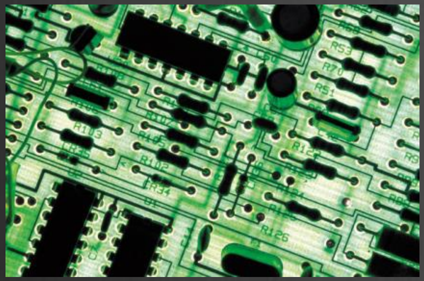
Sampling from combinatorial spaces: Achieving the fine balancing act between independence and scalability

Kuldeep Meel
Rice University

Joint work with Supratik Chakraborty, Daniel J. Fremont, Sanjit A. Seshia, Moshe Y. Vardi

How do we guarantee that systems work *correctly* ?



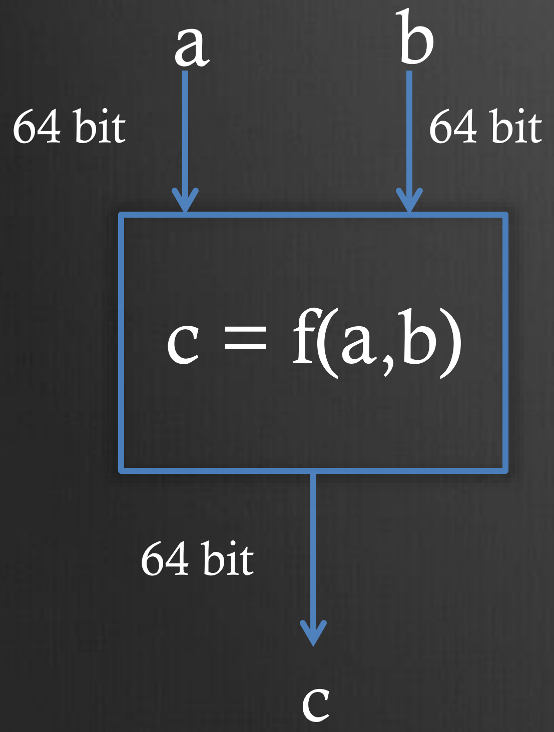
Functional Verification

- 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!

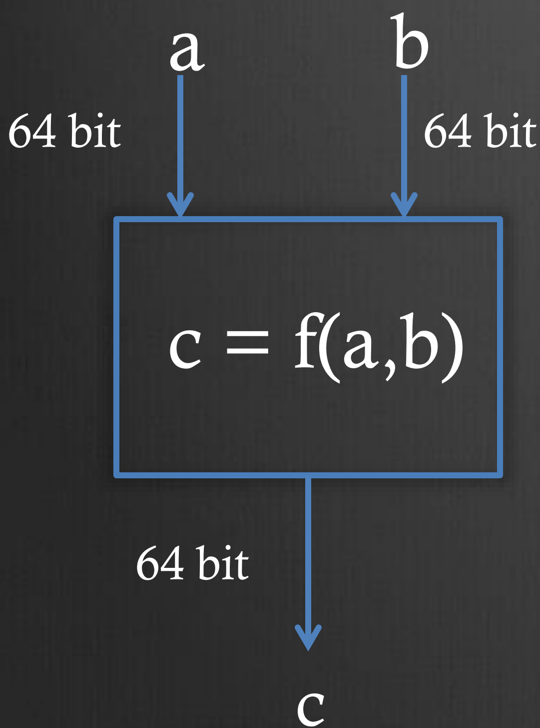
Motivating Example



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

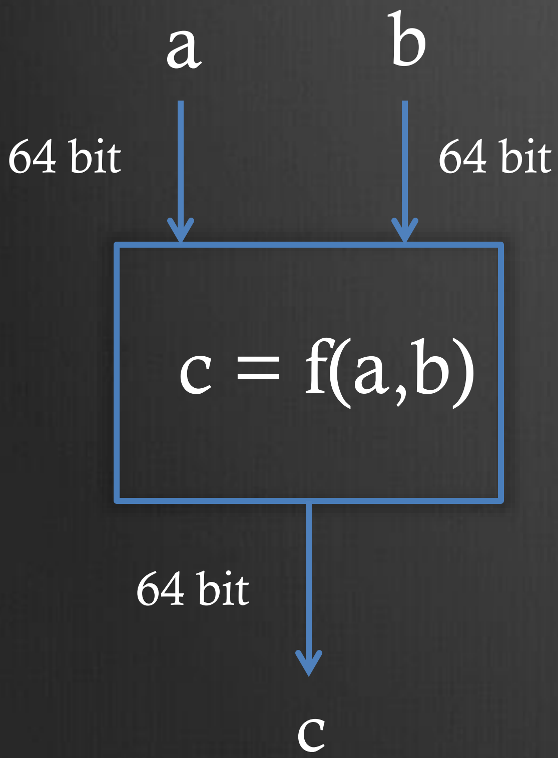


Sources for Constraints

- 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. $232 *_{32} a + b \,!= 1100$
  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

