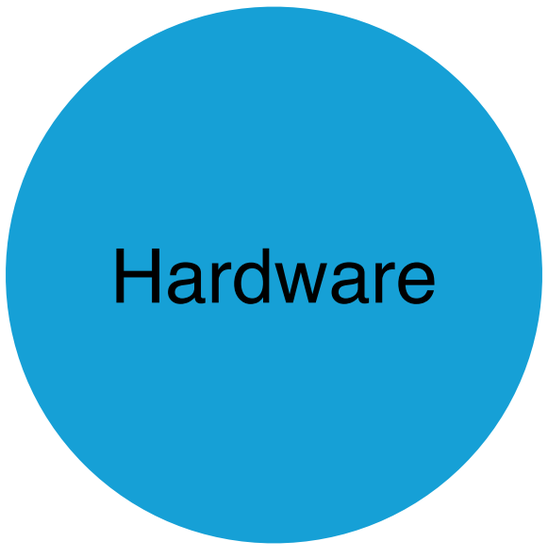
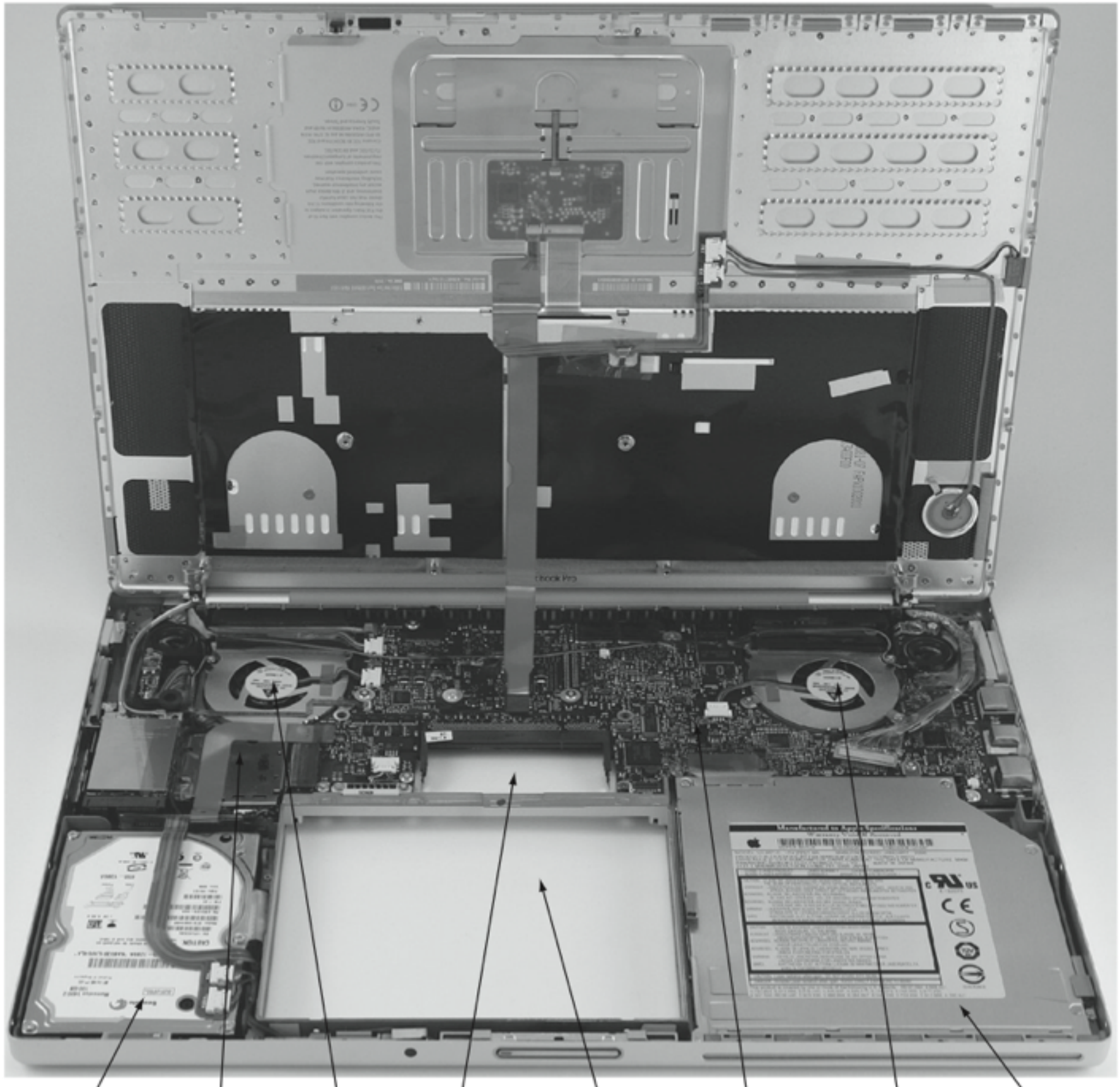
xkcd

