==========

B*-Tree の挿入アルゴリズム

==========

```
1. proc insert_record(P,record)
2. Input:
3.        P  ：対象ノードへのポインタ(最初は root)
4.        record :  挿入するレコード
5.     output
6.      newEnt :  親ノードに挿入するエントリ（なければ NULL)
7.     N = P のさすノード
8.     if N が non-leaf ノードである：
9.      K_i <  entry の key 値  ≤ K_i+1    となる i を探す
10.     newEnt_fromchild = insert(P_i, entry) //再帰
11.     if N に空きがある：
12.             newEnt_fromchild を挿入して  return null;
13.     else : //N に空きがない場合
14.             newEnt = divide_nonleaf(N)
15.             if  ノード N が root ノードである：
16.                     新しい root ノード R を作成
17.                     add_entry(R, newEnt)
18.                     return null
19.             else:
20.                     return newEnt
21.     else : //N が leaf ノードである
22.             if N に空きがある
23.                     add_entry(N, record)
24.                     return null
25.             else
26.                     return divide_leaf(N)
```

==========

B*-Tree のノード分割アルゴリズム

==========

```
proc divide_nonleaf ( N )
    input
        N : 分割する non-leaf ノード
    output
        newEnt : 親ノードに挿入するエントリ
    process:
        ノード M を作る
          N の d+2 番目から 2d+1 番目までの key 値を M へ移動
          N の d+2 番目から 2d+2 番目までのポインタを M へ移動
          N から M へ兄弟ポインタを張る
        newEnt = <N の d+1 番目の key 値, M へのポインタ>
        N の d+1 番目の key 値を削除
        return newEnt;

proc divide_leaf ( N )
     input :
        N : 分割する leaf
     output
        newEnt : 親ノードに挿入するエントリ

     process :
        ノード M を作る
          N の d+1 番目から 2d+1 番目までのエントリを M へ移動
          N から M へ兄弟ポインタを張る
        newEnt = <M の 0 番目の key 値, M へのポインタ>
        return entEnt;
```

==========

B*-Tree のエントリ追加アルゴリズム (non-leaf)

==========

add_entry (newEnt)

input:

newEnt： 挿入するエントリ。key 値とポインタを持つ。

process:

$K_i \leq$ newEnt の key 値 $< K_{i+1}$ となる i を探す

i 番目と i+1 番目の間に空のエントリを追加する

（i+1 番目以降は 1 つずつ順番がずれる）

i+1 番目のポインタに i+2 番目のポインタを代入する

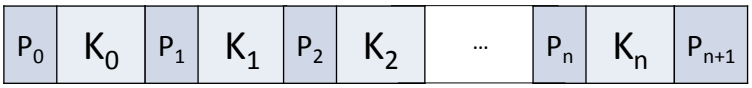
i+2 番目のポインタに newEnt のポインタを代入する

i+1 番目の key 値に newEnt の key 値を代入する

# B*-tree : ノードのデータ構造

ノード (non-leaf)

$P_i$ : $K_{i-1}$<value<$K_{i+2}$へのポインタ
$K_i$ : key値



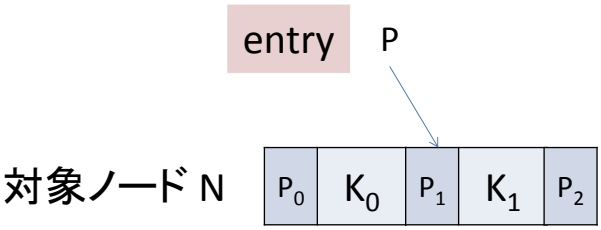

ノード (leaf)


