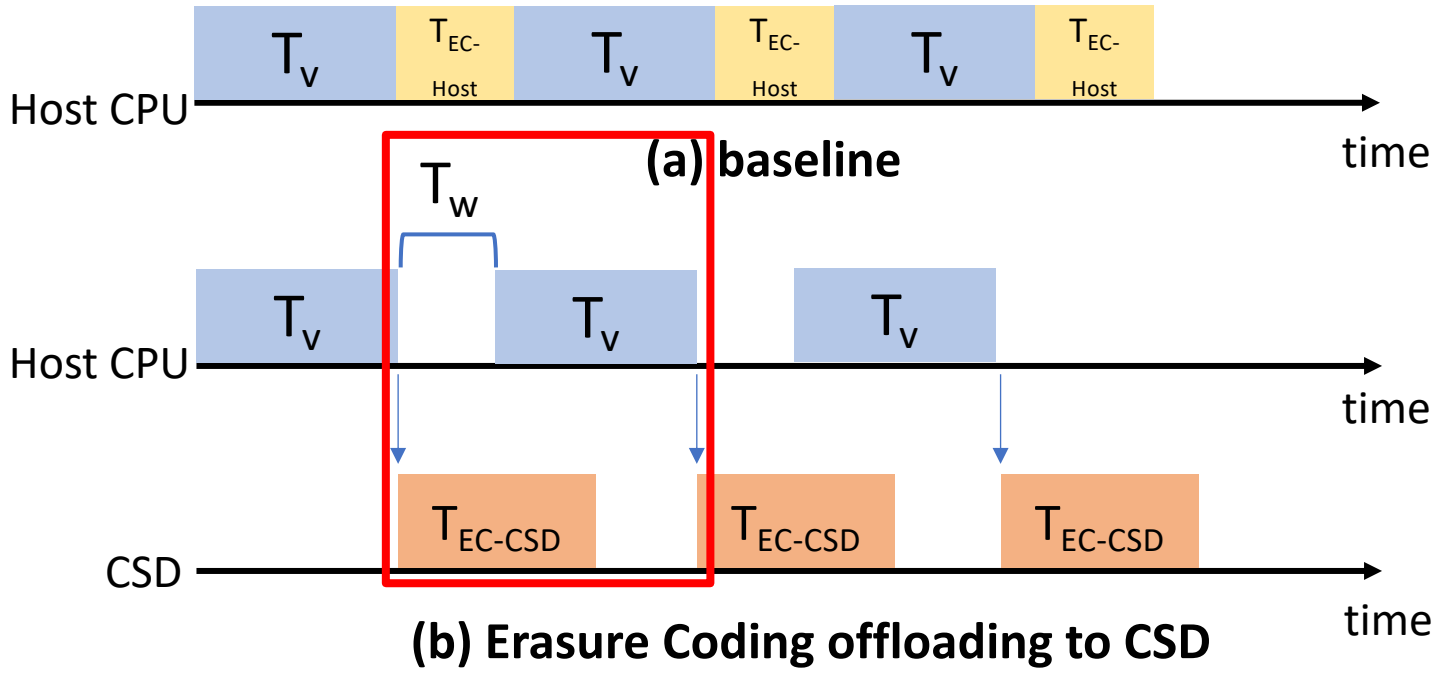
1. $T_v + T_w > T_{EC-CSD}$



T_v = Validation $T_{EC-Host}$ = EC (Host CPU) T_{EC-CSD} = EC (CSD) T_w = waiting time

