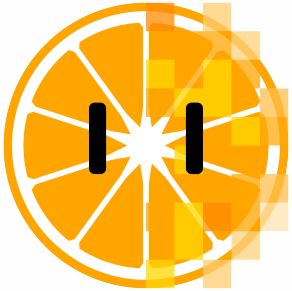
PHP 処理系の
garbage collection を理解する
～メモリはいつ解放されるのか～

nsfisis (いまむら)

PHP カンファレンス名古屋 2025

いまむら

nsfisis



@ デジタルサーカス株式会社

メモリ

コンピュータの一時データ置き場

メモリの容量は増え続けている

メモリの容量は増え続けている

無限ではない

使い果たすと ……

使い果たすと .....

- プロセスの強制終了
- 極端なパフォーマンスの低下

メモリ管理

メモリ管理

- 必要なメモリを確保する
- 不要なメモリを解放する

メモリの「確保」(割り当て)

必要なメモリのサイズを要求する

メモリの「解放」

不要になったメモリを再び利用可能にする

メモリ管理は難しい

メモリ管理は難しい

- Memory leak
- Double free
- Use-after-free
- Buffer overflow

PHP でメモリ管理を直接おこなう必要はない

PHP でメモリ管理を直接おこなう必要はない

Garbage Collection

Garbage Collection (GC)

Garbage Collection (GC)

メモリ管理の自動化

自動でメモリを解放

PHP の GC

PHP の GC

参照カウント + マークアンドスイープ

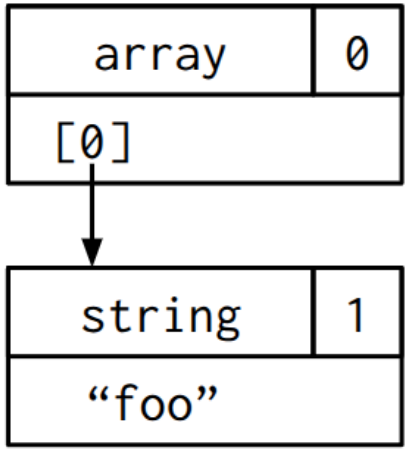
参照カウント

参照カウント

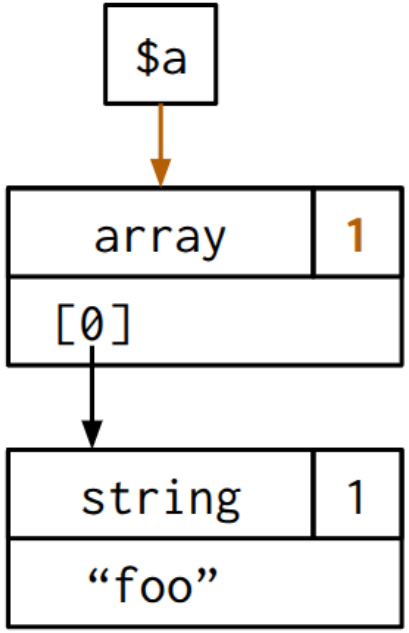
参照されている数を数える
ゼロになったら解放する

```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```

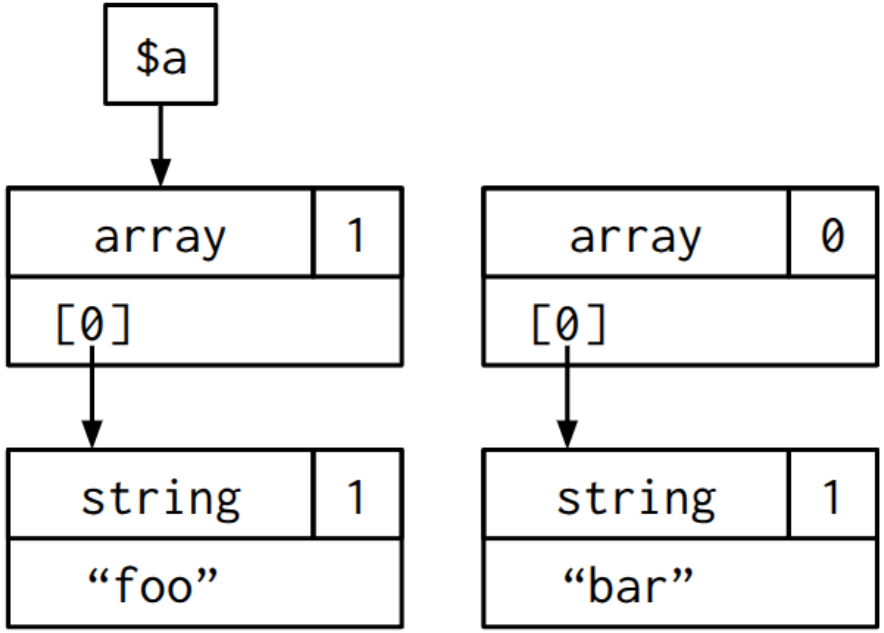
```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```



