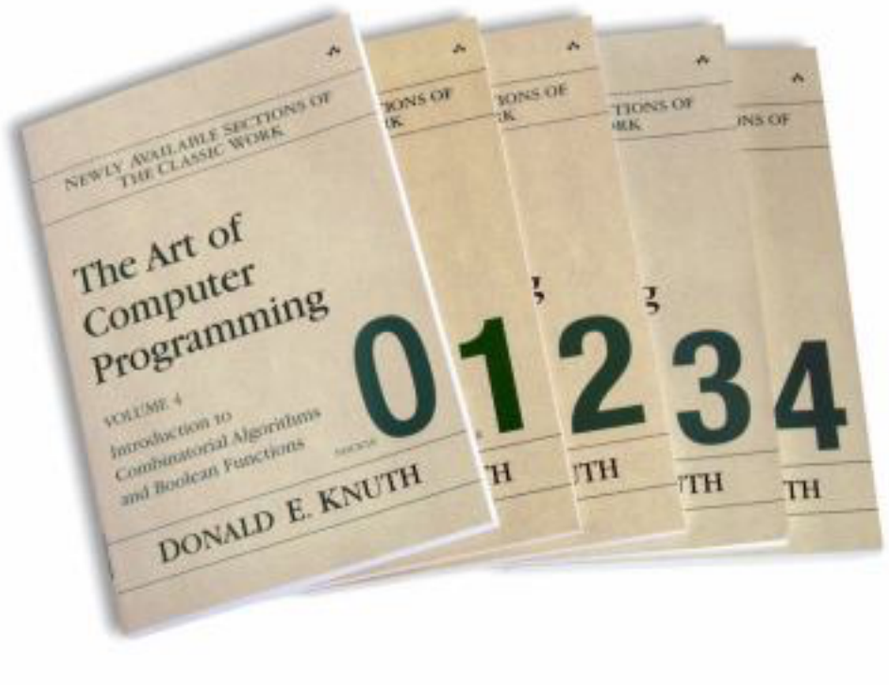
datalab

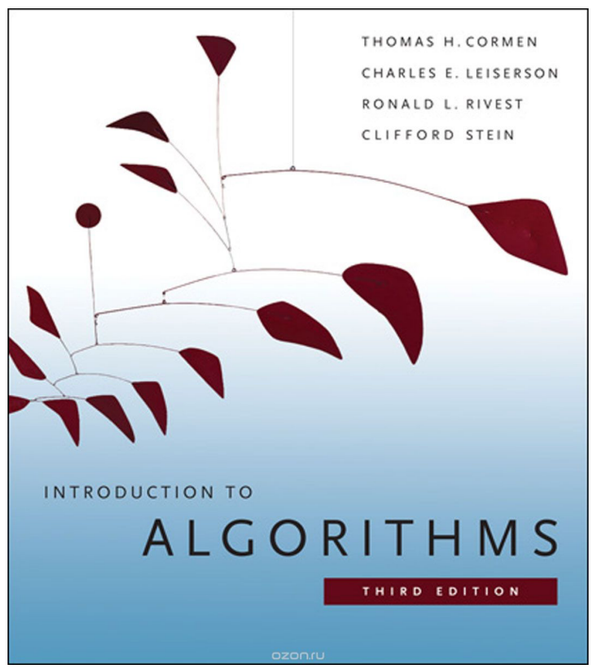
data is everywhere, value is hidden

Relational Databases

Lecturer: Азат Якупов (Azat Yakupov)
https://datalaboratory.one



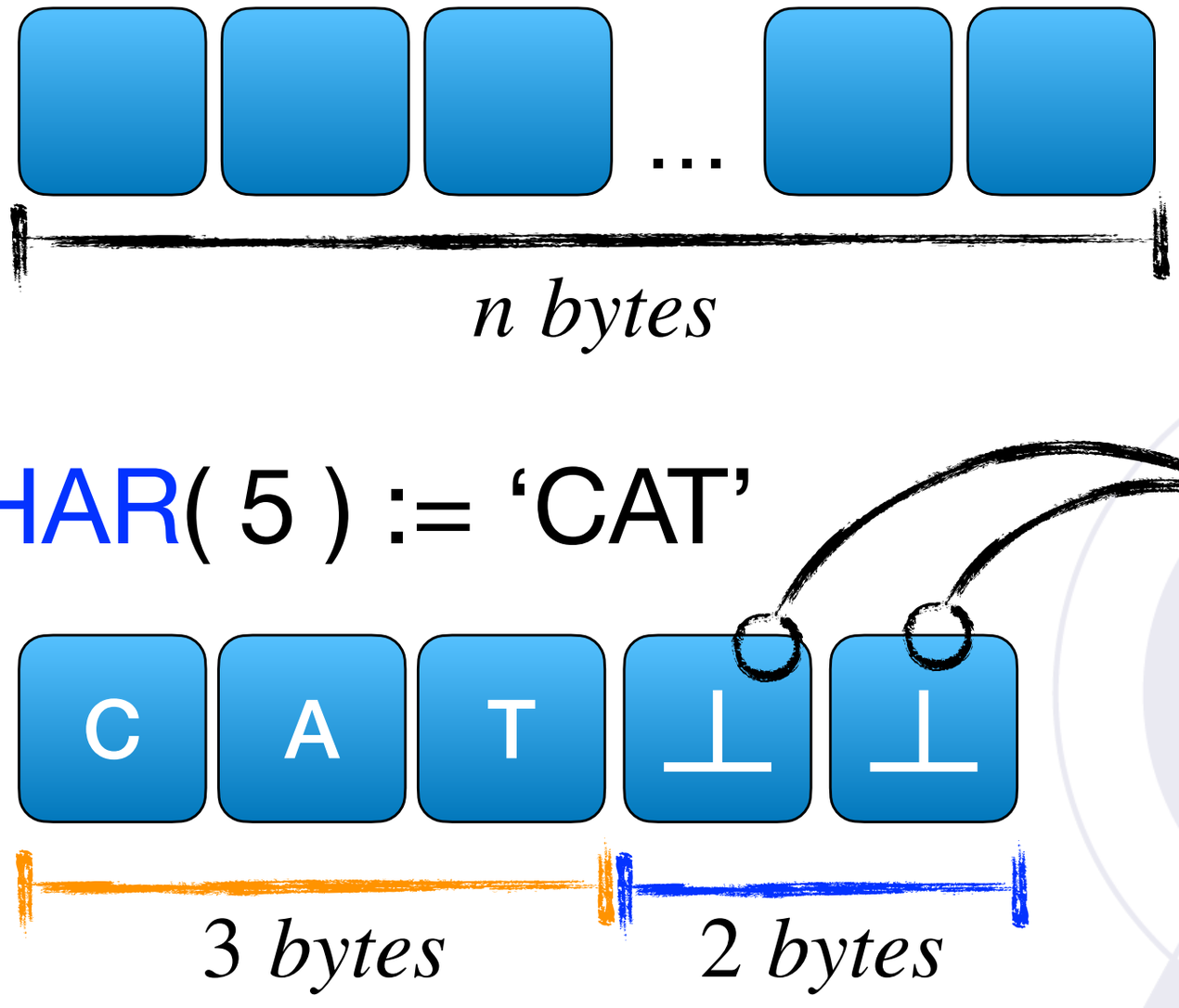
The Art of Computer Programming
by Donald Ervin Knuth



Introduction to Algorithms
by Thomas H. Cormen
Charles E. Leiserson
Ronald L. Rivest
Clifford Stein

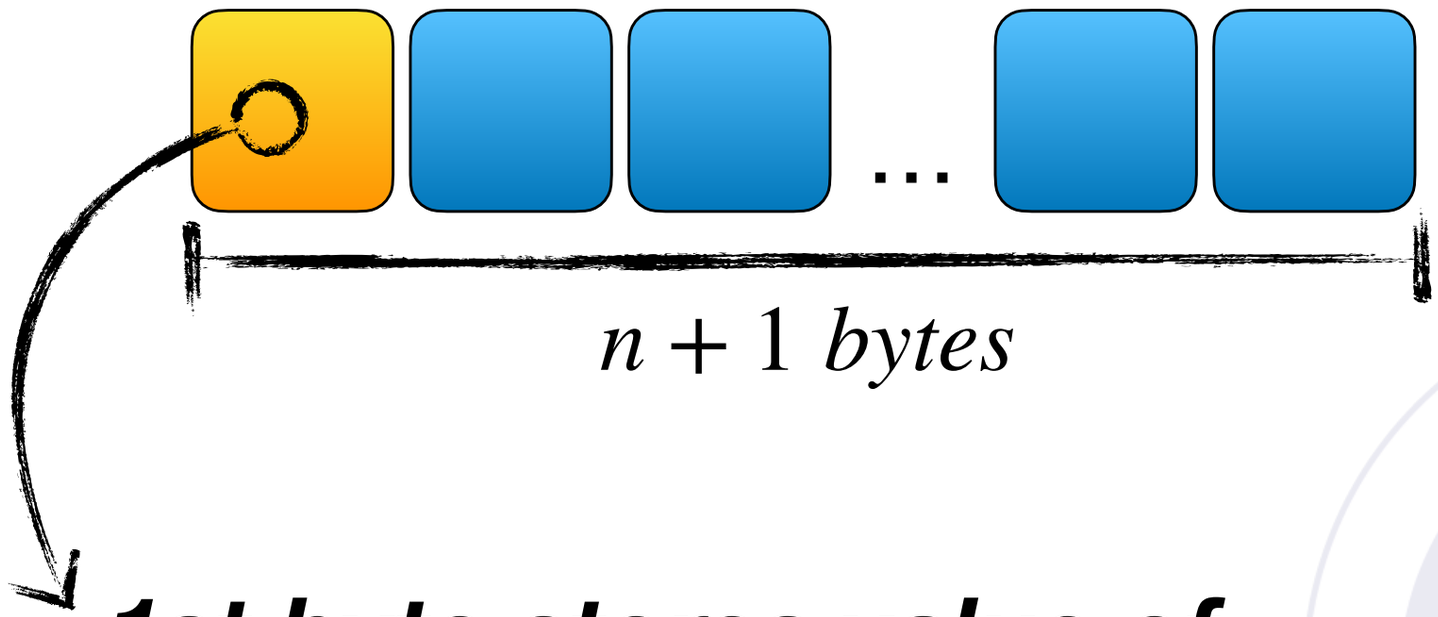
- Numerical types
- Date types
- String types
- Other built-in types
- User defined types

CHAR(n) ~ CHARACTER(n)



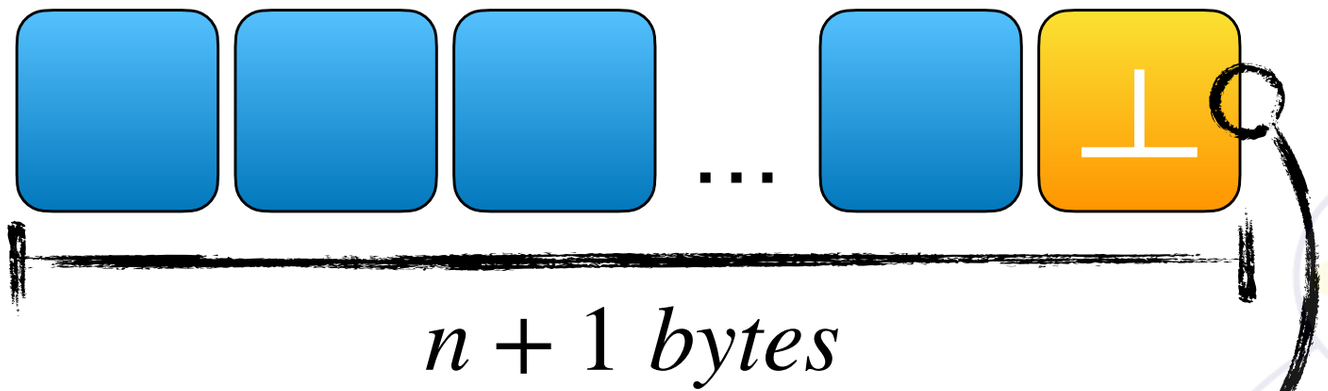
CHAR(5) := ‘CAT’

VARCHAR(n)



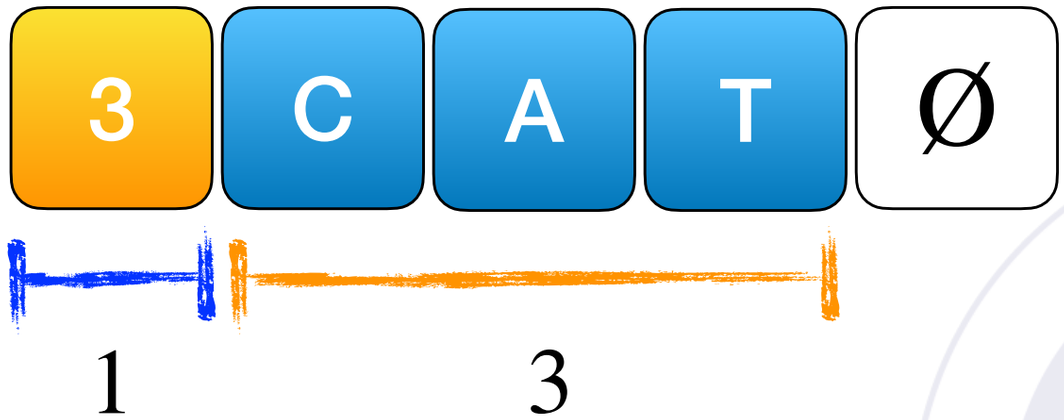
1st byte stores value of
total amount of real bytes

VARCHAR(n)

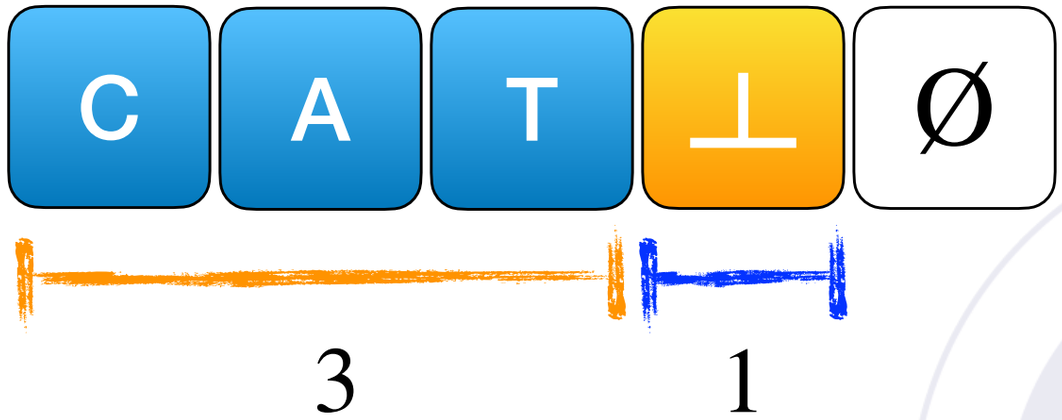


last byte stores
pad symbol

VARCHAR(5) := ‘CAT’

