Data Structures and Algorithms

(**CS210A**)

Semester I – **2014-15**

Lecture 2:

- Model of computation
- Efficient algorithm for **F**(n) **mod** m.

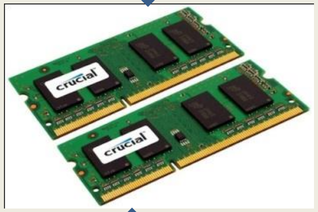
An important Lesson from Lecture 1

- Design of **efficient** algorithm is very important.

(One can learn this lesson in the real sense only after he/she does the corresponding implementation totally himself/herself. Otherwise it is just one of those bookish facts which one believes/remembers)

Recall from Lecture 1:
Current-state-of-the-art Desktop

A processor (CPU)
speed = few GHz
(a few **nanoseconds** to execute an instruction)



Internal memory (RAM)
size = a few GB (Stores few million bytes/words)
speed = a few GHz(a few **nanoseconds** to read a byte/word)

External Memory (Hard Disk Drive)
size = a few tera bytes
speed : seek time = **miliseconds**
transfer rate= a **billion** bytes per second

Models of computation

Why do we need such models?

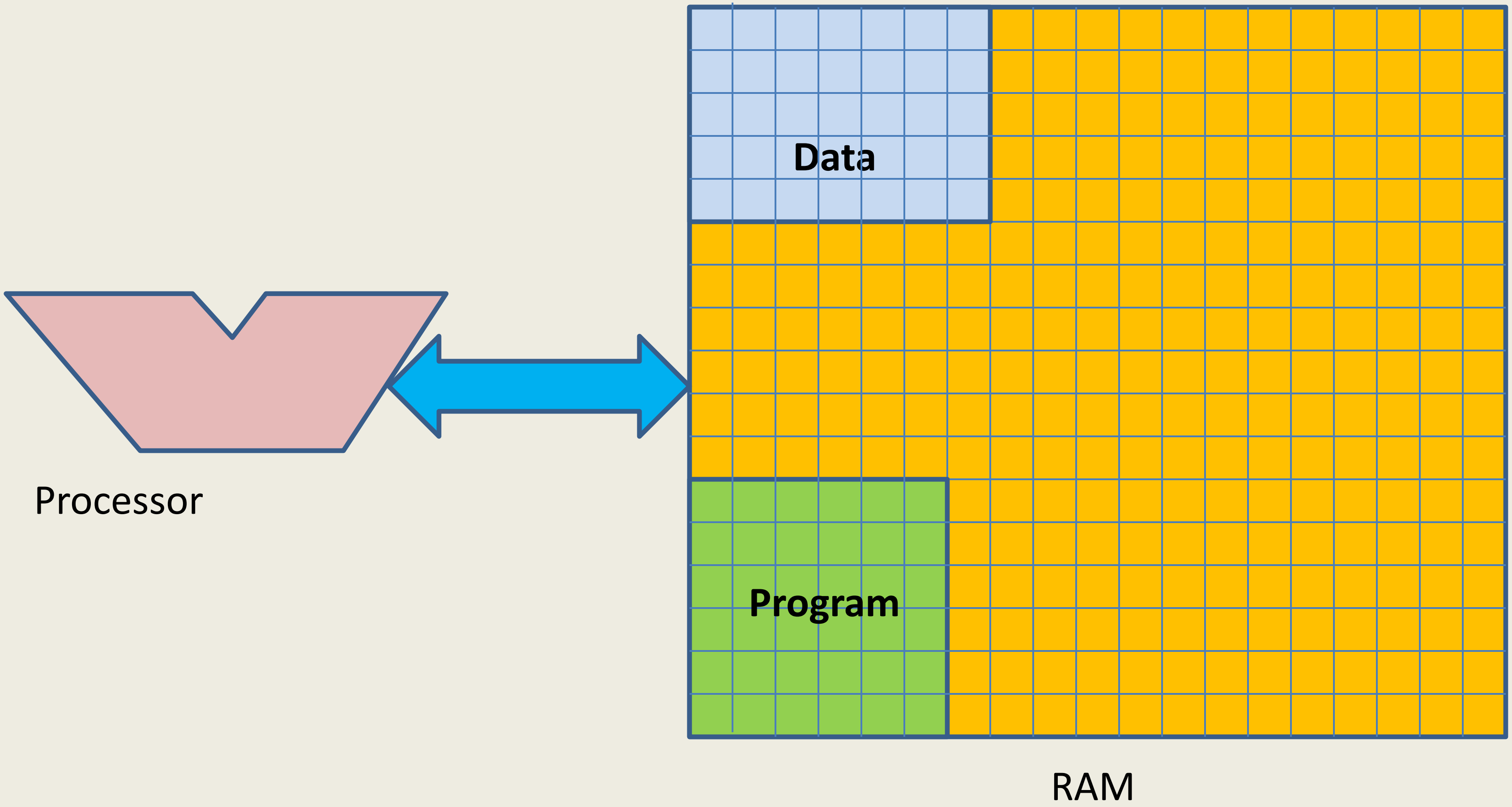
In order to analyze the efficiency of an algorithm, we need a model of computation which is **simpler** and still captures the **essence** of the real world computer.

