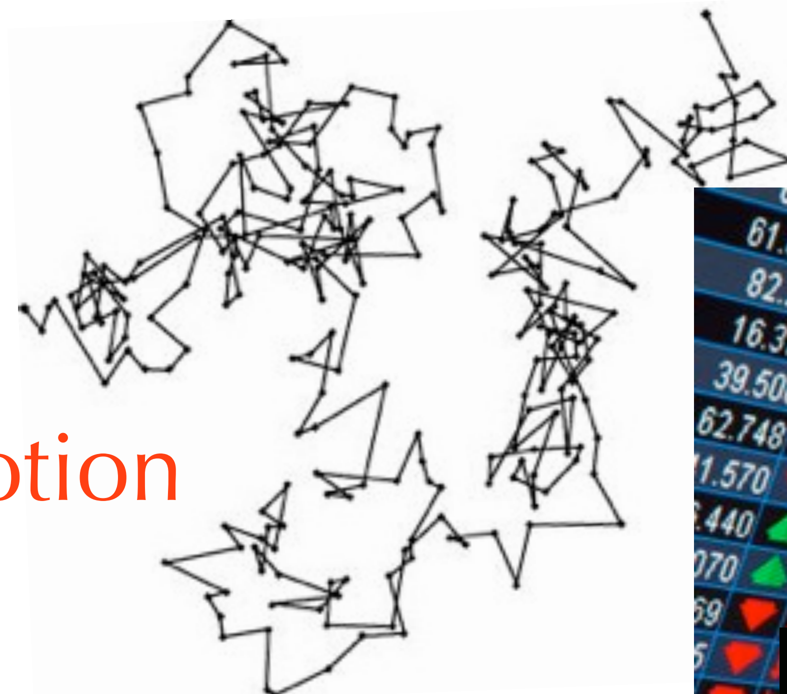
A Brief Introduction to Randomized Algorithms

Outline

- Randomized Algorithms
  - Las Vegas Algorithms
  - Monte Carlo Algorithms
- Probabilistic Analysis
- Selected Topics:
  - Estimation by random sampling
  - Randomized approximation algorithms
  - De-randomization

Randomness



Brownian motion

Stock market

Lottery

Gambling

Randomness

- Uncertainty
  - No structure: Avoid worst case
  - Hard to predict: Cryptography
  - Hard to analyze
- No bias
  - Trading correctness for performance
  - Worst cases may depend on randomness
  - Provide fairness: Estimation by sampling
  - Avoid collisions and deadlocks

Polynomial Identities

- Consider 2 polynomial F(x) and G(x)
  - F(x) is in the product form: $F(x)=(x-r_1)(x-r_2)\ldots(x-r_d)$
  - G(x) is in the canonical form: $G(x)=x^d+c_{d-1}x^{d-1}+\ldots c_1x+c_0$
  - Given x, evaluate F(x) and G(x): $O(d)$
- Question: Is F(x)=G(x)?

Examples

- Is $F_1(x)=G_1(x)$?
  - $F_1(x)=(x-1)(x+1)(x-2)$
  - $G_1(x)=x^3-2x^2-x+2$
- Is $F_2(x)=G_2(x)$?
  - $F_2(x)=(x-1)(x+1)(x-3)$
  - $G_2(x)=x^3+3x^2-x-3$
- Hint: A non-zero polynomial of degree d has d roots.

Deterministic Algorithm 1

- Convert F(x) into the canonical form
- How fast can we do this?
  - $O(d^2)$ multiplications & comparisons
- Pro: a straightforward method
- Con: $O(d^2)$ multiplications are required.
- Can we do better?

Deterministic Algorithm 2

- $H(x)=F(x)-G(x)$: Evaluating $H(x)$ takes $O(d)$
- $F(x)=G(x)$ iff $H(x)=0$.
- Test if $H(0)=H(1)=...=H(d)=0$. (why?)
- If $F(x)=G(x)$, this runs in $O(d^2)$.
- If $F(x)\neq G(x)$, this runs in $\Omega(d)$ and $O(d^2)$.
  - Lucky case: $H(0)\neq 0$
  - Unlucky case: $H(0)=...=H(d-1)=0\neq H(d)$

Randomized Algorithm 1

- Randomly permute $0,\ldots,d$ into $p_0,\ldots,p_d$.
- Perform the check as deterministic algorithm 2.
- We will not always encounter the worst case if $F(x) \neq G(x)$.
- But the worst case still runs in $O(d^2)$.

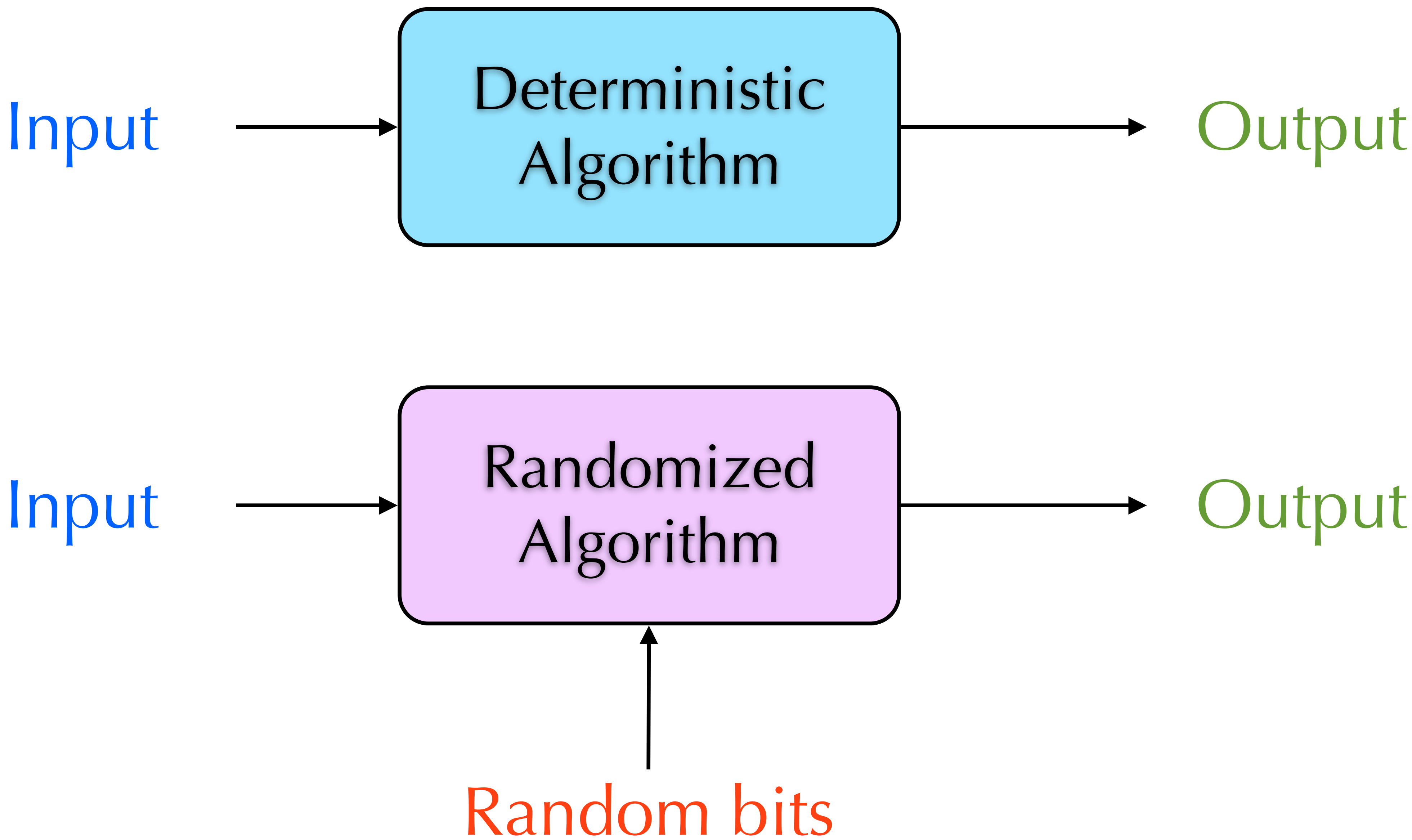
Randomized Algorithm 2

- Randomly sample d+1 distinct numbers $p_0,\ldots,p_d$ from $\{1,\ldots,100d\}$.
- Check if $H(p_0)=H(p_1)=\ldots=H(p_d)=0$.
- Worst case: still $O(d^2)$
- But for $F(x)\neq G(x)$, we can answer no after testing $H(p_0)$ with ≥99% probability.
  - H(x) has only d roots!
  - The expected running time is O(d).

