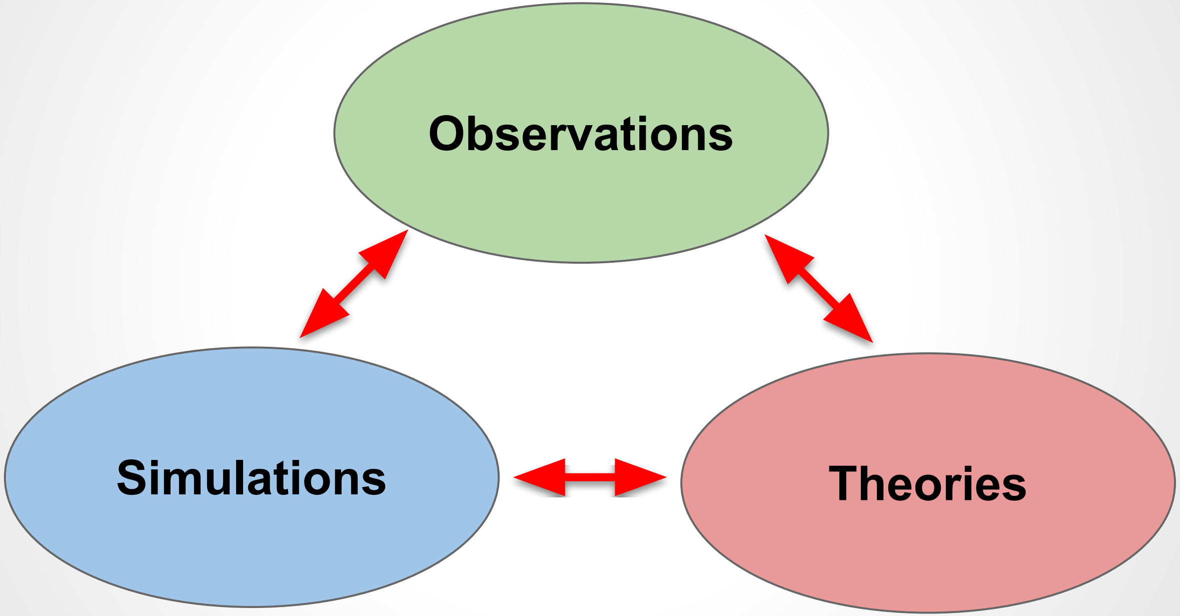
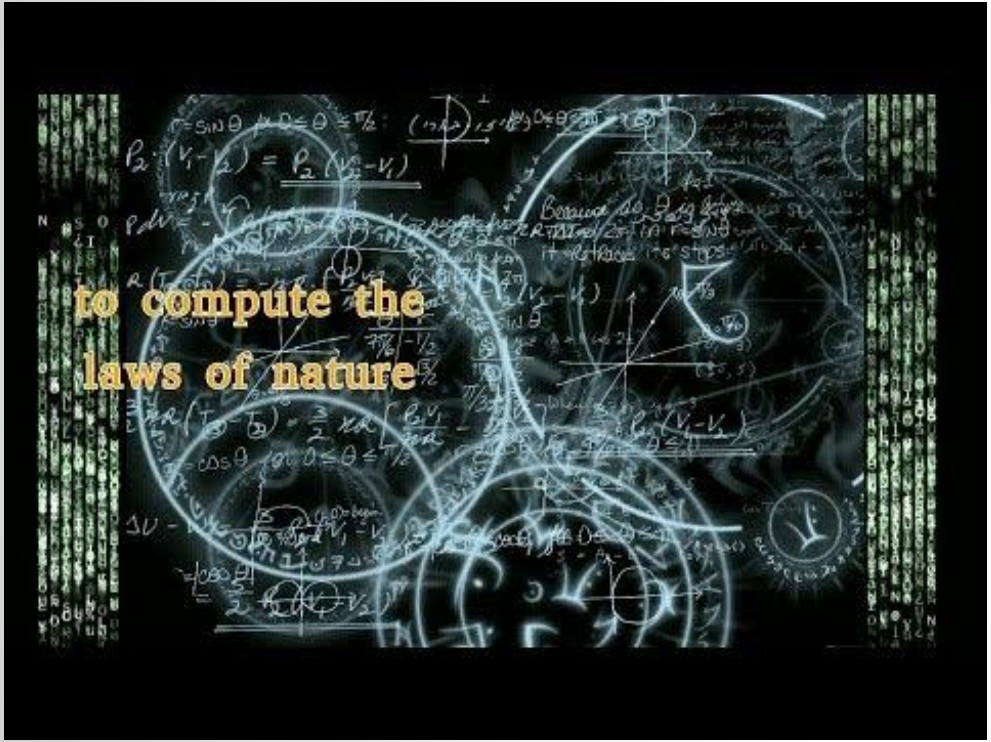
Computational Astrophysics (108-2)

