

# Prednáška 7

Fungovanie DBMS

# Obsah prednášky

- Organizácia
- Ako fungujú DBMS
  - Diskovo orientovaná architektúra
  - Spracovanie dopytu
  - Indexy

# Organizácia


- Zadanie 4
  - Na konci prednášky

# Čo ďalej?

- Diskovo orientovaná architektúra DBMS
  - Spracovanie dopytu (query)
  - Indexy
- 
- Súbežné spracovanie a obnova dát



Dnešná agenda



Transakcie – ACID  
vlastnosti

# Databázy

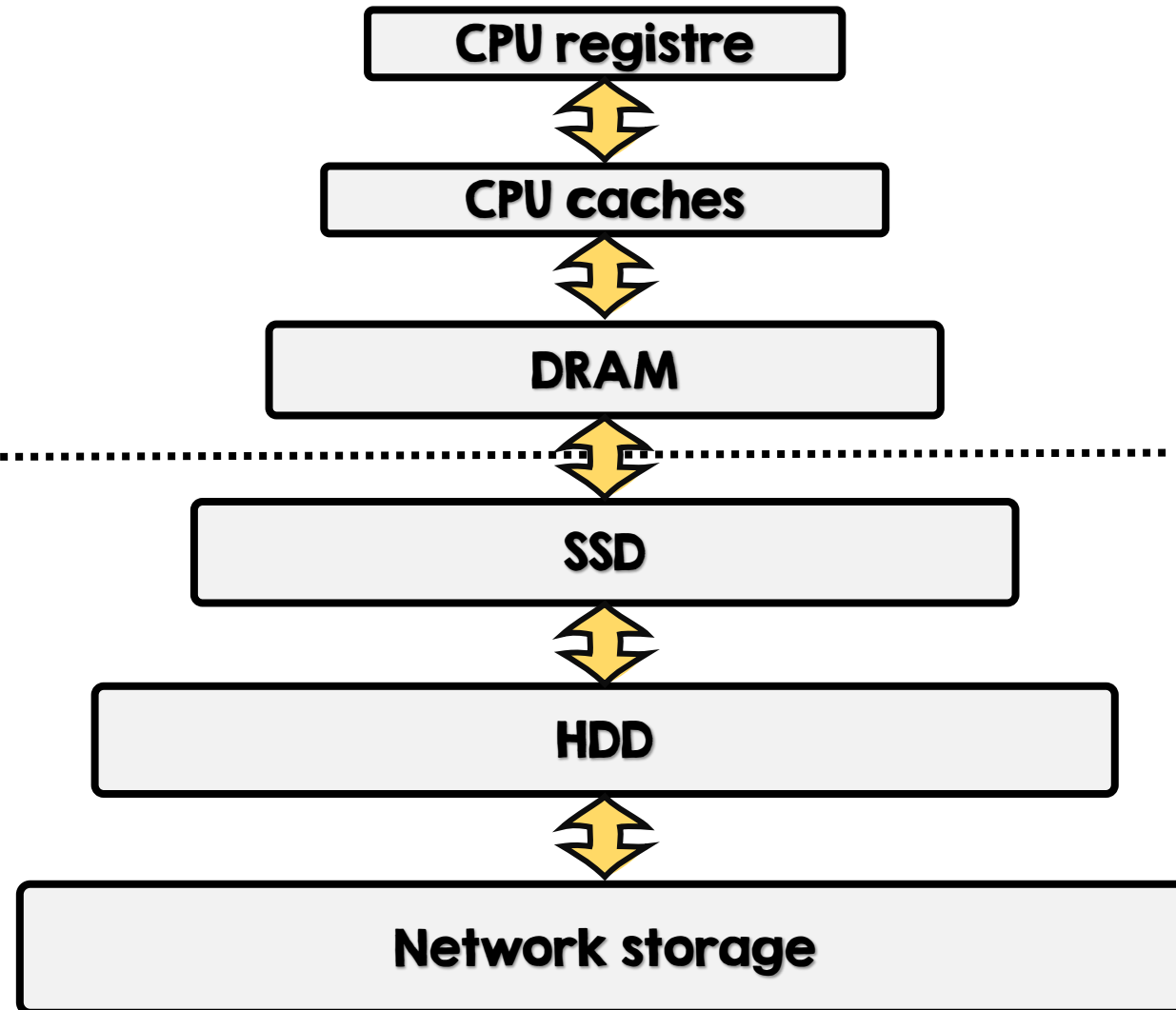
- Diskovo orientovaná architektúra (Disk-oriented architecture)
  - Primárne úložisko dát je disk – nevolatívna pamäť
- In-memory databázy
  - Primárne úložisko je volatívna pamäť
    - Obsahuje mechanizmy ako zachovať dáta aj v prípade výpadku

Zameriame sa na  
tento typ DBMS

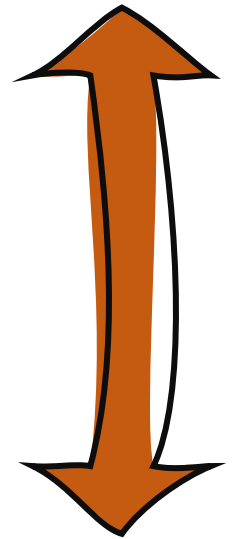
# Hierarchia pamäti

Volatívna pamäť

Nevolatívna pamäť



Rýchlejšia  
Menšia kapacita  
Drahšia



Pomalšia  
Väčšia kapacita  
Lacnejšia

# Oneskorenie

Pre prenesenie do  
pohľadu človeka

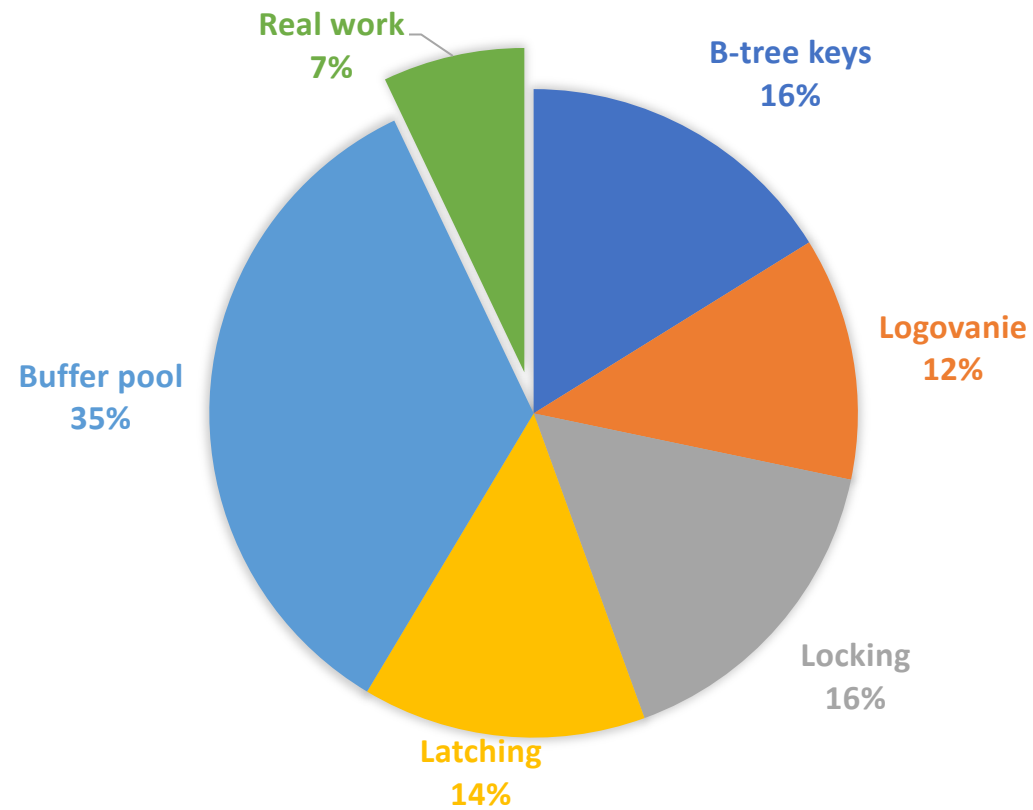
	čas	Vynasobené 1 miliardov
L1 cache	<b>0,5 ns</b>	0,5s
L2 cache	<b>7 ns</b>	7s
DRAM	<b>100 ns</b>	100 s
SSD	<b>150 000 ns</b>	1,7 dňa
HDD	<b>10 000 000 ns</b>	16,5 týždňov
Network storage	<b>30 000 000 ns</b>	11,4 miesacov
Magnetické pásky	<b>1 000 000 000 ns</b>	31,7 rokov

# Sekvenčný vs náhodný přístup

- Perzistentné úložisko (Nevolativná pamäť)
  - Sekvenčný prístup (Sequential access)
  - Dáta uložené v blokoch a blokovo adresovateľné
- Volatilná pamäť
  - Náhodný prístup (Random access)
  - Bytovo adresovateľné dáta



