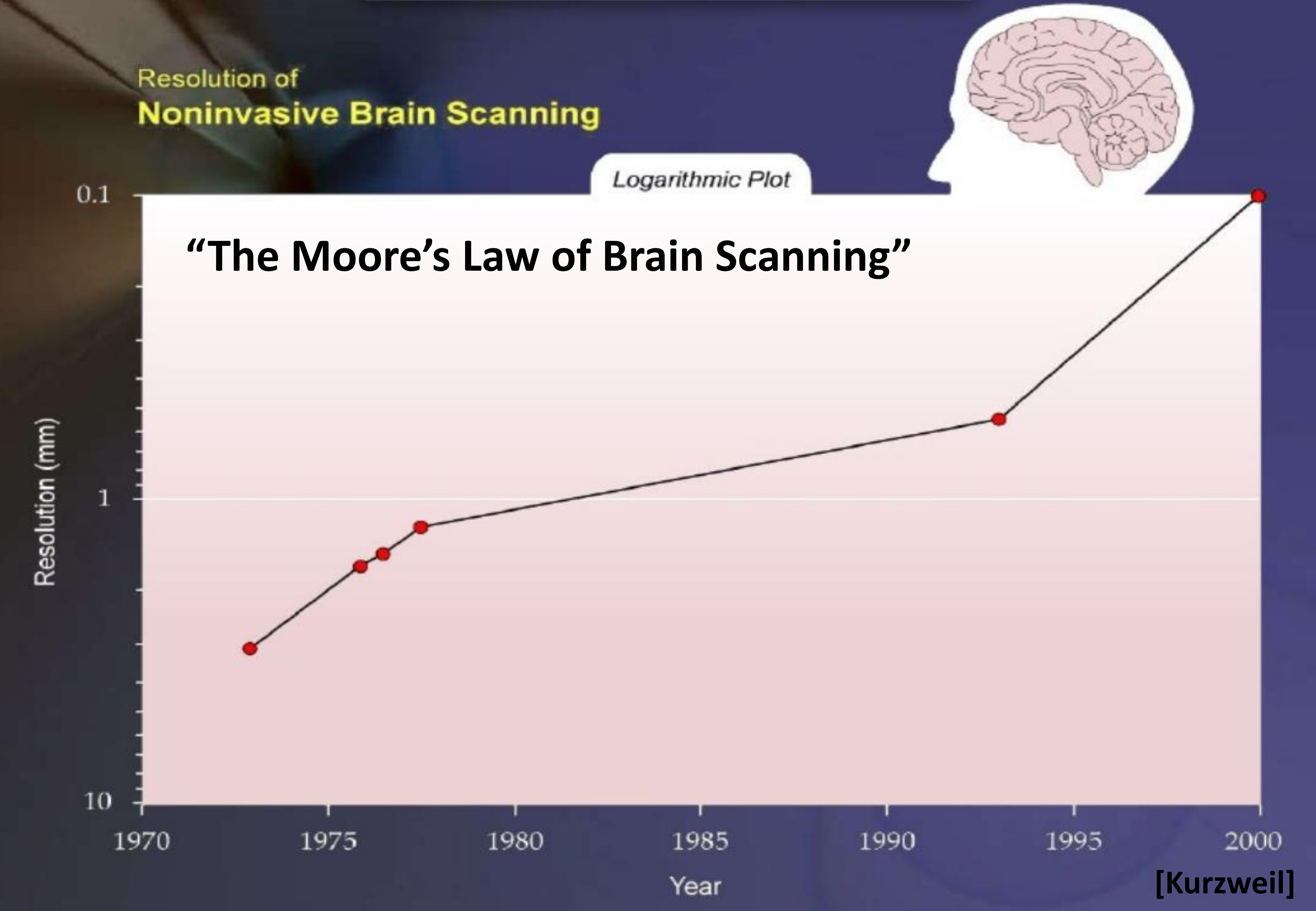
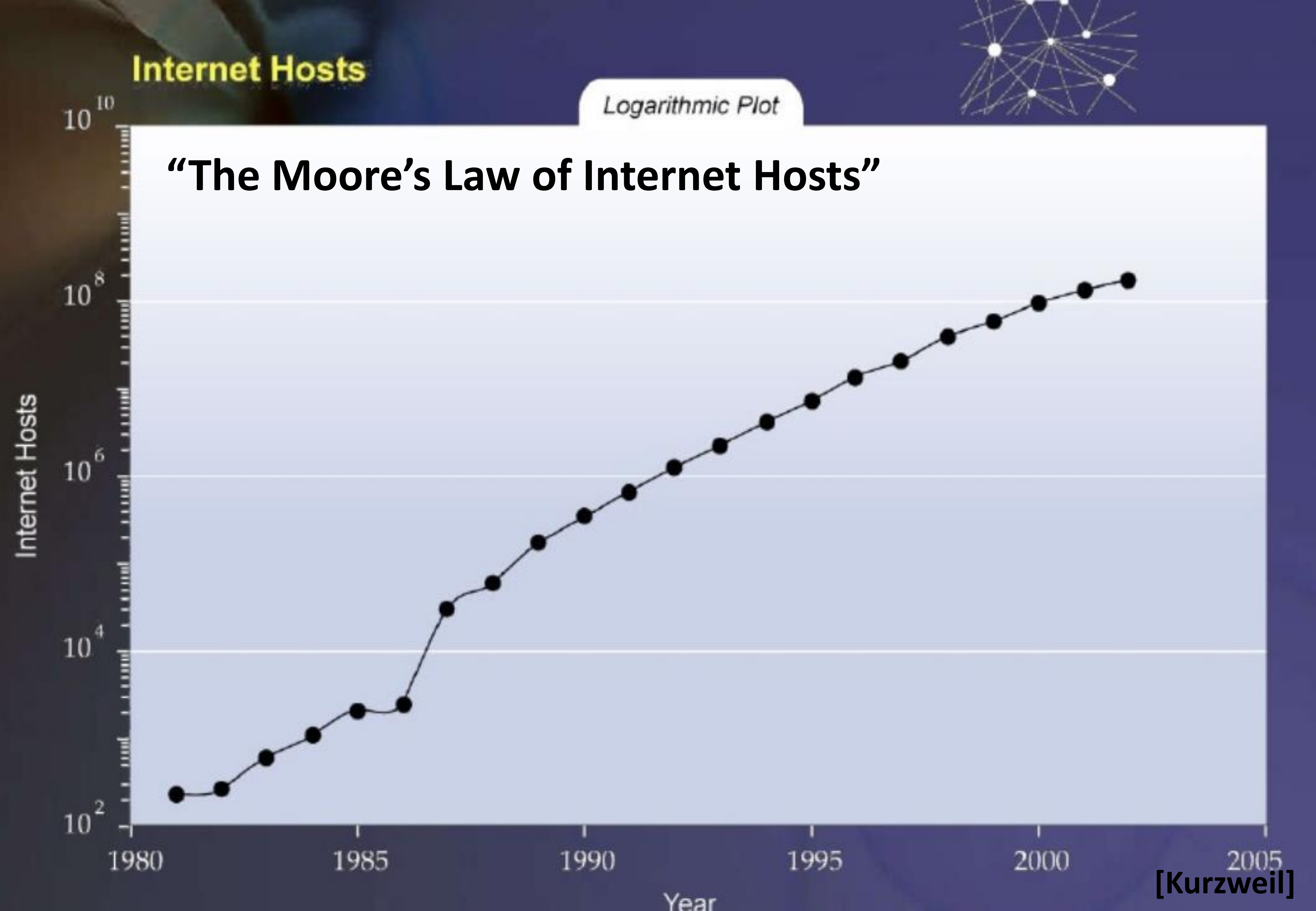
Driving Moore's Law with Python-Powered Machine Learning

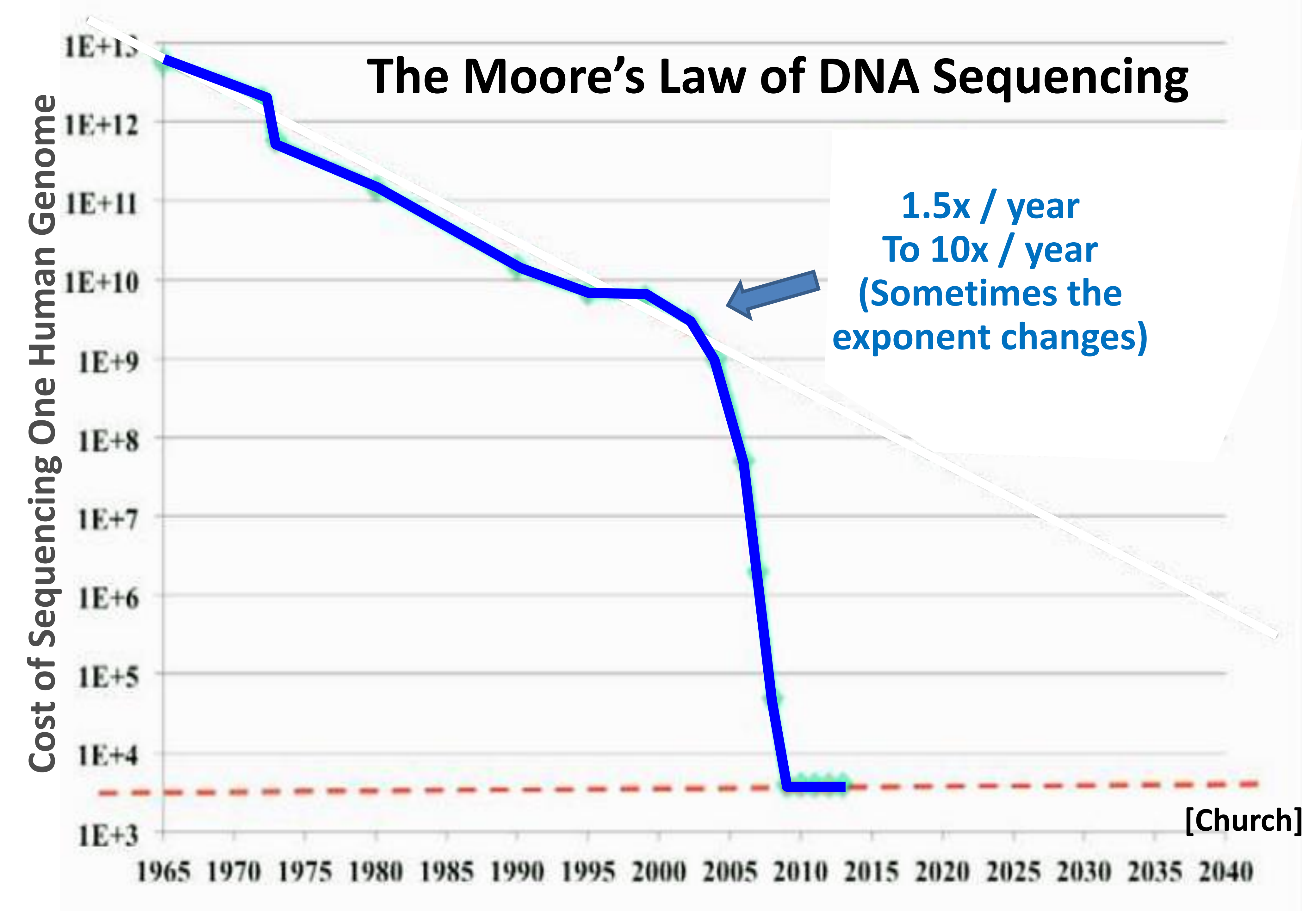
Trent McConaghy, PhD
Founder & CTO @ ADA | Solido | ascribe.io

Outline

- Moore’s Law
- Python, ML, & Moore’s Law

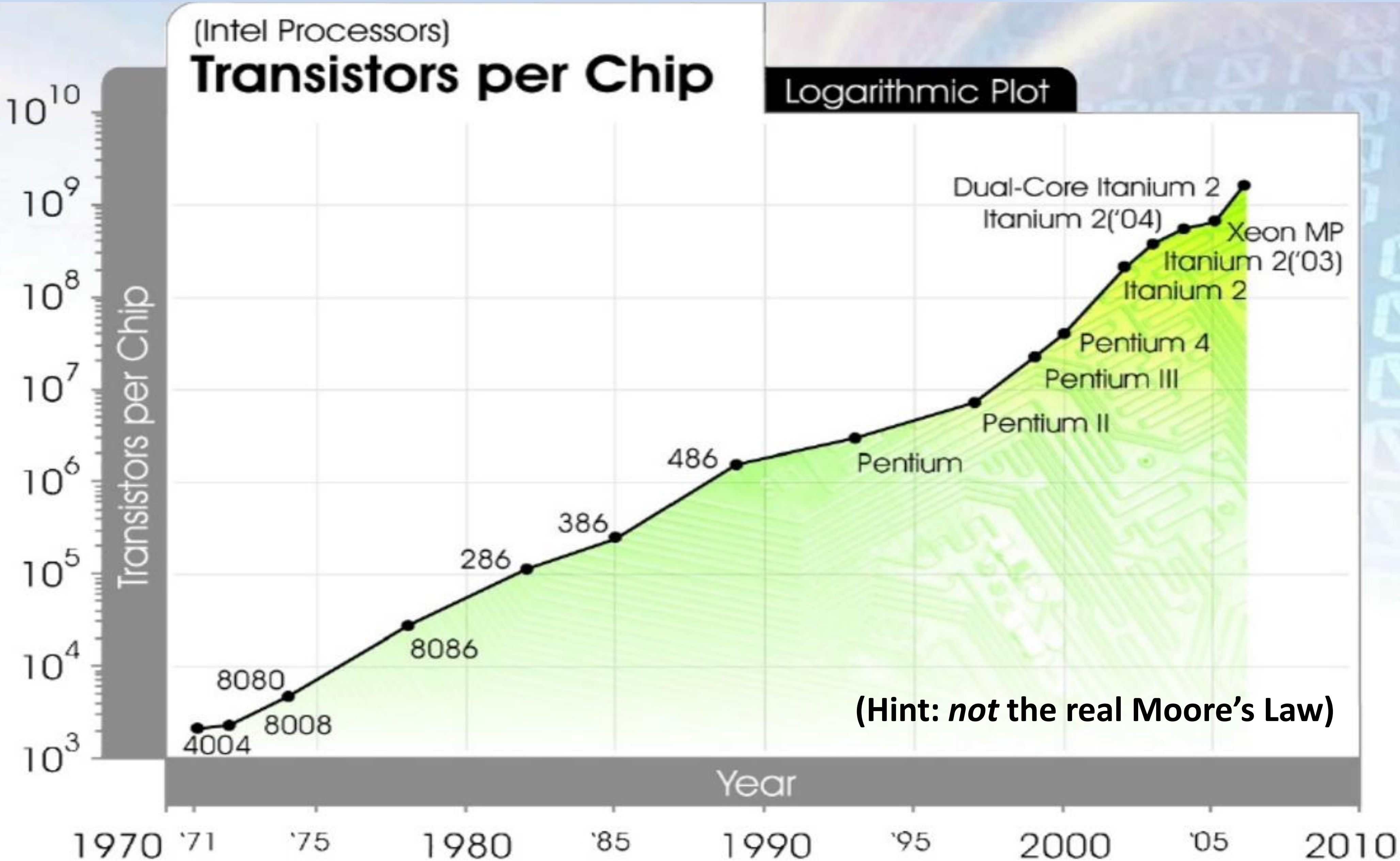

Resolution of
Noninvasive Brain Scanning
Logarithmic Plot
"The Moore's Law of Brain Scanning"
0.1
1
10
Resolution (mm)
1970
1975
1980
1985
1990
1995
2000
Year
[Kurzweil]


Internet Hosts
Logarithmic Plot
"The Moore's Law of Internet Hosts"
Internet Hosts
10^10
10^8
10^6
10^4
10^2
1980
1985
1990
1995
2000
2005
Year
[Kurzweil]



