

# Efficient Algorithms

Ekkapot Charoenwanit

Software Systems Engineering

TGGS

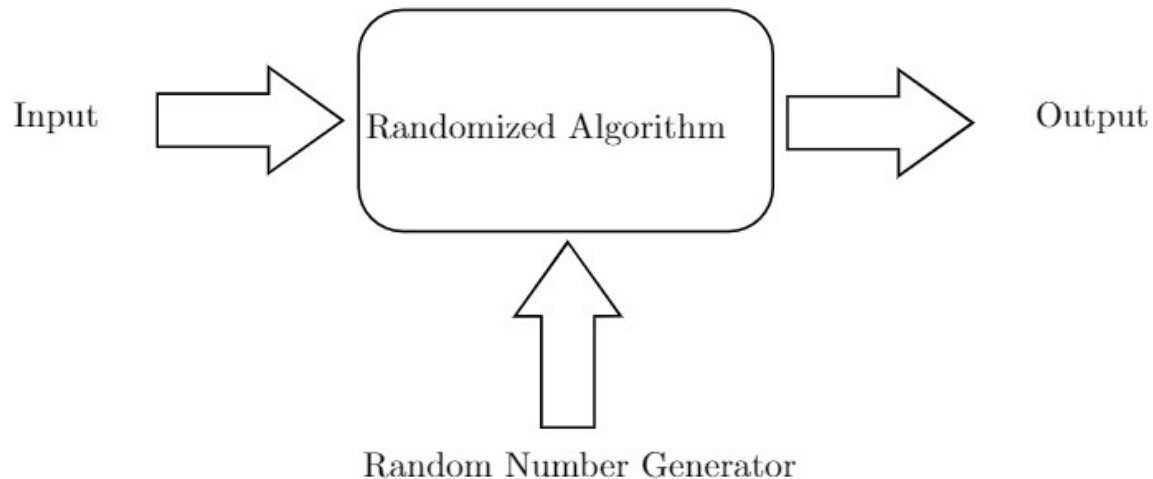
KMUTNB

# Lecture 15: Randomized Algorithms

- Monte Carlo
- Las Vegas

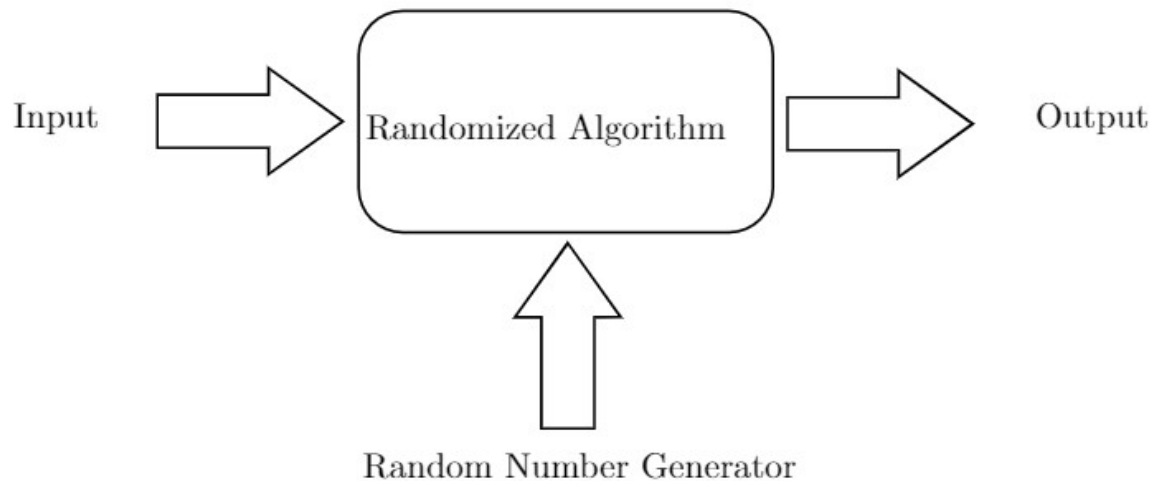
# Randomized Algorithms

**Randomized algorithms** make use of **randomness** through a **random number generator (RNG)** in an attempt to improve the performance over worst-case performance of **deterministic algorithms**.



