

1. Einführung	4. Physische Datenorganisation	7. Datensicherheit und Wiederherstellung
2. Datenbankentwurf	5. Anfrageoptimierung	8. Business Intelligence
3. Datenbankimplementierung	6. Transaktionsverwaltung	

D a t e n b a n k e n

Gliederung

- 1.**
- 2.**
- 3.**
- 4.**
- 5.**
- 6.**
- 7.**
- 8.**

Einführung

Datenbankentwurf

Datenbankimplementierung

Physische Datenorganisation

Anfrageoptimierung

Transaktionsverwaltung

Datensicherheit und Wiederherstellung

Business Intelligence

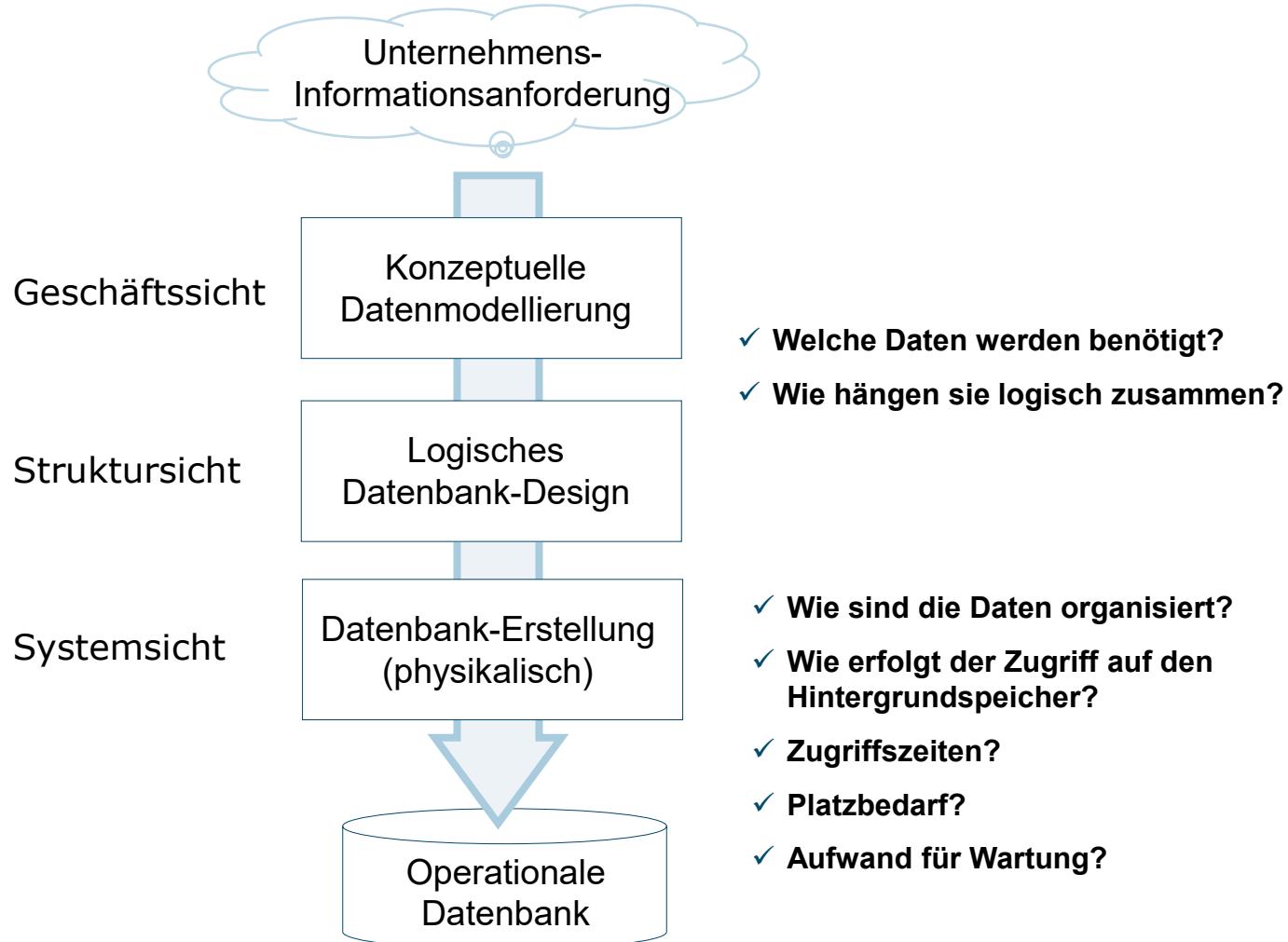
Gliederung

4.

Physische Datenorganisation

- ▶ Aufbau eines DBMS
- ▶ Deklarative vs. Prozedurale Programmiersprache
- ▶ Ablauf einer Anfrageverarbeitung
- ▶ Aufbau und Organisation von Speichermedien
- ▶ Aufbau von Dateien und Zugriff auf die Daten

Schrittweise Erstellung einer Datenbank



Nutzung von Speicherhierarchien

Wunschvorstellung: Der Ideale Speicher...

... hält die Daten, die demnächst benötigt werden, schon „**nahe dem Prozessor**“ bereit, so dass der Prozessor sie ohne Zeitverzug verarbeiten kann. Der ideale Speicher besitzt:

- ✓ nahezu unbegrenzte Speicherkapazitäten
- ✓ geringe Speicherkosten
- ✓ kurze Zugriffszeit bei wahlfreiem Zugriff
- ✓ hohe Zugriffsraten
- ✓ Nichtflüchtigkeit
- ✓ Fähigkeit zu logischen, arithmetischen u. ä. Verknüpfungen



Realität: Einsatz einer Speicherhierarchie ...

Durch den Einsatz einer mehrstufiger **Hierarchie verschiedener Zwischenspeicher**, wird versucht eine Annäherung an den idealen Speicher zu erreichen.

Zur Realisierung einer Speicherhierarchie sind auf jeder Ebene ähnliche **Verwaltungsaufgaben** zu lösen:

- Lokalisieren des gesuchten Datenobjekts,
- Allokation von Speicherplatz für ein neues Objekt,
- Ersetzung des Objektes mit der geringsten Referenzwahrscheinlichkeit,
- Implementierung von Schreib- und Lese – Strategien,
- ggf. Anpassung an verschiedene Transfergranulate

Nutzung von Speicherhierarchien

Rekursives Prinzip

- Kleinere, schnellere und teurere (Cache-) Speicher werden benutzt, um Daten für die Verarbeitung zwischenzuspeichern, die sich in größeren, langsameren und günstigeren Speichern befinden.
Prinzip: „Je kleiner desto schneller und je größer desto langsamer“

Preis-Leistungs-Prinzip

- Schneller Speicher ist teuer und deshalb klein
- Speicher hoher Kapazität ist typischerweise langsamer

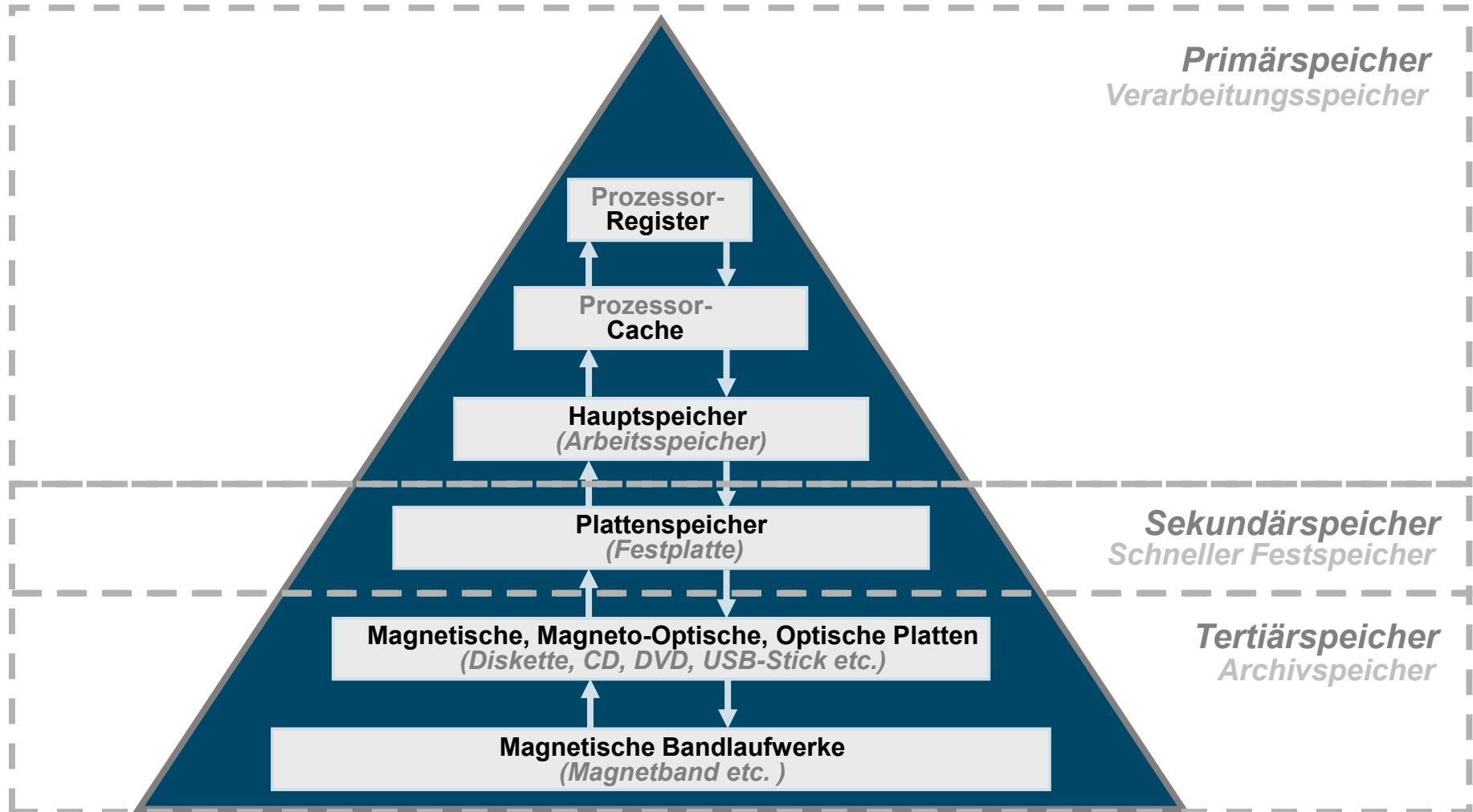
Lokalitätsprinzip

- Zeitliche Lokalität: Pufferung kürzlich referenzierter Objekte
- Räumliche Lokalität: Pufferung benachbart gespeicherter Objekte

► Referenzlokalität und optimierte Ersetzungsverfahren erzielen typischerweise eine hohe **Trefferrate** (Anzahl der Treffer im Speicher / Anzahl aller Speicherreferenzen). Die Trefferrate ist der Indikator, wie gut das Ziel einer Speicherhierarchie verwirklicht werden konnte.

*Beim Cache sollte die Trefferrate je nach Anwendung 96 – 99 % und mehr betragen.
Verbesserung der Trefferrate: Größere Caches durch Clusterbildung für referenzierte Daten (räumliche Lokalität)*

Prinzipieller Aufbau einer Speicherhierarchie



Speicherhierarchie bezeichnet die Anordnung von Arbeits- und Datenspeichern verschiedener Technologien (Speichertypen) nach **sinkender Zugriffsgeschwindigkeit** und **steigender Speicherkapazität** in einer Rechnerarchitektur.

Maßeinheit für Speicherkapazität

- Innerhalb eines Computers lassen sich Daten nur digital (digit = Zahl) verschlüsseln und verarbeiten (digitale Datenverarbeitung). Die kleinste Informationseinheit in der Datenverarbeitung wird als Bit (binary digit, Binärziffer) bezeichnet. Diese Einheit kann die beiden Ausprägungen STROM und KEIN STROM (1 bzw. 0) aufweisen. Dies sind die einzigen Zustände, die ein Computer unterscheiden kann.
- Sobald aber Daten verarbeitet werden sollen, die mehr als nur zwei Ausprägungen aufweisen, müssen mehrere Bits zu einer Einheit zusammengefasst werden. Eine Einheit von 8 Bits ist in der Lage 256 (2^8) Zeichen darzustellen und wird als Byte bezeichnet. Diese Einheit stellt die Maßeinheit für die Speicherkapazität (wie viele Zeichen der Speicher gleichzeitig aufnehmen kann) einer Festplatte dar.

SI - Präfixe

Name	Yotta	Zetta	Exa	Peta	Tera	Giga	Mega	Kilo	Hekto	Deka
Symbol	Y	Z	E	P	T	G	M	k	h	da
Faktor	10^{24}	10^{21}	10^{18}	10^{15}	10^{12}	10^9	10^6	10^3	10^2	10^1
Name	Yokto	Zepto	Atto	Femto	Piko	Nano	Mikro	Milli	Zenti	Dezi
Symbol	y	z	a	f	p	n	μ	m	c	d
Faktor	10^{-24}	10^{-21}	10^{-18}	10^{-15}	10^{-12}	10^{-9}	10^{-6}	10^{-3}	10^{-2}	10^{-1}

Dezimalpräfixe		Unterschied	Binärpräfixe	
Name (Symbol)	Bedeutung analog zu SI-Präfixen ^[G 1]		Name (Symbol)	Tradierte Bedeutungsauffassung
Kilobyte (kB) ^[G 2]	10^3 Byte = 1.000 Byte		Kibibyte (KiB) ^[G 3]	2^{10} Byte = 1.024 Byte
Megabyte (MB)	10^6 Byte = 1.000.000 Byte		Mebibyte (MiB)	2^{20} Byte = 1.048.576 Byte
Gigabyte (GB)	10^9 Byte = 1.000.000.000 Byte		Gibibyte (GiB)	2^{30} Byte = 1.073.741.824 Byte
Terabyte (TB)	10^{12} Byte = 1.000.000.000.000 Byte		Tebibyte (TiB)	2^{40} Byte = 1.099.511.627.776 Byte
Petabyte (PB)	10^{15} Byte = 1.000.000.000.000.000 Byte		Pebibyte (PiB)	2^{50} Byte = 1.125.899.906.842.624 Byte
Exabyte (EB)	10^{18} Byte = 1.000.000.000.000.000.000 Byte		Exbibyte (EiB)	2^{60} Byte = 1.152.921.504.606.846.976 Byte
Zettabyte (ZB)	10^{21} Byte = 1.000.000.000.000.000.000.000 Byte		Zebibyte (ZiB)	2^{70} Byte = 1.180.591.620.717.411.303.424 Byte
Yottabyte (YB)	10^{24} Byte = 1.000.000.000.000.000.000.000.000 Byte	20,9 %	Yobibyte (YiB)	2^{80} Byte = 1.208.925.819.614.629.174.706.176 Byte

1. ↑ SI-Präfixe sind nur für SI-Einheiten standardisiert; Byte ist keine SI-Einheit.
 2. ↑ wird gelegentlich inkorrekt mit „KB“ abgekürzt.
 3. ↑ wird häufig inkorrekt mit „KB“ abgekürzt.

Charakteristische Merkmale einer Speicherhierarchie



Speicher	Geschwindigkeit	Preis	Stabilität	Größe	Granulate
Primär	schnell	teuer	flüchtig (elektronisch)	klein	fein (byte-adressierbar = instruktions-adressierbar → direkte Datenverarbeitung)
Sekundär	langsam	preiswert	stabil (magnetisch, sicher vor Systemausfällen)	groß	grob (block-adressierbar)
Tertiär	sehr langsam	günstig	stabil (magneto-optisch o. magnetisch, sicher vor Systemausfällen)	sehr groß	grob (block-adressierbar o. file-adressierbar)

Aufbau und Organisation einer Festplatte

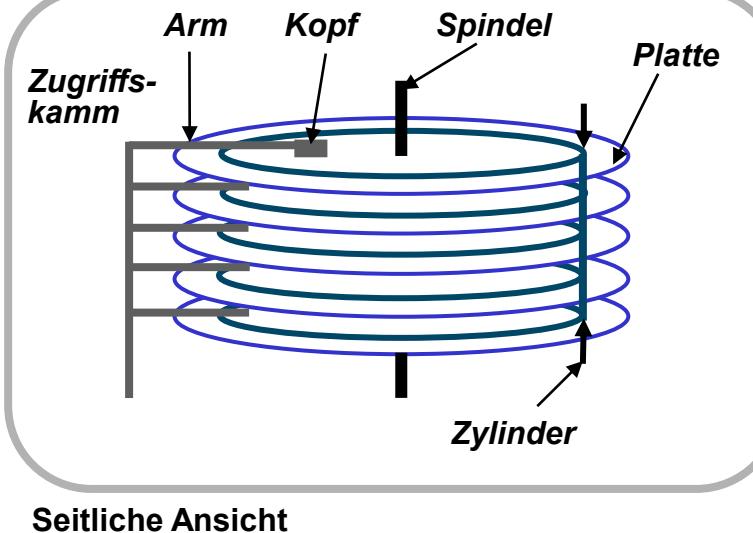
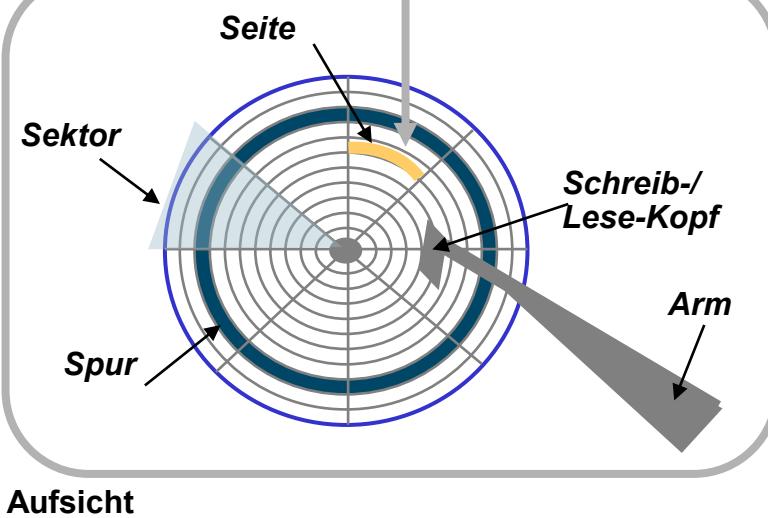
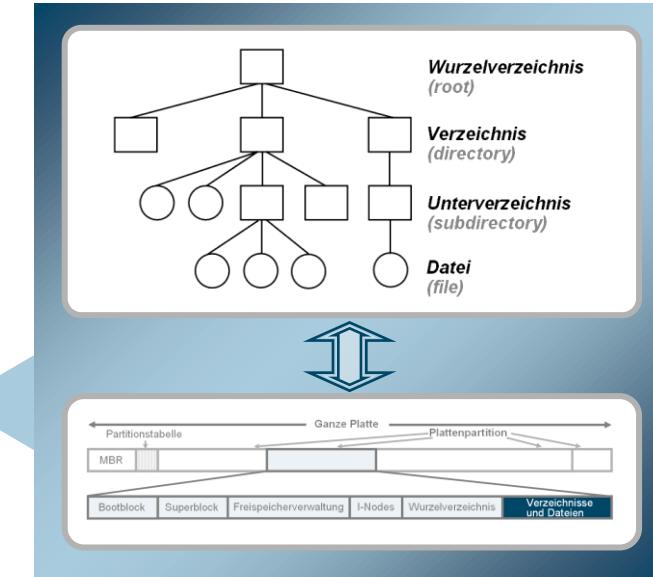
abstrakt

logisch

physisch

Programm

Laufwerksname:\Verzeichnis\Unterverzeichnis\Datei
C:\Documents\Letters\Test.doc

Betriebssystem
Dateisystem

Aufbau und Organisation einer Festplatte

► Durch die physikalische Formatierung werden die Platten in Spuren, Sektoren und Zylinder unterteilt. Über die Angabe dieser drei Werte lässt sich jede Stelle auf der Festplatte eindeutig adressieren. Die physikalische Formatierung wird vom Hersteller durchgeführt.

Spuren: Jede Scheibe einer Festplatte ist in konzentrische Kreise unterteilt. Ein einzelner Kreis wird Spur genannt. Spur 0 ist der äußerste Kreis einer Scheibe. Eine Scheibe enthält typischerweise einige tausend solcher Spuren, meist auf beiden Seiten.

Zylinder: Die Gesamtheit aller gleichen, d. h. übereinander befindlichen, Spuren der einzelnen Plattenoberflächen nennt man Zylinder.

Sektoren: Jede Spur einer Platte ist wiederum in einzelne Abschnitte fester Größe, die Sektoren, aufgeteilt. Jeder Sektor ist in der Lage eine Datenmenge von 512 Byte aufzunehmen. Ein Sektor ist die kleinste physikalische adressierbare Einheit einer Platte.

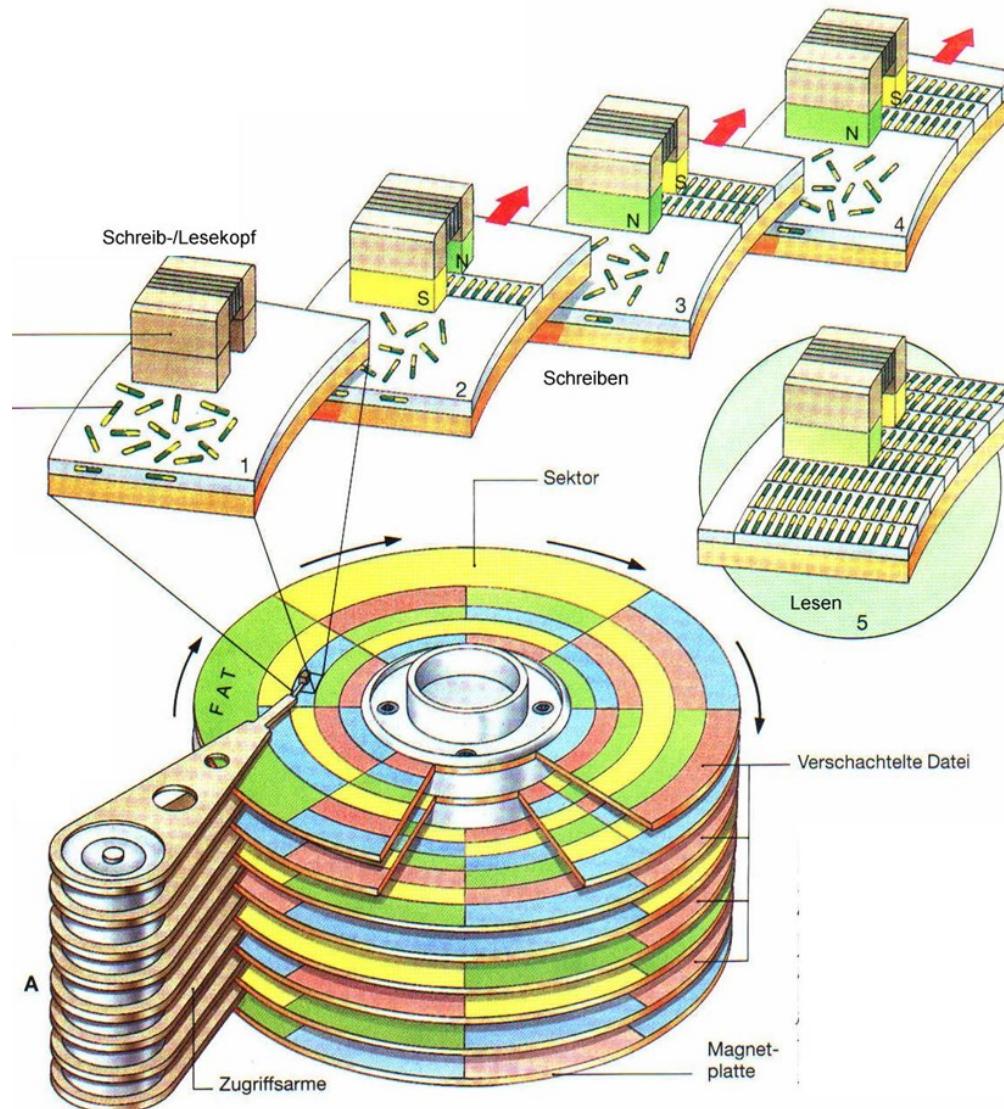
Seiten: Zur Optimierung von Plattenzugriffen und zum Ausgleich der hohen Differenzen in der Zugriffsgeschwindigkeit werden Daten in Einheiten von Seiten (zusammenhängende Folgen von Sektoren einer Spur) zwischen dem Haupt- und dem Sekundärspeicher hin und her transportiert.

Alle E/A-Operationen werden damit auf Seitenebene ausgeführt und dies bedeutet, dass immer ganze Datenseiten gelesen oder geschrieben werden.

Der Speicherplatz der einer Datendatei zugeordnet wird, ist daher logisch in Seiten unterteilt. Eine Seite ist die grundlegende logische Einheit für die Datenspeicherung und wird mittels Zylindernummer, Spurnummer und Sektornummer adressiert.

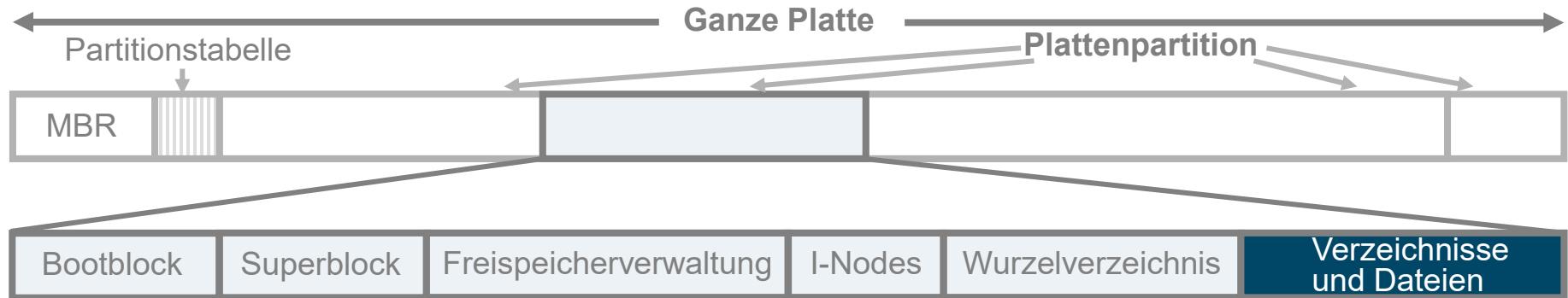
Blöcke: Blöcke sind Auflistungen von mehreren zusammenhängenden Seiten; sie werden zum effizienten Verwalten der Seiten verwendet. Alle Seiten werden in Blöcken gespeichert.

Lesen und Schreiben von Daten



- Mit Hilfe eines Spindelmotors rotieren während der gesamten Einschaltzeit des Rechners synchron mehrere beidseitig mit hochfein polierten Metallociden beschichtete Metallplatten.
- Das Schreiben bzw. Lesen von Daten erfolgt durch das Erzeugen bzw. Abtasten von konzentrischen Kreisen von Magnetisierungsmustern auf den Spuren der Platten durch die Schreib-/Lese-Köpfe (Magnetköpfe).
- Damit werden Daten auf Platten als eine Serie von Bits aufgezeichnet. Die einzelnen Bits verhalten sich hierbei wie kleine Magnete mit Süd- und Nordpol.
- Beim Schreiben von Daten werden die Magnete gegen bzw. mit der Drehrichtung der Platte ausgerichtet:
 - 0: Im Bit gegen die Drehrichtung magnetisiert**
 - 1: Im Bit mit der Drehrichtung magnetisiert**
- Auf diese Weise wird ein Bit als eine magnetische Ladung (positiv oder negativ) auf der Festplattenscheibe gespeichert.
- Beim Lesen erkennt der Schreib-/Lese-Kopf die Ausrichtung der Bits (Polarität).
- Um diese Daten gezielt abzulegen und wiederzufinden muss die Festplatte vorher formatiert (physikalisch und logisch) werden.

Ein mögliches logisches Layout eines Dateisystems



- ▶ Um Daten persistent (dauerhaft) verwalten (speichern, bearbeiten, abrufen) zu können, müssen diese in speziellen Einheiten, den Dateien (files), abgelegt werden. Datei ist hierbei ein abstrakter Mechanismus zur Speicherung und zum Finden von Informationen. Jedes Betriebssystem stellt zum Zwecke der Dateiverwaltung (logische Speicher-verwaltung) ein Dateisystem (filesystem) bereit.
- ▶ In einem Dateisystem sind Daten eines Computers in Form von Verzeichnissen (directories) und Dateien hierarchisch organisiert. Gruppen von Dateien können in Verzeichnissen zusammengefasst werden. Jedes Verzeichnis kann beliebig viele Unterverzeichnisse (subdirectories) oder als Endknoten die Dateien selbst enthalten. Durch die hierarchische Anordnung entsteht (umgekehrt) ein Baum.
- ▶ Das Dateisystem selbst ist ein Ordnungs- und Zugriffssystem für diese Daten. Die Zugriffs Routinen auf das Dateisystem sind Bestandteile des Betriebssystems. Unter einem Dateisystem versteht man also sowohl die Struktur, wie die Daten auf einem Medium abgelegt werden, als auch die Vorschrift, wie dies zu geschehen hat.
- ▶ Eine Festplatte muss vor ihrer ersten Benutzung vom Betriebssystem mit einem Dateisystem formatiert werden. Die entsprechende Partition / das Laufwerk erzeugt hierbei eine spezielle Datei, die virtuelle Dateizuordnungstabelle, in welcher das Betriebssystem die Informationen über die Verzeichnisstruktur dieser Partition speichert. In der Tabelle werden alle physikalischen Zuordnungseinheiten (Blöcke mit Seiten), in welchen sich die Dateien auf dem Speichermedium befinden, verwaltet.
- ▶ Erst im Dateisystem werden abstrakte Angaben wie Dateiname und Dateiattribute in physikalische Adressen auf dem Speichermedium umgesetzt!

Ein mögliches logisches Layout eines Dateisystems

Partitionierung

- Dateisysteme werden auf Platten gespeichert. Platten können in eine oder mehrere Partitionen, mit unabhängigen Dateisystemen auf jeder einzelnen Partition, unterteilt werden.

MBR und Partitionstabelle

- Der Sektor 0 auf der Platte wird MBR (Master Boot Record) genannt und findet beim Booten des Computers Verwendung.
- Das Ende des MBR enthält die Partitionstabelle. Diese Tabelle enthält die Anfangs- und Endadresse jeder Partition. Eine der Partitionen in der Tabelle ist als aktiv markiert.
- Wenn der Computer bootet, so liest das BIOS den MBR ein und führt ihn aus. Als Erstes lokализiert das MBR-Programm die aktive Partition und liest deren ersten Block, den Bootblock, ein und führt ihn aus.

Bootblock

- Das Programm im Bootblock lädt das Dateisystem, welches in der Partition gespeichert ist.

Superblock

- Dieser enthält all die Schlüsselparameter des Dateisystems und wird als Erstes in den Speicher geladen, wenn der Computer gestartet oder auf das Dateisystem zum ersten Mal zugegriffen wird.
u. a. IdentifikationsNr. des Dateisystems, Anzahl der Blöcke im Dateisystem

Freispeicherverwaltung

- Informationen über die freien Blöcke im Dateisystem, zum Beispiel in Form einer Bitmap oder Liste von Zeigern.

I-Nodes

- Hierbei handelt es sich um einen Array von Datenstrukturen. Eine Datenstruktur enthält alle Informationen pro Datei.

Wurzelverzeichnis

- Stellt die Spitze des Dateibaums dar.

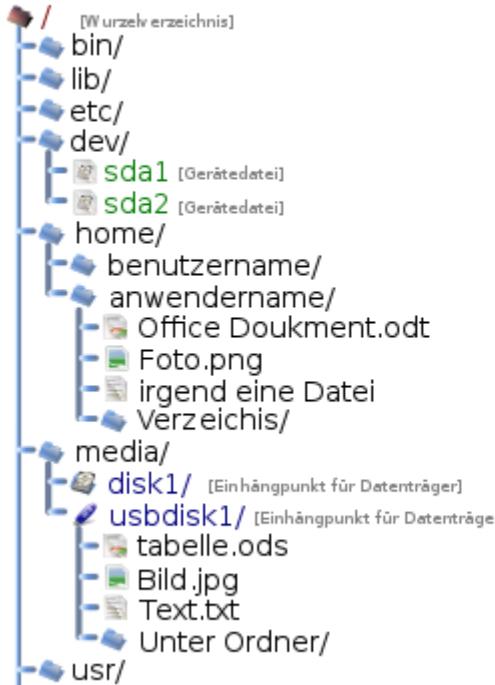
Verzeichnisse und Dateien

- Der Rest der Platte enthält schließlich die eigentlichen Verzeichnisse und Dateien.

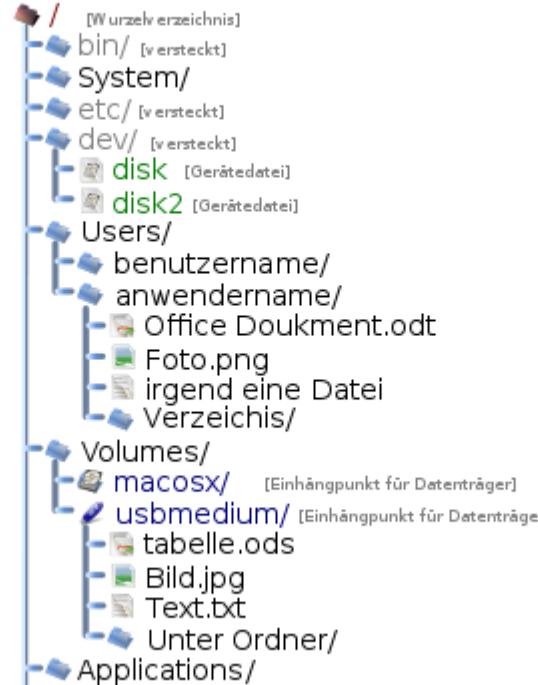
Mögliche abstrakte Layouts eines Dateisystems



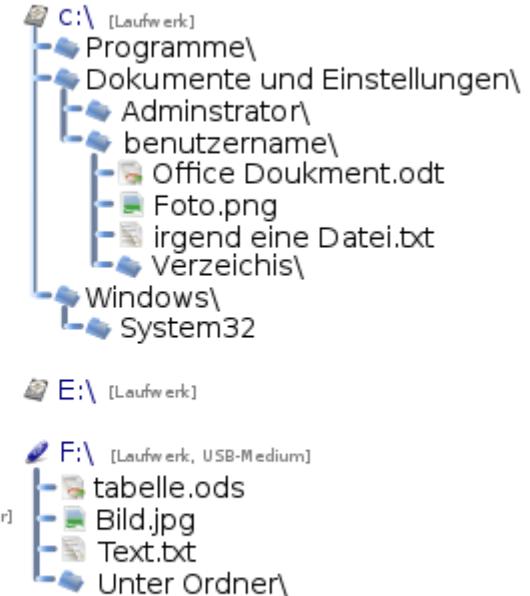
UNIX, Linux, ZETA



Mac OSX



Windows NT/2000/XP



- Unix verfügt über ein **systemweites Dateisystem**, während in Windows-Betriebssystemen für jedes Gerät (Floppy-Laufwerk, Festplatte) ein **unabhängiges Directory** verwaltet wird.
- Unter Windows kann man auf einer Festplatte auch unterschiedliche Filesystem-Typen (FAT und NTFS) konfigurieren. FAT-Dateisysteme werden nur noch aus Kompatibilitätsgründen zu älteren Windows-Derivaten unterstützt. NTFS ist das aktuell gängige Dateisystemformat.

Aufbau von Computersystemen

Computer arbeiten nach dem EVA-Prinzip:

Eingabe (Input)

Über eine Eingabeeinheit gelangen Daten in den Computer, die

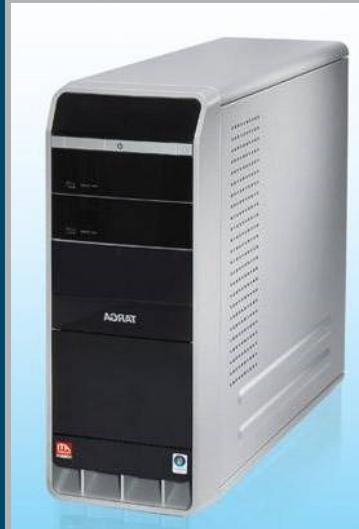
Verarbeitung

dieser Daten findet dann in der Zentraleinheit statt, bevor die

Ausgabe (Output)

über ein Ausgabegerät erfolgt.

Computersystem = Zentraleinheit (Rechner) und Peripherie



Hardware

Zentraleinheit

Prozessor (CPU)

Arbeitsspeicher

Bus-System

E/A-System

Peripherie

Eingabegeräte

Tastatur

Maus

Ausgabegeräte

Bildschirm

Drucker

Ein- / Ausgabe - geräte

Festplatte

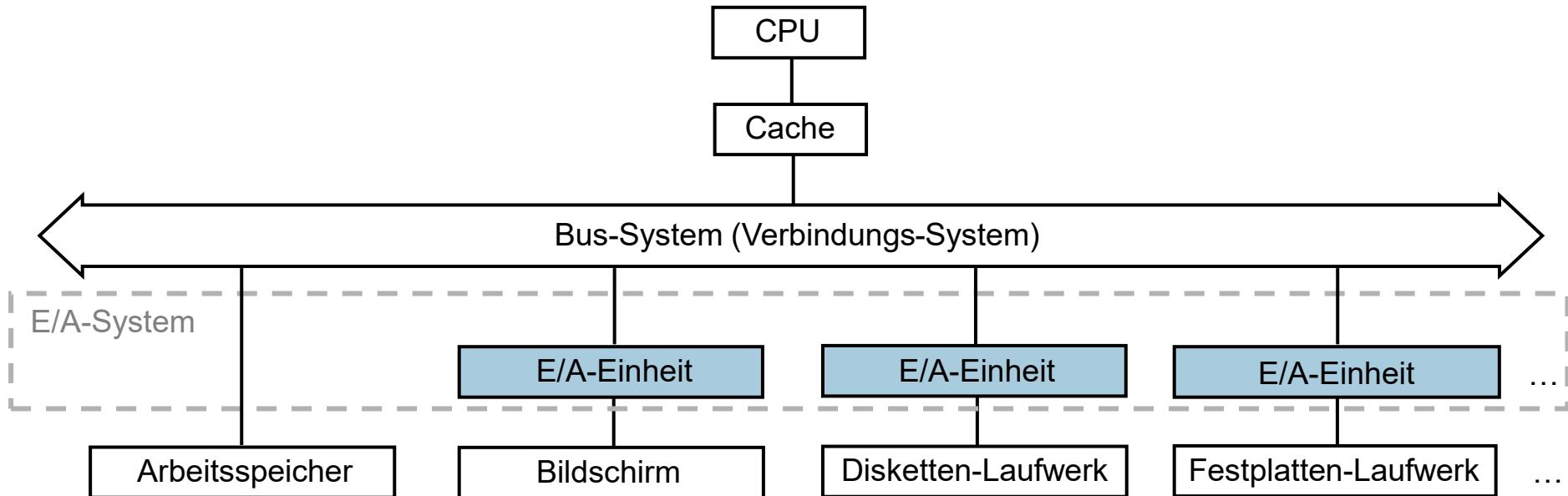
Netzwerk-
karte

Peripherie

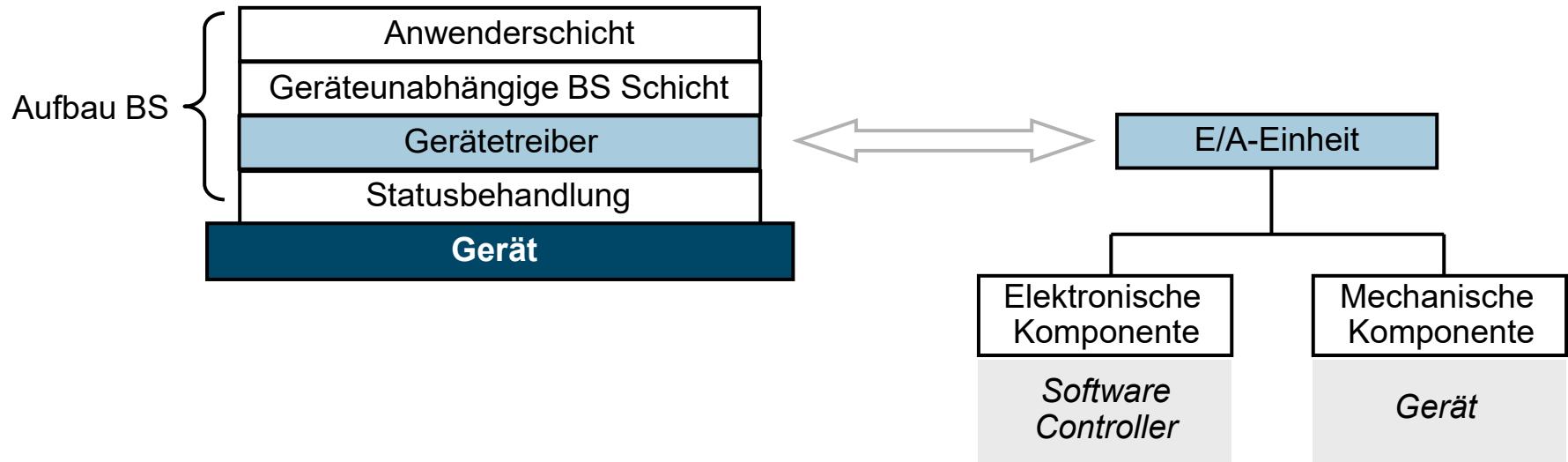
Zur Peripherie zählen alle Geräte, die an einen Rechner angeschlossen sind:

Geräte	Eingabe	Ausgabe	Massenspeicher
Tastatur, Maus, Scanner, Digitalkamera	X		
CD-ROM-Laufwerk	X		X
Grafikkarte, Bildschirm, Drucker, Lautsprecher		X	
Netzwerkkarte, Modem, Soundkarte	X	X	
Festplatte, USB-Stick, CD-Brenner	X	X	X

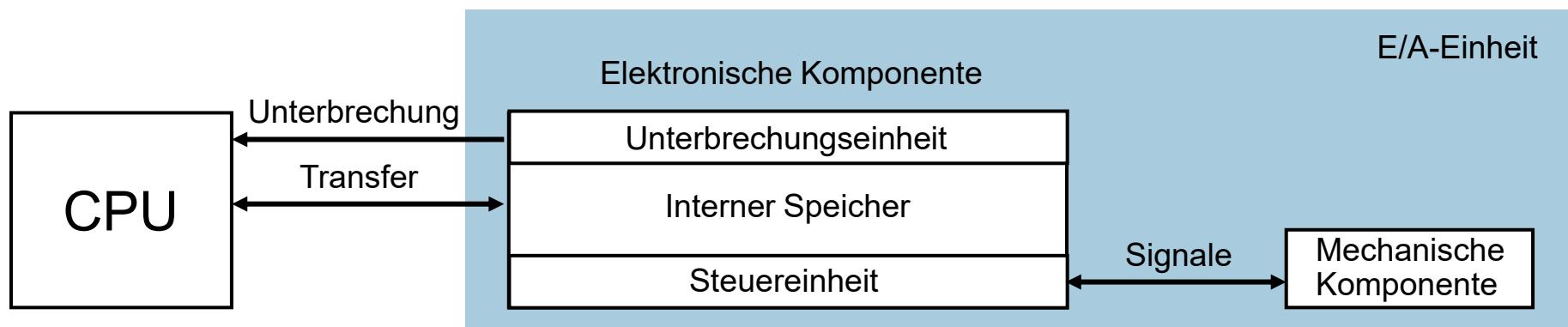
Anschluss von Peripherie an einen Rechner



E/A-Einheit: Gerätetreiber (Teil des Betriebssystems)



Gerätetreiber: Controlleraufbau



Gerätetreiber: Controlleraufbau

CPU

- Die CPU bewegt die Daten vom/zum Hauptspeicher zu/von den lokalen Buffer der Controller

Controller

- Verantwortung für einen bestimmten Typ von E/A-Gerät
- Vermittlung zwischen Prozessor und mechanischer Komponente
- Übersetzung der digitalen Information in Steuerinformation für mechanische Komponente
- Oft einheitliche Schnittstelle zwischen CPU und Gerät, Standards (PCI, USB, IDE, SATA)

Unterbrechungseinheit

- Mitteilung von Zustandsänderungen; Controller informiert CPU über das Ende einer E/A-Operation mit Hilfe einer Unterbrechungsanforderung (interrupt)

Interner Speicher

- Vergleichbar mit Register eines CPU

Steuerungseinheit

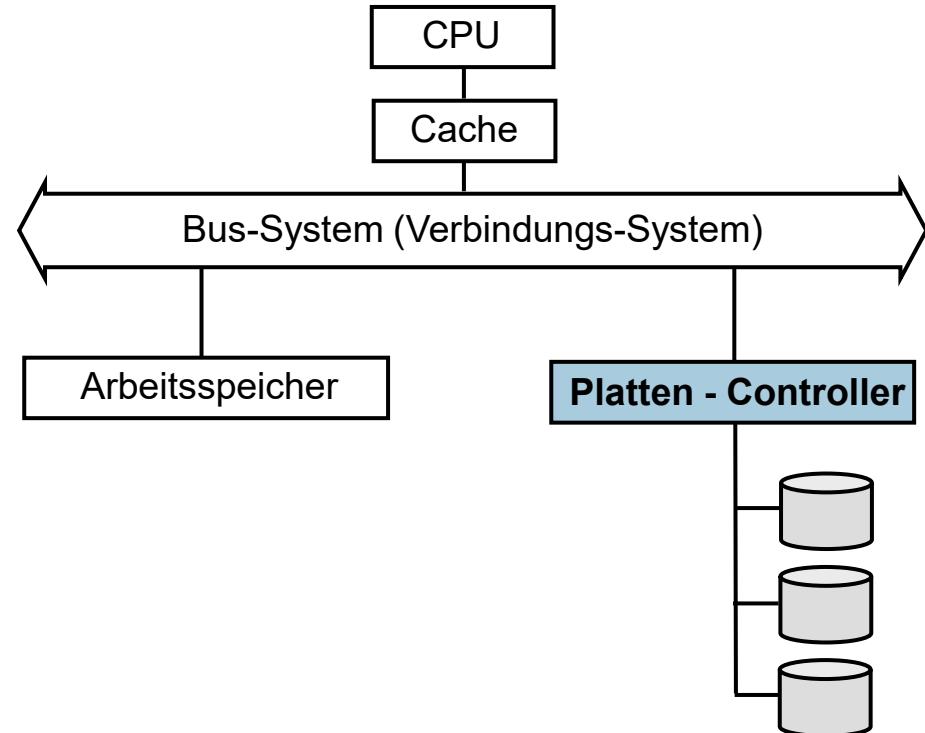
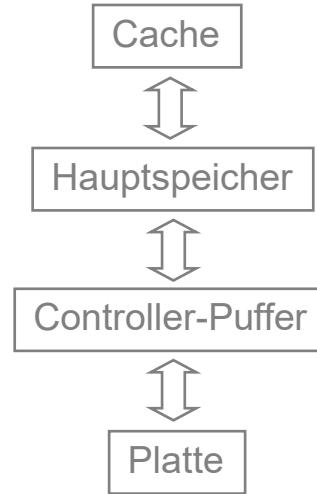
- Physikalische Ansteuerung der mechanischen Komponente

Mech. Komponente

- Verantwortlich für physikalische Umsetzung der digitalen Befehle
- Erkennen von physikalischen Fehlern

(Fest-) Platten-Controller

Datentransfer:



- ▶ Eingeführt zur Entlastung der CPU des Rechners stellt der Controller dem Gerät selbst eine eigene angemessene CPU zur Verfügung, die sich allein der Gerätesteuerung widmet; so kann die Steuerung der Datenübertragung durch die CPU direkt (blockweise o. zeichenweise) oder durch den Platten-Controller (zeichenweise) erfolgen.
- ▶ Controller ist das Bindeglied zwischen dem Gerät und dem Betriebssystem, so nimmt er Befehle bzw. Signale vom Prozessor bzw. Betriebssystem entgegen, positioniert sodann die entsprechenden Köpfe und veranlasst die Ausführung der gewünschten Operation.
- ▶ An einem Controller können mehrere Geräte (Platten) angeschlossen werden. Vorteil ist, dass das Betriebssystem nicht zwingend geändert werden muss, wenn ein anderes Gerät angeschlossen wird.

(Fest-) Platten-Controller: Zugriffsverfahren

► Die Geschwindigkeit eines Plattenzugriffs auf eine Seite wird durch drei Faktoren (mechanische Arbeitsvorgänge) bestimmt:

- | | |
|-------------------------|---|
| 1. <i>Seek Time</i> | <i>Positioniere den Arm auf den gewünschten Zylinder</i> |
| 2. <i>Latency Time</i> | <i>Warte bis die Platte so rotiert ist, dass sich der Anfang der Seite unter dem Arm befindet</i> |
| 3. <i>Transfer Time</i> | <i>Übertrage die Seite in den Hauptspeicher</i> |

$$\text{Zugriffszeit} = \text{Suchzeit} + \text{Latenzzeit} + \text{Transferzeit}$$

► Ein Zugriff ist dementsprechend am schnellsten, wenn sich der Kopf bereits in der passenden Spur befindet, und die Blöcke innerhalb der Spur **sequentiell** (nicht wahlfrei bzw. direkt) gelesen werden.

Chained I/O

- Arm einmal positionieren, danach sequentiell lesen
→ Pergamentrolle

Random I/O

- Jedes mal Arm neu positionieren
→ Buch



(Fest-) Platten-Controller: Zugriffsoptimierungen

► Zugriffsoptimierungen finden im Platten-Controller statt, so verwaltet der Controller die eingehenden Aufträge in einer Warteschlange und verarbeitet diese gemäß einer **Scheduling-Strategie**:

FCFS:

- First Come First Served

SSTF:

- Shortest Seek Time First

Scan:

- Lesen der gesamten Platte von einem Ende zum anderen und zurück

Circular Scan:

- Lesen immer nur in eine Richtung

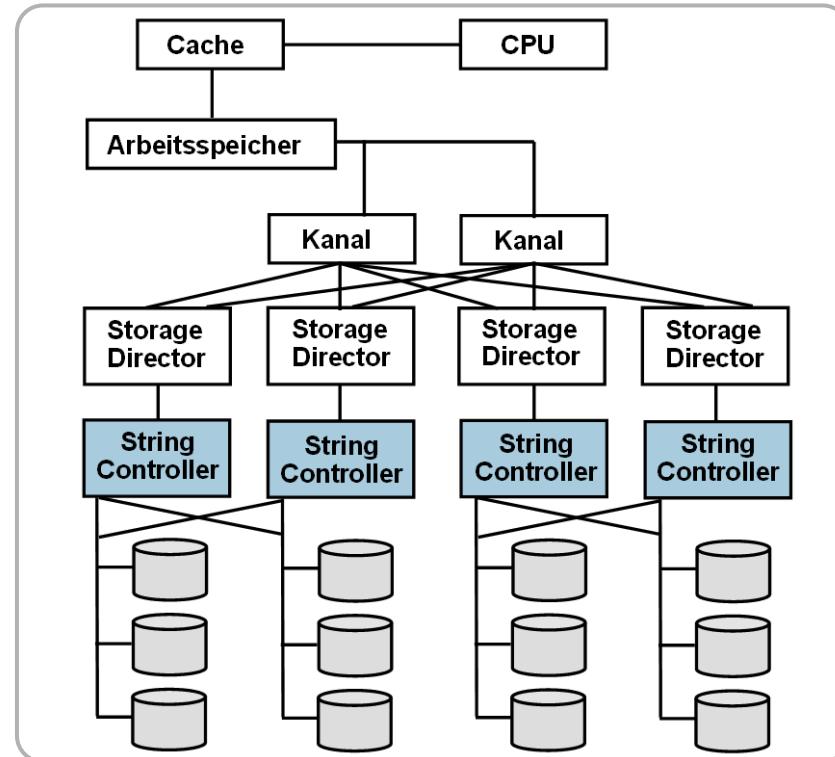
► Zusätzlich zu den rein mechanischen Arbeitsvorgängen entsteht außerdem noch ein nicht unerheblicher Programmaufwand für die Übertragung, Dekodierung und Verwaltung der von der Festplatte eingelesenen Blöcke.

Disk-Farm vs. Disk-Array

- verschiedene Plattenorganisationsformen -

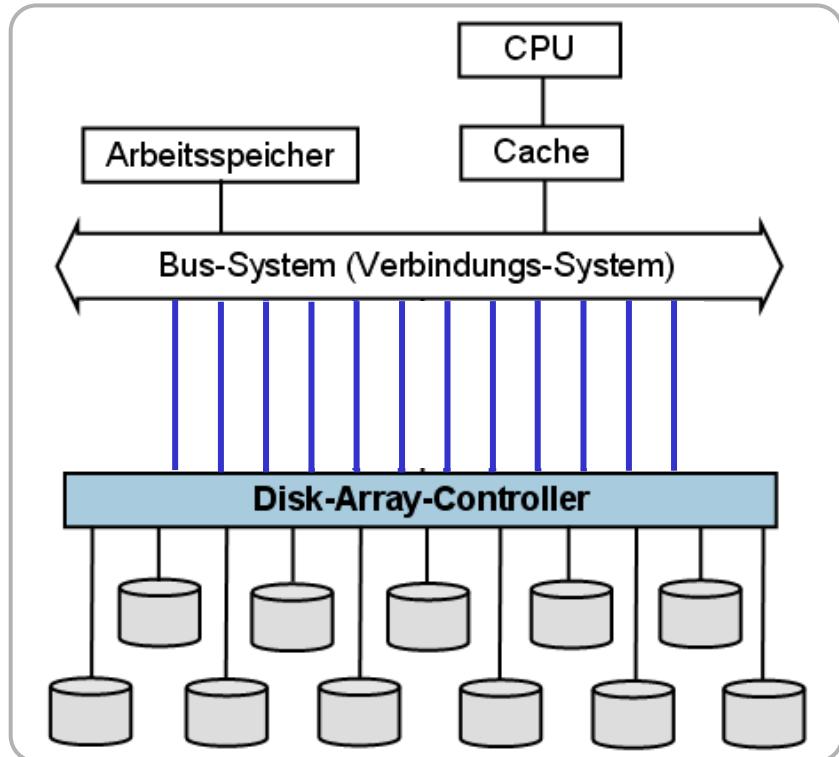
Disk-Farm

Plattenfarm
(auch „JBOD“: Just a Bunch of Disks)



Disk-Array

Plattenverbund



► Viele separate Platten unter expliziter Kontrolle des BS / DBS

► Logisch (aus BS / DBS – Sicht) wie eine einzige Platte angesprochen

Plattenverbund: RAID-Technologie

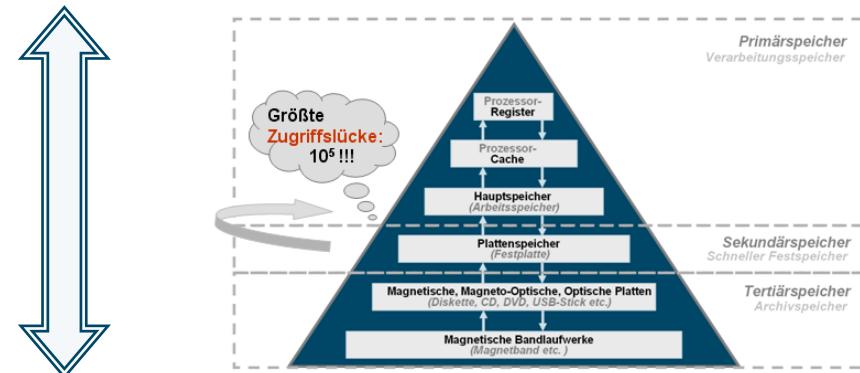
- vom konventionellen Platten-Controller zum RAID-Controller -

Terminologie

- früher: „**redundant array of inexpensive disks**“
Redundante Andordnung/Verbund kostengünstiger Festplatten
- heute: „**redundant array of independent disks**“
Redundante Anordnung/Verbund unabhängiger Festplatten

Background

Die Wartezeiten durch die mechanischen Arbeitsvorgänge in einer Festplatte sind nur schwer reduzierbar. Trotz der hohen Rotations- und Übertragungsgeschwindigkeiten moderner Laufwerke ist es nicht gelungen, die bereits erwähnte **Zugriffslücke** zwischen Haupt- und Hintergrundspeicher zu verkleinern – ganz im Gegenteil, die Lücke **wird eher größer** als kleiner.