Hsi-Yu Schive（薛熙于）
National Taiwan University

Why Simulations?



Example: Large-Scale Structure of the Universe



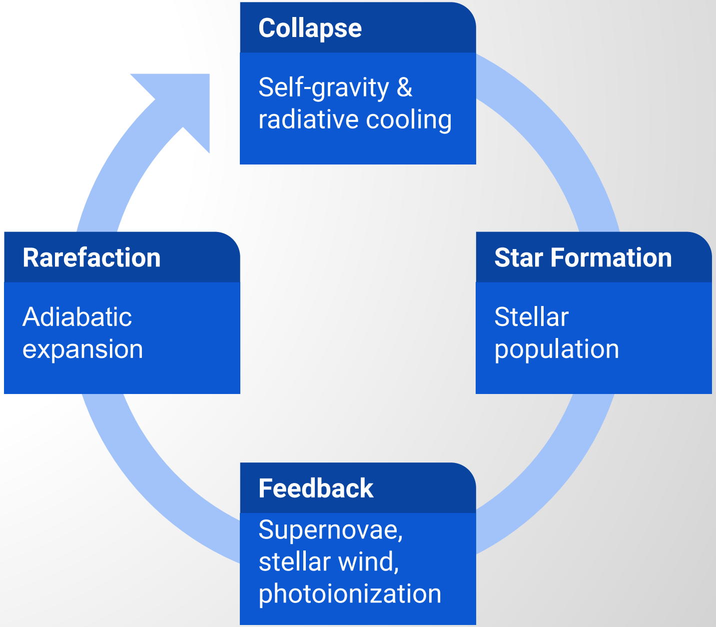
Credit: TNG Collaboration

- Illustris cosmological simulation
  - Create mock observations
- Time: 0.3 Myr ~ 13.8 Gyr
- Size: 106.5 Mpc^3
- ~10^{10} particles & hydro cells
  - DM mass resolution ~ $10^6\ M_\odot$
- Spatial resolution ~ 1 kpc
  - Cover 10^5 spatial range
- Computing resource: 8192 CPU cores
- Computing time: 1.9×10^7 CPU hours
  - ~3 months with 8192 CPU cores
  - ~2000 years on a single PC
- Successor: Illustris TNG
  - https://www.tng-project.org

