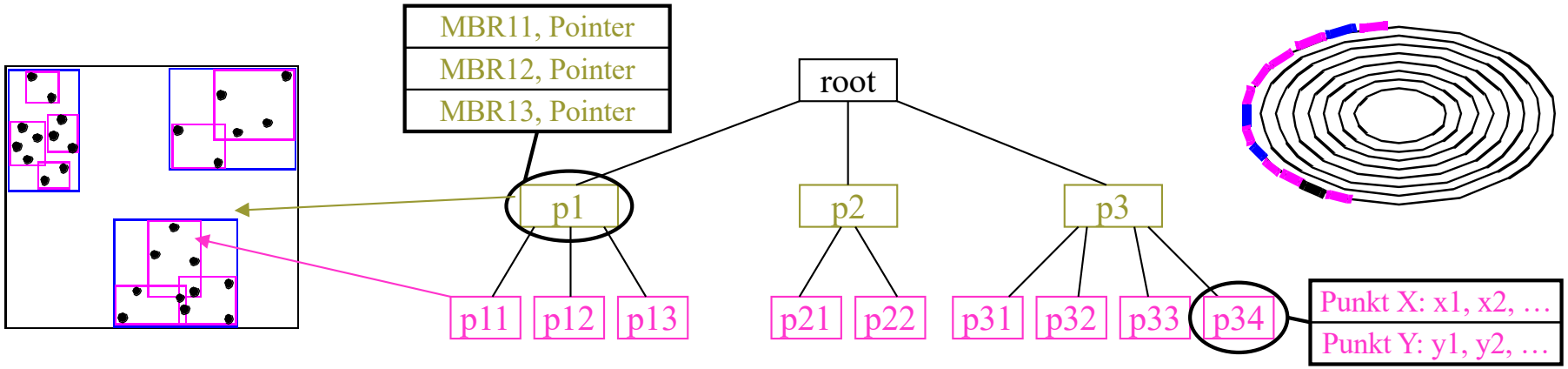
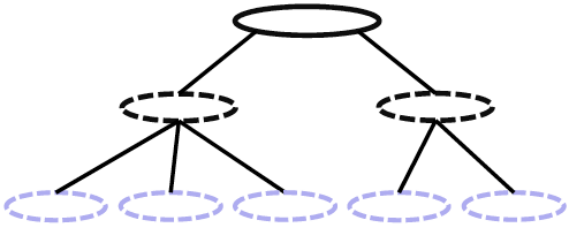
2.2.2 Indexstrukturen

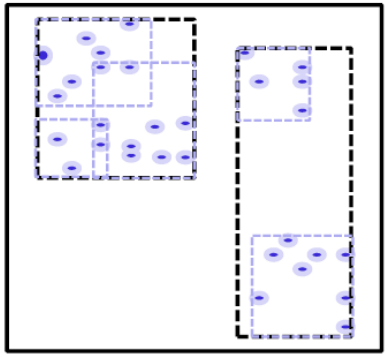
- Beispiel: R*-tree
  - Entworfen für 2D Rechtecksdaten (Punkte sind spezielle Rechtecke), oft verwendet für hochdimensionale Vektordaten
  - Design
    - Balancierter Index mit einheitlich großen Daten-/Directoryseiten
    - Seitenregionen: MBRs
    - Insert-Strategie: Overlap, Volumen
    - Überlaufbehandlung: Forced Re-Insert-Konzept
    - Splitkriterien: Umfang/Oberfläche, Überlappungsvolumen, (toter Raum)



2.2.2 Indexstrukturen

- Struktur eines R-Baums:
  - Wurzel umgibt den gesamten Datenraum und hat maximal M Einträge
  - Seitenregionen werden durch minimal umgebende Rechtecke (MUR) modelliert
  - innere Knoten im R-Baum haben zwischen m und M Nachfolger (wobei $m \leq M/2$)
  - Das MUR eines Nachfolgers ist vollständig im MUR des Vorgängers enthalten
  - Alle Blätter sind auf dem gleichen Level
  - Datenobjekte werden in den Blättern gespeichert
  - Mögliche Datenobjekte:
    - Punkte
    - Rechtecke





