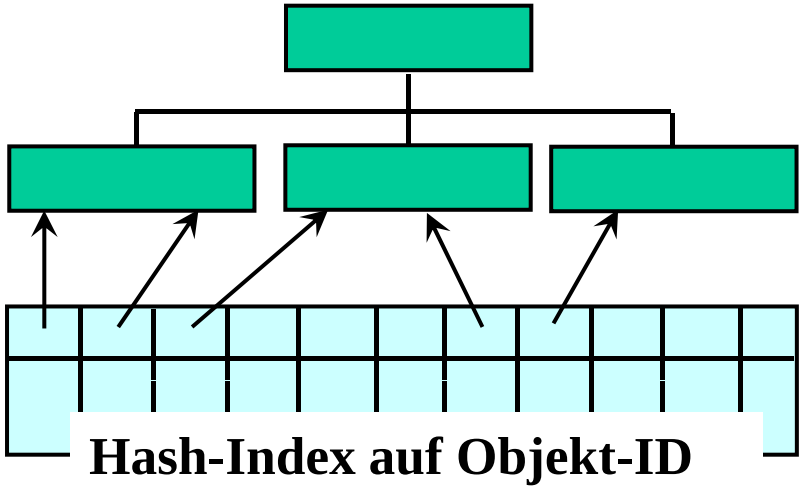
3.2.3 Anfragen an die Gegenwart

- Anfragen beziehen sich auf den aktuellen Zeitpunkt
- Beispiel:
  - Wieviele Objekte befinden sich momentan in Region A?
  - Wo hält sich gerade Objekt B auf?
- Eigenschaften:
  - Daten ändern sich kontinuierlich.
  - Anfragen müssen in Echtzeit durchgeführt werden
    => sehr schnelle Antwortzeiten.
  - Indexstrukturen sollten (sehr) effizient aktualisierbar sein.
  - Anfragen in der Gegenwart werden häufig für kontinuierliche ST-Anfragen (Continuous Queries) benutzt.

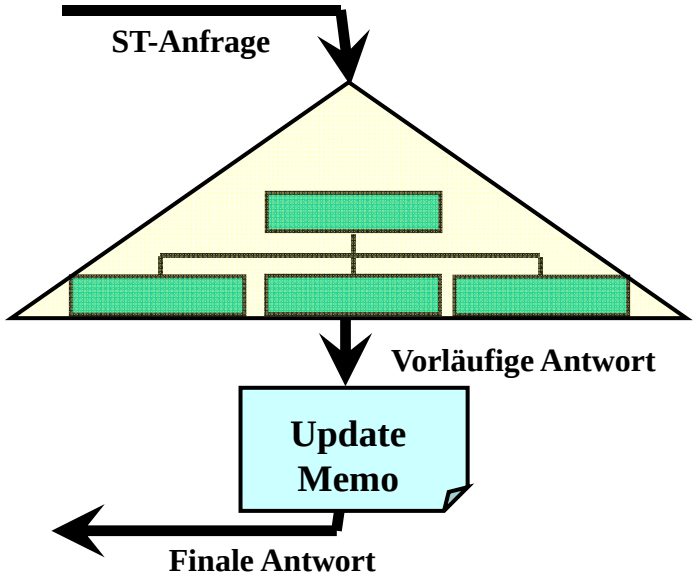
- Effiziente Verwaltung der Gegenwart
  - Eigenschaften von Änderungen (Updates) bei ST-Daten:
    - Sehr lokale Änderungen:
      Z.B. kann sich eine Person innerhalb einer Sekunde höchstens einige Meter von seinem letzten Standort entfernt haben.
    - Sehr hohe Update-Frequenz:
      Beobachtungen von kontinuierlich Bewegungen müssen sehr oft aktualisiert werden.
  - Traditionelle R-Baum-updates werden Top-Down durchgeführt.
  - Desweiteren werden gewöhnlich Änderungen als Lösch- und Einfügeoperation implementiert.
  - Fazit: Lokale Änderungen werden bei dem Top-Down-Update Prozess nicht gut unterstützt.
  - 2 Ansätze zur effizienten Unterstützung von (lokalen) Positionsänderungen:
    - Bottom-Up Update Strategie → Beschleunigung des Update Prozesses
    - Lazy-Update Strategie → Veringert die Update-Rate

– Bottom-Up Update Strategie: [LHJCT03] [KLL02]
  - Idee: Änderung werden direkt auf Blattebene durchgeführt.
    → Vermeidung von unnötigen Lösch- und Einfügevorgängen.
  - Daten (Objekte) die zu ändern sind werden direkt über einen Hilfsindex (Hash-Index auf Objekt-ID) direkt auf der Blattebene des R-Baums zugegriffen und geändert.
  - Eine Änderung wird schrittweise nach oben Richtung Wurzel fortgesetzt.



