Databases

Elements of Data Science and Artificial Intelligence

Prof. Dr. Jens Dittrich

bigdata.uni-saarland.de

January 16, 2020

The "Database"-story so far

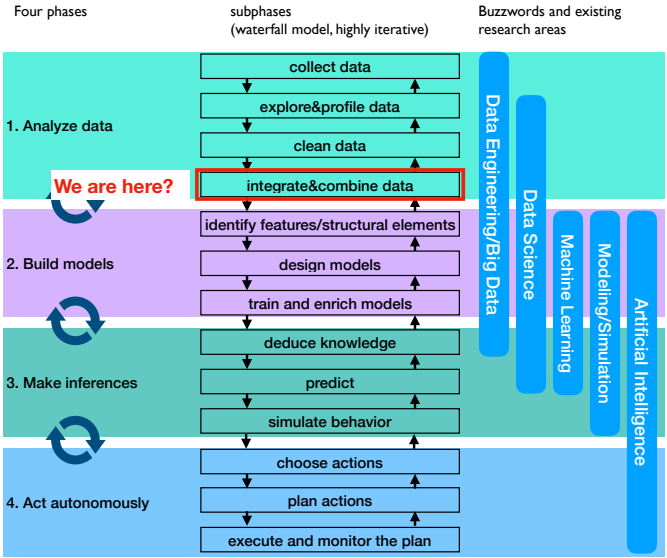
from the Introduction to Data Science-lecture:

"*Databases* are great to integrate and combine data."
(see slide set "02 Introduction to Data Science")

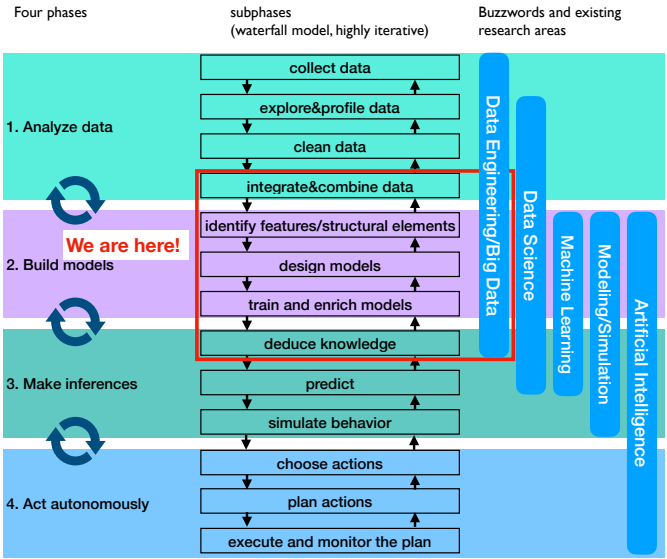
from the NLP-lectures:

:♩ "In NLP you eventually have to ask a *database*..." :
(see NLP slide sets)

DSAI Process Model



DSAI Process Model



Dictionary: Process Model: MLish to Databasish

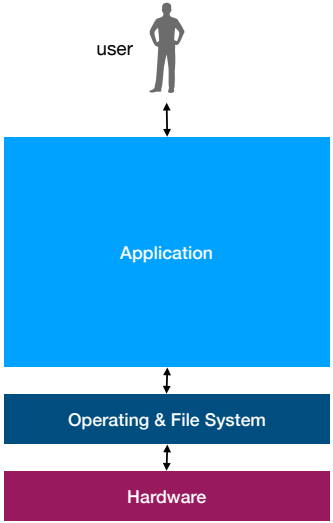
DSAI process model	**MLish**	**Databasish**
high-level idea	**interpretation**	**interpretation**
identify features/structural elements	analyze, abstract (leave away), and enrich data to identify (and add) important attributes	analyze, abstract (leave away), and enrich data to identify (and add) important attributes *and entities*
design models	design a model using neural networks, CNNs, tree-classifier, reinforcement learning, etc., pick/design loss functions	design a data model using entity-relationship modeling and the relational model
train and enrich models	adjust model weights, adjust hyperparameters	implement the relational model in SQL DDL (called the database schema) and load data into the database schema
deduce knowledge	predict something using the model	analyze data model using SQL-queries

ML vs a Database: When to pick what?