## MERANIE CPU INŠTRUKCIÍ



# Organizácia dát (II)

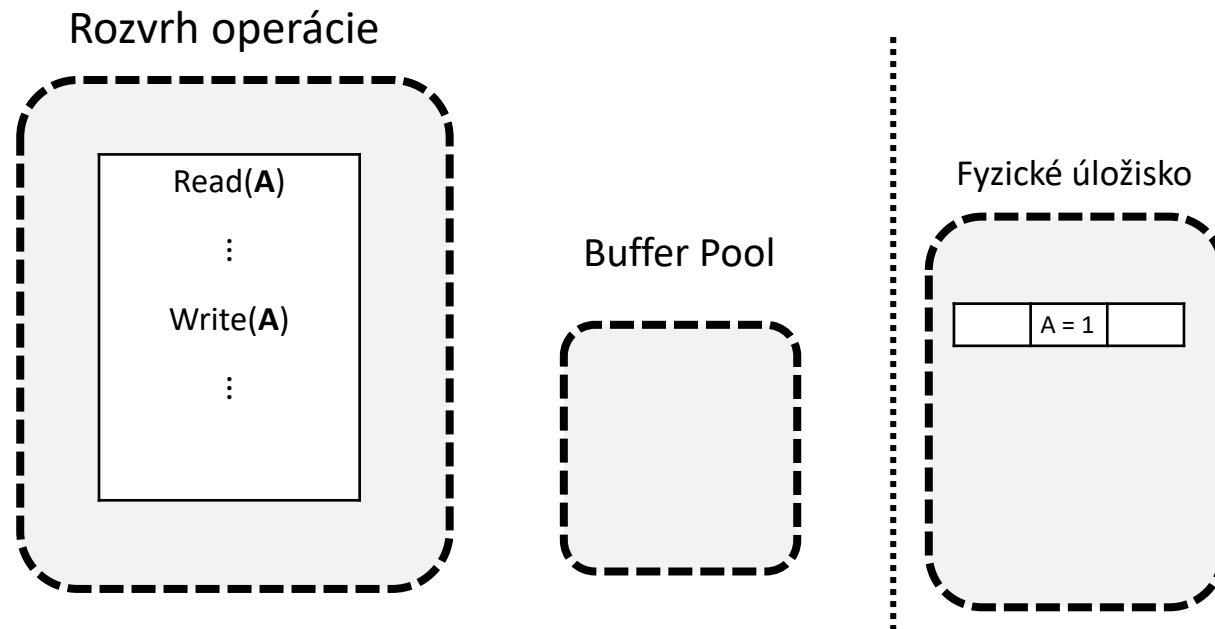
- Heap file
  - Neusporiadaná množina stránok, kde záznamy sú ukladané náhodne
    - Zvyčajne na koniec
- Stránka (page) – blok dát v rámci perzistenej pamäti, ktoré obsahujú uložené jednotlivé záznamy tabuliek
  - Tri koncepty v rámci DBMS
    - Hardware page – zvyčajne 4KB
    - OS page – zvyčajne 4KB
    - DBMS page - 512B - 16KB
- DBMS page (512B-16KB)
  - SQLite – 4KB
  - PostgreSQL 8KB
  - MySQL – 16KB

# Organizácia dát (2)

- Tuple = row = record = záznam tabuľky
  - Má svoju štruktúru v závislosti DBMS
- PostgreSQL špecifiká
  - Identifikátor záznamu – ctid (page, offset )

# Načítanie a ukladanie dát

- je dôležité si uvedomiť, kde sú uložené dáta z databázy a kedy je možné tieto dáta spracovávať (robiť nad nimi operácie)



# Ako prebieha načítanie a ukladanie informácií

## Rozvrh operácií

Je potrebné načítať  
hodnotu **A** v rámci  
dopytu

Read(**A**)

⋮

Write(**A**)

