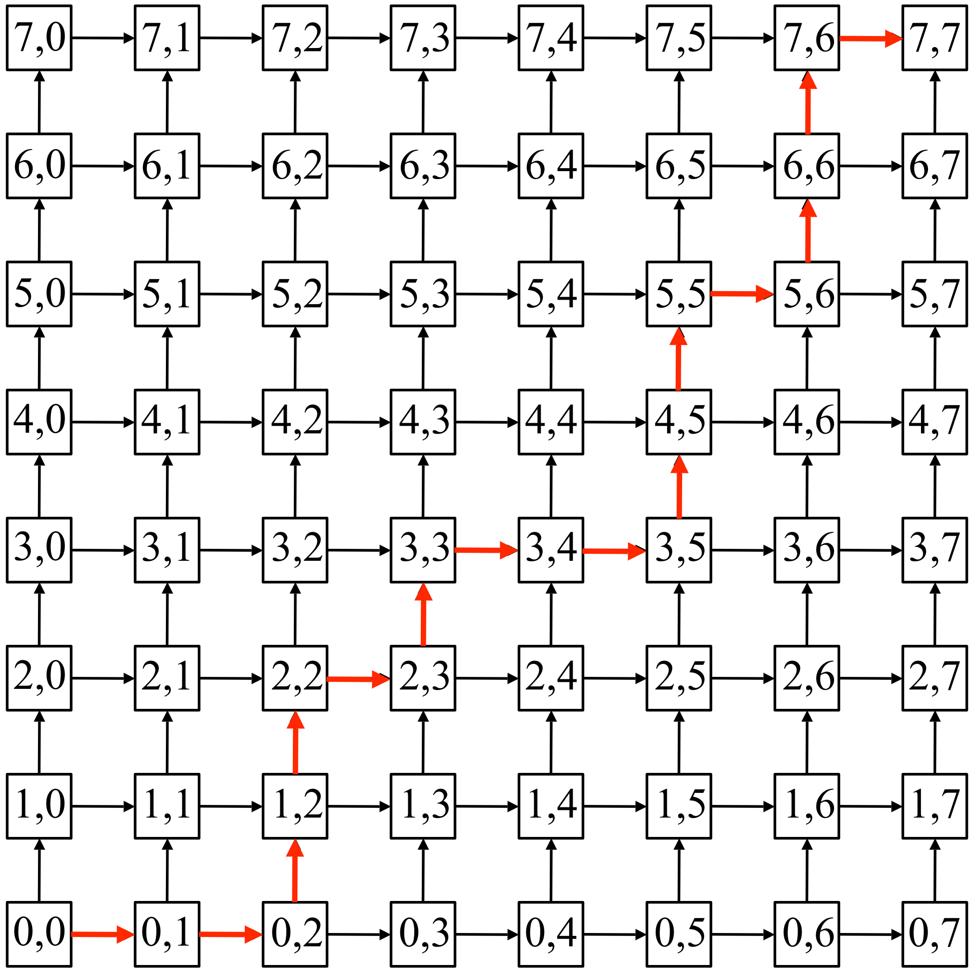
Rewriting Codes for Flash Memories Based Upon Lattices, and an Example Using the E8 Lattice

Brian M. Kurkoski
kurkoski@ice.uec.ac.jp

University of Electro-Communications
Tokyo, Japan

Workshop on
Application of Communication Theory to Emerging Memory Technologies
at
Globecom 2010
Miami, Florida, USA 6 December 2010

Rewriting Codes for Flash



Model of Conventional Rewriting Code

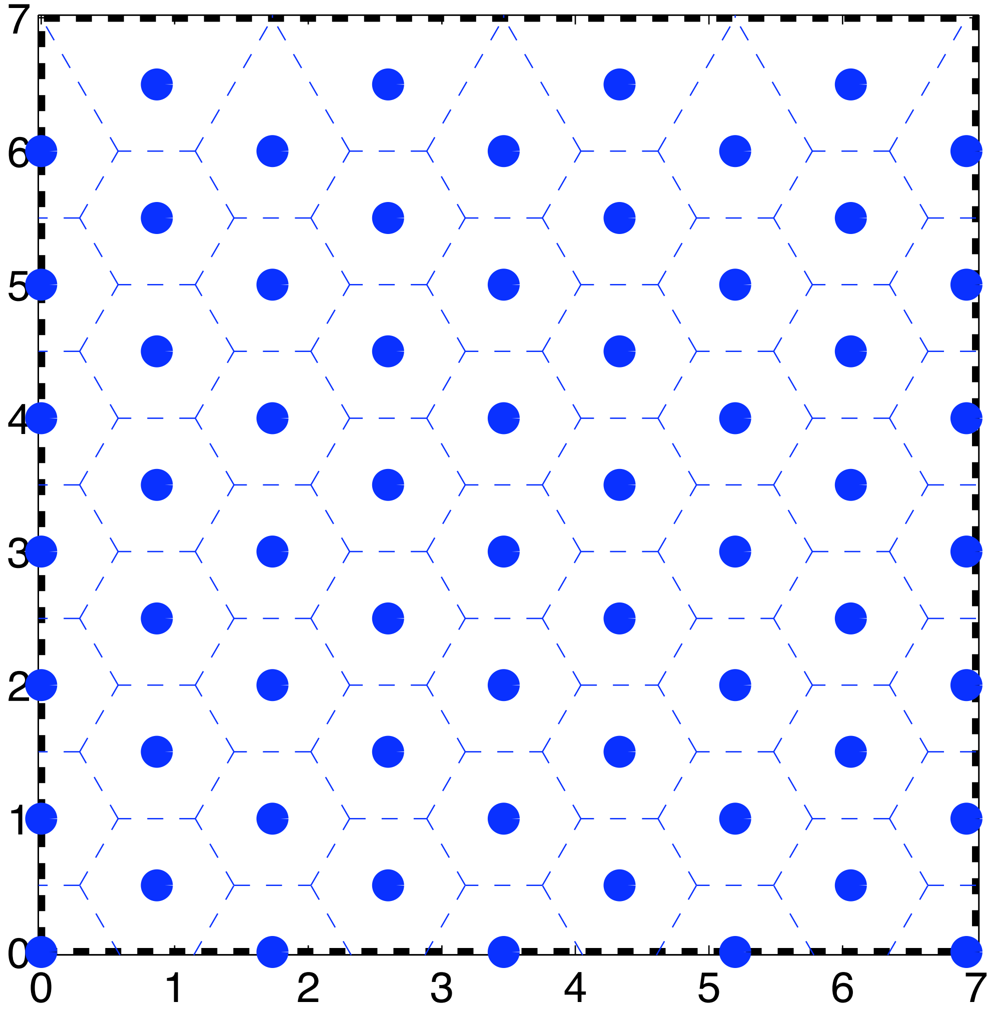
n=2 flash cells, q=8 discrete levels

values can only increase (without erasing)

