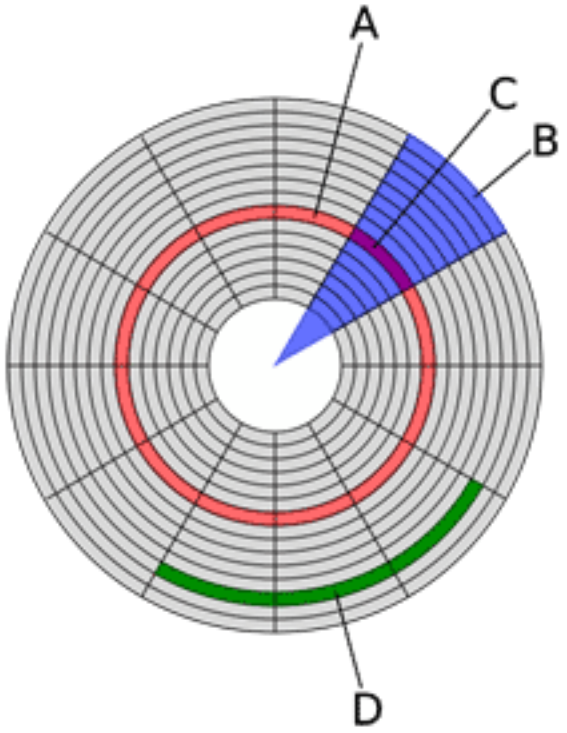
Datastructuren voor schijfblok opslag van gesorteerde data



Dr. D.P. Huijsmans
College 6 9 okt 2013
Universiteit Leiden
LIACS

Image:reviews.cnet.com

Opslag in blokken in secundair geheugen

geheugenhierarchie

Typische grootte (in KB):		Toegangstijd in nsec:
1 (KB)	Registers	1 (nsec)
4.000	L1 Cache	2
40.000	L2 Cache	4
4.000.000	Main Memory	10
400.000.000	Harde schijf	1.000.000 (msec)
4.000.000	Magneetband	10.000.000.000

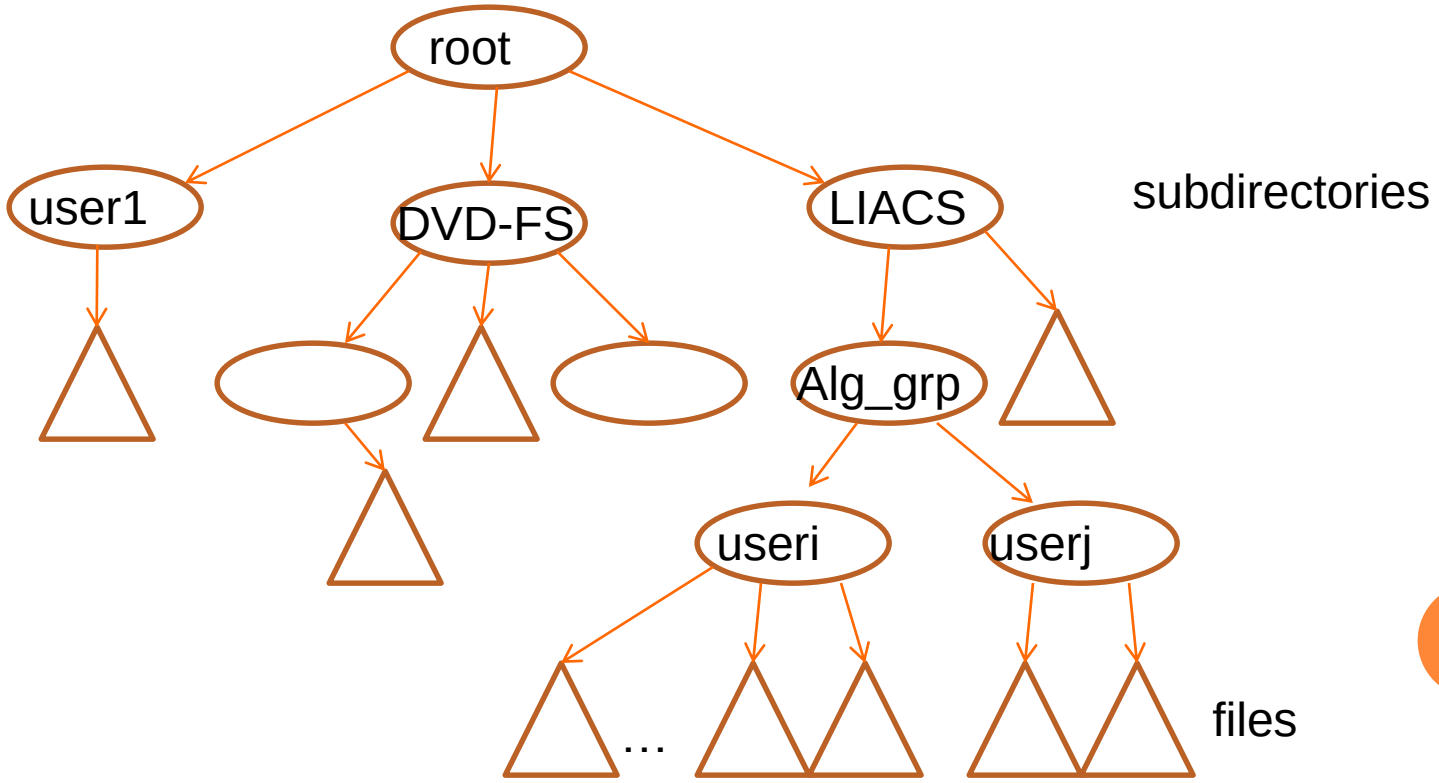
t/m main memory RA adresseerbare bytes; schijf RA per blok

