

# サンプリング適用による 動的データ構造推定の 精度・効率の評価

2018/11/01

東京工業大学 情報理工学院 情報工学系 権藤研究室

小泉 雄太, 荒堀 喜貴, 権藤 克彦

## 0. ABSTRACT

---

- 問題①:
  - 動的アクセスベースのデータ構造解析技術  
→高精度だが動的オーバーヘッドが大きくなる傾向
  - 提案①:
    - サンプリング技術と組み合わせることでオーバーヘッド削減
- 問題②:
  - サンプリングで非冗長な情報が間引かれる可能性  
→特に大きな配列に影響
  - 提案②:
    - 非冗長な情報を冗長な情報へ変換（基底構造）
    - 非冗長な情報を補完（配列予測）
- 小ベンチマーク実験:
  - オーバーヘッド削減が可能なことを確認
  - サンプリングで精度が保存/補完される例と、不正確となる例を確認

## 1. INTRODUCTION

# DATA STRUCTURE ANALYSIS

## 1. INTRODUCTION

### DATA STRUCTURE ANALYSIS

---

- データ構造解析(Data Structure Analysis):  
プログラムの持つデータ構造を解析すること
  - データ構造:
    - 変数とその型
    - 構造体とそのフィールド
    - 配列とその要素数
    - 木構造などの集合体
    - etc.
  - 解析領域:
    - 静的領域
    - 動的領域
    - スタックフレーム
  - 解析手法:
    - 静的手法
    - 動的手法

## 1. INTRODUCTION

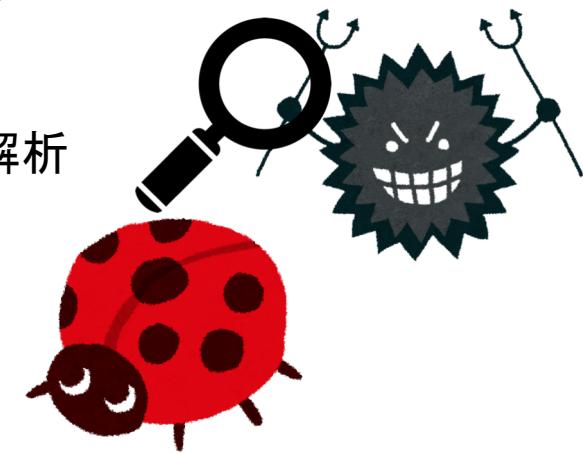
### DATA STRUCTURE ANALYSIS

- データ構造解析(Data Structure Analysis):  
プログラムの持つデータ構造を解析すること
  - データ構造:
    - 変数とその型
    - 構造体とそのフィールド
    - 配列とその要素数
    - 木構造などの集合体
    - etc.
  - 解析領域:
    - 静的領域
    - 動的領域
    - スタックフレーム
  - 解析手法:
    - 静的手法
    - 動的手法

- デバッグ補助  
シンボルテーブル復元  
視覚化 など

- マルウェア解析

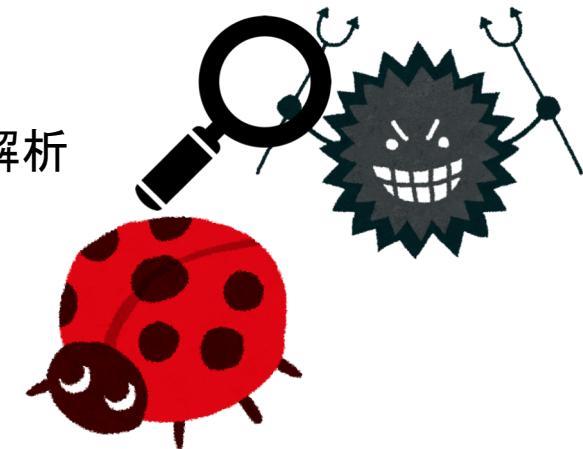
- 脆弱性検査  
バッファオーバーフロー検査など



## 1. INTRODUCTION

### DATA STRUCTURE ANALYSIS

- データ構造解析(Data Structure Analysis):  
プログラムの持つデータ構造を解析すること
  - データ構造:
    - 変数とその型
    - 構造体とそのフィールド
    - 配列とその要素数
    - 木構造などの集合体
    - etc.
  - 解析領域:
    - 静的領域
    - **動的領域**
    - スタックフレーム
  - 解析手法:
    - 静的手法
    - **動的手法**
- デバッグ補助  
シンボルテーブル復元  
視覚化 など
- マルウェア解析
- 脆弱性検査  
バッファオーバーフロー検査など



# 1. INTRODUCTION

## DATA STRUCTURE ANALYSIS: HOWARD

- Howard[17, 18]:

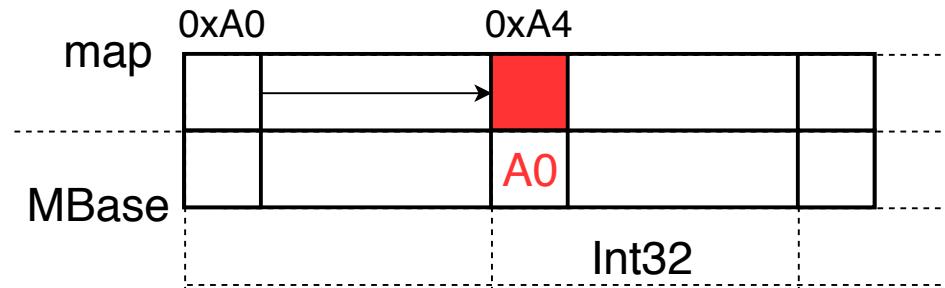
Cバイナリ対象の動的アクセスベースのデータ構造解手法

- 基本戦略: メモリ参照されるアドレスがどう導出されたかを収集

```

1  callq   malloc
2  movq    %rax , -16(%rbp )
3  ...
4  movq    -16(%rbp ) , %rcx
5  movl    $1 , 4(%rcx )

```



- ex.) オブジェクトの先頭アドレスを0xA0と仮定すると  
5行目の参照アドレス0xA4は0xA0から導出された、と捉える
- アドレスの参照関係や型情報などをメタデータ(MBase)にマッピング

- 動的なアクセス情報を集めるので高精度

[17] Howard: a Dynamic Excavator for Reverse Engineering Data Structures [NDSS'11]

[18] DDE: Dynamic data structure excavation [APSys'10]

# 1. INTRODUCTION

## DATA STRUCTURE ANALYSIS: HOWARD

- Howard[17, 18]:

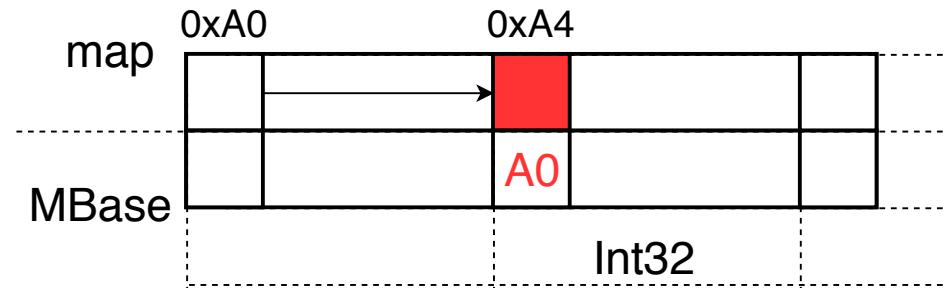
Cバイナリ対象の動的アクセスベースのデータ構造解手法

- 基本戦略: メモリ参照されるアドレスがどう導出されたかを収集

```

1  callq   malloc
2  movq    %rax , -16(%rbp )
3  ...
4  movq    -16(%rbp ) , %rcx
5  movl    $1 , 4(%rcx )

```



- ex.) オブジェクトの先頭アドレスを0xA0と仮定すると  
5行目の参照アドレス0xA4は0xA0から導出された、と捉える
- アドレスの参照関係や型情報などをメタデータ(MBase)にマッピング

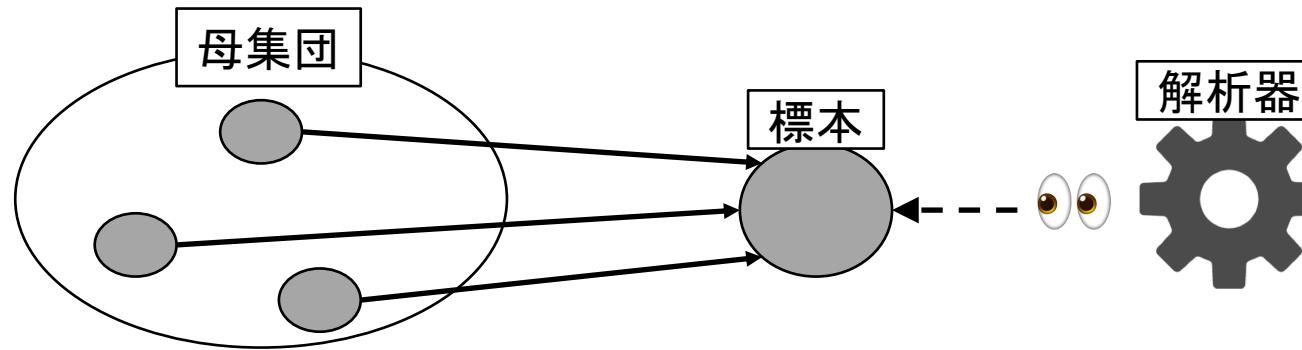
- 動的なアクセス情報を集めるので高精度

[17] Howard: a Dynamic Excavator for Reverse Engineering Data Structures [NDSS'11]