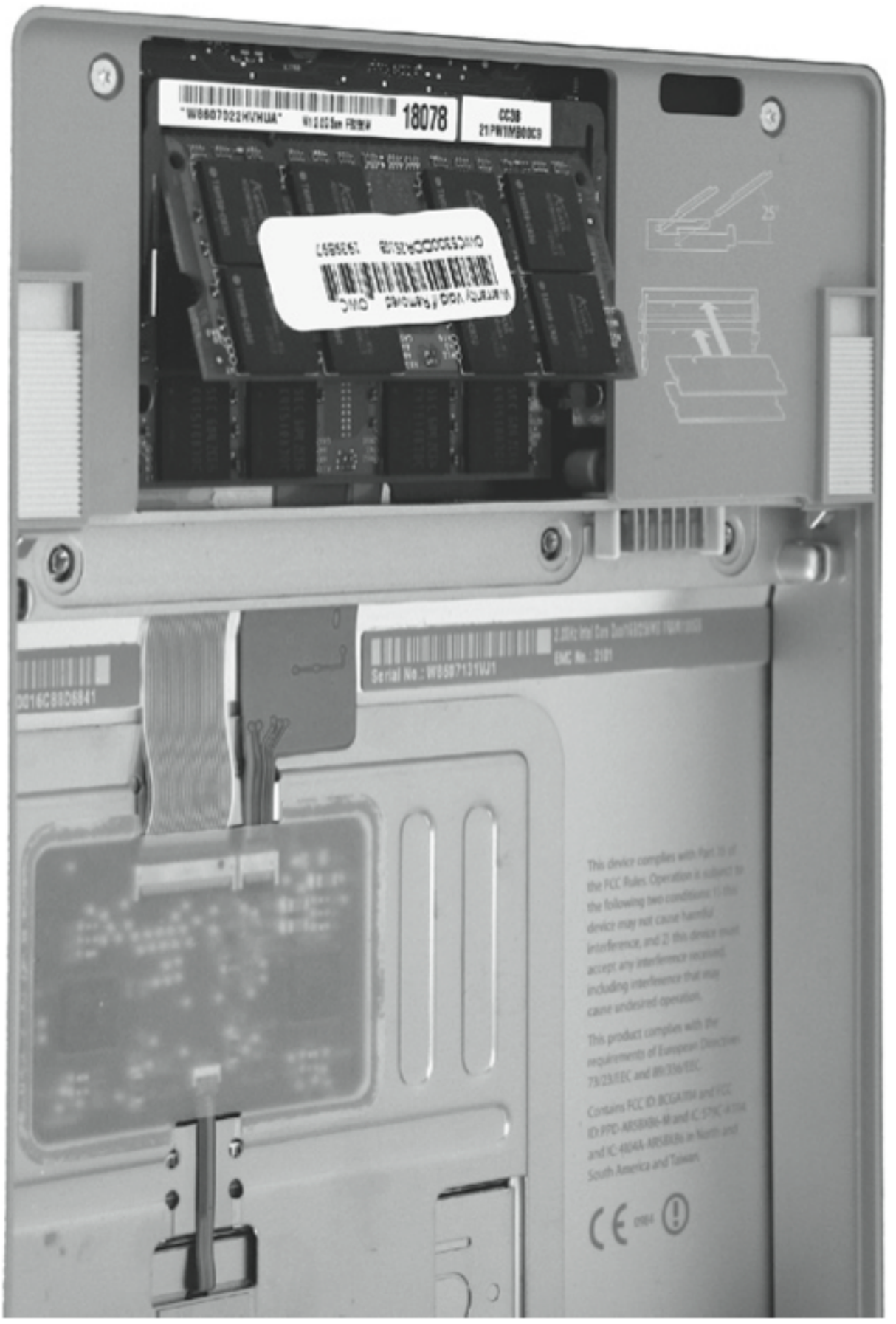
http://xkcd.com/208/

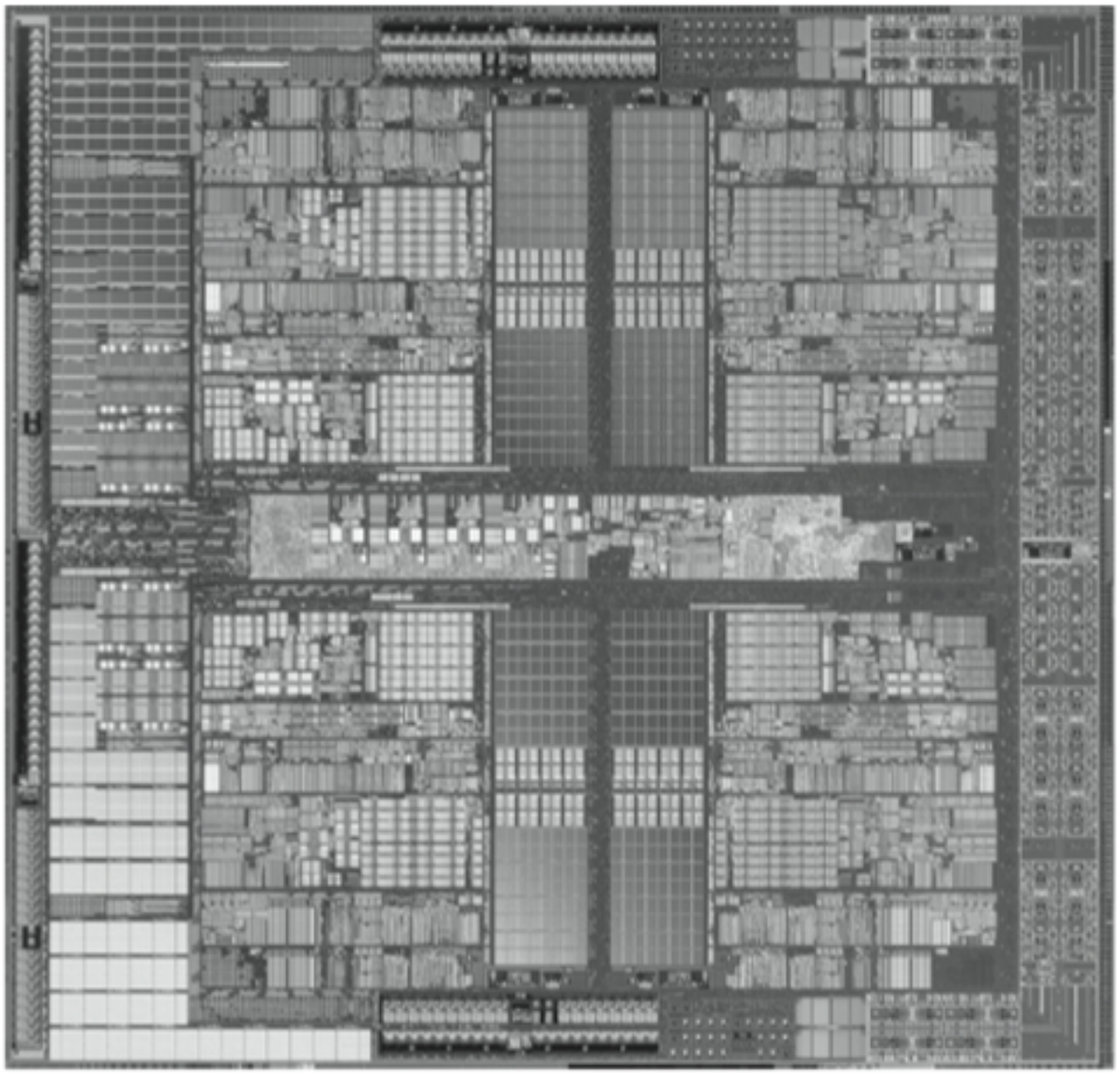
1. Computer Hardware
2. Review of pipes
3. Regular expressions
4. sed
5. awk
6. Editing Files
7. Shell loops
8. Shell scripts

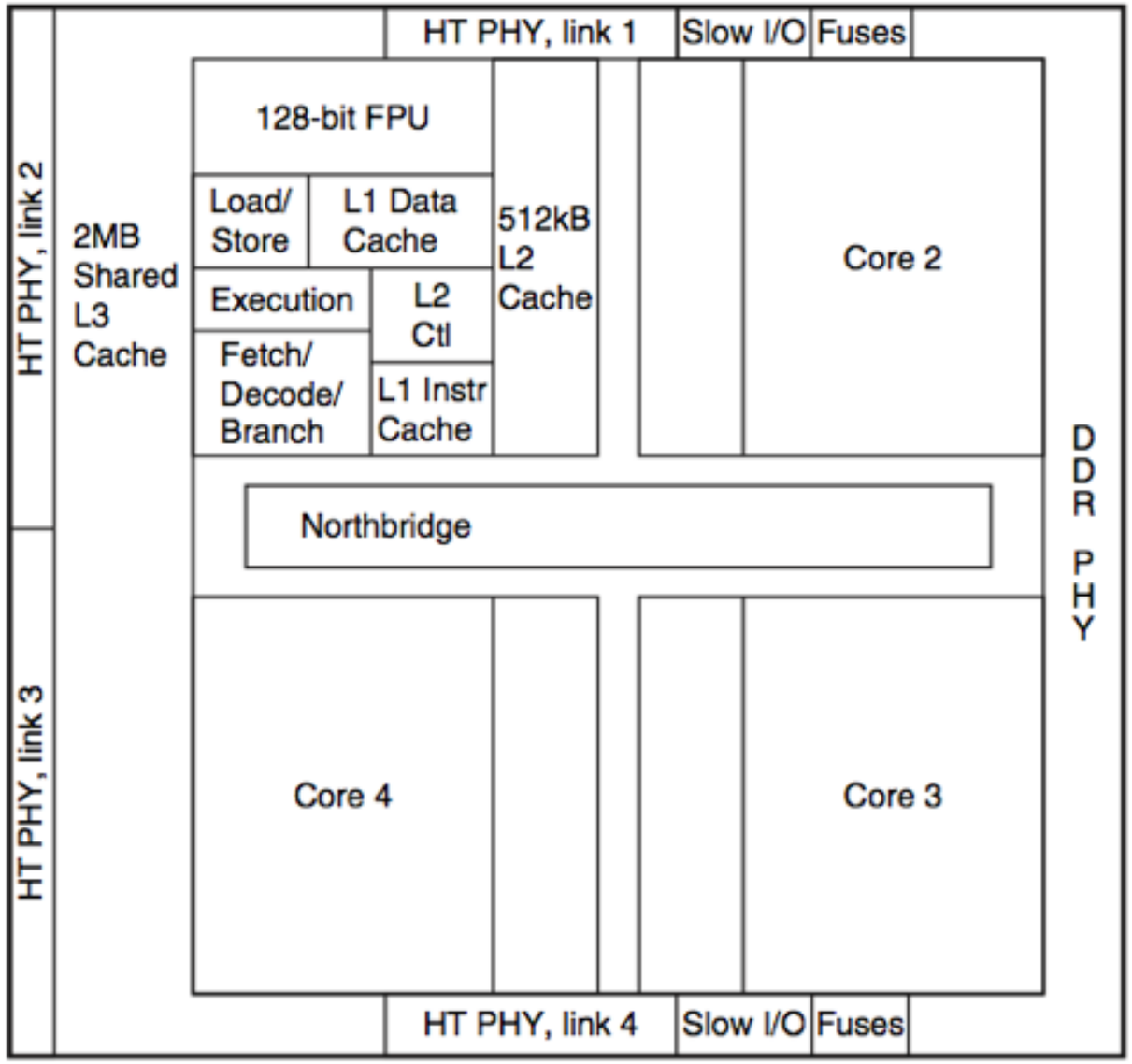

Hardware


Hard drive
Processor
Fan with cover
Spot for memory DIMMs
Spot for battery
Motherboard
Fan with cover
DVD drive

