Optimizing Your Sims

The Idea Behind Optimization

- Make your code run faster

The Idea Behind Optimization

- Make your code run faster
- Luria-Delbrück sim takes 1.62 s

The Idea Behind Optimization

- Make your code run faster
- Luria-Delbrück sim takes 1.62 s
- Do 1000X to get distribution of results -> 27 minutes

The Idea Behind Optimization

- Make your code run faster
- Luria-Delbrück sim takes 1.62 s
- Do 1000X to get distribution of results -> 27 minutes
- 3 parameters (mu, r, n_sample), 20 values of each

The Idea Behind Optimization

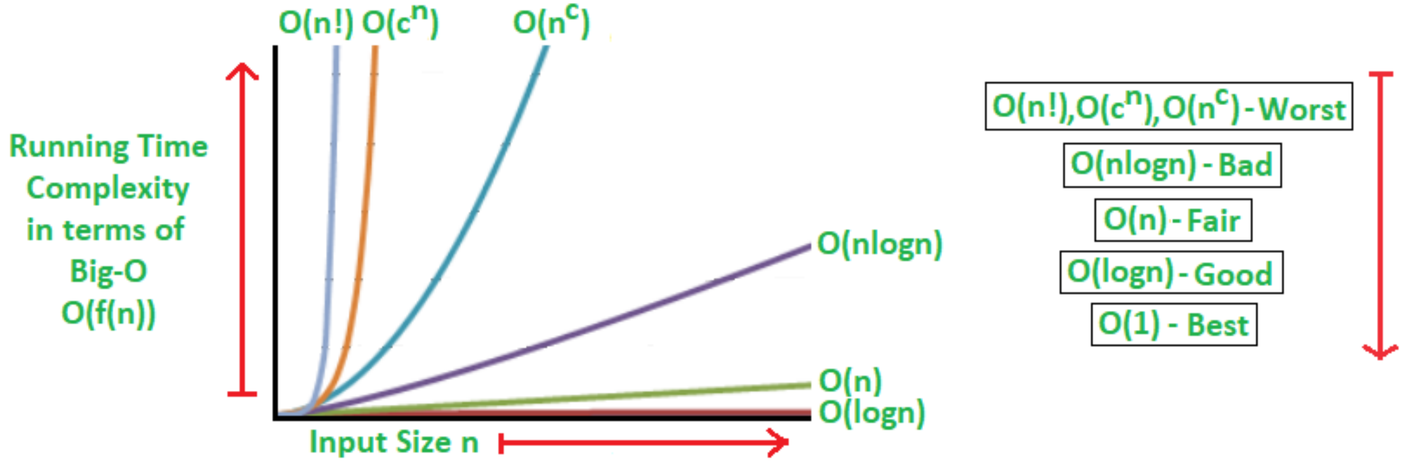
- Make your code run faster
- Luria-Delbrück sim takes 1.62 s
- Do 1000X to get distribution of results -> 27 minutes
- 3 parameters (mu, r, n_sample), 20 values of each
- 20 x 20 x 20 x 1000 = 8,000,000 sims -> 12960000 seconds

The Idea Behind Optimization

- Make your code run faster
- Luria-Delbrück sim takes 1.62 s
- Do 1000X to get distribution of results -> 27 minutes
- 3 parameters (mu, r, n_sample), 20 values of each
- 20 x 20 x 20 x 1000 = 8,000,000 sims -> 12960000 seconds

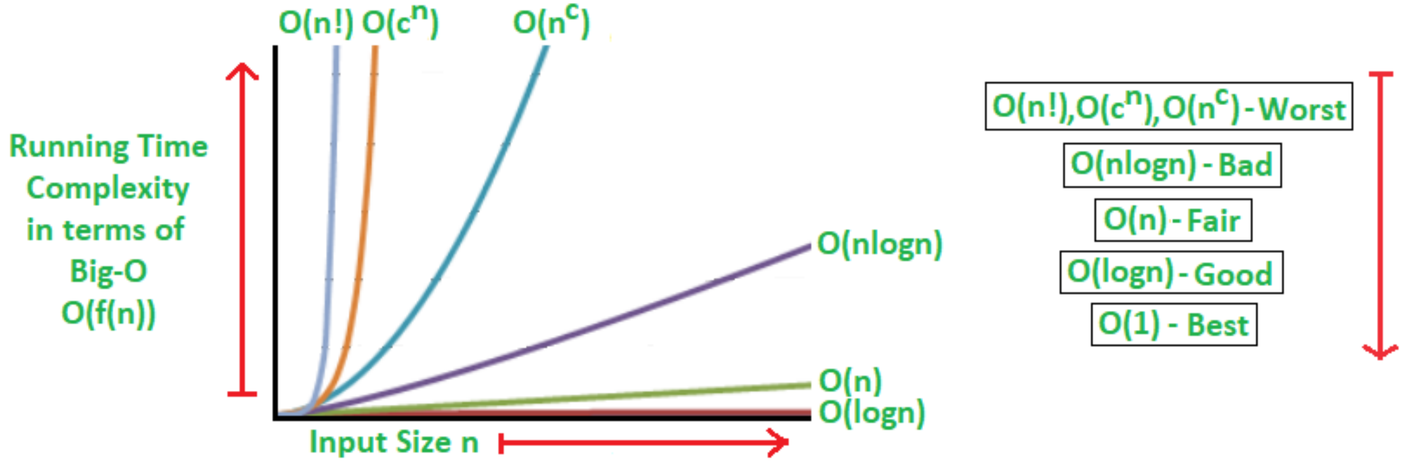
150 days, or 5 months!

