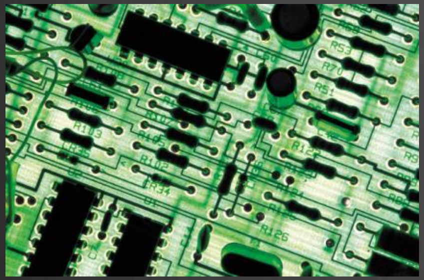
On Parallel Scalable Uniform Generation of SAT Witnesses

Supratik Chakraborty[1], Daniel J. Fremont[2], **Kuldeep S. Meel[3]**, Sanjit A. Seshia[2], Moshe Y. Vardi[3]

[1]Indian Institute of Technology Bombay, India
[2]University of California, Berkeley
[3]Rice University

(Author names are ordered alphabetically by last name)

How do we guarantee that systems work *correctly* ?



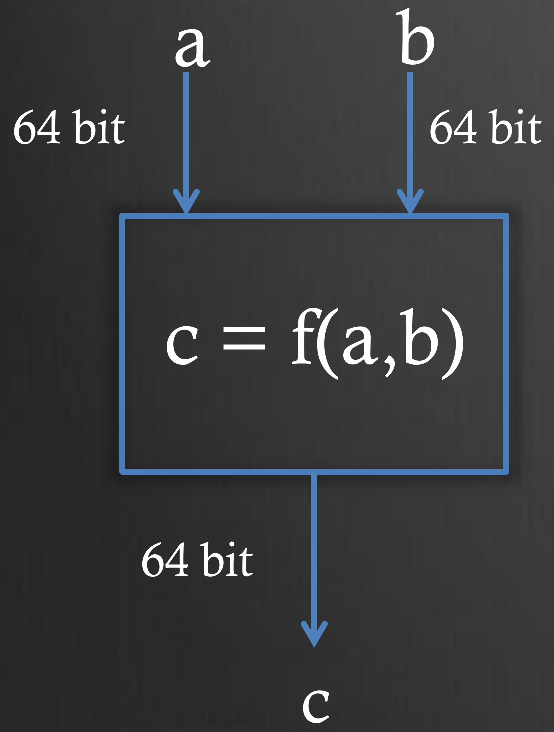
Functional Verification

- Formal verification
  - Challenges: formal requirements, scalability
  - ~10-15% of verification effort
- Dynamic verification: ***dominant approach***

Dynamic Verification

- Design is simulated with test vectors
- Test vectors represent different verification scenarios
- Results from simulation compared to intended results
- **Challenge**: Exceedingly large test space!

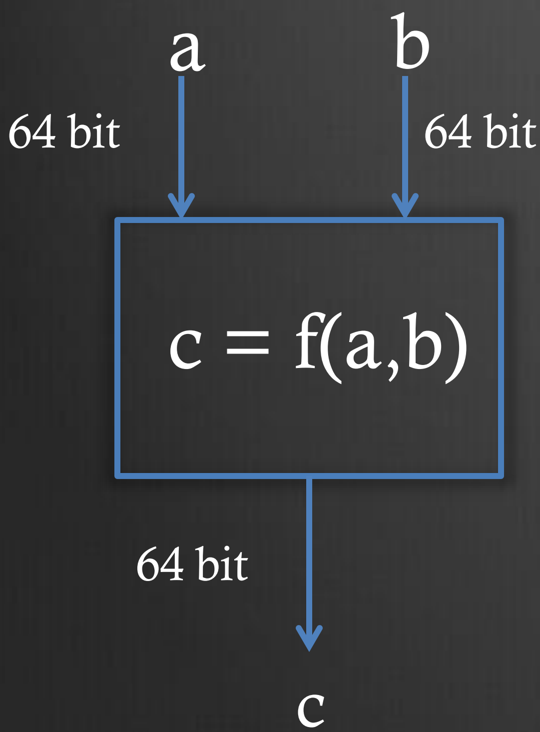
Motivating Example



How do we test the circuit works ?

- Try for all values of a and b
  - 2^{128} possibilities
  - Sun will go nova before done!
  - Not scalable

Constrained-Random Simulation

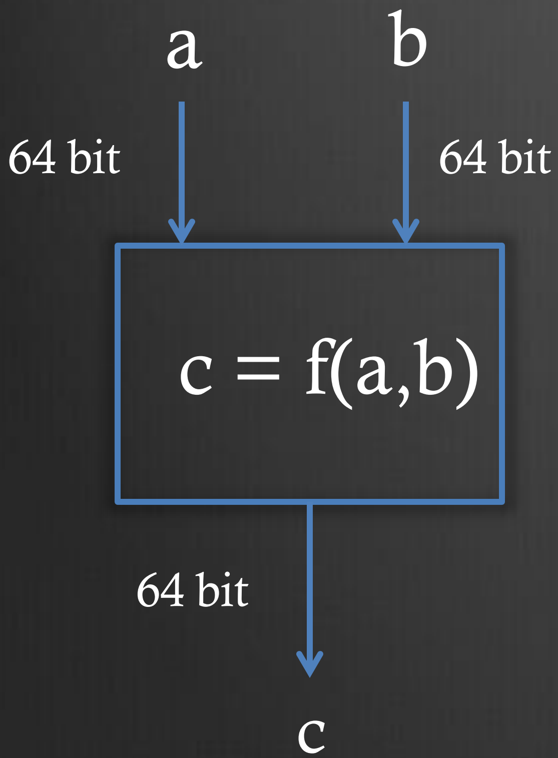


Sources for Constraints

- Designers:
  1. $a +_{64} 11 *_{32} b = 12$
  2. $a <_{64} (b >> 4)$
- Past Experience:
  1. $40 <_{64} 34 + a <_{64} 5050$
  2. $120 <_{64} b <_{64} 230$
- Users:
  1. $232 *_{32} a + b \mathrel{!=} 1100$
  2. $1020 <_{64} (b /_{64} 2) +_{64} a <_{64} 2200$

- Test vectors: solutions of constraints
  - Proposed by Lichtenstein, Malka, Aharon (IAAI 94)