[18] DDE: Dynamic data structure excavation [APSys'10]

## 1. INTRODUCTION SAMPLING

- サンプリング(Sampling: 標本調査)
  - 母集団からの標本を抽出/解析  
→母集団全体の調査が困難/手間がかかる場合に利用
  - プログラム解析では...  
→動的オーバーヘッドの削減目的で利用される



- ただし、プログラム解析では無作為抽出が必ずしも適切ではない
  - 統計を取りたいわけではない  
→解析内容に則したサンプリング手法を選択する必要アリ

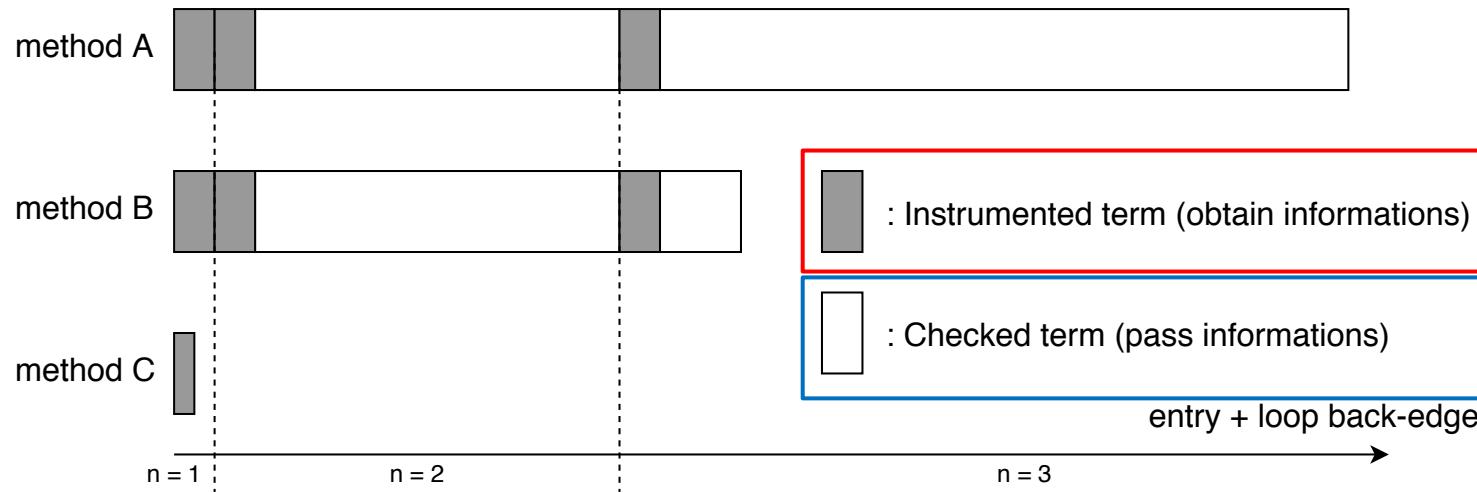
# 1. INTRODUCTION

## SAMPLING: ADAPTIVE BURSTY TRACING (ABT)

- Adaptive Bursty Tracing(ABT)[7]:

メモリアクセスに対してのサンプリング手法の一つ

- 各クラスタ（関数）の実行頻度でサンプリングレートを変える
  - 低頻度ほど高く、高頻度ほど低く
  - 「実行」はループの繰り返しも含む
  - 固定長のサンプリング期間と漸化式的な非サンプリング期間を繰り返す



## 2. MOTIVATION OVERVIEW

---

- 動的アクセスベースのデータ構造解析の問題点
  - 動的なオーバーヘッドが大きくなりがち
- 冗長なメモリアクセスが多い
  - ① サンプリング技術と組合せて軽量化したい
- サンプリングの弊害
  - 取得したい情報が失われる場合がある
  - ② 非冗長なメモリアクセス情報の回復/補完が必要
- ①, ②: 本研究の解決したい項目

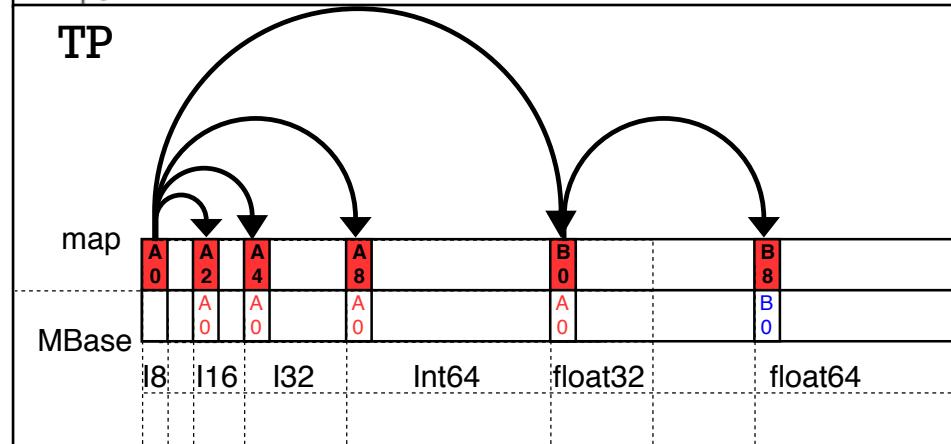
## 2. MOTIVATION

### ① REDUNDANT INFORMATION

- モチベ①  
冗長な情報は間引きたい
- ex.)  
if 各calloc直後のreinit\_TPで充分  
→以降のreinit\_TPの情報は冗長
- (ここでいう)冗長性:  
失われても解析結果に  
無影響なこと
- 間引くことでオーバーヘッド低減

```

20 int main(){
21     int i = 0;
22     while( i < 20 ){
23         TP *tp = calloc( sizeof(TP), 1 );
24         while( abs(tp->a) <= rand() ){
25             reinit_TP( tp );
26         }
27         printf("%d\n", tp->a )
28         free( tp );
29     }
30     return 0;
31 }
```



## 2. MOTIVATION

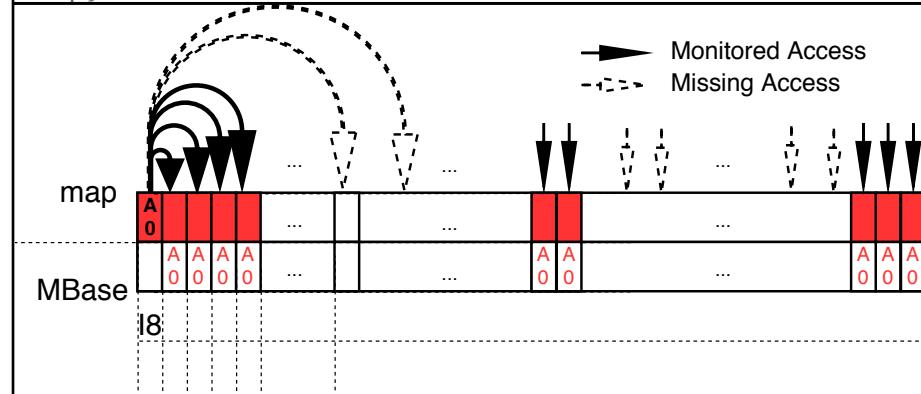
### ② NON-REDUNDANT INFORMATION

- モチベ②  
非冗長な情報は間引きたくない
- 冗長か否かを見極めるのは困難
- ABTの場合:  
サンプリング期間を超える長さの配列へのアクセスは苦手
  - 配列の一部しか分からない
- 方針
  - A: 非冗長な情報を冗長に変換
  - B: 非冗長な情報を補完する

```

1 const char *lyric =
2 "When the night has come\n\
3 And the land is dark\n\
4 And the moon is the only light we'll see\n\
5 ...
6 Oh stand by me, stand by me\n\
7 Oh stand by me, stand by me";
8
9 int main(){
10     char *p = malloc(strlen(lyric));
11     int i = 0;
12     while(i < strlen(lyric)) p[i] = lyric[i];
13
14 }
15

```



### 3. APPROACH

## HOWARD SYNERGY

---

- Howardはオブジェクトをcontextでグループ化
  - context (calling context, allocation context, ...):
    - この場合、mallocなどが呼ばれたときのコールスタックの状態と同義
  - 利点: 同一context（同属）のオブジェクトは同じ型を持つ傾向
    - 一般に同じ変数に代入されるため
    - → グループの型が分かれば属するオブジェクトの構造が全て分かる
    - オブジェクトへ冗長性の付与

```

20 int main(){
21     int i = 0;
22     while( i < 20 ){
23         TP *tp = calloc( sizeof(TP) , 1 );
24         while( abs(tp->a) <= rand() ){
25             reinit_TP( tp );
26         }
27         printf("%d\n" , tp->a )
28         free( tp );
29     }
30     return 0;
31 }
```

