High Performance Scientific computing
Lecture 3

S. Gopalakrishnan

Numerical Modeling ?

- Why do we need need numerical modelling in the first place?
- Usually some complex or not so complex phenomena needs to studied.

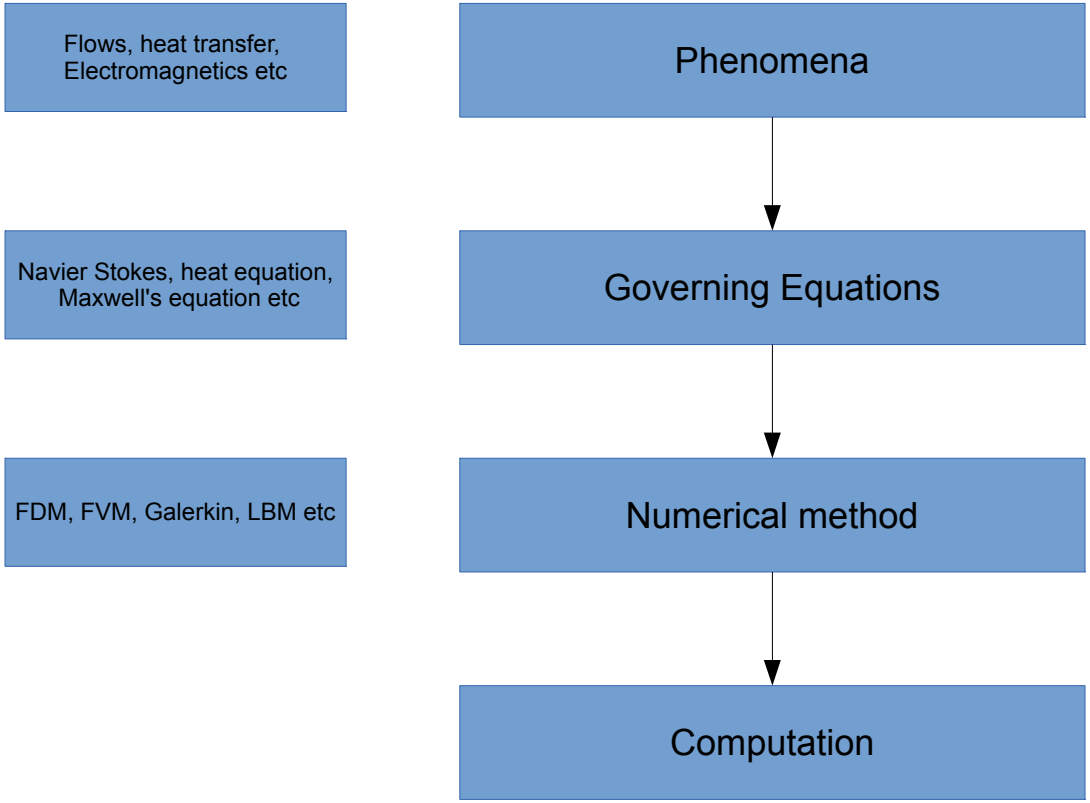
Numerical Modeling ?

- Why do we need need numerical modelling in the first place?
- Usually some complex or not so complex phenomena needs to studied.
- Analytical solutions are not easily available or in most cases not achievable at all.

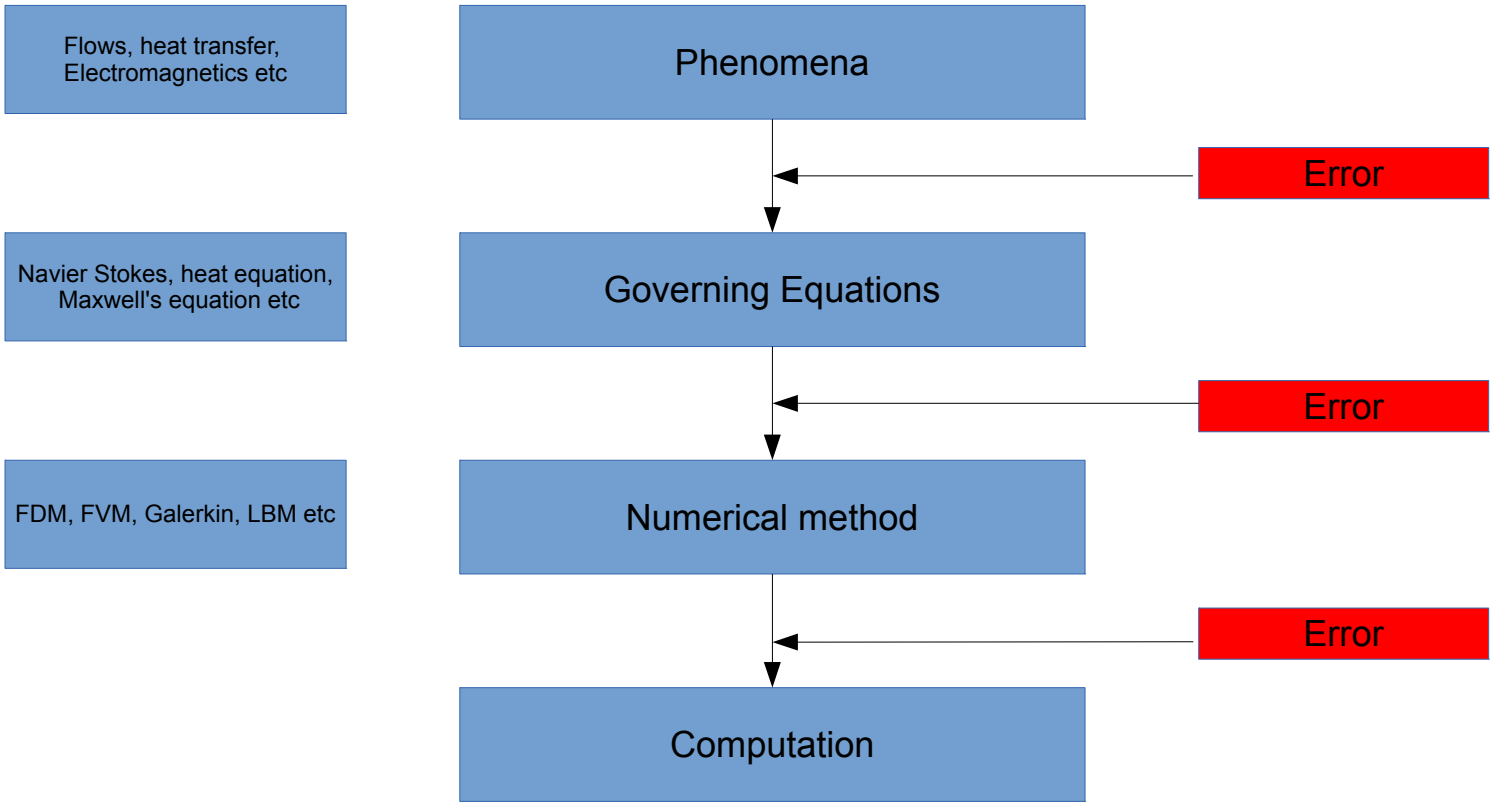
Numerical Modeling ?

