A systemic perspective to data (access) quality

Exploiting the implicit triangle between data, schemas and programs

Groupe de contact FNRS analyse critique et
amélioration de la qualité de l'information numérique
ULB - May 18, 2026

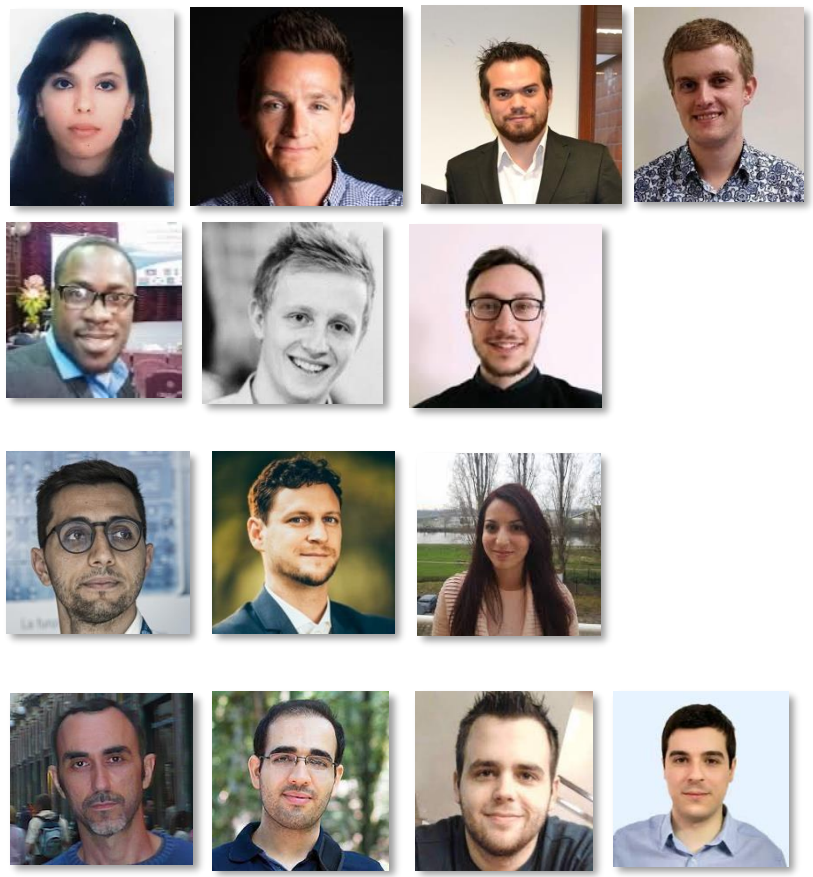
Anthony Cleve

Data-Intensive SysTems Evolution Lab (DISTEL)

PReCISE Research Center - Namur Digital Institute

University of Namur, Belgium

Credits - Data-Intensive SysTems Evolution Lab (DISTEL)



Since 2011

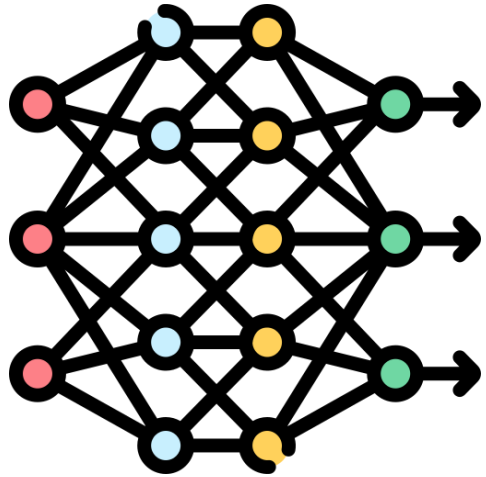
- 8 PhD researchers
  - 37,5% of them have Maxime as firstname
  - 66,7% of Maxime are present today
- 3 teaching assistants
- 3 post-doc researchers
- 4 visiting PhD researchers
- 44 Master thesis students

Introduction

Data-Intensive Systems (DIS)

Write data

Process data

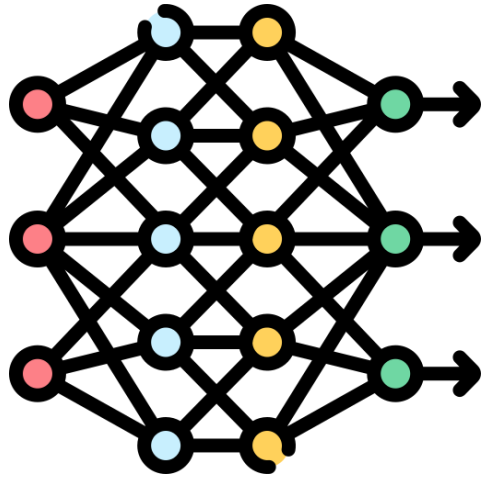


Learn from data

Data-Intensive Systems (DIS)

Write data

Process data

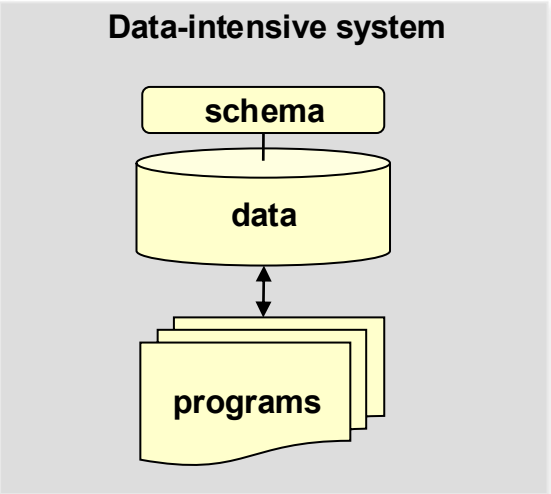


Learn from data

Data-Intensive Systems (DIS)

Data-intensive system = intensive use of data

- one (or several) database(s) containing mission-critical ***data***
- complying to a certain ***schema*** (data structures and constraints)
- a set of ***programs*** read/update the data via queries expressed on that schema



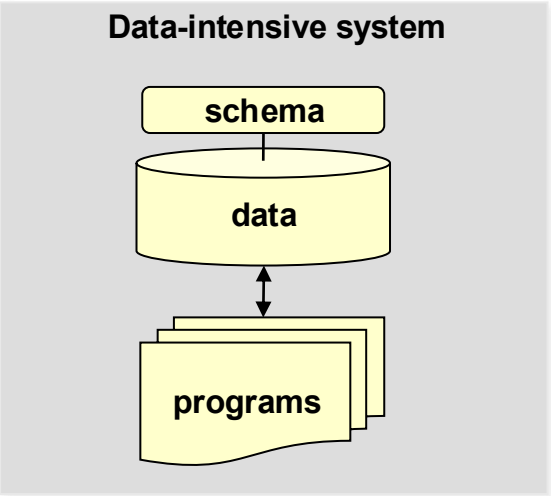
Link with digital information/data quality ?

The notion of *fitness for use*

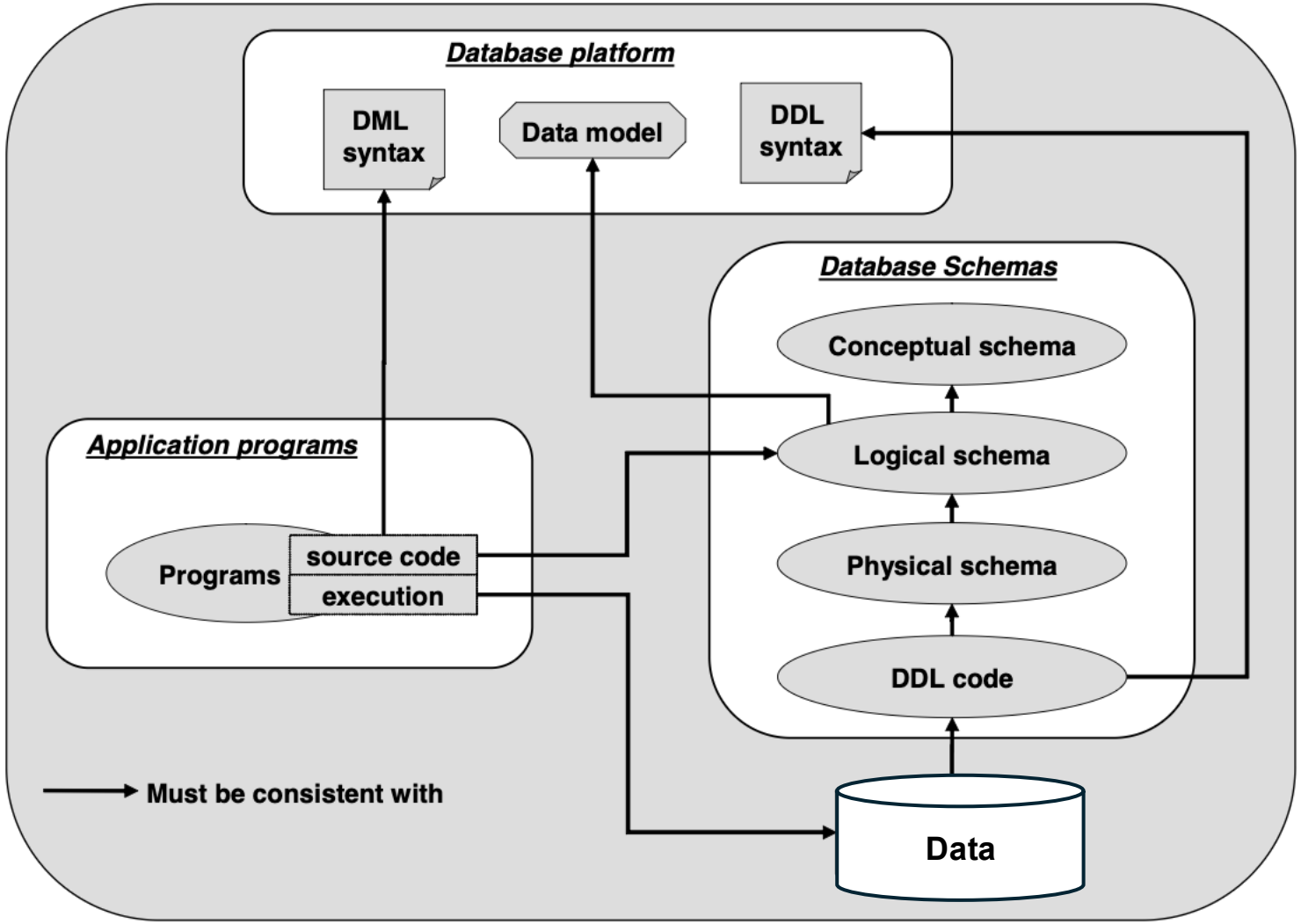
In DIS, data is *used* by programs to offer certain functionalities

Important (implicit) relationships between

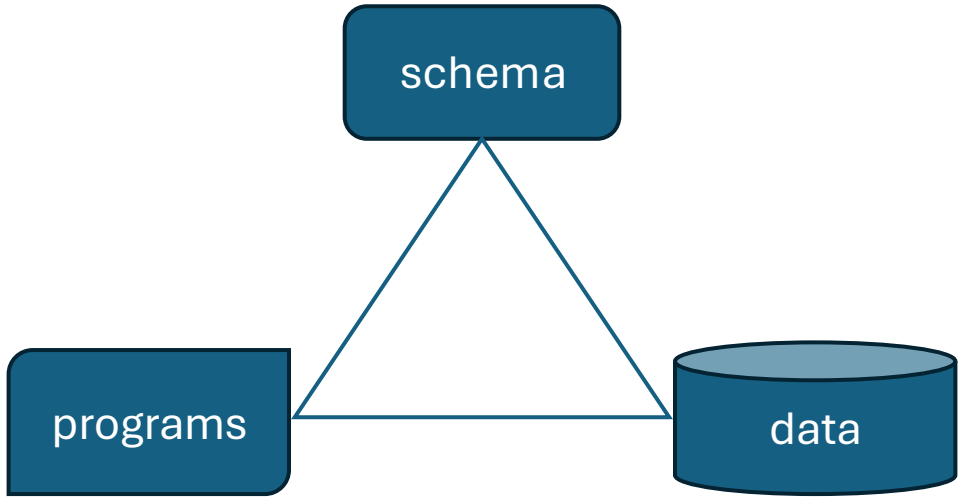
- data quality
- schema quality
- program quality



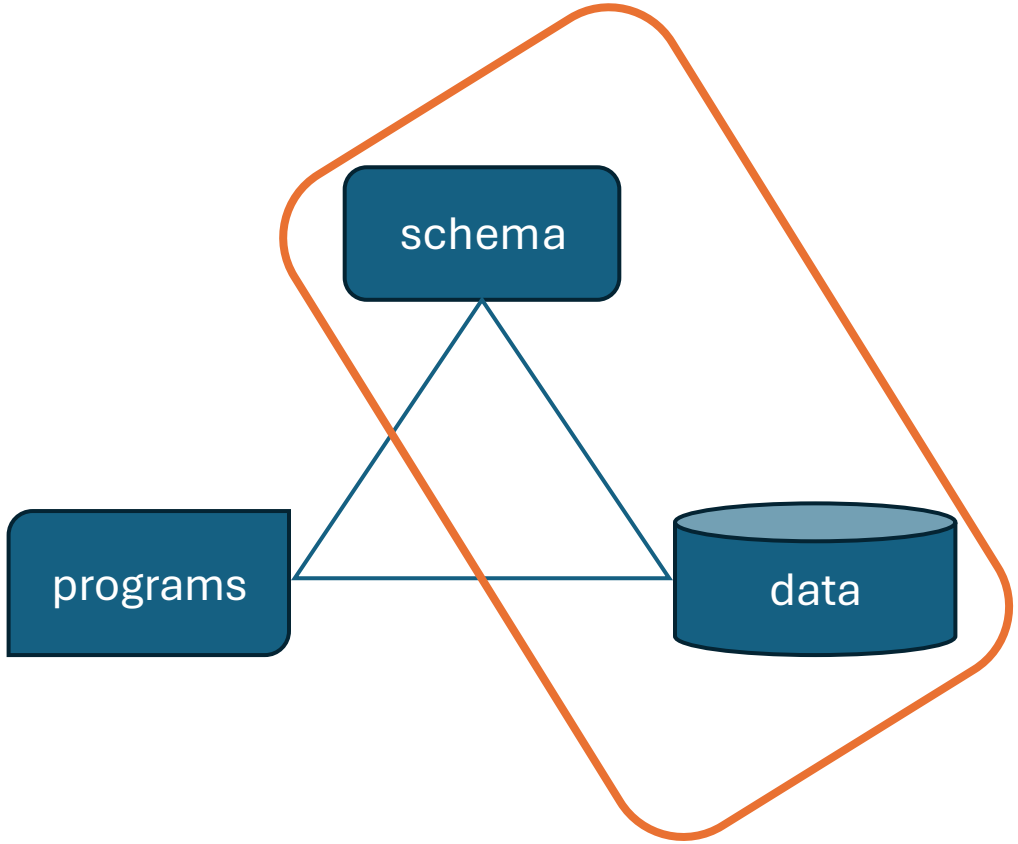
Consistency relationships in DIS (systemic perspective)



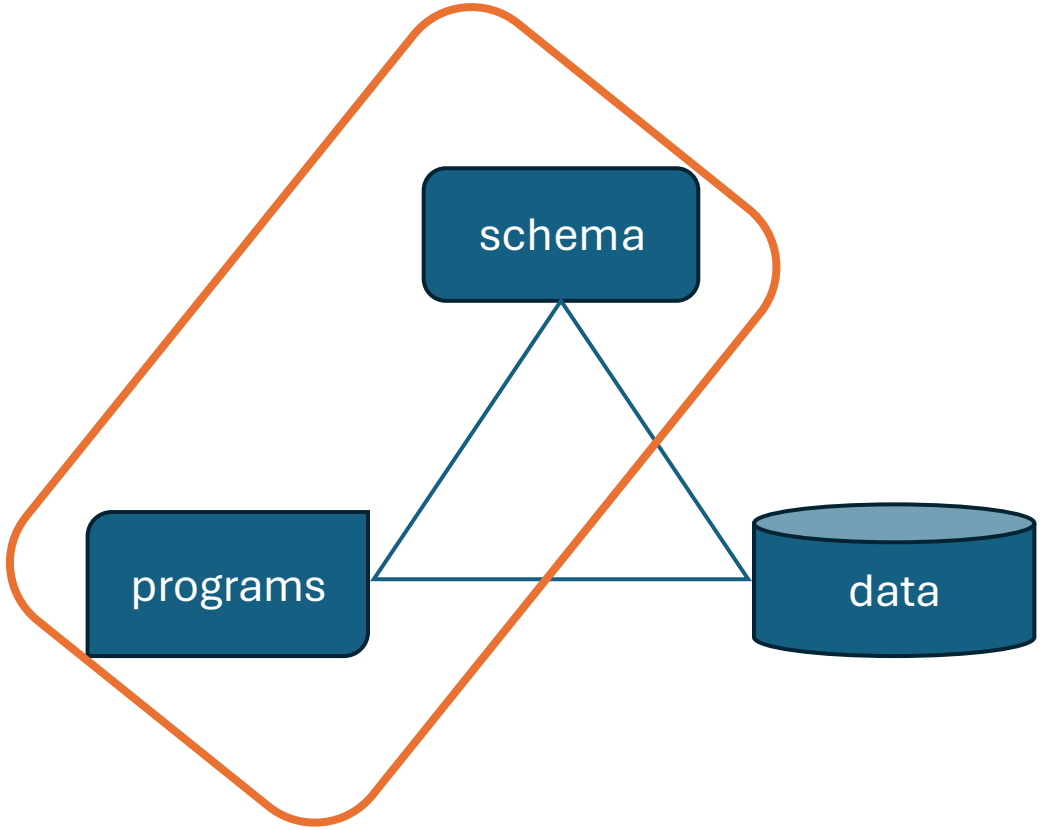
The consistency triangle of DIS



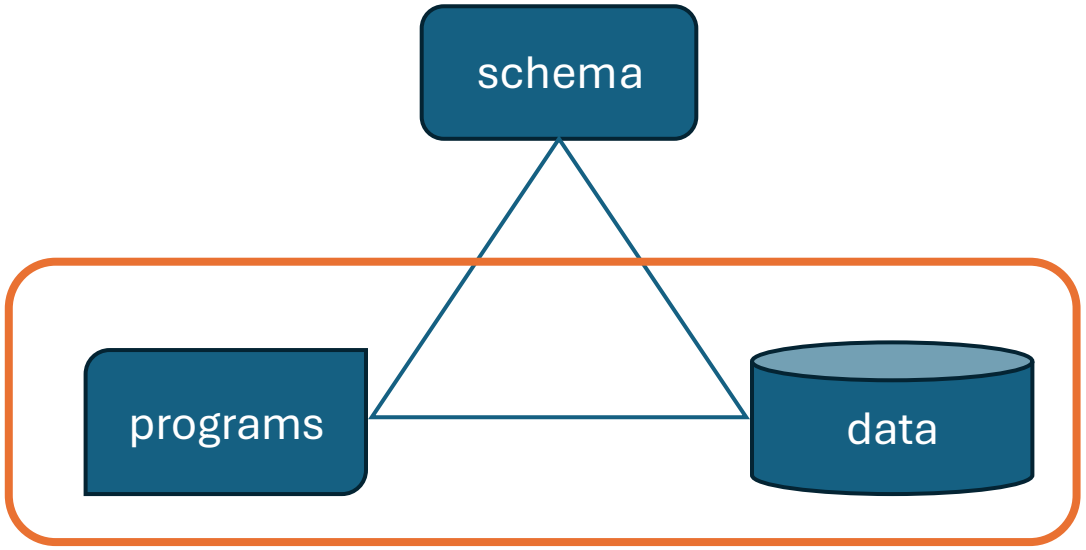
The consistency triangle of DIS



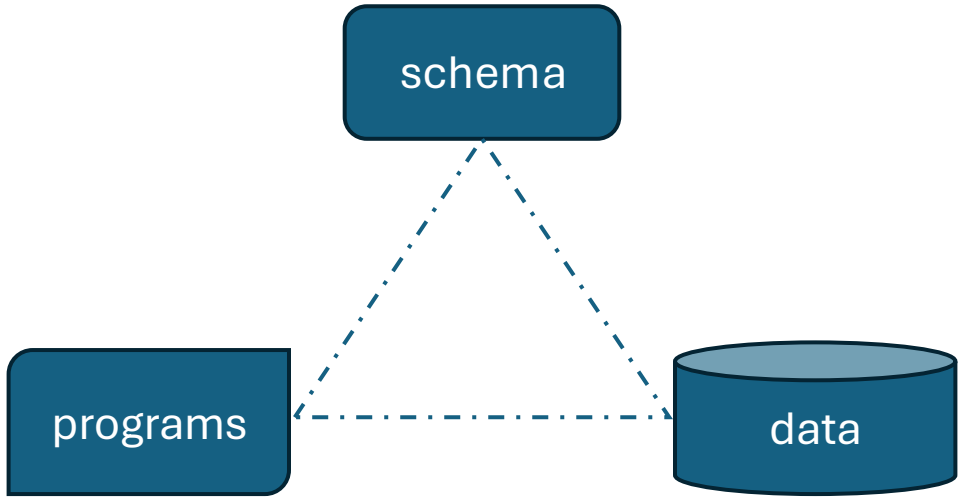
The consistency triangle of DIS



The consistency triangle of DIS

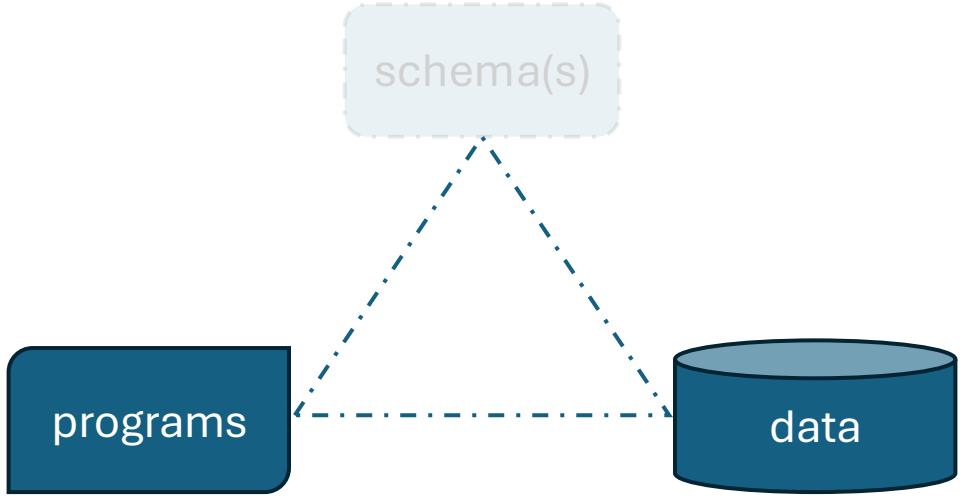


The (implicit) consistency triangle of DIS



The (increasingly implicit) consistency triangle of DIS

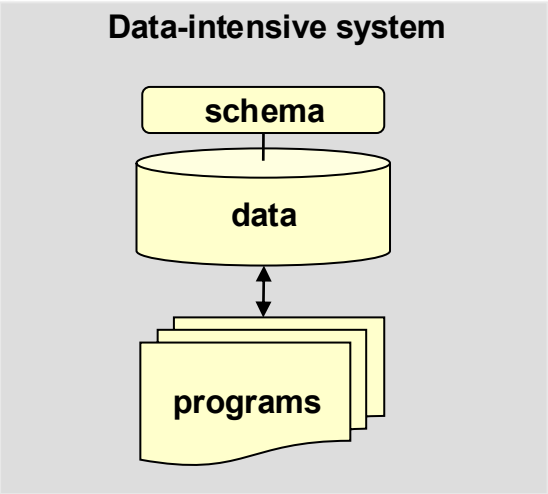
e.g., in the case of schema-less datastores (NoSQL)



Data-Intensive Systems (DIS)

Data-intensive system = intensive use of data

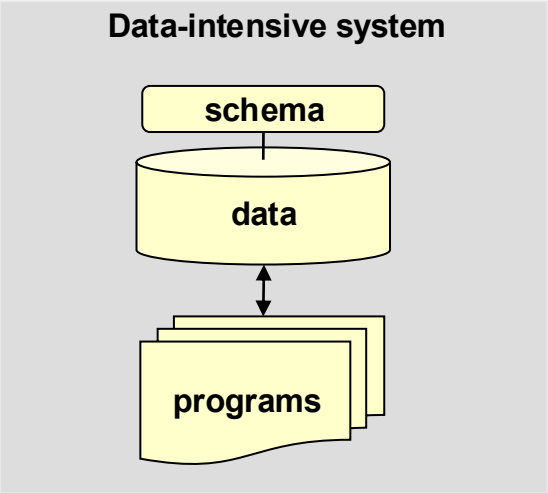
- one (or several) database(s) containing mission-critical ***data***
- complying to a certain ***schema*** (data structures and constraints)
- a set of ***programs*** read/update the data via queries expressed on that schema



Data-Intensive Systems (DIS)

