3. 多次元インデックス

（マルチメディアデータベース序論，全６回）

https://www.kkaneko.jp/cc/multimediadb/index.html

金子邦彦

空間データ

- 多次元の属性を持ったデータ

  (x, y), (x, y, z) など

- しばしば，位相，幾何の情報も扱わねばならない

空間データの種類

- 点／ベクトルデータ
  - 位置情報のみ（「領域」がない）

- 図形データ
  - 位置，領域がある

特徴量

- 静止画像　→　１つの「多次元ベクトル」
- 動画像　→　複数個の「多次元ベクトル」の並び

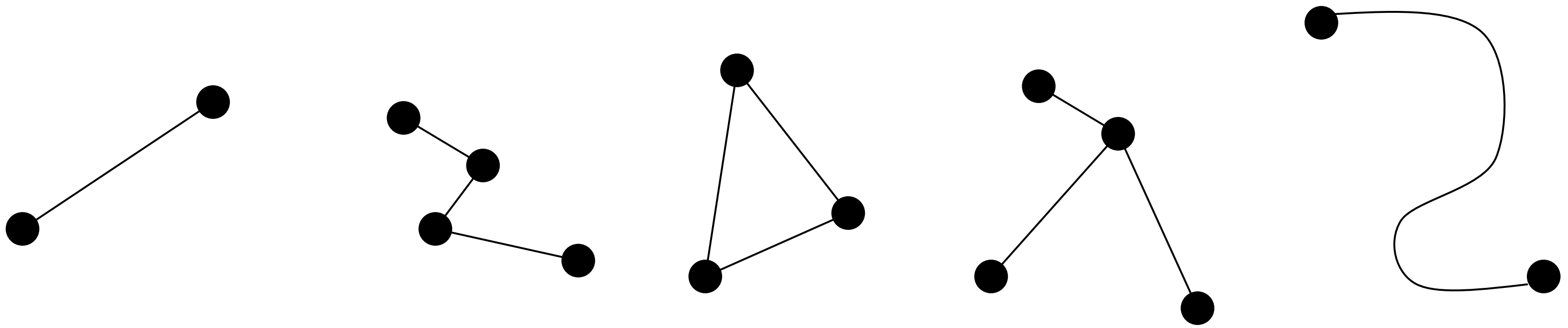
特徴量は「ベクトル」のデータ

図形データの種類

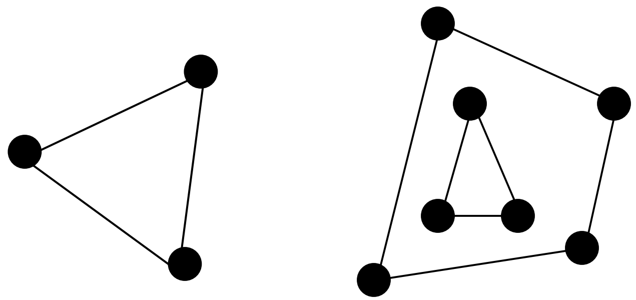
- 図形データは，２次元，３次元

点

線

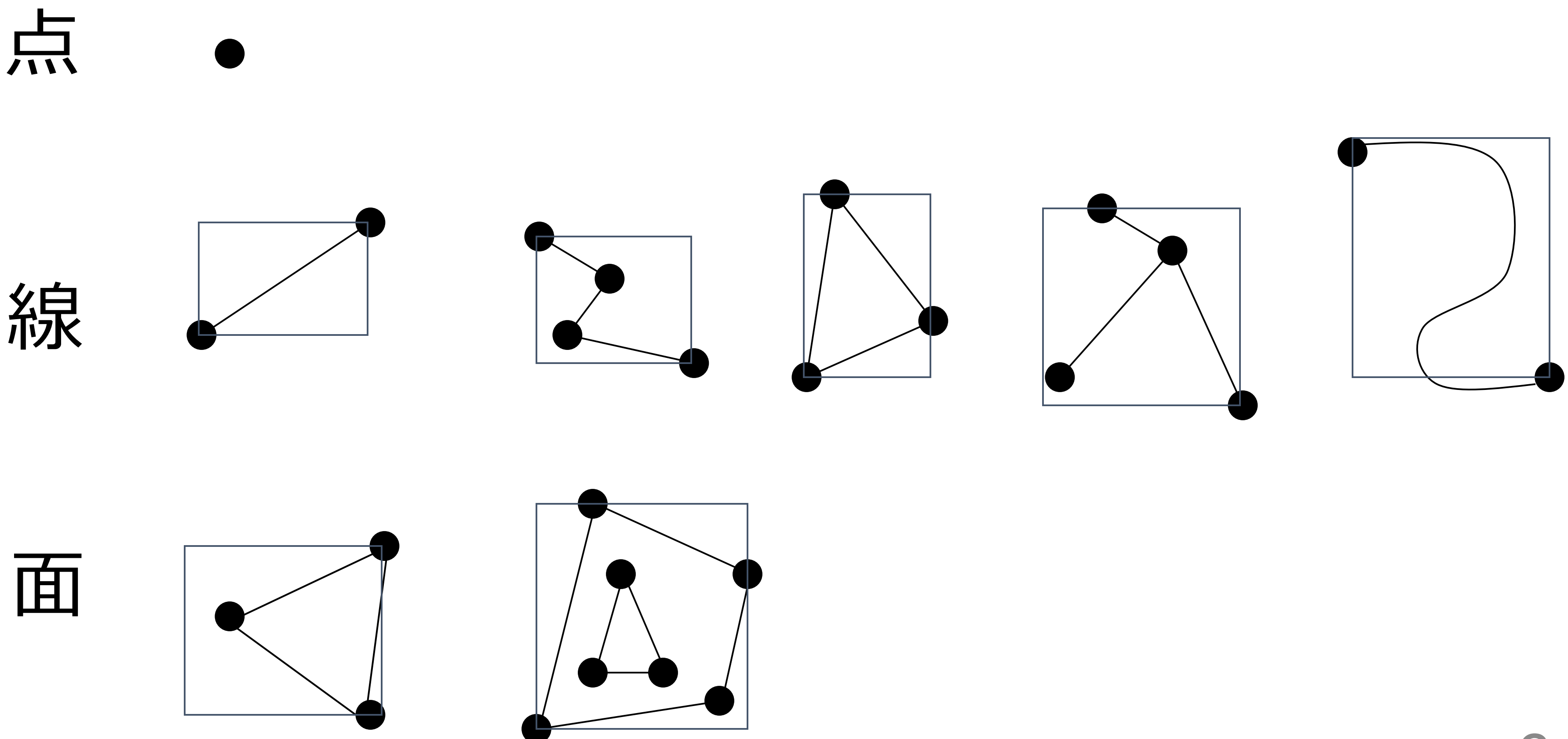


面



近似表現
-　図形データの場合　-

図形データは，近似表現（矩形など）し，
インデックスで管理



空間データ用のインデックスの種類

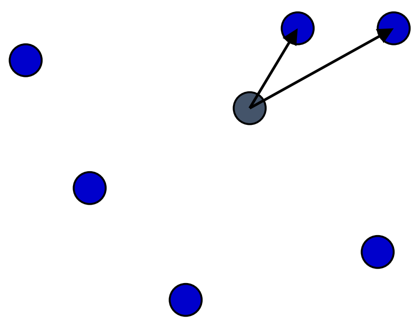
- ハッシュ
  - ある規則で，データを分割
  - Grid Files　：　メッシュ構造によるインデックス
  - Multidimensional Trie　：　Trie の拡張
  - Multikey hashing　：　ハッシュの拡張
- B-tree の利用
  - データを，「１次元空間」上にマッピング
  - 各種の空間充填曲線（Ｚ曲線など）を利用

空間データ用のインデックスの種類

- 木構造
  - データを階層的にクラスタ化
  - 古典的な多次元データ構造
    - 2n tree, k-d Tree, MX-Quadtree
  - R-Tree ファミリ
    - R-tree, R*-tree
  - 最近接点探索用
    - SS-tree, SR-tree, VAM Split R-tree
  - 多次元データ構造（ユークリッド空間）
    - X-tree, LSD –tree, Hybrid tree
  - VA-File ファミリ（ベクトルの近似表現）
    - IQ-tree, A-tree　本来のVA-File は木構造でない
  - metric tree ファミリ
    - metric tree, M-tree, MVP-tree

空間問い合わせの研究課題
－　最近接点探索　－

最も近い　k 個を求めよ
（k と質問点はユーザが指定）



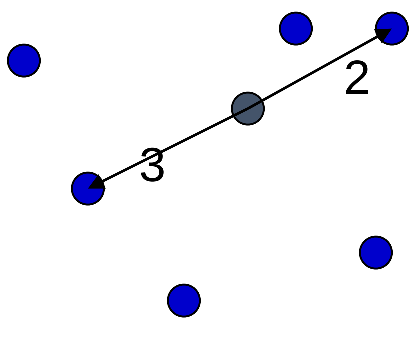
空間問い合わせの研究課題
－　次元ののろい　－

高次元のベクトルデータの「奇妙」な振る舞い

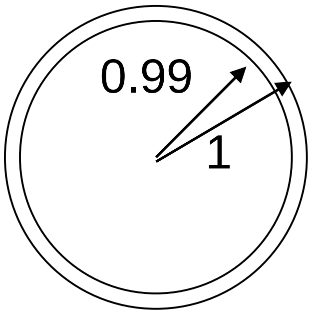
点がランダム分布しているとすると,
k 番目に近い点と k + 1 番目に近い点の比は

1 + 1/(kn)　 (nは次元)

