

コンパイラ演習

第12回

(2012/01/05)

中村 晃一 野瀬 貴史 前田 俊行
秋山 茂樹 池尻 拓朗
鈴木 友博 渡邊 裕貴
潮田 資秀
小酒井 隆広
山下 諒蔵 佐藤 春旗
大山 恵弘 佐藤 秀明
住井 英二郎

今日の内容

- Garbage Collection (GC, ごみ集め)
 - 参照されなくなったメモリ領域を解放すること
- 配列境界検査

Garbage Collection 概論

- 遠藤敏夫先生の資料を参考にしています
 - <http://matsu-www.is.titech.ac.jp/~endo/gc/gc.pdf>
- 遠藤敏夫先生
 - 東京工業大学大学院 特任准教授
 - <http://matsu-www.is.titech.ac.jp/~endo/>

今回紹介する GC の種類

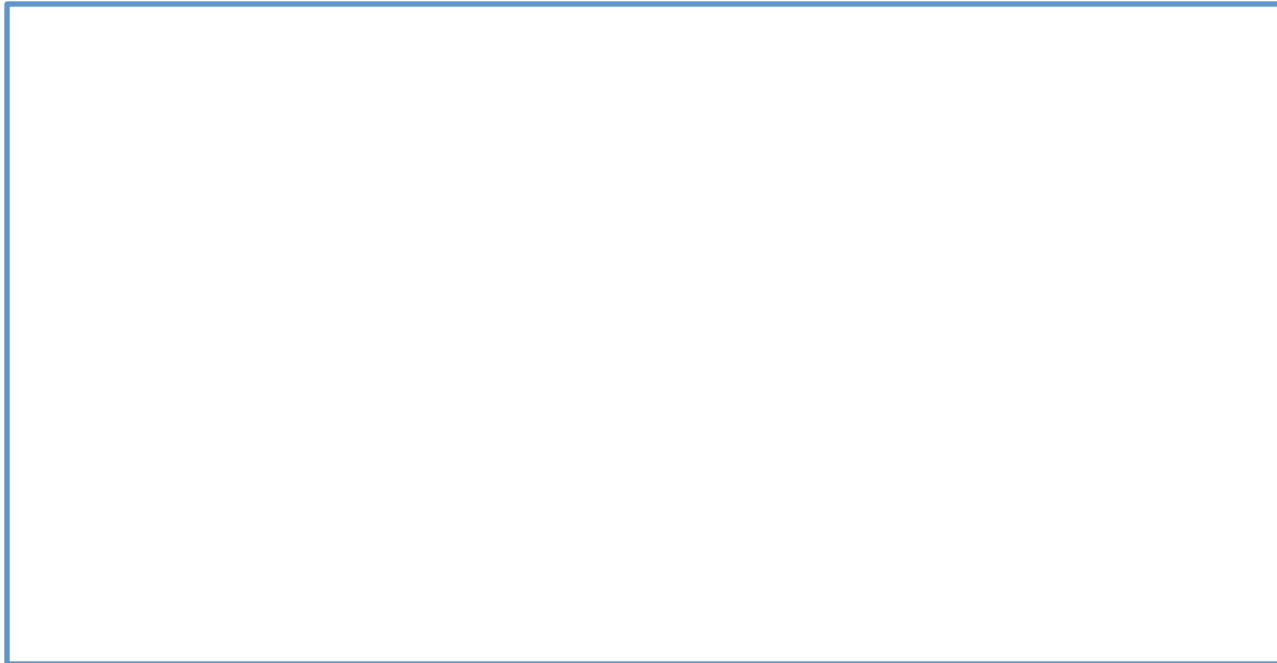
- 参照カウント (reference counting)
- Tracing GC
 - マーク & スweep
 - Copying GC

参照カウント (Reference Counting)

- オブジェクトへの参照 (ポインタ) の数をオブジェクト自身に記録しておく
- 参照を作る/なくすたびにカウントを増減
- カウントが 0 になったらオブジェクトを解放