Example: Milky-Way-Like Galaxy

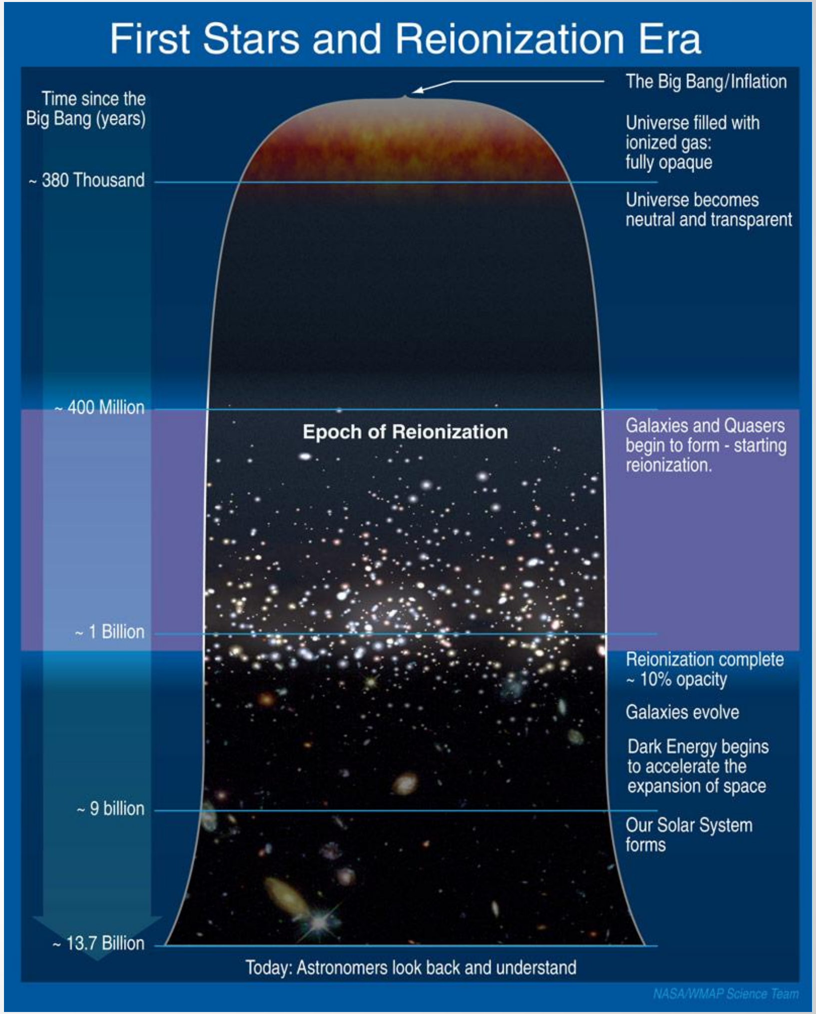
Credit: Advanced Visualization Laboratory at NCSA

- **Isolated disk galaxy simulation**
  - **Similar to our Milky Way**
- **Physics cycle**



