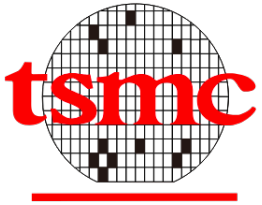
PBA: Percentile-Based Level Allocation for Multiple-Bits-Per-Cell RRAM

Anjiang Wei, Akash Levy, Pu (Luke) Yi, Robert M. Radway,
Priyanka Raina, Subhasish Mitra, Sara Achour

anjiang@stanford.edu

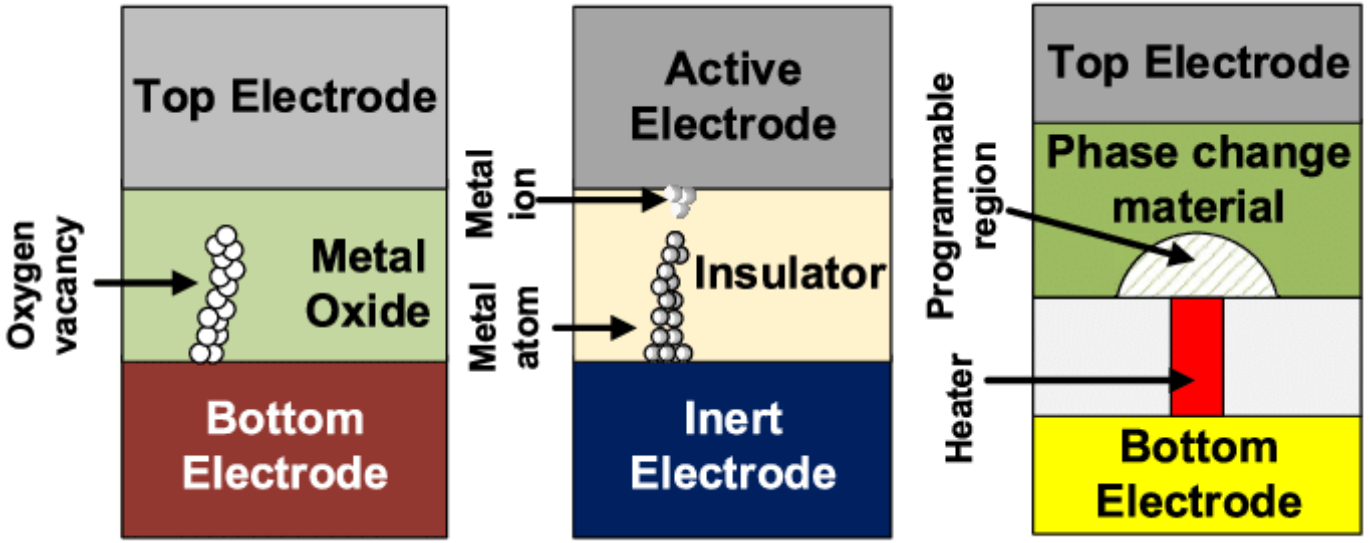


Stanford | ENERGY

SAMSUNG

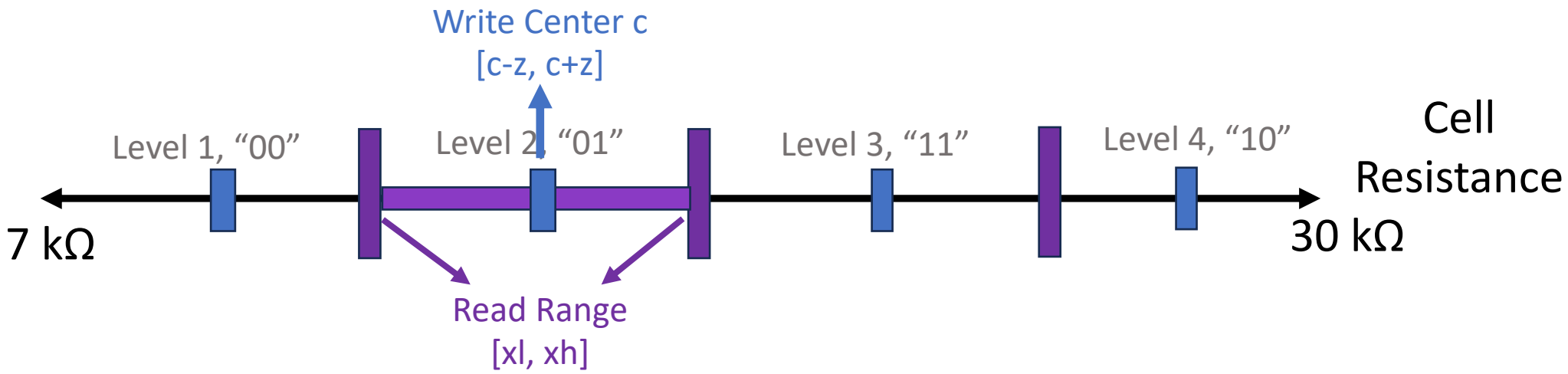
Emerging Non-Volatile Memories

- Emerging non-volatile memory technologies