# B*-tree : 挿入アルゴリズム

**proc insert(P, entry)**

Input: P : 対象ノードへのポインタ
entry : 挿入するエントリ

output newEnt : 親ノードに挿入するエントリ



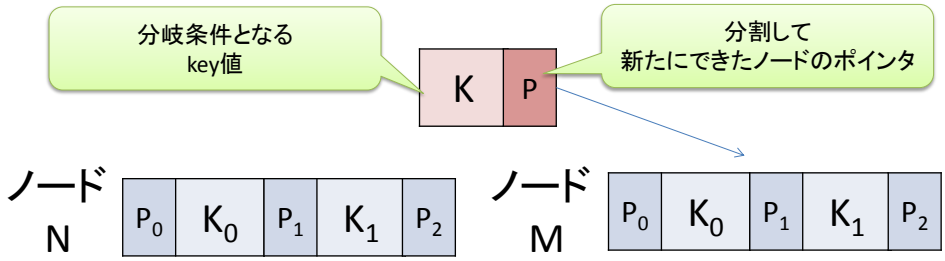
# B*-tree : 挿入アルゴリズム

**proc insert(P, entry)**

Input:　P : 対象ノードへのポインタ

entry : 挿入するエントリ

output　newEnt : 親ノードに挿入するエントリ




# B*-tree: 挿入アルゴリズム (その1 : 全体の流れ)

N = Pのさすノード

if Nがnon-leafノードである :

$K_i$ < entryのkey値 ≤ $K_{i+1}$ となるiを探す

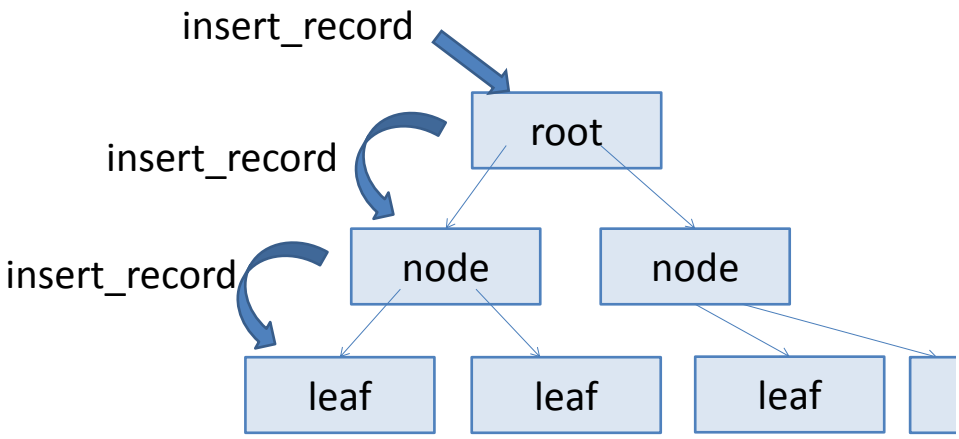
newEnt_fromchild = insert($P_i$, entry) //再帰

if Nに空きがある :

newEnt_fromchildを挿入して return null;

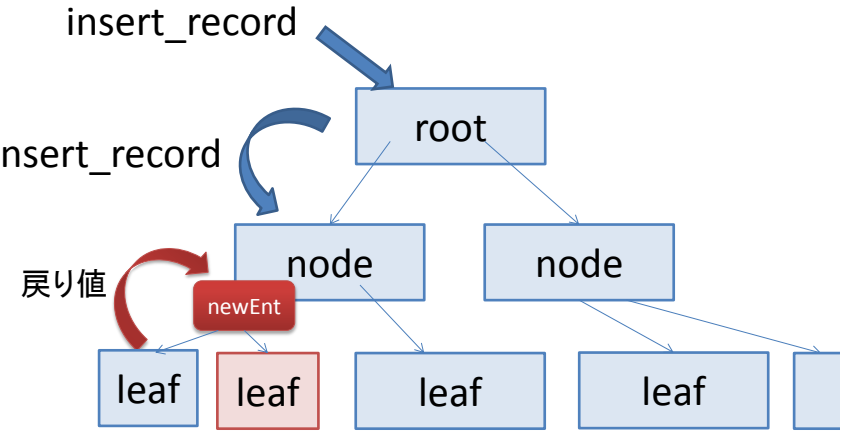
# B*-tree: 挿入アルゴリズム (その1：全体の流れ)