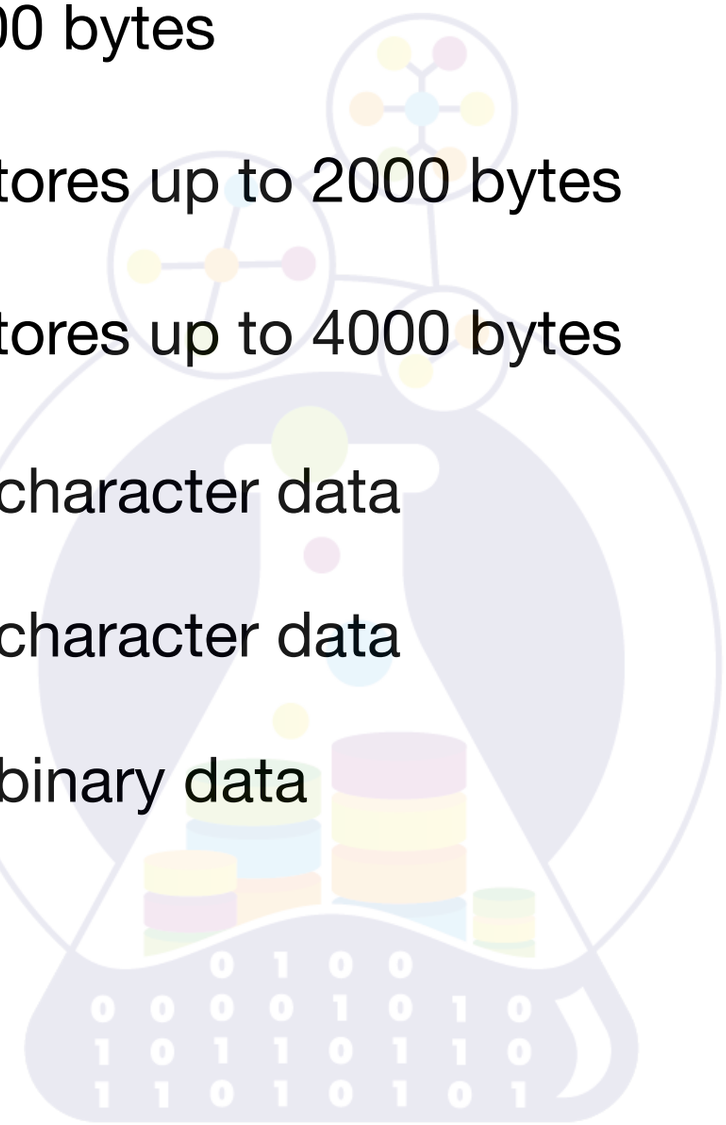


VARCHAR(5) := ‘CAT’



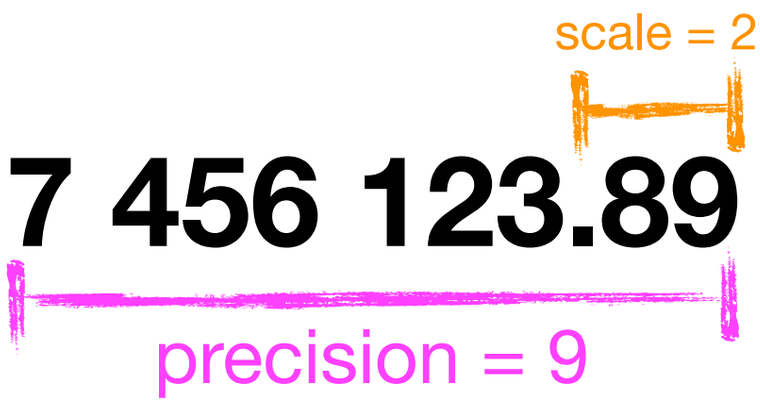
ORACLE

- CHAR (N) — N between 1 and 2000 bytes
- VARCHAR2 (N) — N between 1 and 4000 bytes
- NCHAR (N) — Unicode Data Type stores up to 2000 bytes
- NVARCHAR (N) — Unicode Data Type stores up to 4000 bytes
- CLOB — stores up to 4 GB of character data
- NCLOB — stores up to 4 GB of character data
- BLOB — stores up to 4 GB of binary data



ORACLE

NUMBER (precision [, scale])



precision is decimal digits in a range from 1 to 38
scale is decimal digits in a range from -84 to 127

INPUT DATA	TYPE	STORED AS
7 456 123.89	**NUMBER**	7 456 123.89
7 456 123.89	**NUMBER** (*, 1)	7 456 123.9
7 456 123.89	**NUMBER** (9)	7 456 124
7 456 123.89	**NUMBER** (9,2)	7 456 123.89
7 456 123.89	**NUMBER** (9,1)	7 456 123.9
7 456 123.89	**NUMBER** (6)	exceeds precision
7 456 123.89	**NUMBER** (7,-2)	7456100

ORACLE

- FLOAT (precision) is subtype of the **NUMBER**

  precision is binary bits in a range from 1 to 126

- REAL ~ FLOAT (63)
- DOUBLE PRECISION ~ FLOAT (126)
- INTEGER, INT ~ NUMBER (38)
- SMALLINT ~ NUMBER (38)

ORACLE

- BINARY_FLOAT

  IEEE 32 bit floating-point values

  Precision of 6-7 digits

  Require 4 bytes

```
SELECT 10.2d
FROM dual;
```

- BINARY_DOUBLE

  IEEE 64 bit floating-point values

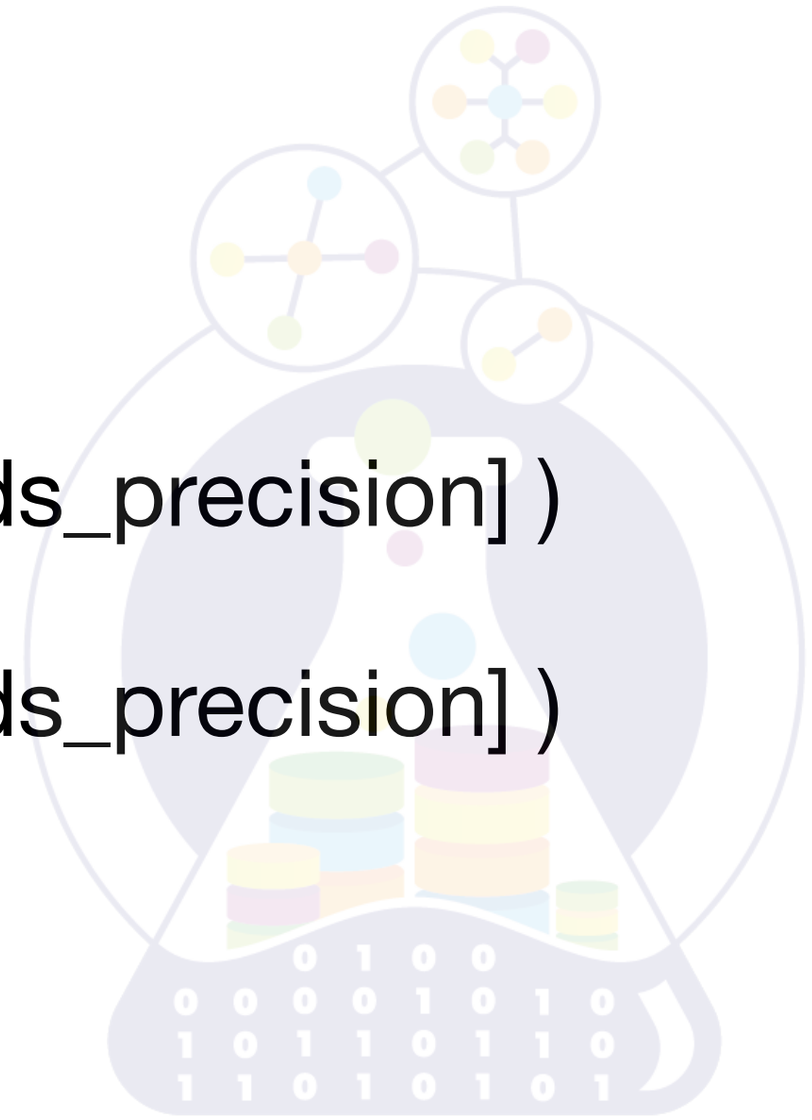
  Precision of 15 digits

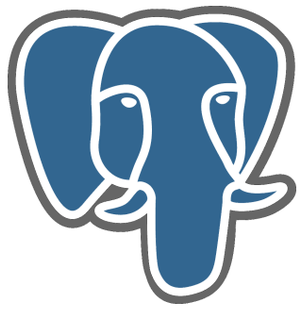
  Require 8 bytes

```
SELECT 32.7f
FROM dual;
```

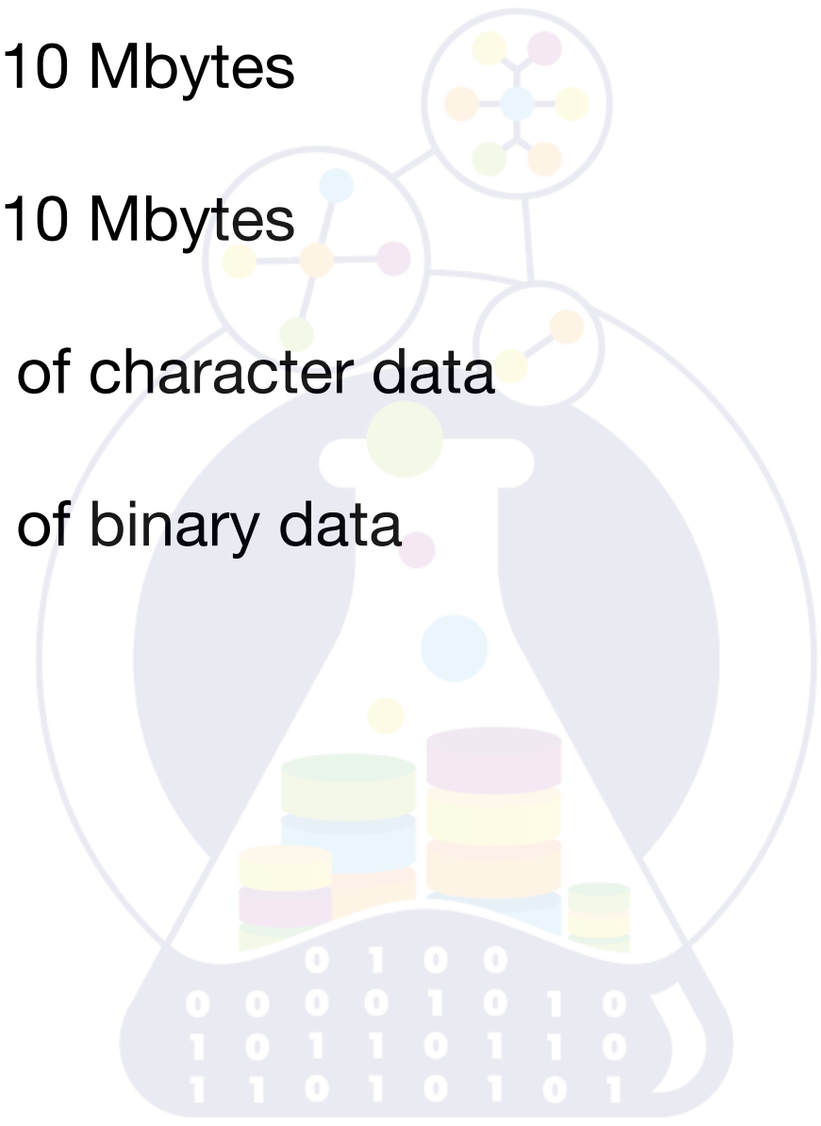
ORACLE

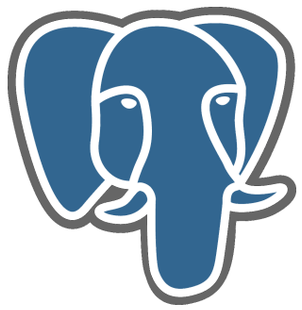
- DATE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND
- TIMESTAMP ([fractional_seconds_precision])
- TIMESTAMP ([fractional_seconds_precision]) WITH TIME ZONE





- CHAR (N) — N between 1 and 10 Mbytes
- VARCHAR (N) — N between 1 and 10 Mbytes
- TEXT — stores up to 1 GB of character data
- BYTEA — stores up to 1 GB of binary data

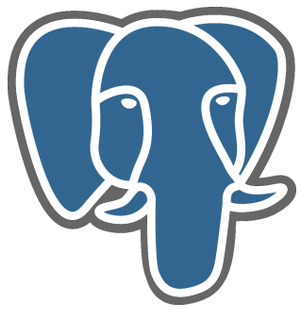




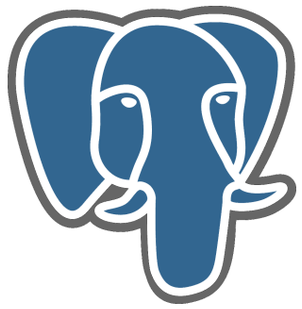
- SMALLINT — 2 bytes [- 32768; + 32767]
- INTEGER — 4 bytes [- 2147483648; + 2147483647]
- BIGINT — 8 bytes [- 9223372036854775808; + 9223372036854775807]

- DECIMAL [(p, s)] — up to 131072 digits before decimal point and 16383 after decimal point
- NUMERIC [(p, s)] ~ DECIMAL [(p, s)]

- REAL — 4 bytes
- DOUBLE PRECISION — 8 bytes



- SMALLSERIAL — 2 bytes [0; + 32767]
- SERIAL — 4 bytes [0; + 2147483647]
- BIGSERIAL — 8 bytes [0; + 9223372036854775807]
- BOOLEAN — 1 byte



- DATE
- TIME [(p)] WITH TIME ZONE
- TIME [(p)] [WITHOUT TIME ZONE]
- TIMESTAMP [(p)] WITH TIME ZONE
- TIMESTAMP [(p)] [WITHOUT TIME ZONE]
- INTERVAL [*fields*] [(p)]

- CHAR (N) / BINARY (N)
- VARCHAR (N) / VARBINARY (N)
- TINYBLOB / BLOB / MEDIUMBLOB / LONGBLOB
- TINYTEXT / TEXT / MEDIUMTEXT / LONGTEXT