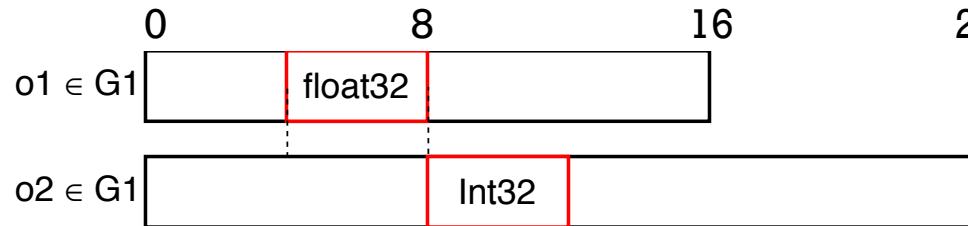
例:  
23行目でグループ化

### 3. APPROACH

## BASIS STRUCTURE

---

- 基底構造: あるContext Groupが持つ固定長の構造（型）
  - 型: ここでは配列を構成し得る要素（プリミティブ型/構造体）
  - 各ヒープオブジェクトの構造  
→ 属するグループの基底構造の繰り返しによって得ることが可能（だと仮定）
  - 基底構造のサイズは同属のオブジェクトのサイズの最大公約数以下

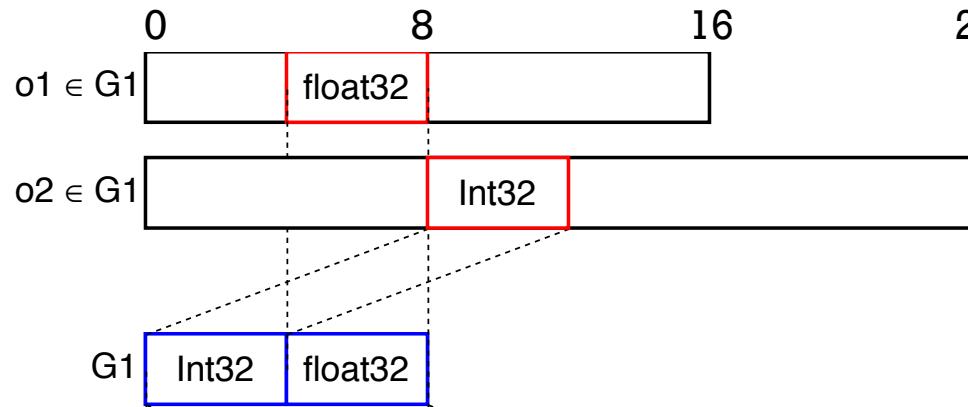


### 3. APPROACH

## BASIS STRUCTURE

---

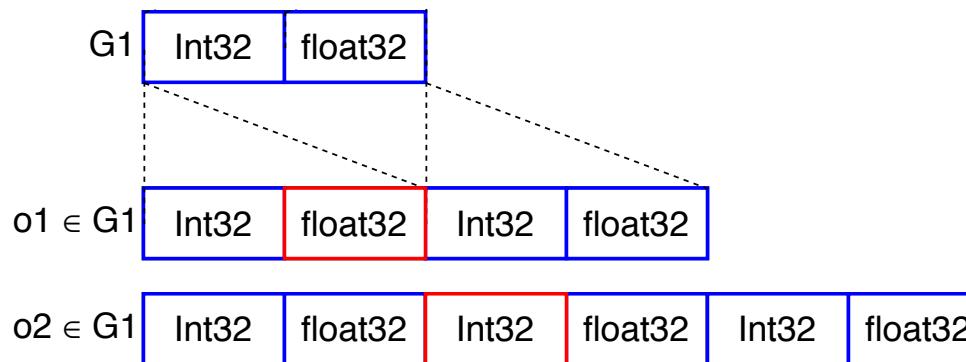
- 基底構造: あるContext Groupが持つ固定長の構造（型）
  - 型: ここでは配列を構成し得る要素（プリミティブ型/構造体）
  - 各ヒープオブジェクトの構造  
→ 属するグループの基底構造の繰り返しによって得ることが可能（だと仮定）
  - 基底構造のサイズは同属のオブジェクトのサイズの**最大公約数以下**



### 3. APPROACH

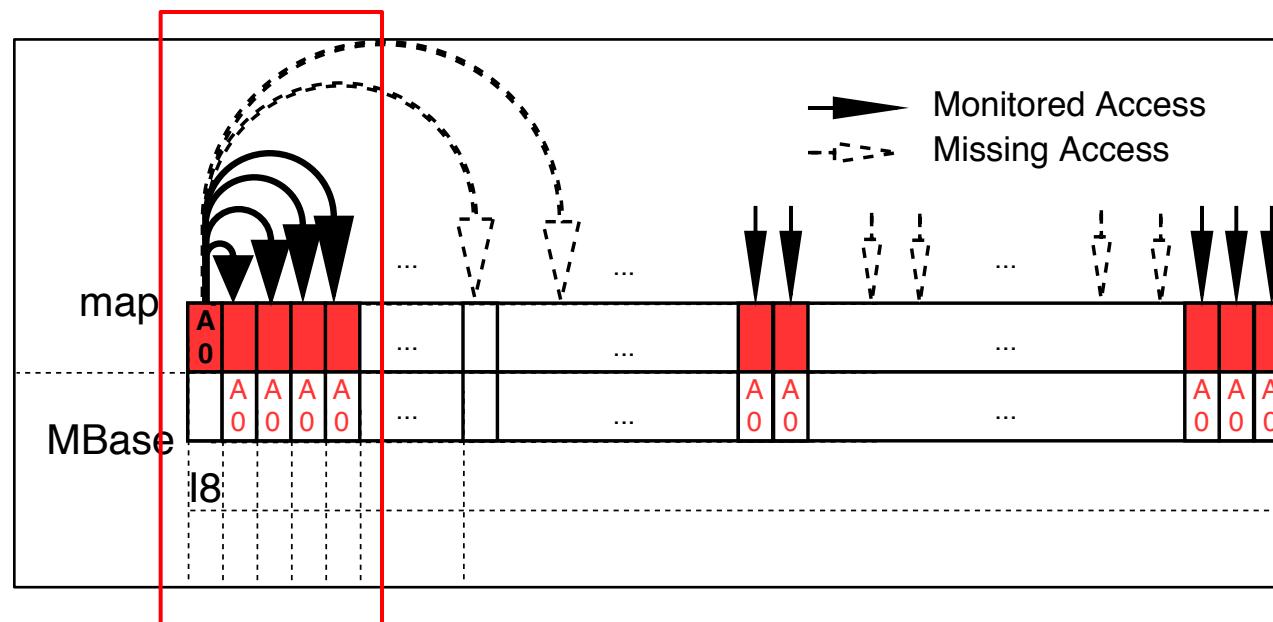
## BASIS STRUCTURE

- 基底構造: あるContext Groupが持つ固定長の構造（型）
  - 型: ここでは配列を構成し得る要素（プリミティブ型/構造体）
  - 各ヒープオブジェクトの構造  
→ 属するグループの基底構造の繰り返しによって得ることが可能  
(だと仮定)
  - 基底構造のサイズは同属のオブジェクトのサイズの最大公約数以下



### 3. APPROACH ABT SYNERGY

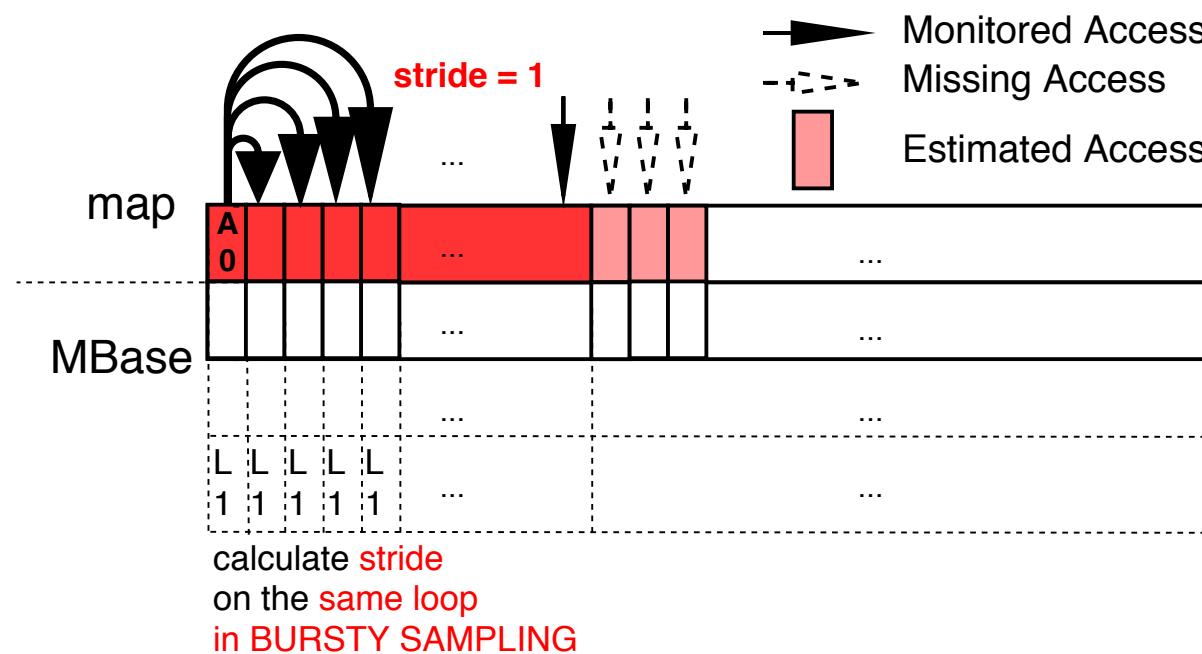
- ABTは得られるサンプルが連続的  
by 固定長サンプリング期間
  - 利点: 配列への連続的なアクセスを検知可能
    - ループの判定と組合せて、配列へのアクセスが予測可能



