Python DeCal

Week 10

Monday

https://forms.gle/2azpxSrSRpsi2Fbv9

Announcements

- HW7 - Due Wednesday before lecture!
- Keep working on your final project :)
- Don’t wait till OHs for help… Send us an email if you need. We can arrange meetings etc.
- Take advantage of the DISCORD!
- The earlier you have a problem solved the better.

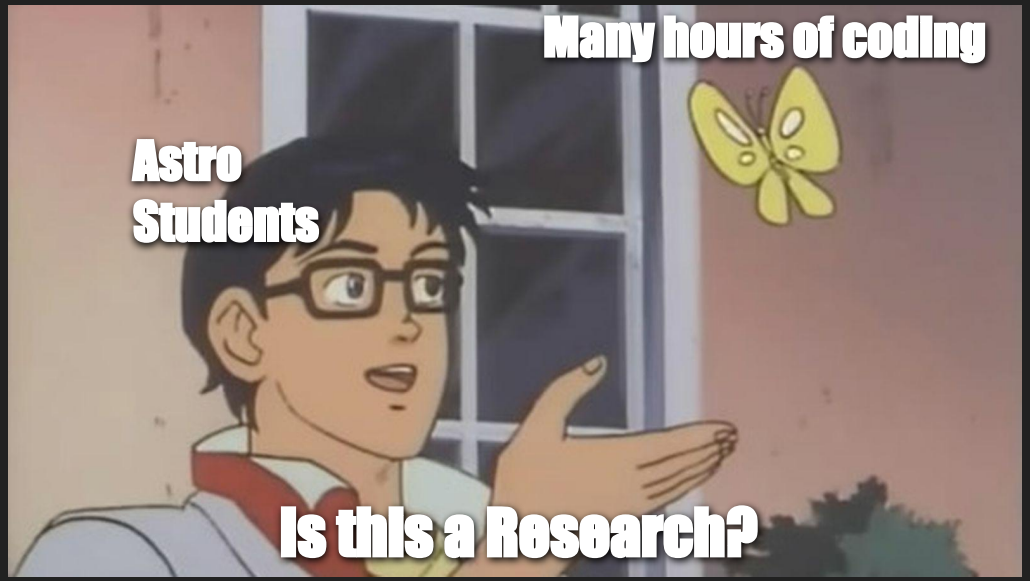
Breakout Rooms

- Have you started coding for the final project yet? (or tried to start)

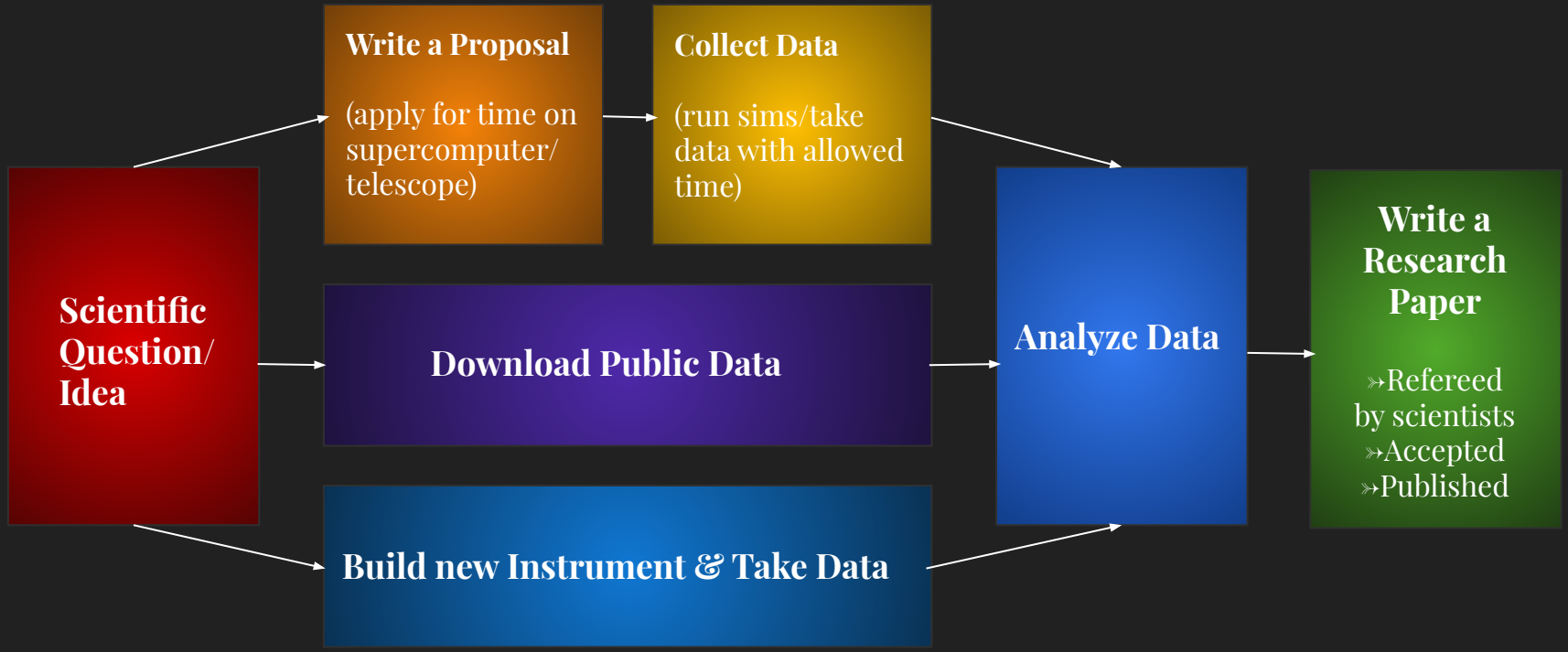
- How nervous are you about your project on a scale of 1 to 10?

- Have you started LaTeXing your CV? What has been a challenge in making it so far?