Rectangular lattice: “uncoded” lattice

Easy to decode. Poor performance.

image thanks: Eitan Yaakobi



Lattices, also called sphere packings

higher packing density

error-correcting properties

can achieve channel capacity $n \to \infty$

Will show lattices can be used for rewriting

Outline

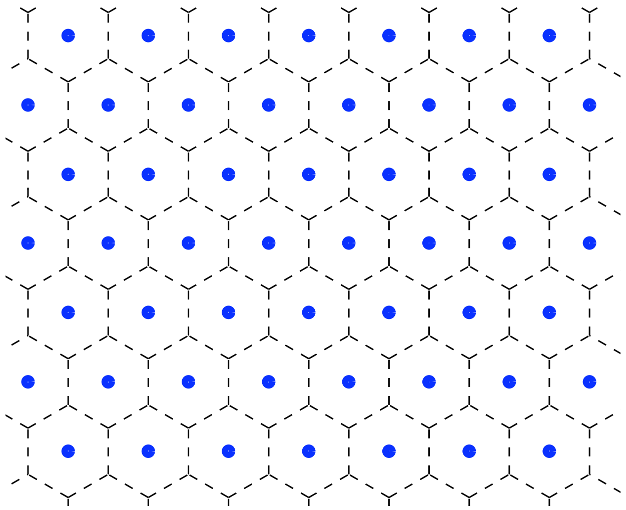
Lattices for re-writing codes. Using two-dimensional examples:

- Code construction — intersection of a lattice and a shaping region
- Encoding — one-to-many mapping
- Maximizing the future number of writes
- Minimum number of writes is equal to D, a code parameter
- "Hash" or permutation to increase the average number of writes

Numerical results on average number of writes using E8 lattice:

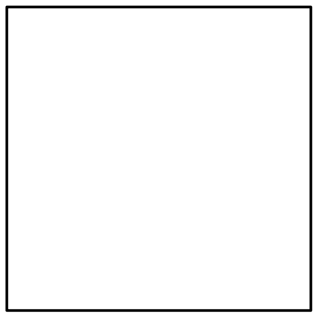
- increasing performance is strongly dependent on q
- Open question: how does performance depend upon n?

Lattices for Flash — Code Construction, Without Rewriting



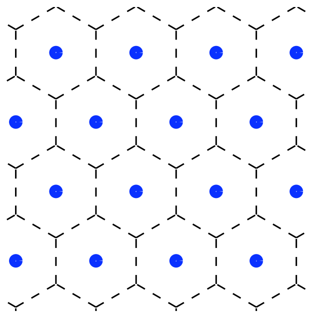
Lattice Λ is infinite
code over reals
"minimum distance"