Big O notation



- Wiki definition:
  - “In computer science, big O notation is used to classify algorithms according to how their run time or space requirements grow as the input size grows.”

Big O notation



- Wiki definition:
  - “In computer science, big O notation is used to classify algorithms according to how their run time or space requirements grow as the input size grows.”
- What’s big O for our sim?
  - # of sims is O(n) without parallelization
  - Simulating a gen is O(2^n). Could this be better?

How to make your code faster

Options

Using a fast language

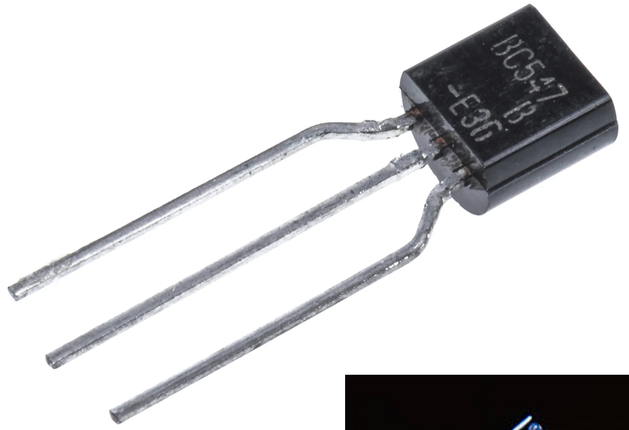
Refactoring

Using a fast library

Parallelization

How do computers work

Transistors



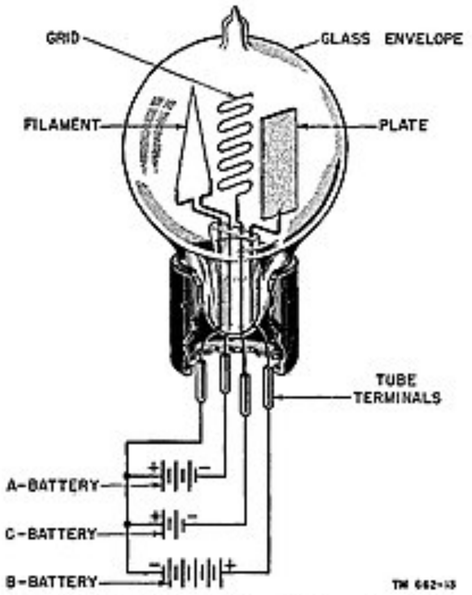


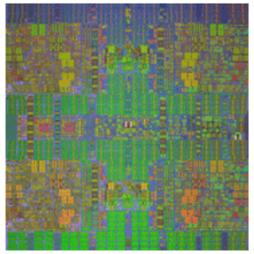
Figure 4. Construction of DeForest's three-element tube, or triode.

