AV-Space

Fast Counting with AV-Space for Efficient Rule Induction

TECHNISCHE UNIVERSITÄT DARMSTADT

Seminar aus maschinellem Lernen

Dr. Heiko Paulheim
Frederik Janssen

Outline

- Regellernen
- AV-Spaces
  - Annahmen
  - Tree & Group
  - Algorithmen
- Auswertung
- Fazit

Regellernen

Covering-Strategie • Flaschenhals • Datenstrukturen

Regellernen

***Covering-Strategie** - Flaschenhals - Datenstrukturen*

- Propositional Rule Learning
  - Gegeben: Menge von Trainingsdaten (Attribut-Wert Paare)
  - Gesucht: Regelmenge
- Separate-and-conquer Regellerner (Covering-Strategie)
  - Lerne eine Regel
  - Füge diese Regel der Regelmenge hinzu
  - Entferne alle Beispiele, die bereits abgedeckt sind
  - Lerne die nächste Regel
- Algorithmen
  - CN2, ELEM2, PRISM, …

Regellernen

*Covering-Strategie - **Flaschenhals** - Datenstrukturen*

- Separate-and-conquer Regellerner
  - Lerne eine Regel
  - Füge diese Regel der Regelmenge hinzu
  - Entferne alle Beispiele, die bereits abgedeckt sind
  - Lerne die nächste Regel
- Berechnungsaufwand
  - Auswahl eines Attribut-Wert Paares
    - Heuristik
    - Wahrscheinlichkeiten berechnen
  - Abdeckung einer (Teil-)Regel (counting query)
  - Abdeckung der noch nicht erschlossenen Beispiele (Rule tiling problem)

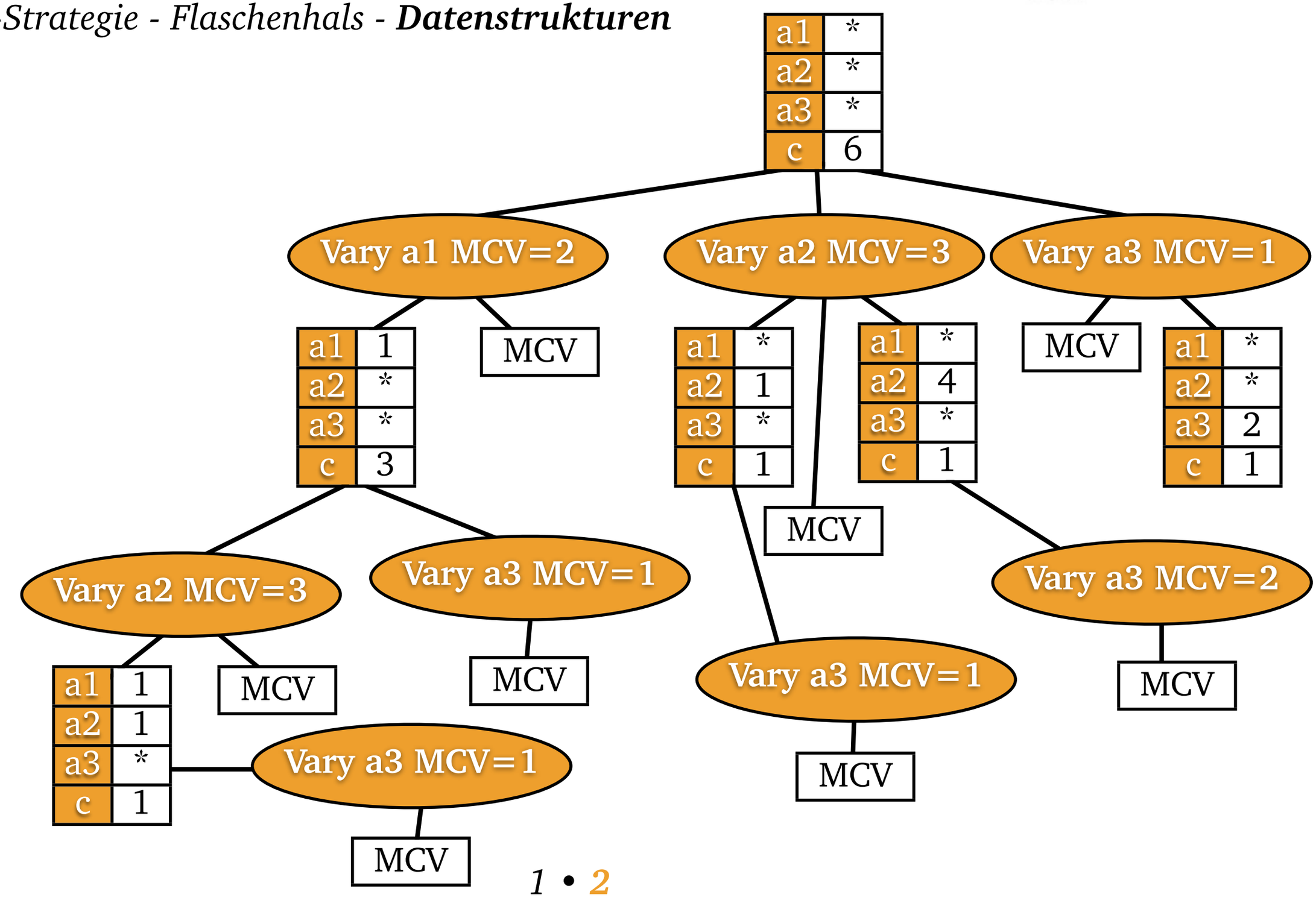
Regellernen

*Covering-Strategie - Flaschenhals - **Datenstrukturen***

- meistens wird ein dynamisches Array bentutzt
  - Ein Eintrag entspricht einem Trainingsbeispiel
  - Abfrage führt zu durchlaufen des Arrays
- AV-space
  - Kombiniert Baum mit Liste
  - Baum für Berechnung
  - Liste um Knoten im Baum direkt zu adressieren
- ADtree
  - Zwei Knotentypen: ADtree node, Vary node
  - Vary node: gibt Attribut an und MCV
  - ADtree node: gibt eine Maske für die anderen an und die Anzahl

Regellernen

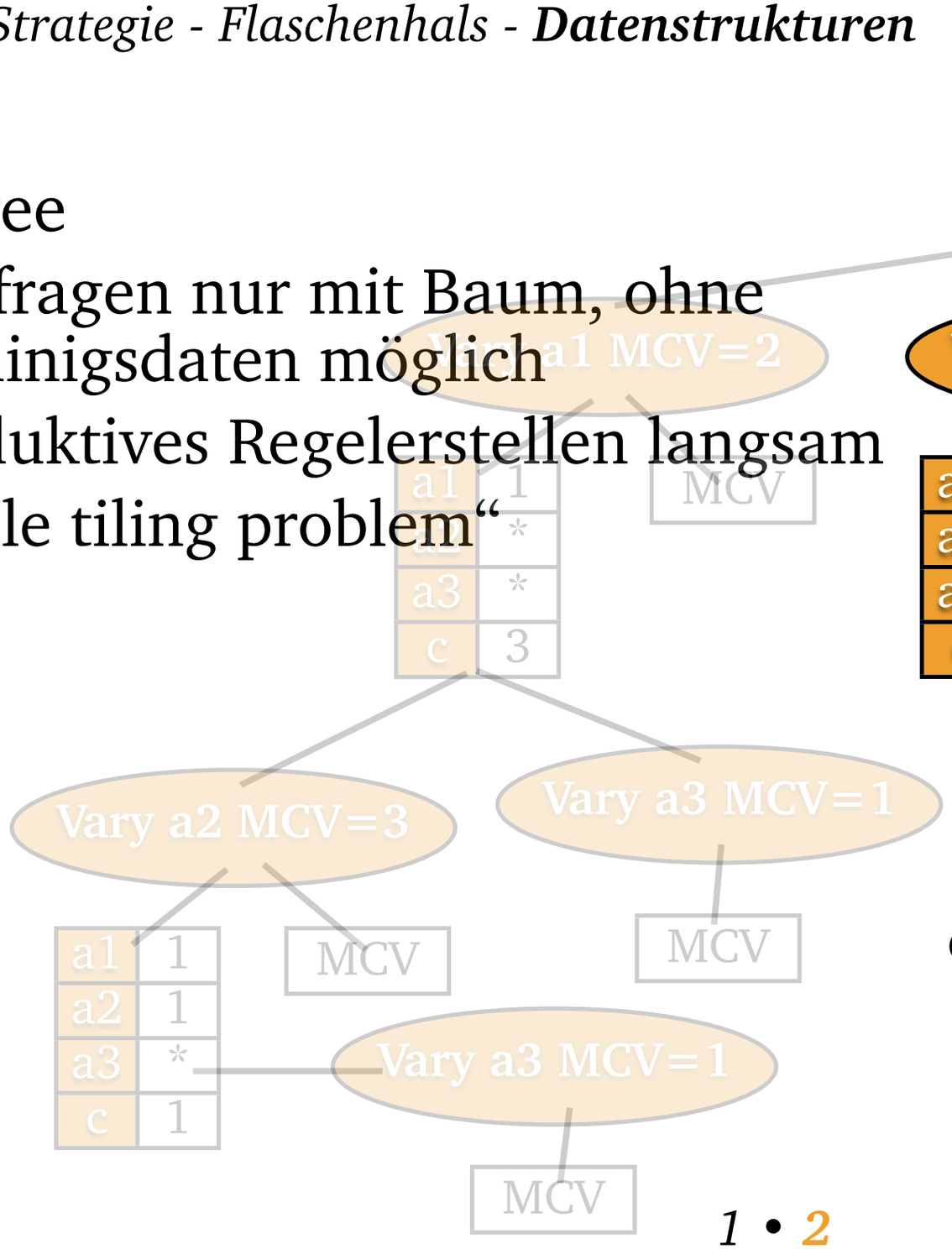
*Covering-Strategie - Flaschenhals - **Datenstrukturen***

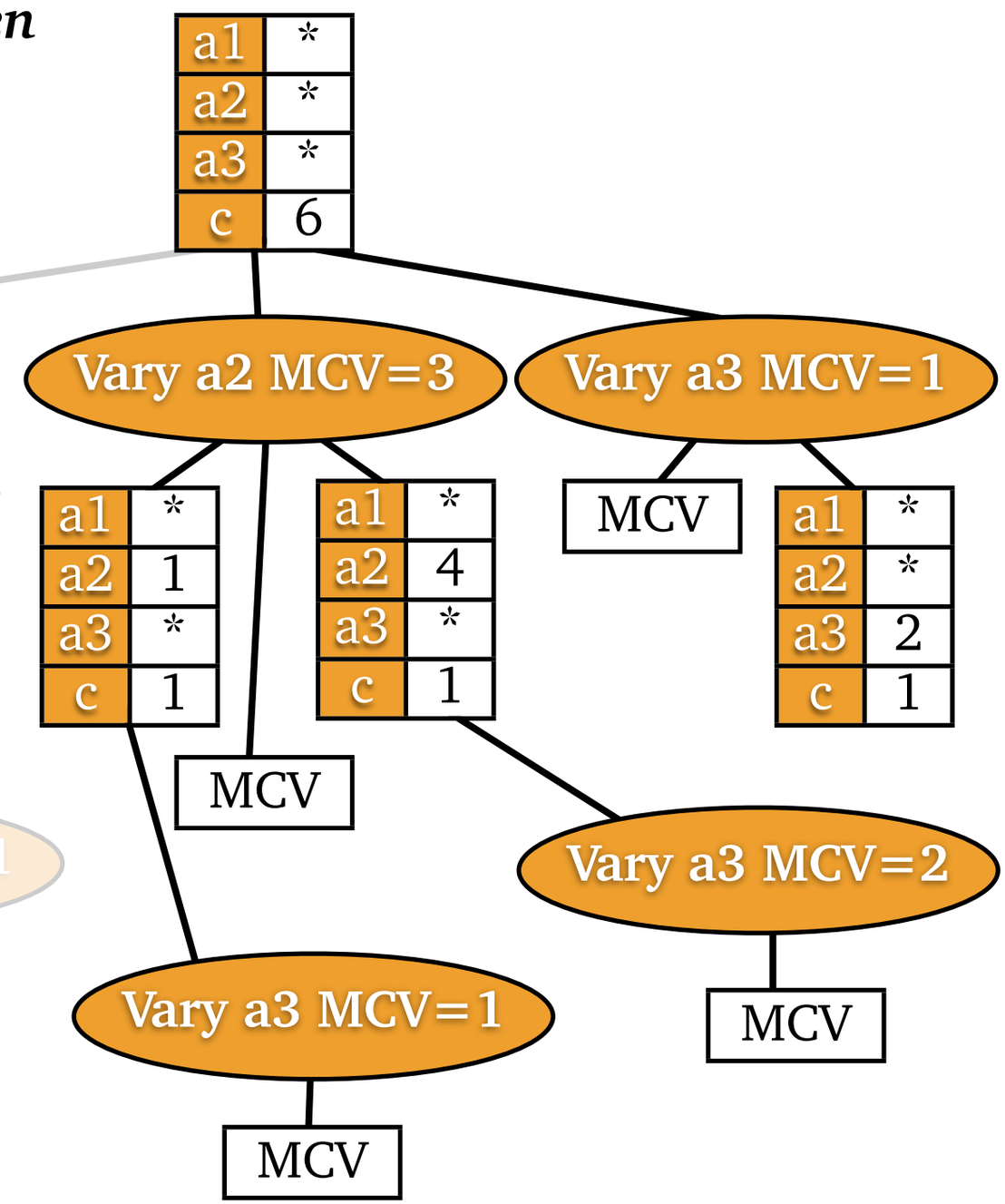


Regellernen

*Covering-Strategie - Flaschenhals - **Datenstrukturen***

- ADtree
  - Abfragen nur mit Baum, ohne Trainigsdaten möglich
  - Induktives Regelerstellen langsam
  - „rule tiling problem“





1 • 2

AV-Space

Annahmen - Überblick - Tree - Group - CS

AV-Space

Annahmen *- Überblick - Tree - Group - CS*

- Werte sind diskret verteilt
- Abfragen sind von folgender Art
  - DNF
  - $<a_i=v_{i1}$ or v_{i2} or …$>$ AND $<a_j = v_{j1}$ or v_{j2} or …$>$
  - Attribute sind sortiert

AV-Space

*Annahmen - **Überblick** - Tree - Group - CS*

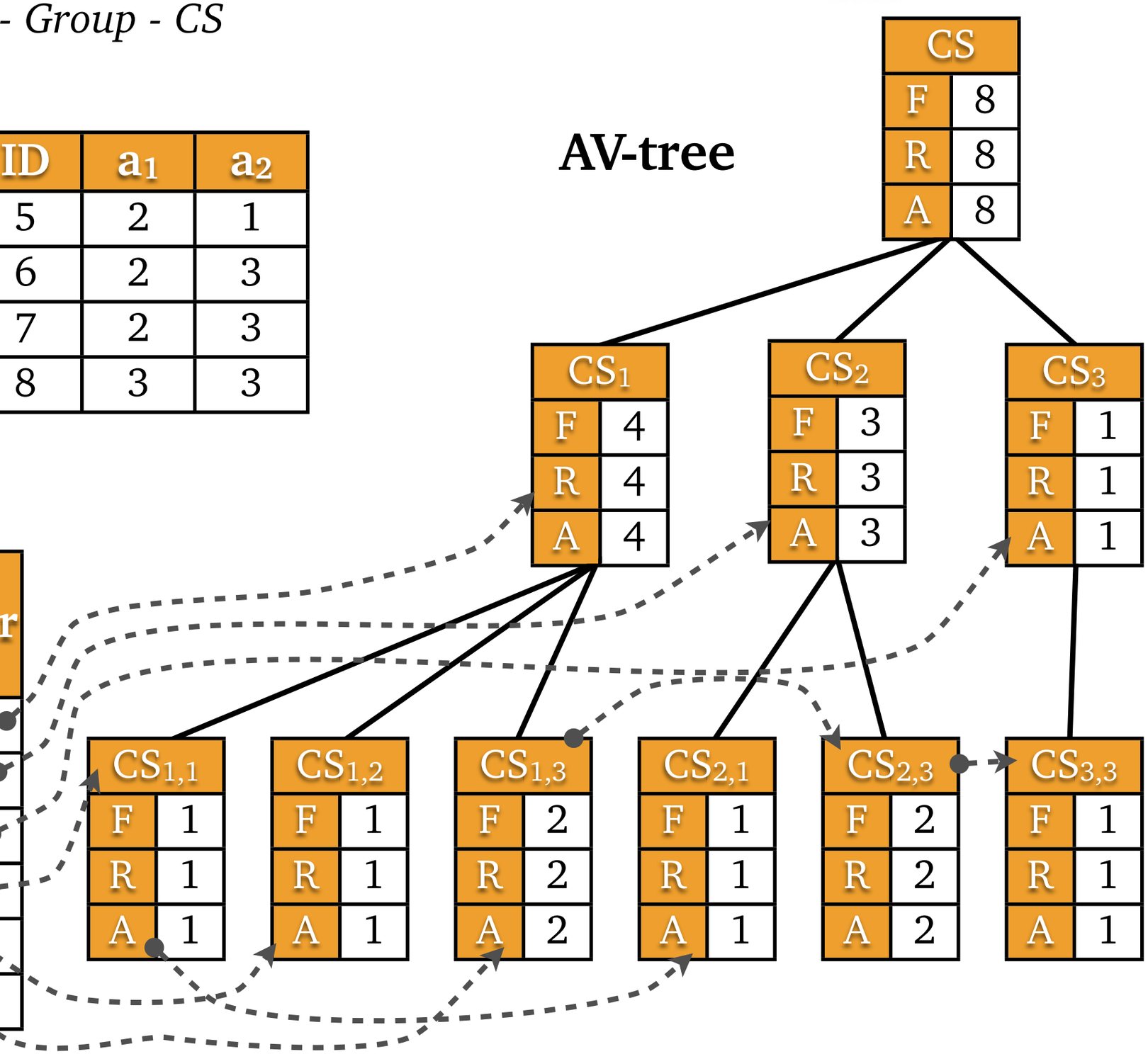
ID	a_1	a_2
1	1	1
2	1	2
3	1	3
4	1	3

ID	a_1	a_2
5	2	1
6	2	3
7	2	3
8	3	3

AVC-group

AVP	Counting Set			Pointer
	F	R	A	
$a_1=1$	4	4	4	
$a_1=2$	3	3	3	
$a_1=3$	1	1	1	
$a_2=1$	2	2	2	
$a_2=2$	1	1	1	
$a_2=3$	5	5	5	

