On-chip Test Generation Using Linear Subspaces

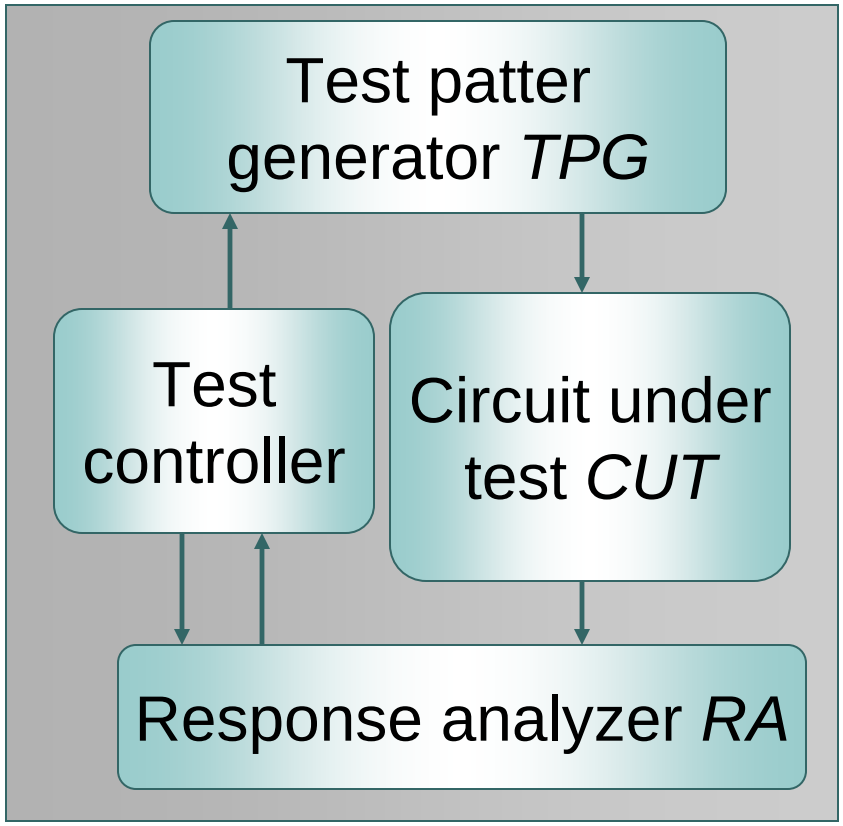
Ramashis Das, Igor Markov, John P. Hayes
University of Michigan
Ann Arbor, MI, USA

Outline

- Introduction
- Theoretical Framework
- Proposed Design
- Experimental Results
- Conclusions

Generic BIST Circuit

- Basic blocks:
  - Test Pattern Generator (TPG)
  - Test controller
  - Response Analyzer
- TPG: Feeds inputs to the circuit under test
- RA: Compares outputs with fault-free responses

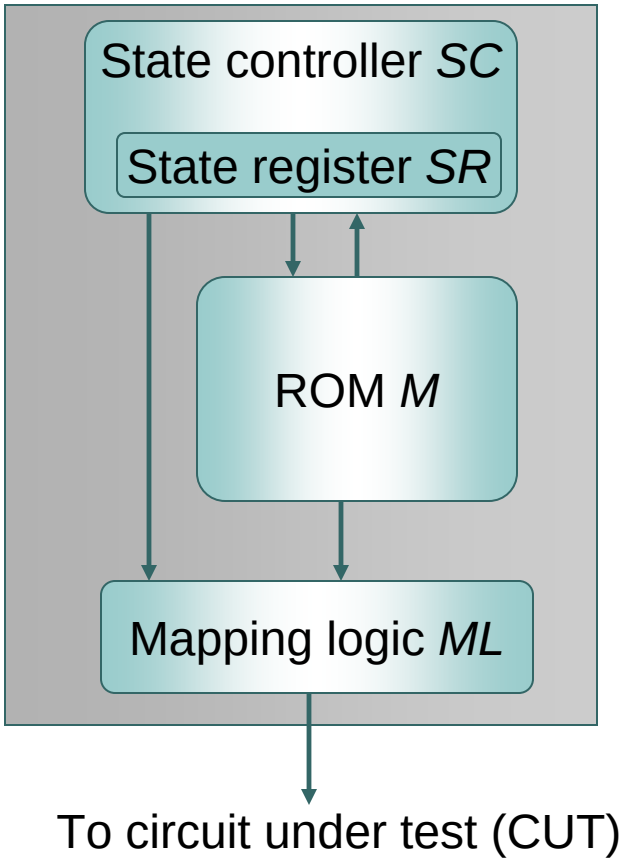


Test Pattern Generator (TPG)

- Performance of BIST determined by
  - Efficiency of TPG in producing good test vectors
- Desired features:
  - Low hardware overhead
  - High fault coverage
  - Short testing time

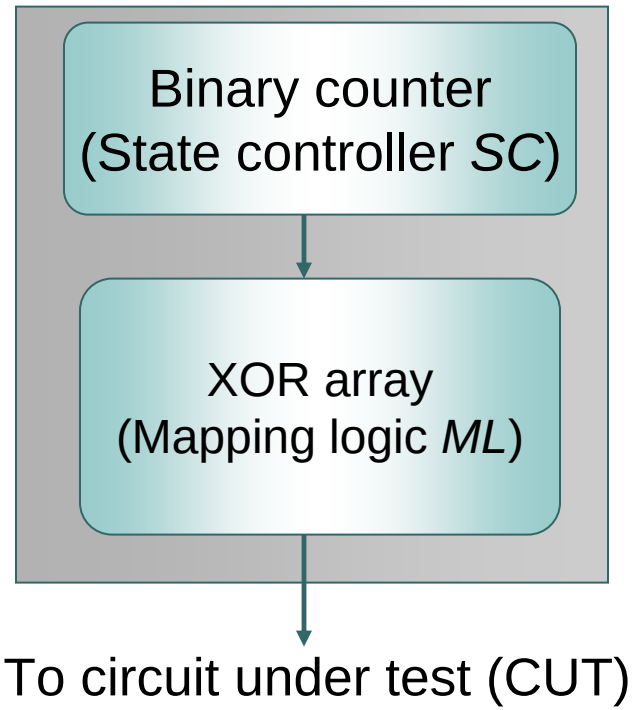
Generic TPG Structure

- Basic blocks:
  - State Controller
  - ROM
  - Mapping Logic
- *SC*: Holds current state of TPG in *SR*
- *ML*: Decodes state into test inputs for CUT
- *M*: Stores pre-determined test data



Some Existing TPG Designs

- Pre-stored tests
- Linear feedback shift registers (LFSRs)
- Linear transformation circuits [Akers, *ITC* 1989]
  - Obtain test set *T* by running an ATPG program
  - Embed *T* in *k* n-bit vectors
  - *SC*: *k*-bit binary counter
  - *ML*: XOR array