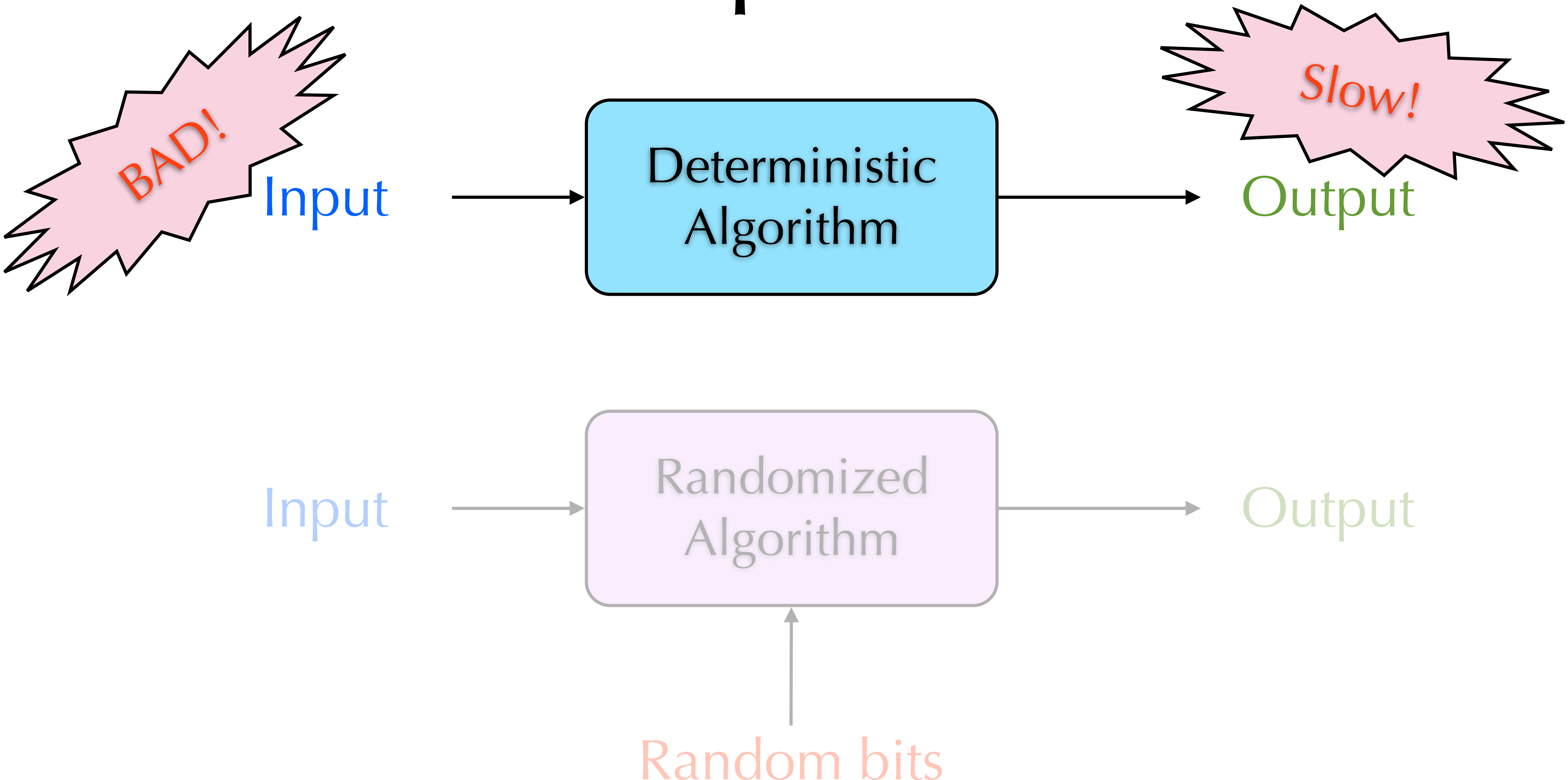
Randomized Algorithm 3

- Randomly sample a number p from {1,...,100d}.
- Output F(x)=G(x) iff H(p)=0.
- Worst case: O(d)
- For F(x)=G(x), we always answer correctly.
- For F(x)≠G(x), we answer correctly with ≥99% probability. Note: probably wrong

Comparison

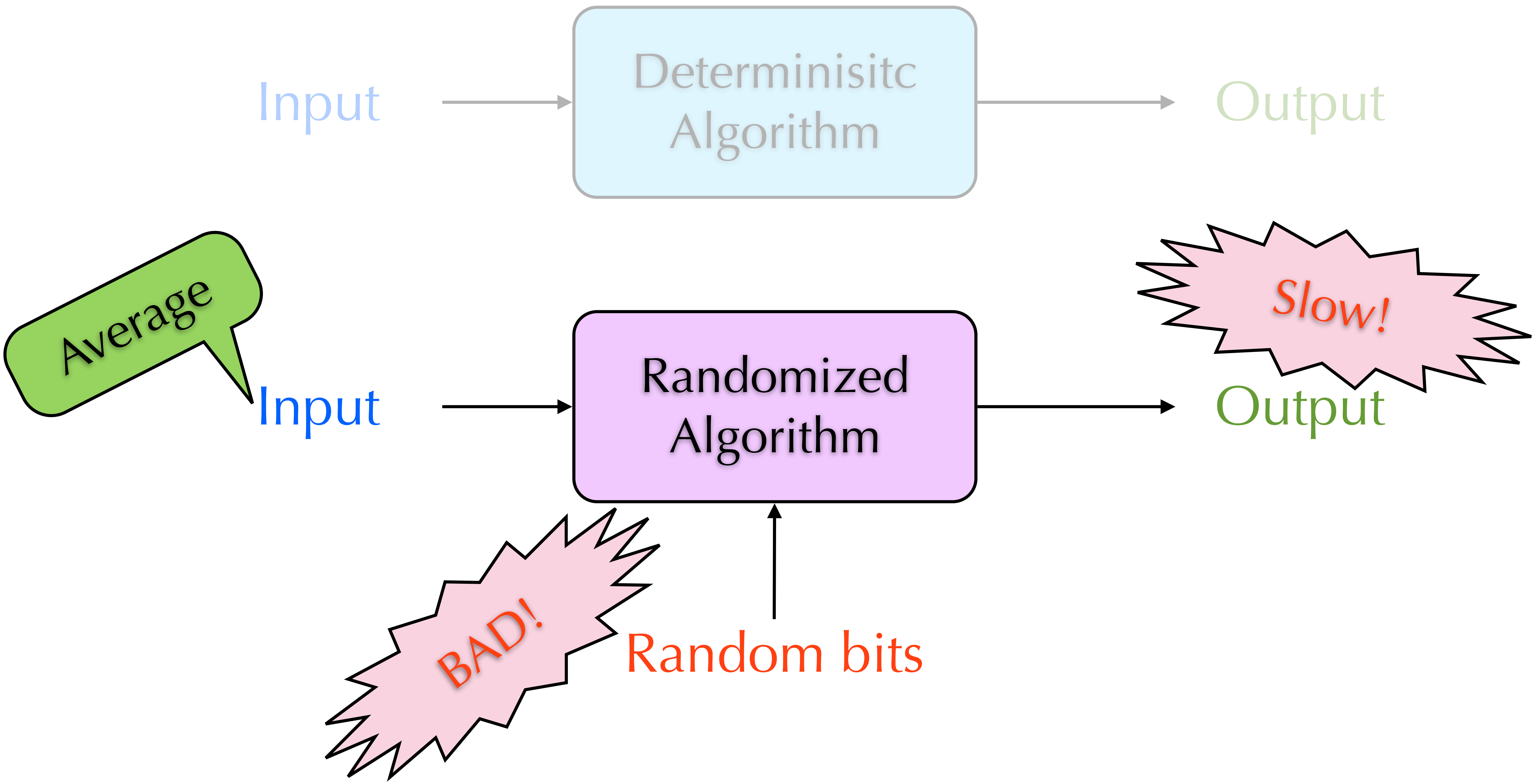

Input
Deterministic Algorithm
Output
Input
Randomized Algorithm
Output
Random bits

