Building Reliable NAND Flash Memory Storage Systems

Kevin M. Greenan, Ethan L. Miller and Darrell D.E. Long
UCSC
Thomas Schwarz
Santa Clara University
Kaladhar Voruganti, Garth Goodson and Jon Elerath
NetApp

NAND Flash Memory Overview

- The Good
  - Fast random reads
  - Low power utilization
  - No moving parts
- The Bad
  - Writing involves erasing/programming
  - Reliability is dependent on usage and time
    - Endurance
    - Retention
    - Raw bit-error rate (RBER)
- Must overcome reliability concerns without hurting performance

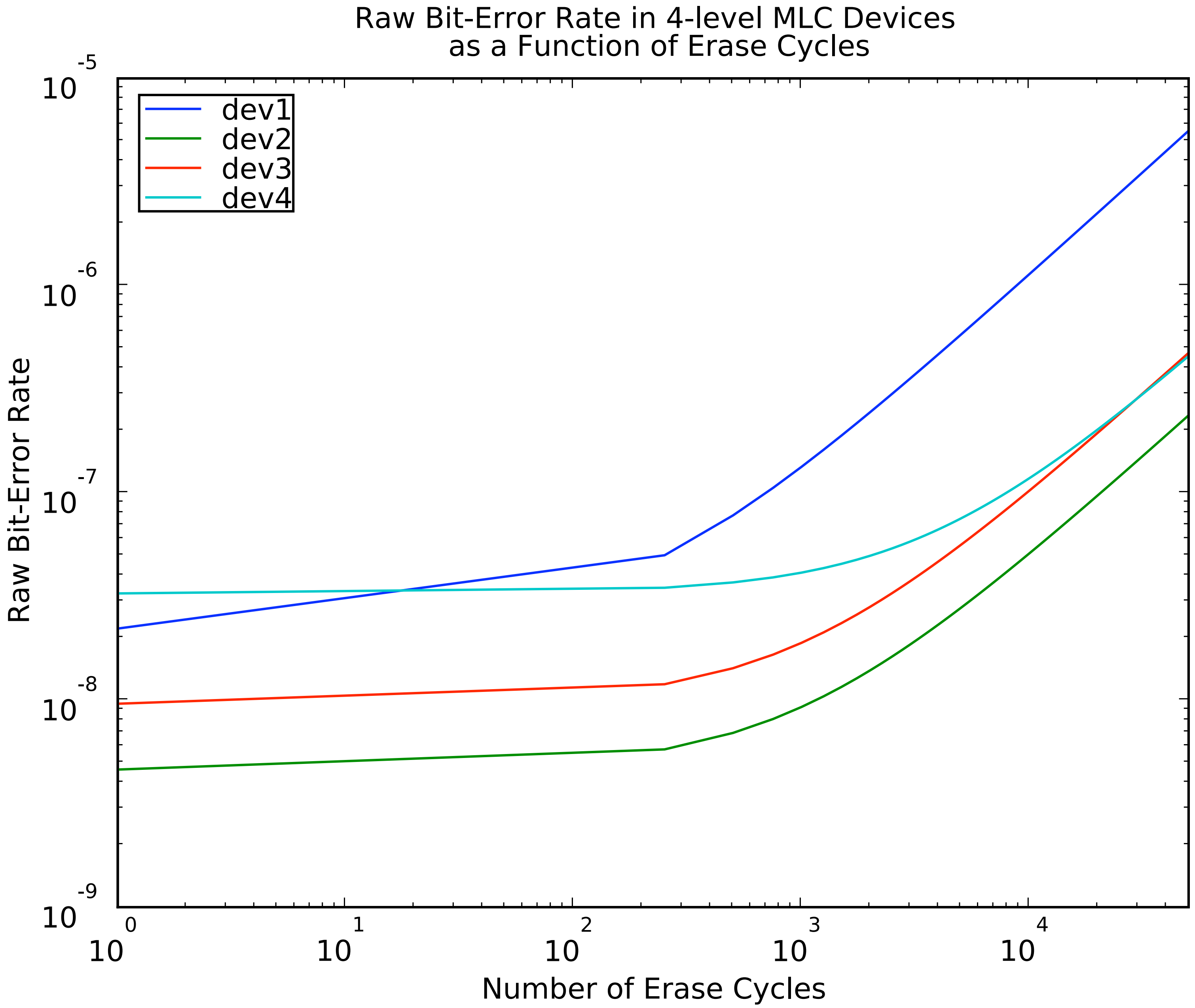
Objectives

- Improve reliability
  - Control all writes to flash
  - Put mechanisms in place to deal with increasing RBER
    - Dynamic mechanisms
    - Trade space and performance for increased fault tolerance
  - Error handling beyond bit errors
- Erasure codes provide great fit
- Maintain good performance using erasure codes
  - Stage writes in other NVRAM or BB-RAM
  - Write across as many chips as possible
  - Write sequentially to each device

Flash Media Reliability

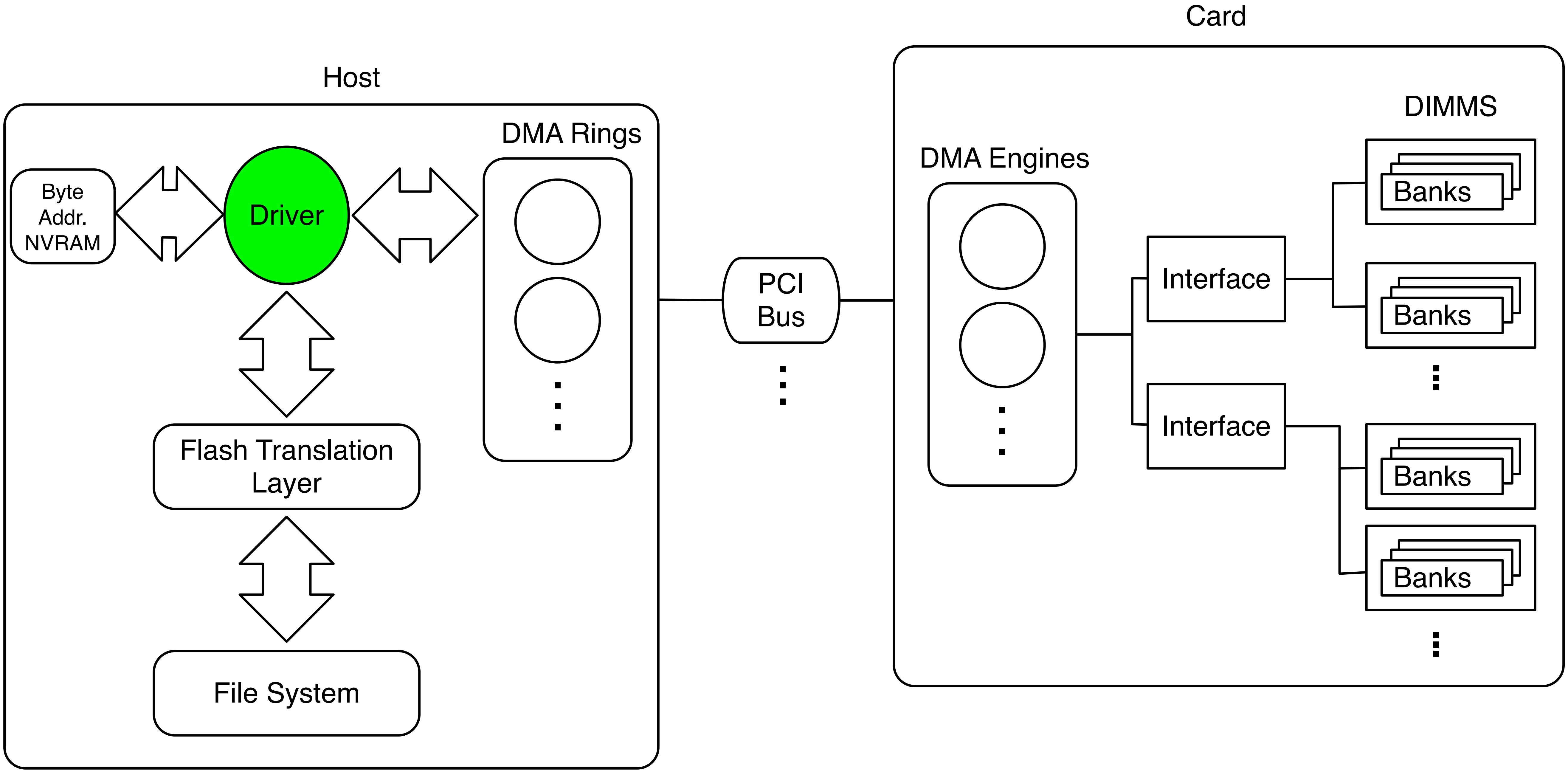
- Reliability is typically given by RBER, retention and endurance
- Each changes with:
  - Manufacturer
  - Bits per cell (i.e. SLC and MLC)
  - Use
  - Time
- Here, we consider the relationship between use and RBER
  - Still figuring out use/time dependency on RBER
- Failure of other components may also lead to data loss
  - Chips, controllers, etc.