- Why do we need need numerical modelling in the first place?
- Usually some complex or not so complex phenomena needs to studied.
- Analytical solutions are not easily available or in most cases not achievable at all.
- What are the steps involved in numerical modelling?

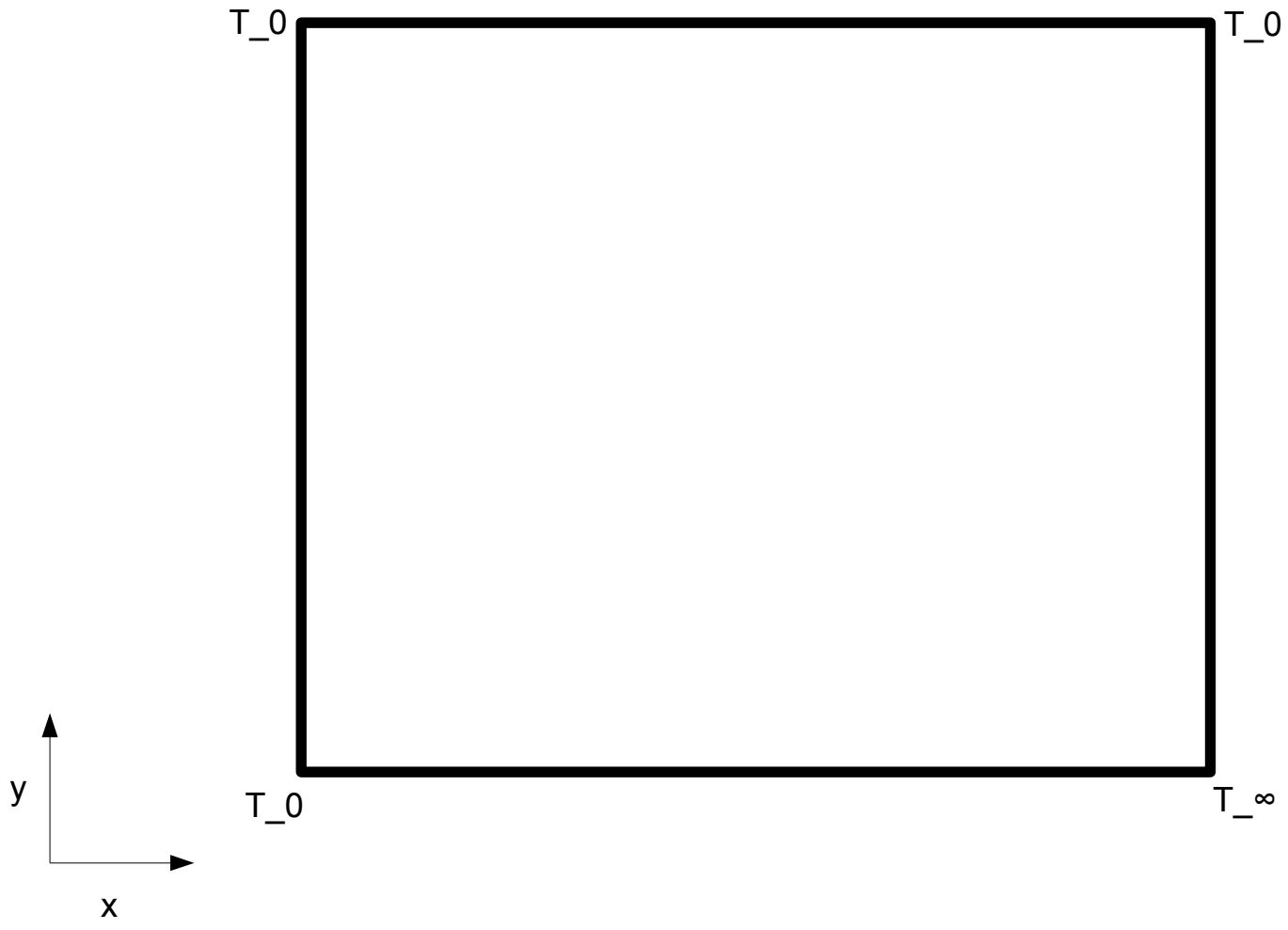
Numerical Modeling



Numerical Modeling



Steady State Heat Conduction

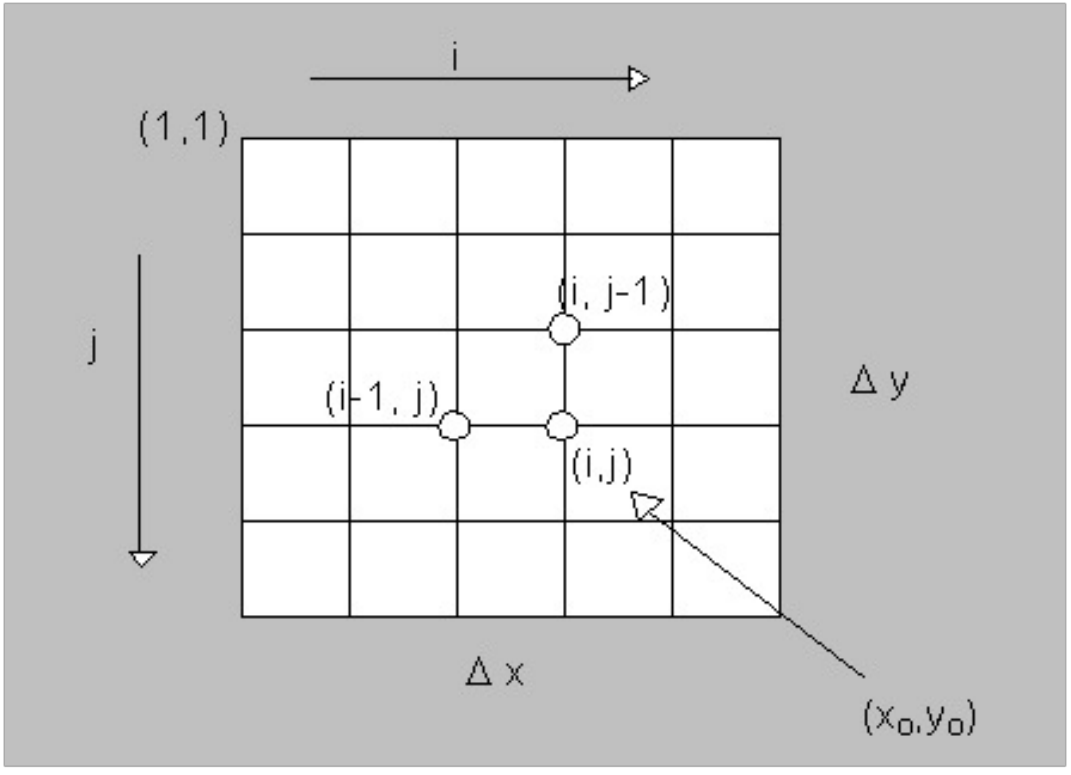


Steady State Heat Conduction

Phenomenon is modelled using Laplace's equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = \nabla^2 T = 0$$

which can be discretised on a grid as,



Steady State Heat Conduction

The discretised equation at a single point (i,j) is

$$\frac{T_{i-1,j} - T_{i,j} + T_{i+1,j}}{(\Delta x)^2} + \frac{T_{i,j-1} - T_{i,j} + T_{i,j+1}}{(\Delta y)^2} = 0$$

Assemble all the equations for all unknown points in the Matrix form and then solve

$$Ax = B$$

You can choose any Linear Algebra Solver (iterative or Direct). Iterative is more efficient.