Lösungsansatz:

Paralleles Betreiben mehrerer (kleiner) Festplatten durch einen **RAID-Controller**

(Fest-) Plattenverbund: RAID - Technologien

Technologie

Ein RAID-System dient zur Organisation / Verwaltung **mehrerer physischer Festplatten** eines Computers zu **einem logischen Laufwerk**.

Grund: Anstelle eines einzigen (entsprechend großen) Laufwerks ist es effizienter mehrere (entsprechend kleinere und günstigere) Laufwerke parallel zu betreiben.

Die preiswerten Laufwerke / Festplatten arbeiten durch einen entsprechenden **RAID-Controller** nach außen transparent (Komplexität wird nach außen verborgen) wie ein einziges logisches (virtuelles) Laufwerk mit vielen unabhängigen Schreib- / Leseköpfen. Aus Sicht des Benutzers oder eines Anwendungsprogramms unterscheidet sich ein logisches RAID-Laufwerk nicht von einer einzelnen Festplatte.

Der Betrieb eines RAID-Systems setzt mindestens zwei Festplatten voraus. Die einzelnen Organisationsformen / Anordnungen, die Arten des Zusammenwirkens der Festplatten und damit der Verteilung (Declustering) der Daten auf die einzelnen Platten, werden als **RAID-Level** spezifiziert.

Besonderheit

Während die meisten in Computern verwendeten Techniken und Anwendungen darauf abzielen **Redundanzen** (das Vorkommen doppelter Daten) zu vermeiden, werden bei RAID-Systemen redundante Informationen gezielt erzeugt, damit beim Ausfall einzelner Komponenten das RAID als Ganzes seine Integrität und Funktionalität behält.

Vorteile

RAID-Systeme bieten folgende Vorteile:

- ✓ **Erhöhung der Fehlertoleranz und Ausfallsicherheit (Redundanz)**
- ✓ **Steigerung der Transferraten durch Parallelität (Leistung)**
- ✓ **Aufbau großer logischer Laufwerke**
- ✓ **Austausch von Festplatten und Erhöhung der Speicherkapazität während des Systembetriebes**
- ✓ **Kostenreduktion durch Einsatz mehrerer preiswerter Festplatten**

Software-RAID vs. Hardware-RAID

Die Begriffe „Software-RAID“ bzw. „Hardware-RAID“ sind etwas irreführend - letztlich benötigen beide Varianten zum Betrieb **Software** - beziehen sich auf die Art und Weise der **Implementierung**.

Software-RAID

Die RAID-Software ist in diesem Fall ein **Teil des Betriebssystems**. Die RAID-Verwaltung wird somit durch den eigentlichen Computer durchgeführt.

Die einzelnen Festplatten sind in diesem Fall entweder über einfache Festplatten-Controller am Computer angeschlossen oder es werden externe Storage-Geräte wie Disk-Arrays, in welchen die Festplatten untergebracht sind, an den Rechner angeschlossen.

Hardware-RAID

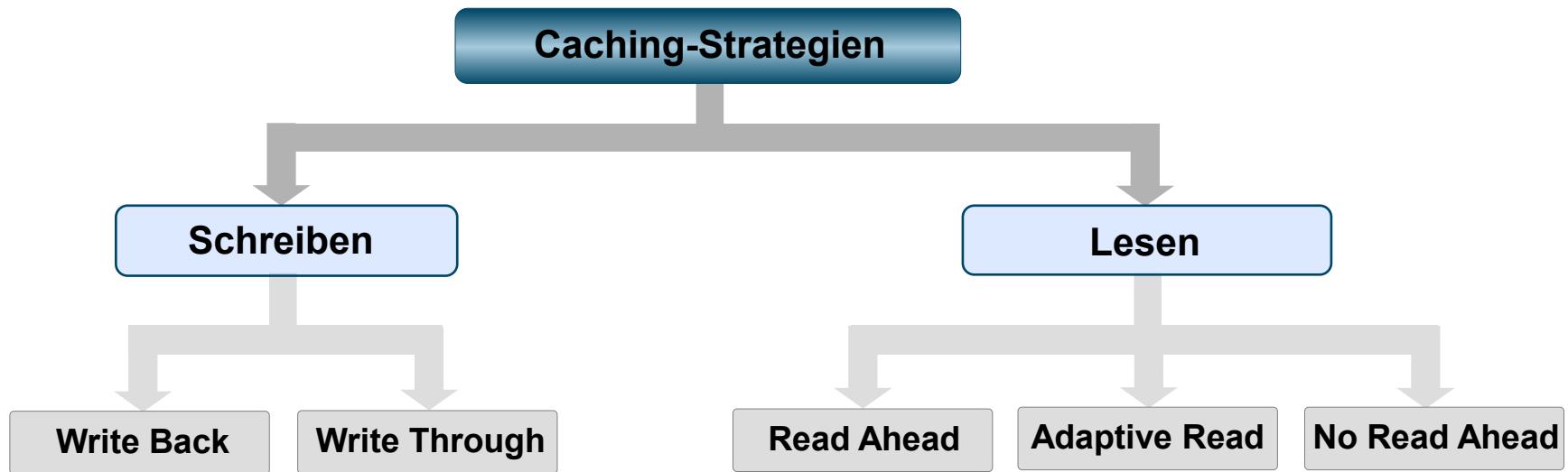
Ein eigener **Hardware-Baustein**, der RAID-Controller, übernimmt die Ansteuerung bzw. die Organisation des Arrays. Die RAID-Verwaltung wird somit durch ein Speicher-Subsystem organisiert.

Der RAID-Controller befindet sich in physikalischer Nähe der Festplatten, häufig in einem eigenen Gehäuse (Disk-Array).

	Vorteile	Nachteile
Software-RAID	<ul style="list-style-type: none">▪ Es wird kein spezieller RAID-Controller benötigt, da die Steuerung von einer RAID-Software erledigt wird, diese ist entweder Teil des Betriebssystems oder wird nachträglich installiert (einfachere Anpassungsmöglichkeiten)▪ Preisgünstigere Lösung	<ul style="list-style-type: none">▪ Hohe Belastung des CPU und der System-Busse▪ Als Cache kann nur der Arbeitsspeicher genutzt werden▪ Arbeitet plattform- und betriebssystemgebunden▪ Zur Steuerung der Laufwerke stehen meist nur zwei oder vier Anschlüsse zur Verfügung, was eine mögliche Parallelisierung der Plattenzugriffe und damit auch die Performance beschränkt
Hardware-RAID	<ul style="list-style-type: none">▪ Entlastung der CPU und damit höhere Performance▪ Bindet die Laufwerke über mehrere Kanäle an, was Laufwerkszugriffe und damit hohe Transferraten ermöglicht	<ul style="list-style-type: none">▪ Teurere Lösung▪ Arbeitet zwar plattformunabhängig, aber auch diese Lösung benötigt zur Verwaltung Software, die auf ein bestimmtes Betriebssystem zugeschnitten ist

RAID-Controller mit Cache-Speicher

- ▶ RAID-Controller besitzen meistens einen zusätzlichen Speicher auf der Steckkarte, um die Daten zwischen Festplatte und Controller zwischenzuspeichern.
- ▶ Der Vorteil einer solchen Lösung besteht in einem hohen **Performance-Gewinn**, da die Daten je nach Anwendung nicht erst von der Festplatte gelesen werden müssen, sondern im schnellen Cache-Speicher zur Verfügung stehen.
- ▶ Um diesen „Buffer“ optimal zu nutzen, bieten die RAID-Controller diverse Konfigurationsmöglichkeiten, die unterschiedliche Schreib- und Lese-Strategien umfassen. Beim Setzen falscher „Settings“ würde man ungewollt auf bis zu 50 Prozent der Storage-Performance verzichten.



- ▶ Welche Read-Strategie die bessere ist, hängt letztlich von der Anwendung ab und von deren Datenstrukturen auf der Festplatte.

Konfigurationsmöglichkeiten des RAID-Controllers

Schreiben

Write Back

RAID-Controller schickt ein Bestätigungsbefehl an das BS, sobald der Pufferspeicher des Controllers die Schreibdaten für die Festplatte vom System erhalten hat.

Der Controller hält die Informationen so lange im Cache, bis er einen geeigneten Zeitpunkt findet, die Daten an die Festplatte zu übertragen. Dies erfolgt zu einem Zeitpunkt, zu dem die Systemressourcen nicht voll beansprucht werden, so dass diese Strategie die Schreibleistung signifikant verbessert.

Nachteile bei Störung der Stromversorgung: Daten, die noch nicht vom Cache-Controller auf die Festplatte geschrieben wurden, gehen unwiderruflich verloren.
Lösung: Cache-Controller mit Pufferbatterie.

Write Through

Der RAID-Controller sendet ein Bestätigungsbefehl erst dann an das BS, wenn die Daten sicher auf die Festplatte geschrieben wurden.

Deshalb kostet das Verfahren Übertragungs- bzw. System-Performance, da die Informationen ohne Zwischenpufferung direkt, ohne Rücksicht auf aktuelle Systemressourcen, auf die Festplatte geschrieben werden.

Lesen

Read Ahead

Neben den tatsächlichen Daten auf der Festplatte fordert der RAID-Controller auch weitere Informationen, die daneben liegen, an.

Diese „vorweggenommene“ Daten werden in den Cache zwischengespeichert.

Werden beim nächsten Request diese Daten verlangt, kann der Controller sie direkt aus dem Puffer lesen und an das Betriebssystem weiterleiten. Ein Zugriff auf die Festplatte ist dann nicht mehr erforderlich.

Adaptive Read

Die „intelligente“ Read Ahead Strategie.

Diese aktiviert Read-Ahead-Lese-Zugriffe erst dann, wenn zwei aufeinander folgende Read-Anweisungen Daten aus zwei hintereinander liegenden Sektoren der Festplatte auslesen.

Erhält der RAID-Controller Daten aus zwei zufälligen Sektoren der Festplatte, schaltet er in den No-Read-Ahead-Modus um.

No Read Ahead

Der RAID-Controller liest nicht „vorausschauend“ die Daten ein.

Das heißt, es wird nur jeweils ein Sektor eingelesen, auch wenn der folgende Nachbarssektor die nächsten Lesedaten enthält.

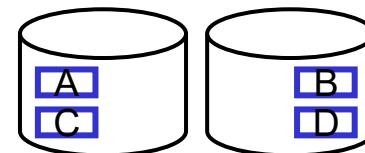
RAID-Level: RAID-Architekturen

- ▶ Die genaue Art des Zusammenwirkens der Festplatten wird durch den RAID-Level spezifiziert.
- ▶ Trotzt der Bezeichnung „Level“ handelt es sich nicht um stufenweise aufbauende Verfahren, sondern um von einander **völlig unabhängige Techniken** (verschiedene Methoden/Varianten/Verfahren), mit denen sich einzelne Platten zu einem Array zusammenfassen lassen bzw. die Verteilung (Declustering, Striping) der Daten auf den verschiedenen Platten stattfindet.
- ▶ Heute reicht die Bandbreite der allgemein verfügbaren Level von RAID 0 bis RAID 6. Hinzu kommen kombinierte Technologien wie RAID 0+1 (RAID 10) oder RAID 50, mit RAID 6 verwandte Verfahren (RAID n+m) sowie herstellerspezifische Varianten (RAID-DP, RAID-X oder RAID n).
- ▶ **Die gebräuchlichsten RAID-Level sind RAID 0, RAID 1 und RAID 5.**
- ▶ Welche der RAID-Level für eine gegebene Anwendung zu bevorzugen ist, hängt vom Anwendungsprofil (z. B. Anteil der Leseoperationen im Vergleich zu Schreiboperationen) und von der zu erzielenden Ausfallsicherheit ab.
- ▶ Heutige kommerziell verfügbare RAID-Systeme erlauben oft eine **flexible Konfiguration** auf den für das jeweilige Anwendungsgebiet optimalen RAID-Level.



- ▶ Trotz der Fehlertoleranz von RAID-Systemen darf die systematische Archivierung und Protokollierung von Datenbankzuständen für die Fehlerrecovery nicht vernachlässigt werden!
- ▶ Die meisten RAID-Level tolerieren nur den gleichzeitigen Ausfall einer einzigen Platte.
- ▶ Der Einsatz von RAID-Systemen kann also nur dazu dienen, die mittlere Zeitdauer bis zu einer höheren Datenbankrecovery zu erhöhen.
- ▶ RAID-Systeme machen die Archivierung und Protokollierung für die Datenbank-Recovery aber nicht obsolet!
...weiterhin Backups!!!

RAID 0: Striping



Vorgehensweise:

- Eine Datei wird in „Streifen“ bzw. Datenblöcke zerlegt und auf alle physischen Laufwerke / Platten verteilt
- Stripinggranularität = Größe der Datenblöcke, Stripingbreite = Anzahl der Platten (hier: 2)

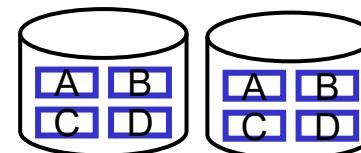
Vorteile:

- Lastbalancierung, gesteigerte Transferraten und hohe Bearbeitungsgeschwindigkeit, da Zugriffe parallel durchgeführt und Blöcke somit von unterschiedlichen Prozessen parallel angefordert werden können
- Doppelte Bandbreite beim sequentiellen Lesen der Datei bestehend aus Blöcken ABCD...

Nachteile:

- Datenverlust wird immer wahrscheinlicher, je mehr Platten man verwendet

RAID 1: Mirroring



Vorgehensweise:

- Spiegelkopie / Redundanz eines jeden physischen Laufwerkes bzw. einer jeden Platte

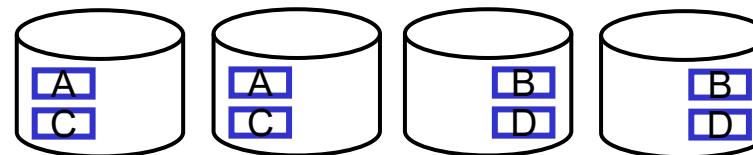
Vorteile:

- Datensicherheit durch Redundanz
- Lastbalancierung beim Lesen:
Leseoperationen werden auf die Laufwerke verteilt, so dass jedes Laufwerk nur noch etwa die Hälfte der Leseanforderungen an die logische Platte zu bearbeiten hat (doppelte Leistung)

Nachteile:

- Doppelter Speicherbedarf
- Doppelarbeit beim Schreiben:
Schreiboperationen auf Blöcken müssen auf beiden Kopien durchgeführt werden, kann jedoch auch parallel geschehen (dauert also nicht doppelt so lange wie das Schreiben nur eines Blocks)

RAID 0+1: Striping + Mirroring



Vorgehensweise:

- Kombiniert RAID 0 und RAID 1:
Datenblöcke werden auf mehrere Laufwerke / Platten aufgeteilt und von diesen Laufwerken existieren Kopien

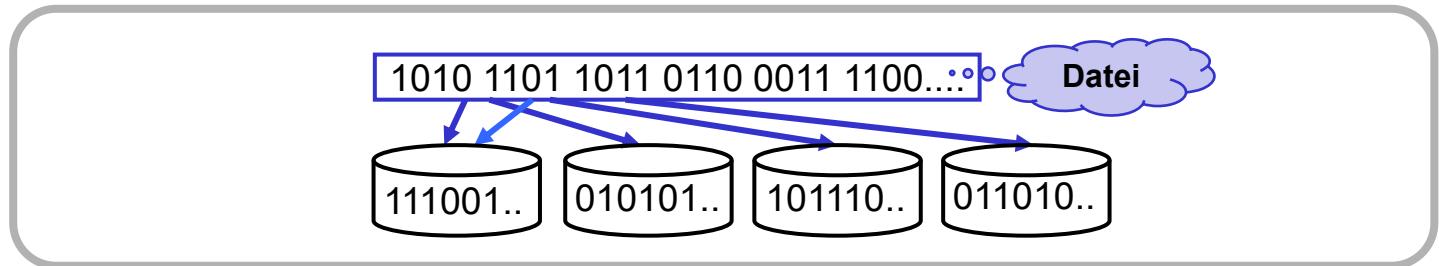
Vorteile:

- Zusätzlich zu RAID 1 erzielt man hierbei auch eine höhere Bandbreite beim Lesen der gesamten Datei ABCD...

Nachteile:

- Immer noch doppelter Speicherplatzbedarf

RAID 2: Bit-Level-Striping



Vorgehensweise:

- Anstatt ganzer Blöcke, wie bei RAID 0 und RAID 0+1, wird das Striping auf Bit- (oder Byte-) Ebene durchgeführt
- Auf einer Platte können noch zusätzlich Fehlererkennungs- und Korrekturcodes gespeichert werden, in der Praxis nicht einsetzbar, da Platten-Controller sowieso schon Fehlererkennungscodes verwalten

Vorteile:

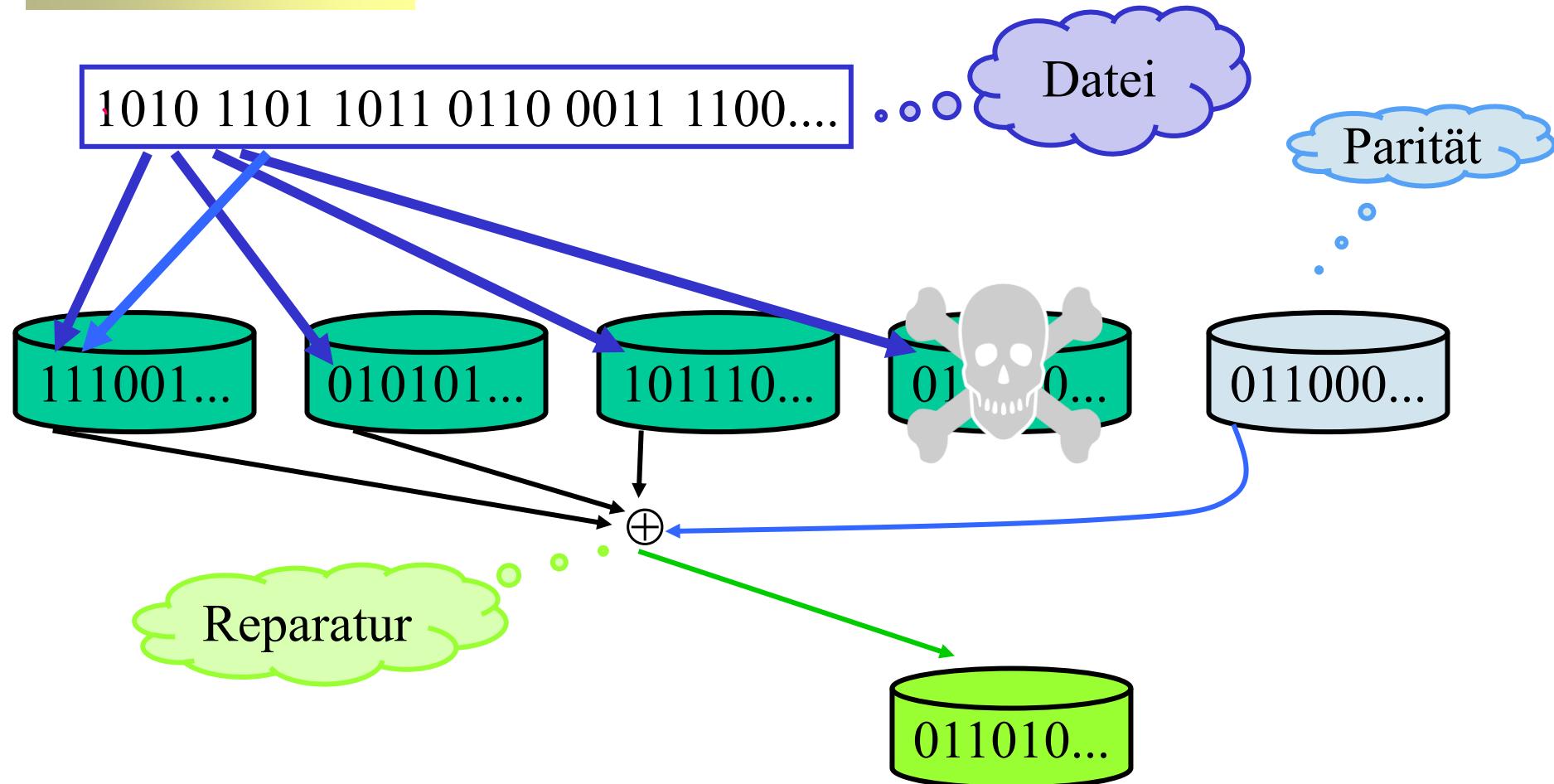
- ...

Nachteile:

- ...

RAID 3: Bit-Level-Striping + separate Paritätsplatte

Plattenausfall ...



RAID 3:

Bit-Level-Striping + separate Paritätsplatte

Vorgehensweise:

- Das Striping wird auf Bit- (oder Byte-) Ebene durchgeführt
- Speicherung von „Redundanzen“ anderer Festplatten auf einer separaten Platte:
 - Über mehrere Daten wird eine Art Prüfsumme, die Parität, berechnet und abgespeichert: Berechnung der Paritätsinformationen durch bitweise Addition der einzelnen Bits der anderen Festplatten (Parität = bitweise xor (+))
 - Mit dieser Prüfsumme kann anschließend festgehalten werden, ob die Daten, die zur Berechnung verwendet wurden, noch korrekt sind und bei Bedarf eine entsprechende Fehlerkorrektur vornehmen
 - Damit dienen die Paritätsinformationen nur zur Fehlerkorrektur bei Ausfall einer der Platten
- Das Lesen einer Datei erfordert den Zugriff auf alle Datenplatten:
Pro Block erfolgt jeweils der Zugriff auf eine der Datenplatten, die Paritätsplatte wird beim Lesen nur in Fehlerfällen verwendet
- Das Schreiben einer Datei erfordert den Zugriff auf alle Platten:
Benötigt werden nicht nur die Datenplatten, sondern auch die Paritätsplatte um die Paritätsinformationen neu zu berechnen
- Bei alle Operationen marschieren die Schreib-/Lese-Köpfe synchron

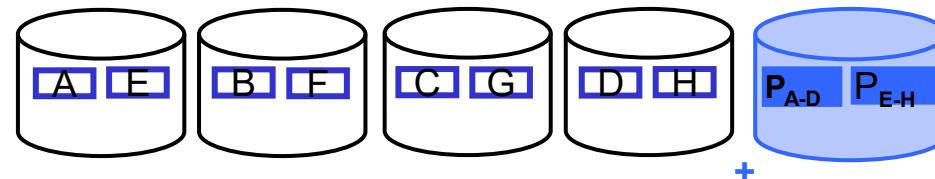
Vorteile:

- Ökonomisch günstigere Vorgehensweise zur Speicherung von „Redundanzen“ anderer Festplatten auf einer separaten Platte

Nachteile:

- Flaschenhals bildet die Paritätsplatte, da jede Schreiboperation auf die eine Paritätsplatte zugreifen muss und diese dementsprechend häufiger ausfällt
- Verschwendungen von Schreib-/Lese-Köpfen

RAID 4: Block-Level-Striping



Vorgehensweise:

- Das Striping wird auf Block-Ebene durchgeführt
- Speicherung von „Redundanzen“ anderer Festplatten auf einer separaten Platte

Vorteile:

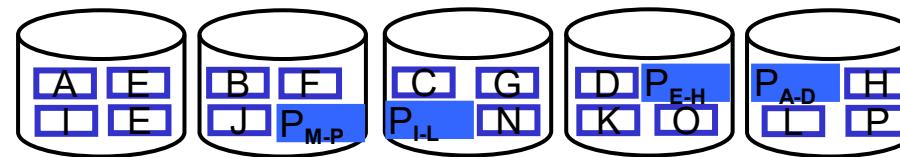
- Bessere Lastbalancierung bei Leseoperationen als bei RAID 3, weil die einzelnen Einheiten, die gelesen werden, größere Datenblöcke und nicht einzelne Bytes darstellen (kleinere Leseanforderungen)

Nachteile:

- Schlechtere Lastbalancierung bei Schreiboperationen als bei RAID 3, da hier sowohl der alte Inhalt des Datenblocks als auch der Paritätsblock gelesen und anschließend der neue Inhalt des Datenblocks und die korrigierte Parität geschrieben werden müssen
 - Bei Modifikation von Block A zu A' wird die Parität PA-D wie folgt neu berechnet: P' A-D := P A-D (+) A (+) A'
- Flaschenhals bildet die Paritätsplatte, da jede Schreiboperation auf die eine Paritätsplatte zugreifen muss und diese entsprechend häufiger ausfällt

RAID 5:

Block-Level-Striping + verteilte Paritätsblöcke



Vorgehensweise:

- Arbeitsweise ähnlich der Arbeitsweise von RAID 4, jedoch Verteilung der Paritätsinformationen auf (mehrere/alle) Laufwerke
- Wird in der Praxis häufig eingesetzt

Vorteile:

- Bessere Lastbalancierung, da (gleichmäßige) Verteilung der Paritätsinformationen
- Die Paritätsplatte bildet jetzt keinen Flaschenhals mehr
- Guter Ausgleich zwischen Platzbedarf und Leistungsfähigkeit

Nachteile:

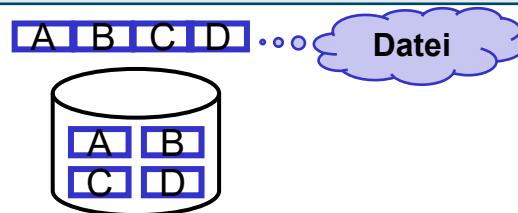
- ...

RAID 6:

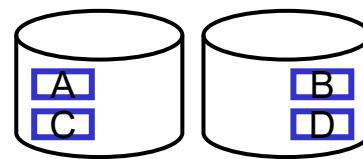
Ist eine Verbesserung der Fehlerkorrekturmöglichkeiten von RAID 5

RAID-Architekturen im Überblick

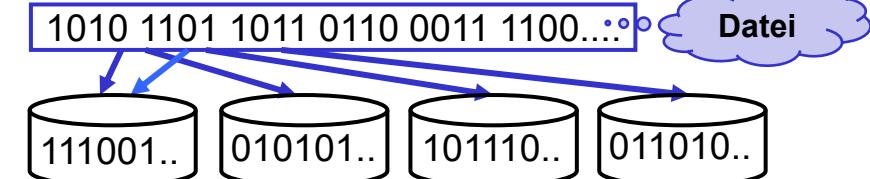
Einsatz einer Festplatte



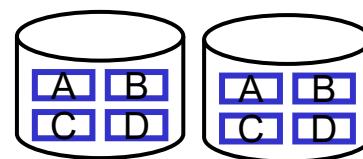
RAID 0: Striping – Beschleunigung ohne Redundanz



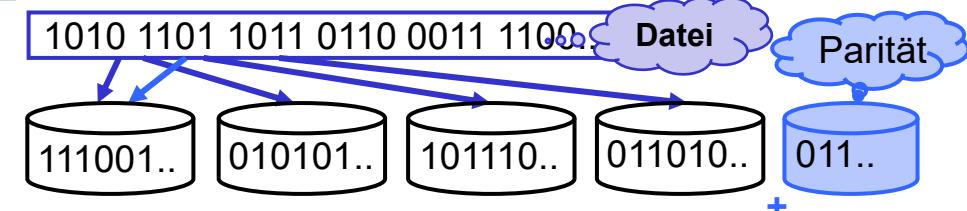
RAID 2: Bit-Level-Striping



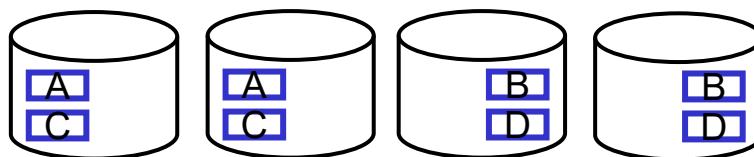
RAID 1: Mirroring – Beschleunigung mit Redundanz



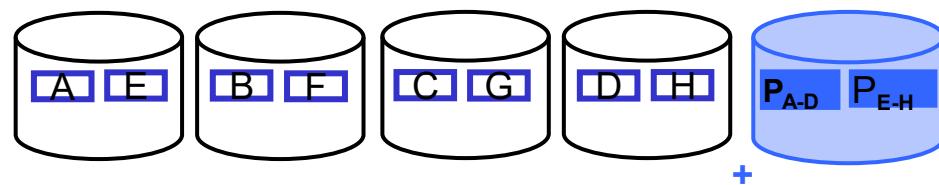
RAID 3: Bit-Level-Striping + separate Paritäts-Platte



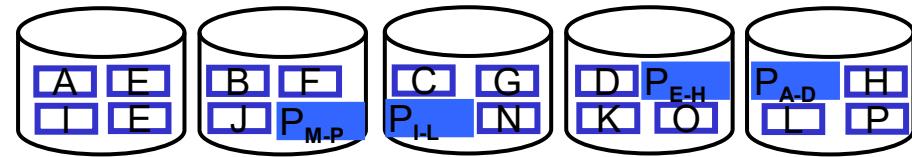
RAID 0+1: Striping und Mirroring



RAID 4: Block-Level-Striping



RAID 5: Block-Level-Striping+verteilte Paritäts-Blöcke



Erhöhte Fehlertoleranz durch Redundanz ...

„The Problem with Many Small Disks: Many Small Faults“

- ▶ Durch viele Platten (sowohl bei Plattenfarmen als auch bei Plattenverbunden)
 - Erhöhte Wahrscheinlichkeit für Ausfall einer Platte,
 - Mögliche Fehlerquellen bei redundanten Datenhaltung:
 - Durch Replikation der Daten (z. B.: Spiegelplatten)
 - Durch zusätzlich zu den Daten gespeicherte Korrekturcodes (z. B.: Paritätsbits)

Durchschnittliche Zeit bis zum Ausfall (mean time to failure; MTTF) ist umgekehrt proportional zur Anzahl der Platten (N): MTTF Array = MTTF Einzelplatte / N



**Ohne Fehlertoleranzmechanismen N-fach erhöhte Ausfallwahrscheinlichkeit
→ System ist unbrauchbar!**

- ▶ Speicherung von Korrekturcodes (z. B. Paritätsbits) ermöglicht Erkennung und Kompensation einzelner Plattenausfälle
- ▶ Kein Ersatz für Backup-Medien; diese werden aber seltener benötigt (geringere Ausfallzeiten)

Partitionierung vs. Allokation

Logische Ebene

Fragen:

Wie teilt man die Datei auf?



Antworten:

Speichere Daten, wo sie gebraucht werden

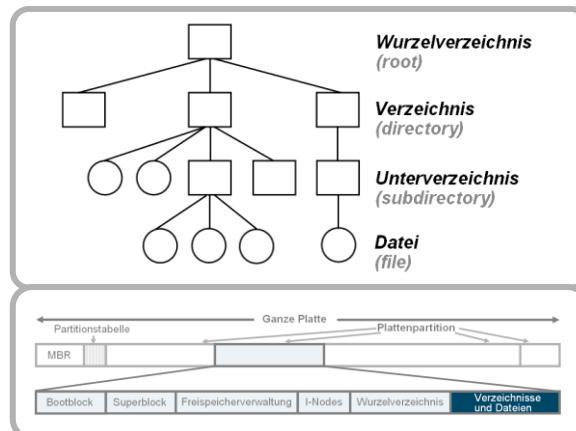
➤ Vermeide Kommunikationskosten



Verfahren:

Partitionierung

➤ Logische Speicherverwaltung (Dateisystem)



Physische Ebene

Wo speichert man die einzelnen Teile?



Verteile Daten gleichmäßig auf alle Platten

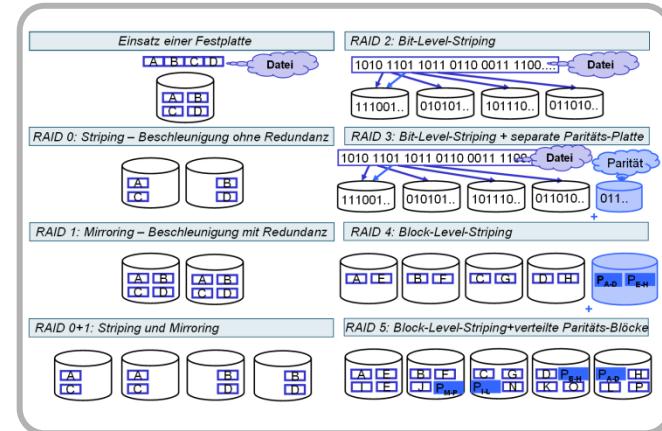
➤ Lastbalancierung

➤ Parallelität – höhere Kommunikation



Allokation

➤ Physische Dateiverteilung und -speicherung auf den Speichermedien



Allokation – Verteilungsverfahren zur Lastbalancierung

- ▶ Die Datenverteilung (Declustering, Striping) erfolgt automatisch durch den RAID-Controller.
- ▶ Ist das Striping-Granulat so groß, dass jede Datei über alle Festplatten verteilt wird, ist dessen Lösung trivial. Ist dies jedoch nicht der Fall, muss man im Hinblick auf eine ausgeglichene Lastverteilung von allen Platten im Disk-Array ein **Verteilungsverfahren** finden, das eben verhindert, dass sich bei I/O – Operationen vor manchen Platten Auftragsschlangen bilden und andere de facto im Leerlauf sind.
- ▶ Maß für die Auslastung einer Platte: Addition der **Hitze** einer jeden Dateipartition

Verteilungsverfahren

Round-Robin-Allokation

Die vorhandenen Platten werden in festgelegter Reihenfolge zyklisch zur Allokation / Belegung einer Dateipartition herangezogen; ein eigentlicher „Hitzeausgleich“ erfolgt nicht.

Vielmehr wird davon ausgegangen, dass bei einer genügend kleinen Partitionierung und einer zahlenmäßigen Gleichverteilung der Partitionen auf jeder Platte im Disk-Array die Hitzeunterschiede einzelner Dateipartitionen gesamt gesehen nicht mehr sonderlich ins Gewicht fallen.

Gibt es jedoch wenige Partitionen, die sehr heiß sind, und viele sehr kalte Partitionen, so kommt es zu einem Ungleichgewicht in der Lastbalancierung der Platten im Disk-Array.

Greedy - Heuristik

- **Voraussetzung:** Jede Datei wird in nicht mehr Partitionen eingeteilt, als Platten im Disk Array vorhanden sind
- **Initialisierung:**
Setze die Gesamthitze aller Platten auf Null
- **Schritt 1: Sortierung**
Alle Dateipartitionen werden nach ihrem Hitzwert sortiert, so dass die heißeste Partition zuerst in der Liste steht.
- **Schritt 2: Abarbeitung der sortierten Liste**
Suche die Platte mit der geringsten Hitze
Alloziere Platz für die aktuelle Partition, sofern auf dieser Platte genügend Platz und keine andere Partition mit derselben Datei vorhanden ist, addiere die Hitze der Partition zur Gesamthitze der Platte

Allokation – Verteilungsverfahren zur Lastbalancierung

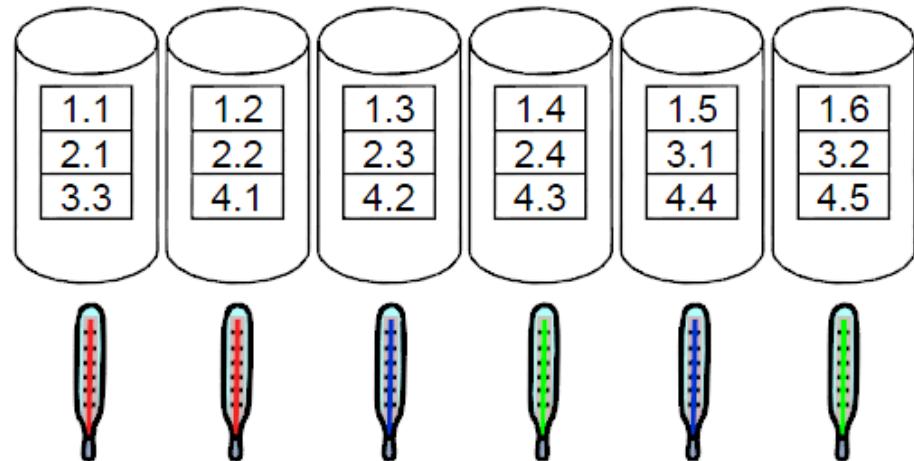
Datei 1						
<table border="1"><tr><td>1.1</td><td>1.2</td><td>1.3</td><td>1.4</td><td>1.5</td><td>1.6</td></tr></table>	1.1	1.2	1.3	1.4	1.5	1.6
1.1	1.2	1.3	1.4	1.5	1.6	
Hitze: 10 4 4 3 2 1						

Datei 2				
<table border="1"><tr><td>2.1</td><td>2.2</td><td>2.3</td><td>2.4</td></tr></table>	2.1	2.2	2.3	2.4
2.1	2.2	2.3	2.4	

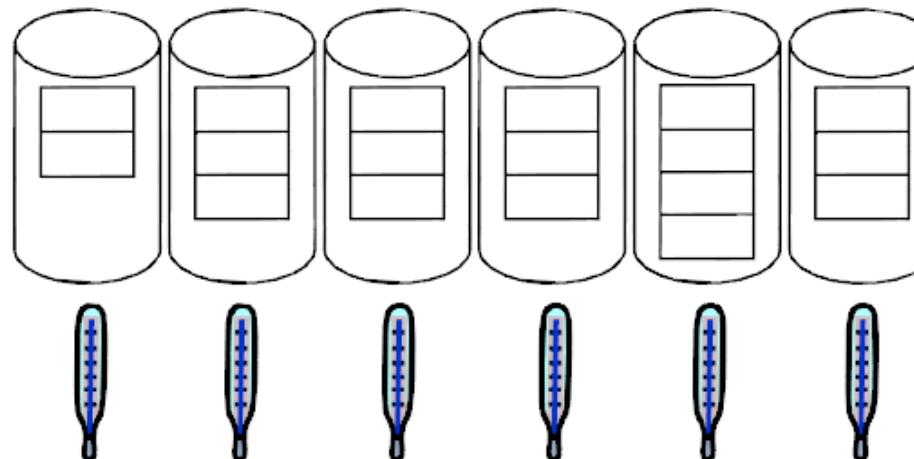
Datei 3			
<table border="1"><tr><td>3.1</td><td>3.2</td><td>3.3</td></tr></table>	3.1	3.2	3.3
3.1	3.2	3.3	

Datei 4					
<table border="1"><tr><td>4.1</td><td>4.2</td><td>4.3</td><td>4.4</td><td>4.5</td></tr></table>	4.1	4.2	4.3	4.4	4.5
4.1	4.2	4.3	4.4	4.5	

Round-Robin-Allokation



Greedy-Heuristik

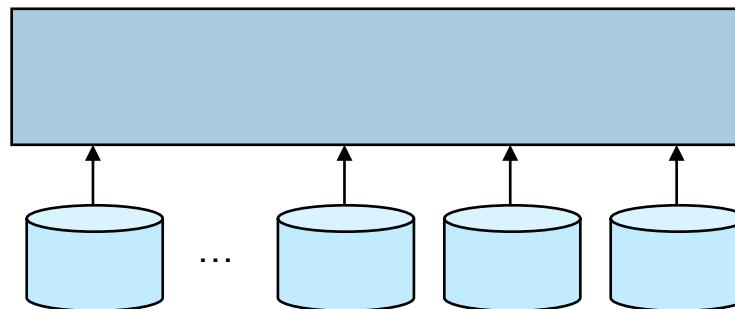


Leistungssteigerung durch Parallelität

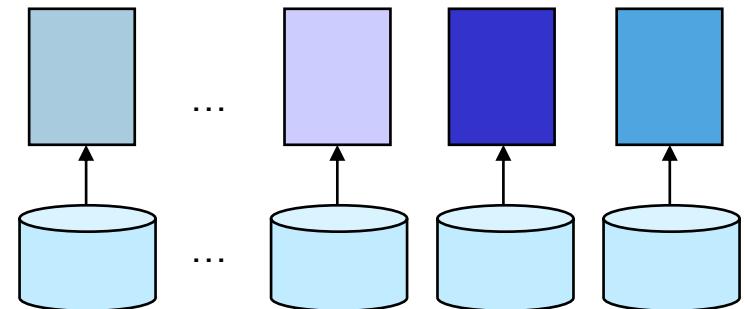
► Voraussetzung: Verteilung / Declustering / Striping von Dateien über mehrere Platten

Arten von E/A - Parallelität

Intra – E/A - Parallelität



Inter – E/A - Parallelität



1 E/A - Auftrag wird von mehreren, parallel ausführbaren Platten umgesetzt und dadurch schneller bearbeitet

N unabhängige E/A – Aufträge können parallel ausgeführt werden, sofern die betreffenden Daten über verschiedene Platten verteilt sind

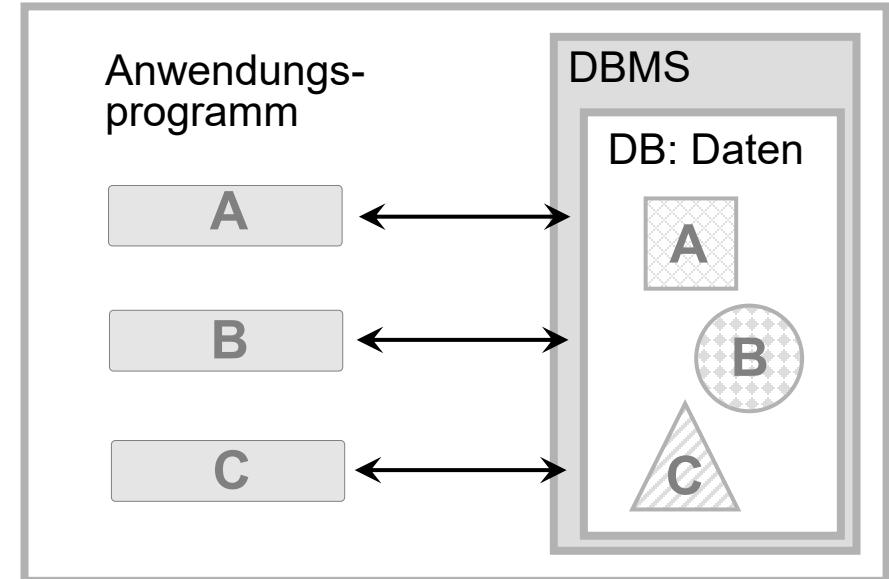
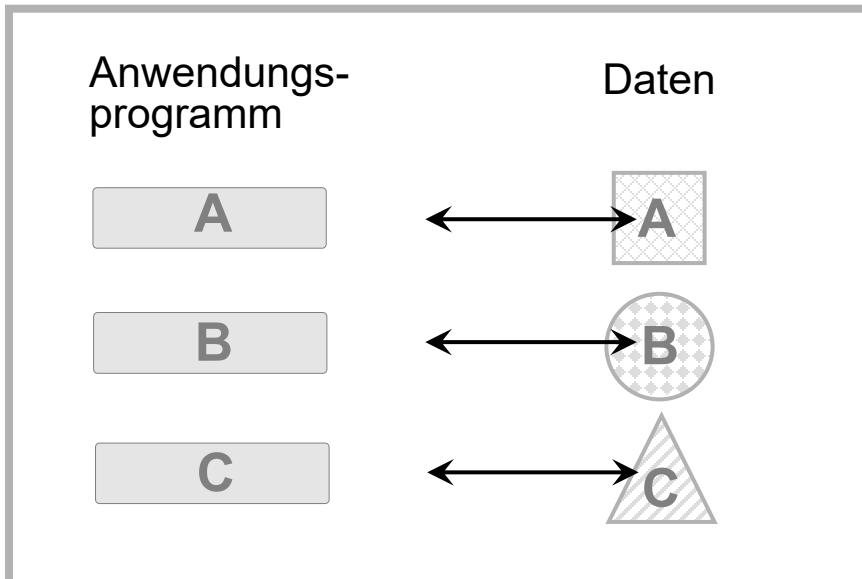
Dateisystem vs. Datenbanksystem

- ▶ Dateien werden zur dauerhaften Speicherung von Programmkonfigurationen, zur Ablage von Datentabellen, zur Ausgabe von Berichten oder auch zur Protokollierung von System- und Programmfunctionen benötigt.
- ▶ Fast jedes Programm arbeitet mit Dateien. Programmiersprachen stellen Anweisungen bereit, um Daten zu verwalten. Je nach Einsatzgebiet und nach dem jeweiligen Dateninhalt sind für die Dateien unterschiedlichen Datentypen sinnvoll (Office-Dokumente, DB-Dateien, etc.).
- ▶ Jedes Betriebssystem stellt zum Zwecke der logischen Speicherverwaltung ein Dateisystem bereit. Dateien im Datenbankumfeld benötigen jedoch eine spezielle Form der Verwaltung....

*Mit einer Artikeldatei ist es in einem Unternehmen nicht getan.
Kunden, Lieferanten, Artikelbuchungen sind weitere Beispiele.*

Dateisystem

Datenbanksystem



Dateisystem vs. Datenbanksystem

► Meistens hat man mit einer je nach Aufgabenstellung mehr oder weniger großer **Anzahl von Dateien** zu tun, deren Daten meist voneinander abhängig sind. Diese logischen **Beziehungen zwischen den Daten** müssen eingehalten, d. h. geprüft werden.

- *Wird ein Artikel an einen Kunden geliefert, muss sichergestellt sein, dass mit dem Lieferschein der Artikelbestand um die ausgelieferte Menge reduziert wird.*
- *Ist bei einem Artikel eine Lieferantennummer eingetragen, müssen auch Lieferantendaten vorhanden sein – und dürfen dann nicht gelöscht werden.*

► Diese und andere Forderungen lassen sich mit unabhängigen Dateien nicht einfach realisieren. Man braucht ein System, das alle gespeicherten Daten überwacht und zum Beispiel auch bei einer Datensicherung dafür sorgt, dass die Konsistenz gewahrt bleibt.

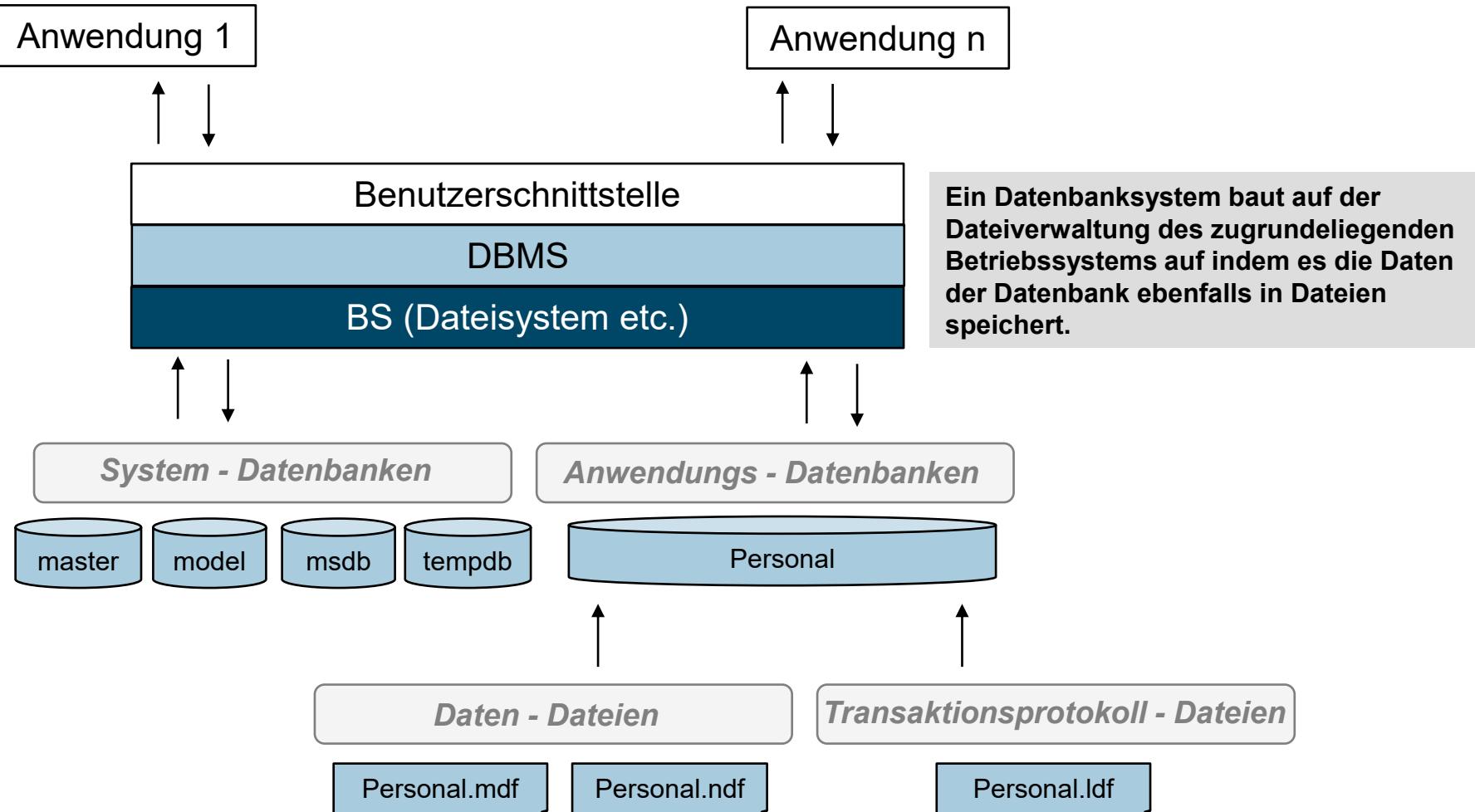
- *Es können nicht die Artikelbestände und später die Lieferscheine gesichert werden, da inzwischen neue Lieferscheine erstellt worden sind und diese damit mit den Artikelbeständen nicht mehr konsistent sind.*

► Systeme, die alle diese Anforderungen abdecken, werden als **Datenbanksysteme** bezeichnet. Die Speicherung der Daten, der Indizes sowie der notwendigen Verwaltungsdaten wird für den Anwender bzw. den Programmierer vollkommen transparent.

- *Artikeldaten können sich über mehrere Festplatten erschrecken (horizontales Splitting),*
- *Indizes werden (auch aus Performance-Gründen) auf andere Platten gespeichert als die indizierten Datensätze und Ähnliches.*
- *Manche Systeme erlauben auch vertikales Splitting (oft benötigte Felder eines Datensatzes werden auf schnellen Platten und selten benötigte Felder auf langsameren Platten gespeichert).*

Dateisystem vs. Datenbanksystem

DBS = DBMS + n * DB



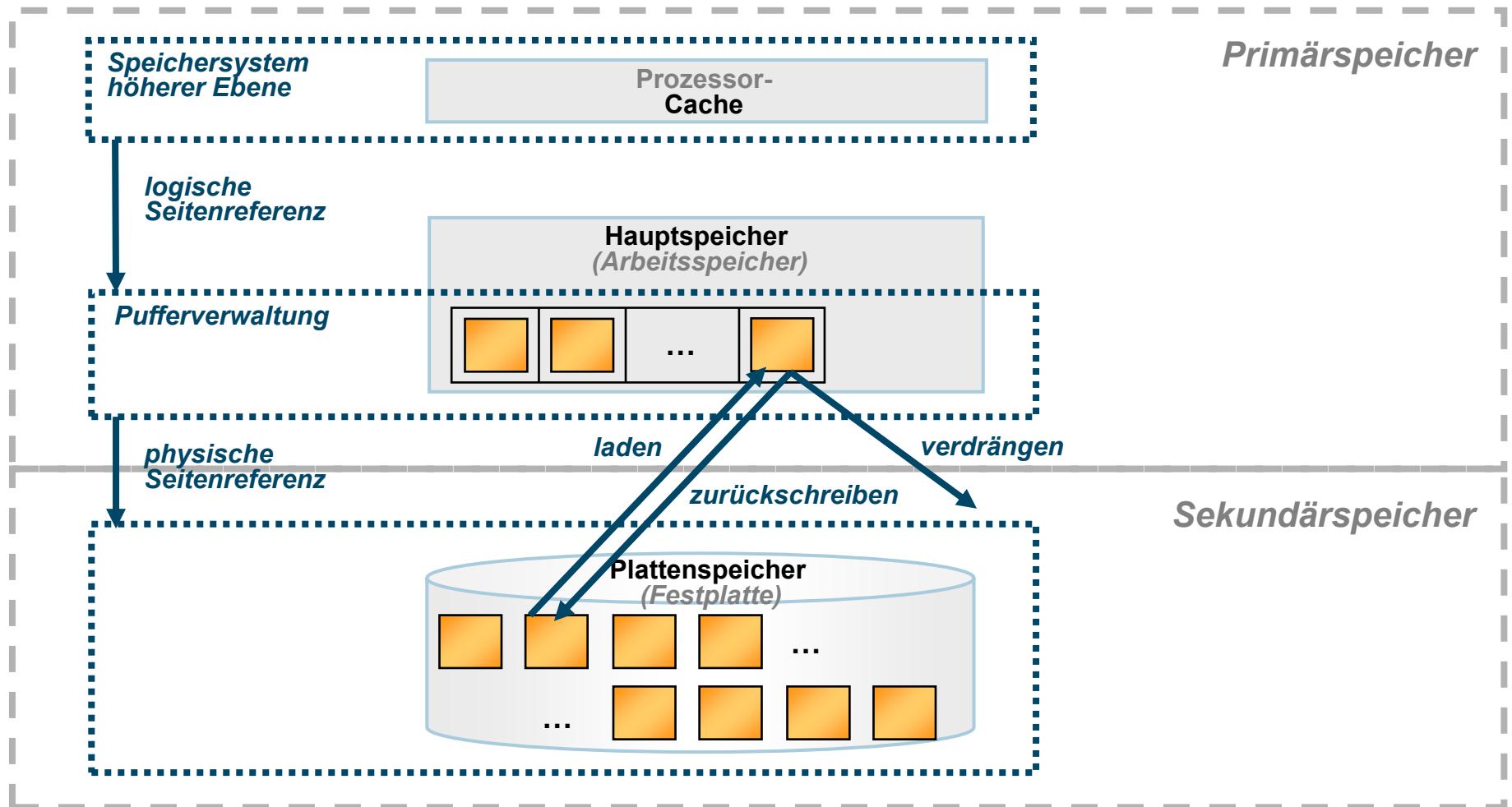
Pufferverwaltung des Hauptspeichers

- Da ein Betriebssystem auf einem File nur eine beschränkte Anzahl an Operationen ausführen kann, werden Datei-Seiten / Blöcke zur Bearbeitung eines Benutzer-Auftrags vom Sekundärspeicher in den Puffer des Hauptspeichers (ca. 25 % des verfügbaren Speicherplatzes) geladen. Dort ist es möglich weitere / für die Anfragen einer Datenbank wesentliche Operationen zu realisieren.
- Aus logischer Sicht enthalten die zwischen Haupt- und Sekundärspeicher bzw. zwischen Puffer und Platte transferierten Seiten die Records eines Files.
- Abbildung der konzeptuellen Ebene auf interne Datenstrukturen:

Konzeptionelle Ebene	Interne Ebene	Platte
Relation (Tabelle)	Log. File (Datei)	Phys. File (Datei)
Tupel (Datensatz)	Record (Satz)	Seiten (Verwaltung in Blöcken)
Attributwert	Feld	Byte
Operationen auf Files		create, delete, open, close, append, read, write, ...
Operationen auf Records		Suchen, Updaten, Einfügen, Ändern oder Löschen

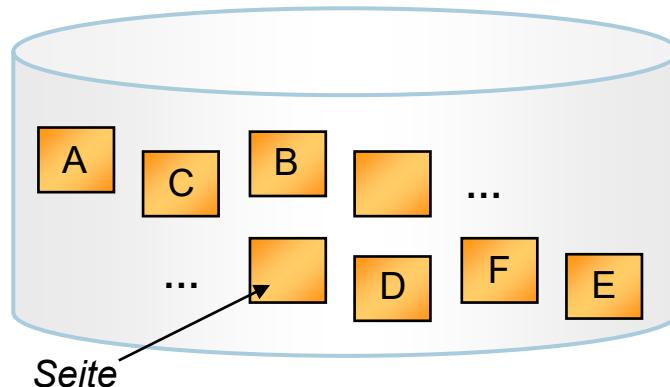
- Varianten der Abbildungen:
 - Beispiel 1: Jede Relation in je einer logischen Datei, diese insgesamt in einer einzigen physischen Datei
 - Beispiel 2: Cluster-Speicherung – mehrere Relationen in einer logischen Datei

Pufferverwaltung des Hauptspeichers



Pufferverwaltung des Hauptspeichers

Datenbank auf dem Hintergrundspeicher
(Festplatte)

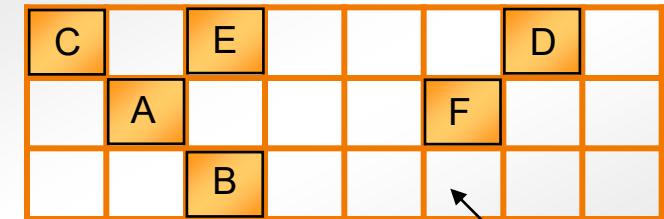


Seite

Einlagerung →
← Auslagerung

Hauptspeicher
(Arbeitsspeicher)

DB-Puffer



Seitenrahmen

DB - Puffer:

- Einem Datenbanksystem wird ein Pufferbereich des Hauptspeichers zugewiesen
- Dieser Pufferbereich wird in Pufferrahmen gegliedert
- Jeder Pufferrahmen kann eine Seite der Platte aufnehmen

Die Aufgaben der Pufferverwaltung sind unter anderem

- die Zuteilung von Speicherplatz für Seiten,
- das Suchen (Suchverfahren) und Ersetzen (Seitenwechselstrategien) von Seiten im Puffer und
- die Optimierung der Lastverteilung zwischen parallelen Transaktionen.