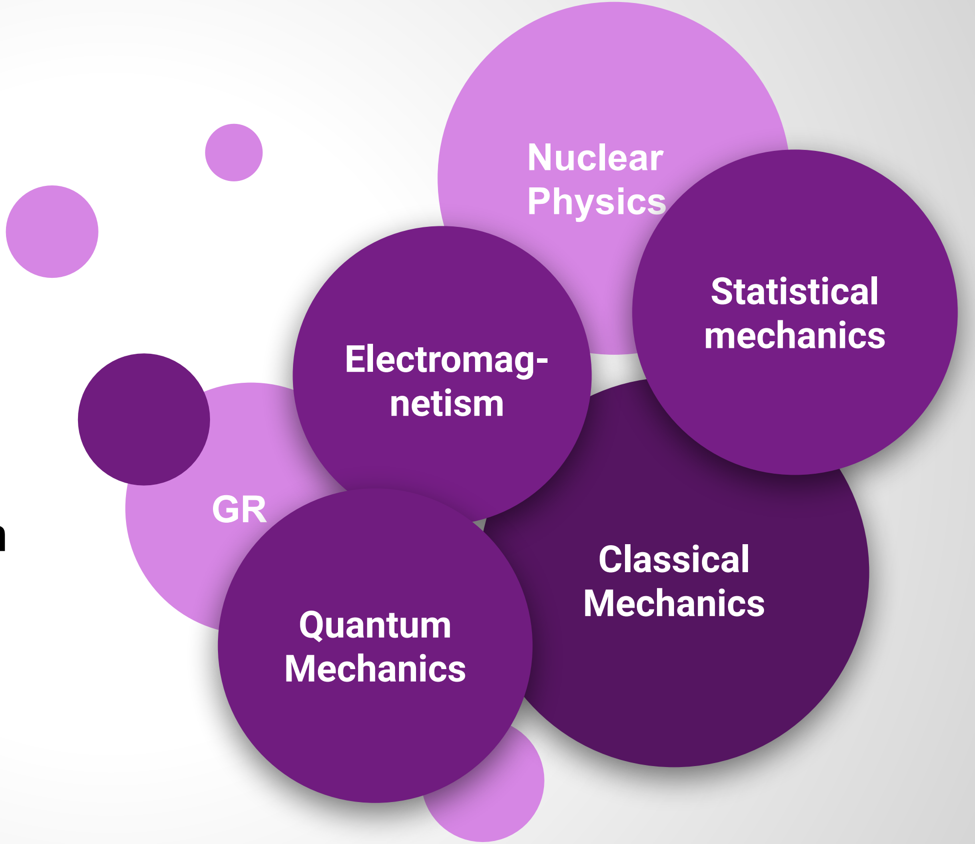
Example: First Stars

Credit: Renaissance Simulations Laboratory, Advanced Visualization Laboratory at NCSA



Key Physics

- Dark matter
- Hydrodynamics
- Self-gravity
- Magnetic field
- Chemistry
- Radiation transfer
  - Cooling, ionization, etc
- Star formation and evolution
- Feedback
  - Supernovae explosion
  - Stellar wind
  - SMBH/AGN jets
  - ...



Key Techniques

- Numerical algorithms
- Parallel computing
  - CPU/GPU parallelization
- Code co-development
- Data analysis and visualization
- Debugging
- Data sharing

Syllabus

週次	日期	單元主題
第1週	03/03	Introduction
第2週	03/10	Initial Value Problems
第3週	03/17	Computational Hydrodynamics I
第4週	03/24	Computational Hydrodynamics II
第5週	03/31	Boundary Value Problems
第6週	04/07	Discrete Fourier Analysis
第7週	04/14	N-body: Gravity Evaluation
第8週	04/21	N-body: Orbit Integration
第9週	04/28	HPC: OpenMP & MPI Parallelization I
第10週	05/05	HPC: OpenMP & MPI Parallelization II
第11週	05/12	HPC: OpenMP & MPI Parallelization III
第12週	05/19	HPC: GPU Programming I
第13週	05/26	HPC: GPU Programming II
第14週	06/02	Invited Talk: Core-collapse Supernovae (Prof. Kuo-Chuan Pan from NTHU)
第15週	06/09	Invited Talk: TBD
第16週	06/16	Final Presentation

