Database Management Systems II
Exercise Session 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Robert Gottstein

gottstein@dvs.tu-darmstadt.de

Exercise 1.4
Cost/Performance differences between RAID Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

b) We will now evaluate the cost/performance differences between RAID levels 1, 4 and 5. We will be comparing configurations with the same usable disk capacity, namely:

- RAID level 4 and 5 arrays with ***D data disks*** - labeled disks 1 through *D*, ***and 1 check (parity) disk*** - labeled disk *D+1*.
- RAID level 1 arrays with ***D data disks and D additional mirror disks*** .

For the sake of simplicity, we will assume that the average positioning time for parallel accesses of D blocks is not substantially different from the average positioning time for a single block access.

Exercise 1.4
Cost/Performance differences between RAID Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

Use the following performance metrics for comparison:

$$speedup = \frac{array\ throughput}{single\ disk\ throughput}$$

$$efficiency = \frac{speedup}{number\ of\ disks\ used}$$

Compare the speedups and efficiencies of RAID levels 1, 4 and 5 when processing requests of the following types:

- small read (r) and small write (w) operations (1 striping unit)
- large read (R) and large write (W) operations (accessing all data disks - full stripe)
- small read-modify-write (m) operations

Exercise 1.4
Cost/Performance differences between RAID Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

Note 1:
To determine **speedup**, consider the **maximum number of disks that can be utilized** at a point in time for processing operations of the respective type. *Ignore disks executing parity operations.*

Note 2:
The "number of disks used" in the **efficiency** equation *includes* ***all additional check (parity) disks*** used for storing redundant information.

Exercise 1.4
Cost/Performance differences between RAID l

RAID 1:

Note: ”IP” xecuted in parallel.

every write must be executed on two disks (data disk and mirror disk) ➔

$$speedup = \frac{D}{1}$$

Op.		Max S	Max E
r	**read(data d.. read(mirror disk *i*)**	*2D*	$\frac{2D}{2D}$
R	**read(data disks 1..D) OR read(mirror disks 1..D)**	*2D*	$\frac{2D}{2D}$
w	**write(data disk *i*) IP write(mirror disk *i*)**	*D*	$\frac{D}{2D}$
W	**write(data disks 1..D) IP write(mirror disks 1..D)**	*D*	$\frac{D}{2D}$
m	**1. read (data disk *i*) OR read (mirror disk *i*) 2. write (data disk *i*) IP write (mirror disk *i*)**	*D*	$\frac{D}{2D}$

Exercise 1.4
Cost/Performance differences between RAID I

RAID 4:

Note: ”IP… executed in parallel.

First: Read data
Second: Write data
(*w can not be executed parallel!)*
In the same time, a single disk could write two *w*!

Op.		Max S	Max E
r	**read(data disk *i*)**	*D*	$\frac{D}{D+1}$
R	**read(data disks 1..D)**	*D*	$\frac{D}{D+1}$
w	**1. read(disk *i*) IP read(disk D+1)** **2. write(disk *i*) IP write(disk D+1)**	*1/2*	$\frac{1 \cdot 0.5}{D+1}$
W	**write(disks 1..D+1)**	*D*	$\frac{D}{D+1}$
m	**1. read(disk *i*) IP read(disk D+1)** **2. write(disk *i*) IP write(disk D+1)**	*1/2*	$\frac{1 \cdot 0.5}{D+1}$

Exercise 1.4
Cost/Performance differences between RAID I

RAID 5:

Note: "IP[illegible] executed in parallel.

First step: Read data (data and parity)
Second step: Write data and parity
In the same time, a single disk could execute two w operations!

Op.		Max S	Max E
r	**read(data disk i)**	*D+1*	$\frac{D+1}{D+1}$
R	**read(data disks 1..D)**	*D+1*	$\frac{D+1}{D+1}$
w	**1. read(disk i) IP read(disk D+1)** **2. write(disk i) IP write(disk D+1)**	$\frac{D+1}{2} \cdot \frac{1}{2}$	$\frac{\frac{D+1}{2} \cdot \frac{1}{2}}{D+1}$
W	**write(disks 1..D+1)**	*D*	$\frac{D}{D+1}$
m	**1. read(disk i) IP read(disk D+1)** **2. write(disk i) IP write(disk D+1)**	$\frac{D+1}{2} \cdot \frac{1}{2}$	$\frac{\frac{D+1}{2} \cdot \frac{1}{2}}{D+1}$

Exercise 1.4
Cost/Performance differences between RAID Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

Access Pattern	RAID 4	RAID 5	RAID 1
Small Read (r)	$\frac{D}{D+1}$	1	1
Large Read (R)	$\frac{D}{D+1}$	1	1
Small Write (w)	$\frac{1}{D+1} \cdot \frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$
Large Write (W)	$\frac{D}{D+1}$	$\frac{D}{D+1}$	$\frac{1}{2}$
Small Read-Modify-Write (m)	$\frac{1}{D+1}$	$\frac{1}{2}$	$\frac{1}{2}$

Exercise 1.4
Summarize the pros and cons of RAID levels

TECHNISCHE UNIVERSITÄT DARMSTADT

Summarize the pros and cons of RAID levels 1 and 5.

RAID 1	RAID 5
(PROS) • **better S for reads** • **better S and E for 'w'** • better S for 'm' • easier recovery after failure • **allows multiple disk failures** *(CONS)* • **bad storage efficiency (more disks needed / higher costs)** (usable capacity: 50%)	*(PROS)* • **better E for large writes** • **better storage efficiency** (usable capacity = 90% for parity group size of 10) *(CONS)* • main problem: **inefficient small writes**, due to the extra I/O to read old data and parity • more seek times per request

Exercise 1.4
Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

Conclusion:
Disk space is nowdays plentiful
➔ *Storage efficiency is not a major problem*
(with exception for largest corporations that need to store extremely large volumes of data).

Throughput is much more important than space efficiency.

Therefore:
RAID 1 tends to be preferred over RAID 5, because it offers double the read bandwidth to each data item at the cost of 1 extra I/O per write. (Jim Gray et al.)

Exercise 2.1

TECHNISCHE UNIVERSITÄT DARMSTADT

Segments and Pages, Addressing and Update Strategies

Exercise 2.1
Segments and Pages, Addressing and Update Strategies

TECHNISCHE UNIVERSITÄT DARMSTADT

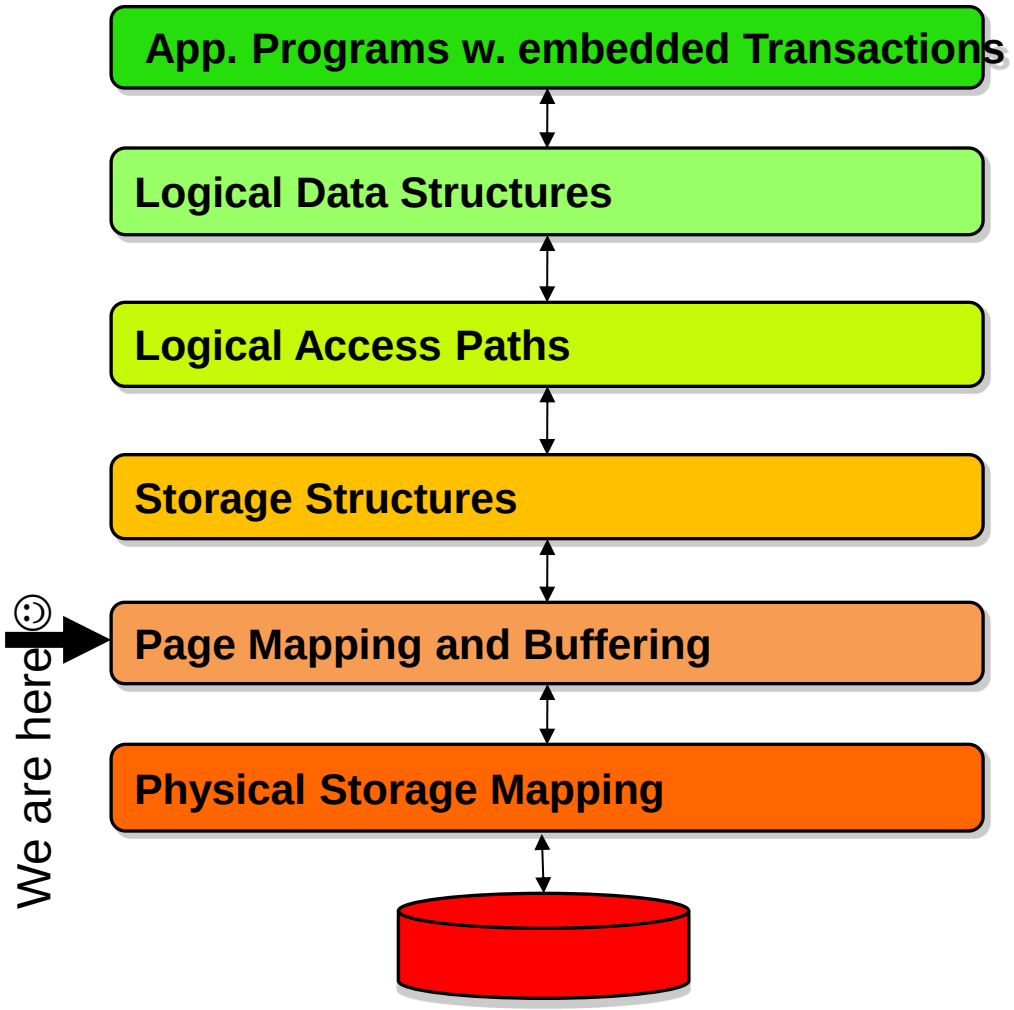
a) Explain the role of **segments** and **pages** in a DBMS

b) Explain the **differences** between:

- **Direct** and **Indirect Page Addressing**
- **Direct** and **Indirect Update Strategy**

Note:
For simplicity:
assume in exercises 1 - 2 that ***database page size = disk block size***.