Constrained-Random Simulation

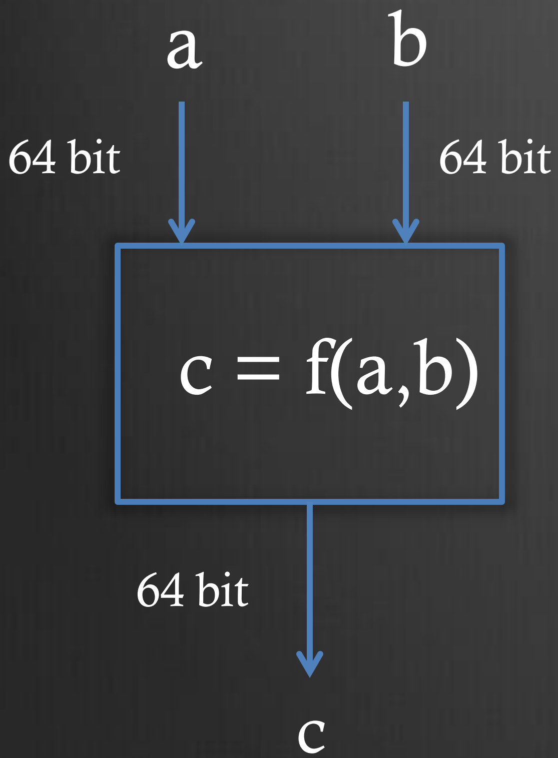


Sources for Constraints

- Designers:
  1. $a +_{64} 11 *_{32} b = 12$
  2. $a <_{64} (b >> 4)$
- Past Experience:
  1. $40 <_{64} 34 + a <_{64} 5050$
  2. $120 <_{64} b <_{64} 230$
- Users:
  1. $232 *_{32} a + b \mathrel{!}= 1100$
  2. $1020 <_{64} (b /_{64} 2) +_{64} a <_{64} 2200$

Problem: How can we *uniformly* sample the values of a and b satisfying the above constraints?

Problem Formulation



Set of Constraints

↓

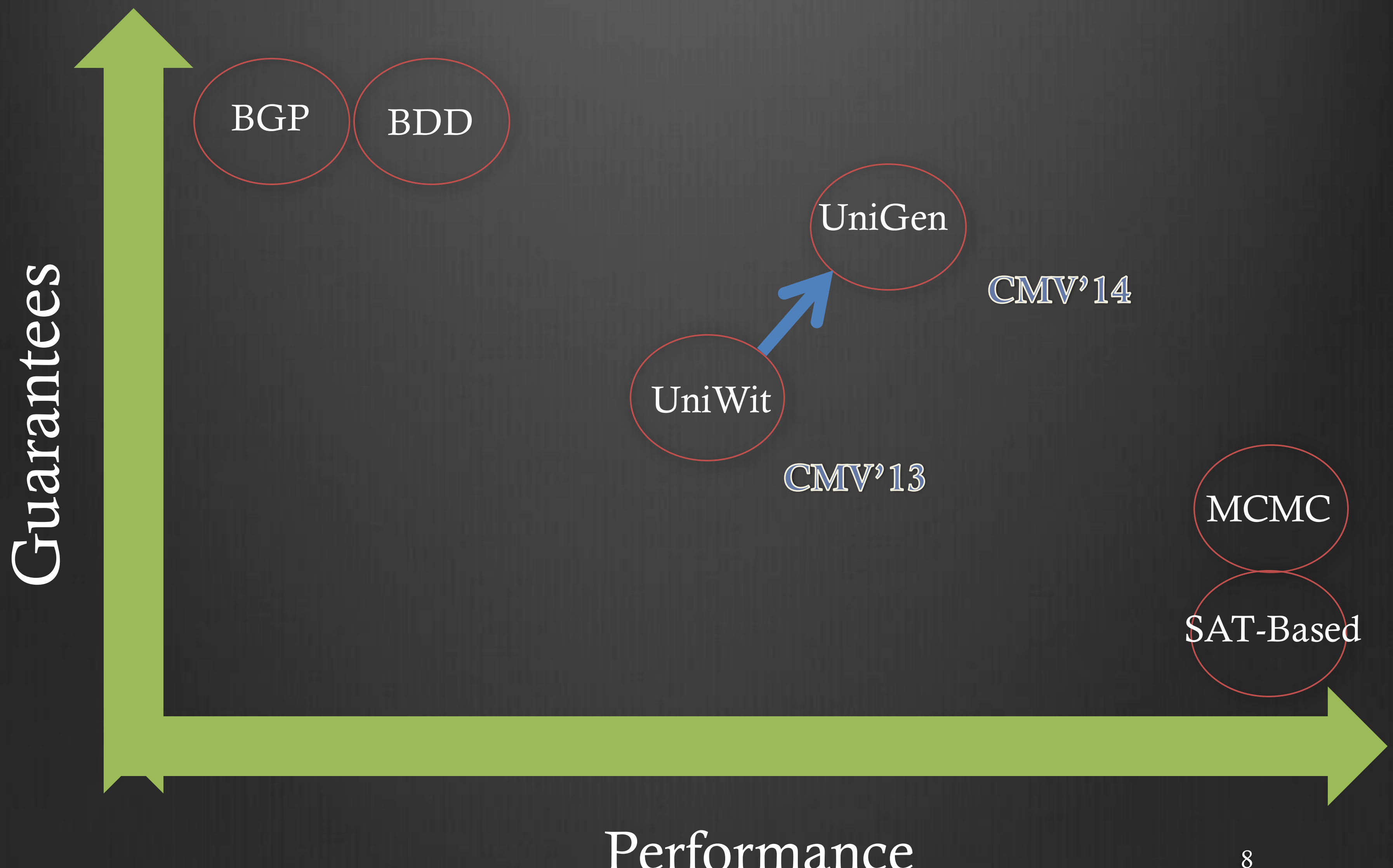
SAT Formula

↓

Sample satisfying assignments uniformly at random

Scalable Uniform Generation of SAT Witnesses

Prior Work


Guarantees
BGP
BDD
UniGen
CMV'14
UniWit
CMV'13
MCMC
SAT-Based
Performance

Guarantees of UniGen

Near-Uniformity

- F : Input formula
- R_F : Solution space

$$\forall y \in R_F, \frac{1}{(1+\varepsilon)|R_F|} \leq \Pr[\text{y is output}] \leq (1+\varepsilon)\frac{1}{|R_F|}$$

Drawbacks of UniGen

Makes a large number (polynomial in $1/\varepsilon$) of calls to a SAT solver to generate a single sample

