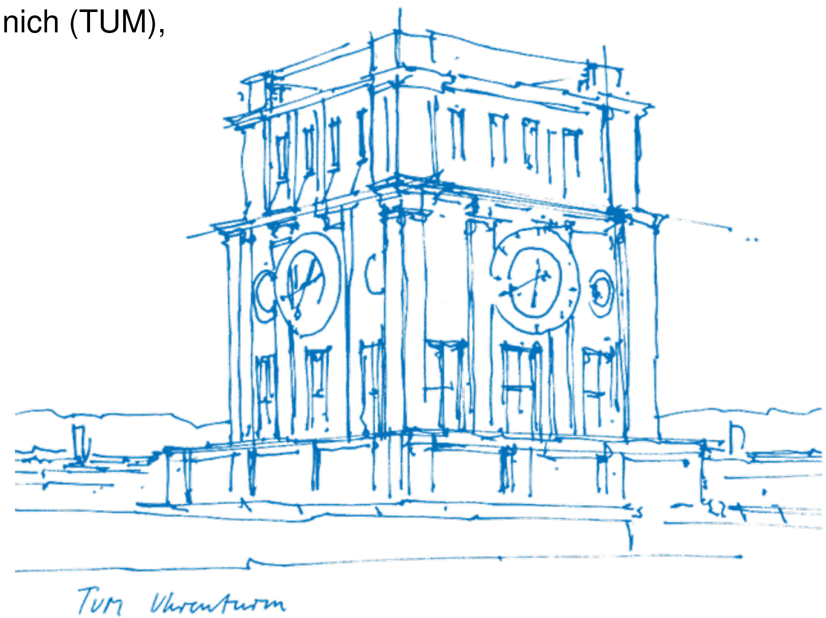
# Trading Partially Stuck Cells with Errors

Haider Al Kim, Sven Puchinger, Ludo Tolhuizen, Antonia Wachter-Zeh
Institute for Communications Engineering, Technical University of Munich (TUM), Germany

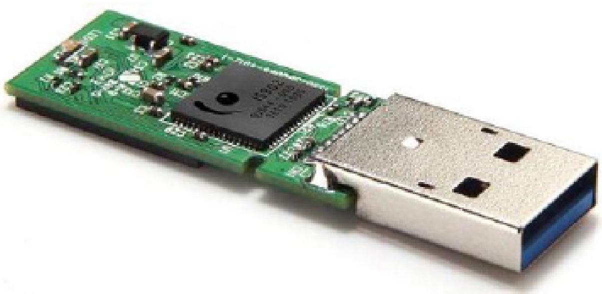
The 14th Annual Non-Volatile Memories Workshop (NVMW'23)
March 13-14, 2023



# Introduction - Codes for Non-Volatile Memories

A non-volatile memory is a memory that stores the information even when powered off.

## Multilevel Flash Memories

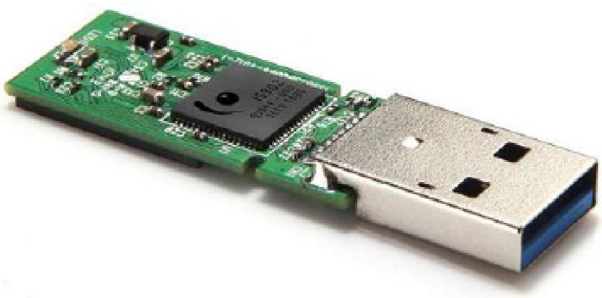


- Electronic charge represents multiple levels

# Introduction - Codes for Non-Volatile Memories

A non-volatile memory is a memory that stores the information even when powered off.

## Multilevel Flash Memories

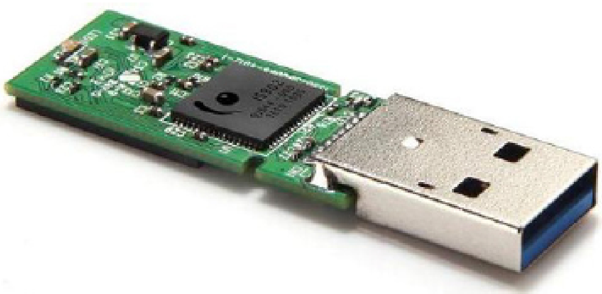


- Electronic charge represents multiple levels
- If charge is trapped, level can only be increased

# Introduction - Codes for Non-Volatile Memories

A non-volatile memory is a memory that stores the information even when powered off.