Segments and Pages, Addressing and Update Strategies



Set-oriented interface, declarative QL

Record-oriented interface

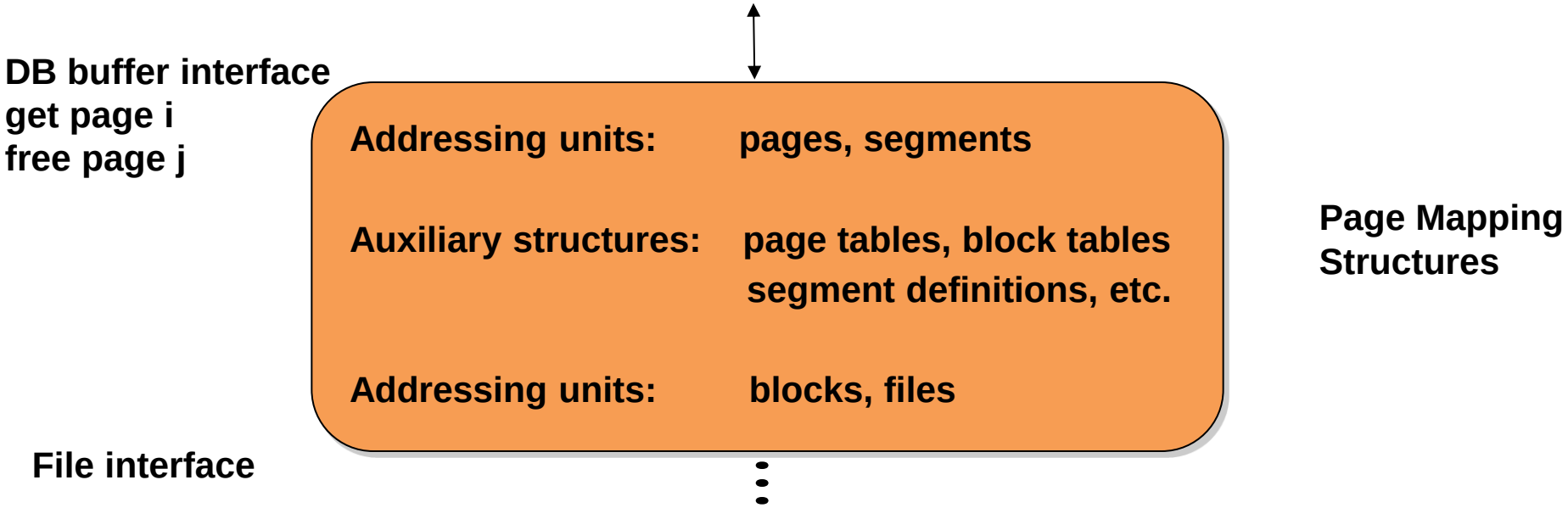
Internal interface

DB-buffer interface

File interface

Device interface

Segments and Pages, Addressing and Update Strategies



Additional level of abstraction above blocks and files

➔Introduction of **pages** and **segments**

Practical example – What does this look like?

TECHNISCHE UNIVERSITÄT DARMSTADT

Code in PostgreSQL bufmgr.c (www.postgresql.org)

…

```
00018  * ReadBuffer() -- find or create a buffer holding the requested page,
00019  *      and pin it so that no one can destroy it while this process
00020  *      is using it.
00021  *
00022  * ReleaseBuffer() -- unpin a buffer
00023  *
00024  * MarkBufferDirty() -- mark a pinned buffer's contents as "dirty".
00025  *      The disk write is delayed until buffer replacement or checkpoint.
```

…

Code in md.c (md ~ Magnetic Disc)

…

```
01583  *  _mdfd_getseg() -- Find the segment of the relation holding the
01584  *      specified block.
```

…

A buffer serves as the „frame“ of a page

Segments and Pages, Addressing and Update Strategies

TECHNISCHE UNIVERSITÄT DARMSTADT

Segments and Pages bring the following advantages:

1. Segments with **special properties** can be defined:
- ***Public*** *segments* for commonly accessible data structures
- ***Private*** *segments* for specific types of data - e.g. log data, statistical data, etc.
- ***Temporary*** *segments* for storing intermediate results with no recovery in case of a failure.

2. Both **direct and indirect updating** strategies can be implemented (in-place updating vs. shadow paging). [Flash Awareness?]

3. Can be used as **locking granules** (for Multi Granularity Locking, as we will see later)

Segments are realized differently in various commercial DBMS systems:
for e.g. **tablespace** in Oracle, **dbspace** in Informix.

Segments and Pages, Addressing and Update Strategies

TECHNISCHE UNIVERSITÄT DARMSTADT

b) Explain the differences between:

- Direct and Indirect Page Addressing
- Direct and Indirect Update Strategy

Note (for simplicity):

Database page size = Disk block size.

Segments and Pages, Addressing and Update Strategies

TECHNISCHE UNIVERSITÄT DARMSTADT

Direct Page Addressing

- Direct mapping of pages in segments to blocks in files
- Requires reservation of space at segment definition time

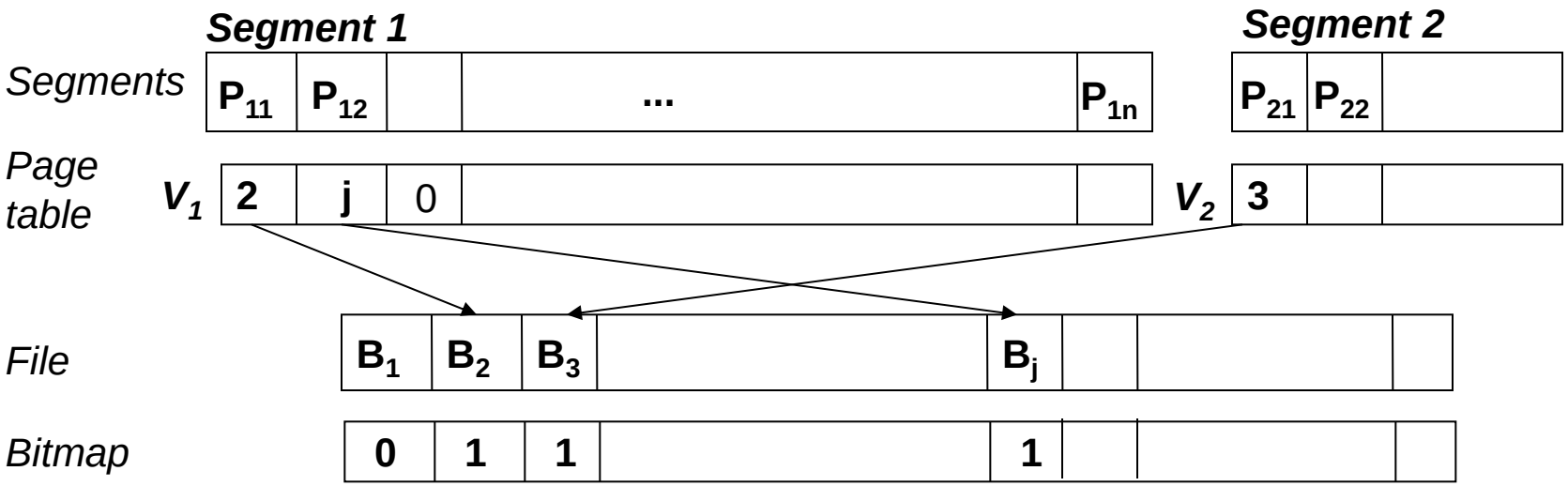
Problem:

Data reserved before needed ➔poor space utilization

(Low density when data is slowly inserted)

Segments and Pages, Addressing and Update Strategies

Indirect Page Addressing



- Space reserved on demand➔**Better space utilization**
- Large page table and bitmaps may have to be mapped to blocks and paged in and out of buffer
  (may cause additional page faults when requesting a page)
- *Additional Mapping Table (Page Table)*

Segments and Pages, Addressing and Update Strategies

TECHNISCHE UNIVERSITÄT DARMSTADT

Direct Page Addressing **static block allocation strategy**	**Indirect Page Addressing** **dynamic block allocation strategy**
Pro ● ***simple*** and easy to implement, easy address calculation ● allows efficient allocation of ***contiguous storage space***	**Pro** ● ***more flexibility*** - e.g. allows indirect update strategies ● ***better space utilization***, dynamic extension of segments
Contra ● ***poor space utilization*** ● overflow requires segment redefinition and possibly reloading	**Contra** ● ***mapping overhead*** introduced - PT often too big to keep in MM ● **space *fragmentation***, difficult to maintain clustering ● flexibility is usually needed at the level of extents and not of blocks

Segments and Pages, Addressing and Update Strategies

TECHNISCHE UNIVERSITÄT DARMSTADT

Direct vs. Indirect Update Strategies

All access through ***DB buffer interface***.
Two approaches for writing modified pages back to disk:

- ***Direct update strategy*** *(also called "Update in Place")*
  page written into the same block from which it was read.
- ***Indirect (delayed) update strategy***
  page written into a newly allocated block, old block not changed.

Choice of strategy is related to the way **transaction recovery** is implemented (to enforce the **atomicity** property of transactions).

The most popular realization of ***indirect update*** strategy is based on the so-called **Shadow Paging Method.**
What about Flash? -> No In Place Updates -> Need to avoid them

Segments and Pages, Addressing and Update Strategies

TECHNISCHE UNIVERSITÄT DARMSTADT

Direct Update Strategies	Indirect Update Strategies
● updates ***not atomic*** - failures may occur during write-back ● state after a crash ***"chaos-consistent“*** ● transaction recovery ***requires Write Ahead Logging (WAL)***	● updates ***atomic*** - enables atomic execution of actions composed of multiple page update operations ● state after a crash ***"action-consistent“*** ● transaction recovery can be simplified (***WAL is not necessarily needed***)

Exercise 2.2

TECHNISCHE UNIVERSITÄT DARMSTADT

Shadow Paging

Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Describe the Shadow Paging Method:
   What ***data structures are used****?*
   What happens when ***pages of a segment are updated*****?**

b) Which data structures need to be written atomically to disk when a segment is closed? Why?

Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

c) **Given is a segment S1 with 8 pages,** stored in a 16 block file F.

Page	1	2	3	4	5	6	7	8
Block	2	3	4	0	0	0	0	0

Lets assume that the **first 8 blocks of F are used** (blocks 1 thru 8), while the rest are free. Go step-by-step through the execution of the following operations on S1 and show how the auxiliary data structures (CMAP, MAP0, MAP1, V10, V11) are used:

- OpenSegment(S1)
- Read(Page 1)
- Update(Page 1)
- Read(Page 2)
- Update(Page 2)
- Update(Page 1)
- CloseSegment(S1) *// Savepoint / Checkpoint*

