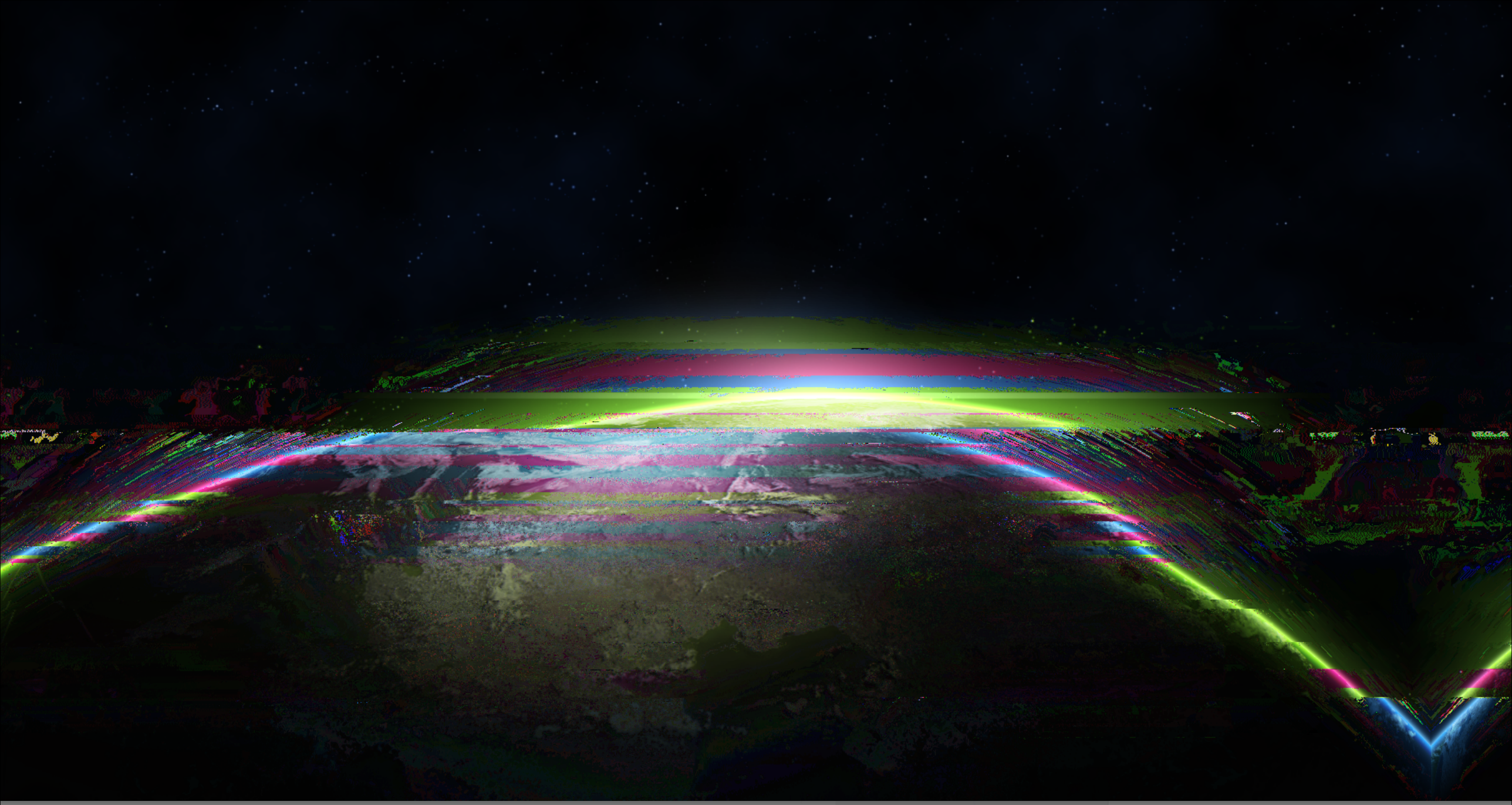
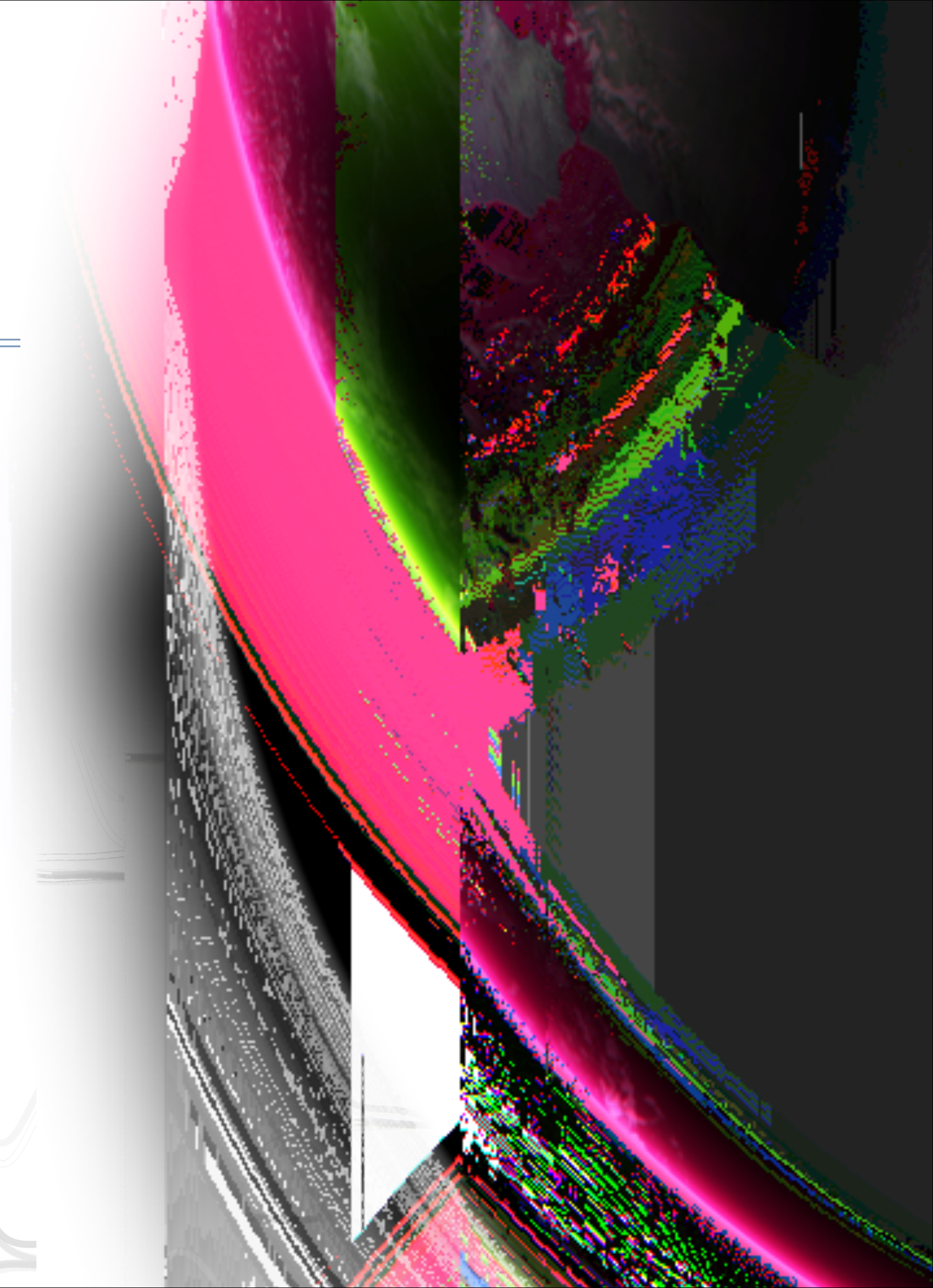


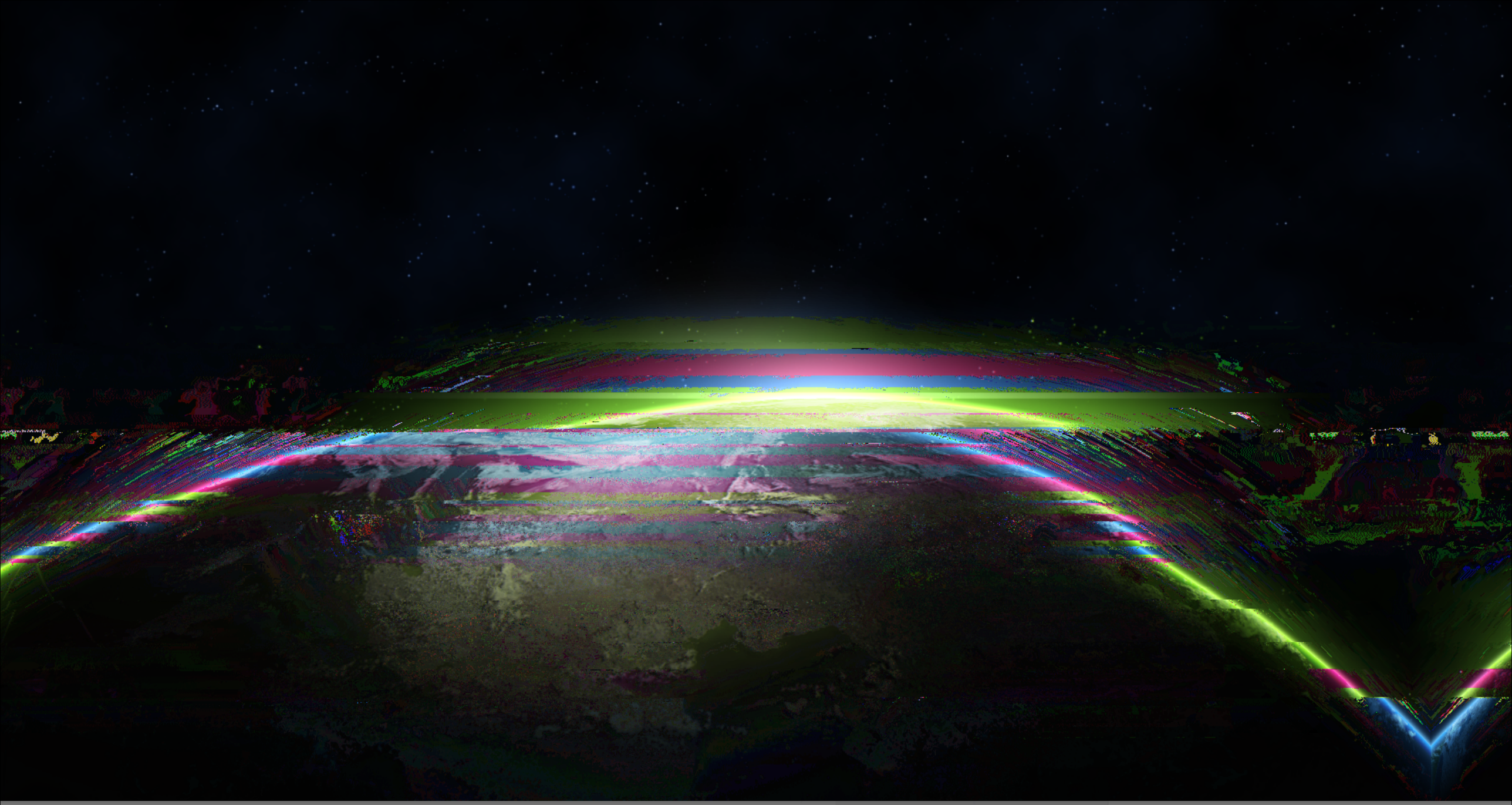
Teaching ECE408 at Scale

Abdul Dakkak

Overview

- IMPACT
- History
- Architecture
- Some data
- Lessons
- Future





Quick Demo

IMPACT

- Nothing similar exists
  - Works for CUDA/OpenCL/OpenACC/…
- "Possibly" the most visible project in our group
  - Thousands of users spent hundreds of hours on the site
- Production vs research implementation

Objective

- Create a programming environment for the Coursera course
- Allow people to develop outside of the environment
- Not be tied into Coursera (want to offer it for summer school)

Previous System

- Many user interface issues
- Had trouble with scaling
- Grading was done offline
- Did not have peer reviewing

New System

- Instant grading and submission back to Coursera
- Peer review implemented
- Scaling was kept in mind
- Making a new MP is simple

How do you grade 4000 Programs?