RBER as a Function of Erase Cycles

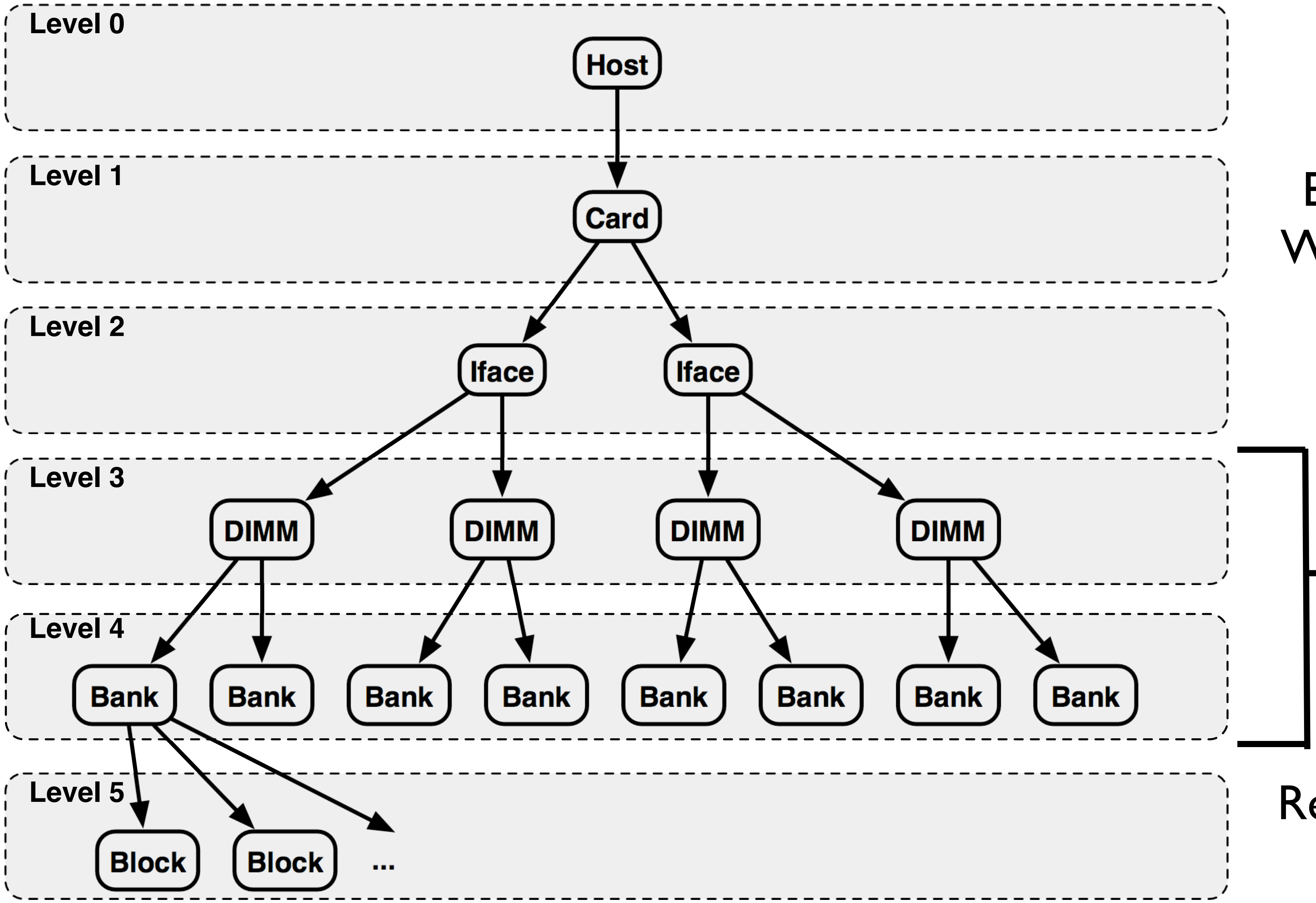


- Use has a dramatic effect on RBER!
- Data taken from Intel-Micron study
- Performed regression over data to extrapolate
- 4 devices: (1)10K cycles, (2) 5K cycles, (1) unspecified

Architecture



Threats in this Architecture



SW/HW errors
Power outage

Error on PCI bus
Whole card outage

SW Error

HW Error
Chip outage

Read/write disturbs
Defects

Options for Handling Errors

- Error Correcting Codes (ECC)
  - Correct e bit errors
  - Can detect 2e bit errors
  - Generally computed in controller (or interface)
  - Applied to sectors or pages
- Hashing
  - Easy to compute
  - Can detect any errors with very high probability
- Erasure Coding
  - Applied at coarser granularity than ECCs (i.e. multiple pages)
  - Can correct known errors via ECC or hash
  - Detect errors with very high probability
  - Easily re-code if implemented in SW

Challenges: Erasure Coding in Flash

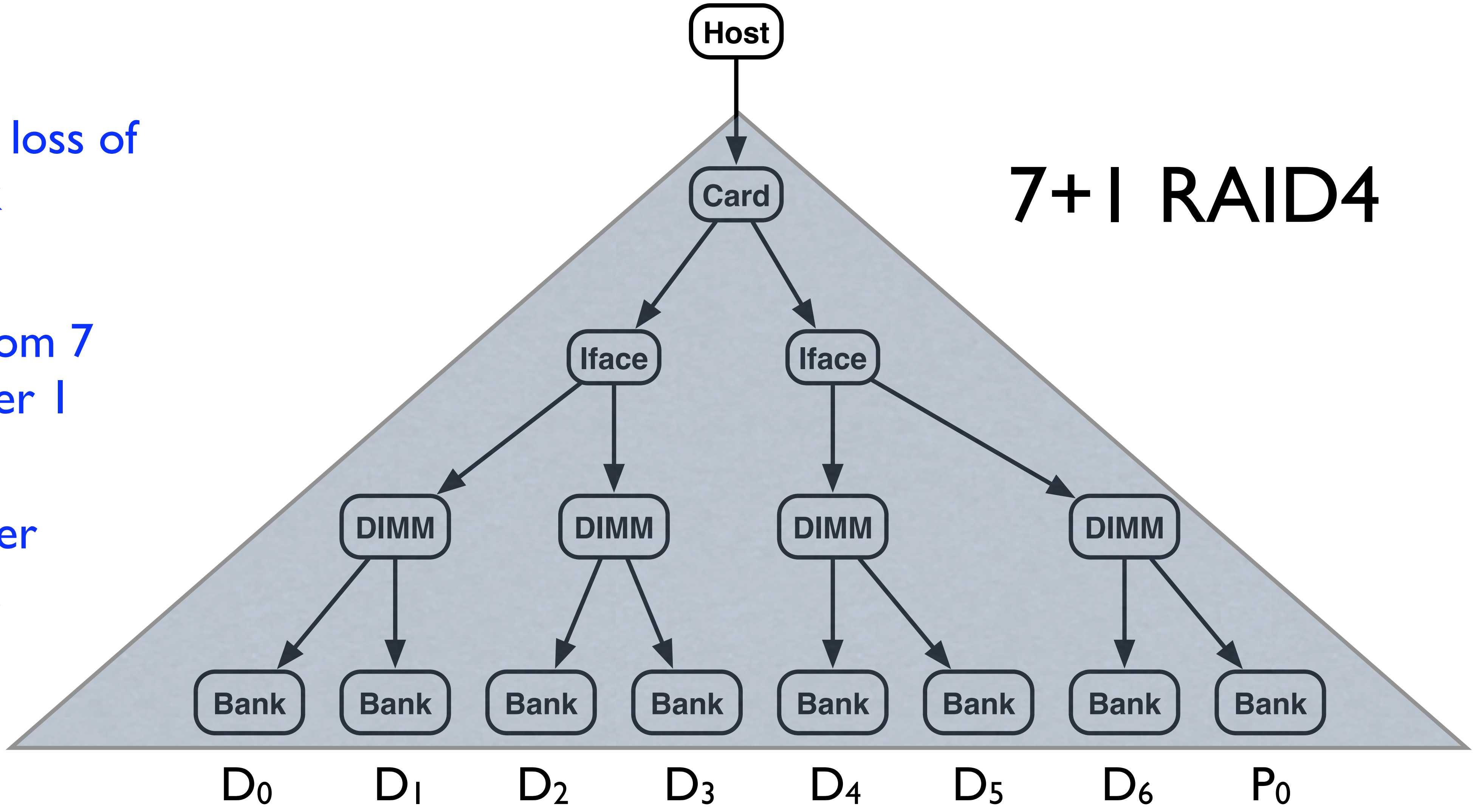
- Block management
  - Given encoding, determine addressable data blocks
- Writing erasure coded data
  - Balance writes across banks
  - Properly handle parity updates
- Rebuilding lost data
  - Localize recovery operations
- Graceful degradation
  - Provide ability to change encoding as RBER increases
- Failover
  - Determine where to put rebuilt data