AV-tree



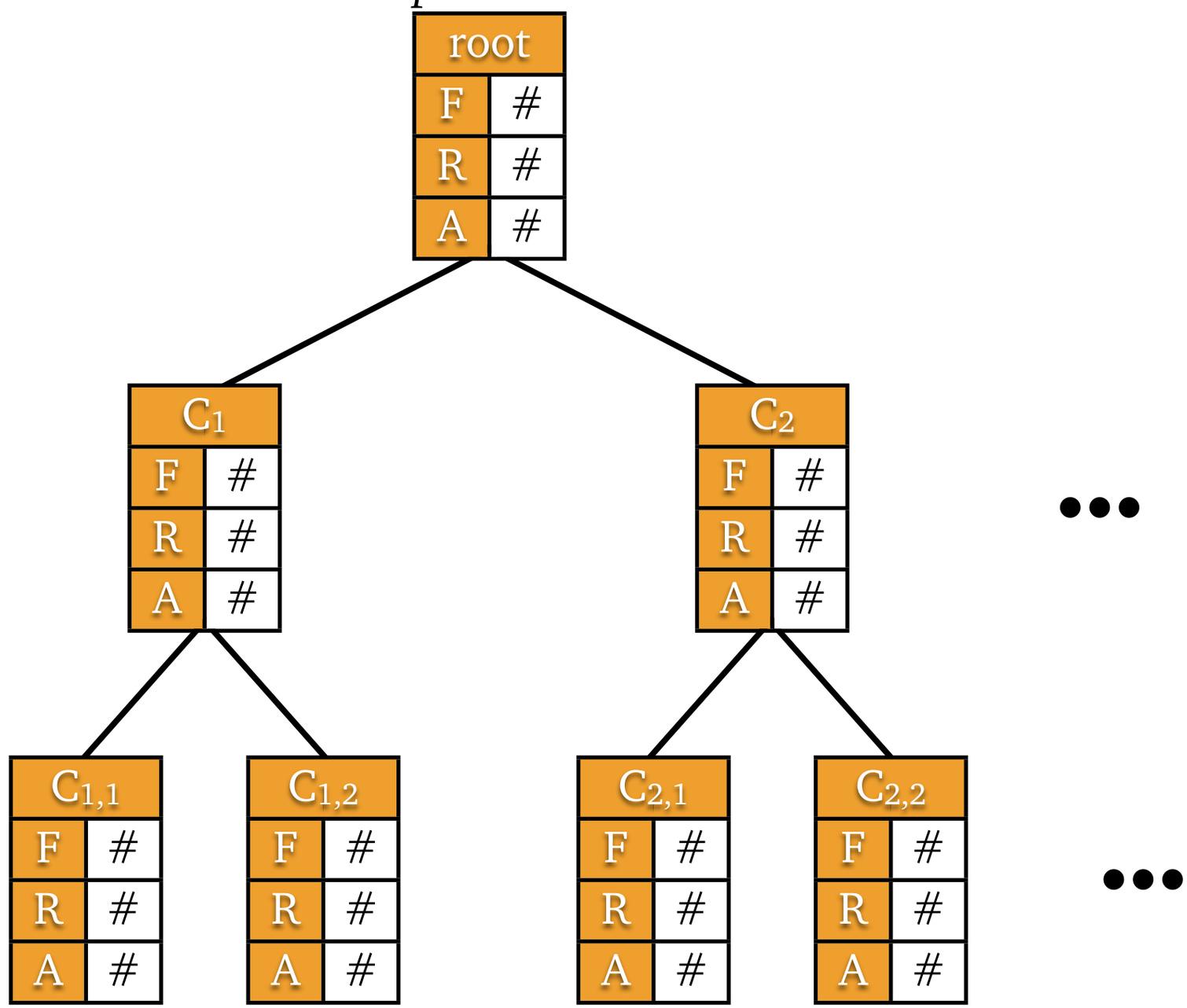
AV-Space

*Annahmen - Überblick - **Tree** - Group - CS*

- Trainingsdaten
  - $\{a_1, a_2, \ldots, a_M\}$ Attribute
  - $\{v_{i,j} \mid i=1\ldots M, j=1\ldots k_i\}$ Werte
  - k_i = Anzahl eindeutiger Werte zu a_i
  - a_M ist *class* Attribut
- AV-Tree
  - $M+1$ Ebenen (Wurzel = Ebene 0)
  - Ebene $i \sim$ Attribut a_i
  - Ebene $M+1 \sim$ *class* Attribut
  - Knoten auf Ebene $i \sim$ Konjunktion über i Attribut-Wert Paare

AV-Space

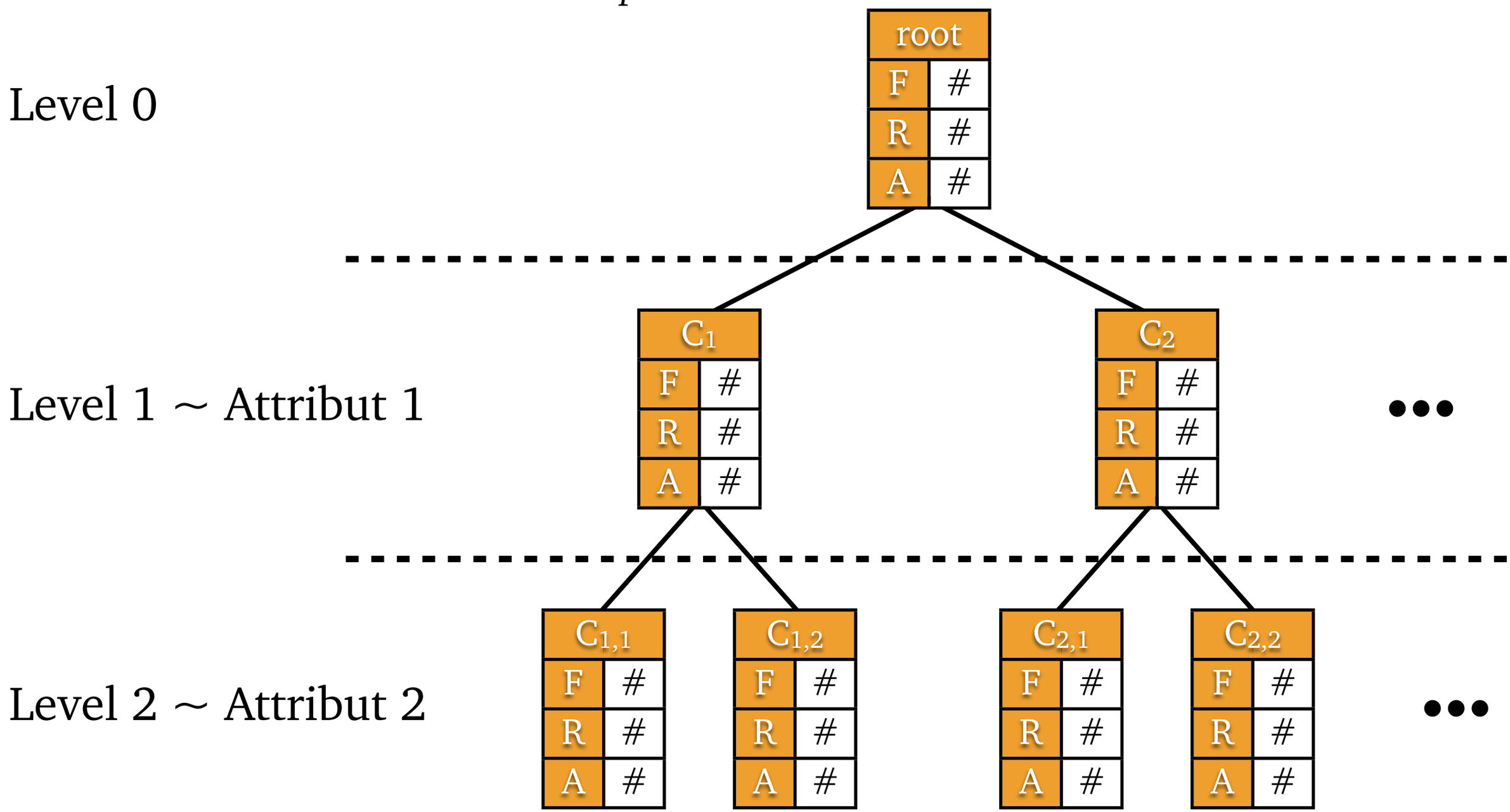
*Annahmen - Überblick - **Tree** - Group - CS*



1** • **2

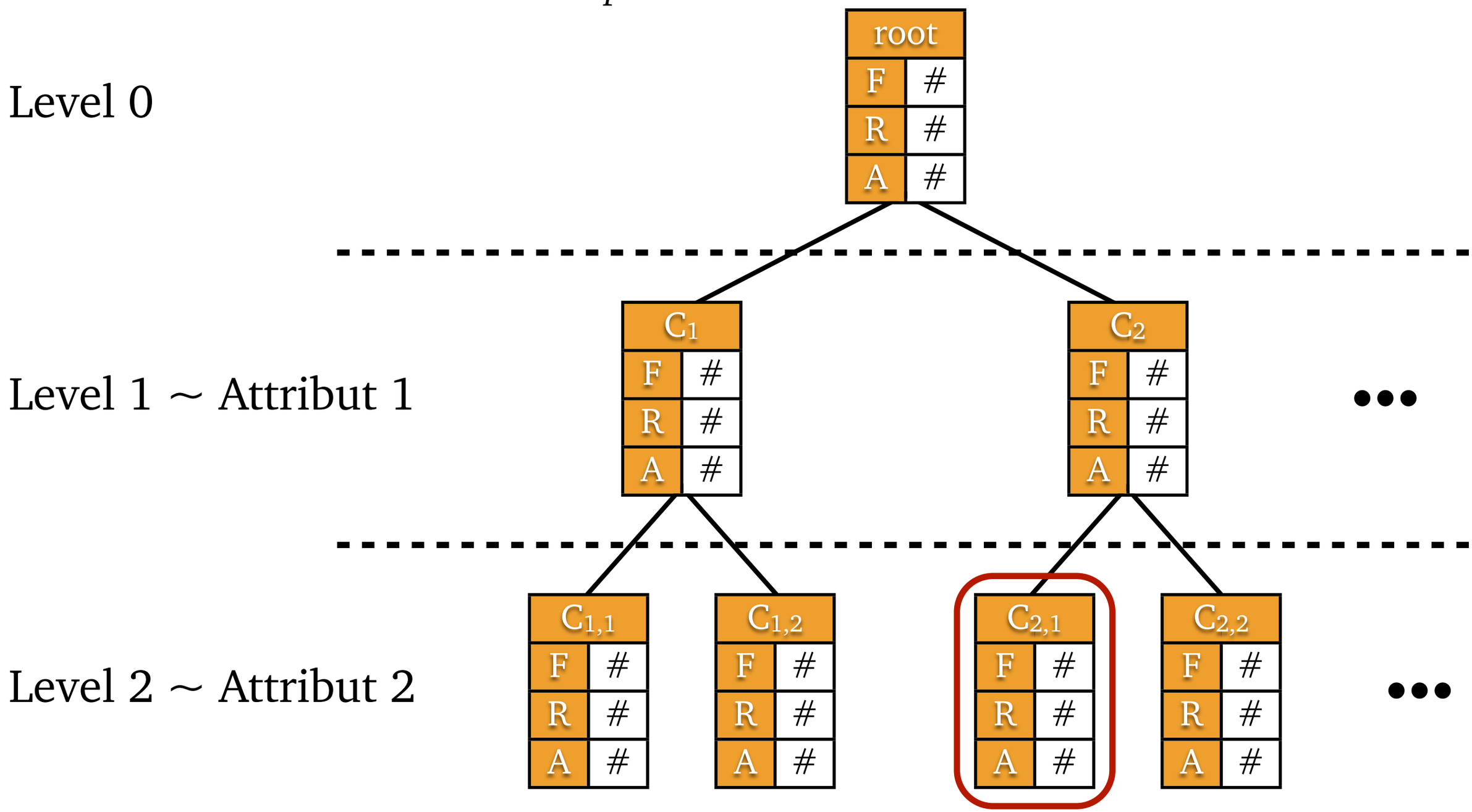
AV-Space

*Annahmen - Überblick - **Tree** - Group - CS*



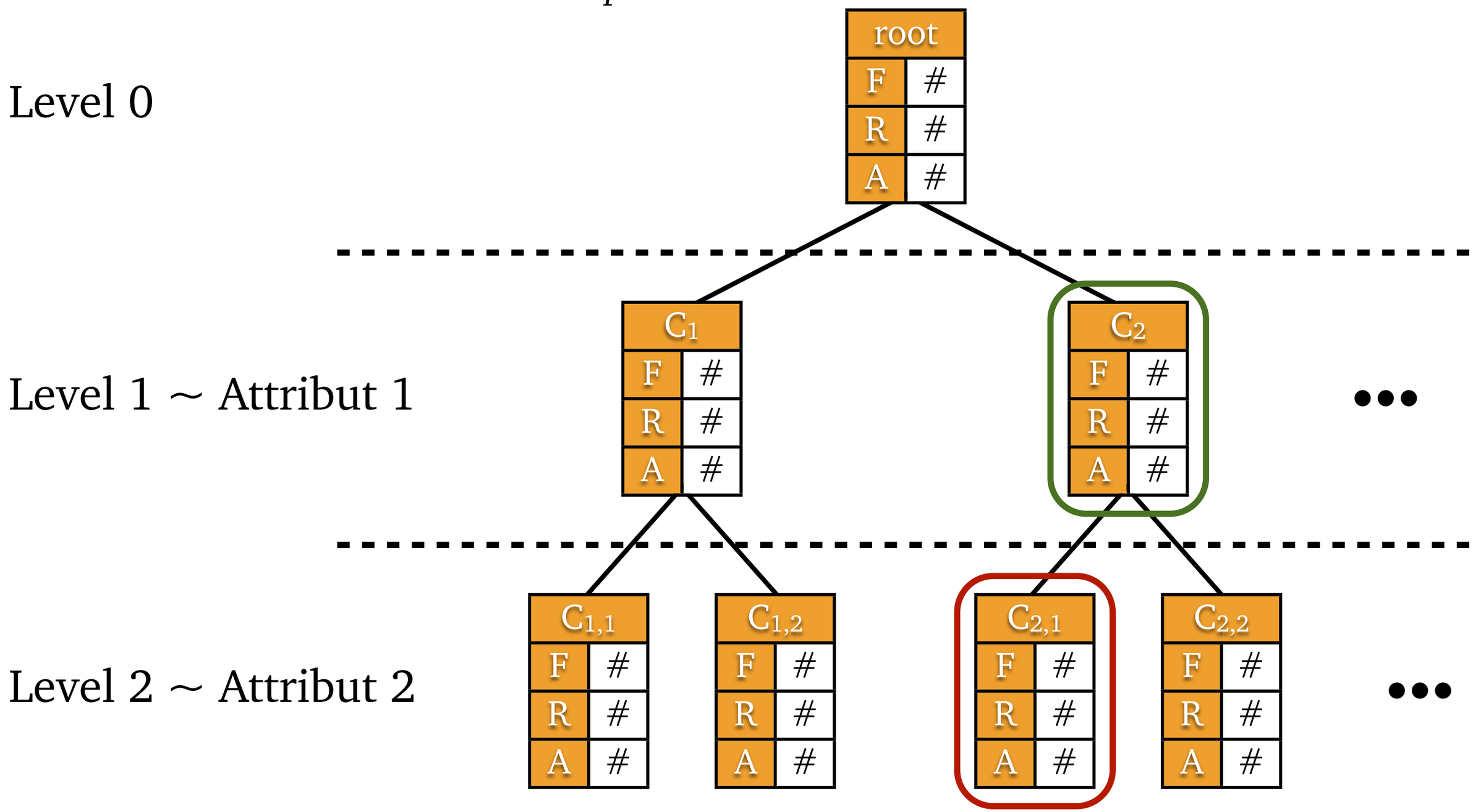
AV-Space

*Annahmen - Überblick - **Tree** - Group - CS*



AV-Space

*Annahmen - Überblick - **Tree** - Group - CS*



$1 \bullet 2$ $(a_1 = v_{1,2})$ & $(a_2 = v_{2,1})$

AV-Space

*Annahmen - Überblick - Tree - **Group** - CS*

- Trainingsdaten
  - $\{a_1, a_2, \ldots, a_M\}$ Attribute
  - $\{v_{i,j} \mid i=1\ldots M, j=1\ldots k_i\}$ Werte
  - k_i = Anzahl eindeutiger Werte zu a_i
  - a_M ist *class* Attribut
- AVC-Group
  - Summe(k_i) Elemente
  - Element besitzt:
    - Counting Set
    - Pointer