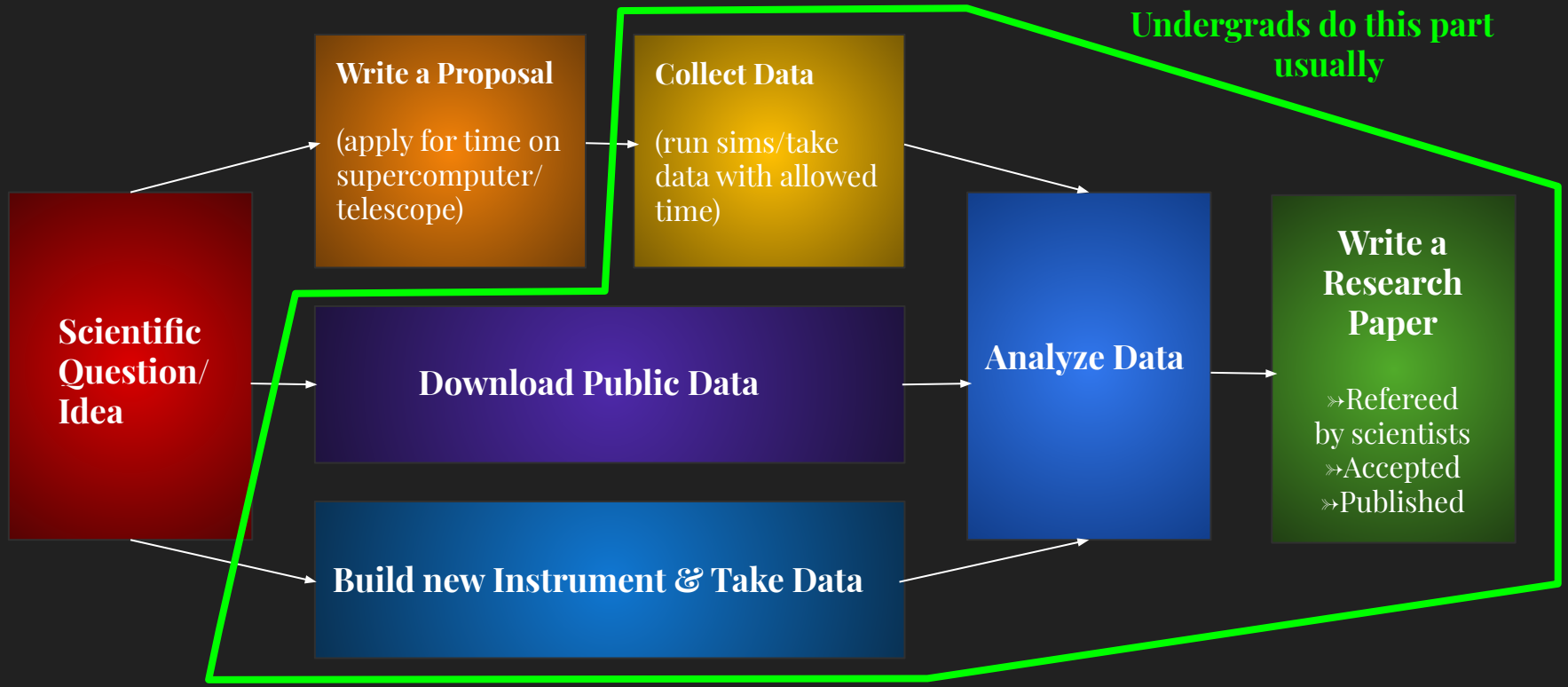
Research



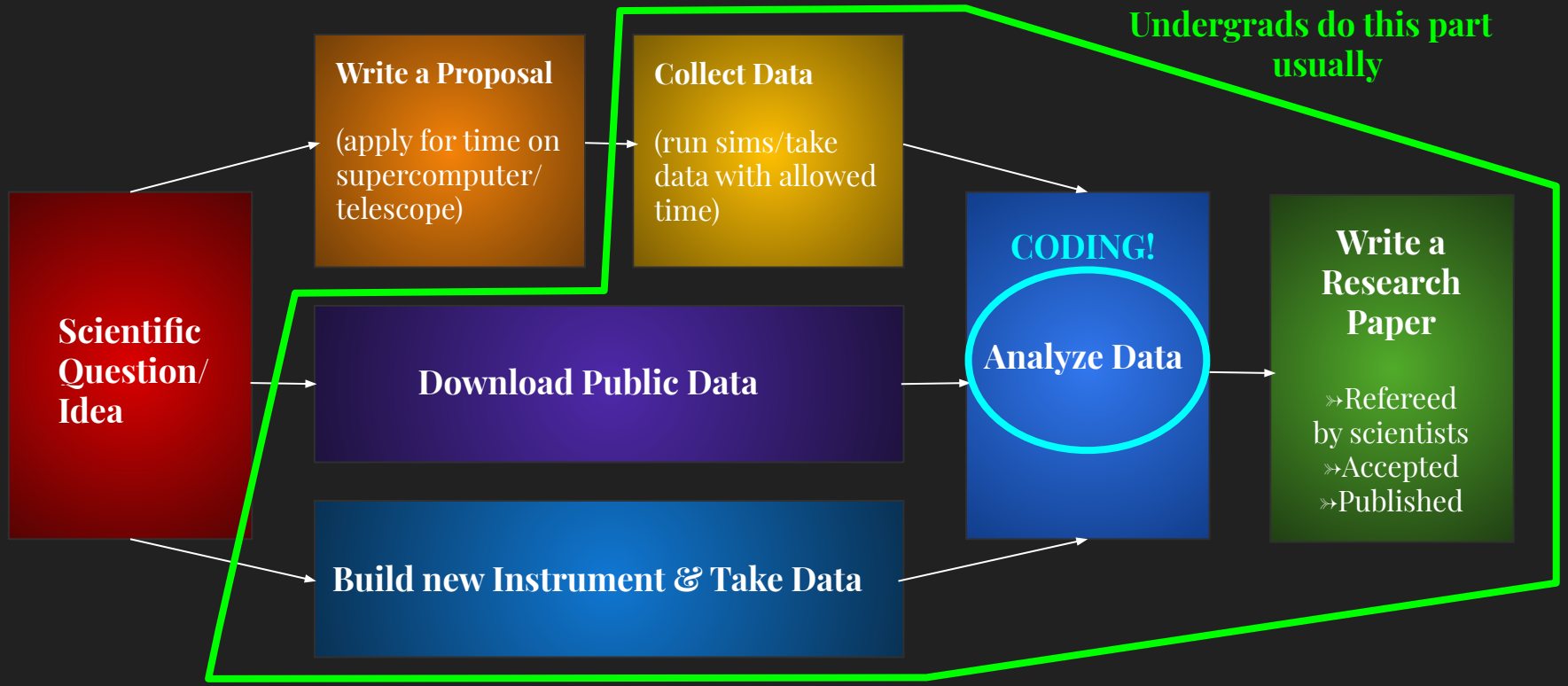
How Research is Done



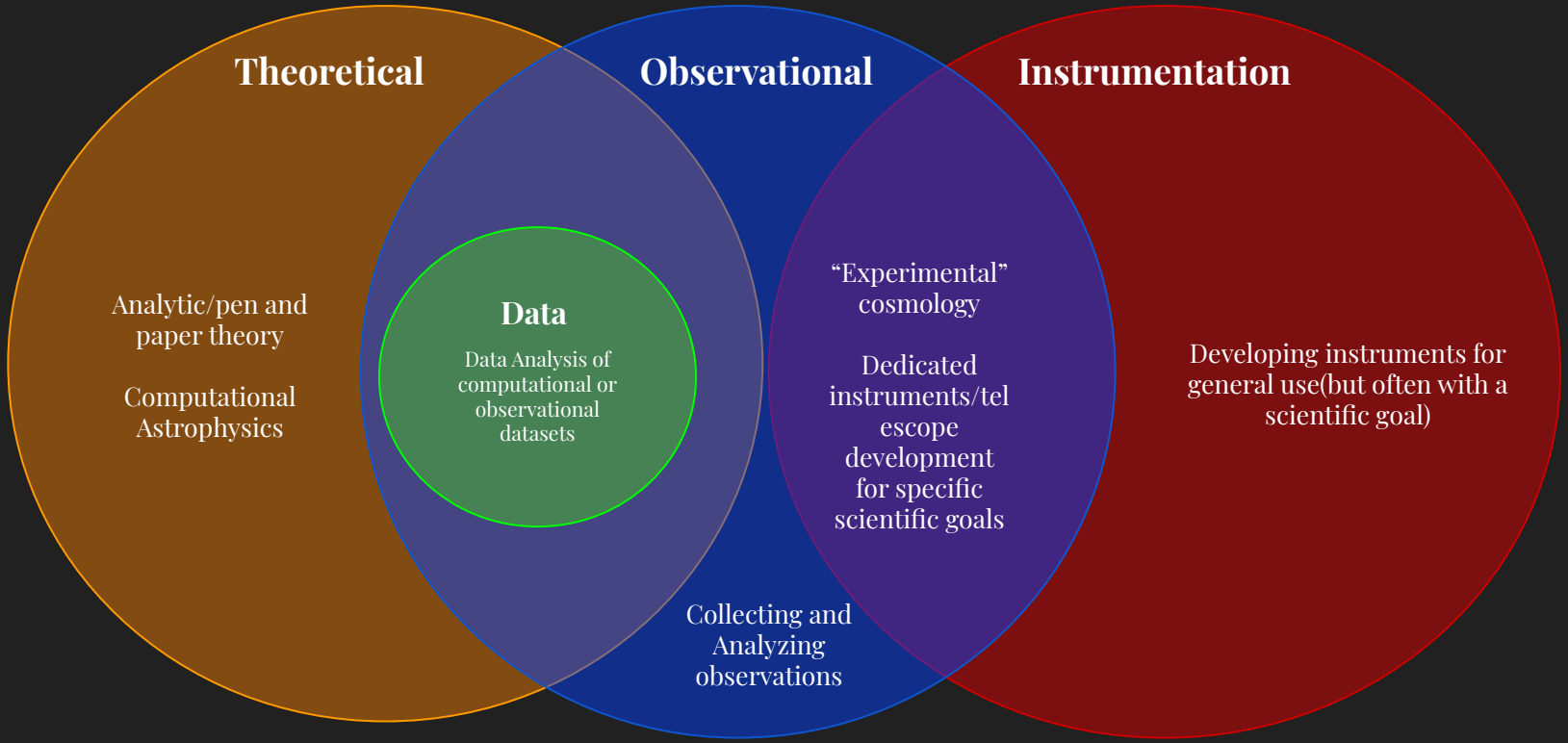
How Research is Done



How Research is Done



Astrophysics Research

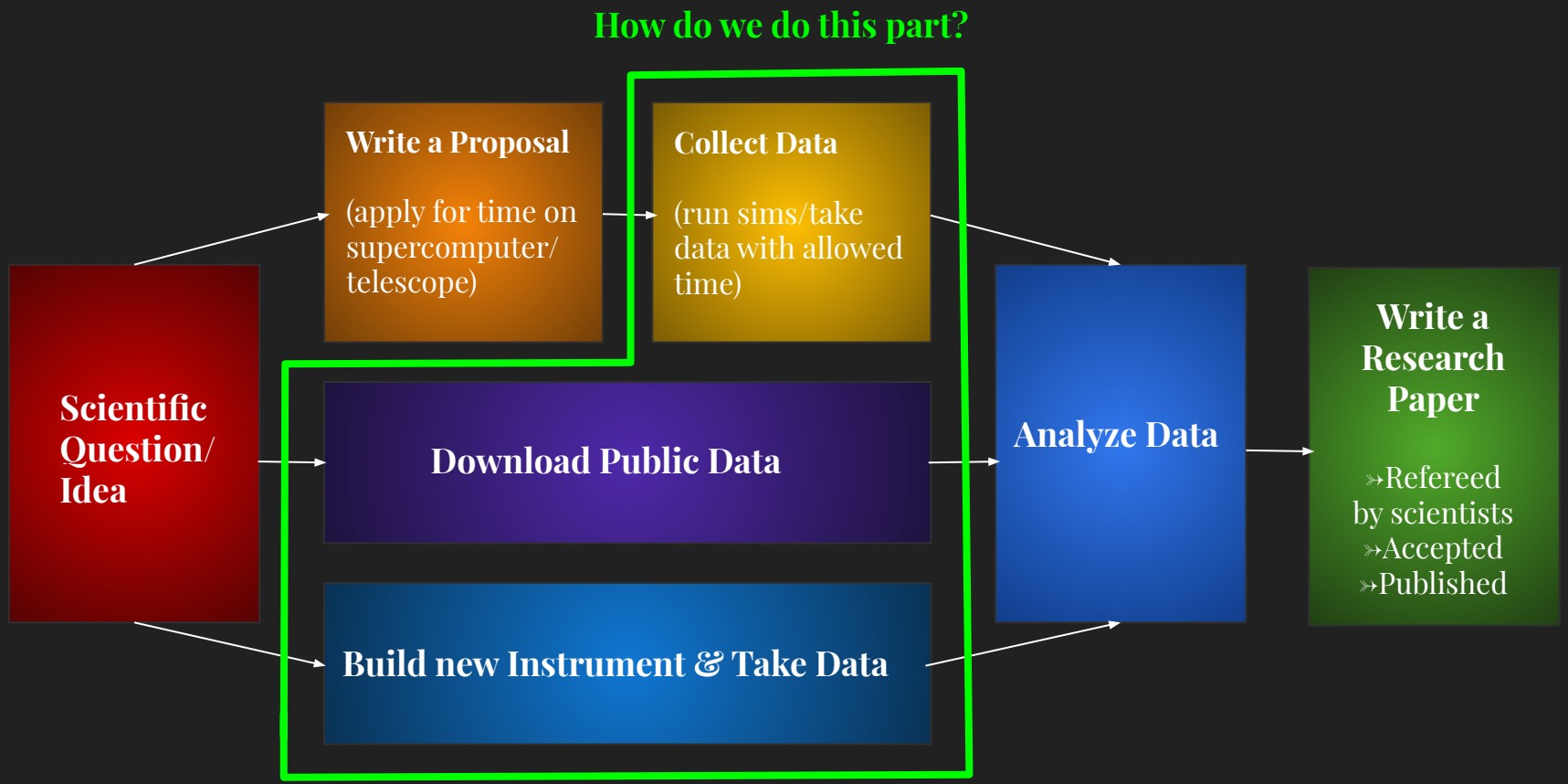


First part of any project....

Literature

- arXiv.org
  - Updating everyday
  - Preprints of papers
  - Many different topics
  - Even Lecture notes are here

- Astrophysics Data System (ADS)
  - Customise your search
  - Generating bibliography
  - All astronomers use it!


How do we do this part?
Write a Proposal
(apply for time on supercomputer/ telescope)
Collect Data
