

# 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

(長期的・平均的には) メモリの容量は増え続けている

(長期的・平均的には) 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)

必要なサイズのメモリを要求する

# 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

自分を参照している別のオブジェクトの数 (参照カウント) を数えておく

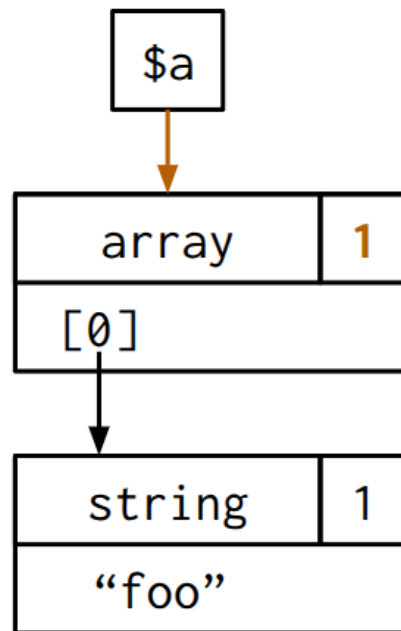
参照が増えたらカウントアップ

参照が減ったらカウントダウン

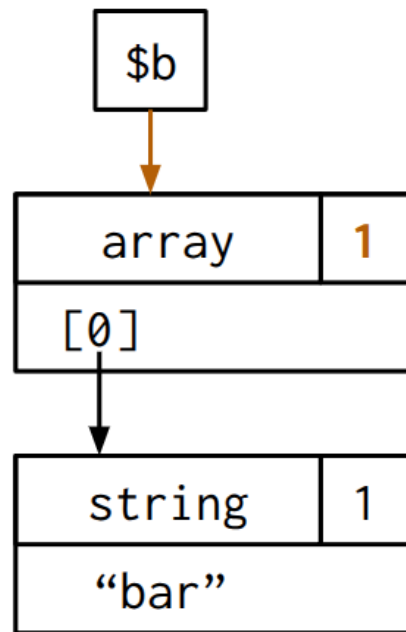
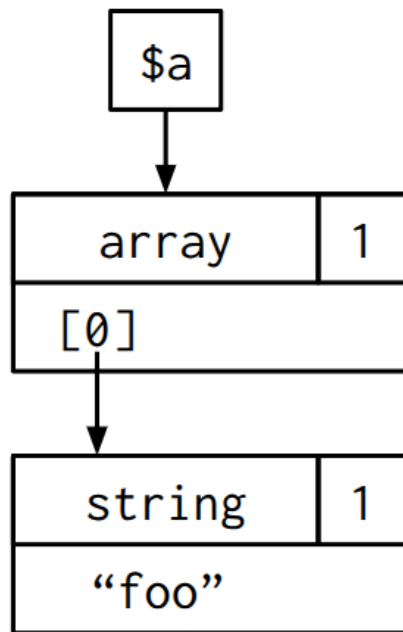
ゼロになったら解放する

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

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

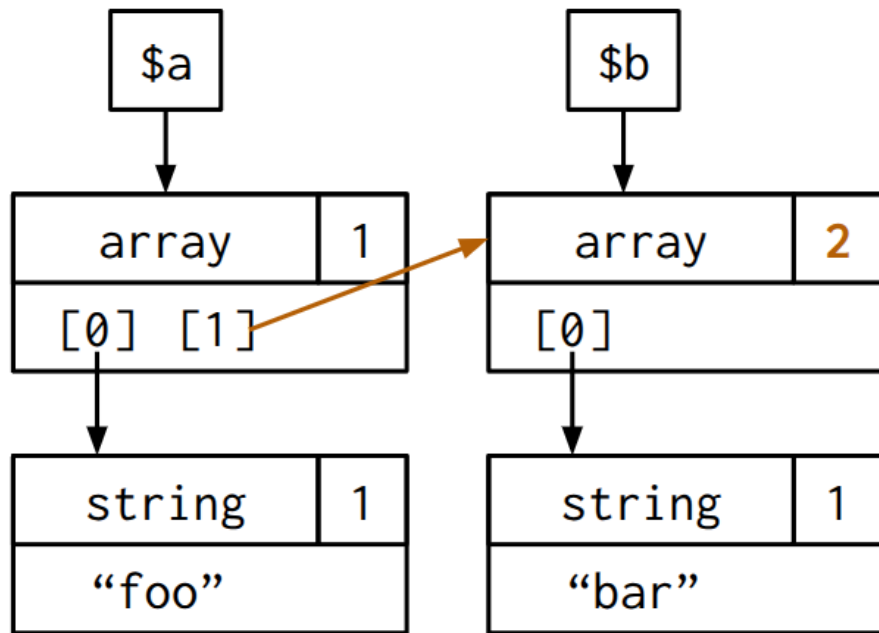


```
$a = ["foo"];  
$b = ["bar"];  
$a[] = $b;  
unset($b);  
unset($a);
```