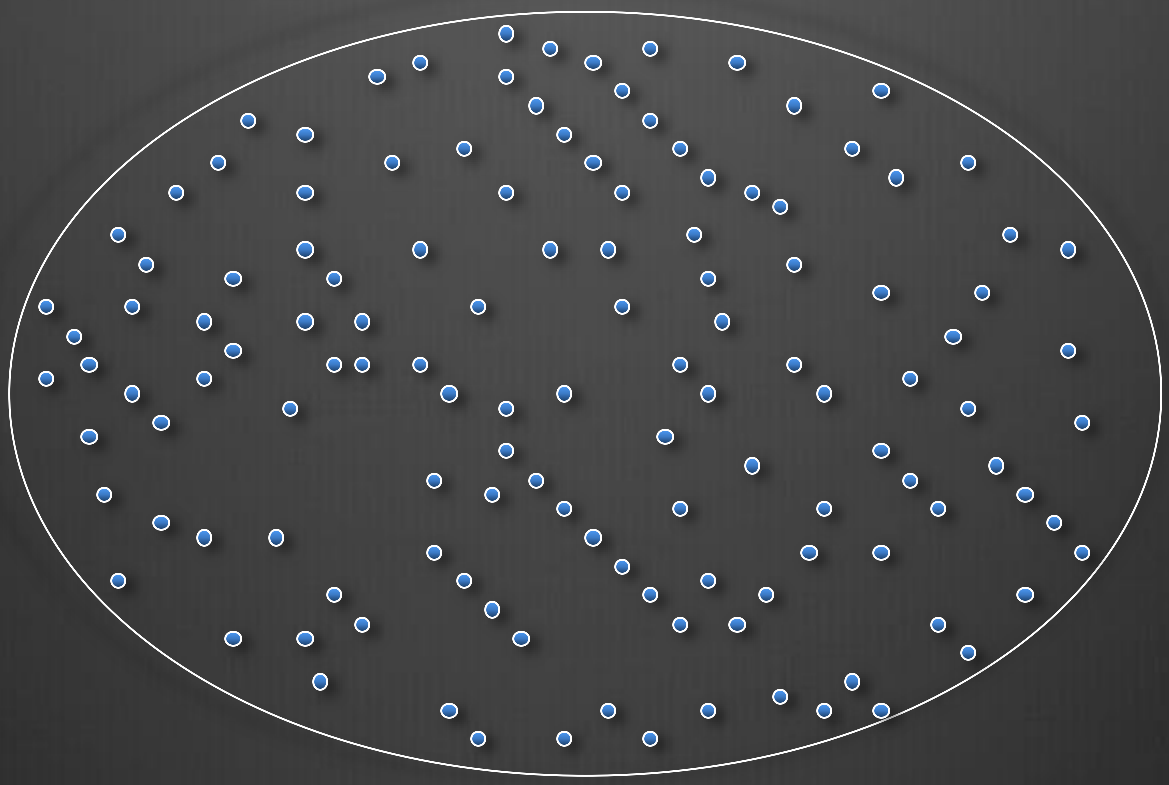
Generator	Relative Runtime
UniGen	470
Desired Uniform Generator*	10
Simple SAT solver	1

*: Based on EDA Industry

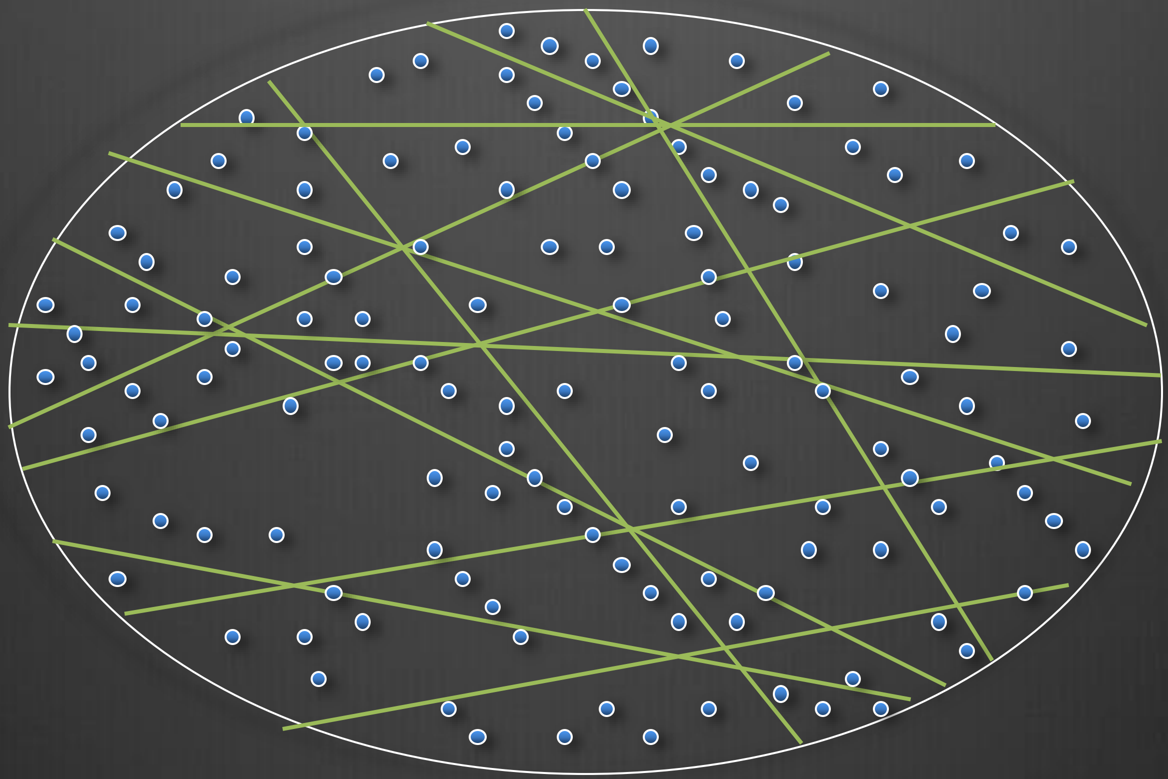
Outline

- UniGen2: Next Generation UniGen
  - Core Technical Ideas
  - Experimental Results
- Parallelization of Constrained Random Simulation
- Conclusion

