File I/O

At least the basics

Here’s how computers work

- Computers remembers things in two ways:
  - Long term (Hard disk)
  - Short term (Random Access Memory)

- All computer processing happens in short term memory.

- Files stored on disk needs to be ‘read’ into memory for processing.

File formats

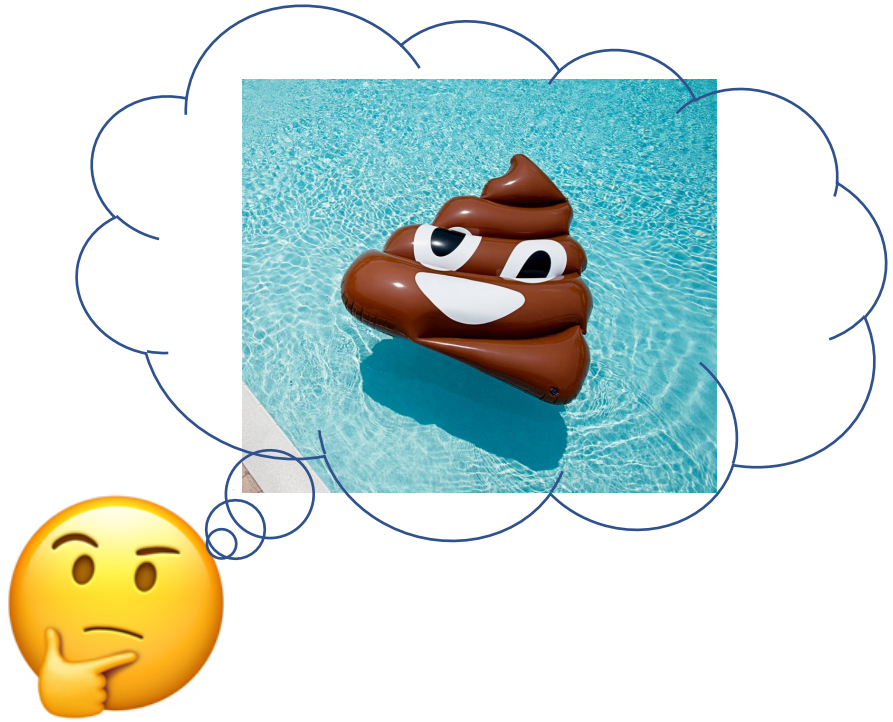
- Text/ascii files (formatted)
- Binary (unformatted)

File States:

- Compressed (zipped)
- Encrypted (sounds cool, but sadly not covered)

Basics of file input

- Q. Do you have enough memory to read 10 million rows of floats?



How many bits to a byte? (or how many bytes to a float?)

- A bit is a binary digit (0 or 1)
- A byte is 8 bits (number of bits used to encode a single character of text)
- 8 bits to a byte
- A double precision float is 8 bytes (memory is typically packaged in trillions of bytes)
- so…

So do you?

- 100×10^6 floats x 8 bytes/float x 10^{-9} Gb/byte = 0.8 Gb
- If you've got around 4gb of memory then YES!
- But be careful if you are interested in doing any operations on these data arrays (more on this later).