# Randomized Algorithms

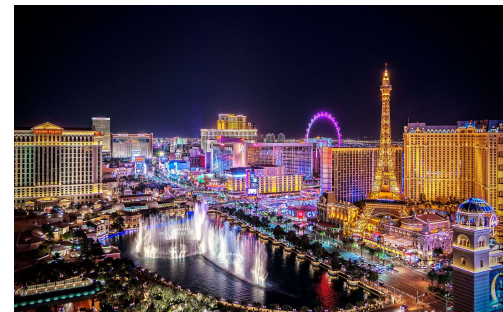
From the diagram, on **two occasions** where a randomized algorithm works on exactly **the same input**, the behavior of the algorithm may be **inconsistent** due to the **randomness** introduced by the **random number generator**.



# Monte Carlo and Las Vegas

A randomized algorithm is a

- **Monte Carlo** randomized algorithm if
  - It returns a result which is probably correct
  - it always return a result
  - its running time is always deterministically bounded
- **Las Vegas** randomized algorithm if
  - it always returns the correct result
  - Its running is bounded in expectation



# Monte Carlo

A **Monte Carlo** randomized algorithm is  $p$ -correct if and only if

$$\Pr\{\textit{Correct Result}\} \geq p$$

For decision problems,

- **One-sided error** algorithms only makes errors in one direction
- **Two-sided error** algorithms makes errors in both directions

# Monte Carlo

We can **amplify** the correctness of a ***p*-correct, one-sided error** algorithm by **repeatedly** running the algorithm multiple times.

- This process of repeatedly running a ***p*-correct, one-sided error** randomized algorithm multiple times is known as **amplification**.

<b>#Steps</b>	<b><i>Pr</i>{Incorrect}</b>	<b><i>Pr</i>{Correct}</b>
1	$1 - p$	$p$
2	$(1 - p)^2$	$1 - (1 - p)^2$
$k$	$(1 - p)^k$	$1 - (1 - p)^k$

# Majority

Given an array  $d$  of number  $n$ , determine whether there are elements that appear more than  $\frac{n}{2}$  times.

**Case I:** If  $d$  has no majority, the randomized algorithm always returns the correct result: **False**

**Case II:** If  $d$  has a majority, the randomized algorithm may return the incorrect result: **False Negative**

---

```
1: procedure RANDOMIZED-MAJORITY( $d[1..n]$ )
2:    $i = \text{RANDOM}(0, 1)$ 
3:    $c = \text{COUNT}(d, i)$ 
4:   return ( $c > n/2$ )
```

---



# Majority: Analysis

If there is, in fact, a majority in the array  $d$ , the probability that the algorithm

- picks a majority is **at least**  $\frac{1}{2}$
- does not pick a majority is **at most**  $\frac{1}{2}$

Thus, the randomized algorithm is  $\frac{1}{2}$ -correct randomized algorithm.

---

```
1: procedure RANDOMIZED-MAJORITY( $d[1...n]$ )
2:    $i = \text{RANDOM}(0, 1)$ 
3:    $c = \text{COUNT}(d, i)$ 
4:   return ( $c > n/2$ )
```

---

# Majority: Analysis

**Corollary 1:** The randomized algorithm is a  $(1 - 2^{-k})$ -correct algorithm if it repeats for  $k$  times.

**Proof:**

The probability that one run of the randomized algorithm reports the incorrect result is **at most**  $\frac{1}{2}$ .

Since each run is independent of each other, if the algorithm runs for  $k$  times, the probability that the algorithm reports a false negative is **at most**  $2^{-k}$ .

Thus, the probability that the algorithm returns the correct result is **at least**  $1 - 2^{-k}$ . ■

---

```
1: procedure RANDOMIZED-REPEATED-MAJORITY( $d[1..n], k$ )
2:   for  $i = 1; i \leq k; i++$  do
3:     if MAJORITY( $d$ ) then
4:       return True
5:   return False
```

---

## 2-SAT

**2-SAT** is a computational problem of determining whether a given **2-CNF (Conjunctive-Normal Form)** formula is **satisfiable**.

