Simulation Ideen-Verbreitung

Projektvorstellung

Arne Struck, Jonathan Werner

Universität Hamburg, Fachschaft Informatik, Praktikum paralleles Programmieren

24. September 2014

Ziel

(Grobe) Simulation von Entwicklung konkurrierender Ideen in einer begrenzten Welt.

Population

Idee

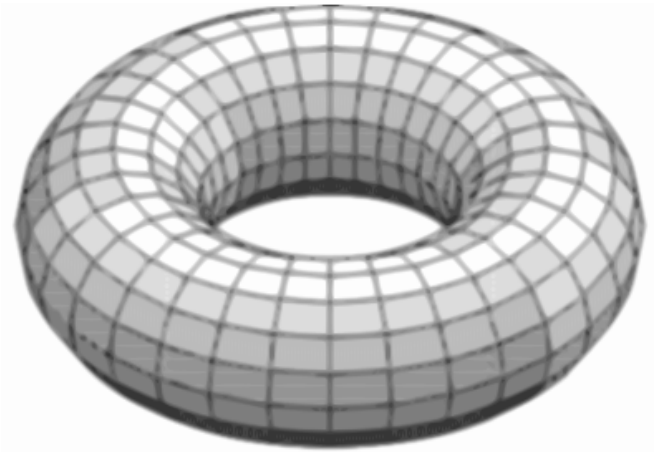
- Qualität
- Komplexität
- Weltanschauung

