STAIR Codes: A General Family of Erasure Codes for Tolerating Device and Sector Failures in Practical Storage Systems

Mingqiang Li and Patrick P. C. Lee

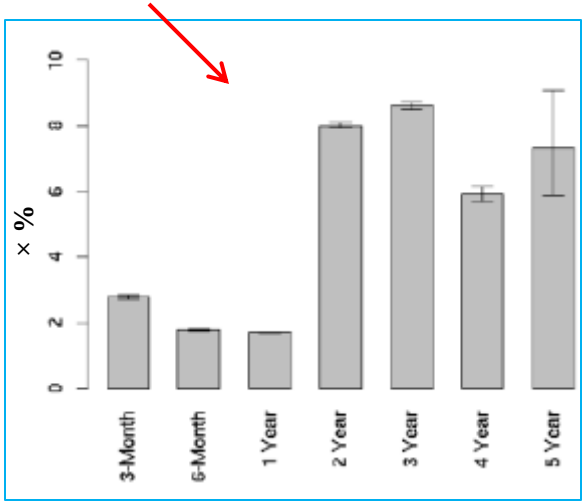
The Chinese University of Hong Kong

FAST '14

Device and Sector Failures

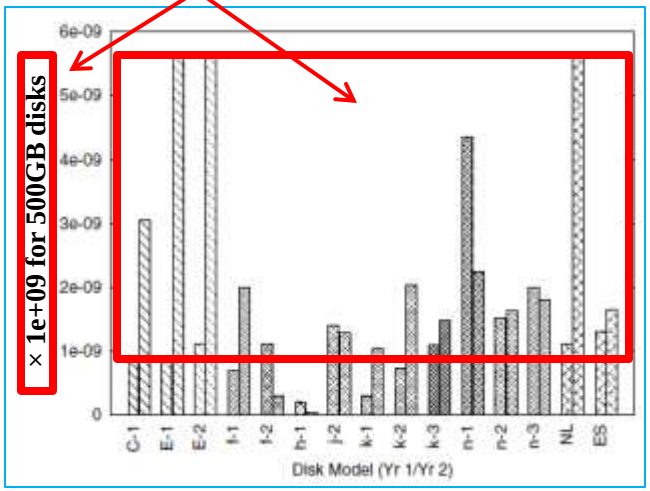
- Storage systems susceptible to both device and sector failures
  - *Device failure*: data loss in an entire device
  - *Sector failure*: data loss in a sector

(a) Annual disk failure rate: 1~10%



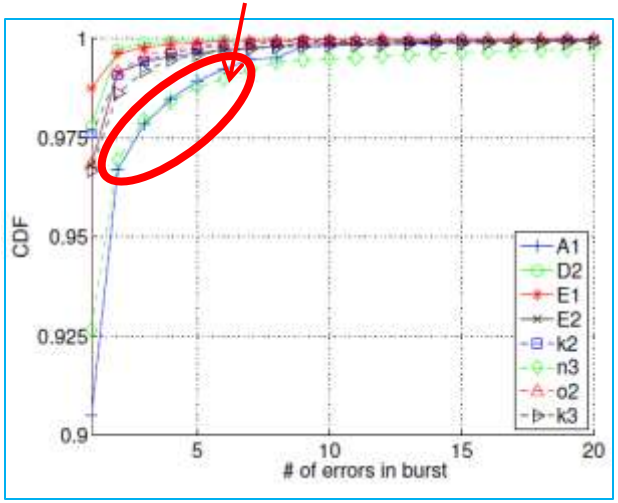
Annual disk failure rate
[Pinheiro et al., FAST'07]

(b) Sector failures can be more frequent than disk failures



Annual sector failure rate
[Bairavasundaram et al., SIGMETRICS '07]

(c) Sector failure bursts can be long (> 5)



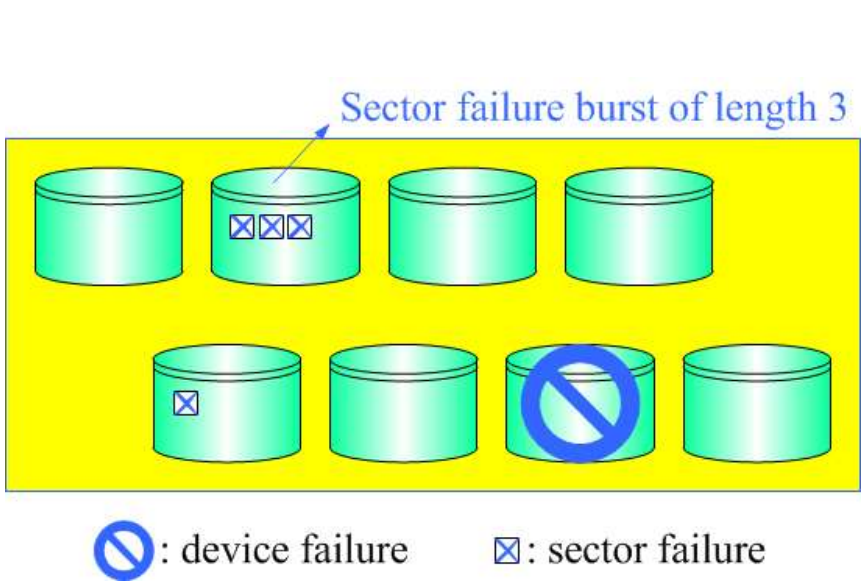
Burstiness of sector failures
[Schroeder et al., FAST '10]

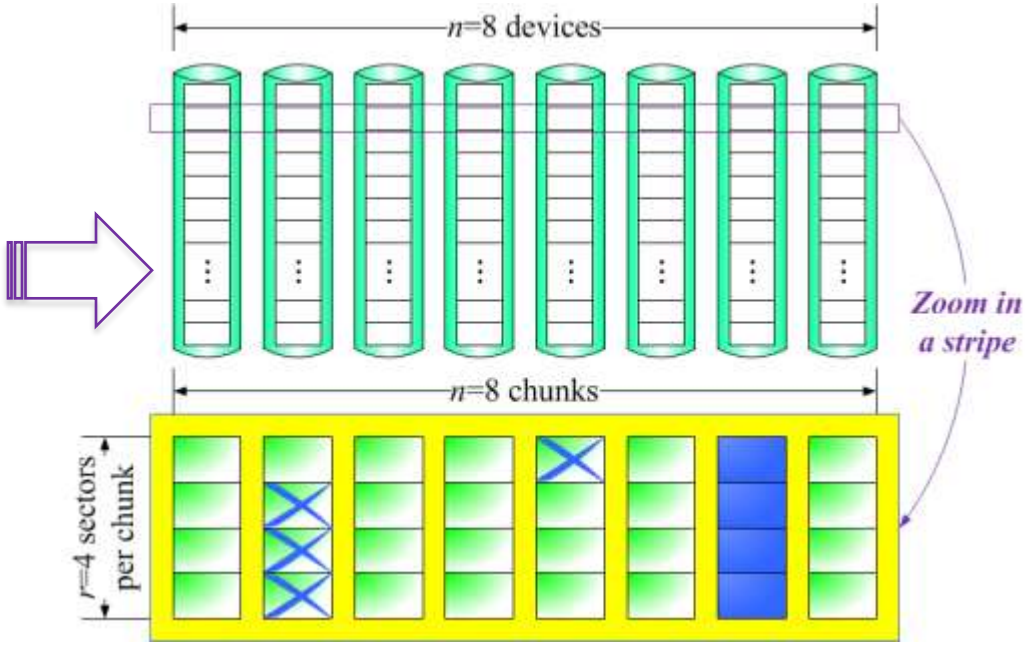
Erasure Coding

- **Erasure coding**: adds redundancy to data
- ***(N,K)* systematic MDS codes**
  - Encodes K data pieces to create N-K *parity* pieces
  - Stripes the N pieces across disks
  - Any K out of N pieces can recover original K data pieces and the N-K parity pieces → fault tolerance
- Three erasure coding schemes:
  - Traditional RAID and erasure codes (e.g., Reed-Solomon codes)
  - Intra-Device Redundancy (IDR)
  - Sector-Disk (SD) codes

Mixed Failure Scenario

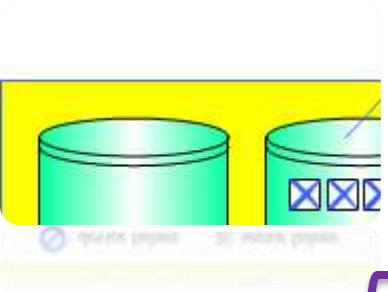
- Consider an example failure scenario with
  - $m=1$ entirely failed device, and
  - $m'=2$ partially failed devices with **1** and **3** sector failures

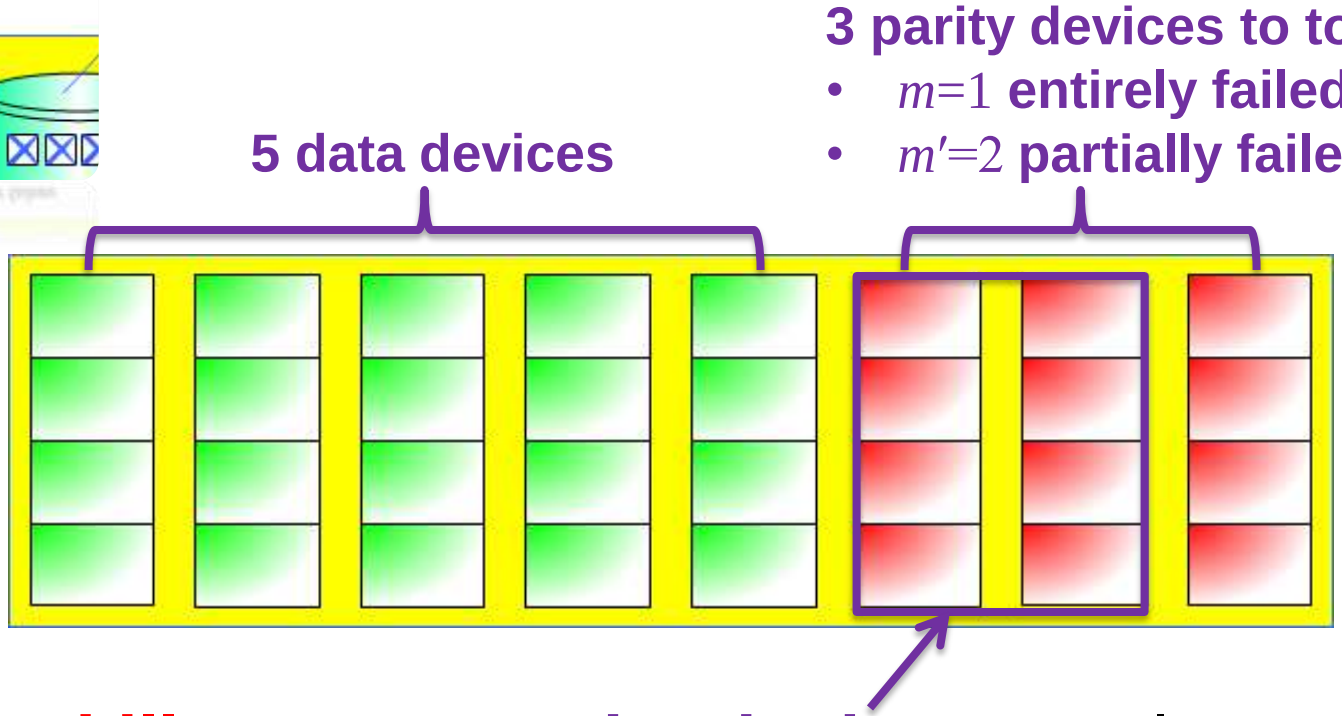




Question: ***How can we efficiently tolerate such a mixed failure scenario via erasure coding?***

Traditional RAID and Erasure Codes

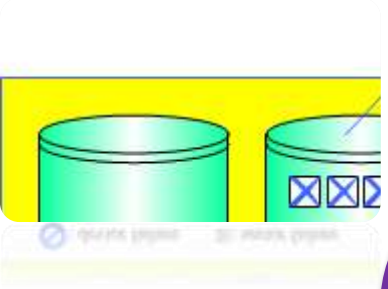


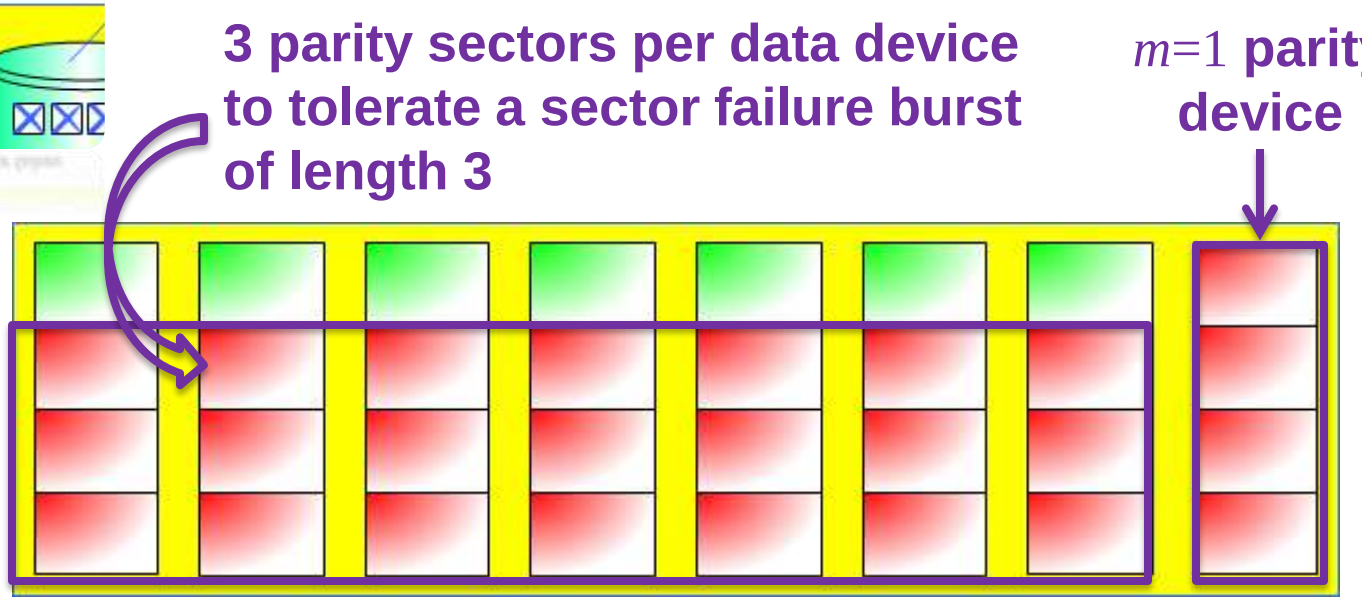


- **Overkill** to use **2 parity devices** to tolerate m'=2 partially failed devices
  - Device-level tolerance only