Comparison

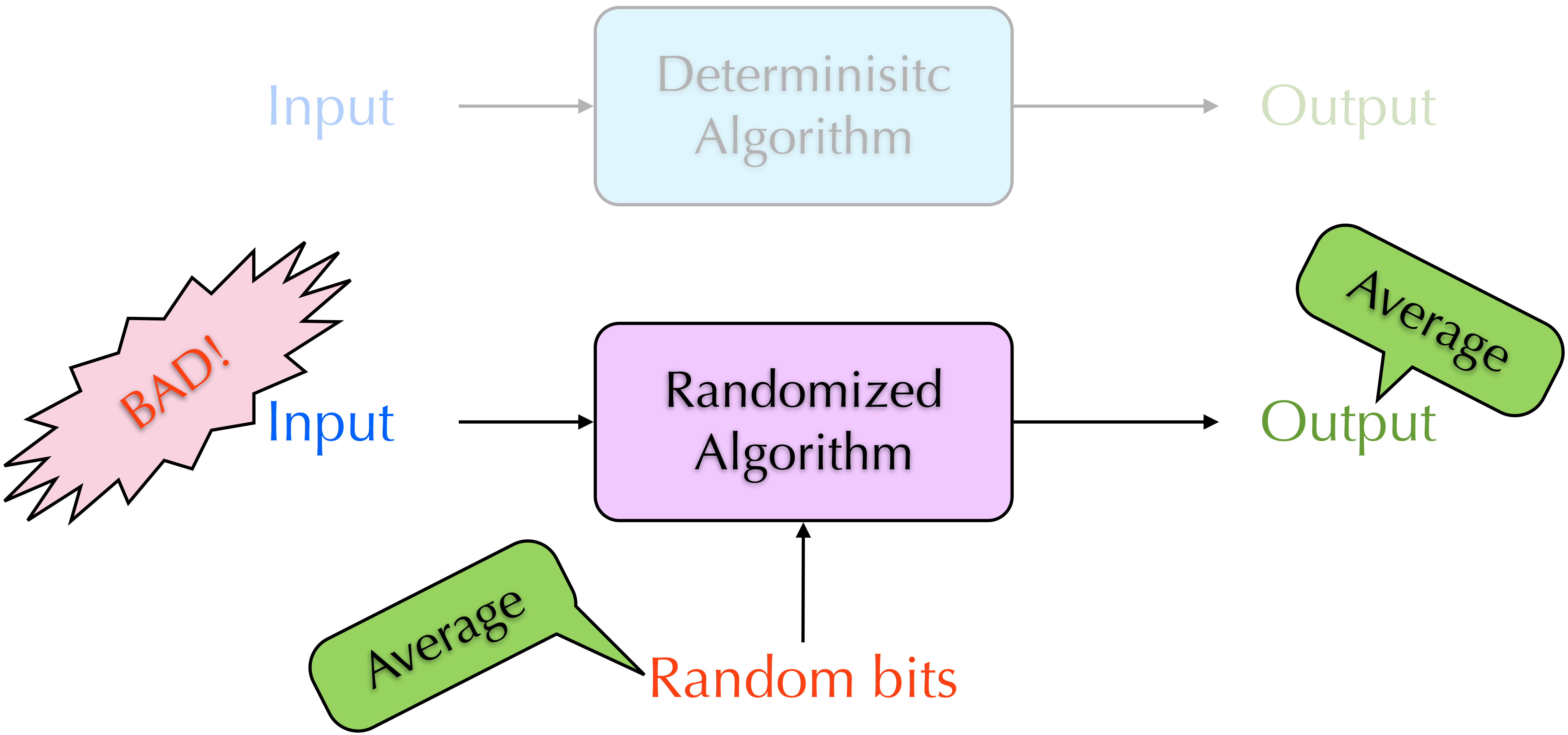


BAD!
Input
Deterministic Algorithm
Output
Slow!
Input
Randomized Algorithm
Output
Random bits

Comparison

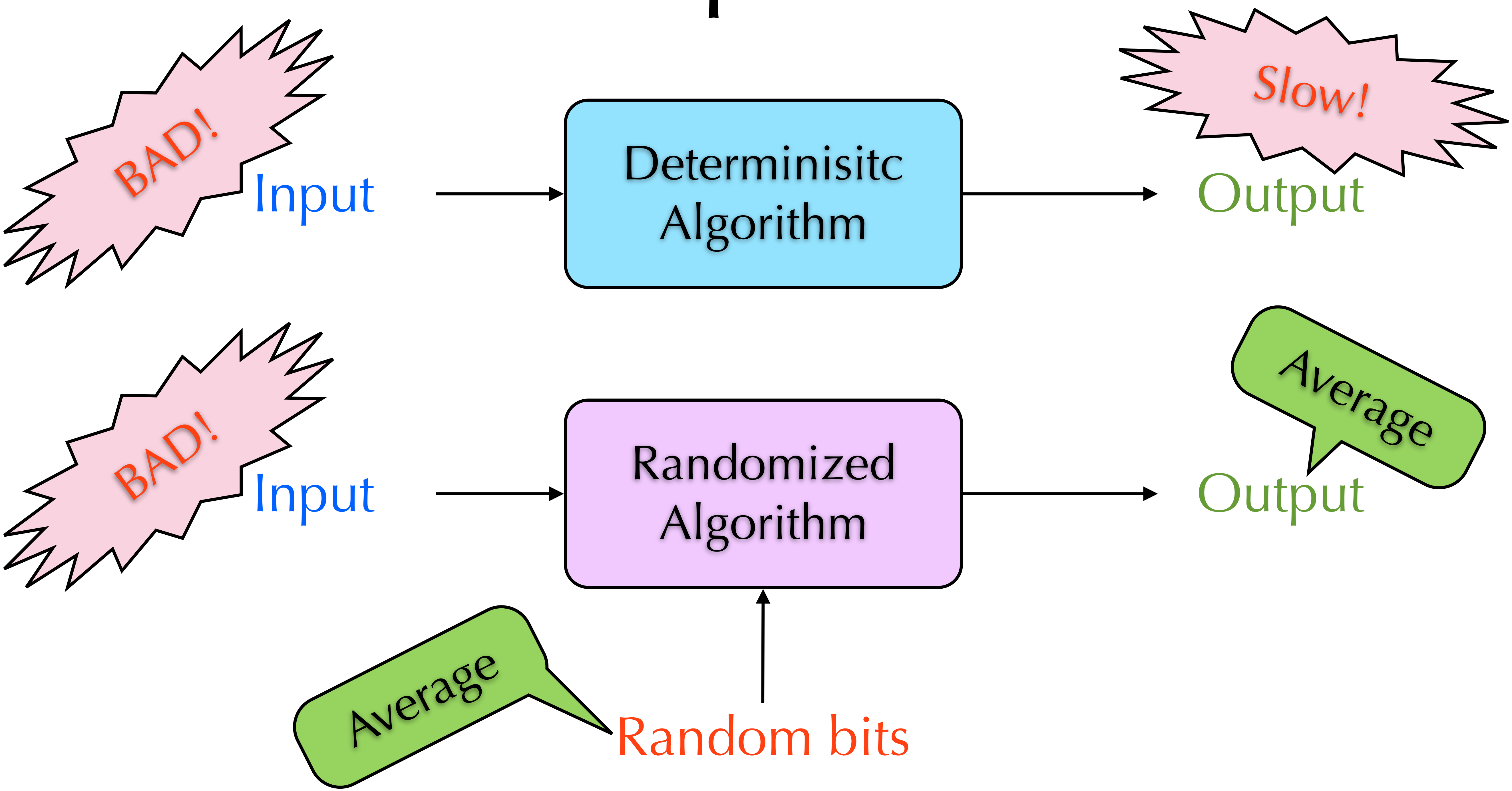

Input
Determinisitc Algorithm
Output
Average
Input
Randomized Algorithm
Output
Slow!
BAD!
Random bits