### 3. APPROACH

## ARRAY ESTIMATION

- 配列予測:
  - ABTでループ内の定間隔アクセスを検出する
    - 非サンプリング期間に入っても同一間隔であればアクセスとして記録



## 4. IMPLEMENTATION

---

- Intel Pin[11, 16]を用いてHowardを再現し、ABTを適用
  - オリジナルHowardからの変更点
    - 対象領域:  
{静的領域, ヒープ領域, スタックフレーム} → {ヒープ領域}
    - Context Groupへの反映タイミング:  
ヒープオブジェクト解放時 → メモリアクセス時
    - 実行前の静的解析:  
今回は未再現
  - 制約点
    - データ構造を無視するライブラリ関数(memsetなど):  
ブラックリスト処理で無視
    - コンパイラ最適化/難読化  
今回は未考慮
    - 可変長配列/共用体:  
今回は未対応

[11] Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation [PLDI'05]

[16] Pin - A Dynamic Binary Instrumentation Tool,

[https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool.](https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool)

# 5. EXPERIMENT ENVIRONMENT

	VM Host	VM Guest
OS	High Sierra 10.13.2	Ubuntu14.04 Linux 3.13.0-137-generic
Processor	Intel Core i7 7567u	Intel Core i7 7567u
Core / Freq	2 / 3.5GHz	2 / 3.5GHz
Memory	16GB	3.9GB
Storage	500GB	101.3GB
	OS Emulator	
name	VirtualBox	
version	5.1.30 r118389 (Qt5.6.3)	

表 1 実験環境

# 5. EXPERIMENT

## SMALL BENCHMARKS

program	lang	Static # of LOC/Routine	Dynamic # of				
			instructions	Call	Access (sampling on/off)	Groups	Objects
small_ex1	C	~100/1,808	7,517,404,632	21,334	5,846,086/1,549,247,755	1	10,000
small_ex2	C	~100/1,809	251,714	2,707	2,873/33,175	1	10
small_ex3	C	~100/1,809	5,336,435	99,718	7,364/1,917,882	1	100
small_ex4	C	~100/1,807	166,782	901	1,487/10,492	2	10
small_ex5	C	~150/1,815	208,941	2413	2,436/23,107	7	65

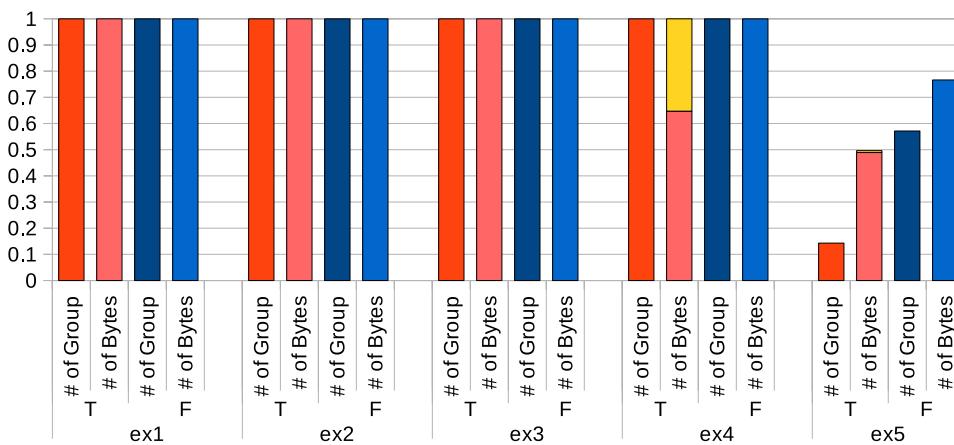
- ex1: 連結リストへの大量アクセス
  - 重い
- ex2: 構造体オブジェクトへの必要最低限のアクセス
  - 期待: グループ化で冗長性の付与
- ex3: ex2の亞種, ランダム要素数の構造体配列への最低限のアクセス
  - 期待: 基底構造の特定でも冗長性の付与
- ex4: サンプリング期間を超える配列アリ
  - 期待: 配列予測による補完
- ex5: 現実装では対応できないケース詰め合わせ

# 5. EXPERIMENT

## ACCURACY

---

- # of Group: グループの正答率（完答の割合）
- # of Bytes: グループ内の各バイトの正答率（部分点も加味）
- T: sampling 有 F: sampling 無



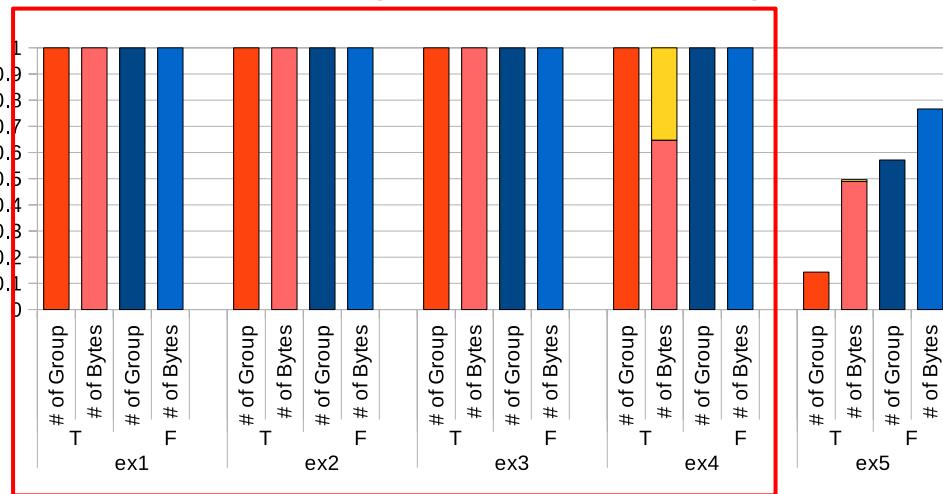
program	Sampling = True		Sampling = False	
	Groups	Bytes	Group	Bytes
small_ex1	1/1	24/24	1/1	24/24
small_ex2	1/1	40/40	1/1	40/40
small_ex3	1/1	40/40	1/1	40/40
small_ex4	2/2	(33+18)/51	2/2	51/51
small_ex5	1*/7	(67*+1)/137	4/7	105/137

表 3 精度実験: スコア

# 5. EXPERIMENT

## ACCURACY

- # of Group: グループの正答率（完答の割合）
- # of Bytes: グループ内の各バイトの正答率（部分点も加味）
- T: sampling 有 F: sampling 無



program	Sampling = True		Sampling = False	
	Groups	Bytes	Group	Bytes
small_ex1	1/1	24/24	1/1	24/24
small_ex2	1/1	40/40	1/1	40/40
small_ex3	1/1	40/40	1/1	40/40
small_ex4	2/2	(33+18)/51	2/2	51/51
small_ex5	1*/7	(67*+1)/137	4/7	105/137

表 3 精度実験: スコア

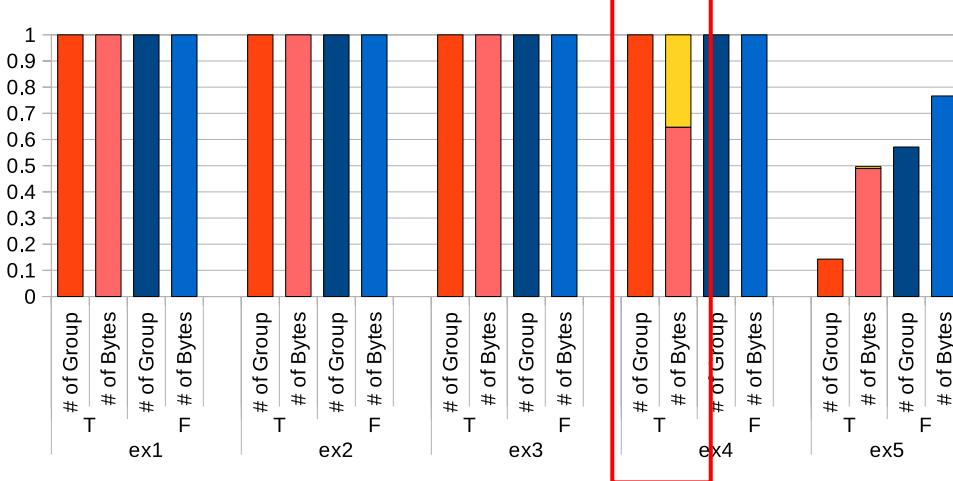
- 総じて精度を保つことに成功
- 
-

# 5. EXPERIMENT

## ACCURACY

---

- # of Group: グループの正答率（完答の割合）
- # of Bytes: グループ内の各バイトの正答率（部分点も加味）
- T: sampling 有 F: sampling 無