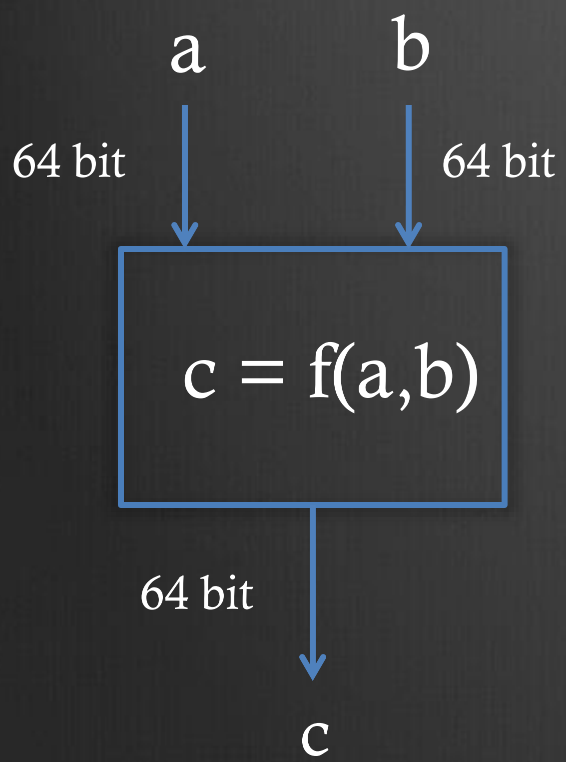


Sources for Constraints

- 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

↓

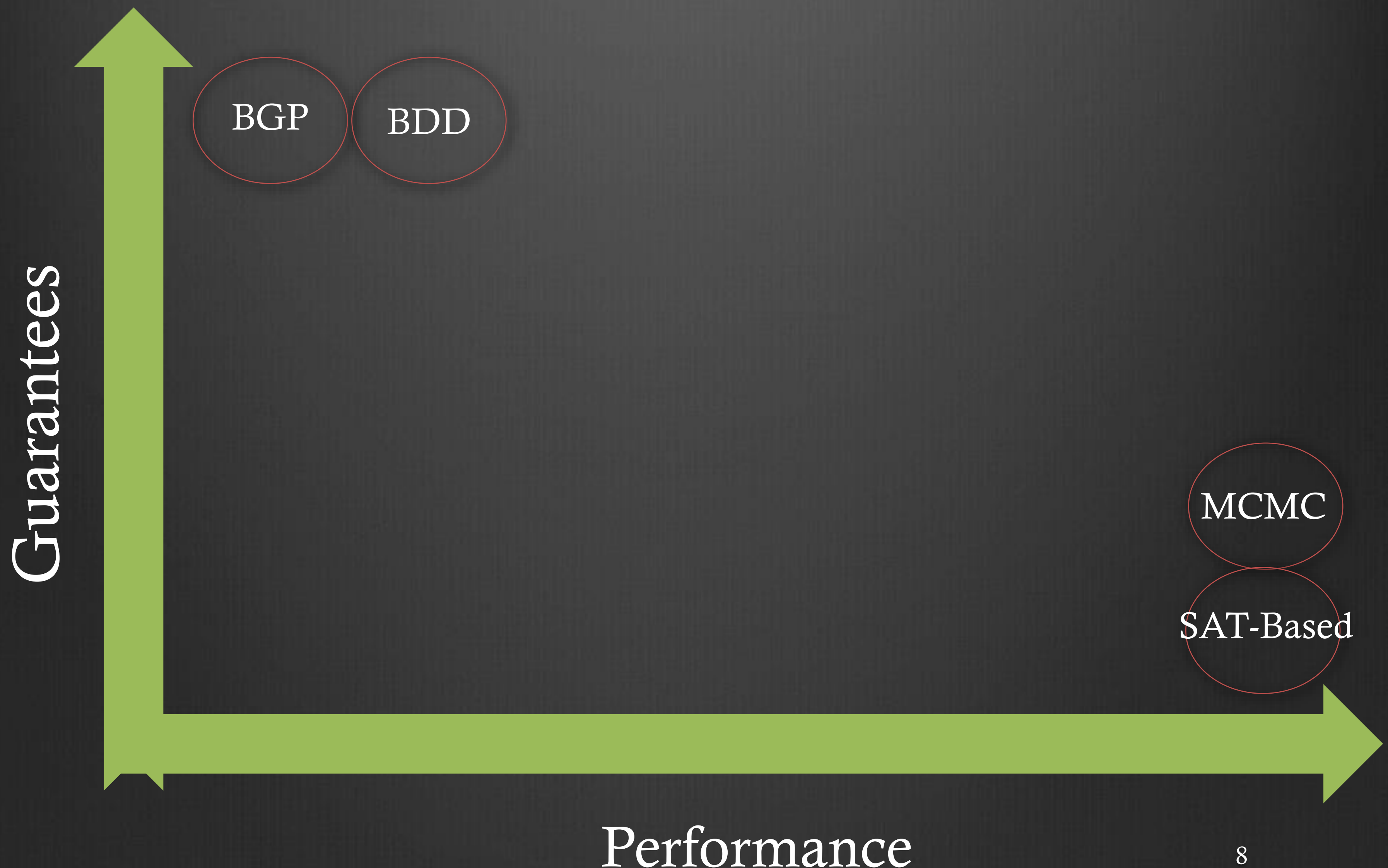
SAT Formula

↓

Sample satisfying assignments uniformly at random

Scalable Uniform Generation of SAT Witnesses

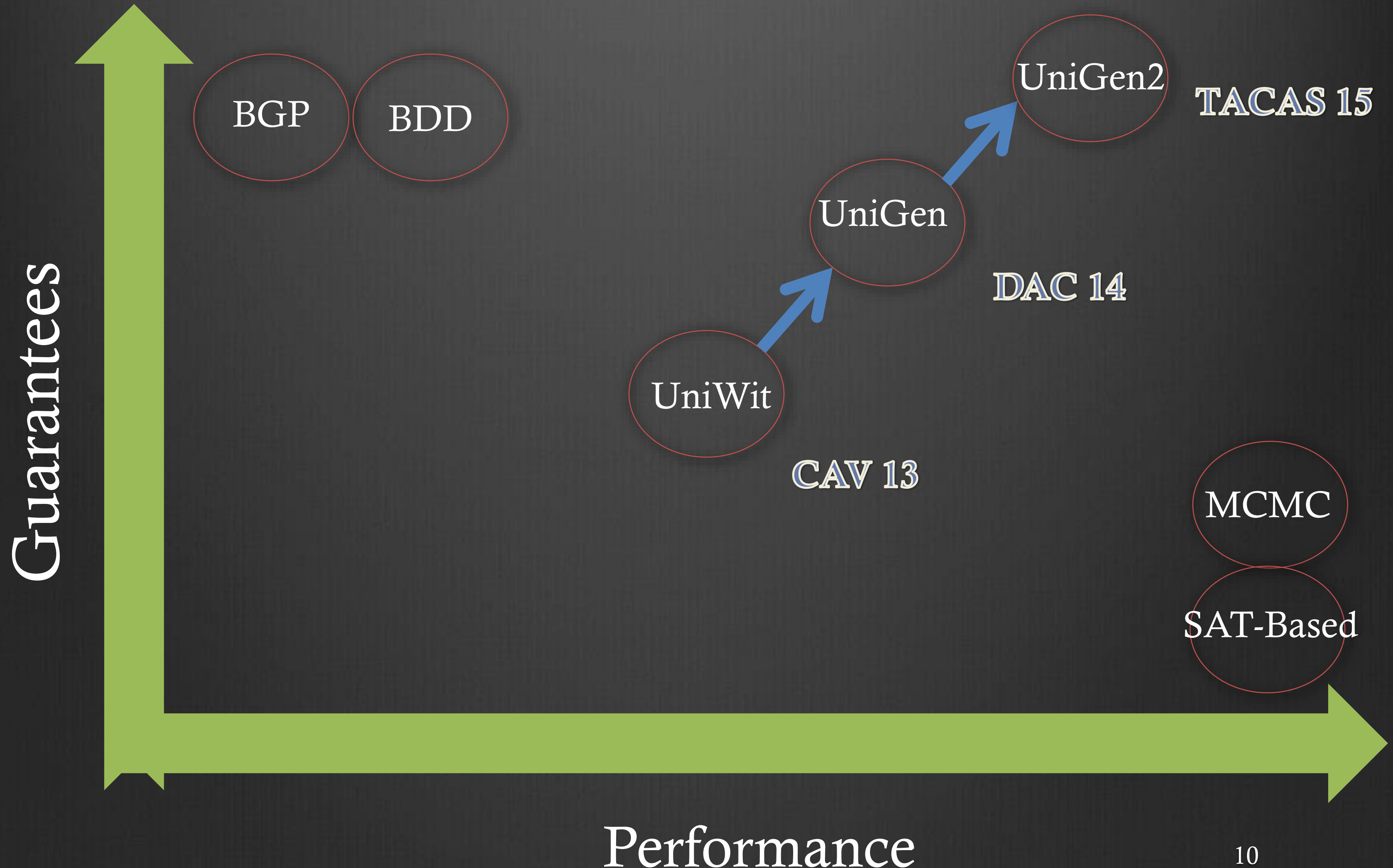
Prior Work



EDA Industry's Desired Performance

Generator	Relative Runtime
XORSample' (weak guarantees)	~50000
Desired Uniform Generator*	10
Simple SAT solver	1

