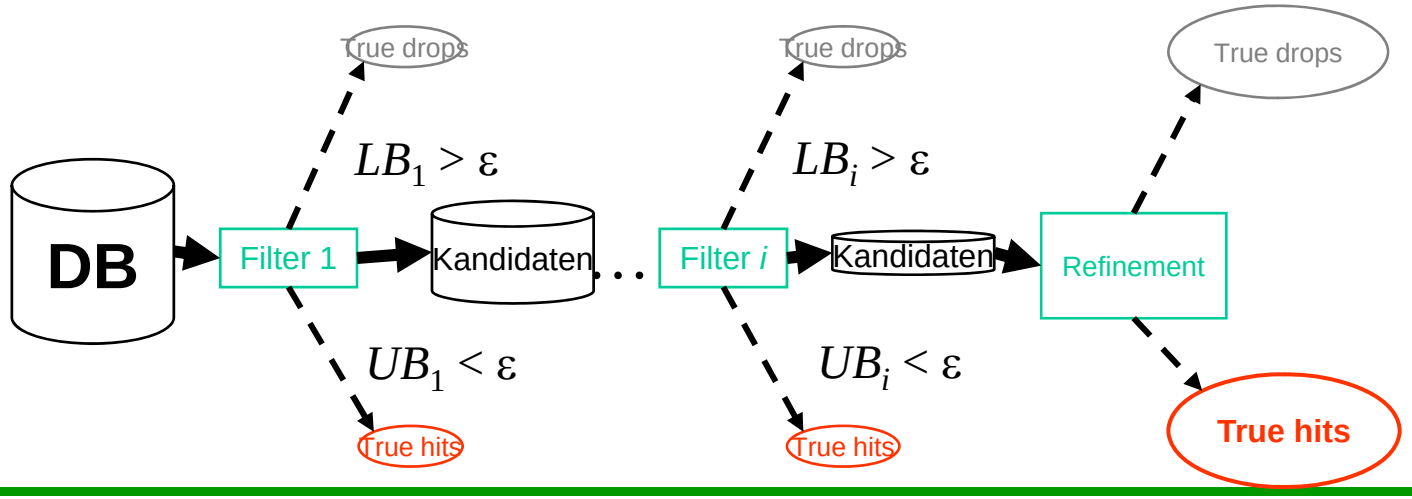
2.3 Mehrstufige Nachbarschafts-Anfragebearbeitung

2.3.1 Mehrstufige Bereichsanfrage

- Anfrage: Ergebnis enthält alle Objekte, die höchstens eine Distanz von ε zu Anfrageobjekt *q* haben.
- Filter:
  - filter-dist$_{LB}$(o,q) > ε => dist(o,q) > ε => Objekt o kann ausgeschlossen werden (true drop)
  - filter-dist$_{UB}$(o,q) < ε => dist(o,q) < ε => Objekt o gehört zum Resultat



- Mehrstufige-Bereichsanfrage Algorithmus: (Filter-/Refinement)
  - Lower Bounding Filterdistanz $dist_{LB}$
  - Upper Bounding Filterdistanz $dist_{UB}$

