Database Architecture

Including but not limited to…

Matematikos ir informatikos fakultetas

Arūnas Janeliūnas
Object Databases

Data saving

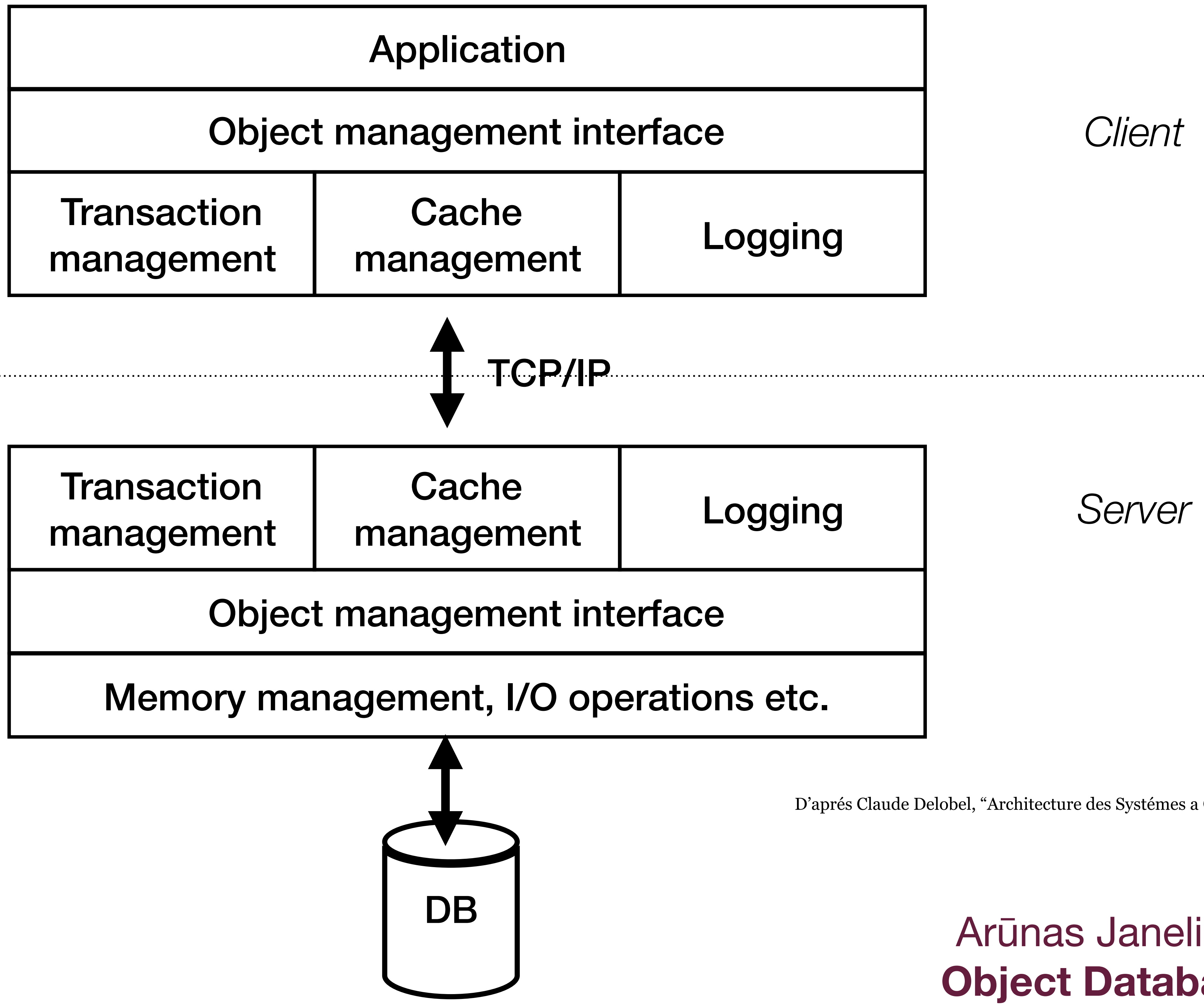
- ODB is an „extension“ to an Object Oriented Programming Language, providing it with data preserving capabilities.
- Then some objects in the program are of „temporal“ nature (to be dismissed after program ends) and some are to be preserved.
- How to know which object is which?