Exercise 2.2
Shadow Paging

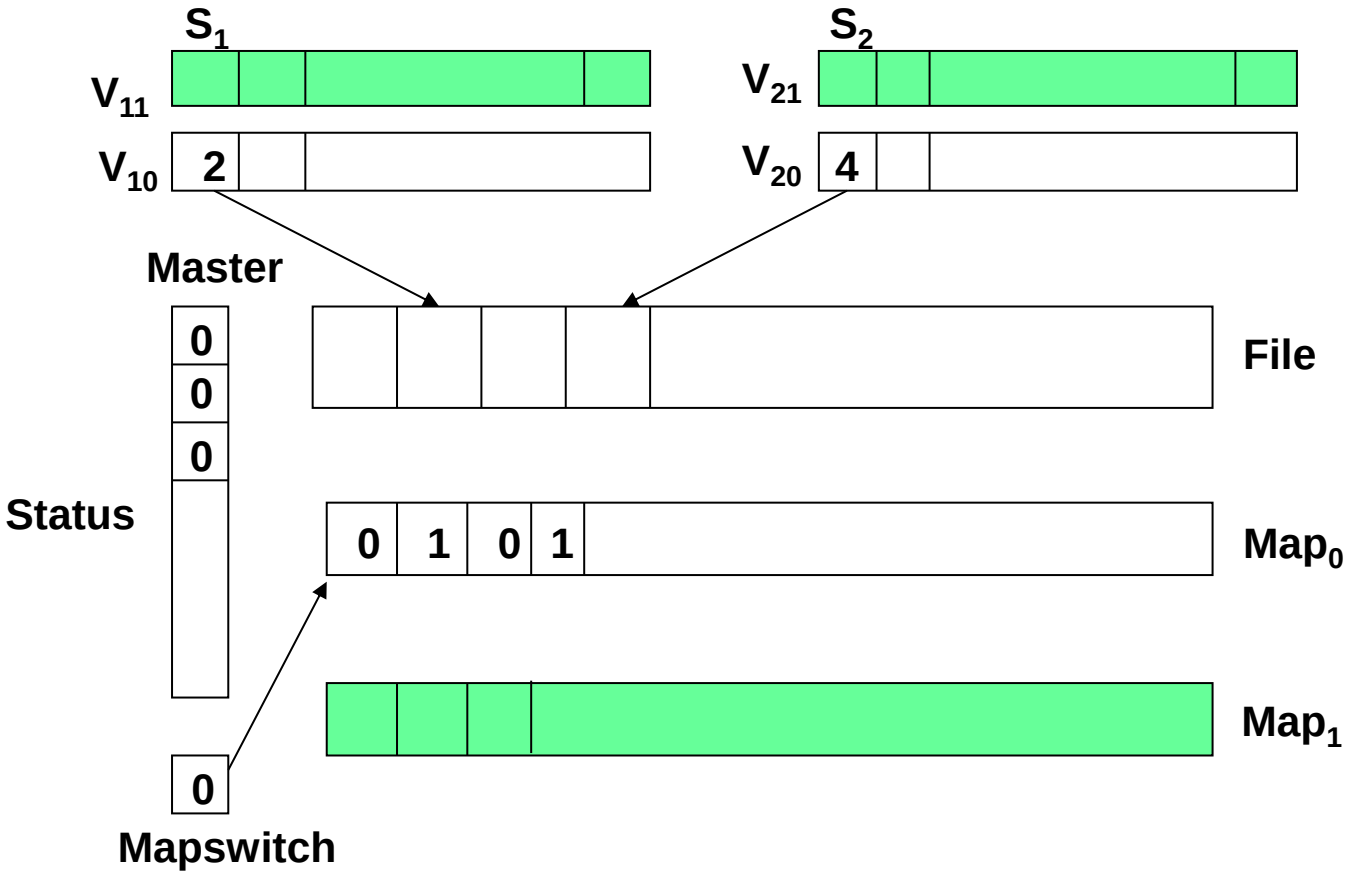
TECHNISCHE UNIVERSITÄT DARMSTADT

a) Describe the Shadow Paging Method.
What data structures are used?
What happens when pages of a segment are updated?

- ***Indirect page addressing*** with *indirect update strategy*
- ***Updated pages written to new blocks***
  ➔2 page tables used
- *Old pages (shadow pages) preserved* until save/checkpoint
- Switch to new page table considered ***atomic***

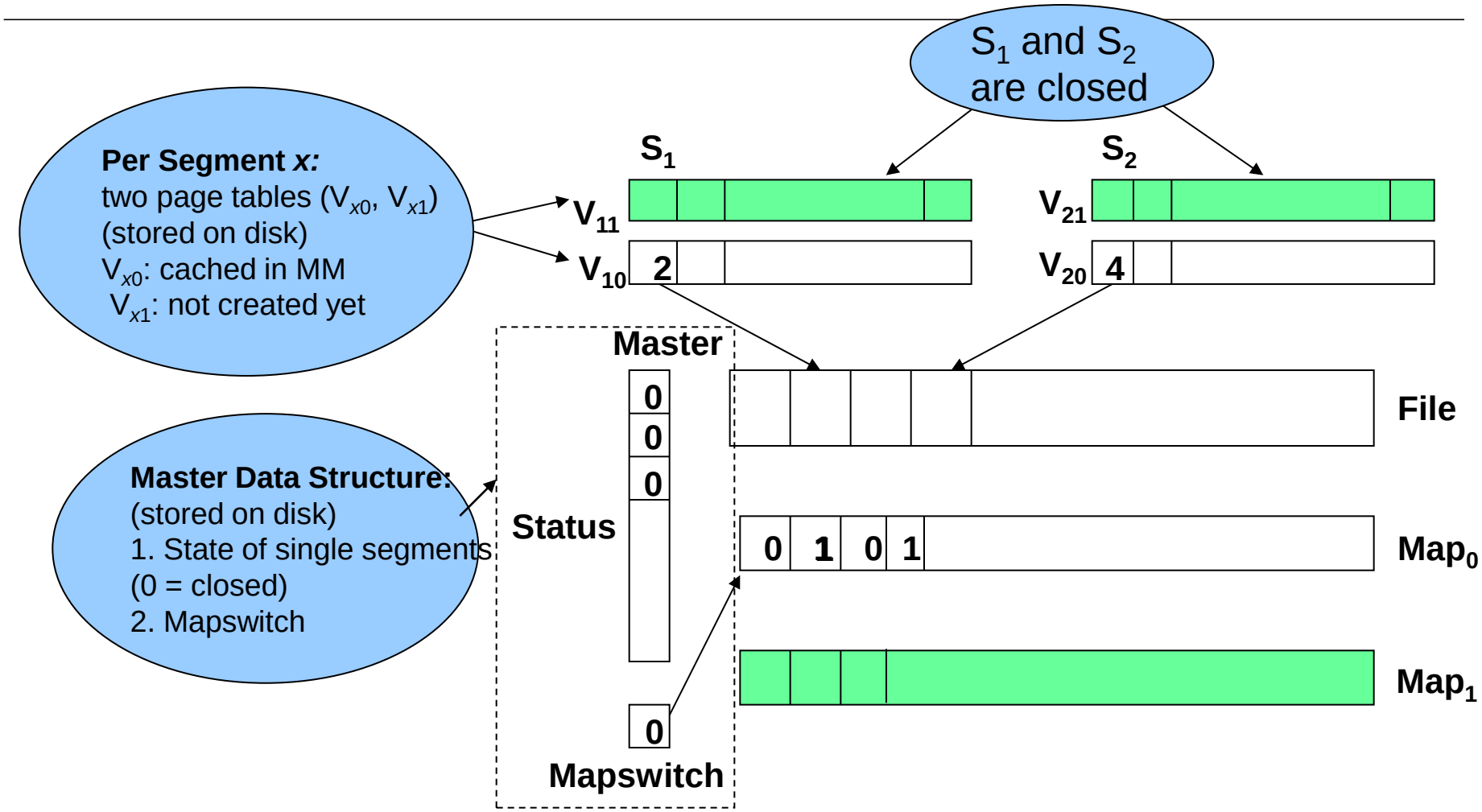
Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT



flashyDB DVS

Exercise 2.2
Shadow Paging Initial Situation

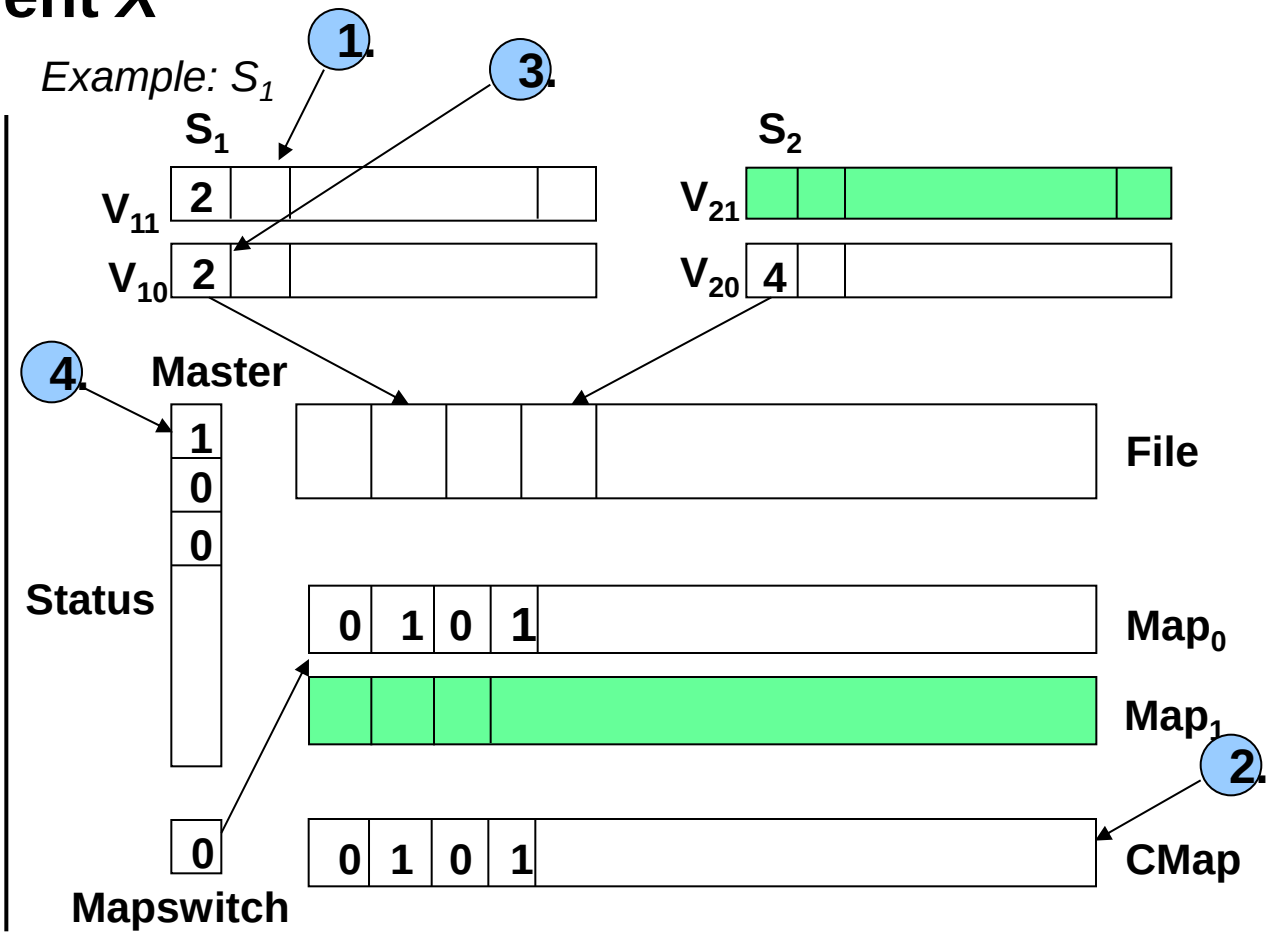


Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

Opening of a Segment *X*

1. **make a shadow copy** of segment's page table
   ➔ copy $\mathbf{V_{x0}}$ to $\mathbf{V_{x1}}$
2. make a **copy of the active file bitmap** in MM
   ➔**copy Map$_y$ to CMap**
3. for (i=1..N):
   init: shadowed[i] = false
4. **mark segment as open**
   ➔ master.status[*x*] = 1
5. **force-write master** record to disk