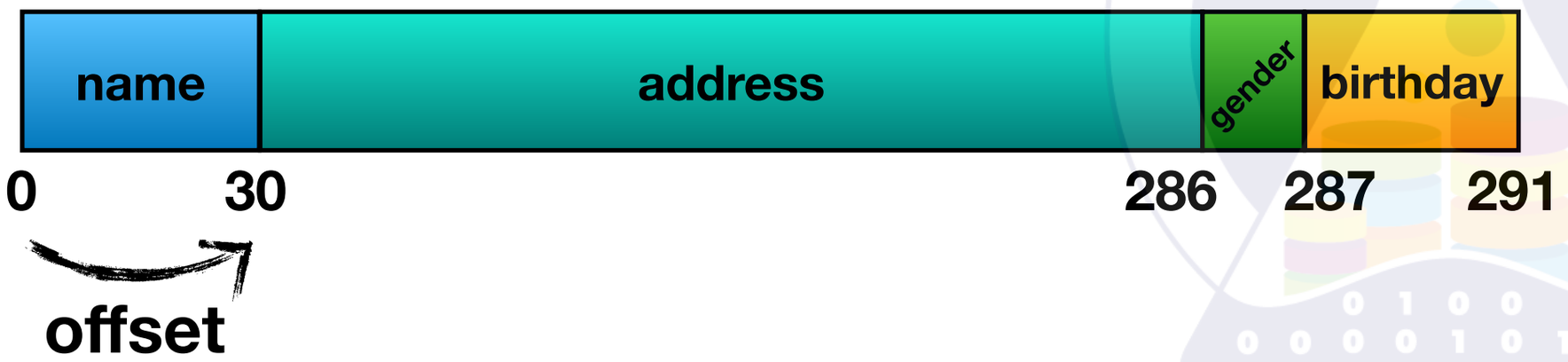
- TINYINT / SMALLINT / MEDIUMINT
- INT
- BIGINT
- DECIMAL
- FLOAT / DOUBLE
- BIT

- DATE
- DATETIME
- TIMESTAMP
- TIME
- YEAR

```
CREATE TABLE Student (
  name CHAR(30),
  address VARCHAR(255),
  gender CHAR(1)
  birthdate DATE
);
```

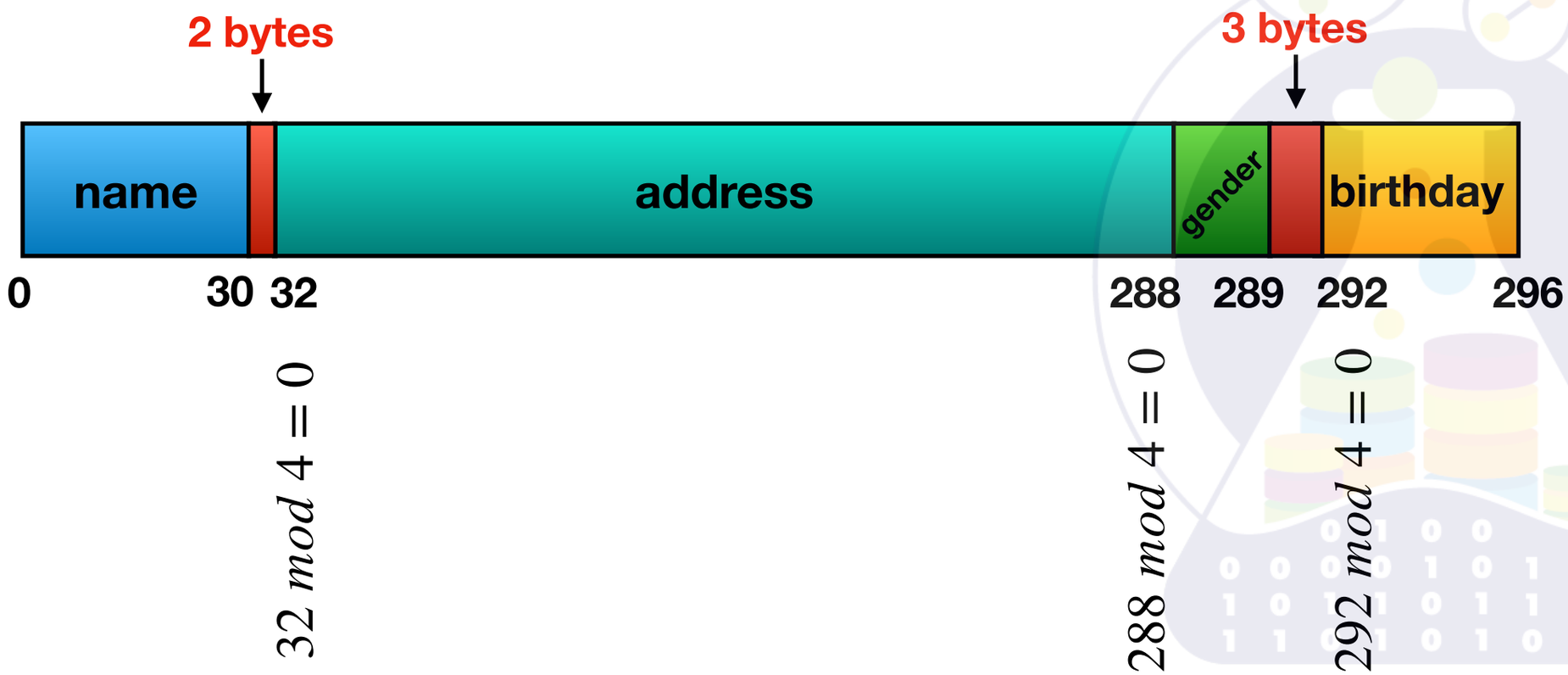
30 *bytes*
256 *bytes*
1 *byte*
4 *bytes*

$$row\ length = 30 + 256 + 1 + 4 = 291\ bytes$$



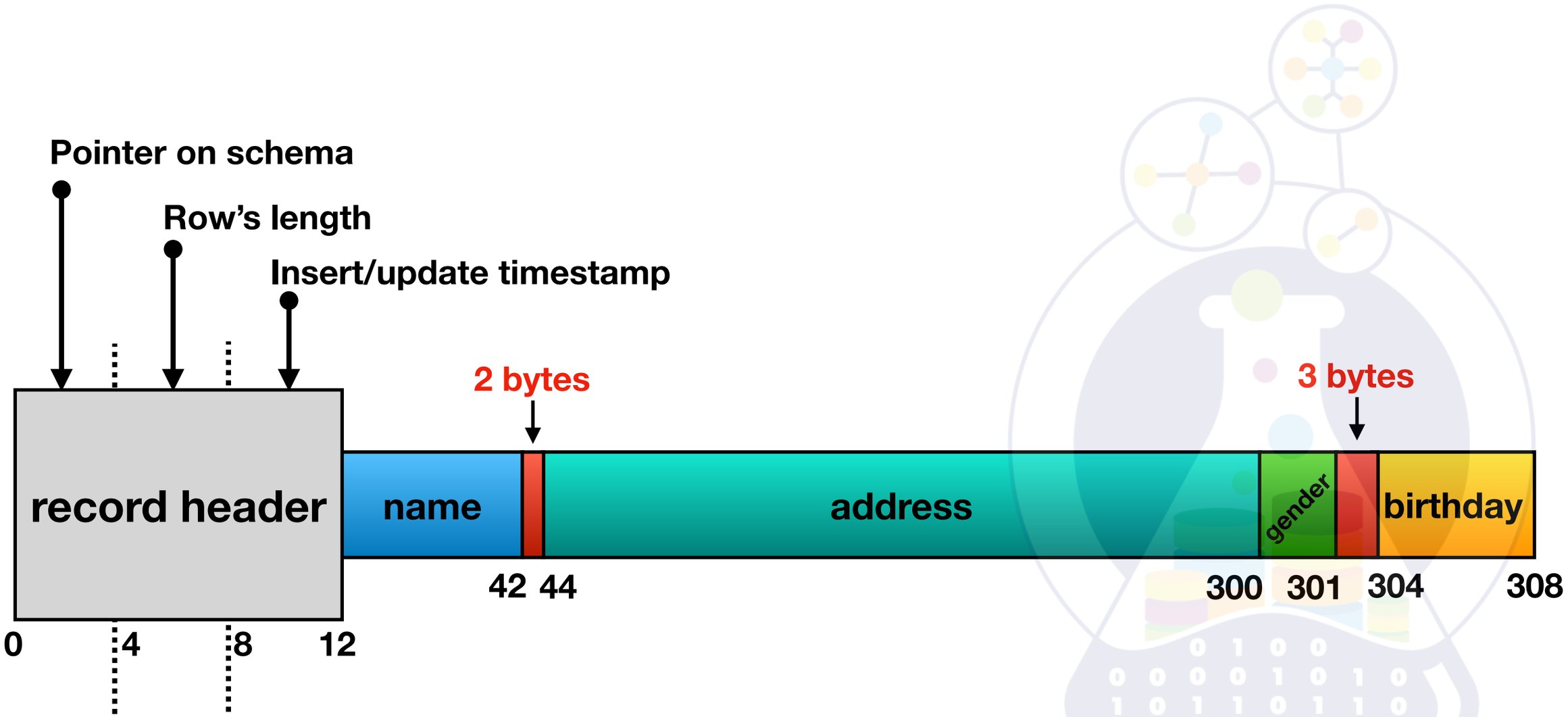
$$row\ length = 30 + 256 + 1 + 4 = 291\ bytes$$

$$row\ length = 296\ bytes\ for\ OS\ block = 4\ KB$$



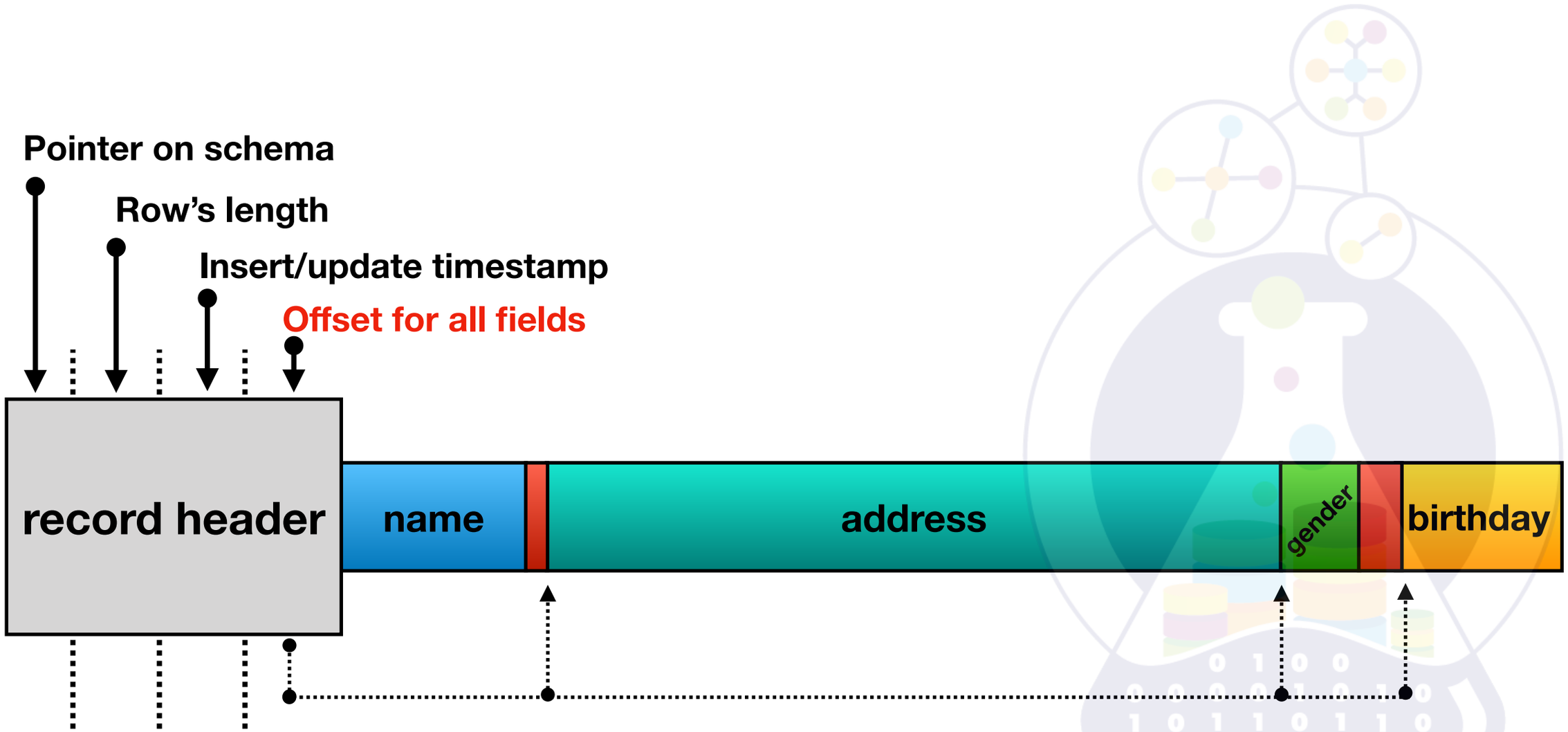
Row’s metadata ~ **record header**

- information about full row’s length
- information about last access to a row
- information about locking, transaction number



