Execution of Task Parallel OpenMP Programs in Distributed Memory Environments - Execution Environment

Johannes Erwerle

2018-02-20

Inhalt

- Überblick OpenMP
- Ausführung auf Distributed Memory System
  - Idee
  - verwandte Projekte
  - Präprozessor
  - Execution Environment
    - Beispiele zur Ausführung von Tasks
  - Aktueller Stand
  - noch fehlende Features und Einschränkungen
- Zusammenfassung

OpenMP

- API um einfach parallelen Code auf Shared Memory Systemen zu schreiben.
- Umgesetzt durch #pragma-Direktiven und div. Funktionen
- klassischerweise Parallelisierung von Schleifen

```
int main() {
    int a[] = new int[1024];

    #pragma omp parallel for
    for(int i = 0; i < 1024; i++){
        a[i] = ...;
    }
}
```

Task-paralleles OpenMP

- normalerweise Parallelisierung von Schleifen
- aber auch Tasks möglich

```
int main(){
    #pragma omp task
    { /*do something*/ }
    #pragma omp task
    { /*do something else*/ }
    #pragma omp task
    { /*do something completely different*/ }
}
```

Einschränkungen von OpenMP:

- Benötigt Shared Memory, deswegen nur 1 Node möglich

Ausführung auf Distributed Memory Systemen

Idee

- Tasks auf anderen Nodes ausführen
- normaler OpenMP C/C++ Code als Eingabe
- Präprozessor
  - identifiziert Tasks und modifiziert Code
  - baut Execution Environment Code ein
- Execution Environment
  - kümmert sich um Ausführung von Tasks auf anderen Knoten
- Speedup! (hoffentlich)

Ausführung auf Distributed Memory Systemen

verwandte Projekte

ompSs

- task-basierte parallele Ausführung
- eigenes API ähnlich wie OpenMP
- erlaubt Ausführung auf Distributed Memory Systemen
- aber nur begrenzt skalierbar:
  - Sämtlicher Speicher muss auf Master-Node passen
  - Nur Master-Node kann Tasks erzeugen
  - Benchmarks hören bei 16 bis 32 Nodes auf

Ausführung auf Distributed Memory Systemen

Präprozessor

- Bachelor Arbeit von Markus Baur
- filtert Tasks heraus und speichert diese mit ID ab
- ersetzt Task durch Aufruf für Execution Environment
- findet heraus auf welche Daten ein Task zugreift

Ausführung auf Distributed Memory Systemen

```
int main(){
    #pragma omp task
    { /*do something*/ }
    #pragma omp task
    { /*do something else*/ }
    #pragma omp task
    { /*do something completely different*/ }
}
```

```
int main(){
    create_task(1, /*more parameters*/);
    create_task(2, ...);
    create_task(3, ...);
}
```

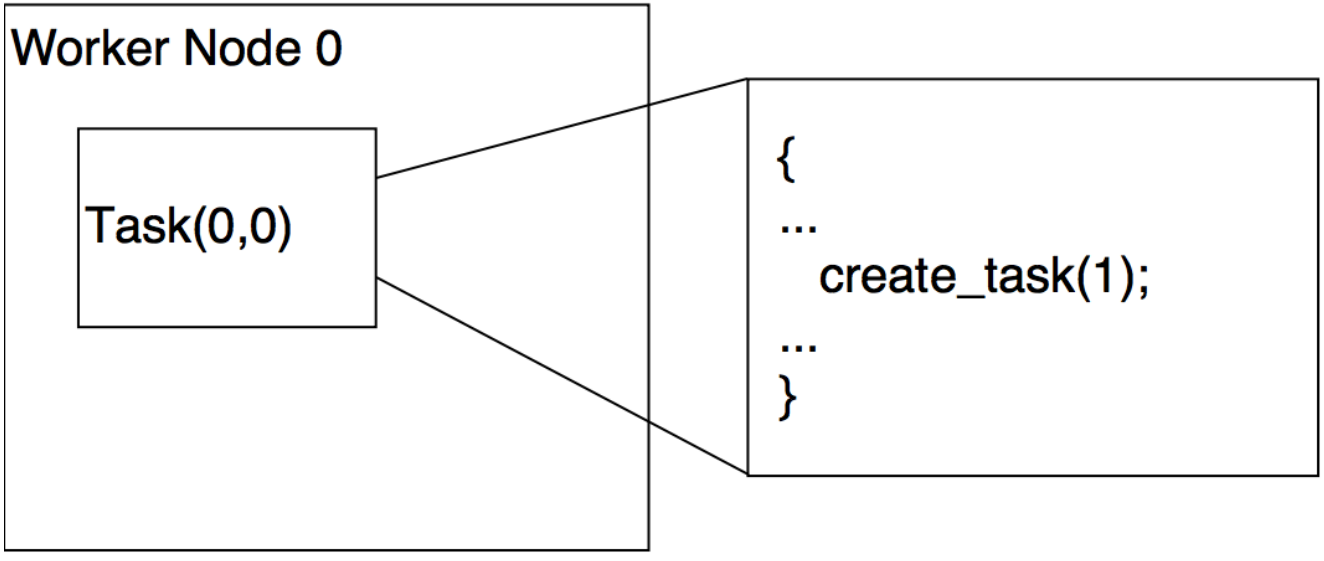
```
1 : { /*do something*/ }
2 : { /*do something else*/ }
3 : { /*do something completely different*/ }
```

