アルゴリズムとデータ構造

外部探索

目次

- 索引順アクセス
- B木

実システムに即した探索法

- 多くの実システムでは，大規模なファイル（データ）から項目を探し出すことが必要となる．
- 巨大な記号表を効率よく扱う探索アルゴリズムは，実用上きわめて重要．

想定する記憶装置のモデル

- 記号表を保持する記憶装置が次の特徴を持つことを想定した上で，効率の良い探索方法を考える．

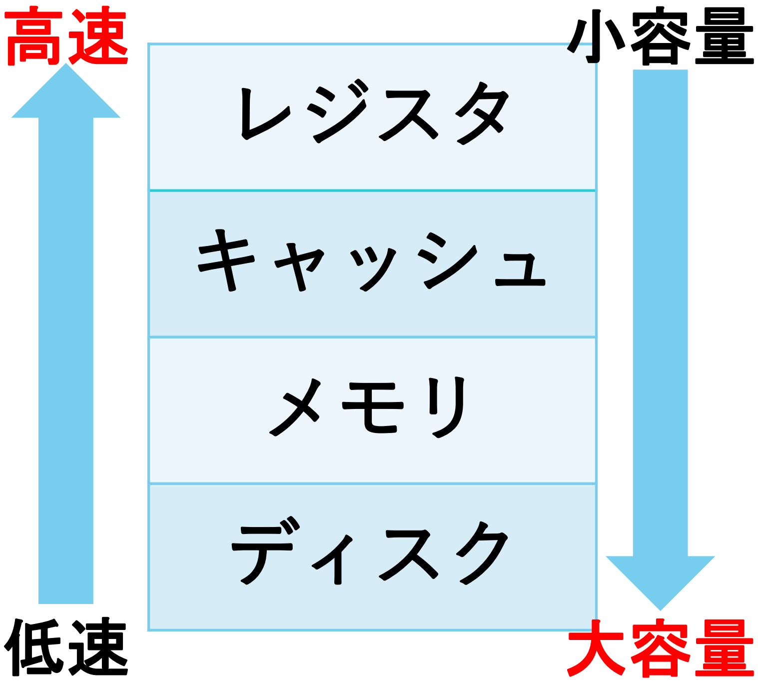
  - ファイルは**ページ単位**に分割されている．
    - ページ：ディスク装置によって効率よく呼び出される連続した情報の区画
  - 各ページは多くの項目を保持する．
  - ページの読み出し時間は，読みだされたページ内の項目を読み出す時間と比べて十分大きい．

想定する記憶装置のモデル

- 想定する特徴は，実際の記憶装置の多くにあてはまる．

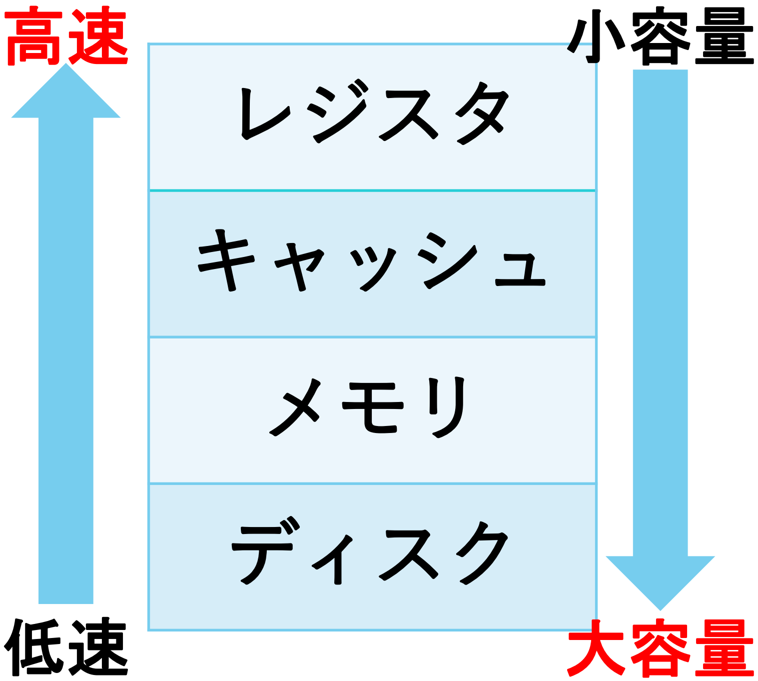
想定する記憶装置のモデル

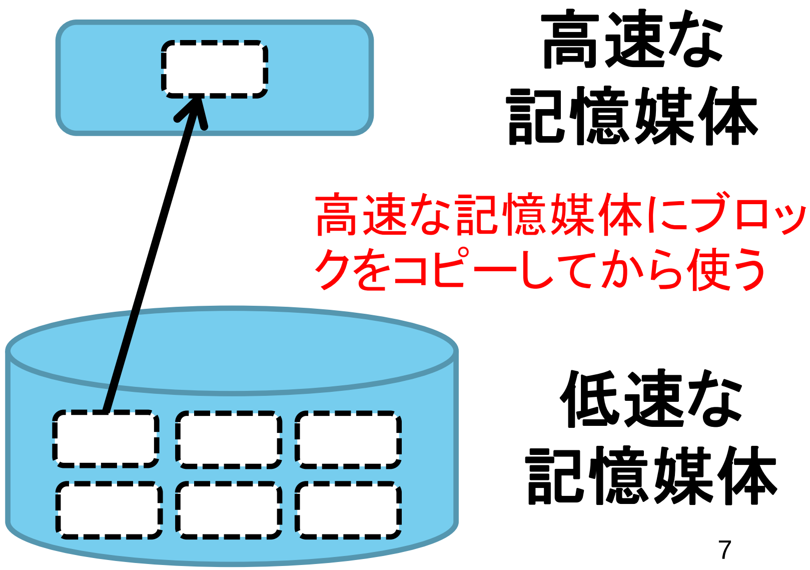
- 想定する特徴は，実際の記憶装置の多くにあてはまる．



想定する記憶装置のモデル

- 想定する特徴は，実際の記憶装置の多くにあてはまる．
  - メインメモリとキャッシュメモリの関係等



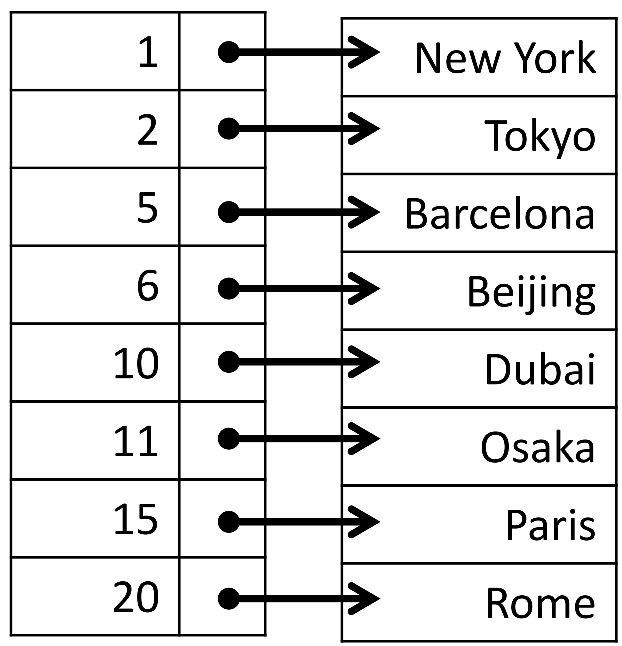


索引

- 記号表を構成する項目はある一定の形式でどこか（例えばディスク）に格納されていると仮定.
- キーと項目への参照（項目が格納されている場所のアドレス等）を対応づけるデータ構造を**索引**と呼ぶ.
- 記号表において，探索，挿入などの操作をする際に，**できるだけページの読み出し回数が少なくなるように索引を設計**することが重要.

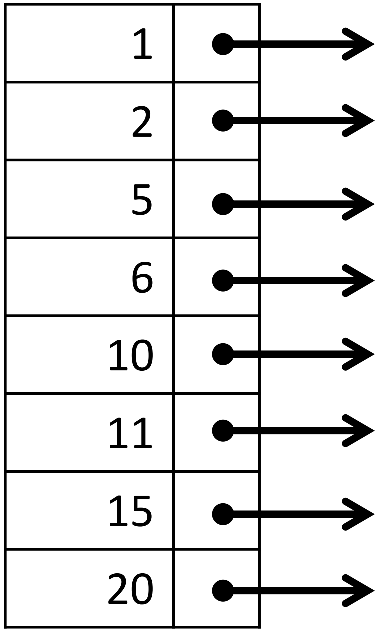
単純な索引

- キーと対応する項目への参照をキーの順に整列しておけば索引として機能する.
  - 2分探索の場合，lgN回の探索が必要



単純な索引

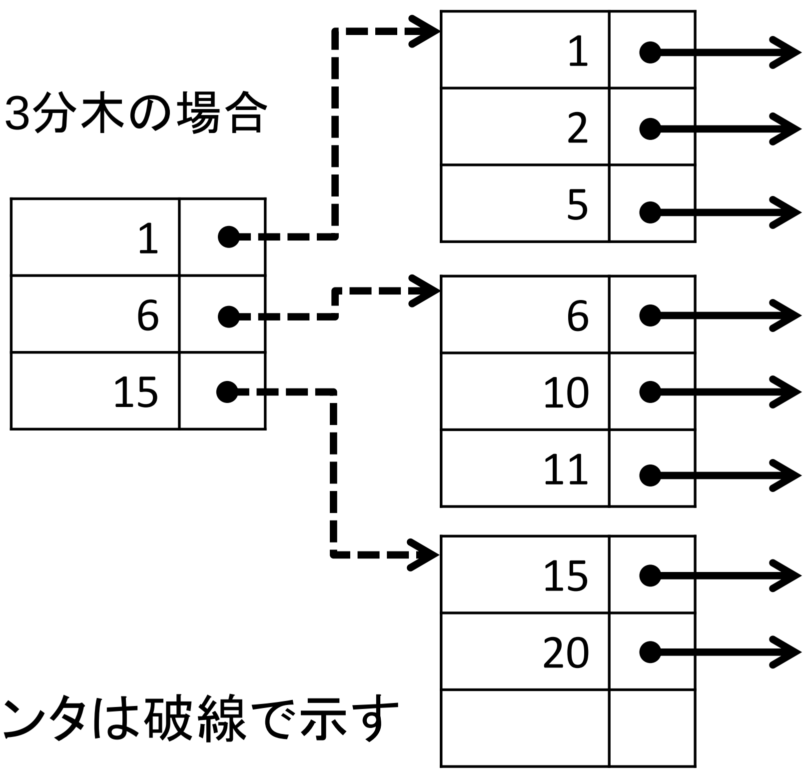
- キーと対応する項目への参照をキーの順に整列しておけば索引として機能する．
  - 2分探索の場合，lgN回の探索が必要
- 巨大なデータを扱う場合，索引が一つのページにおさまらず，次に参照すべきページを見つけられない．
  （一般的に，ページへのポインタを通じてのみ次のページを読み出せる．）



索引順アクセス法

- 平衡多分木を用いることを考える.
  - 内部節点にはキーとページへのポインタを格納.
  - キーと項目への参照は外部節点に置く.

3分木の場合

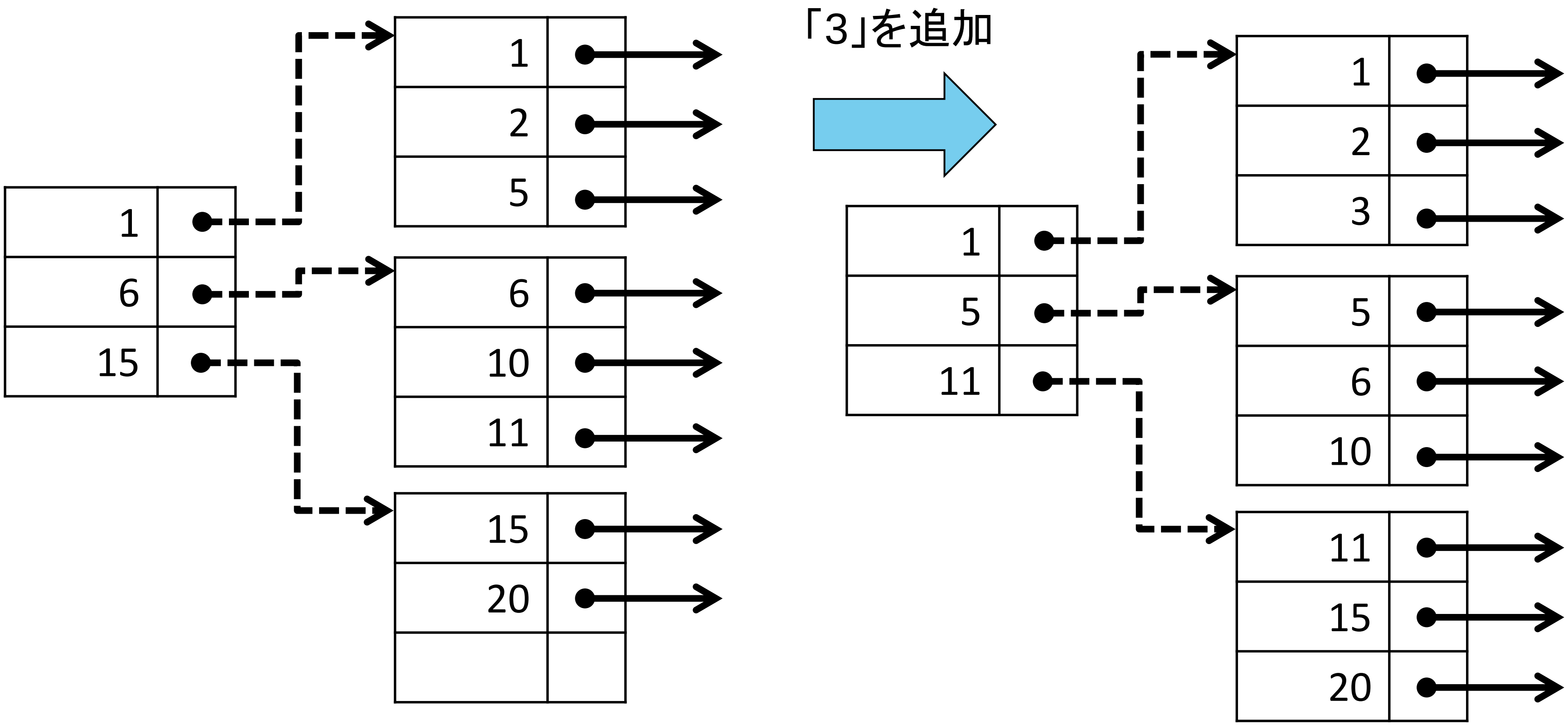


※ ページへのポインタは破線で示す

索引順アクセス法

- 平衡多分木を用いることを考える．
  - 内部節点にはキーとページへのポインタを格納．
  - キーと項目への参照は外部節点に置く．
- ページ内の情報の読み出しコストが十分小さいことを想定すると，M分木による探索であれば $\log_M N$回の探索に抑えることができる．
  - Mが1000，項目数が1兆以下であれば5回以下．（MがNに対して十分に大きければ，$\log_M N$回は定数回と遜色ないほど小さい．）