Main Idea



Partitioning into cells

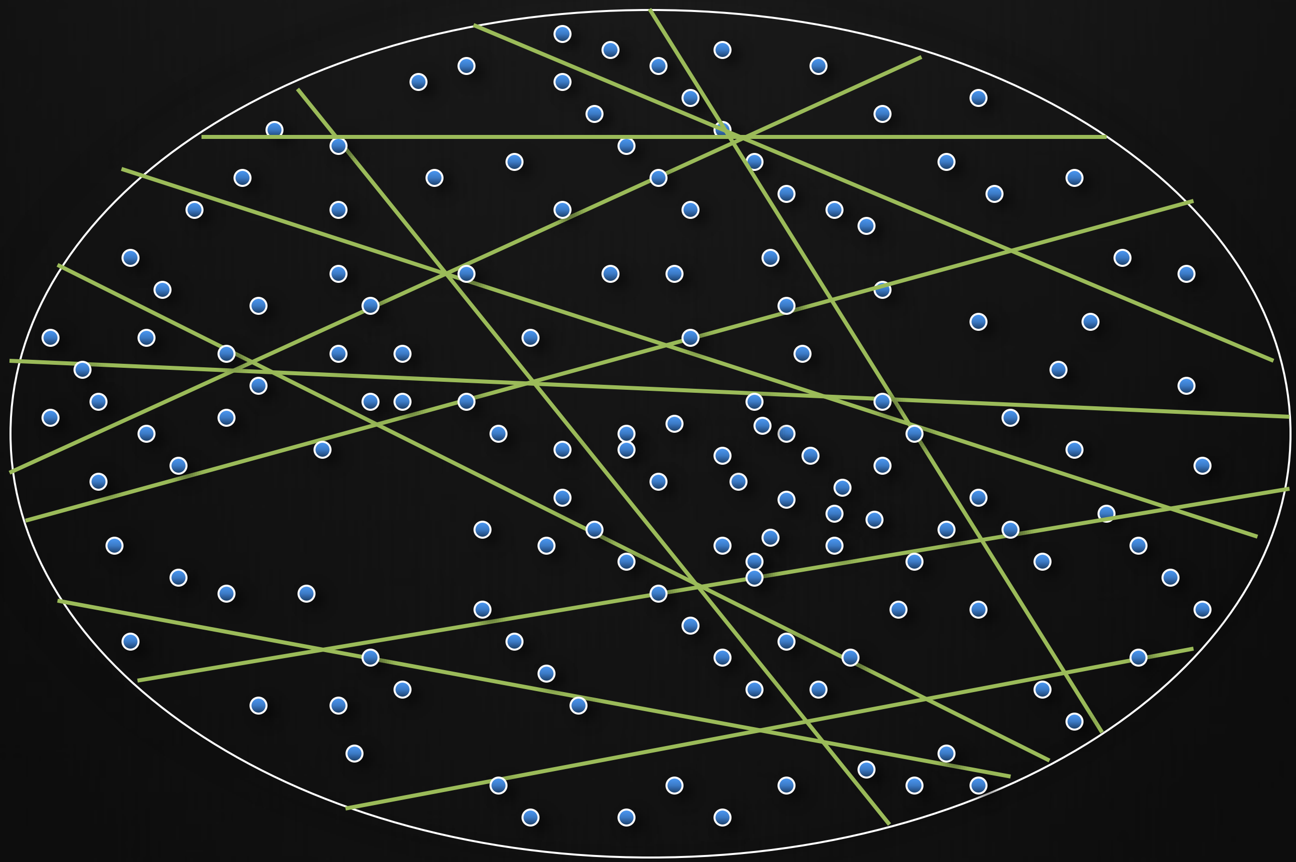


Cells should be roughly equal in size and small enough to enumerate completely

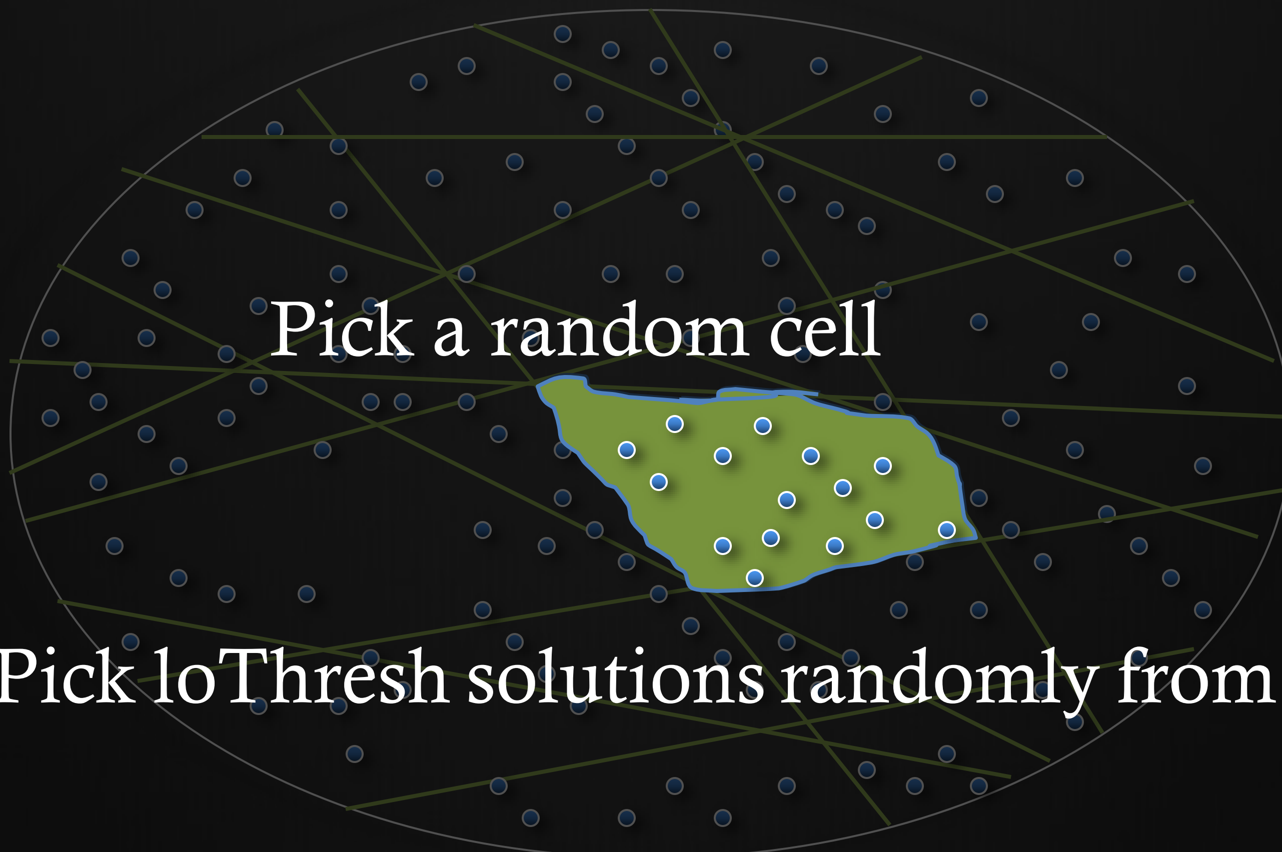
Partitioning into cells

- Too large => Hard to enumerate
- Too small => Variance can be very high
- hiThresh: upper bound on size of cell
- loThresh: lower bound on size of cell
  - E.g., loThresh = 11, hiThresh = 60

Partitioning into cells



Partitioning into cells



Pick a random cell

Pick loThresh solutions randomly from this cell

Partitioning into cells

How can we partition into roughly equal small cells without knowing the distribution of solutions?