Course Goals

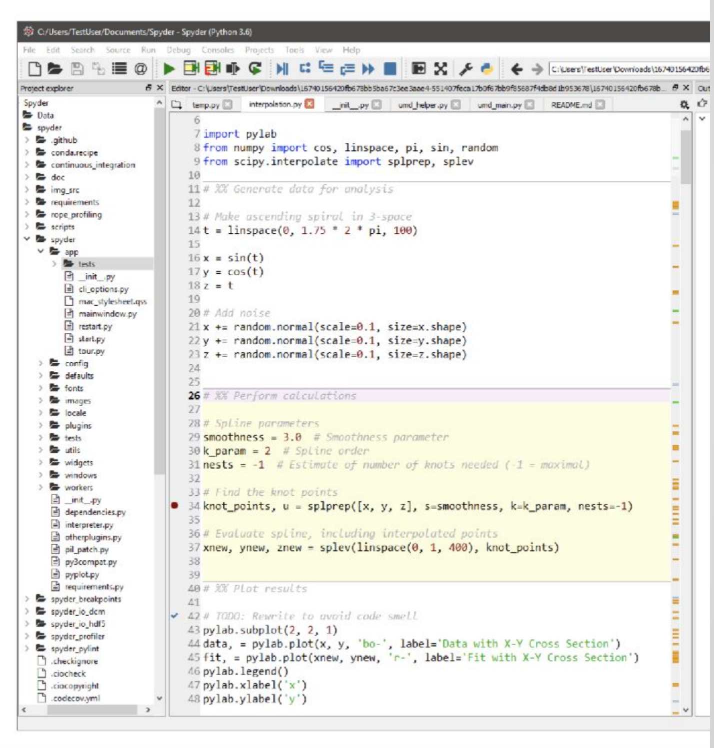
- **Numerical algorithms**
  - Simulations are notorious for "garbage in, garbage out"
  - Numerical errors and their origins
  - Computational complexity
- **Parallel Computing**
  - Astrophysical simulations can be extremely time-consuming
  - Single multi-core CPU → multi-CPU → GPU → CPUs + GPUs
- **Demo**
  - Thinking ≠ Learning ⇒ PRACTICE !!
  - Runnable demos will be provided for most topics
  - In-class practice (bring your laptop!) and homework
- **Code co-development**
  - GitHub
  - Final project

Grading & TA

- **Homework (70%)**
  - **CEIBA: https://ceiba.ntu.edu.tw**
  - **Upload within 2 weeks**
    - **Delay < 1 week: 20% discount**
    - **Delay ≥ 1 week: zero point**
  - **NO COPY**
- **Final project (30%)**
  - **Team work (3-4 members per group)**
  - **Upload code to GitHub**
  - **Demo & oral presentation**
- **Teaching assistant: Po-Hsun Zheng (zengbs@gmail.com)**

Course Prerequisite

- **Basic of Python 2 or 3**
  - Linux-like system: you should have python installed already
  - Windows: **SPYDER** may be a good choice
- **Basic of C/C++**
  - Linux-like system: **gnu** compiler
  - Windows: try **Visual Studio Express**
- **Contact TA if you need any help or don't have access to a working system**

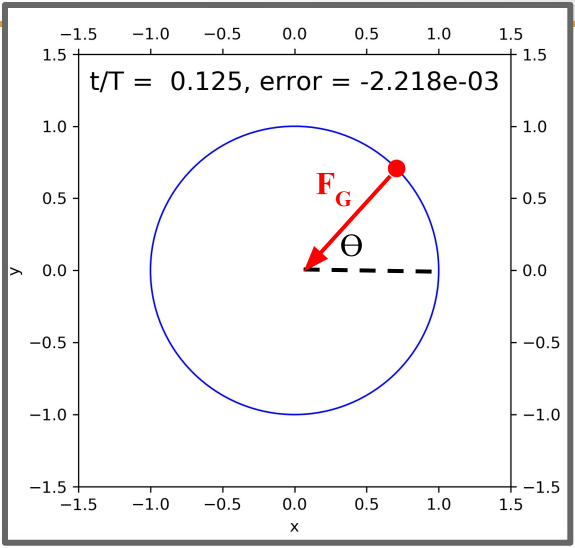


Quick Taste: Keplerian Motion

```
# constants
G  = 1.0       # gravitational constant
M  = 2.0       # central point mass
dt = 1.0e-2    # time interval for data update

# initial condition
t     = 0.0
x     = 1.0
y     = 0.0
r     = ( x**2 + y**2 )**0.5
vx    = 0.0
vy    = ( G*M/r )**0.5
v_abs = ( vx**2 + vy**2 )**0.5
E0    = 0.5*v_abs**2 - G*M/r
```

⟸ Initial total energy for estimating numerical errors later




```
# plotting parameters
period          = 2.0*np.pi*r/v_abs
end_time        = 1.0*period
nstep_per_image = 1
```