Approach: Executing EC on CSD

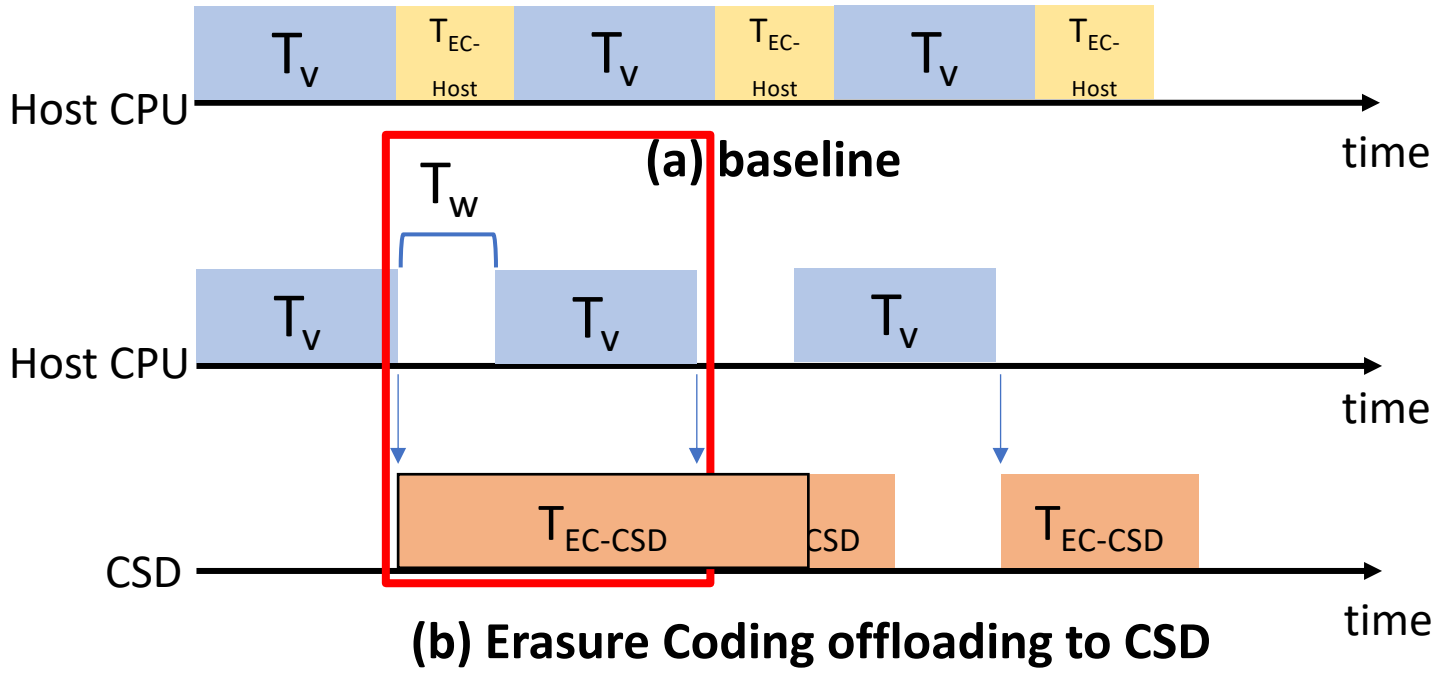
1. $T_v + T_w > T_{EC-CSD}$



T_v = Validation $T_{EC-Host}$ = EC (Host CPU) T_{EC-CSD} = EC (CSD) T_w = waiting time

Approach: Executing EC on CSD

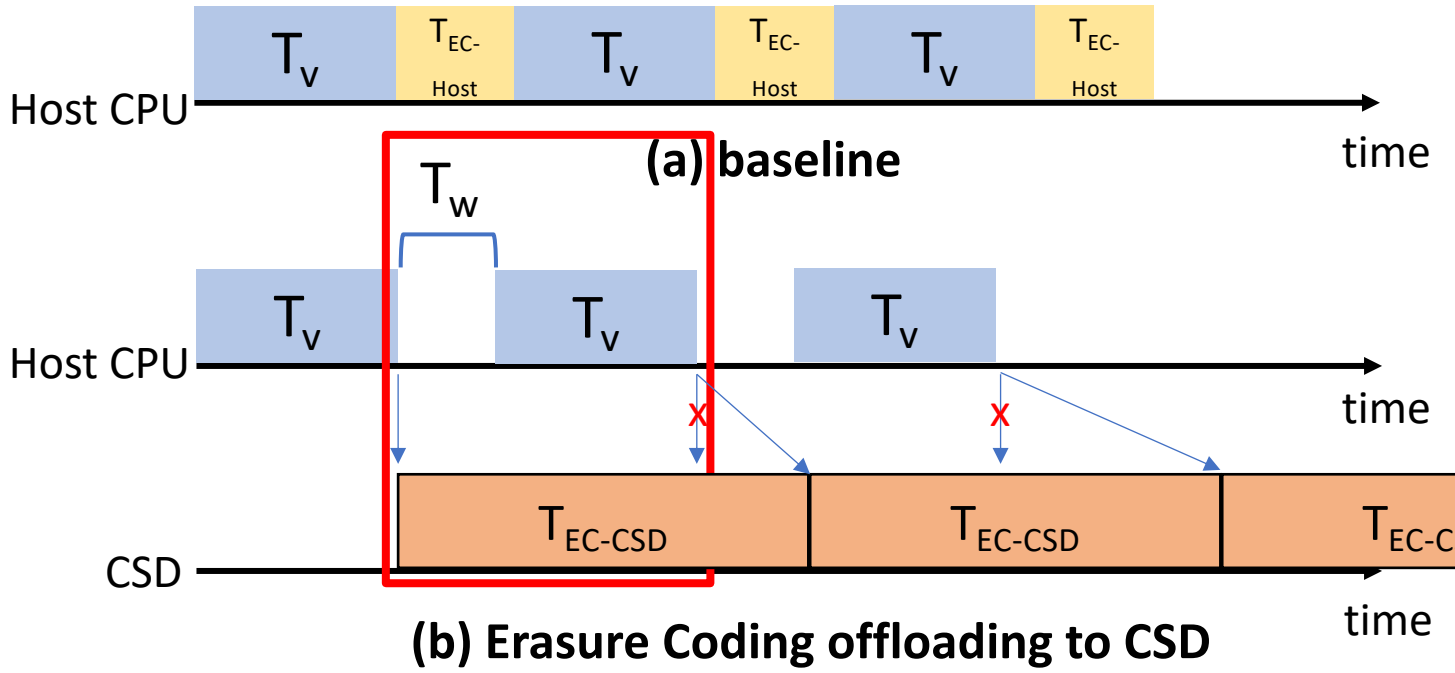
1. $T_v + T_w > T_{EC-CSD}$



T_v = Validation $T_{EC-Host}$ = EC (Host CPU) T_{EC-CSD} = EC (CSD) T_w = waiting time