  - Resistive random-access memory (**RRAM** or **ReRAM**)
  - Conductive bridge random access memory (CBRAM)
  - Phase-change memory (PCM)
- Multiple-bits-per-cell storage for higher memory density



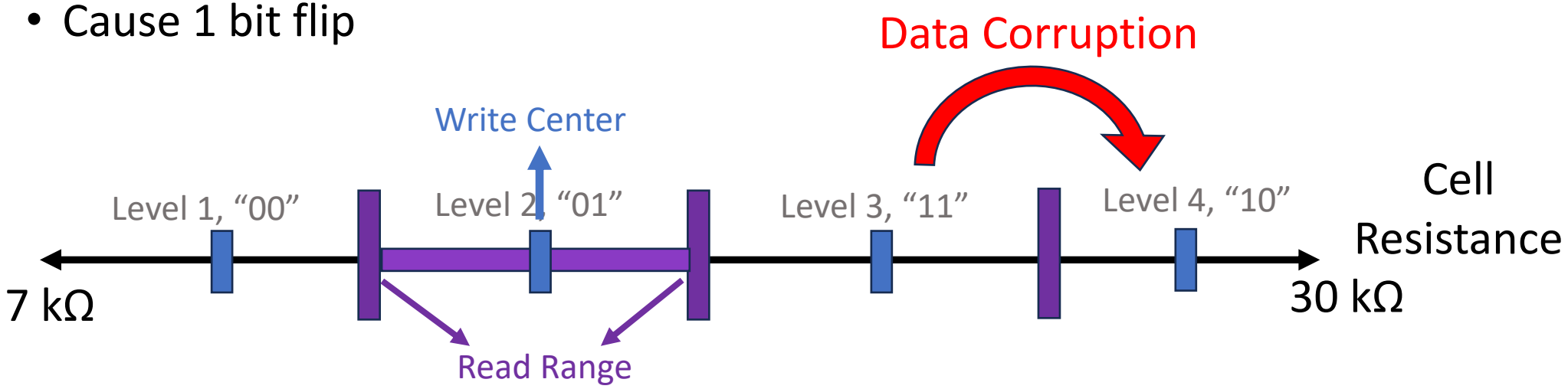
Level Allocation

- Map bit combinations to resistance ranges
  - Example: 2-bits-per-cell needs 4 levels
- Write center c
  - Program-and-verify algorithm; “write tolerance z” set by expert
- Read range [xl, xh]