Pufferverwaltung des Hauptspeichers

Welche Datenobjekte sollen im **Hauptspeicher**, dem Datenpuffer, gehalten werden?

„Halte ein Datenobjekt im Hauptspeicher, falls es innerhalb von etwa 5 Minuten erneut referenziert wird; andernfalls lies es erneut von der Magnetplatte“.

Der prinzipieller Ablauf eines Zugriffs auf eine Seite:

- 1) Die höhere Schicht (Speichersystem) fordert bei der Pufferverwaltung eine Seite an (logische Seitenreferenz).
- 2) Folgende Situationen sind nun möglich:
 - a) Die angeforderte Seite ist im Puffer und wird dem Speichersystem zur Verfügung gestellt.
 - b) Die angeforderte Seite ist nicht im Puffer. Es wird eine physische Seitenreferenz durch die Pufferverwaltung an die Betriebssystemebene weitergegeben.
- 3) Nachdem die gewünschte Seite der Pufferverwaltung zur Verfügung gestellt wurde,
 - a) muss im allgemeinen Fall bei gefülltem Puffer eine Seite aus dem Puffer verdrängt werden.
 - b) Falls die zu verdrängte Seite geändert wurde, muss diese erst auf den Sekundärspeicher zurückgeschrieben werden.

Dateistrukturen und Zugriffsmethoden

Benutzer

Für den Benutzer einer Datenbank ist nur von Interesse,
dass seine **Aufträge** stets **schnell** erledigt werden.



DBMS

Für das DBMS ist es wesentlich, dass die vorhandenen Files so organisiert bzw. gespeichert sind,
dass die genannten **Operationen effizient** ausgeführt werden können.



Betriebssystem

Daten einer Datei werden in Einheiten von Datensatzblöcken eingeteilt und auf Platten abgelegt.
Zur Ablage dieser **Datenblöcke** nimmt das Betriebssystem eine **Zuteilung von Plattenplatz** vor.



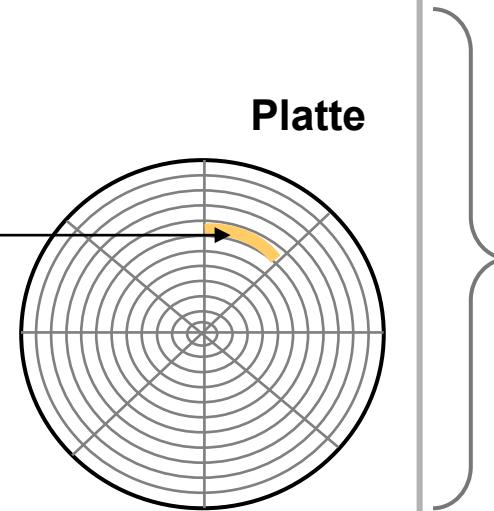
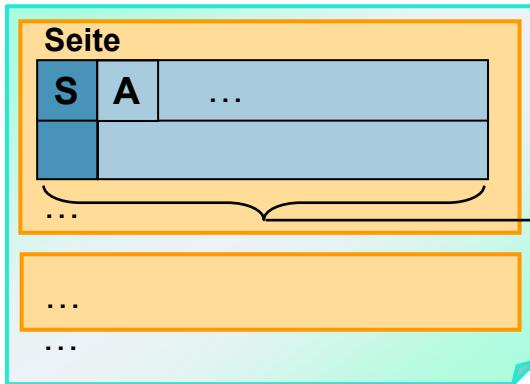
Anwendungsprogrammierer und Datenbankentwickler

Kenntnisse über **Datenstruktur** (Dateistruktur)
und verfügbare **Zugriffsmethoden** sind essentiell für jeden,
der selbst performante Anwendungen entwickeln möchte.



Dateistrukturen und Zugriffsmethoden

Datendatei (Hauptdatei)



Dateiorganisation
=

Dateistruktur

- ▶ **Dateiorganisation:** Art der Anordnung der Datensatzblöcke einer Datei auf einem Speichermedium bzw. Form der Speicherung der internen Relation.
- ▶ Die Organisations- bzw. Speicher – Form einer Datei bestimmt in hohem Maße die möglichen Zugriffsarten (-methoden, -verfahren) und damit auch die Geschwindigkeit bzw. die Effizienz der Operationen auf den Datensätzen sowie den benötigten Speicherplatz.

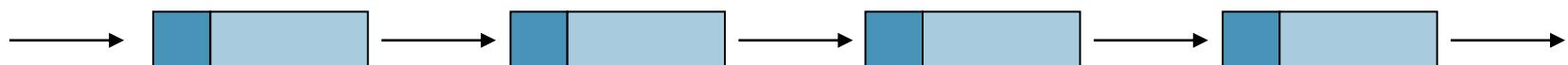


Sequentielle Datei ohne Verweise: Lineare Liste

- Alle Datensatzblöcke eines Files werden physisch zusammenhängend und in aufeinanderfolgenden Speicherzellen (Adressen) abgelegt. Ein Datensatzblock (Tupel) hat folgenden Aufbau:



- Ein Zugriff ist dann einfach und schnell (erfordert wenige Kopfbewegungen), da die Verarbeitung grundsätzlich der Reihe nach (linear) zu erfolgen hat:



Sequentielle Datei mit Verweisen: Verkettete Liste

- Jedes File ist eine verkettete Liste von Datensatzblöcken. Verkettete Listen gehören zu den dynamischen Datenstrukturen, die eine Speicherung von einer im Vorhinein nicht bestimmter Anzahl von miteinander in Beziehung stehenden Objekten erlauben. Dabei beinhaltet jedes Listelement als Besonderheit ein **Verweis auf das nächste Element** bzw. auf die jeweils folgende **Speicher- Zelle/Adresse** des Speichers. Ein Listelement hat folgenden Aufbau:

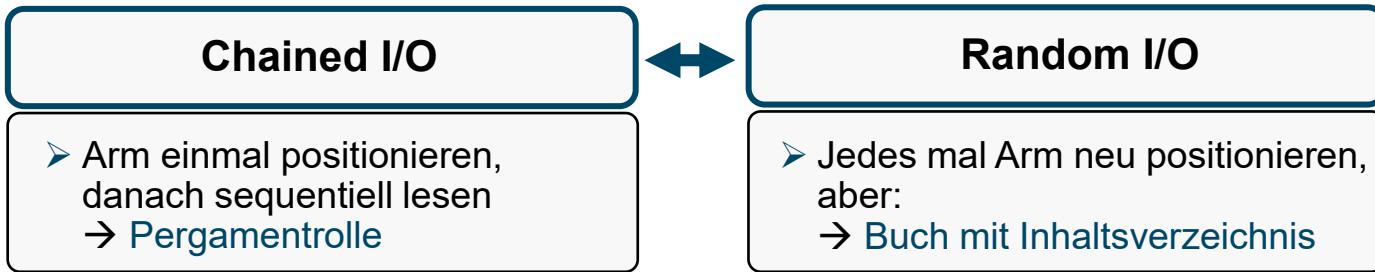


- Verkettete Listen brauchen etwas mehr Speicher um die Verweise zwischen den Listelementen zu speichern. Hierbei wird allerdings nur ein sequentieller Zugriff gut unterstützt. Man unterscheidet grundsätzlich zwischen **einfach** und **mehrfach** oder doppelt verketteten Listen.
Beispiel für die einfach verkettete Liste:



Indizierte Organisationsform einer Datei: Indexstrukturen

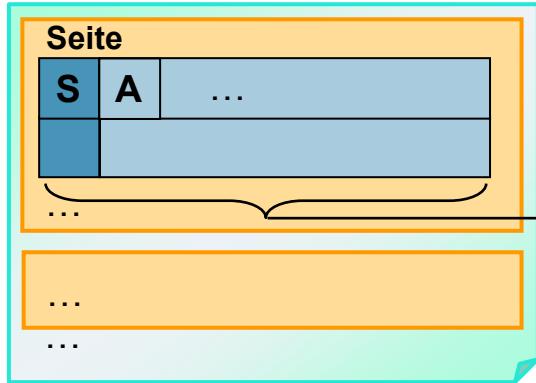
- ▶ Daten werden üblicherweise sequentiell auf einem Speichermedium verwaltet.
- ▶ Sequentielle Bearbeitung einer Suchanfrage ist aber mit linearem Aufwand verbunden, im ungünstigsten Fall müsste bei einer Anfrage auf einen Datensatz der komplette Datenbestand einer Datei durchsucht werden.
- ▶ Sinnvoll wäre es, die Direktzugriffsmöglichkeiten des Sekundärspeichers zu nutzen. Der Zugriff auf einzelne Datensätze einer Datei kann aber erst beschleunigt werden, wenn man zuvor für ein bestimmtes Attribut bzw. Attributmenge einer Relation oder einer Sicht ein zusätzliches „**Inhaltsverzeichnis**“ angelegt hat, welches man als Index bezeichnet.
- ▶ Wird nun ein bestimmter Datensatz anhand eines Suchkriteriums (Attribut bzw. Attributmenge) in einer Datei gesucht, kann über eine Indexstruktur direkt auf den entsprechenden Datensatz zugegriffen und damit eine aufwendige Suche vermieden werden.
- ▶ Der Index erlaubt es, die Position des Datensatzes innerhalb des Mediums schnell zu bestimmen.



Inhalt:	
1	Einleitung
1.1	Thema 1 10
1.2	Thema 2 12
1.3	Thema 3 15
2	Fortsetzung
2.1	Thema 2a 16
2.2	Thema 2b 18
2.3	Thema 2c 20

Dateiorganisation vs. Zugriffspfad

Datendatei (Hauptdatei)

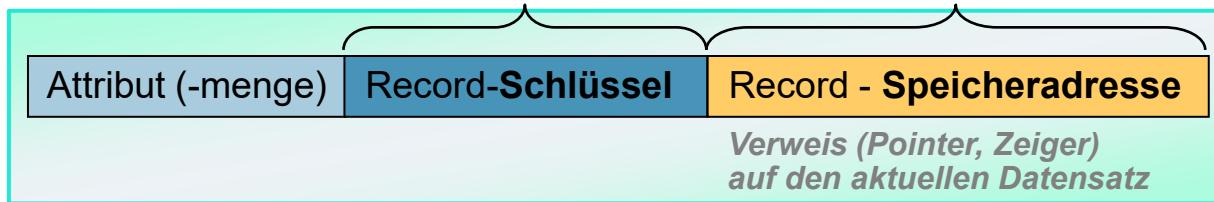


Logische Adresse

Platte

Physische Adresse

Dateiorganisation
bzw.
Dateistruktur



Indexdatei pro Attribut (-menge)

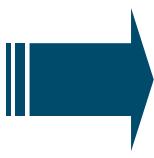
Zugriffspfad

► Zugriffspfad:

Über grundlegende Dateiorganisationsform hinausgehende Zugriffsstruktur,
etwa Indexdatei.

Sequentielle Datei + Index-Datei(en)

- ▶ Ein Index ist eine von der Datenstruktur (Datendatei) getrennte Indexstruktur (zusätzliche Datei) in einer Datenbank.
- ▶ Eine Indexdatei enthält für ein bestimmtes Attribut (Spalte) oder eine Attributmenge (Spalten) einer Tabelle oder einer Sicht die **Adressen (logische und physische) der Records** und ist damit der jeweiligen Tabelle oder Sicht direkt zugeordnet.
- ▶ Mit Hilfe dieser Adressen werden die Suche nach den Attribut-Werten und das Sortieren nach dem entsprechenden Attribut beschleunigt.
- ▶ Durch das Erstellen eines gut durchdachten Indizes kann die Leistung von Datenbankabfragen (Abrufen von Zeilen aus Tabellen oder Sichten) und Anwendungen erheblich gesteigert werden. Indizes können die Datenmenge reduzieren, die gelesen werden muss, um das Resultset der Abfrage zurückzugeben.
- ▶ Ohne Index müsste die Spalte sequentiell durchsucht werden, was selbst mit modernster Hardware und Software viel Zeit in Anspruch nehmen kann.

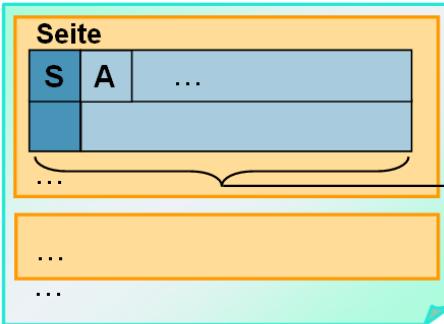


Effiziente Anfrageverarbeitung benötigt Indexstrukturen!

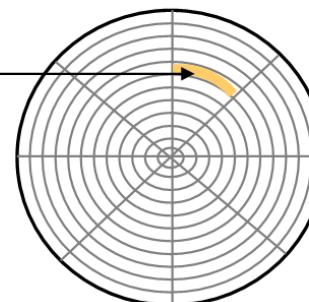
Sequentielle Datei + Index-Datei(en)

Aufbau einer Index-Datei (vereinfacht)

Datadatei (Hauptdatei)



Platte



Logische Adresse

Physische Adresse



Indexdatei pro Attribut (-menge)

Dateien zu einer DB-Tabelle

Daten –
Datei

&

View-
Datei

...

Indizierte Spalte(n) einer Tabelle

Index-
Datei

Index-
Datei

...

Indizierte Spalte(n) eines Views

Index-
Datei

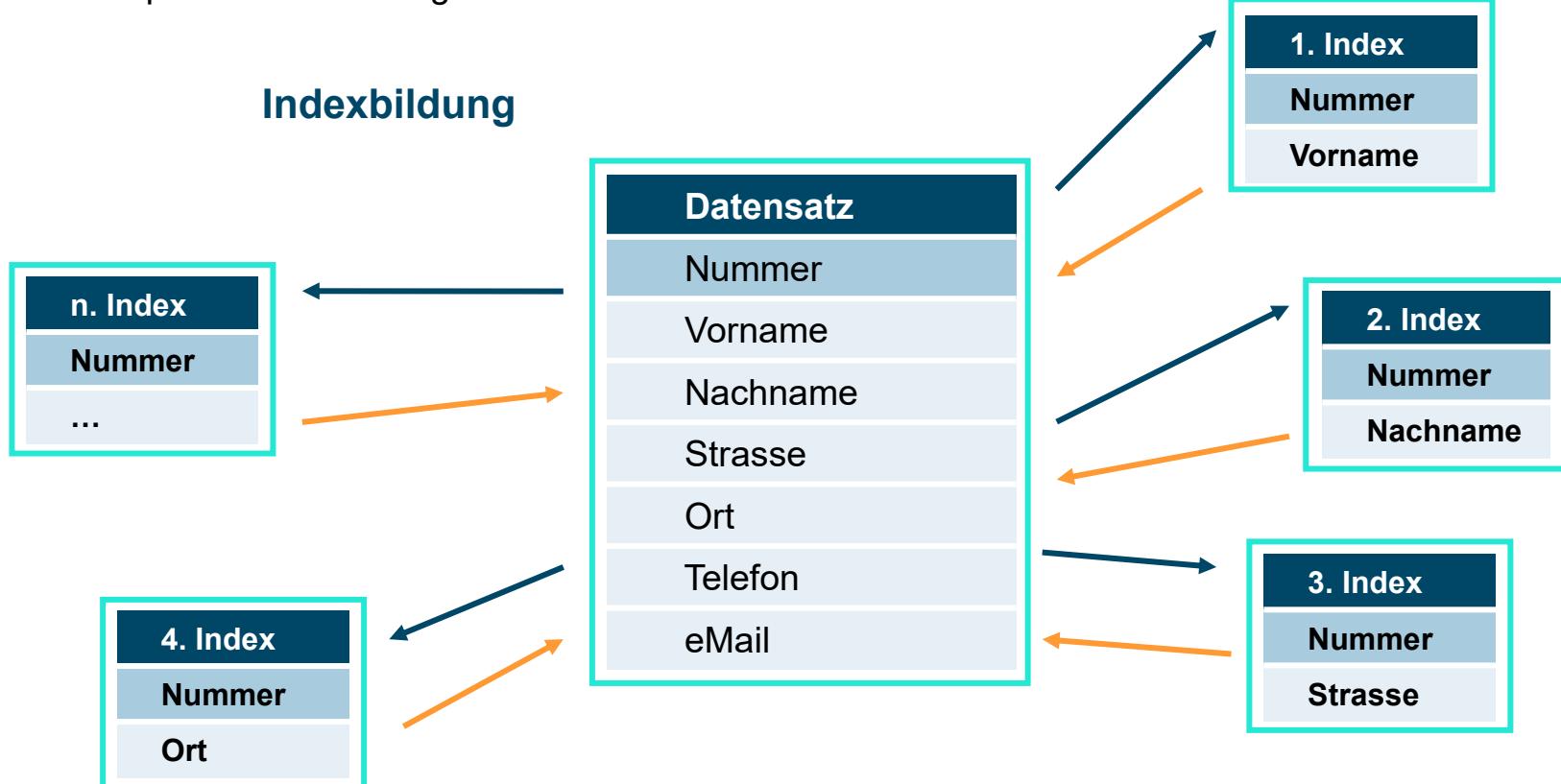
Index-
Datei

...

Sequentielle Datei + Index-Datei(en)

- Indexsequentieller Dateizugriff -

Indexbildung



Indexbasierte Suche mit Zugriff
auf den vollständigen Datensatz

Indexbildung

Indexbildung implizit

Indizes werden **automatisch** erstellt,
wenn **PRIMARY KEY-** und **UNIQUE-Einschränkungen** für Tabellenspalten definiert werden!



Indexbildung explizit

- ▶ **Explizite Erstellung** eines Indexes mit folgendem (vereinfachten) SQL-Befehl:

```
CREATE INDEX Indexname ON Tabellenname ( Spaltenname (n) );
```

- ▶ **Explizite Löschung** eines Indexes mit folgendem SQL-Befehl:

```
DROP INDEX Indexname;
```

Physische Adressierung von Tupeln in einer sortierten Datei

Sowohl die Datensätze als auch der Index werden nach den Schlüsseln geordnet gespeichert:

Datensätze einer Datodatei

- Ein Datensatz besteht aus dem **Schlüssel** (Primär-Schlüssel) und weiteren Informationen (Attributen).

Datensätze einer Indexdatei

- Üblicherweise spricht man bei dem für den Index verwendeten **Suchkriterium** (Zugriffs-Attribut bzw. -Attributmenge) von **Schlüsseln** des Indexes.
- Dieser **Schlüsselbegriff** hat nichts mit den bisher eingeführten **Schlüsseln** zu tun.
- Beispiel: Es ist durchaus möglich die Anzahl der Bestellungen für ein Artikel als Suchkriterium für einen Index zu verwenden, obwohl BestellungNr kein **Schlüssel** der Relation Bestellposition ist.
- Im folgenden sind die erwähnten **Schlüssel** als **Such-Schlüssel** und nicht als **Schlüssel** von Relationen zu verstehen.

Primär-Schlüssel

Relationen-Schlüssel

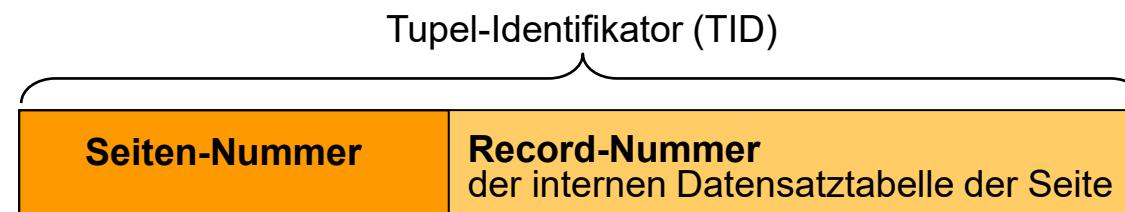
Attribut(menge)

Primär-Schlüssel

Such-Schlüssel

Physische Adressierung von Tupeln in einer sortierten Datei

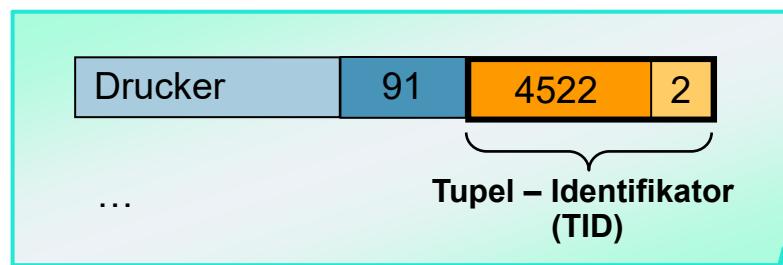
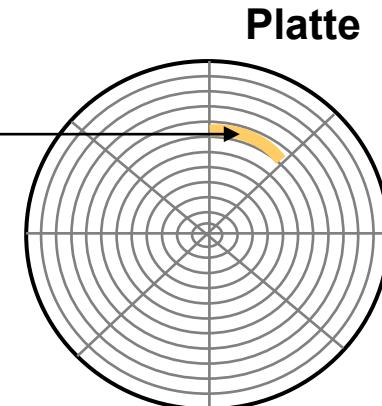
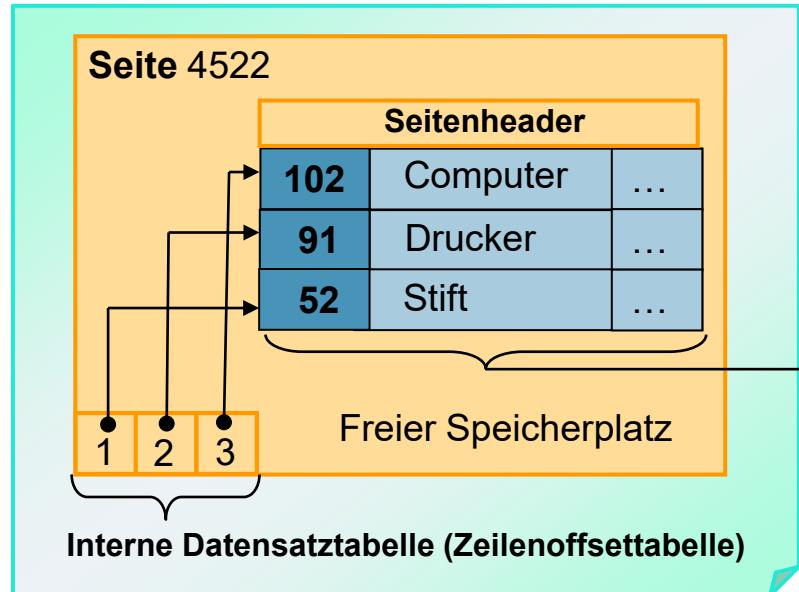
- ▶ Für jede Relation werden mehrere sequentiell angelegte Seiten (Blöcke) auf dem Hintergrundspeicher zu einer Datei zusammengefasst.
- ▶ Jede Seite enthält
 - die **Datensätze** selbst und
 - jeweils eine **interne Datensatztabelle (sortierte Ablage!)**, die intern Verweise auf alle auf der Seite befindlichen Tupel verwaltet
- ▶ Um nun ein Tupel auf der physischen Ebene referenzieren zu können, wird ein sogenannter Tupel-Identifikator (TID) verwendet. Ein **TID** besteht aus zwei Teilen:
 - einer Seitennummer und
 - einer Nummer eines Eintrags in der internen Datensatztabelle, der auf das entsprechende Tupel verweist.



Physische Adressierung von Tupeln in einer sortierten Datei

▶ Speichern von Tupeln auf einer Seite

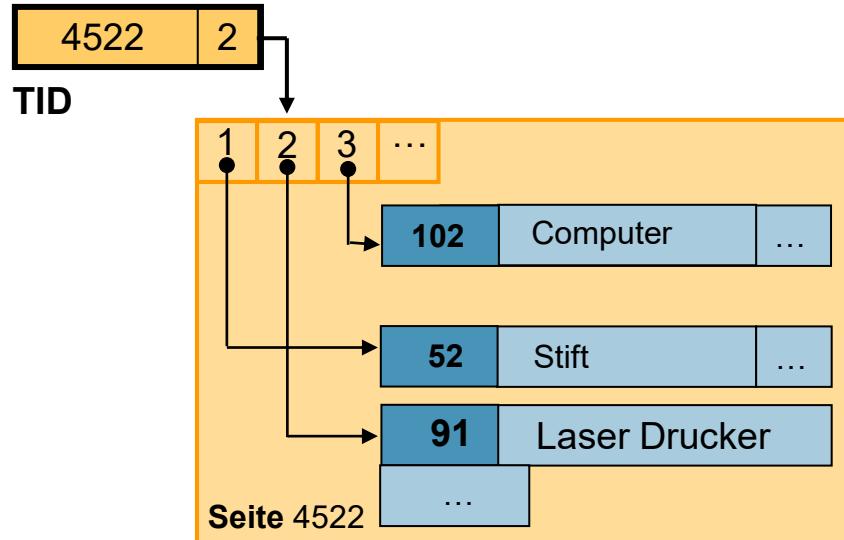
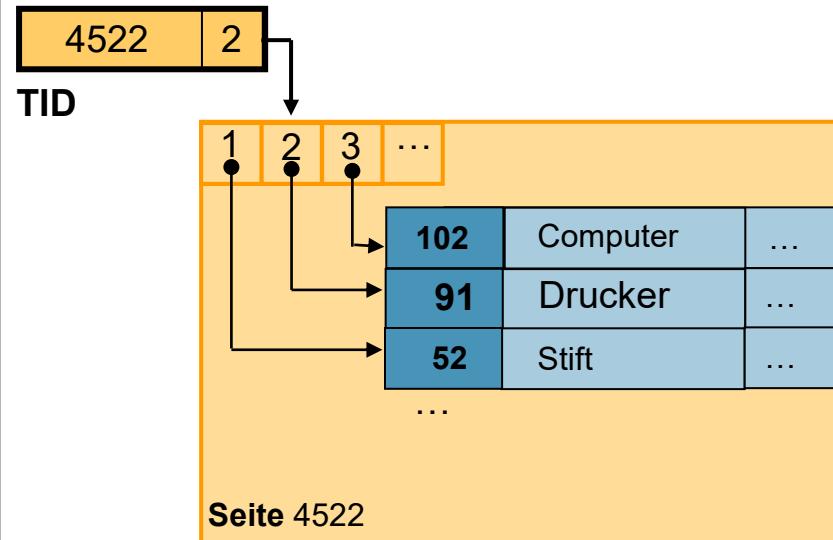
Datendatei (Hauptdatei)



Indexdatei pro Attribut (-menge)

Physische Adressierung von Tupeln in einer sortierten Datei

► Verschieben eines Tupels innerhalb einer Seite

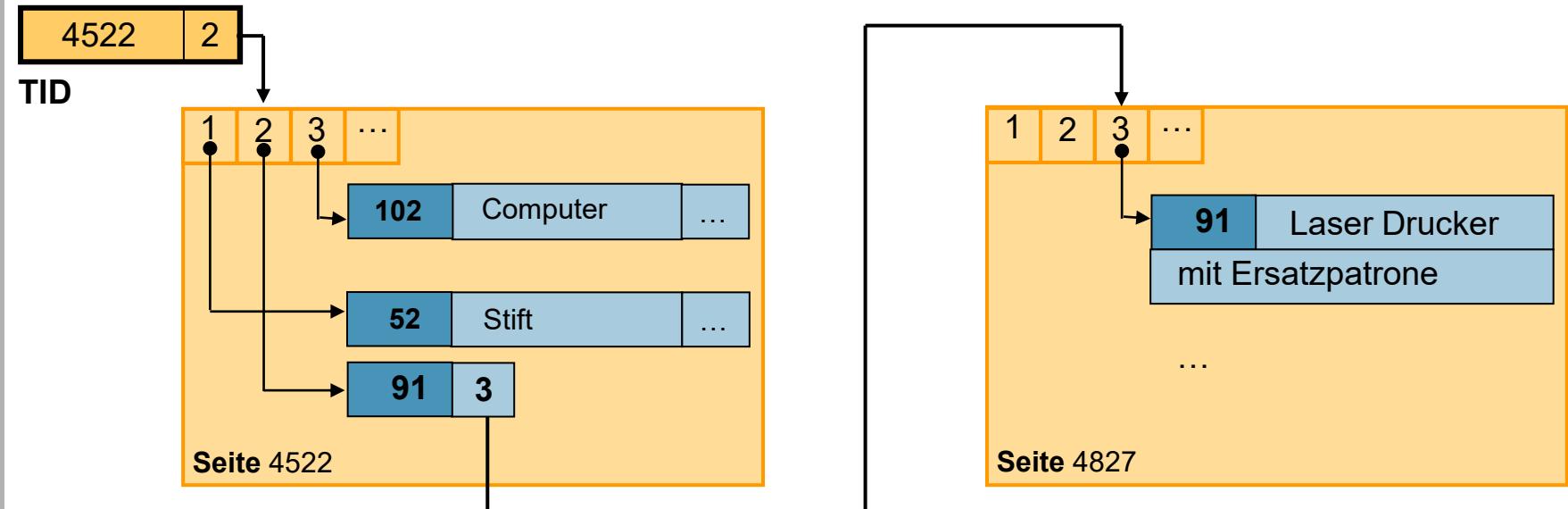


► Diese zusätzliche Indirektion ist nützlich, wenn die Seite intern reorganisiert werden muss.

Beispiel: Das Tupel zum Drucker-Artikel vergrößert sich, auf der Seite ist aber noch genug Platz vorhanden.
Daher bleiben auch alle Verweise auf dieses Tupel gültig.

Physische Adressierung von Tupeln in einer sortierten Datei

Verdrängen eines Tupels von einer Seite



Beispiel: Das Tupel zum Drucker-Artikel vergrößert sich und auf der Seite ist nicht genug Platz vorhanden. Es muss auf eine andere Seite transferiert werden. Um trotzdem die Verweise invariant zu halten, wird an der alten Position des Tupels eine Markierung hinterlassen, wo es jetzt zu finden ist. Das erfordert beim Lesen des Tupels (4522, 2) einen zusätzlichen Seitenzugriff, der vorher nicht notwendig war.

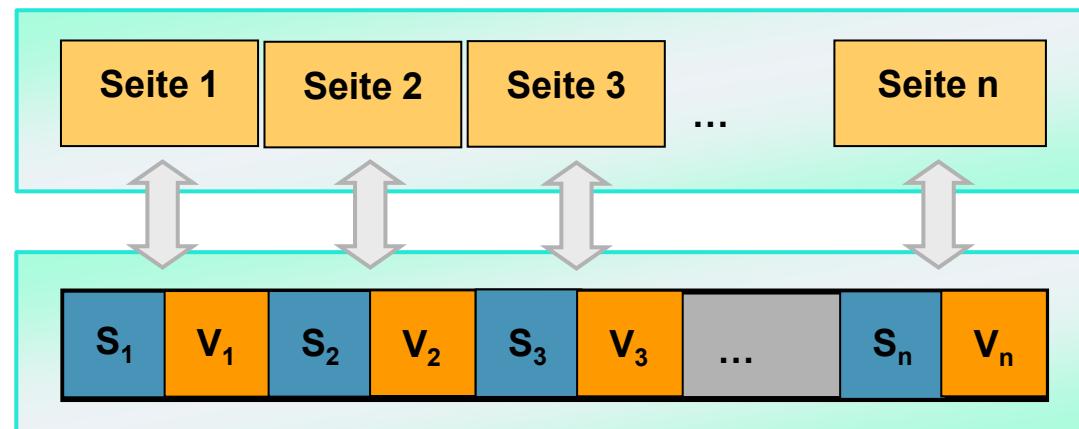
Bei nochmaliger Verdrängung dieses Tupels von der Seite 4827 würde aber kein weiterer Platzhalter eingefügt werden, sondern die Markierung auf der Heimatseite 4522 geändert. Die Länge einer solchen Verweiskette ist damit auf maximal zwei Stellen beschränkt.

Aufbau einer Index-Datei

Datensätze in Indexdatei:

- Zu jeder Seite der Hauptdatei wird genau ein Index – Datensatz in der Indexdatei abgelegt.

Datendatei

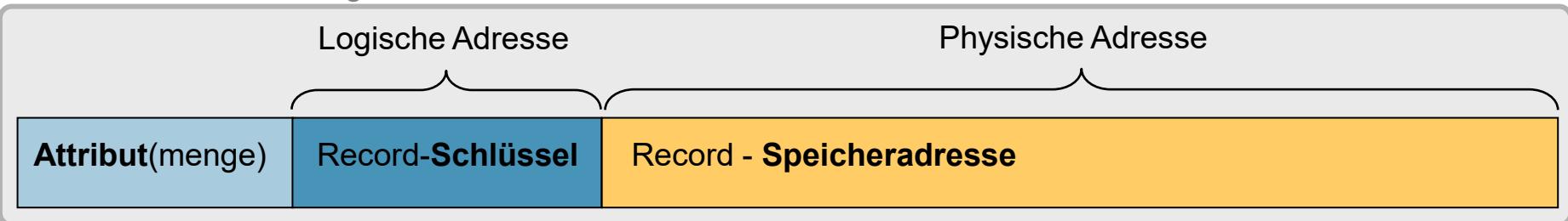


Indexdatei

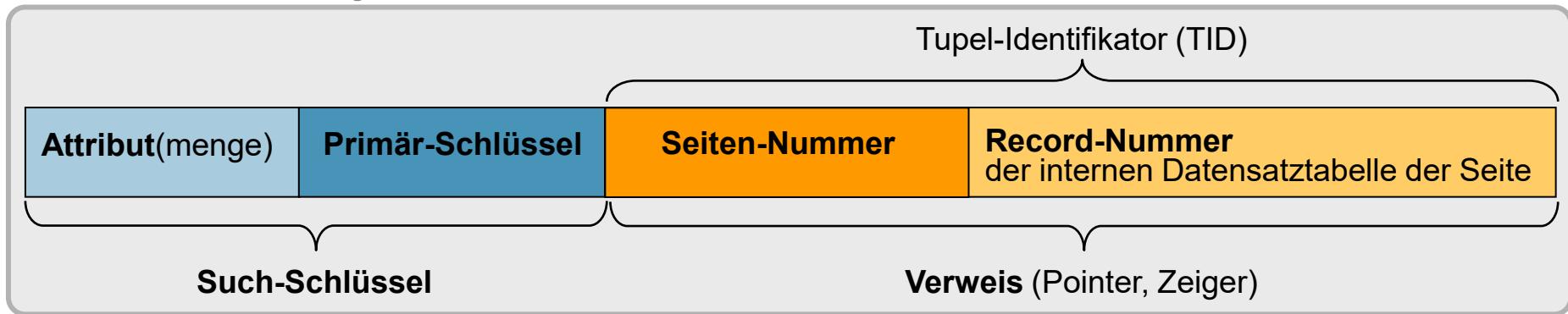
- Index – Datensätze bestehen abwechselnd aus Schlüsseln und Verweisen.

Aufbau einer Index-Datei

Vereinfachte Darstellung



Ausführliche Darstellung

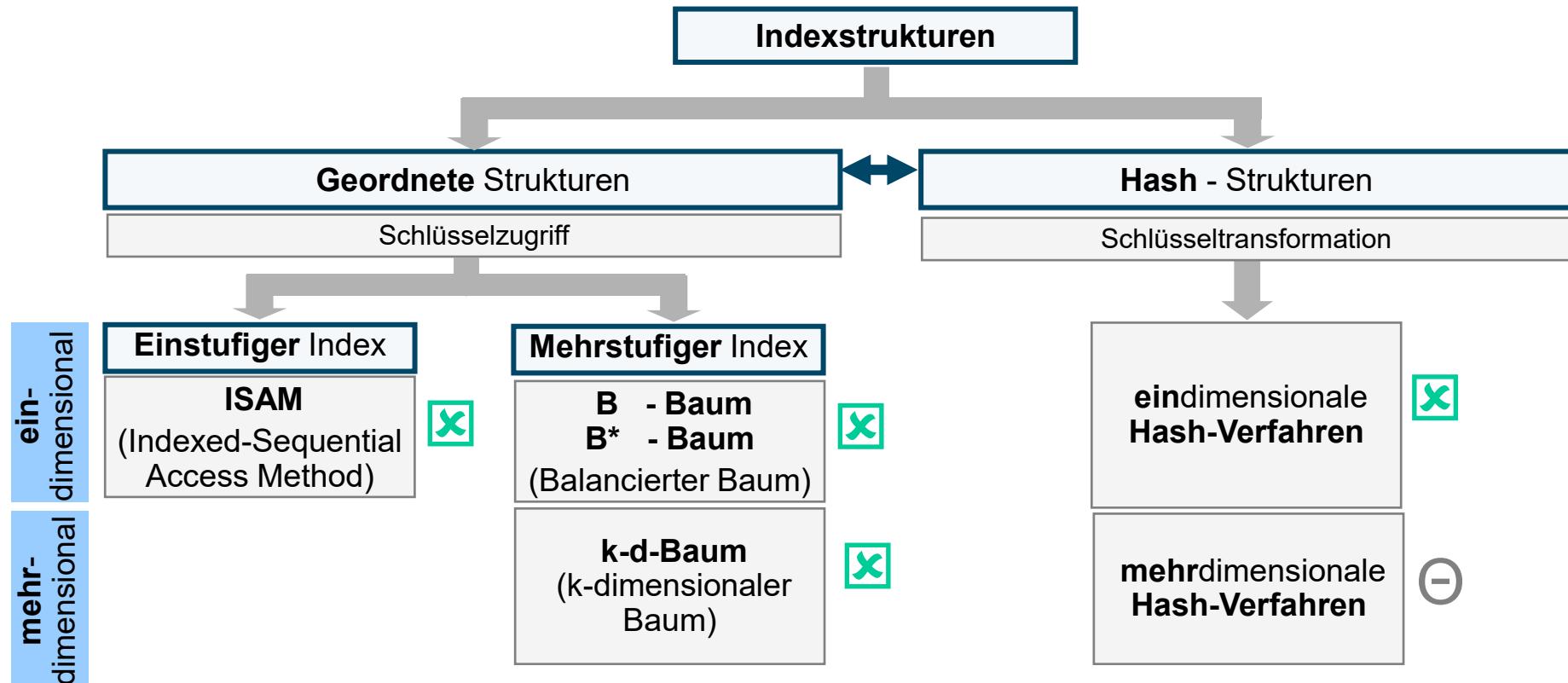


Stark vereinfachte Darstellung



Indexstrukturen – Effizienter Dateizugriff durch Indexierung

Ein Index-Mechanismus dient der Beschleunigung des Zugriffs auf gewünschte Daten dadurch, dass der Weg zum tatsächlichen Speicherungsort der Daten durch „**Abkürzungspointer**“ oder **Berechnungen** vereinfacht wird. Es werden zwei Klassen solcher Mechanismen unterschieden:



Schlüsselzugriff vs. Schlüsseltransformation

Schlüsselzugriff

- Geordnete Indexstrukturen basieren auf einer auf den **Such-Schlüsseln** existierenden Ordnung: Zuordnung von Primärschlüsseln (und weiterer Attribute) zu Speicher-Adressen der Records in Hilfsstruktur wie Indexdatei
- Bei jedem Zugriff wird ein bestimmter Pfad durchlaufen

Schlüsseltransformation

- Speicher-Adresse des Records wird direkt aus den Schlüsselwerten (Primärschlüsseln und evtl. weiterer Attribute) berechnet (Hash-Funktion/Formel/Verfahren) und Record unter so ermittelten Adresse gespeichert
- Speicher-Adresse (Hash-Wert) des Records wird zusammen mit den eigentlichen Daten (Werten) des Records abgelegt
- Seiten einer Hash-Datei sind gruppiert in Buckets. Records können in diesen Speicherbereichen „gestreut“ werden.
- Unterstellen eine Gleichverteilung der Schlüssel über einen „Schlüsselraum“

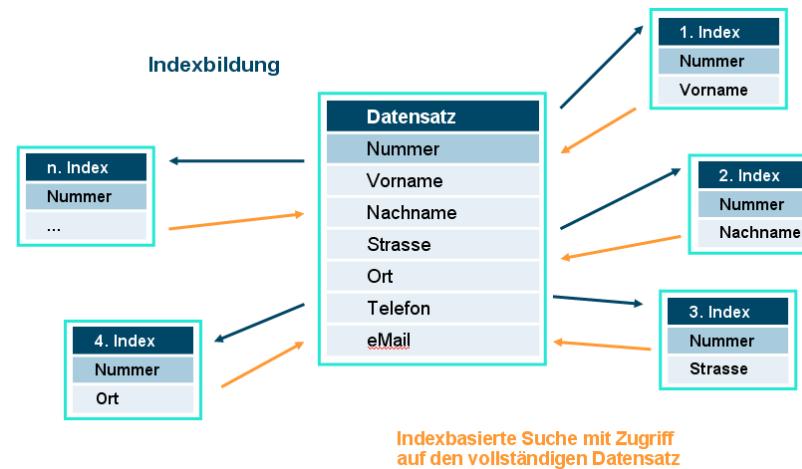
Primärindex vs. Sekundärindex

Primärindex

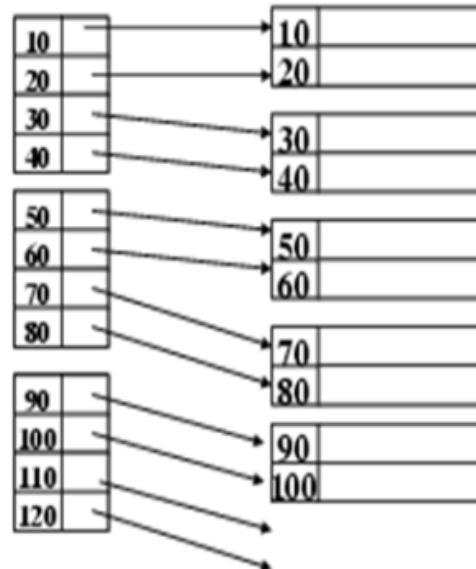
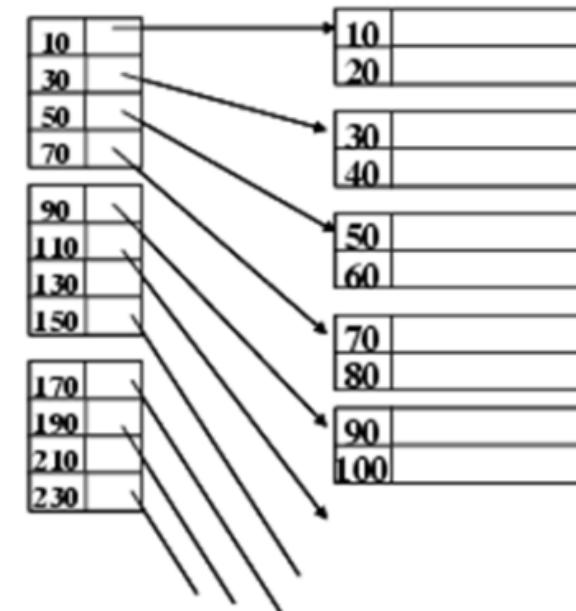
- Primärindex: Zugriffspfad auf interne Relation, der Dateiorganisationsform der internen Relation nutzen kann (Primärindex kann Sortierung nutzen)
- Primärindizes legen die physische Anordnung der indizierten Daten fest.
- In den meisten Fällen wird der Primärschlüssel, als identifizierendes Attribut (-menge) einer Relation, auch vom Primärindex indiziert.
- Für jede Datei kann es daher nur einen Primärindex geben.

Sekundärindex

- Sekundärindex: Jeder weitere Zugriffspfad auf interne Relation, der Dateiorganisationsform der internen Relation nicht nutzen kann
- Sekundärindizes indizieren beliebige Attributmenge.
- Für eine Datei kann es nur einen Primärindex, aber mehrere Sekundärindizes geben.



Dichtbesetzter vs. Dünnbesetzter Index

Dense Index**Sequential File****Sparse Index****Sequential File**

Geclusterter vs. nicht-geclusterter Index

- *geclusterter Index*: in der gleichen Form sortiert wie interne Relation
- Bsp.: interne Relation Studenten Matrikelnummern sortiert ⇒ Indexdatei über dem Attribut Matrikelnummer üblicherweise geclustert

Geclusterter vs. nicht-geclusterter Index

- *nicht-geclusterter Index*: anders organisiert als interne Relation
- Bsp.: über Studienfach ein Sekundärindex, Datei selbst nach Matrikelnummern sortiert
- Primärindex dünnbesetzt und geclustert sein
- jeder dünnbesetzte Index ist auch geclusterter Index, aber nicht umgekehrt
- Sekundärindex kann nur dichtbesetzter, nicht-geclusterter Index sein (auch: invertierte Datei)

Indexstruktur in SQL Server: B-Baum

In SQL Server sind Indizes in Form von B-Strukturen aufgebaut. Jede Seite in der B-Struktur eines Indexes wird als Indexknoten bezeichnet. Der oberste Knoten der B-Struktur wird als Stammknoten bezeichnet. Die Knoten auf der untersten Ebene des Indexes werden als Blattknoten bezeichnet. Alle anderen Indexebenen zwischen dem Stamm- und den Blattknoten werden zusammenfassend als Zwischenebenen bezeichnet. In einem gruppierten Index enthalten die Blattknoten die Datenseiten der zugrunde liegenden Tabelle. Die Stamm- und Zwischenebenenknoten enthalten Indexseiten, in denen Indexzeilen enthalten sind. Jede Indexzeile enthält einen Schlüsselwert und einen Zeiger auf eine Seite einer Zwischenebene in der B-Struktur oder auf eine Datenzeile in der Blattebene des Indexes. Die Seiten auf jeder Ebene des Indexes sind durch eine doppelt verknüpfte Liste miteinander verknüpft.

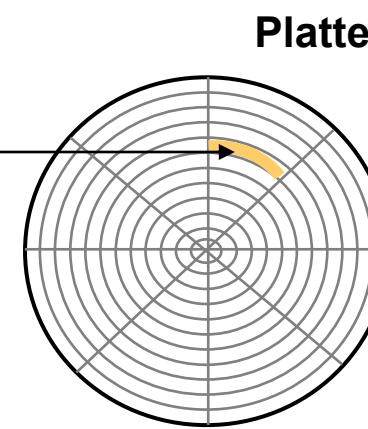
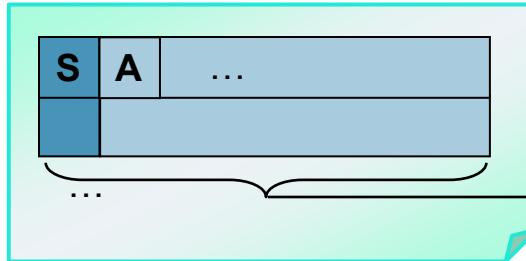
Gruppiert (B-Baum)

Nicht gruppiert (Heap)



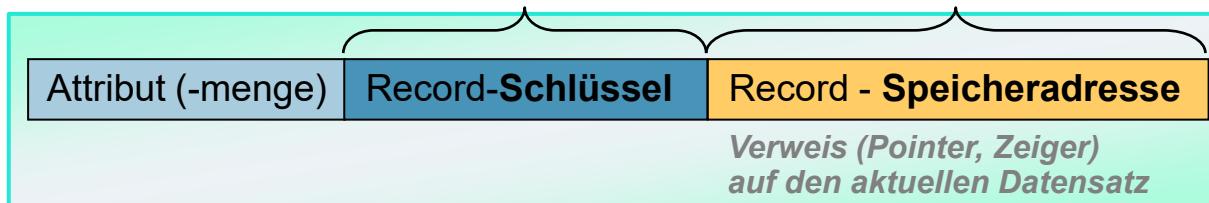
Indizierte Organisationsform: Datenbankindex

Datendatei (Hauptdatei)



Logische Adresse

Physische Adresse

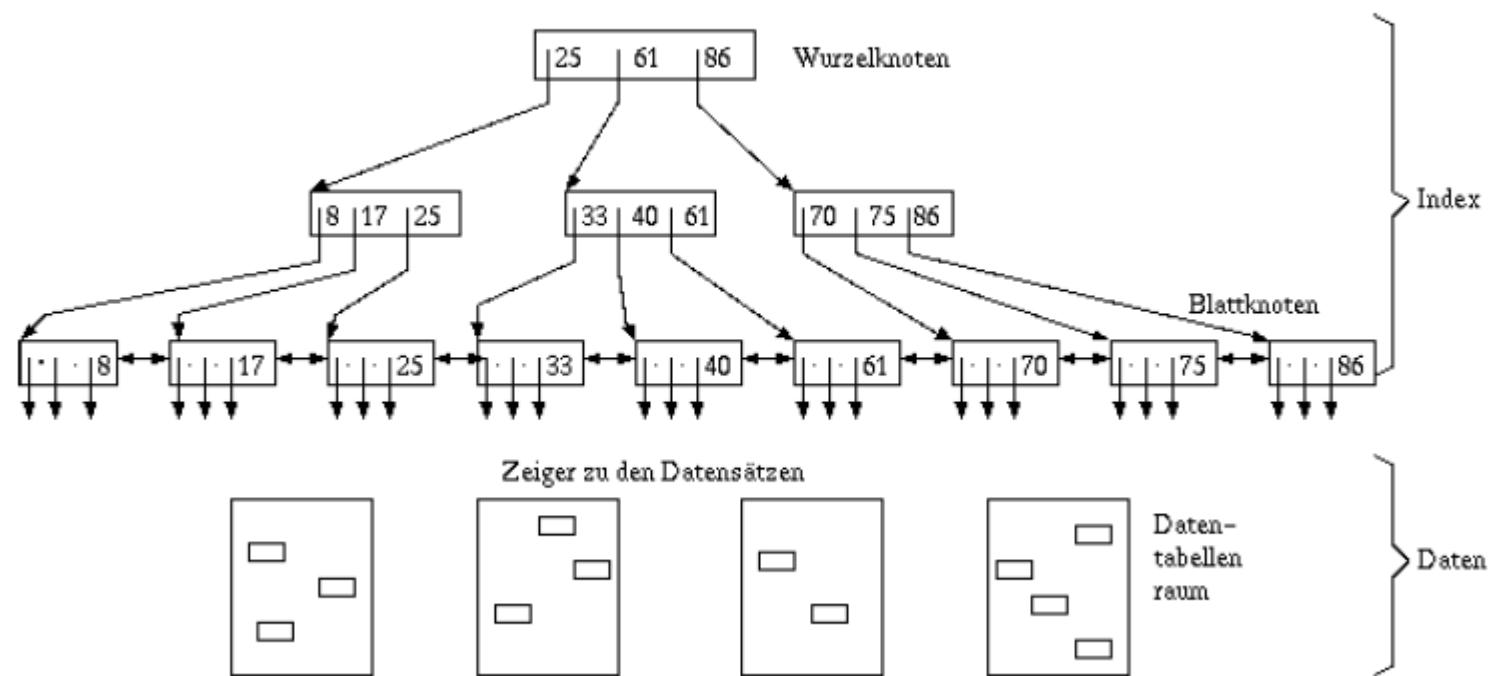


Indexdatei pro Attribut (-menge)

Vor- und Nachteile

Aber die Verbesserung des Zugriffs bekommt man nicht geschenkt:

Wie alle anderen Informationen müssen auch Indices gewartet werden und benötigen einen gewissen Platz.
→ Untersuchen wann das Anlegen eines Indexes vorteilhaft ist.

