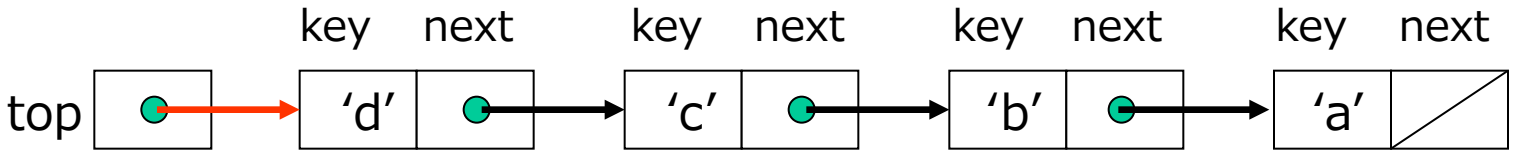
第8週
リストを用いたスタックとキュー

必須課題8-1：push （1/3）



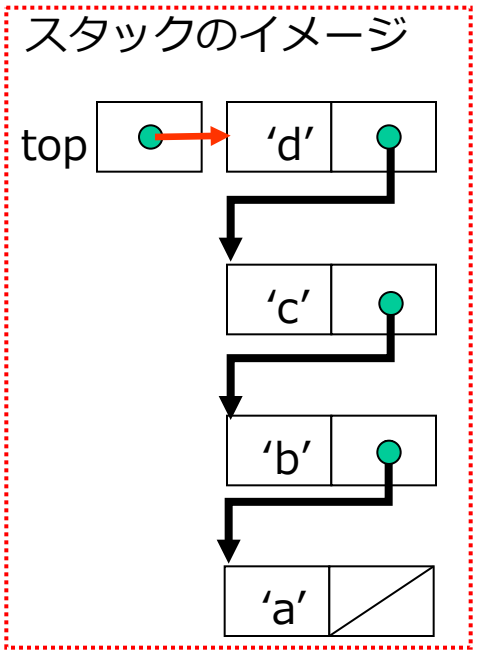
void push (struct data **top, char key);

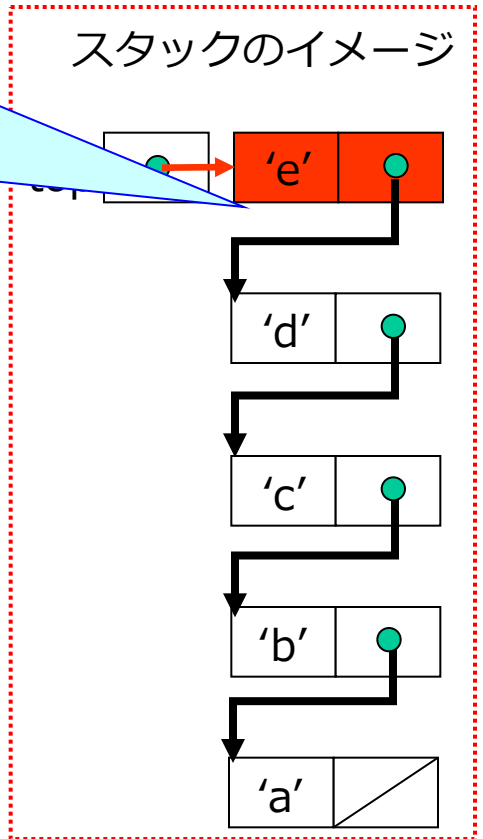
ポインタのポインタ
（あとで説明）

追加される文字

push（要素の追加）
スタックの場合は、
リストの**先頭**に新しい要素
を追加する

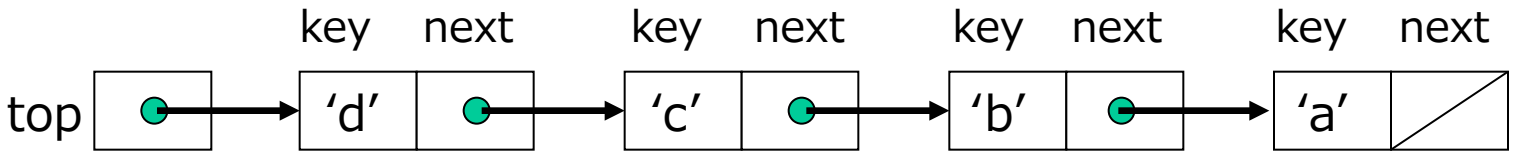
push ‘e’

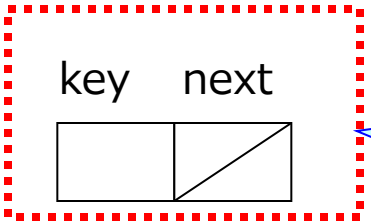




必須課題8-1：push （2/3）

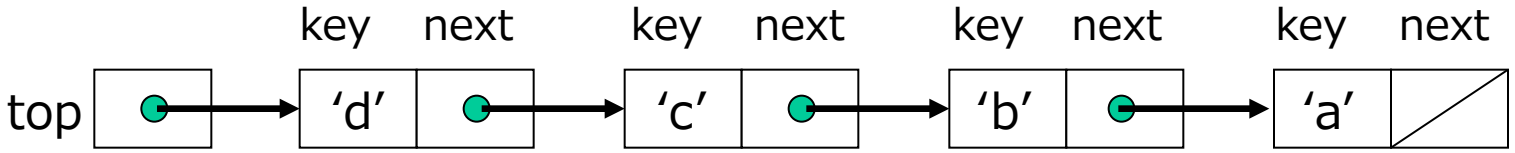
step1:新しいリストの要素のメモリを確保

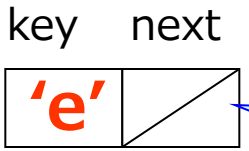




動的メモリ確保（malloc）で
メモリを確保する

step2:新しいリストの要素の初期化（値を設定）

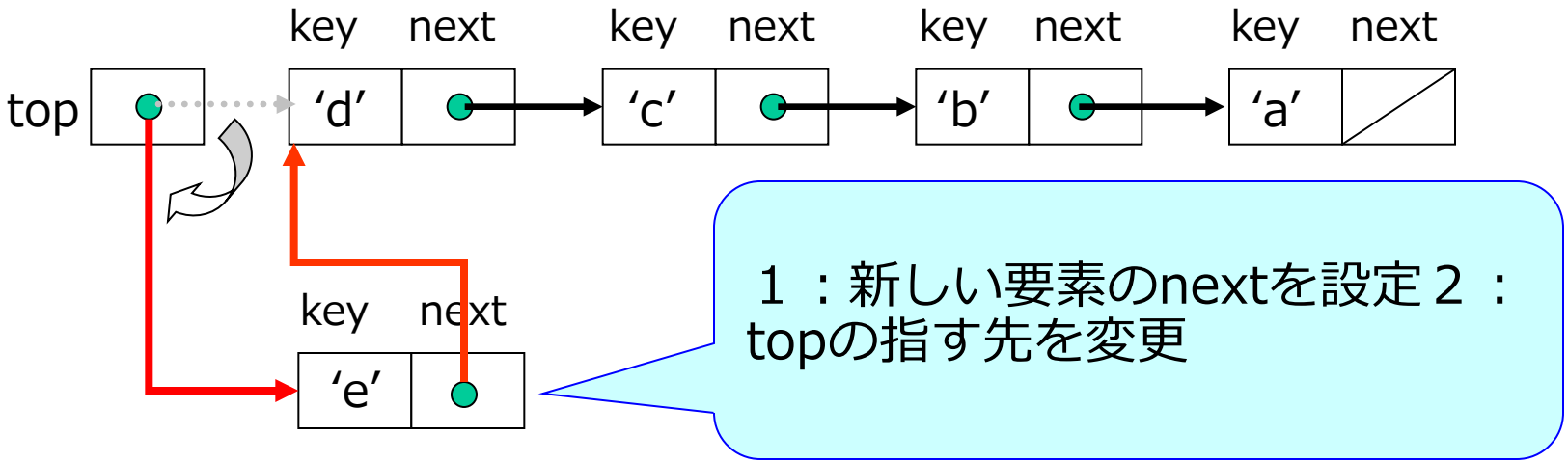




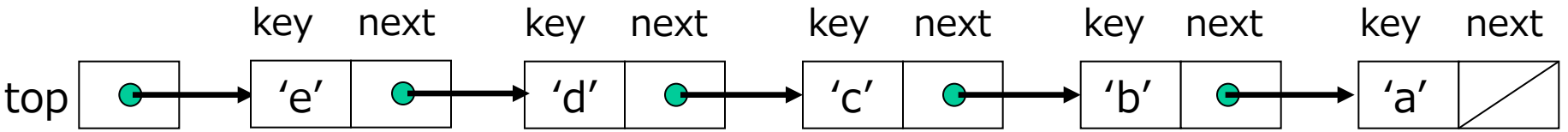
リストの要素の値を設定する

必須課題8-1：push （3/3）

step3:リストを繋ぎ直す

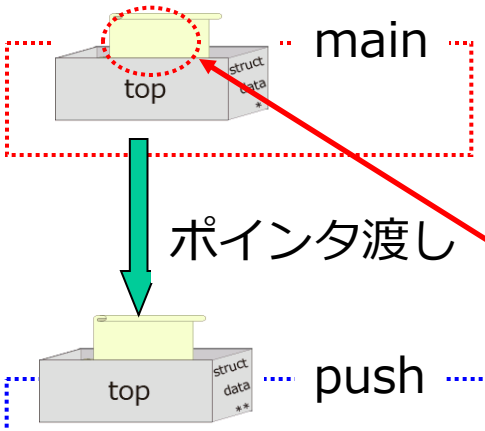


push後のリスト



ポインタのポインタ （1/2）

void push (struct data **top, char key);