Permanente opslag en toegang

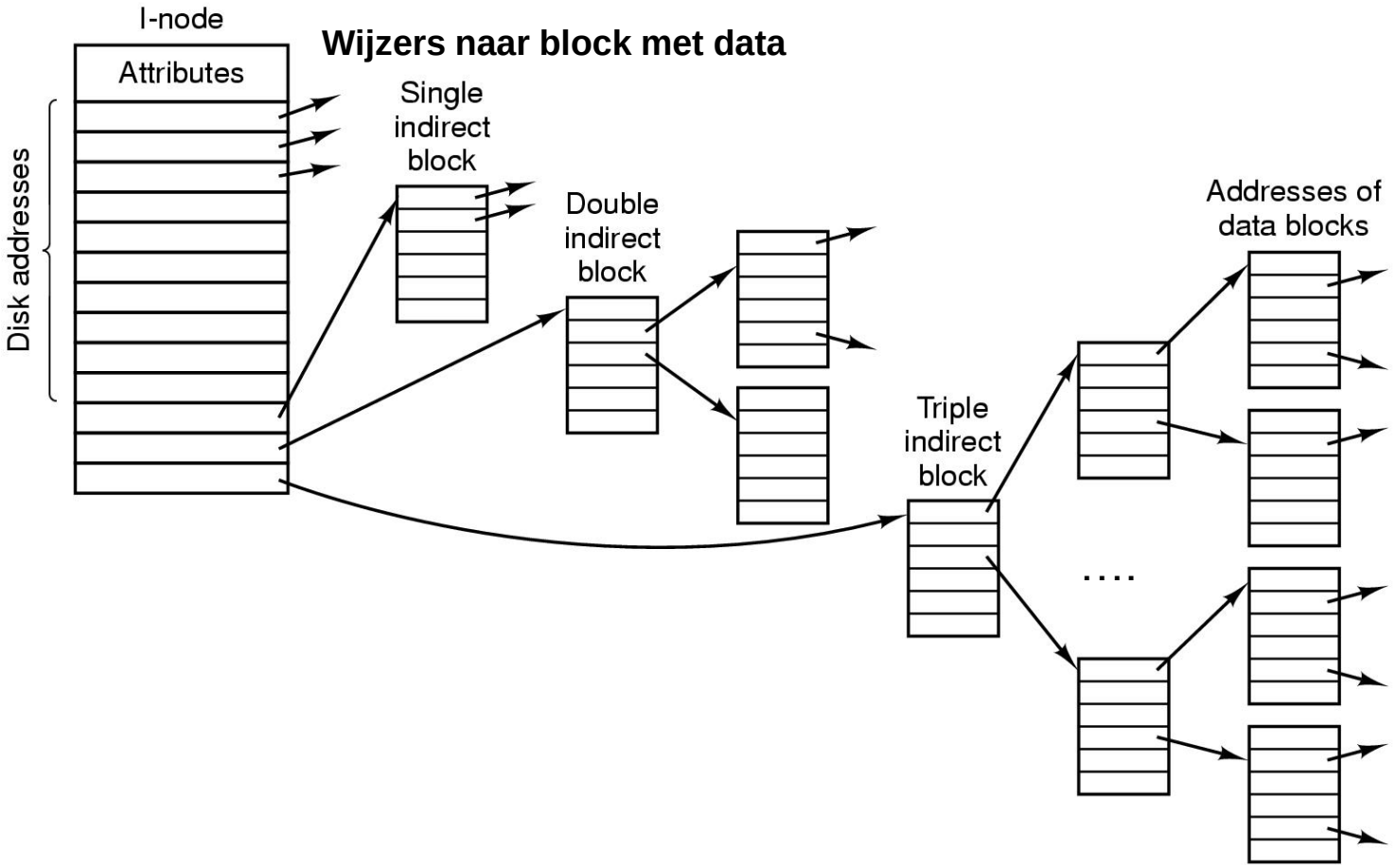
- Eenheid van opslag en hoe toegankelijk (RA of Serieel):
- Registers: 32/64 bit -> 4/8 bytes RA
- Caches: regels met 8-512 bytes RA
- MM: byte RA (1/2/4/8 bytes RA)
- HD: block met 512 bytes RA (byte in block serieel)
- MT: block met 512 bytes Serieel
- Doel1: hoe zorgen we er voor dat de data (bytes) die zo direct nodig zijn, al zo hoog mogelijk in deze geheugenhierarchie zitten?
- Doel2: hoe zorgen we ervoor dat gewijzigde data (bytes) uiteindelijk in het file systeem terechtkomen?

Data opslag en hierarchische naamgeving

- Data georganiseerd in files (bep. interne organisatie)
- Files (benoemd) vormen bladeren in
- (sub)directory structuur (boom structuur)



ADT_i-node
Unix i-node File organisatie



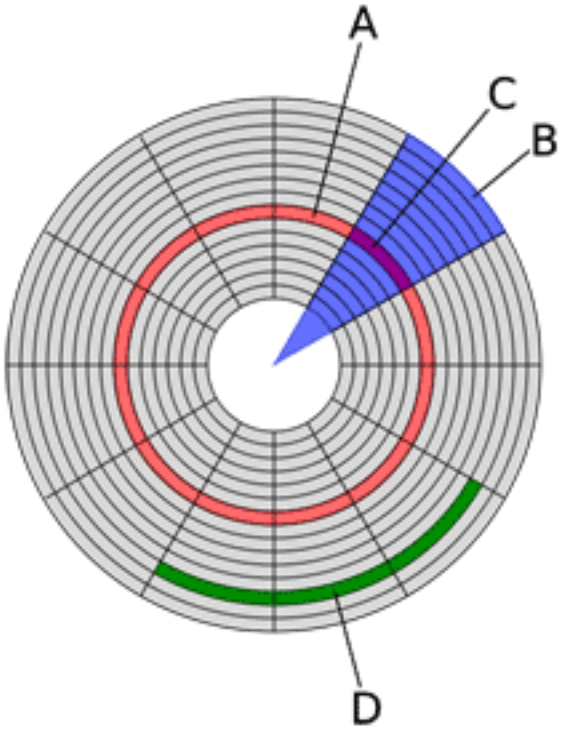
ADT_FileSystem
File operaties

- Creëer een file binnen een File Systeem
- Delete een file uit het FS
- Open een file voor lezen
- Open een file voor schrijven
- Open een file voor lezen en schrijven
- Wijzig een file

- ADT_OS als verzameling ADT's: ADT_FS, ADT_JS…,ADT_NC

- En nog veel meer in college Operating Systemen

Kosten van blok benadering



Om bij de juiste track (A) te komen moet lees/schrijfkop verplaatst worden
Om bij begin juiste sector (C) te komen moet schijf tot onder leeskop roteren
Om stel opeenvolgende blokken (D) te schrijven of uit te lezen is overzendtijd nodig

Dure toegang (msecs) tot blok compenseren door zo veel mogelijk ~gelijktijdig benodigde data gezamenlijk in 1 blok te zetten (Kbytes)

Image:reviews.cnet.com

Evolutie Data Access

- Jaren ‘70 (20e eeuw): serieel,gesorteerd opslag op magneetband voor periodieke (week/maand/jaar) updates/producties
- Jaren ’80: serieel, gesorteerde werkkopie op schijf
- Jaren ‘90: RA recordgewijs op schijf, tape backup; naar dagelijkse updates
- Huidig: RA in geheugen, backup naar schijf; naar momentane updates

Wat is er anders bij blok opslag?