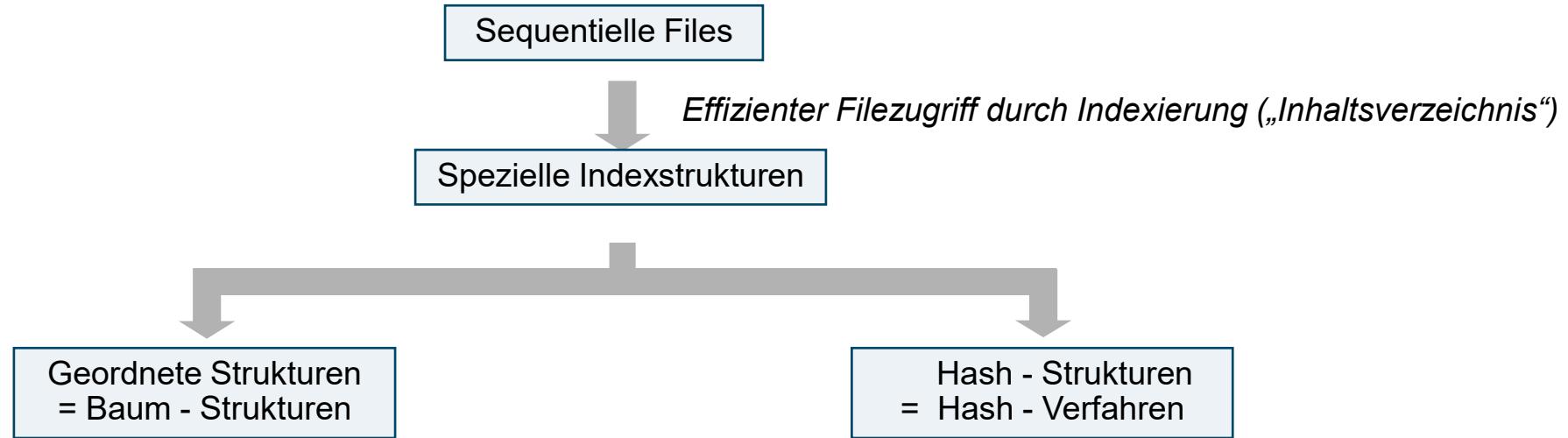


Beispiele für Klassifikationen

- Beispiel für dynamisches und eindimensionales Verfahren: B-Baum (beliebtester Zugriffspfad in relationalen Datenbanksystemen)
- wird mit create index meistens angelegt
- nur exact-match, kein partial-match
- Beispiel für statisches und eindimensionales Verfahren: klassisches Hashverfahren

Indexstrukturen

► Entwicklung diverserer Speicherungs- / Organisations-Formen für Files:



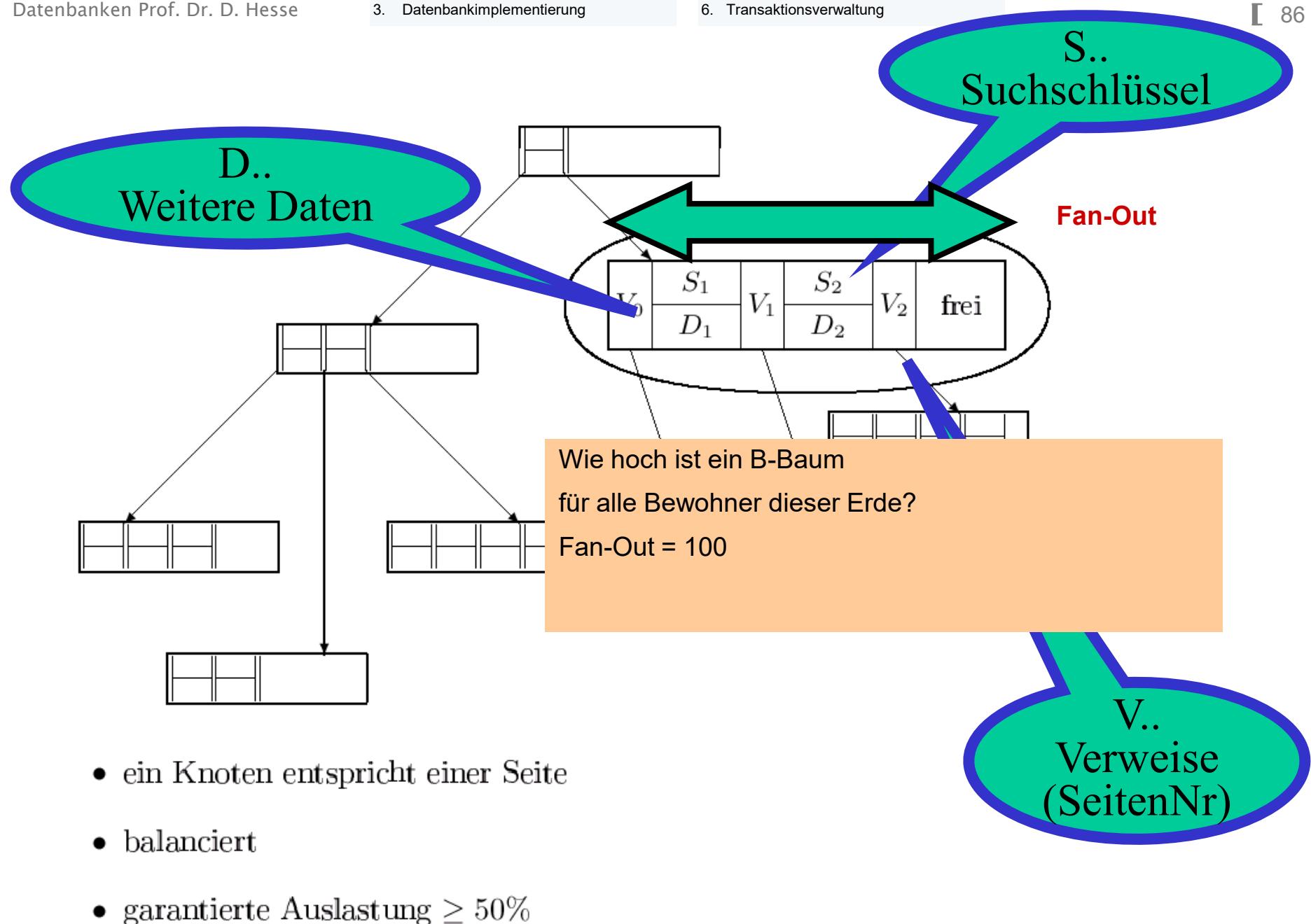
Wichtige Unterscheidungsmerkmale:

- ✓ Arten des Zugriffs
- ✓ Zugriffszeit
- ✓ Aufwand für Einfügen und Löschen
- ✓ Platzbedarf

Indexstrukturen – Effizienter Dateizugriff durch Indexierung

B-Bäume

Balancierte Mehrwege-
Suchbäume
für den Hintergrundspeicher

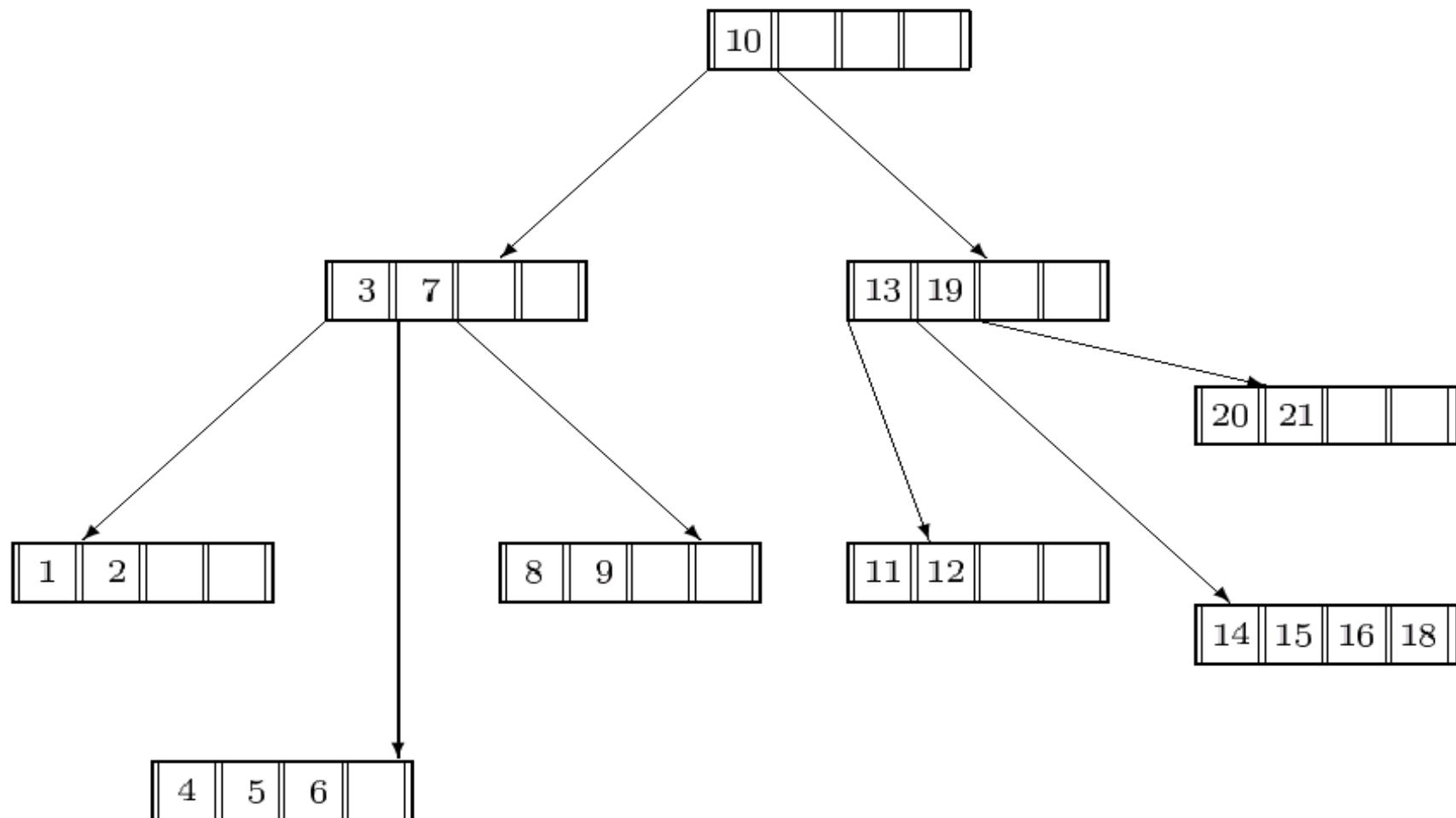


k=minimaler Fan-Out

B-Baum von Grad k :

1. Jeder Weg von der Wurzel zu einem Blatt hat die gleiche Länge.
2. Jeder Knoten außer der Wurzel hat mindestens k und höchstens $2k$ Einträge. Die Wurzel hat höchstens $2k$ Einträge. Die Einträge werden in allen Knoten sortiert gehalten.
3. Alle Knoten mit n Einträgen, außer den Blättern, haben $n + 1$ Kinder.
4. Seien S_1, \dots, S_n die Schlüssel eines Knotens mit $n + 1$ Kindern. V_0, V_1, \dots, V_n seien die Verweise auf diese Kinder. Dann gilt:
 - (a) V_0 weist auf den Teilbaum mit Schlüsseln kleiner als S_1 .
 - (b) V_i ($i = 1, \dots, n - 1$) weist auf den Teilbaum, dessen Schlüssel zwischen S_i und S_{i+1} liegen.
 - (c) V_n weist auf den Teilbaum mit Schlüsseln größer als S_n .
 - (d) In den Blattknoten sind die Zeiger nicht definiert.

Ein Beispielbaum

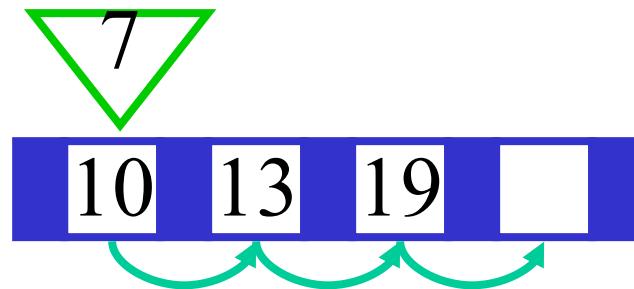


Einfügen eines neuen Objekts (Datensatz) in einen B-Baum

1. Führe eine Suche nach dem Schlüssel durch; diese endet (scheitert) an der Einfügestelle.
2. Füge den Schlüssel dort ein.
3. Ist der Knoten überfüllt, teile ihn
 - Lege einen neuen Knoten an und belege ihn mit den Schlüsseln, die rechts vom mittleren Eintrag des überfüllten Knotens liegen.
 - Füge den mittleren Eintrag im Vaterknoten des überfüllten Knotens ein.
 - Verbinde den Verweis rechts des neuen Eintrags im Vaterknoten mit dem neuen Knoten
4. Ist der Vaterknoten jetzt überfüllt?
 - Handelt es sich um die Wurzel, so lege eine neue Wurzel an.
 - Wiederhole Schritt 3 mit dem Vaterknoten.

- | | | |
|-----------------------------|--------------------------------|--|
| 1. Einführung | 4. Physische Datenorganisation | 7. Datensicherheit und Wiederherstellung |
| 2. Datenbankentwurf | 5. Anfrageoptimierung | 8. Business Intelligence |
| 3. Datenbankimplementierung | 6. Transaktionsverwaltung | |

[90]

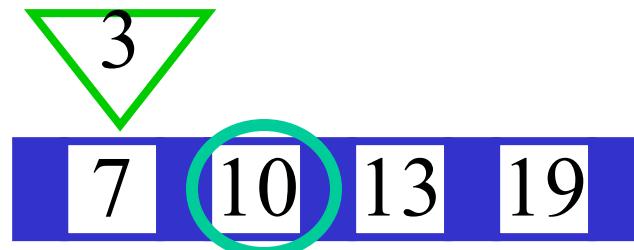


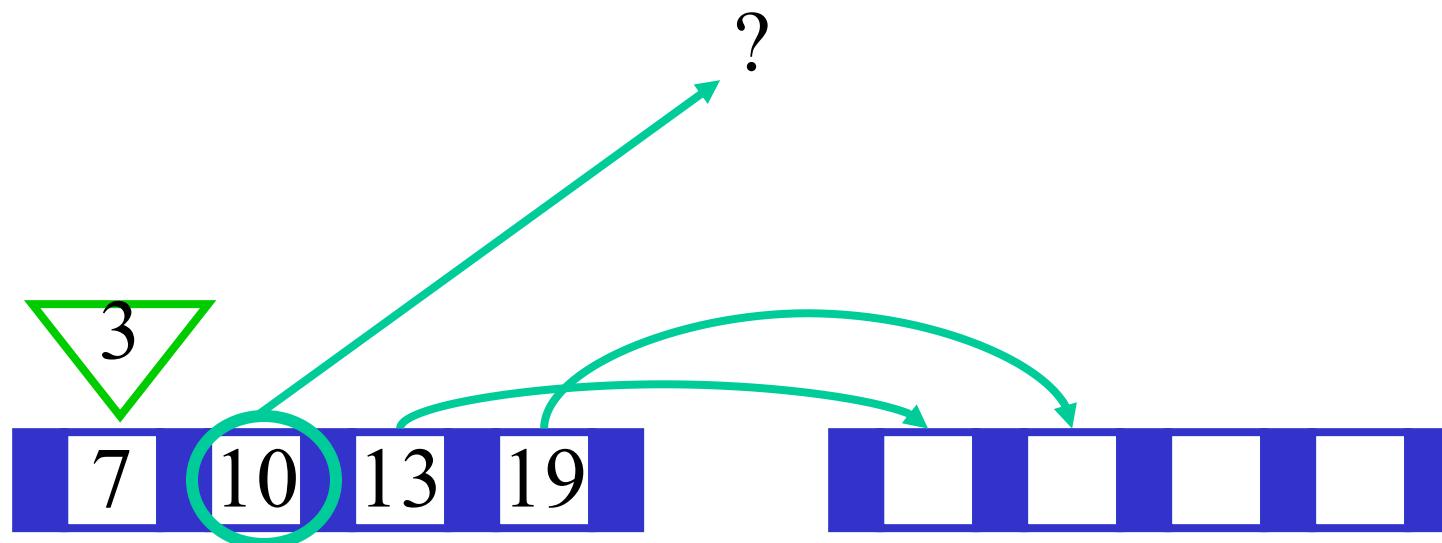
Exkurs:

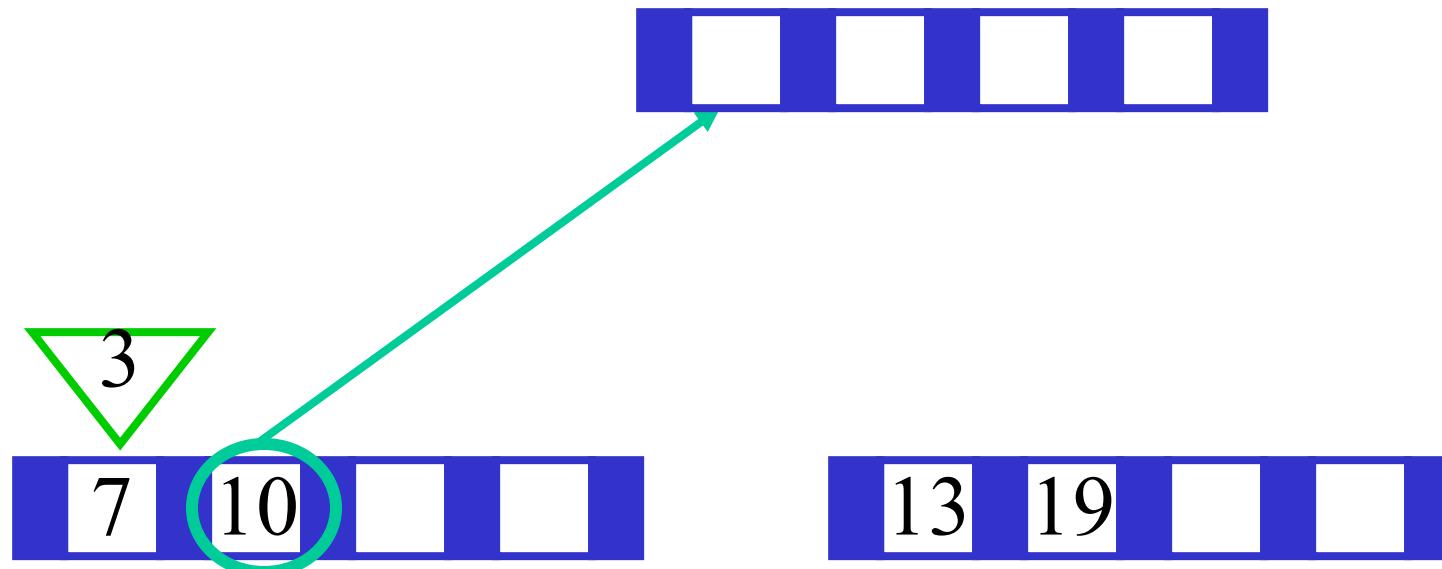
Welches Verfahren wenden Sie zur Suche in einer sortierten Liste an?

Antwort: Sprungsuche (Binärsuche)

$$\log_2 2k$$



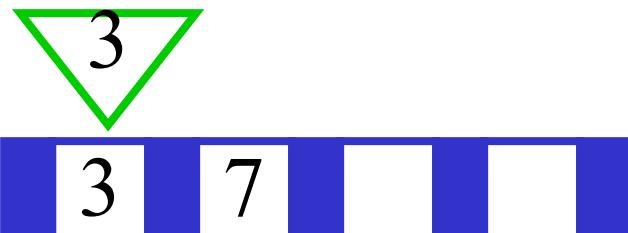


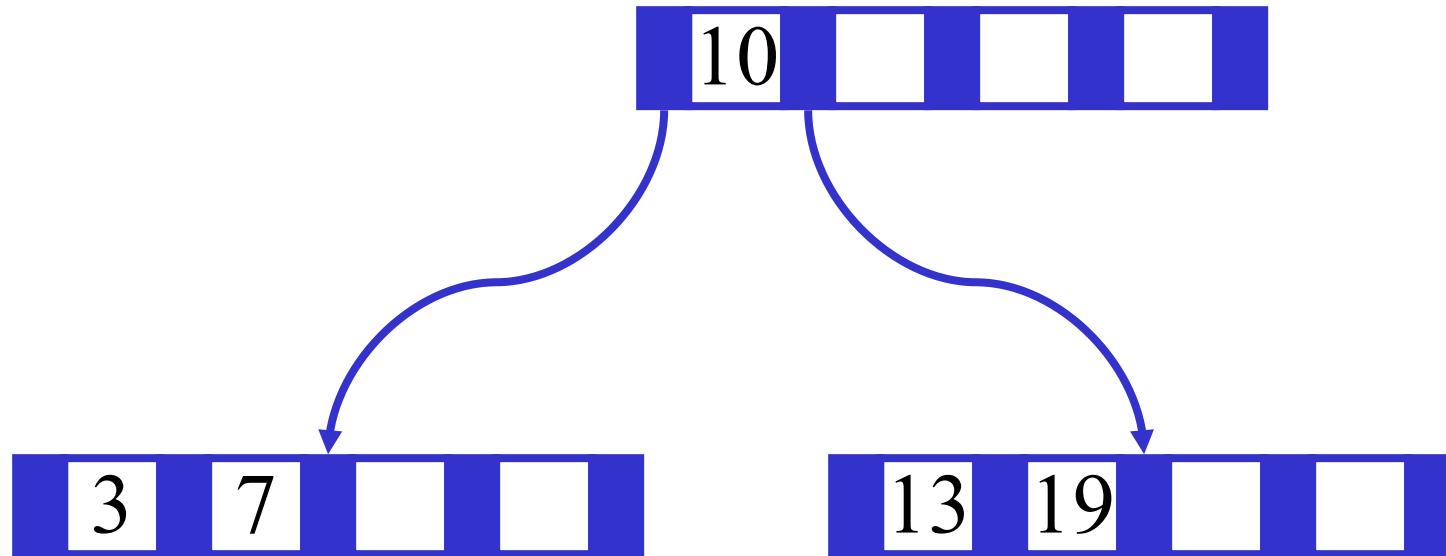


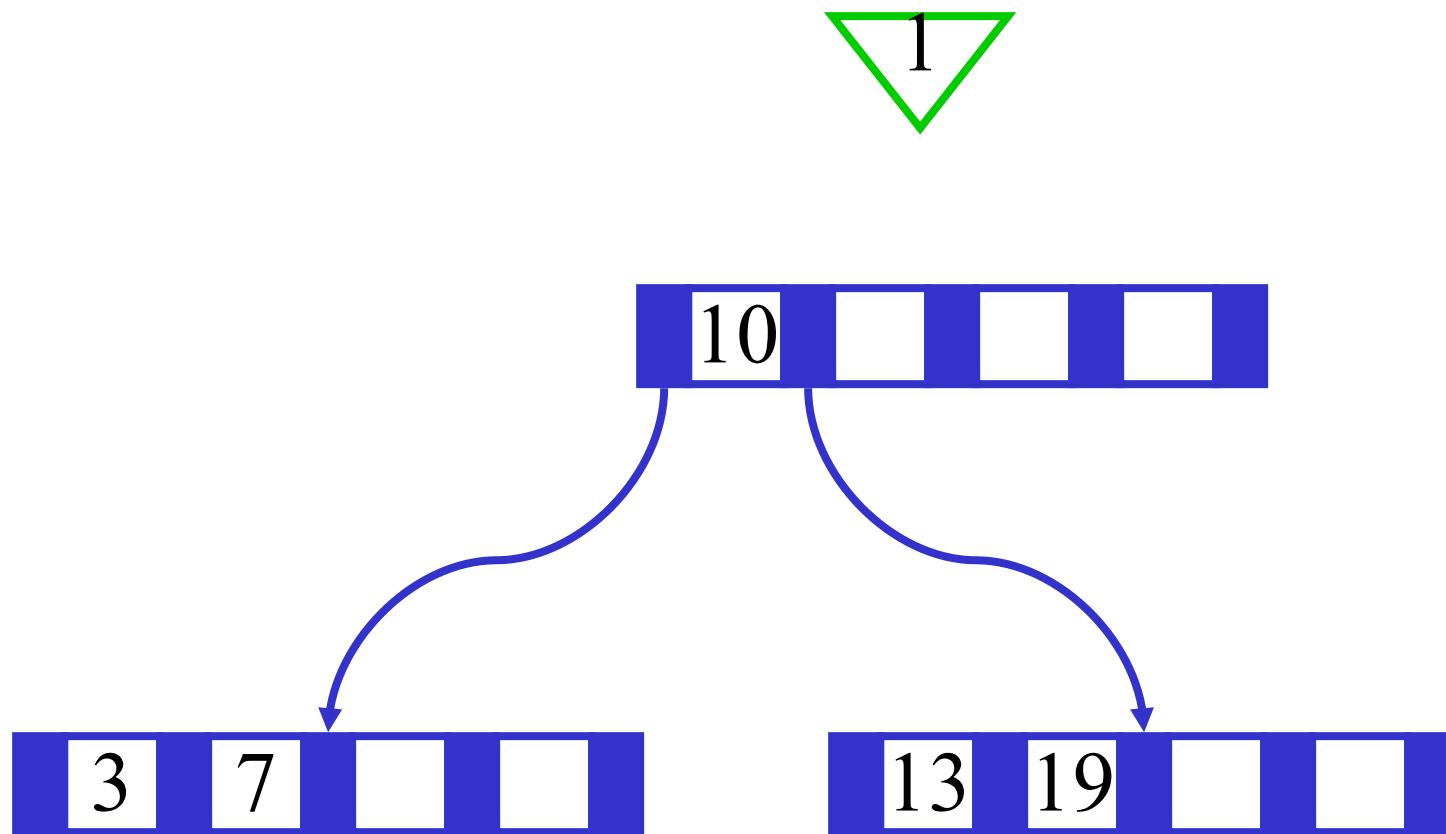
- | | | |
|-----------------------------|--------------------------------|--|
| 1. Einführung | 4. Physische Datenorganisation | 7. Datensicherheit und Wiederherstellung |
| 2. Datenbankentwurf | 5. Anfrageoptimierung | 8. Business Intelligence |
| 3. Datenbankimplementierung | 6. Transaktionsverwaltung | |

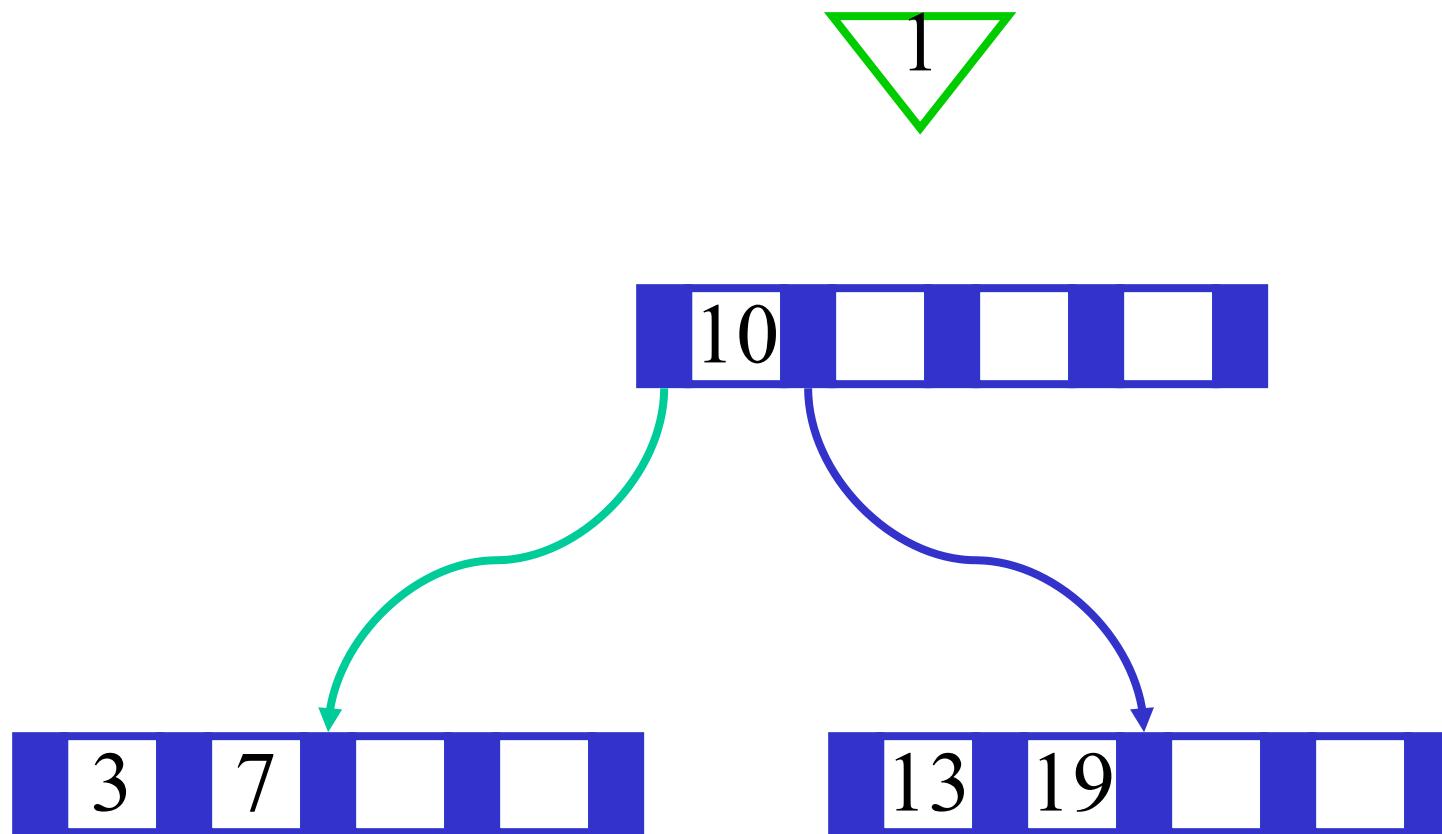
[94]