3-Universal Hashing using random XORs [Carter-Wegman 1979, Sipser 1983]

A Few more Questions?

- How many cells do we need?
- How to enumerate solutions in a cell?

How many cells do we need?

- Need to pick cell count so that cells are in the desired size range with high probability
- Linear search technique based on model counting
  - Preprocessing step
  - Needs to be done **only once** per input formula

Enumerating cell solutions

- A cell can be represented as the conjunction of:
  - Input formula F
  - m random XOR constraints
- 2^m is the number of cells desired
- Use CryptoMiniSAT for CNF + XOR formulas

Strong Guarantees

- $L = \#\ of\ samples < |R_F|$

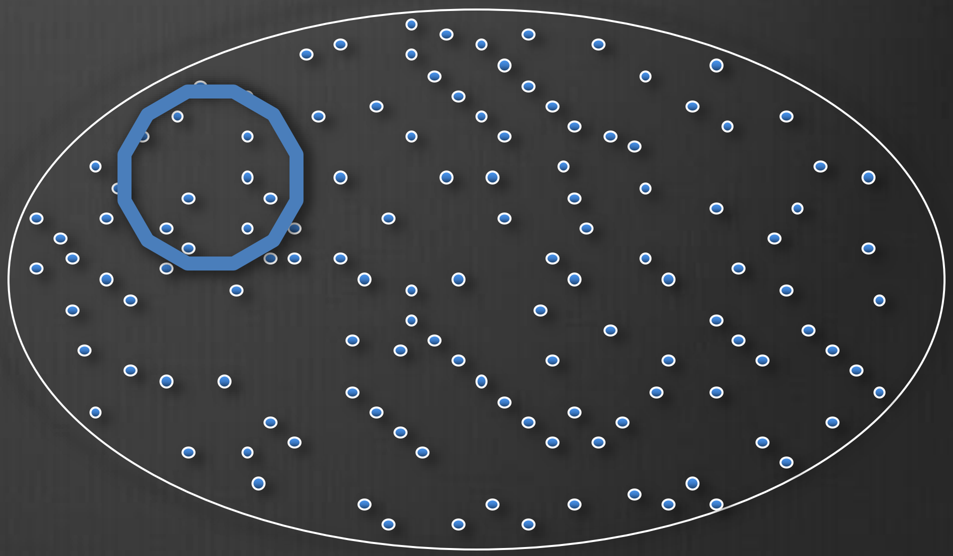
  $\forall y \in R_F,$

  $$\frac{L}{(1+\varepsilon)|R_F|} \le \Pr[\text{y is output}] \le 1.02(1+\varepsilon)\frac{L}{|R_F|}$$

- ~~Polynomial~~ Constant number of SAT calls per sample

A Question from CRV expert

- Are all the samples independent?
  - Independence allows coverage guarantees.



Well, **No but Yes**

3-Universal and Independence of Samples

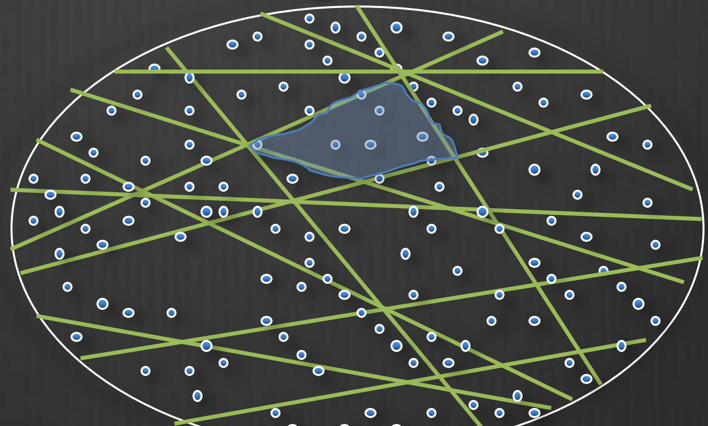
3-Universal hash functions:

- Choose hash function randomly
- For arbitrary distribution on solutions=> All cells are *roughly* equal in <u>expectation</u>

- <u>But:</u>
  - While each input is hashed **uniformly**
  - And each 3-solutions set is hashed **independently**
  - A 4-solutions set ***might not*** be hashed **independently**

3-Universal and Independence of samples

- Choosing 3 samples => Full Independence between samples
- Choosing loThresh (> 3) samples => Loss of full independence among samples
  - "Almost-Independence"
  - Still provides theoretical guarantees of coverage



Bug-finding effectiveness

bug frequency $f = B/R_F$

	UniGen	UniGen2
relative number of SAT calls	$\frac{3 \cdot hiThresh(1+\nu)(1+\varepsilon)}{0.52}$	$\frac{3 \cdot hiThresh}{0.62 \cdot loThresh} \frac{(1+\widehat{\nu})(1+\varepsilon)}{1-\widehat{\nu}}$