Ausführung auf Distributed Memory Systemen

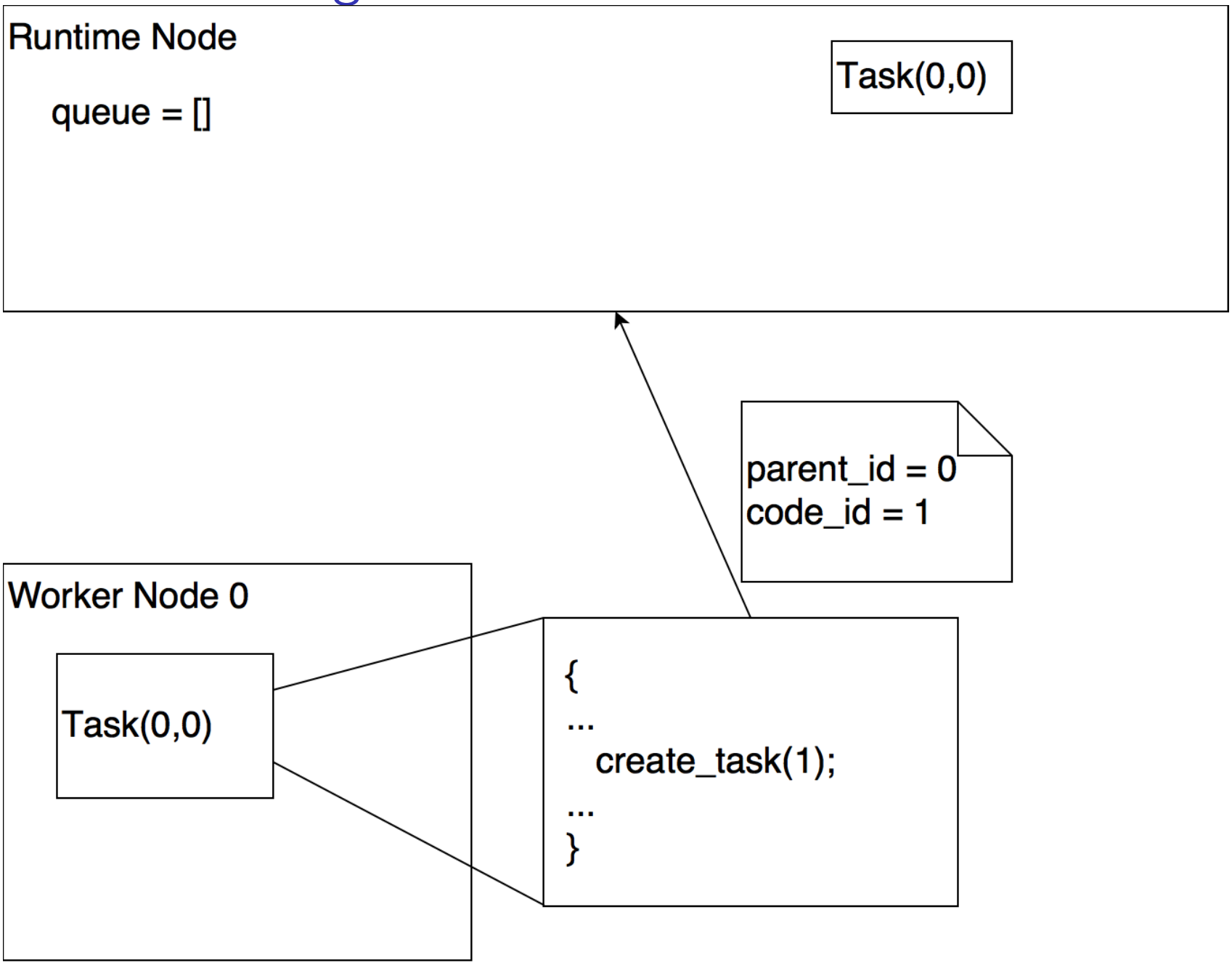
Execution Environment

- Zielarchitektur Distributed Memory Cluster
- Runtime Nodes:
  - Verwalten Tasks und Worker
  - koordinieren Synchronisationskonstrukte z.B. `taskwait`
- Worker Nodes
  - Führen die ihnen zugewiesenen Tasks aus
  - Kümmern sich um die Übertragung des zug. Speichers

