Lecture 1

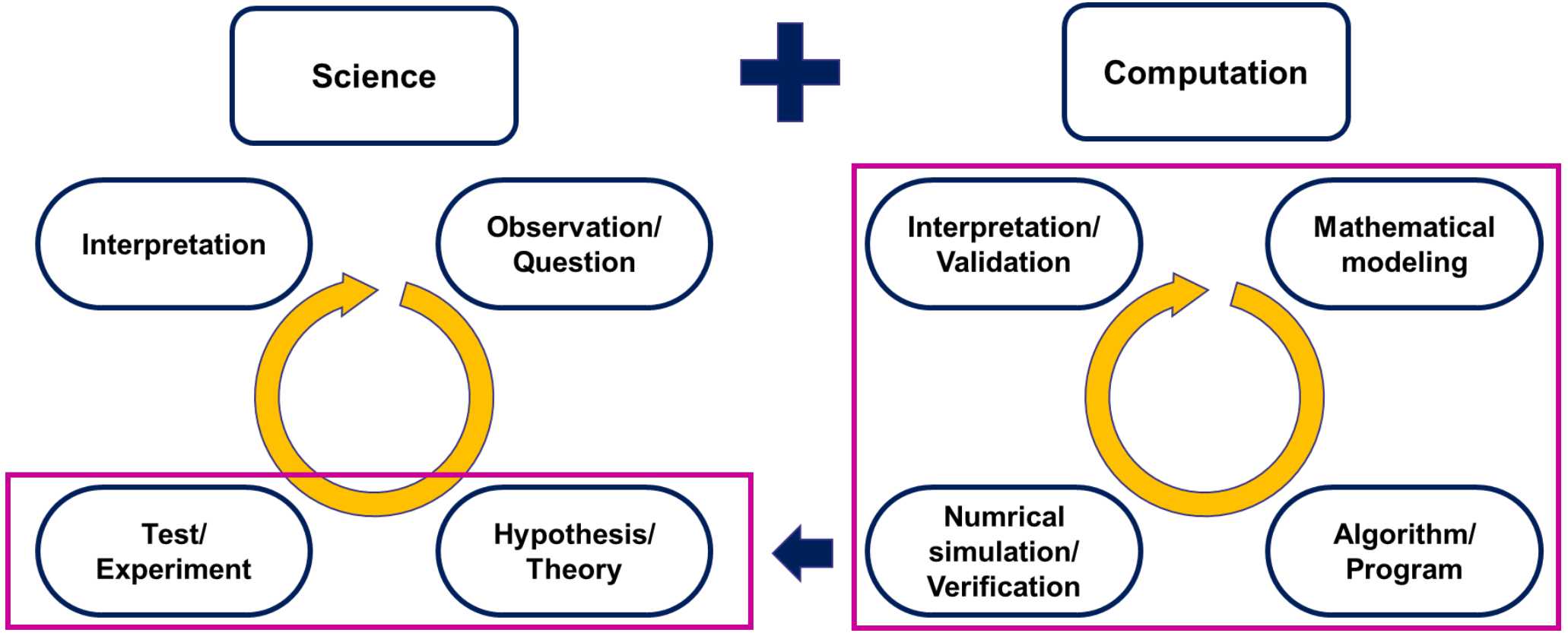
Introduction to Scientific Computation with Python

Jung-Il Choi
School of Mathematics and Computing (Computational Science and Engineering)
YONSEI UNIVERSITY

Scientific Computation

What is scientific computation?

- The process of using mathematical algorithms and computer technology to solve scientific problems



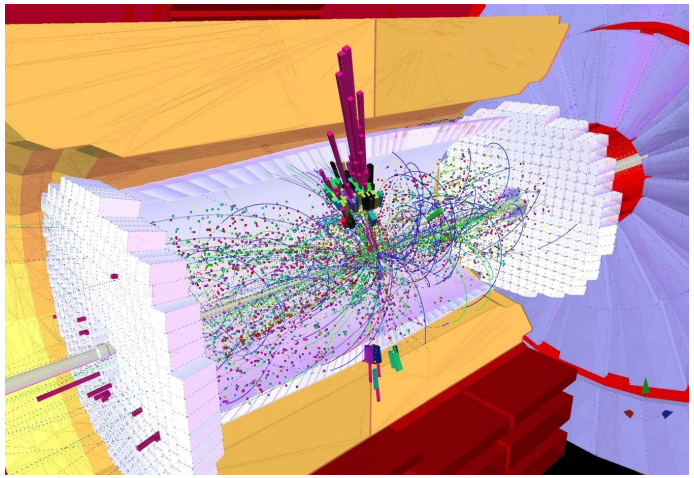
Scientific Computation

Why do we need to learn scientific computation?

- Through the **validated model and verified algorithm**, scientific computation allows

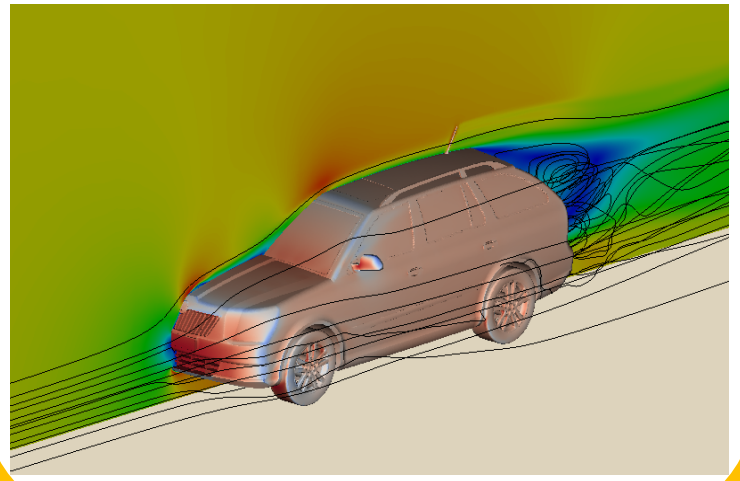
Science

scientists to simulate complex systems and processes that would be difficult or impossible to study through observation or experiment alone



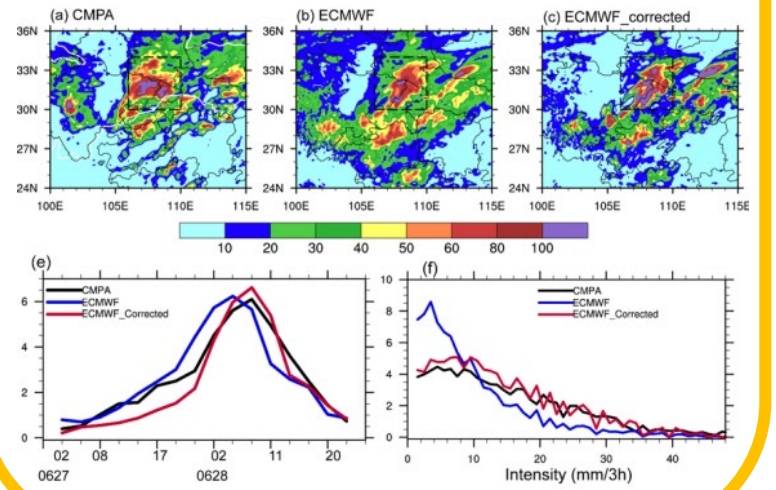
Engineering

engineers to simulate various conditions with various parameters so that they could find optimal conditions while reducing economical and environmental costs



Our lives

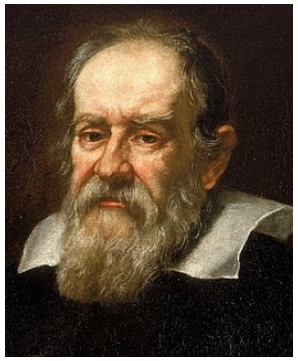
us to prepare in advance for issues that pose significant risks to our lives, such as natural disasters or explosions by simulating those kind of situations