Models :

- Word RAM model
- Bit complexity model
- Universal RAM model
- Cell probe model
- ….

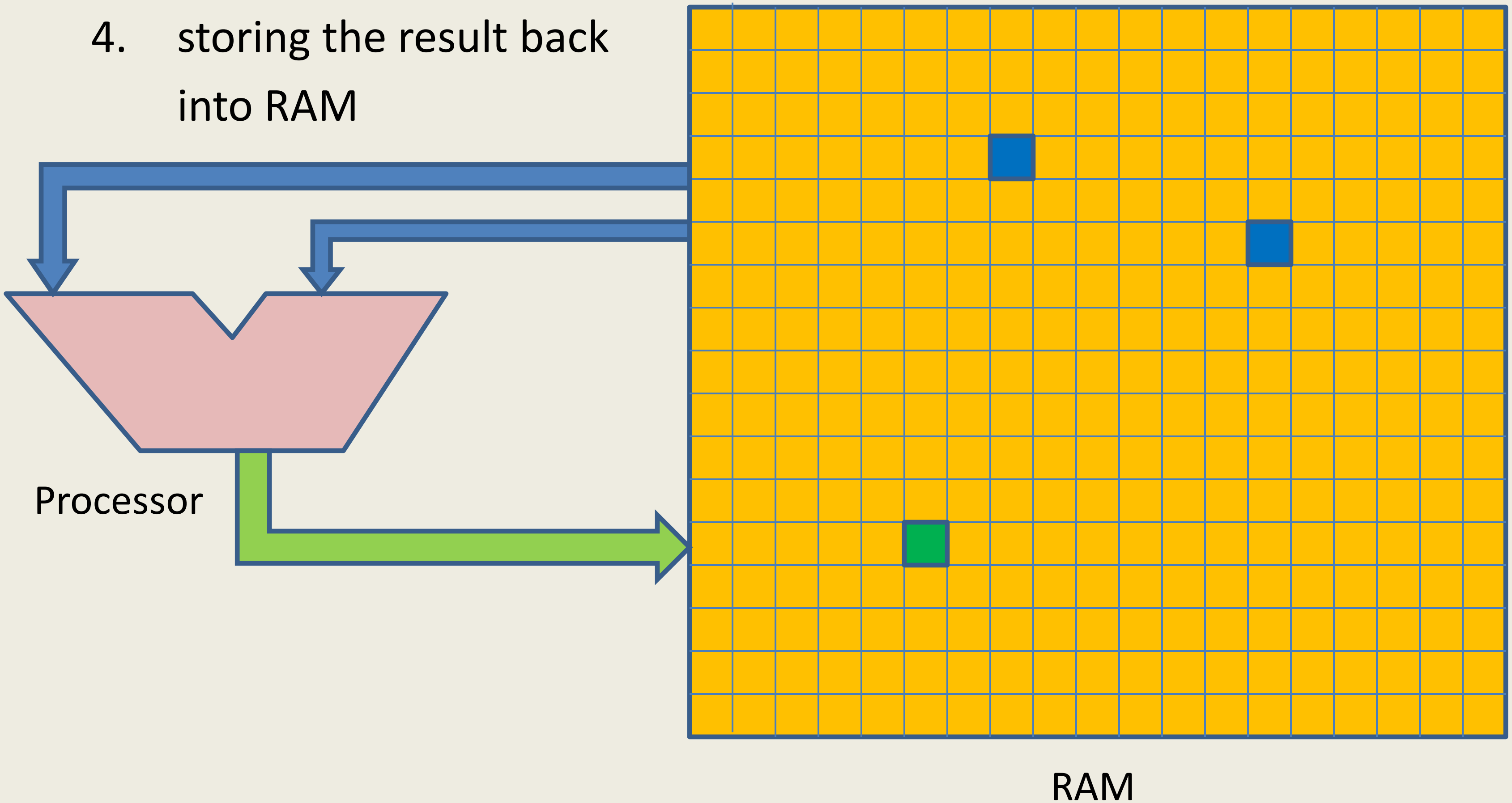
We shall deal mainly with **word-RAM model** due to its **simplicty** and higher degree of **closeness** to the real world computer.