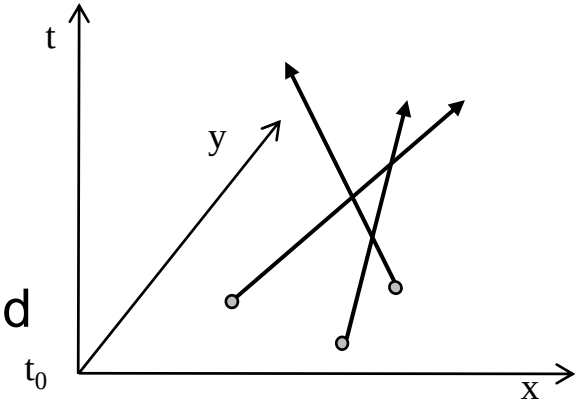
– Lazy-Update Strategie: [SXA09]

- Änderungen werden **nicht** sofort als Lösch/Einfüge-Proceduren an den R-Baum weitergegeben, es wird “zunächst” nur eingefügt.
- “Aktuelle” Änderungen (Unterscheidung zwischen veralteten und aktuellen Daten) werden zunächst in einer Hilfsstruktur (Update-Memo) temporär verwaltet.
- Nach einen gewissen Zeitraum wird der R-Baum mit den neuen Daten aus dem Update-Memo aktualisiert. Diese Daten werden aus den Update Memo gelöscht.
- Eine ST-Anfrage erfolgt zunächst auf (teilweise veralteten) Daten die im R-Baum verwaltet sind.
- Die finale Antwort wird aus den (Superset-)Ergebnissen des R-Baums und der Information im Update-Memo gebildet.



3.2.4 Anfragen an die nahe Zukunft

- Beispiel:
  - Wer befindet sich in der nächsten Stunde in Region A?
  - Welche Objekte treffen in der nächsten Stunde aufeinander?
- Eigenschaften:
  - Zukünftige Bewegung von Objekten wird über Geschwindigkeitsvektoren vorhergesagt.
  - Die Vorhersage ist gültig bis zu einem gewissen Zeithorizont.
- Darstellung der nahen Zukunft
  - Zeit wird wieder als zusätzliche Dimension betrachtet.
  - Zukünftige Bewegung über aktuelle Position (Position zum Zeitpunkt t_0) und **linearer** Bewegungsrichtung (gültig bis zu einem bestimmten Zeithorizont oder der nächsten Aktualisierung der Position (Update)).

