MPI Network Benchmark

MNB

Paul Lindt

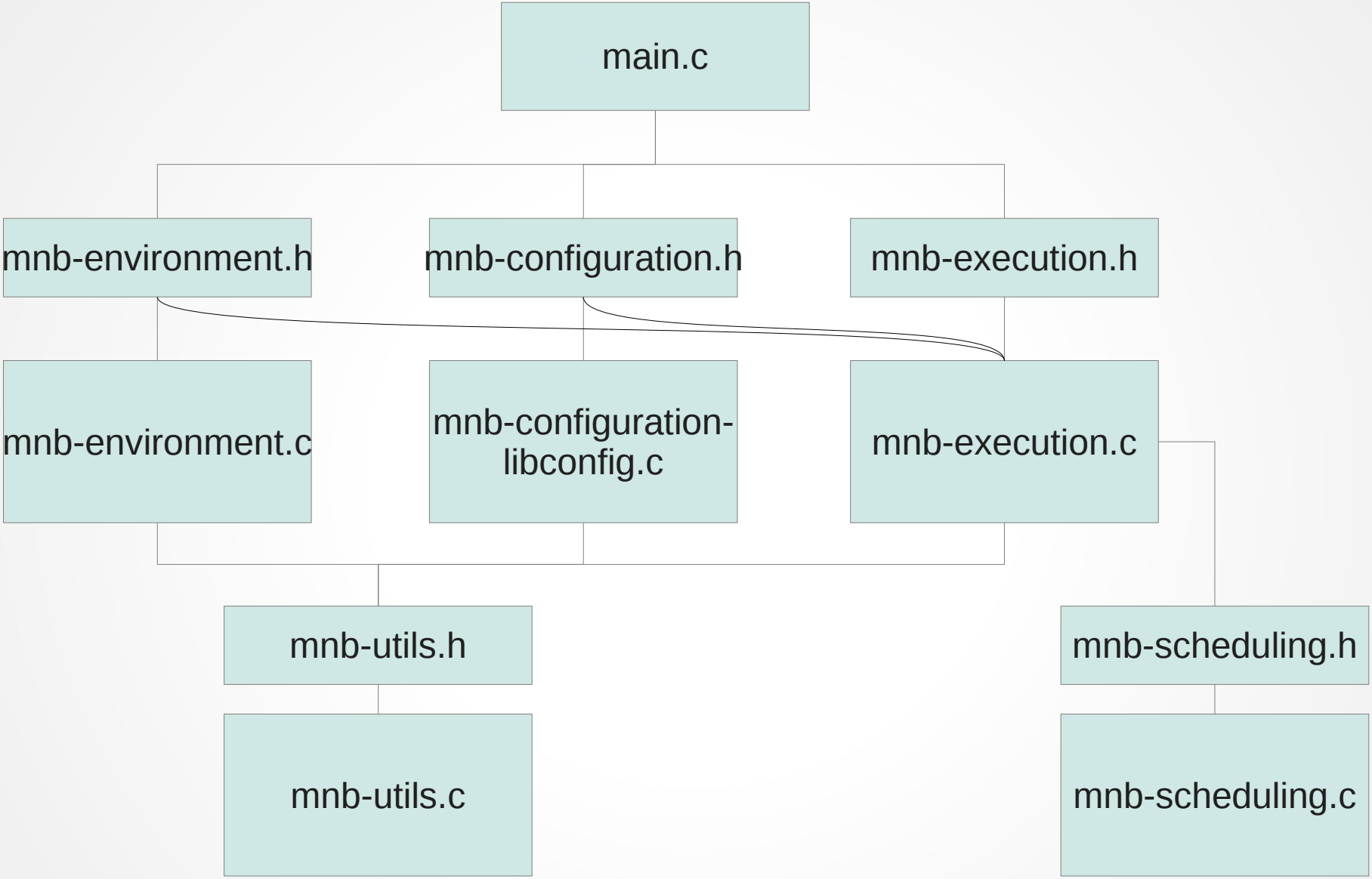
Gliederung

- Motivation
- Programm Struktur
- Funktionsweise
- Verbesserungsmöglichkeiten
- Was habe ich gelernt