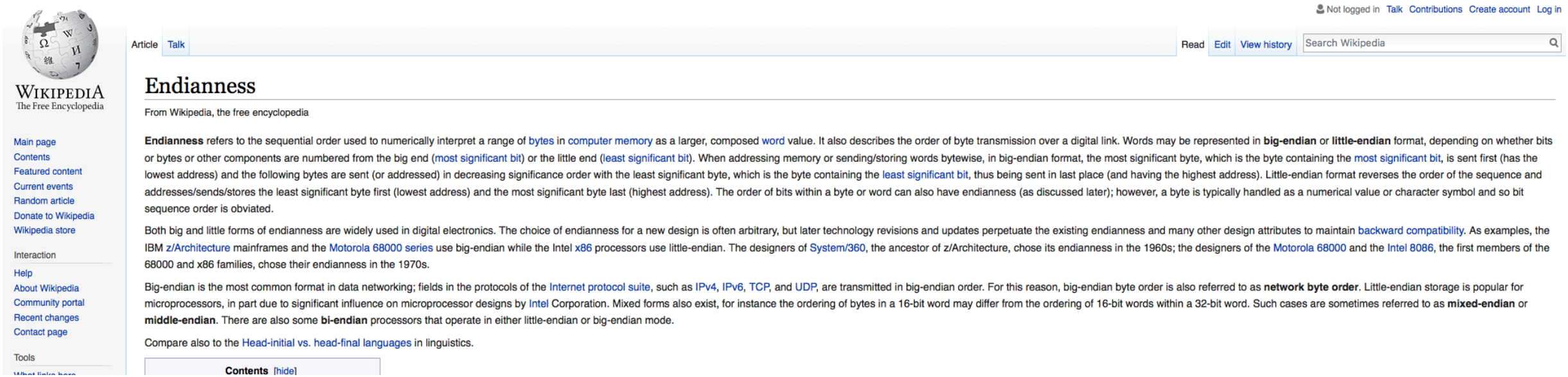
Binary files

- Easily interpreted by the computer.
- Not easily interpreted by humans.

Binary files: read bit by bit (or byte by byte)

- Unlike text file types, the information is stored in data “chunks”.
- You need to know how the information has been recorded to be able to correctly interpret the information that is read.
- Can have binary markers: ‘header’ and ‘footer’ that specifies the total number of bytes written.

So Much Endian-ness



Not logged in Talk Contributions Create account Log in

Article Talk | Read Edit View history | Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

Endianness

From Wikipedia, the free encyclopedia

Endianness refers to the sequential order used to numerically interpret a range of bytes in computer memory as a larger, composed word value. It also describes the order of byte transmission over a digital link. Words may be represented in **big-endian** or **little-endian** format, depending on whether bits or bytes or other components are numbered from the big end (most significant bit) or the little end (least significant bit). When addressing memory or sending/storing words bytewise, in big-endian format, the most significant byte, which is the byte containing the most significant bit, is sent first (has the lowest address) and the following bytes are sent (or addressed) in decreasing significance order with the least significant byte, which is the byte containing the least significant bit, thus being sent in last place (and having the highest address). Little-endian format reverses the order of the sequence and addresses/sends/stores the least significant byte first (lowest address) and the most significant byte last (highest address). The order of bits within a byte or word can also have endianness (as discussed later); however, a byte is typically handled as a numerical value or character symbol and so bit sequence order is obviated.

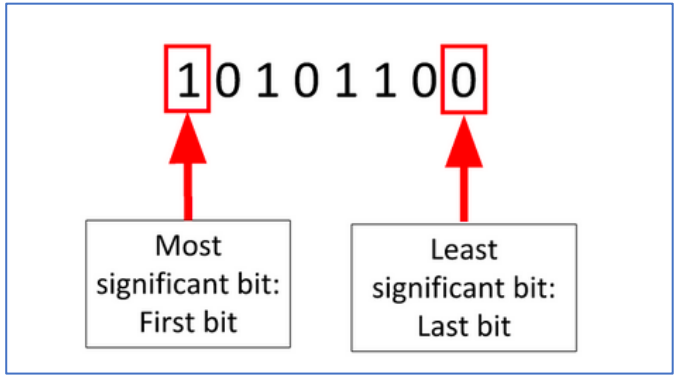
Both big and little forms of endianness are widely used in digital electronics. The choice of endianness for a new design is often arbitrary, but later technology revisions and updates perpetuate the existing endianness and many other design attributes to maintain backward compatibility. As examples, the IBM z/Architecture mainframes and the Motorola 68000 series use big-endian while the Intel x86 processors use little-endian. The designers of System/360, the ancestor of z/Architecture, chose its endianness in the 1960s; the designers of the Motorola 68000 and the Intel 8086, the first members of the 68000 and x86 families, chose their endianness in the 1970s.

Big-endian is the most common format in data networking; fields in the protocols of the Internet protocol suite, such as IPv4, IPv6, TCP, and UDP, are transmitted in big-endian order. For this reason, big-endian byte order is also referred to as **network byte order**. Little-endian storage is popular for microprocessors, in part due to significant influence on microprocessor designs by Intel Corporation. Mixed forms also exist, for instance the ordering of bytes in a 16-bit word may differ from the ordering of 16-bit words within a 32-bit word. Such cases are sometimes referred to as **mixed-endian** or **middle-endian**. There are also some **bi-endian** processors that operate in either little-endian or big-endian mode.

