

Our toolbox

Recap

Decent summaries of Bayesian vs frequentist (likelihood / 'maximum likelihood') estimation:

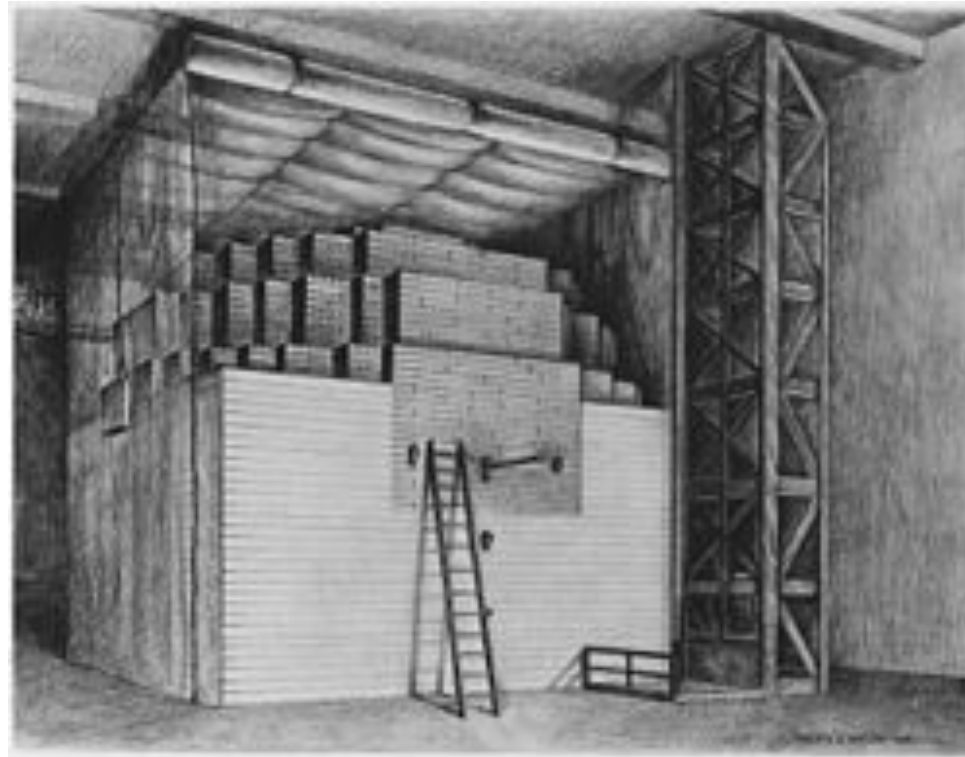
<https://www.youtube.com/watch?v=0LQmZXCWMI>

<https://www.psychologicalscience.org/observer/bayes-for-beginners-probability-and-likelihood>

The tools

- Having great tools and using them is half the battle
- Don't be afraid to ask for help, or offer it
- **Software Carpentry & Data Carpentry**
- Keep attending industry / academic meetups, seminars and hackathons
- Become multilingual but don't lose old skills
- **Be organised.**

Fermi estimation



- Order-of-magnitude (10^n) calculations can get you a long way!
- Also good for wrapping your mind around problems, which in turn will help you **design experiments** and **sanity-check results**

Big power

- Effect size = $\mu - \mu_0$ (difference from null)
- Many of the effect sizes we might want to detect are *incredibly* small.
- So we need **big** samples:

Effect size ($\mu - \mu_0$):	+1.0	+0.1	+0.01
Detect with ANOVA at:	Expected N:	N	N
$\alpha = 0.05, \beta = 0.5$	17	1,570	156,978
$\alpha = 0.01, \beta = 0.5$	25	2,336	233,580
$\alpha = 0.001, \beta = 0.5$	37	3,415	341,499

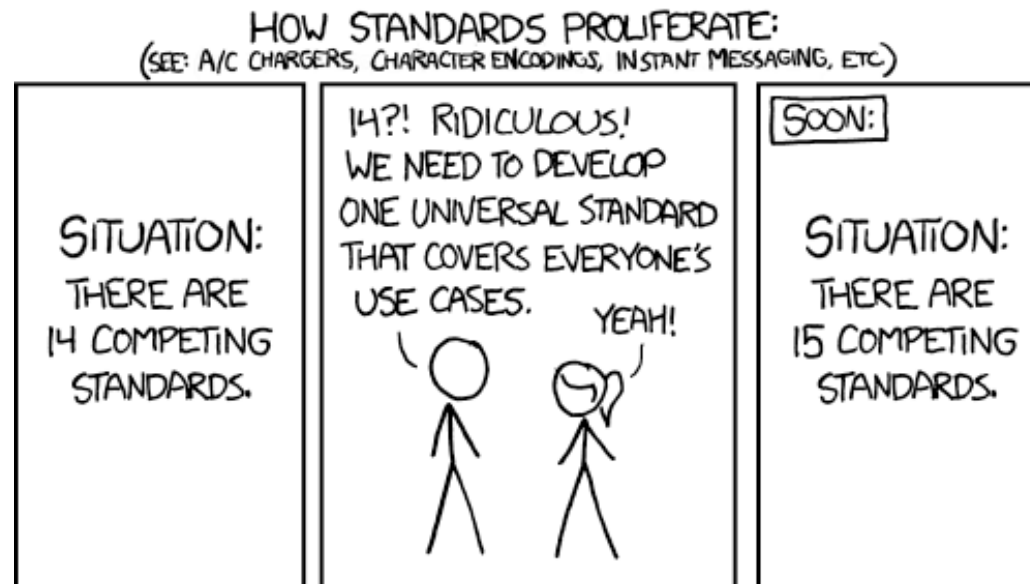
See e.g.