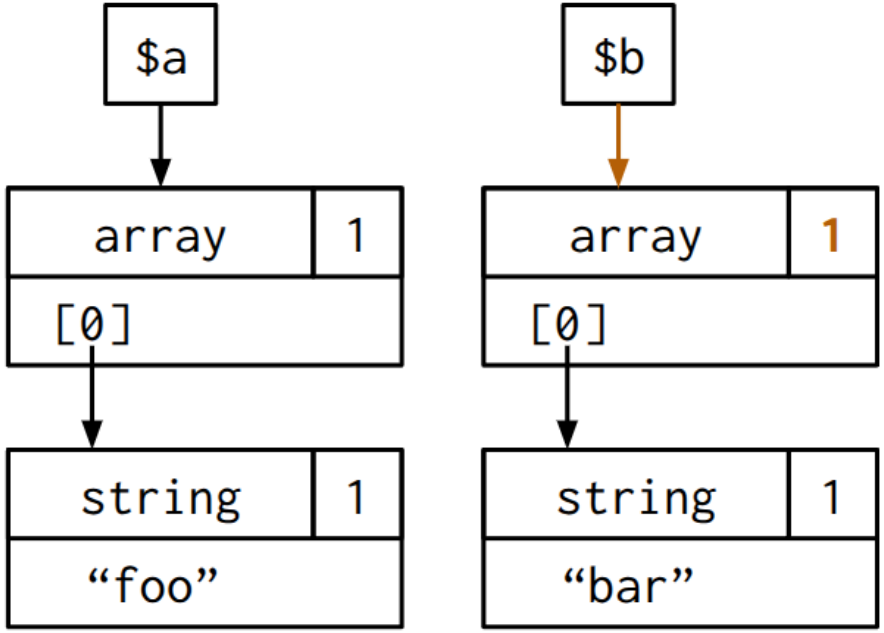
```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```



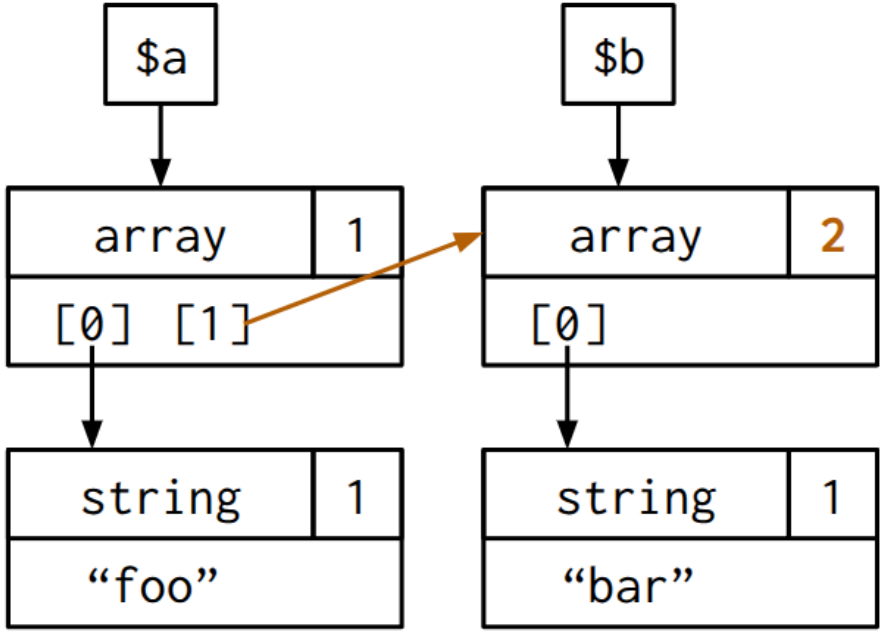
```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```