## Multilevel Flash Memories

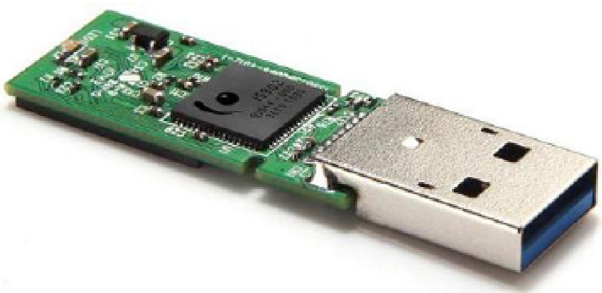


- Electronic charge represents multiple levels
- If charge is trapped, level can only be increased
- Cells can be defective (also called Stuck-at): cannot change their value

# Introduction - Codes for Non-Volatile Memories

A non-volatile memory is a memory that stores the information even when powered off.

## Multilevel Flash Memories

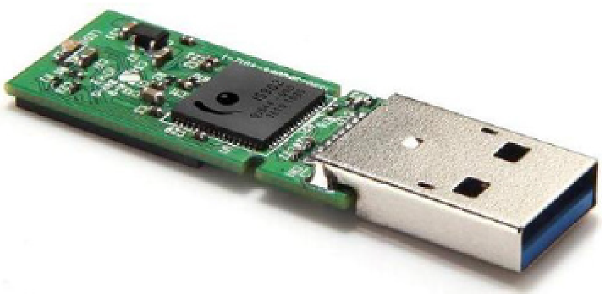


- Electronic charge represents multiple levels
- If charge is trapped, level can only be increased
- Cells can be defective (also called Stuck-at): cannot change their value
- Erasing blocks $\Rightarrow$ (slow and decreases life time)
  $\Longrightarrow$ avoid by only increasing levels in a new write

# Introduction - Codes for Non-Volatile Memories

A non-volatile memory is a memory that stores the information even when powered off.

## Multilevel Flash Memories

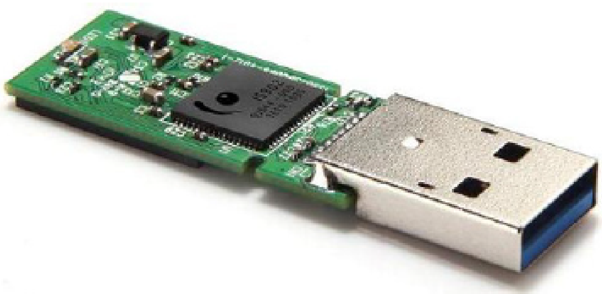


- Electronic charge represents multiple levels
- If charge is trapped, level can only be increased
- Cells can be defective (also called Stuck-at): cannot change their value
- Erasing blocks $\Rightarrow$ (slow and decreases life time)
  $\Longrightarrow$ avoid by only increasing levels in a new write
- Erasing state is the 0 level $\Longrightarrow$ (forbidden!)

# Introduction - Codes for Non-Volatile Memories

A non-volatile memory is a memory that stores the information even when powered off.

## Multilevel Flash Memories



- Electronic charge represents multiple levels
- If charge is trapped, level can only be increased
- Cells can be defective (also called Stuck-at): cannot change their value
- Erasing blocks $\Rightarrow$ (slow and decreases life time) $\Longrightarrow$ avoid by only increasing levels in a new write
- Erasing state is the 0 level $\Longrightarrow$ (forbidden!)
- Substitution errors can happen (only if a cell is partially defective or normal) $\Longrightarrow$ flipping levels of a cell

# (Partially) Stuck Memory Cells (SMC and PSMC)

**PDMC**: Introduced by **Kuznetsov and Tsybakov (1970)**

**Related Works: Heegard (1983); Gabrys, Sala, and Dolecek (2014); and Wachter-Zeh and Yaakobi (2016)**


[1] A. Kuznetsov and B. Tsybakov. “Coding for memories with defective cells,” in: (in Russian) Problems Inf. Transmiss. Vol. 10. 2. 1974, pp. 52–60.

[2] C. Heegard, “Partitioned Linear Block Codes for Computer Memory with’Stuck-at’Defects,” IEEE Transactions on Information Theory, vol. 29, no. 6, pp. 831–842, 1983.

[3] R. Gabrys, F. Sala, and L. Dolecek, “Coding for unreliable flash memory cells,” IEEE Commun. Lett., vol. 18, no. 9, pp. 1491–1494, Sep. 2014.