<http://www.sample-size.net/sample-size-means/>

<http://www.bristol.ac.uk/caite/geocode/newcastleshortcourse/powerandsamplesize.pdf>

Formats

- Get used to dealing with *zillions* of formats, and converting between them
- D-R-Y still applies, so look for libraries to help you first – but always check the output(!)
- When writing research data yourself, consider the person coming after you...



INTRODUCING THE XKCD STACK

EBNF/CSS

BROKEN JAVA APPLET

ARCHIVE.ORG MIRROR

HYPERCARD.JS

QBASIC ON RAILS

[BLOCKED BY ADBLOCKER]

MONGODB/EXCEL

SOME PIECE THAT WORKS SO
NOBODY ASKS ANY QUESTIONS

TRIPLY-NESTED DOCKER

PARAVIRTUAL BOY®

A DEV TYPING REAL FAST

OLDER VERSION
OF OUR SOFTWARE

MYSTERY NETWORKING HORROR

MICROSOFT BOB SERVER®

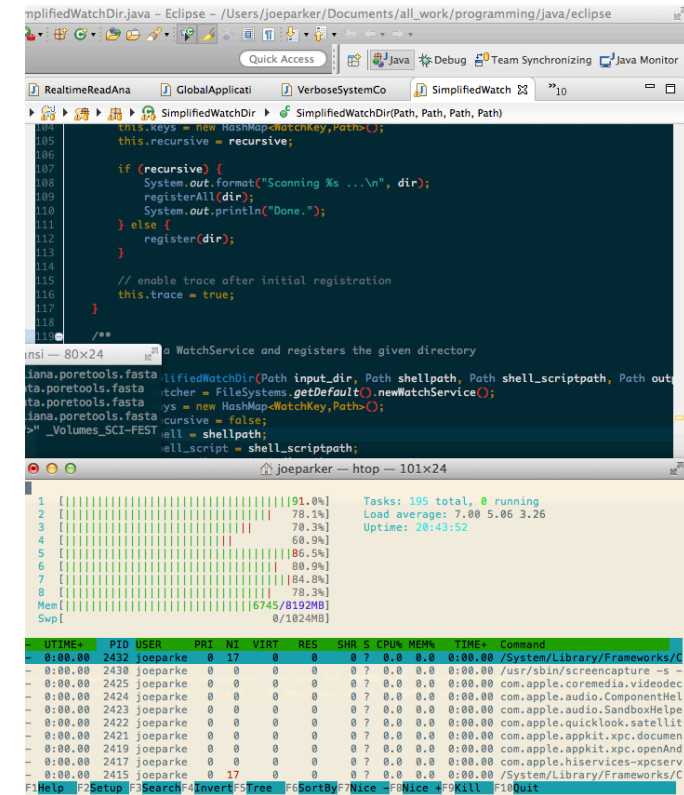
A GIANT CPU SOMEONE
BUILT IN MINECRAFT

Notes, comments and metadata

- Record *everything* if you **ever** want to repeat it.
- Casual notebooks (Evernote, Notes etc) all useful
- Metadata, parameters and versions for analyses
- Good lab-book **as important** as wet science
- Ever more tools appearing to help track analyses (iPython, R notebooks, etc)
- Code needs to be well commented, well named, and well designed
- (~~GOD~~ objects)

IDEs

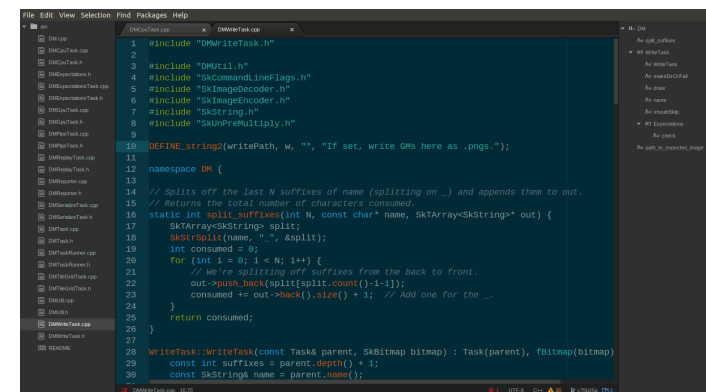
- Integrated development environment – anything that makes it easier to organise, import, write, test, build and publish code.
- Full IDEs:
 - Eclipse (mainly Java)
 - Xcode (iOS, C)
 - Rstudio
- High-powered text editors:
 - Atom
 - TextWrangler



```
104 this.keys = new HashMap<WatchKey, Path>();
105 this.recursive = recursive;
106
107 if (recursive) {
108     System.out.format("Scanning %s ...\\n", dir);
109     registerAll(dir);
110     System.out.println("Done.");
111 } else {
112     register(dir);
113 }
114
115 // enable trace after initial registration
116 this.trace = true;
117
118 /**
119  * Register a WatchService and registers the given directory
120  */
121 void register(Path input_dir, Path shellpath, Path shell_scriptpath, Path out
122     ta, poretools.fasta
123     itcher = FileSystems.getDefault().newWatchService();
124     ta, poretools.fasta
125     ys = new HashMap<WatchKey, Path>();
126     "Volumes_SCI-FEST
127     all = shellpath;
128     ell_script = shell_scriptpath;
```




















```
1  [|||||] 91.0%
2  [|||||] 70.1%
3  [|||||] 70.3%
4  [|||||] 60.9%
5  [|||||] 86.5%
6  [|||||] 80.9%
7  [|||||] 84.8%
8  [|||||] 70.3%
Mem [|||||] 6745/8192MB
Swp [|||||] 0/1024MB
```