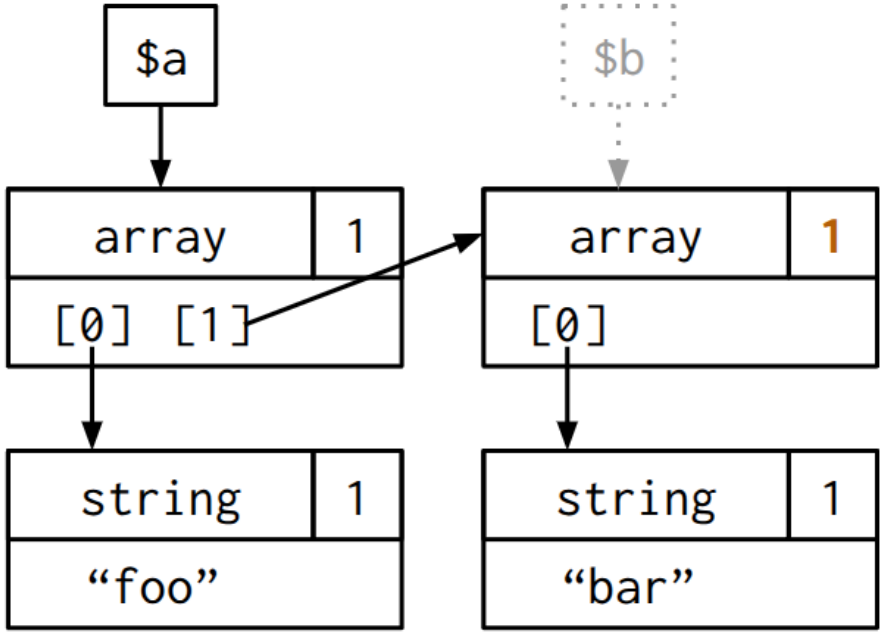
```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```



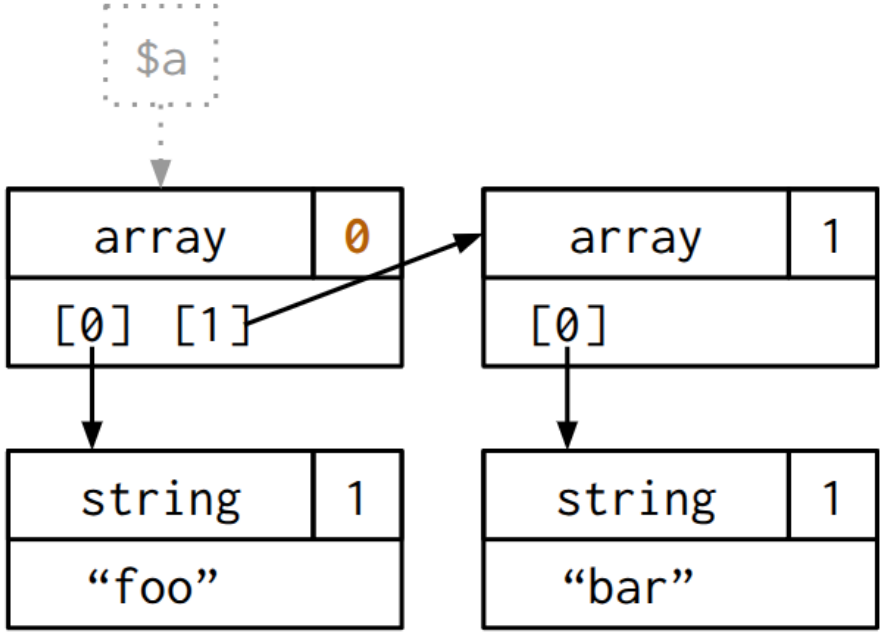
```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```



