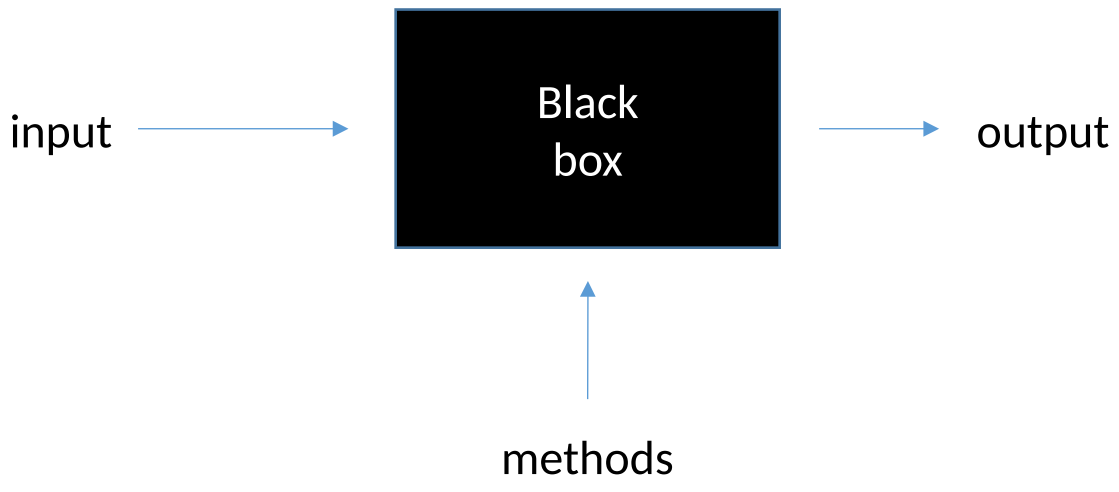
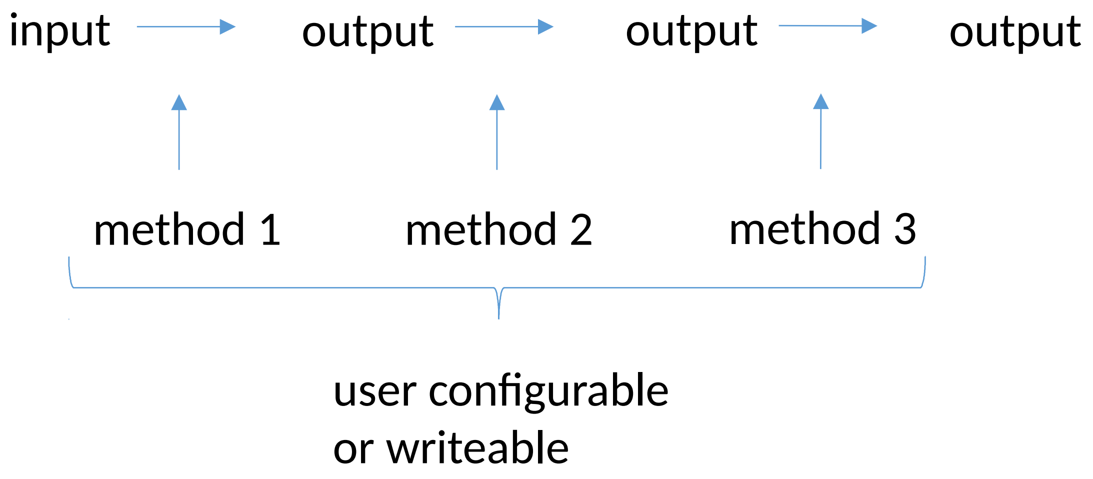
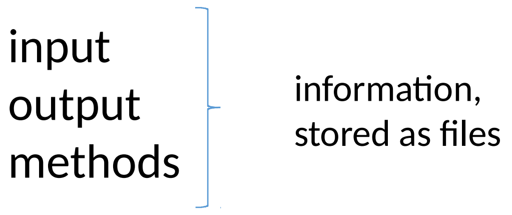
Computers, file systems and data

What do we mean by ‘computer’?



A TV or smart toaster is not a ‘computer’





Specifying a process (work flow / pipeline)

$$\begin{tikzcd}
\text{input} \arrow[rr] & {} & \text{output} \arrow[rr] & {} & \text{output} \arrow[rr] & {} & \text{output} \\
\text{options} \arrow[r] & {} & \text{options} \arrow[r] & {} & \text{options} \arrow[r] & {} & \\
& \text{method 1} \arrow[uu] & & \text{method 2} \arrow[uu] & & \text{method 3} \arrow[uu] &
\end{tikzcd}$$