n が大きいと，この比は 1 に近づく

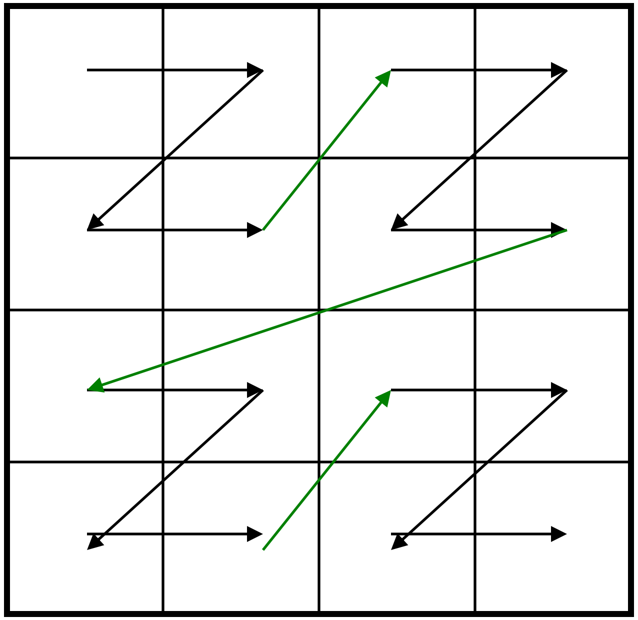


半径 0.99 と 半径 1の超級の体積の比：
高次元では，体積の比が大きくなる



空間重点曲線による方法

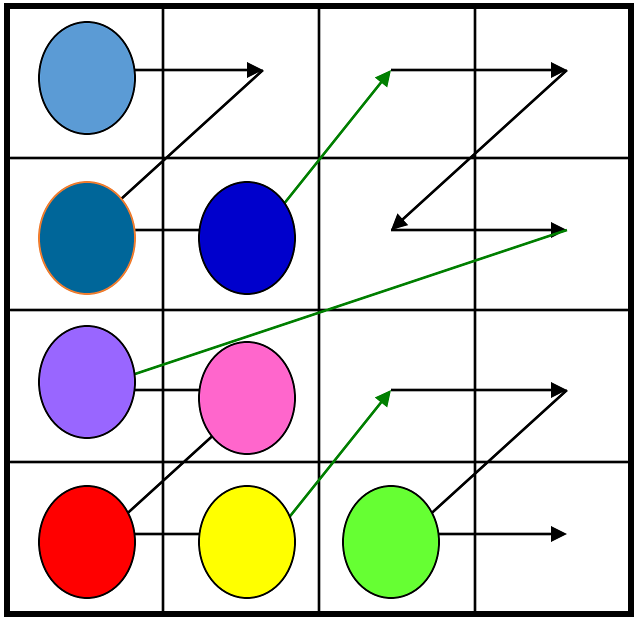
空間充填曲線の例

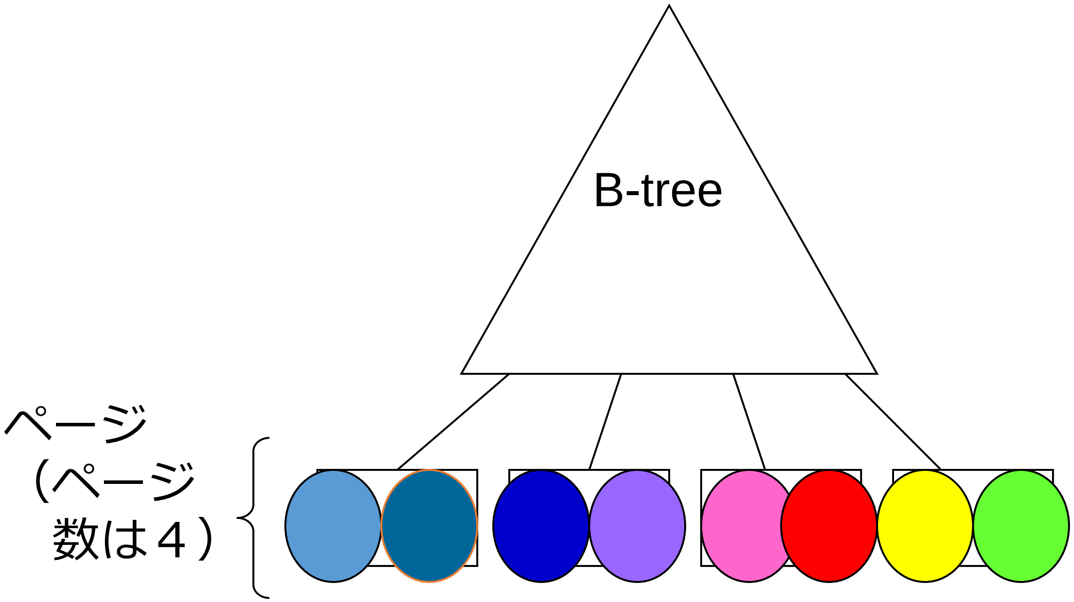


0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

各領域の番号

空間充填曲線と B-tree





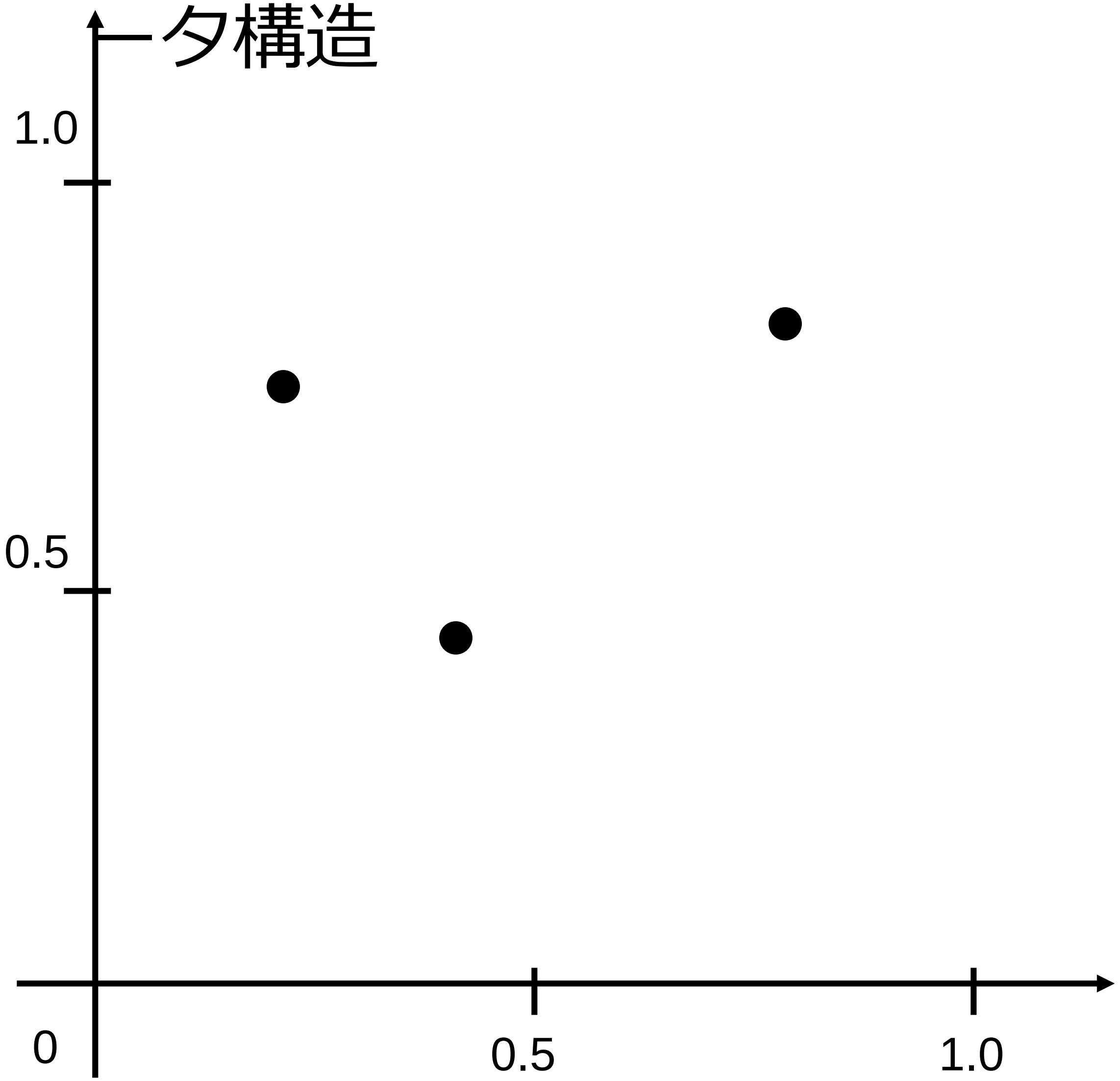
B tree

- 木構造のインデックス
- 木の深さがバランスするよう、挿入、削除時に処理が行われる
- 各ノードは多数の分岐を持ち、結果として深さは少なくなる
  - 各ノードは、K＋１個から２K＋１個までの分岐を持つ（Kはページサイズから決まる）

G-Tree

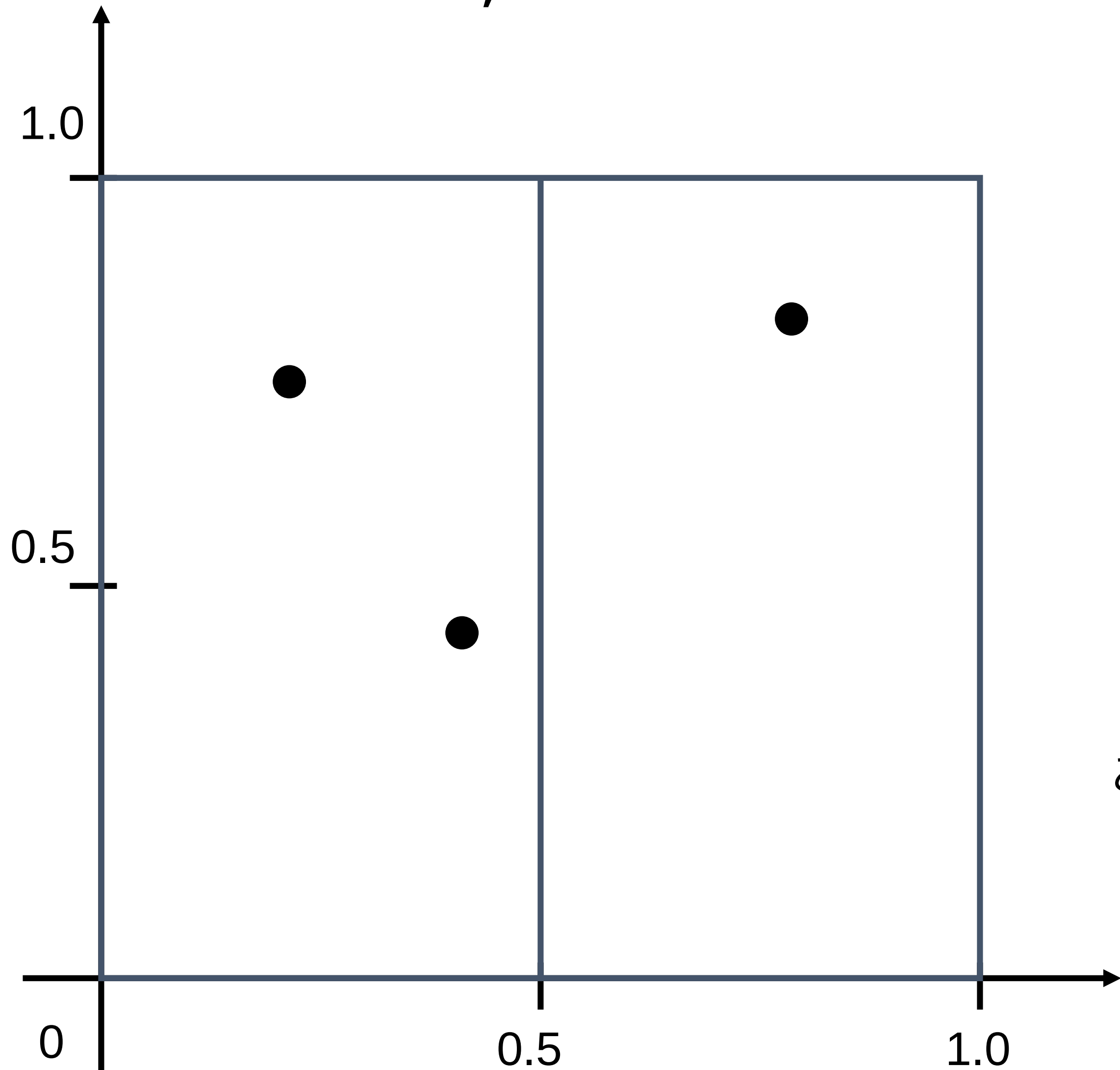
G-Tree データ構造 1/4

• G-tree は，多次元空間中の点の集まりのためのデータ構造



G-Tree データ構造 2/4

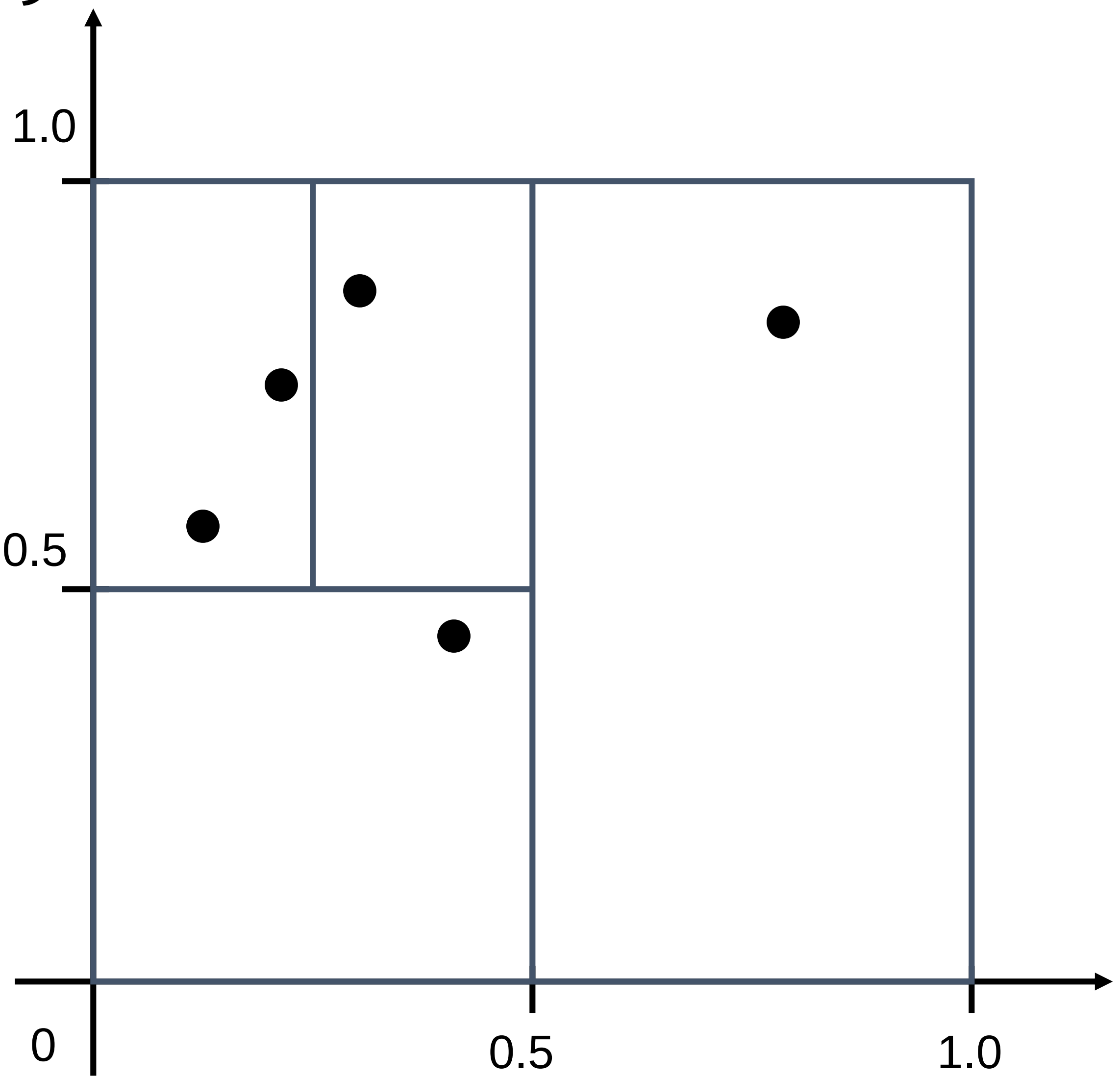
- 1つの区画内の点の数がある制限（ここでは2）を越えないように，空間分割が行われる



