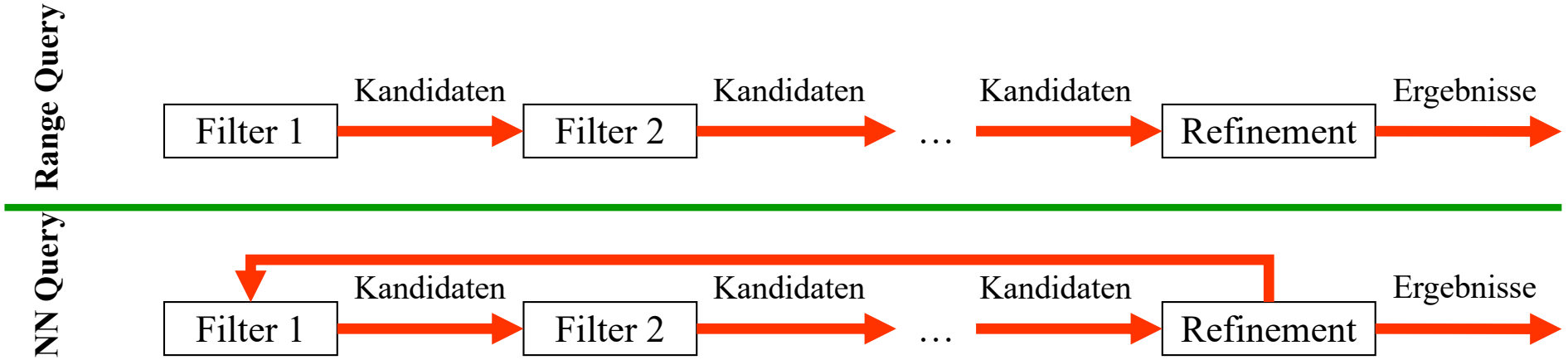
2.4.1 Nächste Nachbarn Anfragen

- Mehrstufiger Algorithmus: Filter/Refinement
  - Allgemeines
    - Algorithmen verwenden meist nur LB-Filter
    - Bei mehreren Filterschritten: $dist_{Filter1} \leq dist_{Filter2} \leq \ldots$
    - Unterschied zu Bereichsanfragen:
      - Auf der Basis des LB-Filers können RQs durch einfache Hintereinanderschaltung der Filterschritte und der Verfeinerung sequentiell ausgewertet werden.
      - Bei NN-Anfragen (NNQ) nicht so leicht möglich. Grund: Der (erste) Filter ist nicht in der Lage **auf der Grundlage des LB-Filters** eine NN-Kandidatenmenge (als kleine Teilmenge der Datenbank) zu identifizieren, die garantiert den exakten NN enthält.

2.4.1 Nächste Nachbarn Anfragen

- Mehrstufiger Algorithmus: Filter/Refinement
  - Allgemeines
    - Idee
      - 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.



2.4.1 Nächste Nachbarn Anfragen

- Im folgenden:
  - Verschiedene Auswertungsstrategien
  - Ein Filterschritt (leicht generalisierbar auf mehrere Filterschritte)
- **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

2.4.1 Nächste Nachbarn Anfragen

- 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

2.4.1 Nächste Nachbarn Anfragen

- **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 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;
```

2.4.1 Nächste Nachbarn Anfragen

- 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 (siehe Kapitel 2.4.3)
    - 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 stopdist überschreitet
    - Verfeinere das erhaltene Objekt und passe ggf. die stopdist 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)

2.4.1 Nächste Nachbarn Anfragen

- Algorithmus

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

Ranking = initialisiere Ranking bzgl. *q* auf Filterdistanz // Kapitel 2.4.3

result = ∅;

stopdist = +∞;

REPEAT

p = Ranking.getNext();

filterdist = $dist_{Filter}(p, q)$;

IF $dist_{Filter}(p, q) \leq$ stopdist **THEN**

IF $dist(q, p) \leq$ stopdist **THEN**

result = *p*;

stopdist = $dist(q, p)$;

UNTIL filterdist > stopdist;

RETURN result;