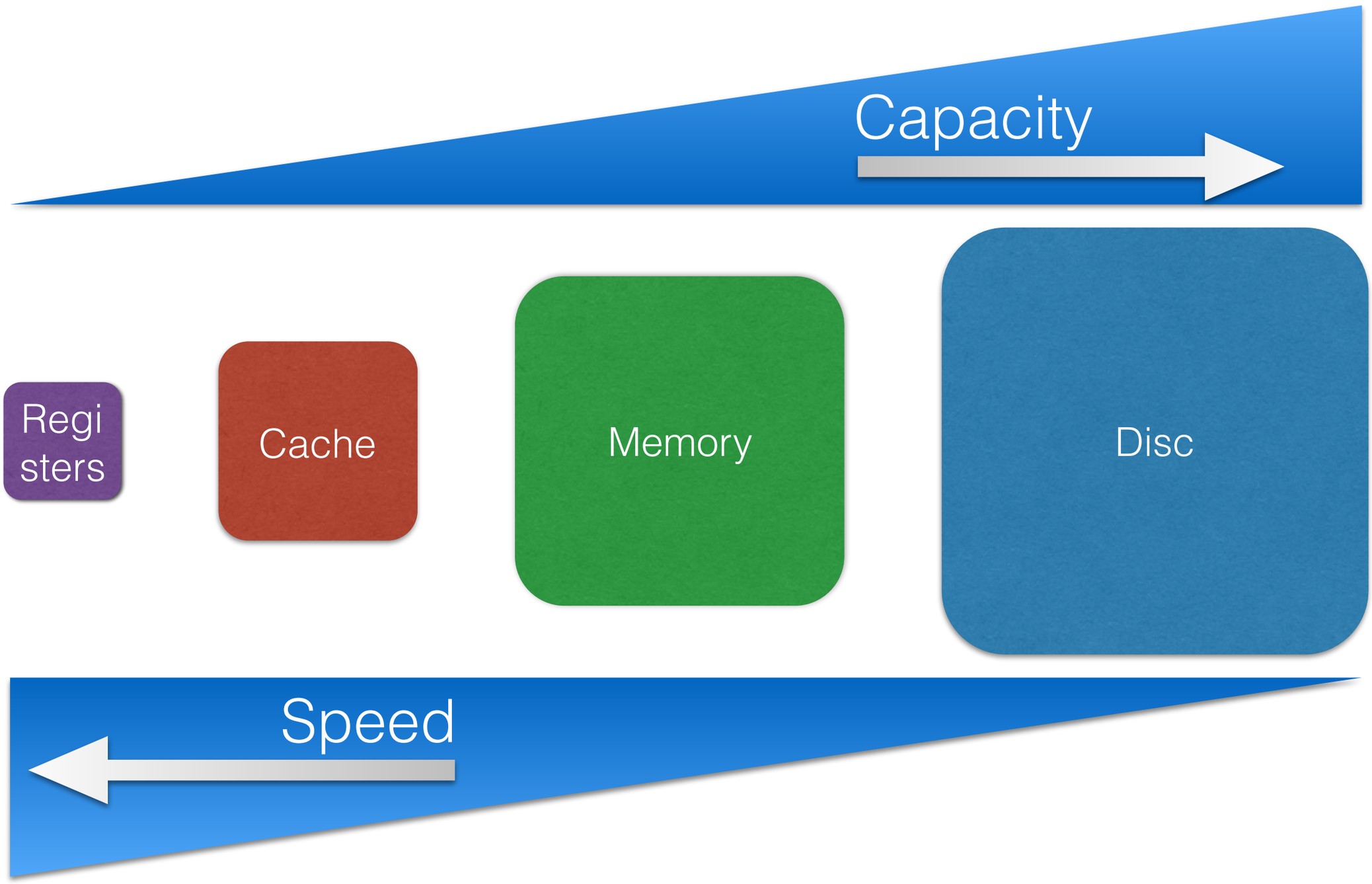
http://www.theverge.com/2011/11/23/2582677/thailand-flood-seagate-hard-drive-shortage


18078
CC38
Serial No : W8807131WJ1
This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.
This product complies with the requirements of European Directives 73/23/EEC and 89/336/EEC.
Contains FCC ID: BCGA1114 and FCC ID: PPD-AR5008x-M and IC: 579C-4114 and IC: 4104A-AR5008x in North and South America and Taiwan.

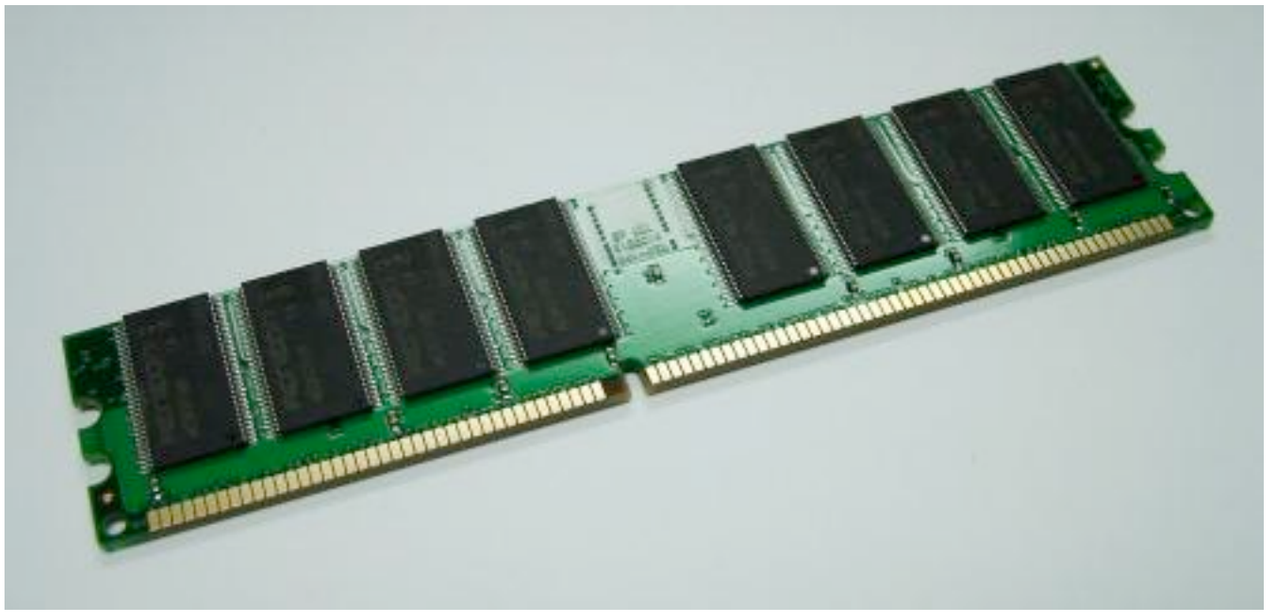


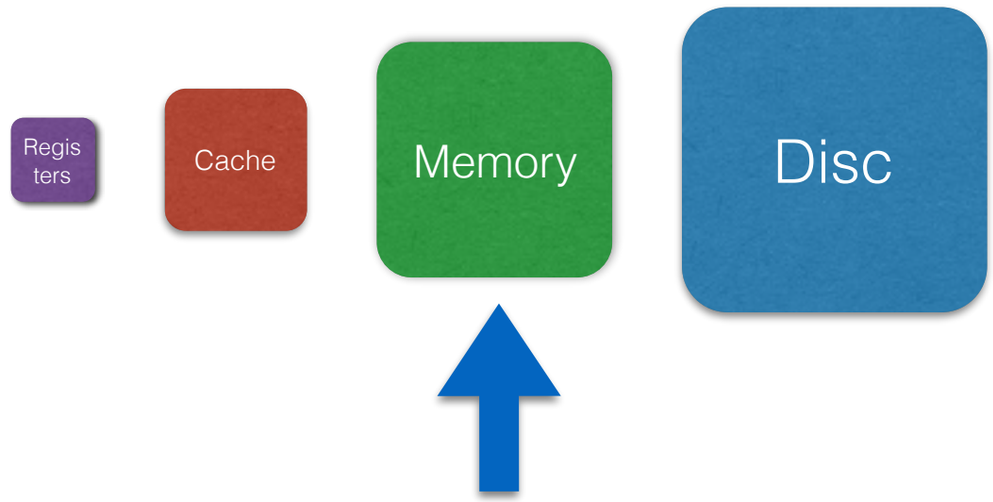

HT PHY, link 1
Slow I/O
Fuses
128-bit FPU
Load/ Store
L1 Data Cache
Execution
L2 Ctl
Fetch/ Decode/ Branch
L1 Instr Cache
512kB L2 Cache
Core 2
HT PHY, link 2
2MB Shared L3 Cache
DDR PHY
Northbridge
HT PHY, link 3
Core 4
Core 3
HT PHY, link 4
Slow I/O
Fuses

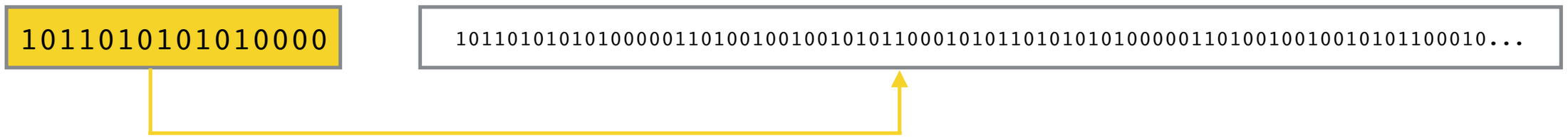
Storing code and data



- Disk access time: 5–20 milliseconds
- Memory access time: 50–70 nanoseconds— 100,000 times faster
- The cost per gigabyte of disk is 30 to 100 times less expensive than memory