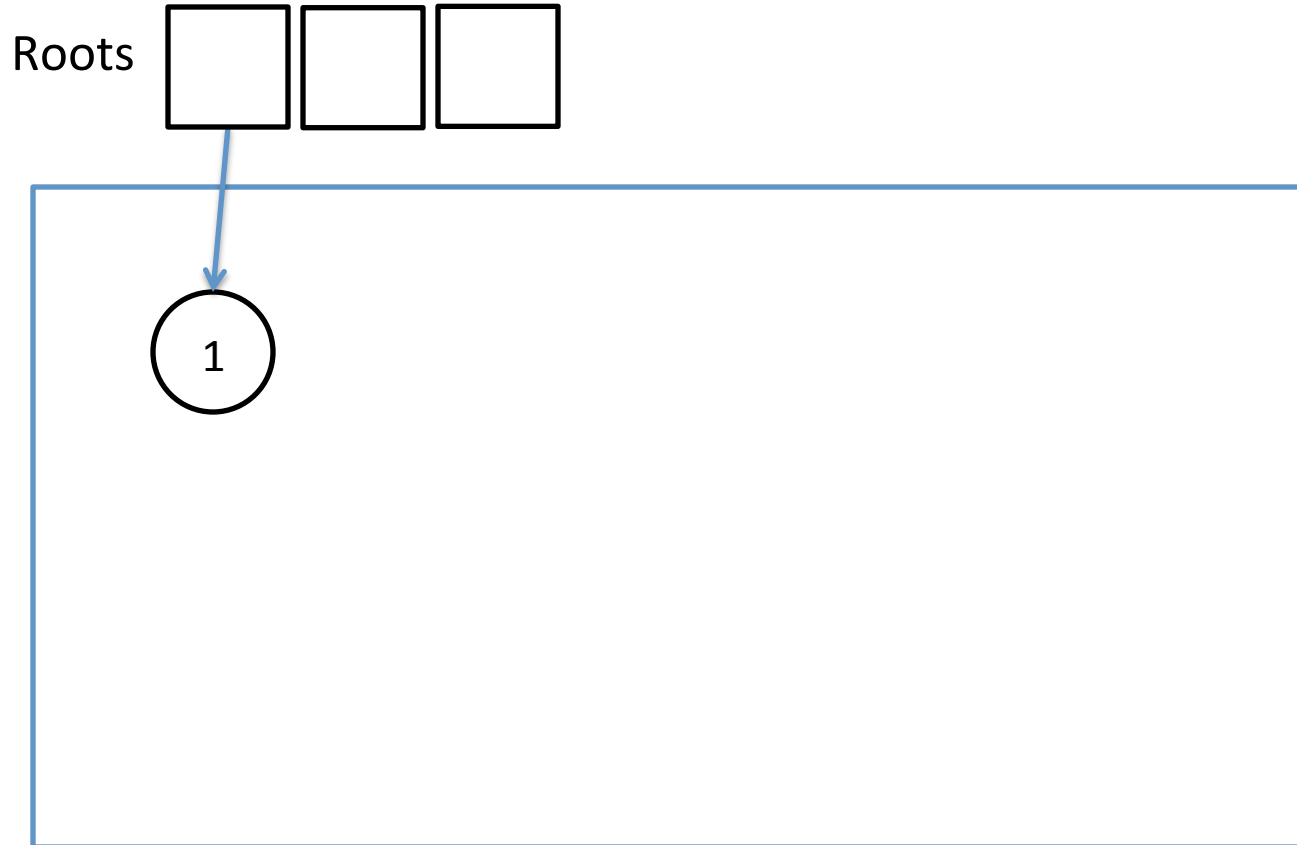
参照カウントの例

Roots

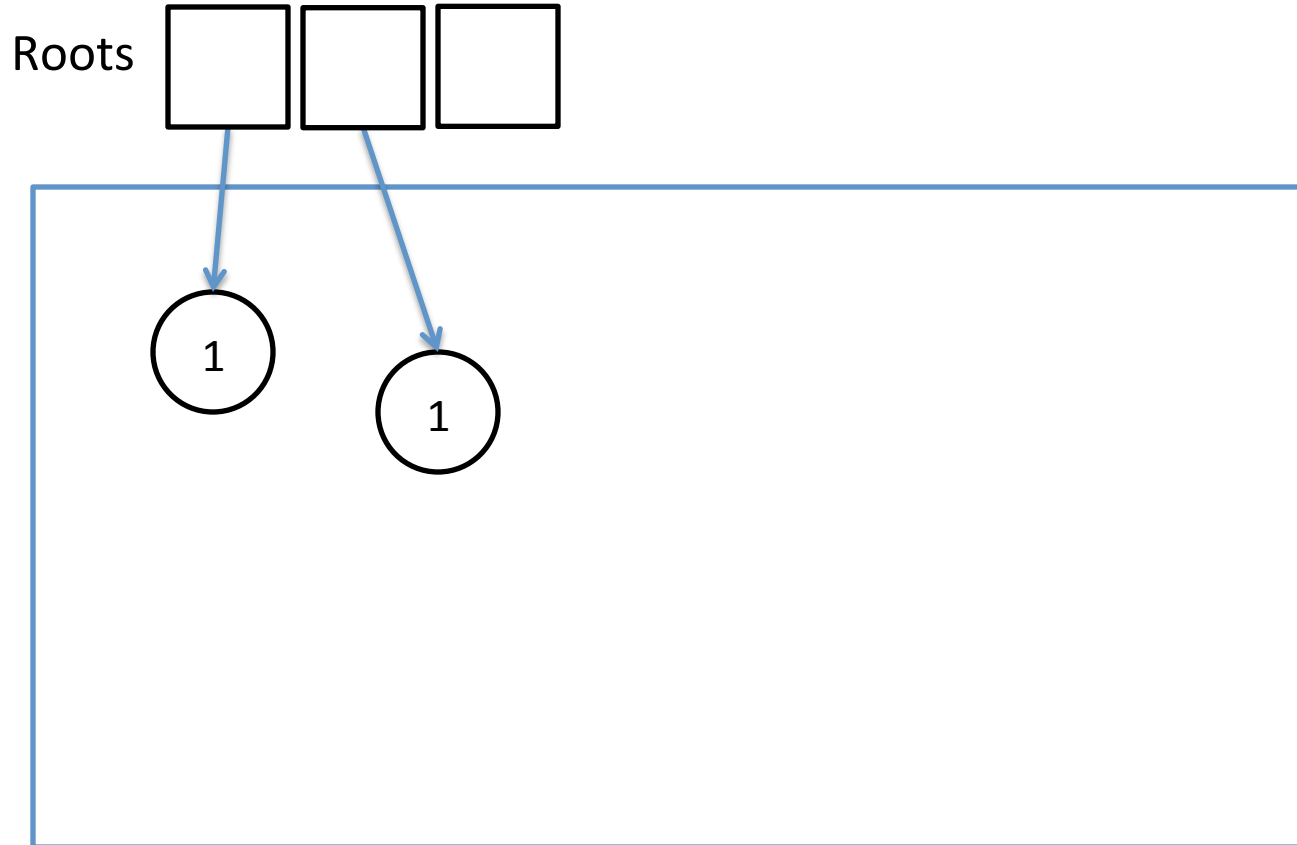


※ ここで roots とは、レジスタやメモリスタックなど
GC の起点となる集合 (参照され得ることが分かっている集合) を表す

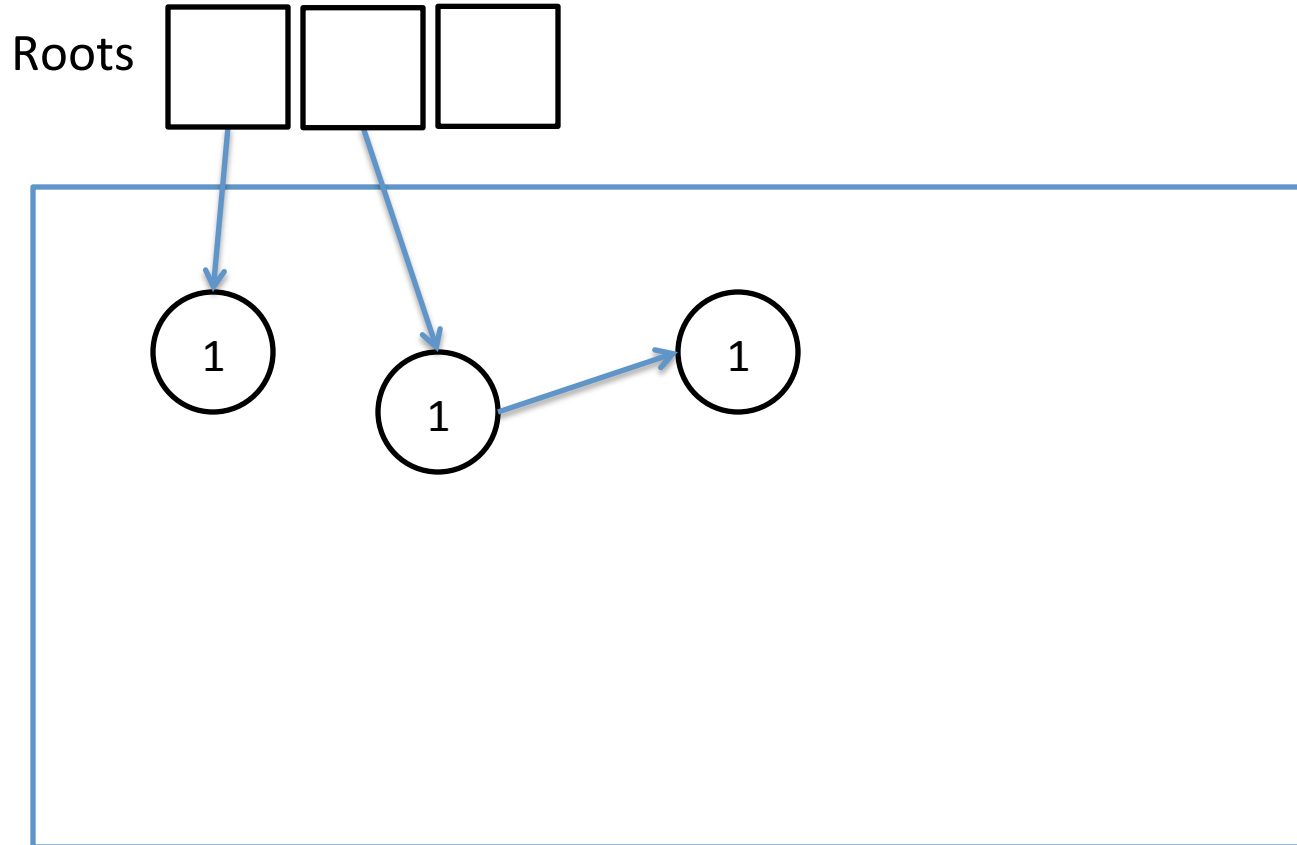
参照カウントの例



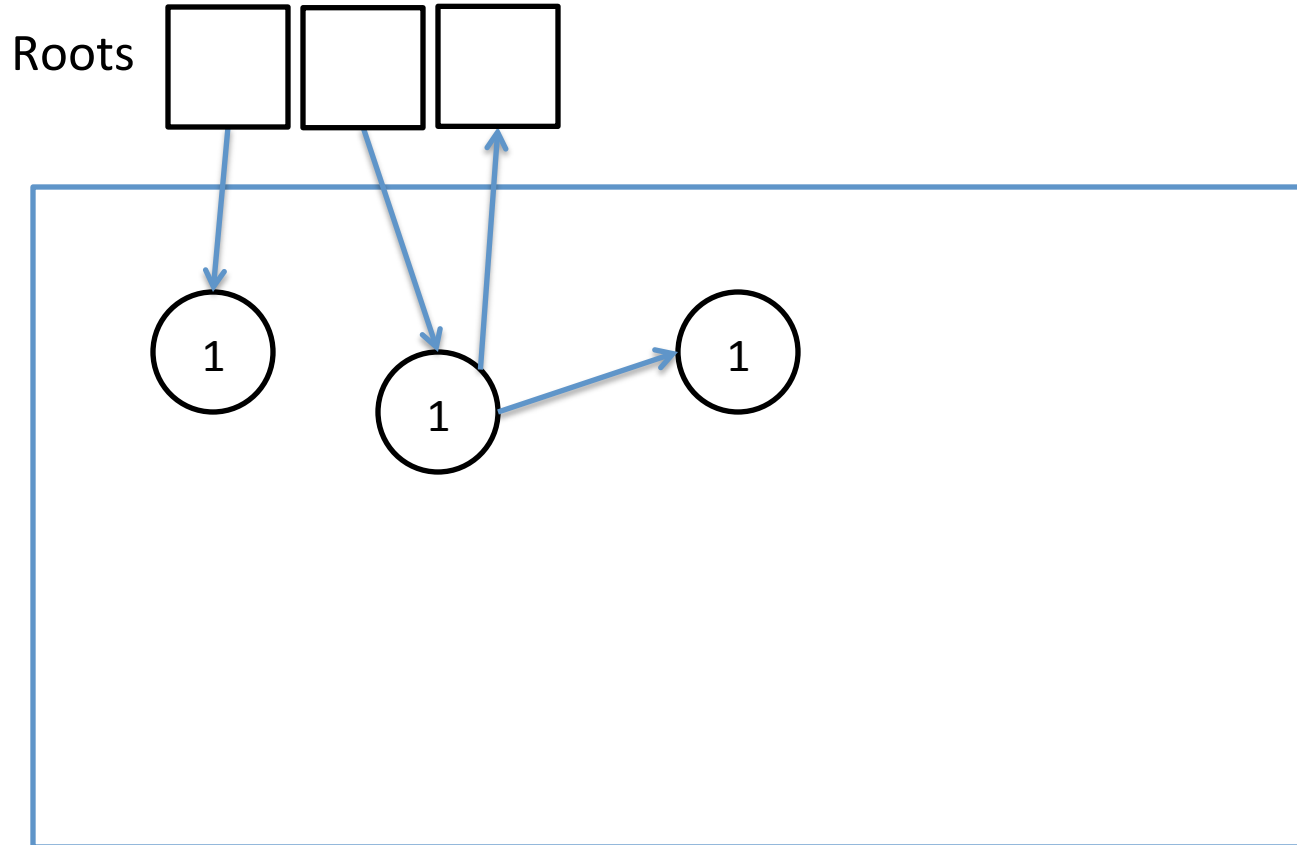
参照カウントの例



