Database Management Systems II

Robert Gottstein

gottstein@dvs.tu-darmstadt.de

Announcement

- Next DB2 **Exam**
  - March 31st 2015
    - Starting at 9:00
    - Presumably 90 Minutes
  - Written exam

Recap Exercise 2

- Segments & Pages
  - Segments with Properties
- Addressing
- Update Strategies
  - Direct (Pro & Con)
  - Indirect – Shadows Paging (How does it work? Structures? Costs?)
  - In Place vs. Out of Place (which one is better for HDD/ SSD?)

Cutting some Topics…

- Recovery - Logging
- W.A.L.?

Excursion ORM

Object Relational Mapper

Slides provided by Prof. Ilia Petrov (University Reutlingen)

The ORM – Missmatch

- „DBMS is working with relations“
- „OO is working with objects“

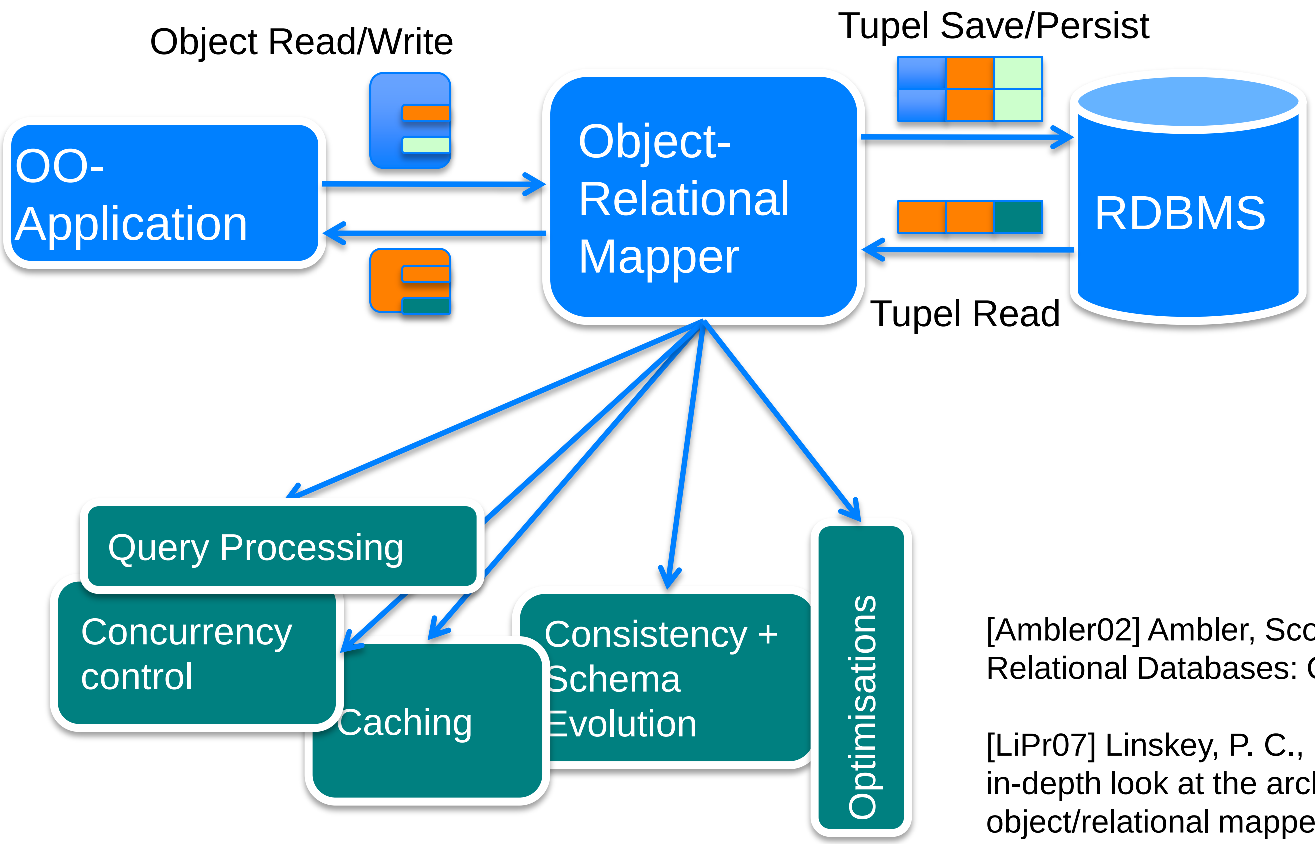
- Objects vs. Relations
  - Relations → mathematical principles; set operations
  - Objects → Behaviour; state; identity; encapsulation

- Want to combine the two worlds
  - Conversion needed! → ORM Mapper (Hibernate…) [Annotations]
    - e.G. String vs. Varchar

- Object Store/ Orion/ Swizzle/ Unswizzle

Slides provided by Prof. Ilia Petrov (University Reutlingen)

ORM – Object-Relational Mapper



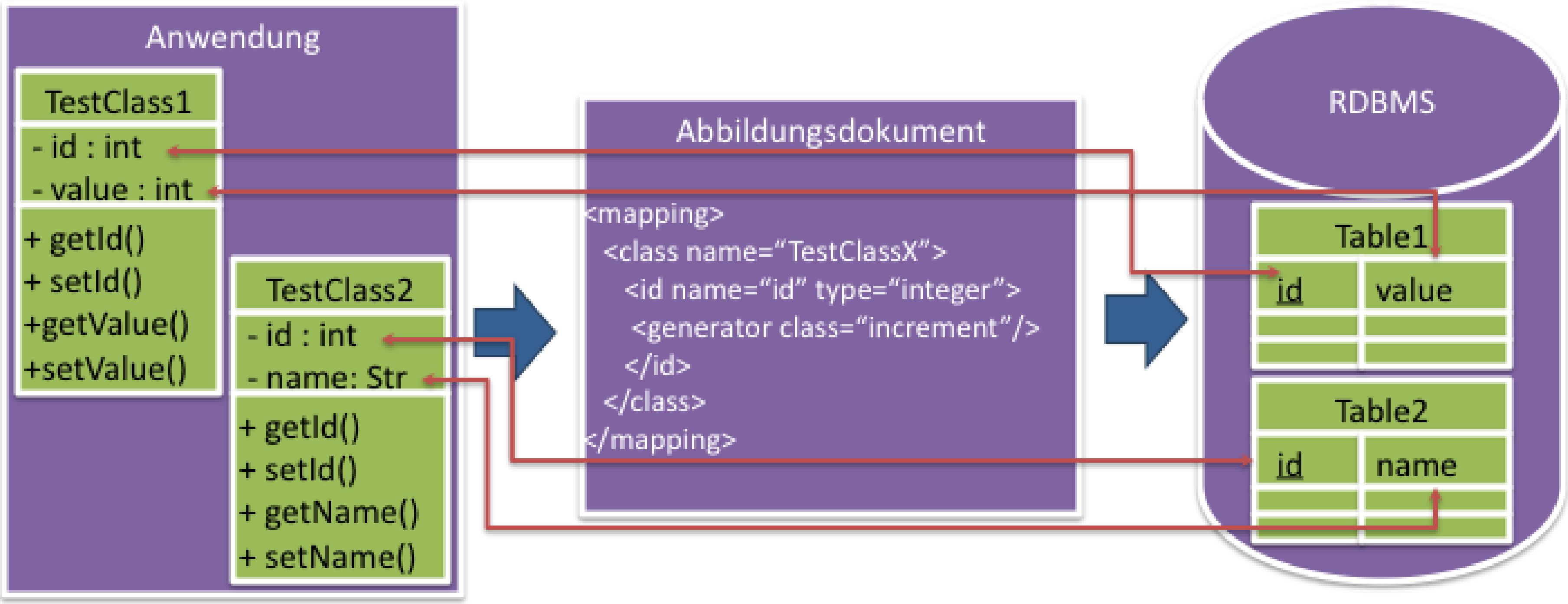
[Ambler02] Ambler, Scott. Mapping Objects to Relational Databases: O/R Mapping In Detail.

[LiPr07] Linskey, P. C., Prud'hommeaux, M. An in-depth look at the architecture of an object/relational mapper. In Proc. SIGMOD 2007.