UTIME+	PID	USER	PR	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
0:00.00	2432	joeparke	0	17	0	0	0	7	0.0	0.0	0:00.00	/System/Library/Frameworks/C
0:00.00	2430	joeparke	0	0	0	0	0	7	0.0	0.0	0:00.00	/usr/sbin/screencapture -s -
0:00.00	2425	joeparke	0	0	0	0	0	7	0.0	0.0	0:00.00	com.apple.coremedia.videodec
0:00.00	2424	joeparke	0	0	0	0	0	7	0.0	0.0	0:00.00	com.apple.audio.ComponentHel
0:00.00	2423	joeparke	0	0	0	0	0	7	0.0	0.0	0:00.00	com.apple.audio.SandboxHelpe
0:00.00	2422	joeparke	0	0	0	0	0	7	0.0	0.0	0:00.00	com.apple.quicklook.satellit
0:00.00	2421	joeparke	0	0	0	0	0	7	0.0	0.0	0:00.00	com.apple.appkit.xpc.documen
0:00.00	2419	joeparke	0	0	0	0	0	7	0.0	0.0	0:00.00	com.apple.appkit.xpc.openAnd
0:00.00	2417	joeparke	0	0	0	0	0	7	0.0	0.0	0:00.00	com.apple.hiservices-xpcserv
0:00.00	2415	joeparke	0	17	0	0	0	7	0.0	0.0	0:00.00	/System/Library/Frameworks/C



```
1 #include "DMNriteTask.h"
2
3 #include "DMUtil.h"
4 #include "SKCommandLineFlags.h"
5 #include "SKImageDecoder.h"
6 #include "SKImageEncoder.h"
7 #include "SKString.h"
8 #include "SKImageMultiply.h"
9
10 DEFINE_string(writePath, w, "", "If set, write GMS here as .pngs.");
11
12 namespace DM {
13
14 // Splits off the last N suffixes of name (splitting on _) and appends them to out.
15 // Returns the total number of characters consumed.
16 static int split_suffixes(int N, const char* name, SKArray<SKString*> out) {
17     SKArray<SKString*> split;
18     SKStrsplit(name, "_", &split);
19     int consumed = 0;
20     for (int i = 0; i < N; i++) {
21         // We're splitting off suffixes from the back to front.
22         out->push_back(split[split.count()-1-i]);
23         consumed += out->back().size() + 1; // Add one for the _
24     }
25     return consumed;
26 }
27
28 WriteTask(writeTask(const Task& parent, SKBitmap bitmap) : Task(parent), fbitmap(bitmap)
29     const int suffixes = parent.depth() + 1;
30     const SKString& name = parent.name();
```

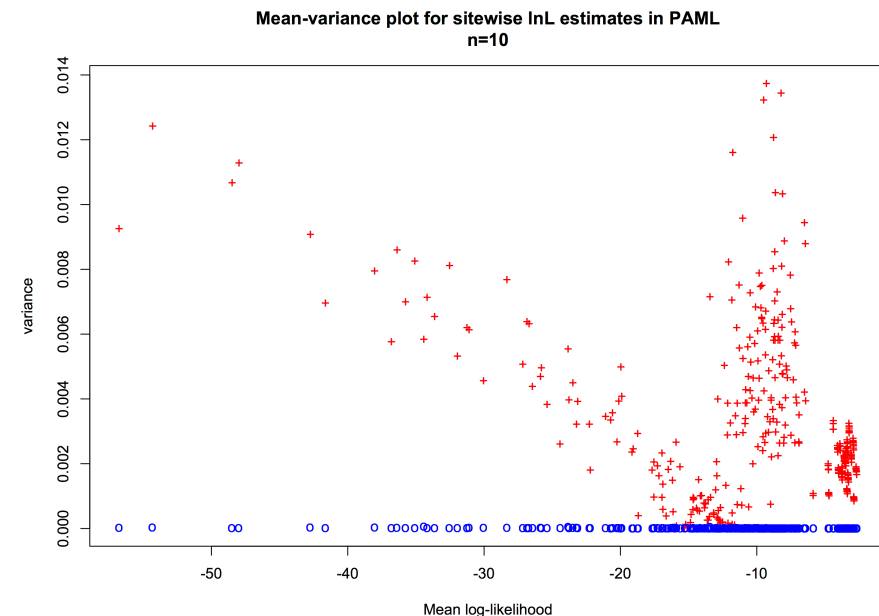
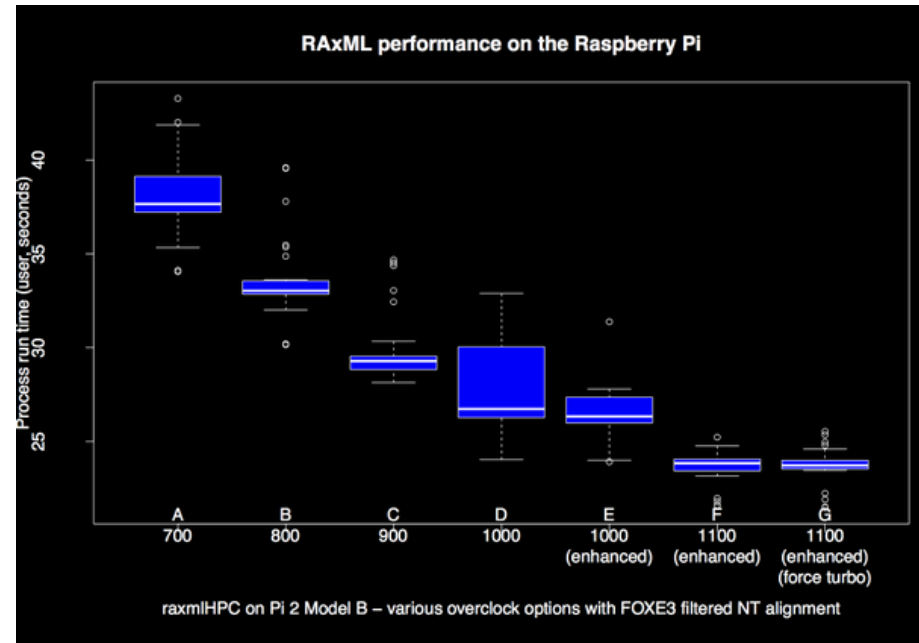