(run sims/take data with allowed time)
Scientific Question/ Idea
Download Public Data
Build new Instrument & Take Data
Analyze Data
Write a Research Paper
»Refereed by scientists
»Accepted
»Published

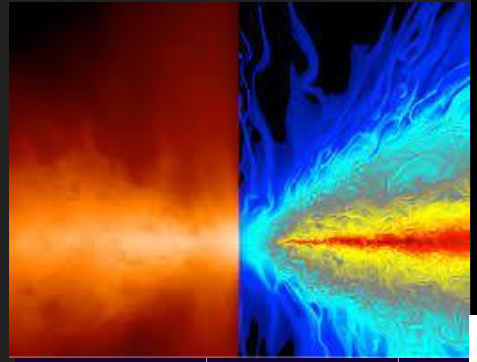
Real Data Acquisition

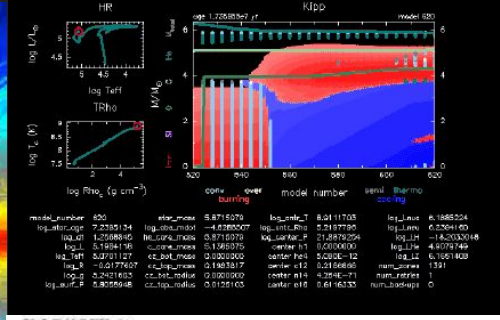
- **VizieR**
  - https://vizier.u-strasbg.fr/viz-bin/VizieR
  - Catalogues, tables from papers, everything!
- **SIMBAD**
  - http://simbad.u-strasbg.fr/simbad/
  - Database for basic information about a target
- **Gaia archive**
  - https://gea.esac.esa.int/archive/
  - Astrometry, photometry
  - Galactic astronomy
- **Get your own data**
