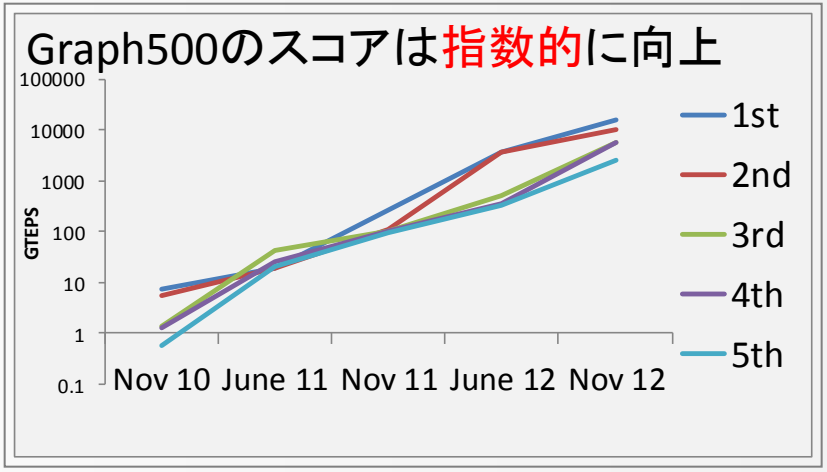
グラフ解析プログラム向けのオブジェクトのメモリレイアウトを整列化できるJavaコンパイラの開発

汐田徹也
千葉研究室

グラフ解析プログラミングの重要性

- グラフ構造：リンクを持った様々なデータを表現可能
  - 道路網や鉄道網
  - タンパク質間相互作用
  - ソーシャルグラフ、ウェブグラフ
- 大規模グラフの解析がますます注目されている
  - E.g. Clustering, Recommendation, Ranking, Optimization

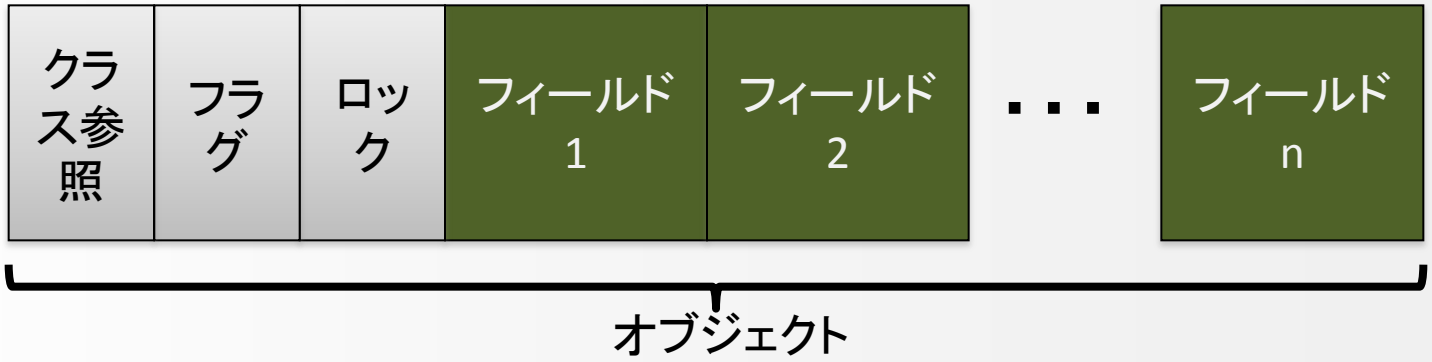


頂点や辺をオブジェクトとする高水準なグラフ表現

- 見通しが良い
  - グラフを巡回して属性を使う処理が直感的に書ける
    e.g. 重み、色、距離、到達フラグ、流量
- 再利用性が高い
  - 巡回アルゴリズムと各オブジェクトでの計算が分離
  - 属性を追加しても、アルゴリズム実装を再利用できる

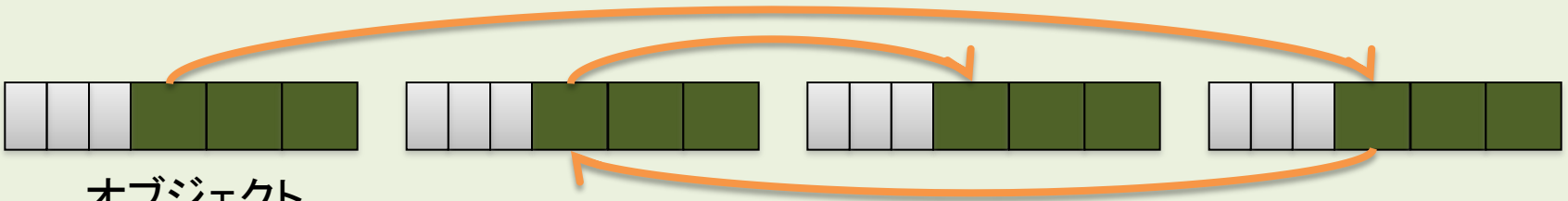
非効率なメモリレイアウトが問題 (1/2)