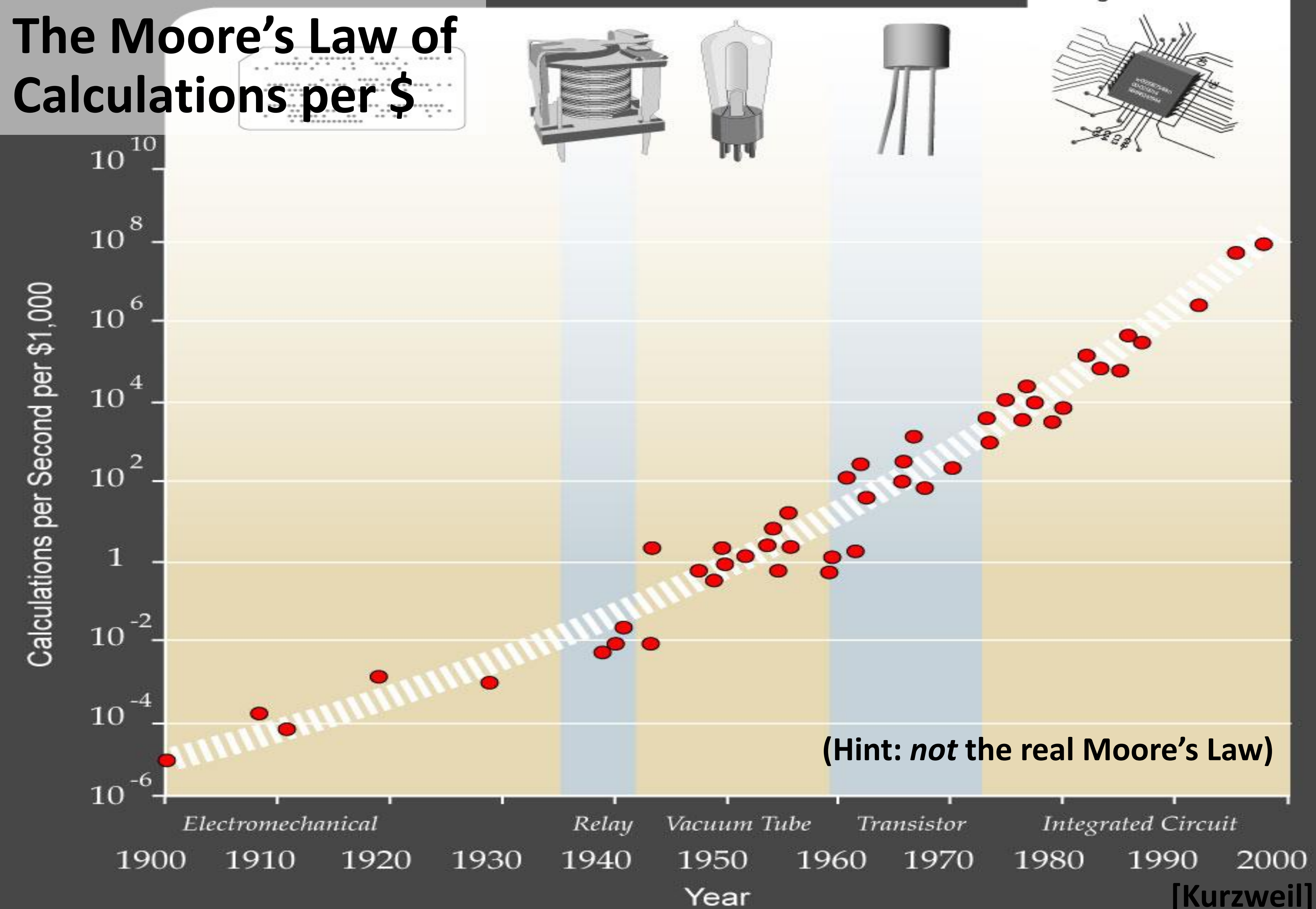
The Moore's Law of DNA Sequencing
Cost of Sequencing One Human Genome
1E+13
1E+12
1E+11
1E+10
1E+9
1E+8
1E+7
1E+6
1E+5
1E+4
1E+3
1965 1970 1975 1980 1985 1990 1995 2000 2005 2010 2015 2020 2025 2030 2035 2040
1.5x / year
To 10x / year
(Sometimes the
exponent changes)
[Church]

Will the Real Moore's Law Please Stand Up? (Please stand up)



[Kurzweil]

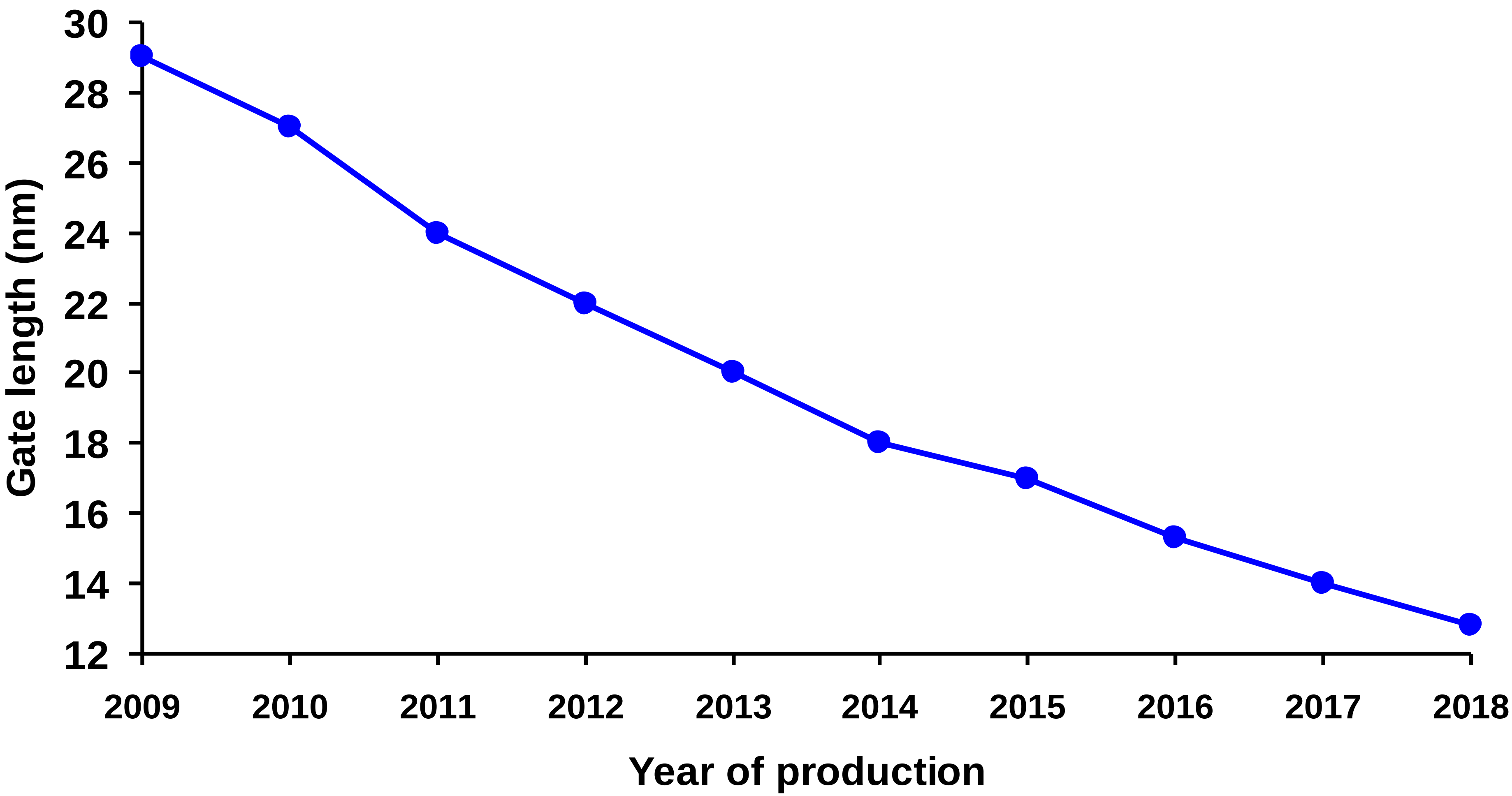
The Moore's Law of Calculations per $



[Kurzweil]

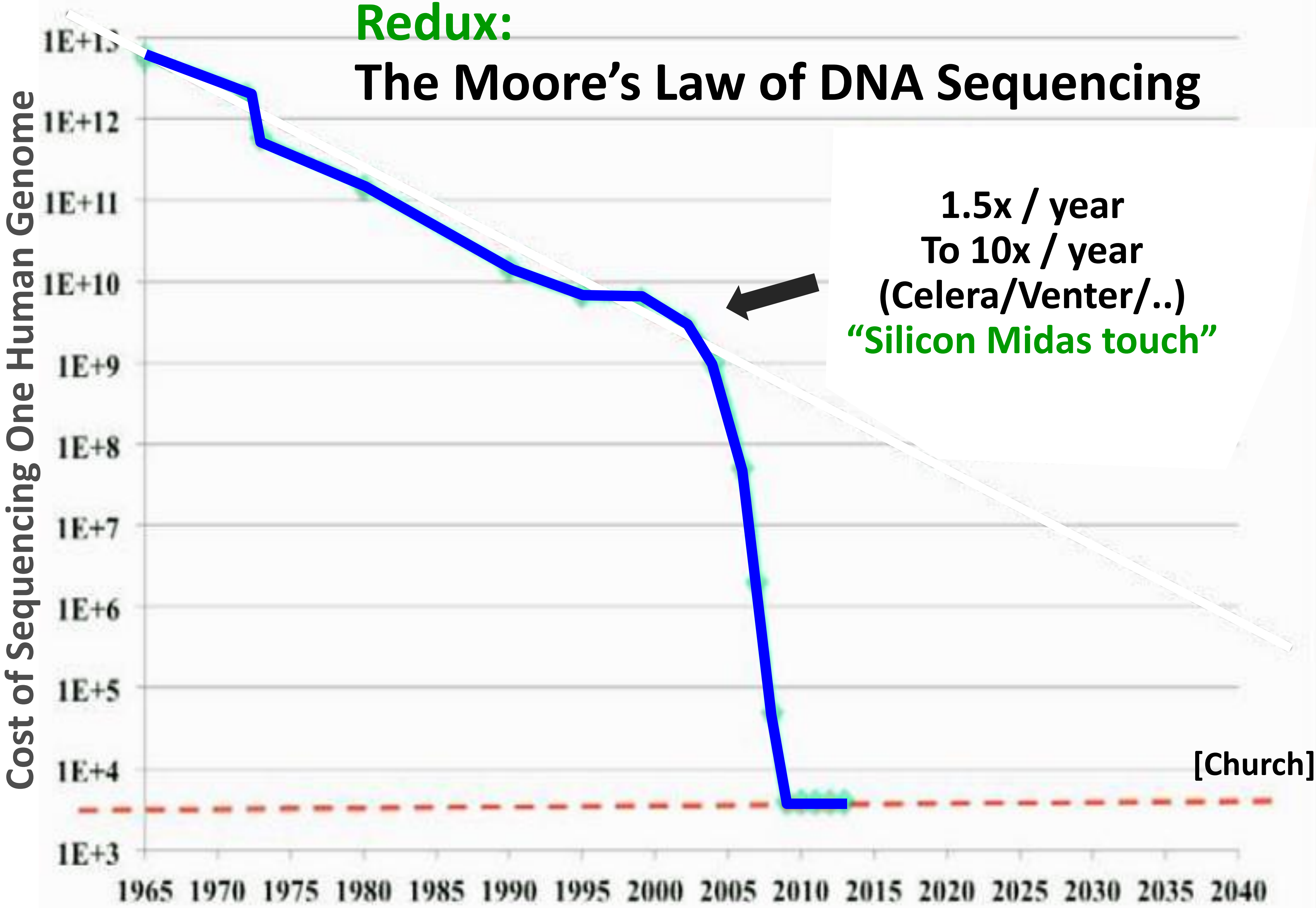
The *Actual* Moore's Law

(About *transistor size*.)



[International Technology Roadmap for Semiconductors, 2011]

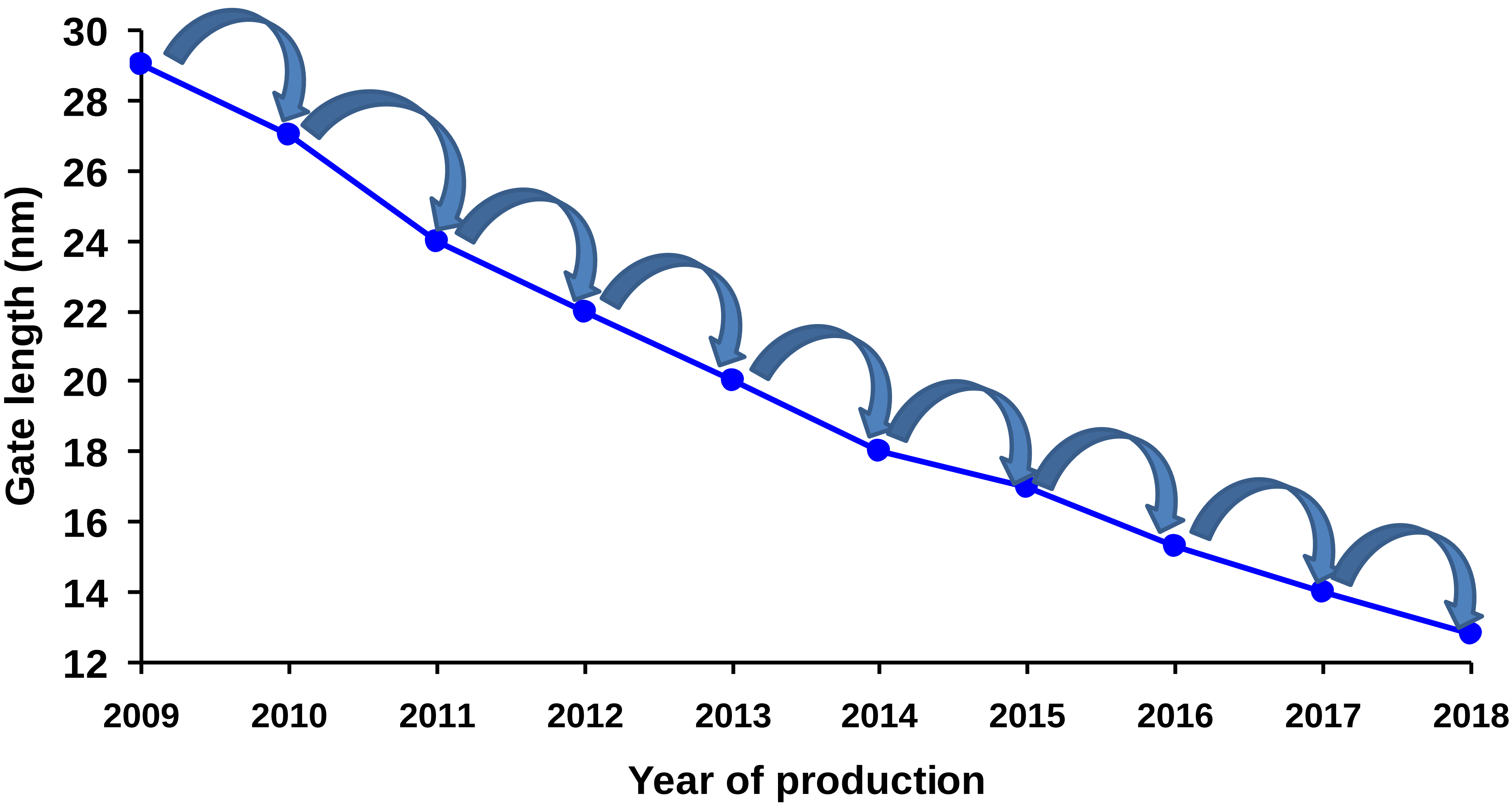
Redux:
The Moore's Law of DNA Sequencing



Moore's Law: *How?*

A: Silicon Midas touch *applied to itself*

One generation of machines, to design the next generation.
The ultimate bootstrap!