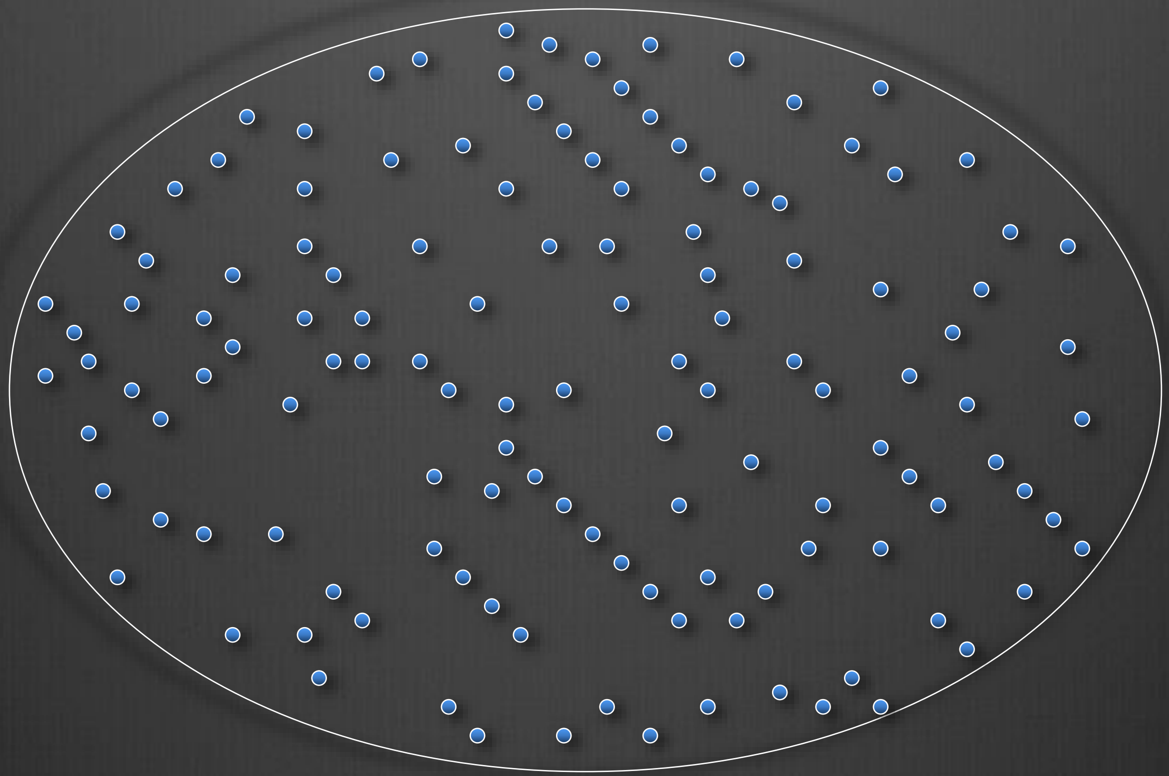
Our Contribution



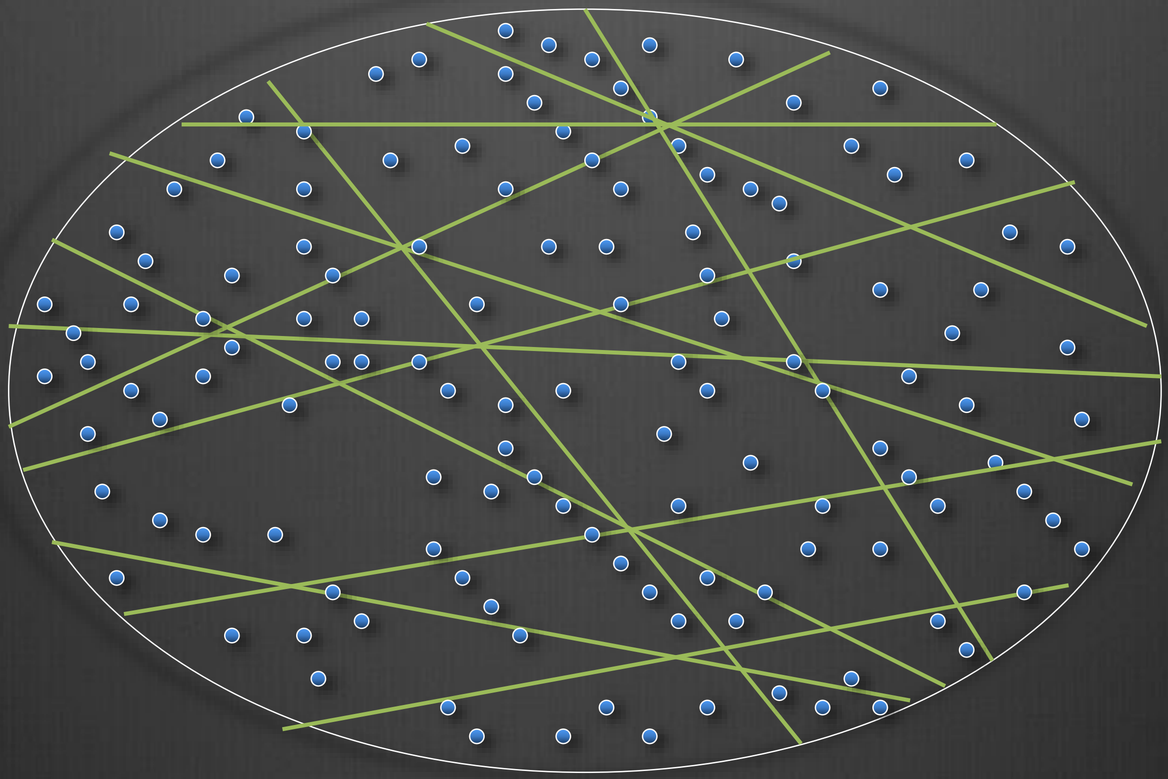
Outline

- Losing Independence of hashing functions
- Losing Independence among samples
- Parallelization of Constrained Random Simulation
- Conclusion

Main Idea



Partitioning into cells

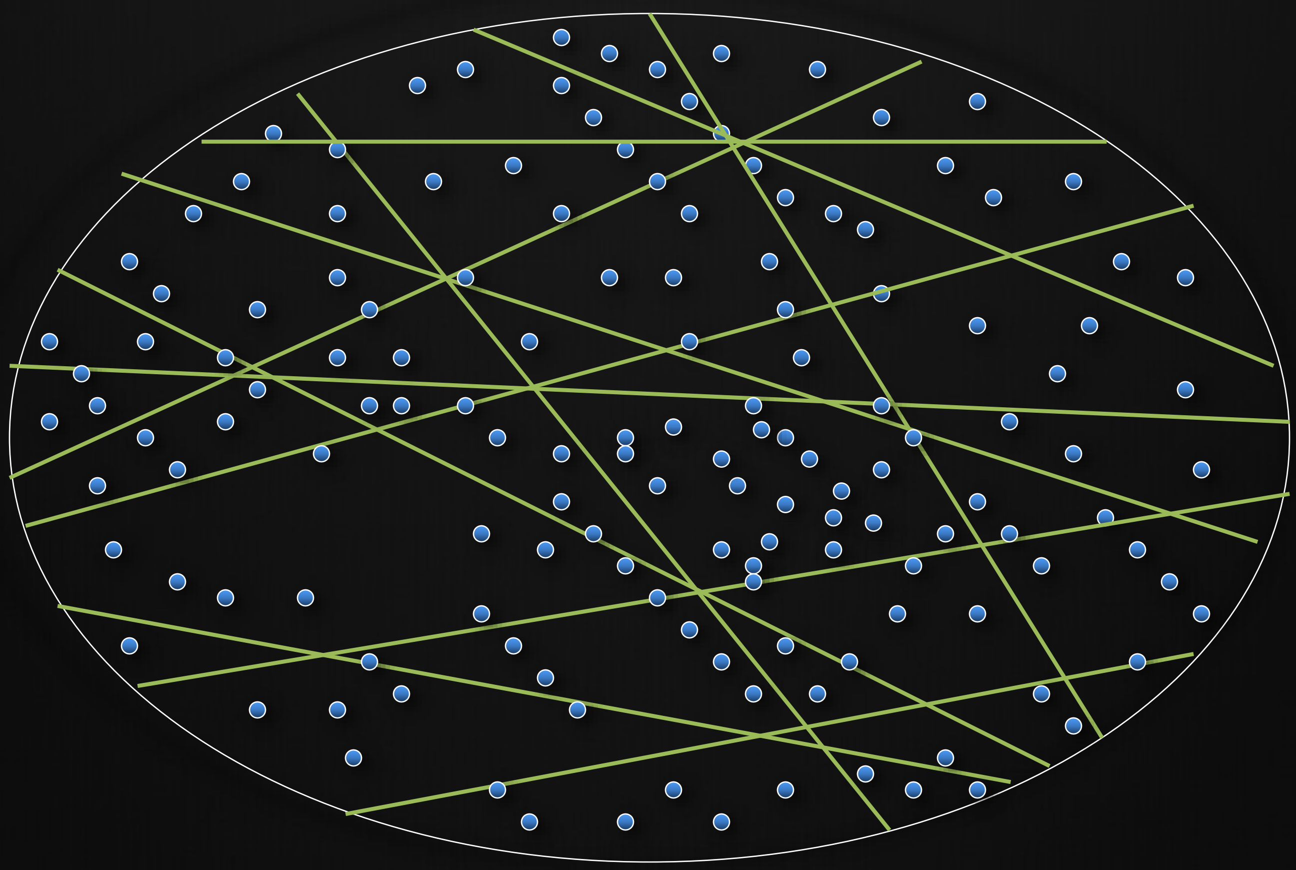


Cells should be roughly equal in size and small enough to enumerate completely

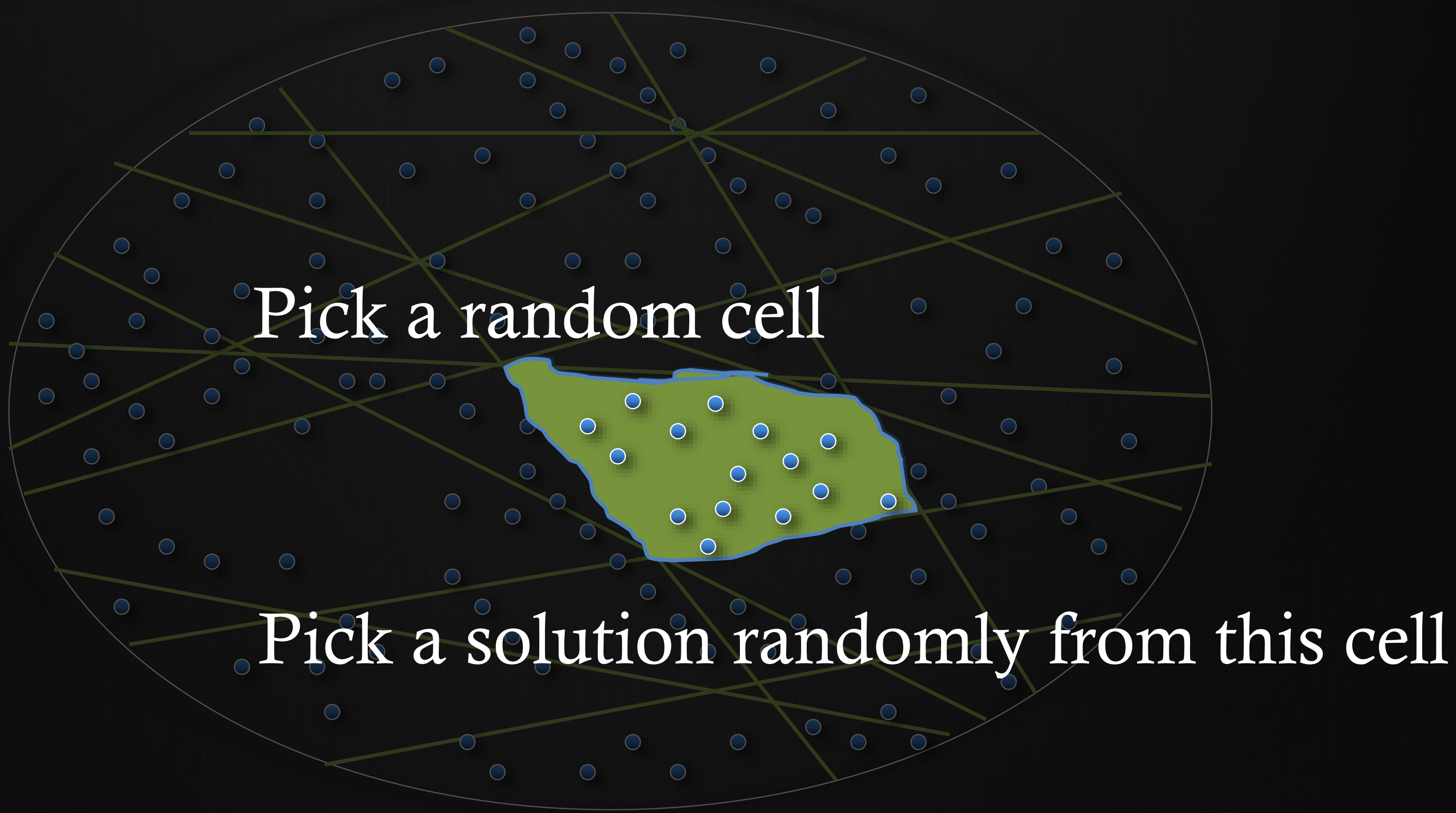
Partitioning into cells

- 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



Partitioning into cells

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

Universal Hashing
(Carter-Wegman 1979)