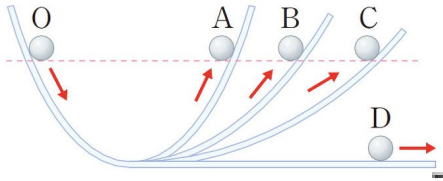


Scientific Computation

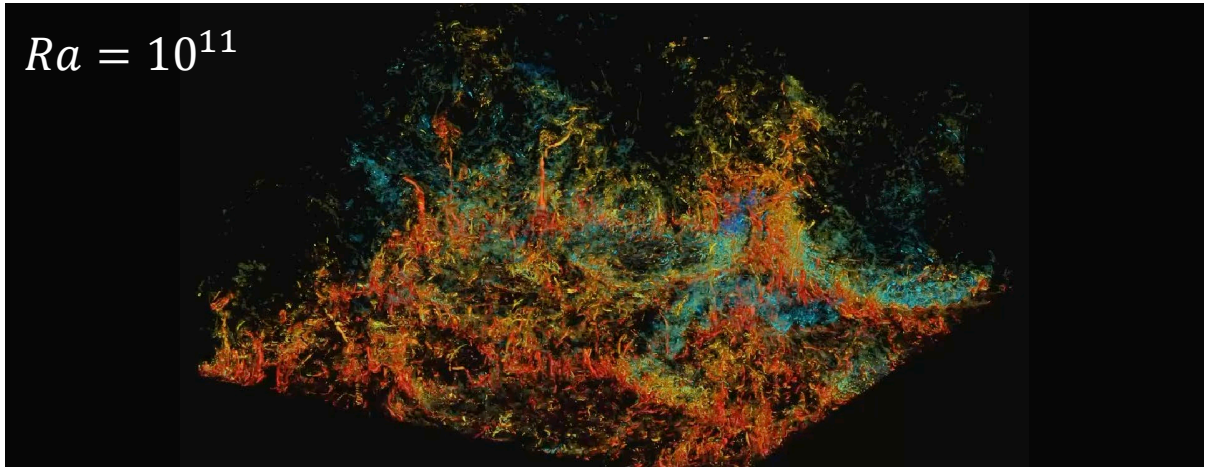
Why do we need scientific computation?

- Advancing science and theory : more macro/microscopic systems

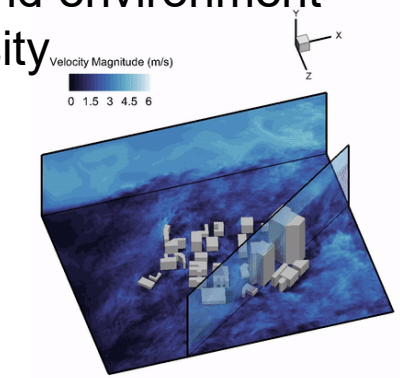


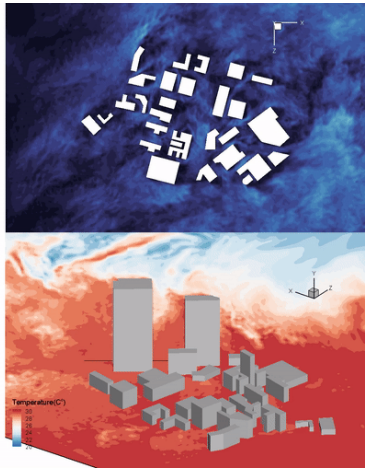


Galileo Galilei's thought experiment



Simulation of the wind environment near Yonsei University



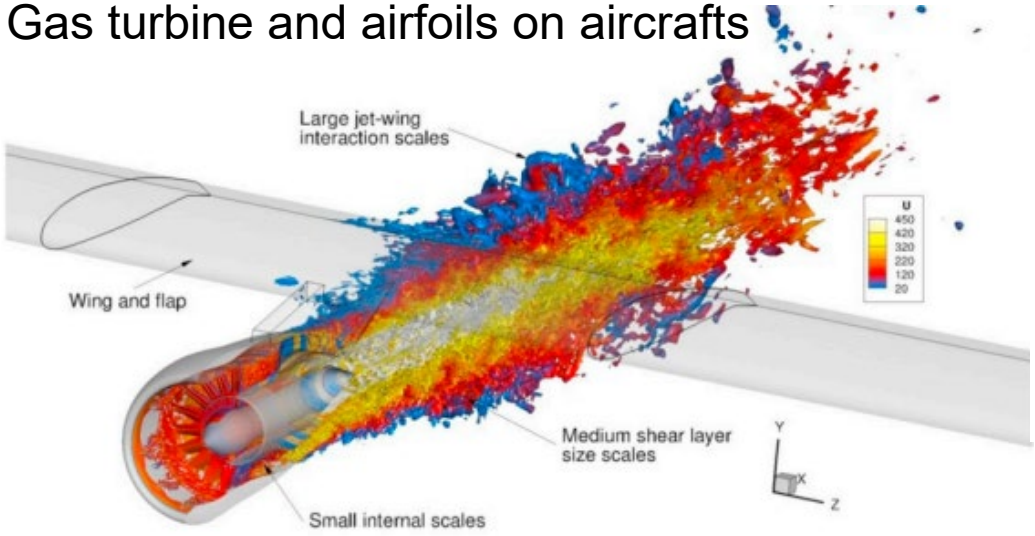


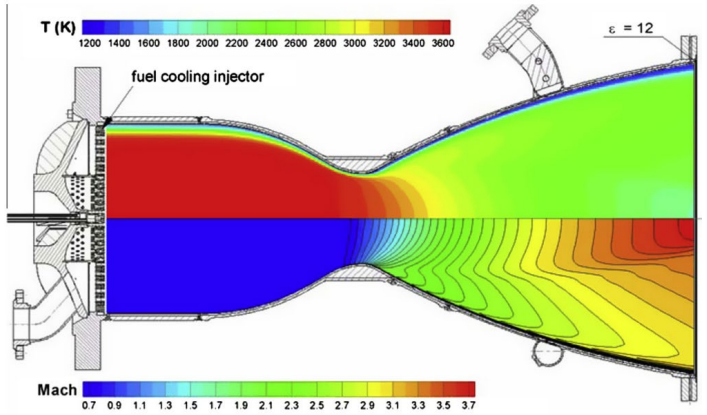
Scientific Computation

Why do we need scientific computation?

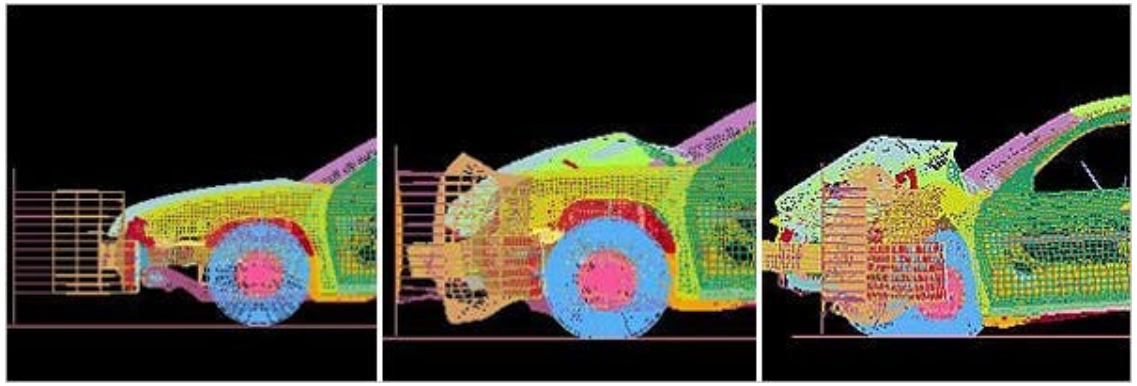
- Application to engineering

Gas turbine and airfoils on aircrafts

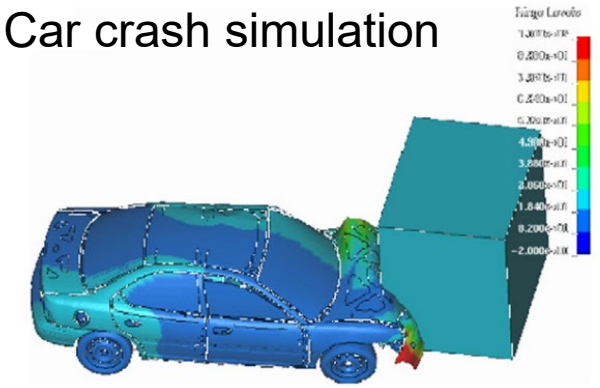


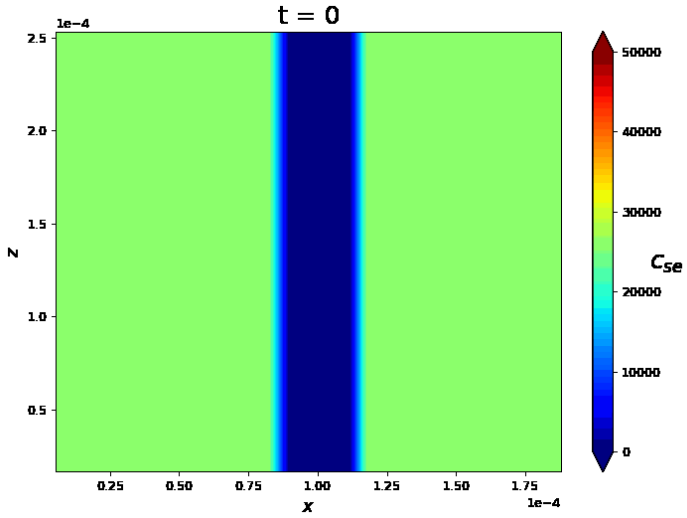


Rocket propulsion



Car crash simulation





Lithium-ion concentration in electrode

Scientific Computation

Why do we need scientific computation?