Moore's Law is a Bull. Riding It Enables…

Content

Cloud Computing

CITRIX
amazon web services
rackspace HOSTING
vmware

Computing

Inphi Think fast.
NXP
Sidense

Communications

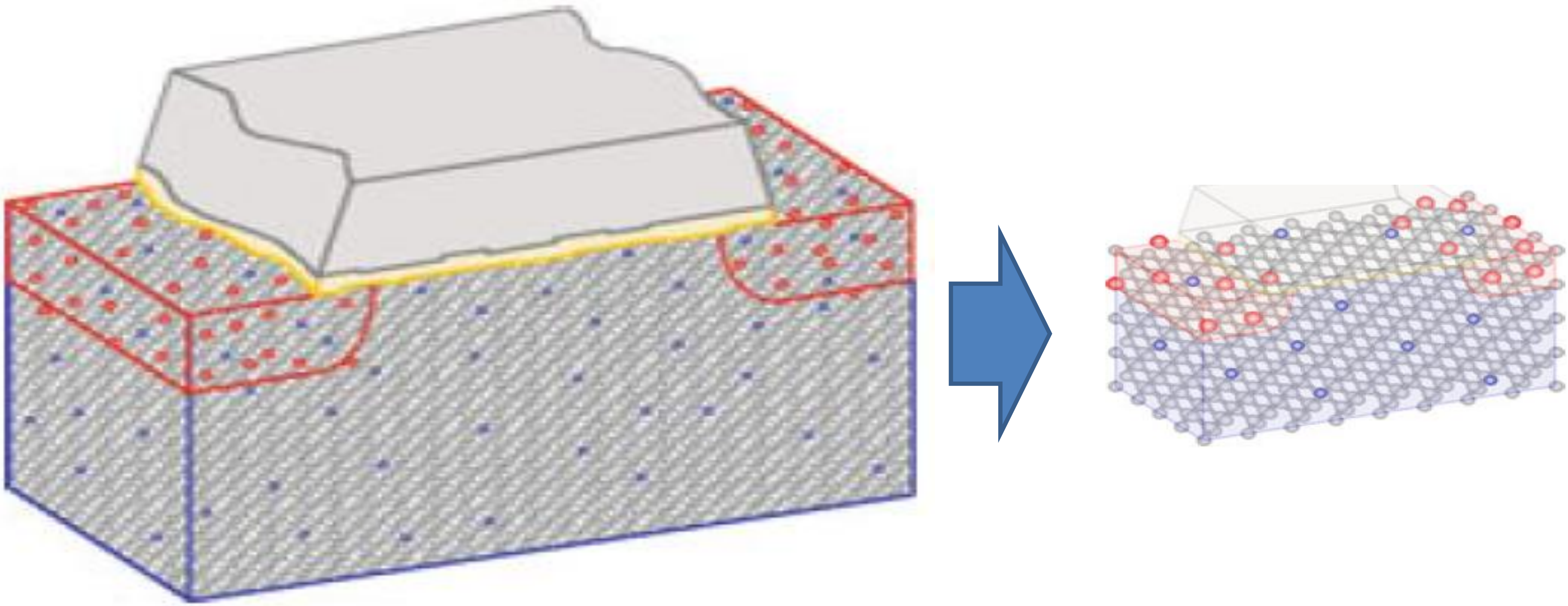
Avago TECHNOLOGIES
BROADCOM
HUAWEI

Consumer

QUALCOMM
SONY

A Challenge to Moore's Law: Variation Gone Wild

Transistors are shrinking
…but atoms aren't.

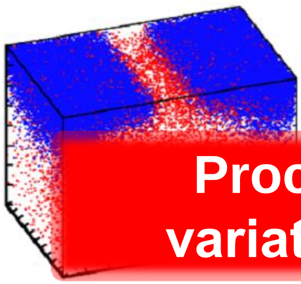


At <22 nm (now), *even one atom out of place* is trouble…

A. Asenov, Extreme Statistics in Nanoscale Memory Design, Springer, 2010

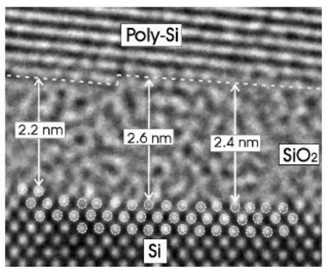
Variation = atoms out of place

…Propagating from devices to performance & yield



Process variation ↑

Random dopant effects

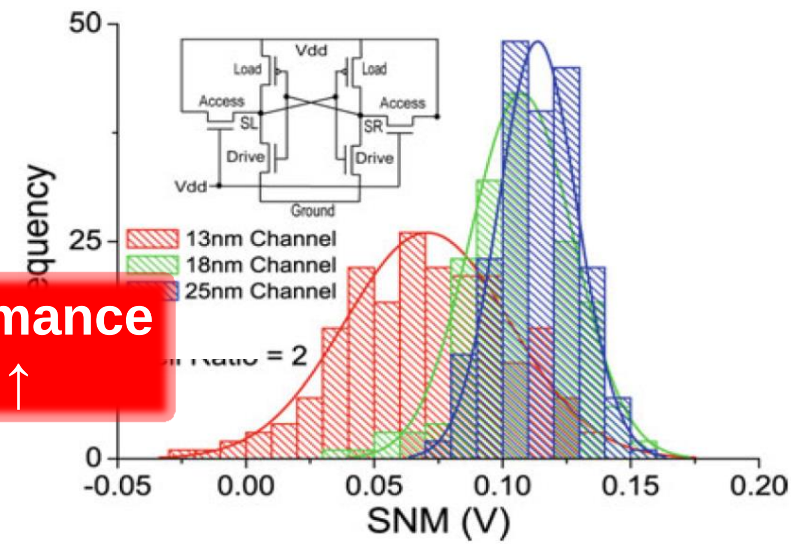


Oxide thickness

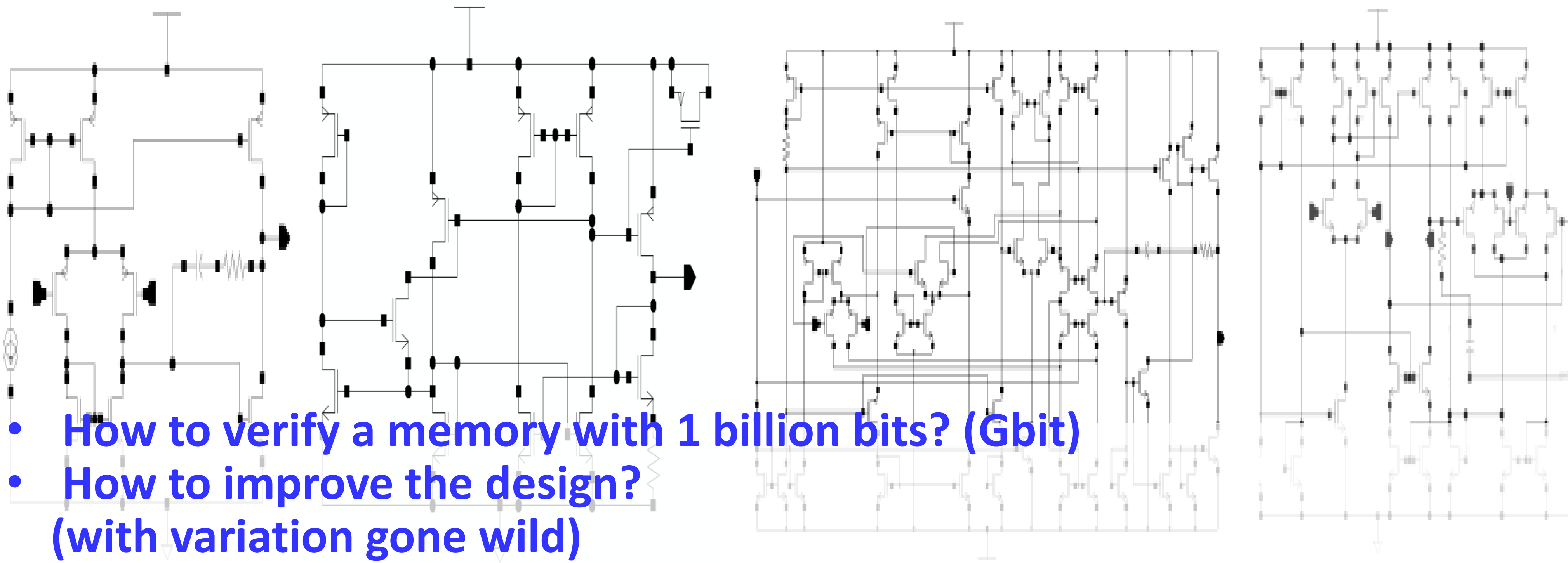
⋮

Device performance variation ↑

Circuit performance variation ↑



Variation-based Circuit Challenges



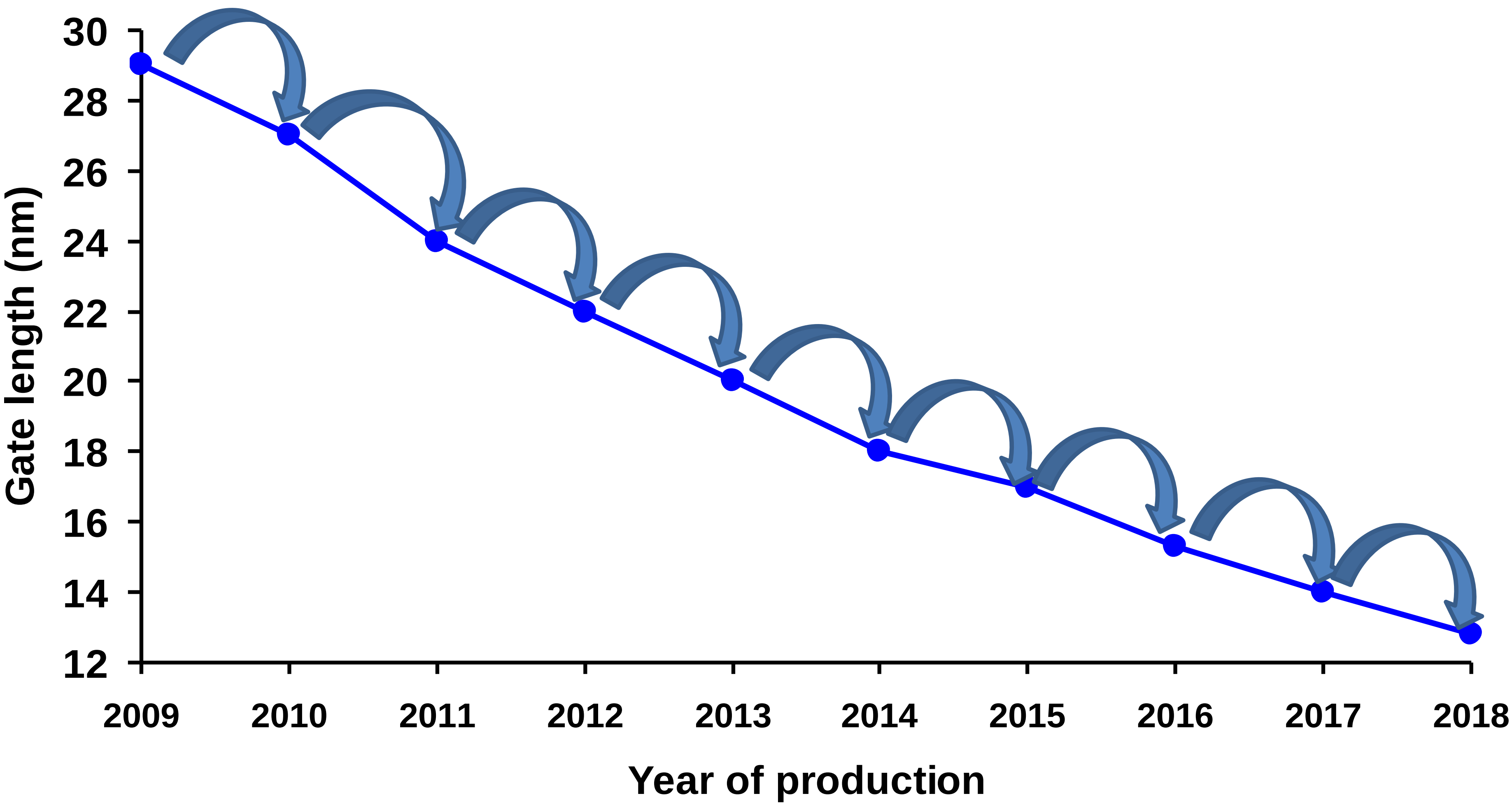
- How to verify a memory with 1 billion bits? (Gbit)
- How to improve the design?
  (with variation gone wild)

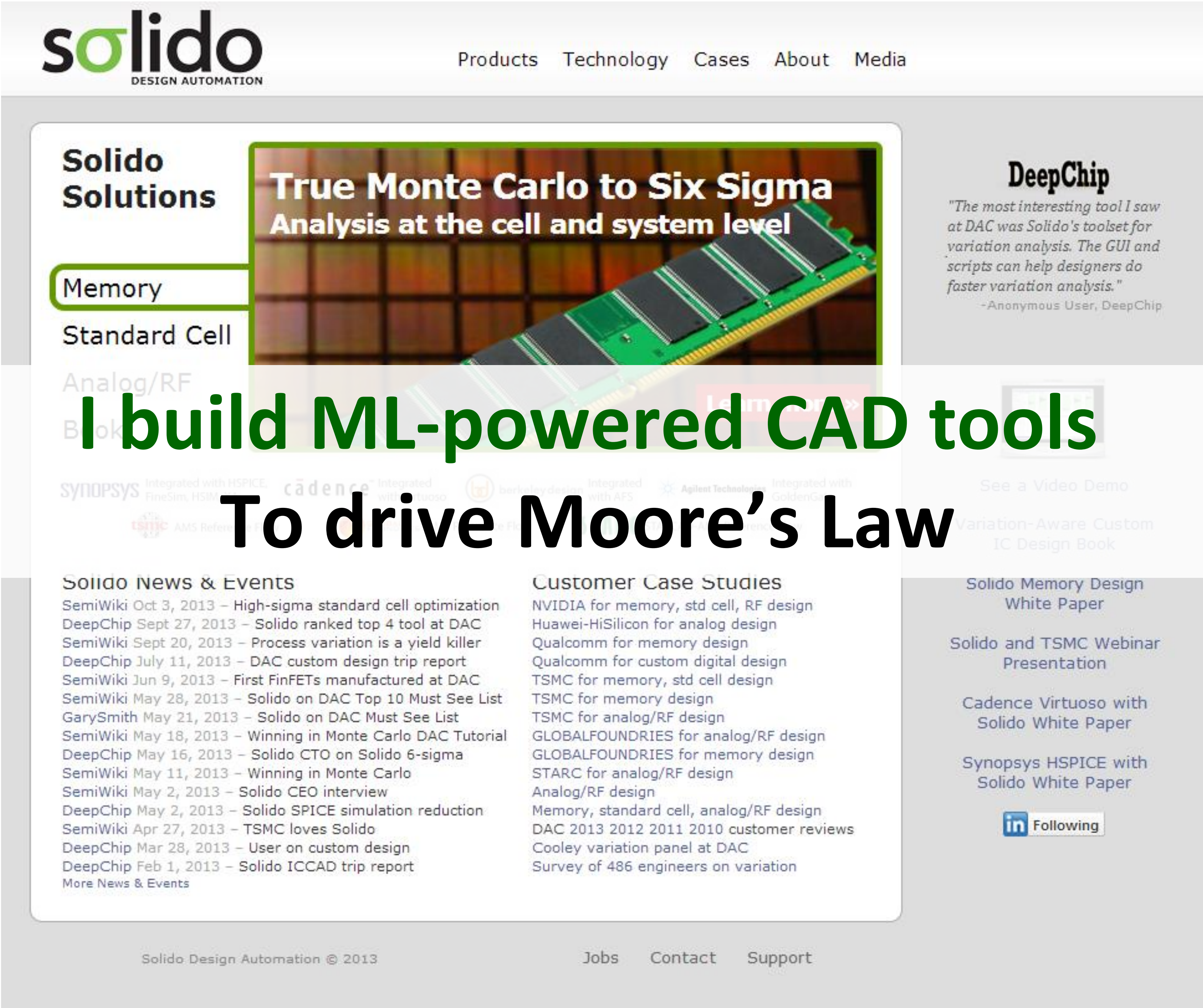
- How to verify a PLL with 3375 PVT corners?
- How to improve the design?
  (with variation gone wild)

- To get lower power, lower delay, lower area, all in less time?

Moore's Law incl. Variation: *How?*

Use ML to abstract away the variation from the perspective of the designer.