- A logic formula is **satisfiable** if and only if there is a **truth assignment** that makes the whole formula **true**.
- **2-CNF** formula is a Boolean formula constructed by a **conjunction of clauses**, where each clause is a **disjunction of exactly two literals**.
- **Literals** are **boolean variables** or their **negations**.

**Example:**  $(x_1 + \bar{x}_2)(x_1 + x_3)(\bar{x}_1 + x_2)(\bar{x}_2 + \bar{x}_3)(\bar{x}_2 + \bar{x}_4)(x_1 + \bar{x}_4)$

## 2-SAT

Given a boolean formular  $\varphi(x_1, x_2, \dots, x_n)$  in the **2-CNF** form

- randomly assign values to boolean variables  $x_1, x_2, x_3, \dots, x_n$ 
  - $x[i] = \text{Random}(0,1) \quad \forall i = 1, 2, \dots, n$
- choose an arbitrary clause  $C$  that is not satisfied
- choose uniformly at random one of the literals in  $C$  and toggle the value of its variable
  - $x[k] = 1 - x[k]$

**Example:**  $\varphi = (x_1 + \bar{x}_2)(x_1 + x_3)(\bar{x}_1 + x_2)(\bar{x}_2 + \bar{x}_3)(\bar{x}_2 + \bar{x}_4)(x_1 + \bar{x}_4)$

$x_1,$	$x_2,$	$x_3,$	$x_4$	$(x_1 + \bar{x}_2)(x_1 + x_3)(\bar{x}_1 + x_2)(\bar{x}_2 + \bar{x}_3)(\bar{x}_2 + \bar{x}_4)(x_1 + \bar{x}_4)$
0	1	1	0	(0 0)(0 1)(1 1)(0 0)(0 1)(0 1)
1	1	1	0	(1 0)(1 1)(0 1)(0 0)(0 1)(1 1)
1	0	1	0	(1 1)(1 1)(0 0)(1 0)(1 0)(1 1)
0	0	1	0	(0 1)(0 1)(1 0)(1 0)(1 1)(1 1)

## 2-SAT: Analysis

Given  $\varphi(x_1, x_2, \dots, x_n)$  in **2-CNF** form, suppose there exists a **satisfying truth assignment** for  $\varphi$ :

- Let  $k$  be the number of boolean variables with the correct boolean values
- That is,  $k \in \{0, 1, 2, \dots, n\}$

**Observations:** We shall refer to each time the algorithm changes the truth assignment as **a step**:

- If  $k = 0$  at the current step,  $k = 1$  at the next step.
  - All variables have been assigned with the incorrect values
  - Flipping the value of any variable will increase  $k$  by one
- If  $k > 0$  at the current step,  $k$  will either decrease or increase **by one** at the next step

## 2-SAT: Analysis

The change in value of  $k$  can be thought as a **random walk** on a number line whose starting point is 0 and ending point is  $n$ .

- When  $k = n$ , a truth assignment has been found.

How long will such a random walk take to reach the ending point  $n$ ?

Let  $s_k$  be the number of steps required before reaching  $n$ .

- $s_n = 0$
- $s_0 = 1 + s_1$
- $s_k = 1 + \frac{1}{2}(s_{k-1} + s_{k+1}) : \forall k \in \{1, 2, \dots, n-1\}$

## 2-SAT: Analysis

Solving the system of linear equations, we have

$$s_k = n^2 - k^2$$

When  $k = 0$ ,  $s_0 = n^2 - 0^2 = n^2$ .

In a worst-case scenario, when the initial truth assignment ends up with all the clauses evaluating to **False**, i.e.,  $k = 0$

→ **The expected number of steps** required before arriving at the ending point  $n$  is  $n^2$ .

→ **The expected number of steps** required before arriving at a satisfying truth assignment is  $n^2$ .