Intra-Device Redundancy (IDR)

[*Dholakia et al., TOS 2008*]



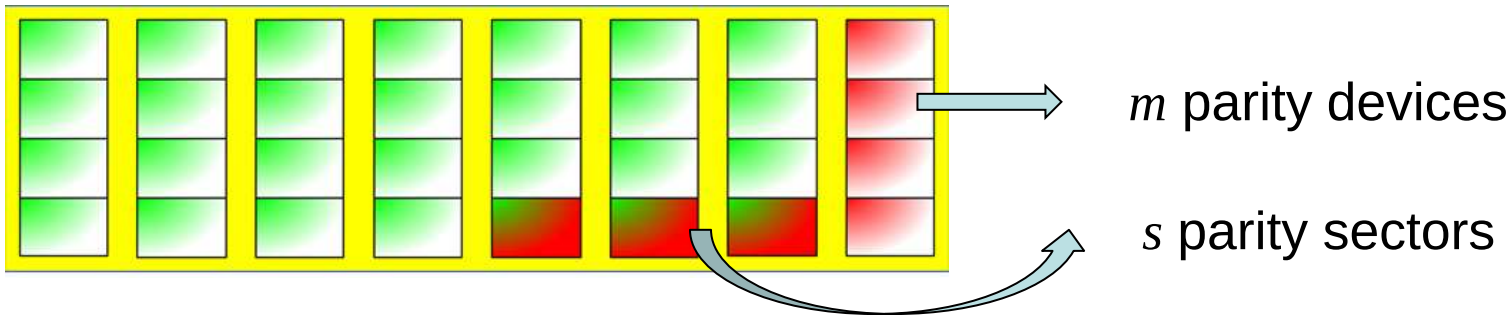


- Still **overkill** to add parity sectors per data device

Sector-Disk (SD) Codes

[*Plank et al., FAST '13, TOS'14*]

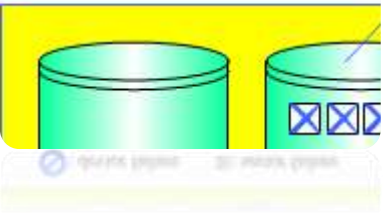
- Simultaneously tolerate
  - m entirely failed devices
  - s failed sectors (per stripe) in partially failed devices
- Construction currently limited to $s \leq 3$

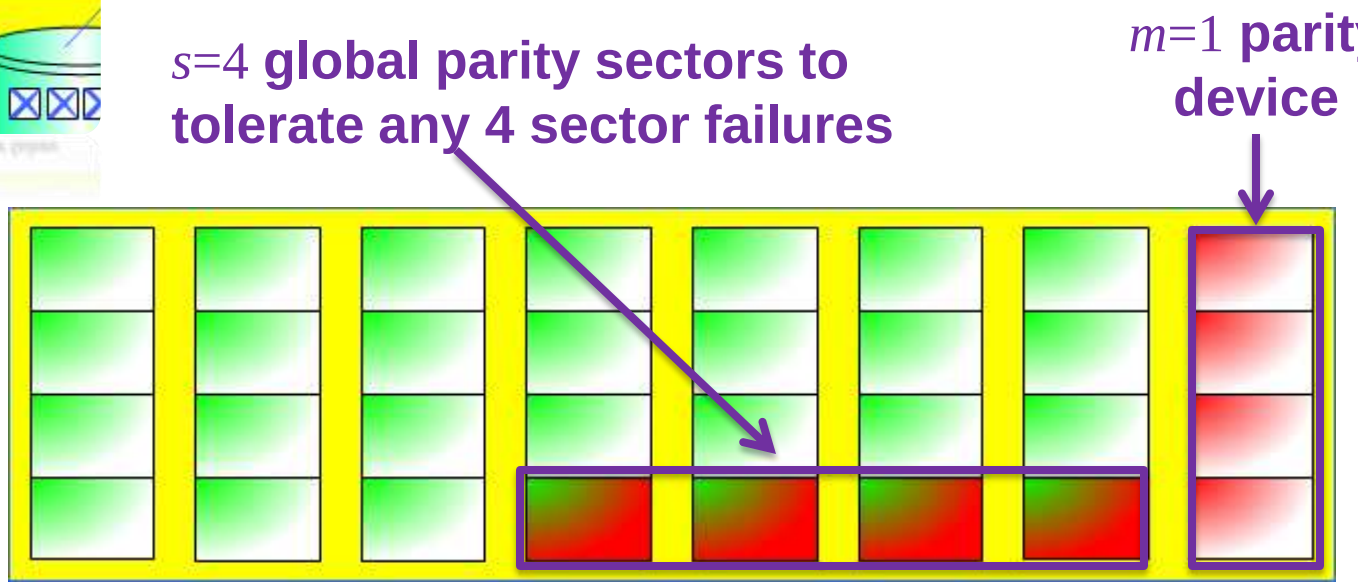


- How to tolerate our mixed failure scenario?
  - $m=1$ entirely failed device, and
  - $m'=2$ partially failed devices with **1** and **3** sector failures

Sector-Disk (SD) Codes

[*Plank et al., FAST '13, TOS'14*]

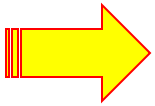




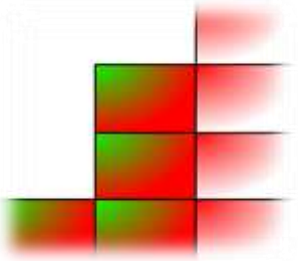
- Such an SD code is **unavailable**

Our Work

- Construct a **general, space-efficient** family of erasure codes to tolerate both device and sector failures
  - **a) General**: without any restriction on
    - *size of a storage array,*
    - *number of tolerable device failures, or*
    - *number of tolerable sector failures*
  - **b) Space-efficient**:
    - number of global parity sectors = number of sector failures (like SD codes)



STAIR Codes



Key Ideas of STAIR Codes

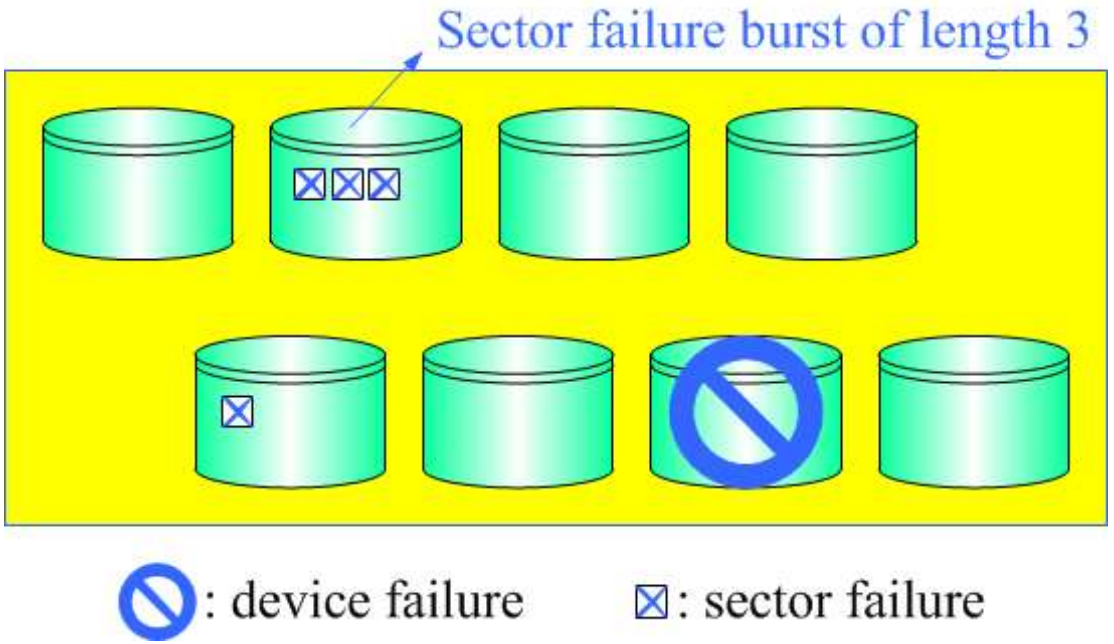
- Sector failure coverage vector $\mathbf{e}$
  - Defines a pattern of how sector failures occur, rather than how many sector failures would occur
- Code structure based on two encoding phases
  - Each phase builds on an MDS code
- Two encoding methods: upstairs and downstairs encoding
  - Reuse computed parity results in encoding
  - Provide complementary performance gains

Sector Failure Coverage Vector

- SD codes define s
  - Tolerate any combination of s sector failures per stripe
  - Currently limited to $s \leq 3$

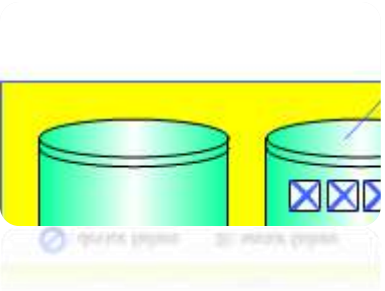
- STAIR codes define sector failure coverage vector
  $\mathbf{e} = (e_0, e_1, e_2, \ldots, e_{m'-1})$
  - Bounds # of partially failed devices m'
  - Bounds # of sector failures per device e_l $(0 \leq l \leq m' - 1)$
    - $\sum e_l = s$
  - Rationale: sector failures come in small bursts
    → Can define small m' and reasonable size e_l for bursts

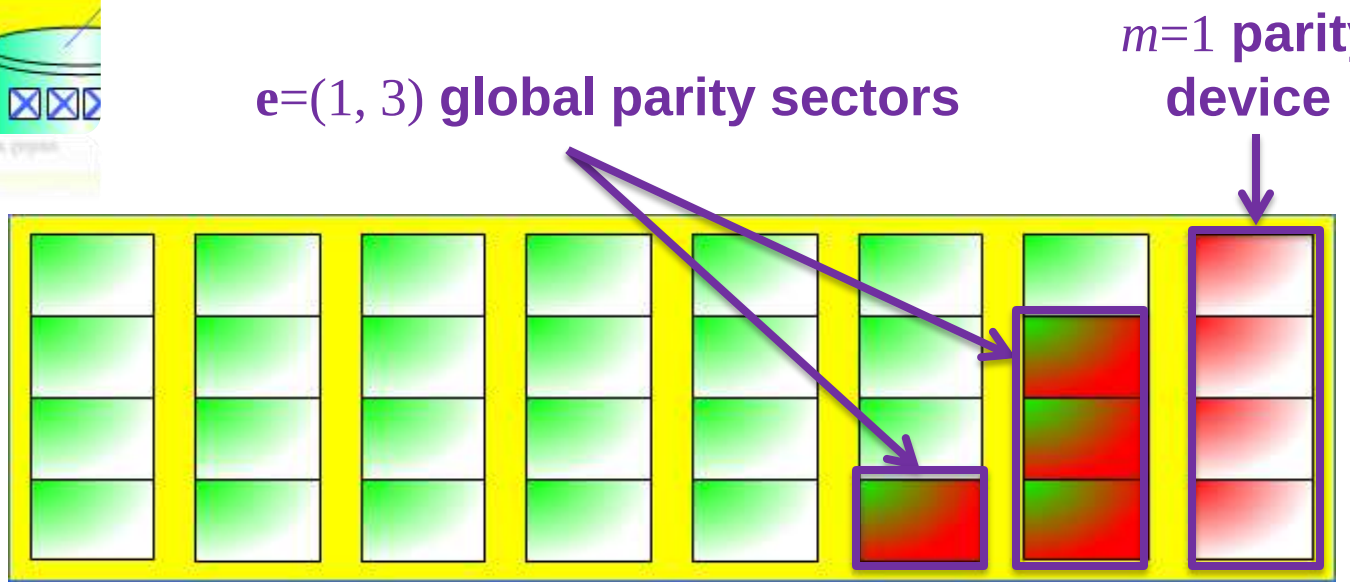
Sector Failure Coverage Vector



- Set **e**=(1, 3):
  - At most **2** devices (aside entirely failed devices) have sector failures
    - One device has at most **3** sector failures, and
    - Another one has at most **1** sector failure

Parity Layout

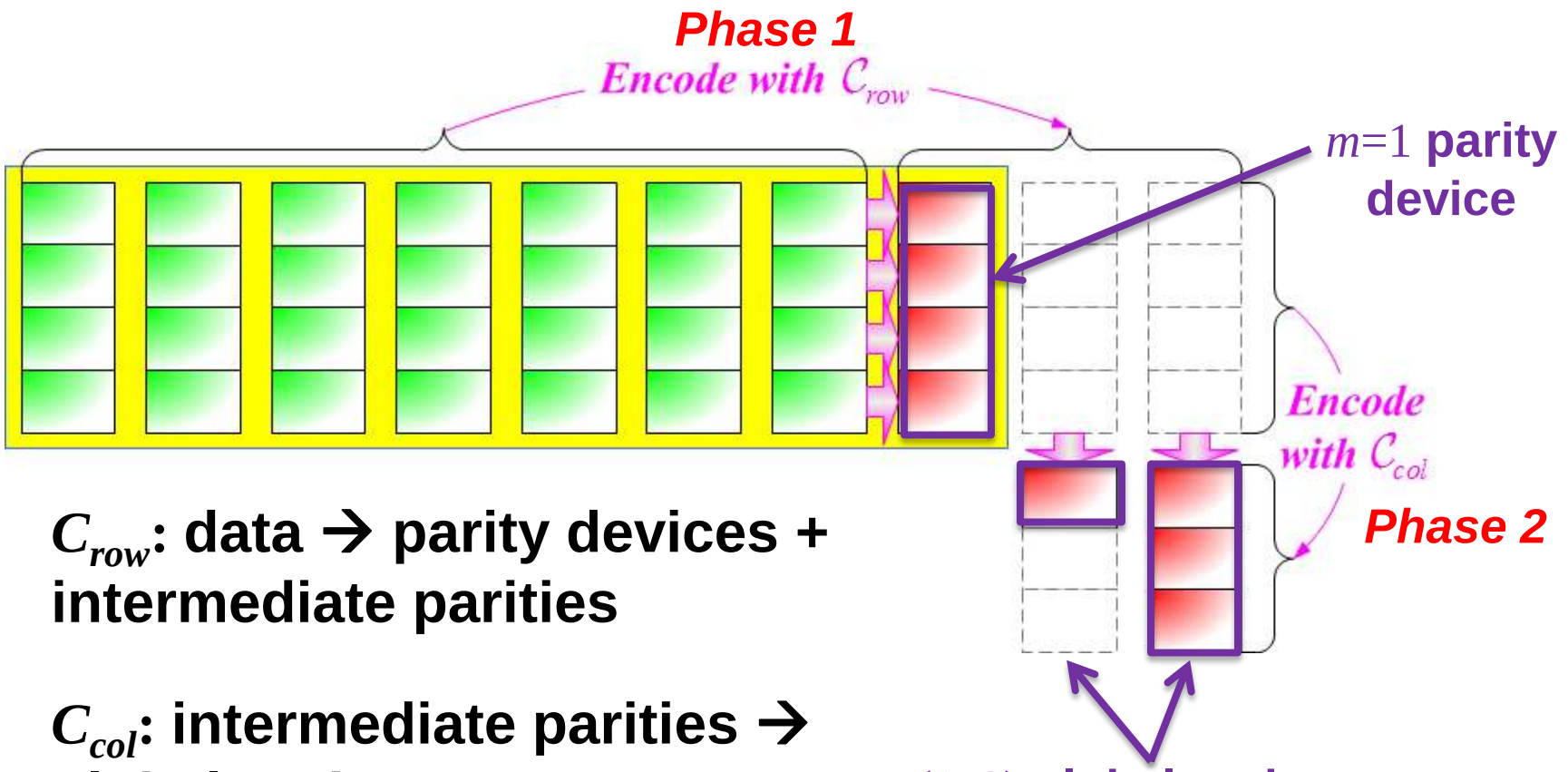




- **Q:** How to generate the **e**=(1, 3) global parity sectors and the m=1 parity device?