Component Protection

Can sustain the loss of any bank

Requires data from 7 banks to recover 1

1 parity update per write to a bank



7+1 RAID4

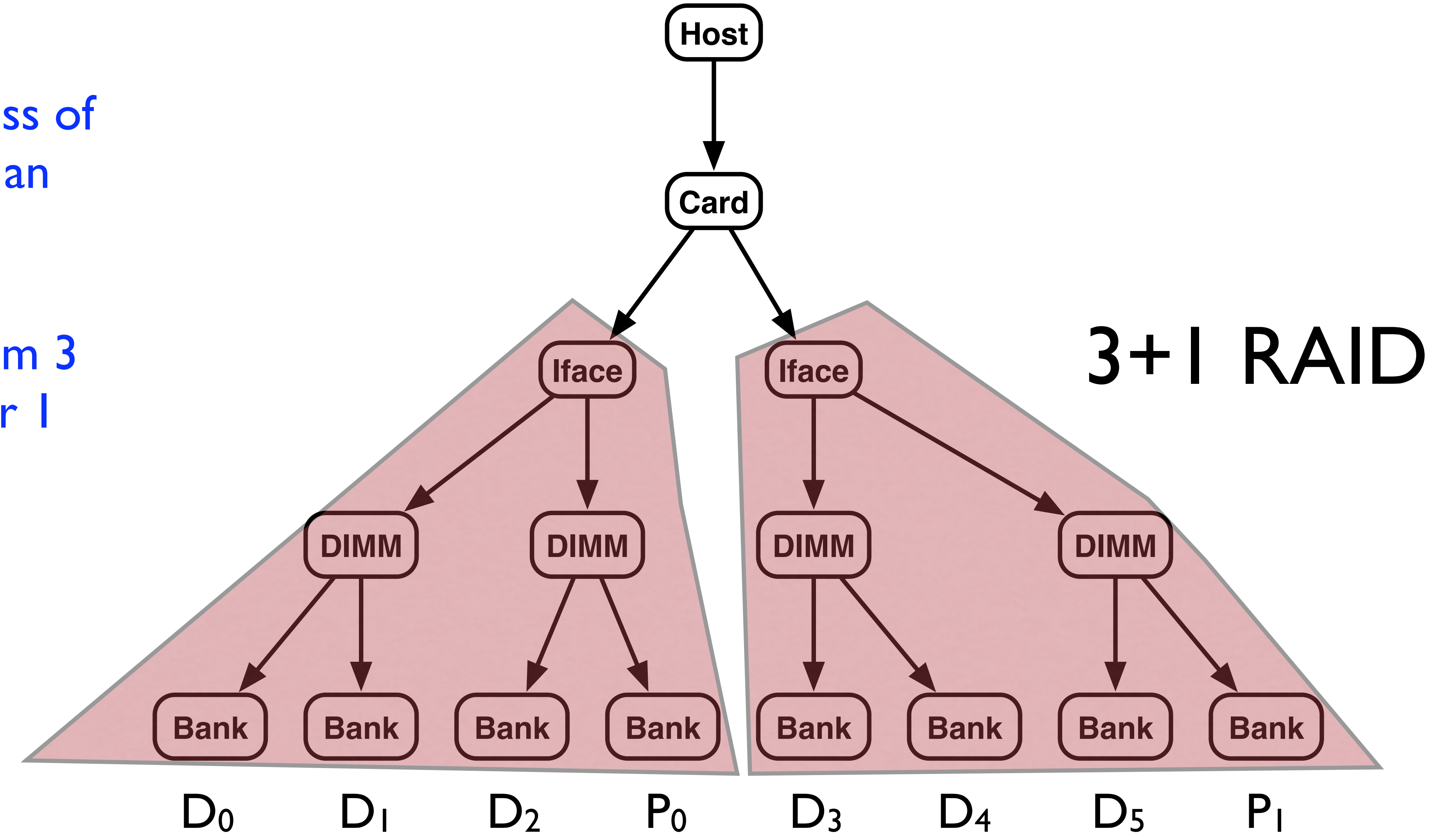
$$D_0 \oplus D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus D_5 \oplus D_6 = D_7$$