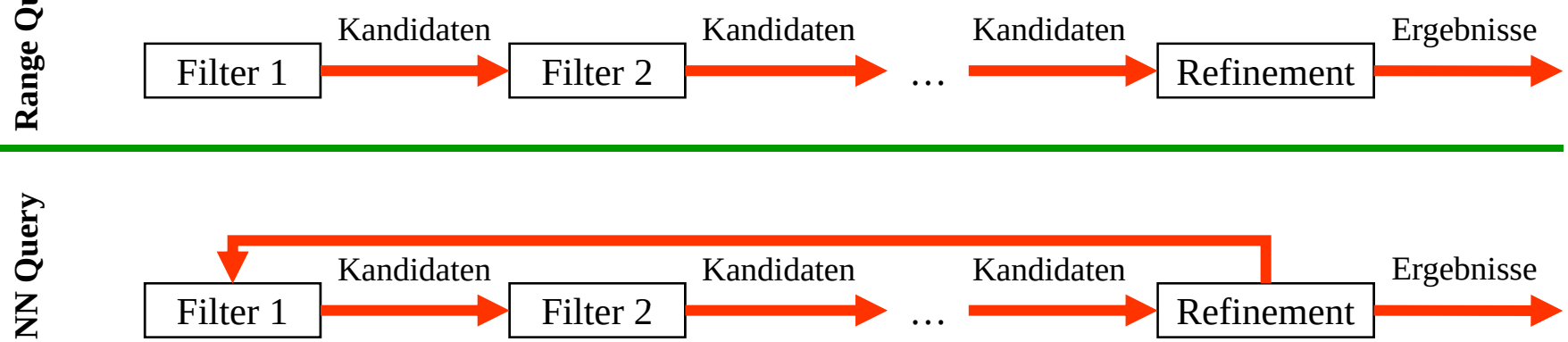
```
RQ-MultiStep(DB, q, ε)
    result = ∅;
    candidates = ∅;

    // Filter
    FOR i=1 TO n DO
        o = DB.getObject(i);
        IF dist_UB(q,o) ≤ ε THEN
            result := result ∪ o;
        ELSE IF dist_LB(q,o) ≤ ε THEN
            candidates := candidates ∪ o;

    // Refinement
    FOR i=1 TO candiates.size() DO
        c = candidates.getObject(i);
        IF dist(q,c) ≤ ε THEN
            result := result ∪ c;
    RETURN result;
```

2.3.2 Mehrstufige k-Nächste-Nachbarn Anfrage

- Allgemeines
  - Algorithmen verwenden meist nur LB-Filter
  - Bei mehreren Filterschritten: $dist_{Filter1} \leq dist_{Filter2} \leq \ldots$
  - Unterschied zu Bereichsanfragen:
    - » RQs können durch einfache Hintereinanderschaltung der Filterschritte und der Verfeinerung ausgewertet werden
    - » Bei NN-Queries nicht so leicht möglich, da der NN-Kandidat des (ersten) Filters nicht notwendigerweise der exakte NN sein muss
    - » Bei geeigneter Filterdistanz ist es aber wahrscheinlich, dass exakter NN unter den ersten NN-Kandidaten des Filterschritts ist
    - » Rückmeldung der im Refinement ermittelten Distanzen an den Filterschritt um aufgrund des Filters Objekte auszuschließen



- Es gibt verschiedene Auswertungsstrategien basierend auf LB-Filter
  - Auswertung mit Bereichsanfrage [Korn et al.,VLDB 1996]
  - Auswertung mit unmittelbarer Verfeinerung
  - Auswertung nach Priorität
- Auswertung mit Bereichsanfrage

  [Korn, Sidiropoulos, Faloutsos, Siegel, Protopapas. Proc. Int. Conf. Very Large Databases (VLDB), 1996]
  [Korn, Sidiropoulos, Faloutsos, Siegel, Protopapas. TKDE 10(6), 1998]
  - Idee
    - » Verfeinerungsdistanz ε eines beliebigen Punktes ist obere Schranke für die NN-Distanz
    - » Folge: ist p der NN von q, so gilt $dist(p, q) \leq \varepsilon$ und $LB_{Filter}(p, q) \leq \varepsilon$
    - » Also: $p \in RQ(q, \varepsilon)$
    - » Gutes ε ist z.B. der NN von q bzgl. der Filterdistanz
  - Prinzip
    - » Auf Filterebene wird zunächst eine NN-Anfrage ausgeführt
    - » Das resultierende Objekt wird verfeinert
    - » Anschließend wird eine Bereichsanfrage (RQ) ausgeführt (mit Index oder ebenfalls mehrstufig)
    - » Auf dem (hoffentlich kleinen) Ergebnis der RQ wird der exakte Test (Refinement) durchgeführt