AV-Space

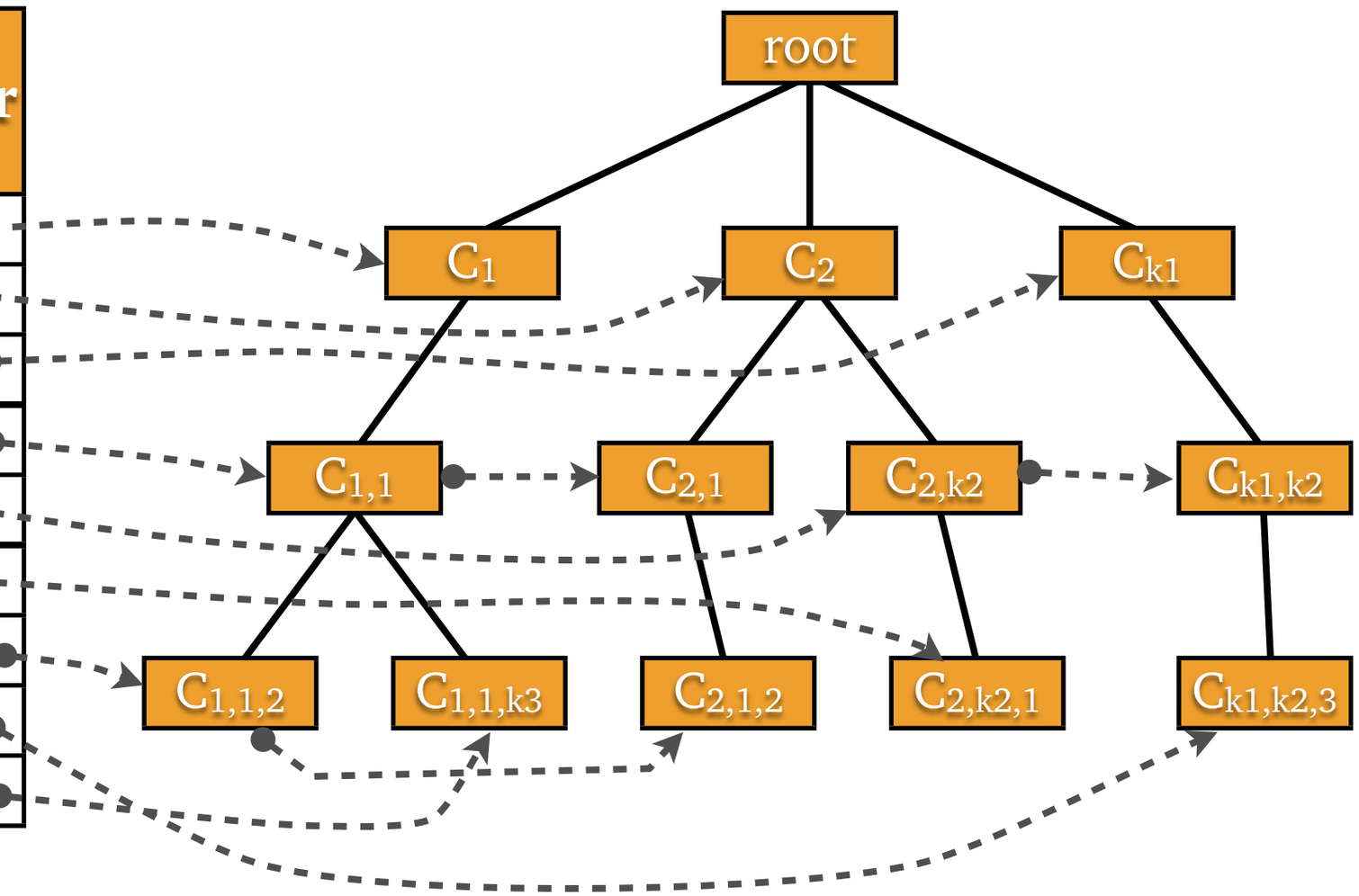
*Annahmen - Überblick - Tree - **Group** - CS*

AVP	Counting Set			Pointer
	F	R	A	
$a_1=v_{1,1}$	#	#	#	#
$a_1=v_{1,2}$	#	#	#	#
$a_1=v_{1,k1}$	#	#	#	#
$a_2=v_{2,1}$	#	#	#	#
$a_2=v_{2,k2}$	#	#	#	#
$a_3=v_{3,1}$	#	#	#	#
$a_3=v_{3,2}$	#	#	#	#
$a_3=v_{3,3}$	#	#	#	#
$a_3=v_{3,k3}$	#	#	#	#

AV-Space

*Annahmen - Überblick - Tree - **Group** - CS*

AVP	Counting Set			Pointer
	F	R	A	
$a_1=v_{1,1}$	#	#	#	#
$a_1=v_{1,2}$	#	#	#	#
$a_1=v_{1,k1}$	#	#	#	#
$a_2=v_{2,1}$	#	#	#	#
$a_2=v_{2,k2}$	#	#	#	#
$a_3=v_{3,1}$	#	#	#	#
$a_3=v_{3,2}$	#	#	#	#
$a_3=v_{3,3}$	#	#	#	#
$a_3=v_{3,k3}$	#	#	#	#



AV-Space

*Annahmen - Überblick - Tree - Group - **CS***

- Count: Family, Remaining, AVP
- Family count: # der Trainigsdaten die…
  - Zur Attr-Wert Paar Konjunktion des Knoten passen
- Remaining count: # der Trainigsdaten die…
  - Zur Attr-Wert Paar Konjunktion des Knoten passen UND
  - Nicht von den bisher erstellten Regeln abgedeckt werden
- AVP count: # der Trainigsdaten die…
  - Zur Attr-Wert Paar Konjunktion des Knoten passen UND
  - Nicht von den bisher erstellten Regeln abgedeckt werden UND
  - Von der Kandidatenregel abgedeckt werden

Algorithmen

Create - Query - Update - Komplexität

Algorithmen

Create *- Query - Update - Komplexität*

- Erstelle AVC-Group mit Sum(k_i) Einträgen
- Iteriere über alle Trainigsdaten
  - Erhöhe Zähler der Wurzel um 1
  - Gehe von der Wurzel aus los
  - Iteriere über alle Attribut-Wert Paare des Beispiels
    - Wenn der passende Nachfolger Knoten existiert, erhöhe Zähler
    - Ansonsten: erstelle ihn, Zähler auf 1, mit Pointer aus Tabelle verbinden
    - Erhöhe die Zähler der entsprechenden Gruppe um 1

Algorithmen

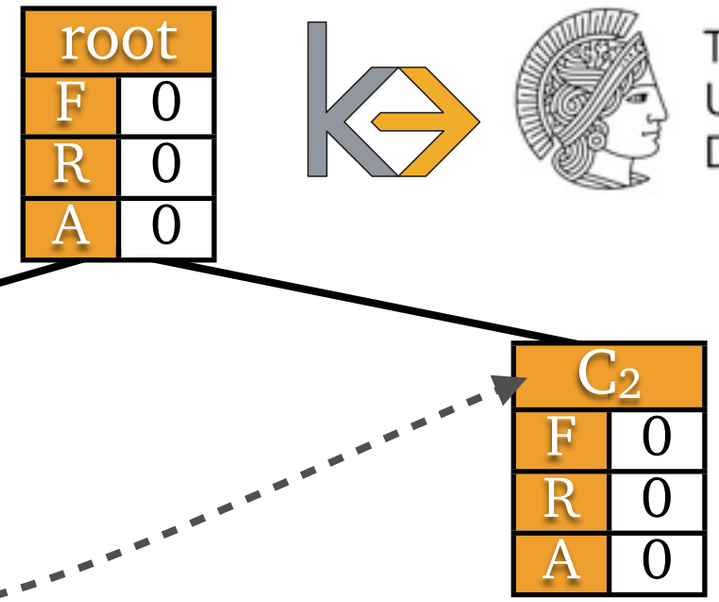
***Create** - Query - Update - Komplexität*

Patient	Headache	Pain	Temperature	Flu
x1	Yes	Yes	Normal	No
x2	Yes	Yes	High	Yes
x3	Yes	Yes	Very High	Yes
x4	No	No	High	No
x5	No	Yes	Normal	No
x6	No	No	High	No
x7	No	Yes	Very High	Yes
x8	No	No	High	Yes
x9	No	Yes	Very High	No
x10	No	No	High	No

Algorithmen

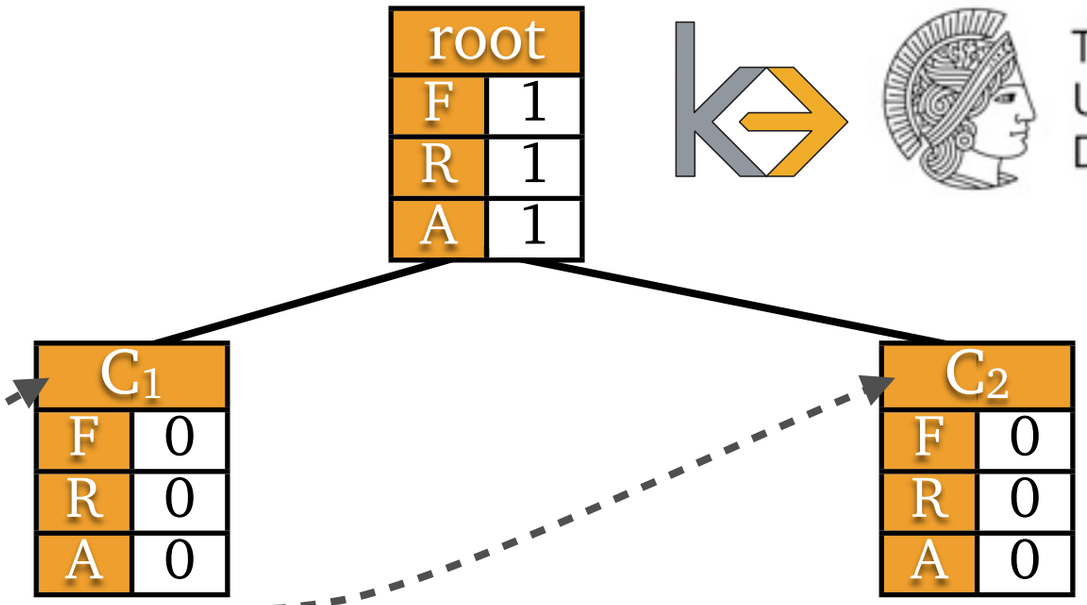
Create *- Query - Update - Komplexität*

AVP	Counting Set		
	F	R	A
<H=N>	0	0	0
<H=Y>	0	0	0
<P=N>	0	0	0
<P=Y>	0	0	0
<T=N>	0	0	0
<T=H>	0	0	0
<T=V>	0	0	0
<F=N>	0	0	0
<F=Y>	0	0	0



Algorithmen

Create *- Query - Update - Komplexität*



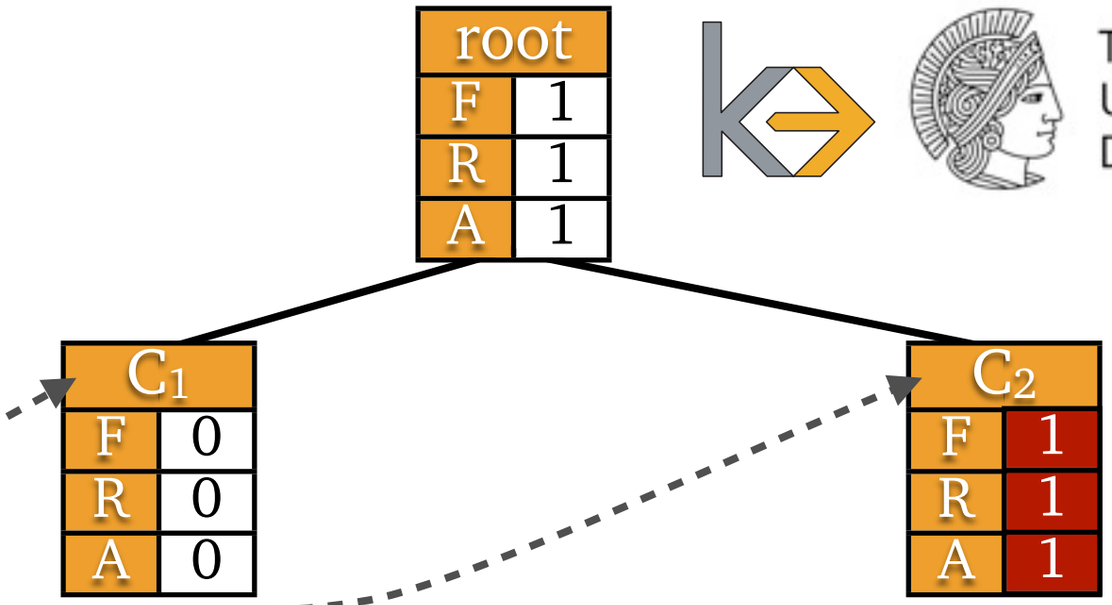
AVP	Counting Set		
	F	R	A
<H=N>	0	0	0
<H=Y>	0	0	0
<P=N>	0	0	0
<P=Y>	0	0	0
<T=N>	0	0	0
<T=H>	0	0	0
<T=V>	0	0	0
<F=N>	0	0	0
<F=Y>	0	0	0

Patient	Headache	Pain	Temperature	Flu
x1	Yes	Yes	Normal	No

Algorithmen

Create - Query - Update - Komplexität

AVP	Counting Set		
	F	R	A
<H=N>	0	0	0
<H=Y>	1	1	1
<P=N>	0	0	0
<P=Y>	0	0	0
<T=N>	0	0	0
<T=H>	0	0	0
<T=V>	0	0	0
<F=N>	0	0	0
<F=Y>	0	0	0