- 総じて精度を保つことに成功
  - ex4では配列へのアクセスも補完
  - …

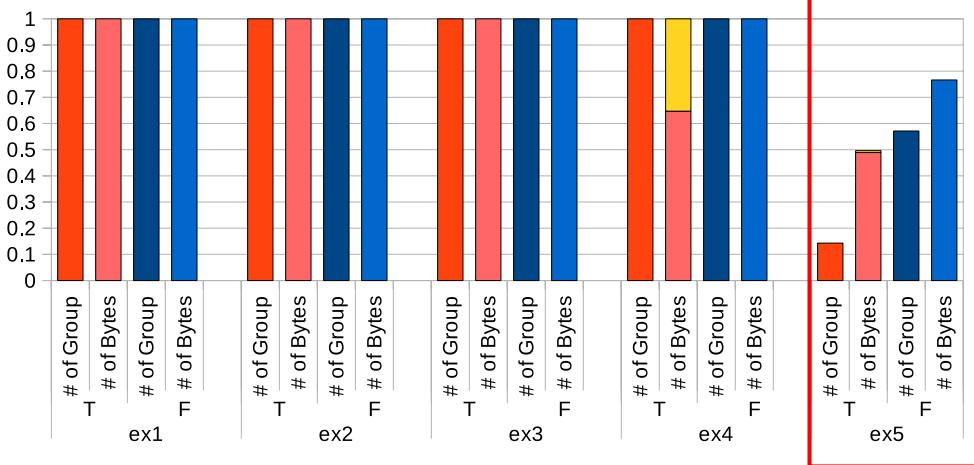
program	Sampling = True		Sampling = False	
	Groups	Bytes	Group	Bytes
small_ex1	1/1	24/24	1/1	24/24
small_ex2	1/1	40/40	1/1	40/40
small_ex3	1/1	40/40	1/1	40/40
small_ex4	2/2	(33+18)/51	2/2	51/51
small_ex5	1*/7	(67*+1)/137	4/7	105/137

表 3 精度実験: スコア

# 5. EXPERIMENT

## ACCURACY

- # of Group: グループの正答率（完答の割合）
- # of Bytes: グループ内の各バイトの正答率（部分点も加味）
- T: sampling 有 F: sampling 無



program	Sampling = True		Sampling = False	
	Groups	Bytes	Group	Bytes
small_ex1	1/1	24/24	1/1	24/24
small_ex2	1/1	40/40	1/1	40/40
small_ex3	1/1	40/40	1/1	40/40
small_ex4	2/2	(33+18)/51	2/2	51/51
small_ex5	1*/7	(67*+1)/137	4/7	105/137

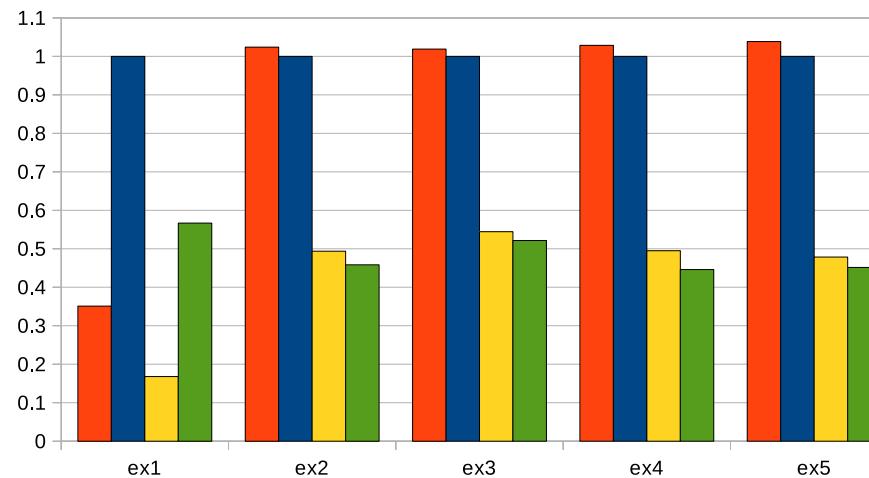
表 3 精度実験: スコア

- 総じて精度を保つことに成功
  - ex4では配列へのアクセスも補完
  - ex5では多次元配列などはまだ補完できないことが示された

## 5. EXPERIMENT

### DYNAMIC OVERHEAD

- 各グラフ サンプリング有 サンプリング無
- サンプリング無の解析時間を1として正規化



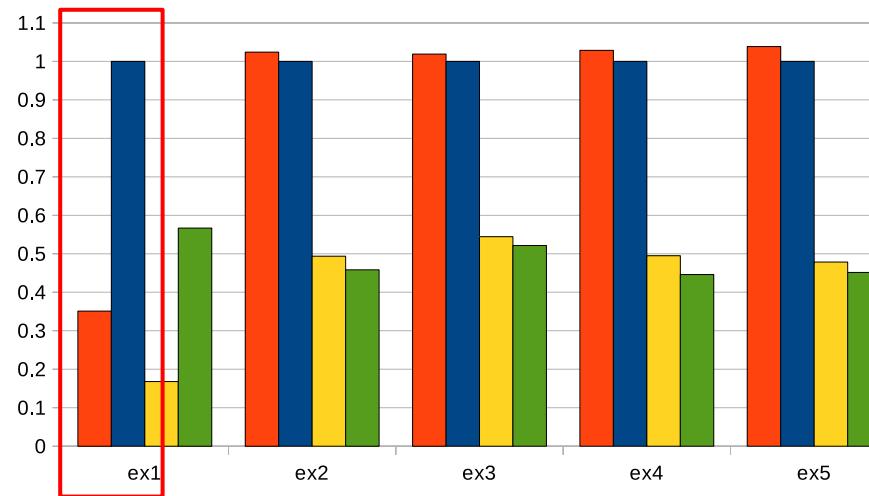
■

■

## 5. EXPERIMENT

### DYNAMIC OVERHEAD

- 各グラフ サンプリング有 サンプリング無
- サンプリング無の解析時間を1として正規化

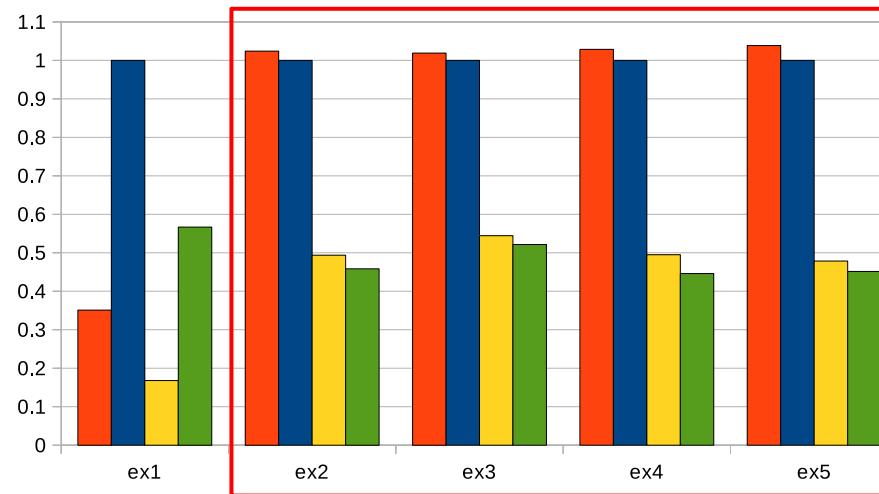


- メモリアクセスが多い検体 (ex1)  
→ 解析時間が約1/3程度に減少
-

## 5. EXPERIMENT

### DYNAMIC OVERHEAD

- 各グラフ サンプリング有 サンプリング無
- サンプリング無の解析時間を1として正規化



- メモリアクセスが多い検体 (ex1)  
→ 解析時間が約1/3程度に減少
- メモリアクセスが少ない検体 (ex2~5)  
→ サンプリング変数の更新オーバーヘッドが未償却

## 7. CONCLUSION & FUTURE WORK

---

- まとめ
  - Howardを再現実装し、ABTを適用
  - サンプリングがデータ構造解析に与える影響を調査
  - サンプリングによる精度低下を鑑みた対処作の例示
  - 実験にて精度を保ちつつオーバーヘッド低減が可能であることを確認
- 展望
  - 性能向上案
    - 各種Howardの未実装ヒューリスティクスによる精度向上
    - 実行前静的解析によるオーバーヘッドの低減
    - ABTにContext-Sensitivity付与をし、サンプリングの偏りを解消
    - など
  - 研究の発展により、より軽量で高精度なデータ構造解析技術に期待

## 参考文献

---

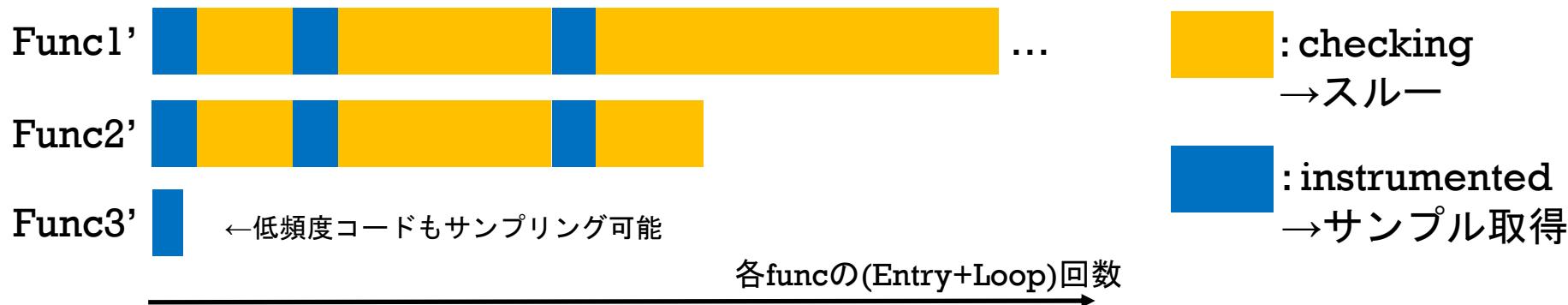
- [7] Low-overhead Memory Leak Detection Using Adaptive Statistical Profiling [ASPLOS'04]
- [11] Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation [PLDI'05]
- [16] Pin - A Dynamic Binary Instrumentation Tool,  
<https://software.intel.com/en-us/articles/pin- a-dynamic-binary-instrumentation-tool>.
- [17] Howard: a Dynamic Excavator for Reverse Engineering Data Structures [NDSS'11]
- [18] DDE: Dynamic data structure excavation [APSys'10]