Mensch

- Idee
- Weltanschauung

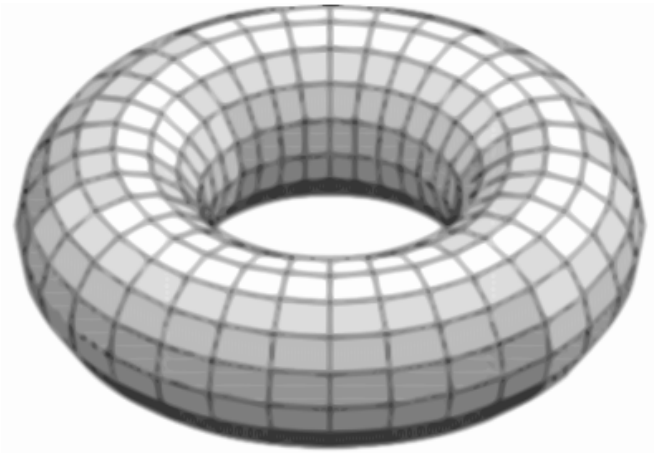
Welt & Bewegung

Die Welt

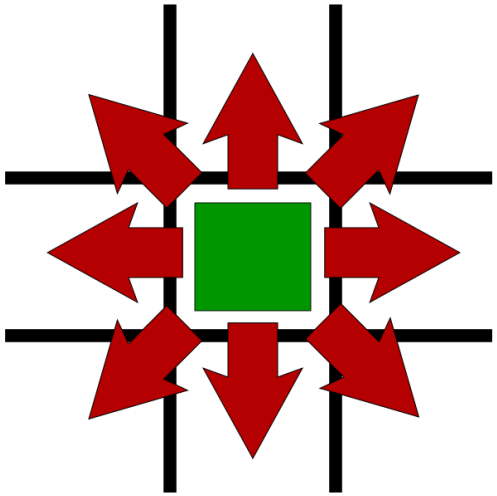


Welt & Bewegung

Die Welt

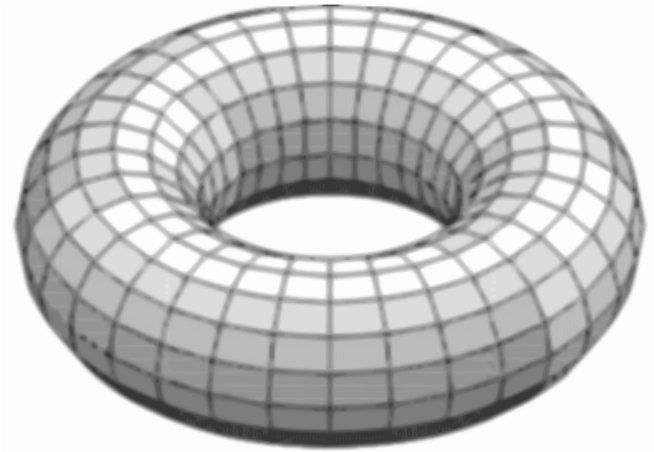


Bewegungsziele

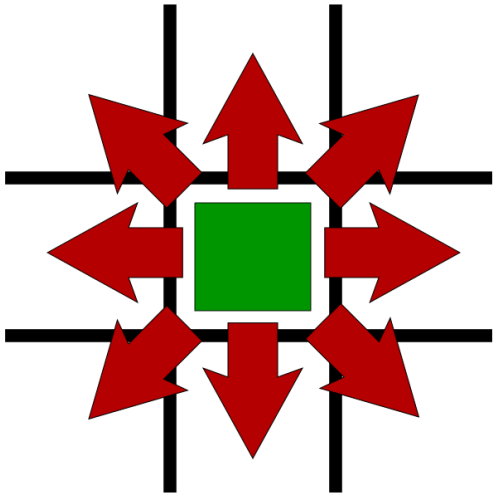


Welt & Bewegung

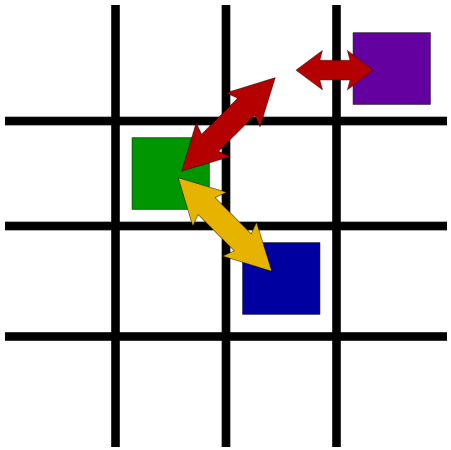
Die Welt



Bewegungsziele



Kommunikation



Kommunikation

3-Phasen:

1. Kompatibilitätscheck
2. Evaluation des Gewinners
3. Aufstellung der neuen Merkmale des Verlierers

Mutation

Qualität

- Wahl der Mutationsrichtung
- Mutation der Qualität
- Kaskadierend hierdurch Mutation der Komplexität

Mutation

Qualität

- Wahl der Mutationsrichtung
- Mutation der Qualität
- Kaskadierend hierdurch Mutation der Komplexität

Weltanschauung

- Wahl der Mutationsrichtung
- Mutation des Idee-Wertes
- Mutation des Mensch-Wertes
- Differenzcheck

Ablauf

- Initialisierung des Feldes
- Zufälliger Spawn der Menschen mit mehrheitlich geringen Qualitätswerten
- Beginn der Simulationsschleife für n Schritte
  - Mutationsevaluation
  - Kommunikationsversuch
  - Bewegung
- Ende der Schleife

Implementation Idee

```
typedef struct {
  int quali, complexity, wordview, human_wordview,
      empty;
} Idea;
```

Implementation Feld

```
#define malloc_idea_matrix(name) \
   Idea **name =
     (Idea **)malloc(num_rows * sizeof(Idea *)); \
   for (int i = 0; i < num_rows; ++i) \
       name[i] =
         (Idea *)malloc(num_cols * sizeof(Idea));\

...

malloc_idea_matrix(field)
malloc_idea_matrix(field_new)
```