Approach: Executing EC on CSD

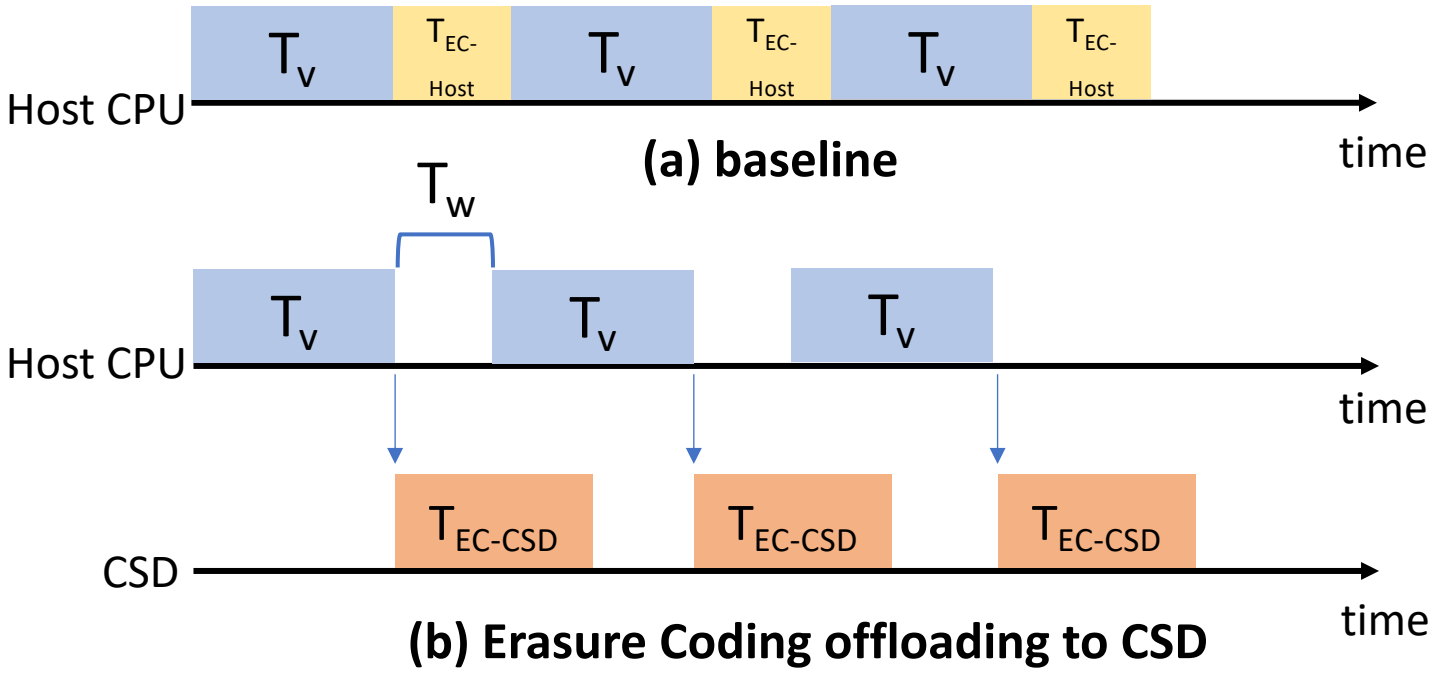
1. $T_v + T_w > T_{EC-CSD}$



T_v = Validation $T_{EC-Host}$ = EC (Host CPU) T_{EC-CSD} = EC (CSD) T_w = waiting time

Approach: Executing EC on CSD

1. $T_v + T_w > T_{EC-CSD}$
2. $\sum T_{EC-Host} > \sum T_w + T_{EC-CSD}$



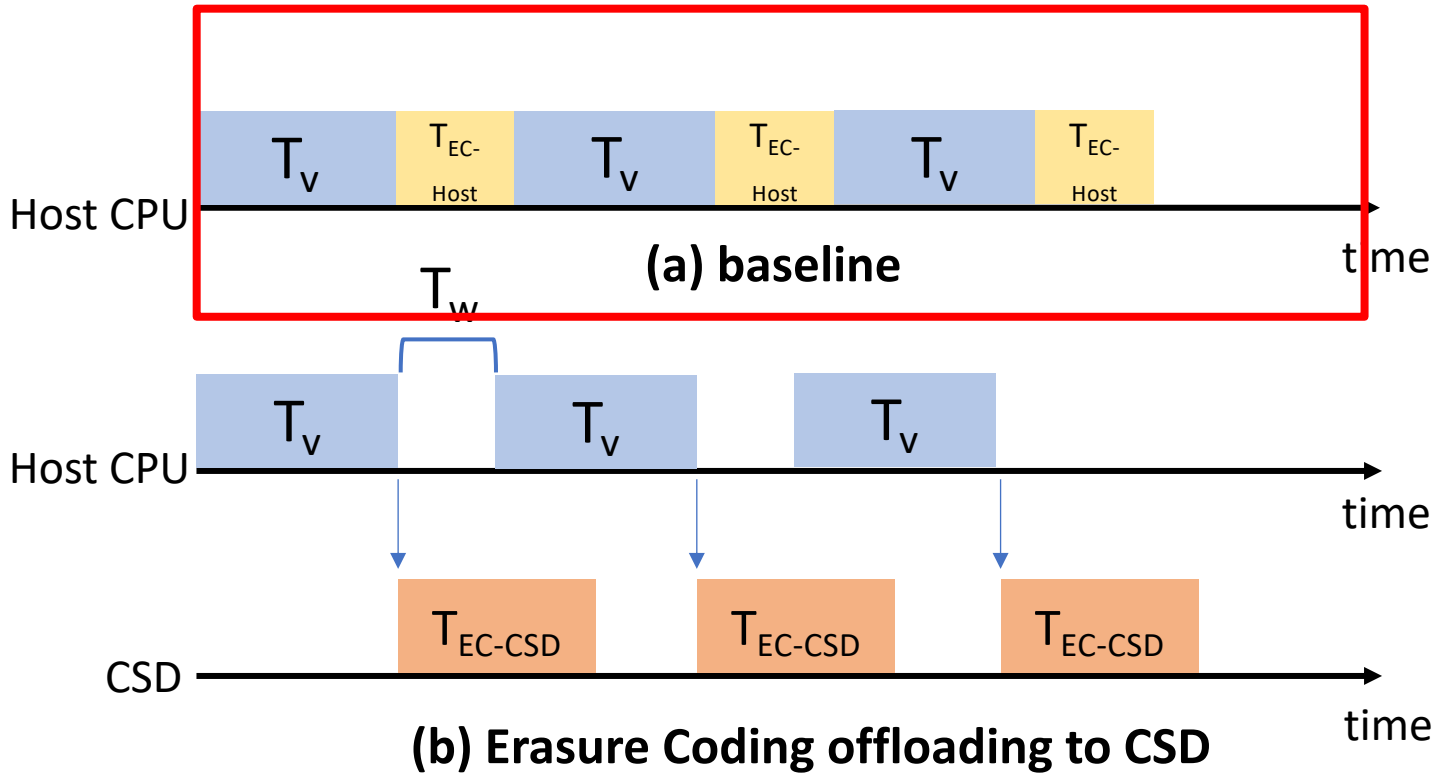
T_v = Validation $T_{EC-Host}$ = EC (Host CPU) T_{EC-CSD} = EC (CSD) T_w = waiting time