So what are the steps involved ?

One writes a computer code,

```
#define NC       1000                  /* Number of Cols        */
#define NR       1000                   /* Number of Rows        */
#define NITER    1000                   /* Max num of Iterations */
#define MAX(x,y) ( ((x) > (y)) ? x : y )

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h> /* only for timing */
#include <sys/time.h>

void initialize( float t[NR+2][NC+2] );
void set_bcs   ( float t[NR+2][NC+2]);

int main( int argc, char **argv ){

  int        niter;                    /* iter counter  */

  float      t[NR+2][NC+2];           /*temperature */
  float      told[NR+2][NC+2];        /* previous temperature*/
  float      dt;                       /* Delta t       */

.................................
```

Compile and Execute!!!

Steady State Heat Conduction

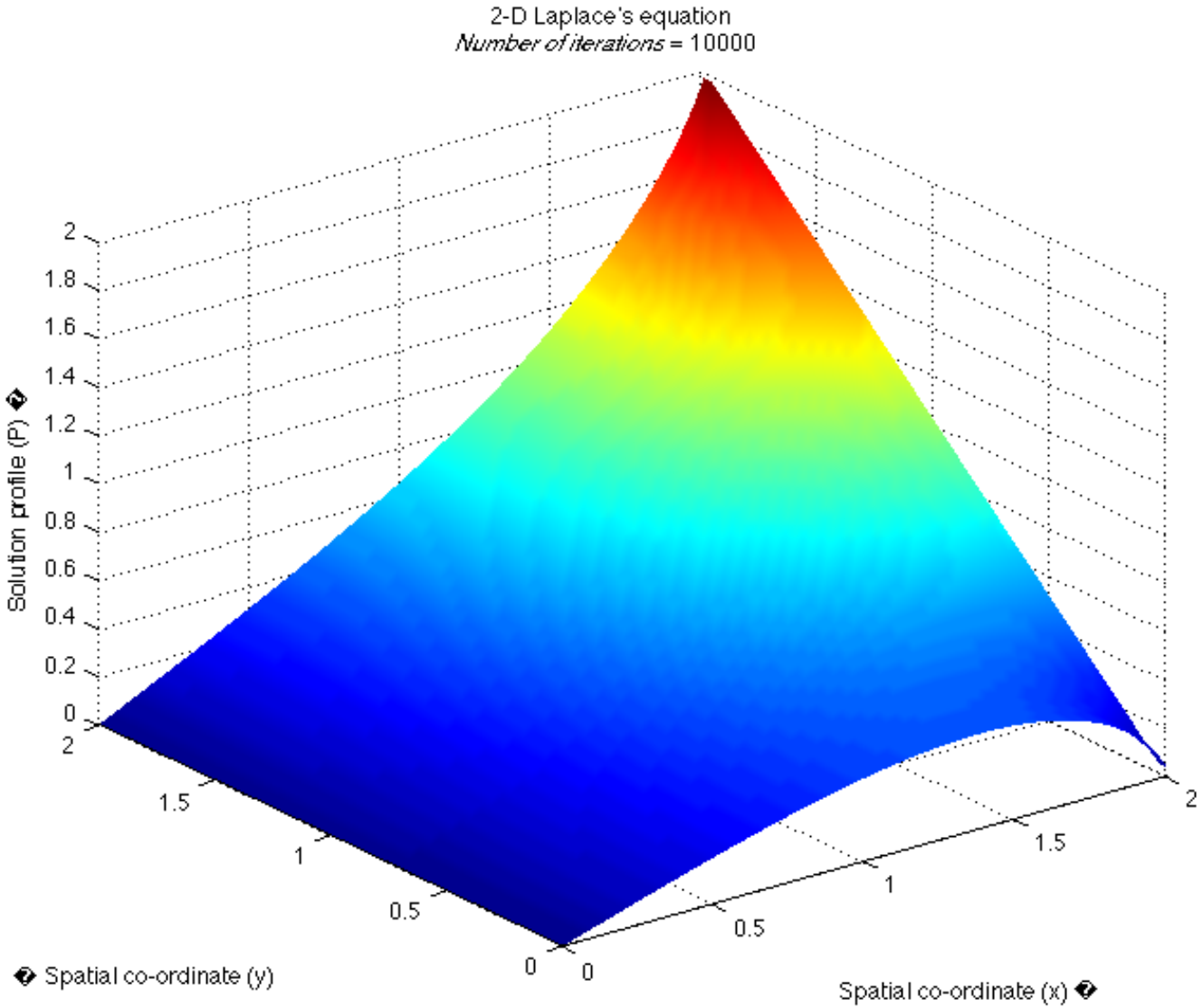


Image: Mathworks

Is it that simple? What happens in between?