⋮

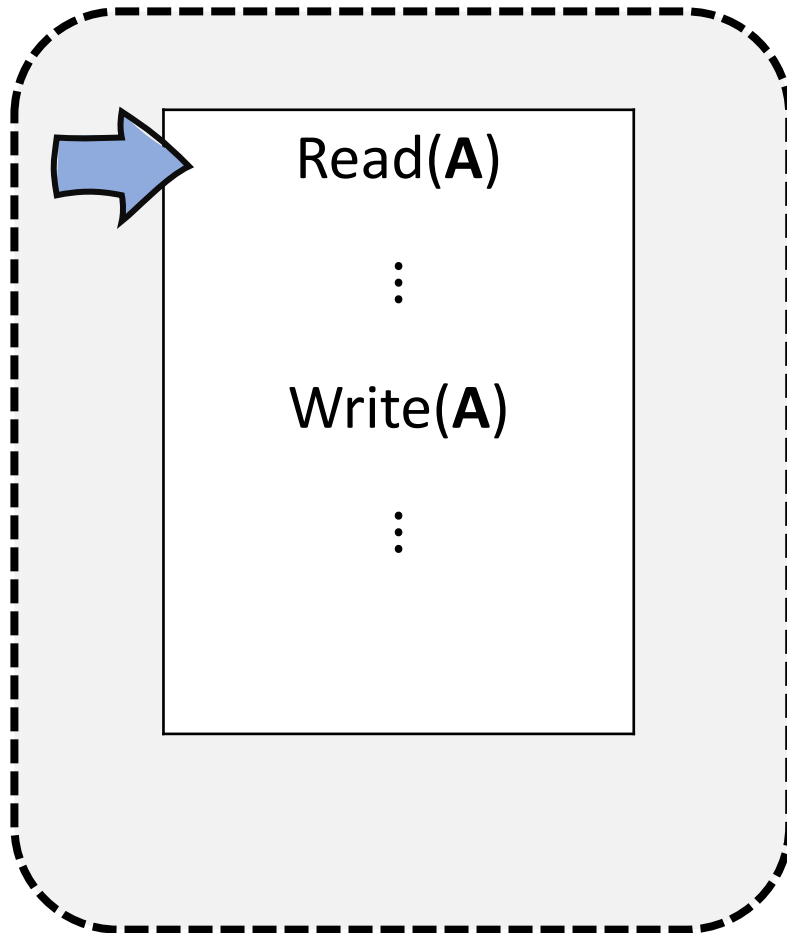
Buffer Pool

## Fyzické úložisko

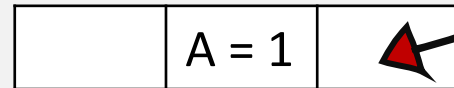
	A = 1	
--	-------	--

# Ako prebieha načítanie a ukladanie informácií

## Rozvrh operácií

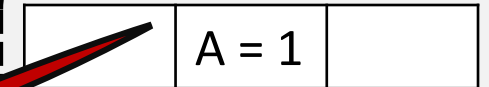


## Buffer Pool



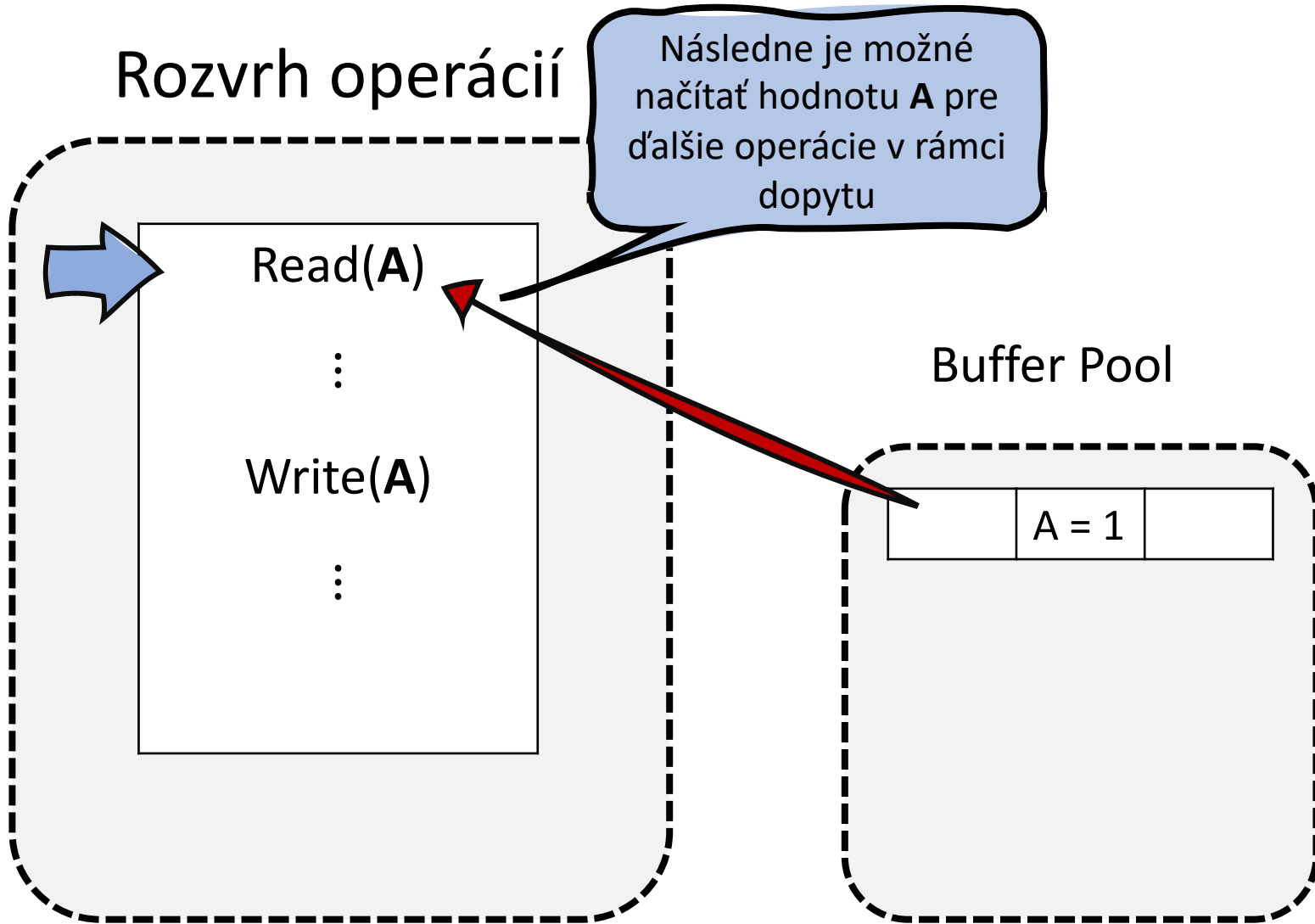
Stránka (Page) je  
skopírovaná z disku  
do hlavnej pamäte