- In computer geheugen (main memory) is elk element direct (RA) toegankelijk
- Op schijf is alles in 1 blok RA toegankelijk
- In een blok passen veel meer data elementen
- Overgang op opslag m.b.v. intervallen of reeksen:
- Interval arithmetiek

- Afwegingen/randvoorwaarden:
- Efficient blokgebruik -> liefst vol blok in blad
- Efficient toevoegen -> reserve plaatsen in blok
- Efficient splitsen -> tweedeling van 1 blok
- Efficient samennemen-> 2 halflege naar 1 vol blok

Interval Arithmetiek

- Interval heeft een linker en een rechtergrens
- Grens hoort wel of niet tot interval (gesloten/open)
- Rol +-∞
- Notaties:
- Elementen <,<=,=,=>,> of),],[],[,(
- Intervalnotatie:
- <, <=, => <=, > <=, =><, ><, =>, > of
- ..), ..], [..], (..], [..), (..), [.., (..
- Open/gesloten interval afhankelijk inhoud knoop

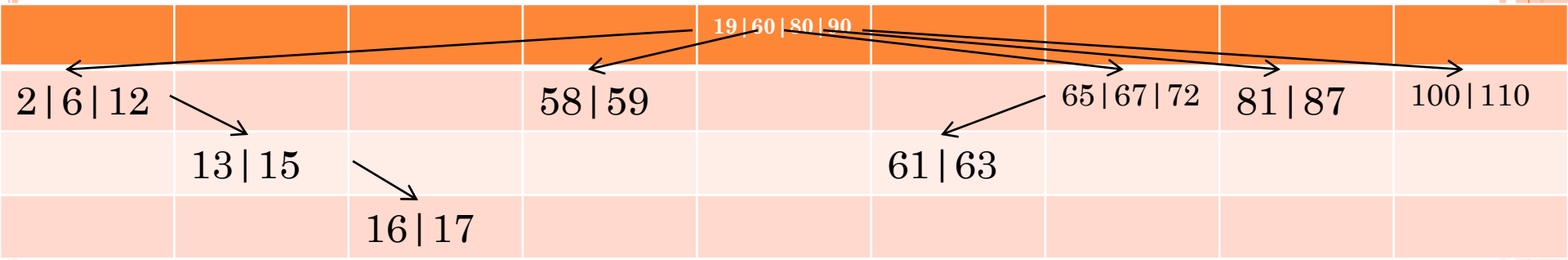
Multi Way Tree
een boom waar meer per knoop in kan

- Elke Knoop heeft m kinderen en m-1 sleutels
- De m-1 Sleutels in een knoop zijn oplopend
- Sleutels in eerste i kinderen zijn < i-de sleutel
- Sleutels in laatste m-i kinderen zijn > i-de sleutel

- Met deze definitie hoeft boom niet compleet of gebalanceerd te zijn
- Elke gevulde BST is dan een multiway-2 tree

Een Multiway 5-tree voorbeeld

INVOEGVOLGORDE:
19,60,80,90,2,6,16,17,13,15,12,58,59,61,63,67,72,65,81,87,100,110



Gesorteerd uit Depth-First LOR in-Order doorloop kost 16 x volgen pointer:
2 | 6 | 12 | 13 | 15 | 16 | 17 | 19 | 58 | 59 | 60 | 61 | 63 | 65 | 67 | 72 | 80 | 81 | 87 | 90 | 100 | 110

Nadeel: opslag in 9 blokken, slecht gebalanceerd, net 60% gevuld
Data op 4 niveaus

Gebalanceerde Multiway Trees
B-Trees

- Bayer en McCreight bij Boeing 1972

- Een multi-way B-boom heeft maximaal m pointers naar kinderen (subtrees) en maximaal m-1 key waarden per knoop
- De wortel (root) heeft minstens 2 subtrees tenzij het een blad is.
- Elk niet wortel knoop en blad is minstens halfvol (extra!)
- Alle bladeren zitten op hetzelfde niveau (extra!)

Binaire zoek boom

- Een binaire boom is een speciaal geval van een multi-way tree, namelijk de minimum vorm voor m=2
- Een binaire boomknoop heeft maximaal 2 pointers naar kinderen en bevat 1 key waarde

- Een complete gebalanceerde binaire boom is een speciaal geval van een B-2 Tree (wanneer?)

Een 5-way B-tree voorbeeld

		15 | 59 | 67 | 90		
2 | 6 | 12 | 13	16 | 17 | 19 | 58	60 | 61 | 63 | 65	72 | 80 | 81 | 87	100 | 110

Zelfde inhoud als in eerder Multiway 5-Tree
Voordeel: nu maar 6 blokken, 92% gevuld, bladeren op gelijk niveau
Data op 2 niveaus

Gesorteerd bij depth-first in-Order doorloop kost 10 x volgen pointer:

2 | 6 | 12 | 13 | 15 | 16 | 17 | 19 | 58 | 59 | 60 | 61 | 63 | 65 | 67 | 72 | 80 | 81 | 87 | 90 | 100 | 110

Nut B-Trees

- Nut valt of staat met opslag keys en/of data bij elkaar in 1 blok en of dat ook vaak direct na elkaar gebruikt wordt.
- Wanneer wel handig:
  - Bij periodieke bijwerking database:
    - Maandelijkse loonbetaling
    - Dagelijkse rente bijschrijving
  - Als wijziging afhankelijk omgevingsdata:
    - Online boeking vliegtuigstoel
- Wanneer niet handig:
  - Verwerking random pin transactie verkeer
  - Online afhandeling klacht

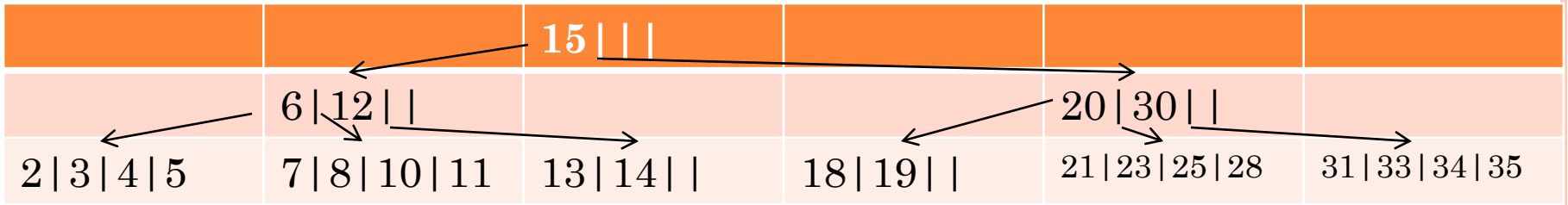
Bijwerken B-bomen

- Eerste opzet en gebruik B-bomen efficient
- Bijwerken: lastige insert/delete situaties
- Eis: minstens half vol en bladeren op 1 niveau
- Insert op volle knoop -> 2 minstens halfvolle knopen met mogelijk extra niveau
- Delete in halfvolle knoop -> samennemen met andere halfvolle knoop die vaak ontbreekt -> mogelijk ingrijpende reorganisatie