	ML	Database
How?	based on model that was trained using old training data; that data does not exist in the model anymore (unless the model overfits)	old training data, all data (typically) still available, i.e. the model simply memorizes all data (the model is in maximum overfit)
Query specification?	simple: based on tasks like classification and regression	complex: based on SQL
Result Quality		
Advantage?	may generalize (beyond what SQL can do)	precise (beyond what ML can do), no loss
Disadvantage?	approximate (possible loss)	missing generalization

Fo some scenarios both approaches may be suitable. Ideally both should be combined. And that is what a lot of current research is about... (systems for ML, ML for systems)

Why Databases? A two-layer Software-Architecture



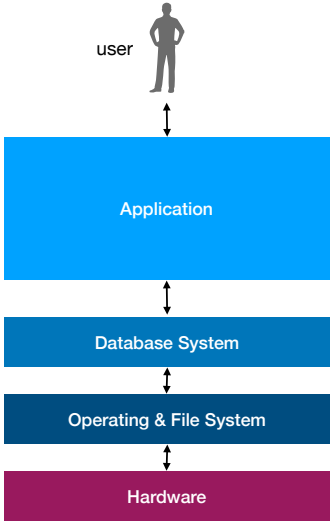
examples:

map-browsing (e.g. Google Maps),
image collection (e.g. Lightroom),
data analytics software (e.g. Tableau)
including data management code

Linux, Windows, OS X,
Android, iOS

CPU, DRAM, SSD, hard
disk

Why Databases? A three-layer Software-Architecture



examples:

map-browsing (e.g. Google Maps),
image collection (e.g. Lightroom),
data analytics software (e.g. Tableau)
without data management code

PostgreSQL, MySQL,
Oracle, SQLite

Linux, Windows, OS X,
Android, iOS

CPU, DRAM, SSD, hard
disk

Advantages of Having a Separate Database Layer

Application developers...

1. do not have to reinvent common and generic data managing tasks over and over again for every application (**separation of concerns**)
2. can (more or less) ignore how exactly data is stored and retrieved by the database system
3. have more time to focus on their actual application which hopefully increases their overall productivity
4. do not have to test the data management code (which is delegated to the developers of the database system!)
5. may easily exchange the database system against a different database system (well, at least that was the idea initially...), e.g. to scale an application

The Laziness Principles in Computer Science

The Laziness Principle

Whenever possible try to map (sub)problems to an existing problem. Then use *existing solutions* to solve that (sub)problem rather than reinventing everything from scratch.

In the context of today's lecture *existing solutions* means: use a database system rather than coding the data manageemnt stuff yourself! But in other contexts it may also mean any other suitable software (sub-)system and/or library.

The Missed Opportunity for Laziness Principle

If you do not know that a (sub)problem could be mapped to an existing problem, you miss the chance to apply The Laziness Principle.

In other words: if you do not know that certain problems can effectively be solved in certain ways, you will not be able to be lazy! For instance, assume you are simply not aware that of a technique X that is always suitable when there is a problem of type Y.

Disavantages of Having a Separate Database Layer

Application developers...

1. have to live with the interfaces and features provided by the DBMS
2. have to know how to use a DBMS (many developers fail miserably here)
3. if you are unhappy with anything done by the DBMS (see 1.),
   ~~you are screwed~~,
   learn, learn, and learn, i.e.: do not blame the DBMS for something which is very likely your fault (see 2.) ...

Database Management Systems (Repeated&Improved)

Key questions:

1. How to store, access, and query data?
2. How to make query processing efficient and scalable?
3. How to make this happen for just any kind of data?
4. How to abstract away physical properties?
5. How to abstract away concurrency control?
6. How to recover after a failure?

Killer contributions: relational model, relational algebra, structured query language (SQL), transactions, and all kinds of algorithms & systems that make the former efficient and robust

Famous products: IBM Db 2, Oracle, PostgreSQL, MySQL, MonetDB, SQLite, MS SQL Server, SAP Hana, Tableau, Spark, ...

Biggest Failures: XQuery (XML query processing), Object-oriented Databases, NoSQL (mostly reinvents very old relational technology), native, non-relational storage (LOL!), debatable horizontal scale-out (for very large installations)

History: huge, very active research field since the early 60ies, ACM SIGMOD, VLDB

In the Following: Learn by Application

rather than introducing and investigating these concepts independently (bottom-up), in the following, we will introduce some key concepts by analyzing a concrete application (top-down)

We will take a closer look at Google Maps (you should recall our initial discussion from the Perspektiven lecture, anyways, I will show again some of those slides in the following). We will have a more technical discussion about this today and in the next weeks.