Component Protection

Can sustain the loss of any bank under an interface

Requires data from 3 banks to recover 1

1 parity update per write to a bank



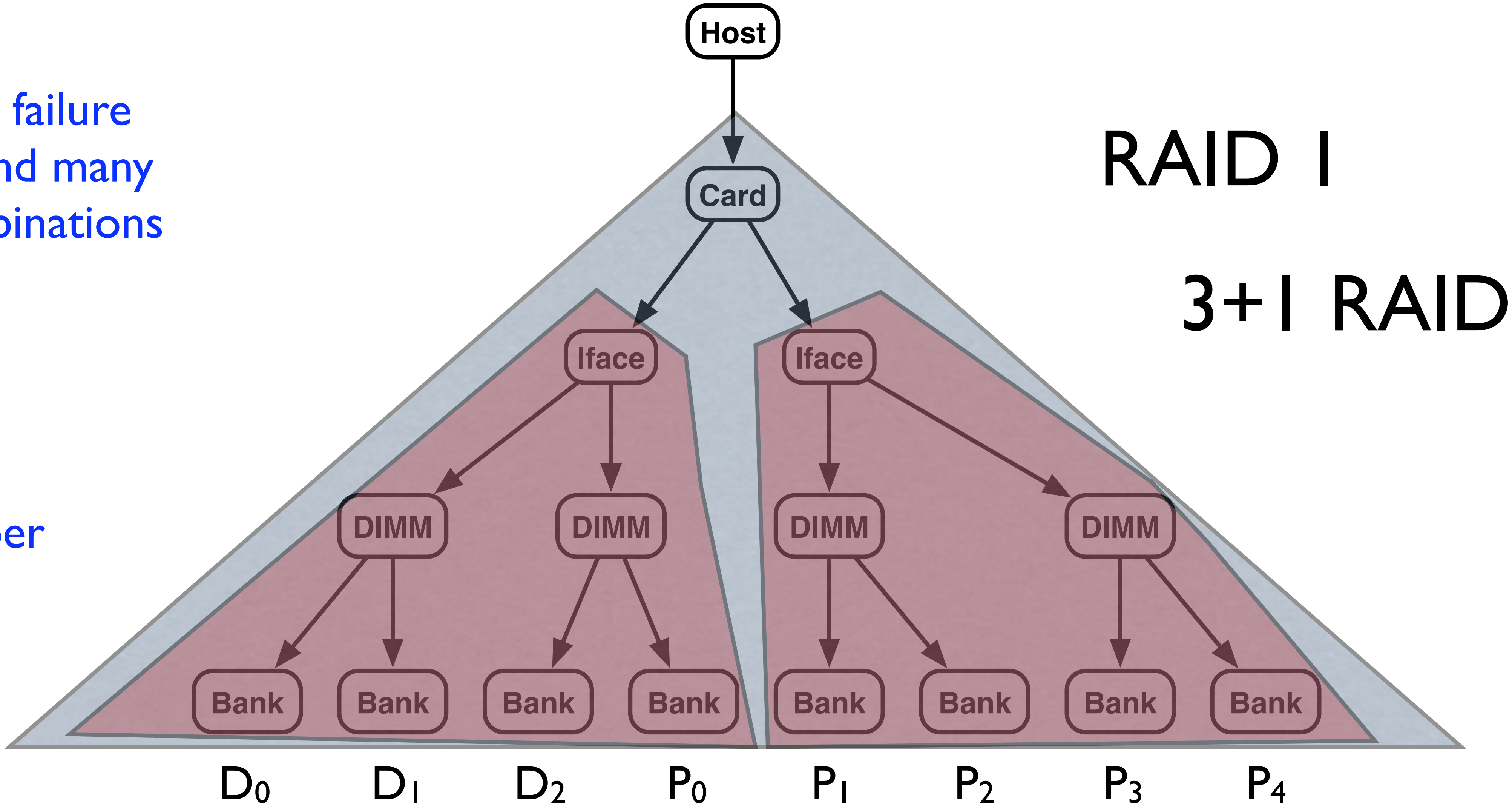
3+1 RAID4

$$D_0 \oplus D_1 \oplus D_2 = P_0$$
$$D_3 \oplus D_4 \oplus D_5 = P_1$$

Component Protection

Can sustain the failure of an interface and many bank failure combinations

3 parity updates per write to a bank



RAID 1

3+1 RAID 4

$D_0 \oplus D_1 \oplus D_2 = P_0$

$D_0 \oplus D_1 \oplus D_2 = P_4$

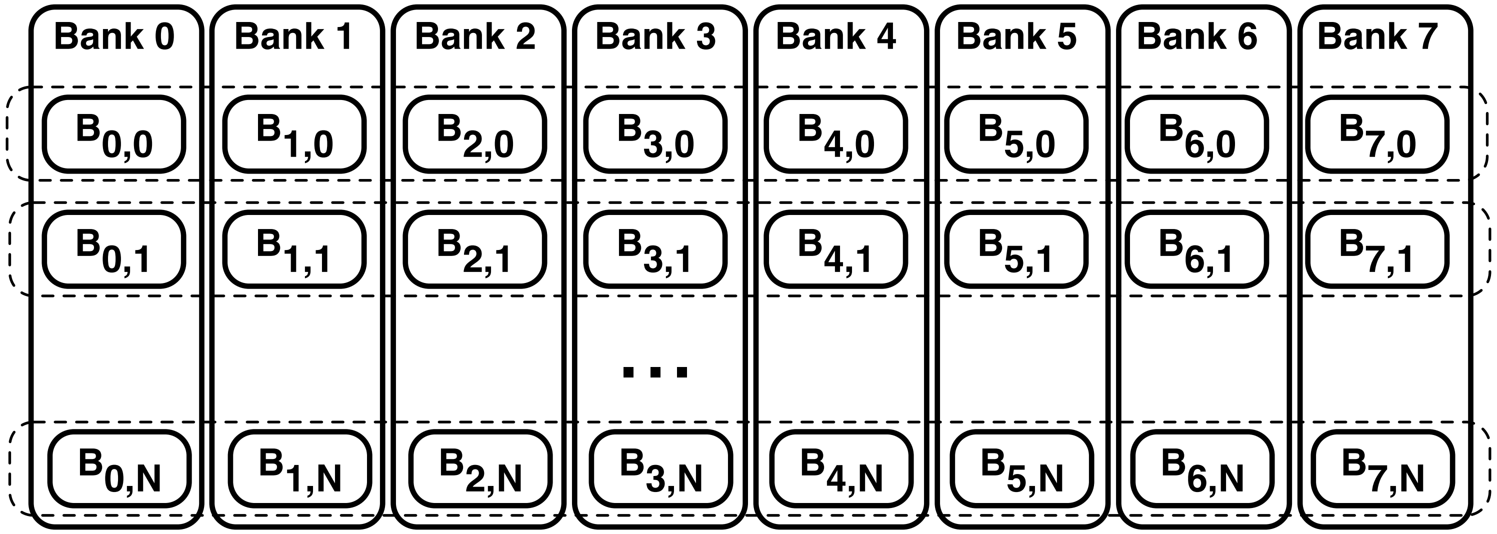
$D_0 = P_1$

$D_1 = P_2$

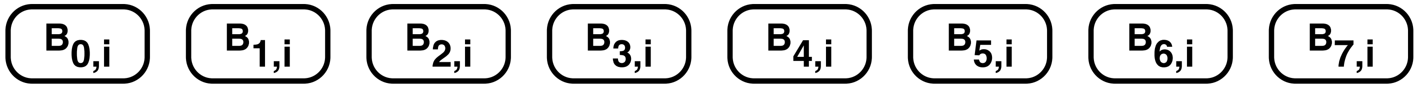
$D_2 = P_3$

Block Groups

One block from each bank is placed in a block group


Bank 0
Bank 1
Bank 2
Bank 3
Bank 4
Bank 5
Bank 6
Bank 7
$B_{0,0}$ $B_{1,0}$ $B_{2,0}$ $B_{3,0}$ $B_{4,0}$ $B_{5,0}$ $B_{6,0}$ $B_{7,0}$
$B_{0,1}$ $B_{1,1}$ $B_{2,1}$ $B_{3,1}$ $B_{4,1}$ $B_{5,1}$ $B_{6,1}$ $B_{7,1}$
...
$B_{0,N}$ $B_{1,N}$ $B_{2,N}$ $B_{3,N}$ $B_{4,N}$ $B_{5,N}$ $B_{6,N}$ $B_{7,N}$

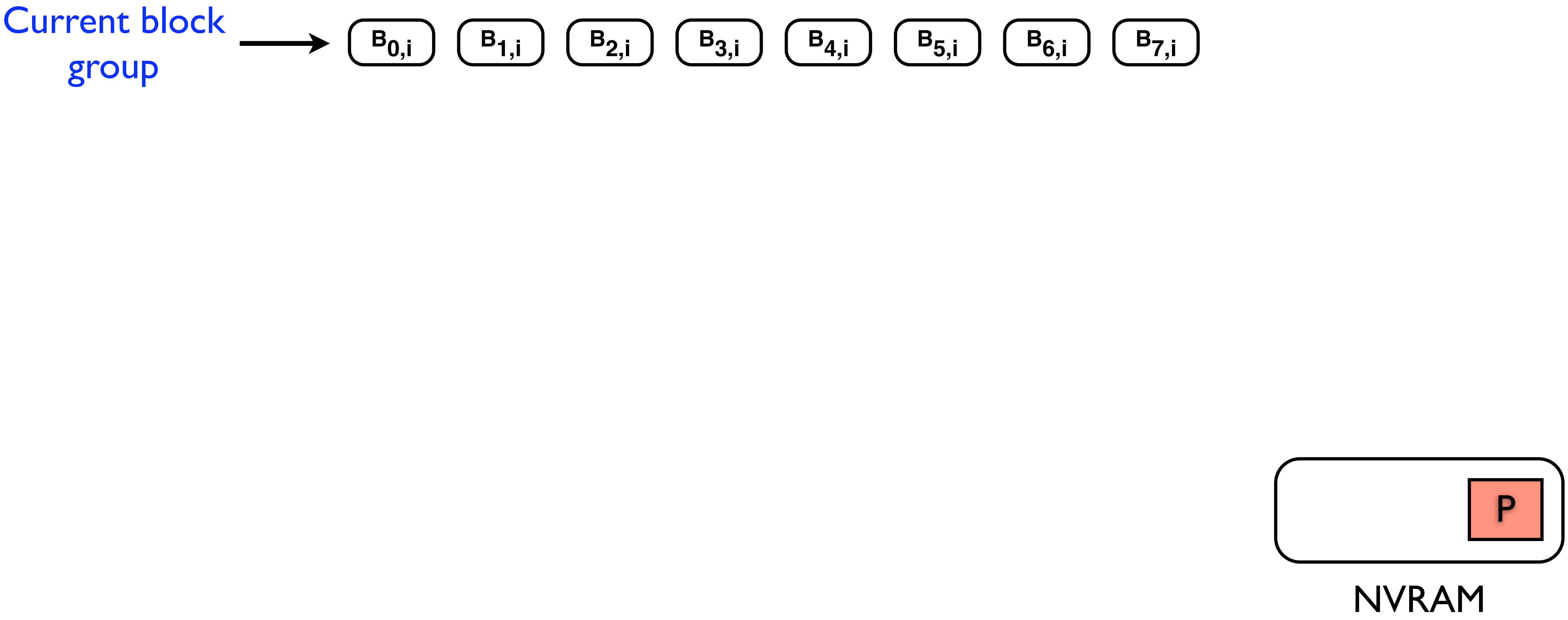
All writes go to current block group


$B_{0,i}$ $B_{1,i}$ $B_{2,i}$ $B_{3,i}$ $B_{4,i}$ $B_{5,i}$ $B_{6,i}$ $B_{7,i}$

$$\text{parity\_map} \leftarrow \{7 = 0 \oplus 1 \oplus 2 \oplus 3 \oplus 4 \oplus 5 \oplus 6\}$$
$$\text{data\_map} \leftarrow \{0 \rightarrow 7, 1 \rightarrow 7, 2 \rightarrow 7, 3 \rightarrow 7, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 7\}$$

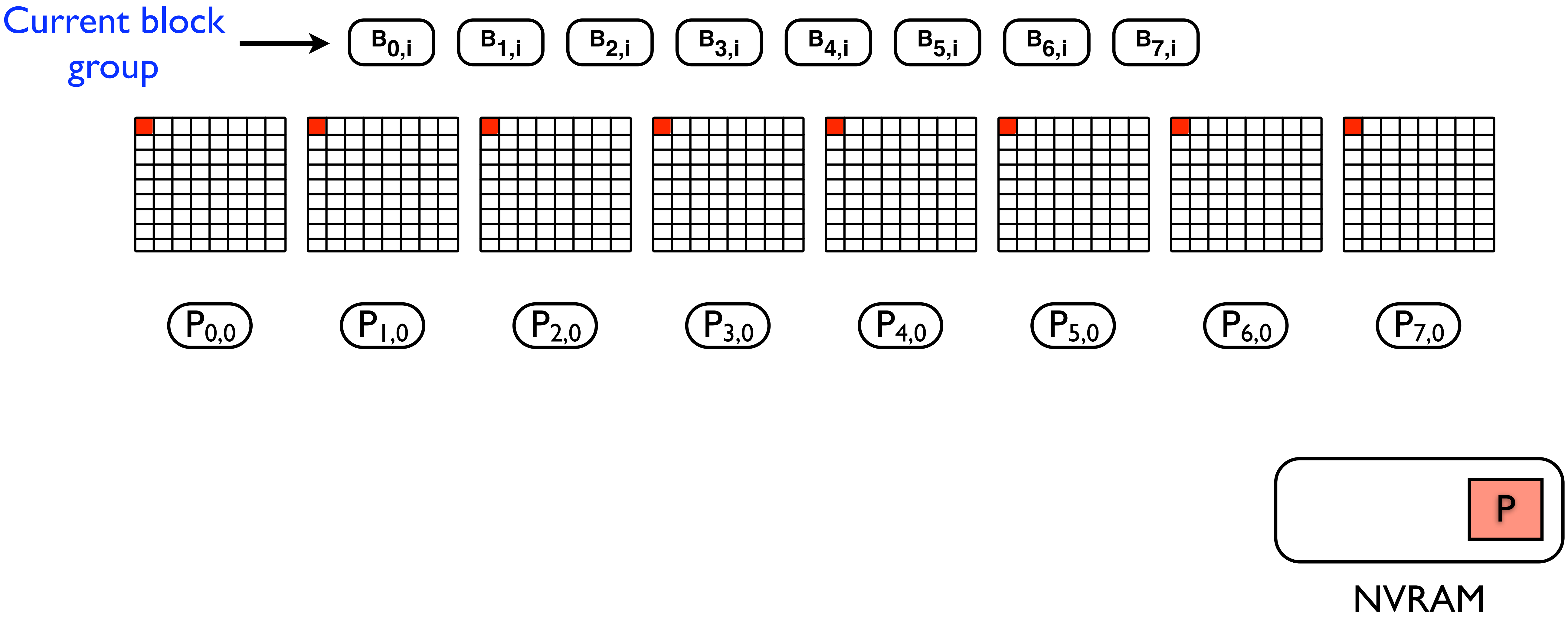
An erasure code instance is associated with a block group

