Praktikum Parallele Programmierung, Sommer 2014

Genetischer Algorithmus
entwirft
Tower Defense Maze

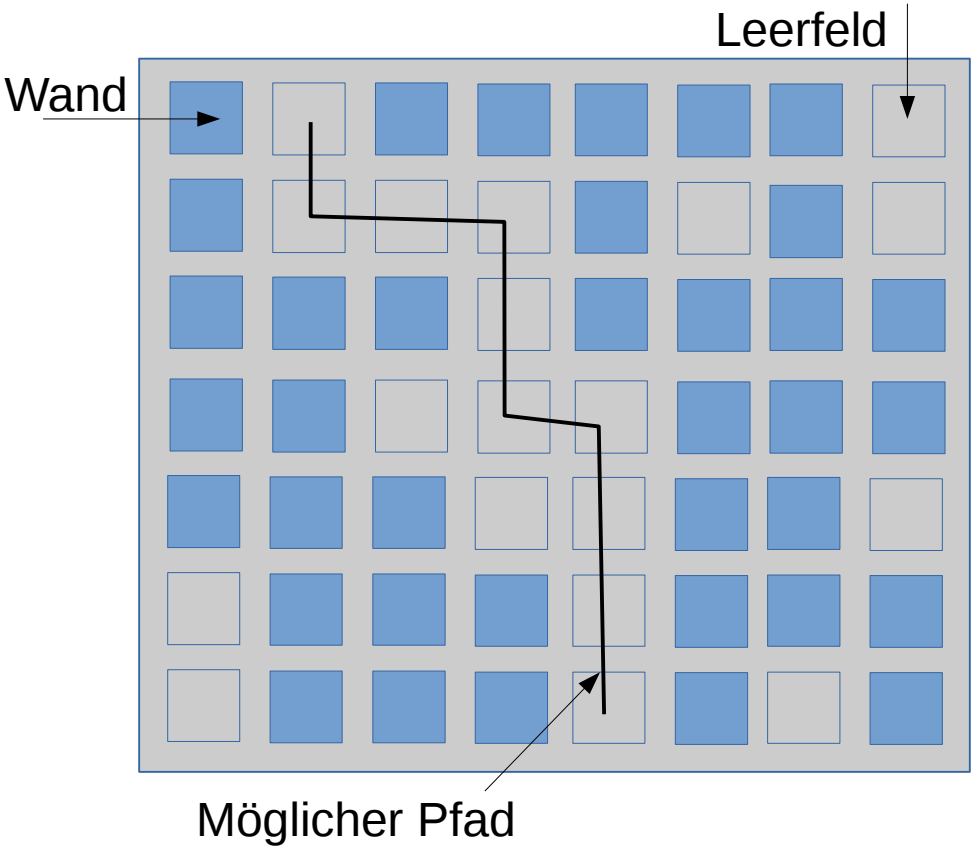
Author: Sascha Schulz
Universität Hamburg, Matr-Nr.: 6434677, 2sschulz@informatik.uni-hamburg.de

Idee

- Triviales Problem verwenden, um mit dem Genetischen Algorithmus zu experimentieren
- Den Funmaps von Warcraft III entliehen: Konstruktion eines Mazes, sodass Kreaturen möglichst lange brauchen von oben nach unten zu gelangen