Motivation

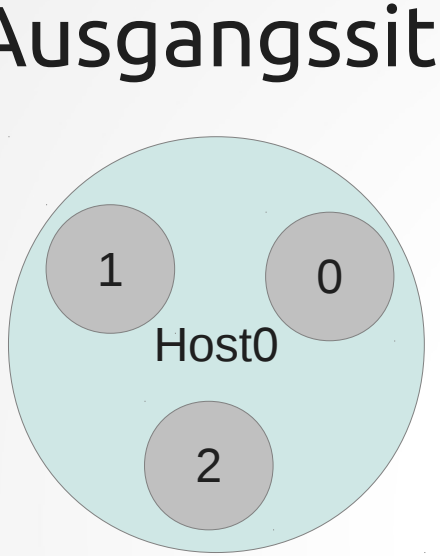
- Leistungsanalyse von MPI
- Fehlersuche innerhalb von MPI-Implementierungen
- Dabei
  - Möglichst effiziente Ausführung
  - Möglichst unverfälschte Ergebnisse

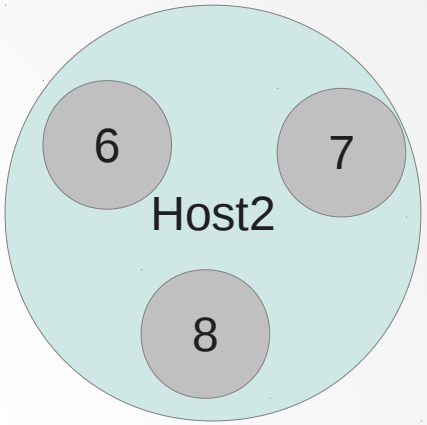
Programmstruktur

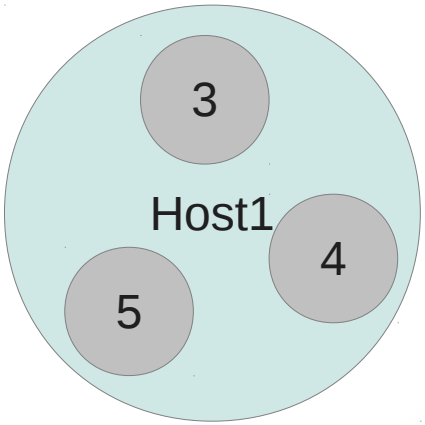


Funktionsweise