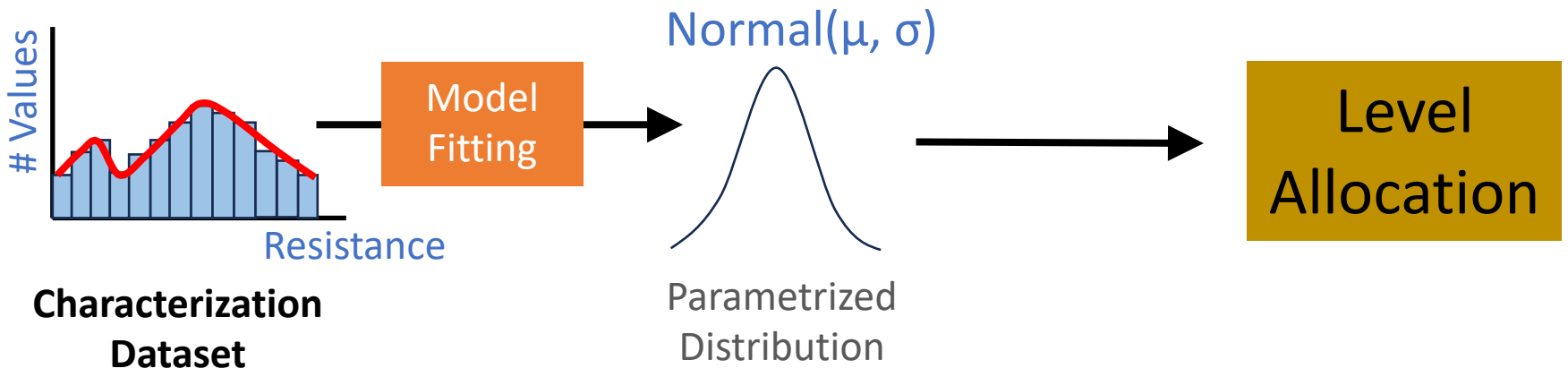
Data Corruption

- Hardware non-idealities
  - Read noise, process variation, relaxation (over time), etc.
- Example
  - Write to Level 3, “11”
  - Read out from Level 4, “10”
  - Cause 1 bit flip



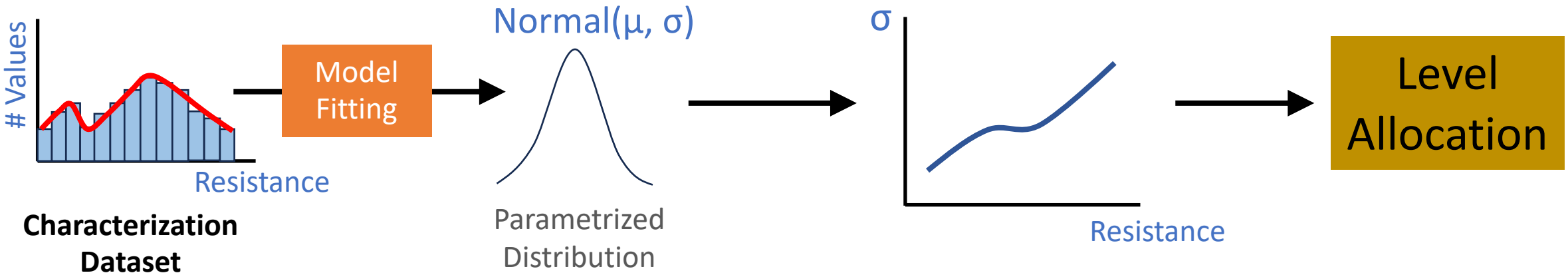
Prior Work on Level Allocation

- Collect characterization datasets
- Fit a common, parameterized distribution
  - E.g., normal or lognormal
- Produce level allocation using distribution parameters
  - E.g., standard deviation σ