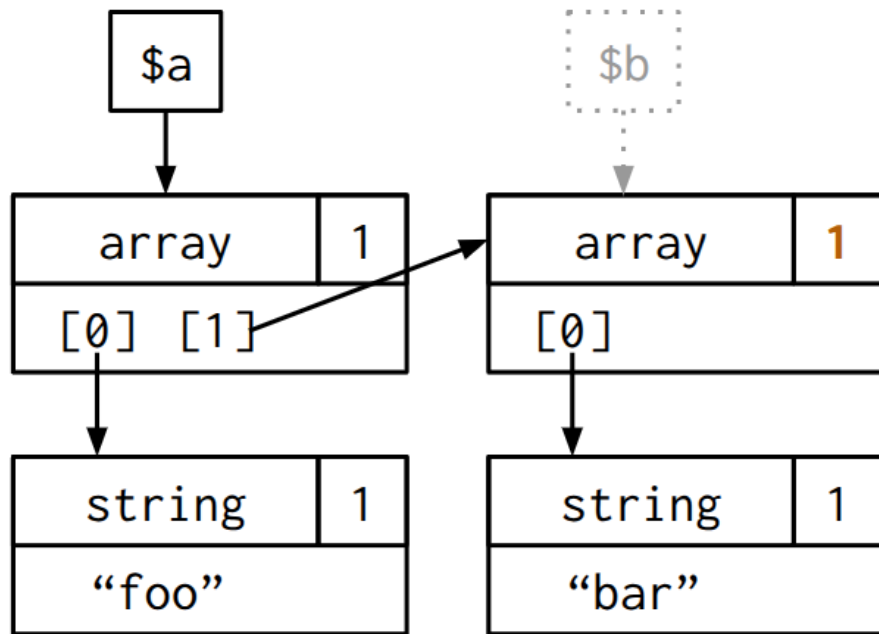




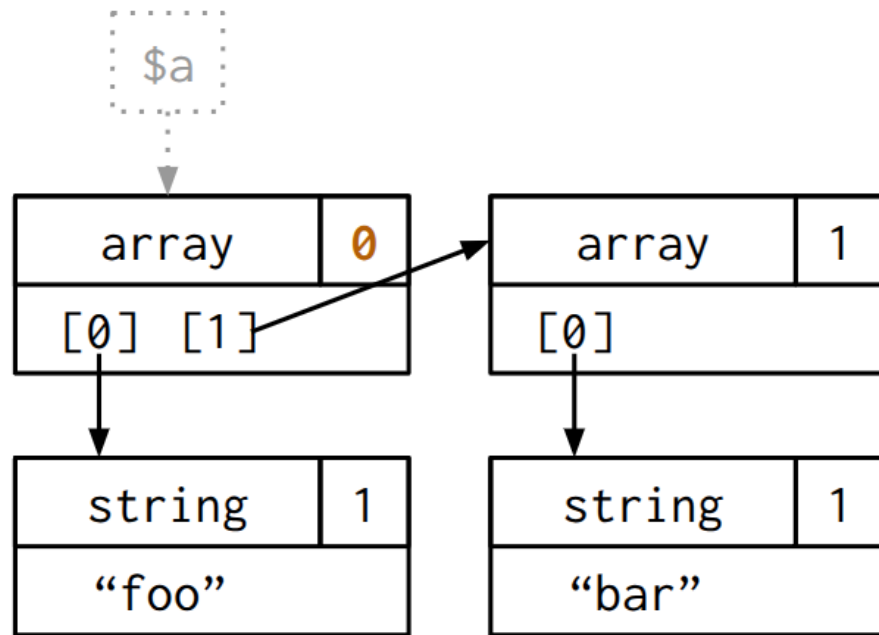
```
$a = ["foo"];  
$b = ["bar"];  
$a[] = $b;  
unset($b);  
unset($a);
```