Data saving

3 data saving models:

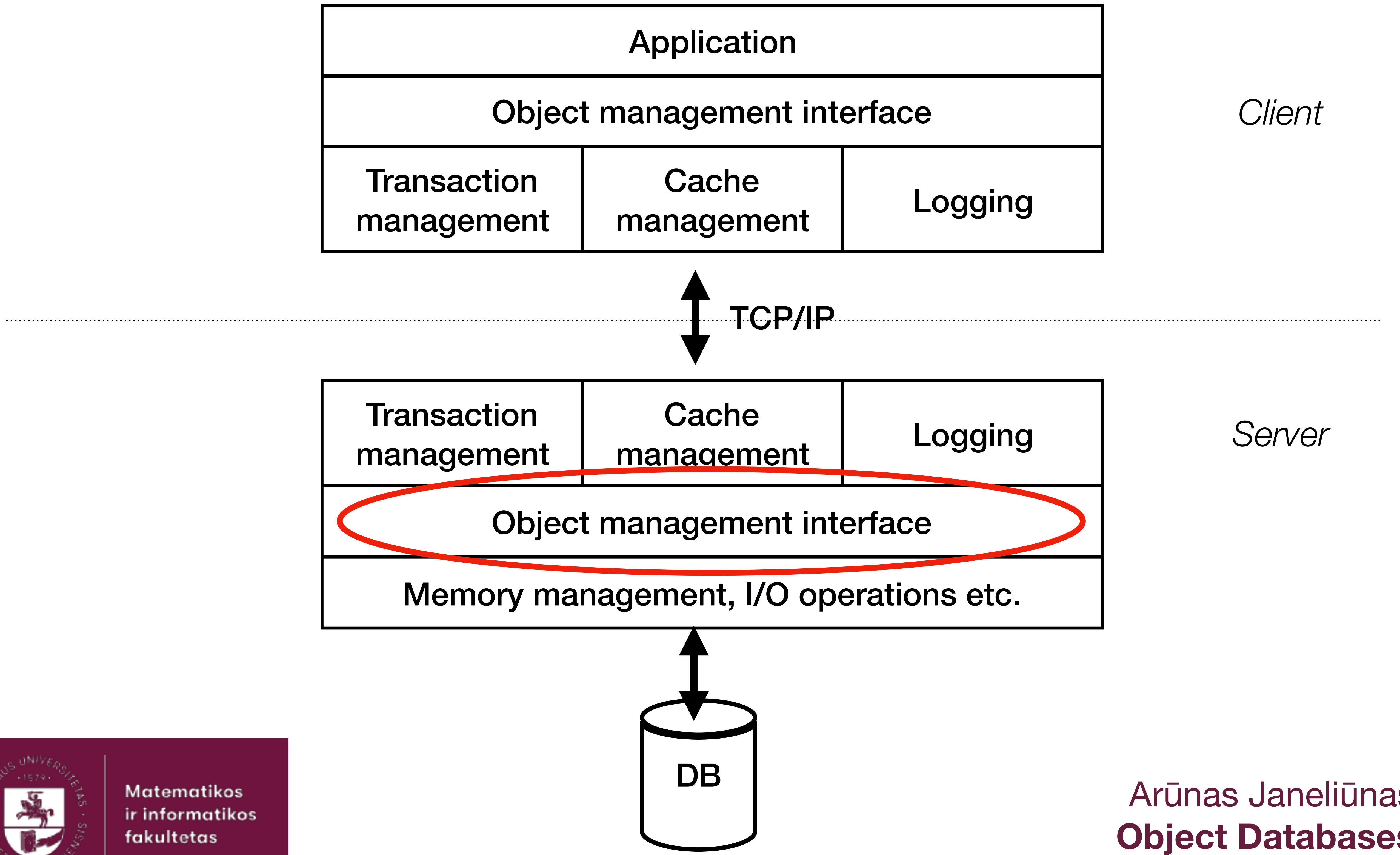
- **Persistent classes.** Some classes are declared to be persistent and every object of that class persists.
- **Persistent *new*.** Objects that are to be stored in the database are created with specific *persistent new* operator.
- **Persistence by accessibility.** We call a „save“ method to the object and then every other object accessible by associations to that object is stored as well automatically.