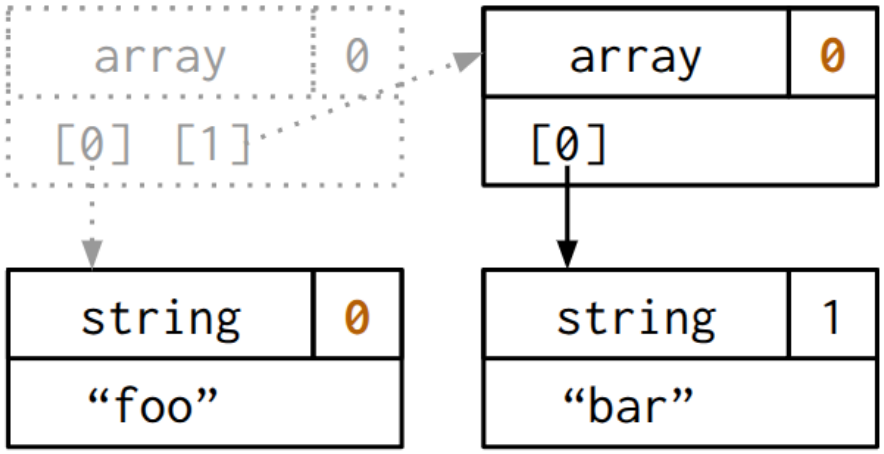
```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```



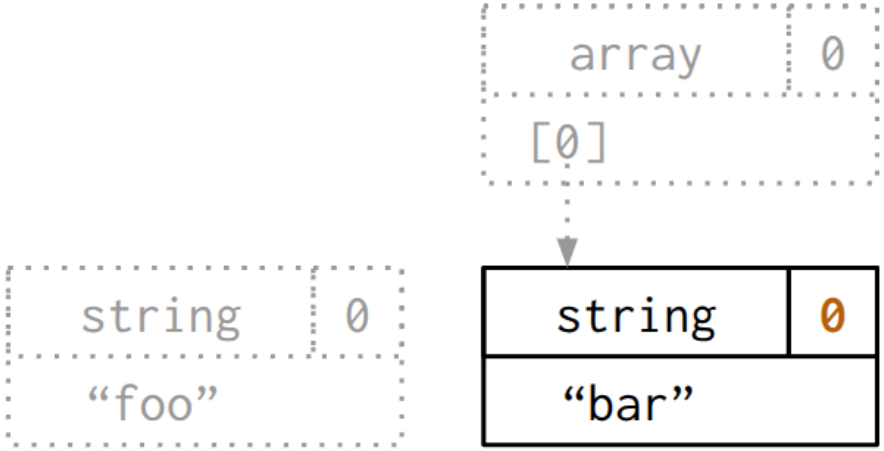
```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```



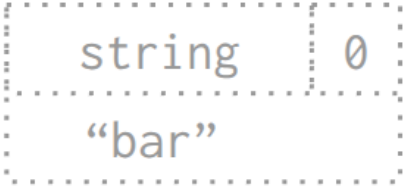
```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```



```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```



```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```



```php
$a = ["foo"];
$b = ["bar"];
$a[] = $b;
unset($b);
unset($a);
```

参照カウントの利点

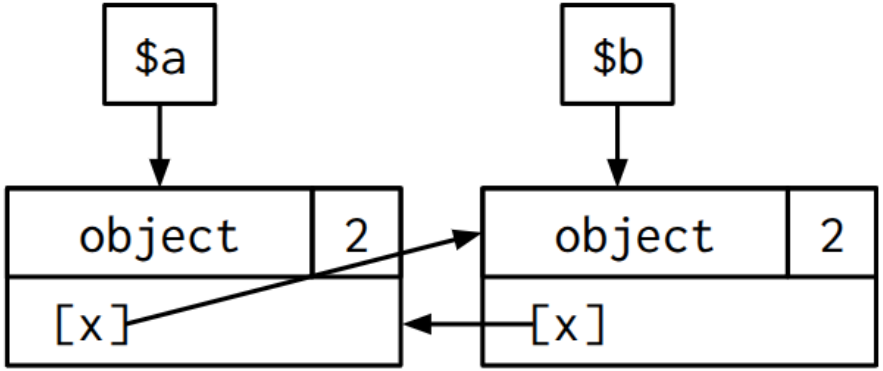
- 未使用オブジェクトが即座に解放される
- 解放にかかるコストが全体に分散する

参照カウントの欠点

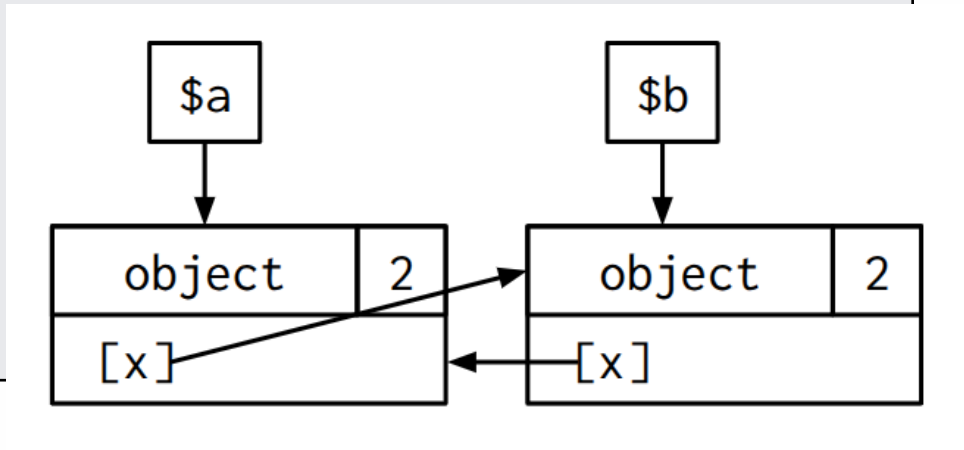
- 参照の追加にコストがかかる
- マルチスレッド / プロセス環境に向かない
- 循環参照を扱えない