ポインタのポインタ

関数内でポインタ変数の値(アドレス)を変更するため等で利用

例：前ページのstep3のtopの指している先を変更したい場合に利用

push内で**ここの値**を操作したい

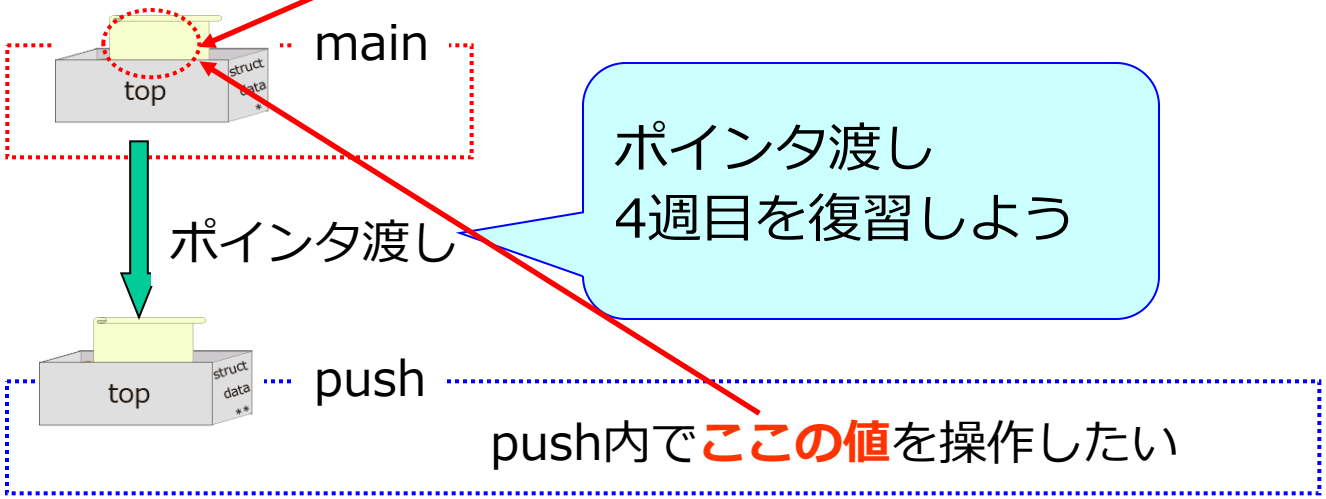
```
int main(){
  struct data *top;
  /*リストの初期化省略*/
  push( &top, 'a');
  return 0;
}
```

ポインタ変数に**&**をつけるとポインタ渡し（ポインタのポインタ）ができる

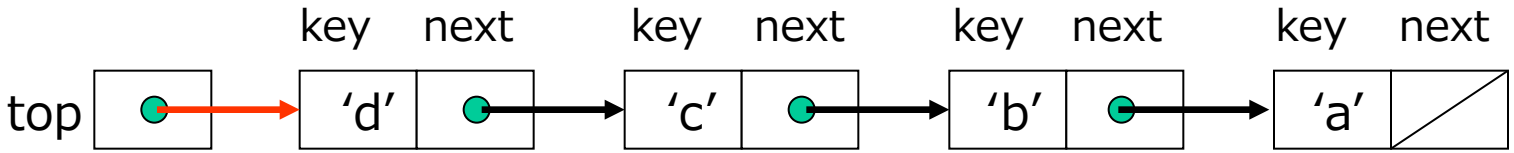
ポインタのポインタ　(2/2)

```
void push (struct data **top, char key){
    ...
}
```

通常のポインタ変数と同じ使い方ができる
＊をつけるとポインタの指している先の値を参照できる
この場合は＊topでmainで宣言したtopの値（アドレス）を操作できる



必須課題8-2：pop （1/3）

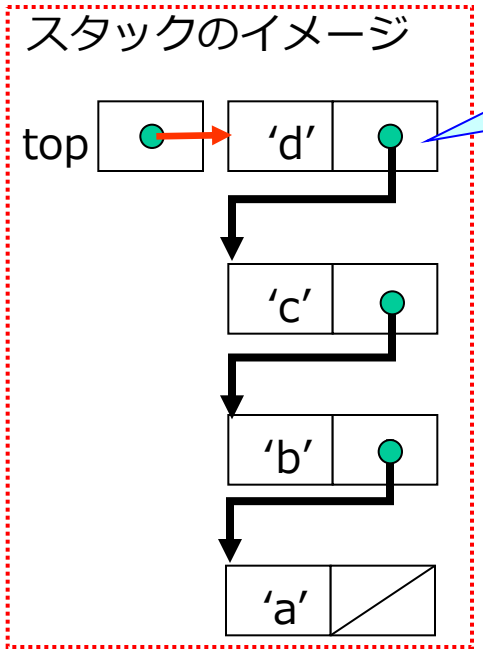



```
char pop (struct data **top);
```