flashyDB DVS

Exercise 2.2
Shadow Paging

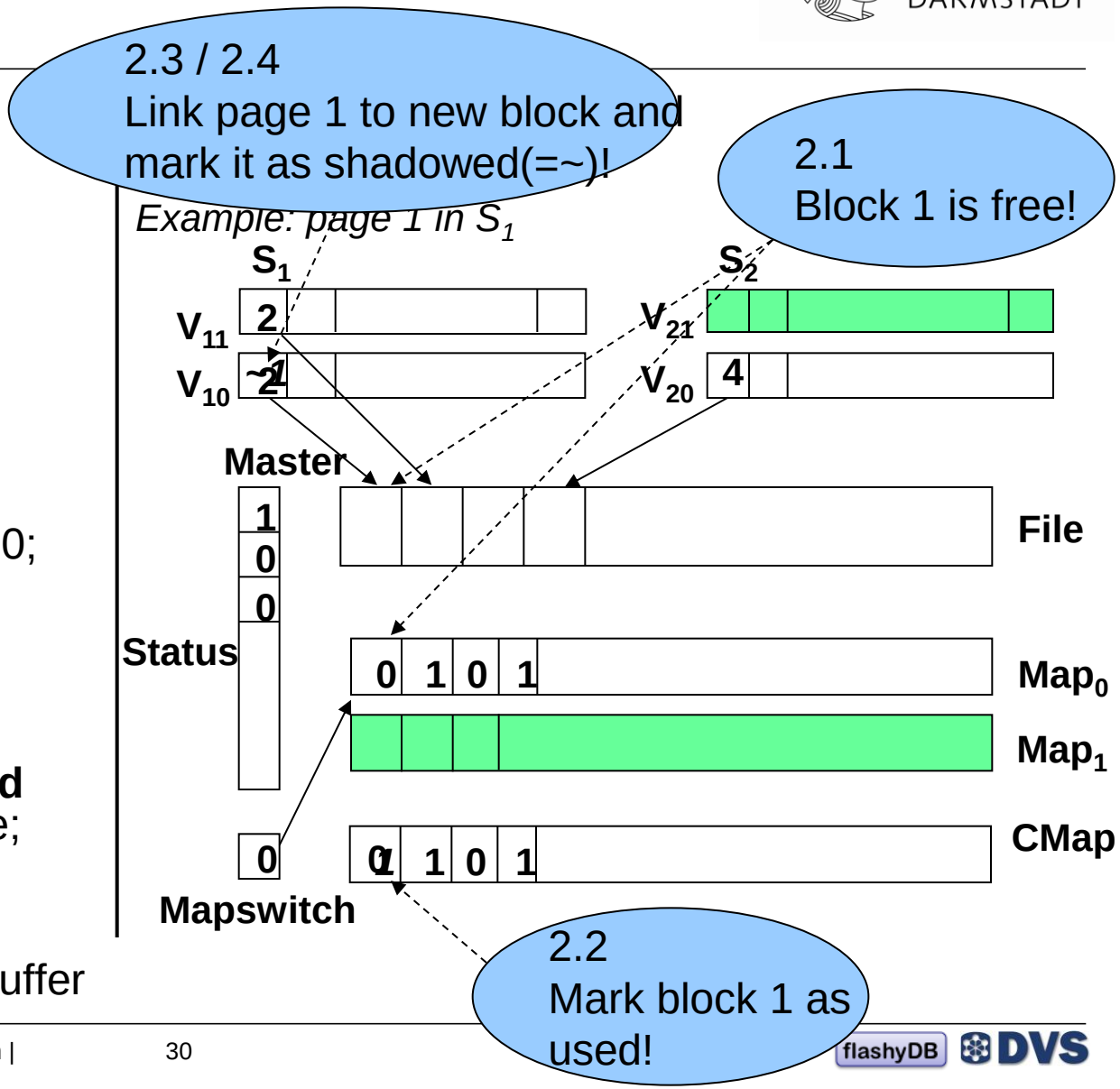
TECHNISCHE UNIVERSITÄT DARMSTADT

Update of Page *K*

1. if (page *k* not in buffer) {
   load it
   read file block $V_{x0}[k]$
   in buffer
   }

2. if (shadowed[*k*] = false) {
   find free file block
   ➔ find '*b*': CMap[*b*] = 0;
   mark block as used
   ➔CMap[*b*] = 1;
   link page to new block
   ➔ $V_{x0}[k] = b$;
   mark page as shadowed
   ➔ shadowed[*k*] = true;
   }

3. do **update** on page *k* in buffer



flashyDB DVS

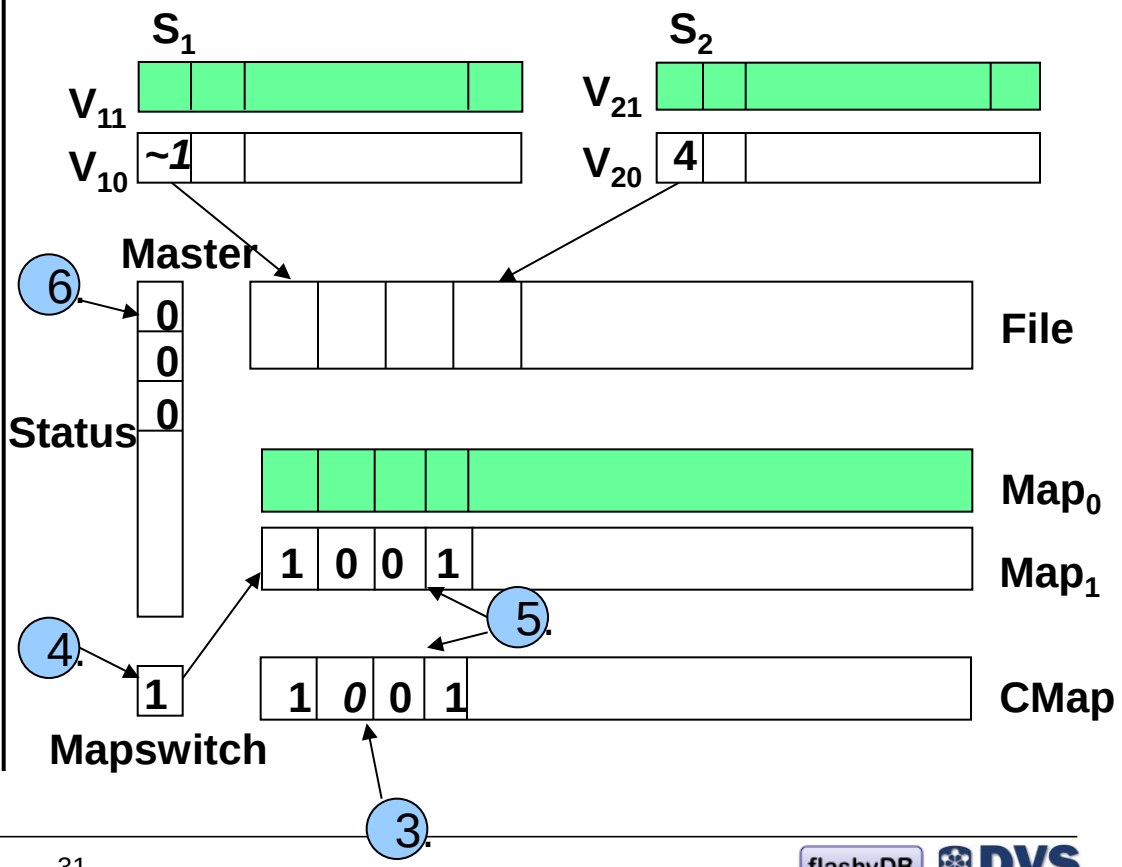
Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

Closing of Segment ‘x' (Save/Checkpoint)

1. **flush** all changes of PT $\mathbf{V_{x0}}$ to disk
2. **flush** all **dirty pages** of segment ‘*x*' to disk
3. **mark blocks occupied by shadows of updated pages as free**:
   for each page ‘*k*' such that (shadowed[*k*] = true)
   set CMap[V_{x1}[*k*]] = 0
4. **toggle active file Map**
   ➔Mapswitch = !Mapswitch
5. **copy CMap to Map$_y$,**
   where *y* = Mapswitch
6. **mark segment as closed**
   ➔ master.status[*k*] = 0
7. **force write master record** to disk

Example: page 1 in S_1



flashyDB DVS

Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

c) Given is a segment S_1 with 8 pages, stored in a 16 block file F.

