Parallel Programming
in C with MPI and OpenMP

Michael J. Quinn

Chapter 10

Monte Carlo Methods

Chapter Objectives

- Introduce Monte Carlo methods
- Introduce techniques for parallel random number generation

Outline

- Monte Carlo method
- Sequential random number generators
- Parallel random number generators
- Generating non-uniform random numbers
- Monte Carlo case studies

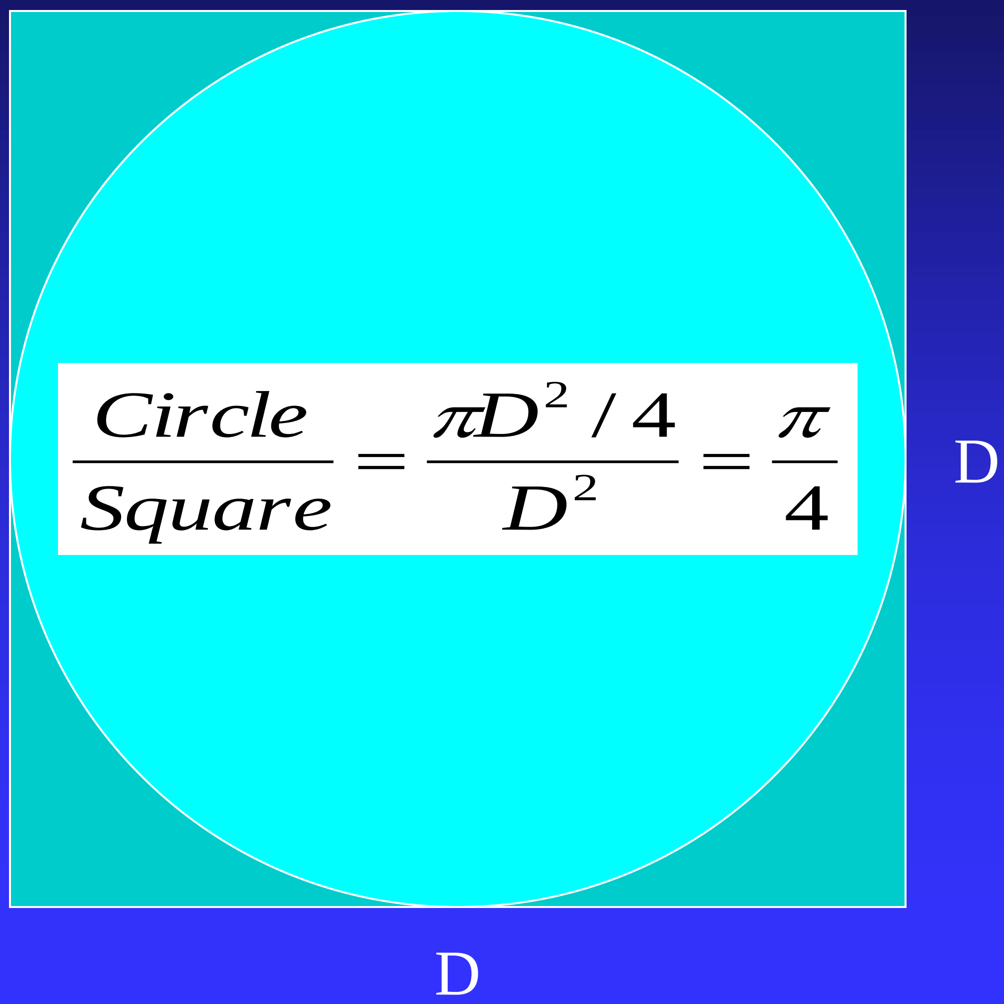
Monte Carlo Method

- Solve a problem using statistical sampling
- Name comes from Monaco's gambling resort city
- First important use in development of atomic bomb during World War II

Applications of Monte Carlo Method

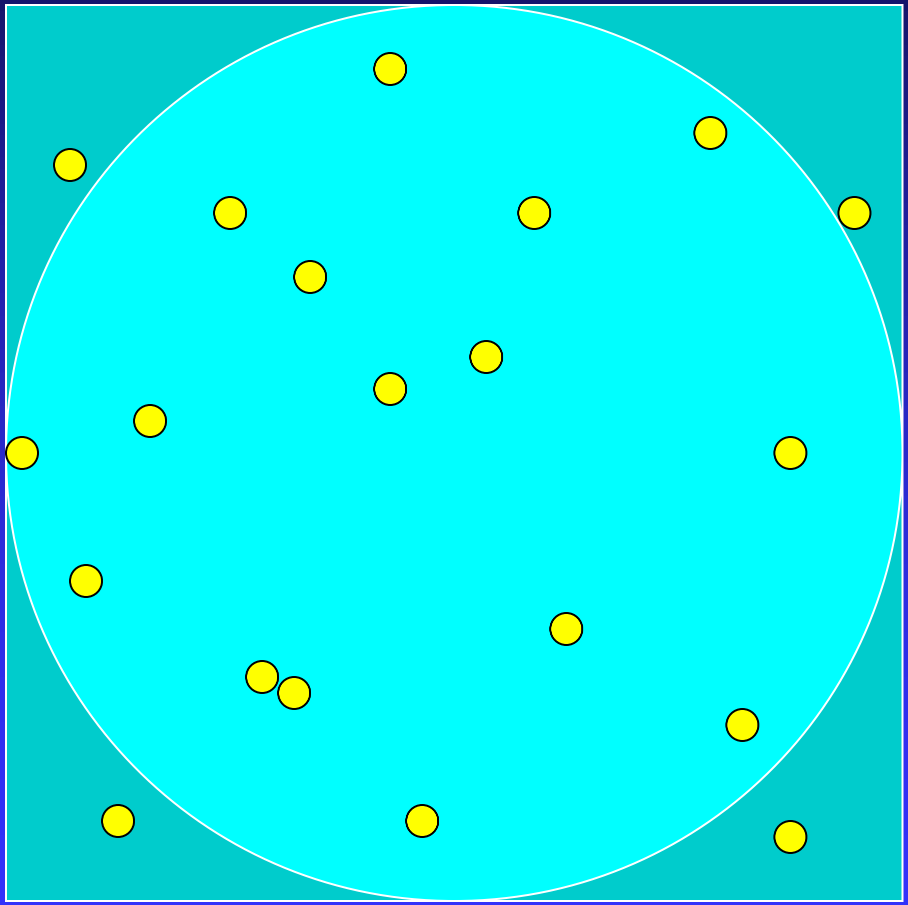
- Evaluating integrals of arbitrary functions of 6+ dimensions
- Predicting future values of stocks
- Solving partial differential equations
- Sharpening satellite images
- Modeling cell populations
- Finding approximate solutions to NP-hard problems

