Gears OS の並列処理

伊波立樹@河野研

メタ計算を使った並列処理

- 並列処理のチューニングや信頼性の保証は難しい
  - ✦ 従来のテストやデバッグでは非決定的な実行の対処が困難
  - ✦ GPU などのアーキテクチャに合わせたプログラミング

- Gears OS は 通常の計算をノーマルレベル、信頼性と拡張性の計算をメタレベルに階層化することを目指して開発している

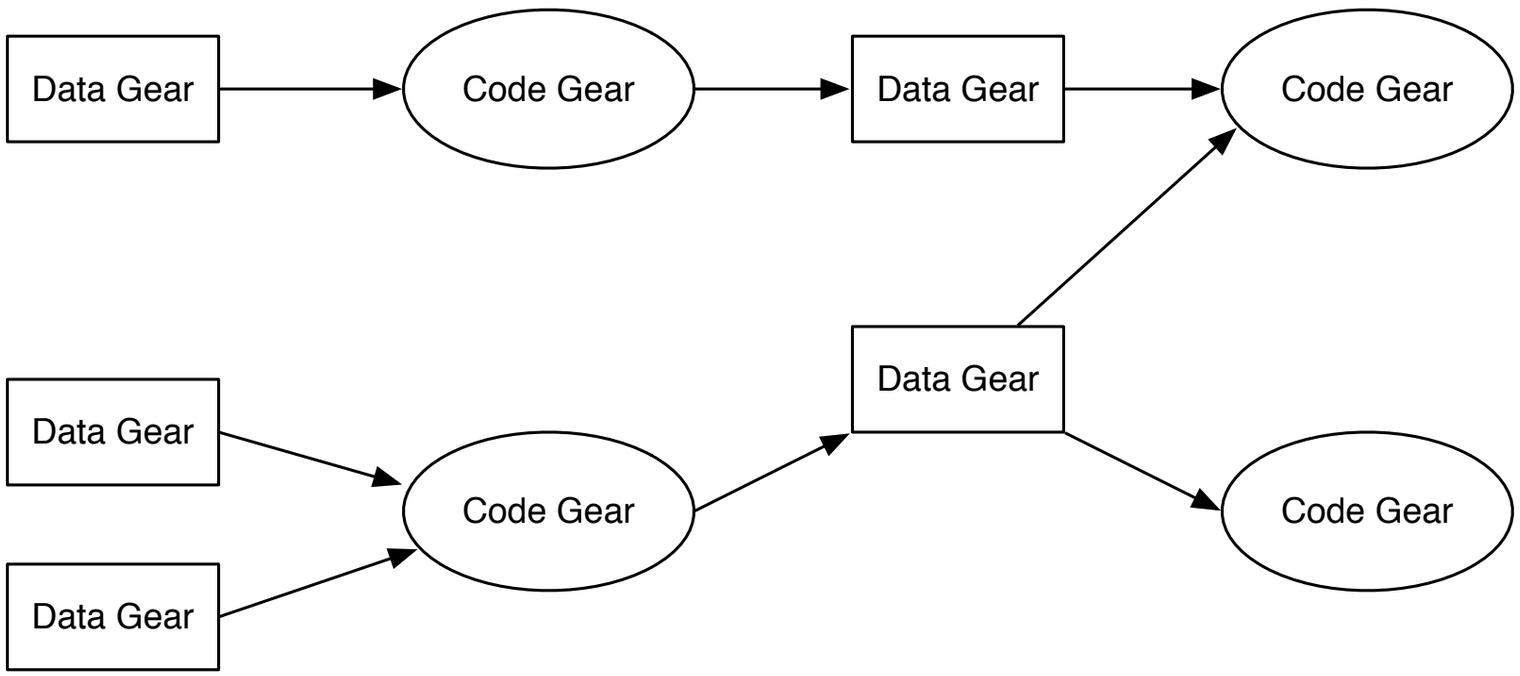
- 本研究では Gears OS の並列処理機構、並列処理構文、 Gears OS を実装するに連れて必要になったモジュール化の導入を行う

