指定されたメソッドの呼び出し中でオブジェクトのメモリレイアウトを再整列化できるグラフ解析向けJavaコンパイラ

汐田徹也　佐藤芳樹　千葉滋
東京大学大学院
情報理工学系研究科

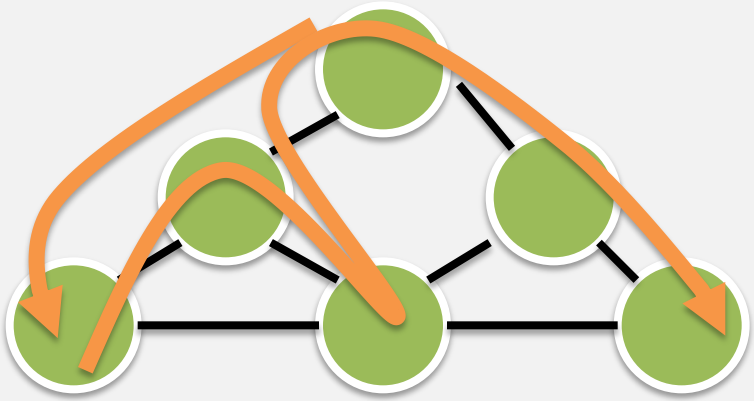
グラフ解析プログラミングの重要性

- 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
- 高速なグラフ解析プログラムを簡単に記述することの需要
  - グラフ解析向けDSLやフレームワークの登場
    - Pregel [Malewicz et al.'10]
    - Green-Marl [Hong et al.'12]

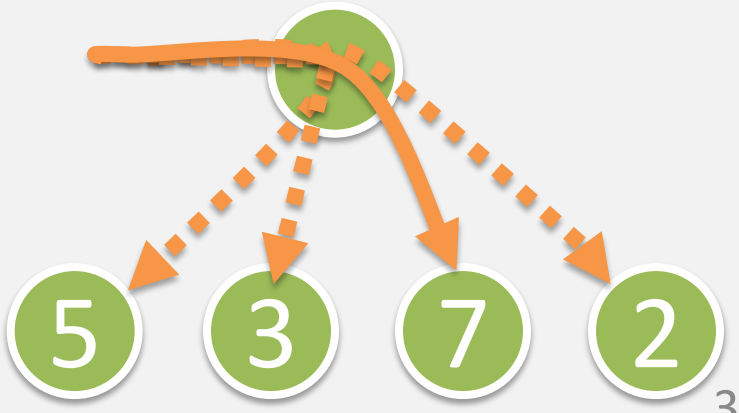
頂点や辺をオブジェクトとする高水準なグラフ表現

- 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
  - 例： 巡回アルゴリズムと各頂点での計算を分離