Example of Monte Carlo Method



$$\frac{Circle}{Square} = \frac{\pi D^2 / 4}{D^2} = \frac{\pi}{4}$$

Example of Monte Carlo Method



$$\frac{16}{20} \approx \frac{\pi}{4} \Rightarrow \pi \approx 3.2$$

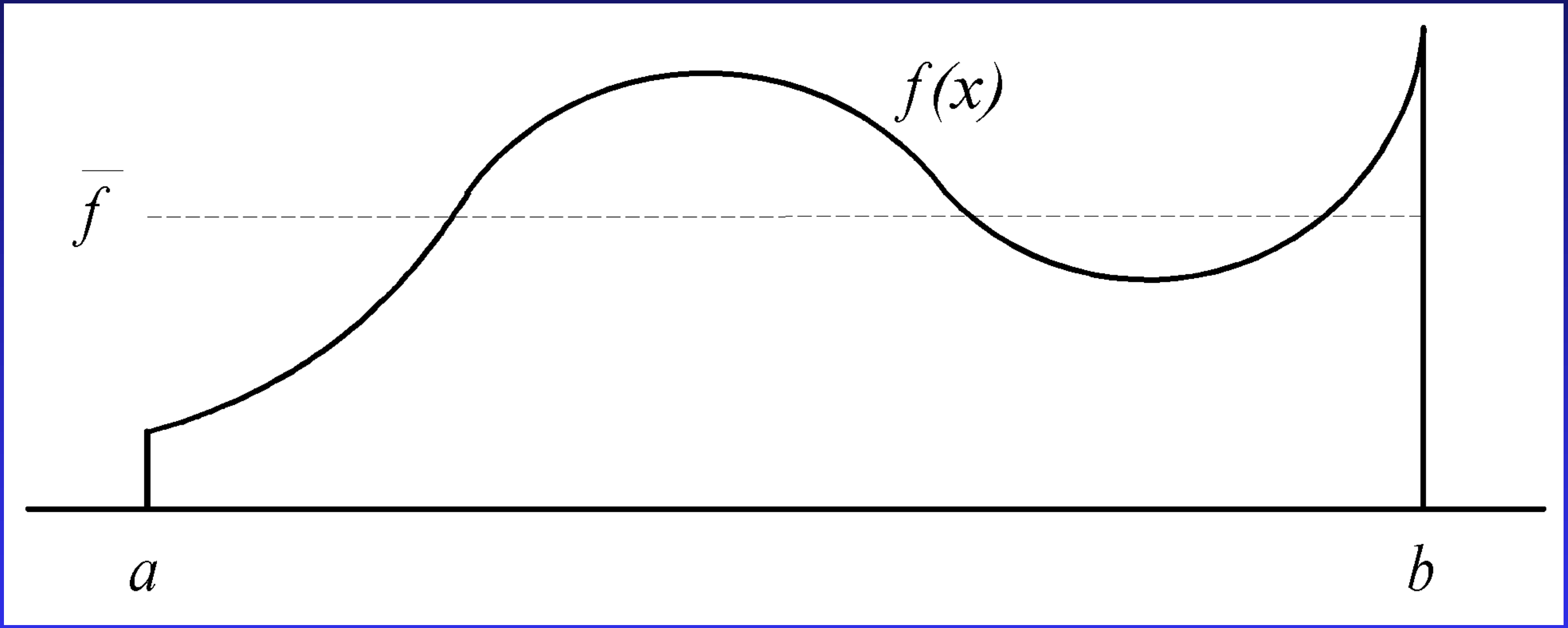
Absolute Error

- Absolute error is a way to measure the quality of an estimate
- The smaller the error, the better the estimate
- a: actual value
- e: estimated value
- Absolute error = $|e-a|/a$

Increasing Sample Size Reduces Error

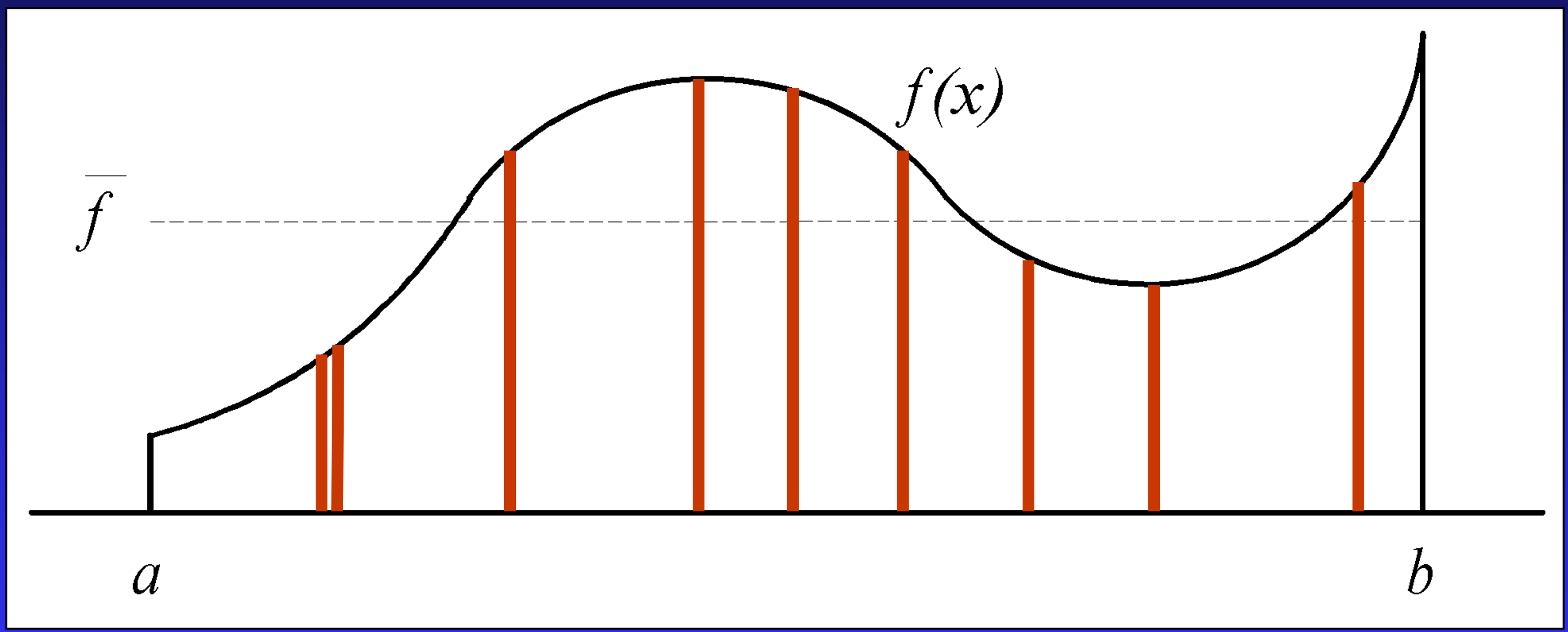
n	*Estimate*	*Error*	*$1/(2n^{1/2})$*
10	2.40000	0.23606	0.15811
100	3.36000	0.06952	0.05000
1,000	3.14400	0.00077	0.01581
10,000	3.13920	0.00076	0.00500
100,000	3.14132	0.00009	0.00158
1,000,000	3.14006	0.00049	0.00050
10,000,000	3.14136	0.00007	0.00016
100,000,000	3.14154	0.00002	0.00005
1,000,000,000	3.14155	0.00001	0.00002