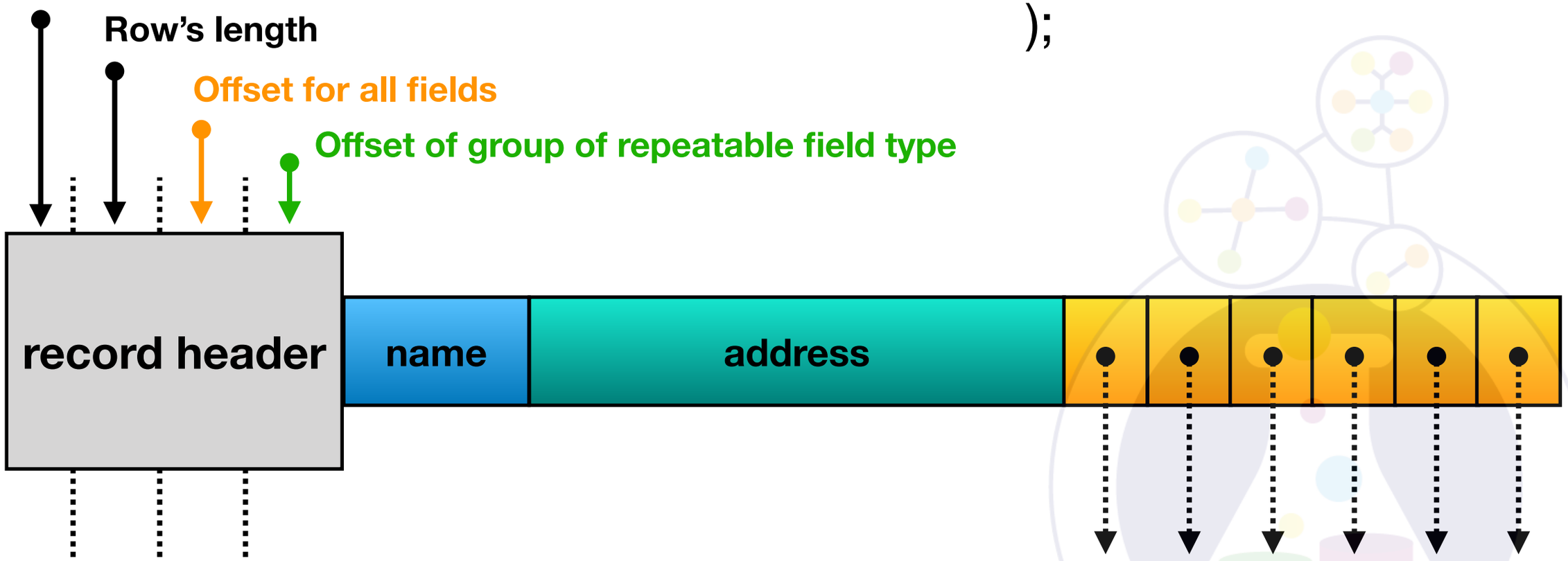
Row’s metadata ~ **record header**

- information about full row’s length
- information about last access to a row
- information about locking, transaction number
- information about offsets for all fields



Case #1

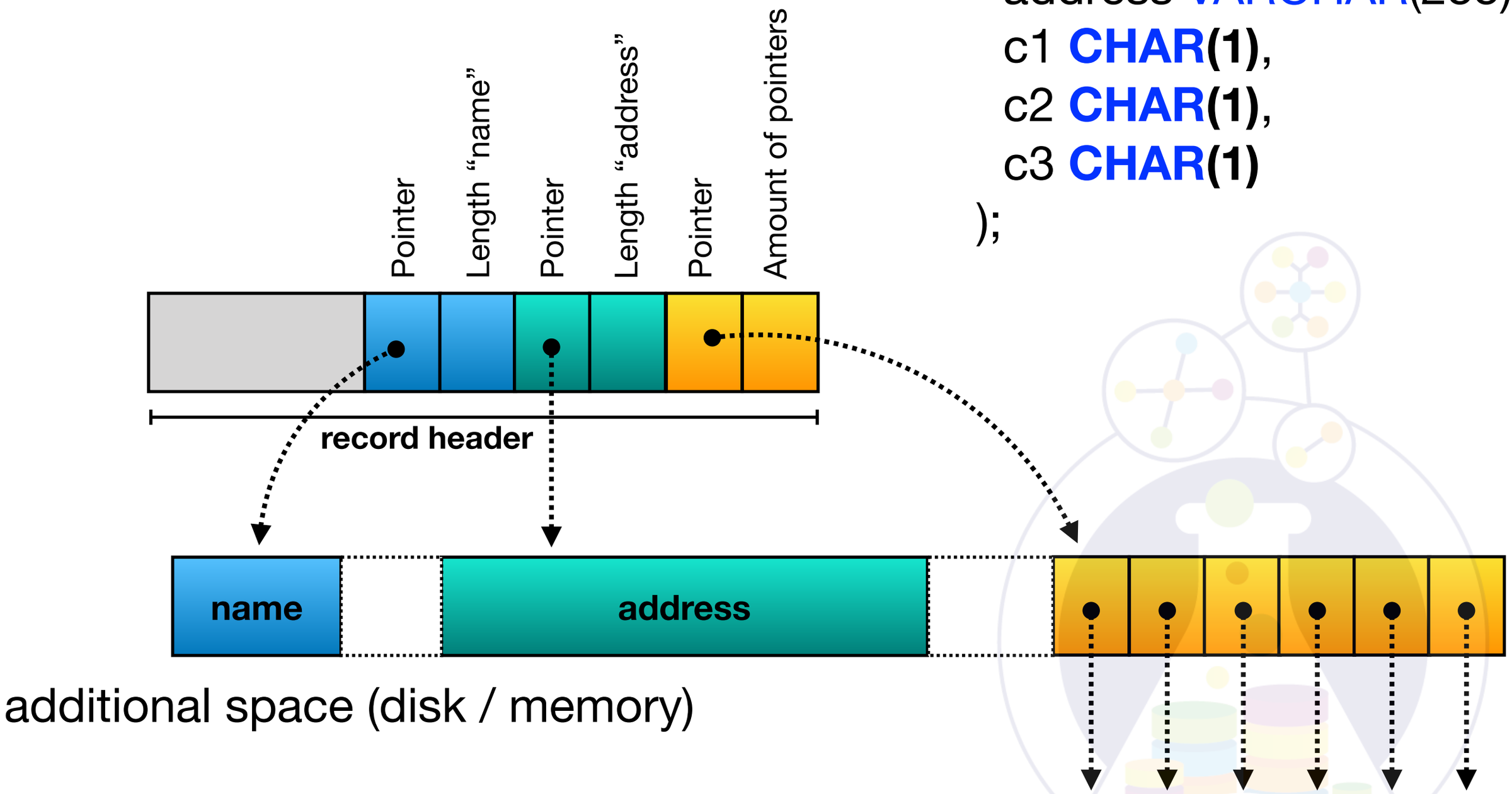
```
CREATE TABLE Student (
  name CHAR(30),
  address VARCHAR(255),
  c1 CHAR(1),
  c2 CHAR(1),
  c3 CHAR(1)
);
```

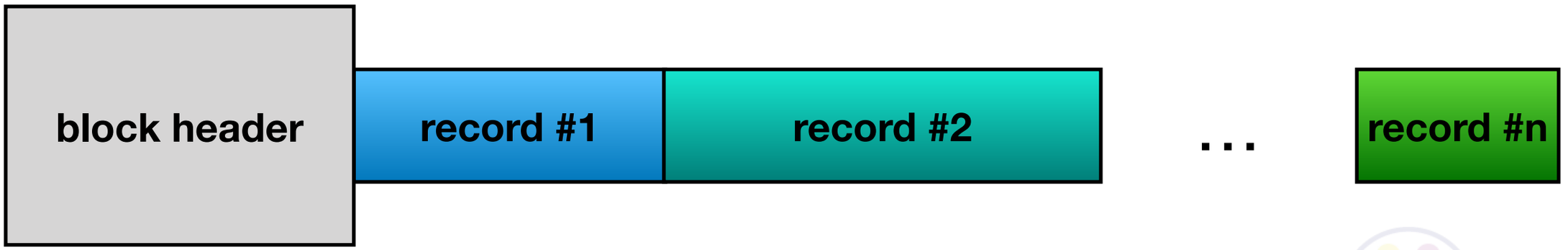


Pointers on attributes with the same type *F*

Case #2

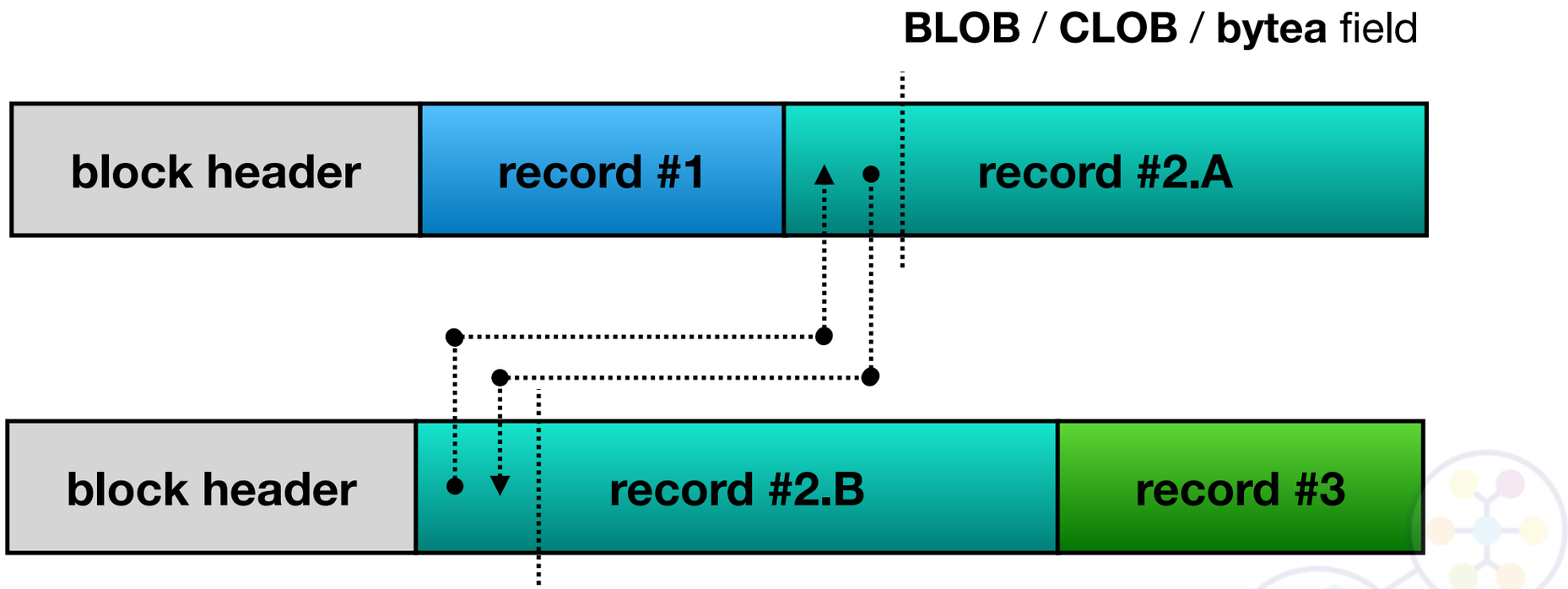
```
CREATE TABLE Student (
  name CHAR(30),
  address VARCHAR(255),
  c1 CHAR(1),
  c2 CHAR(1),
  c3 CHAR(1)
);
```





Block’s metadata ~ **block header**

- information about relation inside a block
- information about offset for each record
- information about block ID
- information about last access timestamp to a block