Code Gear/Data Gear

- Gears OS は処理の単位としてCode Gear、 データの単位としてData Gear で構成される

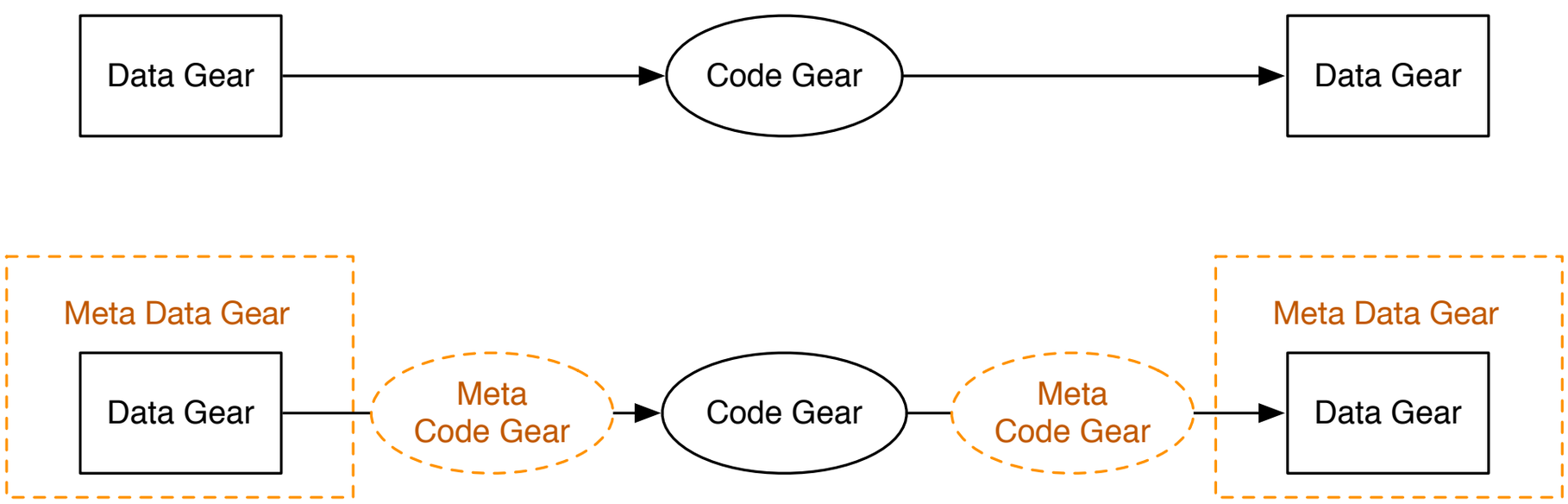
- Code Gear は必要な Input Data Gear が揃ったら実行し、 Output Data Gear を生成する