Client-Server architecture



D'aprés Claude Delobel, "Architecture des Systémes a Objets", 1997

Server knowledge levels



Server knowledge levels

Low knowledge level

Server side knows only the size of an object and it’s ID. It regards objects just as identifiable byte arrays.

PROS

- Easily built
- May be applied to various data models

CONS

- Data interpretation may be done only on Client
- Data navigation (and associative access models) cannot be done on server

Server knowledge levels

Medium knowledge level

Server side knows objects structure, but still cannot execute methods on server side.

PROS

- Data navigation (and associative access models) can be done on server
- Query predicates can be evaluated on server
- Query optimisation is possible

CONS

- Query predicates and other sub-queries involving methods can be calculated only on client-side

Server knowledge levels

High knowledge level

Server side knows everything about objects structure and can execute.

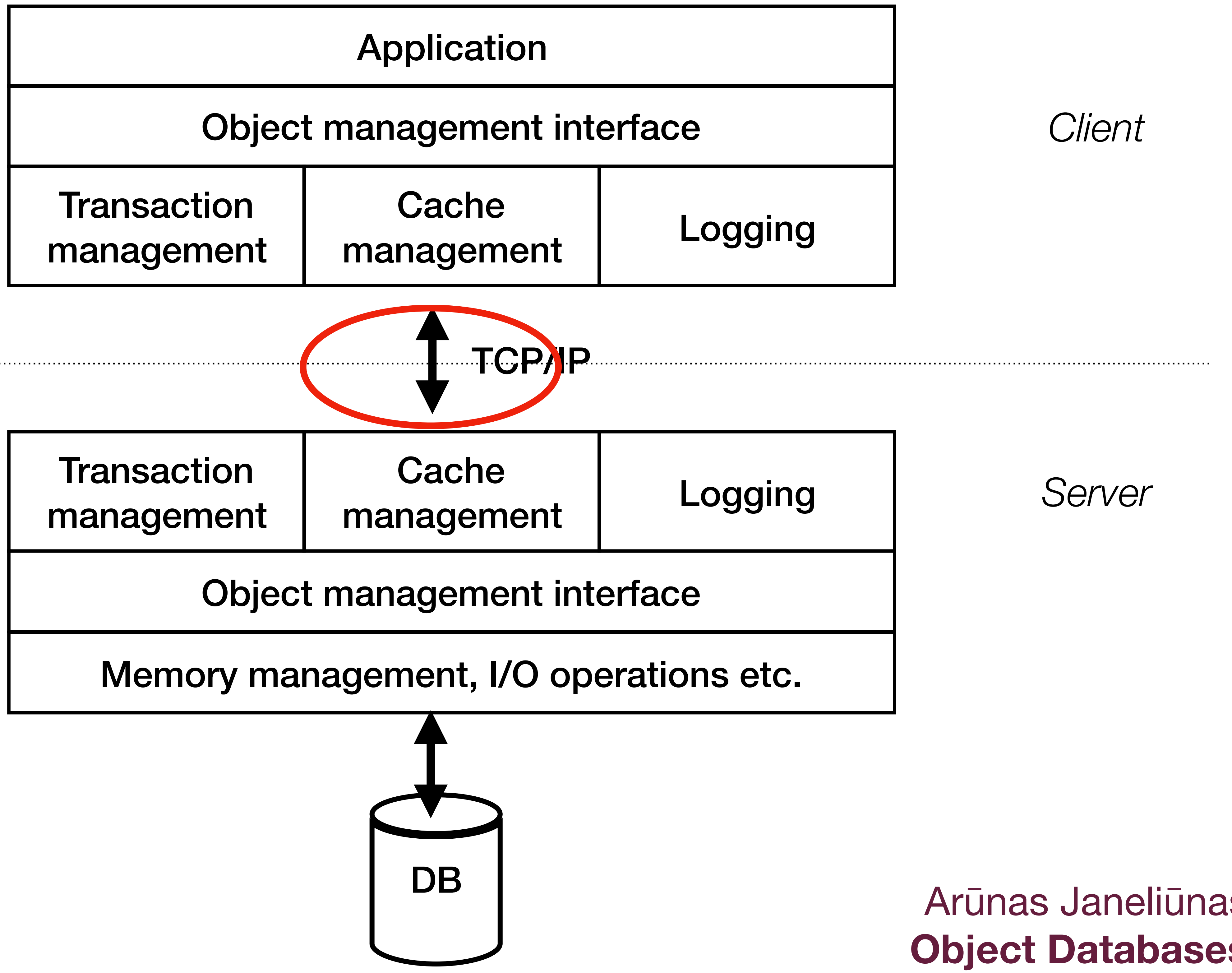
PROS

- Maximum server capabilities
- Possible detection of commutative operations (concurrency control)
- you name it…