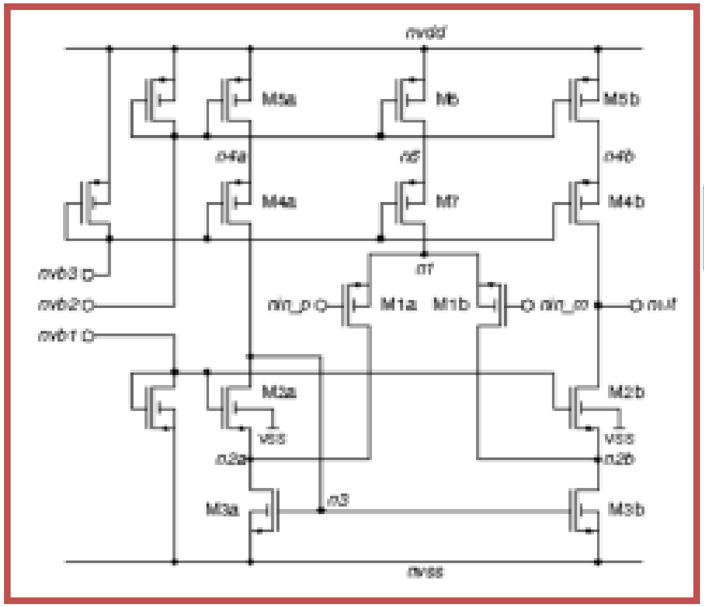


I build ML-powered CAD tools

To drive Moore's Law

Python, ML & Moore’s Law

Example: ML-based whitebox models of circuits



SPICE

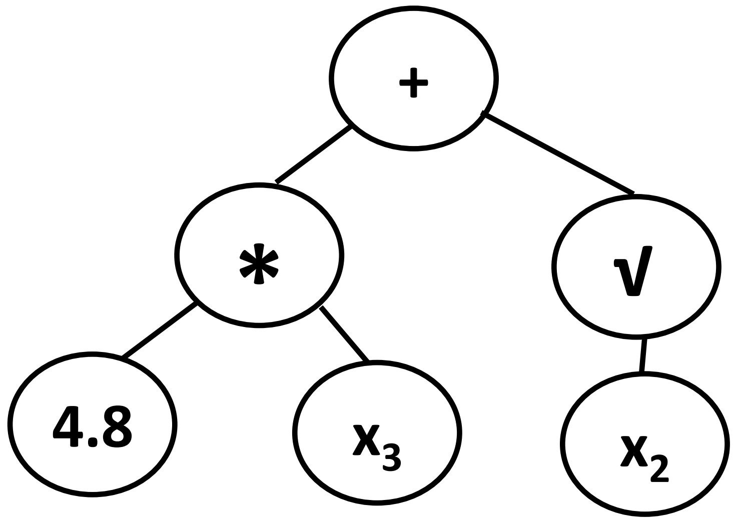
Symbolic Modeling

Perf.	Expression
A_{LF}	-10.3 + 7.08e-5 / id1 + 1.87 * ln(-1.95e+9 + 1.00e+10 / (vsg1*vsg3)+ 1.42e+9 *(vds2*vsd5) / (vsg1*vgs2*vsg5*id2))
f_u	10^(5.68 - 0.03 * vsg1 / vds2 - 55.43 * id1+ 5.63e-6 / id1)
PM	90.5 + 190.6 * id1 / vsg1 + 22.2 * id2 / vds2
V_{offset}	- 2.00e-3
SR_p	2.36e+7 + 1.95e+4 * id2 / id1 - 104.69 / id2 + 2.15e+9 * id2 + 4.63e+8 * id1
SR_n	- 5.72e+7 - 2.50e+11 * (id1*id2) / vgs2 + 5.53e+6 * vds2 / vgs2 + 109.72 / id1

Example: ML-based whitebox models of circuits
How: Genetic Programming

"A function is a *tree*"

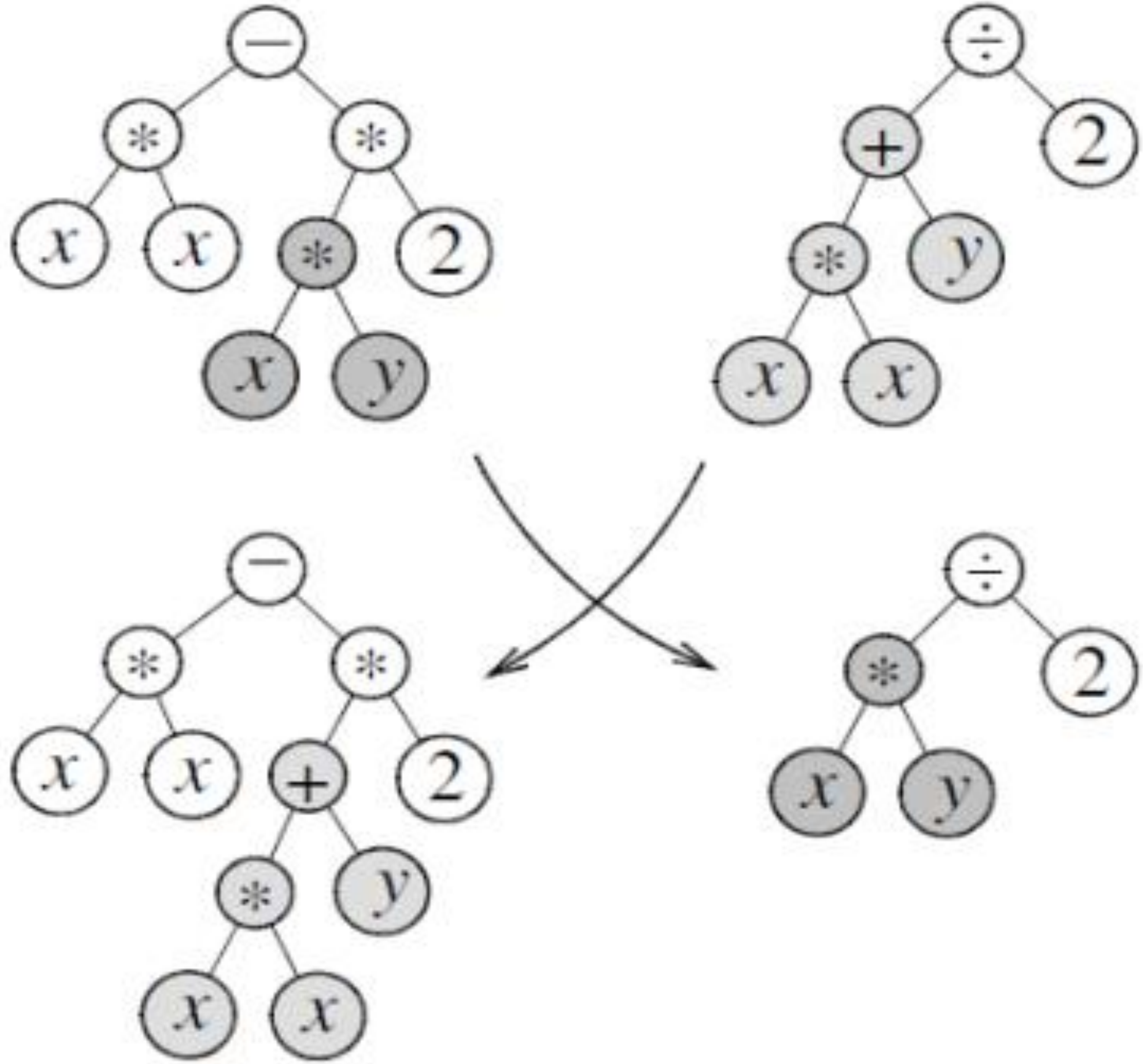
$$f(x) = 4.8 * x_3 + \sqrt{x_2}$$



Searches through the space of trees:

1. Initial random population; evaluate
2. Create children from parents via operators; evaluate
3. Select best; goto 2

Example: ML-based whitebox models of circuits
Crossover Operator in Genetic Programming



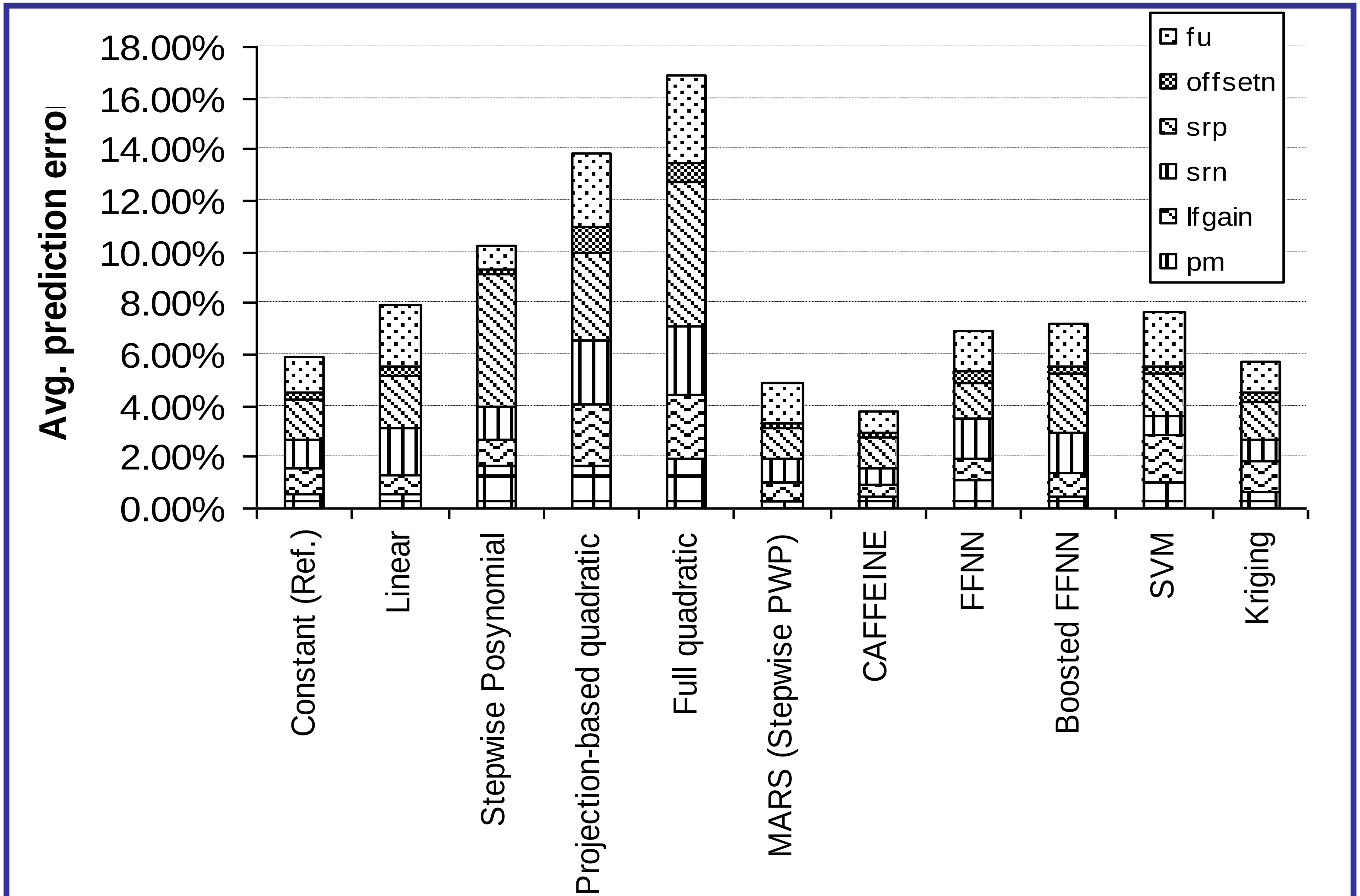
Example: ML-based whitebox models of circuits

Models with <10% error

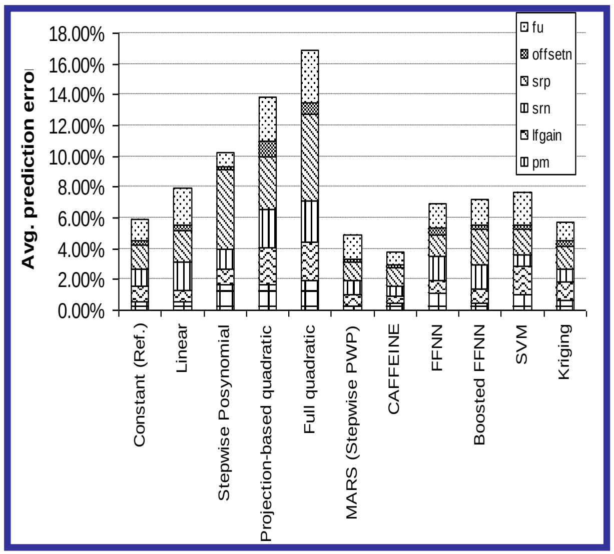
Perf.	Expression
A_{LF}	-10.3 + 7.08e-5 / id1 + 1.87 * ln(-1.95e+9 + 1.00e+10 / (vsg1*vsg3) + 1.42e+9 *(vds2*vsd5) / (vsg1*vgs2*vsg5*id2))
f_u	10^(5.68 - 0.03 * vsg1 / vds2 - 55.43 * id1+ 5.63e-6 / id1)
PM	90.5 + 190.6 * id1 / vsg1 + 22.2 * id2 / vds2
v_{offset}	- 2.00e-3
SR_p	2.36e+7 + 1.95e+4 * id2 / id1 - 104.69 / id2 + 2.15e+9 * id2 + 4.63e+8 * id1
SR_n	- 5.72e+7 - 2.50e+11 * (id1*id2) / vgs2 + 5.53e+6 * vds2 / vgs2 + 109.72 / id1

Example: ML-based whitebox models of circuits
Prediction Performance

Summary: Lower prediction error than FFNNs, Boosted FFNNs, SVMs, GPMs, ..



Example: ML-based whitebox models of circuits
The Stack



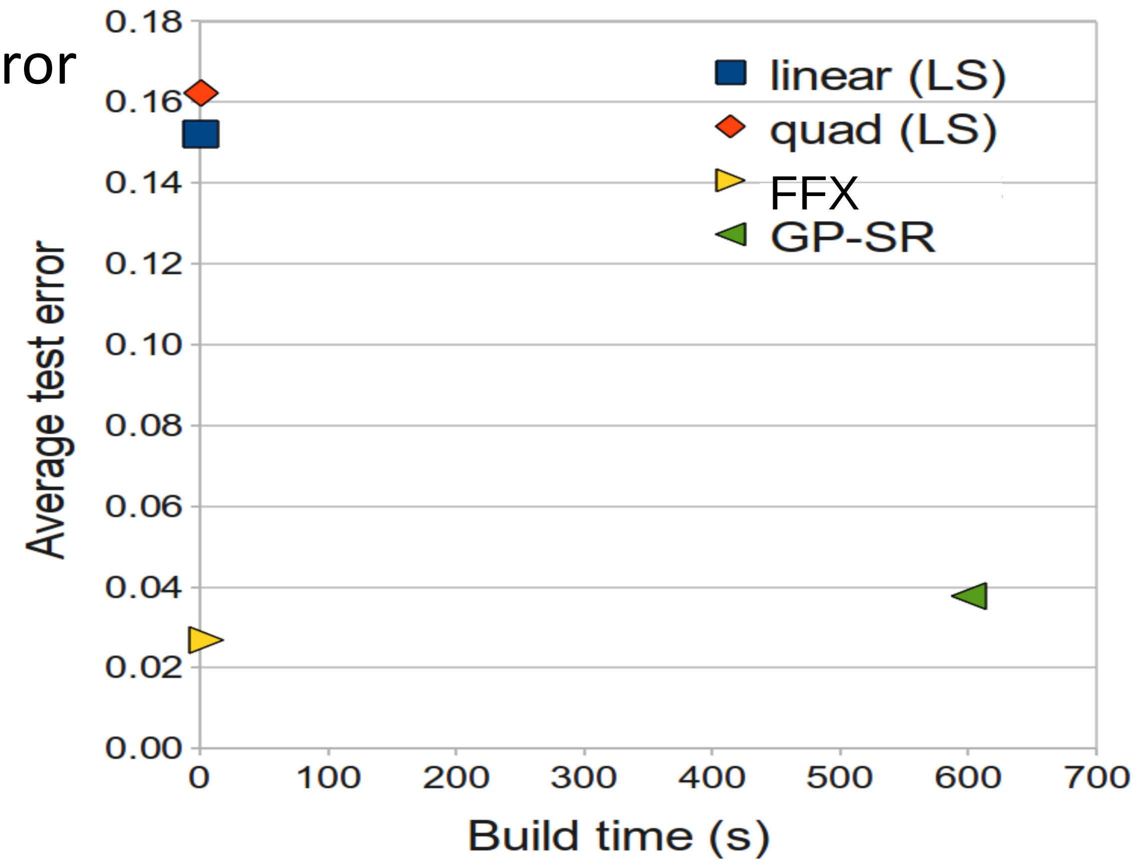
- 100% Python
  - Python 2.7, numpy, scipy
  - Custom ML algorithm
    - grammar-constrained genetic programming
    - function-grammar
- 3rd party circuit simulator

Example: ML-based whitebox models redux (FFX)

Problem: Scales poorly past >20 variables

Algorithm:

1. Explode # basis functions (e.g. 13 → 100K)
2. Pathwise learning on elastic net formulation (**BHALR**), track # variables vs. train error
3. Nondominated filter on test error