Is it that simple? What happens in between?

- Computers are incredibly stupid machines. All they understand are zeros and ones!

Is it that simple? What happens in between?

- Computers are incredibly stupid machines. All they understand are zeros and ones!
- Microprocessors have logic circuits to perform basic mathematical operations such as (ADD, MUL, SUB etc) on binary numbers.

Is it that simple? What happens in between?

- Computers are incredibly stupid machines. All they understand are zeros and ones!
- Microprocessors have logic circuits to perform basic mathematical operations such as (ADD, MUL, SUB etc) on binary numbers.
- Each operation consists a set of "instructions" which the processor performs

Is it that simple? What happens in between?

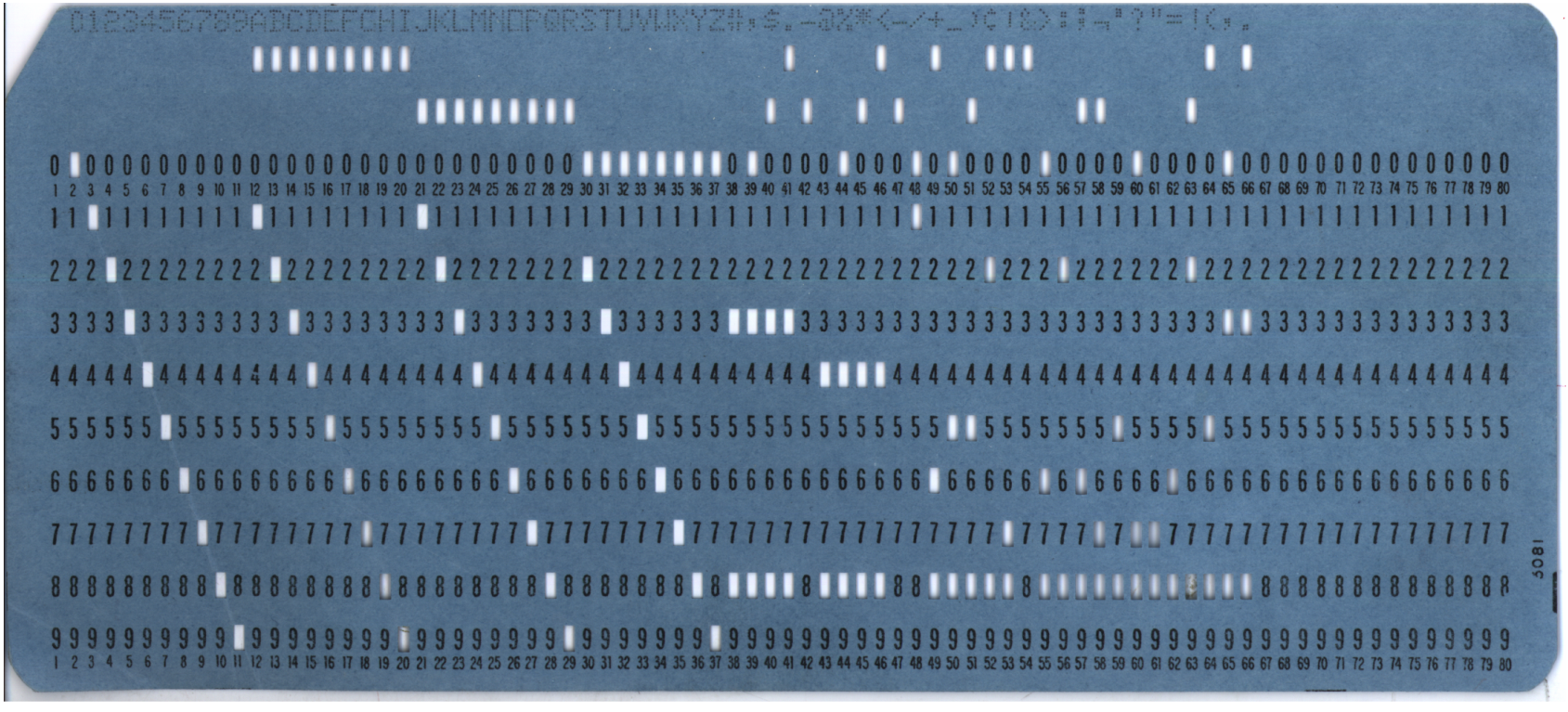
- Computers are incredibly stupid machines. All they understand are zeros and ones!
- Microprocessors have logic circuits to perform basic mathematical operations such as (ADD, MUL, SUB etc) on binary numbers.
- Each operation consists a set of "instructions" which the processor performs
- Our calculations are broken down into thousands (or even millions) of such instructions which are then executed.

Is it that simple? What happens in between?