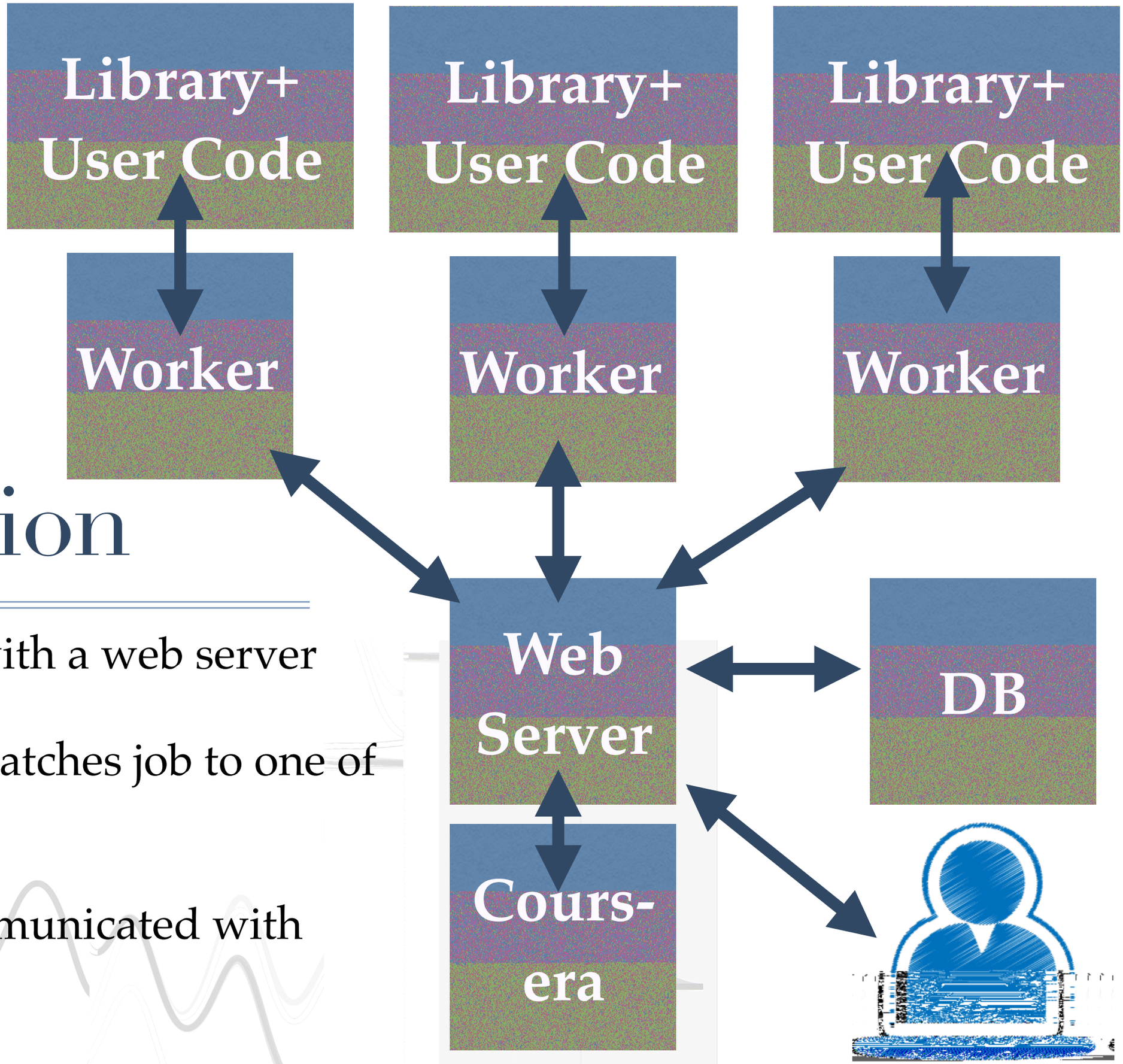
"You don't."

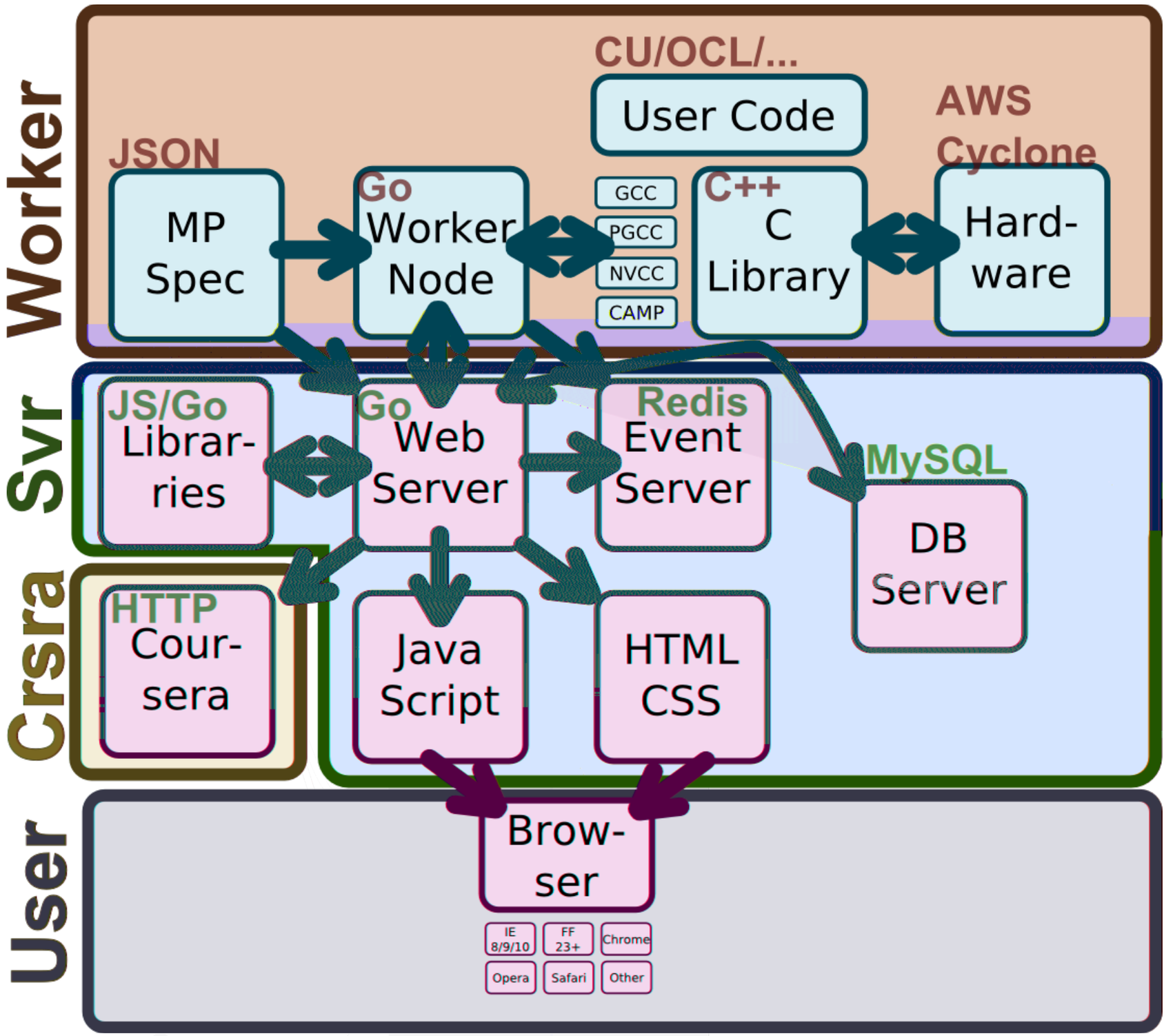
Architecture

Life of a Program Submission

Life of a Program Submission

- User interacts with a web server
- Web server dispatches job to one of many workers
- Grades get communicated with Coursera