巡回アルゴリズム
例：深さ優先探索

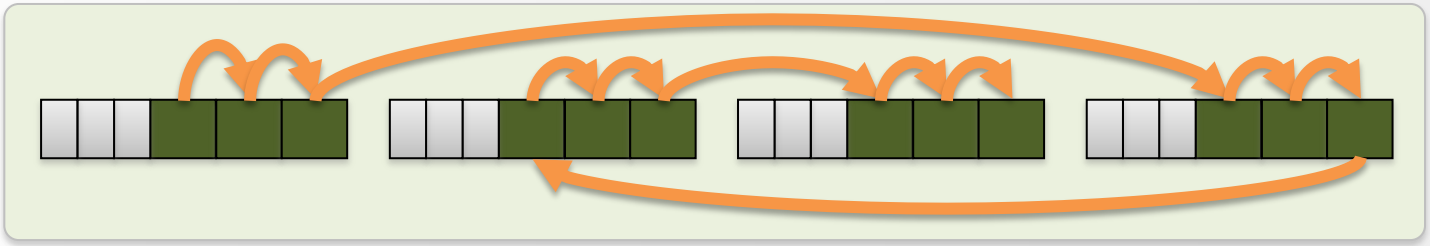


各頂点の計算
例：次の頂点の選択アルゴリズム

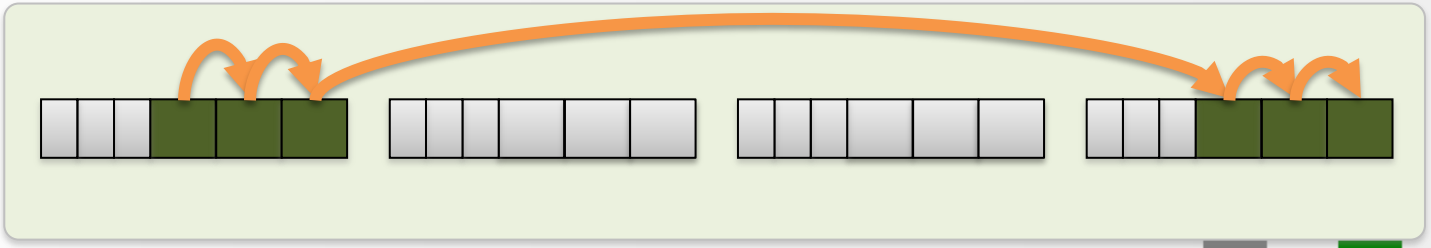


Javaによるオブジェクト指向実装では非効率なメモリレイアウトが問題

- グラフの頂点を巡回しながら行う計算でキャッシュミスが多発する可能性
  - 巡回する順に頂点オブジェクトが整列されていない



  - 巡回対象の頂点オブジェクトが非連続



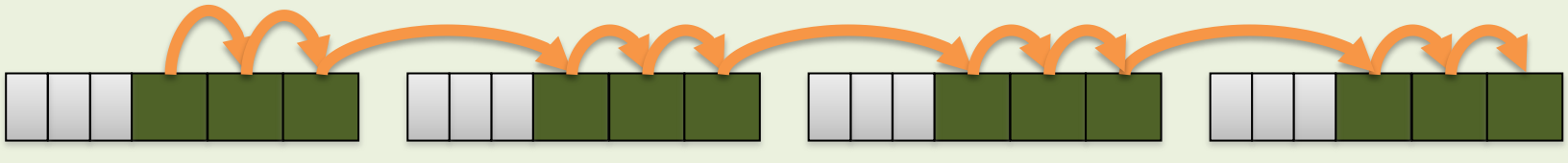
Javaでは性能の良いオブジェクトレイアウトを知っていても反映させることが難しい

- オブジェクトをどのような順でメモリ上に配置するか開発者が指定できない
  - JavaVMがすべて管理
  - C言語の構造体配列のようなものはない
- 初期の整列順がプログラム終了まで保たれるとは限らない
  - GC (JVM)がコンパクション時にオブジェクト整列順序を変更

ユーザが指定した順序に倣ってオブジェクトを“再”整列化できるJavaコンパイラ

- グラフ解析直前に“再”整列
  - GCによるかく乱を修復
- 順序の指定方法：アノテーション
  - プログラム中のユーザが指定したオブジェクトのリストの順序を受け取る
  - メソッドごとに指定し、その実行の直前で順序を反映する

再整列化されたオブジェクトへのアクセス



実現方法:
拡張コンパイラによる静的コード変換

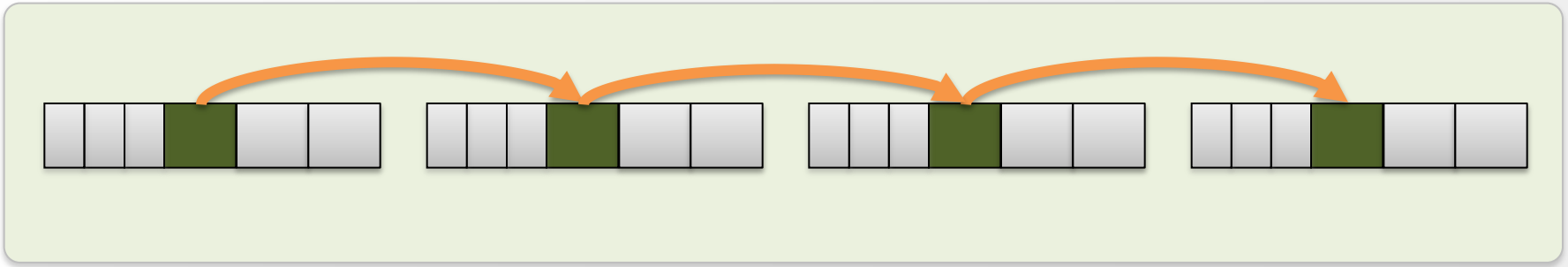
- 再整列化方法:連続クローン
  - 連続的に生成されたオブジェクトはその順序でメモリ内で配置される傾向を利用
  - Object#clone()を利用
  - OpenJDK1.6で傾向を発見
- コード変換
  - オブジェクトを連続クローンするコードを挿入
  - 再整列可能なオブジェクト表現への変換
  - オブジェクトの生成や参照方法の変換