Universal Hashing

- Hash functions: mapping $\{0,1\}^n$ to $\{0,1\}^m$
  - (2^n elements to 2^m cells)
- Random inputs => All cells are *roughly* equal (in <u>expectation</u>)
- Universal family of hash functions:
  - Choose hash function randomly from family
  - For ***arbitrary*** distribution on inputs => All cells are ***roughly equal*** (in <u>expectation</u>)

r-Universal Hashing

- Each solution is hashed uniformly
- Every r-subset of solutions is hashed independently
- For r=2,

$$\forall \text{ distinct } y_1, y_2 \text{ and } \forall \alpha_1, \alpha_2$$
$$Pr[h(y_1) = \alpha_1 \wedge h(y_2) = \alpha_2] = Pr[h(y_1) = \alpha_1]Pr[h(y_2) = \alpha_2]$$

- r-wise universal hash function => polynomial of degree r-1

Why Independence matters?

We pick a random cell and define following random variables

$$I_k = 1 \text{ if } y_k \text{ is in the cell}$$

$$\text{Let } I_1, I_2, I_3, ...I_n \text{be r-wise independent variables in } [0,1],$$

$$\text{then for } I = \sum I_k$$

$$Pr[|I - \mu| < \delta\mu] \geq c^{-r}$$

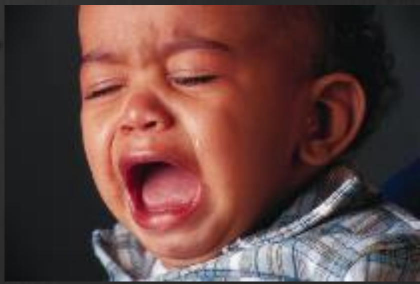
Number of solutions in a randomly picked cell

Deviation

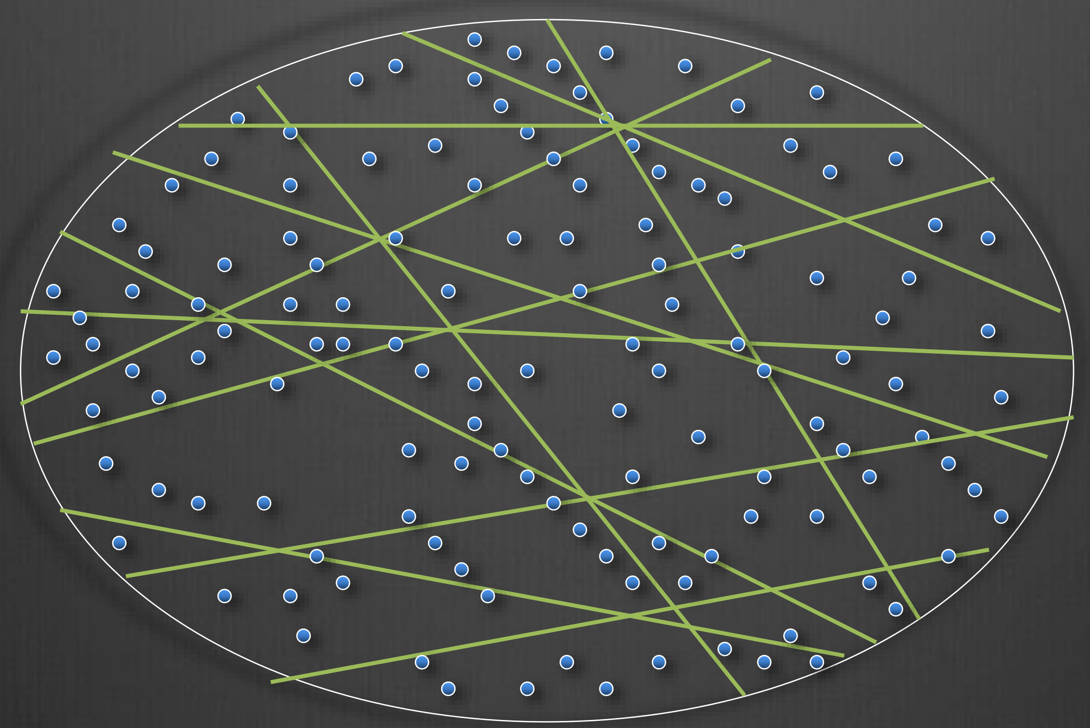
Tradeoff

Higher universality => Stronger Guarantees

Higher universality => Polynomials of higher degree

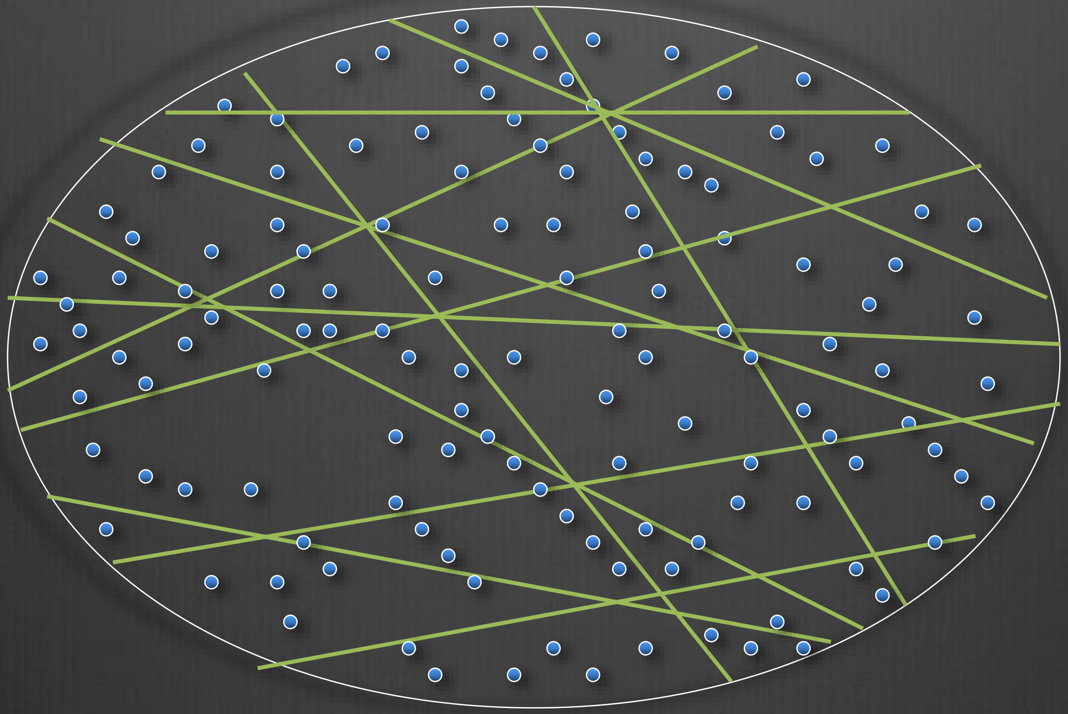


Prior Work



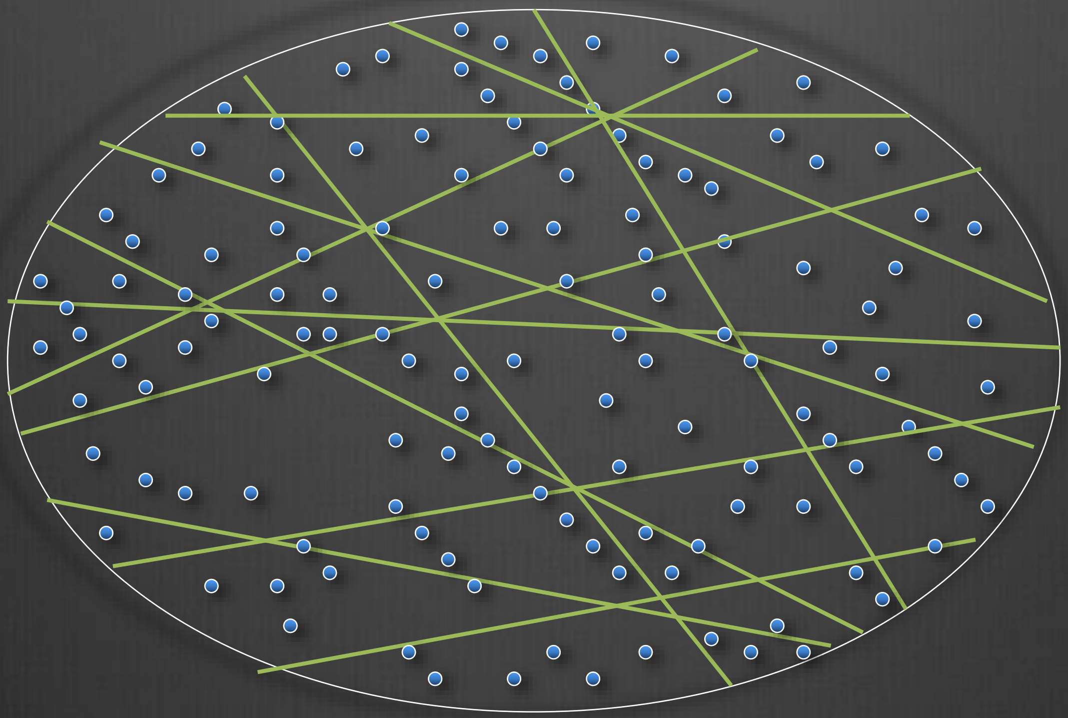
- Choose n-wise universal hash functions and all cells are guaranteed to be small with high probability
- Theoretical guarantee of uniformity