Writing Data using Block Groups



$$\text{parity\_map} \leftarrow \{7 = 0 \oplus 1 \oplus 2 \oplus 3 \oplus 4 \oplus 5 \oplus 6\}$$
$$\text{data\_map} \leftarrow \{0 \rightarrow 7, 1 \rightarrow 7, 2 \rightarrow 7, 3 \rightarrow 7, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 7\}$$

Writing Data using Block Groups



Current block group

$B_{0,i}$ $B_{1,i}$ $B_{2,i}$ $B_{3,i}$ $B_{4,i}$ $B_{5,i}$ $B_{6,i}$ $B_{7,i}$

$P_{0,0}$ $P_{1,0}$ $P_{2,0}$ $P_{3,0}$ $P_{4,0}$ $P_{5,0}$ $P_{6,0}$ $P_{7,0}$

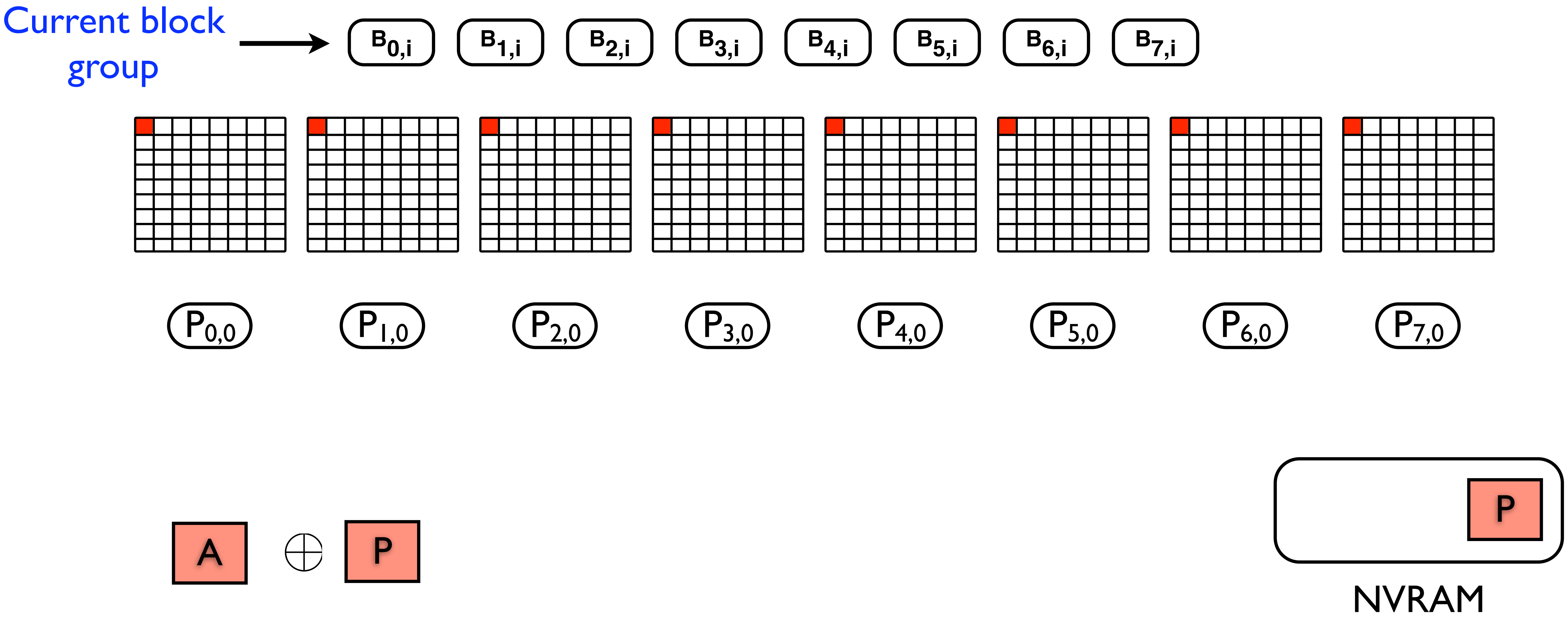
P

NVRAM

$$\text{parity\_map} \leftarrow \{7 = 0 \oplus 1 \oplus 2 \oplus 3 \oplus 4 \oplus 5 \oplus 6\}$$

$$\text{data\_map} \leftarrow \{0 \rightarrow 7, 1 \rightarrow 7, 2 \rightarrow 7, 3 \rightarrow 7, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 7\}$$

Writing Data using Block Groups

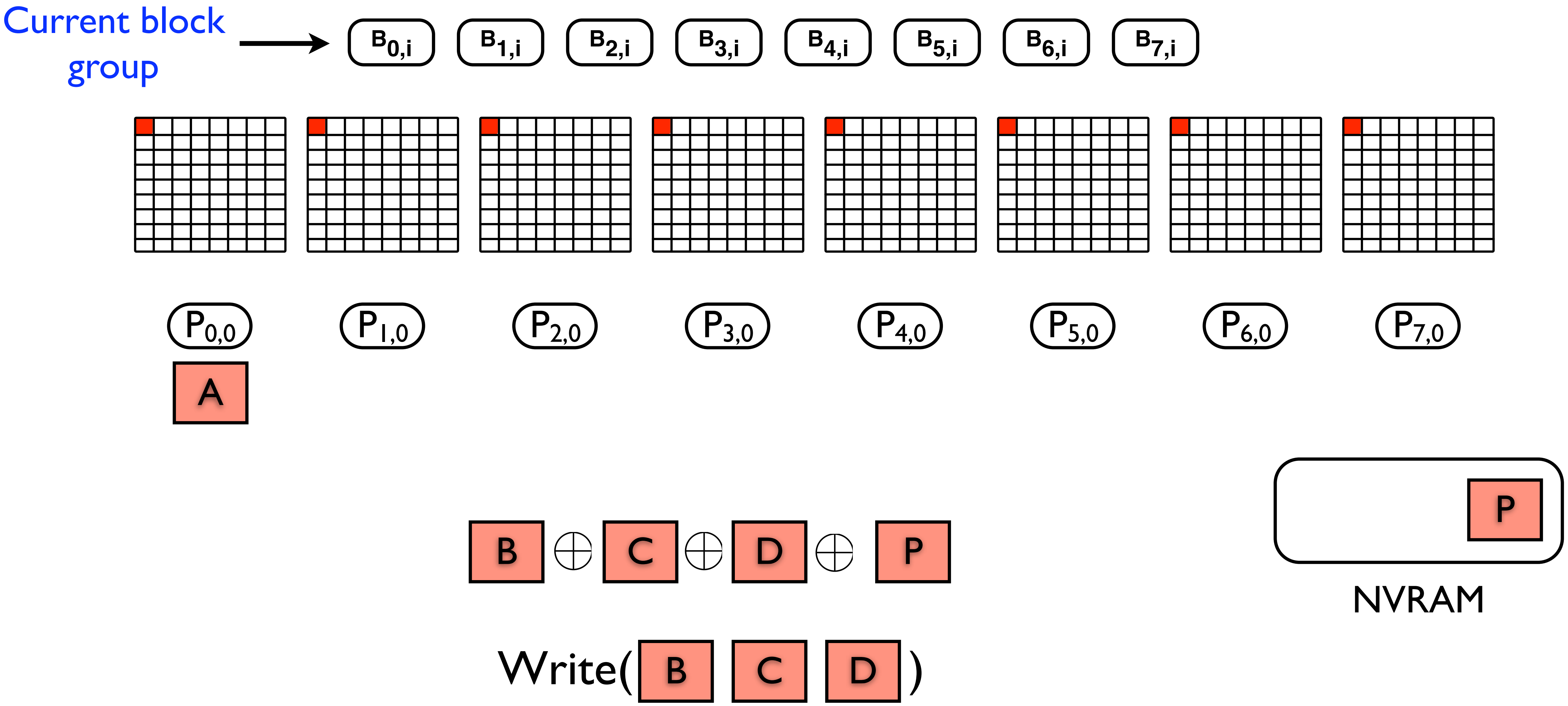


Write($\boxed{A}$)

$$\text{parity\_map} \leftarrow \{7 = 0 \oplus 1 \oplus 2 \oplus 3 \oplus 4 \oplus 5 \oplus 6\}$$

$$\text{data\_map} \leftarrow \{0 \rightarrow 7, 1 \rightarrow 7, 2 \rightarrow 7, 3 \rightarrow 7, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 7\}$$