Vector Spaces

- Space: Largest set that satisfies closure property on a field F
- For test vectors, vector space V is defined over bit vectors and the field $F_2 = \{0, 1\}$
  - $\oplus$ – bit-wise XOR
  - $\bullet$ – bit-wise AND
- For $n = 3$, the vector space is $V_3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

Clusters (Subspaces)

- Cluster (subspace): subset of vector space
  - closed under bit-wise XORs
- Some clusters of V_3:
  - {000}
  - {000, 001}
  - {000, 001, 010, 011}
- Any cluster includes {000…0}

Bases

- Basis (of a cluster): set of vectors
  - Must produce the entire cluster by bitwise XOR operations (linear combinations)
  - Smallest such set (not unique)

Cluster V_3'	Basis B_3'
{000, 001}	{001}
{000}	{ }
{000, 001, 010, 011}	{001, 010} or {001, 011}, or {010, 011}

$011 = 001 \oplus 010$

Example: Test set Compression

ISCAS-85 c499 benchmark circuit

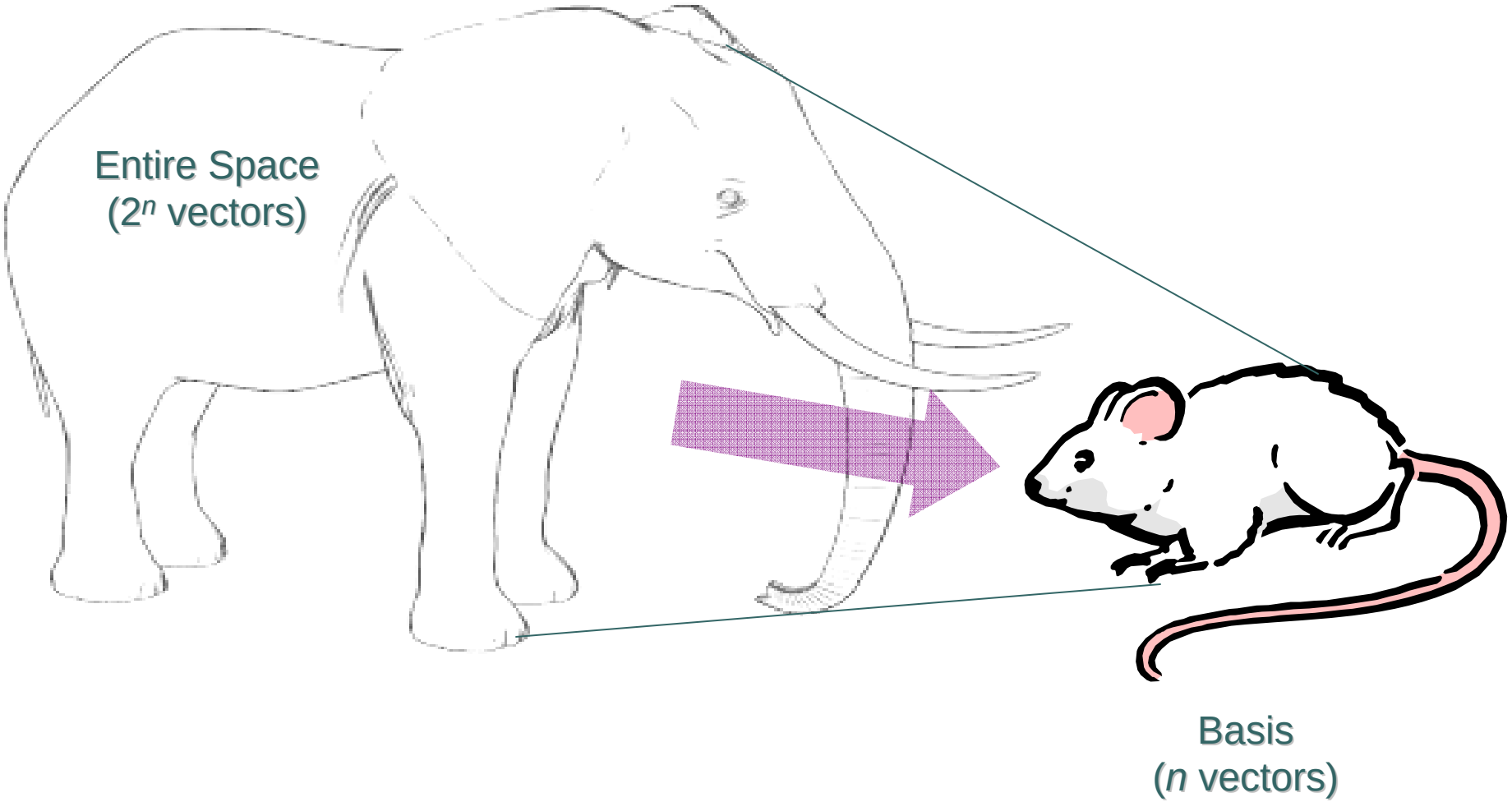
c499


```
1:  0110011110011000001100010101101001011000
2:  1011000100101011111111011011000111110111
3:  1101010111101000000101010111000111010101
4:  1010111000010000110101011000101110100110 1
5:  0010001001001011001111110101100101110101 1
6:  0010000001111010000000011111011100101111 1
7:  1111110110011011010000001010000001101000 0
8:  1111101110010111101001000011110000101000 1
9:  0000111011000011111110011101010011001100
10: 1101010110001011000101011111011000011100
11: 1001101011100001111111010110001111100111 0
12: 0110000010111000011101110110101110011000 1
...
...
51: 0000100001000010001010101111011111010100
52: 1001010010001101011101000111011001011100
53: 1111111100000000011010010010000110110110 1
```