Approach: Executing EC on CSD

1. $T_v + T_w > T_{EC-CSD}$
2. $\sum T_{EC-Host} > \sum T_w + T_{EC-CSD}$

Total execution time

(a) $= \sum T_v + \sum T_{EC-Host}$



T_v = Validation $T_{EC-Host}$ = EC (Host CPU) T_{EC-CSD} = EC (CSD) T_w = waiting time

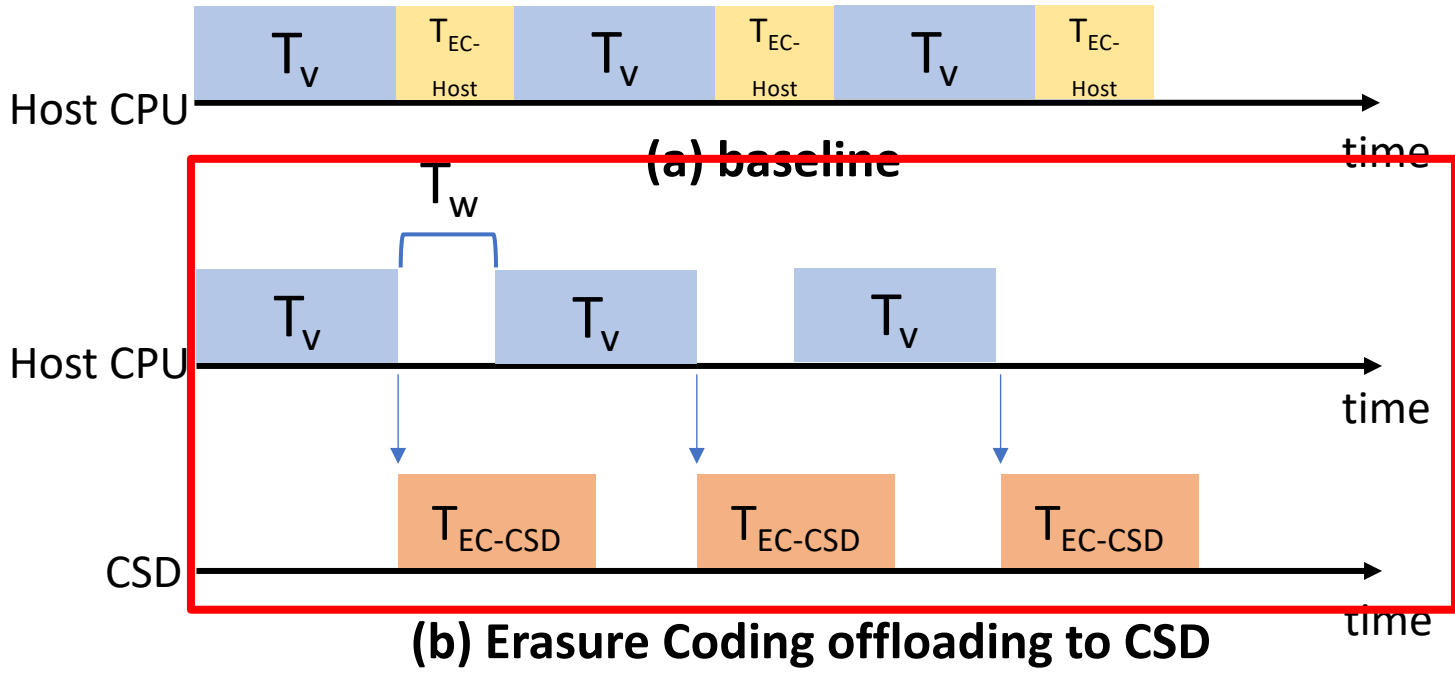
Approach: Executing EC on CSD

1. $T_v + T_w > T_{EC-CSD}$
2. $\sum T_{EC-Host} > \sum T_w + T_{EC-CSD}$

Total execution time

(a) $= \sum T_v + \sum T_{EC-Host}$

(b) $= \sum T_v + \sum T_w + T_{EC-CSD}$



T_v = Validation $T_{EC-Host}$ = EC (Host CPU) T_{EC-CSD} = EC (CSD) T_w = waiting time