B-tree
lastige insert
(bladeren op 1 niveau houden)

		6 | 12 | 20 | 30		
2 | 3 | 4 | 5	7 | 8 | 10 | 11	14 | 15 | 18 | 19	21 | 23 | 25 | 28	31 | 33 | 34 | 35

Insert 13 in volle B-5 boom
Root minstens 2
pointers naar knopen
die elk minstens 2 keys bevatten



Zie Drozdek pag 308

B-Tree deletes

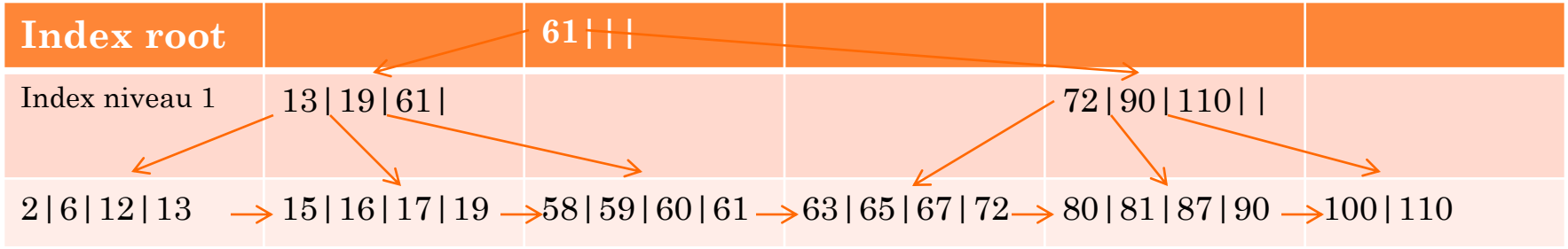
- Delete key kan lokaal verwerkt worden als:
- Knoop een blad is en na weglating blad nog minstens halfvol
- Als blad na weglating <halfvol, eerst proberen met herschikken tussen inhoud Ouder knoop en diens kinderen de zaak op te lossen
- Als hele subtree waar delete in plaatsvindt te leeg voor lokale oplossing, dan hele boom plus niveaus aanpassen

- Zie Drozdek 306-312 voor voorbeelden

B-Boom varianten

- B*-bomen: alle knopen minstens voor 2/3 vol zodat er gemiddeld nog minder knopen (elke knoop is een blok op schijf) nodig zijn
- B^+-bomen: alle key waarden zitten in de bladeren, daarboven alleen indexkeys (eventueel kopie van de key in het blad), tevens zijn de bladeren inOrder gelinkt, zodat alleen de bladeren sequentieel doorlopen hoeven te worden voor een complete gesorteerde lijst

Een 5-way B+-tree voorbeeld



Zelfde inhoud als in eerdere B 5-Tree
Pointergrenzen nu <=,><=,> ofwel ..],(..],(..
Voordeel: nu 6 data blokken, 92% gevuld, alle op 1 (blad) niveau,
Blokken mogelijk sequentieel in opeenvolgende sectoren binnen track of aangrenzende tracks
Index niveaus kunnen naar MM bij openen file of
met extra pointer van blad naar blad snelle gesorteerde doorloop met 8 x pointer volgen:

2 | 6 | 12 | 13 | 15 | 16 | 17 | 19 | 58 | 59 | 60 | 61 | 63 | 65 | 67 | 72 | 80 | 81 | 87 | 90 | 100 | 110

Overzicht multiway 5 tree voorbeeld

	M-Tree	B-Tree	B^+-Tree
Blokken	9	6	3 index
			6 data
Vulling	60%	92%	60%
			92% data
Doorloop	16x	10x	8x
Data Niv.	4	2	1

- B^+-Tree typisch voorbeeld van meer gewicht geven aan gesorteerde doorloop snelheid dan aan blokgeheugen gebruik

Meer B-Boom varianten

- Prefix B^+-bomen: index key in niet-blad is een zo kort mogelijk prefix van uiteindelijke key in blad
- Door de beperking tot een sleutel die het unieke begin van een sleutel eronder weergeeft, kan er sneller gematched worden bij het doorlopen van de prefix boom.
- Bit-trees: het idee van prefixen opgebouwd uit te matchen karakters kan nog verder versneld worden door niet byte voor byte maar bit voor bit de start (prefix) van keys te vergelijken.

PAUZE

T-Tree
Index tree for Main Memory data

- Tegenwoordig wordt werk op grotere databases vanwege snelheidswinst vaak uitgevoerd op een in MM gehaalde kopie van de database
- Een snelle toegang tot de data hoeft dan de data zelf niet meer te bevatten.
- De T-Tree is een soort binaire zoekboom waarin alleen het index gedeelte van het zoekpad opgeslagen staat: de indexpointers verwijzen uiteindelijk naar de al in MM staande data elementen

Voorspelbaarheid van blokbenadering

- Liefst alleen opslag in bladeren
- Liefst alle bladeren op zelfde niveau in boom
- Niveaus boven blad niveau (index niveaus) naar geheugen