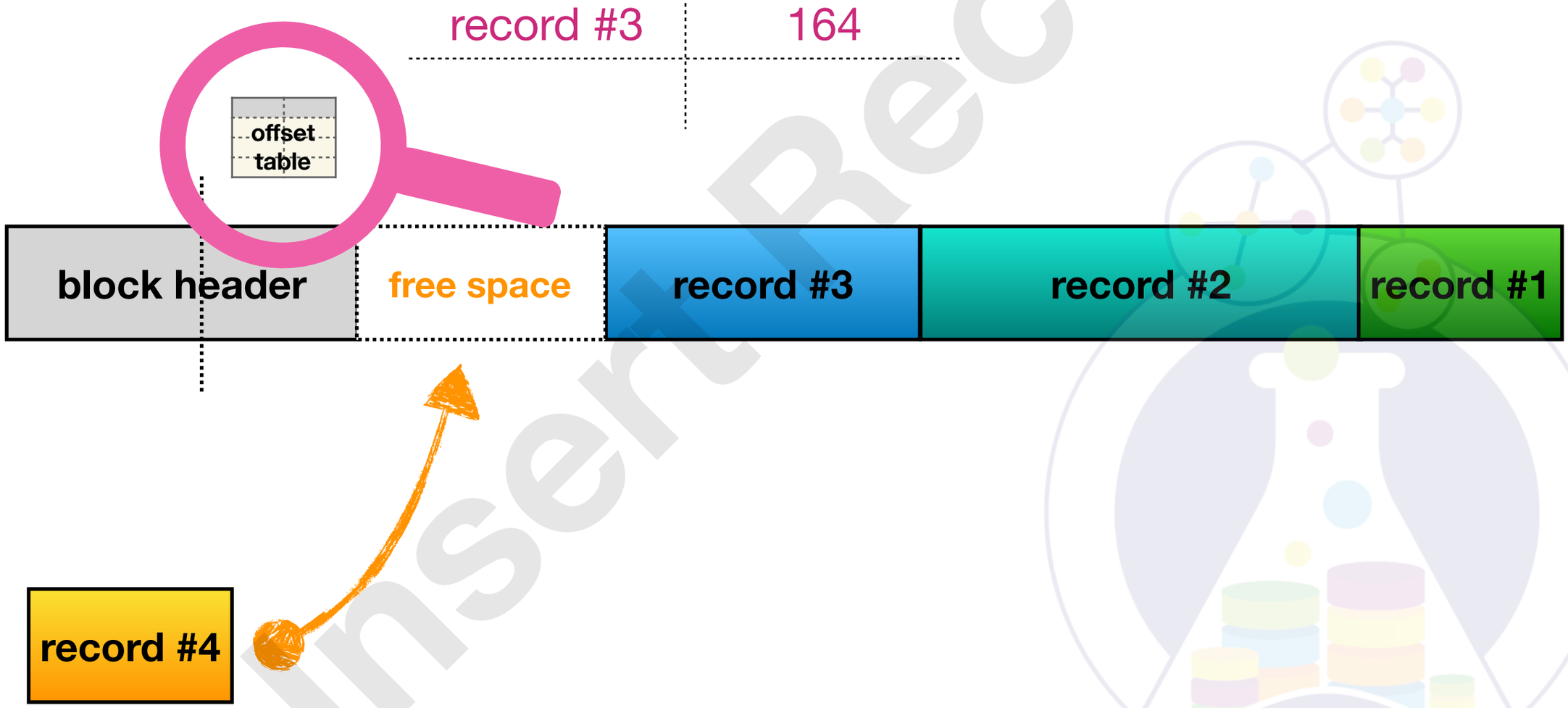


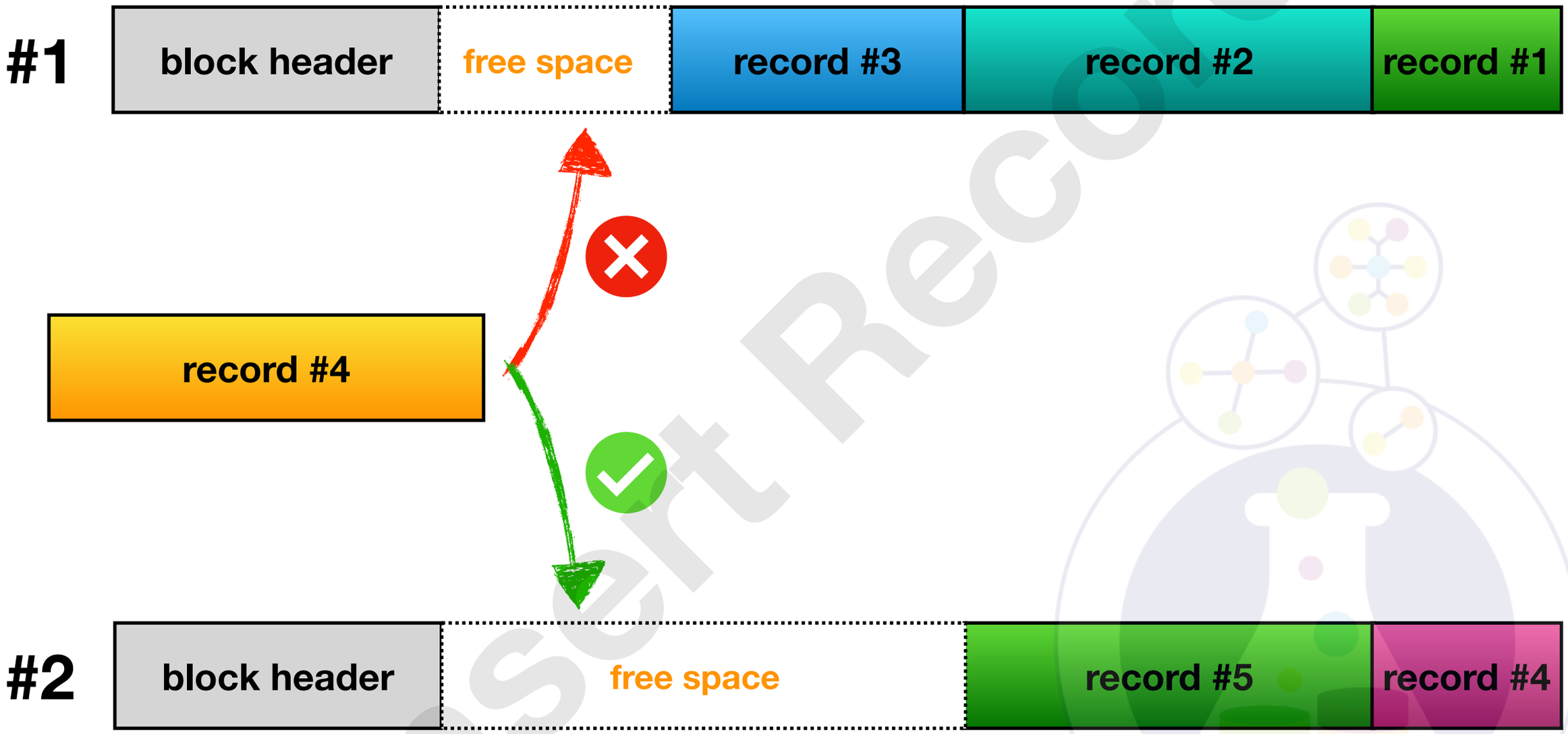
Row’s metadata ~ **record header**

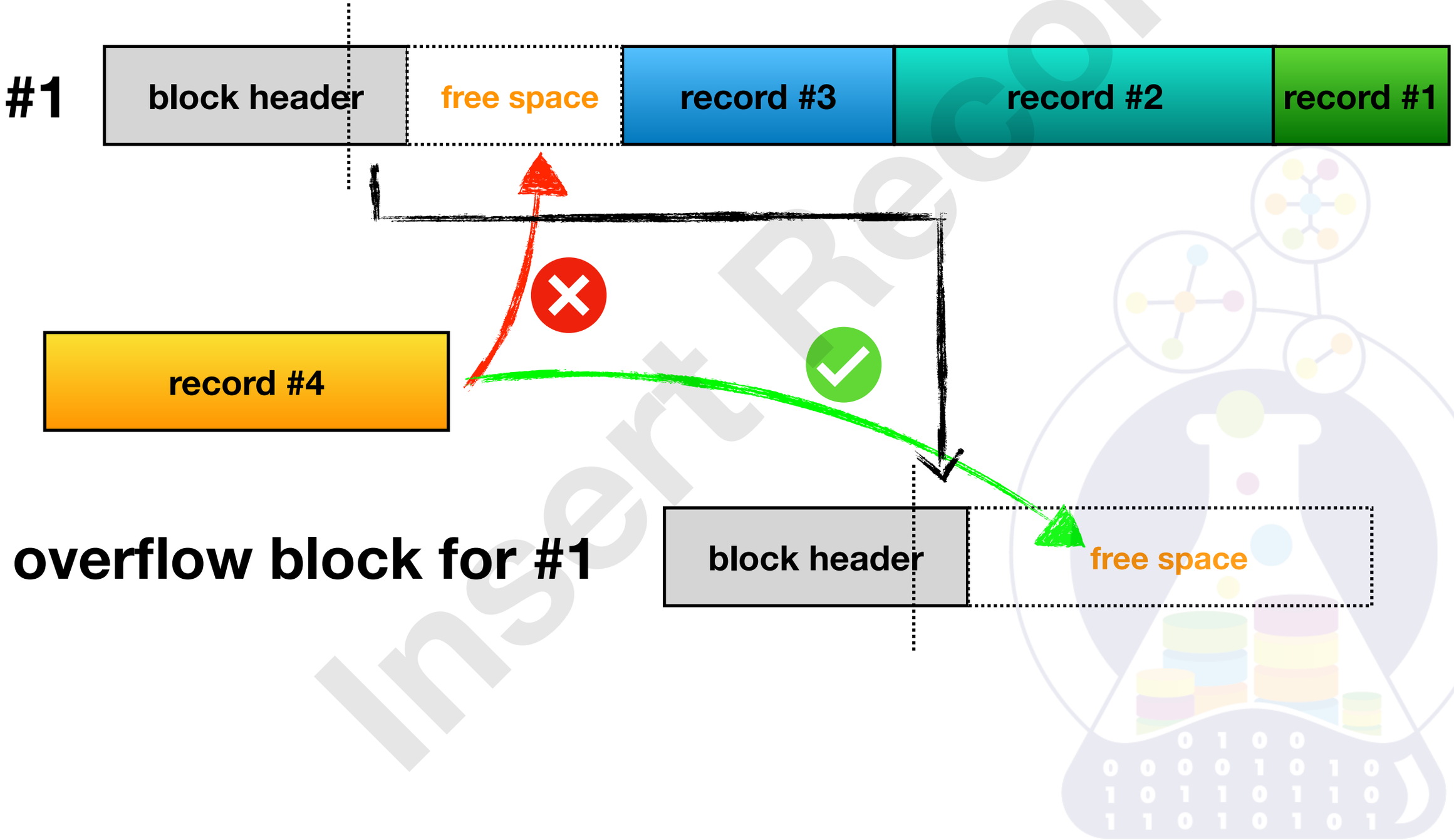
- information about is this row is a row’s fragment
- information about ordering number in a branch of fragments
- information about pointers on the next/previous fragments

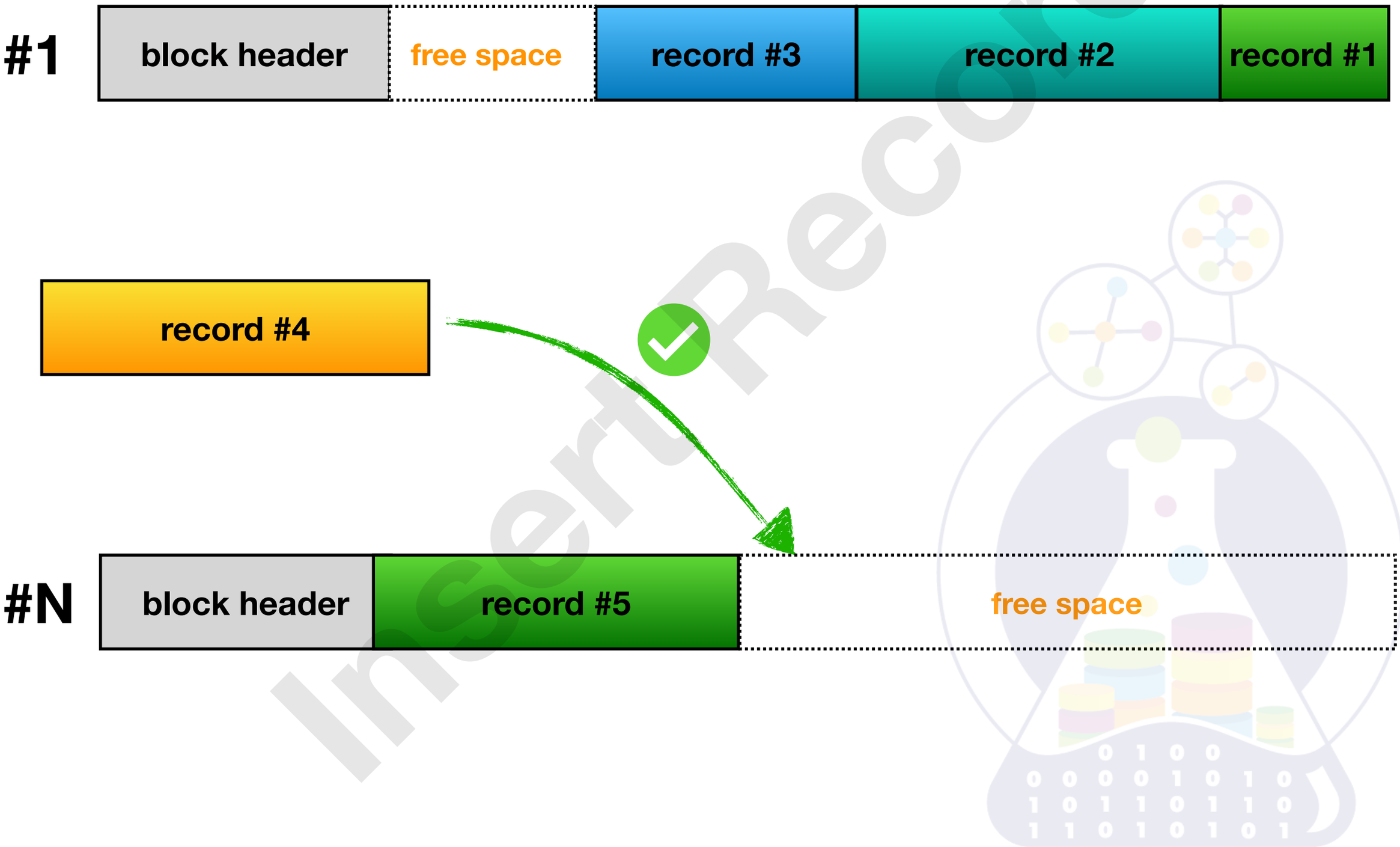
offset table

record	offset
record #1	25
record #2	82
record #3	164



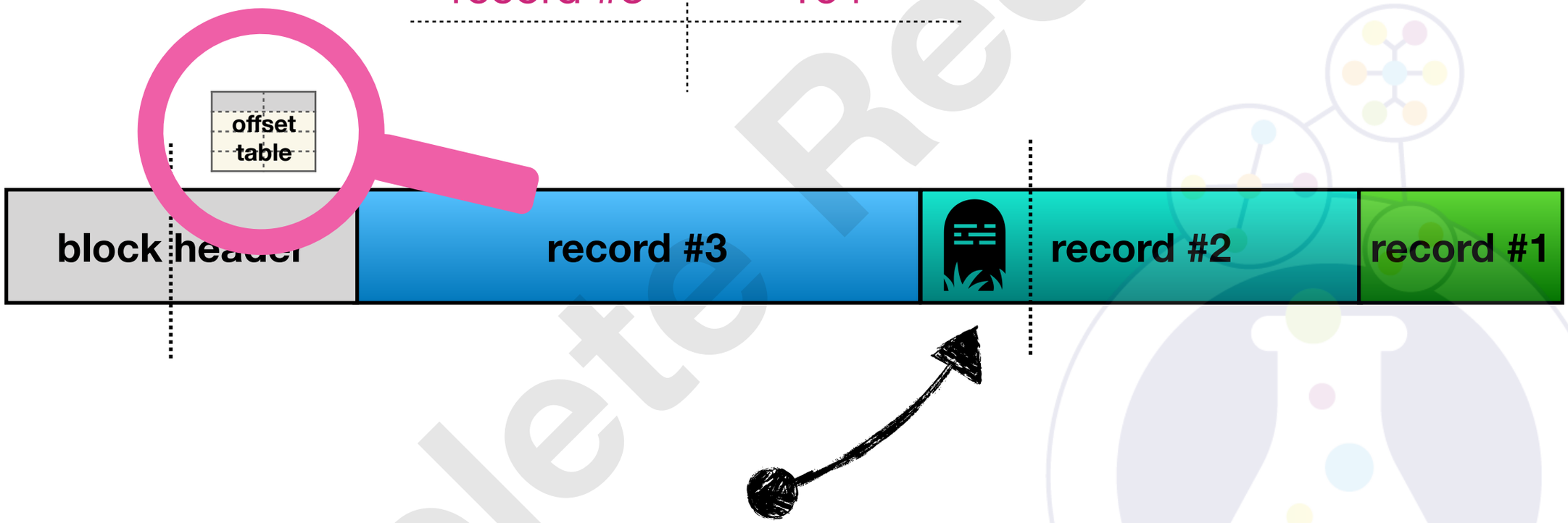




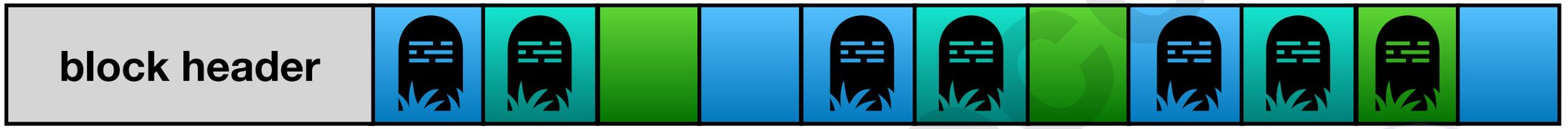


offset table

record	offset
record #1	25
record #2	82
record #3	164



deleted record with status ***Tombstone***



Azat Yakupov

Physical Addresses (have a huge size)