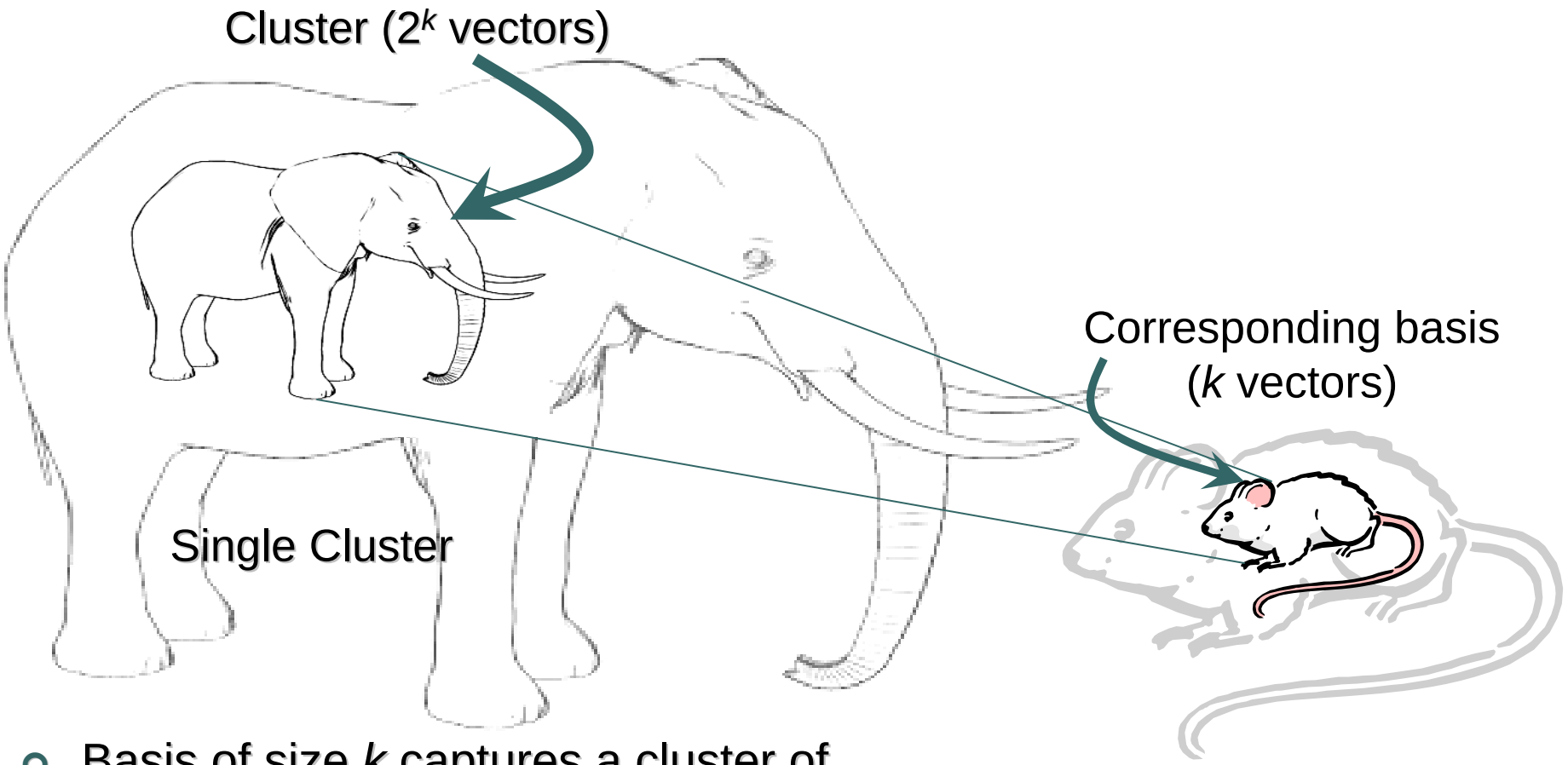
Compression ratio of 6.6!