Prior Work on RRAM Level Allocation

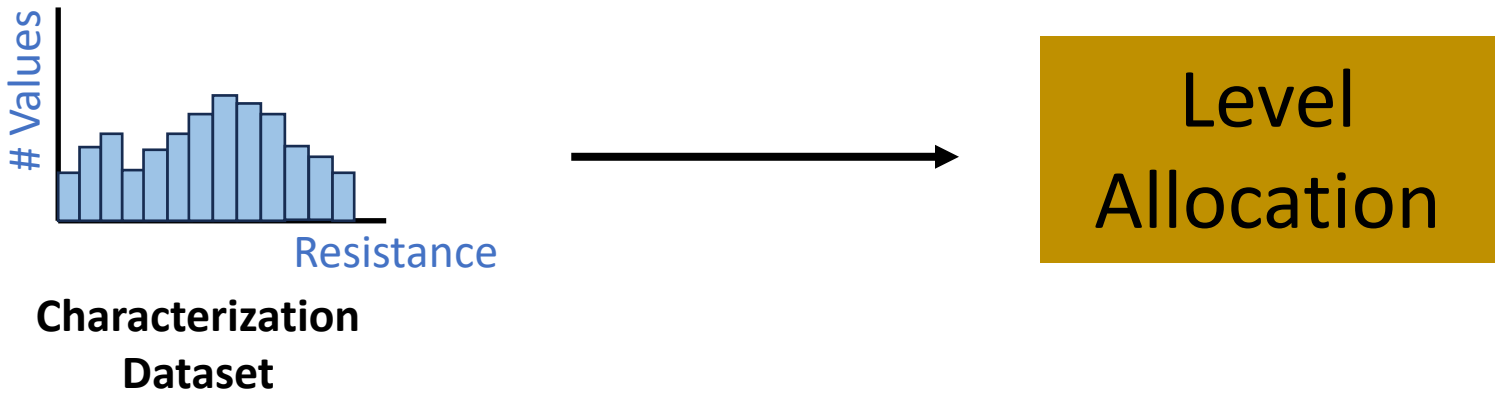
- SBA[1]: Sigma-Based Allocation
- Fit data with normal distribution
  - Standard deviation σ is a function of resistance
- Such approximation misrepresents analog behaviors
  - The raw data points are highly non-normal



[1]Le *et al*. "Resistive RAM with multiple bits per cell: Array-level demonstration of 3 bits per cell." TED 2018.

PBA: Percentile-Based Allocation

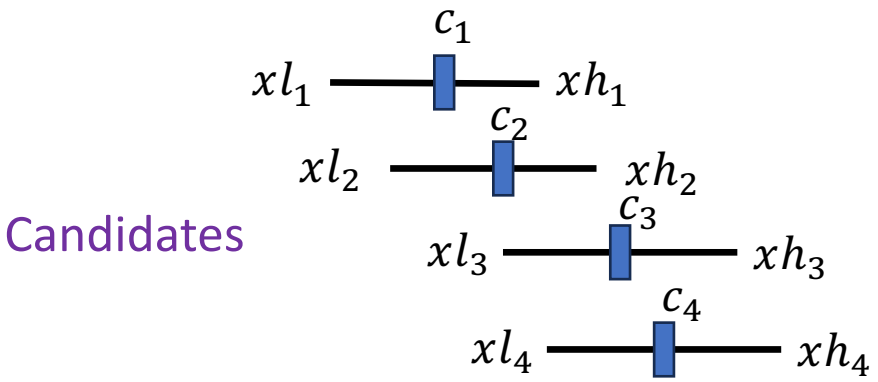
- Directly work with the characterization data
- No fitting a parameterized model, no approximation
- More accurately capture analog behaviors present in the data

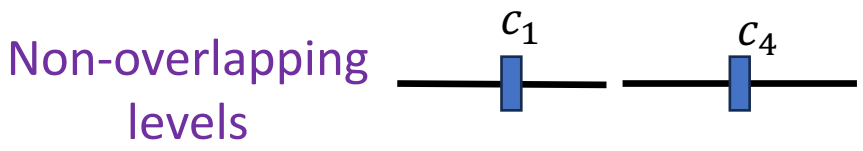


How does PBA work?