Kanada
Ozean
Arktischer
Ozean
Japan
Japanisches Meer
USA
Nordwestpassage
Baffin Bay
Arktischer
Ozean
Südkorea
Ostchinesisches
Meer
Karasee
Russland
Grönlandsee
Barentssee
Mongolei
Grönland
Labradorsee
China
Europäisches
Nordmeer
Island
Schweden
Finnland
Kuba
Norwegen
Kasachstan
Kirgisistan
Thailand
Karibisches
Meer
Puerto Rico
Vereinigtes
Königreich
Irland
Dänemark
Weißrussland
Usbekistan
Nepal
Golf von
Thailand
Polen
Deutschland
Ukraine
Nord
Atlantischer
Ozean
Turkmenistan
Indien
Venezuela
Österreich
Pakistan
Golf von
Bengalen
Frankreich
Rumänien
Spanien
Italien
Türkei
Iran
Guyana
Portugal
Griechenland
Syrien
Irak
Suriname
Lakkadivensee
Mittelmeer
Oman
Marokko
Tunesien
Arabisches
Meer
Saudi-Arabien
Westsahara
Algerien
Ägypten
Libyen
Jemen
Golf von Aden
Mauretanien
Mali
Sudan
Niger
Äthiopien
Guinea
Burkina
Faso
Tschad
Somalia
Ghana
Nigeria
Südsudan
Kenia
Golf von
Guinea
Google
Demokratische
Tansania
Satellit
Kartendaten © 2019 Google,INEGI
Deutschland
Bedingungen
Feedback geben
1000 km

Application Equivalence Classes: more Opportunity for Laziness (1/2)

Google Maps is technically highly related to several other applications:

- medicine: image data from MRTs or any other radiology device
- material sciences: any image data from any "see-through"-device
- astronomy: 3d star-catalogues, e.g.. Sloan Digital Sky Survey
  `https://en.wikipedia.org/wiki/Sloan_Digital_Sky_Survey`
- geography/geology/meteorology data over time: 4D-data about the state of the planet, e.g. `https://www.washingtonpost.com/graphics/2019/national/climate-environment/thermometers-climate-change/`
- computer (online) games: when to load which texture, when to show which player/avatar
- biology: 3D-brain/molecule/organ/plant/animal/etc.-catalogues, e.g. The Human Brain Project: `https://www.humanbrainproject.eu/en/explore-the-brain/`

Application Equivalence Classes: more Opportunity for Laziness (2/2)

- cellphone: 4D-data on which device is where and when?,
  e.g., the recent NYT article about public data on this:
  `https://www.nytimes.com/interactive/2019/12/19/opinion/location-tracking-cell-phone.html`
- traffic: 4D-vehicle data: which car/flight/ship is where and when?
  e.g. FlightRadar
  `https://www.flightradar24.com/`
- self-driving cars: 2D street maps, 4D-free space maps, e.g. slides by Bernt Schiele
- census data: who lived where and when?
  e.g. US Census Data
  `https://www.census.gov/programs-surveys/geography/data/interactive-maps.html`
- election data: who voted for which party and when?

Survey

Why is it important to think along application equivalence classes?

(A): The professor can brag about how important his field is.

(B): Techniques that were used in a particular application X may be useful for other applications Y in that class as well.

(C): It might be a starting point to think along more generic applications that support a larger subset of the applications in a class.

(D): An application X in a particular class may be easily adapted to become application Y.

Solution (A–D)

all correct!

Major Challenges with this Application Equivalence Class

Potential problems:

- potentially large volumes of data:
  does not fit into main memory and/or on local machine, hence: high load on storage and network
- large number of concurrent users
  high load on storage and network

Requirements:

- seamless user-experience, i.e. seamless zooming and panning
- do not overload servers, network, and clients
- close to zero downtime (in particular in case of hardware failures)

The Key Questions with Google Maps (1/2)

Key questions:	**for this concrete application (Google Maps):**
1. How to store, access, and query data?	**where** and **how** to store and cache the data?
2. How to make query processing efficient and scalable?	which queries?: (a) 2-dimensional range queries, (b) text search on geonames. *How does a database process such a query?*
3. How to make this happen for just any kind of data	what data?: (a) satellite images (raster data), (b) roads, borders, etc. (vector data), (c) geographic names (text)

The Key Questions with Google Maps (2/2)

Key questions:	**for this concrete application (Google Maps):**
4. How to abstract away physical properties?	physical properties: (a) huge network of servers distributed around the globe (hardware), (b) decision for certain data structures and algorithms used internally (how to compute stuff) *How come we do not have to worry about this?*
5. How to abstract away concurrency control?	many Google Maps users access the same map data concurrently, is that a problem?
6. How to recover after a failure?	what if any server goes down or storage space is lost? Will Google Maps still work?