2.2.2 Indexstrukturen

- Das Objekt x ist in einen R-Baum einzufügen

  Durch Überlappung können 3 Fälle auftreten:

  - **Fall 1**: x fällt in genau ein Directory-Rechteck D

    ⇒ Einfügen in Teilbaum von D

  - **Fall 2**: x fällt in mehrere Directory-Rechtecke D1, ... , Dn

    ⇒ Einfügen in Teilbaum von Di, das die geringste Fläche aufweist

  - **Fall 3**: x fällt in kein Directory-Rechteck

    ⇒ Einfügen in Teilbaum von D, das den geringsten Flächenzuwachs erfährt (in Zweifelsfällen, das die geringste Fläche hat)

    - ⇒ D muss entsprechend vergrößert werden

2.2.2 Indexstrukturen

(im Folgenden wird von inneren Knoten ausgegangen: Objekte sind MURs)

Der Knoten K läuft mit $|K| = M+1$ über:

$\Rightarrow$ Aufteilung auf zwei Knoten K_1 und K_2, so dass $|K_1| \geq m$ und $|K_2| \geq m$

- **Quadratischer Algorithmus**
  - 1. Wähle das Paar von Rechtecken R_1 und R_2 mit dem größten Wert für den „toten Raum" im MUR, falls R_1 und R_2 in denselben Knoten K_i kämen.

    $d(R_1, R_2) :=$ Fläche(MUR($R_1 \cup R_2$)) - Fläche(R_1) - Fläche(R_2)
  - 2. Setze $K_1 := \{R_1\}$ und $K_2 := \{R_2\}$
  - 3. Wiederhole den folgenden Schritt bis zum STOP:
    - wenn alle R_i zugeteilt sind: STOP
    - wenn alle restlichen R_i benötigt werden, um den kleineren Knoten minimal zufüllen: teile sie alle zu und STOP
    - sonst: wähle das nächste R_i und teile es dem Knoten zu, dessen MUR den kleineren Flächenzuwachs erfährt. Im Zweifelsfall bevorzuge den K_i mit kleinerer Fläche des MUR bzw. mit weniger Einträgen.

2.2.2 Indexstrukturen

- **Linearer Algorithmus**
  - Der lineare Algorithmus ist identisch mit dem quadratischen Algorithmus bis auf die Auswahl des initialen Paars (R_1, R_2).
  - Wähle das Paar von Rechtecken R_1 und R_2 mit dem „größten Abstand", genauer:
    - Suche für jede Dimension das Rechteck mit dem kleinsten Maximalwert und das Rechteck mit dem größten Minimalwert (maximaler Abstand).
    - Normalisiere den maximalen Abstand jeder Dimension, indem er durch die Summe der Ausdehnungen der R_i in der Dimension dividiert wird (setze den maximalen Abstand der Rechtecke ins Verhältnis zur ihrer Ausdehnung).
    - Wähle das Paar von Rechtecken mit dem größten normalisierten Abstand bzgl. Aller Dimensionen. Setze $K_1 := \{R_1\}$ und $K_2 := \{R_2\}$.

Dieser Algorithmus ist linear in der Zahl der Rechtecke (2m+1) und in der Zahl der Dimensionen d.

2.2.2 Indexstrukturen

- **Idee der R*-Baum Splitstrategie**
  - sortiere die Rechtecke in jeder Dimension nach beiden Eckpunkten und betrachte nur Teilmengen nach dieser Ordnung benachbarter Rechtecke

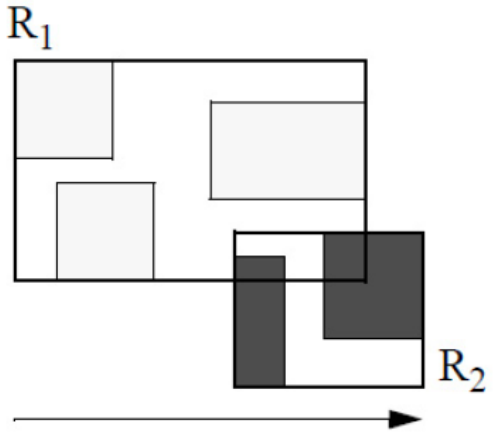
  Laufzeitkomplexität ist $O(d \cdot M \cdot \log M)$ für d Dimensionen und M Rechtecke