JPA – Java Persistence API

- Persist Objects transparent – general classes
  - EJB – same API More functionality
- Management of persistent Objects
  - Inheritance, Aggregation, Composition
  - Refresh and Merge
  - Track Changes
  - Load
- Search and Find Objects
- Object-Caching
- Transactions

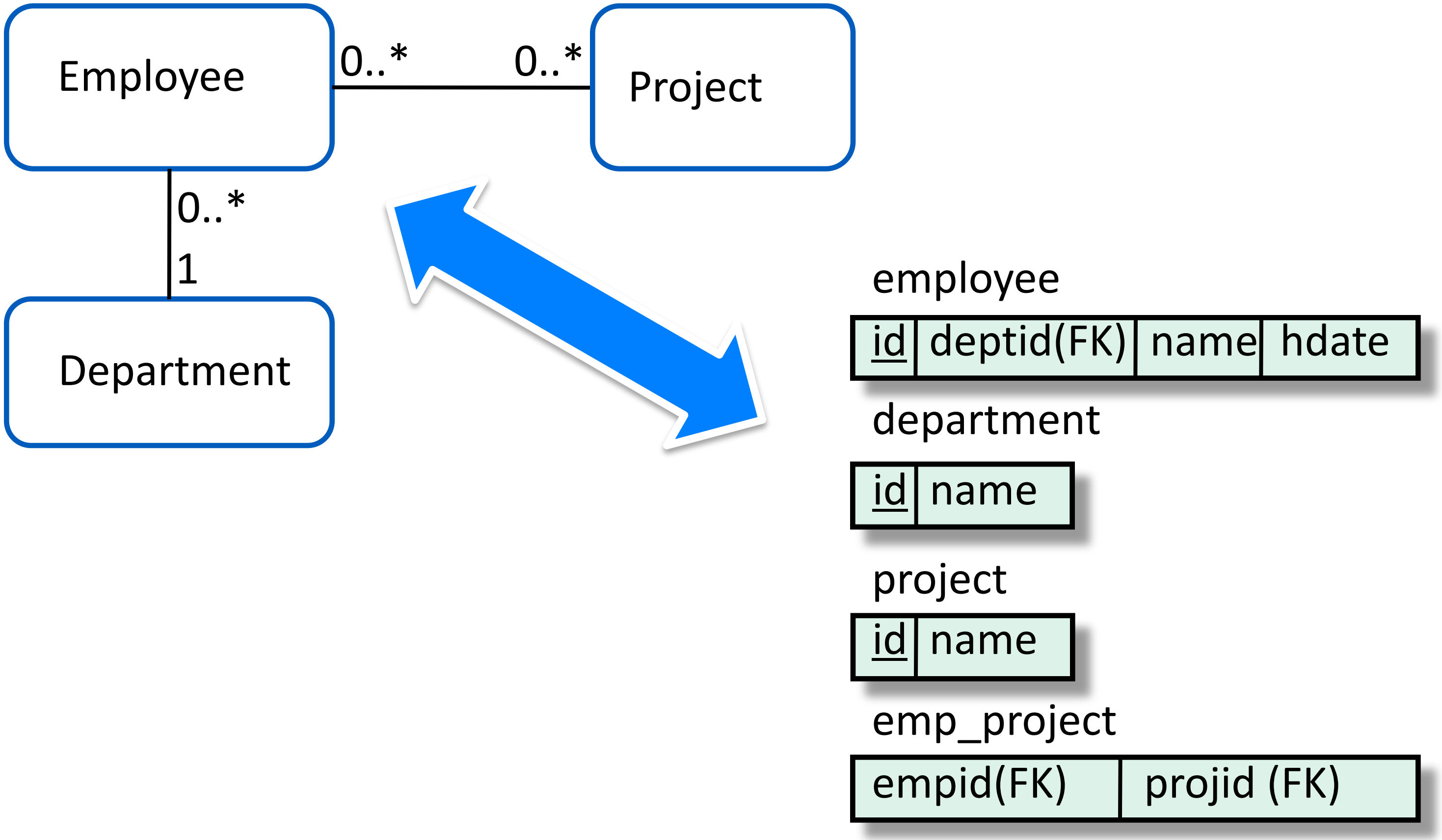
Mapping


Anwendung
TestClass1
- id : int
- value : int
+ getId()
+ setId()
+getValue()
+setValue()
TestClass2
- id : int
- name: Str
+ getId()
+ setId()
+ getName()
+ setName()
Abbildungsdokument
RDBMS
Table1
id
value
Table2
id
name

```
<mapping>
  <class name="TestClassX">
    <id name="id" type="integer">
     <generator class="increment"/>
    </id>
  </class>
</mapping>
```

Simple Mapping

- Schema Mapping: **Entity to** tables (vice versa)



Employee 0..* 0..* Project

Employee 0..* 1 Department

employee

id	deptid(FK)	name	hdate

department

id	name

project

id	name

emp_project

empid(FK)	projid (FK)

Example – POJO (Plain Old Java Object)

```
public class Employee {

  public Integer id;
  public String name;
  public long salary;
  public Department department;
          public Address address;

  public List<Project> projects;


}
```

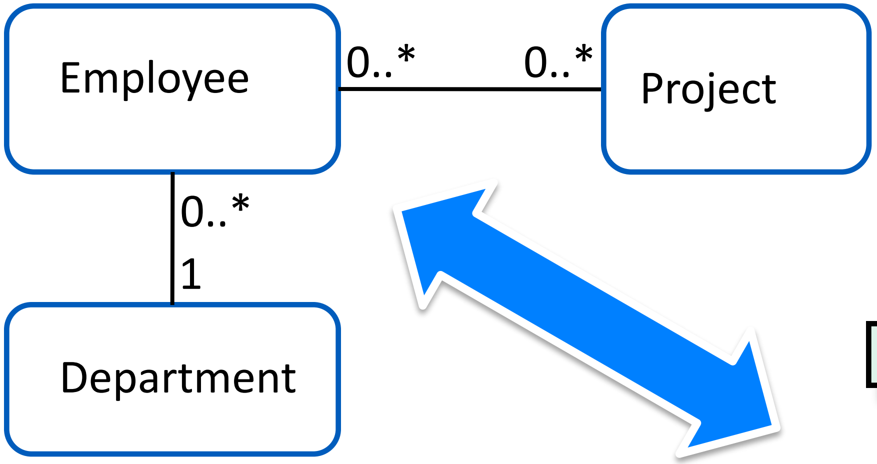
- Standard-Constructor (No Arguments)
- Define Identity Primary Key(p.key, id)
- No use of **final** allowed
- Annotation of fields / getter setter methods

Example – POJO (Plain Old Java Object)

```
@Entity (name=“Employee")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE | JOINED | TABLE_PER_CLASS)
public class Employee {

    @Id  //@GeneratedValue(strategy=TABLE | SEQUENCE | AUTO)
    public Integer id;

    @Column(name="NAME“, table=“Employee“, unique=false, nullable=false, insertable=true,
    updatable=true)
    public String name;
    public long salary;

    @ManyToOne
    @Basic(fetch=FetchType.EAGER | LAZY)
    public Department department;

    @OneToOne(cascade = CascadeType.ALL | MERGE | PERSIST | REFRESH | REMOVE)
    public Address address;

    @ManyToMany(mappedBy = "emps")
    public List<Project> projects; //public class Project{ ... @ManyToMany List<Employee> emps;...}
}
```

Simple Mapping

- Schema Abbildung: **Entitäten** auf Tabellen und umgekehrt



employee

id	deptid(FK)	name	hdate

department

id	name

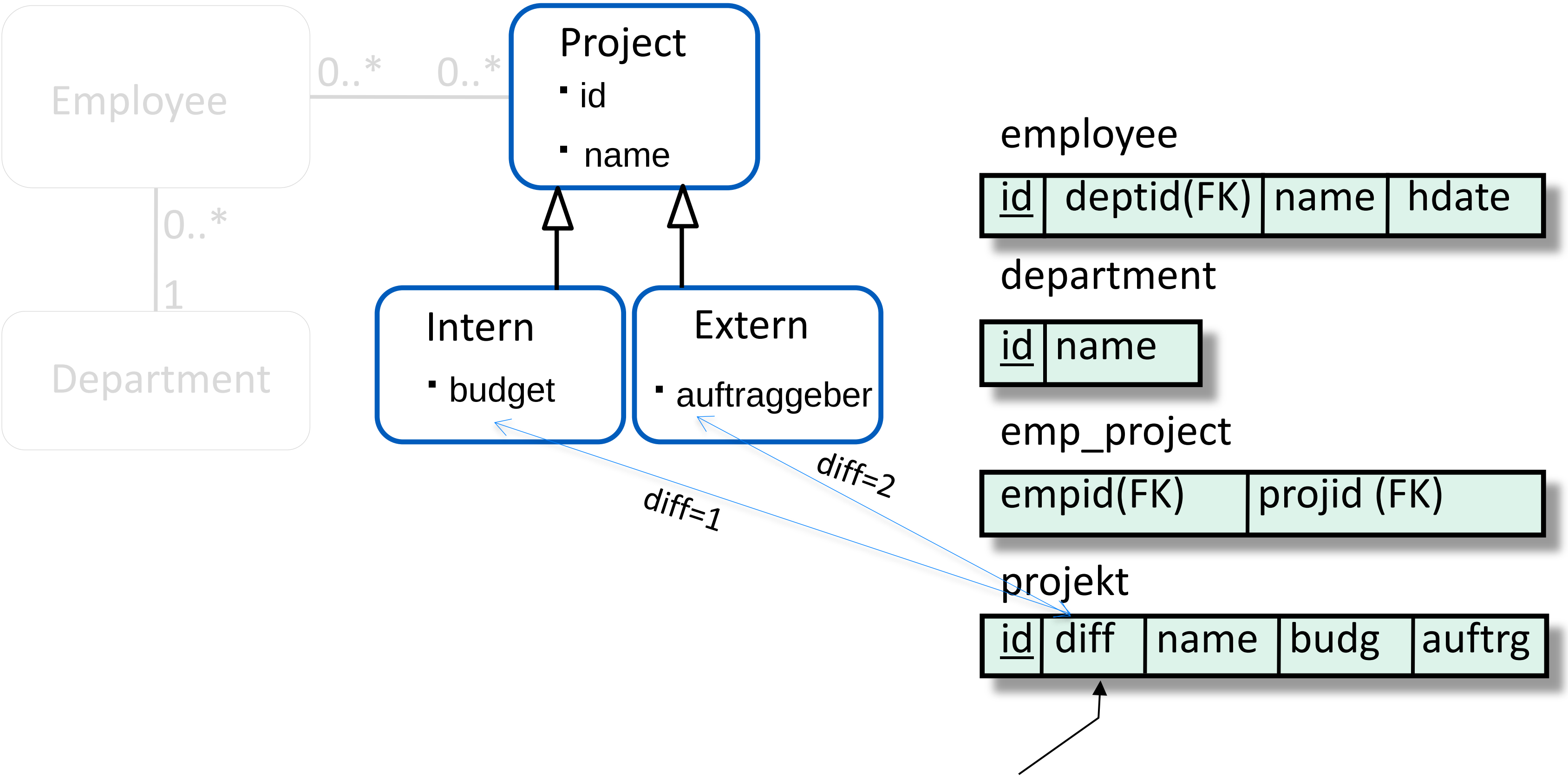
project

id	name

emp_project

empid(FK)	projid (FK)