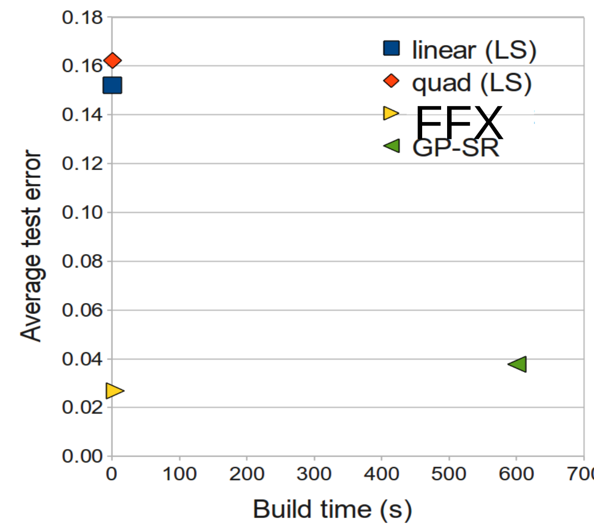


Result: scalability & speed ↑

- 10K+ input variables
- 100 – 100K+ training points

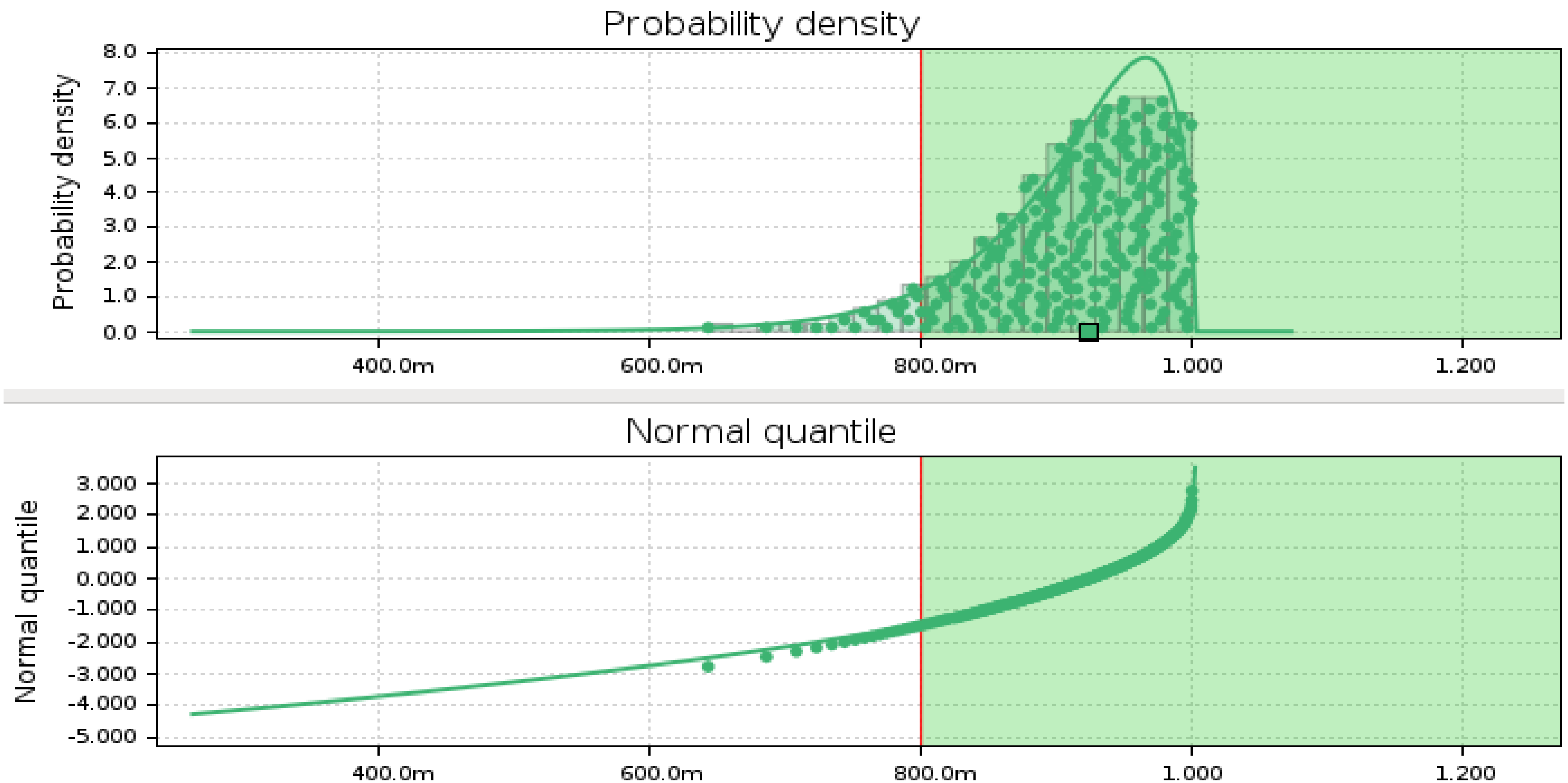
Example: ML-based whitebox models redux (FFX)
The stack

- 100% Python
  - Python 2.7, numpy
  - Scikit-learn
    - Coordinate descent pathwise learning
  - Custom ML: FFX
    - Explode # basis functions
    - Nondominated filtering



- General enough for other domains
- Extends to classification too
- Open source at trent.st/ffx

Example: Density Estimation with Sane Extrapolation

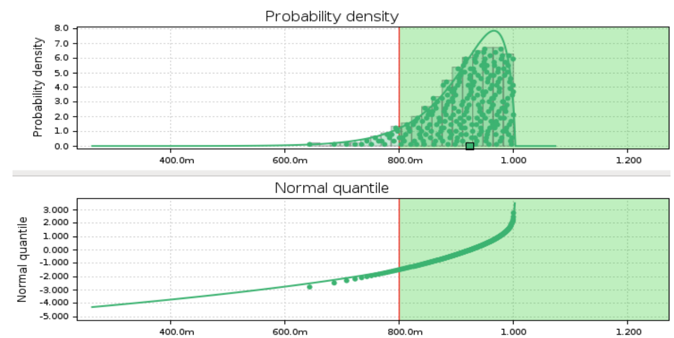


Algorithm:

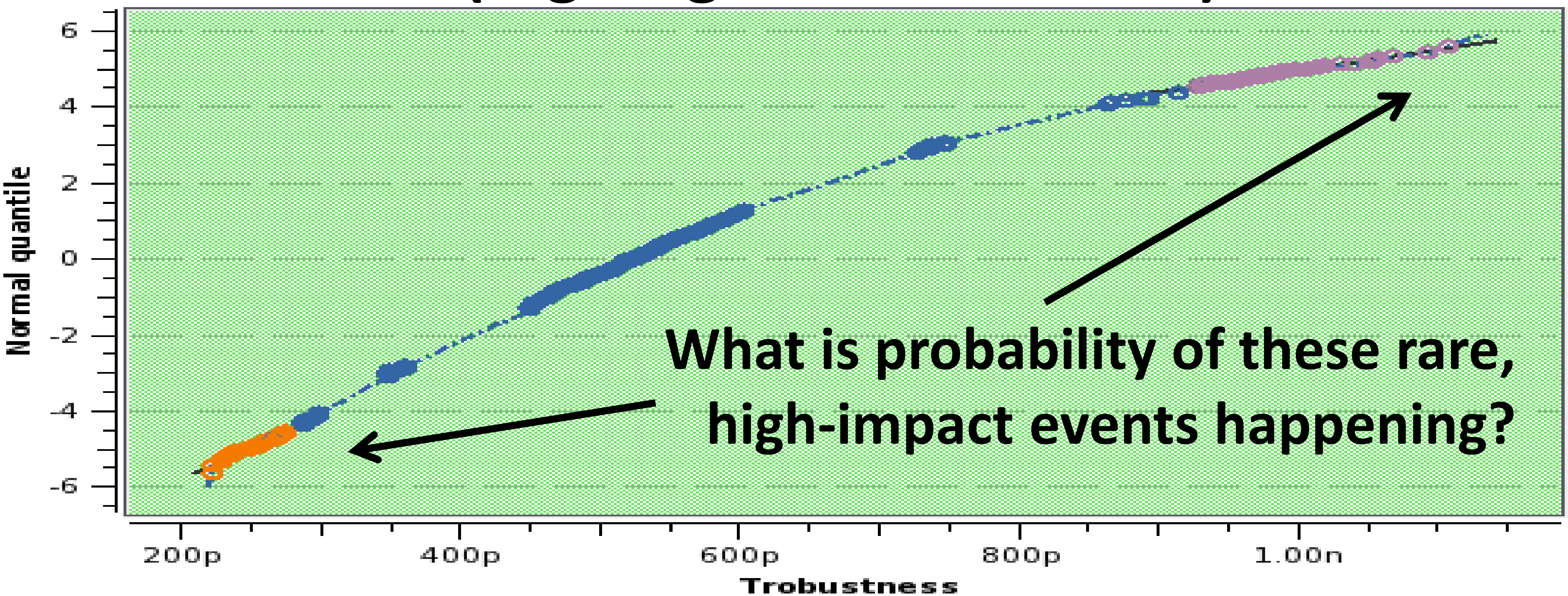
1. Build many different density models: Gaussian, mixture of 2-4 Gaussians, lognormal, uniform, Rayleigh, KDE, and more.
2. Pick model with the best fit in NQ space *(not MLE).*

Example: Density Estimation with Sane Extrapolation: The Stack

- 100% Python
  - Python 2.7, numpy
  - Scipy – kde, optimize (BFGS), specific distributions
  - Custom ML algorithm
    - Conversion to/from NQ space
    - Special-case distributions (e.g. uniform, spike)
- 3rd party circuit simulator



Example: ML-driven Rare Event Estimation (High Sigma Monte Carlo)



Problem: Brute force takes 2 months on 100 cores

Algorithm:

1. Active learning on 10K+ dimensions to learn X-> y
2. Draw & rank 10G pts (≈scale of Google search)
3. Simulate from highest-rank first (≈ top 10 search results)

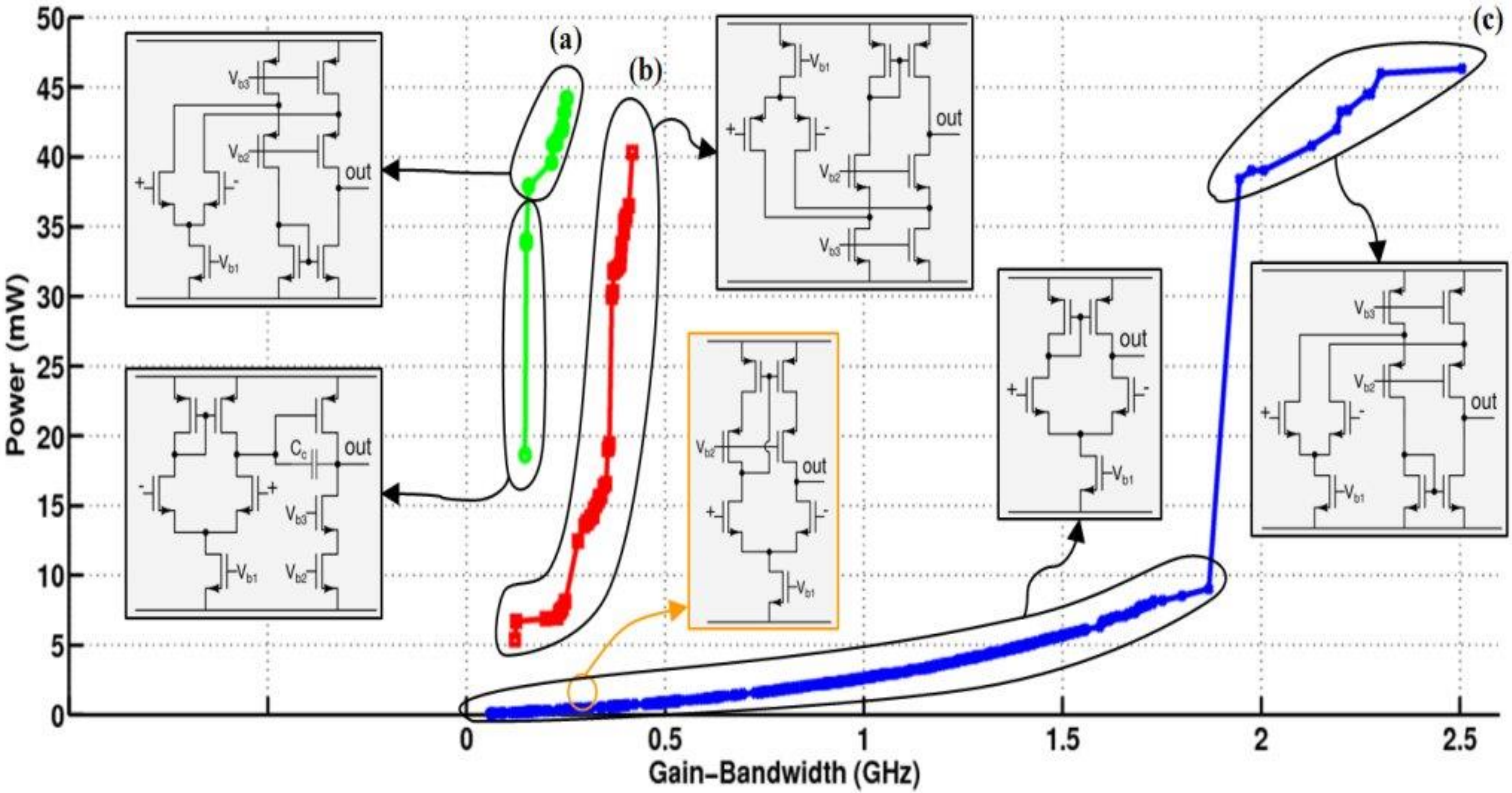
Result: 20 min on 10 cores

Example: ML-driven Rare Event Estimation (High Sigma Monte Carlo): The Stack

- 99% Python
  - Python 2.7, numpy, scipy
  - scikit-learn pathwise learning
  - Custom high-dimensional regression (FFX)
  - Qt4, Chaco
- 1% C
  - Random number generation - Mersenne Twister. (incidentally, traditional LCG is inadequate because period is too small.)
  - Simulate regressor on each of 10G points
- 3rd party circuit simulator, env’t

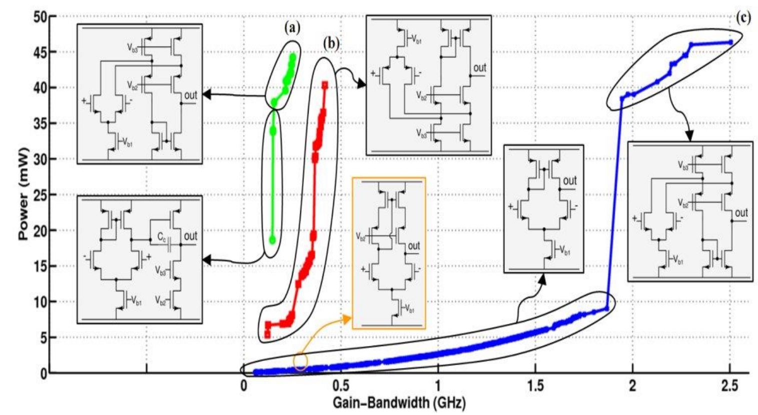
Example: ML to synthesize analog circuit topologies

How: Design a language for circuit topologies; populate it; then do grammar-constrained multi-obj. tree search



Example: ML to synthesize analog circuit topologies: The Stack

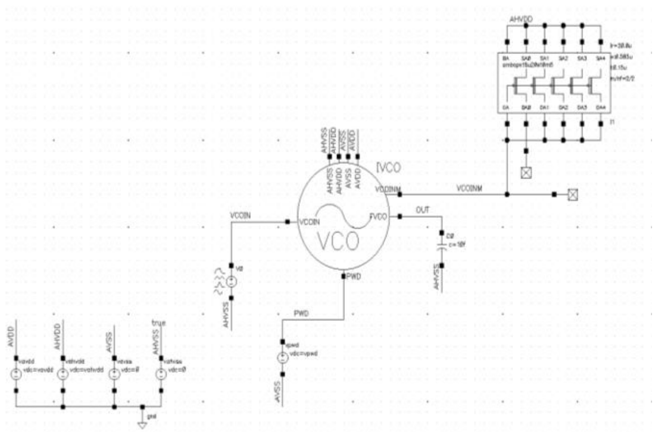
- 100% Python
  - Python 2.7, numpy, scipy
  - Custom ML algorithms
    - grammar-constrained genetic programming
    - circuit grammar
    - derivative-free optimizer
    - high-resolution interpolator
- 3rd party circuit simulator