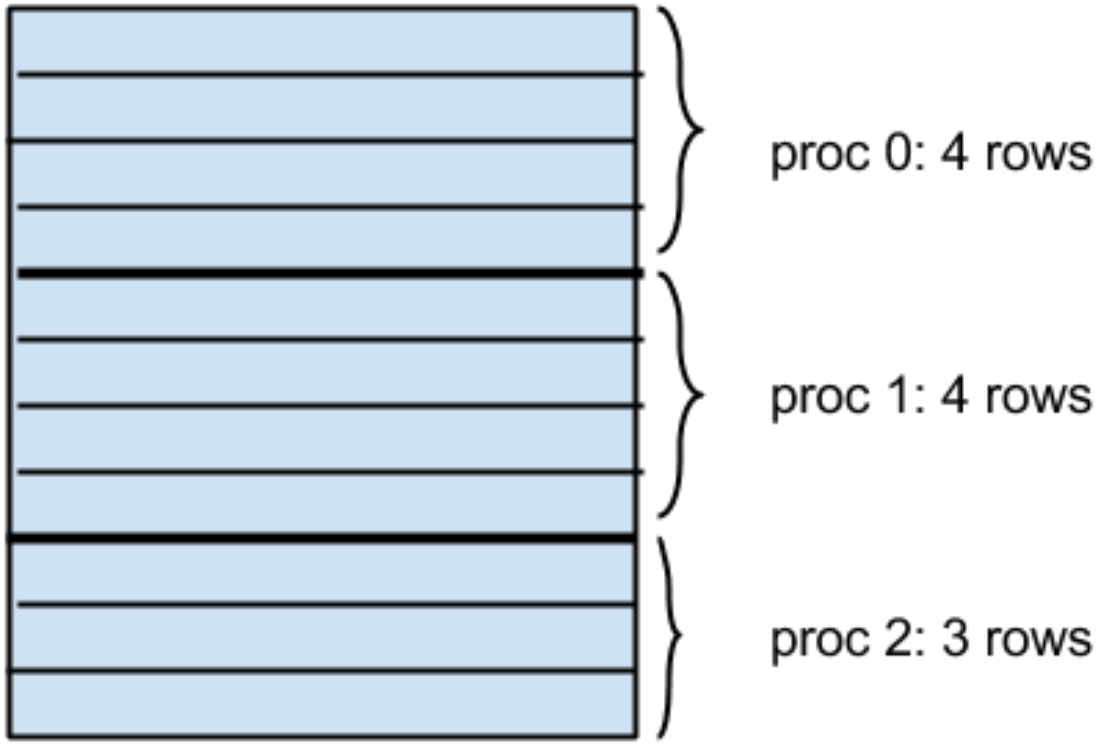
field und field_new

- Beginn der Runde: field und field_new gleicher Inhalt
- Iterieren über field
- Bewegte Ideen werden in field_new geschrieben
- Modifizierte Ideen werden in field und field_new geschrieben
- Ende der Runde: field_new in field kopieren

Implementation Kopieren

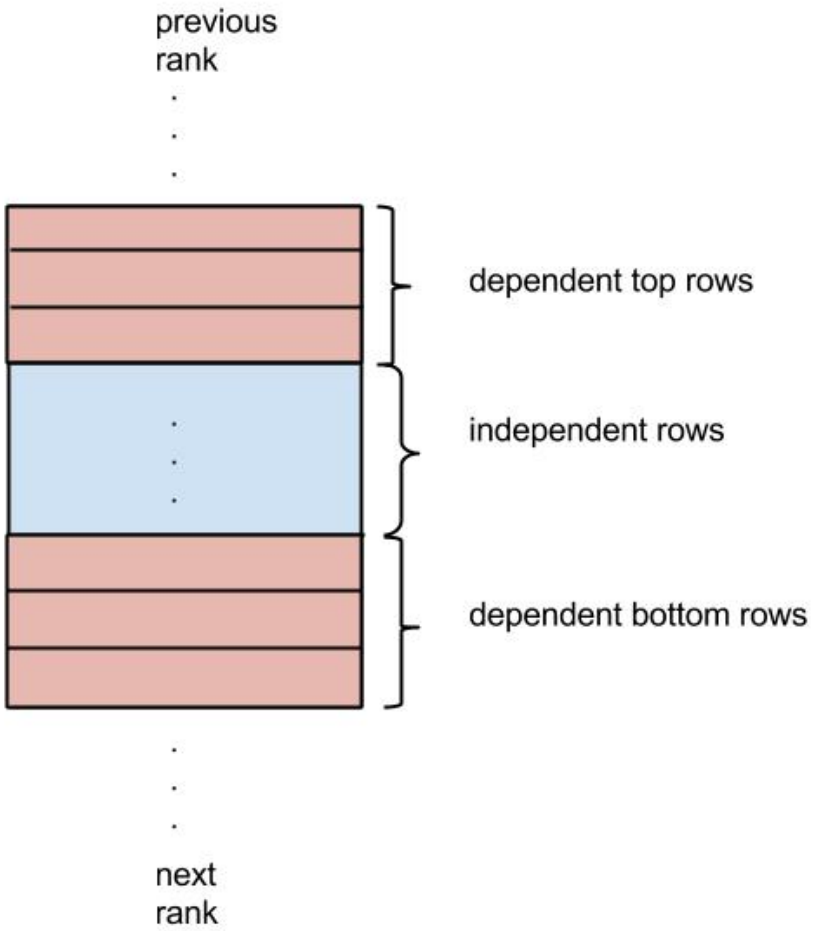
```
#define for_every(i, size, f)
  for(int i=0; i<size; i++) { f; }

...

#define copy_field_new_into_field()      \
  for_every(i, num_rows, {               \
    for_every(j, num_cols, {             \
      field[i][j] = field_new[i][j];     \
    });                                  \
  });
```