この例では：
点データ：　３個
分割数：　2

G-Tree データ構造 3/4

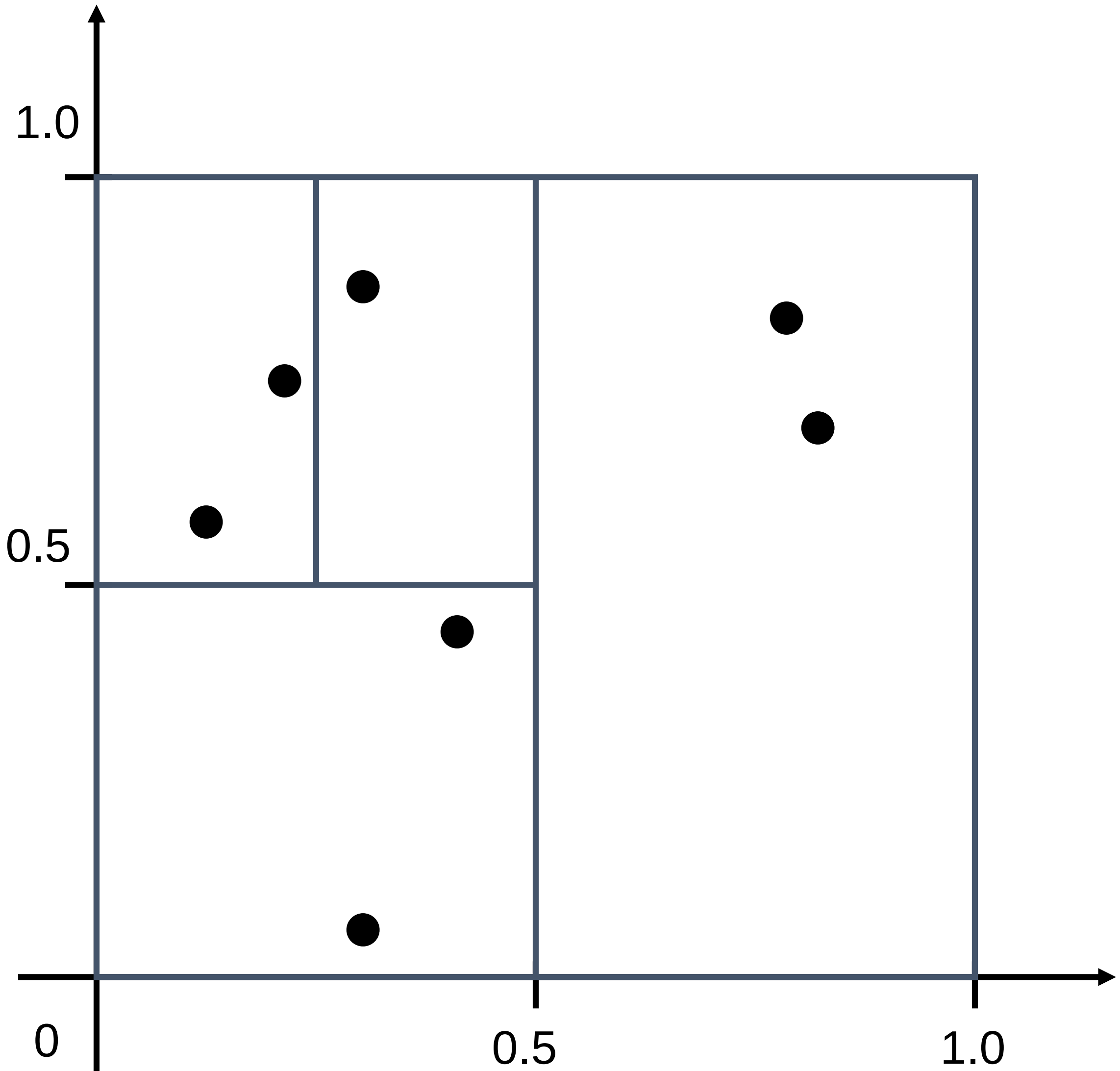
- 空間分割は，区画を半分に分割することを繰り返す



この例では：
　点データ：　5 個
　分割数：　4

G-Tree データ構造 4/4

- 点の数が増えたからといって，必ずしも区画が分割されるわけではない.



この例では：
点データ：　７個
分割数：　４

G-Tree の考え方

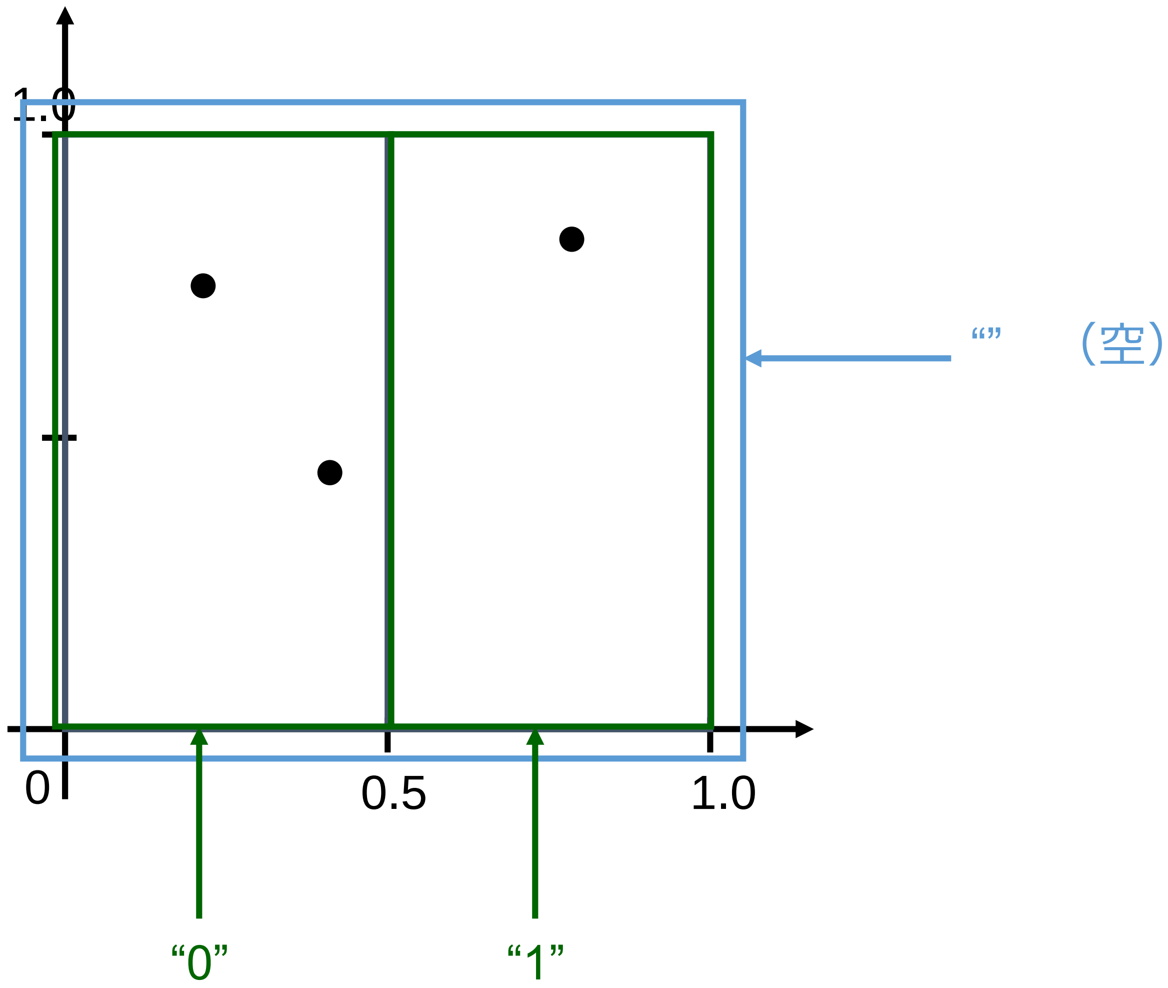
- データ格納の単位は「区画」
  - ある「区画」に含まれる点データの数は，ある「最大値」を超えない．
  - 「最大値」は，１区画分のデータが１ページに収まるように決める

- ツリー構造
  - 「区画を，半分の面積に分割」することを繰り返す
  - 分割軸は x → y → x → y ... のように交互に交代する

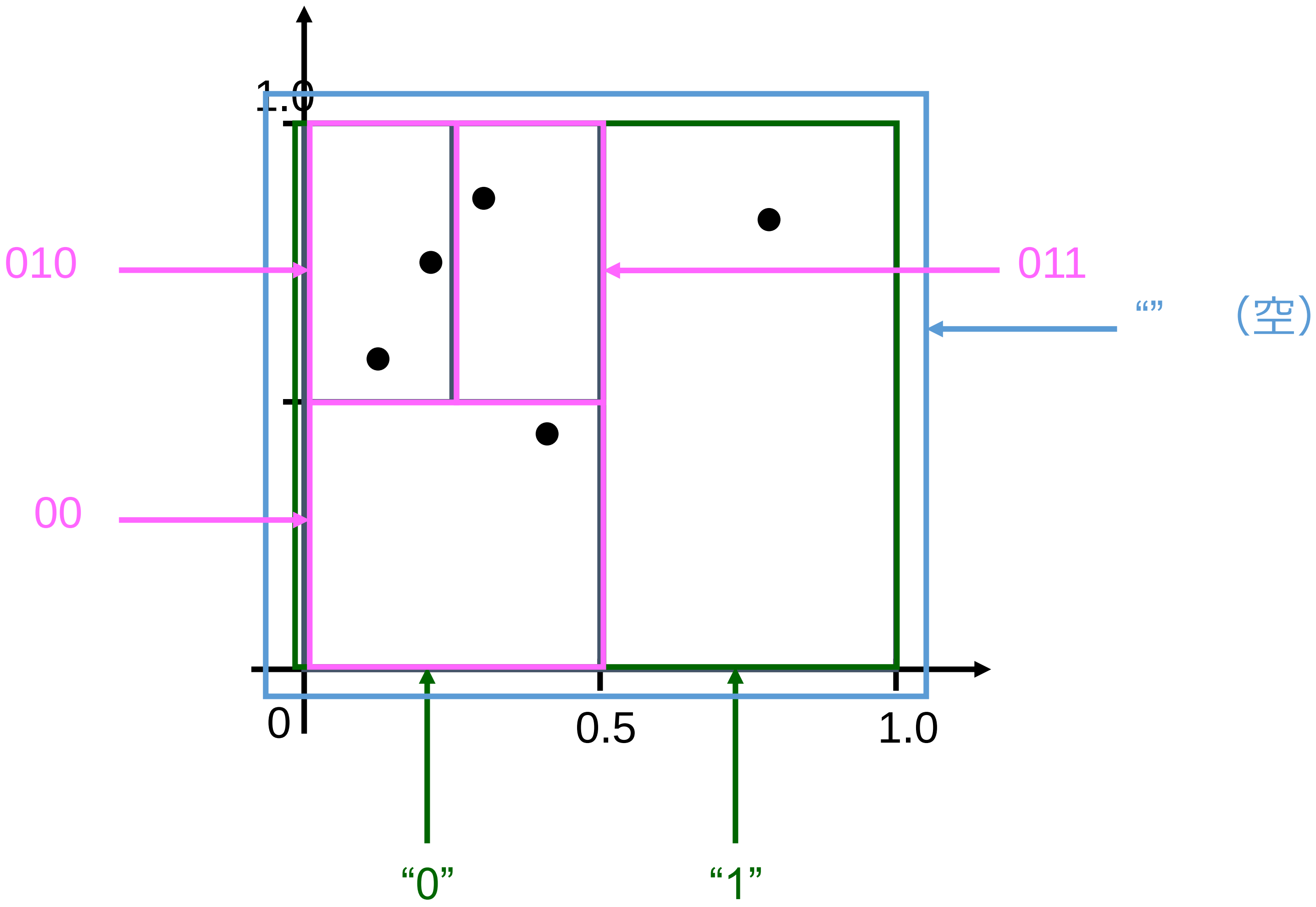
G-tree

- 今までの説明では，次のことを仮定していた
  - 次元数：　　　　　　2 次元
  - データの範囲：　0 から 1
- しかし，本来の G-Tree は，次元数，データの範囲に制限は無い
  - 次元数：　　　　　　2 次元以上
  - データの範囲：　広い