Simply put,

#of SAT calls for UniGen2 << # of SAT calls for UniGen

Bug-finding effectiveness

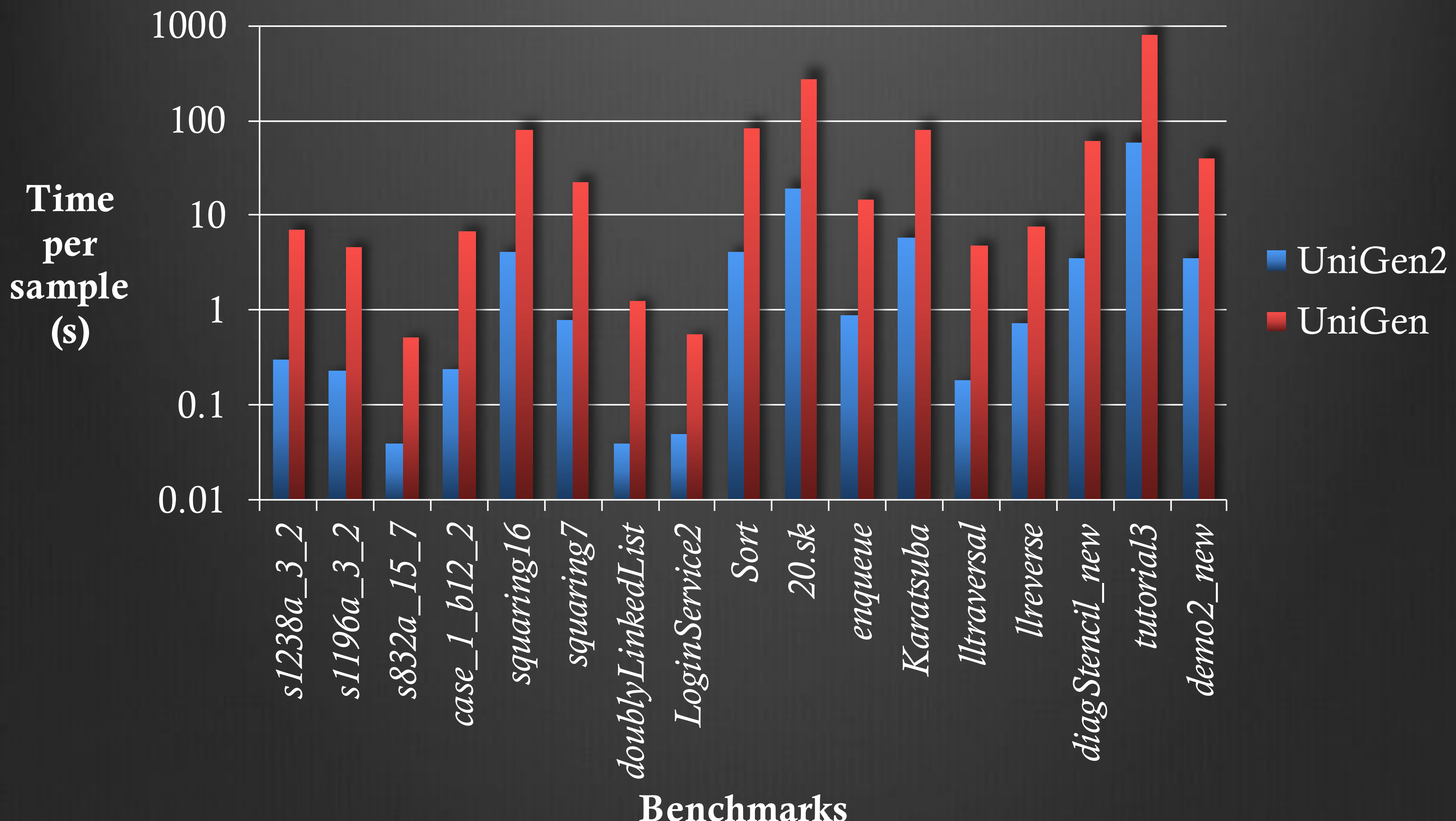
bug frequency $f = 1/10^4$

find bug with probability $\geq 1/2$

	UniGen	**UniGen2**
Expected number of SAT calls	4.35×10^7	3.38×10^6

An order of magnitude difference!

~20 times faster than UniGen



Runtime Performance

Experiments over 200+ benchmarks

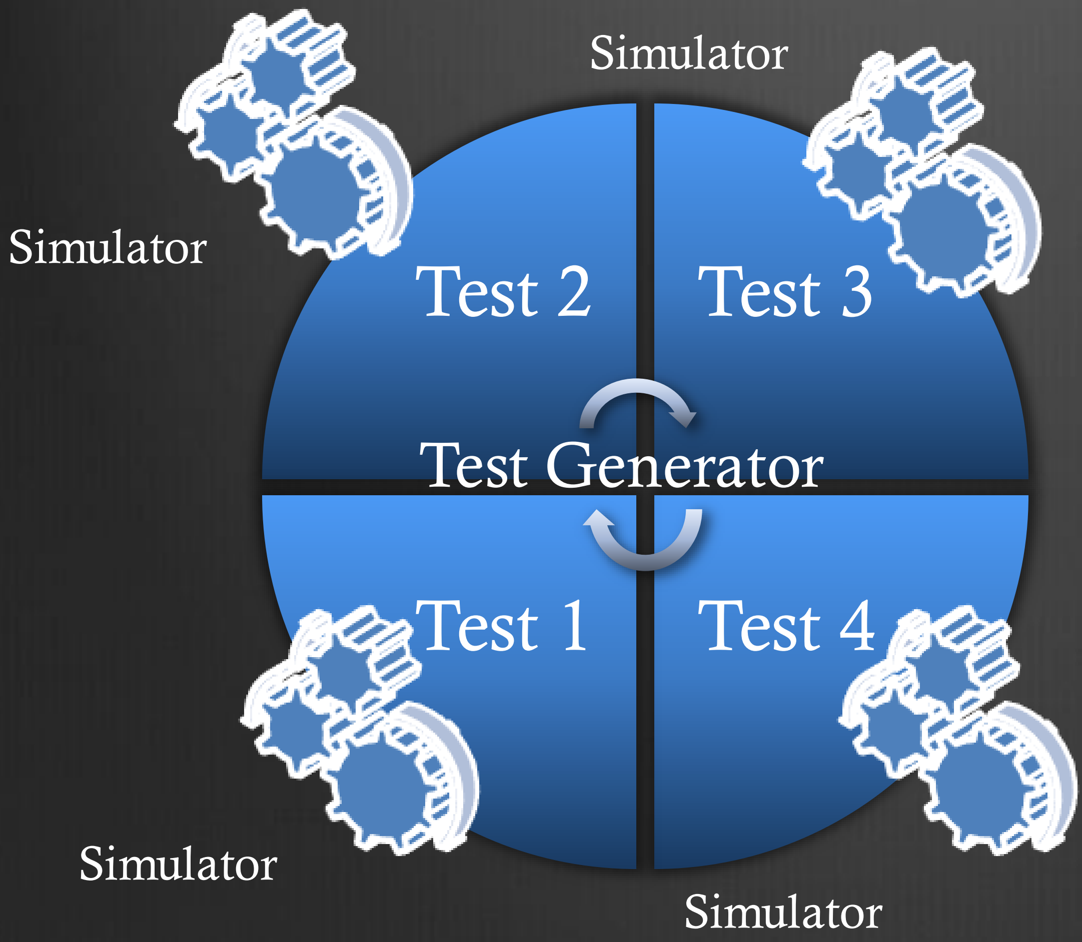
Generator	Relative Runtime
UniGen	470
UniGen2	21
Desired Uniform Generator*	10
Simple SAT solver	1

*: Based on EDA Industry

Outline

- UniGen2: Next Generation UniGen
  - Core Technical Ideas
  - Experimental Results
- **Parallelization of Constrained Random Simulation**
- Conclusion

Current Paradigm of Simulation-based Verification



- Can not be parallelized since test generators maintain "global state"
- Loses theoretical guarantees (if any) of uniformity