- Code Gear と Input/Output Data Gear 対応から依存関係を解決し、Code Gear の並列実行する



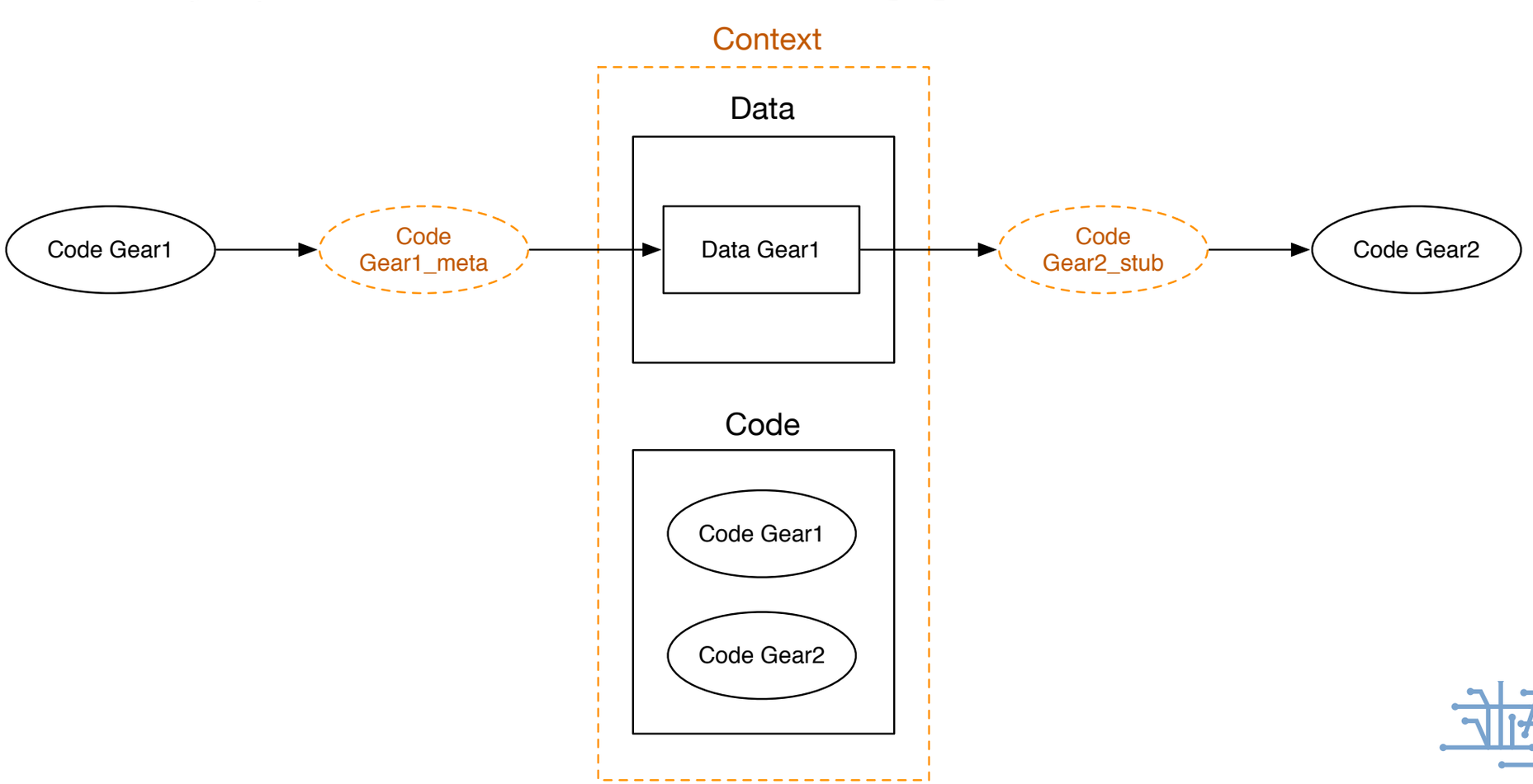
メタ計算

- 通常の計算を実行するための計算
- メモリ管理、CPU、GPUの資源管理等
- Gears OS ではメタ計算は通常の計算とは別の階層のメタレベルに分ける
- メタ計算は Code Gear の接続間で Meta Code/Data Gear を使用して実行



Context

- 接続可能な Code/Data Gear のリストを持っている Meta Data Gear
- Gears OS は Context から 必要な Code/Data Gear をメタ計算(stub Code Gear)で取り出す
- Context は従来のスレッドやプロセスに対応する



Interface

- Gears OS のモジュール化の仕組み
- Interface はある Data Gear と API である Code Gear の集合を表現する
- Queue や Stack 等のデータ構造を仕様と実装に分けて記述できる
- Interface の定義にはAPI の引数群の型、 API 自体の Code Gear の型を記述する
- 定義から stub Code Gear を自動生成する

並列処理の構成

- 今回はInterface を用いて並列処理機構の実装を行った
  - ✦ Task
  - ✦ TaskManager
  - ✦ Worker
  - ✦ SynchronizedQueue