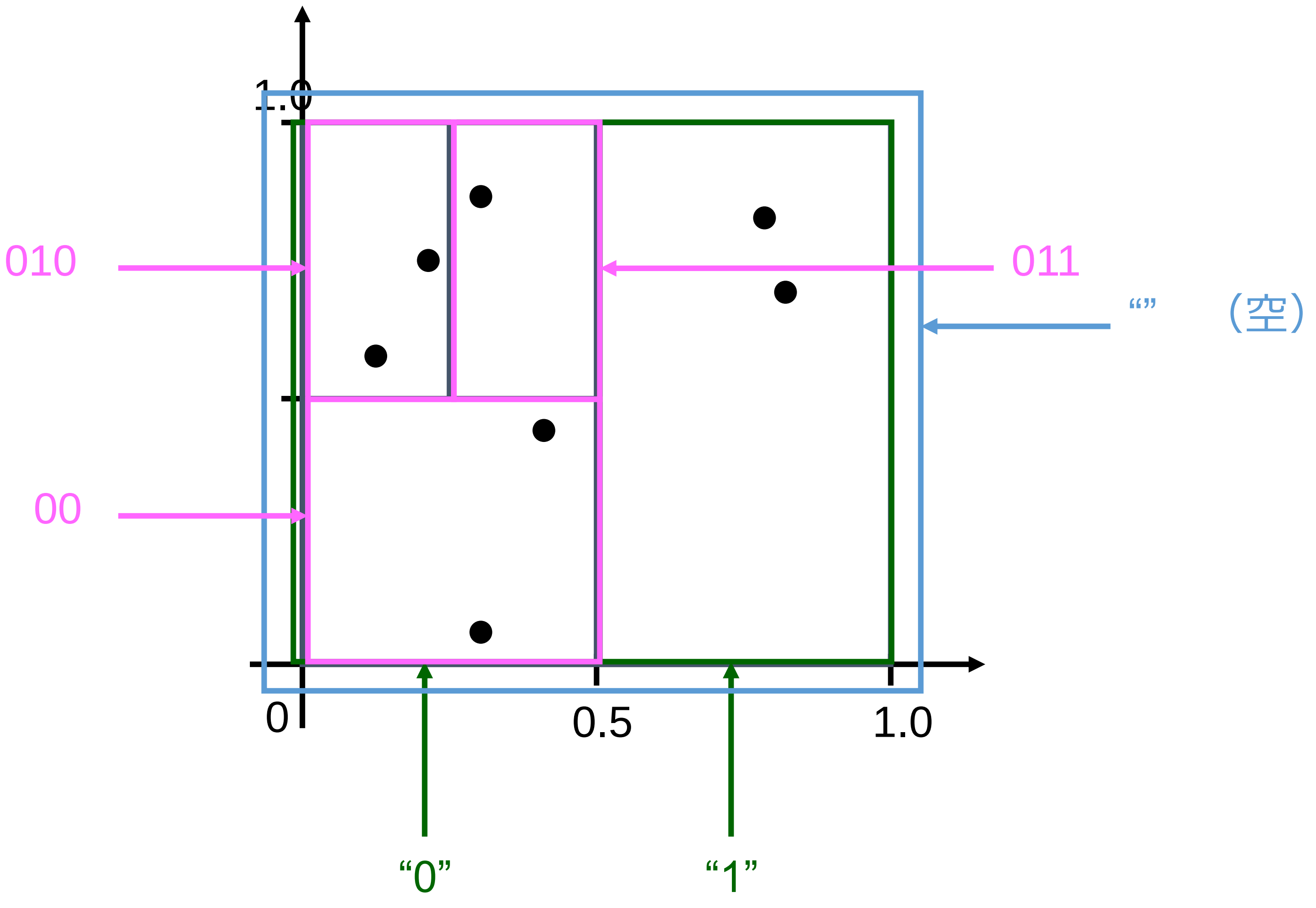
区画のビット列表現例 1/3



区画のビット列表現例 2/3



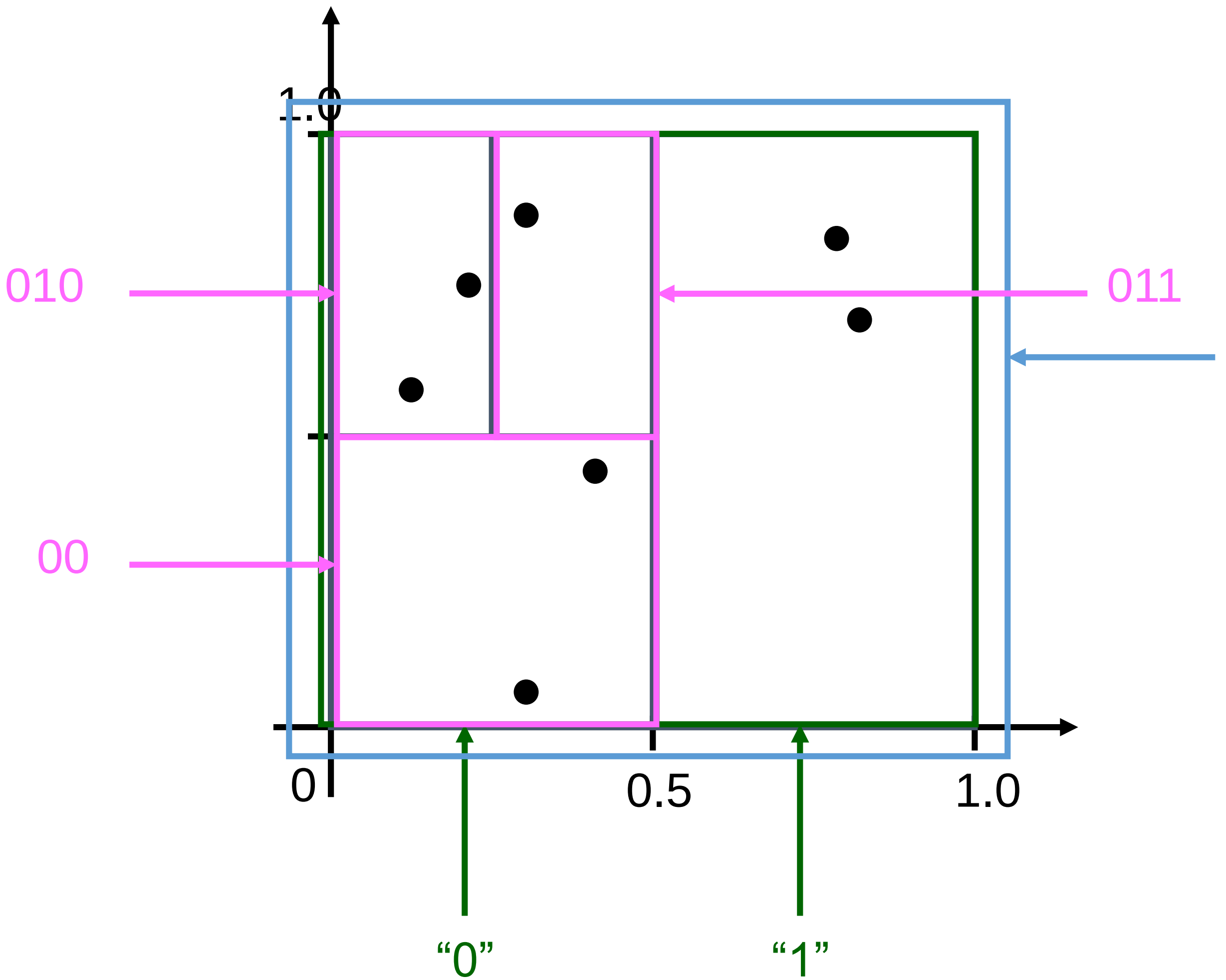
区画のビット列表現例 3/3



区画のビット列表現

- 区画を　０，１のビット列で表現
- 全空間は「空文字列」とする
- 区画が分割されると，ビット列の長さが１つ増える
- 最大ビット列長（前ページの例では３）は，区画の分割回数で決まる