```
$a = new stdClass();
$b = new stdClass();
$a->x = $b;
$b->x = $a;
```

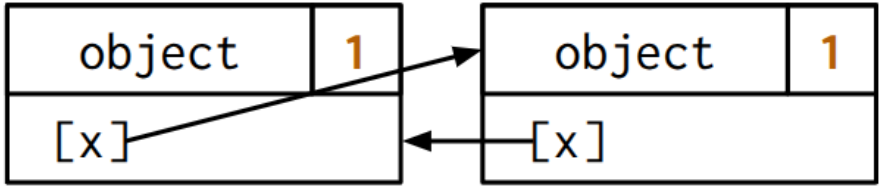
```php
$a = new stdClass();
$b = new stdClass();
$a->x = $b;
$b->x = $a;
```



```php
$a = new stdClass();
$b = new stdClass();
$a->x = $b;
$b->x = $a;
unset($b);
unset($a);
```



```php
$a = new stdClass();
$b = new stdClass();
$a->x = $b;
$b->x = $a;
unset($b);
unset($a);
```



参照カウントでは循環参照を解放できない

マークアンドスイープ (mark & sweep)

マークアンドスイープ (mark & sweep)

PHP では循環参照の解放のみ担う

マークアンドスイープ (mark & sweep)

- 基本的なマークアンドスイープ
- PHP でのマークアンドスイープ

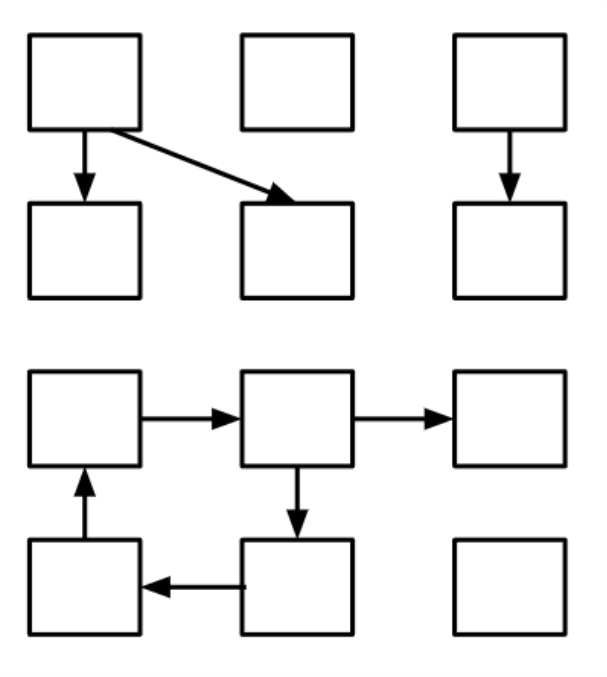
マークアンドスイープの流れ