[4] A. Wachter-Zeh and E. Yaakobi, “Codes for Partially Stuck-at Memory Cells,” IEEE Transactions on Information Theory, vol. 62, no. 2, pp. 639–654, 2016.

# Full Version Papers

This work partially summarizes one of our constructions and bounds in [6] and [7] for tolerating partially stuck-at cells and correcting substitution error, and suggests treating some partially stuck cells as errors.


- [6] H. Al Kim, S. Puchinger, A. Wachter-Zeh "Bounds and Code Constructions for Partially Defect Memory Cells" Seventeenth International Workshop on Algebraic and Combinatorial Coding Theory (October 11-17, ACCT 2020)
- [7] H. Al Kim, S. Puchinger, L. Tolhuizen, and A. Wachter-Zeh, "Coding and Bounds for Partially Defective Memory Cells," (submitted to) the journal Designs, Codes and Cryptography, 2022. Arxiv version is here: https://arxiv.org/pdf/2202.07541.pdf.

# Stuck Vs Partially Stuck Memory Cells

## Stuck Cells (Classical Defects)

- Binary cells: cell can be stuck at level 0 or 1
- $q$-ary cells: cell can be stuck at any level $\boldsymbol{s} \in \mathbb{F}_q^n$
- A stuck cell cannot change its level!
- We "mask" our information by assuring the exact $s$ level to match the stuck positions
- Output: vector *vec* with $c_i = s_i$ for $i \in \phi$, where $\phi$ of size $u$ indexes the stuck positions.

# Stuck Vs Partially Stuck Memory Cells

## Stuck Cells (Classical Defects)

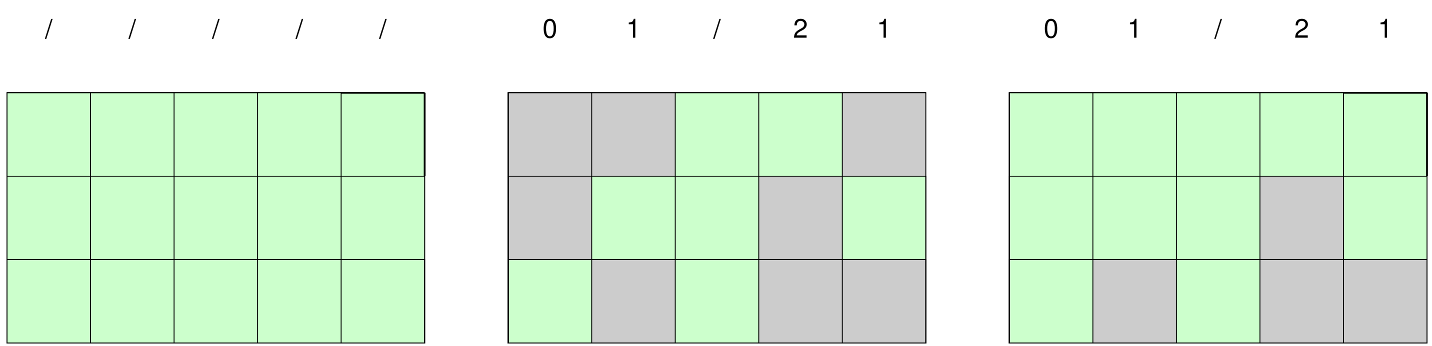
- Binary cells: cell can be stuck at level 0 or 1
- $q$-ary cells: cell can be stuck at any level $\boldsymbol{s} \in \mathbb{F}_q^n$
- A stuck cell cannot change its level!
- We "mask" our information by assuring the exact $s$ level to match the stuck positions
- Output: vector *vec* with $c_i = s_i$ for $i \in \phi$, where $\phi$ of size $u$ indexes the stuck positions.

## Partially Stuck Memory Cells

- $q$-ary cells: cell can be partially stuck at any level $ves \in \mathbb{F}_q^n$
- A partially stuck cell can only store levels at least $s$
- If $s = 0$: anything can be stored, (equivalent to) a normal cell
- Output: vector $\boldsymbol{c}$ with $c_i \geq s_i$ for $i \in \phi$, where $\phi$ of size $u$ indexes the partially stuck positions.