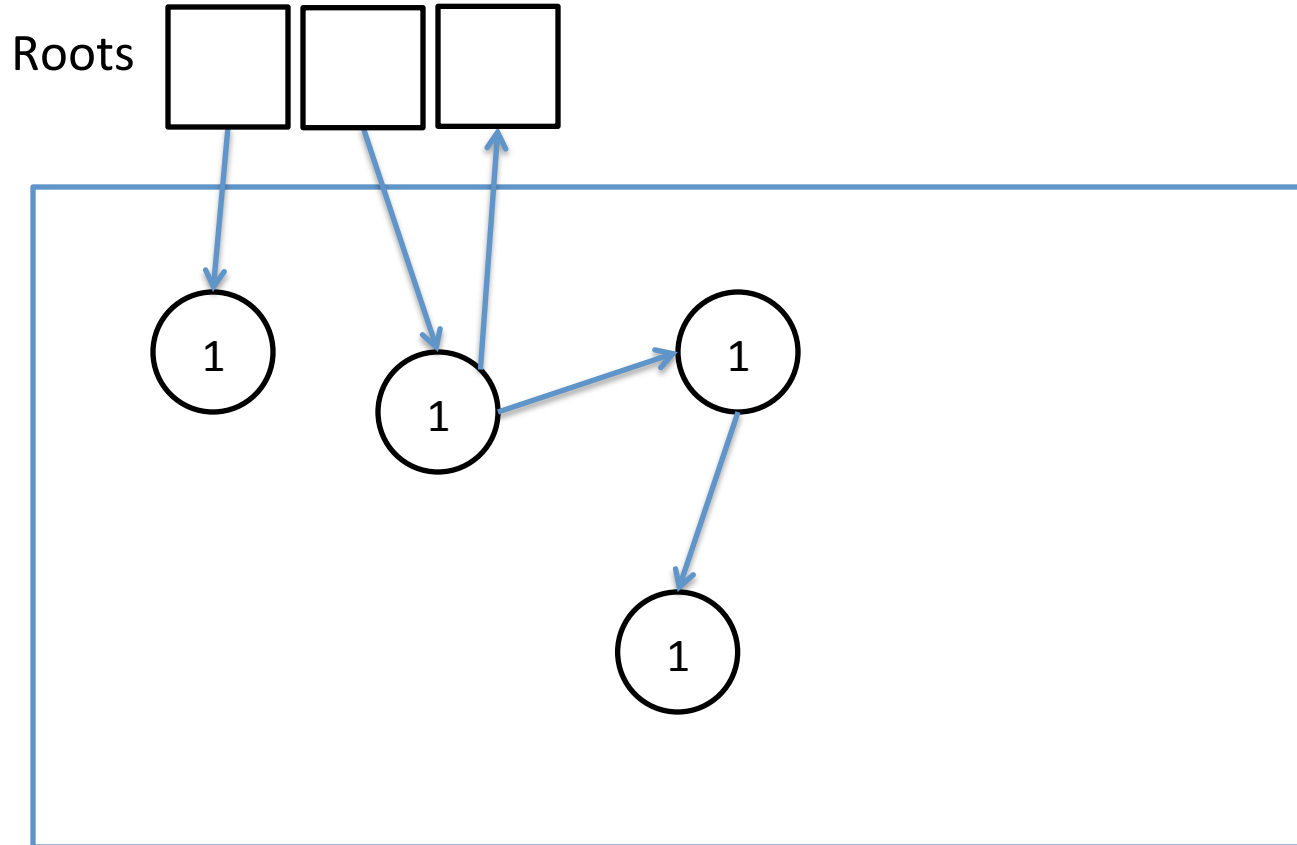
参照カウントの例



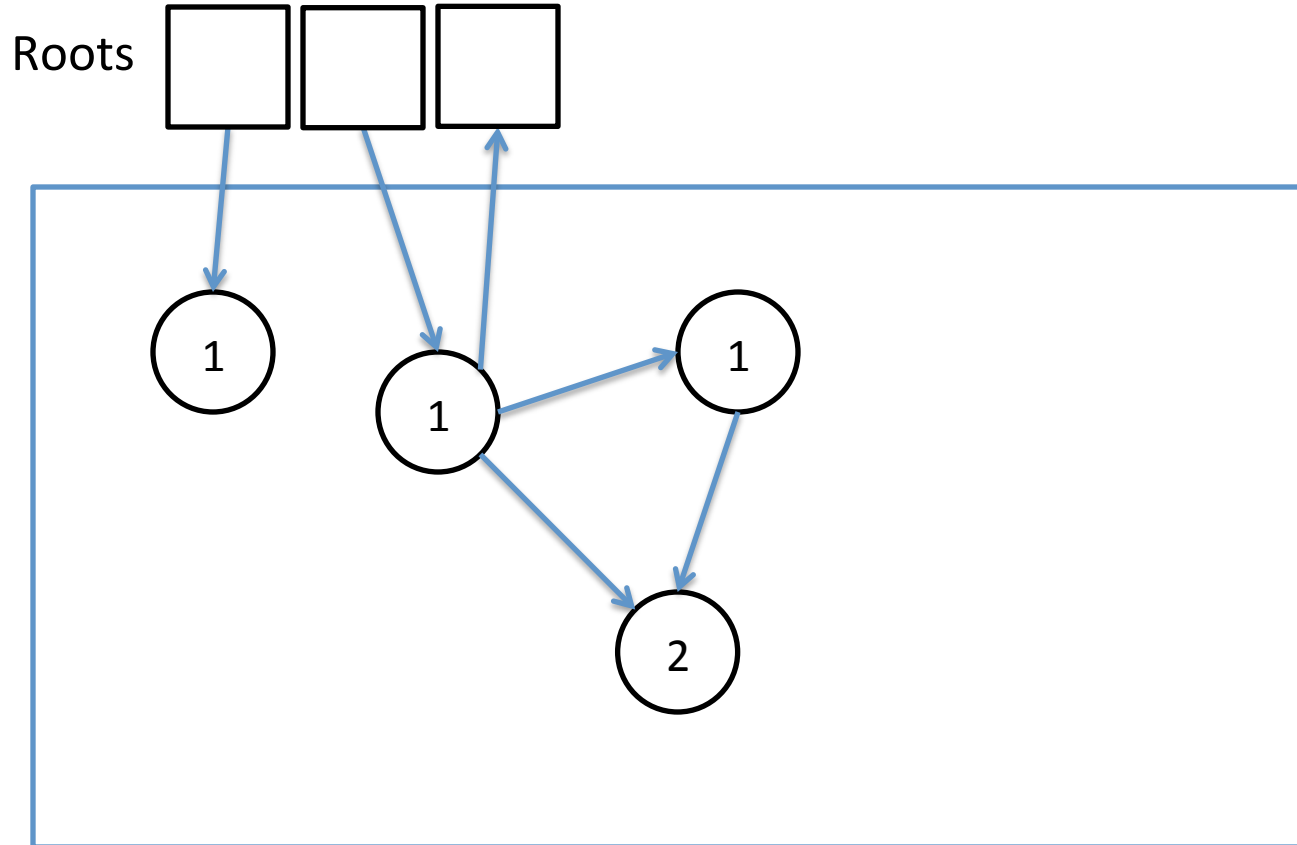
参照カウントの例



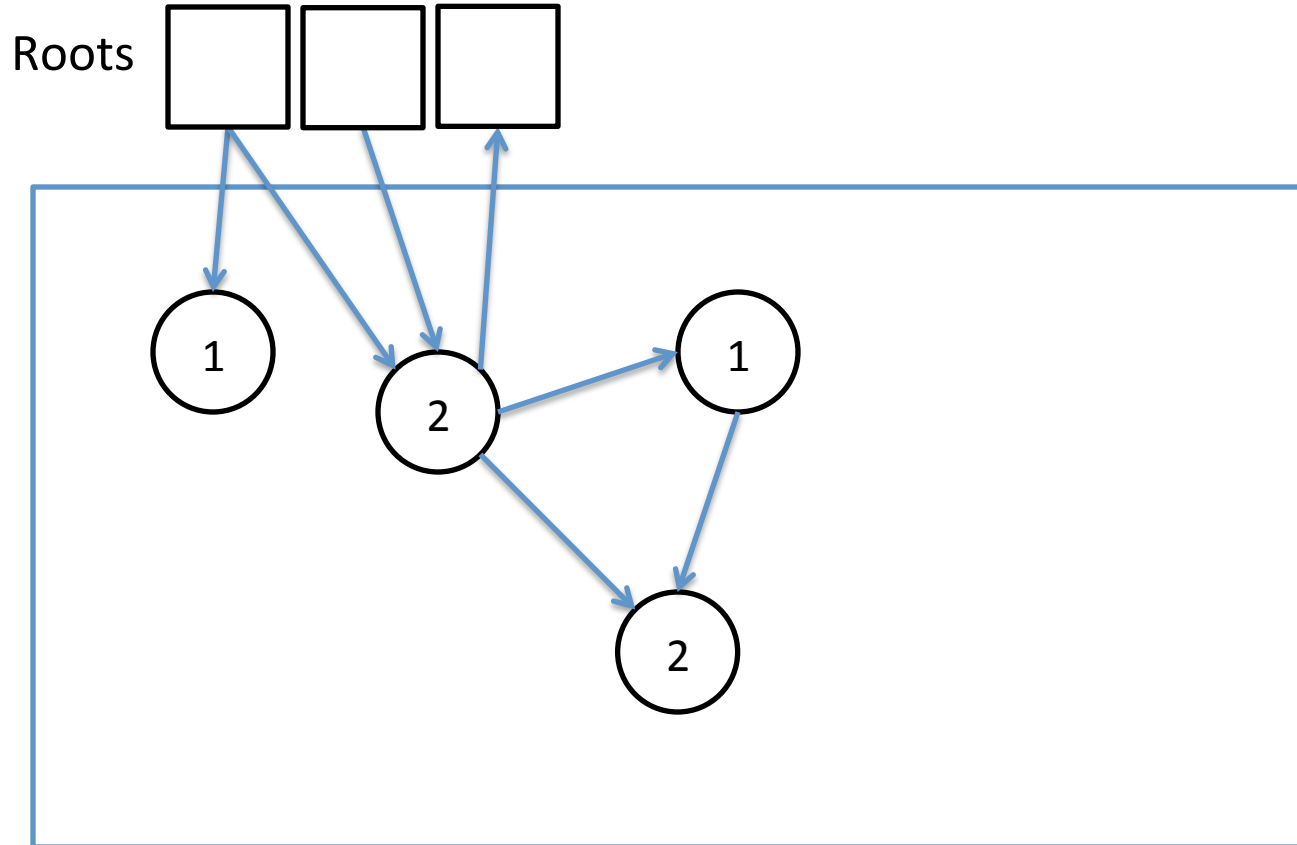
参照カウントの例



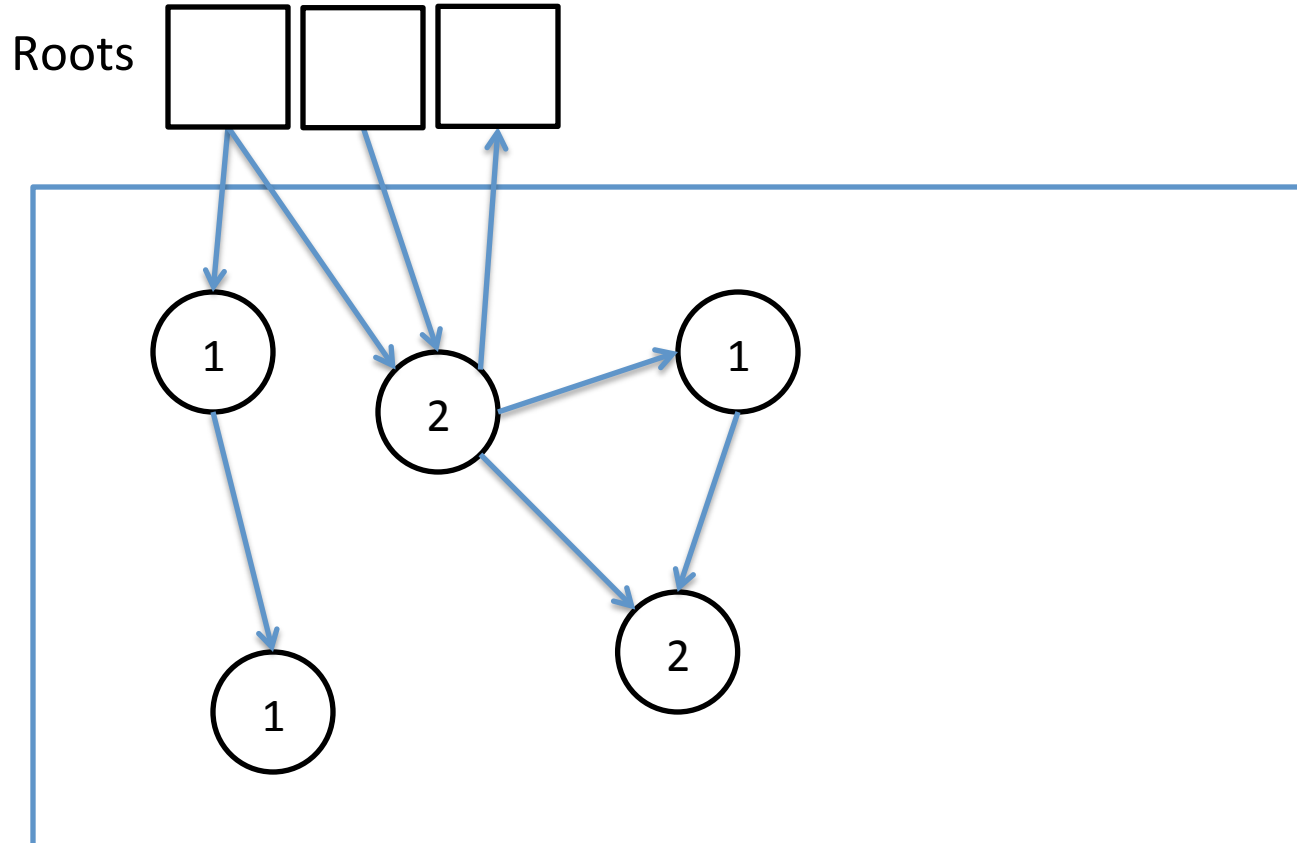
参照カウントの例



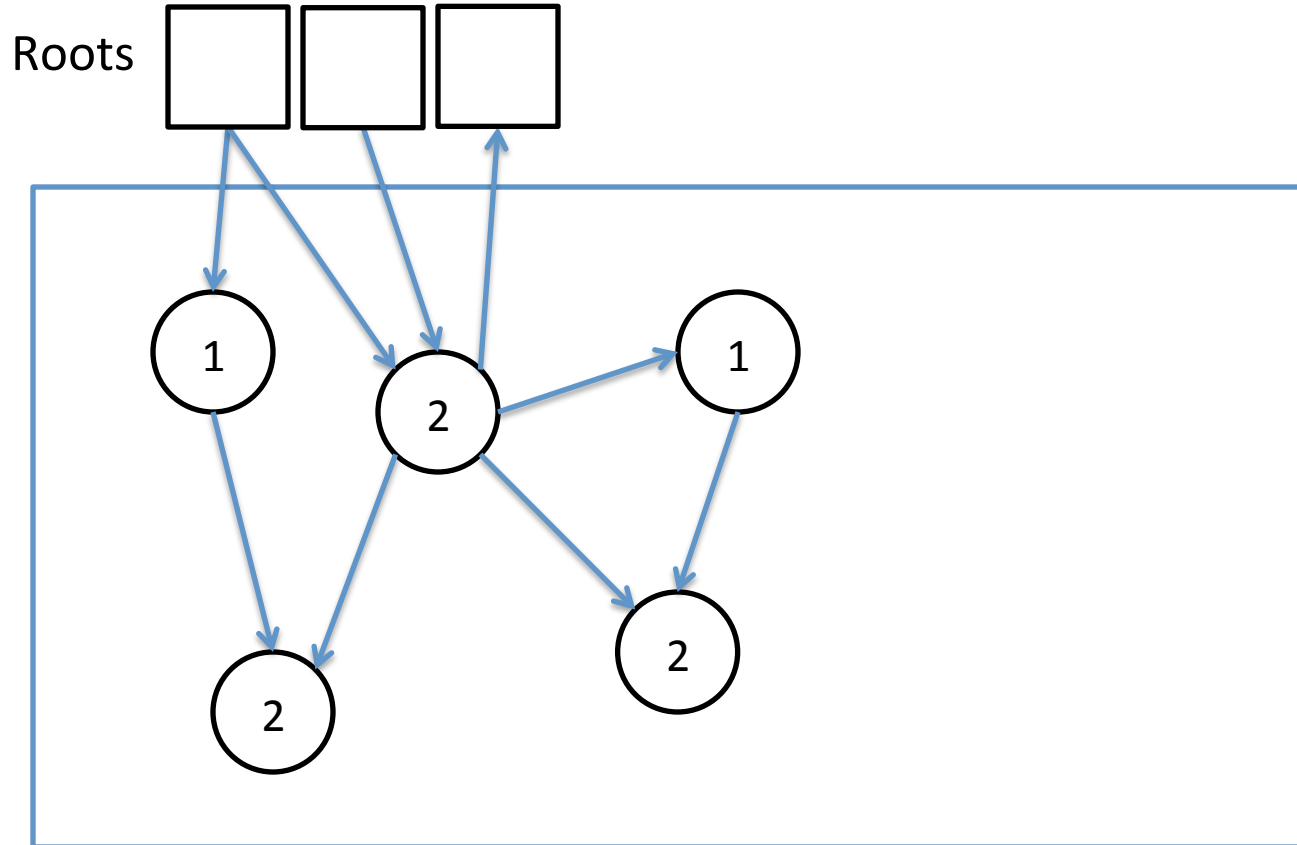
参照カウントの例