How do computers work

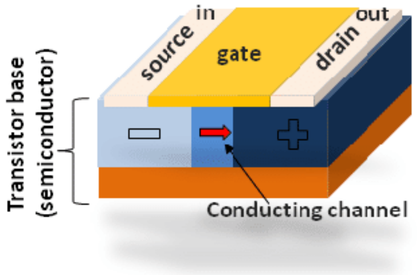
Transistors

A

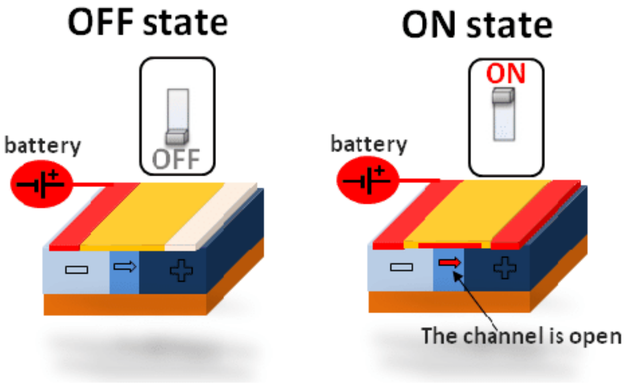
Microprocessor Chip

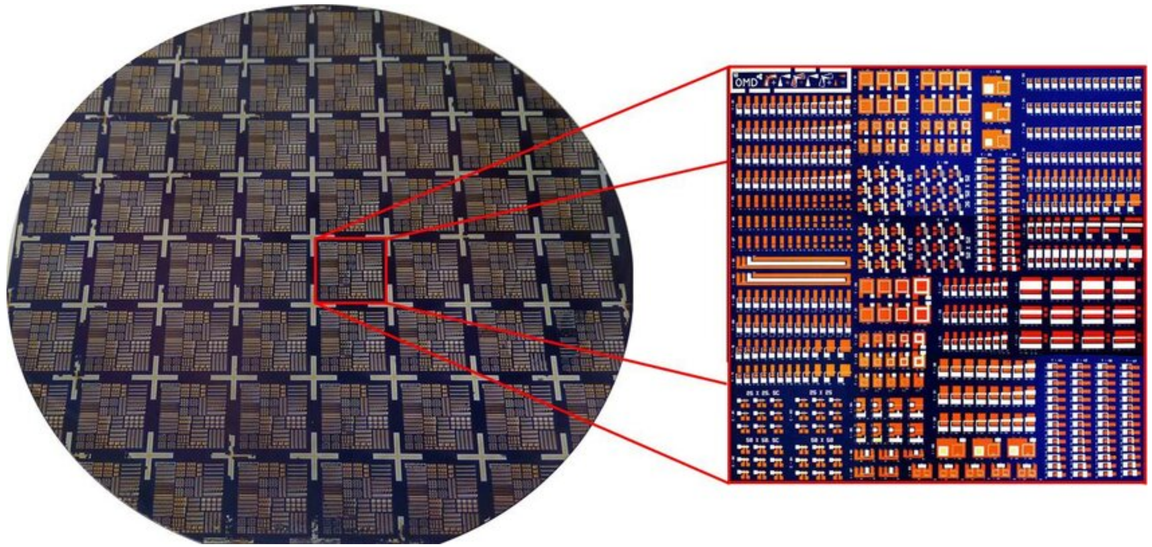


3D view of a transistor



B

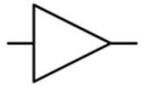




How do computers work?

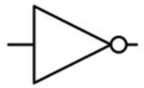
Logic Gates

Buffer



Input	Output
0	0
1	1

Inverter



Input	Output
0	1
1	0

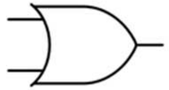
AND

A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

NAND

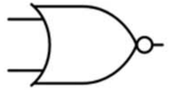
A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

OR



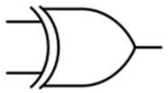
A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

NOR



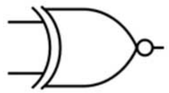
A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

XOR



A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

XNOR



A	B	Output
0	0	1
1	0	0
0	1	0
1	1	1

How do computers work?