CONS

- Hard to implement
- Server „weight“

Server output

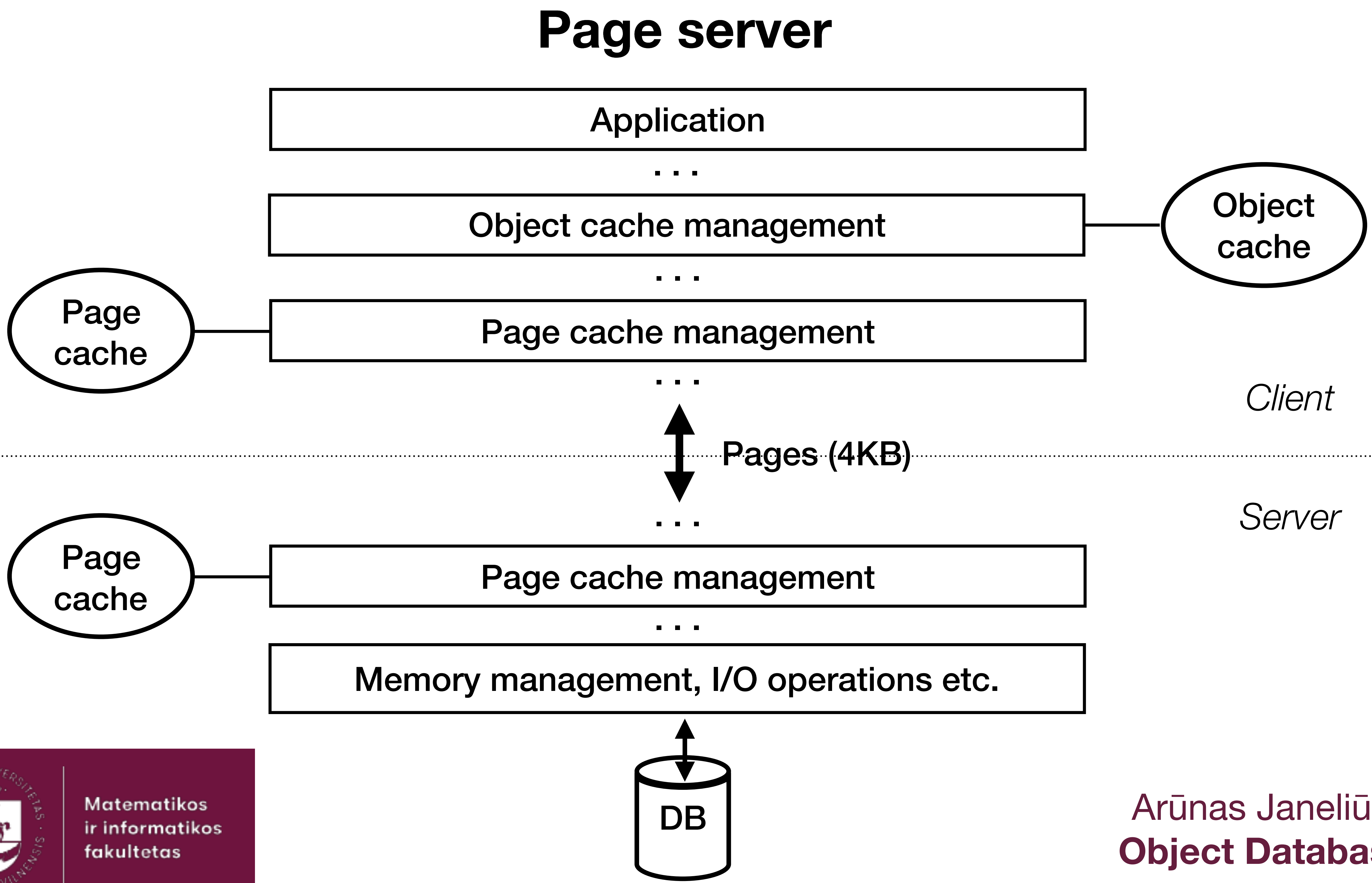


Server output

Page server

Server sends out pages (4KB).