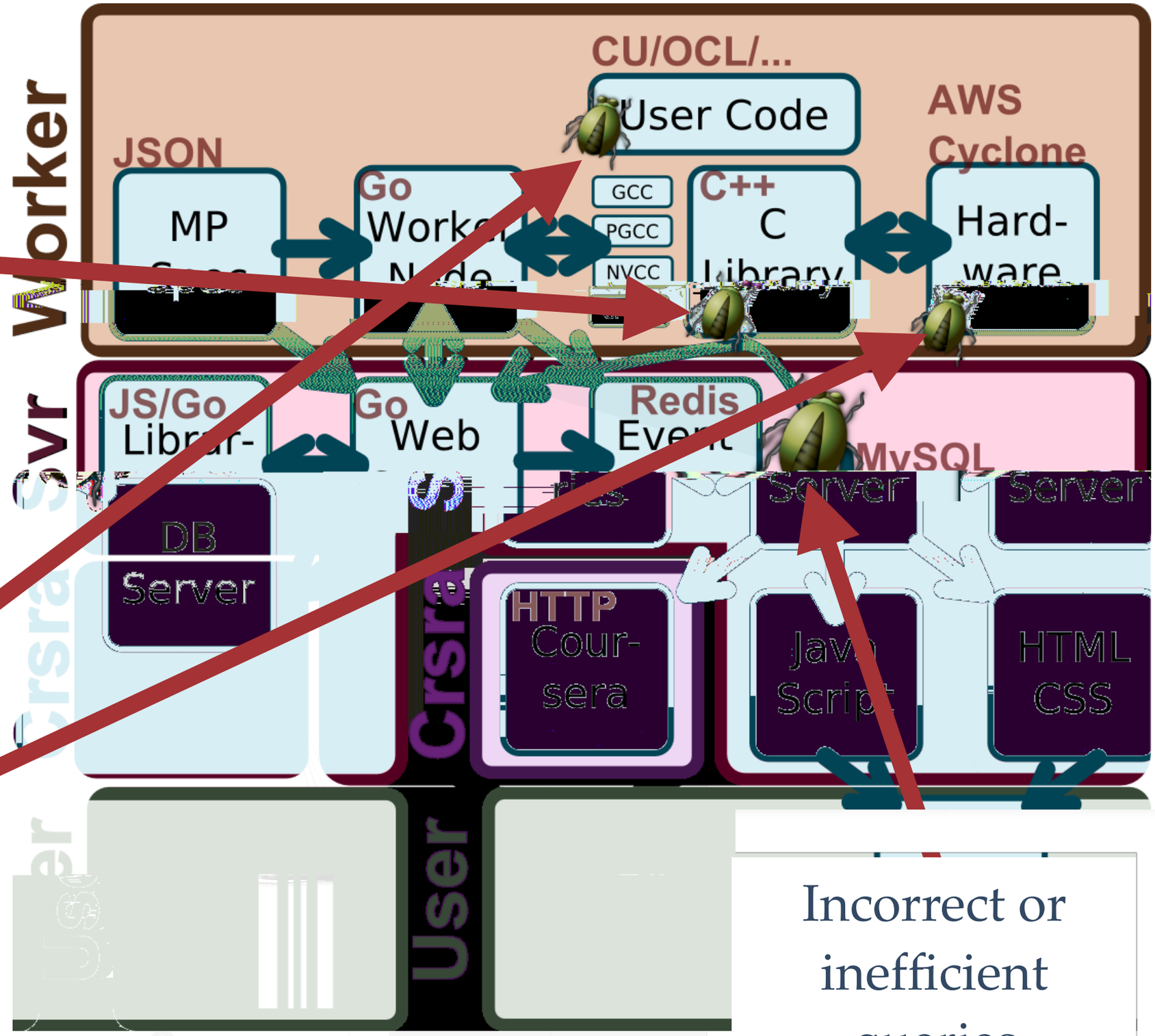


Detailed Architecture

Source of Bugs

Made it public, so got some help from students

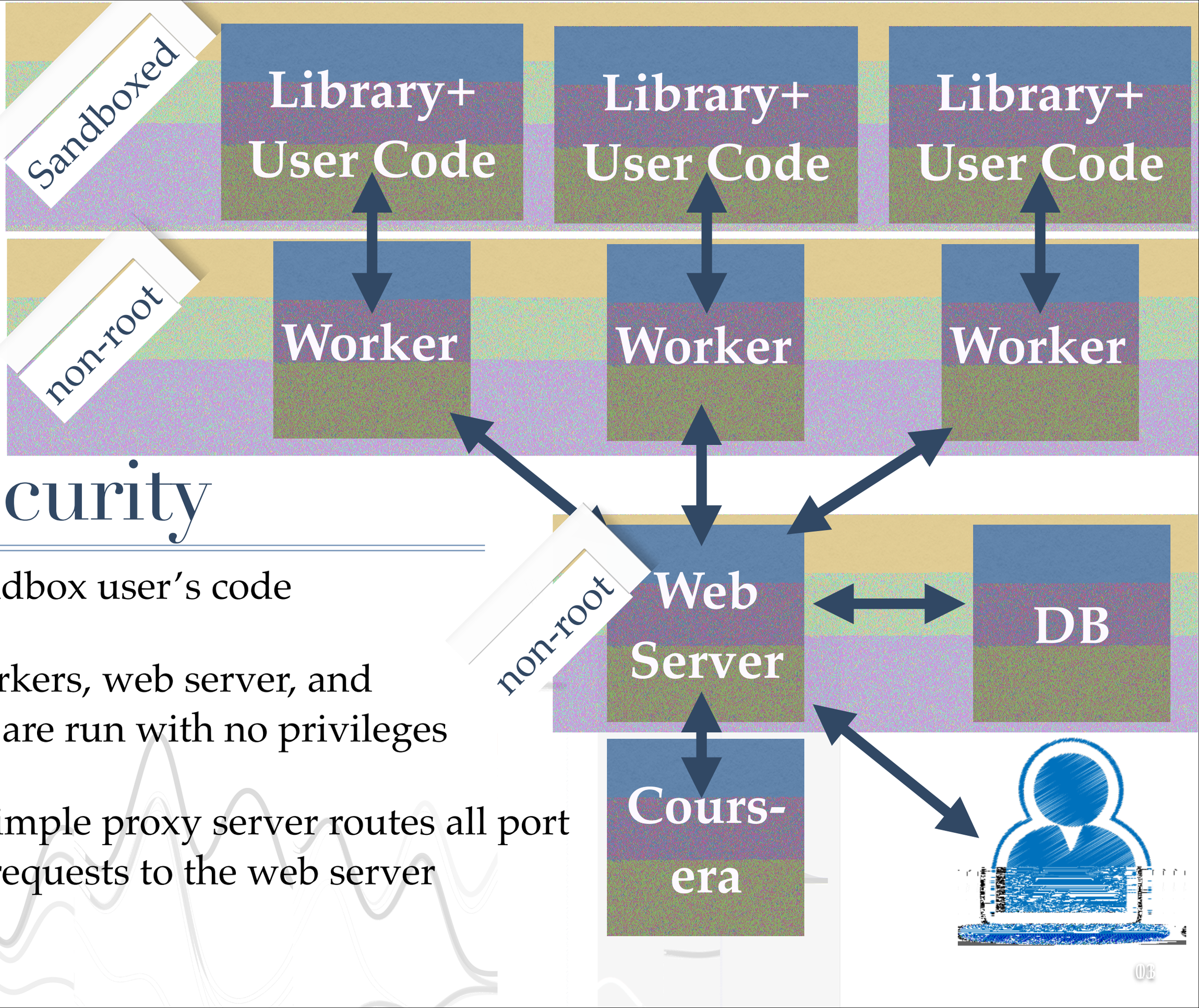
Correlated and source of major problems (happens on Cyclone and AWS)



Incorrect or inefficient queries

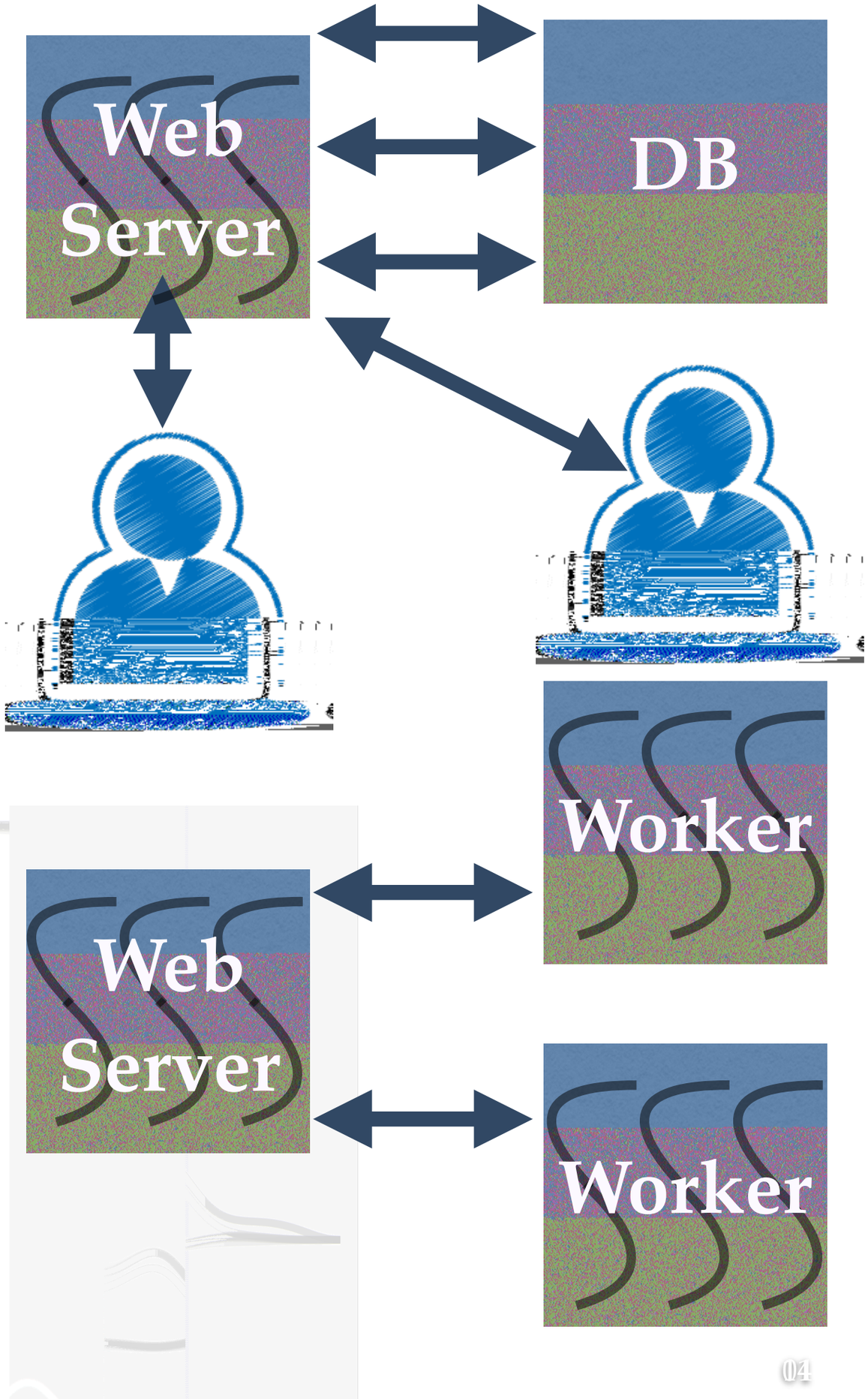
Security

- Sandbox user's code
- Workers, web server, and DB are run with no privileges
- A simple proxy server routes all port 80 requests to the web server