## Fyzické úložisko

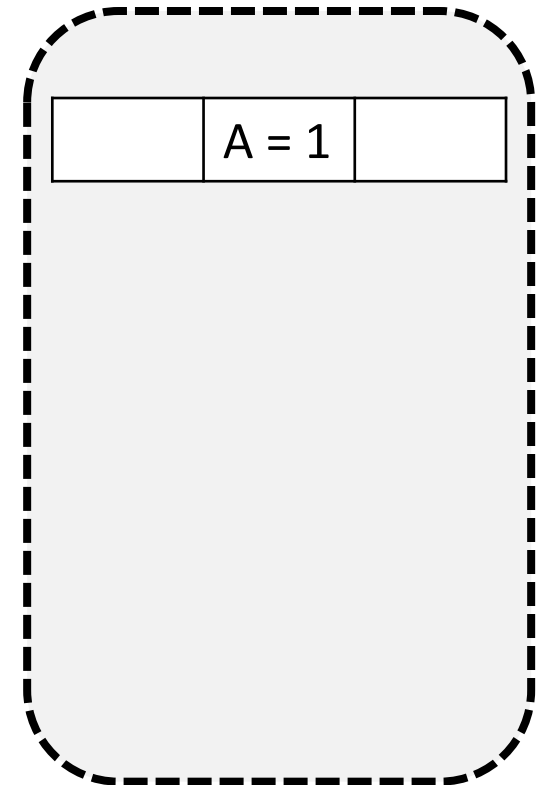


# Ako prebieha načítanie a ukladanie informácií

## Rozvrh operácií

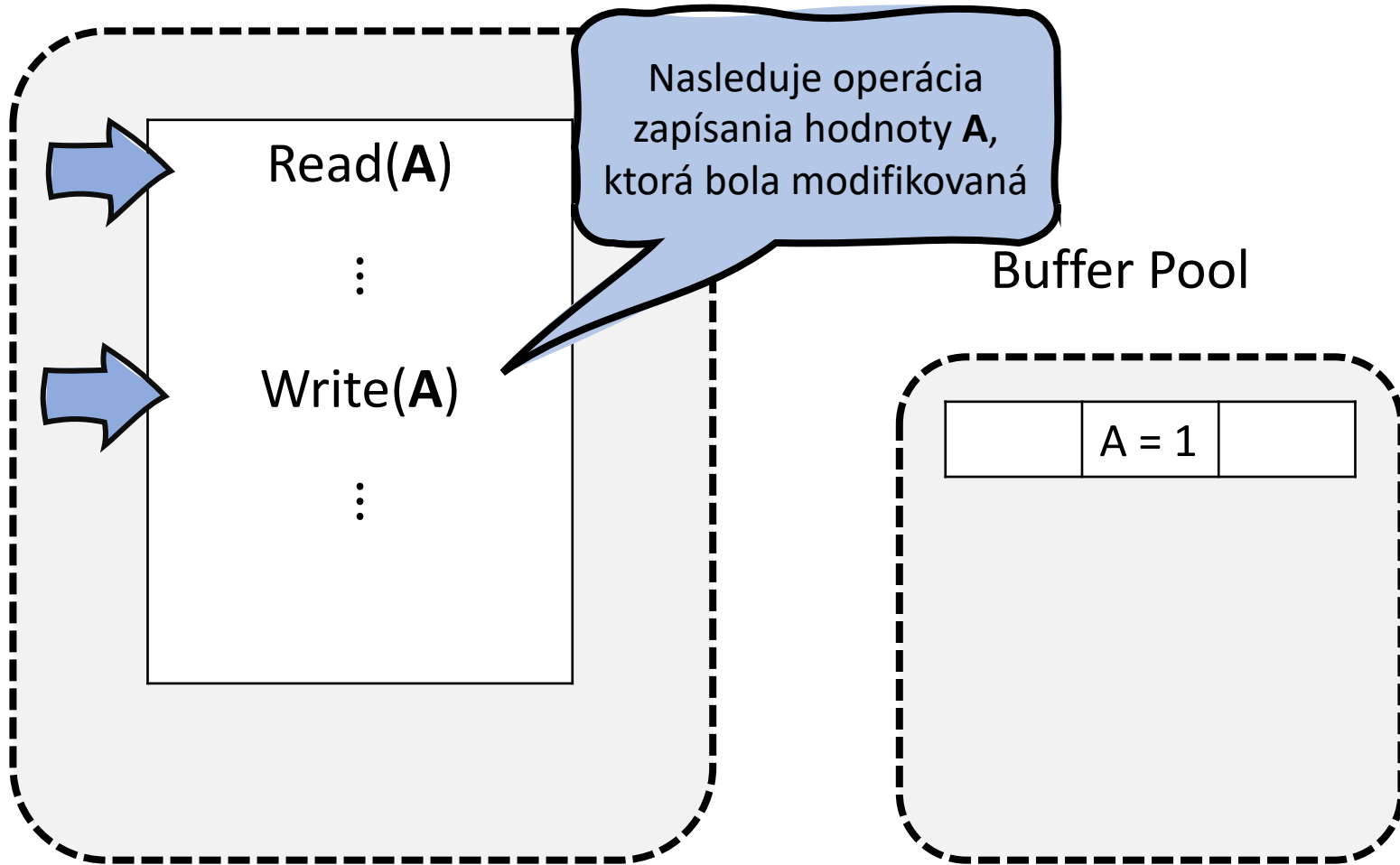


## Fyzické úložisko

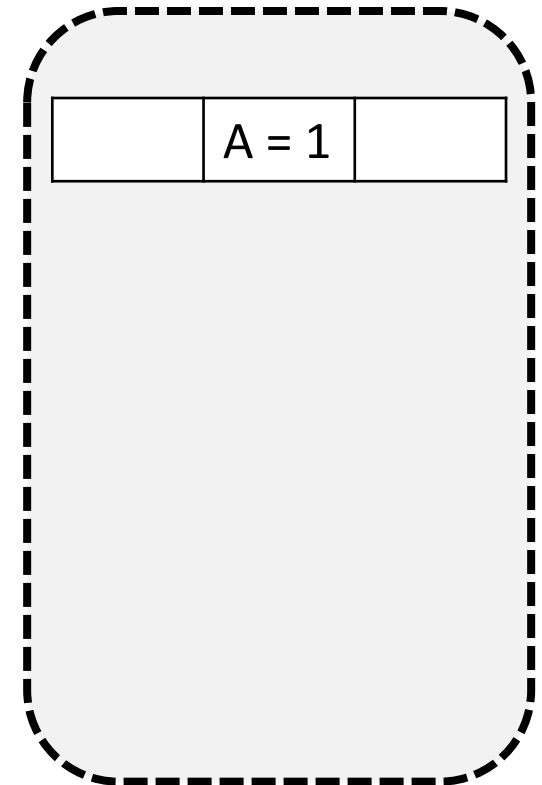


# Ako prebieha načítanie a ukladanie informácií

## Rozvrh operácií



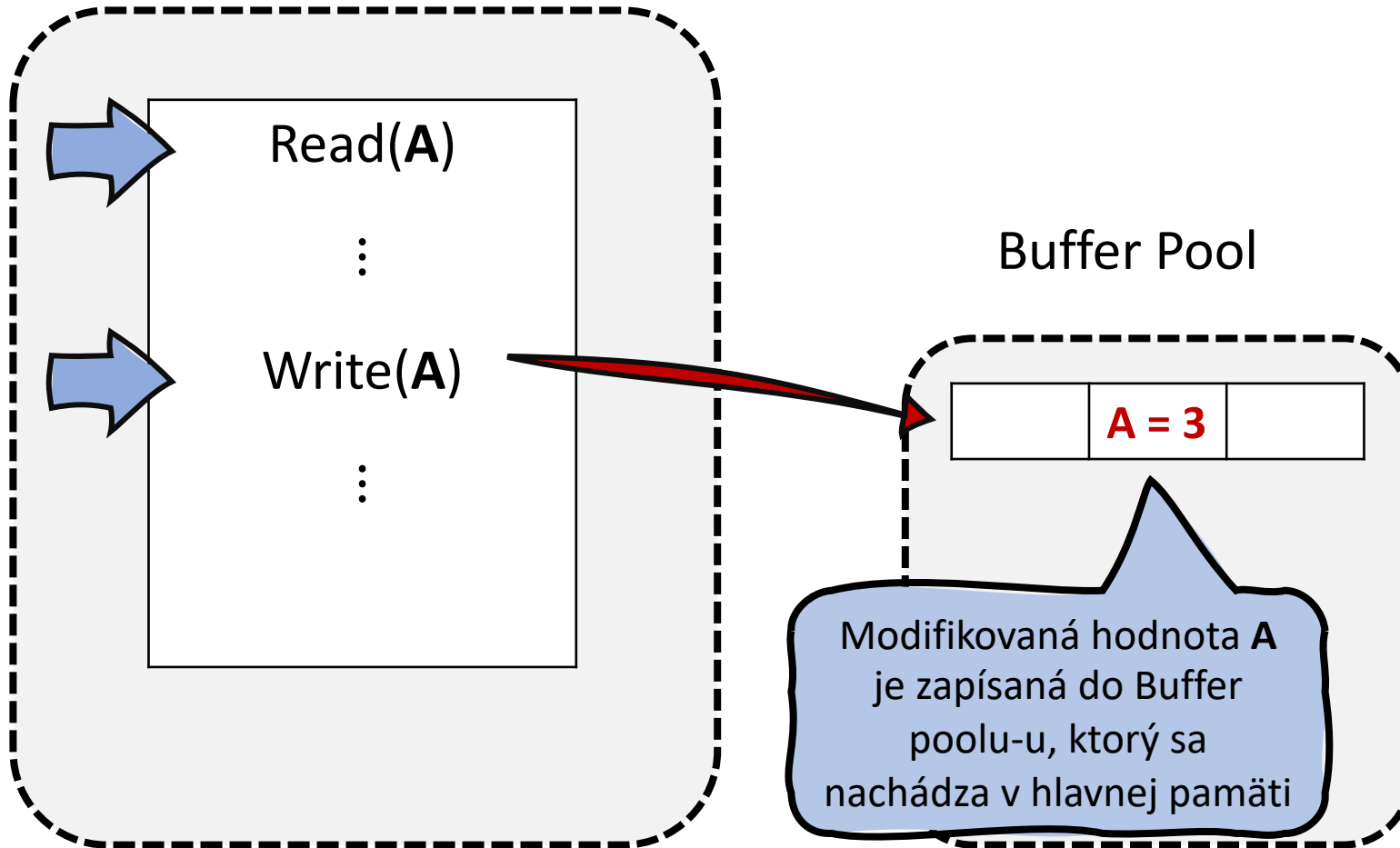
## Fyzické úložisko



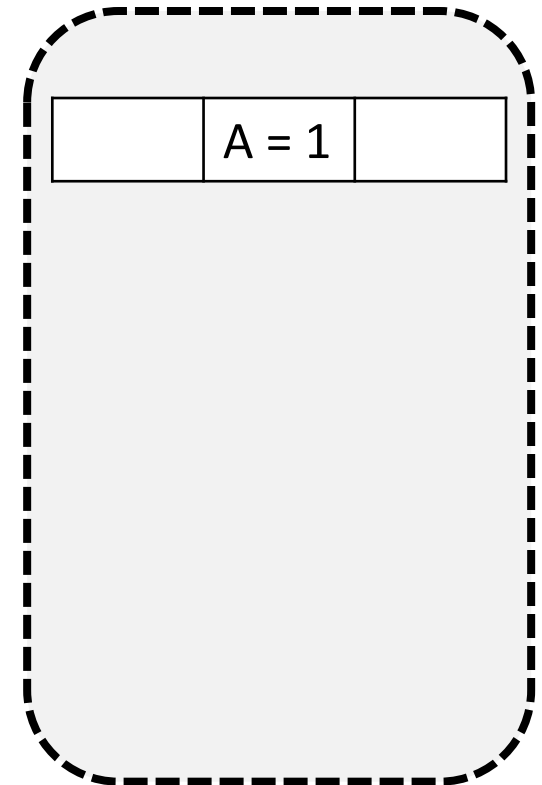


# Ako prebieha načítanie a ukladanie informácií

## Rozvrh operácií

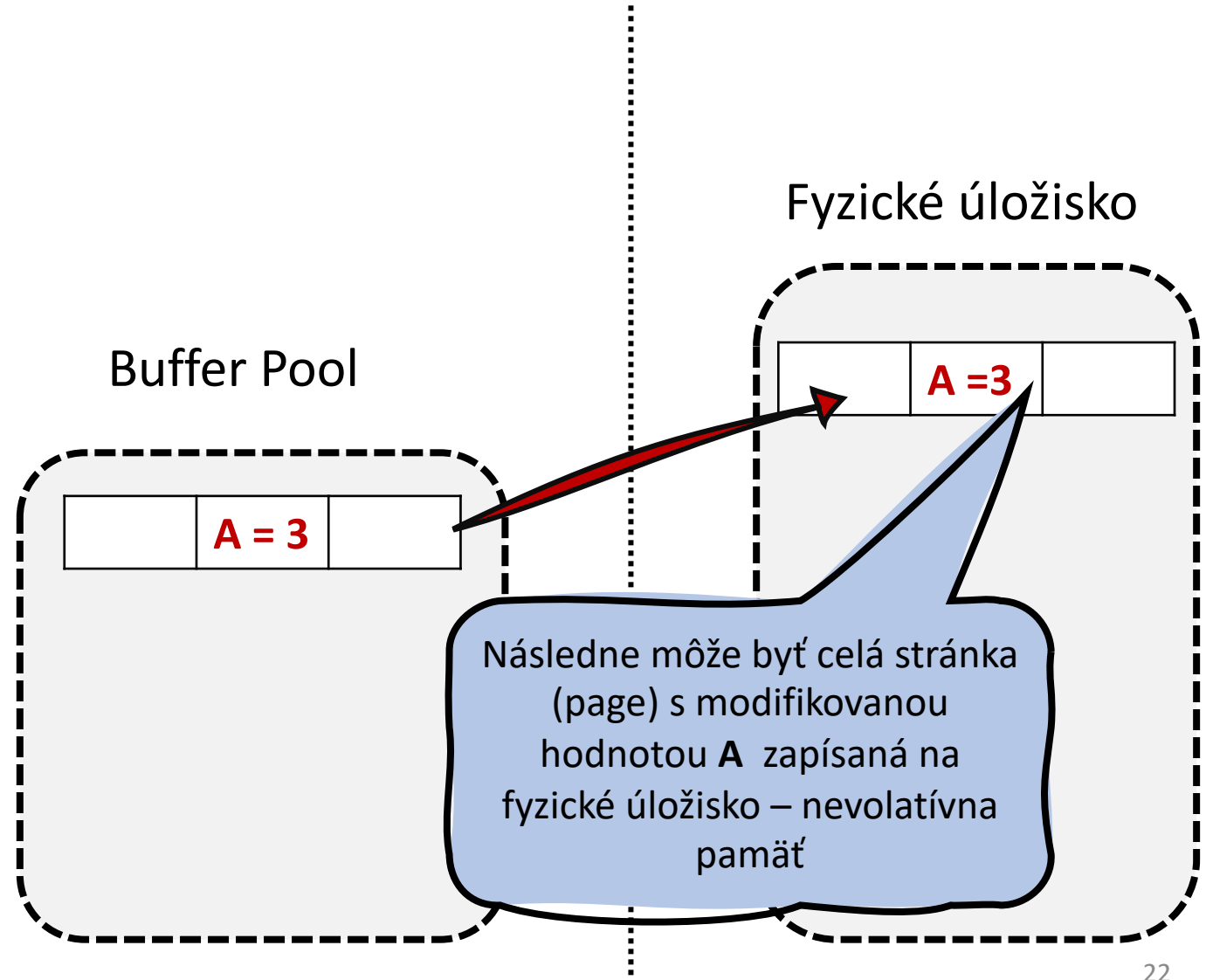
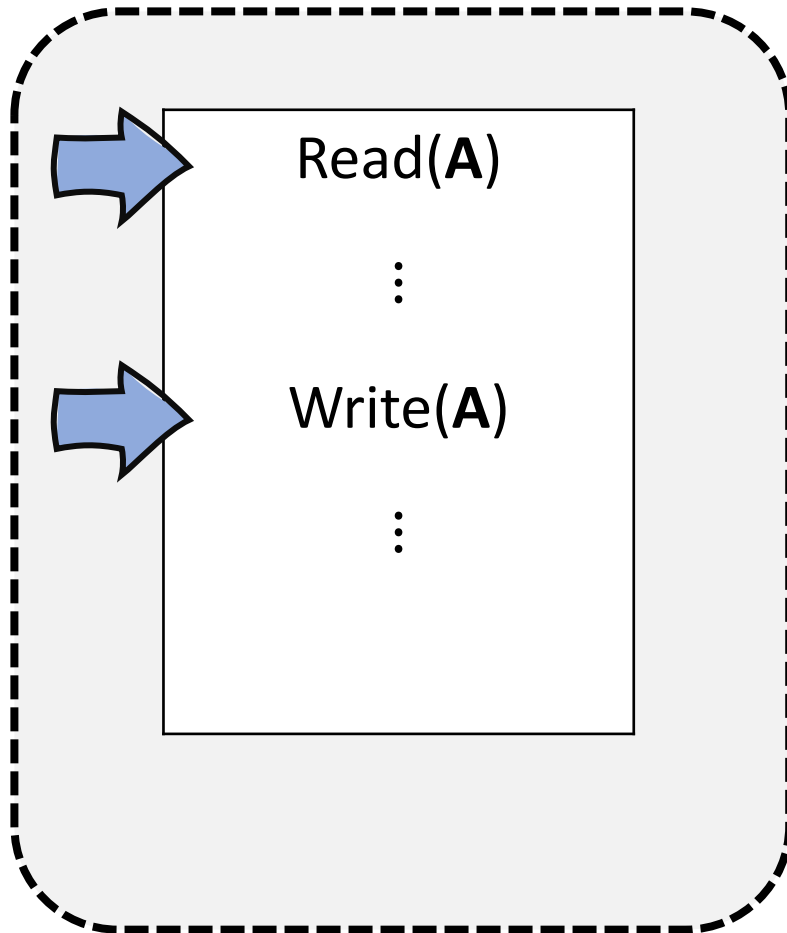