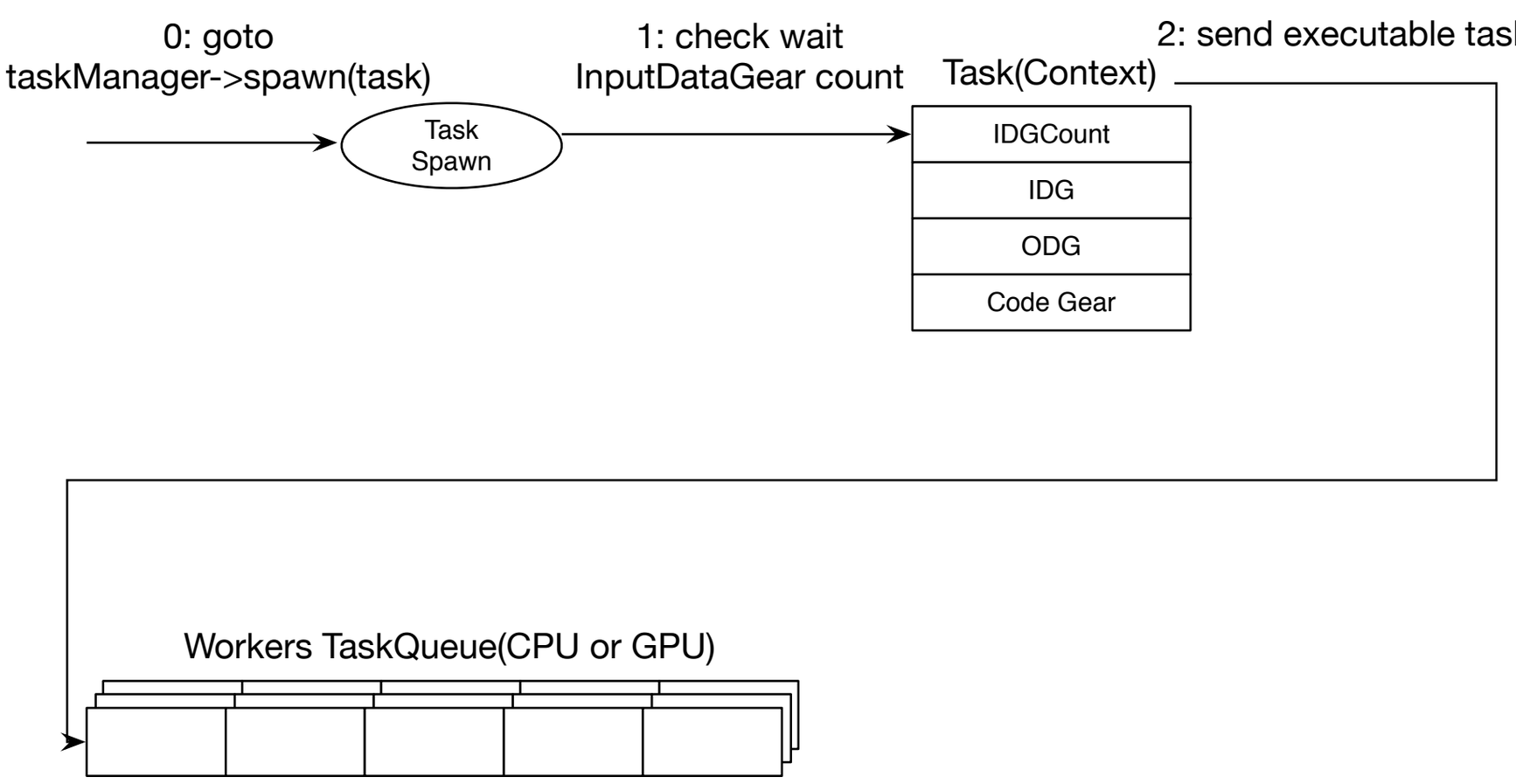
Task

- Gears OS では Context が並列実行の Task に相当する
- Task の生成、実行は **par goto** 構文で行われる
  - ✦ par goto 構文はメタレベルに変換され、 Context の設定を行う

```
__code code1(Integer *integer1, Integer *integer2, Integer
*output) {
    par goto add(integer1, integer2, output, __exit);
    goto code2();
}
```