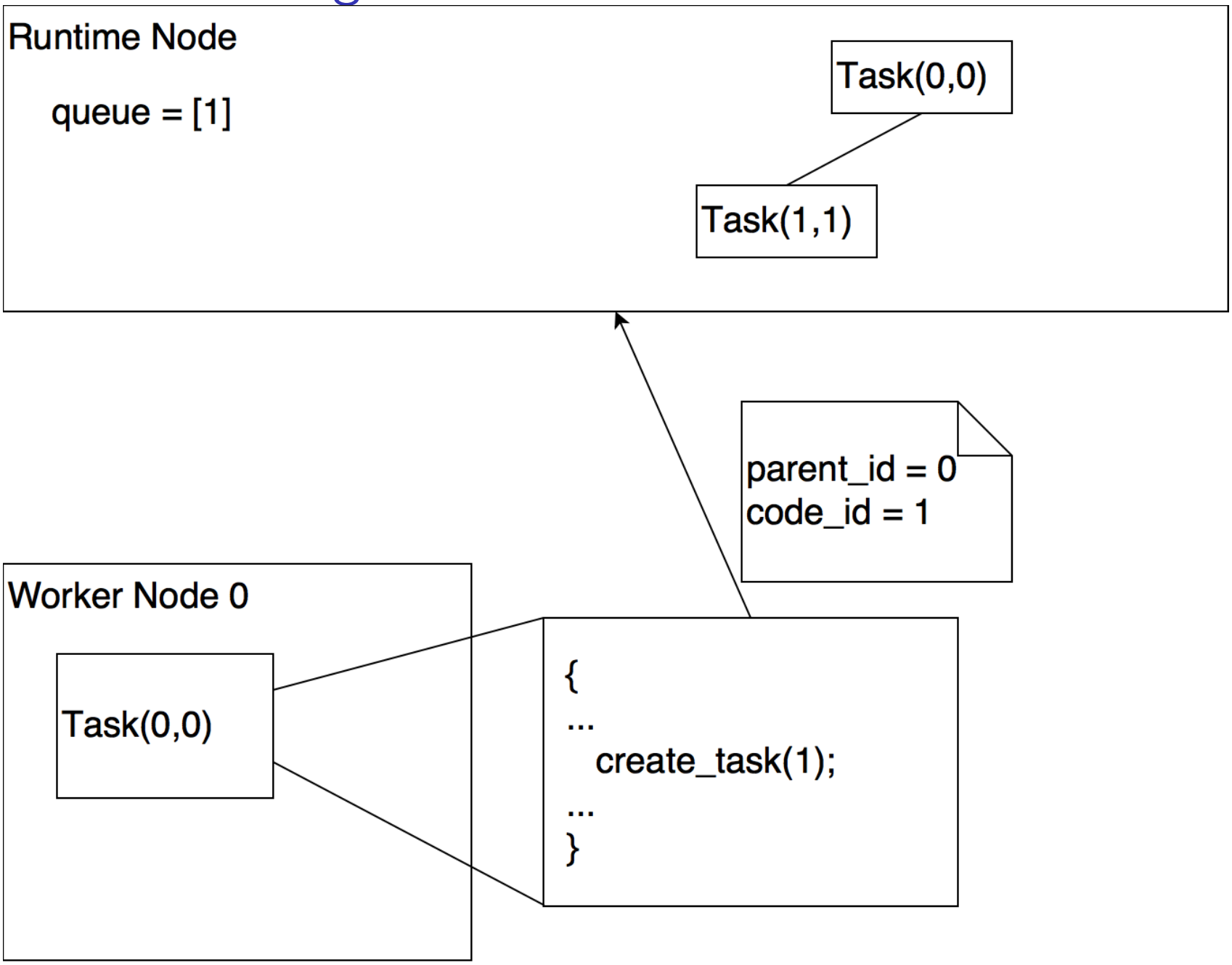
Neue Tasks erzeugen



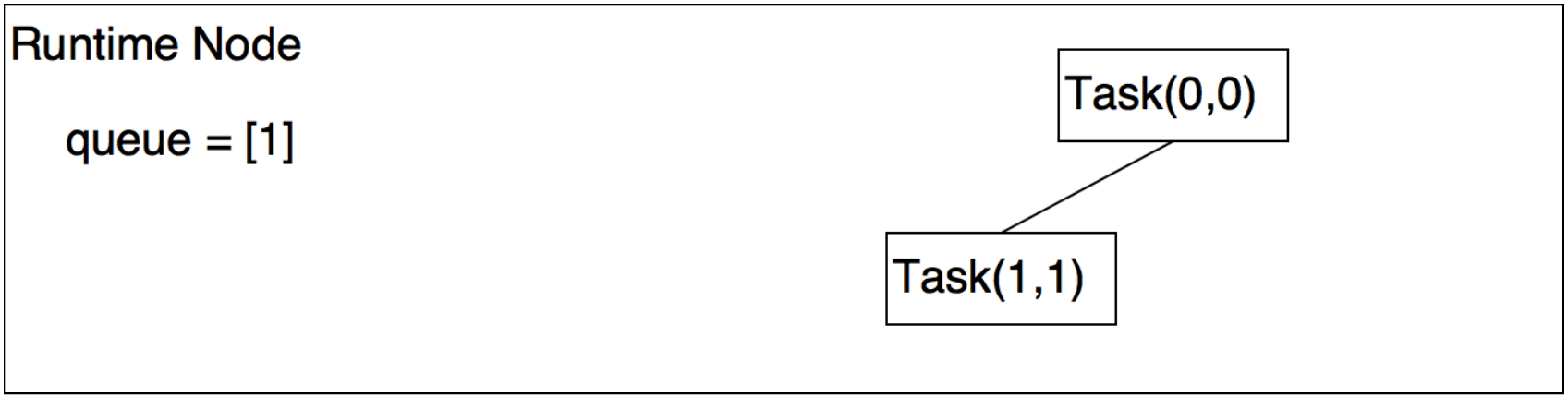
Neue Tasks erzeugen

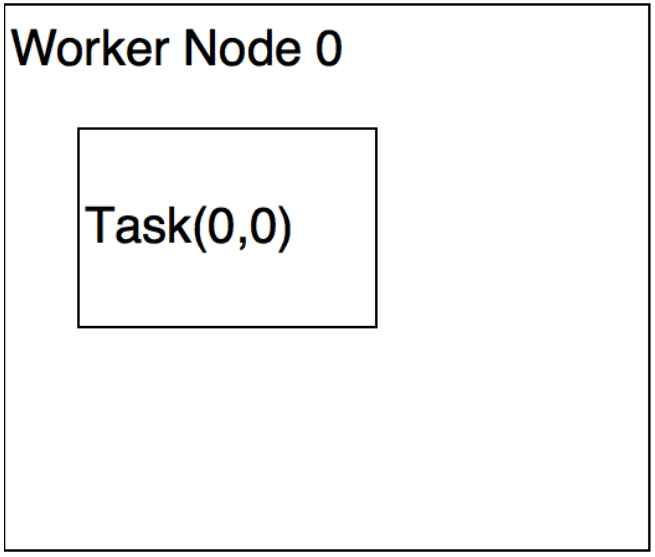


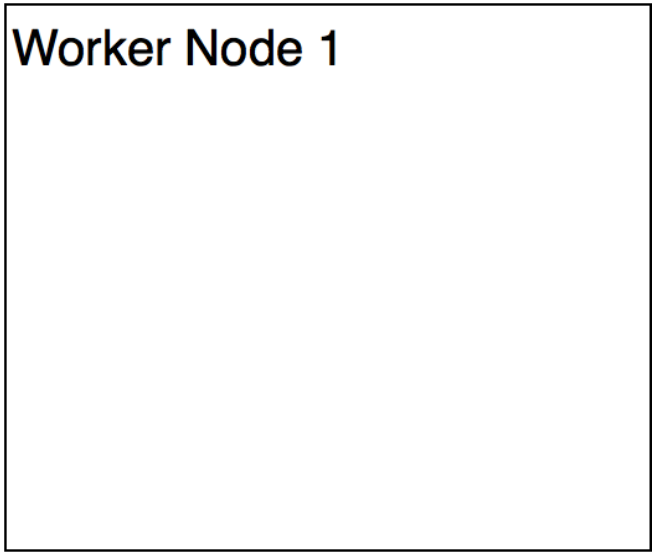