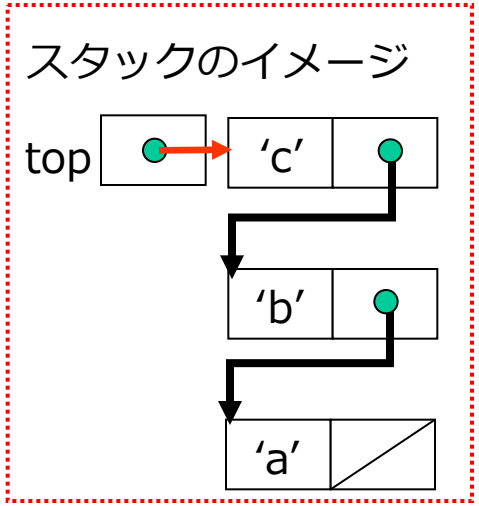
取り出したリストの要素の値（key）

ポインタのポインタ（すでに説明）

pop（要素の取り出し）
スタックの場合は、
リストの**先頭**の要素
を取り出す

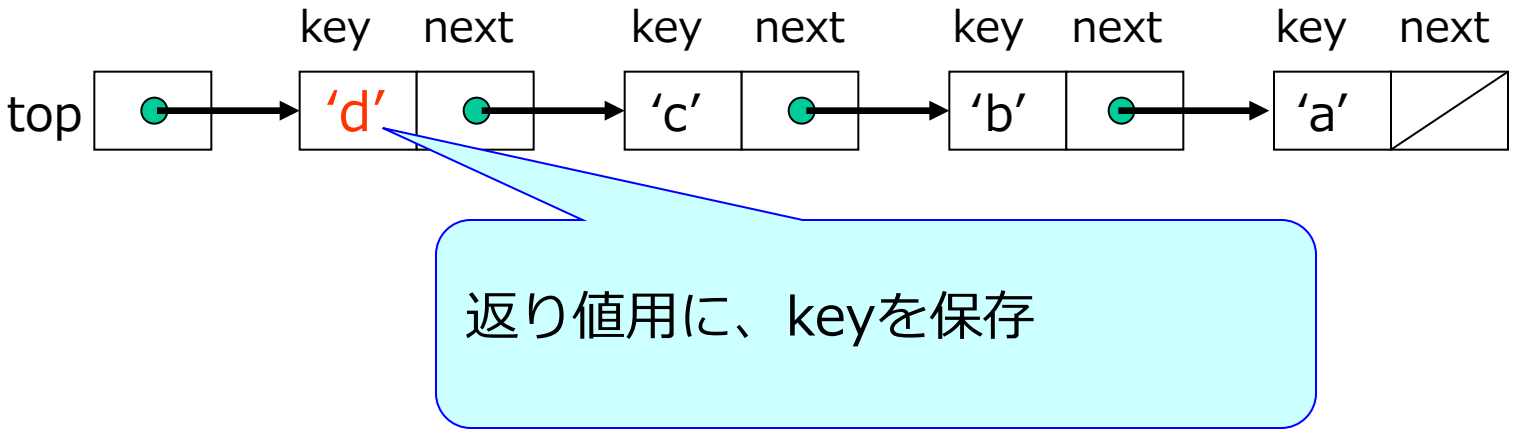


pop

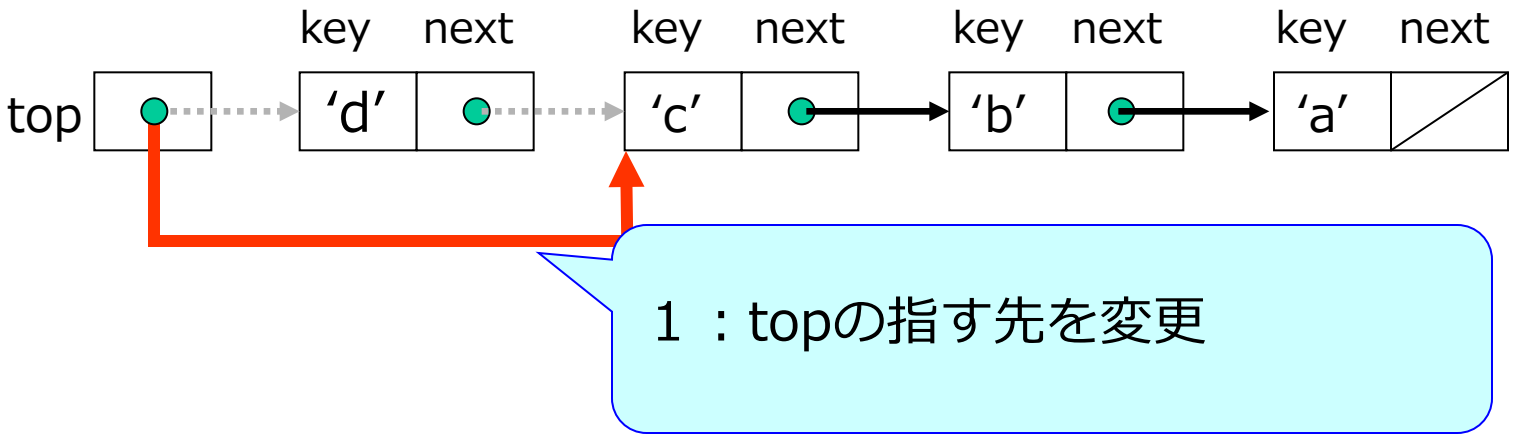


必須課題8-2：pop （2/3）

step1:先頭要素の値(key)を保存

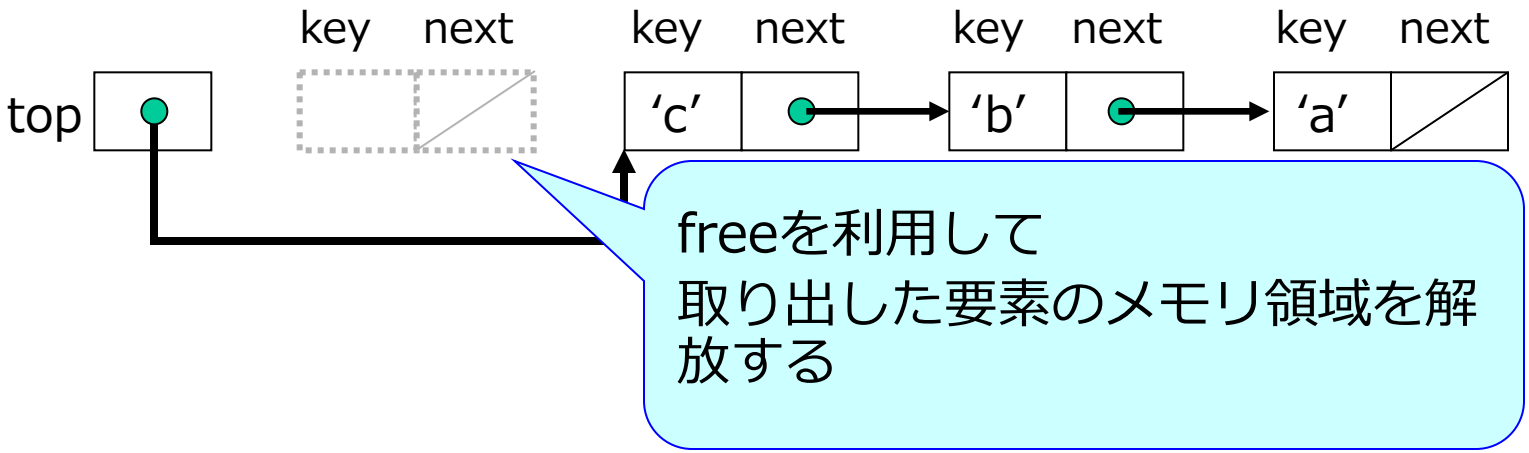


step2:リストを繋ぎ直す

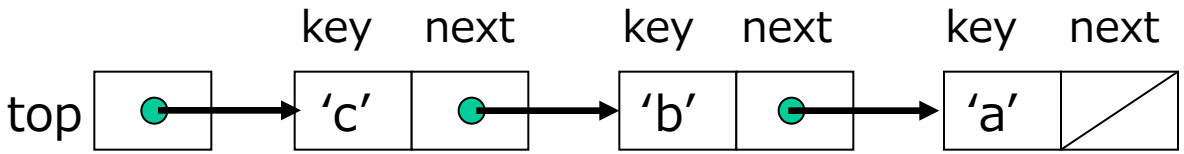


必須課題8-2：pop （2/3）

step3:取り出した要素のメモリを解放する



step4:保存した値（key）を返り値として返す
pop後のリスト

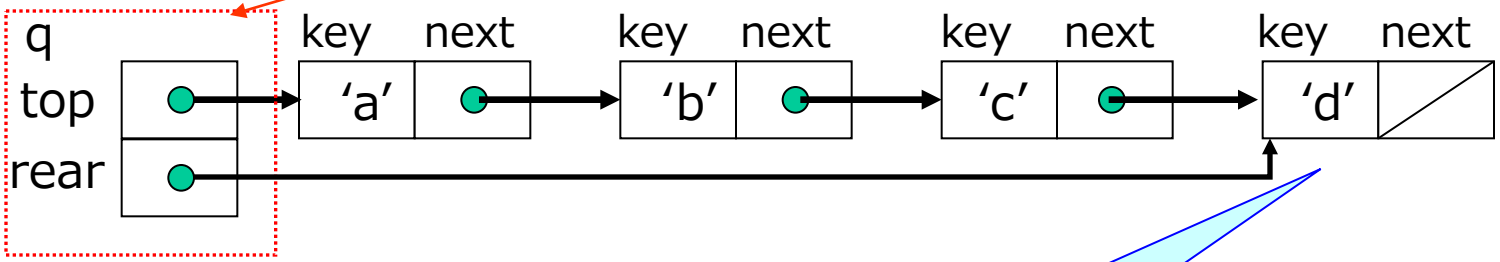


必須課題8-3：enqueue （1/3）

void enqueue (struct queue *q, char key);