$\cap$

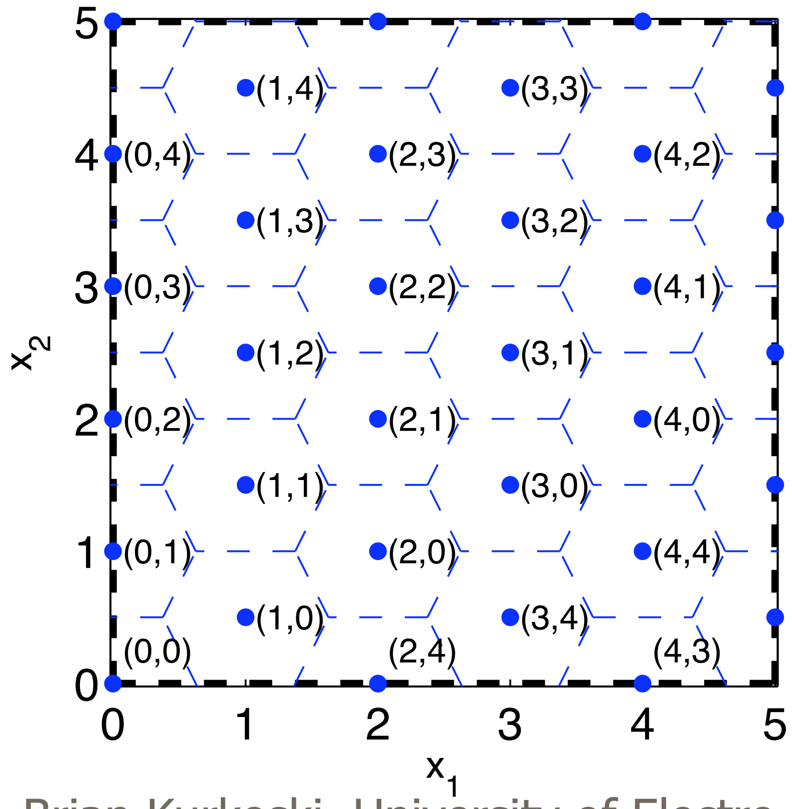


Shaping region B
finite

$=$



codebook $\Lambda \cap B$
is finite



Writing in 2 cells: 2-dimensional examples

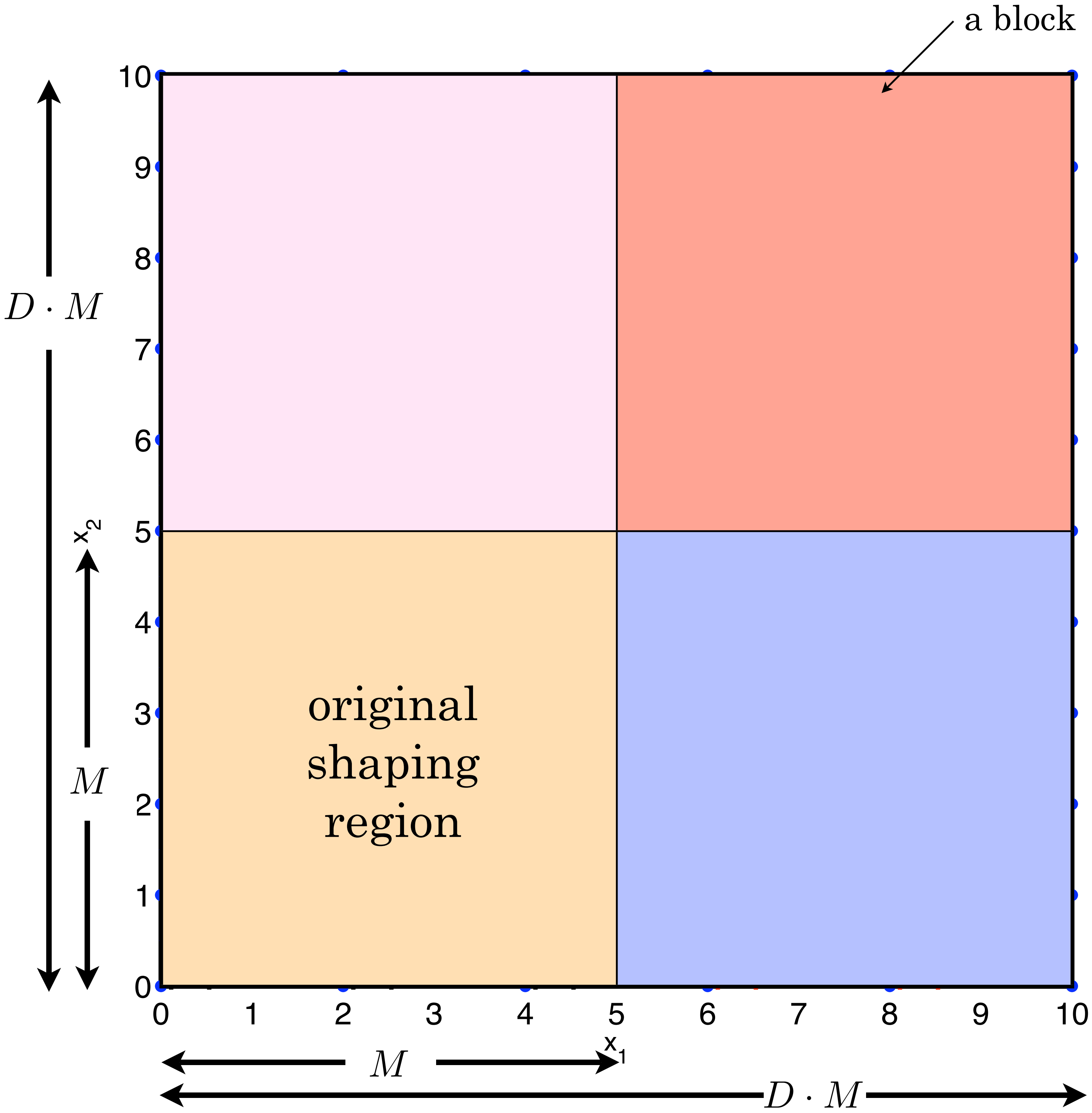
Cell value is from 0 to q-1

Lattice scaling: Volume of Voronoi region is 1.
Same as rectangular lattice, used by conventional rewriting codes.

One-to-one mapping from information to codebook

- If lattice generator matrix is triangular, then mapping is straightforward

Lattices for Flash —
Code Construction, WITH Rewriting



Two code parameters:

D copies of shaping region in each dimension

M: side length of each

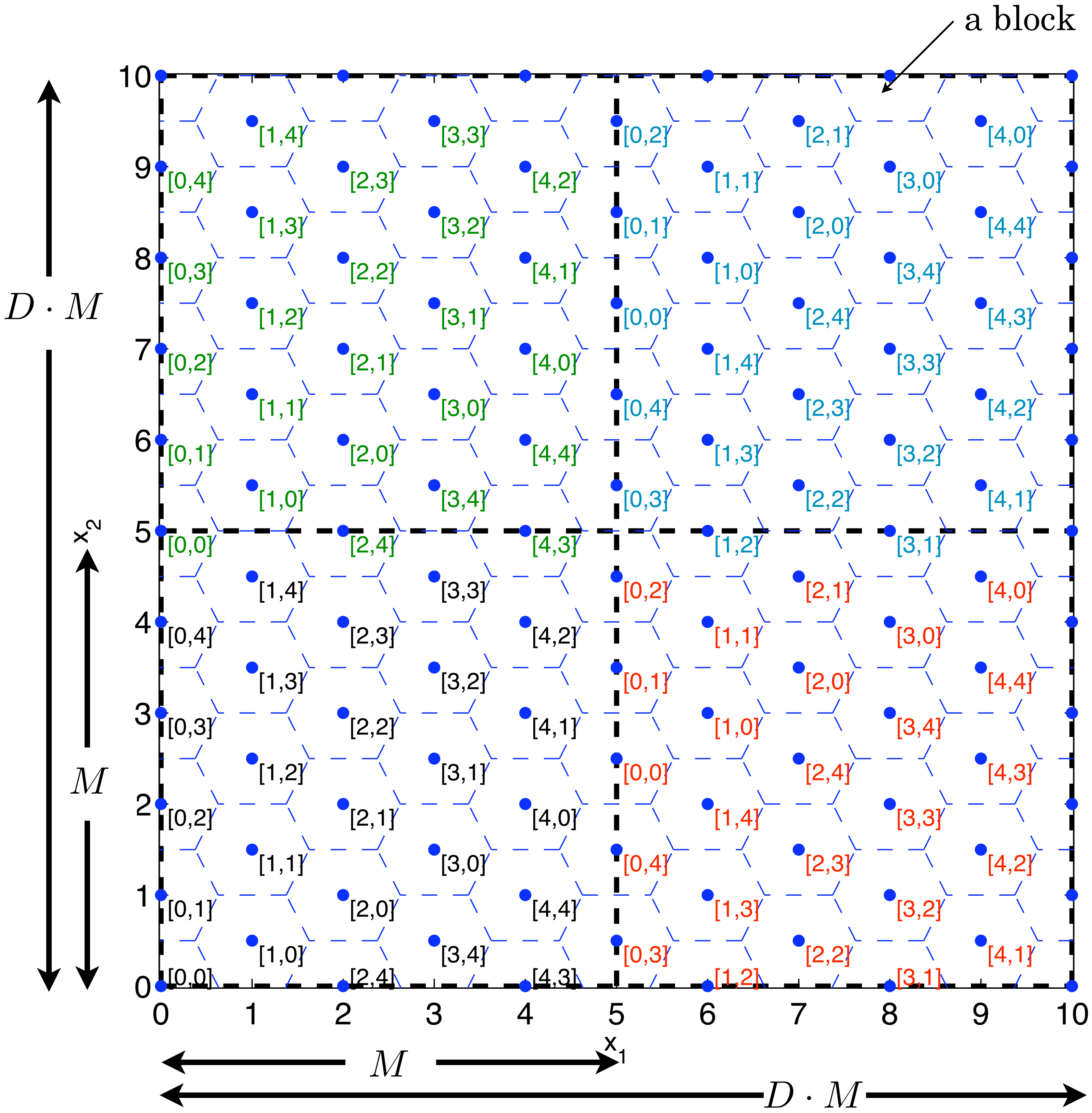
$DM = q - 1$

D^n blocks, each one has a one-to-one mapping.