What if we lower the hashing to 3-universal



All cells are not guaranteed to be small anymore with high probability

What if we lower the hashing to 3-universal



But a randomly picked cell is guaranteed to be small with high probability

Guarantees of almost-uniformity

Strong Theoretical Guarantees

- Uniformity (BGP with n-universal)

$$\Pr[\text{y is output}] = \frac{1}{|R_F|}$$

- Almost- Uniformity (UniGen with 3-universal)

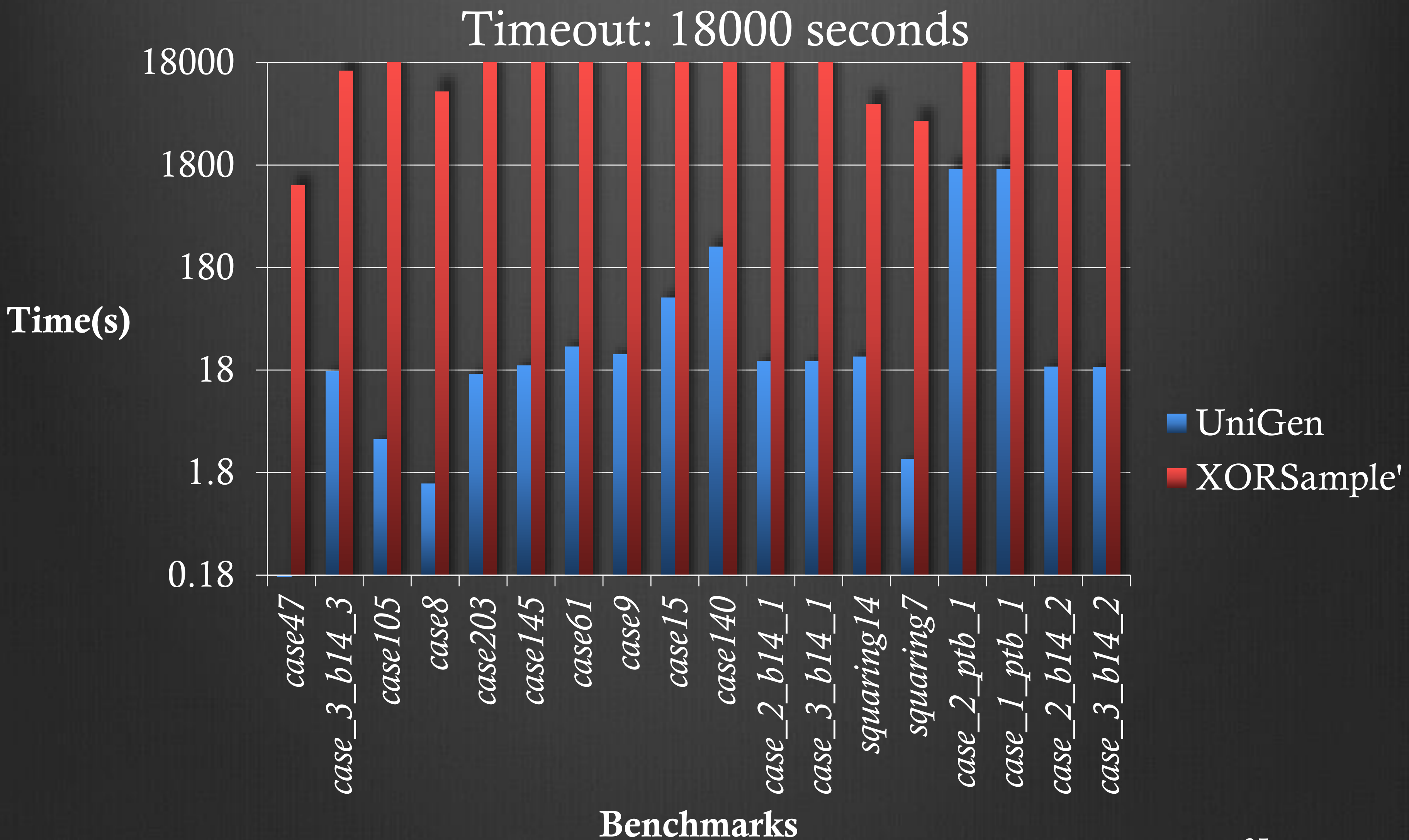
$$\forall y \in R_F, \frac{1}{(1+\varepsilon)|R_F|} \leq \Pr[\text{y is output}] \leq (1+\varepsilon)\frac{1}{|R_F|}$$

- Polynomial number of SAT calls

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

2-3 Orders of Magnitude Faster



Runtime Performance

Experiments over 200+ benchmarks

Generator	Relative Runtime
XORSample' (weak guarantees)	~50000
UniGen	470
Desired Uniform Generator*	10
Simple SAT solver	1

*: Based on EDA Industry

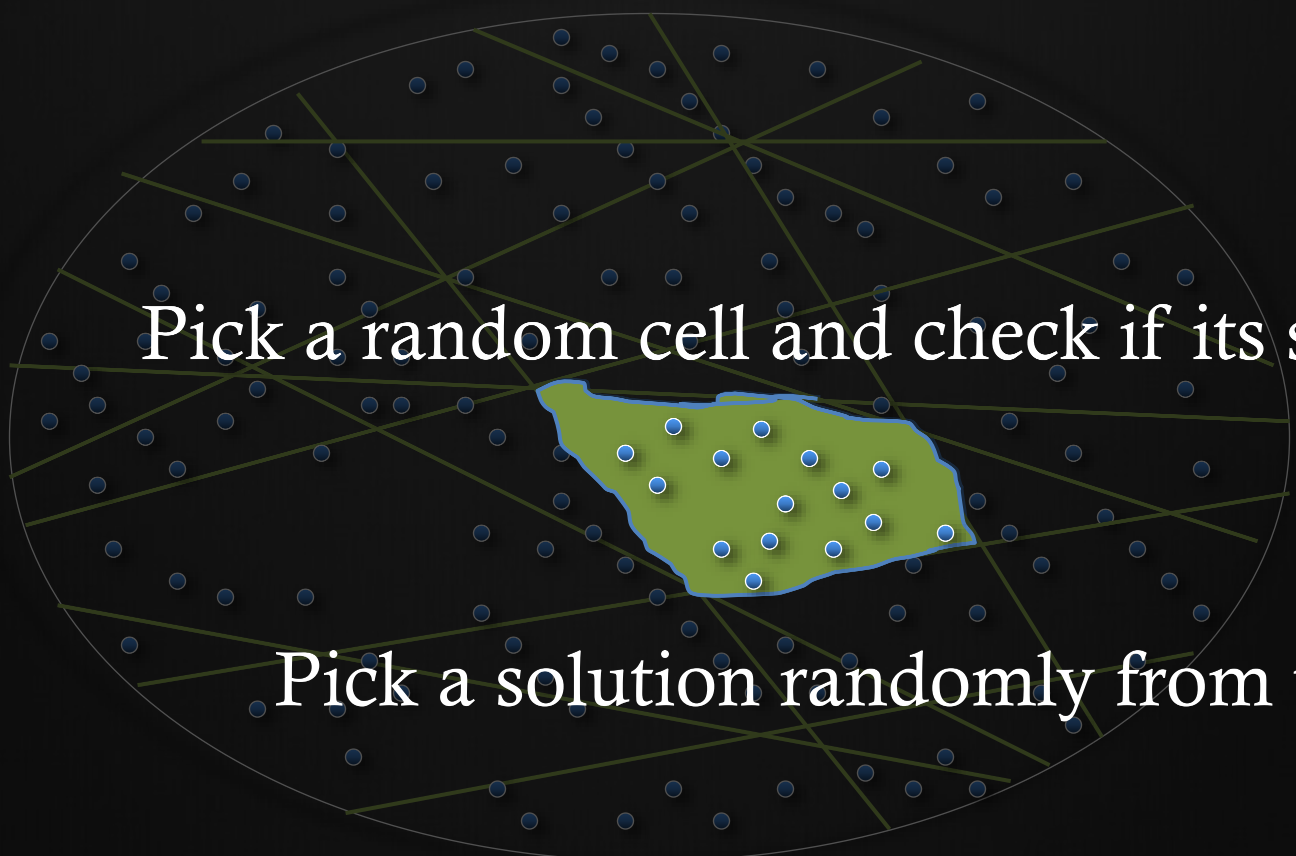
Outline

- Losing Independence of hashing functions
- **Losing Independence among samples**
- Parallelization of Constrained Random Simulation
- Conclusion

How many solutions are generated per sample?