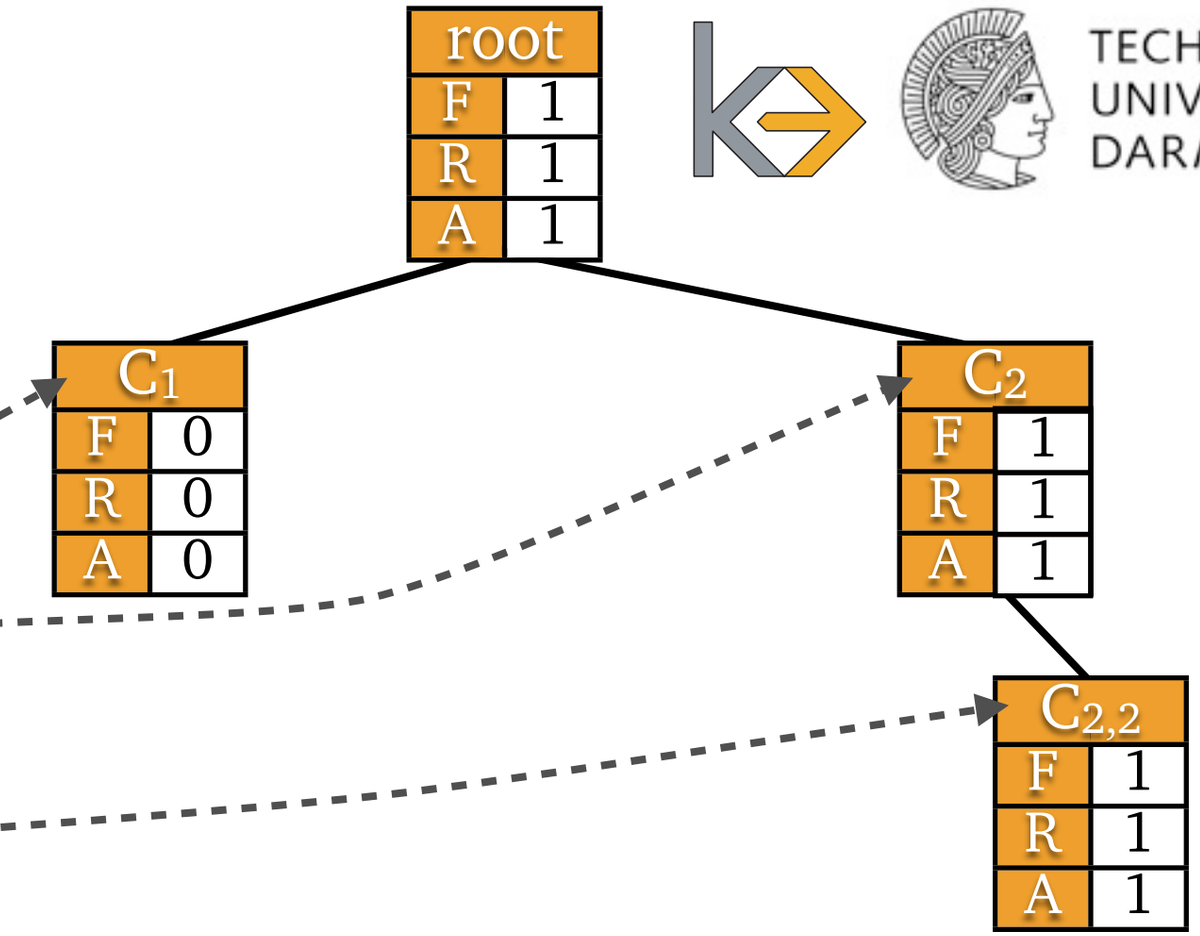


Patient	Headache	Pain	Temperature	Flu
x1	Yes	Yes	Normal	No

Algorithmen

Create - Query - Update - Komplexität

AVP	Counting Set		
	F	R	A
<H=N>	0	0	0
<H=Y>	1	1	1
<P=N>	0	0	0
<P=Y>	1	1	1
<T=N>	0	0	0
<T=H>	0	0	0
<T=V>	0	0	0
<F=N>	0	0	0
<F=Y>	0	0	0

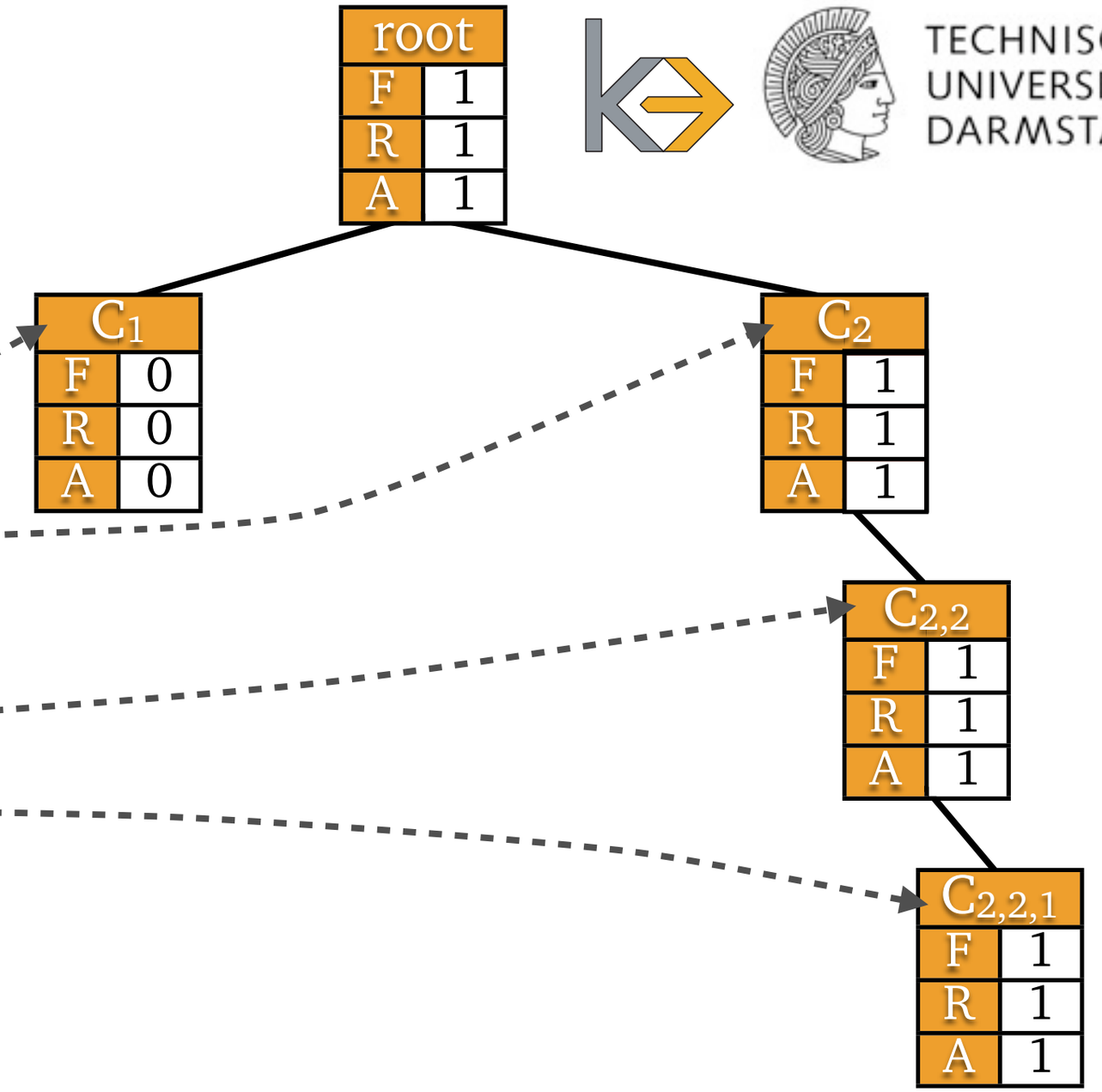


Patient	Headache	Pain	Temperature	Flu
x1	Yes	Yes	Normal	No

Algorithmen

***Create** - Query - Update - Komplexität*

AVP	Counting Set		
	F	R	A
<H=N>	0	0	0
<H=Y>	1	1	1
<P=N>	0	0	0
<P=Y>	1	1	1
<T=N>	1	1	1
<T=H>	0	0	0
<T=V>	0	0	0
<F=N>	0	0	0
<F=Y>	0	0	0

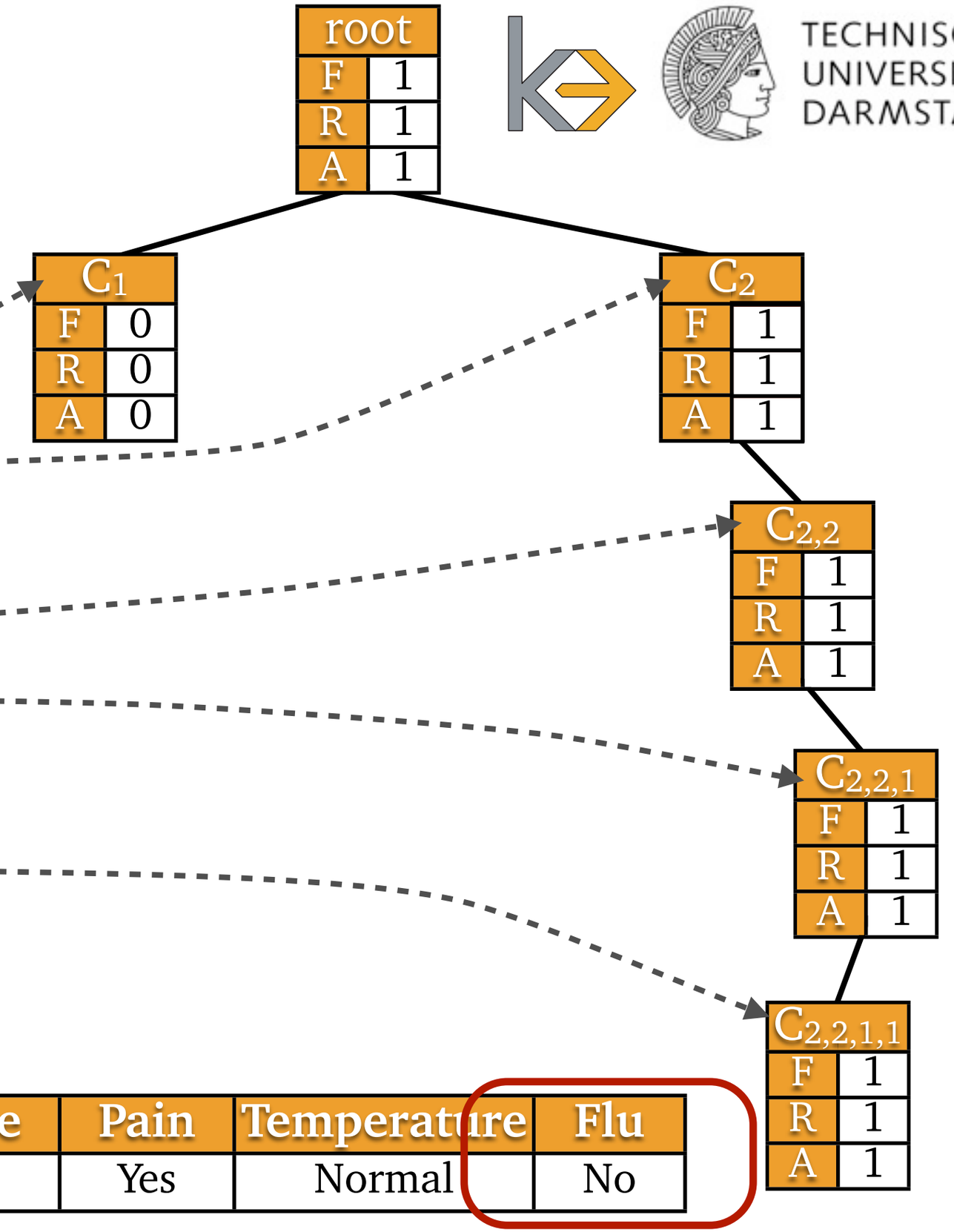


Patient	Headache	Pain	Temperature	Flu
x1	Yes	Yes	Normal	No

Algorithmen

Create *- Query - Update - Komplexität*

AVP	Counting Set		
	F	R	A
<H=N>	0	0	0
<H=Y>	1	1	1
<P=N>	0	0	0
<P=Y>	1	1	1
<T=N>	1	1	1
<T=H>	0	0	0
<T=V>	0	0	0
<F=N>	1	1	1
<F=Y>	0	0	0

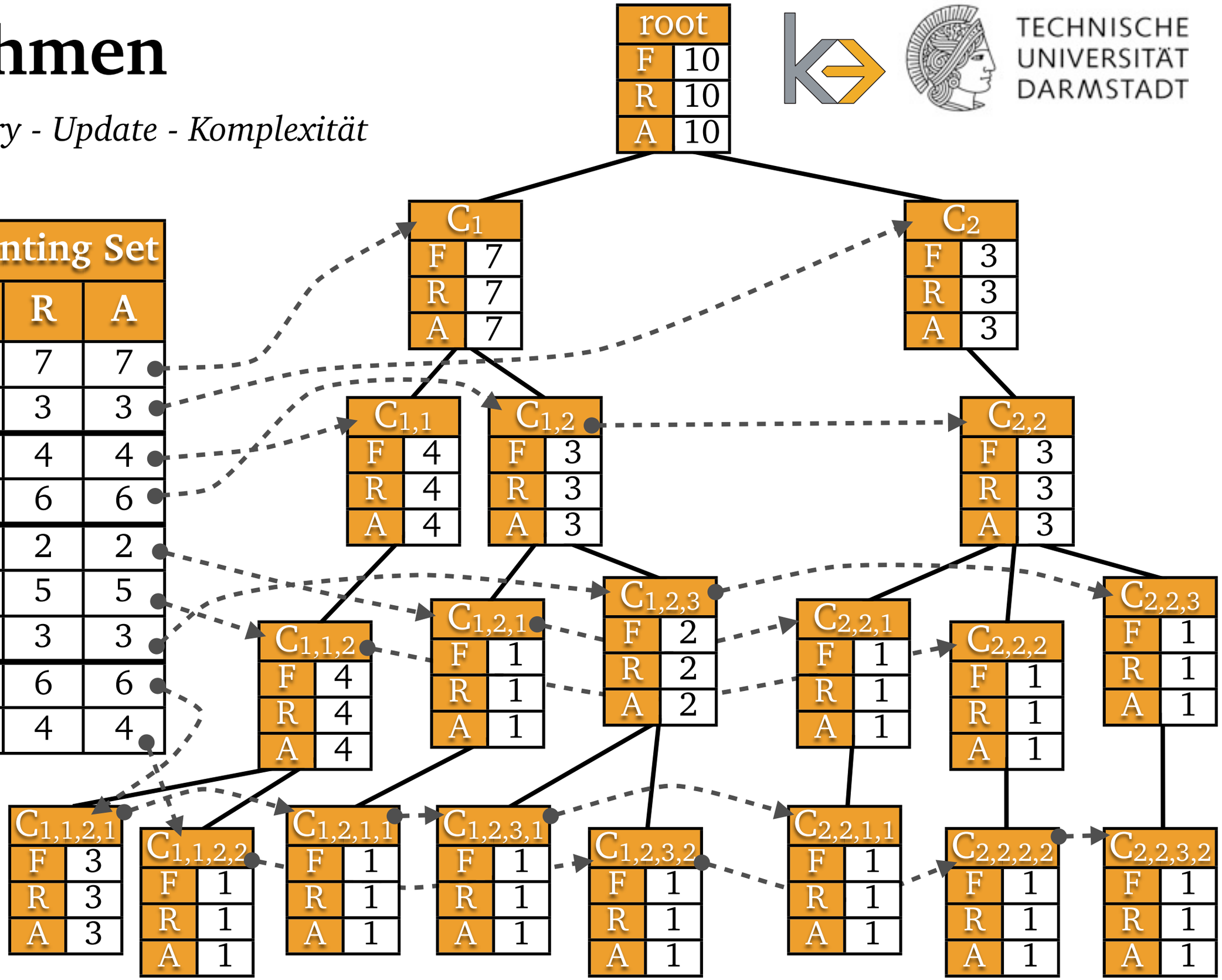


Patient	Headache	Pain	Temperature	Flu
x1	Yes	Yes	Normal	No

Algorithmen

Create *- Query - Update - Komplexität*

AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	3	3
<P=N>	4	4	4
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	5
<T=V>	3	3	3
<F=N>	6	6	6
<F=Y>	4	4	4



Algorithmen

*Create - **Query** - Update - Komplexität*

- Top-down Vorgehen
- Zähler Typen decken verschiedenen Situationen ab

$$<a_i=v_{i1} \text{ or } v_{i2} \text{ or } \ldots> \text{ and } <a_j = v_{j1} \text{ or } v_{j2} \text{ or } \ldots> \ldots$$

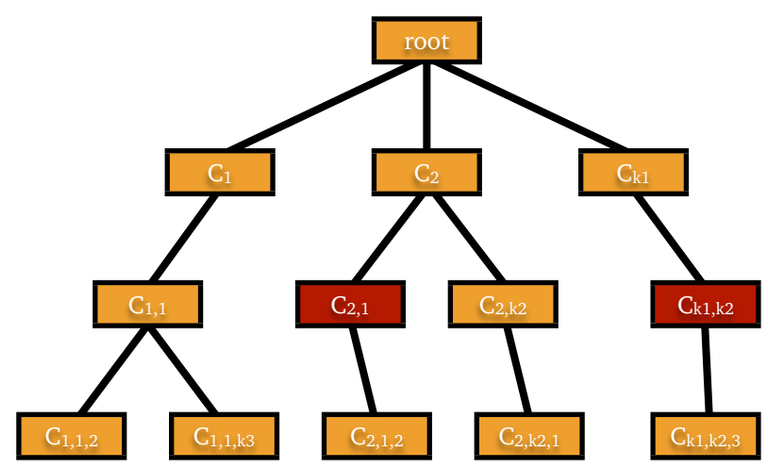
Algorithmen

*Create - **Query** - Update - Komplexität*

- Top-down Vorgehen
- Zähler Typen decken verschiedenen Situationen ab

$<a_i=v_{i1}$ or v_{i2} or $\ldots>$ and $<a_j = v_{j1}$ or v_{j2} or $\ldots>$ …

AVP	Counting Set		
	F	R	A
<H=N>	#	#	#
<H=Y>	#	#	#
<P=N>	#	#	#
<P=Y>	#	#	#
<T=N>	#	#	#
<T=H>	#	#	#
<T=V>	#	#	#
<F=N>	#	#	#
<F=Y>	#	#	#