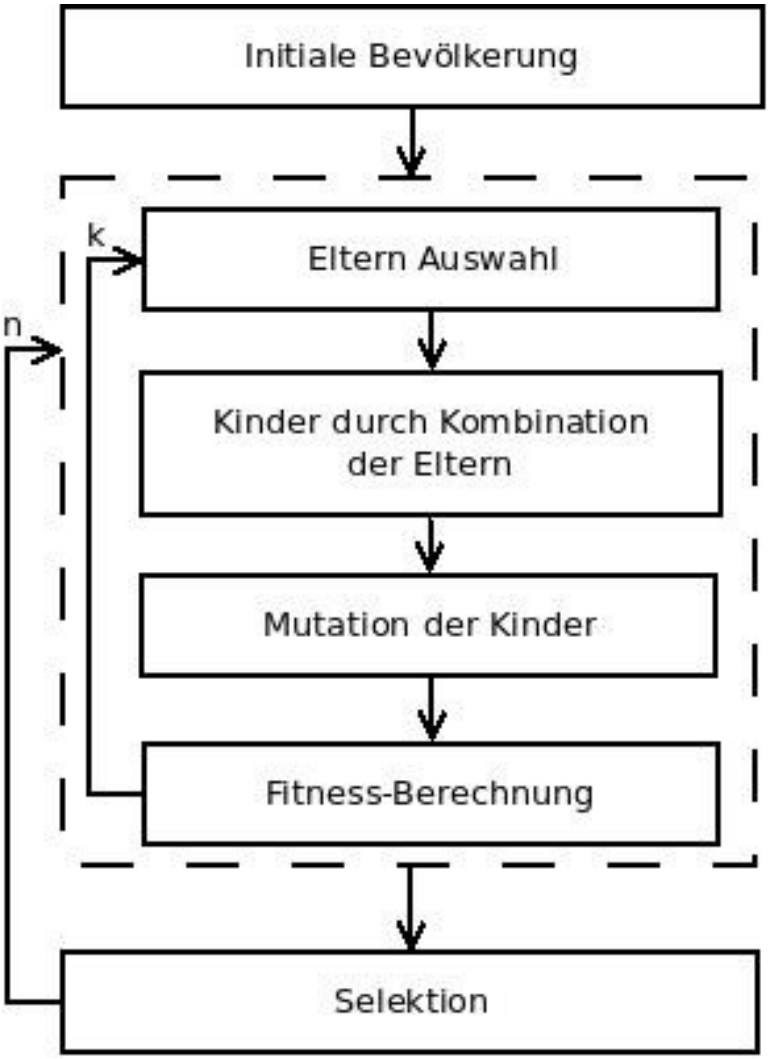
Modell

- Spielfläche rastern und auf Wände/Leerfelder reduzieren.
- Maze in Graphen umwandeln: Freies Feld zu Vertex, Nachbarn durch Kanten verbinden
- Graph nutzen, um kürzesten Weg zu suchen