## 2-SAT: Analysis

**Case I:** If the algorithm returns **True**, there **certainly** exists a satisfying truth assignment.

**Case II:** If the algorithm returns **False**, one of the following two possibilities holds:

- There is **no** satisfying truth assignment for the given **2-CNF** formula
- There is a satisfying truth assignment but the algorithm gave up and terminated too early



## 2-SAT: Analysis

**Theorem 1:** If there is a satisfying truth assignment for a **2-CNF** formula  $\varphi(x_1, x_2, \dots, x_n)$ , if the algorithm carries out a random walk for  $2n^2$  steps and terminates, the probability that the algorithm returns a **false negative** result is at most  $1/2$ . That is, the algorithm is a  $1/2$ -correct Monte Carlo algorithm.

**Proof:**

**Markov's Inequality:**  $\Pr\{X \geq a\} \leq \frac{E[X]}{a}$

Let  $X$  be a random variable for the number of steps in a random walk before reaching a satisfying truth assignment.

$$\begin{aligned} E[X] &= n^2 && [\text{Solving recurrence } S(k)] \\ \Pr\{X \geq 2n^2\} &\leq \frac{n^2}{2n^2} \\ &= \frac{1}{2} \end{aligned}$$

■

## 2-SAT: Analysis

**Corollary II:** The randomized algorithm is a  $(1 - 2^{-k})$ -correct algorithm if it repeats for  $k$  times.

**Proof:** By **Theorem I**, the algorithm is a  $1/2$ -correct algorithm.

Each time the algorithm runs and returns false, the probability that the algorithm reports a false negative is **at most**  $1/2$ .

Since each run is independent of each other, if the algorithm runs for  $k$  times, the probability that the algorithm reports a false negative is **at most**  $2^{-k}$ .

Thus, the probability that the algorithm returns the correct result is **at least**  $1 - 2^{-k}$ . ■

# Freivalds' algorithm

Given three  $n \times n$  matrices  $A$ ,  $B$  and  $C$ , determine whether  $C = AB$ .

The naïve deterministic algorithm takes  $\Theta(n^3)$  time.

- Multiply  $A$  and  $B$  :  $\Theta(n^3)$
- Compare  $AB$  and  $C$ :  $\Theta(n^2)$

A more sophisticated algorithm takes  $\Theta(n^{2.8074})$  time.

- Multiply  $A$  and  $B$  with **Strassen algorithm** :  $\Theta(n^{2.8074})$
- Compare  $AB$  and  $C$ :  $\Theta(n^2)$

# Freivalds' algorithm: Analysis

Let  $D = AB - C$

and

$r$  be a  $n \times 1$  **binary** vector.

Thus,  $Dr$  is a  $n \times 1$  vector.

Let  $P = Dr$ .

## Observations:

If  $D = 0$ ,  $P = 0$ , no matter what  $r$  is.

If  $D \neq 0$ ,  $P$  may be 0 or may be not.

[**one-sided error**]

# Freivalds' algorithm: Analysis

Freivalds' algorithm:

- Generate a **0/1** binary vector  $r$  at random
- Compute  $P = A(Br) - Cr$
- Return **True** if  $P = (0,0,0 \dots, 0)^T$ ; **False**, otherwise

Let  $p$  denote the probability that the algorithm returns an incorrect result.

- if  $AB = C$ ,  $p = 0$ , which is **independent** of the value of  $r$
- If  $AB \neq C$ , we have that at least one element of  $D$  is non-zero

# Freivalds' algorithm: Analysis

**Case:**  $AB \neq C$

Suppose that  $d_{ij} \neq 0$ .

Since  $P = Dr$ , we have

$$p_i = \sum_{k=1}^n d_{ik} r_k = d_{ij} r_j + \sum_{k \neq j} d_{ik} r_k = d_{ij} r_j + y$$

where  $y = \sum_{k \neq j} d_{ik} r_k$ .

By **Bayes' Theorem**,

$$\Pr\{p_i = 0\} = \Pr\{p_i = 0 | y = 0\} \Pr\{y = 0\} + \Pr\{p_i = 0 | y \neq 0\} \Pr\{y \neq 0\}$$

Since  $d_{ij} \neq 0$ ,  $p_i = 0$  and  $y = 0$  implies  $r_j = 0$

- Since  $\Pr\{r_j = 0\} = \frac{1}{2}$ ,  $\Pr\{p_i = 0 | y = 0\} = \Pr\{r_j = 0\} = \frac{1}{2}$ .