Neue Tasks erzeugen



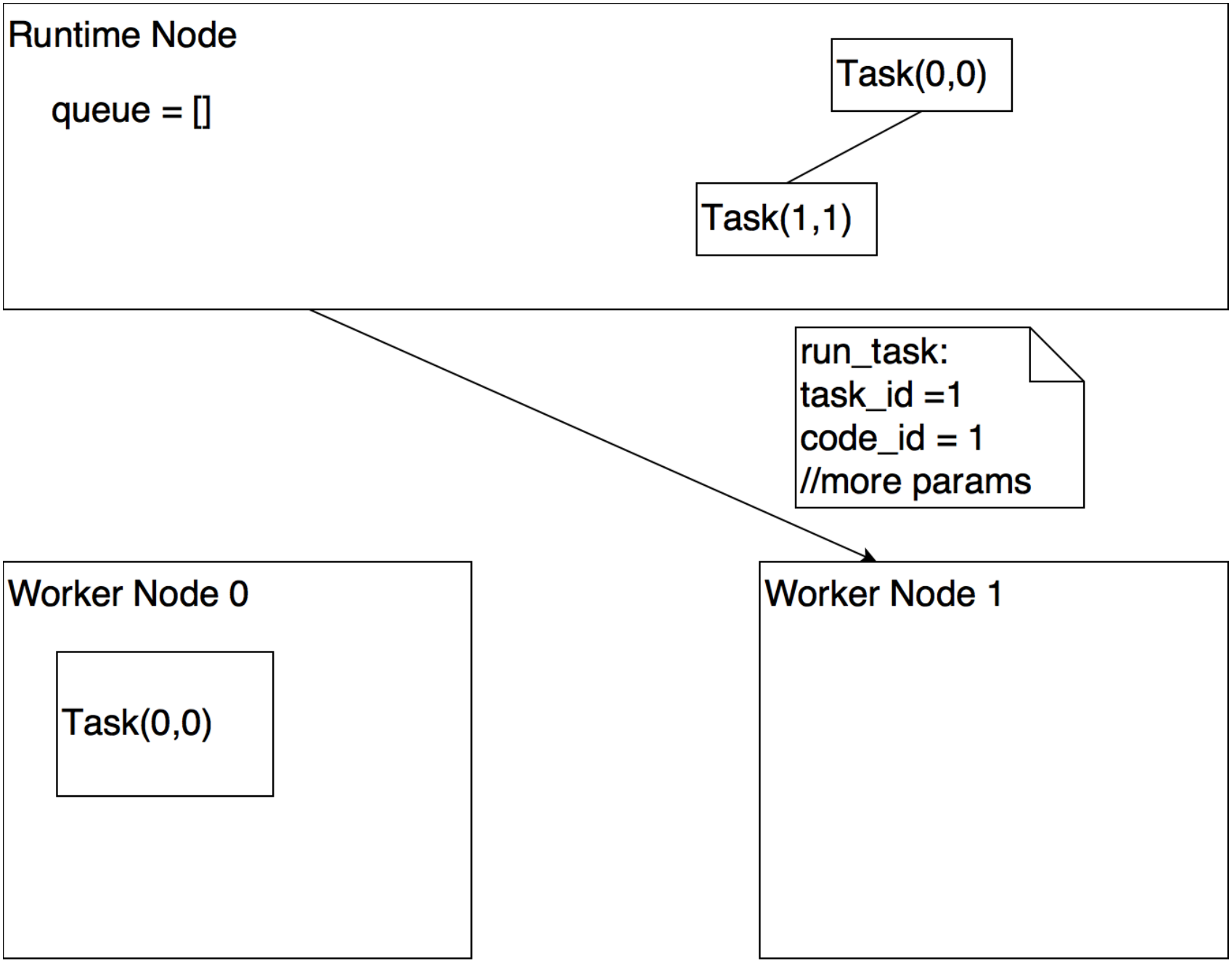
Task starten



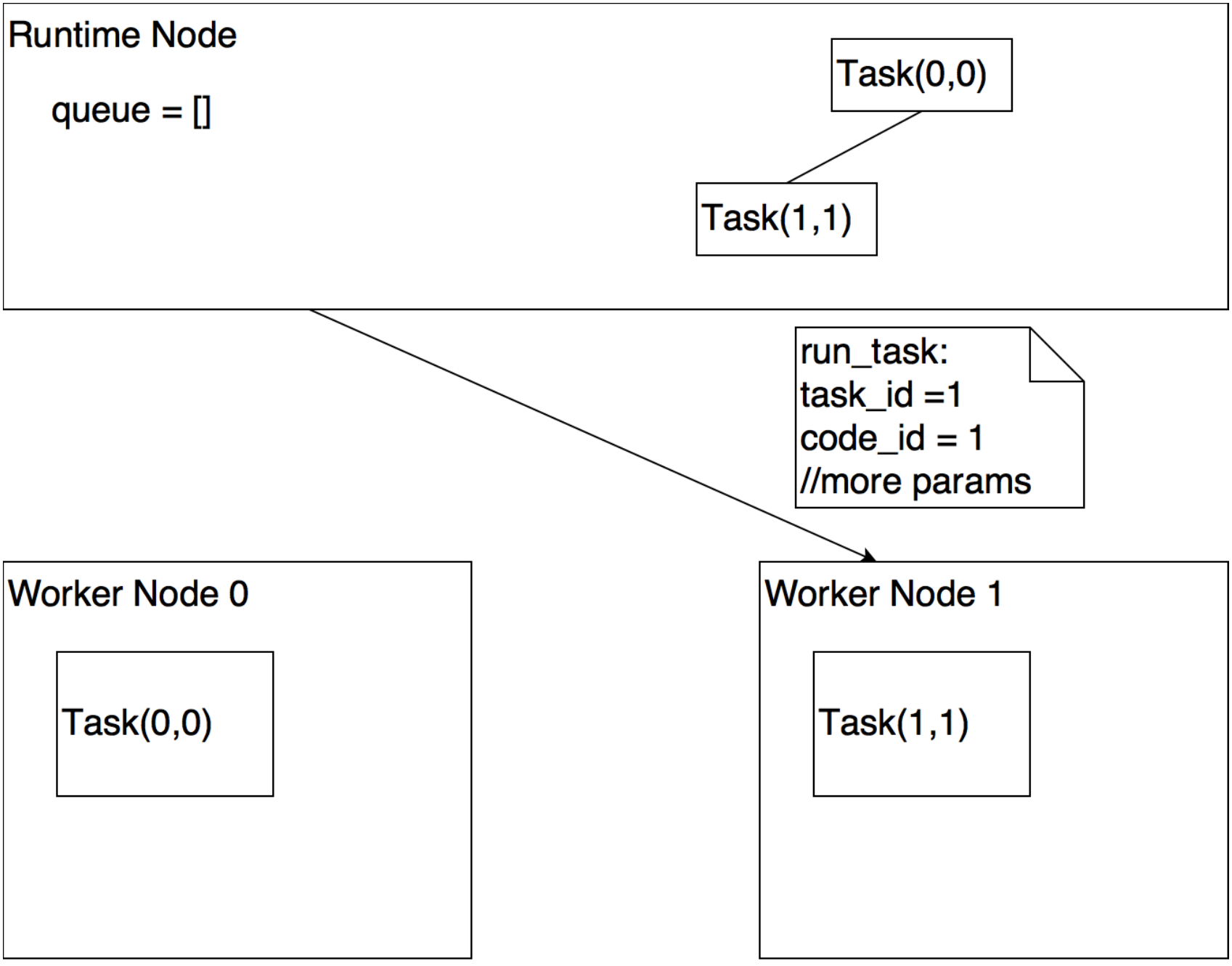




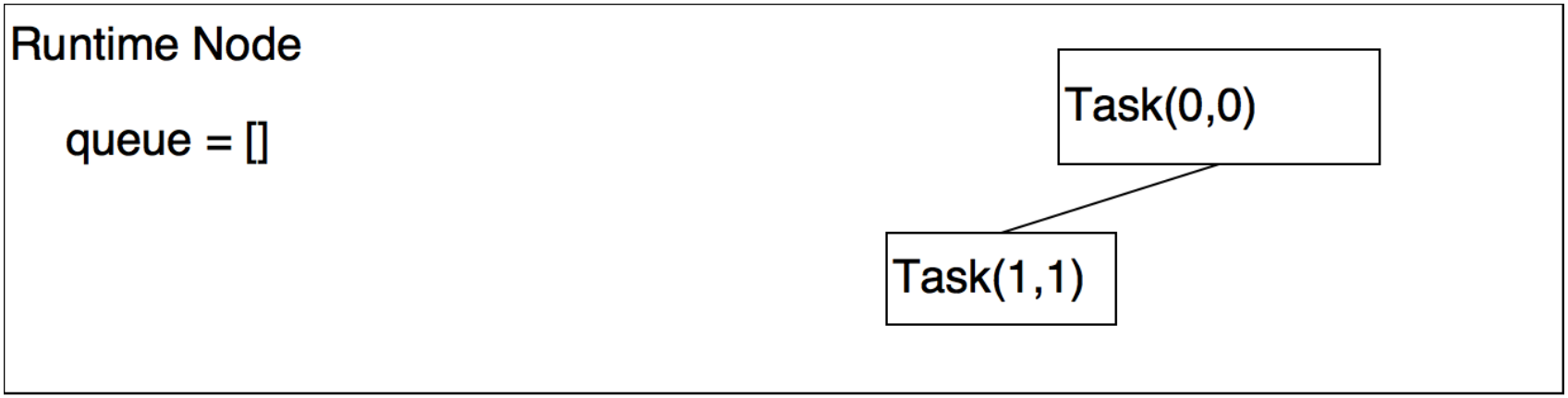
Task starten