⟸ Simulate for a single orbit period

⟸ Plotting frequency (i.e., # of updates between two images)

lec01-demo01.py

Quick Taste: Keplerian Motion

```
def update_orbit( i ):
   global t, x, y, vx, vy
   for step in range( nstep_per_image ):
#     calculate acceleration
      r     = ( x*x + y*y )**0.5
      a_abs = G*M/(r*r)
      ax    = -a_abs*x/r
      ay    = -a_abs*y/r
#     update orbit (Euler's method)
      x  = x + vx*dt
      y  = y + vy*dt
      vx = vx + ax*dt
      vy = vy + ay*dt
#     update time
      t = t + dt
      if ( t >= end_time ):   break

#  calculate energy error
   E   = 0.5*( vx**2 + vy**2 ) - G*M/r
   err = (E-E0)/E0
```

⇐ **Use global instead of local variables**

⇐ **This loop is simply for reducing the plotting frequency (which could also be time-consuming!)**

⇐

$$a_x = -\frac{GM}{r^2}cos(\theta)$$

$$a_y = -\frac{GM}{r^2}sin(\theta)$$

⇐ **Euler's integration:** $f(t+\Delta t) = f(t) + f'(t)\Delta t + O(\Delta t^2)$

⇐ **Stop when reaching the target time**

⇐ **Assuming star mass = 1 for simplicity**

lec01-demo01.py

Run **lec01-demo01.py**

Lessons Learned

- Error $\propto \Delta t$ (error per step $\propto \Delta t^2$)
- Possible origin of errors?
  - Spatial discretization ❌
  - Calculating gravity ❌
  - Updating orbit (Euler's method) ✔
- Validate your code very very very carefully
  - Never trust it without thorough validation
  - How? PHYSICS!
    - Conserved quantity
    - Analytical solution
    - Always ask WHY
  - Real challenge is usually not coding but debugging
- Simulation time $\propto \Delta t^{-1}$
- Data analysis and visualization is NOT free

Simple Improvement on Orbit Update

Original

```
#     calculate a(t)
      r     = ( x*x + y*y )**0.5
      a_abs = G*M/(r*r)
      ax    = -a_abs*x/r
      ay    = -a_abs*y/r

#     use v(t) and a(t) to update position
#     and velocity by dt
      x  = x + vx*dt
      y  = y + vy*dt
      vx = vx + ax*dt
      vy = vy + ay*dt
```

⟸ Be careful about the order of update

Revised

```
#     drift: update position by 0.5*dt
      x = x + vx*0.5*dt
      y = y + vy*0.5*dt

#     kick: calculate a(t+0.5*dt) and use that
#     to update velocity by dt
      r     = ( x*x + y*y )**0.5
      a_abs = G*M/(r*r)
      ax    = -a_abs*x/r
      ay    = -a_abs*y/r
      vx    = vx + ax*dt
      vy    = vy + ay*dt

#     drift: use v(t+dt) to update position
#     by another 0.5*dt
      x = x + vx*0.5*dt
      y = y + vy*0.5*dt
```

Run **lec01-demo02.py**

Lessons Learned

- Error $\propto \Delta t^2$ (error per step $\propto \Delta t^3$)
- Computational complexity with N particles
  - Position/velocity update: N
  - Computing gravity: N (external gravity), N^2 (self-gravity)
- Computing time only increases slightly!
  - Position update: 1 → 2 per step
  - Velocity update: still 1 per step
  - Computing gravity: still 1 per step
- Performance
  - Efficient algorithm
  - Scalability
  - Hardware acceleration
  - Extensibility and sustainability

References

- Python tutorials
  - Programming with Python (good start)
  - Learn Python - Free Interactive Python Tutorial (online practice)
  - The Python Tutorial — Python 3.8.2 documentation (official tutorial)
  - Python for Beginners (on YouTube)
- Online Python interpreters
  - https://www.python.org/shell
  - https://repl.it/languages
  - https://www.onlinegdb.com