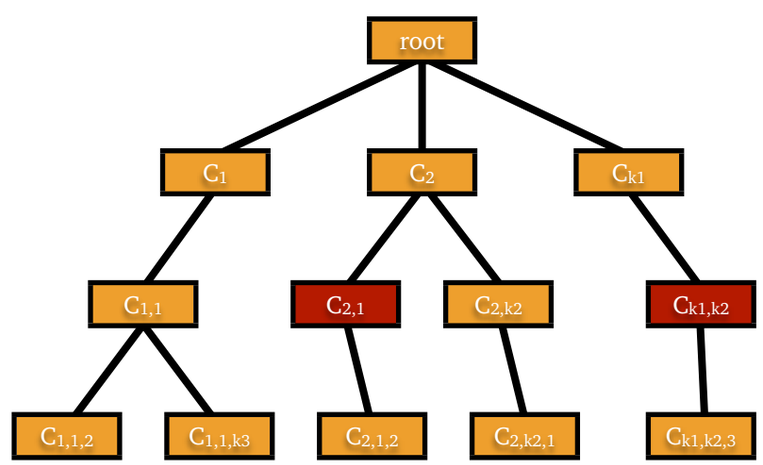
Algorithmen

*Create - **Query** - Update - Komplexität*

- Top-down Vorgehen
- Zähler Typen decken verschiedenen Situationen ab

$<a_i=v_{i1}$ or v_{i2} or …> and $<a_j = v_{j1}$ or v_{j2} or …> …

AVP	Counting Set		
	F	R	A
<H=N>	#	#	#
<H=Y>	#	#	#
<P=N>	#	#	#
<P=Y>	#	#	#
<T=N>	#	#	#
<T=H>	#	#	#
<T=V>	#	#	#
<F=N>	#	#	#
<F=Y>	#	#	#

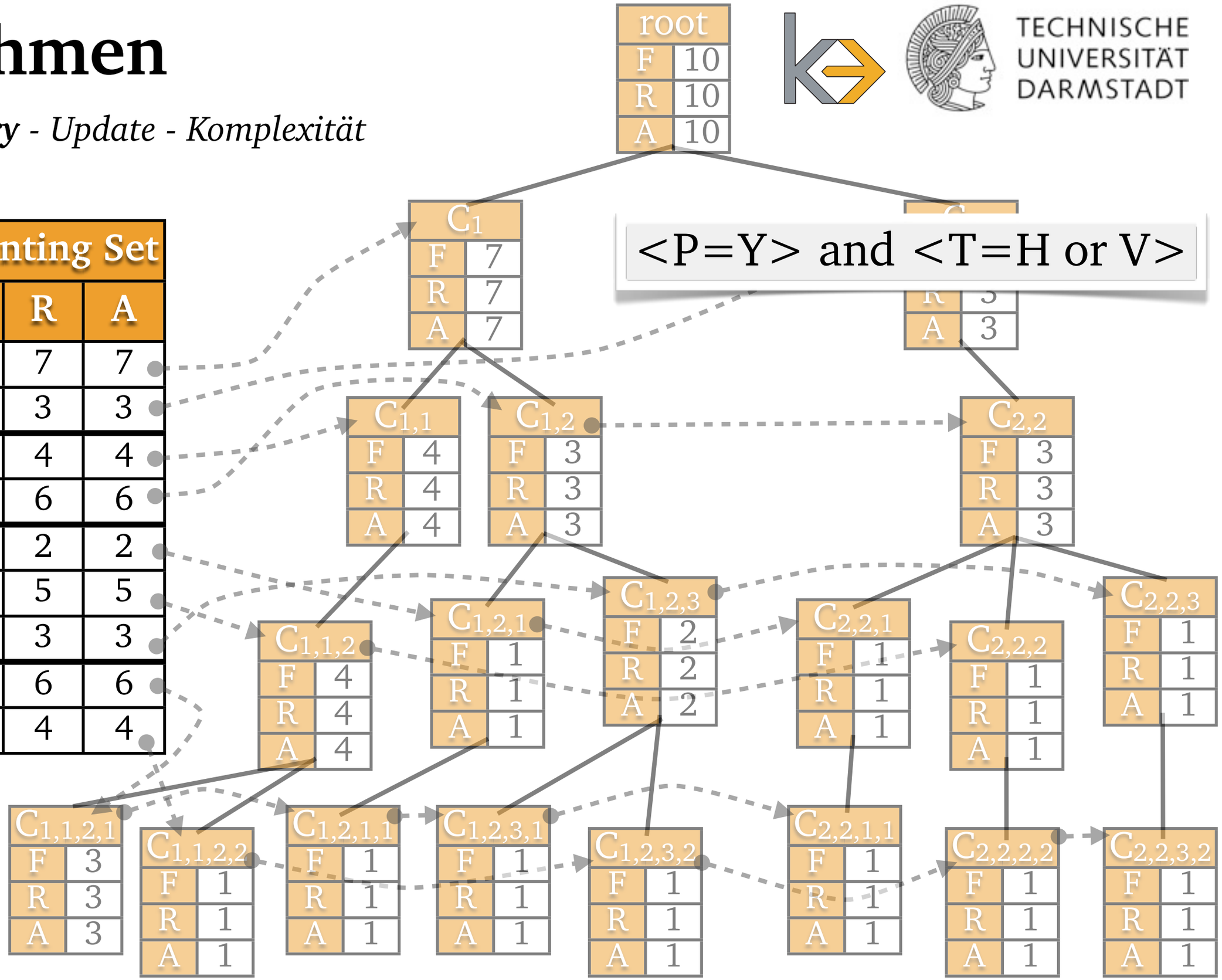


+ # matches

Algorithmen

*Create - **Query** - Update - Komplexität*

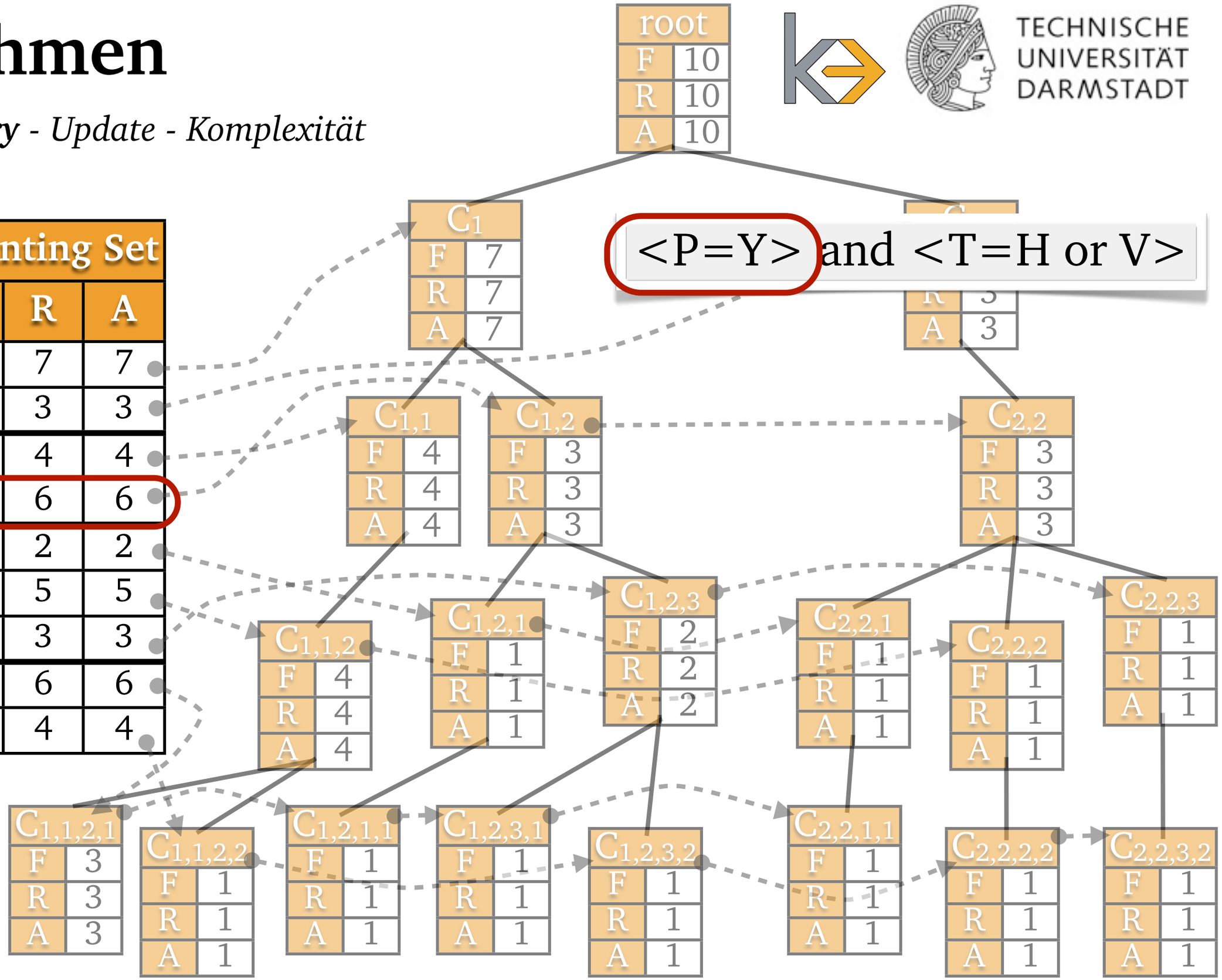
AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	3	3
<P=N>	4	4	4
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	5
<T=V>	3	3	3
<F=N>	6	6	6
<F=Y>	4	4	4



Algorithmen

*Create - **Query** - Update - Komplexität*

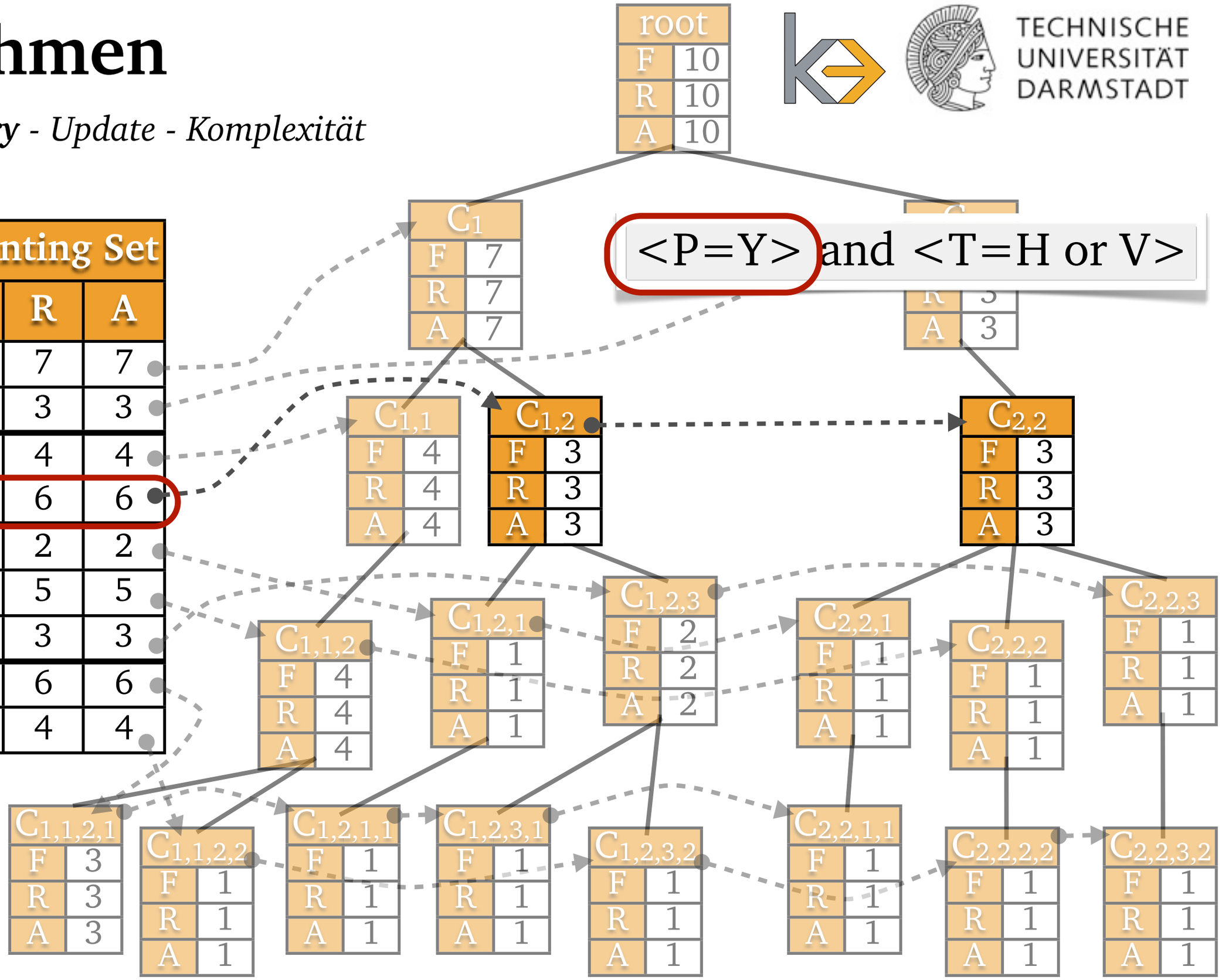
AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	3	3
<P=N>	4	4	4
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	5
<T=V>	3	3	3
<F=N>	6	6	6
<F=Y>	4	4	4



Algorithmen

*Create - **Query** - Update - Komplexität*

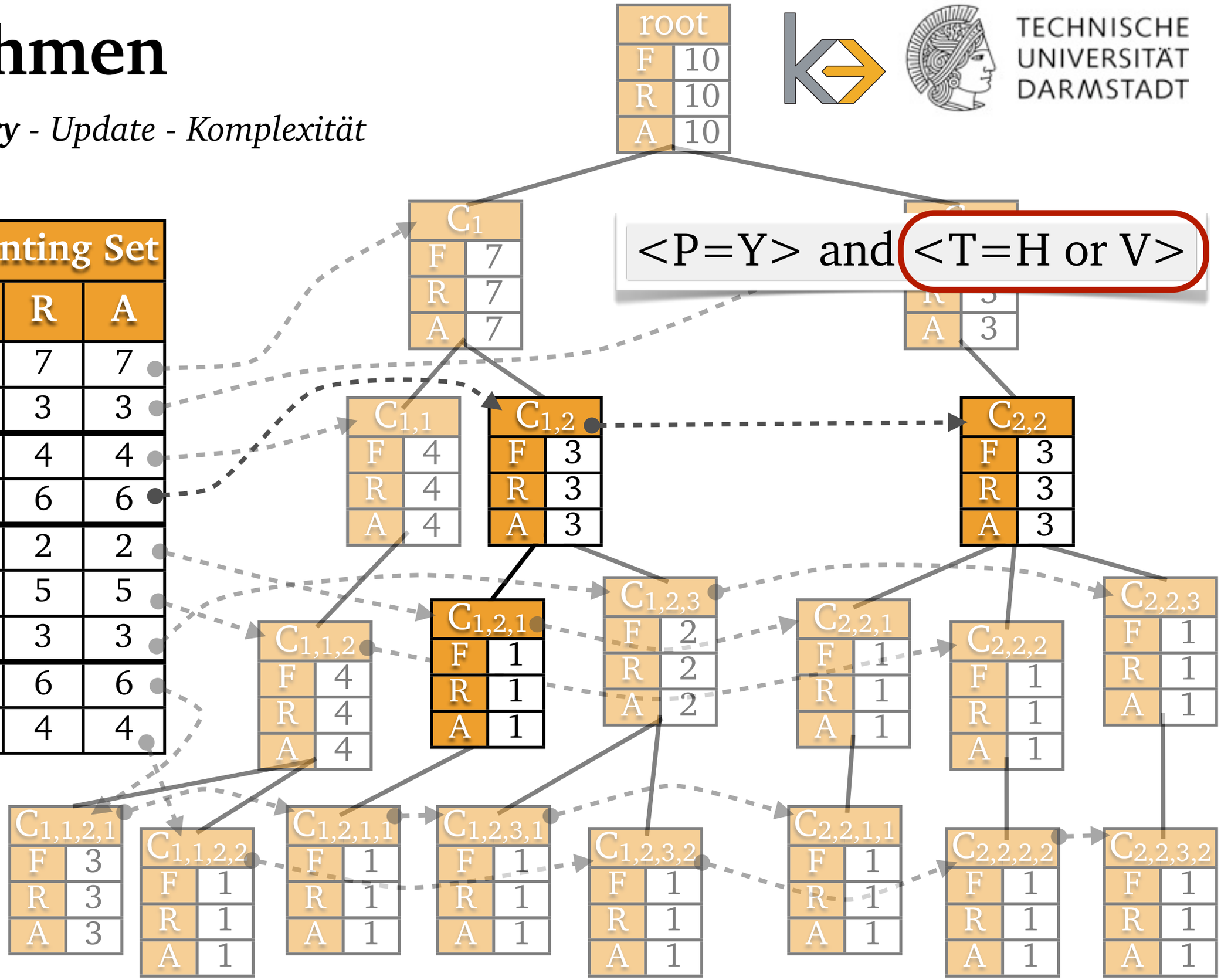
AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	3	3
<P=N>	4	4	4
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	5
<T=V>	3	3	3
<F=N>	6	6	6
<F=Y>	4	4	4