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE
EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	 4 WEEKS	 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	 8 WEEKS	 DAYS	 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	 4 WEEKS	 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES		6 MONTHS	 5 WEEKS	 DAYS	 DAY	2 HOURS
	1 HOUR		10 MONTHS	2 MONTHS	 DAYS	 DAYS	5 HOURS
	6 HOURS				2 MONTHS	 2 WEEKS	 DAY
	 DAY					 8 WEEKS	 DAYS

Code optimisation...

Benchmarking

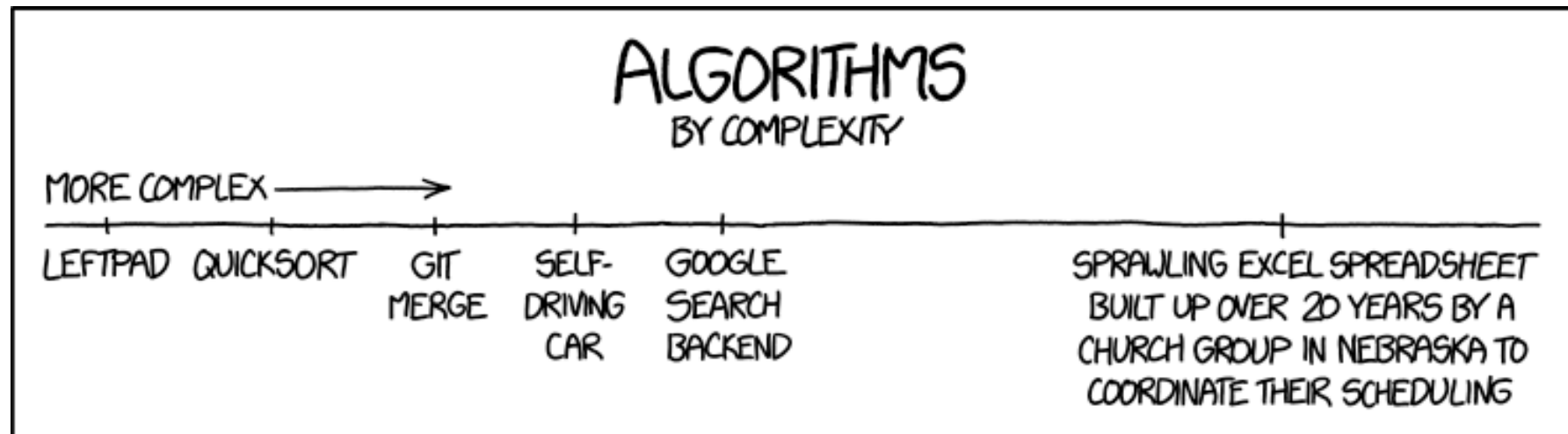
- Analyse execution times and system usage:
 - Which steps are taking the most time/RAM?
 - Has the performance improved? Worsened?
 - **HOW MUCH MORE \$£ € do we need...**
- Measurements aren't precise, just so *happens* to be a... statistical problem(!)



Algorithm complexity

- We have a model with **N** parameters:
How long will it take? And if $N+1$? $N+m$?
- **Algorithmic complexity** attempts to analyse the likely best- worst- and average-case execution time as a function of problem size
- Simple for simple algorithms, e.g.
- Less so for complex ones
- An apparently slower method may have better complexity implications than faster, more complex one.

Method	Best	Worst
Quick sort	$O(n \log n)$	$O(n^2)$
Bubble sort	$O(n)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$



Parallelisation

- “for (n in 1:lots)”
- Parallelisation is an obvious way to speed up / power up analyses
- Not all parallelisations are the same
- Main types:
 - Array jobs
 - Highly parallel/distributed jobs (MPI)

Array jobs

- We have multiple replicates / subsets
- Each is effectively independent – but we have a **lot** of them to get through
- Analyse separately, together, or sequentially
- Synchronisation irrelevant
- “I *could* run each of these on my personal machine, but it takes ages”
- e.g. BLAST metagenomes; parameter sweeps
- Optimisation is easy; optimise each replicate and scale

Highly parallel / MPI jobs

- A very large dataset, but our **inference depends on all of the information**
- We can/need to split the algorithm across several / 000s of cores
- Working simultaneously – **synchronisation** issues relevant
- MPI (message passing interface) / OpenMPI
- Latency (network speeds) between node and disk / each other
- e.g. particle simulations; interactome modelling
- Optimisation is **much harder**