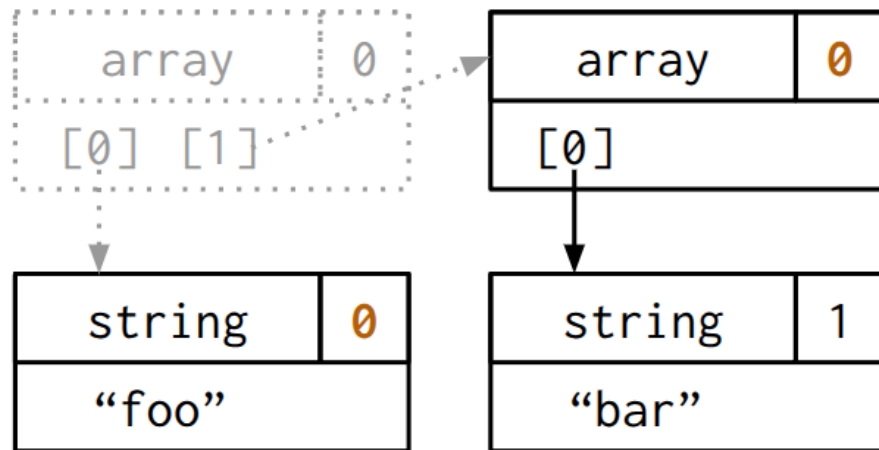
```
$a = ["foo"];  
$b = ["bar"];  
$a[] = $b;  
unset($b);  
unset($a);
```



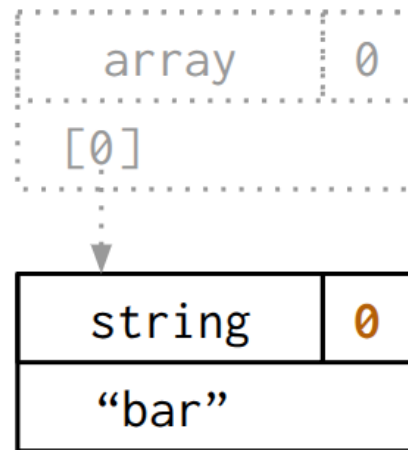
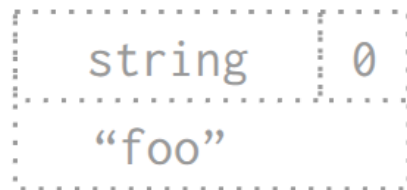
```
$a = ["foo"];  
$b = ["bar"];  
$a[] = $b;  
unset($b);  
unset($a);
```



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



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



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

string	0
"bar"	

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

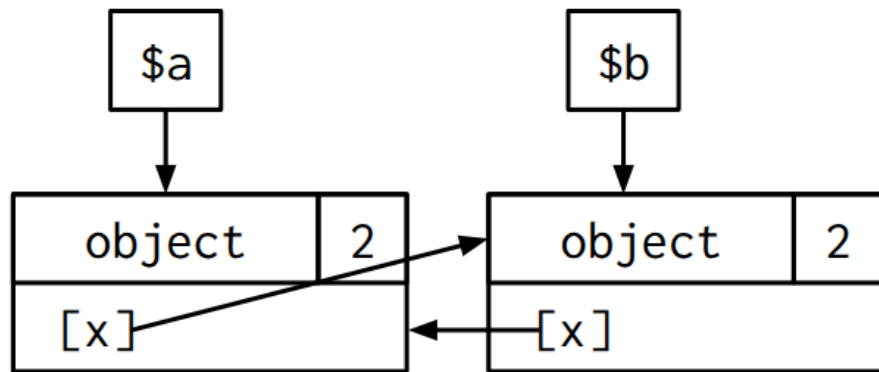
## 参照カウントの特徴

- 未使用オブジェクトが即座に解放される
- 循環参照を扱えない

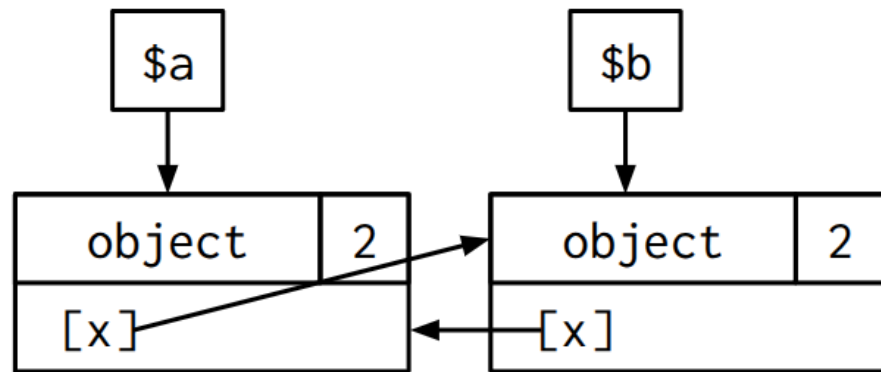


```
$a = new stdClass();  
$b = new stdClass();  
$a->x = $b;  
$b->x = $a;
```