Data-intensive system = intensive use of data

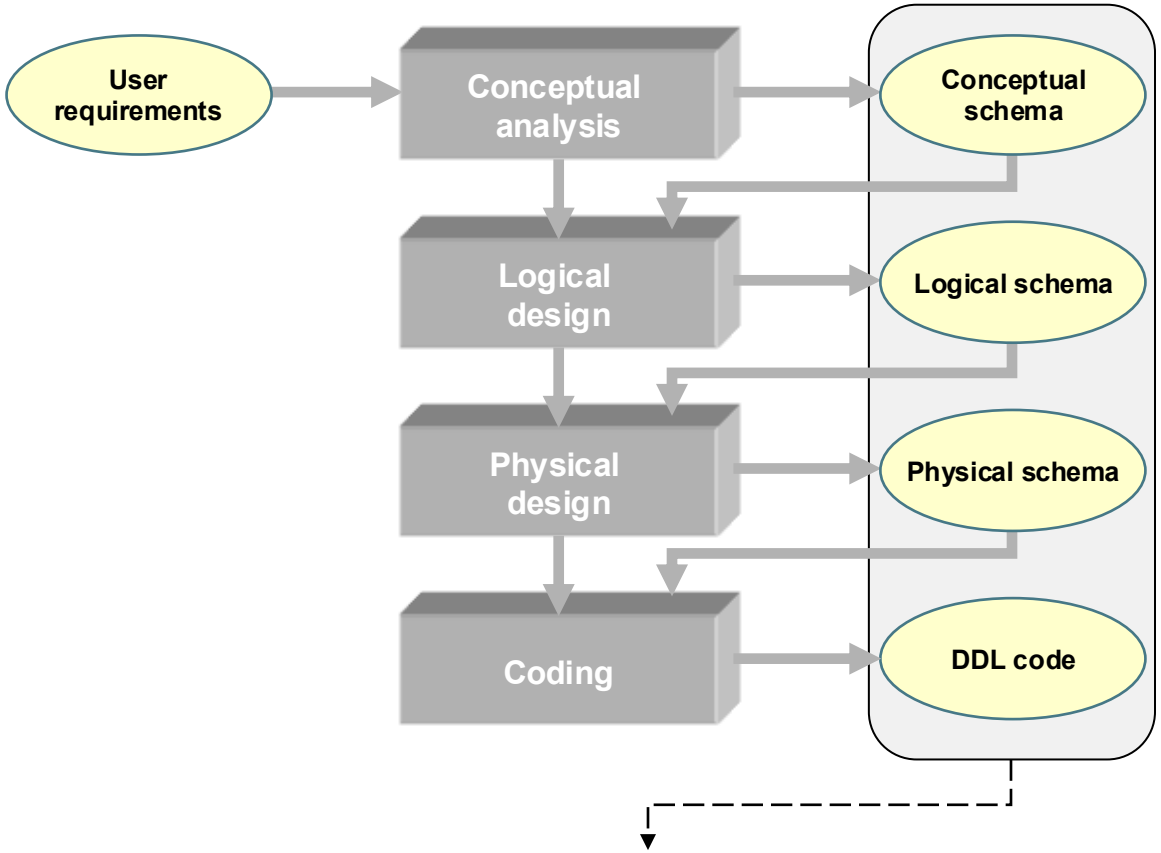
- one (or several) database(s) containing mission-critical ***data***
- complying to a certain ***schema*** (data structures and constraints)
- a set of ***programs*** read/update the data via queries expressed on that schema



There are *<u>several</u>* database schemas at different levels of abstraction

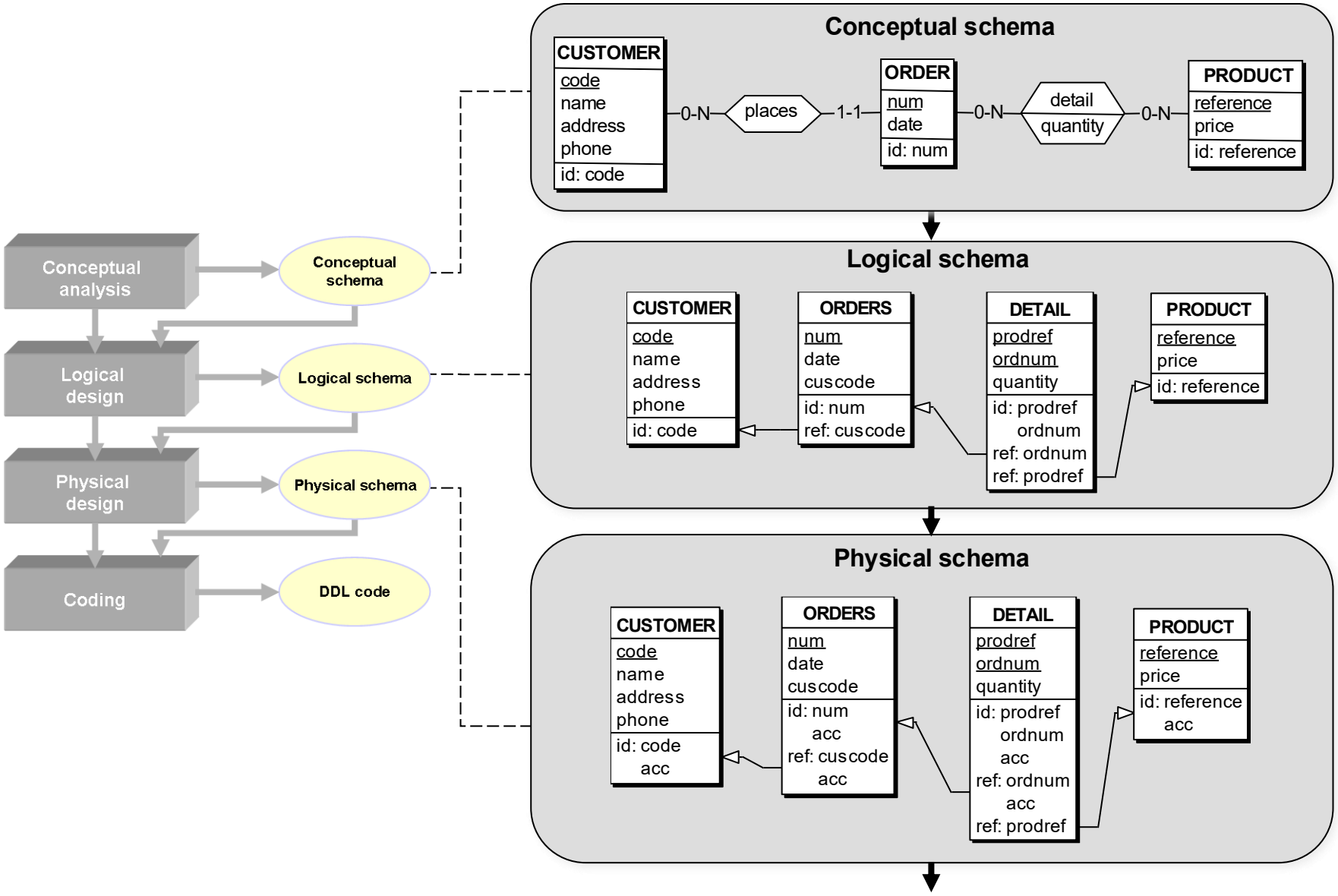
… at least in an ideal world

In an ideal world…

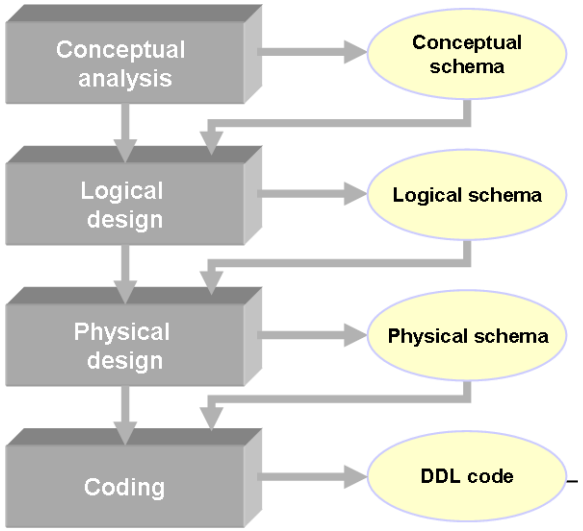


All schemas should form a complete and up-to-date documentation of the database(s)

In an ideal world…



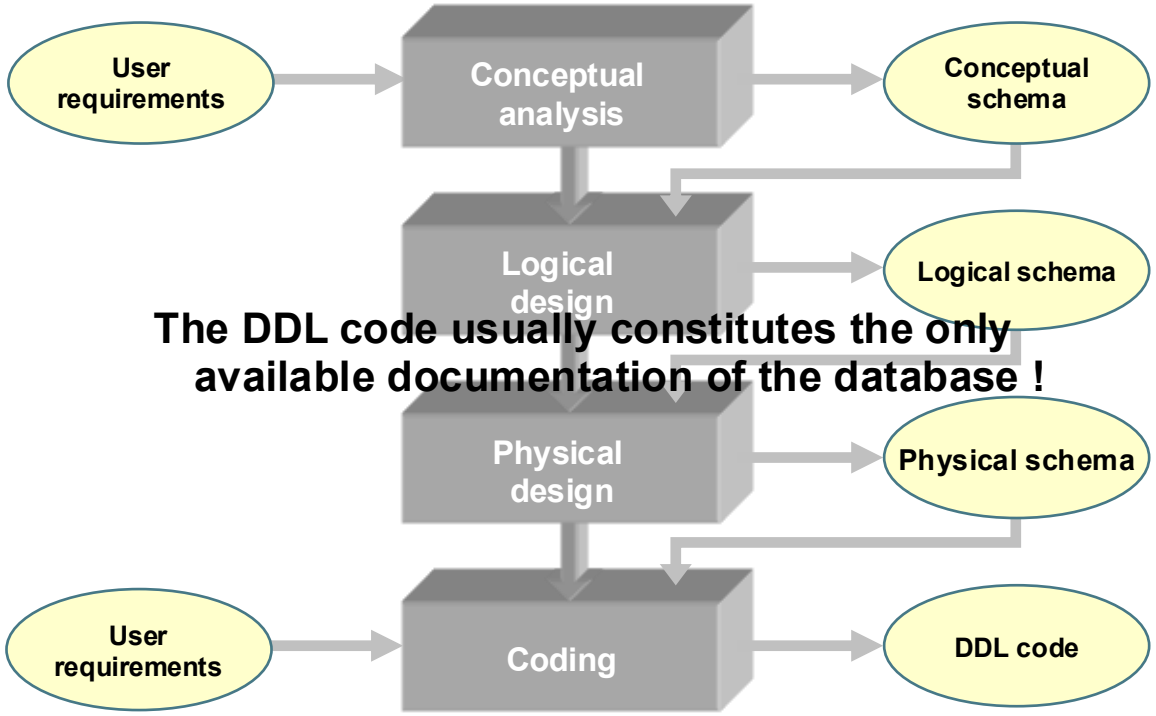
In an ideal world…



DDL code

```
create table CUSTOMER (
    code char(6) not null,
    name char(20) not null,
    address char(40) not null,
    phone numeric(12) not null,
    constraint ID_CUSTOMER primary key (code));
create table DETAIL (
    prodref char(6) not null,
    ordnum char(6) not null,
    quantity numeric(6) not null,
    constraint ID_DETAIL primary key (prodref, ordnum));
create table ORDERS (
    num char(6) not null,
    date date not null,
    cuscode char(6) not null,
    constraint ID_ORDERS primary key (num));
create table PRODUCT (
    reference char(6) not null,
    price numeric(6,2) not null,
    constraint ID_PRODUCT primary key (reference));
alter table DETAIL add constraint REF_DET_ORD_FK
    foreign key (ordnum) references ORDERS;
alter table DETAIL add constraint REF_DET_PRO
    foreign key (prodref) references PRODUCT;
alter table ORDERS add constraint REF_ORD_CUS_FK
    foreign key (cuscode) references CUSTOMER;
create unique index CUSTOMER_IND on CUSTOMER (code);
create unique index DET_IND on DETAIL (prodref, ordnum);
create index DET_ORD_IND on DETAIL (ordnum);
create unique index ORD_IND on ORDERS (num);
create index ORD_CUS_IND on ORDERS (cuscode);
create unique index PRODUCT_IND on PRODUCT (reference);
```

In real life…



The DDL code usually constitutes the only available documentation of the database !

To each problem, a solution

No (sufficient) database documentation?

➔*recover it through* ***Database Reverse Engineering (DBRE)***

« A collection of methods and tools to help an organization determine the structure, function, and meaning of its data » (E. Chikofsky,1996)

DBRE as (valuable) initial phase

- Database maintenance
- Program maintenance
- Database schema evolution
- Database platform migration
- Data conversion
- Database integration/federation
- Data warehousing
- (Big) data analytics
- Machine learning

All those processes require a detailed, up to date documentation of the database(s)

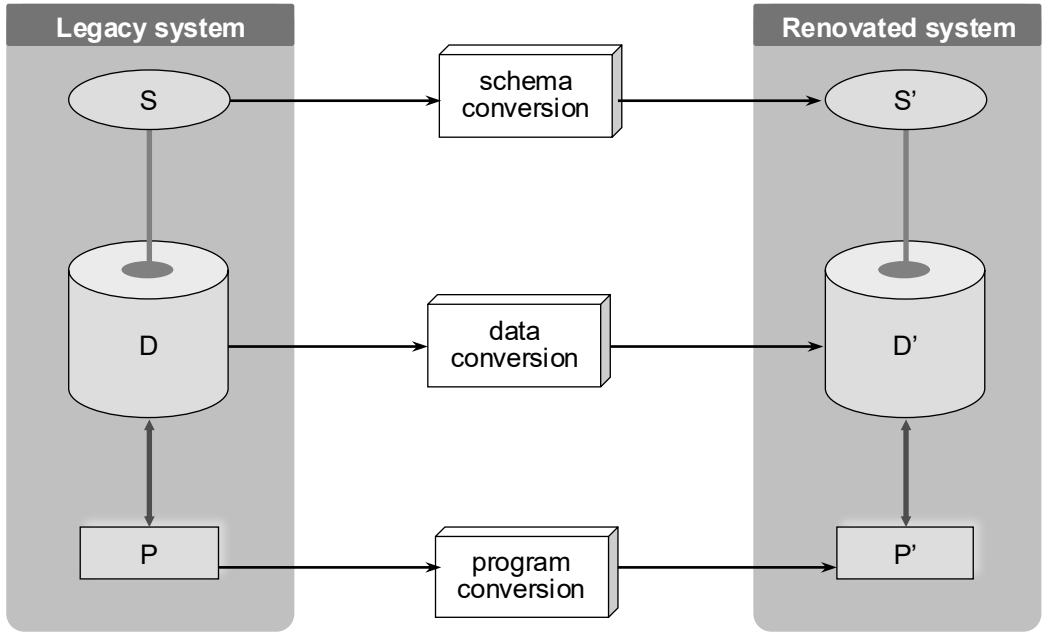
DBRE as (valuable) initial phase

- Database maintenance
- Program maintenance
- Database schema evolution
- **Database platform migration** ← **motivating example**
- Data conversion
- Database integration/federation
- Data warehousing
- (Big) data analytics
- Machine learning

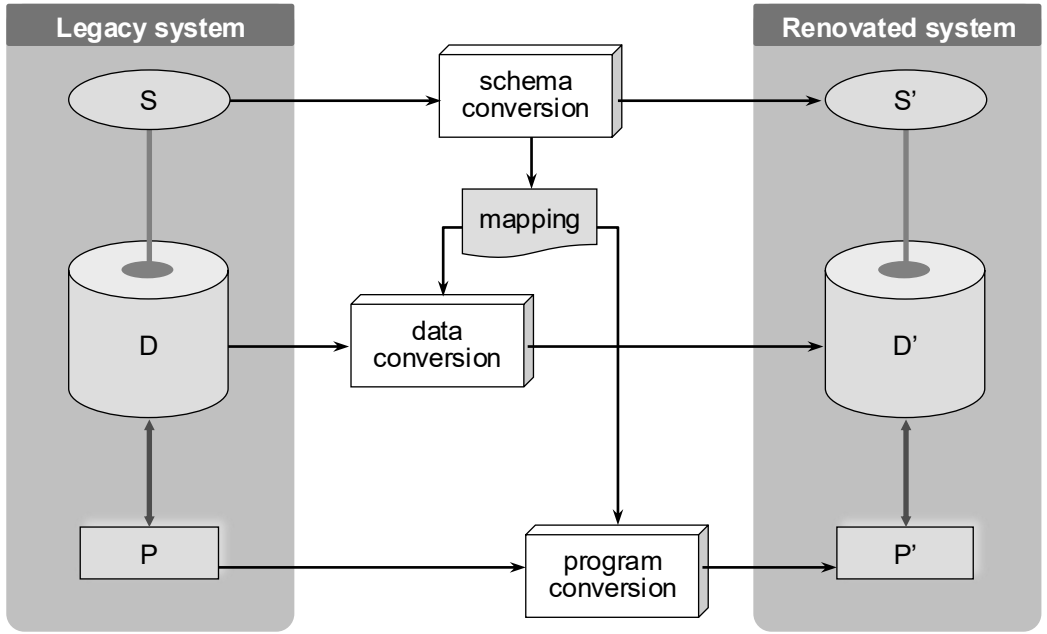
All those processes require an detailed, up to date documentation of the database(s)

Motivation example:
Database plaform migration

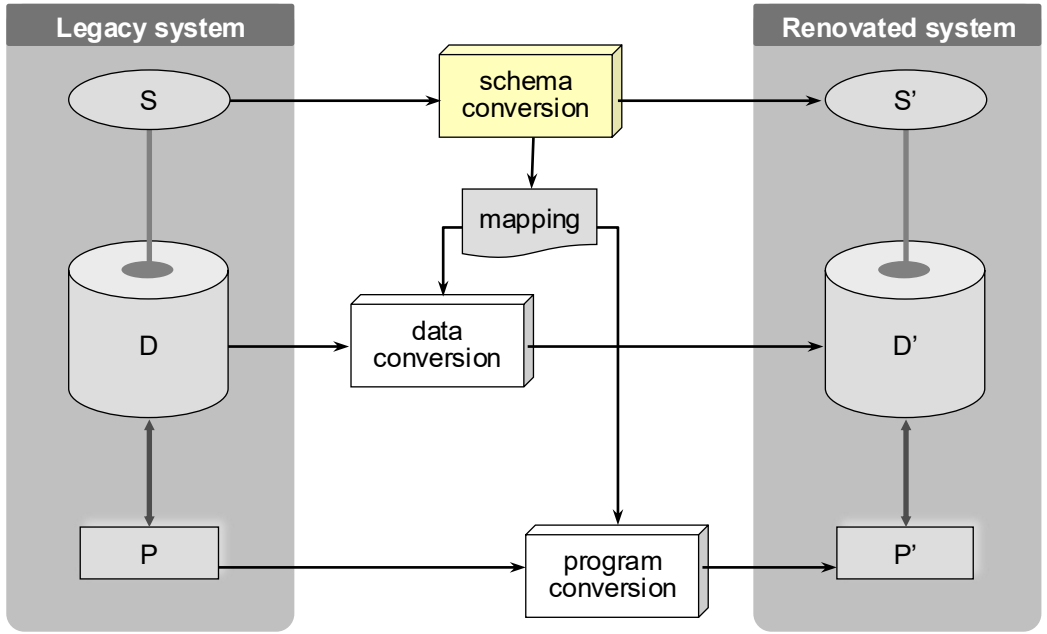
Database platform migration : overview



Database platform migration : overview

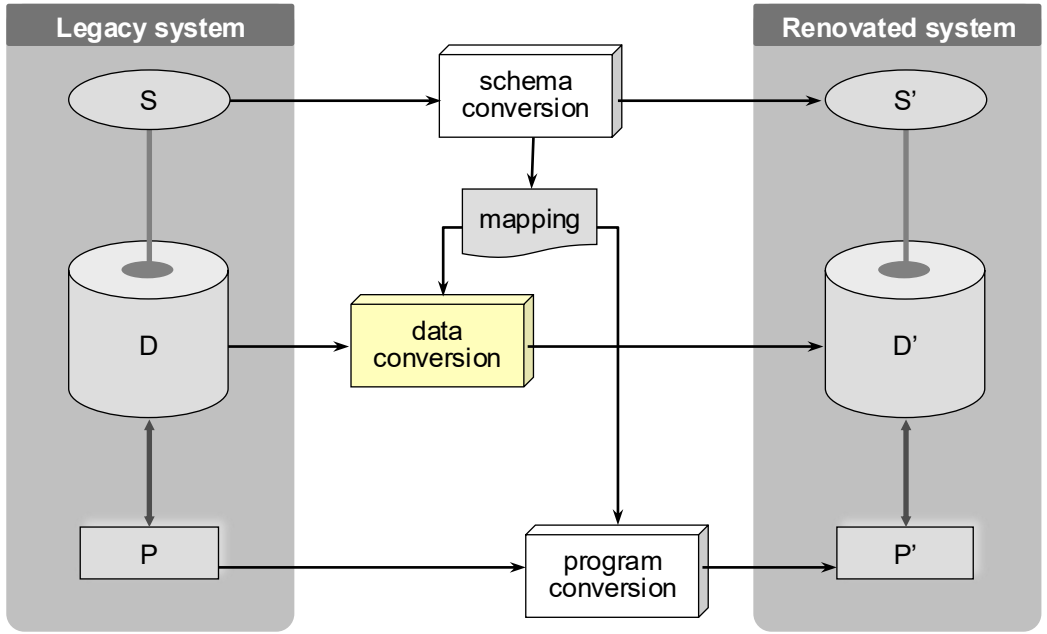


Database platform migration : schema conversion



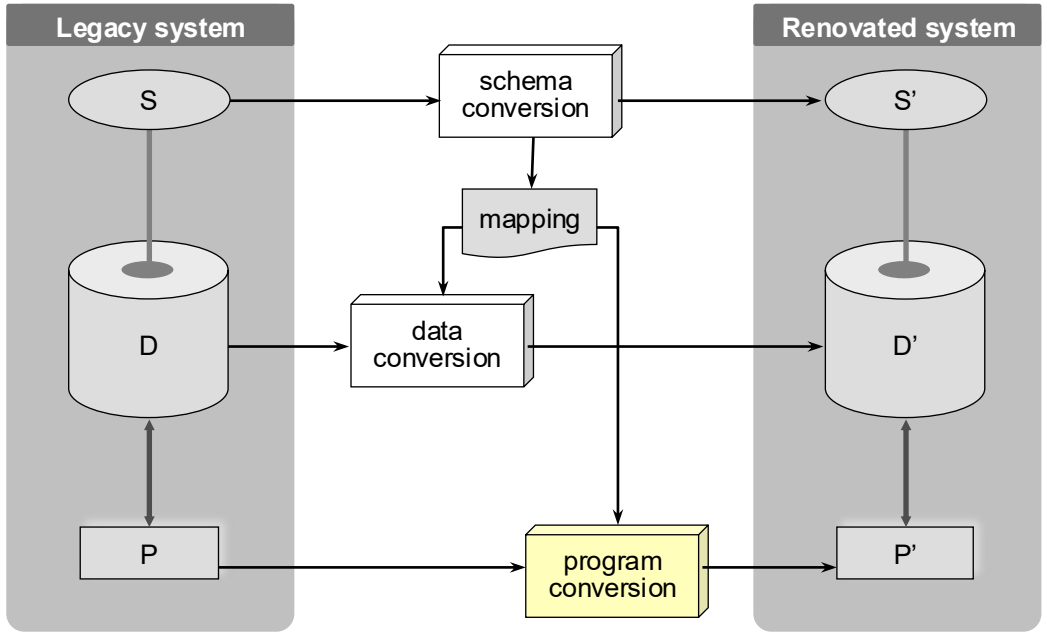
Schema conversion is the translation of the source database structure, or schema, into an equivalent database structure expressed in the target technology

Database platform migration : data conversion



Data conversion is the migration of the data instances from the source database to the target one. This migration involves data transformations that derive from the schema conversion step.

Database platform migration : program conversion

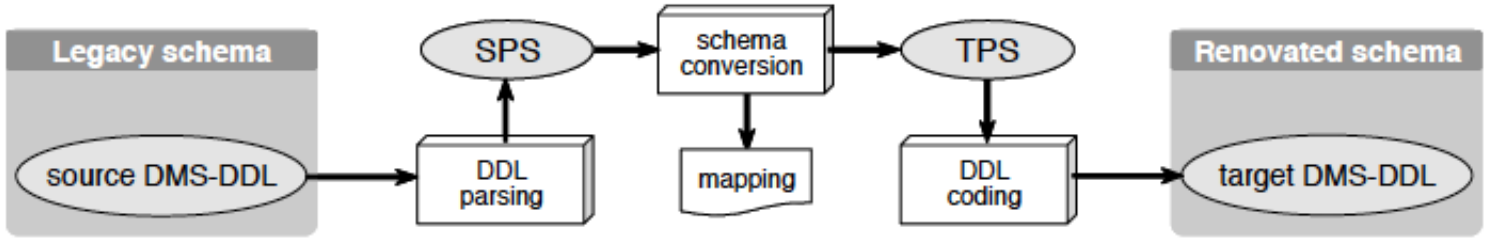


Program conversion is the modification of the programs so that they now access the migrated database instead of the source data. The functionalities of the programs are left unchanged, as well as the programming language and user interface.