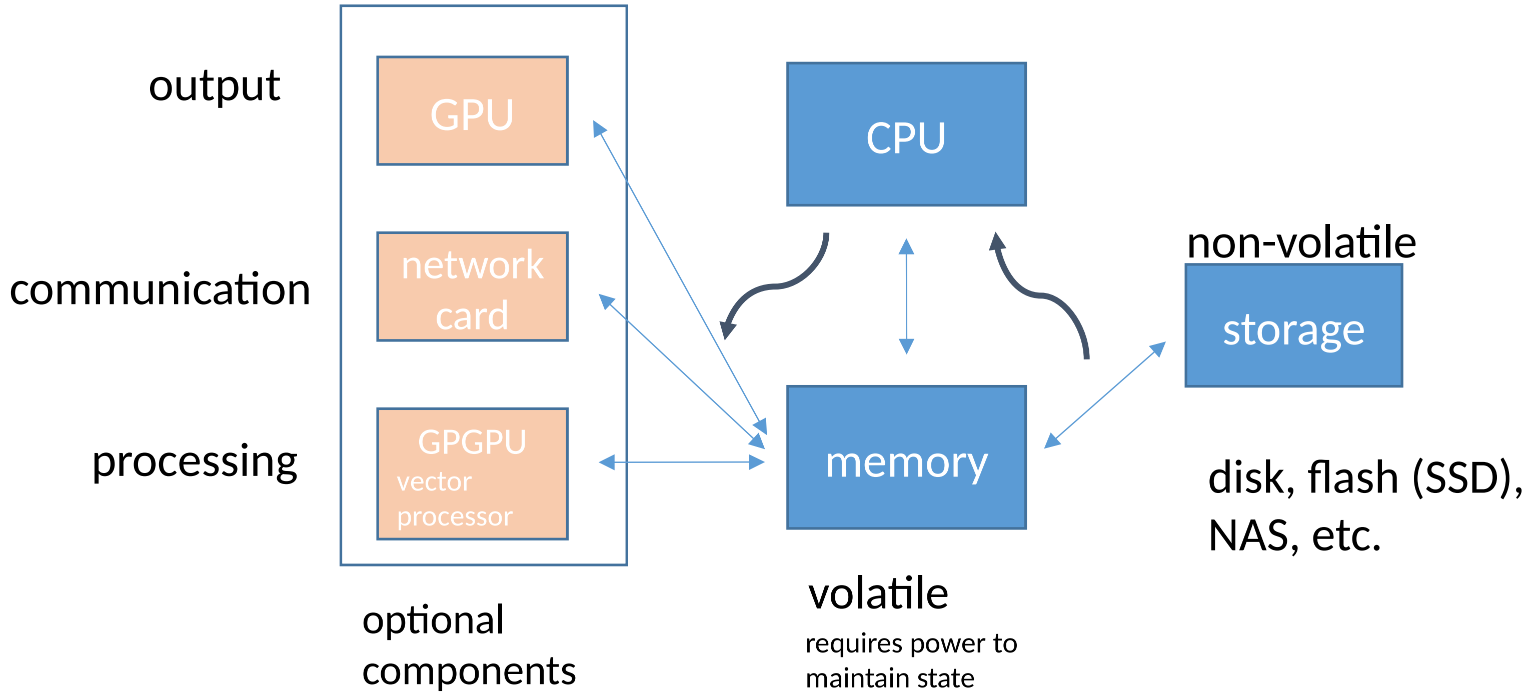
$$\underbrace{\text{method 1} \quad \text{method 2} \quad \text{method 3}}_{}$$

easily specified as a script
containing a series of
commands

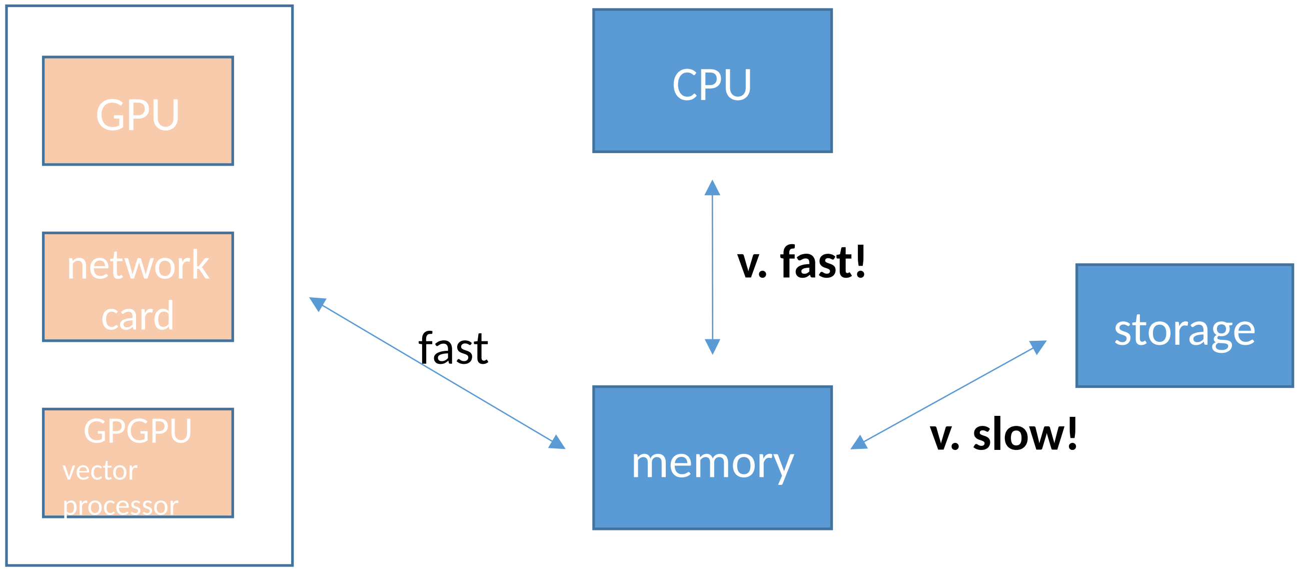
Easy with good command line interface.