- You can use astropy to query the data
  - `astroquery`
  - https://astroquery.readthedocs.io/en/latest/index.html

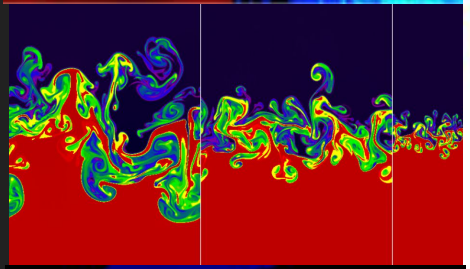
gaia

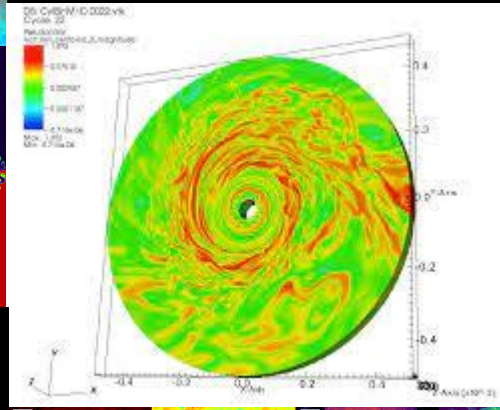
Simulations

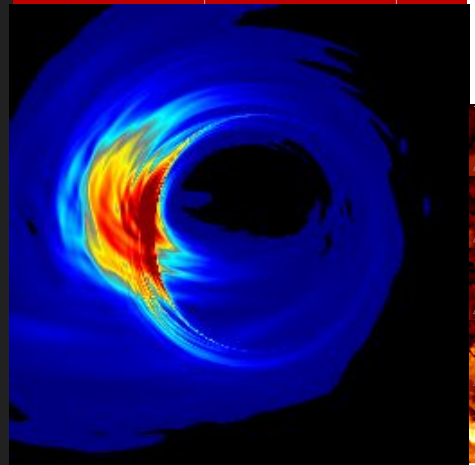
- Can be coded/ran by you (smaller scale simulation ~ simple-ish)
- We use software packages to run your own complex simulations
  - MESA (1D Stellar Simulations)
  - Athena++ (3D Magnetohydrodynamics)
- Really complex sims are ran on supercomputers (EXPENSIVE)
- We can download their sim data
  - IllustrisTNG (Cosmology Simulation of the Universe + MHD + Dark Matter)
- More of this on Wednesday :)