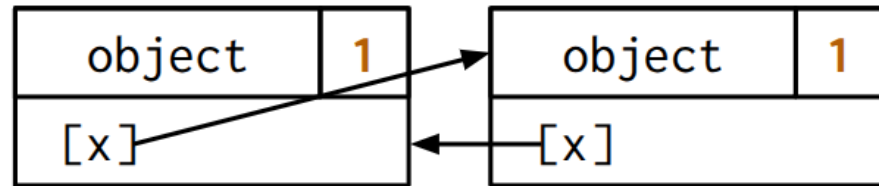
```
$a = new stdClass();  
$b = new stdClass();  
$a->x = $b;  
$b->x = $a;
```



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



```
$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



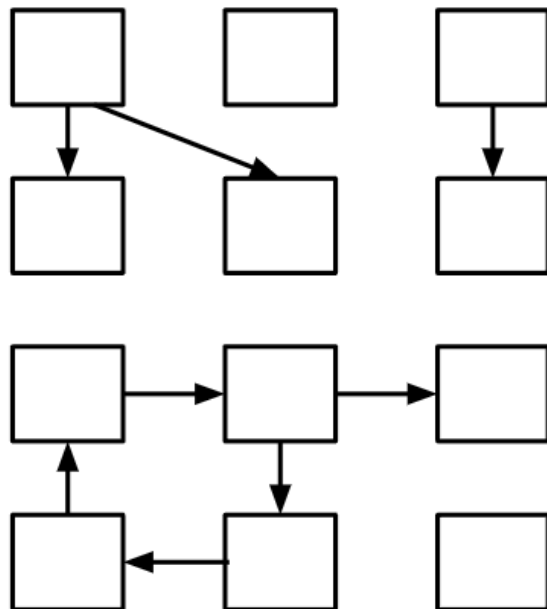
# マークアンドスイープの流れ

- マークフェーズ
- スイープフェーズ

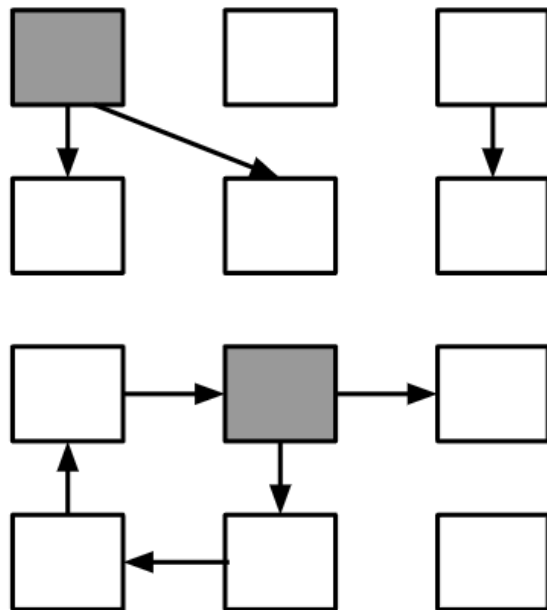
# マークフェーズ

- 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

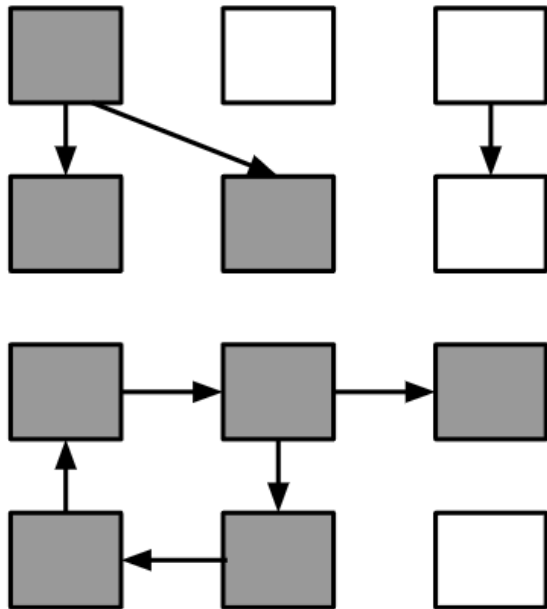
# マークフェーズ



# マークフェーズ



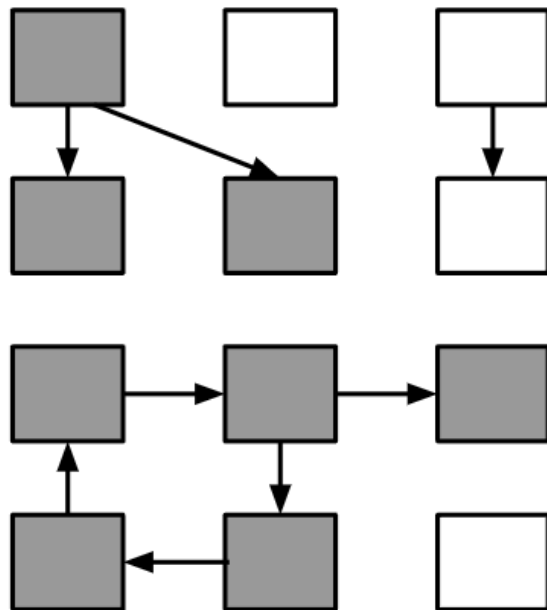