word RAM : a model of computation



How is an instruction executed?

1. Decoding instruction
2. fetching the operands
3. performing arithmetic/logical operation
4. storing the result back into RAM



➔ Each instruction takes a <u>few cycles (click ticks) </u>to get executed.

word RAM model of computation: Characteristics

- Word is the **basic storage** unit of RAM. Word is a collection of few bytes.
- Each input item (number, name) is stored in **binary format**.
- RAM can be viewed as a huge array of words. Any arbitrary location of RAM can be **accessed** in the same time **irrespective** of the location.
- Data as well as Program **reside fully** in RAM.
- Each arithmetic or logical operation (+,-,*,/,or, xor,…) involving a constant number of words takes **a constant number of cycles (steps)** by the CPU.

Homework 1 from Lecture 1

Computing F(n) mod m

Algorithms for Fibonacci numbers

Fibonacci numbers

F(0) = 0;

F(1) = 1;

F(n) = F($n-1$) + F($n-2$) for all n >1;

Exercise 1 : Using induction or otherwise, show that F(n)>$2^{\frac{n-2}{2}}$

Algorithms you must have implemented for computing F(n) :

- **Iterative**
- **recursive**

Iterative Algorithm for F(n) mod m

Total number of instructions=
4+3(n-2)+1 < **3n**

IFib(n,m)

```
if n = 0  return 0;
    else if n =1 return 1;
        else {      a ← 0;  b ← 1;
                    For(i =2 to n) do
                    {    temp ← b;
                         b ← a + b  mod m ;
                         a ← temp;
                    }
              }
  return b;
```

4 instructions

n-2 iterations

3 instructions per iteration

the final instruction

Let us calculate the number of instructions executed by IFib(n,m)

Recursive algorithm for F(n) mod m

RFib(n, m)

```
{   if n =0 return 0;
      else if n =1 return 1;
           else return((RFib(n − 1,m) + RFib(n − 2,m) ) mod m)
}
```

Let G(n) denote the number of instructions executed by RFib(n, m)

- **G(**0**) = 1; G(**1**) = 2;**
- For n **>1 G(**n**) = G(**$n-1$ **)+G(**$n-2$**) +** 4

Observation 1: **G**(n)>**F**(n) for all n;

It follows from **Observation 1** and **Exercise 1** that **G**(n)> $2^{(n-2)/2}$**!!!**

Algorithms for F(n)mod m

- # instructions by **Recursive** algorithm **RFib**(n): $> 2^{\frac{n-2}{2}}$
  (exponential in n)
- # instructions by **Iterative** algorithm **IFib**(n): $3n$
  (linear in n)

None of them works for entire range of **long long int** n and **int** m

Question: Can we compute **F**(n)**mod** m quickly ?

How to compute F(n)mod m quickly ?

… need some better insight …

A warm-up example

How good are your programming skills ?

Compute $x^n \mathbf{mod}\ m$

Problem: Given three integers x, n, and m, compute $x^n \mathbf{mod}\ m$.