New Paradigm of Simulation-based Verification

Simulator

Simulator

- Preprocessing needs to be done only once
- No communication required between different copies of the test generator
- Scales linearly with number of cores in practice

Simulator

Simulator

Desired Performance with 2 cores

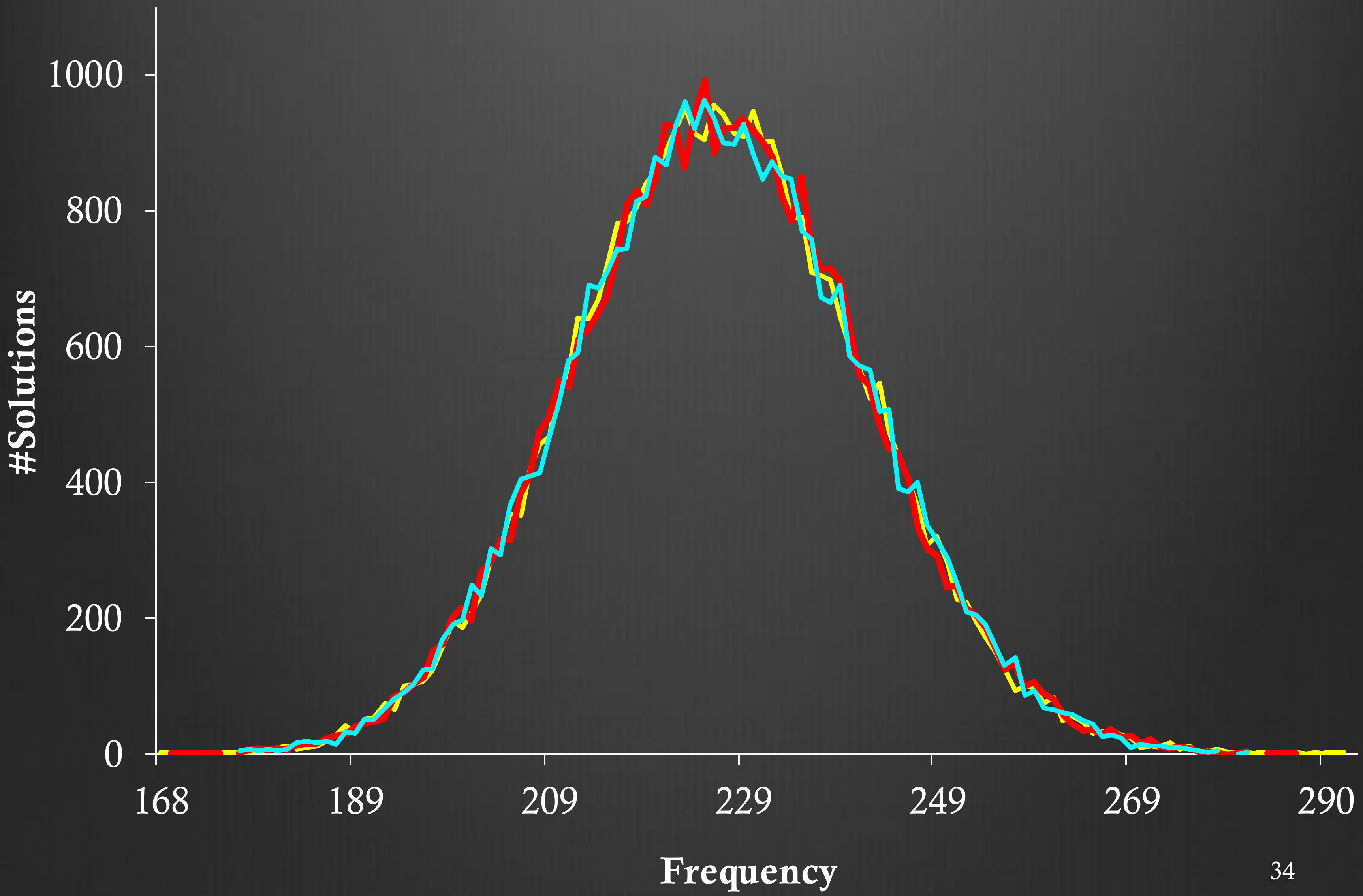
Generator	Relative Runtime
UniGen	470
UniGen2	21
Parallel UniGen2 (2 cores)	~10
Desired Uniform Generator*	10
Simple SAT solver	1