Javarac:
Java Reordering and Arraying Compiler

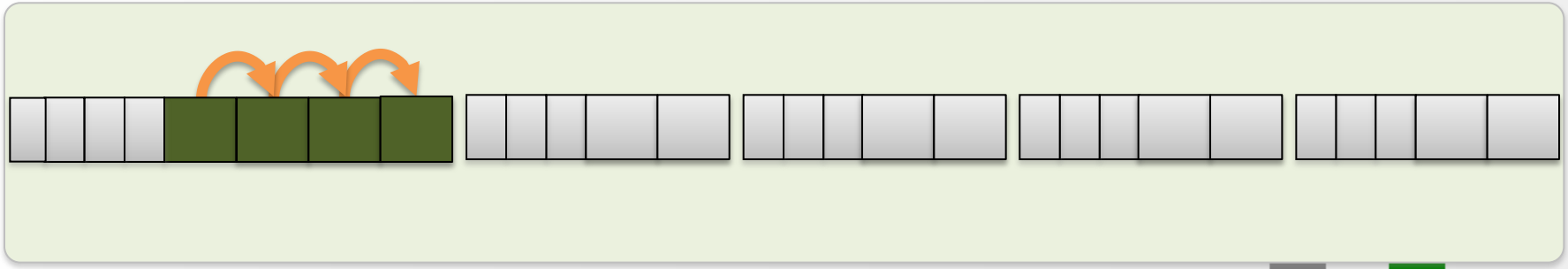
- Javarac
  - アノテーションを解析し、Javaコードを静的変換
  - JastAddJ（拡張可能なJavaコンパイラ）を拡張
  - バックエンドにJavacを使用
- 機能
  - オブジェクトの再整列化
  - フィールドの配列化

さらに、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
  - 開発者がアノテーションで指定



配列化



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;
    /* 略 */
}
```

変換対象クラスの指定

配列化対象フィールドの指定

整列化の順序を指定

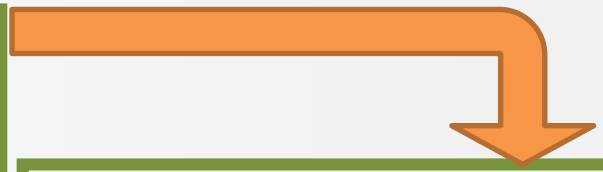
```
@Reorder(classes={"Vertex"},orders={"g.allVertices()"})
stati  int dijkstra(Graph g, Vertex src, Vertex dst){
    fo  Vertex v : vertices){
        ist = inf;
        ited = false;
    }
}
```

オブジェクトの整列化を指定

@Arraying付フィールドの変換

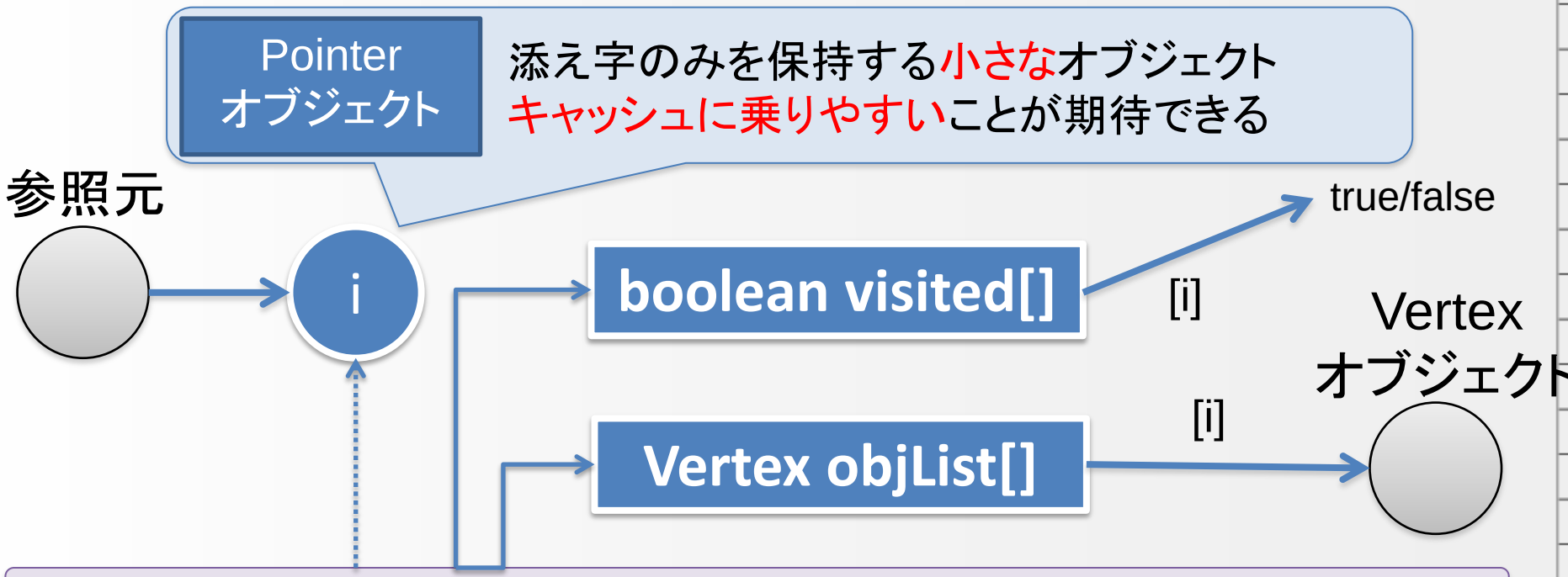
- クラス定義の変換
  - 配列化対象のフィールドはオブジェクトから除去
  - 外部でまとめて配列として管理