索引順アクセス法



- キーを追加する際に，索引全体を再構成しなければならない可能性がある．

索引順アクセス法の性質

- 性質16.1
  - 索引順アクセスファイル上の探索では，一定回数※のページ読み出しが実行される．挿入では，ファイル全体の再構成を必要とすることがある．

※ MがNに対して十分に大きいことを想定して，一定回数としている．

B木

- 項目の挿入／削除に対しても頑健な探索構造を作るため，多分木の各節点が持つキーの数をちょうどM個としない．
- 各節点では，1ページに収まり，かつ，探索経路を短く保つのに十分な数のキーを保持する．（根は例外）

B木の定義

- M次のB木は空か，k-節点からなる木である．k-節点はk-1個のキーを持ち，それらのキーで区切られるk個の区間のそれぞれを表す部分木へのk個のリンクを持つ．B木は次の構造的性質をもつ．
  - 根では，$2 \leq \mathrm{k} \leq \mathrm{M}$ となる．
  - 根以外の節点では，$\frac{\mathrm{M}}{2} \leq \mathrm{k} \leq \mathrm{M}$ となる．
  - 根から葉への距離は全て同じ．

B木の定義（4次の場合）

- 4次のB木は空か，k(2,3,4)-節点からなる木である．K(2,3,4)-節点はk-1(1,2,3)個のキーを持ち，それらのキーで区切られるk(2,3,4)個の区間のそれぞれを表す部分木へのk(2,3,4)個のリンクを持つ．4木は次の構造的性質をもつ．
  - 根では，$2 \leq k \leq 4$ となる．
  - 根以外の節点では，$2 \leq k \leq 4$ となる．
  - 根から葉への距離は全て同じ．

2-3-4木は4次のB木と同一．

2-3-4木（復習）

- 2-3-4木(2-3-4 search tree)は，空か3種類の節点からなる木である
  - 2-節点は1個のキーとそれより小さいキーを持つ木への左リンクとそれより大きいキーを持つ木への右リンクを持つ
  - 3-節点は2つのキーと，2個のキーより小さい全てのキーを持つ木への左リンク，2個のキーの間にある全てのキーへの中央リンク，2個のキーより大きい全てのキーを持つ木への右リンクを持つ
  - 4-節点は3個のキーと4つのリンクを持つ．それぞれ3個のキーで決まる4つの区間へのリンクである
- 平衡2-3-4木は，根から外部節点への距離が全て等しい2-3-4木である

項目の挿入

- 根から順に，挿入する項目をの位置を探す．
- 途中で飽和節点（M個のリンクを持つ節点）に出会ったら，節点を二つに分割し，中央のキーを親節点に挿入する．
- 根が飽和節点の場合は，節点を二つに分割し，中央のキーを新たな根として設置する．
  - 根の節点数はM/2よりも小さくなることを許す必要がある．