One-to-one schema conversion - method

Requires no documentation of the DB – therefore very popular

The ***one-to-one*** schema conversion is the **cheapest** but also the **worst** approach since it may significantly degrade the quality of the system



One-to-one schema conversion – example

```
SELECT CLIENT ASSIGN TO "CUST.DAT"
ORGANIZATION IS INDEXED
RECORD KEY IS CUST_ID.
FD CUST-FILE.
01 CUSTOMER.
   02 CUST-ID PIC X(12).
   02 CUST-INFO PIC X(80).
   02 CUST-HIST PIC X(1000).
```

⇒

CUSTOMER
CUST-ID: char (12)
CUST-INFO: char (80)
CUST-HIST: char (1000)
id: CUST-ID

⇓

CUSTOMER
CUST_ID: char (12)
CUST_INFO: char (80)
CUST_HIST: char (1000)
id: CUST_ID

⇐

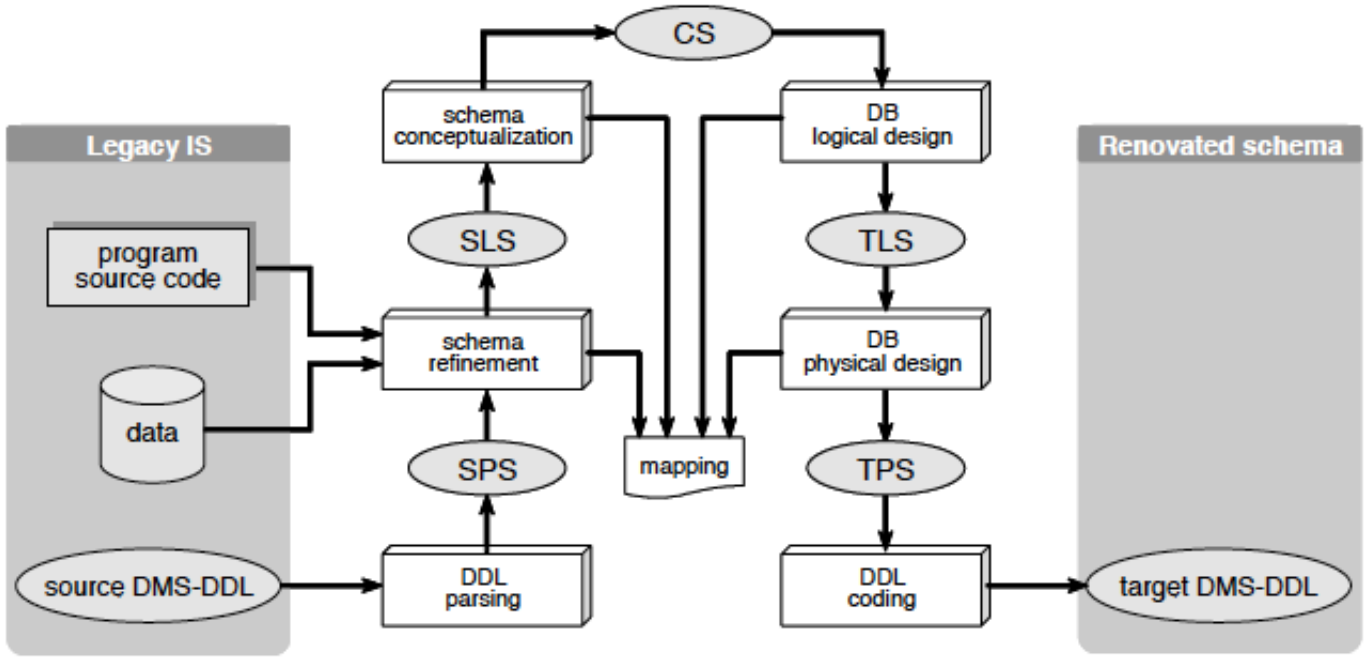
```
create table CUSTOMER(
    CUST_ID char(12) not null,
    CUST_INFO char(80) not null,
    CUST_HIST char(1000) not null,
    primary key (CUST_ID));
```

no added value

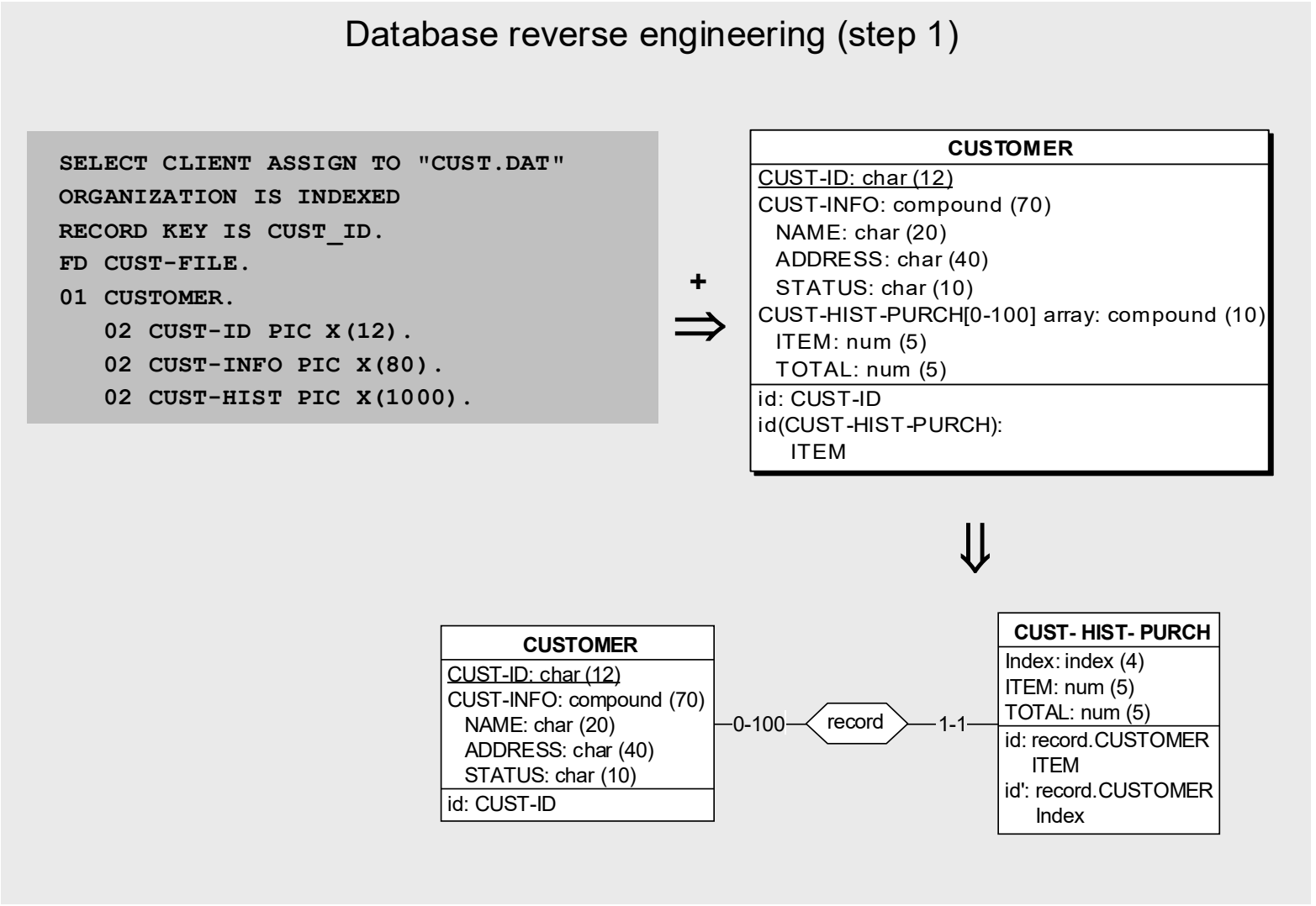
Semantic schema conversion - method

Requires a complete, up to date documentation of the DB

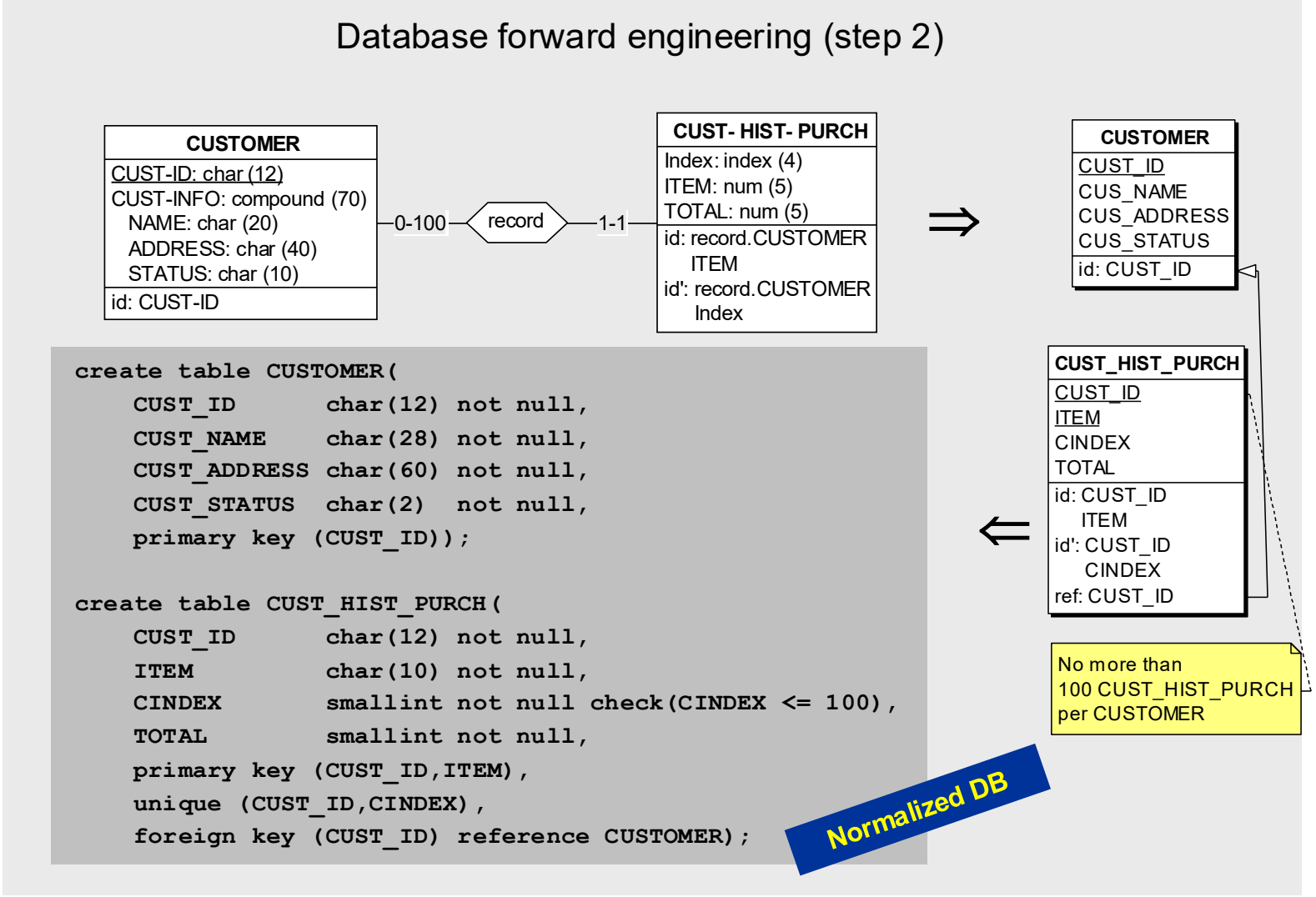
The ***semantic*** schema conversion approach is based on an in-depth understanding of the data structures. It produces a high quality target schema, strong basis for the future.



Semantic schema conversion – example



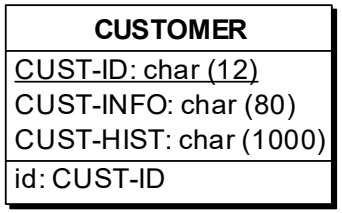
Semantic schema conversion – example




```
create table CUSTOMER(
    CUST_ID      char(12) not null,
    CUST_NAME    char(28) not null,
    CUST_ADDRESS char(60) not null,
    CUST_STATUS  char(2)  not null,
    primary key (CUST_ID));

create table CUST_HIST_PURCH(
    CUST_ID      char(12) not null,
    ITEM         char(10) not null,
    CINDEX       smallint not null check(CINDEX <= 100),
    TOTAL        smallint not null,
    primary key (CUST_ID,ITEM),
    unique (CUST_ID,CINDEX),
    foreign key (CUST_ID) reference CUSTOMER);
```

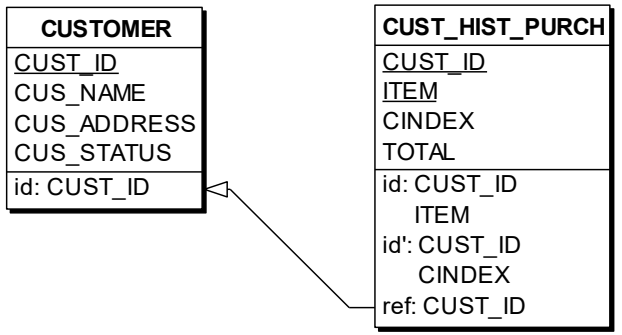
One-to-one versus semantic schema conversion

One-to-one conversion




```
create table CUSTOMER(
    CUST_ID char(12) not null,
    CUST_INFO char(80) not null,
    CUST_HIST char(1000) not null,
    primary key (CUST_ID));
```

Semantic conversion

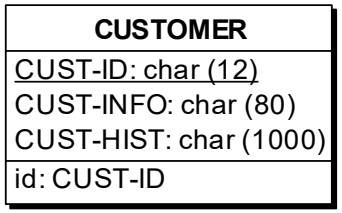



```
create table CUSTOMER(
    CUST_ID      char(12) not null,
    CUST_NAME    char(28) not null,
    CUST_ADDRESS char(60) not null,
    CUST_STATUS  char(2)  not null,
    primary key (CUST_ID));

create table CUST_HIST_PURCH(
    CUST_ID      char(12) not null,
    ITEM         char(10) not null,
    CINDEX       smallint not null check(CINDEX <= 100),
    TOTAL        smallint not null,
    primary key (CUST_ID,ITEM),
    unique (CUST_ID,CINDEX),
    foreign key (CUST_ID) reference CUSTOMER);
```

One-to-one versus semantic schema conversion

What if we need for a new functionality: ***compute total sales per item***

One-to-one conversion

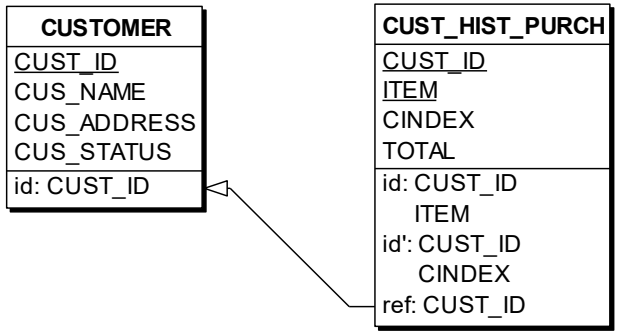


⇓

?

- **where is the required information?**
- **how to extract it from the CUSTOMER table?**
- **who will develop the (C, Java, Python) program?**
- **… and when?**

Semantic conversion



⇓

```
select ITEM, sum(TOTAL)
from   CUST_HIST_PURCH
group  by ITEM;
```

- **clearly visible + documentation if needed**
- **just name the columns**
- **by any non expert**
- **immediately, 2 minutes**

Database reverse engineering:
An industrial project

DBRE – Industrial project

- **Context:** Belgian car distributor - Spare parts management application
- **Objective:** migrating an **IDMS** database to a **RDBMS**
- **Size:** medium size database (324 record types)

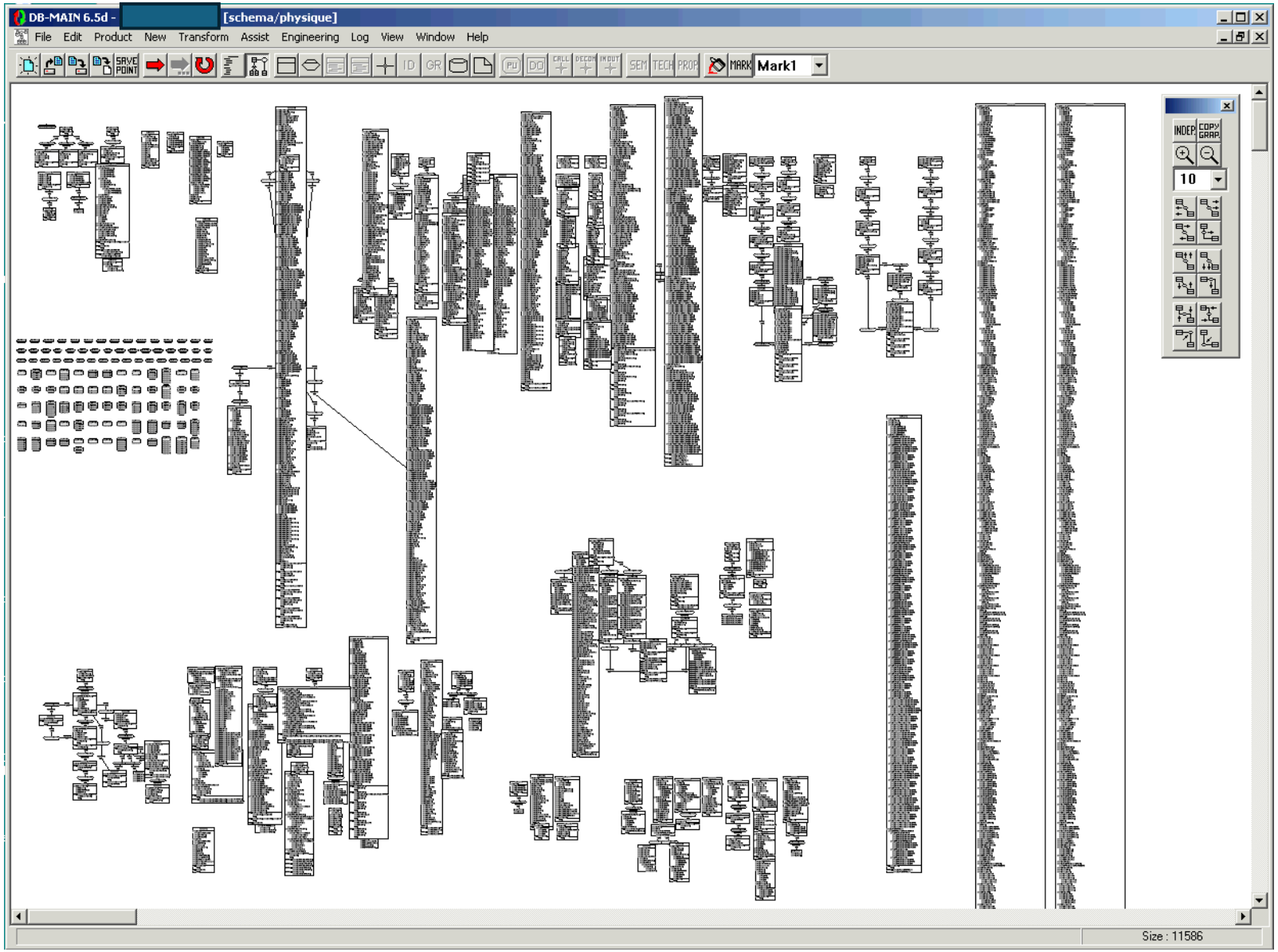
First trial : **one-to-one conversion**, outsourced

Result : **failure**, poor data access performance, unreadable code,
the migrated system could not be maintained any longer…

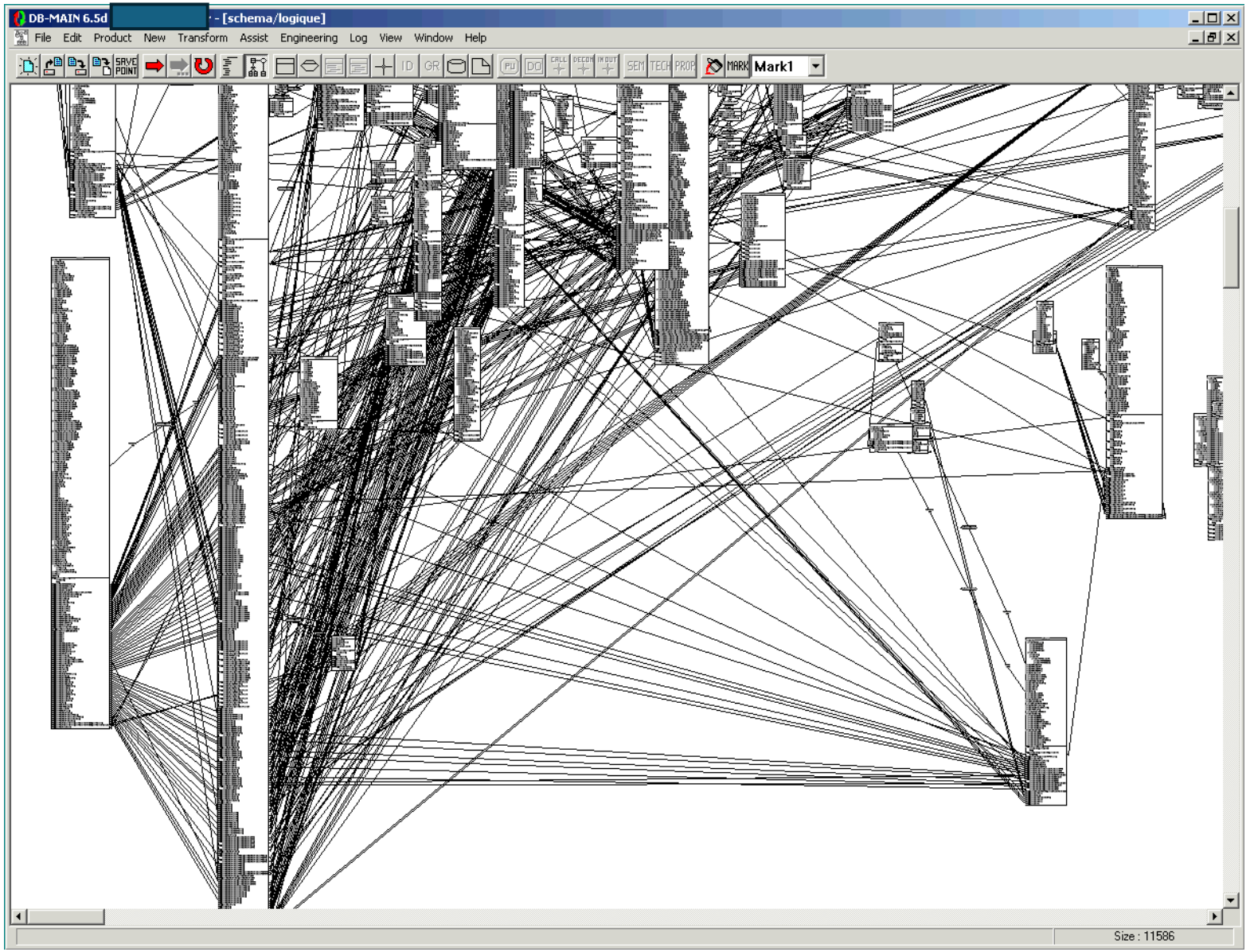
Second trial : **semantic conversion**, in house (with UNamur researchers)

Result : **success**; satisfying performance, maintainable result.
the project leader received the "*IT manager of the year*" award !