HPC vs HTC

- HPC: *High performance computing*
- HTC: *High throughput computing*

```

1  [|||||] 17.5% 45 [|||||] 22.5% 89 [ 0.0%] 133 [|||] 6.3%
2  [|||||] 14.8% 46 [|||||] 51.6% 90 [ 0.0%] 134 [|||||] 8.9%
3  [|||||] 8.7% 47 [|||||] 19.4% 91 [ 0.0%] 135 [|||||] 13.0%
4  [|||||] 13.0% 48 [|||||] 17.8% 92 [ 0.0%] 136 [|||||] 13.2%
5  [|||||] 4.0% 49 [|||||] 12.9% 93 [ 0.0%] 137 [|||||] 7.8%
6  [|||||] 5.4% 50 [|||||] 21.2% 94 [ 0.0%] 138 [|||||] 2.6%
7  [|||||] 9.1% 51 [|||||] 18.2% 95 [|||||] 2.6% 139 [|||||] 12.4%
8  [|||||] 10.8% 52 [|||||] 20.0% 96 [|||||] 5.8% 140 [|||||] 13.9%
9  [|||||] 8.5% 53 [|||||] 25.8% 97 [|||||] 3.6% 141 [|||||] 8.3%
10 [|||||] 5.3% 54 [|||||] 8.9% 98 [|||||] 6.8% 142 [|||||] 12.0%
11 [|||||] 7.8% 55 [|||||] 22.4% 99 [|||||] 17.3% 143 [|||||] 23.5%
12 [|||||] 5.3% 56 [|||||] 12.4% 100 [|||||] 8.4% 144 [|||||] 29.0%
13 [|||||] 3.7% 57 [|||||] 1.5% 101 [|||||] 9.2% 145 [|||||] 1.0%
14 [|||||] 0.5% 58 [|||||] 15.9% 102 [|||||] 26.0% 146 [|||||] 1.0%
15 [|||||] 7.6% 59 [|||||] 9.0% 103 [|||||] 10.4% 147 [|||||] 20.1%
16 [|||||] 4.2% 60 [|||||] 22.4% 104 [|||||] 7.3% 148 [|||||] 9.9%
17 [|||||] 2.1% 61 [|||||] 15.9% 105 [|||||] 9.8% 149 [|||||] 12.3%
18 [|||||] 0.5% 62 [|||||] 25.9% 106 [|||||] 14.3% 150 [|||||] 17.8%
19 [|||||] 2.1% 63 [|||||] 10.6% 107 [|||||] 4.1% 151 [|||||] 7.8%
20 [|||||] 1.0% 64 [|||||] 3.6% 108 [|||||] 8.0% 152 [|||||] 13.6%
21 [|||||] 3.1% 65 [|||||] 7.8% 109 [|||||] 10.3% 153 [|||||] 12.9%
22 [|||||] 0.0% 66 [|||||] 59.4% 110 [|||||] 13.2% 154 [|||||] 16.7%
23 [|||||] 0.0% 67 [|||||] 15.4% 111 [ 0.0%] 155 [ 0.0%] 0.0%
24 [|||||] 0.0% 68 [|||||] 18.4% 112 [ 0.0%] 156 [ 0.0%] 0.5%
25 [|||||] 0.0% 69 [|||||] 20.7% 113 [ 0.0%] 157 [ 0.0%] 1.0%
26 [|||||] 0.0% 70 [|||||] 17.2% 114 [ 0.0%] 158 [ 0.0%] 0.5%
27 [|||||] 0.0% 71 [|||||] 17.6% 115 [ 0.0%] 159 [ 0.0%] 0.5%
28 [|||||] 0.0% 72 [|||||] 24.0% 116 [ 0.0%] 160 [|||||] 3.1%
29 [|||||] 0.0% 73 [|||||] 20.7% 117 [ 0.0%] 161 [|||||] 1.0%
30 [|||||] 0.0% 74 [|||||] 25.4% 118 [ 0.0%] 162 [|||||] 0.0%
31 [|||||] 0.0% 75 [|||||] 21.9% 119 [ 0.0%] 163 [|||||] 1.0%
32 [|||||] 100.0% 76 [|||||] 15.5% 120 [ 0.0%] 164 [|||||] 1.0%
33 [|||||] 0.5% 77 [|||||] 17.4% 121 [ 0.0%] 165 [|||||] 7.8%
34 [|||||] 0.0% 78 [|||||] 15.8% 122 [ 0.0%] 166 [|||||] 12.4%
35 [|||||] 0.0% 79 [|||||] 13.4% 123 [ 0.0%] 167 [|||||] 5.2%
36 [|||||] 0.0% 80 [|||||] 9.1% 124 [ 0.0%] 168 [|||||] 6.2%
37 [|||||] 0.5% 81 [|||||] 8.6% 125 [ 0.0%] 169 [|||||] 7.3%
38 [|||||] 48.5% 82 [|||||] 6.8% 126 [ 0.0%] 170 [|||||] 4.2%
39 [|||||] 0.0% 83 [|||||] 5.3% 127 [ 0.0%] 171 [|||||] 6.7%
40 [|||||] 0.5% 84 [|||||] 2.1% 128 [ 0.5%] 172 [|||||] 8.9%
41 [|||||] 0.0% 85 [|||||] 3.2% 129 [ 1.0%] 173 [|||||] 5.8%
42 [|||||] 0.0% 86 [|||||] 5.9% 130 [ 0.0%] 174 [|||||] 7.3%
43 [|||||] 0.0% 87 [|||||] 8.4% 131 [ 0.0%] 175 [|||||] 9.3%
44 [|||||] 0.0% 88 [|||||] 0.0% 132 [ 0.0%] 176 [|||||] 8.8%

Mem [|||||] 81.9G/1.48T Tasks: 121, 679 thr; 19 running
Swp [|||||] 48.8M/12.8G Load average: 12.71 10.78 10.60
Uptime: 72 days, 08:03:29