Memory: RAM - Random Access Memory

32-bit computer: address fields are 32 bits wide

allows 2^{32} possible addresses, or 4Gb

machines are now 64-bit, allowing memories up to 2^{64}

```
>0
TGCAGGTATATCTATTAGCAGGTTTAATTTTGCCTGCACTTGGTTGGGTACATTATTTTAAGTGTATTTGACAAG
>1
TGCAGGTTGTTGTTACTCAGGTCCAGTTCTCTGAGACTGGAGGACTGGGAGCTGAGAACTGAGGACAGAGCTTCA
>2
TGCAGGGCCGGTCCAAGGCTGCATGAGGCCTGGGGCAGAATCTGACCTAGGGGCCCCTCTTGCTGCTAAAACCAT
>3
TGCAGGATCTGCTGCACCATTAACCAGACAGAAATGGCAGTTTTATACAAGTTATTATTCTAATTCAATAGCTGA
>4
TGCAGGGGTCAAATACAGCTGTCAAAGCCAGACTTTGAGCACTGCTAGCTGGCTGCAACACCTGCACTTAACCTC
```

cat seqs.fa

PIPE

grep ACGT

```
TGCAGGTATATCTATTAGCAGGTTTAATTTTGCCTGCACTTGGTTGGGTACATTATTTTAAGTGTATTTGACAAG
>1
TGCAGGTTGTTGTTACTCAGGTCCAGTTCTCTGAGACTGGAGGACTGGGAGCTGAGAACTGAGGACAGAGCTTCA
>2
TGCAGGGCCGGTCCAAGGCTGCATGAGGCCTGGGGCAGAATCTGACCTAGGGGCCCCTCTTGCTGCTAAAACCAT
>3
TGCAGGATCTGCTGCACCATTAACCAGACAGAAATGGCAGTTTTATACAAGTTATTATTCTAATTCAATAGCTGA
>4
TGCAGGGGTCAAATACAGCTGTCAAAGCCAGACTTTGAGCACTGCTAGCTGGCTGCAACACCTGCACTTAACCTC
```

cat seqs.fa

PIPE

grep ACGT

Does
“>0”
contain “ACGT”?

Yes?

No?

Output

NULL

```
>1
TGCAGGTTGTTGTTACTCAGGTCCAGTTCTCTGAGACTGGAGGACTGGGAGCTGAGAACTGAGGACAGAGCTTCA
>2
TGCAGGGCCGGTCCAAGGCTGCATGAGGCCTGGGGCAGAATCTGACCTAGGGGCCCCTCTTGCTGCTAAAACCAT
>3
TGCAGGATCTGCTGCACCATTAACCAGACAGAAATGGCAGTTTTATACAAGTTATTATTCTAATTCAATAGCTGA
>4
TGCAGGGGTCAAATACAGCTGTCAAAGCCAGACTTTGAGCACTGCTAGCTGGCTGCAACACCTGCACTTAACCTC
```

cat seqs.fa

PIPE

grep ACGT

Does
“TGCAGGTATATCTATTAGCAGGTTTAATTTTGCCTGCACTTG...G”
contain “ACGT”?

Yes? → Output

No? → NULL

```
TGCAGGTTGTTGTTACTCAGGTCCAGTTCTCTGAGACTGGAGGACTGGGAGCTGAGAACTGAGGACAGAGCTTCA
>2
TGCAGGGCCGGTCCAAGGCTGCATGAGGCCTGGGGCAGAATCTGACCTAGGGGCCCCTCTTGCTGCTAAAACCAT
>3
TGCAGGATCTGCTGCACCATTAACCAGACAGAAATGGCAGTTTTATACAAGTTATTATTCTAATTCAATAGCTGA
>4
TGCAGGGGTCAAATACAGCTGTCAAAGCCAGACTTTGAGCACTGCTAGCTGGCTGCAACACCTGCACTTAACCTC
```

cat seqs.fa

PIPE

grep ACGT

Does
“>1”
contain “ACGT”?

Yes?

No?

Output

NULL

cat seqs.fa

PIPE

grep ACGT

Does
“TGCAGGGGTCAAATACAGCTGTCAAAGCCAGACTTTGAGCAC...C”
contain “ACGT”?

Yes? → Output

No? → NULL

Regular Expressions

Text often follows human conventions

Dates have a country-specific format:
Europe: day-month-year
US: month-day-year

Telephone numbers: xxx-xxx-xxxx

Zip codes must be five digits long

Structure: certain number of columns or rows

Conventions can change: prior to 2000, dates were always 3 sets of two numbers

Regular Expressions, ctd.

If we can encode search context we can make much more powerful searches.

What sort of information would we like to specify?

Whether text is:

1. Composed of numbers
2. Composed of letter
3. A certain length
4. Full of spaces or tabs
5. First or last on the line

Regular Expressions, ctd.

Encoding	Modern Equivalent	Pattern Type
`.`		a single character
`.+`		one or more characters
`.*`		zero or more characters
`.?`		Maybe present
`^`		first on the line
`$`		last on the line
`[0-9]`	`\d`	digits
`[a-zA-Z]`	`\w`	letters
`' '`	`\s \t`	space
`{3}`		must be exactly 3 characters long
`{3,5}`		between 3-5 characters long
`[ACGT]`		a specific set of characters (a class)

Regular Expressions, ctd.

Expression	Regular Expression
341341	.....
	[0-9]+
	[0-9]{6}
julian catchen	[a-z]+ [a-z]+
541-485-5128	[0-9]{3}\-[0-9]{3}\-[0-9]{4}
	[0-9\-]+
June 3, 1978	[a-zA-Z]+ [0-9], [0-9]{4}
AGCCCCTAGGACTGAAATTCC	[ACGT]+

Regular Expressions, ctd.

Expression	Regular Expression
341341	“......”
	“[0-9]+”
julian catchen	“[a-z]+ [a-z]+”
541-485-5128	“[0-9]{3}\-[0-9]{3}\-[0-9]{4}”
	“[0-9\-]+”
June 3, 1978	“[a-zA-Z]+ [0-9], [0-9]{4}”
	“[a-zA-Z]+ [0-9]+, [0-9]{4}”
	“[a-zA-Z]+ ?[0-9]?,? [0-9]{4}”

1. Decompress the file into your working directory:

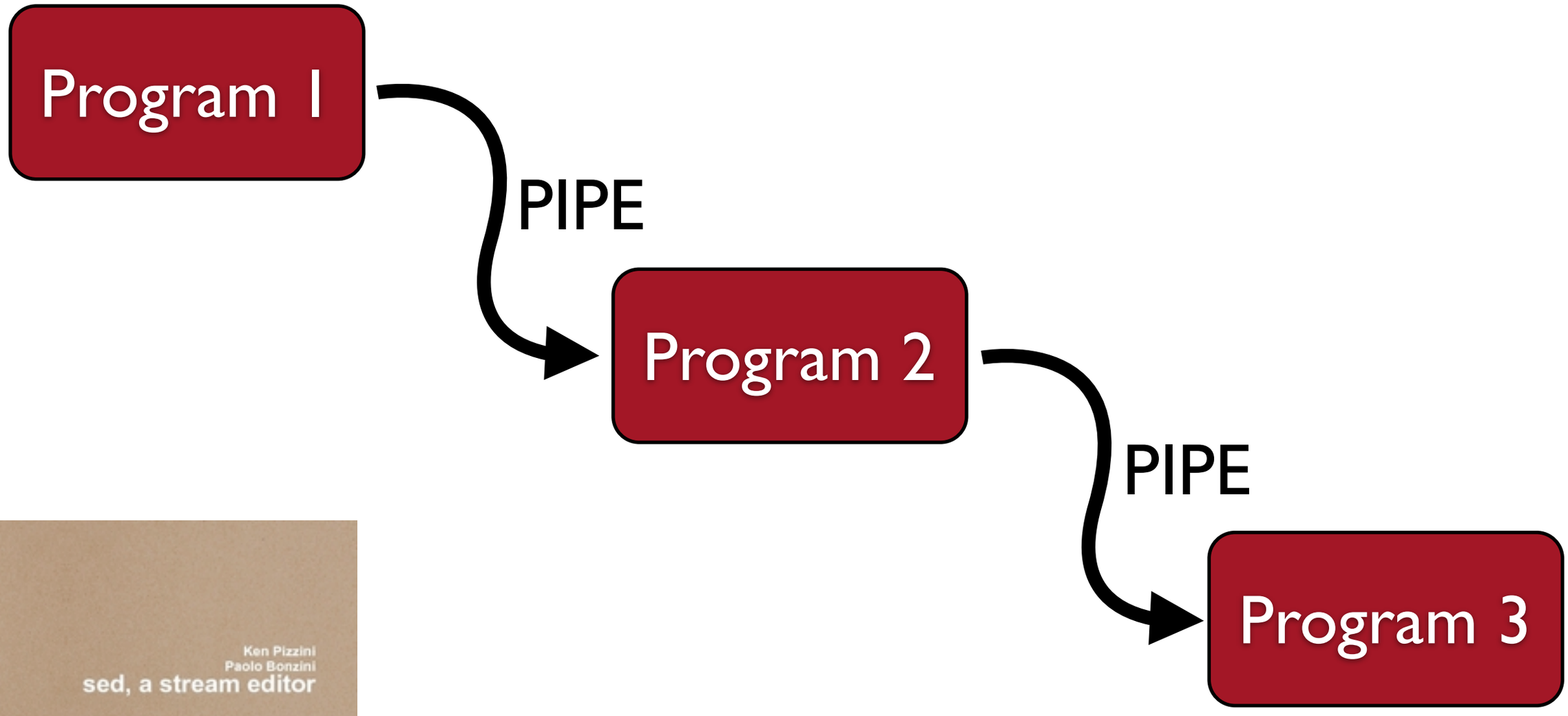
   ```
   ~/workshop_data/unix/record.tsv.gz
   ```