Approach: Executing EC on CSD

1. $T_v + T_w > T_{EC-CSD}$
2. $\sum T_{EC-Host} > \sum T_w + T_{EC-CSD}$

Total execution time

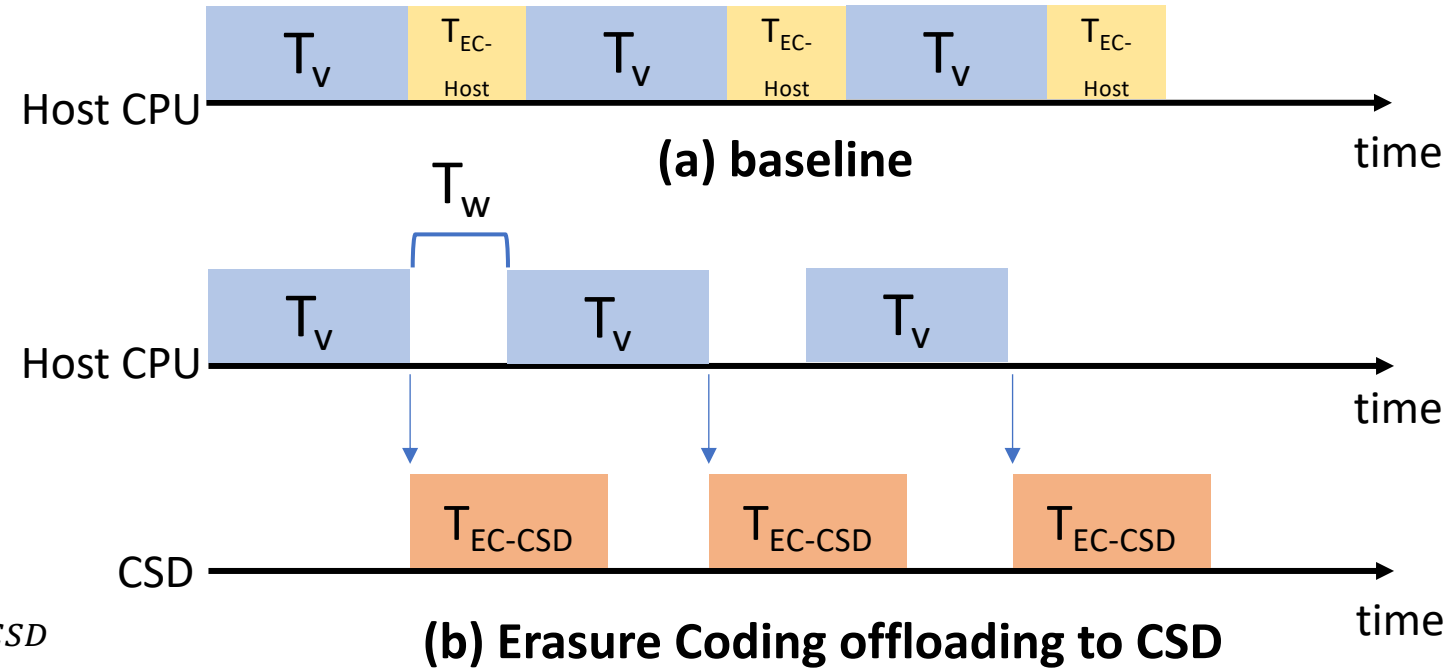
(a) $= \sum T_v + \sum T_{EC-Host}$

(b) $= \sum T_v + \sum T_w + T_{EC-CSD}$

(a) > (b)