Server output



Server output

Page server

Server sends out pages (4KB).

PROS

- Easier server architecture
- Easier communication
- Good usage of objects grouping and associative access techniques

CONS

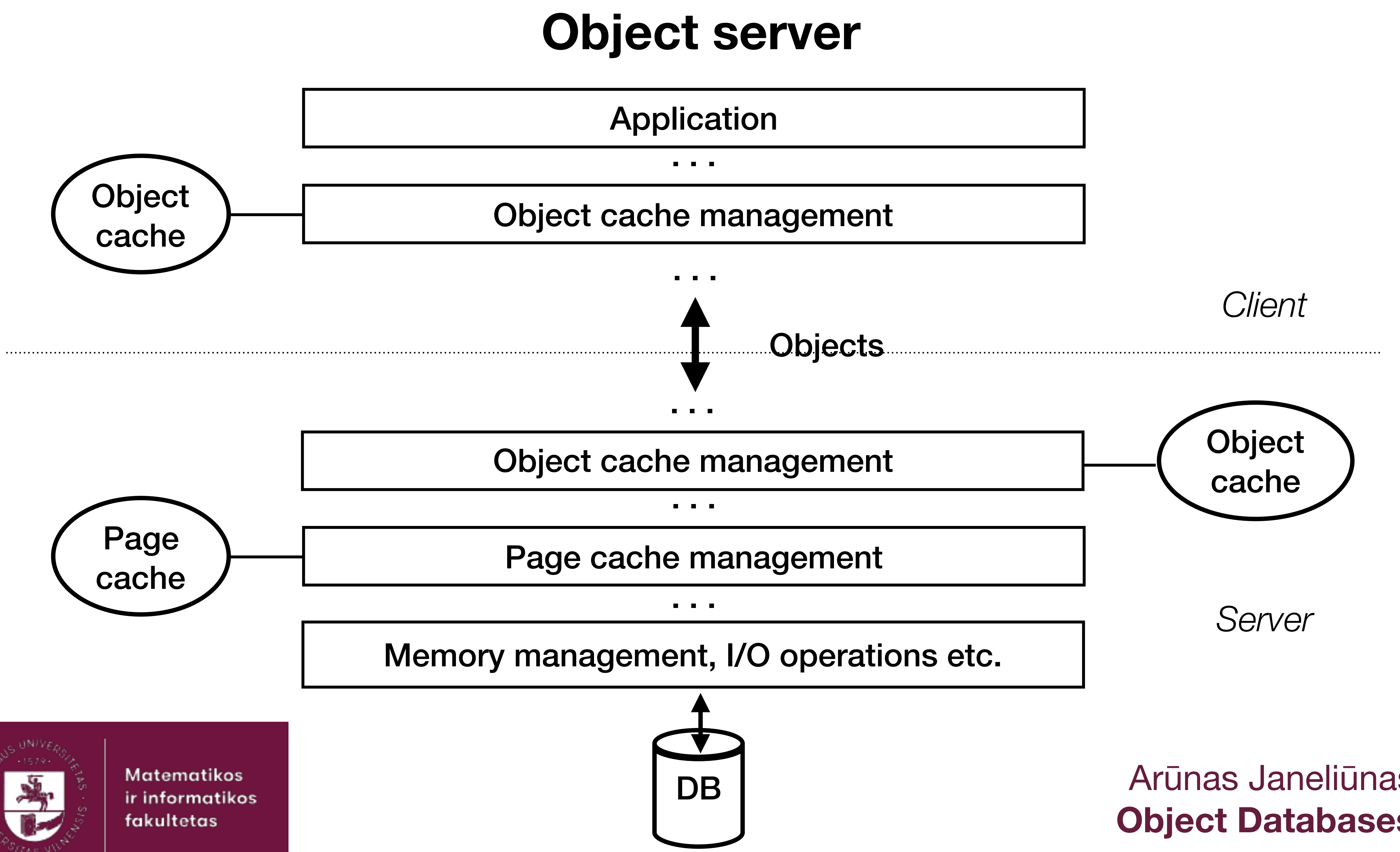
- Busy communication while sending many small objects
- Concurrent access control is set to pages (and every data on the same page, regardless whether it is occupied ATM or not)

Server output

Object server

Server sends out objects.