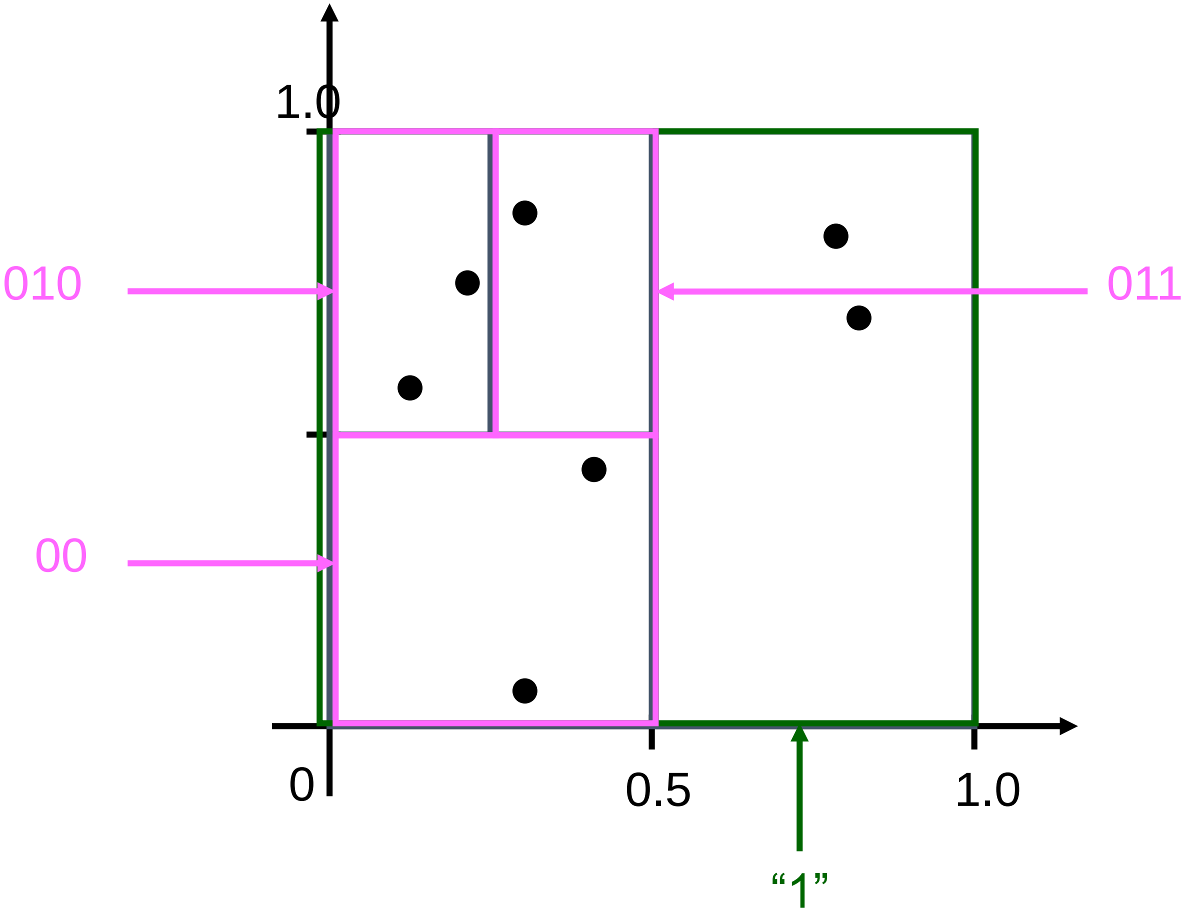
ビット列表現の性質



ある区画Rが別の区画Xを含むなら，
Xのビット列表現は S"0" 以上で，S"1"以下

ビット列表現による区画の順序付け

- この例では，順序は，00 → 010 → 011 → 1



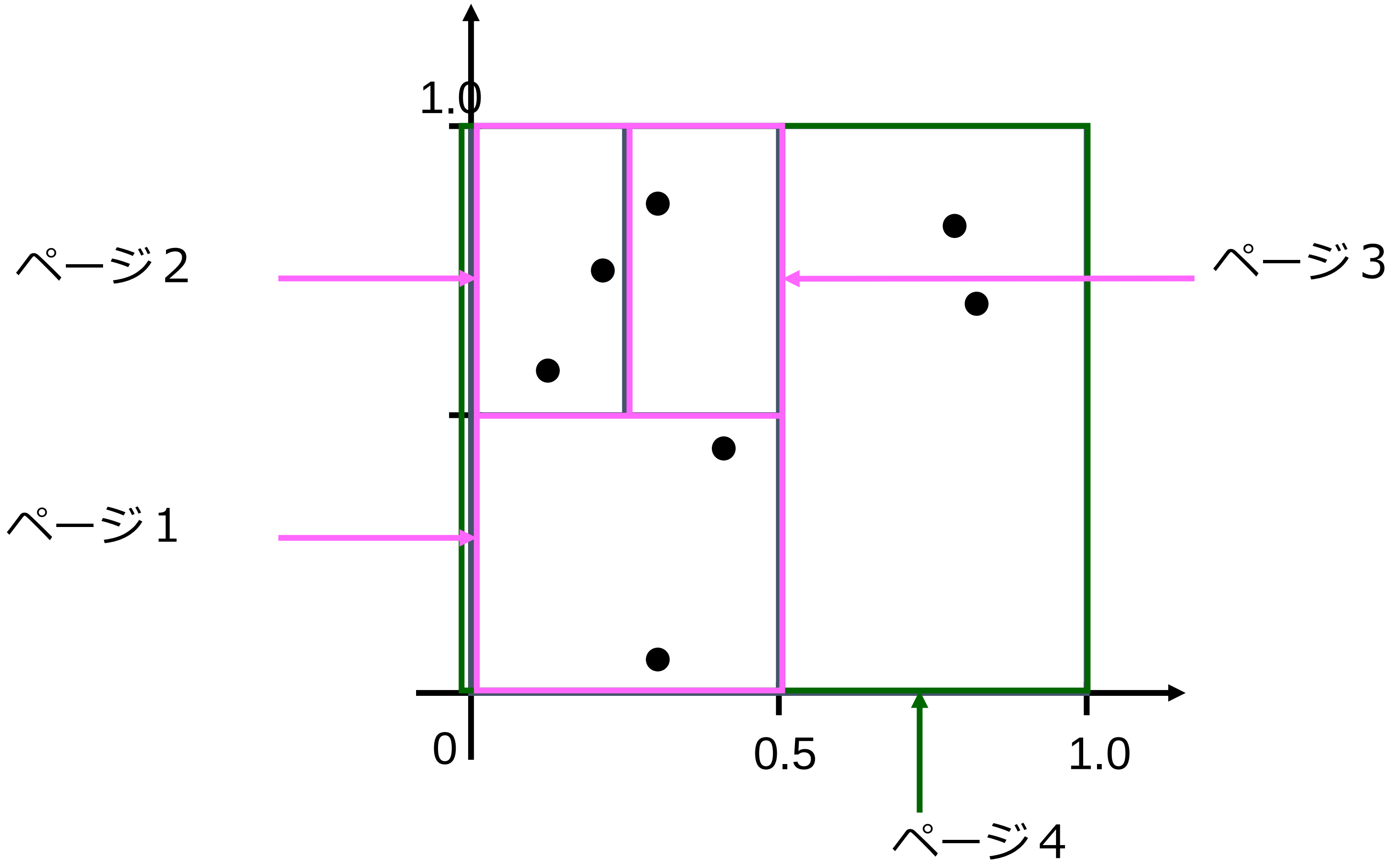
区画を順序付けることの意味

- 区画を，あらかじめ「ソート」でき，点データの各種の操作を高速化できる

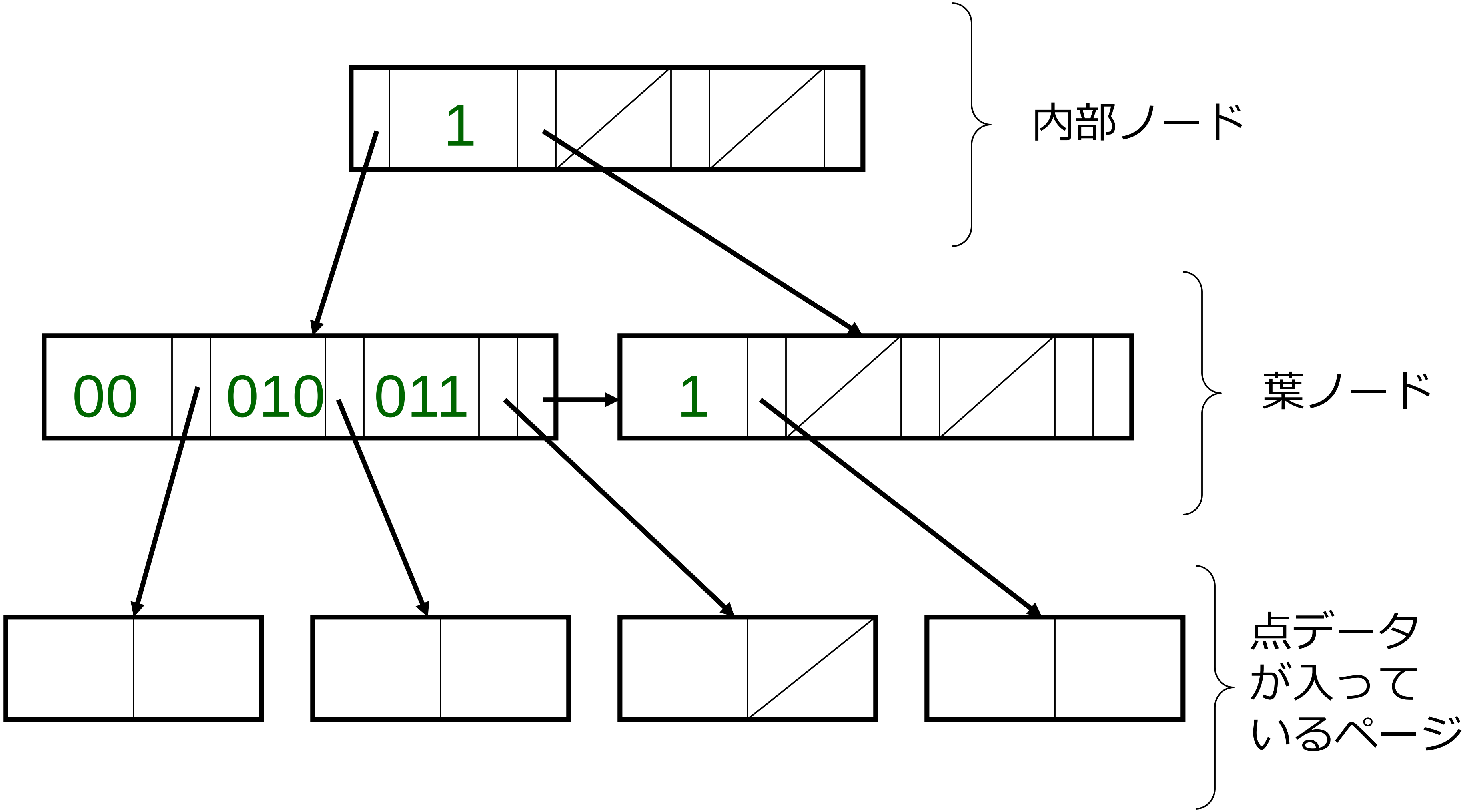
- 区画の「ビット列」は，点データを検索するための「検索キー」と見立てることができる

G-tree の格納構造　1/2

- １つの区画で１ページ
- 各ページは順序付けられている



G-tree の格納構造　2/2



G-tree のノード

- 葉ノード
  - データページへのポインタを持つ
  - 「次」の葉ノードへのポインタを持つ

- 内部ノード
  - 下位レベルのノードへのポインタを持つ

G-Treeのオペレーション

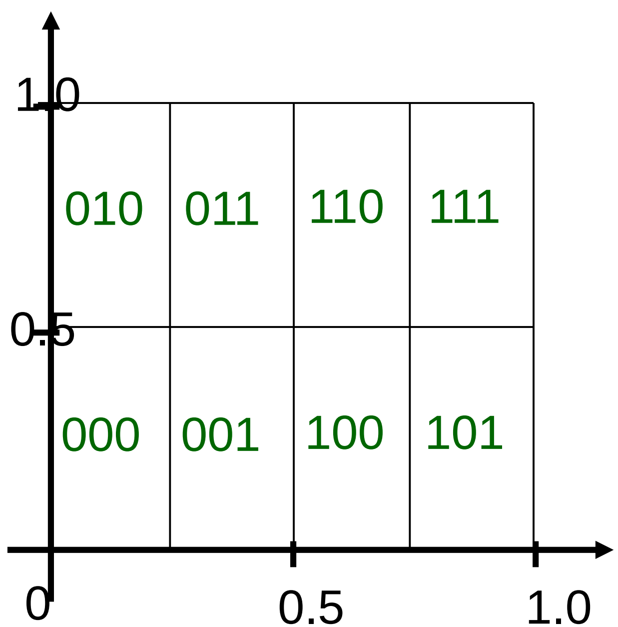
- Searching for a Point
- Searching for All Points In a Region (Range Query)
- Inserting a Point
- Deleting a Point

Searching for a Point

- 探したい点の座標値（x,y）は分かっている
- 座標値(x,y)を使って，当該データが存在するかしないかを調べたい
  - もし，インデックスが無ければ，データの全件を調べねばならない

Searching for a Pointの手順 1/2

- 最大ビット列長は，前もってどこかに覚えておく
  例えば：　3
- 与えられた座標値と，最大ビット列長とから，ビット列を求める．
  例えば：　(0.8, 0.9) のビット列は，“111”