Comparison

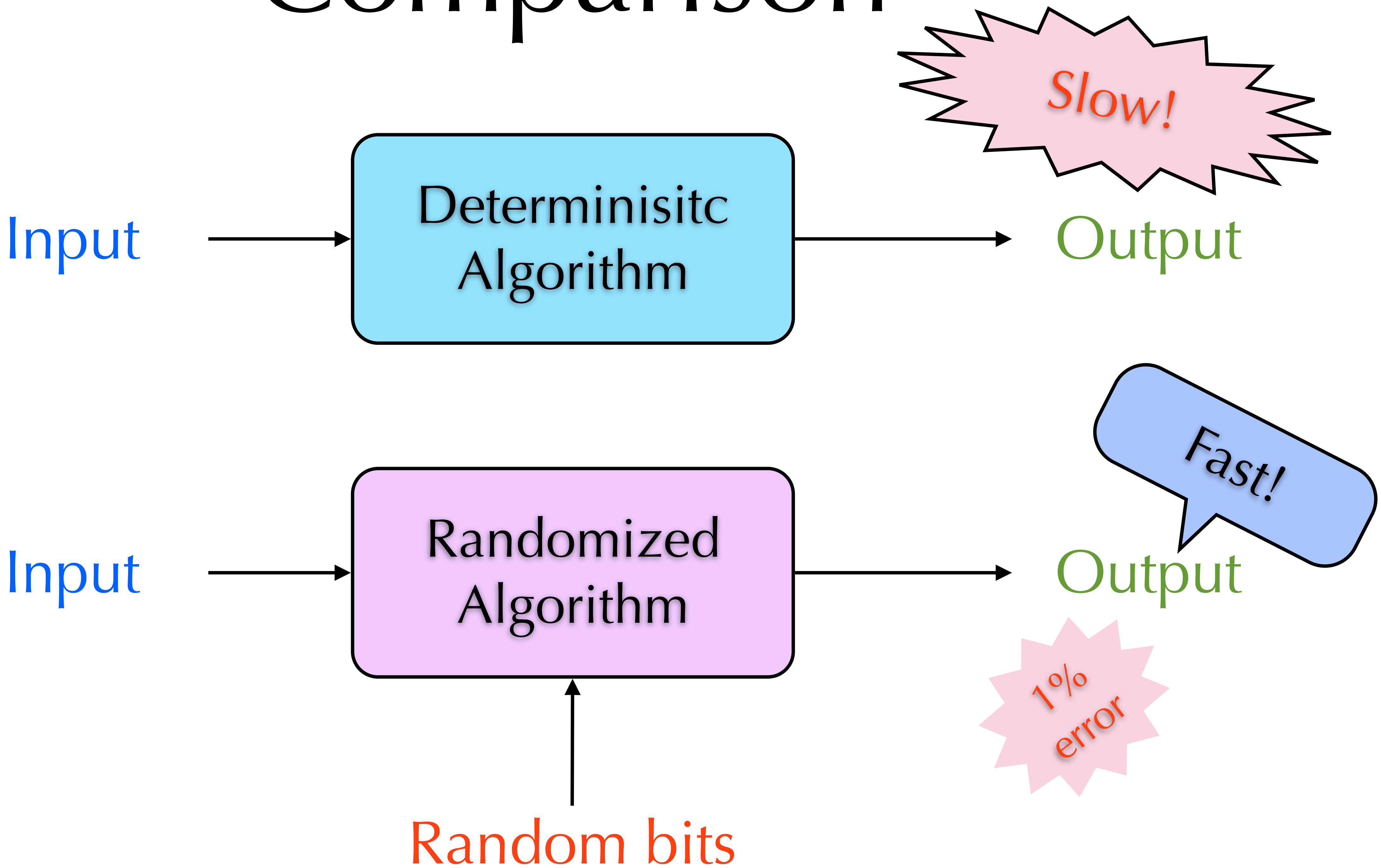


Input
Determinisitc Algorithm
Output
BAD!
Input
Randomized Algorithm
Average
Output
Average
Random bits

Comparison


BAD!
Input
Determinisitc
Algorithm
Slow!
Output
BAD!
Input
Randomized
Algorithm
Average
Output
Average
Random bits

Comparison


Slow!
Input
Determinisitc Algorithm
Output
Fast!
Input
Randomized Algorithm
Output
1% error
Random bits

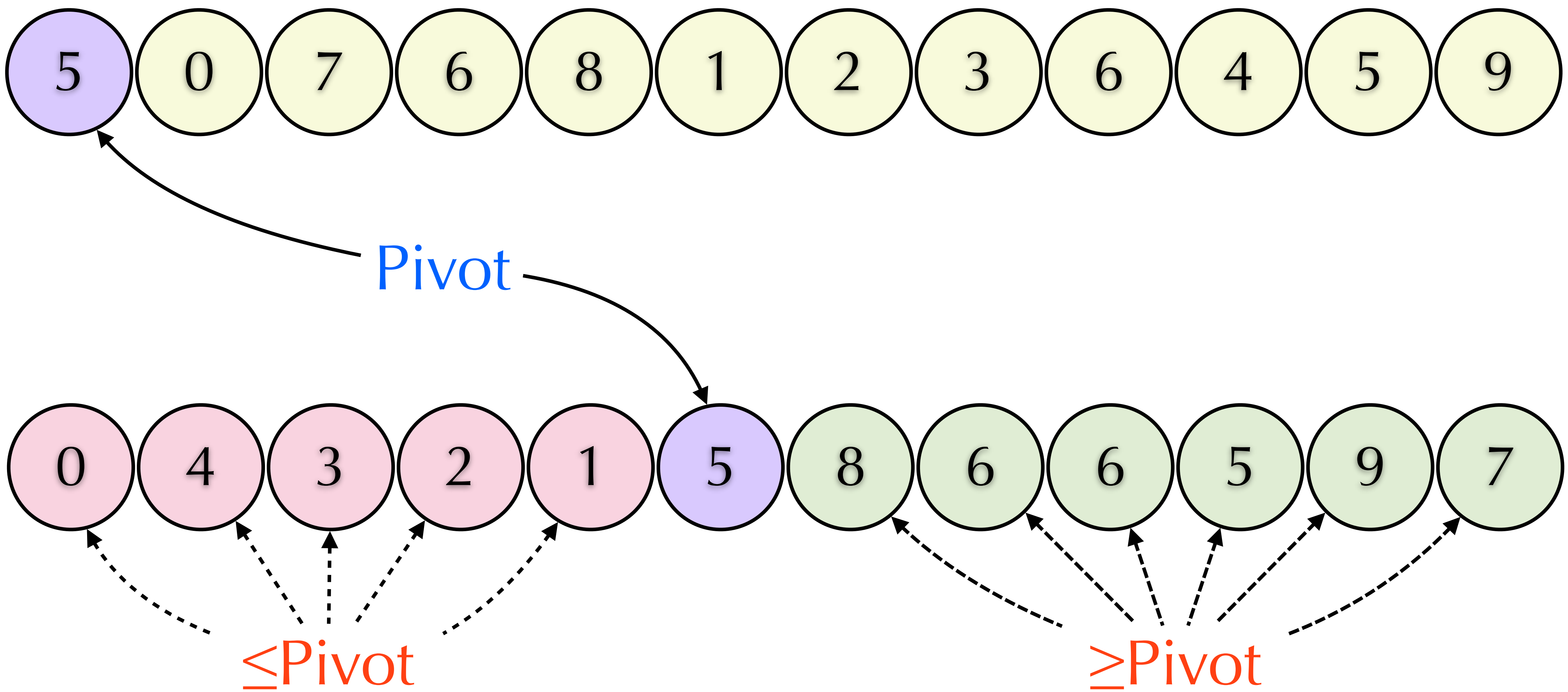
Las Vegas Algorithms

- Randomized algorithms those
  - always output correct answers
  - are fast in average, but they have a small chance to encounter the worst case.
- Example:
  - Randomized algorithm 2 for polynomial identities.
  - Randomized quicksort