```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
128191	jp93kg	20	0	15.8G	2112M	1610M	S	743.0	0.1	54h00:12	blastn -db /data/blast/db/nt -query everything.fasta -evalue 1 -num_threads 80 -num
85158	root	30	10	23.9G	17.2G	3780	R	84.9	1.1	36h08:35	/opt/CLCGenomicsServer/webapps/CLCServer/WEB-INF/exec/linux-64/clc_assembler_ilo -w
127333	ah43kg	20	0	21.4G	19.3G	4716	R	85.9	1.3	5h07:26	gstacks -B all.bam -O ./ --paired -t 24 -m 2
120683	jp93kg	20	0	28276	6280	3232	R	6.3	0.0	26:41.92	htop
37975	jp93kg	20	0	28212	6144	3208	R	6.3	0.0	0:07.58	htop

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

HPC vs HTC

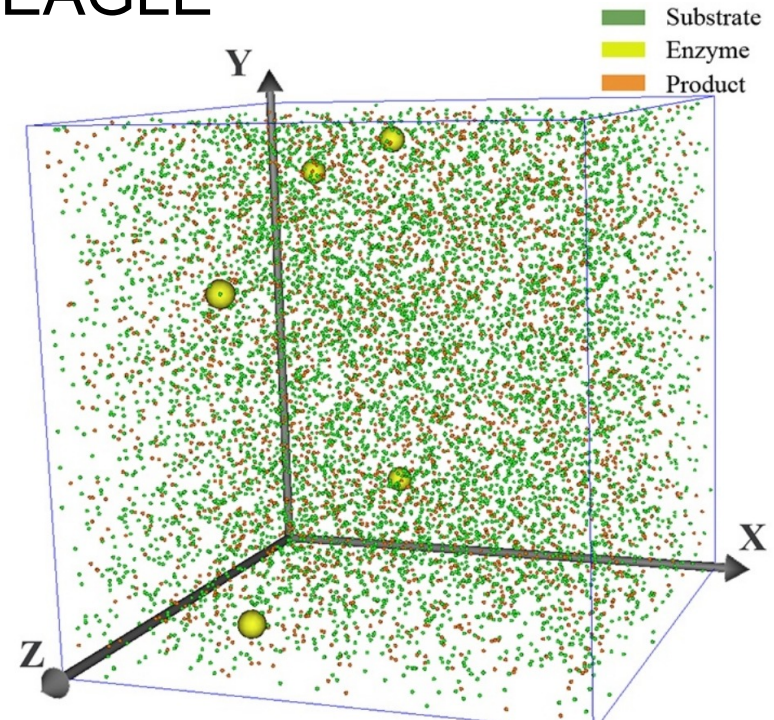
- HPC: High **performance** computing:
 - Fast cores and interconnects for MPI
 - **Lots** of RAM
 - Problems with significant memory footprint
 - e.g. *de novo* genome assemblies, protein folding
 - Each job a significant share of total capacity

HPC vs HTC

- HTC: High **throughput** computing:
 - Lots and lots (>000s) of cores
 - RAM per core more limited
 - Highly parallel / long job time / speed unimportant
 - e.g. public BLAST server, gene annotation, phylogenomics
 - Very large number of cores means capacity for multiple tasks at once

GPUs and MCMC/agent-based modelling

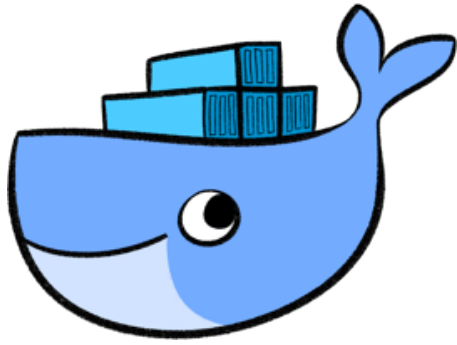
- Certain types of computation require so many parallel but low-level calculations they benefit from quite different architecture
- This can be run effectively on GPUs with some optimisation and libraries e.g. BEAGLE
- e.g. swarming algorithms; agent-based simulation; coalescent phylogenomics; basecalling / HMMs



Cloud

- It doesn't really matter where our machines are
- At the extreme end we can use **cloud computing**
- Rented, temporary time on remote computer
- Private (Amazon AWS; Google) and public (ResearchCloud) providers
- Usage normally paid per-unit time
- Can improve portability
- Costs need to be controlled carefully
- Still has a CO₂ footprint(!)