構造体queueへのポインタ

追加される文字

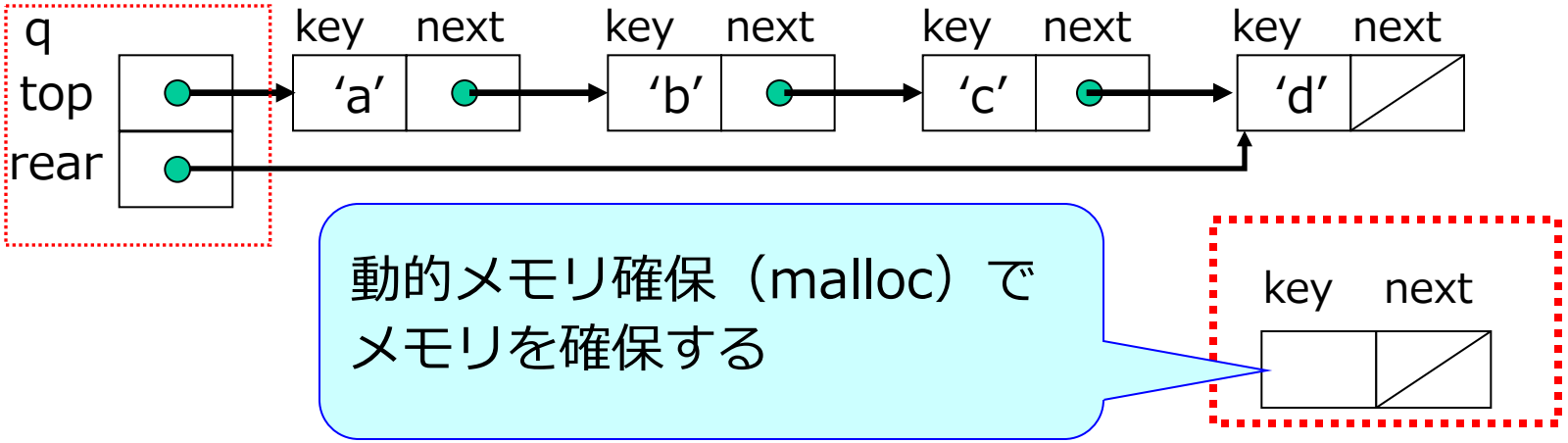


enqueue（要素の追加）

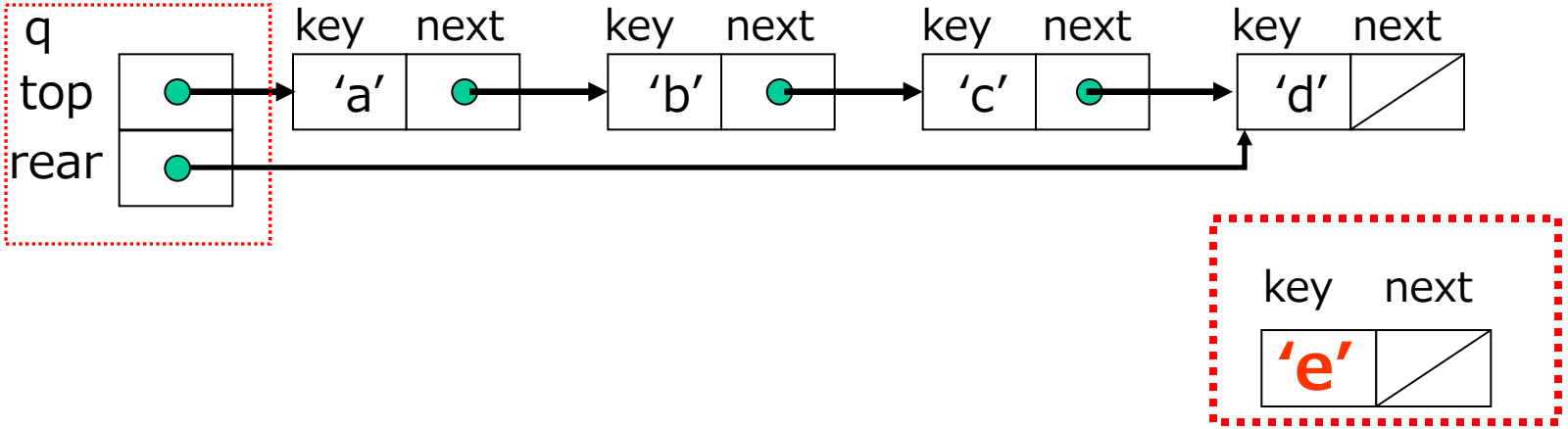
キューの場合は、
リストの**末尾**に新しい要素を追加する

必須課題8-3：enqueue （2/3）

step1:新しいリストの要素のメモリを確保

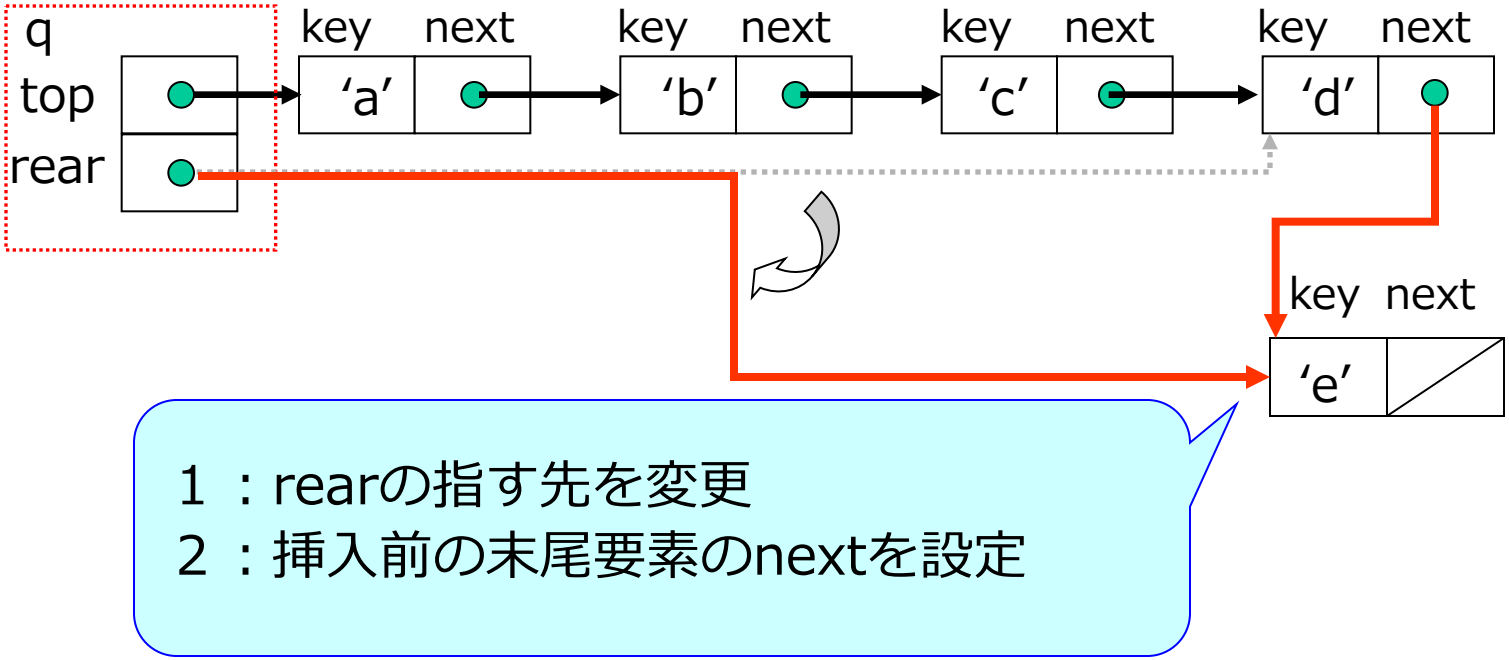


step2:新しいリストの要素の初期化（値を設定）



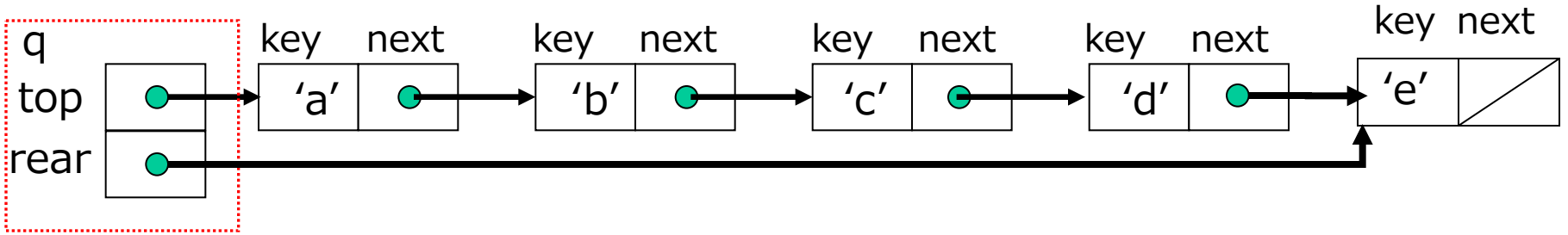
必須課題8-3：enqueue （3/3）

step3:リストを繋ぎ直す



1 ：rearの指す先を変更
2 ：挿入前の末尾要素のnextを設定

enqueue後のリスト

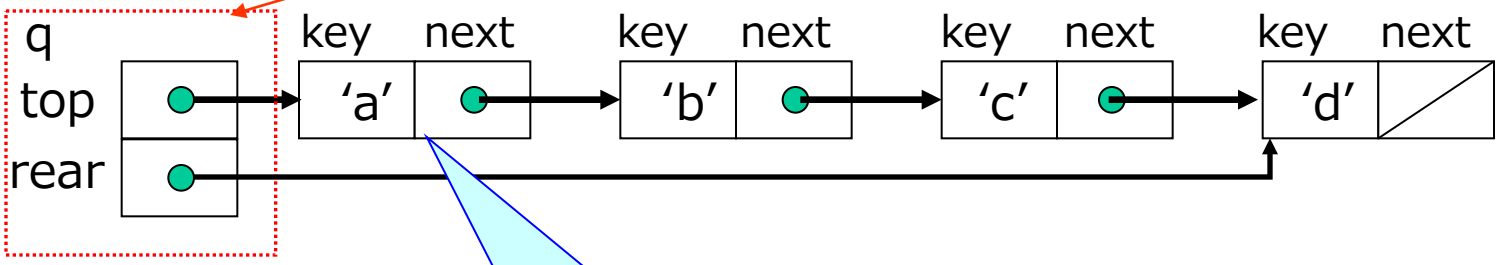