- Sustaining our lives and environment

The 17 Goals

Adopted by all UN Member States in 2015.
Part of the 2030 Agenda for Sustainable Development.

Environmental fluid mechanics

Substantial role in SDGs no. 6, 7, 9, 11, 13, and 14.

Sustainable cities & communities

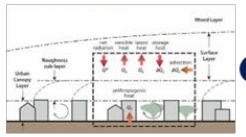
make cities and human settlements inclusive, safe, resilient and sustainable.

Scientific Computation

Why do we need scientific computation?

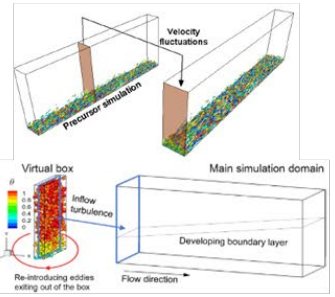
- Sustaining our lives and environment

Radiation



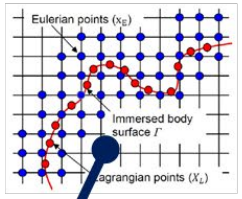
Inflow Generation

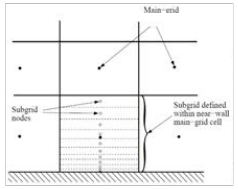
- Precursor Method
- Synthetic Eddy Method

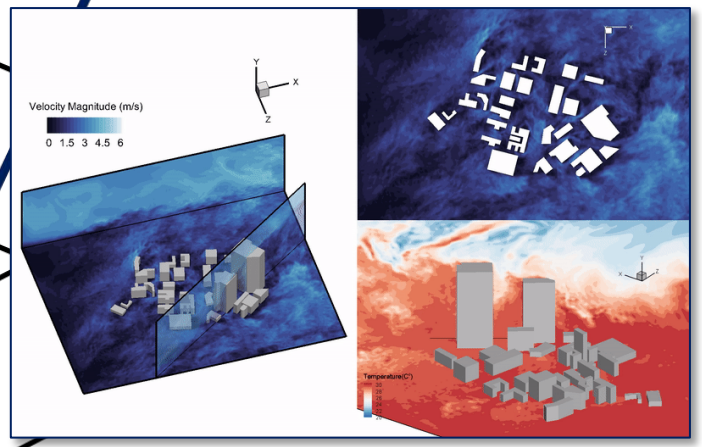


Immersed Boundary Method

- Cell Classification
- GIS data
- Wall-model

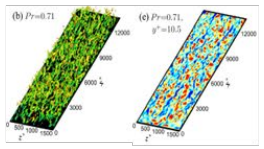






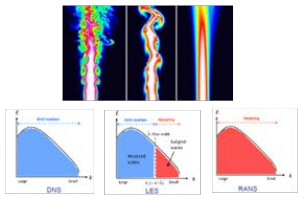
Turbulent Thermal Boundary Layer

- Navier-Stokes eqn
- Energy eqn
- Fractional Step Methods

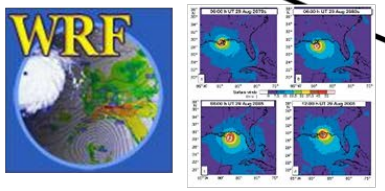


Large Eddy Simulation

- Const. & Dyna. Smagorinsky
- Const. & Dyna. Vreman



Weather Research & Forecasting



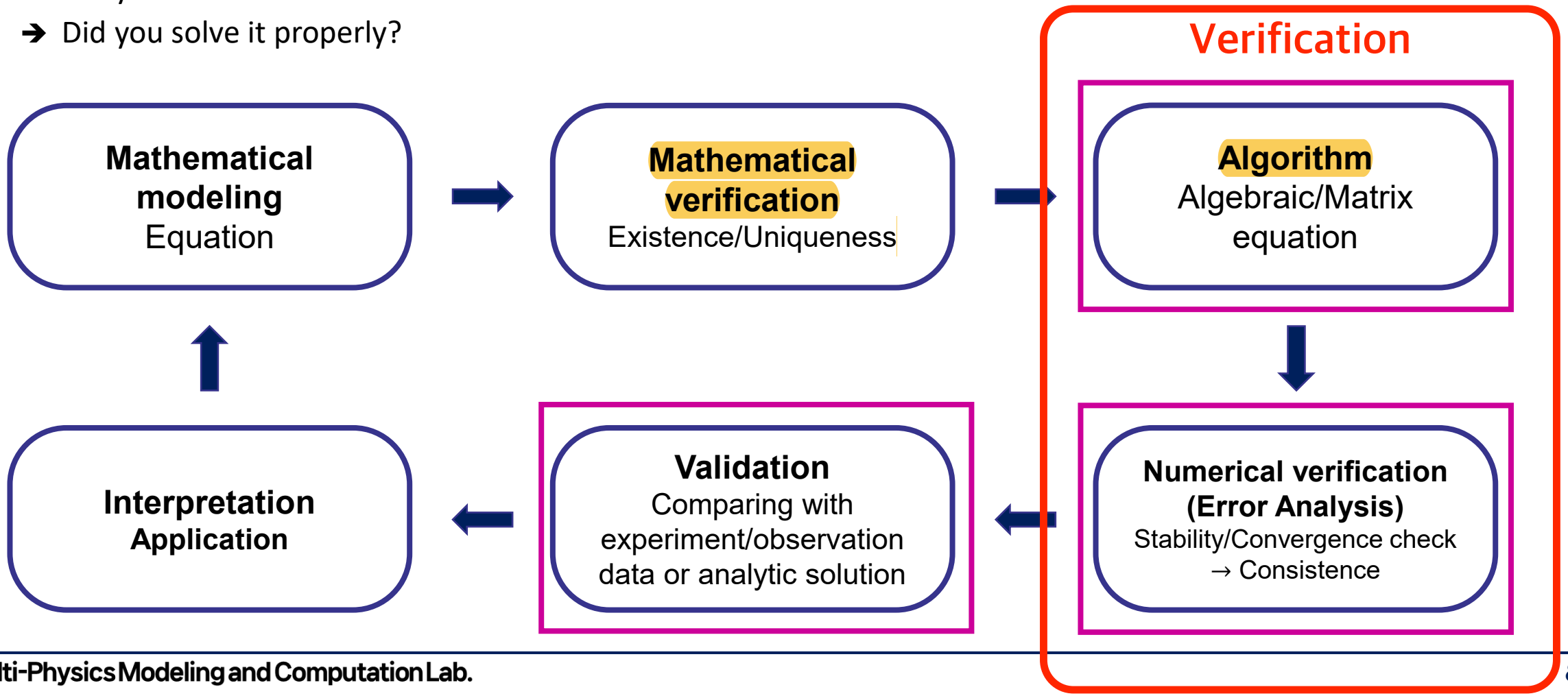
High Performance Computing

- MPI/CUDA
- Parallelized Algorithm

Scientific Computation

Verification and validation(V&V) : Numerical simulation into science

- To bring numerical simulation into the realm of 'scientific' computation, two questions must be asked.
  - ➔ Did you solve it well?
  - ➔ Did you solve it properly?



Scientific Computation

Solving equation using computer

Governing equations

Continuity Eqn.

$$\nabla \cdot \boldsymbol{u} = 0$$

Navier-Stokes Eqn.