The Key Questions with Google Maps (1/2)

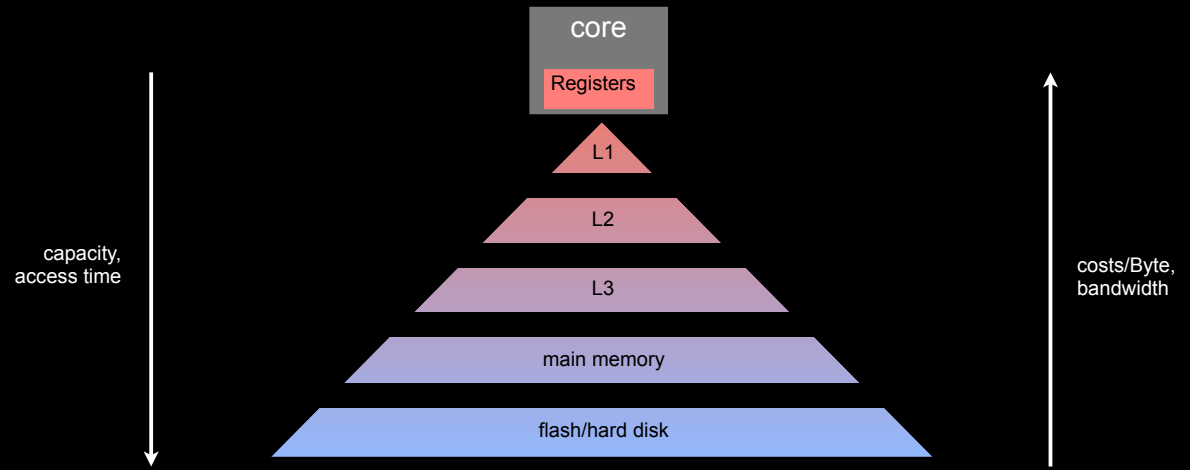
Key questions:

1. How to store, access, and query data?

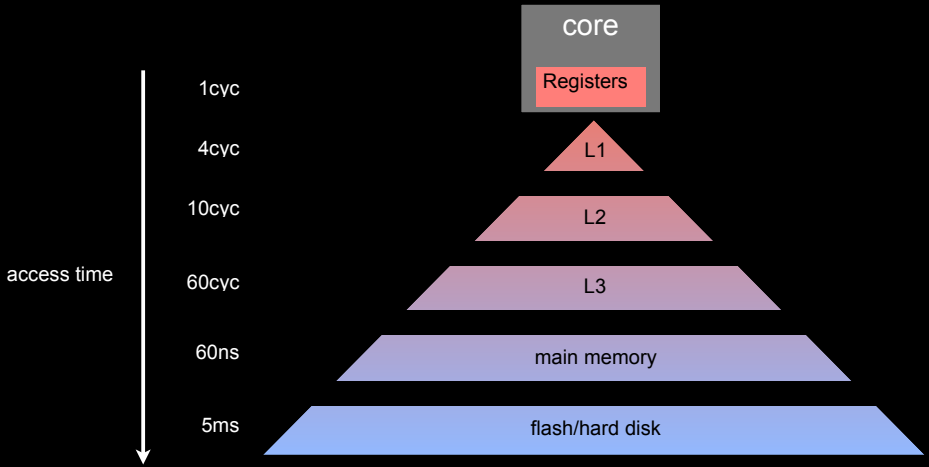
for this concrete application (Google Maps):

where and how to store and cache the data?

The Storage Hierarchy



Typical Access Times



Relative Distances!

Factor 45

Factor 15

Factor 2.5

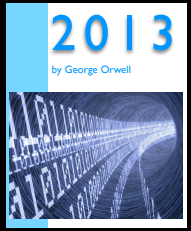
"L1 cache is like grabbing a piece of paper from your desk
(2 second),

L2 cache is picking up a book from a nearby shelf
(5 seconds),

L3 cache is picking up a book from the next room
(30 seconds),

DRAM is taking a walk down the hall to buy a Twix bar
(90 seconds)."