Possible with a graphical interface, but this almost inevitably limits options, and difficult to implement. **Does not reduce complexity**. Only good if you don't need to understand what you are doing.

The parts of a computer



Component communication



accessing storage can be 1000s times slower than memory

(but SSDs and emerging tecnologies are changing this)

Processor units

CPU

1-32 independent cores
access data in main memory

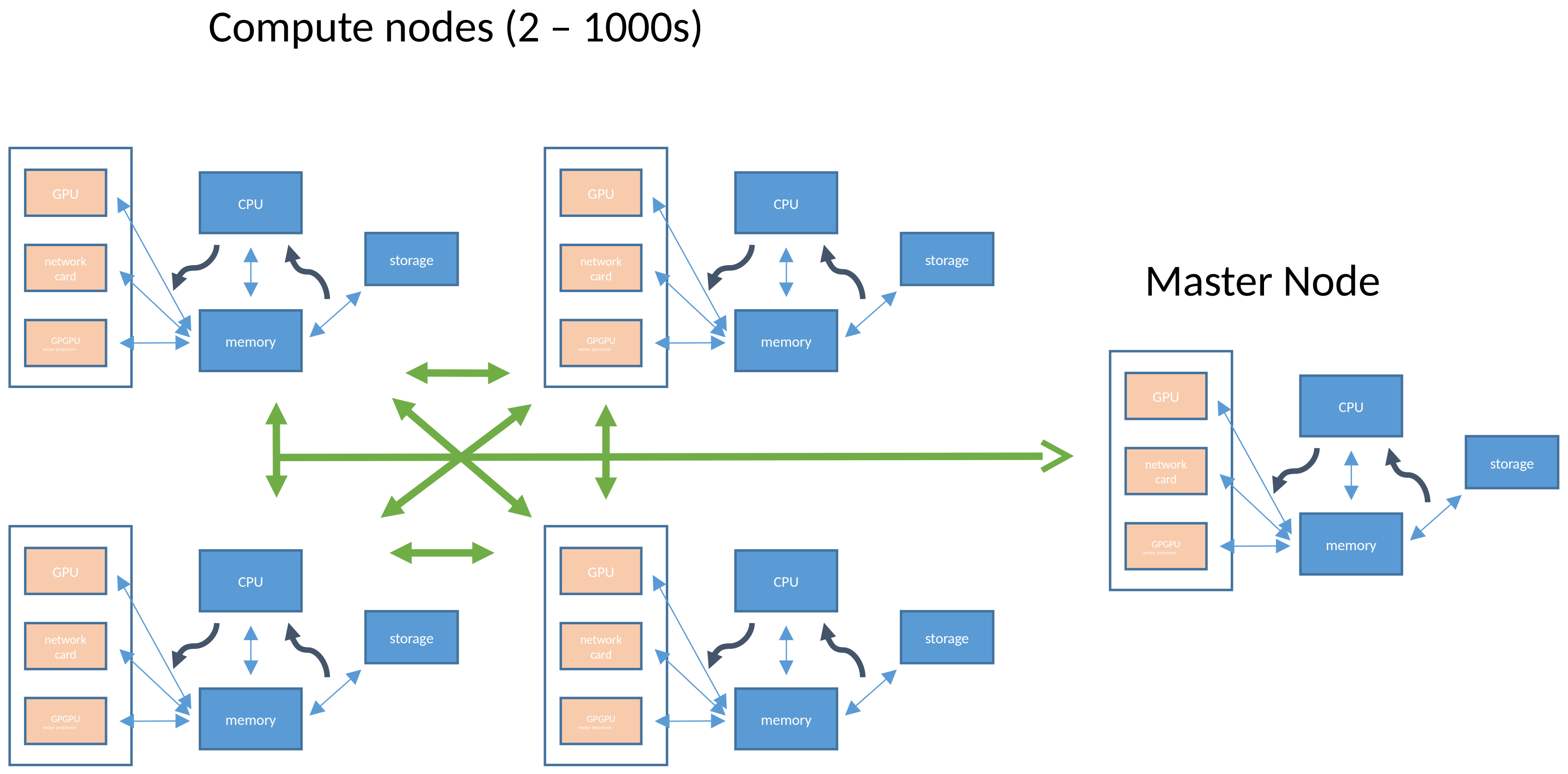
GPU

100s-1000s independent cores
access data in graphics card memory

- Each core can run independent calculations*
- GPUs can typically process data much faster than CPUs, but they are more limited in terms of the type of processing.
- Processing speed of GPU often limited by memory access times.