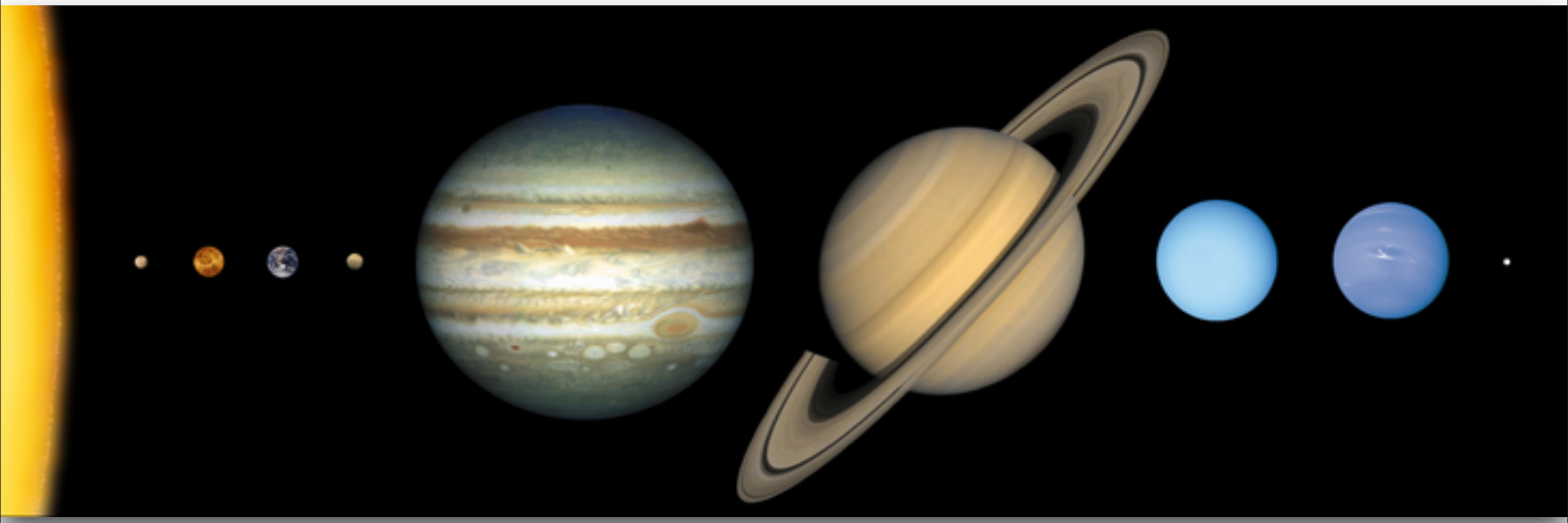


How to Scale?

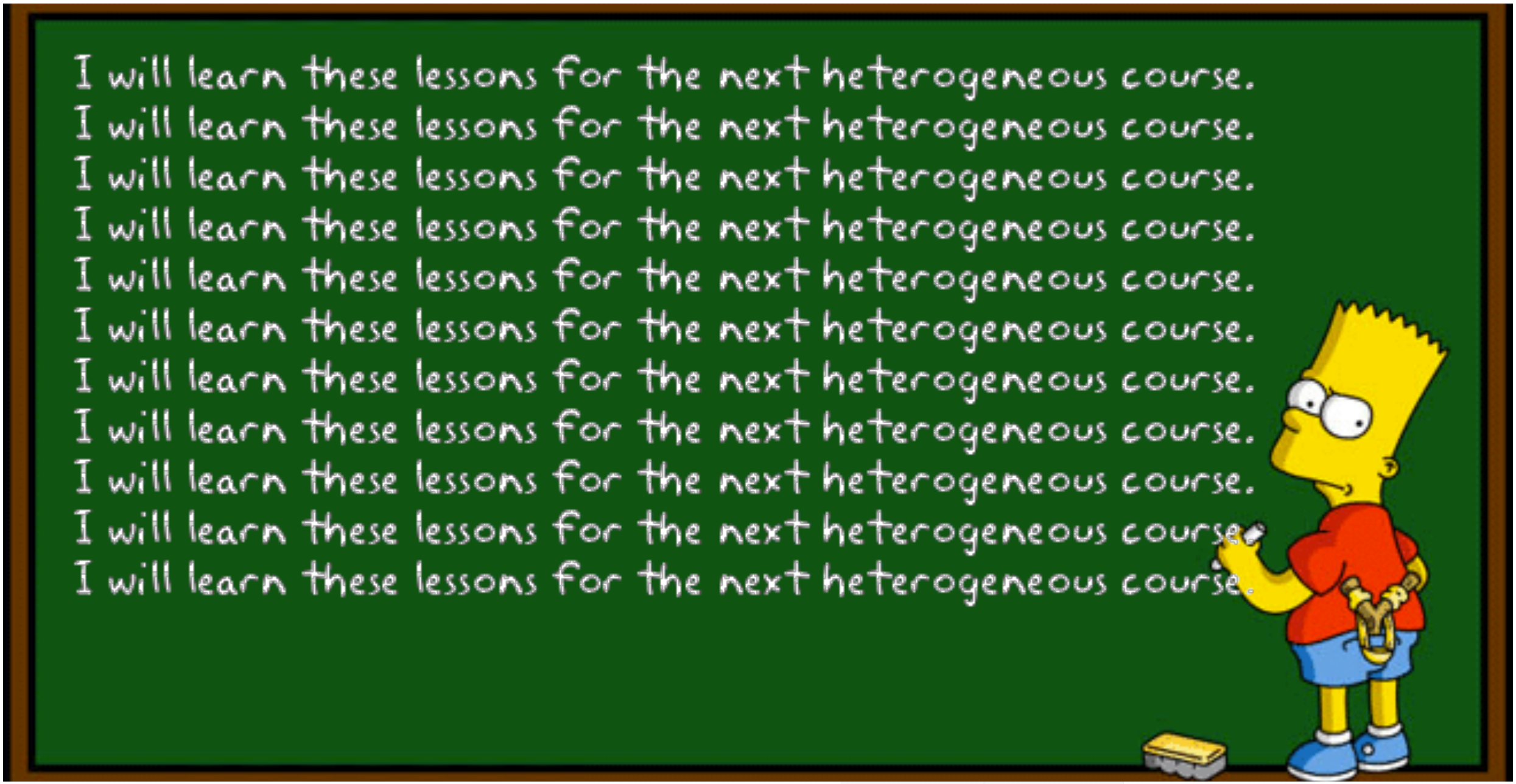
- Each request is a lightweight thread (thousands of these)
- The lightweight threads get mapped to *n* threads (*n* being the number of cores)
- Create a connection pool with the database server
- Most of operations are asynchronous
- Master can communicate to any number of workers
- Worker will run programs on different GPUs if available



Scale



- What works for 2 people may not work for 1000 people
- What works for 1000 people may not work when all are logged in at the same time
- What works for 1000 people on day 1 may not work for 1000 people on day 50

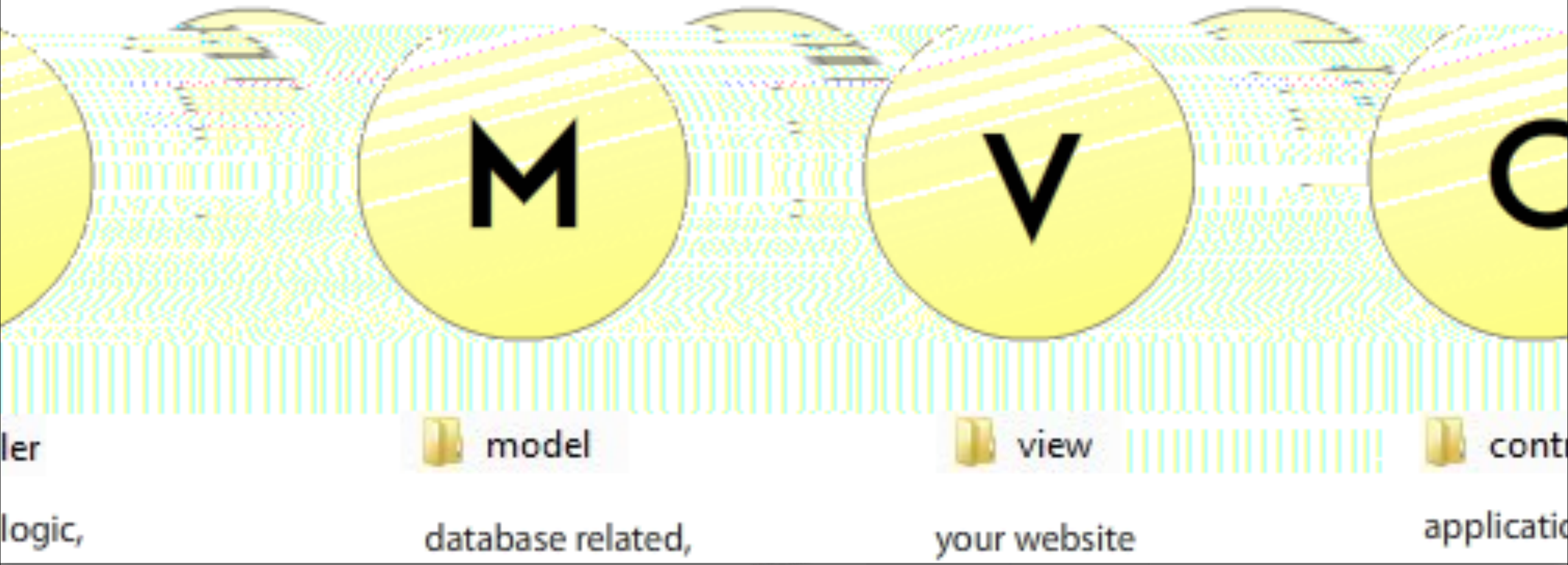


Lessons Learned (Implementation)