Mean Value Theorem



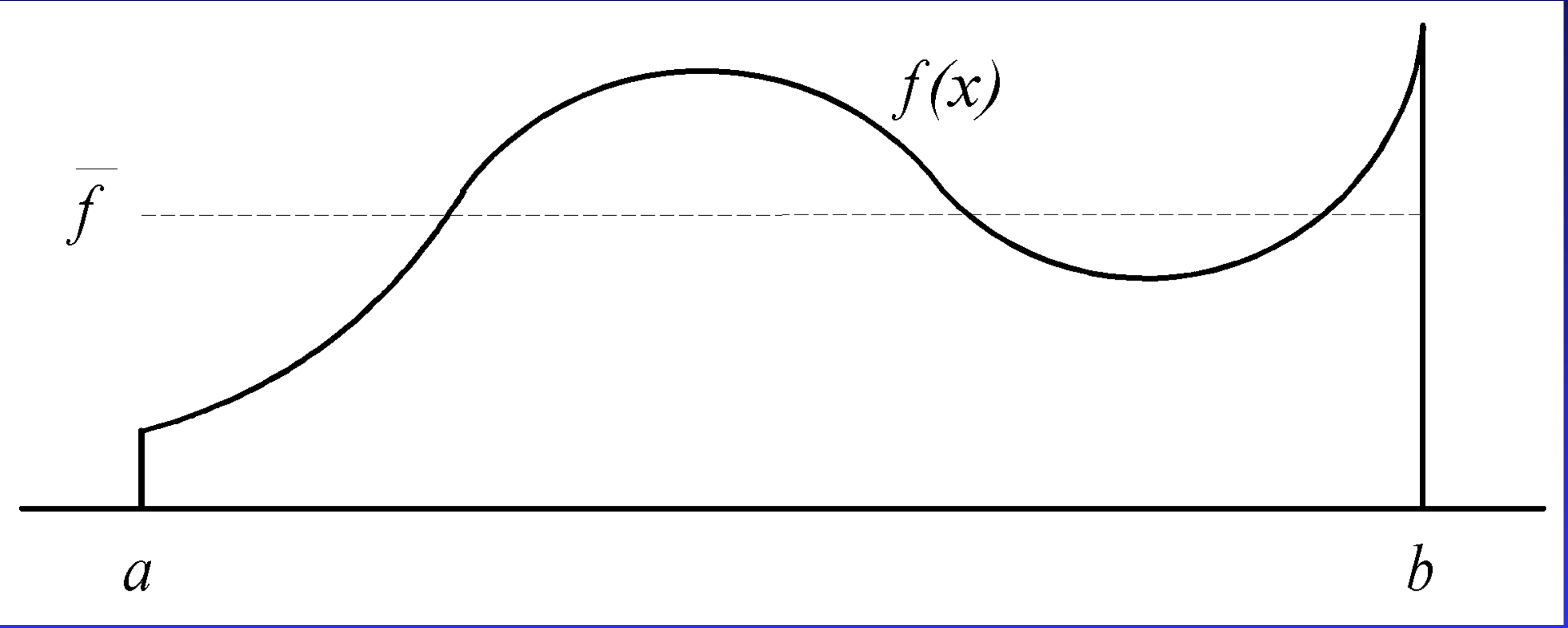
$$\int_a^b f(x)dx = (b-a)\,\bar{f}$$

Estimating Mean Value



The expected value of $(1/n)(f(x_0) + \ldots + f(x_{n-1}))$ is $\bar{f}$

Why Monte Carlo Works



$$\int_a^b f(x)dx = (b-a)\,\bar{f} \approx (b-a)\frac{1}{n}\sum_{i=0}^{n-1} f(x_i)$$

Why Monte Carlo is Effective

- Error in Monte Carlo estimate decreases by the factor $1/n^{1/2}$
- Rate of convergence independent of integrand's dimension
- Deterministic numerical integration methods do not share this property
- Hence Monte Carlo superior when integrand has 6 or more dimensions

Parallelism in Monte Carlo Methods

- Monte Carlo methods often amenable to parallelism
- Find an estimate about *p* times faster

OR

- Reduce error of estimate by $p^{1/2}$

Random versus Pseudo-random

- Virtually all computers have "random number" generators
- Their operation is deterministic
- Sequences are predictable
- More accurately called "pseudo-random number" generators
- In this chapter "random" is shorthand for "pseudo-random"
- "RNG" means "random number generator"

Properties of an Ideal RNG

- Uniformly distributed
- Uncorrelated
- Never cycles
- Satisfies any statistical test for randomness
- Reproducible
- Machine-independent
- Changing "seed" value changes sequence
- Easily split into independent subsequences
- Fast
- Limited memory requirements

No RNG Is Ideal

- Finite precision arithmetic ⇒ finite number of states ⇒ cycles
  - Period = length of cycle
  - If period > number of values needed, effectively acyclic
- Reproducible ⇒ correlations
- Often speed versus quality trade-offs

Linear Congruential RNGs

$$X_i = (a \times X_{i-1} + c) \bmod M$$

Multiplier

Additive constant

Modulus

Sequence depends on choice of seed, X_0

Period of Linear Congruential RNG

- Maximum period is M
- For 32-bit integers maximum period is 2^{32}, or about 4 billion
- This is too small for modern computers
- Use a generator with at least 48 bits of precision

Producing Floating-Point Numbers

- X_i, a, c, and M are all integers
- X_is range in value from 0 to M-1
- To produce floating-point numbers in range [0, 1), divide X_i by M

Defects of Linear Congruential RNGs

- Least significant bits correlated
  - Especially when M is a power of 2
- k-tuples of random numbers form a lattice
  - Especially pronounced when k is large

Lagged Fibonacci RNGs

$$X_i = X_{i-p} * X_{i-q}$$

- p and q are lags, $p > q$
- * is any binary arithmetic operation
  - Addition modulo M
  - Subtraction modulo M
  - Multiplication modulo M
  - Bitwise exclusive or

Properties of Lagged Fibonacci RNGs

- Require *p* seed values
- Careful selection of seed values, *p*, and *q* can result in very long periods and good randomness
- For example, suppose *M* has *b* bits
- Maximum period for additive lagged Fibonacci RNG is $(2^p - 1)2^{b-1}$

Ideal Parallel RNGs

- All properties of sequential RNGs
- No correlations among numbers in different sequences
- Scalability
- Locality

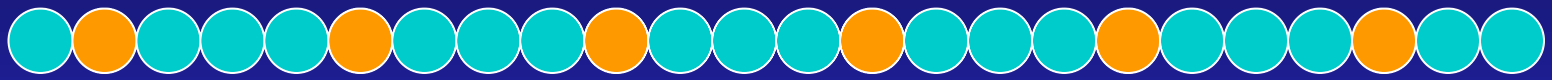
Parallel RNG Designs

- Manager-worker
- Leapfrog
- Sequence splitting
- Independent sequences

Manager-Worker Parallel RNG

- Manager process generates random numbers
- Worker processes consume them
- If algorithm is synchronous, may achieve goal of consistency
- Not scalable
- Does not exhibit locality

Leapfrog Method



Process with rank 1 of 4 processes

Properties of Leapfrog Method

- Easy modify linear congruential RNG to support jumping by *p*
- Can allow parallel program to generate same tuples as sequential program
- Does not support dynamic creation of new random number streams

Sequence Splitting

Process with rank 1 of 4 processes

Properties of Sequence Splitting

- Forces each process to move ahead to its starting point
- Does not support goal of reproducibility
- May run into long-range correlation problems
- Can be modified to support dynamic creation of new sequences