Page	1	2	3	4	5	6	7	8
Block	2	3	4	0	0	0	0	0

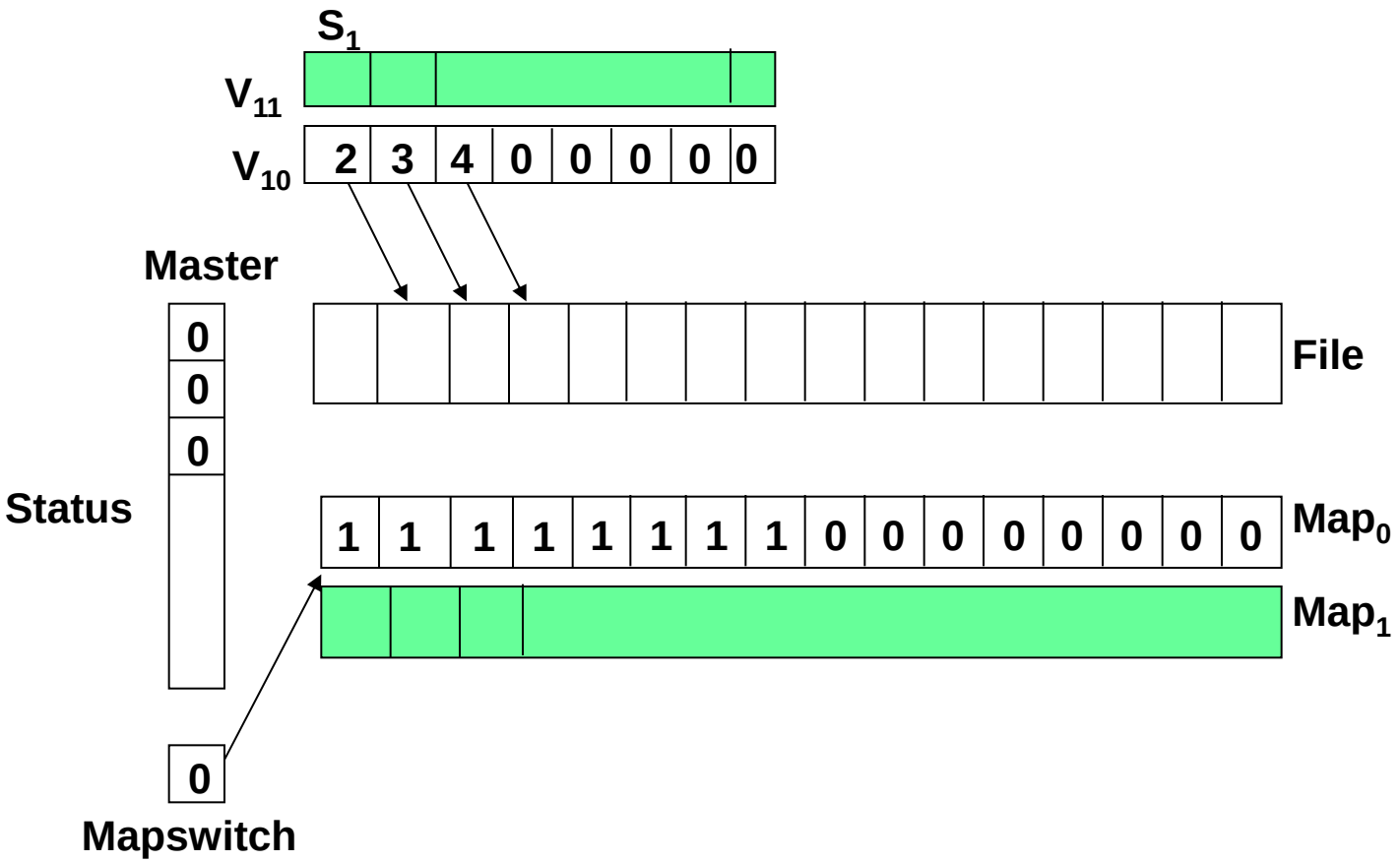
Lets assume that the first 8 blocks of F are used (blocks 1 thru 8), while the rest are free. Go step-by-step through the execution of the following operations on S_1 and show how the auxiliary data structures (CMAP, MAP_0, MAP_1, V_{10}, V_{11}) are used:

- OpenSegment(S1)
- Read(Page 1)
- Update(Page 1)
- Read(Page 2)
- Update(Page 2)
- Update(Page 1)
- CloseSegment(S1) *// Savepoint / Checkpoint*

Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

State before opening segment S1

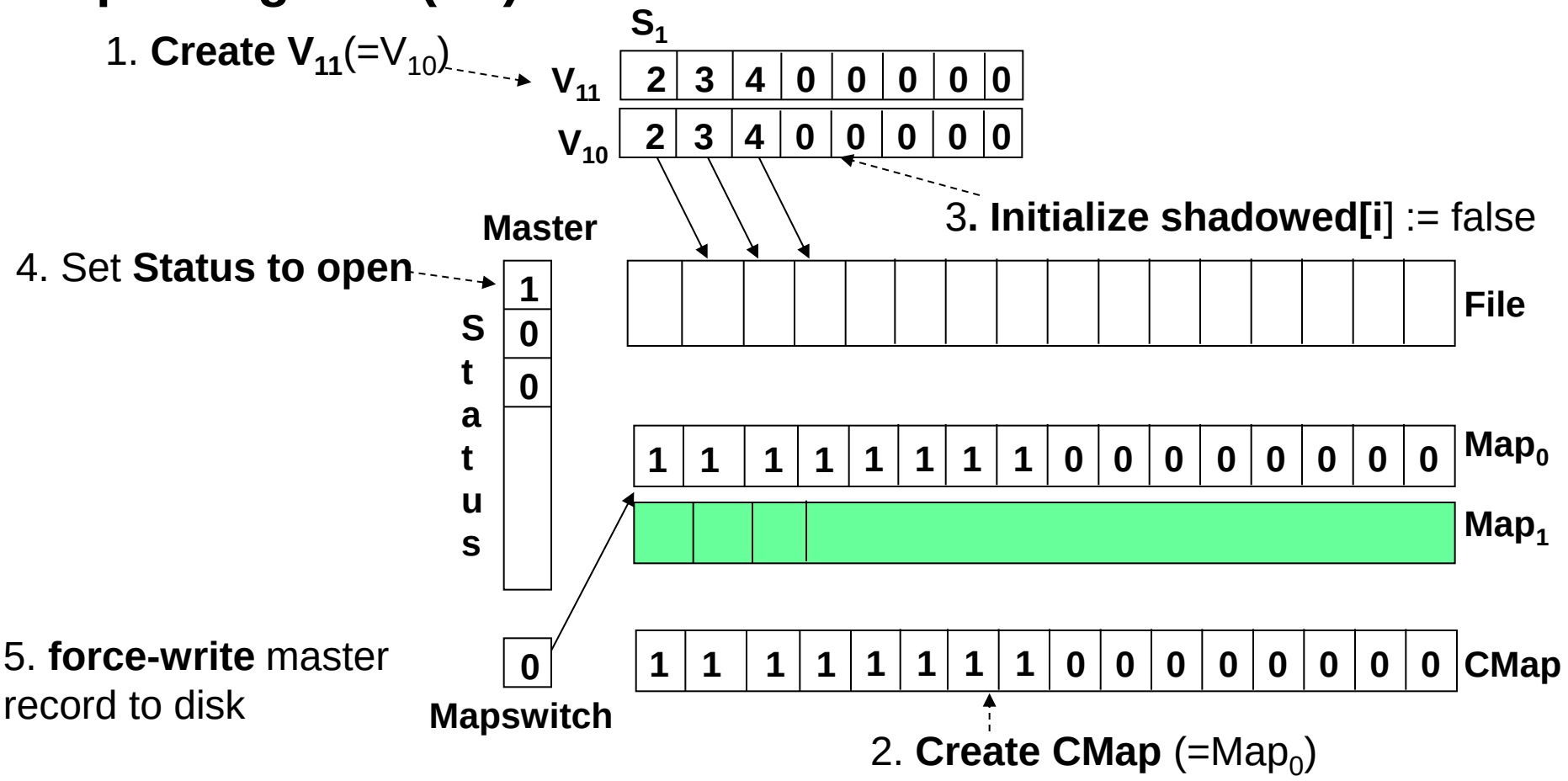


flashyDB DVS

Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

OpenSegment(S1)



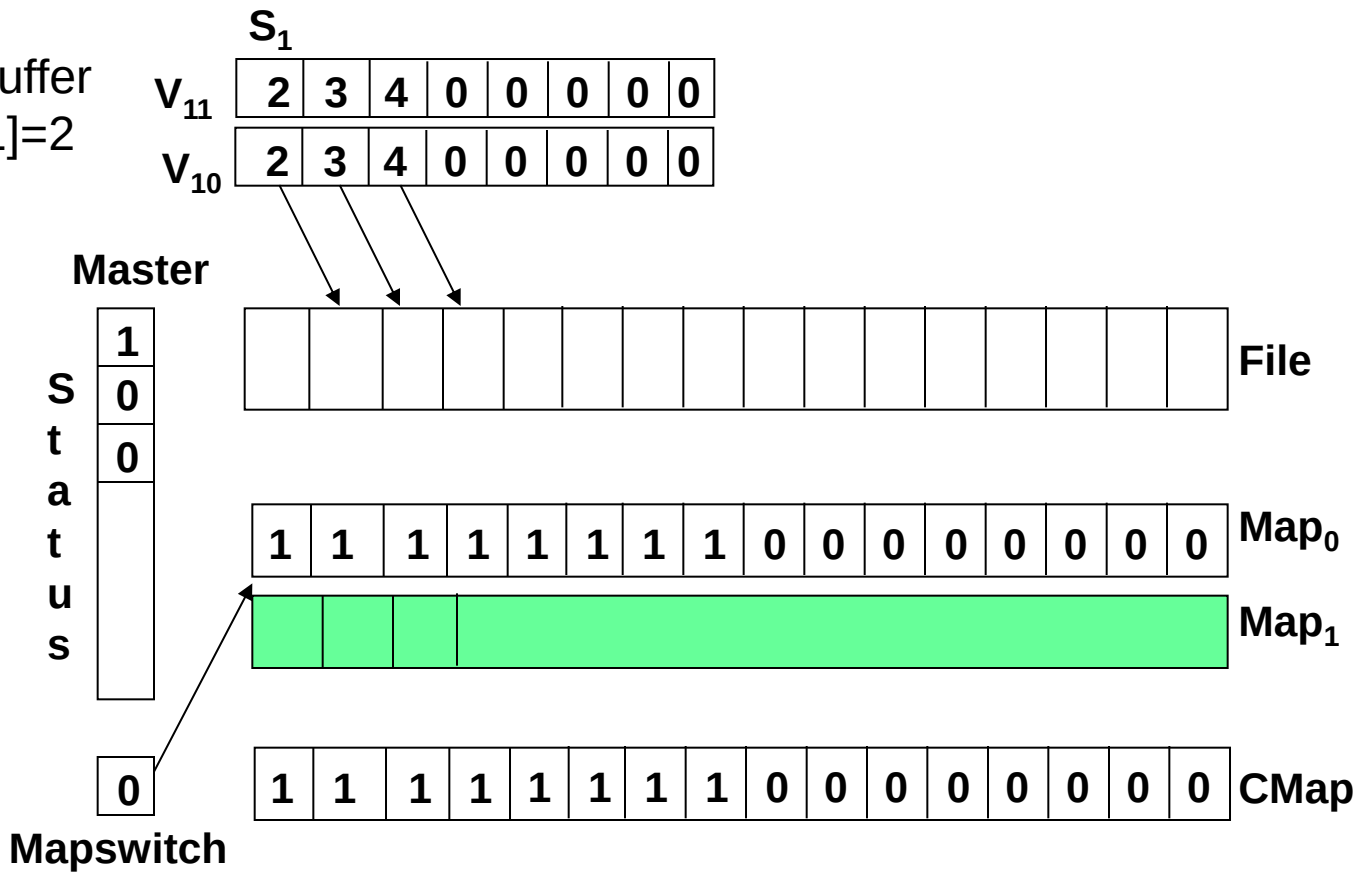
flashyDB DVS

Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

ReadPage(1)