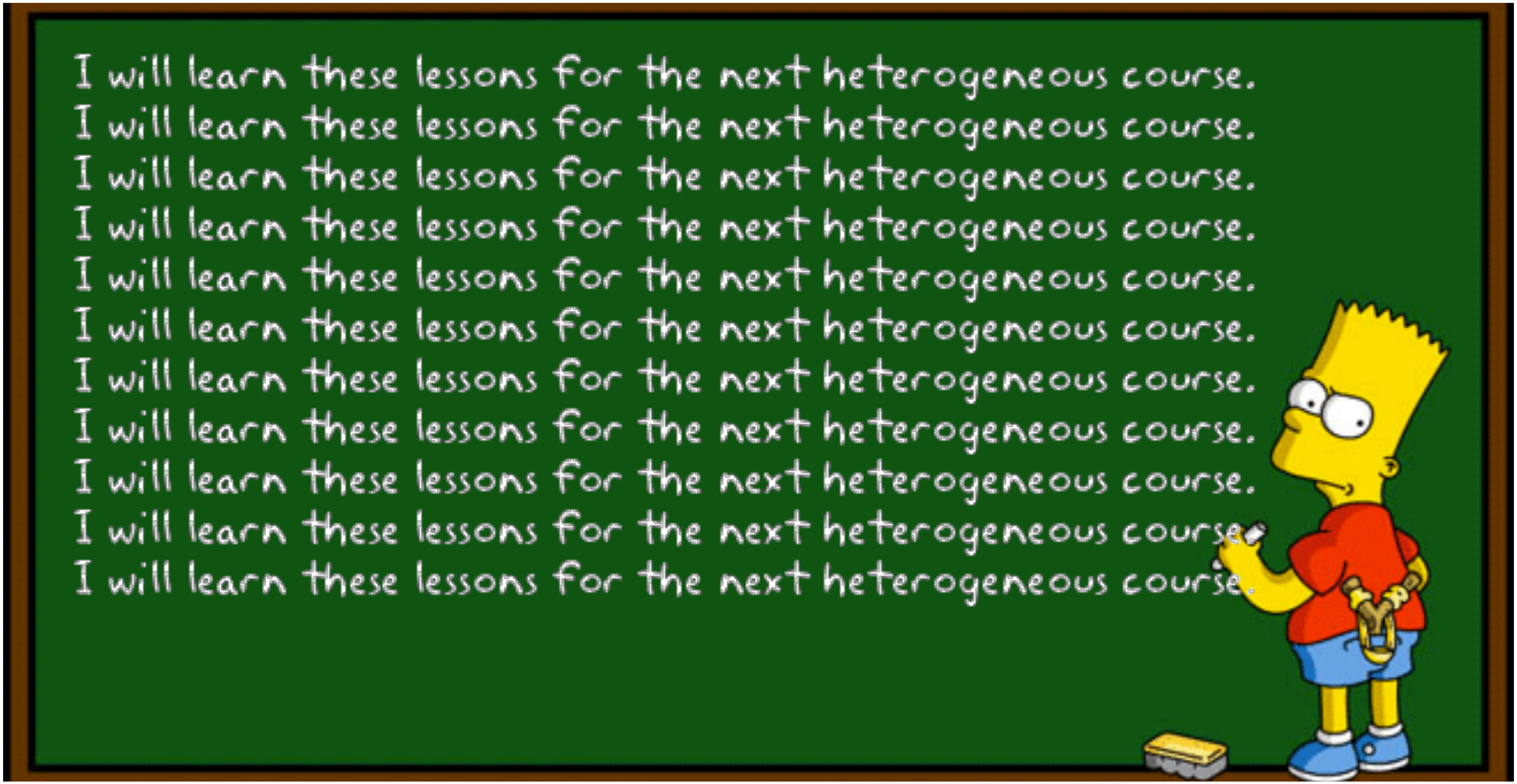
Abstraction is not a Good Thing

- At the start of the course, we were using a library to simplify database queries — it did simplify them, but generated complex queries that were not possible to debug

Abstraction is a Great Thing



- When we replaced the library, had to replace just the model code in the application



Lessons Learned (Human)

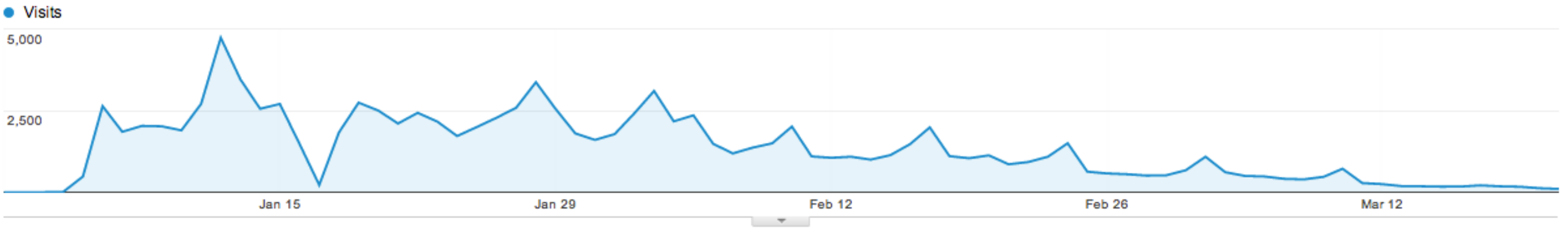
Lessons Learned

- Negative criticism speaks louder than positive ones
  - You will feel discouraged and moody for the rest of the day
- People do not read documentation or search the forums
  - You will answer the same thing over and over … and over again
- People will send you their code and ask you to debug them
  - They will feel like you are not doing your job if you tell them no

Data

Data Collected

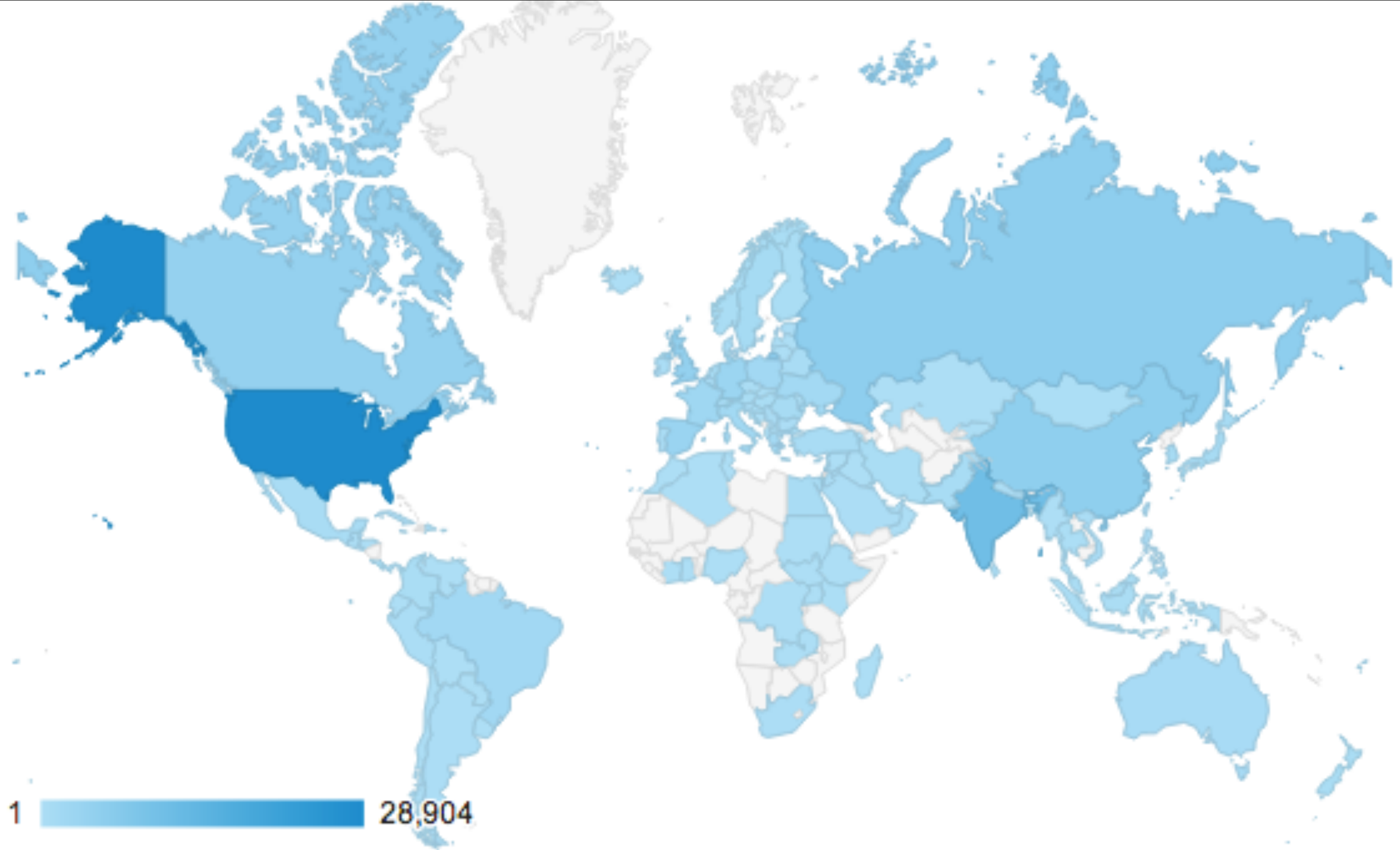
- Google Analytics
- A 13Gb database containing
  - 2 Million program revisions
  - 700 thousand program runs
    - Runtime/Compilation time / errors
  - 6 thousand graded programs
- A 7Gb Event database containing
  - CPU/GPU information
  - What pages users Visited