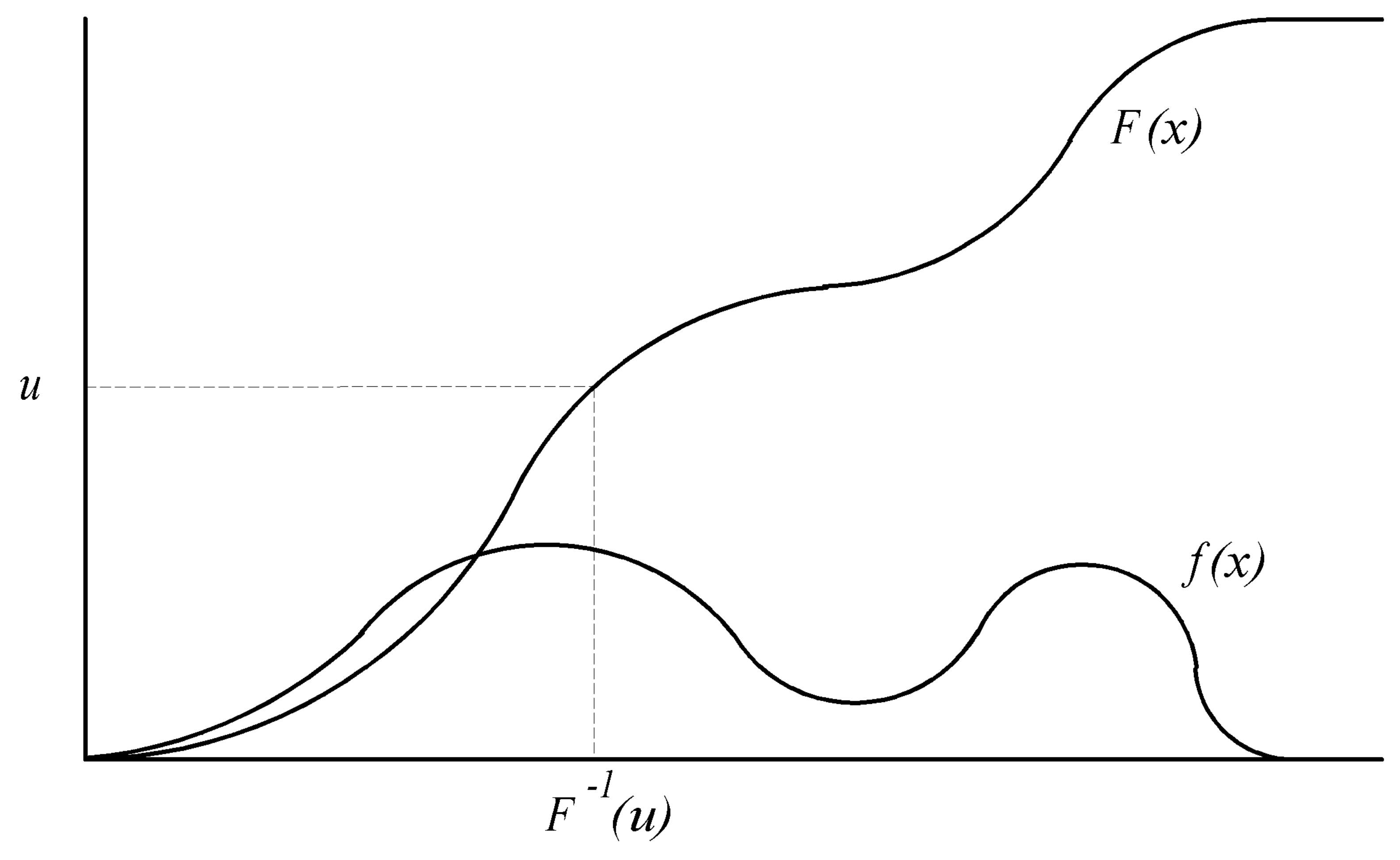
Independent Sequences

- Run sequential RNG on each process
- Start each with different seed(s) or other parameters
- Example: linear congruential RNGs with different additive constants
- Works well with lagged Fibonacci RNGs
- Supports goals of locality and scalability

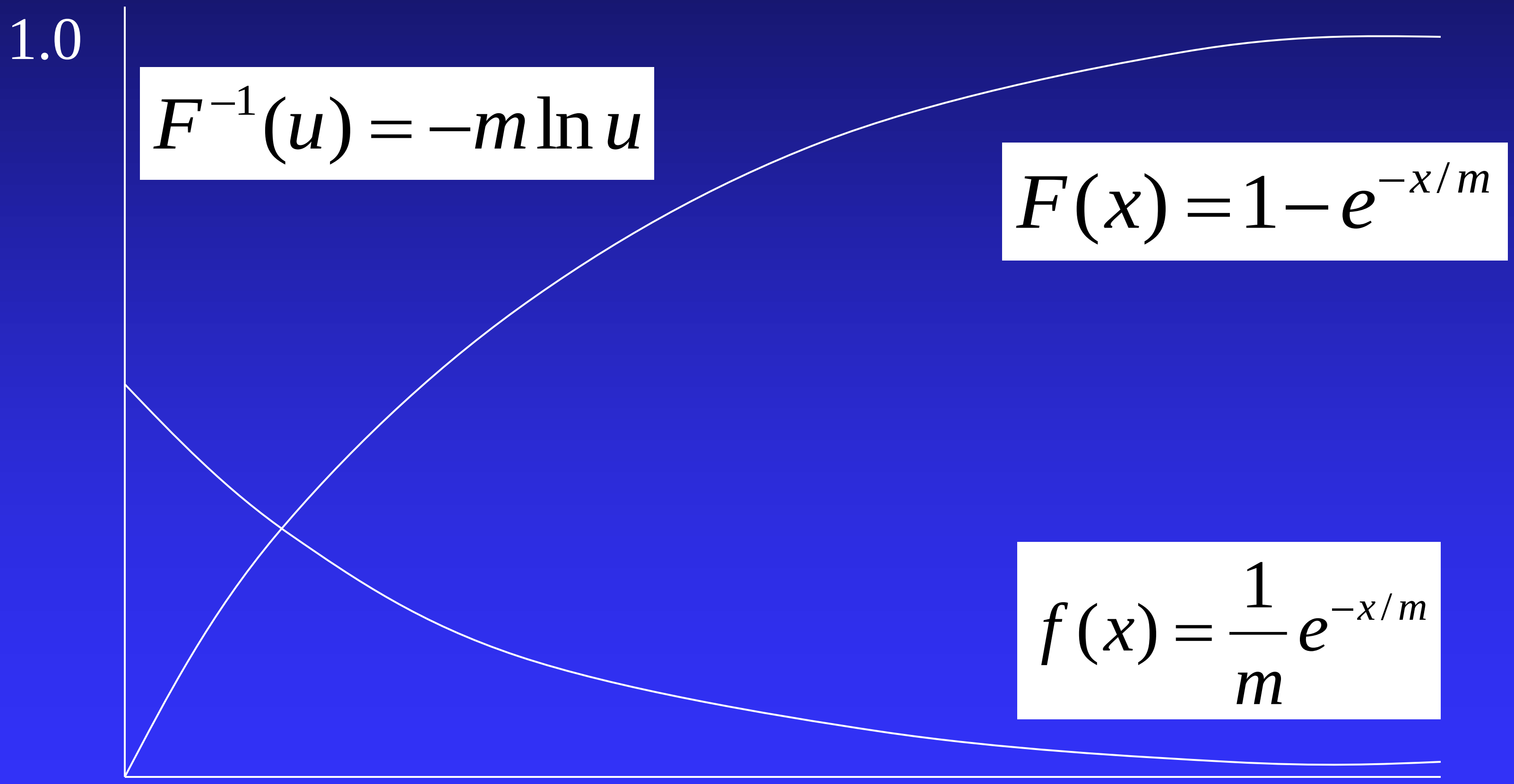
Other Distributions

- Analytical transformations
- Box-Muller Transformation
- Rejection method

Analytical Transformation



Exponential Distribution



Example 1:

- Produce four samples from an exponential distribution with mean 3
- Uniform sample: 0.540, 0.619, 0.452, 0.095
- Take natural log of each value and multiply by -3
- Exponential sample: 1.850, 1.440, 2.317, 7.072

Example 2:

- Simulation advances in time steps of 1 second
- Probability of an event happening is from an exponential distribution with mean 5 seconds
- What is probability that event will happen in next second?
- 1/5
- Use uniform random number to test for occurrence of event

Box-Muller Transformation

- Cannot invert cumulative distribution function to produce formula yielding random numbers from normal (gaussian) distribution
- Box-Muller transformation produces a pair of standard deviates g_1 and g_2 from a pair of normal deviates u_1 and u_2

Box-Muller Transformation

repeat

$v_1 \leftarrow 2u_1 - 1$

$v_2 \leftarrow 2u_2 - 1$

$r \leftarrow v_1^2 + v_2^2$

until $r > 0$ and $r < 1$

$f \leftarrow$ sqrt (-2 ln r/r)