# APPEND SLIDES

---

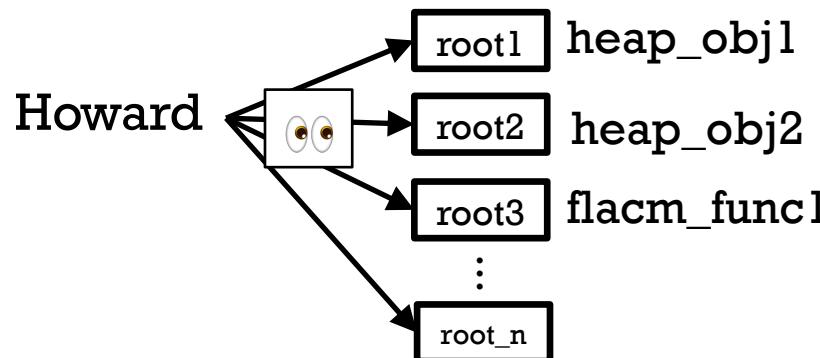
# 研究テーマ背景: サンプリング

- サンプリング: (統計学とかで出てくるアレ)
  - 対象となる全事象から一部分を抽出、解析すること
    - 対象: 今回の例ではメモリアクセスなど
  - サンプリング手法は色々
    - SWAT[3]のAdaptive Bursty Tracing(ABT: 下図)を基準に適用を画策
    - ABT: 関数単位でEntryとLoopBackEdgeの累計をカウント  
固定長のサンプル取得期間と漸化式長のスル一期間を繰り返す
      - 利点: 高頻度関数は低レート、低頻度関数は高レートで抽出可能



## 3.2: 提案手法(POINTER TRACKING) OVERVIEW

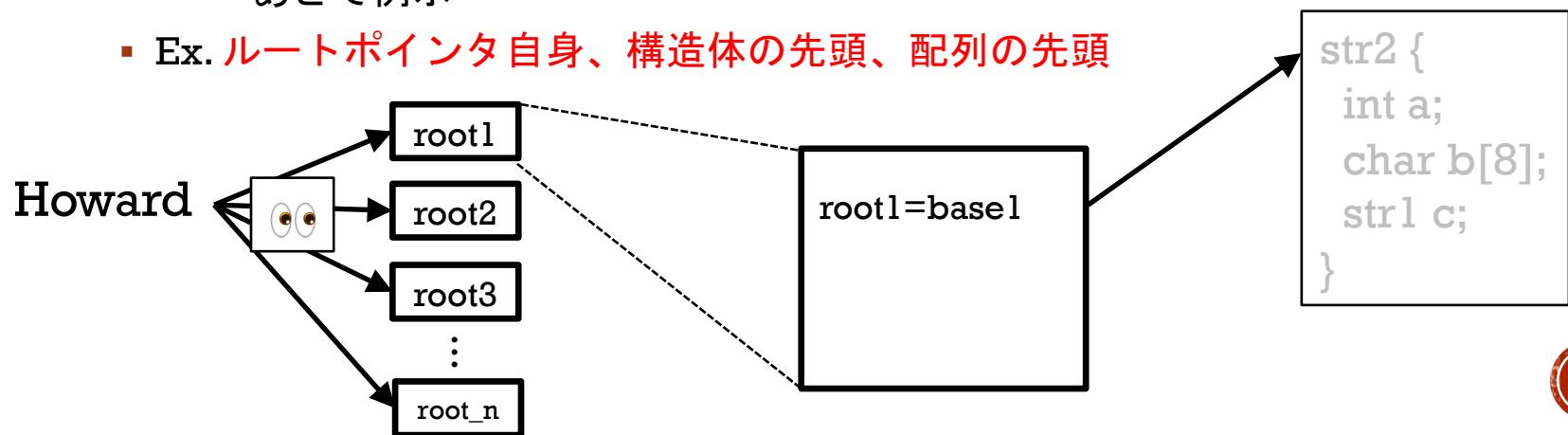
- Pointer Tracking
- 用語: ルートポインタ
  - 他のポインタから導出できないポインタ(アドレス)
    - i.e. ルートポインタが異なる  $\Rightarrow$  監視対象が異なる
    - Ex. 静的/グローバル変数、ヒープオブジェクト、各関数フレーム
- 用語: ベースポインタ
  - 共通のルートポインタ内の区割りを示すポインタ(アドレス)
    - i.e. ベースポインタが異なる  $\Rightarrow$  区割りにおける帰属が異なる
    - →あとで例示
  - Ex. ルートポインタ自身、構造体の先頭、配列の先頭



# 3.2: 提案手法(POINTER TRACKING) OVERVIEW

---

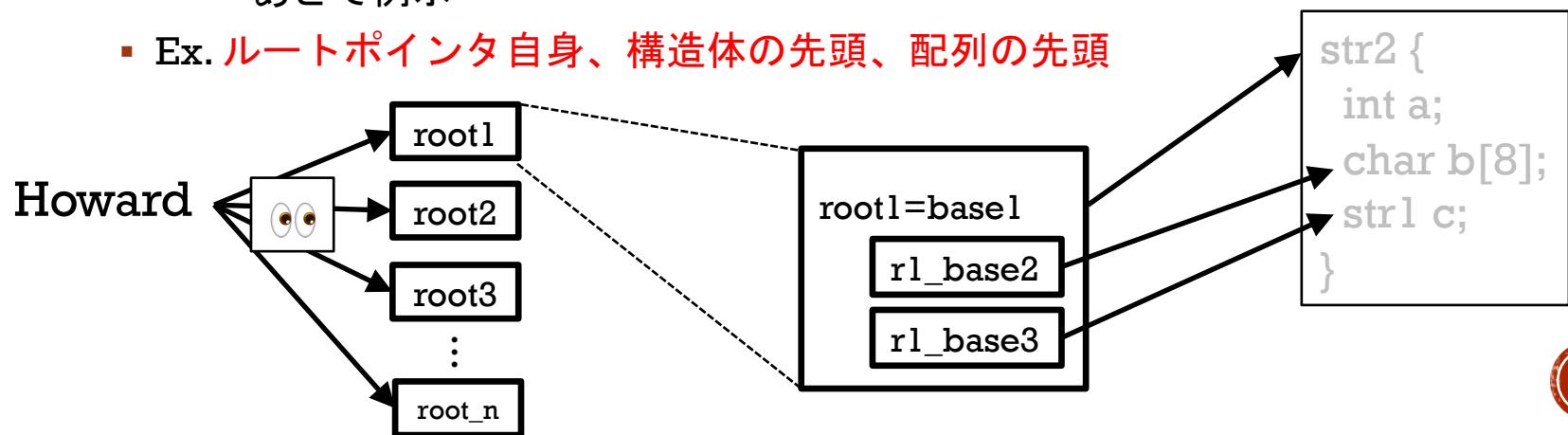
- Pointer Tracking
- 用語: ルートポインタ
  - 他のポインタから導出できないポインタ(アドレス)
    - i.e. ルートポインタが異なる  $\Leftrightarrow$  監視対象が異なる
    - Ex. 静的/グローバル変数、ヒープオブジェクト、各関数フレーム
- 用語: ベースポインタ
  - 共通のルートポインタ内の区割りを示すポインタ(アドレス)
    - i.e. ベースポインタが異なる  $\Leftrightarrow$  区割りにおける帰属が異なる
      - →あとで例示
  - Ex. ルートポインタ自身、構造体の先頭、配列の先頭



# 3.2: 提案手法(POINTER TRACKING) OVERVIEW

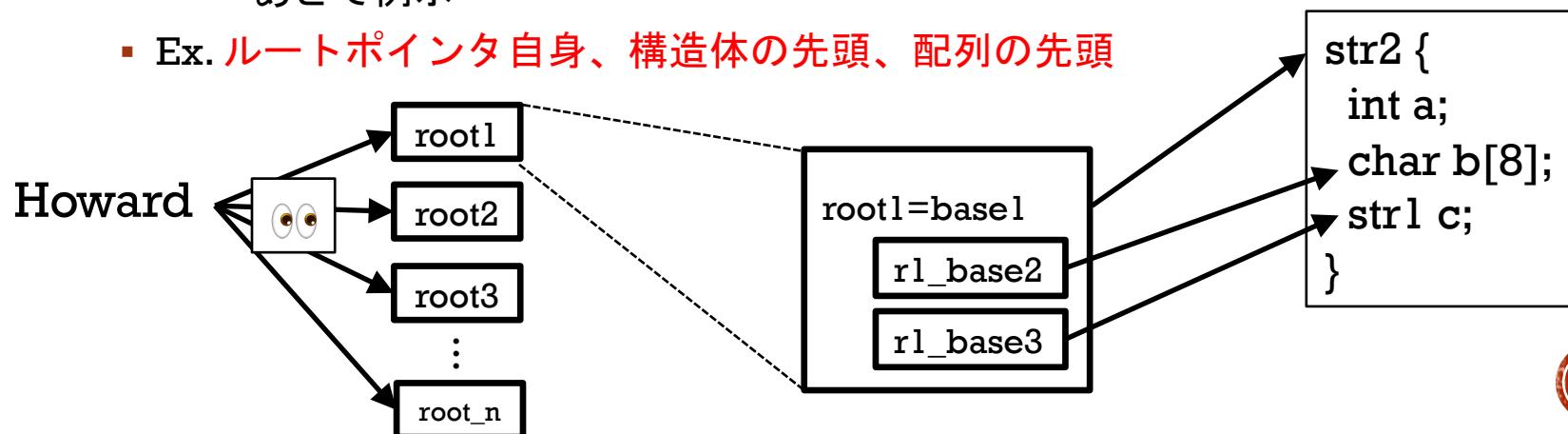
---

- Pointer Tracking
- 用語: ルートポインタ
  - 他のポインタから導出できないポインタ(アドレス)
    - i.e. ルートポインタが異なる  $\Leftrightarrow$  監視対象が異なる
    - Ex. 静的/グローバル変数、ヒープオブジェクト、各関数フレーム
- 用語: ベースポインタ
  - 共通のルートポインタ内の区割りを示すポインタ(アドレス)
    - i.e. ベースポインタが異なる  $\Leftrightarrow$  区割りにおける帰属が異なる
    - →あとで例示
  - Ex. ルートポインタ自身、構造体の先頭、配列の先頭