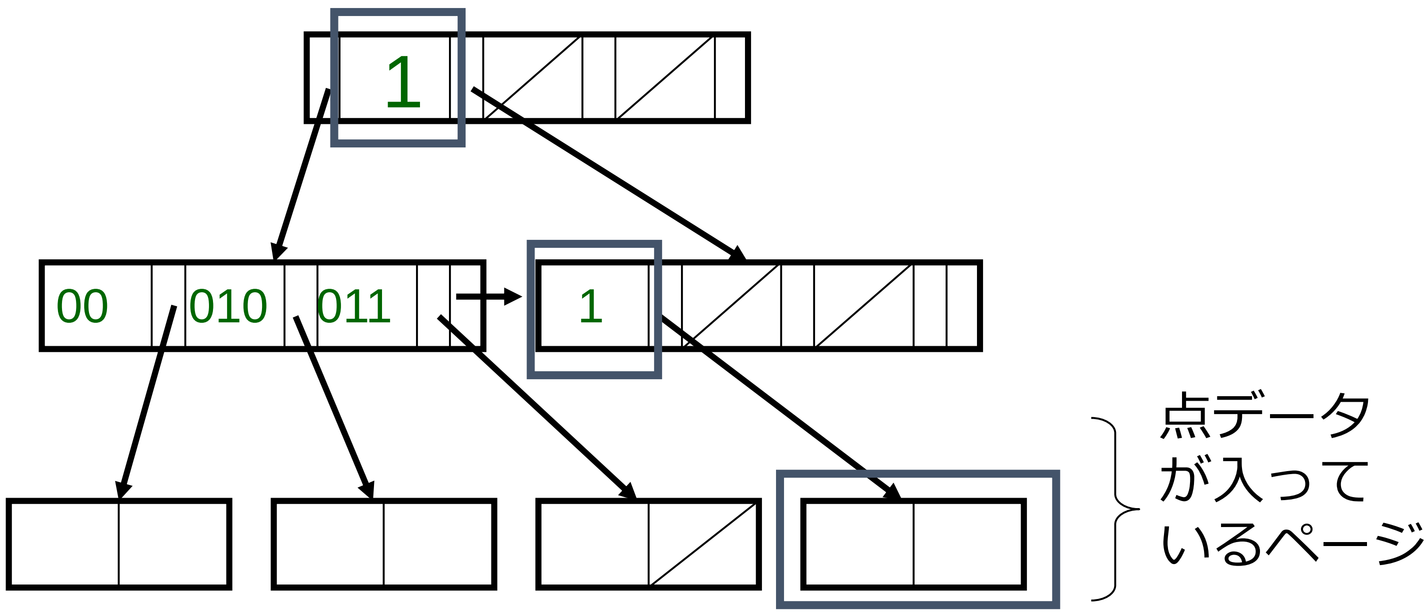


Searching for a Pointの手順 2/2

- 2で求めたビット列を使って，G-tree をたどり，探したい点を含むページ番号を求めたい

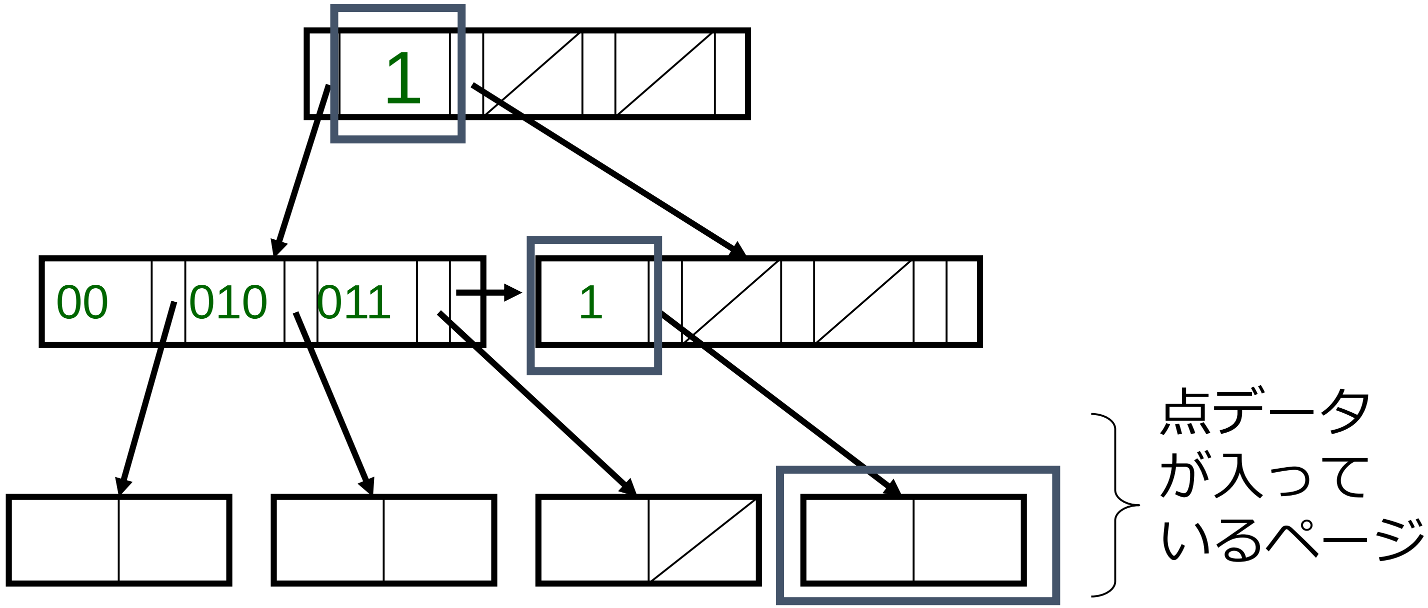
(例)　“111” を使って，下の G-tree をたどる

G-tree に“111” は無いが，“1” はある



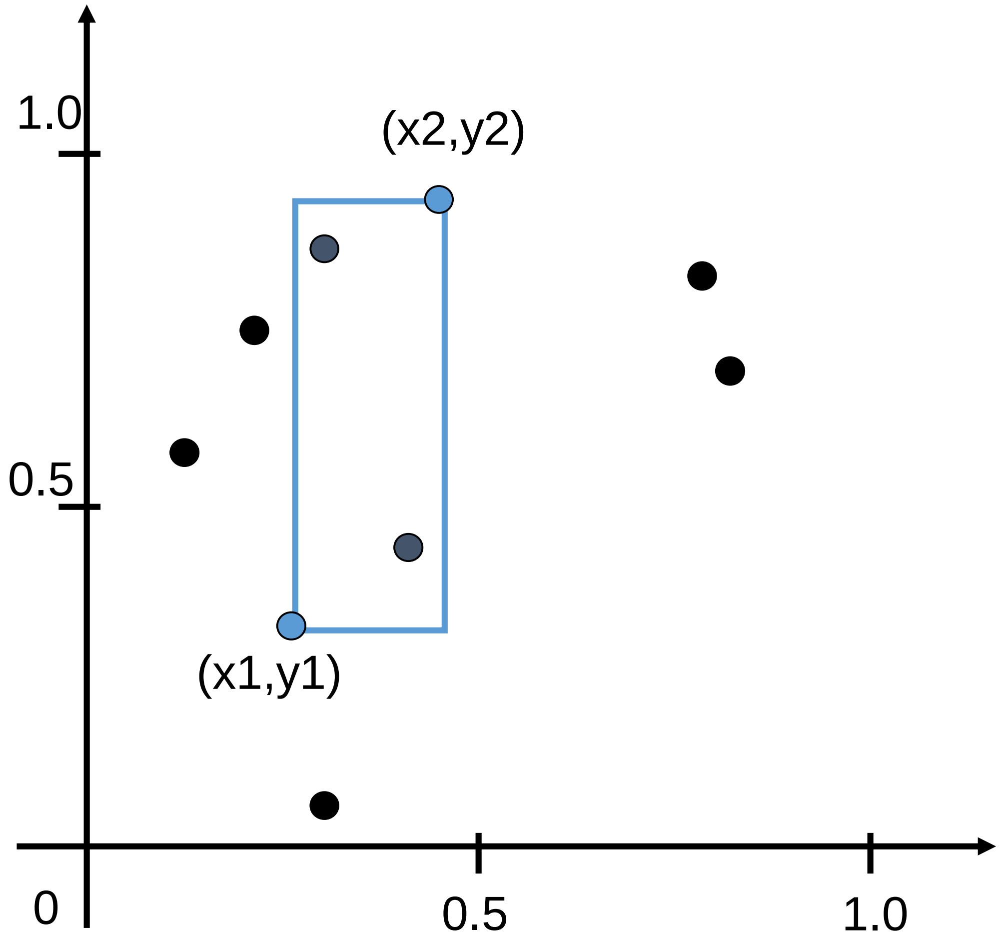
Searching for a Point　について

- G-tree を，ルートノードからたどり，与えられた「点」が入っているであろうページを得ることができた



Range Query

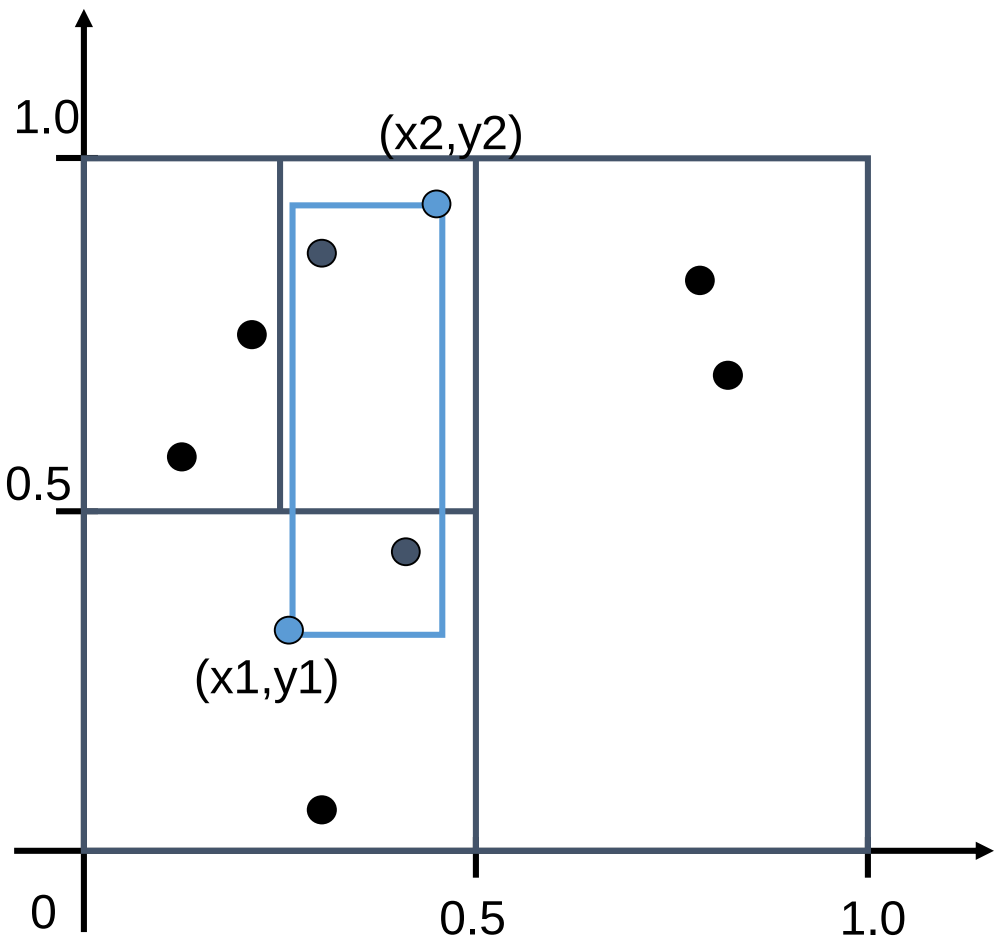
- 矩形[x1,y1,x2,y2] による検索
- 矩形内のすべての点を求める



Range Query の手順 1/2

- (x1,y1)から，３ビット表現“００１”を得る
- (x2,y2)から，３ビット表現“０１１”を得る

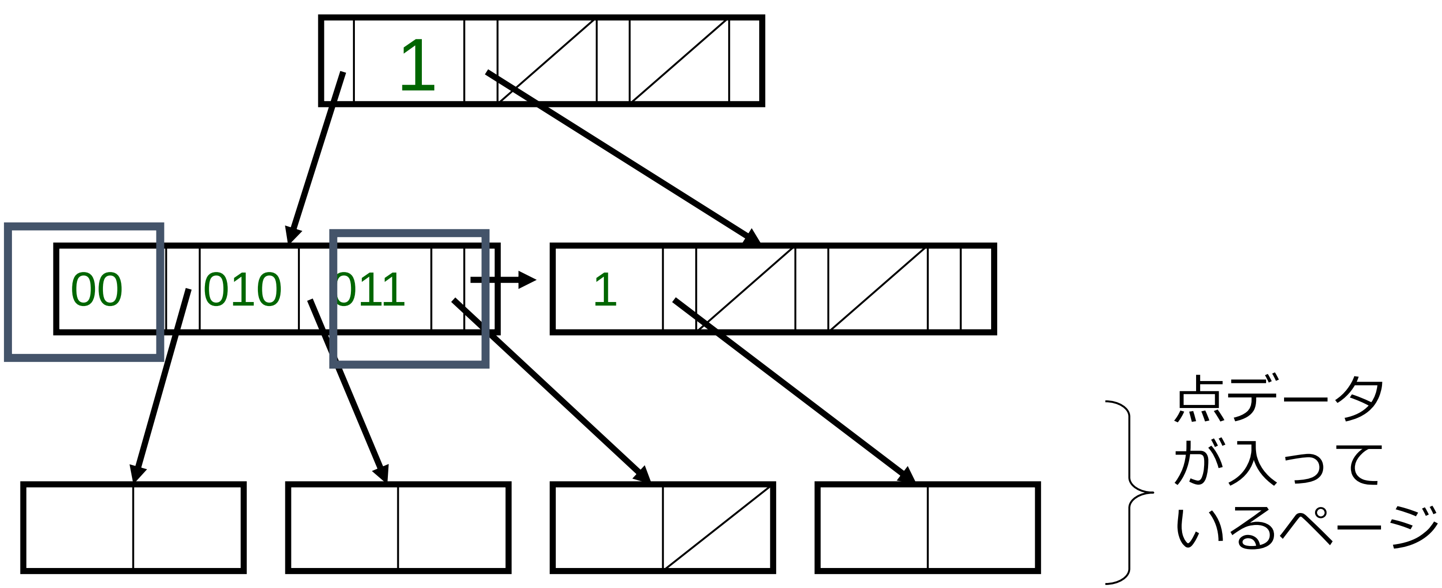
→　解は　００１，０１０，０１１の範囲内だと分かる



Region Query の手順 2/2

- G-tree の葉ノードをたどり，それぞれ矩形 [x1,y1,x2,y2] の範囲と重なるかを調べる

(例) 00, 010, 011 を辿る． 00 と 011 は重なる



Inserting a Point

- 挿入したい点(x,y) について

１． (x,y) を使って， Searching for a Point　を実行.

ページ番号を得る

２．１の結果，すでに，点(x,y)が存在していることが分かれば，何もしない

３．点を挿入した結果，ある区画内の点の数が「制限」を超えそうなら，区画を分割する

（１）　データページを１つ増やす

（２）　必要なら，葉ノード，内部ノードを調整する

Deleting a Point

• 削除したい点(x,y) について

１．(x,y) を使って， Searching for a Point を実行.
ページ番号を得る

２．すでに，点(x,y)が存在しているはず（無ければエラー）

３．点を削除した結果，ある区画内の点の数が 0 になるなら，区画を結合する.