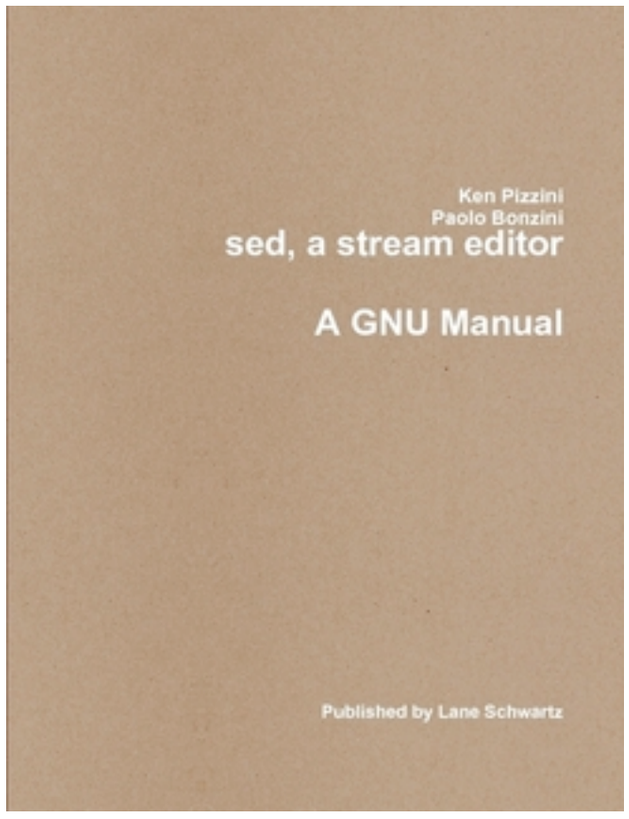
2. Use grep and search the file for the different patterns above:

   ```
   % grep -E “pattern” record.tsv
   ```

3. `cat` the file afterwards to examine what does and doesn’t match with each pattern.

sed, a stream editor





Search and replace on data flowing through the pipe (a stream)

sed, a stream editor, ctd.

`s/pattern/replace/`

Two new encodings:

Record a match: `(pattern)`

Back references: `\1 \2 \3`

```
% sed -E 's/[a-z]+ [a-z]+/foo/'
```

sed, a stream editor, ctd.

s/pattern/replace/

```
% cat record.tsv | sed -E 's/[a-z]+ [a-z]+/foo/'

% cat record.tsv | sed -E 's/([a-z]+) [a-z]+/\1/'
```

```
% cat record.tsv | sed -E 's/[0-9]+//'

% cat record.tsv | sed -E 's/[0-9]+//g'

% cat record.tsv | sed -E 's/^[0-9]+    //'
```

sed, a stream editor, ctd.

`s/pattern/replace/`

Create a complex command:

```
% cd samples
% ls -1
fish_001.tags.tsv
fish_003.tags.tsv
fish_004.tags.tsv
fish_005.tags.tsv

% ls -1 | sed -E 's/^(fish_[0-9]+\.tags)\.tsv/mv \1\.tsv \1\.loc/'
```

fish_310.tags.tsv.gz

`~/workshop_data/unix/fish_310.tags.tsv.gz`

`ls`
`gunzip`
`man`
`more`
`cat`
`wc`
`head`
`cut`
`grep`
`sed`
`tr`
`>`
`|`

1. Decompress the file
2. Extract out the consensus sequences (2,191 sequences)
3. Extract out the ID and sequence for each consensus
4. Construct a sed expression to match the ID and consensus. Use sed to reverse the columns.
5. Use sed/tr to convert the two column input into to a FASTA file.

1. use ctrl-v tab to get a literal tab on the command line
2. The tr command can replace one single character with another. You could use it to replace the "|" character with a new line "\n" where "\n" will create a newline in the output.

sed, a stream editor, ctd.

Selecting specific lines

```
% cat s_1_sequence.txt | sed -n '6p'

% cat s_1_sequence.txt | sed -n '1,10p'

% cat s_1_sequence.txt | sed -n '2~4p'
```

Keep your data files zipped

```
% gzip s_1_sequence.txt
% zcat s_1_sequence.txt.gz | sed -n '2~4p'
```

Determine the count of all barcodes in the file

```
% zcat s_1_sequence.txt.gz | sed -n '2~4p' | cut -c 1-5 | sort -n | uniq -c | sort -n
```

sed, a stream editor, ctd.