*rather oversimplified for GPUs

Clusters of compute nodes



Running a program

somewhat simplified:

1. Program loaded from disk to memory
2. Instructions read from memory to CPU
3. Instructions executed one by one*
    a) data loaded to memory from storage**
    b) data & instructions loaded to CPU
    c) calculations carried out on data in CPU
    d) results of calculations written to memory & storage

- data has to be loaded to memory to be worked on
- memory quantity often limiting factor

* with multiple cores or CPUs several instructions can be carried out in parallel
** data can also be entered by user (eg. word processor) or loaded from the network or any combination of the above.

Operating Systems

- Start and manage processes (~applications)
- Manage memory
- Manage communication with physical devices (eg. network card, keyboard, mouse, monitor, disk system, etc…)
- Manage users and their privileges

User interfaces (shells)

- Provide an interface between the user and the computer
- Referred to as shells. Roughly divided into 2 types:
    - Command line interfaces (CLI). Commands typed in plain text using a keyboard.
    - Graphical user interfaces (GUI). Commands to computer entered by clicking, dragging & dropping icons all over the place.

Operating Systems & Shells

Boundaries between OS and shell often rather blurred.

- Windows & MacOS: Includes both low level process management and user interfaces
- Linux: depends on who's talking
  - Android: runs on Linux kernel (the low level OS), but uses different user (and application) interfaces. Not usually considered as Linux.
  - Linux distributions (eg. Ubuntu, Fedora, SuSE): run on Linux kernel, include multiple user interfaces and large collections of applications.

regardless, the different OSes, all pretty much do the same thing, and to the user it is primarily the interface and applications that matter

User interfaces (shells)

Graphical shells

- good discoverability:
  click around randomly and something is likely to happen, and you can learn from this
- good for common tasks:
  well defined tasks can be implemented in efficient ways (example ?)
- good for manual input (eg. drawing, etc…)
- difficult to design flexible interfaces & to combine arbitrary tasks

Command line shells

- bad discoverability:
  random typing unlikely to get you very far
- difficult to remember infrequently used commands
- no graphical feedback, only text
- user must be able and willing to read

essential for bleeding edge $\left[\begin{array}{l} \text{} \end{array}\right.$

- easy to create interfaces
- trivial to combine arbitrary tasks
- great expressive power

Which Operating System?

in Bioinformatics: **Linux**

1. Open system allowing modification at any level (important for high performance computing (HPC)).
2. No licensing costs (also important for HPC which may use very large numbers of computers).
3. Standard compilation toolchain, making it easy to:
   1. write new programs
   2. compile and run programs created elsewhere
   3. understand program logic by inspecting source code
   4. fix bugs or add features to programs
4. Standard scripting environment (Perl, Python, Bash, etc…)
5. Nice command line environments (Bash & terminals)

the place where all the tools work…

Data & files

data and programs are usually* stored in files

file:	a named unit of data that can be accessed through the file system

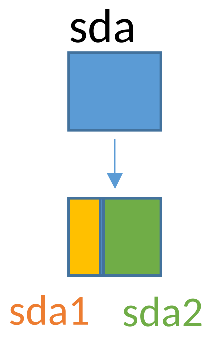
- independent set of data
- readable name specified by a user or a program

file system:	a system that keeps track of the physical locations of files

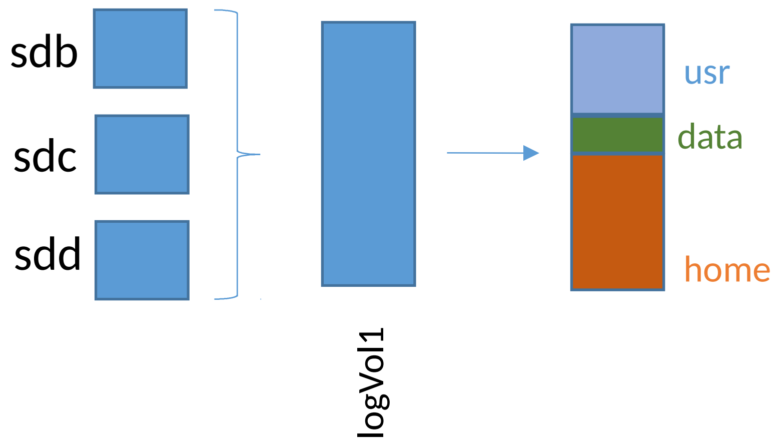
- divides up physical media into user addressable regions
- keeps track of:
    - locations of data associated with a file
    - file metadata (eg. creation time, ownership, permissions)

*data can also be in databases, but even then it's often stored within a set of files

Disks & file systems on Linux



- a physical device (usually a disk) can be divided into several partitions.
- each partition is an independent file system (there are several types)