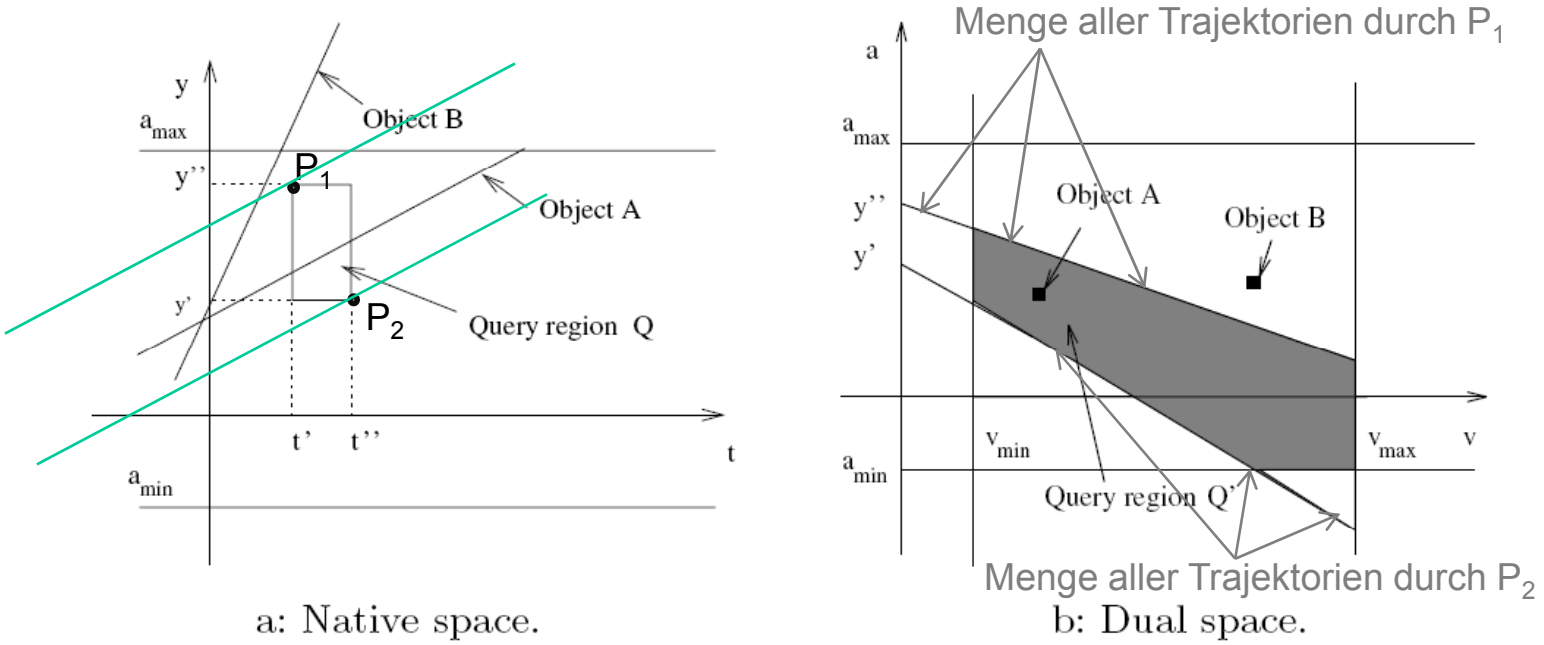


– Effiziente Verwaltung der Zukunft
  - Indexstrukturen die lineare Bewegungen effizient Verwalten können:
    – Indexstrukturen basierend auf Dual-Raum Transformation: Beispiele: **STRIPES** [PCC04], MB-Index [EEK03].
    – Indexstrukturen basierend auf zeit-parametrisierte Seitenregionen: Beispiele: **TPR-Baum** [SJLL00] und Varianten.
    – Nur für Anfragen in die Zukunft, keine Verwaltung der Vergangenheit.
  - Dual-Raum Transformation

    Idee:
    – Transformation aller Trajektorien der Form y(t) = vt + a in den Dual-Raum V x A (V = Menge aller Geschwindigkeiten, A = Menge aller Positionen im (nativen) Raum)

      Trajektorie y(t) = vt + a → Punkt (v,a).
    – Betrachtung von nur positiven Geschwindigkeiten. Dazu werden alle Geschwindigkeiten vorab normalisiert mit $v' = v + v_{max} \Rightarrow v' \in [0,2v_{max}]$.
    – Verwaltung der Trajektorien im Dual-Raum über Punktzugriffsstrukturen (z.B: Quadtree, R-Baum, etc.)
    – Anfragerechteck → Anfrage-Polygon im Dual-Raum

– Beispiel:



a: Native space. b: Dual space.

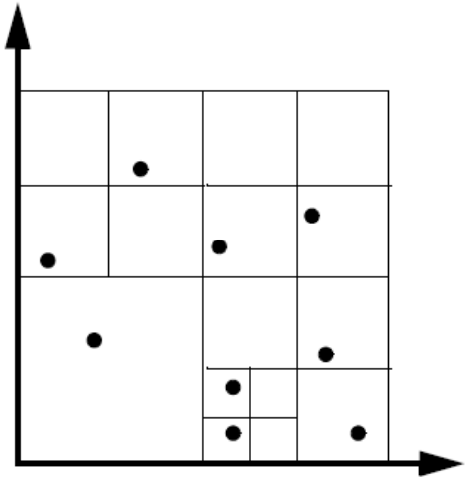
– Trajektorien im ursprünglichen Raum (native space) als Punkte im Dual-Raum (dual space).
– Fensteranfrage im ursprünglichen Raum entspricht Polygon im Daul-Raum.

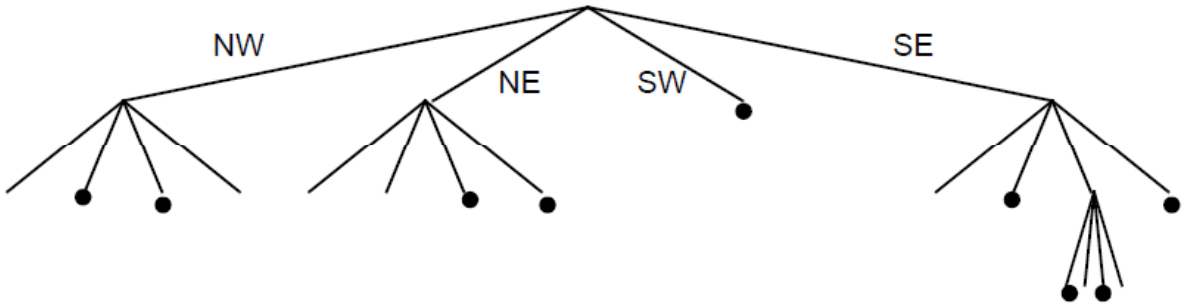
  Für jede Trajektorie y=vt+a die das Anfrage-Fenster schneidet:

  » existiert eine Trajektorie y=vt+a'' mit der gleichen Geschwindigkeit die durch Punkt P_1 geht mit a<a'', d.h. a'' = y'' – vt' für P_1=(t',y'').