# Stuck Vs Partially Stuck Memory Cells

| | |
|---|---|
| (Partially) Stuck at - 2 | 2 |
| (Partially) Stuck at - 1 | 1 |
| (Partially) Stuck at - 0 | 0 |
| Normal Cell holds any value | / |
| The value that cell can hold | |
| No value can be stored | |



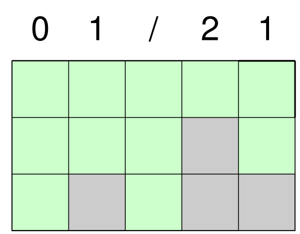

(A) Normal Cells (B) Stuck-at (C) Partially stuck-at

# A Scenario of a Memory with PSMC and Errors[1]

A scenario of $q = 2^{\mu}$ levels memory in which cells indexed by the support of the vector $\boldsymbol{s}$ denoted by $\phi \subseteq [n]$ of size $u$ are partially stuck (defective), so zeros are forbidden in these positions.

1. If substitution errors occur (e.g., due to inter-cell interference) in the partially defective cells, they concede to the partially defective constraints (i.e., $\{c_i + e_i \neq 0 \,|\, i \in \phi\}$).

2. Errors happen in the healthy cells or in the area above the partially stuck level ($s$), namely ($q - s$).




PSMC: Flipped 1 $\rightarrow$ 2 or 2 $\rightarrow$ 1

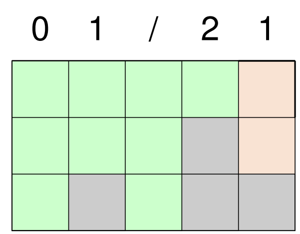
[1] H. Al Kim, S. Puchinger, and A. Wachter-Zeh, 2020

# A Scenario of a Memory with PSMC and Errors[1]

A scenario of $q = 2^{\mu}$ levels memory in which cells indexed by the support of the vector $\boldsymbol{s}$ denoted by $\phi \subseteq [n]$ of size $u$ are partially stuck (defective), so zeros are forbidden in these positions.

1. If substitution errors occur (e.g., due to inter-cell interference) in the partially defective cells, they concede to the partially defective constraints (i.e., $\{c_i + e_i \neq 0 \,|\, i \in \phi\}$).

2. Errors happen in the healthy cells or in the area above the partially stuck level ($s$), namely ($q - s$).




PSMC: Flipped 1 $\rightarrow$ 2 or 2 $\rightarrow$ 1

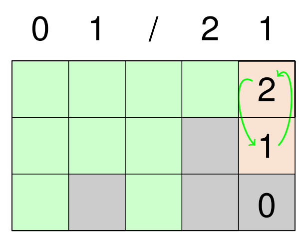
[1] H. Al Kim, S. Puchinger, and A. Wachter-Zeh, 2020

# A Scenario of a Memory with PSMC and Errors[1]

A scenario of $q = 2^{\mu}$ levels memory in which cells indexed by the support of the vector $\boldsymbol{s}$ denoted by $\phi \subseteq [n]$ of size $u$ are partially stuck (defective), so zeros are forbidden in these positions.

1. If substitution errors occur (e.g., due to inter-cell interference) in the partially defective cells, they concede to the partially defective constraints (i.e., $\{c_i + e_i \neq 0 \,|\, i \in \phi\}$).

2. Errors happen in the healthy cells or in the area above the partially stuck level ($s$), namely ($q - s$).




PSMC: Flipped $1 \rightarrow 2$ or $2 \rightarrow 1$

[1] H. Al Kim, S. Puchinger, and A. Wachter-Zeh, 2020

# Our Contributions

1. **Code construction**:

# Our Contributions

1. **Code construction**:

- Presenting code construction for masking partially stuck cells while additionally correcting errors [6] and [7].

# Our Contributions

1. **Code construction**:

- Presenting code construction for masking partially stuck cells while additionally correcting errors [6] and [7].

- The process of "masking" finds a word whose entries coincide with writable levels at the (partially) stuck cells.

# Our Contributions

1. **Code construction**:

- Presenting code construction for masking partially stuck cells while additionally correcting errors [6] and [7].
- The process of "masking" finds a word whose entries coincide with writable levels at the (partially) stuck cells.
- Deriving Gilbert-Varshamov-type bound for our code construction [6] and [7].

# Our Contributions

2. **Trading (some) partially stuck cells as errors**:

# Our Contributions

2. **Trading (some) partially stuck cells as errors**:

- Instead of masking PSMC directly by our construction, PSMC can be partially tolerated using the error correction capability of the code.

# Our Contributions

2. **Trading (some) partially stuck cells as errors**:

- Instead of masking PSMC directly by our construction, PSMC can be partially tolerated using the error correction capability of the code.