Inheritance Mapping – Single Table



employee

id	deptid(FK)	name	hdate

department

id	name

emp_project

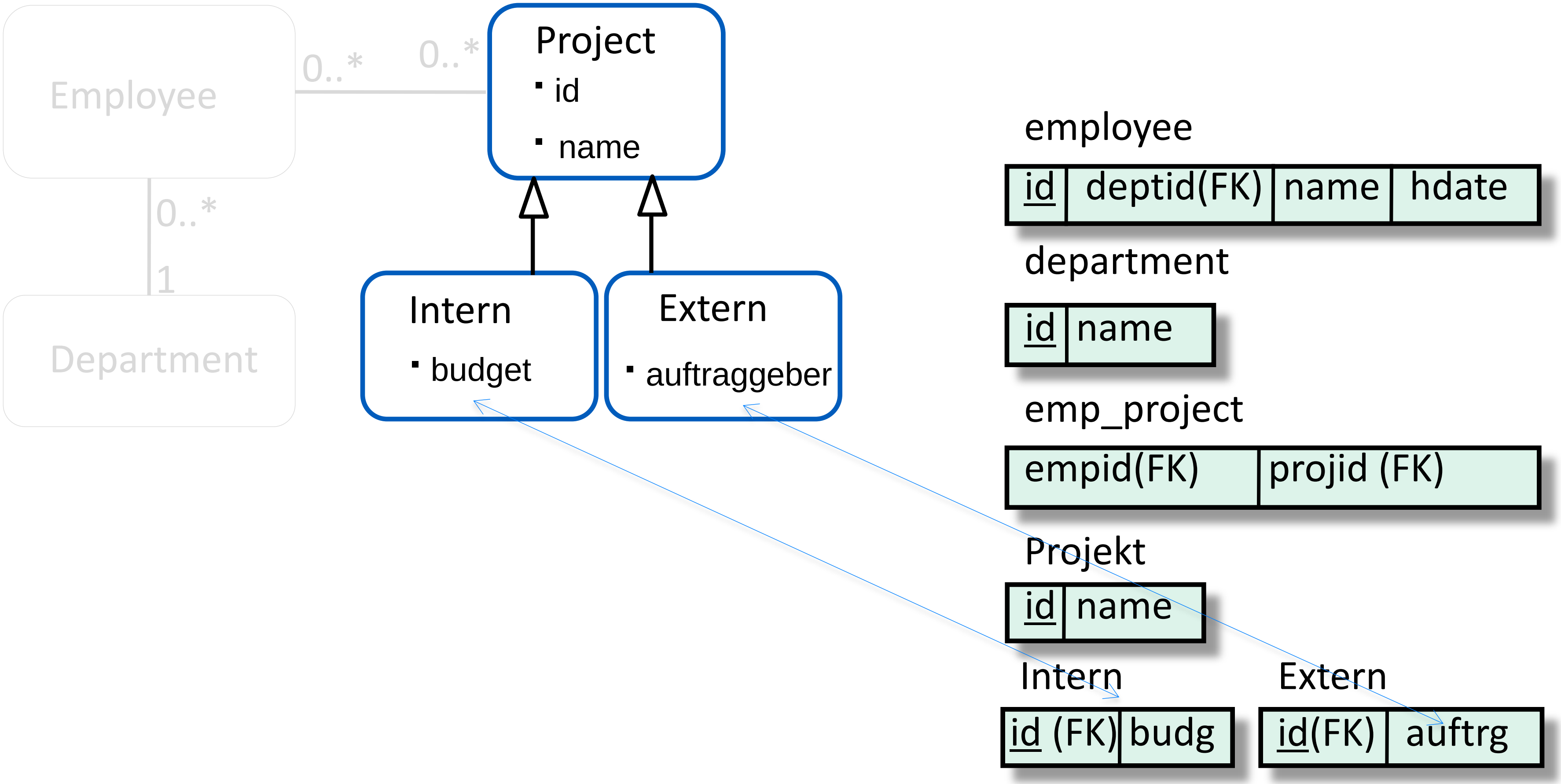
empid(FK)	projid (FK)

projekt

id	diff	name	budg	auftrg

Differential Attribut – not visible externaly

Table per Class



employee

id	deptid(FK)	name	hdate

department

id	name

emp_project

empid(FK)	projid (FK)

Projekt

id	name

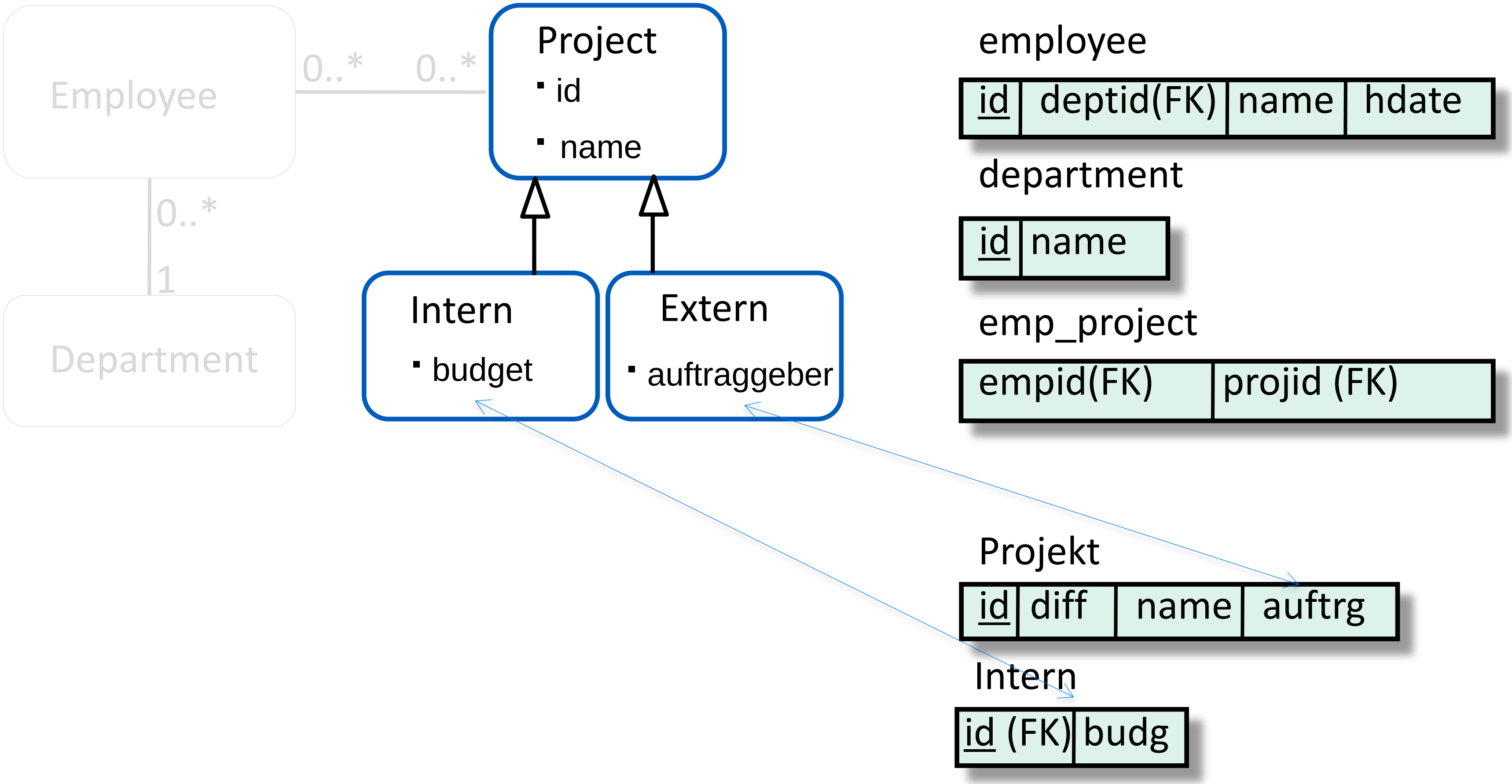
Intern

id (FK)	budg

Extern

id(FK)	auftrg

(Incomplete) Table per Subclass



employee

id	deptid(FK)	name	hdate

department

id	name

emp_project

empid(FK)	projid (FK)