必須課題8-4：dequeue （1/3）

```
char dequeue (struct queue *q);
```

取り出したリストの要素の値（key）

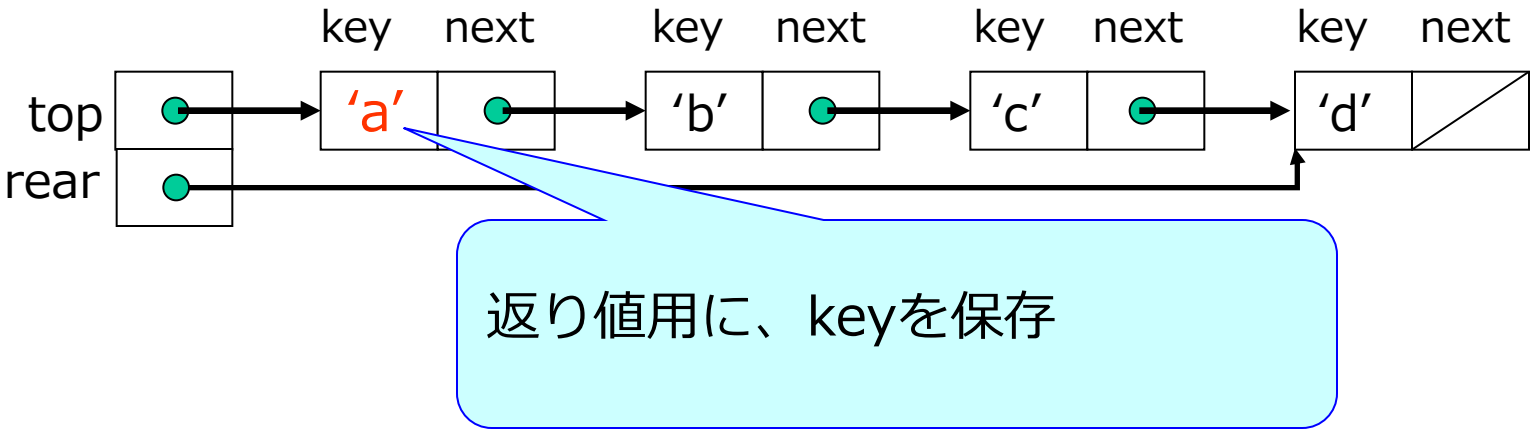
構造体queueへのポインタ



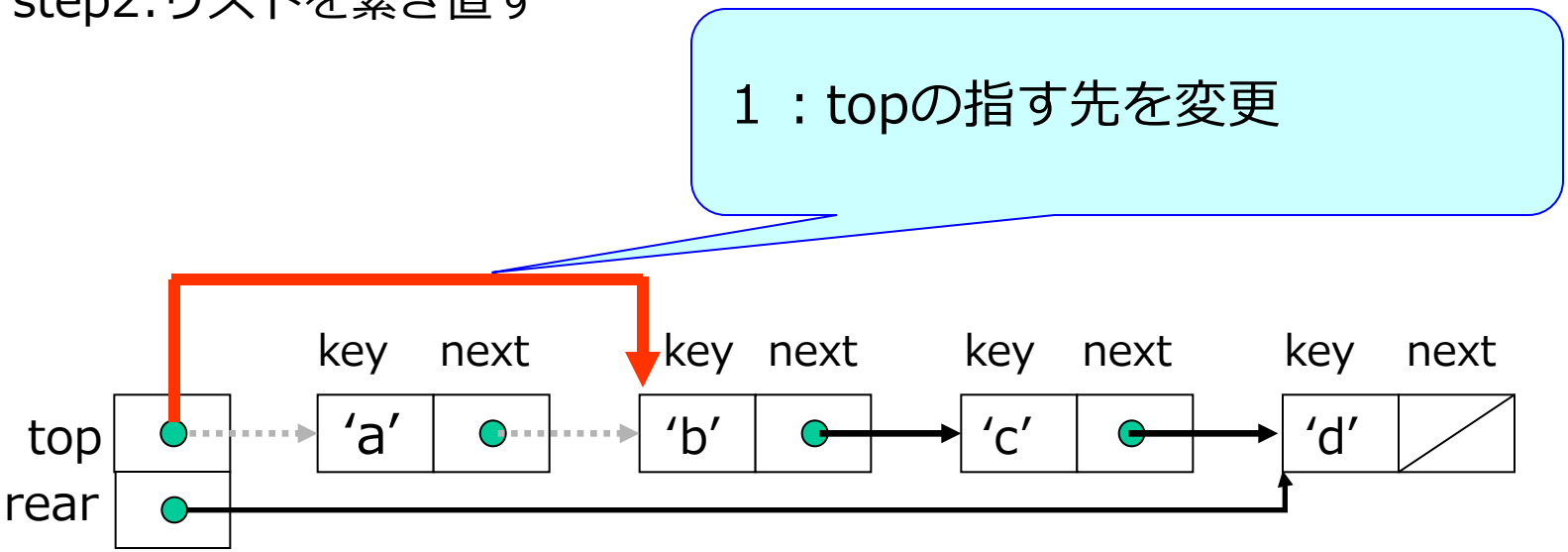
dequeue（要素取り出し）
キューの場合も
リストの先頭の要素
を取り出す

必須課題8-4：dequeue （2/3）

step1:先頭要素の値(key)を保存

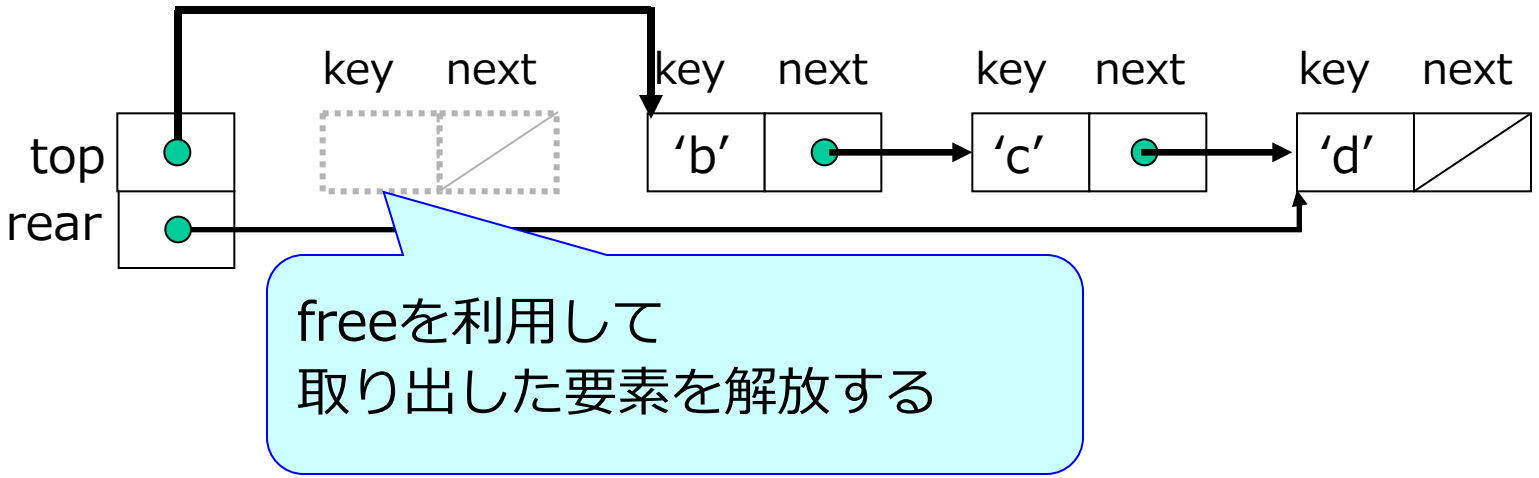


step2:リストを繋ぎ直す



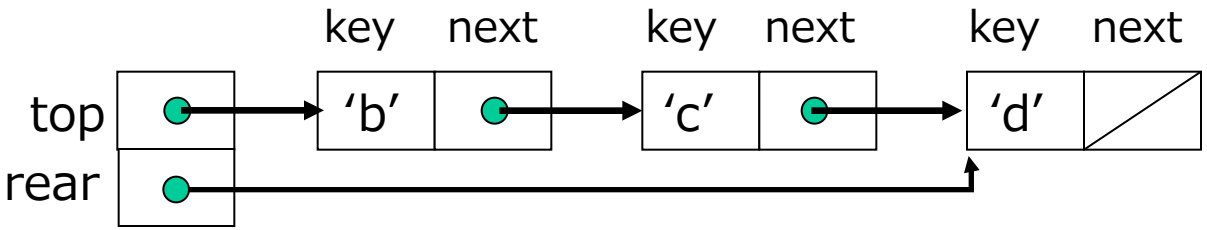
必須課題8-4：dequeue （3/3）

step3:取り出した要素のメモリを解放する