Adders, Subtracters, Etc


MEMORY
MRC
M−
M+
√
%
→
7
8
9
÷
CE
4
5
6
×
AC
1
2
3
−
0
00
.
=
+

How Code Runs

Assembly Language/Machine Code

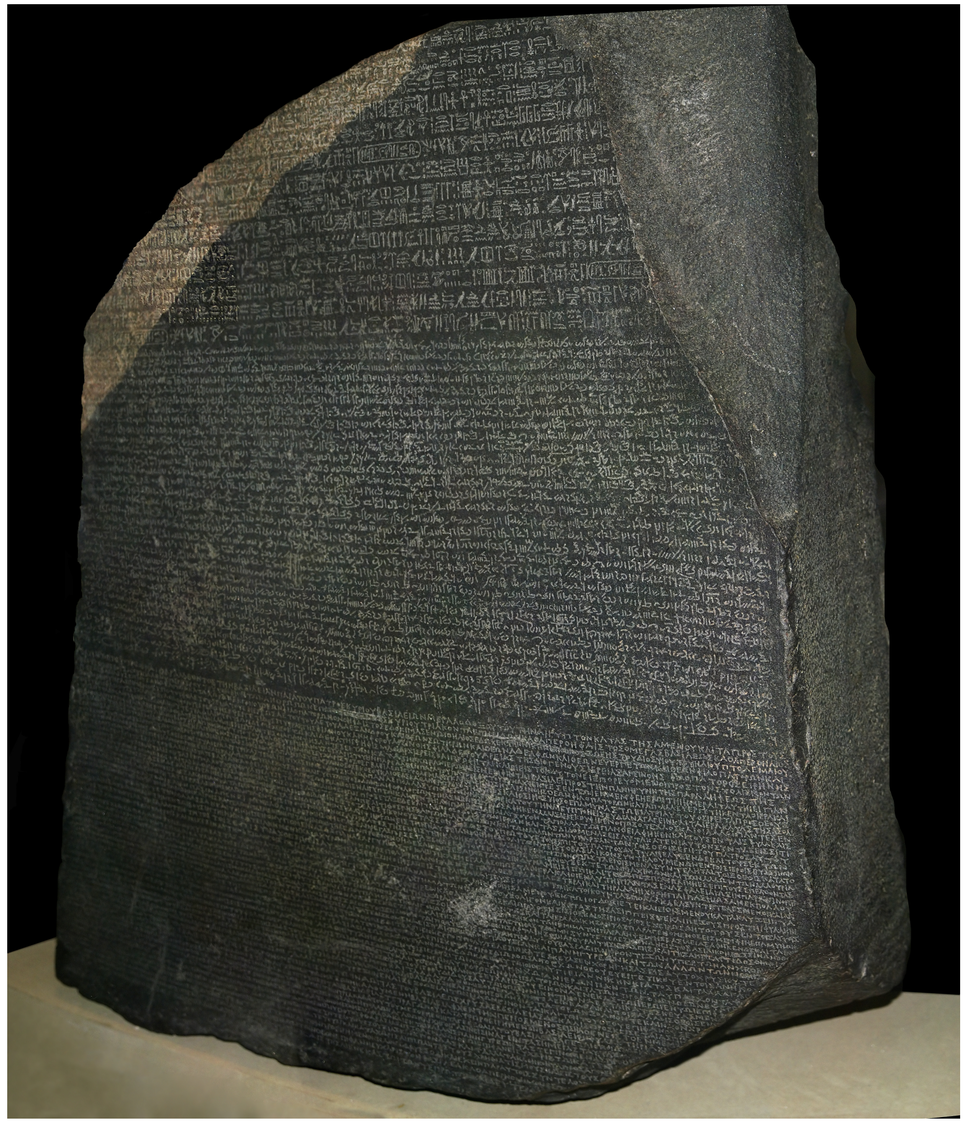
```
begin
PAUSE
MOV Rsw Rsrc1
MOV Rsw R7seg
PAUSE
MOV Rsw Rsrc2
MOV Rsw R7seg
ADD
PAUSE
MOV Rdest R7seg
end
```

```
0000
F000
0079
0078
F000
007A
0078
1100
F000
00B8
```

How Code Runs

Compilers and Interpreters

- Python and R are interpreted
- Much of both are written in C++, C, or Fortran



C++, C, Fortran

Low Level Languages

- Closer to machine code
- Run much faster than R or Python
  - High level languages
- Trade-off is in readability
  - Writing time vs Running time

How to make your code faster

- Change language
  - C++ is king, Julia is rising star

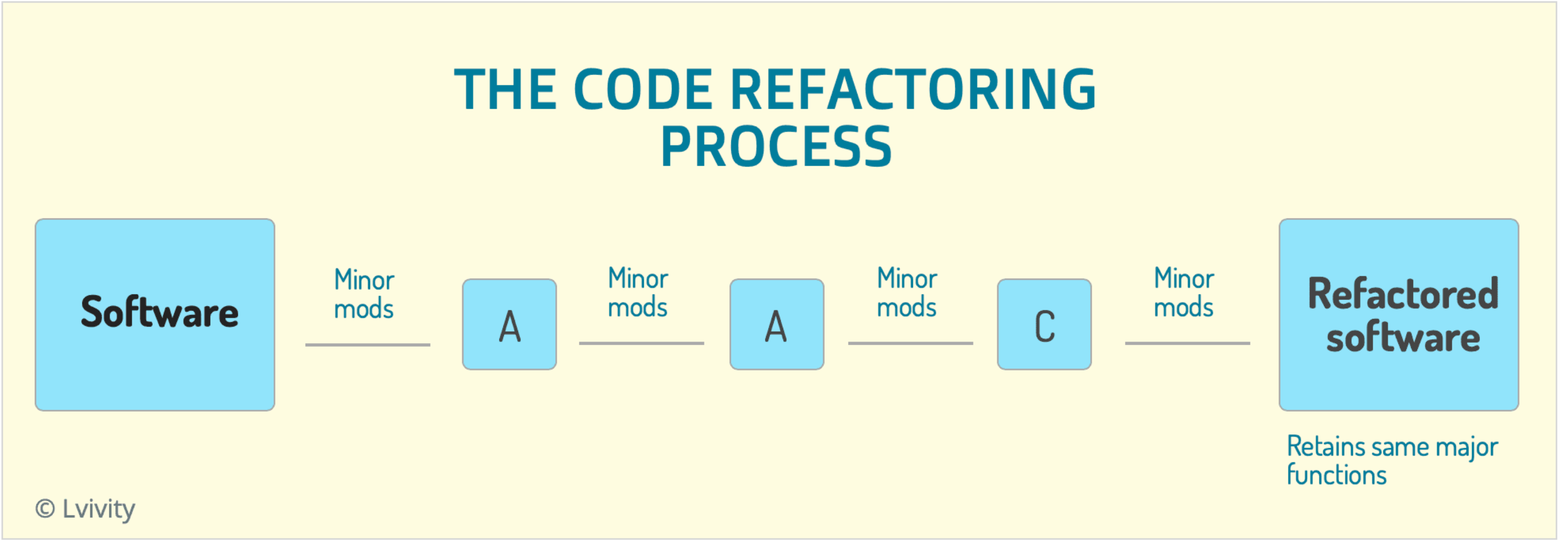
How to make your code faster

- Change language
  - C++ is king, Julia is rising star
- Use libraries written in low level languages
  - For python: Numpy and Numba