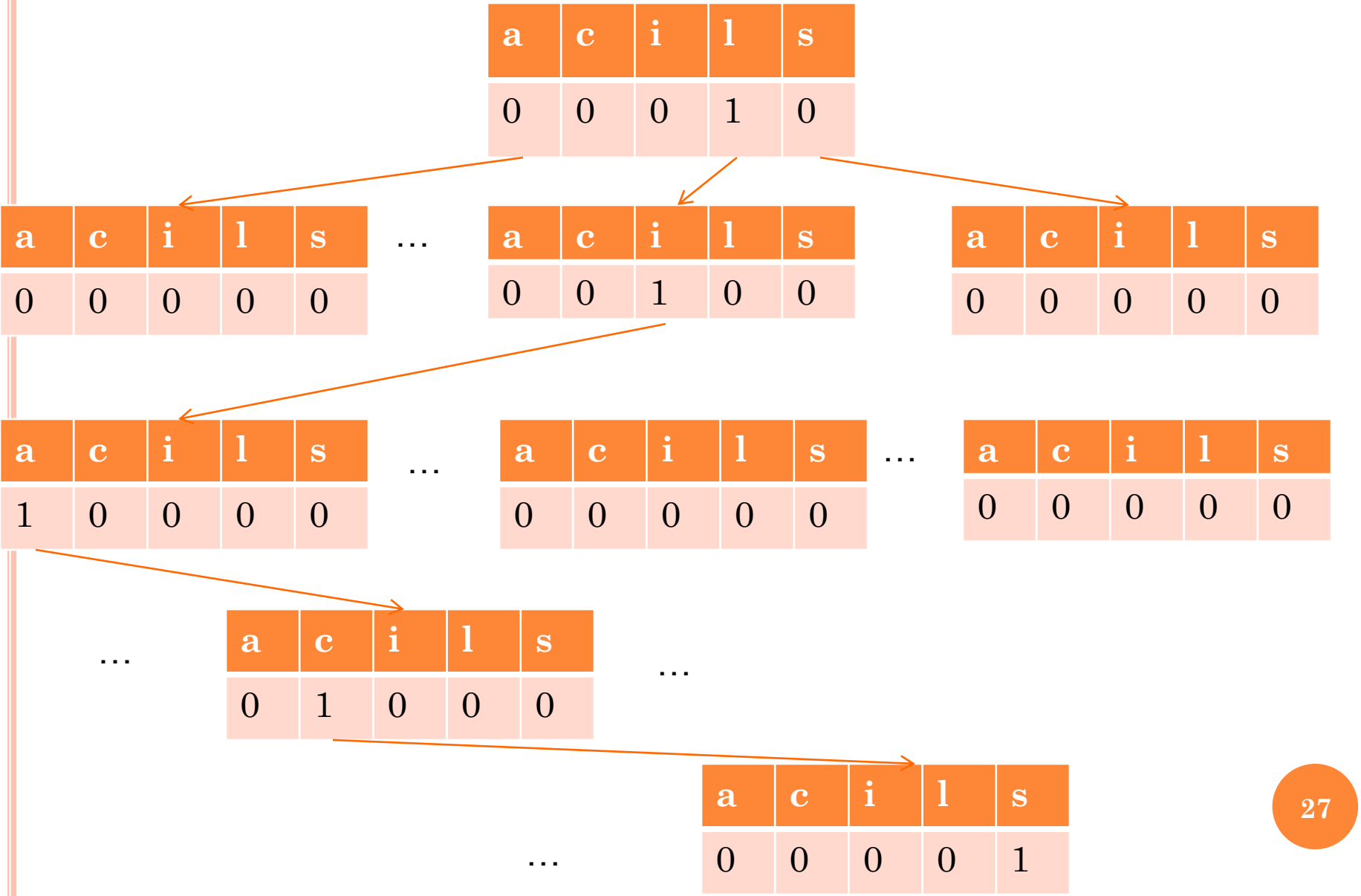
- Ontwikkelde multiway trees:
- B-tree en varianten B^* en B^+
- T-tree

- Complexiteit: al deze boomstructuren O(logN) voor Opzoeken, Insert, Delete

SPELLINGCHECKER MET TRIES

- In plaats van strings (keywaardes) te vergelijken kunnen we ook gebruik maken van het feit dat elke string opgebouwd is uit een beperkt aantal karakters b.v. [a..z](alfabet)
- Als elke string maximaal k karakters lang is, dan bouwen we een multiway tree op die op elk van de k niveaus bestaat uit alfabetbitvectoren waarvan de link bij het karakter dat in de string op plaats l voorkomt wordt gezet (bij het vullen met toegestane woorden) of genomen (bij het kontroleren of een woord toegestaan is)
- Verschillende stringlengtes kunnen worden gebruikt als de string en de alfabet bitvector met een afsluitteken wordt uitgebreid.

SPELLINGCHECKER TRIE-2



Spellingchecker Trie-3

- Door het afsluitteken in het alfabet nooit een kindpointer te geven kan m.b.v. louter deze bitvectoren een verzameling toegestane woorden gezet en/of gekontroleerd worden op voorkomen.
- Omdat het aantal bitvectoren exponentieel toeneemt, kan als besparing bij het opbouwen pas een bitvector worden toegevoegd als er bij vullling ook werkelijk gebruik van gemaakt wordt; bij kontrole eindigt het pad dan op een nil pointer (net als bij het afsluitkarakter)

DATASTRUCTUREN VOOR RUIMTELIJKE GEGEVENS

- Er komen steeds meer toepassingen waarbij data elementen worden gebruikt die inherent meerdimensionale kenmerken vertonen en geindexeerd worden met makkelijk te sorteren kenmerken zoals lengte-,breedtegraad, tijdstip:
- 2D: kaartmateriaal, plattegrond
- 3D: gebouwontwerp, productontwerp, mijnbouw
- Ruimtelijke data nodig o.a. voor werken met:
- Plaatsing, orientatie, schaling
- Selecties binnen een gebied
- Afstanden tussen objecten
- Objecten binnen een straal rond

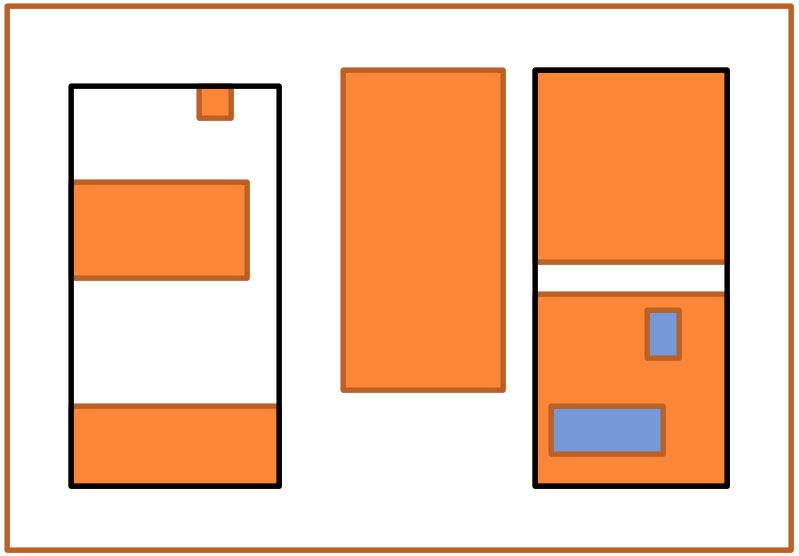
R(echthoek)-Trees

- R-bomen zijn opgezet om te kunnen werken met:
- posities van rechthoeken
- nestingen van rechthoeken
- Overlap van rechthoeken
- Omsluitende rechthoek van rechthoeken

- Voor een architect is een geneste opdeling van een ruimte met een beperkt aantal ruimtes daarbinnen zo veelvoorkomend dat een multi-way opzet (hierarchisch) meer voor de hand ligt dan een binaire opzet (gesorteerd maar 1 dim)