- Bestimmung der Splitdimension
  - Sortiere für jede Dimension die Rechtecke gemäß beider Extremwerte
  - Für jede Dimension:
    - Für jede der beiden Sortierungen werden M-2m+2 Aufteilungen der M+1 Rechtecke bestimmt, so dass die 1. Gruppe der j-ten Aufteilung die ersten m-1+j Rechtecke und die 2. Gruppe die übrigen Rechtecke enthält
    - U_G sei die Summe aus dem Umfang der beiden MURs R_1 und R_2 um die Rechtecke der beiden Gruppen
  - U_S sei die Summe der U_G aller berechneten Aufteilungen
  - ⇒ Es wird die Dimension mit dem geringsten U_S als Splitdimension gewählt.

2.2.2 Indexstrukturen

- **Bestimmung der Aufteilung**
  - Es wird die Aufteilung der gewählten Splitdimension genommen, bei der R_1 und R_2 die geringste Überlappung haben.
  - In Zweifelsfällen wird die Aufteilung genommen, bei der R1 und R2 die geringste Überdeckung von toten Raum besitzen.

    ⇒ Die besten Resultate hat bei Experimenten m = 0,4·M ergeben.



2. Aufteilung
(bei M = 4, m =2)

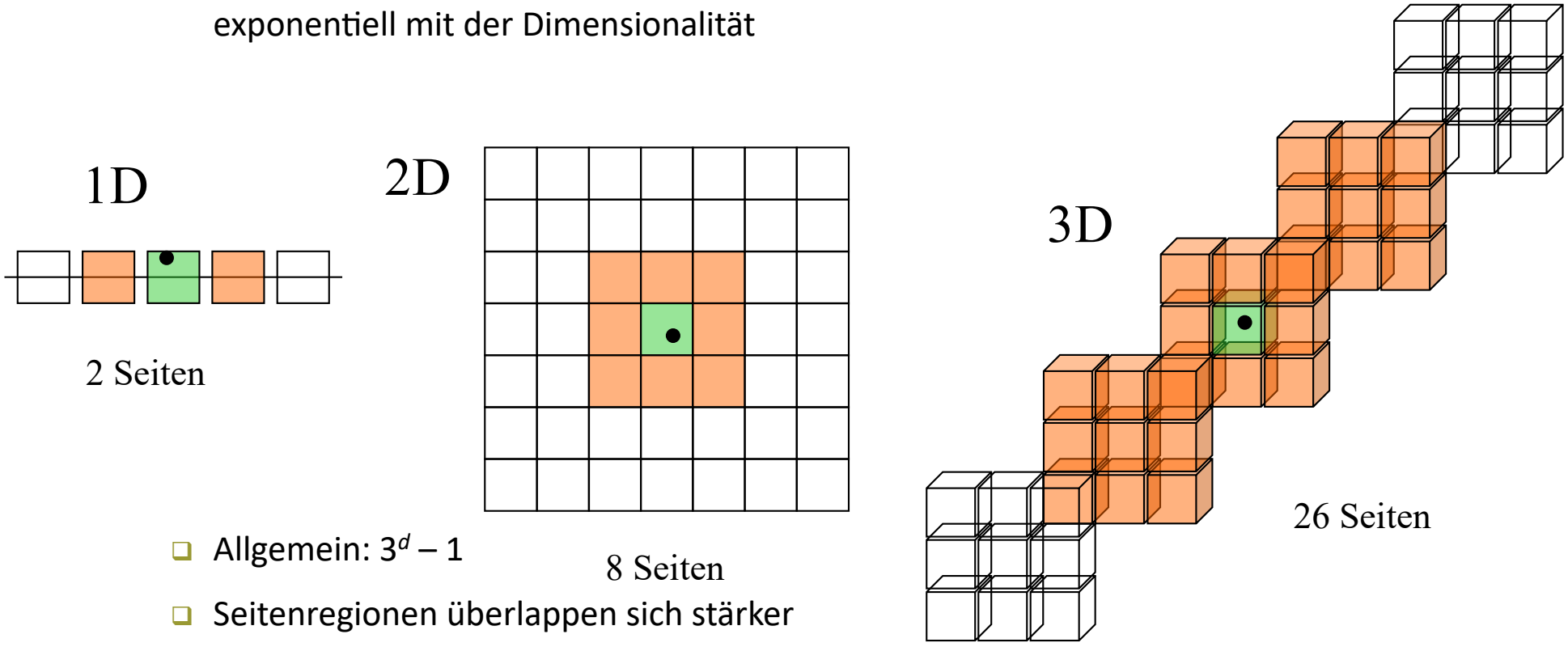
UG =
Umfang von R_1 +
Umfang von R_2

2.2.2 Indexstrukturen

- Das Objekt x ist aus dem R-Baum zu löschen
- Löschen :
  - Teste ob Seite S nach entfernen von x unterfüllt ist: |S| < m
  - Falls nicht, entferne x und STOP
  - Falls ja bestimme, welche Vorgängerknoten ebenfalls unterfüllt sind
  - Für jeden unterfüllten Knoten:
    - Lösche die unterfüllte Seite aus dem Vorgängerknoten
    - Füge die restlichen Elemente der Seite in den R-Baum ein
    - Falls die Wurzel nur noch 1 Kind enthält wird Kind zur neuen Wurzel (Höhe verringert sich)
- Bemerkungen:
  - Löschen ist mit diesem Algorithmus nicht auf einen Pfad beschränkt
  - Im Worst Case sehr teuer

2.2.2 Indexstrukturen

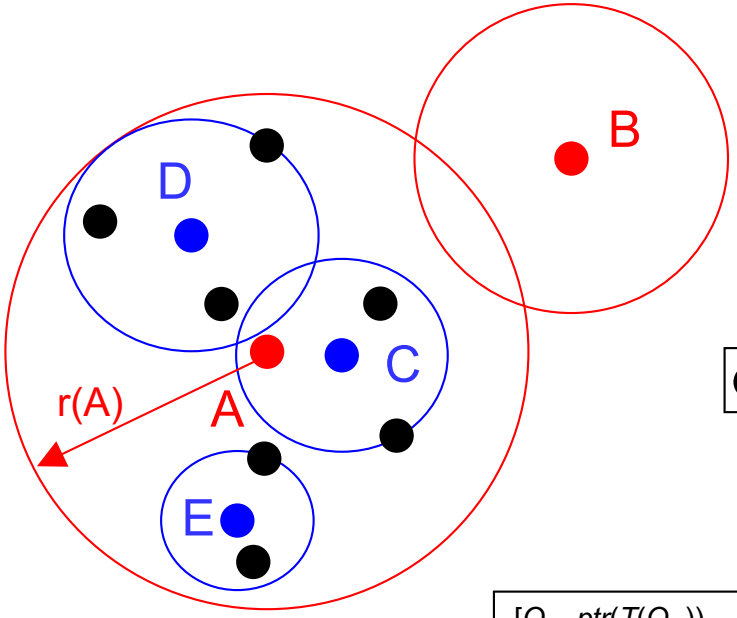
- Curse of Dimensionality (Vektordaten)
  - Anfrageleistung von Indexstrukturen verschlechtern sich mit zunehmender Dimension
    - Anzahl Seiten, die zugegriffen werden müssen, um Ergebnis zu garantieren, steigt exponentiell mit der Dimensionalität