1. Load page 1 in buffer
   ➔read block $V_{10}[1]=2$



flashyDB DVS

Exercise 2.2
Shadow Paging

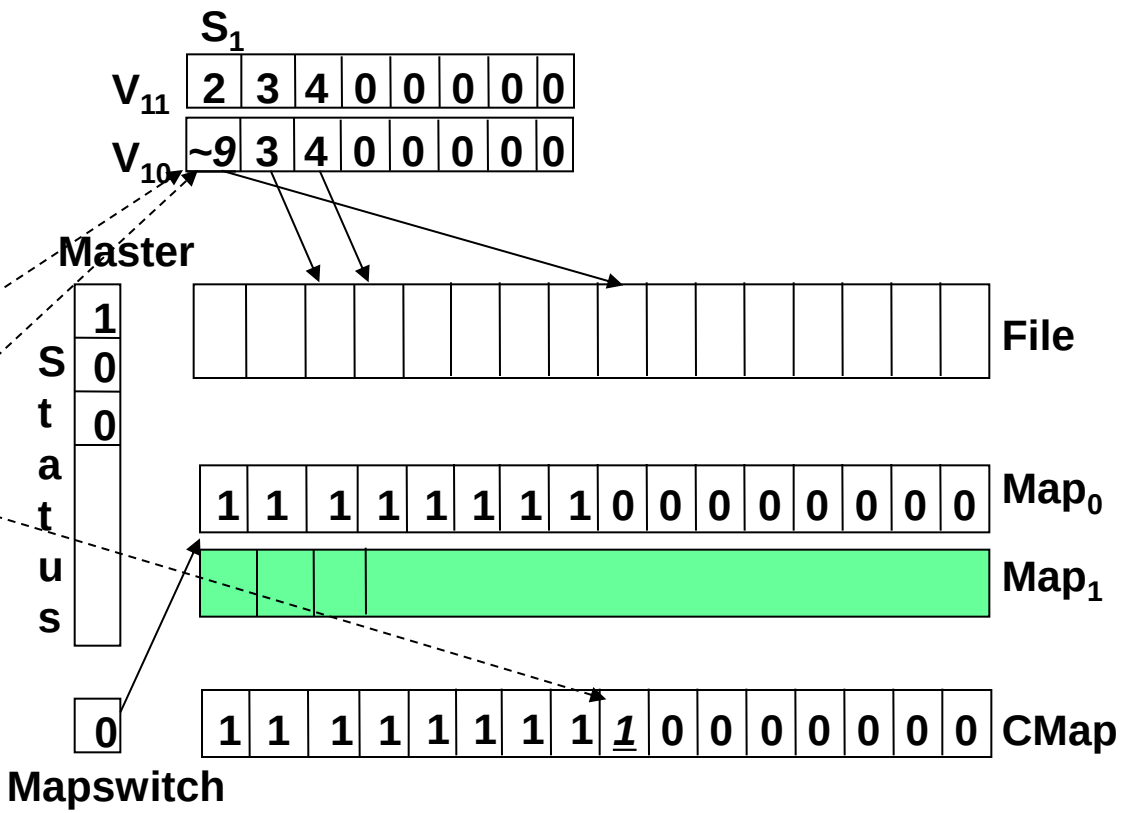
TECHNISCHE UNIVERSITÄT DARMSTADT

UpdatePage(1)

1. page already in buffer

2. since (shadowed[1] = false):
   - find free block '*b*': (CMap[*b*] = 0) →block 9
   - CMap[9] := 1
   - V_{10}[1] := 9
   - shadowed[1] := true;

3. do update on page 1 in buffer



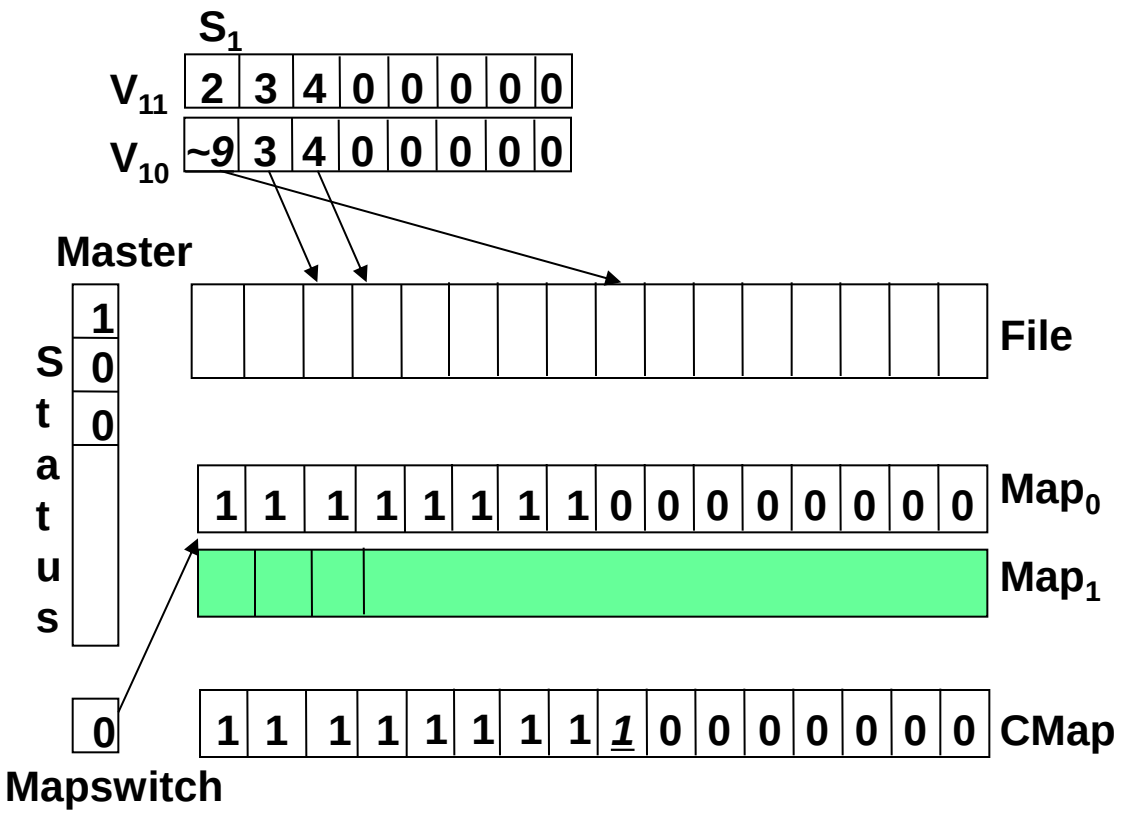
flashyDB DVS

Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

ReadPage(2)

1. Load page 2 in buffer
   ➔read block V_{10}[2]=3



flashyDB DVS

Exercise 2.2
Shadow Paging

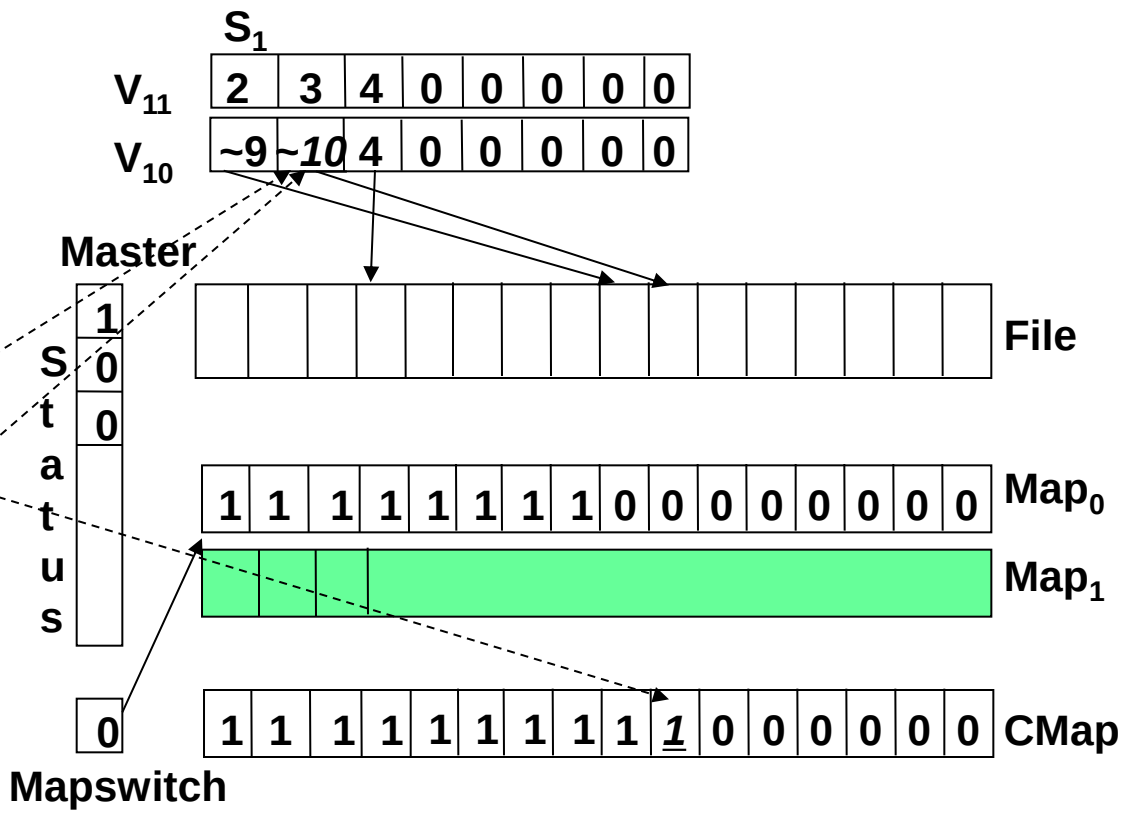
TECHNISCHE UNIVERSITÄT DARMSTADT

UpdatePage(2)

1. page already in buffer

2. since (shadowed[2] = false):
   - find free block '*b*': (CMap[*b*] = 0) →block 10
   - CMap[10] := 1
   - V_{10}[2] := 10
   - shadowed[1] := true;

3. do update on page 2 in buffer

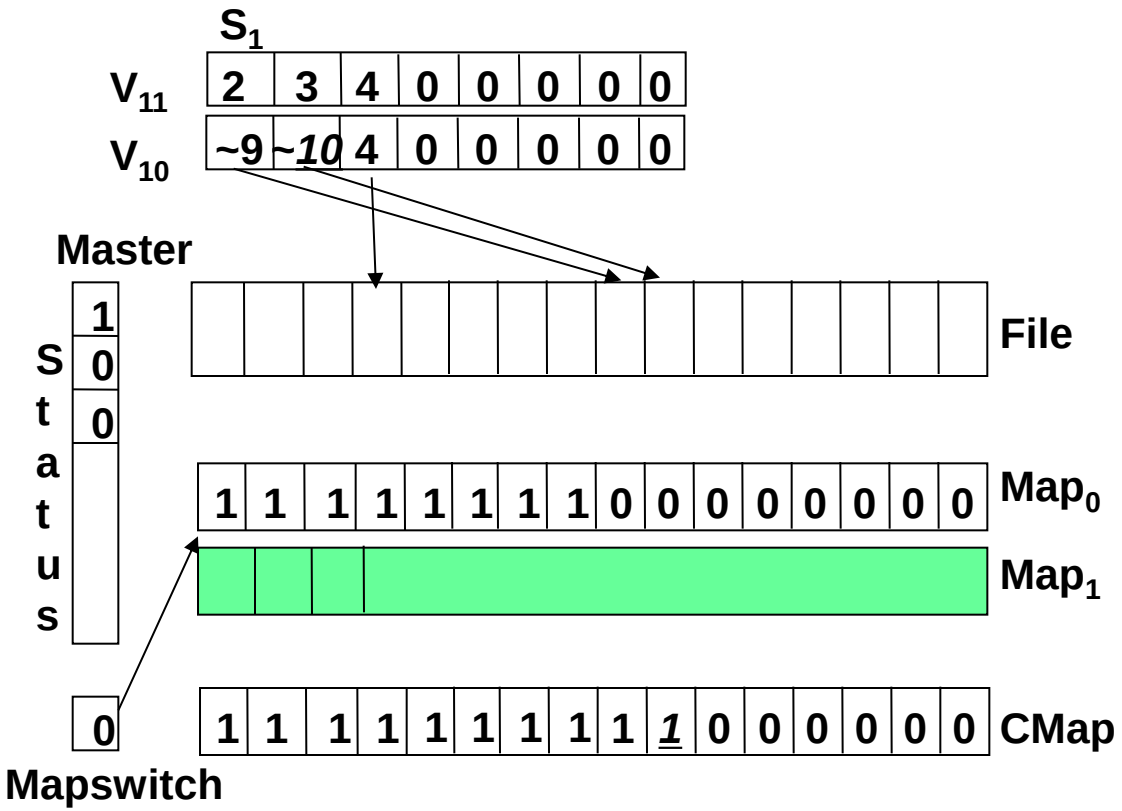


Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

UpdatePage(1)