R-TREE
INFORMATIE IN HOGERE NIVEAUS

Hierarchie middels MER's (minimal Enclosing Rectangles)
Geneste niveaus

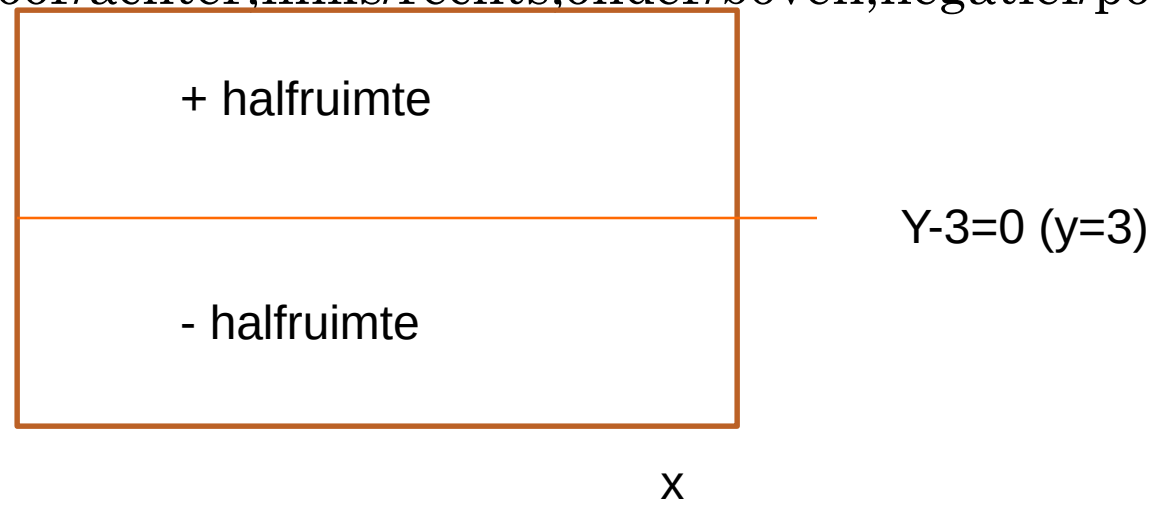


Manieren om 2D/3D ruimtes recursief op te delen

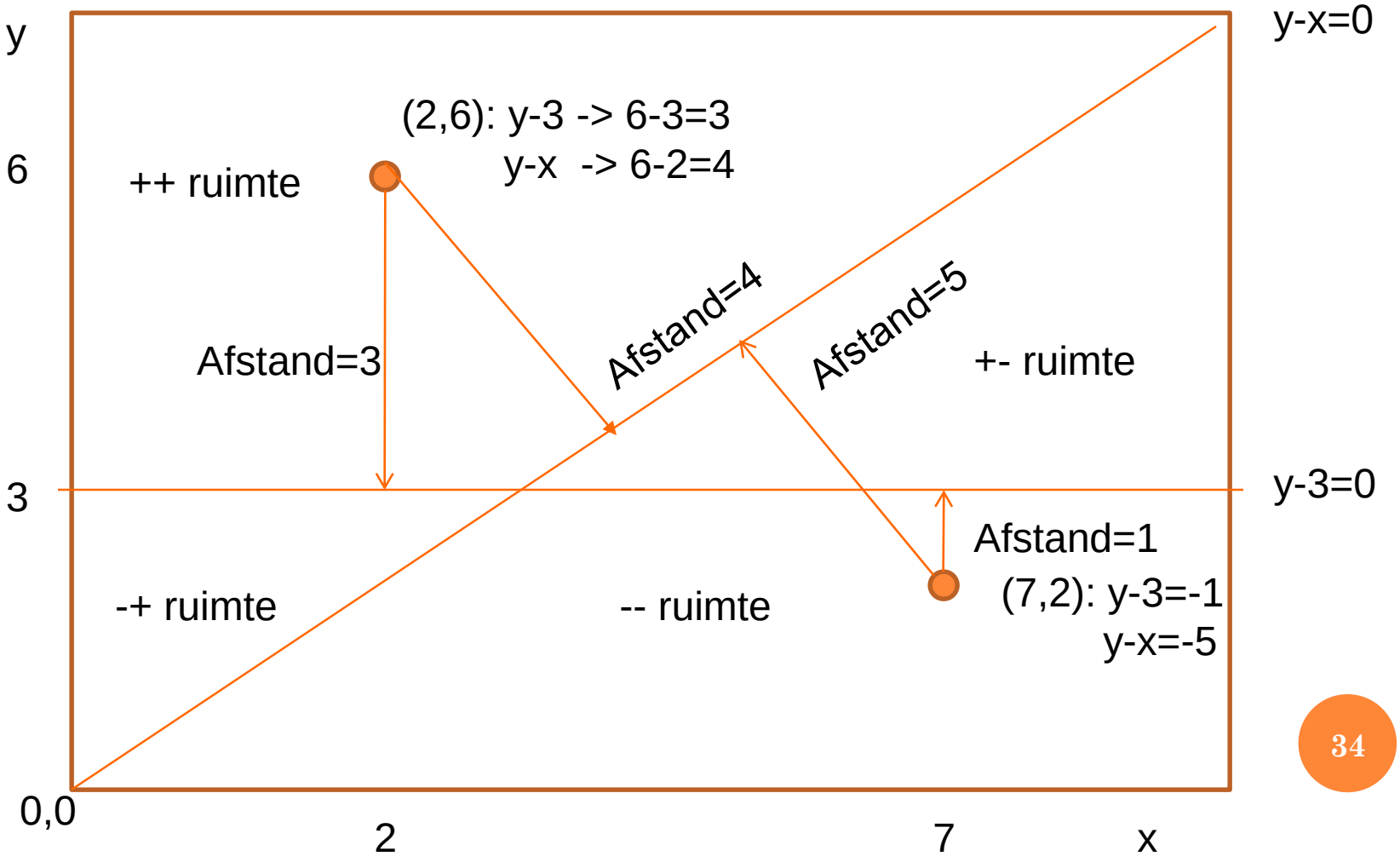
- Voor het snel genereren van afbeeldingen van 3D scenes opgebouwd uit vlakken zijn meerdere ruimtelijke datastructuren bedacht:
- Ze onderscheiden zich in het aantal vlakken waarmee niveaus worden onderverdeeld en de manier waarop een onderverdeling van de ruimte tot stand komt
- BSP Binary Space Partitioning Tree: elk vlak(3D) of elk lijnstuk(2D) dat voorkomt in de definitie van een scene deelt de ruimte recursief in 2-en
- Quad Tree: de 2D ruimte wordt volgens een vast grid in 2 richtingen gehalveerd -> opdeling in quadranten, dit wordt recursief herhaald tot quadrant geheel leeg of geheel gevuld
- Oct Tree: de 3D ruimte wordt volgens een vast grid in 3 richtingen gehalveerd > opdeling in octanten, dit wordt recursief herhaald tot octant geheel leeg of geheel gevuld.