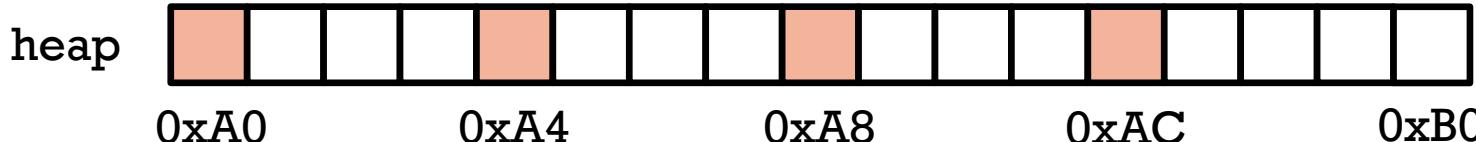
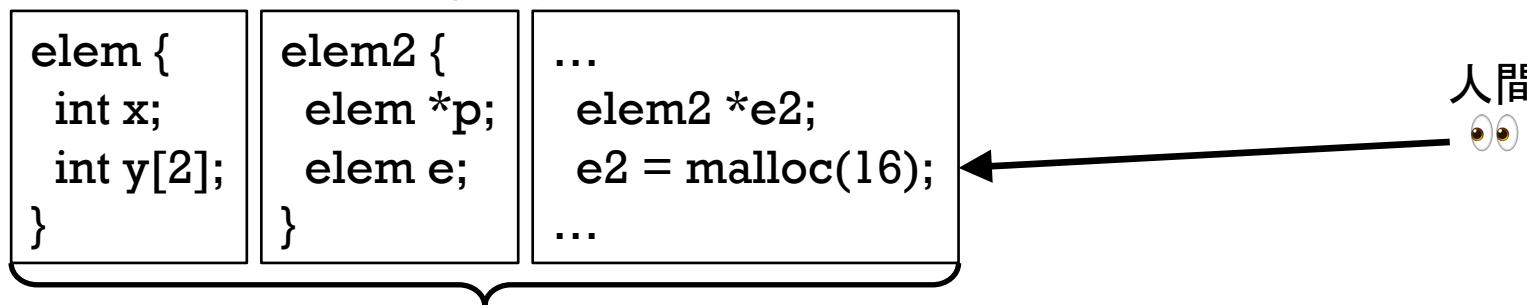
# 3.2: 提案手法(POINTER TRACKING) OVERVIEW

- Pointer Tracking
- 用語: ルートポインタ
  - 他のポインタから導出できないポインタ(アドレス)
    - i.e. ルートポインタが異なる  $\Leftrightarrow$  監視対象が異なる
    - Ex. 静的/グローバル変数、ヒープオブジェクト、各関数フレーム
- 用語: ベースポインタ
  - 共通のルートポインタ内の区割りを示すポインタ(アドレス)
    - i.e. ベースポインタが異なる  $\Leftrightarrow$  区割りにおける帰属が異なる
      - →あとで例示
  - Ex. ルートポインタ自身、構造体の先頭、配列の先頭



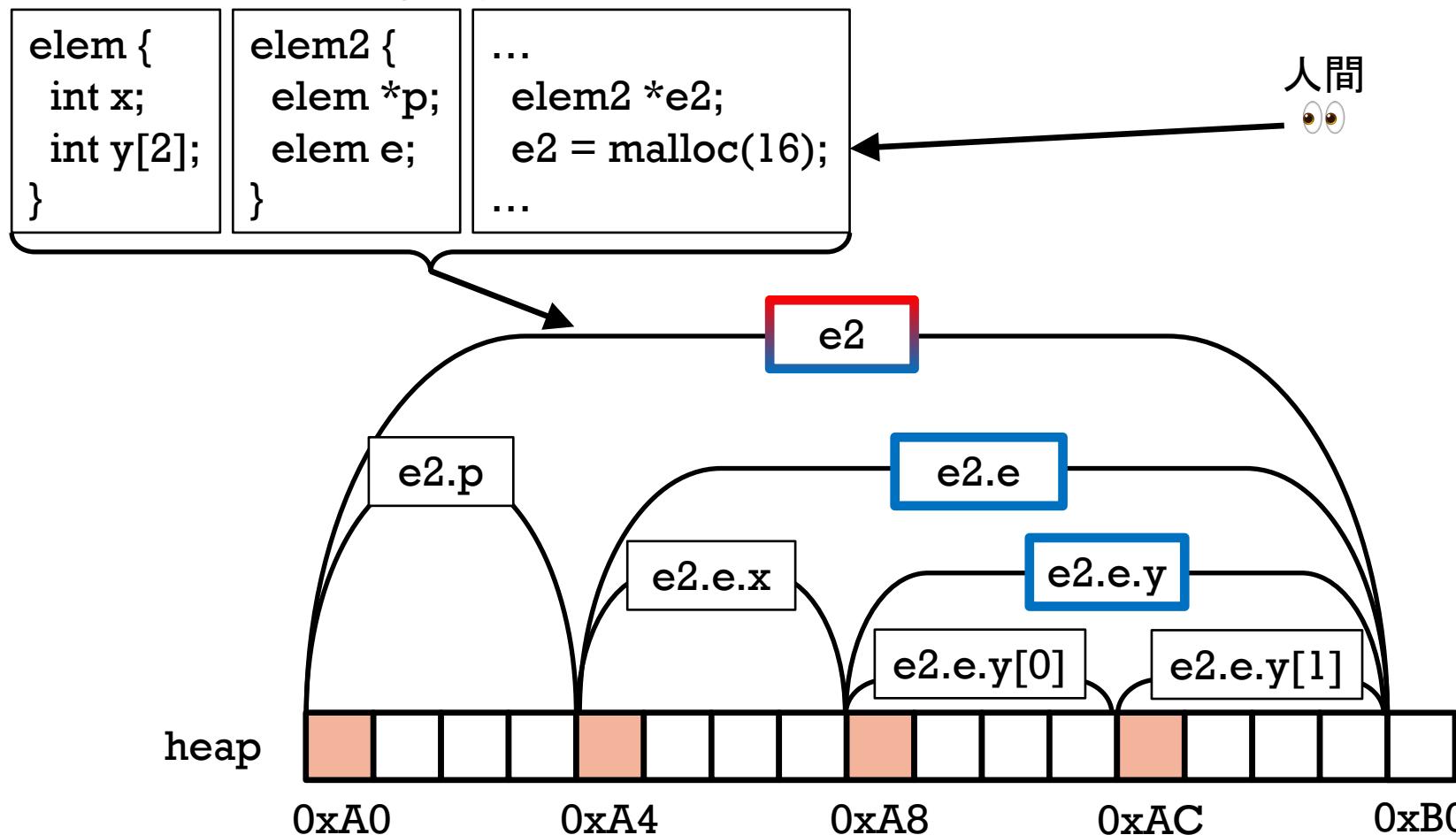
## 3.2: 提案手法(POINTER TRACKING) OVERVIEW

- ベースポインタ間には木構造のような親子関係が存在
  - 全ノードは親を辿ればルートポインタへ収束



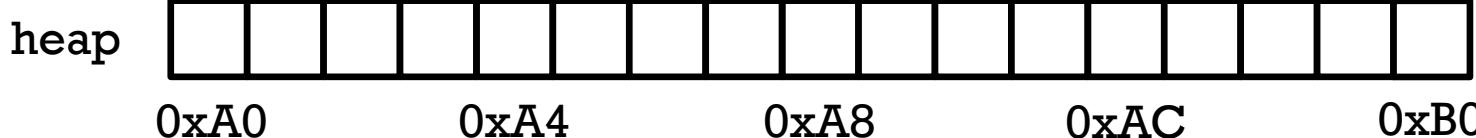
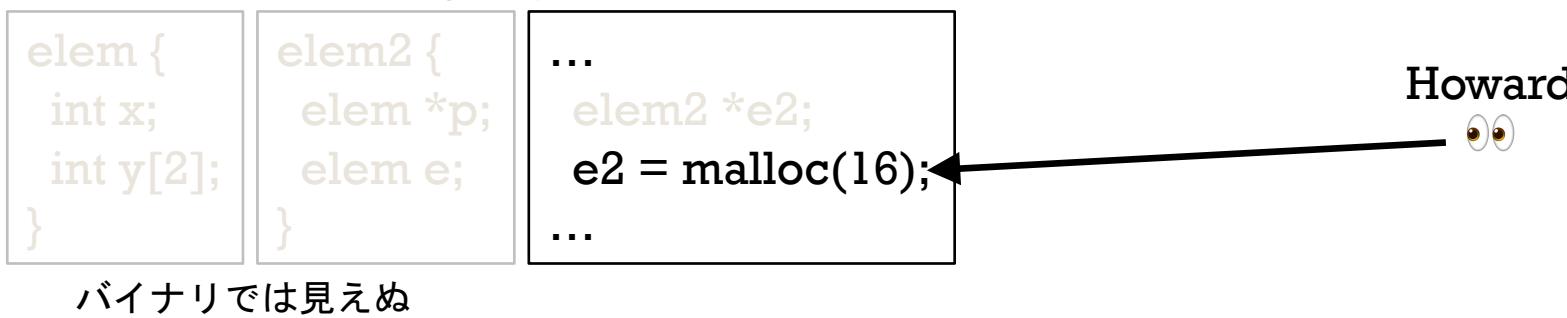
## 3.2: 提案手法(POINTER TRACKING) OVERVIEW