$g_1 \leftarrow f\, v_1$

$g_2 \leftarrow f\, v_2$

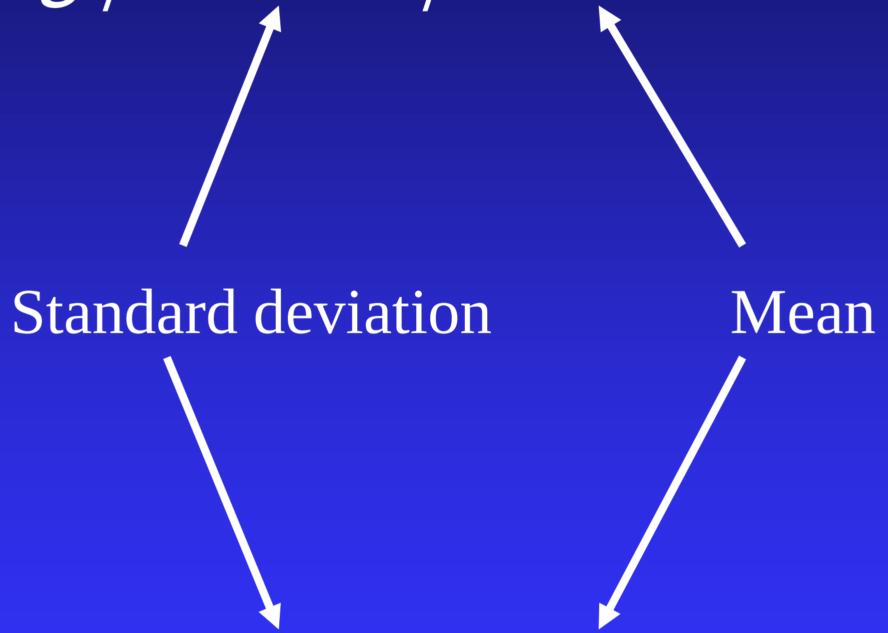
Example

- Produce four samples from a normal distribution with mean 0 and standard deviation 1

u_1	u_2	v_1	v_2	r	f	g_1	g_2
0.234	0.784	-0.532	0.568	0.605	1.290	-0.686	0.732
0.824	0.039	0.648	-0.921	1.269			
0.430	0.176	-0.140	-0.648	0.439	1.935	-0.271	-1.254

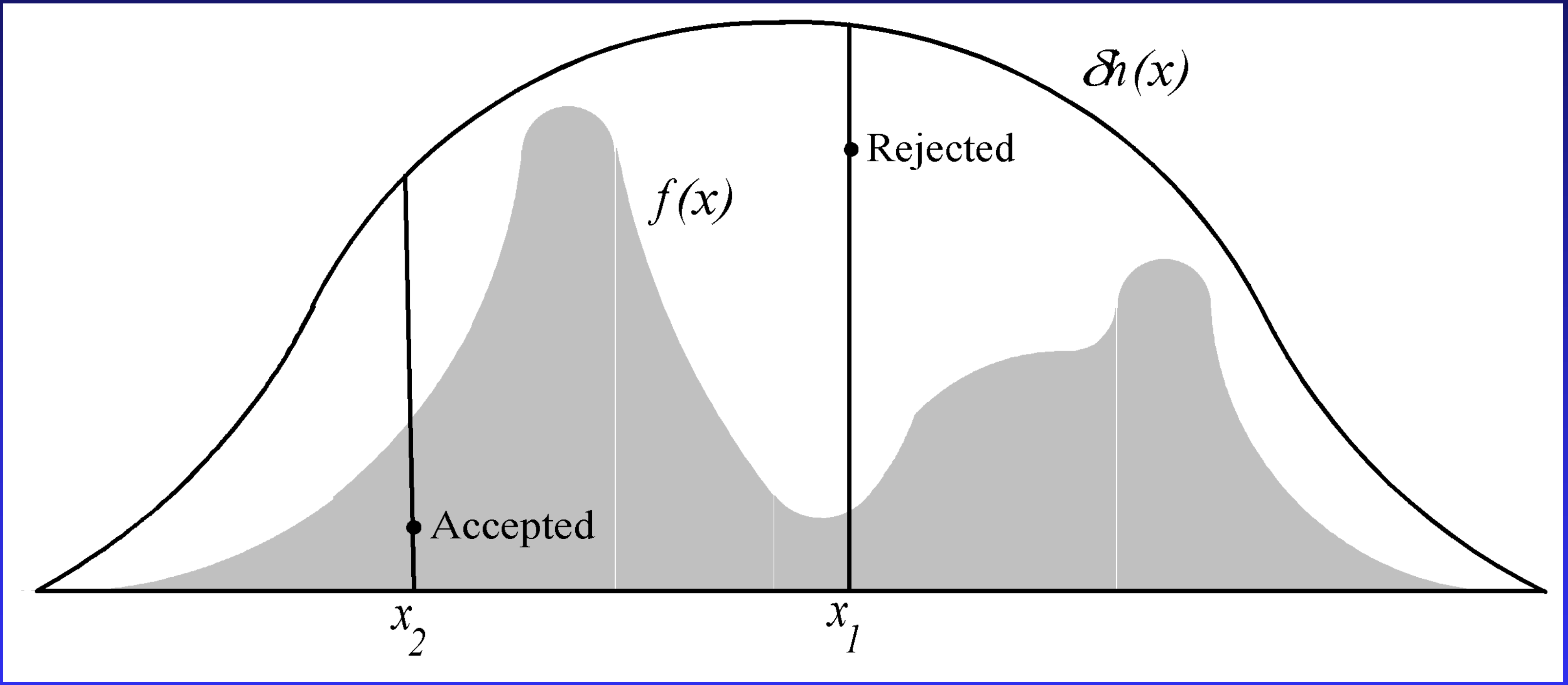
Different Mean, Std. Dev.

$$g_1 \leftarrow s\, f\, v_1 + m$$



$$g_2 \leftarrow s\, f\, v_2 + m$$

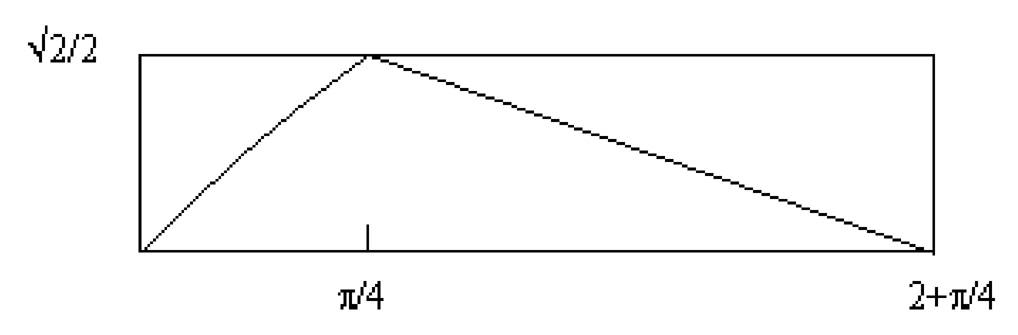
Rejection Method



Example

- Generate random variables from this probability density function

$$f(x) = \begin{cases} \sin x, & \text{if } 0 \le x \le \pi/4 \\ (-4x + \pi + 8)/(8\sqrt{2}), & \text{if } \pi/4 < x \le 2 + \pi/4 \\ 0, & \text{otherwise} \end{cases}$$



Example (cont.)

$$h(x) = \begin{cases} 1/(2+\pi/4), & \text{if } 0 \le x \le 2+\pi/4 \\ 0, & \text{otherwise} \end{cases}$$

$$\delta = (2+\pi/4)/(\sqrt{2}/2)$$

$$\delta h(x) = \begin{cases} \sqrt{2}/2, & \text{if } 0 \le x \le 2+\pi/4 \\ 0, & \text{otherwise} \end{cases}$$

So $\delta\, h(x) \ge f(x)$ for all x

Example (cont.)