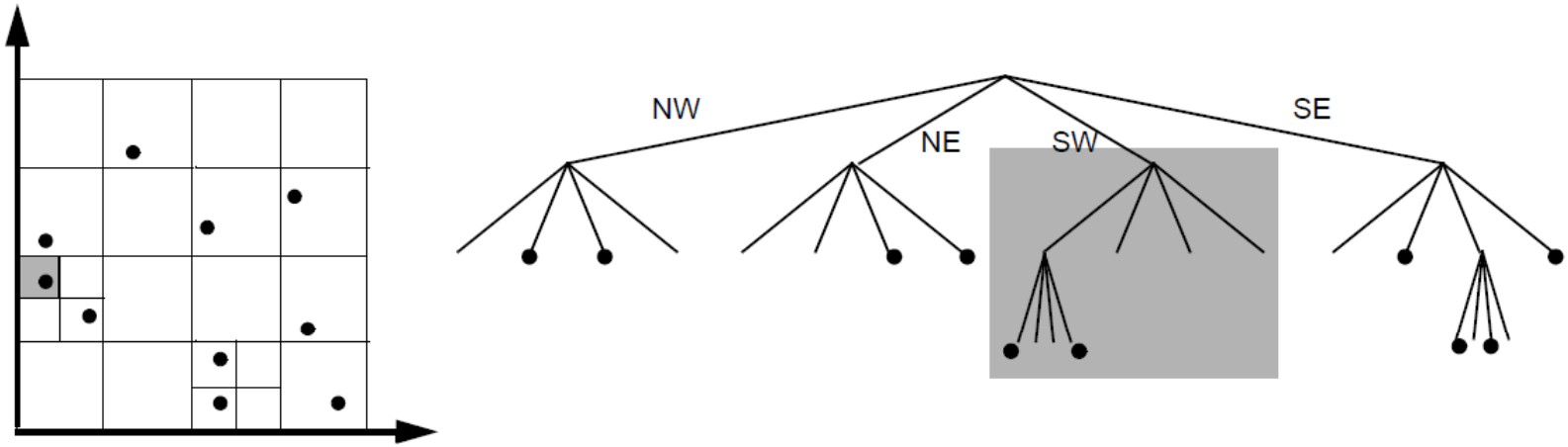
  » existiert eine Trajektorie y=vt+a' mit der gleichen Geschwindigkeit die durch Punkt P_2 geht mit a'<a, d.h. a' = y' – vt'' für P_2=(t'',y').

– Der STRIPES Index [PCC04]
  - Basiert auf einer Sekundärspeicher-basierten Point-Region (PR) Bucket Quadtree Struktur.
  - Exkurs: Point-Region Quadtree
    – variable Auflösung des Datenraums
    – Komprimierung eines internen Knotens falls im Teilbaum höchstens m Datensätze vorhanden sind (m = Kapazität einer Datenseite)
    – Dann werden die Datensätze direkt in dem internen Knoten abgespeichert und dessen vier Kinderknoten werden freigegeben.
    – Jeder interne Knoten besitzt mindestens (m+1) Punkte in den darunterliegenden Teilbäumen.





– Einfügen



– Suche den Einfügeknoten N.
– Falls N weniger als m Punkte enthält, so füge den Datensatz in N ein.
– Andernfalls teile den Datenraum des Teilbaums von N solange rekursiv auf, bis die enthaltenen Punkte in mind. zwei unterschiedlichen Quadranten (Knoten) aufgeteilt sind.

– Der STRIPES Index (Fortsetzung)

- Basiert auf einer Sekundärspeicher-basierten Point-Region (PR) Bucket Quadtree Struktur.
- Für den (d+1) dimensionalen (Original-) Raum (native space) betrachte man d Dual-Unterräume.
- Jeder dieser d Dual-Unterräume wird jeweils in 4 gleichgroße Quadranten unterteilt.
- Der Dual-Raum wird somit bei jeder Quadtreezellenzerlegung in $4^d = 2^{2d}$ Partitionen zerlegt.
- Jeder interne Knoten des Quadtrees hat somit 4^d Kinder.
- Jeder Blattknoten enthält bis zu m Punktobjekte.
- Ein Blattknoten speichert zudem:
  - *level*: Level auf dem sich der Blattknoten befindet (= definiert die Seitenregion).
  - *grid*: Angabe um welche Quadtree-Zelle es sich bei dem Knoten handelt.
  - *size*: Die Anzahl der enthaltenen Punktobjekte.

- Ein interner Knoten speichert:
  - *level*: Level auf dem sich der interne Knoten befindet (= definiert die Seitenregion).
  - *grid*: Angabe um welche Quadtree-Zelle es sich bei dem Knoten handelt.
  - *children_ptr*: Array der Größe 2^{2d} mit den Zeigern zu den Kindknoten.
  - *isleaf_array*: Array der Größe 2^{2d} mit Angabe ob die Zeiger auf die Kindseiten auf interne Knoten oder auf Blattknoten zeigen.
  - *size*: Die Anzahl der in den darunterliegenden Teilbaum enthaltenen Punktobjekte.

- Einfügen in STRIPES:
  - Suche rekursiv nach dem entsprechenden Blattknoten.
  - Falls das Blatt Platz hat füge den Punkt ein.
  - Ansonsten: Wandle den Blattknoten in ein internen Knoten und erzeuge 2^{2d} Kindknoten auf die die Einträge des Blattknotens entsprechend der Partitionierung aufzuteilen sind. (evtl. muß der Split rekursiv fortgesetzt werden).

- Entfernen in STRIPES:
  - Suche rekursiv nach dem entsprechenden Blattknoten.
  - Falls bei dem Durchlauf der internen Knoten die Anzahl der darunterliegenden Punkte (Attribut *size*) die Zahl m+2 unterschritten wird, werden alle darunterliegenden Einträge gesammelt, der aktuelle Knoten zum Blattknoten gewandelt und die gesammelten Einträge dort eingefügt.
  - Der Eintrag wird in dem final erreichten Blattknoten gelöscht.