Edit the FASTQ headers

`s_1_sequence.txt.gz`

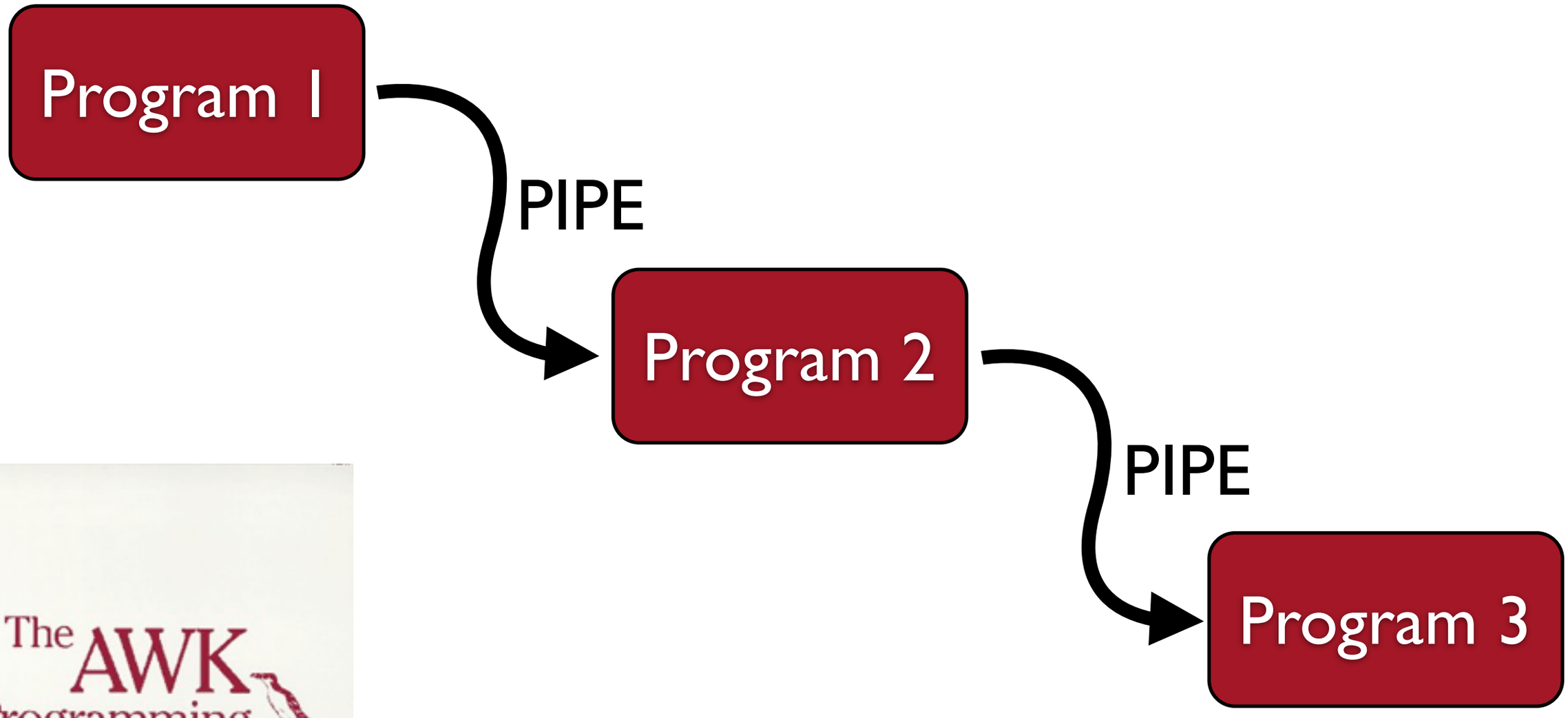
`s/pattern/replace/`

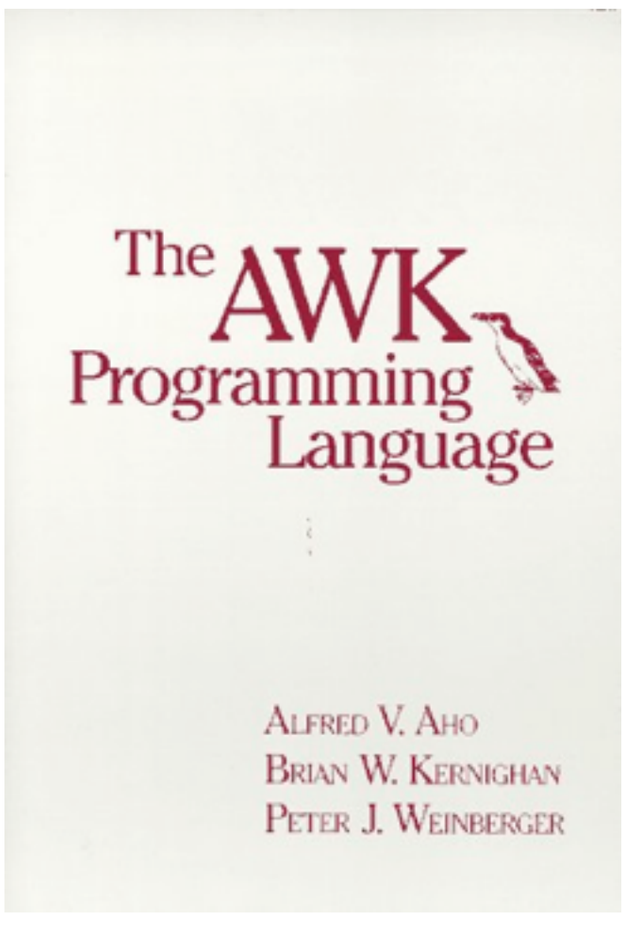
1. Construct a sed expression to match the components of the FASTQ ID:

   ```
   HWI-EAS303_0002:1:1:1072:14491#0/1
   ```

   and remove the first element (`HWI-EAS303_0002:`)
2. Capture the output into a new FASTQ file

awk, a stream programming language





Apply formulas to data flowing through the pipe (a stream)

awk, a stream programming language, ctd.

```
pattern {action}
```

1. Awk is column (*field*) aware:

   `$0` - the whole line
   `$1` - column one
   `$2` - column two
   …

2. `pattern` can be any logical statement:

```
$3 > 0                  - if column 3 is greater than 0
$1 == 32                - if column 1 equals 32
$1 == $3                - if column 1 equals column 3
$1 == "consensus"       - if column 1 contains the string, "consensus"
```

If pattern is true, everything in {…} is executed

awk, a stream programming language, ctd.

```
pattern {action}
```

Apply action to every line

```
{action}
```

Execute action once at start

```
BEGIN {action} pattern {action}
```

Execute action once at end

```
pattern {action} END {action}
```

```
BEGIN {action} pattern {action} END {action}
```

awk, a stream programming language, ctd.

```
pattern {action1; action2; action3}
```

1. Built in variables

```
NR - number of records seen so far (aka line number)
NF - number of fields in the current record
FILENAME - name of the current file being read
```

2. Built in functions

```
length(x) - length of the field
print(x)  - print a field
rand()    - generate a random number
sqrt(x)   - calculate square root of x
sub(x, y) - substitute s for r in $0
```

3. User defined variables

```
increment: n  = n + 1
multiply:  n += $2 * $3
```

awk, a stream programming language, ctd.

Sum a series of numbers in the third column:

```
awk '{sum+=$3} END {print sum}'
```

Sum a series of numbers in the third column larger than 500:

```
awk '$3 > 500 {sum+=$3} END {print sum}'
```

Add line numbers to the output:

```
awk '{print NR, $0}'
```

Print the length of each line:

```
awk '{print length($0)}'
```

Compute the average:

```
awk '{sum+=$3} END {print sum/NR}'
```

`~/workshop_data/unix/contigs.fa.gz`

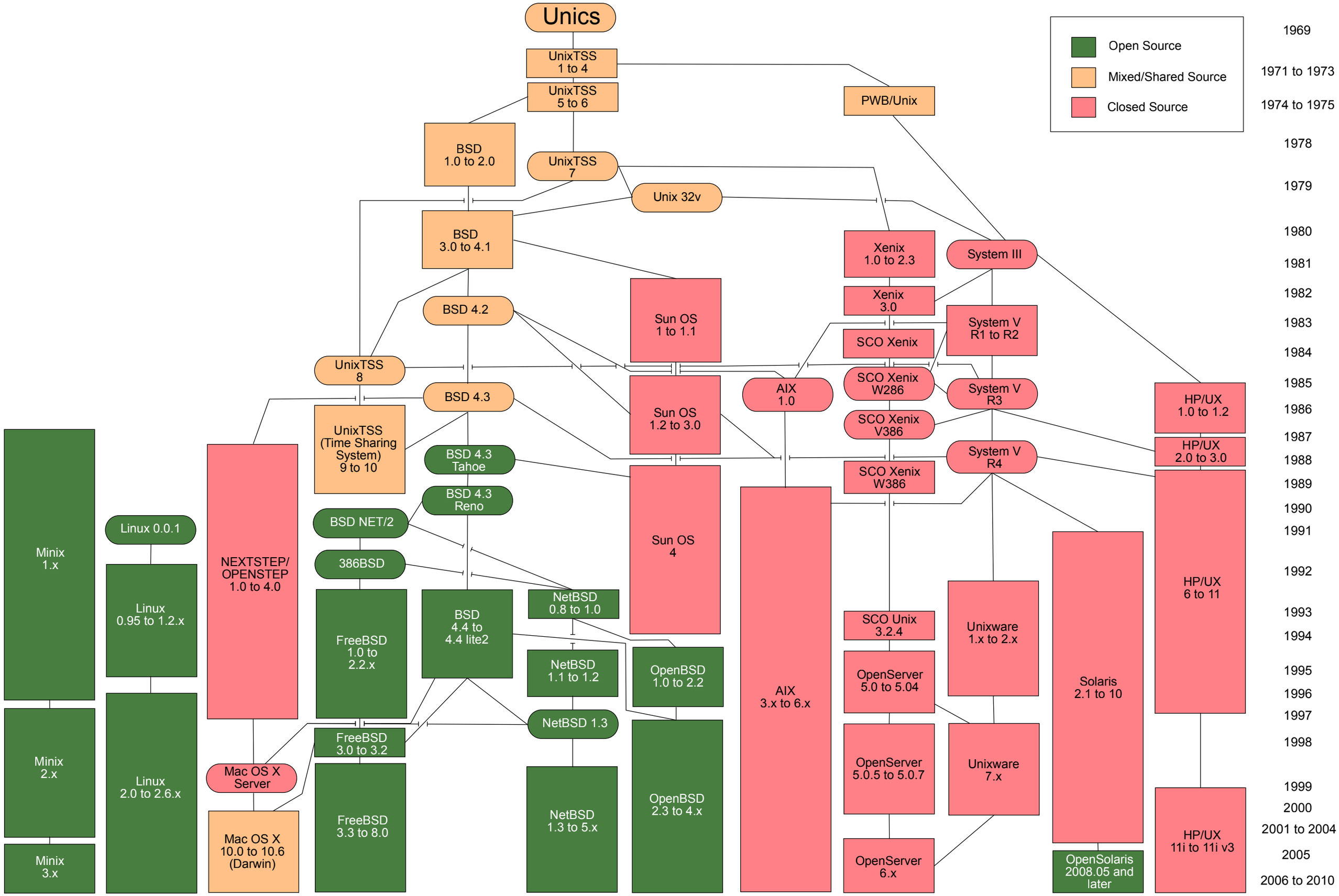
1. Determine the average contig length and the total length of all contigs.
    1. Extract out the length from the Velvet FASTA headers
    2. Sum them and compute the average
2. Determine the average and total length of contigs longer than 500bp.

`~/workshop_data/unix/trimmed_reads.fq.gz`

3. Determine the length distribution of the trimmed reads
    1. Extract out the sequence
    2. Measure its length
    3. Compute the distribution of lengths

```
ls
man
more
cat
head
grep
sed
awk
sort
uniq
  |
```

Unix History



Editing on UNIX

Emacs

Richard Stallman - 1976
Founded GNU Project

Vi

Bill Joy - 1976
BSD/Sun Microsystems

Decompress the example file into your home directory.

`~/workshop_data/unix/manifesto.gz`

```
% emacs <filename>

% emacs /absolute/path/to/file

% emacs ../../../relative/path/to/file

% emacs file1 /path/file2 ../../file3
```

Command mode versus **Text-entry mode**

Your mouse cannot help you! (mostly.)