Visits	Unique Visitors	Pageviews	Pages / Visit
106,427	17,865	998,517	9.38

Avg. Visit Duration	Bounce Rate	% New Visits
00:14:20	20.54%	16.60%

Thousands of Visitors



Users from All over the World

Visits
106,427

Unique Visitors
17,865

Pageviews
998,517

Pages / Visit
9.38

Avg. Visit Duration
00:14:20

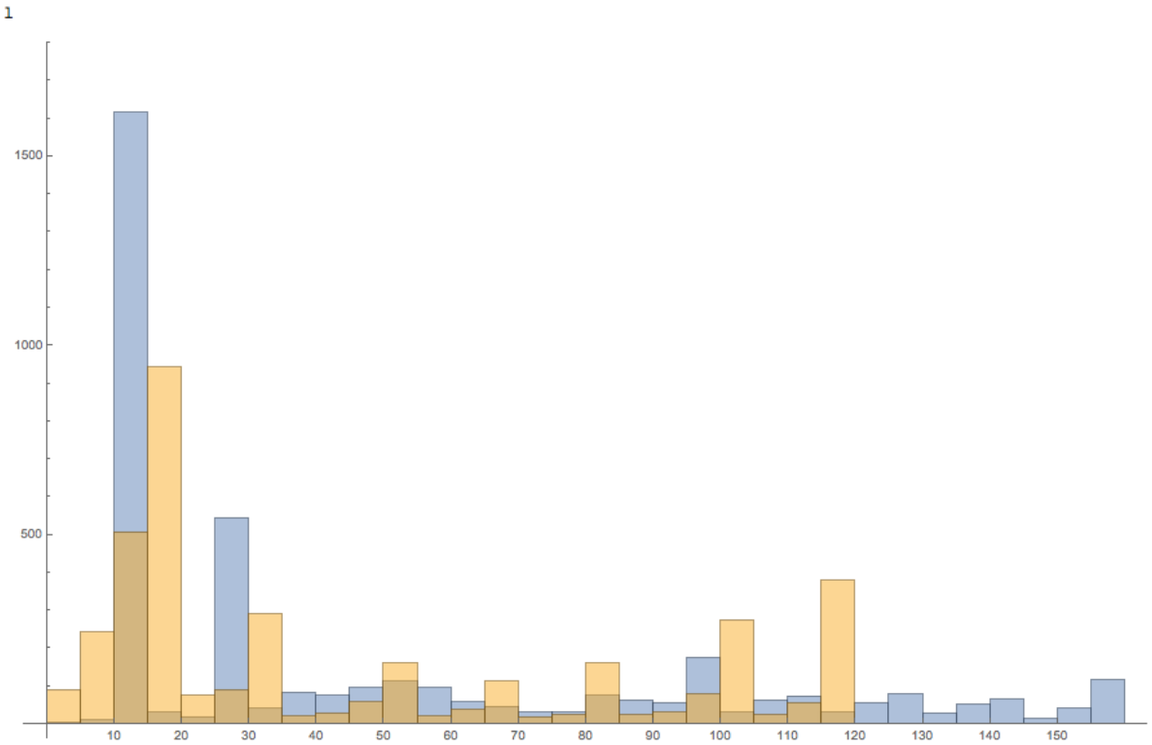
Bounce Rate
20.54%

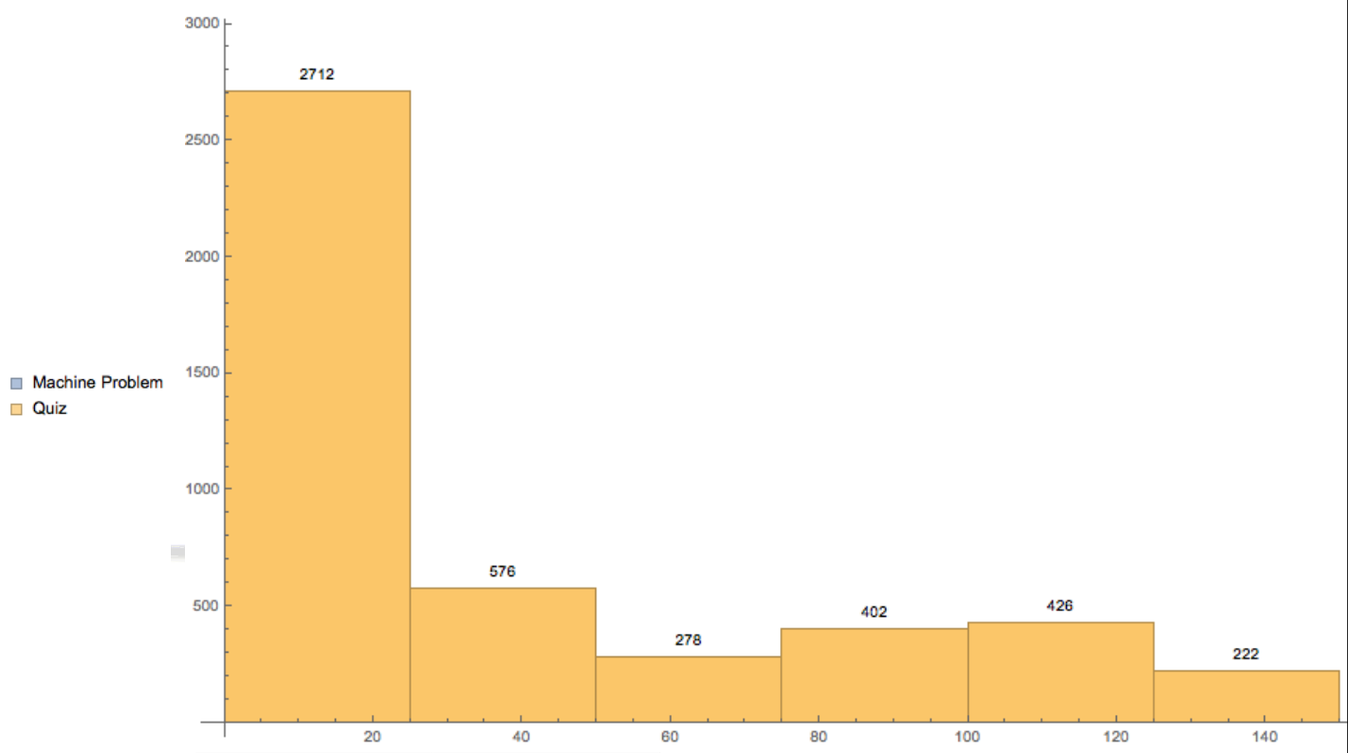
% New Visits
16.60%

People Spend Time on the Site

Grades

Most People Passed





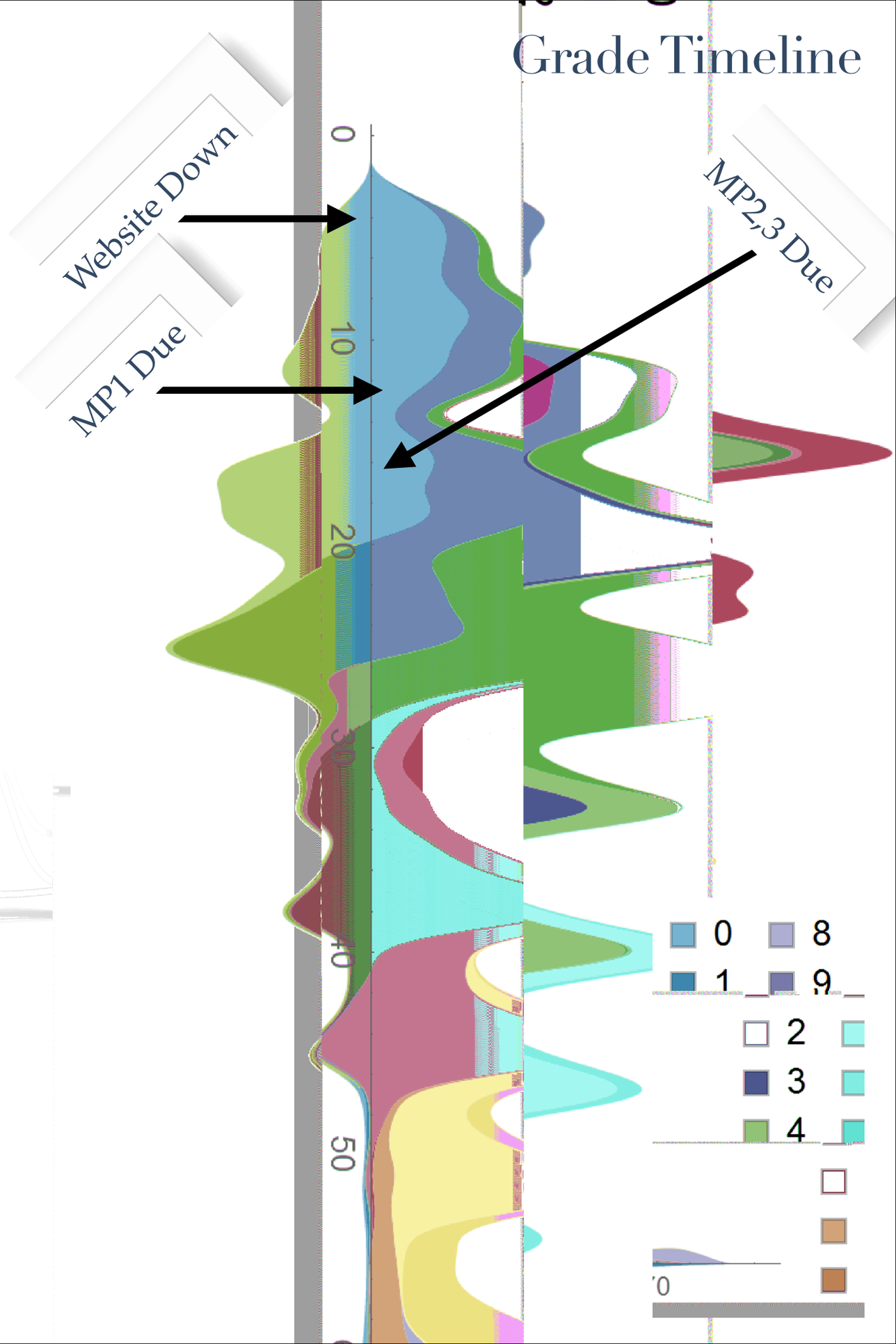
- 939 actually started the course
- 989 got over 80%
- 648 got over 100%

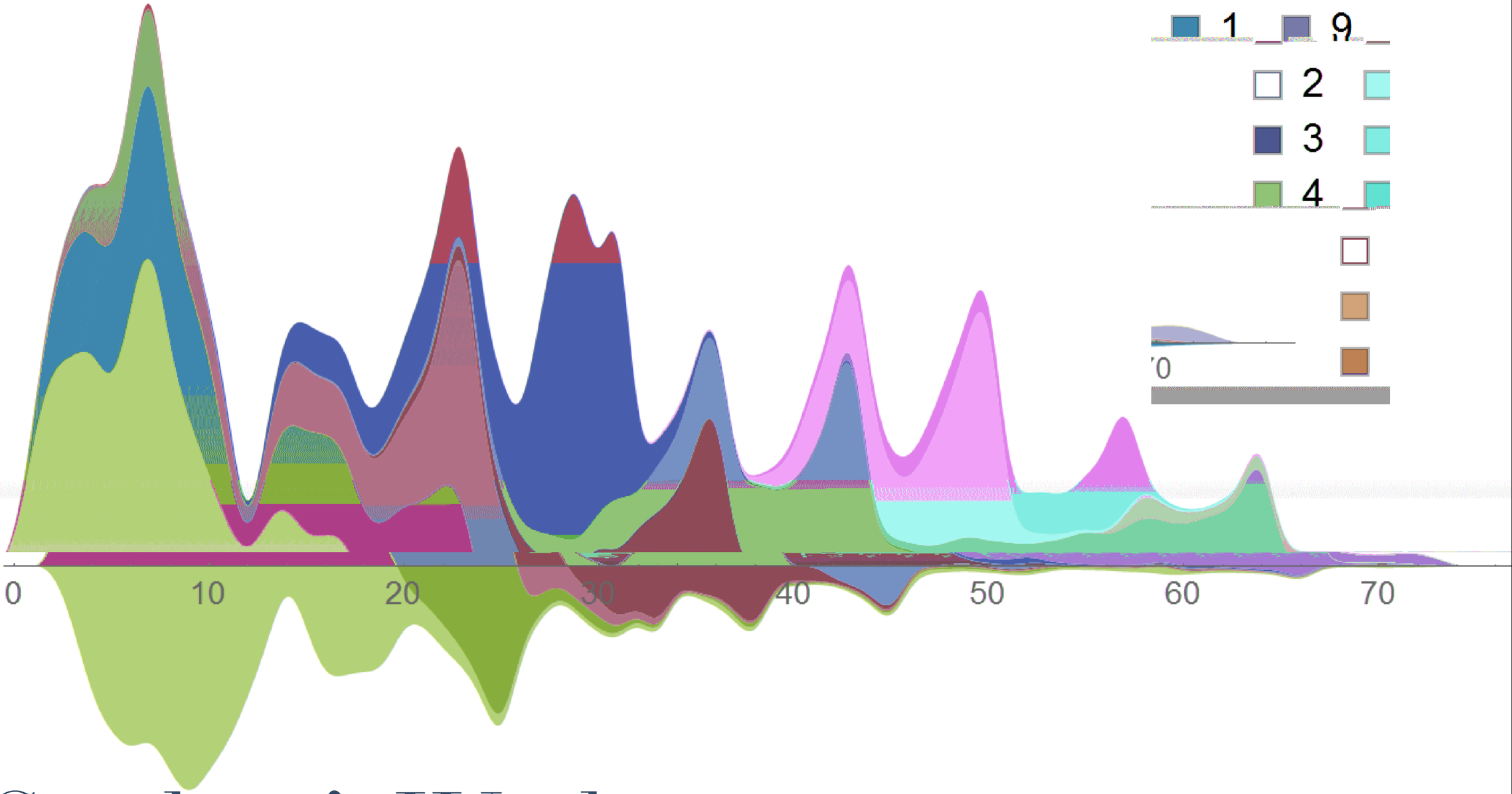
Some Data Insights

Date

People Submit at the Last Minute

- We made all course MPs available on day 1
- Everyone waited until the last minute
- Every MP has 2 bumps — one for submitting the code and another for the peer review