Since  $d_{ij} \neq 0$ ,  $p_i = 0$  and  $y \neq 0$  implies  $r_j \neq 0$ , which immediately implies  $r_j = 1$ , and  $d_{ik} = -y$

- Since  $\Pr\{r_j = 1\} = \frac{1}{2}$ ,  $\Pr\{p_i = 0 | y \neq 0\} = \Pr\{r_j = 1 \wedge d_{ik} = -y\} \leq \Pr\{r_j = 1\} = \frac{1}{2}$ .

# Freivalds' algorithm: Analysis

$$\begin{aligned}\Pr\{p_i = 0\} &\leq \frac{1}{2}\Pr\{y = 0\} + \frac{1}{2}\Pr\{y \neq 0\} \\ &= \frac{1}{2}\Pr\{y = 0\} + \frac{1}{2}(1 - \Pr\{y = 0\}) \\ &= \frac{1}{2}\end{aligned}$$

$$\begin{aligned}\text{Thus, } \Pr\{P = 0\} &= \Pr\{p_1 = 0 \wedge p_2 = 0 \wedge \dots \wedge p_i = 0 \wedge \dots \wedge p_n = 0\} \\ &\leq \Pr\{p_i = 0\} \\ &= \frac{1}{2}\end{aligned}$$

That is, the probability that the algorithm returns an **incorrect YES (False Positive)** is **at most**  $\frac{1}{2}$ :

- $P = 0$  despite the fact that  $D \neq 0$ .

Thus, the probability that the algorithm produces the correct result is **at least**  $1 - \frac{1}{2} = \frac{1}{2}$ .

Thus, Freivalds' algorithm is a  $\frac{1}{2}$ -correct algorithm.

# Contention Resolution

Suppose there are  $n$  agents  $P_1, P_2, \dots, P_n$ , which are all competing for a shared resource  $R$ .

Imagine a scenario where these  $n$  agents are processes in a ***distributed system*** attempting to access a shared database.

## **Observations:**

- If all processes behaved ***identically***, this would lead to ***no progress*** as all processes would be attempting to access the shared database simultaneously at all times.
- ***Randomness*** can help break ***symmetry*** in this kind of scenarios.



# Contention Resolution: Analysis

Suppose that each process will attempt to access the database in each round with probability  $p$ .

**Notation:**

Let  $A[i, t]$  denote the event that  $P_i$  attempts to access the database in round  $t$ .

We know that each process attempts to access the database in each round with probability  $p$  so

$$\Pr\{A[i, t]\} = p$$

For every event, there is a **complementary** event, indicating that the event did not occur.

$\overline{A[i, t]}$  denotes the event that  $P_i$  does not attempt to access the database in round  $t$ .

## Contention Resolution: Analysis

$$\Pr\{\overline{A[i, t]}\} = 1 - \Pr\{A[i, t]\} = 1 - p$$

Let  $S[i, t]$  denote the event that  $P_i$  succeeds in accessing the database.

Thus, we can define  $S[i, t]$  as

$$S[i, t] = A[i, t] \cap \left( \bigcap_{j \neq i} \overline{A[j, t]} \right)$$

That is,  $P_i$  attempts to access the database in round  $t$  and the others do not attempt to access the database in round  $t$ .

# Contention Resolution: Analysis

Therefore,

$$\Pr\{S[i, t]\} = \Pr\{A[i, t]\} \cdot \prod_{j \neq i} \Pr\{\overline{A[j, t]}\} = p(1 - p)^{n-1} = f(p)$$

How do we choose  $p$  so that the success probability is maximized?