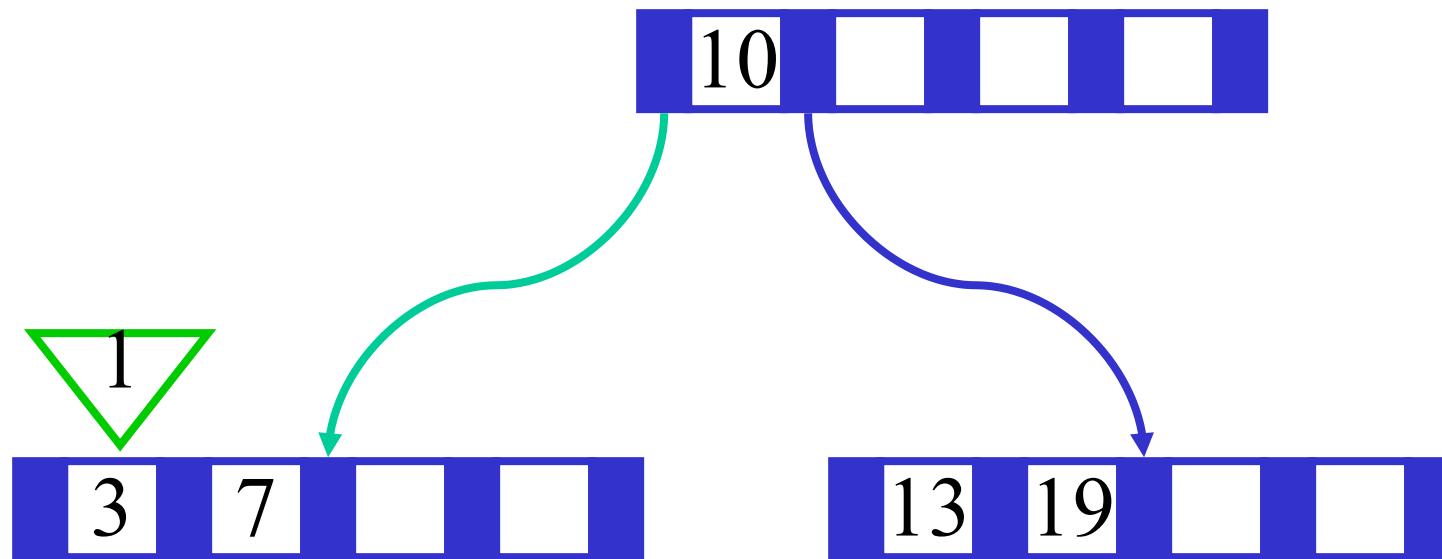
10

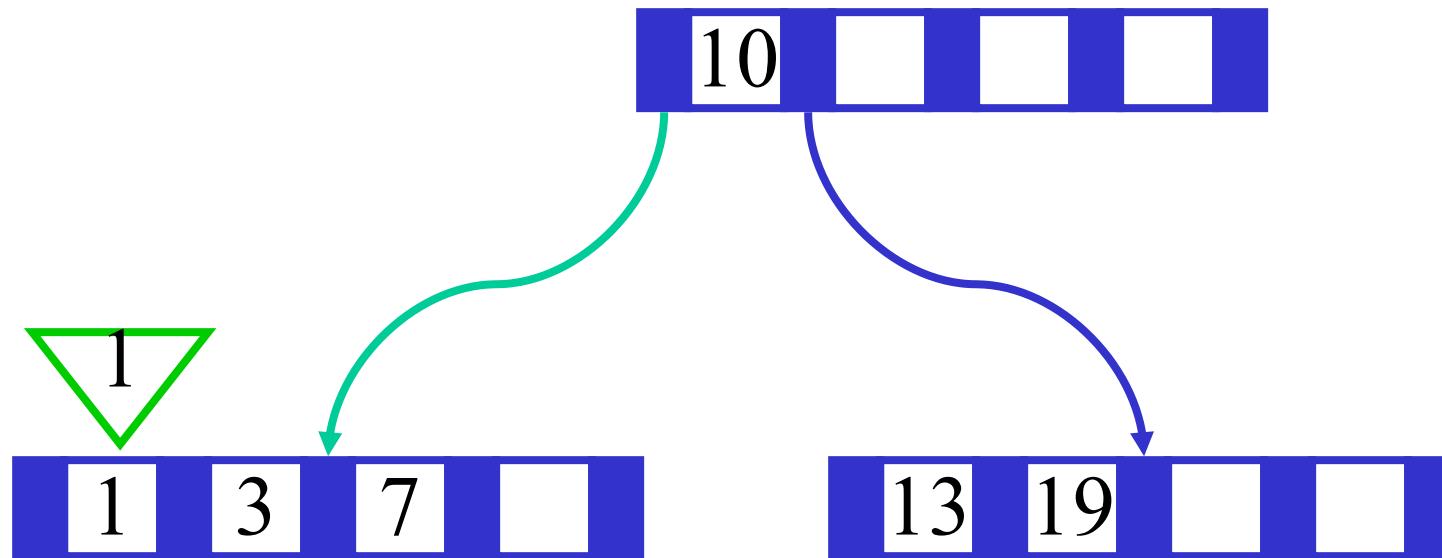


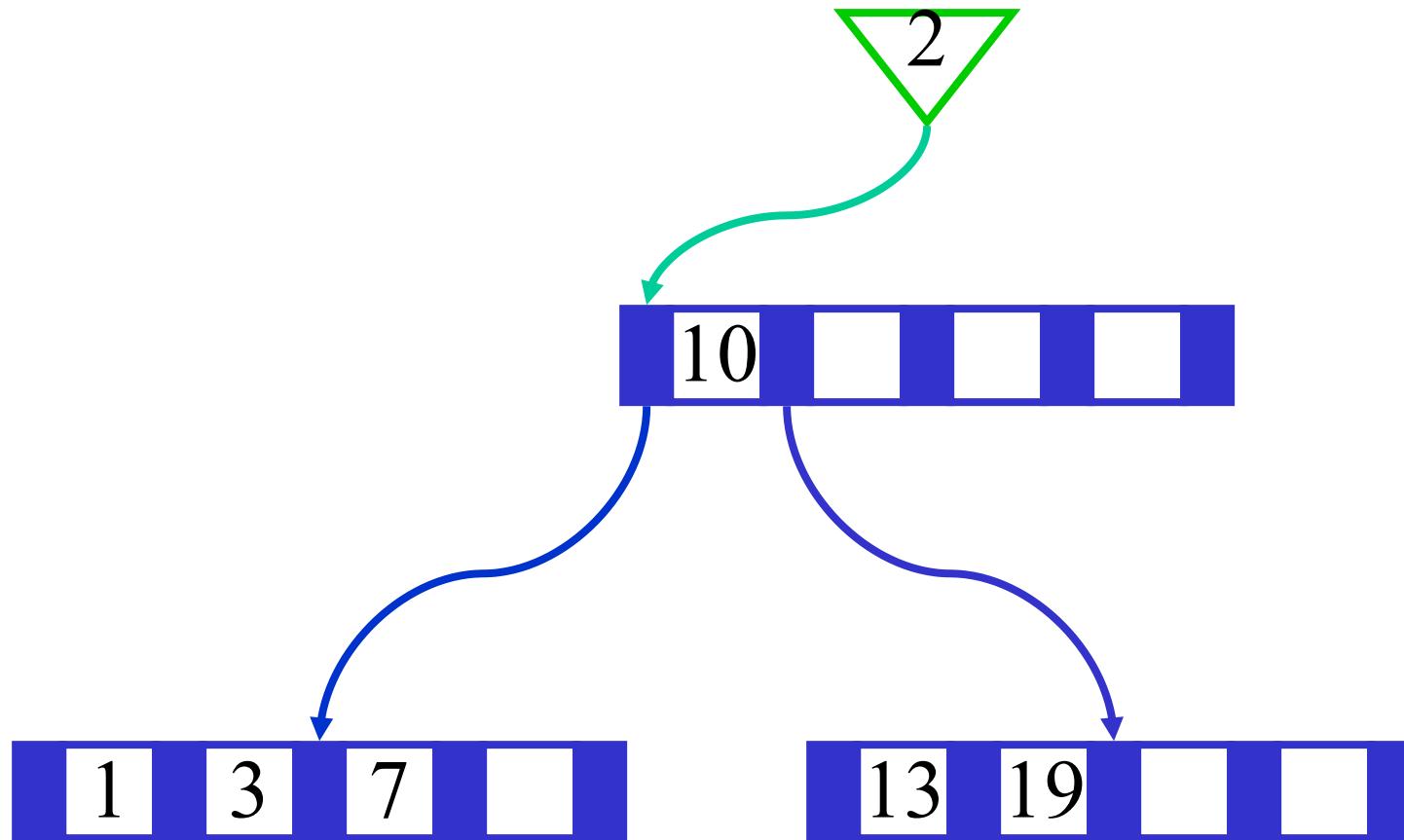


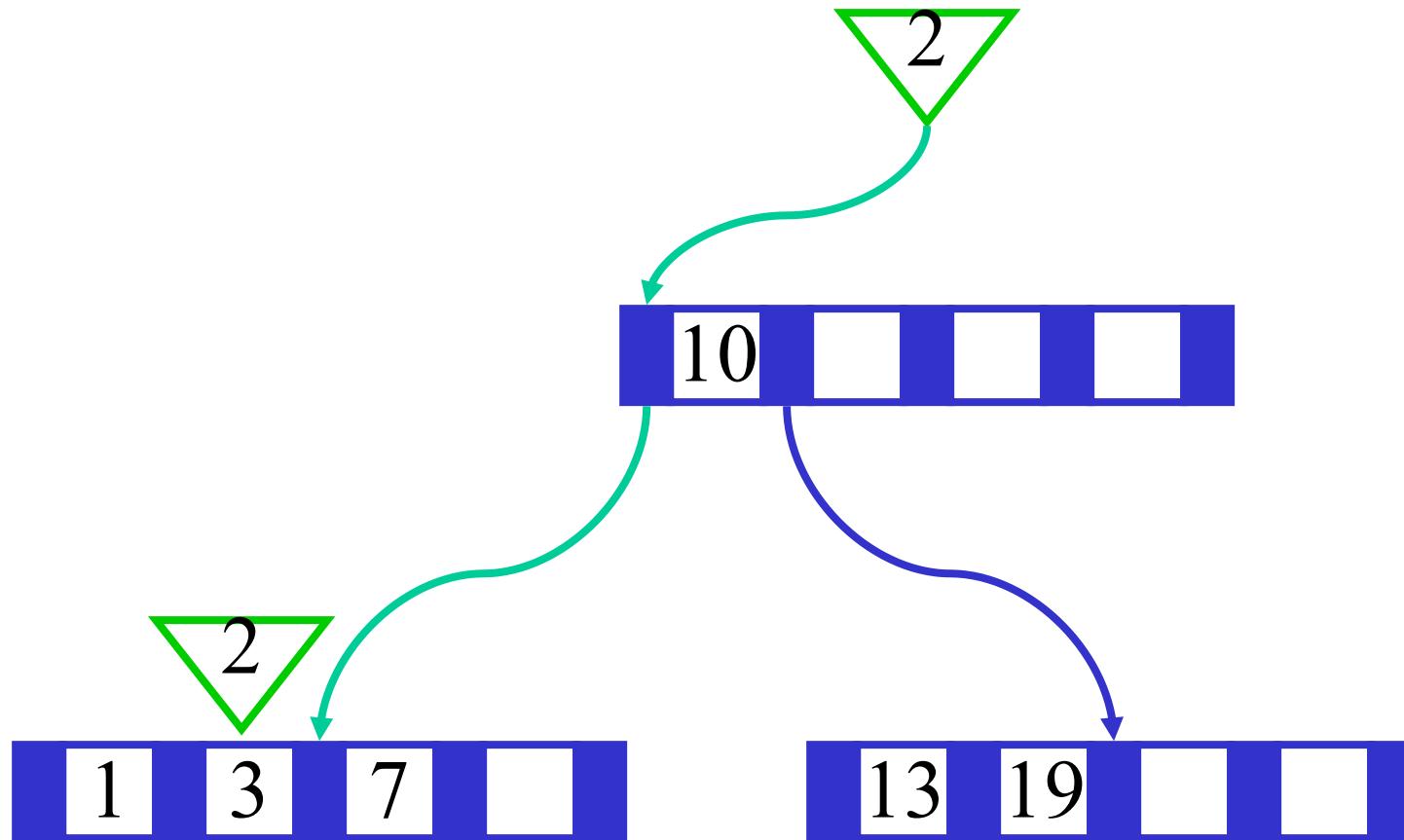


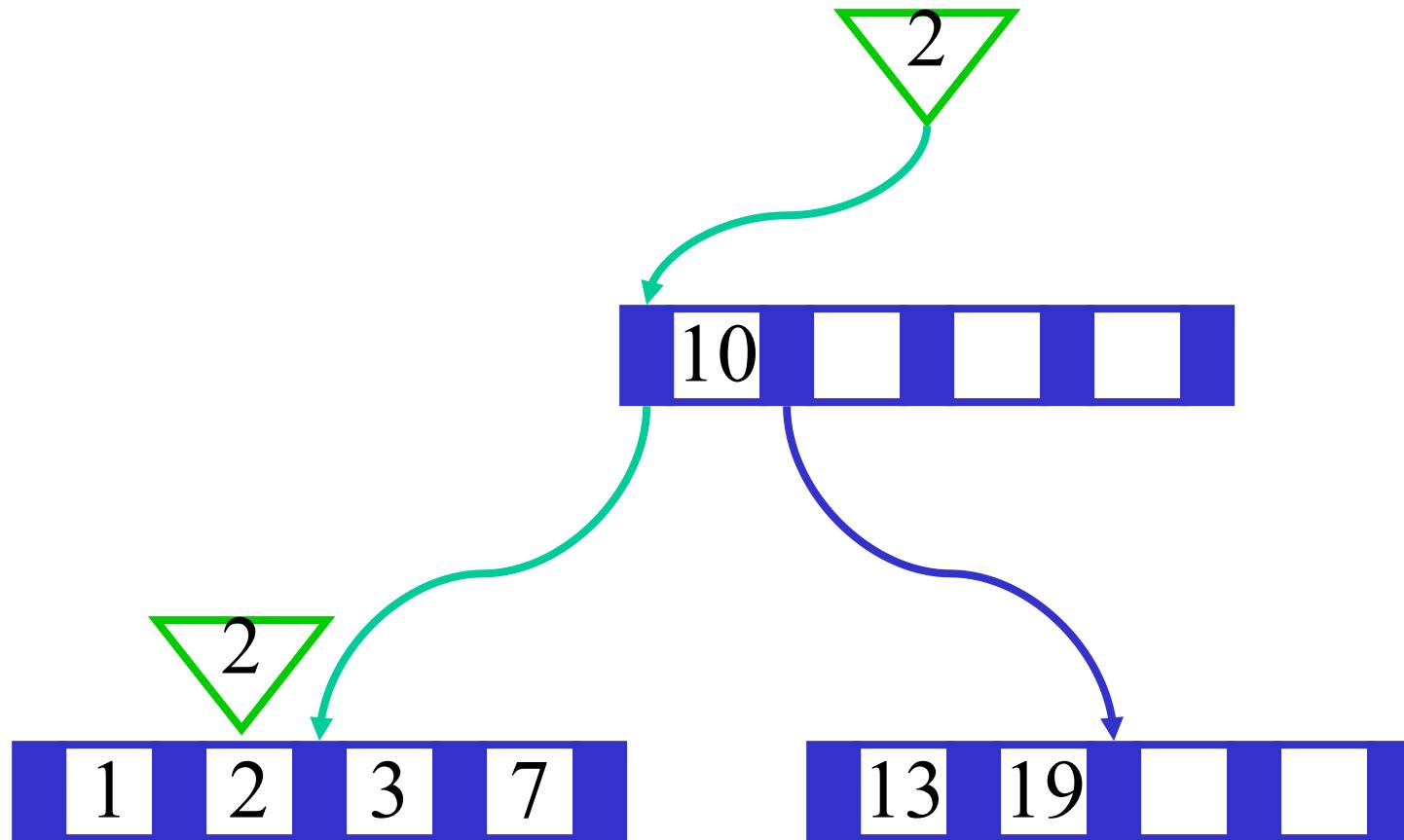


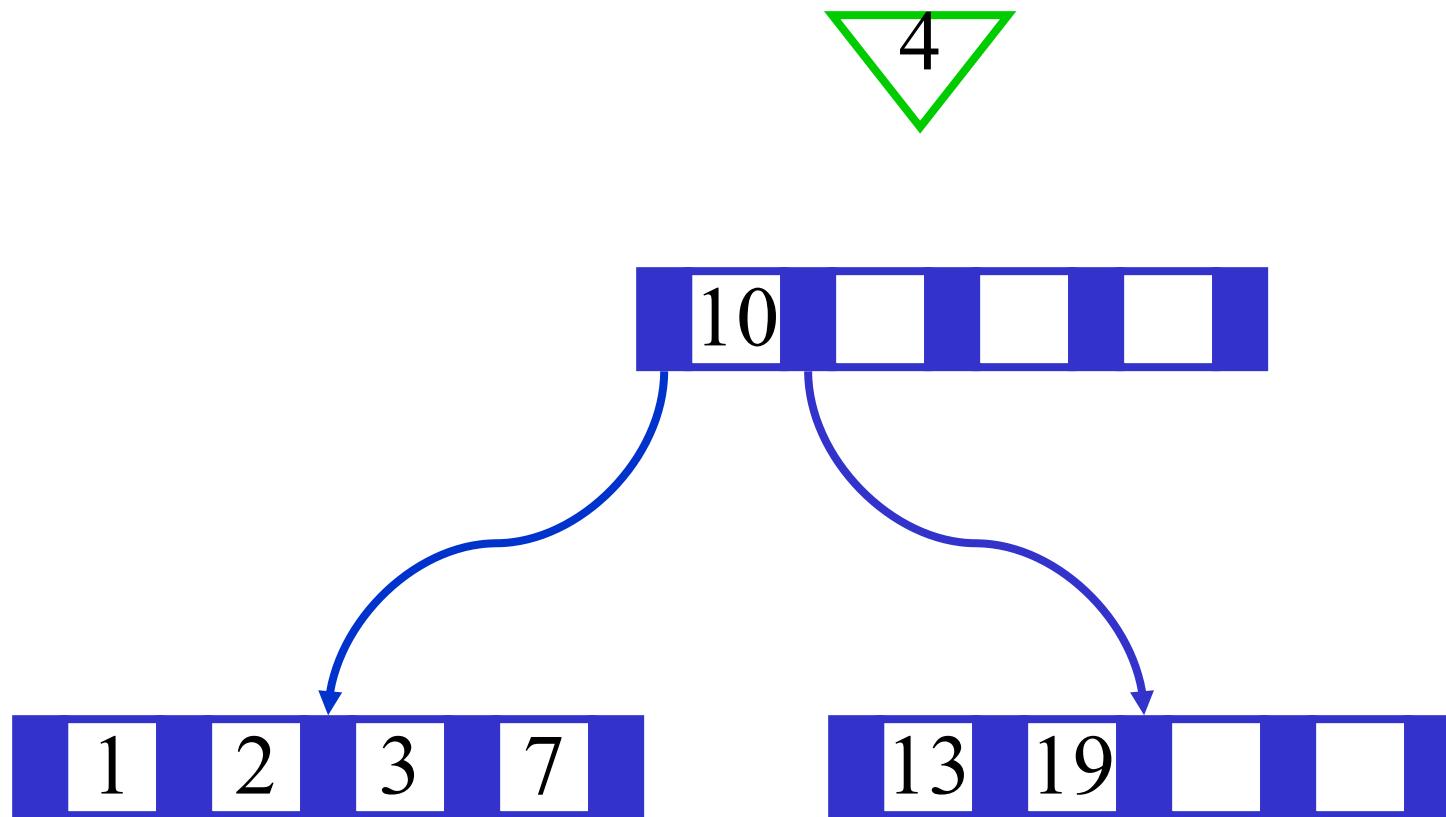


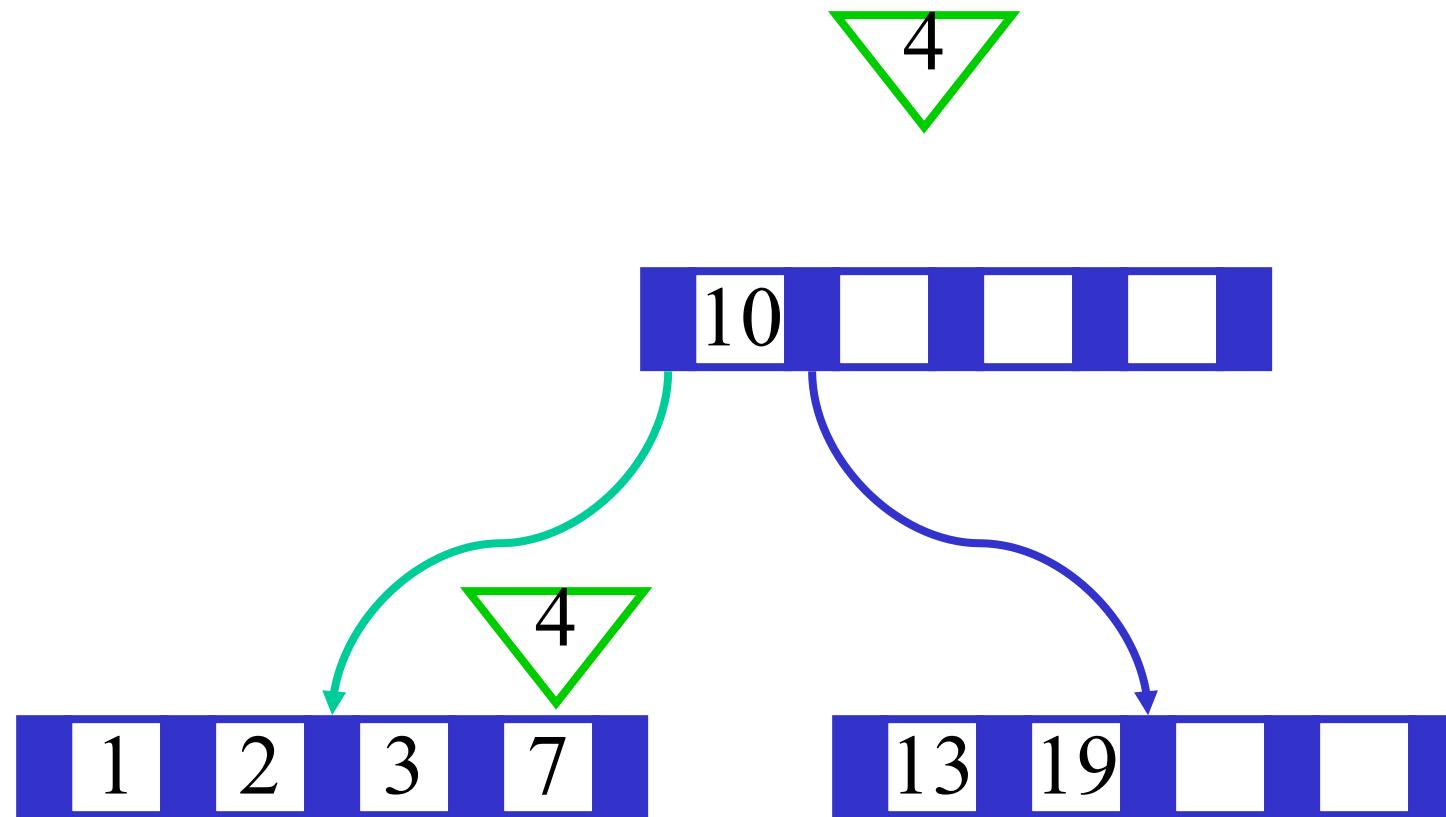


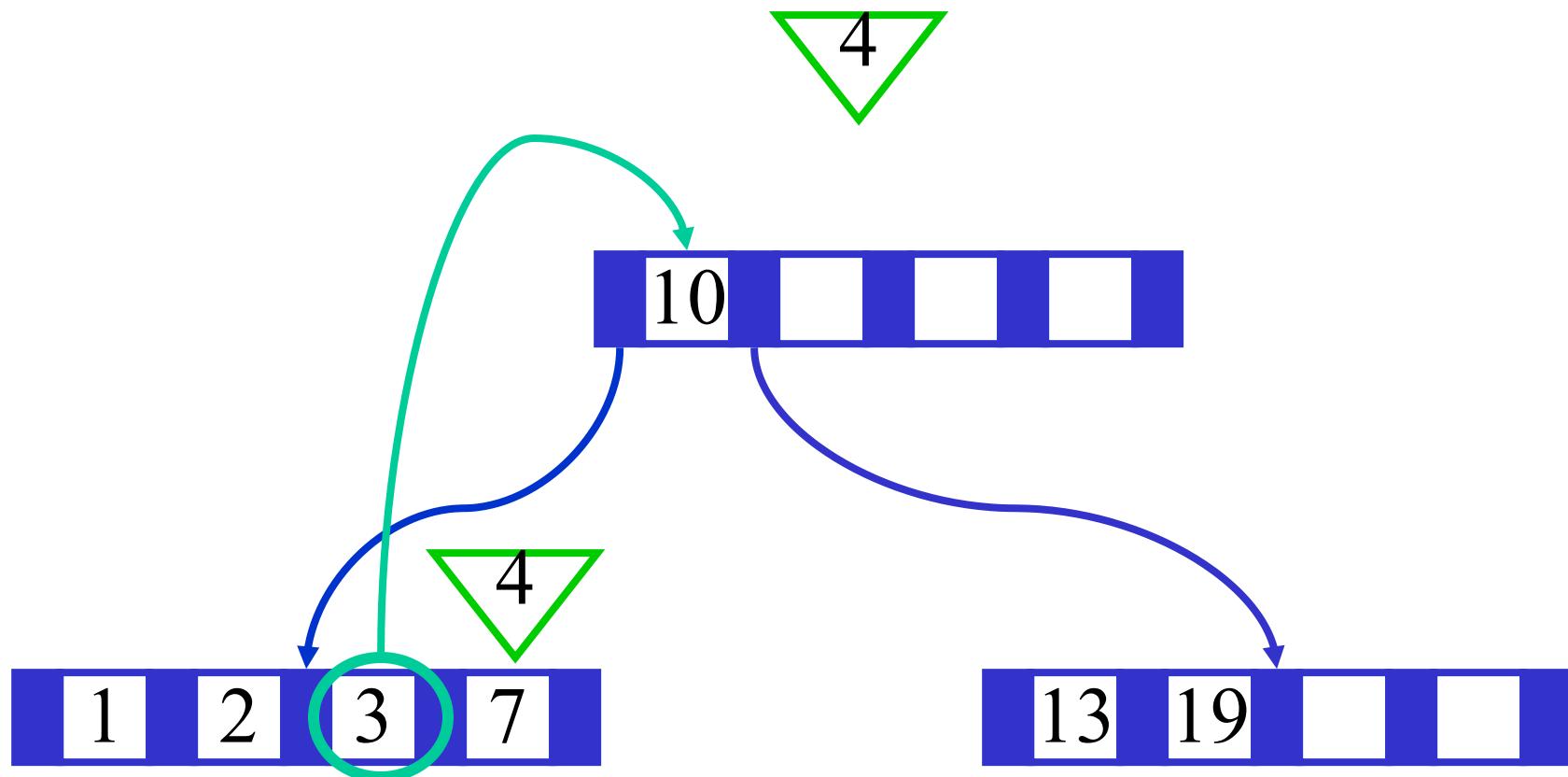


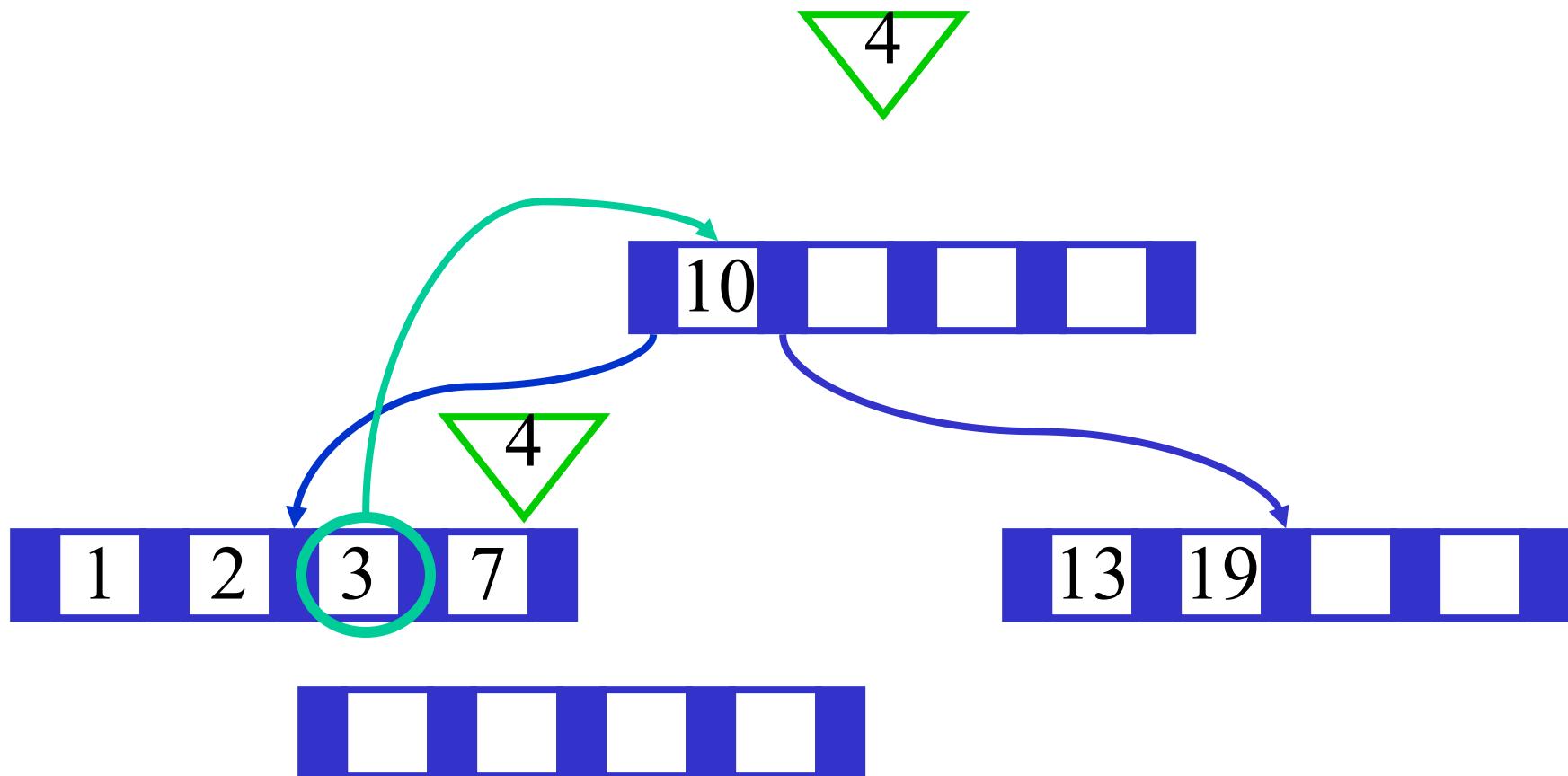


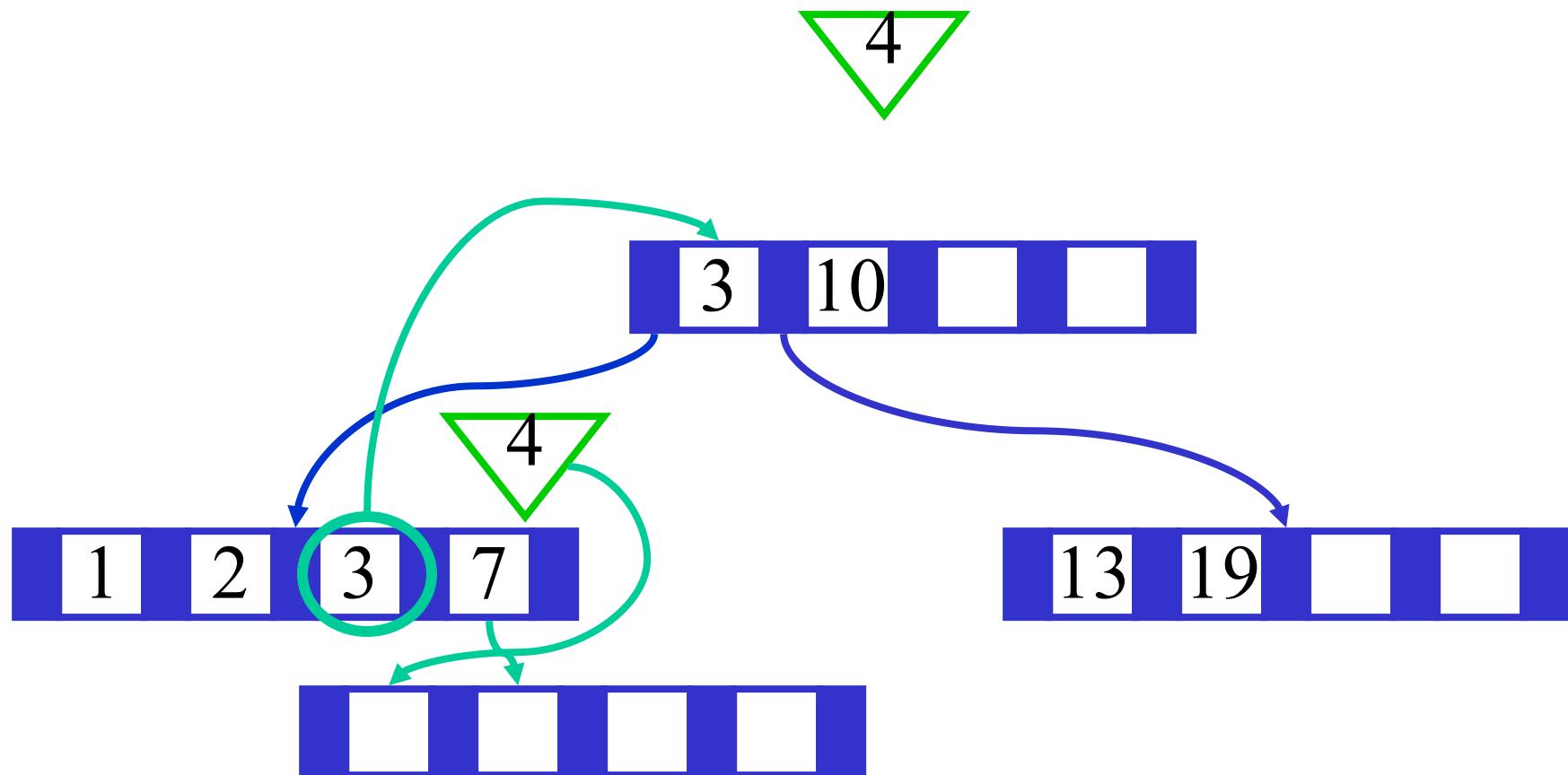


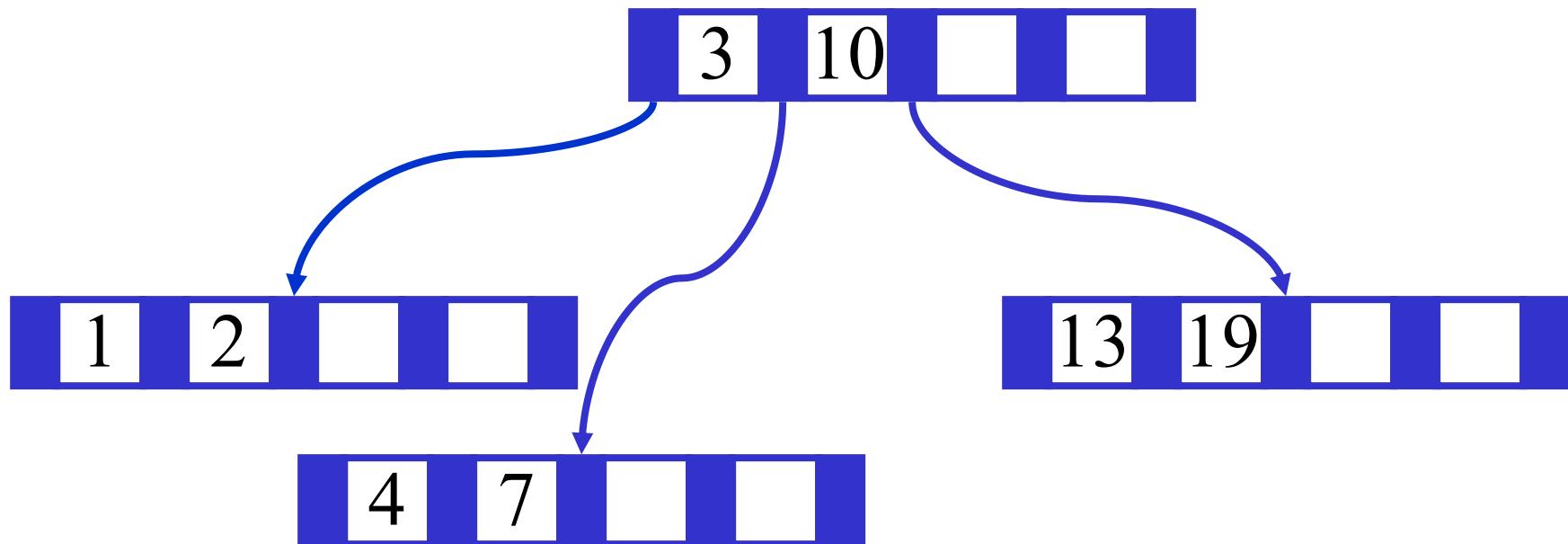


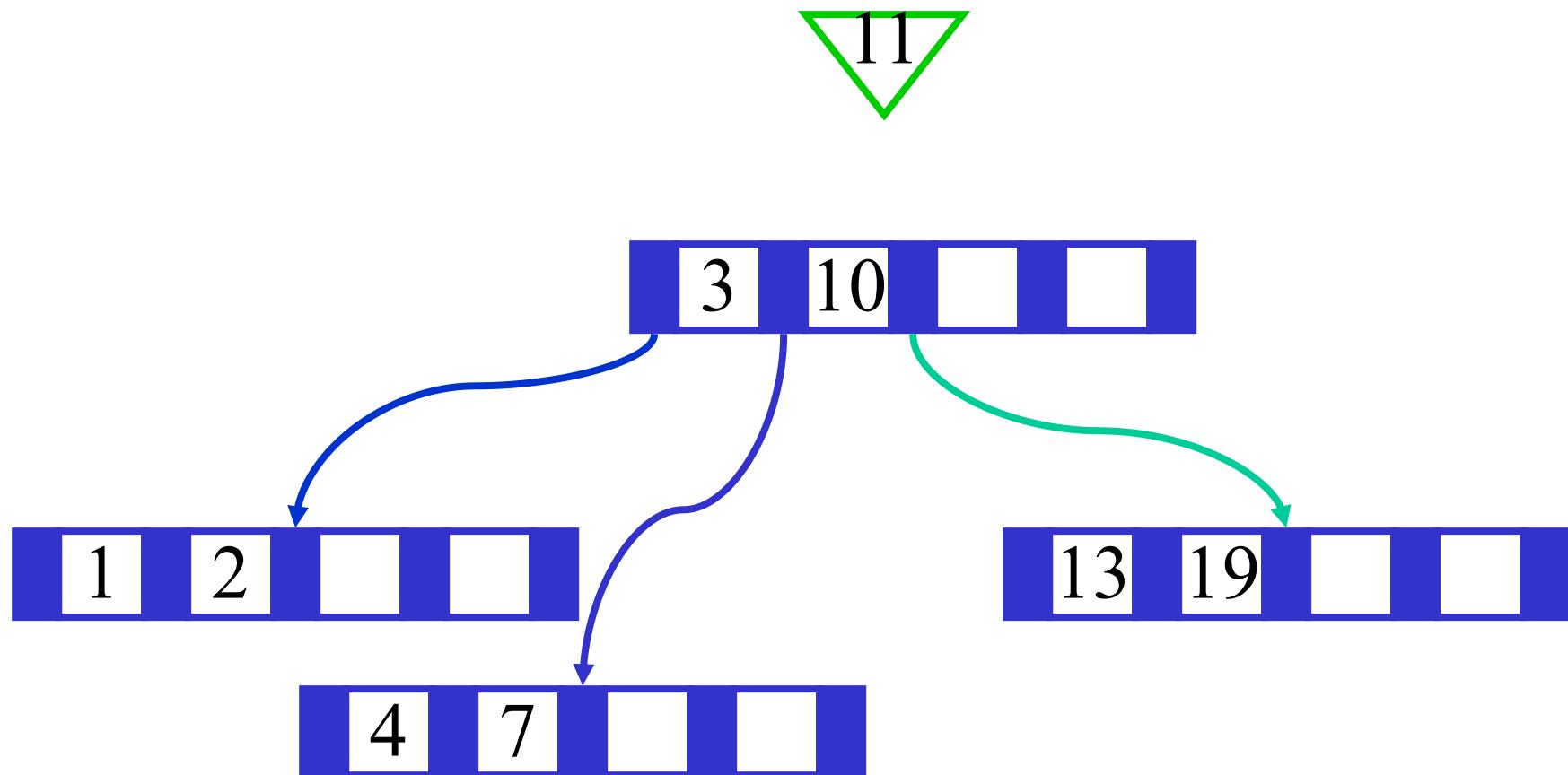


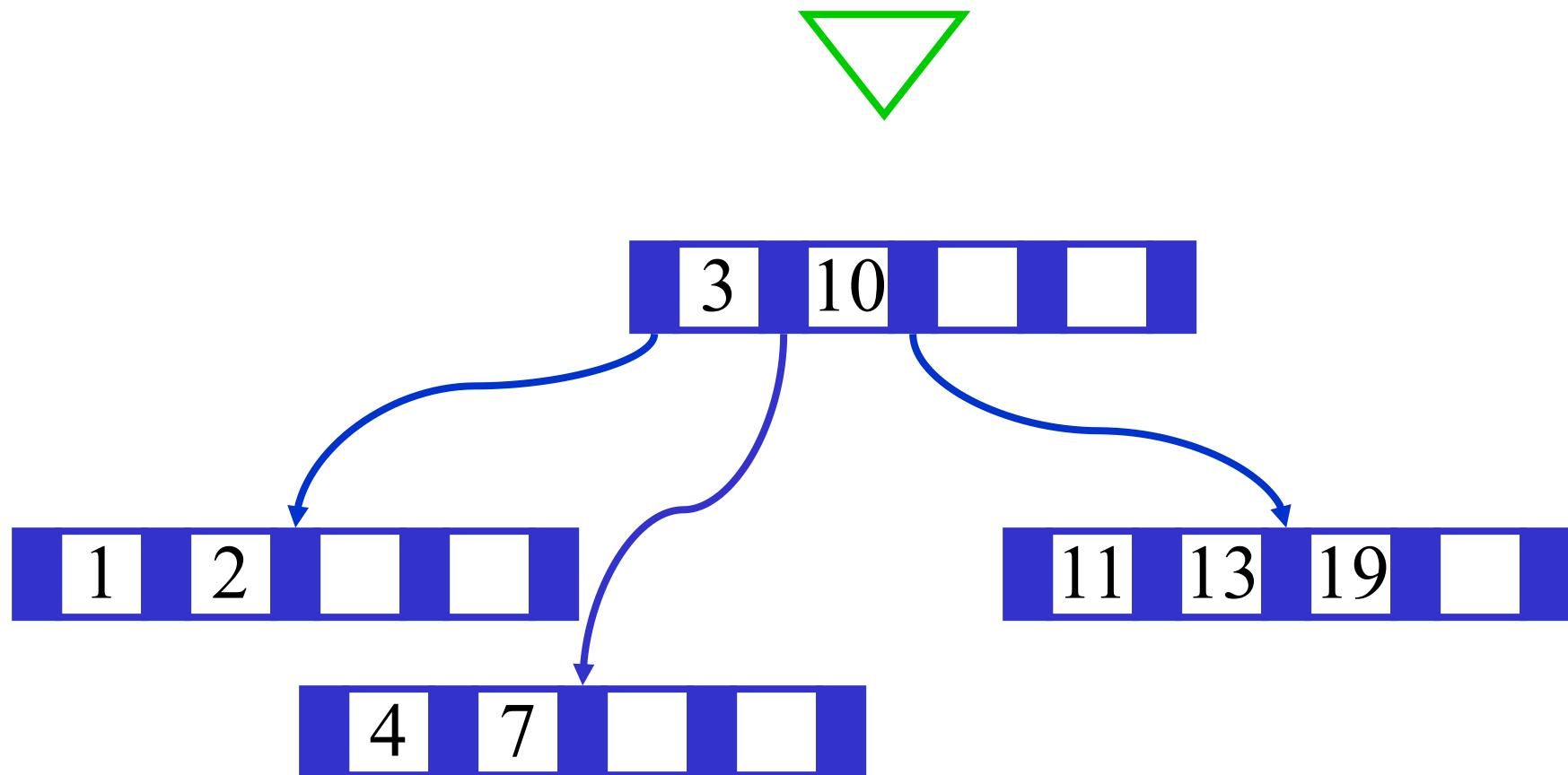


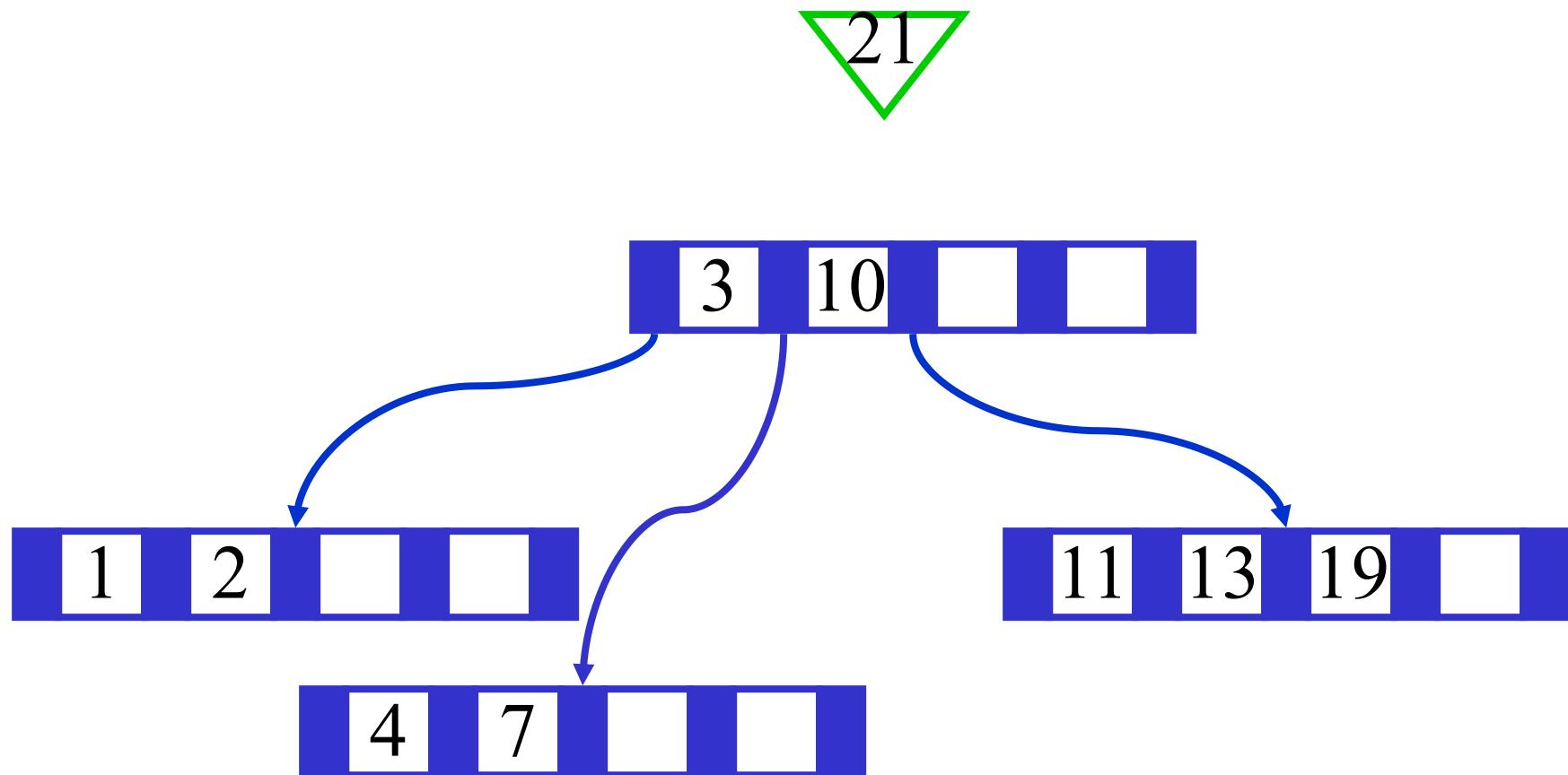


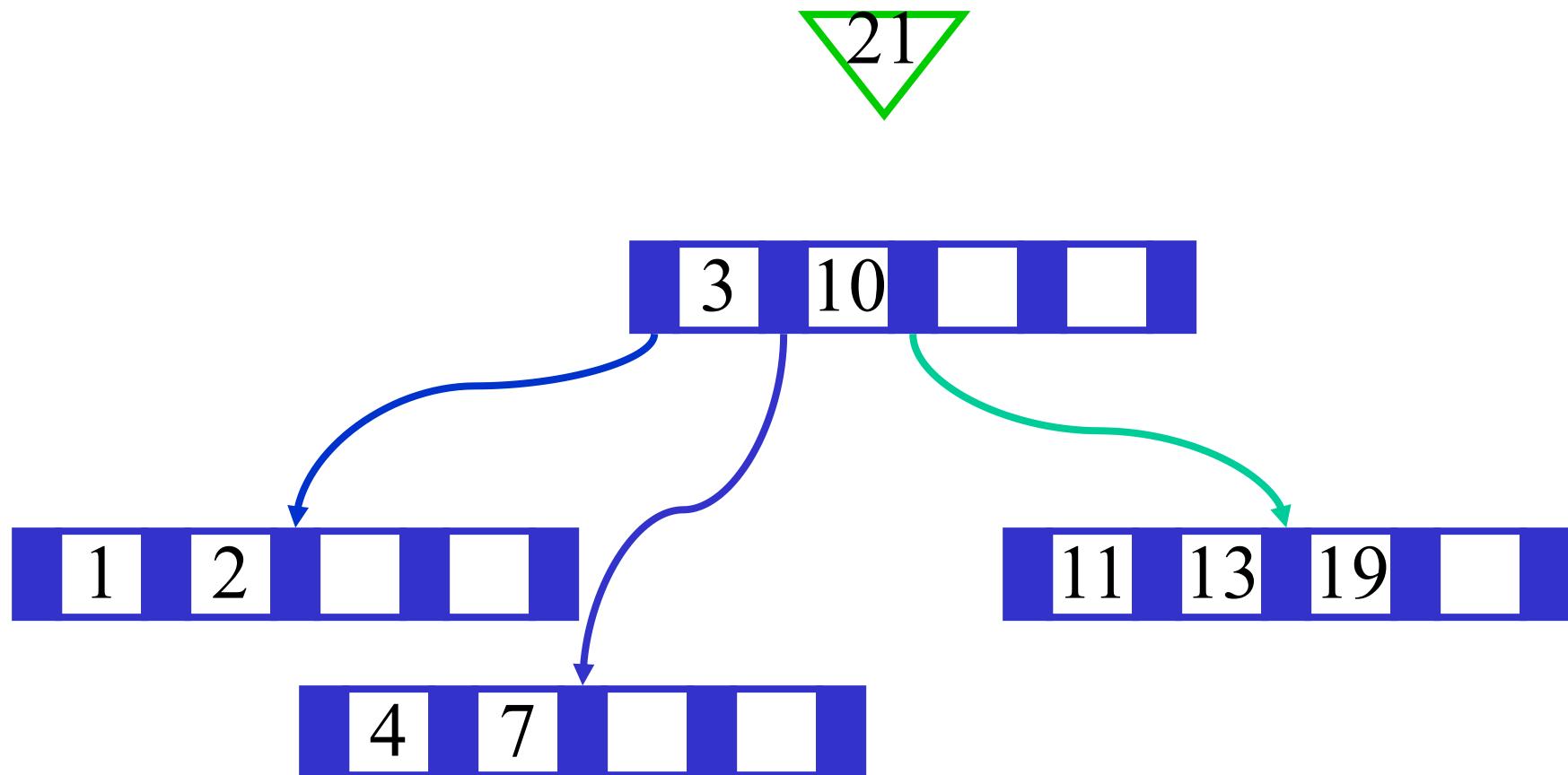


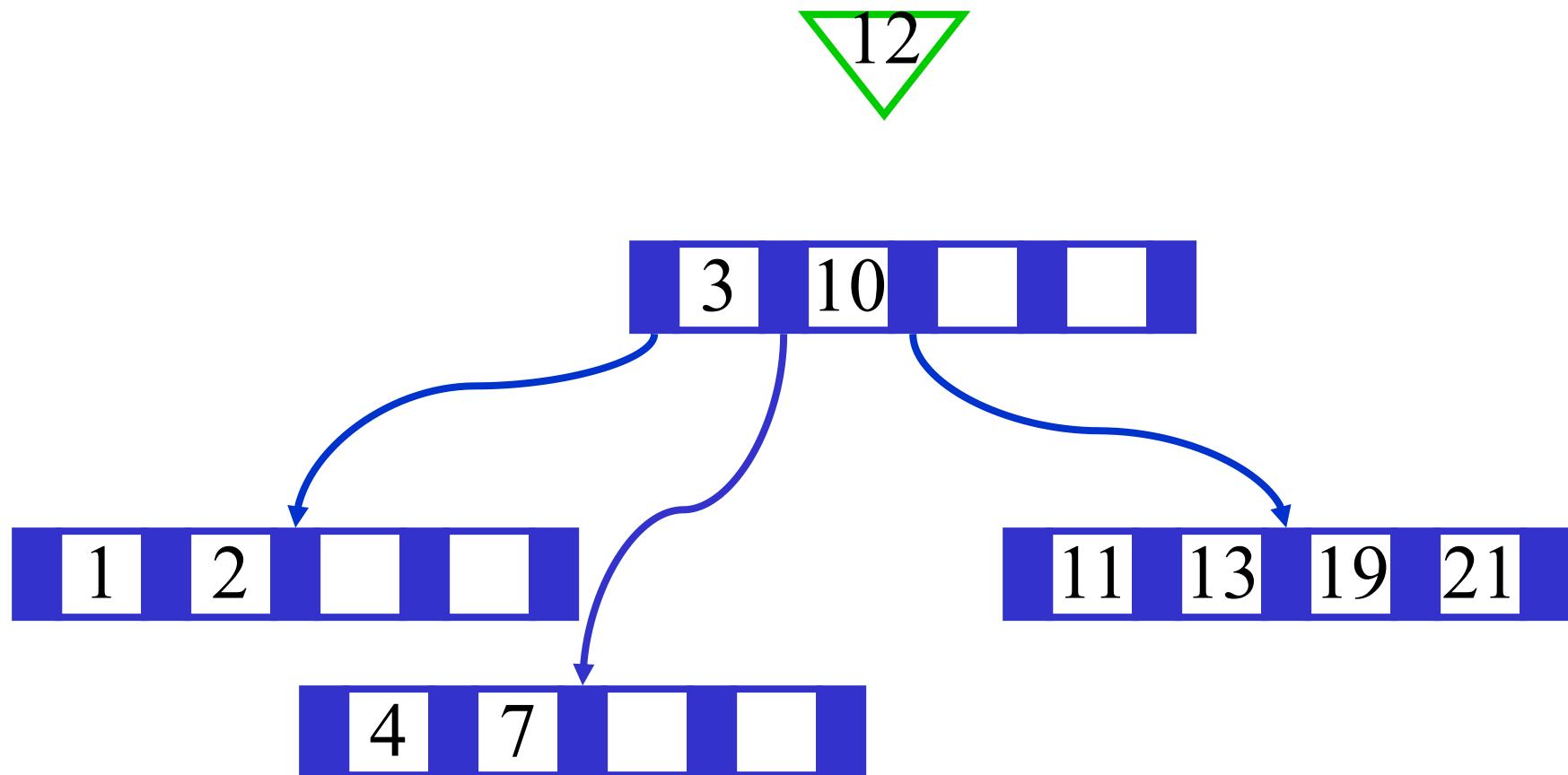


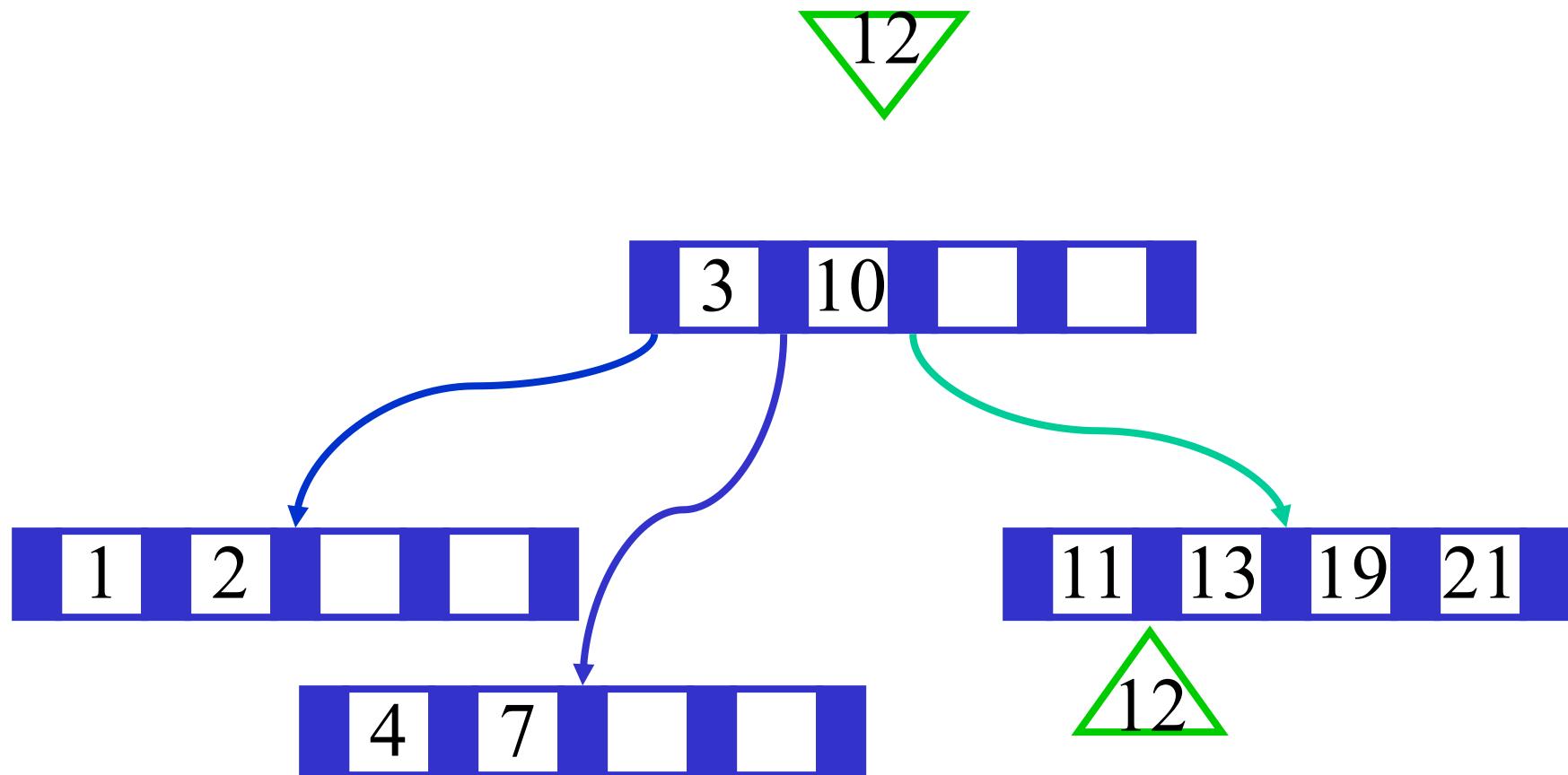


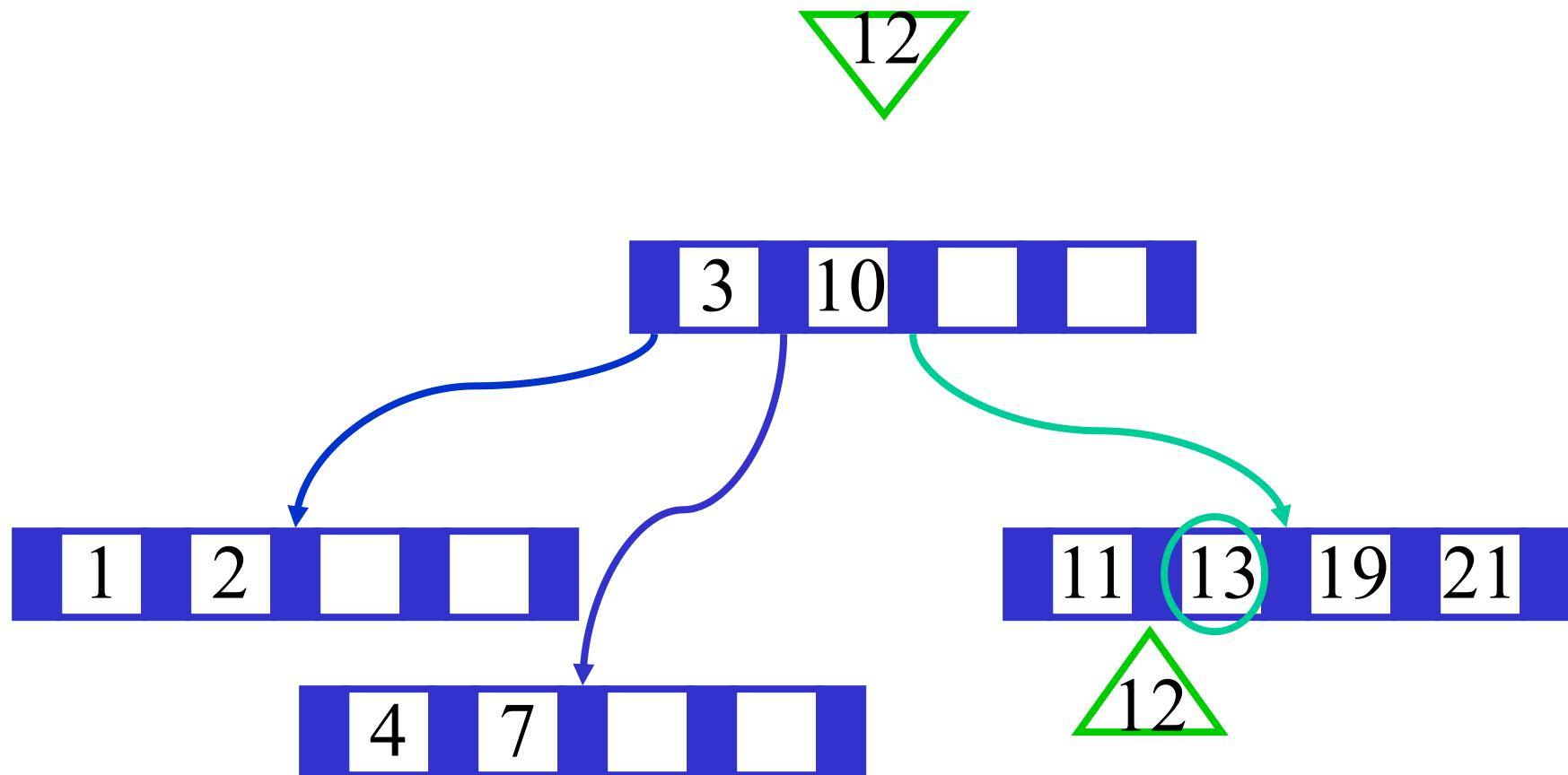


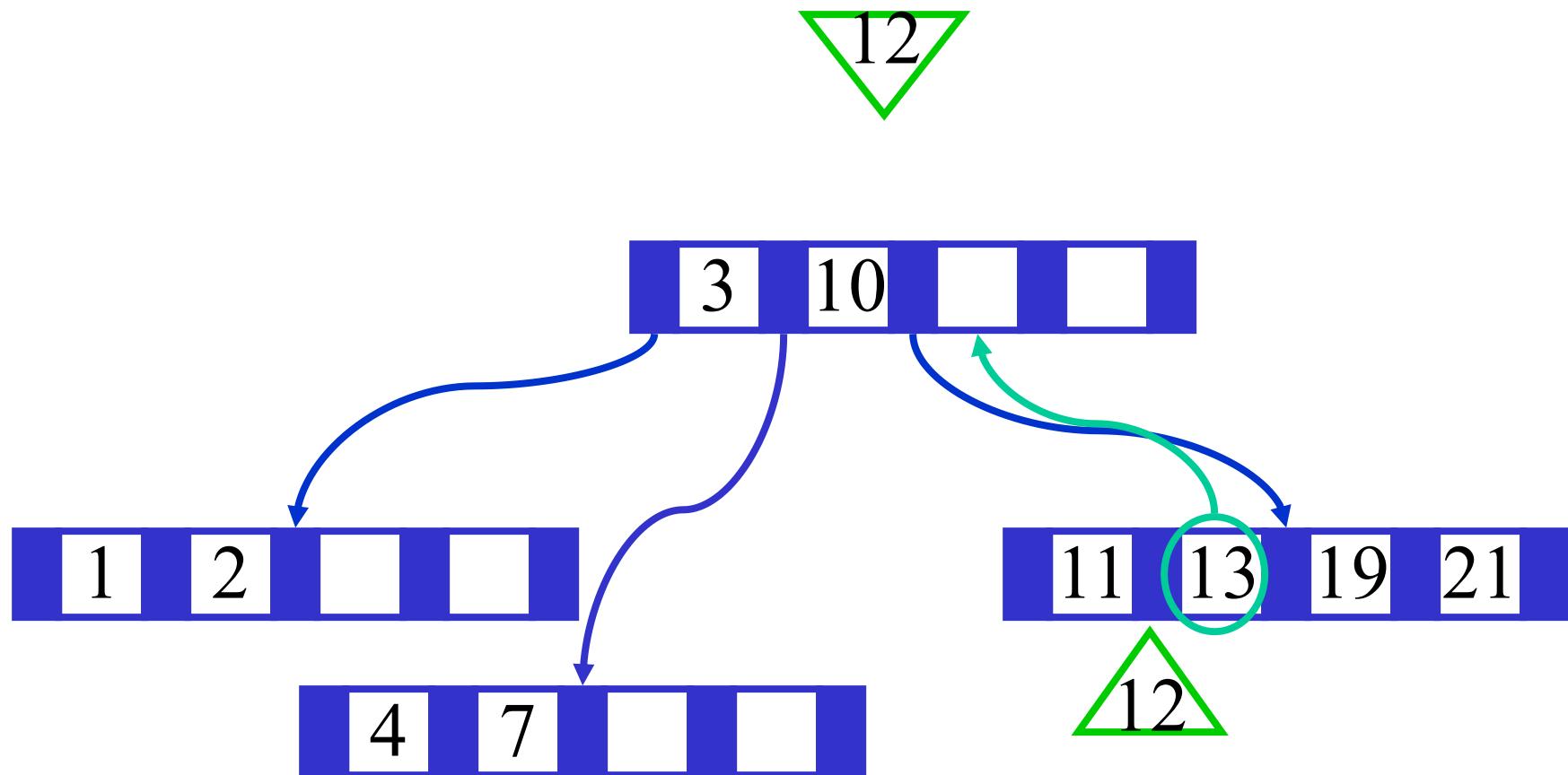


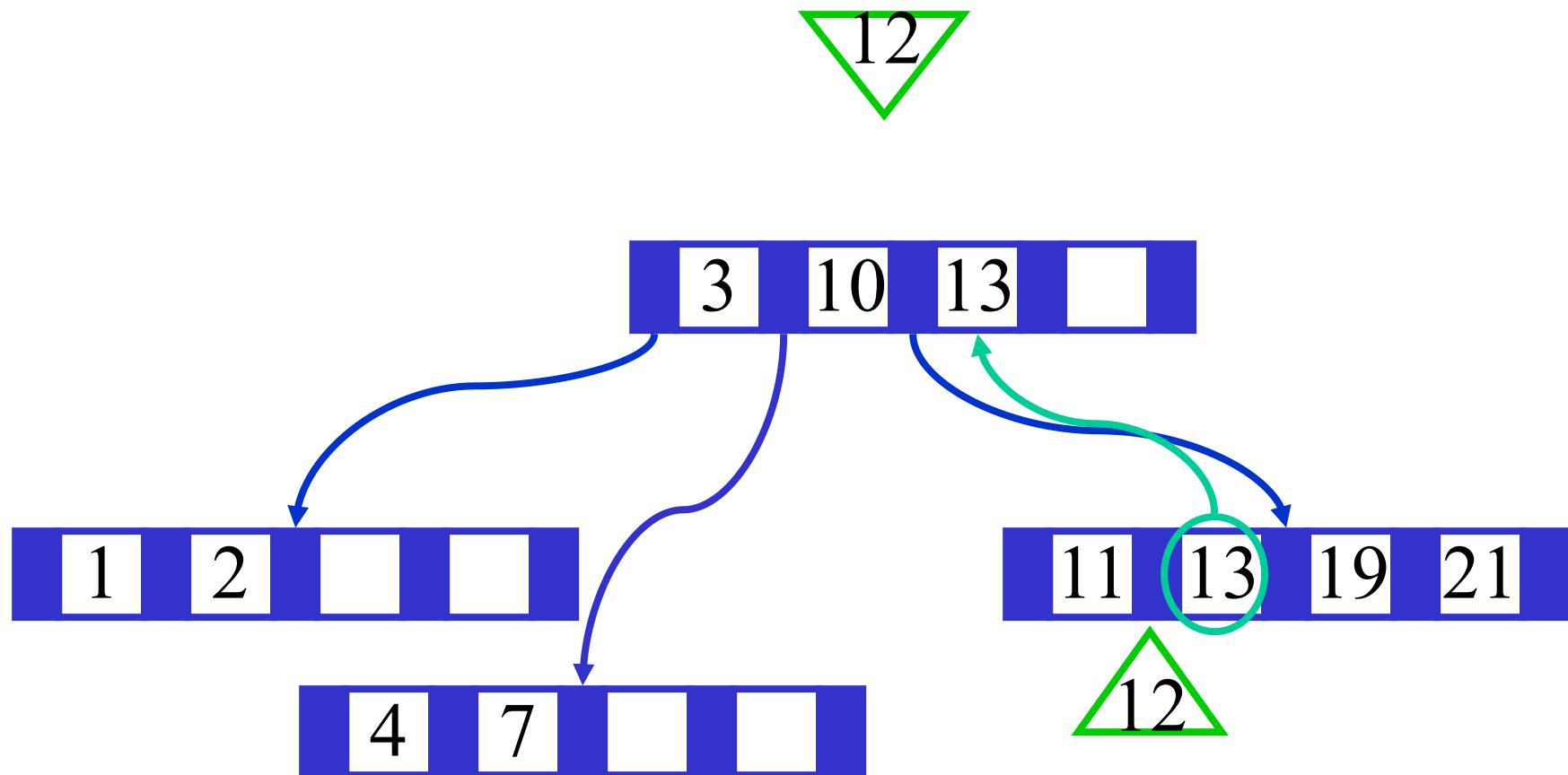


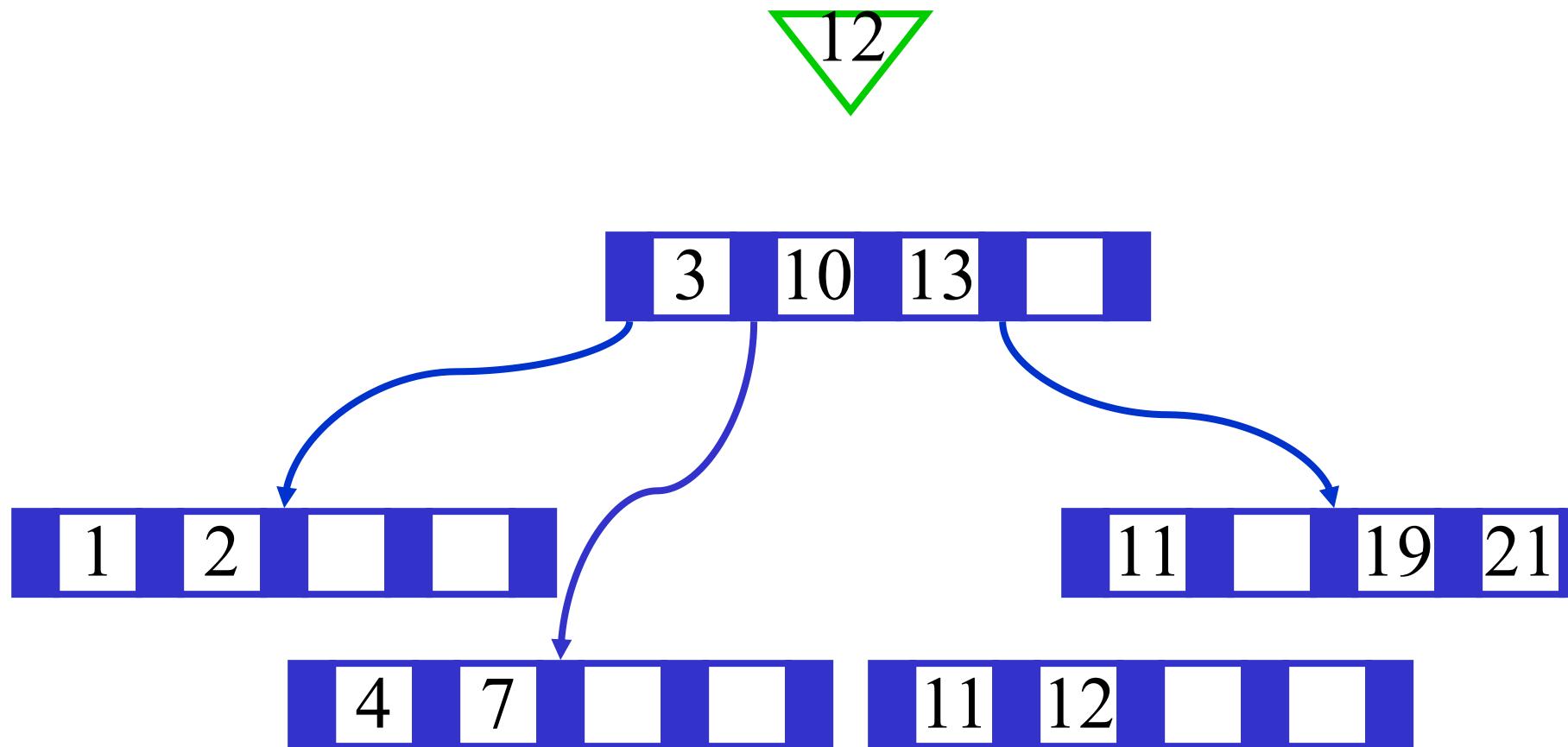


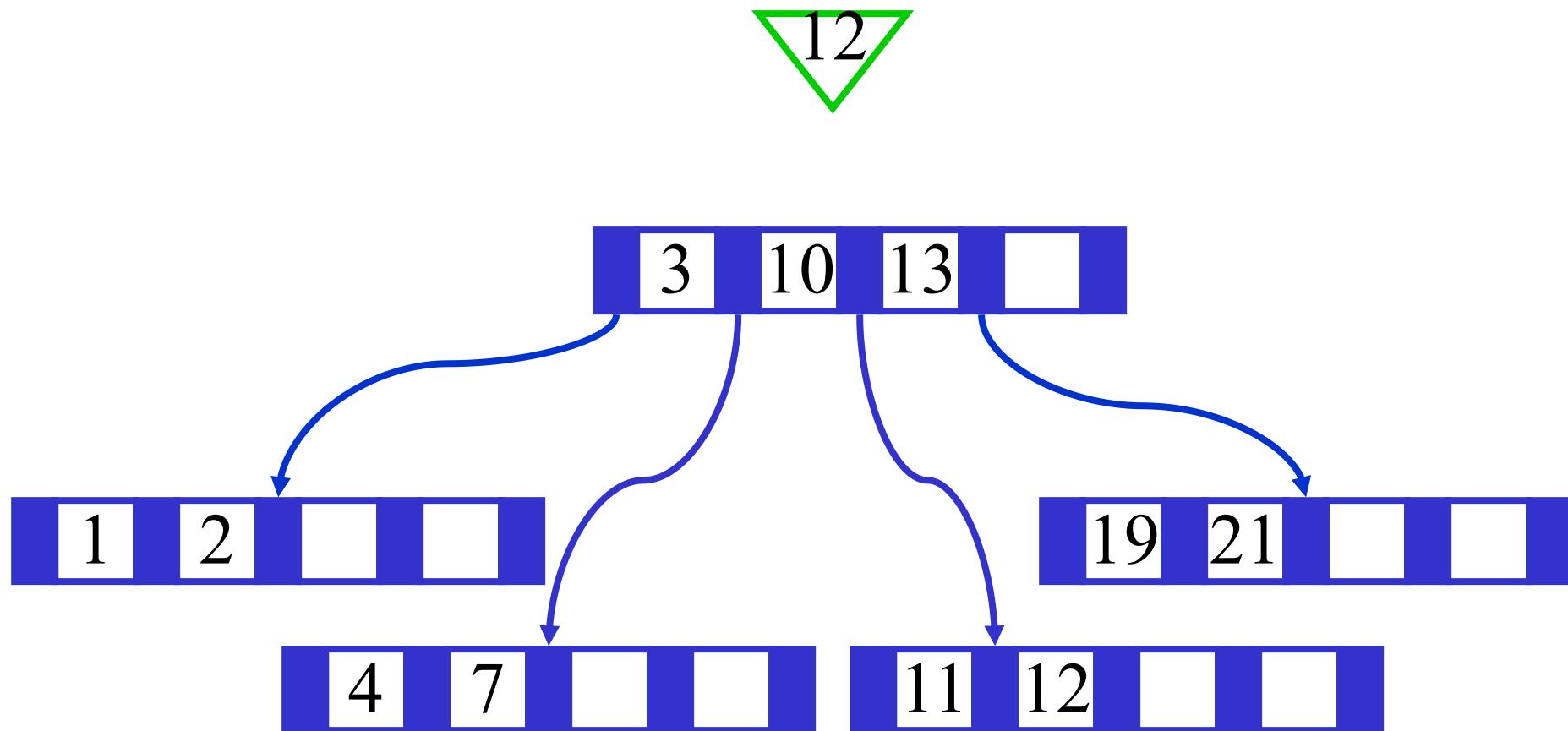


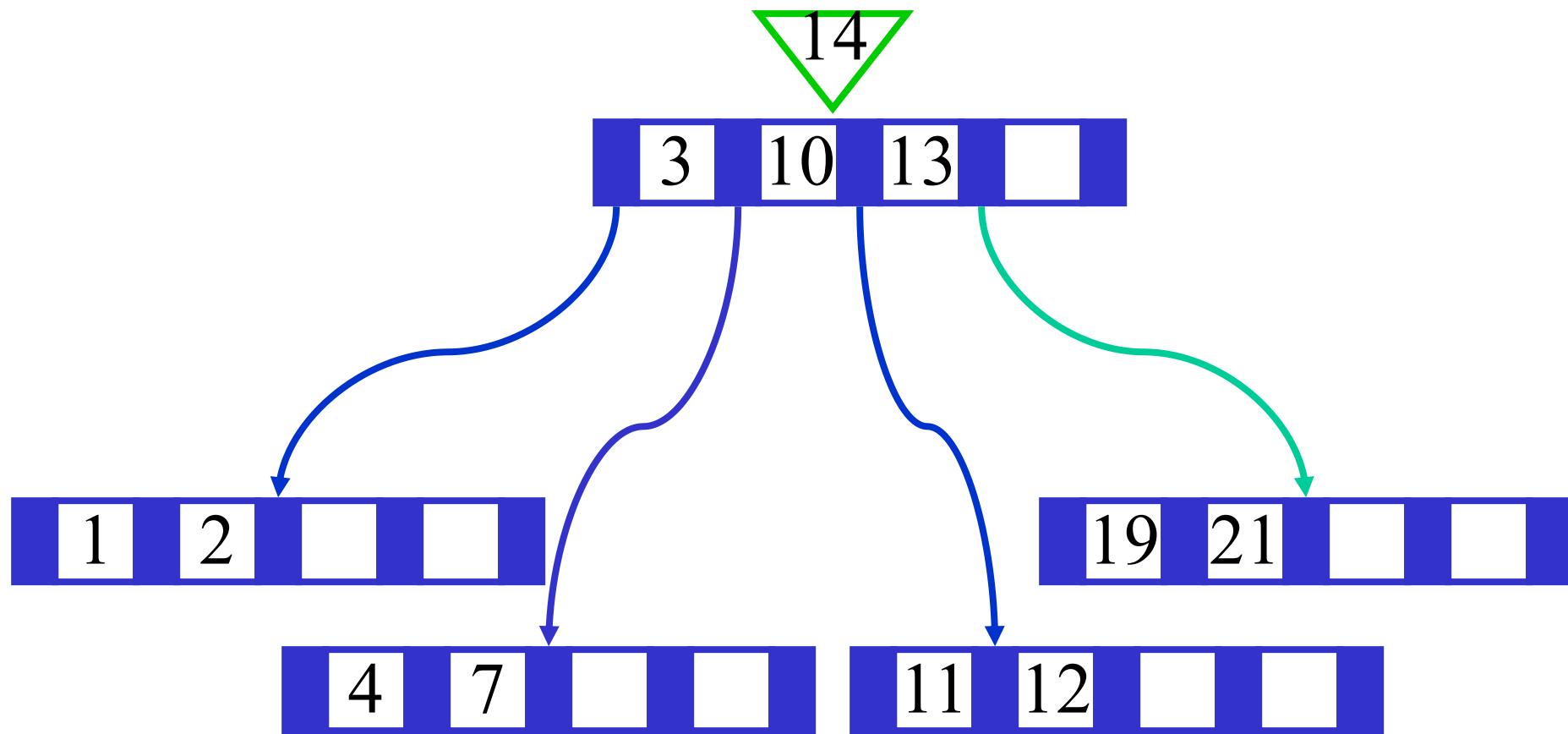


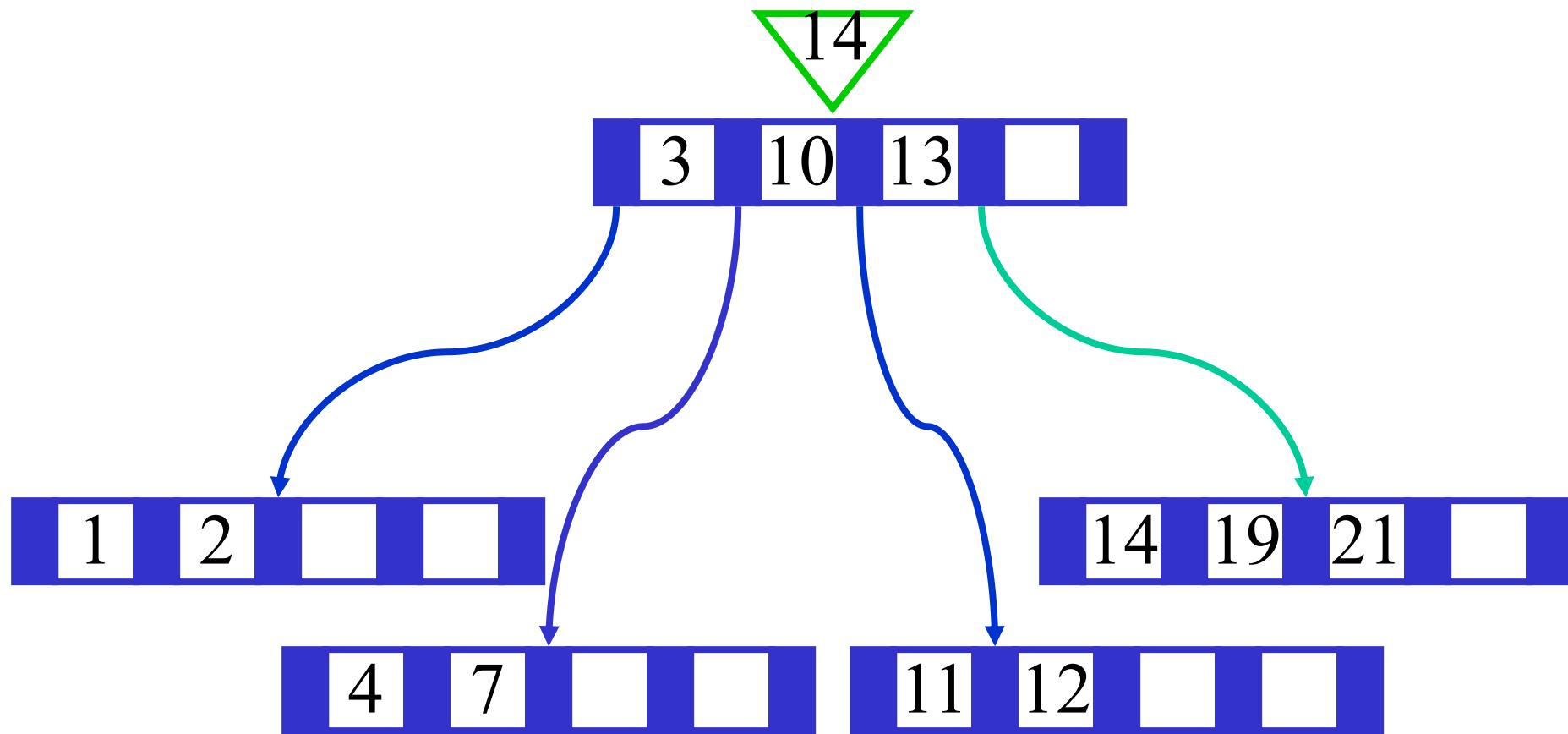


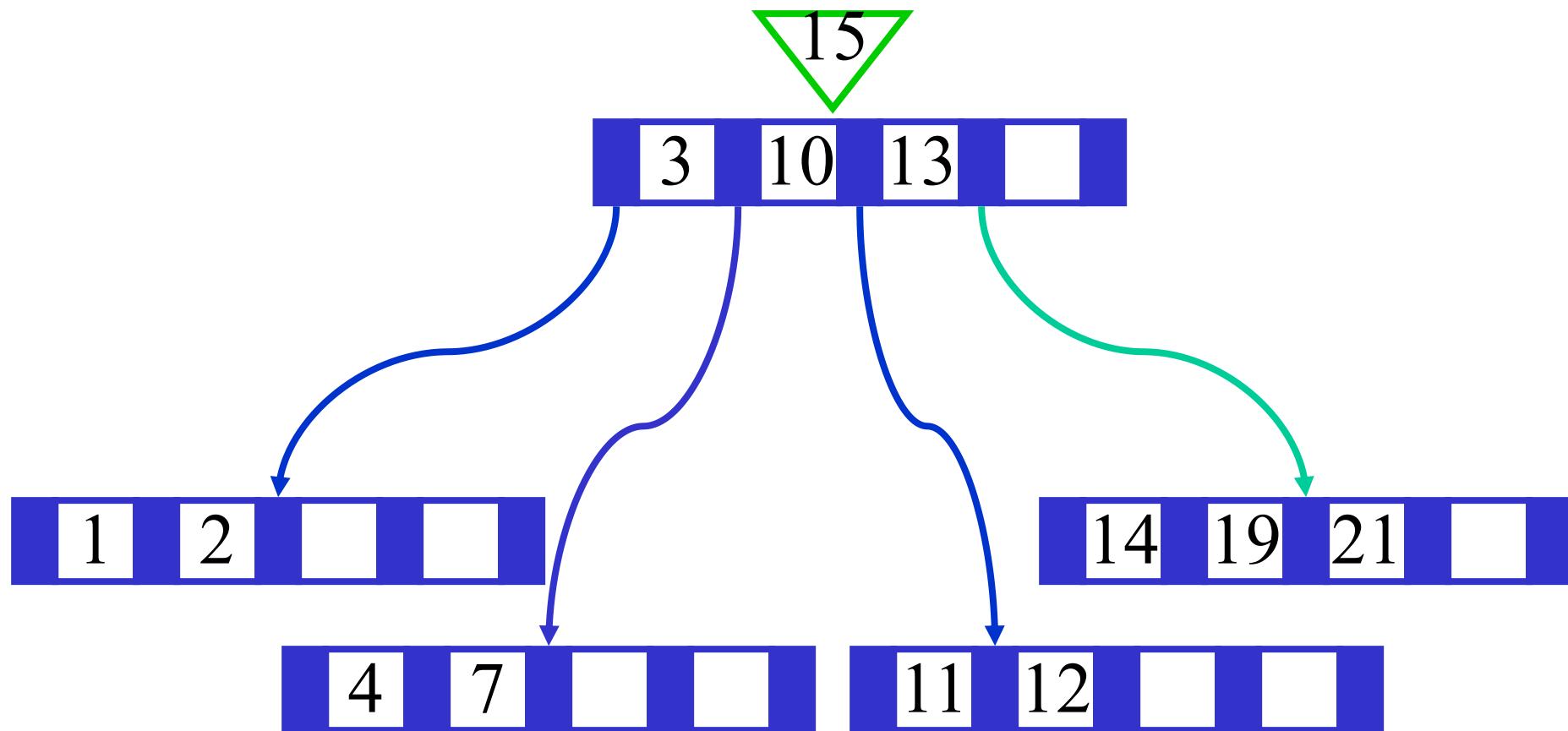


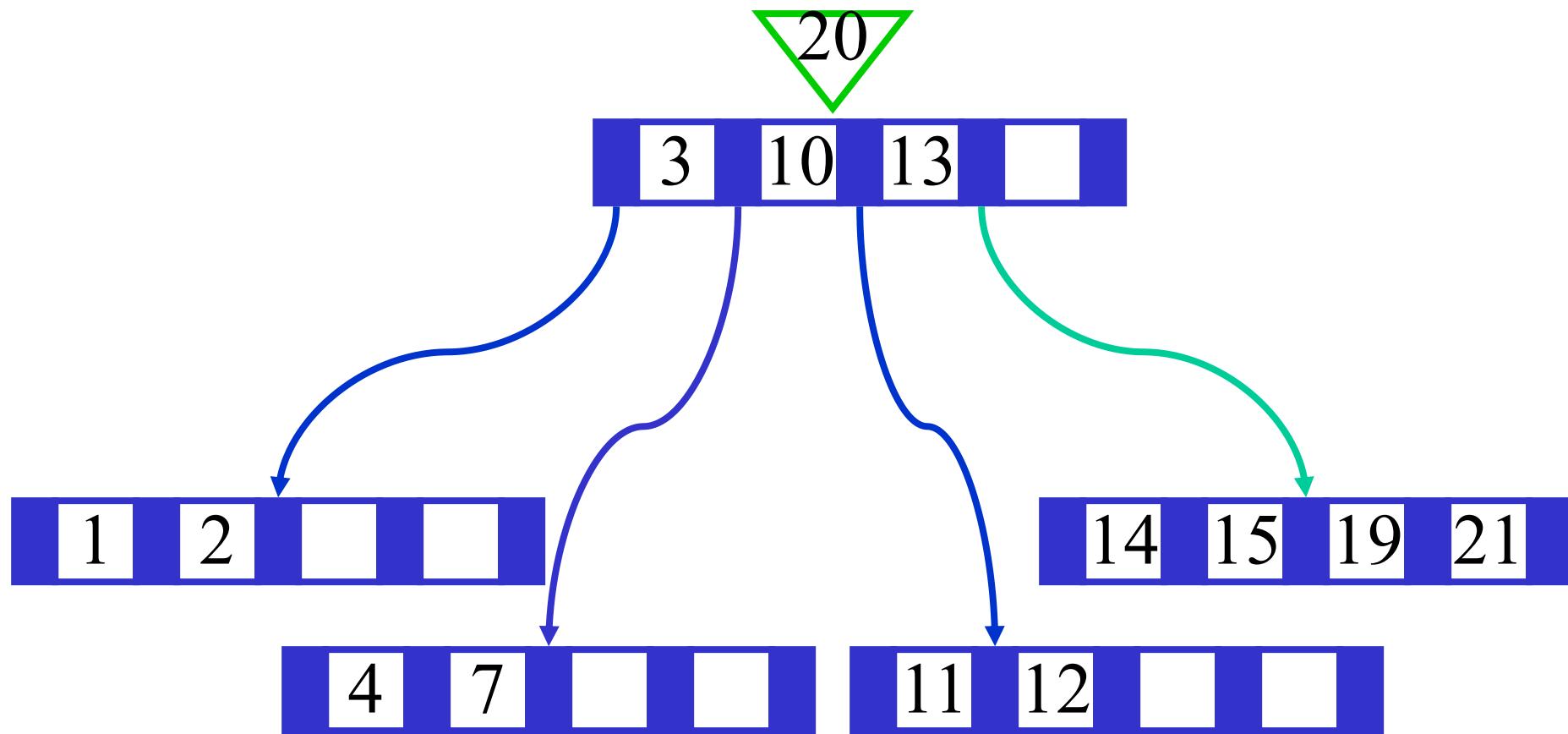