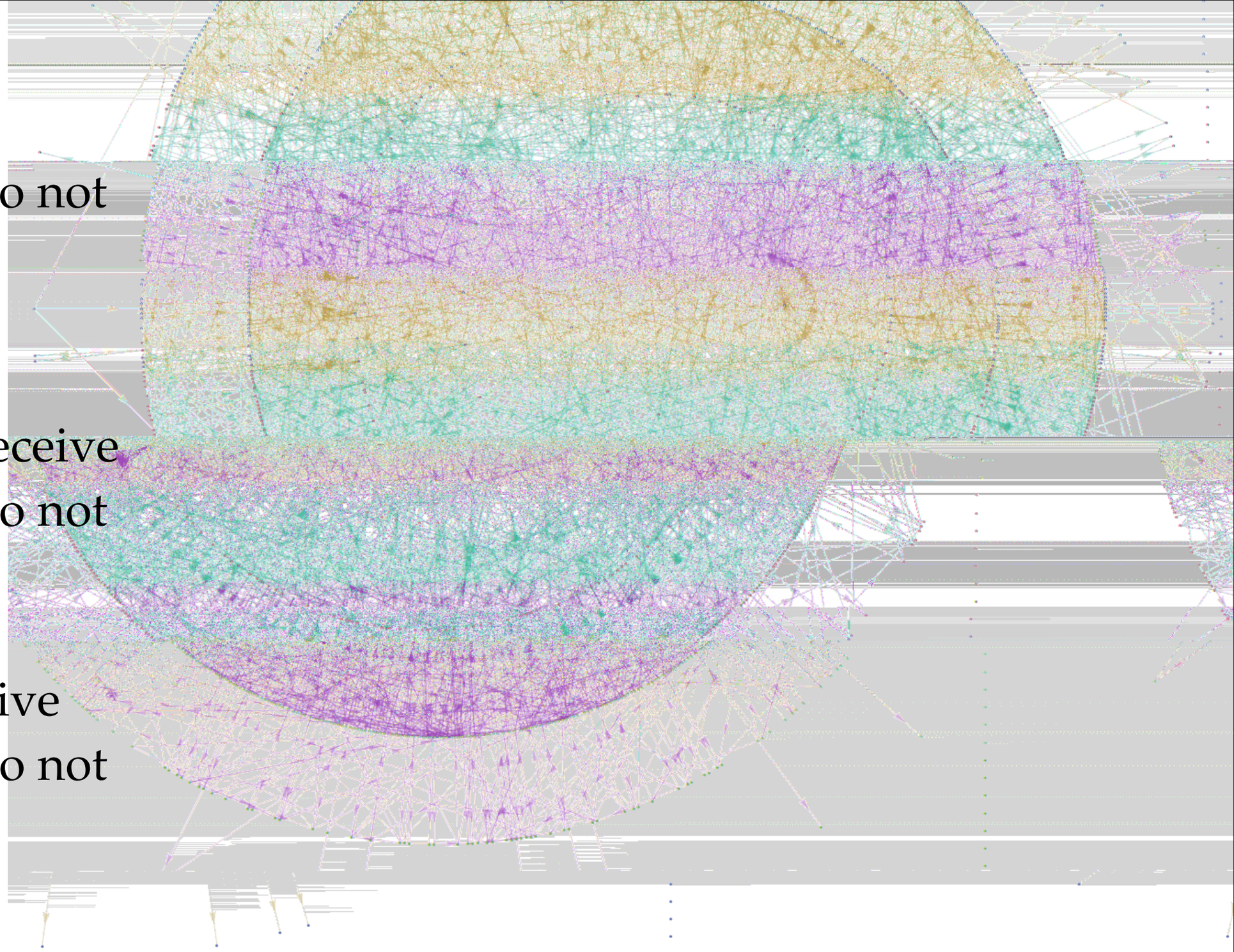




Student's Work

Program save timeline

- Some people do not give or receive feedback
- Some people receive feedback but do not give any
- Some people give feedback but do not receive any



Peer Review Relies on Participation

Image Convolution MP

Data Analysis Opportunities

SGEMM Implementation

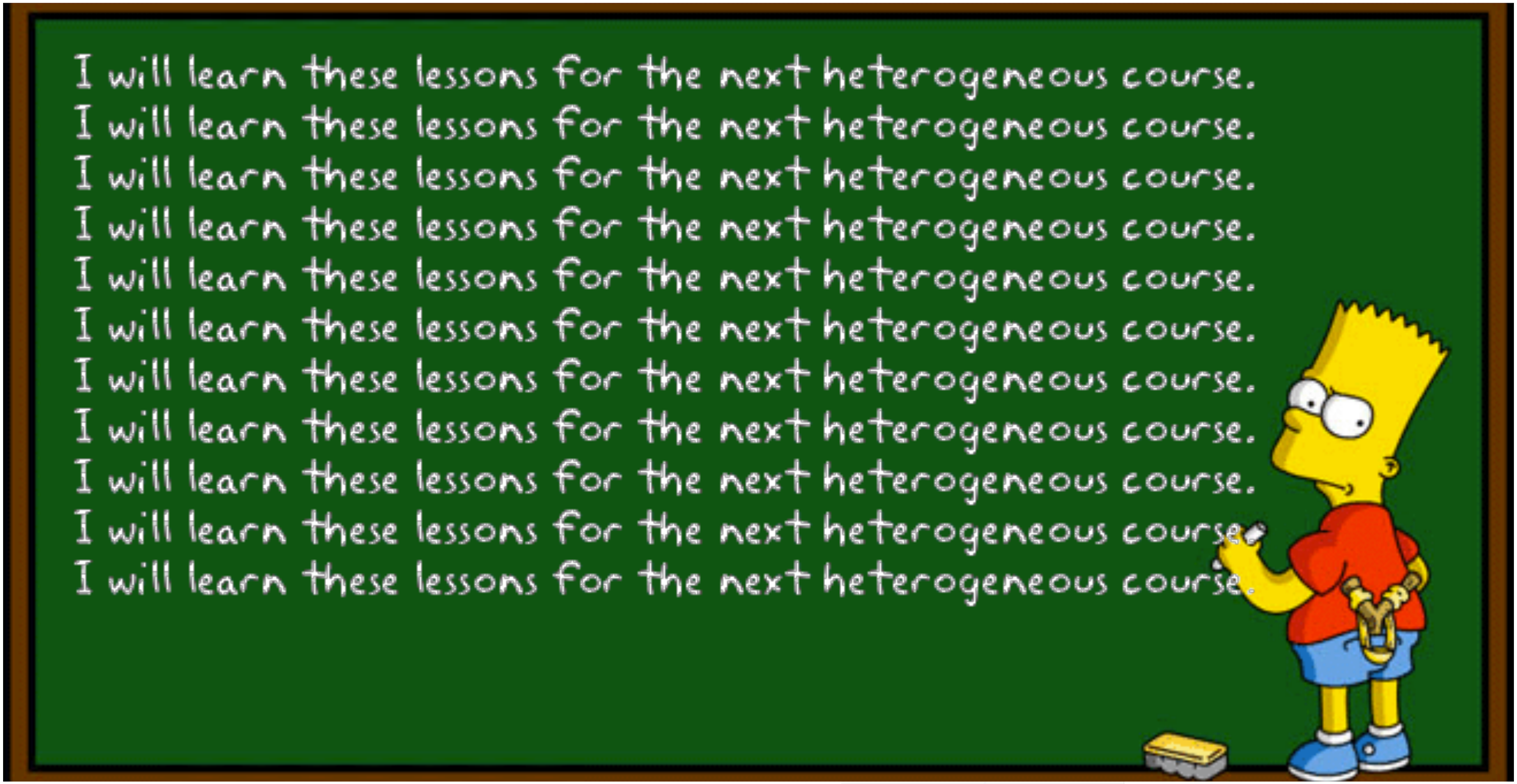
```
#define BLOCK_SIZE 16

__global__ void MatMulKernel(float *DA, float *DB, float *DC, int Ah, int Aw,
                             int AwTiles, int Bh, int Bw) {
  // Block row and column
  int blockRow = blockIdx.y;
  int blockCol = blockIdx.x;

  // Thread row and column within Csub
  int row = threadIdx.y;
  int col = threadIdx.x;

  int cellRow = blockRow * BLOCK_SIZE + row;
  int cellCol = blockCol * BLOCK_SIZE + col;

  // Each thread computes one element of Csub
  // by accumulating results into Cvalue
  float Cvalue = 0.0;

// Loop over all the sub-matrices of A and B that are
// required to compute Csub
// Multiply each pair of sub-matrices together
// and accumulate the results

#pragma unroll
  for (int m = 0; m < AwTiles; ++m) {

    // Shared memory used to store Asub and Bsub respectively
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    // Load Asub and Bsub from device memory to shared memory
    // Each thread loads one element of each sub-matrix
    int aRow = BLOCK_SIZE * blockRow + row;
    int aCol = BLOCK_SIZE * m + col;

    float aValue = ((aRow < Ah) && (aCol < Aw));
    aValue *= DA[Aw * aRow + aCol];
    As[row][col] = aValue;

    int bRow = BLOCK_SIZE * m + row;
    int bCol = BLOCK_SIZE * blockCol + col;

    float bValue = ((bRow < Bh) && (bCol < Bw));
    bValue *= DB[Bw * bRow + bCol];
    Bs[row][col] = bValue;

    // Synchronize to make sure the sub-matrices are loaded
    // before starting the computation
    __syncthreads();

    // Multiply Asub and Bsub together
    for (int e = 0; e < BLOCK_SIZE; ++e)
      Cvalue += As[row][e] * Bs[e][col];

    // Synchronize to make sure that the preceding
    // computation is done before loading two new
    // sub-matrices of A and B in the next iteration
    __syncthreads();
}

  if (cellRow >= Ah)
    return;
  if (cellCol >= Bw)
    return;

  DC[cellRow * Bw + cellCol] = Cvalue;
}

__global__ void nop() {}


//
//       CITE: Vasily Volkov, UC Berkeley.... Intresting approach to using
//caching a bit differently... had to give it a try...
//
__device__ void saxpy(float a, float *b, float *c) {
  c[0] += a * b[0];
  c[1] += a * b[1];
  c[2] += a * b[2];
  c[3] += a * b[3];
  c[4] += a * b[4];
  c[5] += a * b[5];
  c[6] += a * b[6];
  c[7] += a * b[7];
  c[8] += a * b[8];
  c[9] += a * b[9];
  c[10] += a * b[10];
  c[11] += a * b[11];
  c[12] += a * b[12];
  c[13] += a * b[13];
  c[14] += a * b[14];
  c[15] += a * b[15];
}

__global__ void optimisedDLA(const float *A, int lda, const float *B, int ldb,
                             float *C, int ldc, int k) {
  const int inx = threadIdx.x;
  const int iny = threadIdx.y;
  const int ibx = blockIdx.x * 64;
  const int iby = blockIdx.y * 16;
  const int id = inx + iny * 16;

  A += ibx + id;
  B += inx + __mul24(iby + iny, ldb);
  C += ibx + id + __mul24(iby, ldc);

  const float *Blast = B + k;

  float c[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

  do {
    float a[4] = {A[0 * lda], A[1 * lda], A[2 * lda], A[3 * lda]};

    __shared__ float bs[16][17];
    bs[inx][iny] = B[0 * ldb];
    bs[inx][iny + 4] = B[4 * ldb];
    bs[inx][iny + 8] = B[8 * ldb];
    bs[inx][iny + 12] = B[12 * ldb];
    __syncthreads();

    A += 4 * lda;
    saxpy(a[0], &bs[0][0], c);
    a[0] = A[0 * lda];
    saxpy(a[1], &bs[1][0], c);
    a[1] = A[1 * lda];
    saxpy(a[2], &bs[2][0], c);
    a[2] = A[2 * lda];
    saxpy(a[3], &bs[3][0], c);
    a[3] = A[3 * lda];

    A += 4 * lda;

    saxpy(a[0], &bs[4][0], c);
    a[0] = A[0 * lda];
    saxpy(a[1], &bs[5][0], c);
    a[1] = A[1 * lda];
    saxpy(a[2], &bs[6][0], c);
    a[2] = A[2 * lda];
    saxpy(a[3], &bs[7][0], c);
    a[3] = A[3 * lda];

    A += 4 * lda;
    saxpy(a[0], &bs[8][0], c);
    a[0] = A[0 * lda];
    saxpy(a[1], &bs[9][0], c);
    a[1] = A[1 * lda];
    saxpy(a[2], &bs[10][0], c);

    a[2] = A[2 * lda];
    saxpy(a[3], &bs[11][0], c);
    a[3] = A[3 * lda];

    A += 4 * lda;
    saxpy(a[0], &bs[12][0], c);
    saxpy(a[1], &bs[13][0], c);
    saxpy(a[2], &bs[14][0], c);
    saxpy(a[3], &bs[15][0], c);

    B += 16;
    __syncthreads();
  } while (B < Blast);

  for (int i = 0; i < 16; i++, C += ldc)
    C[0] = c[i];
}
….
```