- Ausgangssituation





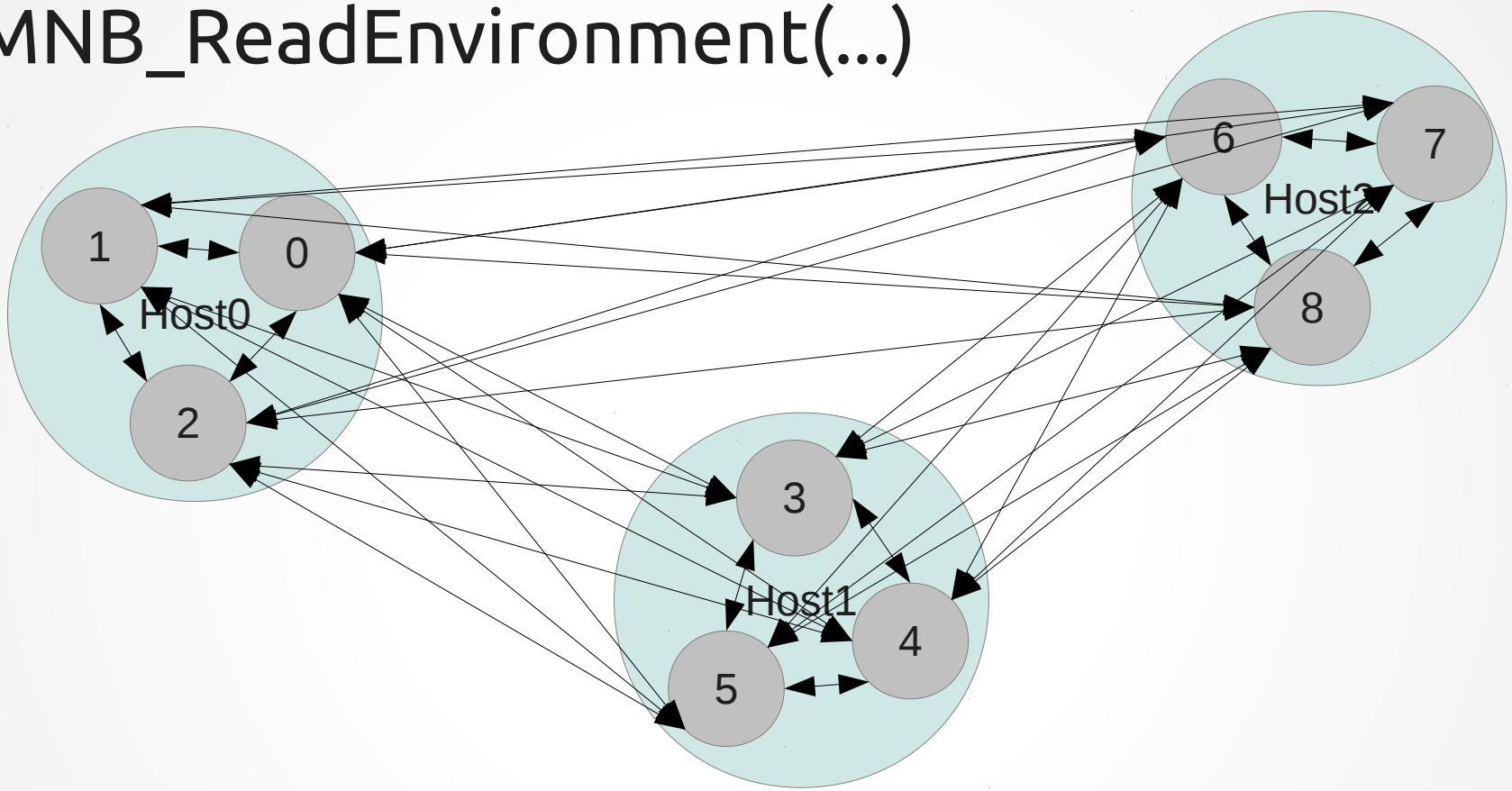


Funktionsweise

- MNB_ReadEnvironment(...)
  - Alle Tauschen die Hostnamen untereinander aus
    - MPI_Allgather(...)
  - Gleicher Hostname → gleicher Node
  - Node mit den wenigsten Ranks wird zum Scheduler

Funktionsweise

- MNB_ReadEnvironment(...)