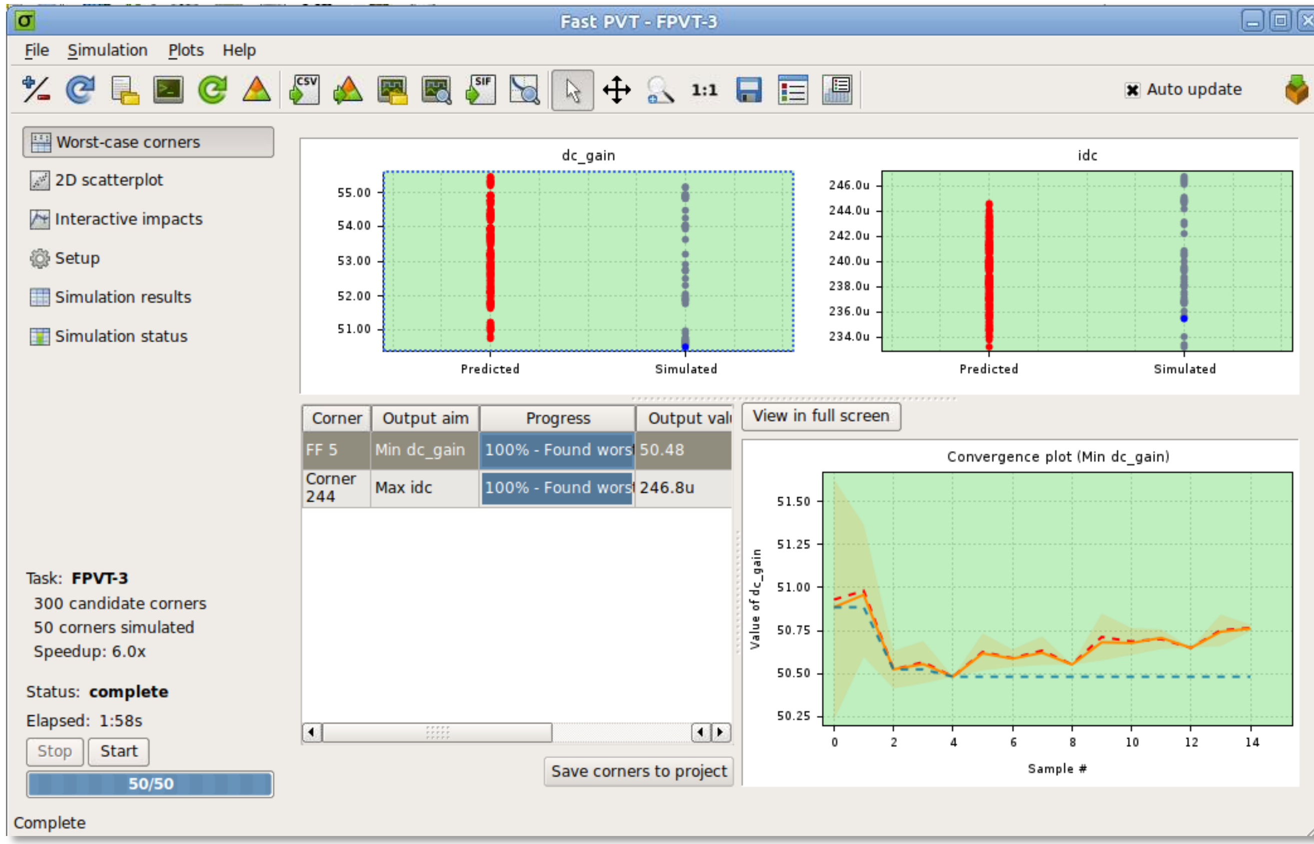


- General enough for other domains
- Open source at trent.st/mojito

Example: ML-driven Corners Analysis(Fast PVT)

- TSMC 28nm, VCO of a PLL
- Specs: 48.3 < duty cycle < 51.7 %, 3 < Gain < 4.4GHz/V
- ***Traditional: 3375*** **PVT corners to simulate** (temp, voltage1, ..)
- **With ML: 275 corners to simulate, as thorough as before**



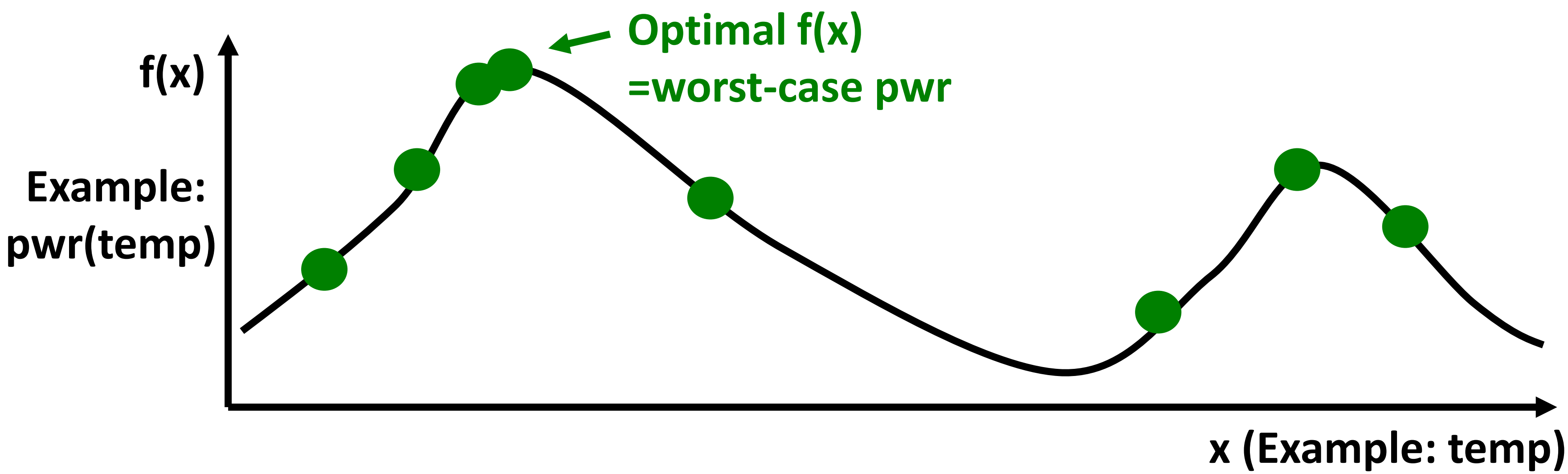


Example: ML-driven Corners Analysis

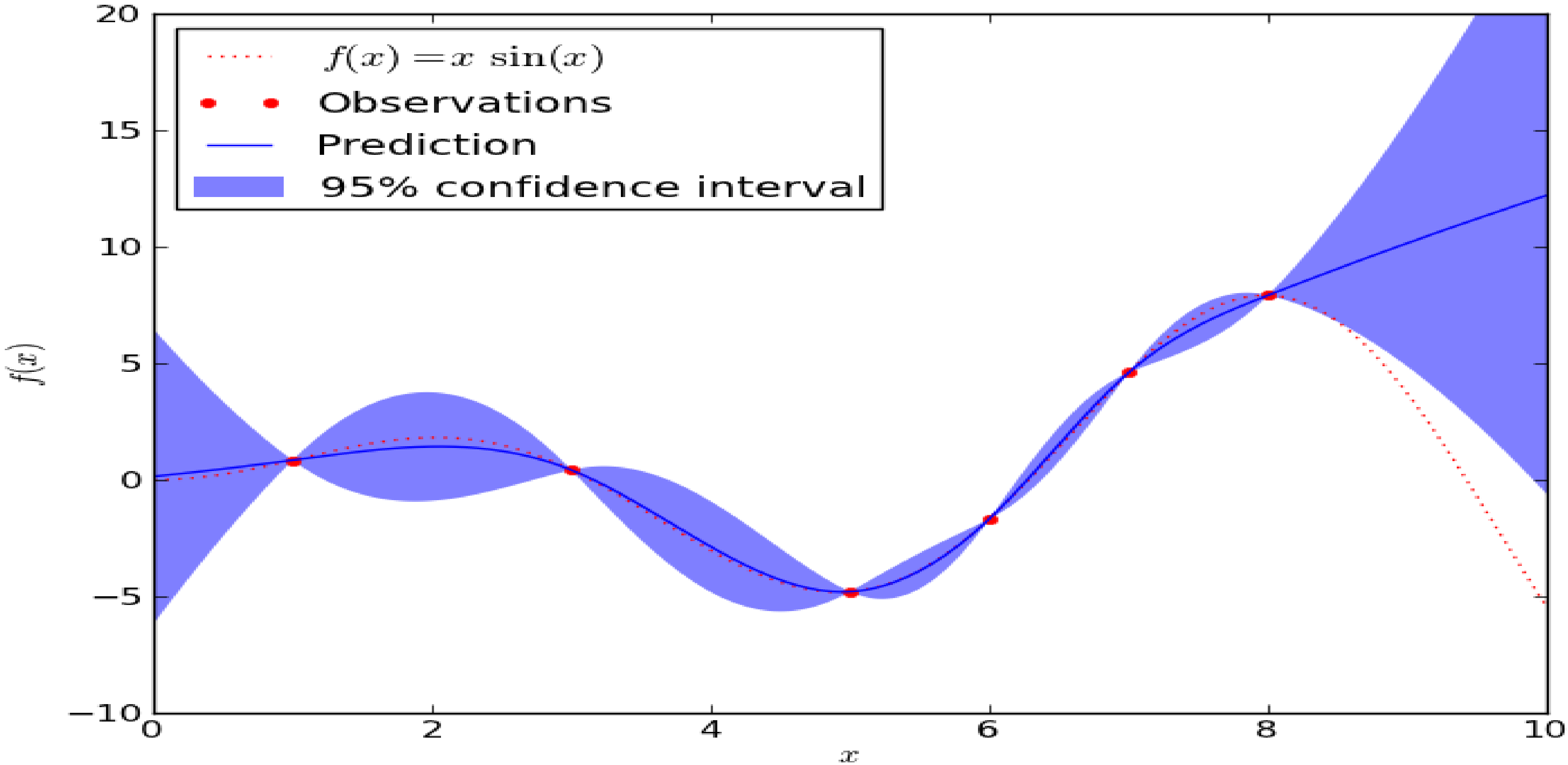
Cast PVT verification as a global optimization problem:

- **Search through space of “corners” - x**
- **Minimize / maximize simulated output value f(x)**

Then, solve the optimization problem reliably.



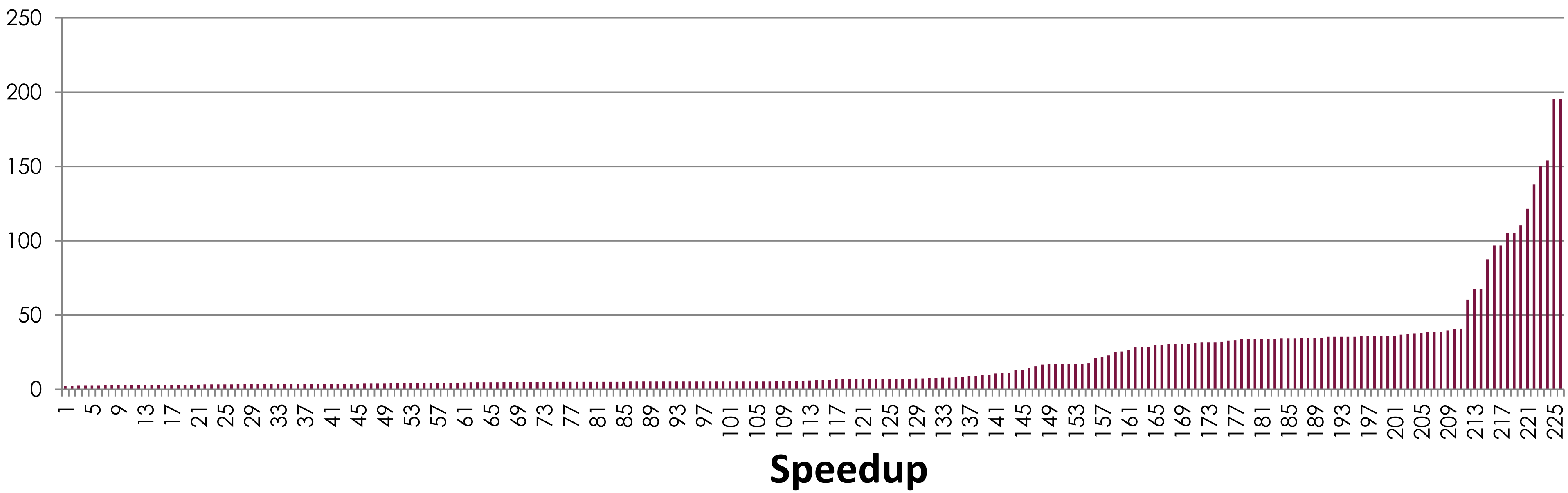
ML-driven Corners Analysis: underlying Model



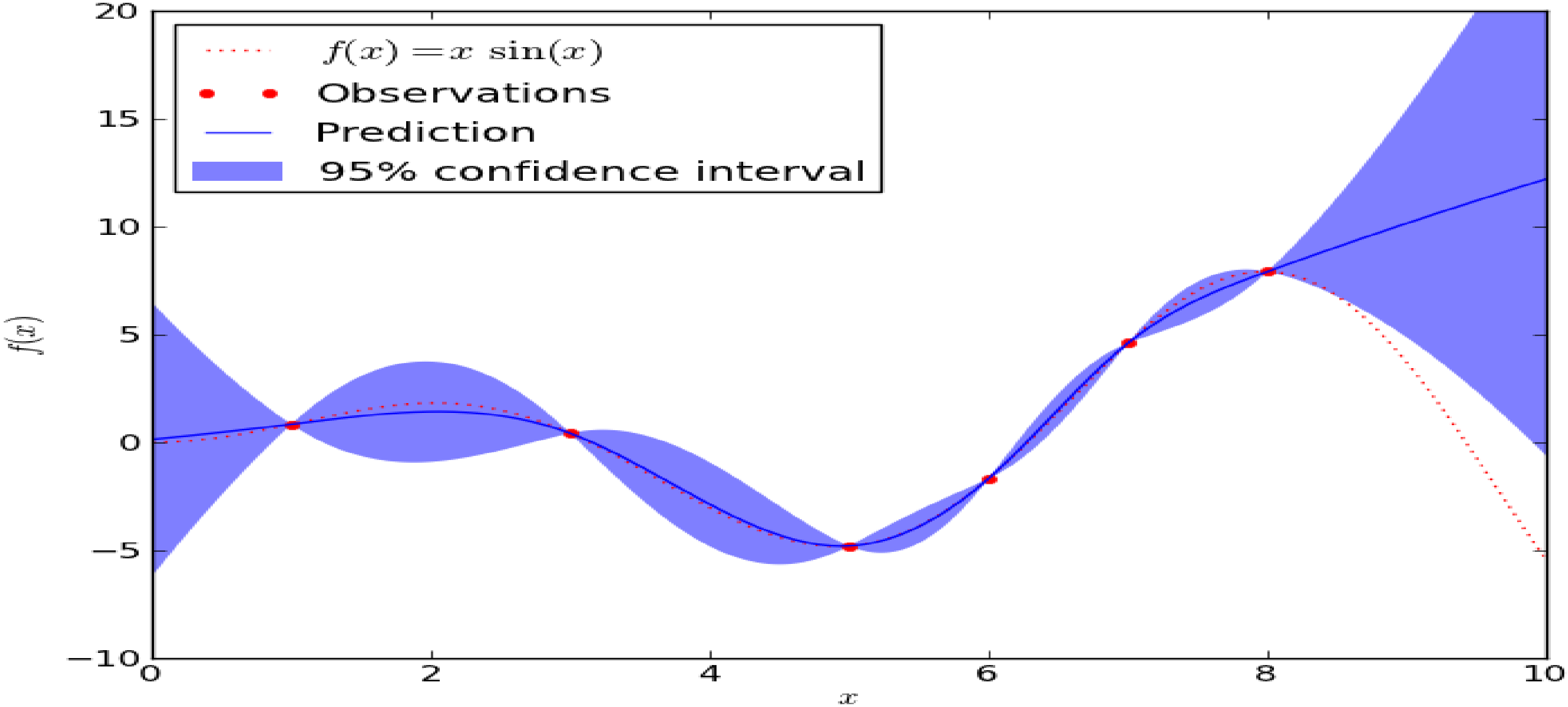
- Typically a Gaussian Process Model (GPM)
  - Natural interpolator
  - Convenient confidence intervals
  - Well-behaved, no crazy extrapolation (usually)