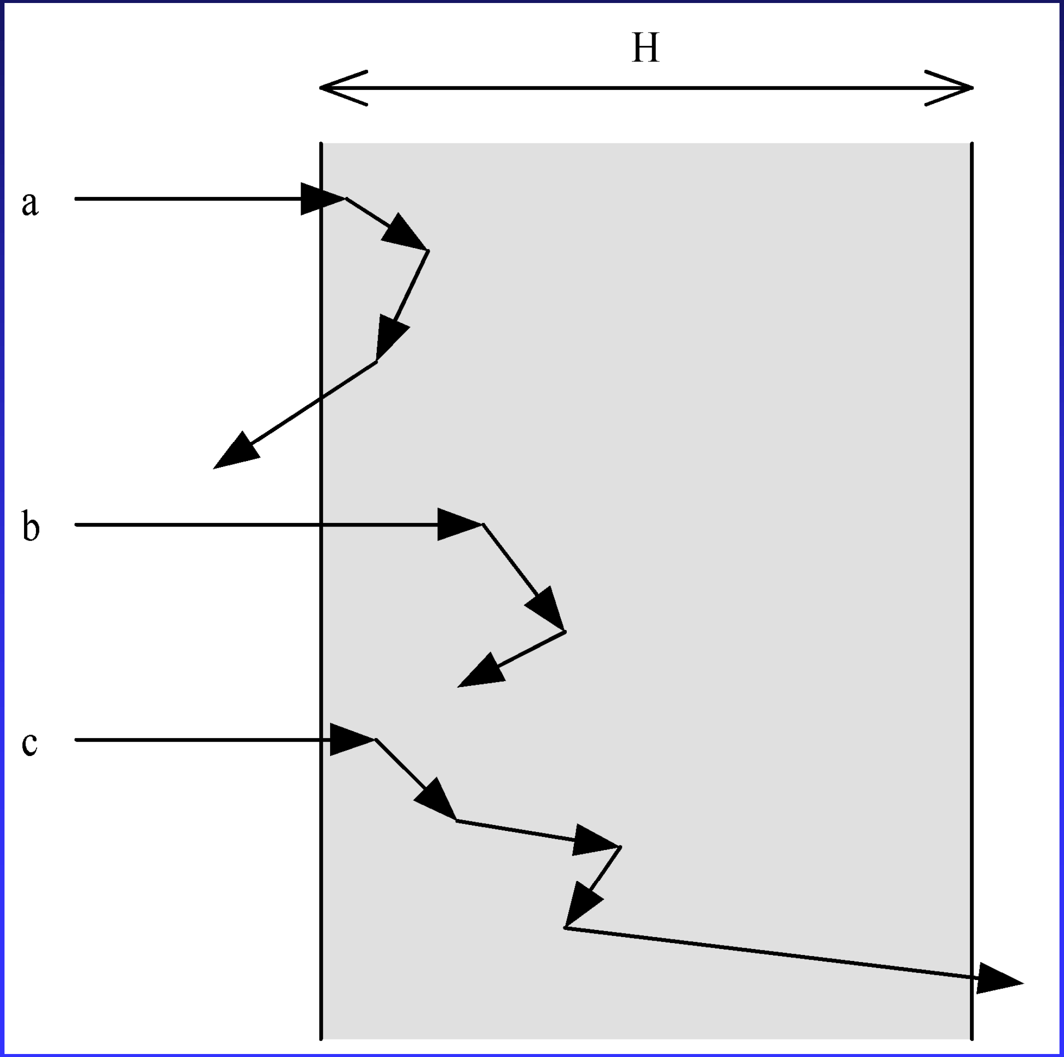
x_i	u_i	$u_i \delta h(x_i)$	$f(x_i)$	**Outcome**
0.860	0.975	0.689	0.681	Reject
1.518	0.357	0.252	0.448	Accept
0.357	0.920	0.650	0.349	Reject
1.306	0.272	0.192	0.523	Accept

Two samples from $f(x)$ are 1.518 and 1.306

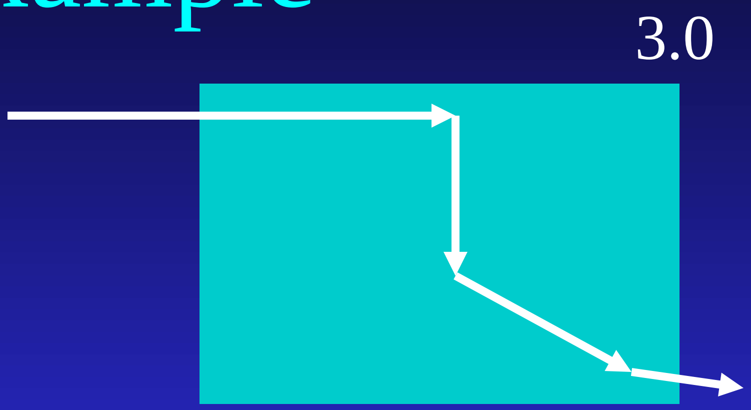
Case Studies (Topics Introduced)

- Neutron transport (Monte Carlo time)
- Temperature inside a 2-D plate (Random walk)
- Two-dimensional Ising model (Metropolis algorithm)
- Room assignment problem (Simulated annealing)
- Parking garage (Monte Carlo time)
- Traffic circle (Simulating queues)