- Providing a general theorem for the exchange of $j$ errors with $j$ masked cells.

# Our Contributions

2. **Trading (some) partially stuck cells as errors**:

- Instead of masking PSMC directly by our construction, PSMC can be partially tolerated using the error correction capability of the code.
- Providing a general theorem for the exchange of $j$ errors with $j$ masked cells.
- Improving this theorem by introducing two lemmas (fewer errors can be corrected $\longleftrightarrow$ more PSMCs can be treated).

# Our Contributions

2. **Trading (some) partially stuck cells as errors**:

- Instead of masking PSMC directly by our construction, PSMC can be partially tolerated using the error correction capability of the code.
- Providing a general theorem for the exchange of $j$ errors with $j$ masked cells.
- Improving this theorem by introducing two lemmas (fewer errors can be corrected $\longleftrightarrow$ more PSMCs can be treated).

Numerical Analysis :

# Our Contributions

2. **Trading (some) partially stuck cells as errors**:

- Instead of masking PSMC directly by our construction, PSMC can be partially tolerated using the error correction capability of the code.
- Providing a general theorem for the exchange of $j$ errors with $j$ masked cells.
- Improving this theorem by introducing two lemmas (fewer errors can be corrected $\longleftrightarrow$ more PSMCs can be treated).

## Numerical Analysis :

1. We check if our construction for any $u$ partially stuck cells $\leq n$ satisfy Gilbert-Varshamov bound.

# Our Contributions

2. **Trading (some) partially stuck cells as errors**:

- Instead of masking PSMC directly by our construction, PSMC can be partially tolerated using the error correction capability of the code.
- Providing a general theorem for the exchange of $j$ errors with $j$ masked cells.
- Improving this theorem by introducing two lemmas (fewer errors can be corrected $\longleftrightarrow$ more PSMCs can be treated).

Numerical Analysis :

1. We check if our construction for any $u$ partially stuck cells $\leq n$ satisfy Gilbert-Varshamov bound.
2. We compare the direct application of our construction with the general theorem of trading that further compared to improving Lemmas.

# Code Construction over $\mathbb{F}_{2^\mu}$

# Code Construction over $\mathbb{F}_{2^\mu}$, $\mu > 1$

## Construction 1

Let $\mu > 1$. Suppose $\boldsymbol{G}$ is a $k \times n$ generator matrix of an $[n, k, d]_{2^\mu}$ code $\mathcal{C}$ of the form

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{G}_1 \\ \boldsymbol{x} \end{bmatrix} \quad (1)$$

where

1. $\boldsymbol{H}_0 \in \mathbb{F}_2^{l \times n}$ is a parity check matrix of an $[n, n-l, d_0]_2$ code $\mathcal{C}_0$,
2. $\boldsymbol{G}_1 \in \mathbb{F}_{2^\mu}^{(k-l-1) \times n}$,
3. $\boldsymbol{x} \in \mathbb{F}_{2^\mu}^{1 \times n}$ has Hamming weight $n$.

From the code $\mathcal{C}$, a PSMC can be obtained, whose encoder and decoder are shown in Algorithm 5 and Algorithm 6 in [7].

## Theorem 1

The coding scheme in Construction 1 is a $2^\mu$-ary $(2^{\mu-1}d_0 - 1, 1, \lfloor \frac{d-1}{2} \rfloor)$ PSMC of length $n$ and cardinality $2^{\mu(k-l-1)} 2^{l(\mu-1)}$.

# Encoding and Decoding - Theorem 1

**Algorithm 1: Encoding ($\boldsymbol{m}$; $\boldsymbol{m}'$; $\phi$)**

**Input:**

- Message:
  $(\boldsymbol{m}', \boldsymbol{m}) \in \mathcal{F}^l \times \mathbb{F}_{2^\mu}^{k-l-1}$, where
  $\mathcal{F} = \{\sum_{i=1}^{\mu-1} x_i\beta_i \mid (x_1, \ldots, x_{\mu-1}) \in \mathbb{F}_2^{\mu-1}\}$.
- Positions of partially stuck-at-1 cells: $\phi$
- Notions introduced in Construction 1.