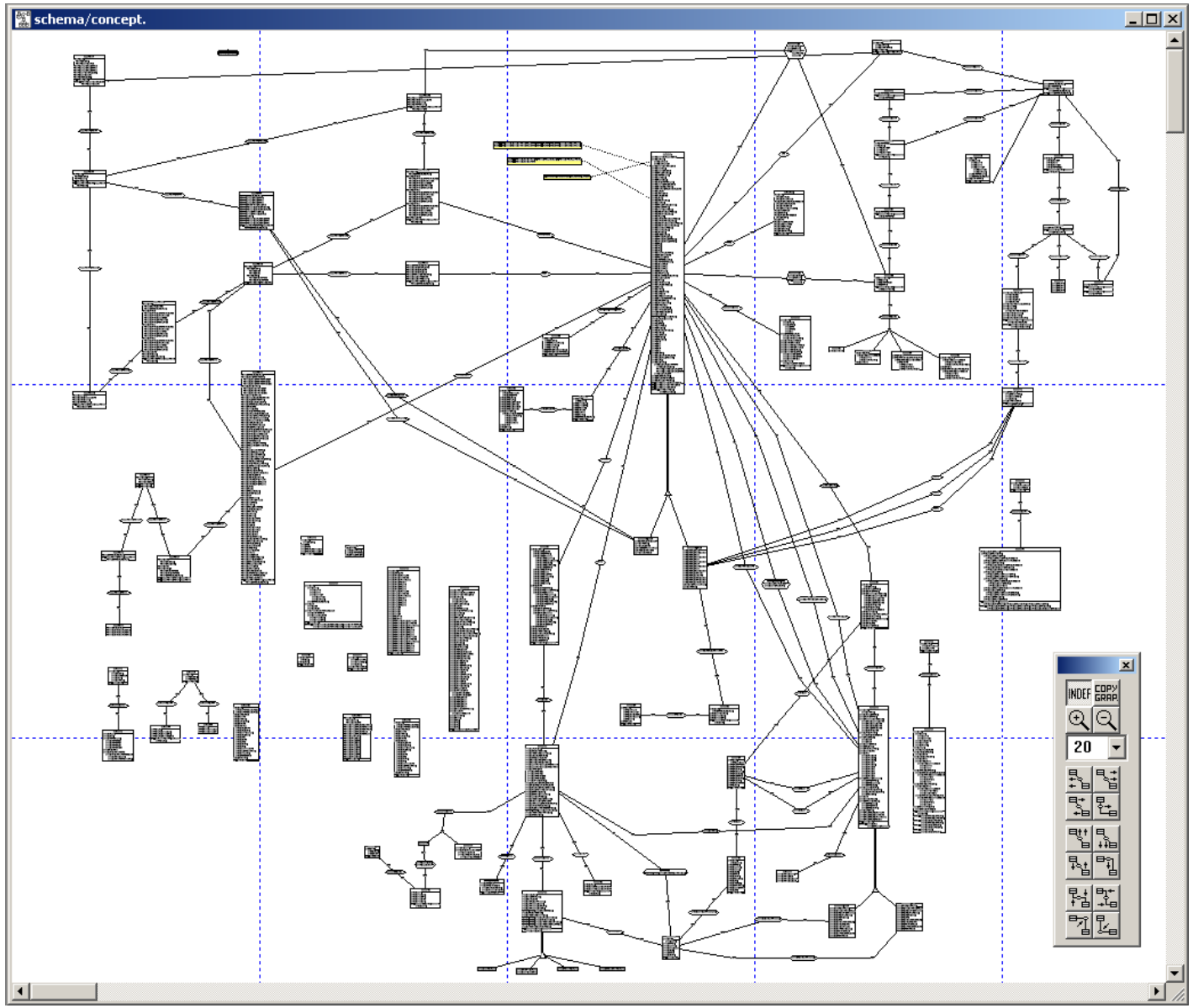
DBRE – Physical schema (IDMS) - *Excerpts*



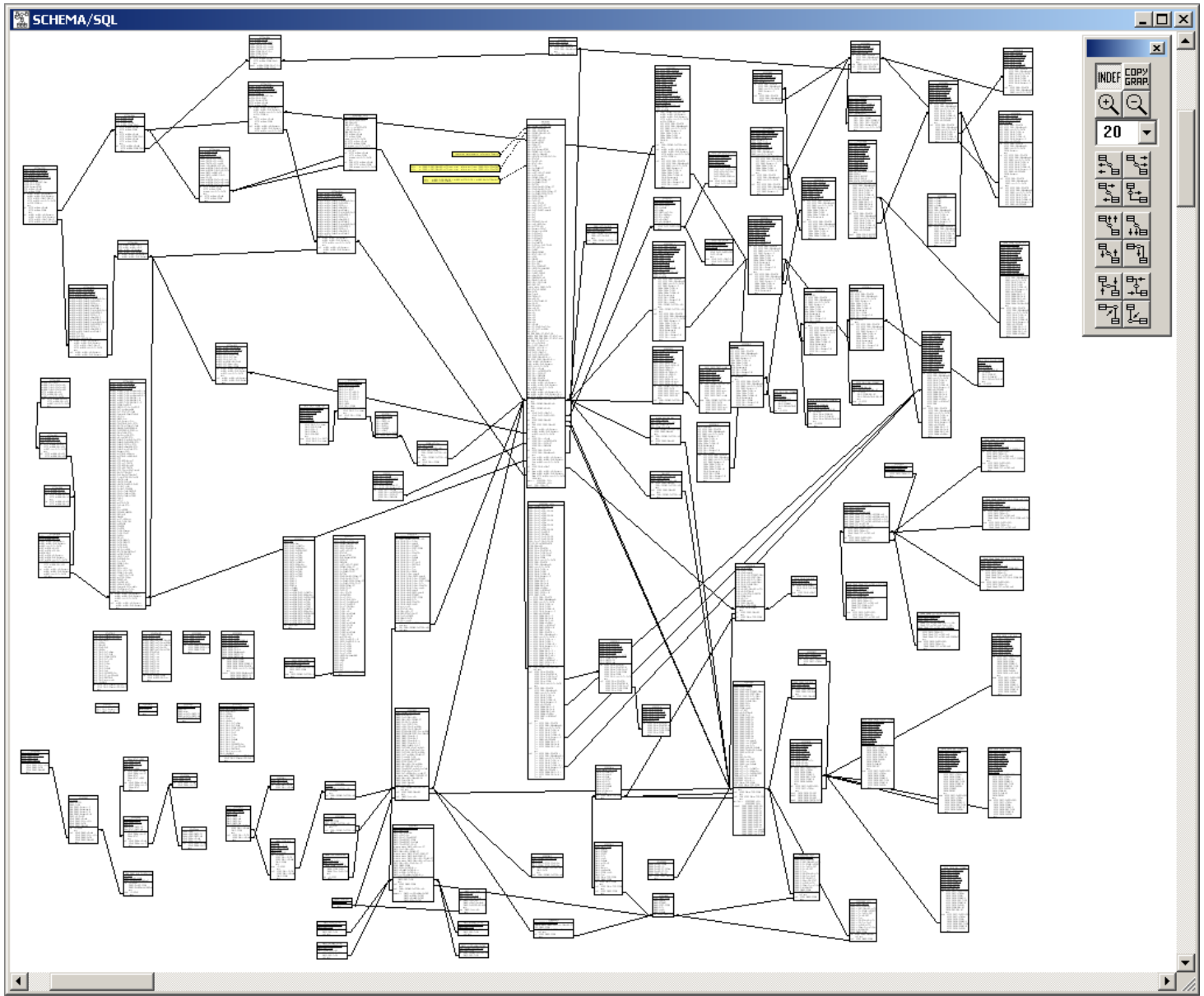
DBRE - (Raw) Logical schema (IDMS) - *Excerpts*



DBRE – Conceptual schema (IDMS) - *Excerpts*



DBRE – Logical schema (relational) - *Excerpts*

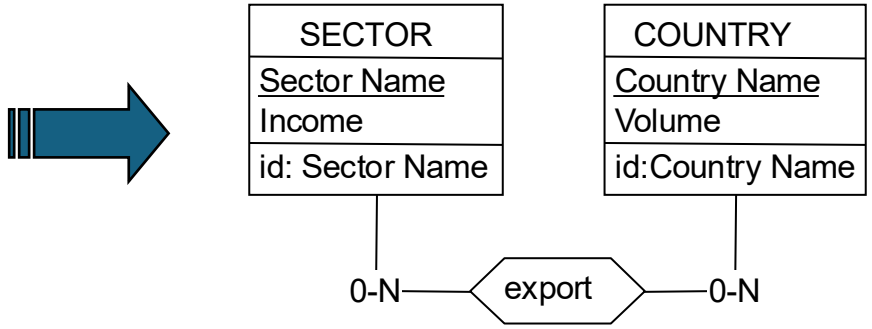


Database reverse engineering:
A DMS-independent method

Database reverse engineering

the ideal view

```
create table COUNTRY (
   CNAME char(24) not null,
   VOLUME numeric(12) not null,
   primary key (CNAME));

create table EXPORT (
   CNAME char(24) not null,
   SNAME char(18) not null,
   primary key (CNAME, SNAME),
   foreign key (CNAME)
       references COUNTRY,
   foreign key (SNAME)
       references SECTOR);

create table SECTOR (
   SNAME char(18) not null,
   INCOME numeric(14) not null,
   primary key (SNAME));
```



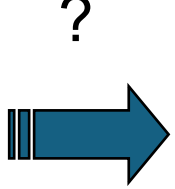
Database reverse engineering

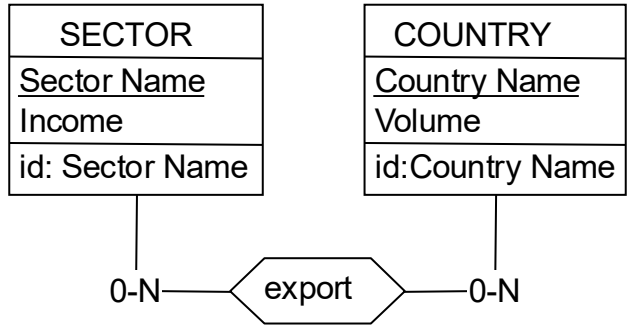
in reality

```
create table TBL_C (
   IDC char(24) not null,
   COL_2 numeric(12) not null);

create table TBL_X (
   IDX char(42) not null);

create table TBL_S (
   IDS char(18) not null,
   COL_2 numeric(14) not null);

create unique index ID_C on TBL_C(IDC);
create unique index ID_X on TBL_X(IDX);
```

implicit constructs

?





DBRE aims at recovering ***implicit schema constructs***, including :

- finer-grained structures of entity types and attributes
- referential constraints (foreign keys)
- exact cardinalities of attributes
- identifiers of multi-valued attributes

Database reverse engineering

Where to find the (*implicit)* data structures and constraints?

- DDL analysis or catalog extraction (+/- 40%)
- Schema analysis (+/- 10%)
- Data analysis (+/- 10%)
- HMI analysis (+/- 10%)
- Program analysis (+/- 30%)

Approximate percentages (based on experience)

Database reverse engineering

Schema analysis

M. H. Alalfi, J. R. Cordy, and T. R. Dean, SQL2XMI: Reverse engineering of UML-ER diagrams from relational database schemas. In *Proc. of WCRE 2008*, pp. 187–191, IEEE, 2008.

HMI analysis

Ravi Ramdoyal, Anthony Cleve, and Jean-Luc Hainaut. Reverse engineering user interfaces for interactive database conceptual analysis. In Proc. of CAiSE'10, volume 6051 of LNCS. Springer, 2010.

James F. Terwilliger et al. The user interface is the conceptual model. In Proc. of ER'06, volume 4215 of LNCS, pages 424–436. Springer, 2006.

Static program analysis

A. Cleve, J. Henrard, and J.-L. Hainaut, Data reverse engineering using system dependency graphs, in *Proc. of WCRE 2006*, pp. 157–166, IEEE, 2006.

C. Marinescu, Discovering the objectual meaning of foreign key constraints in enterprise applications, in *Proc. of WCRE 2007*, pp. 100–109, IEEE, 2006

Dynamic program analysis

A. Cleve, J.-L. Hainaut, Dynamic Analysis of SQL Statements for Data-Intensive Applications Reverse Engineering. In *Proc. of WCRE 2008*, pp. 192-196, IEEE, 2008.

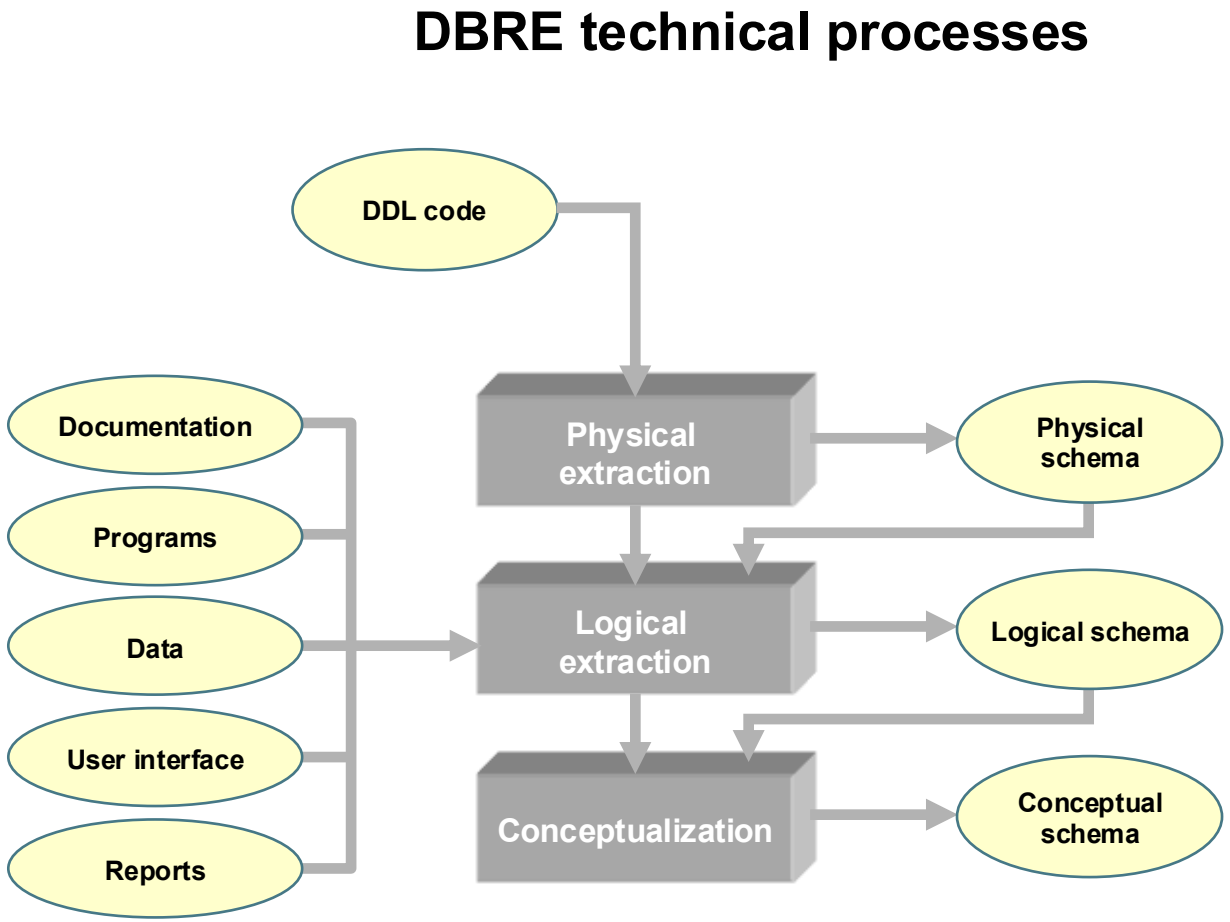
A. Cleve, N. Noughi, and J.-L. Hainaut, Dynamic program analysis for database reverse engineering, in *post-Proc. of GTTSE 2011*, Springer, 2011.

Database reverse engineering

What are the challenges?

The DDL code (or the data dictionary) gives us the **explicit** data structures and constraints	⇒	**Physical extraction**
Many undeclared, therefore hidden, **implicit** data structures and constraints	⇒	**Logical extraction**
Many complex, **non-standard** constructs - difficult to interpret	⇒	**Conceptualization**

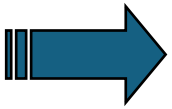
DBRE - Method

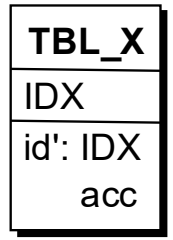

DBRE technical processes
DDL code
Documentation
Programs
Data
User interface
Reports
Physical extraction
Logical extraction
Conceptualization
Physical schema
Logical schema
Conceptual schema

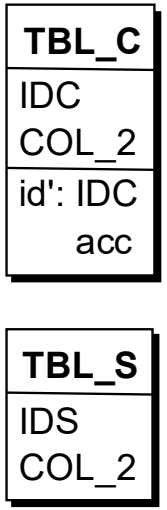
DBRE - Method

Step 1: Physical extraction

```
create table TBL_C (
   IDC char(24) not null,
   COL_2 numeric(12) not null);

create table TBL_X (
   IDX char(42) not null);

create table TBL_S (
   IDS char(18) not null,
   COL_2 numeric(14) not null);

create unique index ID_C on
TBL_C(IDC);
create unique index ID_X on
TBL_X(IDX);
```

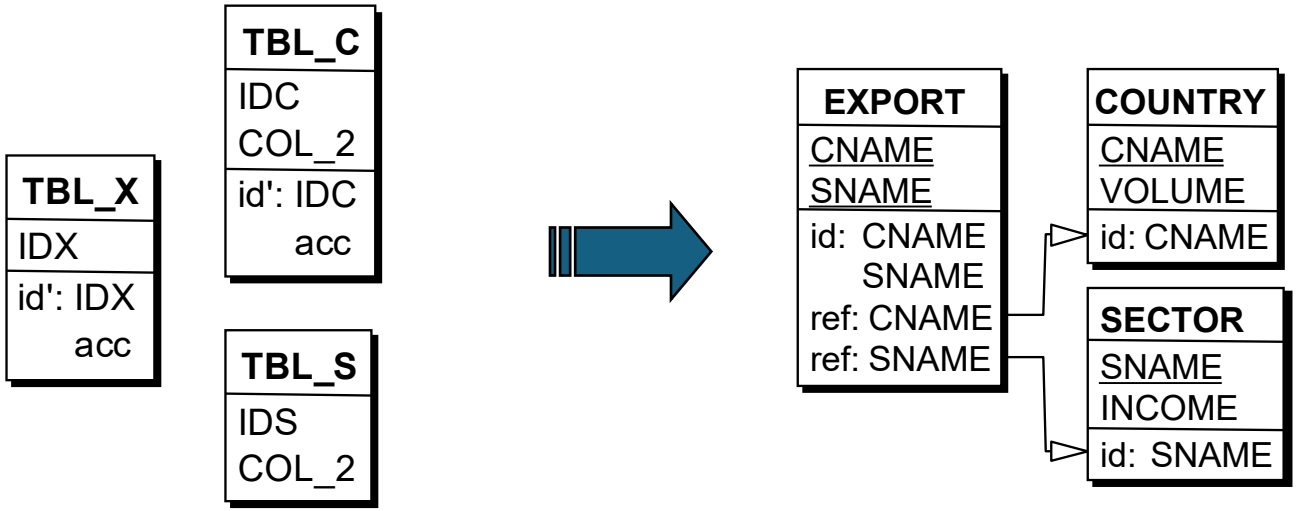






DBRE - Method

Step 2: Logical extraction



cryptic names

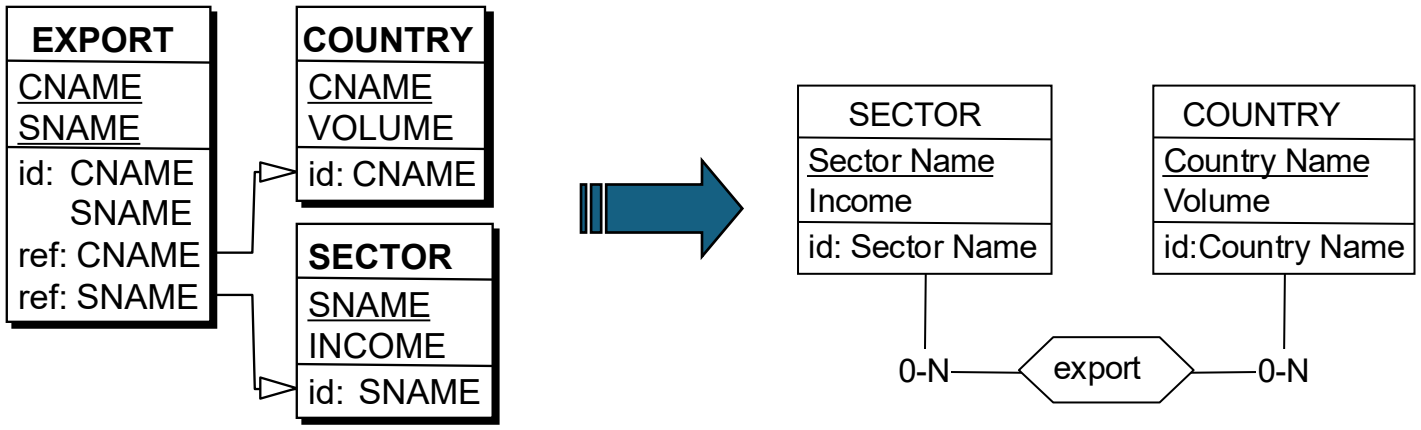
missing unique keys

missing foreign keys

concatenated columns

DBRE - Method

Step 3: Conceptualization



technology-dependent constructs

unclear semantic interpretation

DBRE - Method

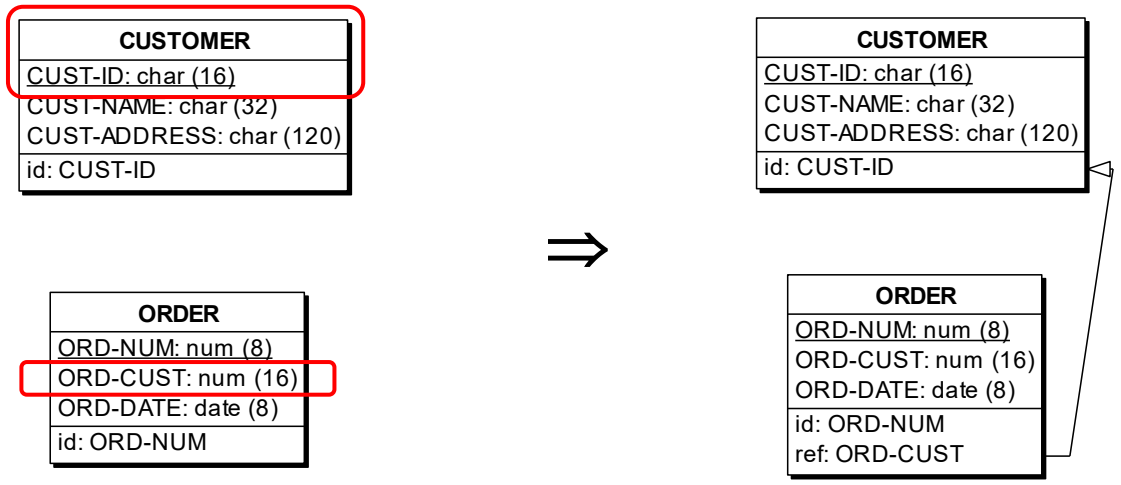
Analysis techniques for Logical extraction

1. Schema analysis
2. Data analysis
3. Static program analysis
4. Dynamic program analysis

DBRE - Method

Analysis techniques for Logical extraction

1. Schema analysis

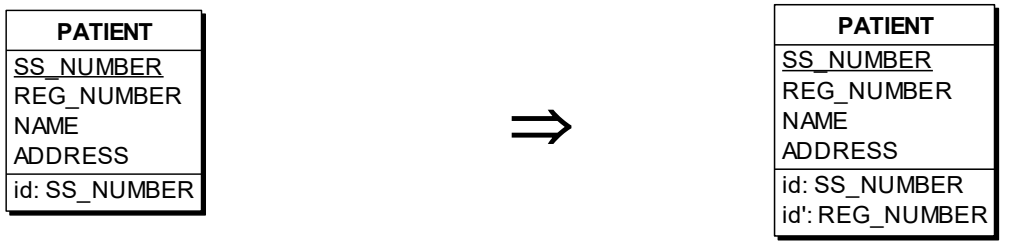


One of the main (only) heuristics in the 80's

DBRE - Method

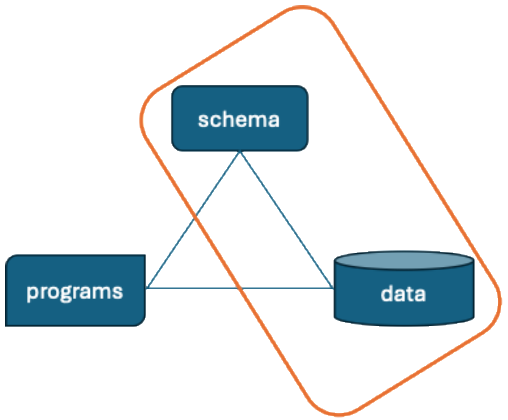
Analysis techniques for Logical extraction