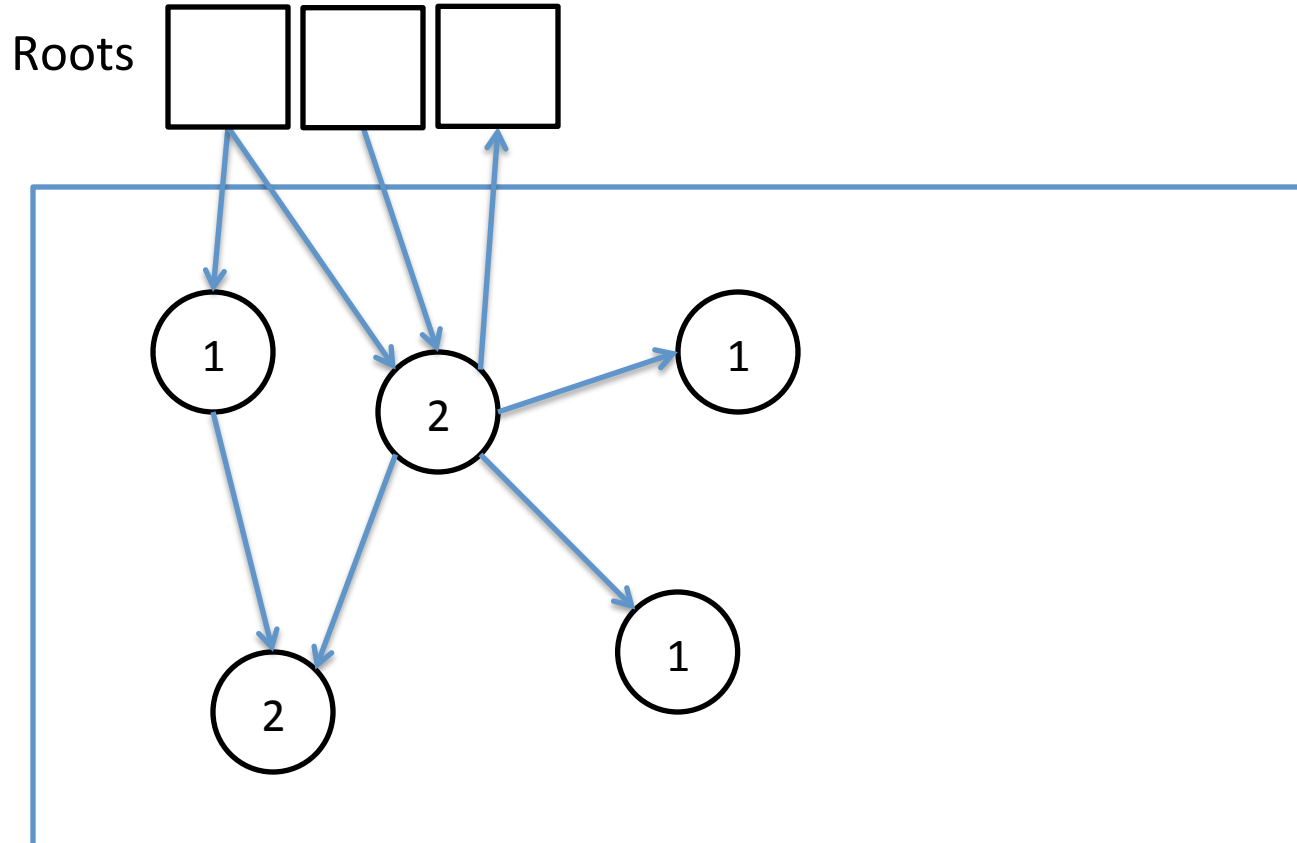
参照カウントの例



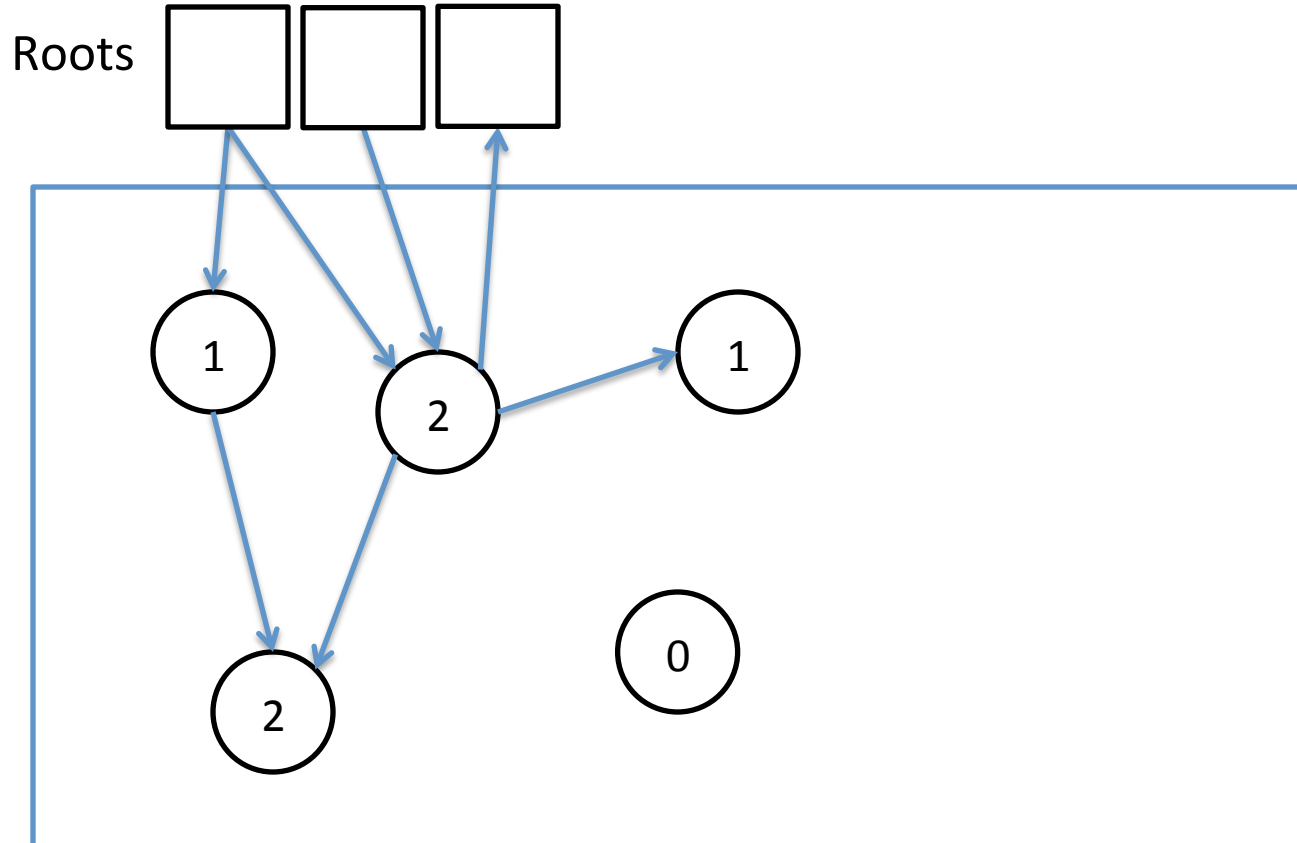
参照カウントの例



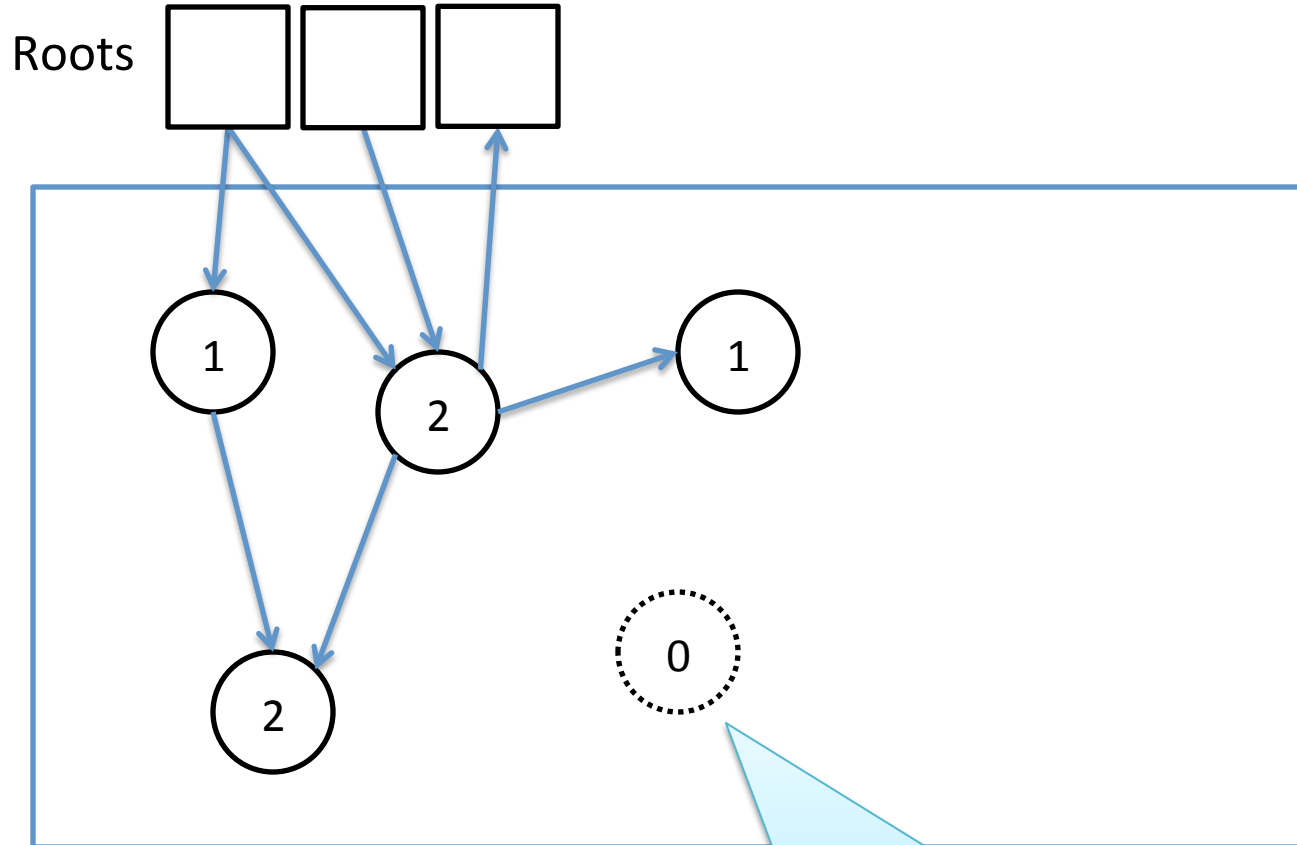
参照カウントの例



参照カウントの例

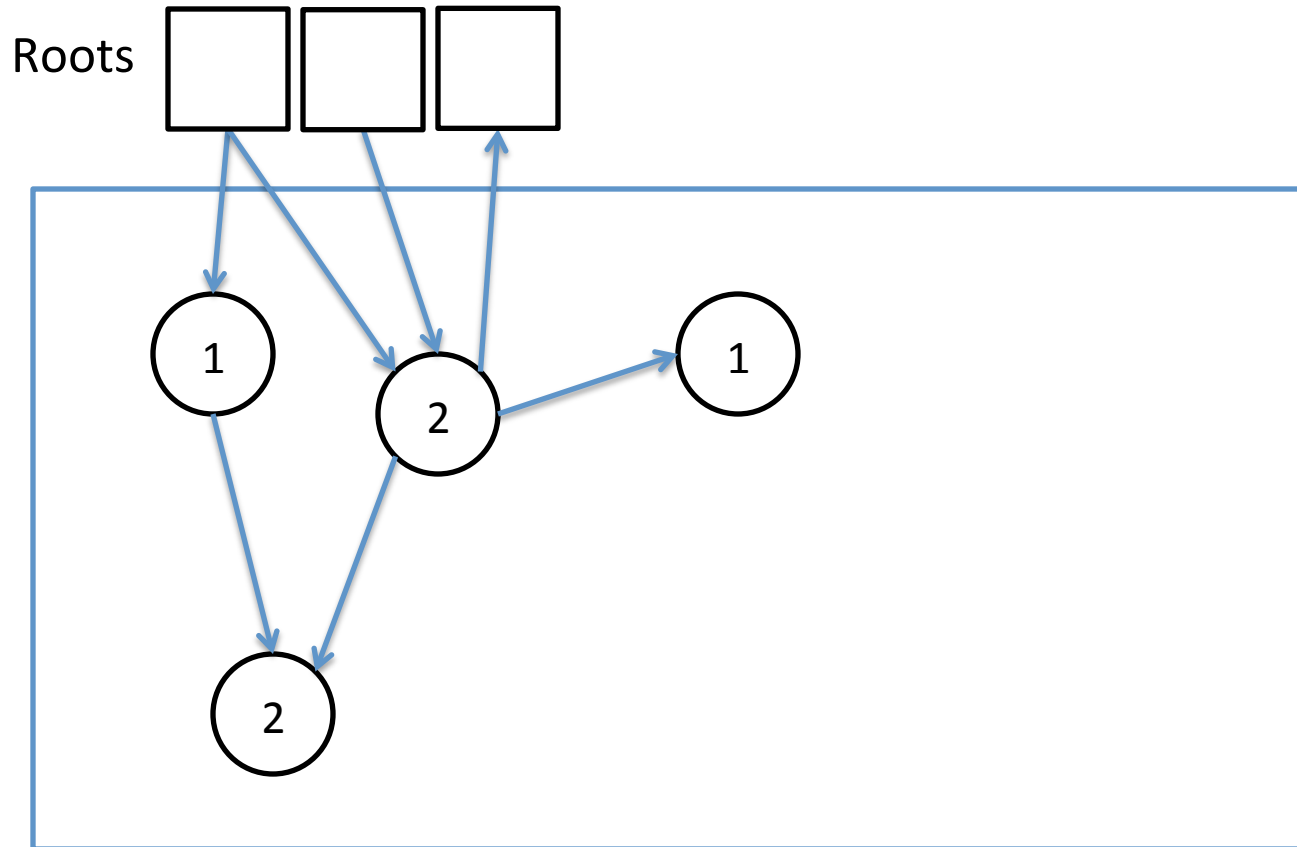


参照カウントの例

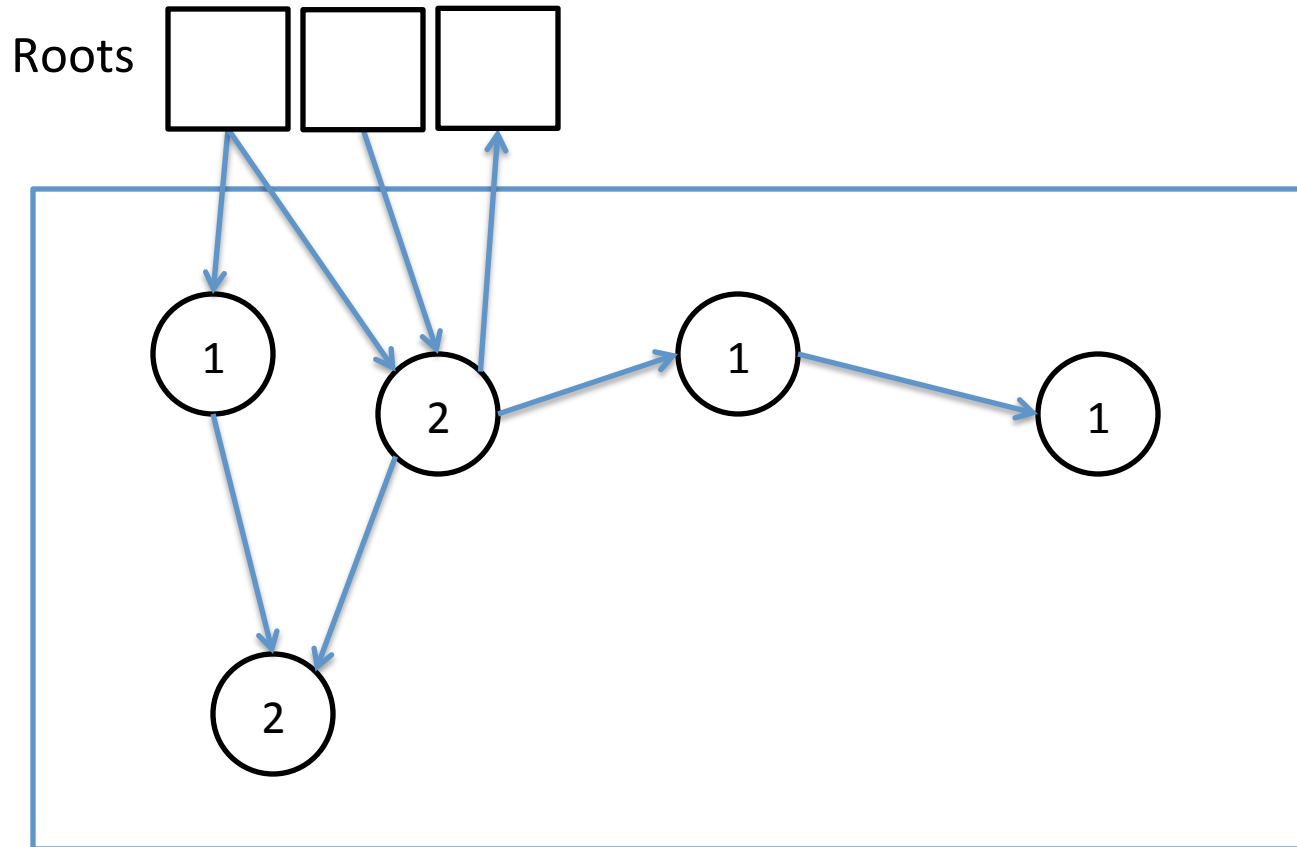


カウントが 0 になったら解放