Metal Balls in Bowery



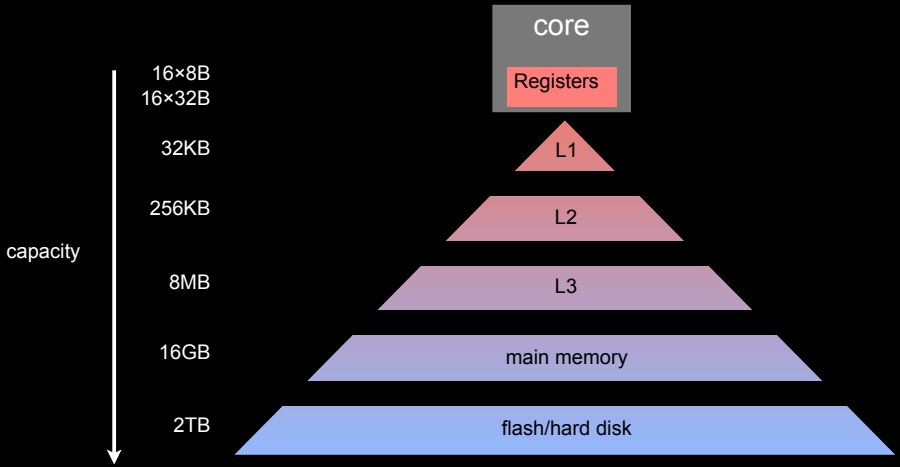
Factor 3,750,000

“hard disk is like
walking from Saarland to Hawaii.“

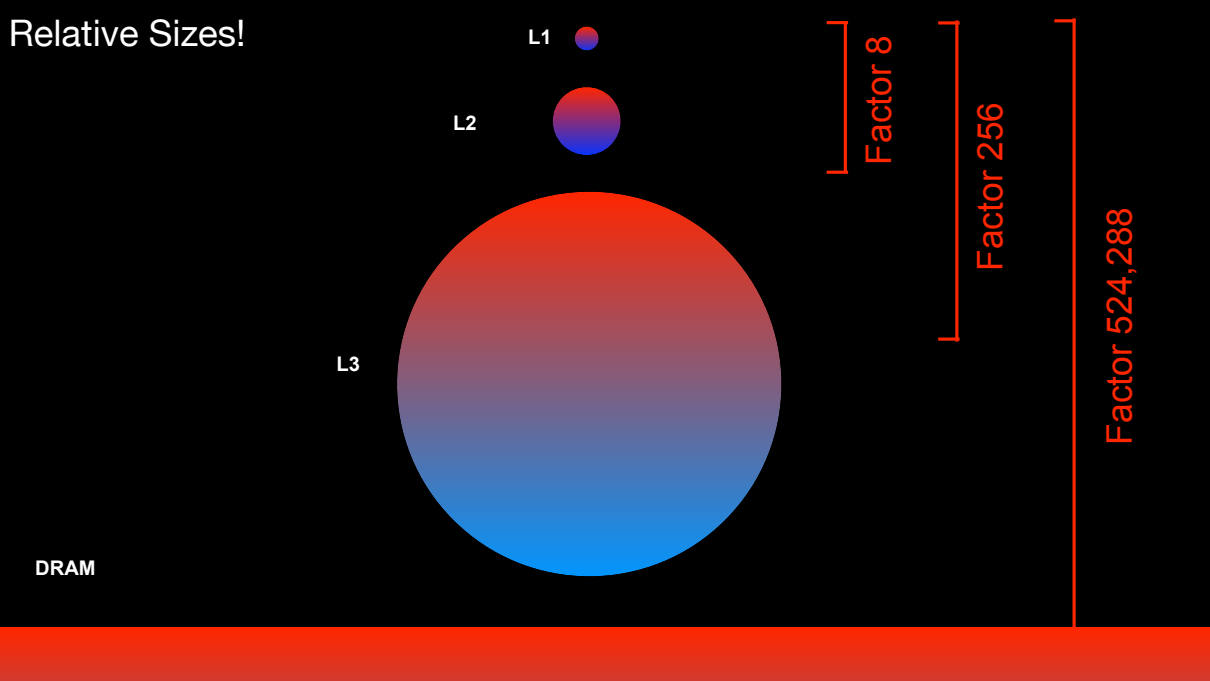
7,500,000 seconds of walking!