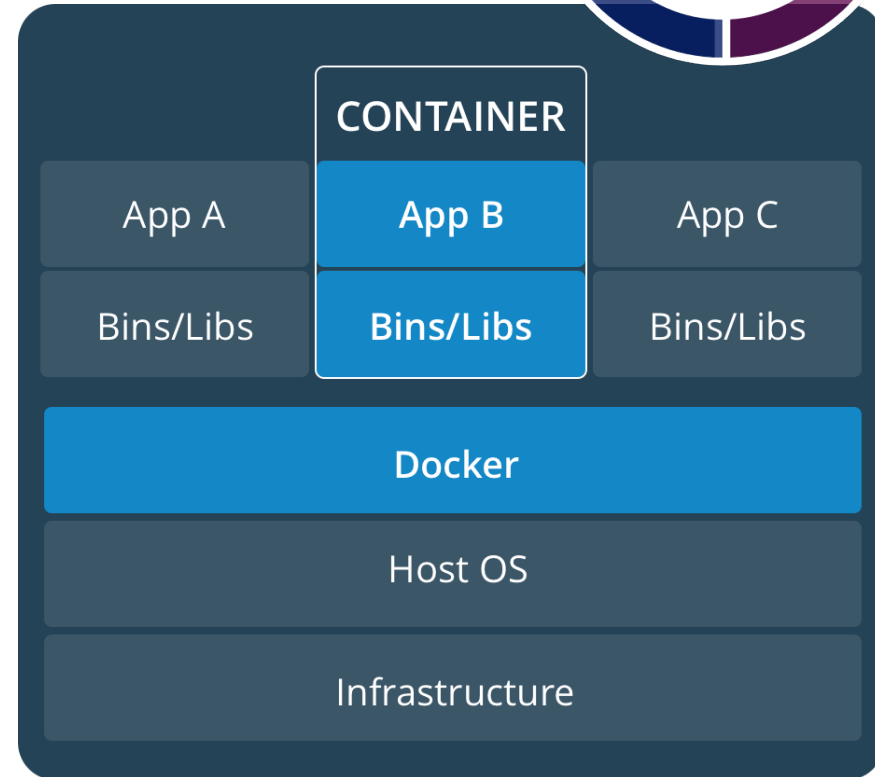




Docker and VMs



- **VMs** (virtual machines) provide a means to ensure the computing environment is replicable.
- Also enhances portability
- VM + something to run on it = **container**
- Docker and Singularity are just systems for creating, sharing and running containers



Docker and VMs

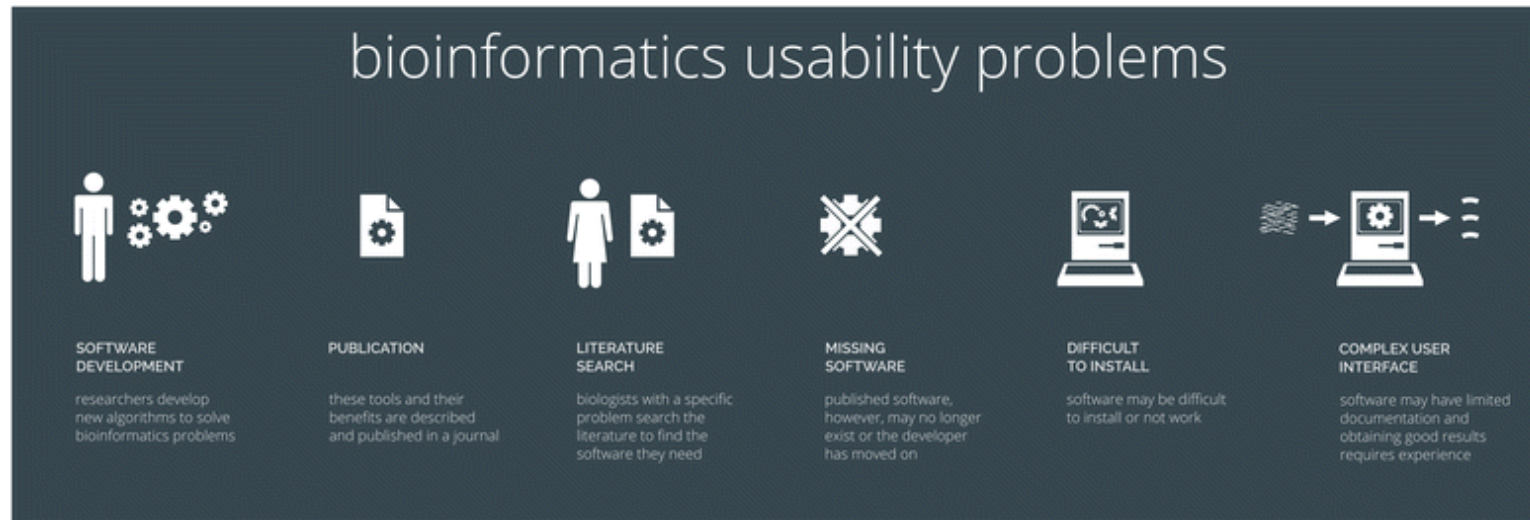
- **Containers** usually contain:
 - environment setup and dependencies
 - script(s) to ingest, analyse and report results
- But usually **don't** contain the research data itself.
- Data normally read/written from attached **volumes**
- Containers are frequently used to scale up analyses, e.g.
 - My Great Pipeline
 - Containerise: `MyGreatContainer`
 - Replicate:

```
MyGreatContainer(dataset_01);  
MyGreatContainer(dataset_02);  
MyGreatContainer(dataset_03);
```