PC address for distributed system

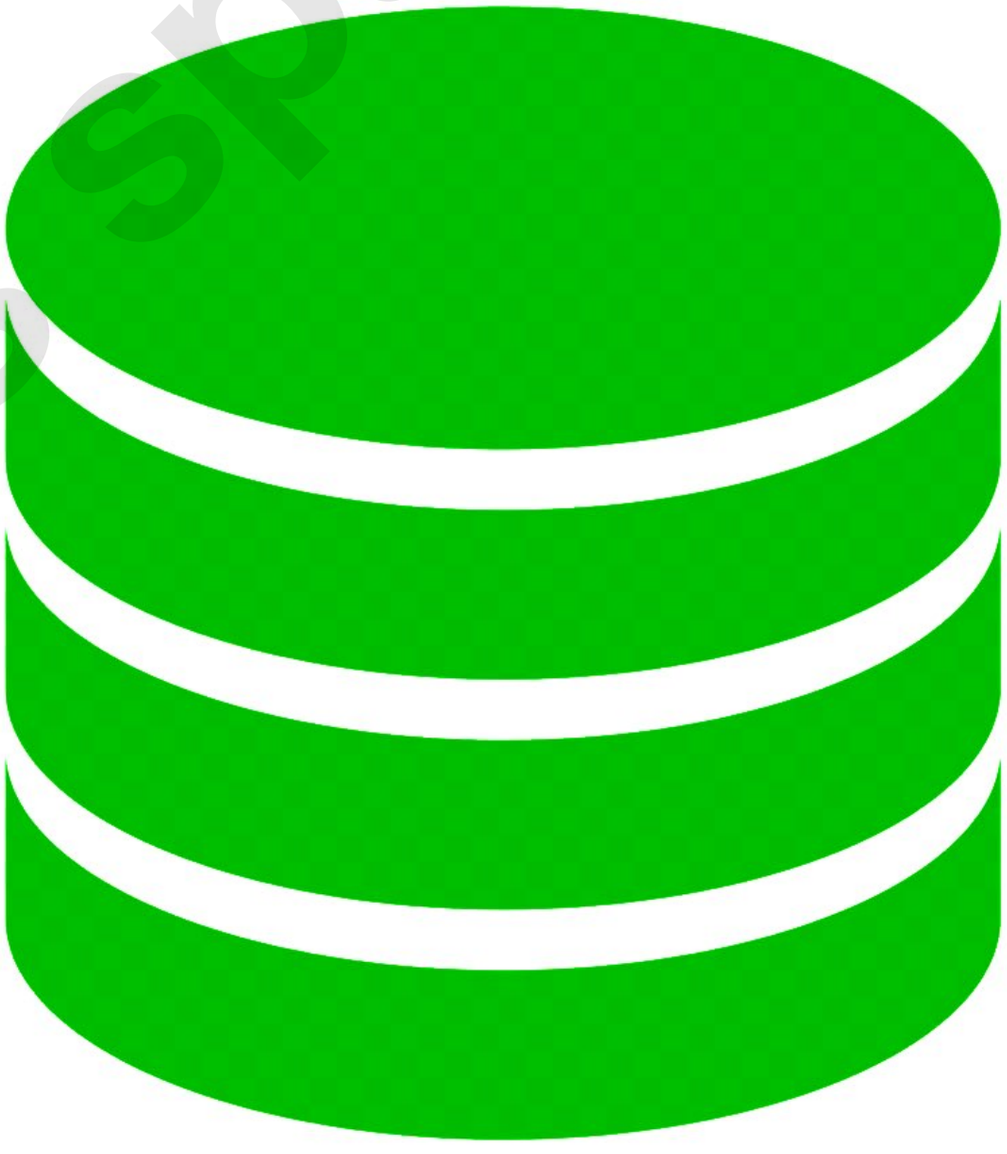
ID disk with data block

Number of disk's cylinder

Number of disk's track

Number of track's block

Offset of the 1st byte of record

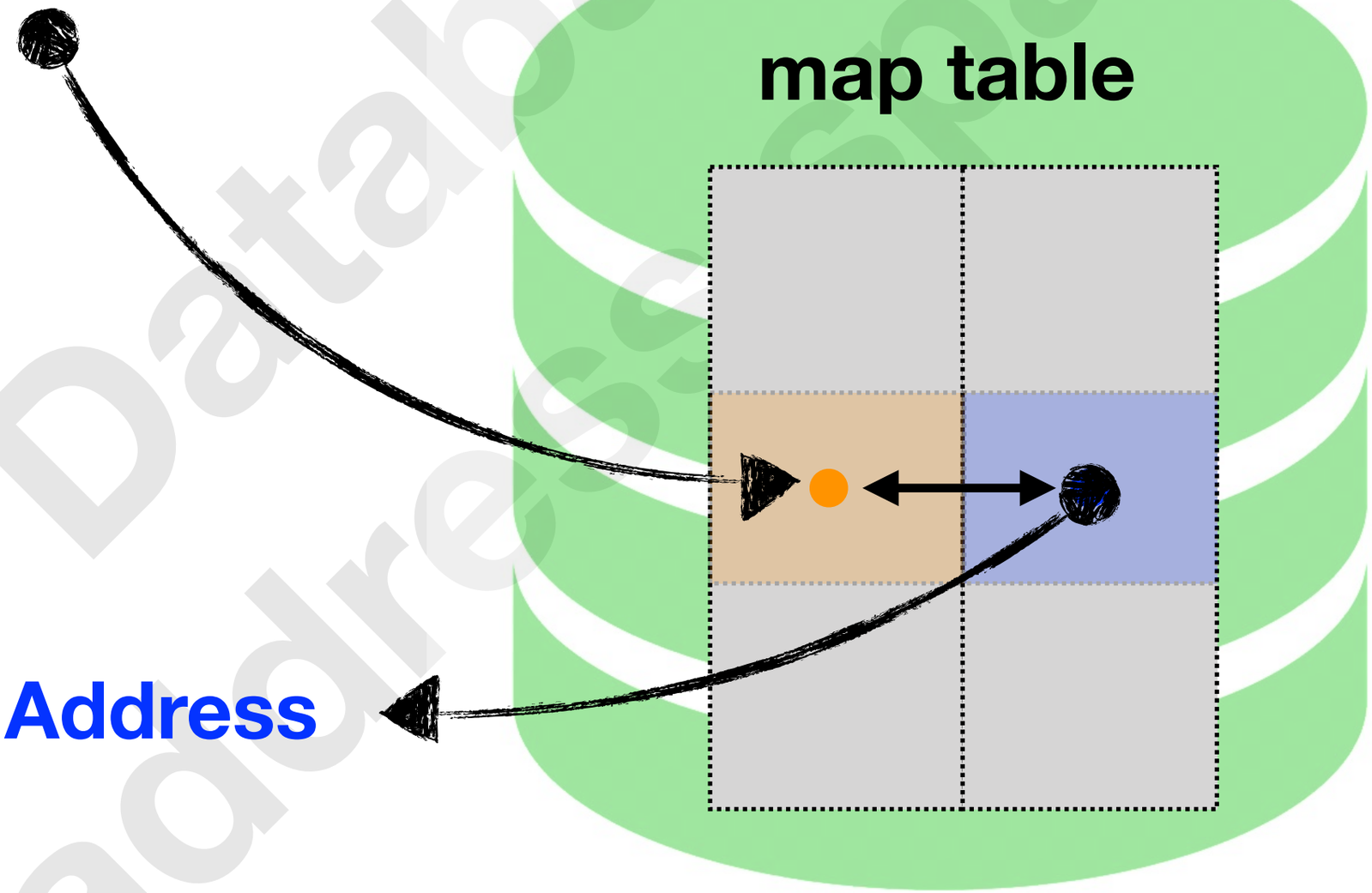


Logical Addresses (have a good size)