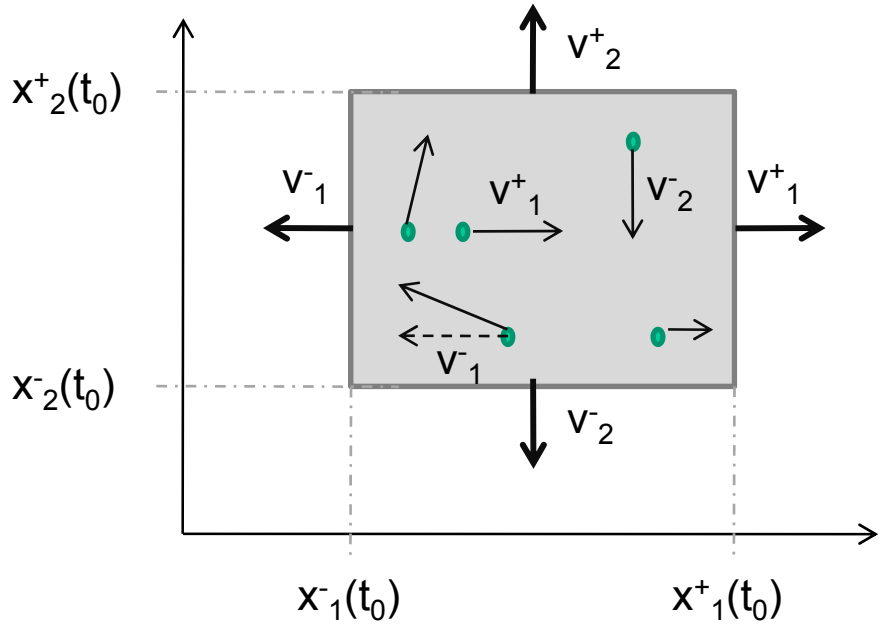
- Anfragen in STRIPES:
  - Unterstützte Anfragen:
    - » Time-Slice Anfrage (Anfragen beziehen sich auf einen Zeitpunkt).
    - » ST Fenster Anfrage (siehe Übung) (Anfragen beziehen sich auf ein Zeitintervall).
    - » Dynamische ST Anfrage (ST Fenster Anfragen wobei das räumliche Anfragefenster über die Zeit hinweg variiert).

– Der Time-Parametrized R-Baum (TPR-Baum) [SJLL00]
  - R-Baum-Variante, d.h. Prinzip der überlappenden Seitenregionen.
  - Dimensionalität der Seitenregionen äquivalent zur Dimensionalität des räumlichen Attributs.
  - Prinzip der zeitparametrisierten Seitenregionen
    – Seitenregionen speichern neben der Ausdehnung zur Zeit t_0 einen Vektor über die (lineare) Veränderung der Ausdehnung über die Zeit.
    – Koordinaten der Seitenregionen sind Funktionen der Zeit.
  - Eine Seitenregion garantiert die konservative Approximation der enthaltenen Objekte, solange der Bewegungsvektor der enthaltene Objekte konstant bleibt.

- Seitenregion eines Blattknotens
  - Achsenparalleles Rechteck, mit zeitparametrisierten Koordinaten

    $[(x^-_1(t),x^+_1(t)),(x^-_2(t),x^+_2(t)),..,(x^-_d(t),x^+_d(t))]$ wobei

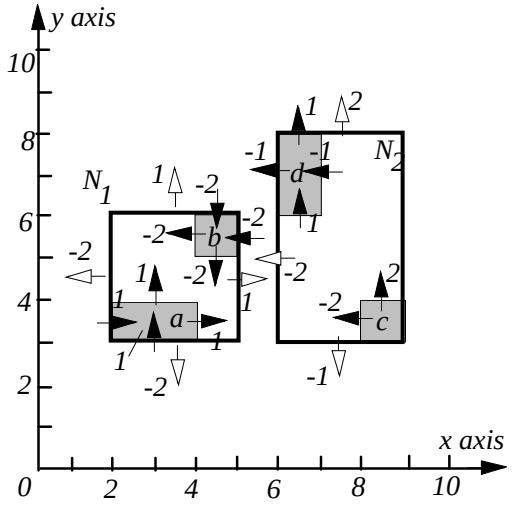
    $(x^-_i(t),x^+_i(t)) = (x^-_i(t_0) + v^-_i(t-t_0)\ ,\ x^+_i(t_0) + v^+_i(t-t_0))$, $1 \le i \le d$.



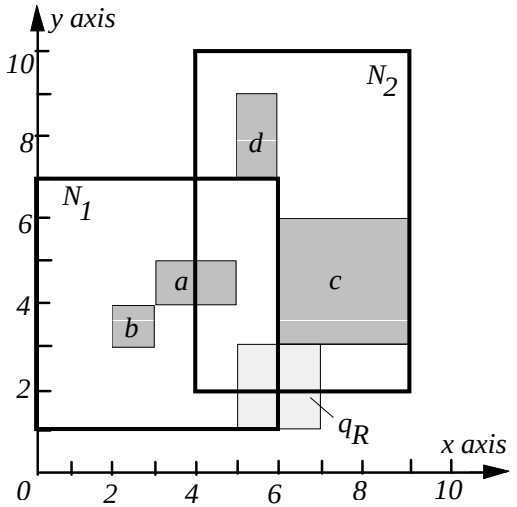
  - v^-_i (v^+_i): minimale (maximale) Geschwindigkeit der enthaltenen Objekte bzgl. Dimension $1 \le i \le d$.