（１） データページを１つ減らす

（２） 必要なら，葉ノード，内部ノードを調整する

G-tree と B-tree の違い

- B-tree は，数値，文字など「順序付け」可能なデータのためのデータ構造
- 点データは，（基本的には）順序付けできない
- G-tree では，「区画」をビット列表現し，順序を付ける
  - ある区画Rが別の区画Xを含むなら，Xのビット列表現は S"0" 以上で，S"1"以下
  - 点データに特徴的な検索として， Searching for All Points In a Region がある

k-d Tree

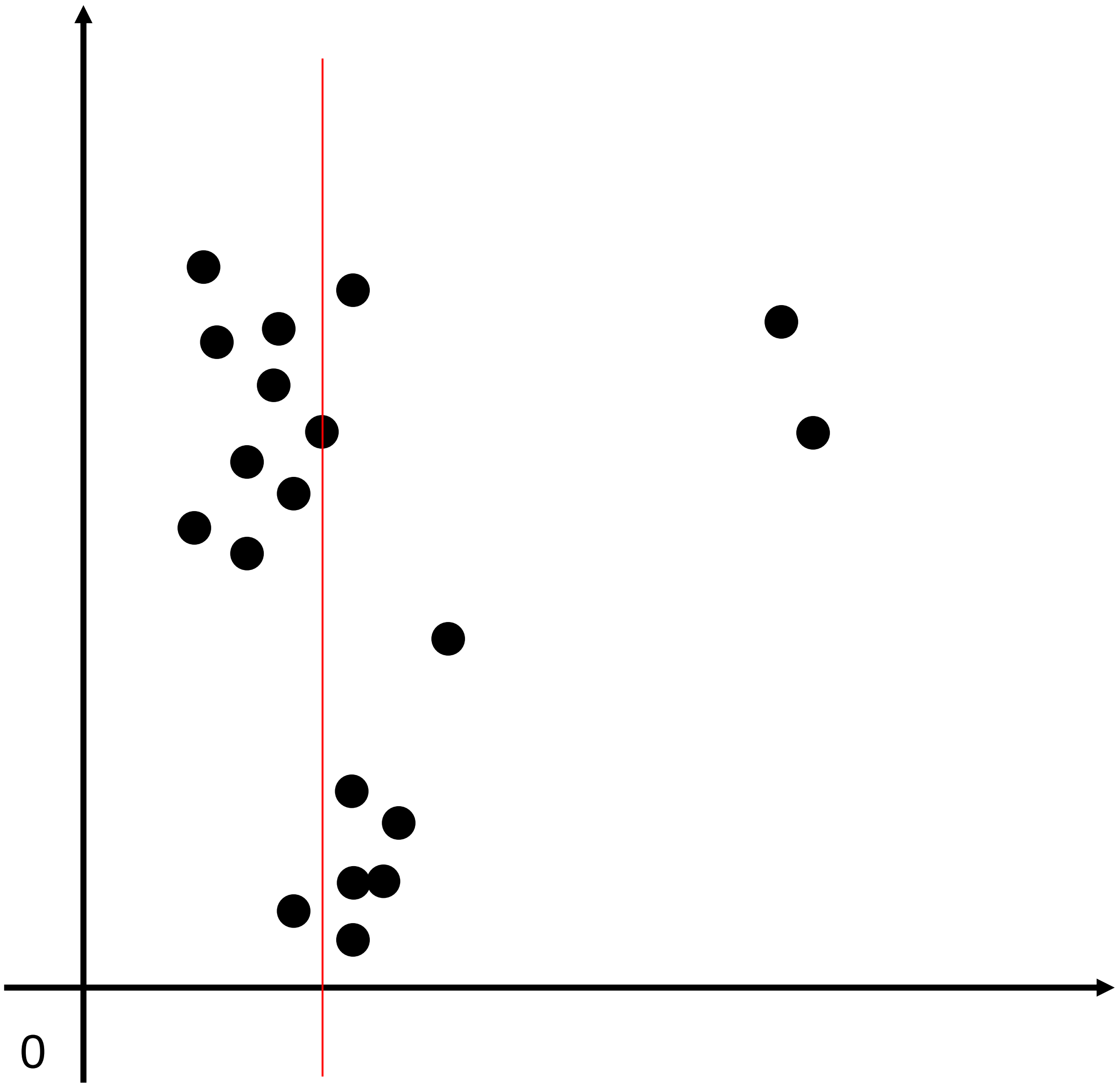
k-d Tree

- 各座標値についてデータをソートし，その中央値で，データを２つに分割

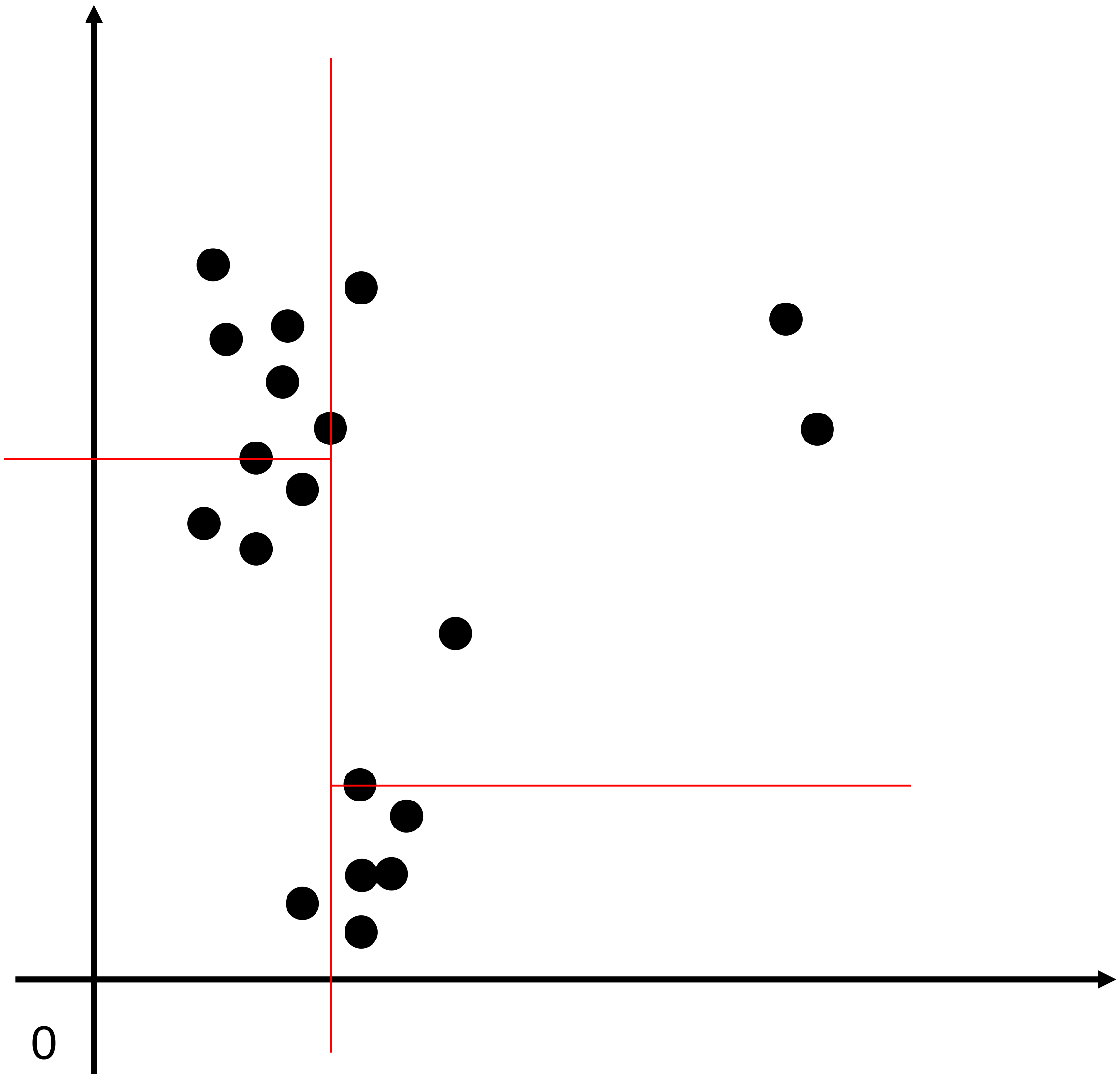
- 分割する軸の選択法
  - 巡回：　x → y → z → x → y → z →
  - その他