- Javaでのグラフオブジェクトのレイアウト
  - オブジェクト毎にグラフ属性(フィールド)はまとめられる
  - JVMが管理、ユーザは直接変更できない



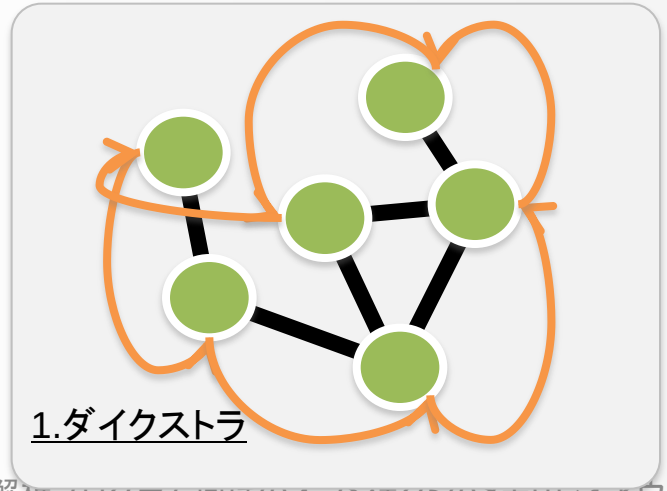
非効率なメモリレイアウトが問題 (2/2)

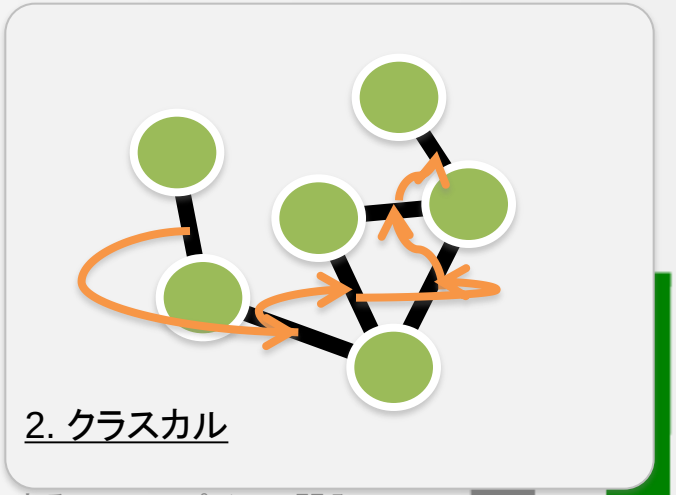
- Javaでのグラフオブジェクトのレイアウト
  - オブジェクト毎にグラフ属性(フィールド)はまとめられる
  - GCが管理、ユーザは直接変更できない
- グラフを巡回した計算でキャッシュミスが多発
  - 例：全頂点を巡回してフィールド1のみを参照する計算



目的:参照の局所性を改善して高速化

- 実行するアルゴリズムに合わせてメモリレイアウトを変更したい
  - アルゴリズムによってアクセスパターンが異なり最適なメモリレイアウトが一意に決められない
  - 例えば
  1. 素朴なダイクストラ法では頂点の到達フラグを逐次的に参照
  2. クラスカル法では辺の重みとその端点を連続的に参照