$\rightarrow \sum T_v + \sum T_{EC-Host} > \sum T_v + \sum T_w + T_{EC-CSD}$

$\rightarrow \sum T_{EC-Host} > \sum T_w + T_{EC-CSD}$

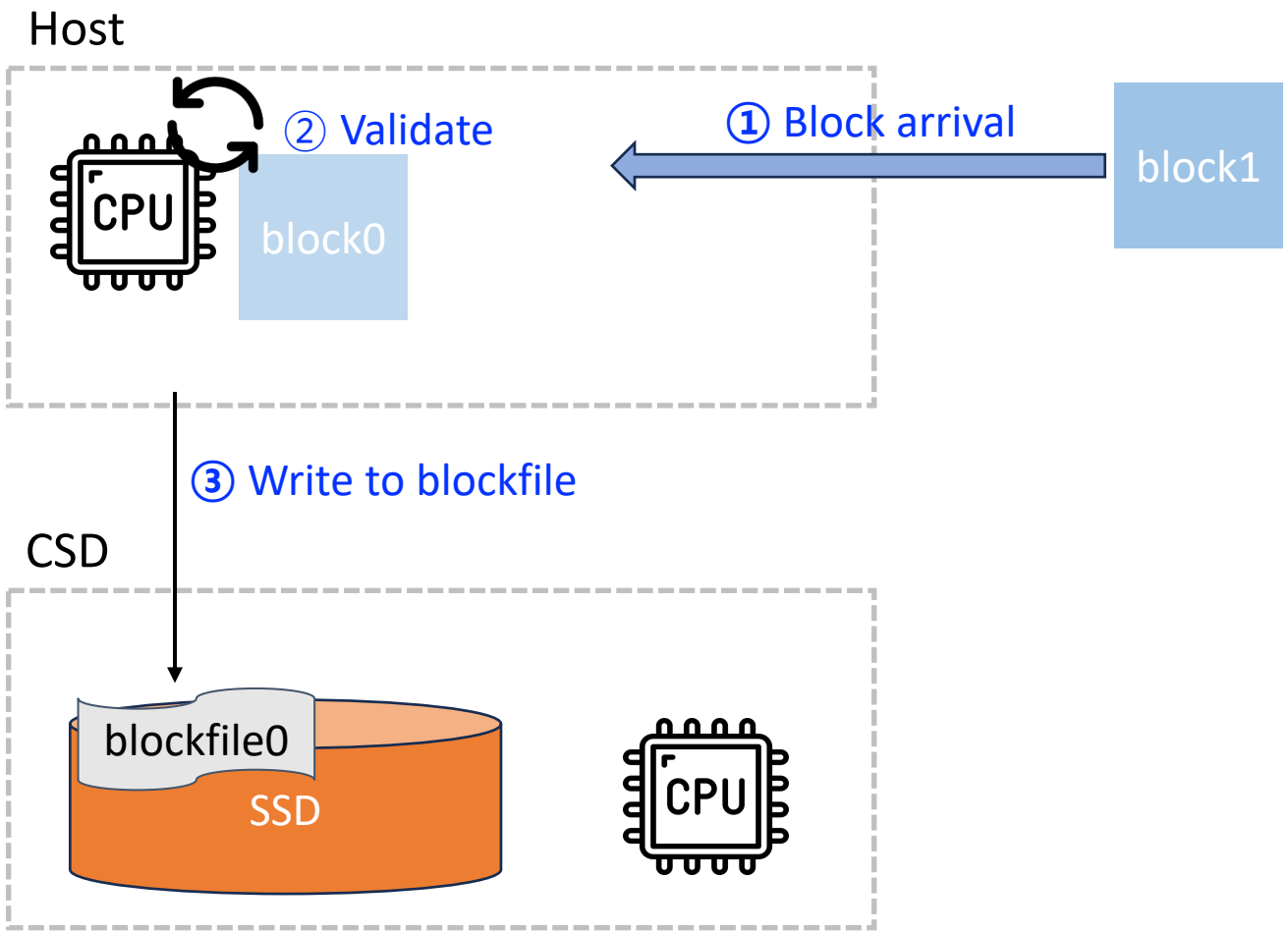


T_v = Validation $T_{EC-Host}$ = EC (Host CPU) T_{EC-CSD} = EC (CSD) T_w = waiting time

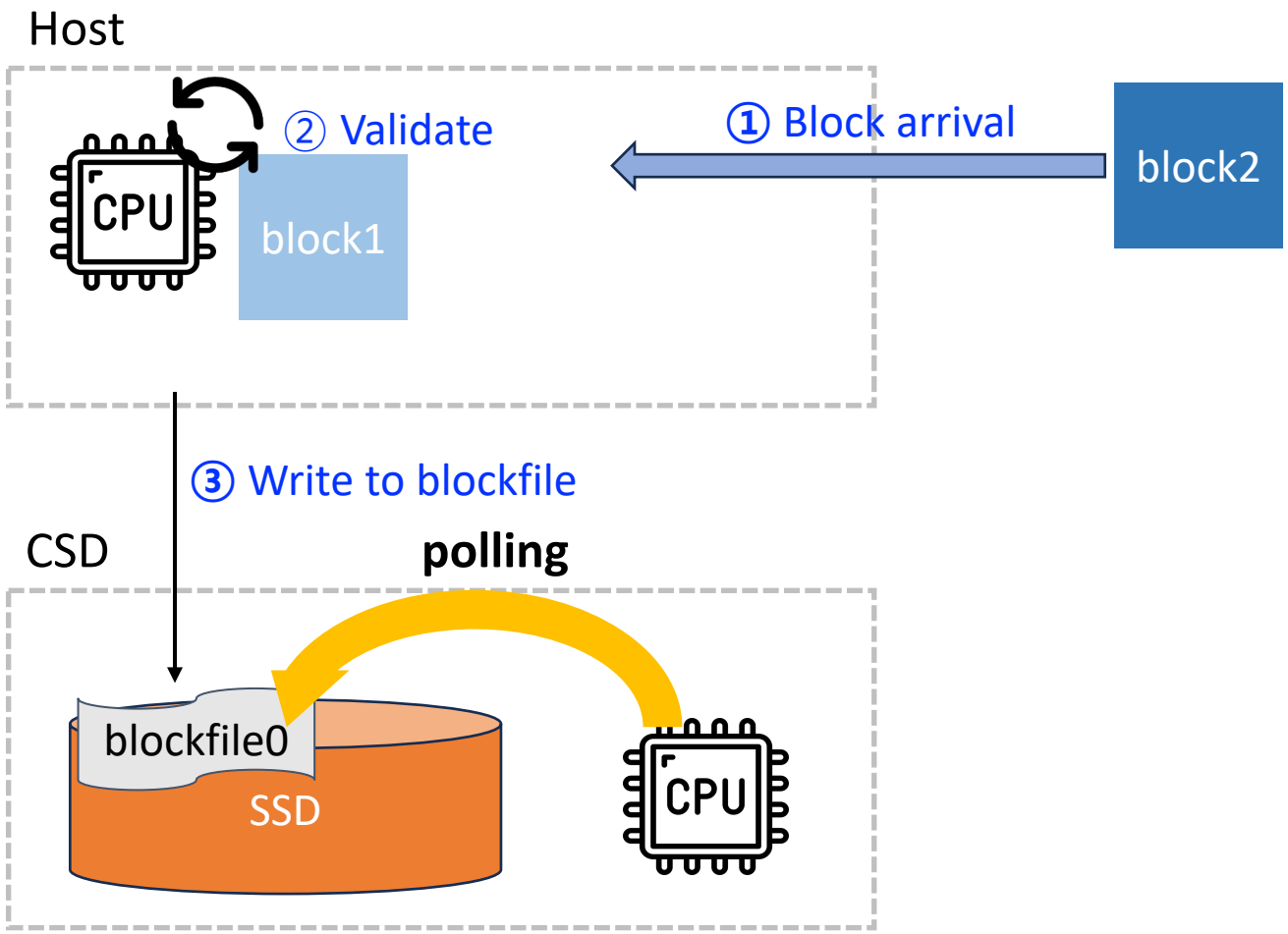
Outline

- Background
- Motivation
- Approach: Mathematical Modeling
- **Use Case Study: Newport CSD (NGD System)**
- Concluding Remark