トップダウン2-3-4木への挿入（復習）

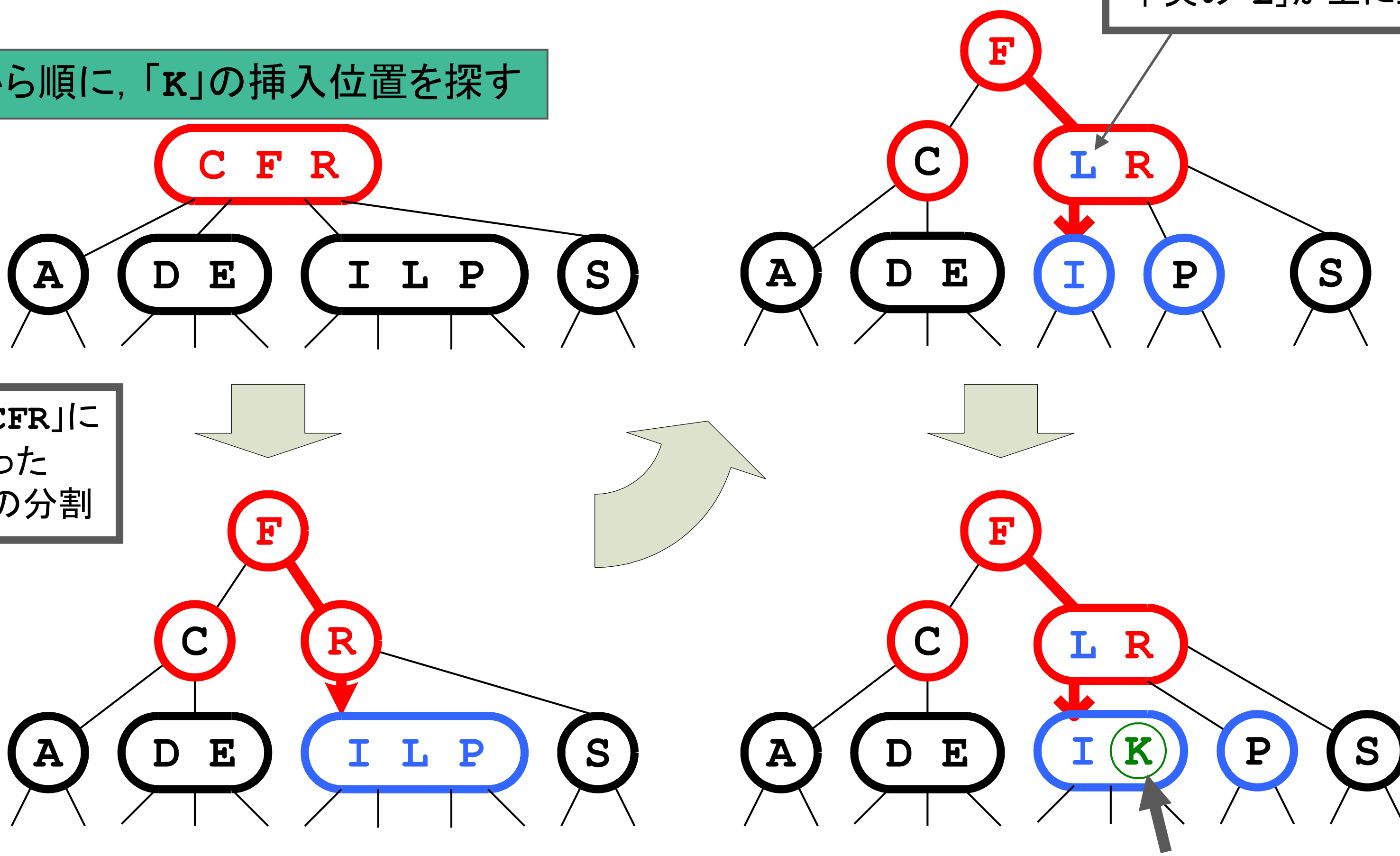
「K」の挿入

探索途中で，4-節点に出合ったら，その度に，
4-節点の分割を行う

根から順に，「K」の挿入位置を探す

中央の「L」が上に上がった

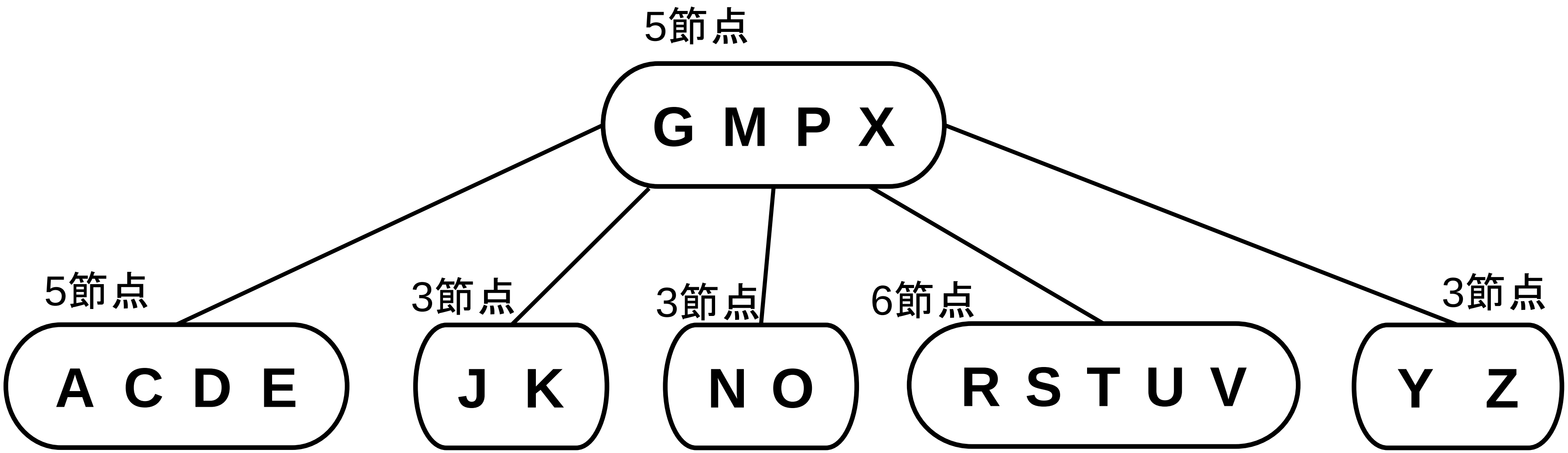
4-節点「CFR」に
出合った
→ 節点の分割



4-節点「ILP」に出合った → 節点の分割

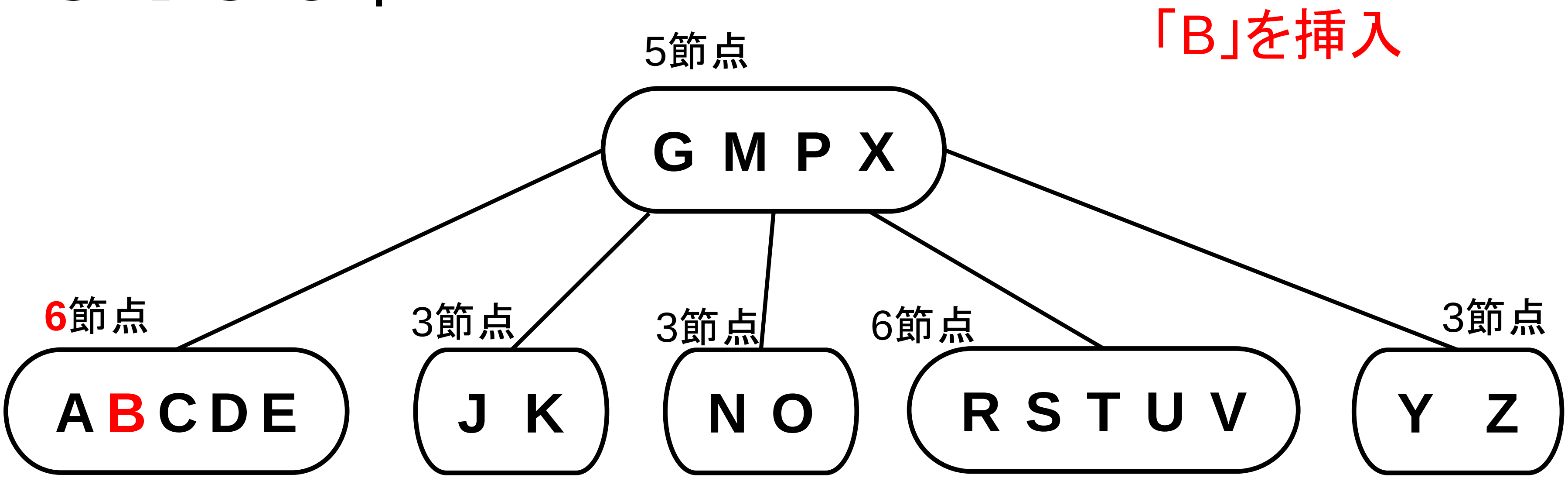
葉に到達．「K」を挿入する

3-4-5-6木



※ 外部節点への枝は省略

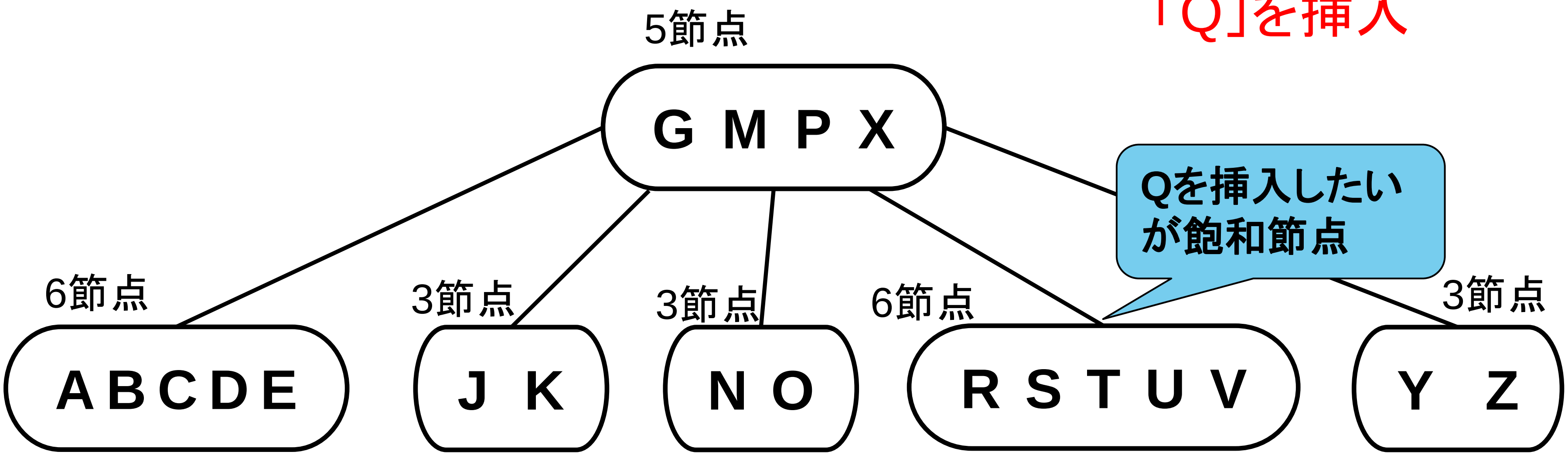
3-4-5-6木



※ 外部節点への枝は省略

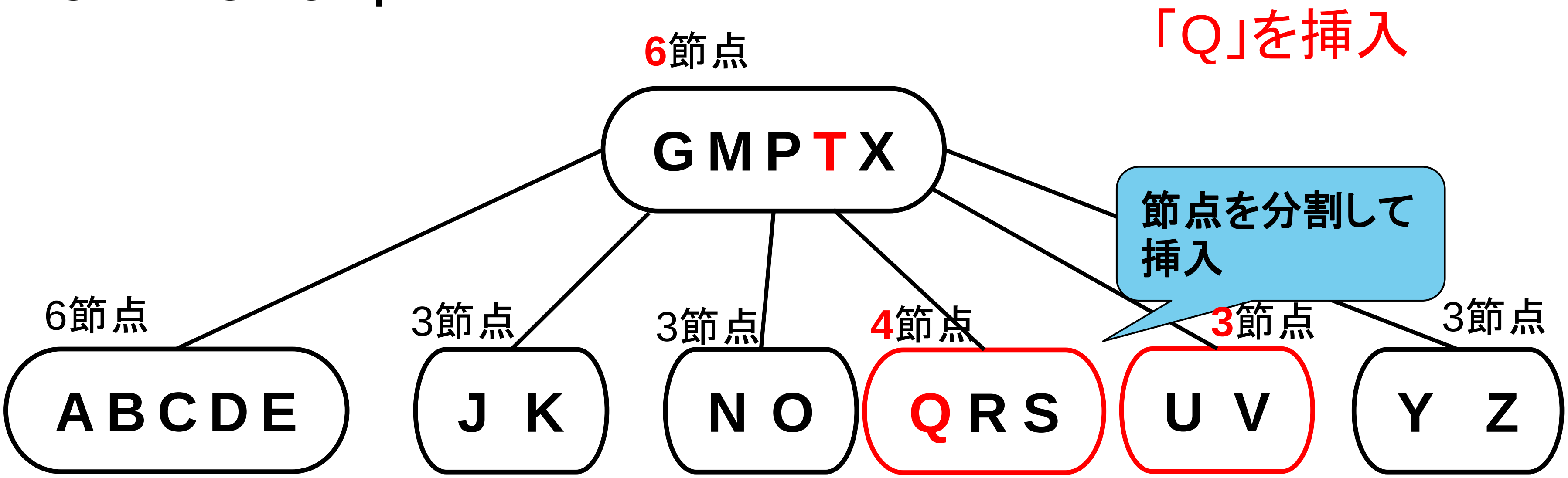
3-4-5-6木

「Q」を挿入



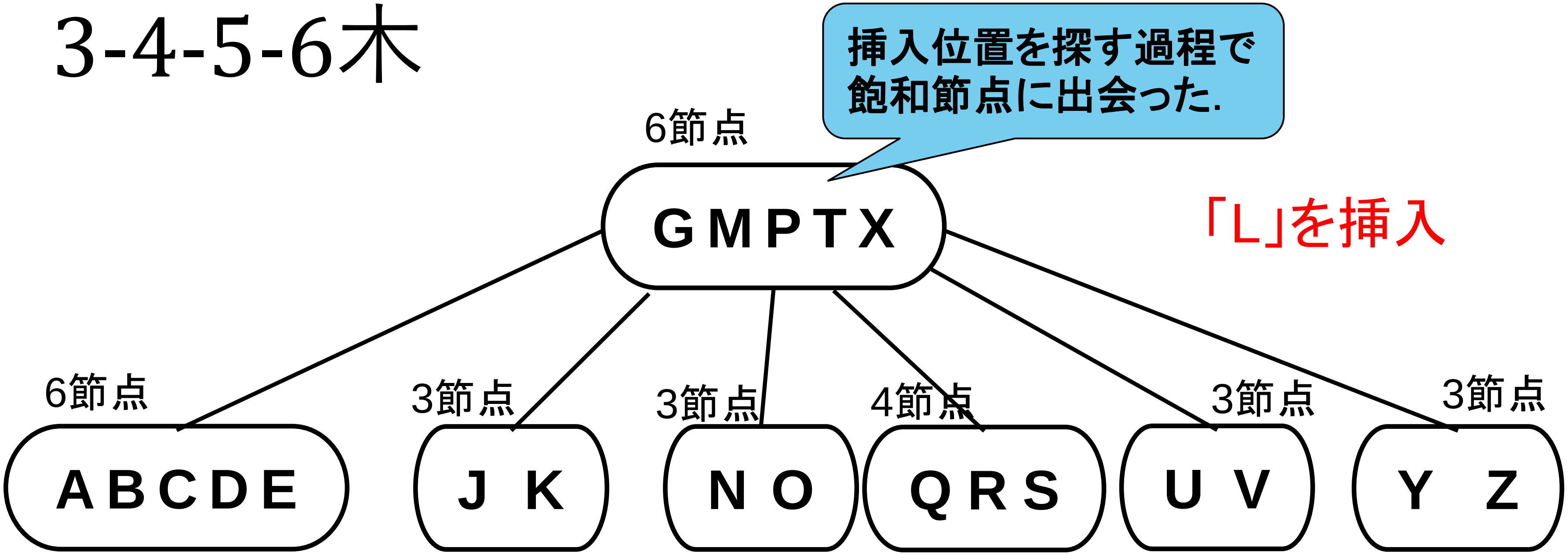
※ 外部節点への枝は省略

3-4-5-6木



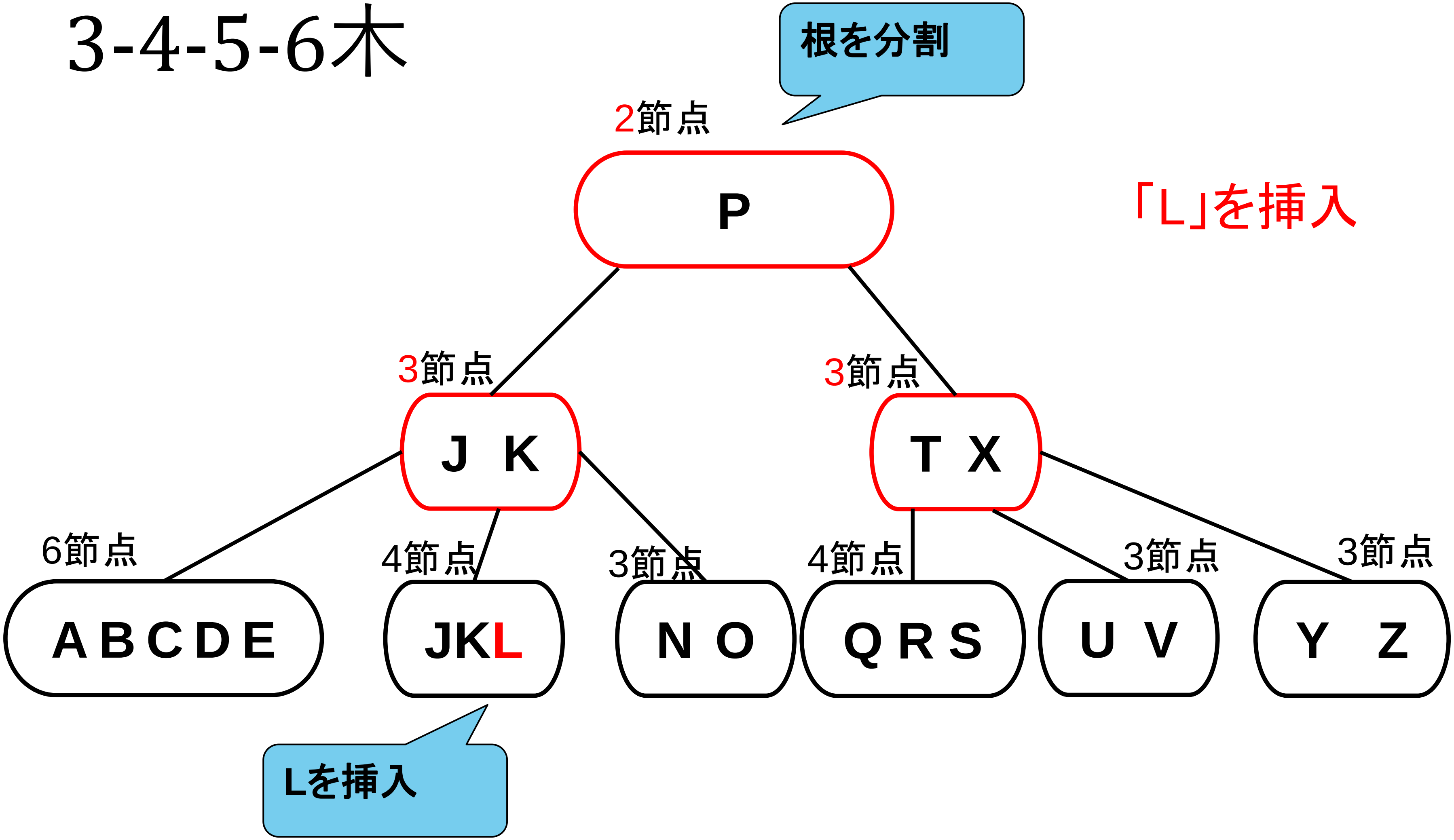
※ 外部節点への枝は省略

3-4-5-6木



※ 外部節点への枝は省略

3-4-5-6木



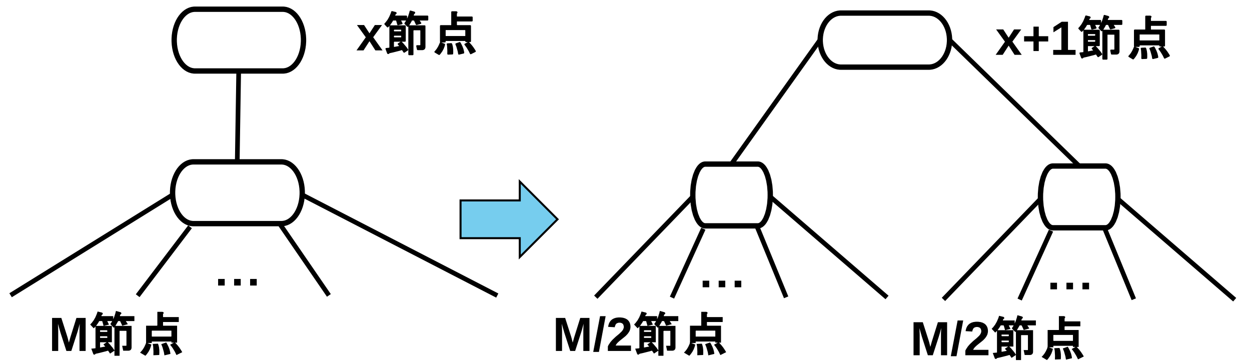
※ 外部節点への枝は省略

B木の性質

- 性質16.2
  - N個の項目を持つM次のB木では，探索と挿入は $\log_M N$から$\log_{M/2} N$ の回数の探査を実行する．
    - B木において，根と葉以外の内部節点は，M/2からMまでの個数の子節点を持つ．
      - 内部節点はM個の子節点をもつ節点が分割されるときに生成され，その大きさはM/2．子節点が分割されたときにその親の節点数が一つ増えるため．
    - そのため，最良の場合は次数Mの完全木となるときであり，最悪の場合は次数M/2の完全木となるときであることから性質16.2が導かれる．

実用的には定数回と遜色ない．

B木の性質



- 性質16.2
  - N個の項目を持つM次のB木では，探索と挿入は$\log_M N$から$\log_{M/2} N$の回数の探査を実行する．
    - B木において，根と葉以外の内部節点は，M/2からMまでの個数の子節点を持つ．
      - 内部節点はM個の子節点をもつ節点が分割されるときに生成され，その大きさはM/2．子節点が分割されたときにその親の節点数が一つ増えるため．
    - そのため，最良の場合は次数Mの完全木となるときであり，最悪の場合は次数M/2の完全木となるときであることから性質16.2が導かれる．

実用的には定数回と遜色ない．

項目の削除

- 項目の削除は，多くの場合は葉からキーと項目への参照を除去するのみ．ただし，節点内の項目数がM/2以上であることを保証するために，兄弟節点から節点を移動させる，または，兄弟節点の節点数もM/2である場合は節点をマージする作業が必要となる．

キーkの削除の手順

- kが内部節点x（x＝葉）に存在する場合はkを削除.
- kが内部節点x（x ≠葉）に存在する場合
  - kの直前／直後の子節点のうち(M/2+1)-節点以上の節点を根とするいずれかの部分木から，直前／直後のキーk'を削除．kをk'に置き換える.
  - kの直前／直後の子節点の節点数がいずれも(M/2)-節点の場合，直前の子節点にkを移動し，直後の子節点も直前の子節点にマージしたのち，直前の子節点を根とする部分木からkを削除.

※ ここでは，Mは偶数であることを仮定する.

キーkの削除の手順（続き）

- kが内部節点x（x ≠葉） に存在しない場合, kを含む部分木の根yを特定する． この時, 部分木の根が(M/2+1)-節点以上となることを保証するために, 以下を実行する.
  - yが(M/2)-節点であっても, 隣り合う兄弟に(M/2+1)-節点以上の節点zがある場合は, xから一つキーをyに移し, zからxにキーを一つ移す.
  - yとその隣り合う兄弟が全て(M/2)-節点の場合, yをいずれかの兄弟とマージする.