2. Data analysis




```
select REG_NUMBER, count(*) as N
from   PATIENT
group by REG_NUMBER
having count(*) > 1
```

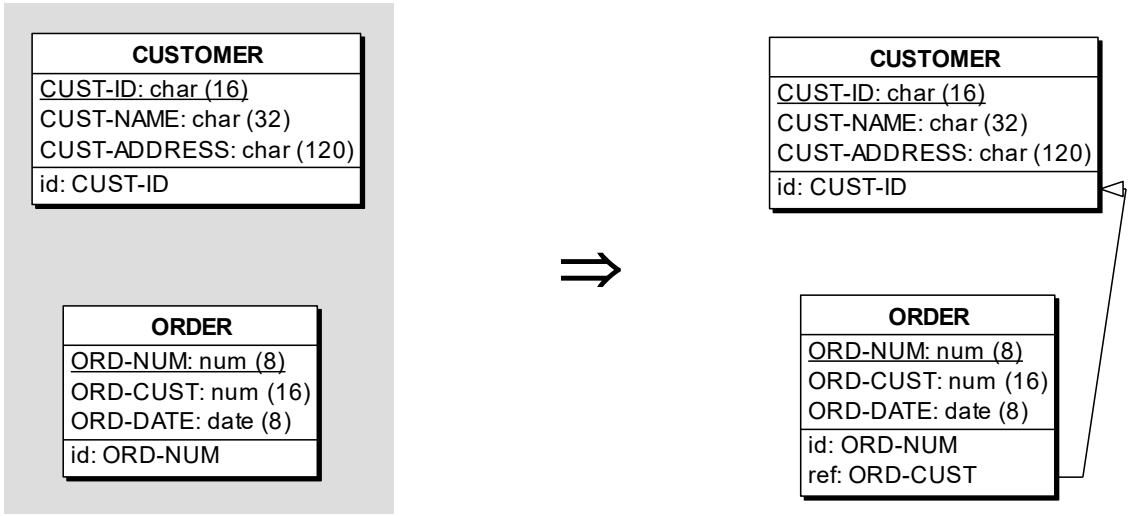
```
REG_NUMBER |   N
-----------|------
           |
```



DBRE - Method

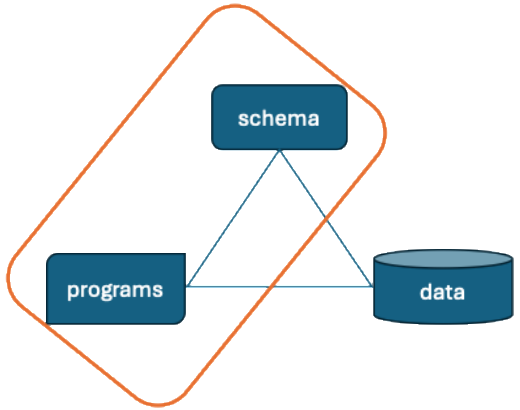
Analysis techniques for Logical extraction

3. Static program analysis (intra-query analysis)



\+

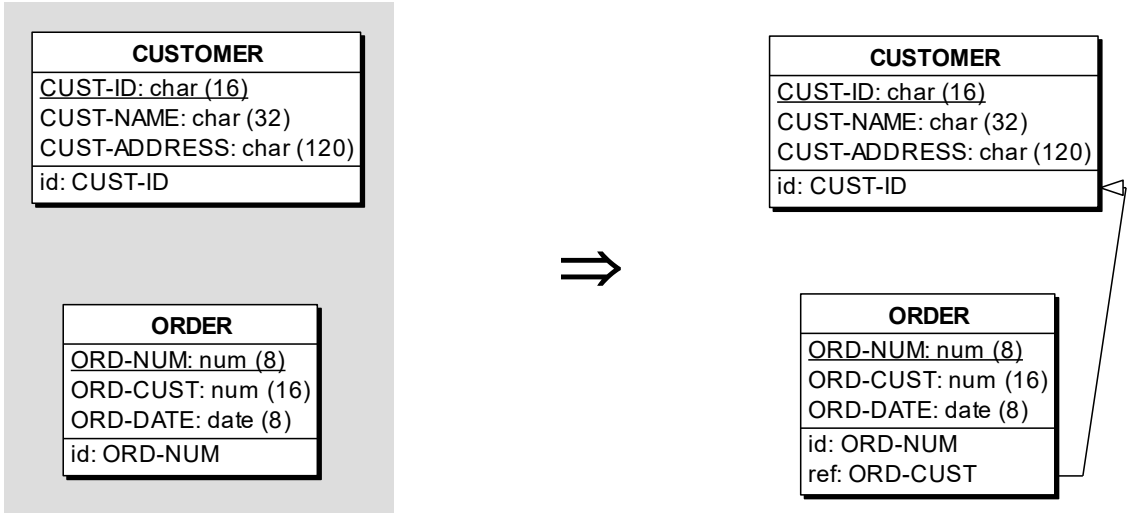
```
select *
from  CUSTOMER, ORDER
where CUST_ID = ORD_CUST
```

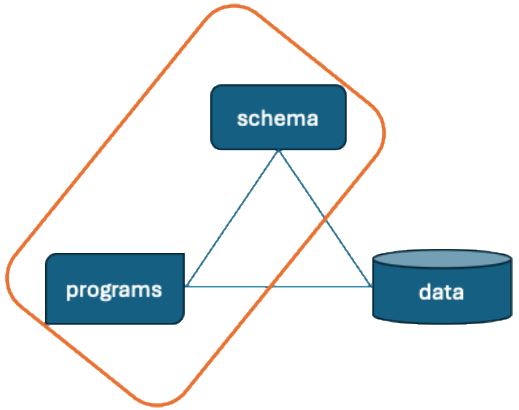


DBRE - Method

Analysis techniques for Logical extraction

3. Static program analysis (inter-query analysis)

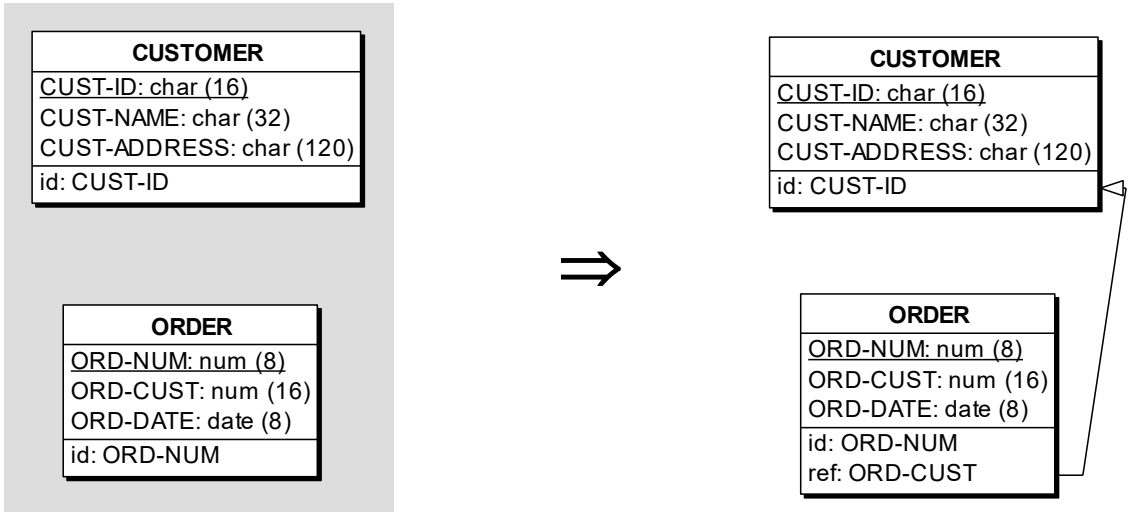




DBRE - Method

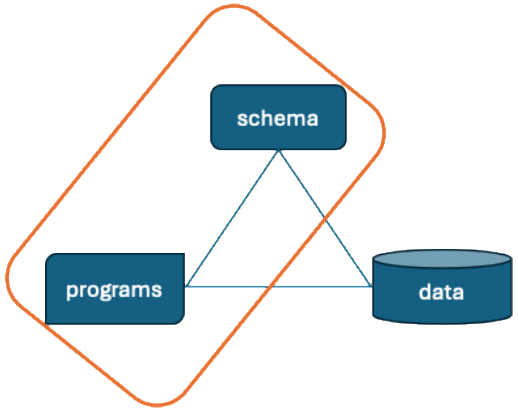
Analysis techniques for Logical extraction

3. Static program analysis (inter-query analysis)



\+

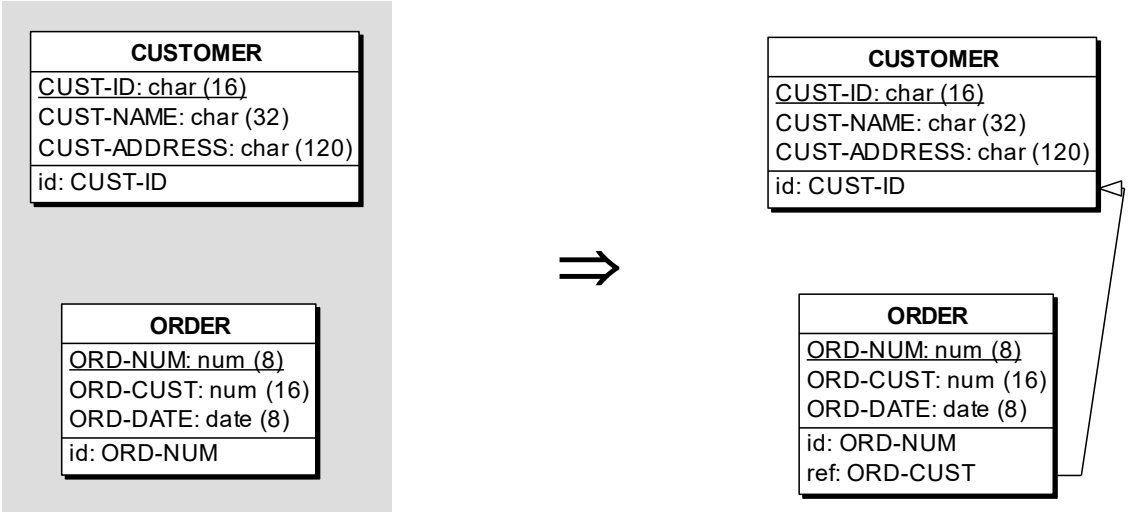
```
select ORD_CUST into :OC from ORDER
where  ORD_NUM = :ON;
C = "" + OC;
insert into CUSTOMER values(:C,'?','?');
```



DBRE - Method

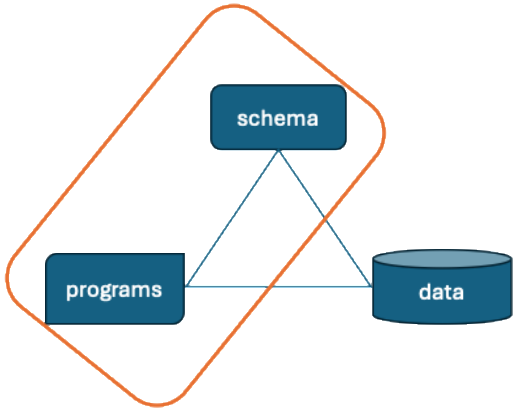
Analysis techniques for Logical extraction

4. Dynamic program analysis (FK usage)



+

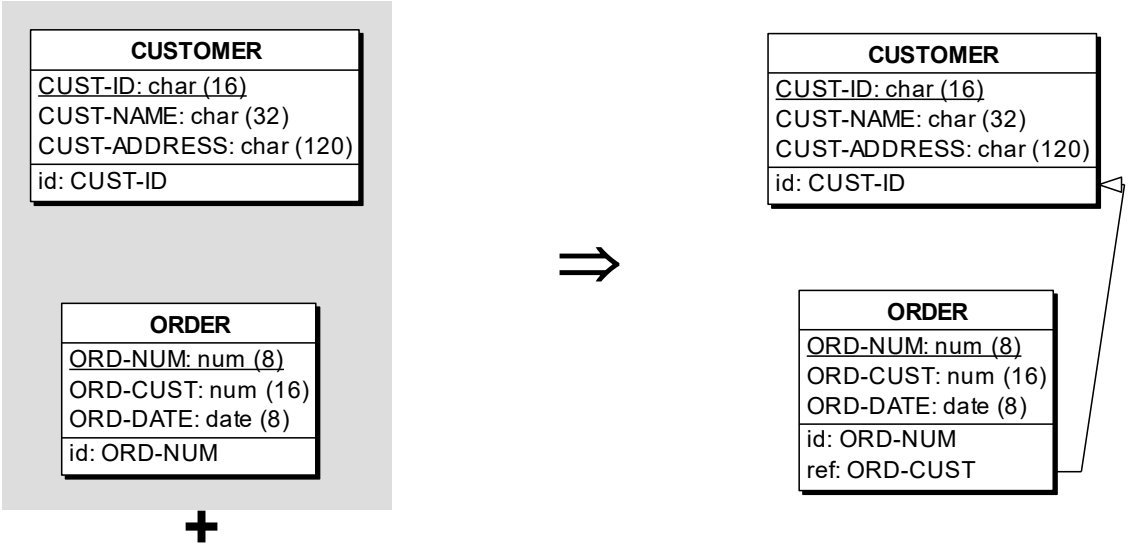
```
select ORD_CUST from ORDER where ORD_DATE = '2008-20-06'
getString(1) = C400
select CUST_NAME, CUST_ADDRESS from CUSTOMER where CUST_ID = 'C400'
getString(1) = B314
select CUST_NAME, CUST_ADDRESS from CUSTOMERS where CUST_ID = 'B314'
getString(1) = C891
select select CUST_NAME, CUST_ADDRESS from CUSTOMERS where CUST_ID = 'C891'
```



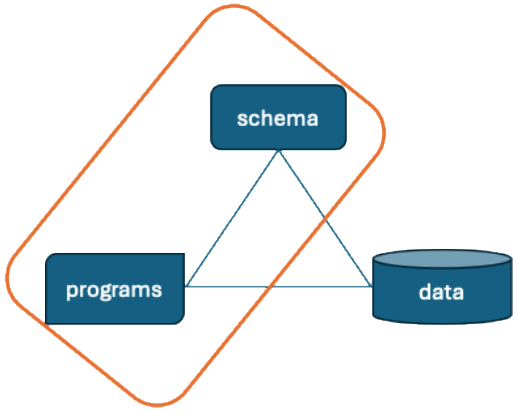
DBRE - Method

Analysis techniques for Logical extraction

4. Dynamic program analysis (FK validation)




```
select count(*) from CUSTOMER where CUST_ID = 'C400'
getInt(1) = 1
insert into ORDERS(ORD_NUM, ORD_CUST, ORD_DATE) values (456, 'C400', '2008-06-20')
select count(*) from CUSTOMER where CUST_ID = 'C152'
getInt(1) = 0
select count(*) from CUSTOMER where CUST_ID = 'C251'
getInt(1) = 1
insert into ORDERS(ORD_NUM, ORD_CUST, ORD_DATE) values (457, 'C251','2008-06-20')
```



Dynamic program analysis for DBRE:
An open-source case study

Dynamic analysis for DBRE – A case study

- Based on http://webcampus.fundp.ac.be
  - E-learning web application
  - Based on open-source project Claroline
  - 465 000 lines of code (PHP/MySQL)

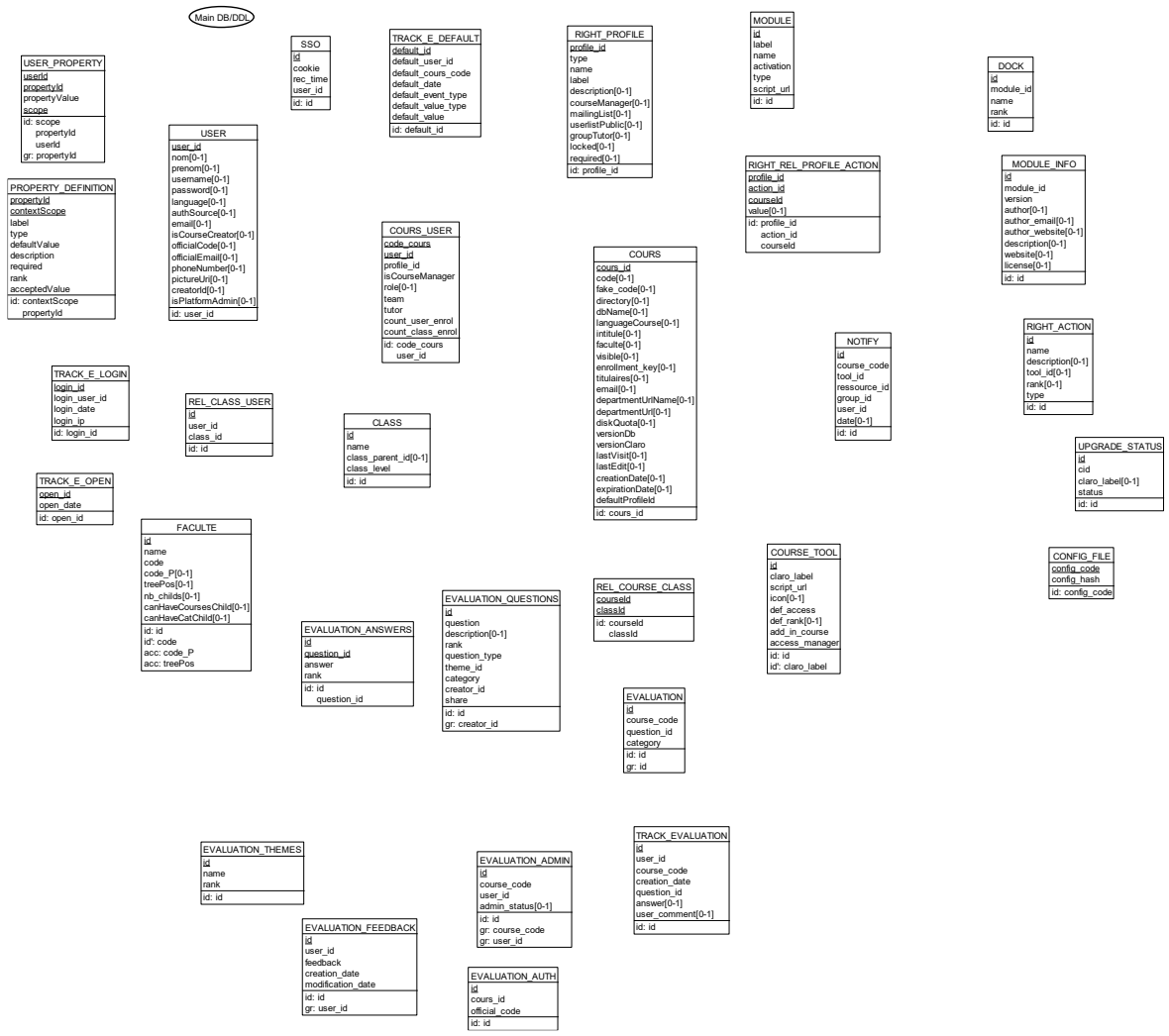
- Dynamic program analysis
  - Semi-automatic source code instrumentation
  - Capturing query executions & query results

- 14 execution scenarios
  - register as a student/as an administrator
  - create a course
  - delete a course
  - delete a course category (faculty)
  - add a course plugin
  - …

Dynamic analysis for DBRE – A case study

DDL code (main DB)

- 33 tables
- 198 columns



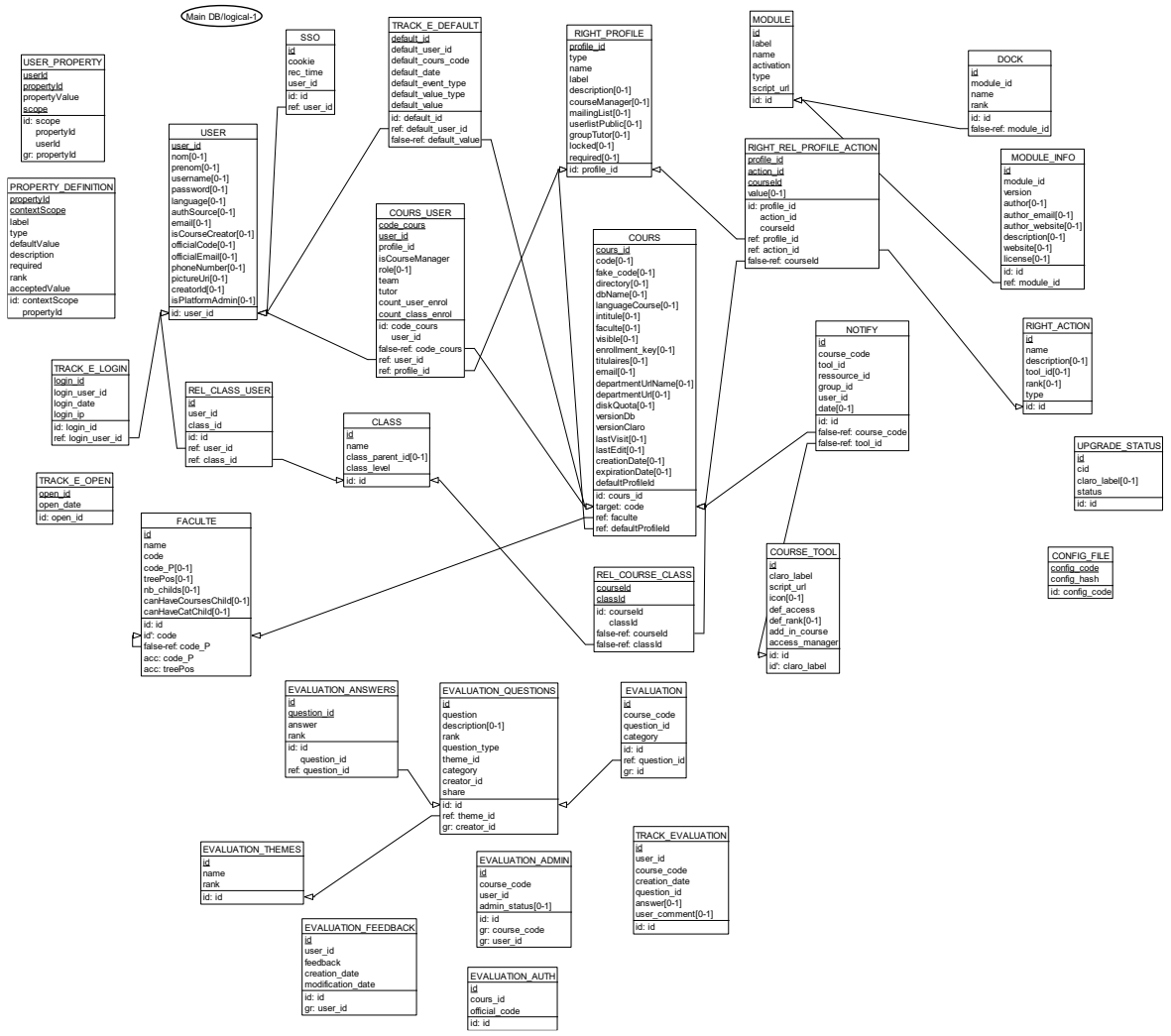
Dynamic analysis for DBRE – A case study

Complete logical schema

- 33 tables
- 198 columns

+ 35 implicit foreign keys

(known by the programmers)



Dynamic analysis for DBRE – A case study

Table 1. Size of collected SQL traces, by execution scenario.