    (Vorzeichen der Geschwindigkeit bestimmt die Bewegungsrichtung, z.B. negative Geschwindigkeit -5m/s bedeutet Objekt bewegt sich nach links mit der Geschwindigkeit 5m/s)

- Seitenregion eines Directory Knotens
  - Ebenfalls Achsenparalleles Rechteck, mit zeitparametrisierten Koordinaten
  - v^-_i (v^+_i): minimale (maximale) Ausdehnungs-Geschwindigkeit der enthaltenen Seitenregionen.
- Eigenschaften der Seitenregionen:
  - Zum **Zeitpunkt t_0** (Gegenwart) werden alle enthaltenen Objekte **minimal** umfasst.
  - Zum **Zeitpunkt $t > t_0$** werden die Objekte zwar **konservativ** aber **nicht** mehr unbedingt **minimal** umfasst
  - Beispiel:

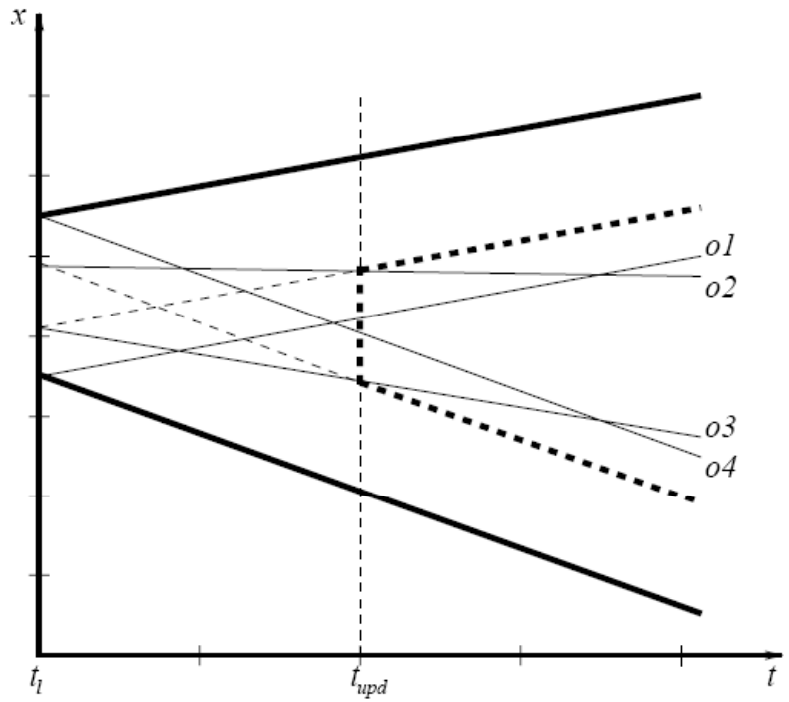


at time 0

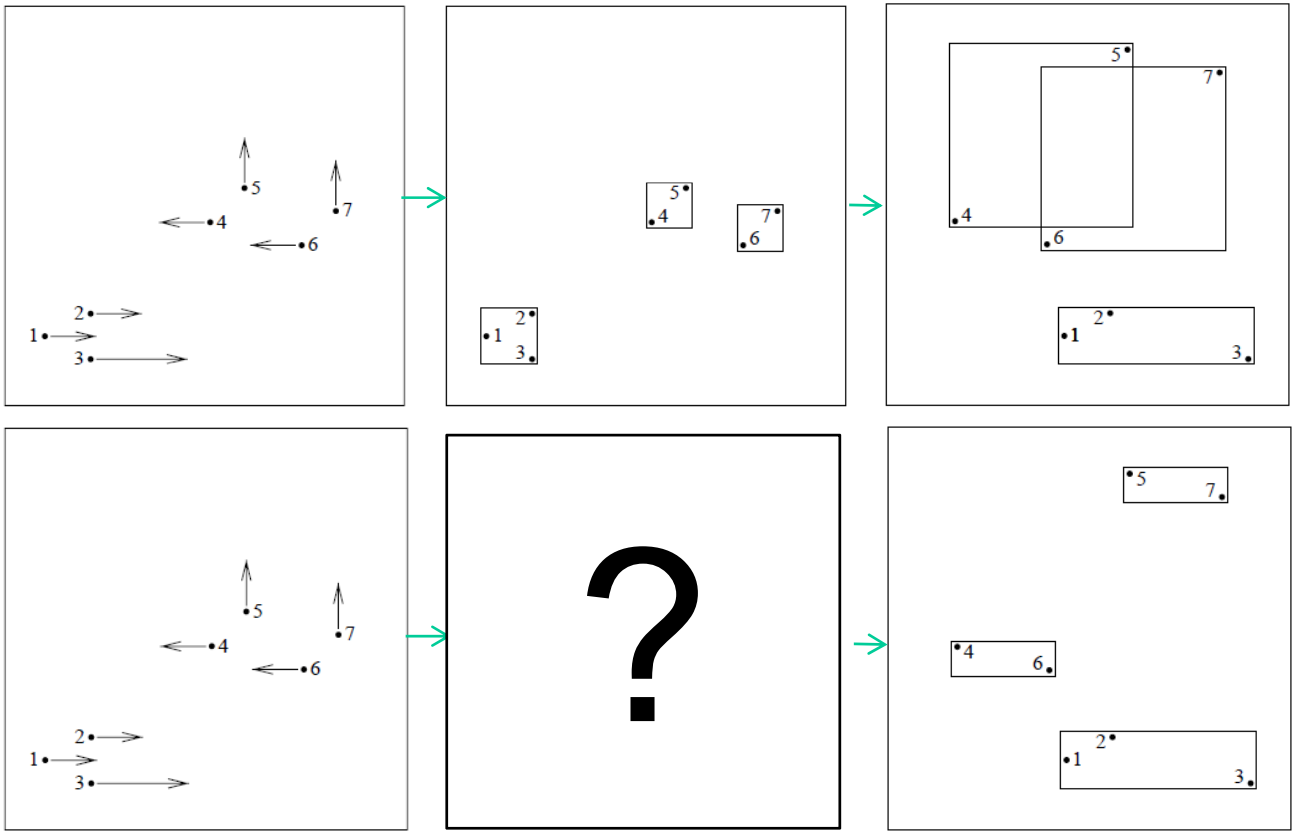


at time 1

- Aktualisierung der Seitenregionen
  - Seitenregionen wachsen kontinuierlich
    => kontinuierliche Degenerierung der Approximationsgüte
  - Seitenregion ist solange gültig bis zu einem Zeitpunkt t_{upd} (Zeithorizont) zu dem es sich lohnt diese zu erneuern
    => Steigerung der Approximationsgüte
  - Beispiel:



- Einfügestrategie
  - Welche Objekte sollen zusammen in einen Knoten eingefügt werden?
  - Räumliche Nähe nicht unbedingt optimal:



Bewegung der Objekte | Seitenregionen zum Zeitpunkt t_0 | Seitenregionen zum Zeitpunkt $t>t_0$