- 再帰的に実行




# B*-tree: 挿入アルゴリズム (その1：全体の流れ)

- 再帰的に実行



# B*-tree: 挿入アルゴリズム (その1：全体の流れ（再掲））

N = Pのさすノード

if Nがnon-leafノードである：

$K_i$ ＜ entryのkey値 ≤ $K_{i+1}$ となるiを探す

newEnt_fromchild = insert($P_i$, entry) //再帰

if Nに空きがある：

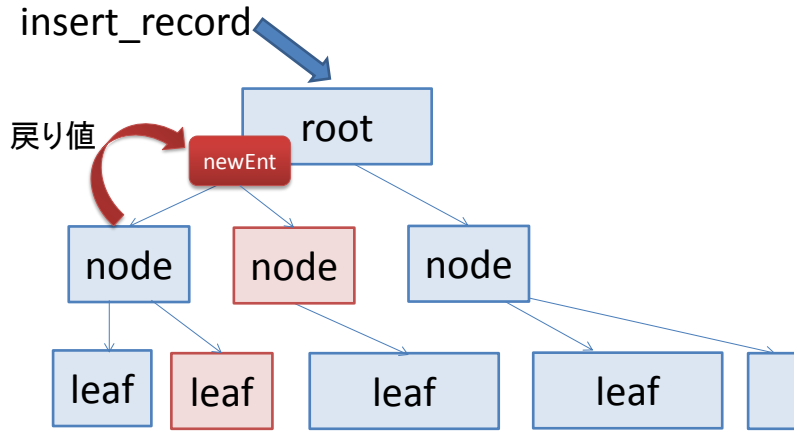
newEnt_fromchildを挿入して return null;

# B*-tree：挿入アルゴリズム （その2：ノードの分割）

```
11.  else : //Nに空きがない場合
12.      newEnt = divide_nonleaf(N)
13.      if ノードNがrootノードである：
14.          新しいrootノードRを作成
15.          add_entry(R, newEnt)
16.          return null
17.      else:
18.          return newEnt
```

# B*-tree: 挿入アルゴリズム (その2:ノードの分割)

- 再帰的に実行

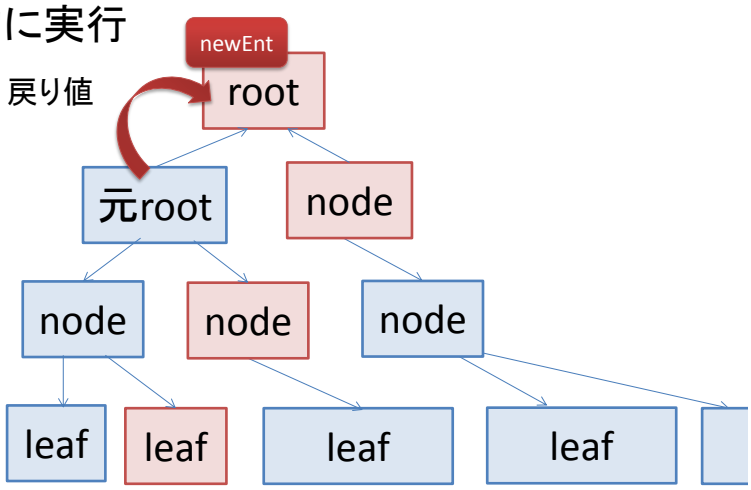



# B*-tree : 挿入アルゴリズム (その2:ノードの分割)

```
11.  else : //Nに空きがない場合
12.      newEnt = divide_nonleaf(N)
13.      if ノードNがrootノードである:
14.          新しいrootノードRを作成
15.          add_entry(R, newEnt)
16.          return null
17.      else:
18.          return newEnt
```