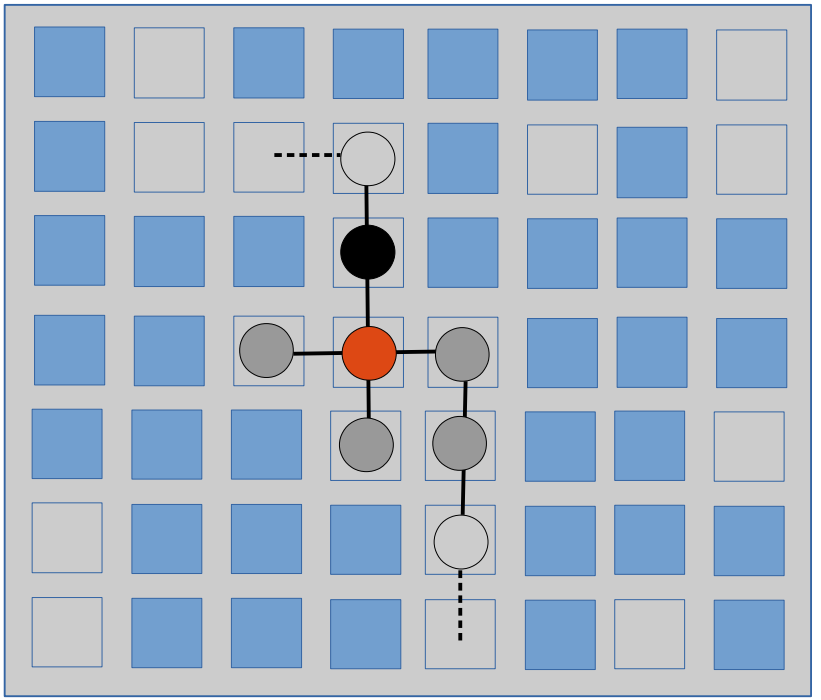
Genetischer Algorithmus

- k Elternpaare pro Iteration
- n Iterationen insgesamt

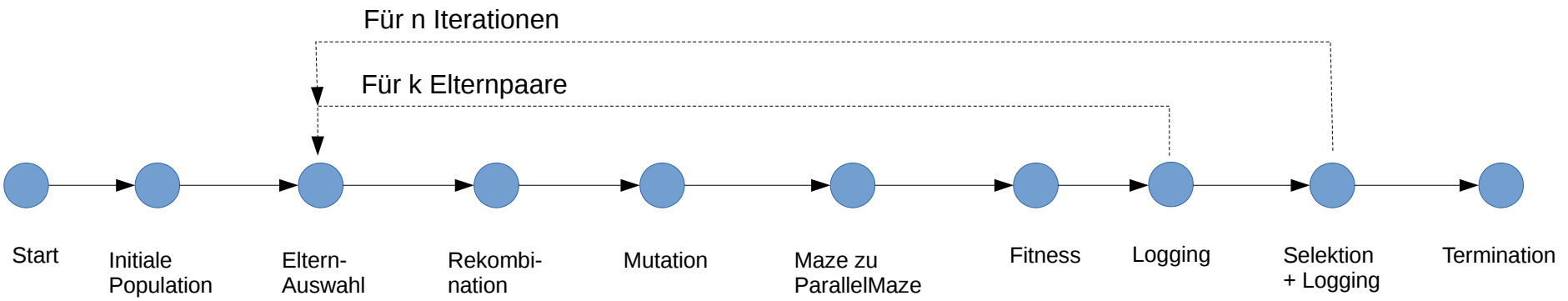


Kürzester Weg: Bellmann-Ford

- Für jeden Knoten alle Nachbarn prüfen, ob darüber kürzester Weg möglich
- Nach max |G(V)| Schritten ist kürzester Weg gefunden
- Lässt sich gut parallelisieren