```
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 */
}
```

再整列化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 */
```

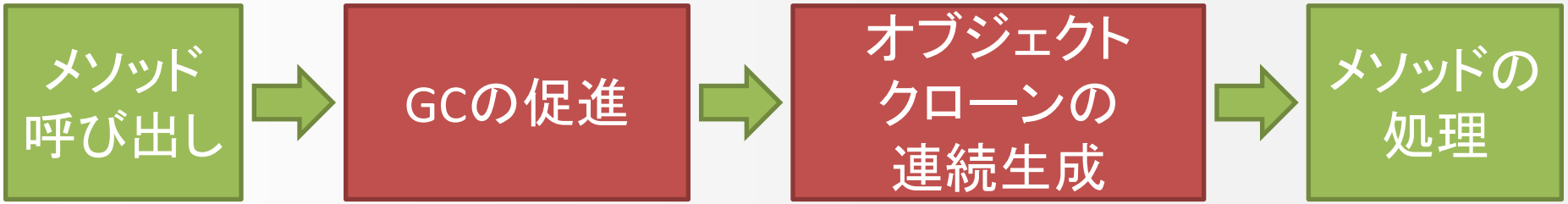
```
Pointer src = new Vertex().getNewPointer();
Vertex.visited[src.id] = false;
Vertex.objList.get(src.id).dist = 0;
```

ポインタオブジェクトの効果

- オブジェクト再生成による参照の更新の効率化
  - 添え字の更新とオブジェクトリスト(objList)の参照の更新のみで実現
- メモリ解放の実現
  - 弱参照を活用
- 低オーバーヘッド
  - ポインタオブジェクトはキャッシュに乗りやすい
    - 小さなオブジェクトである

@Reorder付きメソッドの変換

- 再整列化するコードを挿入
- 再整列化方法
  1. GCを促進
     - GC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



OpenJDK 1.6で実験:
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. Pointerオブジェクト導入のみ
2. Pointerオブジェクト+再整列化(ランダム)
3. Pointerオブジェクト+再整列化(参照順)