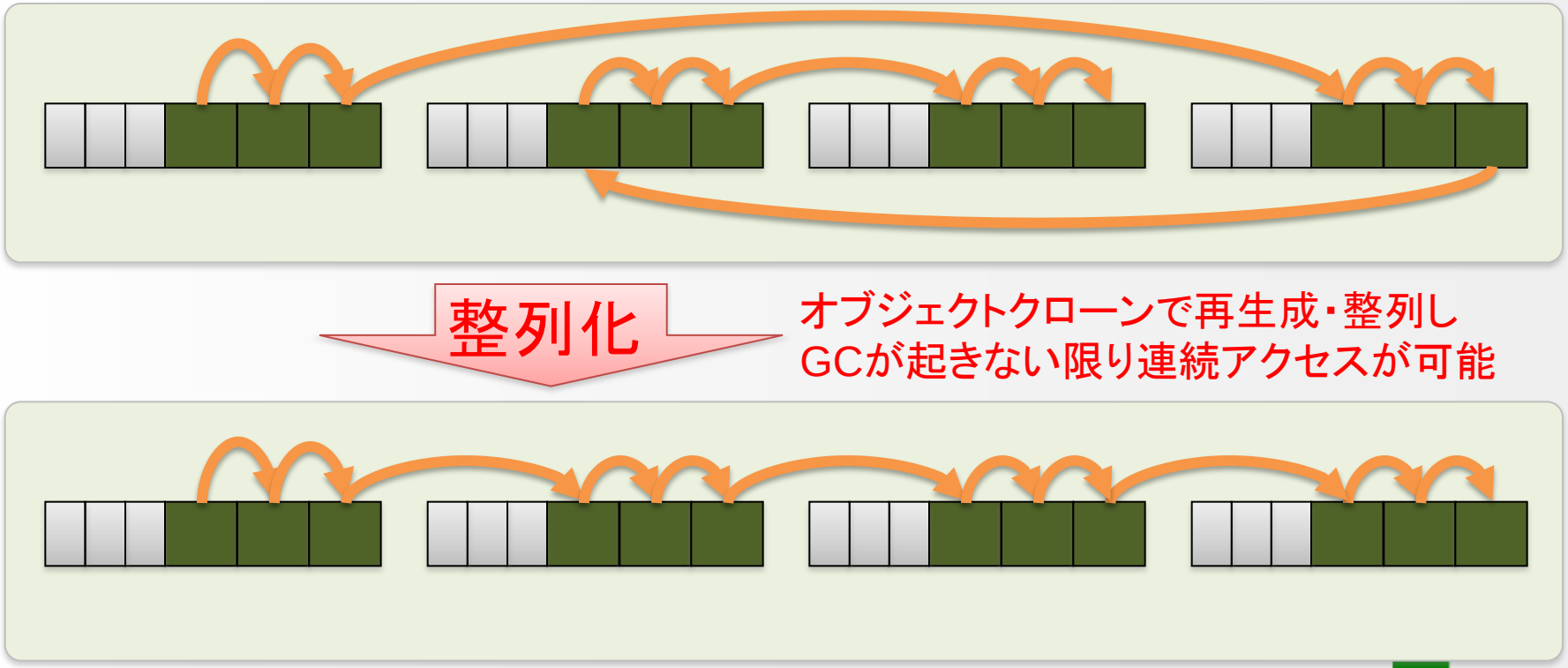


提案: アノテーションにしたがって メモリレイアウトを自動変換できるコンパイラ

- オブジェクト指向に基づくグラフ構造のメモリレイアウトを変換
  - 実行時のデータ整列化を指定できるアノテーション
    - アルゴリズムに合わせてメソッドごとに
  - 整列するフィールドやオブジェクトをアノテーションで指定
  - Java向けにコンパイラ実装
- 扱えるレイアウト変換
  - フィールド配列化 : フィールドを配列化してまとめて配置する
  - オブジェクト整列化 : ユーザ指定順序のオブジェクトを配置する