ML-driven Corners Analysis
Benchmarks on 226 Circuit PVT Verification Problems

- 226 test cases in benchmark suite:
  - From Solido customers, in-house realistic cases, and in-house corner cases targeting challenging problems
  - Many contain complex interactions, non-linearities, discontinuities, etc.
- **226/226 (100%) of cases find true optimum**
- Speedup **2.34X to 226X**
- **Median speedup is 22X**

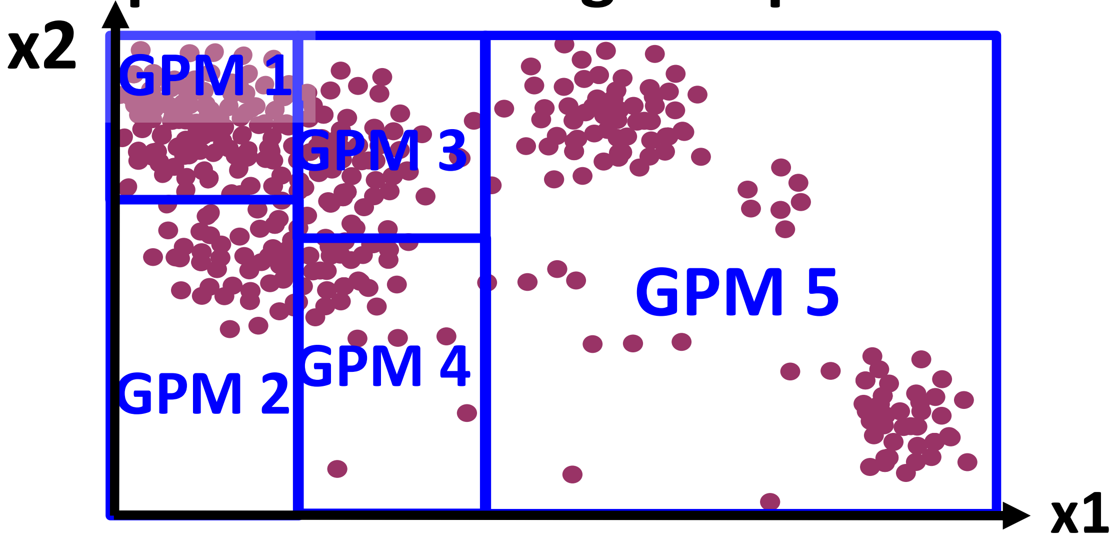


ML-driven Corners Analysis: Scalability Challenge



- **Problem: GPM training is $O(N^3)$ on # Training Samples**
- **Becomes *very* unhappy when >1000 samples**
- This happens for circuit verification problems with larger # dimensions and highly nonlinear circuit
- First solution: just cut loose and sim all
- Is there a better way?

ML-driven Corners Analysis:
Divide-and-Conquer on Training Samples



- New model is a *set* of Gaussian Process Models (GPM)
- One GPM for each region of input x space
- Regions are automatically determined at build time
  - Via classic CART learning
  - Stop at a leaf when <700 samples
- Build a GPM on each leaf's samples (and *k* neighbors)
  - Each GPM is O(1) on # training samples because N=const
  - CART learning is O(N log N) on # samp with tiny constant

ML-driven Corners Analysis:
Benchmarking: GPM vs Divide-and-Conquer GPM

				GPM			Divide-and-conquer GPM		
Problem	# vars	# train pts	# test pts	Build Time (s)	Test Time (s)	Error	Build Time	Test Time	Error
Low-dimensional									
opamp-pvt-bandwidth	10	4425	1475	667.4	91.1	0.044	**55.6**	**7.4**	**0.006**
opamp-pvt-dc_gain	10	4425	1475	741.9	91.5	**0.001**	**57.9**	**8.5**	**0.003**
opamp-pvt-gain_margin	10	4425	1475	319.9	92.2	0.313	**59.6**	**8.2**	**0.168**
opamp-pvt-gbw	10	4425	1475	845.7	92.8	**0.010**	**62.4**	**8.8**	**0.008**
opamp-pvt-idc	10	4425	1475	775.2	91.7	**0.000**	**41.2**	**8.2**	**0.000**
opamp-pvt-phase_margin	10	4425	1475	268.2	90.9	**0.149**	**49.8**	**6.6**	**0.155**
High-dimensional									
senseamp_pwr	125	3750	1250	failed	failed	failed	**165.8**	**37.9**	**4.139**
opamp_AV	215	600	200	38.3	18.2	**2.933**	**23.3**	**9.8**	**3.628**
opamp_SR	215	600	200	34.8	18.2	**2.604**	**37.3**	**9.5**	**2.515**
compar_bw	639	1502	500	246.2	56.9	**16.010**	**73.7**	**23.0**	**16.458**
opamp_PM	215	600	200	63.9	18.3	3.678	**26.7**	**9.4**	**2.441**
opamp_BW	215	600	200	34.9	18.3	**1.800**	**31.6**	**9.6**	**2.084**
mem	385	7500	2500	failed	failed	failed	**422.4**	**78.3**	**0.480**
senseamp_delay	125	3750	1250	failed	failed	failed	**286.0**	**38.4**	**5.135**

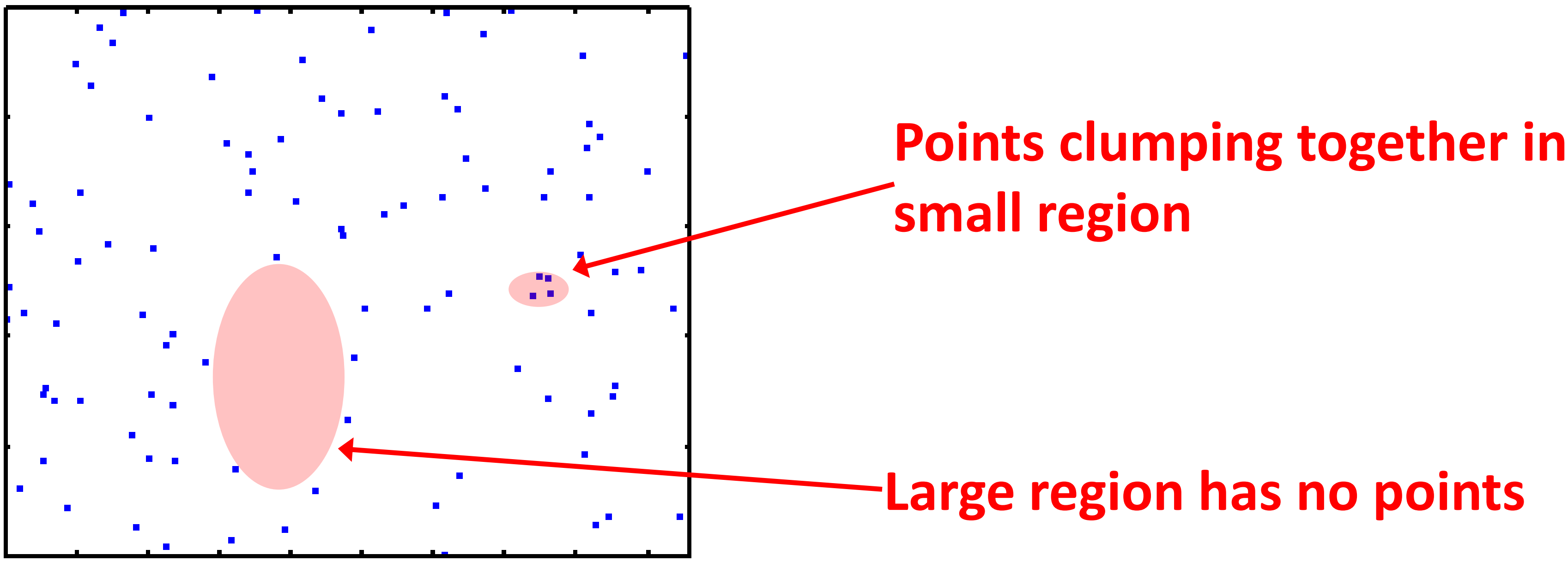
ML-driven Corners Analysis: The Stack

- 100% Python
  - Python 2.7, numpy, scipy
  - scikit-learn for base GPM
  - Custom ML:
    - Customized GPM for high # samples
    - Inner optimization via random search and derivative-free optimization
  - Qt4, Chaco
- 3rd party circuit simulator, environment

Example: Low-Discrepancy Sampling
Status quo: Pseudo-Random Sampling

- The typical simplistic approach to generate samples
- Draws each point separately from other points, using a pseudo-random number generator (e.g. Mersenne Twister)
- Has issues…

100 samples drawn from 2-d uniform distribution:

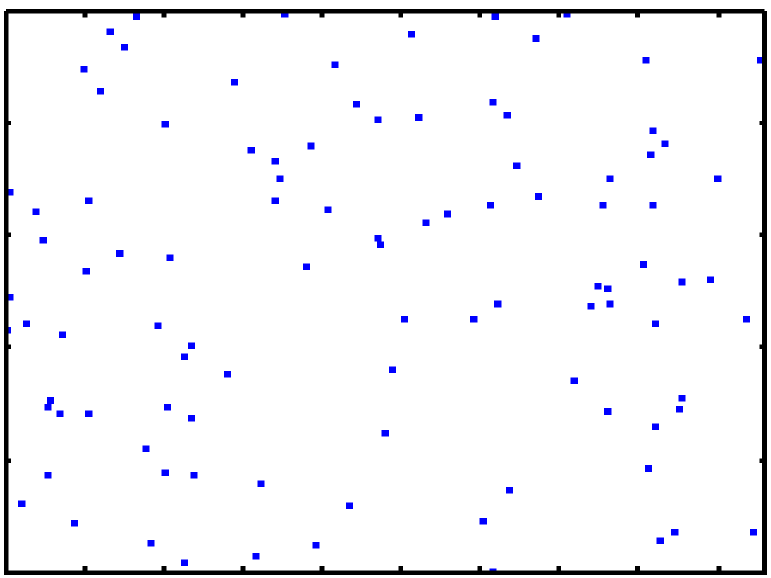


Example: Low-Discrepancy Sampling
Approach: Lattice Rules

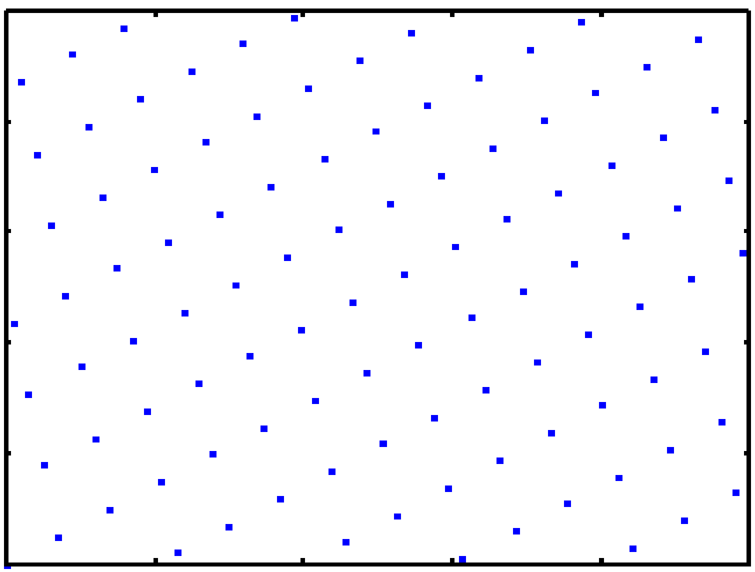
- Considers all the variables simultaneously (unlike Latin Hypercube)
- Works well in high dimensionality (unlike digital nets, e.g. Sobol')
- No heuristics necessary (unlike modified Sobol')

Example: 100 uniformly-distributed 2d points:

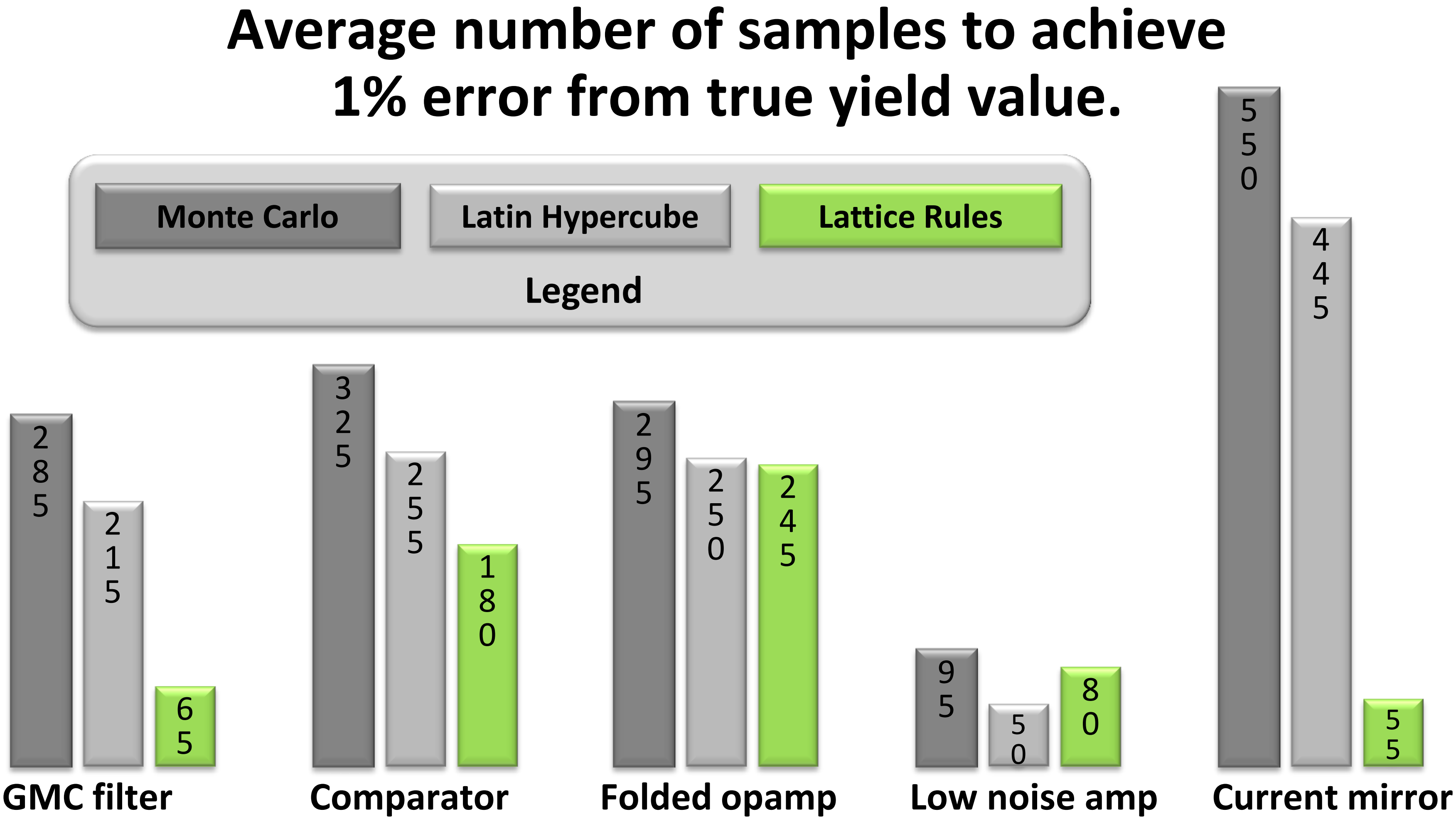
Pseudo-Random



Lattice Rules

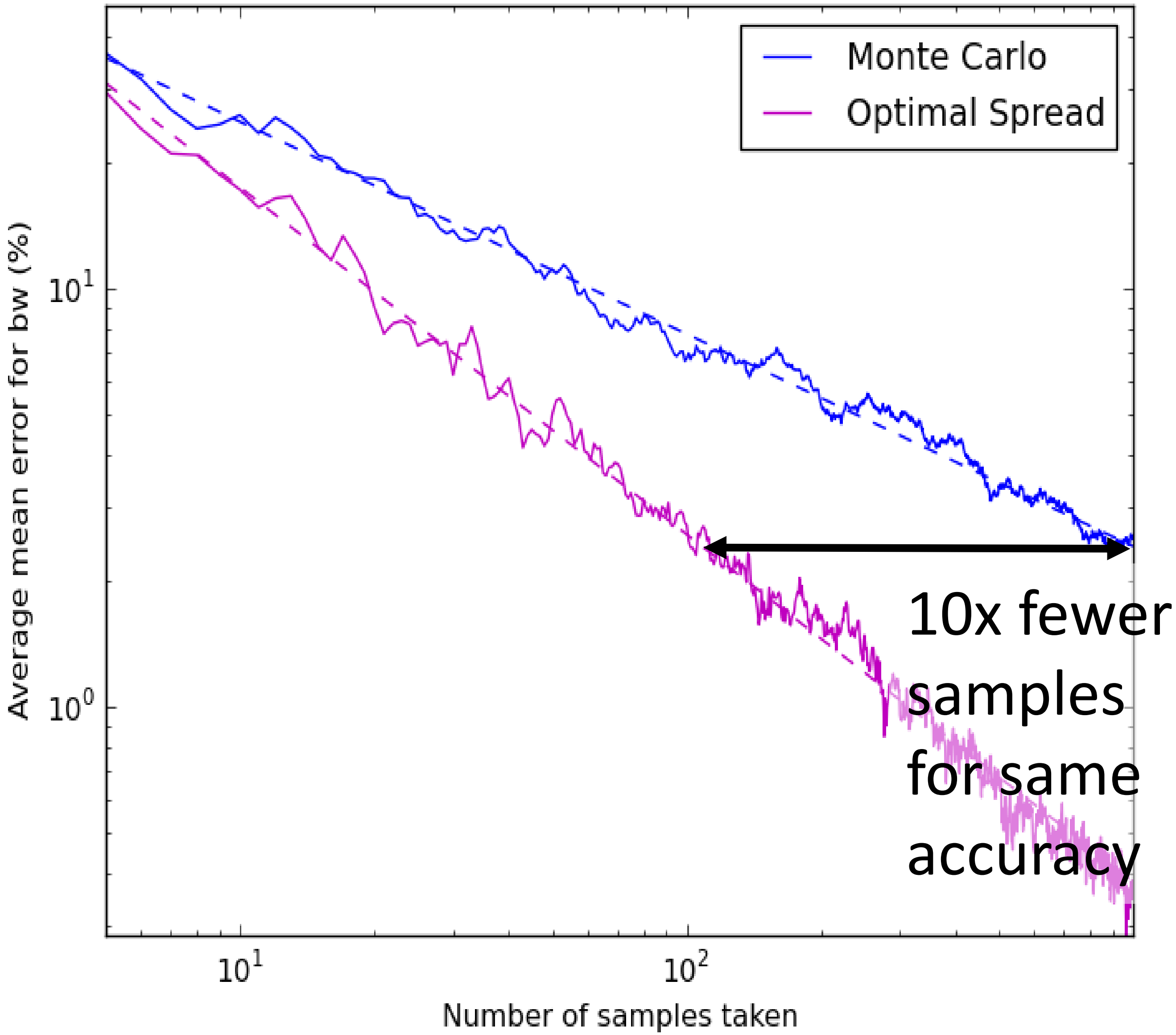


Example: Low-Discrepancy Sampling Benchmark for Yield Estimation

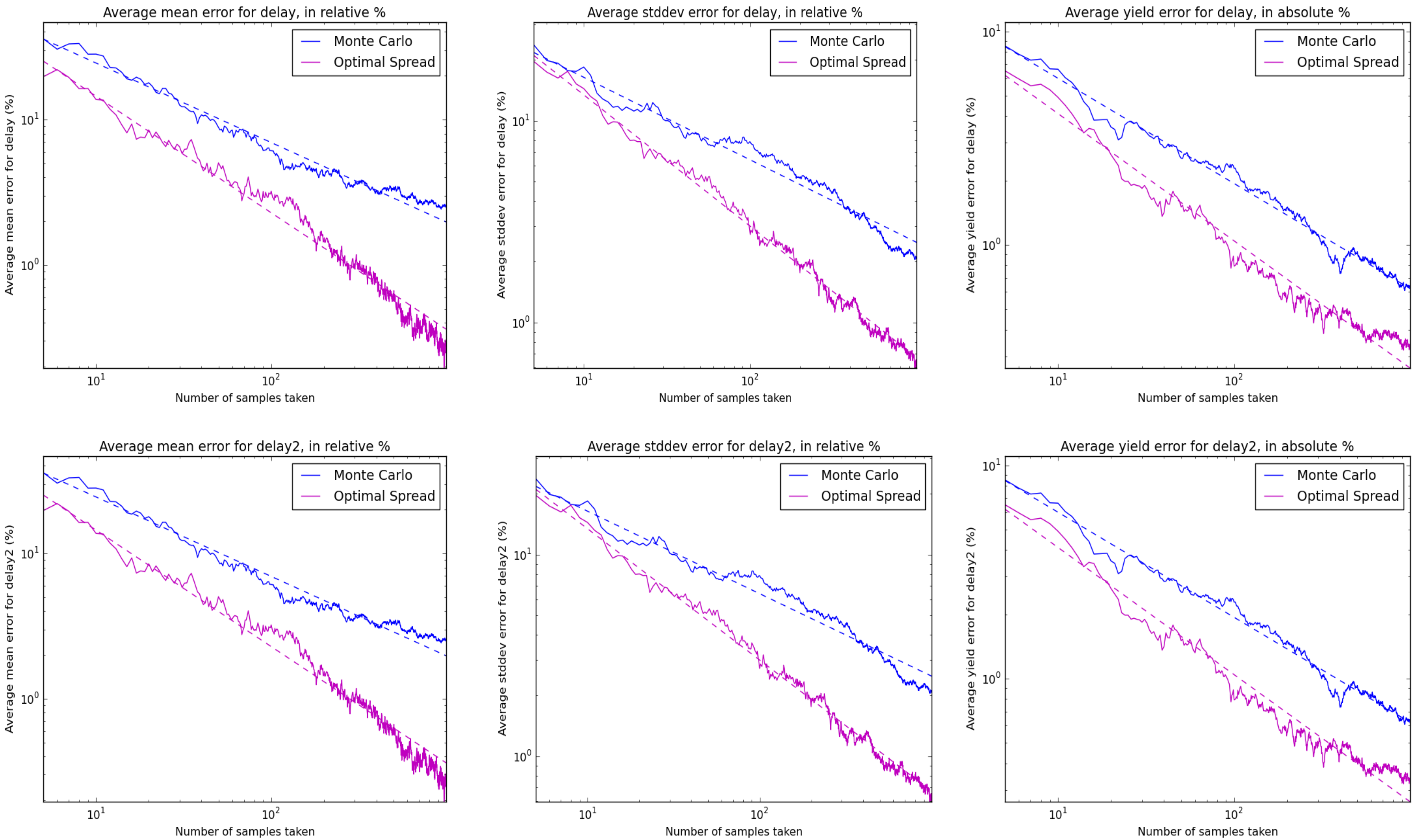


Example: Low-Discrepancy Sampling

Convergence of Pseudo-Random vs. OSS (In estimating mean of VGA bw)



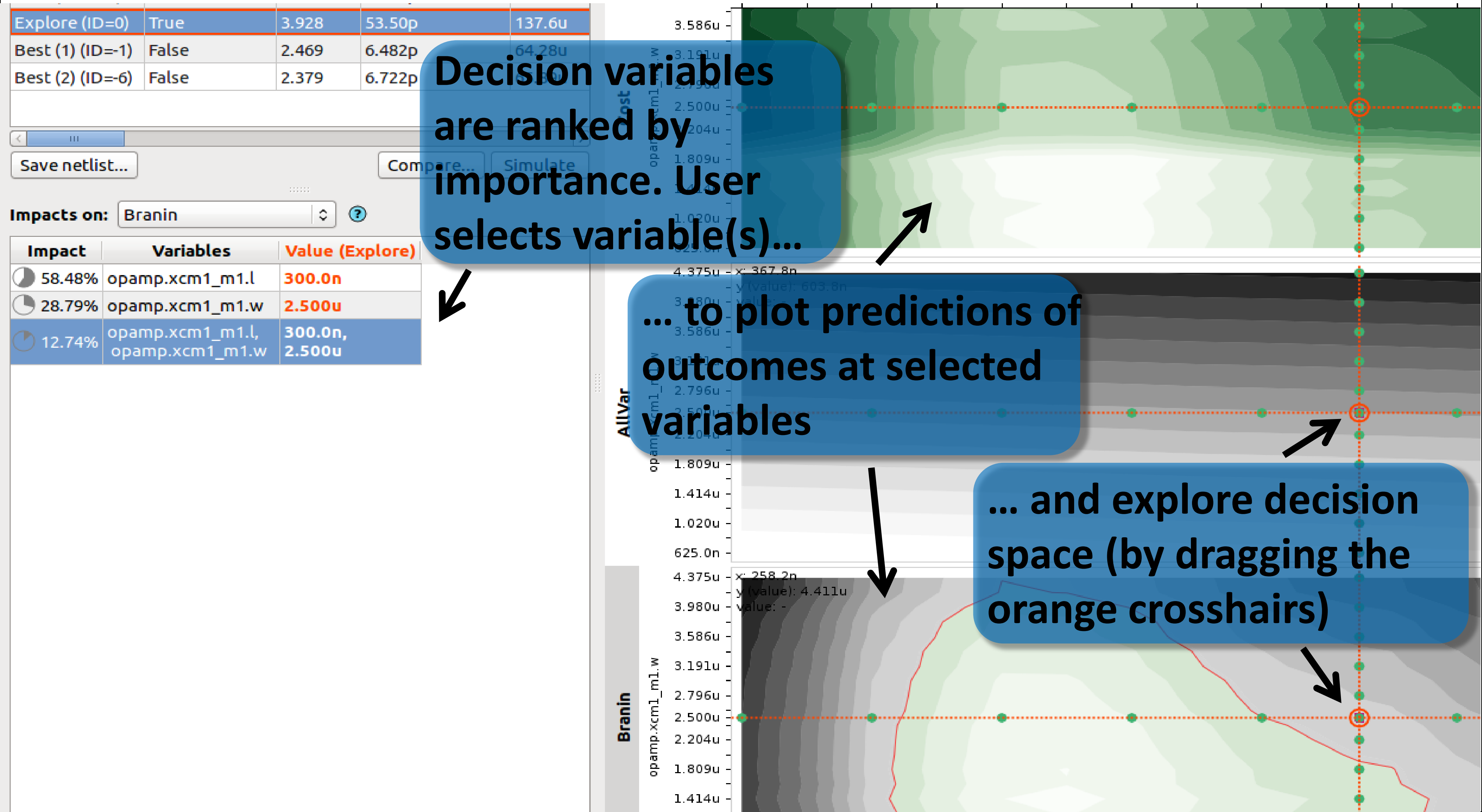
Example: Low-Discrepancy Sampling On Ring Oscillator



Example: ML-driven Design Space Exploration

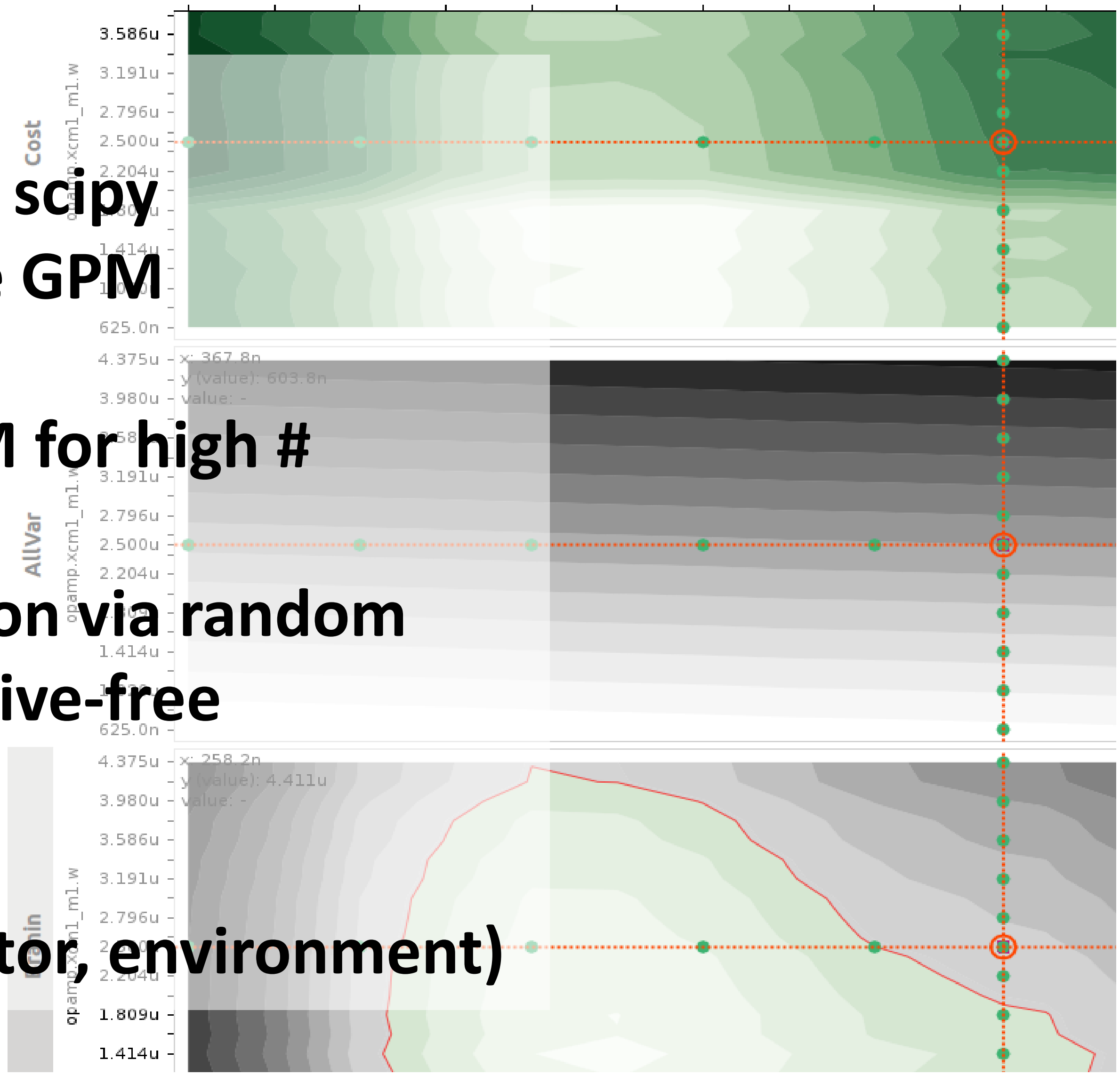
How: GPMs / high-dim Bayesian opt. + natural interface

Benefit: Speed of opt. with control & insight of manual



ML-driven Design Space Exploration: The Stack

- 100% Python
  - Python 2.7, numpy, scipy
  - scikit-learn for base GPM
  - Custom ML:
    - Customized GPM for high # samples
    - Inner optimization via random search & derivative-free optimization
  - Qt4, Chaco
- 3rd party circuit simulator, environment)



Summary of Python-powered ML inside Solido

- Regression with interpolation & CIs (KRC: scalability via divide-and-conquer on GPM)
- Model-based optimization, reliably finds global optimum by accounting for error in CIs

- 1-d density estimation (extrapolate via NQ)
- Low-discrepancy sampling (High dimensionality via modified Lattice Rules)
- Data mining for variable sensitivities
- Fast-evaluation opt. (evolutionary progr.)
- Regression w/ interpolation; model-based opt.

- Rare-event estimation (HSMC algorithm: transform into ranking problem, solve with adaptive sampling)
- High-dimensional regression (FFX: pathwise learning on huge # basis functions)
- High-dimensional classification (FFXC: pathwise ..)
- Data mining for variable sensitivities

Fast PVT
2-50X faster verification across PVT corners

Fast MC
2-10x faster 3σ verification, statistical corners

High-Sigma MC
Fast, accurate, scalable, verifiable 6σ Monte Carlo

Hier. MC
Fast statistical memory array / column analysis

Cell Optimizer
Auto variation-aware design space exploration of memory/std cells

Fast Design Sweep
Fast, thorough manual variation-aware design space exploration

- Model-based optimization
- Regression with interpolation & CIs (KRC: scalability via divide-and-conquer on GPM)

- Active learning via model-based optimization
- Regression with interpolation & CIs (KRC: scalability via divide-and-conquer on GPM)
- High-dimensional visualization / sweep exploration
- Data mining for variable sensitivities
- Data mining for variable-interaction sensitivities

- MC sampling on hierarchically organized design (Fast Hier MC algorithm: transform into ranking problem, solve with adaptive sampling)
- High-dimensional regression (FFX)
- High-dimensional classification (FFXC)
- Data mining for variable sensitivities

Conclusion: Python & ML Help Drive Moore's Law

Silicon Midas touch *applied to itself*

(It helped to design the phone in your pocket, the servers on the cloud, …)