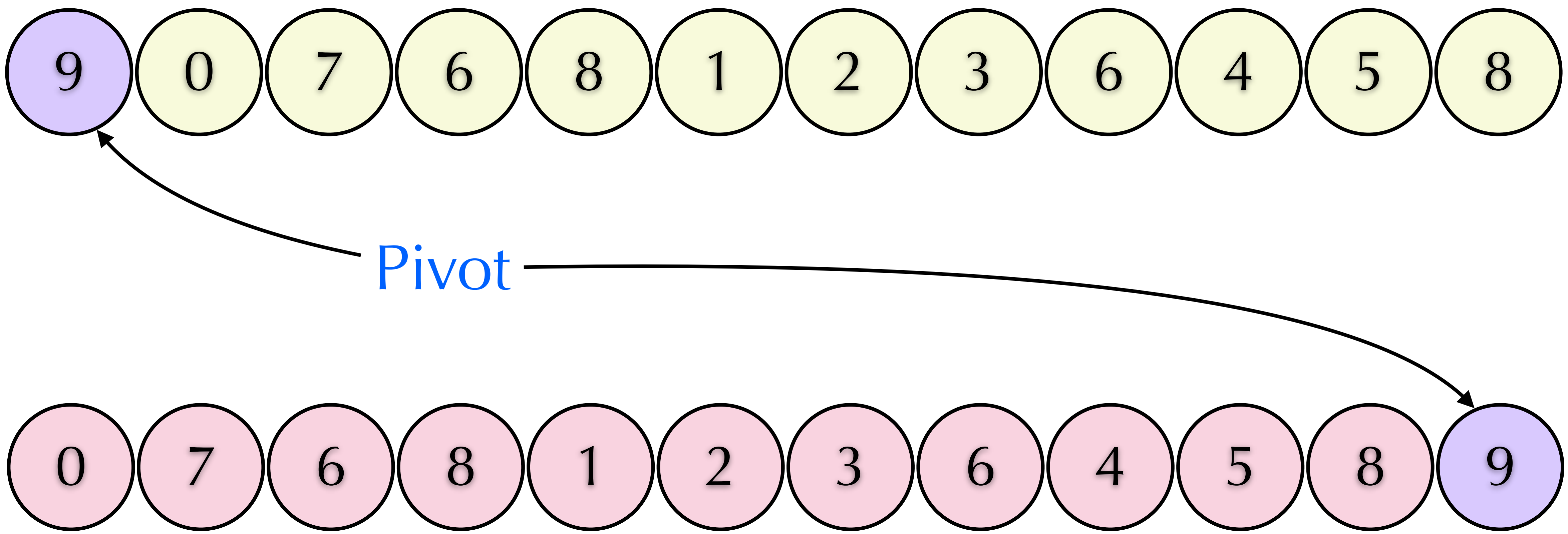
Randomized Quicksort

- Ordinary quicksort has worst case $O(n^2)$.
  - When sorting a sorted array, it is always worst.
- Partition: If the pivot is the max or the min element, the ordinary quicksort encounters the worst case.
- Randomized quicksort: pick the pivot at random.

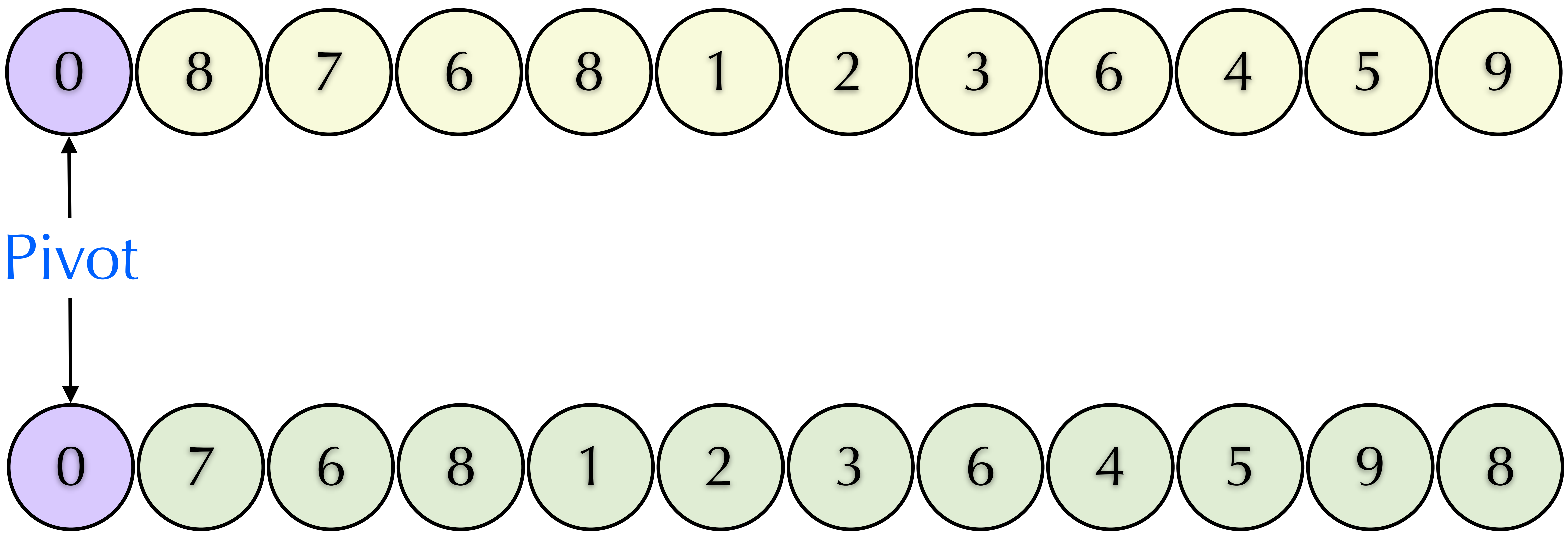
Partition



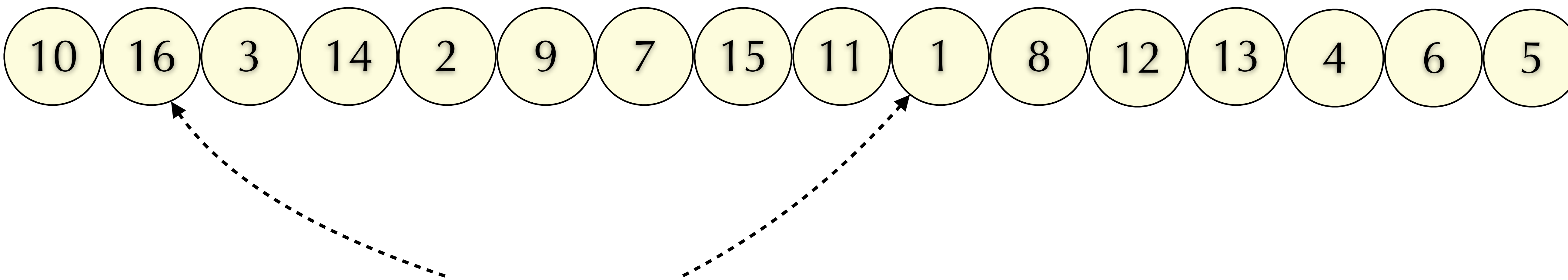
Partition



Partition



Randomized Quicksort



Worst pivot choice Chosen with prob. 2/n

Worst case = n–1 consecutive worst pivot choices!