×

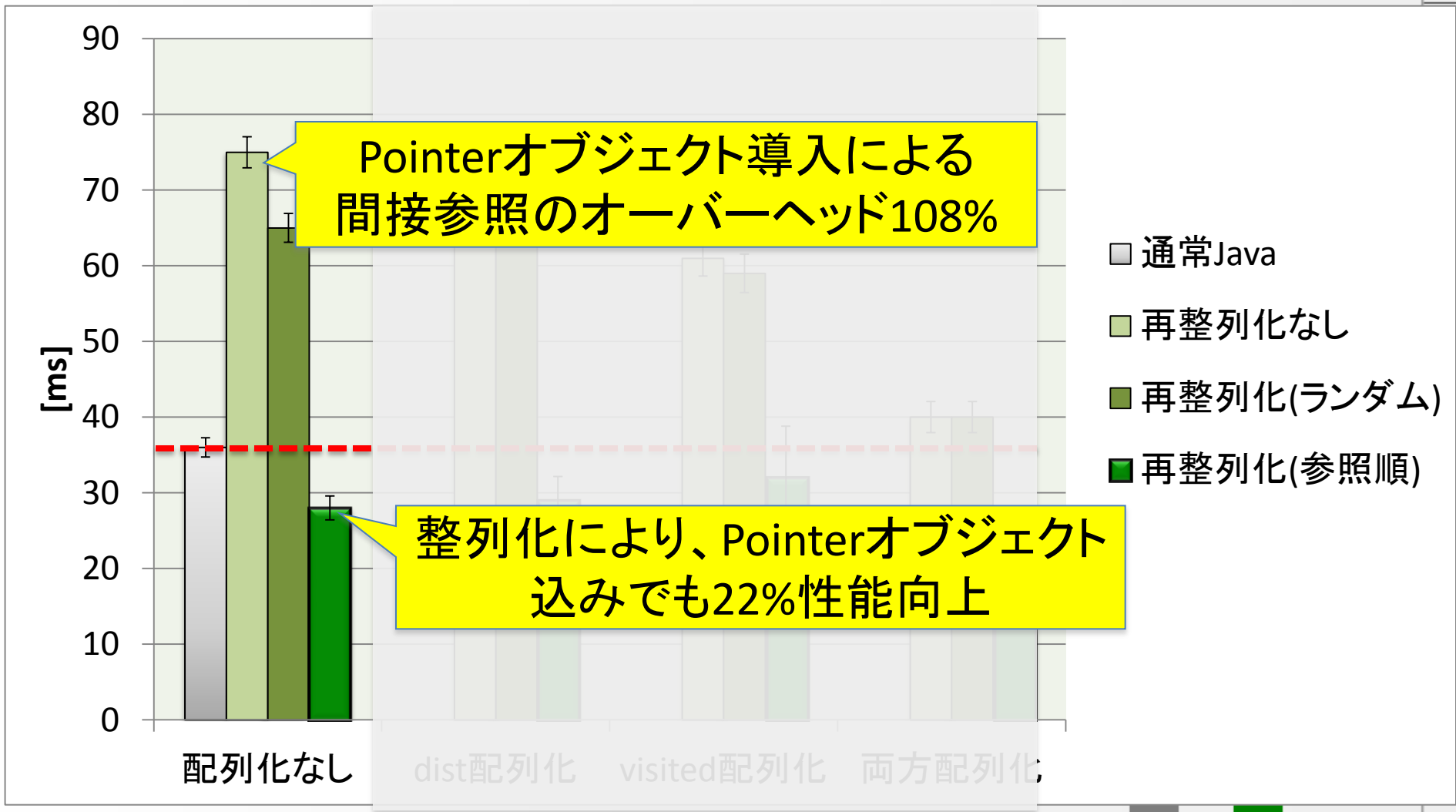
1. 配列化無し
2. distを配列化
3. visitedを配列化
4. distとvisitedを配列化

を変化させたJavaracと通常のJavaコンパイラを比較

- 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
    - 頂点オブジェクトへの連続アクセスの計算時間割合も低い
- 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)
  - GCにかかる時間は除去
  - FUJITSU FX10 SPARC64 Ixfx 1.848GHz 16 core, RAM32GB