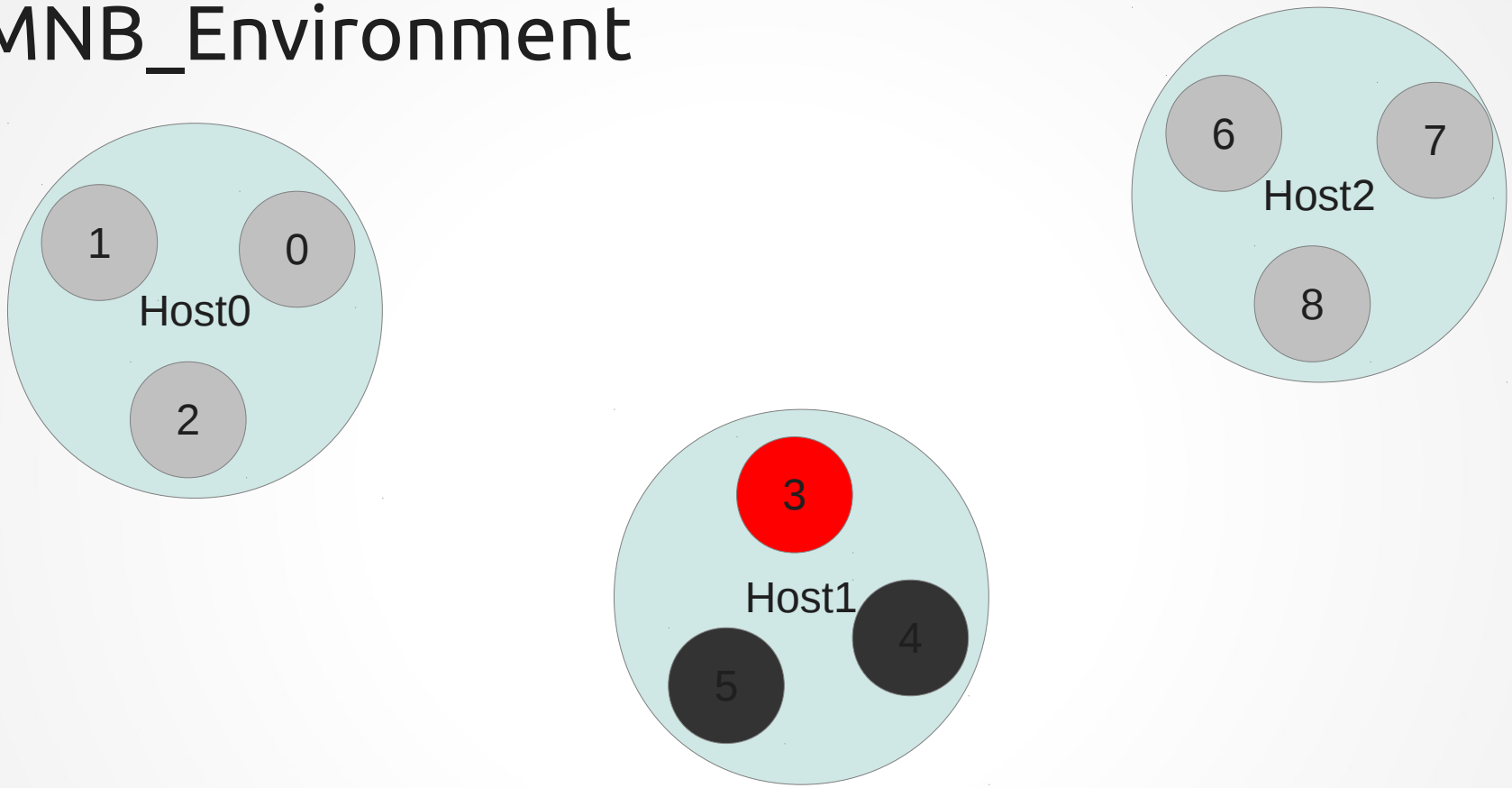
Funktionsweise

```
typedef struct{
  int numberOfNodes;
  int numberOfRanks;
  int schedulerRank;
  MNB_Node * schedulerNode;
  MNB_Node * localNode;
 struct MNB_List * testerNodes;
}MNB_Environment;
```

```
typedef struct{
char * hostname;
struct MNB_List *
ranks;
enum MNB_NodeStatus
status;
} MNB_Node;
```

Funktionsweise

- MNB_Environment



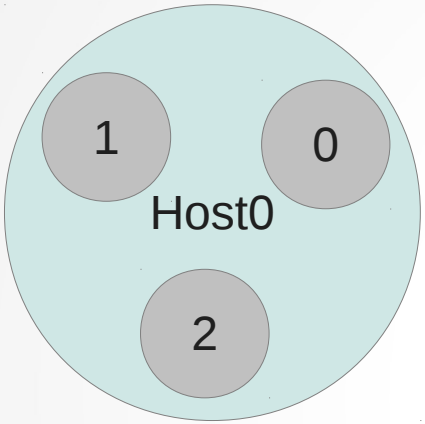
Funktionsweise

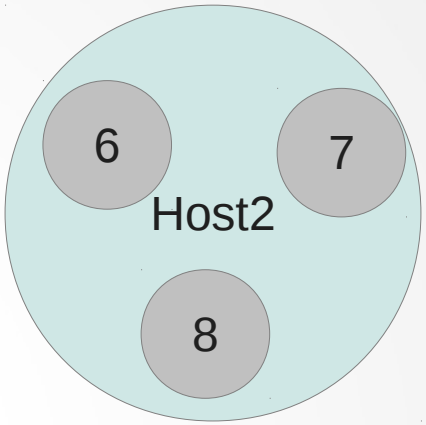
- MNB_ReadConfiguration(...)
  - Benutzt libconfig um die Datei zu Listen von elementaren Typen zu konvertieren
  - Für die Testliste
    - Löst verweise auf andere Listen auf (Größen,etc)
    - Konvertiert zu Liste vom MNB_Tests