- マークフェーズ
- スイープフェーズ

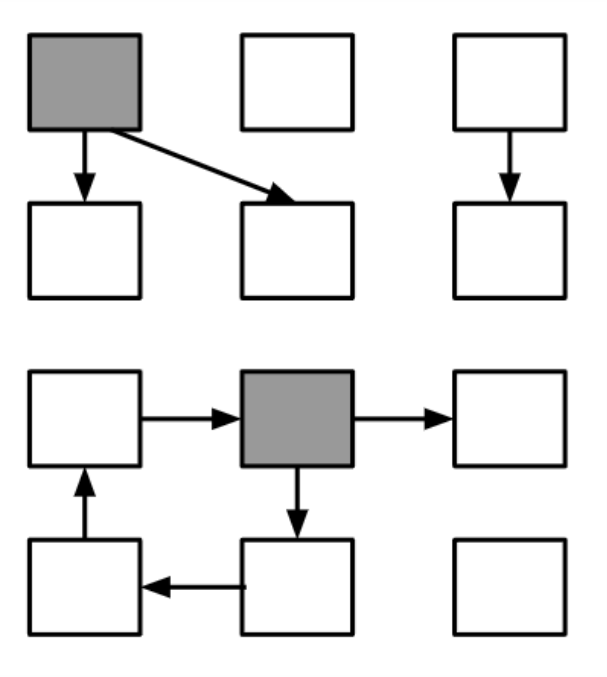
マークフェーズ

- 確実に使われているオブジェクト (ルート) に印を付ける
- そこから辿れるオブジェクトに印を付ける

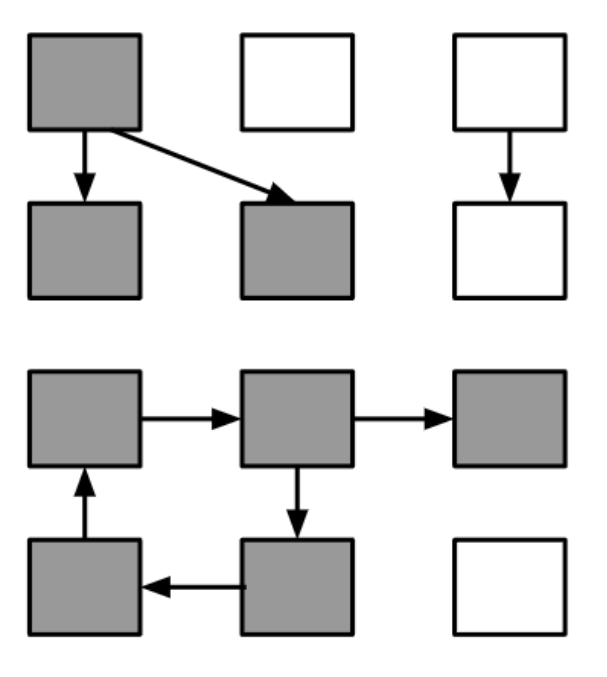
マークフェーズ



マークフェーズ



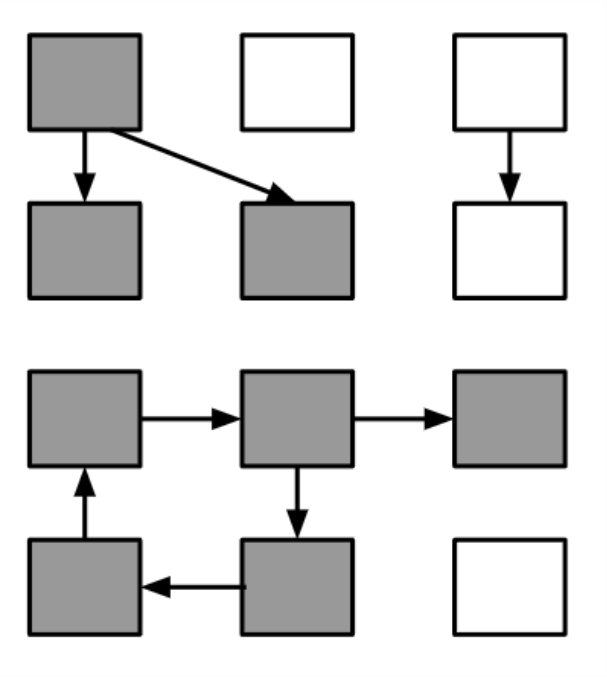
マークフェーズ



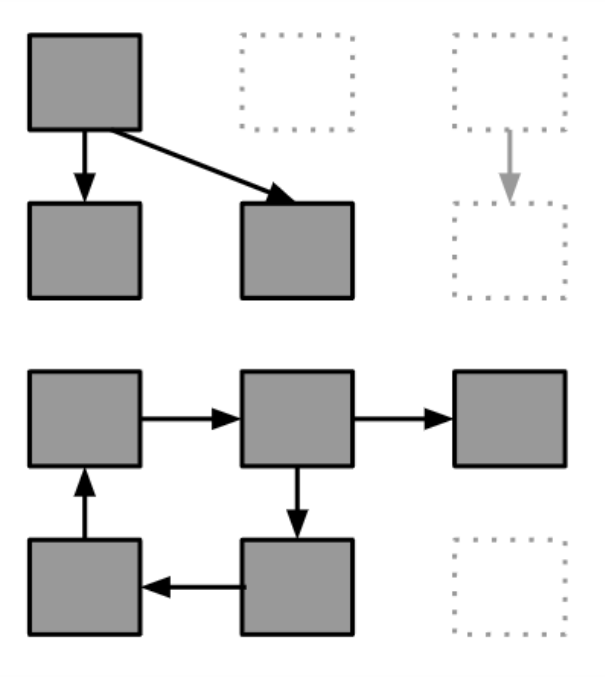
スイープフェーズ

- 全オブジェクトを調べて、印が付いていなければ解放する

スイープフェーズ



スイープフェーズ



PHP でのマークアンドスイープ

PHP でのマークアンドスイープ

ほとんどのオブジェクトは参照カウントで解放される
全オブジェクトを走査しなくていい

PHP でのマークアンドスイープ

循環参照「かもしれない」オブジェクトを登録しておく
そのオブジェクトとそこから辿れるオブジェクトだけ走査する

PHP でのマークアンドスイープ

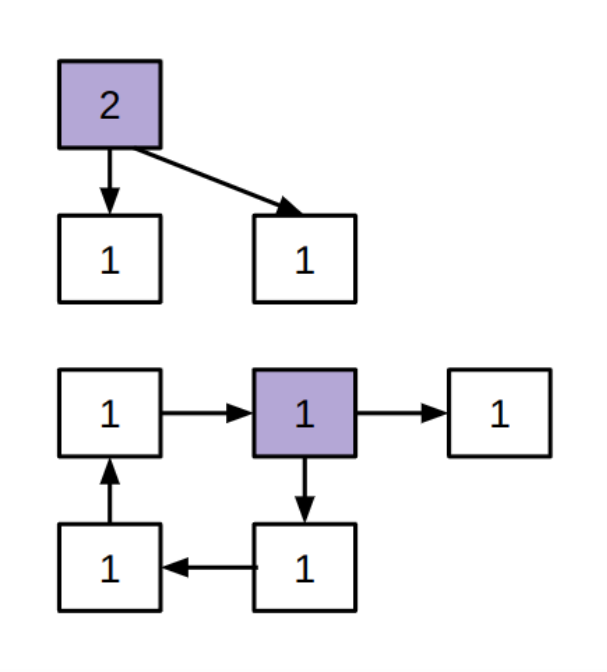
循環参照「かもしれない」オブジェクト

refcount を減らしたときに 0 にならなかったもの