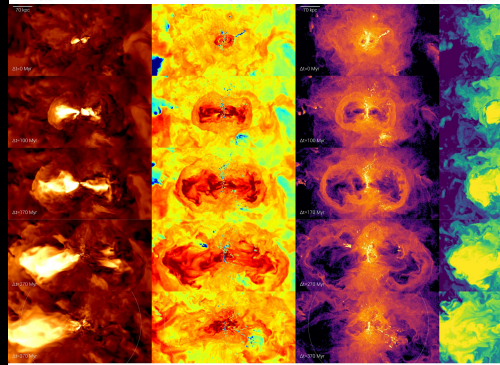


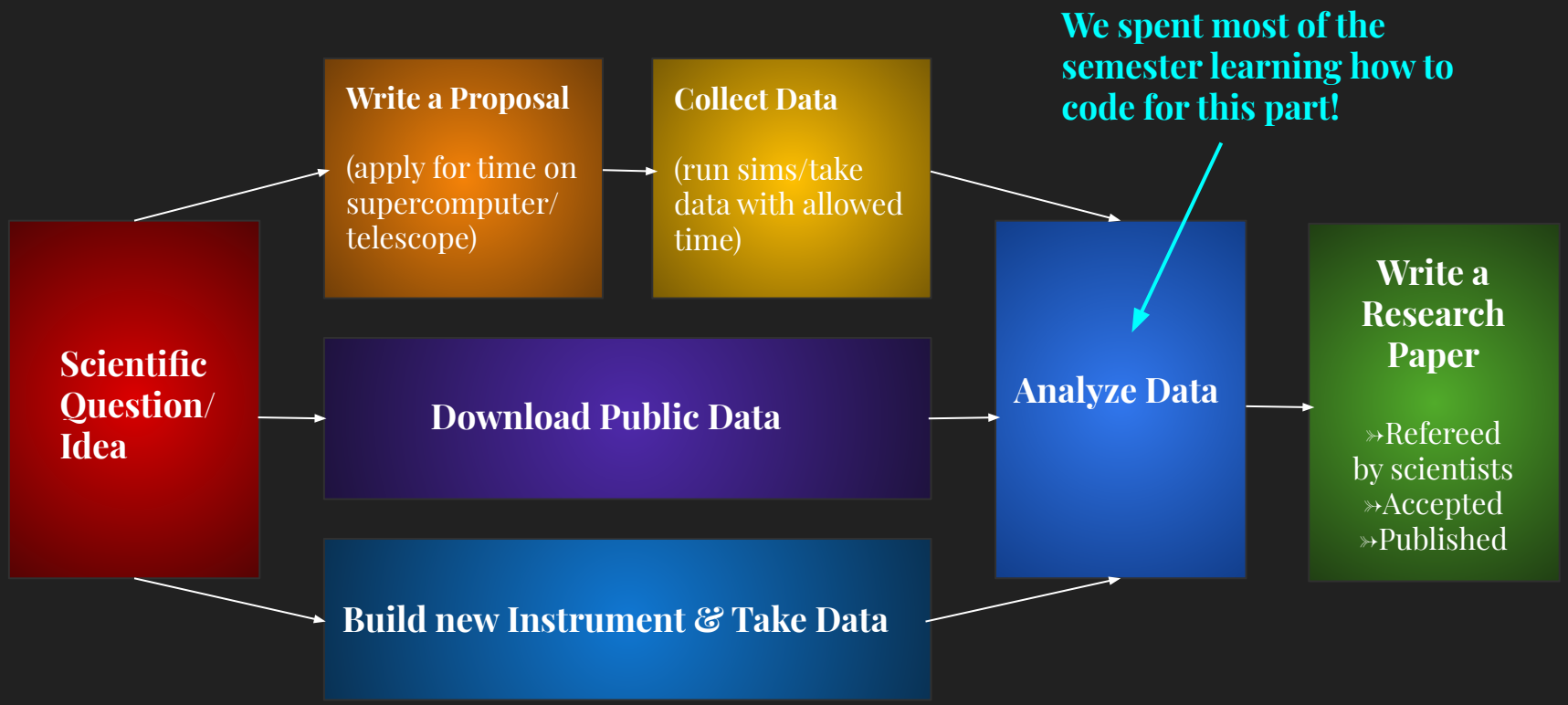






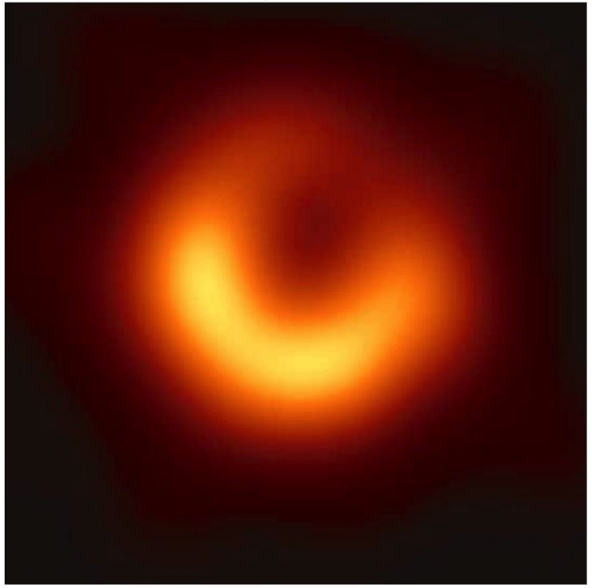




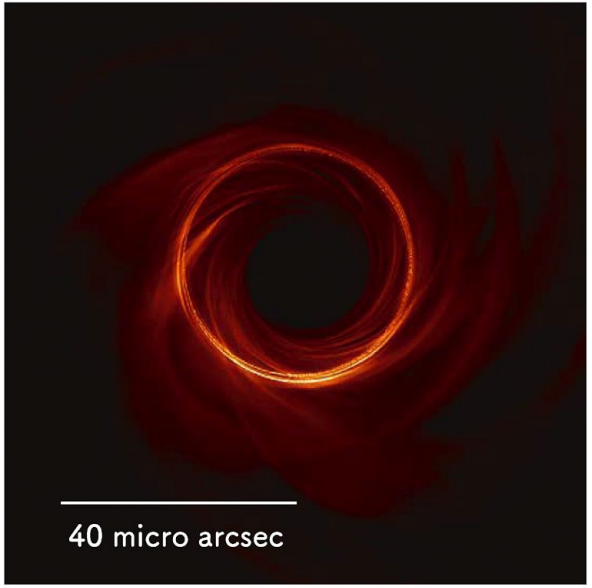

We spent most of the semester learning how to code for this part!
Write a Proposal
(apply for time on supercomputer/ telescope)
Collect Data
(run sims/take data with allowed time)
Scientific Question/ Idea
Download Public Data
Build new Instrument & Take Data
Analyze Data
Write a Research Paper
»Refereed by scientists
»Accepted
»Published

Compare and Analyze Data

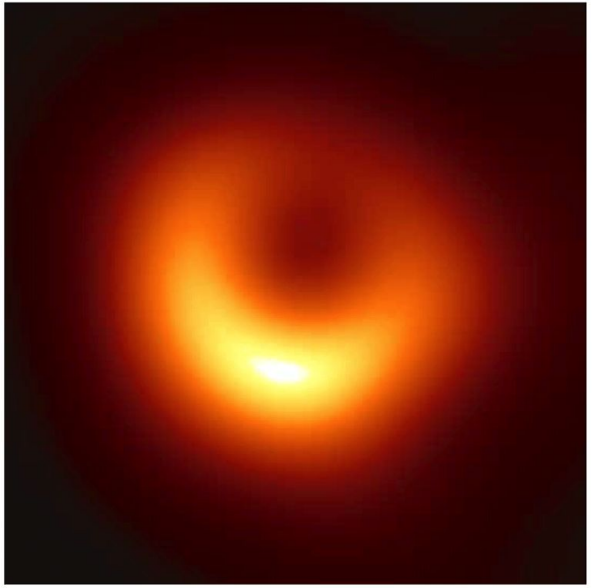
M87 (Apr 6. 2017)

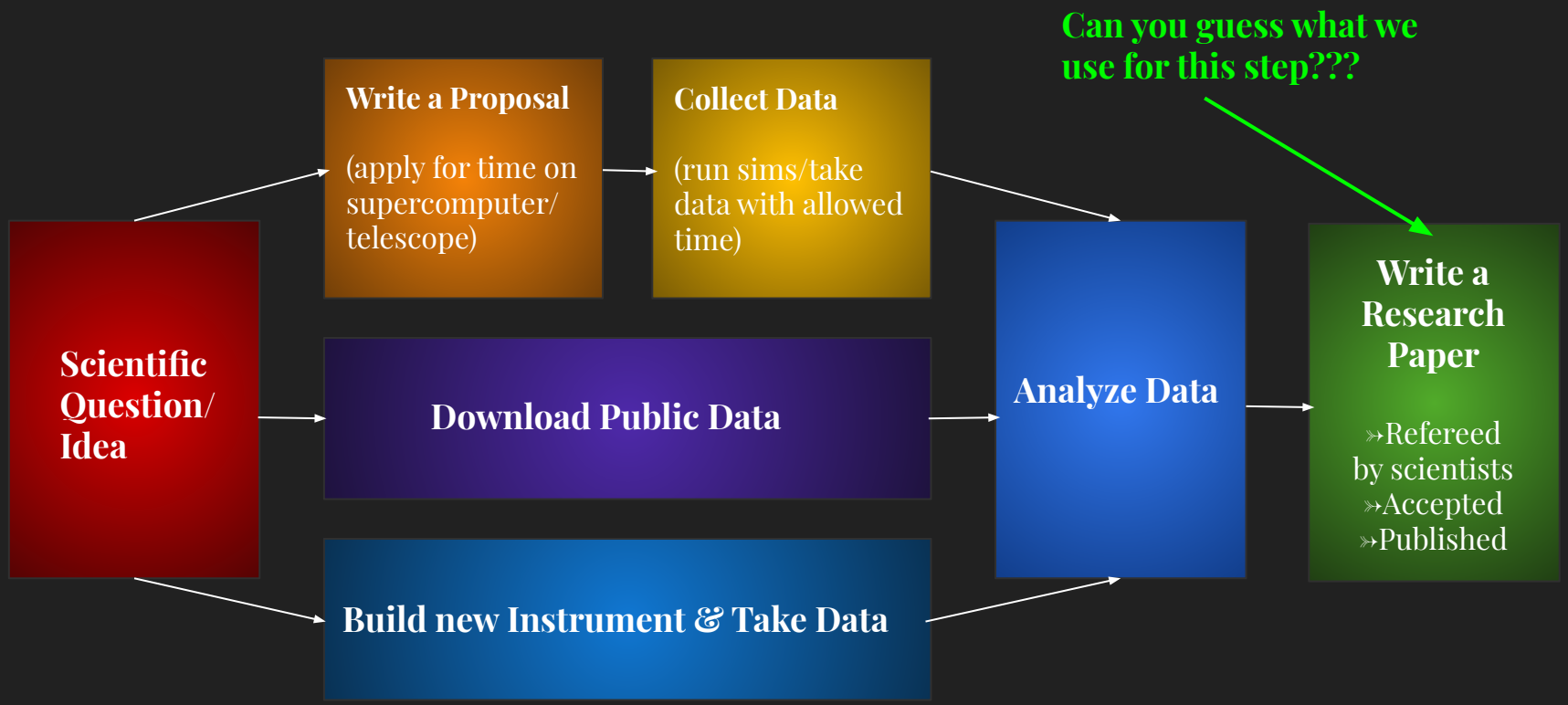


Simulation



Blurred Simulation




Can you guess what we use for this step???
Write a Proposal
(apply for time on supercomputer/ telescope)
Collect Data
(run sims/take data with allowed time)
Scientific Question/ Idea
Download Public Data
Build new Instrument & Take Data
Analyze Data
Write a Research Paper
»Refereed by scientists
»Accepted
»Published

WEDNESDAY

https://forms.gle/P2vxZzpQyrhigi2X6

Announcements

- New Assignment (HW8) will be available on bCourses
  - Build your own Personal Research Website (due Friday next week)
  - We will be giving some advice on Monday on how to do this, for now check bCourses files tab

- CV's (HW7) Due just now!