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\nabla p + \frac{1}{Re}\nabla^2 \boldsymbol{u} + F_b$$

Energy Eqn.

$$\frac{\partial T}{\partial t} + \boldsymbol{u} \cdot \nabla \mathrm{T} = \frac{1}{RePr}\nabla^2 T$$

Novel methods

Approximate LU Decomposition
Numerical Analysis schemes
Multigrid method
FFT
Novel TDMA solver
Parallel computing skill(MPI, CUDA)…
etc.

$$\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n}}{\triangle t}+\frac{1}{2}\left(\mathbf{u}^{n+1}\cdot\mathcal{G}\mathbf{u}^{n+1}+\mathbf{u}^{n}\cdot\mathcal{G}\mathbf{u}^{n}\right)=-\mathcal{G}p^{n+\frac{1}{2}}+\frac{1}{2}\sqrt{\frac{Pr}{Ra}}\left(\mathcal{L}\mathbf{u}^{n+1}+\mathcal{L}\mathbf{u}^{n}\right)+\mathbf{F}(\theta^{n+\frac{1}{2}})+\mathbf{mbc}_{\mathrm{CN}}^{n+\frac{1}{2}}$$

$$\mathcal{D}\mathbf{u}^{n+1}=cbc_{\mathrm{CN}}^{n+1},$$

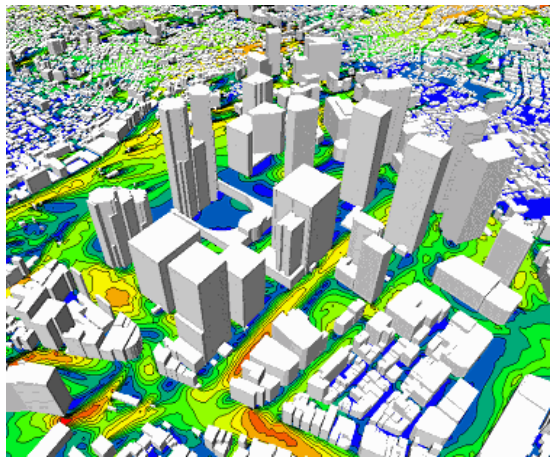
$$\frac{\theta^{n+\frac{1}{2}}-\theta^{n-\frac{1}{2}}}{\triangle t}+\frac{1}{2}\mathbf{u}^{n}\cdot\left(\mathcal{G}\theta^{n+\frac{1}{2}}+\mathcal{G}\theta^{n-\frac{1}{2}}\right)=\frac{1}{2\sqrt{RaPr}}\left(\mathcal{L}\theta^{n+\frac{1}{2}}+\mathcal{L}\theta^{n-\frac{1}{2}}\right)+cbc_{\mathrm{CN}}^{n},$$

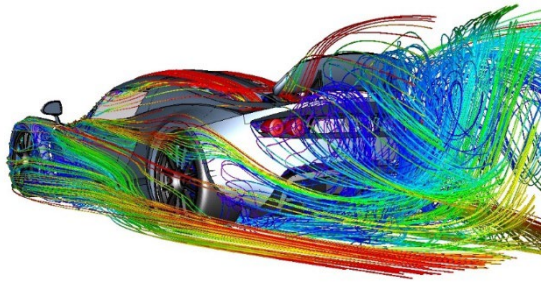
$$\begin{pmatrix}\mathcal{A}_{\theta}^{n-\frac{1}{2}} & \mathcal{O} & \mathcal{O}\\ -\mathbf{F} & \mathbb{A}^{n} & \mathcal{G}\\ \mathcal{O} & \mathcal{D} & \mathcal{O}\end{pmatrix}\begin{pmatrix}\theta^{n+\frac{1}{2}}\\ \mathbf{u}^{n+1}\\ p^{n+\frac{1}{2}}\end{pmatrix}=\begin{pmatrix}\mathcal{A}_{\theta}^{n-\frac{1}{2}} & \mathcal{O} & \mathcal{O}\\ -\mathbf{F} & \mathbb{A}^{n} & \mathcal{O}\\ \mathcal{O} & \mathcal{D} & -\triangle t\mathcal{D}\mathcal{G}\end{pmatrix}\begin{pmatrix}\mathcal{I} & \mathcal{O} & \mathcal{O}\\ \mathcal{O} & \mathbb{I} & \triangle t\mathcal{G}\\ \mathcal{O} & \mathcal{O} & \mathcal{I}\end{pmatrix}\begin{pmatrix}\theta^{n+\frac{1}{2}}\\ \mathbf{u}^{n+1}\\ p^{n+\frac{1}{2}}\end{pmatrix}$$

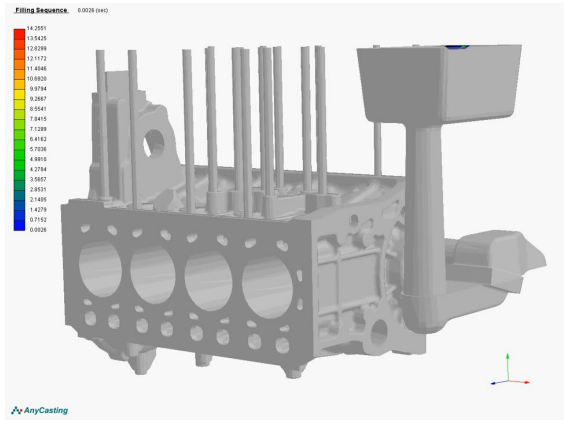
$$\mathbb{A}^{n}\approx\frac{1}{\triangle t}\begin{pmatrix}\prod_{k=1}^{3}(\mathcal{I}+\triangle t(\mathbb{M}_{11}^{n})^{k}) & \mathcal{O} & \mathcal{O}\\ \triangle t\mathbb{M}_{21}^{n} & \prod_{k=1}^{3}(\mathcal{I}+\triangle t(\mathbb{M}_{22}^{n})^{k}) & \mathcal{O}\\ \triangle t\mathbb{M}_{31}^{n} & \triangle t\mathbb{M}_{32}^{n} & \prod_{k=1}^{3}(\mathcal{I}+\triangle t(\mathbb{M}_{33}^{n})^{k})\end{pmatrix}\begin{pmatrix}\mathcal{I} & \triangle t\mathbb{M}_{12}^{n} & \triangle t\mathbb{M}_{13}^{n}\\ \mathcal{O} & \mathcal{I} & \triangle t\mathbb{M}_{23}^{n}\\ \mathcal{O} & \mathcal{O} & \mathcal{I}\end{pmatrix}$$

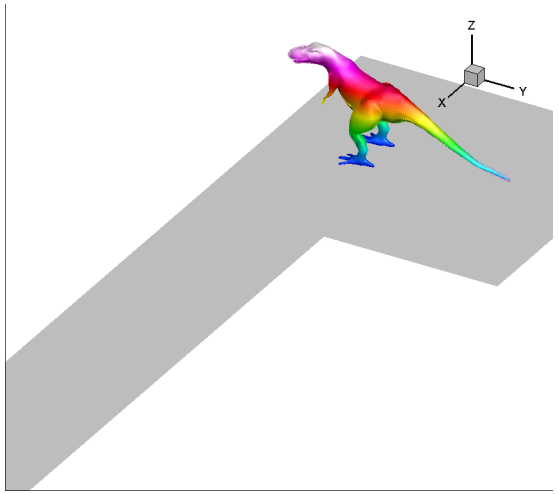
Efficient, reliable, large-scale, and fast computation for real problems