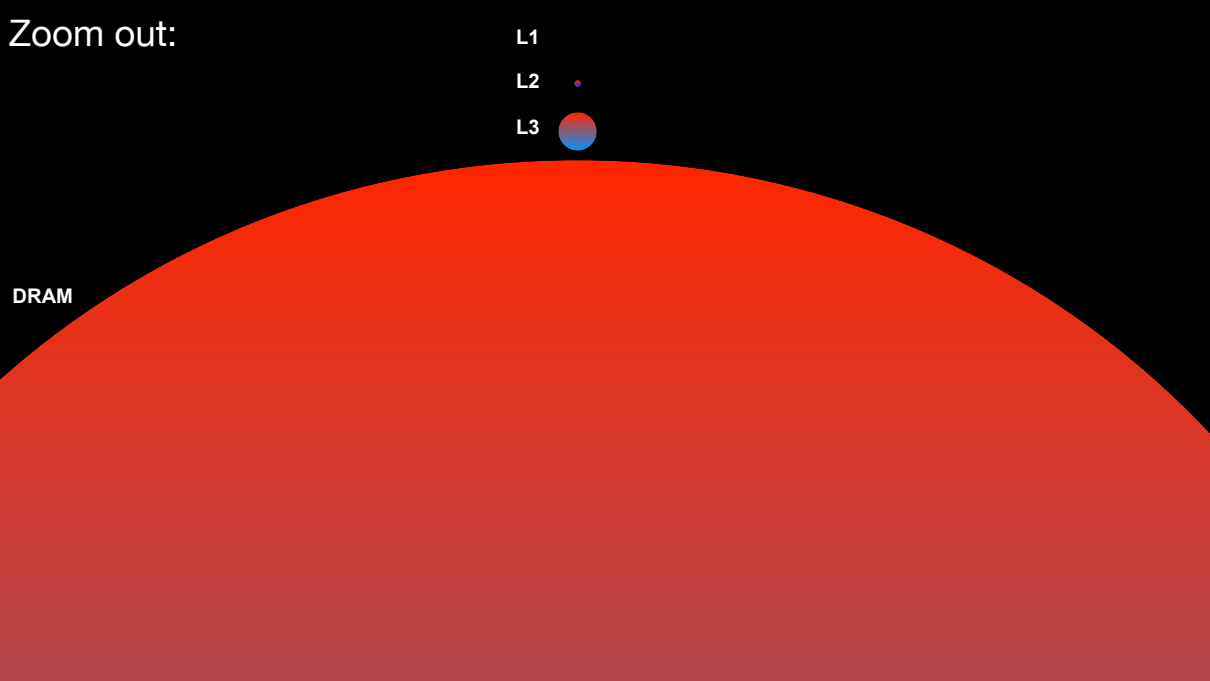
= 86.8 days!

Typical Sizes



Relative Sizes!




Zoom out:
L1
L2
L3
DRAM

Tasks of Each Layer in a Storage Hierarchy

Four major tasks:

1. localization of data objects:
   Is data item x available in this layer?
2. caching of data from lower (slower) levels:
   Shall we store data item x in this layer?
3. data replacement strategies:
   Which data item x should we remove to make room for new data items in this layer?
4. writing modified data:
   If data item x was modified, should we also modify the copies of x in the layers underneath?

Distribution **In**dependence in a Storage Hierarchy

Distribution **In**dependence

When working with computer memory we typically do not see whether that memory is mapped to a particular location. All of this is hidden for us and handled automatically by the computer system (operating system and hardware, in particular through virtual memory management). We **do not have control** over how these tasks are performed[a].

[a] Well, basically: there are many tricks around this...

This term was coined by Edd Codd, one of the founding fathers of relational database technology:
`https://en.wikipedia.org/wiki/Edgar_F._Codd`,
`https://en.wikipedia.org/wiki/Codd%27s_12_rules`

Partial Distribution Independence

Partial Distribution Independence

Most computer systems provide a mix where Distribution **In**dependence holds for some of the storage layers while Distribution Dependence holds for others.

Examples:

In a CPU, as long as we talk about everything in-between L1, L2, L3, and main memory, distribution independence holds:

- As long as data is "in memory", we simply see a linear address space $[0, \ldots, N]$.
- We can then addresss memory, e.g. readByte(42) to read the byte at position 42 and writeByte(42,17), to write byte 17 to position 42.

In contrast, in-between hard disk (and/or SSDs) $\leftrightarrow$ main memory distribution independence typically does **not** hold:

Distribution Dependence on the Storage Layer

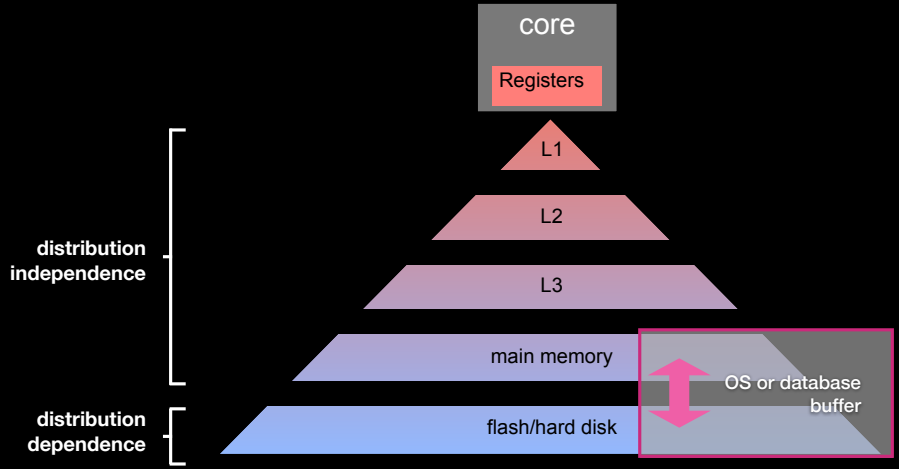
Distribution Dependence

For certain layers of the storage layer we **do have control** on when data is read and/or written and/or how the different tasks are performed on that storage layer.