Neutron Transport



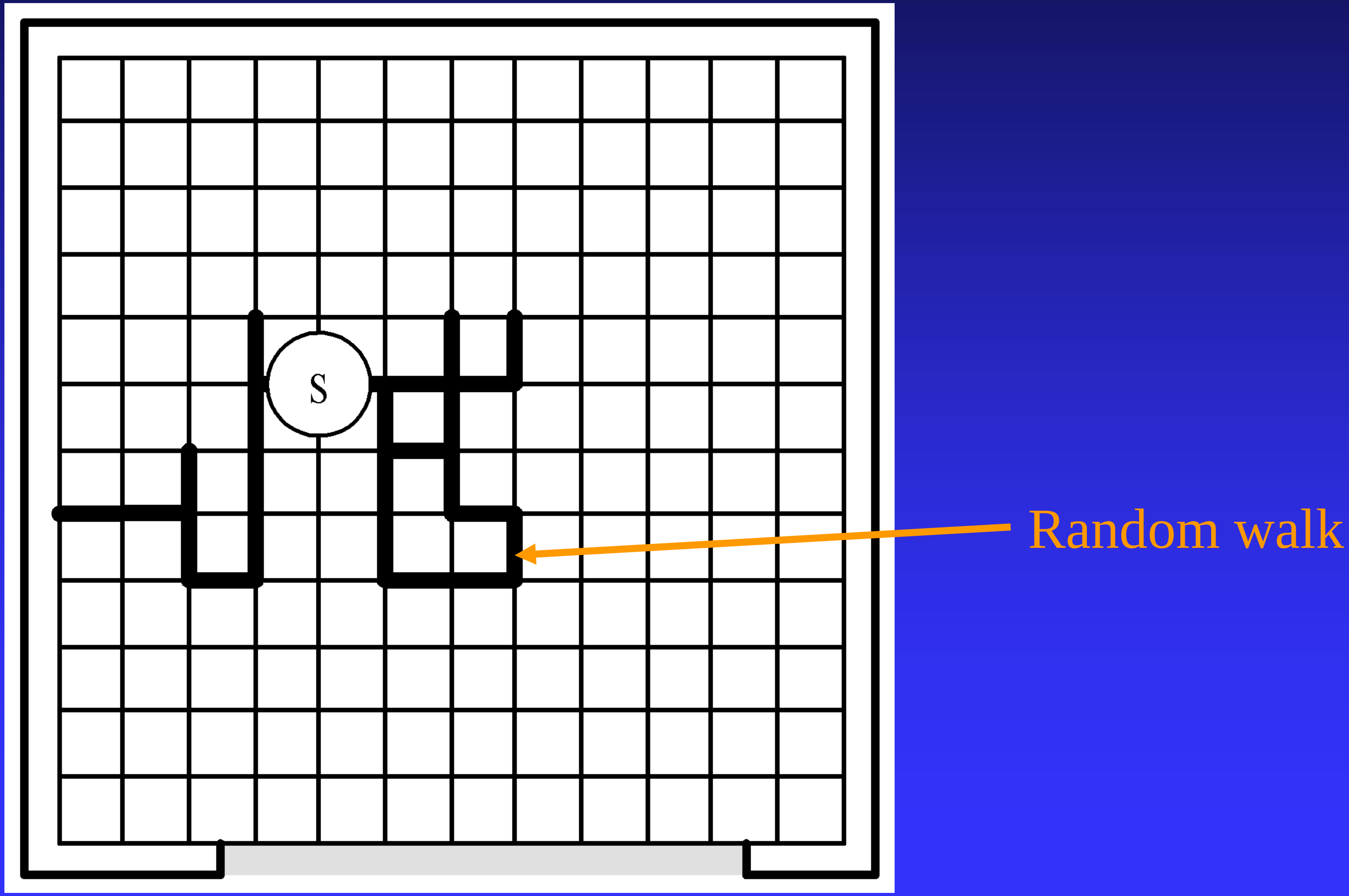
Example



Monte Carlo Time

D (0-π)	Angle	u (0-1)	L (-ln u)	Lcos$D$	Dist.	Absorb? (0-1)
0.00	0.0	0.20	1.59	1.59	1.59	0.41 (no)
1.55	89.2	0.34	1.08	0.01	1.60	0.84 (no)
0.42	24.0	0.27	1.31	1.20	2.80	0.57 (no)
0.33	19.4	0.60	0.52	0.49	3.29	

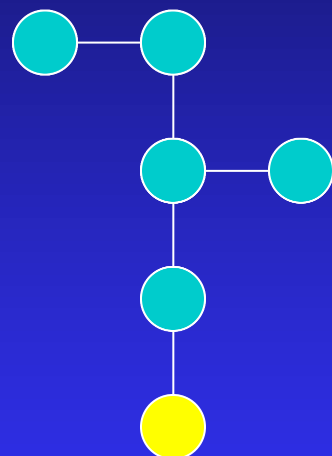
Temperature Inside a 2-D Plate



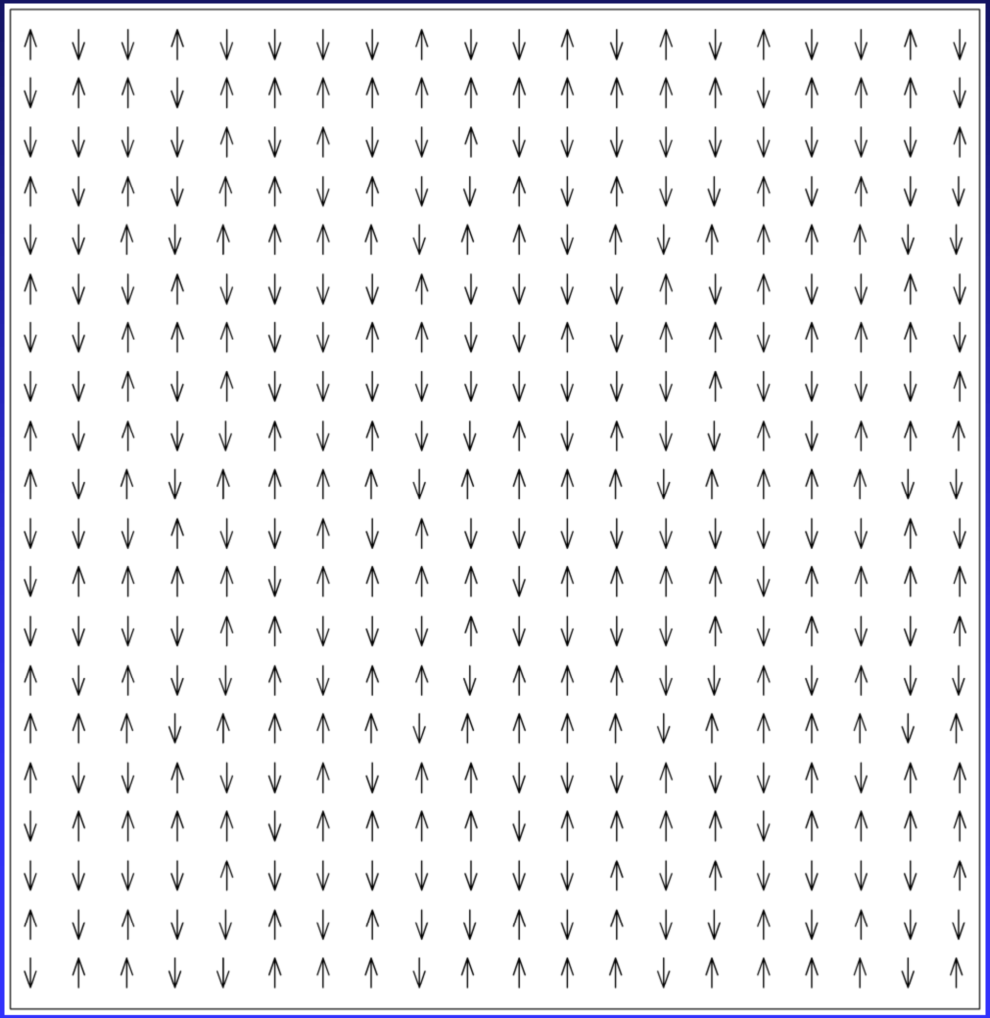
Example of Random Walk

$$0 \le u < 1 \Rightarrow \lfloor 4u \rfloor \in \{0,1,2,3\}$$

2



2-D Ising Model



Metropolis Algorithm

- Use current random sample to generate next random sample
- Series of samples represents a random walk through the probability density function
- Short series of samples highly correlated
- Many samples can provide good coverage

Metropolis Algorithm Details

- Randomly select site to reverse spin
- If energy is lower, move to new state
- Otherwise, move with probability $\rho = e^{-\Delta/kT}$
- Rejection causes current state to be recorded another time

Room Assignment Problem

	A	B	C	D	E	F
A	0	3	5	9	1	6
B	3	0	2	6	4	5
C	5	2	0	8	9	2
D	9	6	8	0	3	4
E	1	4	9	3	0	5
F	6	5	2	4	5	0

"Dislikes" matrix

Pairing A-B, C-D, and E-F leads to total conflict value of 32.

Physical Annealing

- Heat a solid until it melts
- Cool slowly to allow material to reach state of minimum energy
- Produces strong, defect-free crystal with regular structure

Simulated Annealing

- Makes analogy between physical annealing and solving combinatorial optimization problem
- Solution to problem = state of material
- Value of objective function = energy associated with state
- Optimal solution = minimum energy state