Time to test: $2^8 = 256$

Size Reduction



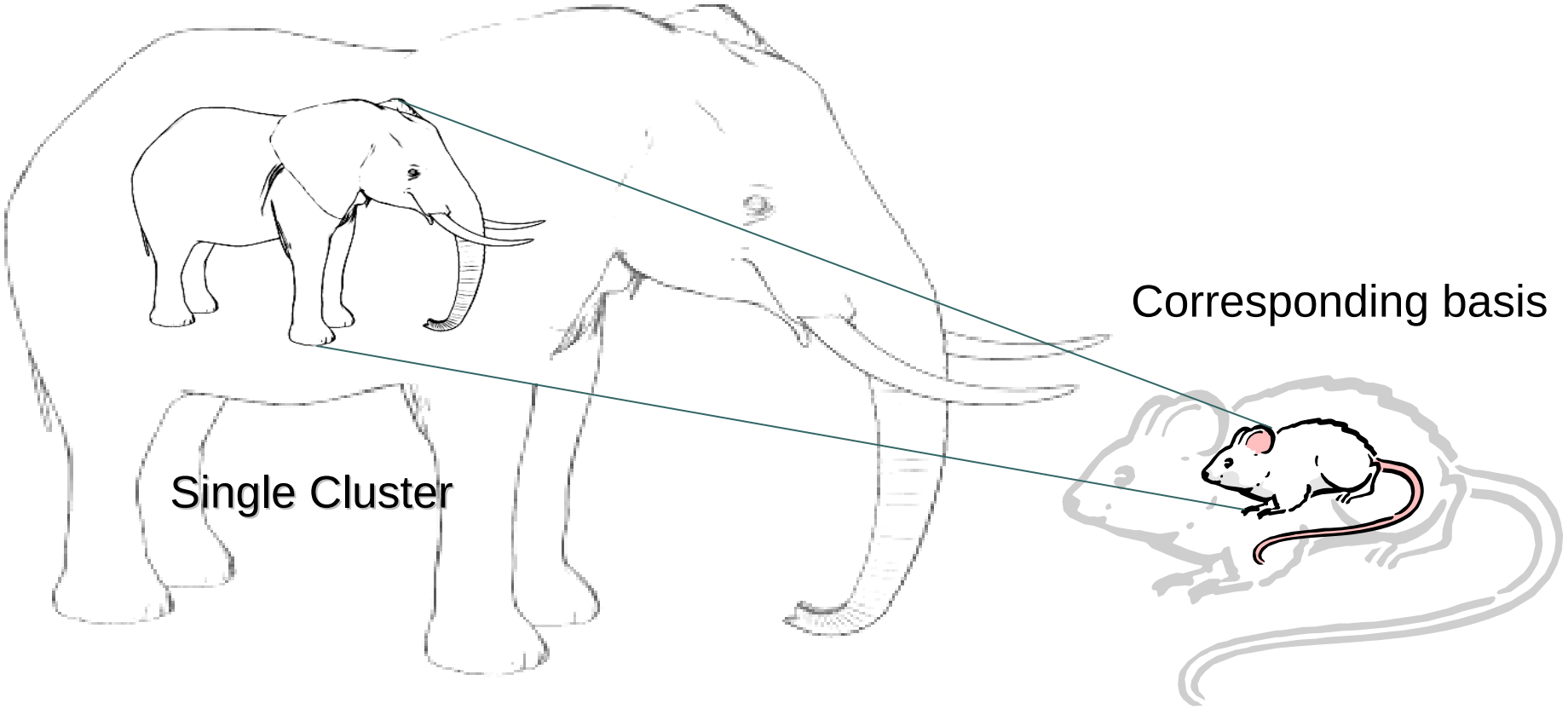
Size Reduction



- Basis of size k captures a cluster of size 2^k
- Compression ratio: $2^k/k$

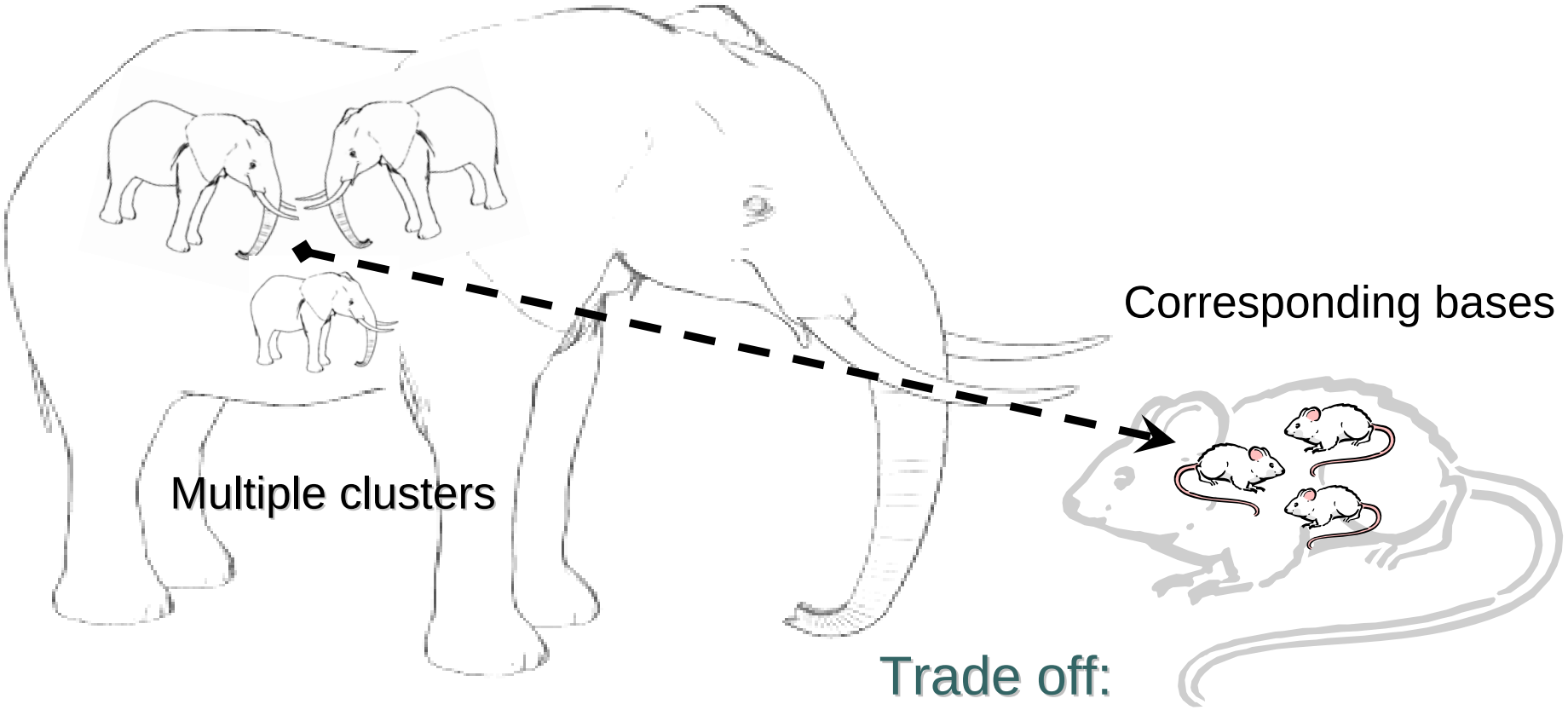
Storage overhead: k
Testing time: 2^k

Single vs. Multiple Clusters



Storage overhead: k
Testing time: 2^k

Single vs. Multiple Clusters

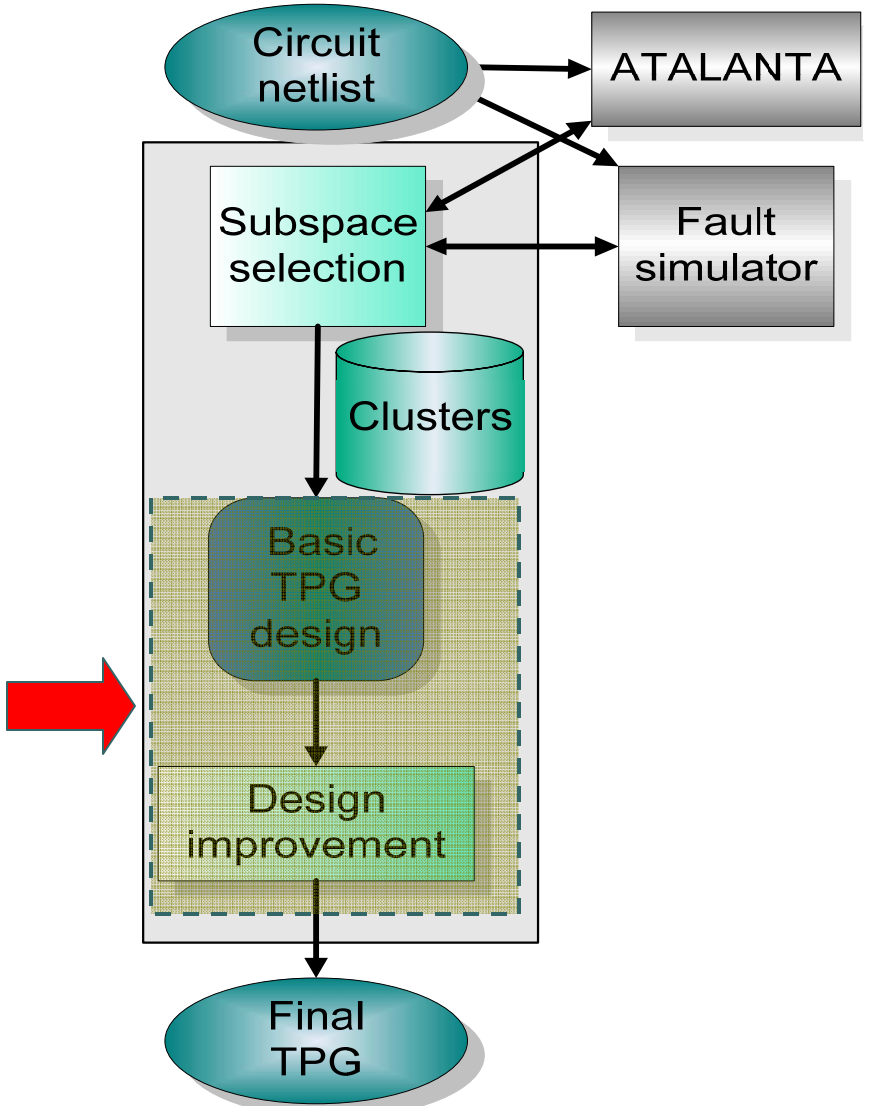


Trade off:

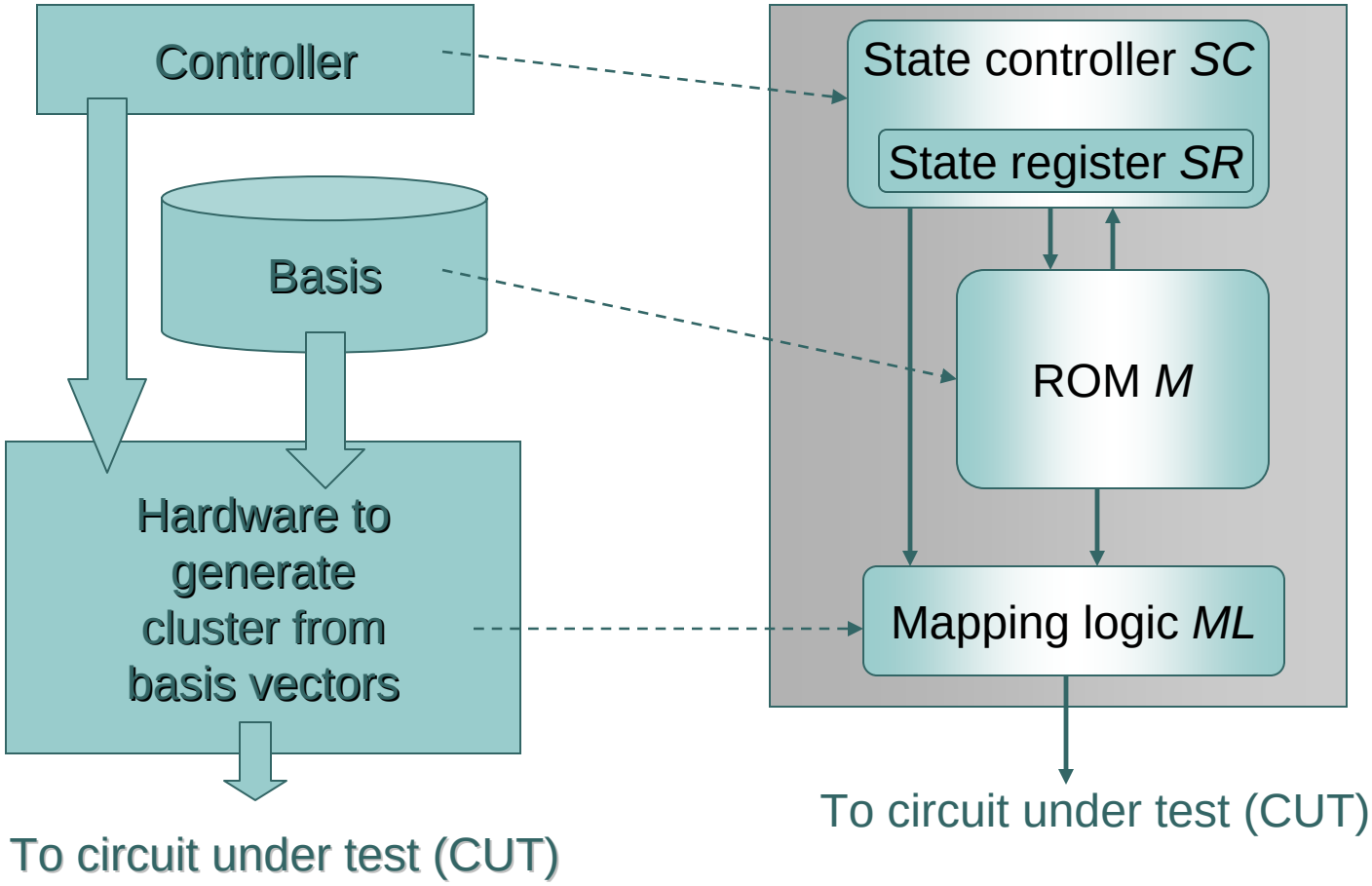
Storage overhead: $k_1 + k_2 + k_3$ ($> k$, $\sim k$)

Testing time: $2^{k_1} + 2^{k_2} + 2^{k_3}$ ($<< 2^k$)

Our Design Flow



Basic Idea of our TPG Design



Generating a Cluster

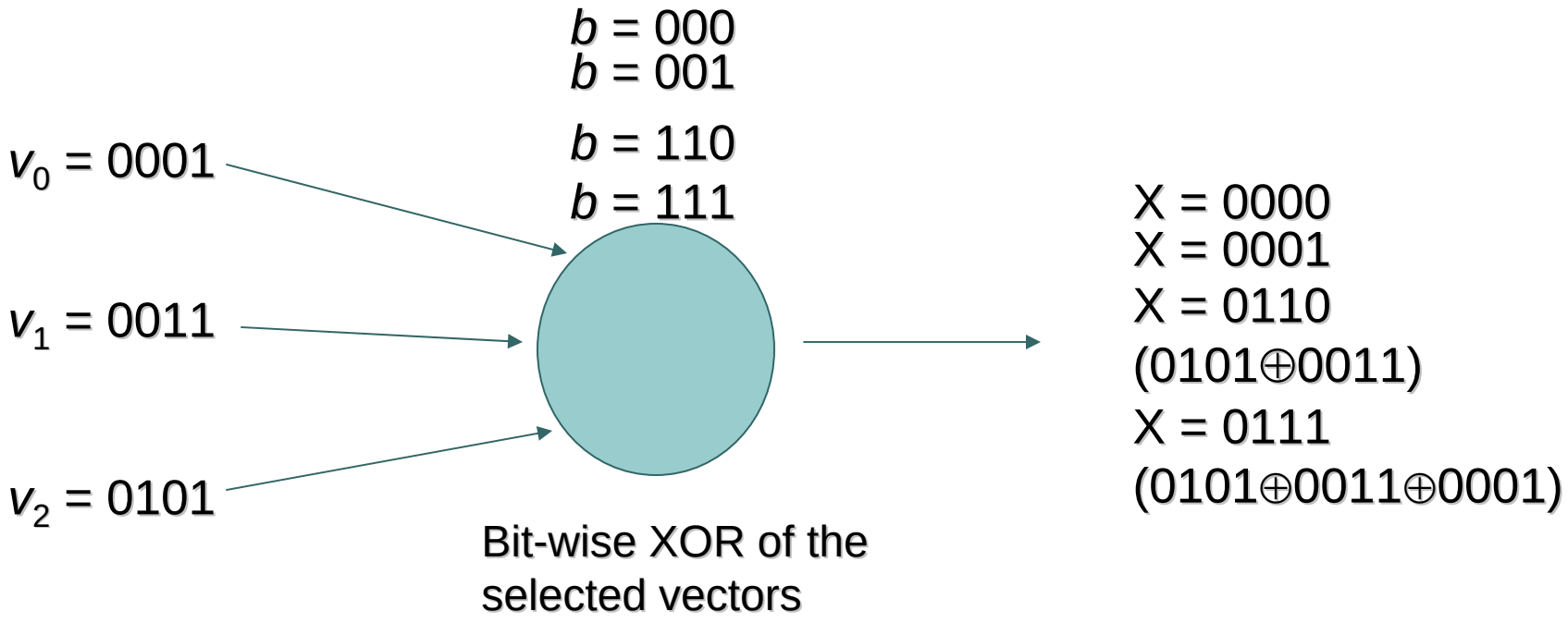
- Enumerate all possible sums of basis vectors
  - Use a *k*-bit binary counter *b=0..2^k-1*
  - *j*th bit of count *b* includes/excludes v_j (basis vector)

$$X_b = \sum_{j=0}^{k} b_j v_j, 0 \le b \le 2^{k-1}$$

- Thus, *b* determines a sum of basis vectors X_b
- All possible X_b = all vectors in the subspace
  - Enumerated with a binary counter (or another counter!)