循環参照は、間接的に自分で自分を指している

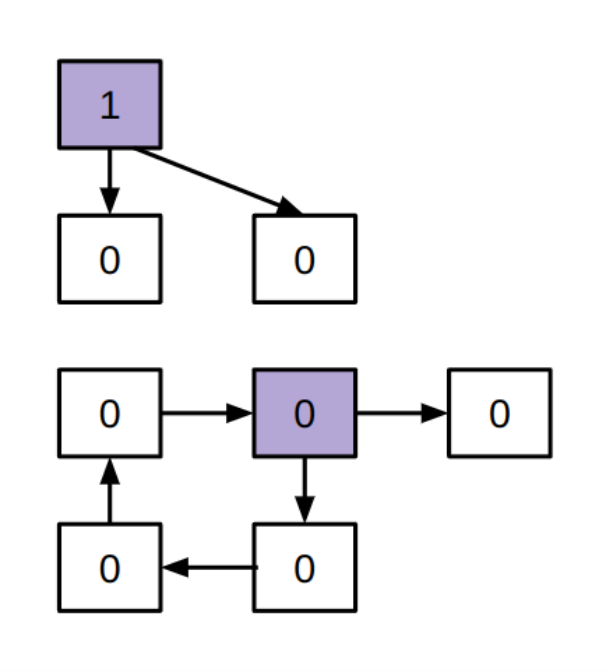
マークフェーズ

- 循環参照かもしれないオブジェクトをルートバッファへ登録する
- ルートバッファから辿れる全オブジェクトの ref-count を 1 減らす

マークフェーズ



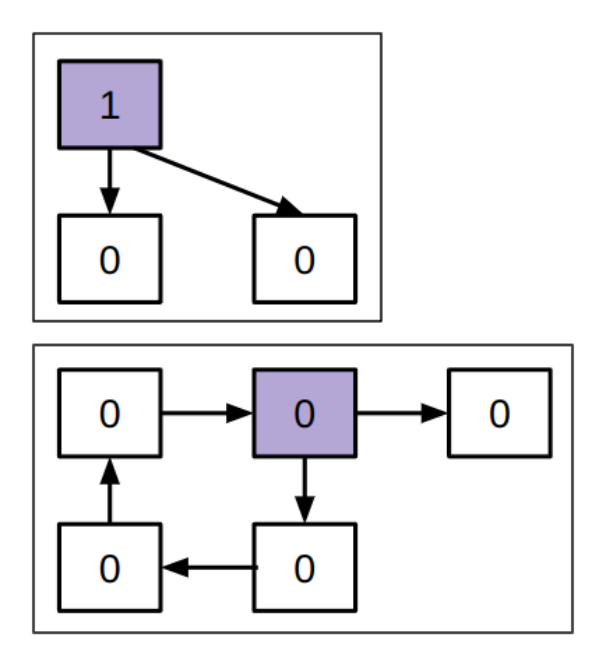
マークフェーズ



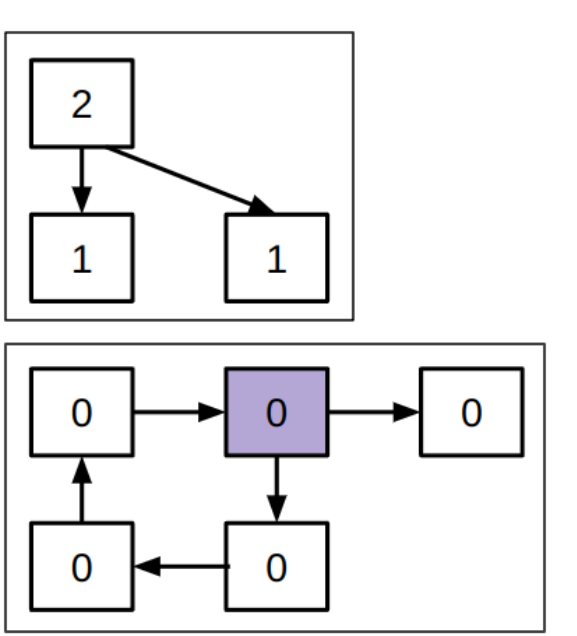
スイープフェーズ

- すべてのルートバッファから辿れるオブジェクトについて、
  - refcount が 0 でないなら 1 増やす
  - refcount が 0 なら解放する
- 処理後はルートバッファから取り除く

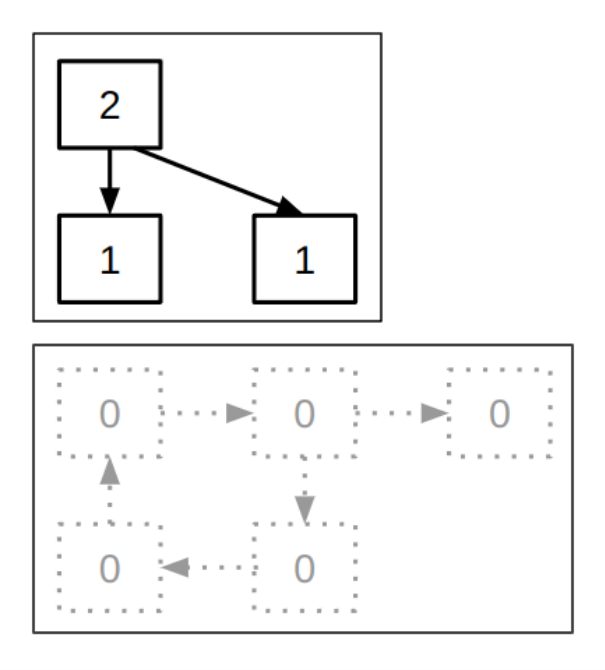
スイープフェーズ



スイープフェーズ



スイープフェーズ



マークアンドスイープが走るタイミング

- `gc_collect_cycles()` を呼んだとき
- ルートバッファが一杯になったとき
  - デフォルトは 10,000

マークアンドスイープの利点

- 循環参照を解放できる
- GC が動いていないときのオーバーヘッドがない

マークアンドスイープの欠点

- GC に時間がかかる