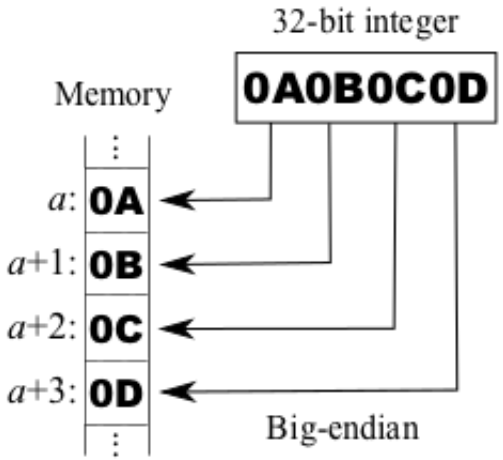
Compare also to the Head-initial vs. head-final languages in linguistics.

Contents [hide]

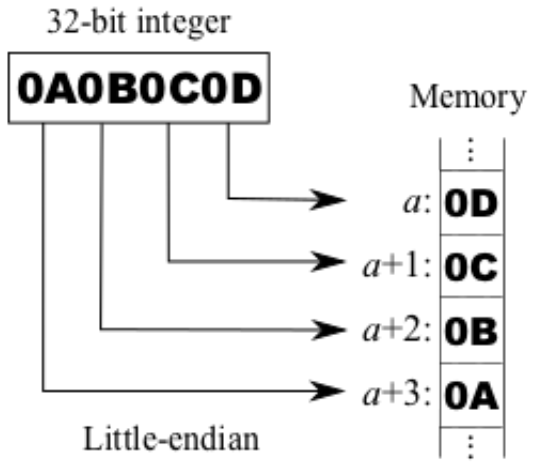
So Much Endian-ness



Big Endian



Little Endian



Example of code for binary read in

```
import numpy as np
data = np.fromfile('path_to_file' , '< f8')
```

< - little endian

> - big endian

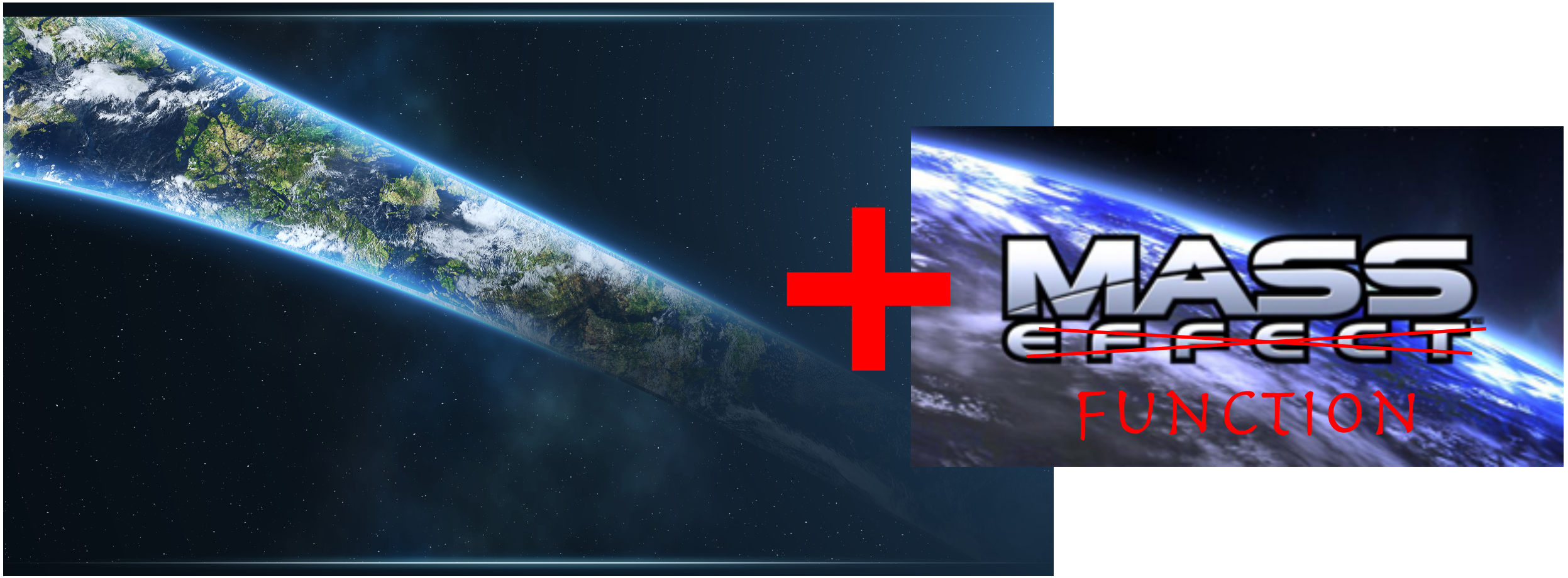
Why I’m here…

- I’ve been working on the halo mass function (HMF).