- **A:** Use two MDS codes C_{row} and C_{col}

Two Encoding Phases

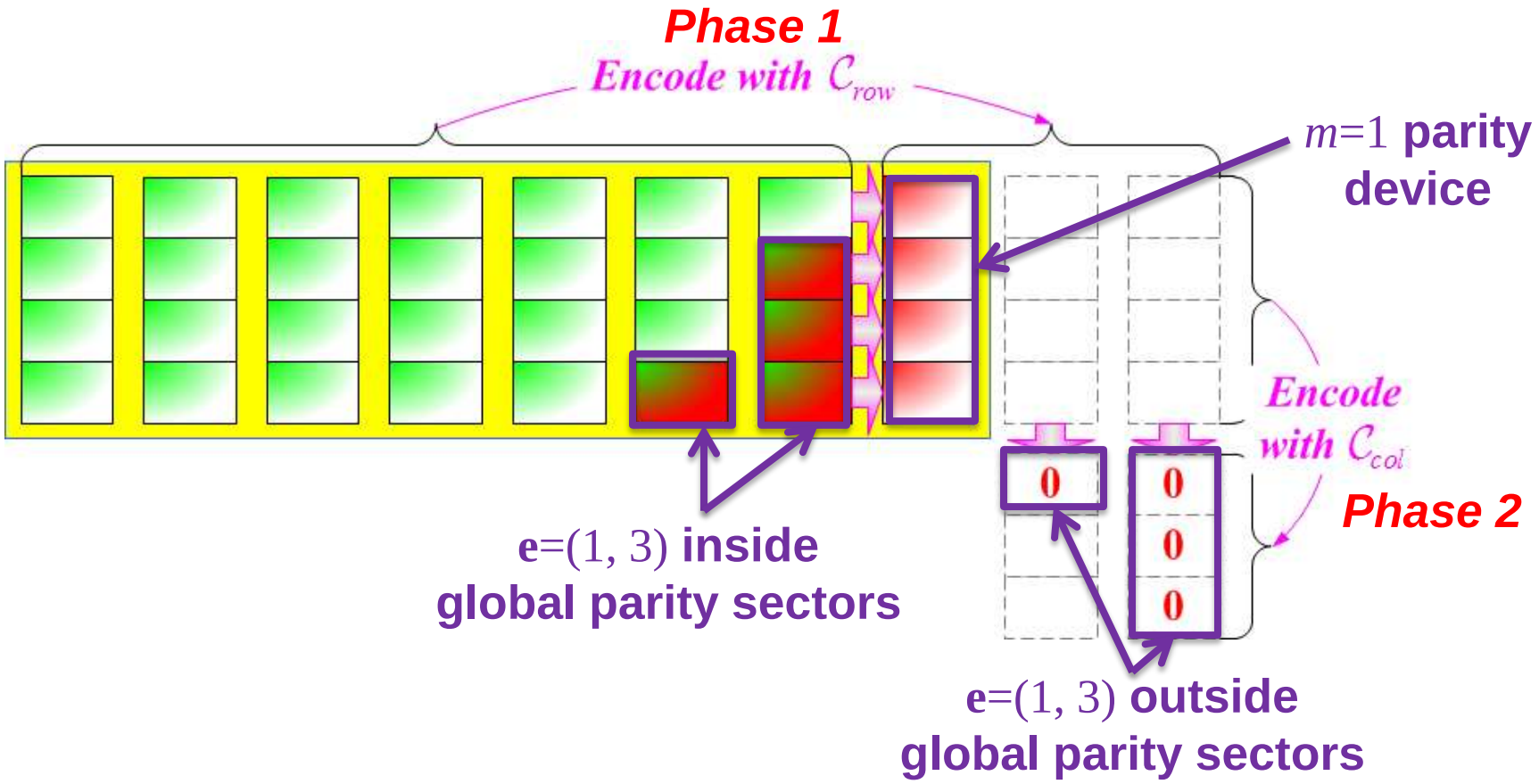


C_{row}: data → parity devices + intermediate parities

C_{col}: intermediate parities → global parity sectors

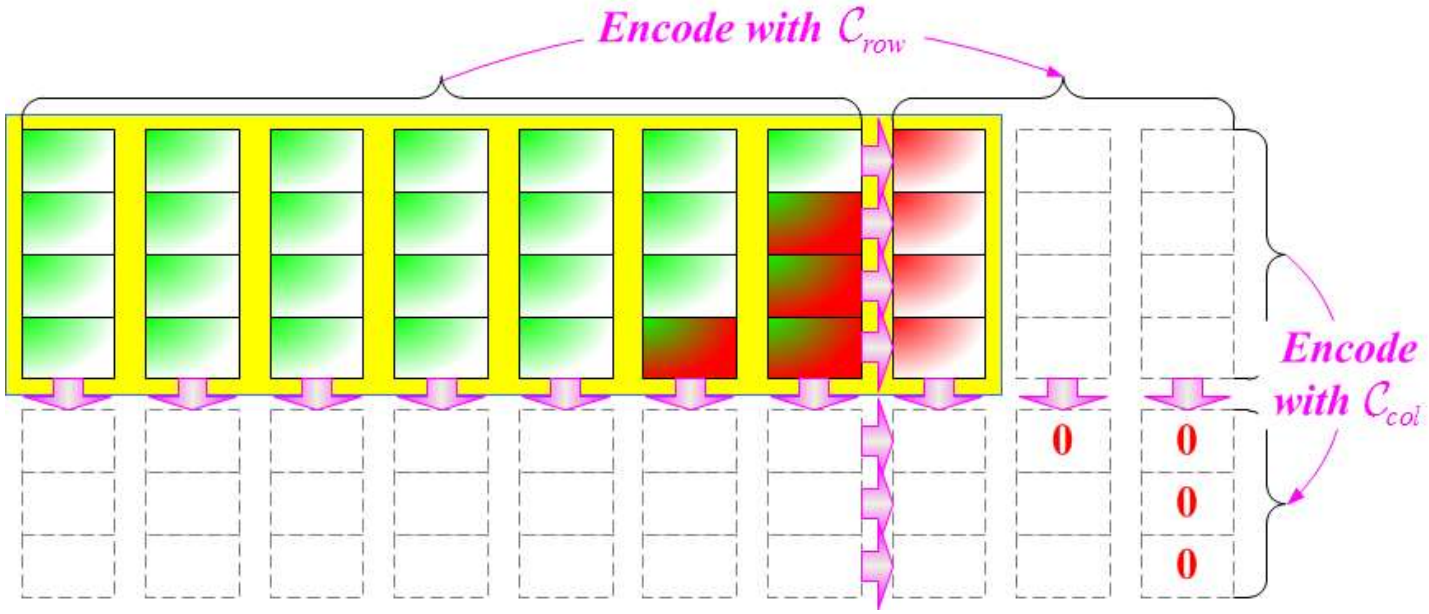
***Q:** How to keep the global parity sectors inside a stripe?*

Two Encoding Phases



- ***A:*** set outside global parity sectors as **zeroes**; reconstruct inside global parity sectors

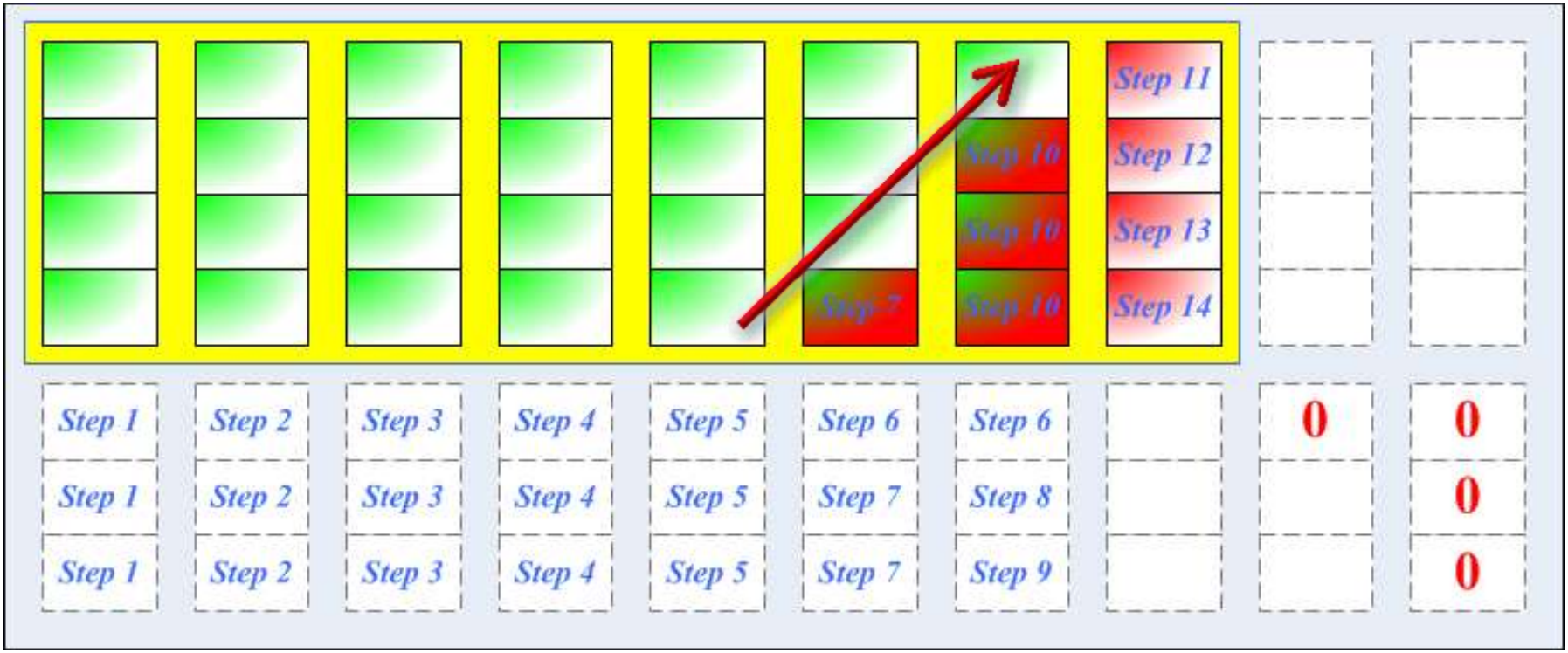
Augmented Rows



- Q: How do we compute inside parity sectors?
  - A: Augment a stripe
- Encode each column with C_{col} to form **augmented rows**
  - Generate virtual parities in augmented rows
- Each augmented row is a codeword of C_{row}

Upstairs Encoding

- Idea: Generate parities in upstairs direction



- Can be generalized as **upstairs decoding** for recovering failures

Upstairs Encoding

- Detailed steps:

C_{row}: (10,7) code **C_{col}: (7,4) code**

Upstairs Encoding

- Detailed steps:

C_{row}: (10,7) code C_{col}: (7,4) code

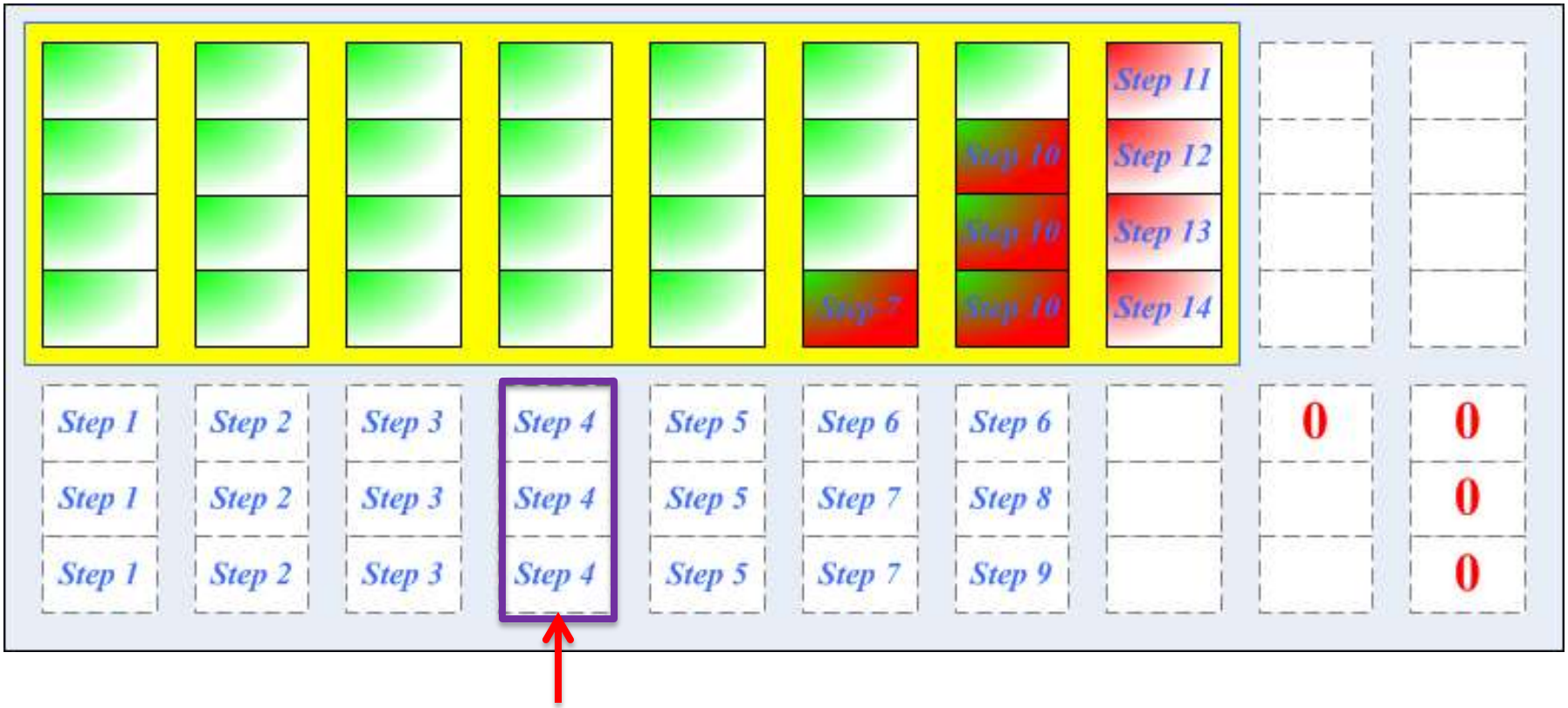
Upstairs Encoding

- Detailed steps:

C_{row}: (10,7) code C_{col}: (7,4) code

Upstairs Encoding

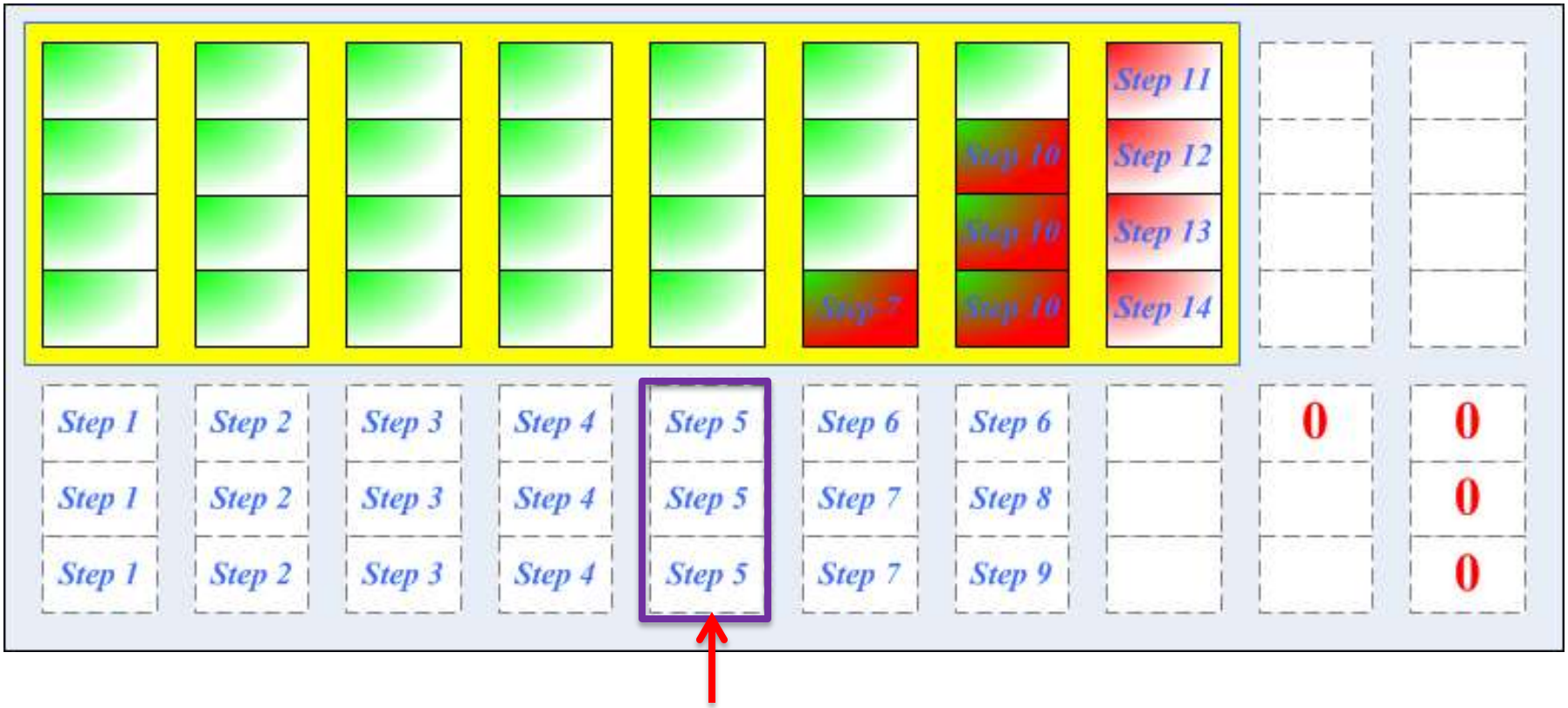
- Detailed steps:



C_{row}: (10,7) code C_{col}: (7,4) code

Upstairs Encoding

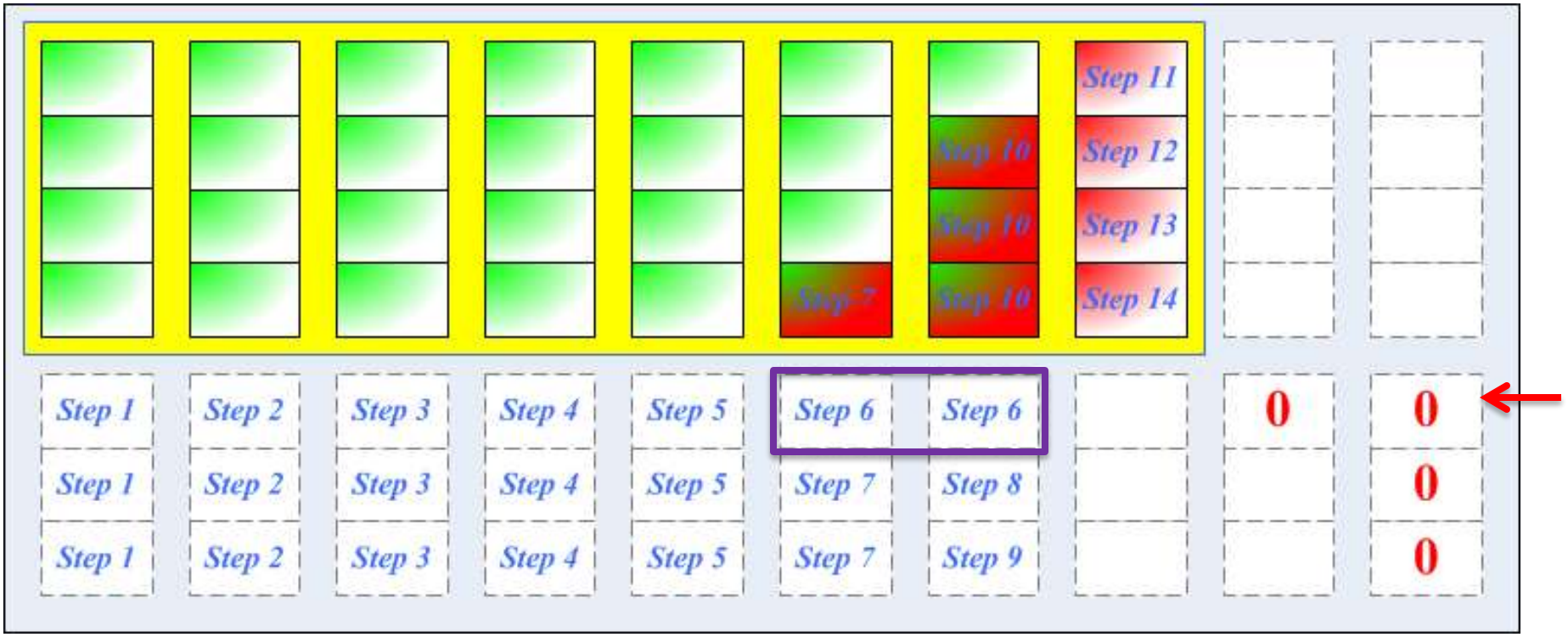
- Detailed steps:



C_{row}: (10,7) code C_{col}: (7,4) code

Upstairs Encoding

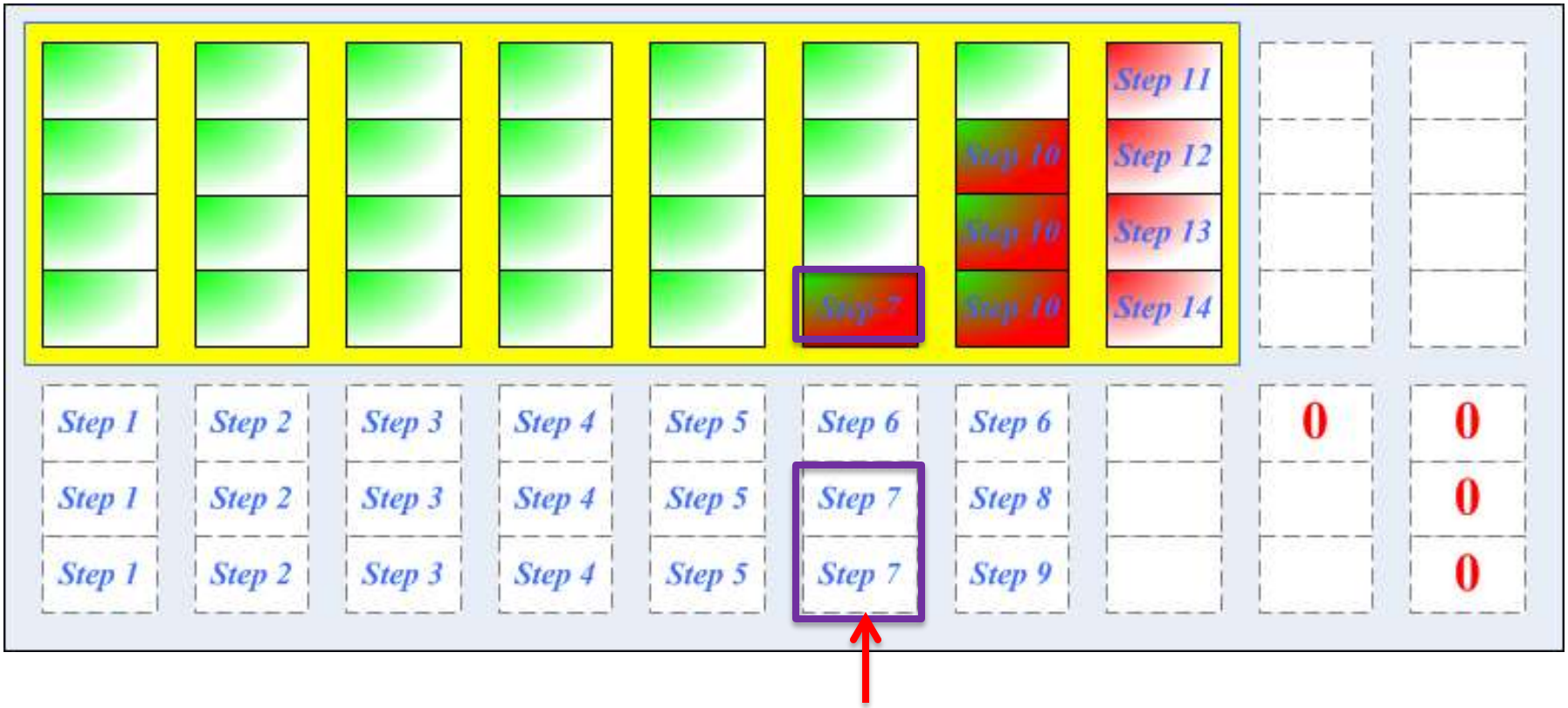
- Detailed steps:



C_{row}: (10,7) code C_{col}: (7,4) code

Upstairs Encoding

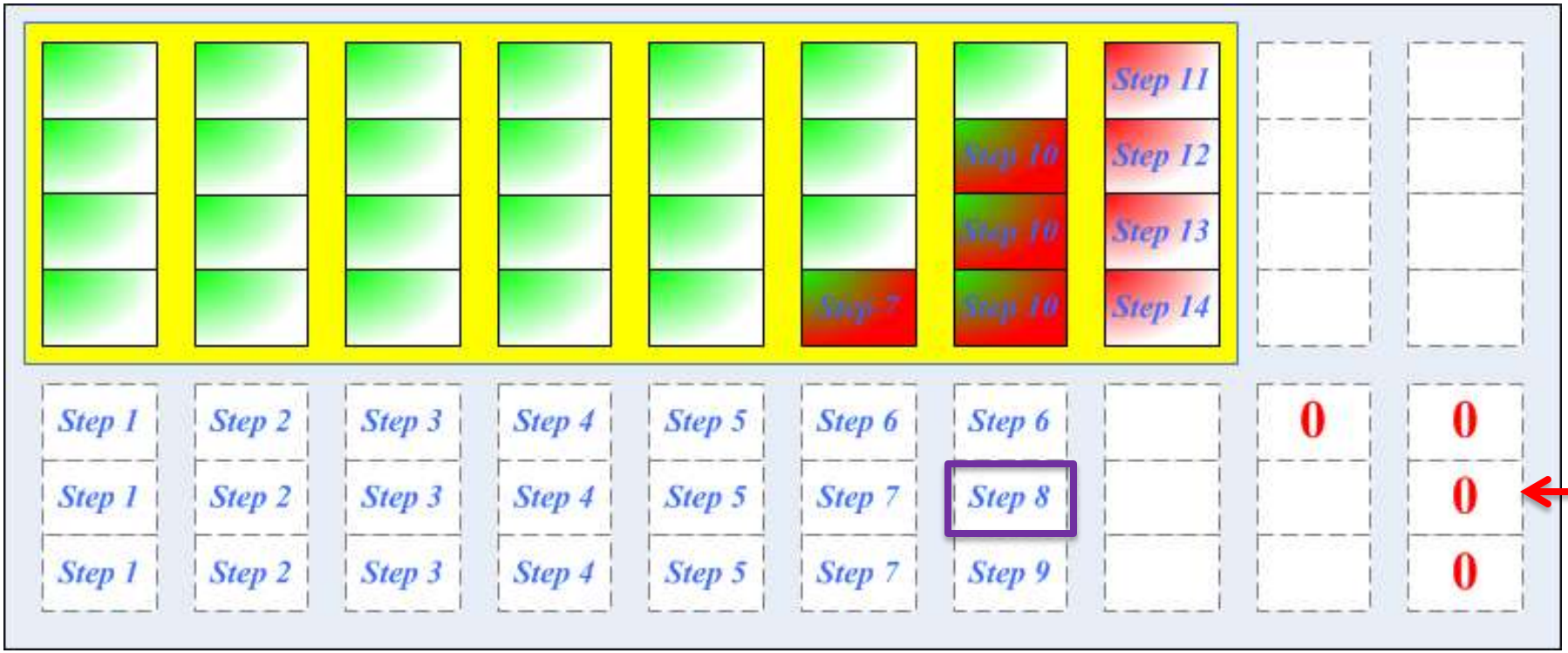
- Detailed steps:



C_{row}: (10,7) code C_{col}: (7,4) code

Upstairs Encoding

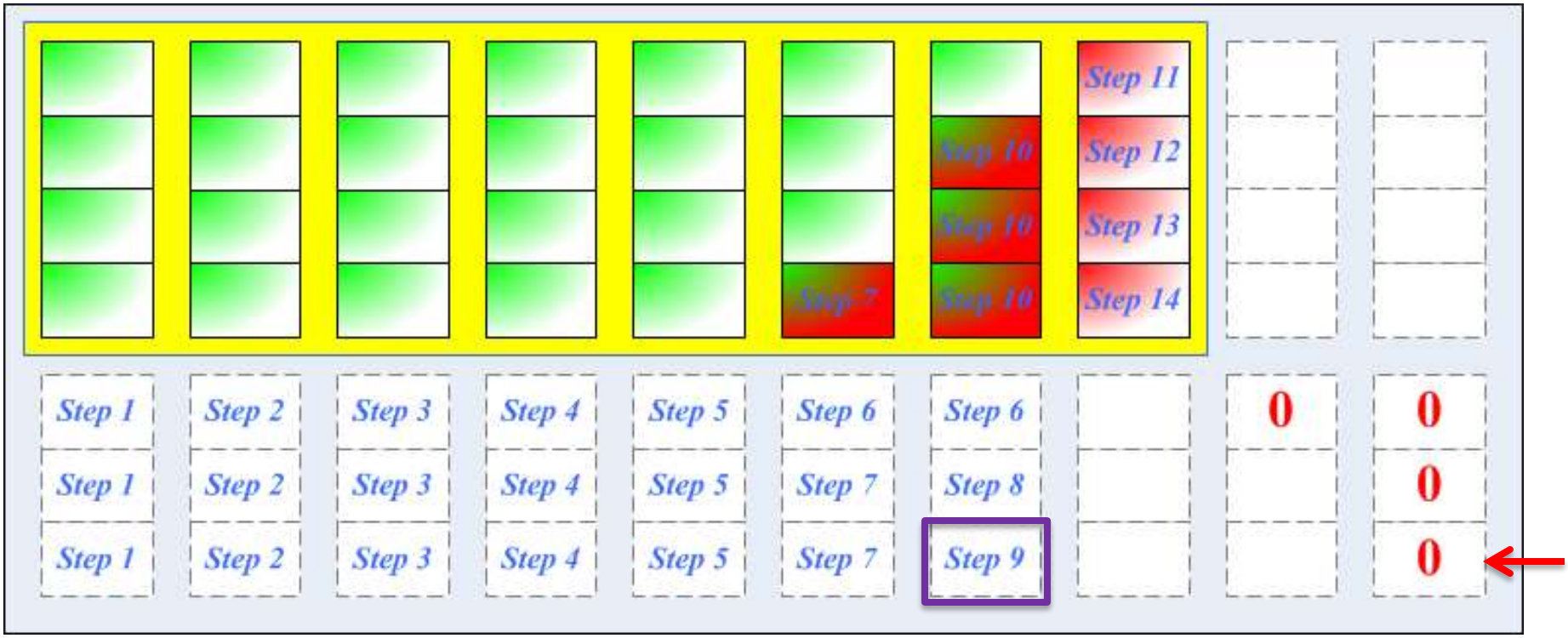
- Detailed steps:



C_{row}: (10,7) code **C_{col}: (7,4) code**

Upstairs Encoding

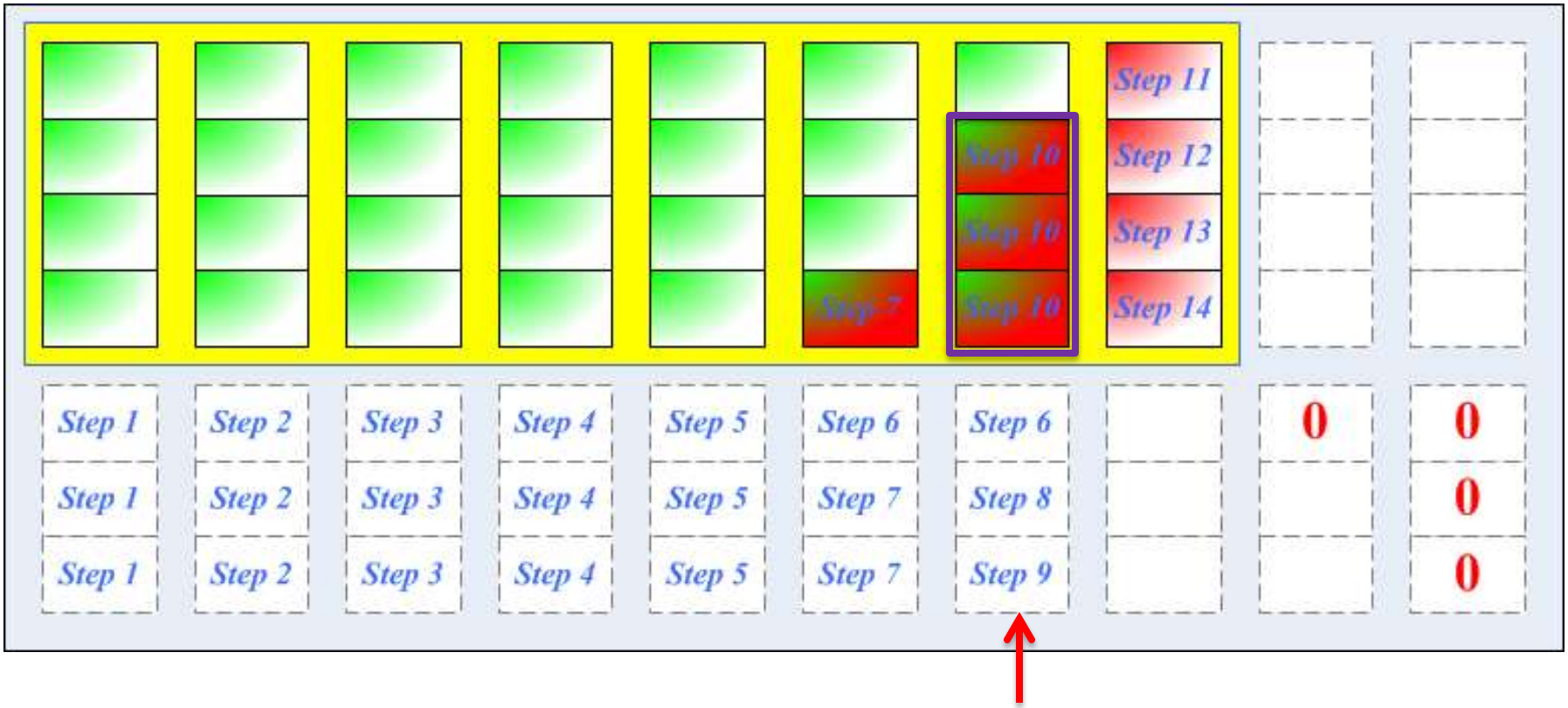
- Detailed steps:



C_{row}: (10,7) code C_{col}: (7,4) code

Upstairs Encoding

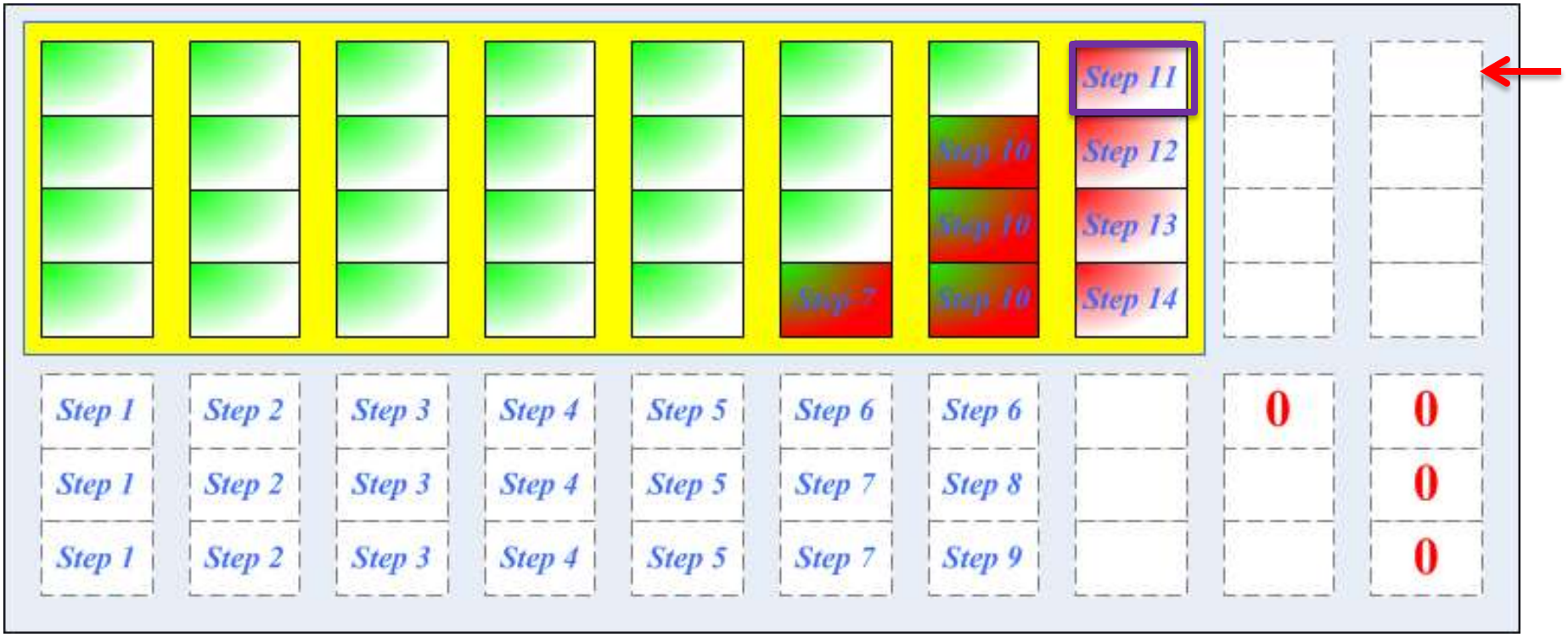
- Detailed steps:



C_{row}: (10,7) code **C_{col}: (7,4) code**

Upstairs Encoding

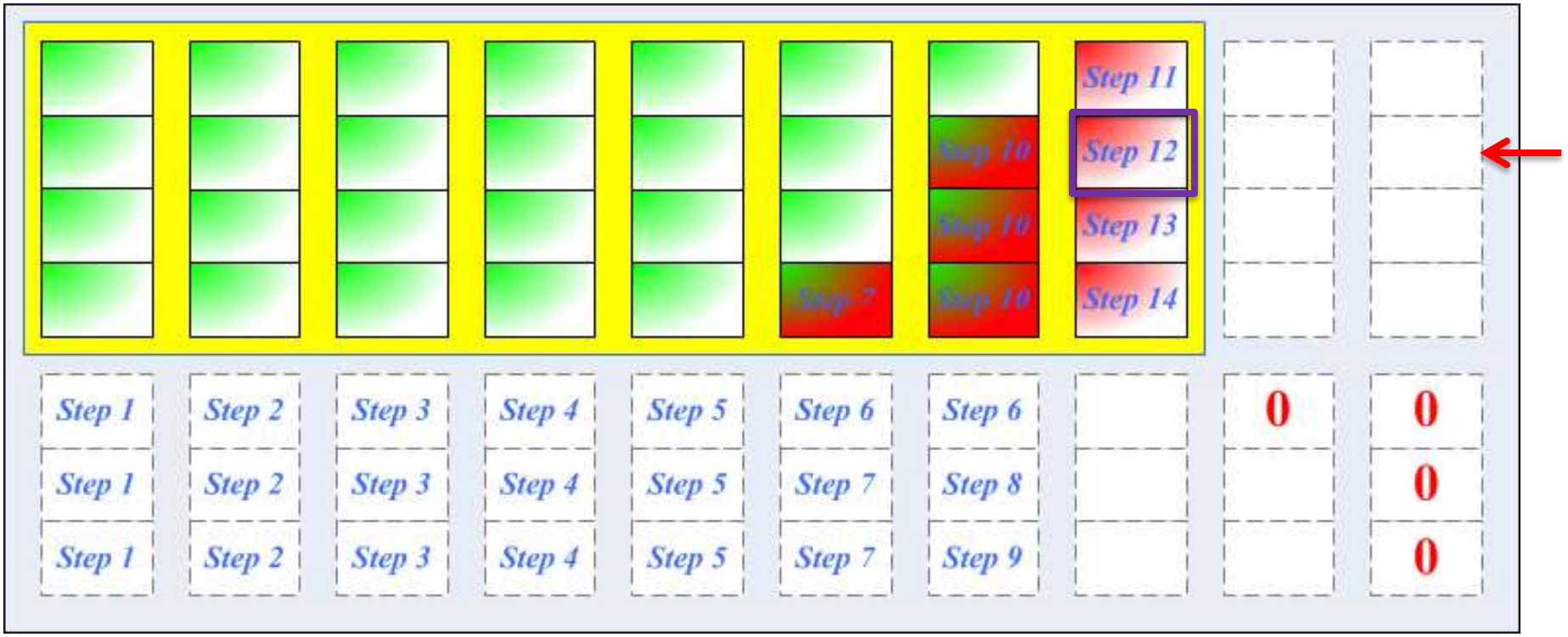
- Detailed steps:



C_{row}: (10,7) code C_{col}: (7,4) code

Upstairs Encoding

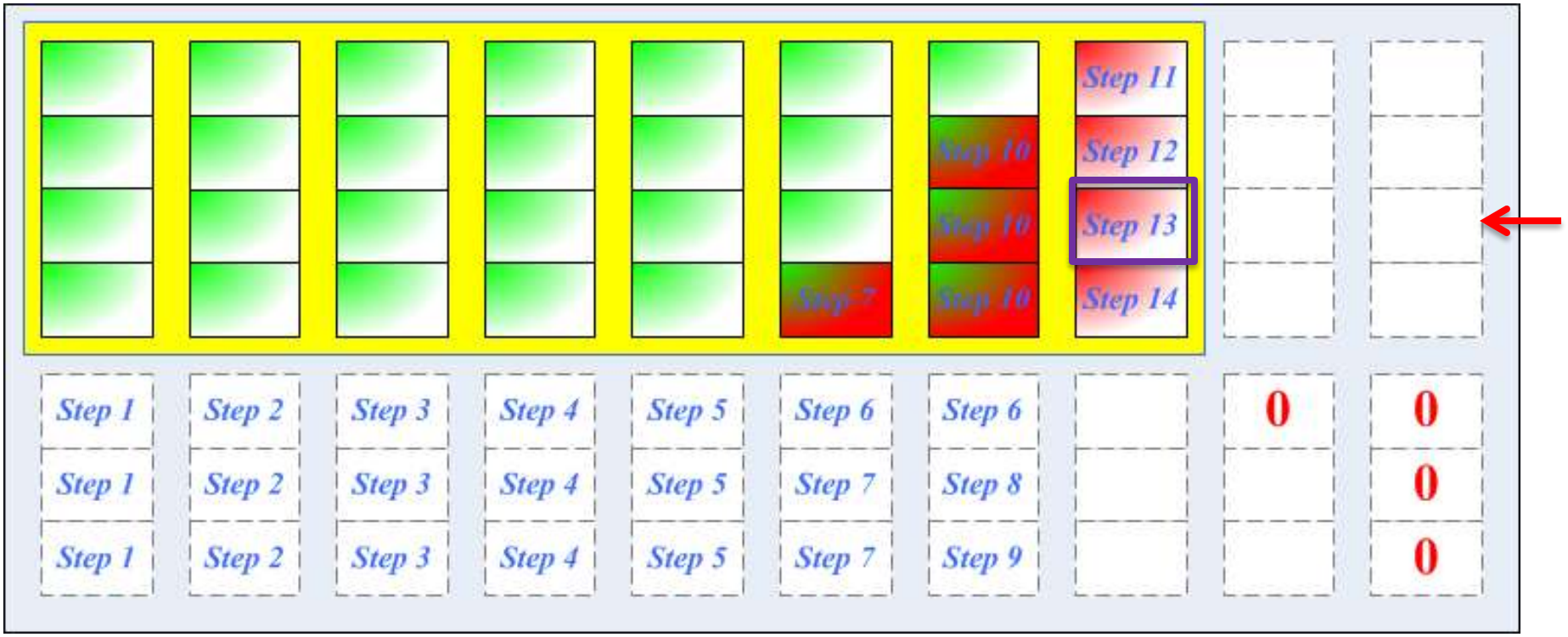
- Detailed steps:



C_{row}: (10,7) code C_{col}: (7,4) code

Upstairs Encoding

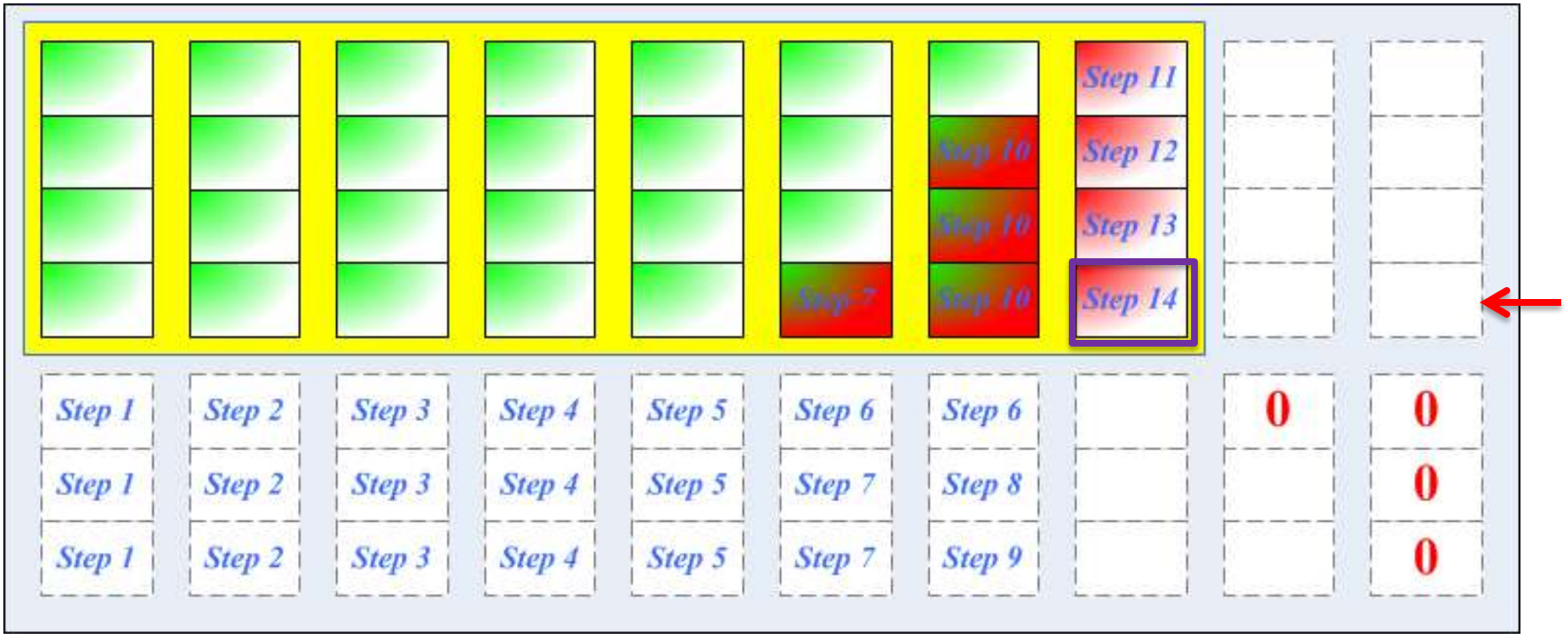
- Detailed steps:



C_{row}: (10,7) code **C_{col}: (7,4) code**

Upstairs Encoding

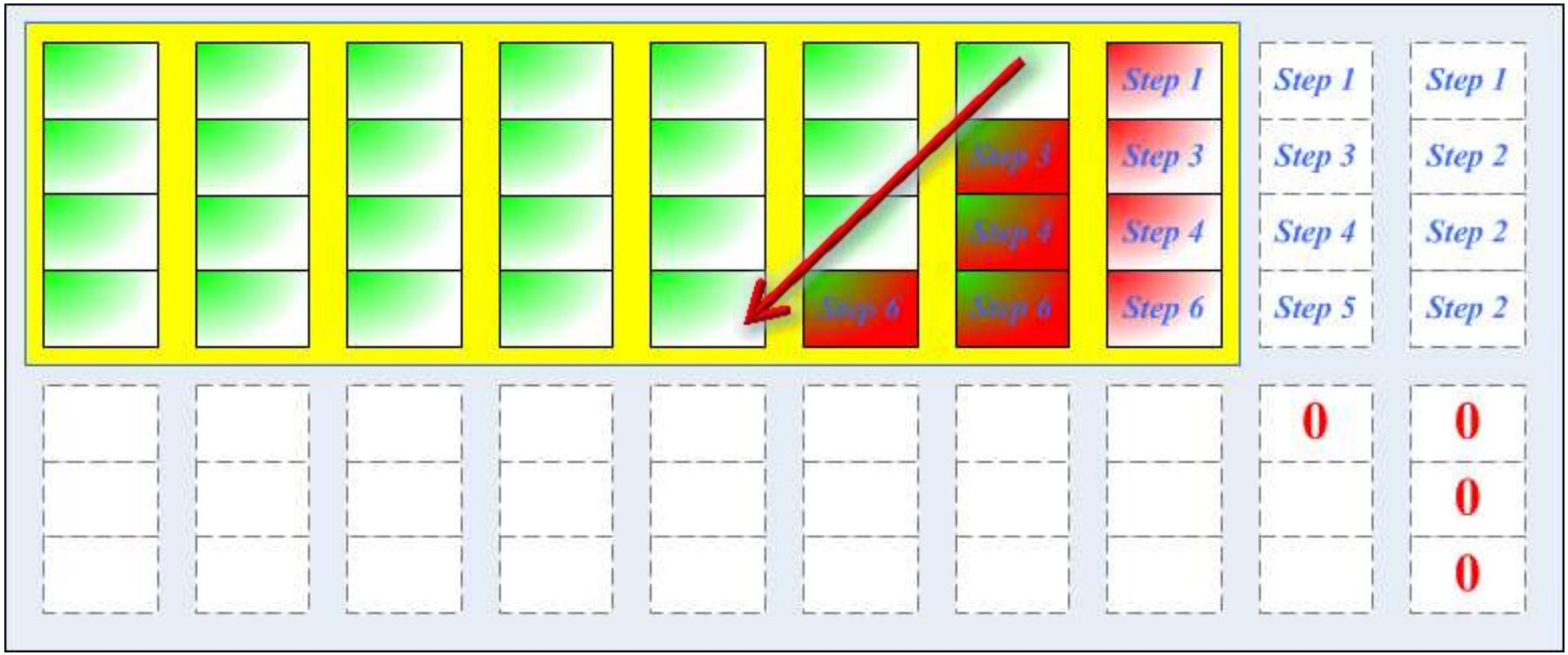
- Detailed steps:



Notes: parity computations reuse previously computed parities

Downstairs Encoding

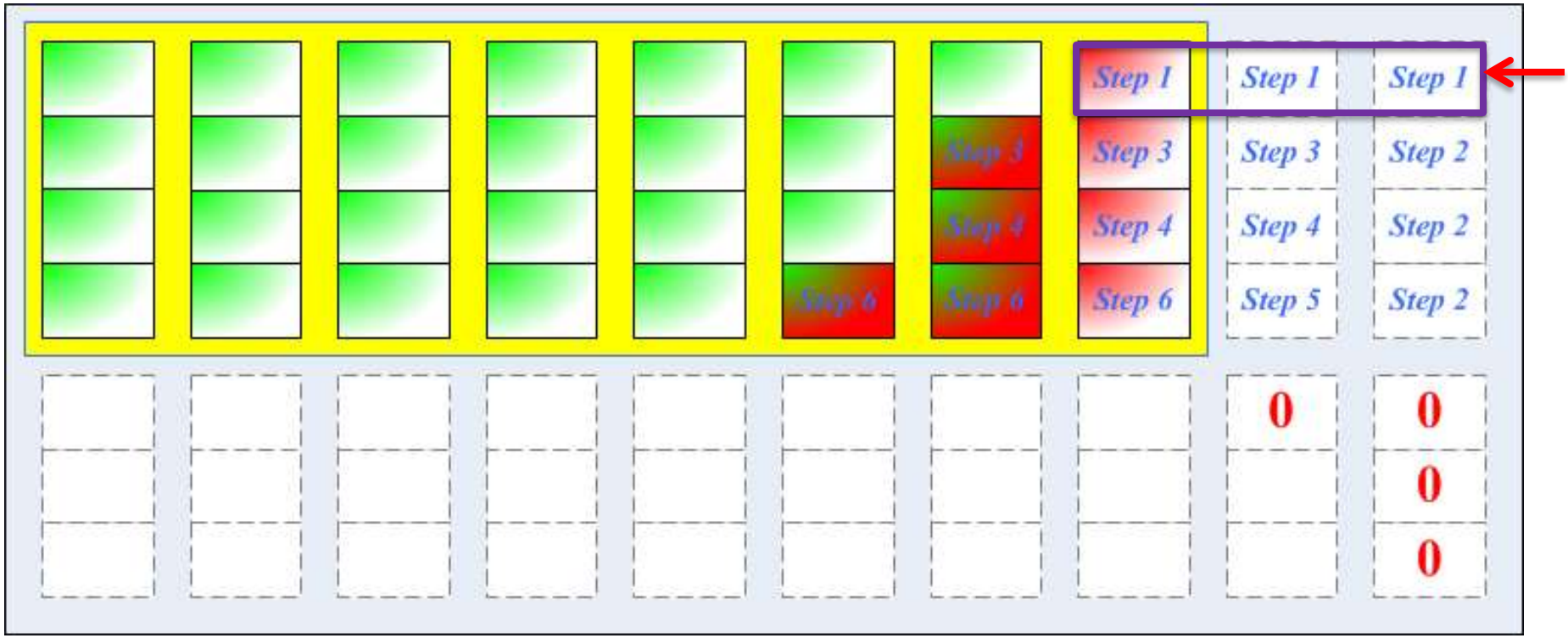
- Another idea: Generate parities in downstairs direction