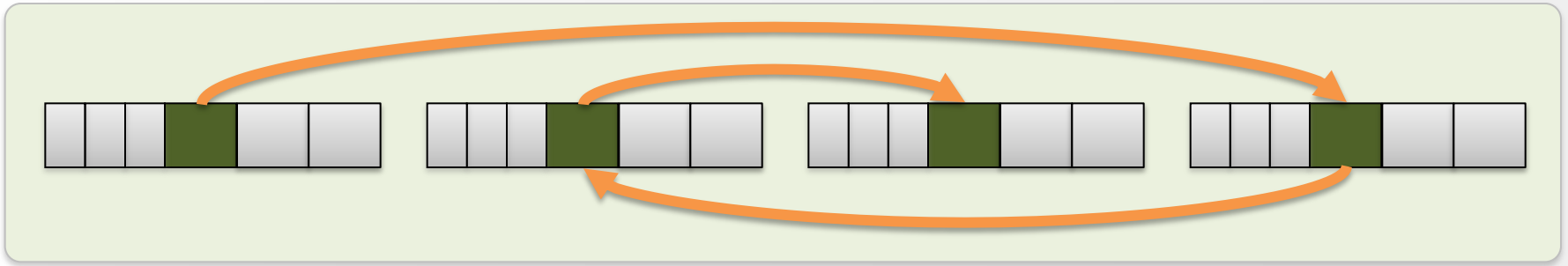
オブジェクト整列化

- 参照順序に沿ってオブジェクトを整列する



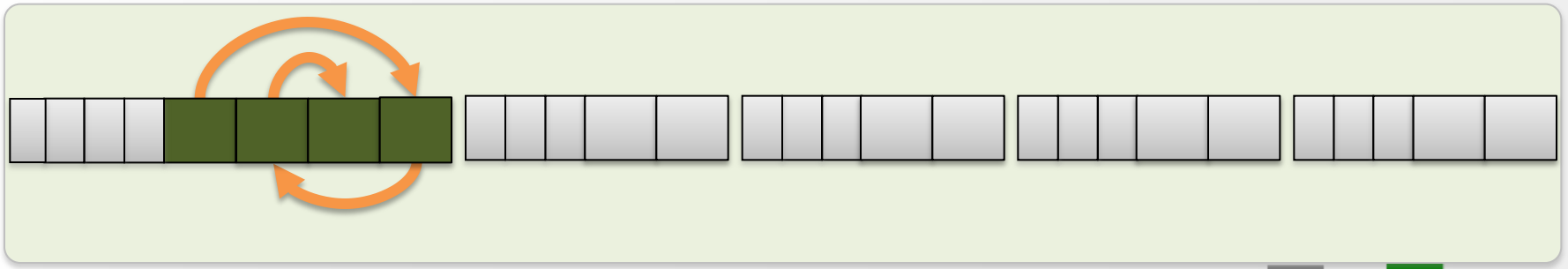
フィールド配列化

- 参照が集中するフィールドを抜き出す

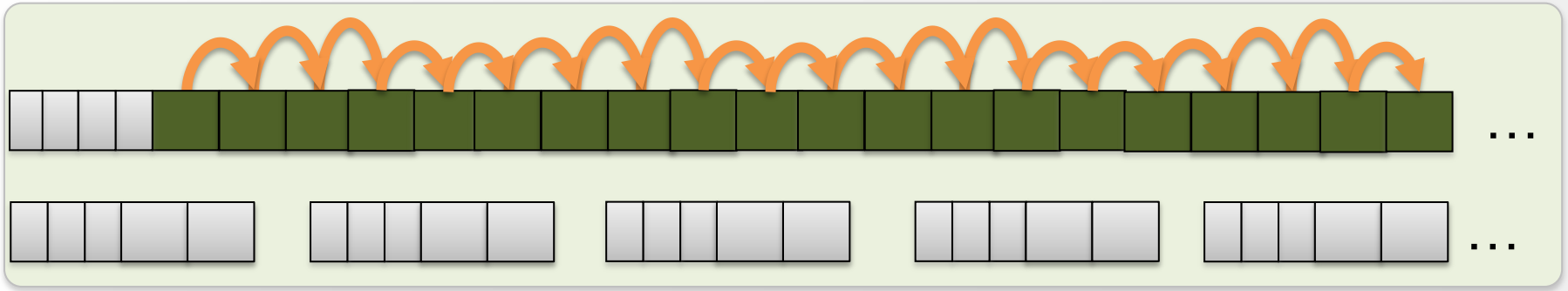


配列化

クラス定義や参照元をコード変換



フィールド配列化+オブジェクト整列化



データレイアウトを変換できるコード例

```
@Target
class Vertex{
   @Arraying
   public boolean visited;
   public int dist;
   private List<Edge> edges;
   /* 略 */
}
```

変換対象クラスの指定

配列化対象フィールドの指定

データレイアウトを変換できるコード例

オブジェクトの整列化

整列化の順序を指定

```
@Reorder(classes={"Vertex"},orders={"g.allVertices()"})
static int dijkstra(Graph g, Vertex src, Vertex dst){
    for(Vertex v : vertices){