1. $\boldsymbol{w} \leftarrow \boldsymbol{m}' \cdot \boldsymbol{H}_0 + \boldsymbol{m} \cdot \boldsymbol{G}_1 + z \cdot \boldsymbol{x}$ where $z \in \mathbb{F}_{2^\mu}$ is chosen such that $|\{i \in \phi \mid w_i \in \mathbb{F}_2\}| \leq d_0 - 1$.
2. Choose $\gamma \in \mathbb{F}_2^l$ such that $(\gamma \boldsymbol{H}_0)_i = 1 - \boldsymbol{w}_i$ for all $i \in \phi$ for which $\boldsymbol{w}_i \in \mathbb{F}_2$.

**Output:** Codeword $\boldsymbol{c} = \boldsymbol{w} + \gamma \cdot \boldsymbol{H}_0 \in \mathcal{C}$

# Encoding and Decoding - Theorem 1

## Algorithm 2: Decoding

**Input:**

- $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e} \in \mathbb{F}_{2^\mu}^n$, where $\boldsymbol{c}$ is a valid output of Algorithm 1 and $\boldsymbol{e}$ is an error of Hamming weight at most $t$.
- Notions introduced in Construction 1.
- $\hat{\boldsymbol{c}} \leftarrow$ decode $\boldsymbol{y}$ in the code $\mathcal{C}$

1. $\hat{\boldsymbol{c}} \leftarrow$ decode $\boldsymbol{y}$ in the code $\mathcal{C}$
2. Obtain $\boldsymbol{a} \in \mathbb{F}_{2^\mu}^l$, $\hat{\boldsymbol{m}} \in \mathbb{F}_{2^\mu}^{k-l-1}$, $\hat{z} \in \mathbb{F}_{2^\mu}$ such that $\hat{\boldsymbol{c}} = \boldsymbol{a}\boldsymbol{H}_0 + \hat{\boldsymbol{m}}\boldsymbol{G}_1 + \hat{z}\boldsymbol{x}$.
3. Obtain $\hat{\boldsymbol{m}}' \in \mathcal{F}^{k-l-1}$ and $\hat{\gamma} \in \mathbb{F}_2^{k-l-1}$ such that $\boldsymbol{a} = \hat{\boldsymbol{m}}' + \hat{\gamma}$.

**Output:** Message vector $(\hat{\boldsymbol{m}}, \hat{\boldsymbol{m}}')$

## Trading PSMCs with Errors

(General Theorem)

# Partial Masking of Partially Stuck Memory Cells

**Proposition 1**

If there is an $(n, M)_q(u, 1, t)$ PSMC, then for any $j$ with $0 \leq j \leq t$, there is an $(n, M)_q(u + j, 1, t - j)$ PSMC.

**Theorem 2**

Let $\Sigma \subset \mathbb{F}_q^n$, and assume that there exists an $(n, M)_q(\Sigma, t)$ PSMC $\mathcal{C}$. For any $j \in [t]$, there exists an $(n, M)_q(\Sigma^{(j)}, t - j)$ PSMC $\mathcal{C}_j$, where

$$\Sigma^{(j)} = \left\{ \boldsymbol{s}' \in \mathbb{F}_q^n \mid \exists \boldsymbol{s} \in \Sigma \left[ d(\boldsymbol{s}, \boldsymbol{s}') \leq j \text{ and } \boldsymbol{s}' \geq \boldsymbol{s} \right] \right\}.$$

# Encoding and Decoding Algorithms of Theorem 2

**Algorithm 3 - Encoder $\mathcal{E}_j$**

**Input:** $(\mathbf{m}, \mathbf{s}') \in \mathcal{M} \times \Sigma^{(j)}$.

1. Determine $\mathbf{s} \in \Sigma$ such that $d(\mathbf{s}, \mathbf{s}') \leq j$ and $\mathbf{s}' \geq \mathbf{s}$.
2. Let $\mathbf{c} = \mathcal{E}(\mathbf{m}, \mathbf{s})$.
3. Define $\mathbf{c}' = \mathcal{E}_j'(\mathbf{m}, \mathbf{s}')$ as $c_i' = \max(c_i, s_i')$ for $i \in [n]$.