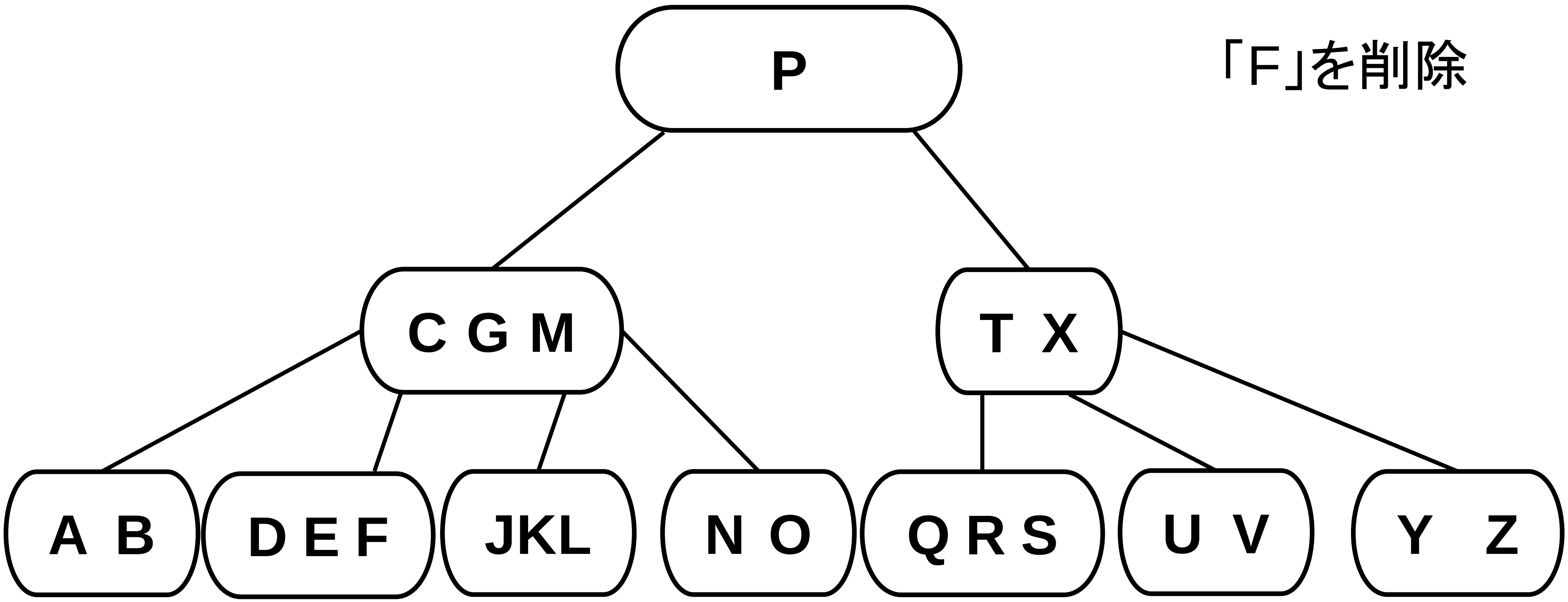
※ ここでは, Mは偶数であることを仮定する.

キーkの削除の手順

- kが内部節点x（x＝葉）に存在する場合はkを削除.
- kが内部節点x（x ≠葉）に存在する場合
  - kの直前／直後の子節点のうち(M/2+1)-節点以上の節点を根とするいずれかの部分木から，直前／直後のキーk’を削除．k をk’に置き換える.
  - kの直前／直後の子節点の節点数がいずれも(M/2)-節点の場合，直前の子節点にkを移動し，直後の子節点も直前の子節点にマージしたのち，直前の子節点を根とする部分木からkを削除.

※ ここでは，Mは偶数であることを仮定する.

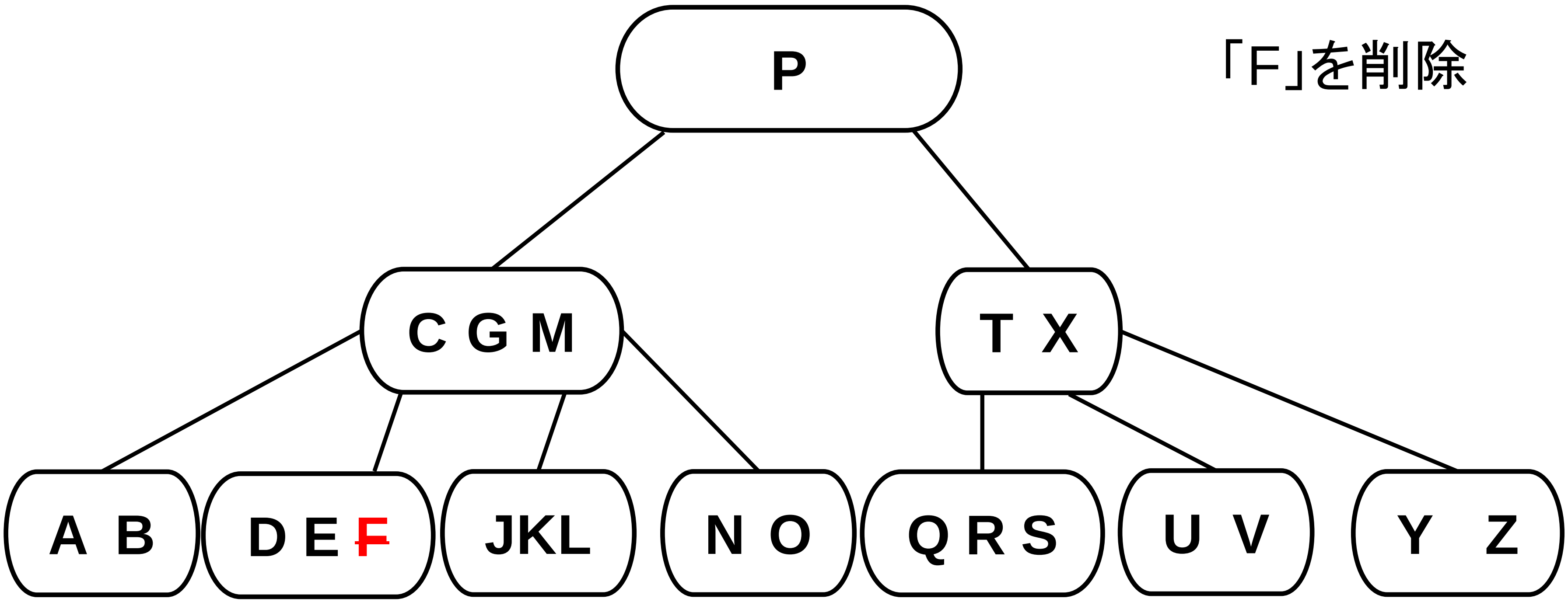
3-4-5-6木



「F」を削除

※ 外部節点への枝は省略

3-4-5-6木



「F」を削除

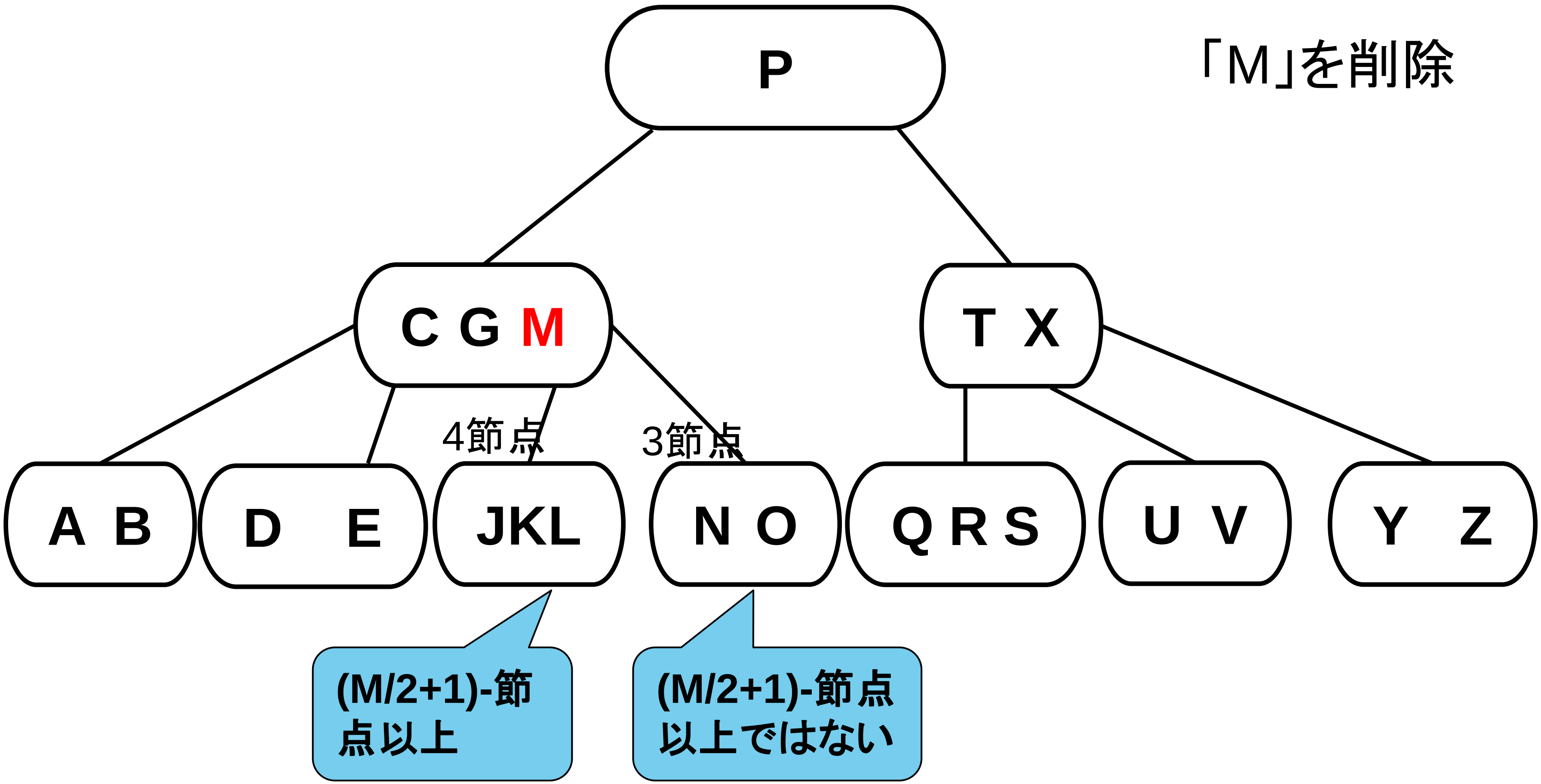
※ 外部節点への枝は省略

キーkの削除の手順

- kが内部節点x（x＝葉）に存在する場合はkを削除.
- kが内部節点x（x ≠葉）に存在する場合
  - kの直前／直後の子節点のうち(M/2+1)-節点以上の節点を根とするいずれかの部分木から，直前／直後のキーk’を削除．kをk’に置き換える.
  - kの直前／直後の子節点の節点数がいずれも(M/2)-節点の場合，直前の子節点にkを移動し，直後の子節点も直前の子節点にマージしたのち，直前の子節点を根とする部分木からkを削除.

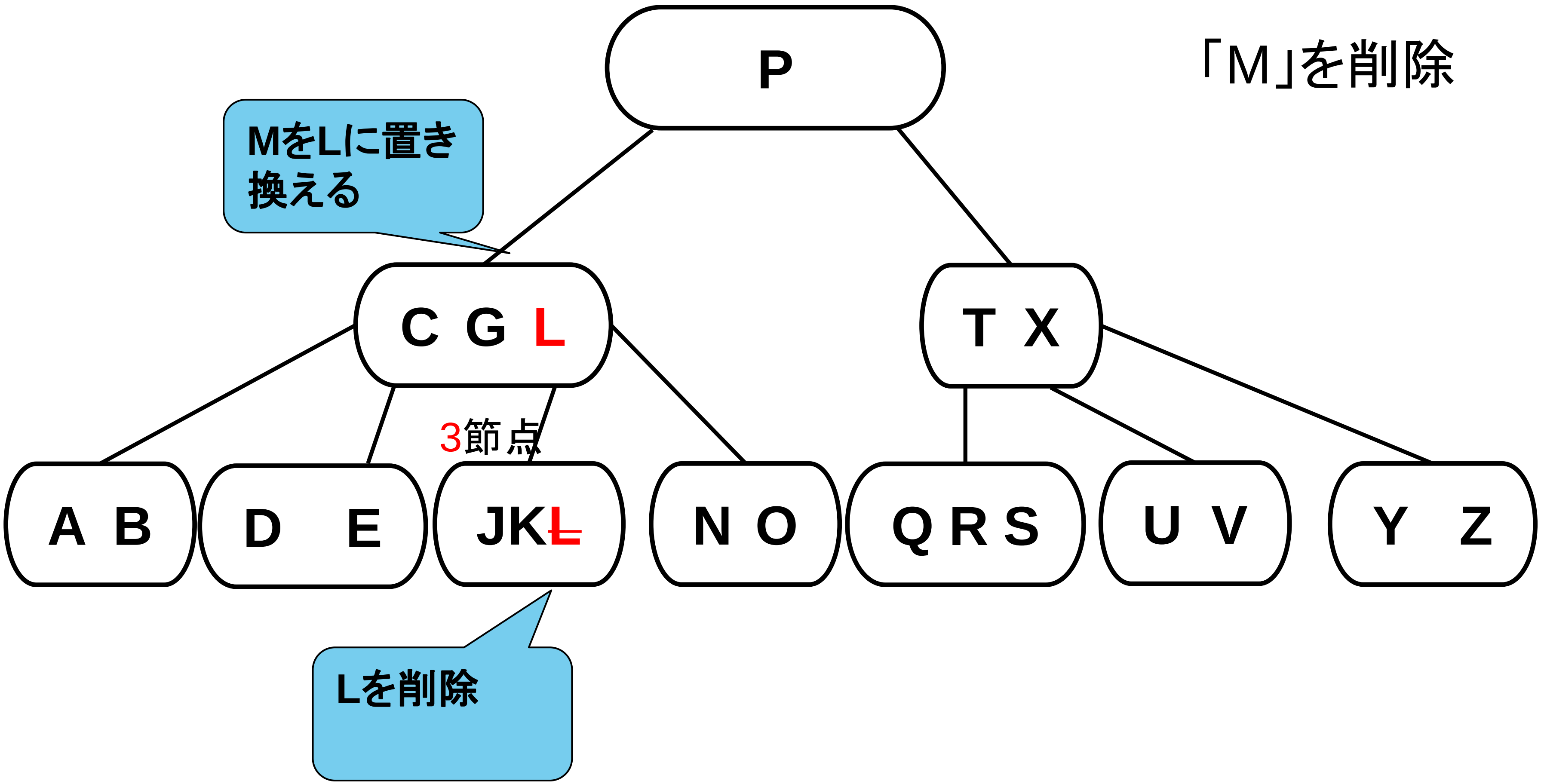
※ ここでは，Mは偶数であることを仮定する.