Algorithmen

*Create - **Query** - Update - Komplexität*

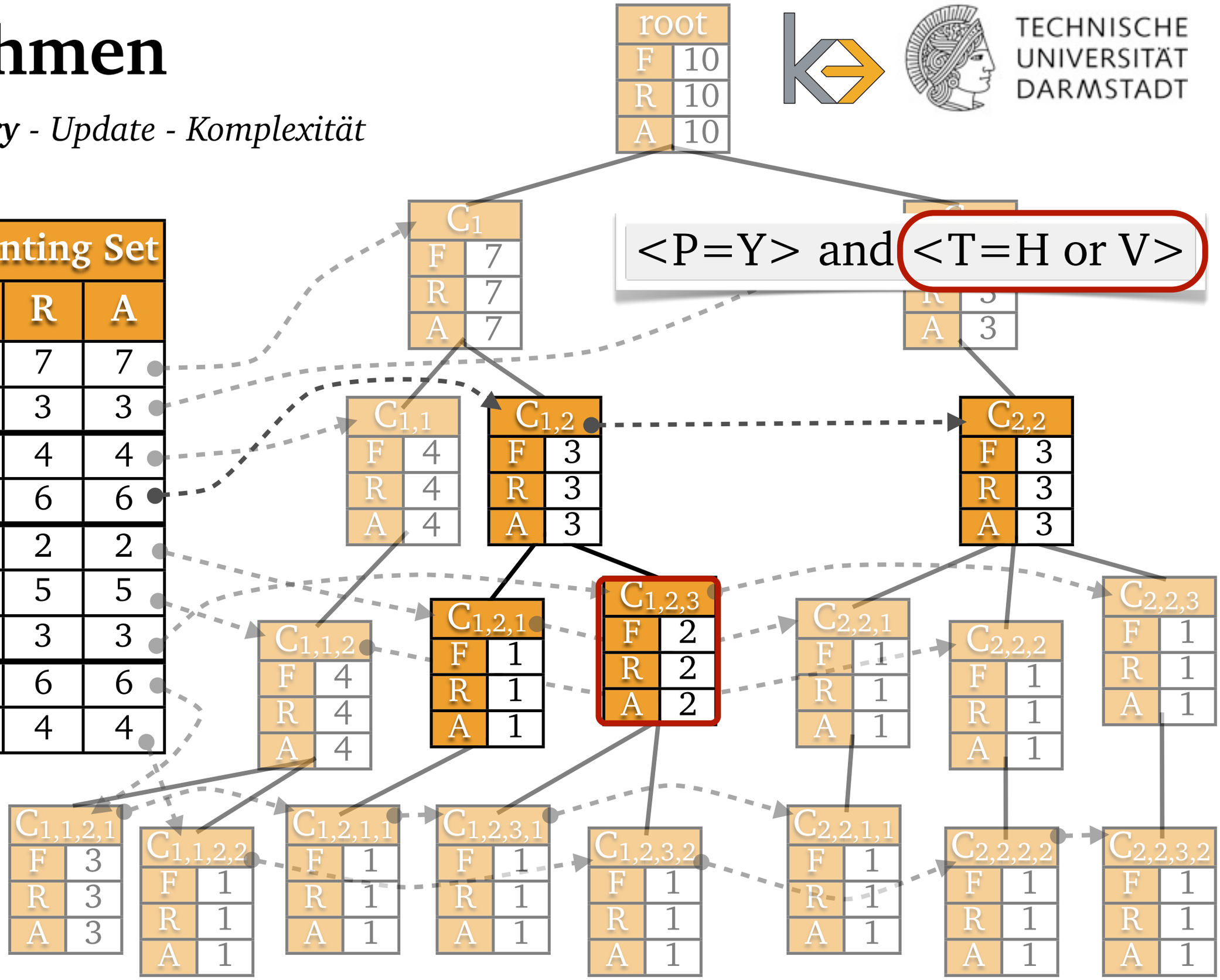
AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	3	3
<P=N>	4	4	4
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	5
<T=V>	3	3	3
<F=N>	6	6	6
<F=Y>	4	4	4



Algorithmen

*Create - **Query** - Update - Komplexität*

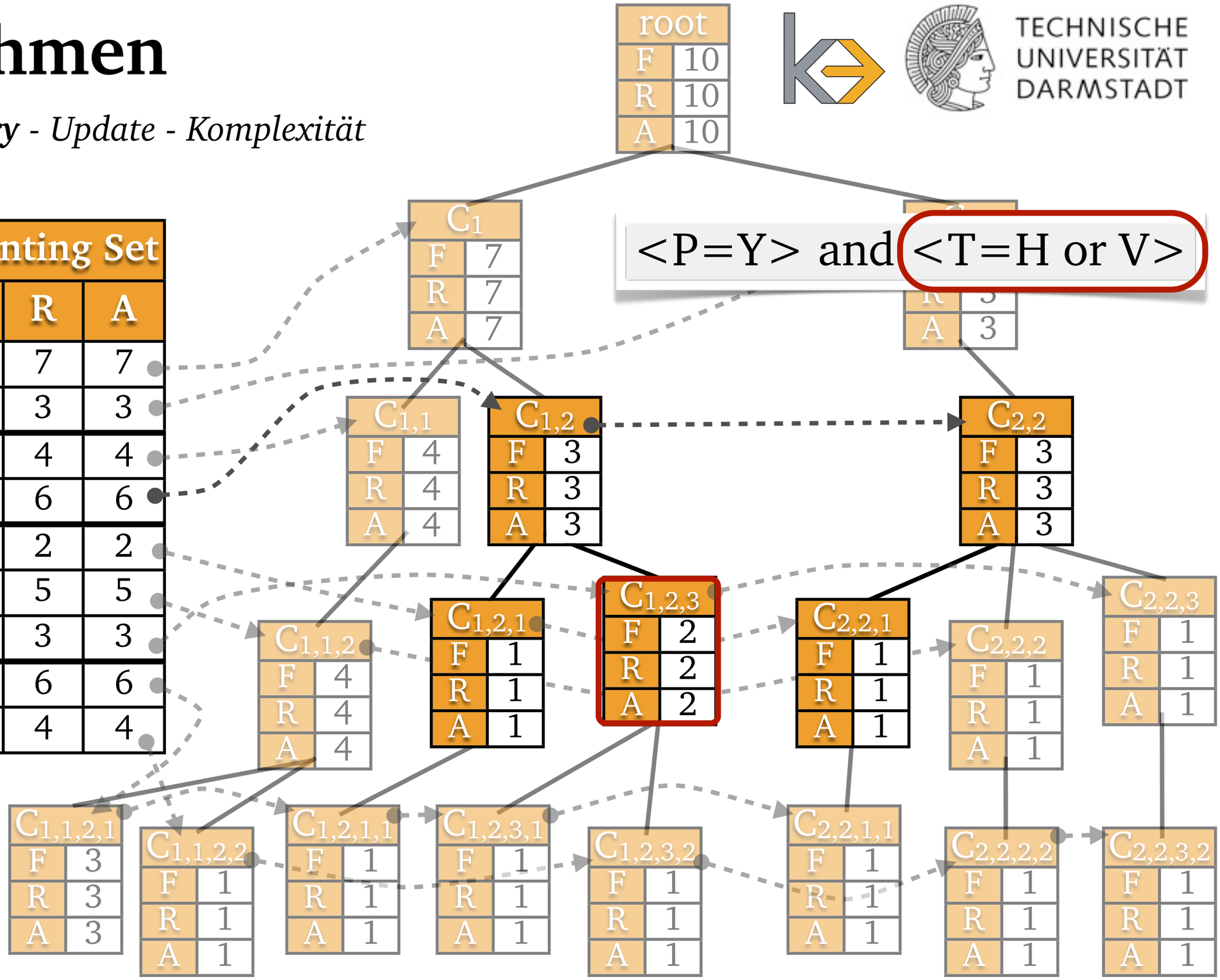
AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	3	3
<P=N>	4	4	4
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	5
<T=V>	3	3	3
<F=N>	6	6	6
<F=Y>	4	4	4



Algorithmen

*Create - **Query** - Update - Komplexität*

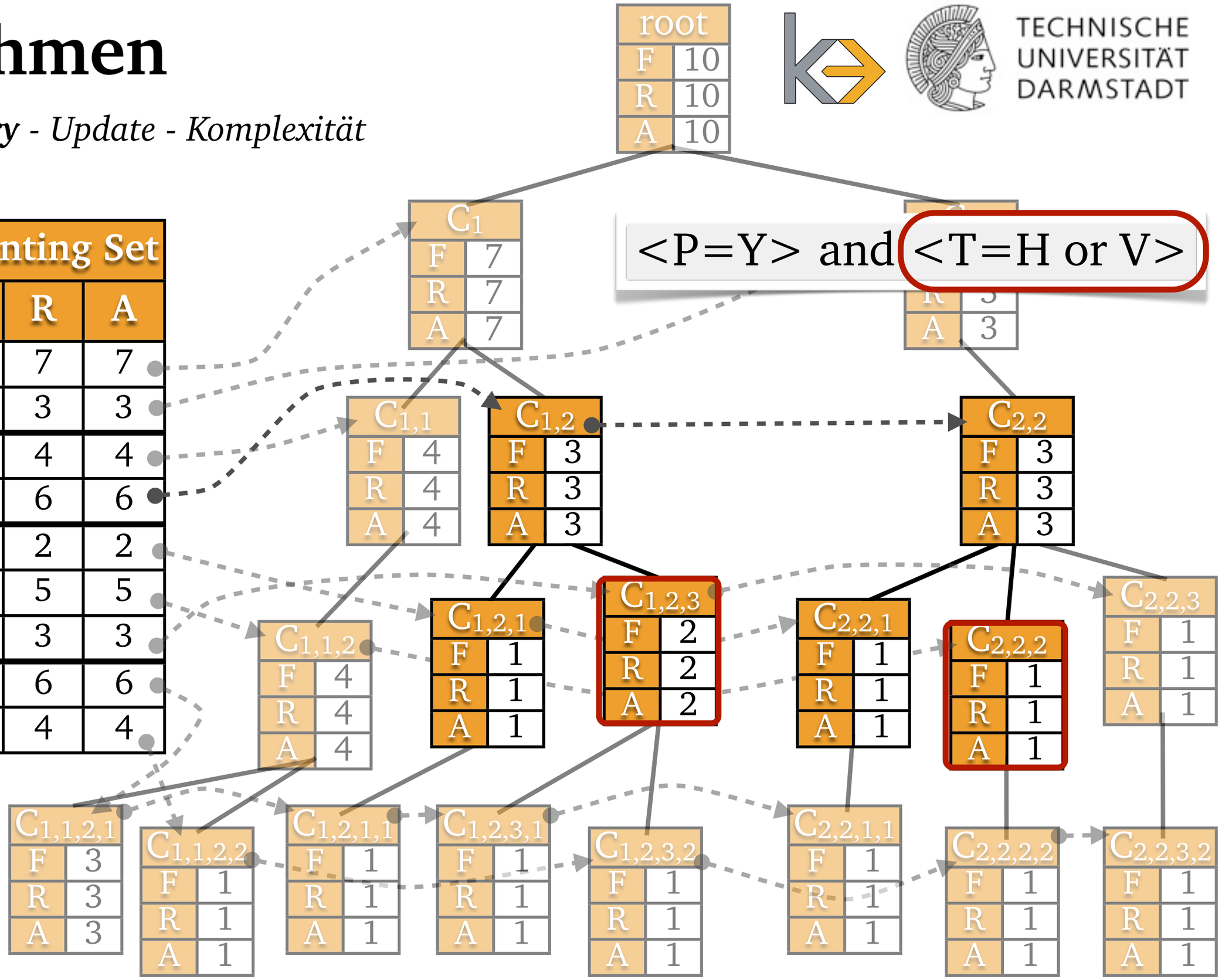
AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	3	3
<P=N>	4	4	4
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	5
<T=V>	3	3	3
<F=N>	6	6	6
<F=Y>	4	4	4



Algorithmen

*Create - **Query** - Update - Komplexität*

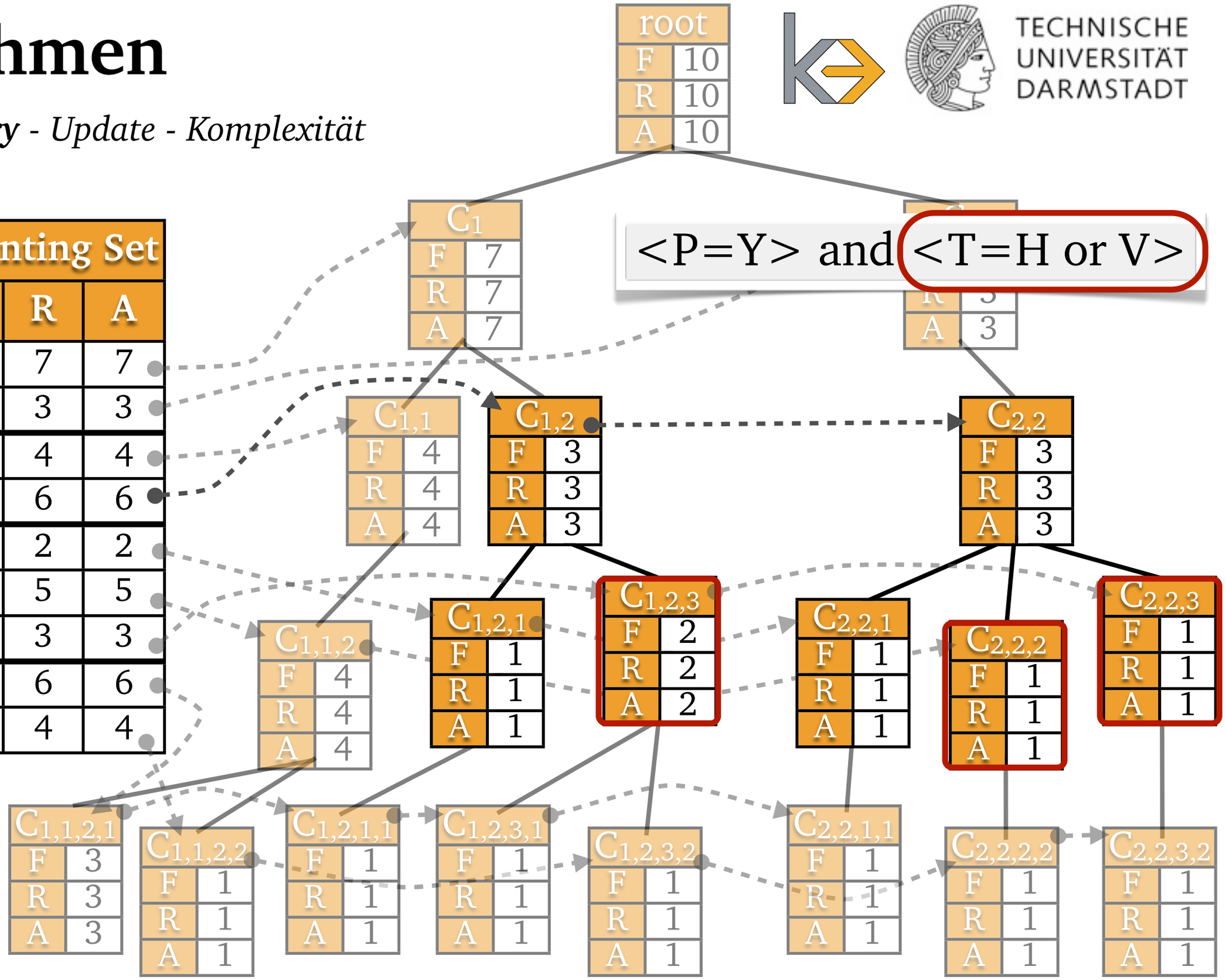
AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	3	3
<P=N>	4	4	4
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	5
<T=V>	3	3	3
<F=N>	6	6	6
<F=Y>	4	4	4