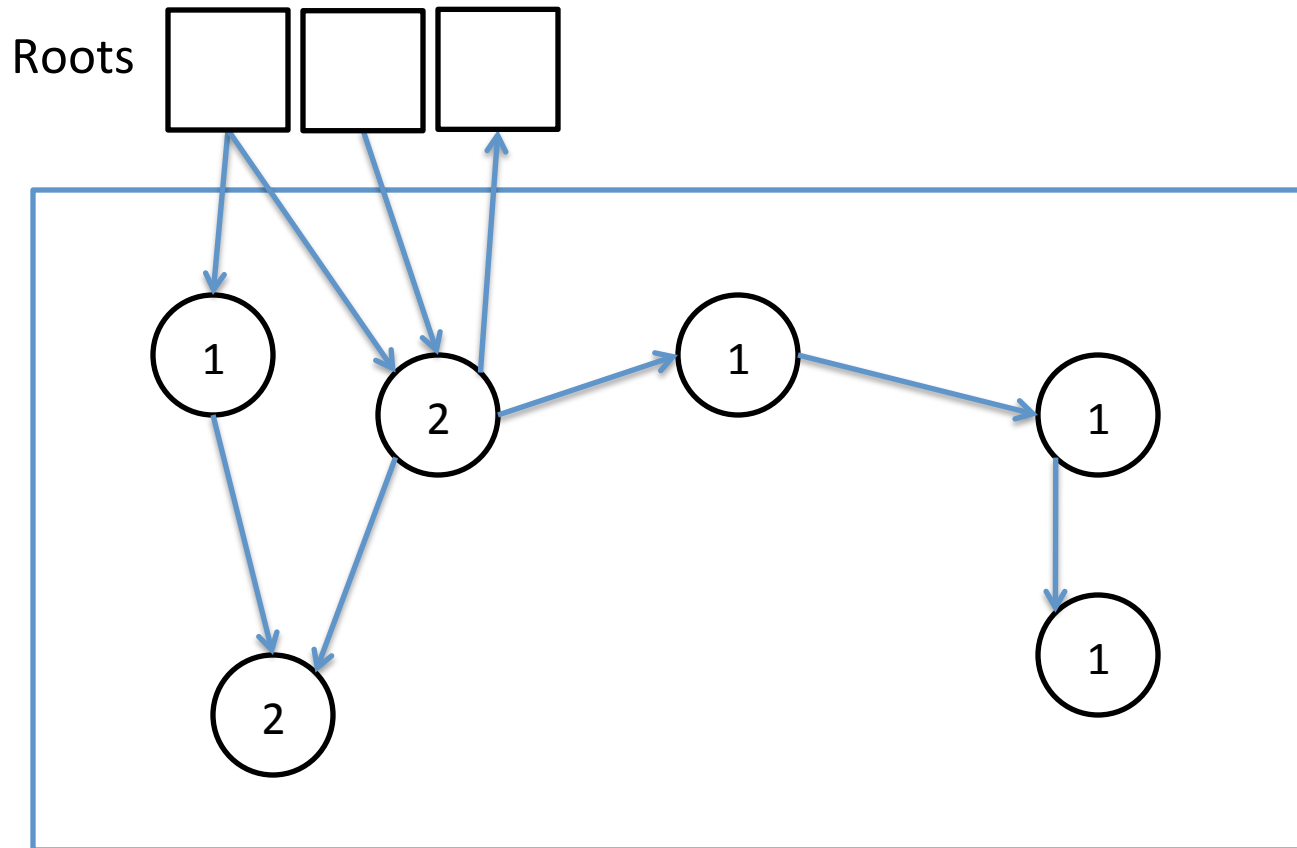
参照カウントでは上手くいかない例



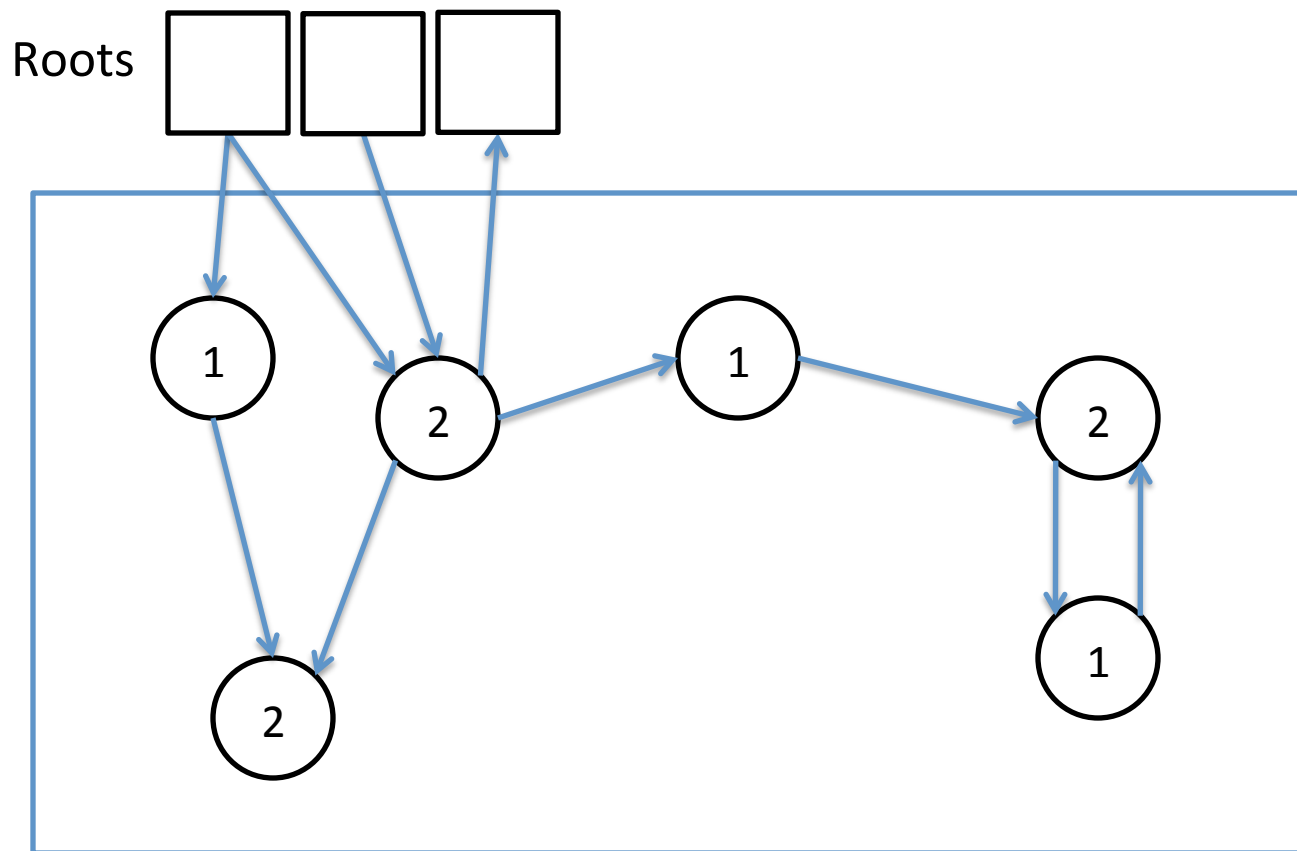
参照カウントでは上手くいかない例



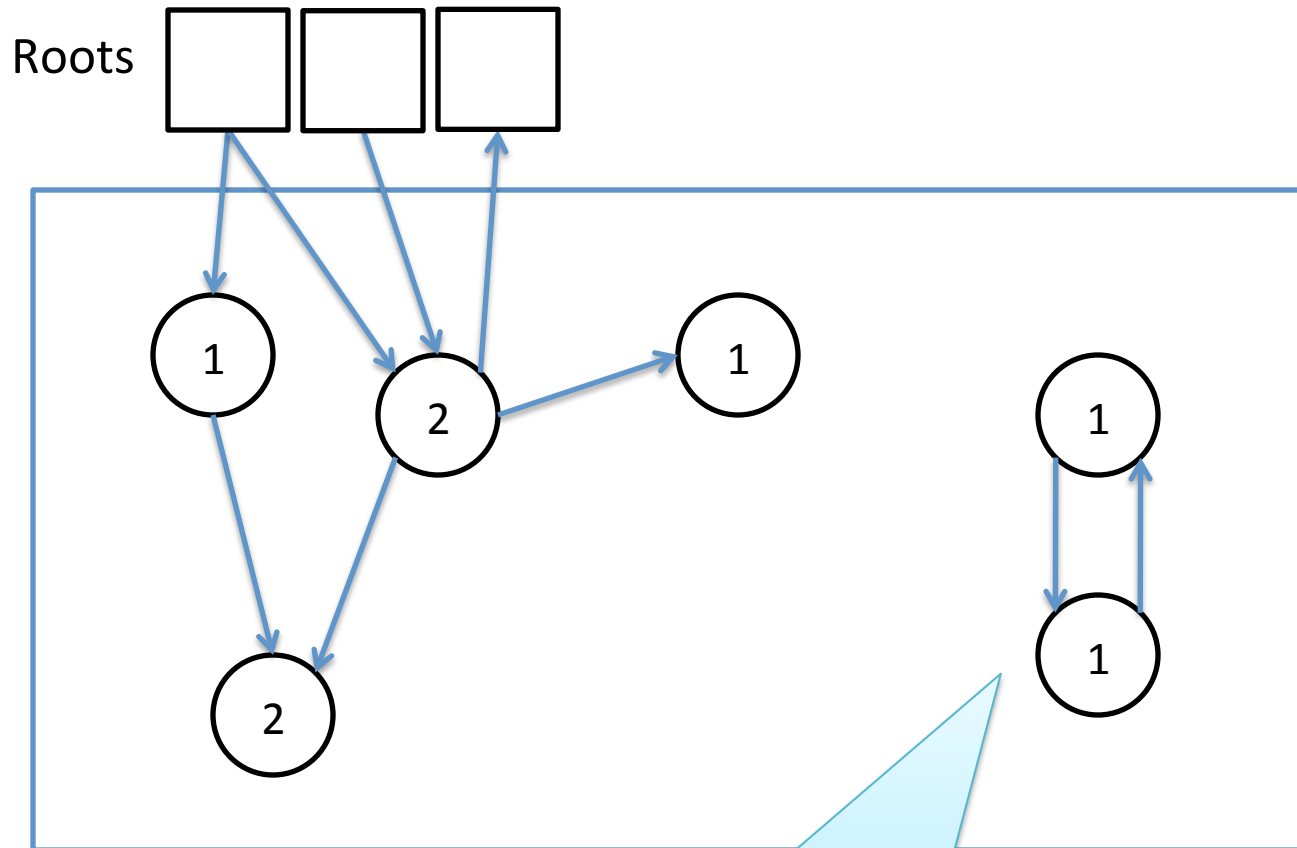
参照カウントでは上手くいかない例



参照カウントでは上手くいかない例



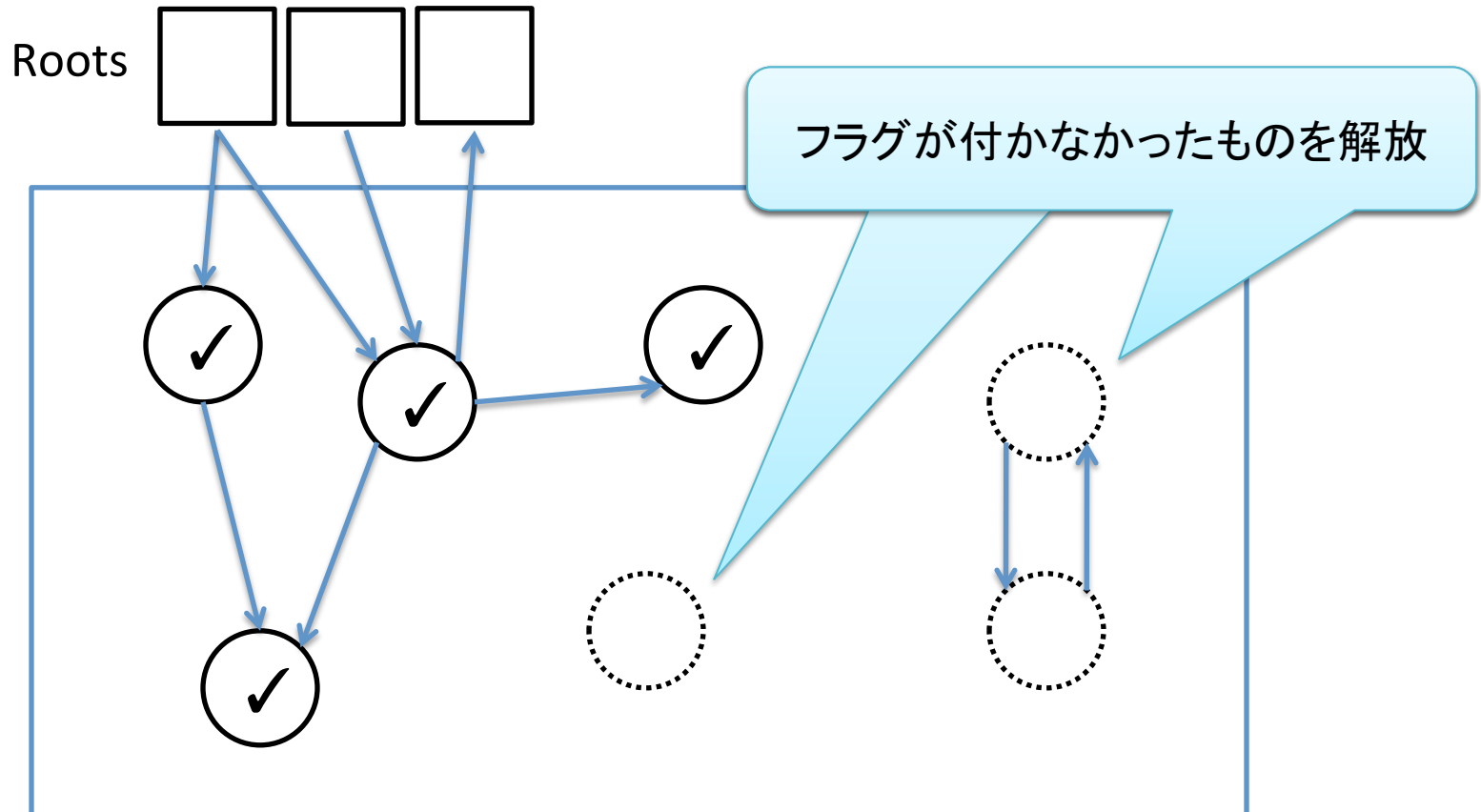
参照カウントでは上手くいかない例



循環参照はどうする?