Writing Data using Block Groups



Current block group

$B_{0,i}$ $B_{1,i}$ $B_{2,i}$ $B_{3,i}$ $B_{4,i}$ $B_{5,i}$ $B_{6,i}$ $B_{7,i}$

$P_{0,0}$ $P_{1,0}$ $P_{2,0}$ $P_{3,0}$ $P_{4,0}$ $P_{5,0}$ $P_{6,0}$ $P_{7,0}$

A

P

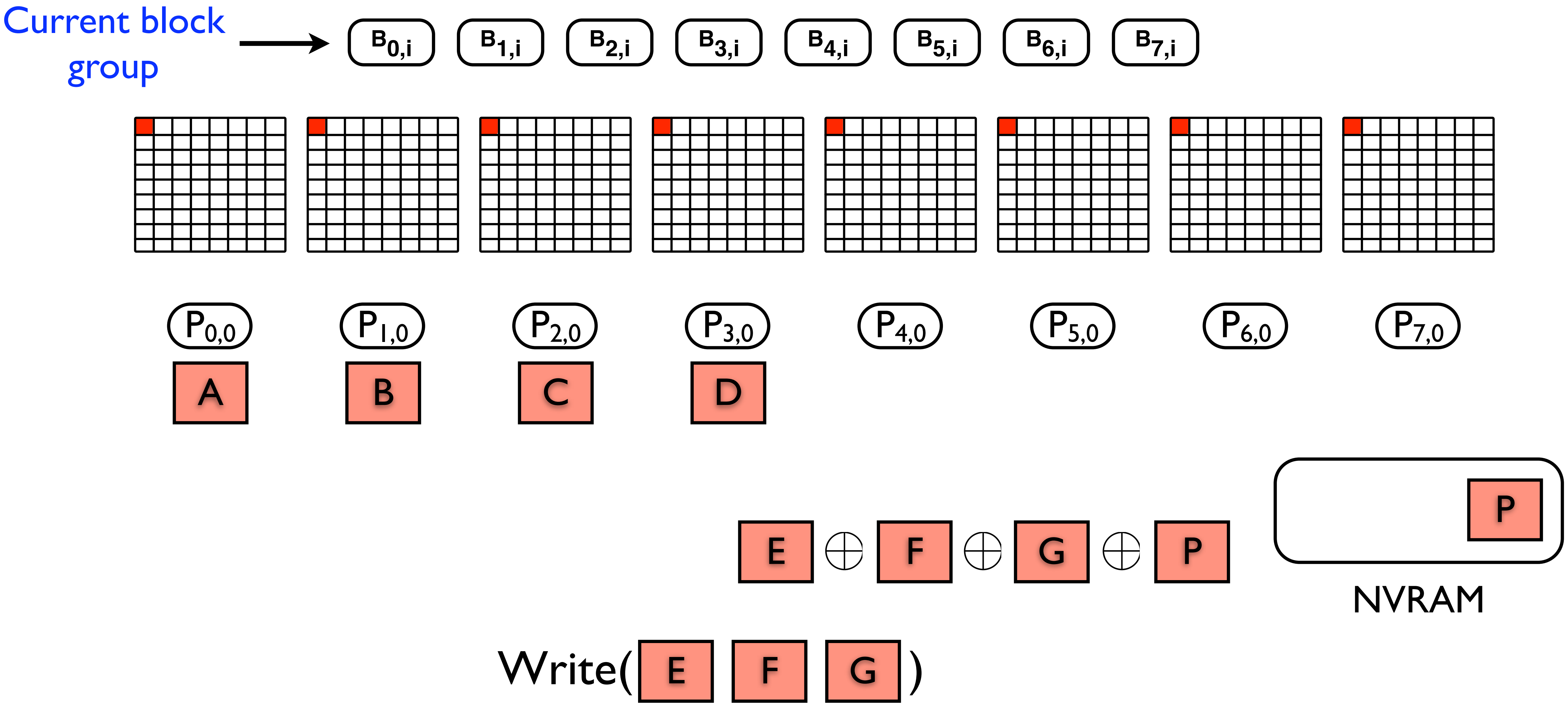
NVRAM

B $\oplus$ C $\oplus$ D $\oplus$ P

Write(B C D)

$$\text{parity\_map} \leftarrow \{7 = 0 \oplus 1 \oplus 2 \oplus 3 \oplus 4 \oplus 5 \oplus 6\}$$
$$\text{data\_map} \leftarrow \{0 \rightarrow 7, 1 \rightarrow 7, 2 \rightarrow 7, 3 \rightarrow 7, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 7\}$$

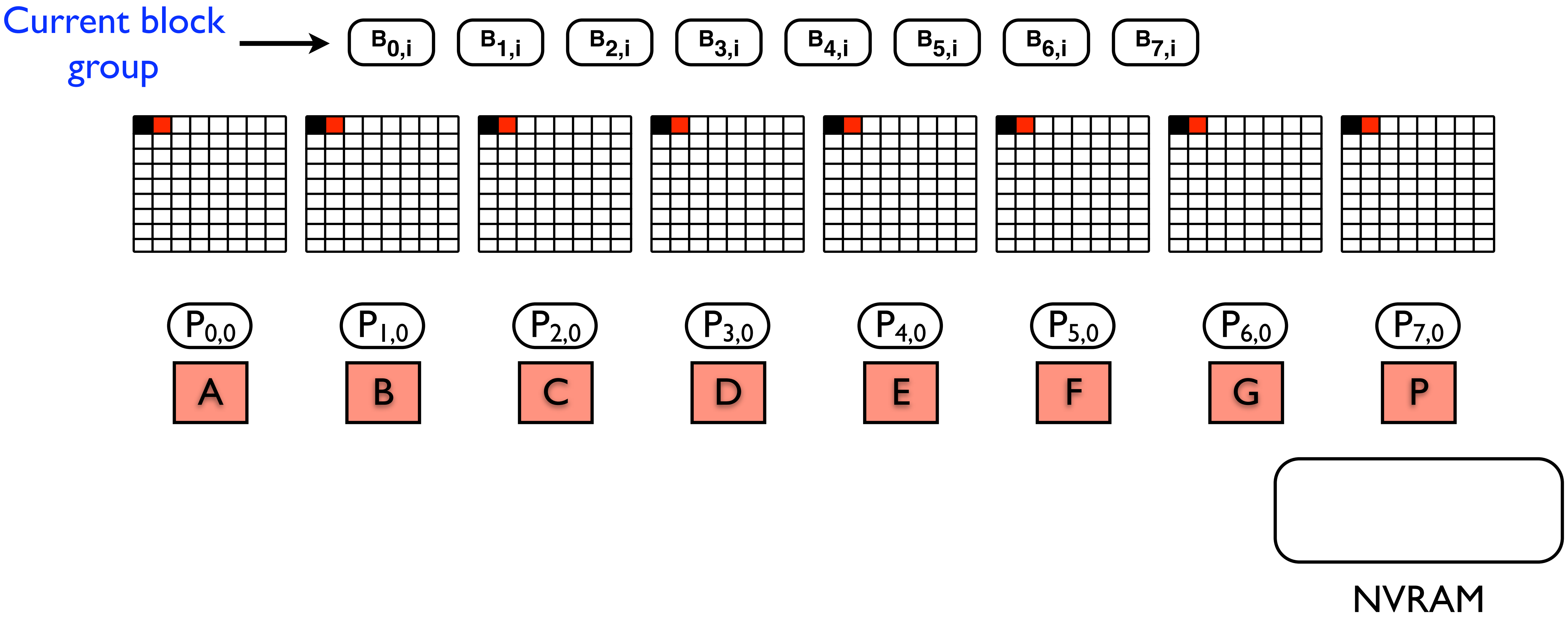
Writing Data using Block Groups



$$\text{parity\_map} \leftarrow \{7 = 0 \oplus 1 \oplus 2 \oplus 3 \oplus 4 \oplus 5 \oplus 6\}$$