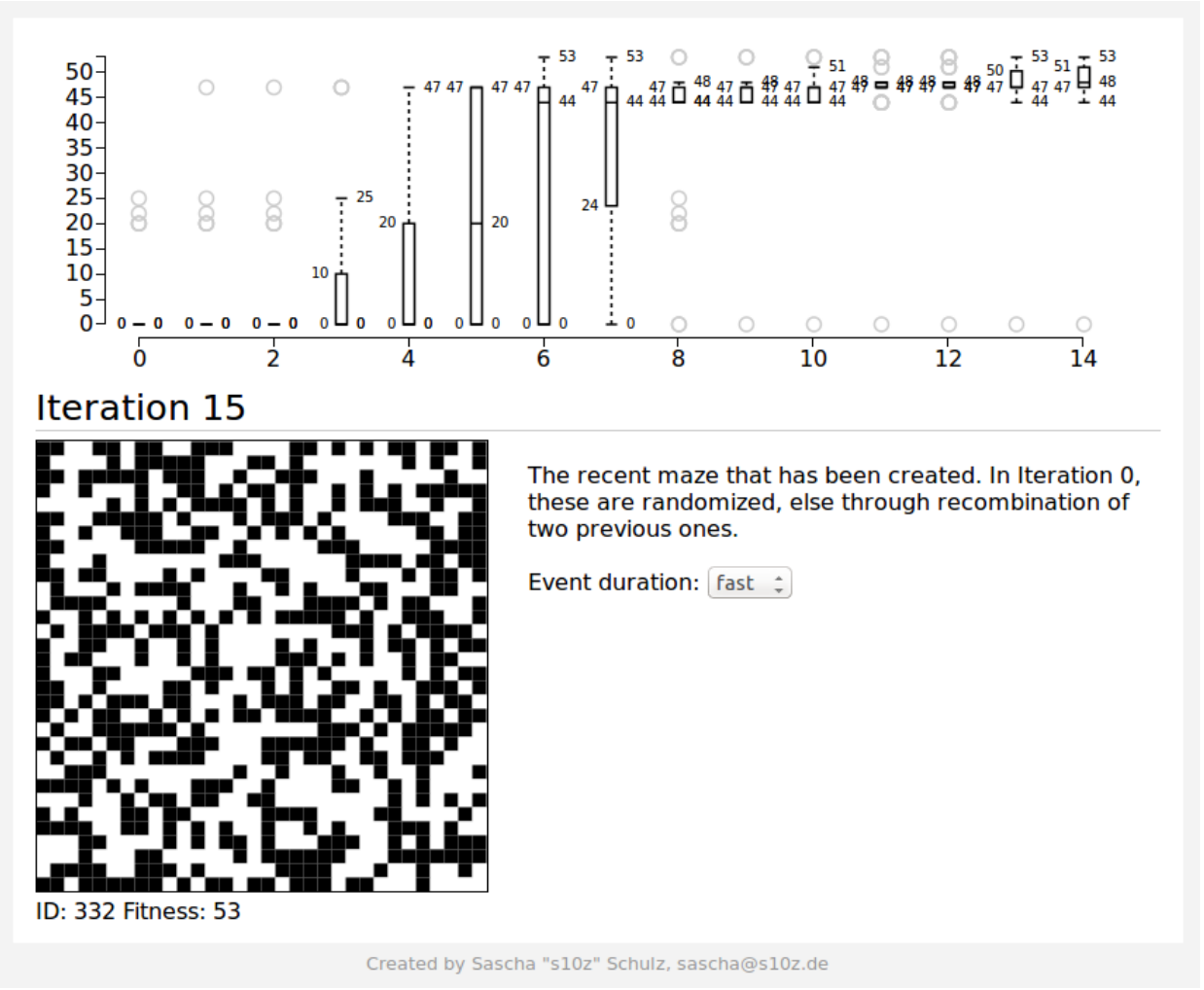


Programmablauf



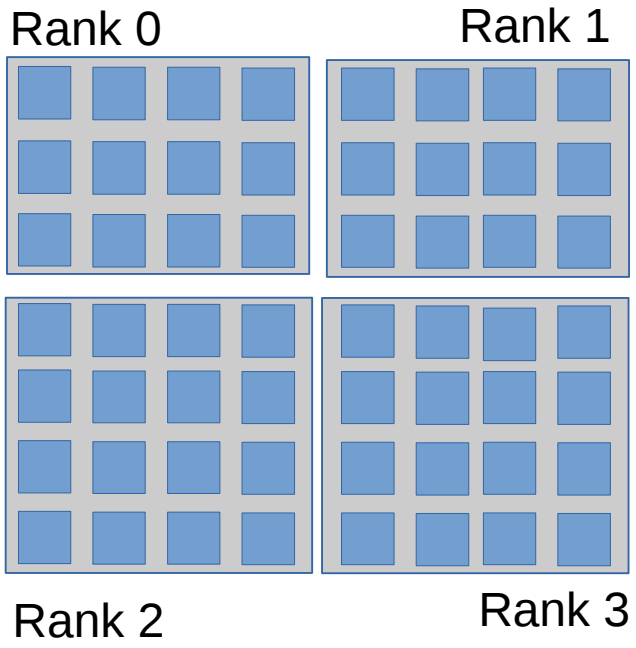
User Interface

- Event Log von JavaScript interpretiert

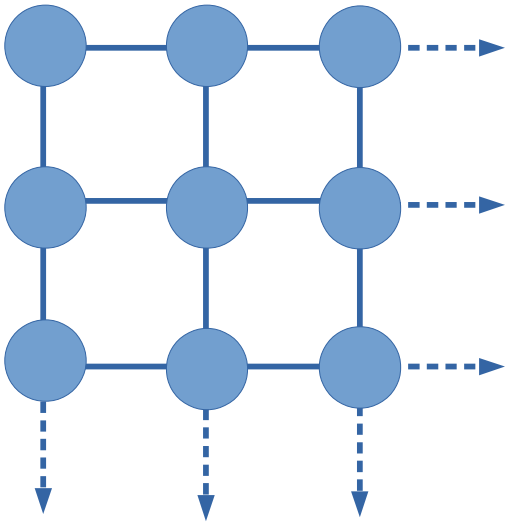


Parallelisierungsschema (MPI)