  - Performanz des TPR-Baum stark abhängig vom Zeithorizont H:
    - » H ≈ 0 -> Anfragen beziehen sich auf Zeitpunkte nahe bei Referenzzeitpunkt t_0, damit kein negativer Einfluss veränderter Bounding Rectangles durch hohe Überschneidung.
    - » H >> 0 -> Gruppiere Punkte so, dass die Größe der Seitenregionen zu allen Zeiten $[t_0, t_0+H]$ möglichst gering ist.
  - Gute Performanz, wenn v_{min}-v_{max} für jede Dimension gering ist (Seitenregion verändert dann ihre Größe nur geringfügig)
  - Minimiere Überschneidung, Umfang und Volumen wie bei der R*-Baum Einfügestrategie (beachte aber dabei auch die Zeitliche Komponente in $[t_0, t_0+H]$)

- Unterstützte Anfragen:
  - Time-Slice Anfragen (Anfragen beziehen sich auf einen Zeitpunkt).
  - ST Fenster Anfragen (Anfragen beziehen sich auf ein Zeitintervall).
  - Dynamische Anfragen (ST Fenster Anfragen wobei das räumliche Anfragefenster über die Zeit hinweg variiert).

- Diskussion:
  - Nachdem der TPR-Baum Bewegungen vorhersagt sind die Ergebnisse einer Anfrage immer approximativ.
  - Objekte im TPR-Baum können zur Laufzeit korrigiert werden
  - Somit hängt das Ergebnis einer Anfrage von dem Zeitpunkt an dem die Anfrage abgesetzt wurde ab und kann somit unterschiedliche Ergebnisse liefern!
  - Anfragen die sich auf Zeiträume weit in der Zukunft beziehen sind somit weniger sinnvoll.