# マークフェーズ



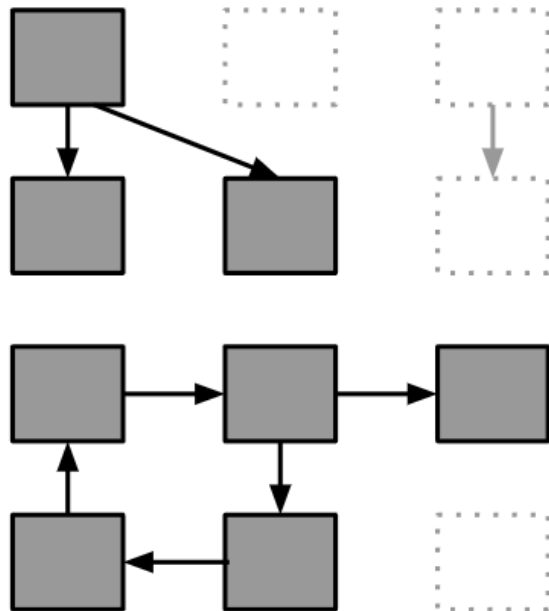
# スリープフェーズ

- 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weepフェーズ



# スリープフェーズ





# PHP でのマークアップスイープ

# 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

循環参照だけ考えればいい

# PHP でのマークアンドスイープ

循環参照「かもしれない」オブジェクトをルートバッファに登録しておく

ルートバッファから本物の循環参照を探す

# 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

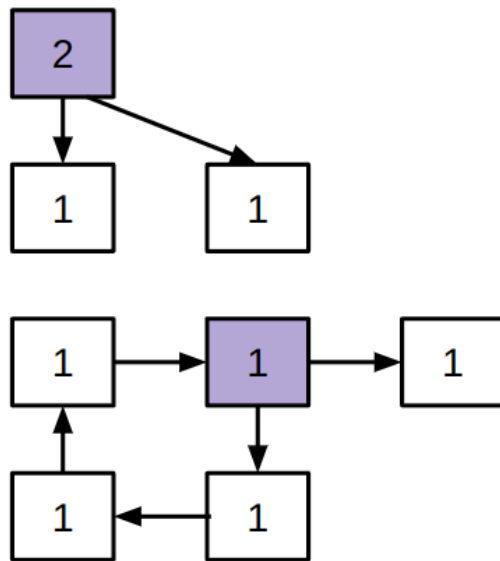
参照カウントを減らしたときに 0 にならなかったもの

循環参照は、間接的に自分で自分を指している

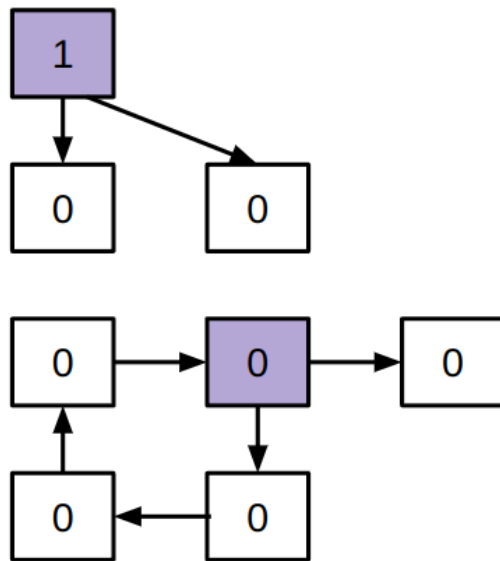
## マークフェーズ

- ルートバッファから辿れる全オブジェクトの参照カウントを 1 減らす