Examples:

- From disk/SSD to main memory we all of a sudden make it explicit: "let's load/save that file".
- From the Internet to our machine/smartphone we say: "let's download/upload that file/webpage".
- From our machine to an external disk we say: "let's make a backup on that external disk".

Operating System vs Database Buffer



Buffer Replacement Strategies

Buffer

A buffer at a given storage layer keeps a copy of k data items from a lower (more distant) storage layer. A buffer has the following task/functions:

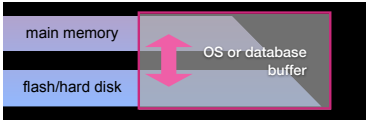
get(item): return a handle to a data item, assumes that a copy of the data item is already kept in the buffer

load(item): load a data item into the buffer

evict(): determine a data item to remove from the buffer, may trigger a write operation on a lower (more distant) storage layer

A buffer may be implemented in Software and/or Hardware.
The major decision when implementing a buffer is how to implement evict().

Example: Main-Memory Buffer



data items: 'pages' of 4KB each

get(pageID): return a handle to the page with pageID

load(pageID): load page with pageID from disk into main memory

evict(): determine a page to remove from the buffer, if that page was modified in main memory over the version on disk, we first have to write the changed version back to disk/flash

Buffer Replacement Strategies

The decision which data item to evict is called *replacement strategy*.
Well known strategies are:

- Least Recently Used (LRU): the data item that was *used the longest time ago* will be evicted
- First-In-First-Out (FIFO): the data item that was *loaded the longest time ago* will be evicted
- Least Frequently Used (LFU): the data item that was *used the least* will be evicted; this is implemented through some form of reference counting

see Jupyter notebook "LRU buffer"

Layer Entanglement

Storage Layer Task Implementation and Entanglement

How to implement the four different tasks on a particular storage layer depends on:

1. the physical properties of that layer (capacity, access times, bandwidth), **and**
2. its interaction with the other layers, **and**
3. what we want to do with the computer system!

General Purpose vs Domain-specific

General Purpose Storage Layer Implementation

The storage layer is implemented with the goal to support a very diverse set of applications.

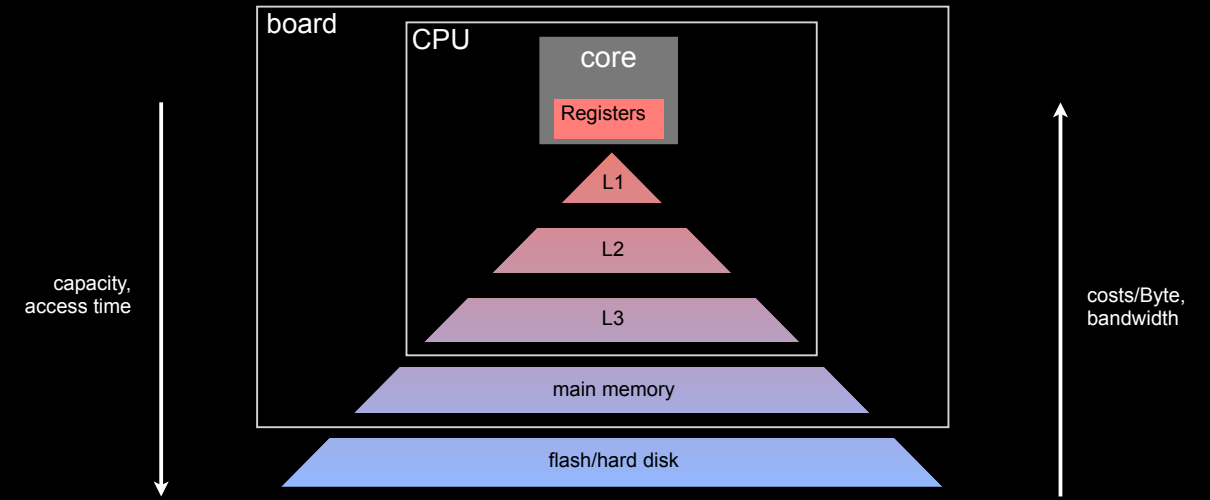
Example: the page cache of the Linux operating system, it implements tasks to handle hard disk (and/or SSDs) $\leftrightarrow$ main memory