Monte Carlo Algorithm

- Randomized algorithms those
  - are always "fast"
  - have a relative small chance (≤1/3) to output wrong answer in the worst case.
- Example:
  - Randomized algorithm 3 for polynomial identities.
  - Matrix product verification

Matrix Product Verification

- Given n-by-n matrices A, B, and C, to determine whether AB=C?
- Straightforward method: multiply A and B, then check whether the product is C.
  - Time: $O(n^3)$ by standard multiplication.
  - Fastest known matrix multiplication: $O(n^{2.376})$ by Coppersmith-Winograd

Matrix Product Verification

- Pick a random 0-1 n-dimensional vector v.
- Compute A(Bv) and Cv.
- Check whether A(Bv)=Cv.
- Time: $O(n^2)$. Multiplying a n-by-n matrix and a n-dimensional vector is $O(n^2)$.
- If AB=C, then correct prob. is 1.
- If $AB \neq C$, then correct prob. is at least 0.5.

Example

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, C = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\underbrace{\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}}_{\text{A}}\underbrace{\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}}_{\text{B}}\underbrace{\begin{pmatrix} 1 \\ 0 \end{pmatrix}}_{\text{v}} = \underbrace{\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}}_{\text{A}}\underbrace{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}_{\text{Bv}} = \underbrace{\boxed{\begin{pmatrix} 2 \\ 1 \end{pmatrix}}}_{\text{ABv}}, \underbrace{\begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}}_{\text{C}}\underbrace{\begin{pmatrix} 1 \\ 0 \end{pmatrix}}_{\text{v}} = \underbrace{\boxed{\begin{pmatrix} 2 \\ 1 \end{pmatrix}}}_{\text{Cv}}$$

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \boxed{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}, \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \boxed{\begin{pmatrix} 1 \\ 0 \end{pmatrix}}$$

Boosting the correctness

- Is the error probability too high (>1/3)?
  - By repeating the algorithm 100 times, the error probability decreases exponentially to 2^{-100}!
- The running time dramatically drops to $O(n^2)$!

Complexity Classes

- P: Deterministic polynomial time
- NP: Non-deterministic polynomial time
- Randomized algorithm
  - RP: Only has false positive (No is no)
  - coRP: Only has false negative (Yes is yes)
  - BPP: Two-sided error $\leq 1/3$
  - PP: Two-sided error $<1/2$
  - ZPP: Correct answer or unknown

Complexity Classes

- P: Deterministic polynomial time
- NP: Non-deterministic polynomial time
- Randomized algorithm

  Monte Carlo

  - RP: Only has false positive (No is no)
  - coRP: Only has false negative (Yes is yes)
  - BPP: Two-sided error ≤ 1/3
  - PP: Two-sided error <1/2

  - ZPP: Correct answer or unknown

  Las Vegas

Pseudorandomness

- How can you get random bits?
- Bad new: no truly random bits from computation (ALU)
- You can get some from timer, radio noise, tossing coins, the great nature, etc.
  - But the amount is usually not enough.
- Pseudorandom: not random, but appears to be random.

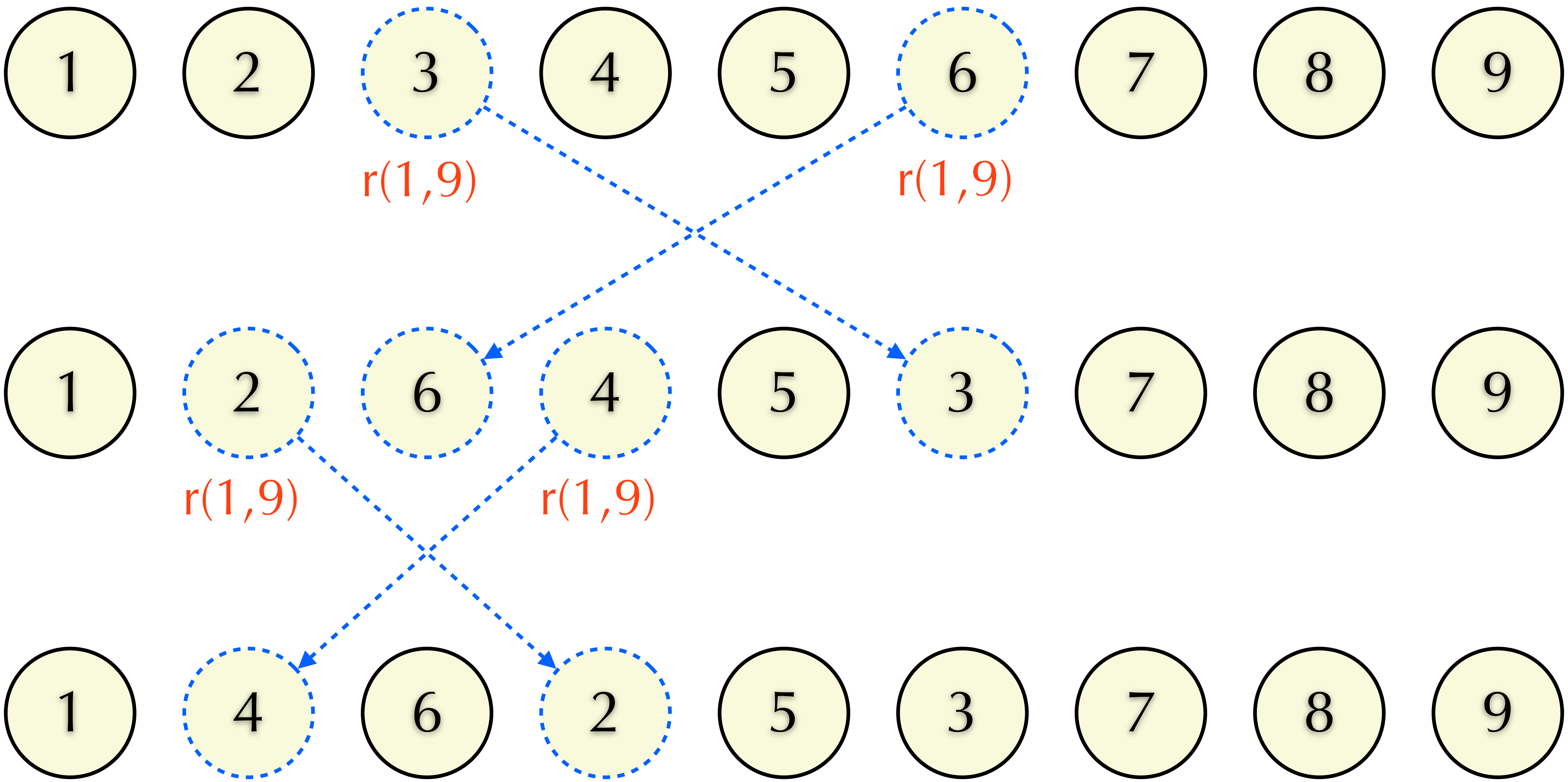
Pseudo Random Number Generator

- Generate numbers which are "like random numbers" but using no truly random bits
- Linear congruential generator
  - $X_{n+1}=aX_n+b \bmod c$
  - Easy to compute and widely applied.
  - When you pick a number from 0,…,n−1 by LCG, please use n*(X/c). X%n may be very regular. Not secure! Unsuitable for cryptography