Algorithmen

*Create - **Query** - Update - Komplexität*

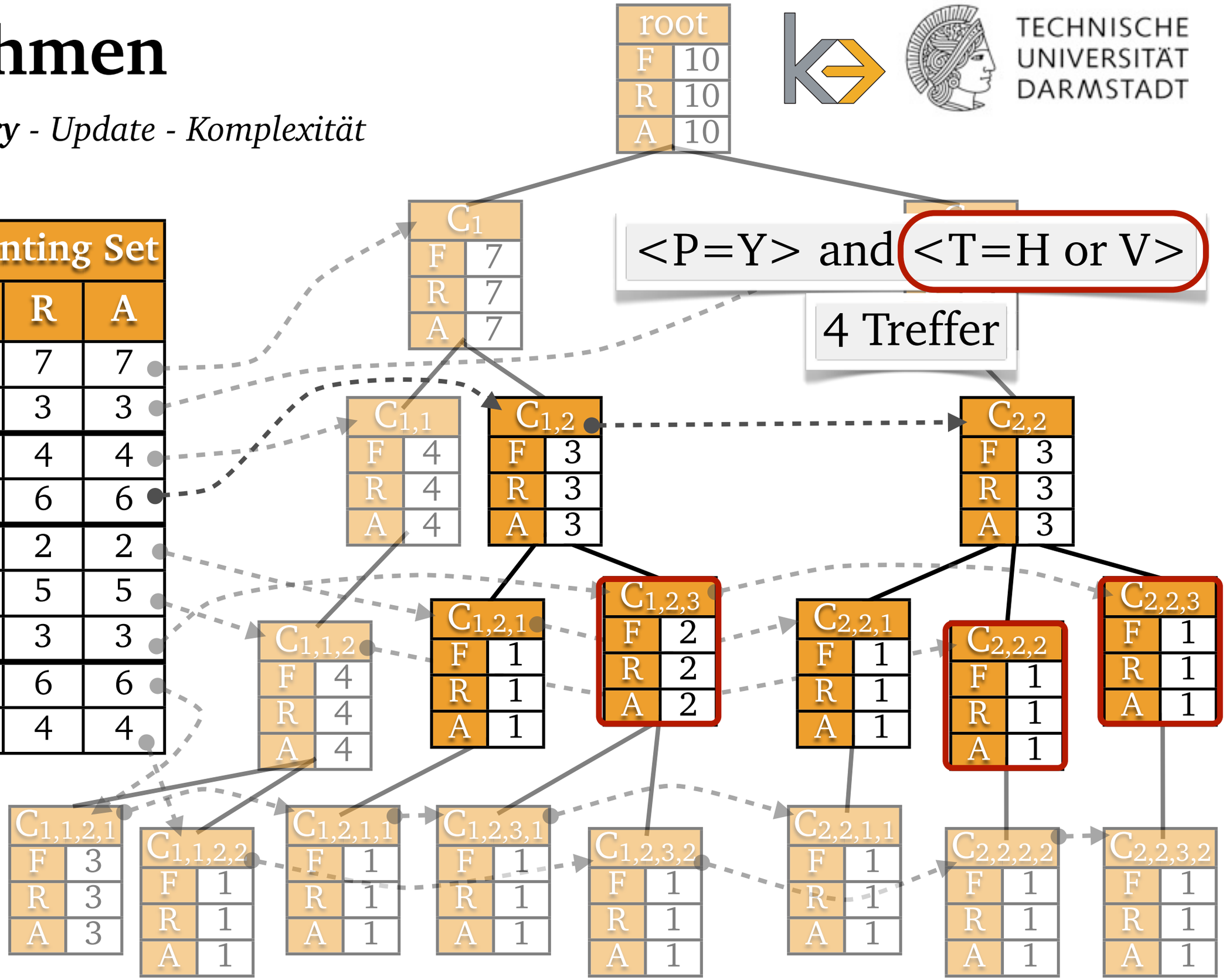
AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	3	3
<P=N>	4	4	4
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	5
<T=V>	3	3	3
<F=N>	6	6	6
<F=Y>	4	4	4



Algorithmen

*Create - **Query** - Update - Komplexität*

AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	3	3
<P=N>	4	4	4
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	5
<T=V>	3	3	3
<F=N>	6	6	6
<F=Y>	4	4	4



Algorithmen

*Create - Query - **Update** - Komplexität*

- 3 Fälle, Update nach...
  - Hinzufügen einer Regel zur Regelmenge
  - Hinzufügen eines Attr-Wert Paares zur Kandidatenregel
  - Erlernen einer Klasse

Algorithmen

*Create - Query - **Update** - Komplexität*

- Update nach *Erlernen einer Klasse*
  - AVP & Remaining Count stimmen nicht mehr
  - kopiere Family Count auf die anderen Zähler

Algorithmen

*Create - Query - **Update** - Komplexität*

- Update nach *Hinzufügen eines Attr-Wert Paares zur Kandidatenregel*
  - Manche AVP Counts müssen angepasst werden
  - „Invertiere“ hinzugefügtes Paar
  - Setzte alle AVP Counts die zu der invertierten Suche passen, auf Null
  - Passe die Vorgänger/Nachfolger Knoten an
  - Passe die Einträge in der AVC-Group an

Algorithmen

*Create - Query - **Update** - Komplexität*

- Beispiel: <Pain=Yes> wird zur leeren Kandidatenregel hinzugefügt
  - Invertiere <Pain=Yes> zu <Pain=No>
  - Setzte bei allen Passenden Einträgen AVP Count auf Null
  - Passe die Vorgänger/Nachfolger Knoten an
  - Passe die Einträge in der AVC-Group an

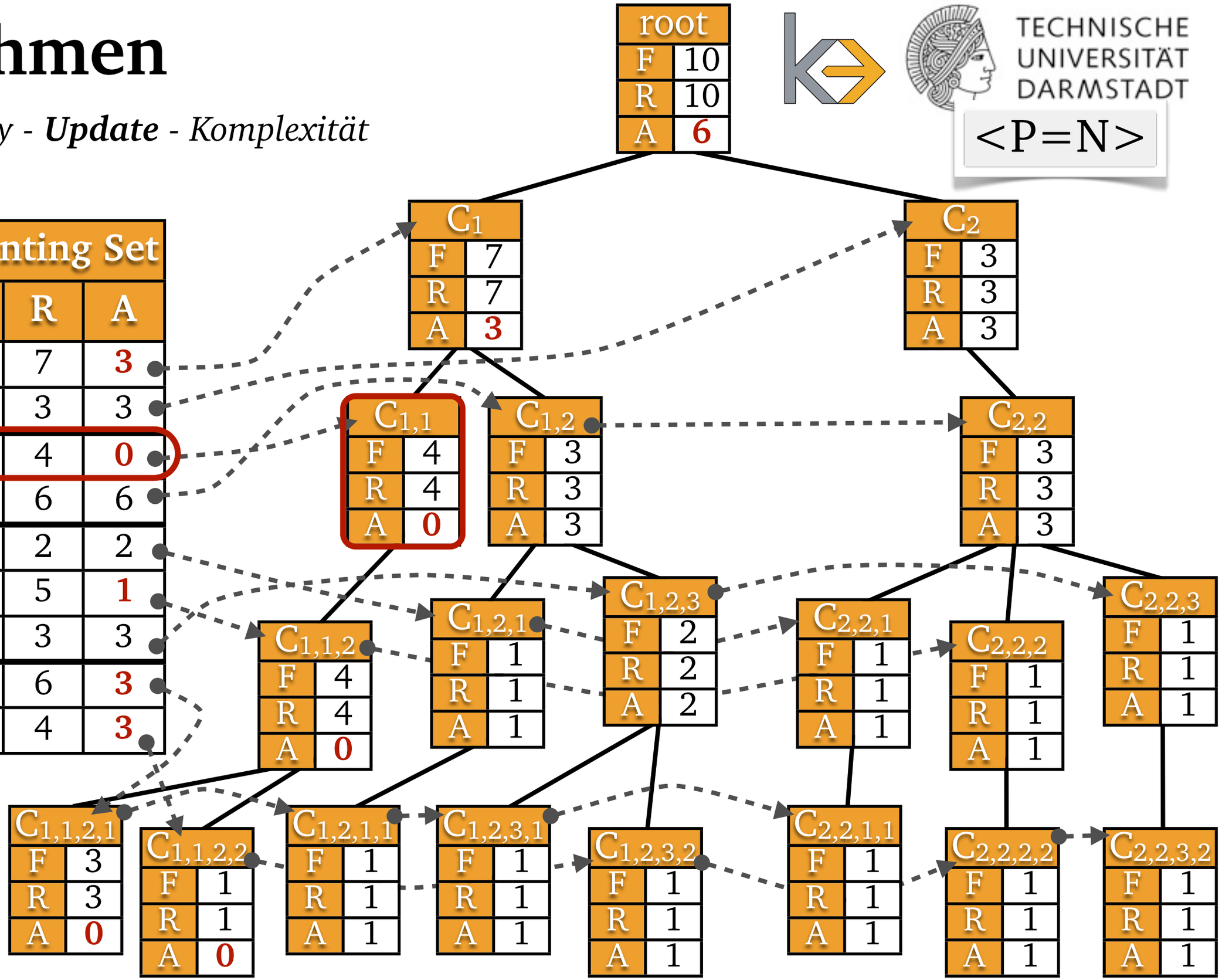
Algorithmen

*Create - Query - **Update** - Komplexität*

TECHNISCHE UNIVERSITÄT DARMSTADT

<P=N>

AVP	Counting Set		
	F	R	A
<H=N>	7	7	3
<H=Y>	3	3	3
<P=N>	4	4	0
<P=Y>	6	6	6
<T=N>	2	2	2
<T=H>	5	5	1
<T=V>	3	3	3
<F=N>	6	6	3
<F=Y>	4	4	3



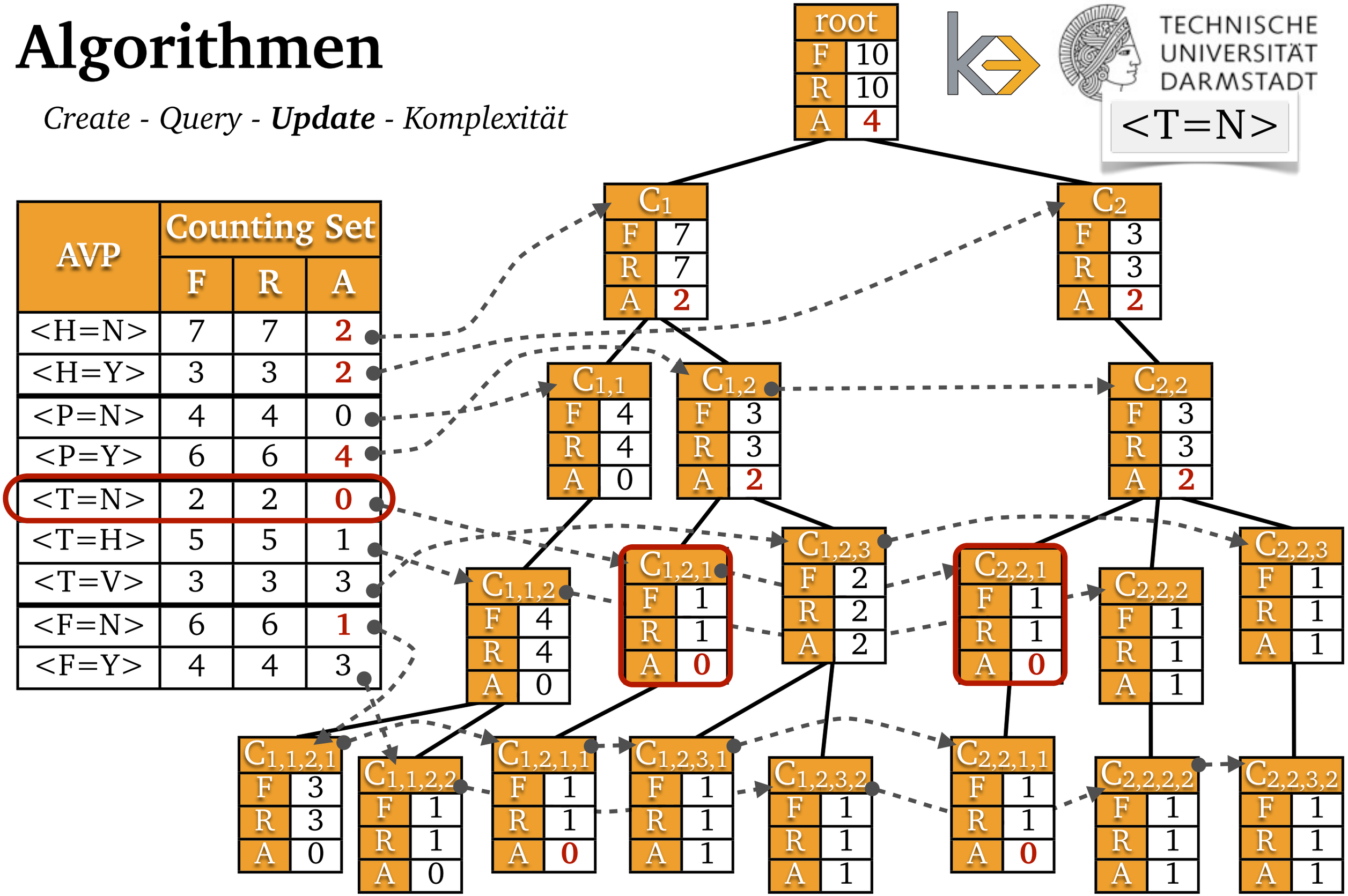
Algorithmen

*Create - Query - **Update** - Komplexität*

- Beispiel: Kandidatenregel ist <Pain=Yes>
- Nun wird <Temperature=Very High or High> hinzugefügt
  - Invertiere <Temperature=Very High or High> zu <Temperature=Normal>
  - Setzte bei allen Passenden Einträgen AVP Count auf Null
  - Passe die Vorgänger/Nachfolger Knoten an
  - Passe die Einträge in der AVC-Group an

Algorithmen

*Create - Query - **Update** - Komplexität*



AVP	Counting Set		
	F	R	A
<H=N>	7	7	2
<H=Y>	3	3	2
<P=N>	4	4	0
<P=Y>	6	6	4
<T=N>	2	2	0
<T=H>	5	5	1
<T=V>	3	3	3
<F=N>	6	6	1
<F=Y>	4	4	3

Algorithmen

*Create - Query - **Update** - Komplexität*

- Hinzufügen einer Regel zur Regelmenge
  - AVP & Remaining Count stimmen nicht mehr
  - Schreibe die neue Regel in eine Abfrage um
    - Finde alle Blätter die auf diese Abfrage passen
    - Subtrahiere von den Remaining Counts aller Vorfahren den Remaining Count der Blätter
    - Setzte den Remaining Count der Blätter auf Null
  - Setzte alle AVP Counts auf den Wert der Remaining Counts
  - Update die Tabelle

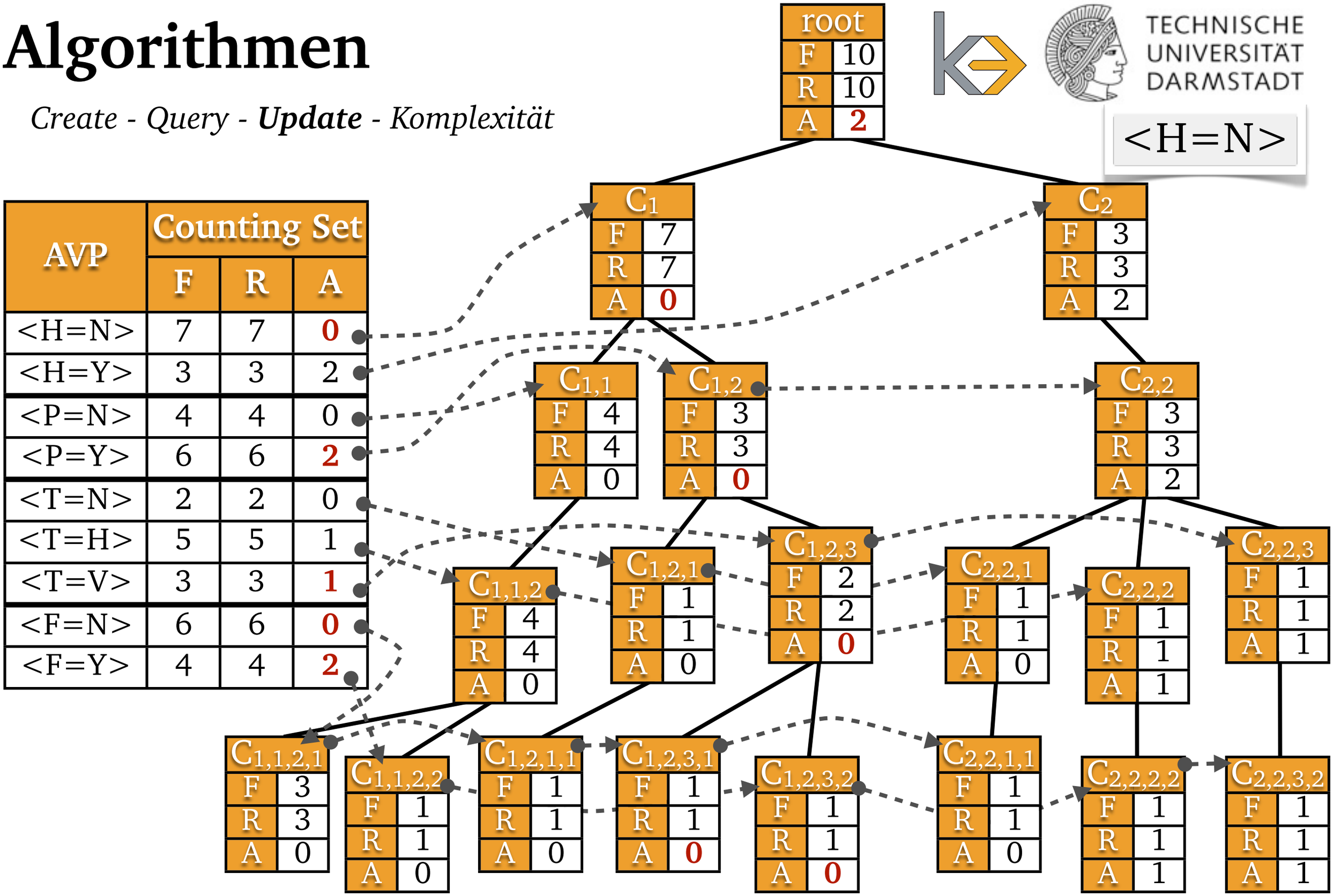
Algorithmen

*Create - Query - **Update** - Komplexität*

- Beispiel:
  - Kandidatenregel:
    <P=Y> and <T=V or H> and <H=Y> then <F=Y>
  - Umgeschrieben als Abfrage:
    <P=Y> and <T=V or H> and <H=Y> **and** <F=Y>