How to make your code faster

- Change language
  - C++ is king, Julia is rising star
- Use libraries written in low level languages
  - For python: Numpy and Numba
- Refactor your code

Refactoring



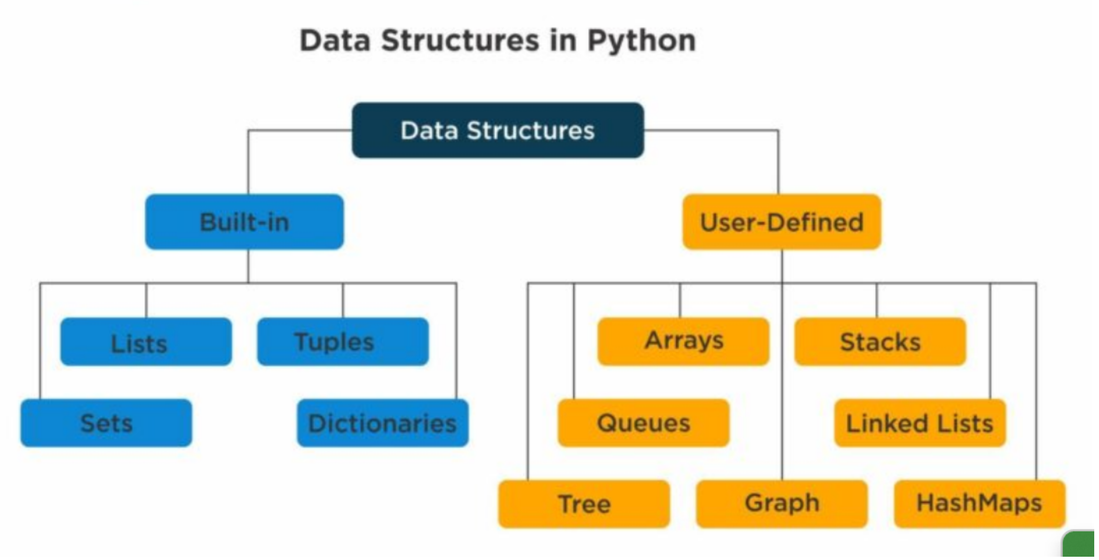
- Changing internals without changing existing behavior
- New way to do the same thing

Refactoring Strategies

- Understand data structures and how to access them correctly
- Take stock of the number of operations
- Be efficient with memory
- Vectorize!!
- Use pre-optomized functions

Understanding Data Structures

- Python and R are object oriented languages
- Saves on time and memory
- Eg:
  - Accessing by name -> Dictionary
  - Accessing by position -> List
  - Only need unique values -> Set



How many operations?

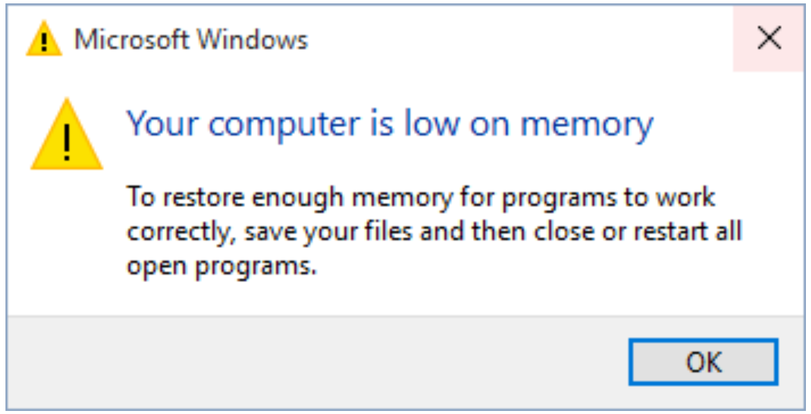
- Each step takes time -> less steps, less time
- Number of operations ~ Big O
- Eg:
  - For if then trees, break out
  - In for loops, avoid calculating the same value

```
def col_to_num(col):
    if col == 'red':
        val = 1
    if col == 'blue':
        val = 2
    if col == 'green':
        val = 3
    if col == 'yellow':
        val = 4
```

```
def col_to_num(col):
    if col == 'red':
        return = 1
    if col == 'blue':
        return = 2
    if col == 'green':
        return = 3
    if col == 'yellow':
        return = 4
```