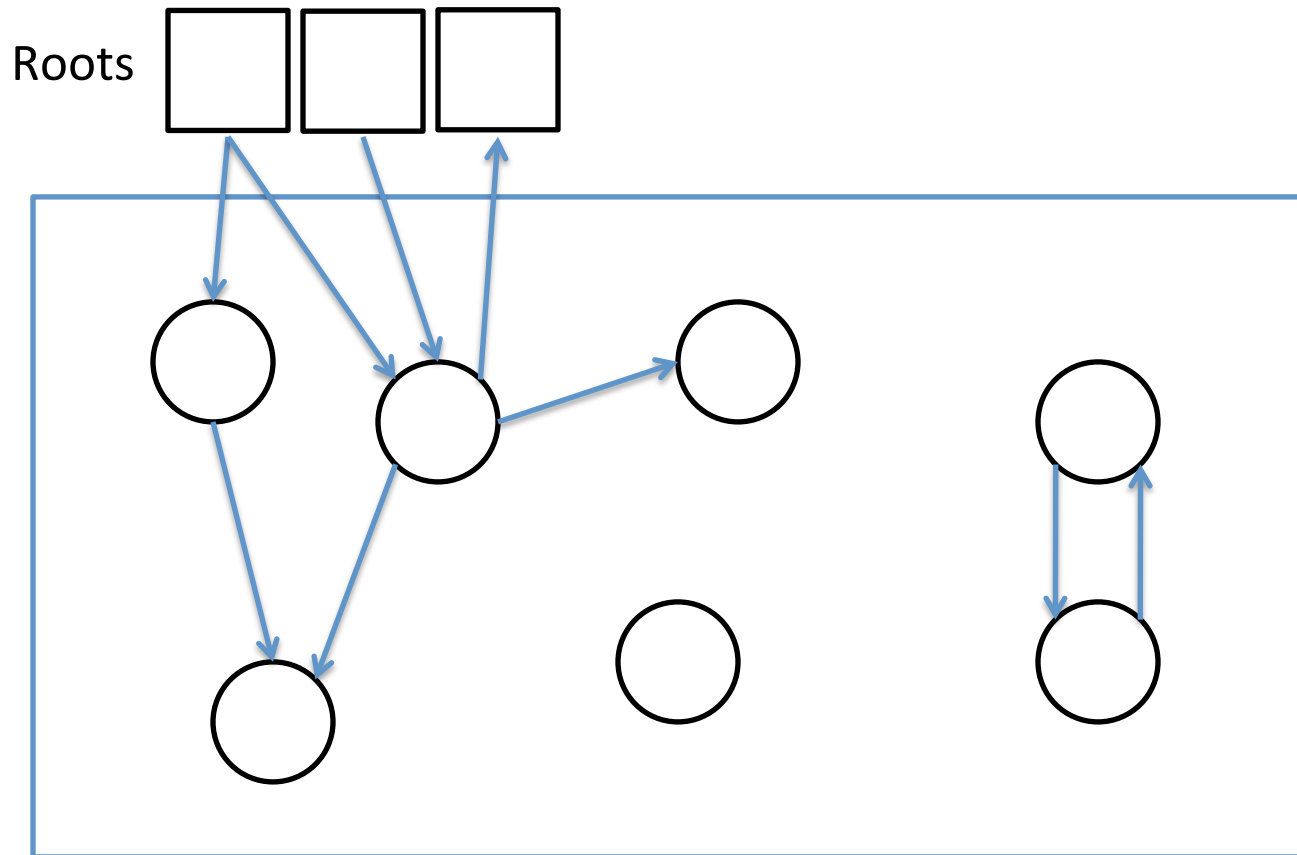
マーク & スweep

- マークとスweepの 2 段階からなる
 - マーク: 参照可能なオブジェクトにフラグを立てる
 - スweep: フラグの立っていないオブジェクトを解放



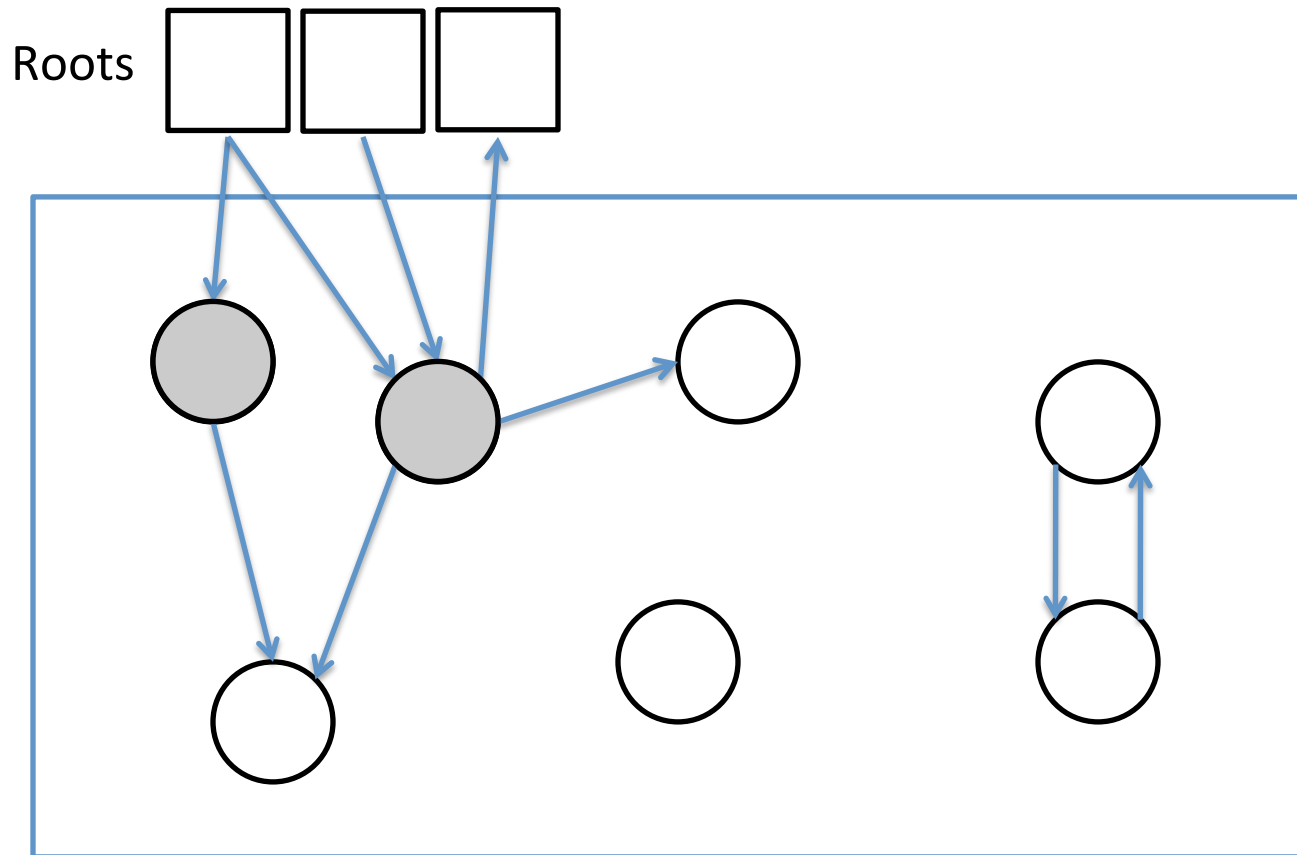
3色マーク & スニープの例

- まず全て「白」にする



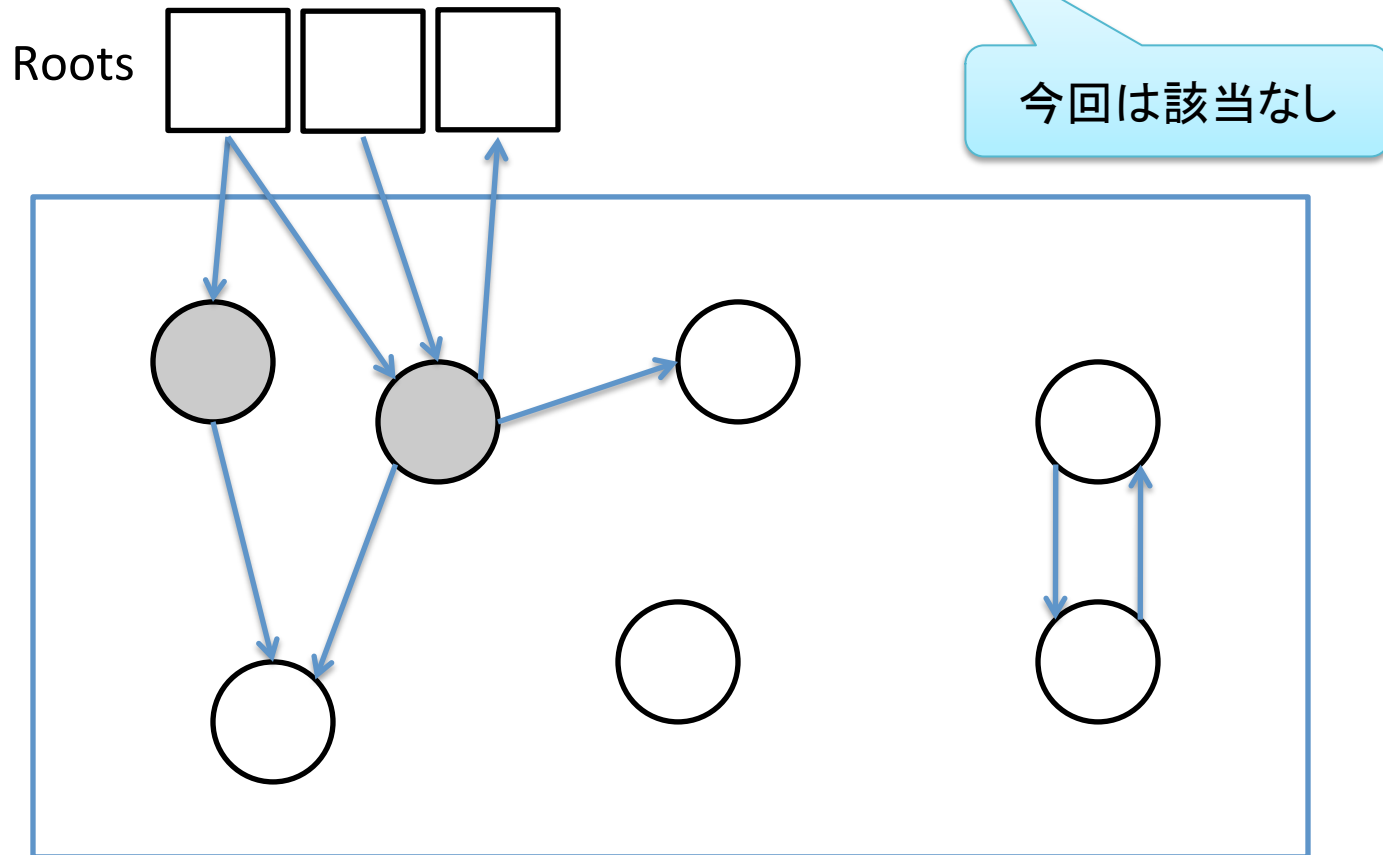
3色マーク & スィープの例

- 到達可能なオブジェクトを「灰」にする



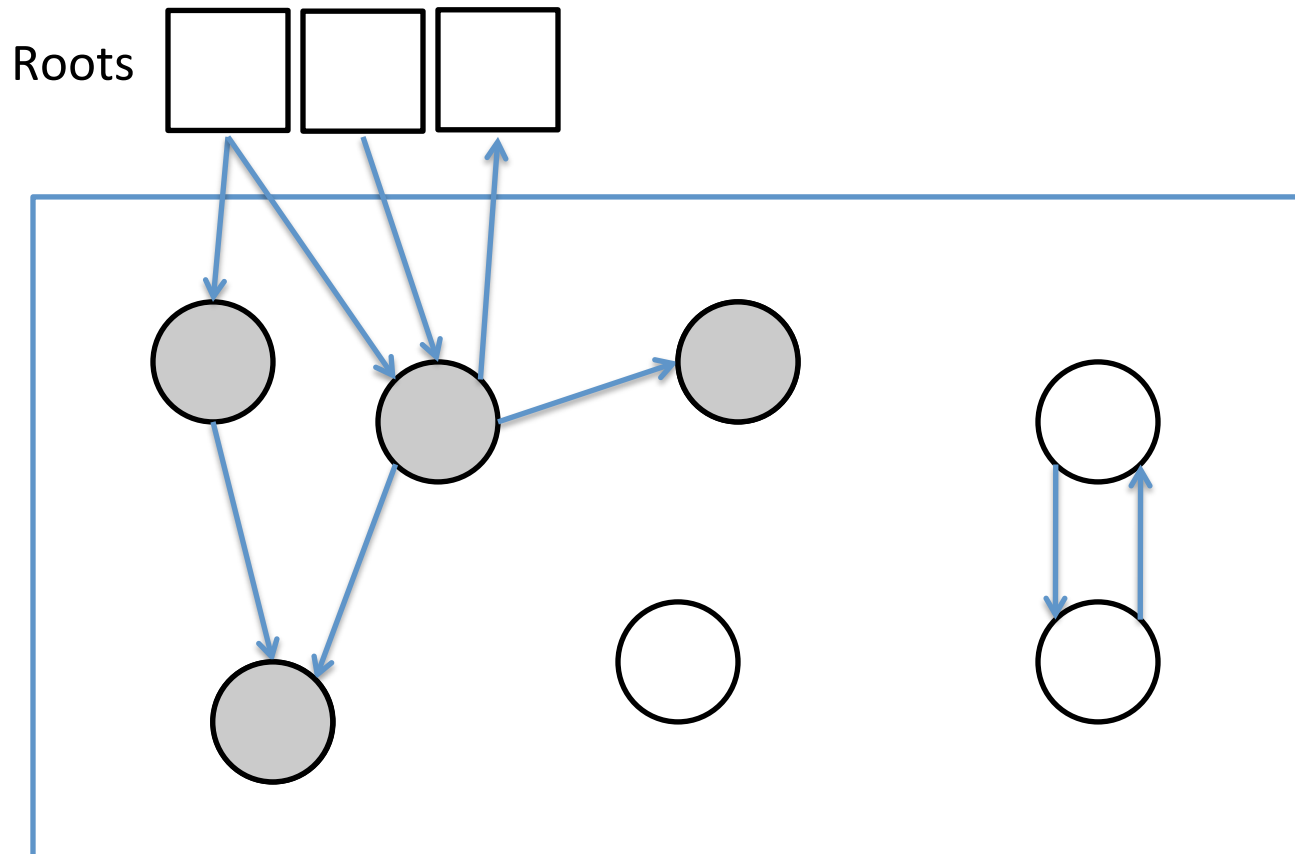
3色マーク & スィープの例

- 未探索の参照のない「灰」を「黒」にする



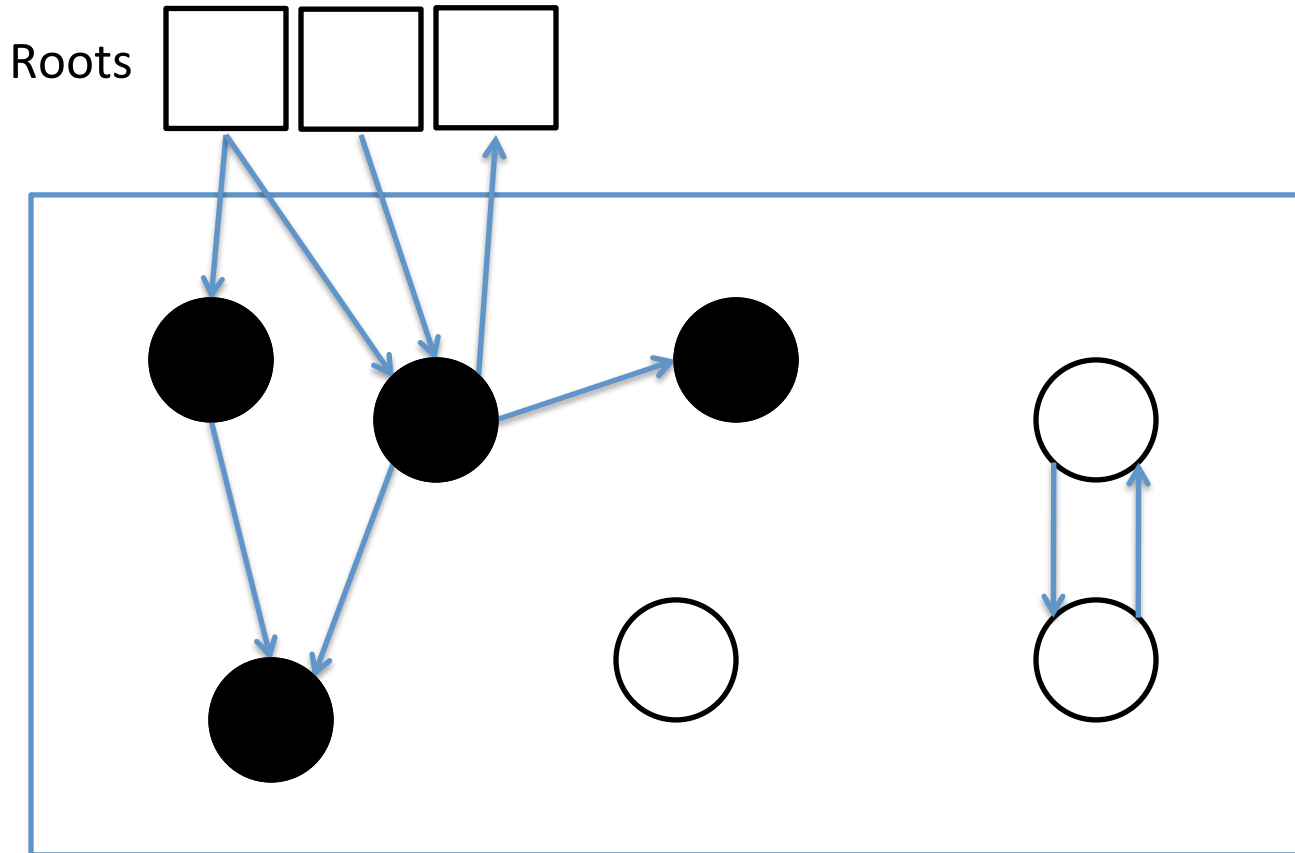
3色マーク & スィープの例

- 到達可能なオブジェクトを「灰」にする



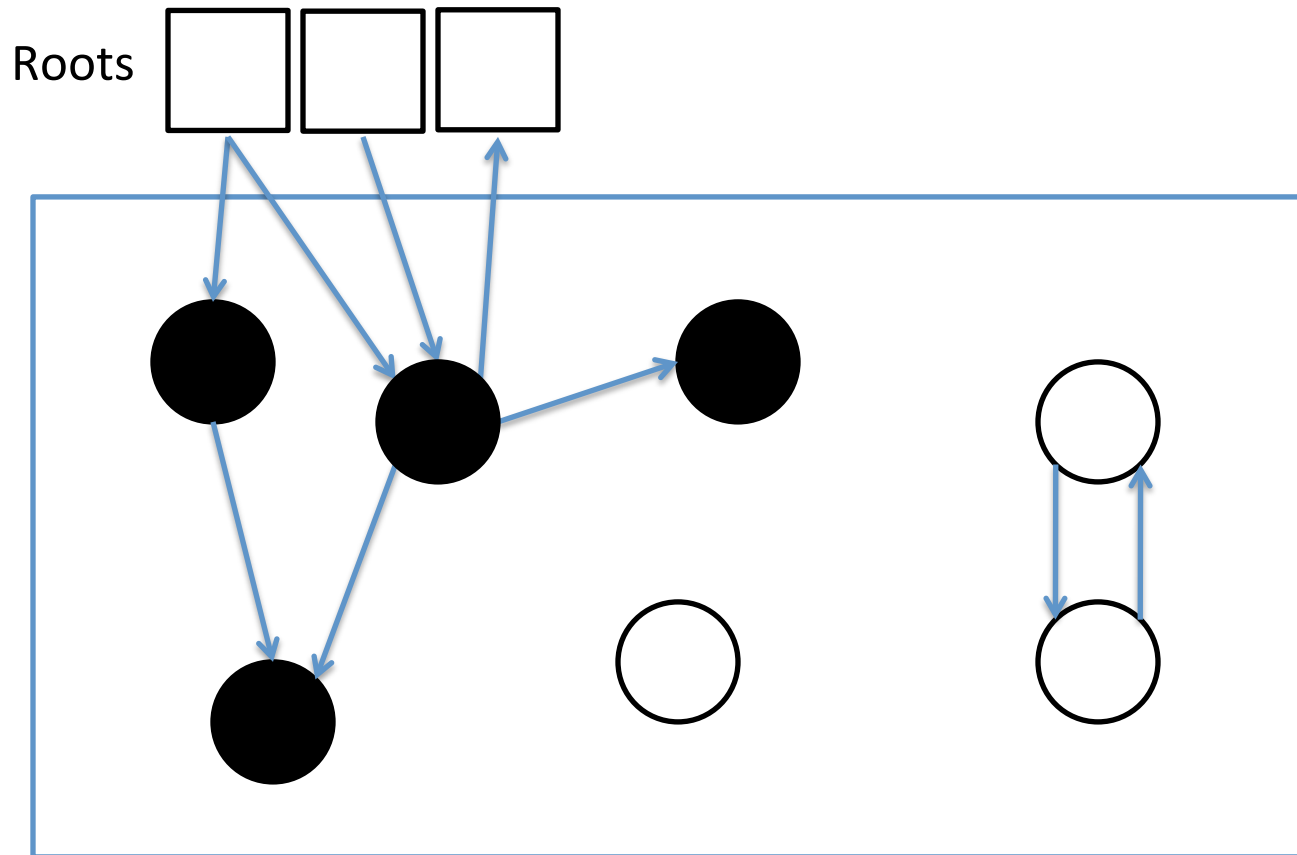
3色マーク & スィープの例

- 未探索の参照のない「灰」を「黒」にする



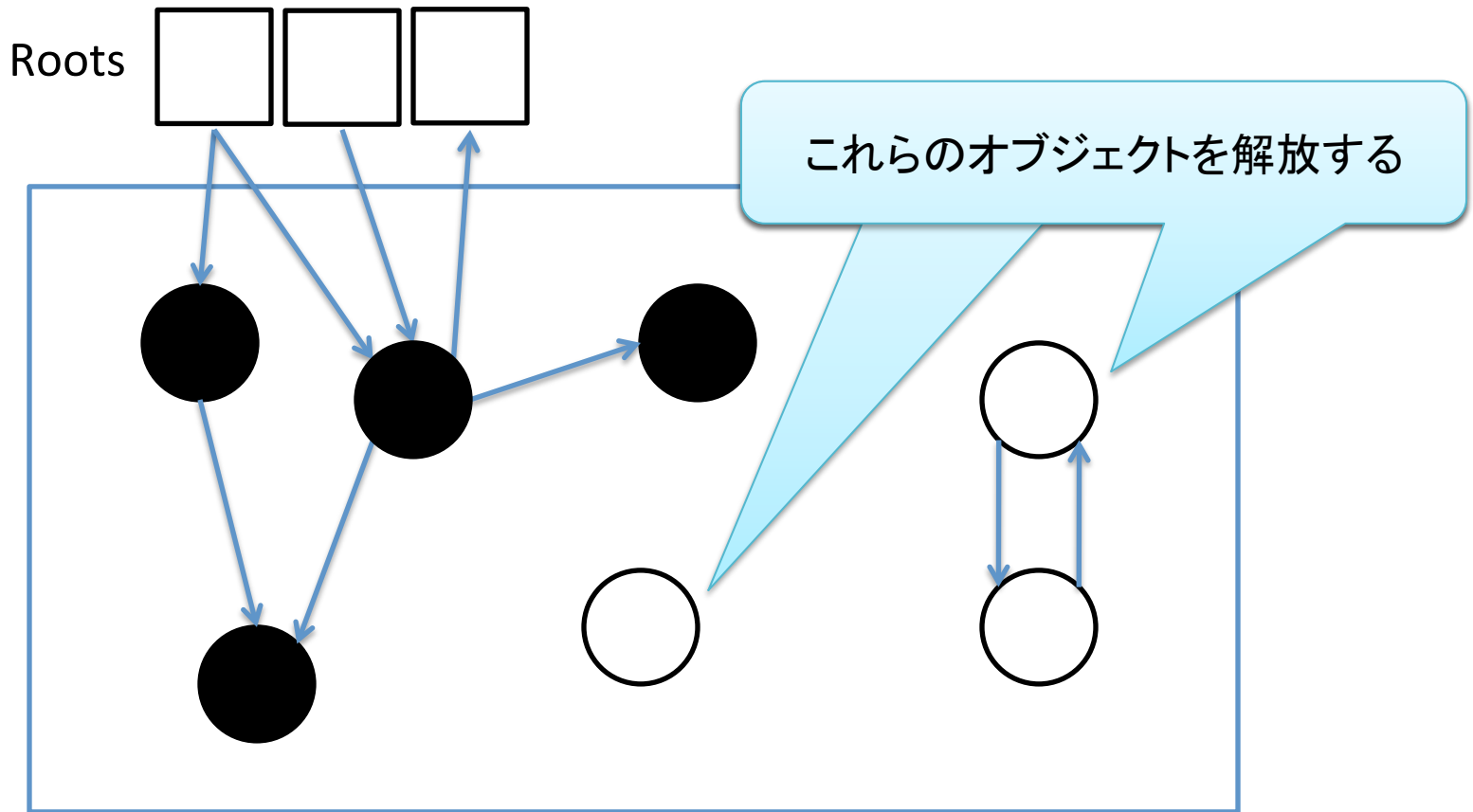
3色マーク & スィープの例

- 「灰」が無くなったのでマーク終了



3色マーク & スィープの例

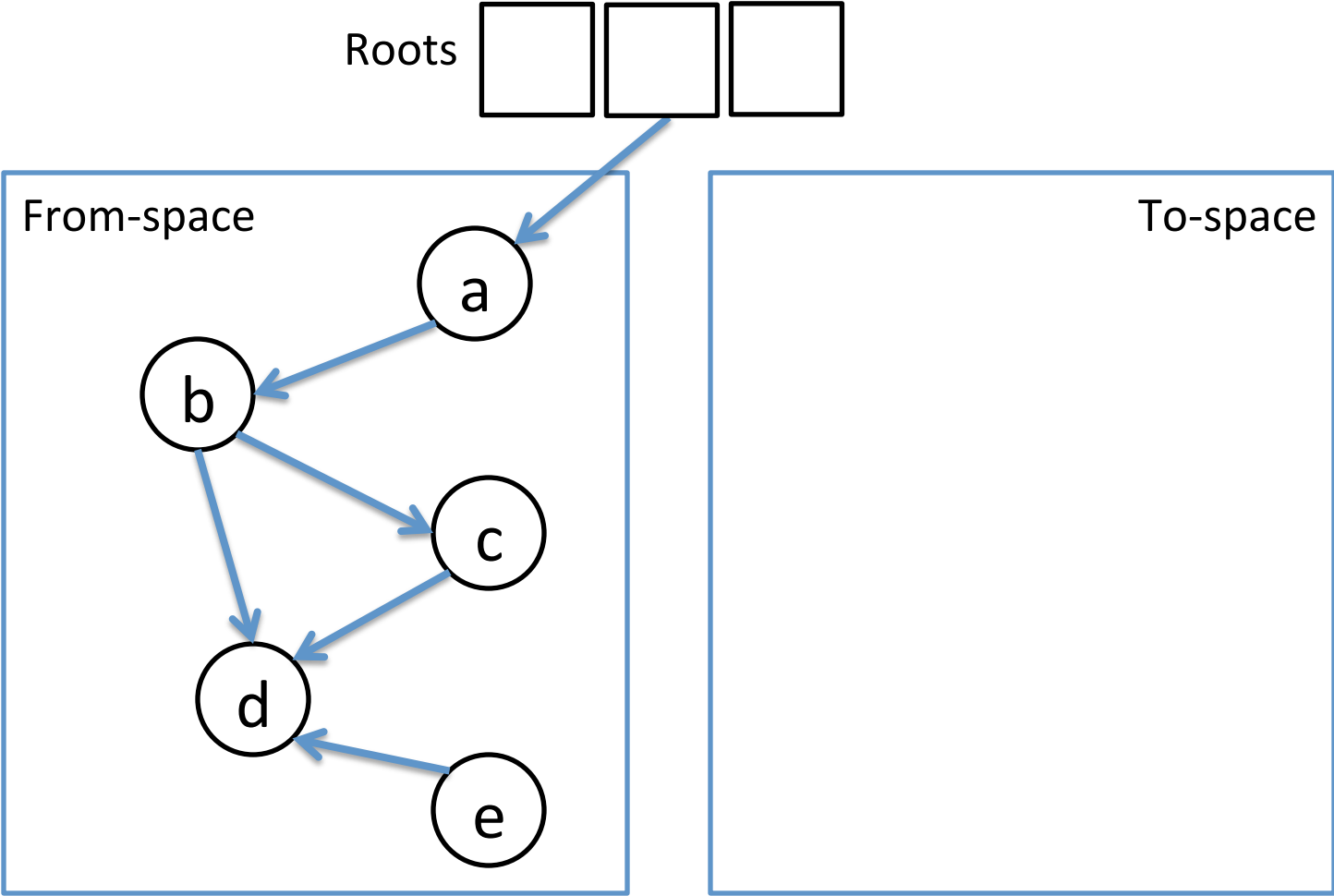
- 全ての「白」を解放する (スィープ)