# マークフェーズ



# マークフェーズ

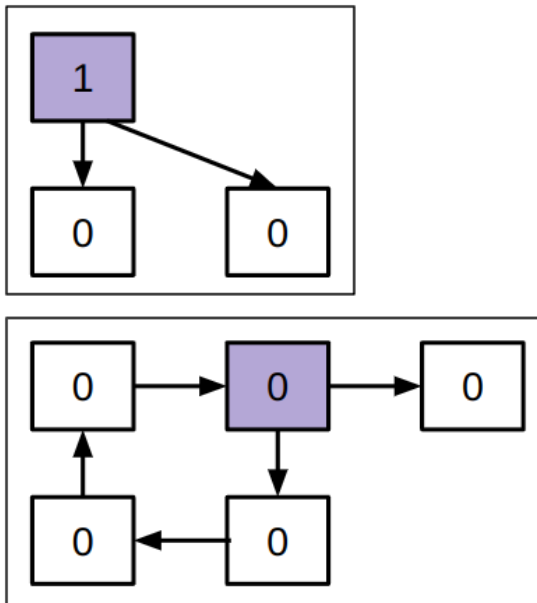


# スリープフェーズ

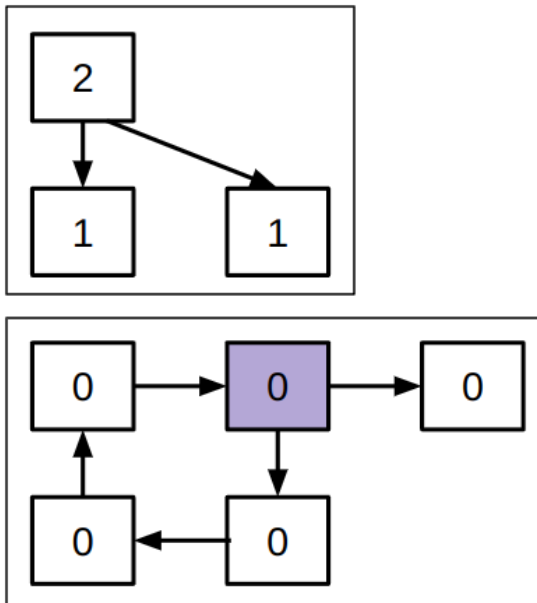
- すべてのルートバッファから辿れるオブジェクトについて、
  - 参照カウントが 0 でないなら 1 増やす
  - 参照カウントが 0 なら解放する
- 処理後はルートバッファから取り除く



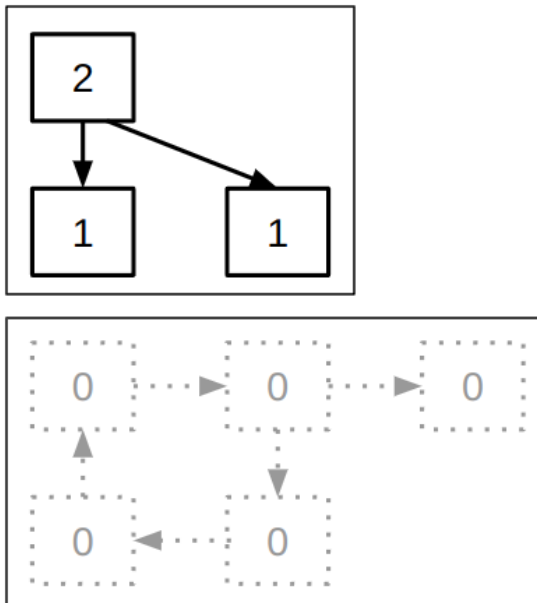
# スリープフェーズ



# スリープフェーズ



# スワイプフェーズ



# 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

# 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

循環参照でない多くの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

おわり

以降時間あれば



`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`