1. page already in buffer
2. page **already shadowed** (shadowed[1] = true):
3. do update on page 1 in buffer



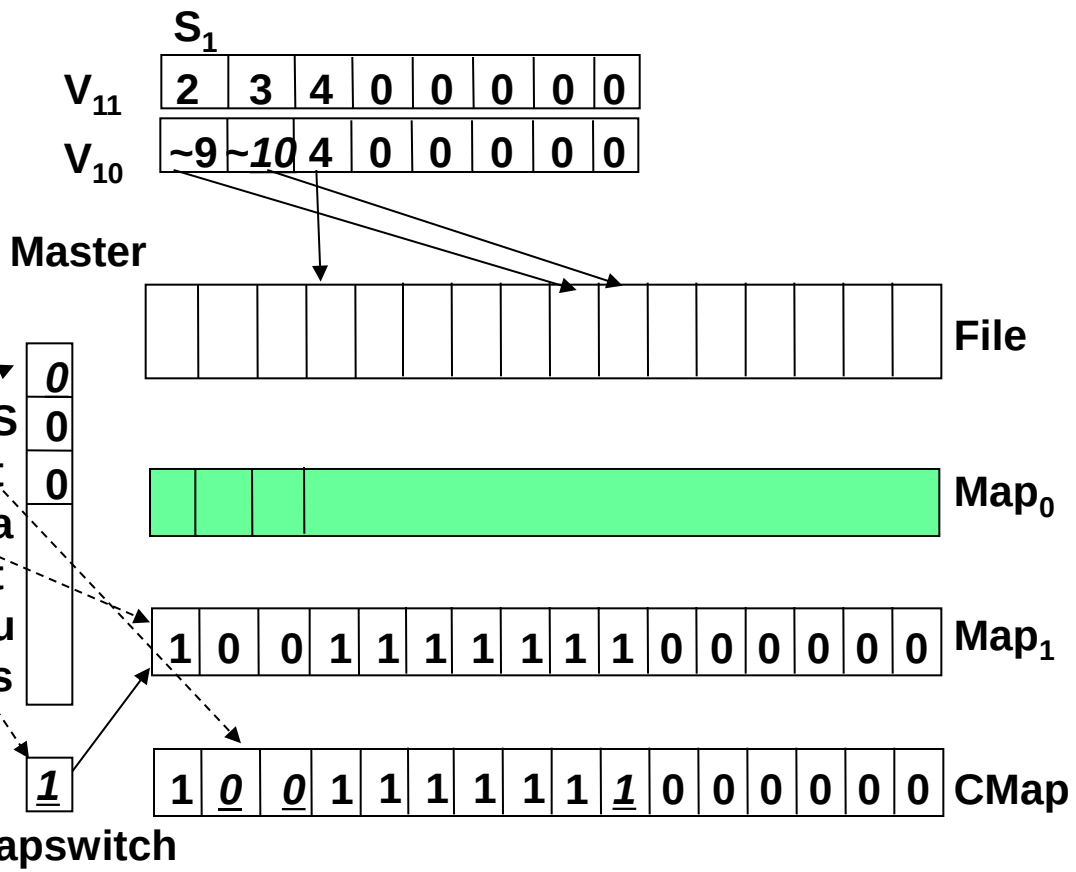
flashyDB DVS

Exercise 2.2
Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

CloseSegment(S1)

1. **flush** pages 1 and 2 to disk [block 9 and 10]
2. **discard shadows** (→CMap)
3. **master.mapswitch** := 1
3. **MAP_1 := CMAP**
4. **master.status[1] = 0**
5. **flush** all changes on PT **V_{10}** to disk
6. **force write master record** to disk



Exercise 2.3

TECHNISCHE UNIVERSITÄT DARMSTADT

Costs of Shadow Paging

Exercise 2.3
Costs of Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

Given is a segment ***S* of size 128 MB** stored in a file *F*:

- **Database page size** = 4 KB
- **Disk block size** = 512 Byte
- **Size of *F*** = 2x (Size of *S*) = 256 MB (Why 2x the size of S?)

a) Assuming that *S* is full and that there are no other segments stored in *F*, what is the ***maximum number of pages*** that ***can be updated*** with a single opening/closing of *S*?

b) Calculate ***the sizes*** (in blocks) ***of the free space bitmap Map_0*** (assuming that 1 bit is used per page) and ***the page table V*** of *S* (assuming that addressing is performed at the page level).

c) Calculate the number of block accesses at save/checkpoint if:
- ***only 1 page was updated***
- ***all pages were updated***

Assume that master record size is 2 blocks and that all updated pages need to be flushed at savepoint - i.e. none of them has already been flushed.

d) What is the overhead in c)?

Exercise 2.3
Costs of Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Maximum number of pages that can be updated with a single opening/closing of S

Each update requires 1 shadow page.
S is full (128MByte) and |F| = 2|S|:

➔F can accommodate all pages of S together with their shadows.

➔All pages can be updated with a single opening/closing of S.

Note:
The ***block size of F differs from the page size of S***!
Each page of S is stored in 8 blocks of F.

Exercise 2.3
Costs of Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

b) Calculate the sizes (in blocks) of the free space **bitmap Map_0** (assuming that 1 bit is used per page) and the page table *V* of *S* (assuming that addressing is performed at the page level).

Size of Map_0:

$$|F| = 2 \cdot |S| = 2 \cdot 128MB = 256MB$$

$$\#\ pages\ in\ F = \frac{256 \cdot 1024KB}{4KB} = \frac{2^8 \cdot 2^{10}}{2^2} = 2^{16}\ Pages \longrightarrow$$

$$|Map_0| = 2^{16}\ Bits = \frac{2^{16}}{8}\ Bytes = 2^{13}\ Bytes$$

$$|Map_0| = \frac{2^{13}}{512}\ Blocks = \frac{2^{13}}{2^9}\ Blocks = 2^4\ Blocks = 16 Blocks$$

Exercise 2.3
Costs of Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

b) Calculate the sizes (in blocks) of the free space bitmap Map_0 (assuming that 1 bit is used per page) and the **page table *V*** of *S* (assuming that addressing is performed at the page level).

Size of *V*:

$$\#\ pages\ of\ D = 2^{16}$$

$$\Rightarrow \text{Can use } 16 \text{ Bit Pointers } (2 \text{ Byte per address}):$$

$$|V| = 2^{15} \cdot 2 = 2^{16} Bytes = \frac{2^{16}}{512} Blocks = \frac{2^{16}}{2^9} Blocks = 2^7 Blocks = 128 Blocks$$

Pages in each V
128*1024/4

Need Both! |V10|
and |V11|

flashyDB DVS

Exercise 2.3
Costs of Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

c) Calculate the number of block accesses at save/checkpoint if:
- ***only 1 page was updated***
- ***all pages were updated***

Assume that *master record size is* ***2 blocks*** and that all updated pages need to be flushed at savepoint - i.e. none of them has already been flushed.

1. Only 1 page was updated

Data Structure	# Bl. Accesses
Updated page (4k, 512b) flushed	8
Dirty page table	1
Map_1 written	16
Master record written	2
TOTAL	8+19

4k/512b=8

1 Block is Dirty

Size of the Map

Exercise 2.3
Costs of Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

2. All pages were updated

Data Structure	# Bl. Accesses
All 2^{15} pages of S flushed	*$(2^{15})*8$*
All 128 blocks of page table dirty	***128***
Map_1 (16 blocks) written	***16***
Master record (2 blocks) written	***2***
TOTAL	**$(2^{15})*8 + 146$**

Exercise 2.3
Costs of Shadow Paging

TECHNISCHE UNIVERSITÄT DARMSTADT

d) What is the overhead in c)?

1 page (=8 blocks) updated ➔ 19 blocks overhead per Page!

All pages updated ➔ 146 blocks overhead but $(2^{15})*8$ Payload!

Exercise 2.4

TECHNISCHE UNIVERSITÄT DARMSTADT

Shadow Paging and Transactions

Exercise 2.4
Shadow Paging and Transactions

TECHNISCHE UNIVERSITÄT DARMSTADT

a) To what extent can ***the Shadow Paging Algorithm*** be used as a basis for implementing a ***Recovery Mechanism*** in a Transaction Processing System? What problems need to be addressed when ***concurrent transactions update the same segment***?

b) What are the ***advantages/disadvantages of recovery systems based on shadowing*** in comparison to conventional recovery systems based on in-place updating?

Exercise 2.4
Shadow Paging and Transactions

TECHNISCHE UNIVERSITÄT DARMSTADT

If **only one transaction** is allowed to **update a segment at a time**, then **recovery needs no logging** and the implementation of restart is straightforward. (No Concurrency, other TX may wait for the segment)

If **more than one transaction** can update a segment at a time, we **can't close the segment before all of them commit**. This increases transaction response time! (Latency - some have to wait for others)

2 possible approaches to address this problem:

- Use additional **Write-Ahead-Logging (WAL) with logical logging** to implement recovery.
- **Transaction-Oriented Shadow Paging** - Assuming that strict 2PL with page-level locks is employed, the shadow paging algorithm can be modified to support transaction-oriented recovery.