Algorithmen

*Create - Query - **Update** - Komplexität*

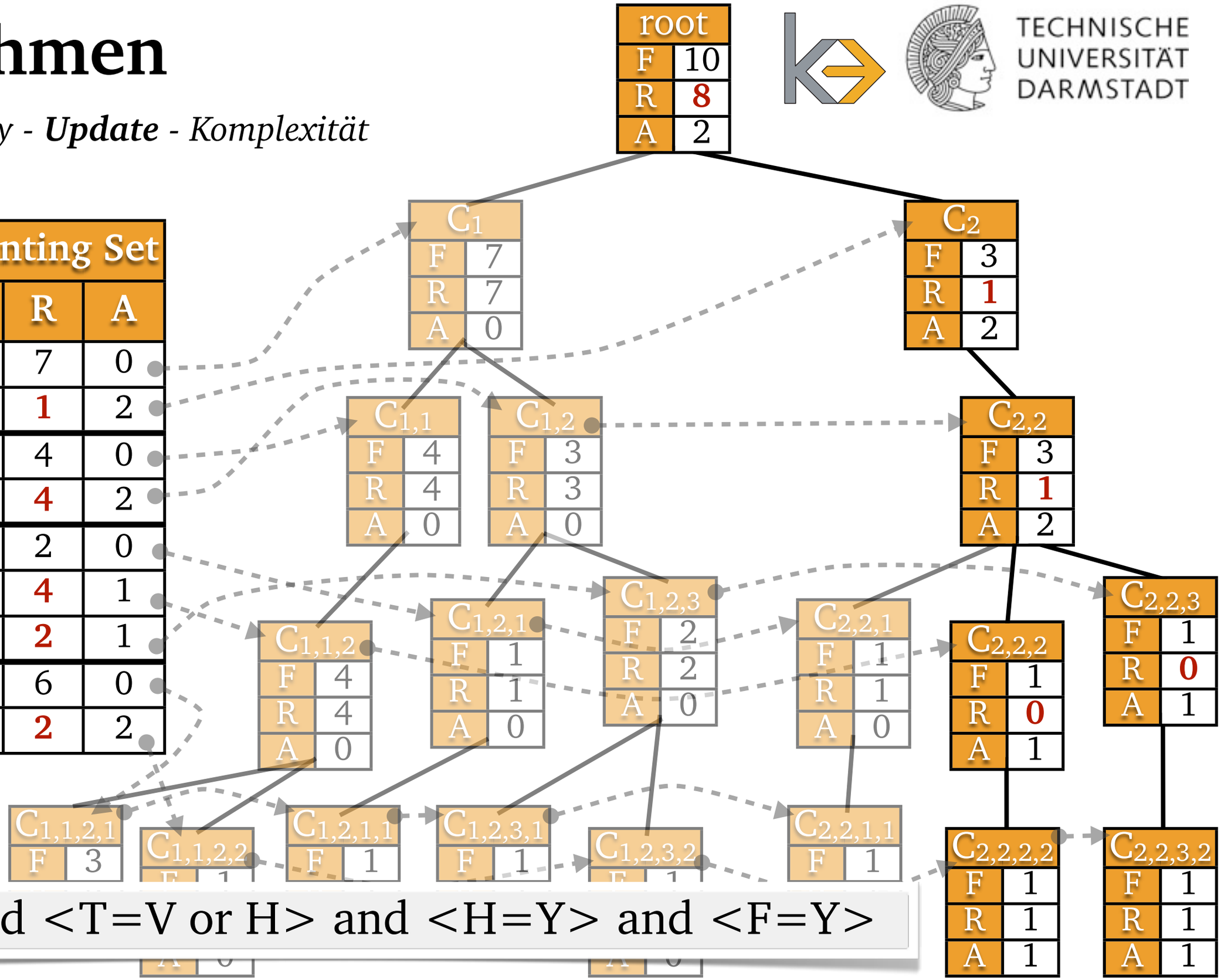


AVP	Counting Set		
	F	R	A
<H=N>	7	7	0
<H=Y>	3	3	2
<P=N>	4	4	0
<P=Y>	6	6	2
<T=N>	2	2	0
<T=H>	5	5	1
<T=V>	3	3	1
<F=N>	6	6	0
<F=Y>	4	4	2

Algorithmen

*Create - Query - **Update** - Komplexität*

AVP	Counting Set		
	F	R	A
<H=N>	7	7	0
<H=Y>	3	1	2
<P=N>	4	4	0
<P=Y>	6	4	2
<T=N>	2	2	0
<T=H>	5	4	1
<T=V>	3	2	1
<F=N>	6	6	0
<F=Y>	4	2	2

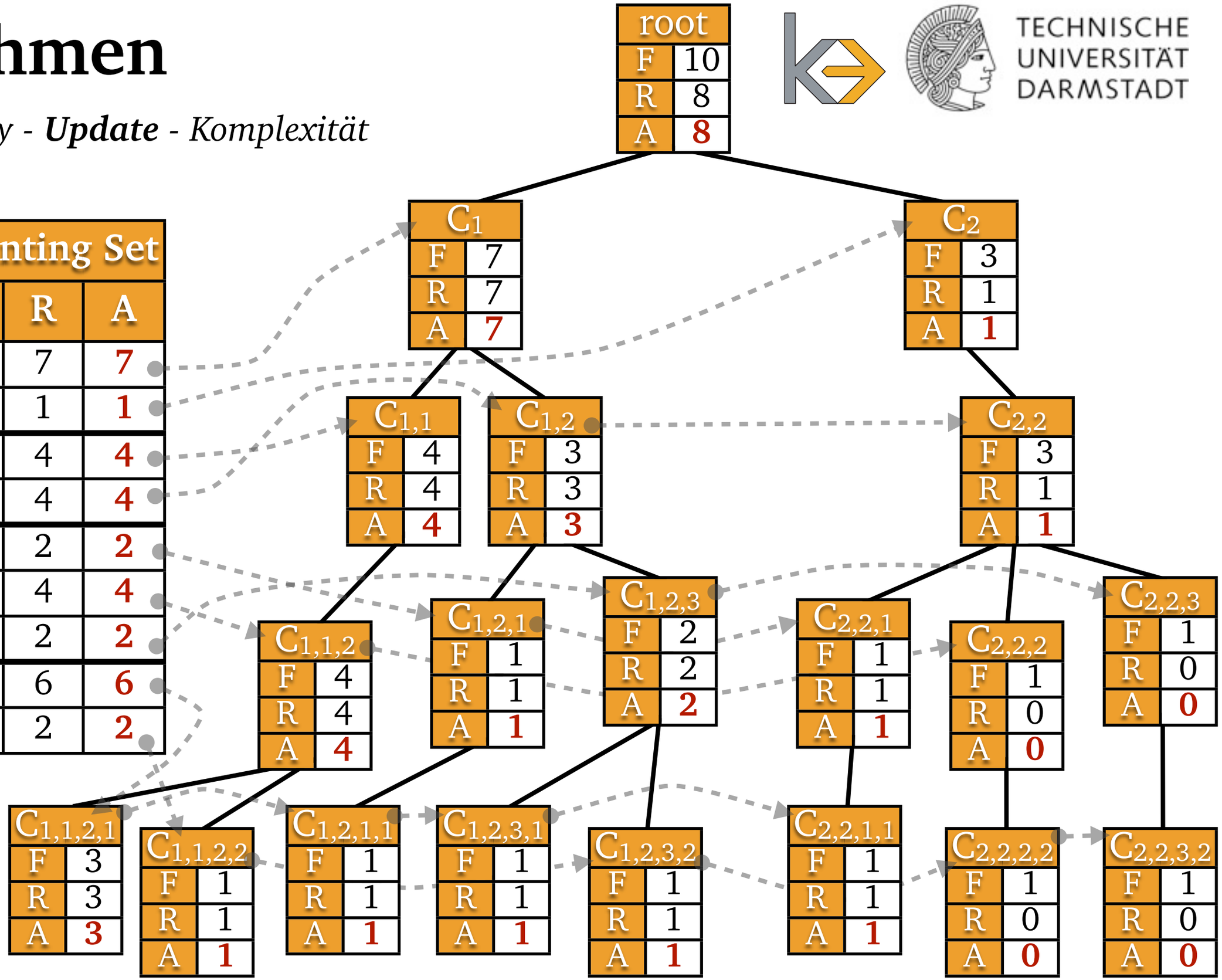


<P=Y> and <T=V or H> and <H=Y> and <F=Y>

Algorithmen

*Create - Query - **Update** - Komplexität*

AVP	Counting Set		
	F	R	A
<H=N>	7	7	7
<H=Y>	3	1	1
<P=N>	4	4	4
<P=Y>	6	4	4
<T=N>	2	2	2
<T=H>	5	4	4
<T=V>	3	2	2
<F=N>	6	6	6
<F=Y>	4	2	2



Algorithmen

*Create - Query - Update - **Komplexität***

Method	Best	Worst
AV-Space erstellen	$O(MN)$	
Abfrage mit allen Attributen	$O(k^M)$	
Update, nachdem Regelmenge fertig	$O(k^M)$	
Update, nach erweitern der Kandidatenregel	$O(M)$	$O(k^M)$
Update, nach hinzufüge der Kandidatenregel zur Regelmenge	$O(M)$	$O(k^M)$

M Attribute
N Beispielen
k eindeutigen Werten pro Attribut

Auswertung

Datensätze - AV-space vs Example-set - AV-space vs ADtree - Speicherverbrauch

Auswertung

Datensätze *- AV-space vs Example-set - AV-space vs ADtree - Speicherverbrauch*

- Aus UCI Repository
- Kontinuierliche Werte wurden diskretisiert

Data set	# of examples	# of attributes	# of classes	# of nodes in AV-space	# of attribute values
Flu	10	4	2	20	8
Lenses	24	5	3	63	24
Monks	122	7	2	379	122
Flag	194	27	8	3.024	193
Glass	214	10	6	702	129
Balance Scale	625	5	3	1.406	625
Credit Scale	653	16	2	3.618	604
Breast Cancer	683	10	2	3.099	449
Mushroom	5.644	23	2	17.304	5.644
Nursey	12.598	9	5	29.403	12.960
Adult	45.222	15	2	111.259	24.206

Auswertung

*Datensätze - **AV-space vs Example-set** - AV-space vs ADtree - Speicherverbrauch*

- Vergleich: AV-space mit Example-set (dynamisches Array)

Data Set	Time Use	AV-space	Example-set	Improvement	# examples
Flu	Loading Rule induction	0,140 0,031	0,172 0,031	18,6 % 0,0 %	10
Lenses	Loading Rule induction	0,141 0,031	0,141 0,032	0,0 % 3,1 %	24
Monks	Loading Rule induction	0,204 0,078	0,219 0,109	6,8 % 28,4 %	122
Flag	Loading Rule induction	0,172 0,360	0,203 0,484	15,3 % 25,6 %	194
Glass	Loading Rule induction	0,016 0,219	0,172 0,406	90,7 % 46,1 %	214
Balance Scale	Loading Rule induction	0,031 0,516	0,031 1,484	0,0 % 65,2 %	625
Credit Screening	Loading Rule induction	0,187 0,516	0,203 1,156	7,9 % 55,4 %	653
Breast Cancer	Loading Rule induction	0,172 0,156	0,187 0,422	8,0 % 63,0 %	683
Mushroom	Loading Rule induction	0,438 0,297	1,125 2,859	61,1 % 89,6 %	5644
Nursey	Loading Rule induction	0,328 10,110	0,594 67,469	44,8 % 85,0 %	12598
Adult	Loading Rule induction	1,313 76,110	3,641 219,109	63,9 % 65,3 %	45222

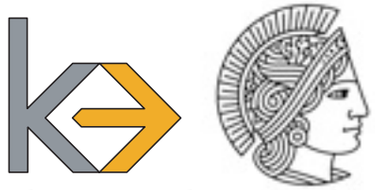
Auswertung

*Datensätze - **AV-space vs Example-set** - AV-space vs ADtree - Speicherverbrauch*

- Vergleich: AV-space mit Example-set (dynamisches Array)

Data Set	AV-space (bytes)	Example-set (bytes)	AV-space/ Example-set	# of examples
Flu	484	384	1,26	10
Lenses	1.308	908	1,44	24
Monks	7.108	5.104	1,39	122
Flag	56.428	24.788	2,28	194
Glass	13.576	11.844	1,15	214
Balance Scale	25.656	20.272	1,27	625
Credit Screening	66.352	50.560	1,31	653
Breast Cancer	57.236	36.616	1,56	683
Mushroom	313.260	588.328	0,53	5.644
Nursey	529.748	622.460	0,85	12.598
Adult	2.004.836	3.257.624	0,62	45.222

Auswertung



TECHNISCHE UNIVERSITÄT DARMSTADT

*Datensätze - **AV-space vs Example-set** - AV-space vs ADtree - Speicherverbrauch*

- Covtype Datensatz, US Forest Service
- 40 Attribute, sehr dünn besetzt

Numbers of examples	Algorithm	Memory (bytes)	Loading time (sec)	Rule induction time (sec)
581.011	AV-space	19.588	12,469	0,109
	Example-set	102.258.976	330,281	187,891
300.000	AV-space	19.500	6,390	0,110
	Example-set	52.801.040	168,735	81,843
50.000	AV-space	18.276	1,156	0,109
	Example-set	8.801.040	26,281	14,750
10.000	AV-space	17.880	0,406	0,141
	Example-set	1.761.040	4,688	3,032
2.000	AV-space	13.468	0,203	0,109
	Example-set	353.040	1,203	0,484

Auswertung

- Vergleich: AV-space mit ADtree
- Werte für ADtree sind einem anderem Paper entnommen

Data set	# of Attributes	# of Examples	AV-space		ADtree	
			Memory cost (M)	Build Time (sec)	Memory cost (M)	Build Time (sec)
Adult 1	15	15.060	0,9	0,515	7,0	6,0
Adult 2	15	30.162	1,5	0,937	10,9	10,0
Adult 3	15	45.222	2,0	1,313	15,5	15,0

Auswertung

Datensätze - AV-space vs Example-set - AV-space vs ADtree - ***Speicherverbrauch***

- Speicherverbrauch des AV-tree
  - Nicht abhängig von der # der Trainingsdaten
  - Abhängig von der # der Attribute und eindeutige Werte
  - Attribute mit wenig Werten besser dicht bei der Wurzel

Worst case Speicherverbrauch:

$$a * \sum_{i=1}^{M} k^i = \frac{a}{k-1} k^{M+1} - \frac{a}{k-1}$$

a Speicherplatz pro Knoten
M Attribute
N Beispielen
k eindeutigen Werten pro Attribut

Fazit

Fazit

- Datenstruktur sieht vielversprechend aus
- Algorithmus braucht die Anzahl der eindeutigen Werte
- Vergleich mit ADtree nicht Aussagekräftig
- Implementierung mit anderen Algorithmen wäre interessant

Vielen Dank

Anhang

Algorithmen

Create *- Query - Update - Komplexität*

```
Input: data set D,
1. create AVC-Group with sum(k_i) entries
2. create root node with k_1 childs
3. for each example e in D do
4.     set counts of root += 1
5.     node = root
6.     for each <a_i = v_ij> in e (from a_i to a_M)
7.         currentNode = node
8.         node = addNode(a_i, v_ij, k_i, currentNode)
```

Algorithmen

Create *- Query - Update - Komplexität*

```
Input: a_i, v_ij, k_i, currentNode
1. increment counts of AVC-Group entry <a_i=v_ij> by 1
2. if jth child node of currentNode is NULL
3.    then
4.       create a new Node, newNode
5.       set counts of newNode to 1
6.       set jth child node pointer of currentNode to
            newNode
7.       link newNode to the list headed by <a_i=v_ij> in
            the AVC-Group
8.    else
9.       increment counts of jth child node of
               currentNode by 1
10.return the jth child node of currentNode
```