Server output



Server output

Object server

Server sends out objects.

PROS

- Concurrency control is more sophisticated

CONS

- More communication on checking data size which is sent over
- Ineffective communication when sending small objects
- More complex to implement

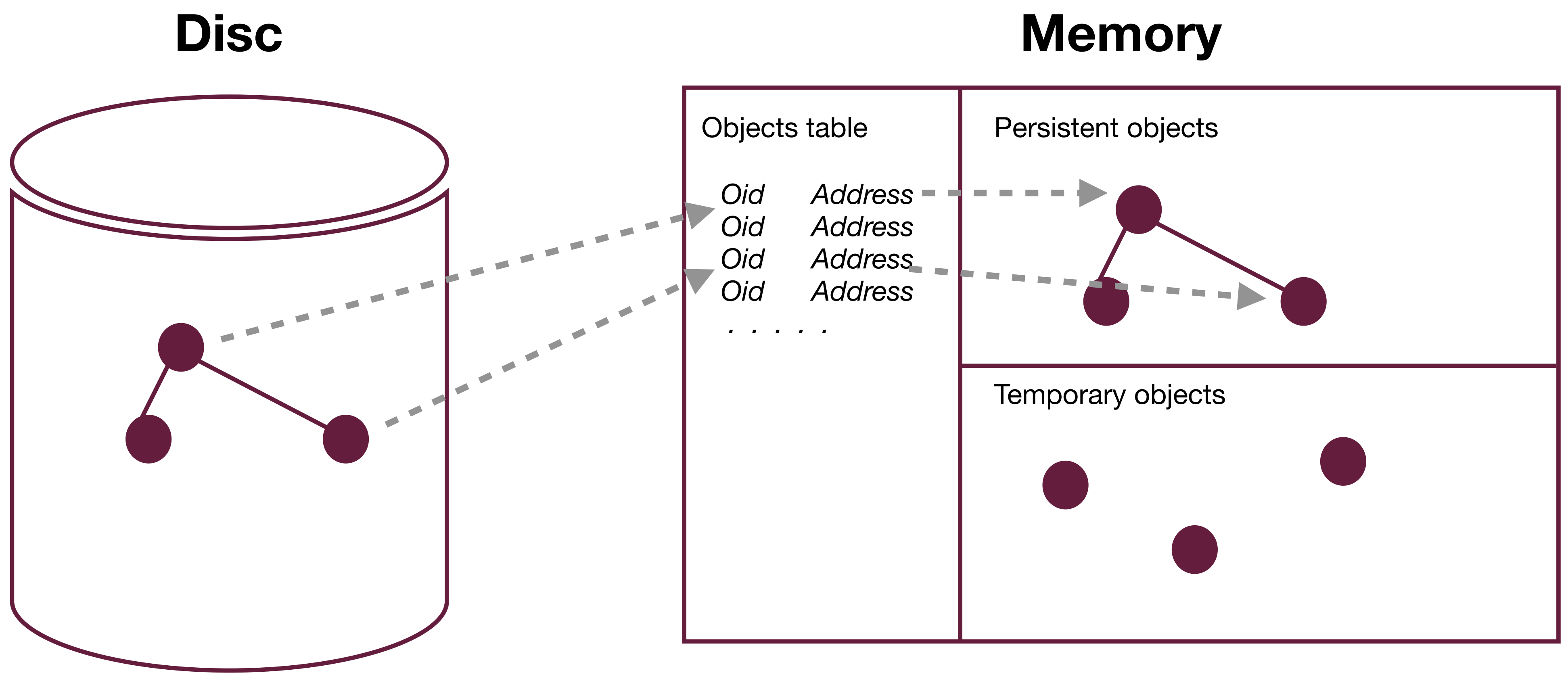
Object identification

- Usually object IDs contain information (address) where the object is to be found (in the memory)

- If some objects „live“ both on the server (disc) and application (memory), how their IDs are constructed and supported thren?