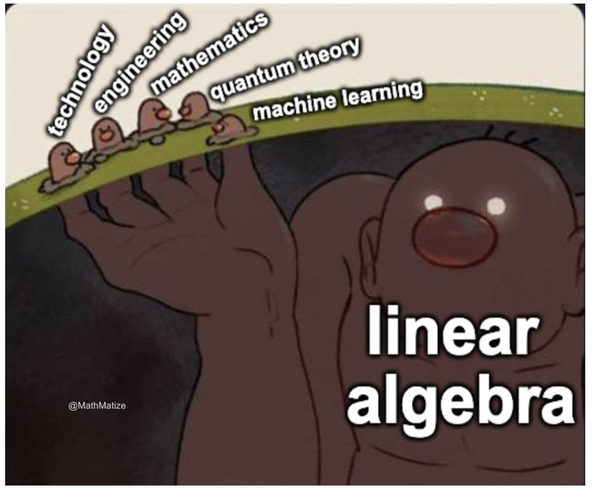
Be efficient with memory

- Don’t needlessly store variables
  - A = DataFrame([1,2,3,4])
  - A_merge = A.merge(B)
  - A_merge_filt = A_merge[A_merge > 2]
- Understand a copy vs a view
- Only read in part of a large file
  - Dask library python, bigtabulate in R
- Store data in binary
  - Pickle library in python, Save() in R



Vectorize!!

- Replacement for for loops, much faster
  - Loop infrastructure is costly, avoid it as much as possible
- Linear algebra
- Eg: Multiplying arrays
  - O(1) operation instead of O(n)
- Many existing functions for vectorization built-in & optimized
  - Even for non-math stuff
- Apply or map for custom functions
  - Beware, if a function is too complicated, apply is just doing a for loop



Use pre-made functions

- Most of the time the fastest code is the one someone else made
- Rework your problem to use existing code

How to make your code faster

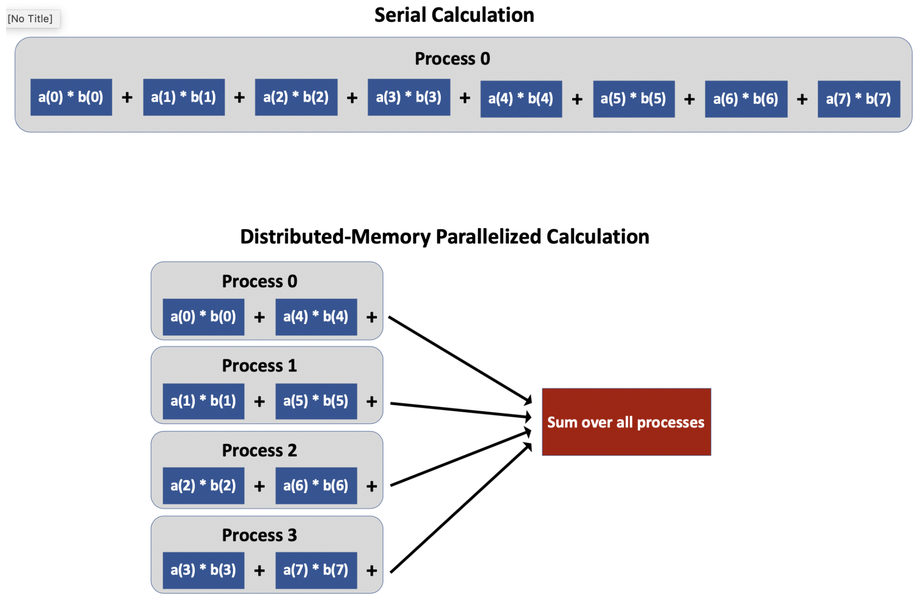
- Change language
  - C++ is king, Julia is rising star
- Use libraries written in low level languages
  - For python: Numpy and Numba
- Refactor your code
  - Understand data structures
  - How many operations?
  - Be efficient with memory
  - Vectorize!
  - Use existing functions

How to make your code faster

- Change language
  - C++ is king, Julia is rising star
- Use libraries written in low level languages
  - For python: Numpy and Numba
- Refactor your code
  - Understand data structures
  - How many operations?
  - Be efficient with memory
  - Vectorize!
  - Use existing functions
- Parallelize