        v.dist = inf;
        v.visited = false;
    }
    /* 略 */
}
```

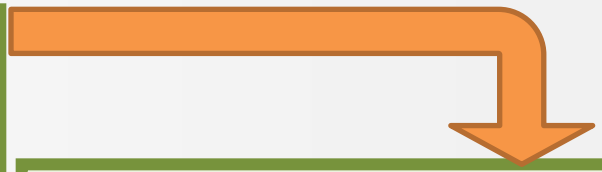
実装方法

- 拡張コンパイラで静的コード変換 (JastAddJ利用)
  1. @Target/@Arraying付きクラスの変換
  2. @Reorder可能な実行時オブジェクト表現
  3. 参照元コードの変換
  4. @Reorder付きメソッドの変換
- 実装上のチャレンジ
  - 標準のVMが使える
  - アルゴリズムに合わせて実行中に整列化が可能
  - ガベージコレクション可能
  - 実際に実行速度が改善する

@Target/@Arraying付きクラスの変換

- クラス定義の変換
  - 配列化対象のフィールドはオブジェクトから除去
  - 外部でまとめて配列として管理

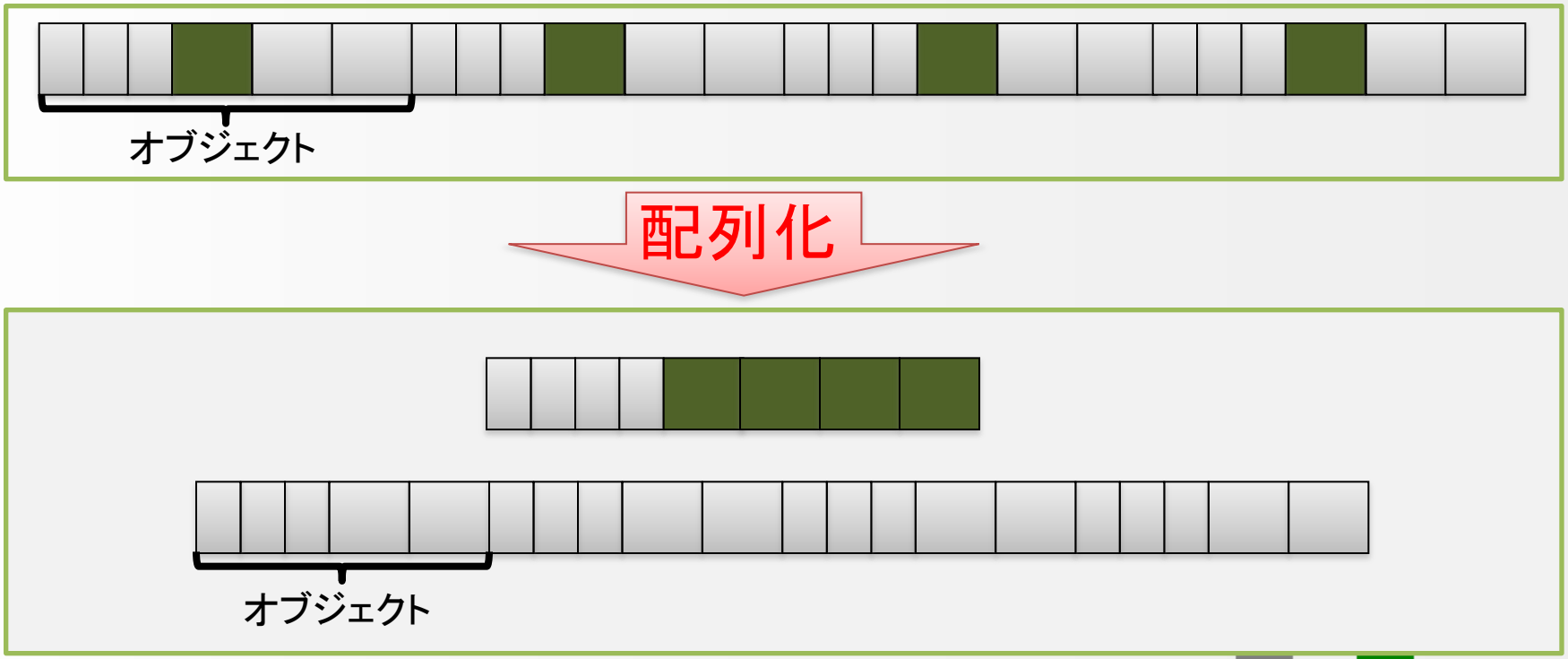
```
@Target
class Vertex{
 @Arraying
 public boolean visited;
 public int dist;
 private List<Edge> edges;
 /* 略 */
}
```