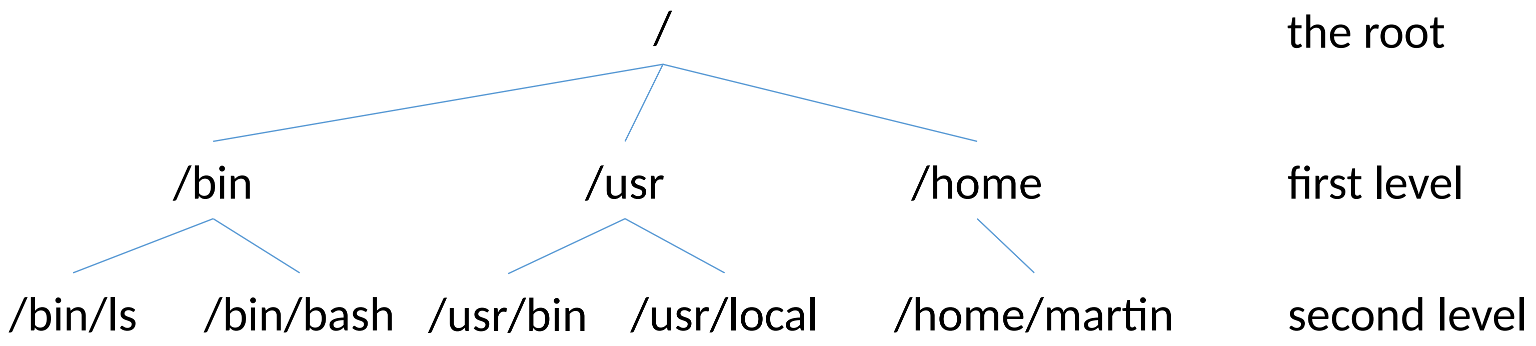


- several physical devices can be merged into a logical volume
- the logical volume can then be divided into several partitions, each with it's own filesystem

(This is not so important for you!)

Disks & file systems on Linux

Linux uses a singly rooted hierarchical tree of directories to specify locations of files:



- partitions may be mounted anywhere within the hierarchy
- /home contains user data and is often mounted on a seperate partition. This allows the rest of the system to be reformated without touching user data
- partitions can also be present on networked computers

MacOS is Unix and does the same thing. Windows does that thing with C:, D: and so on, reserving still A: and B: for floppy disks. Except it now seems to want to complicate things, so I don't really know what's going on there.

(But this is rather useful)

File names

The complete file name contains the location in the hierarchy:

```
/home/lmj/genomes/bowtie_index/Gv1_1
```

and the file name:

```
Gv1_cs_command
```

with directories denoted by the ‘/’ character:

```
/home/lmj/genomes/bowtie_index/Gv1_1/Gv1_cs_command
```

renaming & moving the file are the same thing (as the full name contains the location)

- moving to a different location on the same file system is a simple rename procedure
- moving to a different filesystem involves copying the file to the new system and deleting from the old

Programs and the file system

- Running processes are also associated with a location in the file system
- Location depends on how the application is started
    - the command line: then the location of the command line process
    - graphically: implementation dependent (i.e. I'm not sure)
- A running process usually has the same permissions as the user that started it (in terms of which files can be read and modified)

Specifying file system locations

- Absolute path: simply the full path starting from /.
  eg: /home/lmj/genomes/cod/Gv1/Gv1.fa
- Relative path: relative to the current working directory of the process. If
  CWD = /home/lmj and
  full path = /home/lmj/cod/Gv1/Gv1.fa
    - relative path = cod/Gv1/Gv1.fa

File permissions

a file is associated with:

- an owner
- a group of users

a file can be any combination of:

- executable (1)
- writable (2)
- readable (4)

specified independently for:

- the owner
- the group
- the world

File permissions

to see file permissions from a terminal use ‘ls -l’,
combined with ‘-h’ below.

```
>ls -lh
-rwxrwxr-x 1 lmj lmj 2,4K juni  16  2015 bowtie
-rw-r--r-- 1 lmj lmj 1,4K mars   9  2015 bowtie_main.cpp
```

- bowtie: an executable file has ‘read, write, execute’ permissions for the owner and group members and ‘read, execute’ permissions for anyone.
- bowtie_main.cpp: a source code file has ‘read, write’ permissions for the owner and ‘read’ only for other users.

Setting file permissions and bitwise masks

- File permissions are specified by 3 numbers, one each for owner, group and world.
- Each number is specified by the binary OR combination of 1 (execute), 2 (write) and 4 (read)

```
>ls -lh
-rwxrwxr-x 1 lmj lmj 2,4K juni  16  2015 bowtie
-rw-r--r-- 1 lmj lmj 1,4K mars   9  2015 bowtie_main.cpp
```

Here, the permission are:

bowtie:	775
bowtie_main.cpp:	644

How does that work?