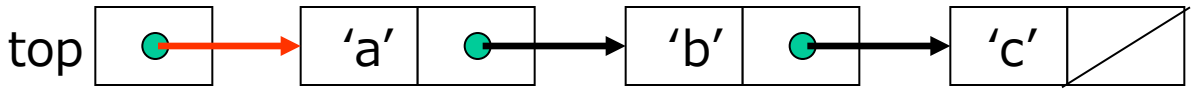
step4:保存した値（key）を返り値として返す

dequeue後のリスト



補足：リストのメモリ解放

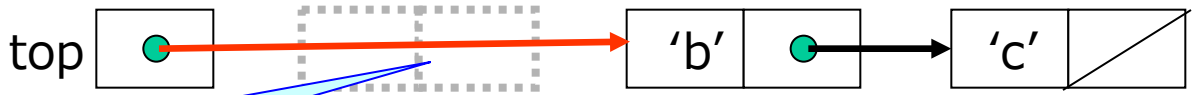
リストの利用後はメモリを解放しよう



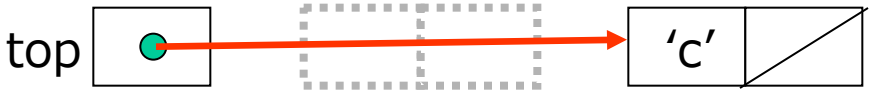
リストを辿りながらメモリを解放する

辿り方は７－２、７－４を参考にしよう

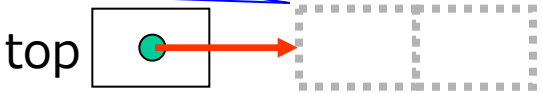
リストの繋ぎ直し方＆メモリの解放は
８－２と同様



リストを繋ぎ直してから
メモリを解放すること



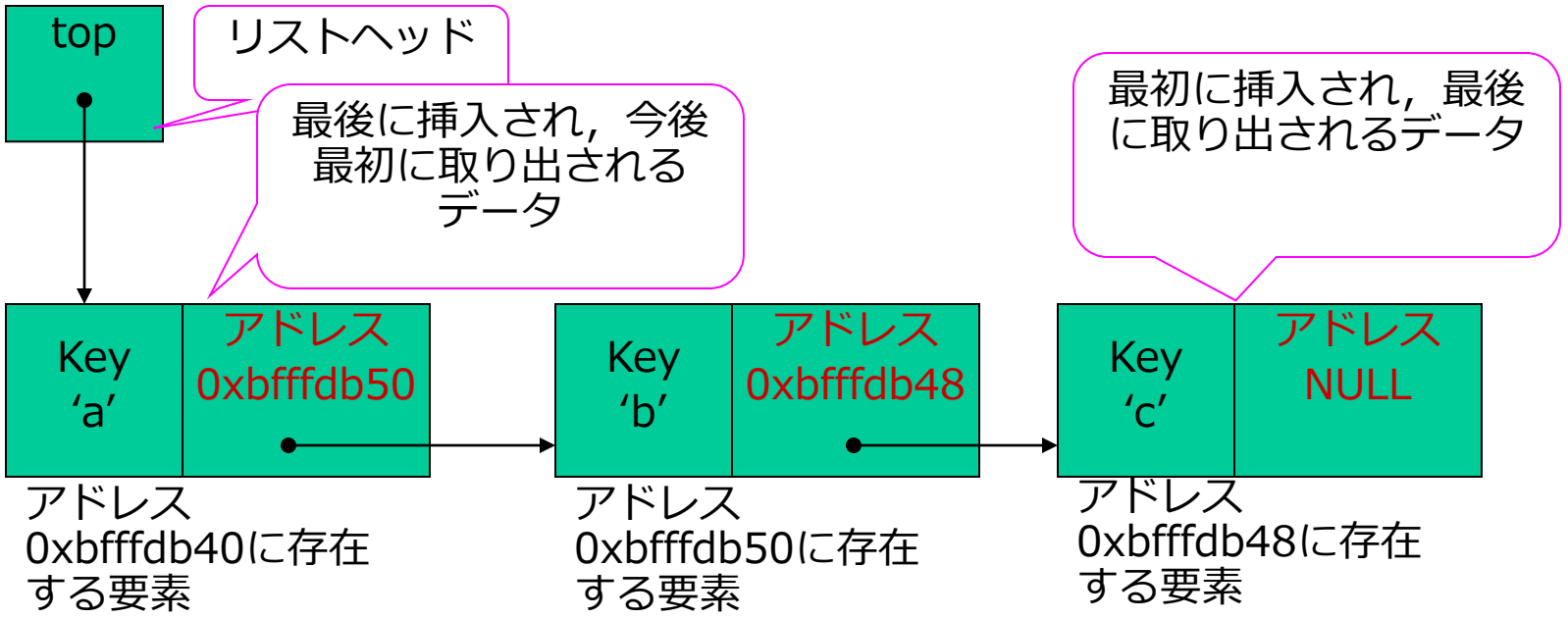
NULLになったら
（すべて解放したら）終了



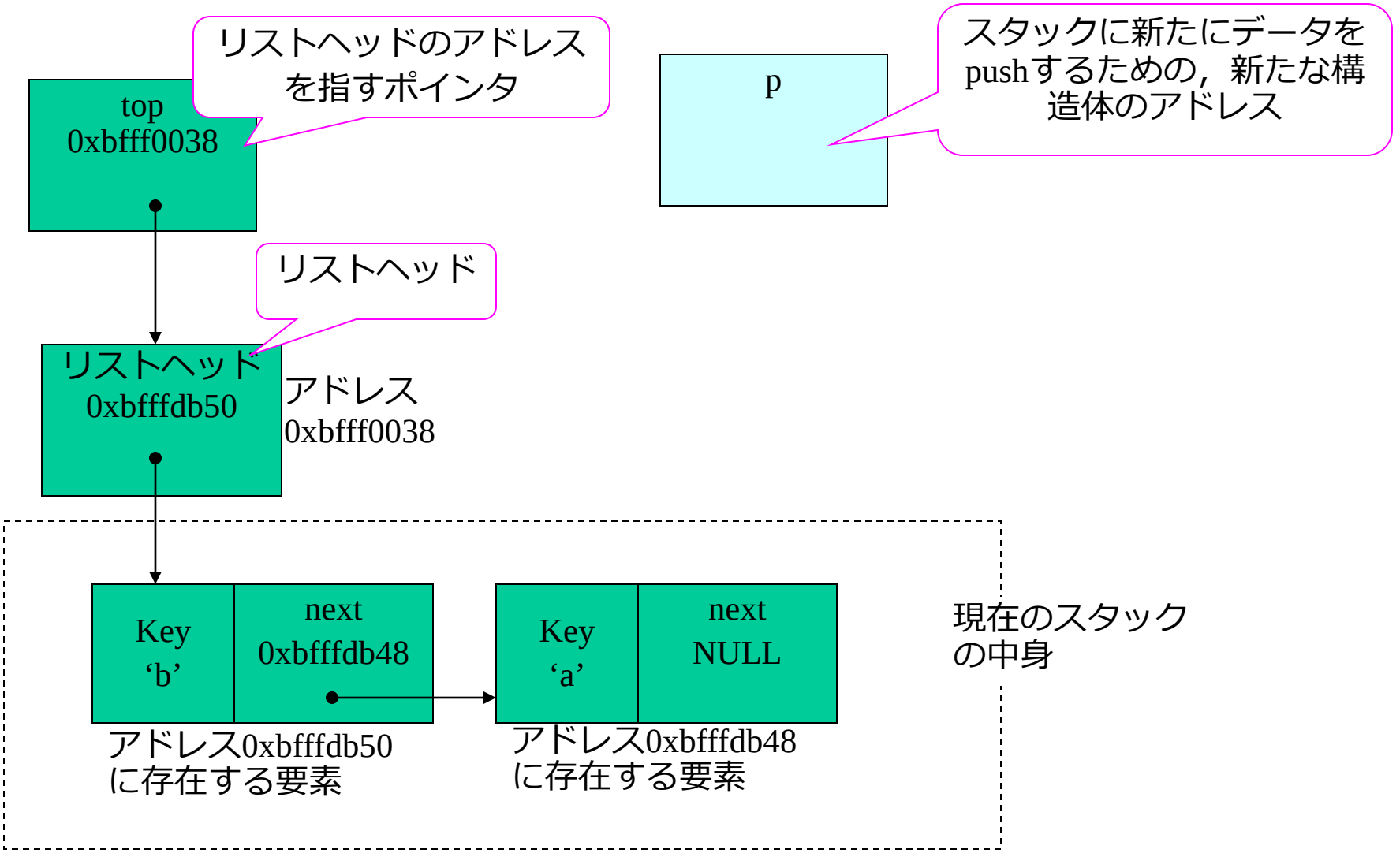
参考資料
関数のイメージ(1)