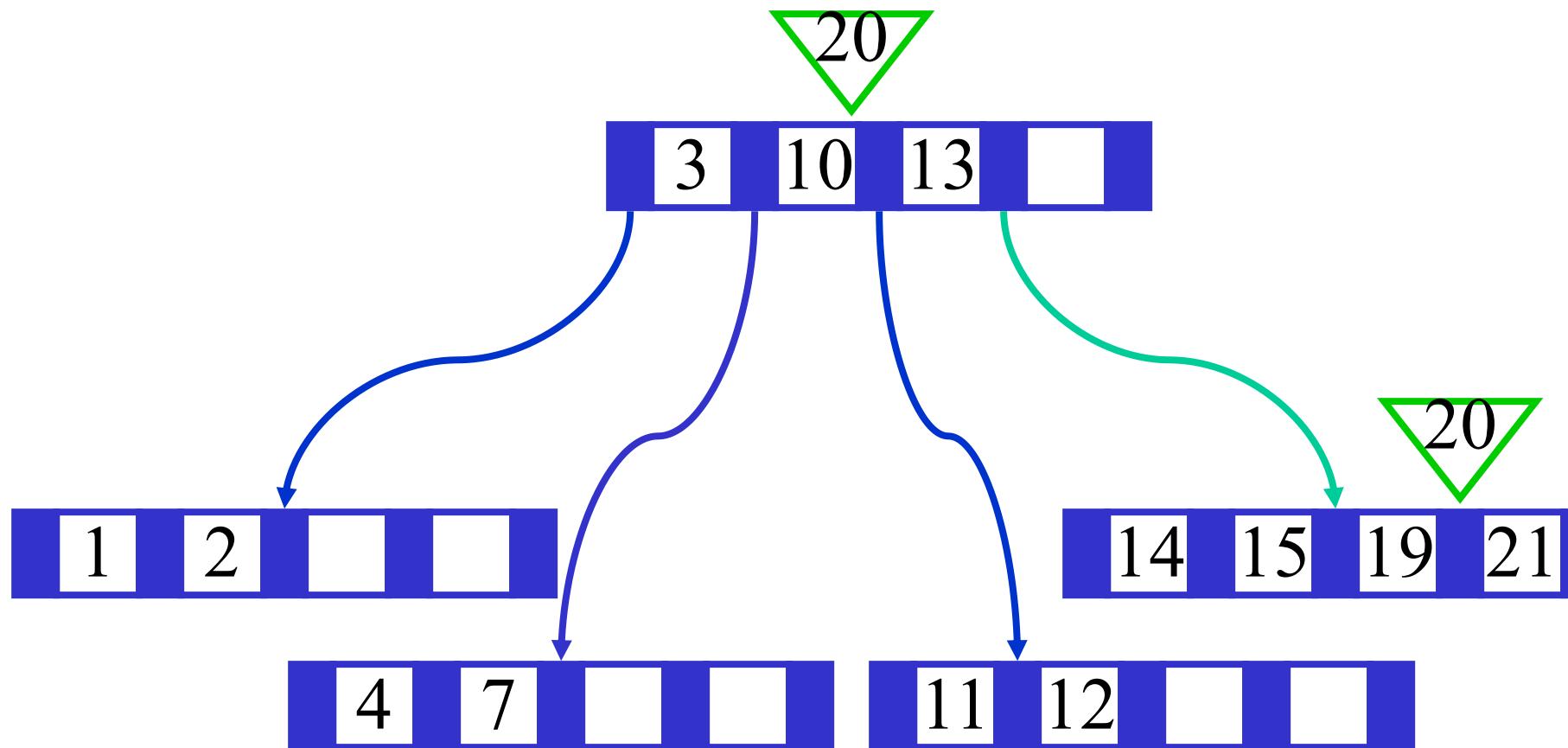


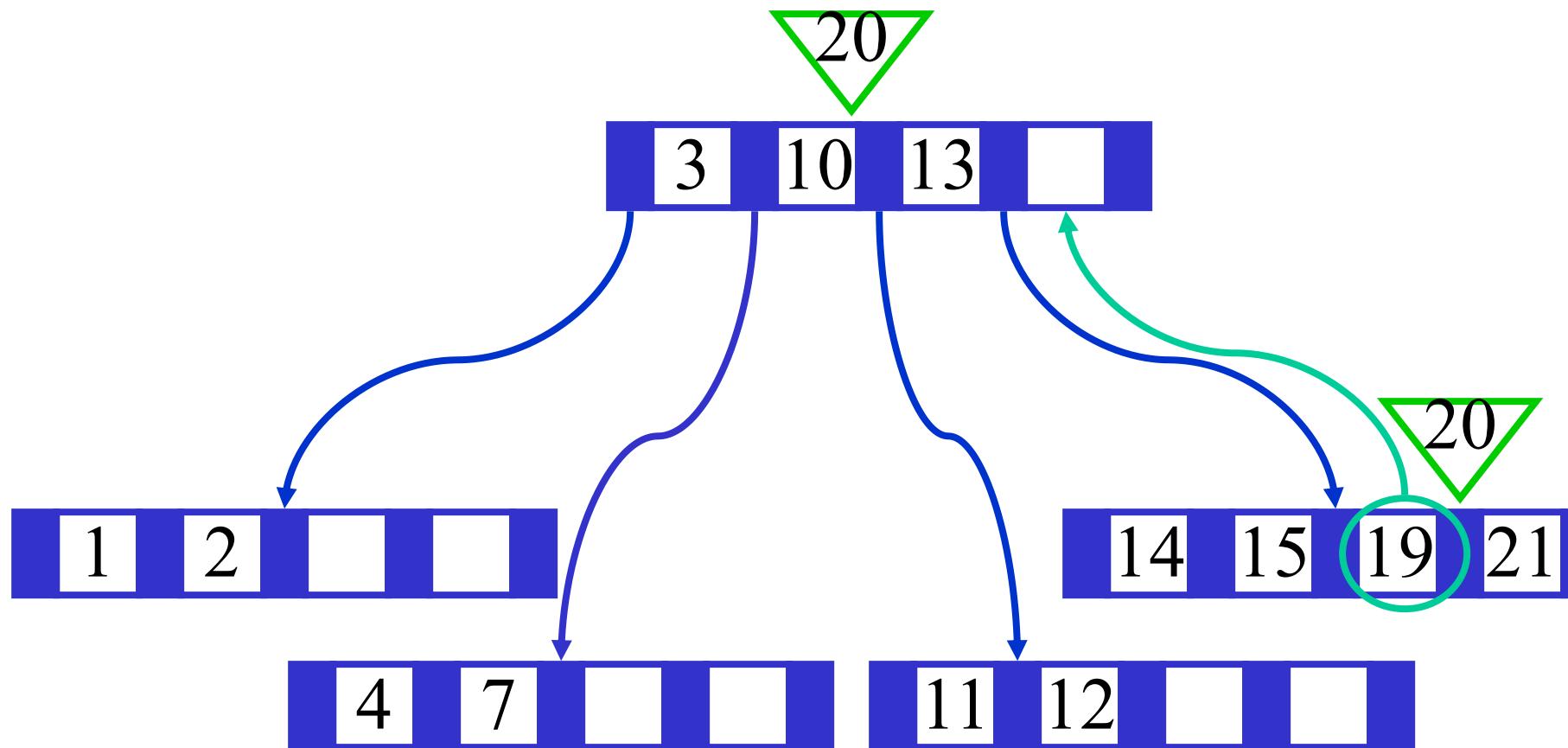


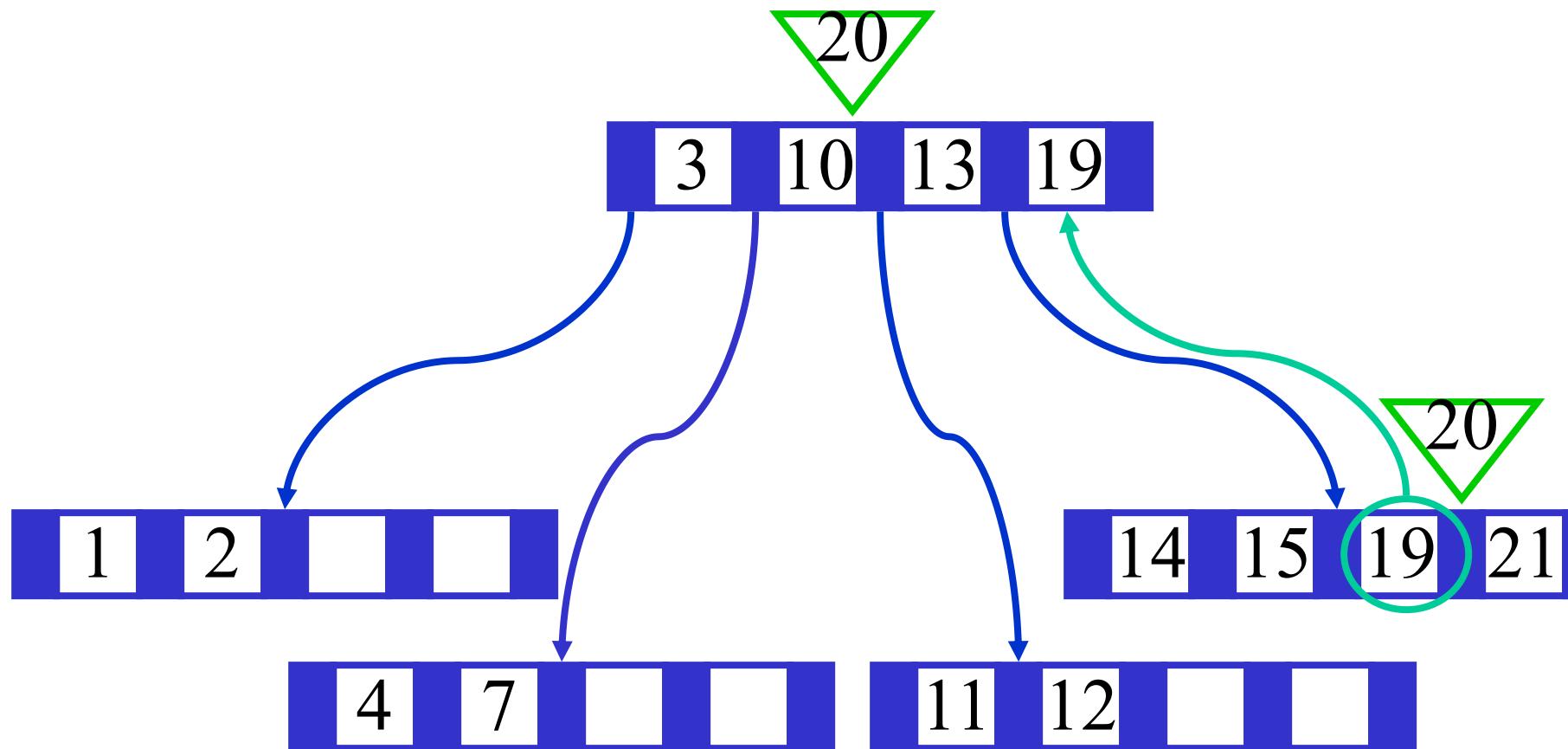


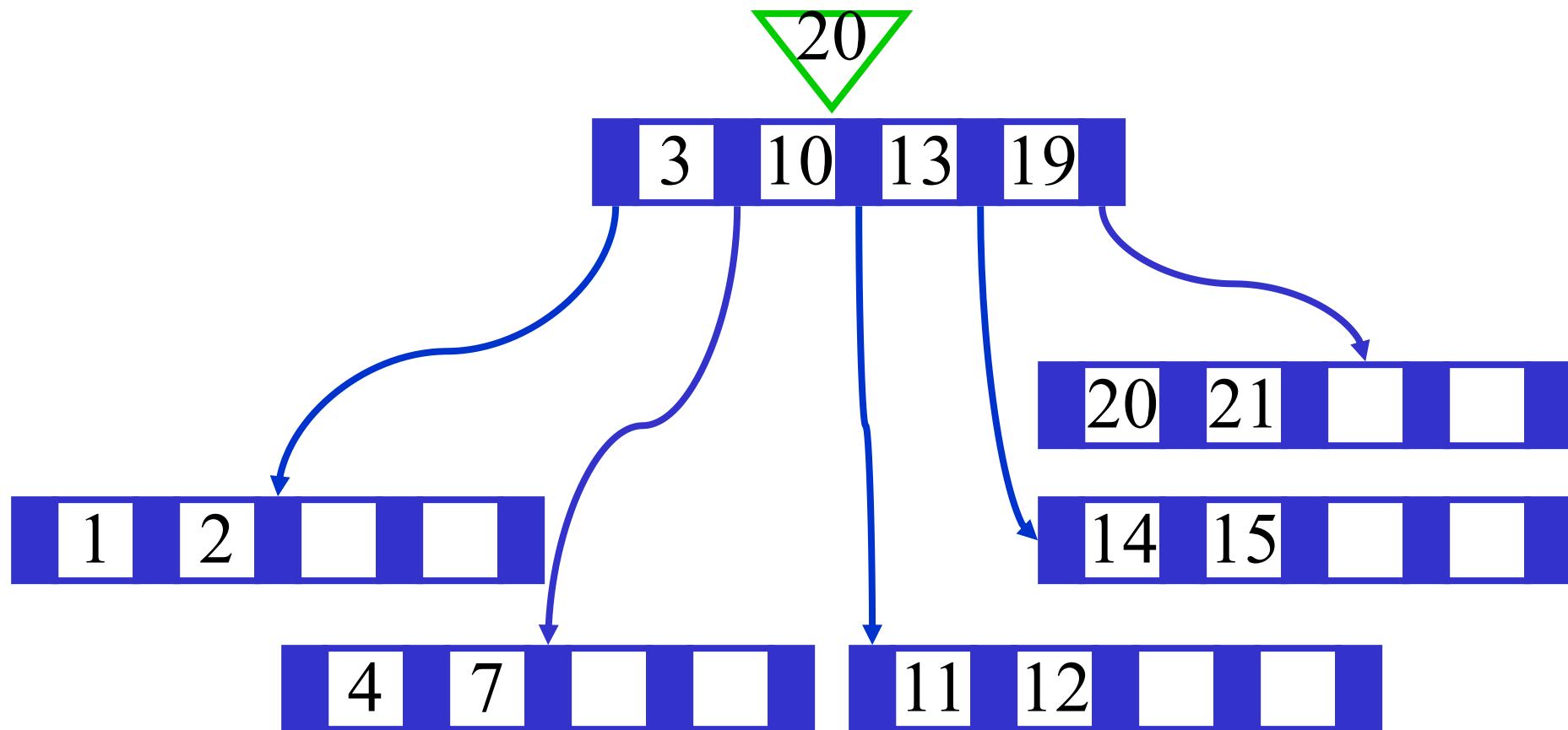


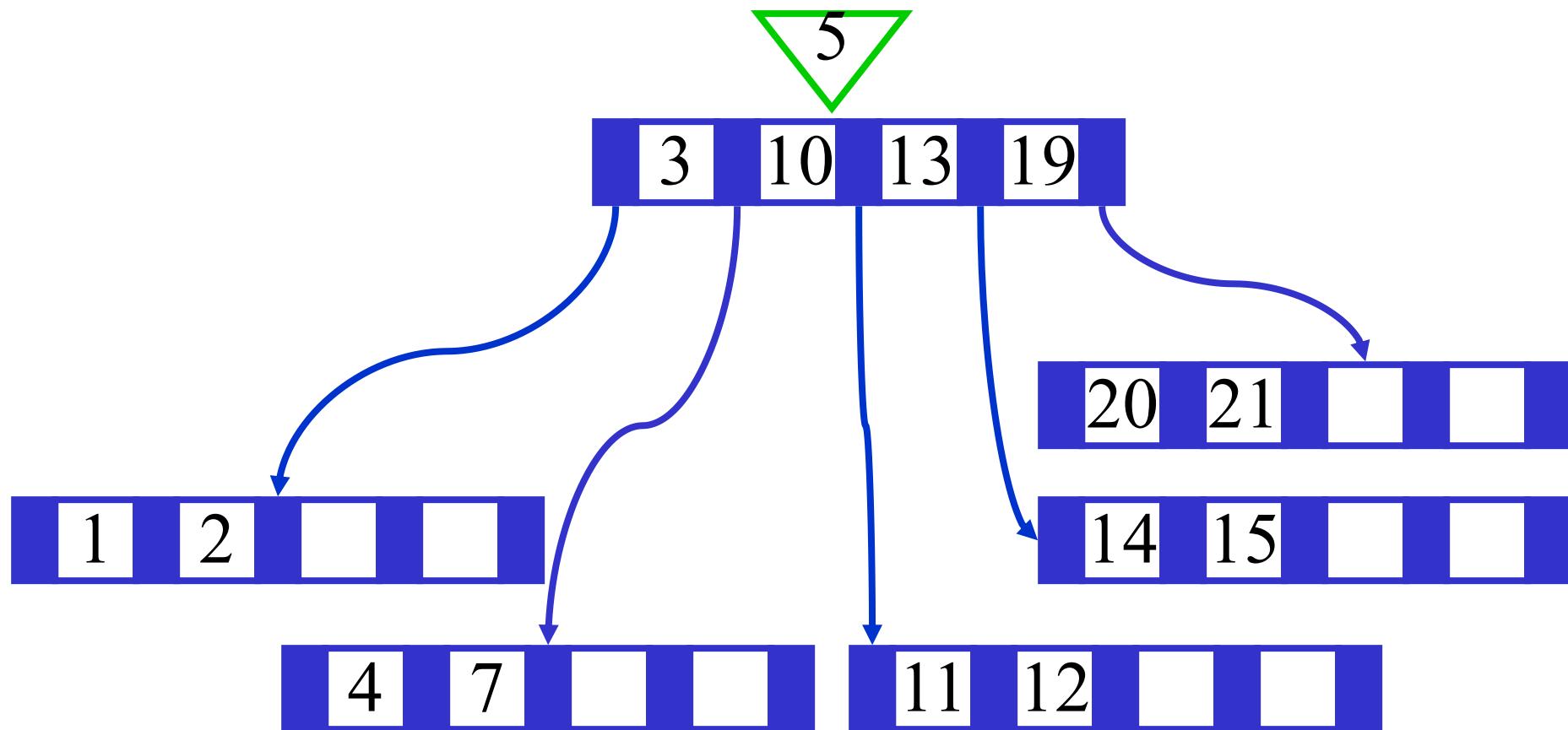


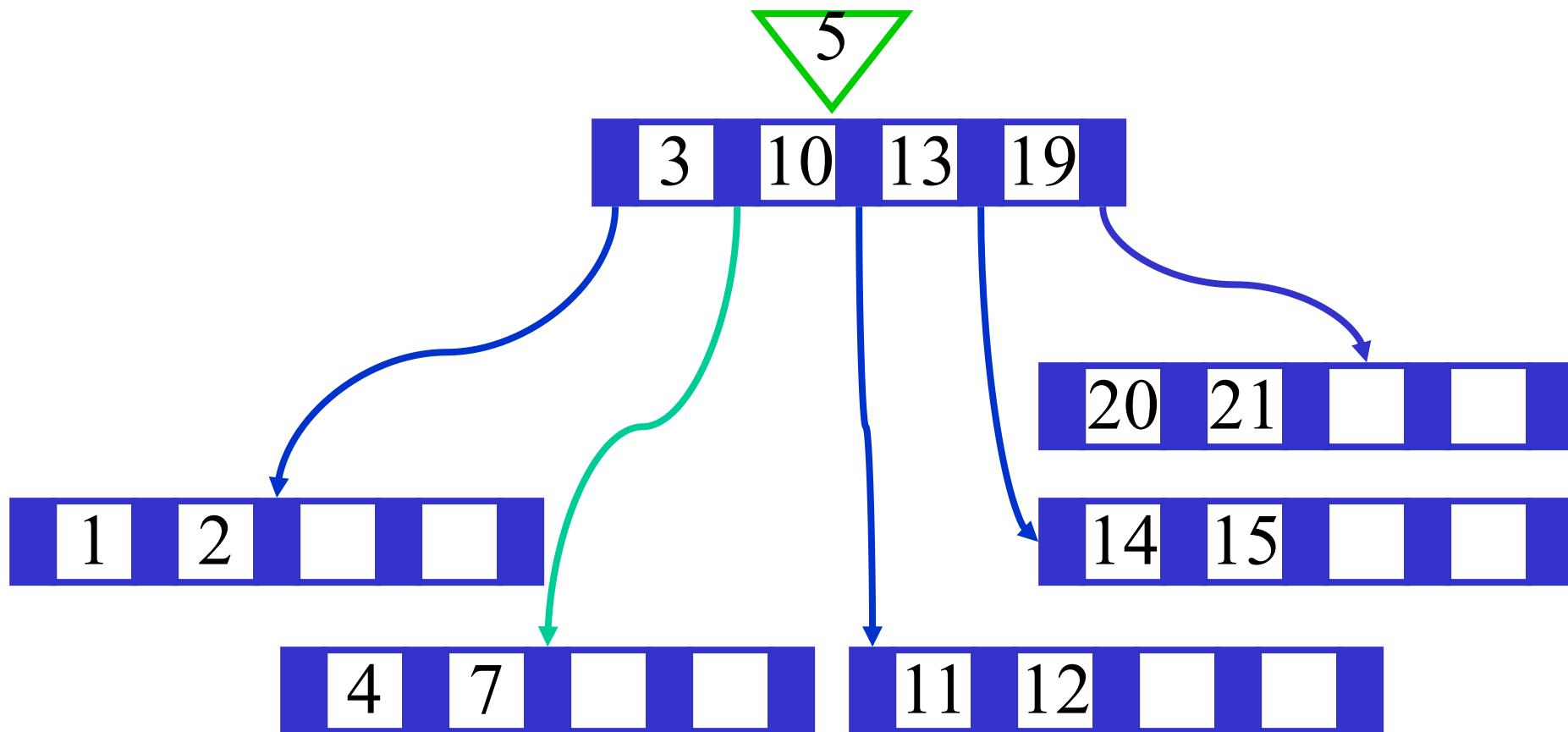


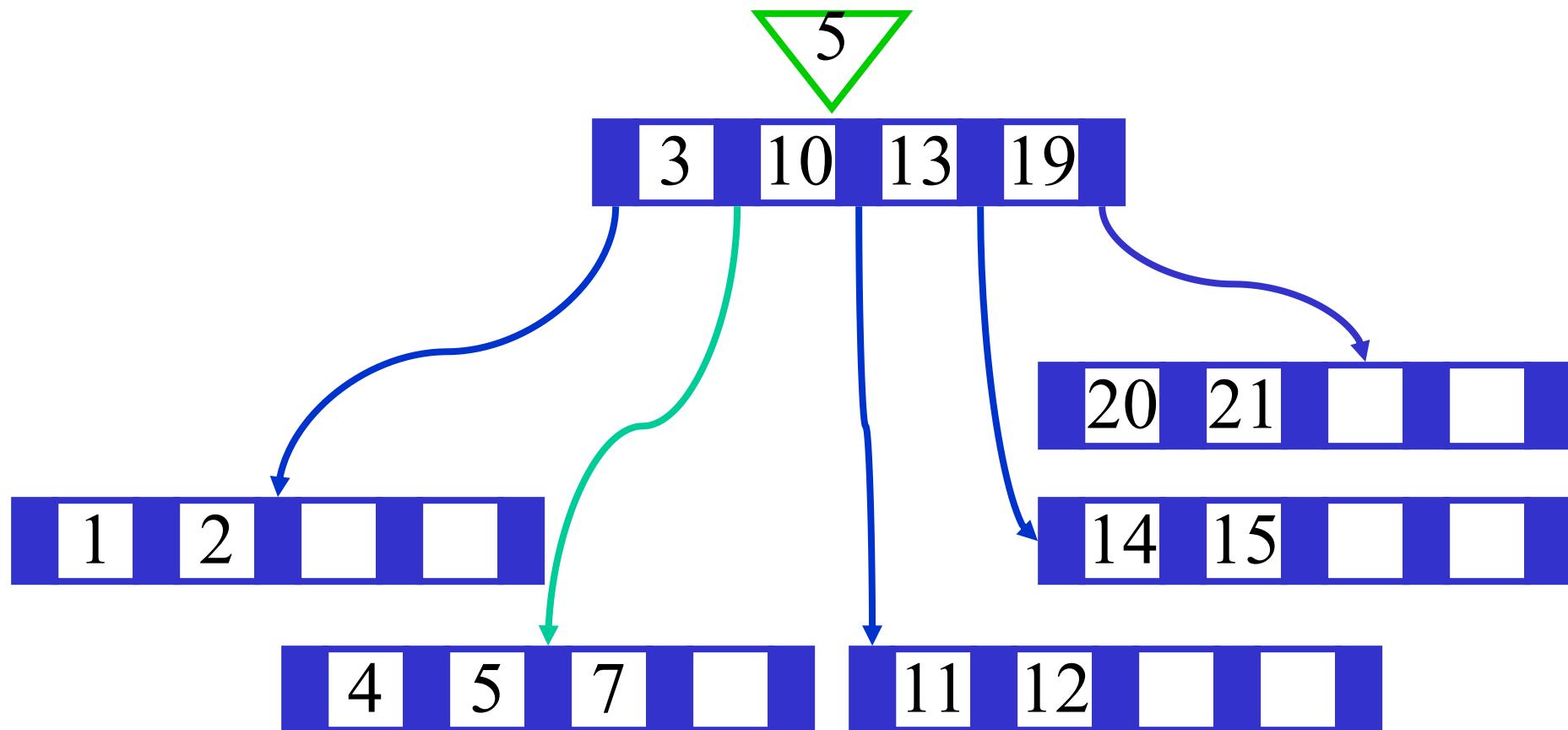


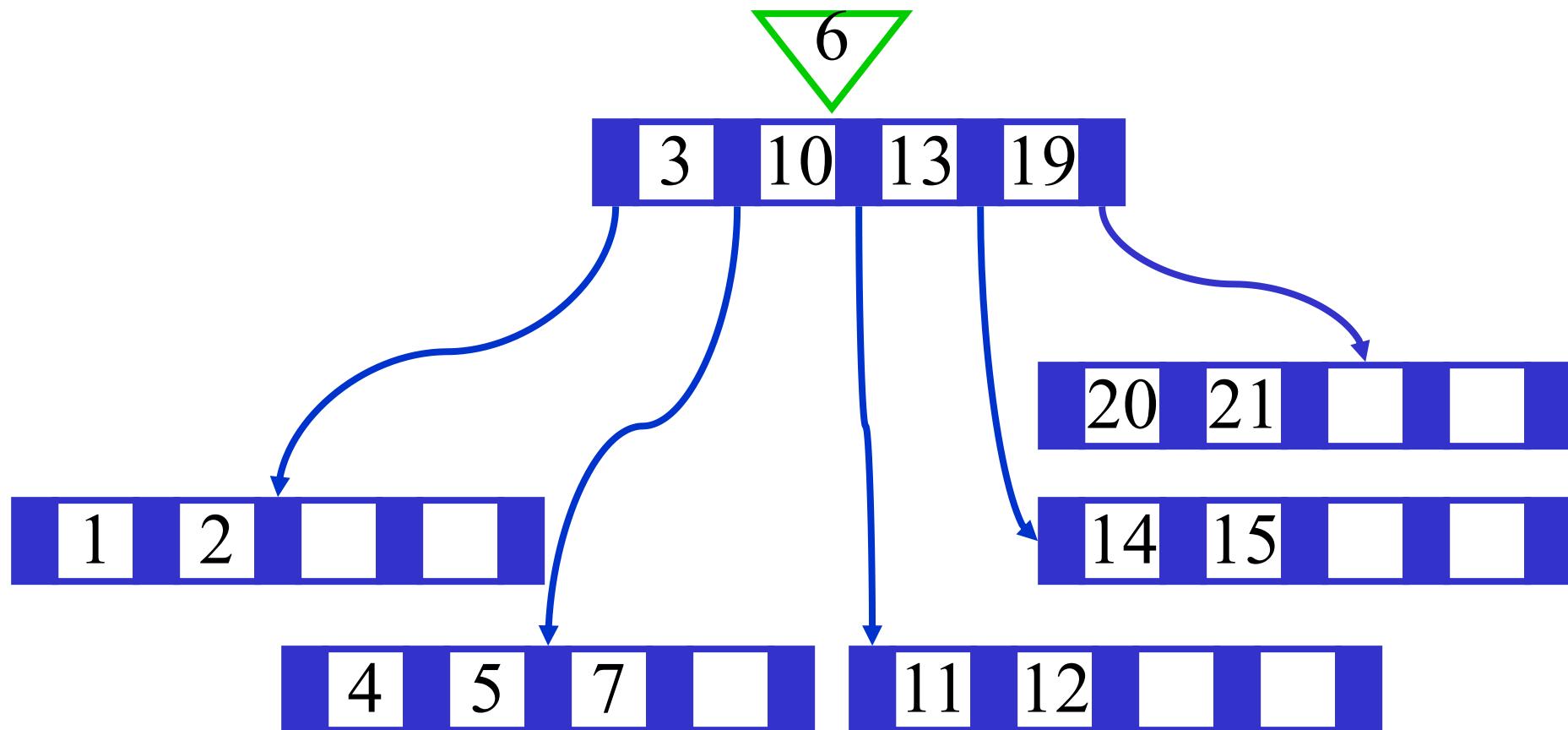


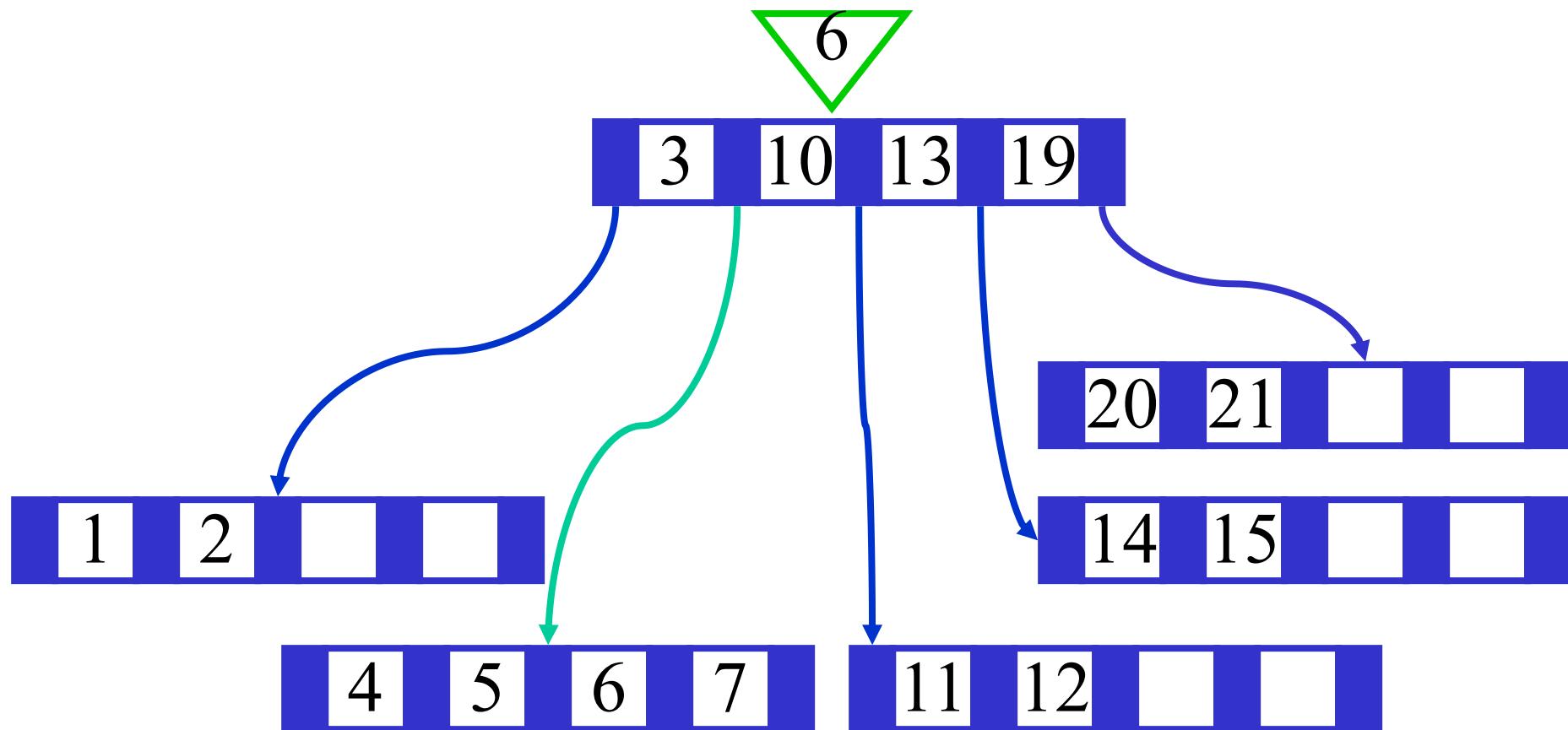


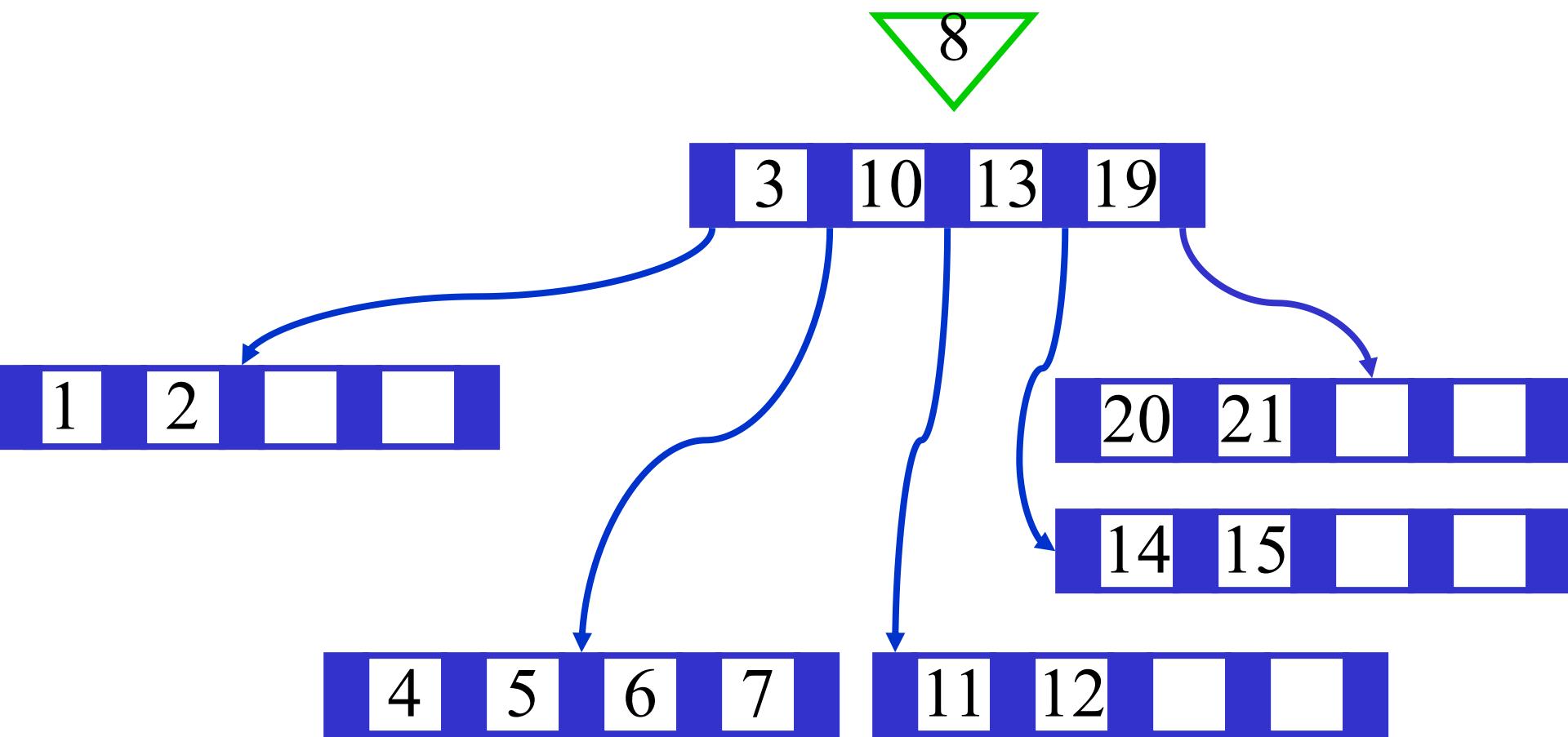


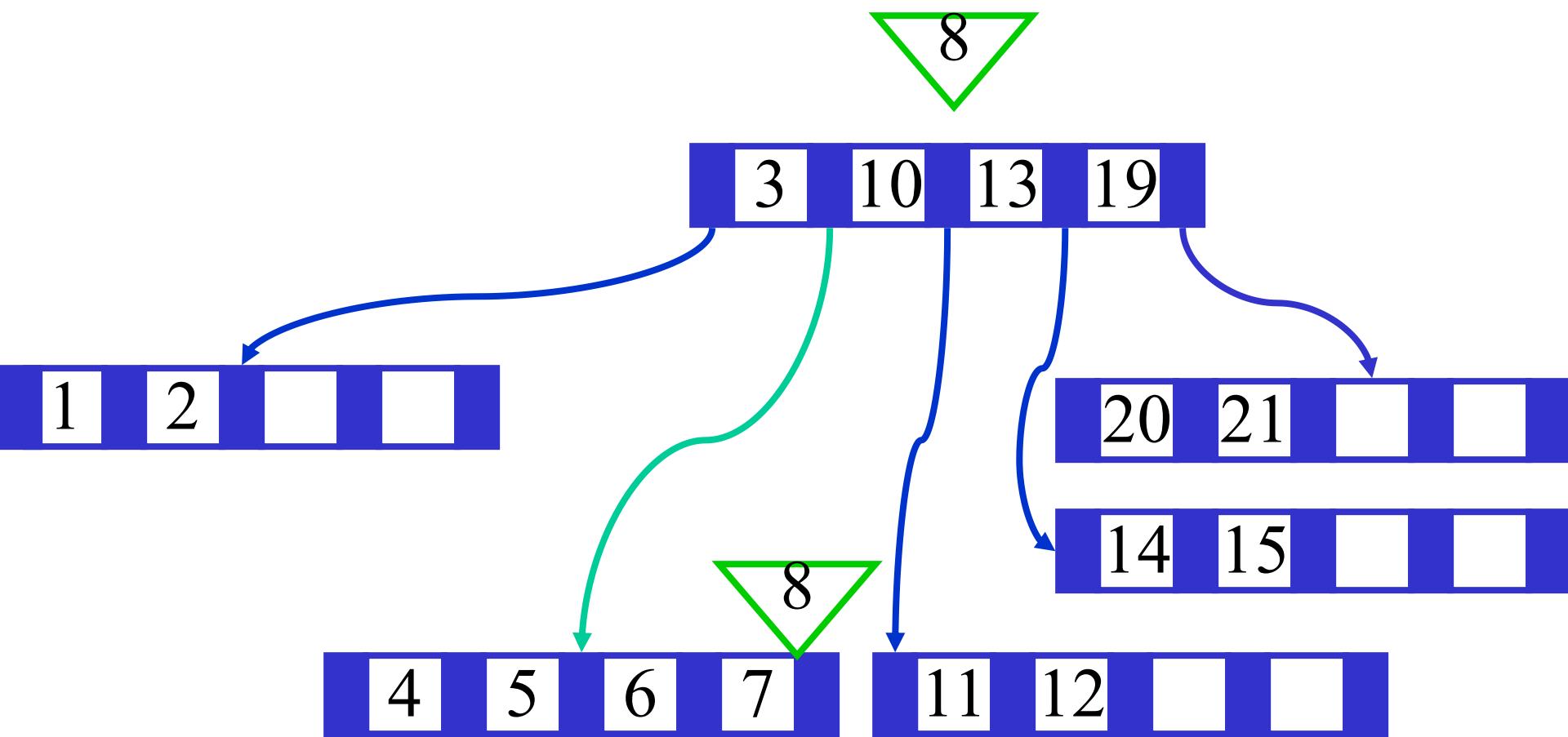


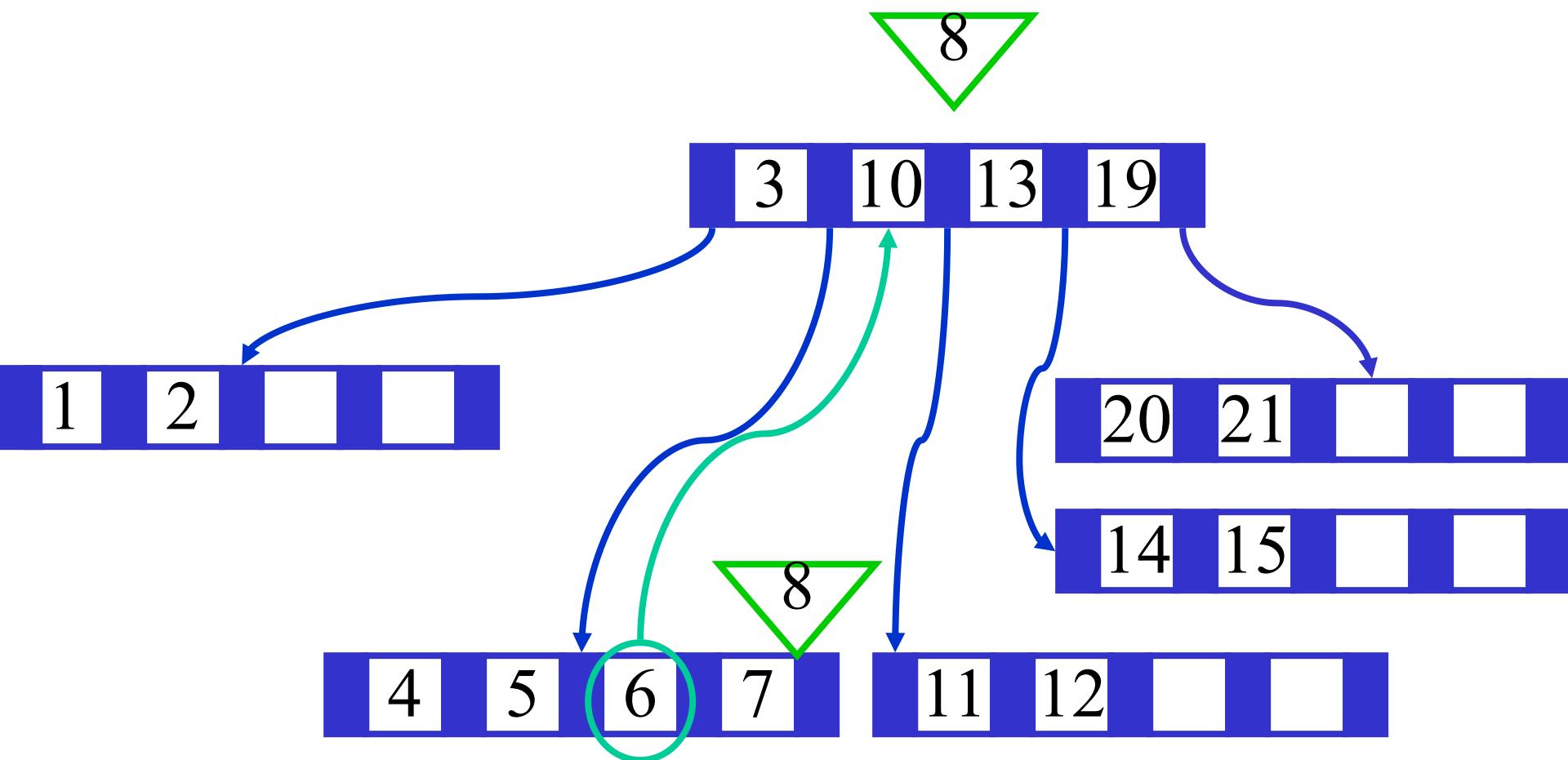


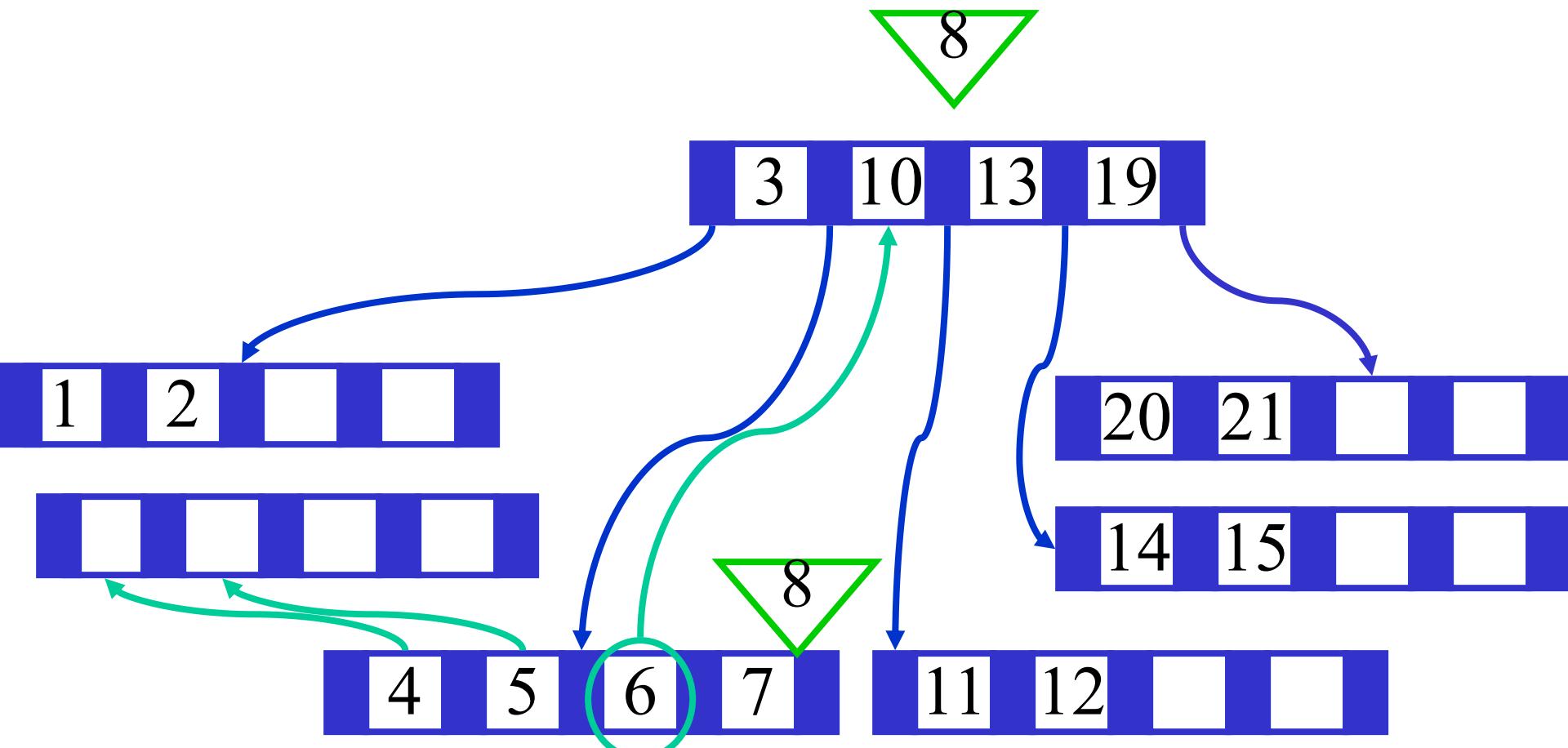


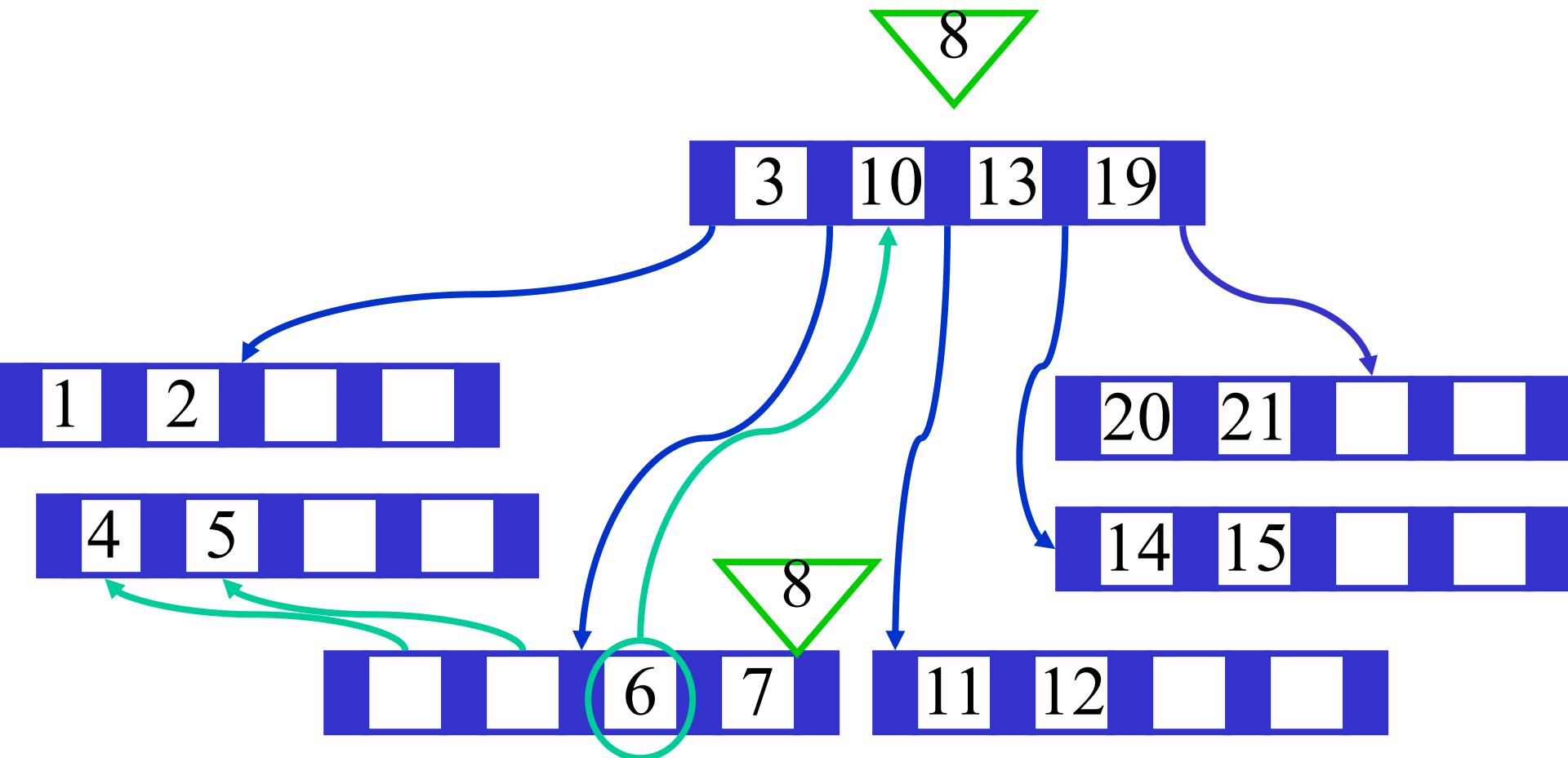


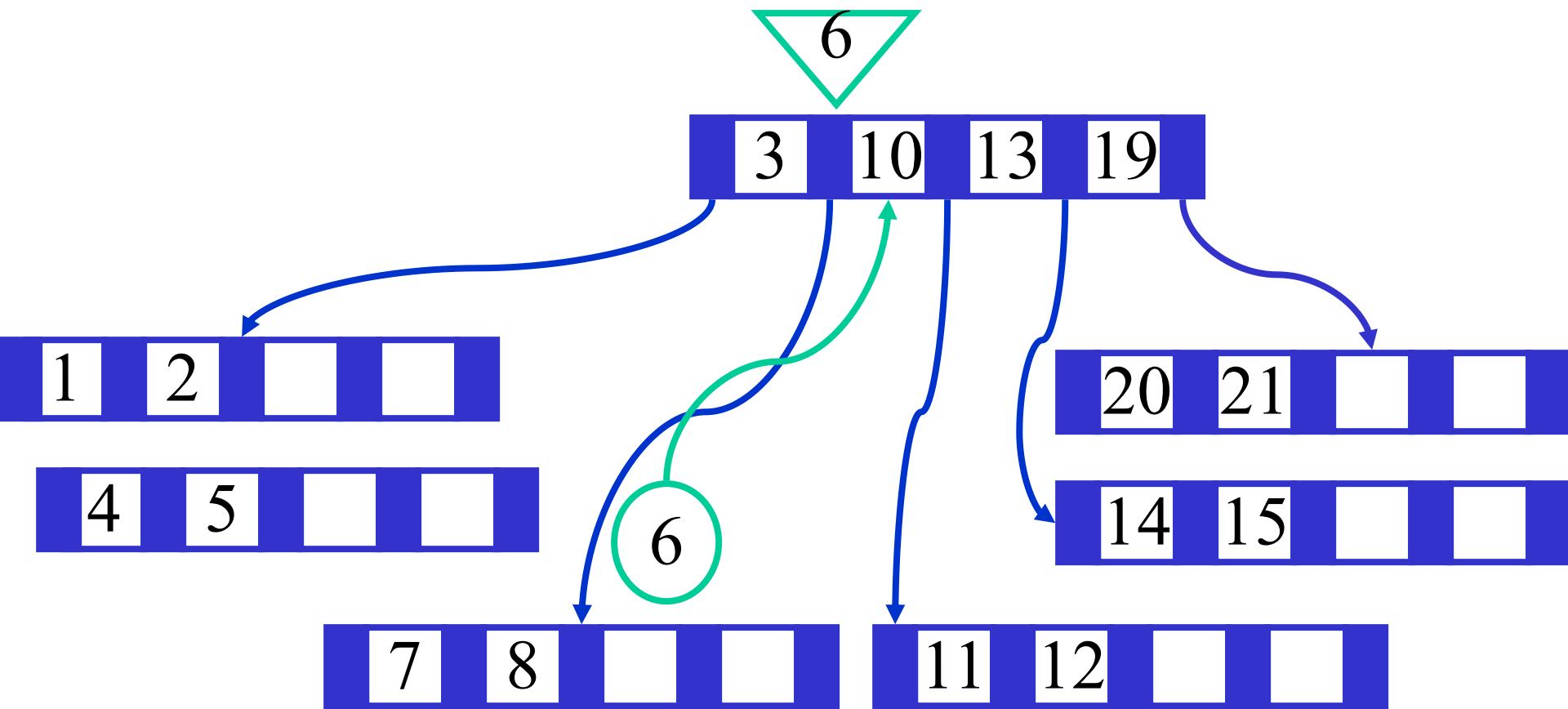


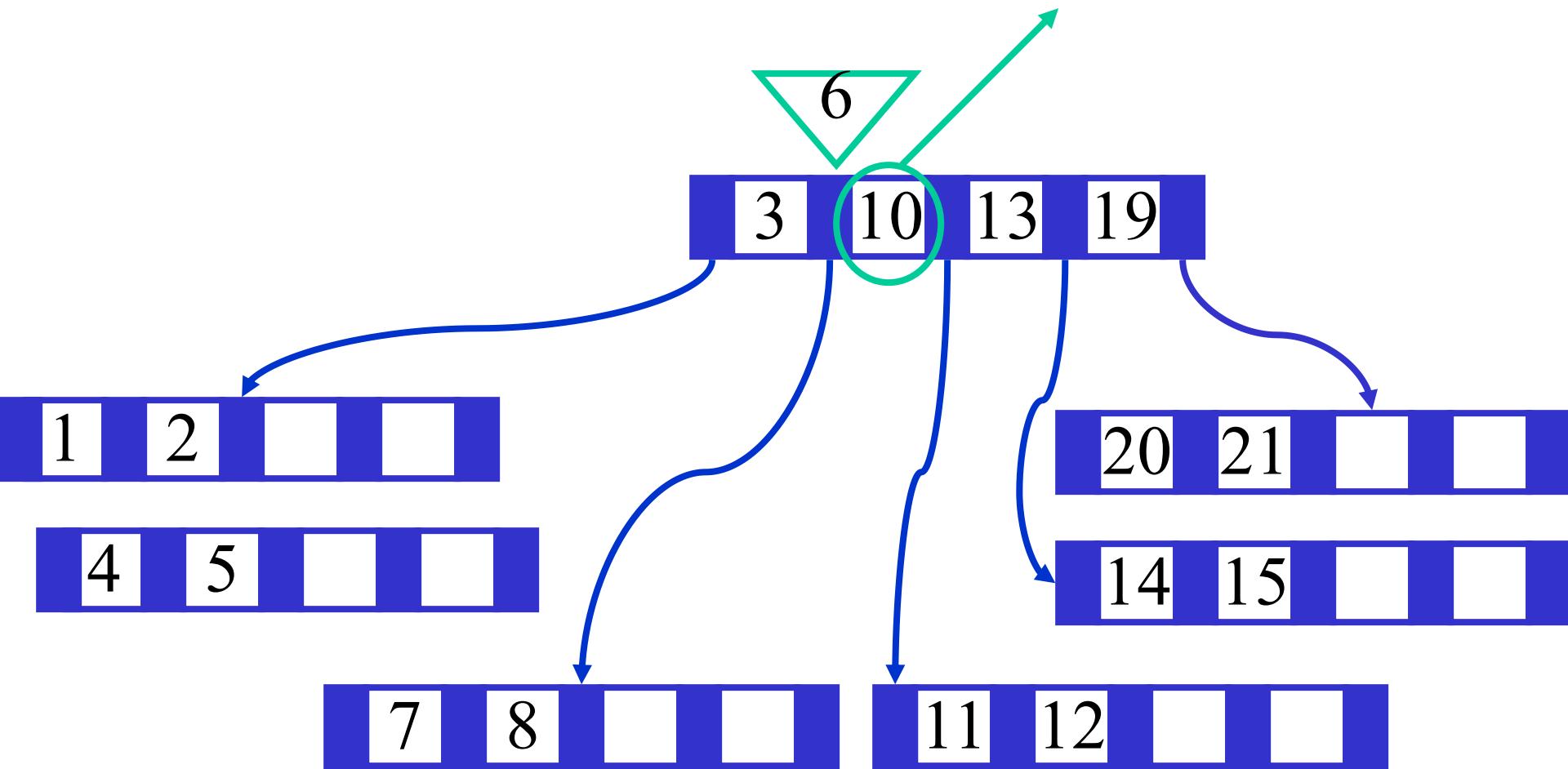


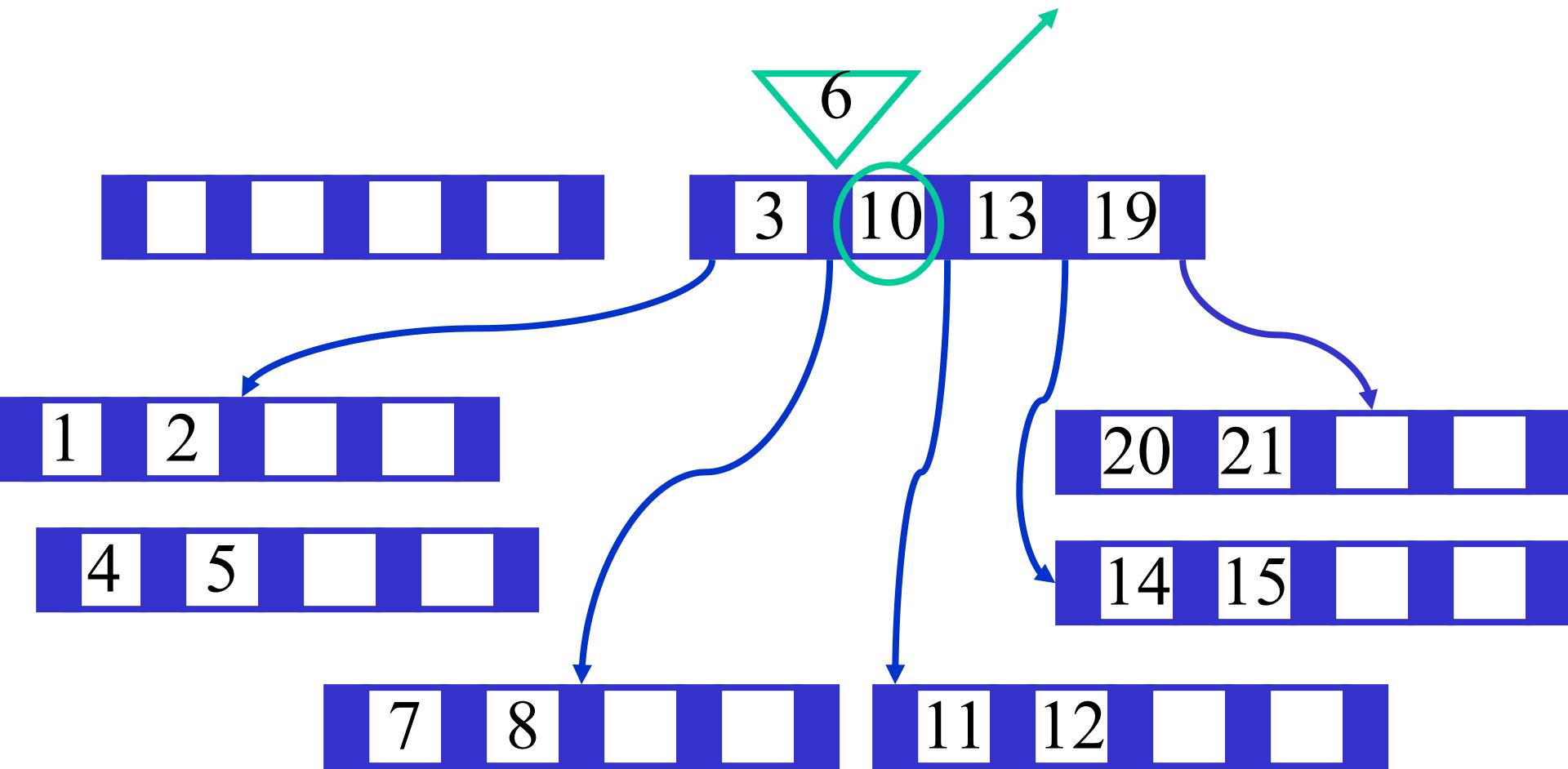


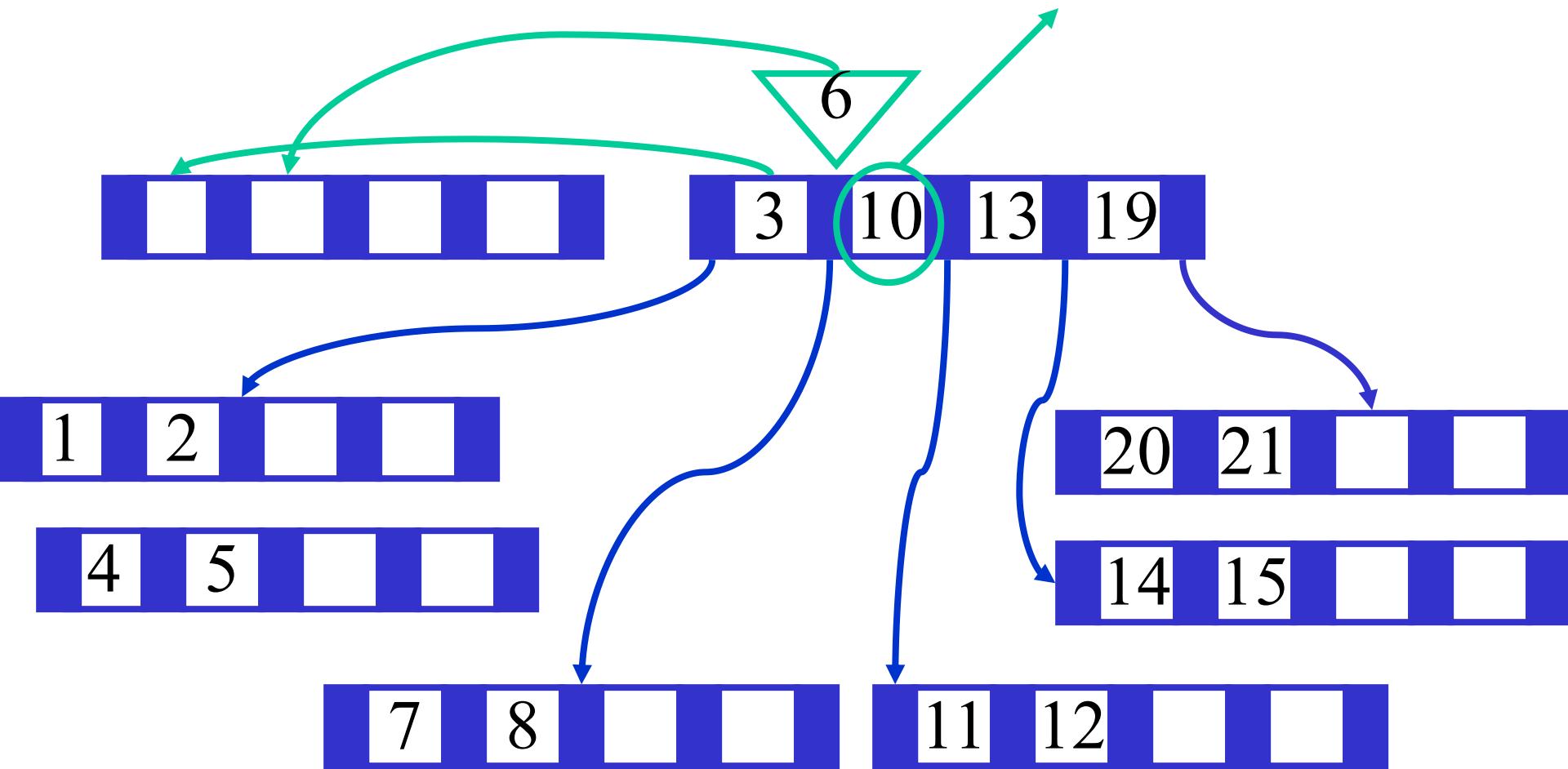


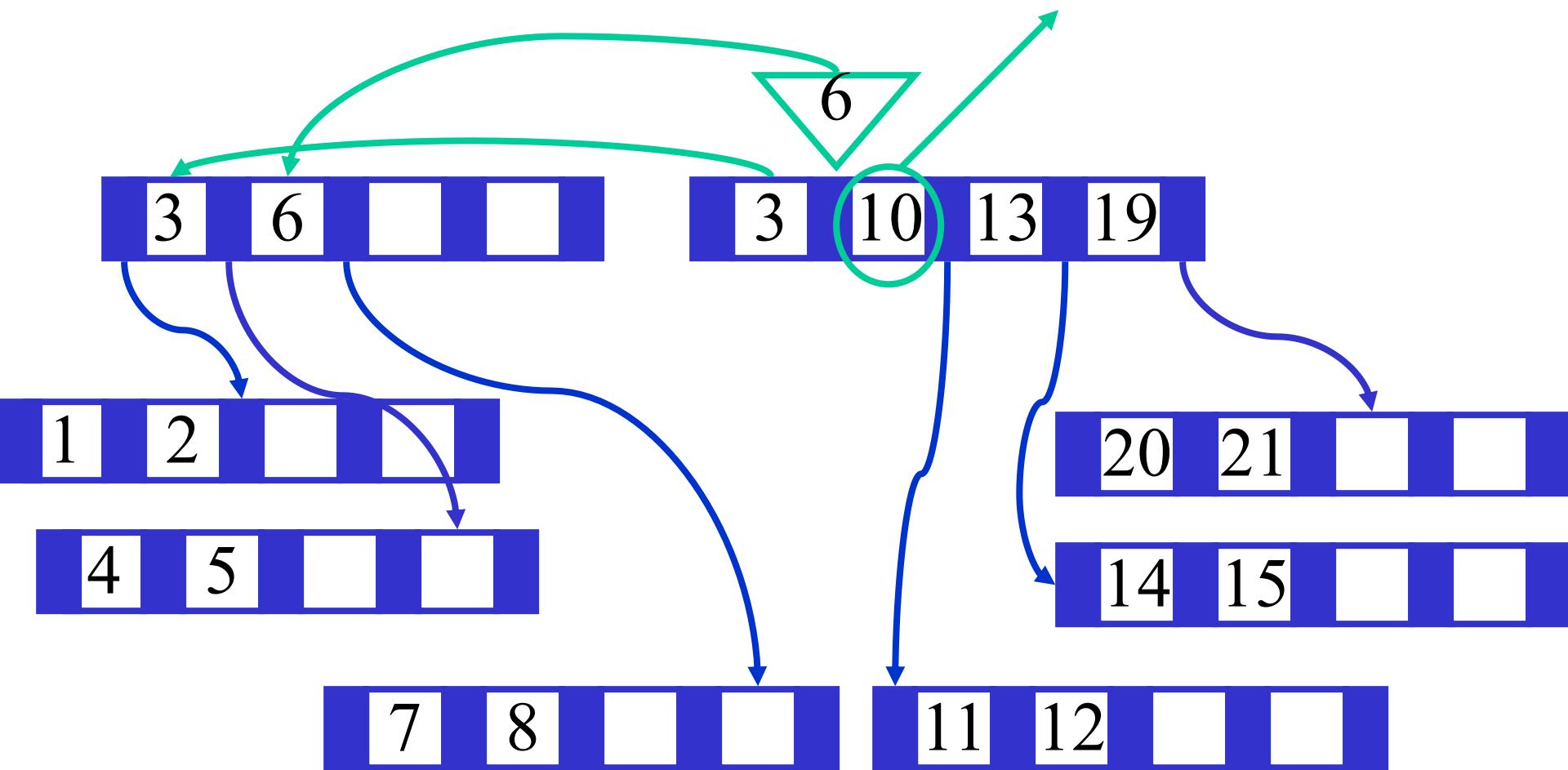


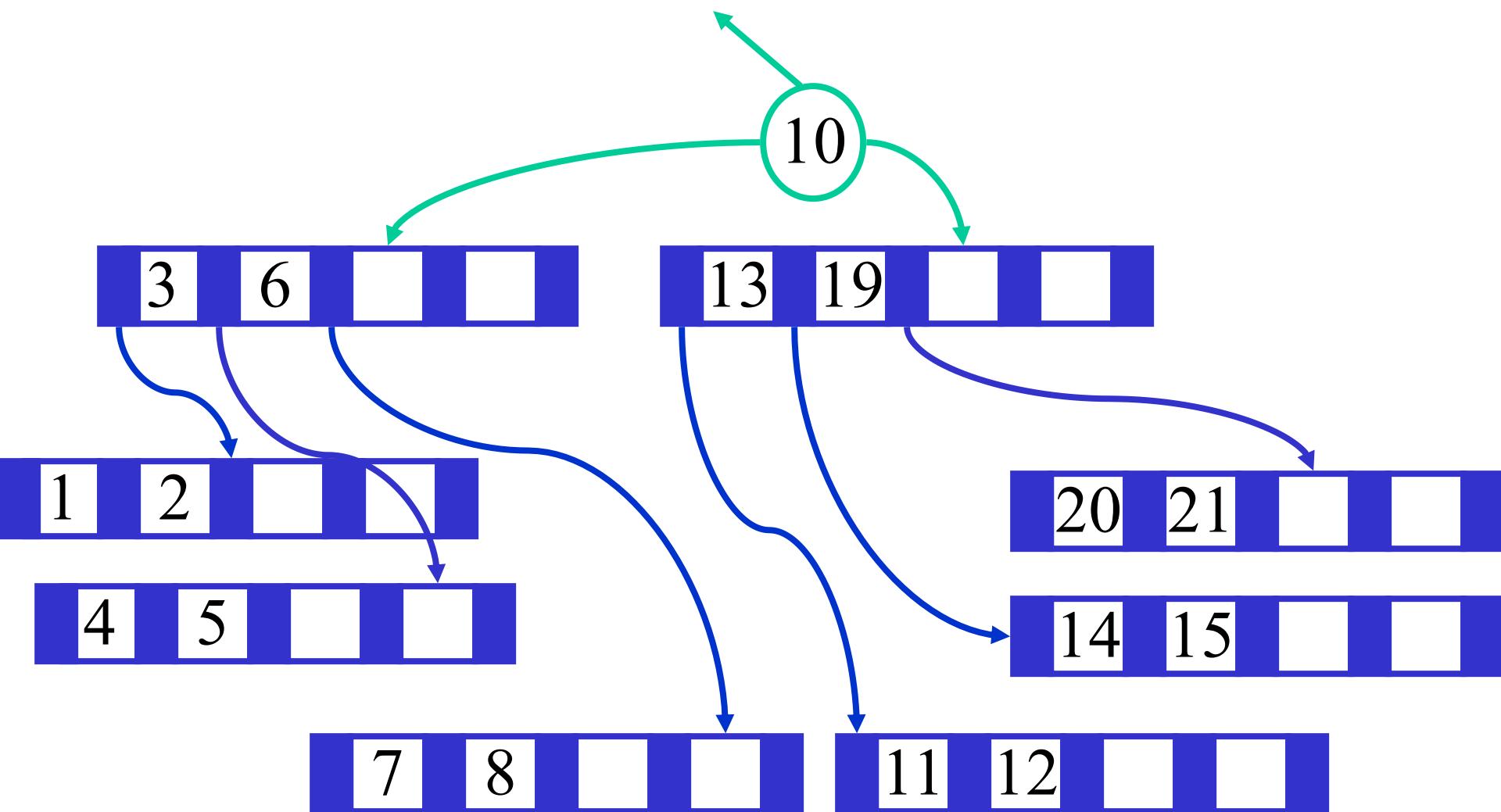


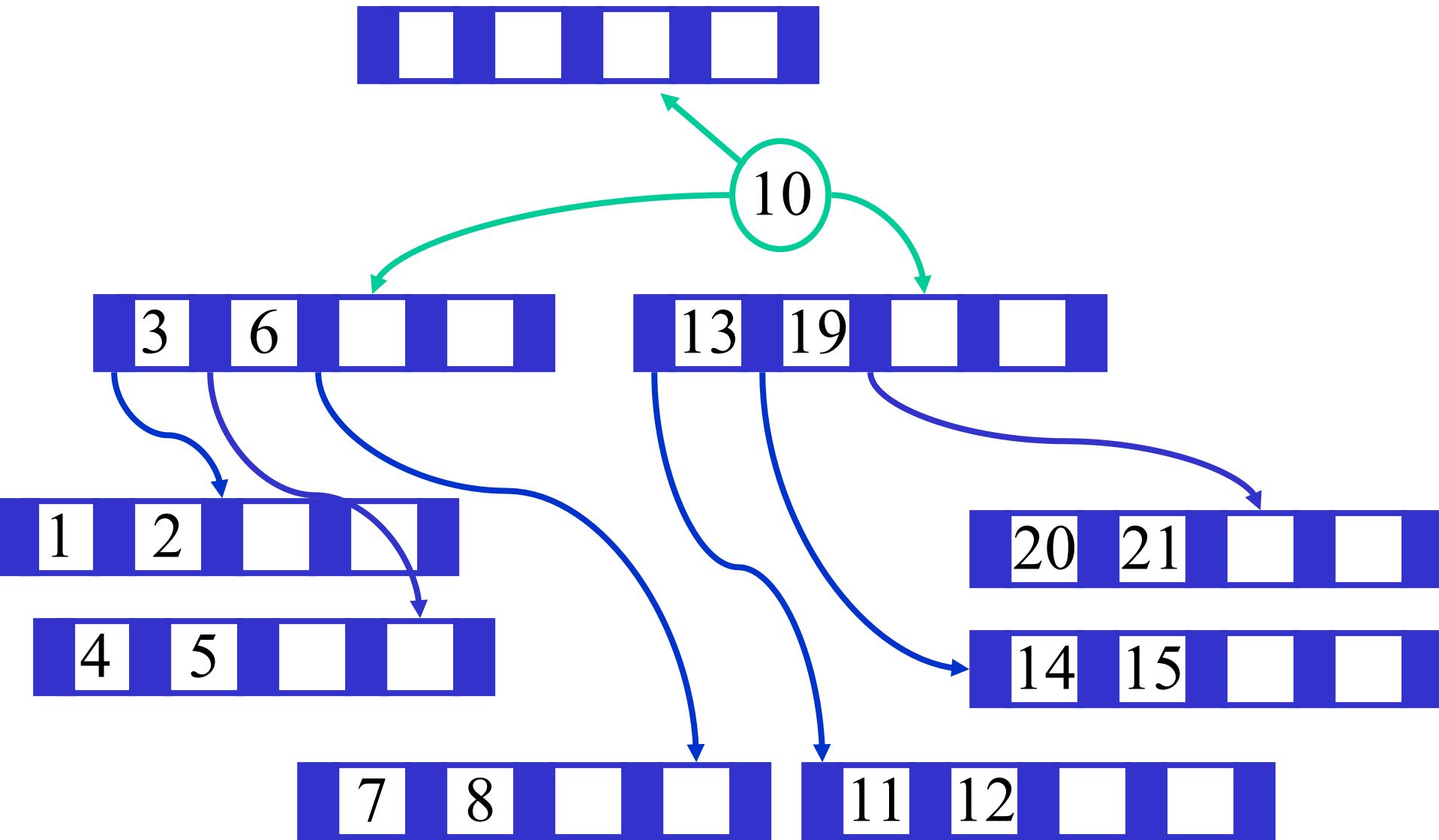


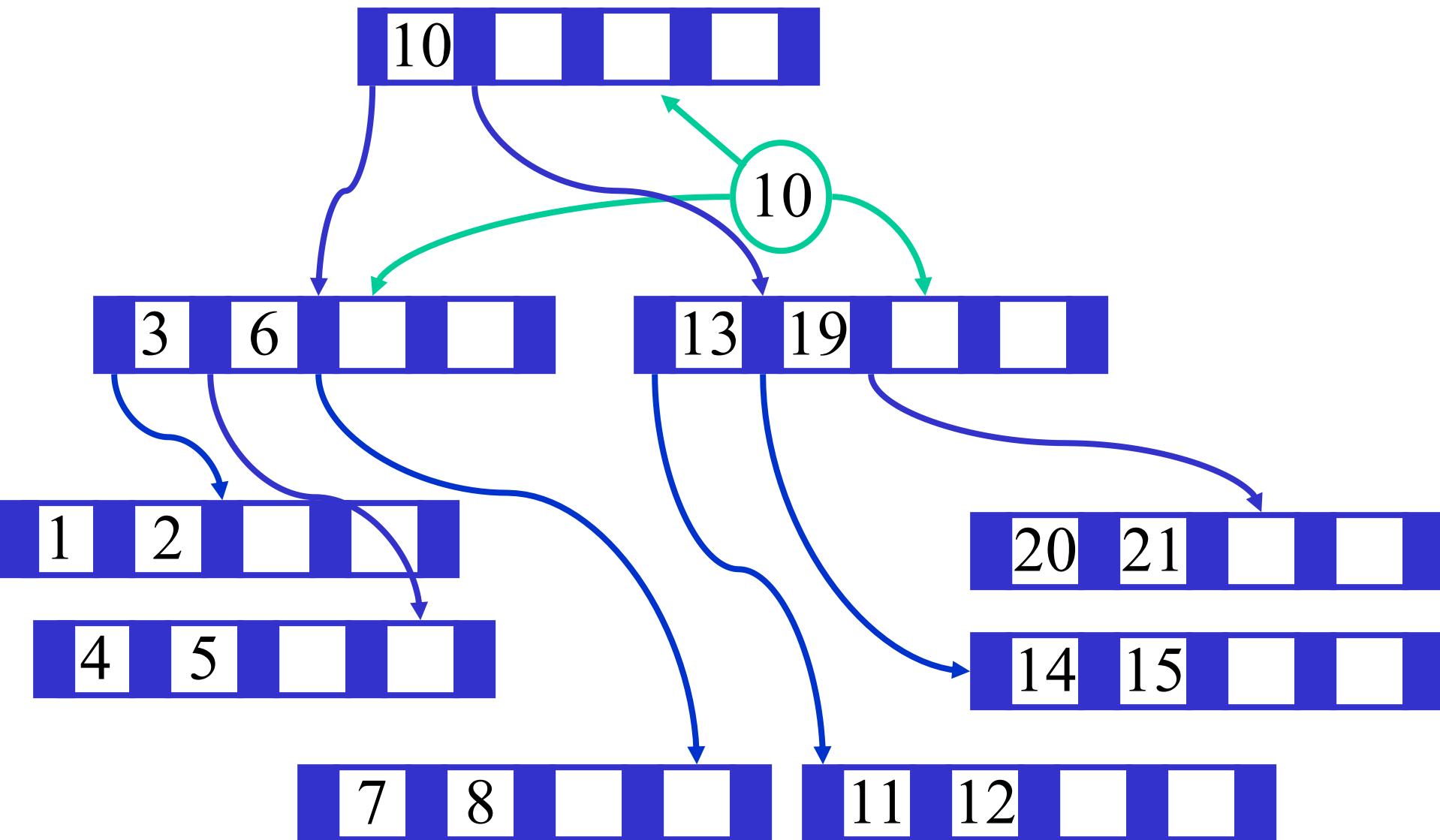


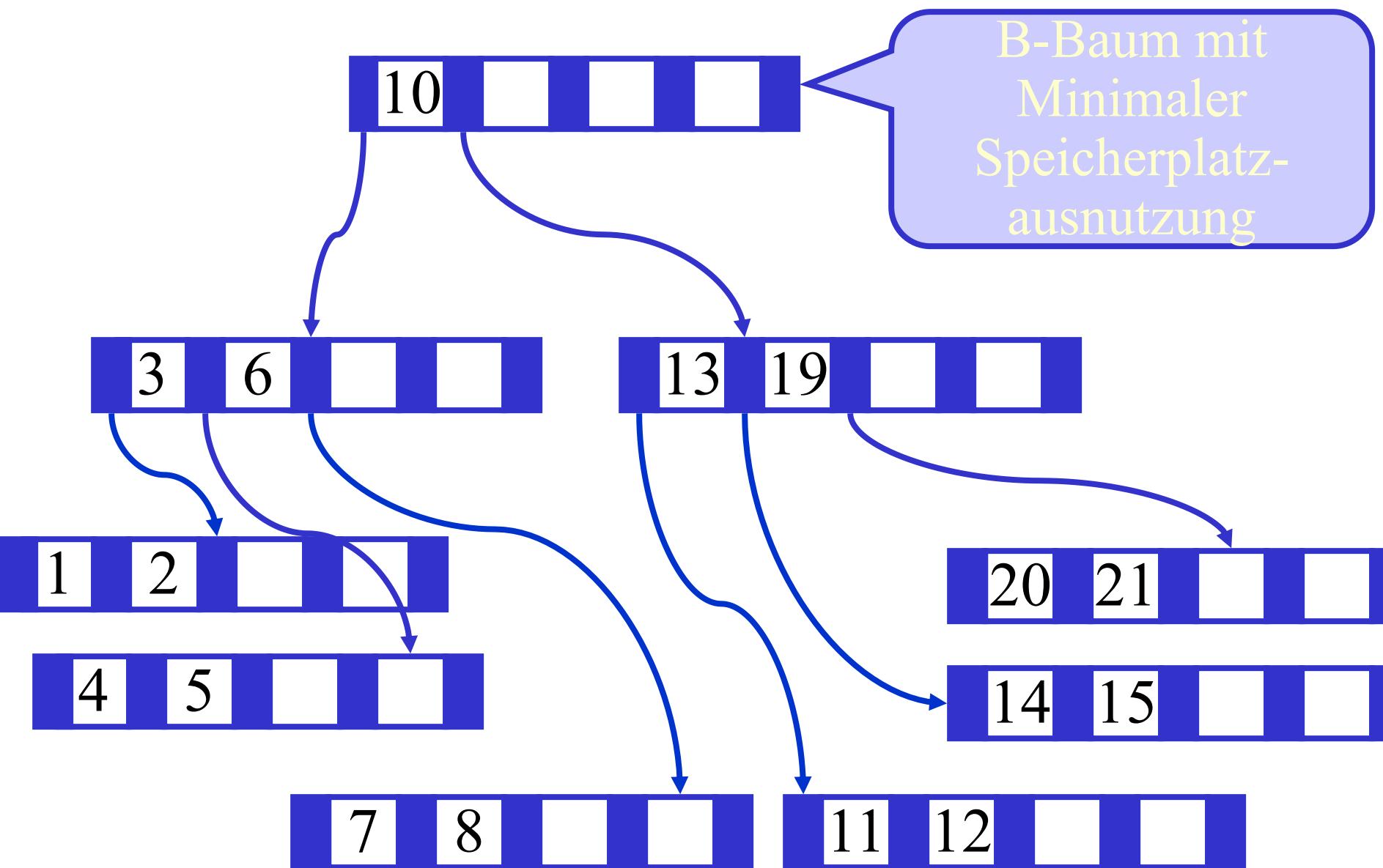


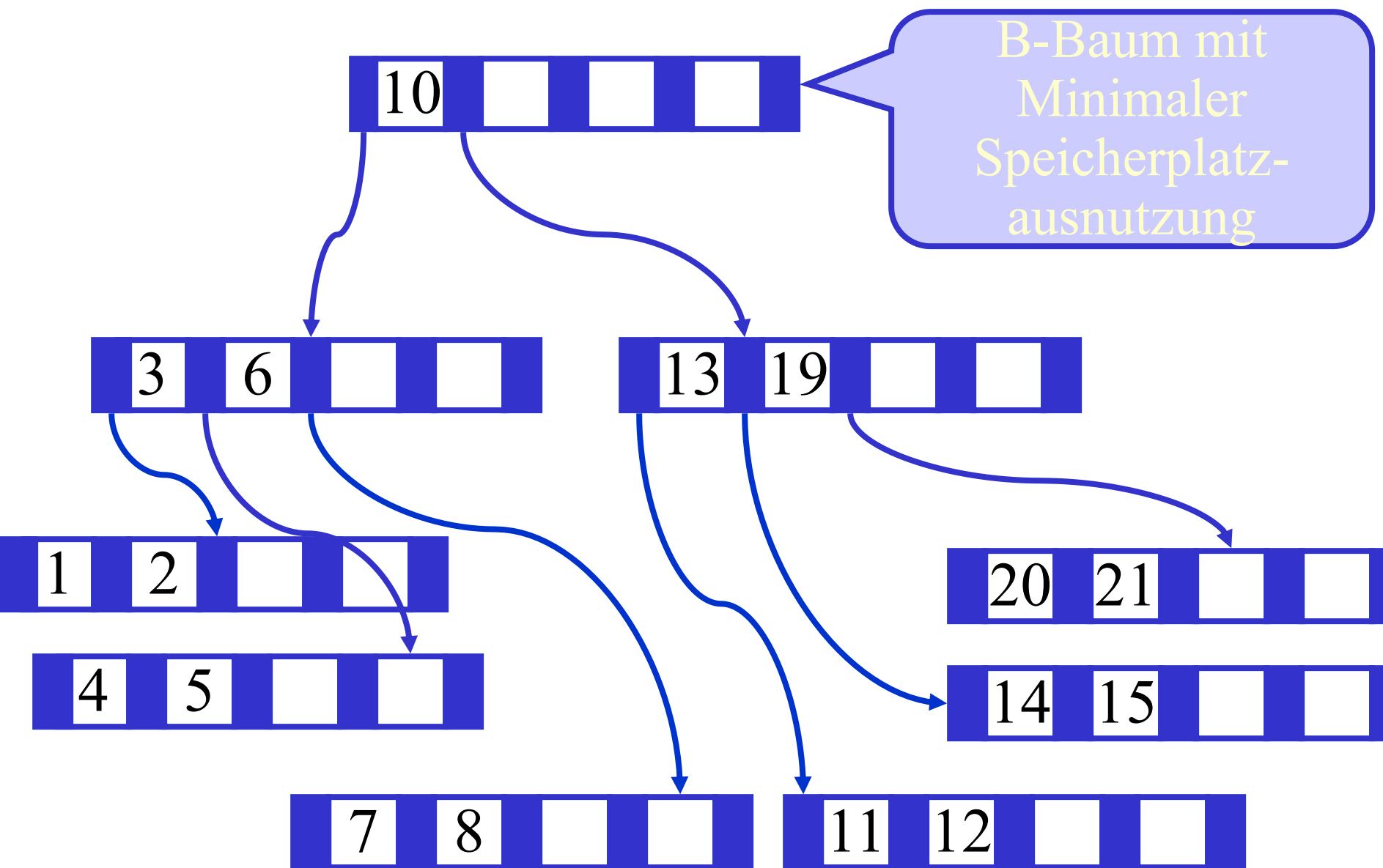


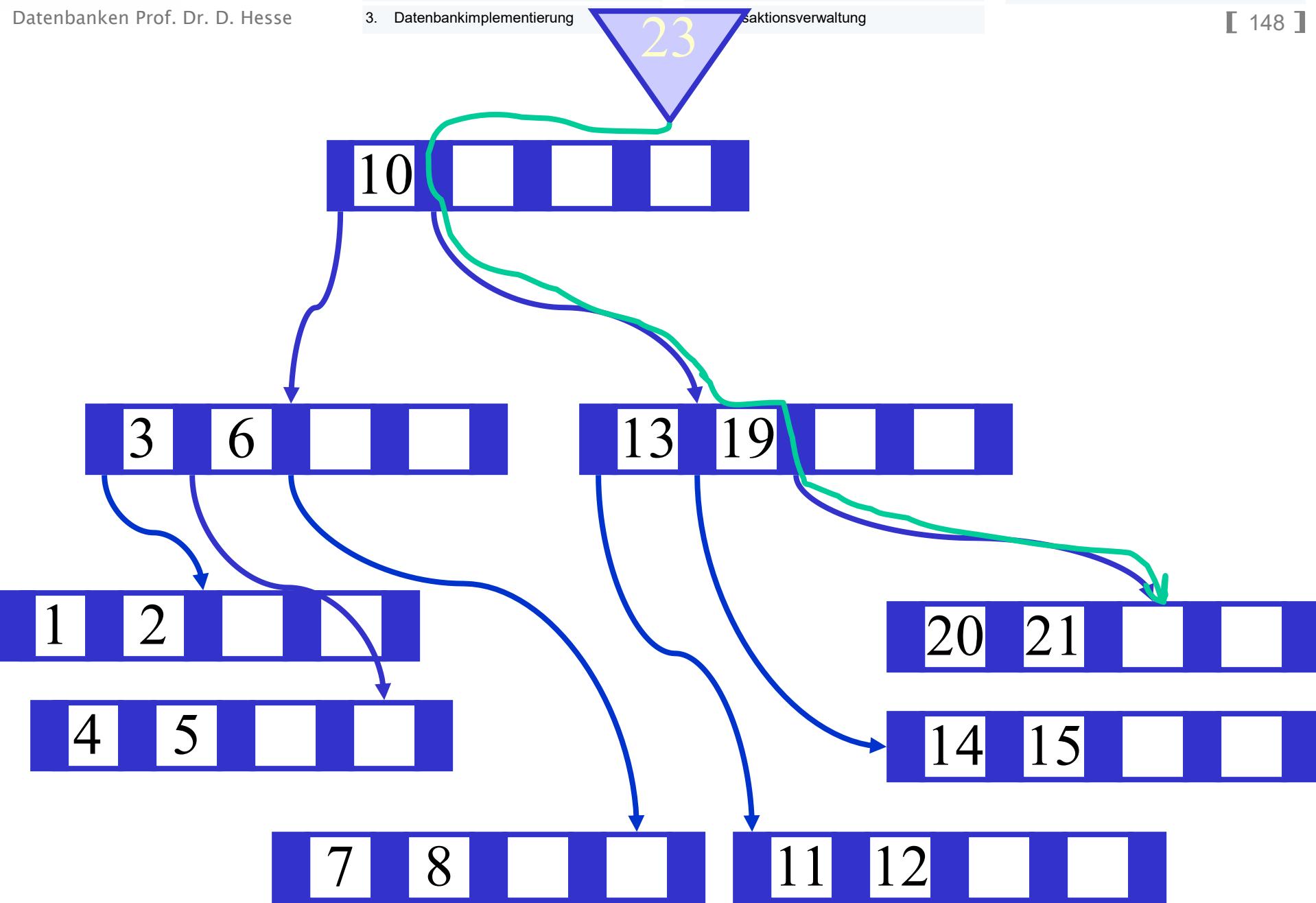


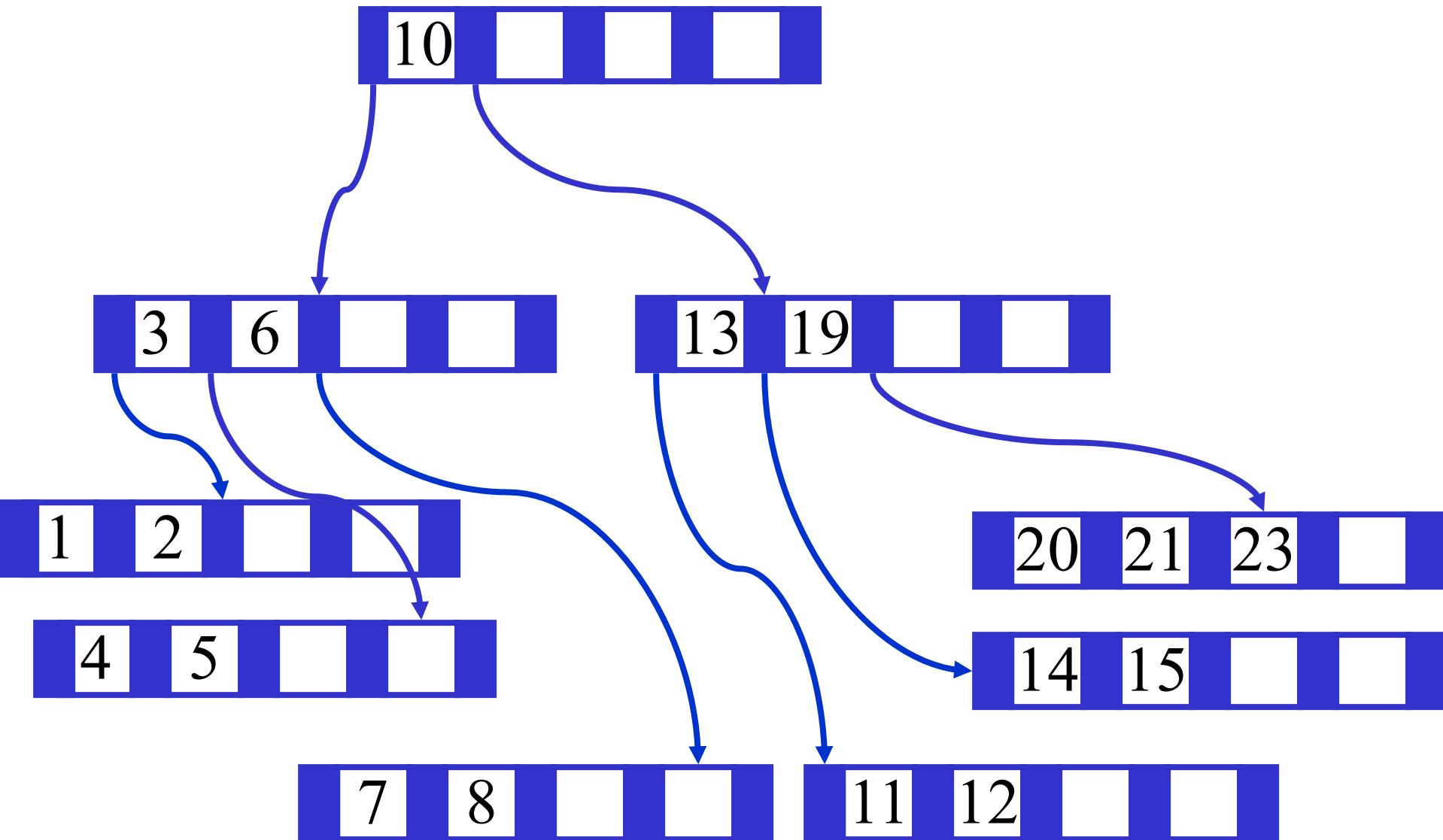