## Fyzické úložisko



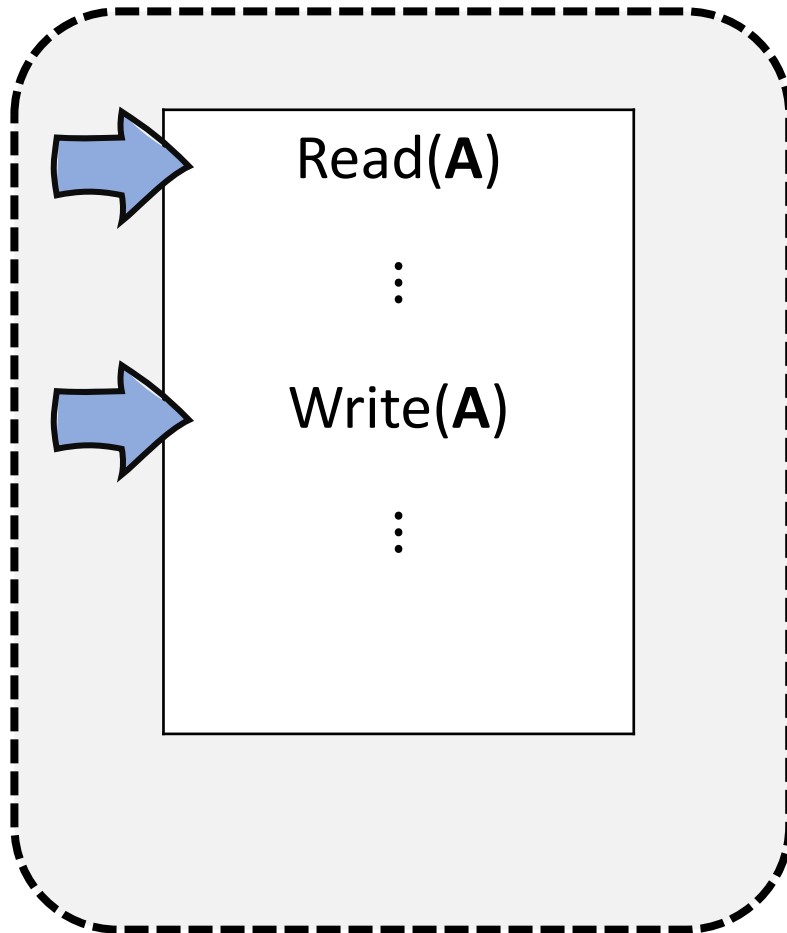
# Ako prebieha načítanie a ukladanie informácií

## Rozvrh operácií



# Ako prebieha načítanie a ukladanie informácií

## Rozvrh operácií

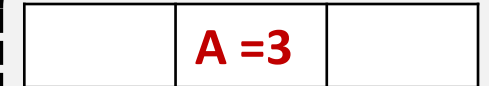


## Buffer Pool



Stránka môže zostať v buffer pool-e, alebo môže byť odstránená v závislosti od spravovania Buffer Pool-u