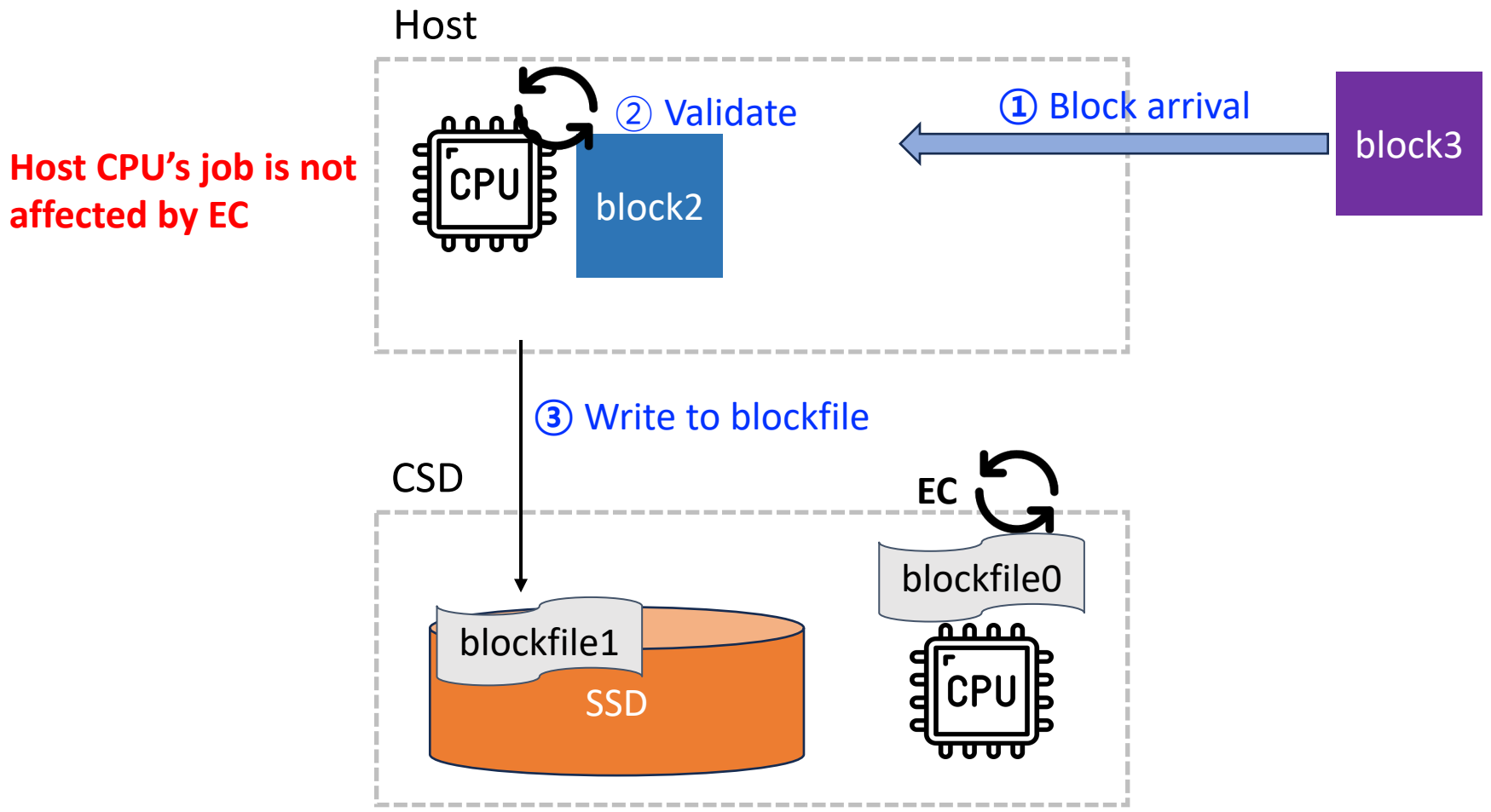
EC on CSD: Implementation



EC on CSD: Implementation



EC on CSD: Implementation



Experimental Setup

- Fabric network configuration: 20 peers, 1 orderer
- Each peer is virtualized by a Docker container within a single server
- EC parameter: n = 20, k = 14
  (generates 14 chunks divided from original data and 6 parity chunks)

CPU	AMD EPYC™ 7352, 48 Cores (96 Threads), 2.3GHz (Up to 3.2GHz) 128 MB L3 Cache
Socket	2 NUMA Node
Memory	256 GB (64 GB × 4) DRAM DDR4 3200MHz
OS	Centos 7.92.2009 (Core) / Linux Kernel 4.14

<Specifications of experimental environment >

Experimental Setup

- Fabric network configuration: 20 peers, 1 orderer
- Each peer is virtualized by a Docker container within a single server
- EC parameter: n = 20, k = 14
  (generates 14 chunks divided from original data and 6 parity chunks)

Type	Clients (#)	Block arrival rate (CPU 100%)
Heavy	16	7.8MB/s
Medium	8	6.2MB/s
Light	1	2.4MB/s

<Specifications of workloads>

Experimental Setup

- Fabric network configuration: 20 peers, 1 orderer
- Each peer is virtualized by a Docker container within a single server
- EC parameter: n = 20, k = 14
  (generates 14 chunks divided from original data and 6 parity chunks)

Type	Clients (#)	Block arrival rate (CPU 100%)
Heavy	16	7.8MB/s
Medium	8	6.2MB/s
Light	1	2.4MB/s