Copying GC

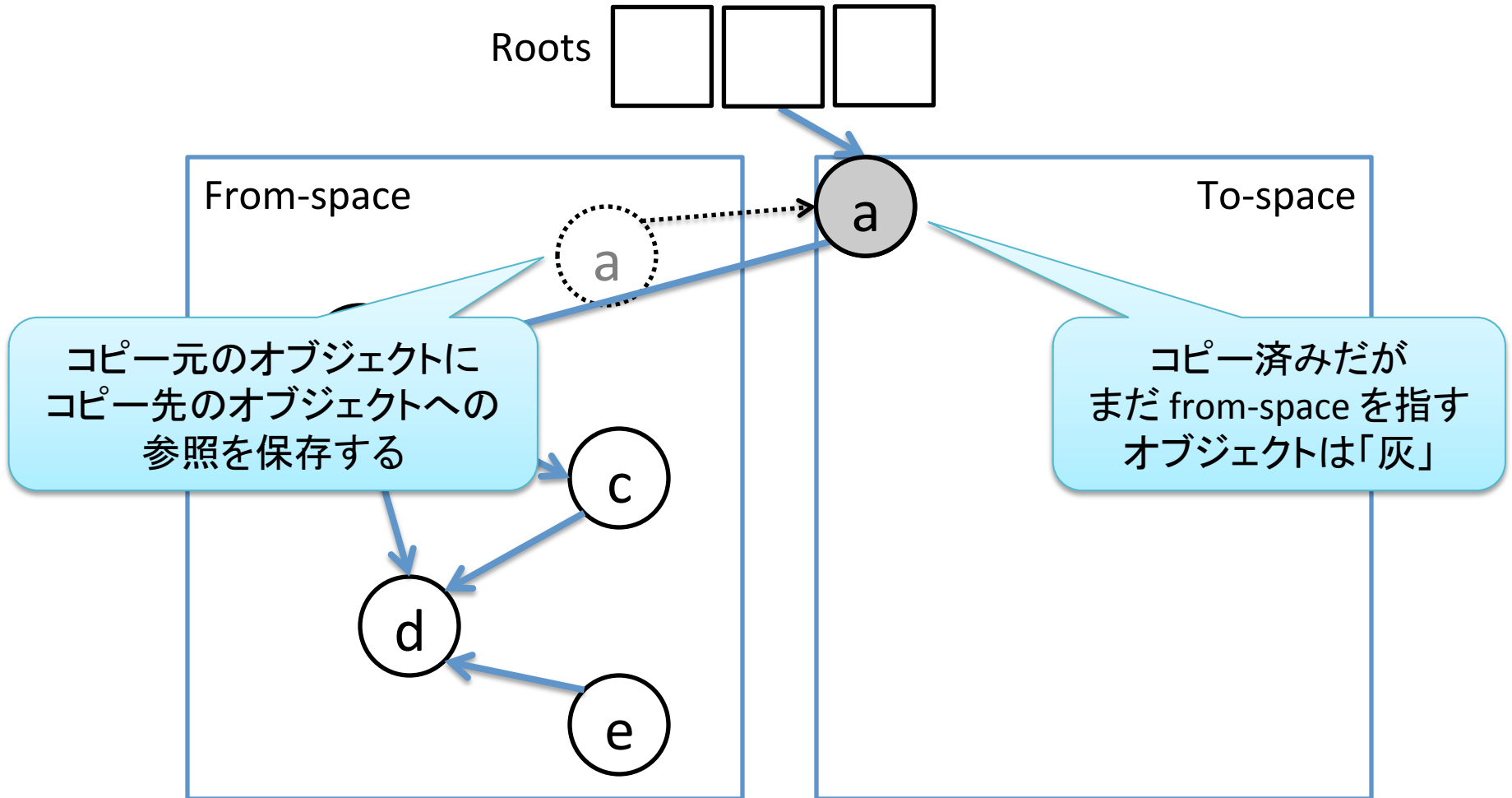
- ヒープを二分割し、片方だけを使う
- 片方が埋まったら、参照可能なオブジェクトをもう片方に移す
 - 移されなかったオブジェクトはそのまま解放
- 移すついでにヒープのデフラグができる

Copying GC の例 (GC 開始前)