Overall code has one-to-D^n mapping

Example has $D = 2$, $M = 5$. Compare with $q = 11$

Lattices for Flash — Code Construction, WITH Rewriting



Two code parameters:

D copies of shaping region in each dimension

M: side length of each

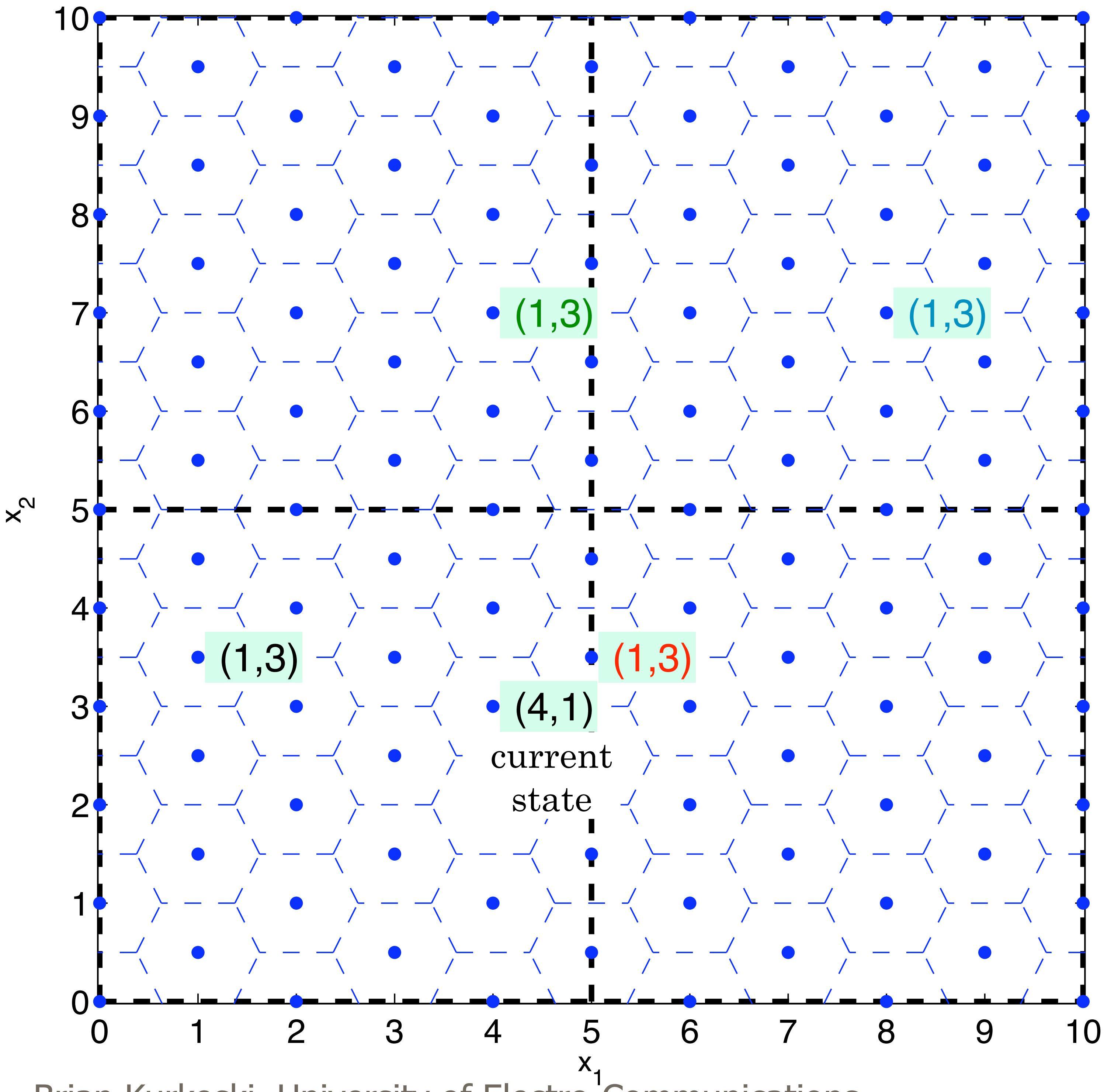
$DM = q - 1$

D^n blocks, each one has a one-to-one mapping.

Overall code has one-to-D^n mapping

Example has $D = 2$, $M = 5$. Compare with $q = 11$

Lattices with Rewriting — Encoding



Memory has state s = (4,1)