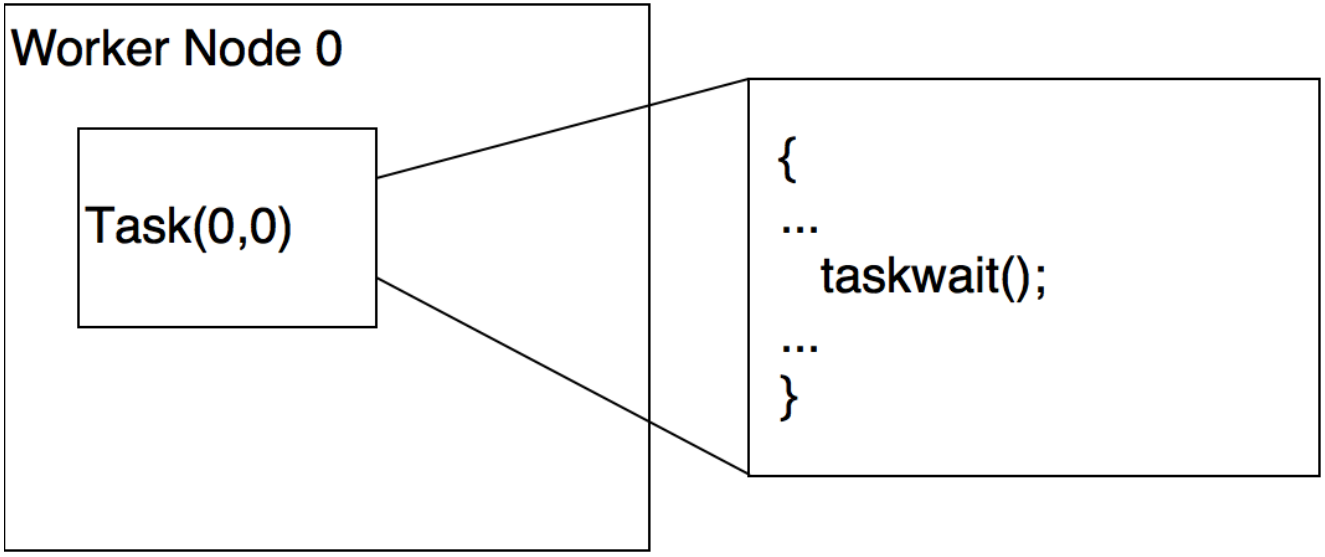


Task starten

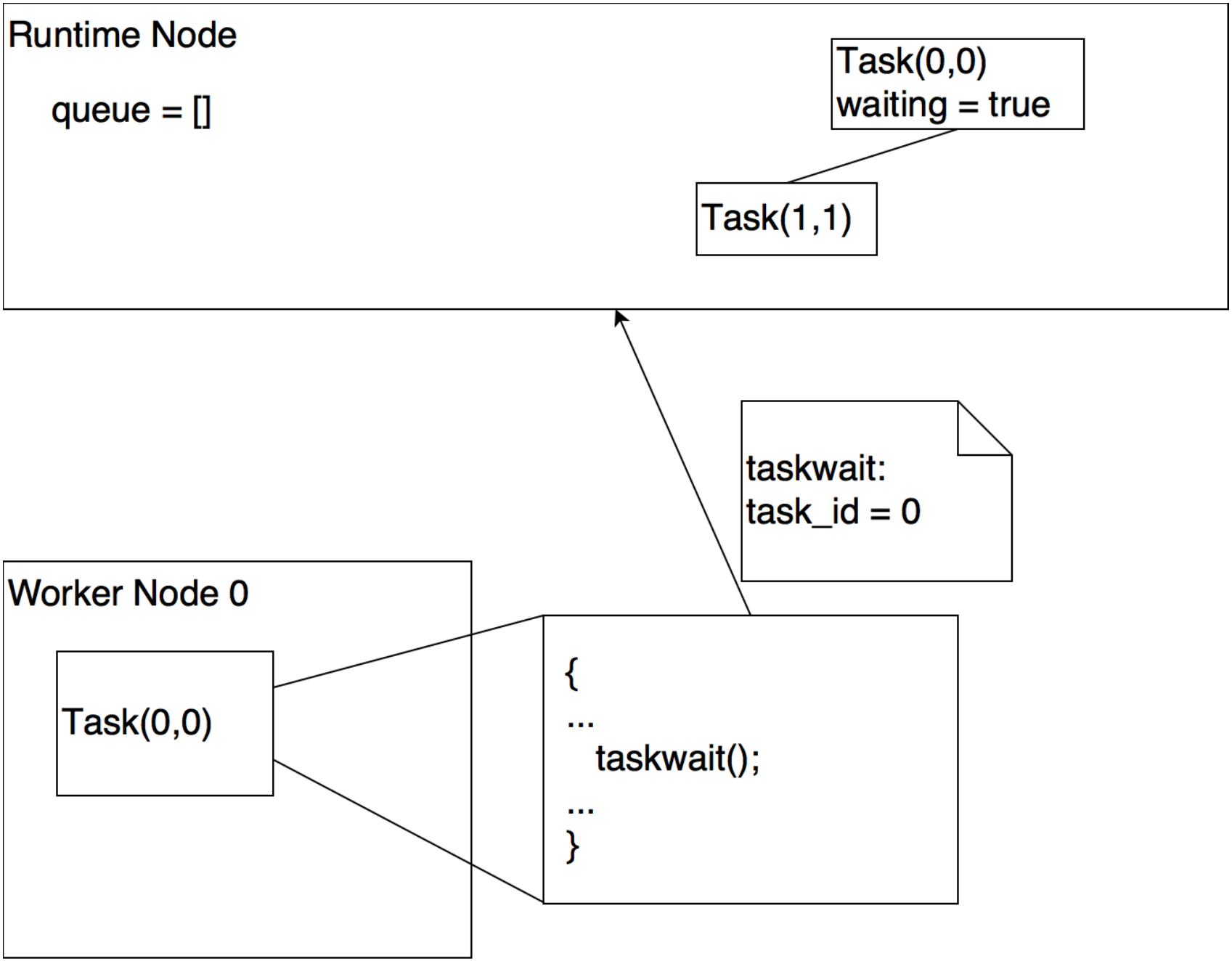


taskwait

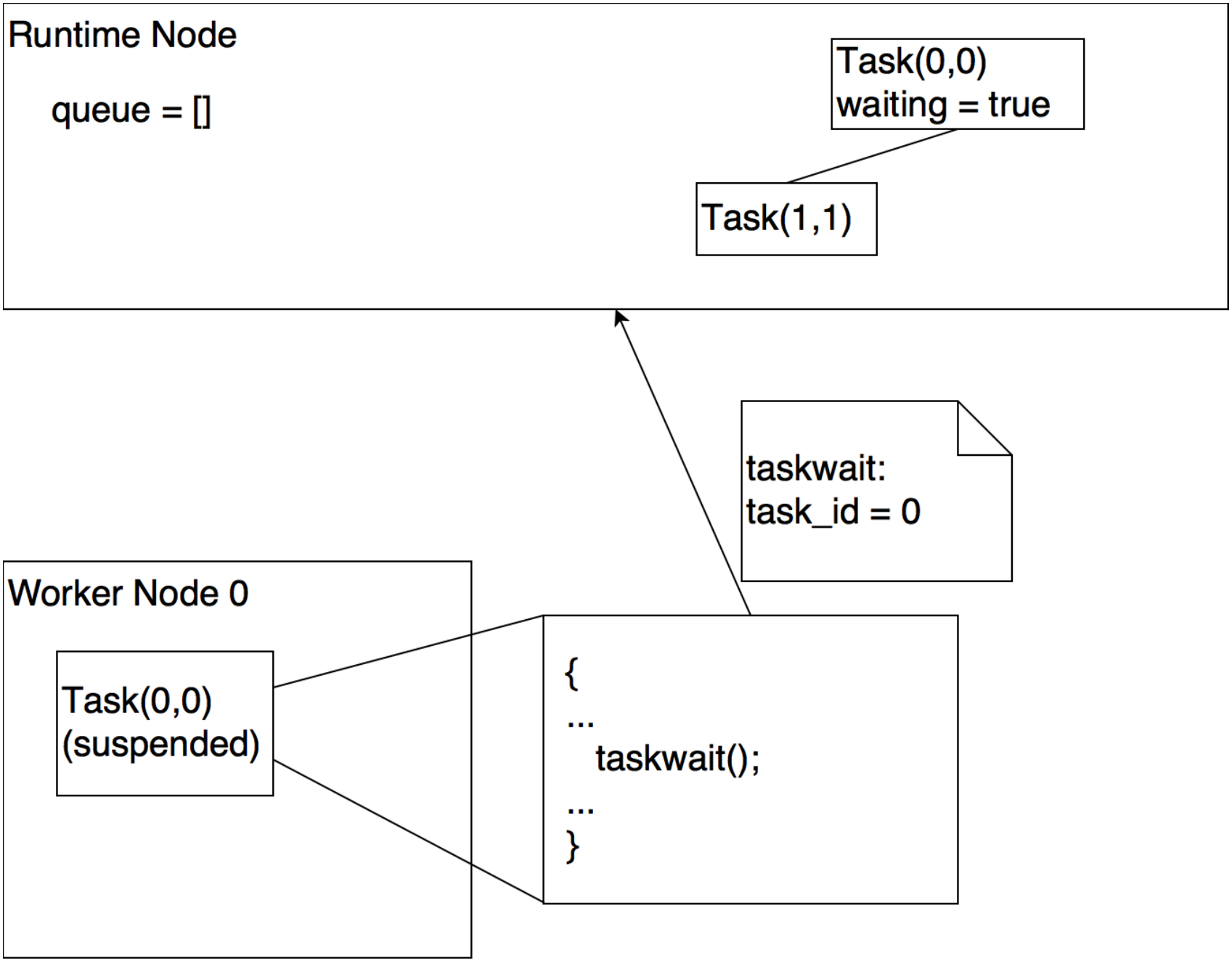

Runtime Node
queue = []
Task(0,0)
Task(1,1)


Worker Node 0
Task(0,0)
{