Differentiating  $f(p) = p(1 - p)^{n-1}$  with respect to  $p$ ,

we get

$$f'(p) = (1 - p)^{n-1} - (n - 1)p(1 - p)^{n-2}$$

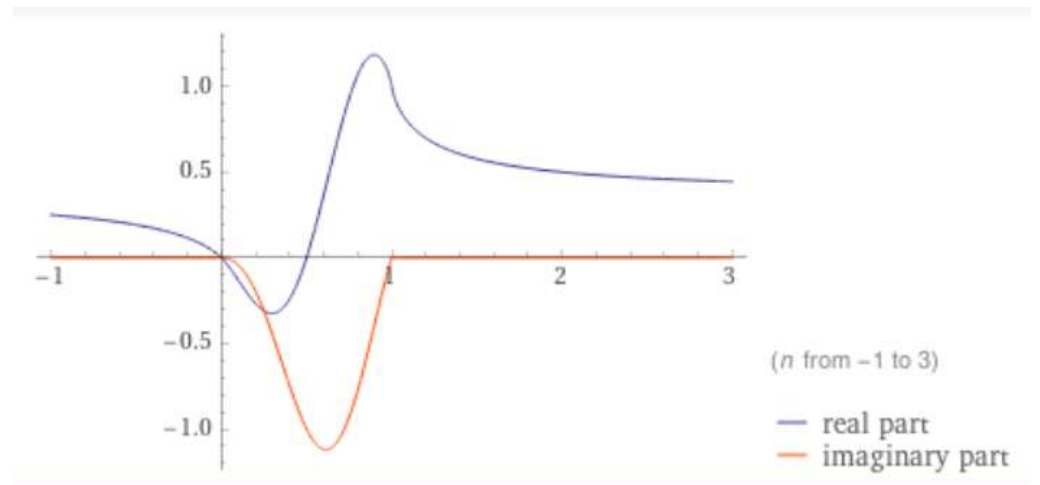
There is only local maximum between  $0 < p < 1$ .

That local maximum is  $p = \frac{1}{n}$ .

# Contention Resolution: Analysis

$$\Pr\{S[i, t]\} = \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1}$$

As  $n$  increases from 2, the function  $\left(1 - \frac{1}{n}\right)^{n-1}$  monotonically converges from  $\frac{1}{2}$  upto  $\frac{1}{e}$ .



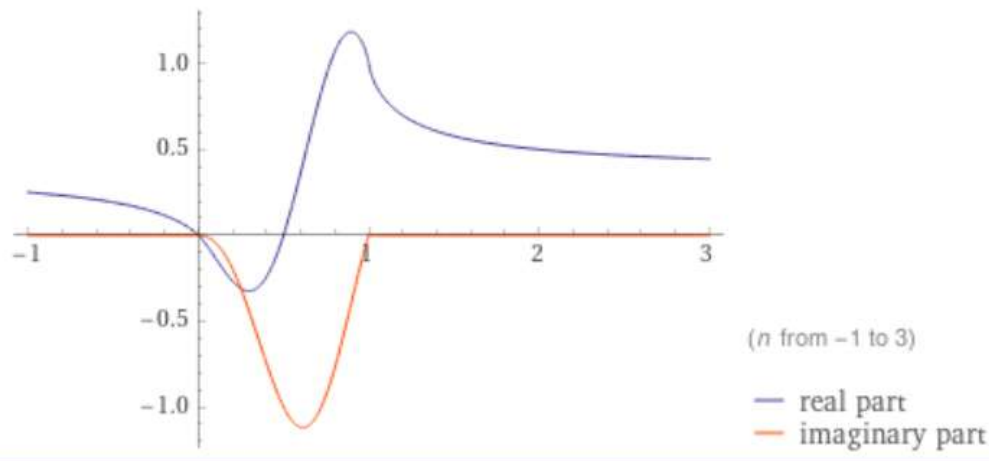
# Contention Resolution: Analysis

Hence, we can bound  $\Pr\{S[i, t]\}$  as follows:

$$\frac{1}{en} \leq \Pr\{S[i, t]\} \leq \frac{1}{2n}$$

Asymptotically,

$$\Pr\{S[i, t]\} = \Theta\left(\frac{1}{n}\right)$$



# Contention Resolution: Analysis

How long does it take for any process  $P_i$  to succeed at least once?

Let  $F[i, t]$  denote the failure event that  $P_i$  does not succeed in any of the rounds 1 through  $t$ .