Projekt

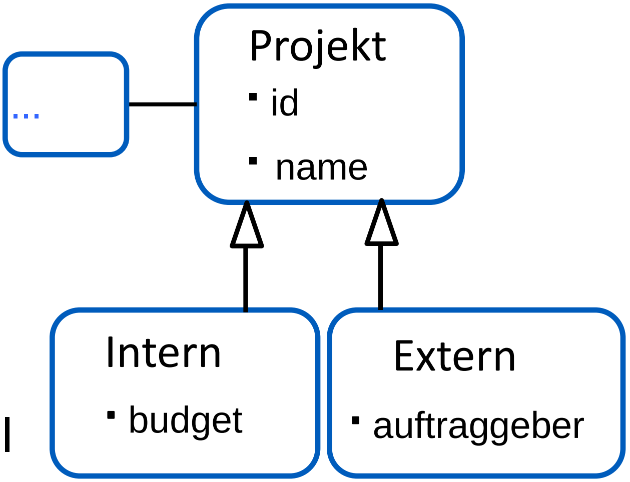
id	diff	name	auftrg

Intern

id (FK)	budg

Inheritance

- ORM supports:
  - Inheritance
  - Polymorph relations



- Different Mappings
  - Table per Class - Polymorphism & Class-Relations not considered in Relational Model
  - Table per Class Structure – Polymorphism enabled through denormalising of Rel.Schema
    - Use differential attributes for Type-Information
  - Table per Subclass- “is a“ relation and “has a” (Foreign Key) relations

Exercise 3.1

Buffer Management Basics

Exercise 3.1
Buffer Management Basics

a) Discuss the ***main principles of Buffer Management*** in a DBMS.

b) Explain the three operations of the buffer manager interface - ***FixPage(x), UnfixPage(x), FlushPage(x).*** What is the difference between ***physical and logical page references***? Under what circumstances are database buffer pages flushed?

c) Describe how techniques such as ***"blocked I/O", "double buffering"*** and ***"page prefetching"*** can be used to boost performance.

Exercise 3.1
Buffer Management Basics

a) Discuss the main principles of Buffer Management in a DBMS.

- Provide infinitely ***large logical address space***
- ***Make pages available***
  (if page not in buffer ➔load page)
- Fix / Unfix pages
- Implement replacement strategies

  a.k.a. Eviction strategies
- Allocate space to transactions, etc…

Exercise 3.1
Buffer Management Basics

DBMS vs. OS Buffer Management

- OS doesn’t offer quite the right services (e.g. replacement strategies)
- DB-prefetch is more efficient (it knows its data)
- *Different Operation Systems*: Portability issues
- *Limitations of OS BM*: files can’t span disks.

Exercise 3.1
Buffer Management Basics

Size of buffer and availability of pages:

- **In general:**
  Buffer of a DB is ***much smaller*** than the DB
- Gap between main memory and external memory access
- **Example:**
  DB-Size = 100 * buffer size
  ➔ Random request for page references
  ➔ 1 % of all requests can be fulfilled without ***page fault***
- ***Reference locality***
  - ***lokal vs. temporal***

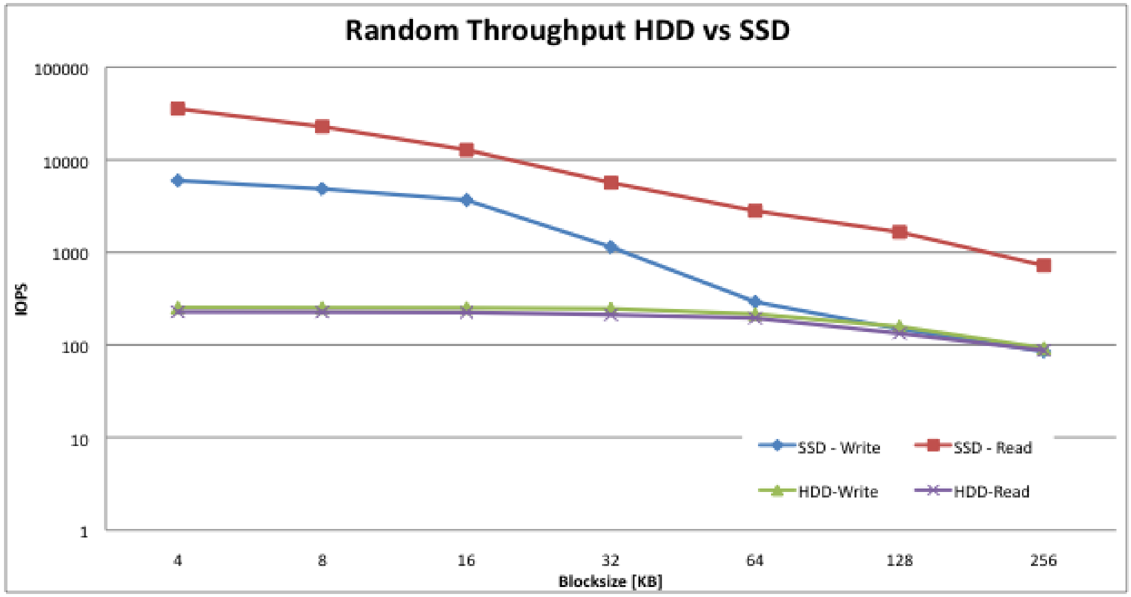
Therefore:

- Size of buffer is important
- Need for **good allocation and replacement strategies**

Exercise 3.1
Buffer Management Basics

FLASH - Size of buffer and availability of pages:

- Random Reads much faster than random writes
- 4K Random Accesses much faster than 16K (more IOPS)
- Implications on Buffersize?
- Eviction Stratgies? (Dirty Pages vs. Non Dirty Pages?)



Exercise 3.1
Buffer Management Basics

b) ***Explain the three operations*** of the buffer manager interface:

- FixPage(*x*)
- UnfixPage(*x*)
- FlushPage(*x*)

Under what circumstances are *database buffer pages flushed?*

Exercise 3.1
Buffer Management Basics

FixPage(*x*) (PinPage(x)) = Logical Page Reference

Transaction wants to access a page ‘*x’* for reading / writing

➔ Sends **FixPage(*x*)** ***(PinPage(x))*** command to BM.

1. If *'x' not in buffer* (**page fault**)
   ➔ *load it* from the stable DB.
   (Called **fetching**, result: **physical page reference)**
   If *buffer is full*
   ➔ A page needs to be selected for replacement
   ➔ buffer's replacement strategy.

2. Increment **pin-count** of 'x' and pass address to caller.

Exercise 3.1
Buffer Management Basics

UnfixPage(*x*) (UnpinPage(*x*)):

- decrement pin-count of '*x*'

FlushPage(x):

- copy '*x*' from the DB buffer to the stable DB
  ➔ synch. stable page '*x*' with cached page '*x*'.

Exercise 3.1
Buffer Management Basics

When are pages flushed:

- **At page fault:**
  Page selected for replacement (victim) is ***dirty***
- **Asynchronously:**
  By a background process flushing pages as needed to make sure there is always ***a certain amount of clean pages*** (free or unmodified) ***available***
- **At transaction commit**:
  Flushing can be ***forced*** by the recovery manager in order to enforce WAL (in difference to ***no force***)

Exercise 3.1
Buffer Management Basics

c) Describe how techniques such as ***"blocked I/O", "double buffering" and "page prefetching"*** can be used to boost performance.

Blocked I/O:

Issuing a single request to read (or write) a **block of several consecutive pages** can be much cheaper than reading (or writing) the same pages through independent I/O requests.
(Cost of Ttrans + Tpos + Trot)
- What about a SSD with 35.000 Read IOPS per 4k Page?

Note:
In the following by block we mean a group of several consecutive DB pages.

Exercise 3.1
Buffer Management Basics

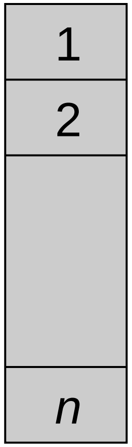
Double Buffering:

- **Goal:**
  Keep ***the CPU busy*** while an I/O request is being processed - to *overlap CPU and I/O processing*.
- ***Split the buffers in two parts:***
  CPU can start processing a block once its transfer to MM is completed. Parallel: the disk I/O processor can be reading and transferring the next block into the second part.
- If ***(CPU process time p. block < transfer time from disk p. block)***
  ➔ ***seek times & rotational delays are eliminated*** for all, but the first block transfer!
- May ***reduce response times for individual queries***,
  !!! but ***may not have a significant impact on throughput***. !!!

Exercise 3.1
Buffer Management Basics

Example Double Buffering:

Buffer size = *n* pages



Without Double Buffering:

1. Transfer *n* pages into buffer
2. Process *n* pages

With Double Buffering:

1. Transfer *n*/2 pages into buffer
2. Process *n/2* pages **and** transfer next block(n/2 pages) into buffer

Exercise 3.1
Buffer Management Basics

Page Prefetching:

Often the BM can *anticipate* the next several page requests and fetch the ***corresponding pages into memory before*** they are ***requested***.