**Output:** Codeword $\mathbf{c}' \in \mathbb{F}_q^n$.

**Algorithm 4 - Decoder $\mathcal{D}_j$**

**Input:** Retrieved $\mathbf{y} = \mathbf{c}' + \mathbf{e}$ where $wt(\mathbf{e}) \leq t - j$ and $\mathbf{y} \geq \mathbf{s}'$

1. Message $\mathbf{m} = \mathcal{D}(\mathbf{y})$

**Output:** Message vector $\mathbf{m}$

# Proof of Theorem 2

## Proof of Theorem 2

1. Let the encoder $\mathcal{E}_j$ and the decoder $\mathcal{D}_j$ for $\mathcal{C}_j$ be Algorithm 3 and Algorithm 4, respectively.
2. By definition, $\mathbf{c}' \geq \mathbf{s}'$.
3. Moreover, if $s_i = s_i'$, then $c_i \geq s_i = s_i'$, so $c_i = c_i'$.
4. As a result, $d(\mathbf{c}, \mathbf{c}') \leq j$.
5. In Algorithm 4, the decoder $\mathcal{D}$ of $\mathcal{C}$ is directly used for decoding $\mathcal{C}_j$.
6. As $\mathbf{y} \geq \mathbf{s}'$, surely $\mathbf{y} \geq \mathbf{s}$.
7. Moreover, we can write $\mathbf{y} = \mathbf{c} + (\mathbf{c}' - \mathbf{c} + \mathbf{e})$.
8. As shown above, $\mathrm{wt}(\mathbf{c}' - \mathbf{c}) \leq j$, and so $wt(\mathbf{c} - \mathbf{c}' + \mathbf{e}) \leq t$.
9. As a consequence, $\mathcal{D}(\mathbf{y}) = \mathbf{m}$.

# Trading PSMCs with Errors

(Improvements on the General Theorem)

# Improvements on Theorem 2

Improving on Theorem 2 for Construction 1 by the following Lemma.

**Lemma 1**

Given an $[n, k, d]_q$ code as defined in [4, Construction 3], then for any $j$ such that $0 \leq j \leq \lfloor \frac{d-1}{2} \rfloor$, there is a $2^\mu$-ary $(2^{\mu-1}(d_0 + j) - 1, 1, \lfloor \frac{d-1}{2} \rfloor - j)$ PSMC of length $n$ and size $q^{k-l-1}$.

**Proof of Lemma 1**

1. Let $\phi \subset [n]$ has size $u \leq 2^{\mu-1}(d_0 + j) - 1$.
2. We use the notation from Algorithm 1.
3. After Step 1, $\boldsymbol{w}$ has at most $u_0 = \lfloor \frac{2u}{2^\mu} \rfloor \leq d_0 + j - 1$ binary entries in the positions from $\phi$.
4. After Step 2, at least $d_0 - 1$ of these entries in $\boldsymbol{c}$ differ from 0.
5. By setting the at most $j$ other binary entries in the positions from $\phi$ equal to 1, the encoder introduces at most $j$ errors, and guarantees that the partially-stuck-at conditions are satisfied.

# Improvements on Theorem 2

Another approach for introducing errors in order to satisfy the stuck-at conditions

**Lemma 2**

Given an $[n, k, d]_q$ code containing a word of weight $n$, for any $j$ with $0 \leq j \leq \lfloor \frac{d-1}{2} \rfloor$, there is a $q$-ary $(q-1+qj, 1, \lfloor \frac{d-1}{2} \rfloor - j)$ PSMC of length $n$ and size $q^{k-1}$.

**Proof of Lemma 2**

1. Let $\phi \subset [n]$ have size $u \leq q-1+qj$, and $\boldsymbol{x}$ be a codeword of weight $n$.
2. For each $i \in \phi$, there is exactly one $v \in \mathbb{F}_q$ such that $w_i + vx_i = 0$, and so

$$\sum_{v \in \mathbb{F}_q} |\{i \in \phi \mid w_i + vx_i = 0\}| = u.$$

3. Consequently, there is $v \in \mathbb{F}_q$ such that $\boldsymbol{c} = \boldsymbol{w} + v\boldsymbol{x}$ has at most $\lfloor \frac{u}{q} \rfloor \leq j$ entries in $\phi$ equal to zero.
4. By setting these entries of $\boldsymbol{c}$ to a non-zero value, the encoder introduces at most $j$ errors.
5. As $\mathcal{C}$ can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors, it can correct these $j$ errors and additionally up to $\lfloor \frac{d-1}{2} \rfloor - j$ substitution errors.