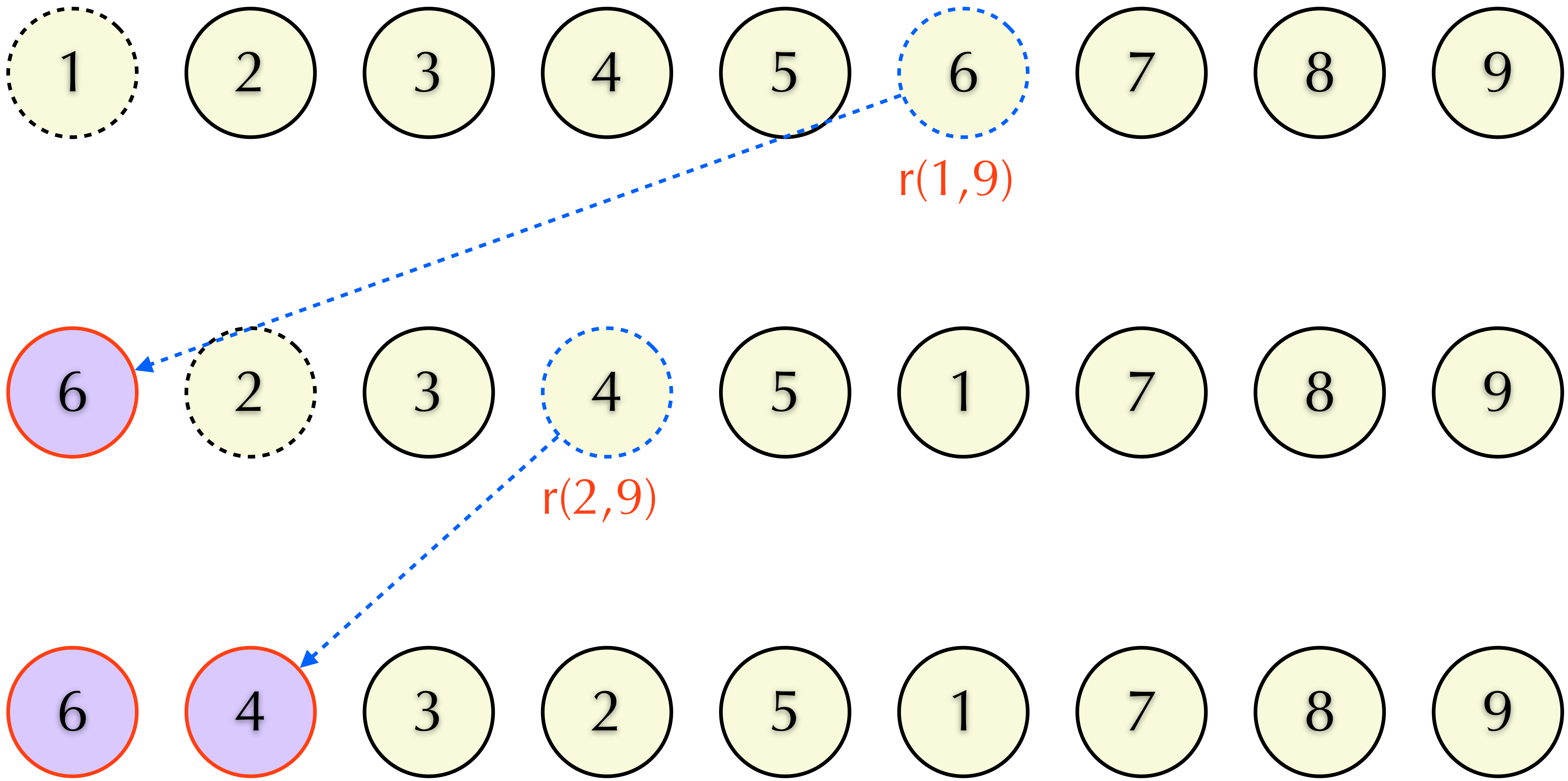
Generating Random Permutation

- How to generate a random permutation of 1,...,n?
- r(L,U): a random integer pick from [L,U].
- Swap(i,j): Exchange the i^{th} and j^{th} positions.
- Strategy 1:
  Perform n−1 times Swap(r(1,n), r(1,n))
- Strategy 2:
  for i=1 to n−1 do Swap(i,r(i,n))

Strategy 1



Strategy 2



Generating Random Permutation

- Both strategies can generate any possible permutation.
- Which is more uniform?
  - Strategy 1: n^{2n} possible results. (NOT A MULTIPLE of n!)
  - Strategy 2: n! possible results.
- If you need a random permutation, please use Strategy 2.

Probability Notations

- Sample space Ω: set of possible samples
- Probability distribution:
  - A function: $\Omega \to [0,1]$
  - $\Sigma_{\sigma\in\Omega}\Pr[\sigma]=1$
- Random variable X: $\Omega \to R$. A function maps samples into reals.

Indicator

- An event is a subset of the sample space Ω.
- Indicator random variable I[event]
  - If event happens, then I[event]=1. Otherwise I[event]=0.
- $\Pr[\text{event}] = \Sigma_{\sigma \in \text{event}} \Pr[\sigma]$

Expected Value

- Expected value of random variable X: $E[X]=\Sigma_x x\Pr[X=x]$
- Linearity: $E[X+Y]=E[X]+E[Y]$
- Expectation value of indicators: $E[I[e]]=1\times\Pr[I[e]=1]=\Pr[e]$
- Tossing a biased coin (head with prob. p) until we get a head. Let X be the number of tosses, what is E[X]?

Solution

- $E[X]=1\times p+(1+E[X])\times(1-p)$
- $pE[X]=1$
- $E[X]=1/p$
- On average, we need $1/p$ tosses to get a head.

Coupon Collection

- You get a coupon if you buy a cup of tea.
- There are n different kinds of coupons, and you can win the price if you have all n kinds.
- Assume any kind of coupons appears with probability 1/n.
- How many cups of tea are expected to buy if you want to win the price?

Solution

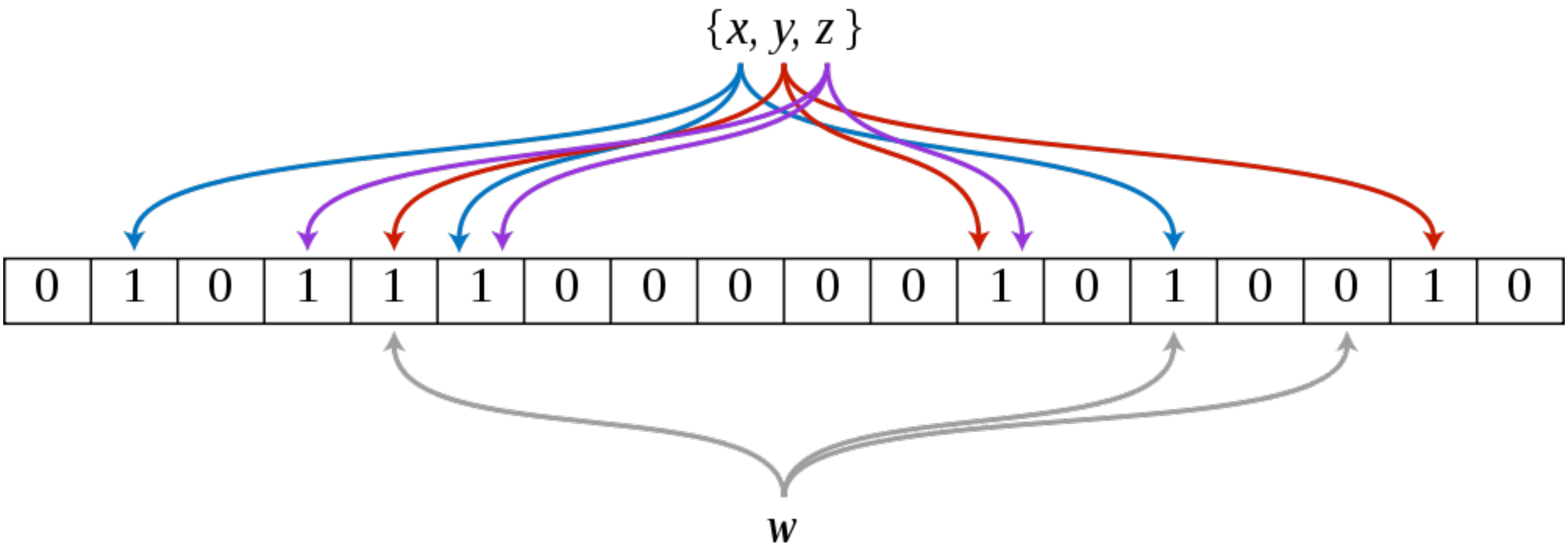
- Let X_i be the number of cups for collect the i^{th} kinds of coupon.
  - $E[X_i]=n/(n+1-i)$ (why?)
- $Ans=X_1+\ldots+X_n$
- $E[Ans]=E[X_1]+\ldots+E[X_n]$
  $=(n/n)+(n/(n-1))+\ldots+(n/1)$
  $=n(1+(1/2)+(1/3)+\ldots+(1/n))$
  $=O(n\log n)$

Hash Table

- Using a hash function to balance the load.
  - Pseudorandom function can be a hash function.
- In C++11, there are hash_map and hash_set in STL.
- O(n) space for storing n elements.
- What if we can tolerate some false positive when implementing a set?