Execution scenario	# of queries	# of `select`	# of `insert`	# of `delete`	# of `update`	# of query results
register user	27	24	3	0	0	163
add course manager	194	190	4	0	0	2 391
add course user	155	151	4	0	0	1 908
create course	29	20	9	0	0	299
delete course	132	123	1	7	0	1 700
delete course user	84	83	0	1	0	996
delete department	37	37	0	0	0	419
install applet	88	82	4	0	2	721
install tool	2 169	2 039	126	4	0	24 180
unistall applet	78	68	0	9	1	573
uninstall tool	1 896	1 888	0	8	0	22 419
register to a course	64	63	1	0	0	708
register to Webcampus	32	30	2	0	0	184
unregister from course	19	17	1	1	0	155
Total	**5 004**	**4 815**	**155**	**30**	**3**	**56 816**

Dynamic analysis for DBRE – A case study

		A	B	C	D	E	F
	class → class	0	0	0	0	0	0
✓	course → faculty	1927	1840	12	1 832	87	0
✓	course → right_profile	1927	51	12	0	1	0
✓	course_user → course	55	1927	9	9	47	5
✓	course_user → right_profile	55	51	9	0	9	0
✓	course_user → user	55	41	9	7	50	4
	desktop_portlet_data → user	0	41	0	0	0	0
✓	dock → module	244	496	14	58	297	0
✓	faculty → faculty	1840	1840	12	3	0	0
✓	course_addons → course	1838	1927	9	0	3 842	0
	course_program → course	0	1927	0	0	0	0
✓	user_addons → user	11	41	3	0	15	0
	user_addons → program	11	5	3	0	0	0
✓	im_message → course	3	1927	3	0	5	0
✓	im_message_status → user	4	41	3	0	2	0
	im_message_status → im_message	4	3	3	0	0	2
✓	im_recipient → user	4	41	3	0	2	0
	im_recipient → im_message	4	3	3	0	0	6
✓	log → user	3	41	3	0	1	0
✓	module_contexts → module	38	496	14	34	11	0
✓	module_info → module	17	496	4	13	11	0
✓	notify → course	9	1927	6	0	3	0
✓	notify → user	9	41	6	0	11	0
	notify → course_tool	9	452	2	0	0	0
	property_definition → user_property	0	1	0	0	0	0
	rel_class_user → user	0	41	0	0	0	0
	rel_class_user → class	0	0	0	0	0	0
	rel_course_class → class	1	0	0	0	0	0
✓	rel_course_class → course	1	1927	1	0	3	0
✓	right_rel_profile_action → course	161	1927	6	0	22	0
✓	right_rel_profile_action → right_profile	161	51	7	0	711	0
✓	right_rel_profile_action → right_action	161	178	7	32	810	8
	sso → user	0	41	0	0	0	0
✓	tracking_event → user	3	41	2	0	2	0
✓	user_property → user	1	41	1	1	0	0

A. #queries accessing t1

B. #queries accessing t2

C. #scenarios accessing t1 & t2

D. #joins between t1 & t2

E. #output-input dependencies query(t1) <-> query(t2)

F. #input-input dependencies query(t1) <-> query(t2)

Dynamic analysis for DBRE – A case study

Unexpected foreign key ($t_1 \rightarrow t_2$)	# of joins	Explanation
`course_tool` → `module`	372	actual implicit foreign key
`right_action` → `course_tool`	18	actual implicit foreign key
`notify` → `course_user`	6	t_1 and t_2 reference the same table t_3

Dynamic analysis for DBRE – A case study

- #True positives (TP) = number of *actual* foreign keys *detected*
- #False positives (FP) = number of *wrong* foreign keys *detected*
- #False negatives (FN) = number of *actual* foreign keys *undetected*

- Recall = TP / (TP + FN)
  - 25/37 = 71%
  - 25/29 = 86% (considering only *potentially detectable* FK's)

- Precision = TP / (TP + FP)
  - 25/(25+1) = 96%

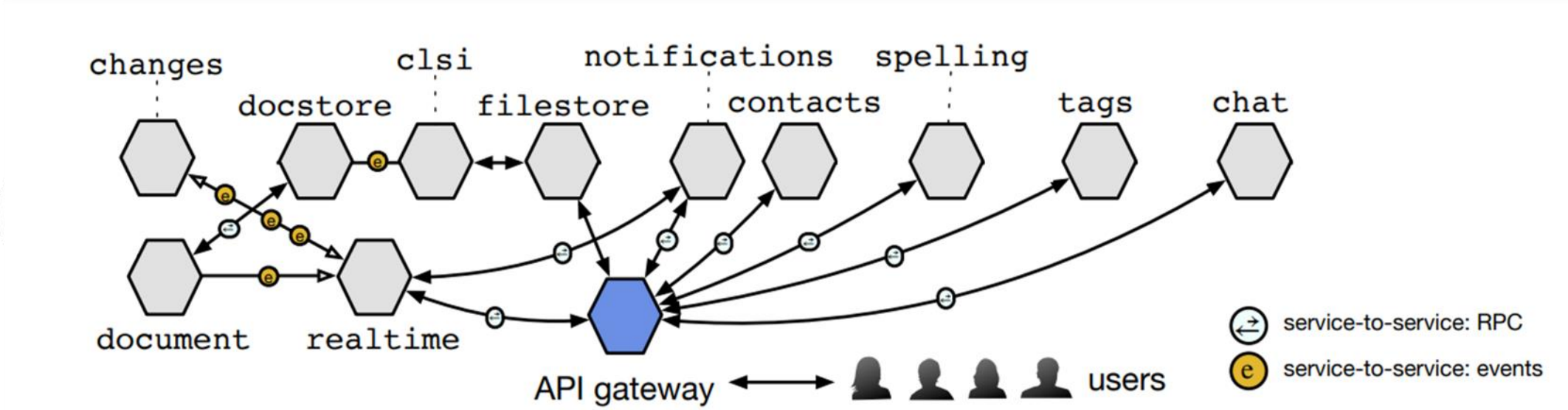
Recent research

Database access analysis and visualization

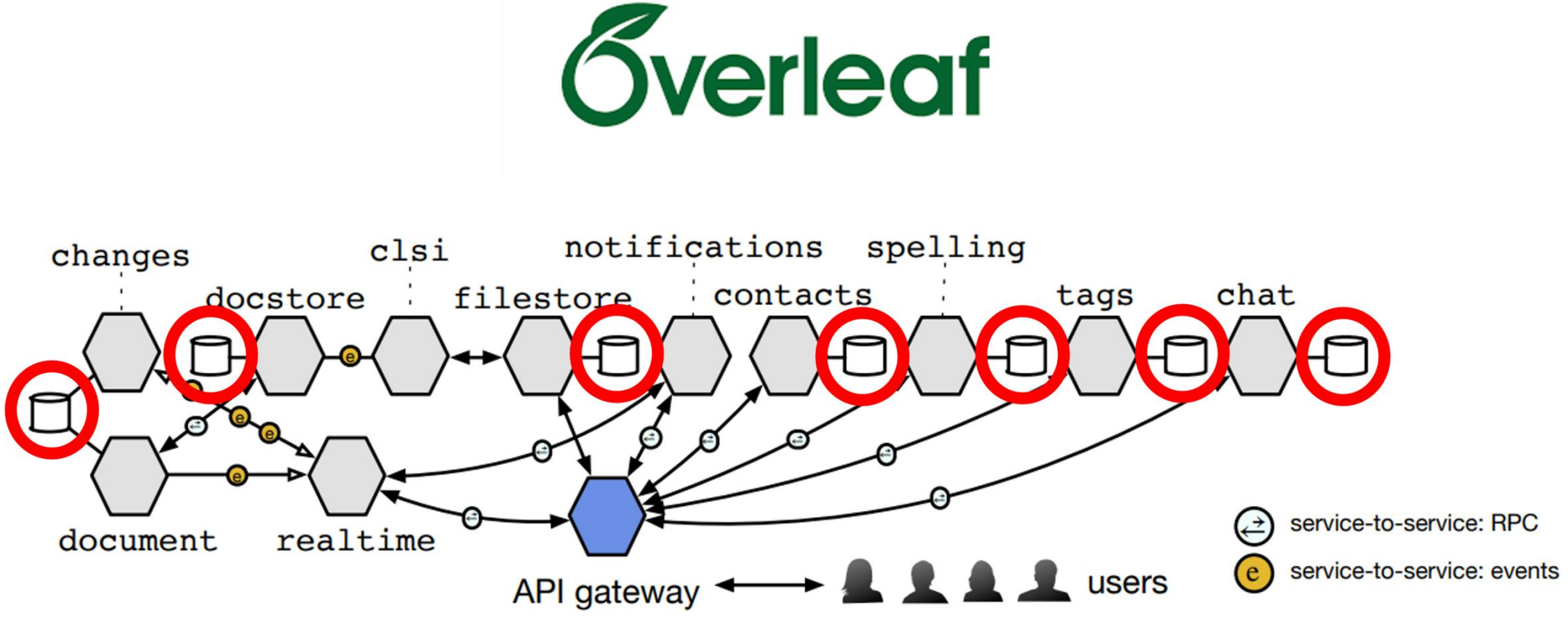
Focus on microservice applications

Microservice applications

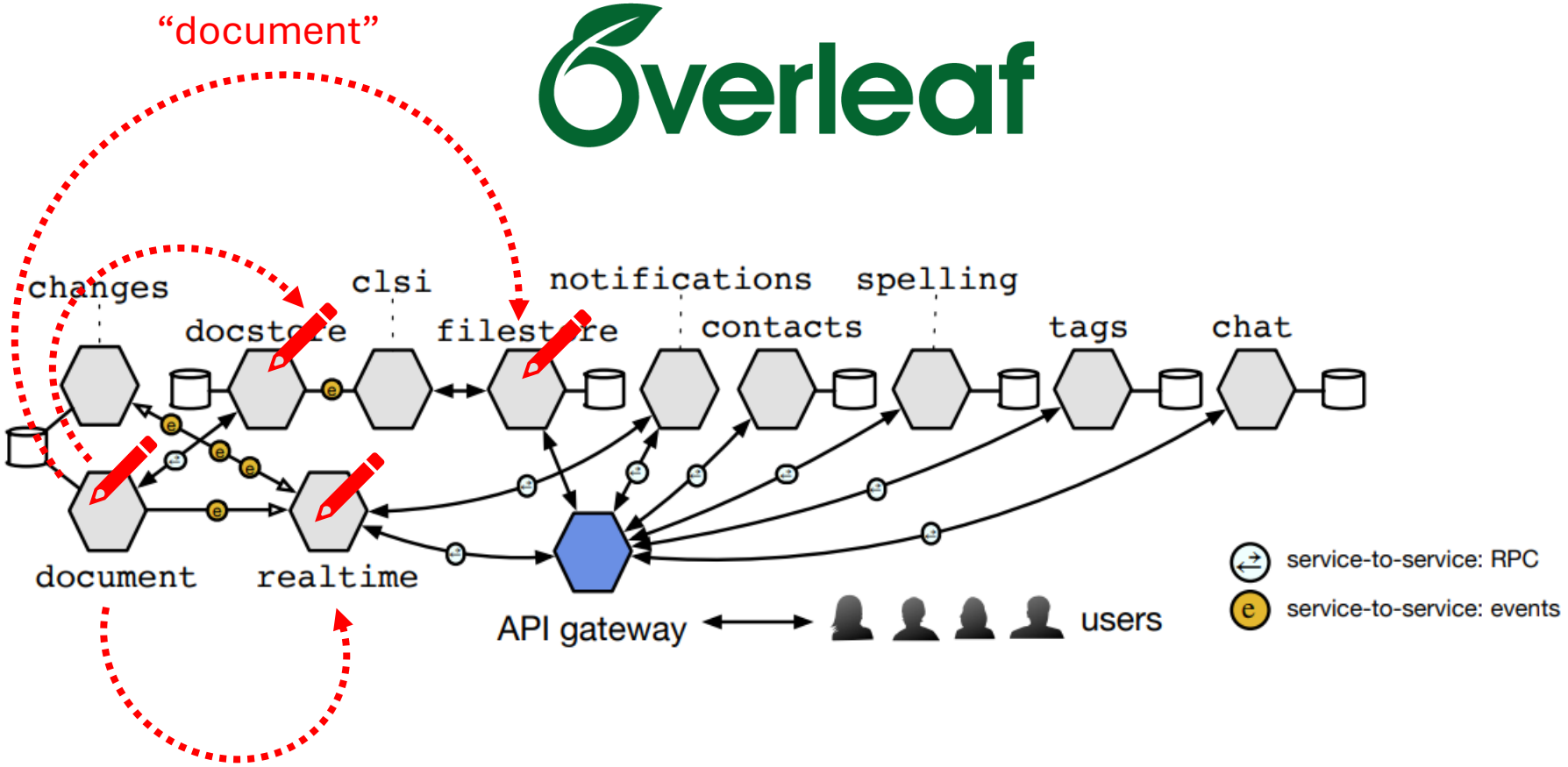
Overleaf



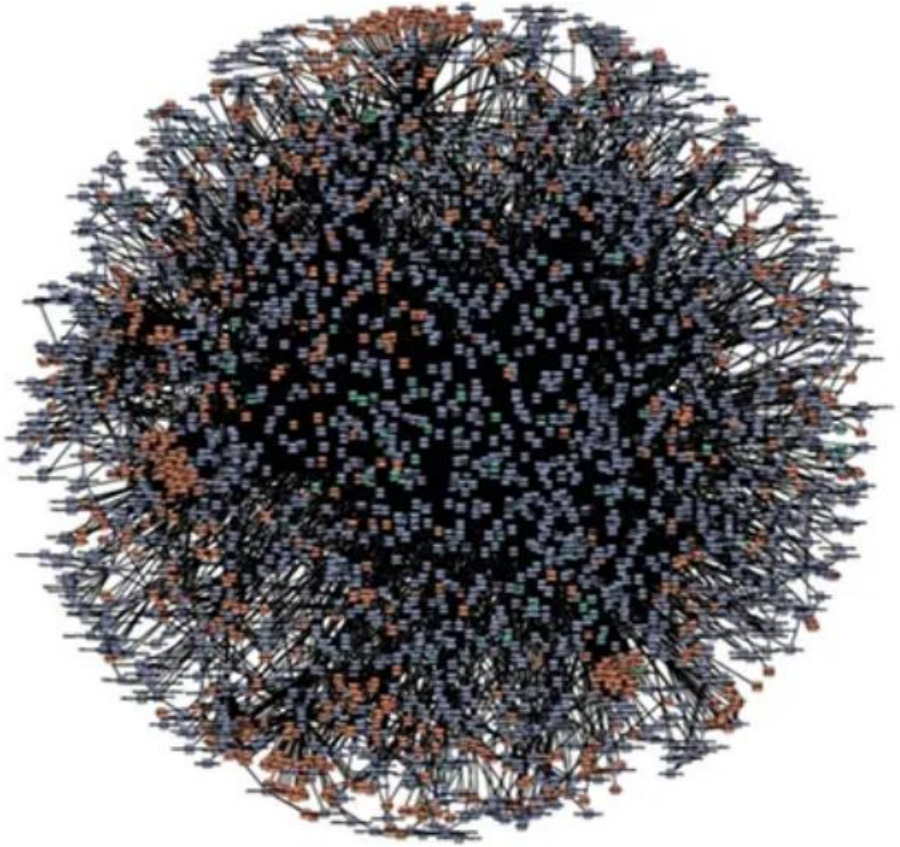
Microservice applications

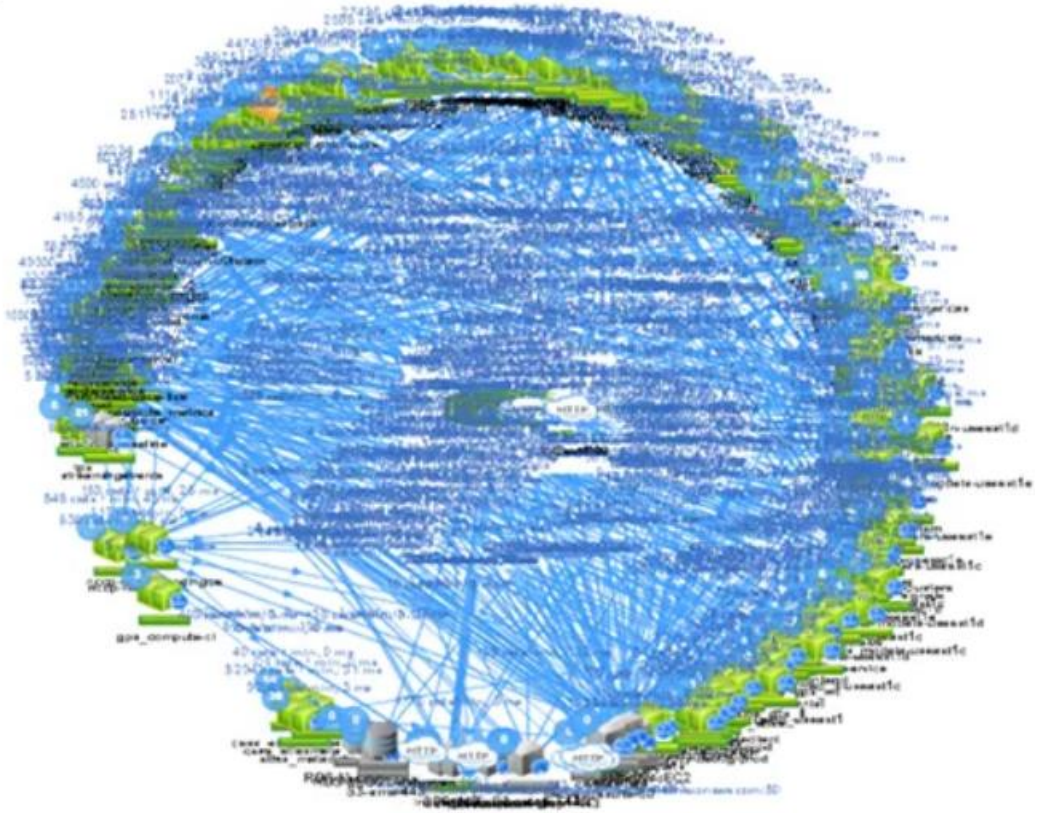


Microservice applications

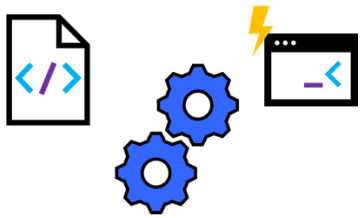


Other (huge) microservice applications



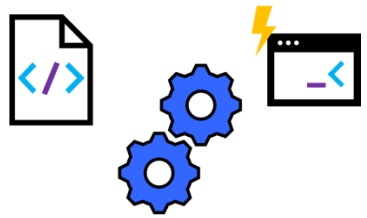


First research question



*How to **recover** database accesses from the large & dynamic source code of microservice applications?*

First research question

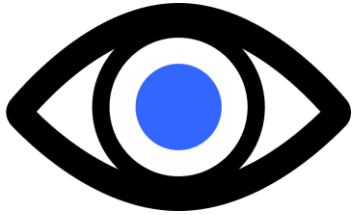


*How to **recover** database accesses from the large & dynamic source code of microservice applications?*

Static analysis

Dynamic analysis

Second research question



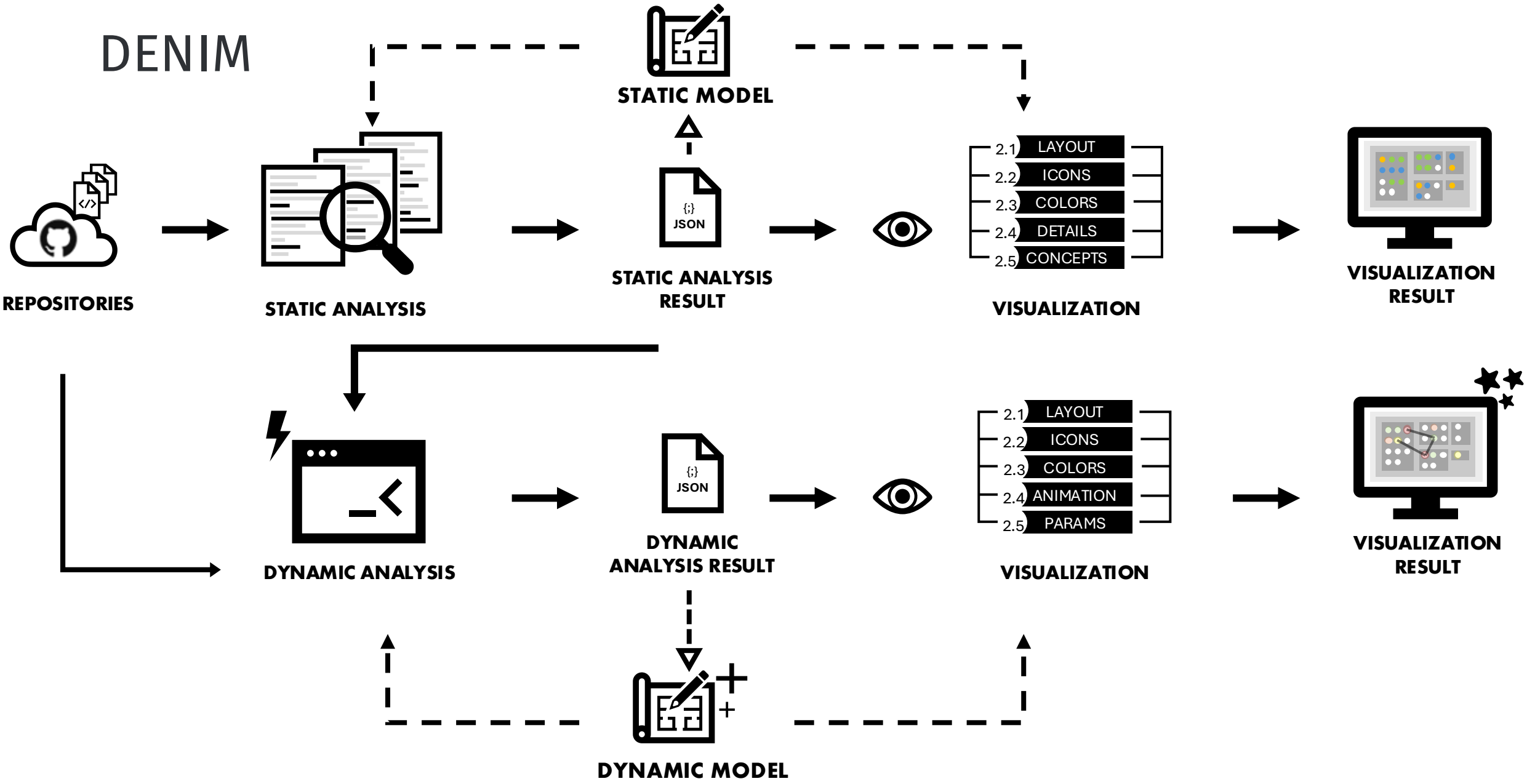
*How to **<u>understand</u>** the database access behavior of large microservice applications?*

Static analysis

Dynamic analysis

Visualization

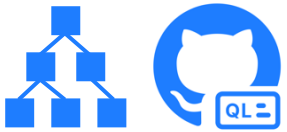
DENIM


STATIC MODEL
REPOSITORIES
STATIC ANALYSIS
{;}
JSON
STATIC ANALYSIS RESULT
2.1 LAYOUT
2.2 ICONS
2.3 COLORS
2.4 DETAILS
2.5 CONCEPTS
VISUALIZATION
VISUALIZATION RESULT
DYNAMIC ANALYSIS
{;}
JSON
DYNAMIC ANALYSIS RESULT
2.1 LAYOUT
2.2 ICONS
2.3 COLORS
2.4 ANIMATION
2.5 PARAMS
VISUALIZATION
VISUALIZATION RESULT
DYNAMIC MODEL

1 ACQUISITION

Static analysis

2 INITIALIZATION



3 IDENTIFICATION

4 EXTRACTION

5 INTERPRETATION

6 PRESENTATION

Static analysis

```
…  // ...
41 export async function findAllThreadRooms(projectId) {
42   return await db.rooms
43     .find(
44       {
45         project_id: new ObjectId(projectId.toString()),
46         thread_id: { $exists: true },
47       },
48       {
49         thread_id: 1,
50         resolved: 1,
51       }
52     )
53     .toArray()
54 }
…  // ...
```

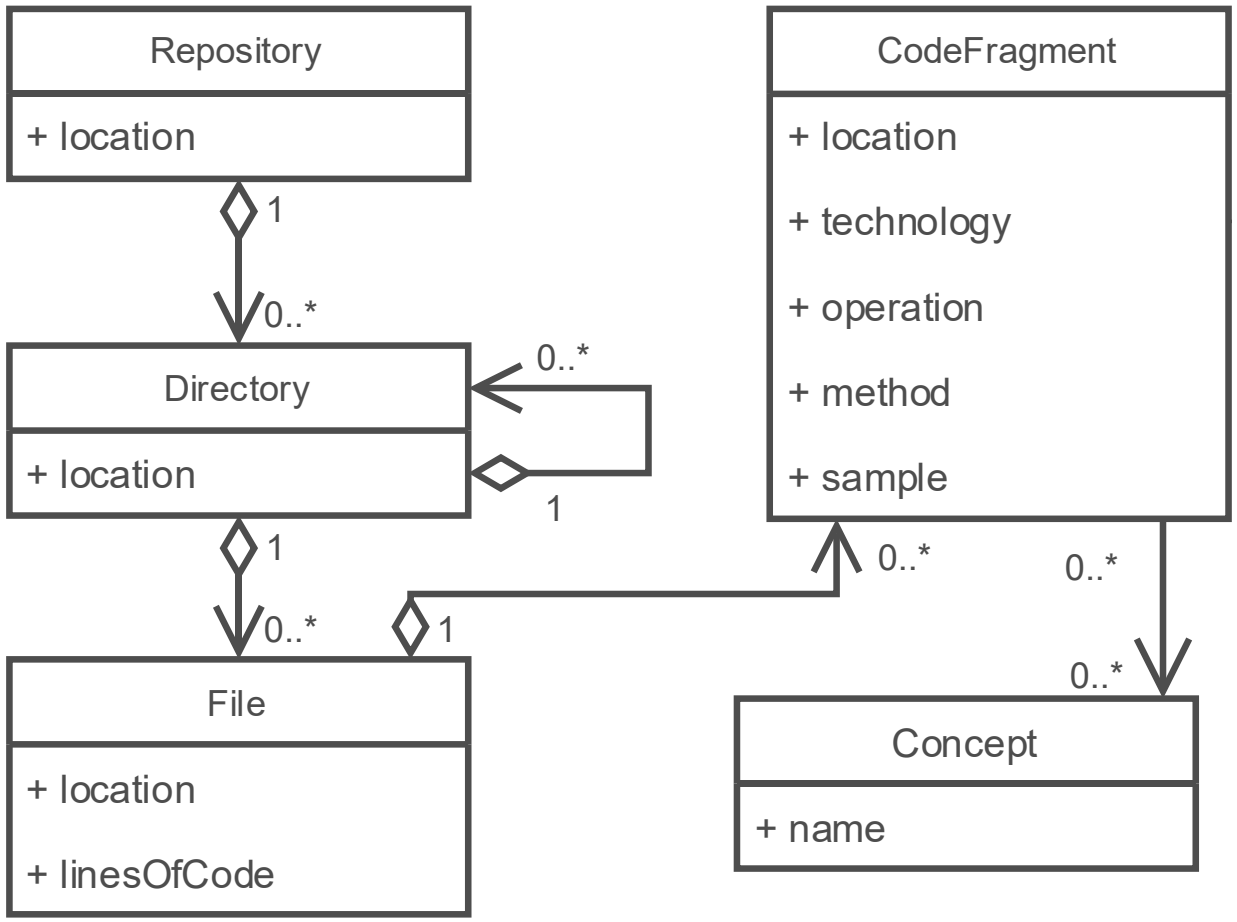
Static analysis

```
…  // ...
41 export async function findAllThreadRooms(projectId) {
42   return await db.rooms
43     .find(
44       {
45         project_id: new ObjectId(projectId.toString()),
46         thread_id: { $exists: true },
47       },
48       {
49         thread_id: 1,
50         resolved: 1,
51       }
52     )
53     .toArray()
54 }
…  // ...
```

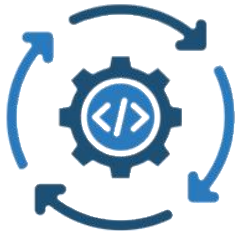
IDENTIFICATION

INTERPRETATION

Static analysis results

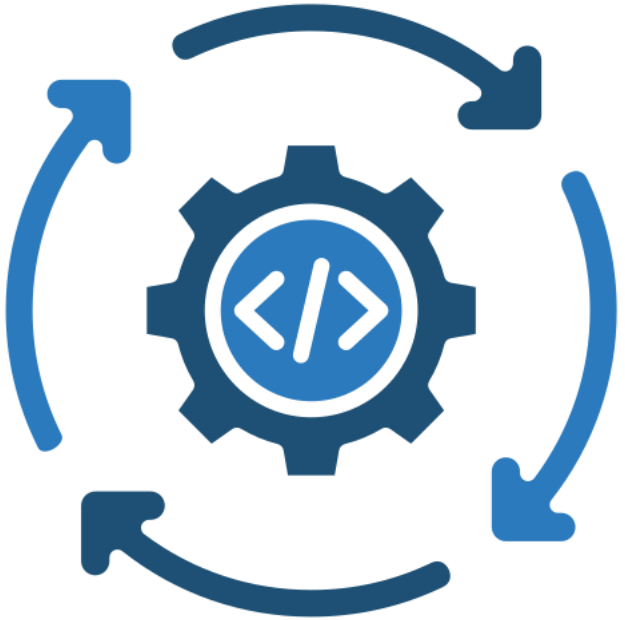


Capturing data access traces



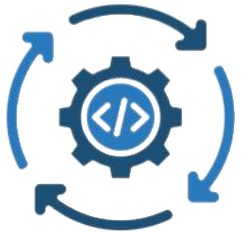
Where?