PHP の GC

参照カウント + マークアンドスイープ

PHP の GC

参照カウント + マークアンドスイープ

循環参照以外は参照カウントで、
循環参照はマークアンドスイープで解放

PHP の GC

参照カウント + マークアンドスイープ

多くの (循環参照でない) オブジェクトは未使用になると即座に解放される
循環参照を形成しているオブジェクトは遅れて解放される

`fclose()` は明示的に呼ぶべきか？

`fclose()` は明示的に呼ぶべきか？

明示的に呼ばなくても、GC で解放されたタイミングでクローズされる

`fclose()` は明示的に呼ぶべきか？

常に呼ぶべき

`fclose()` は明示的に呼ぶべきか？

常に呼ぶべき

- 呼ばなくても問題ないかを判定するのが困難
- 呼ばなくても問題ない状態を維持できるとは限らない

`fclose()` は明示的に呼ぶべきか？

常に呼ぶべき

- 呼ばなくても問題ないかを判定するのが困難
- 呼ばなくても問題ない状態を維持できるとは限らない

循環参照になっていないか、 解放が遅れても問題ないなら呼ばなくてよい

話せなかったこと

- 複数スレッド / プロセス間での共有
- リクエスト毎に確保・解放されるメモリとグローバルなメモリ
- Copy on Write
- 弱参照
- `memory_limit`