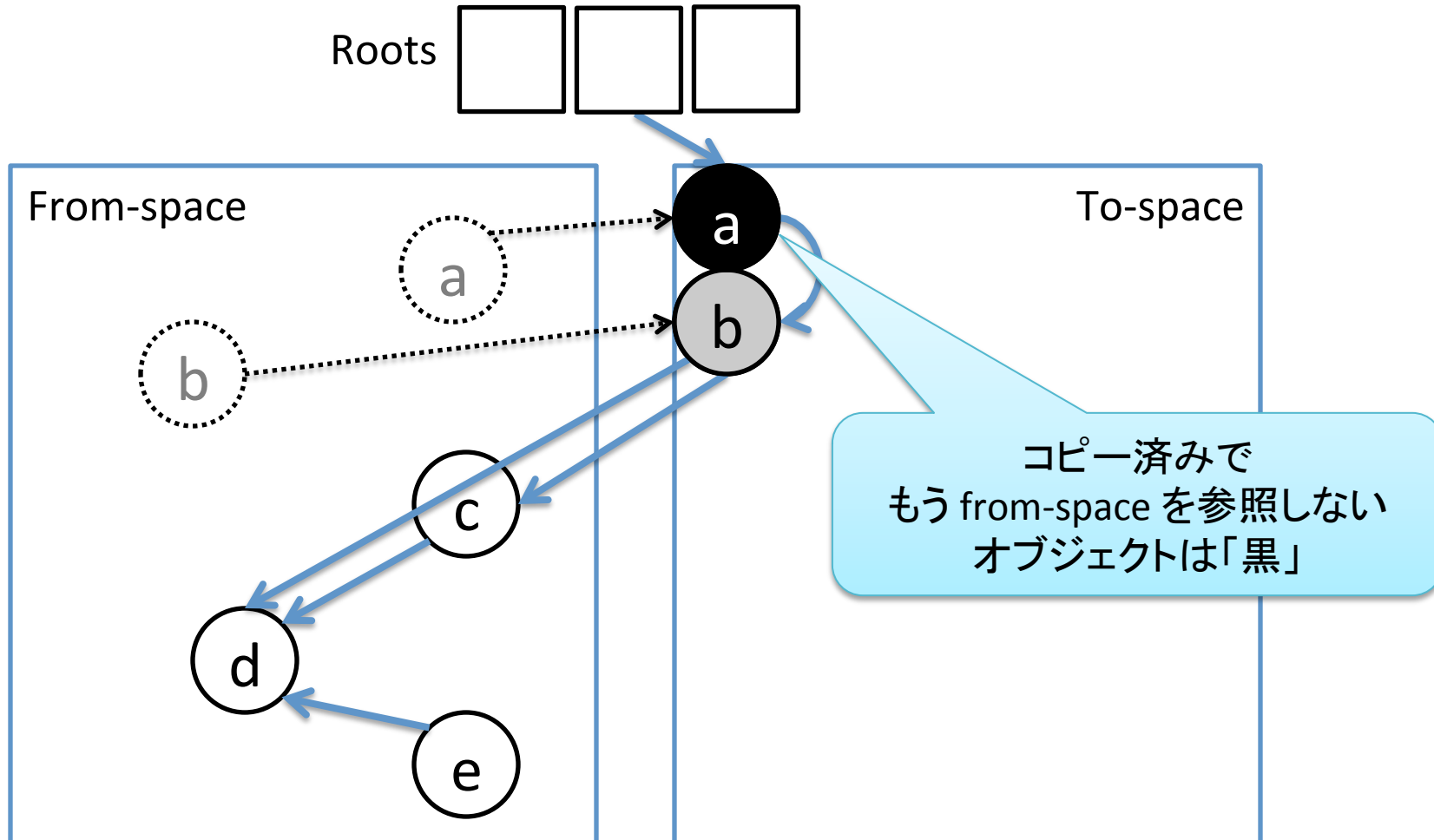
Copying GC の例

到達可能なオブジェクトを to-space へコピー



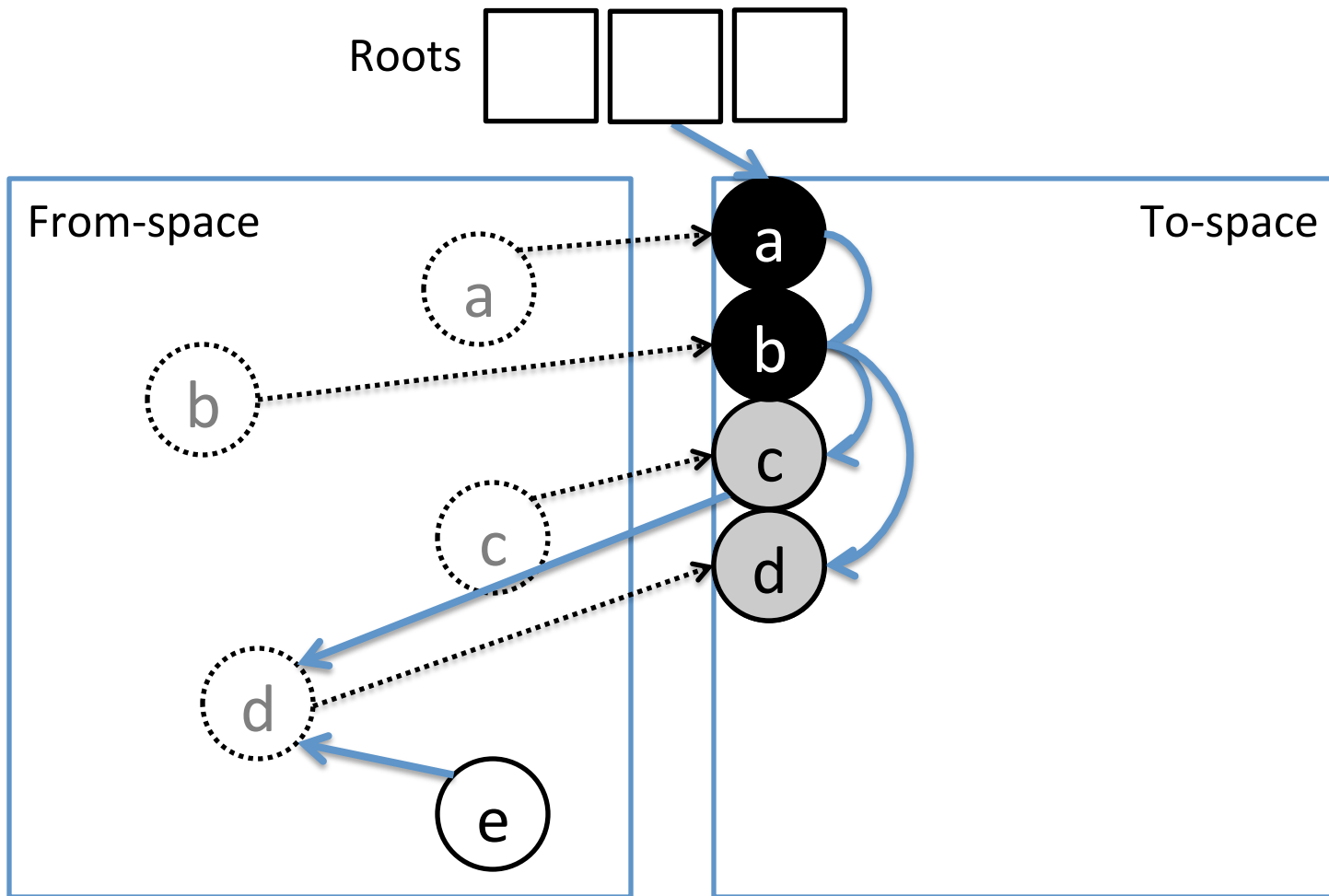
Copying GC の例

「灰」のオブジェクトから参照されるオブジェクトをコピー



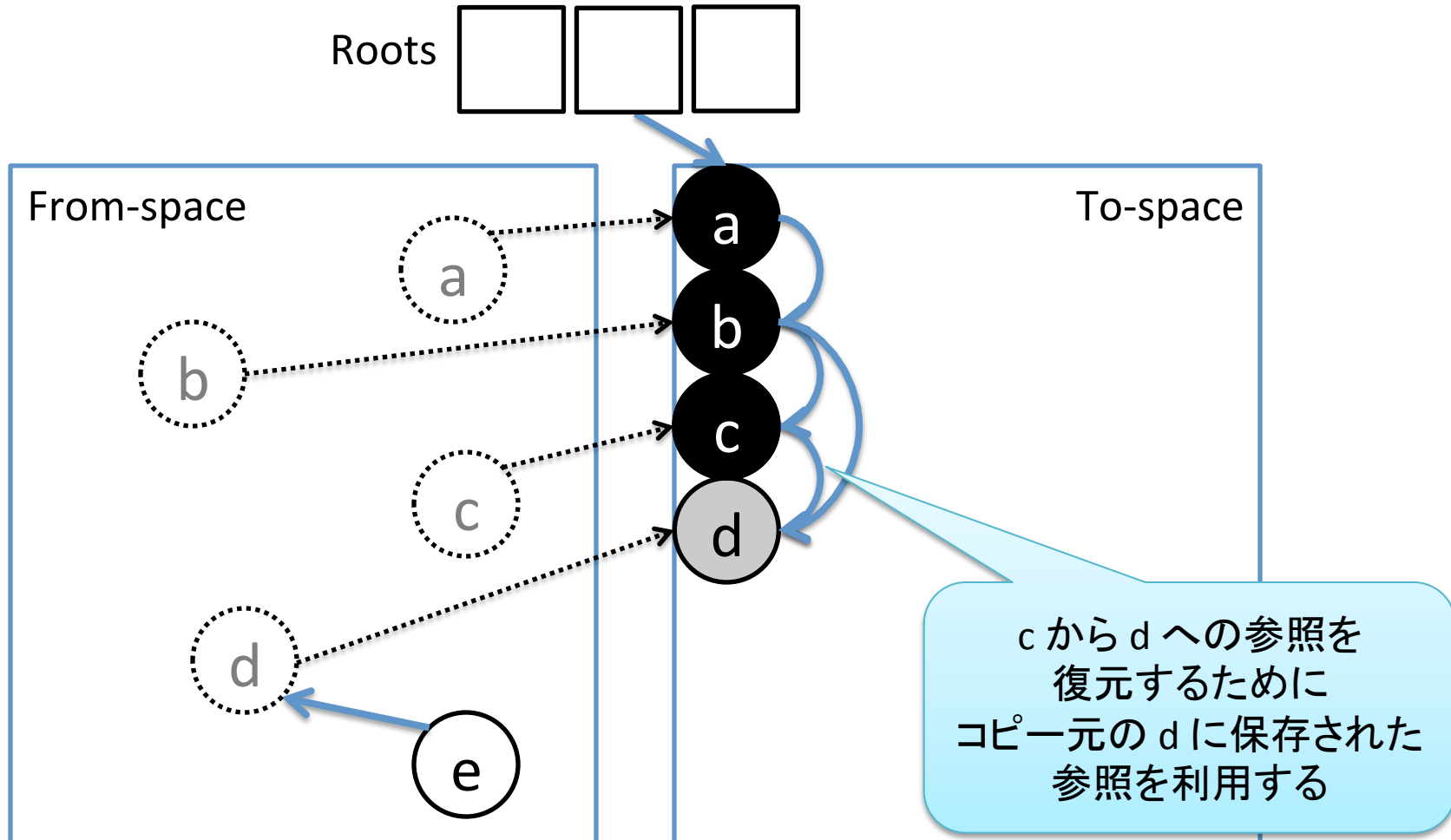
Copying GC の例

「灰」のオブジェクトから参照されるオブジェクトをコピー



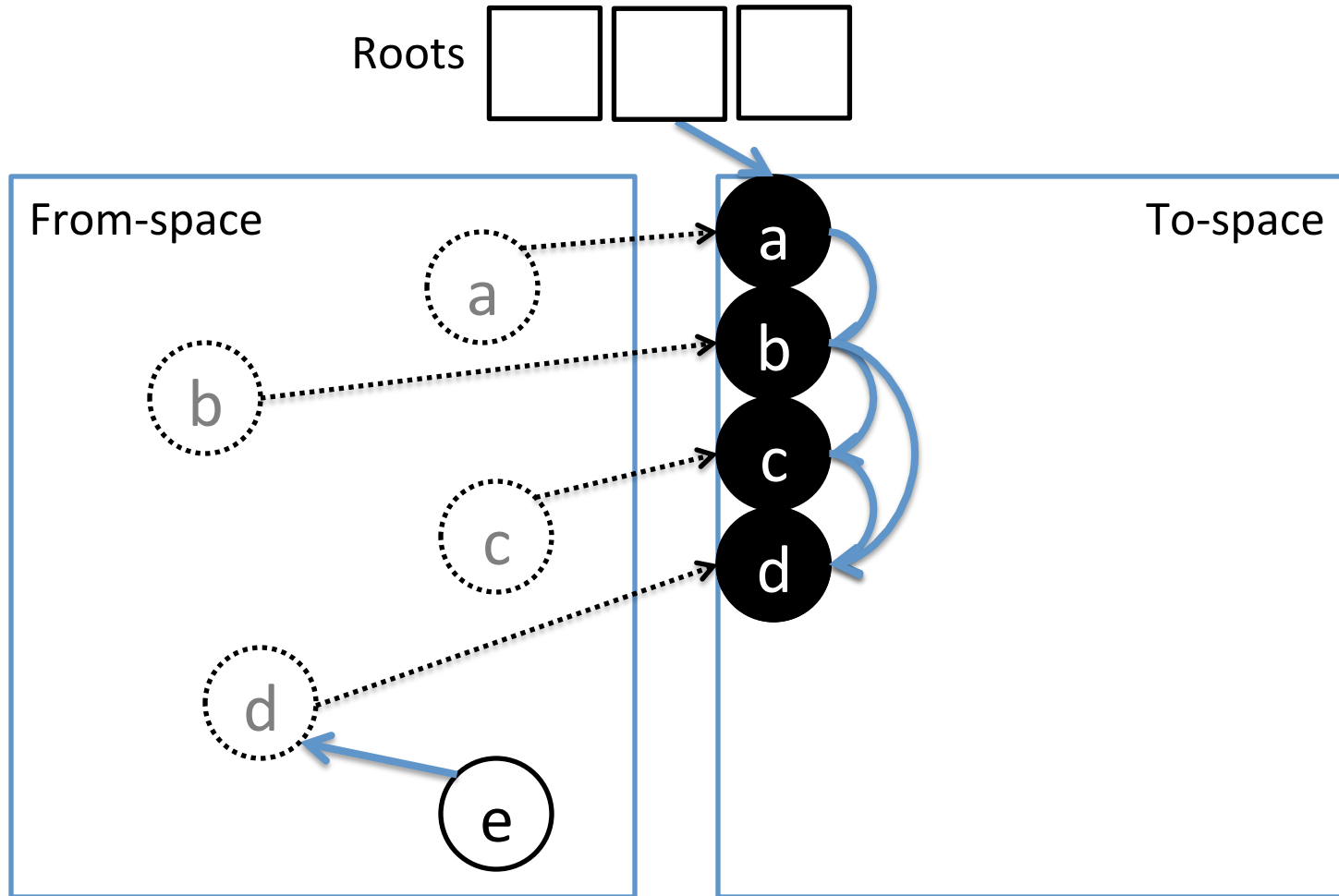
Copying GC の例

「灰」のオブジェクトから参照されるオブジェクトをコピー

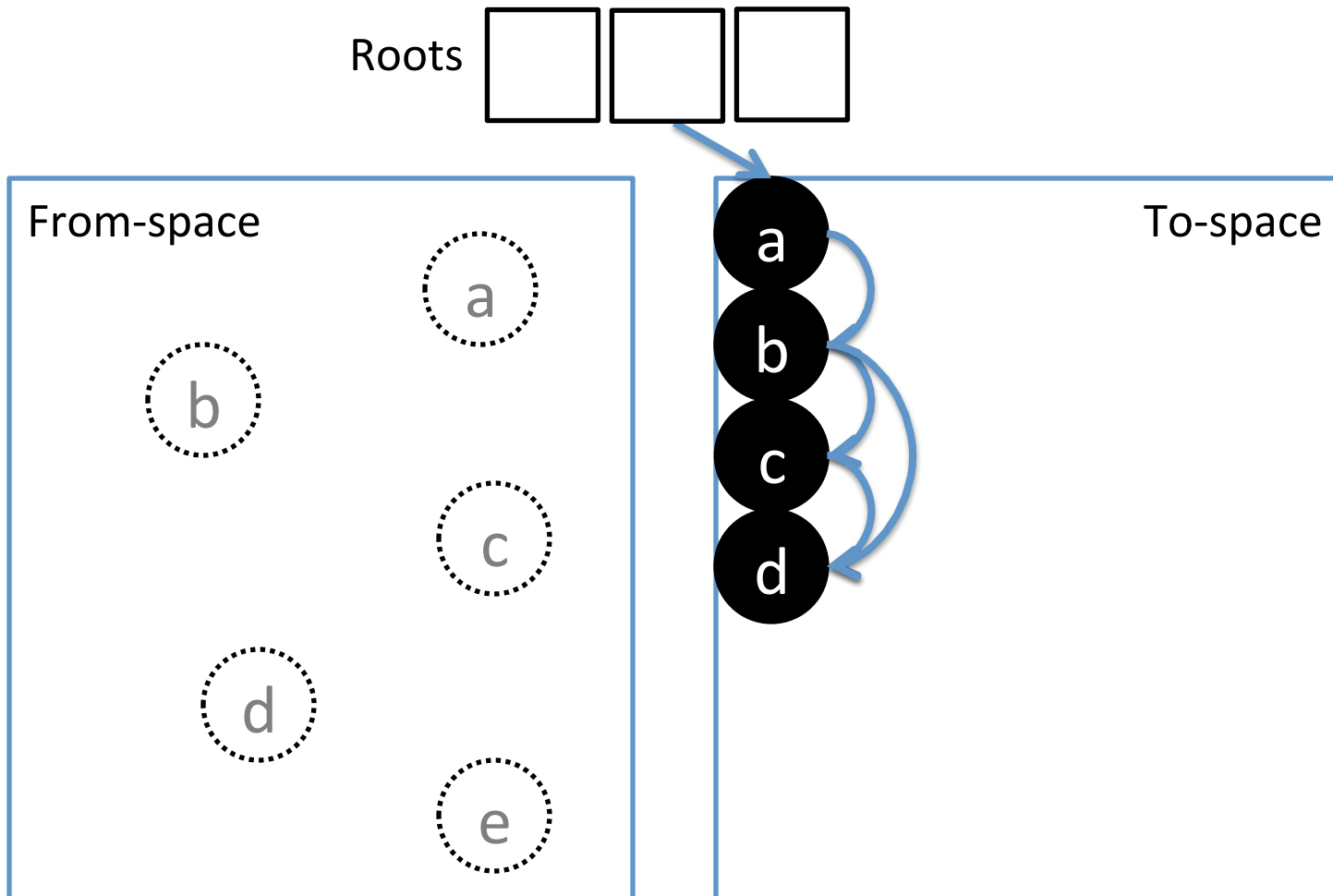


Copying GC の例

「灰」のオブジェクトから参照されるオブジェクトをコピー



Copying GC の例 (GC 終了)



GC の利点・欠点

- malloc/free vs. 参照カウント vs. tracing GC

	malloc/free	参照カウント	Tracing GC
プログラムの書きやすさ	×	○ (△?)	○
停止時間	○	○	×
実行時間全体	○	×	△ (○?)
メモリ効率	○	△	×

- マーク & スweep vs. copying
 - メモリ確保は大抵 copying GC が勝つ
 - メモリ効率は大抵マーク & スweep が勝つ
 - 1 回の GC にかかる時間はスweepを lazy に行うマーク & スweep が勝つ

Tracing GC の改良

- インクリメンタル GC: ちよつとづつ GC
 - GC の処理を細切れにして
プログラムの実行と交互に行う手法
- 世代別 GC (Generational GC)
 - ヒープを幾つかの「世代」に分ける手法
 - 大抵のオブジェクトはすぐ不要になるので
「若い世代」の GC だけで済ませる
 - 寿命の長いオブジェクトの探索回数を減らせる

ただし、どちらの手法も write barrier が必要になる

Write barrier = ここでは、オブジェクトへの書き込み時に特別な処理を行うこと

GC のその他の話題

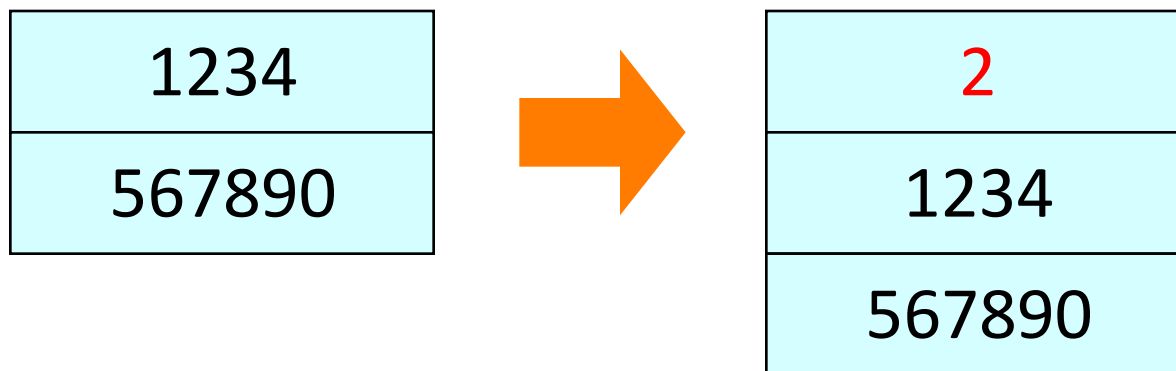
- 保守的な GC (conservative GC)
- 並行 GC (concurrent GC) と
並列 GC (parallel GC)
- 分散環境の GC
 - 分散参照カウント
 - 分散 tracing GC

“Conservative” garbage collection とは

- 参照と整数の区別がつかない環境でも行える garbage collection のこと
 - 例えば C 言語ではメモリの中身を見ても整数とポインタを区別できない
- 解決方法:
とりあえず全部ポインタだと思う

配列境界検査

- GC のための拡張と同様に
配列データに長さを表すタグを付加する
- 配列アクセスを行う命令列の前に
配列長と添字を比較する命令を出力する



共通課題

- MinCaml に以下の改造を加えよ
 - 配列長を表すタグを配列に付加する
 - 配列アクセスの前に境界検査を行い配列の範囲外にアクセスしようとしていたらプログラムを終了させるなどの処理を行う

コンパイラ係用選択課題

- Garbage collection を自作コンパイラまたは MinCaml に実装せよ
 - 使用する GC アルゴリズムは自由
 - ゴミをたくさん作りながら動くサンプルプログラムを記述しそれが複数回の GC を経て動き続けることを確認すること

演習でどの GC を実装するか？

- たぶん copying GC が一番楽
- 興味があれば mark-sweep GC に挑戦してみるのもよい
- コンパイラ側の改造が多いのをいとわなければ reference count もおもしろい
- さらに余裕があれば incremental GC や generational GC も

GC 導入のための コンパイラの変更 (1)

- ヒープを作成する (stub の) コードの改造
 - Copying GC を採用する場合はヒープを 2 つ用意する等
- ヒープ内にメモリを確保するコードの改造
 - ヒープポインタとヒープリミットを比較し
ヒープがあふれそうなら GC ルーチンに飛ぶ
- ヒープ内のデータを扱うコードの改造
 - 配列やタプルなどに型やサイズを表すタグを付加
 - 採用する GC によっては
mark bit 領域や reference count 領域などを追加し
それらを操作するコードも追加

GC 導入のための コンパイラの変更 (2)

- GC が rootset を得るための機構の追加
 - スタックに積まれた各ワードの型を
GC ルーチンが知るための機構の導入
 - フレームに保存された PC の値から
そのフレームの各ワードの型を得られるような表を作る
 - スタックに値をプッシュする際に
その値の型の情報もどこかに書き込んでおく
 - など
 - 大域変数のアドレスと型を
GC ルーチンが知るための機構の導入

課題の提出先と締め切り

- 提出先: `compiler-report-2011@yl.is.s.u-tokyo.ac.jp`
- 締め切り: 2 週間後 (1/19) の午後 1 時
 - コンパイラ係用課題の締め切り: 2012 年 2 月 27 日
- Subject: **Report 12** <学籍番号: 5 桁>


半角スペース 1 個ずつ

 - 例: **Report 12 11099**
- 本文にも氏名と学籍番号を明記のこと
- ◆ 質問は `compiler-query-2011@yl.is.s.u-tokyo.ac.jp` まで

コンパイラ系の 成績評価について (1)

- 選択課題を 1 つ以上提出してください
 - また選択課題と同等以上の難度の言語機構の実装をもって選択課題の提出とみなすことも可能です
 - 事前に相談してください

コンパイラ系の 成績評価について (2)

- 各コンパイラ係に前田と TA が面談をします
 - 基本的にはレイトレ競技会の当日または付近の日
 - `compiler-query-2011@yl.is.s.u-tokyo.ac.jp`にメールして予約をとって下さい
 - 場所は地下端末室、時間は 20 分程度
 - やること:
 - 自作コンパイラの特徴や独創的な点などの説明
 - 自作コンパイラによる、プログラム（レイトレ含む）のコンパイル・実行のデモ
 - 自作コンパイラでコンパイルしたレイトレが自作 CPU またはシミュレータ上で動く様子を見せて下さい