+



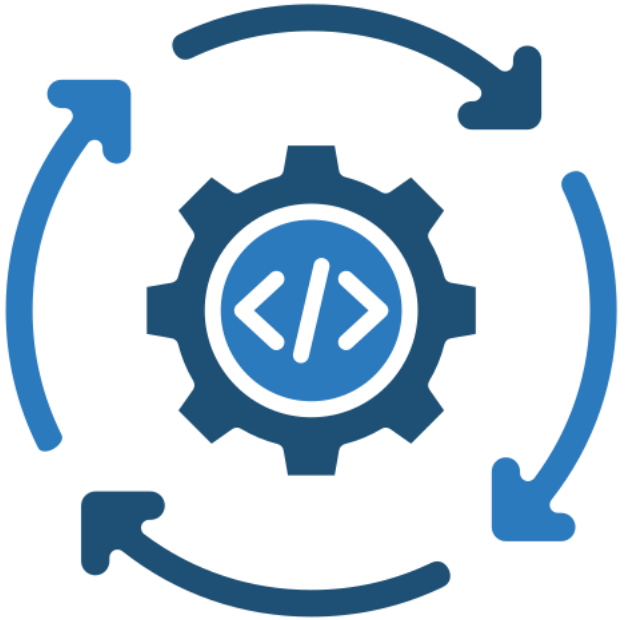
Dynamic instrumentation

Capturing data access traces



Where?

+



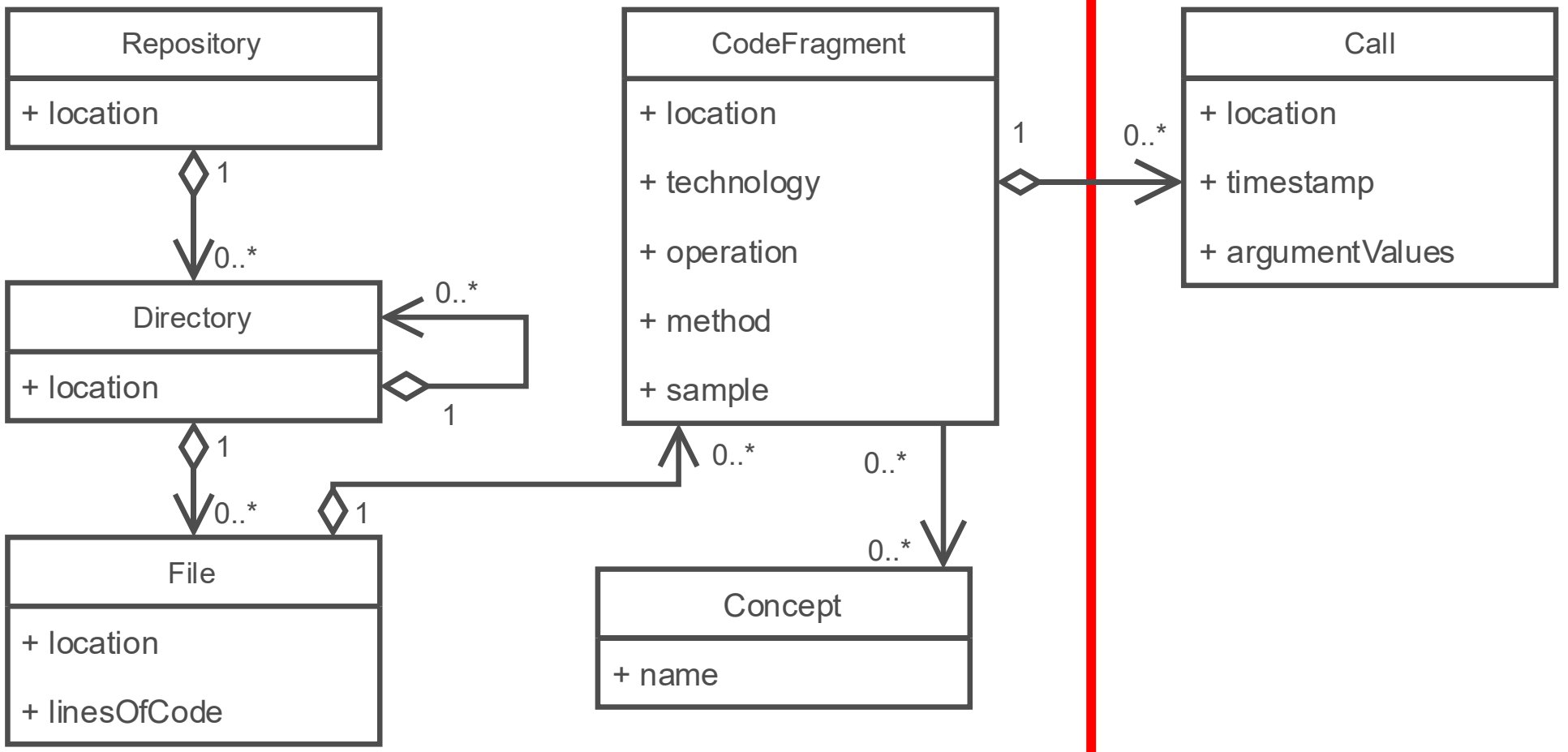
Dynamic instrumentation

➔ Relying on static analysis results

➔ Relying on *NodeProf*

Static analysis results

Dynamic analysis results



Evaluation – Setup

Static analysis

Overleaf

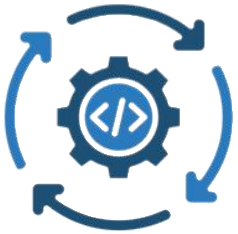
+ 9 other microservice applications

700k lines

865

12 microservices

Evaluation – Scenario



Overleaf

- 6 days
- 1 user
- Writing a paper

Evaluation – Static versus dynamic analysis

Static analysis

Static statistics

javascript-db-redis-call:	javascript-api-express-call:	javascript-db-mongo-call:
CREATE 8 (2.9%)	CREATE 465 (40.0%)	CREATE 78 (9.3%)
READ 103 (37.5%)	READ 574 (49.4%)	READ 477 (57.0%)
UPDATE 54 (19.6%)	UPDATE 39 (3.4%)	UPDATE 212 (25.3%)
DELETE 46 (16.7%)	DELETE 84 (7.2%)	DELETE 65 (7.8%)
OTHER 64 (23.3%)	OTHER 0 (0.0%)	OTHER 5 (0.6%)
TOTAL 275 (12.1%)	TOTAL 1162 (51.1%)	TOTAL 837 (36.8%)

Dynamic statistics

javascript-db-redis-call:	javascript-api-express-call:	javascript-db-mongo-call:
CREATE 10652 (11.0%)	CREATE 1295 (24.6%)	CREATE 25 (0.3%)
READ 30498 (31.6%)	READ 3595 (68.3%)	READ 7103 (96.7%)
UPDATE 24580 (25.5%)	UPDATE 35 (0.7%)	UPDATE 209 (2.8%)
DELETE 220 (0.2%)	DELETE 338 (6.4%)	DELETE 6 (0.1%)
OTHER 30609 (31.7%)	OTHER 0 (0.0%)	OTHER 0 (0.0%)
TOTAL 96559 (88.5%)	TOTAL 5263 (4.8%)	TOTAL 7343 (6.7%)

Dynamic analysis

Fragments

Redis Express Mongo

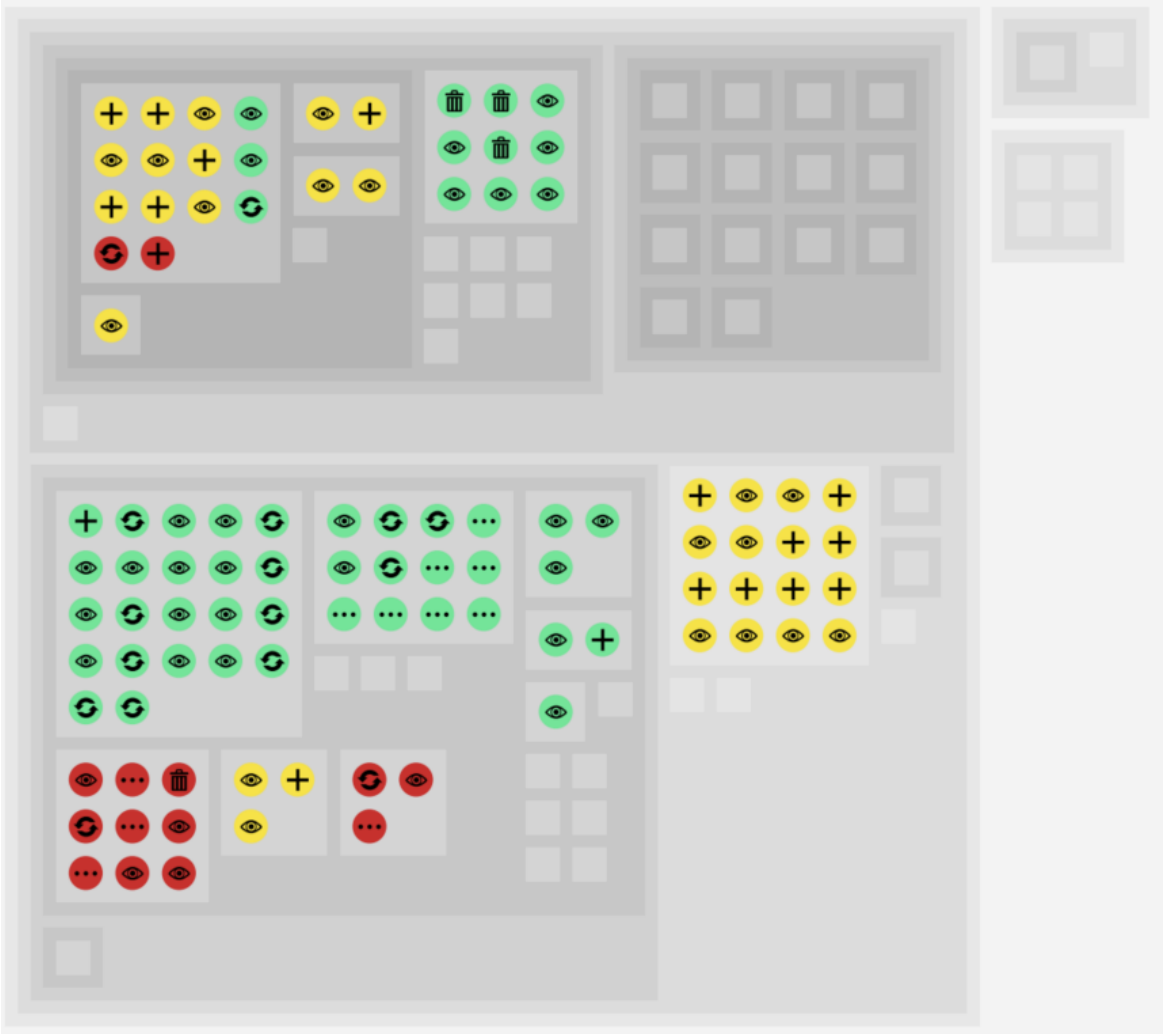
Calls

Redis Express Mongo

Overleaf

Visualization

Nested rectangles represent repository, directory, and file hierarchy

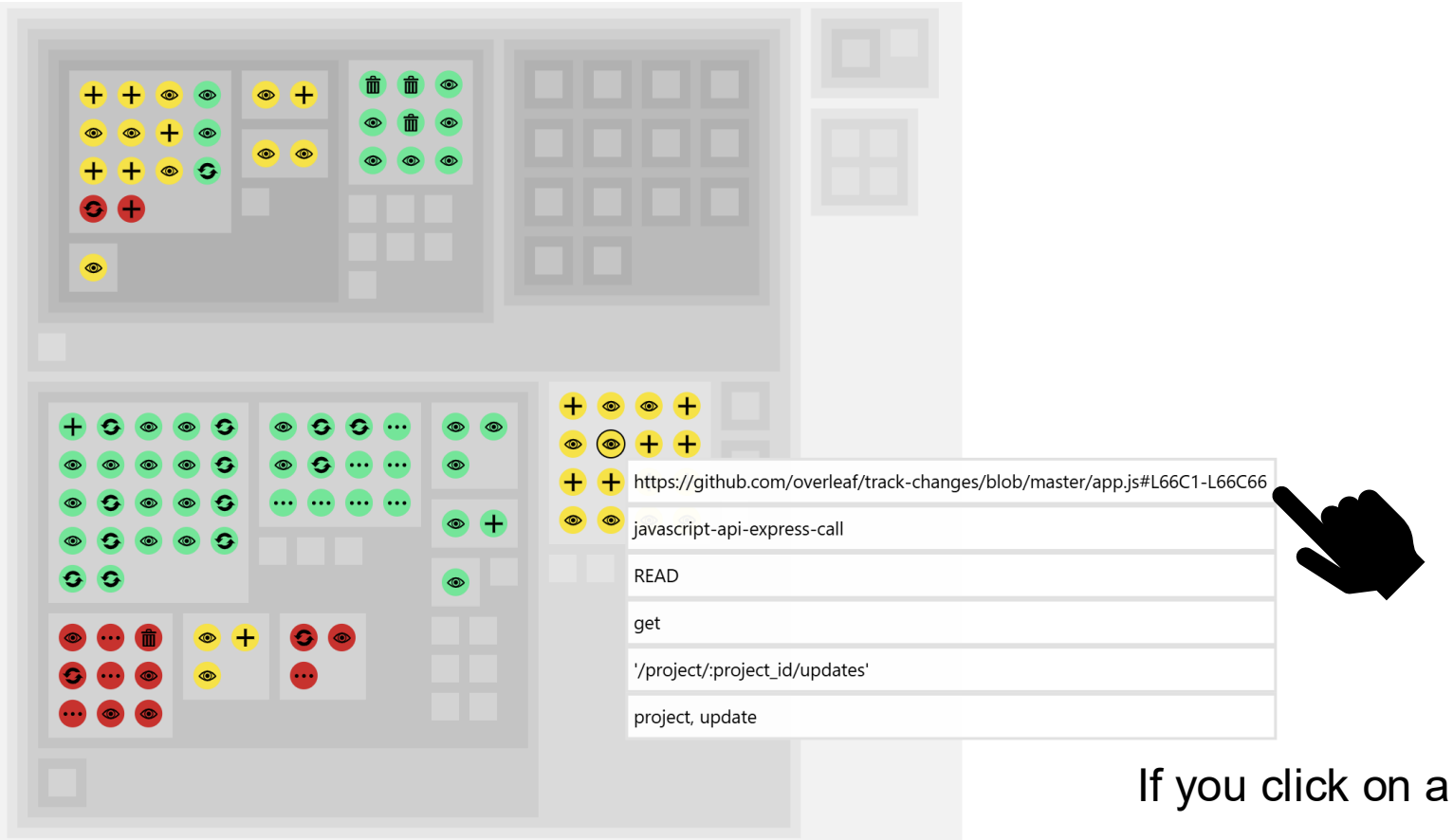


File
Code Fragment
ExpressJS
MongoDB
Redis

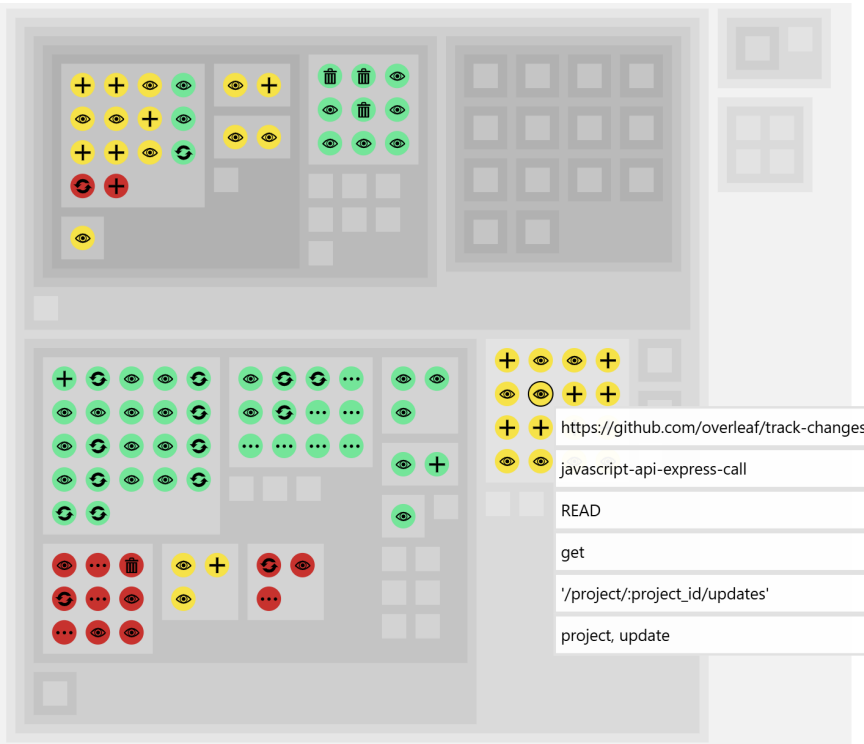
+ CREATE
READ
UPDATE
DELETE
... OTHER

Each coloured circle represents a data access

Visualization

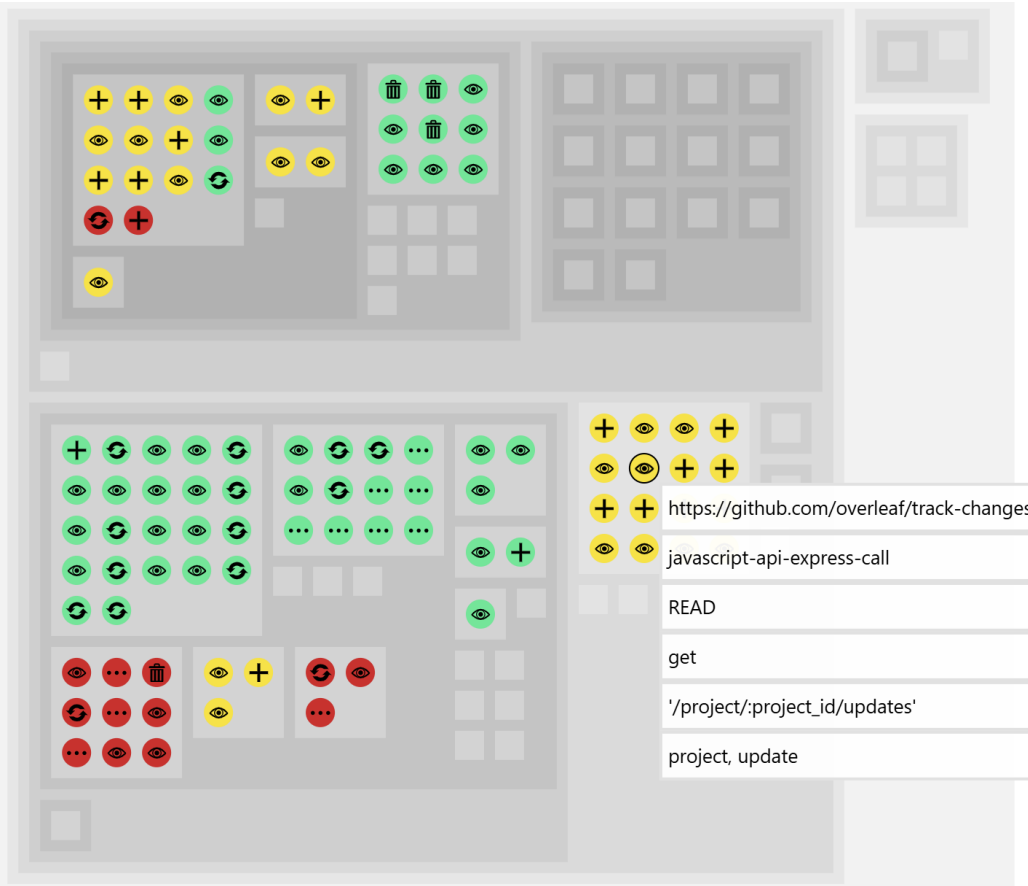


If you click on a given circle, you get further details



Visualization

If you click on a given circle, you get further details



https://github.com/overleaf/track-changes/blob/master/app.js#L66C1-L66C66

javascript-api-express-call

READ

get

'/project/:project_id/updates'

project, update

track-changes/app.js at master

github.com/overleaf/track-changes/blob/master/app.js#L66C1-L66C66

Files

master

Go to file t

.github

app

config

test

.dockerignore

.eslintignore

.eslintrc

.gitignore

track-changes / app.js

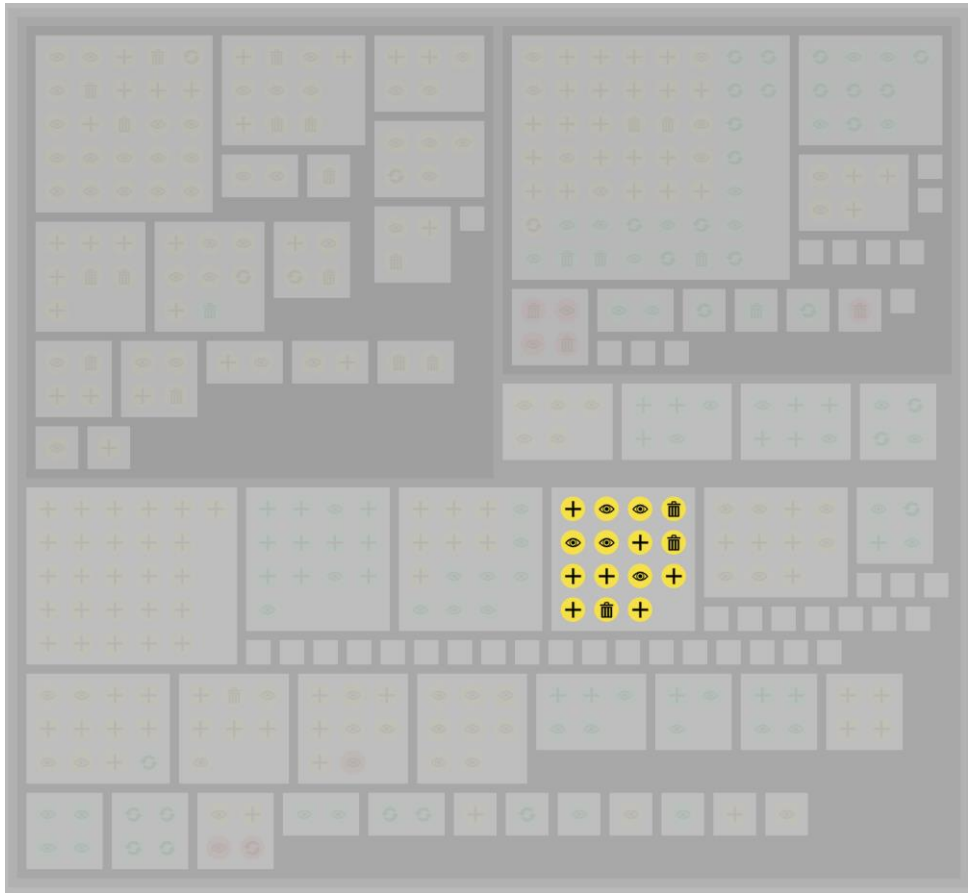
Code Blame 161 lines (132 loc) · 4.38 KB

```
19  const truncateFn = updates =>
62  app.get('/project/:project_id/doc/:doc_id/diff', HttpController.getDiff)
63
64  app.get('/project/:project_id/doc/:doc_id/check', HttpController.checkDoc)
65
66  app.get('/project/:project_id/updates', HttpController.getUpdates)
67  app.get('/project/:project_id/export', HttpController.exportProject)
68
69  app.post('/project/:project_id/flush', HttpController.flushProject)
70
71  app.post(
72    '/project/:project_id/doc/:doc_id/version/:version/restore',
73    HttpController.restore
74  )
75
```