Binary notation

1 OR 1 = 1
1 OR 0 = 1
0 OR 1 = 1
0 OR 0 = 0

powers of 2

decimal	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
4	0	1	0	0
OR	0	1	1	1

= 7

eg.:

read (4)	write (2)	execute (1)	decimal	
0	0	0	0	no permission
0	0	1	1	execute
0	1	1	3	read, execute
1	1	1	7	read, write, execute

bitwise masks are commonly used for encoding data flags in lots of file formats.
You should understand their use.

Running programs

(from the command line)

- Only files that have the executable bit can be run
- If the file containing the application you wish to run is present in a directory listed in the PATH variable:
  - Just type the name of the program (and hit ENTER)
- Otherwise specify the path:
  - Absolute path starting with /
    /usr/local/bin/bowtie
  - Relative to current working directory (CWD). eg. if CWD = /usr/local
    bin/bowtie
  - If file in CWD (CWD = /usr/local/bin):
    ./bowtie

Program options and arguments

Input data is usually passed as arguments
Options are usually passed using - or -- notation.

```
bowtie -C -f -p 10 -S -t --chunkmbs 2000 --fr -u 1000000 --skip 5000000 \
       -n 3 -e 100 -m 1 \
       -Q seq1.QV.qual  \
       ~/genomes/bowtie_index/Gv1_1/Gv1_cs \
       seq1.csfasta \
       test_single.sam
```

- Anything following the name of the program is an argument
- Arguments are seperated by spaces
- Arguments beginning with a - or – are options which may be optional
- the ‘\’ character is the universal escape character. Here it is used to escape the special meaning of end of line (to run the command).

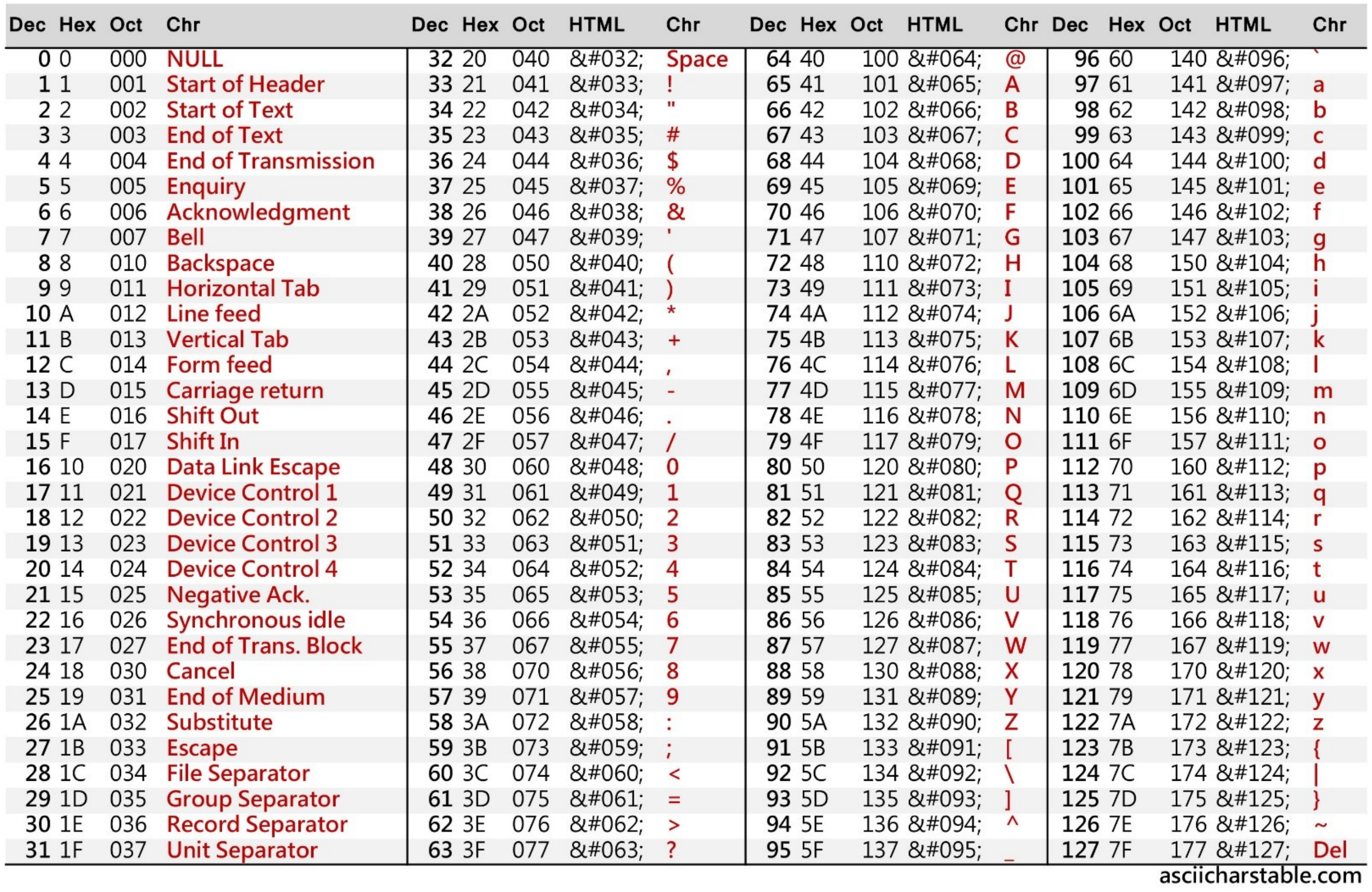
Data and file types

text	human readable formats. Contain plain text in a variety of formats. Numbers are represented using text encoding. eg: fasta, fastq, sam, wiggle, csv.
binary	non-readable. May contain anything including text and numbers. Numbers are represented a binary format. Files may be compressed and require decompression to be used. eg: bam, bigwig, bigbed, Microsoft Word, Excel, etc…

this is not
a text file!