Exercise 2.4
Shadow Paging and Transactions

TECHNISCHE UNIVERSITÄT DARMSTADT

Pros of Shadow Paging	Cons of Shadow Paging
• **Enables atomic updates** • **Rollback** to an action-consistent state is easy. • **Can be used to completely avoid WAL.** • **Allows logical logging.** Logical logging has much smaller overhead than physical logging. • In case of a failure in which the log is destroyed, the probability of being able to restore the database to a consistent state is much higher if shadowing is used instead of update-in-place	• **Page tables are bulky** for large databases and their management causes **additional I/O overhead and page faults.** • **Checkpointing is slow –** FORCE approach used • **Destroys clustering** - data fragmentation leads to slower access times. • Storage of shadow pages requires additional disk space. This limits the length of update intervals.

Exercise 2.4
Shadow Paging and Transactions

TECHNISCHE UNIVERSITÄT DARMSTADT

Conclusion:

Update-in-place is considered the better strategy for most databases.

(Shadow Paging is good for very small DB)

Appendix

ORACLE Architecture and Terminology

Based on: ORACLE7 Server Concepts Manual

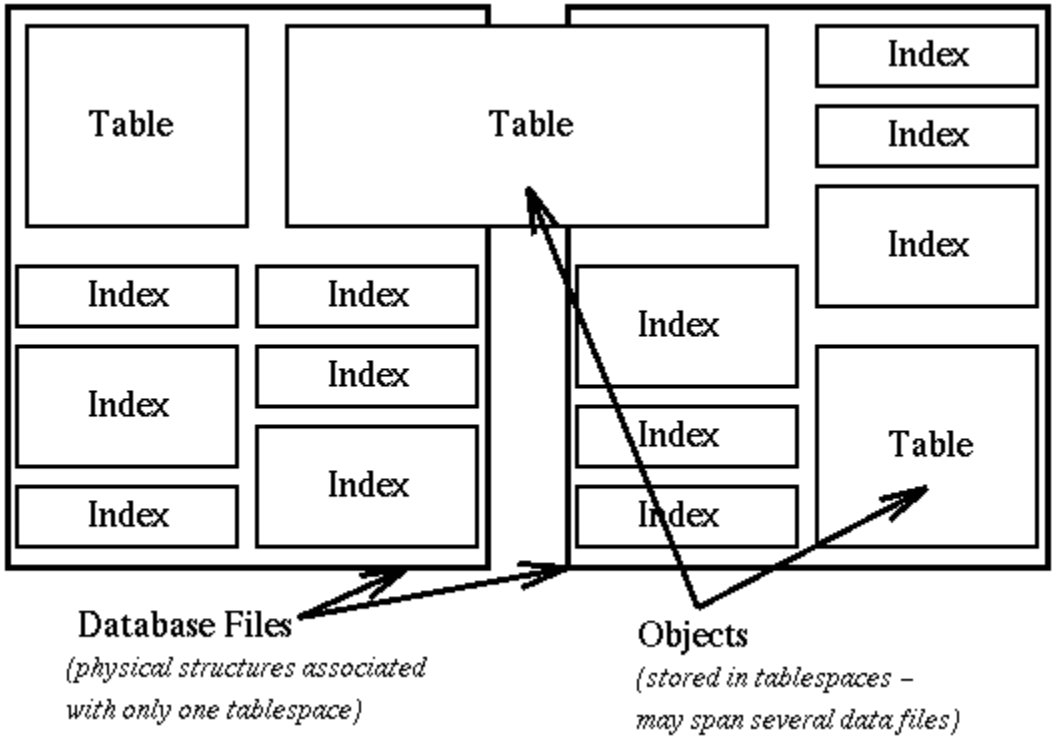
TECHNISCHE UNIVERSITÄT DARMSTADT

- **Databases and Tablespaces**
  An ORACLE database: one or more logical storage units called tablespaces. The database's data is collectively stored in the database's tablespaces.

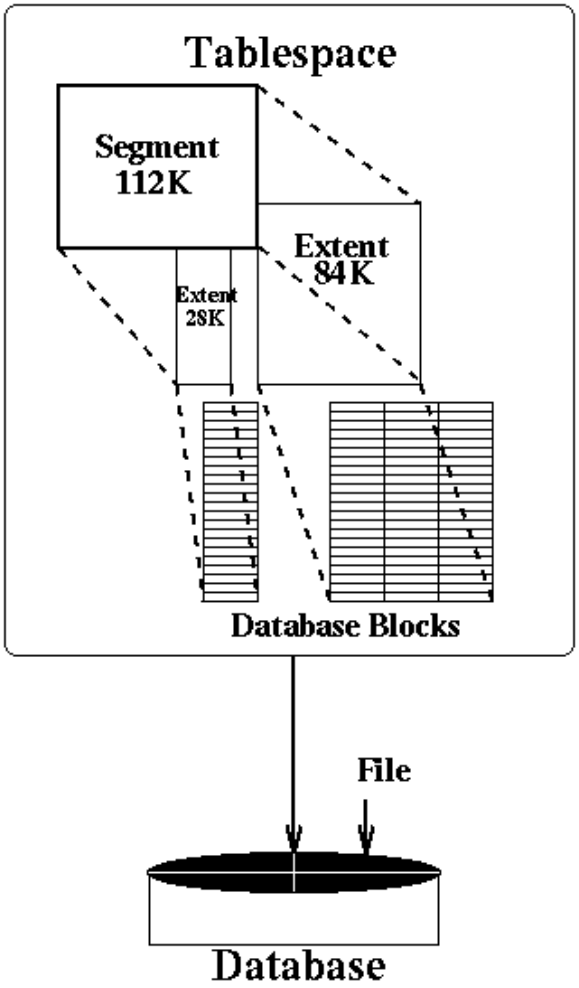
- **Tablespaces and Data Files**
  Each tablespace in an ORACLE database is comprised of one or more operating system files called data files. A tablespace's data files physically store the associated database data on disk.

- **Schema Objects, Segments, and Tablespaces**
  When a schema object such as a table or index is created, its *segment is created within a designated tablespace* in the database. The space for this table's data segment is allocated in one or more of the data files that constitute the specified tablespace. An object's segment allocates space in only one tablespace of a database.

ORACLE Architecture and Terminology

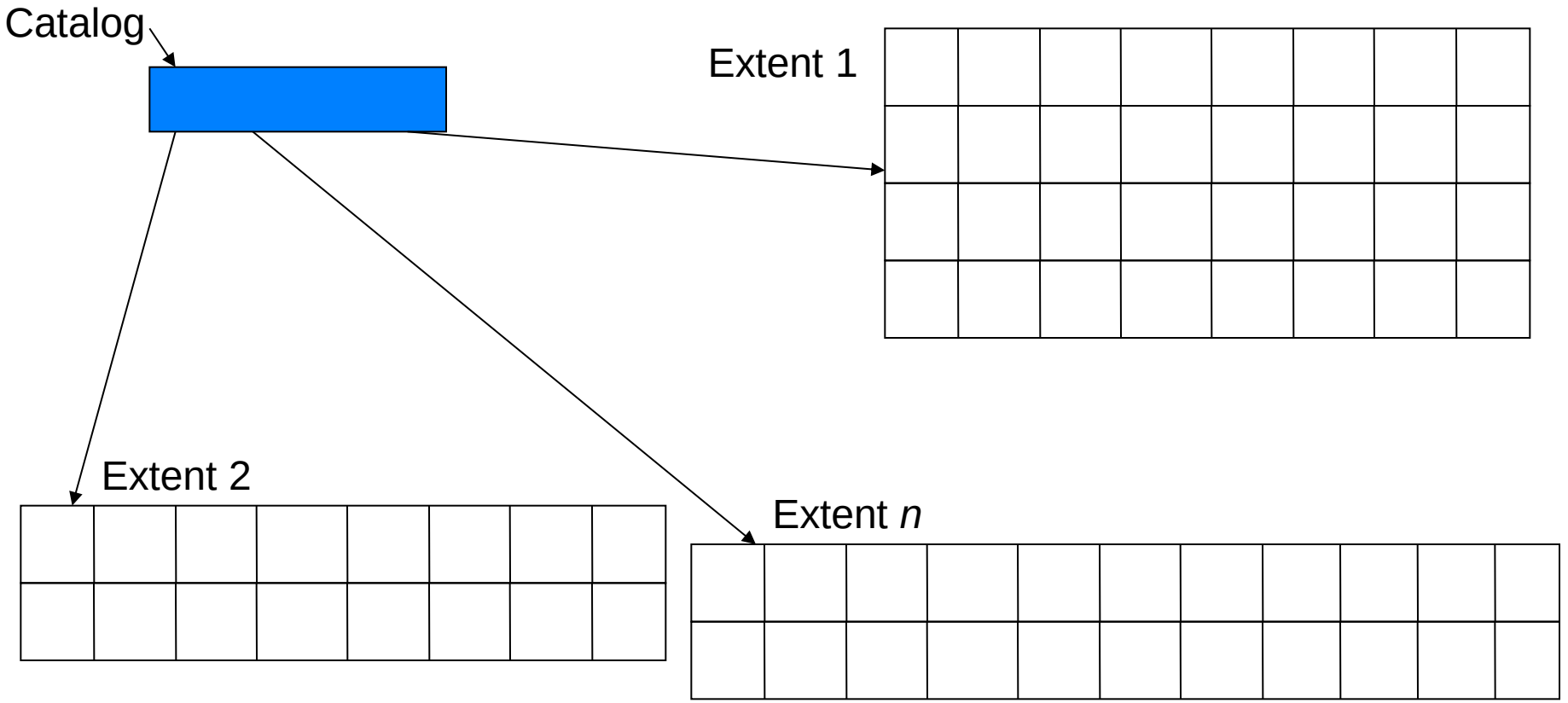


ORACLE Architecture and Terminology



- **Blocks**
- **Extents**
  An Extent is a specific number of contiguous data blocks that are allocated for storing a specific type of information. ***An Extent cannot span files.***
- **Segments**
  A Segment is a set of Extents which have been allocated for a specific type of data structure, and all are stored in the same Tablespace.
  A Segment can span files.

Dynamical Extent Allocation



Compromise between flexibility of dynamic allocation and the benefits of contiguous static allocation

DB2: Tables Spaces

http://www.ibm.com/developerworks/data/library/techarticle/0212wieser/0212wieser.html

TECHNISCHE UNIVERSITÄT DARMSTADT

Catalog table space

There is only one catalog table space per database, and it is created when the CREATE DATABASE command is issued. The catalog table space holds the system catalog tables. This table space is always created when the database is created.

Regular table spaces

Regular table spaces hold table data and indexes. It can also hold long data such as Large Objects (LOBs) unless they are explicitly stored in long table spaces. A table and its indexes can be segregated into separate regular table spaces…

Long table spaces

Long table spaces are used to store long or LOB table columns and must reside in DMS table spaces. They can also store structured type columns or index data. If no long table space is defined, then LOBs will be stored in regular table spaces.

System temporary table spaces

System temporary table spaces are used to store internal temporary data required during SQL operations such as sorting, reorganizing tables, creating indexes, and joining tables. At least one must exist per database.

User temporary table spaces

User temporary table spaces store declared global temporary tables. No user temporary table spaces exist when a database is created….