- Make sure you are working on your final projects too!!! We will have a check-in with your group in the near future.

Breakout Rooms

- Go to bCourses ↠Files↠Homework↠HW8

- Read HW8 and discuss with your room, how do you think you will make your own personal website? Is there anything you think you would want to put on it?
- If you have extra time, what did you learn from Monday's Lecture?


Simulations

Simulations!

- This is a more fun type of lecture to see some cool shit you can do with python!

EXAMPLES

Solar System

What yours could look like if you are doing this for your project

Let's Write a Game!

Key Principles

- Each Time step you need to increment all the relevant values that change with time
- Could be: position, velocity, acceleration, phase, other values that change with time (maybe mass or luminosity?)
- Need to have a way of solving for these values at each new time step
- Need to continue incrementing time steps until sufficient time is passed

(While Loop?)

The Main Loop

- The main loop is the loop everything runs inside of, if the loop isn't running your simulation isn't running

  Five Key Points in a Main Loop:

  - Get relevant values (**x**, **v**, **a**, **t**, etc...)
  - Calculate Changes From Physics
  - Apply Changes to old values (**v** + **dv**) or (**x** + **dx**) and (**t** + **dt**)
  - Draw Results
  - Repeat

Let's Make a Game

Step 1: Make your objects

I like to do this in the form of objects and classes if there is going to be lots of them.

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def len(self):
        return math.sqrt(self.x*self.x + self.y*self.y)
    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)
    def __sub__(self, other):
        return Vector(self.x - other.x, self.y - other.y)
    def __mul__(self, other):
        return Vector(self.x * other, self.y * other)
    def __rmul__(self, other):
        return Vector(self.x * other, self.y * other)
    def __truediv__(self, other):
        return Vector(self.x / other, self.y / other)
    def angle(self):
        return math.atan2(self.y, self.x)
    def norm(self):
        if self.x == 0 and self.y == 0:
            return Vector(0, 0)
        return self / self.len()
    def dot(self, other):
        return self.x*other.x + self.y*other.y

class Paddle:
    def __init__(self,x,y):
        self.position = Vector(x, y - 30)
        self.velocity = Vector(0, 0)
        self.width = 20
        self.height = 100

class Ball:
    def __init__(self,x,y):
        self.radius = 10
        self.color = white
        self.position = Vector(x,y)
        self.velocity = Vector(0,0)

ball = Ball(400,250)
R_Paddle = Paddle(750, 250)
L_Paddle = Paddle(50, 250)
Paddles = [L_Paddle, R_Paddle]
```

Let's Make a Game

Step 2: Initialize Everything

There are different ways of doing this, I found making a main function is the best way

In this example, I used a library called pygame to handle all the screen drawing, but you can use whatever tool you want (matplotlib, vpython, etc....)

```
import pygame
from pygame.locals import *
import random, math, sys

pygame.init()
#initialize and create some variables
width, height = 800, 500
Surface = pygame.display.set_mode((800,500))
pygame.display.set_caption("Pong")
black = (0,0,0)
white = (255, 255, 255)
red = (255, 0, 0)
blue = (0,0,200)
```

```
def main():
    score_p = 0
    score_c = 0
    speed = ball.velocity.len()
    while True:
        GetInput()
        Move()
        Collision()
        score()
        Draw()


if __name__ == '__main__': main()
```

Let’s Make a Game

Step 3: Get input (optional)

If you wanted to make a playable game, then you would need to find a way of getting input from the user. This can be done in pygame as well.

For a simulation, this is not at all needed, so skip this slide unless interested

```
def GetInput():
    keystate = pygame.key.get_pressed()
    for event in pygame.event.get():
        if event.type == pygame.QUIT or keystate[pygame.K_ESCAPE]:
            pygame.quit(); sys.exit()
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                R_Paddle.velocity.y = -10
            elif event.key == pygame.K_DOWN:
                R_Paddle.velocity.y = 10
            elif event.key == pygame.K_SPACE:
                ball.velocity.x = random.randrange(-15,15)
                ball.velocity.y = random.randrange(-15,15)
        elif event.type == pygame.KEYUP:
            if event.key == pygame.K_UP:
                R_Paddle.velocity.y = 0
            elif event.key == pygame.K_DOWN:
                R_Paddle.velocity.y = 0
```

Let’s Make a Game

Step 4: Physics functions!

Write helper functions that handle the load of using physics to calculate what happens in each frame

This example has a function to handle collisions and a function to handle movement

```
def Collision():
    Circle = ball
    if Circle.position.y < Circle.radius and Circle.velocity.y < 0:
        Circle.velocity.y *= -1
    if Circle.position.y > height-Circle.radius and Circle.velocity.y > 0:
        Circle.velocity.y *= -1
    if Circle.position.x < L_Paddle.position.x + L_Paddle.width + ball.radius/2 and Circle.velocity.x < 0:
        if Circle.position.y > L_Paddle.position.y and Circle.position.y < L_Paddle.position.y + L_Paddle.height:
            Circle.velocity.x *= -1
    if Circle.position.x > R_Paddle.position.x - R_Paddle.width + ball.radius/2 and Circle.velocity.x > 0:
        if Circle.position.y > R_Paddle.position.y and Circle.position.y < R_Paddle.position.y + R_Paddle.height:
            Circle.velocity.x *= -1


def Move():
    for pad in Paddles:
        pad.position = pad.position + pad.velocity
    ball.position = ball.position + ball.velocity

    L_Paddle.velocity.y = ball.velocity.y * 0.95
```

Let’s Make a Game

Step 5: Have a way of keeping score

In your simulation, you might want to track the number of times a specific condition is met.

Example: How many times does a pendulum reach a certain height

Example: How many times do two asteroids collide?