Example: Cluster Generation

Basis: $\{v_0 = 0001, v_1 = 0011, v_2 = 0101\}$
Size = 3 $\rightarrow$ need a 3-bit counter (state $b = b_2b_1b_0$)



Pros and Cons of Binary Enumeration

- Pros
  - Basis selection is flexible
  - Allows one to ensure full fault coverage
- Cons
  - Explicit XOR of all vectors $\rightarrow$ 2-D array of XORs
  - Large delays and area overhead due to XORs
- Similar to test embedding [Akers, *ITC'89*]

Use Gray Codes Instead!

- In Gray codes (000,001,011,010,110…)
  - Two consecutive counts differ by a single bit
- We replace a binary counter with a Gray-code counter in

$$X_g = \sum_{j=0}^{k} g_j v_j, 0 \le g \le 2^{k-1}$$

- X_{g+1} differs from X_g by basis vector v_j, such that $j=h$ is the flipped bit

$$X_{g+1} = X_g \oplus v_h$$

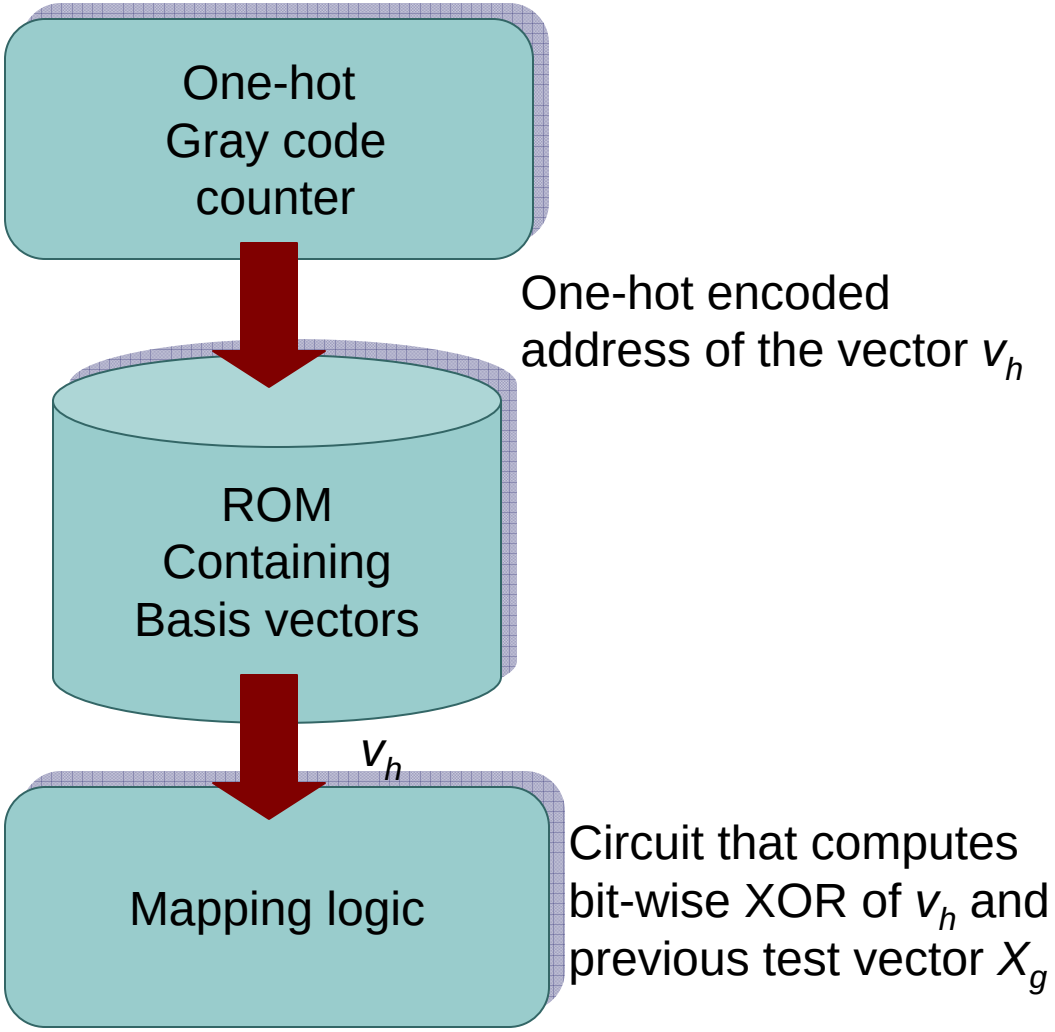
Fewer XOR gates required !