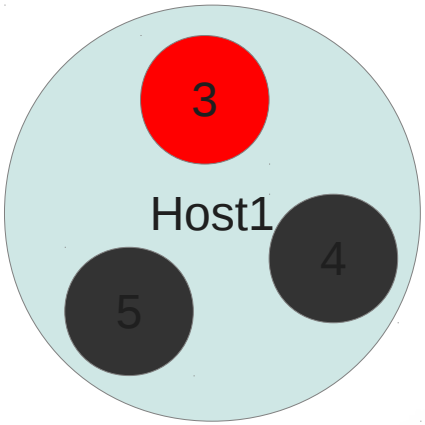
Funktionsweise

- MNB_ReadConfiguration:

keine Kommunikation







Funktionsweise

- MNB_Execute(...):Scheduler
  - Suche möglichs viele, Tests die sich gleichzeitig ausführen lassen.
  - Informiere auf jedem Node einen Rank über alle Tests in der nächsten Runde.
  - Wärend die Slaves noch arbeiten, suche die nächsten Tests aus.
  - Wenn keine Tests mehr da, Informiere alle über das Ende

Funktionsweise

- MNB_GetFirstSchedulableTest(...)
  - Wird solange aufgerufen wie es Tests zurückgibt
  - Tests sind:
    - Möglichst groß (Node/Rank Anforderung)
    - Möglichst perfekt passend
  - Iterative Implementierung einer Tiefensuche

Funktionsweise

```
typedef struct {
    char * name;
    char * test;
    struct MNB_List * sizes;
    struct MNB_List *
            nodeConf;
    int minRepeats;
    int maxDuration;
} MNB_Test;
```

```
typedef struct {
    int type;
    int numberOfBlocks;
    MNB_MessageBlock *
messageBlocks;
} MNB_Message;
```

```
typedef struct {
    MNB_MessageBlockHeader *
                              header;
    MNB_MessageBlockBody * body;
} MNB_MessageBlock;
```

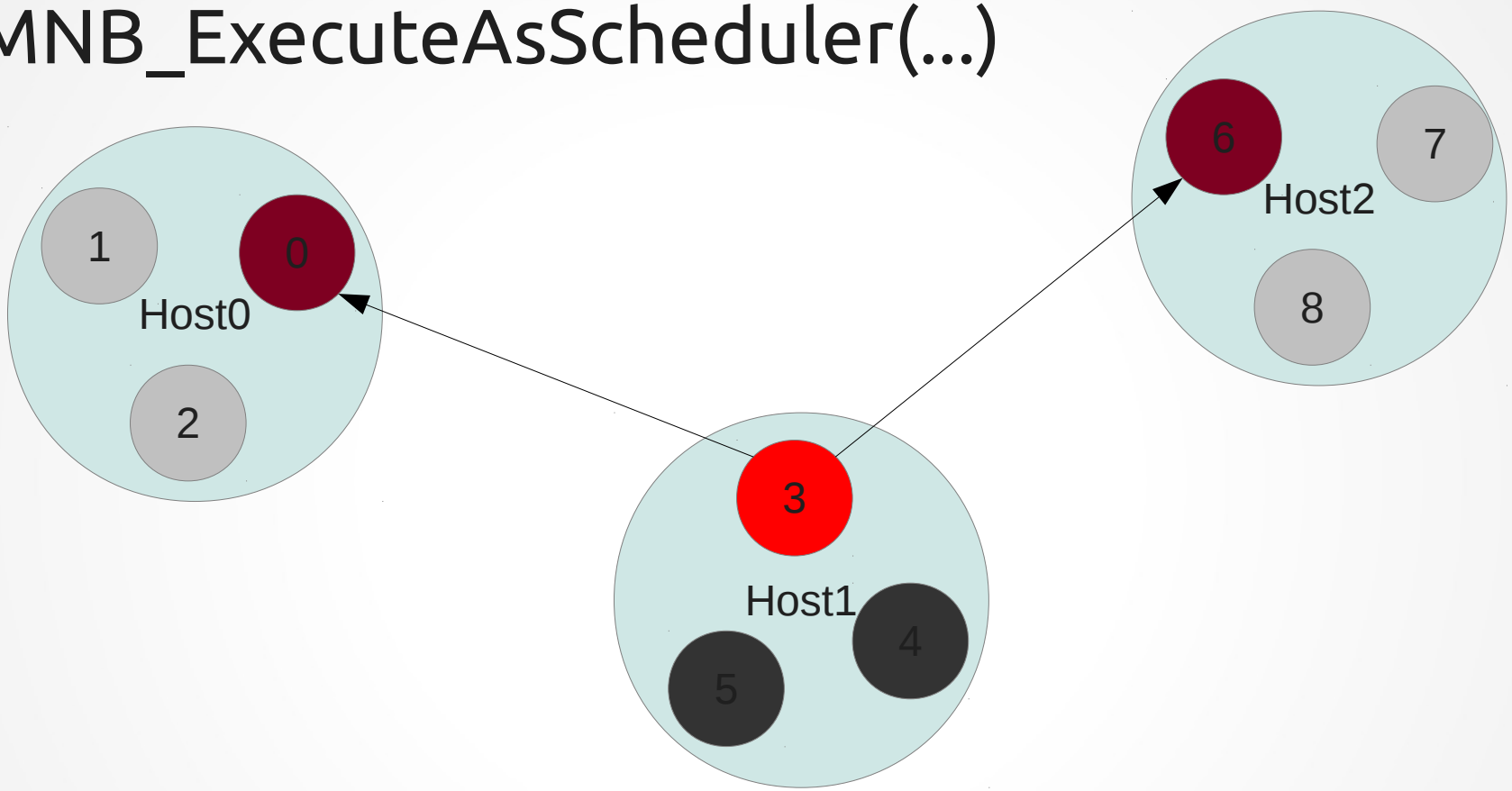
Funktionsweise

```
typedef struct{
    int numberOfNodes;
    int numberOfRanks;
    int numberOfSizes;
    int sizeOfTestName;
    int sizeOfTestSource;
}
MNB_MessageBlockHeader
;
```

```
typedef struct{
    int  * ranks;
    int  * sizes;
    char * testName;
    char * testSource;
    int minRepeats;
    int maxDuration;
}MNB_MessageBlockBody;
```