- Initial geplant:



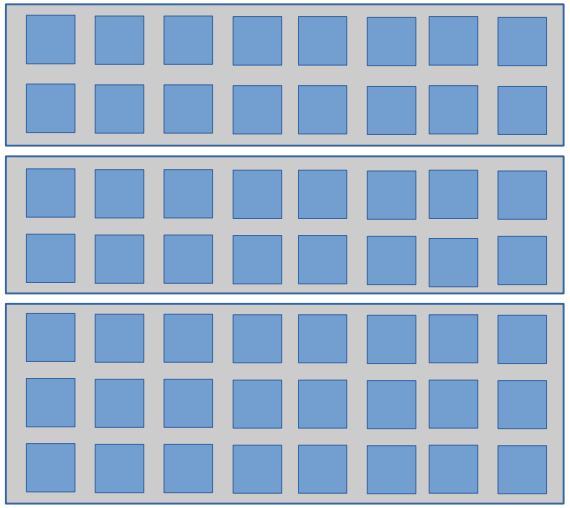
- Abstrakte Darstellung:



=> Viele Kommunikationsgrenzen, komplizierter pro Rank zu wissen, auf welchen globalen Feldern operiert wird

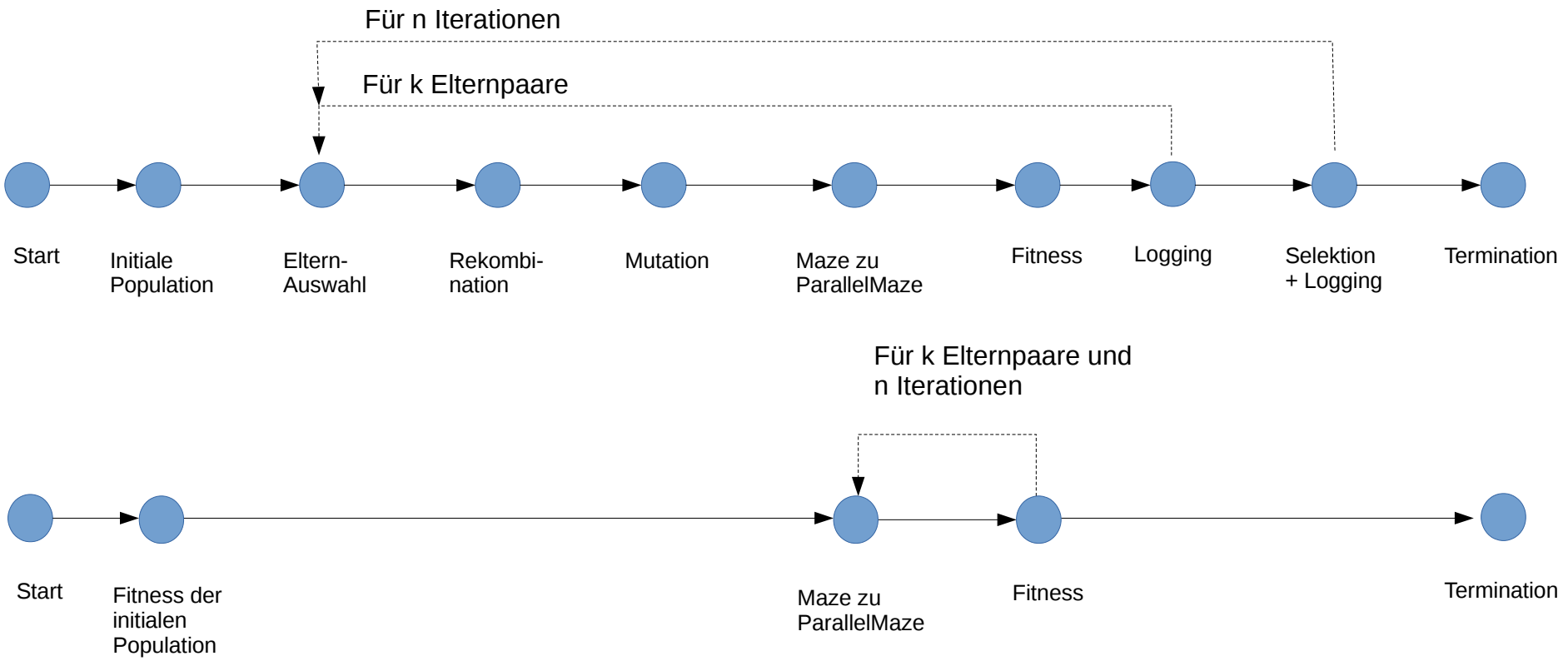
Parallelisierungsschema (MPI)