- Determine a more physical explanation for a function that encapsulates the collapse mechanisms for halos.

What is the HMF?

It’s not…



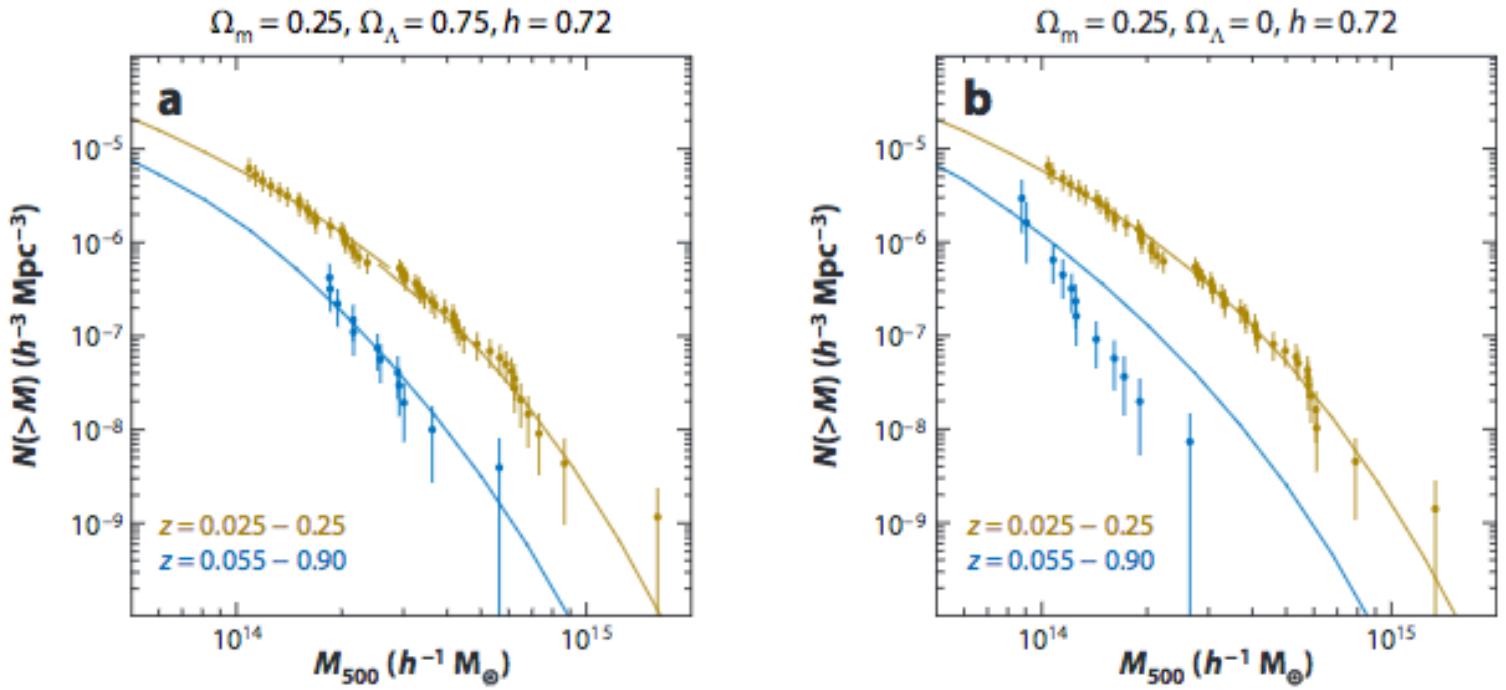
HMF (cont.)