3-4-5-6木



※ 外部節点への枝は省略

3-4-5-6木



※ 外部節点への枝は省略

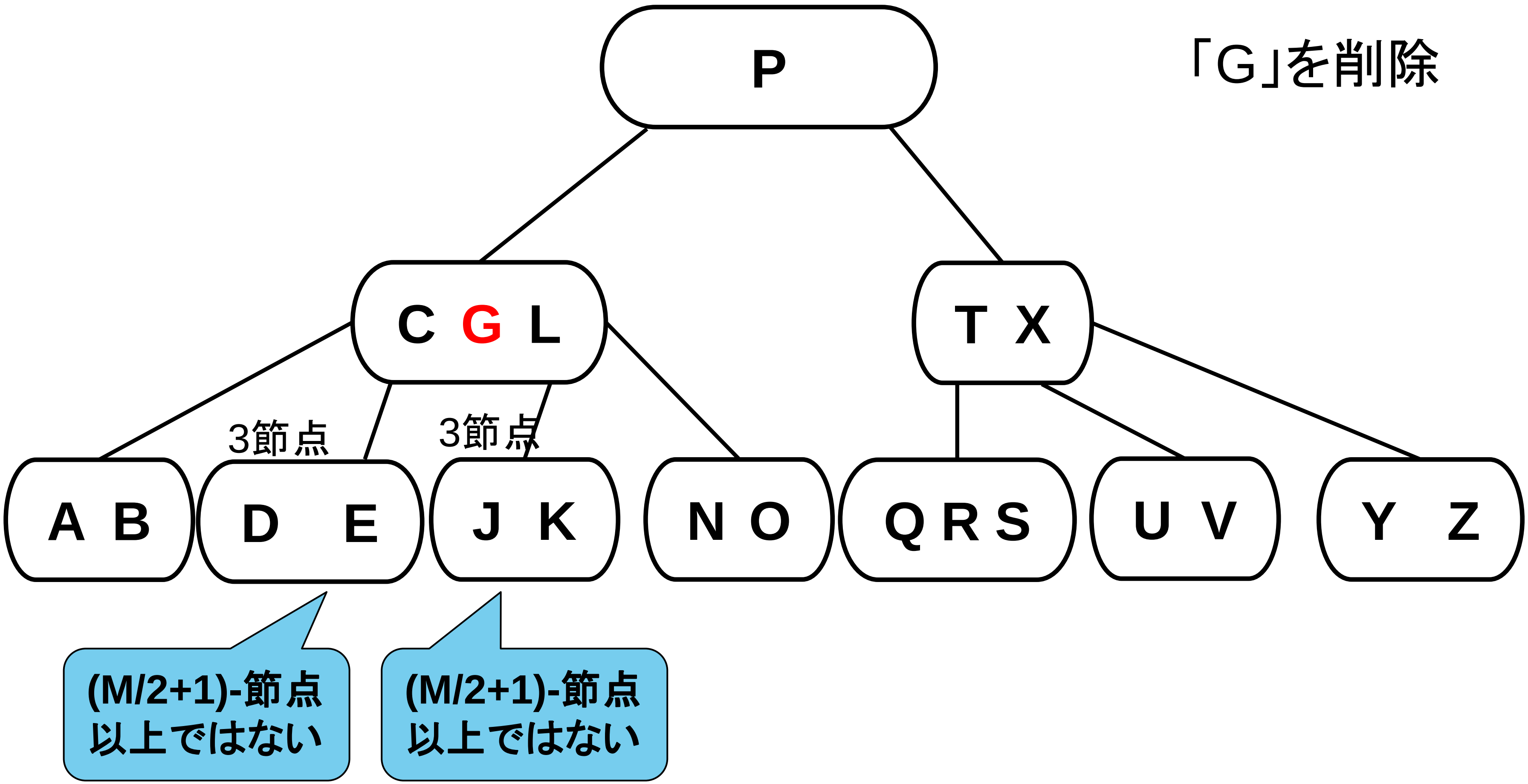
キーkの削除の手順

- kが内部節点x（x＝葉）に存在する場合はkを削除.
- kが内部節点x（x ≠葉）に存在する場合
  - kの直前／直後の子節点のうち(M/2+1)-節点以上の節点を根とするいずれかの部分木から，直前／直後のキーk’を削除．kをk’に置き換える.
  - kの直前／直後の子節点の節点数がいずれも(M/2)-節点の場合，直前の子節点にkを移動し，直後の子節点も直前の子節点にマージしたのち，直前の子節点を根とする部分木からkを削除.

※ ここでは，Mは偶数であることを仮定する.

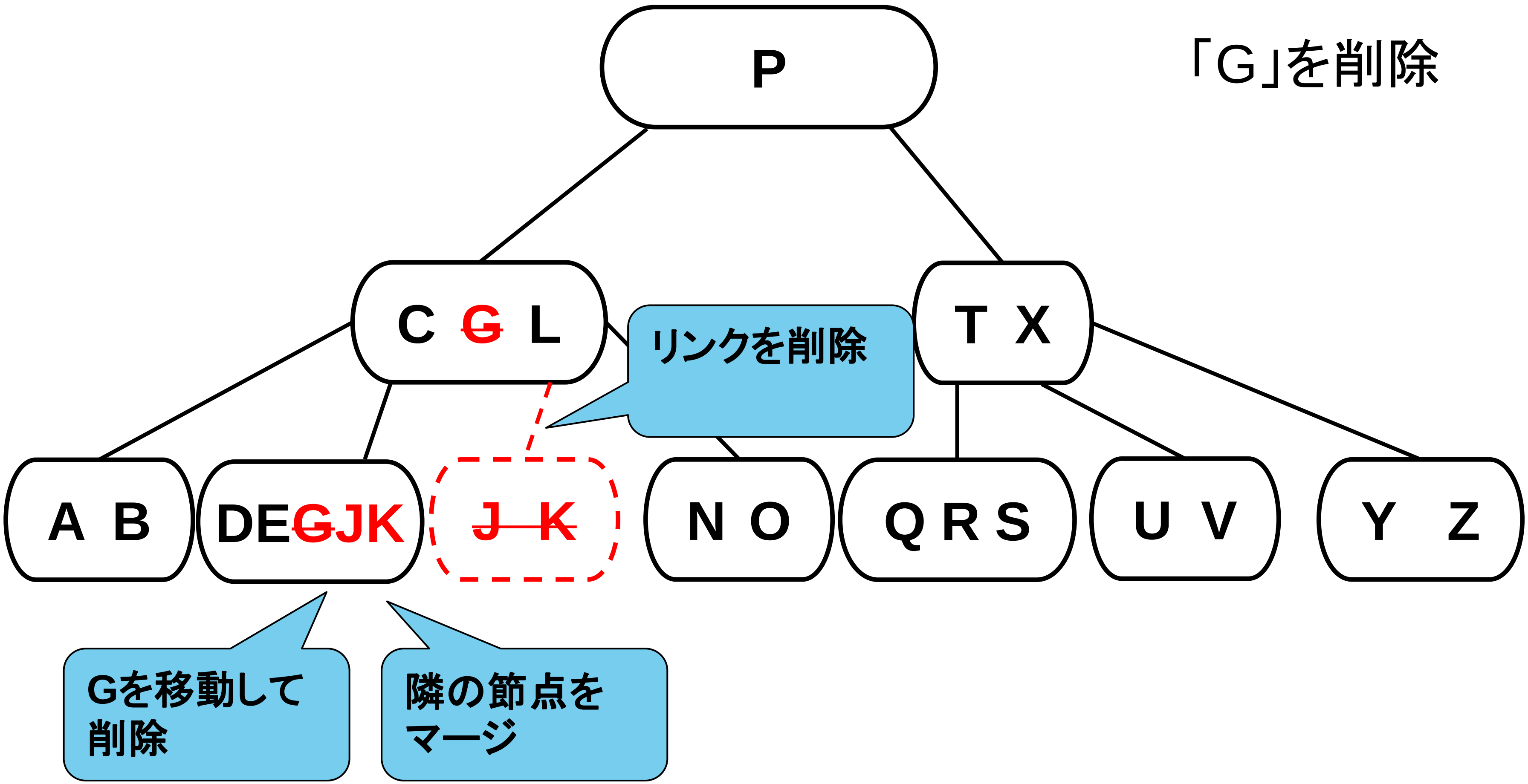
3-4-5-6木

「G」を削除



※ 外部節点への枝は省略

3-4-5-6木



※ 外部節点への枝は省略

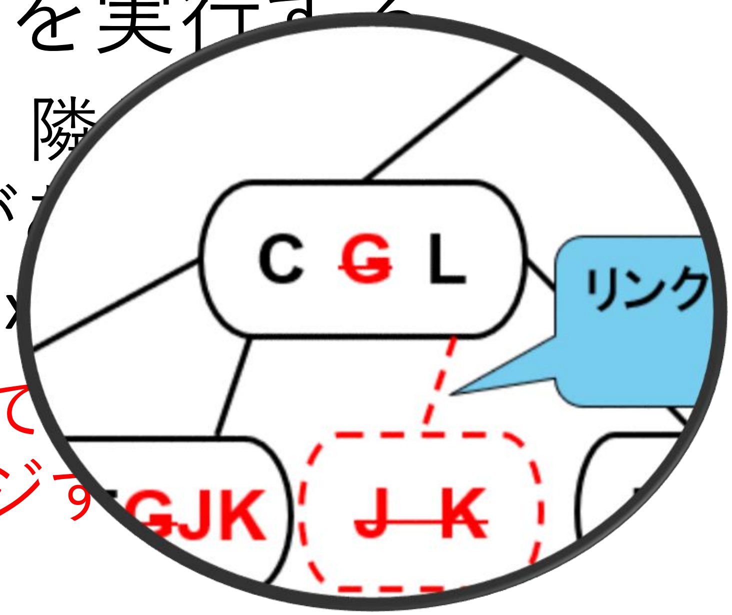
キーkの削除の手順（続き）

- kが内部節点x（x ≠葉） に存在しない場合，kを含む部分木の根yを特定する．この時，部分木の根が(M/2+1)-節点以上となることを保証するために，以下を実行する．
  - yが(M/2)-節点であっても，隣り合う兄弟に(M/2+1)-節点以上の節点zがある場合は，xから一つキーをyに移し，zからxにキーを一つ移す．
  - yとその隣り合う兄弟が全て(M/2)-節点の場合，yをいずれかの兄弟とマージする．

※ ここでは，Mは偶数であることを仮定する．

キーkの削除の手順（続き）

- kが内部節点x（x ≠葉） に存在しない場合，kを含む部分木の根yを特定する． この時，部分木の根が(M/2+1)-節点以上となることを保証するために， 以下を実行する
  - yが(M/2)-節点であっても， 隣
    (M/2+1)-節点以上の節点zがあ
    一つキーをyに移し，zからx
  - yとその隣り合う兄弟が全て
    yをいずれかの兄弟とマージす



※ ここでは，Mは偶数であることを仮定する．

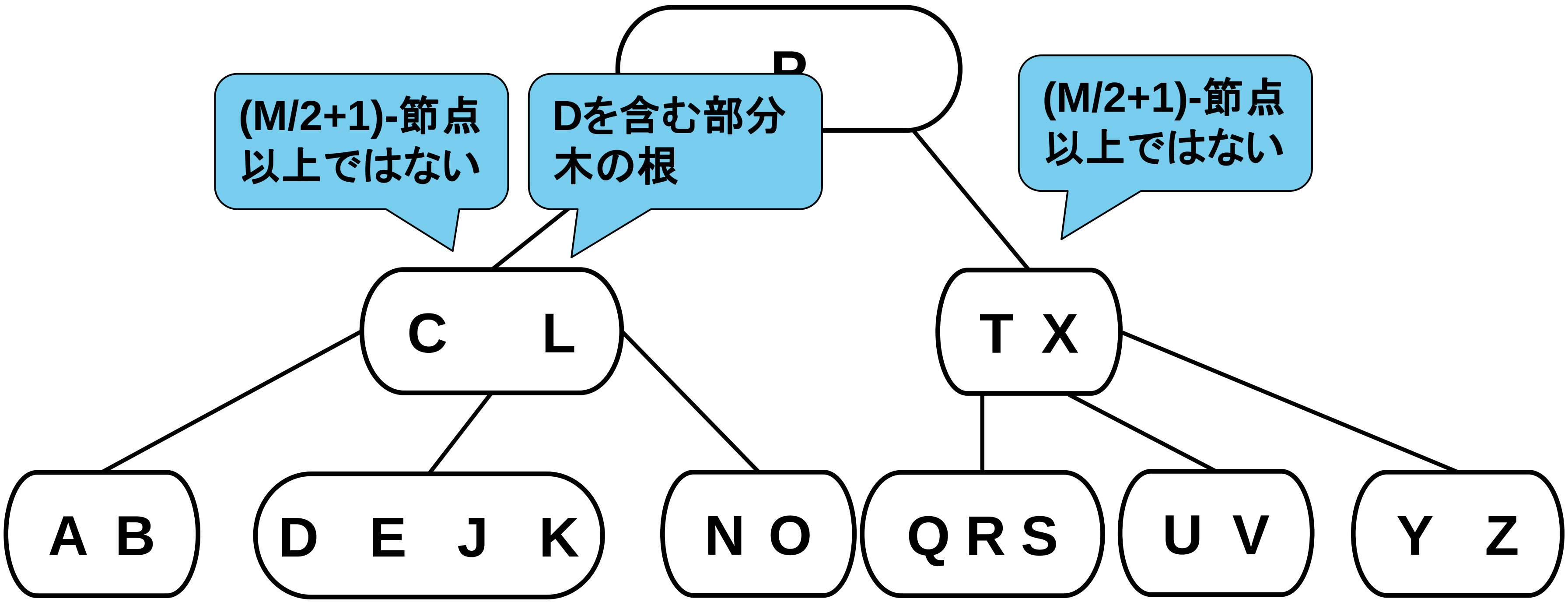
キーkの削除の手順（続き）

- kが内部節点x（x ≠葉） に存在しない場合，kを含む部分木の根yを特定する． この時，部分木の根が(M/2+1)-節点以上となることを保証するために， 以下を実行する．
  - yが(M/2)-節点であっても， 隣り合う兄弟に(M/2+1)-節点以上の節点zがある場合は， xから一つキーをyに移し， zからxにキーを一つ移す．
  - yとその隣り合う兄弟が全て(M/2)-節点の場合，yをいずれかの兄弟とマージする．