Memory value can only increase

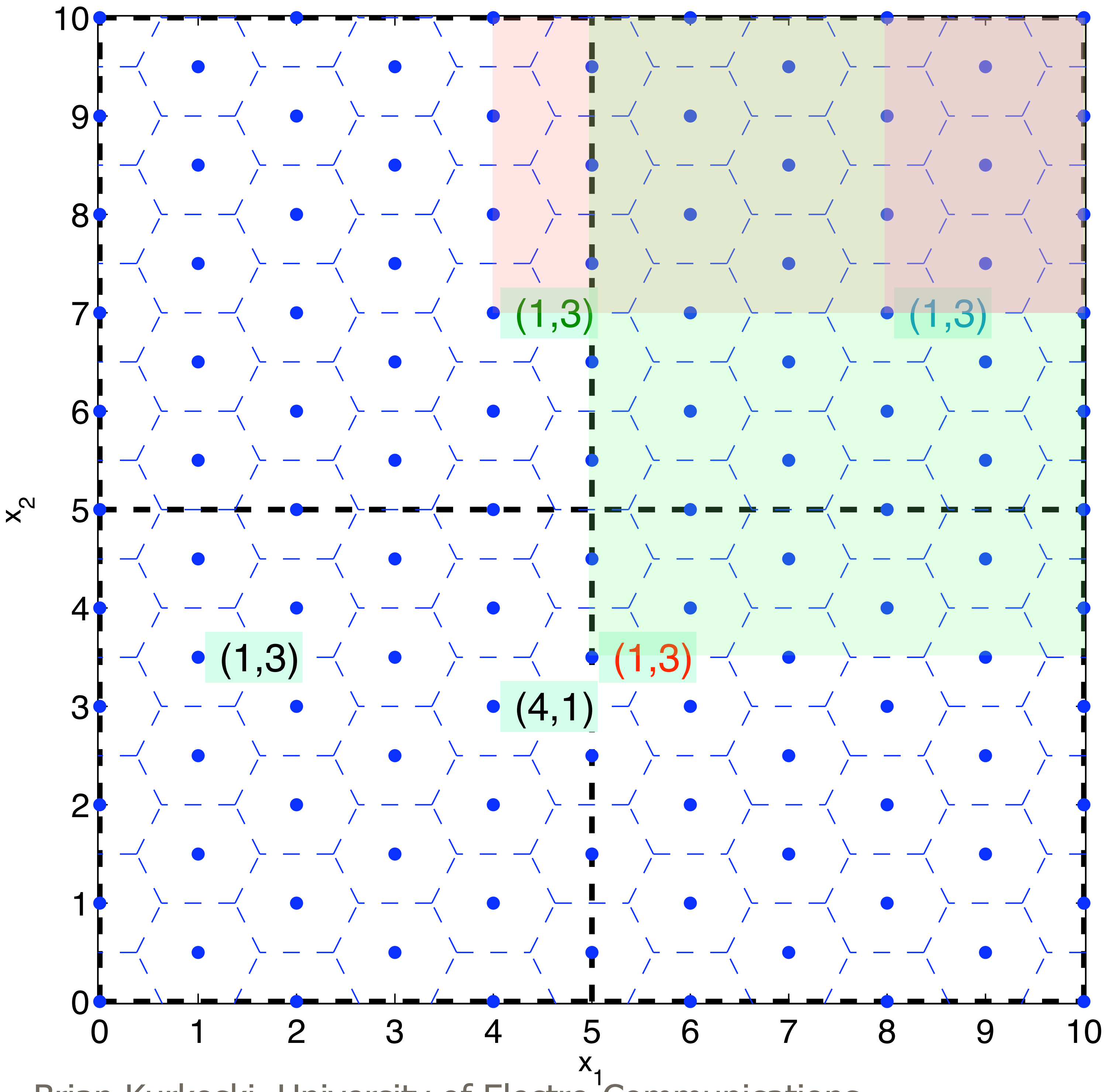
Given new information sequence (1,3), there are D^n candidates

Choose candidate which maximizes the remaining "volume"

- If overall code has a linear encoding, this is straightforward.
- But, to improve the average number of writes, we'll destroy the *global* linearity

As a result, search over 2^n-1 neighboring blocks to maximize remaining volume.

Maximizing the Remaining Volume



Goal: maximize the future number of writes

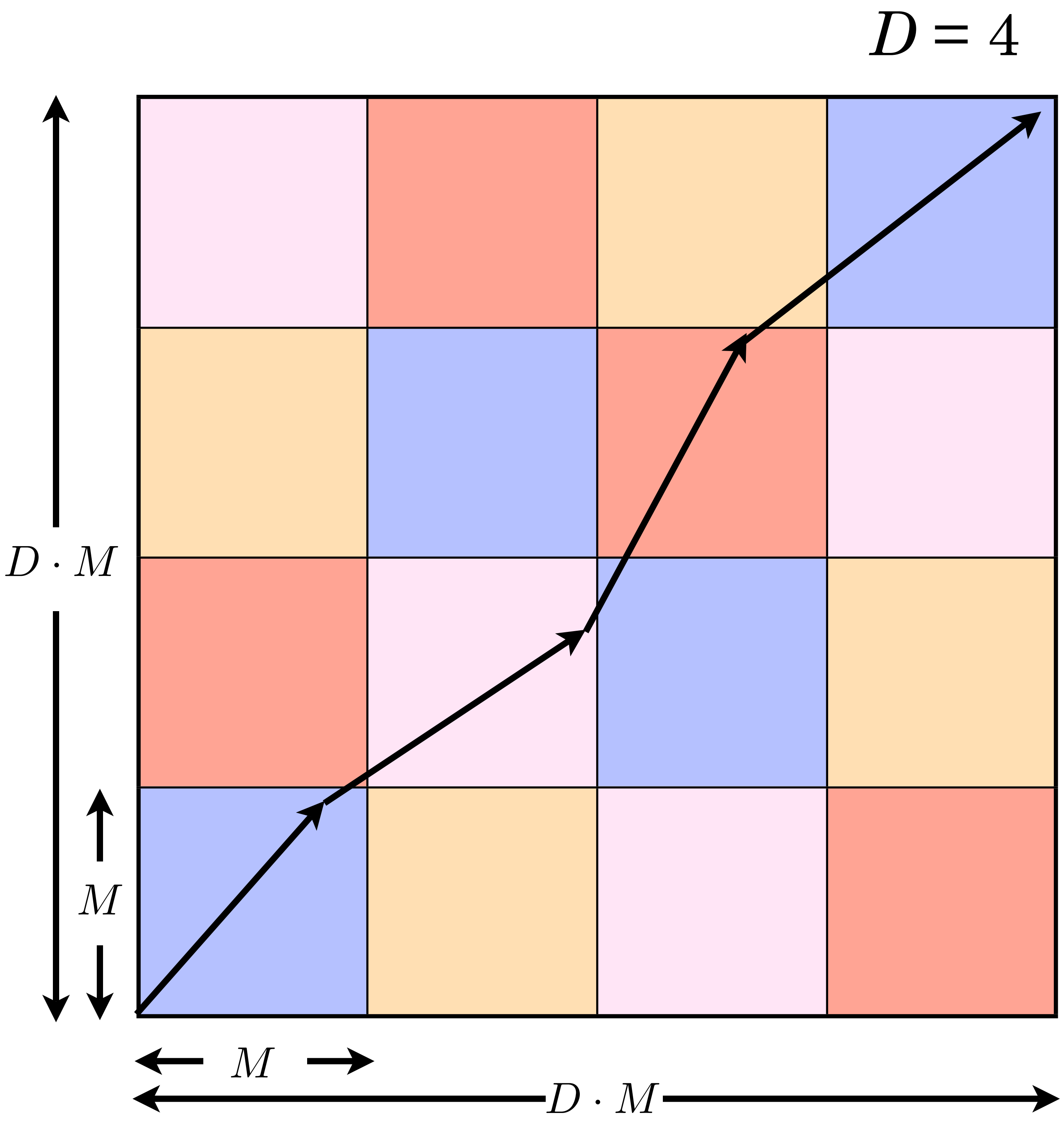
Difficult to count "accessible" lattice points