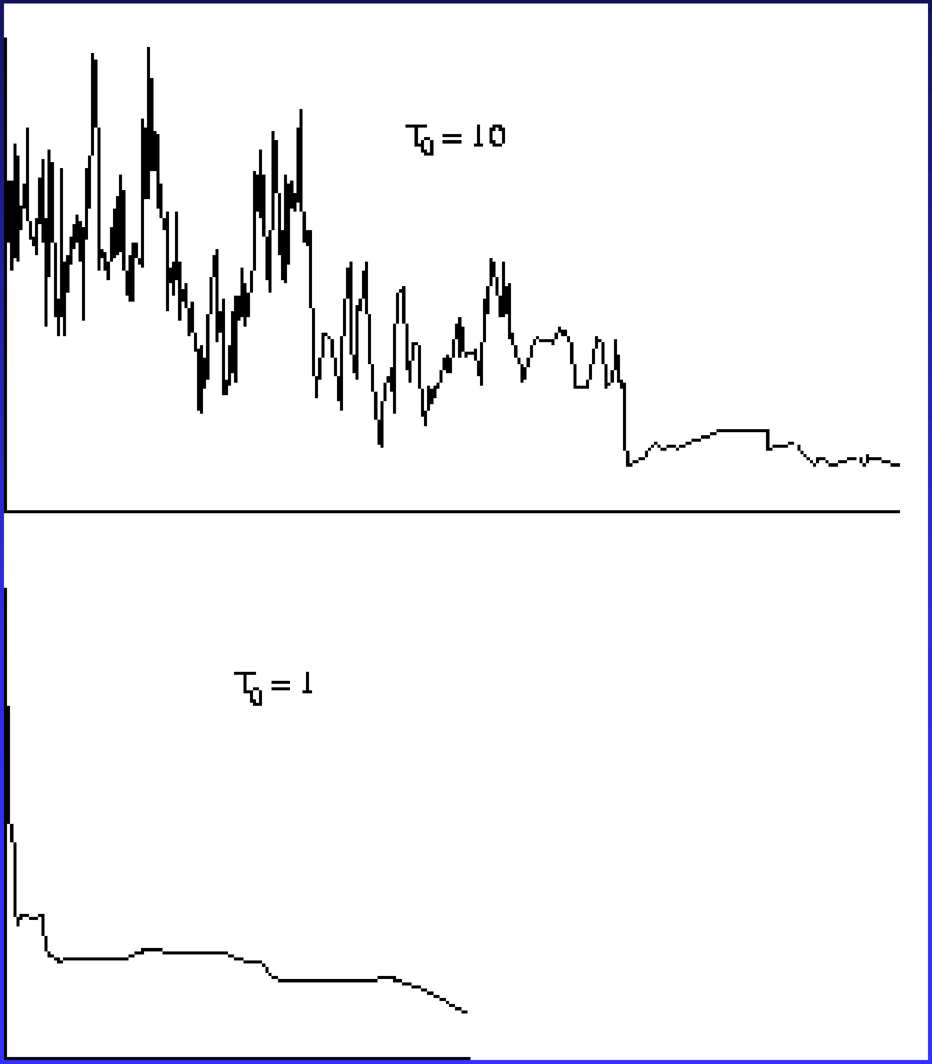
How Simulated Annealing Works

- Iterative algorithm, slowly lower T
- Randomly change solution to create alternate solution
- Compute Δ, the change in value of objective function
- If $\Delta < 0$, then jump to alternate solution
- Otherwise, jump to alternate solution with probability $e^{-\Delta/T}$

Performance of Simulated Annealing

- Rate of convergence depends on initial value of T and temperature change function
- Geometric temperature change functions typical; e.g., $T_{i+1} = 0.999\ T_i$
- Not guaranteed to find optimal solution
- Same algorithm using different random number streams can converge on different solutions
- Opportunity for parallelism

Convergence



Starting with higher initial temperature leads to more iterations before convergence

Parking Garage

- Parking garage has S stalls
- Car arrivals fit Poisson distribution with mean A
- Stay in garage fits a normal distribution with mean M and standard deviation M/S

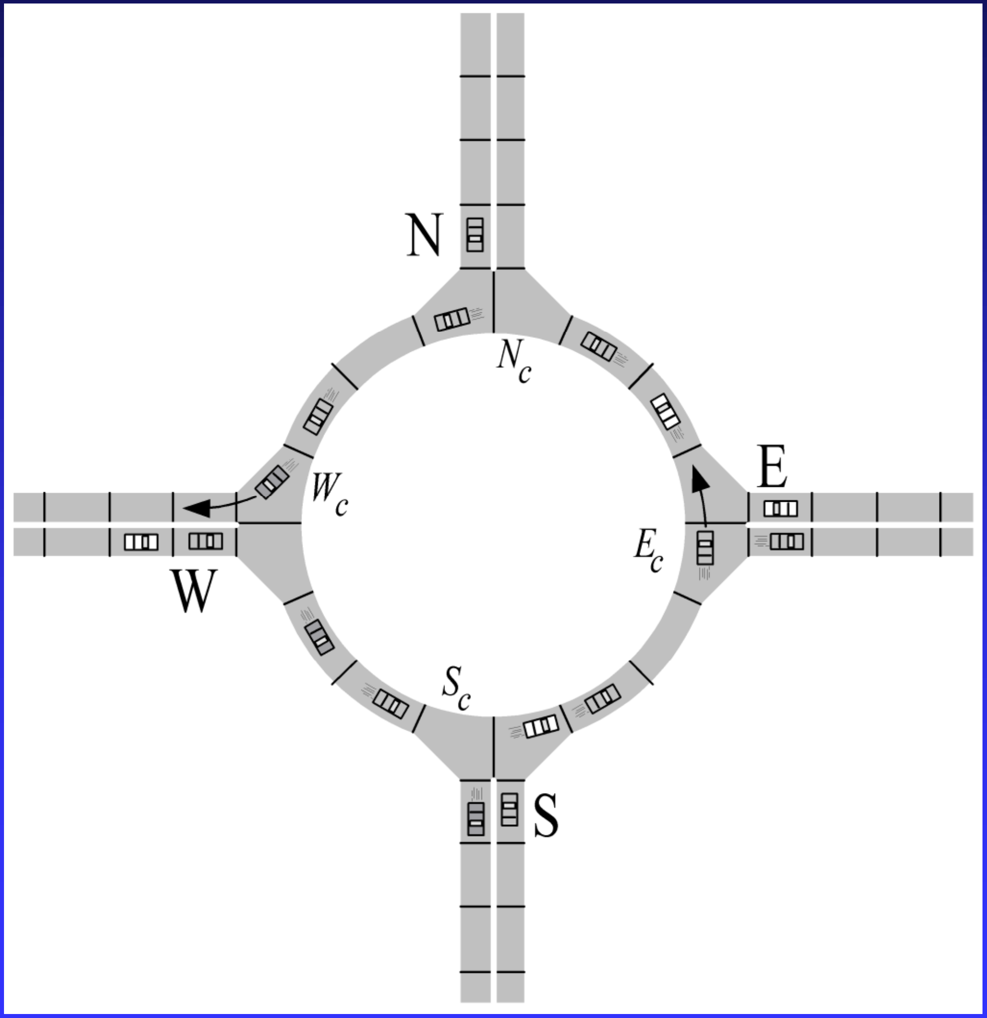
Implementation Idea

Times Spaces Are Available

101.2	142.1	70.3	91.7	223.1

Current Time	Car Count	Cars Rejected
64.2	15	2

Traffic Circle



Traffic Circle Probabilities

	F
N	0.33
E	0.50
S	0.25
W	0.33

D	N	E	S	W
N	0.1	0.2	0.5	0.2
E	0.3	0.1	0.2	0.4
S	0.5	0.3	0.1	0.1
W	0.2	0.4	0.3	0.1

Traffic Circle Data Structures

71 *Iteration*

N	W	S	E	
0	4	8	12	*Offset*
1	0	0	1	*Arrival*
26	23	22	38	*ArrivalCnt*
19	13	11	26	*WaitCnt*
1	2	0	3	*Queue*
37	49	20	41	*QueueAccum*

4	-1	4	-1	8	8	12	0	-1	-1	12	12	8	0	0	0	*Circle*
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

Summary (1/3)

- Applications of Monte Carlo methods
  - Numerical integration
  - Simulation
- Random number generators
  - Linear congruential
  - Lagged Fibonacci

Summary (2/3)

- Parallel random number generators
  - Manager/worker
  - Leapfrog
  - Sequence splitting
  - Independent sequences
- Non-uniform distributions
  - Analytical transformations
  - Box-Muller transformation
  - Rejection method

Summary (3/3)

- Concepts revealed in case studies
  - Monte Carlo time
  - Random walk
  - Metropolis algorithm
  - Simulated annealing
  - Modeling queues