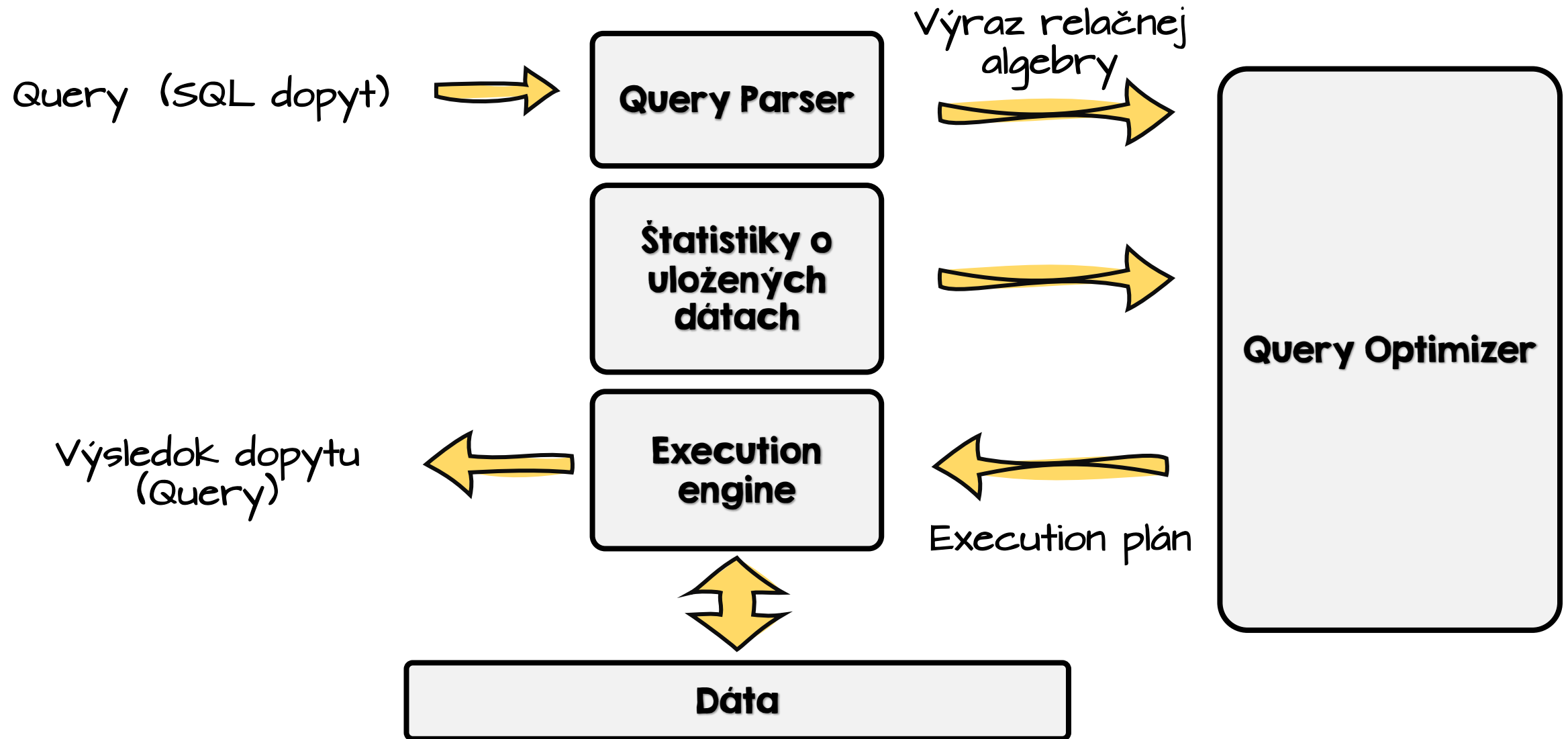
## Fyzické úložisko



Hodnota je uložená a v prípade výpadku je modifikovaná hodnota **A** zachovaná

Spracovanie dopytu

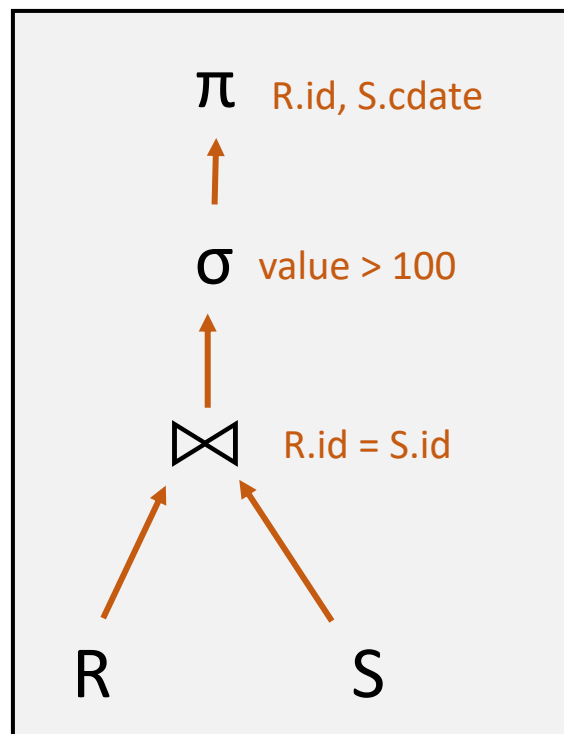
# Spracovanie dopytu (Query)



# Query optimizer

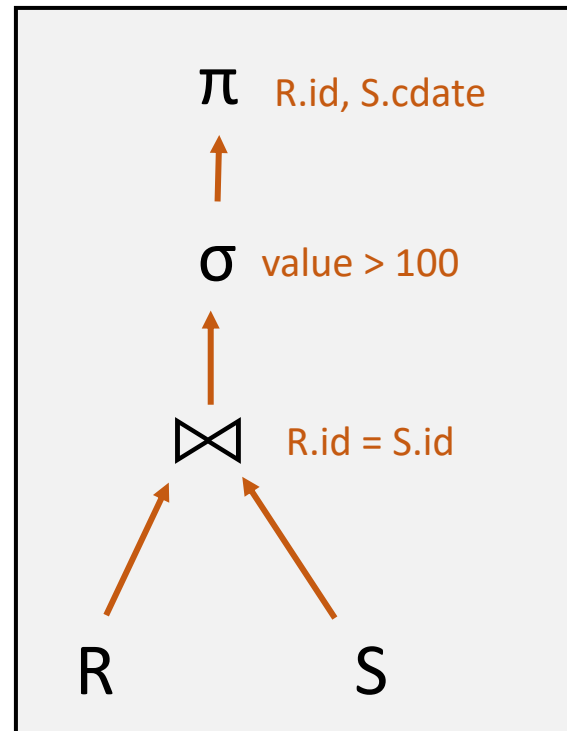
- Nazývaný tiež aj **query planner**
- Komponent databáz pre transformáciu SQL výrazu na **execution plan**
  - Proces nazývaný aj **compiling** alebo **parsing**
- Dva typy optimalizátorov
  - **Cost-Based Optimizer (CBO)**
  - **Rule-Based Optimizer (RBO)**

```
SELECT R.id, S.cdate  
FROM R JOIN S  
      ON R.id = S.id  
WHERE S.value > 100
```

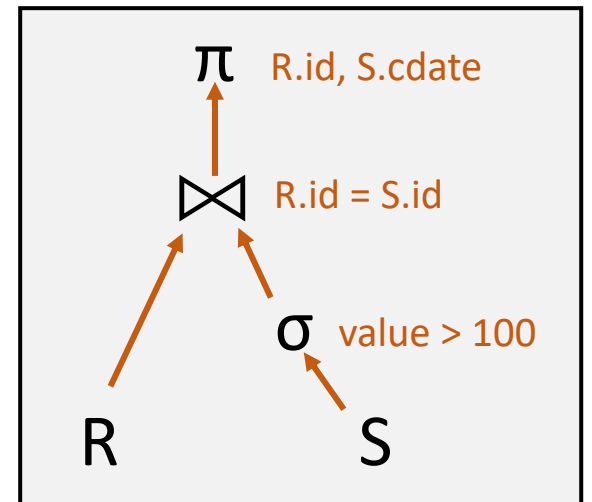


Išlo by to spraviť inak ?

```
SELECT R.id, S.cdate
FROM R JOIN S
      ON R.id = S.id
WHERE S.value > 100
```




Išlo by to spraviť inak ?





# Cost-Based Optimizer

- Generovanie execution plánu na základe vypočítanej ceny pre každý plán
- Na výpočet ceny má vplyv
  - Prístup na disk (IO operácie)
  - CPU time pre vykonanie jednotlivých operácií
  - Možnosť paralelizácie jednotlivých operácií
  - Rýchlosť komunikácie
  - Veľkosť dát, nad ktorými je potrebné uskutočňovať jednotlivé operácie
- Veľké DB systémy najviac “trpia” nutnosťou pristupovať k disku pre načítanie dát



AZURE s obmedzením  
na IO operácie

# Cost-Based Optimizer - PostgreSQL

- Cost v rámci PostgreSQL
  - Prístup k stránke sekvenčne - seq\_page\_cost
  - Náhodný prístup ku stránke - random\_page\_cost
    - zníženie hodnoty ma za následok menej používania indexu v prípade index scanu
  - Spracovanie záznamu indexu - cpu\_index\_tuple\_cost
  - Spracovanie samotného záznamu - cpu\_tuple\_cost
  - a ďalšie
- Viac info na: <https://www.postgresql.org/docs/13/runtime-config-query.html>

# Rule-Based Optimizer

- Pevne stanovené pravidla na základe, ktorých dochádza k rozhodnutiu o vytvorení execution plánu
  - Napr. v prípade, že existuje index pre atribút X, tak ho použi
- V súčasnosti veľmi zriedkavo používané

# Query optimizer - query plan

- Ako zobrazit' **query plan** ?
  - V závislosti od DBMS
    - Oracle - **EXPLAIN PLAN FOR**
    - PostgreSQL - **EXPLAIN**
    - MySQL - **EXPLAIN**
    - SQL SERVER - **rôzne spôsoby** (set statistics profile on, grafická reprezentácia)
- PostgreSQL
  - **EXPLAIN** vs **EXPLAIN ANALYZE**

PostgreSQL príklad

**EXPLAIN** SELECT \* FROM students

**EXPLAIN ANALYZE** SELECT \* FROM students

# Ukážka - explain

- PostgreSQL <https://www.postgresql.org/docs/13/runtime-config-query.html>

Indexy

# Indexy

- je perzistentný databázový objekt na jednoduchšie nájdenie informácií
  - cieľ – zvýšenie výkonu pri získavaní údajov
  - rozdiel medzi prehľadávaním celej tabuľky a takmer okamžitým nájdením záznamov
  - je vytvorený nad stĺpcom alebo stĺpcami tabuľky
- index sa nevyberá pri písaní dopytu ručne, ale je to automatický proces
  - optimizer vyberie najlepší spôsob pre vykonanie daného dopytu
- zaberá dodatočné miesto
  - niekedy veľa miesta
- môžeme prirovnať ku indexu v knihe, tiež zaberajú dodatočné miesto
  - v rámci knihy však netreba tento index spravovať – kniha sa vytlačí a už sa nedá meniť obsah. V prípade DB sa však údaje neustále menia, aktualizujú ....