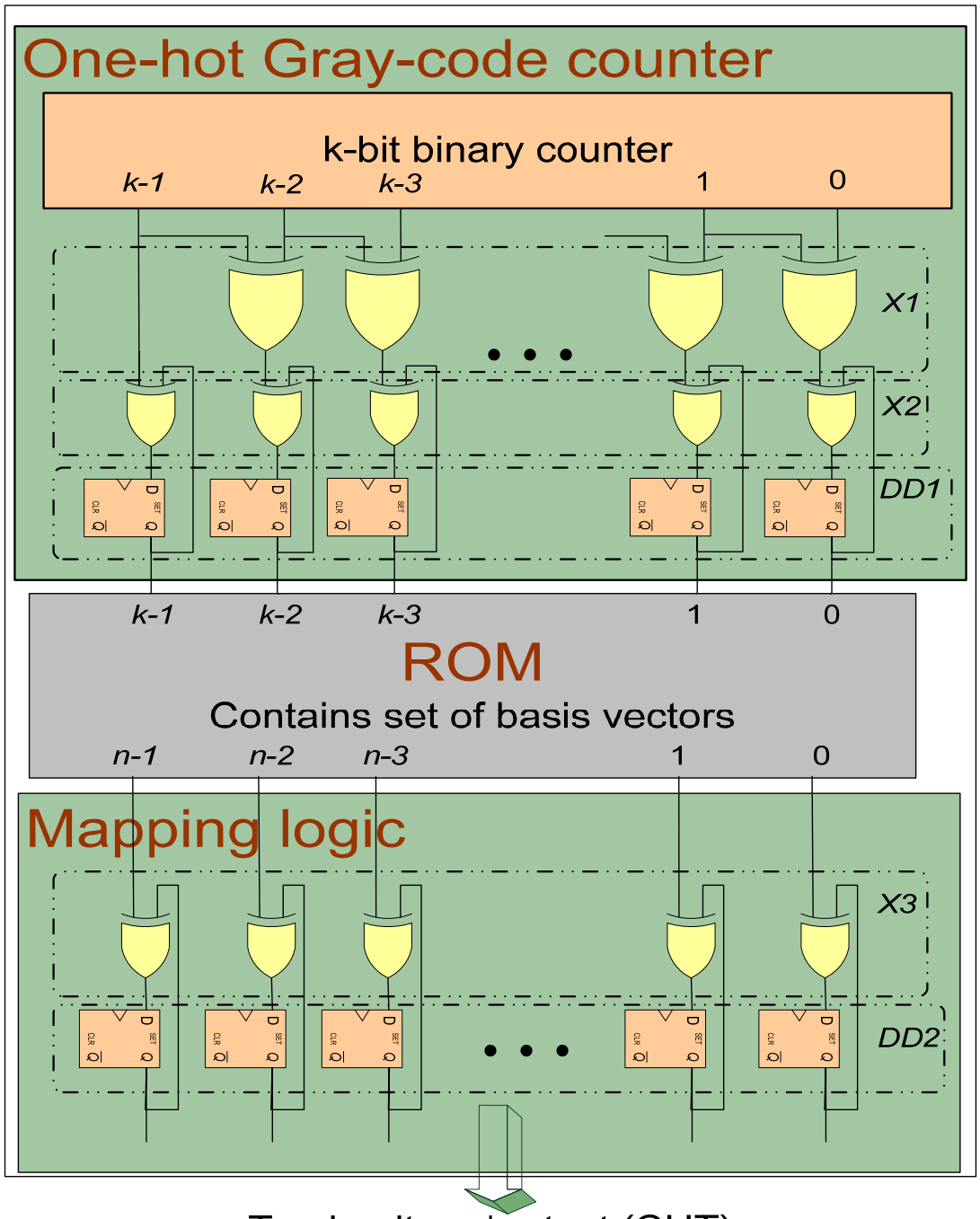
Example: Using Gray Codes

Basis: $\{v_0 = 0001, v_1 = 0011, v_2 = 0101\}$
Size = 3 $\rightarrow$ need a 3-bit counter (state $g = g_2 g_1 g_0$)

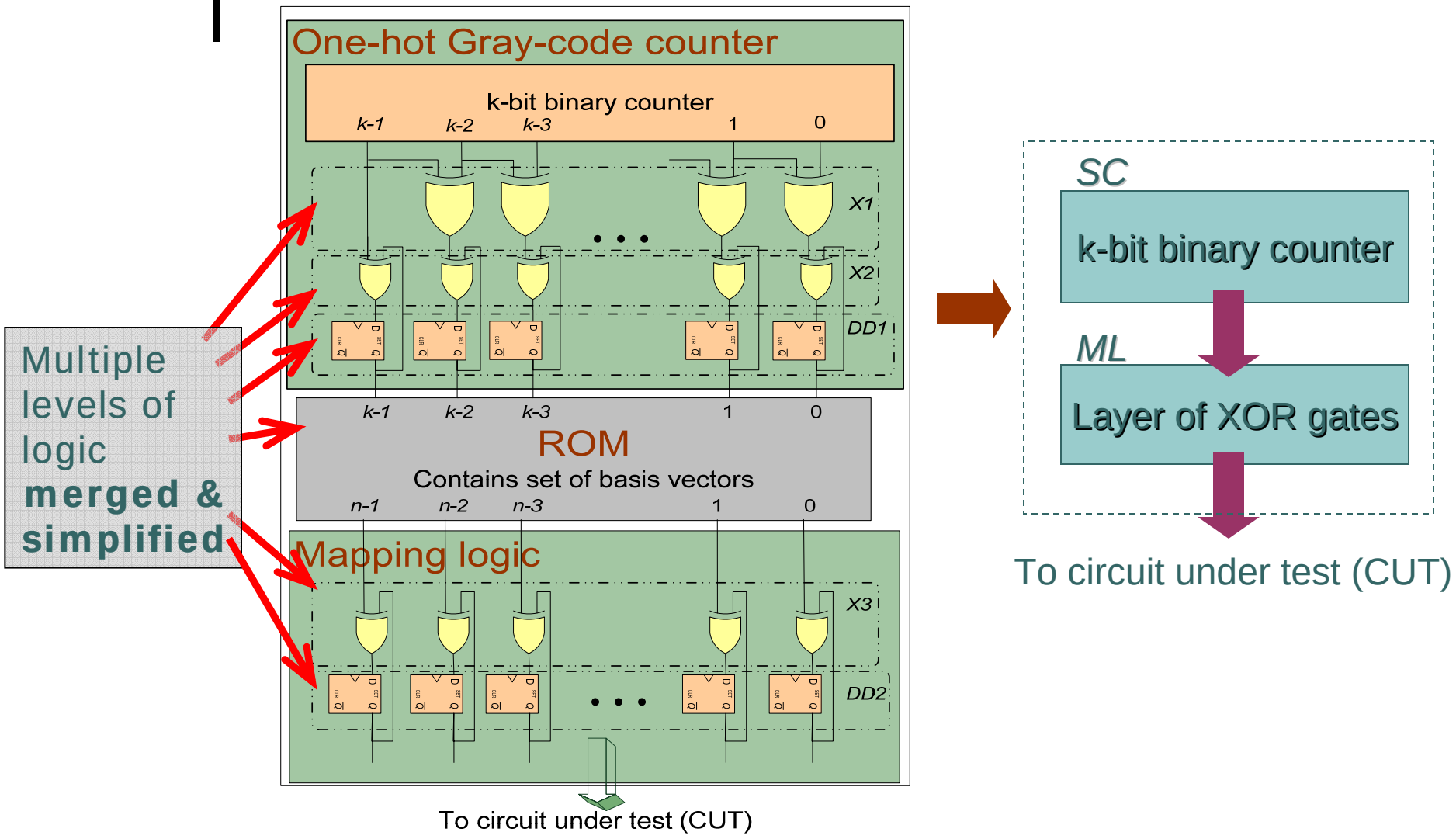
Gray code (g)	Flipping bit index (h)	One-hot Gray code	X_g	$\oplus v_h$	$= X_{g+1}$
000	-	-	0000	-	0000
001	0	001	0000	0001	0001
011	1	010	0001	0011	0010
010	0	001	0010	0001	0011
110	2	100	0011	0101	0110
111	0	001	0110	0001	0111
101	1	010	0111	0011	0100
100	0	001	0100	0001	0101