Bloom Filters

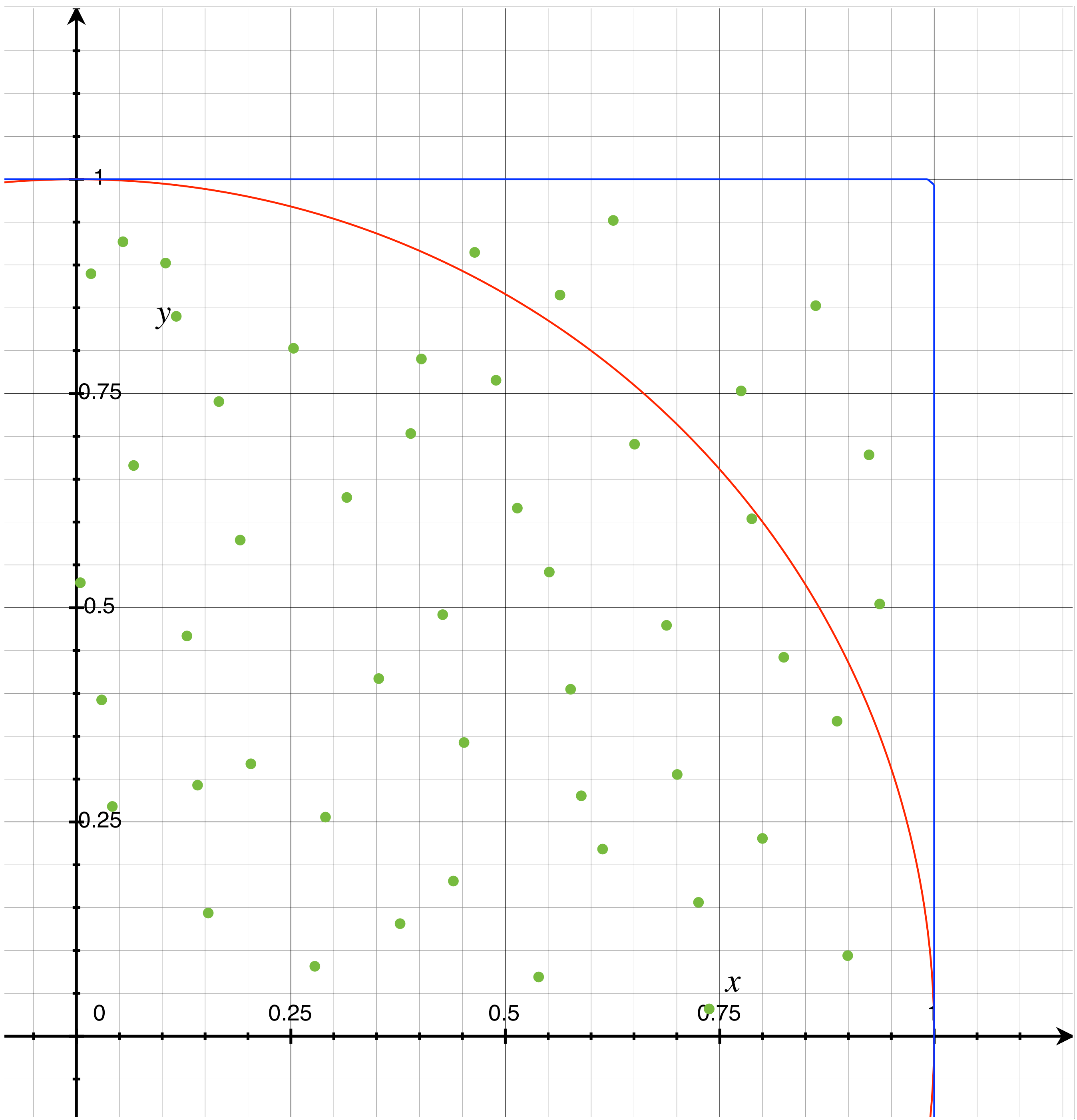
- Use multiple hash functions.
- n items by m bits with error $e^{-\frac{m}{n}(\ln 2)^2}$



Estimation by Random Sampling

- Ex: Computing π by Random Sampling
- Generate n samples $(x_1,y_1),\ldots,(x_n,y_n)$ by picking x_i,y_i from [0,1] uniformly at random.
- Let m be the number of samples satisfying $x^2+y^2\leq 1$.
- $\pi=4m/n$

Estimation by Random Sampling



- The sample point fall into the accepting region with probability π/4.
- The expected number m of samples falling into the region is nπ/4.
- π=4m/n

Optimization Problem

- A problem has many feasible solution.
- Finding “best” solution from all feasible solutions.
- Natural version:
  - Finding the sweetest melon from all kinds of melons.
- Sometimes it can be very hard to find the optimal solution.

Approximation

- Sometimes it is hard to find the optimum but easy to find good suboptimal solution.
  - In order to get 100 in the final, you have to spend ≥10 hours.
  - But you'll get 80 just by spending 5 minutes. (In some schools, 80=GPA 4.0)
  - Why don't you just spend 5 minutes?
  - Note: this is just an example. NOT your final
- Use randomized algorithms!

Max 3CNF-SAT

- Conjunctive Normal Form
  - $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee x_3 \vee x_4)$
- Given a 3CNF boolean formula Φ, find an assignment that satisfies the largest number of clauses.
- It is NP-hard to find out the optimal solution.
  - 3SAT is NP-hard.

Max 3CNF-SAT

- 100% is hard, but 87.5% is simple.
- Consider a random assignment to any clause $c=(x \vee y \vee z)$
  - $\Pr[x=y=z=\text{false}]=0.5^3=0.125$
  - $\Pr[c \text{ is satisfied}]=1-0.125=0.875$
- Note: $0.875+\varepsilon$ is also NP-hard by Håstad.

De-randomization

- Let $\Phi_0=\Phi$. Φ_j is obtained by fixing variables $x_1,\ldots,x_j$ in Φ.
- Y_j be the number of satisfied clauses of Φ_j
- For i = 1 to n do
  - Set x_i=true iff $E[Y_{i-1}|x_i=\text{true}]>E[Y_{i-1}|x_i=\text{false}]$
  - Fix x_i to obtain a new formula Φ_i.

Example

- $\Phi_0=(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee x_3 \vee x_4)$
  - $E[Y_0|x_1=\text{true}]=2.75>E[Y_0|x_1=\text{false}]=2.5$
    - set x_1=true
- $\Phi_1=(T \vee x_2 \vee x_3) \wedge (T \vee \overline{x}_3 \vee \overline{x}_4) \wedge (F \vee x_3 \vee x_4)$
  - $E[Y_1|x_2=\text{true}]=2.75=E[Y_1|x_2=\text{false}]=2.75$
    - set x_2=false

Example

- $\Phi_2=(T \vee F \vee x_3) \wedge (T \vee \overline{x}_3 \vee \overline{x}_4) \wedge (F \vee x_3 \vee x_4)$
  - $E[Y_2|x_3=\text{true}]=3>E[Y_2|x_3=\text{false}]=2.5$
    - set $x_3=\text{true}$
- $\Phi_3=(T \vee F \vee T) \wedge (T \vee F \vee \overline{x}_4) \wedge (F \vee T \vee x_4)$
  - $E[Y_3|x_4=\text{true}]=3=E[Y_3|x_4=\text{false}]=3$
    - set $x_4=\text{false}$
- $\Phi_4=(T \vee F \vee T) \wedge (T \vee F \vee T) \wedge (F \vee T \vee F)$
  - $(x_1,x_2,x_3,x_4)=(\text{true},\text{false},\text{true},\text{false})$