Analysis Opportunities

- What errors are most common?
  - Can we detect those and give feedback
- What optimizations are most beneficial?
  - Which did users have the most problems with?
- What causes GPUs to crash?
  - Can we avoid those?
- Did people plagiarize?
- What does the peer review tell us?

Source of Data

- This is a big data problem
- Different analysis can be performed on different parts
- Program analysis on programs
- Power analysis on recorded GPU power draw
- NLP on questions and peer review



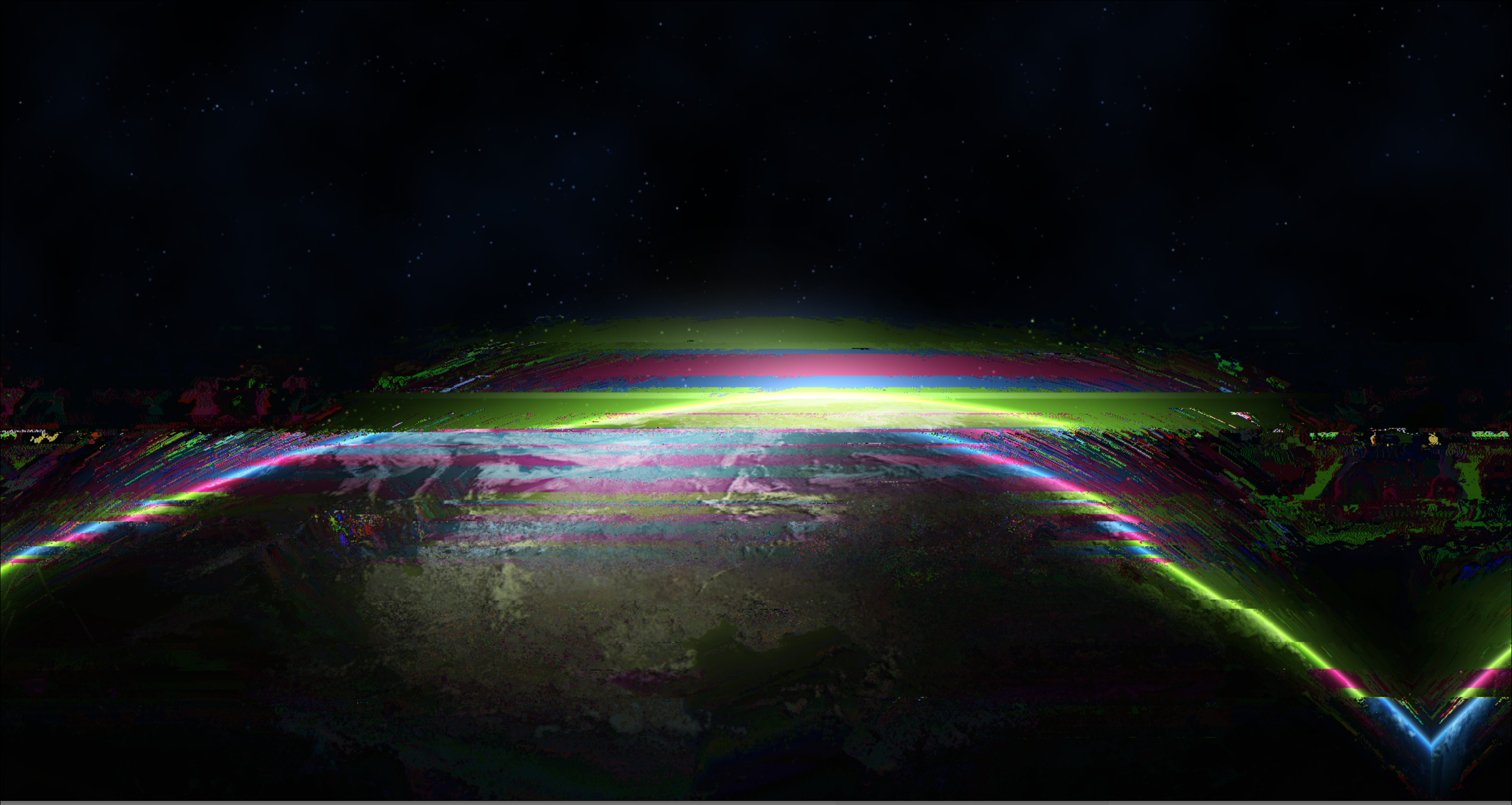
What would be different?

Lessons Learned

- Volunteer TAs were very helpful and allowed me to spend less than 30 hours a day on the forums

- Not everyone will perform peer reviews, which means that not everyone will get feedback

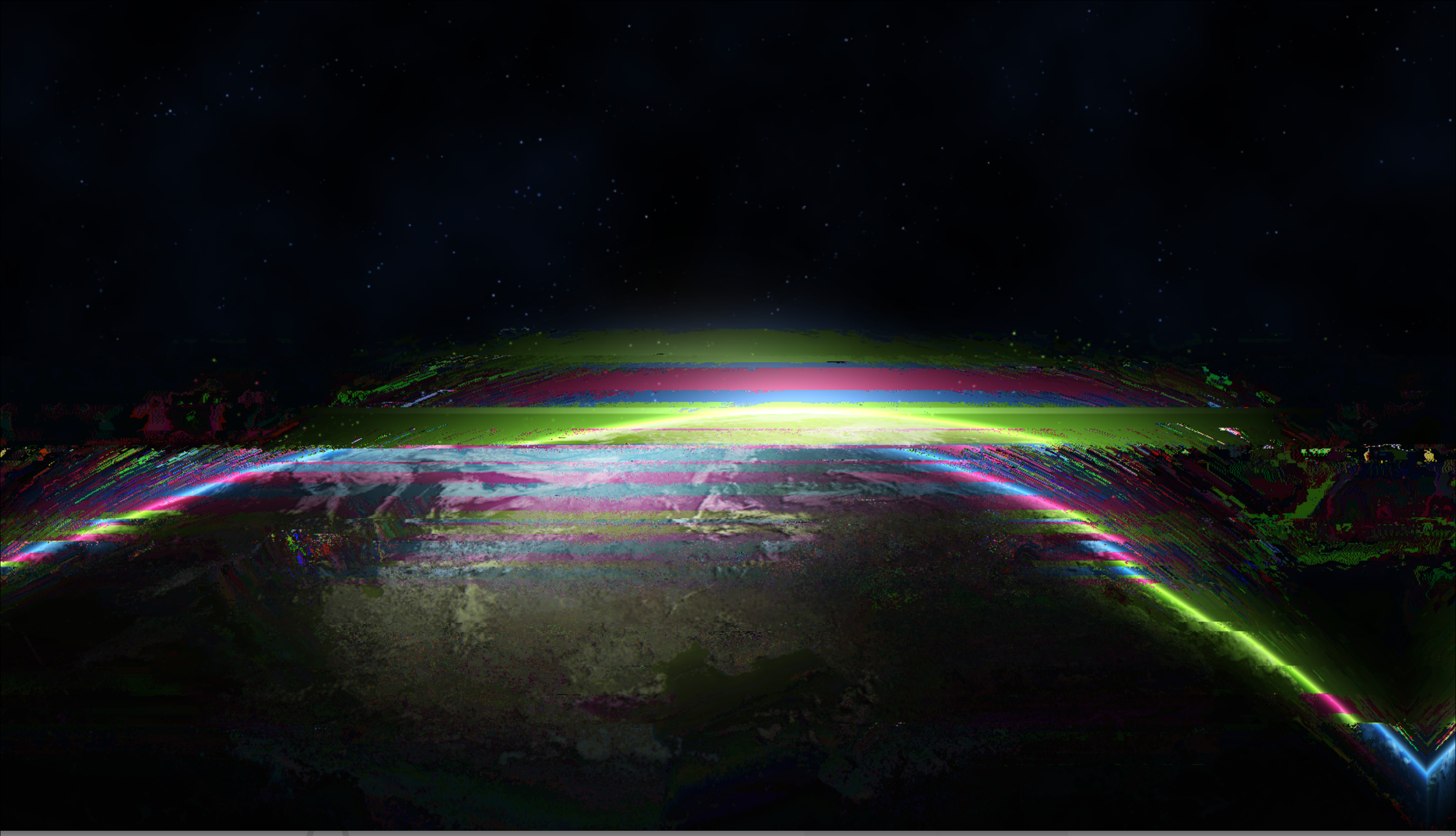
- Some people just criticize for the sake of it … you need to develop a thicker shell



Current Work

Current Work

- Build up some tools for data analysis
- Course work projects
- Porting Parboil to threaded Java and Renderscript
- ZOne — a compiler/language to explore Map/Reduce compiler optimizations
- Otpmization — Machine learning + Vision applications



Questions?