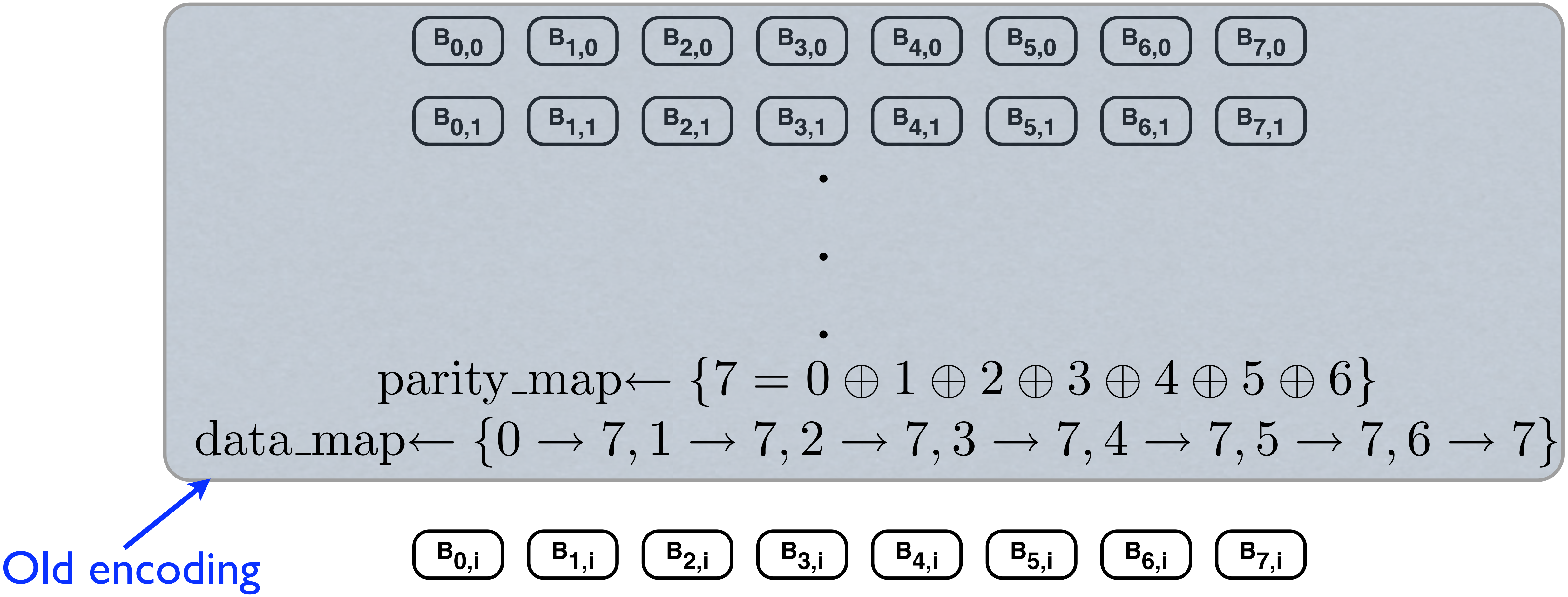
$$\text{data\_map} \leftarrow \{0 \rightarrow 7, 1 \rightarrow 7, 2 \rightarrow 7, 3 \rightarrow 7, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 7\}$$

Writing Data using Block Groups



$$\text{parity\_map} \leftarrow \{7 = 0 \oplus 1 \oplus 2 \oplus 3 \oplus 4 \oplus 5 \oplus 6\}$$
$$\text{data\_map} \leftarrow \{0 \rightarrow 7, 1 \rightarrow 7, 2 \rightarrow 7, 3 \rightarrow 7, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 7\}$$

Graceful Degradation



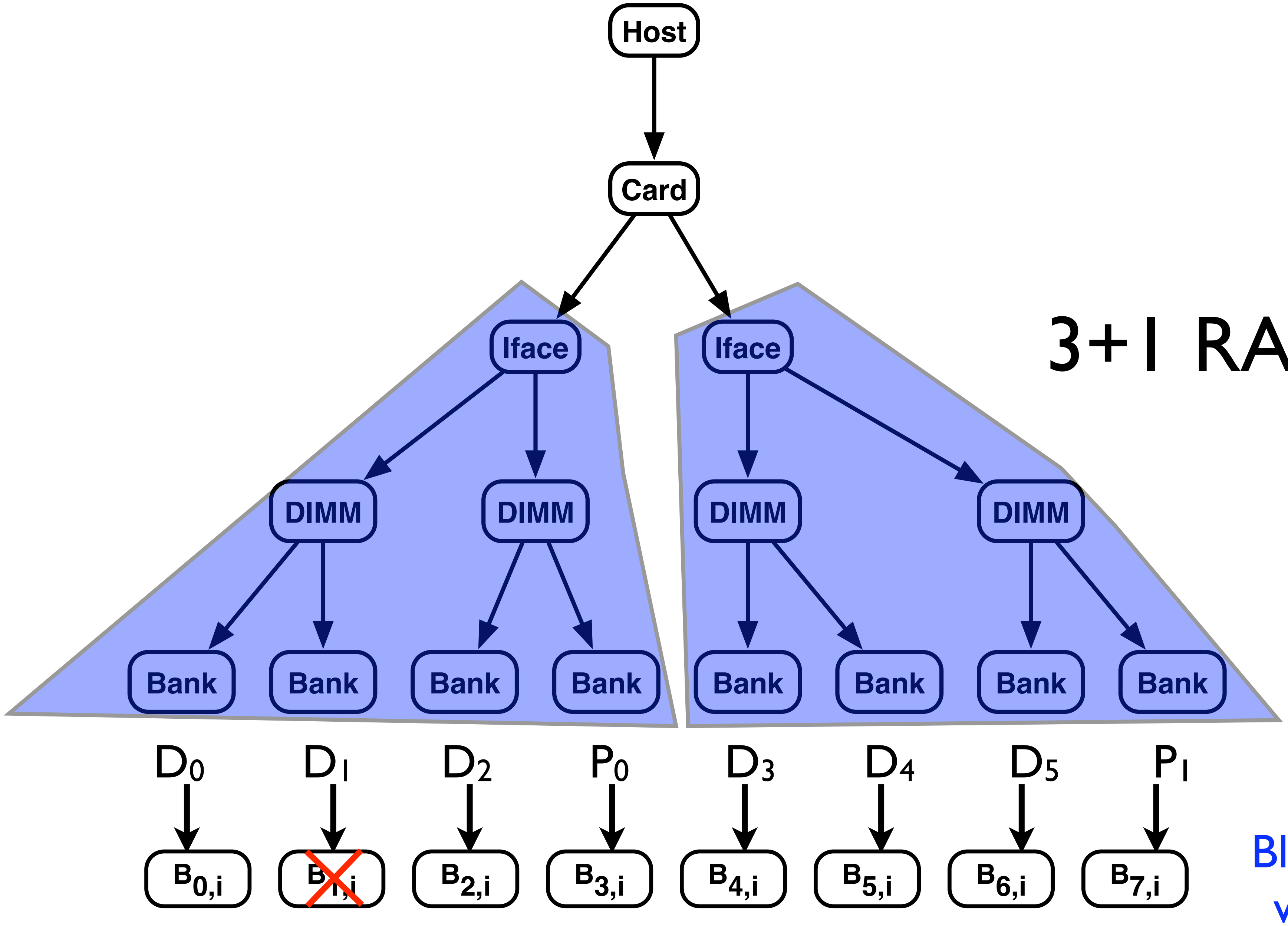
$B_{0,0}$ $B_{1,0}$ $B_{2,0}$ $B_{3,0}$ $B_{4,0}$ $B_{5,0}$ $B_{6,0}$ $B_{7,0}$
$B_{0,1}$ $B_{1,1}$ $B_{2,1}$ $B_{3,1}$ $B_{4,1}$ $B_{5,1}$ $B_{6,1}$ $B_{7,1}$
Old encoding
$B_{0,i}$ $B_{1,i}$ $B_{2,i}$ $B_{3,i}$ $B_{4,i}$ $B_{5,i}$ $B_{6,i}$ $B_{7,i}$

$$\text{parity\_map} \leftarrow \{7 = 0 \oplus 1 \oplus 2 \oplus 3 \oplus 4 \oplus 5 \oplus 6\}$$
$$\text{data\_map} \leftarrow \{0 \rightarrow 7, 1 \rightarrow 7, 2 \rightarrow 7, 3 \rightarrow 7, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 7\}$$

$$\text{parity\_map} \leftarrow \{3 = 0 \oplus 1 \oplus 2, 7 = 4 \oplus 5 \oplus 6\}$$
$$\text{data\_map} \leftarrow \{0 \rightarrow 3, 1 \rightarrow 3, 2 \rightarrow 3, 4 \rightarrow 7, 5 \rightarrow 7, 6 \rightarrow 7\}$$

- Block groups allow us to change the encoding
- Two encodings: current and old
- All block groups with old encoding are more likely to be cleaned