Benefits:

- ***Pages are available*** in buffer when requested
- ***Blocked I/O can be exploited***
  Fetching the pages with a single I/O request can lead to faster I/O
  (Order of retrieval can be chosen that min. seek times & rotational delays)
- Can be negative on SSDs (Mixed Workload → r/w - seq./rand.)
  - Pages are read unnecessary → unused effort → SSDs On Disk Cache polluted

Exercise 3.2

TECHNISCHE UNIVERSITÄT DARMSTADT

Allocation and Replacement Strategies. Effect of Virtual Memory.

Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory

a) Explain the difference between

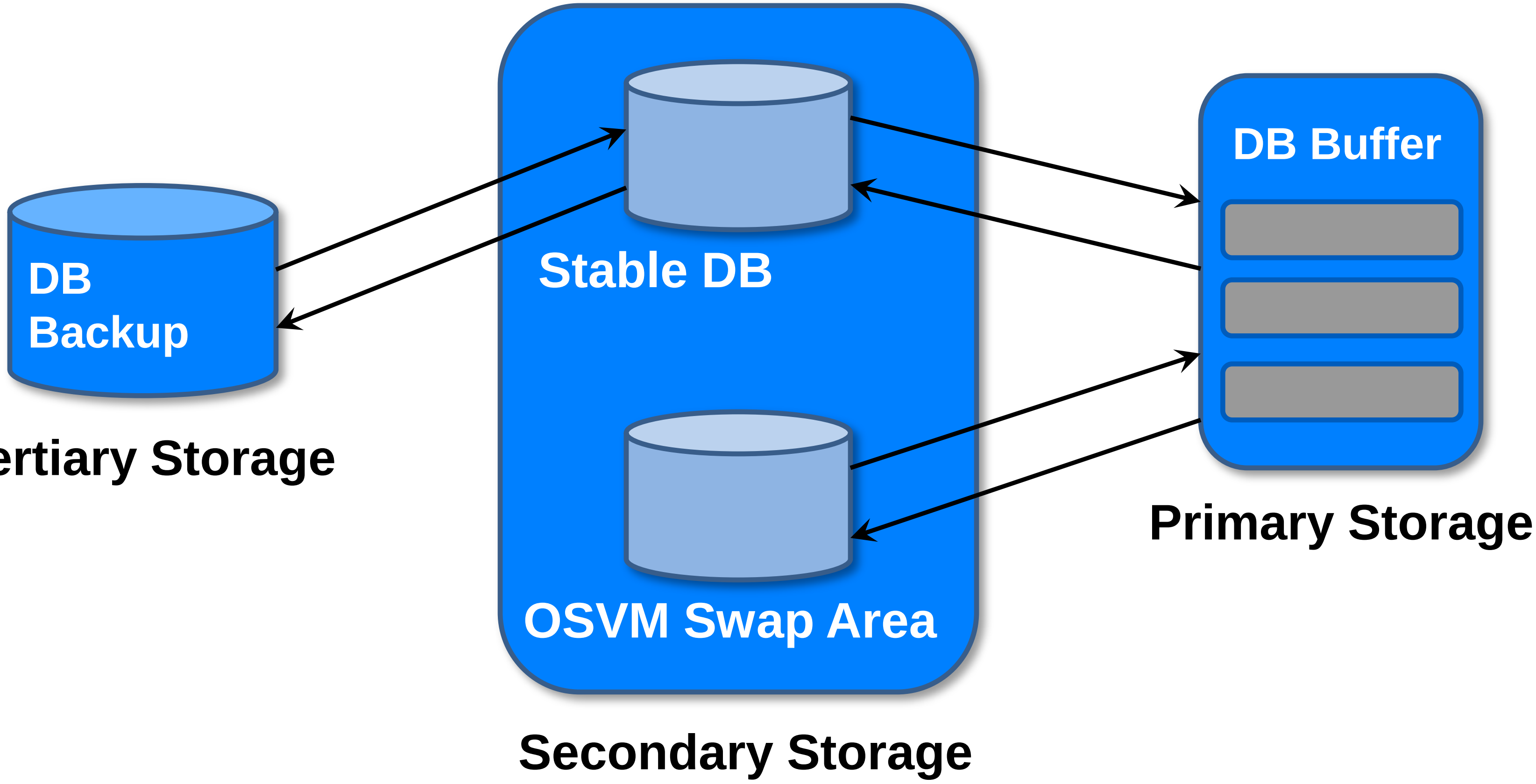
global, local and page-type specific allocation strategies.

How are allocation strategies related to replacement strategies?

b) Under what circumstances are database pages copied (transferred) among the different storage locations in the diagram (next slide)?

The database buffer generally resides in the virtual memory of the OS. This could lead to a situation referred to as a **double page fault**. How many I/O accesses would in this case be required for fetching a page in the buffer?
What can be done to avoid double page faults?

Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory



Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory

a) Explain the difference between

global, local and page-type specific allocation strategies.

How are allocation strategies related to replacement strategies?

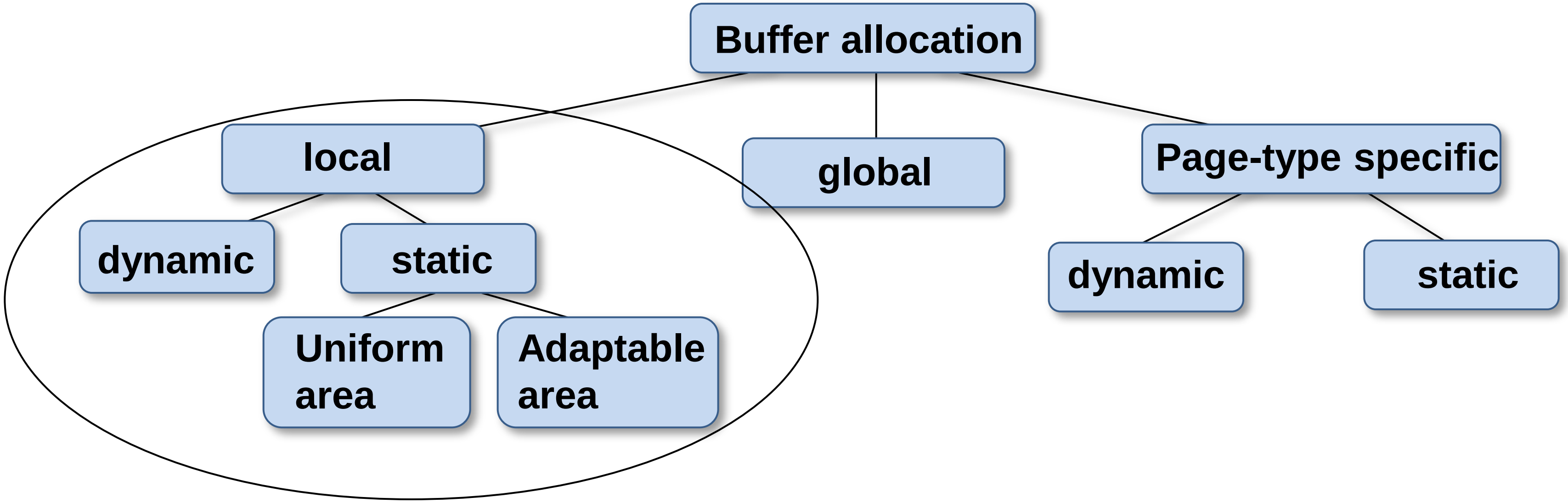
Allocation strategies:

- determine how buffer space is allocated for use by running processes / transactions.

Replacement strategies:

- determine how allocated buffer space is managed
  Which page to replace when a buffer is full and a new page must be fetched?

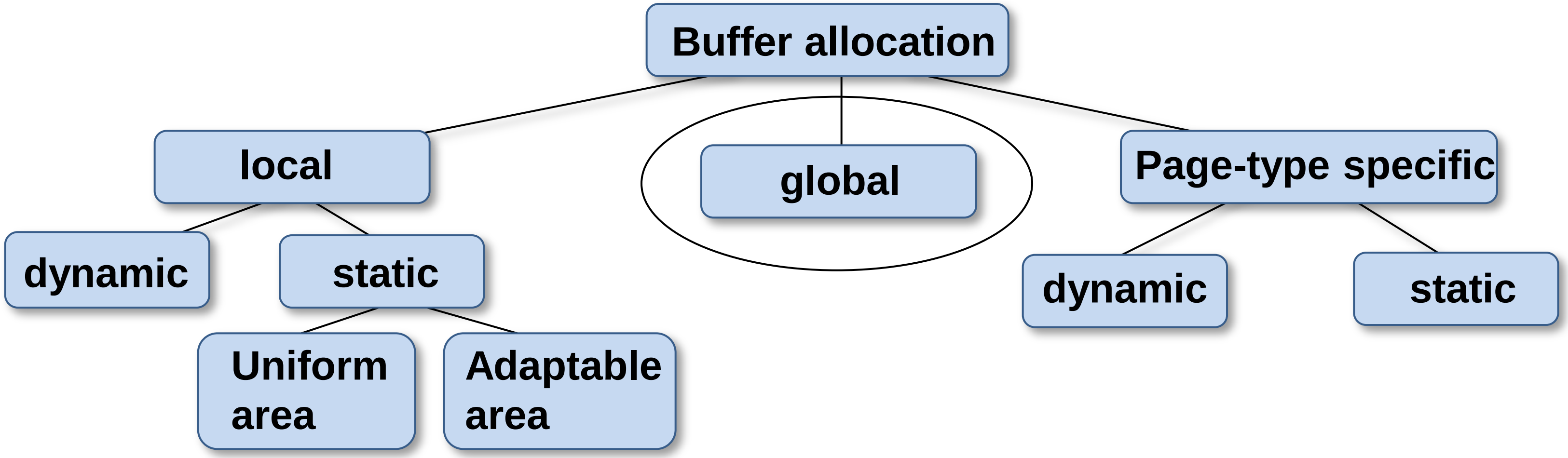
Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory



Local (static vs. dynamic):

- buffer space allocated on a ***per-transaction basis***
  - **static** ⇔ transaction's buffer *space fixed*
  - **dynamic** ⇔ transaction's buffer *space dynamic*
- transaction's buffer space managed independently
- *replacement* strategy *based on transaction's individual* reference behavior

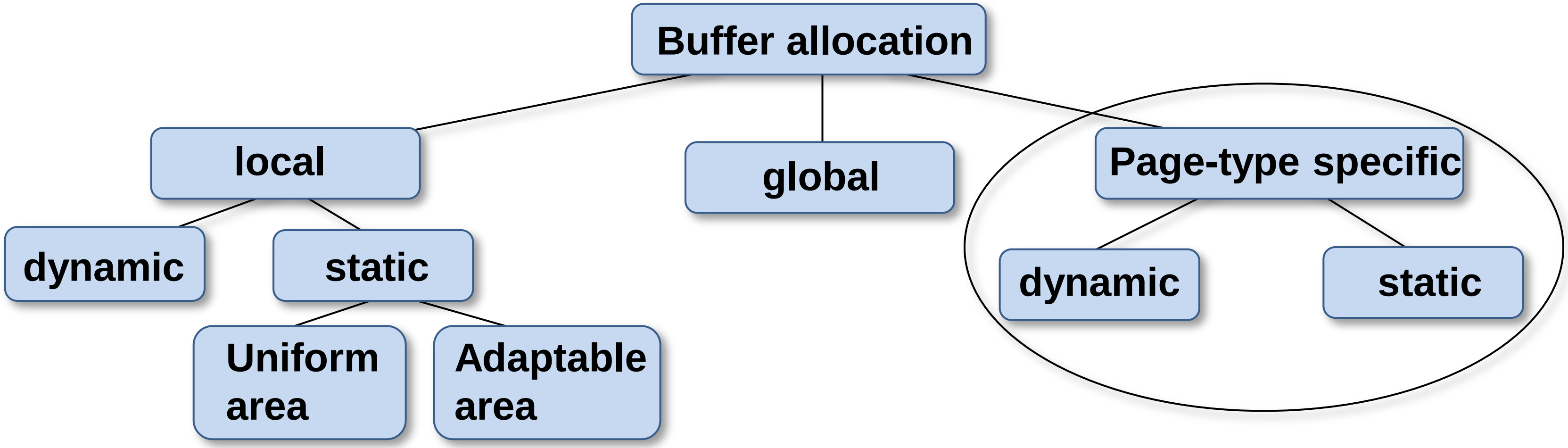
Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory



Global:

- all transactions share the same buffer space - ***one global buffer***
- buffer space is managed globally
  - *replacement* strategy *based on reference behavior of* ***all*** *transactions* no special attention paid to reference behavior (and needs) of individual transactions

Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory



Page-Type Specific / Partition-Oriented (static vs. dynamic):

- buffer partitioned into areas (of static/dynamic size) that are **exclusively used for different types of data** (e.g. user data, access path data, system data, catalog data, log data etc..).
- partitions *managed independently*

Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory

Static allocation strategies do not allow dynamic load balancing (tx's length and load are not taken into account)

➔ **Static allocation strategies are:**

- **a) extremely inefficient** and
- b) impractical.

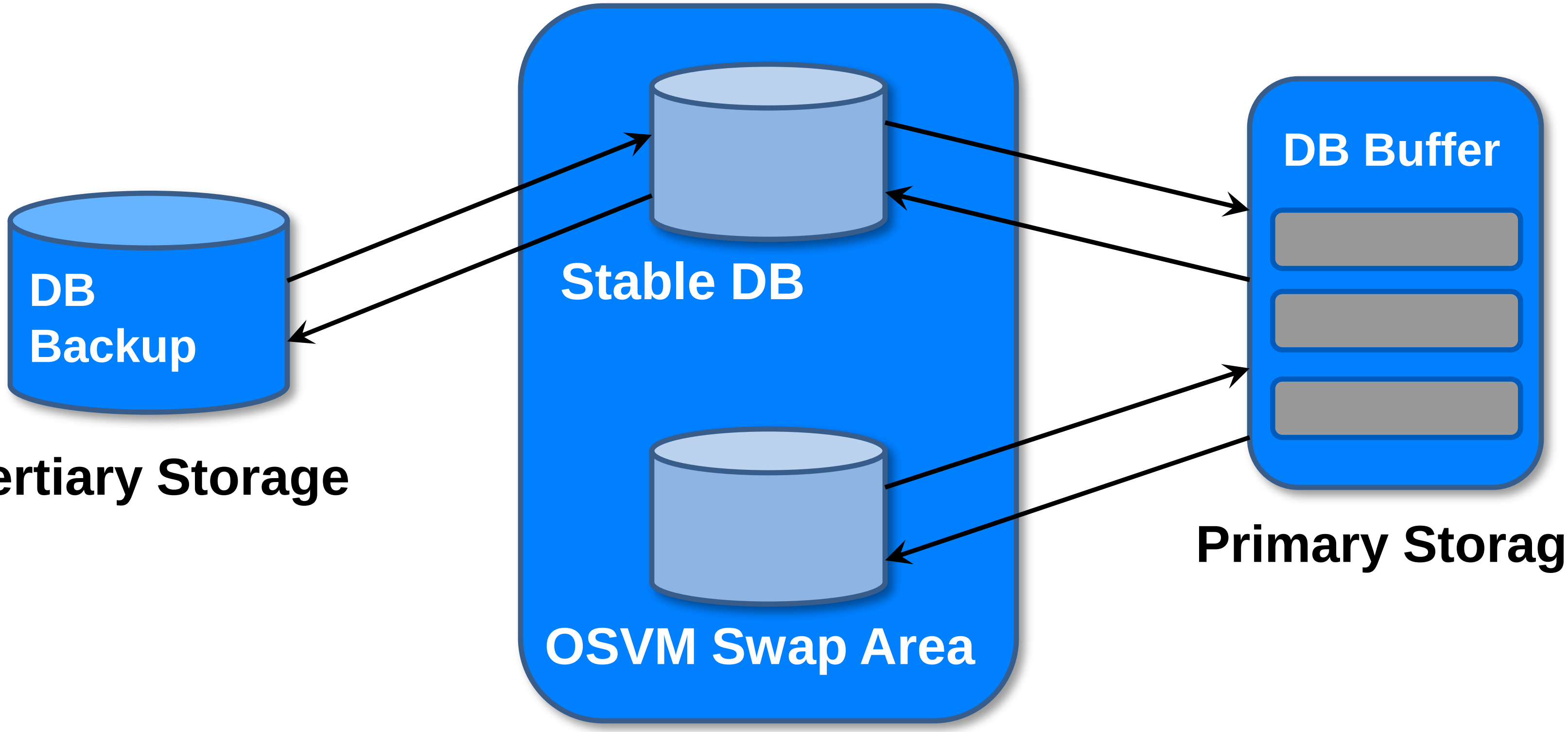
In practice **dynamic local/partition-oriented strategies** have proven to **be most effective**.

Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory

b) Under what circumstances are database pages copied (transferred) among the different storage locations in the diagram (next slide)?

The database buffer **generally resides** in the virtual memory of the OS.
This could lead to a situation referred to as a **double page fault**.
How many I/O accesses would in this case be required for fetching a page in the buffer?
What can be done to avoid double page faults?

Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory



Tertiary Storage

Secondary Storage

Primary Storage

Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory

Database Page Fault:
FixPage(x) operation referring to a page that is not in the database buffer.

VMM Page Fault:
Trying to access a virtual memory page, that has been swapped out by the VMM.

➔Double Page Fault =
VMM Page Fault + Database Page Fault

Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory

A logical page reference could lead to a DB Page Fault, which in turn could lead to a VMM Page Fault
(buffer is full and selected page for replacement is both dirty and paged out of MM):

- "page in" dirty buffer page to be replaced - OSVMM
- **flush page** to disk - BM
- **load requested DB page** in freed buffer frame – BM

➔In the worst case 3 I/Os!