Each block ~ sequence of bytes

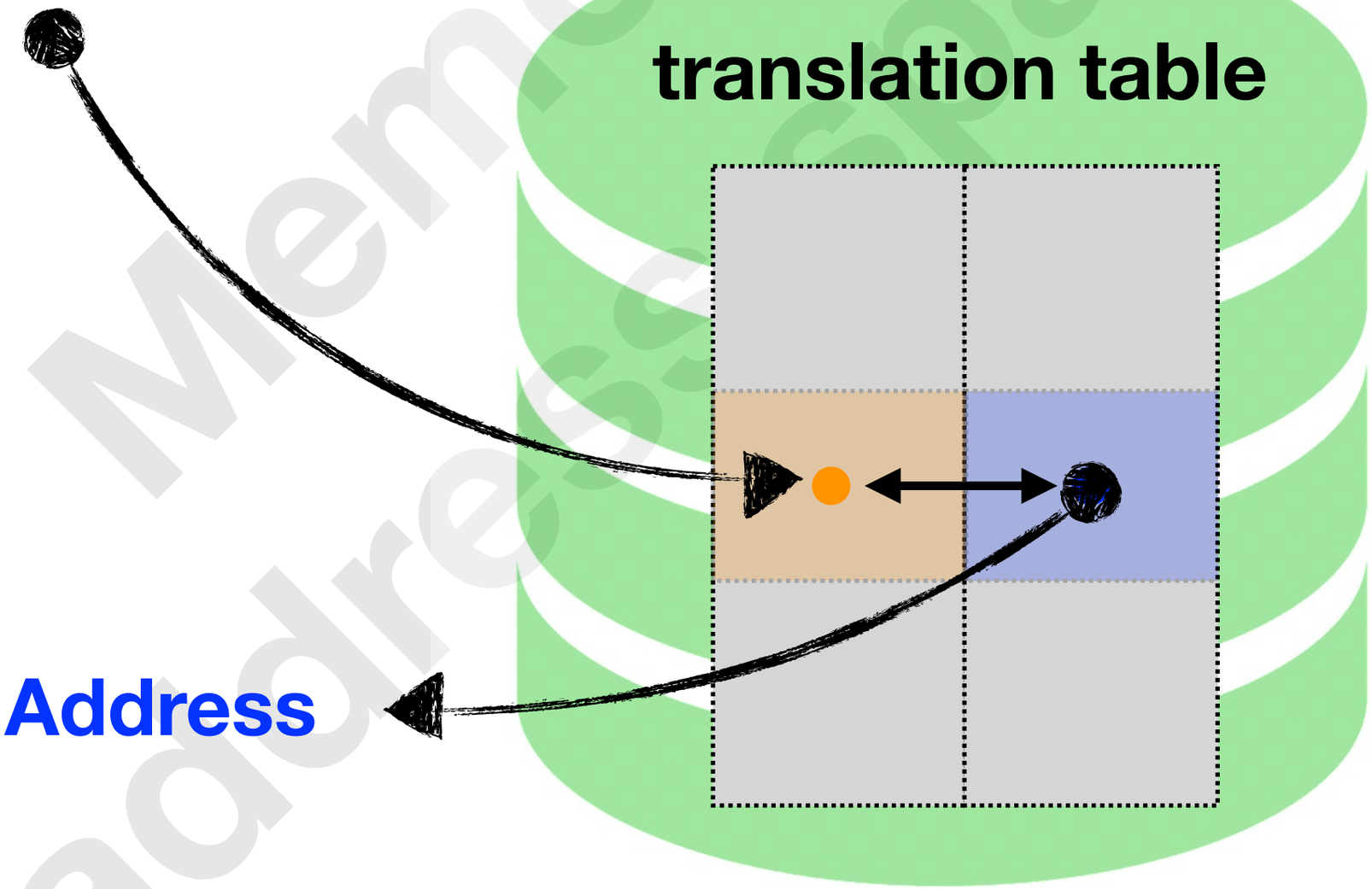
Each record ~ sequence of bytes

Logical Address

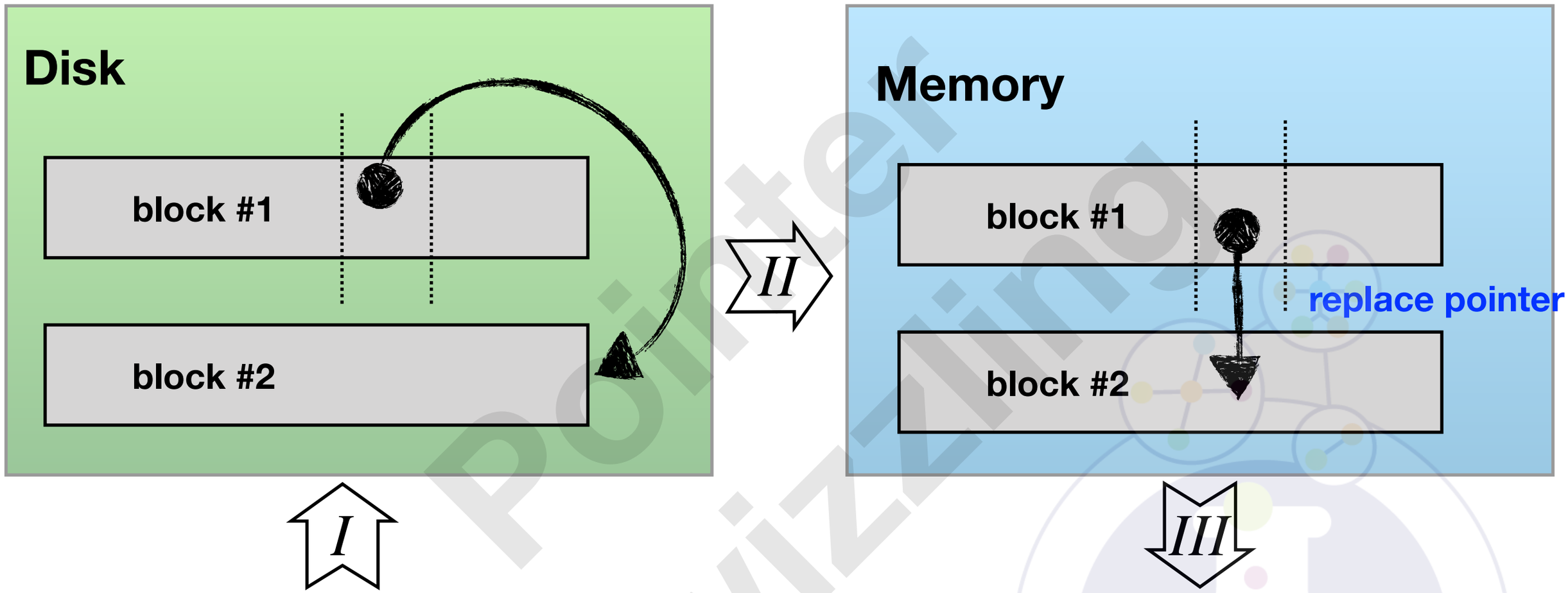


Physical Address

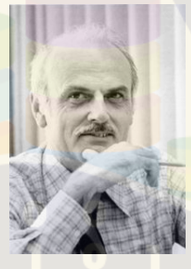
Database Address

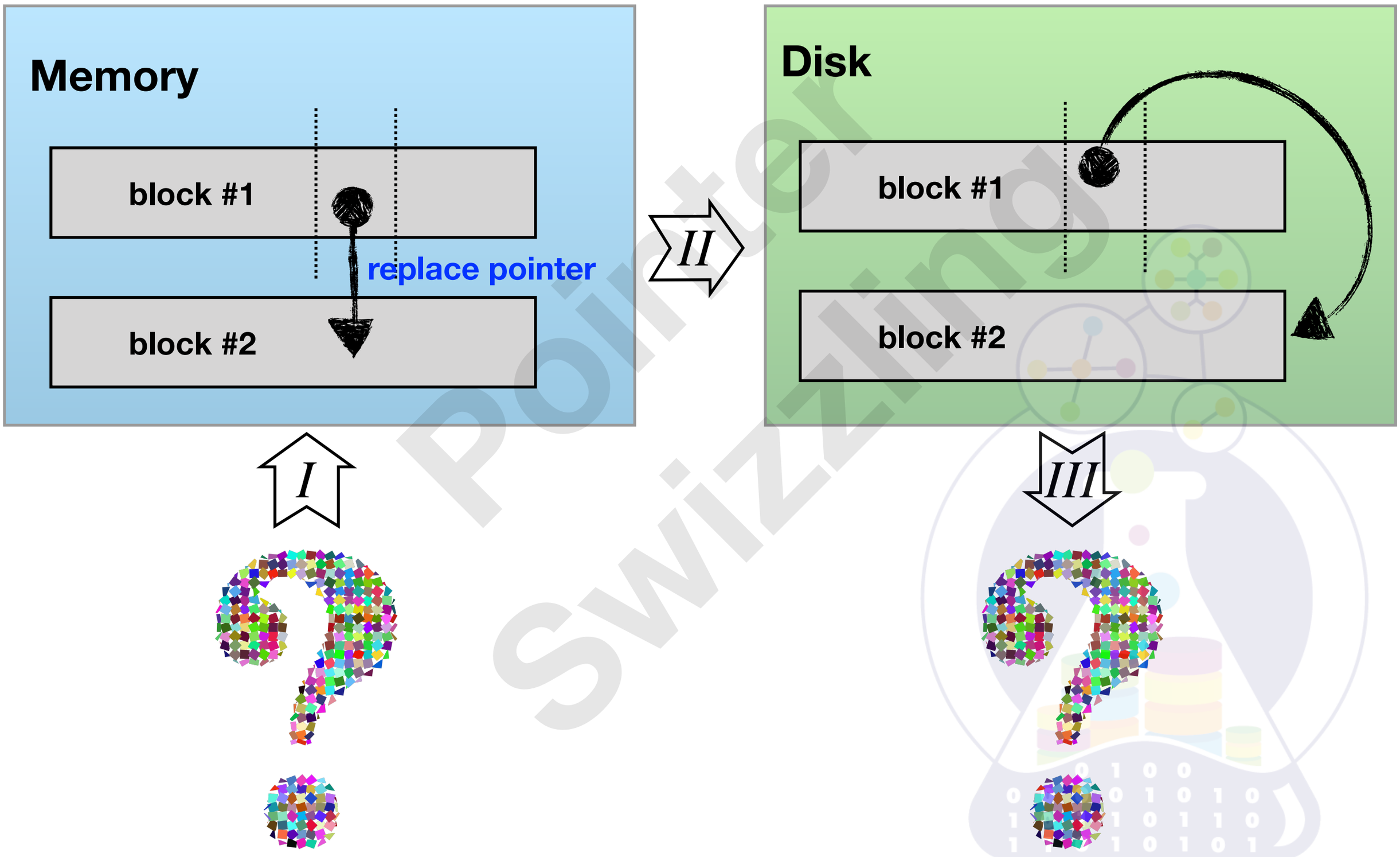


Memory Address

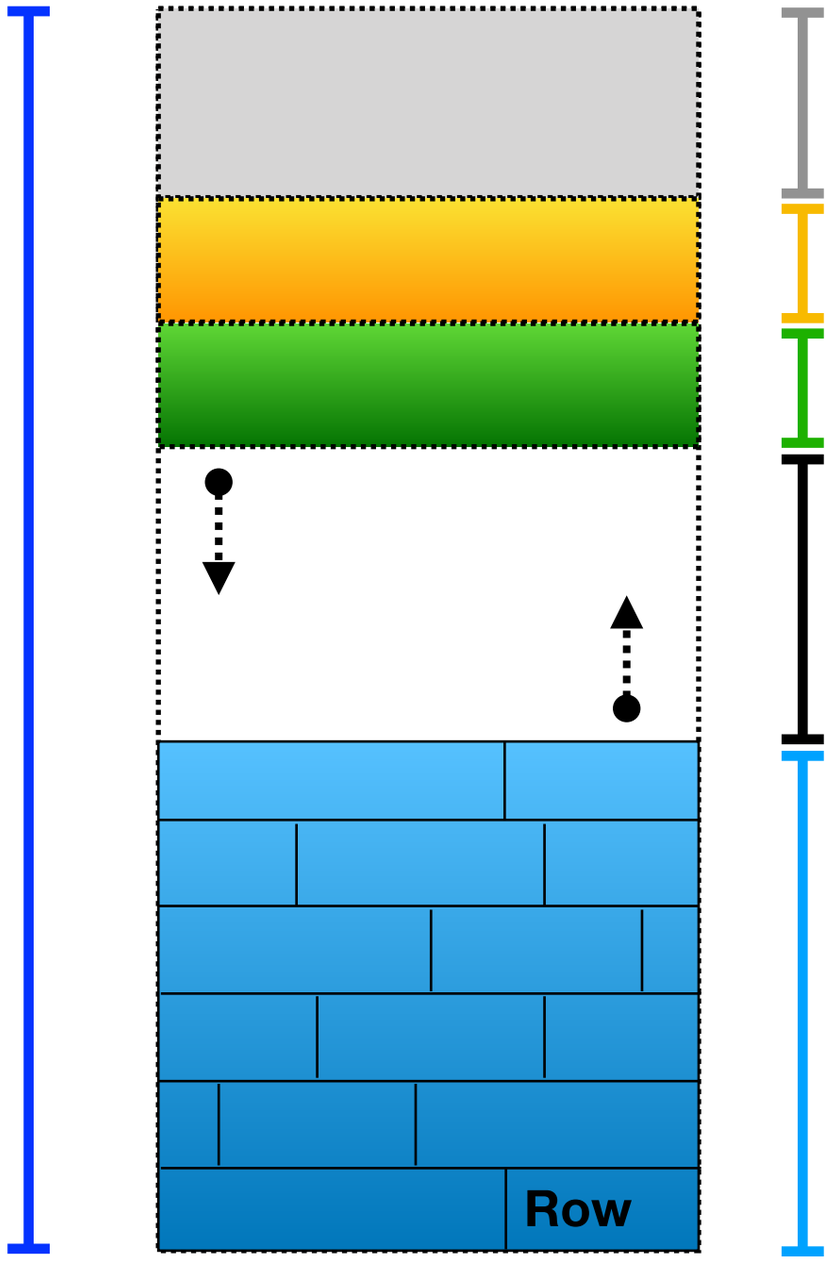



```
SELECT *
   FROM Scientists
WHERE ID = 2;
```