...
taskwait();
...
}

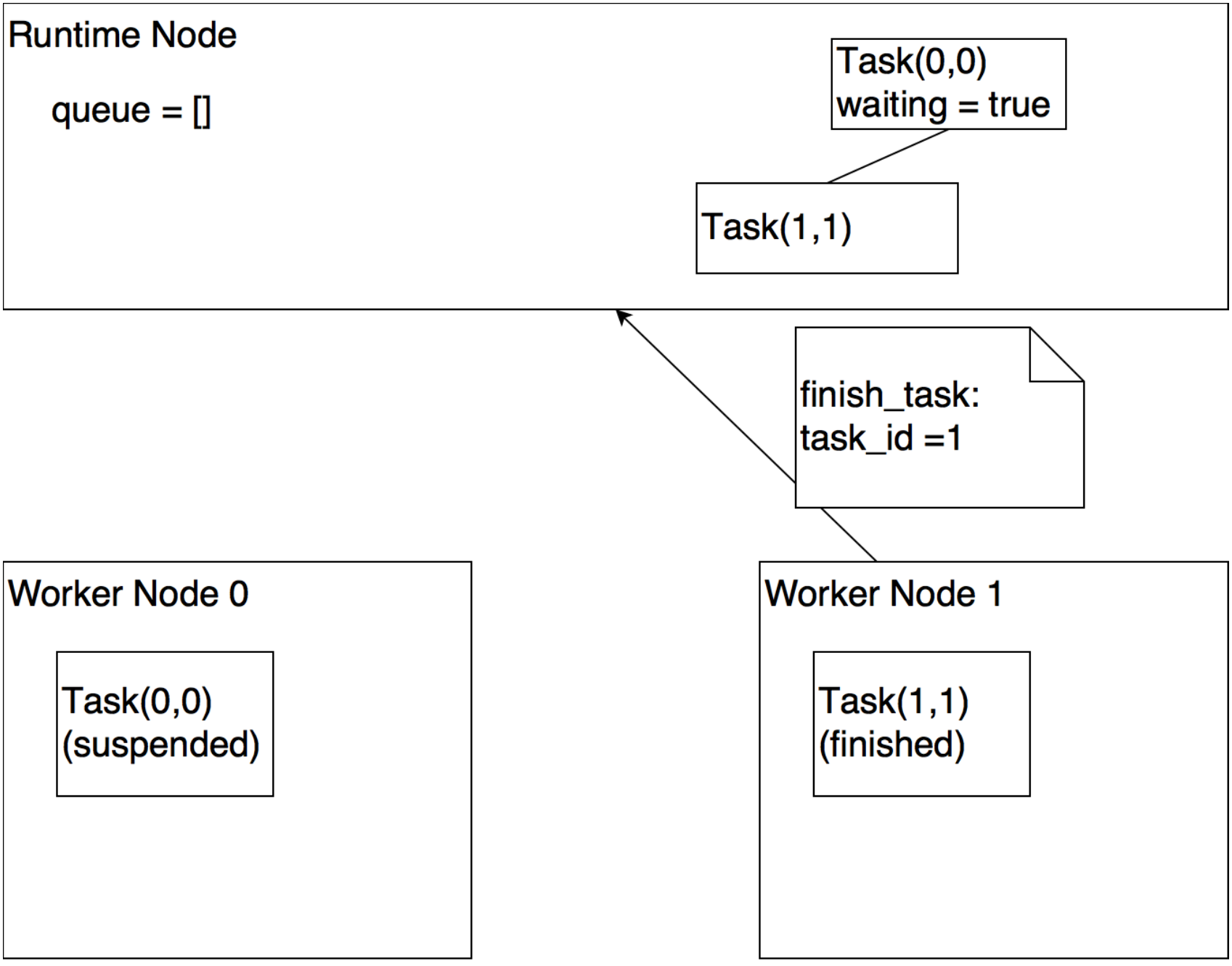
taskwait


Runtime Node
queue = []
Task(0,0)
waiting = true
Task(1,1)
taskwait:
task_id = 0
Worker Node 0
Task(0,0)
{
...
taskwait();
...
}