線状リストによるスタックの実現

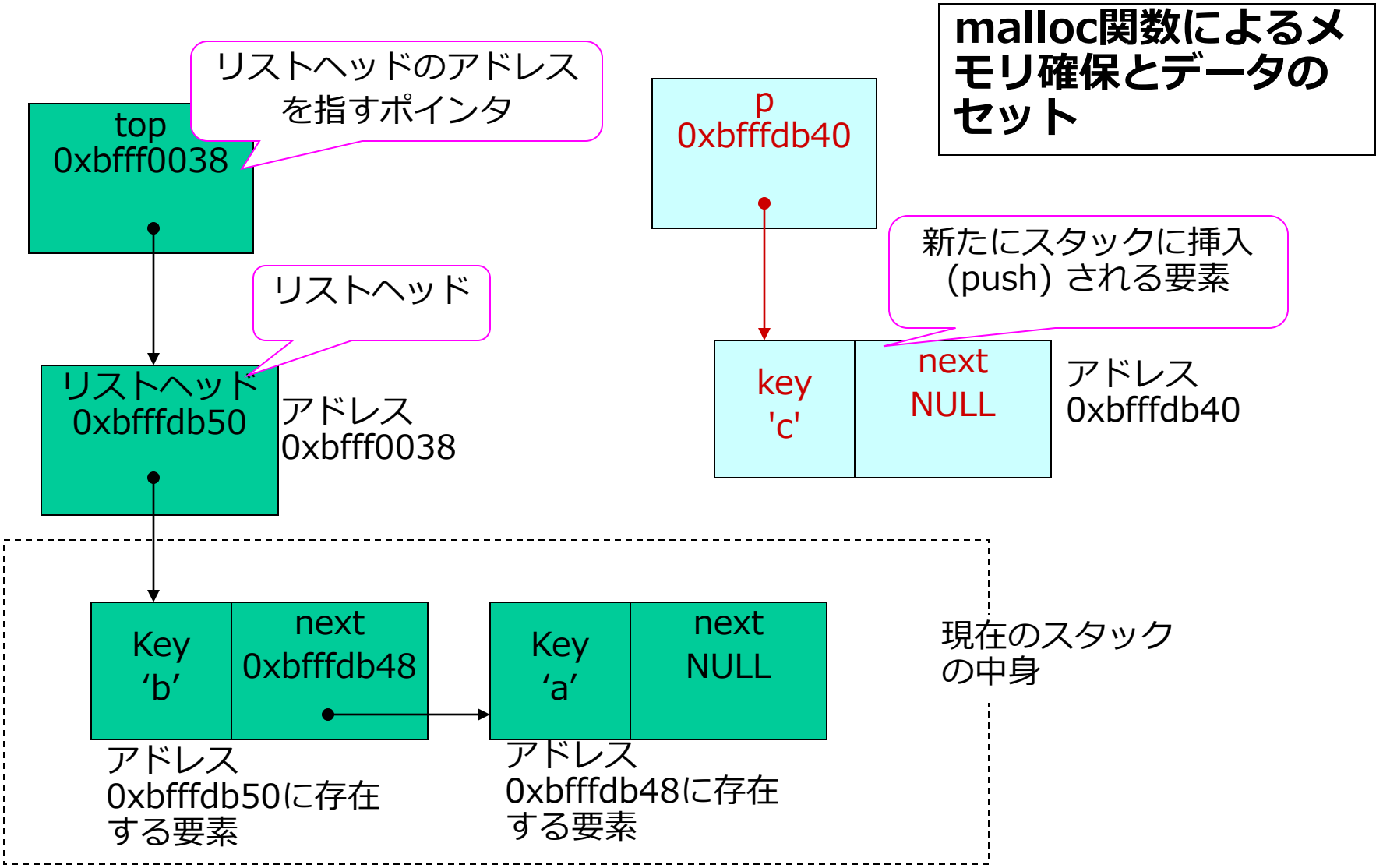
- リストヘッドtopを持つ線状リスト
  - 先入れ後出し (Last-In First-Out)を，リストの先頭における挿入，削除により実現
- push操作 (データ挿入): リストの先頭へのデータ挿入
- pop操作 (データ取り出し): リストの先頭のデータ削除



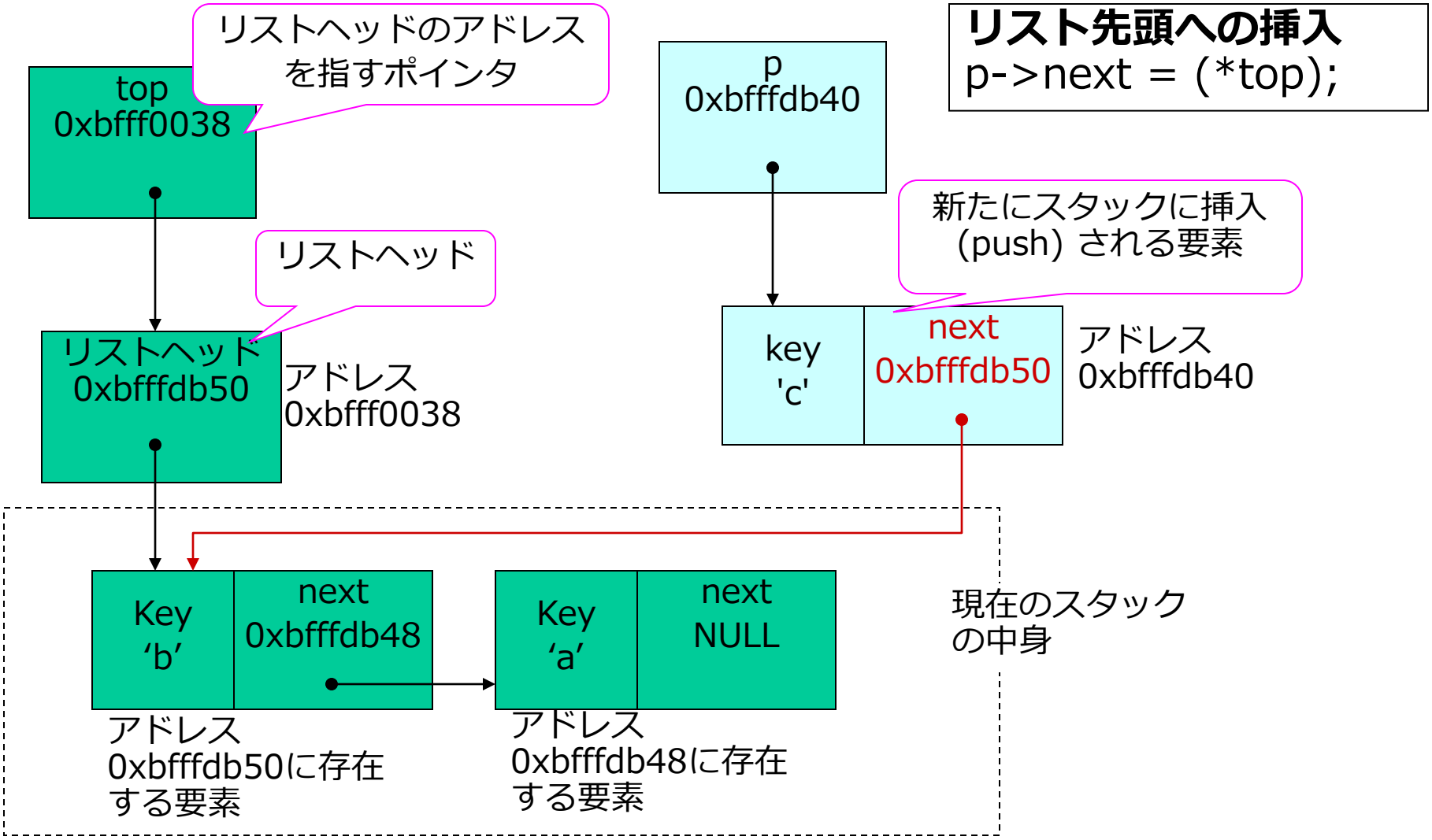
関数push(struct data **top, char c)のイメージ(1)