Recursieve Opdeling in 2D met lijnen

- Lijn gerepresenteerd als f(x,y)=0
- Invulling van bepaald punt x1,y1 in deze functie:
  - f(x1,y1)=0 : punt ligt op de lijn
  - f(x1,y1)=d : geeft afstand tot die lijn
  - Teken van d geeft aan aan welke kant van die lijn het punt ligt
- Elke lijn deelt de 2D ruimte in 2 halfruimtes (voor/achter;links/rechts;onder/boven;negatief/positief)



Opdeling in halfruimtes met meerdere lijnen in 2D



Recursieve opdeling in 3D met vlakken voor Hidden Surface Algoritme

- In 3D is het opdeel element een vlak f(x,y,z)=0
- Bij een vlak hoort een oppervlaktenormaal:
- Buitenoppervlak (met textuur) aan kant normaal
- f of normaal kan zo gekozen worden dat normaal aan + kant
- Binnenoppervlak aan – kant

- BSP (Binary Space Partitioning Tree):
- Recursieve onderverdeling d.m.v. georienteerde vlakken

- Plaats van (gezichts)punt in halfruimteopdeling bepaalt welke (buiten)vlakken vandaar uit zichtbaar zijn en de zichtbare vlakken kunnen op afstand gesorteerd worden en van veraf naar dichtbij volgorde afgebeeld (+texturen)

Modellering oppervlak of volume

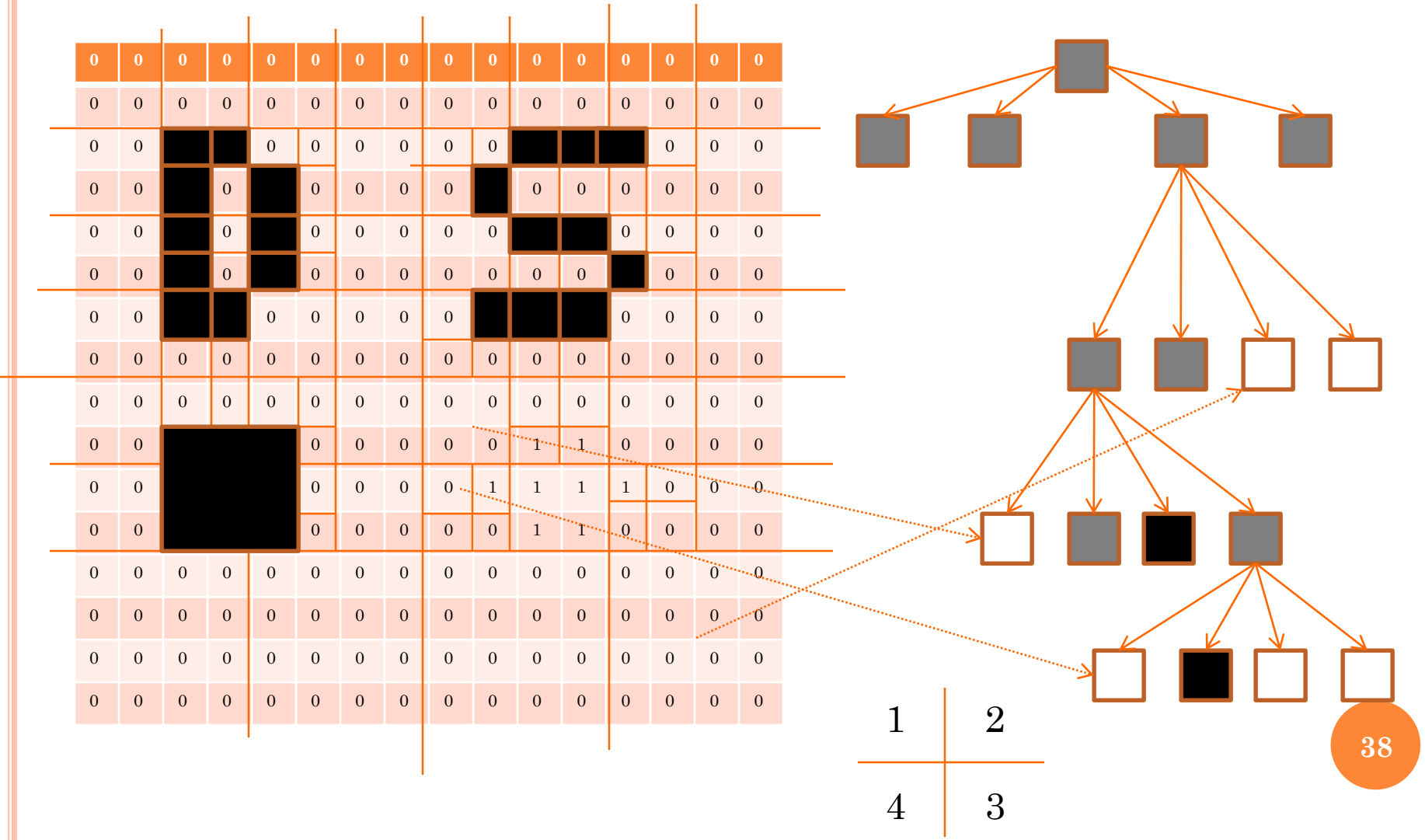
- Voor grafische weergave is modellering van het oppervlak voldoende; hierbij past de BSP het best
- Voor modellering van objecten wordt de inhoud(2D) of het volume(3D) beschreven; hierbij past een Quad- of Oct-Tree het beste

- Er zijn algoritmes waarmee een volume beschrijving kan worden omgezet in een oppervlakbeschrijving (b.v. Marching cubes algoritme) zodat voor weergave de BSP ingezet kan worden voor opslag van de buitenvlakjes

Inhoudsbenadering met Pixels/voxels

- Zoals een pixel een kleinste onderdeel van een 2D opname is, is het voxel (volume element) het kleinste deel van een inhoudsvorm.
- Per element is niet meer dan een bit/cel nodig:
- 0=cel valt buiten 2D/3D vorm
- 1=cel valt binnen 2D/3D vorm
- Uit de hoogste resolutielaag is volgens het vaste opdeelschema snel een Quad/Oct Tree op te bouwen waarin knopen op niveaus onder een geheel gevuld of geheel leeg quadrant/octant een blad vormen binnen de datastructuur