Object identification

Disc-oriented addresses



Object identification

Disc-oriented addresses

Each time a persistent object is created:

- the space in the disc is allocated
- *oid* is created for the object
- the entry in the Objects table is created
- only then the object is placed in the memory

Object identification

Disc-oriented addresses

PROS

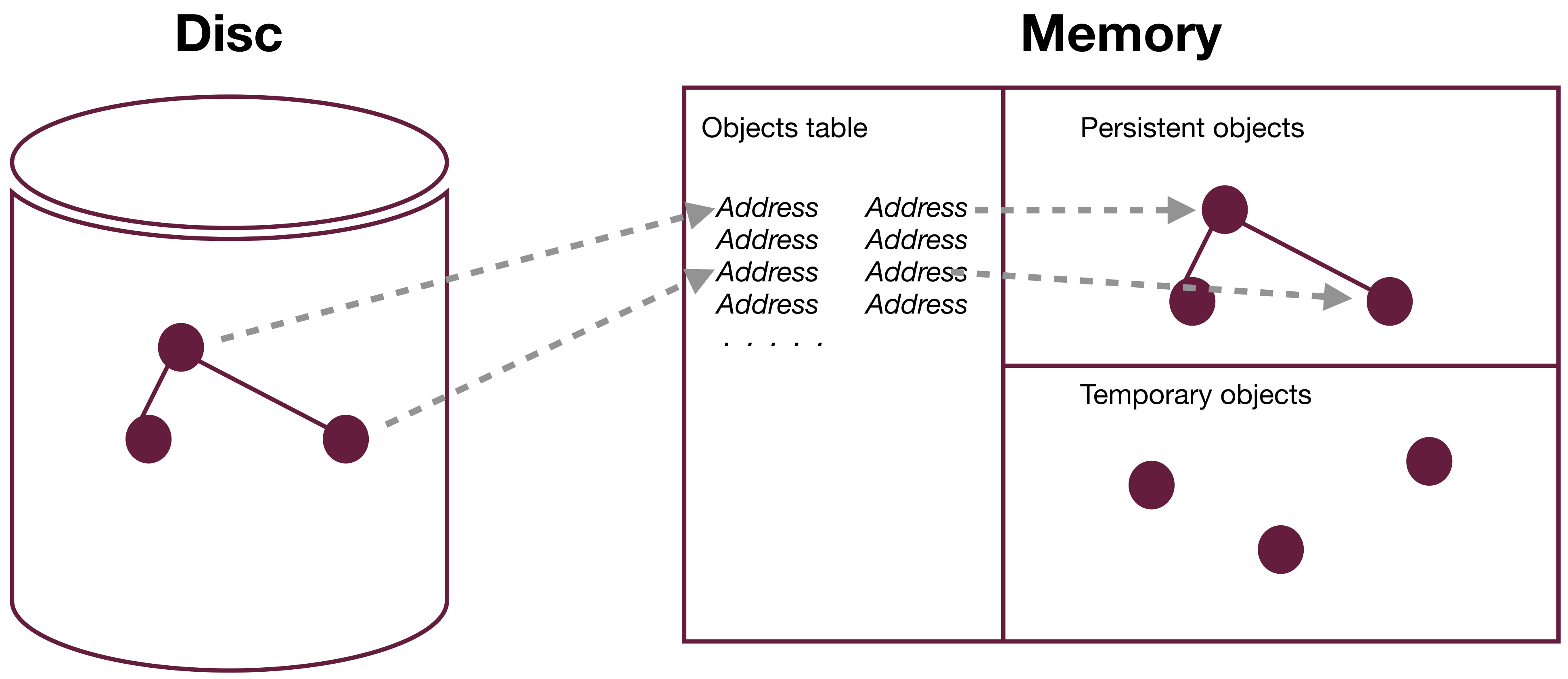
- The storage space in the disc is controlled
- No need to convert *oid*s on copying the object from/to database to the memory

CONS

- The link between objects in the memory is always indirect: you've got the *oid* of the other object, then you go to the Objects table, find this *oid* and only then you know where this object is located

Object identification

Two-levels addressing



Object identification

Two-levels addressing

The object addressing is always converted on moving it from/to the memory. Objects in the disc are addressed directly by the address in the disc. If it is copied in the memory, then it stats being addresses by its address in the disc.