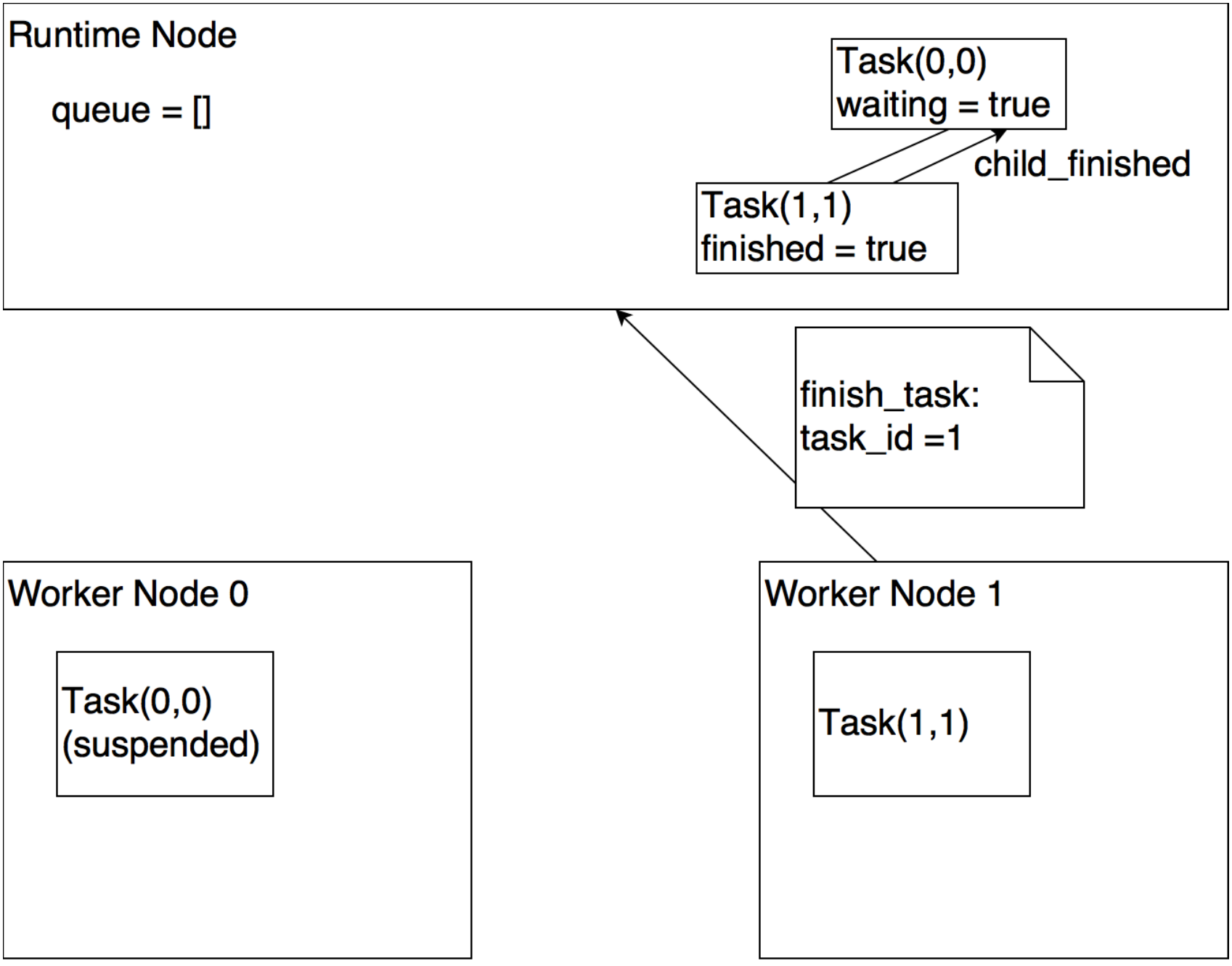
taskwait


Runtime Node
queue = []
Task(0,0)
waiting = true
Task(1,1)
taskwait:
task_id = 0
Worker Node 0
Task(0,0)
(suspended)
{
...
taskwait();
...
}