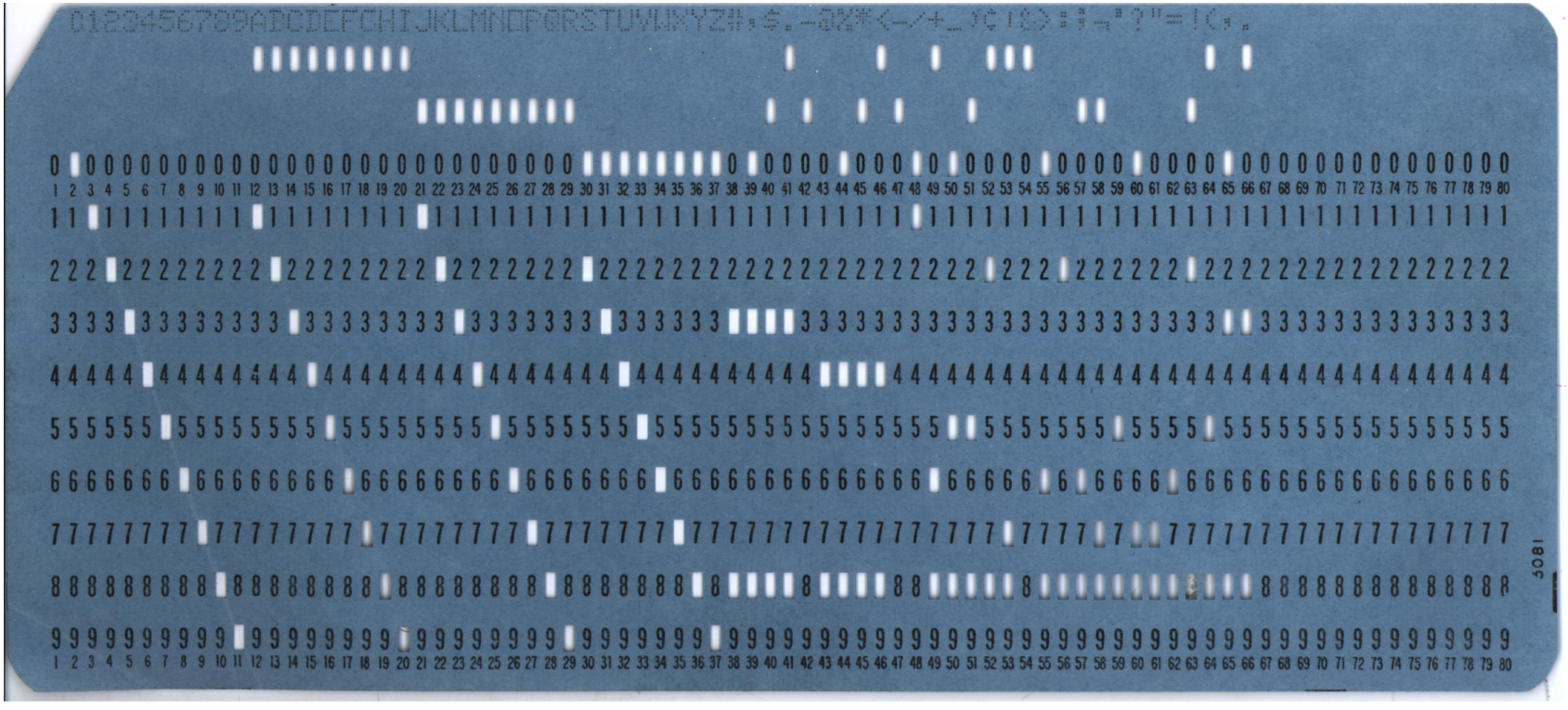
- Computers are incredibly stupid machines. All they understand are zeros and ones!
- Microprocessors have logic circuits to perform basic mathematical operations such as (ADD, MUL, SUB etc) on binary numbers.
- Each operation consists a set of "instructions" which the processor performs
- Our calculations are broken down into thousands (or even millions) of such instructions which are then executed.
- All these operations are basically hidden from the User.

So let's dive into the Rabbit hole: Blue pill or Red Pill?

How many of you recognise this?



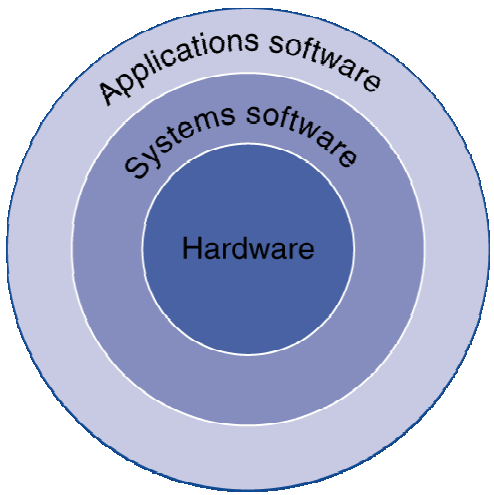
How many of you recognise this?



A Punch card !! Image: Wikipedia

Computer System Layers

- Application software :Written in high-level language
- System software
- Compiler: translates HLL code to machine code
- Operating System: service code
  - Handling input/output
  - Managing memory and storage
  - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers



Abstractions

- Abstraction helps us deal with complexity
- Hide lower-level detail.
- Instruction set architecture - ISA: An abstract interface between the hardware and the lowest level software of a machine
  - Encompasses all the information necessary to write a machine language program that will run correctly, including instructions, registers, memory access, I/O
  - Eg. x86, x86_64, IA-32, ARM, SPARC, Motorola 68K, MIPS
- ABI (application binary interface): The user portion of the instruction set plus the operating system interfaces used by application programmers
  - Defines a standard for binary portability across computers