Applications





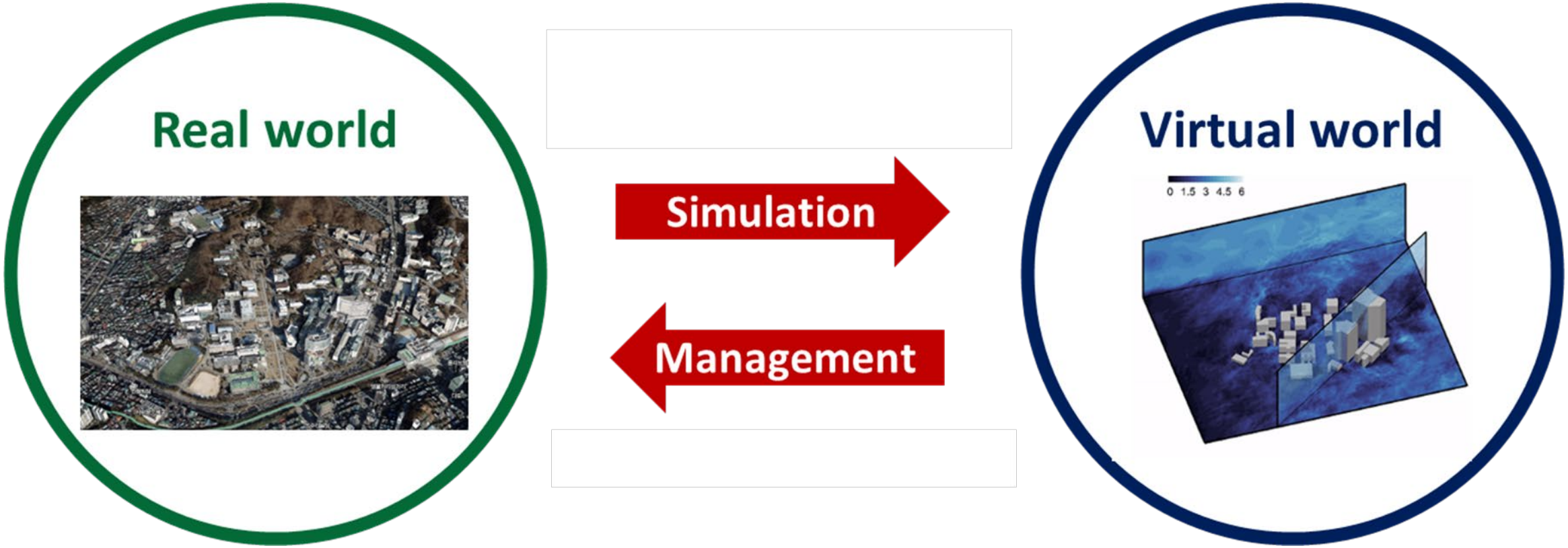




Scientific Computation

Why do we need a faster computer?

- For real-time application to the real world, high-speed calculation is required (Digital Twin)



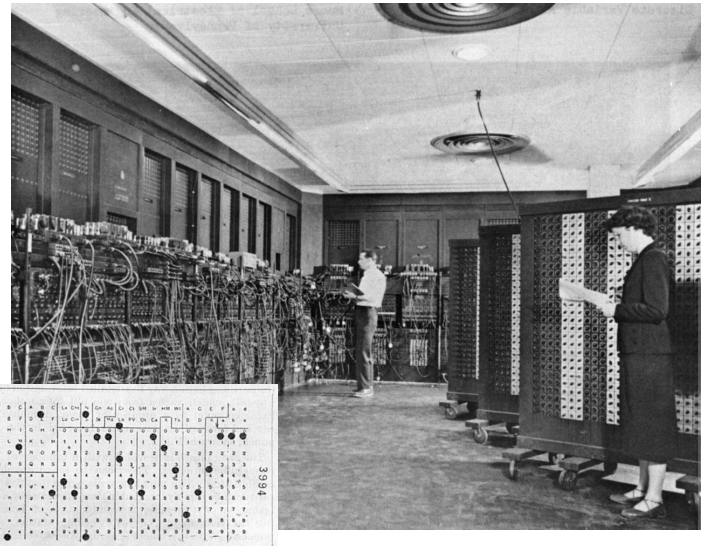
Scientific Computation

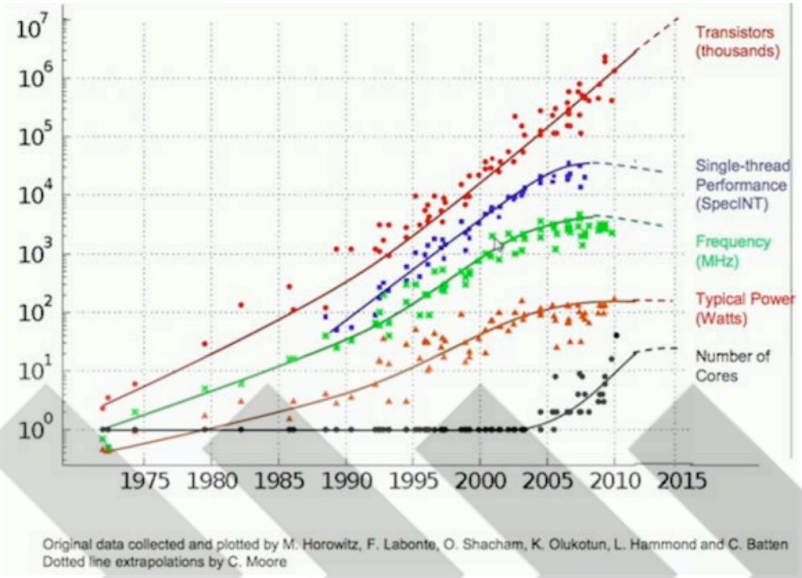
Why do we need a faster computer?

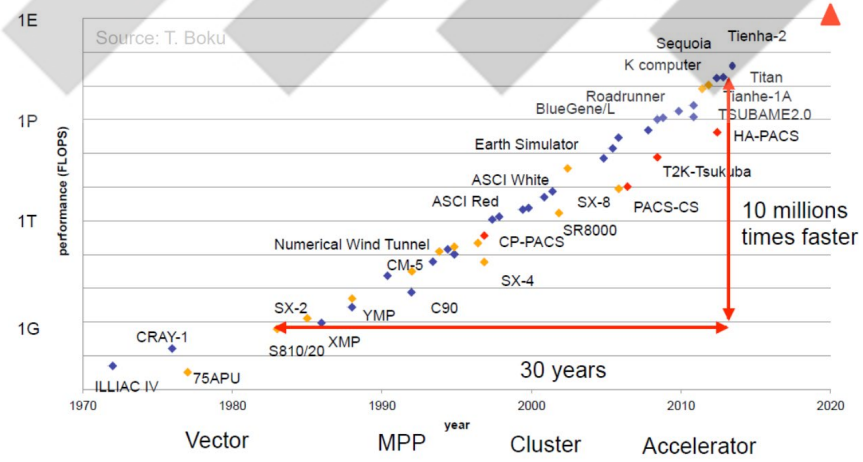
Challenges	Enabling Technologies
Data management, data privacy and security, data quality	Digital platforms, cryptography and blockchain technologies, big data technologies
Real-time communication of data and latency	Data compression, communication technologies like 5G and internet of things technologies
Physical realism and future projections	Sensor technologies, high fidelity physics-based simulators, data-driven models
Real time modeling	Hybrid analysis and modeling, reduced order modeling, multivariate data-driven models
Continuous model updates, modeling the unknown	Big data cybernetics, hybrid analysis and modeling, data assimilation, compressed sensing and symbolic regression
Transperancy and interpretability	Hybrid analysis and modeling, explainable artificial intelligence
Large scale computation	Computational infrastructure, edge, fog and cloud computing
Interaction with physical asset	Human machine interface, natural language processing, visualization augmented reality and virtual reality

Computer & Language

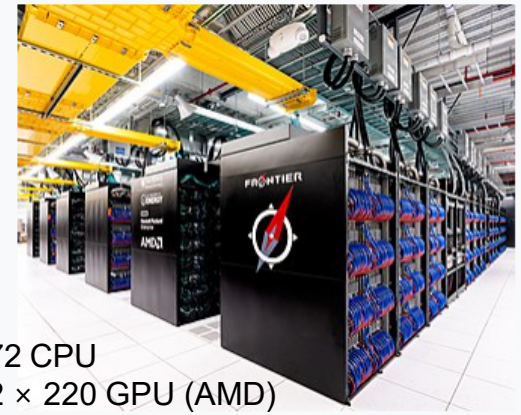
Evolution of comptuers







Frontier



591,872 CPU
36,992 × 220 GPU (AMD)

Active	Deployment: Sep. 2021 Completion: May 2022
Operators	Oak Ridge National Laboratory and U.S. Department of Energy
Location	Oak Ridge Leadership Computing Facility
Power	21 MW
Operating system	HPE Cray OS
Space	680 m^2 (7,300 sq ft)
Speed	1.102 exaFLOPS (Rmax) / 1.685 exaFLOPS (Rpeak)[1]
Cost	US$600 million (estimated cost)
Purpose	Scientific research and development
Website	www.olcf.ornl.gov/frontier/

Computer & Language

Development of programming language

- Machine language
  - ➔ Language written in bits that the hardware can decode and execute directly
- Assembly language
  - ➔ More readable than machine language but still closely tied to the instruction set architecture of the specific computer
- High-level programming language
  - ➔ an easy(?)-to-understand programming language without relying on computer architecture

High Level Language	Assembly Language	Maching Language
`#include <stdio.h>`	`MOV R0, 4 ;`	`11010101101001`
	`MOV R1, 5 ;`	`01001010101100`
`int main(){`	`ADD R0, R0, 2 ;`	`10101010010100`
`int a=4,b=5,c;`	`ADD R1, R1, 3 ;`	`11010101000101`
`a+=2; b+=3;`	`SUB R2, R0, R1 ;`	`10101101010101`
`c=a-b;`		`11001010010101`
		`11010101000101`
`return 0;`		`10101101010101`
`}`		