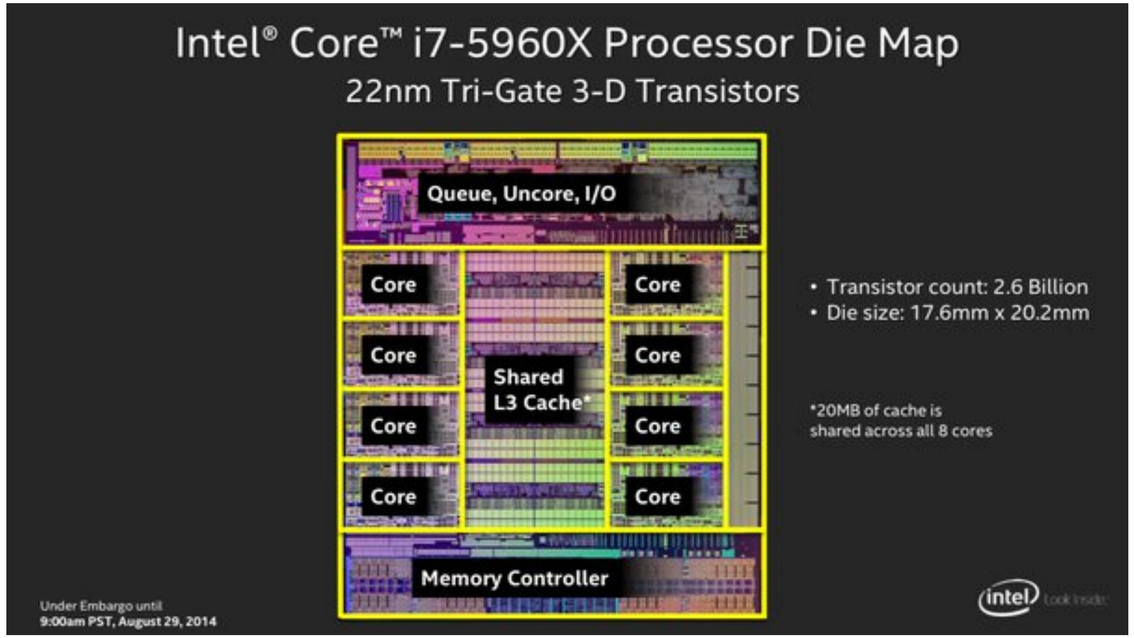
Parallelization

- Ideal for repetitive AND independent tasks
- Two levels
  - Parallel processing on local machine
  - Computing Cluster



Parallelizing on local machine

- Running code in parallel on different cores in your CPU
- Foreach R, multiprocess in Python
- Can be tricky to get going
  - Keeps your local machine busy
- Big Caveat: Not always faster
  - Infrastructure is expensive



Using a computing cluster

Aka Wallace

- Take advantage of our very powerful and expensive computing cluster!
- Best way to get a lot of heavy code to run fast
  - Frees up your personal machine
- Jobs scheduled with Slurm
- Submitted with sbatch

Tricks for speed on Wallace

- See what nodes are busy - squeue
- Request minimal resources
- Launch a job to submit your jobs
  - Create a python script to write sbatch commands
- Group tasks/iterations together
  - Initializing, Loading packages, writing err/out files all takes time

Submitting Jobs with a Job

```
#!/bin/bash
#SBATCH --job-name=SiM_Jimmy_2_central
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --time=200:00:00
#SBATCH --mem=1G
#SBATCH --error=/home/jacobsen/SLiM_sims/error/SLiM.%J.err
#SBATCH --output=/home/jacobsen/SLiM_sims/error/SLiM.%J.out
#SBATCH --partition=global

eval "$(/data/modules/python/python-anaconda3-2020.02/bin/conda shell.bash hook)"
conda activate slim;

for i in {1..1000}
do
  while (( `squeue -n SiM_Jimmy_2 | wc -l` > 30 ))
  do
    sleep 5
  done

  echo $i

  sbatch --job-name=SiM_Jimmy_2 --ntasks=1 --nodes=1 --time=10:00:00 --mem=5G --error=/home/jacobsen/SLiM_sims/error/gen_sim.%J.err --output=/home/jac
obsen/SLiM_sims/error/gen_sim.%J.out --partition=highmemnew --wrap="python3 CwSd_ParamSweep.py --rep ${i}"
done
```

Script to run sim

```
import subprocess
import argparse(len(Cws),len(sds)))
import numpy as nprate(Cws):
    for j, sd in enumerate(sds):
parser = argparse.ArgumentParser()_command(Cw = Cw, disp_sd = sd, K = 200)], shell = True, stdout = subprocess.PIPE)
parser.add_argument('--rep', dest = 'rep')split('\n')[-3].split()
args = parser.parse_args()

def slim_command(pop_size = 10, mu = 1e-7, r = 1e-7, lunar_qtl_effect_sd = 0.5, lunar_qtl_proportion = 0.1, day_sd = 0.25, day_mating_interaction_sd = 0.00000001,  Cw = 0.5, initial_x = 15, initial_y = 0, disp_sd =
 0.1, K = 200, num_kids = 10, total_gens = 1500):
    slim_command = """
slim \
-d pop_size=%s \
-d mu=%s \
-d r=%s \
-d lunar_qtl_effect_sd=%s \
-d lunar_qtl_proportion=%s \
-d day_sd=%s \
-d day_mating_interaction_sd=%s \
-d comp_sd=%s \
-d initial_x=%s \
-d initial_y=%s \
-d larval_dispersal=%s \
-d K=%s \
-d num_kids=%s \
-d total_gens=%s \
reduced_model.slim
    """ % (pop_size, mu, r, lunar_qtl_effect_sd, lunar_qtl_proportion, day_sd,  day_mating_interaction_sd, Cw, initial_x, initial_y, disp_sd, K, num_kids, total_gens)
    return(slim_command)

def count_peaks(res):
        peaks = 0
        prev = 0
        going_up = True
        hist = np.histogram(res, bins = 30, range = (0,30))[0]
        for num in hist:
            if (num < prev) and going_up:
                peaks += 1
                going_up = False
            if (num > prev):
                going_up = True
            prev = num
        return peaks

Cws = np.linspace(0.025,1,40)
sds = np.linspace(0.025,1,40)

arr = np.zeros((len(Cws),len(sds)))
for i, Cw in enumerate(Cws):
    for j, sd in enumerate(sds):
        res = subprocess.run([slim_command(Cw = Cw, disp_sd = sd, K = 200)], shell = True, stdout = subprocess.PIPE)
        vals = res.stdout.decode('utf-8').split('\n')[-3].split()
        vals = np.array([float(x) for x in vals])
        arr[i,j] = count_peaks(vals)

np.save('CwSd_paramSweep/rep%s.npy' % args.rep,arr)
```