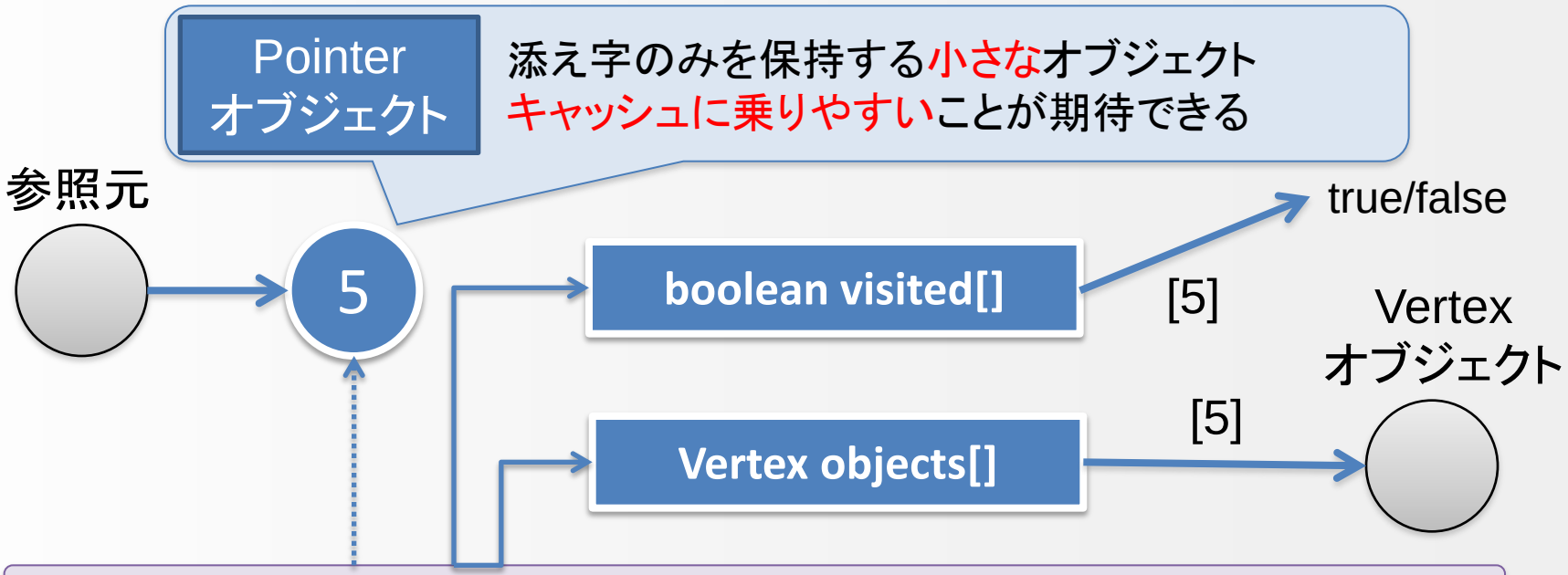
```
class Vertex{
 static public boolean visited[];
 public int dist;
 private List<Edge> edges;
 /* 略 */
}
```

@Target/@Arraying付きクラスの変換

- クラス定義の変換
  - 対象のフィールドはオブジェクトから除去
  - 外部でまとめて配列として管理



@Reorder可能な実行時オブジェクト表現



Vertexクラス
・Pointerオブジェクトを弱参照により監視
・必要に応じて元オブジェクトや配列化したフィールドを解放
・フィールド配列や元オブジェクト配列を整列化