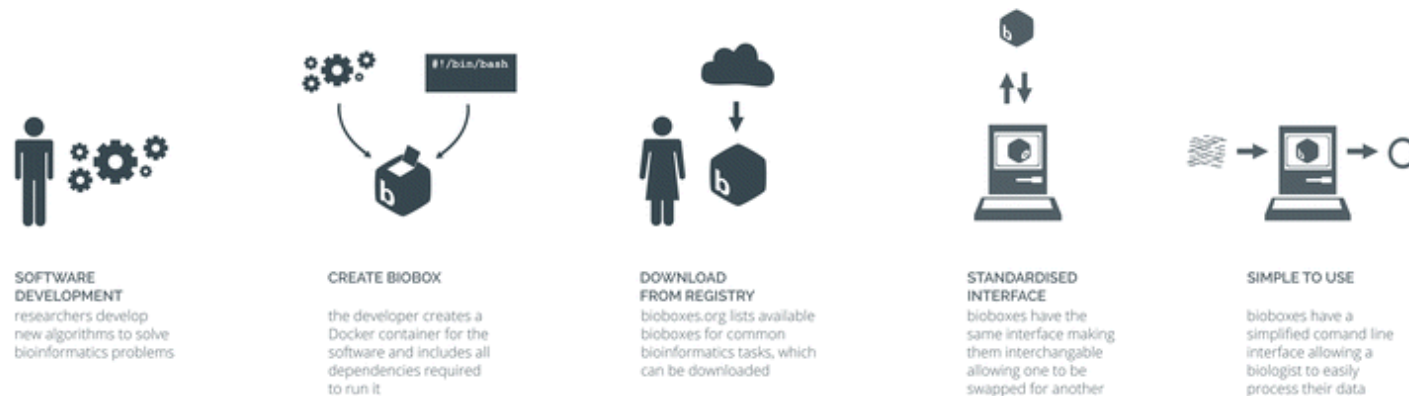
Automated testing / integration

- Build testing and integration commonly used in enterprise computing to ensure build quality, and flag up bugs as they arrive:
- Commit change → Build → test → Integrate passing builds to production code
- Tests fired automatically whenever changes made
- Changes (so improvements) can be made continuously (*'continuous integration'*)
- Requires the existence of test methods and data, written early to be effective
- **Continuous analysis** is an emerging idea in bioinformatics, mirroring this philosophy

Bioboxes



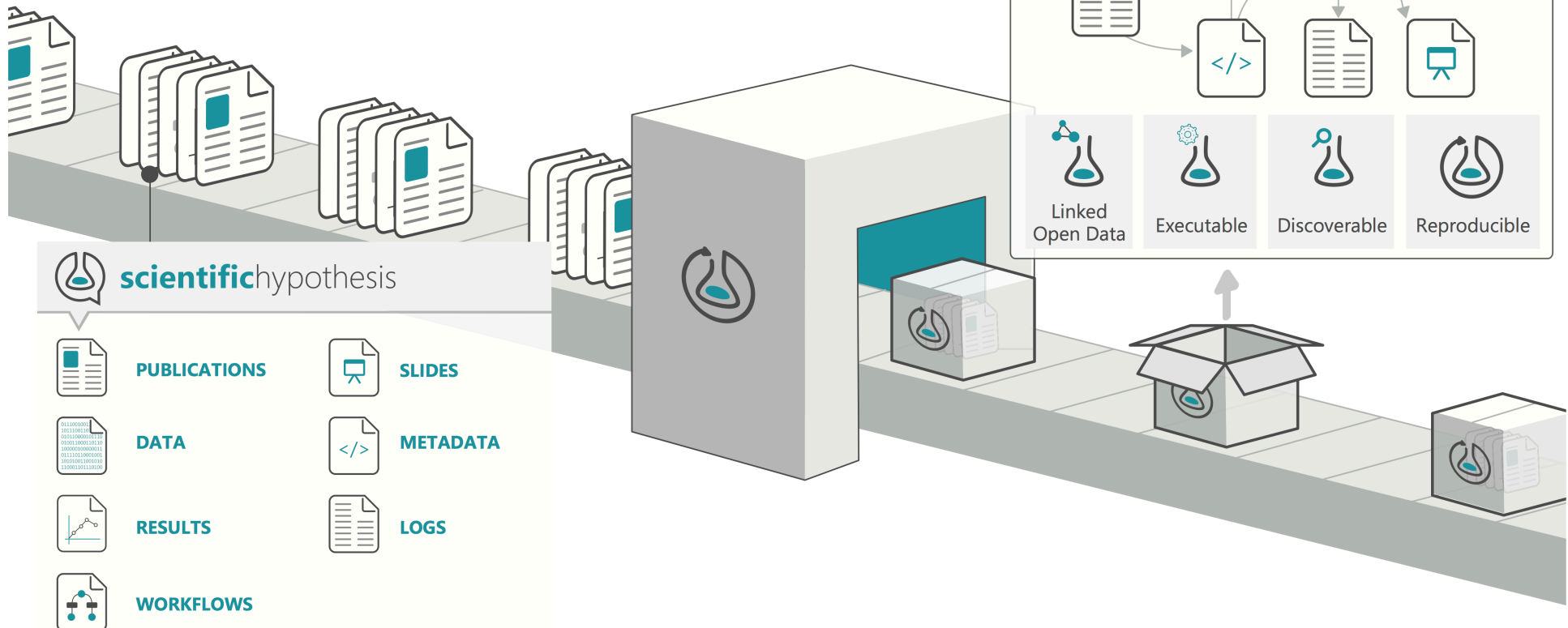
alternative with **bioboxes**



Effectively a wrapper for containerised Docker pipeline

ResearchObject

 Enabling **reproducible**, transparent research.



Integrated data import / analysis / report generation: relatively new but something like this will take off soon...