※ ここでは，Mは偶数であることを仮定する．

3-4-5-6木

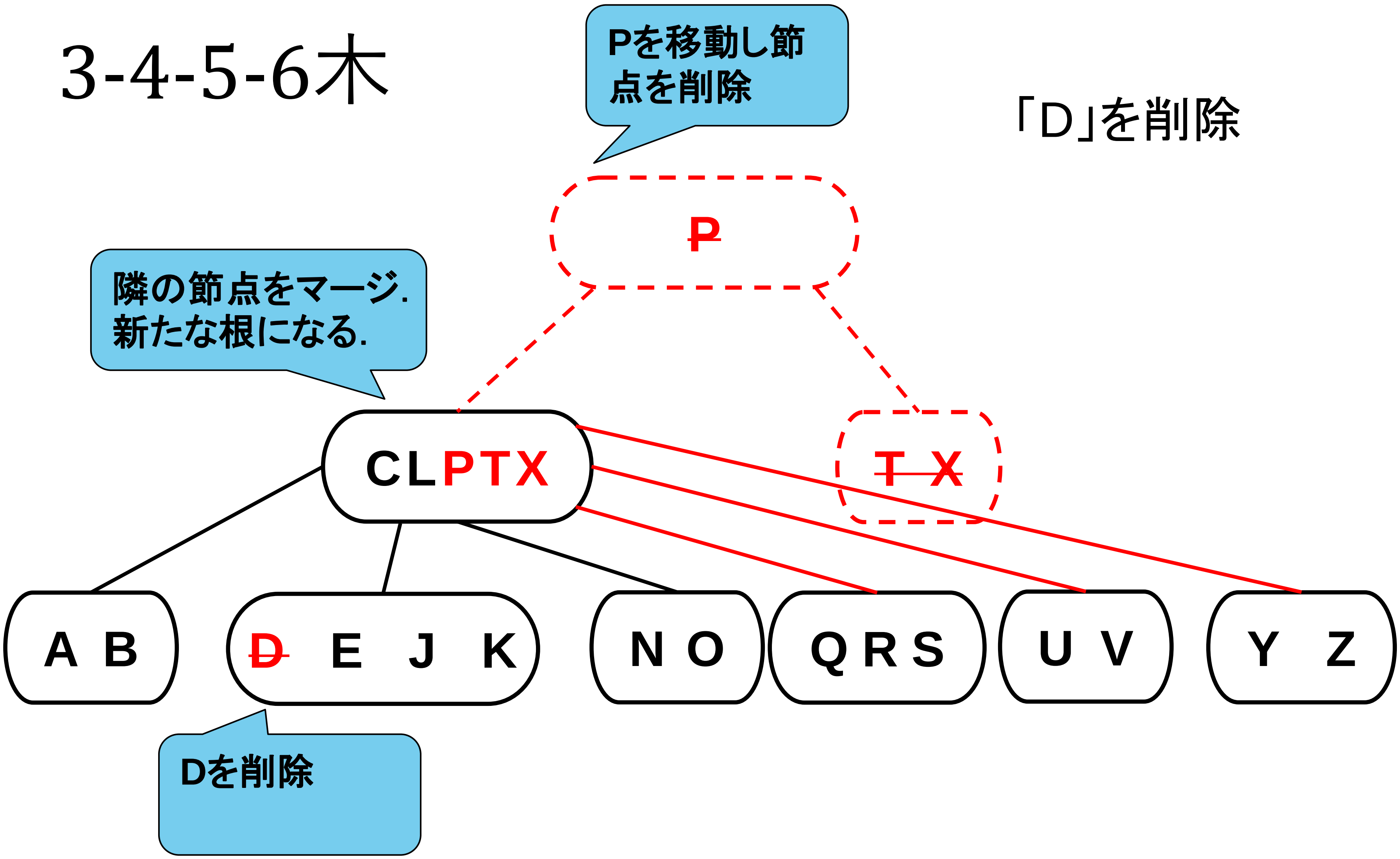
「D」を削除



※ 外部節点への枝は省略

3-4-5-6木

「D」を削除



※ 外部節点への枝は省略

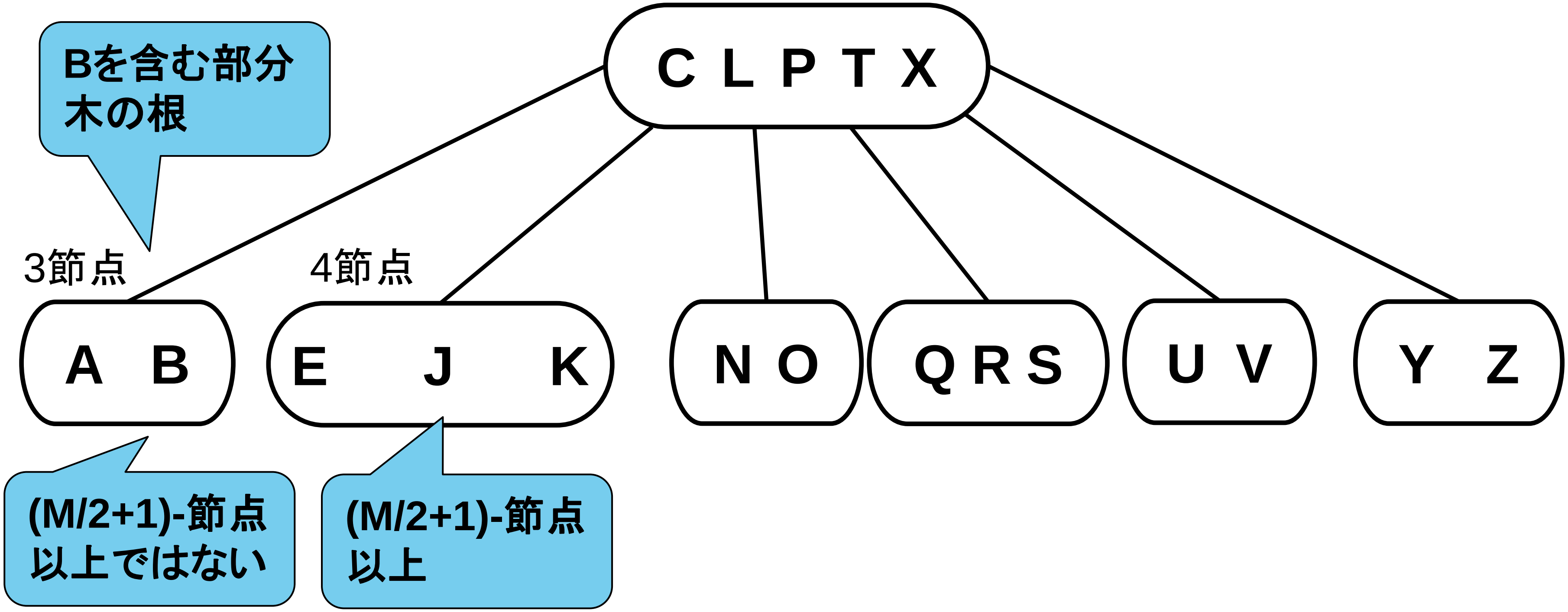
キーkの削除の手順（続き）

- kが内部節点x（x ≠葉） に存在しない場合, kを含む部分木の根yを特定する． この時, 部分木の根が(M/2+1)-節点以上となることを保証するために, 以下を実行する．
  - yが(M/2)-節点であっても， 隣り合う兄弟に(M/2+1)-節点以上の節点zがある場合は， xから一つキーをyに移し， zからxにキーを一つ移す．
  - yとその隣り合う兄弟が全て(M/2)-節点の場合, yをいずれかの兄弟とマージする．

※ ここでは, Mは偶数であることを仮定する.

3-4-5-6木

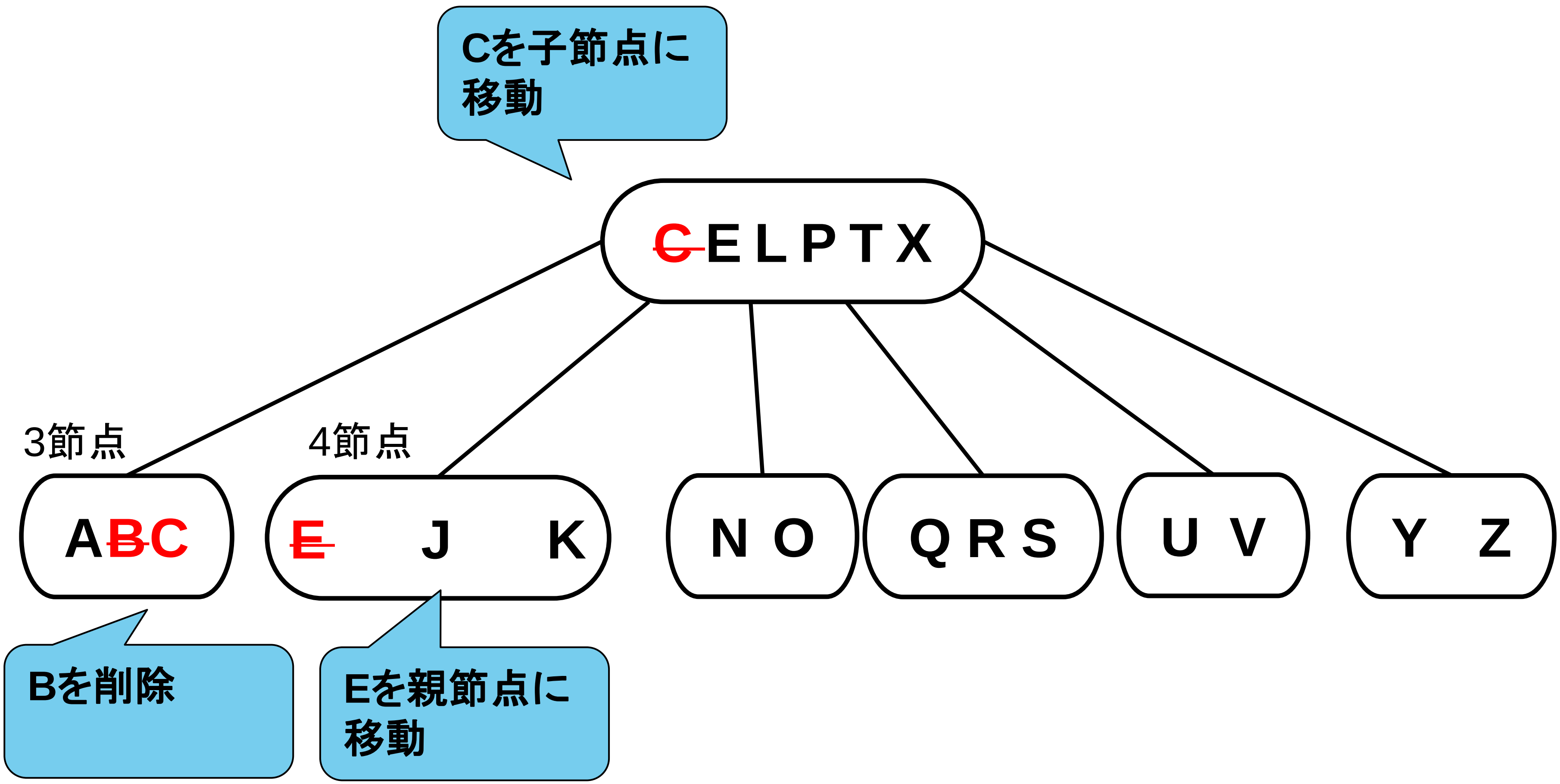
「B」を削除



※ 外部節点への枝は省略

3-4-5-6木

「B」を削除



※ 外部節点への枝は省略

B木の変種

- B*木
  - B木で生じる節点内の無駄な空間を削減する方式．以下により各節点のキーの数を増やす．
    - 飽和節点xと隣り合う兄弟yが(M-2)-節点以下の時，xとyに含まれるキーを均等に割り当てなおす．
    - 兄弟節点yが(M-1)-節点以上であれば，新たに節点zを生成し，xとyのキーを均等にx, y, zに割り振る．
    - 根では4M/3程度のキーを持つことを許す．

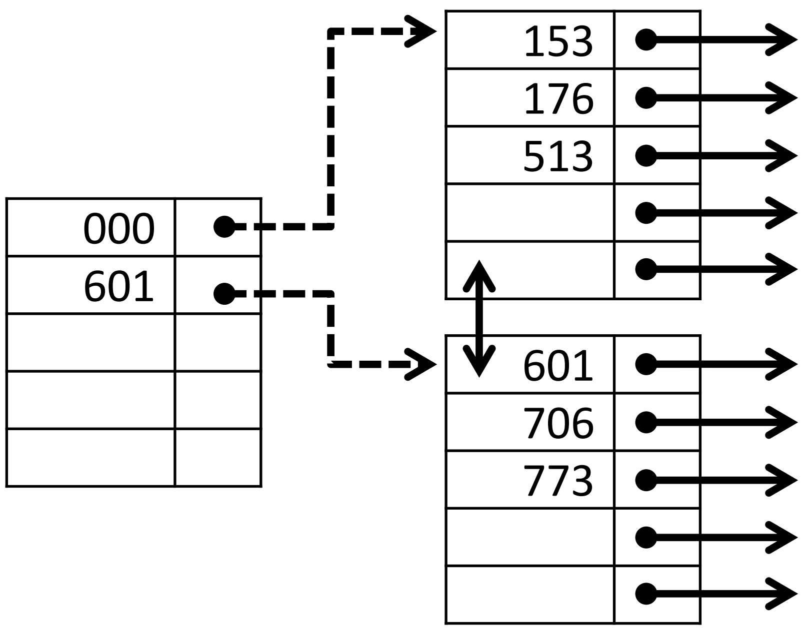
B木の変種

- B+木
  - 全項目（もしくは項目への参照）を葉におき，内部節点では子節点へのポインタのみを持つ．
    - 各節点に含められるキーの数を増やせるため，木の高さを低くできる．
  - 葉同士をつなぐポインタを設置する．
    - 範囲検索を高速化できる．