Text encoding

In the old days, text was simple. It was encoded using ASCII. One byte (8 bits) represented one character. The lower valued bytes representing non-printable characters.



Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

fastq uses ASCII character encoding to use single readable bytes to represent 66 individual values

Text encoding

Today text includes encodings for languages across the world, many of which have more than 256 printable characters. Needs more than one byte.

UTF16	2 or 4 bytes per character. Used mainly by Windows
UTF8	1,2,3,4 bytes per character. Used by everyone else
UTF32	4 bytes per character. Mostly internal use.

bioinformaticians usually care about ACTG, so bioinformatics packages may have problems with text that contains non-ASCII characters.

encoding DNA sequence in UTF16 would be rather silly.

Text file formats

- use structured text
- often use end of line to seperate features

the fasta sequence file format (the universal sequence format)

```
>seq1Name optional comments
ACTAGACTAGACTAGAGACATACATAGVAACATAGACAT
ATACGATACATAGACGATA
>seq2Name optional comments
ACTAGAGACTAGACTAGAGAGACATGACTACGATACGAAGA
ACGTAGACTAGACAGATAVCAG
```

the fastq sequence file format (the next generation sequence format)

- Line 1 begins with a '@' character and is followed by a sequence identifier and an *optional* description (like a FASTA title line).
- Line 2 is the raw sequence letters.
- Line 3 begins with a '+' character and is *optionally* followed by the same sequence identifier (and any description) again.
- Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.