Credit: Based on notes from EC232 UMass – Amherst

Von Neumann Architecture

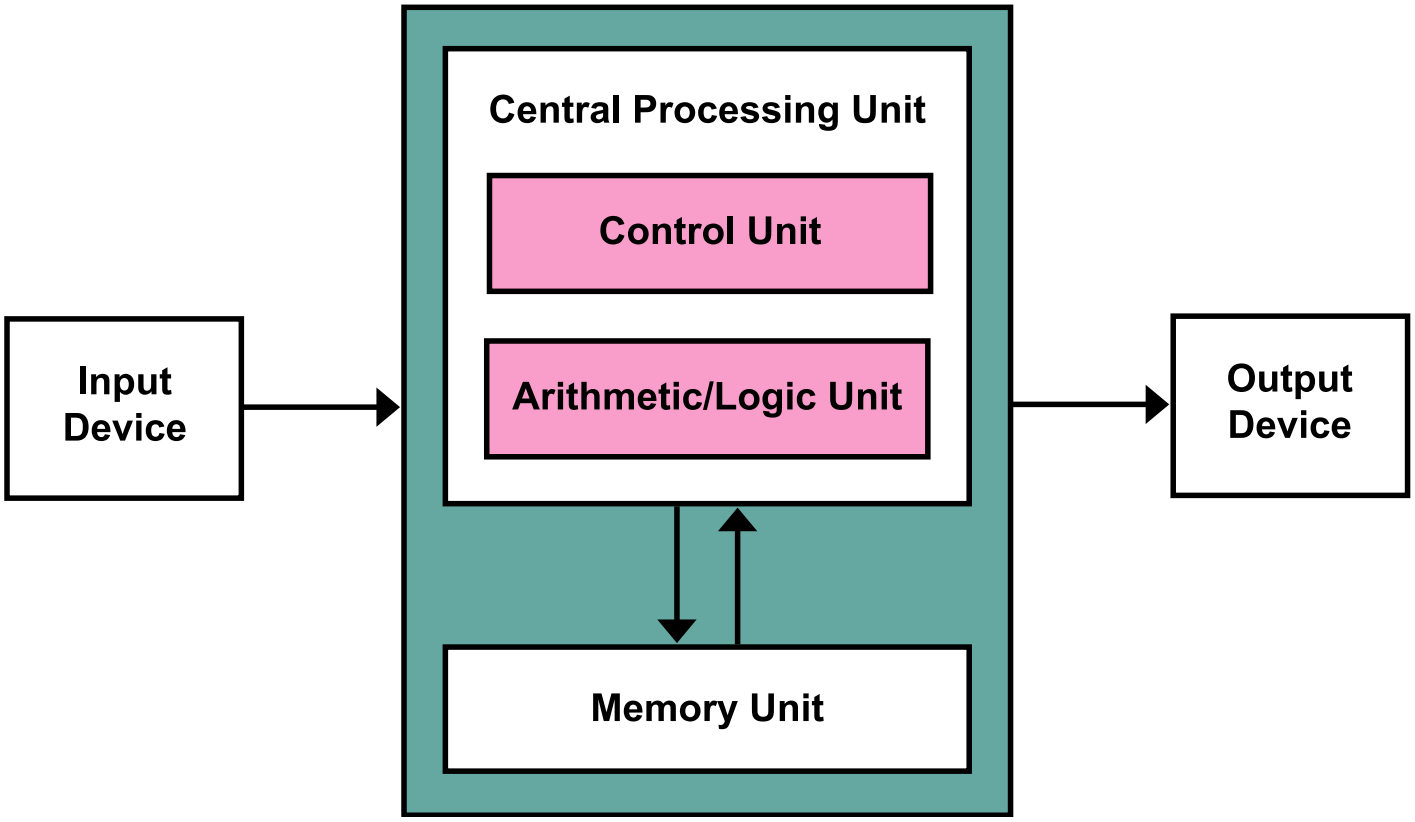


Image: Wikipedia

Program Code Levels

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability

```
for( i=0; i<=NR+1; i++ )        /* Copy the values into told */
    for( j=0; j<=NC+1; j++ )
      told[i][j] = t[i][j];
```

Program Code Levels

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability

```
for( i=0; i<=NR+1; i++ )        /* Copy the values into told */
    for( j=0; j<=NC+1; j++ )
      told[i][j] = t[i][j];
```

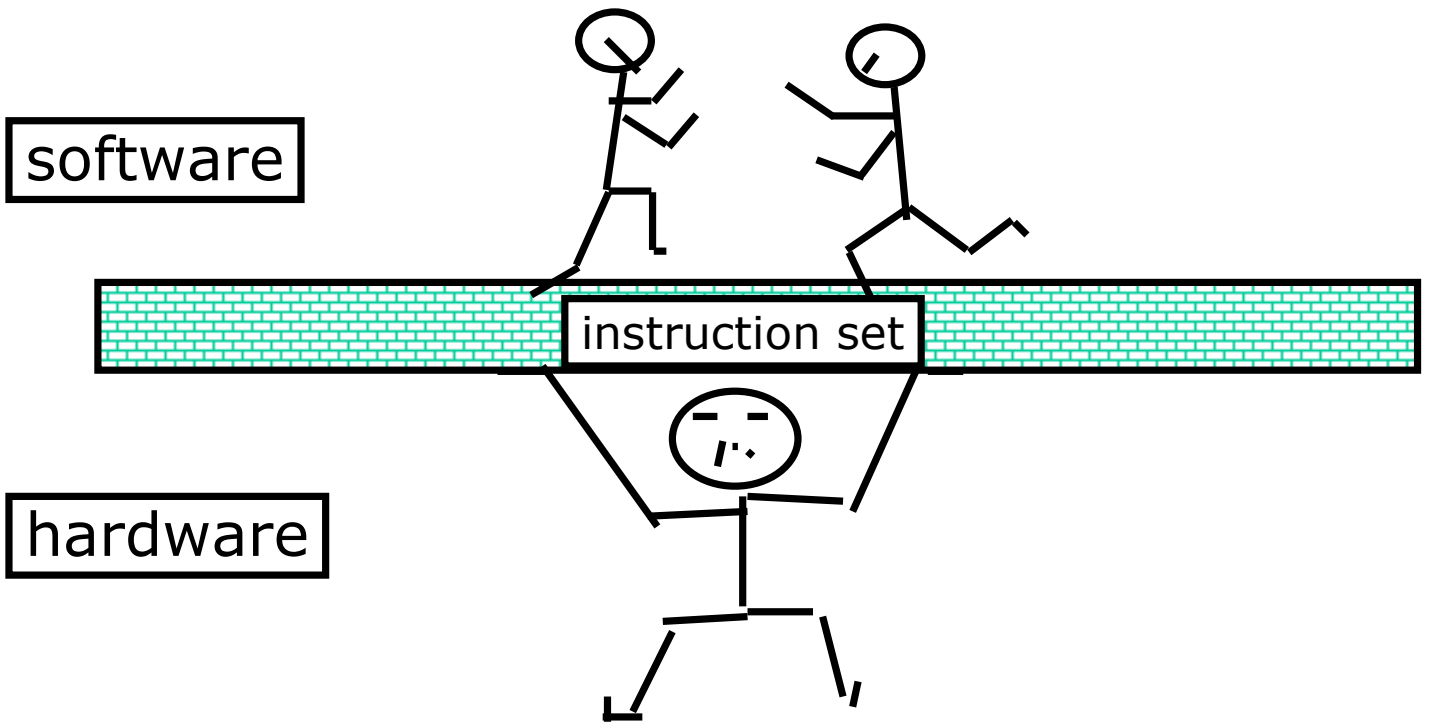
- Assembly language
  - Textual representation of instructions.
  - Compilation and linking of program will yield this

```
 je else
 jmp endif
else:
endif:
 sarl $1, %edx
 movl %edx, %eax
 addl %eax, %edx
 addl %eax, %edx
 addl $1, %edx
 addl $1, %ecx
```

- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

Instruction set is a critical interface

The actual programmer visible hardware view



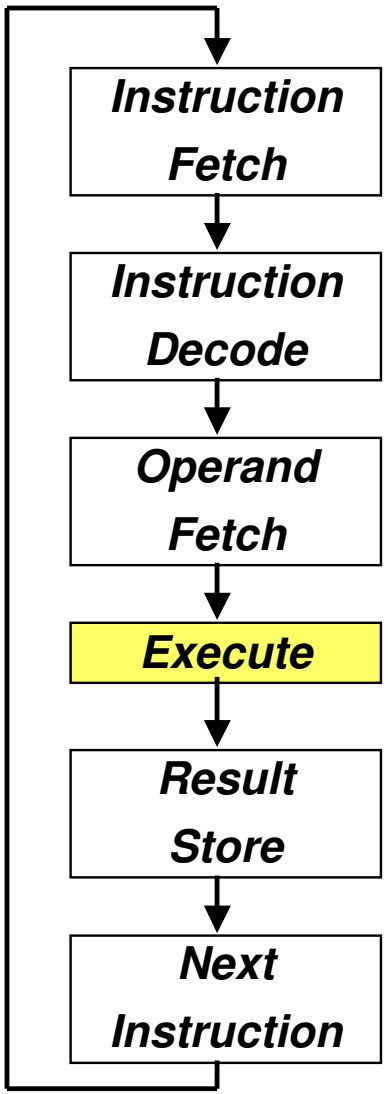
Credit: Based on notes from EC232 UMass – Amherst

Von Neumann Computer

- Stored Program Concept
  - A instruction is a string of bits
  - A program is written as a sequence of instructions
  - Instructions are stored in a memory
  - They are read one by one, decoded and executed
  - Also called Von Neumann Computer after the inventor of the stored program concept
- The First Von Neumann Computer was built at the University of Manchester in 1948
  - Vacuum Tube
  - Magnetic Drum Memory

Credit: Based on notes from EC232 UMass – Amherst

Execution cycle



Obtain instruction from program storage

Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

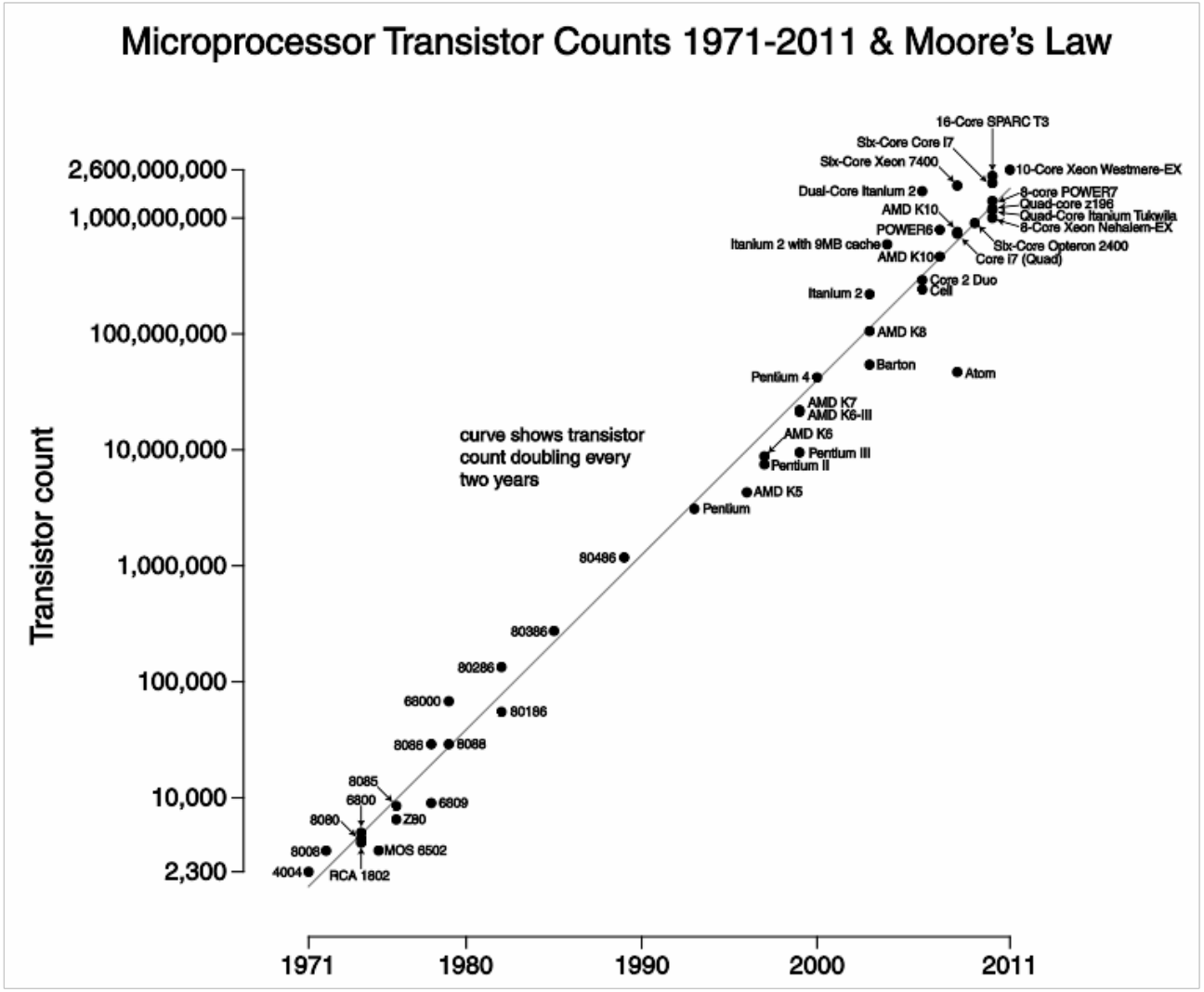
Deposit results in storage for later use

Determine successor instruction

Credit: Based on notes from EC232 UMass – Amherst

Back to Moore's Law

What are we doing with all those transistors?