No *a priori* knowledge of future data points

Assume that lattice points are uniformly distributed

- maximizes number of points for future writes
- ignore the encoding/ mapping
- Assumption resembles the "continuous approximation"

Minimum Number of Writes is D



In the worst case:

- a codeword near the upper-right hand corner of each block is written

It is relatively easy to see:

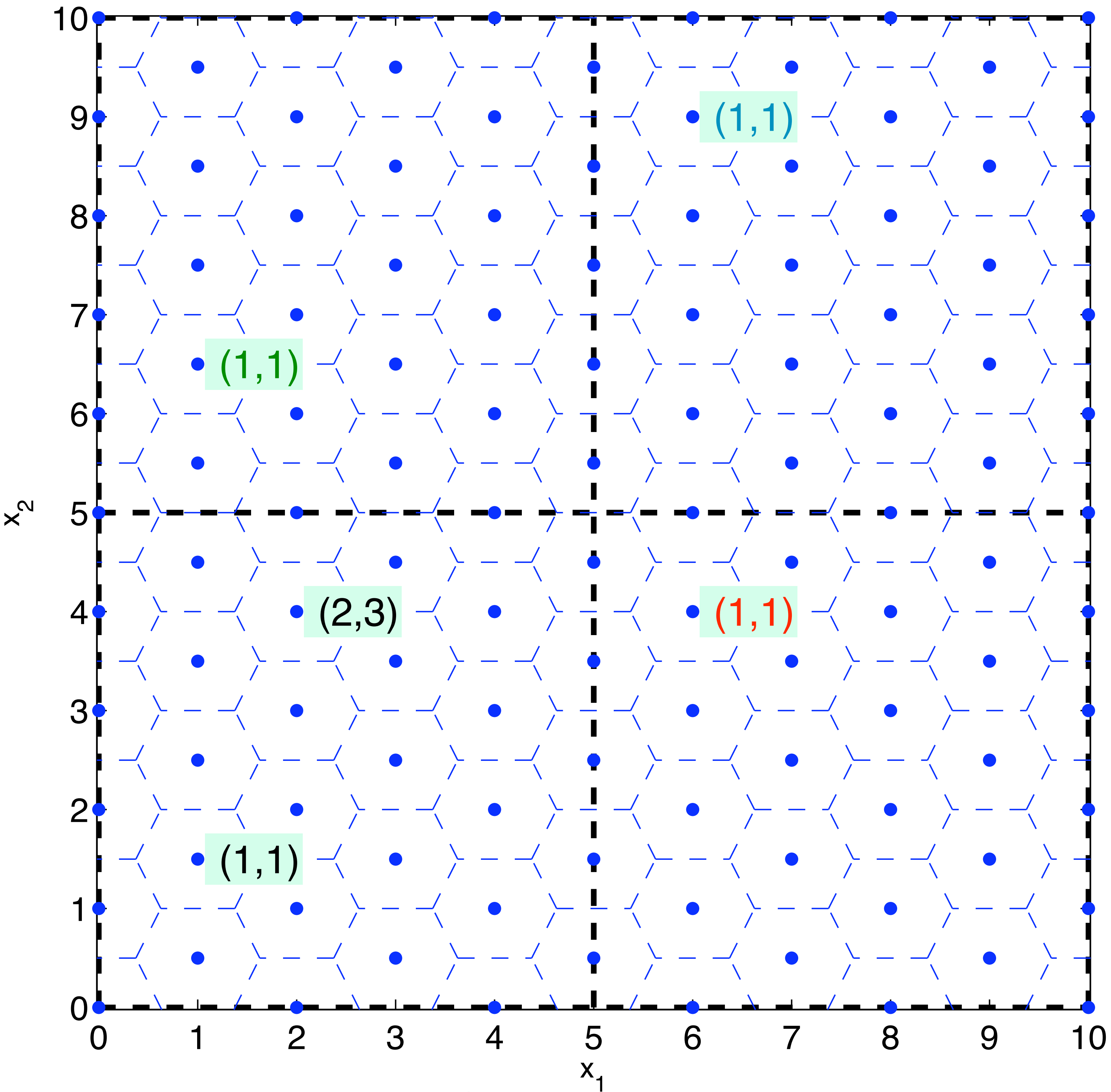
- Minimum number of writes is D

Note D is not related to n:

$$\begin{aligned} R &= \log_2 M \\ DM &= q - 1 \end{aligned}$$

Minimum number of writes is independent of the lattice dimension (block length)

Increasing the Average Number of Writes with a Random "hash" or permutation



Two code properties:

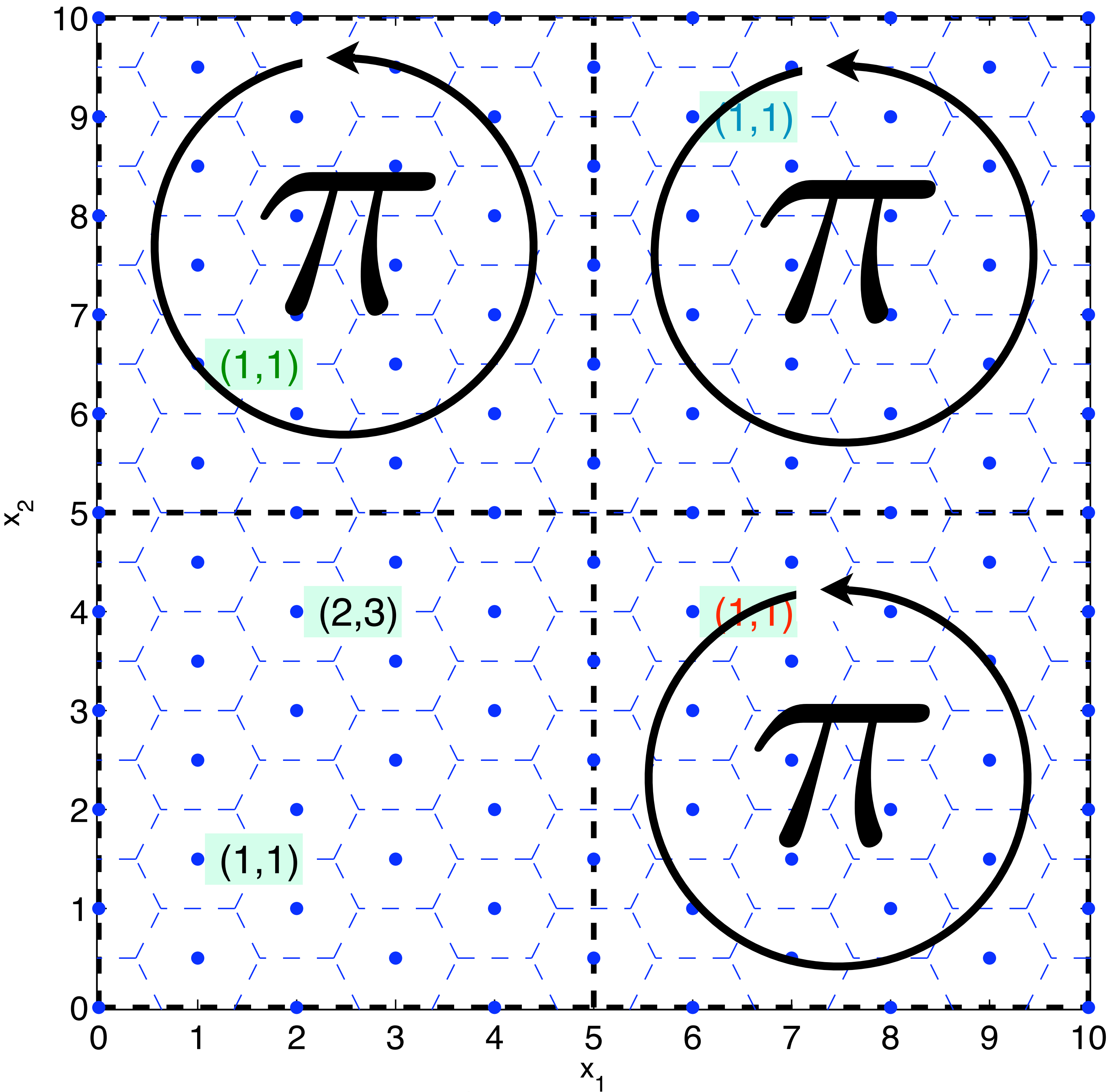
- triangular generator matrix
- code linearity

If (A) is not accessible, then (B) is not accessible

To increase the number of accessible points:

- each block gets a pseudo-random "hash" or permutation
- No linearity between blocks
- (in-block linearity remains)

Increasing the Average Number of Writes with a Random “hash” or permutation



Two code properties:

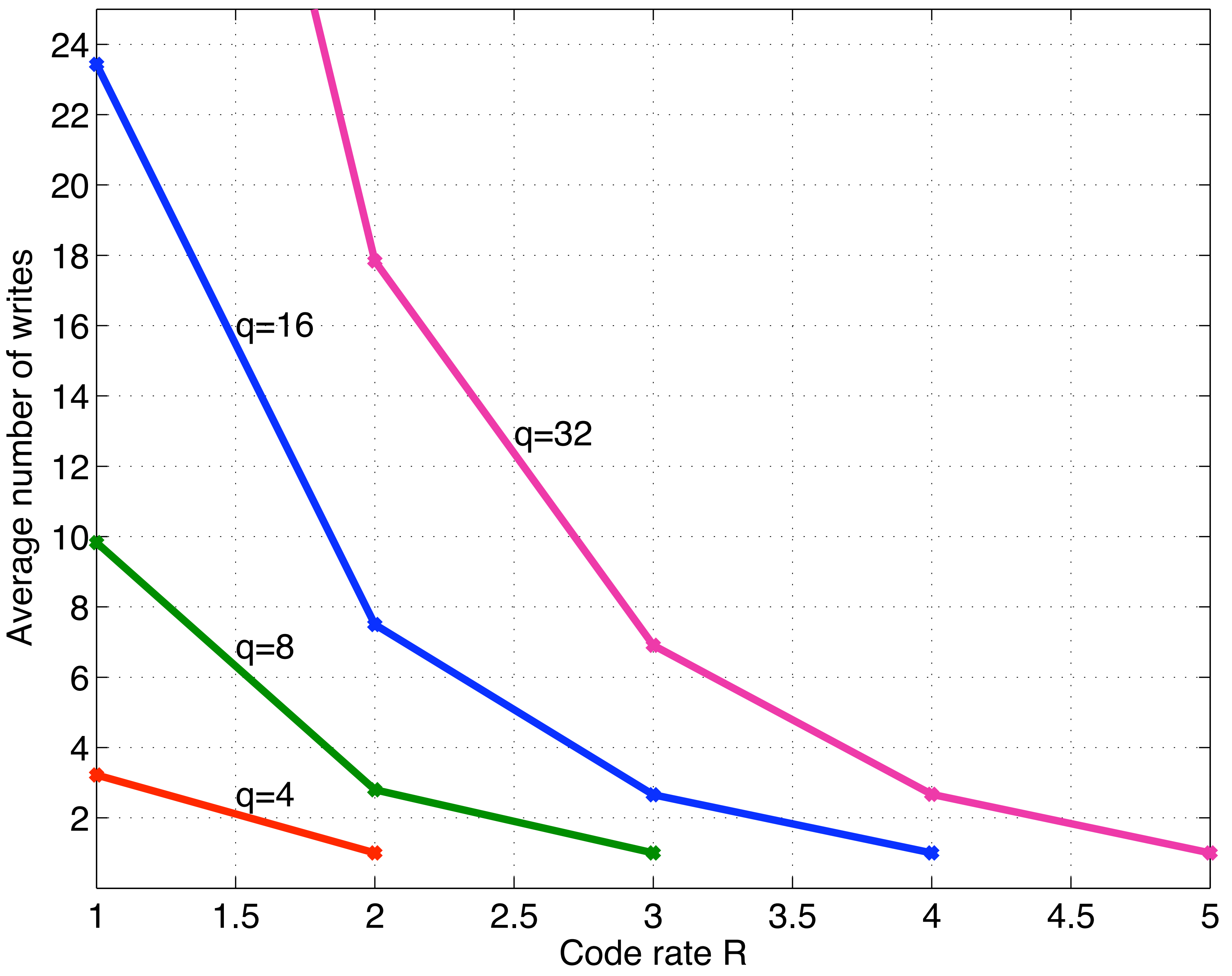
- triangular generator matrix
- code linearity