>LoThresh

UniGen

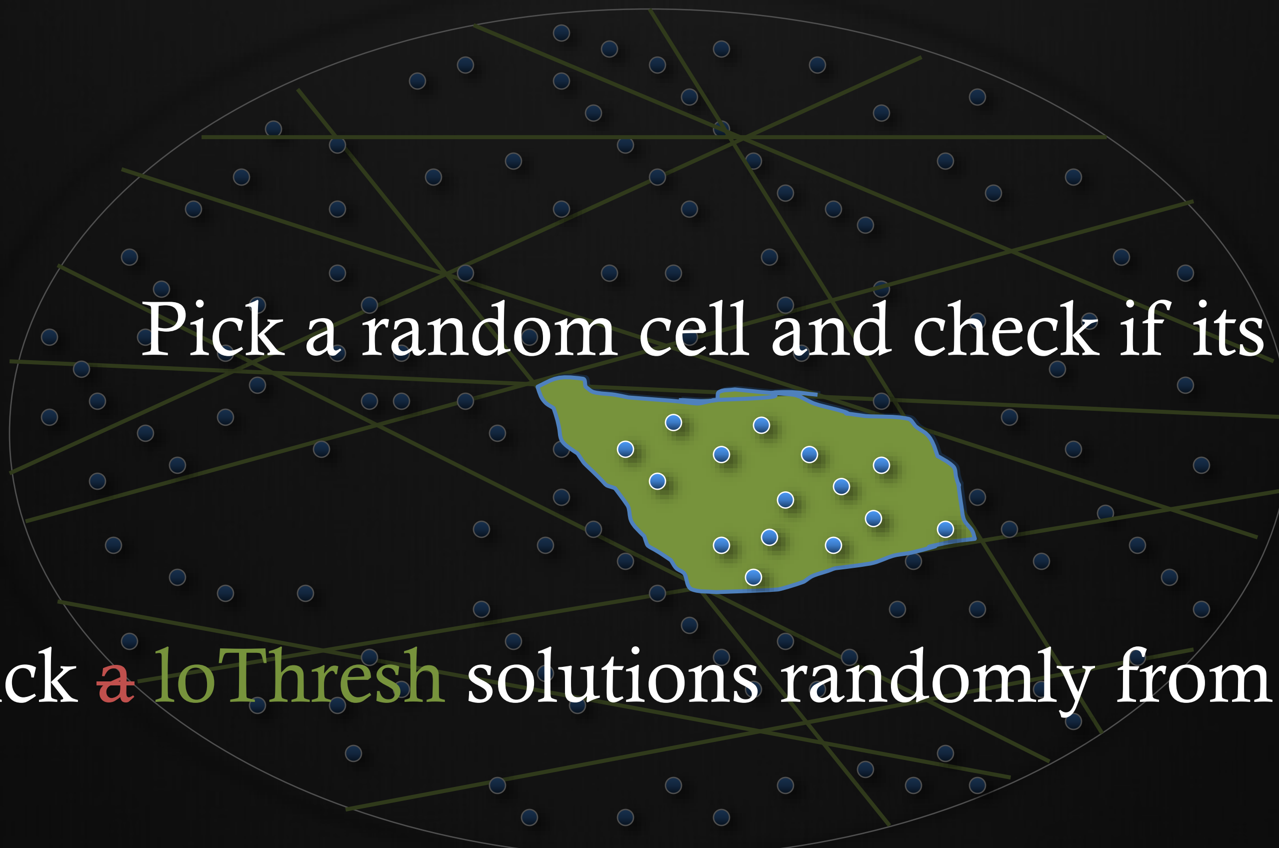


Pick a random cell and check if its small

Pick a solution randomly from this cell

# of solutions in a small cell is between loThresh and hiThresh

UniGen



Pick a random cell and check if its small

Pick ~~a~~ loThresh solutions randomly from this cell

# of solutions in a small cell is between loThresh and hiThresh

3-Universal and Independence of Samples

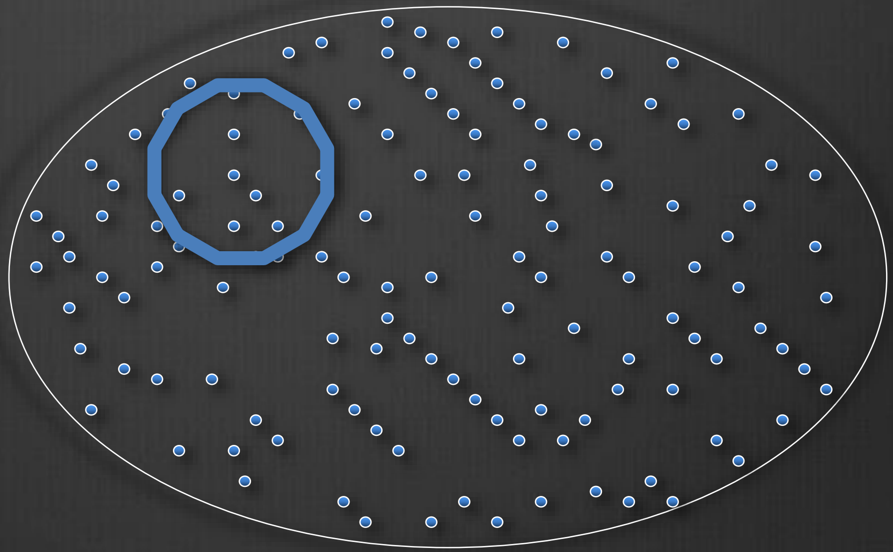
3-Universal hash functions:

- 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**

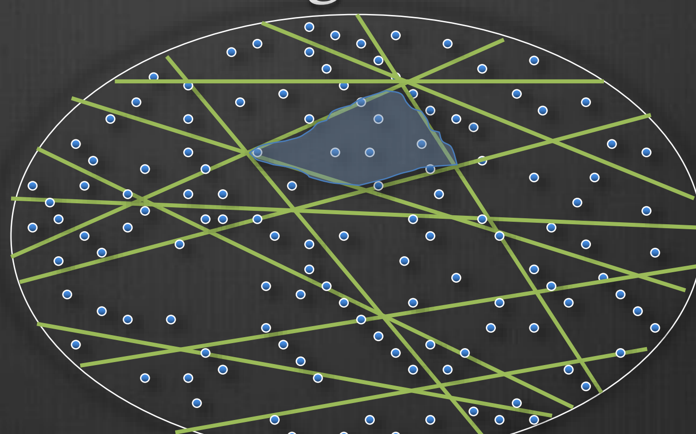
3-Universal and Independence of samples

- Choosing up to 3 samples => Full Independence between samples
- Independence provides coverage guarantees.