Computer & Language

Development of programming language

- Evolution of programming language

  ➔ an easy(?)-to-understand programming language without relying on computer architecture

  - ✓ FORTRAN(FORmular TRANslation, 1957) : The first successful advanced language, mainly for scientific and technological calculations
  - ✓ ALGOL60(ALGOritmic Language 60, 1959): Languages that greatly influenced the development of later languages such as **pascal**, **C language**, etc
  - ✓ COBOL(COmmon Business Oriented Language, 1960) : Language Created for Office Processing
  - ✓ C (1972) : language that provides access to computer structures using **pointer**s, used for programming computer systems
  - ✓ C++ (1983) : Expanded C language with **object orientation**
  - ✓ Python (1991) : **Interpreter-style, object-oriented, interactive language**

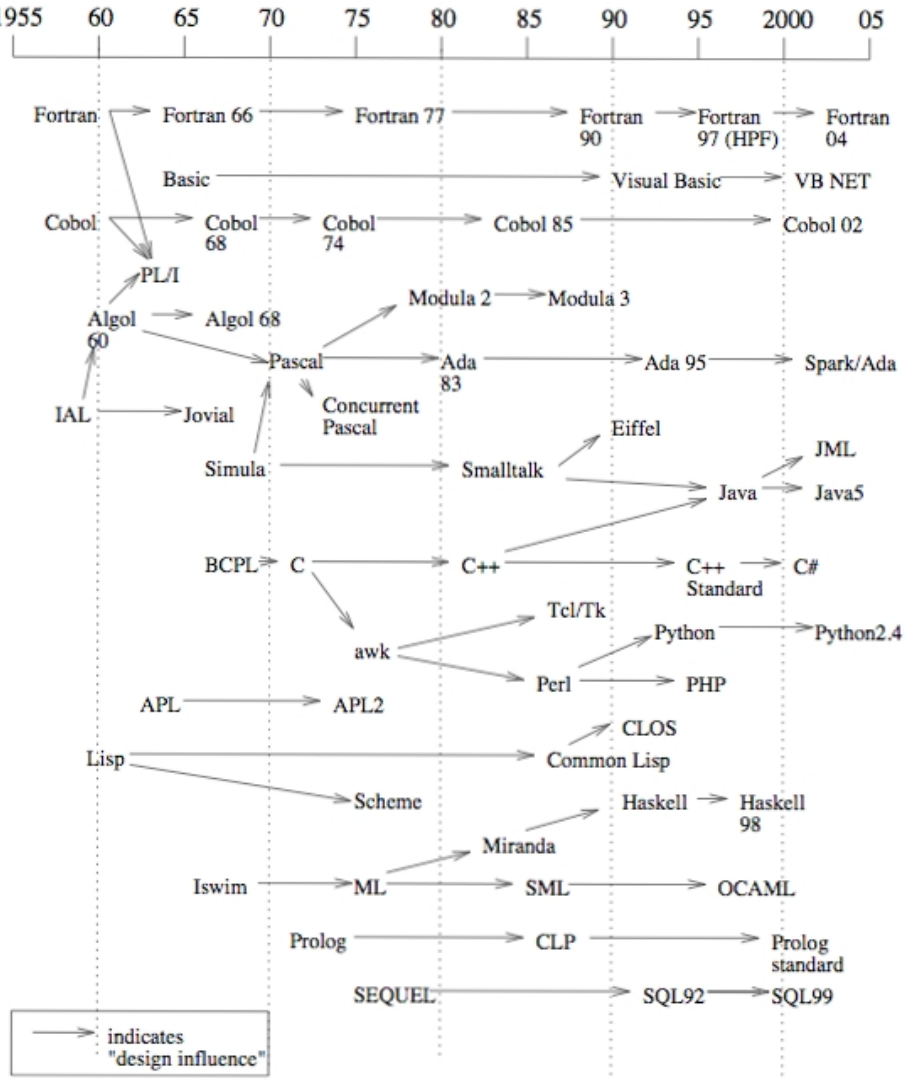


Figure 1.2: A Snapshot of Programming Language History

Python

A Brief History and Current Status

- Python was created in late 1980s by Guido van Rossum and first appeared 1991
- Python 2.0 was released in 2000 with major new features.
- Python 3.0 was released in 2008 with features backported to Python 2.6.x and 2.7.x.
- Python 2.7's end-of-life was postponed to 2020.
- Python 3.9.2 and 3.8.8 were expedited in 2021 due to security issues.
- Python 3.11.0 is the current stable releases with increased program execution speed and improved error reporting.
- The older series, including 3.9, 3.8 and 3.7, will only receive security fixes in the future.
- Four new releases were made in September 2022 due to a potential denial-of-service attack: 3.10.7, 3.9.14, 3.8.14, and 3.7.14.

Guido van Rossum
Creator of Python programming language.

Python

Why do we use Python in this class?

- Simple and clear syntax
- Huge and excellent library
- Good support for interfacing C, C++, and Fortran (F2py)
- Python is available for FREE!

Comparison to languages

- Cubic spline code Python vs MATLAB vs Fortran

```
>>> import numpy as np
>>> from scipy.interpolate import CubicSpline
>>> import matplotlib.pyplot as plt
>>> x = np.arange(10)
>>> y = np.sin(x)
>>> cs = CubicSpline(x, y)
```

Python

Interactive language

```
x = [0 1 2.5 3.6 5 7 8.1 10];
y = sin(x);
xx = 0:.25:10;
yy = spline(x,y,xx);
```

MATLAB

Fortran

```
program cubic_spline

implicit none

integer     :: i
real*8      :: x(10), y(10)
real*8      :: xx, yy

do i = 1,10
   x(i) = dble(i)
   y(i) = sin(x(i))
enddo

xx = 6.5
call spline(x, y, 10, 0.0d0, 0.0d0, xx, yy)

end program cubic_spline

subroutine spline(x_data, y_data, n, g_0, g_n, x, cs)
   implicit none
   integer, intent(in) :: n
   integer :: i, j
   real*8, intent(in) :: x, g_0, g_n
   real*8, dimension(0:n), intent(in) :: x_data, y_data
   real*8, dimension(0:n) :: RHS, LHS
   real*8, dimension(0:n) ::
   real*8, dimension(1:n-1,
   real*8, intent(out) :: cs

   delta(0)=x_data(1)-x_data(
   do i=0, n-1
      delta(i)=x_data(i+1)-x
   end do
   delta(n)=x_data(n)-x_data(

   !Make TDM
   do i=1, n-1
      TDM(i, i)=(delta(i+1)+
   end do
   do i=1, n-2
      TDM(i, i+1)=delta(i)/(
      TDM(i+1, i)=delta(i)/(
   end do


   RHS(1)=(y_data(n)-y_data(n
   do i=1, n-1
      RHS(i)=(y_data(i+1)-y_
   end do

   RHS(0)=y_data(0)
   RHS(1)=RHS(1)-delta(0)/6.*g_0
   RHS(n-1)=RHS(n-1)-delta(n)/6.*g_n
   RHS(n)=y_data(n)

   LHS(0)=g_0
   call TDMA(TDM, RHS(1:n-1), 1, n-1, LHS(1:n-1))
   LHS(n)=g_n

   do i=0, n-1
      if ((x .ge. x_data(i)).and.(x.le.x_data(i+1))) then
         cs=(LHS(i)/6)*((x_data(i+1)-x)**3/delta(i)-delta(i)*(x_data(i+1)-x))&
         +(LHS(i+1)/6)*((x-x_data(i))**3/delta(i)-delta(i)*(x-x_data(i)))&
         +y_data(i)*(x_data(i+1)-x)/delta(i)+y_data(i+1)*(x-x_data(i))/delta(i)
      end if
   end do

   return
end subroutine spline

subroutine TDMA(TDM, RHS, bs, bf, LHS)
   implicit none
   integer, intent(in):: bs, bf
   real*8, dimension(bs:bf, bs:bf), intent(in) :: TDM
   real*8, dimension(bs:bf), intent(in)  :: RHS
   real*8, dimension(bs:bf), intent(out) :: LHS
   real*8, dimension(bs:bf) :: b, SOL
   real*8, dimension(bs+1:bf) :: a
   real*8, dimension(bs:bf-1) :: c
   integer :: i

   SOL=RHS
   do i=bs, bf-1
      a(i+1)=TDM(i+1, i)
      b(i)=TDM(i, i)
      c(i)=TDM(i, i+1)
   end do
   b(bf)=TDM(bf,bf)
   c(bs)=c(bs)/b(bs)
   SOL(bs)=SOL(bs)/b(bs)
   do i=bs+1, bf-1
      c(i)=c(i)/(b(i)-a(i)*c(i-1))
      SOL(i)=(SOL(i)-a(i)*SOL(i-1))/(b(i)-a(i)*c(i-1))
   end do
   SOL(bf)=(SOL(bf)-a(bf)*SOL(bf-1))/(b(bf)-a(bf)*c(bf-1))
   do i=bf-1, bs, -1
      SOL(i)=SOL(i)-c(i)*SOL(i+1)
   end do
   LHS=SOL

   return
end subroutine
```