We can define  $F[i, t]$  as the **intersection** of the complementary events  $\overline{S[i, r]}$  for  $r = 1, 2, \dots, t$ .

$$F[i, t] = \bigcap_{r=1}^t \overline{S[i, r]}$$

Since these events  $\overline{S[i, r]}$  for  $r = 1, 2, \dots, t$  are **all independent**,

$$\begin{aligned} \Pr\{F[i, t]\} &= \Pr\left\{\bigcap_{r=1}^t \overline{S[i, r]}\right\} = \prod_{r=1}^t \Pr\{\overline{S[i, r]}\} \\ &= \left(1 - \frac{1}{n}\right)^t \end{aligned}$$

# Contention Resolution: Analysis

Recall that the probability of success is  $\frac{1}{en} \leq \Pr\{S[i, t]\} \leq \frac{1}{2n}$  *after one round*.

Thus,

$$\Pr\{F[i, t]\} = \prod_{r=1}^t \Pr\{\overline{S[i, r]}\} \leq \left(1 - \frac{1}{en}\right)^t$$

Setting  $t = \lceil en \rceil$ ,

$$\Pr\{F[i, t]\} \leq \left(1 - \frac{1}{en}\right)^{\lceil en \rceil} \leq \left(1 - \frac{1}{en}\right)^{en} \leq \frac{1}{e}$$

[\*\*\*  $\lim_{n \rightarrow \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e}$ ]

# Contention Resolution: Analysis

Therefore, the probability that any process  $P_i$  does not succeed in any of rounds 1 through  $t$  is bounded by the **constant**  $\frac{1}{e}$ , independent of the number of processes  $n$ .

Now if we set  $t = \lceil en \rceil (c \ln n)$ ,

$$\begin{aligned}\Pr\{F[i, t]\} &\leq \left(1 - \frac{1}{en}\right)^t = \left(1 - \frac{1}{en}\right)^{\lceil en \rceil c \ln n} \\ &\leq \frac{1}{e}^{c \ln n} = n^{-c}\end{aligned}$$

After  $\Theta(n)$  rounds, the probability that  $P_i$  has not yet succeeded is bounded by a constant.

Between then and  $\Theta(n \log n)$  rounds, this probability drops to a quantity that is quite small, bounded by an **inverse of polynomial** in  $n$ .



# Contention Resolution: Analysis

How many rounds must elapse before there is **high probability** that all the processes have **succeeded** in accessing the database **at least once**?

We say that the protocol fails after  $t$  rounds if some processes have not yet succeeded.

Let  $F_t$  denote the event that the protocol fails after  $t$  rounds.

**Goal:** Find a reasonably small value of  $t$  for which  $\Pr\{F_t\}$  is small.

# Contention Resolution: Analysis

**Observation:** The event  $F_t$  occurs if and only if one of the events  $F[i, t]$  occurs.

$$F_t = \bigcup_{i=1}^n F[i, t]$$

Therefore,

$$\Pr\{F_t\} \leq \sum_{i=1}^n \Pr\{F[i, t]\} \quad [\text{The Union Bound}]$$

Setting  $t = 2 \lceil en \rceil \ln n$ ,

$$\Pr\{F_t\} \leq \sum_{i=1}^n \Pr\{F[i, t]\} \leq n \cdot n^{-2} = \frac{1}{n}$$

# Contention Resolution: Analysis

Therefore, we can conclude that with probability at least  $1 - \frac{1}{n}$ , all processes succeed in accessing the database at least once within  $t = 2 \lceil en \rceil \ln n$  rounds.

## Observations:

- If we choose  $t = cn \ln n$  where  $c$  is a very small quantity, then an upper bound on  $\Pr\{F[i, t]\}$  that is larger than  $\frac{1}{n}$  and hence an upper bound on  $\Pr\{F_t\}$  that is larger than 1, which is a completely useless bound.
- The algorithm is a **Las Vegas** algorithm as the running time cannot be bounded deterministically, but it will terminate with **high probability** within  $2 \lceil en \rceil \ln n$  rounds.
- correctness is always ensured in the sense that none of the processes can access the database simultaneously.