Aufteilung Feld auf Prozesse



Kommunikation über "ghost rows"

Parallelisierungsschema



1. Bewegung der independent ideas
2. Bewegung der top dependent ideas, Kommunikation dieser
3. Bewegung der bottom dependent ideas, Kommunikation dieser

Parallelisierungsschema

```
#define send_ideas(ideas_arr, to, tag, req)
  MPI_Isend(ideas_arr, num_cols,
    mpi_idea_type, to, tag, MPI_COMM_WORLD, &req)

#define receive_ideas_into(ideas_arr, from, tag, req)
  MPI_Irecv(ideas_arr, num_cols,
    mpi_idea_type, from, tag, MPI_COMM_WORLD, &req)
```

Parallelisierungsschema

```
#define mpi_define_idea_type()
   int blocklengths[5] = {1,1,1,1,1};
   MPI_Datatype types[5] = {MPI_INT,MPI_INT,MPI_INT,
       MPI_INT, MPI_INT};
   MPI_Datatype mpi_idea_type;
   MPI_Aint offsets[5];
   offsets[0] = offsetof(Idea, a);
   offsets[1] = offsetof(Idea, b);
   offsets[2] = offsetof(Idea, c);
   offsets[3] = offsetof(Idea, h);
   offsets[4] = offsetof(Idea, empty);
     MPI_Type_create_struct(5, blocklengths, offsets
         , types, &mpi_idea_type);
     MPI_Type_commit(&mpi_idea_type);
```

MPI-Code Überblick I

```
// INDEPENDENT ROWS
if (num_rows >= 7) move_ideas(2, num_rows-5);
barrier();

// DEPENDENT ROWS TOP
move_ideas(0, 3);
barrier();

send_top_rows(field_new);
receive_into_bottom_rows(field_new);
barrier();
```

MPI-Code Überblick II

```
// DEPENDENT ROWS BOTTOM
move_ideas(num_rows - 4, 3);
barrier();

send_bottom_rows(field_new);
receive_into_top_rows(field_new);
barrier();

copy_field_new_into_field();
```