3-Universal and Independence of samples

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



Strong Guarantees

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

  $\forall y \in R_F,$

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

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

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

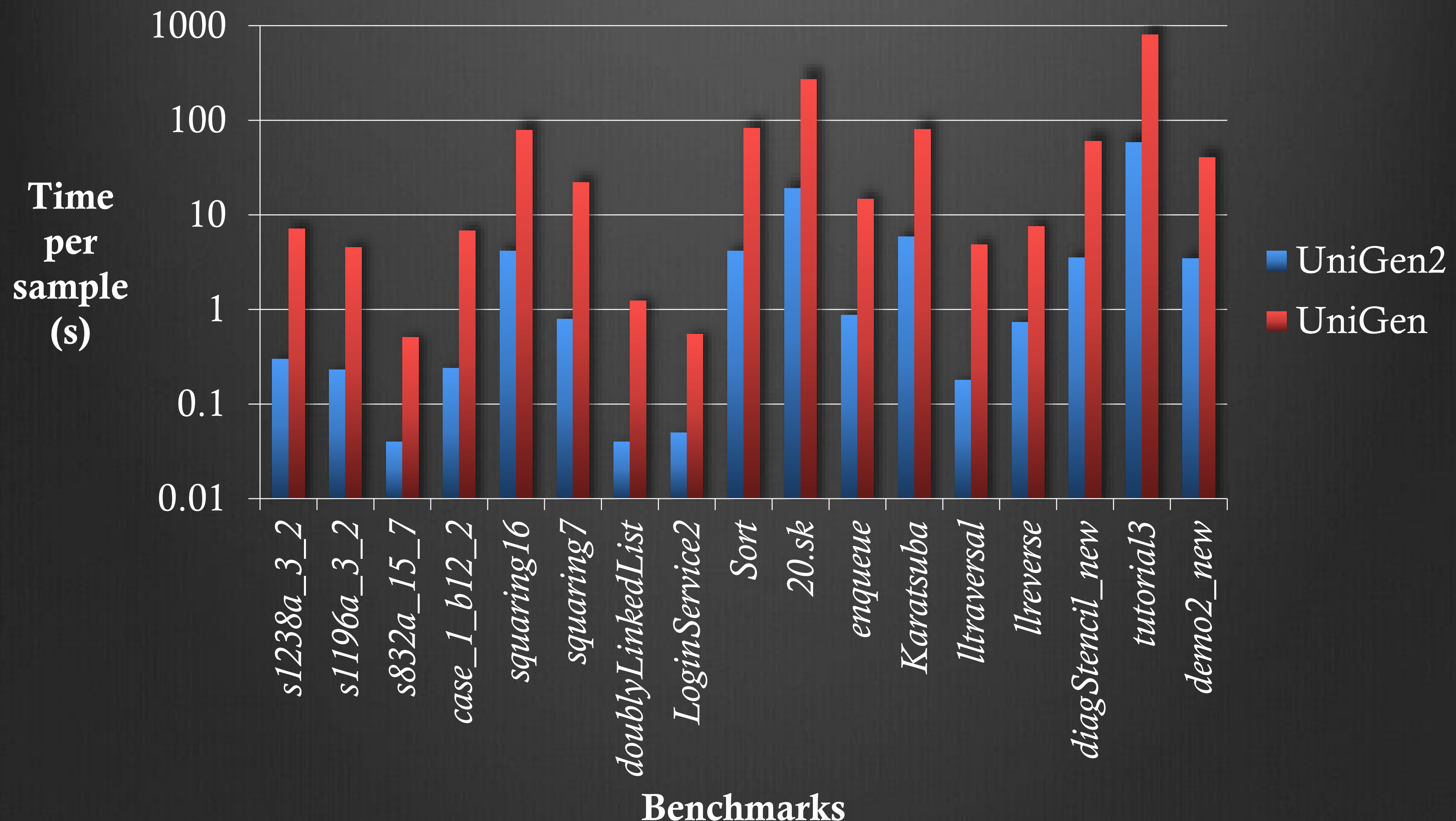
bug frequency $f = 1/10^4$

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

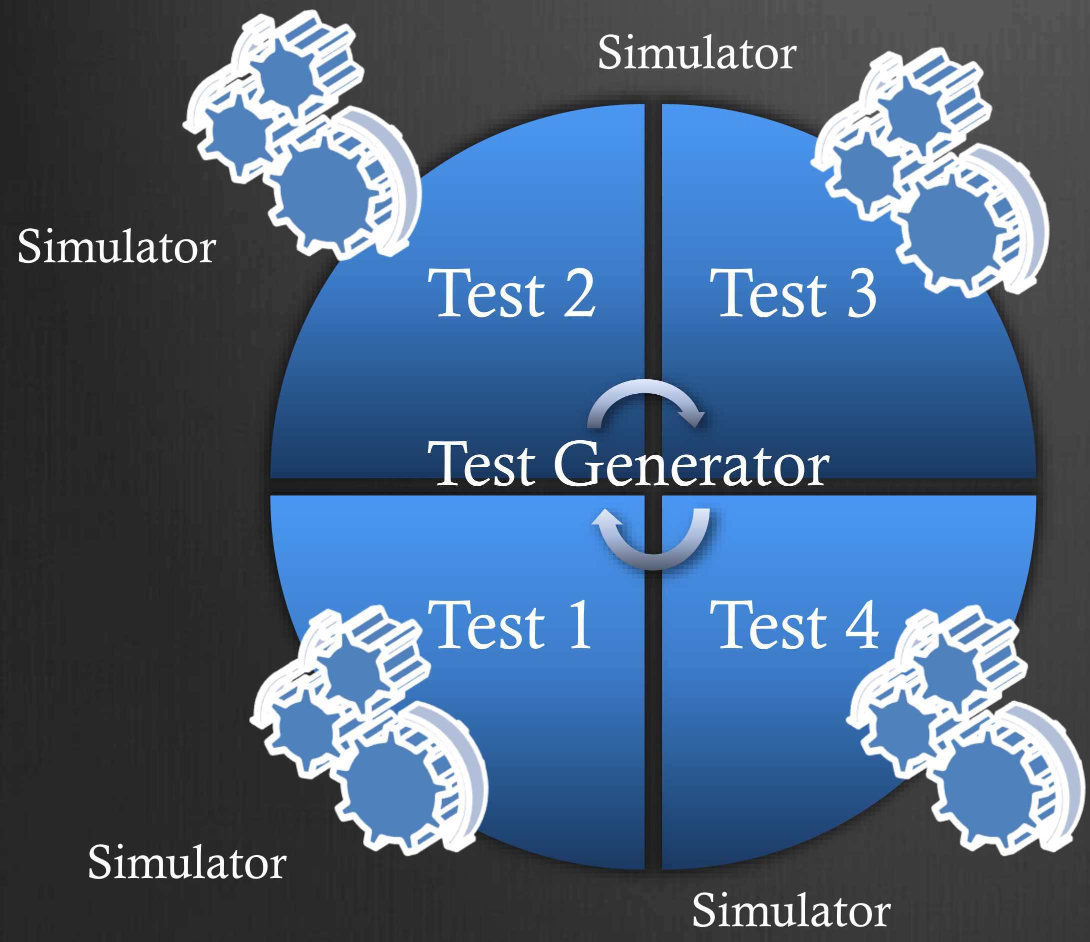
Generator	Relative Runtime
XORSample' (weak guarantees)	~50000
UniGen	470
UniGen2	21
Desired Uniform Generator*	10
Simple SAT solver	1

*: Based on EDA Industry

Outline

- Losing Independence of hashing functions
- Losing Independence among samples
- <u>**Parallelization of Constrained Random Simulation**</u>
- 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

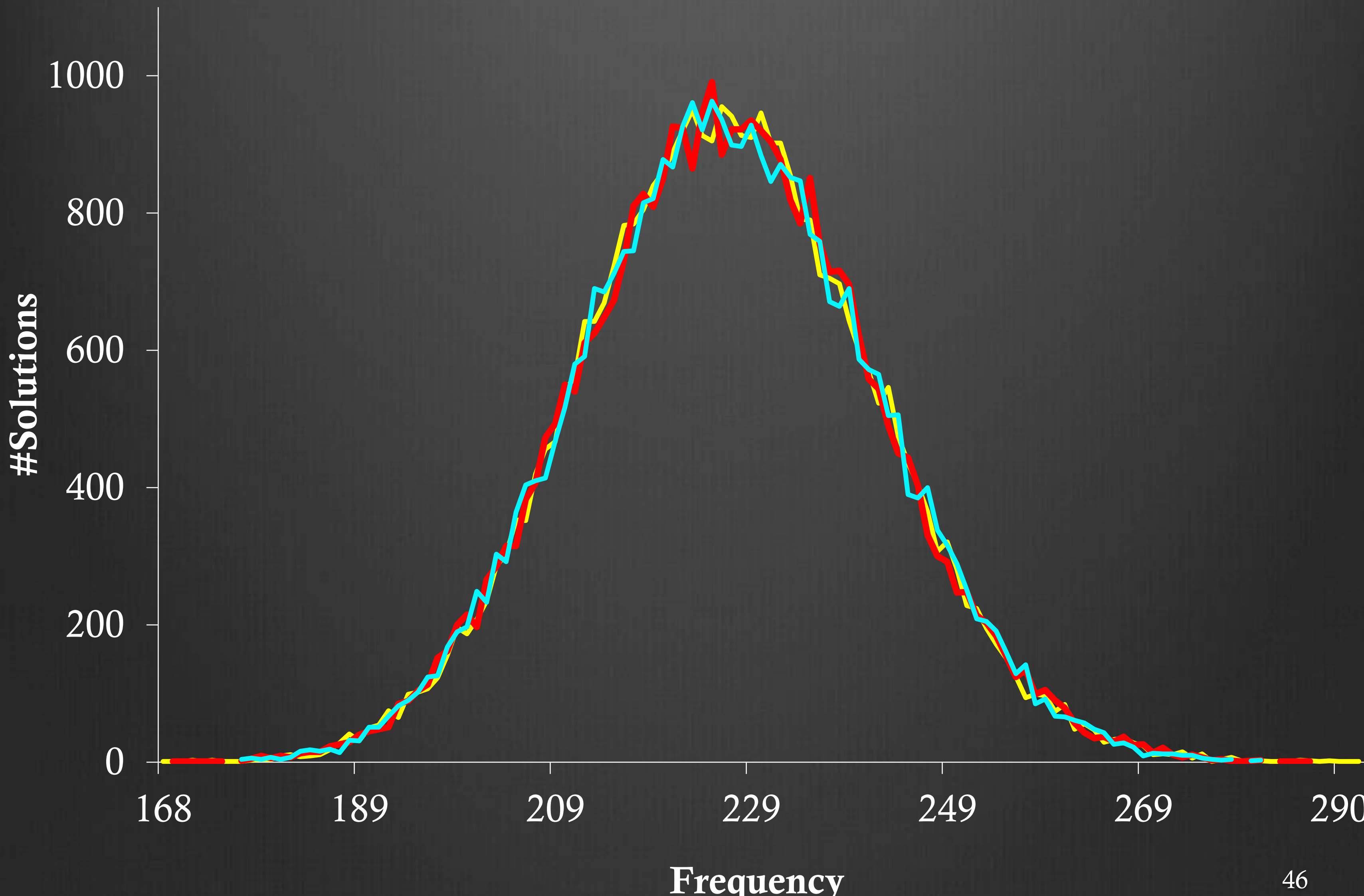
Generator	Relative Runtime
UniGen	470
UniGen2	21
Parallel UniGen2 (2 cores)	~10
Desired Uniform Generator*	10
Simple SAT solver	1