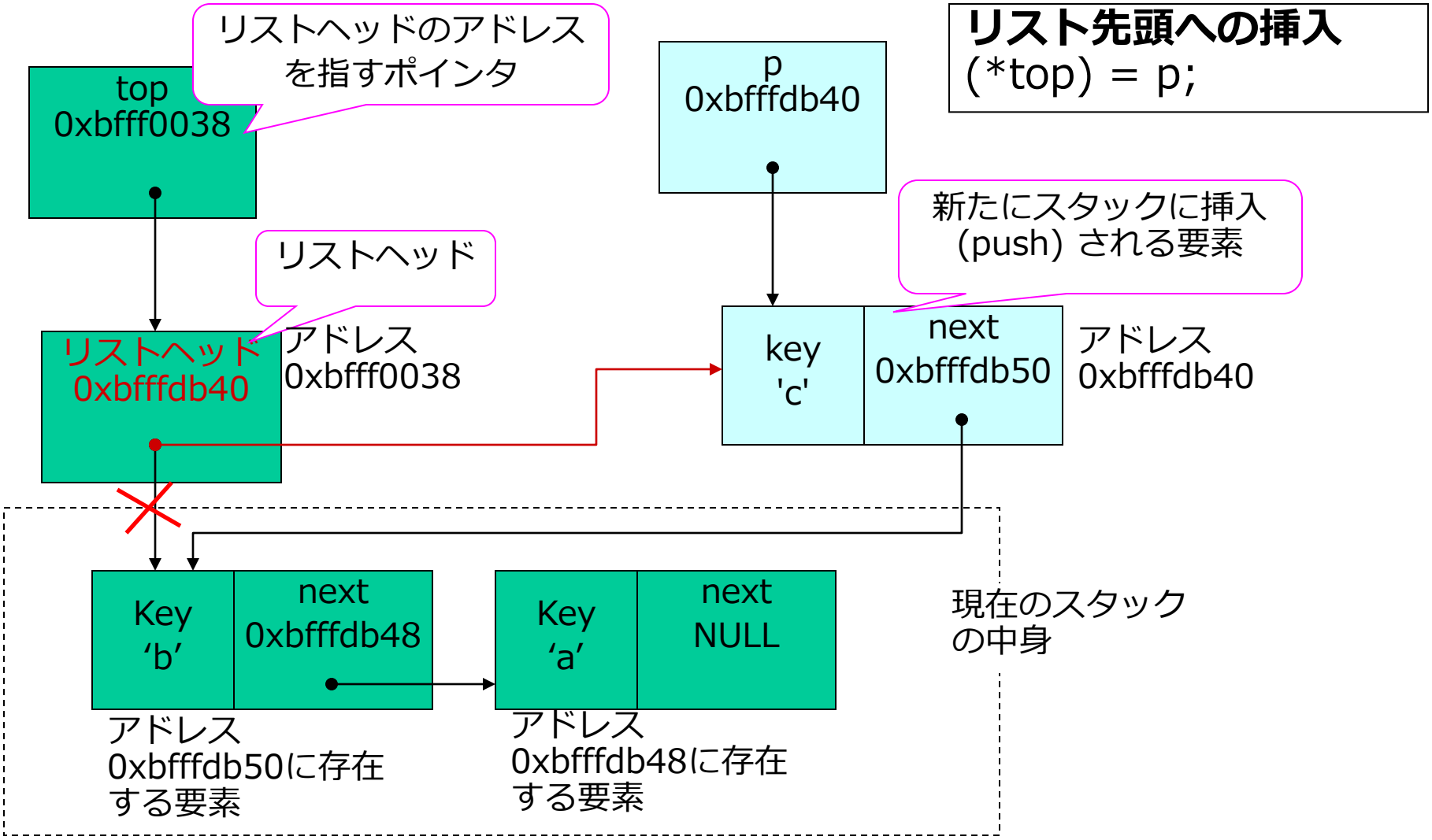
関数push(struct data **top, char c)のイメージ(2)



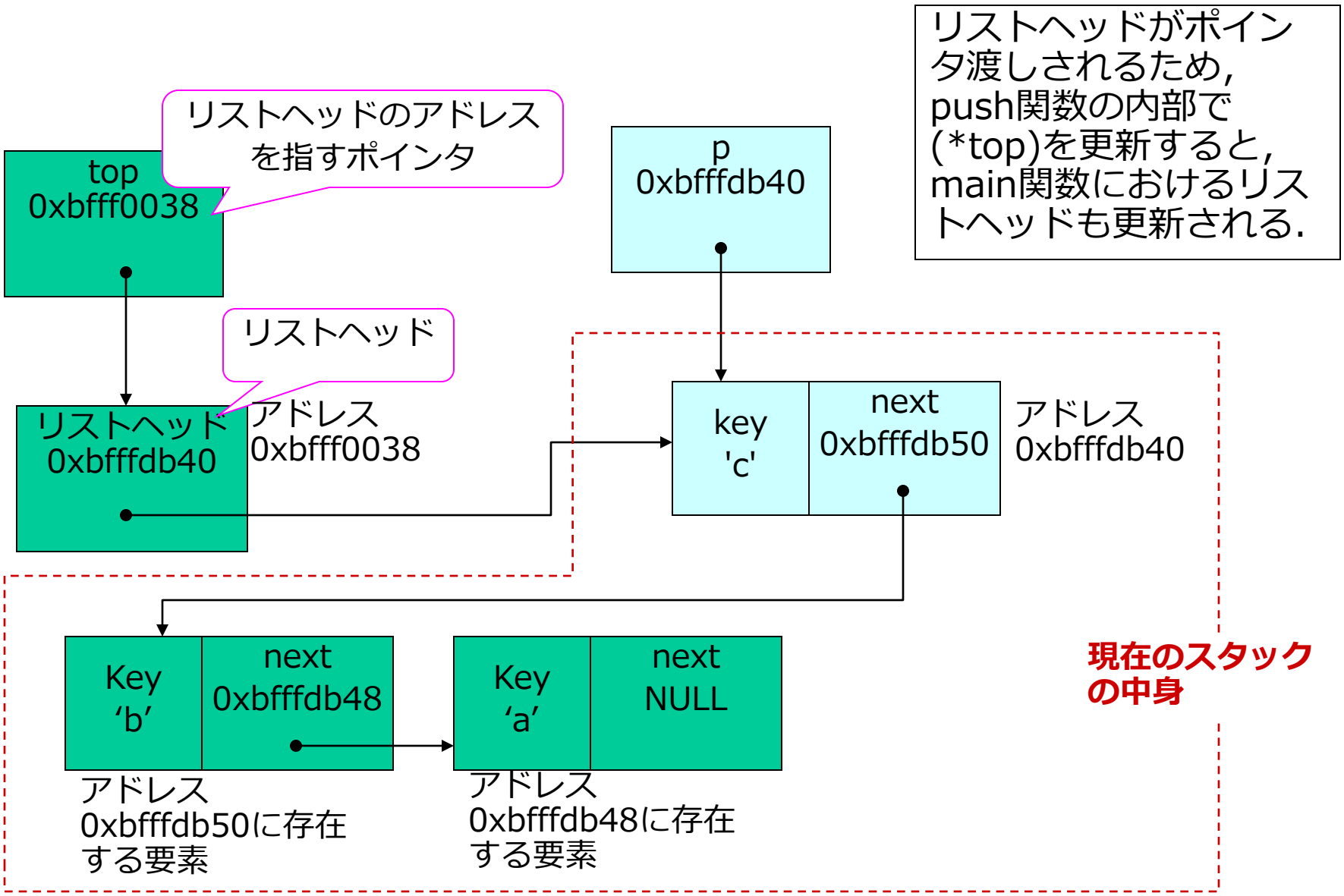
関数push(struct data **top, char c)のイメージ(3)



関数push(struct data **top, char c)のイメージ(4)



関数push(struct data **top, char c)のイメージ（5）



著者リスト

1. 安積 卓也（情報システム学科）
2. 原田 史子（情報システム学科