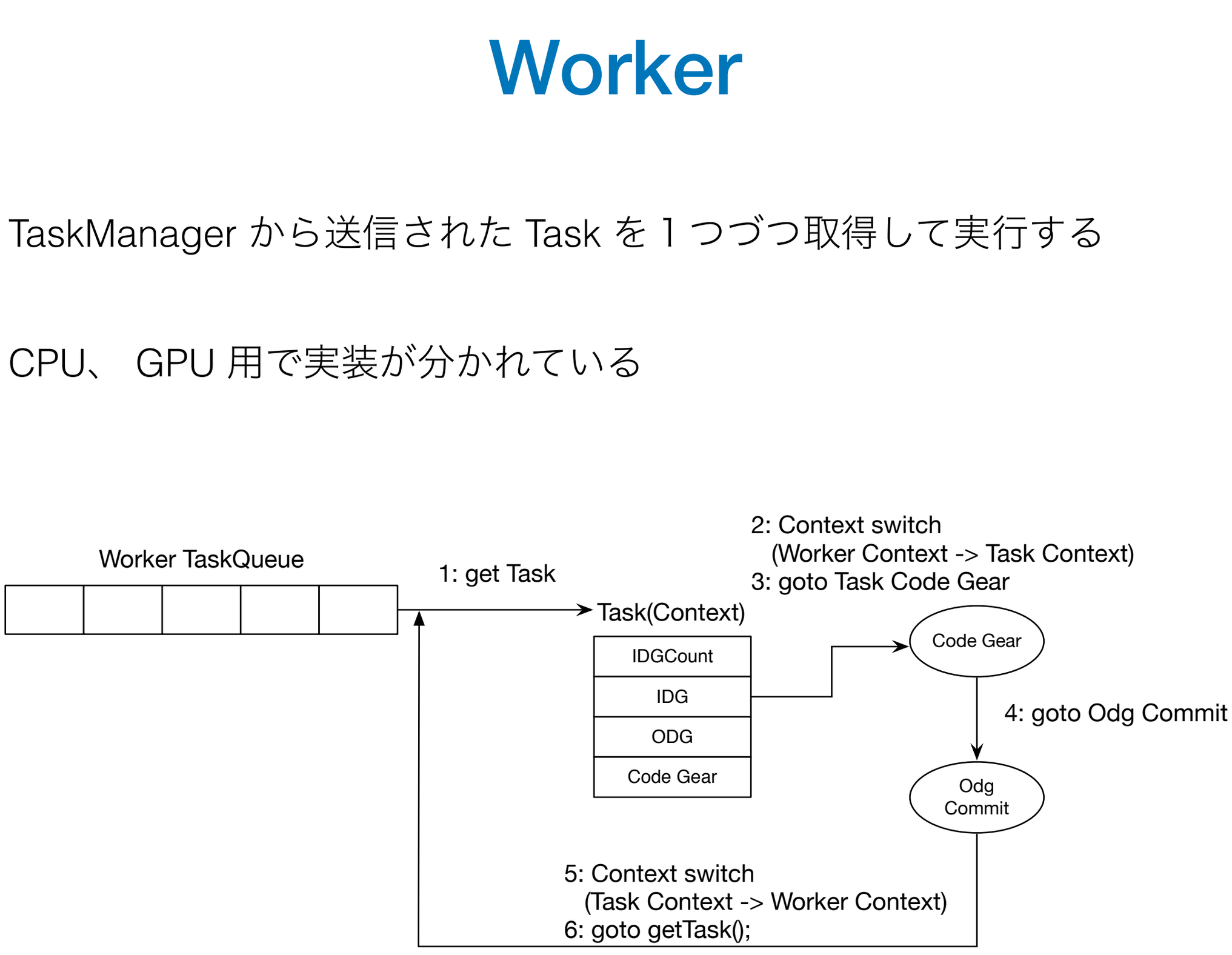
TaskManager

- Worker の生成、管理を行う
- 依存関係の解決した Task を各 Worker の Queue に送信する



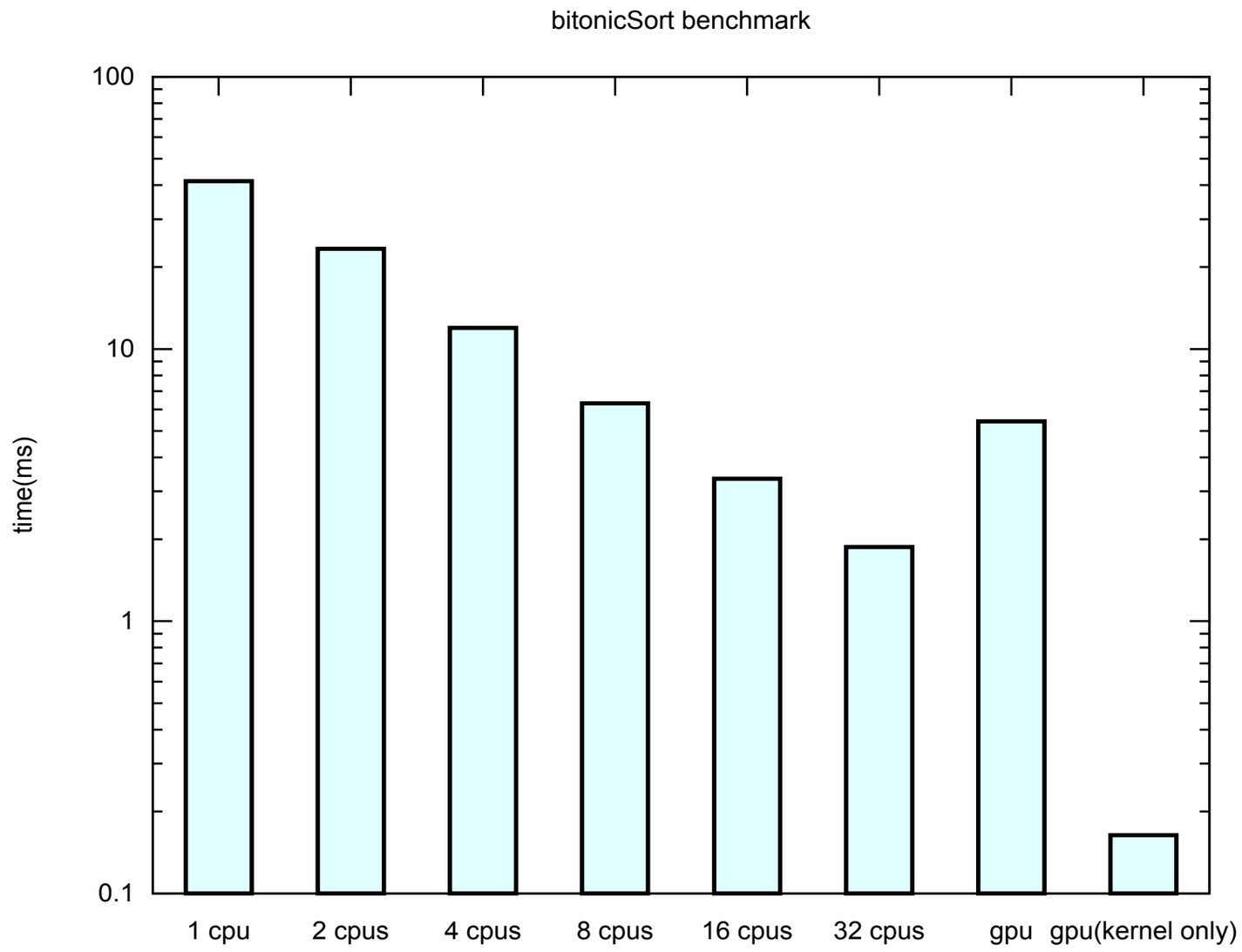
Worker

- TaskManager から送信された Task を 1 つづつ取得して実行する
- CPU、 GPU 用で実装が分かれている



並列処理の測定

- 並列処理の例題として BitonicSort を実装、測定を行った
- 1 CPU と 32 CPU では 約**22.12**倍の速度向上



まとめ

- Gears OS の並列処理機構を Interface を用いて実装を行った
- Interface を導入することで、見通しの良い Gears OS のプログラミングが可能になった
- par goto 構文を導入することでノーマルレベルで並列処理の記述が可能になった

今後の課題

- 並列処理の信頼性をメタレベルで保証
  - ✦ 証明支援系の Agda での証明
  - ✦ CbC で記述されたモデル検査器 akasha を用いたモデル検査
- 並列処理の最適化
  - ✦ パイプラインの実装
  - ✦ Gears OS 自体の速度向上