Cluster Generation in H/W

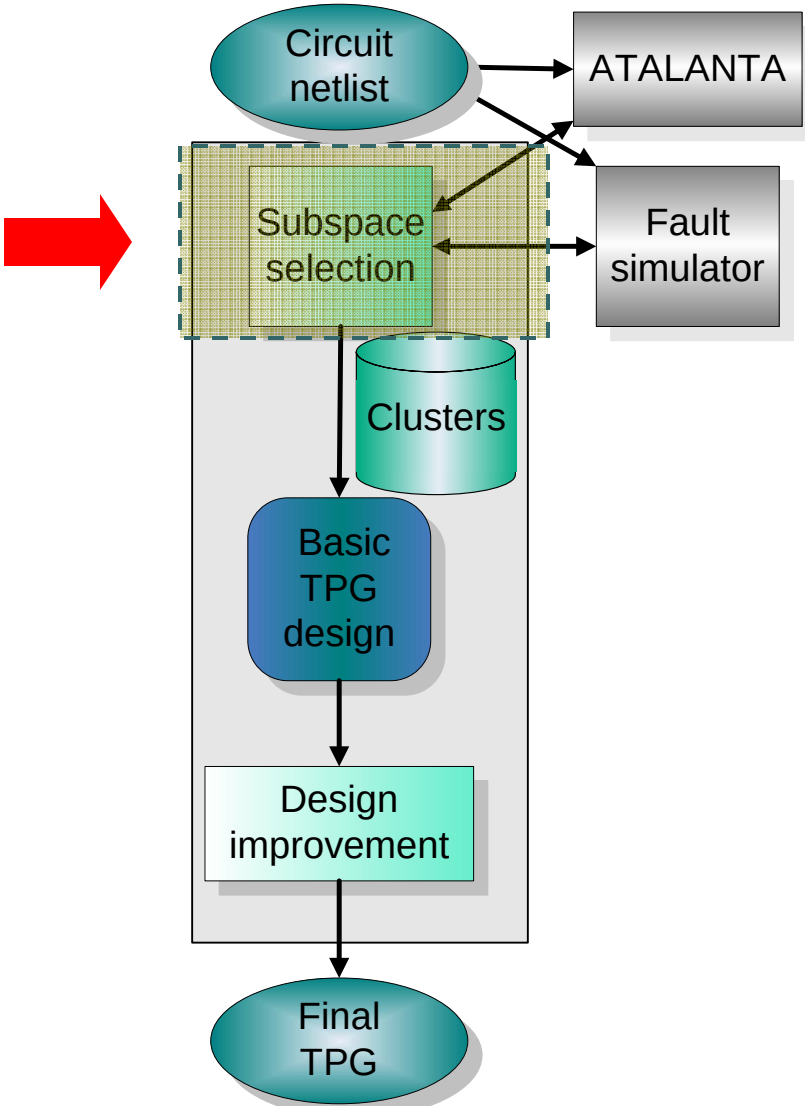



One-hot Gray-code counter
k-bit binary counter
k-1
k-2
k-3
1
0
X1
X2
DD1
k-1
k-2
k-3
1
0
ROM
Contains set of basis vectors
n-1
n-2
n-3
1
0
Mapping logic
X3
DD2
To circuit under test (CUT)

Design Improvement



Our Design Flow - Revisited



Subspace/Cluster Selection

- Find clusters & bases for a circuit
  - To achieve full fault coverage (by subspaces)
- Optimize performance metrics:
  - Area overhead: # basis vectors
  - Testing time: size of largest cluster

1. Start with {000…0} vector in cluster *S*
2. Find faults not detected by *S* (fault simulation)
3. Run ATPG to obtain test vectors *T*
4. Find $t \in T$ that best increases fault coverage of *S*
5. Add *t* it to *S*
6. Unless full fault coverage reached, go to step 2

Results: Testset Compression

Bench-mark circuits	No of inputs	ATALANTA	Proposed Method				
		No. of test patterns (n_1)	Max cluster size	No. of basis vectors (n_2)	Total no. of clusters	Compres-sion ratio (n_1/n_2)	Test size
c432	36	51	8	8	1	6.37	256
c499	41	53	8	8	1	6.63	256
c880	60	58	9	13	2	4.46	527
c1355	41	86	10	10	1	8.60	1024
c1908	33	115	11	14	2	8.21	2055
c2670	233	101	11	40	4	2.53	6269
c3540	50	144	12	14	2	**10.29**	4099
c5315	178	116	11	11	1	**10.55**	2048
c6288	32	31	7	7	1	4.43	128
c7552	207	212	13	41	4	5.17	24577

Results: Testset Size

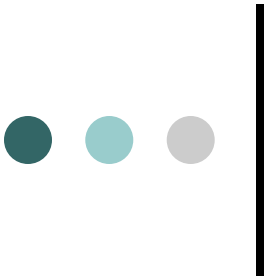
Benchmark circuits	Test set size					
	Weighted random pattern	Akers and Jansz	Multiple seeds/ polynomials	Use of counters	GLFSR	Our TPG
c432	636	1024	320	125	n/a	256
c499	1125	1024	679	22064	n/a	256
c880	765	8192	1596	29	640	527
c1355	3059	4096	1447	122344062	1760	1024
c1908	3539	8192	3659	1169	4700	2055
c2670	7689	65536	33000	n/a	6128	6269
c3540	3351	8192	6592	970	4828	4099
c5315	2279	8192	1843	62	n/a	2048
c6288	39	512	43	98003134	n/a	128
c7552	9276	n/a	32800	n/a	n/a	24577

Results: Area Overhead

Benchmark circuits	Akers and Jansz	GLFSR	CAPS	Our TPG
c432	630	827	922	468
c499	784	926	1044	555
c880	1208	1365	1531	1099
c1355	738	926	1044	683
c1908	725	761	847	715
c2670	3668	5309	5951	9631
c3540	1027	1086	1255	1127
c5315	2708	3984	4517	3216
c6288	429	745	824	387
c7552	n/a	4617	5245	9834

Conclusions

- New on-chip test generation technique
  - Uses linear subspaces and bases
  - Uses Gray-code enumeration
  - Multi-level logic optimization (see paper)
  - End result: compact hardware design
- Heuristic for selecting subspaces & bases
- Salient features:
  - Achieves complete fault coverage
  - Relatively low hardware cost
  - Relatively small test set (testing time)



Thank You

Future Work

- Handling incompletely specified test sets (don’t cares)
- Overlap of Clusters
- Trade off between fault coverage and area overhead and/or testing time
- Improvement in heuristic used to select clusters
- Extension to scan-based testing