Emacs commands start with either **ctrl** or **meta** (esc-x)

The **Mark**

1. No mouse, so we need a way to indicate/highlight regions of text.
2. `ctrl-space` sets the mark
3. Use the arrow keys to highlight a region of text
4. Issue a command, e.g. copy/paste, or just press the space bar to unhighlight

Useful Emacs commands

`ctrl-x ctrl-s`	save file
`ctrl-x ctrl-f`	open a new file
`ctrl-space`	set the mark
`esc-w`	copy highlighted text
`ctrl-w`	cut highlighted text
`ctrl-y`	paste text
`ctrl-x u`	undo
`ctrl-x b`	switch to another file (buffer)
`ctrl-s`	search for text
`esc %`	search and replace text
`ctrl-]`	quit current command
`ctrl-v`	page-down
`esc-v`	page-up
`esc-g g`	goto line
`ctrl-x ctrl-c`	quit emacs

manifesto.gz

`~/workshop_data/unix/manifesto.gz`

1. Start emacs, opening `manifesto`
2. Copy the title and paste it one line below.
3. Search for the term '`GNU`', how many instances can you find?
4. Search/replace the phrase '`free software`' with '`proprietary software`'. How many instances did you replace?
5. Now, undo the replacements so that we remain `free`
6. Cut out the first paragraph of text.
7. Open a new file, `manifesto2`, paste the paragraph, save, quit emacs, view the file with `more`

Download the example file using `wget`

1. Visit in your web browser:

`http://creskolab.uoregon.edu/stacks/`

2. Right click on the "Download Stacks" link and select "Copy Link Location" (or a similar variant)

3. Paste the link into the terminal and use `wget` to fetch it. Untar and decompress the archive.

File Permissions, Users+Groups

```
rwxrwxr-x
111111101
```

Owner	Group	Other
rwx	rwx	r-x
111	111	101
7	7	5

```
ubuntu@ip-10-4-193-188:~$ tar xfz stacks-0.998.tar.gz
ubuntu@ip-10-4-193-188:~$ cd stacks-0.998/
ubuntu@ip-10-4-193-188:~/stacks-0.998$ ls -la
total 500
drwxrwxr-x  7 ubuntu ubuntu   4096 2012-01-09 22:24 .
drwxr-xr-x 20 ubuntu ubuntu   4096 2012-03-06 23:20 ..
-rw-rw-r--  1 ubuntu                                :11 aclocal.m4
-rwxr-xr--  1 ubuntu                                :27 autogen.sh
-rw-r--r--  1 ubuntu                                :23 ChangeLog
drwxrwxr-x  2 ubuntu                                :24 config
-rw-rw-r--  1 ubuntu                                :14 config.h.in
-rwxr-xr-x  1 ubuntu                                :11 configure
-rw-r--r--  1 ubuntu                                :51 configure.ac
-rw-r--r--  1 ubuntu                                :27 INSTALL
-rw-r--r--  1 ubuntu                                :27 LICENSE
-rw-r--r--  1 ubuntu                                :11 Makefile.am
-rw-rw-r--  1 ubuntu                                :12 Makefile.in
drwxrwxr-x  3 ubuntu                                :24 php
-rw-r--r--  1 ubuntu                                :56 README
drwxrwxr-x  2 ubuntu                                :24 scripts
drwxrwxr-x  2 ubuntu                                :24 sql
drwxrwxr-x  2 ubuntu                                :24 src
ubuntu@ip-10-4-193-18
```

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

```
chmod 777 README     chmod 644 README     chmod 600 README
```

What is a variable?

What is a variable, ctd.

1. A variable can have almost any name:
   - `foo`
   - `bar`
   - `cats`
   - `line`

2. You assign it a value like this:
   - `foo=32`
   - `bar=27.4938193`
   - `cats="LOLzzzz"`
   - `line="sample_01.fq"`

3. You refer to it using a dollar sign:
   - `$foo`
   - `${foo}`

What is a variable, ctd.

Variables often have types, depending on the language

integer	`1, 2, 3, 4, 5...` `-1, -2, -3, -4, -5...`
float (double)	`3.14159265358979`
string (of characters)	`“My dog is Billy”`

What is a variable, ctd.

1. Try it out: set a variable on the command line:
    - `foo=32`
2. Display the value the variable holds:
    - `echo $foo`
3. Set a new value and display it:
    - `foo="The cat likes thai food"`
    - `echo $foo`

Shell Loops

My favorite command: `ls -1`

The *while* loop

while read line; do **`command $line`**; done

Pipe `ls -1` to a `while` loop and watch magic happen

```
fish_001.tags.tsv
fish_003.tags.tsv
fish_004.tags.tsv
fish_005.tags.tsv
```

`ls -1`

PIPE

while read line; do command $line; done

```
command: grep -c $line
$line =
```

```
fish_003.tags.tsv
fish_004.tags.tsv
fish_005.tags.tsv
```

`ls -1`

PIPE

while read line; do command $line; done

```
command: grep -c “ACGT” $line
$line = fish_001.tags.tsv

grep -c “ACGT” fish_001.tags.tsv
```

```
ls -1
```

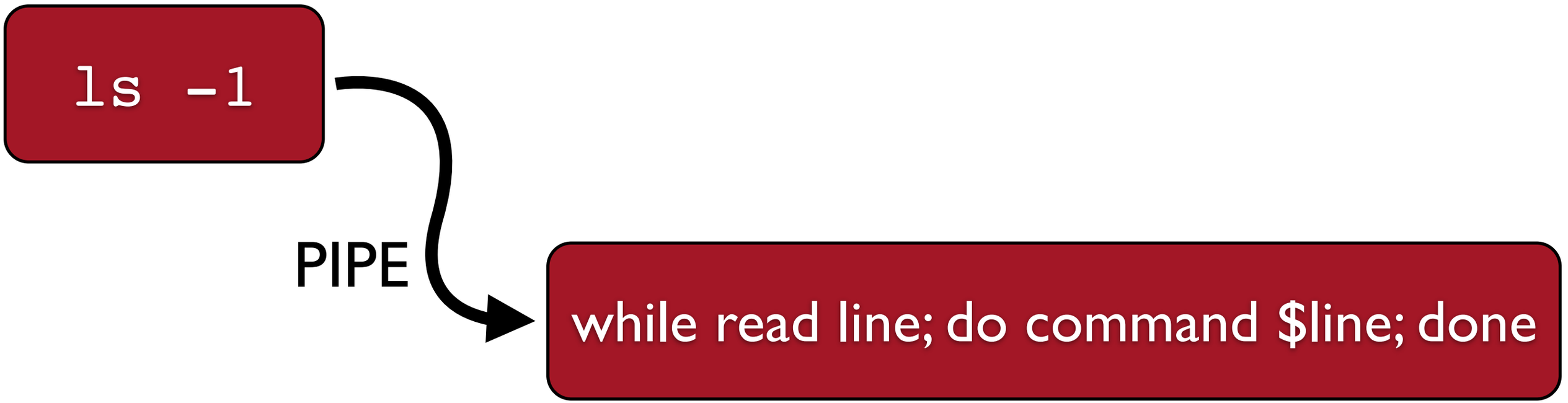
```
fish_004.tags.tsv
fish_005.tags.tsv
```

PIPE

while read line; do command $line; done

```
command: grep -c “ACGT” $line
$line = fish_003.tags.tsv

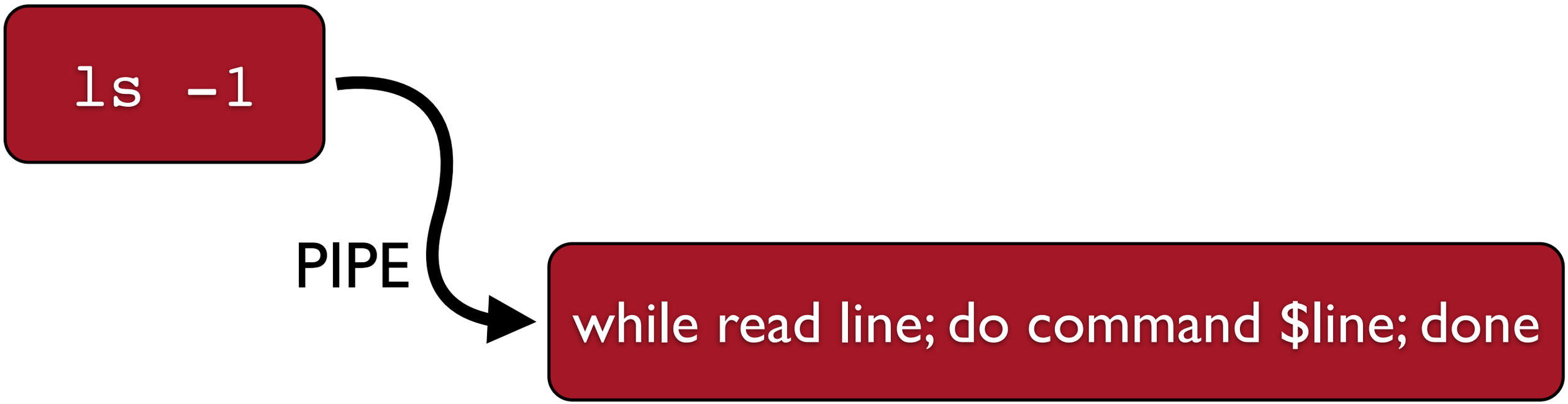
grep -c “ACGT” fish_003.tags.tsv
```




```
command: grep -c “ACGT” $line
$line = fish_005.tags.tsv

grep -c “ACGT” fish_005.tags.tsv
```