Domain-specific Storage Layer Implementation

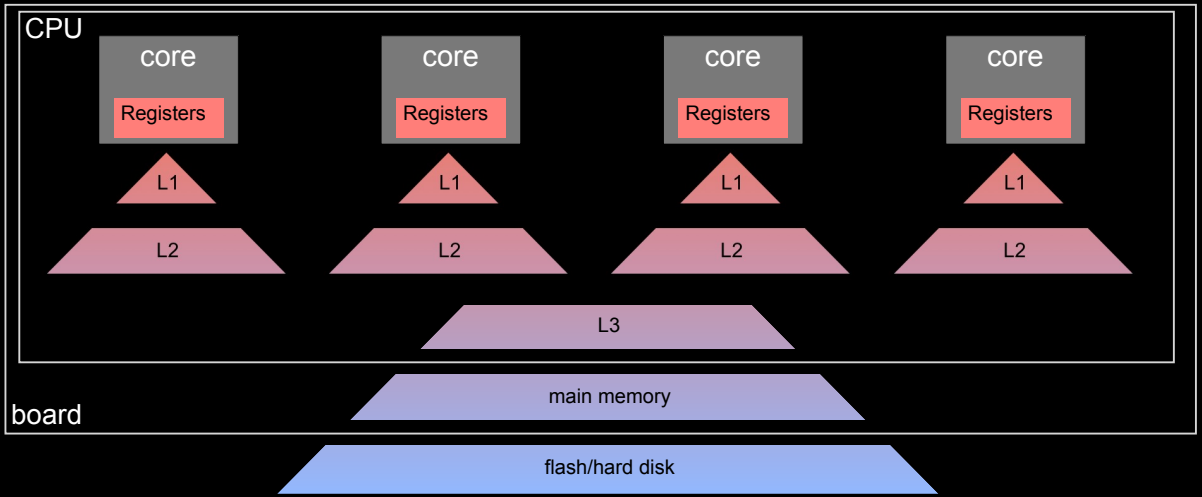
The storage layer is implemented with the goal to support a specific class of applications (i.e., an application domain).

Example: the database buffer as implemented by a database system X: it does more or less the same as the file cache of the Linux operating system, however: as a database system is more restricted in what kind of applications it supports, it can perform optimizations targeted to a specific class of applications

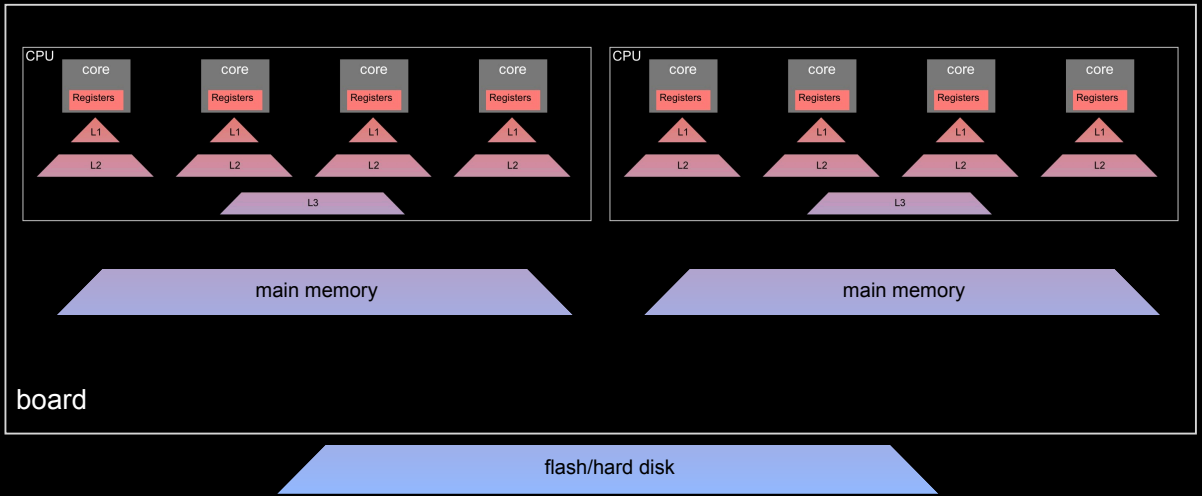
A Single-Core Storage Hierarchy



A Multicore Storage Hierarchy



Non-Uniform Memory Access (NUMA)



The Network is just Another Layer!

Simplification

Layers in a network can often be modeled just as like other storage layer. It is merely a matter of adjusting the constants (mainly access times, bandwidth, and storage sizes; everything else is details that can be ignored in most cases)

One computer in a Network