PBA Level Allocation Algorithm

- Input:
  - n # number of levels to allocate
  - ε # minimum granularity of error





Function LevelAlloc(n, ε):

The loop can be binary search

for $\gamma \in [0, \varepsilon, 2\varepsilon, \ldots, 1]$ **do**

Candidates = CandidateGen(γ)

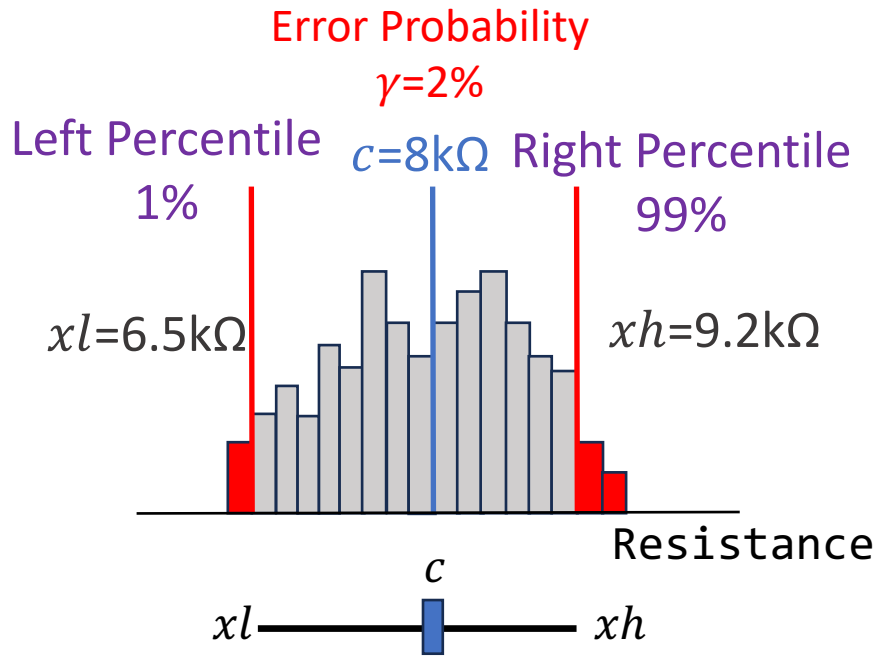
Result = FindNonOverlap(Candidates)

if Result.length == n **then**

return Result

Candidate Level Generation

- From dataset
  - C # list of write centers
  - t # relaxation time
- Example candidate:

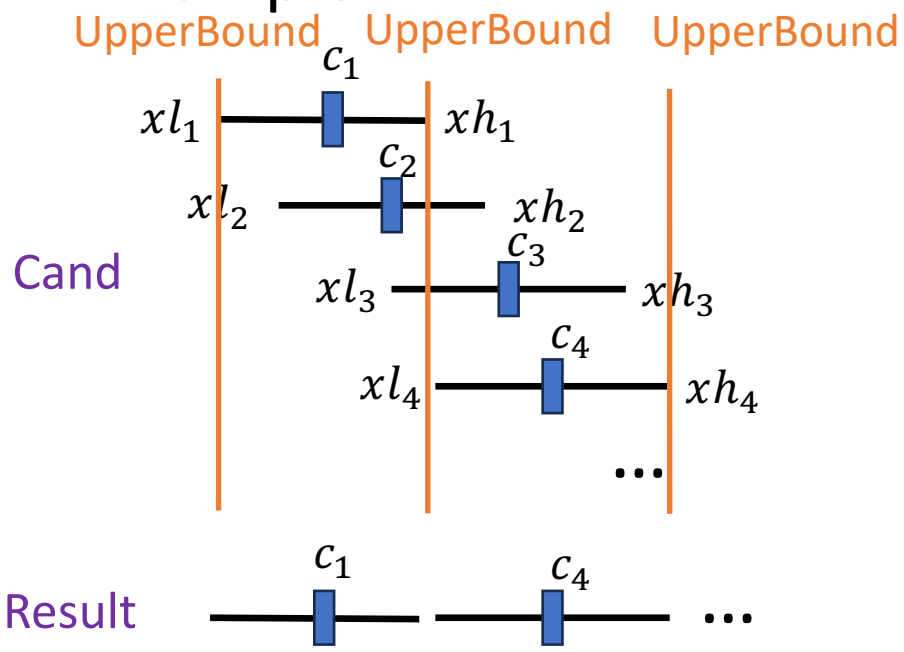


Function CandidateGen (γ):
    Candidates = []
    for c **in** C **do**
        R = GetData(c, t) # List of resistance data points
        xl = Percentile(R, $\frac{1}{2}\ \gamma$)
        xh = Percentile(R, $1 - \frac{1}{2}\gamma$)
        Candidates.append($\langle c, xl, xh \rangle$)
    return Candidates

Function Percentile(R, perc):
    Rsorted = sort(R) # Sort points in increasing order
    index = perc × size(R)
    return Rsorted[index]

Find Maximum Non-Overlapping Levels

- Example



- Optimality proof:
  - Find maximum number of levels

Function FindNonOverlap ($Cand$):

```
Result = []  # Non-overlapping levels
SortCand = sort(Cand, key=xh)  # Sort by xh
UpperBound = 0
for ⟨c, xl, xh⟩ ∈ SortCand do
    if xl ≥ UpperBound then
        Result.append(⟨c, xl, xh⟩)
        UpperBound = xh  # Update the bound
return Result
```

Evaluation

Experimental Setup

- Evaluate on 3 fabricated RRAM storage arrays

Chip	# Total Cells	Readout	Resistance	# Tested write centers	# Tested cells
Sapiens[1]	64k	Off-chip	7.8 - 40 kΩ	32	Random 200
Ember[2] 1	3M	On-chip ADC	1 – 64 levels	64	16k
Ember 2	3M	On-chip ADC	1 – 64 levels	64	16k

- Baseline: Sigma-Based Allocation (SBA)
- 4-level (2 bits-per-cell) and 8-level (3 bits-per-cell) allocations

[1] Li *et al*. “SAPIENS: A 64-kb RRAM-based non-volatile associative memory for one-shot learning and inference at the edge.” TED 2021.
[2] Upton *et al*. “EMBER: a 100 MHz, 0.86mm2, Multiple-Bits-per-Cell RRAM Macro in 40 nm CMOS with Compact Peripherals and 1.0 pJ/Bit Read Circuitry,” ESSCIRC 2023

Results for Bit Error Rate

- Gray coding to map bits to levels
- Bit error rate (BER) = $\frac{\#\ bit\ flips}{\#\ total\ bits}$

Chip	BPC	SBA BER	PBA BER	Rel. ΔBER
Sapiens	2	0.93%	0.27%	**-71%**
Sapiens	3	3.4%	2.4%	**-30%**
Ember1	2	0.05%	0%	**-100%**
Ember1	3	0.74%	0.38%	**-49%**
Ember2	2	0%	0%	**N/A**
Ember2	3	0.7%	0.37%	**-48%**

- Relative BER reduction by 30% - 71% (with 100% an outlier)

Results for ECC Overhead

- Error Correcting Code (ECC) overhead
  - Reliable storage medium: 10^{-14} unrecoverable bit error rate
  - Codeword size of at most 12 bits

Chip	BPC	SBA ECC	PBA ECC	Rel. ΔECC
Sapiens	2	13%	8%	**-41%**
Sapiens	3	30%	23%	**-22%**
Ember1	2	4.6%	0%	**-100%**
Ember1	3	12%	9.1%	**-26%**
Ember2	2	0%	0%	**N/A**
Ember2	3	12%	9%	**-23%**

- Relative ECC overhead reduction by 22% - 41% (with 100% an outlier)

Ablation Study

- Which factor contributes more to PBA's advantage over SBA?
  - Finding longest non-overlapping levels ($\Delta ECC1$ = SBA – PBA Norm)
  - Avoiding parameterized distribution ($\Delta ECC2$ = PBA – PBA Norm)