*: Based on EDA Industry

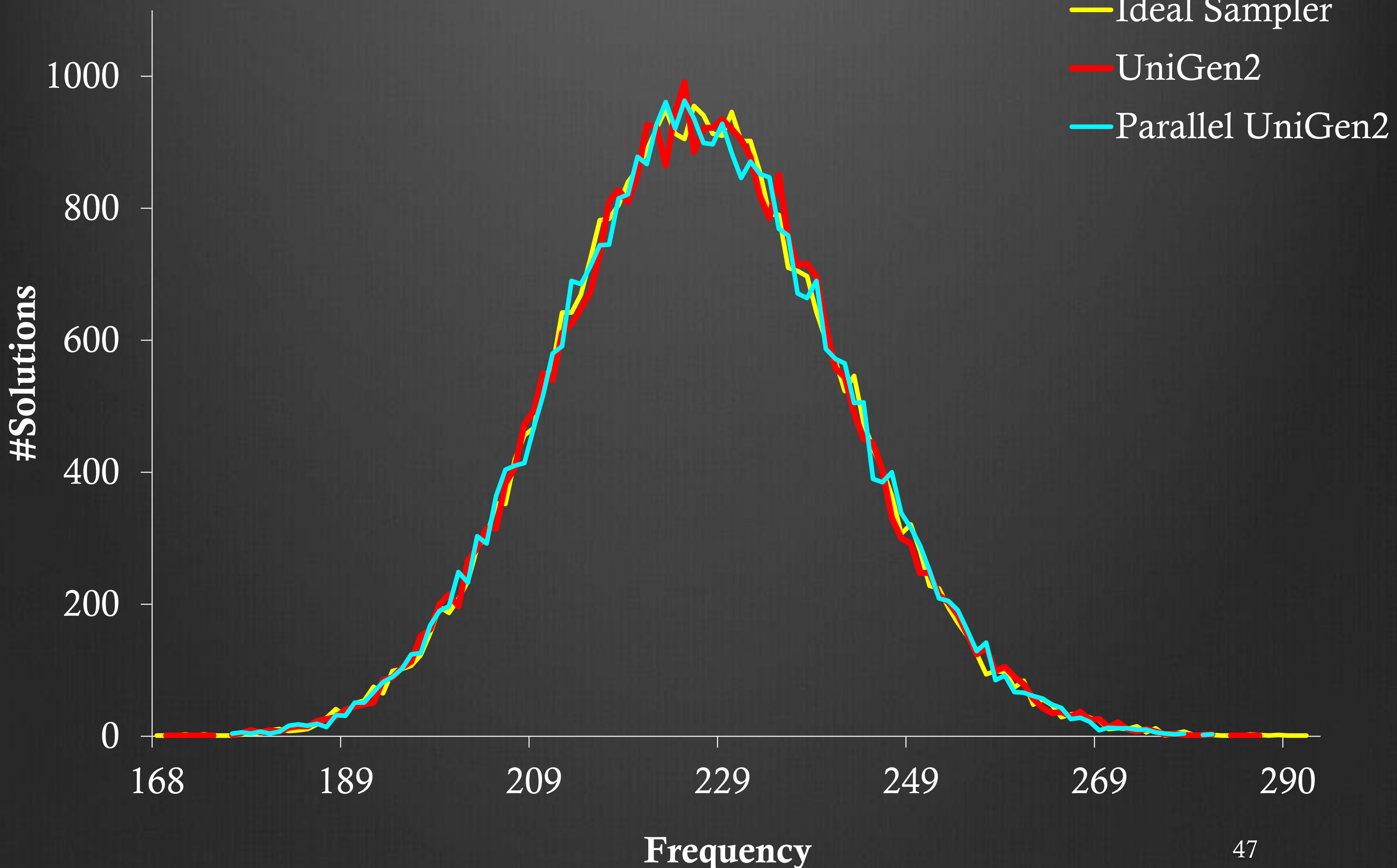
Uniformity Comparison

- 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

- Losing Independence of hashing functions
- Losing Independence among samples
- Parallelization of Constrained Random Simulation
- **Conclusion**

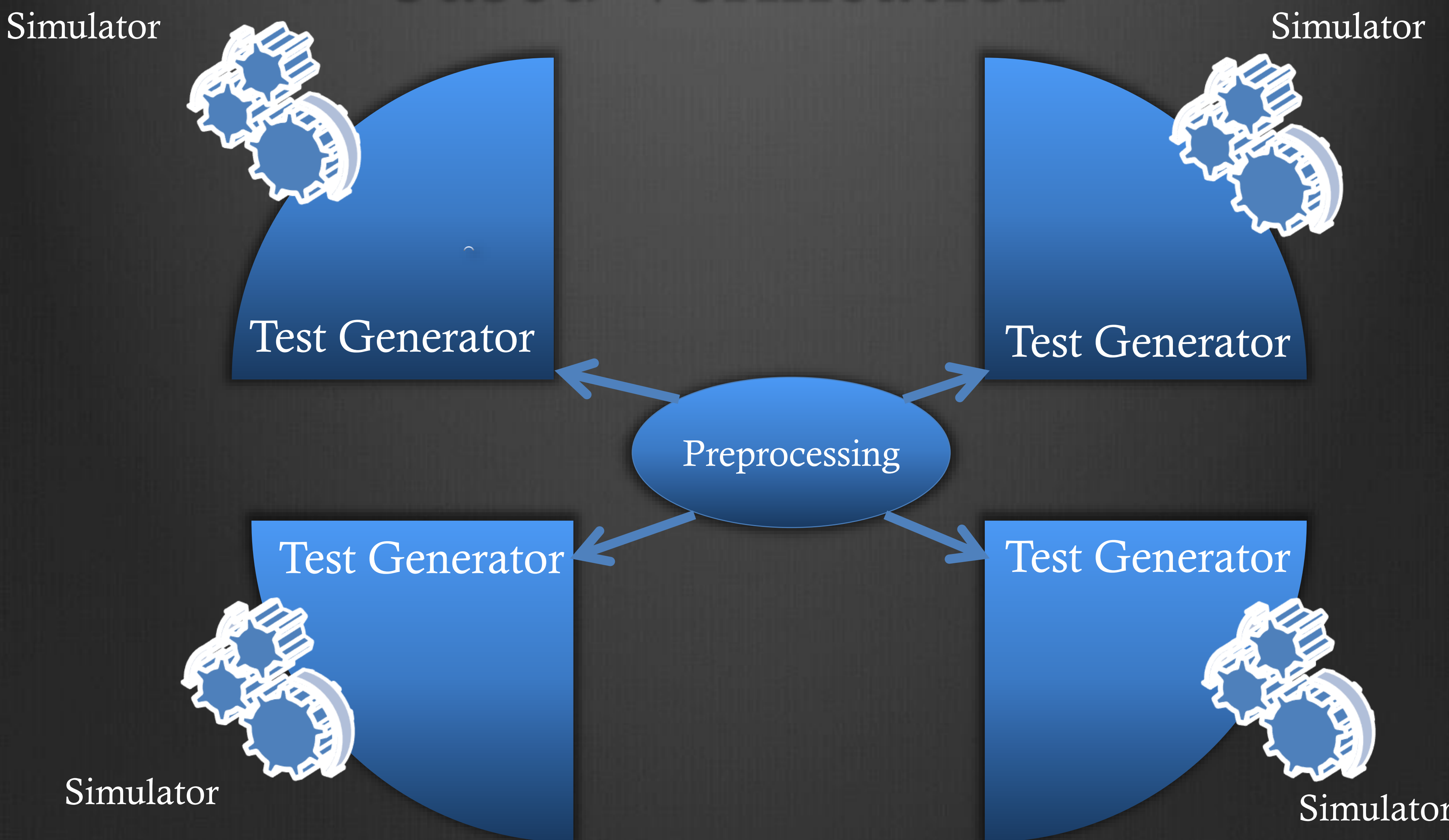
How well did we tradeoff Independence?

Relaxation Independence	Loss	Gain
Hashing	Uniformity to Almost Uniformity	• 2-3 orders of magnitude performance improvement
Sample	Weakened Almost Uniformity	• Still provides coverage guarantees • 20 x improvement • Parallelization • Achieved desired performance

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
- Scales linearly with number of cores
- Achieves desired performance by EDA Industry

New Paradigm of Simulation-based Verification



And one more thing!

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

  http://tinyurl.com/unigen2

- Visit www.kuldeepmeel.com for papers/reports