- Vereinfacht:



- Abstrakte Darstellung:

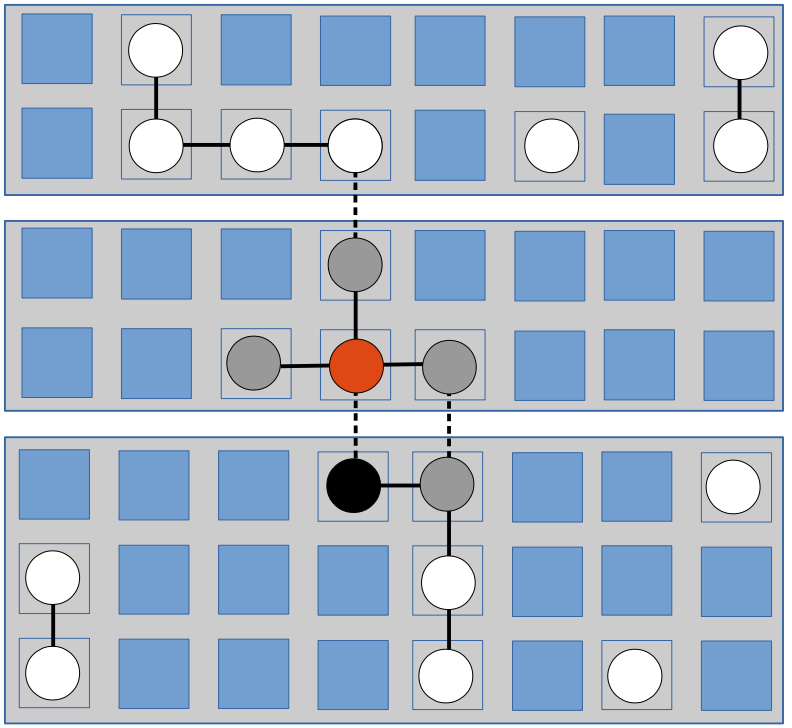
Master/Slave



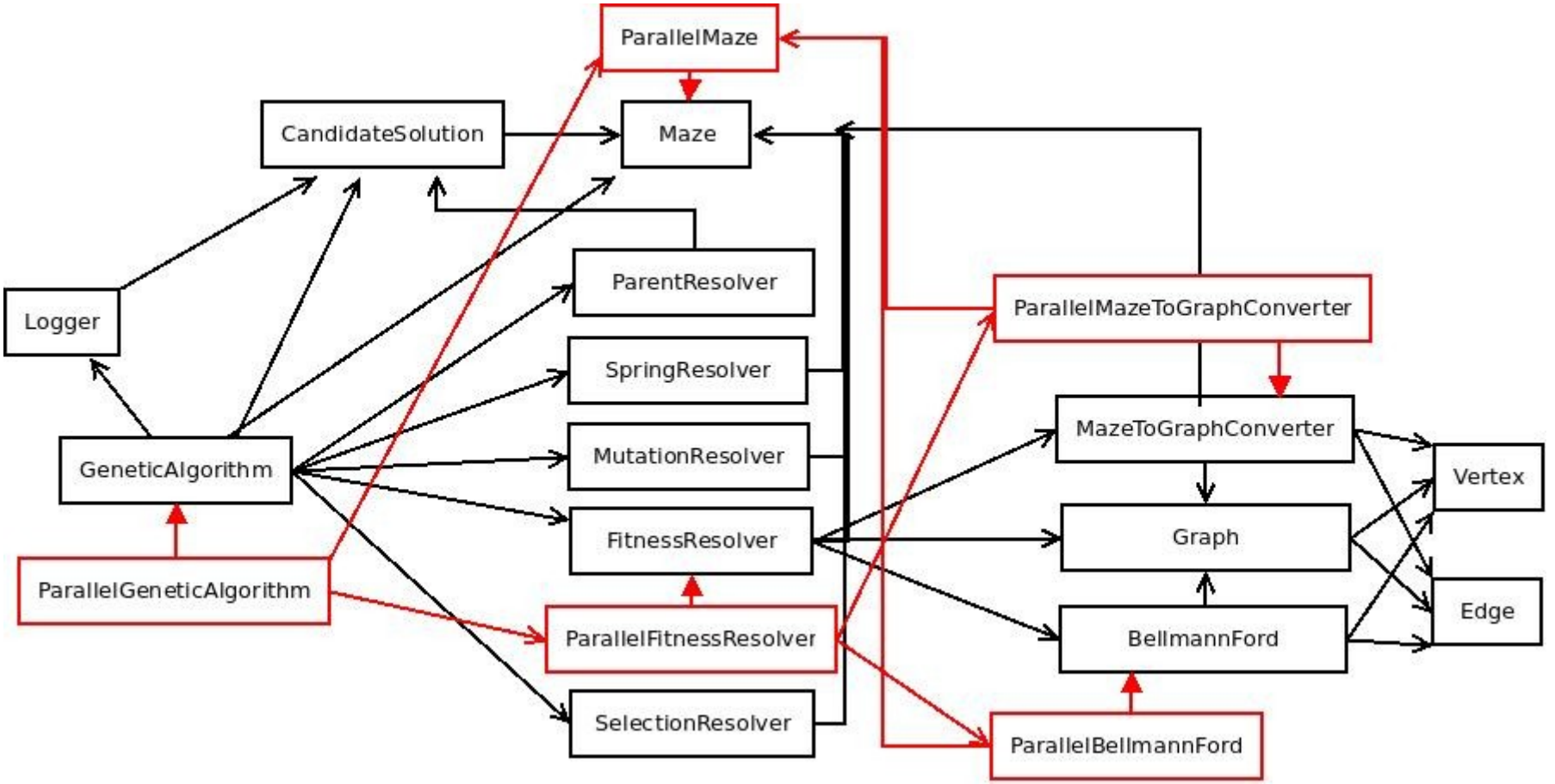
Aufgaben des Masters (oben) gegenüber des Slaves (unten)

Kommunikation:

- Maze zu Graph: Knoten erhalten globale Feldnummer als ID.
- Kanten über Prozessgrenzen = Kanten ins Nichts
- Bei Bellmann-Ford dann jedoch Adjazenz bekannt (orange ↔ schwarz)



Parallelität als eigener Belang



Fazit: Hilft zur Übersicht, aber stört bei der Optimierung

(Parallele) Tests mit gtest

```
TEST(ParallelMazeToGraphConverter, zigzagVerticesCreated)
{
   int size = MPI::COMM_WORLD.Get_size();
   int rank = MPI::COMM_WORLD.Get_rank();

   Maze maze = createZigZagMaze();
   ParallelMazeToGraphConverter converter;
   converter.createGraph(maze);
   Graph g = converter.getGraph();
   std::set<Vertex> vertices = g.getVertices();

   if (rank == 0) {
    EXPECT_TRUE(converter.hasVertexRepresentation(0));
    // ...
   } else if (rank == size - 1) {
    EXPECT_TRUE(converter.hasVertexRepresentation(2));
    // ...
   } else {
    EXPECT_TRUE(converter.hasVertexRepresentation(4));
    // ...
   }
}
```

Fazit

- Implementierungsaufwand wurde deutlich unterschätzt, sodass Analyse nicht mehr möglich war
- Parallelisierung mit MPI bereitete initial große Probleme bei der Vorstellung des Kontrollflusses, war dann aber gut handhabbar
- Entwicklung hat zwar enormen Aufwand bereitet, jedoch sehr spannend unter C++ mit MPI zu schreiben und zu testen