内部ページ／外部ページ

- 項目への参照を持つキーを木の底にある外部ページに保持し，ページへの参照を持つキーのコピーを内部ページに保持する．

M=5の木

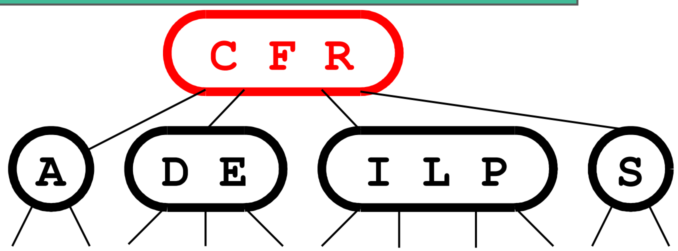


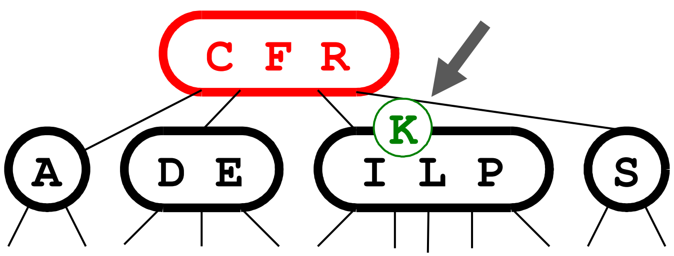
※ ページへのポインタは破線で示す

ボトムアップの挿入

「K」の挿入

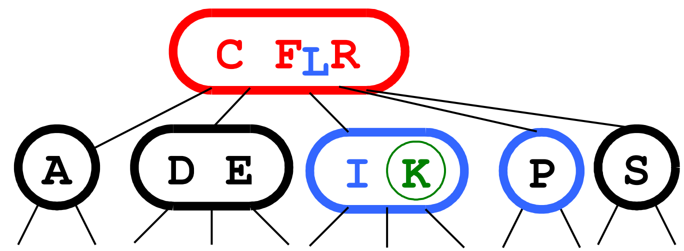
根から順に，「K」の挿入位置を探す



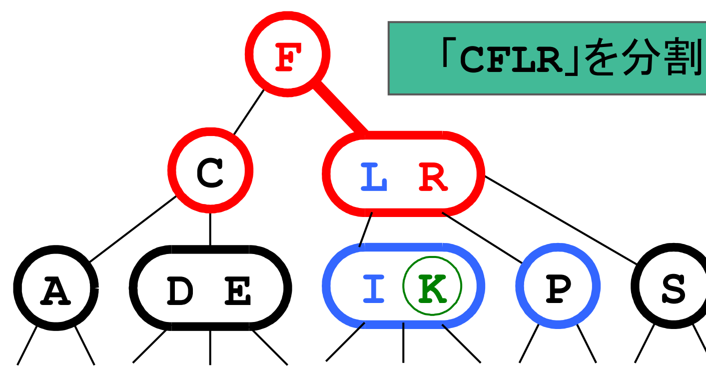


葉に到達．「K」を仮に挿入する．

「IKLP」を分割して，「L」を親節点に仮に押し上げる．



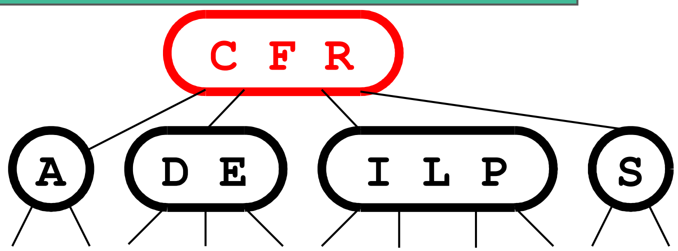
「CFLR」を分割

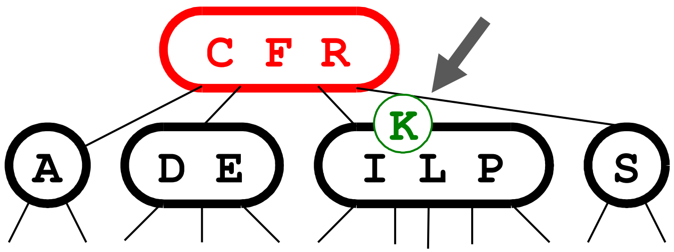


ボトムアップの挿入

「K」の挿入

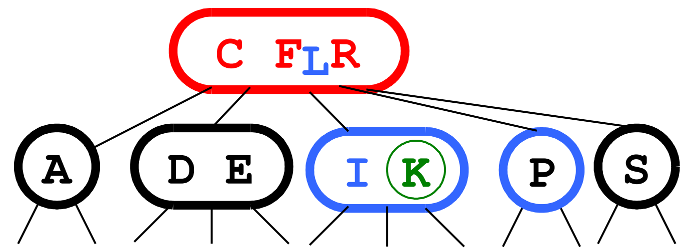
根から順に，「K」の挿入位置を探す



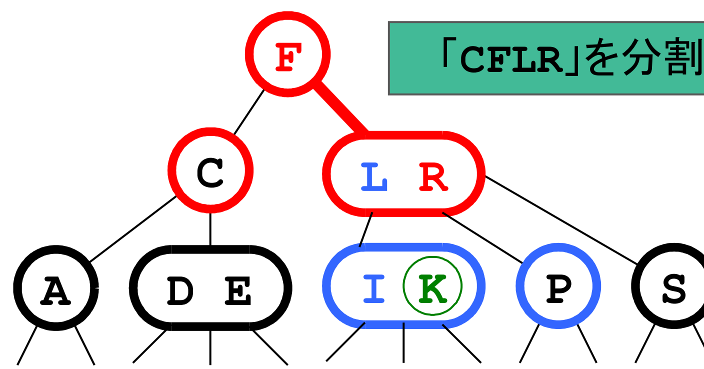


葉に到達．「K」を仮に挿入する.

「IKLP」を分割して，「L」を親節点に仮に押し上げる.



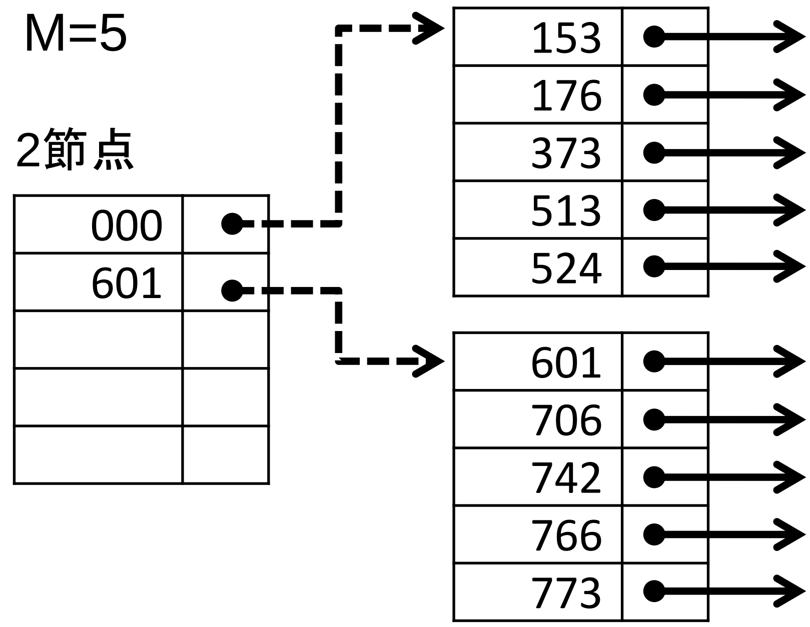
「CFLR」を分割



項目の挿入（ボトムアップ）

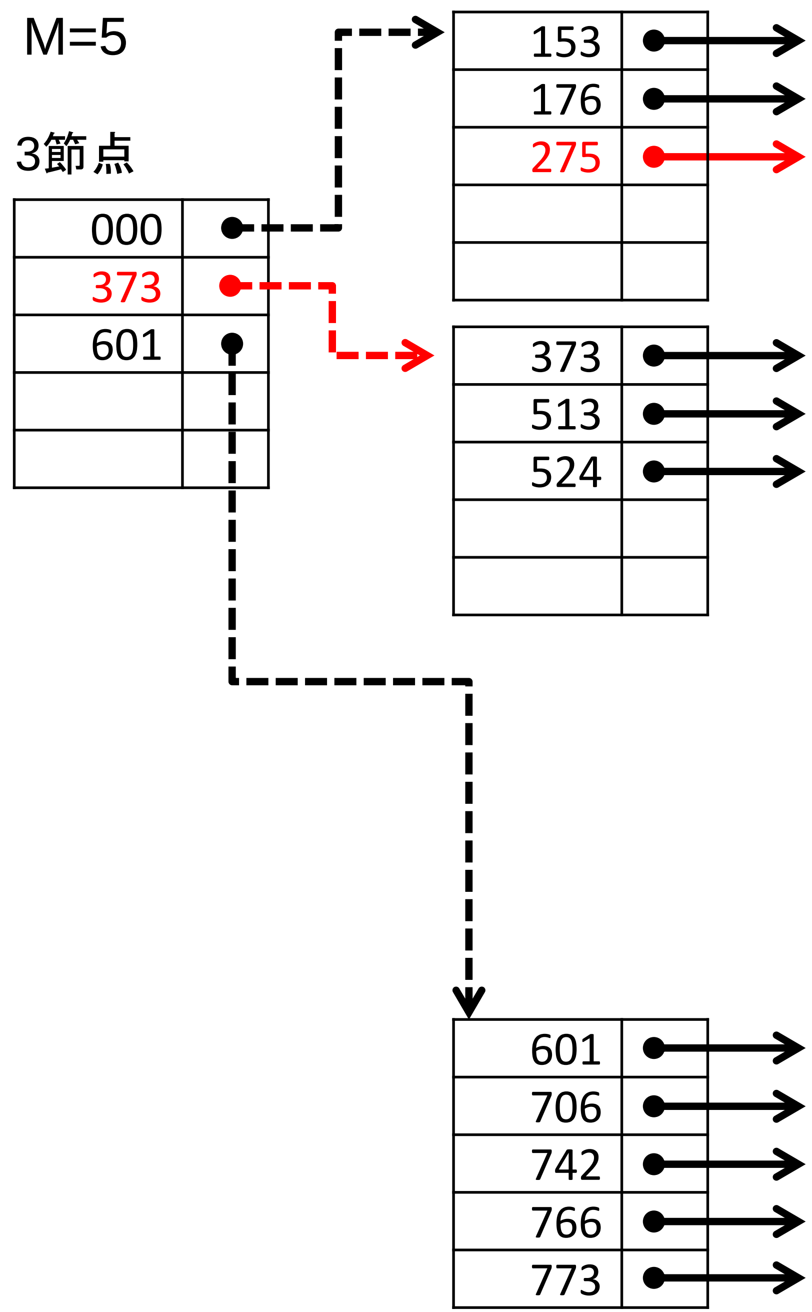
- 新たな項目への参照は，木の底に挿入するが，M/2-…-M木と同様の操作で索引を構築する．
- 索引を構築する際に，ページがM個の要素を持った場合，そのページをそれぞれM/2個の要素を持つ2つのページに分割し，新しいページへの参照を親節点に挿入する．
- 根の分割では2つの子節点をもつ新しい根を作るため，根のキーの数はM/2よりも小さくなることを許す必要がある．