```
def score():
    global score_p
    global score_c
    score_p = 0
    score_c = 0
    if ball.position.x < L_Paddle.position.x:
        score_p += 1
        ball.position.x = 400
        ball.position.y = 250
        speed = 0
    elif ball.position.x > R_Paddle.position.x:
        score_c += 1
        ball.position.x = 400
        ball.position.y = 250
        speed = 0
```

Let’s Make a Game

Step 6: Drawing the simulation

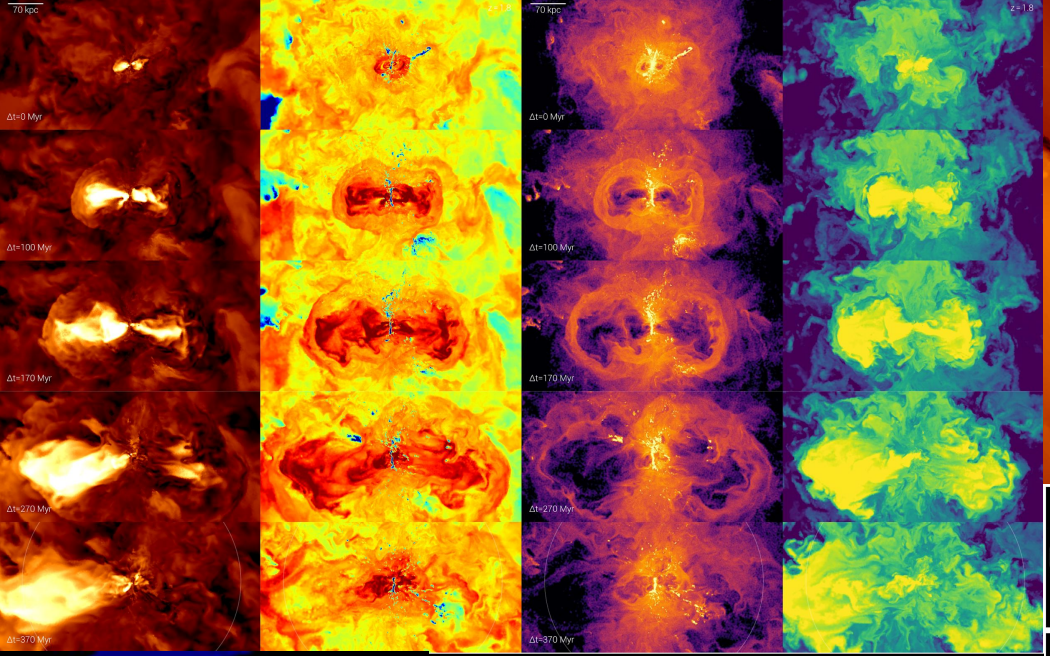
This can be vastly different depending on what library you use to draw. I used PyGame but you can use matplotlib or vpython, etc...

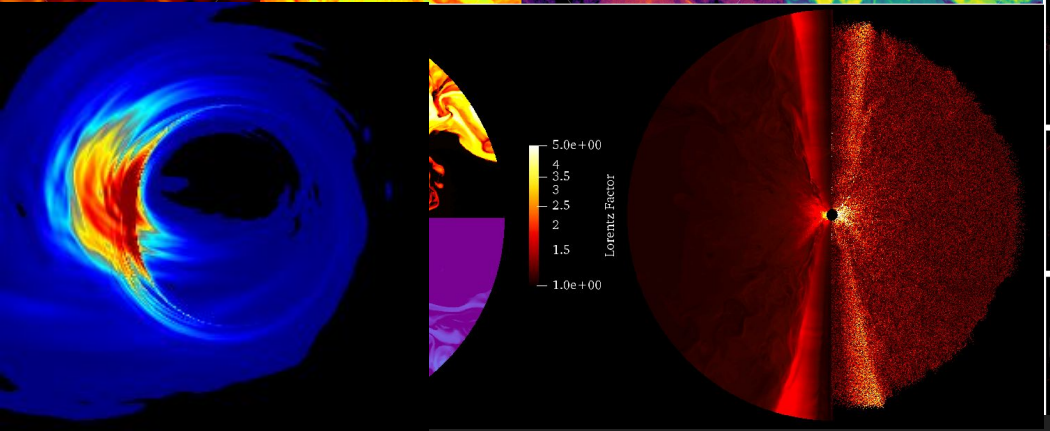
```
def Draw():
    clock = pygame.time.Clock()
    Surface.fill(black)
    pygame.draw.rect(Surface, red, [R_Paddle.position.x,R_Paddle.position.y,R_Paddle.width, R_Paddle.height])
    pygame.draw.rect(Surface, blue, [L_Paddle.position.x,L_Paddle.position.y,L_Paddle.width, L_Paddle.height])
    pygame.draw.circle(Surface,ball.color,(int(ball.position.x),int(ball.position.y)),ball.radius)
    font = pygame.font.Font('batmfa__.ttf', 18)
    text = font.render('Computer: {0:1.0f}'.format(score_c), True, blue,black)
    textRect = text.get_rect()
    textRect.center = (210,25)
    Surface.blit(text, textRect)
    font2 = pygame.font.Font('batmfa__.ttf', 18)
    text2 = font.render('Player: {0:1.0f}'.format(score_p), True, red,black)
    textRect2 = text2.get_rect()
    textRect2.center = (590,25)
    Surface.blit(text2, textRect2)
    pygame.display.flip()
```

Result!

What's the difference?

- 1 big difference: A video game asks for input from the user to change what's happening
- Simulations just have a set of physics rules that tell the program what to do each iteration of the main loop
- Simulations are easier to code than video games! (but they're harder to understand)


70 kpc
z = 1.8
Δt=0 Myr
Δt=100 Myr
Δt=170 Myr
Δt=270 Myr
Δt=370 Myr


5.0e+00
4
3.5
3
2.5
2
1.5
1.0e+00
Lorentz Factor