# Indexy - typy

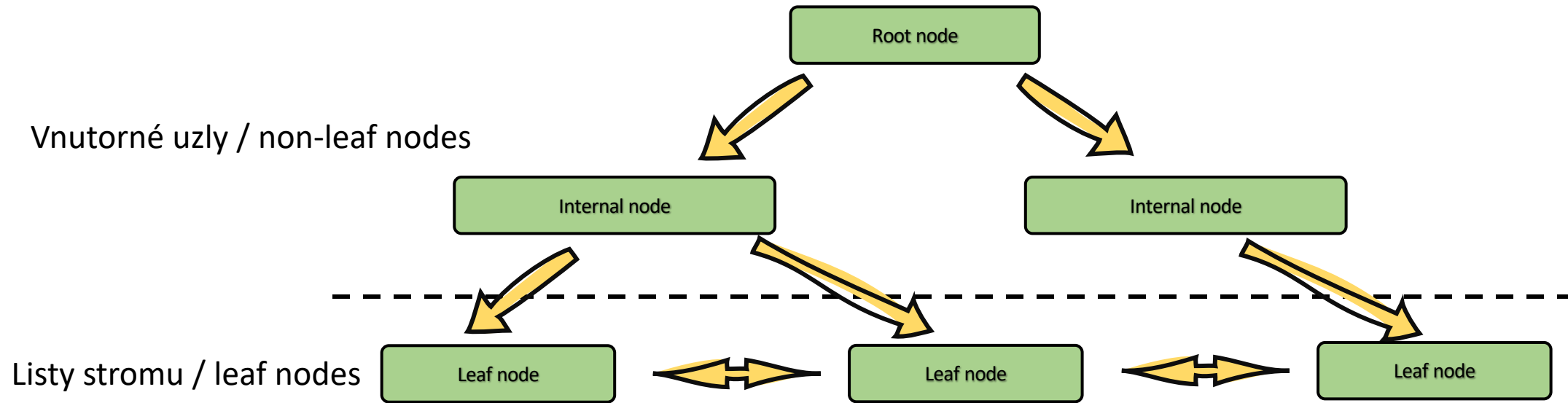
- Hash index
- B+tree
- Bitmap index
- Ostané typy indexov – **inverted, trie, radix ...**
- PostgreSQL
  - **B-tree, hash, GiST, SP-GiST, GIN a BRIN** (dokumentácia - verzia 13 - release date 24 September 2020 )



# B+tree

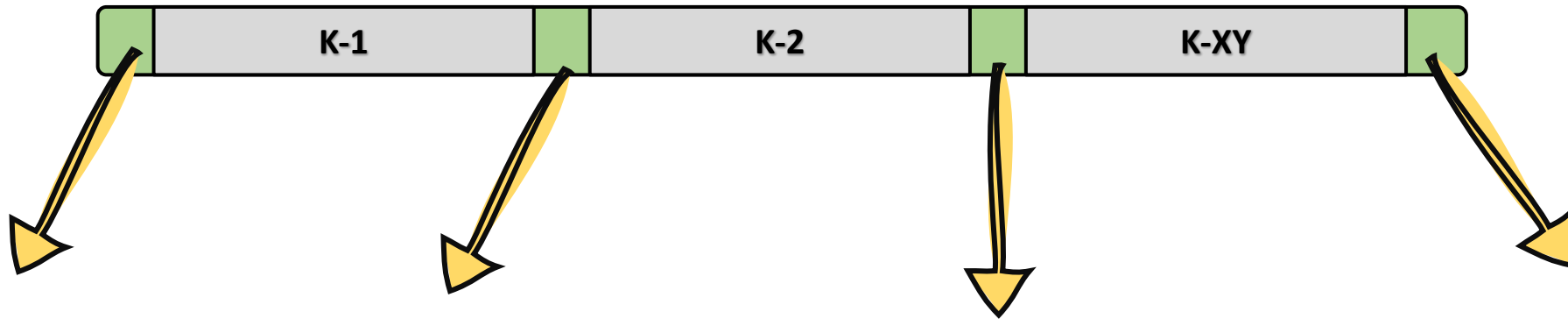
- Predstavujú rozšírenie stromov B-tree
  - B-tree zamieňaný s binárnymi stromami
- Vyvážený strom, samotné dáta sú obsiahnuté iba v listoch (leaf nodes)
- Definícia b-tree
  - Každý uzol má najviac  $m$  potomkov
  - Každý uzol, ktorý nie je list má najmenej  $m/2$  potomkov
  - Root uzol má najmenej 2 potomkov
  - Uzol, ktorý nie je list má  $k$  potomkov a  $k-1$  kľúčov
  - Všetky listy sa nachádzajú na rovnakej úrovni.
- Logaritmická zložitosť

# B+tree: štruktúra stromu



Len **Leaf nodes** obsahujú dáta, ktoré sú usporiadané na základe hľadaného kľúča

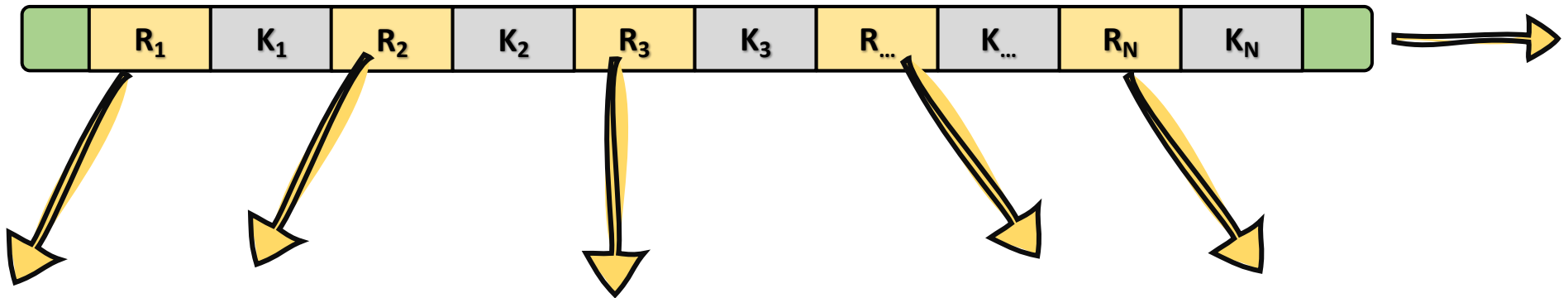
# B+tree: non-leaf uzly



**Ukazovateľe** (Pointers) na ďalšie stránky B+tree

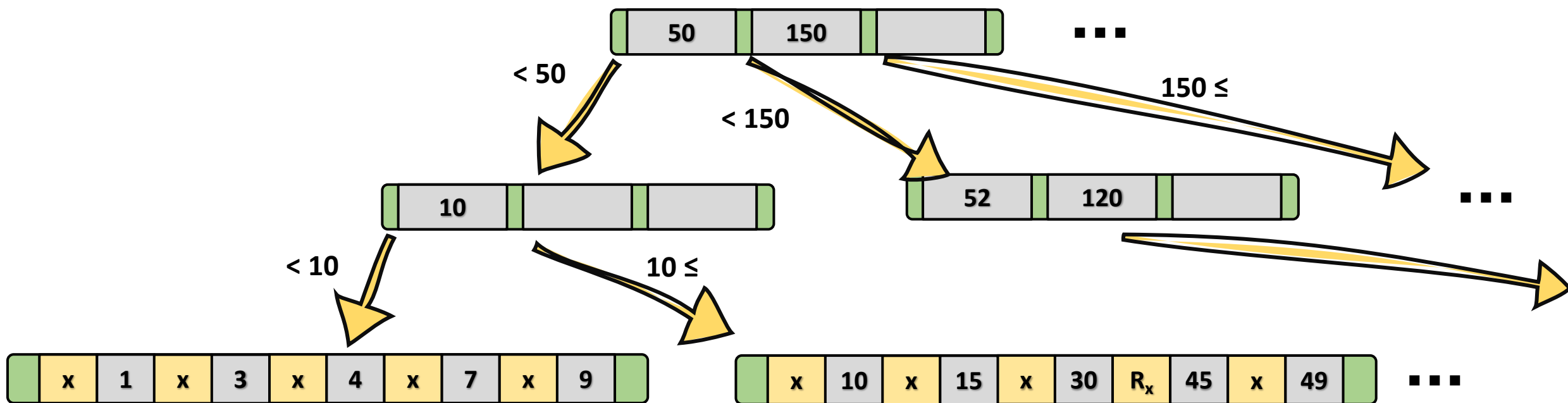
# B+tree: leaf nodes(II)

**Ukazovateľ** na  
predchádzajúci  
leaf uzol



**Ukazovateľe** (Pointers) na konkréty záznam tabuľky (record)

# B+tree



# B+tree: vizualizácia

- <https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html>

# B+tree: duplicita klúčov

- Viacero záznamov obsahuje rovnaký kľúč
- Možnosti ako riešiť
  - Záznamy s rovnakým kľúčom sú uložené v rámci jednej stránky + overflow page
  - Záznamy s rovnakým kľúčom sú uložené v rámci listov
    - Nutnosť modifikovať prehľadávanie v rámci listov

# B+tree: v praxi

- Ak **rad-d** B+tree je 100
  - Počet potomkov **m** pre daný uzol je  $d \leq m \leq 2d$
- Typický fill factor je 67%
  - Potom priemerný počet potomkov sa bude pohybovať **133**
- Kapacita B+tree
  - 3 úroveň:  $133^3 = 2\,352\,637$  záznamov
  - 4 úroveň:  $133^4 = 312\,900\,700$  záznamov
- Možnosť uloženia stránok v rámci buffer poola
  - 1 level                      1 stránka                      =                      8KB
  - 2 level                      133 stránok                      =                      1MB
  - 3 level                      17689 stránok =                      133MB



# B+tree vs B-tree

## Difference Between B-Tree And B+ Tree

B-Tree	B+ Tree
Data is stored in leaf nodes as well as internal nodes.	Data is stored only in leaf nodes.
Searching is a bit slower as data is stored in internal as well as leaf nodes.	Searching is faster as the data is stored only in the leaf nodes.
No redundant search keys are present.	Redundant search keys may be present.
Deletion operation is complex.	Deletion operation is easy as data can be directly deleted from the leaf nodes.
Leaf nodes cannot be linked together.	Leaf nodes are linked together to form a linked list.