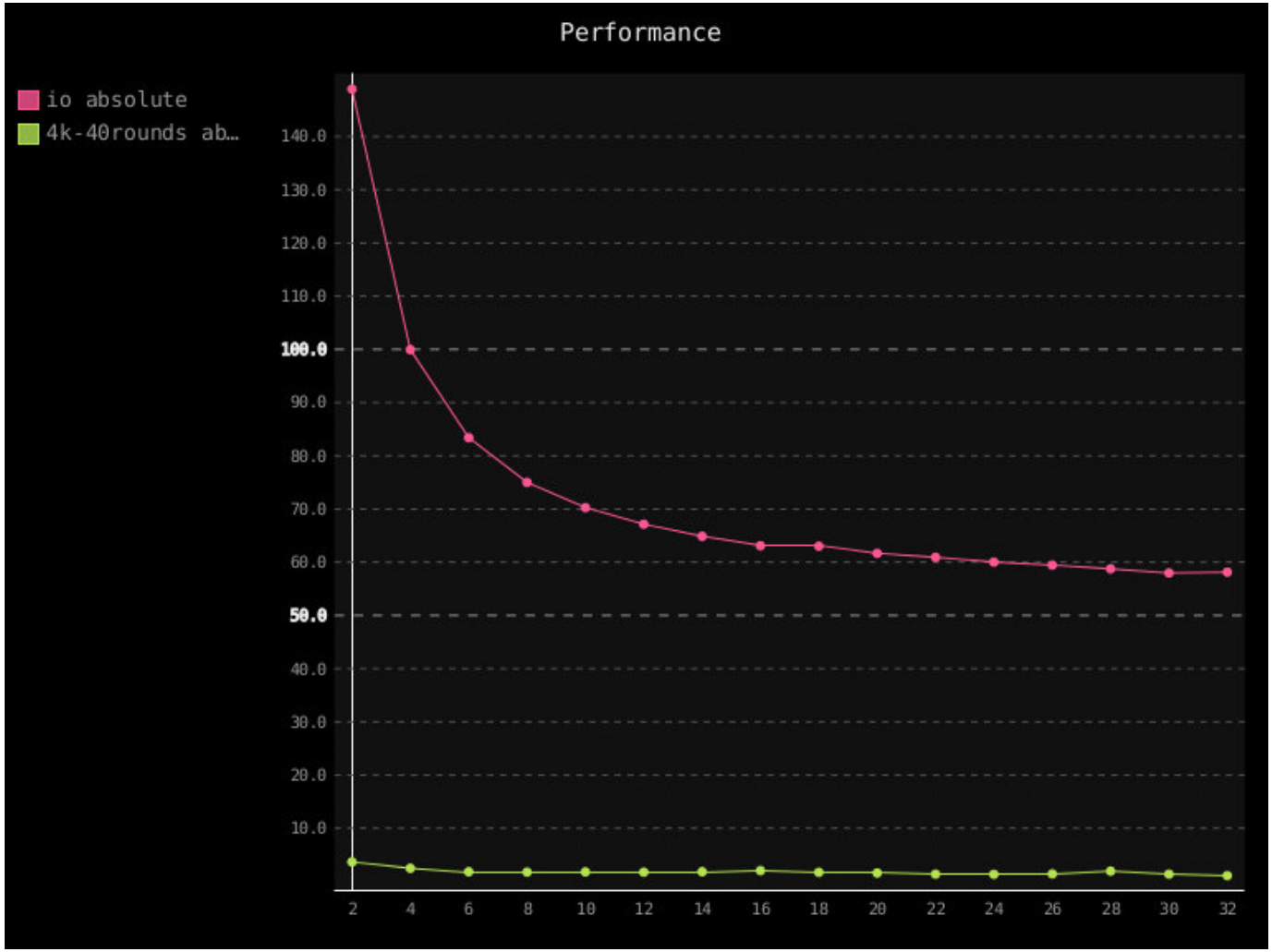
Implementation Visualisierung

- lokal mit Python/Pygame
- Output von C pro rank in out/$round-$rank files
- im Nachhinein: Rundenweise Einlesen der Files im Pygame-Loop
- Problem mit Integration des Clusters: rsync bottleneck