– Algorithmus

NN-MultiStep-RQ(DB, *q*)
r = NN-Query auf der Filterdistanz; // beliebig implementierbar
ε = dist(*q*, *r*);
candidates = **RQ-MultiStep**(DB, *q*, ε);
result = *r* ;
stopdist = ε;

// Refinement
FOR EACH p∈candidates **DO**
IF dist(p,q) ≤ stopdist **THEN**
stopdist = dist(*q*, *p*)
result = *p*;
RETURN result;

– Vorteil
» Einfacher Algorithmus
– Nachteil
» Leistung stark von Filterselektivität abhängig: schlechter Filter => großes ε => große Ergebnismenge der RQ => hohe Kosten für Verfeinerung

- Auswertung mit unmittelbarer Verfeinerung
  - Idee
    - » Jedes Objekt, das nicht aufgrund des Filters ausgeschlossen werden kann, wird sofort verfeinert
    - » Einbau in einen beliebigen NN-Algorithmus, z.B. in NN-Index-Simple-TS (S. 61)
  - Algorithmus

```
NN-MultiStep-Simple(pa, q)        // pa = Diskadress z.B. der Wurzel des Indexes
    result = ∅;
    p := pa.loadPage();
    IF p.isDataPage() THEN
        FOR i=0 TO p.size() DO
            IF dist_Filter(q, p.getObject(i)) ≤ stopdist THEN
                IF dist(q, p.getObject(i)) ≤ stopdist THEN
                    result := getObject(i);
                    stopdist = dist(q, p.getObject(i));
    ELSE                          // p ist Directoryseite
        FOR i=0 TO p.size() DO
            IF MINDIST(q, p.getRegion(i)) ≤ stopdist THEN
                result := NN-MultiStep-Simple(p.childPage(i), q)
    RETURN result;
```

- Vorteil
  - » Gute Speicherplatzkomplexität (je nach NN-Algorithmus!!!), da keine Kandidaten zwischen gespeichert werden müssen
  - » Einfache Erweiterung eines beliebigen NN-Algorithmus
- Nachteil
  - » Hohe Verfeinerungskosten (fast alle Punkte), wenn Filter wenig selektiv ist oder NN-Algorithmus langsam konvergiert

- **Auswertung nach Priorität**

  [Seidl, Kriegel. Proc. ACM Int. Conf. Management of Data (SIGMOD),1998]

  - Auf Filterebene läuft „Ranking Query“ ab

    (Erweiterung von k-NN-Index-HS auf Folie 63)
    - » Funktion getNext(): liefert beim ersten Aufruf den 1. Nachbarn, beim zweiten Aufruf den 2. Nachbarn, usw.
    - » Rufe solange getNext() auf, bis das erhaltene Objekt die aktuelle pruningdist überschreitet.
    - » Verfeinere das erhaltene Objekt und passe ggf. die pruningdist an.
  - Vorteil
    - » Beweisbar: Algorithmus optimal bzgl. der Anzahl der Verfeinerungen, d.h. eine minimale Anzahl von Kandidaten werden verfeinert.
  - Nachteil
    - » Komplexität des Ranking-Algorithmus (Speicher und/oder Zeit)

– Algorithmus

k-NN-MultiStep-Optimal(DB, *q*)

Globale Variablen: pruningdist = +∞;

result = ∅; // Heap mit (o,dist(q,o)) tupel, erster Eintrag hat höchsten dist()-wert, maximale Heapgröße = k (Bei Überlauf wird letztes Element gelöscht).

Ranking = initialisiere Ranking bzgl. *q* auf Filterdistanz

FOR i=1..k **DO**
    p = Ranking.getNext();
    result.insert(p,dist(q,p)); // Verfeinerung
pruningdist = result.first.dist;
REPEAT
  p = Ranking.getNext();
  IF $dist_{LB}(p, q) \leq$ pruningdist **THEN** // Filter
    IF dist(*q*, *p*) ≤ pruningdist **THEN** // Verfeinerung
      result.insert(*p*);
      pruningdist = result.first.dist;
UNTIL $dist_{LB}(p, q) >$ pruningdist;
RETURN result;