参照元コードの変換

- アクセス部のコードを変換

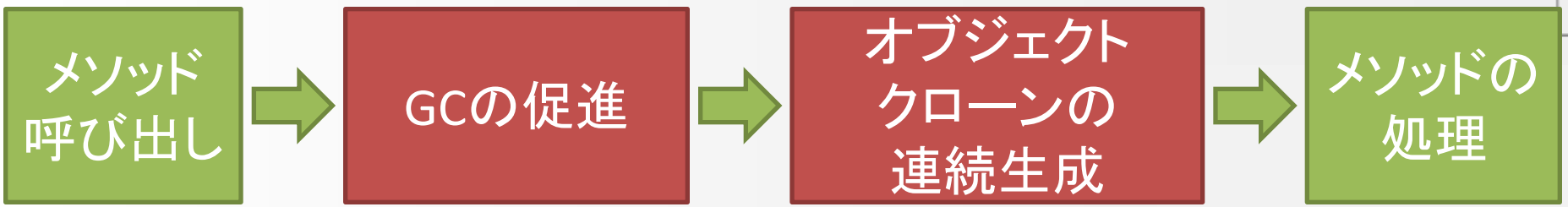
```
Vertex src = new Vertex();
src.visited = false; /* 配列化したフィールド */
src.dist = 0;        /* 通常のフィールド */
```

```
Pointer src = new Vertex().getNewPointer();
Vertex.visited[src.id] = false;
Vertex.objList.get(src.id).dist = 0;
```

@Reorder付きメソッドの変換

```
@Reorder(classes={"Vertex"},
orders={"g.allVertices()"})
```

- 整列化するコードを挿入
- 整列化方法
  1. ガベージコレクタを促進
     - ガベージコレクタによる攪乱を防止するため
  2. 順序にしたがってオブジェクトを連続クローン
     - 連続クローンにより連続配置を期待



実験:
グラフ解析処理の性能改善とオーバーヘッドを評価

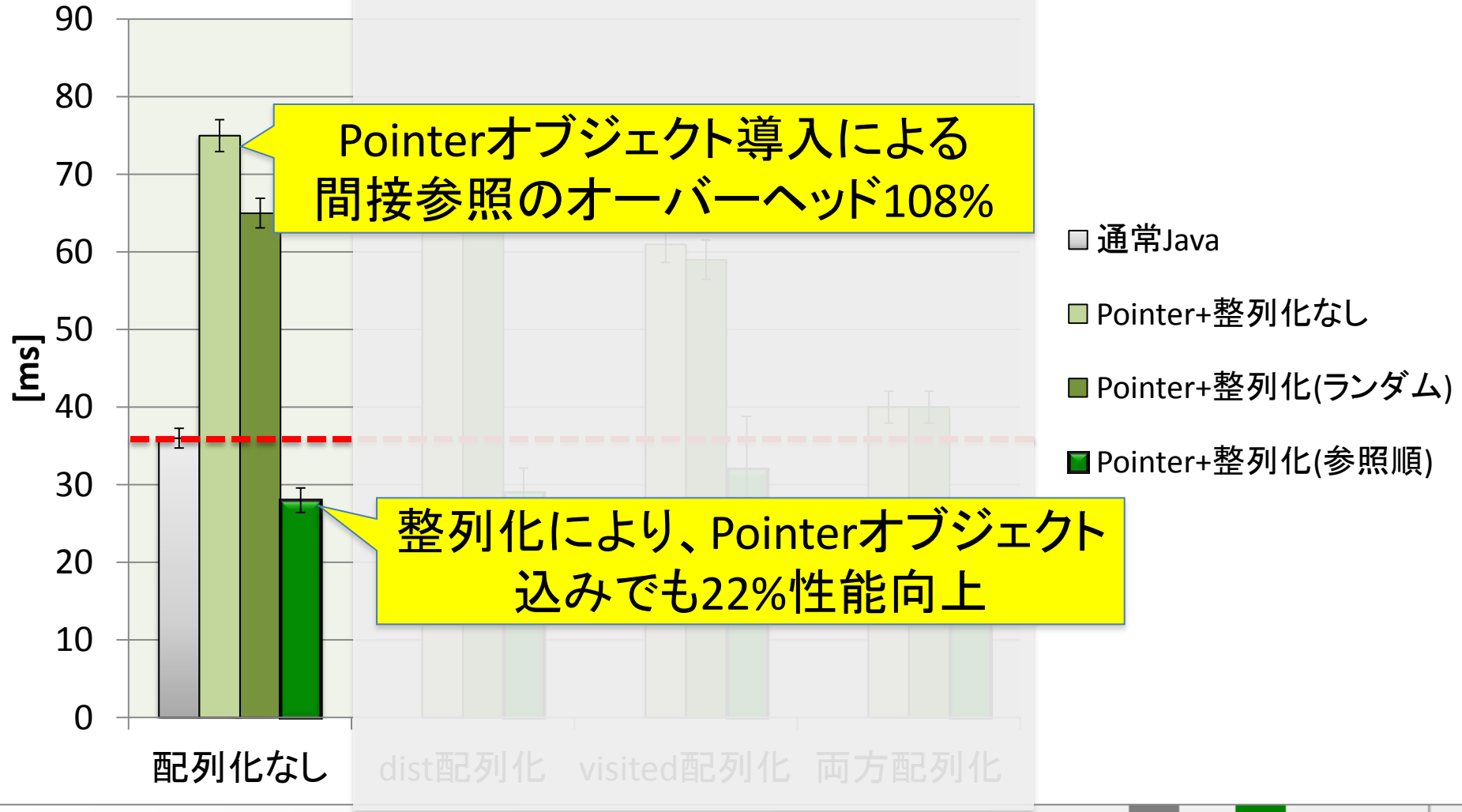
- 実験項目:ダイクストラ法で頂点オブジェクトについて

1. Javaでの通常実装 2. Pointerオブジェクト導入のみ 3. Pointerオブジェクト+整列化(ランダム) 4. Pointerオブジェクト+整列化(参照順)	×	1. 配列化無し 2. 配列化有り

- 入力
  - 疎グラフ: メモリ内に頂点オブジェクトの割合が高いケース
  - 密グラフ: メモリ内に頂点オブジェクトの割合が低いケース
- 実験方法と環境
  - 64クエリを与え実行時間の平均(最後から40回分)
  - FUJITSU FX10 SPARC64 Ixfx 1.848GHz 16 core, RAM32GB
  - OpenJDK 1.6.0_24-b24 with C2 JIT, serial GC