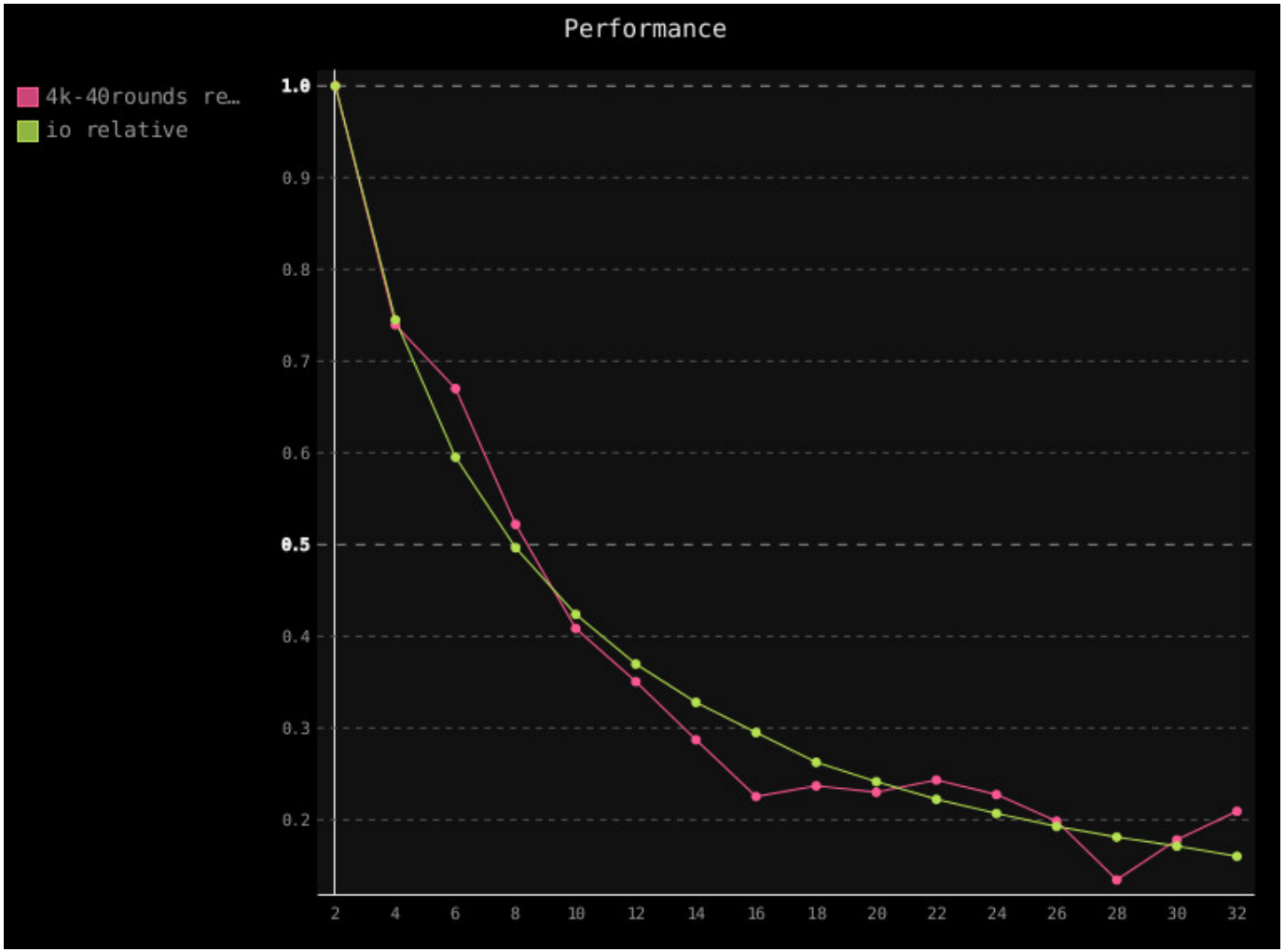
Optimierung

- Ersetzen von Send/Recv mit Isend/Recv (10 % Speedup)
- field / field_new Kopieren per memcpy: fail
- Kommunikationsmuster: erst: nur gerade # ranks erlaubt, da Abfolge war: (1) alle gerade ranks senden alles, ungerade ranks empfangen alles; (2) vice versa
- Generell: gefühlte Fragilitaet von MPI Code bewog zu defensivem Verhalten - bloß nichts mehr kaputt machen (Beispiel: lieber mehr als zuwenig Barriers)
- Motivationsmangel: Performance von C eh mehr als ausreichend, Bottleneck bei Visualisierung und rsyncing vom Cluster