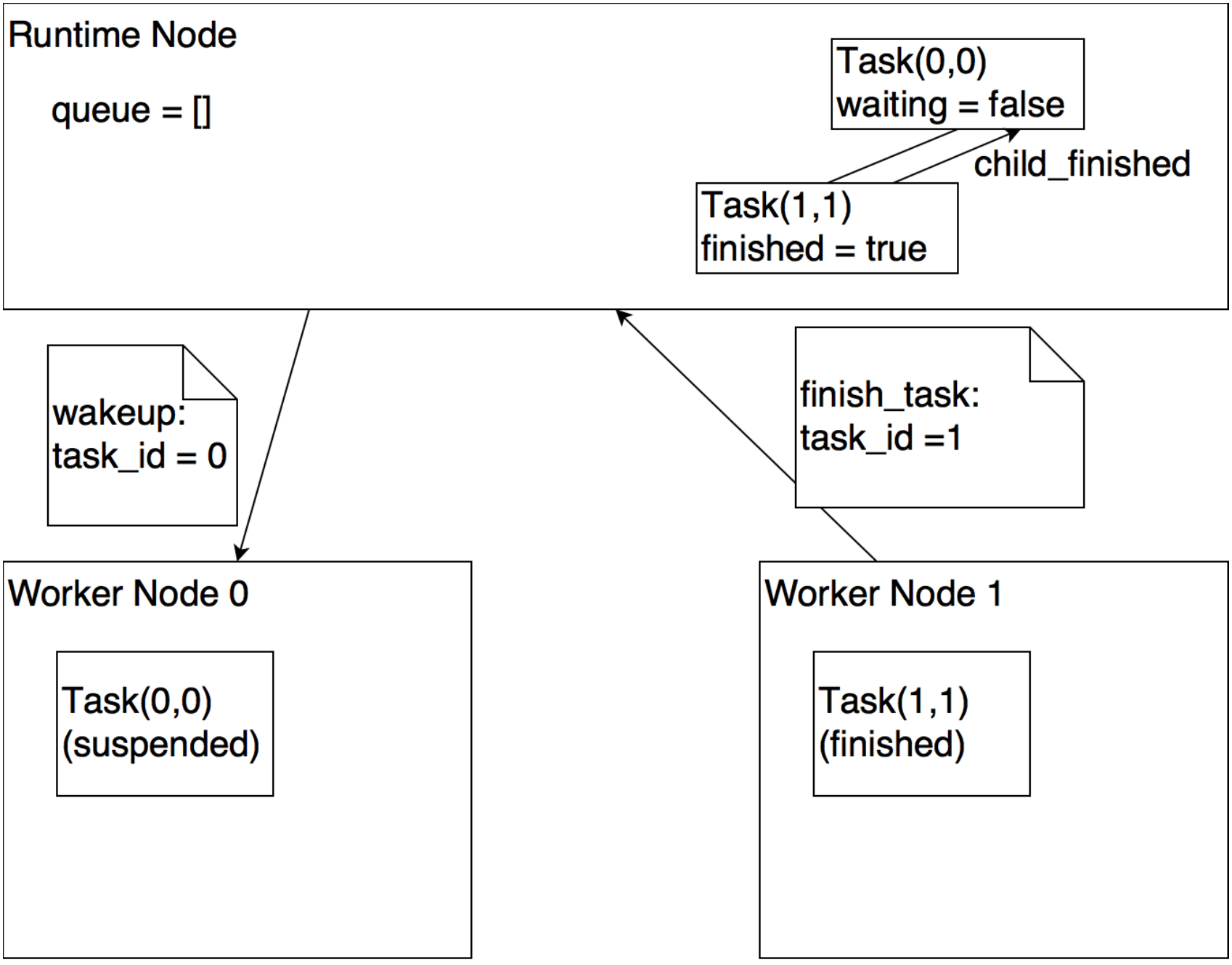
Task beenden



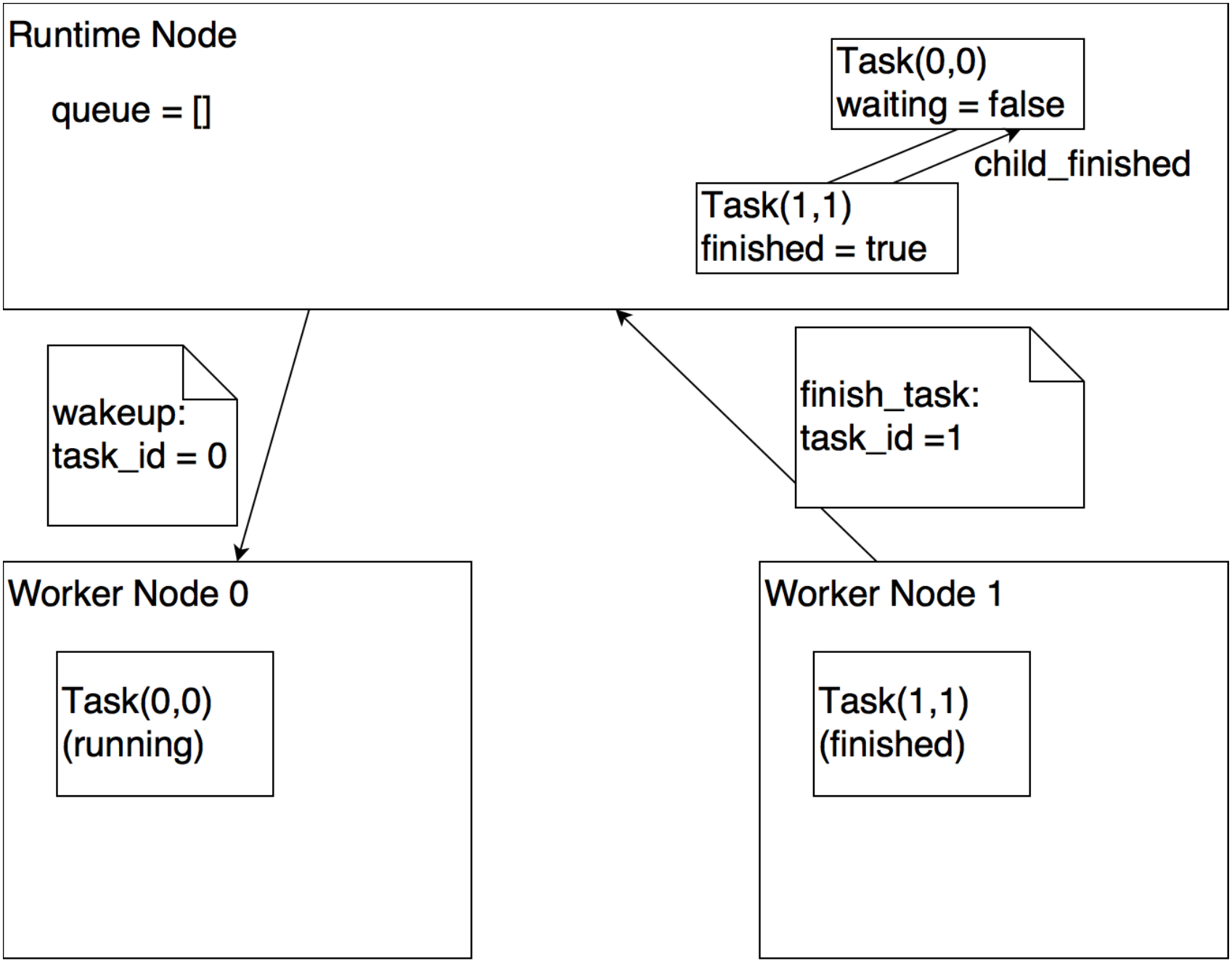
Task beenden



Task beenden



Task beenden



Aktueller Stand

- Prototyp
  - nicht alle Features von OpenMP
  - Scheduler nur Round Robin
- Umgesetzt als MPI-Programm
- 1 MPI-Prozess pro Node mit mehreren Threads
- 1 Runtime Node mit bel. vielen Workern
- Tasks erzeugen
- mehrere Tasks pro Worker ausführen
- `taskwait` und warten am Ende von `parallel` Blöcken

Features die noch implementiert werden müssen

- Speichertransfer
  - teuer
- Execution Environment mit Präprozessor verbinden
- Task Dependencies
- Mehrere Runtime-Nodes für Skalierbarkeit

Zusammenfassung

- OpenMP Tasks können auf anderen Nodes ausgeführt werden
  - eventuell nicht alle OpenMP Konstrukte sinnvoll umsetzbar
- Vorhandener Code kann hoffentlich beschleunigt werden
- einige Features fehlen noch
- Speicherübertragung ist kritischer Punkt
- viele Möglichkeiten für Performance-Steigerungen
  - Scheduler
  - Shared Memory auf Workern ausnutzen
  - Daten-Lokalität beachten