Actual Sim in SLiM

```
initialize(){
        initializeSLiMModelType("nonWF");
        initializeSLiMOptions(dimensionality='xy', periodicity='x');
        initializeSex('A');

        initializeMutationRate(mu);
        initializeMutationType('m1',0.5,'f',0.0); //neutral
        initializeMutationType('m2',0.5,'n',0.0, lunar_qtl_effect_sd); //circalunar

        m2.convertToSubstitution = F;

        initializeGenomicElementType('g1', c(m1,m2), c(1.0,lunar_qtl_proportion));
        initializeGenomicElement(g1, 0, 99999);
        initializeRecombinationRate(r);

        initializeInteractionType(1,'x',reciprocal=T, maxDistance=day_mating_interaction_sd);
        i1.setInteractionFunction('f',1.0);

        initializeInteractionType(2,'y',reciprocal=T, maxDistance=comp_sd*3);
        i2.setInteractionFunction('n',1.0,comp_sd);
}

mutationEffect(m2) {return 1.0;}

1 first() {
        sim.addSubpop("p1", pop_size);
        p1.setSpatialBounds(c(0,-1.5,30,1.5));

        init_pos = c();
        for (i in seq(0,pop_size - 1,1)) {
                init_pos = c(init_pos,c(asInteger(rnorm(1,initial_x,1)),rnorm(1,initial_y,0.05)));
        }
        p1.individuals.setSpatialPosition(init_pos);

    community.rescheduleScriptBlock(s1, start=total_gens, end=total_gens);
}

first(){
        // emergence according to genotype
        inds = sim.subpopulations.individuals;
        days = floor(inds.sumOfMutationsOfType(m2) + rnorm(length(inds),0,day_sd)) + initial_x;
        inds.x = days;
        depths = ((sin(2*PI*(days/15) - 5))/2 - 0.5);
        inds.y = depths;
        inds.setSpatialPosition(p1.pointPeriodic(inds.spatialPosition));
        i1.evaluate(p1);
}

reproduction(NULL, 'F'){
        inds = sim.subpopulations.individuals;
        inds.tag = asInteger(i1.strength(individual));
        possible_mates = p1.subsetIndividuals(sex = 'M',tag = 1);
        if (length(possible_mates) > 0){
                father = sample(possible_mates,1);
                mean_fit = mean(c(individual.fitnessScaling,father.fitnessScaling))*num_kids;
        clutchSize = asInteger(mean_fit);
        for (j in seqLen(clutchSize)){
                offspring = p1.addCrossed(individual,father);
                offspring.y = individual.y;

           }
   }
}

early(){
        inds = sim.subpopulations.individuals;
        // killing adults
        ages = inds.age;
        inds.fitnessScaling = 1-ages;
        // random dispersal
        inds.y = inds.y + rnorm(length(inds),0,larval_dispersal);
}

late(){
        i2.evaluate(p1);
        inds = sim.subpopulations.individuals;
        // selection due to competition
        strengths = i2.totalOfNeighborStrengths(inds);
        // competition = 1 - (strengths/K);
        competition = K/(exp(10*strengths/K) + K);
        // selection due to environment
        y = inds.y;
        fitness_depth = 1/(2+exp(6*y + 1)) + 0.5;
        // total fitness
        inds.fitnessScaling = competition * fitness_depth;
}


s1 late(){
```

How to make your code faster

- Change language
  - C++ is king, Julia is rising star
- Use libraries written in low level languages
  - For python: Numpy and Numba
- Refactor your code
  - Understand data structures
  - How many operations?
  - Be efficient with memory
  - Vectorize!
  - Use existing functions
- Parallelize
  - Locally on different cores
  - Remotely on Wallace