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x \times x^{n-1} & \text{otherwise} \end{cases}$$

Power(x, n, m)

If ($n = 0$) return 1;

else {

$temp \leftarrow$ **Power**($x, n-1, m$);

$temp \leftarrow (temp \times x)\ \mathbf{mod}\ m$;

return $temp$;

}

4 instructions excluding the **Recursive** call

Compute $x^n \textbf{ mod } m$

Problem: Given three integers x, n, and m, compute $x^n \textbf{ mod } m$.

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x \times x^{n-1} & \text{otherwise} \end{cases}$$

Power(x, n, m)

↓

Power($x, n-1, m$);

↓

Power($x, n-2, m$);

↓

⋮

Power($x, 0, m$)

No. of instructions executed by **Power**(x, n, m) = $4n$

Compute $x^n \textbf{ mod } m$

Problem: Given three integers x, n, and m, compute $x^n \textbf{ mod } m$.

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x^{n/2} \times x^{n/2} & \text{if } n \text{ is } \textbf{even} \\ x^{n/2} \times x^{n/2} \times x & \text{if } n \text{ is } \textbf{odd} \end{cases}$$

```
Power(x, n, m)
    If (n = 0) return 1;
    else {
        temp ← Power(x, n/2, m);
        temp ← (temp × temp) mod m ;
        if (n mod 2 = 1 ) temp ←(temp × x)mod m ;
        return temp;
    }
```

5 instructions excluding the **Recursive** call

Compute $x^n \bmod m$

Problem: Given three integers x, n, and m, compute $x^n \bmod m$.

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x^{n/2} \times x^{n/2} & \text{if } n \text{ is } \textbf{even} \\ x^{n/2} \times x^{n/2} \times x & \text{if } n \text{ is } \textbf{odd} \end{cases}$$

Power(x, n, m)

↓

Power(x, $n/2$, m)

↓

Power(x, $n/4$, m)

⋮

Power(x, 0, m)

No. of instructions executed by **Power**(x, n, m) = $5 \log_2 n$

Efficient Algorithm for $\mathbf{F}(n)$mod m

Efficient algorithm for F(n)mod m

Idea1 : Can we express **F**(n) as a^n for some constant a ?

Unfortunately **no**.

Idea 2

$$\begin{pmatrix} F(n) \\ F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} F(n-1) \\ F(n-2) \end{pmatrix}$$

?

Unfolding the RHS of this equation, we get …

$$\begin{pmatrix} F(n) \\ F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \times \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

A clever algorithm for F(n)mod m

Clever-algo-Fib(n, m)

{ $\quad A \leftarrow \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$;

$\quad B \leftarrow A^{n-1}$ mod m;

$\quad C \leftarrow B \times \binom{1}{0}$;

$\quad$ return $C[1]$; $\qquad$ // the first element of vector C stores **F**(n)**mod** m

}

Question: How to compute $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1}$ efficiently ?

Answer :

Inspiration from Algorithm for $x^n \,\mathbf{mod}\, m$

A clever algorithm for $\mathrm{F}(n)$ mod m

Let $\boldsymbol{A}$ be a 2×2 matrix.

- If n is even, $\boldsymbol{A}^n = \boldsymbol{A}^{n/2} \times \boldsymbol{A}^{n/2}$
- If n is odd, $\boldsymbol{A}^n = \boldsymbol{A}^{n/2} \times \boldsymbol{A}^{n/2} \times \boldsymbol{A}$

Question: How many instructions are required to multiply two 2×2 matrices ?

Answer: 12

Question: Number of instructions for computing $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1}$?

Answer : $27 \, \mathbf{log}_2 \, (n-1)$

Question: Number of instructions in **New-algo-Fib**(n, m)

Answer: $27 \, \mathbf{log}_2 (n-1) + 6$

Which algorithm is better ?

Algorithm for F(n)mod m	No. of Instructions
RFib(n,m)	$> 2^{(n-2)/2}$
IterFib(n,m)	$3n$
Clever_Algo_Fib(n,m)	$27 \log_2 (n-1) + 6$

Find out yourself ?

Assignment 1