*: Based on EDA Industry

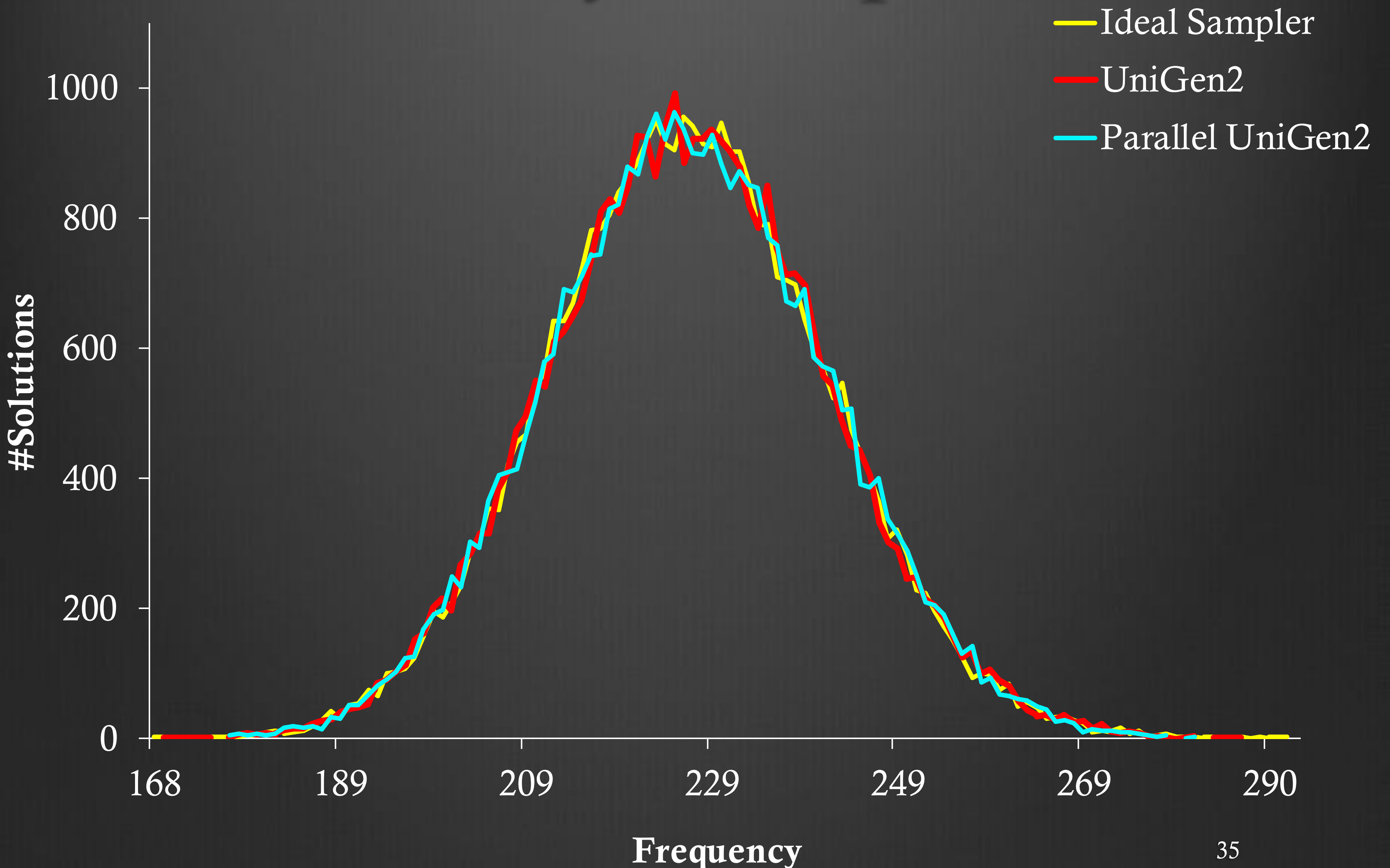
Uniformity Comparison

- Benchmark with 16,384 solutions
- Ideal Generator: Enumerate all solutions and pick one randomly
- Generated 4M samples for Ideal, UniGen2 & parallel (on 12 cores) UniGen2
- Group solutions according to their frequency
- Plot # of solutions vs Frequency
  - (200,250): 250 solutions appeared 200 times each
- In theory, we expect a Poisson distribution

Uniformity Comparison



Uniformity Comparison



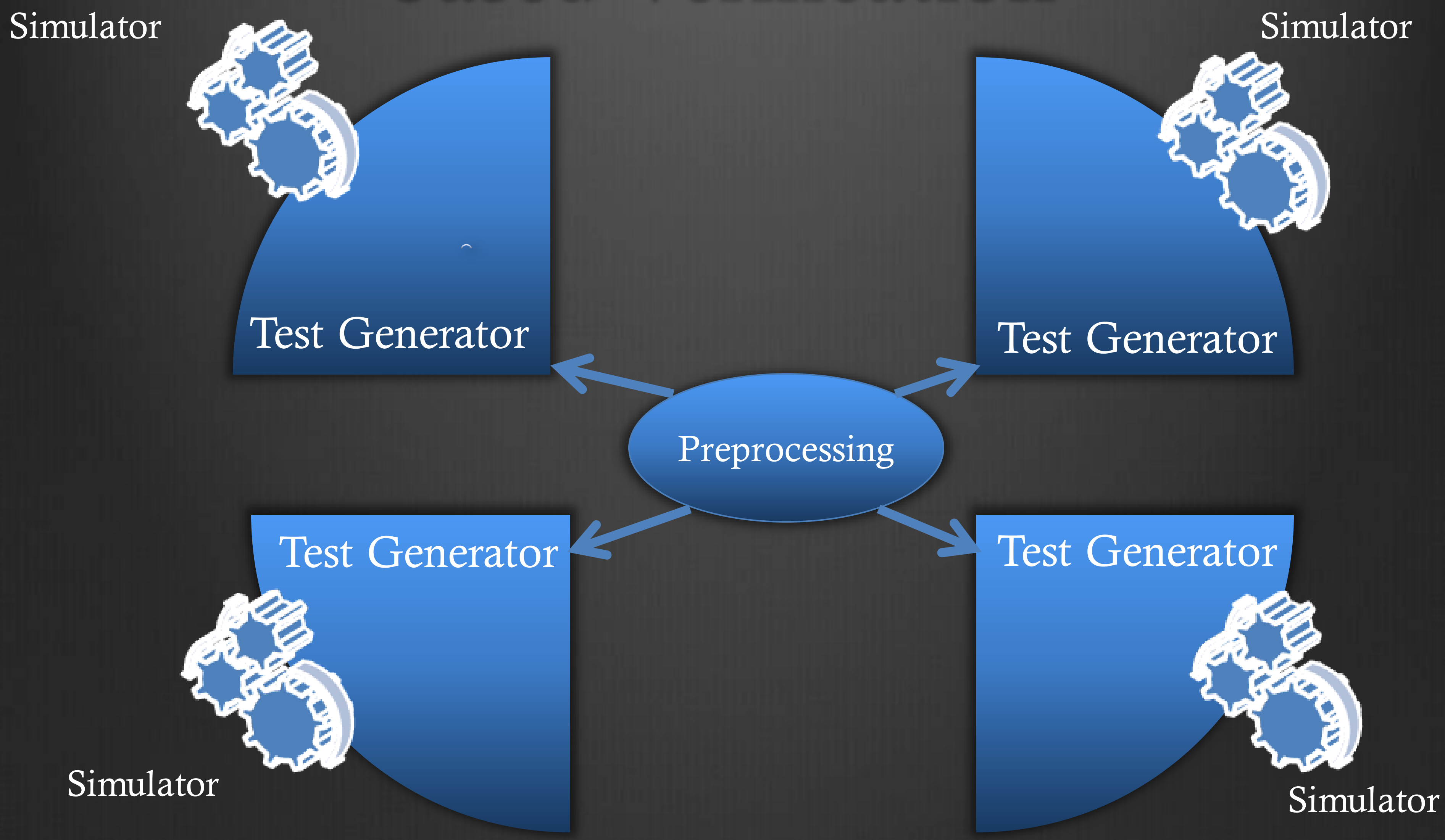
Outline

- UniGen2: Next Generation UniGen
  - Core Technical Ideas
  - Experimental Results
- Parallelization of Constrained Random Simulation
- **Conclusion**

Takeaways

- Uniform generation has diverse applications
- Proposed the first scalable parallel approach that provides strong guarantees
- Requires ~~polynomial~~ constant number of SAT calls per sample
- Runs ~20 times faster than prior state-of-the-art tools, even on a single core
- Scales linearly with number of cores

New Paradigm of Simulation-based Verification



And one more thing!

- Tool (along with source code) is available online:

http://tinyurl.com/unigen2