Recovering Page and Block Errors



3+1 RAID4

Block Group with Error

$$D_0 \oplus D_1 \oplus D_2 = P_0$$
$$D_3 \oplus D_4 \oplus D_5 = P_1$$

$$B_{0,i} \oplus B_{2,i} \oplus B_{3,i} \rightarrow R_{1,i}$$

Write to current block group

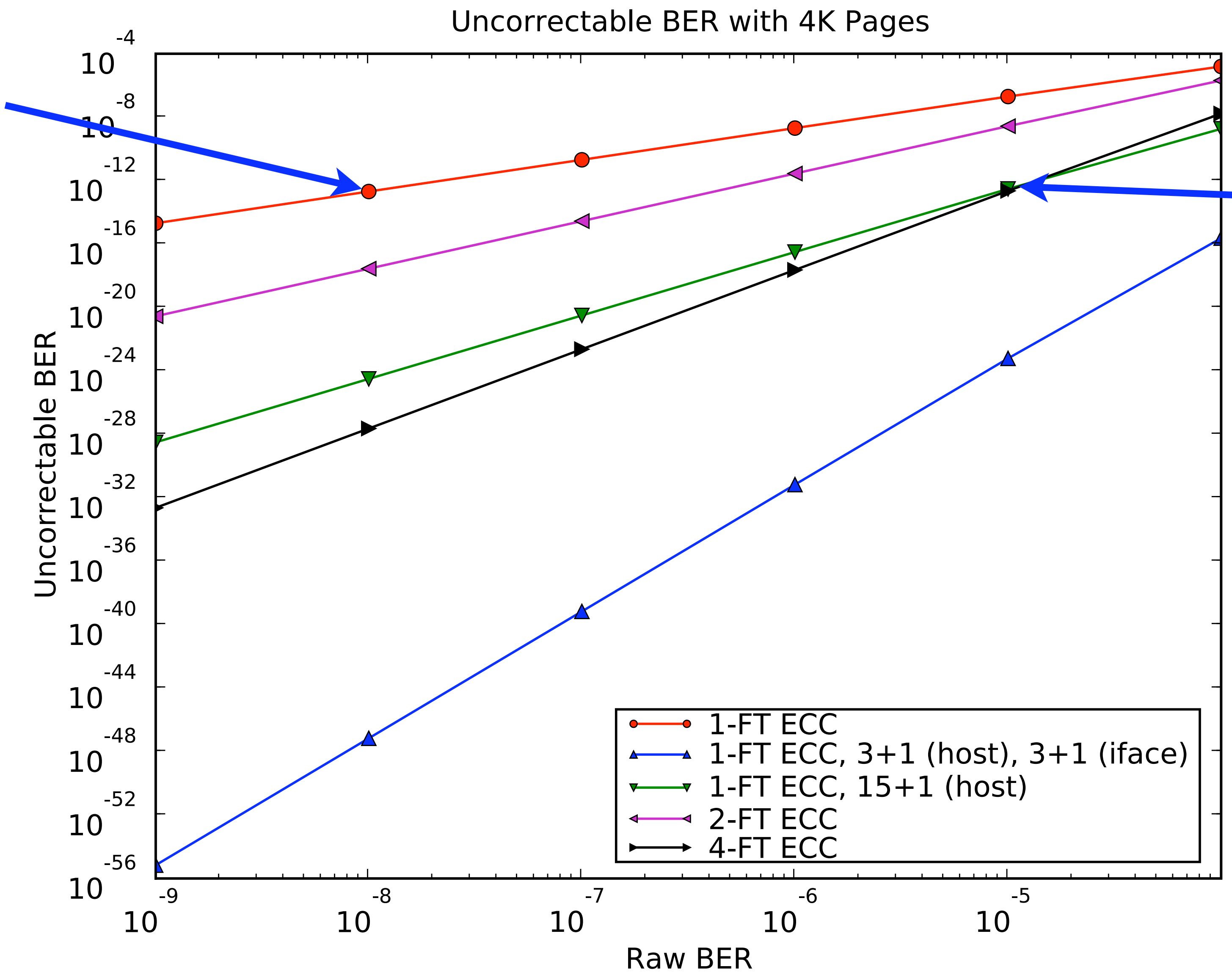
Failover

- Page and block errors
  - Write data to current block group
  - Try to use page or block again once block group is cleaned
  - If we get a write error, then mark page or block as bad
- Can deal with bank errors with spares
- Spare-less component errors
  - Try to reconstruct data
  - Mark banks under failed components as bad
  - Reform block groups without bad banks

Performance

- Based on flash simulator from NetApp
  - 4 DMA channels/card (Libra card has 2)
  - 2 interfaces/card
  - 2 DIMMs/interface
  - 2 banks/DIMM (16 total banks)
  - 64 blocks/bank
  - 64 pages/block
  - 1.2 ms (erase), 0.2 ms (prog), 0.025 ms (read)
- 2 cards connected to a host
- All functionality resides in driver on the host
- Evaluate write performance/reliability
  - No erasure code
  - 15+1 (across 16 banks)
  - 3+1 host-level, 3+1 iface level (across 16 banks)

Erasure Coding and Reliability

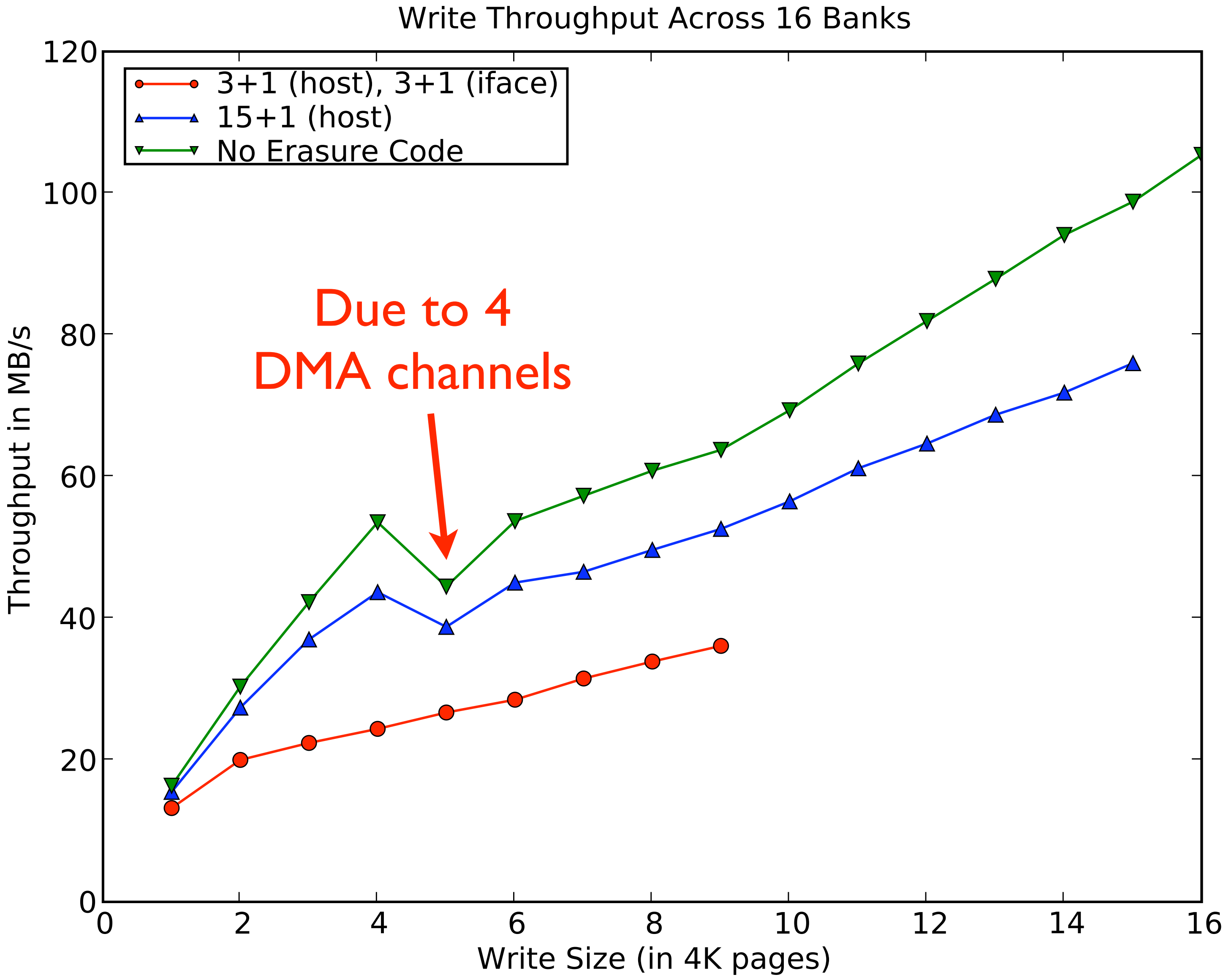


Upcode to 15+1

Upcode to multi-level scheme

- Increasing page-level ECC decreases RBER
  - May not be possible on-the-fly
- Easier to keep page-level ECC and change erasure code
  - Up-code when expected RBER gets too high

Performance



- Write size max is number of data pages in a stripe
- Rebuild performance
  - 3.41 MB/s (15+1) , 16.52 MB/s (3+1)
- Current encoder does not compute full stripe parity

Other Challenges and Concerns

- Cleaning
- Wear leveling with block groups
- Bad block management
- Reliability and performance after failover
- Smart write policies
  - Coalesce page updates into single parity computation
  - Exploit parallelism in the hierarchy

Questions?