- 2-d Tree は２次元の点の集まりを、
- 3-d Tree は３次元の点の集まりを扱う

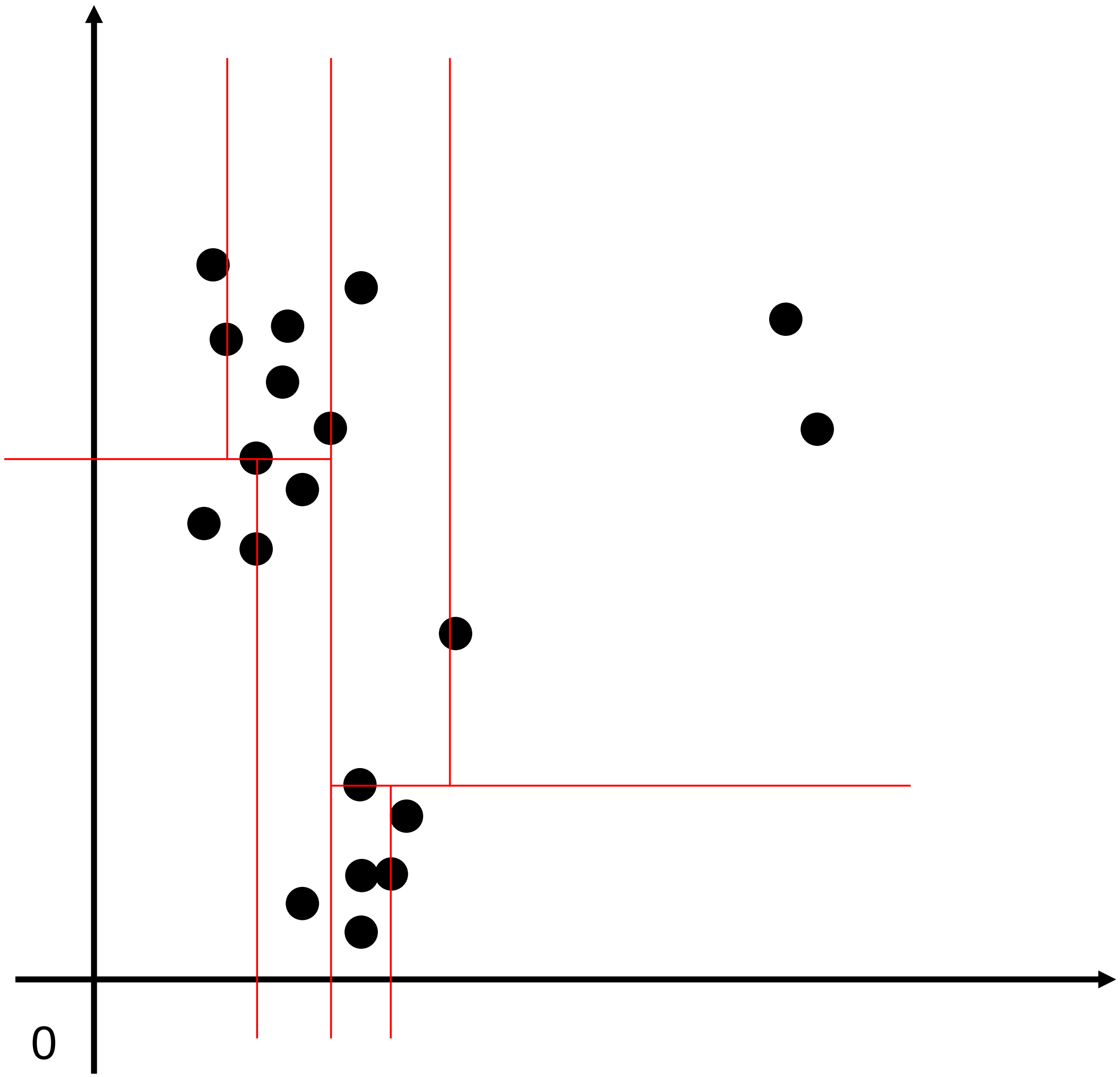
k-d Tree



k-d Tree



k-d Tree



2-d Tree

- x　軸で分割すると

  N.LLINK が指すノードMは、

  M.XVAL < N.XVAL

  N.RLINK が指すノードPは、

  P.XVAL ≧ N.XVAL

- y　軸で分割すると

  N.LLINK が指すノードMは、

  M.YVAL < N.YVAL

  N.RLINK が指すノードPは、

  P.YVAL ≧ N.YVAL

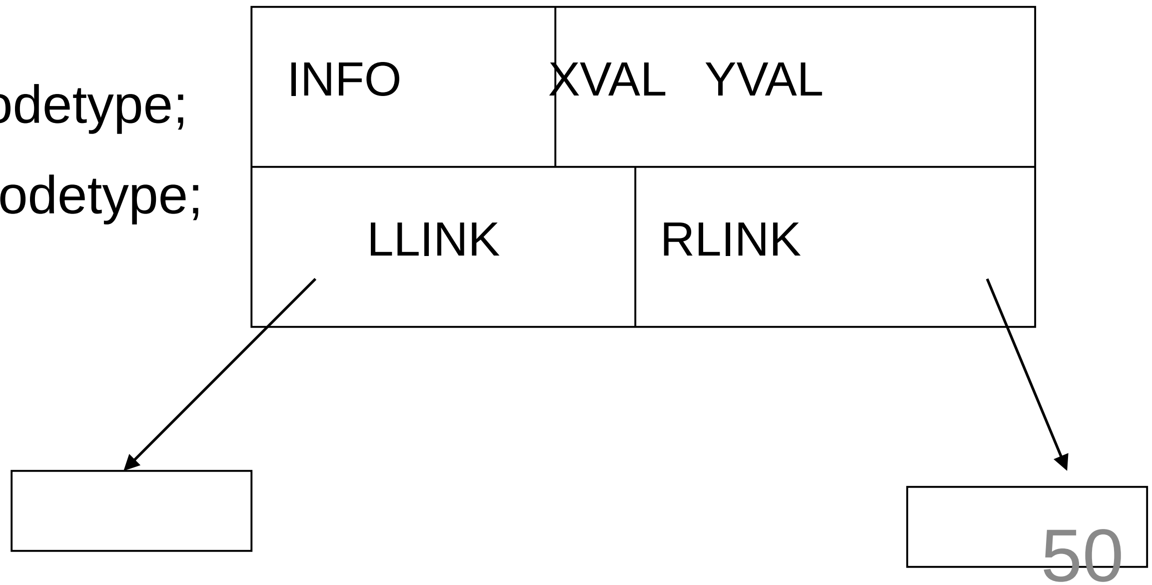
k-d tree の特徴

- データの集まりから，k-d tree の一括生成
  - k-d tree は完全にバランスする
- データを逐次，追加，削除する場合
  - k-d tree のバランスは崩れる

k-D Tree のデータ構造

- INFO: 利用者が好きなデータを格納
- XVAL, YVAL: 「点」の座標値
- LLINK, RLINK: ２つの子ノードへのポインタ
- １ノードが１つの点に対応する

```
nodetype = record
        INFO: infotype;
        XVAL: real;
        YVAL: real;
        LLINL: pointer to nodetype;
        RLINK: pointer to nodetype;
end
```



2-d Tree を使った挿入

- 挿入したいノードをN, 2-d Tree の根ノードをT とする

- N と T のXVAL, YVAL が同じなら、Tを上書きして終了
- さもなくば，次の手順で子ノードを探す

x 軸で分割しているとき

  - N.XVAL < T.XVAL なら左へ分岐
  - N.XVAL≧T.XVAL なら右へ分岐

他の軸についても同様

- 以上の手順を繰り返す

2-d Tree を使った削除

- 削除したい点を (x,y) とする
- 2-d Tree から点 (x,y) を探す（Nとする）
- Nが根ノードなら：

Nの親ノードの、RLINK, LLINK のうち N を指している方を NIL に設定

- Nが根ノードでなければ：

１．「候補」Rとして Nのサブツリーの中から適当なノードを選ぶ

２．Rの INFO, XVAL, YVAL の値で、Nを書き換える

３．Rについて、「ノードRの削除」を行う（再帰処理になる）

候補Rの選び方

- ノードNの削除では、「候補」Rは、次の条件を満たさねばならない
- T の左側のサブツリー内の任意のノードMについて：

M.XVAL　＜　R.XVAL　（ x 軸で分割のとき）

M.YVAL　＜　R.YVAL　（y軸で分割のとき）

- T の右側のサブツリー内の任意のノードMについて：

M.XVAL　≧　R.XVAL　（ x 軸で分割のとき）

M.YVAL　≧　R.YVAL　（ y 軸で分割のとき）

k-D Tree を使った範囲検索

- 範囲検索とは
  範囲を指定して、その範囲内の点の集まりを求めること

- k-D Tree の各ノードN

　　　Nと、そのサブツリー内の点の範囲が決まっている（ RN とする）

- 指定された範囲と、あるノードNの範囲 RNが重なっていなければ、Nとそのサブツリー内には、範囲検索の解はない

範囲検索について

- 「範囲」の例

（１）矩形

右上の点の座標値と、左下の点の座標値

（２）円

中心の座標値を、半径

k-D Tree について

- k-D Treeでは、木の形は、データの挿入順で決まる

- k-D Treeでは、領域を２つあるいは４つに分割する

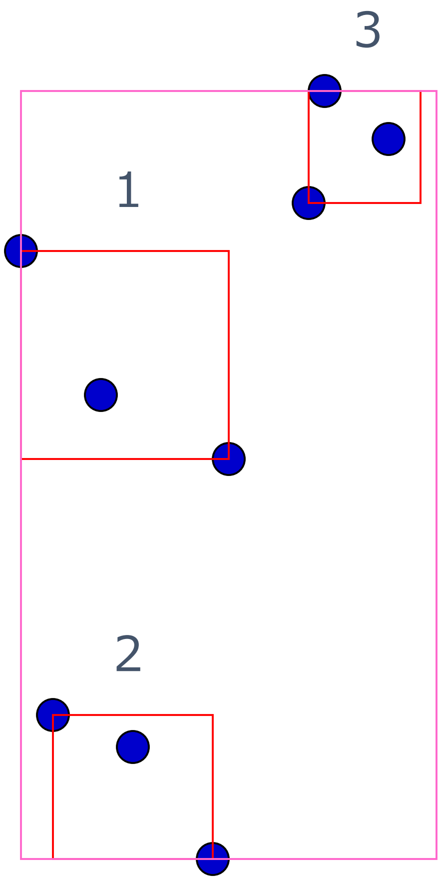
- 分割された領域の大きさは、おのずと不均等になる (XVAL, YVAL値で決まる)

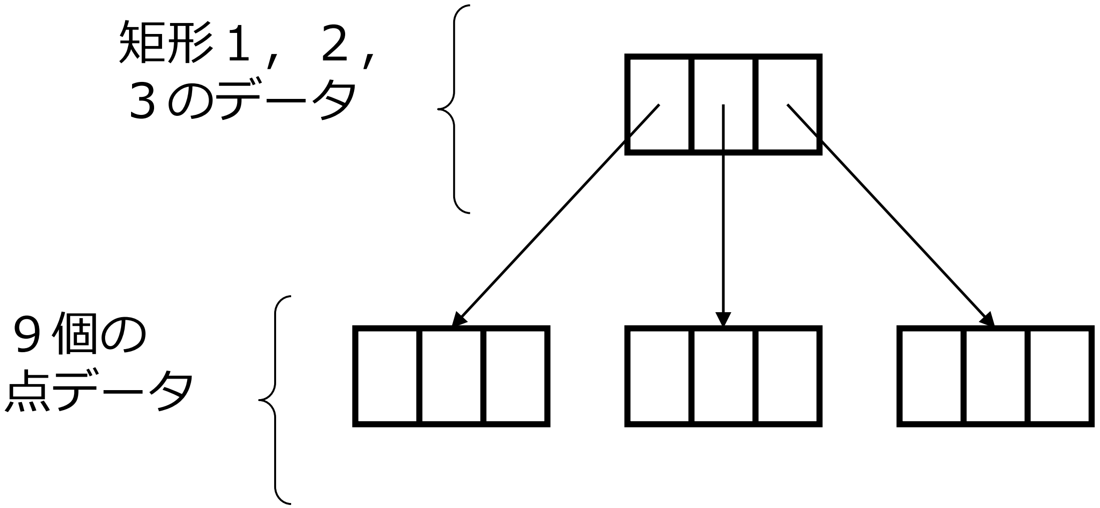
R-Tree / R*-tree

「領域」情報を扱うための手法

R-tree, R*-tree の構造

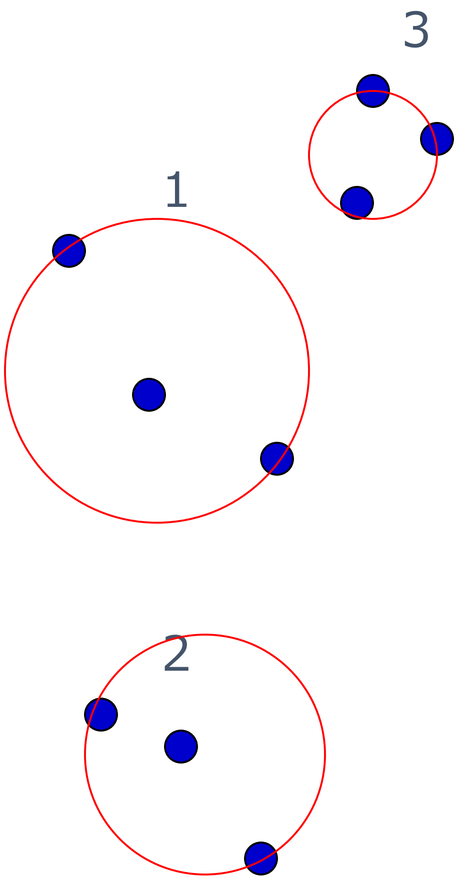
９個のベクトルデータ

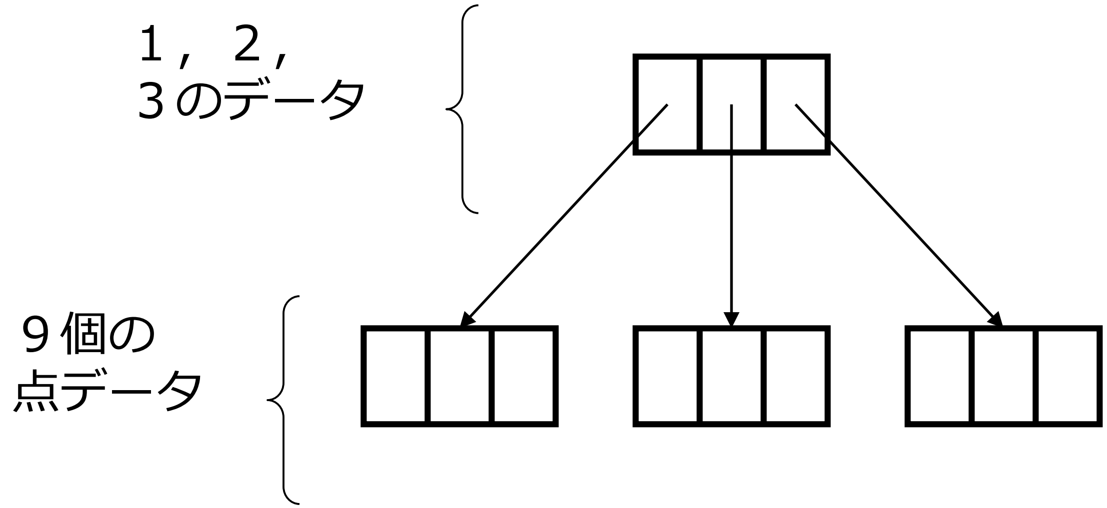




SS-tree の構造

９個のベクトルデータ





MX QuadTree

MX QuadTree

- MX QuadTree　は、点の座標値にかかわらず、ノードを均等に分割
  - 木の形は、データの挿入順に依存しない
  - データの挿入、削除の操作が簡単になる

MX QuadTree

- 根ノード
  - 領域 [（０，０）, (2^k ,2^k)]

[（XLB，YLB）, (XUB ,YUB)]の子ノード

NW： [（XLB，YLB+w/2）, (XLB+w/2, YLB+w)]

SW: [（XLB，YLB）, (XLB, YLB+w/2)]

NE: [（XLB+w/2，YLB+w/2）, (XLB+w/2, YLB+w)]

SE: [（XLB+w/2，YLB）, (XLB, YLB+w/2)]

MX QuadTree の性質

- データは、すべて、根ノードにある
- 根でないノードのサブツリーは、必ず、データの入った根ノードを含む
- 挿入、削除は、根ノードについて行う

MX QuadTree を使った削除

- 削除したい点を (x,y) とする
- MX QuadTree から点 (x,y) を探す（Nとする）．
  - Nは必ず葉ノードである.
- Nを削除する
- Nの親ノードをMとする.
- MのエントリのうちNを指している方をNILとする
- M のNW,SW,NE,WEが全てNILならば、Mを削除する（削除は再帰的に繰り返す）

空間インデックス研究課題

- 検索効率
  - 領域検索
  - 最近接点探索
- データの追加，削除，更新の効率
- 空間使用量
- 大規模なデータセット
- 高次元のデータ