- Halo mass function (HMF) is used to predict the number of halos for given mass range per volume element.

$$\frac{dn}{dM} = \frac{\rho_m}{M} \frac{dln\sigma}{dM} f(\sigma)$$

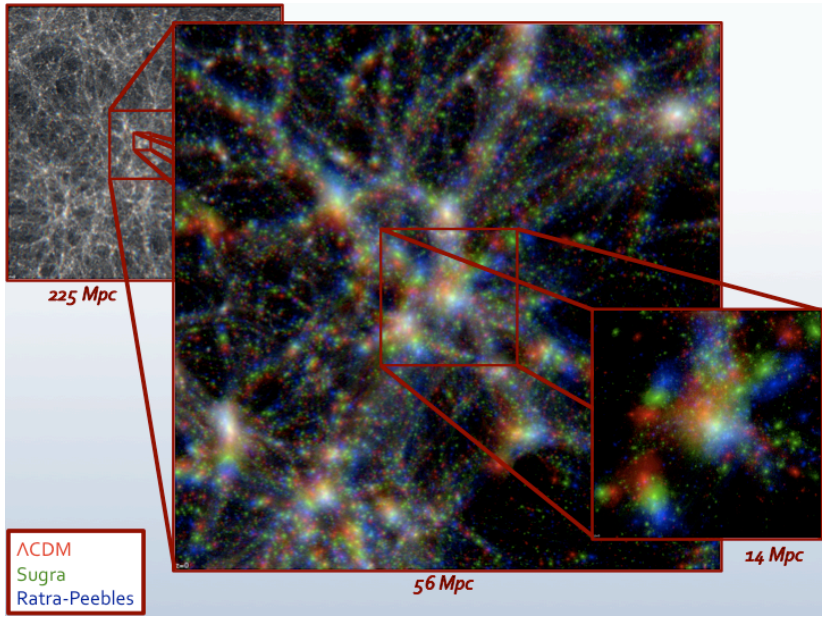
HMF (cont.)

- Constrain cosmological models using large hydrodynamical N-body simulations.
- ‘Large’ datasets (up to trillions of particles, positions, velocities etc.)



Data files: format and endianness

- Binary files in big endian format.
- Number of particles => masses of halos => histogram => HMF.