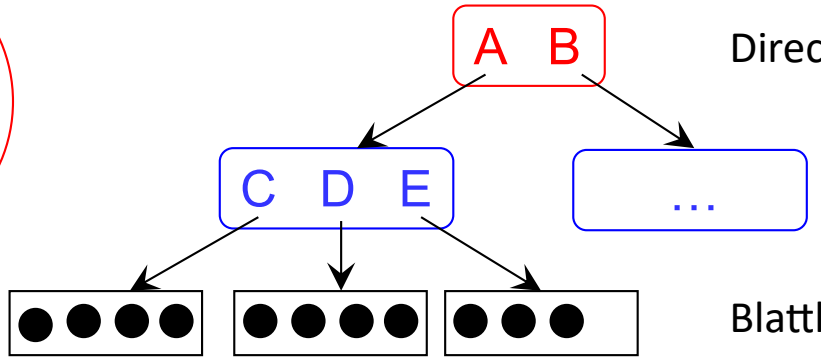


- Allgemein: $3^d - 1$
- Seitenregionen überlappen sich stärker
- Folge: oft muss komplettes Directory gelesen werden

2.2.2 Indexstrukturen

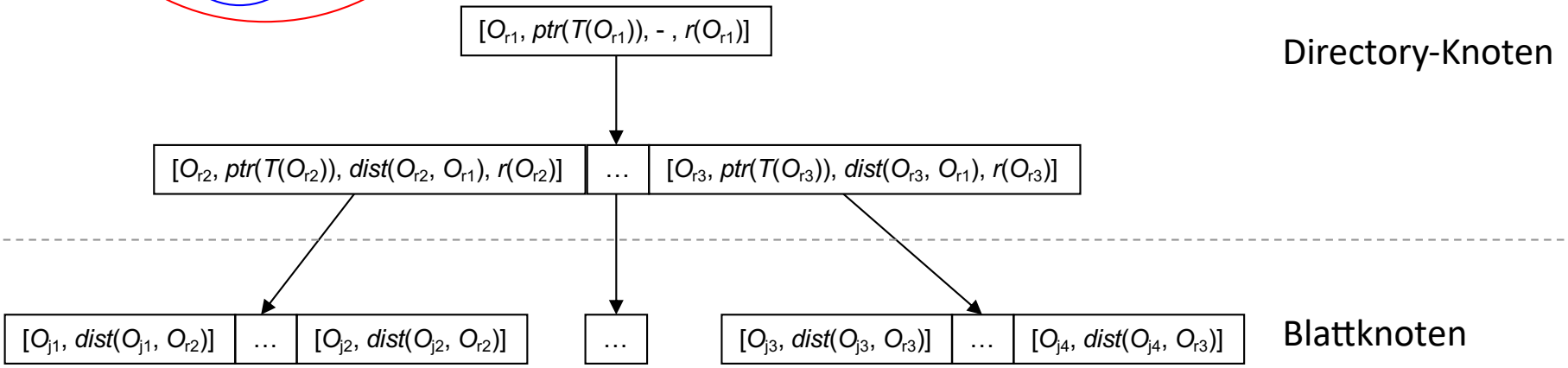
- M-tree Struktur





Directory-Knoten

Blattknoten



Directory-Knoten

Blattknoten

2.2.2 Indexstrukturen

- M-tree
  - Dynamische Indexstruktur für allgemeine metrische Räume
  - Die Distanzfunktion zur Berechung der Ähnlichkeit zweier Objekte muss die Eigenschaften einer Metrik erfüllen
  - Design
    - Balancierter Index mit einheitlich großen Daten-/Directoryseiten
    - Die indexierten Datenbank-Objekte werden in den Blattknoten abgespeichert
    - Die Directory-Knoten enthalten sog. *Routing Objekte*
    - Routing Objekte entsprechen Databank-Objekten, denen eine *Routing Rolle* zugewiesen wurde
    - Wenn ein Knoten überläuft und geplittet werden muß, vergibt der Splitalgoritmus eine Routing Rolle an ein Objekt
    - Zusätzlich zur Objektbeschreibung enthält ein Routing Objekt einen Zeiger auf seinen zugehörigen Unterbaum und den Radius, in dem sich alle Objekte des Unterbaums befinden
    - Wahl der Routing Objekte: die beiden am weitesten voneinander entfernt liegenden Objekte der übergelaufenen Seite