Python

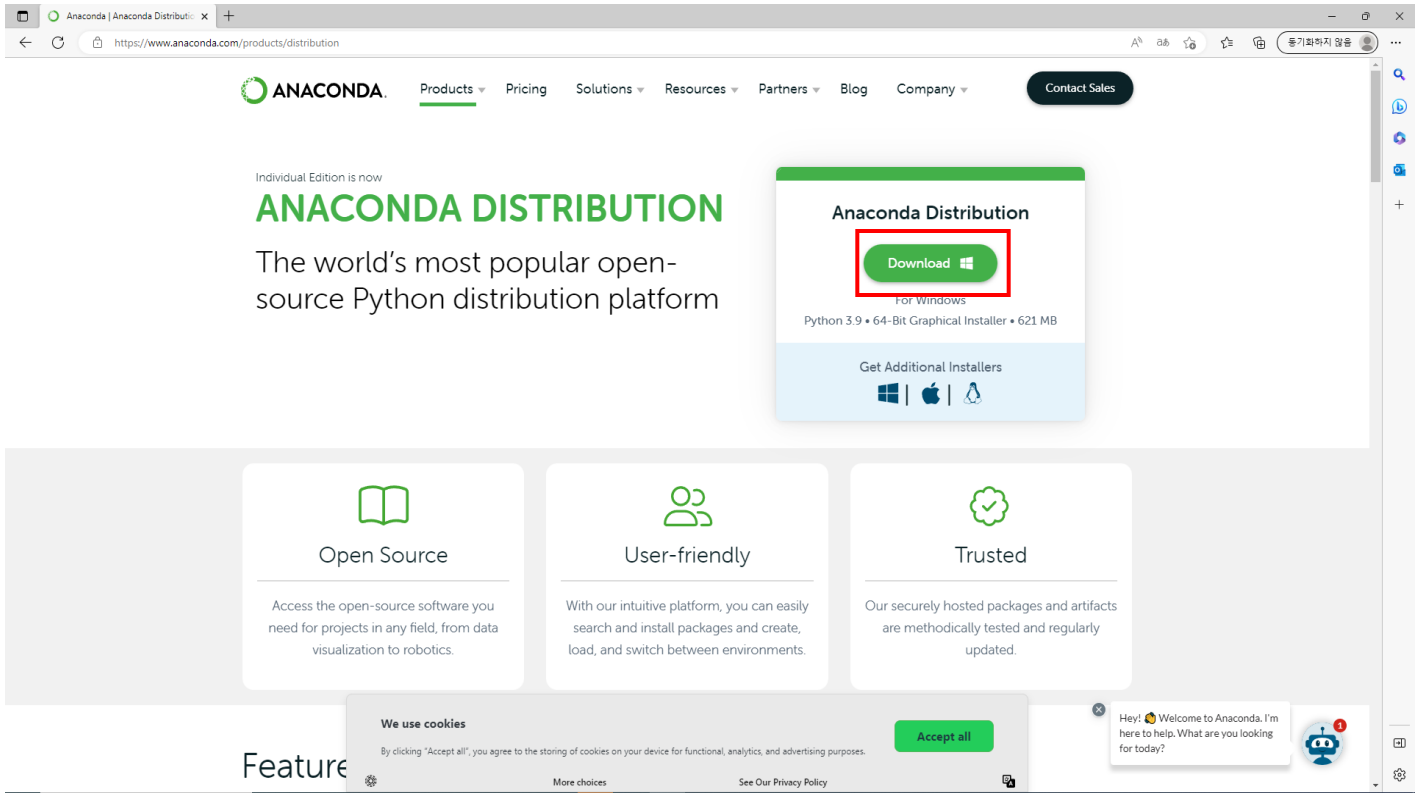
How to build environment for python

- There are many IDE(Integrated Development Environment) to programming with Python
  ➔ ex) Jupyter, IDLE, Pycharm, VScode….
- You can use whatever you prefer for your purpose when it comes to IDEs.
- Here, explain how to use Jupyter notebook.

Python

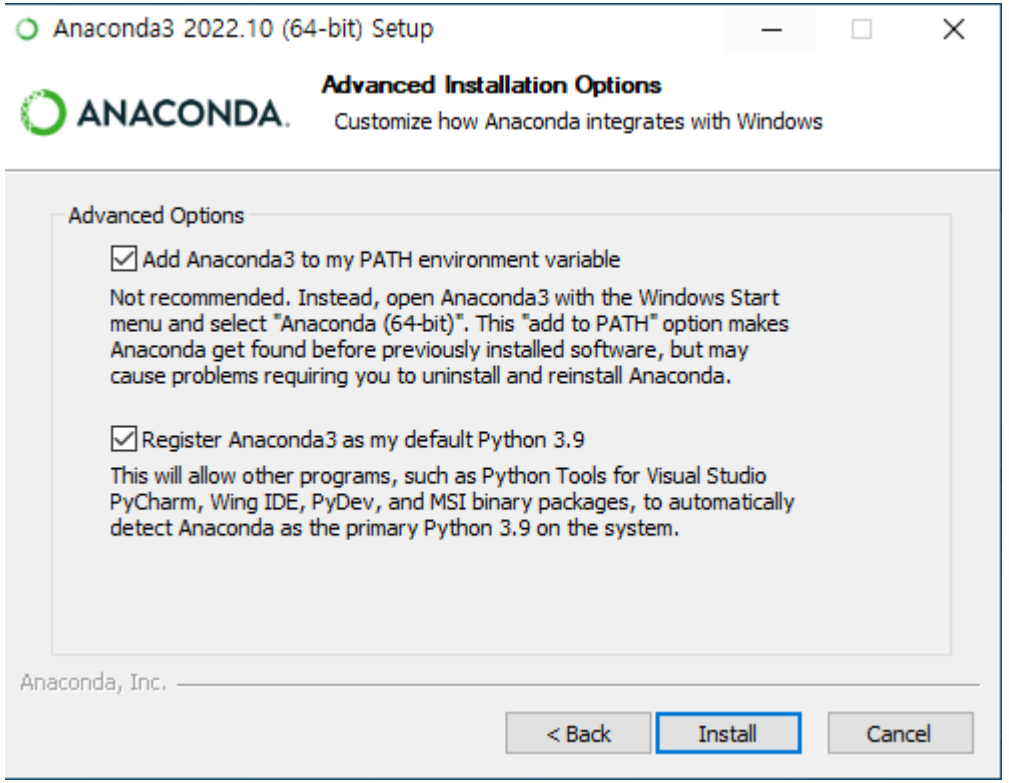
1. Download Anaconda3

- Anaconda3 is a distribution of the Python for sciectific computing, that aims to simplify package management and deployment.
- https://www.anaconda.com/products/distribution