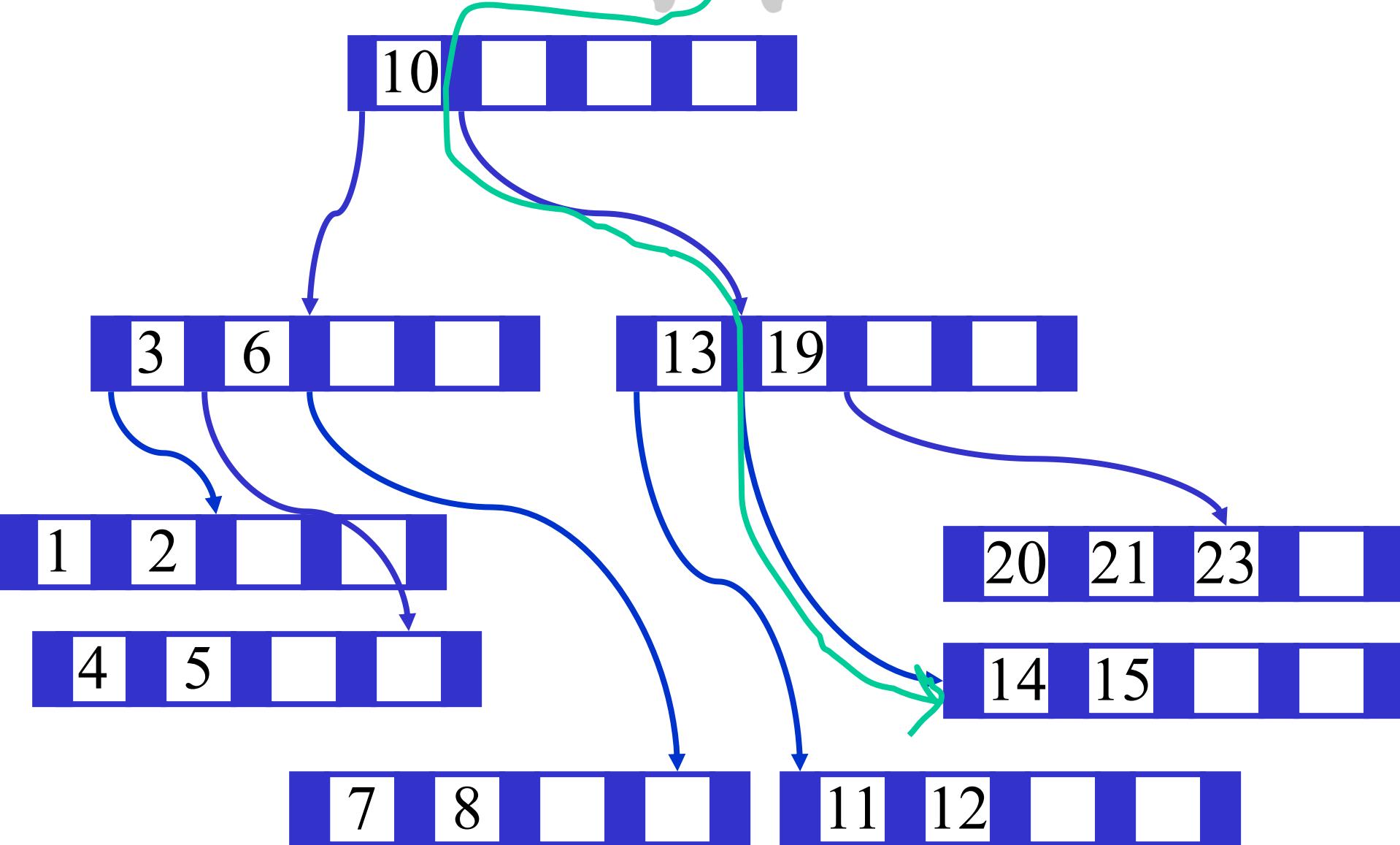


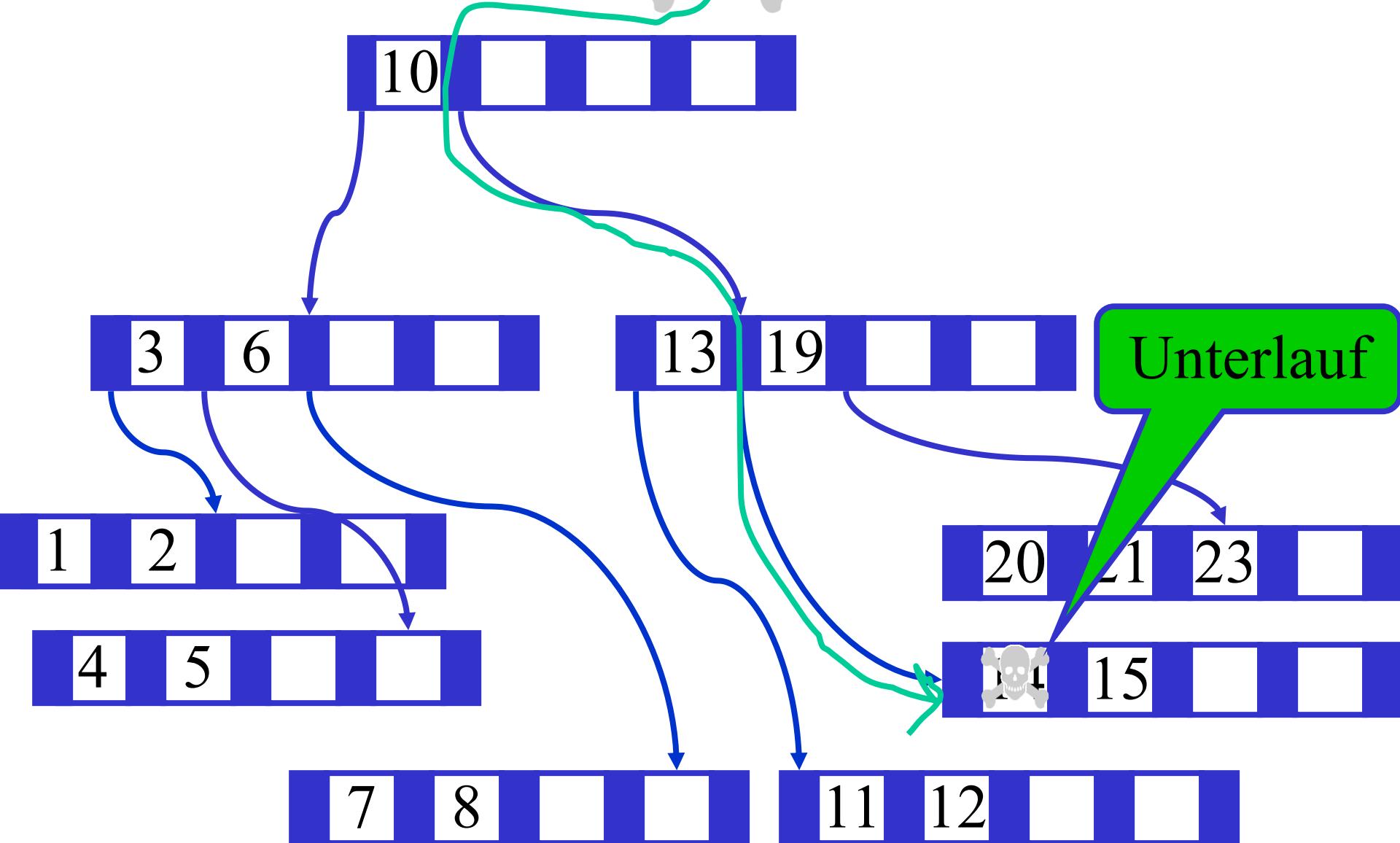


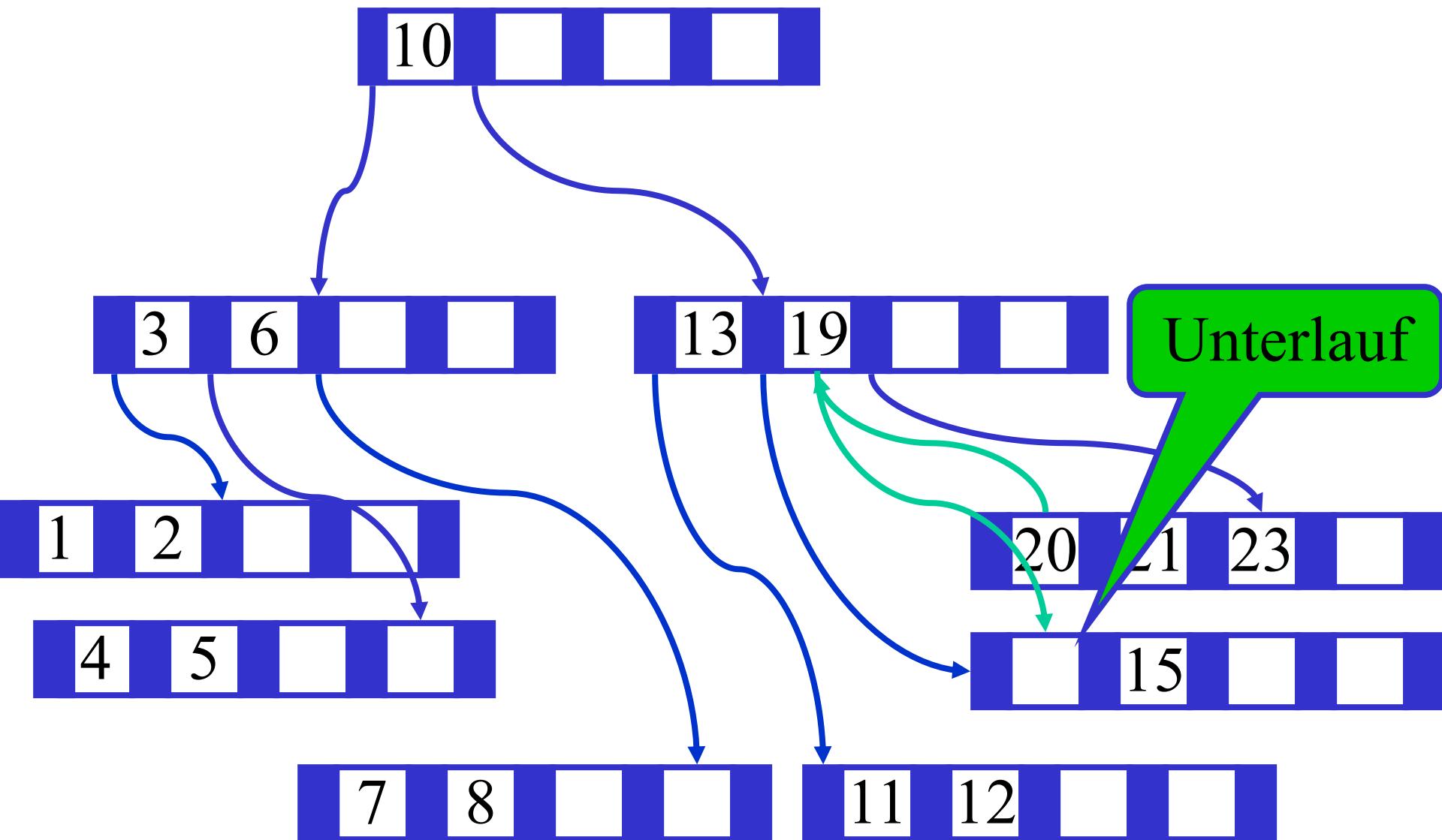


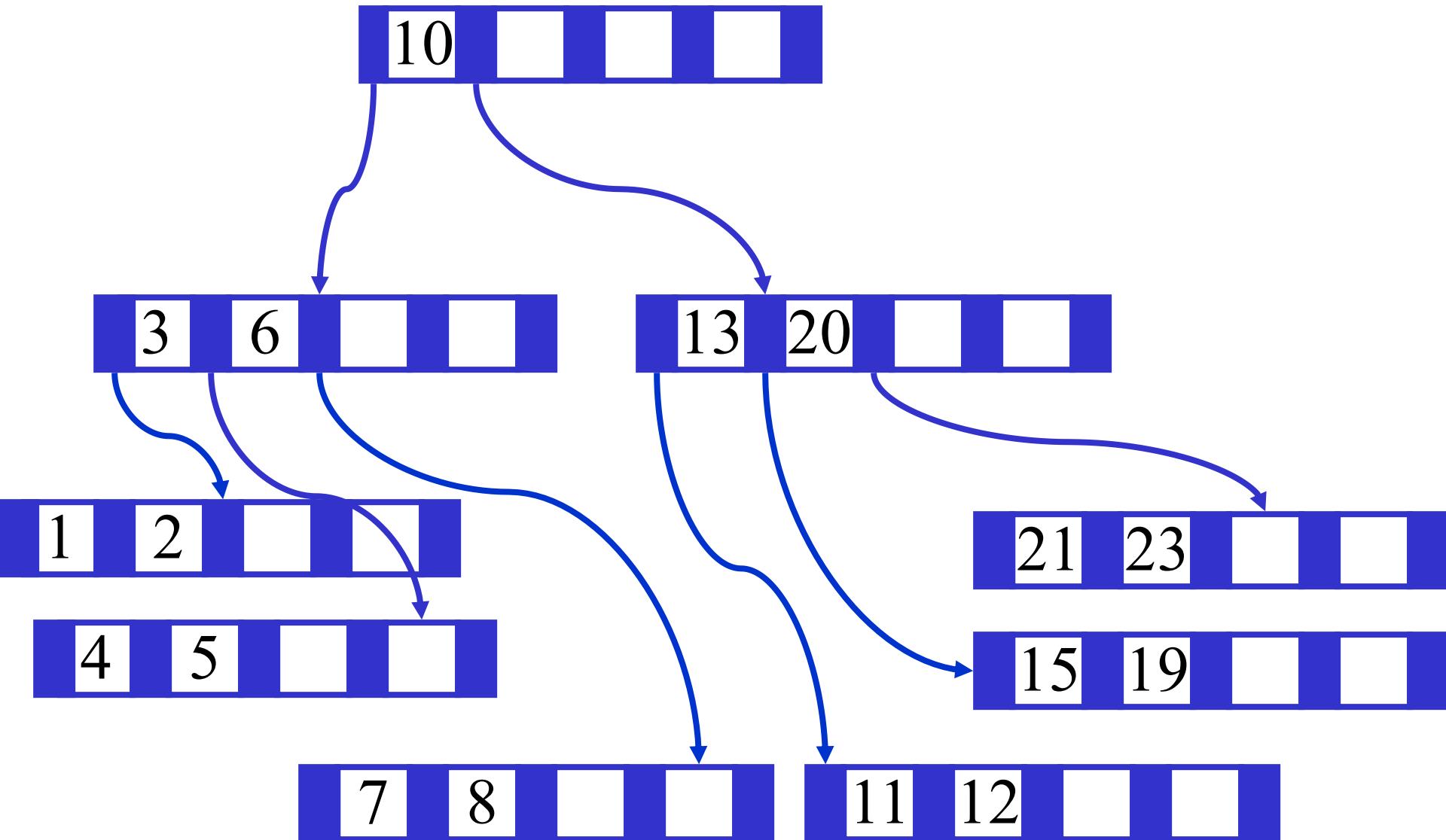


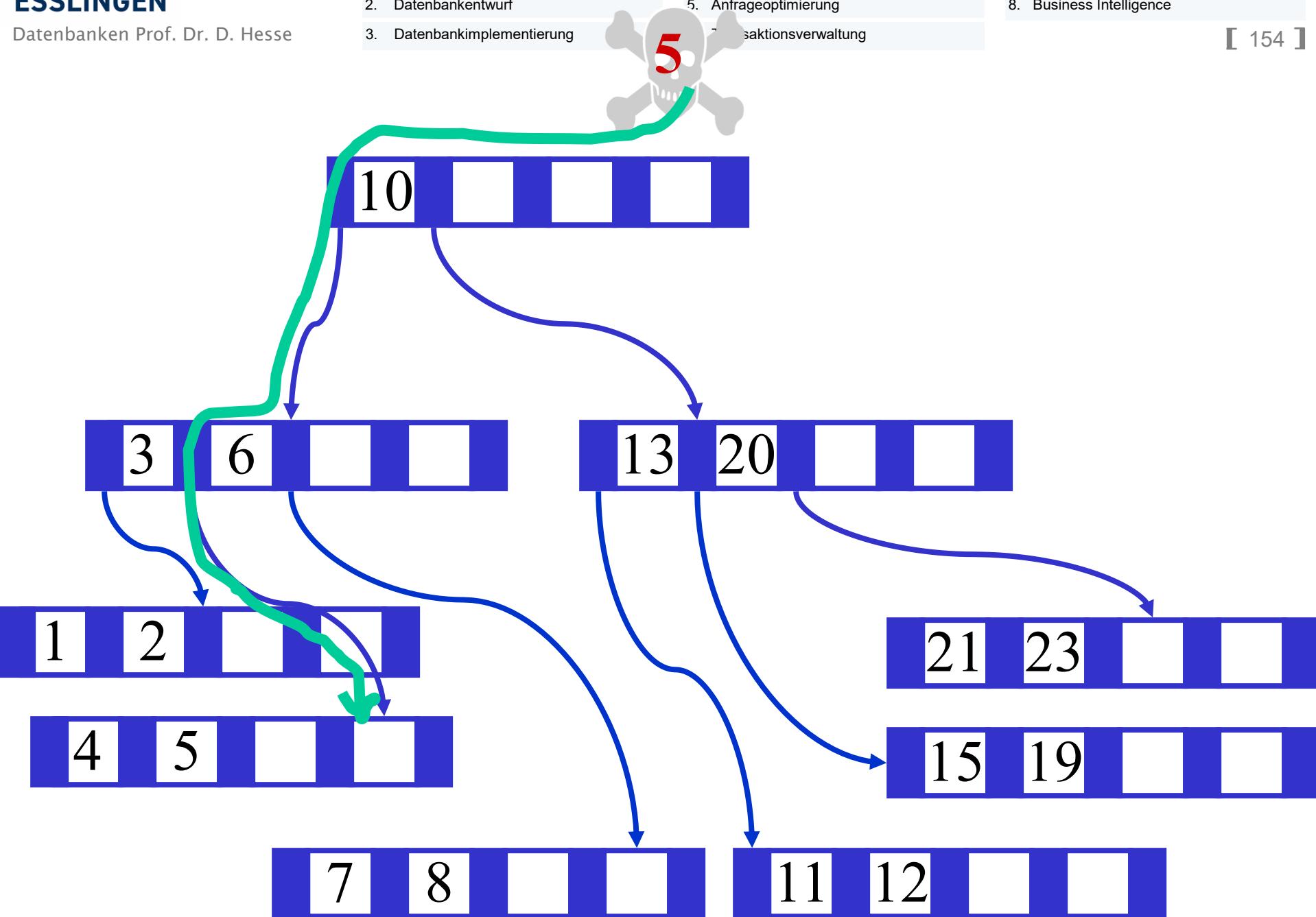


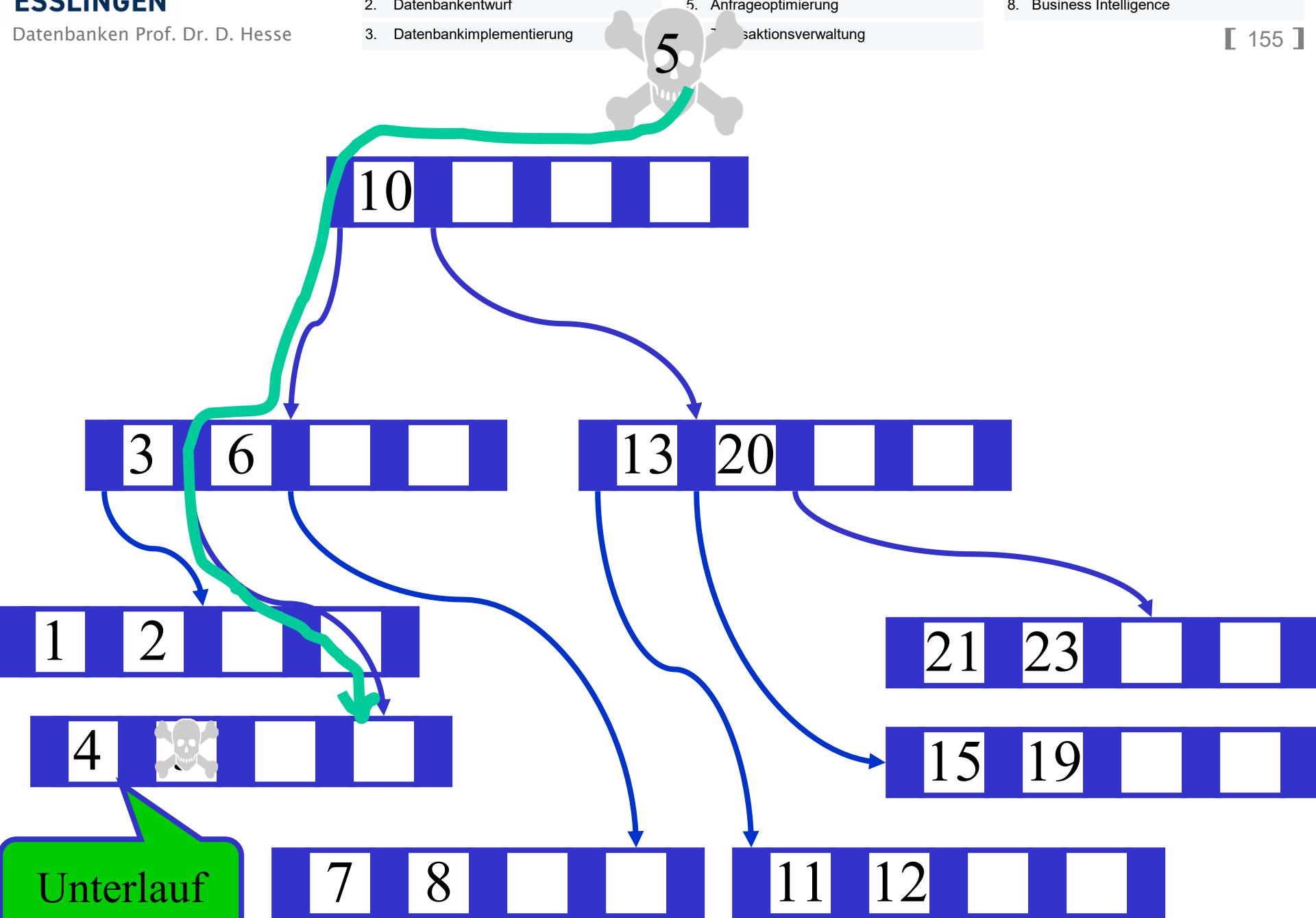


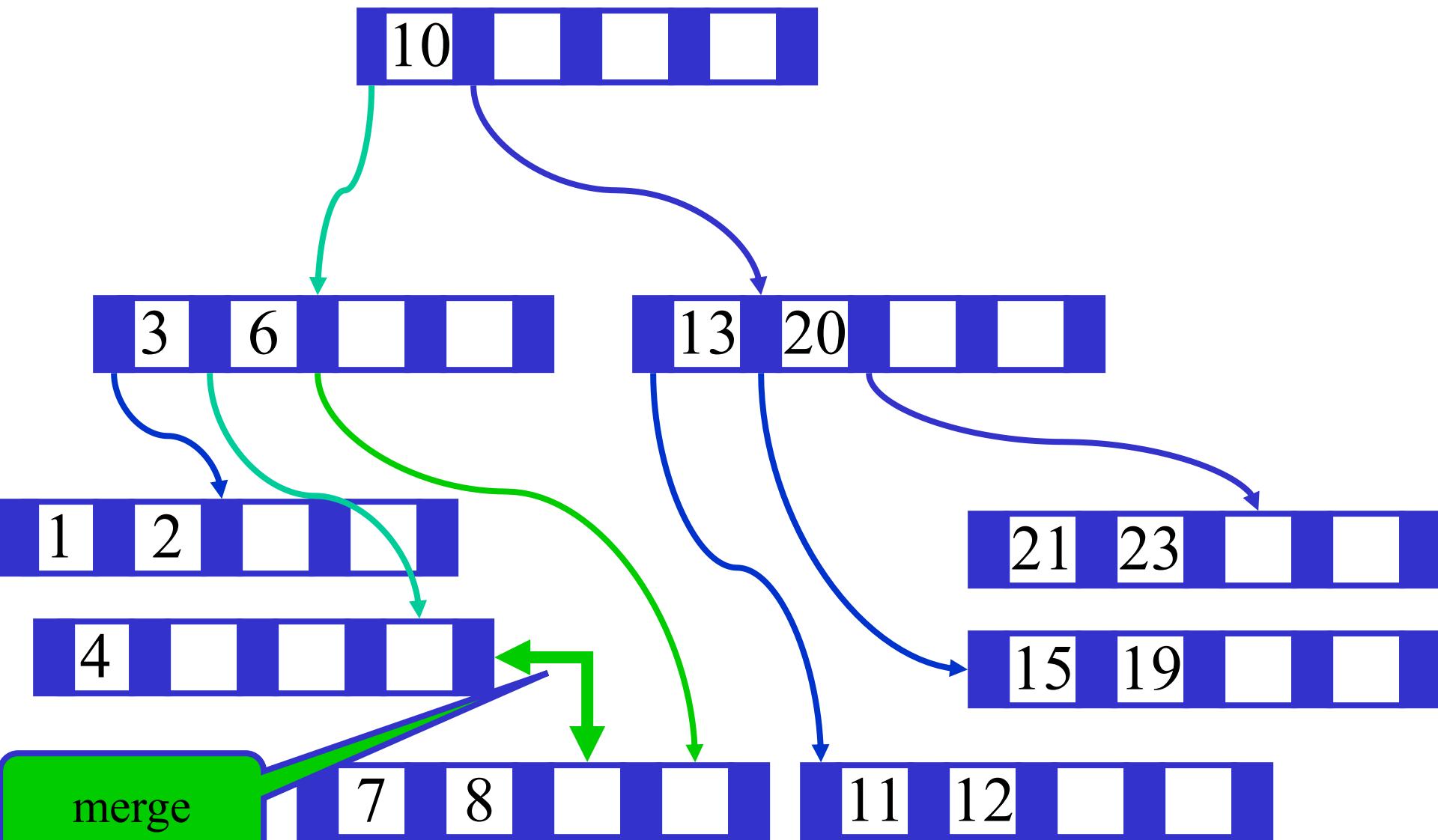


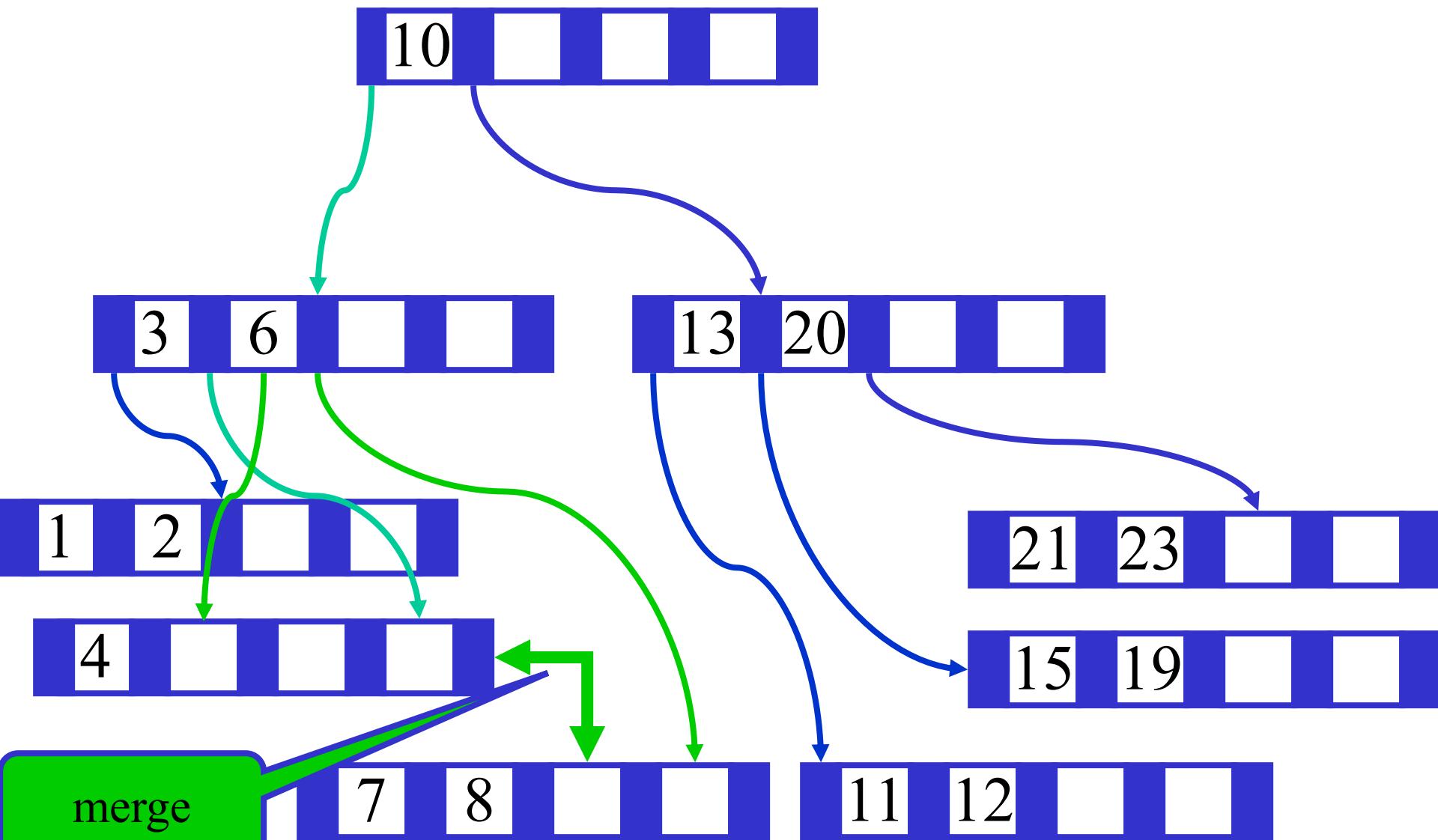


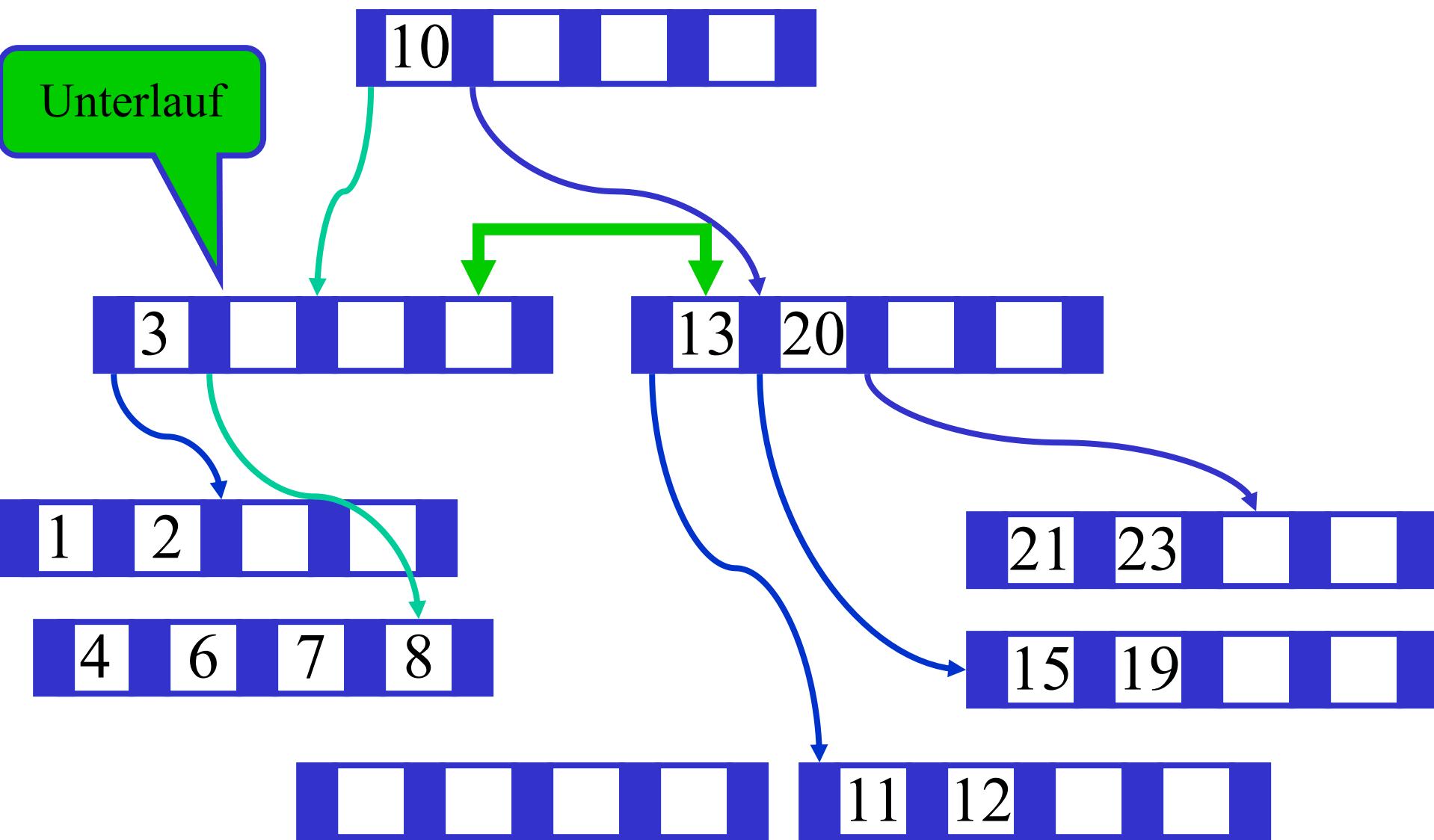


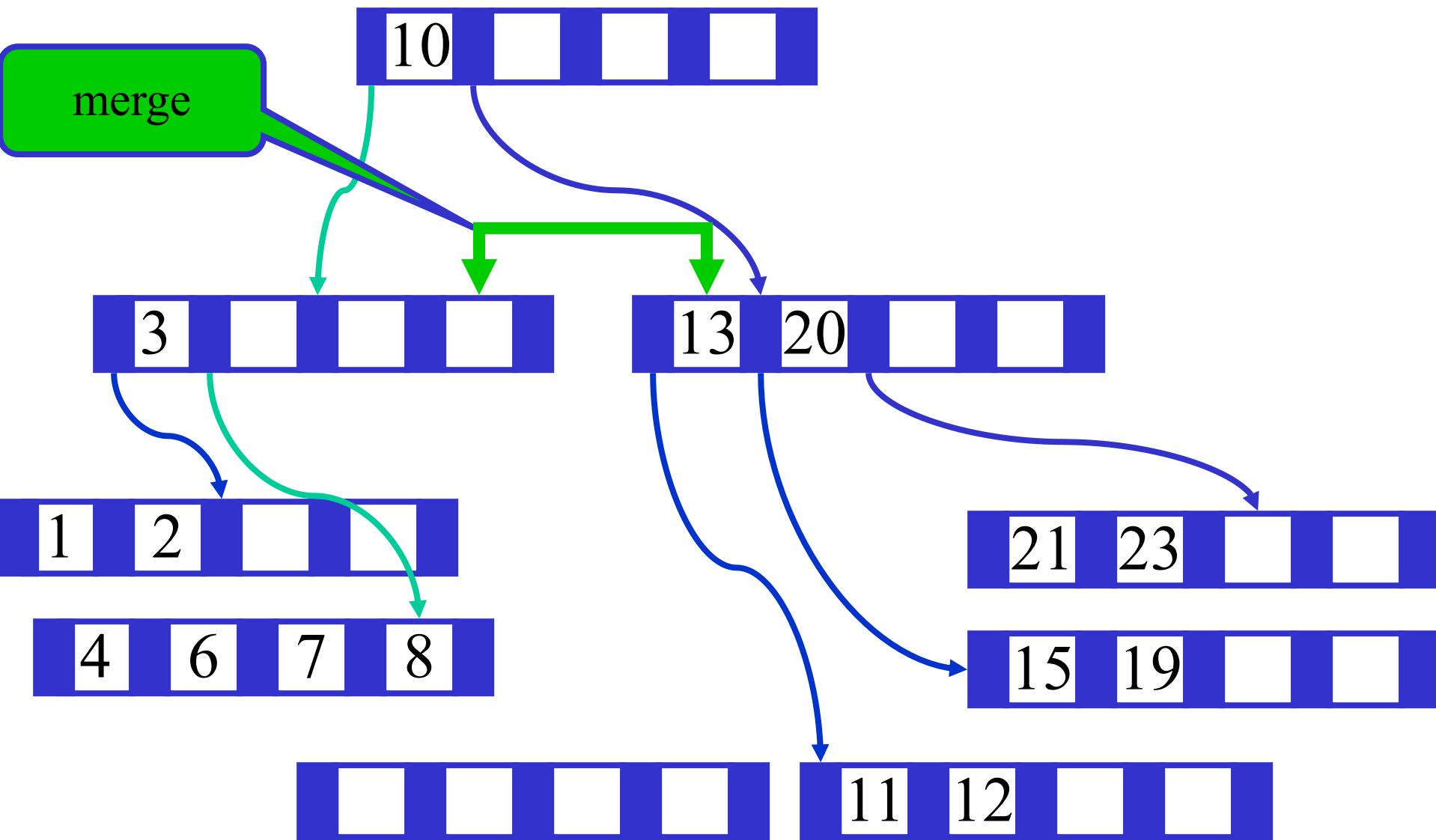


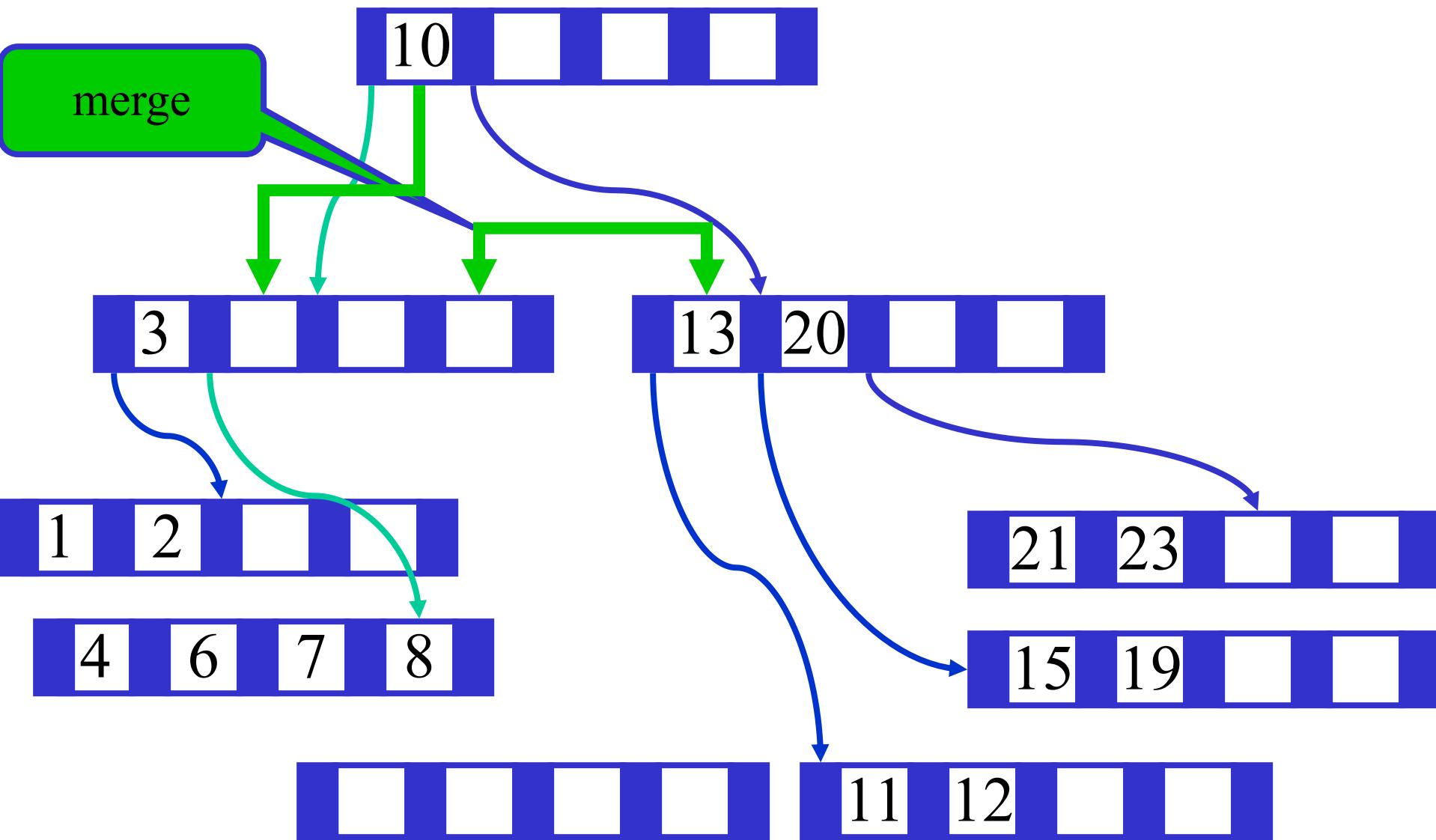


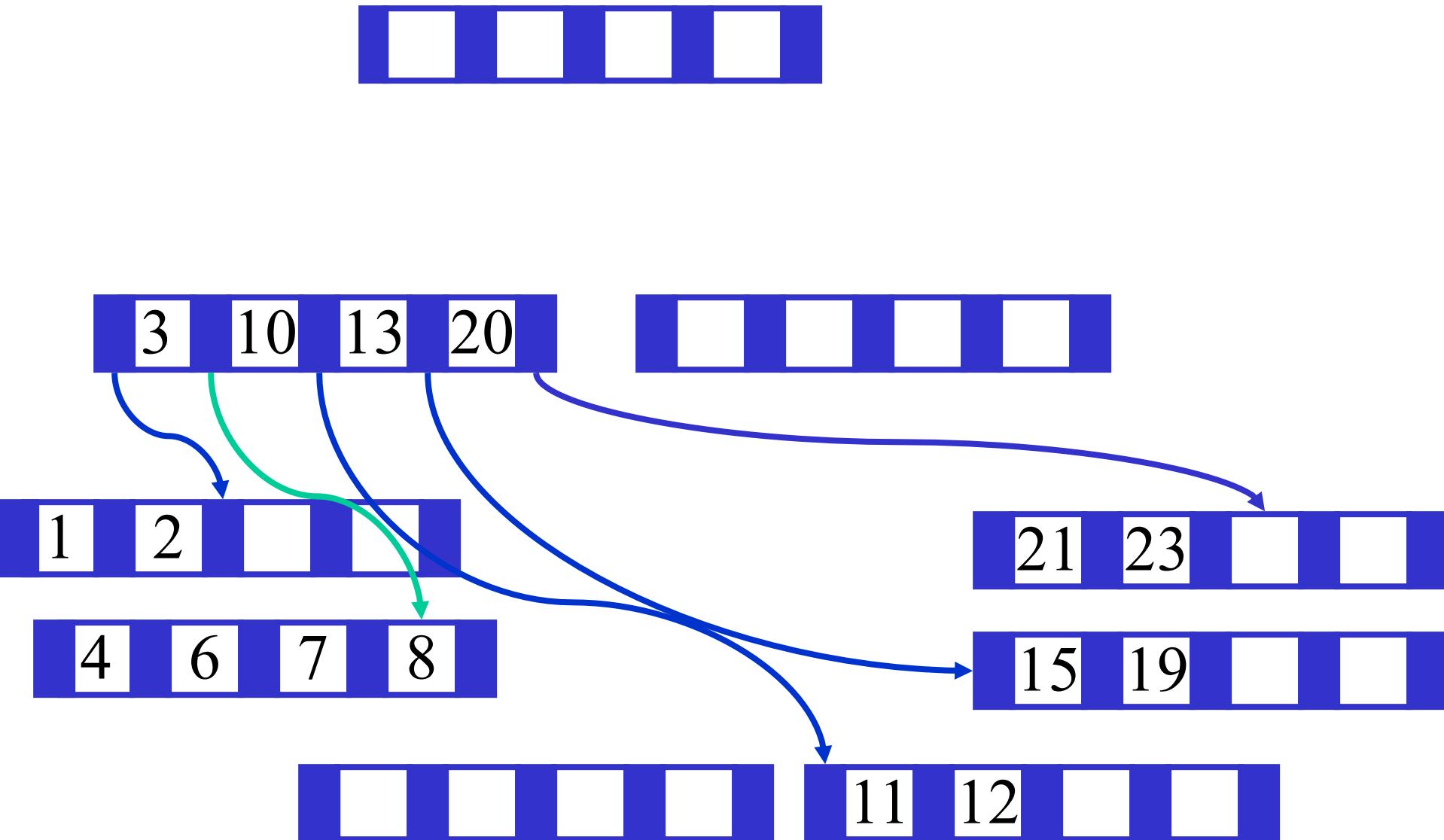


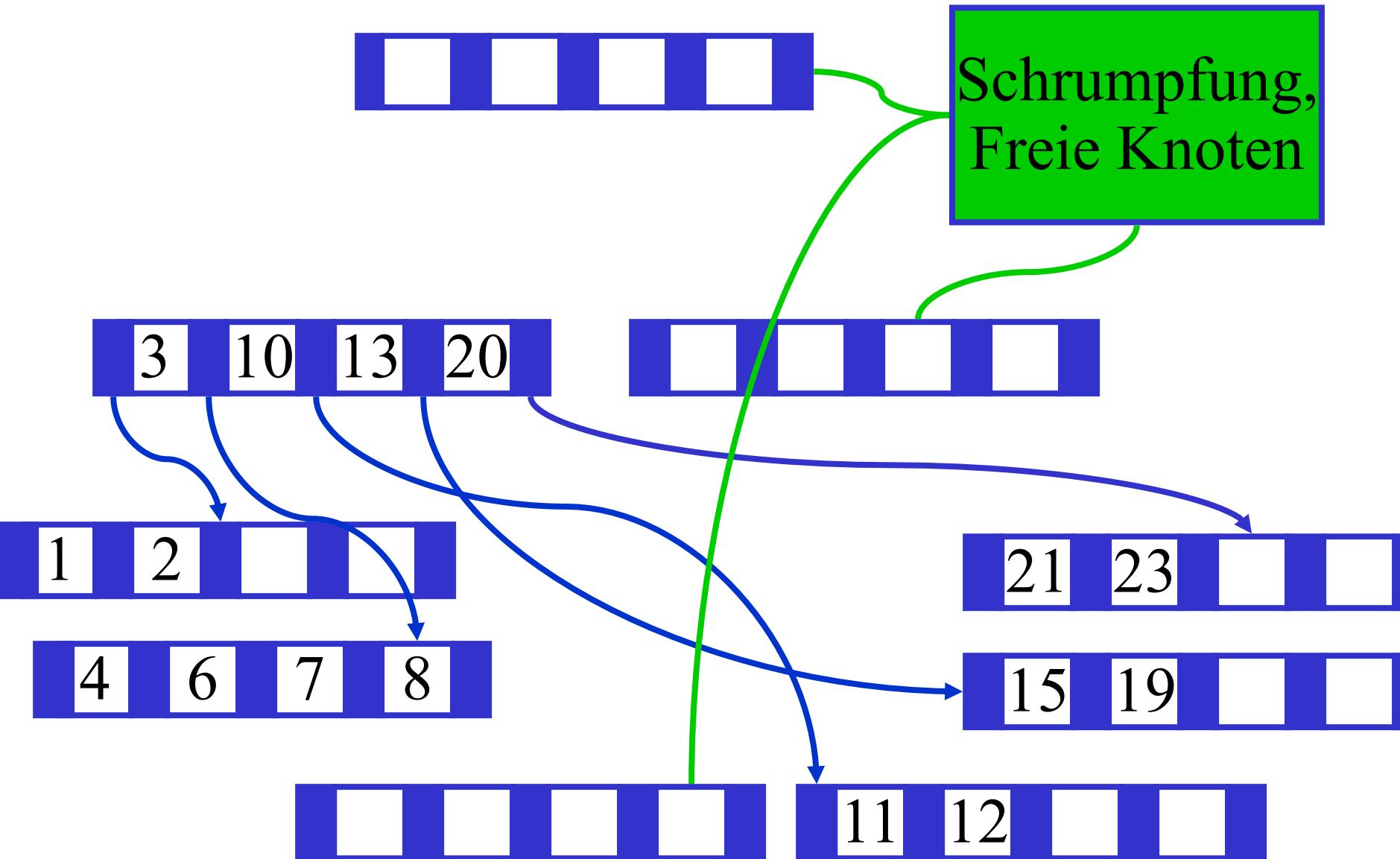






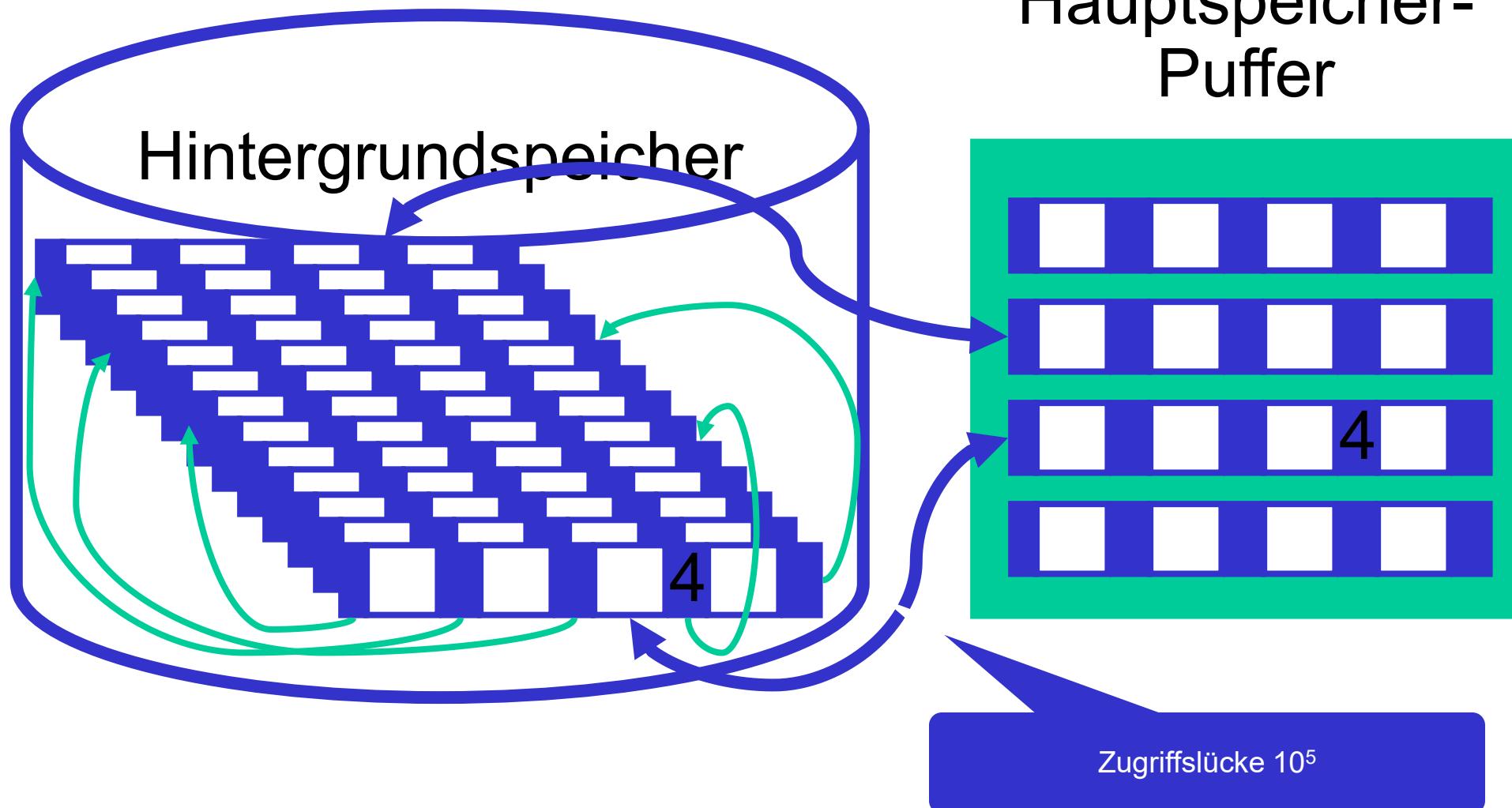


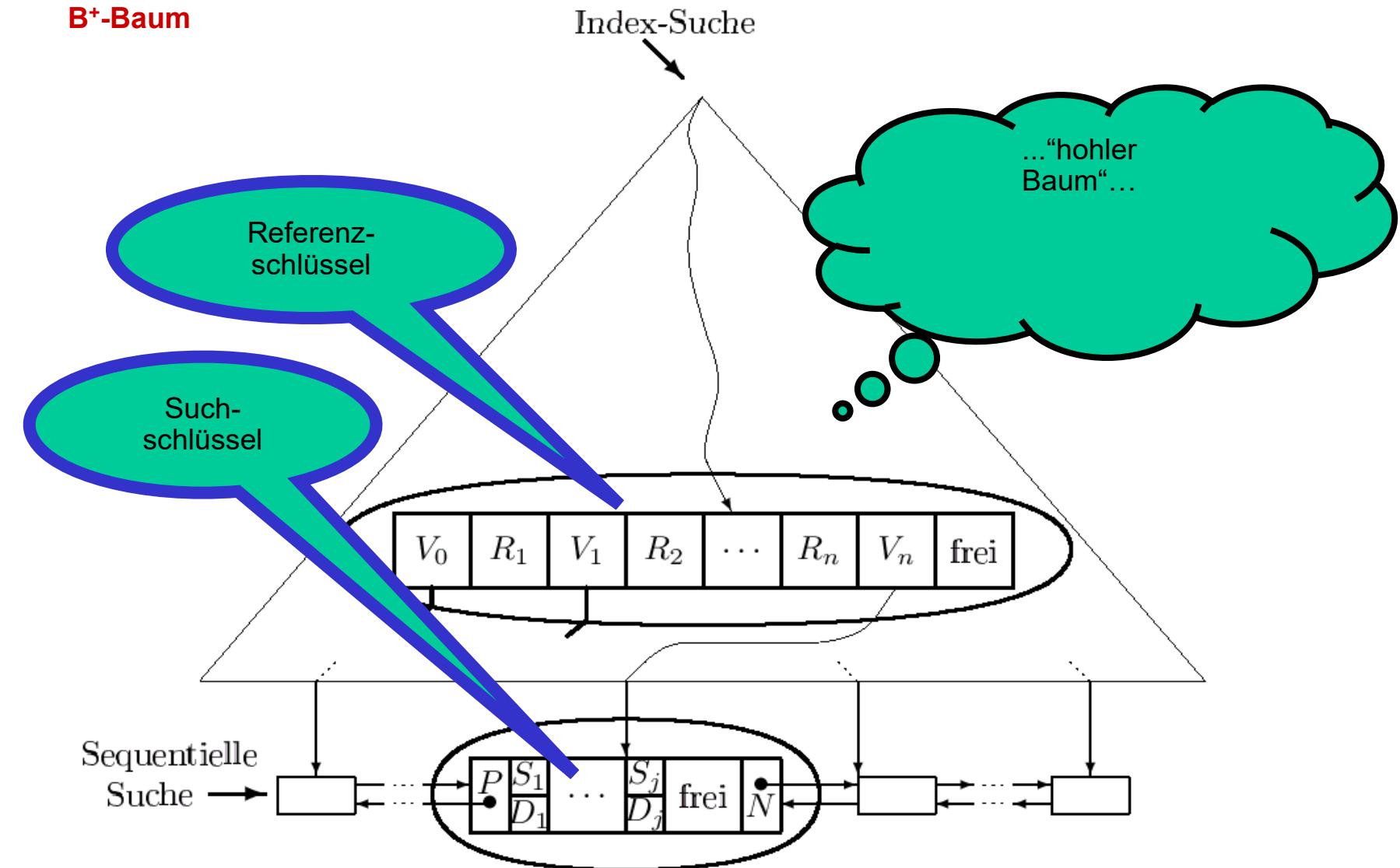




Zusammenspiel: Hintergrundspeicher <-> Hauptspeicher

Hauptspeicher- Puffer



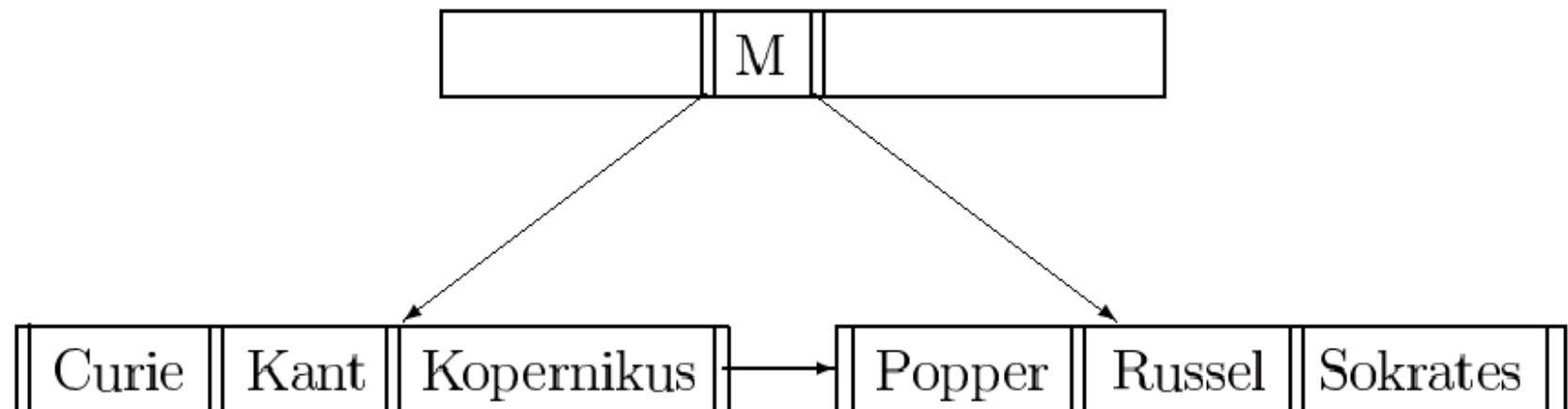
B⁺-Baum

Ein B^+ -Baum vom Typ (k, k^*) hat also folgende Eigenschaften:

1. Jeder Weg von der Wurzel zu einem Blatt hat die gleiche Länge.
2. Jeder Knoten – außer Wurzeln und Blättern – hat mindestens k und höchstens $2k$ Einträge. Blätter haben mindestens k^* und höchstens $2k^*$ Einträge. Die Wurzel hat entweder maximal $2k$ Einträge, oder sie ist ein Blatt mit maximal $2k^*$ Einträgen.