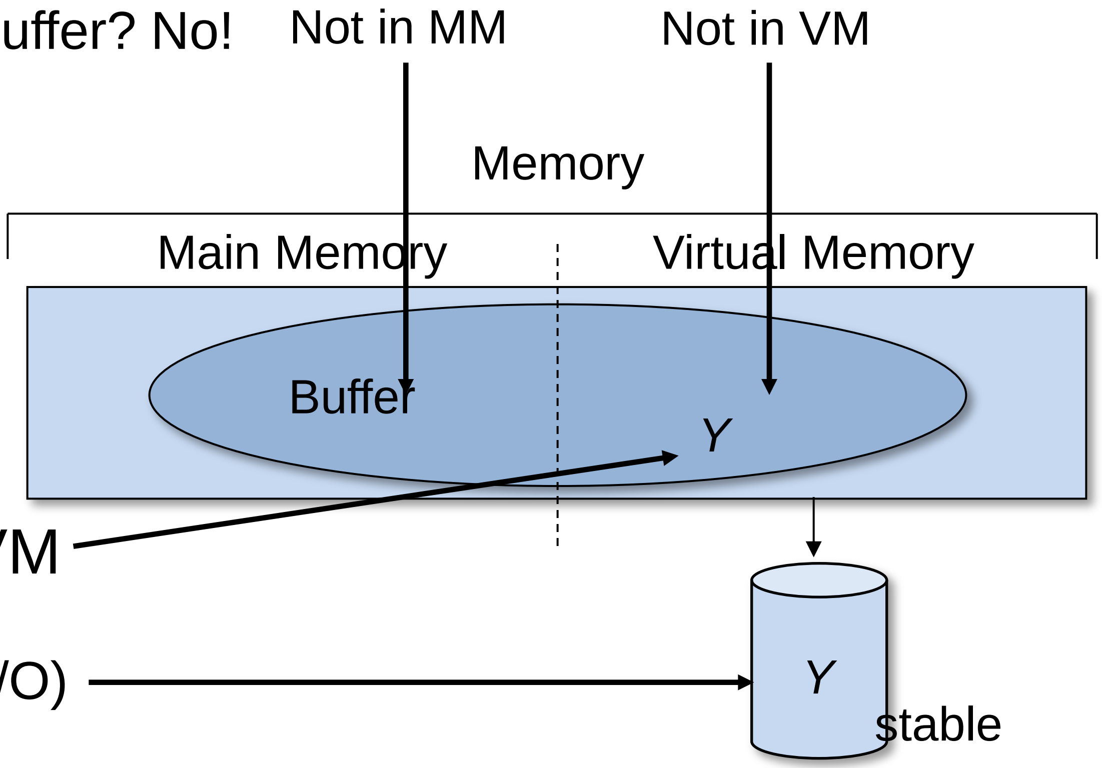
Exercise 3.2
Allocation and Replacement Strategies.
Effect of Virtual Memory

1. FixPage(*X*) ➔Is page in Buffer? No!

2. Buffer is full
   - ➔Needs to replace Page
   - ➔Select page *Y*

3. Page *Y* is dirty and in VM
   - ➔Read *Y* (1. I/O)
   - ➔Write *Y* to stable DB (2. I/O)

4. Transfer Page *X* into buffer
   - ➔Read *X* from stable DB (3. I/O)



VMM Page Fault and Database Page Fault

➔Double Page Fault

Exercise 3.2
Allocation and Replacement Strategies. Effect of Virtual Memory

To avoid double page faults:

- Flush pages asynchronously
- Do not allow dirty pages to be paged out of RAM by OS

Exercise 3.3

Working−Set Allocation Strategy, LRU Replacement Strategy

Exercise 3.3
Working−Set Allocation Strategy, LRU Replacement Strategy

Given is the following logical page reference string:

"T2:L T1:A T1:B T1:C T2:K T1:D T1:E T2:L T1:F T1:G T1:H T2:K T1:I“

"T*i*:*X*" denotes a logical reference to page *X* by transaction T*i*. The size of the **database buffer pool is 4 pages** and they are initially free.

a) Show how the buffer space would be managed during the execution of the above references if **global allocation with LRU replacement** strategy is employed.

b) Show how the buffer space would be managed under the **Working−Set Method**, with a window size $\tau = 2$.
Describe how the auxiliary structures W(T*i*, t), TRC(T*i*) and LRC(T*i*, *X*) for *i*=1..2 are used assuming that initially TRC(T*i*)=0.

Exercise 3.3 Working−Set Allocation Strategy, LRU Replacement Strategy

LRU (Least Recently Used)

- Considers ***age*** and ***references***
- Replaces page that hasn‘t been ***referenced the longest***
- Two possibilities of interpretation of the word “used“
  - ***Least Recently Referenced***
  - ***Least Recently Unfixed***

Which Structure is to be used?

Exercise 3.3
Working−Set Allocation Strategy, LRU Replacement Strategy

a)

t	T1	T2	LRU queue	Page(out→in)
1		L	L	→L
2	A		L, **A**	→A
3	B		L, A, **B**	→B
4	C		L, A, B, **C**	→C
5		K	A, B, C, **K**	**L**→K
6	D		B, C, K, **D**	A→D
7	E		C, K, D, **E**	B→E
8		L	K, D, E, **L**	C→**L**
9	F		D, E, L, **F**	**K**→F
10	G		E, L, F, **G**	D→G
11	H		L, F, G, **H**	E→H
12		K	F, G, H, **K**	L→**K**
13	I		G, H, K, **I**	F→I

L

K

***Evaluation**:*

- •T1 has 9 page faults
- •T2 has 4 page faults

T2 reference locality:

4 references / 2 pages

➔ T1 ‘overtakes’ T2

➔ T2’s reference locality is overlooked!

Exercise 3.3
Working−Set Allocation Strategy, LRU Replacement Strategy

b) Show how the buffer space would be managed under the Working−Set Method, with a window size τ ***= 2.***
Describe how the auxiliary structures ***W(Ti, t)***, ***TRC(Ti)*** and ***LRC(Ti, X)*** for *i*=1..2 are used assuming that initially ***TRC(Ti)=0***.

For each active Transaction *Ti* keep a counter: *TRC(Ti)*
Each page *k* in the buffer has a reference counter *LRC(Ti,k)* for each transaction that has been active on it
Whenever *Ti* references page *k*,
□Counter *TRC(Ti)* is incremented and stored in *LRC(Ti,k)*

Exercise 3.3 Working−Set Allocation Strategy, LRU Replacement Strategy

Working Set **W(t,τ):**

set of pages that was referenced by a transaction within its ***last τ references*** measured at time t

τ size of window given as # ***of references***

w(t,τ) = |W(t,τ)|

size of working set at time t

The average size of the ***WS*** of a transaction indicates its ***reference locality.***

Exercise 3.3
Working−Set Allocation Strategy, LRU Replacement Strategy

Principle of operation:

- pages in the working set of a transaction should be kept in buffer
- pages that are no longer in the WS can be replaced
- when locality is high, WS shrinks

Important:

- window size τ must be specified
- since working set method doesn‘t distinguish among the elements in the WS, an additional replacement strategy must be defined

Refinement (priorities):

- use τ of different sizes based on transaction classes

Exercise 3.3
Working−Set Allocation Strategy, LRU Replacement Strategy

TRC(T*i*)

One counter per Transaction T*i*:

LRC(T*i*,*k*):

Each page *k* in the buffer has a reference counter for for each transaction that has been active on it

Whenever T*i* references page *k*, the counter **TRC(T*i*)** is incremented and stored in **LRC(Ti,k)**

A page may be substituted if for all active transactions:

$$\mathbf{TRC(Ti) - LRC(Ti,k)} \geq \tau$$

Exercise 3.3
Working−Set Allocation Strategy, LRU Replacement Strategy

Buffer size: 4

t	T1	TRC T1	LRC(T1,[])	T1 W(t,2)	T2	TRC T2	T2 W(t,2)	LRC (T2, [])	Buffer
0		0				0			
1		0			L	**1**	**L**	**[L]=1**	**L**
2	A	**1**	**[A]=1**	**A**					L,**A**
3	B	**2**	[A]=1, **[B]=2**	A, **B**					L,A,**B**
4	C	**3**	[B]=2, **[C]=3**	B, **C**					L,A,B,**C**
5					K	**2**	L, **K**	L=[1], **K=[2]**	L,**K**,B,C
6	D	**4**	[C]=3, **[D]=4**	C, **D**					L,K,**D**,C
7	E	**5**	[D]=4, **[E]=5**	D, **E**					L,K,D,**E**
8					L	**3**	L, K	**L=[3]**, K=[2]	L,K,D,E
9	F	**6**	[E]=5, **[F]=6**	E, **F**					L,K,**F**,E
10	G	**7**	[F]=6, **[G]=7**	F, **G**					L,K,F,**G**
11	H	**8**	[G]=7,**[H]=8**	G, **H**					L,K,**H**,G
12					K	**4**	L, K	L=[3], **K=[4]**	L,K,H,G
13	I	9	[H]=8, **[I]=9**	H **I**					L,K,H,**I**

Exercise 3.3
Working−Set Allocati t
Strategy

Buffer size: 4

TRC(1) - LRC (1,B) =3-2<τ
TRC(1) - LRC (1,C) =3-3<τ
TRC(2) - LRC (2,L) =2-1<τ
TRC(1) - LRC (1,A) =3-1=τ
➔Replace A

t	T1	TRC T1	LR					LRC (T2, [])	Buffer
0		0							
1		0					L	**[L]=1**	**L**
2	A	**1**	**[A]=1**	**A**					L,**A**
3	B	**2**	[A]=1, **[B]=2**	A, **B**					L,A,**B**
4	C	**3**	[B]=2, **[C]=3**	B, **C**					L,A,B,**C**
5					K	**2**	L, **K**	L=[1], **K=[2]**	L,**K**,B,C
6	D	**4**	[C]=3, **[D]=4**	C, **D**					L,K,**D**,C
7	E	**5**							L,K,D,**E**
8							L, K	**L=[3]**, K=[2]	L,K,D,E
9	F	**6**							L,K,**F**,E
10	G	**7**	[F]=6,						L,K,F,**G**
11	H	**8**	[G]=7,**[H]=8**	G, **H**					L,K,**H**,G
12					K	**4**	L, K	L=[3], **K=[4]**	L,K,H,G
13	I	9	[H]=8, **[I]=9**	H **I**					L,K,H,**I**