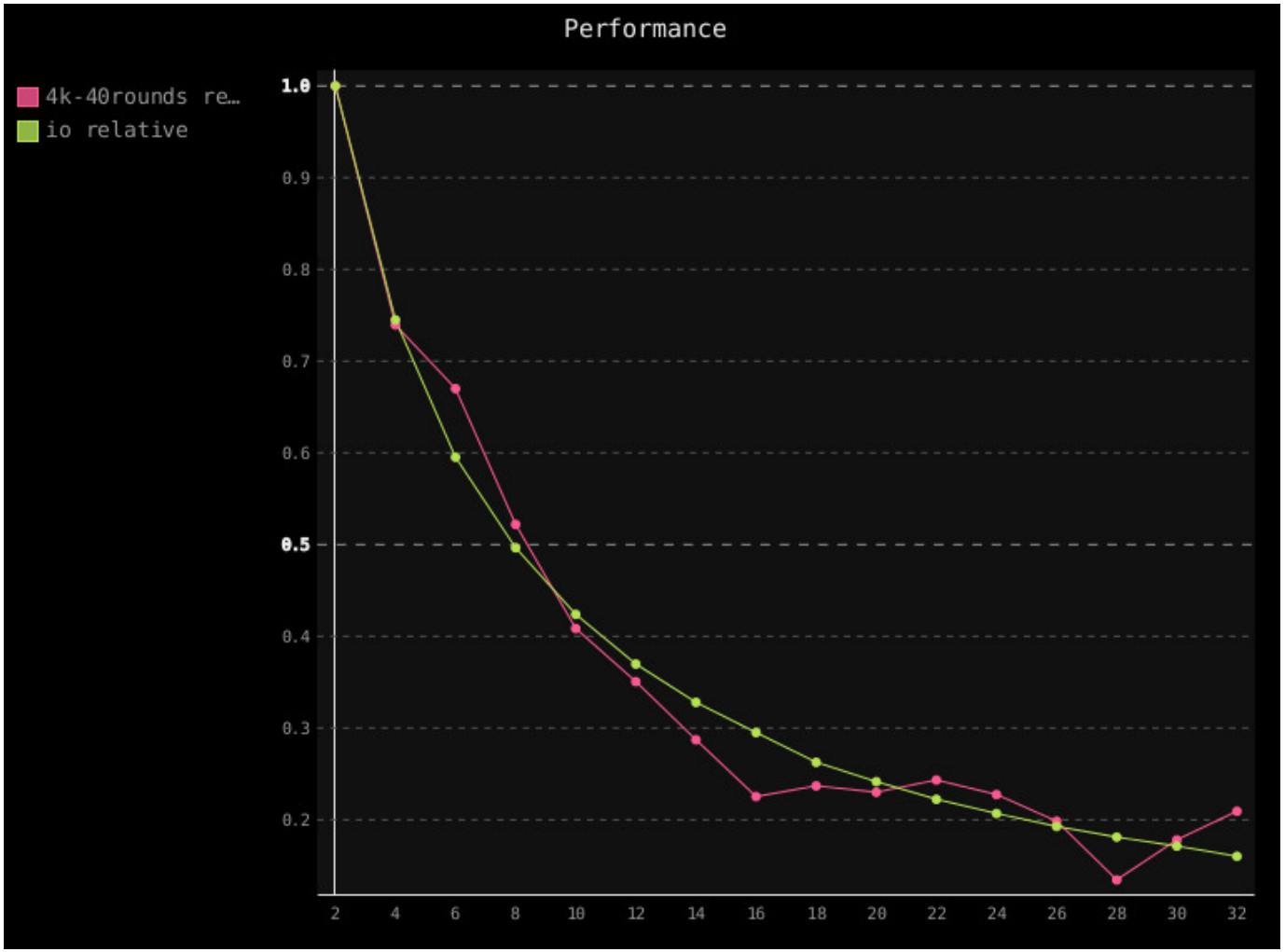
Profiling IO vs no IO absolut



Performance IO vs no IO relativ

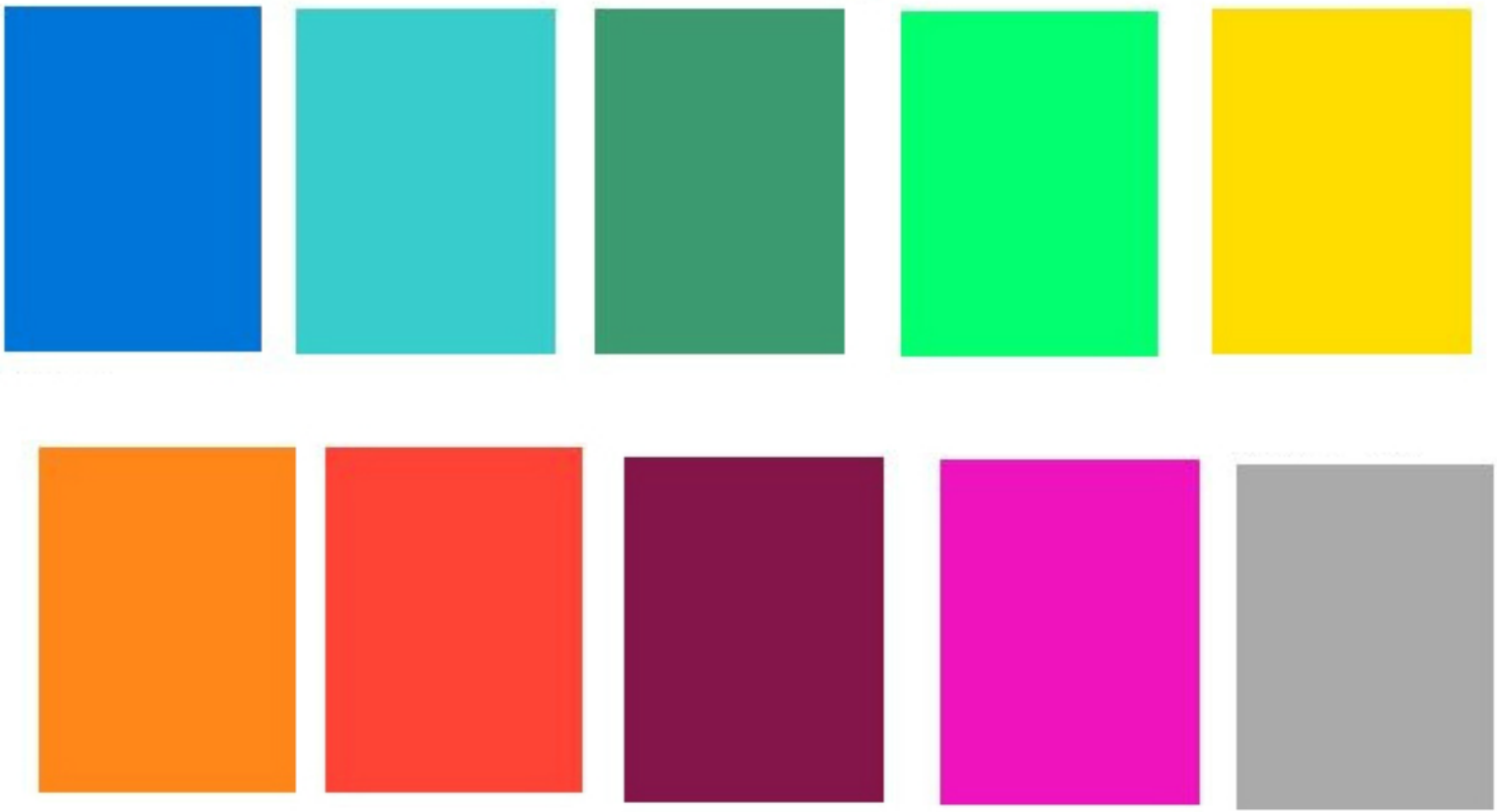


Performance IO vs no IO relativ



Performance no IO absolut

Ideen WV Farbcodierung



Demo

Inhaltliche Erkenntnisse

- Qualität nimmt über die Zeit zu
- Obwohl andere selten vollständig entfernt, bilden 2-3 Ideen eine Majorität aus
- Komplexitätsgrad nimmt über die Zeit zu
- Es bleiben einige Menschen mit Ideen niedriger Qualität
- Selten: Durch Mutation entwickelt sich eine verdrängte Idee zur dominanten

Probleme

Probleme beim Debuggen

- Logik und Bewegung größtenteils unter Beteiligung von Zufallselementen
- oft nicht reproduzierbare Bugs

Real-Time Visualisierung

- große Datenmenge
- Uns war nicht klar wie/ob X-Forwarding mit dem Cluster funktioniert wird