Voorbeeld quadtree



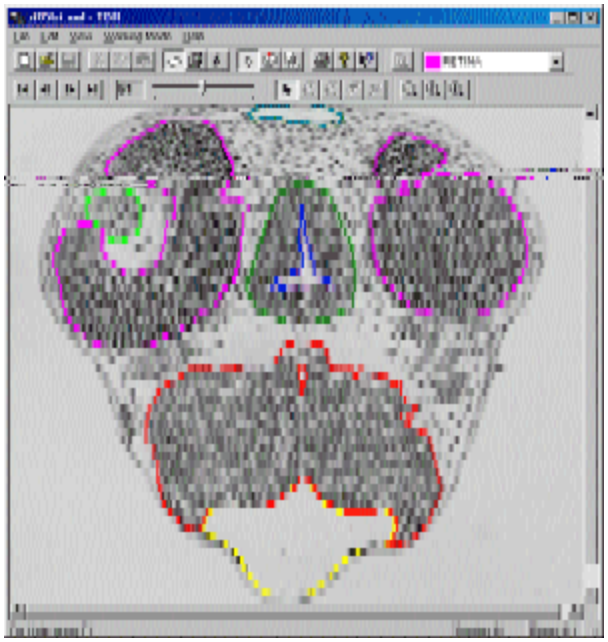
Quad Tree is geen B-Tree

- Quad Tree is van orde 4
  - Geen orde m multi-way want geen waarde in de knoop als knoop niet blad
  - Bladwaarde is 0 of 1
  - Blad kan op elk niveau zitten (ook root)

- Opbouw QT:
- Vanuit root recursief bezoek quadrant 1-4:
  - als alle elementen gelijk krijgt knoop die waarde,
  - Als ongelijk bezoek quadrant 1-4 van die knoop
- Vanuit elementen:
  - Genereer complete QT t/m diepste niveau
  - zet waarde elk element
  - Ga quadrant voor quadrant, niveau voor niveau -> root en zet parent waarde als alle kinderen gelijk en delete dan kinderen

3D versie van de BT: Oct Tree

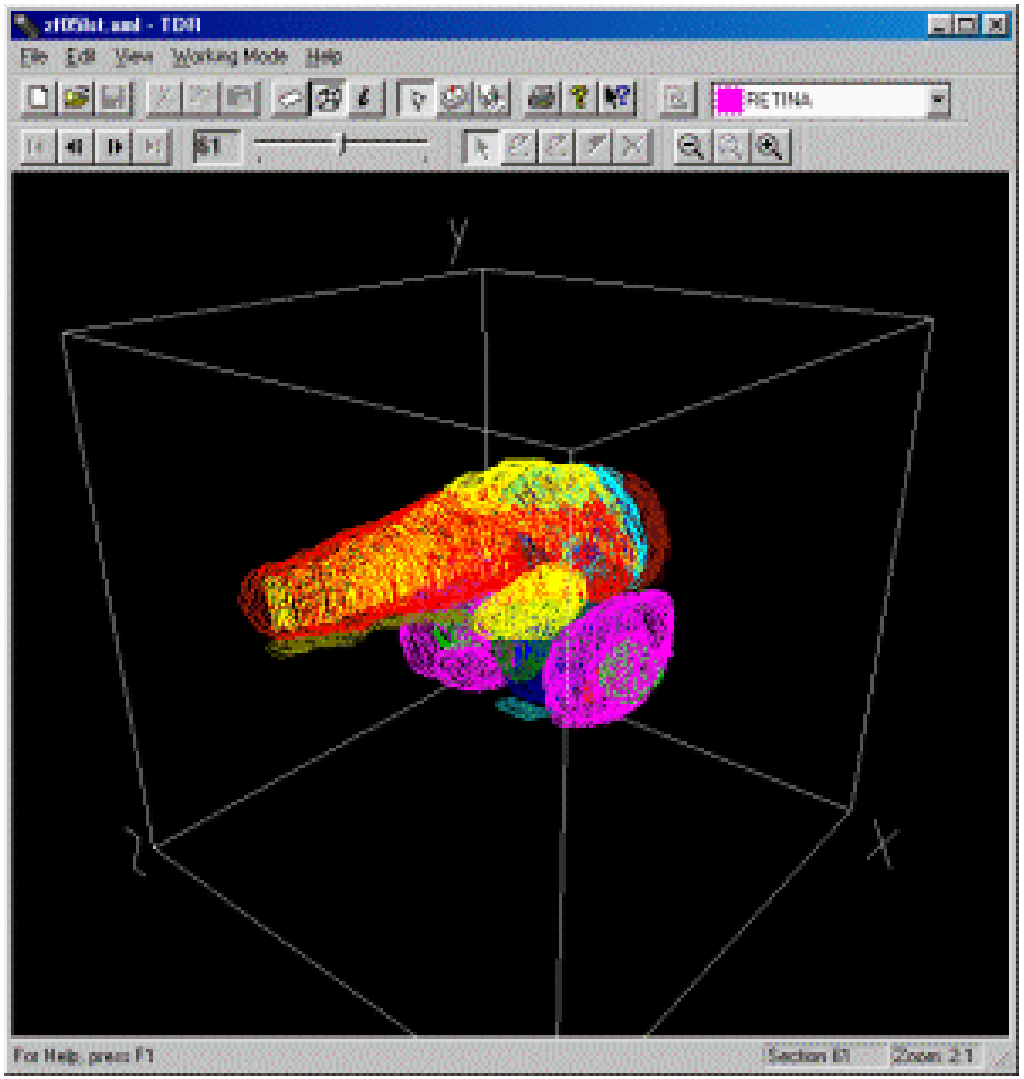
- Binaire recursieve opdeling in 8 octanten
- Elke bladcel bevat:
- Bit: wel/niet deel volumemodel
- Byte: dichtheid volume ter plaatse
- Weefselcode: deel van welke weefselstructuur



Voorbeeld 2D invoer plak voor gelaagd 3D model zebravis

Bron: F.J. Verbeek

Datastructuren voor opslag en grafische weergave oppervlak van 3D volume



Voor deze afbeelding zijn in elke 2D plak van het 3D model buitencontouren van het weefsel gevormd, waarna er tussen 2 opeenvolgende contouren van aangrenzende doorsnedes een driehoeksbelegging is gevormd; deze vlakjes zijn als draadraammodel gezien vanuit een bepaald gezichtspunt in de kleur van het weefseltype afgebeeld.

Bron: F.J. Verbeek

Drozdek

- H 7.1, 7.2, 7.4