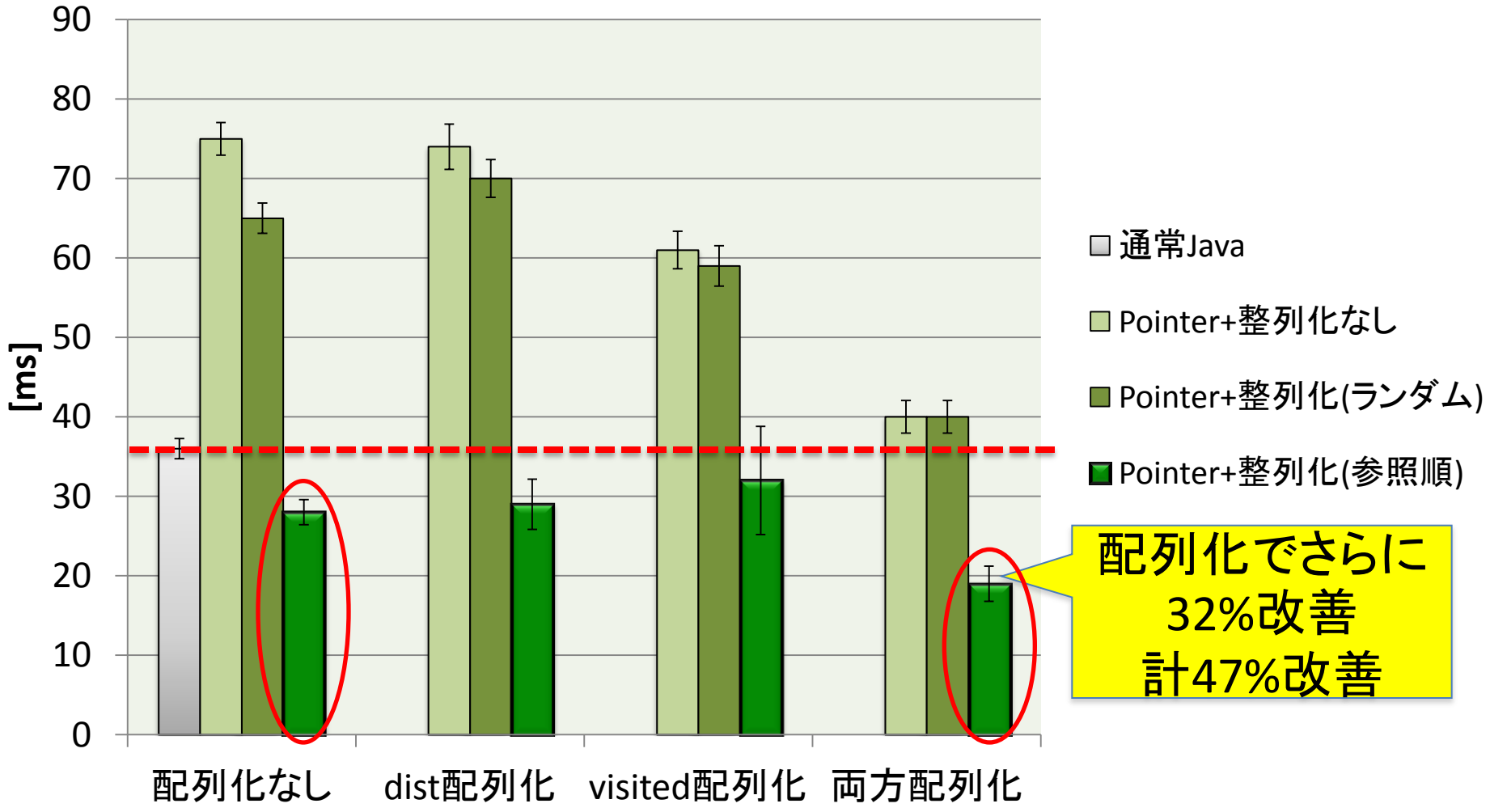
実験結果: 疎グラフ

V=1000, E=V*16のグラフ、Graph500のジェネレータから作成



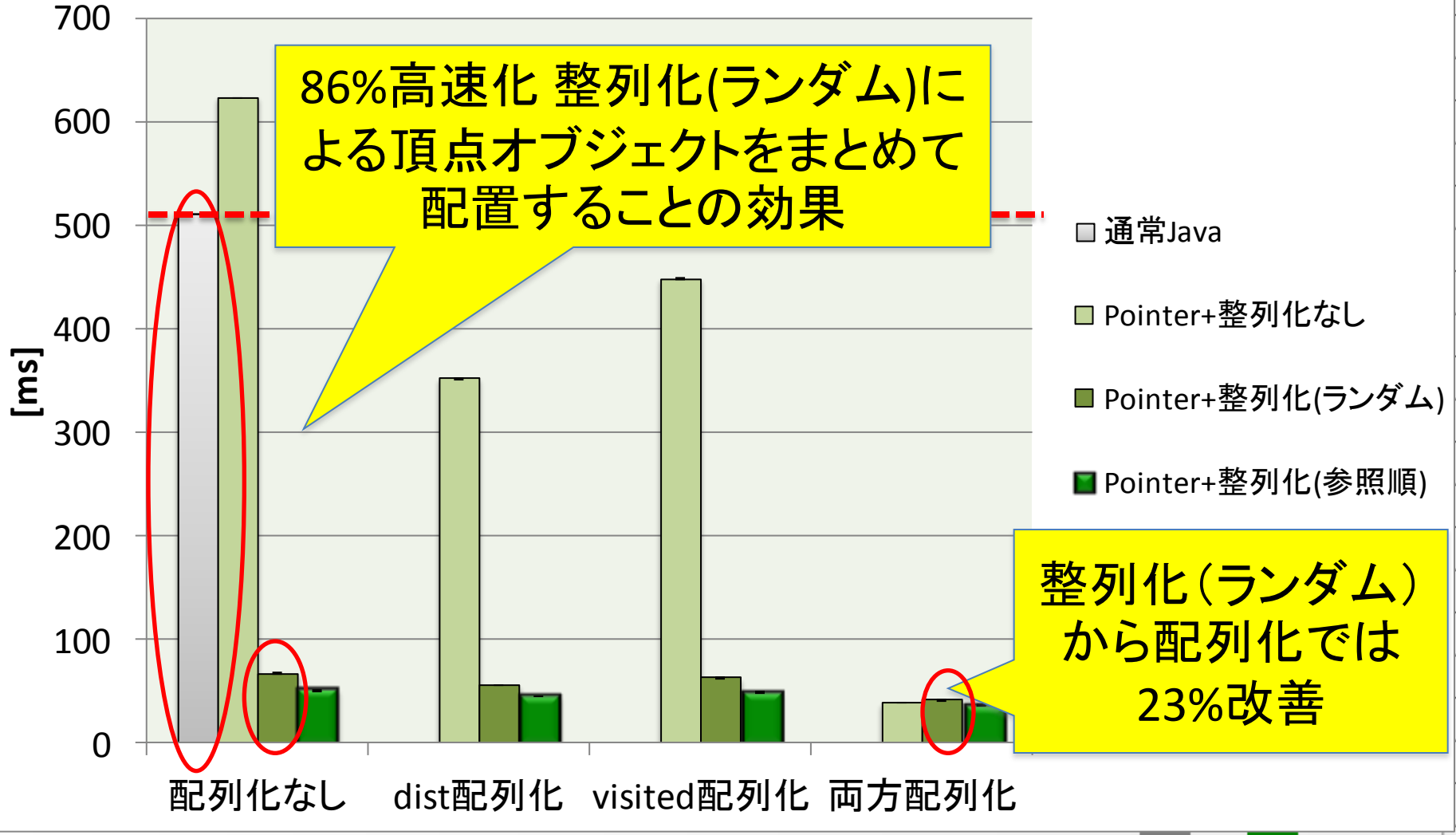
実験結果: 疎グラフ

V=1000, E=V*16のグラフ、Graph500のジェネレータから作成



実験結果: 密グラフ

V=1000,E=V*(V-1)の完全グラフ TSPLIBの座標データから作成



関連研究

- コンパイラ・コード変換などによるデータレイアウト最適化
  - Structure Splitting [Chilimbi et al. '99]
    - Javaでオブジェクトのデータを参照頻度に応じて分割
  - Array Reshaping [Zhao et al. '07]
    - C/C++で構造体の分割や並び替えを行う
  - CのマクロによるAoS, SoA変換 [Strzodka '12]
- Copy GC時のデータレイアウト最適化
  - Object Reordering [Huang et al. '04]
    - 参照頻度の高いオブジェクト
- グラフ向けDSL
  - Green-Marl [Hong et al. '12]
    - メソッドローカルフィールドなどサポート

まとめ

- OOPプログラムをアノテーションにしたがって性能を考慮したメモリレイアウトに自動変換するコンパイラを開発した
  - フィールド配列化、オブジェクト整列化に対応
- 実装上のチャレンジ
  - JVMの改造をしない（ポジティブ
  - 実行中のオブジェクト整列化が可能
  - ガベージコレクション可能
- ダイクストラ法を用いた実験で実行速度が向上することを確かめた

発表活動

- 第15 回ソフトウェア科学会　プログラミングおよびプログラミング言語ワークショップPPL2013 2013 年3 月(ポスター)
- MODULARITY: aosd・13 - Aspect-Oriented Software Development. 2013年3月. (ポスター)
- 第98回情報処理学会　プログラミング研究会 (PRO) 論文誌投稿予定 2014年3月