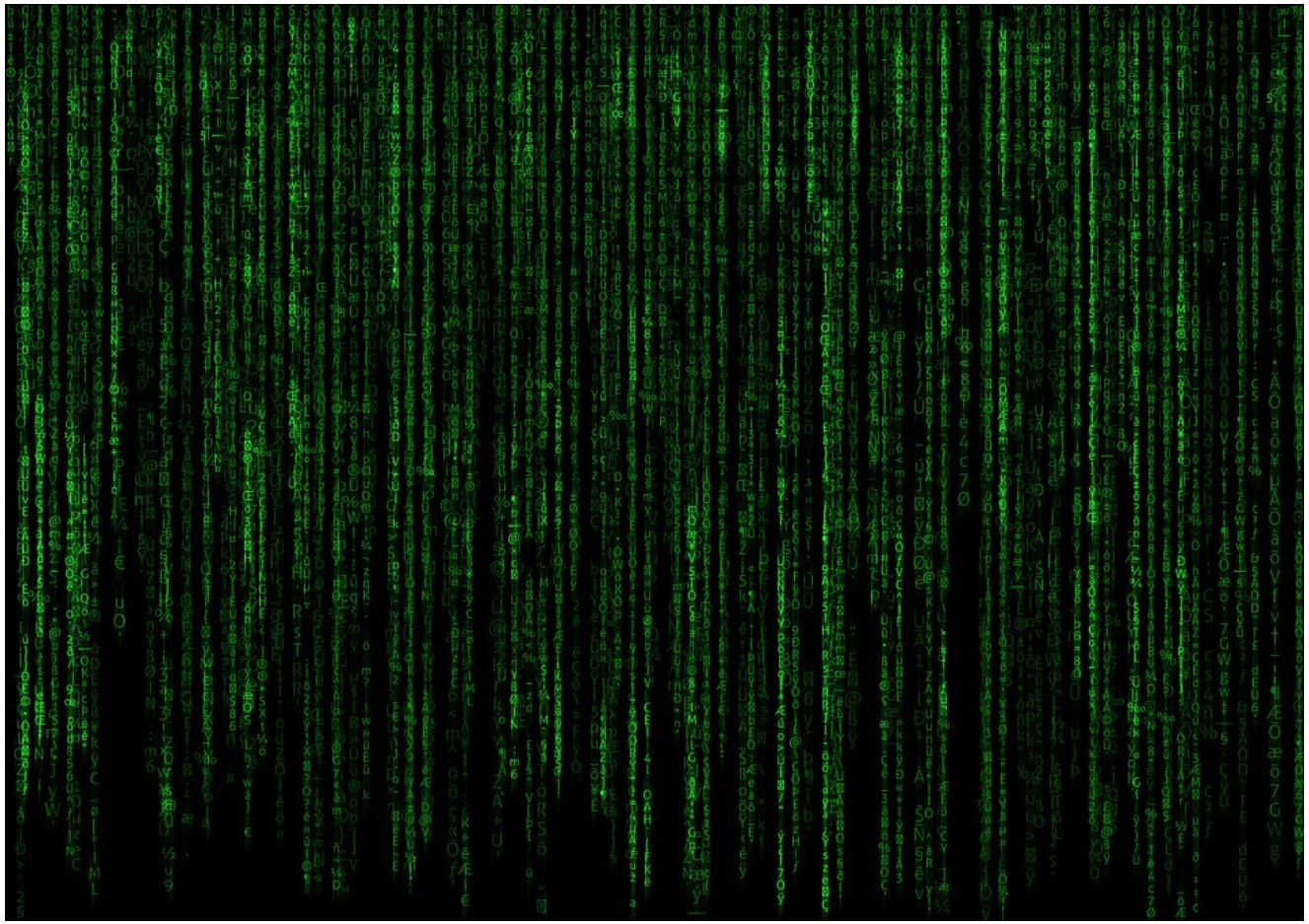


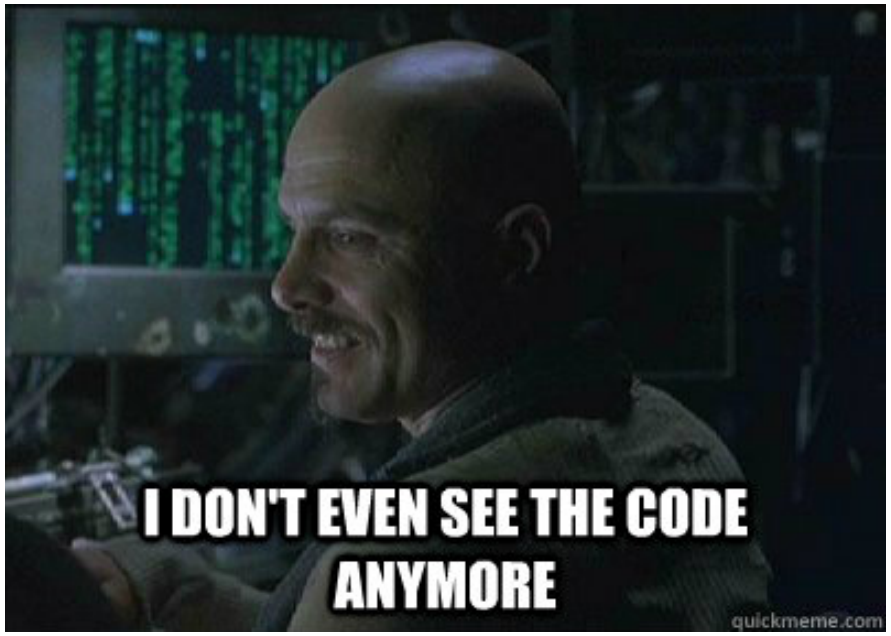
Halo position file description

nhalos	integer(4)	Number of halos in the file
positions	(int(8)+int(4)+real(4)∗3)∗npart	id,npart,x,y,z, of each halo

Let’s look at some code

after hours of debugging…





Working around slow read ins

- Python reads in files real slow.
- Have enough memory? In this case yes (only 170 million halos!)
- Read in text file, write out binary (at least 10x faster for read in).

What if I don’t have enough memory?

- Read in line by line (yeah but python loops are real slow, remember?)
- Read in chunks?
- Python package pandas has ‘chunksize’ arguments for read in.
- Any other ideas?

The end…


WHAT IF I TOLD YOU
THAT BEER YOU ARE DRINKING ISN'T A CRAFT BEER


DATA SCIENTISTS
CAN PARTY
JAMSO
imgflip.com