# Hash index vs b+tree index

- Hash
  - Vhodný na hľadanie s rovnosťou (=)
  - Konštatnej časovej zložitosti pre vkladanie a hľadanie
- B+tree
  - Vhodný pre hľadanie rovnosti ale aj rozsahu

# B+tree: leaf nodes - hodnoty

- Hodnoty v rámci leaf node
- Prístup 1. – Record ID
  - Ukazovateľ na lokalitu záznamu tabuľky, pre ktorý korešponduje záznam v indexe
- Prístup 2. – Dáta záznamu
  - Obsahuje samotný záznamu tabuľky
    - InnoDB v rámci MySQL vytvára b+tree na základe primárneho kľúča
  - V prípade použitia ďalších indexov sa dané indexy odkazujú na record ID



PostgreSQL

ORACLE



ORACLE



# Bitmap index

- Špeciálny typ indexu
- Každý bitmap index je vytvorený nad jedným hľadaným kľúčom
  - Napr. žena, muž
- Vhodný pre low cardinality
- Drahé vkladanie a mazanie záznamov
  - Nutnosť úpravy bitmapy
- Využíva sa vo forme in-memory indexu, ktorý je dočasne vytvorený len pre daný dopyt (query)
  - Napr. v rámci Bitmap scan

# Bitmap index

Record number	ID	Gender	Income_level
0	11111	m	L1
1	22222	f	L2
2	12345	f	L1
3	89879	m	L4
4	12345	f	L3

Bitmap index pre  
atribút Gender

m **10010**

w **01101**

Bitmap index pre  
atribút Income\_level

L1 **10100**

L2 **01000**

L3 **00001**

L4 **00010**

L5 **00000**

# Ako vyzerá vyhľadávanie

- Index access – prechod stromom k jednotlivým listom
  - toto nie je problém, keďže strom je balanced
- Index range scan – prehľadávanie zoznamu listov (môže prejsť veľkú časť indexu)
  - ak musí prejsť veľa tak nastáva problém spojený s prístupom k tabuľkám (Table access), čo spôsobí spomalenie
- Table access – vytiahnutie dát z tabuľky
  - Nastáva problém pokiaľ je treba prejsť veľa tabuliek

# PostgreSQL - prístupové metódy

- Seq Scan
- Index scan
- Bitmap scan
- Hash scan

# Indexy

- Vytvorenie indexu neprináša iba pozitívne vlastnosti
  - Spomalenie pri manipulácii dát
  - Zvýšenie požiadaviek na úložisko
  - Údržba indexov



# Clustered index

- Tabuľka je uložená v rámci disku podľa primárneho indexu - rýchly sekvenčný sken
  - Napr. MySQL

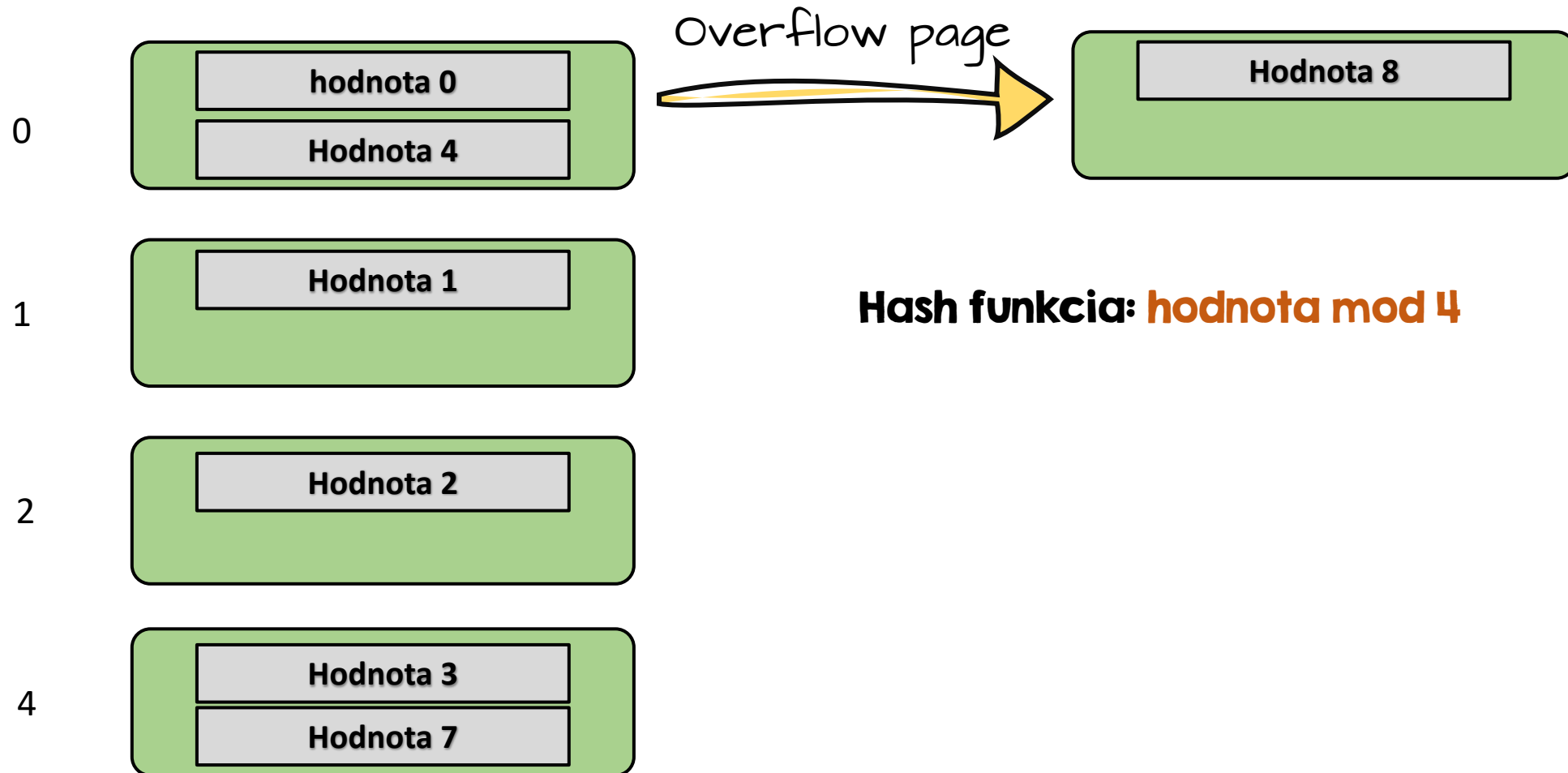
# Hash index

- Využívanie hash funkcie pre výpočet pozície, kde bude záznam uložený
  - Dôležitosť výber vhodnej funkcie, ktorá je rýchla a má primeraný počet kolízií pre dané dáta
- Efektívny pre rovnosť (equality search)
- Nemožno použiť pri range prehľadávaní (range search) alebo partial scan
- Časová zložitosť  $O(1)$  + prehľadávanie prípadných overflow pages
- Typy Hash indexov
  - Statické
  - Dynamické (extendible, dynamic)

# Static Hash Index

- Je kolekcia bucket
  - Bucket – primárna stránka + overflow stránka
  - Každý bucket – obsahuje viacero záznamov
- $h(k) \bmod N$ 
  - $N$  - počet bucketov
- Záznamy s rôznymi kľúčmi môžu byť umiestnené v rámci to istého bucketu
  - V prípade zaplnenia primárnej stránky + potreba overflow page

# Hash index - statické



# Užitočné linky

- PostgreSQL – vnútorne fungovanie
  - <https://www.interdb.jp/pg/index.html>

# Zadanie 4