T1 references A and TRC(T1)=1
➔LCR(T1,A)=1

Exercise 3.3
Working−Set Allocation Strategy, LRU Replacement Strategy

Evaluation:

- T1 has 9 page faults, since it has *no reference locality*
- T2 has 2 page faults – **minimum**

➔ Working Set Method - good for Intra-Tx locality

Exercise 3.4
Appendix: BM in Practice

Buffer Management in Practice

Exercise 3.4
Buffer Management in Practice

Discuss what kind of buffer management techniques are implemented in the major commercial database systems

Exercise 3.4
Buffer Management in Practice

- **IBM DB2** and **Sybase ASE** allow buffers to be partitioned into *named pools*.
  Each database, table or index can be bound to one of the pools.
  Each pool can be configured to use either *LRU* or *clock replacement* in ASE.
  DB2 uses a variant of clock replacement, with the initial clock value based on the nature of the page (e.g. index non-leaves get a higher starting clock value, which delays their replacement).
  DB2 V6 also supports FIFO.

Exercise 3.4
Buffer Management in Practice

- **Informix** and **Oracle 7** both maintain a single global buffer pool using *LRU*.
- In **Oracle 8** tables can be bound to one of two pools - one has high priority and the system attempts to keep pages in this pool in memory.
- **Microsoft SQL Server** has a single pool using *clock* replacement.

Exercise 3.4
Buffer Management in Practice

Beyond setting a maximum number of pins for a given transaction, there are typically no features for controlling buffer pool usage on a per-transaction basis.

MS SQL Server, however, supports a reservation of buffer pages by queries that require large amounts of memory (e.g. queries involving sorting or hashing).

Exercise 3.4
Buffer Management in Practice

Prefetching:

- In **DB2** both sequential and list prefetch (prefetching a list of pages) are supported. In general the prefetch size is 32 4 KB pages, but this is configurable. For some sequential type database utilities (e.g. COPY, RUNSTATS), DB2 will prefetch up to 64 4KB pages.
- **Oracle 8** uses prefetching for sequential scan, retrieving large objects, and for certain index scans.
- **Microsoft SQL Server** supports prefetching for sequential scan and for scans along the leaf level of a B+tree index and the prefetch size can be adjusted as a scan progresses.
- **Informix** supports prefetching with a user-defined prefetch size.

Appendix: Queues

LRU queues

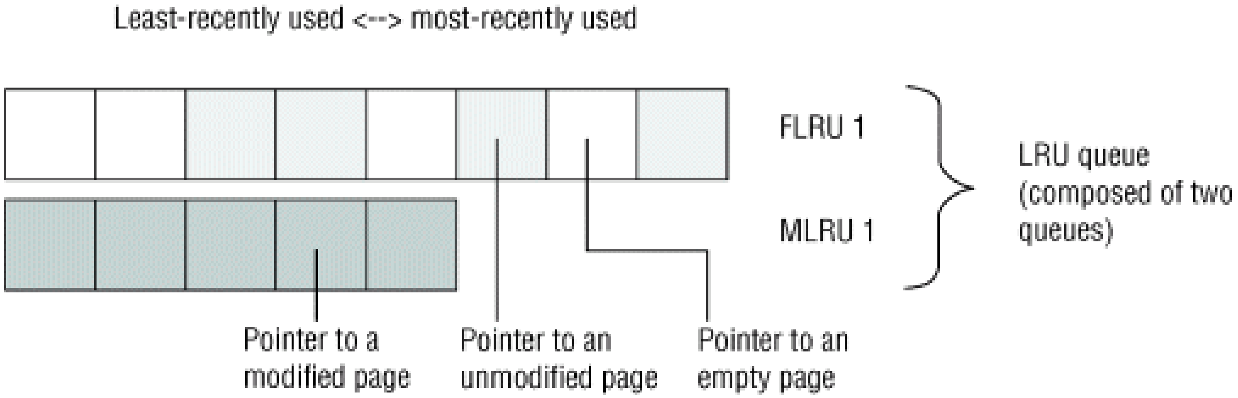
Based on: IBM Informix Dynamic Server Administrator's Guide

http://publib.boulder.ibm.com/infocenter/ids9help/index.jsp?topic=/com.ibm.admin.doc/admin290.htm

LRU-Queues

- A database server uses 1 to *n* LRU Queues to replace cached data held in buffer.
- **Each LRU queue** is composed of a pair of *linked lists:*
  - **FLRU** (free least-recently used) list:
    tracks free or unmodified pages in the queue

  - **MLRU** (modified least-recently used) list:
    tracks modified pages in the queue


Least-recently used <--> most-recently used
FLRU 1
MLRU 1
LRU queue (composed of two queues)
Pointer to a modified page
Pointer to an unmodified page
Pointer to an empty page

LRU-Queues

- Pages are maintained in ***least-recently*** to ***most-recently used order*** in the queues.

- *Each buffer* (frame) has *one entry* in one of the buffer queues

- All LRU queues on one server have same size:
  Size of LRU queue = Buffers / #LRU queues

Example:

Buffers: 10000 pages #LRU queues 10

➔1000 buffers per LRU queue

LRU-Queues

Processing

Init State:

All page buffers are empty.
Every buffer is represented by an entry in a FLRU queue.

User thread needs to acquire a page:

1. Select ***randomly*** one FLRU queue

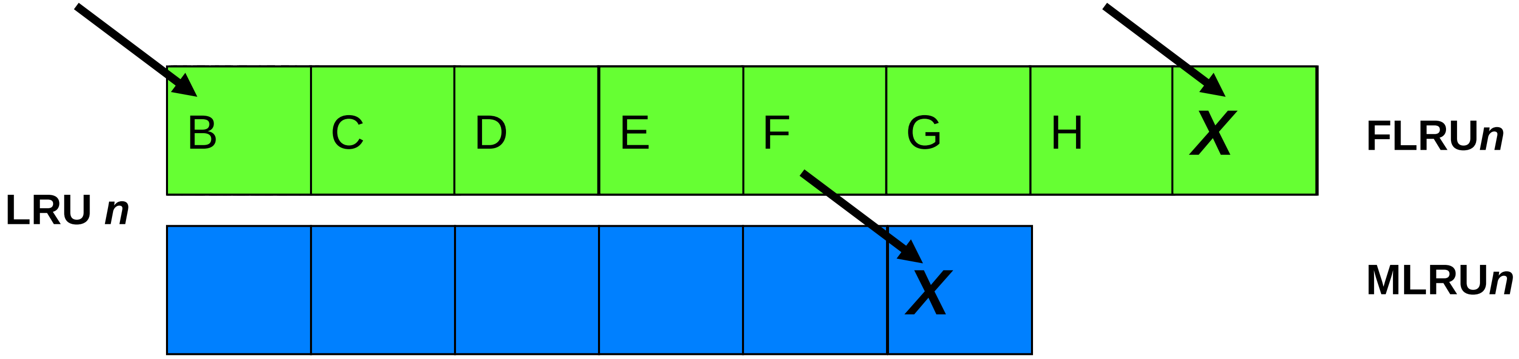
2. Try to ***latch least-recently used (or oldest) entry***
   If *not possible* (queue is locked)➔select another FLRU queue

3. User thread finishes and releases buffer:
   Page was modified: Place page at the *most-recently used end* of an **MLRU queue**.
   Page not modified: Place page at the *most-recently used end* of an **FLRU queue.**

LRU-Queues

1. Client requests page *X.*

2. Server selects randomly a FLRU queue.
3. Latch least recently used buffer (A)
4. Remove A and load *X*
5. Client release buffer:
    1. Buffer was not modified
    2. Buffer was modified



LRU Queues

Why multiple LRU queues?

- ***Reduce user-thread contention*** for the queues.
- Allow ***multiple cleaners*** to flush dirty pages and maintain the percentage of ***dirty pages at an acceptable level.***

Note:

Number of queues is recommended based on numbers of CPUs (VPs).

LRU Queues

Cleaning LRU Queues

- LRU queues are cleaned by ***Cleaner Threads***
- Parameters:
  LRU_MAX_DIRTY:
  Max percentage of dirty pages in LRU queue.
  LRU_MIN_DIRTY:
  Acceptable percentage of dirty pages.
- LRU writes are performed **as *background writes*** by cleaner threads.
- LRU writes may also be triggered by foreground writes who alert the cleaner thread of the write
- Cleaning LRU queues in background avoids having to wait for a clean page when a transaction needs it
- Number of cleaning threads depends on number of disks (one per frequently updated disk) as well as length of queues and parameters.

LRU Queues

Example:

LRU_MAX_DIRTY:	60 %
LRU_MIN_DIRTY:	40 %
Buffers	10000

Page cleaning begins when 6000 buffers are in the MLRU queue.

Page cleaning must continue until 4000 buffers are left in the MLRU queue (acceptable number of dirty pages) and can continue beyond this point.