# B*-tree: 挿入アルゴリズム (その2：ノードの分割)

- 再帰的に実行

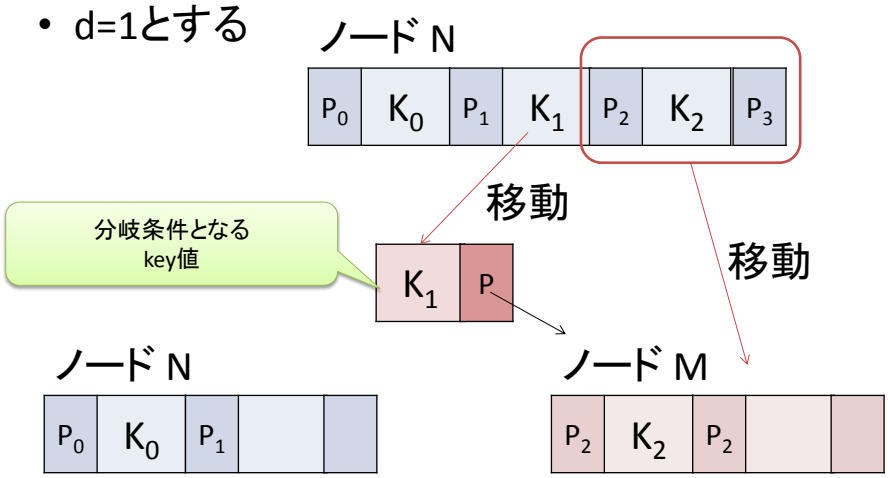



# B*-tree: 分割アルゴリズム (nonleafノードの場合)

```
proc divide_nonleaf ( N )
    input
        N : 分割する non-leaf ノード
    output
        newEnt : 親ノードに挿入するエントリ
    process:
        ノード M を作る
          N の d+2 番目から 2d+1 番目までの key 値を M へ移動
          N の d+2 番目から 2d+2 番目までのポインタを M へ移動
          N から M へ兄弟ポインタを張る
        newEnt = <N の d+1 番目の key 値, M へのポインタ>
        N の d+1 番目の key 値を削除
        return newEnt;
```

# B*-tree：ノード分割アルゴリズム

- d=1とする

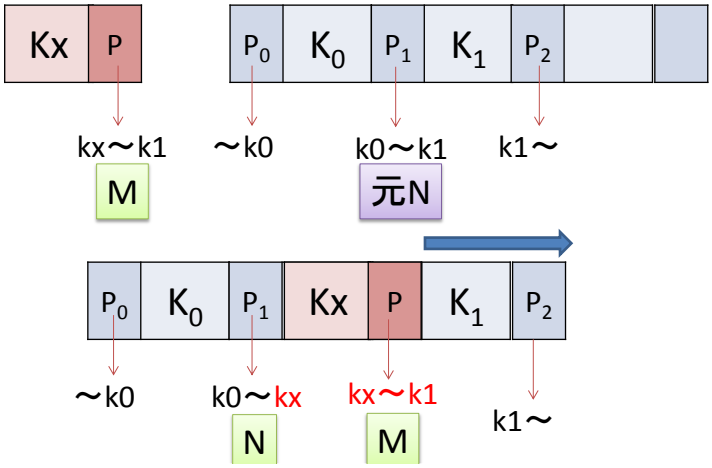



# B*-tree：ノードエントリ追加アルゴリズム (non-leaf)

add_entry (newEnt)
  input:
    newEnt： 挿入するエントリ。key 値とポインタを持つ。
  process:
    $K_i \leq$ newEnt の key 値 $< K_{i+1}$ となる i を探す
    i 番目と i+1 番目の間に空のエントリを追加する
      （i+1 番目以降は 1 つずつ順番がずれる）
    i+1 番目のポインタに i+2 番目のポインタを代入する
    i+2 番目のポインタに newEnt のポインタを代入する
    i+1 番目の key 値に newEnt の key 値を代入する

- K0<Kx<K1となる以下のエントリを挿入する場合

つまり、kxによって N（k0～kx） M（kx～k1） と分割されている




# B*-tree: 分割アルゴリズム (leafノードの場合）

```
proc divide_leaf ( N )
    input :
      N : 分割する leaf
    output
      newEnt : 親ノードに挿入するエントリ

    process :
      ノード M を作る
        N の d+1 番目から 2d+1 番目までのエントリを M へ移動
        N から M へ兄弟ポインタを張る
      newEnt = <M の 0 番目の key 値, M へのポインタ>
      return entEnt;
```

# B*-tree : 分割アルゴリズム (leafノードの場合)

- d=1とする