- ベースポインタ間には木構造のような親子関係が存在
  - 全ノードは親を辿ればルートポインタへ収束



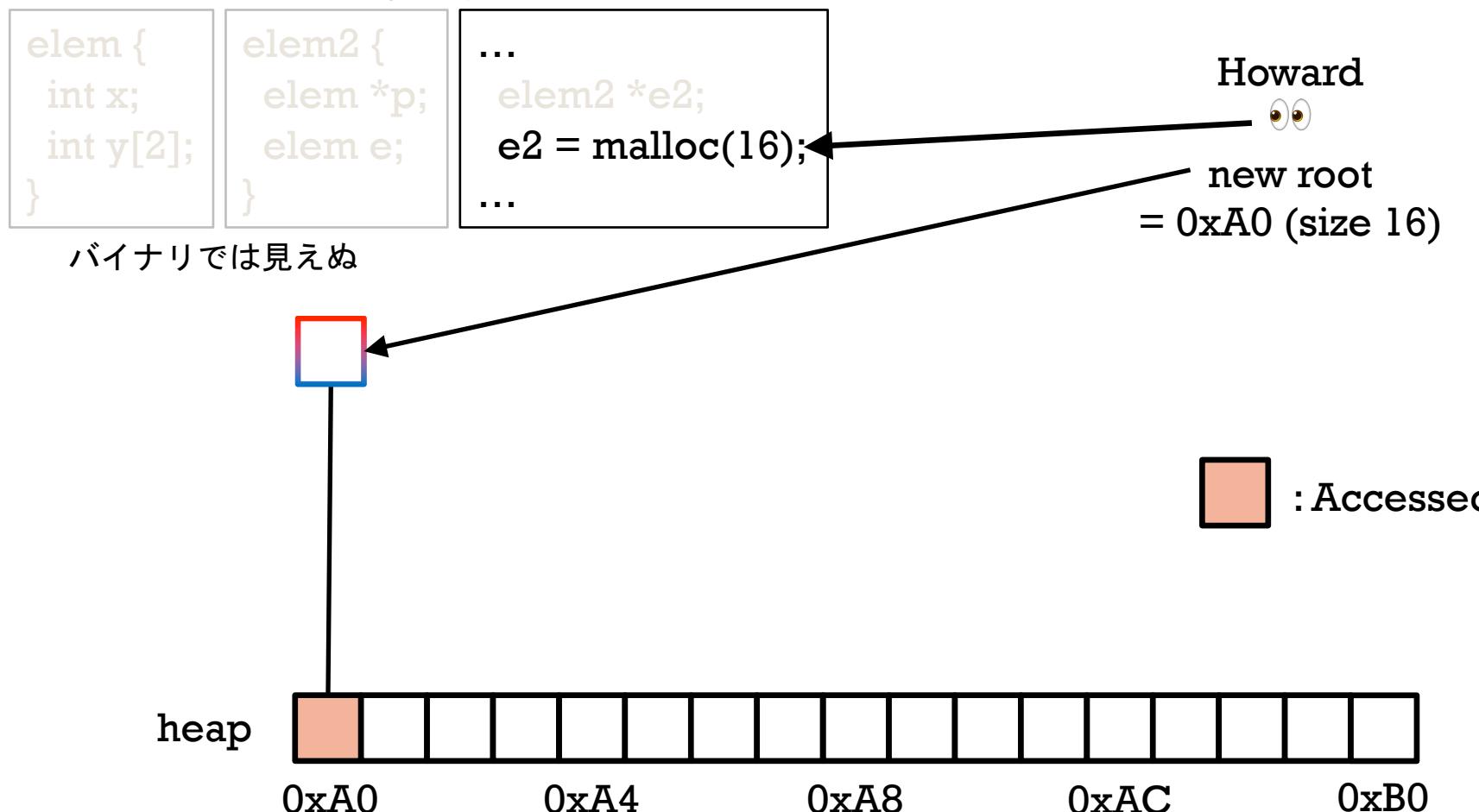
## 3.2: 提案手法(POINTER TRACKING) OVERVIEW

- ベースポインタ間には木構造のような親子関係が存在
  - 全ノードは親を辿ればルートポインタへ収束



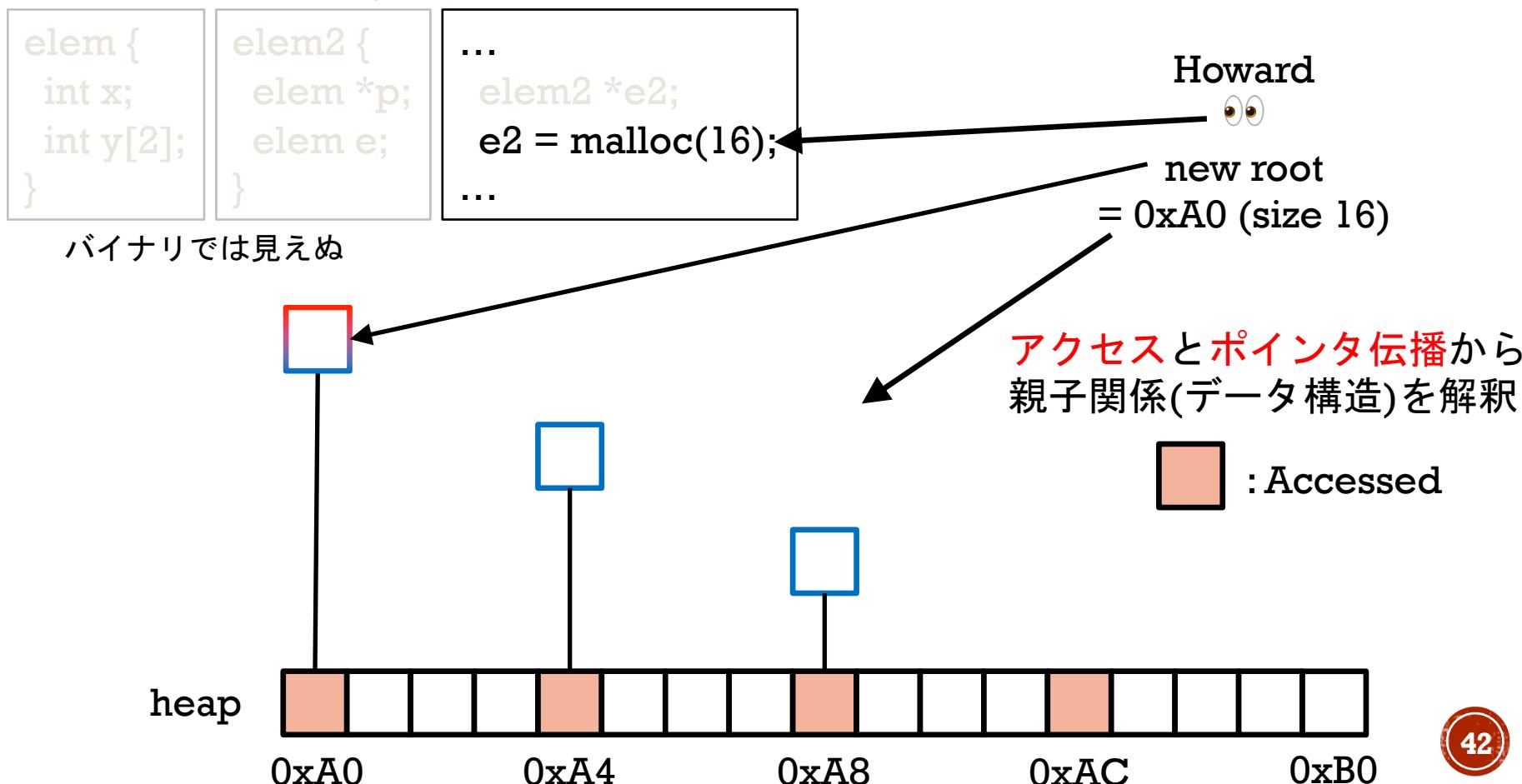
## 3.2: 提案手法(POINTER TRACKING) OVERVIEW

- ベースポインタ間には木構造のような親子関係が存在
  - 全ノードは親を辿ればルートポインタへ収束



## 3.2: 提案手法(POINTER TRACKING) OVERVIEW

- ベースポインタ間には木構造のような親子関係が存在
  - 全ノードは親を辿ればルートポインタへ収束



## 3.2: 提案手法(POINTER TRACKING) OVERVIEW

- ベースポインタ間には木構造のような親子関係が存在
  - 全ノードは親を辿ればルートポインタへ収束