If (A) is not accessible, then (B) is not accessible

To increase the number of accessible points:

- each block gets a pseudo-random “hash” or permutation
- No linearity between blocks
- (in-block linearity remains)

Average Number of Writes Using E8 Lattice



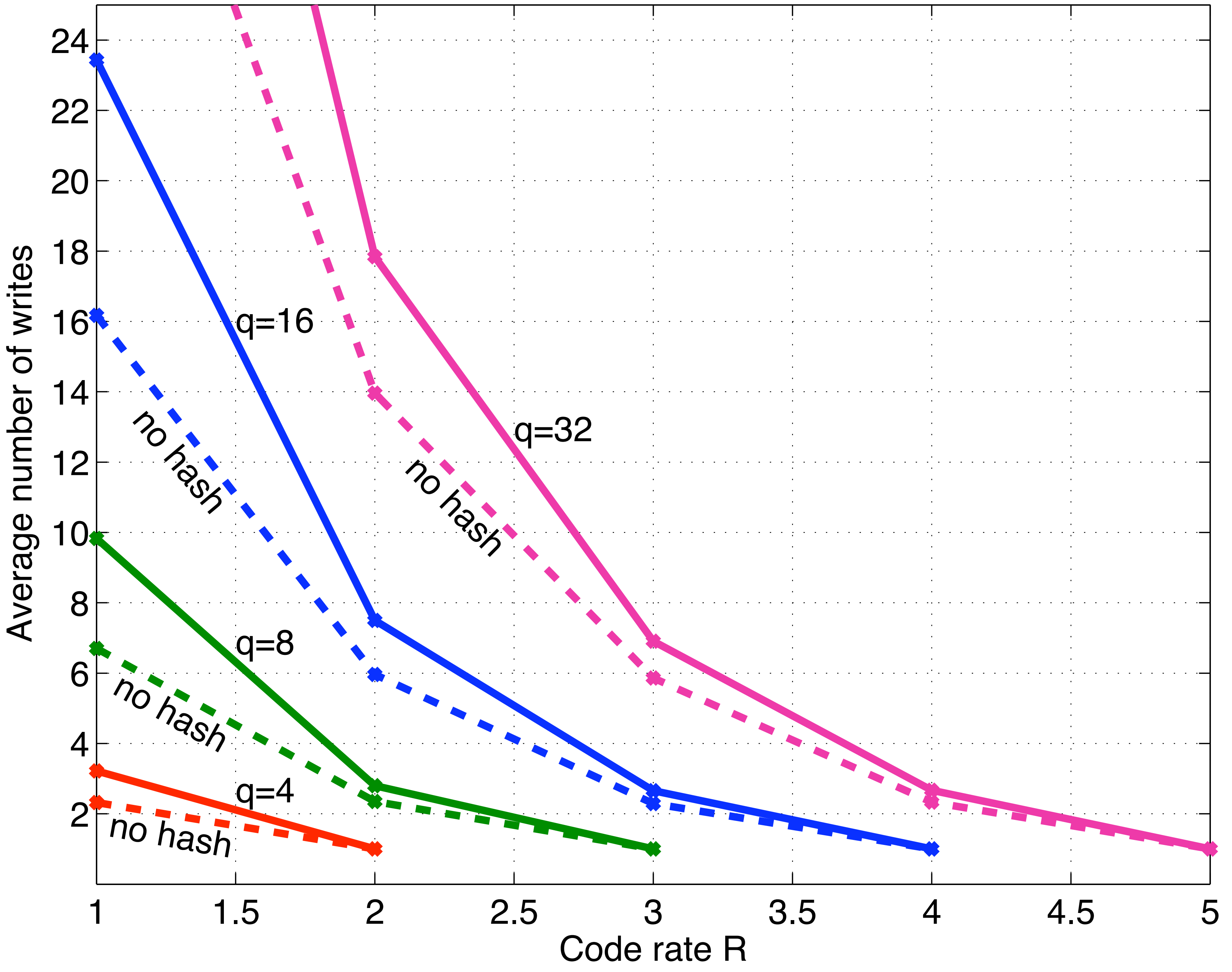
E8 lattice:

- best-known lattice in 8 dimensions
- triangular generator

Numerical evaluation:

- Rate-rewriting tradeoff
- rewriting capability increases in q
- High rate codes

Average Number of Writes Using E8 Lattice



E8 lattice:

- best-known lattice in 8 dimensions
- triangular generator

Numerical evaluation:

- Rate-rewriting tradeoff
- rewriting capability increases in q
- Construct high rate codes

The pseudo-random hash

- Helps at low rates
- Little effect at high rates

Notes and Caveats

Distinctions with existing rewriting codes

- Proposed construction rewrites information **words**.
- Existing codes rewrite information **bits**.

Lattices components are non-integer

- Read and write analog values in cells; voltage between 0 and V.
- E8 lattice has only integer and half-integer components

Looks like coded modulation (TCM...)

- Wireless AWGN channels have
  - an average power constraint: **Spherical shaping region**
  - synchronization required
- Flash memory has voltage range 0 to V (or $q - 1$):
  - peak power constraint**: Cubical shaping region**
  - inherently synchronized

Discussion

Showed that lattices can be used for rewriting in flash memories:

- Average number of writes **increases** in number of levels q
- Minimum number of writes **does not increase** in block length n

Fiat and Shamir (1984) used directed acyclic graph model:

"The significant improvement in memory capability is linear with the DAG depth. For a fixed number of states a 'deep and narrow' DAG cell is always preferable to a 'shallow and wide' DAG cell"

- deep and narrow: large q, small n
- shallow and wide; small q, large n

This observation is consistent with numerical results

"Reptiles"

M.C. Escher