3. Jeder Knoten mit n Einträgen, außer den Blättern, hat $n + 1$ Kinder.
4. Seien R_1, \dots, R_n die Referenzschlüssel eines inneren Knotens (d.h. auch der Wurzel) mit $n + 1$ Kindern. Seien V_0, V_1, \dots, V_n die Verweise auf diese Kinder.
 - (a) V_0 verweist auf den Teilbaum mit Schlüsseln kleiner oder gleich R_1 .
 - (b) V_i ($i = 1, \dots, n - 1$) verweist auf den Teilbaum, dessen Schlüssel zwischen R_i und R_{i+1} liegen (einschließlich R_{i+1}).
 - (c) V_n verweist auf den Teilbaum mit Schlüsseln größer als R_n .

Präfix-B⁺-Bäume

- Es werden nur *Referenzschlüssel* benötigt.



- beliebiger Referenzschlüssel R mit
 $\text{Copernikus} \leq R < \text{Popper}$

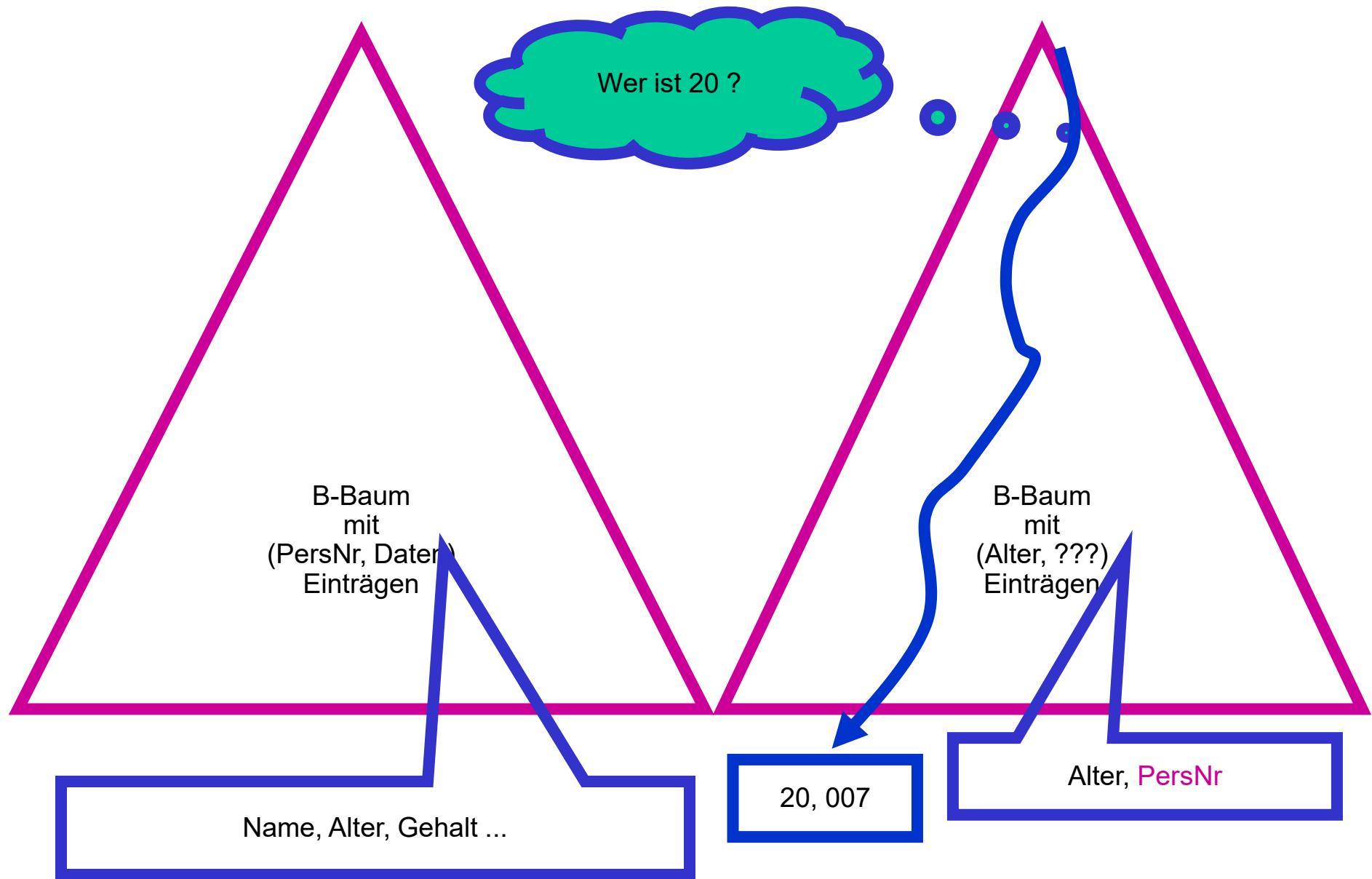
Mehrere Indizes auf denselben Objekten

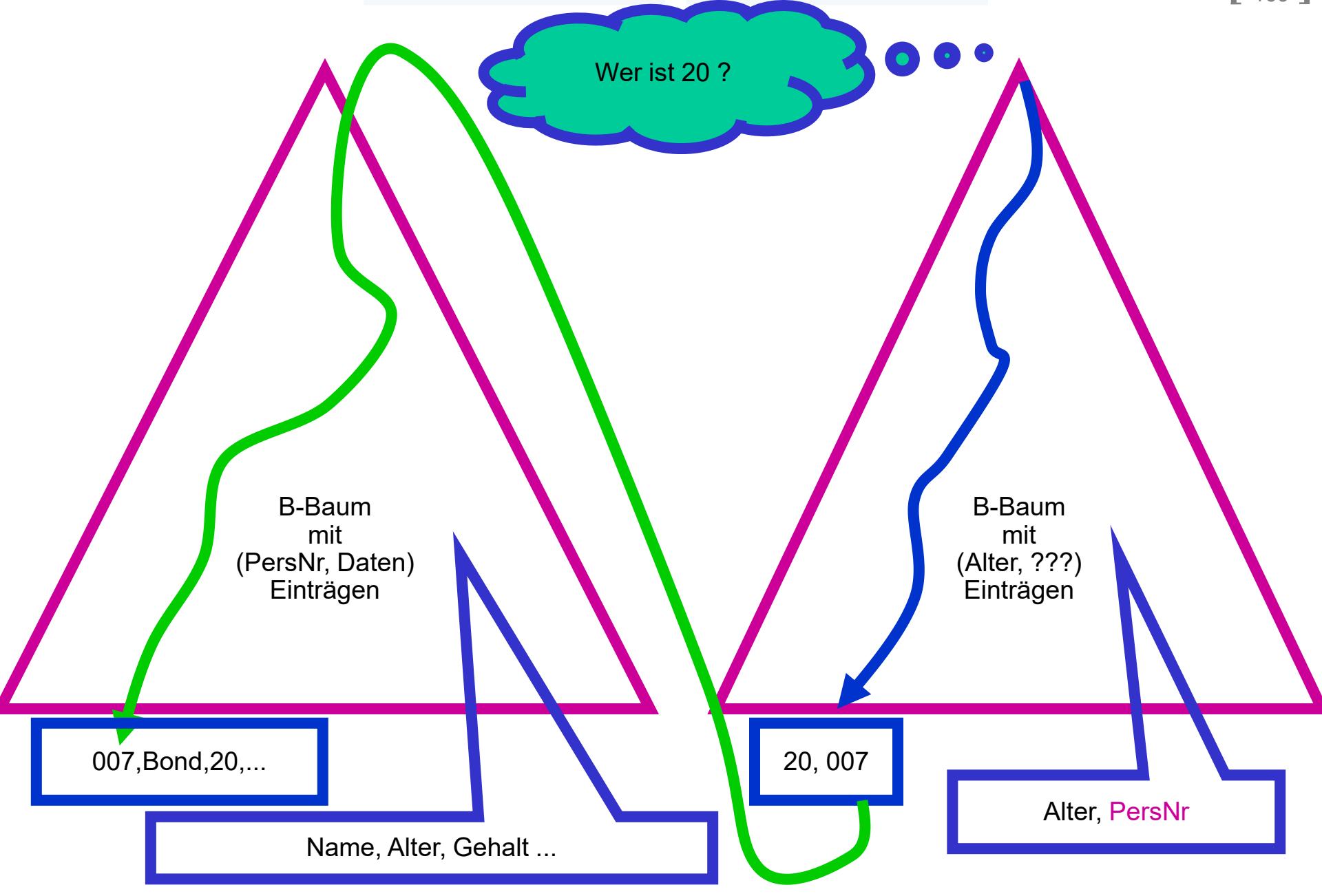
B-Baum
Mit
(PersNr, Daten)
Einträgen

Name, Alter, Gehalt ...

B-Baum
Mit
(Alter, ???)
Einträgen

Alter, PersNr





Geclusterter vs. nicht geclusterter Index

Gruppiert vs. nicht gruppierter Index

vgl. MS SQL SERVER Index Architecture (VIDEO)