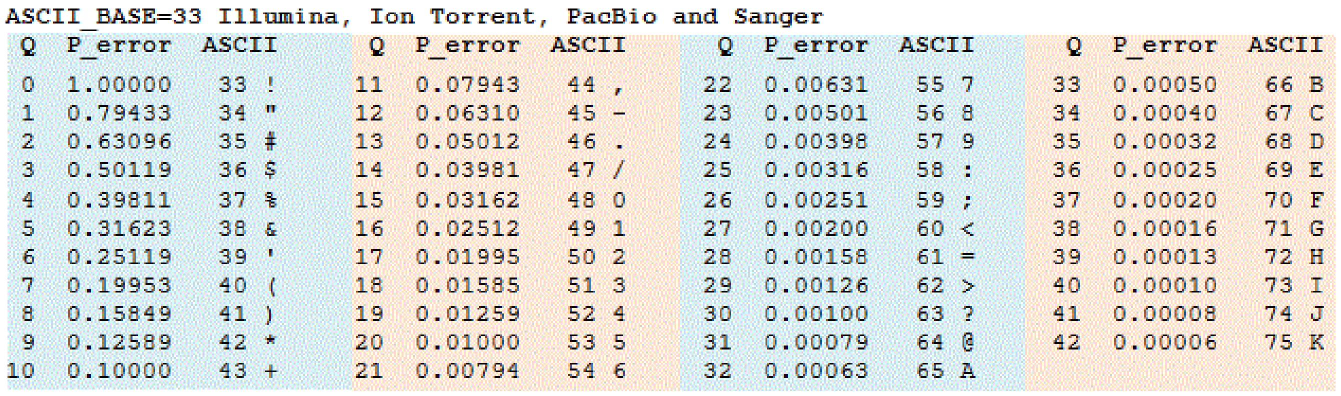
```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

quality values??
these start from '!' or
something else

Phred score encoding

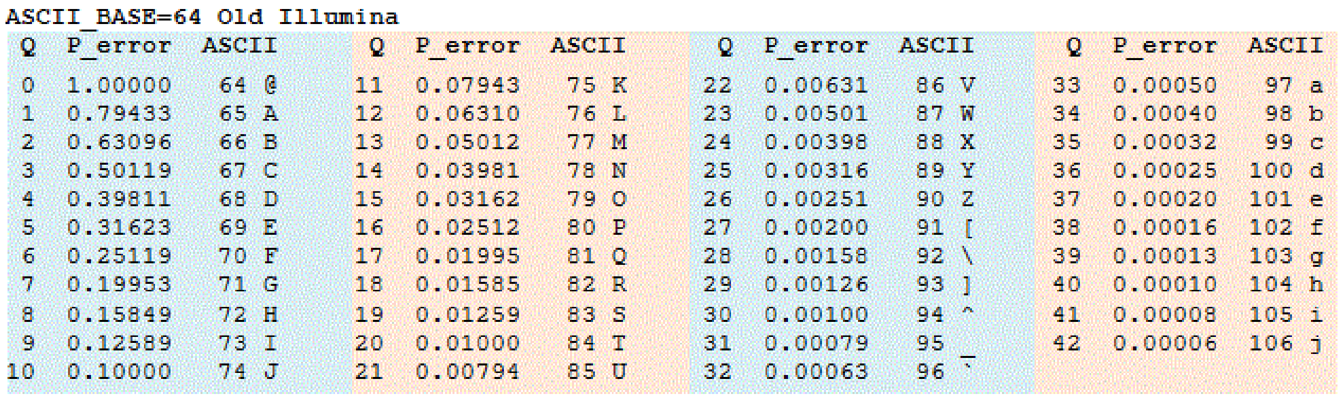
Q is Phred score and P is the error of an error, then:

$Q = -10 \log_{10}(P)$



ASCII_BASE=33 Illumina, Ion Torrent, PacBio and Sanger

Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII
0	1.00000	33 !	11	0.07943	44 ,	22	0.00631	55 7	33	0.00050	66 B
1	0.79433	34 "	12	0.06310	45 -	23	0.00501	56 8	34	0.00040	67 C
2	0.63096	35 #	13	0.05012	46 .	24	0.00398	57 9	35	0.00032	68 D
3	0.50119	36 $	14	0.03981	47 /	25	0.00316	58 :	36	0.00025	69 E
4	0.39811	37 %	15	0.03162	48 0	26	0.00251	59 ;	37	0.00020	70 F
5	0.31623	38 &	16	0.02512	49 1	27	0.00200	60 <	38	0.00016	71 G
6	0.25119	39 '	17	0.01995	50 2	28	0.00158	61 =	39	0.00013	72 H
7	0.19953	40 (	18	0.01585	51 3	29	0.00126	62 >	40	0.00010	73 I
8	0.15849	41)	19	0.01259	52 4	30	0.00100	63 ?	41	0.00008	74 J
9	0.12589	42 *	20	0.01000	53 5	31	0.00079	64 @	42	0.00006	75 K
10	0.10000	43 +	21	0.00794	54 6	32	0.00063	65 A			



ASCII_BASE=64 Old Illumina

Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII	Q	P_error	ASCII
0	1.00000	64 @	11	0.07943	75 K	22	0.00631	86 V	33	0.00050	97 a
1	0.79433	65 A	12	0.06310	76 L	23	0.00501	87 W	34	0.00040	98 b
2	0.63096	66 B	13	0.05012	77 M	24	0.00398	88 X	35	0.00032	99 c
3	0.50119	67 C	14	0.03981	78 N	25	0.00316	89 Y	36	0.00025	100 d
4	0.39811	68 D	15	0.03162	79 O	26	0.00251	90 Z	37	0.00020	101 e
5	0.31623	69 E	16	0.02512	80 P	27	0.00200	91 [	38	0.00016	102 f
6	0.25119	70 F	17	0.01995	81 Q	28	0.00158	92 \	39	0.00013	103 g
7	0.19953	71 G	18	0.01585	82 R	29	0.00126	93]	40	0.00010	104 h
8	0.15849	72 H	19	0.01259	83 S	30	0.00100	94 ^	41	0.00008	105 i
9	0.12589	73 I	20	0.01000	84 T	31	0.00079	95 _	42	0.00006	106 j
10	0.10000	74 J	21	0.00794	85 U	32	0.00063	96 `			

unusual in new data

http://drive5.com/usearch/manual/quality_score.html

Encoding numbers

integers
- 1 (char), 2 (short), 4 (int), or 8 (long) bytes
- signed
- unsigned

	s char	u char	s short	u short	s int	u int	s long	u long
bytes	1	1	2	2	4	4	8	8
min	-128	0	-32768	0	-2,15E+09	0,00E+00	-9,2E+18	0
max	127	255	32677	65535	2,15E+09	4,29E+09	9,22E+18	1,84E+19

from /usr/include/limits.h

floating point (reals)
- encode continuously variable values
- 4 (float) or 8 (double) bytes

not so easy to get the limits for floats & doubles

the bit depths used for the encoding can vary, but for most machines this will be true.

Numbers and genomes

Note1: a signed 32 bit integer (pretty much the default) has a maximum value of about 2e9

that’s less than the size of the human genome
we do need 64 bit integers.

Note2: A 32 bit computer can directly address about 4e9 bytes of memory

an uncompressed human genome needs about 3 GB
we do need 64 bit machines

Note3: An uncompressed suffix array index of the zebra fish genome requires about 25 GB of memory.

more memory is a good thing

using the wrong sized integers can and does lead to problems in a lot of applications

Why bother learning this?

biological data sets are big and getting bigger

straining the limits of the usual computers

you should be able to estimate what computing resources are reasonable for your question

eg. how much memory do you need?

no. data points * (data point size)

data points: bytes, shorts, ints, longs, floats ?