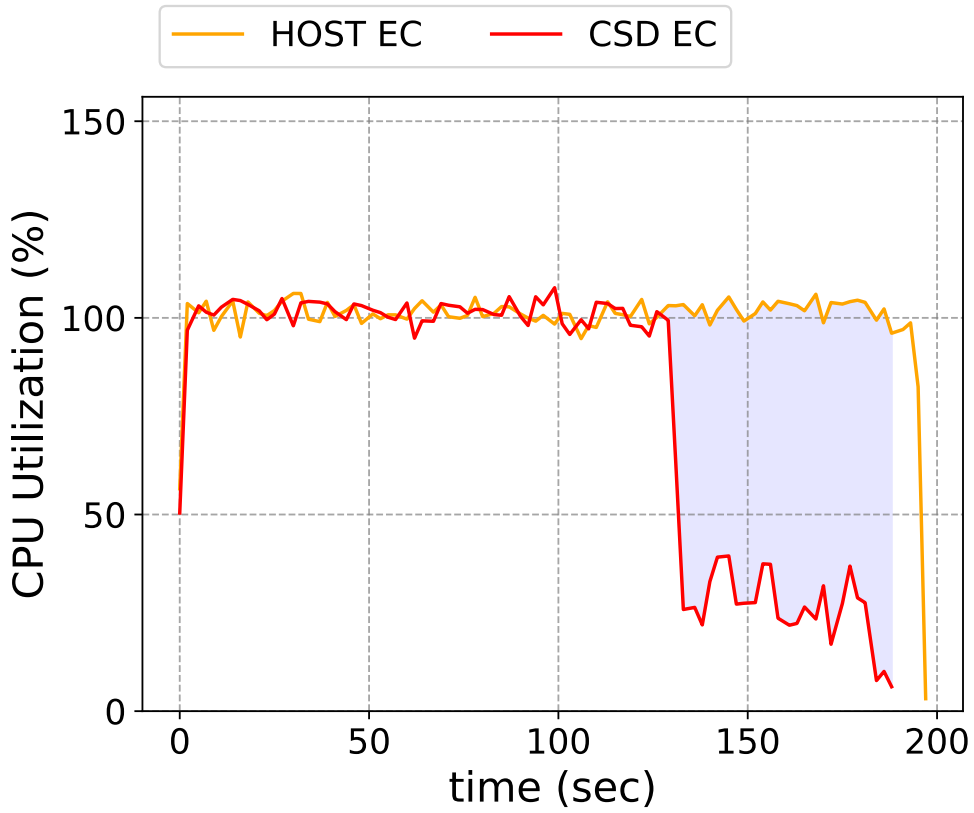
<Specifications of workloads>

Experimental Setup

- CSD: Newport CSD (NGD Systems)

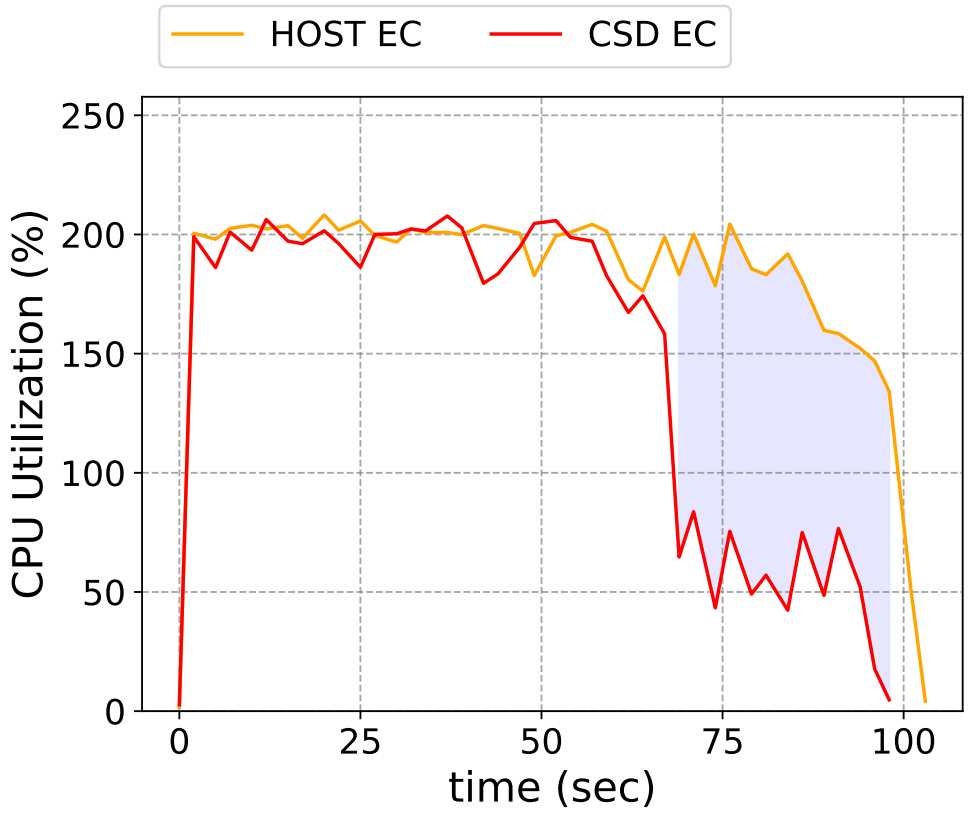
	Newport CSD
Storage Capacity	8 TB
Host Interface	PCIe Gen3×4 (U.2)
In-Stroage Processing Engine	ARM Cortex-A53 1.0 GHz, 4 Cores
	8 GB DDR4 DRAM
	OS : Linux Kernel 4.14

Evaluation – Offloading EC to CSD



(a) CPU Core Max 100%

28% decreased



(b) CPU Core Max 200%

23% decreased

Outline

- Background
- Motivation
- Approach: Mathematical Modeling
- Use Case Study: NewportCSD (NGD System)
- **Concluding Remark**

Conclusion & Future works

- We observed Erasure Coding overhead at Hyperledger Fabric.
- To reduce the overhead, we tried to offload Erasure Coding to NewportCSD.
- We plan to investigate other CSDs and study more suitable offloading mechanism for Hyperledger Fabric.

Thank you

Presenter: Junghyun Ryu

Contact: jhryu@sogang.ac.kr

Back up