# Numerical Comparisons

(Gilbert–Varshamov Bound on PSMCs)

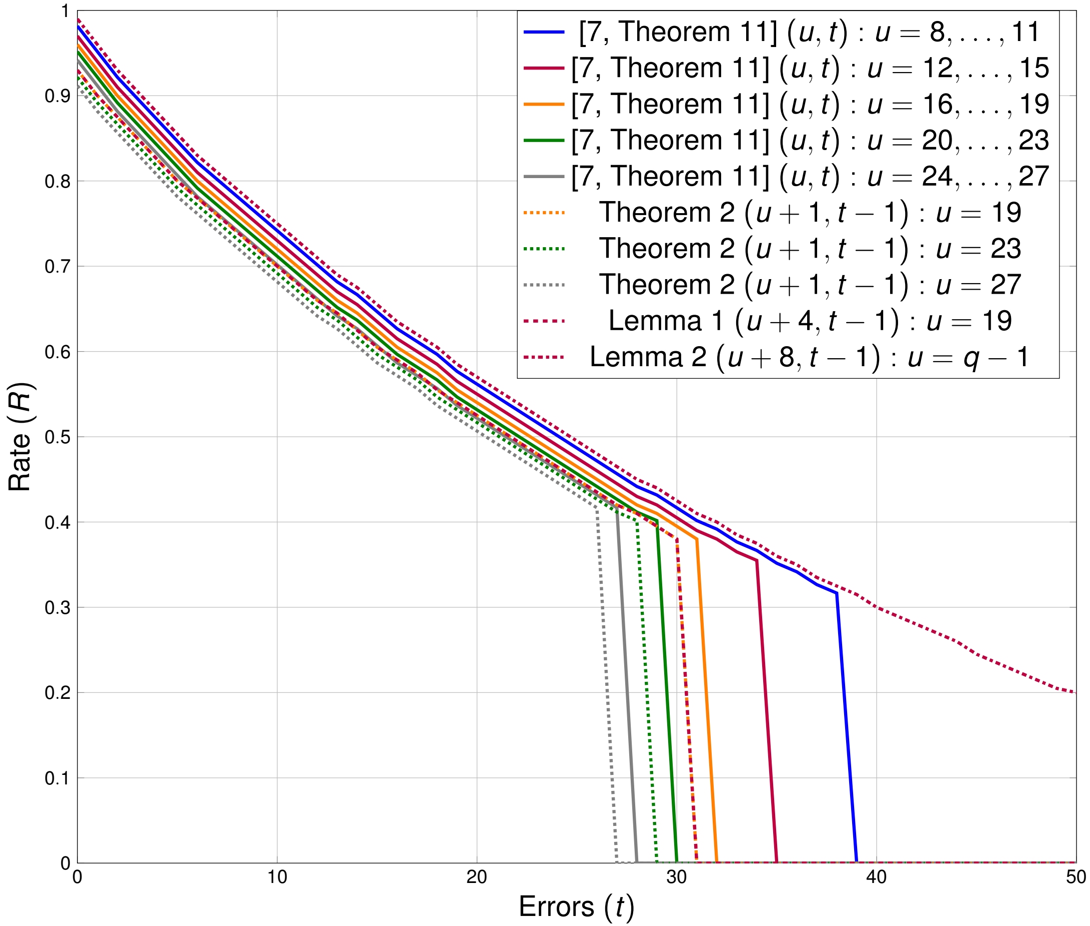



Figure: The achievable rates $R = \frac{1}{n}\log_{2^3}\mathcal{M}$ of GV bounds for different $u$, and $t$ for $n = 200$ and $q = 2^3$ in [7, Theorem 11]. The solid plots are the rates from the derived GV like bound and the dotted lines are the rates after trading $u+1, t-1$ by Theorem 2. Trading one correctable error by Lemma 1 and Lemma 2 increases $u$ by $2^{\mu-1}$ and $2^{\mu}$, respectively. Lemma 2 gives slightly higher rates for *all* $t \leq 50$ while treating the same number of $u$ cells compared to the corresponding curves from [7, Theorem 11], Theorem 2, and Lemma 1.

# Summary

- This work considers coding for *partially stuck* memory cells.
- Such memory cells can only store partial information as some of their levels cannot be used due to, e.g., wearout.
- We proposed a $2^{\mu}$-ary partially stuck cell code construction for *masking* partially stuck cells while correcting substitution errors.
- We formulated a GV-like bound (found in [7, Theorem 11] on the cardinality and the minimum distance.
- We investigated a technique where the encoder, after a first masking step, introduces errors at some partially stuck positions of a codeword in order to satisfy the stuck-at constraints.
- It turns that treating some of the partially stuck cells as erroneous cells can decrease the required redundancy for some parameters, e.g., by Lemma 2.

Thank You

Any questions ... ?