Hardware level parallelism
or
Instruction level parallelism (ILP)

Instruction Pipelining

Increases instruction throughput, i.e number of instructions executed per second. But time for each instruction remains the same.

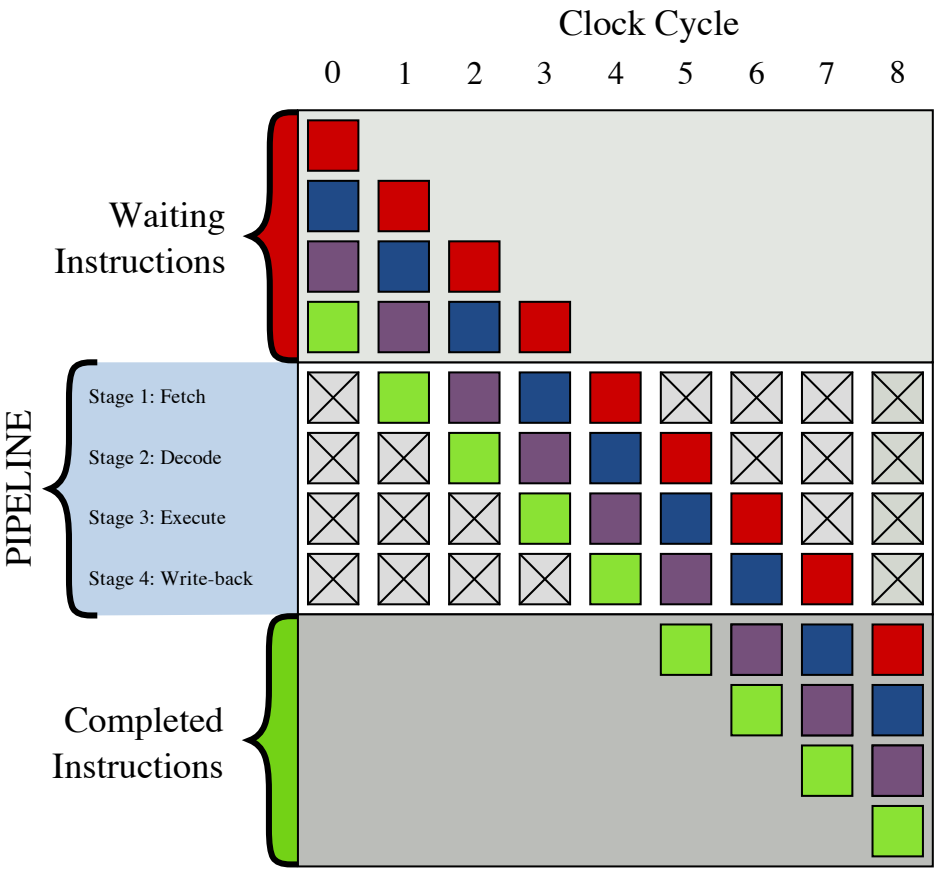


Image: Wikipedia

Pipelining Issues

Data dependencies can result in slowing of throughput.

```
for( i=0; i<=LAST; i++ )
a[i]=b[i]+c[i]
```

No deadlock in previous sample, pipelining will work like a charm.

Pipelining Issues

Data dependencies can result in slowing of throughput.

```
for( i=0; i<=LAST; i++ )
a[i]=b[i]+c[i]
```

No deadlock in previous sample, pipelining will work like a charm.

```
for( i=0; i<=LAST; i++ )
a[i]=b[i]+a[i-1]
```

This is a deadlock situation since $a[i]$ depends on $a[i-1]$

Superscalar execution

Multiple instructions being executed simultaneously. E.g,VLIW or EPIC in IA-64 Architecture.

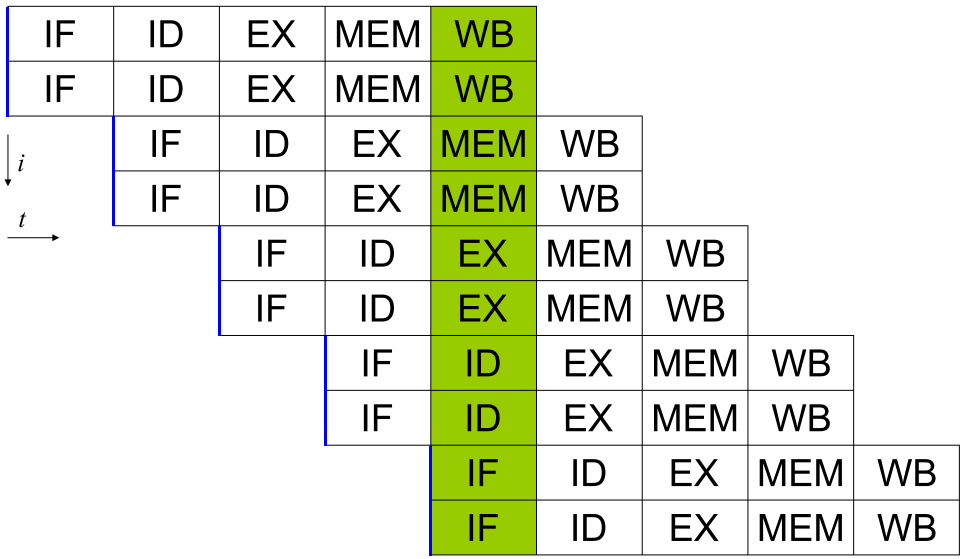


Image: Wikipedia

Other ILP techniques

- **Out-of-order execution** where instructions execute in any order that does not violate data dependencies. Note that this technique is independent of both pipelining and superscalar.
- **Speculative execution** which allows the execution of complete instructions or parts of instructions before being certain whether this execution should take place.
- **Branch prediction** which is used to avoid stalling for control dependencies to be resolved. Branch prediction is used with speculative execution.

Credit: Wikipedia

So what are you stacked against?

- Physical Phenomena
- Governing Equation
- Numerical Method
- Your Pseudocode / Algorithm
- Application: Higher level Language
- Compiler and Linker
- Assembly Language
- Binary Code
- Hardware