「275」を追加



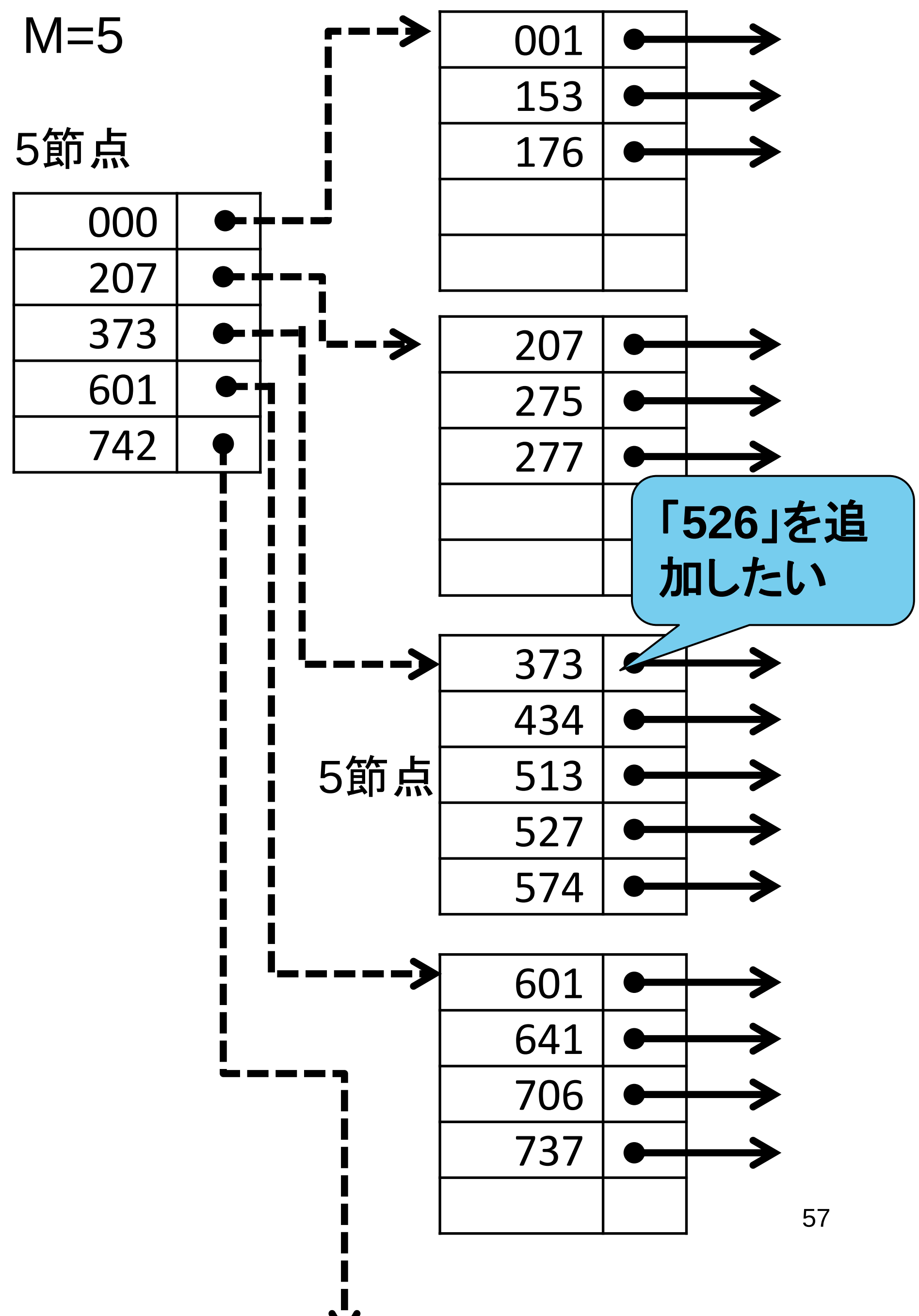
※ ページへのポインタは破線で示す

「275」を追加



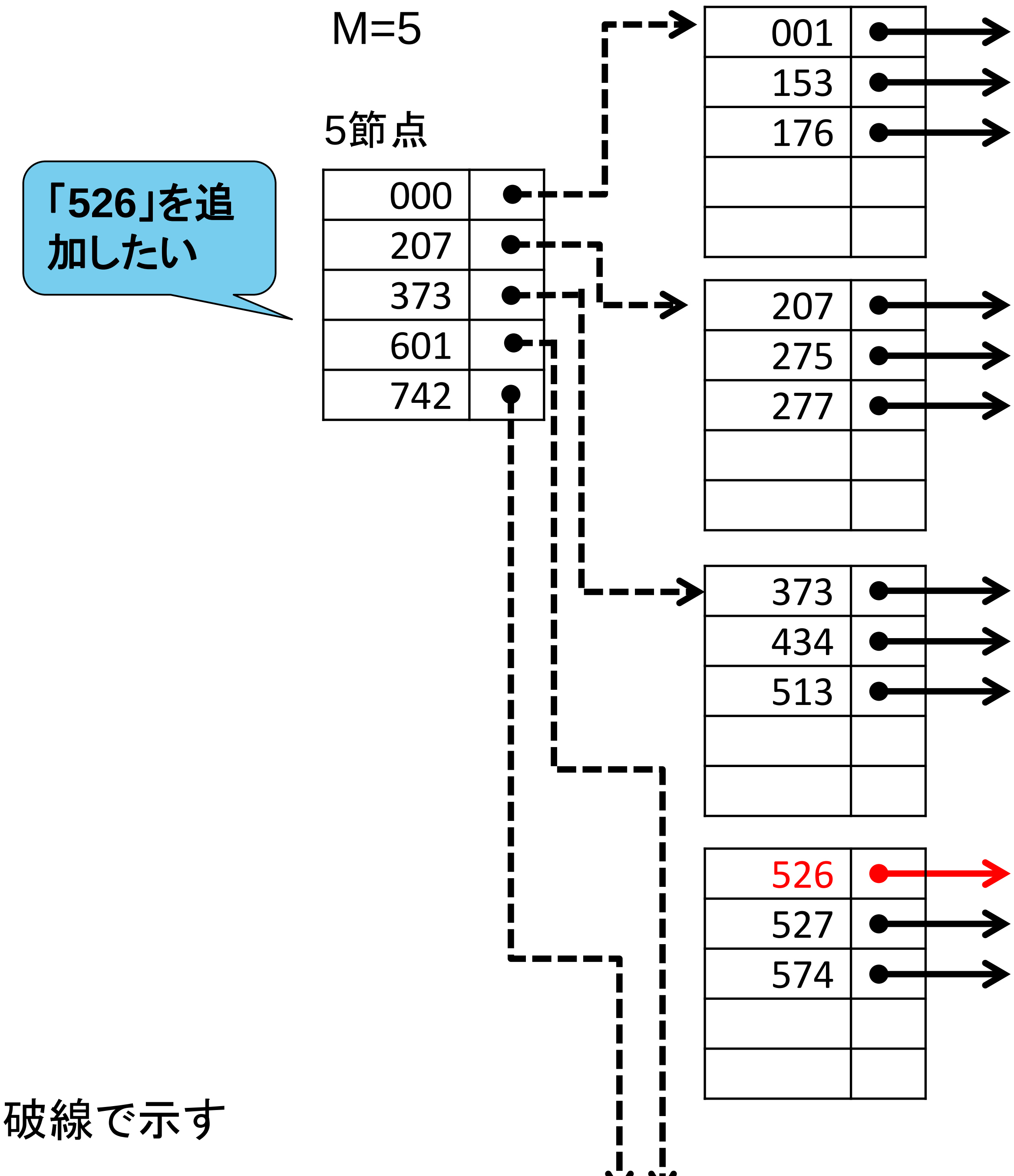
※ ページへのポインタは破線で示す

「526」を追加



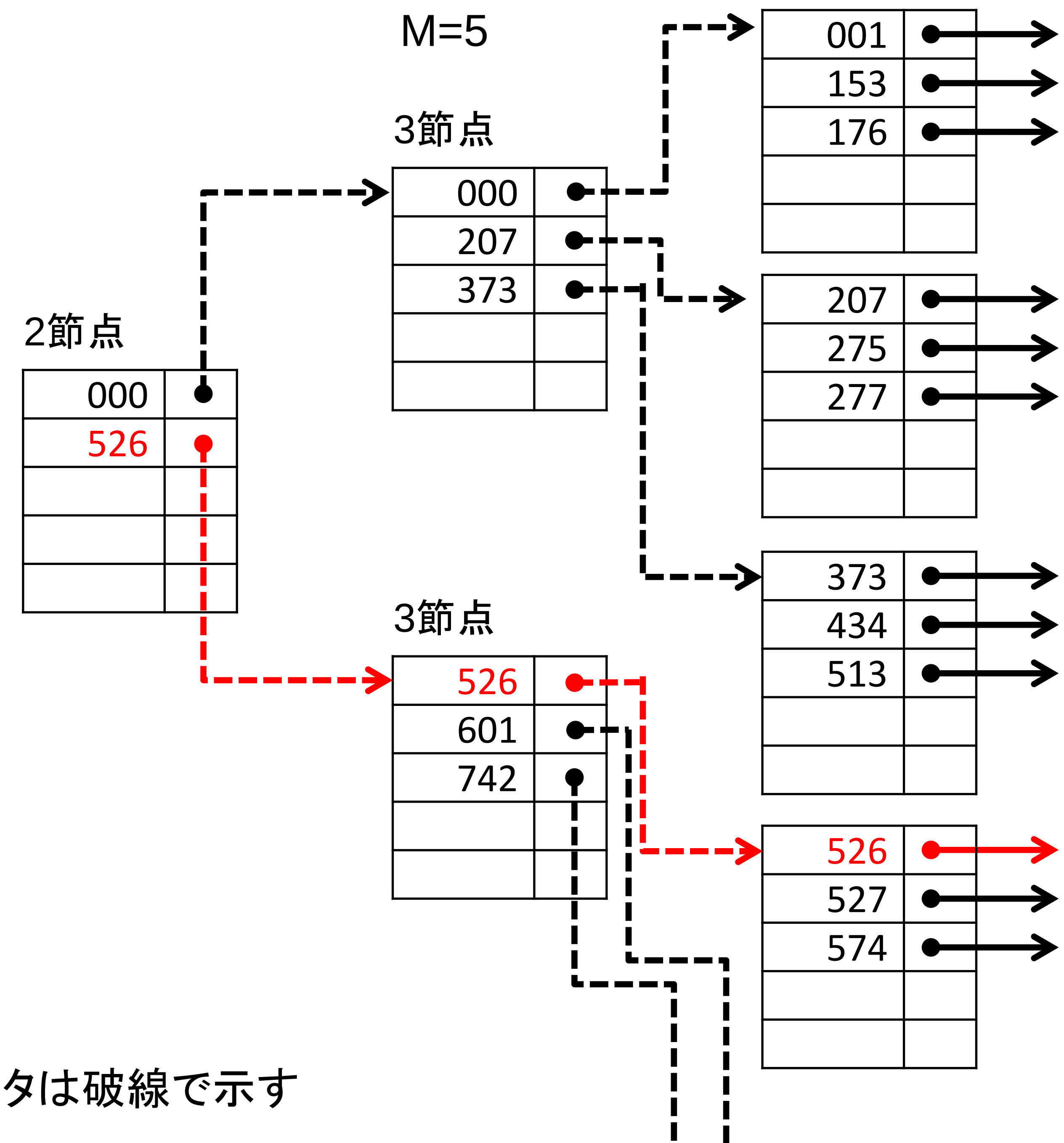
※ ページへのポインタは破線で示す

「526」を追加



※ ページへのポインタは破線で示す

「526」を追加



※ ページへのポインタは破線で示す

B木の実装

```
#define M 5
#define ws (M+1)

typedef struct STnode *link;

typedef struct
{Key key; union{link next; Item item;} ref;} entry;
struct STnode{entry b[ws]; int m;};


link NEW()
{
    link x = malloc(sizeof *x);
    x->m = 0;
    return x;
}

void STinit(int maxN){head = NEW(); H=0; N=0;}
```

各節点は最大M個のリンク／項目を持つ

葉の場合は項目，それ以外の内部節点の場合は子へのリンクを持つように共用体で宣言

※ このプログラムではMは奇数であることを想定している.

探索

```
Item searchR(link h, Key v, int H)
{   int j;
    if(H == 0)
      for(j = 0; j < h->m; j++)
        if(eq(v, h->b[j].key))
          return h->b[j].ref.item;
    if(H != 0)
      for(j = 0;j < h->m; j++)
        if((j+1 == h->m)||less(v, h->b[j+1].key))
          return searchR(h->b[j].ref.next, v, H-1);
    return NULLitem;
}
Item STsearch(Key v) { return searchR(head, v, H); }
```

葉の場合

項目

葉以外の内部節点の場合

次のページを指す
ポインタ

插入

```
void STinsert(Item item)
{
    link t, u = insertR(head, item, H);
    if(u == NULL) return;
    t = NEW(); t->m = 2;
    t->b[0].key = head->b[0].key;
    t->b[0].ref.next = head;
    t->b[1].key = u->b[0].key;
    t->b[1].ref.next = u;
    head = t; H++;
}
```

挿入
（続き）

```
link insertR(link h, Item item, int H)
{    int i, j; Key v = key(item); entry x; link t, u;
     x.key = v; x.ref.item = item;
     if(H == 0)
        for(j = 0;j < h->m; j++)
            if(less(v, h->b[j].key)) break;
     if(H != 0)
        for(j = 0;j < h->m; j++)
            if((j+1 == h->m)||less(v, h->b[j+1].key))
              {    t = h->b[j++].ref.next;
                   u = insertR(t, item, H-1);
                   if(u == NULL) return NULL;
                   x.key = u->b[0].key; x.ref.next = u;
                   break;}
     for(i = (h->m)++; i > j; i--)
        h->b[i] = h->b[i-1];
     h->b[j] = x;
     if(h->m < ws) return NULL; else return split(h);
}
```

葉の場合

挿入個所（j番目）を探す

葉以外の内部節点の場合

j+1番目の子を根とする部分木に挿入項目を挿入

j+1番目の子を根とする部分木に挿入項目を挿入

j番目以降の要素を右にシフト

j番目に挿入

挿入
（続き）

```
link insertR(link h, Item item, int H)
{   int i, j; Key v = key(item); entry x; link t, u;
    x.key = v; x.ref.item = item;
    if(H == 0)
      for(j = 0;j < h->m; j++)
         if(less(v, h->b[j].key)) break;
    if(H != 0)
      for(j = 0;j < h->m; j++)
         if((j+1 == h->m)||less(v, h->b[j+1].key))
           {   t = h->b[j++].ref.next;
               u = insertR(t, item, H-1);
               if(u == NULL) return NULL;
               x.key = u->b[0].key; x.ref.next = u;
               break;}
    for(i = (h->m)++; i > j; i--)
      h->b[i] = h->b[i-1];
    h->b[j] = x;
    if(h->m < ws) return NULL; else return split(h);
}
```

葉の場合

挿入個所（j番目）を探す

葉以外の内部節点の場合

j+1番目の子を根とする部分木に挿入項目を挿入

j+1番目の子を根とする部分木に挿入項目を挿入

j番目以降の要素を右にシフト

j番目に挿入

挿入（続き）

```
link split(link h)
{
    int j; link t = NEW();
    for(j = 0; j < ws/2; j++)
      t->b[j] = h->b[ws/2+j];
    h->m = ws/2; t->m = ws/2;
    return t;
}
```

新たに節点を作る

元の節点の後半を新たに作った節点にコピー

新たに作った節点へのリンクを返す

目次

- 索引順アクセス
- B木

演習1（必須課題）

- ST_BST3.cおよび関連するプログラムをコンパイルし，動作を確認せよ．
- 提出不要

演習2（加点課題）

- B木に含まれるページ数と項目数を表示するプログラムを実装せよ．

- 提出
  - 記号表にランダム，あるいは何らかの偏りを持たせて項目を挿入し，B木のメモリ効率（木に含まれるページが保持できる最大の項目数に対する実際の項目数の割合）について考察したレポートを提出せよ．Mを様々な値に変化させたり，木の高さに関する考察もあるとなおよい．
  - ソースコードを添付ファイルとして提出すること．

演習3（加点課題）

- B木を表示するプログラムを実装せよ．表示方法は各自で定義してよい．
- 提出
  - 記号表に20個の要素をランダムに挿入した場合，昇順に挿入した場合の両方について，レポート本文として提出すること．
  - レポート本文の冒頭に，各自で定義した木の表示フォーマットの説明を記載すること．
  - ソースコードを添付ファイルとして提出すること．

演習3（補足）

- 木の表示例
  - ‘|’で囲まれた数列がページ内のキー（項目）を示す．各行は同じレベルに位置する内部節点を昇順に表示する．（この表示方法では，親子のリンクを陽に示さないが，次の行において同じキーを含むページが子に対応する．）

昇順:
1, 10|
1, 4, 7|10, 13, 16|
1, 2, 3|4, 5, 6|7, 8, 9|10, 11, 12|13, 14, 15|16, 17, 18, 19, 20|

ランダム:

30886, 36915, 47793, 80540, 89383|
2362, 13926, 16649, 20059, 30886|36915, 38335, 41421|47793, 55736, 60492, 68690|80540, 83426, 85386, 89172|89383, 90027, 92777, 97763|