Funktionsweise

- MNB_ExecuteAsScheduler(...)

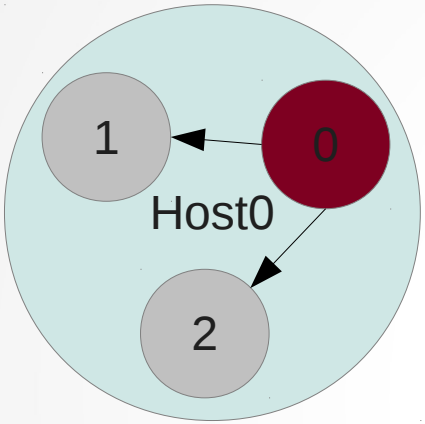


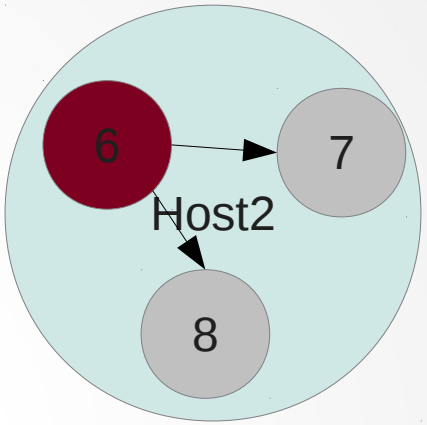
Funktionsweise

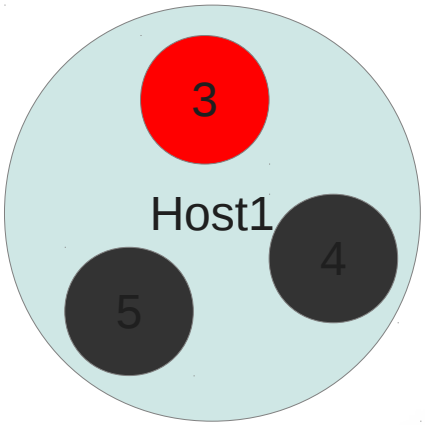
- MNB_Execute(...):Tester
  - Warte auf Informationen über die nächste Runde
    - Wenn Quit-Nachricht → Ende
    - Wenn Resender → schicke die Info an alle Ranks auf dem Node
  - Erzeuge alle notwenigen MPI_Group/MPI_Comm
  - Wenn selbst Teil eines der Comms, führe dazugehörigen Test aus.

Funktionsweise

- MNB_ExecuteAsTester(...)