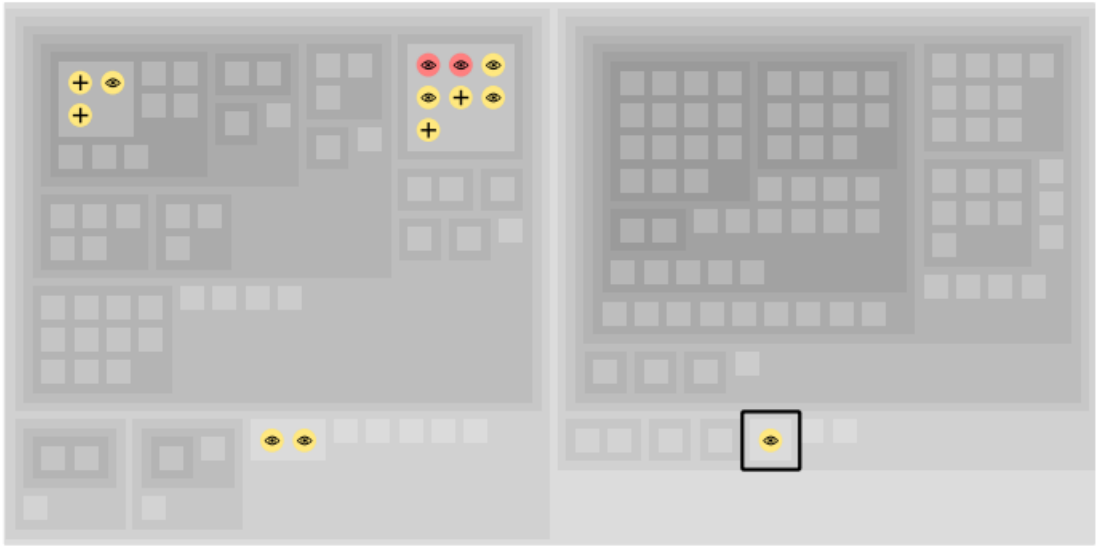
And you can inspect the correspond source code fragment

1

2

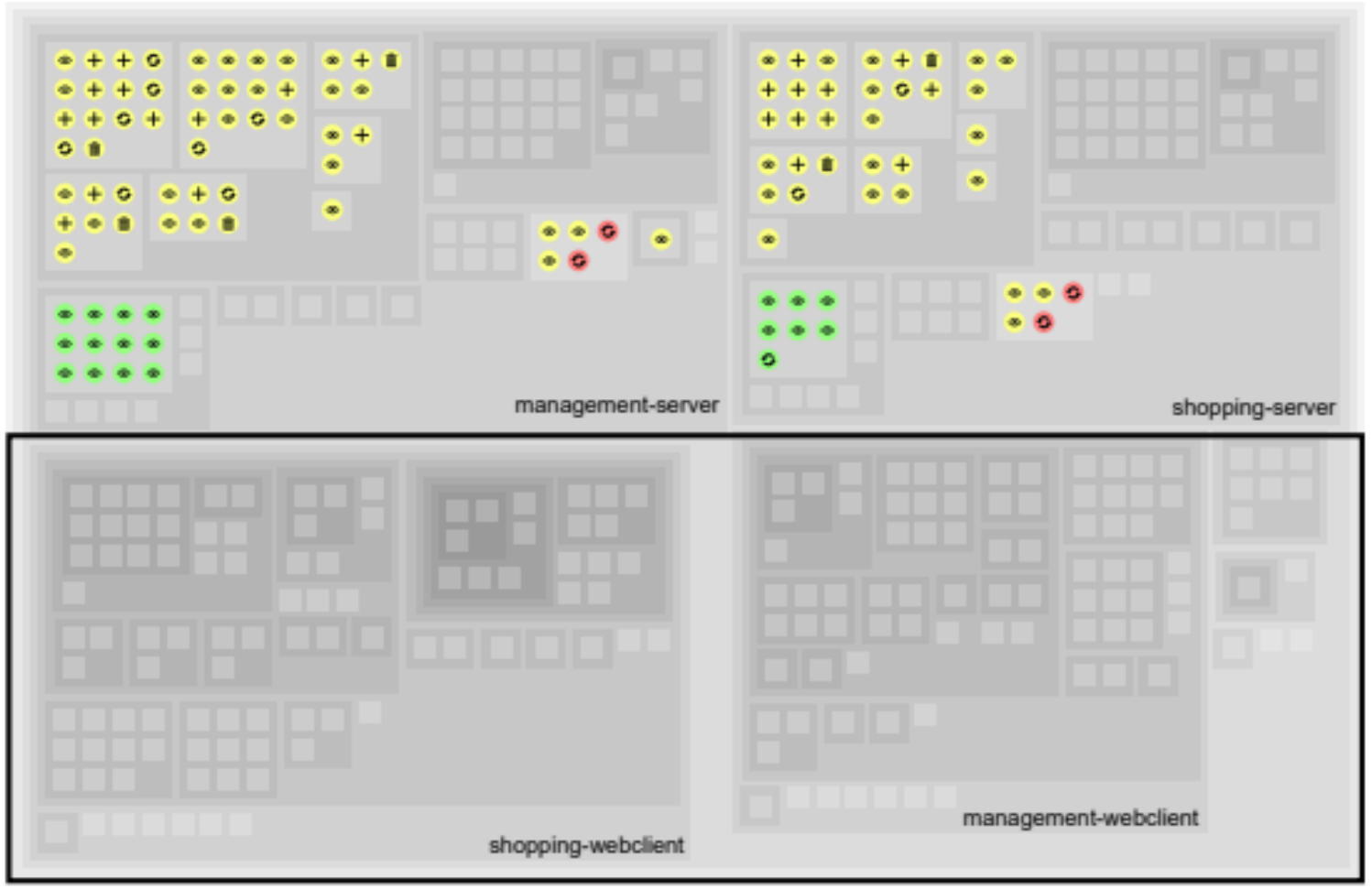


Focus on a particular data entity concept
spread access (1) versus centralized access (2)

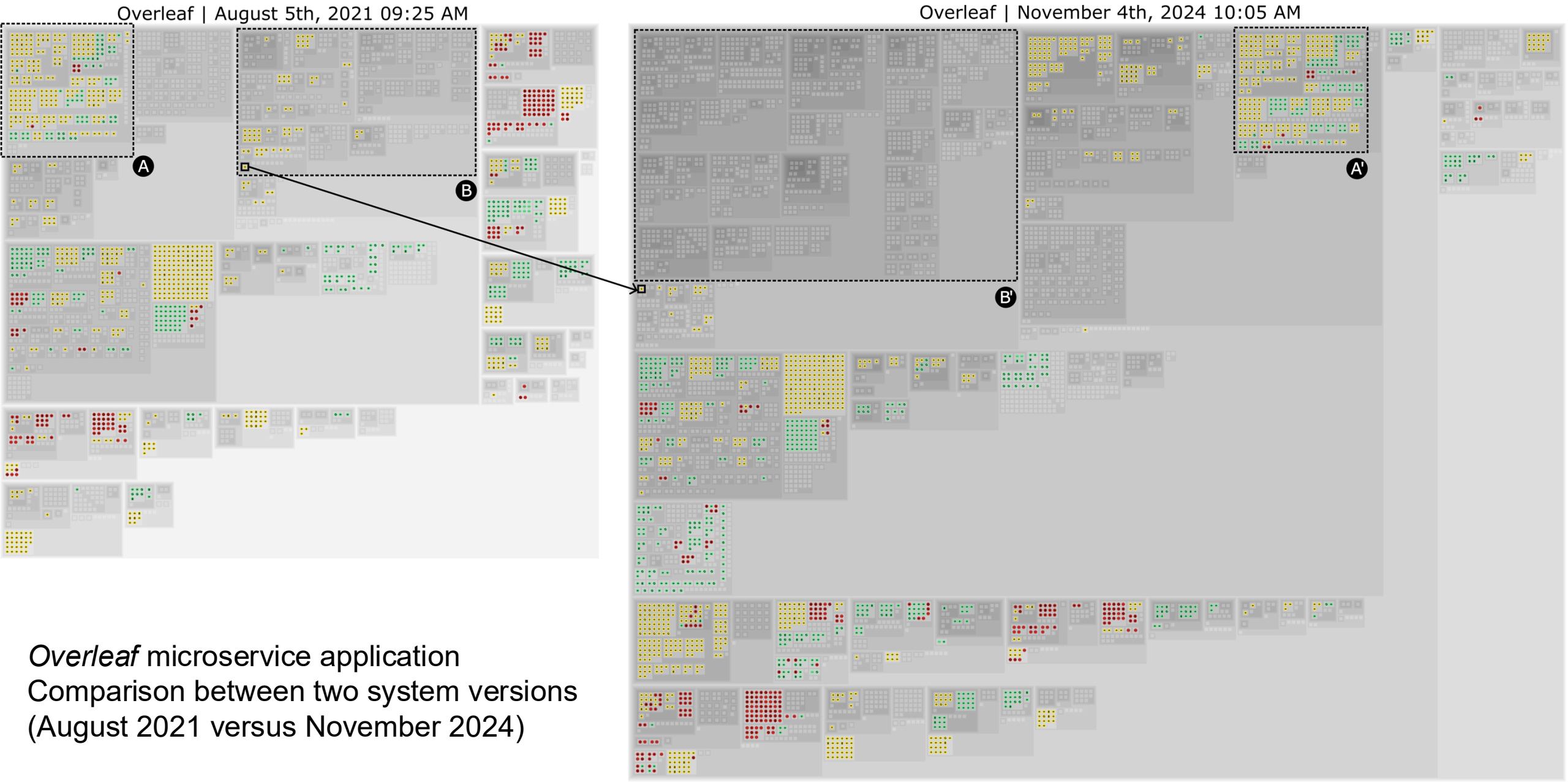


Do not forget to update this isolated data access in case of schema change !

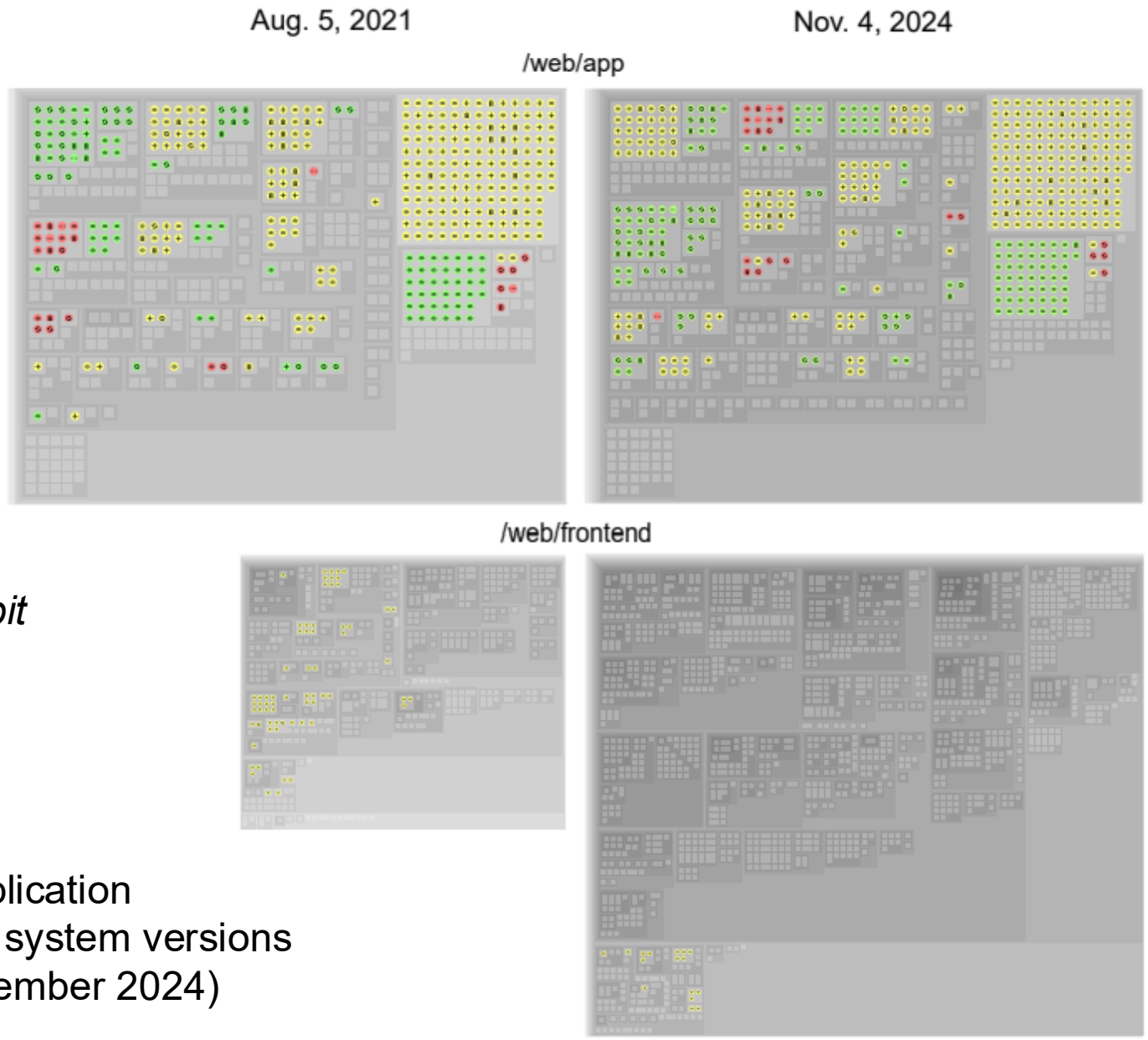
RAIR microservice application
different distribution of data access technologies in the system



Veniqa microservice application
clear separation between backend (top) and frontend (bottom) microservices.

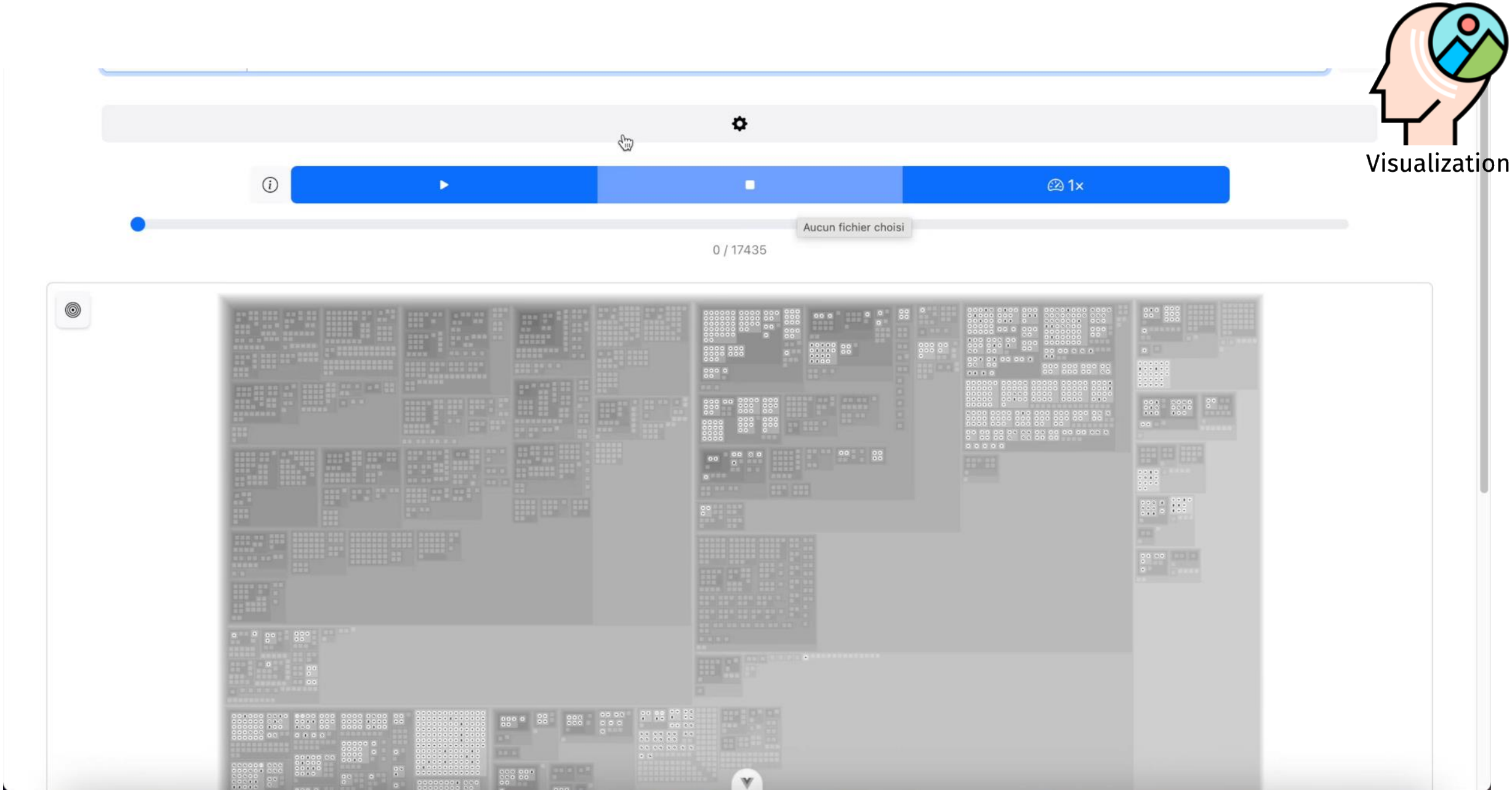


Overleaf microservice application
Comparison between two system versions
(August 2021 versus November 2024)

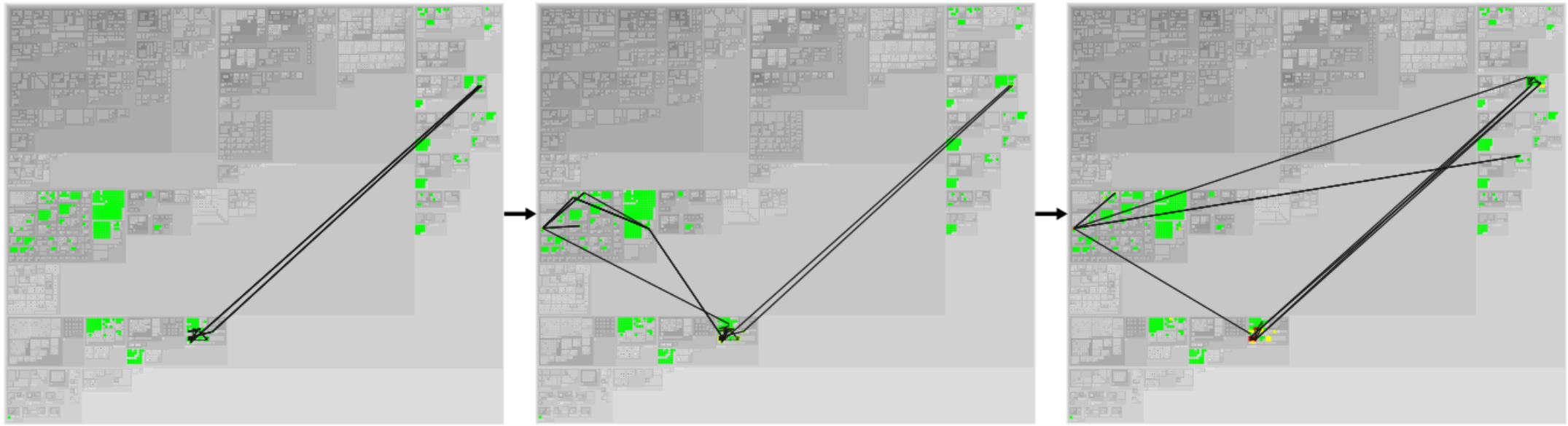


Let's zoom a bit

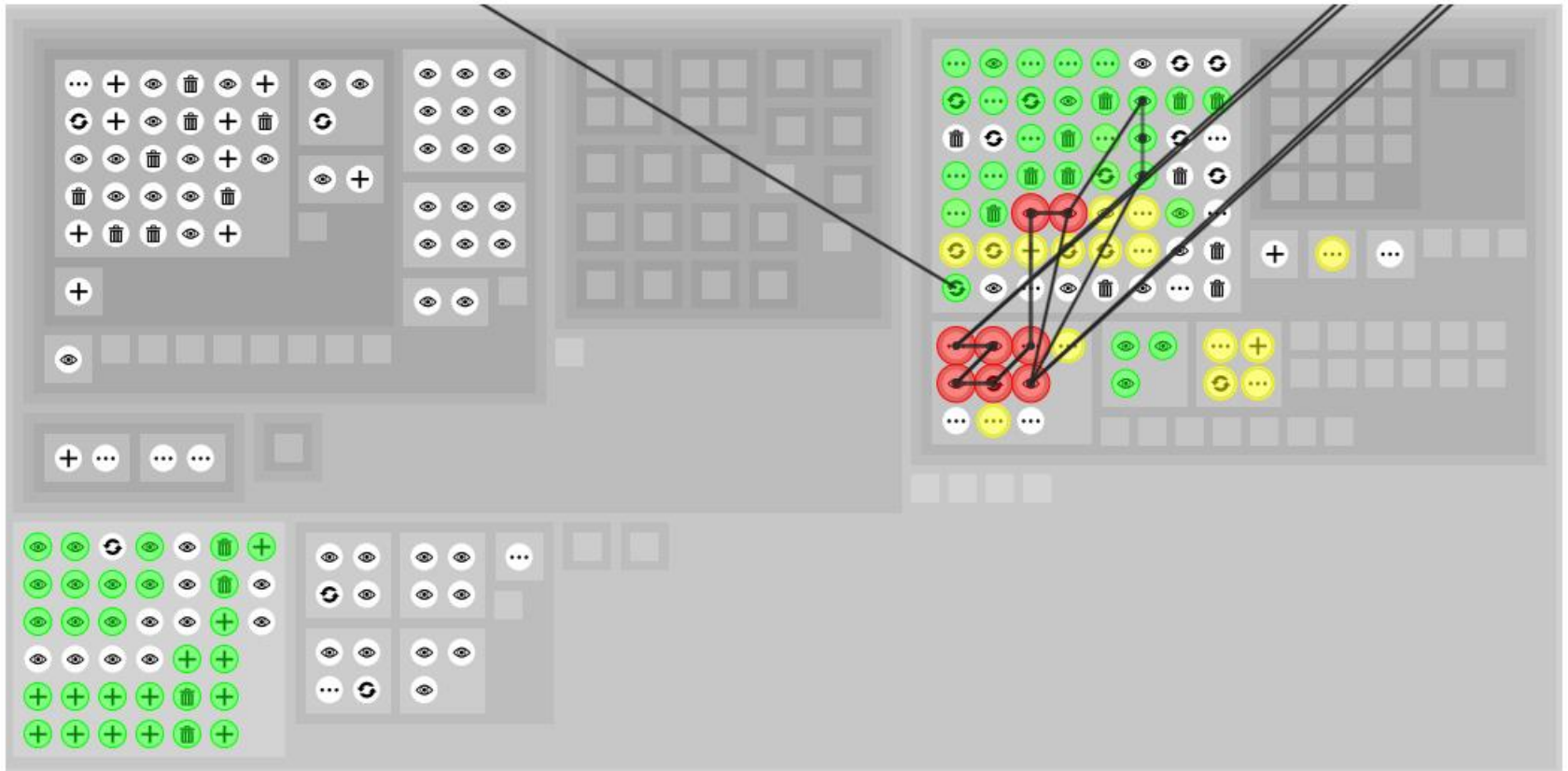
Overleaf microservice application
Comparison between two system versions
(August 2021 versus November 2024)



Overleaf microservice application - Dynamic analysis visualization demo



Analysis of the ordered sequence of data access execution



Identification of data access hotspots

Conclusions and perspectives

Conclusions and perspectives

- The (implicit) triangle between data, schema and programs enables a systemic perspective to data (access) quality
- Automating database reverse engineering tasks is reachable in real contexts
- The complexity of the database reverse engineering process constantly increases
- Short-term research perspectives
  - AI-based DIS – adequate data management support in data/AI pipelines
  - AI-based database reverse engineering – data/schema/program analysis to reduce technical debt
  - Migration of legacy DIS towards agentic architectures – keeping control with human in the loop