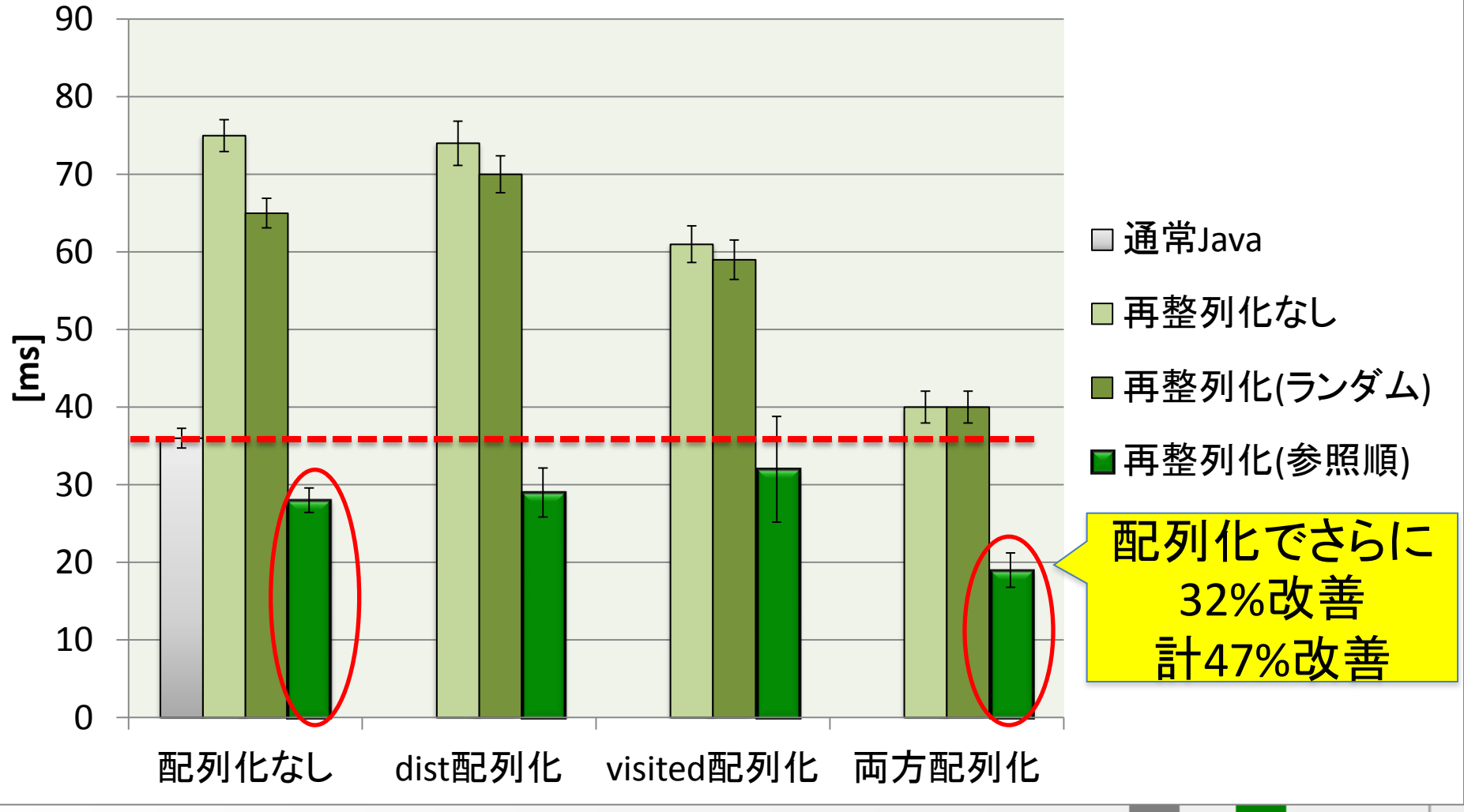
実験結果: 疎グラフ

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



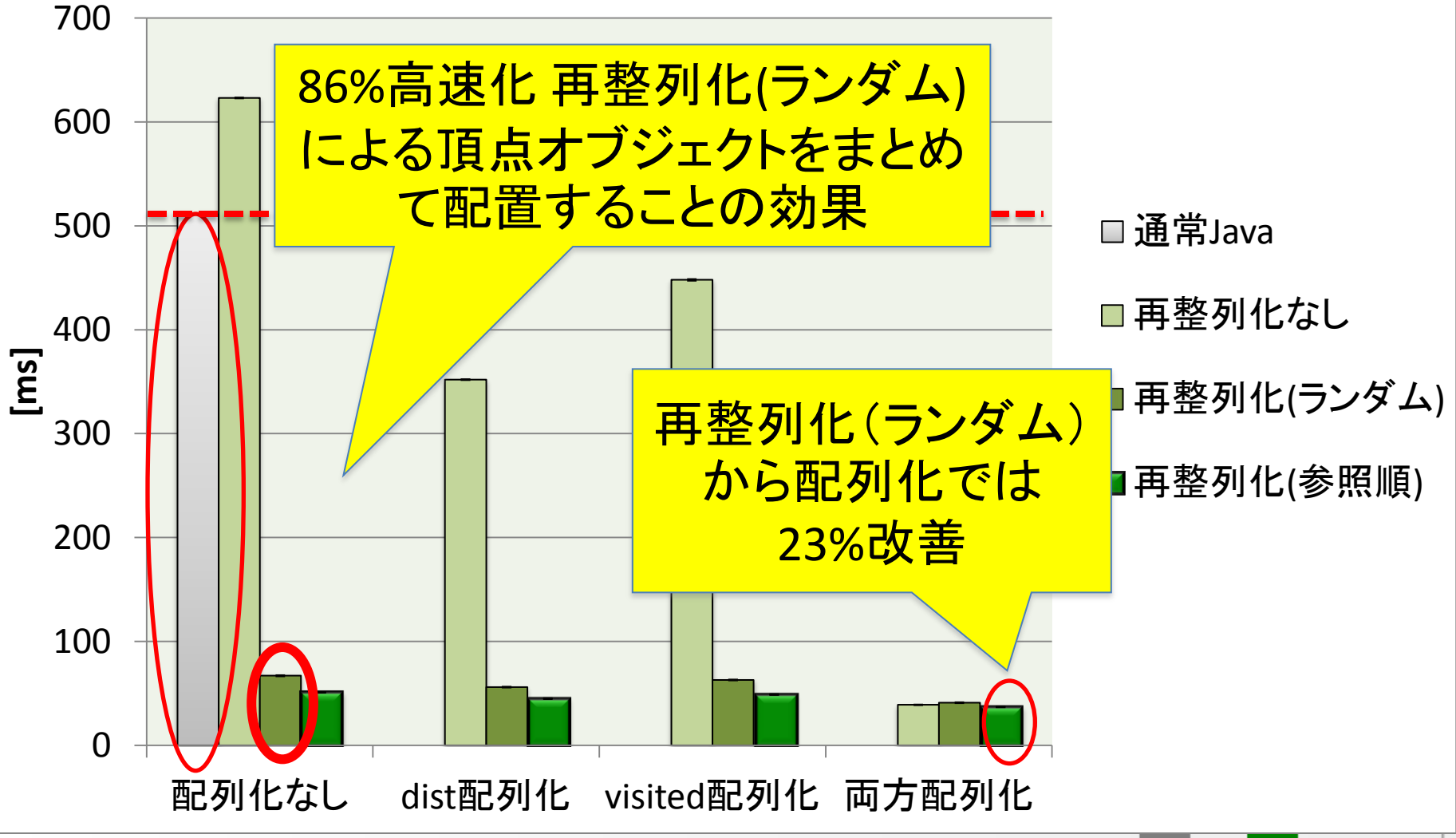
実験結果: 疎グラフ

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

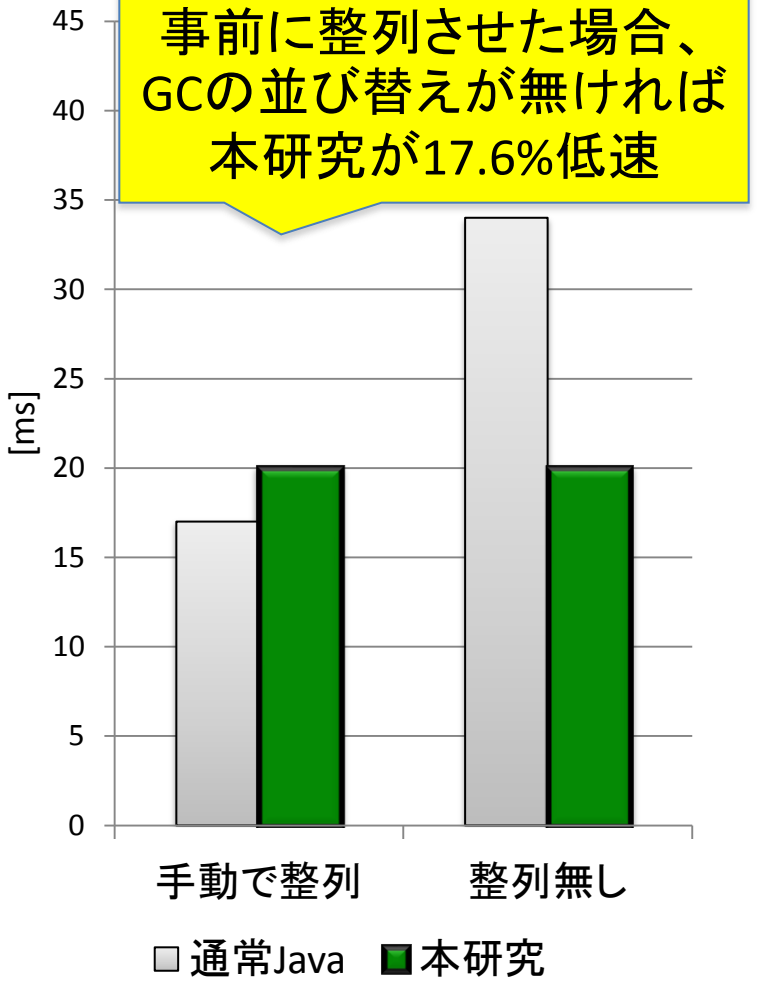


実験結果: Concurrent Mark&Sweep GC

通常のGCと違い自動並び替えが置きにくい

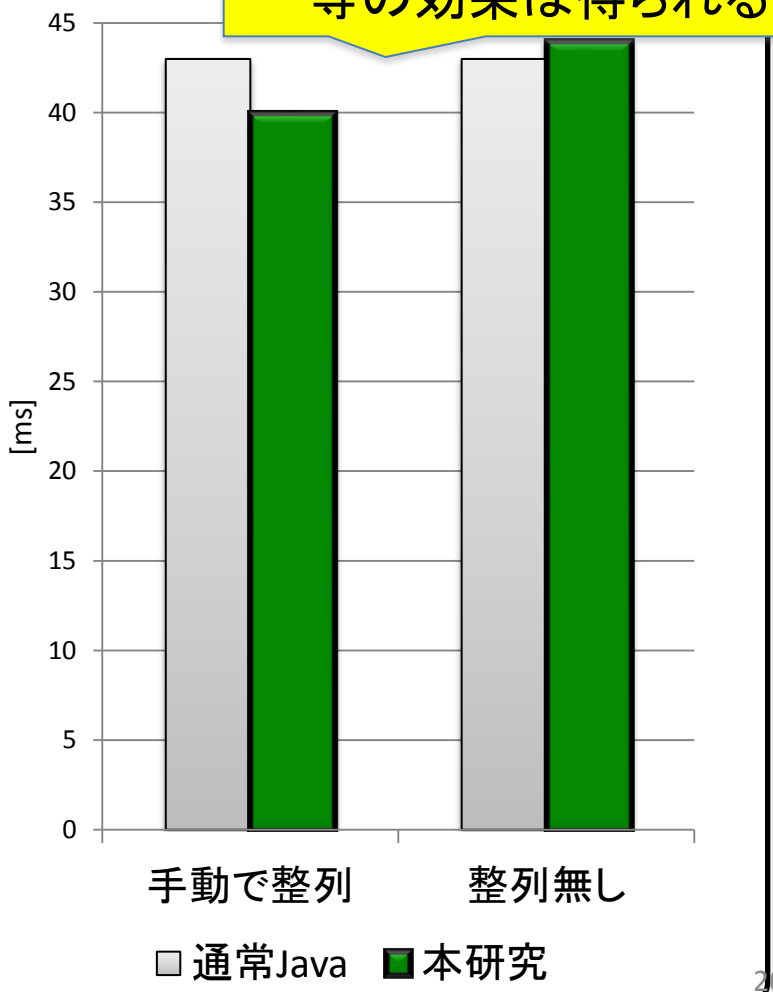
疎グラフ

事前に整列させた場合、GCの並び替えが無ければ本研究が17.6%低速



密グラフ

手動整列でも本研究と同等の効果は得られる



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]
- 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をあつめる
  - Custom Object Layout [Novark et al. '06]
    - JVMを改変し、GC時の並び替えアルゴリズムを実装可能に
- 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Javaracを開発した
  - フィールド配列化、オブジェクト再整列化に対応
- アプリケーションレベルでGCやJVMを制御
- 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
- 現在の制限
  - クローン可能な再整列化できない
  - 分割コンパイル
  - ポリモーフィズムの@Target付クラスでの利用

今後の課題

- OpenJDK1.6以外の環境で再生成による再整列化が実現できるかの検証
- より柔軟なレイアウト変更の実現
  - クラスをまたがるオブジェクトの整列化
  - 複数のフィールドを一つの配列にまとめる変更
- 再整列化を行うタイミング調整機能の実現
  - GCによるコンパクションを検知する
  - 参照速度の低下を検知するなど