- **Cannot** be generalized for decoding

Downstairs Encoding

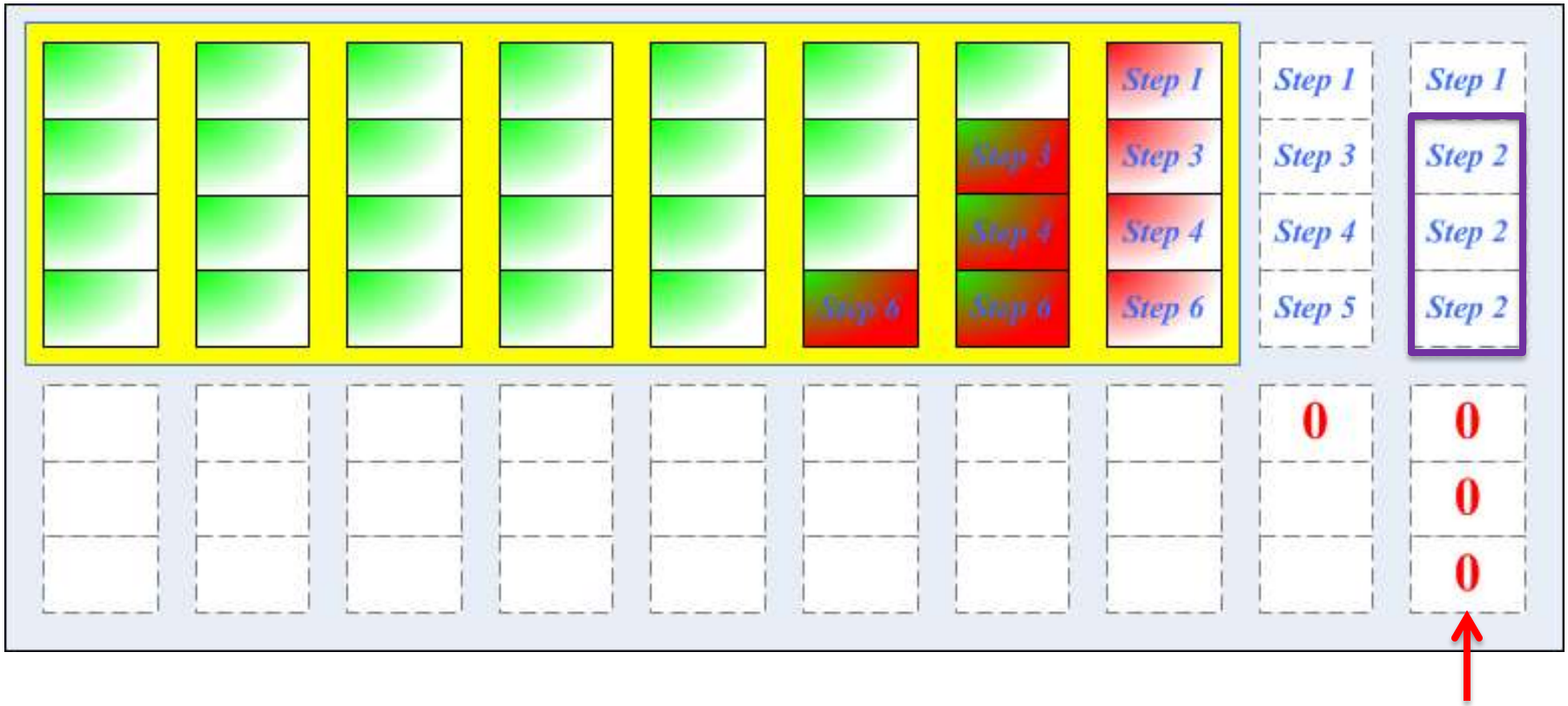
- Detailed steps:



C_{row}: (10,7) code **C_{col}: (7,4) code**

Downstairs Encoding

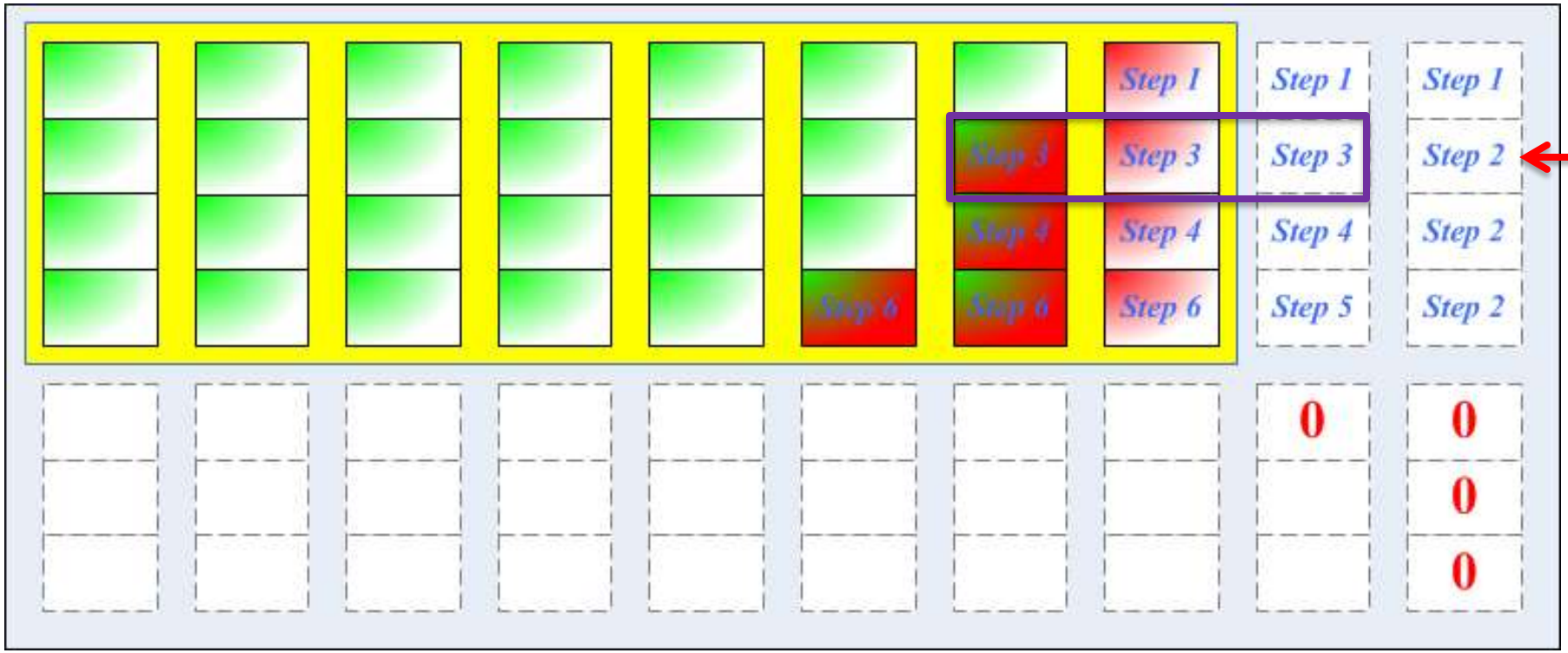
- Detailed steps:



C_{row}: (10,7) code C_{col}: (7,4) code

Downstairs Encoding

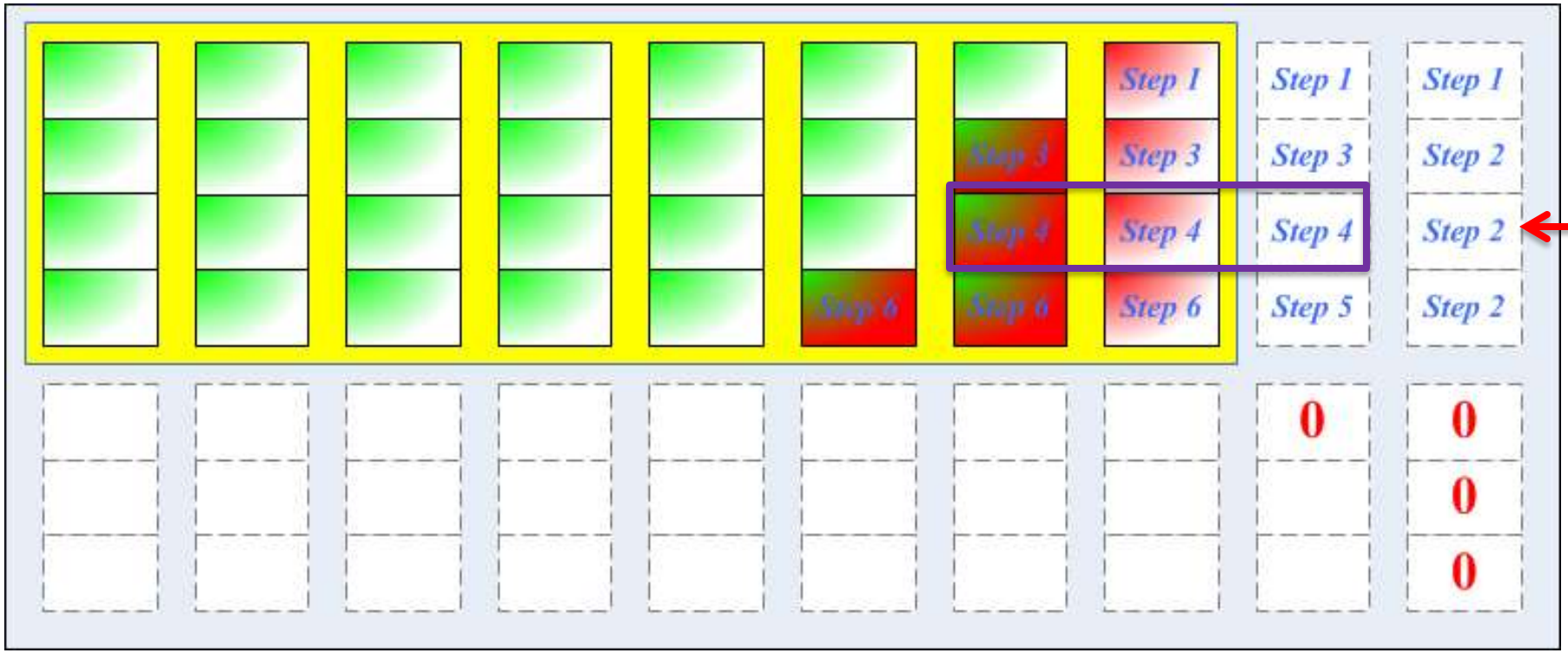
- Detailed steps:



C_{row}: (10,7) code C_{col}: (7,4) code

Downstairs Encoding

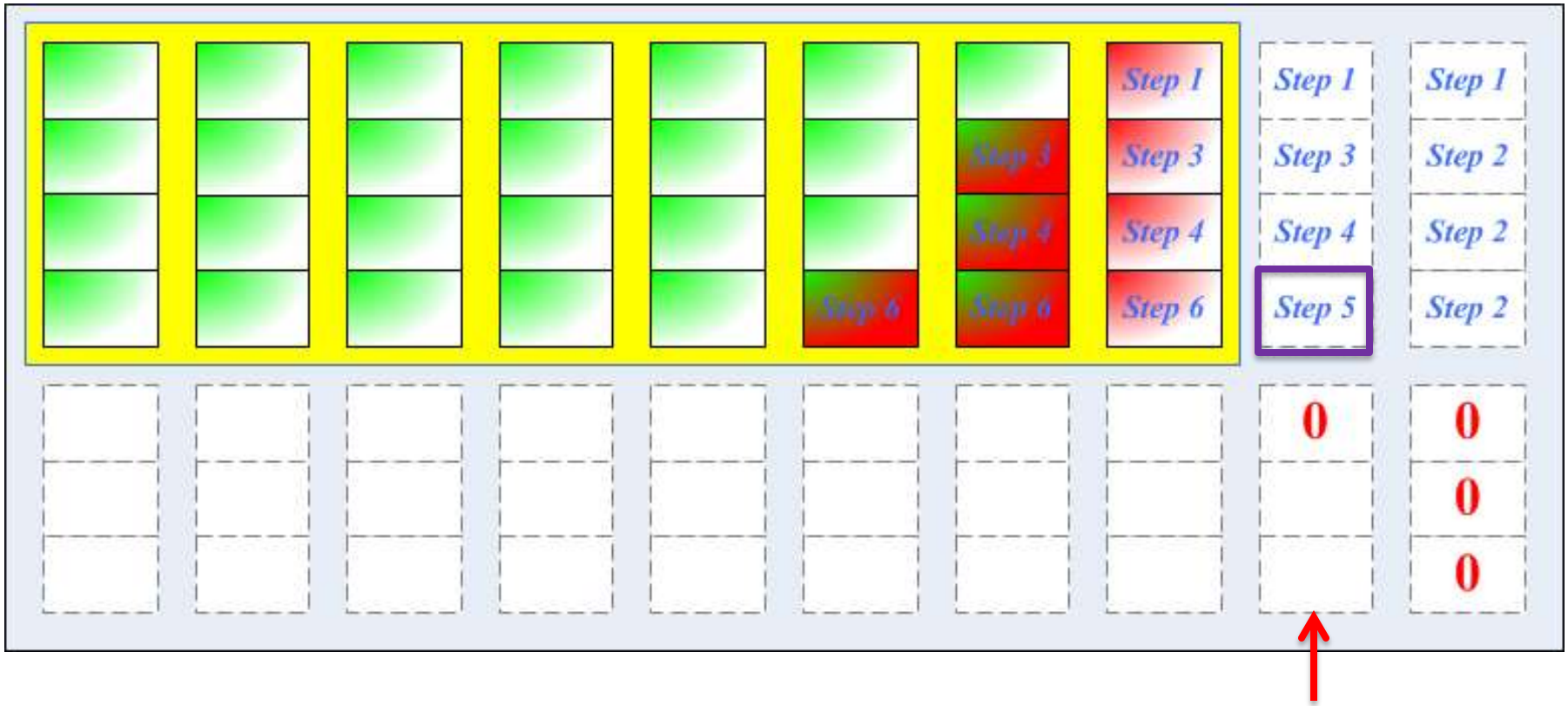
- Detailed steps:



C_{row}: (10,7) code C_{col}: (7,4) code

Downstairs Encoding

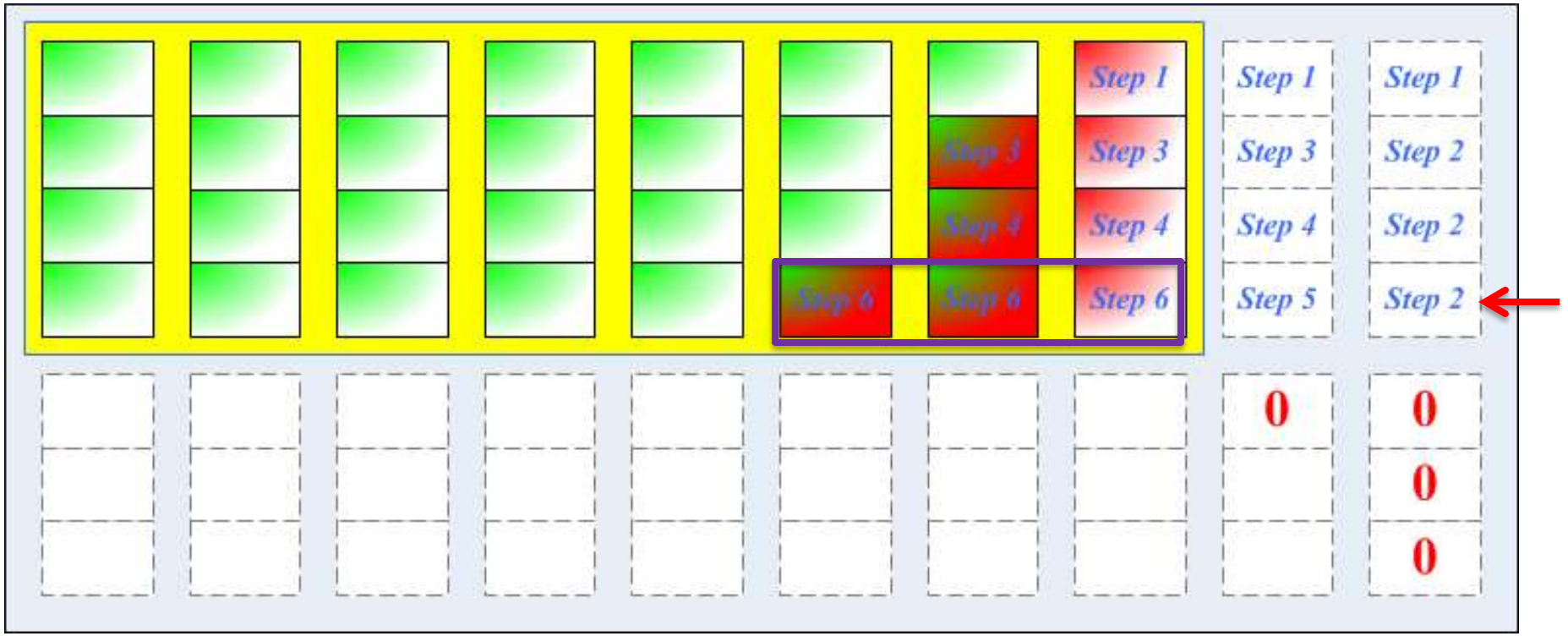
- Detailed steps:



C_{row}: (10,7) code **C_{col}: (7,4) code**

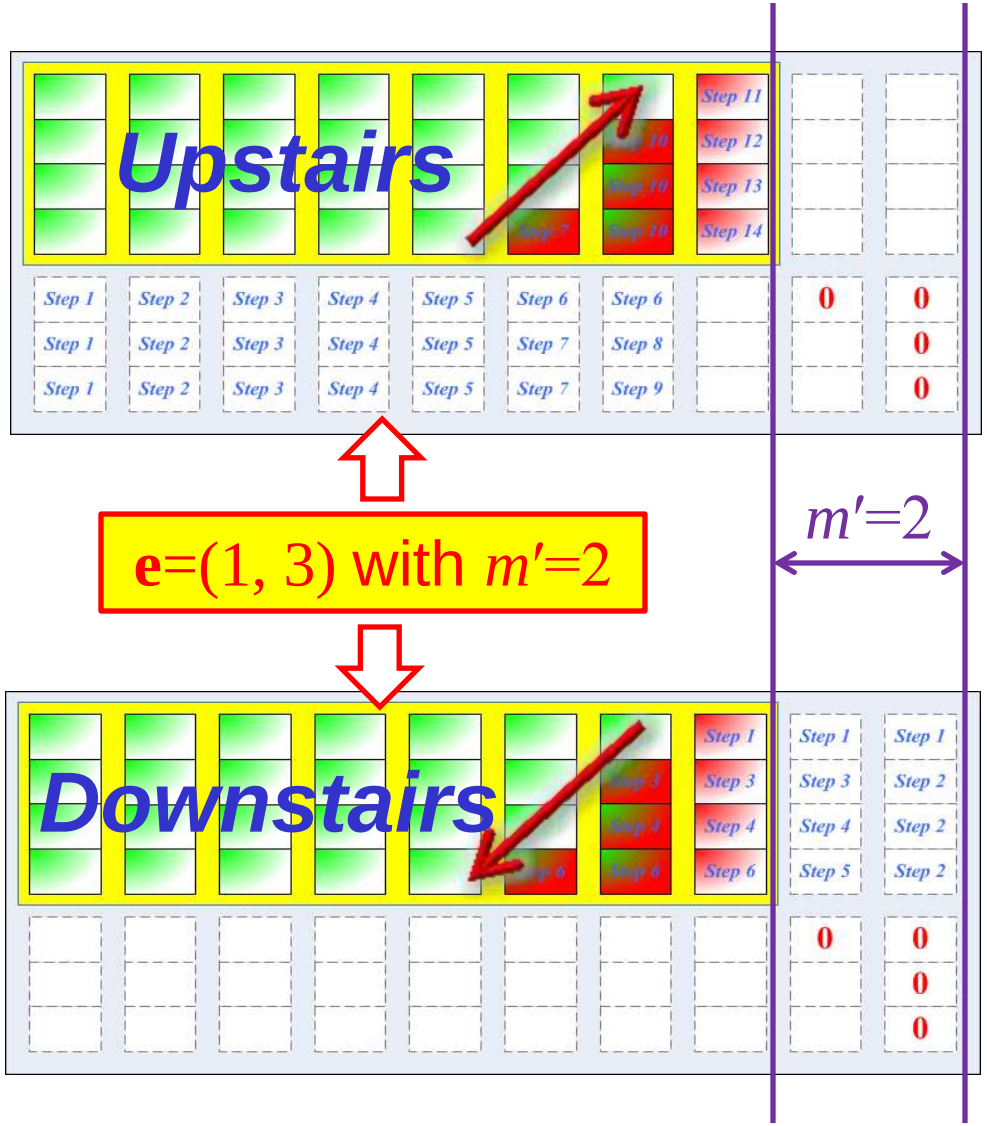
Downstairs Encoding

- Detailed steps:



Like upstairs encoding, parity computations reuse previously computed parities

Choosing Encoding Methods



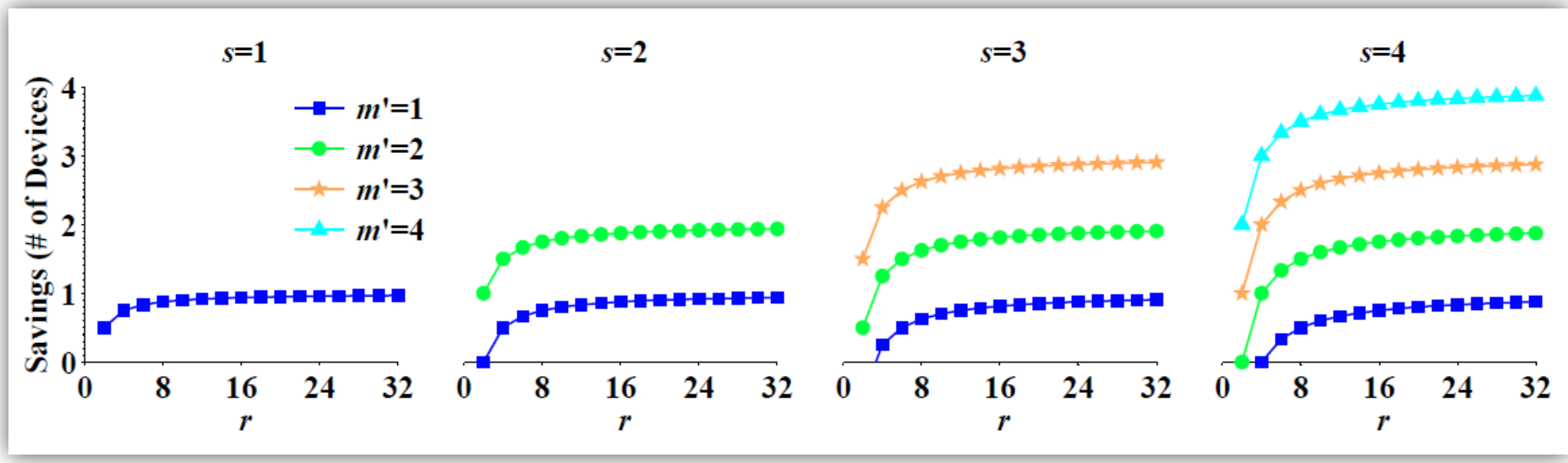
- The two methods are **complementary**
- Intuition:
  - Choose upstairs encoding for large m'
  - Choose downstairs encoding for small m'
- Details in paper

Evaluation

- Implementation
  - Built on libraries Jerasure [Plank, FAST'09] and GF-Complete [Plank, FAST'13]
- Testbed machine:
  - Intel Core i5-3570 CPU 3.40GHz with SSE4.2
- Comparisons with RS codes and SD codes
  - Storage saving
  - Encoding/decoding speeds
  - Update cost

Storage Space Saving

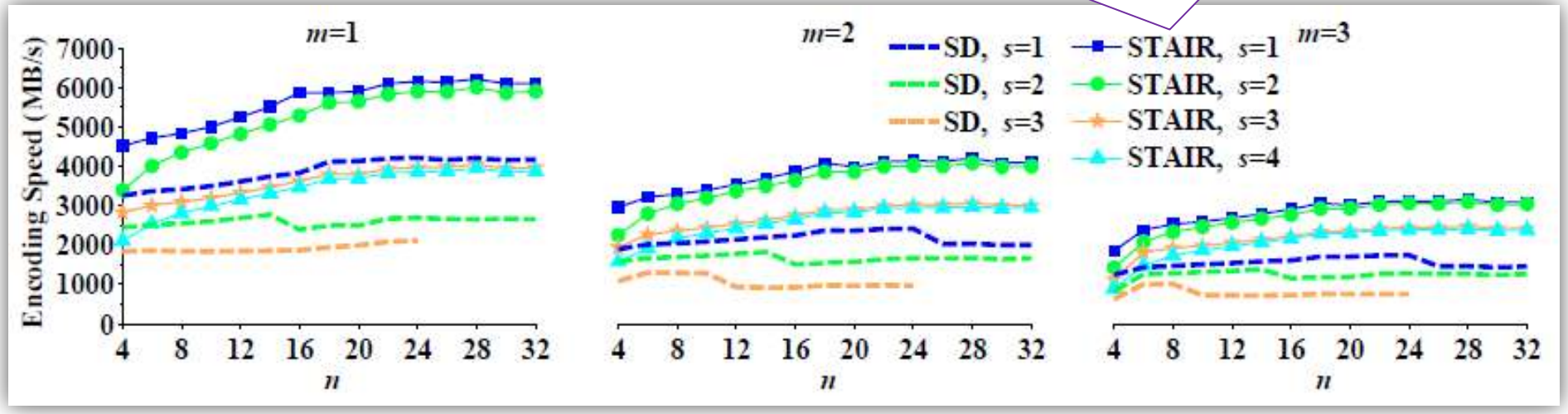
- STAIR codes save $m' - \frac{s}{r}$ devices compared to traditional erasure codes using device-level fault tolerance
  - s = # of tolerable sector failures
  - m' = # of partially failed devices
  - r = chunk size



As r increases, # of devices saved ≈ m'

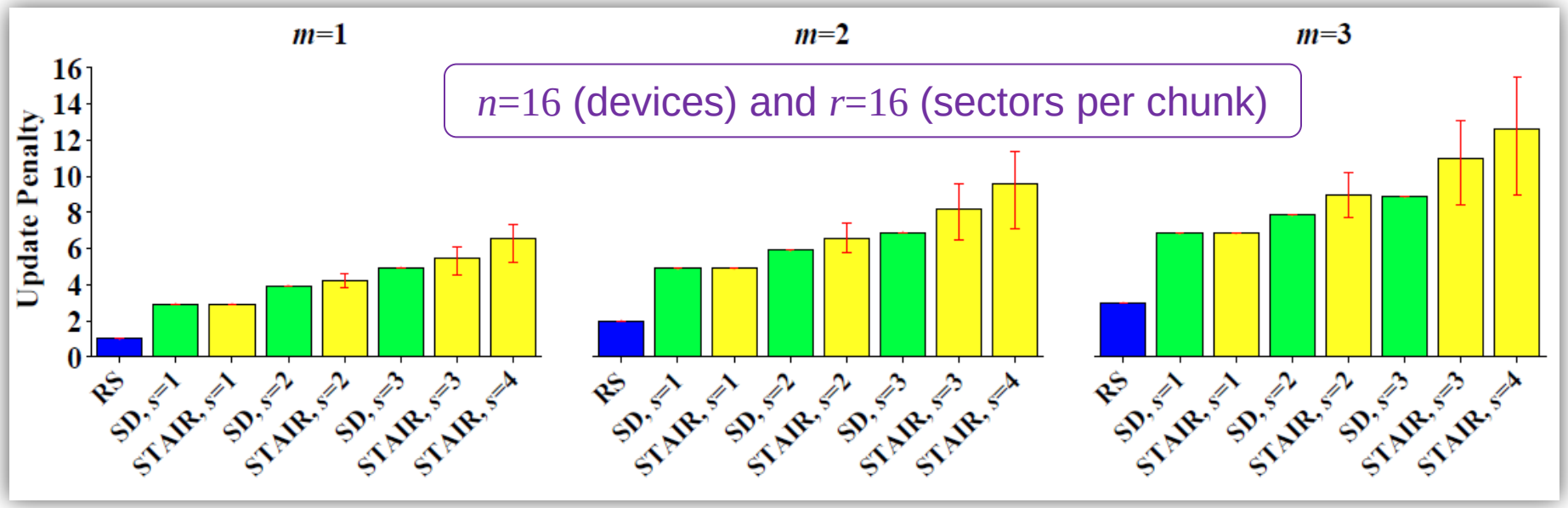
Encoding Speed

n = number of devices
r=16 (sectors per chunk)



- Encoding speed of STAIR codes is on order of **1000MB/s**
- STAIR codes **improve** encoding speed of SD codes **by ~100%**, due to parity reuse
- Similar results for decoding

Update Cost



(Update penalty: average # of updated parity sectors for updating a data sector)

- **Higher update penalty than traditional codes**, due to global parity sectors
- Good for systems with rare updates (e.g., backup) or many full-stripe writes (e.g., SSDs) [*Plank et al., FAST '13, TOS'14*]

Conclusions

- STAIR codes: a general family of erasure codes for tolerating both device and sector failures in a space-efficient manner

- Complementary upstairs encoding and downstairs encoding with improved encoding speed via parity reuse

- Open source STAIR Coding Library (in C):
  - **http://ansrlab.cse.cuhk.edu.hk/software/stair**