1. “line” can be any variable name: foo, bar, etc.
2. multiple commands can be put together with a semicolon.




```
command: echo -n “$line   “; grep -c “ACGT” $line
$line = fish_005.tags.tsv

echo -n “fish_005.tags.tsv    “; grep -c “ACGT” fish_005.tags.tsv
```

1. “line” can be any variable name: foo, bar, etc.
2. multiple commands can be put together with a semicolon.

samples.tar.gz

1. Expand the archive: **`tar -xvf`**

```
fish_001.tags.tsv
fish_003.tags.tsv
fish_004.tags.tsv
fish_005.tags.tsv
```

2. Move into the samples directory
3. Execute a command that can identify the consensus sequences in this file.
4. Try out the `ls -1` command
5. Combine parts 3 and 4 with a while loop to count the number of consensus sequences in each file

```
4
5
4
6
```

6. Modify the command to prefix each output with the file name.

```
fish_001.tags.tsv     4
fish_003.tags.tsv     5
fish_004.tags.tsv     4
fish_005.tags.tsv     6
```

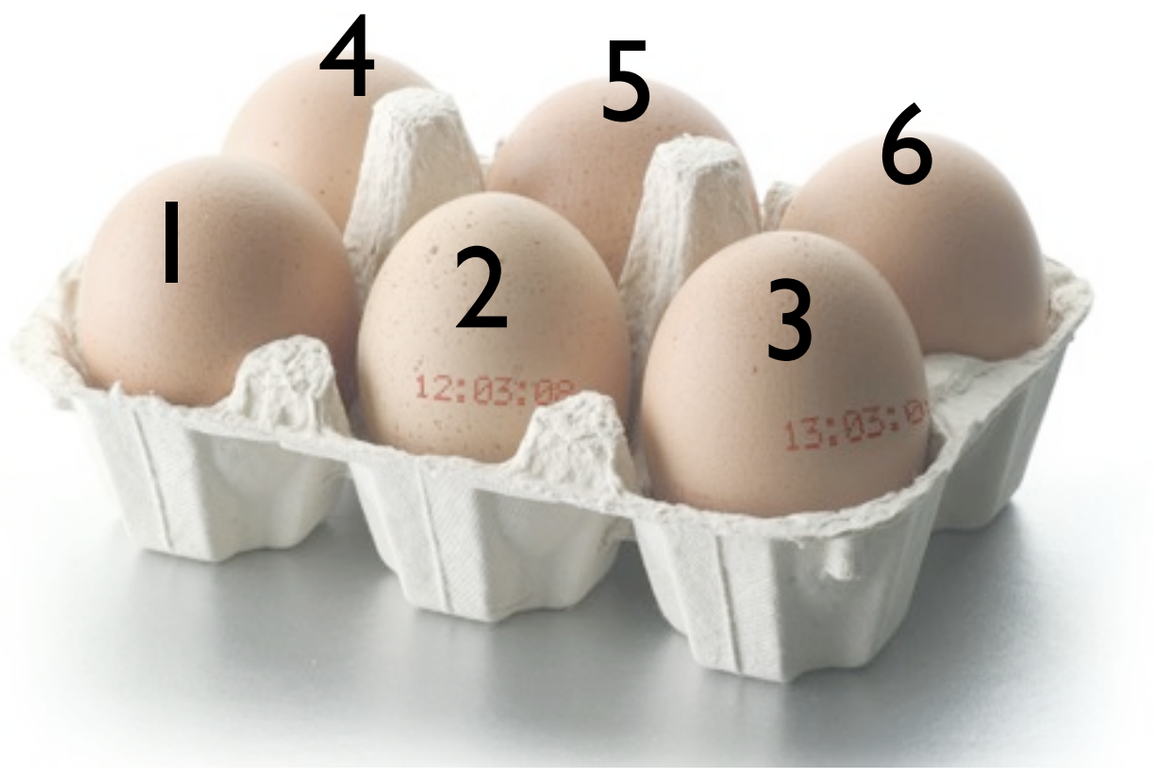
while read line; do **`command $line`**; done

Shell Scripts

1. Anything you can do on the shell can be placed in a shell script
2. Shell scripts often end in the suffix ".sh"
3. Shell scripts must be executable (`chmod 755`)
4. *Comments* can be written in scripts with a "#"
5. Variables can be used to shorten long paths
6. Shell loops can be used to process lots of files
7. "\" can be used to wrap long commands across multiple lines
8. #!/bin/bash must be the first line - specifies interpreter

Shell Scripts, ctd.

What is an array?



Shell Scripts, ctd.

What is an array?

1. A variable can have almost any name:
    - `foo`
    - `bar`
    - `cats`
    - `line`

2. You assign it a value like this:

    - ```
      foo="progeny_10.fa
      progeny_11.fa
      progeny_12.fa
      progeny_13.fa
      progeny_14.fa
      progeny_15.fa"
      ```
      ```

Shell Scripts, ctd.

```
[catchen@genome]:~/research/seq/dre_hap% more build_tags.sh
#!/bin/bash

ROOT=$HOME/research/seq
bin=$ROOT/radtags/stacks/trunk/
src=$ROOT/dre_hap
db=dre_hap_radtags
batch_id=1
date=2011-05-17
desc="Danio rerio haploid map"

cd $bin
nice -n 19 $bin/scripts/denovo_map.pl -m 3 -B $db -b $batch_id -t -a $date -D "$desc" -e $bin -T 40 \
  -o $src/nstacks \
  -p $src/samples/female.fq \
  -r $src/samples/progeny_01.fq \
  -r $src/samples/progeny_02.fq \
  -r $src/samples/progeny_03.fq \
  -r $src/samples/progeny_05.fq \
  -r $src/samples/progeny_06.fq \
  -r $src/samples/progeny_08.fq \
  -r $src/samples/progeny_09.fq \
  -r $src/samples/progeny_10.fq \
  -r $src/samples/progeny_13.fq \
  -r $src/samples/progeny_14.fq \
  -r $src/samples/progeny_16.fq \
  -r $src/samples/progeny_17.fq \
  -r $src/samples/progeny_18.fq \
  -r $src/samples/progeny_19.fq \
  -r $src/samples/progeny_20.fq \
  -r $src/samples/progeny_23.fq \
  -r $src/samples/progeny_24.fq \
  -r $src/samples/progeny_25.fq \
  -r $src/samples/progeny_27.fq \
  -r $src/samples/progeny_33.fq \
  -r $src/samples/progeny_34.fq \
  -r $src/samples/progeny_35.fq \
  -r $src/samples/progeny_36.fq \
  -r $src/samples/progeny_37.fq \
  -r $src/samples/progeny_38.fq
```

Shell Scripts, ctd.

catchen@genome.uoregon.edu:/home/catchen — ssh — 178×47

```
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash

ROOT=$HOME
src=$ROOT/research/seq/or_phylo
bwa_db=$ROOT/research/bwa/gac_gen_broads1_e64
bowtie_db=$ROOT/research/bowtie/gac_gen_broads1_e64
bin=$ROOT/research/stacks/trunk

files="stl_1274.31
stl_1274.32
stl_1274.33
stl_1274.34
stl_1274.35
stl_1274.36
stl_1274.37
stl_1274.38
stl_1274.39
"

#
# Align with GSnap
#
for file in $files
do
   echo $file >> aligned/gsnap.log
   gsnap -t 24 -n 1 --quiet-if-excessive --terminal-threshold=10  -A sam -m 5 -i 2 -d gac_gen_broads1_e64 \
         -D /home/catchen/research/gsnap/gac_gen_broads1_e64 $src/samples/${file}.fq > $src/aligned/${file}.sam 2>> $src/aligned/gsnap.log
done

-UU-:----F1  build_tags.sh   All L29    (Shell-script[bash])----------
Wrote /home/catchen/build_tags.sh
```

Shell Scripts, ctd.

`~/workshop_data/unix/seqs.tar.gz`

1. Move into the seqs directory
2. Use `ls -1` combined with a shell loop to count the number of lines in each file
3. Use a shell loop to concatenate the files into a single file, use the >> operator to redirect the output
4. Count the lines in the final file, make sure they match the sum of the lines in the individual files.
5. Create a shell script to do all of the above.

Advanced Bash-Scripting Guide: http://tldp.org/LDP/abs/html/

Shell Scripts, ctd.

The *for* loop

```
files="stl_1274.31
stl_1274.32
stl_1274.33
stl_1274.34
stl_1274.35
stl_1274.36”

for file in $files
do
    commands
done
```

Shell Scripts, ctd.

`~/workshop_data/unix/fst.tar.gz`

Your goal is to prepare these F_{ST} data files for plotting. We need to separate the data by chromosome.

1. Expand the archive and move into the fst directory.
2. Use `ls -1` to determine a list of files to examine.
3. Use `cut/grep/sort/uniq` to determine a list of chromosomes to process, ignoring scaffolds.
4. Work out the command to extract the chromosomes from one F_{ST} file and redirect them into their own file.
5. Write a shell script that uses two shell loops, one nested inside the other, to separate each F_{ST} file into separate files, one for each chromosome
6. Add variables to make paths absolute.