– Gesamt-Überblick über S-, T- und ST-Indexstrukturen

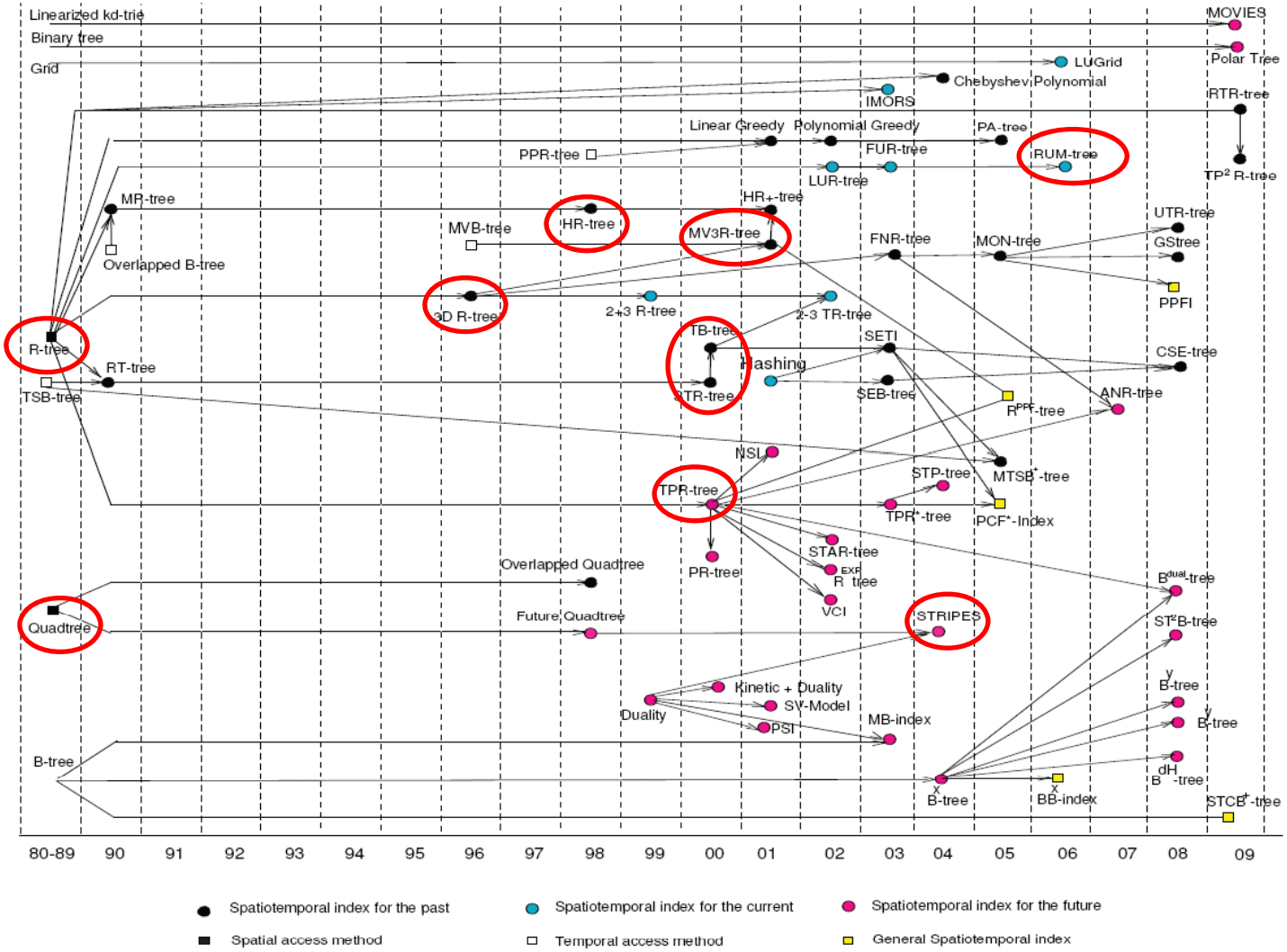


Figure 1: Survey of Spatiotemporal Access Methods.

[Quelle: NAM10]

– Referenzen:

– [TVS96] Yannis Theodoridis, Michael Vazirgiannis, and Timos Sellis. Spatio-temporal Indexing for Large Multimedia Applications. In Proceeding of the IEEE Conference on Multimedia Computing and Systems, ICMCS, pages 441-448, Hiroshima, Japan, June 1996.

– [PJT00]: D. Pfoser, C. S. Jensen, Y. Theodoridis. Novel Approaches to the Indexing of Moving Object Trajectories. In Proc. of VLDB, 2000.

– [PCC04]: J.Patel,Y.Chen,andV.Chakka. STRIPES: An efficient index for predicted trajectories. SIGMOD 2004.

– [EEK03]: K.M.Elbassioni, A.Elmasry, and I.Kamel. An efficient indexing scheme for multi-dimensional moving objects. In ICDT,pages 422–436, 2003.

– [SJLL00]: S.Saltenis, C.Jensen, S.Leutenegger, and M.Lopez. Indexing the positions of continuously moving objects. In SIGMOD, pages 331–342, 2000.

– [LHJCT03]: M. Lee, W. Hsu, C. Jensen, B. Cui, and K. Teo. Supporting Frequent Updates in R-Trees: A Bottom-Up Approach. In Proc. of the Intl. Conf. on Very Large Data Bases, VLDB, Sept. 2003.

– [SXA09]: Y. N. Silva, X. Xiong, and W. G. Aref. The RUM-tree: Supporting frequent updates in R-trees using memos. VLDB J., 18(3):719–738, 2009.

– [KLL02]: D. Kwon, S. Lee, and S. Lee. Indexing the Current Positions of Moving Objects Using the Lazy Update R-tree. In Mobile Data Management, MDM, pages 113–120, Jan. 2002.

Gute Überblicksarbeiten:

– [MGA03]: Mohamed F. Mokbel , Thanaa M. Ghanem , Walid G. Aref: Spatio-Temporal Access Methods (2003), IEEE Data Eng. Bull., 26(2):40–49, 2003.

– [NAM10]: Long-Van Nguyen-Dinh, Walid G. Aref, Mohamed F. Mokbel: Spatio-Temporal Access Methods: Part 2 (2003 - 2010). IEEE Data Eng. Bull. 33(2): 46-55 (2010).