But then it means, if you copy the object from the disc to the memory, you should fill in all its links to the other objects by their correct address in the memory. For that you should copy the linked object to the memory to get a „memory address“ for it. And so on, and so on… Possible chain reaction

Object identification

Disc-oriented addresses

PROS

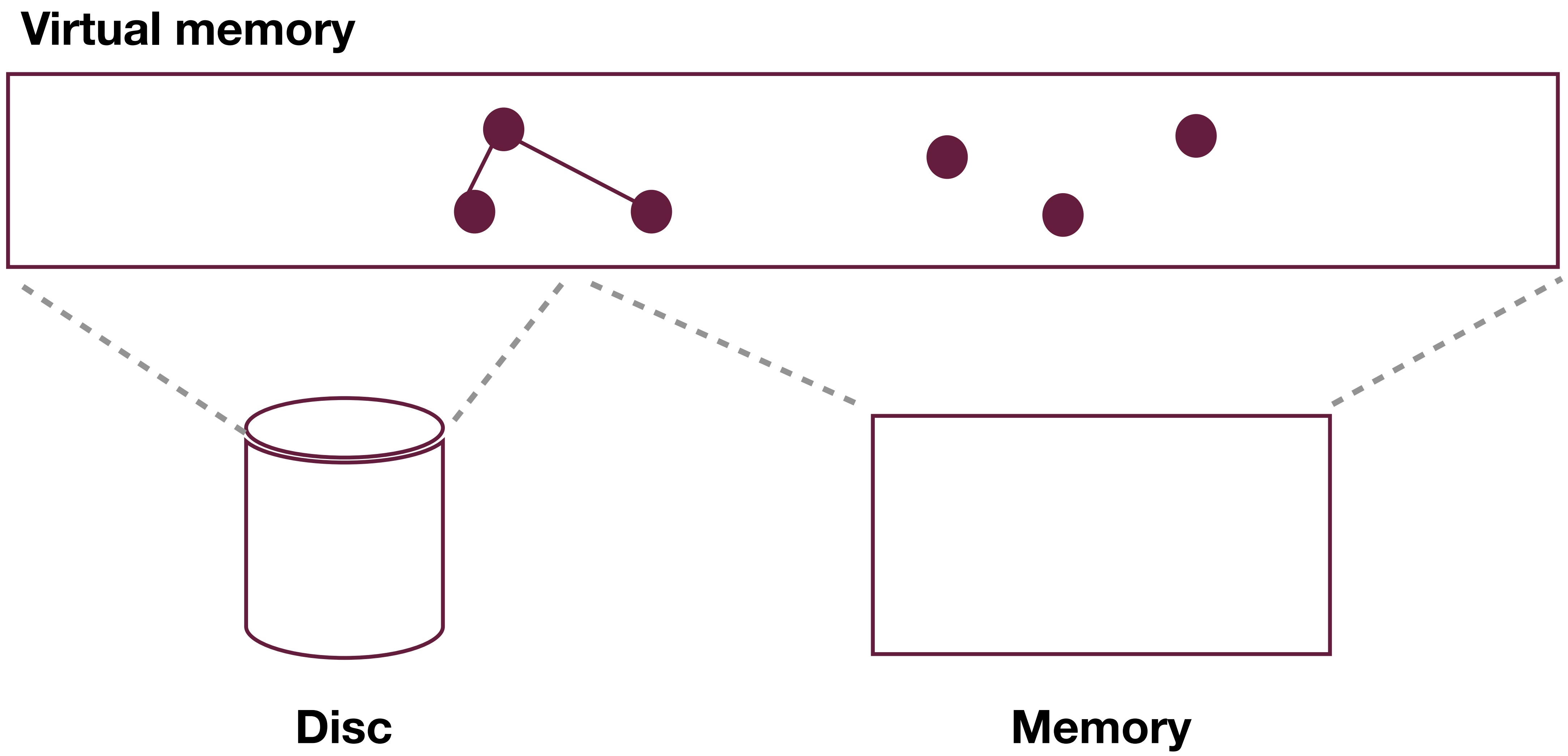
- The storage space in the disc is controlled
- Programming language can have no idea where objects are comming from, they all are linked the same.
- The link between objects is quick.

CONS

- Costly copying objects from the disc to the memory and *vice versa*.

Object identification

Single-level addressing



Object identification

Single-level addressing

All objects, regardless their „physical“ location, are operating in a single Virtual Memory. And the mapping function behind it is responsible to maintain this virtuality.

How temporary objects are to be distinguished in such a Virtual Memory?

What about Virtual Memory Fragmentation?