Chip	BPC	PBA Norm ECC	SBA ECC	Abs. ΔECC1	PBA ECC	Abs. ΔECC2
Sapiens	2	10%	13%	**+3%**	8%	**-2%**
Sapiens	3	31%	30%	**-1%**	23%	**-8%**
Ember1	2	5%	5%	**0%**	0%	**-5%**
Ember1	3	12%	12%	**0%**	9%	**-3%**
Ember2	2	0%	0%	**0%**	0%	**0%**
Ember2	3	13%	12%	**-1%**	9%	**-4%**

- $|\Delta ECC2| > |\Delta ECC1|$:
  Major improvement comes from **avoiding a parametrized distribution**

Statistical Study of RRAM Datasets

- D'Agostino's K^2 normality tests for write and relaxation dataset
  - Both resistance and its reciprocal (conductance)
  - Radar, Tech A, Tech B, Tech C are open-source RRAM datasets

Dataset	# Cells	Resistance: Write Normal %	Resistance: Relax Normal %	Conductance: Write Normal %	Conductance: Relax Normal %
Sapiens	200	2.1%	0%	2.1%	0%
Radar[1]	8192	0%	-	0%	-
Tech A[2]	16384	-	8.2%	-	8.7%
Tech B[2]	32768	-	4.5%	-	6.6%
Tech C[2]	16292	-	0.6%	-	3.6%

- Result: < 10% of the studied distributions are normal

[1]Le et al. "RADAR: A fast and energy-efficient programming technique for multiple bits-per-cell RRAM arrays." TED 2021. https://github.com/akashlevy/RRAM-RADAR-Tuning
[2]Levy et al. https://github.com/akashlevy/RRAM-Relaxation-Master-Model

Multi-Chip Datasets and Level Allocation

- Multi-chip datasets
  - 100/0 contains only the **target** chip's data
  - 0/100 contains only the **non-target** chip's data

Dataset	Ember1 SBA BER	Ember 1 PBA BER	Ember 1 Rel. Δ BER	Ember2 SBA BER	Ember 2 PBA BER	Ember 2 Rel. Δ BER
100/0	0.74%	0.38%	**-49%**	0.70%	0.37%	**-48%**
50/50	0.62%	0.46%	**-26%**	0.80%	0.49%	**-29%**
10/90	0.78%	0.65%	**-17%**	0.86%	0.64%	**-26%**
0/100	0.75%	0.64%	**-15%**	1.00%	0.72%	**-28%**

- Observation:
  - PBA **consistently** outperforms on multi-chip datasets
  - PBA can deliver **more** benefits if provided with **sufficient** target chip data

Reduced Datasets and Level Allocation

- Use only 25%, 50%, 75%, 90% of the original datasets

Dataset	Ember1 SBA BER	Ember 1 PBA BER	Ember 1 Rel. Δ BER	Ember2 SBA BER	Ember 2 PBA BER	Ember 2 Rel. Δ BER
25%	0.75%	0.74%	**-1%**	0.78%	0.57%	**-27%**
50%	0.76%	0.53%	**-30%**	0.70%	0.50%	**-29%**
75%	0.68%	0.43%	**-38%**	0.68%	0.46%	**-32%**
90%	0.68%	0.42%	**-38%**	0.67%	0.38%	**-43%**
100%	0.74%	0.38%	**-49%**	0.70%	0.37%	**-48%**

- Observation
  - PBA outperforms SBA across **different sizes** of datasets
  - Improvement over SBA is **larger** given **more data**

Conclusion

- Emerging non-volatile memories for MBPC storage
- We present PBA, a percentile-based level allocation algorithm
  - Outperforms state-of-the-art RRAM algorithm: SBA
  - Multi-chip and reduced dataset scenarios
  - Do not fit data to parameterized distributions
- Potentially generalize to a broader set of emerging memory technologies that support MBPC storage