Python

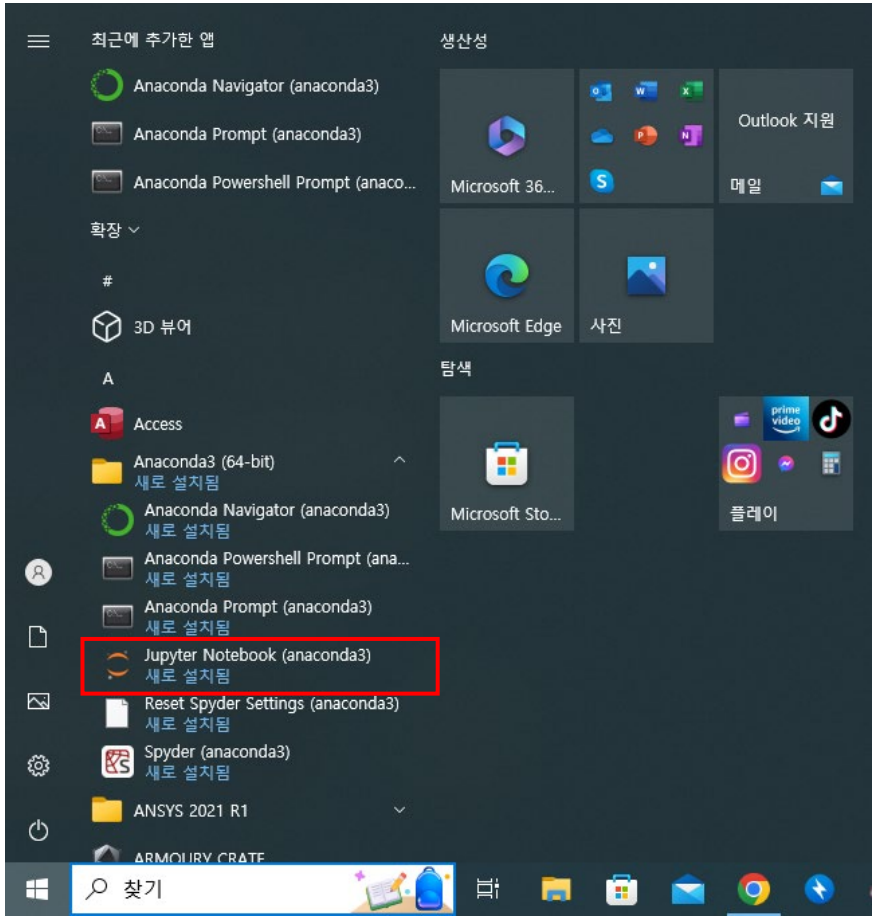
1. Download Anaconda3



Python

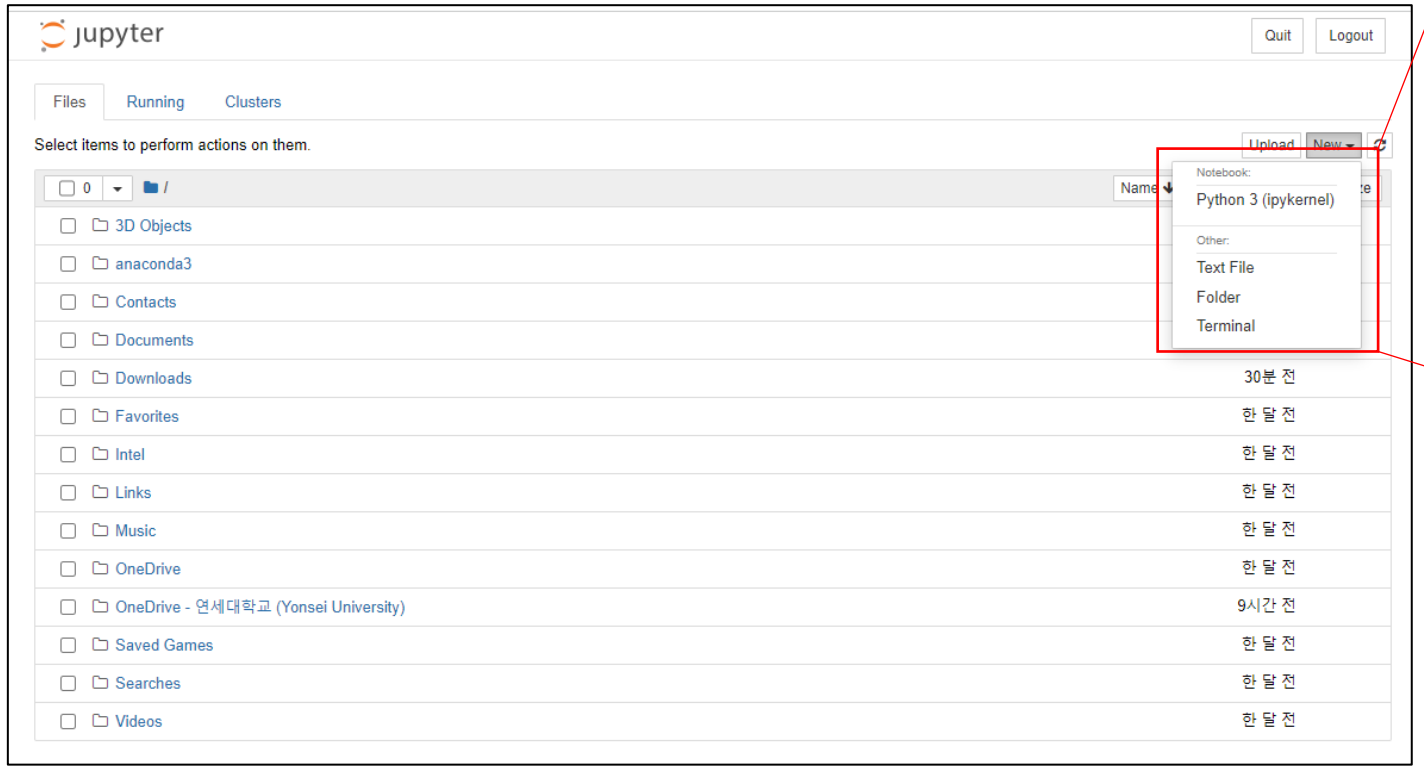
1. Download Anaconda3

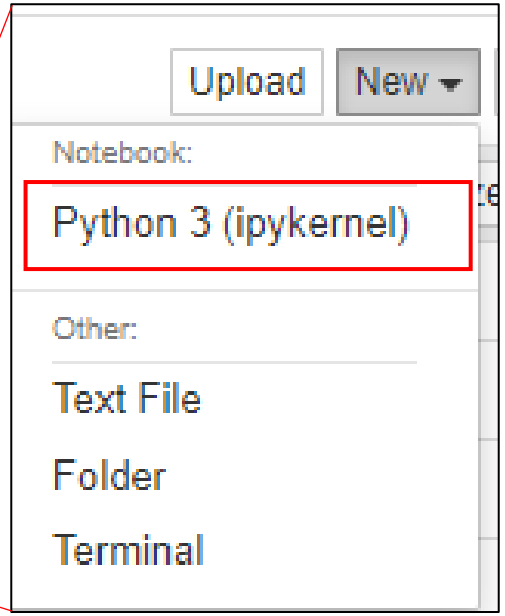
- If you correctly downloaded Anaconda3, there is Jupyter Notebook in menu.



Python

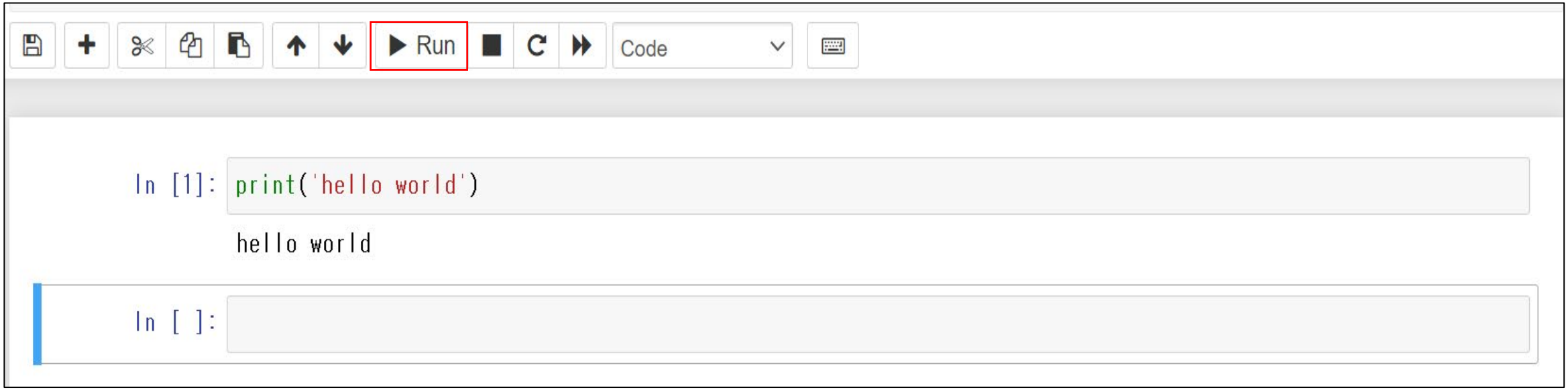
2. Run Jupyter Notebook and create file.





Python

3. Print “hello world”



Q&A

Thanks for listening