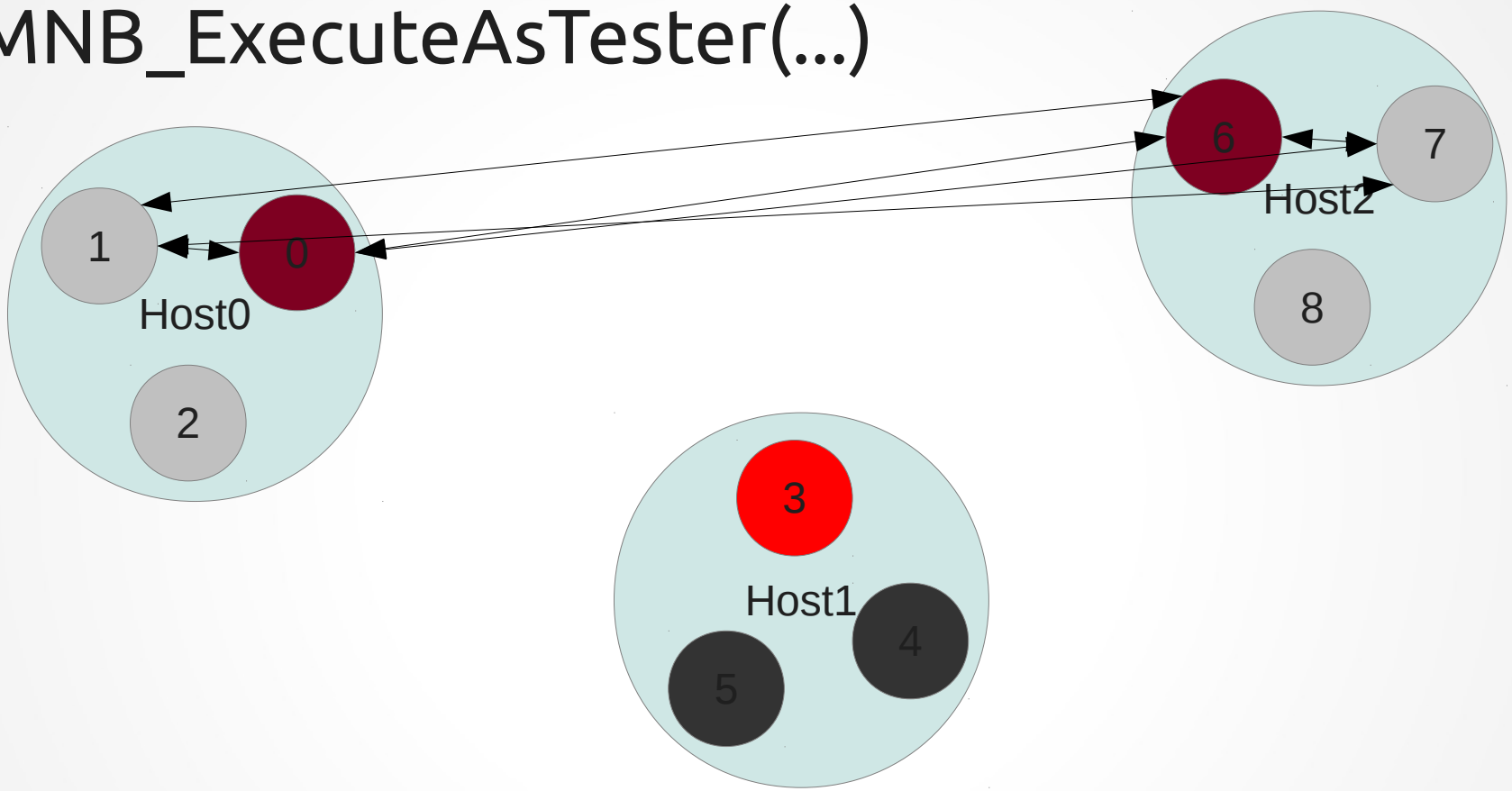






Funktionsweise

- MNB_ExecuteAsTester(...)



Verbesserungsmöglichkeiten

- Ausgelagerte Suche nach dem idealen Scheduling
- Segmentierung der tester Nodes um dynamisches Scheduling zu ermöglichen
- Erweiterbarkeit durch Tests aus Plugins
- Grafische aufbereitung der Ergebnisse

Was habe ich gelernt?

- Einblick in für paralleles Programmieren wichtige Denkmuster.
- Unit-tests sind auch bei scheinbar kleinen Projekten sinnvoll
- Möglichst früh die echte Umgebung benutzen
- Die Dinge sind in Wahrheit meist Aufwendiger als sie Scheinen.