ID	Name	Photo
2	Edgar Frank Codd	



ORACLE



Common and Variable Header

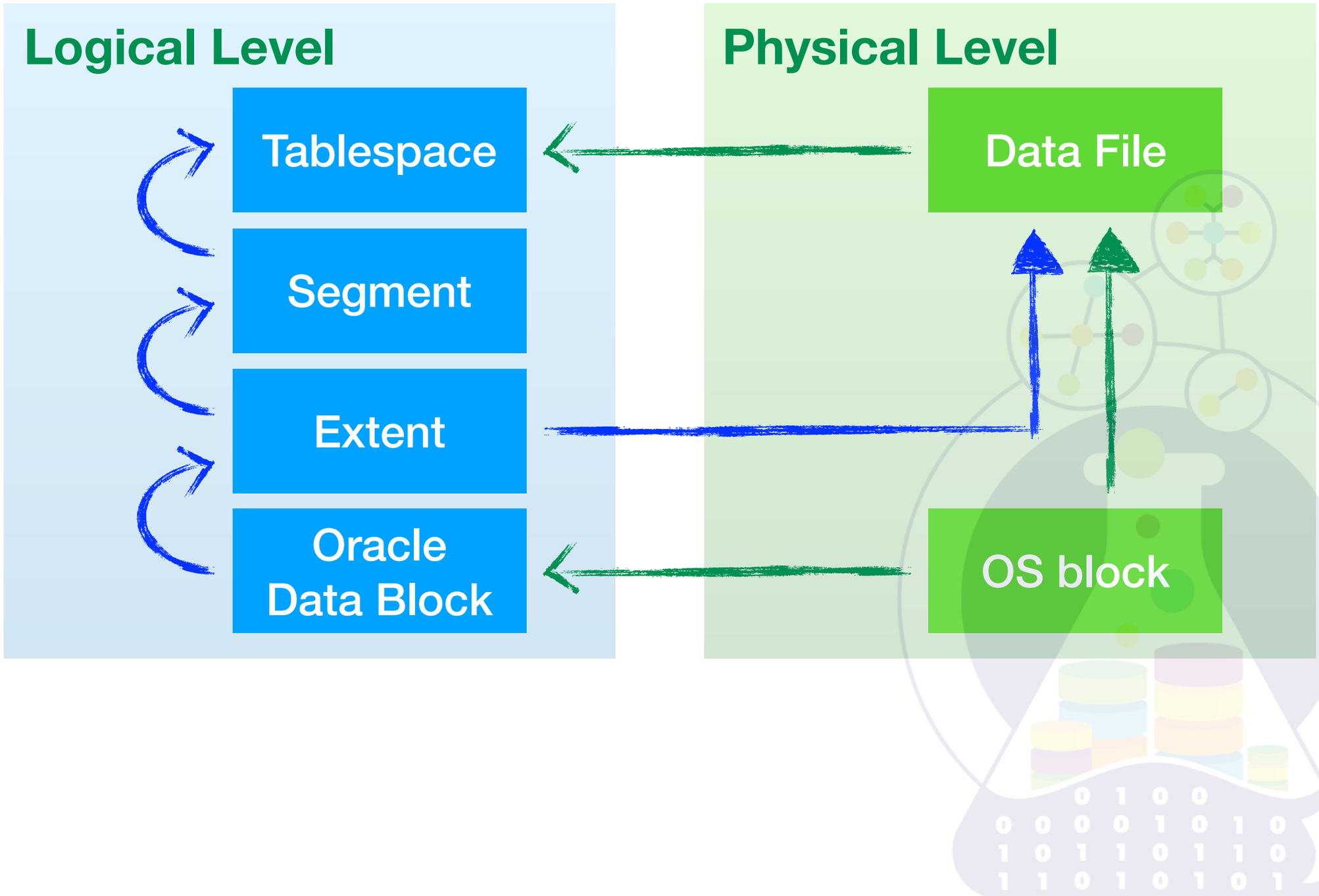
Table Directory

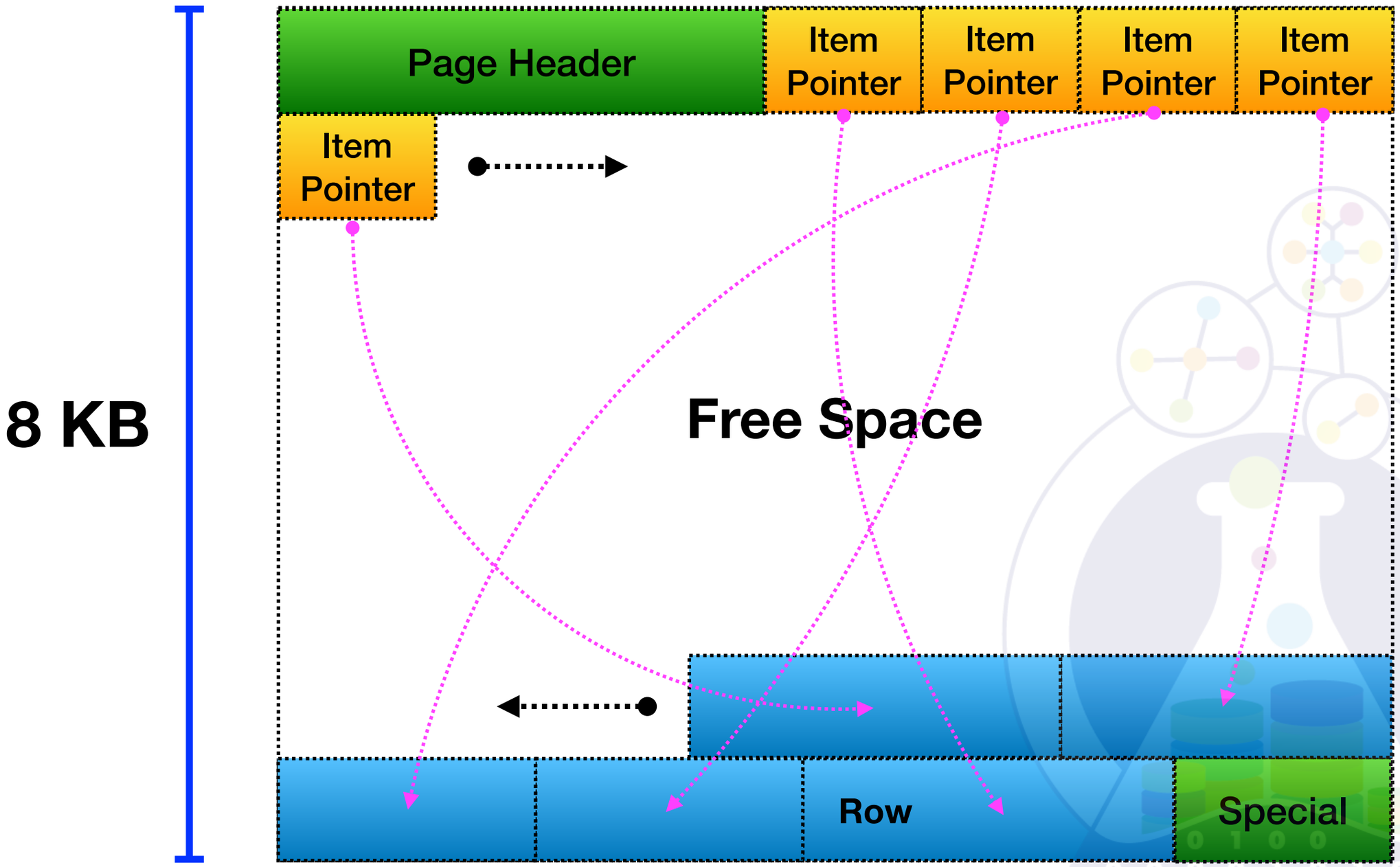
Row Directory

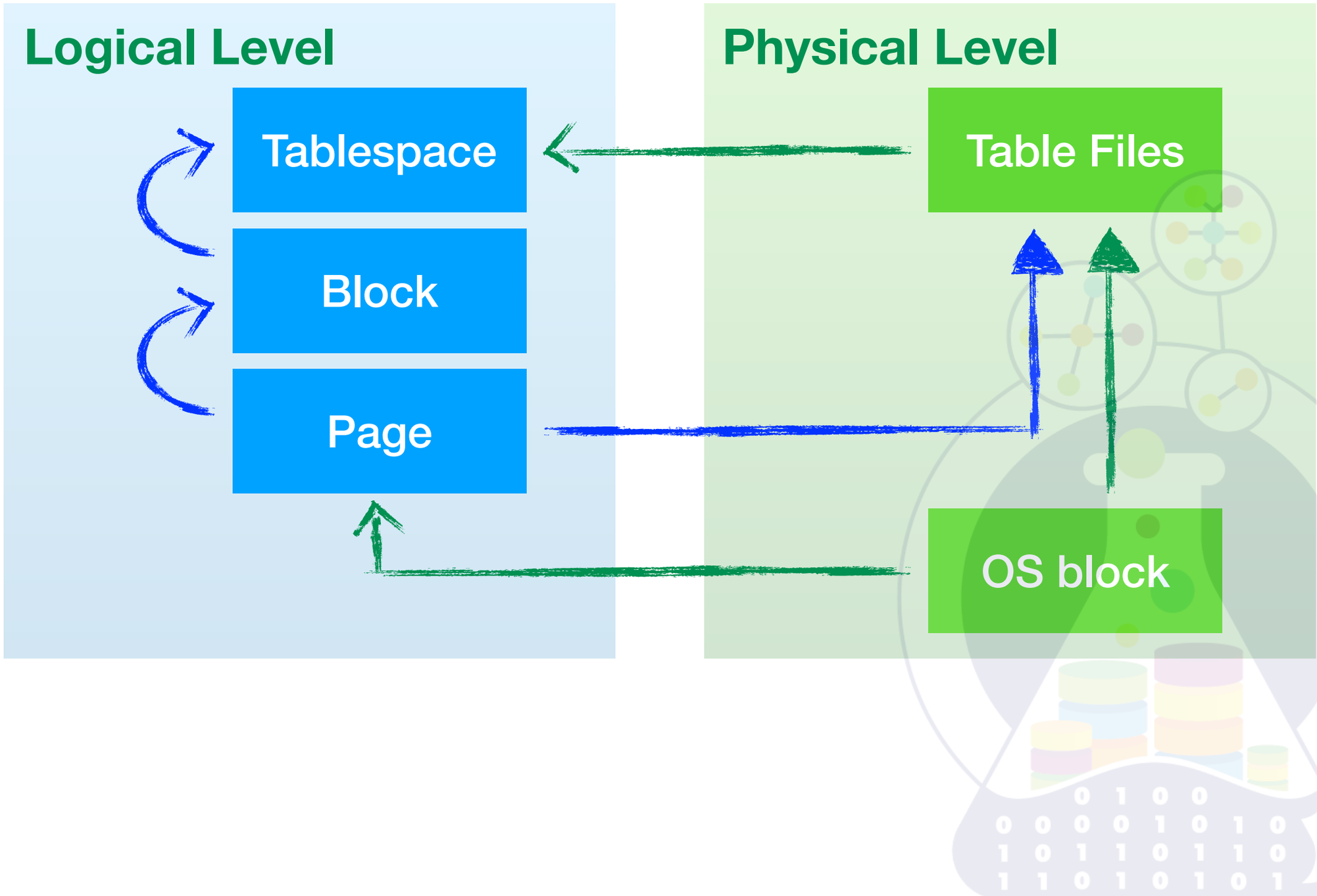
Free Space

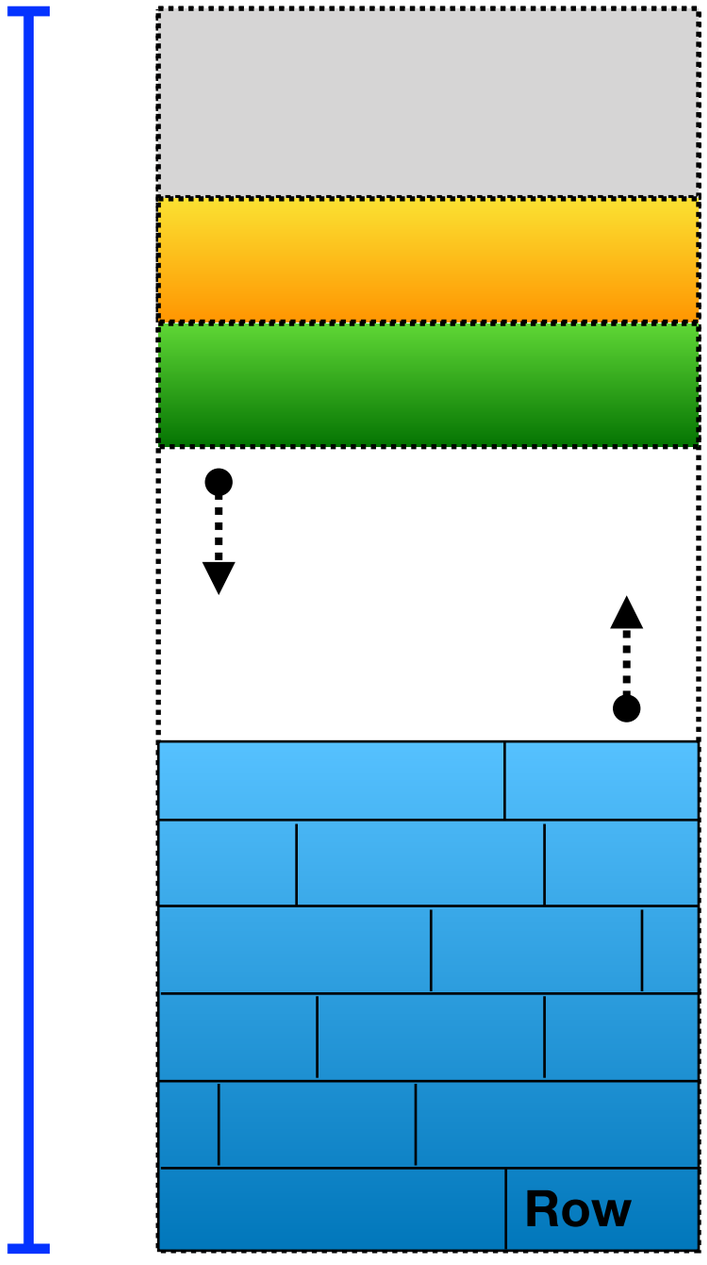
User Data

ORACLE









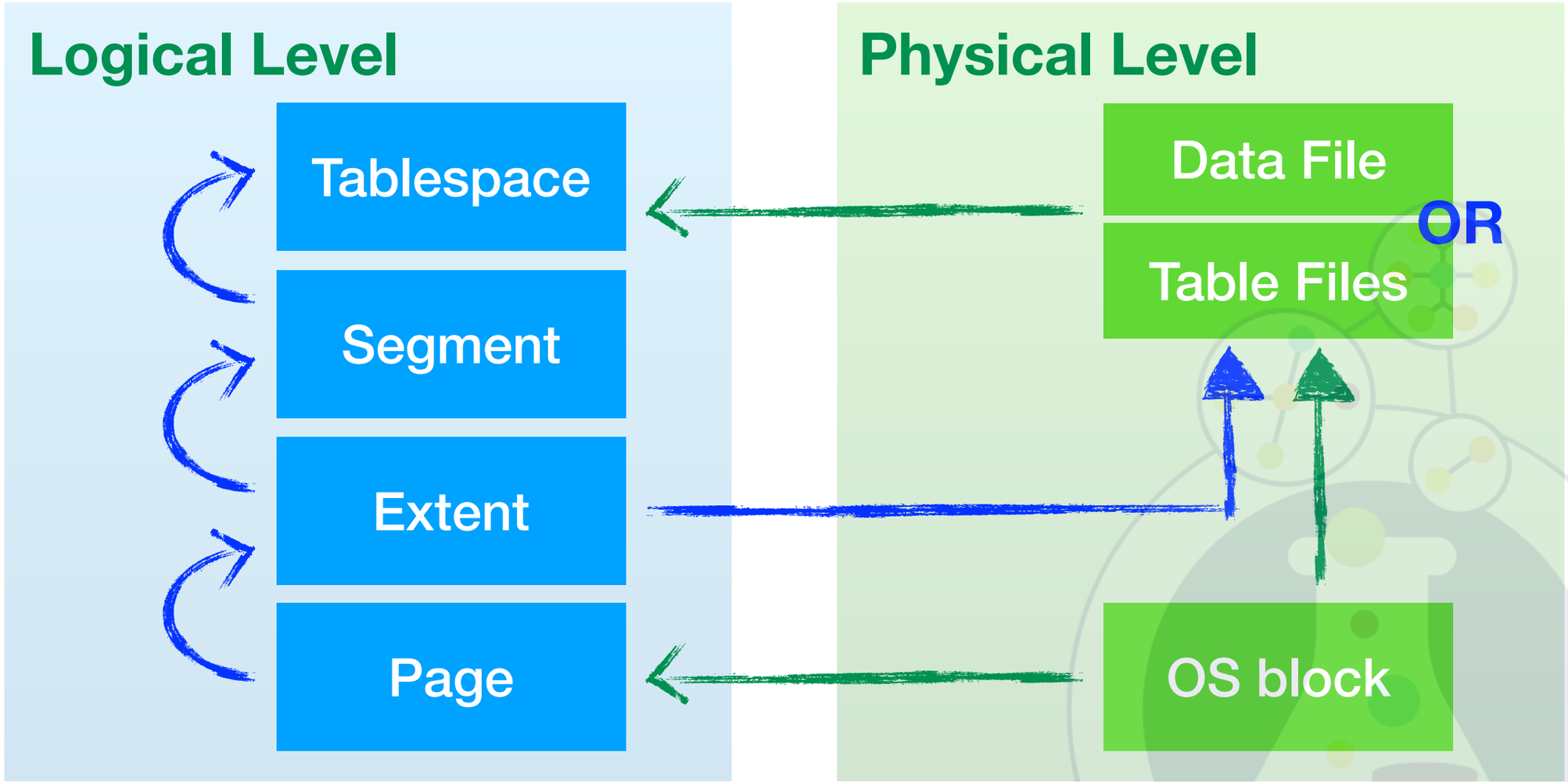
Common and Variable Header

Table Directory

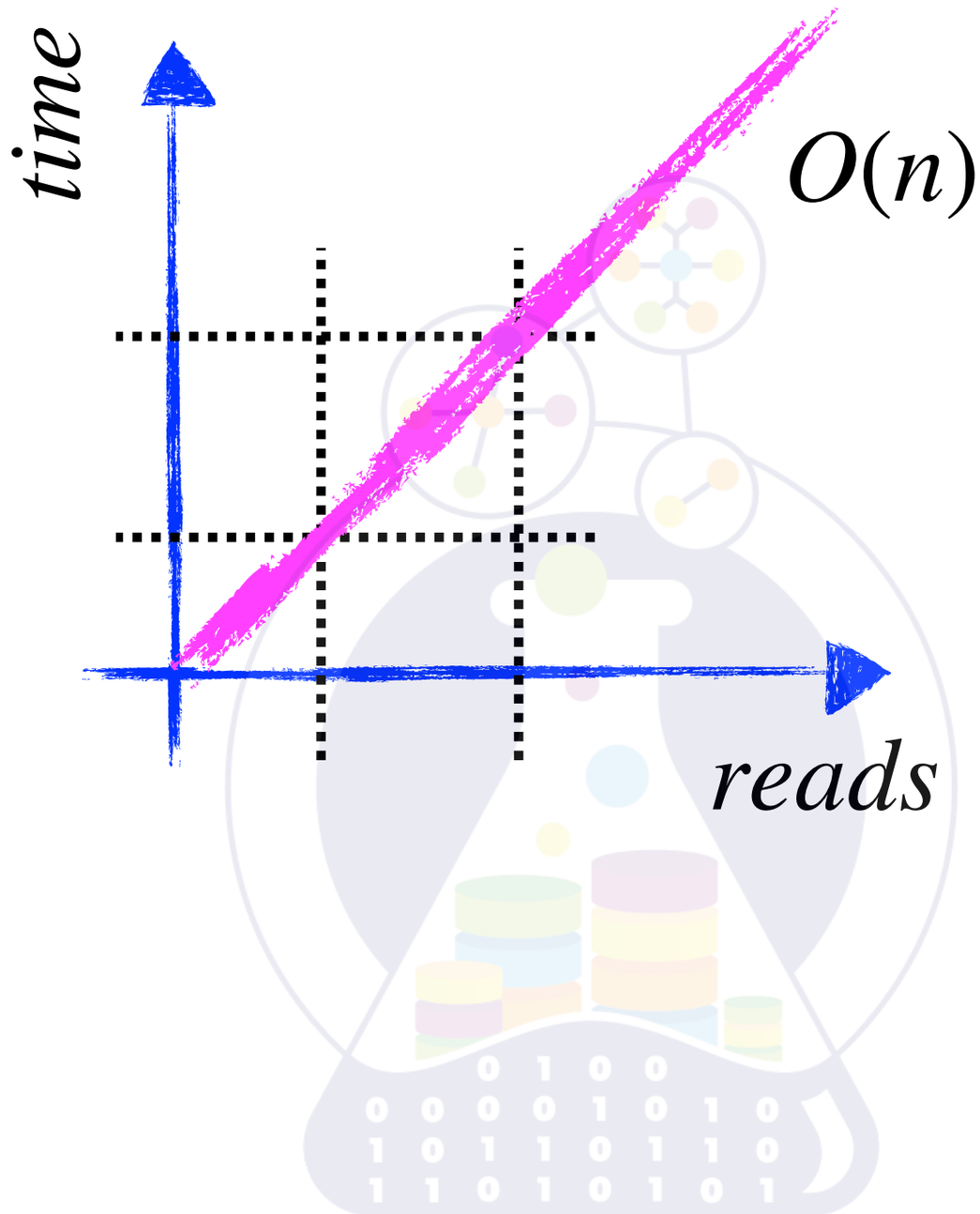
Row Directory

Free Space

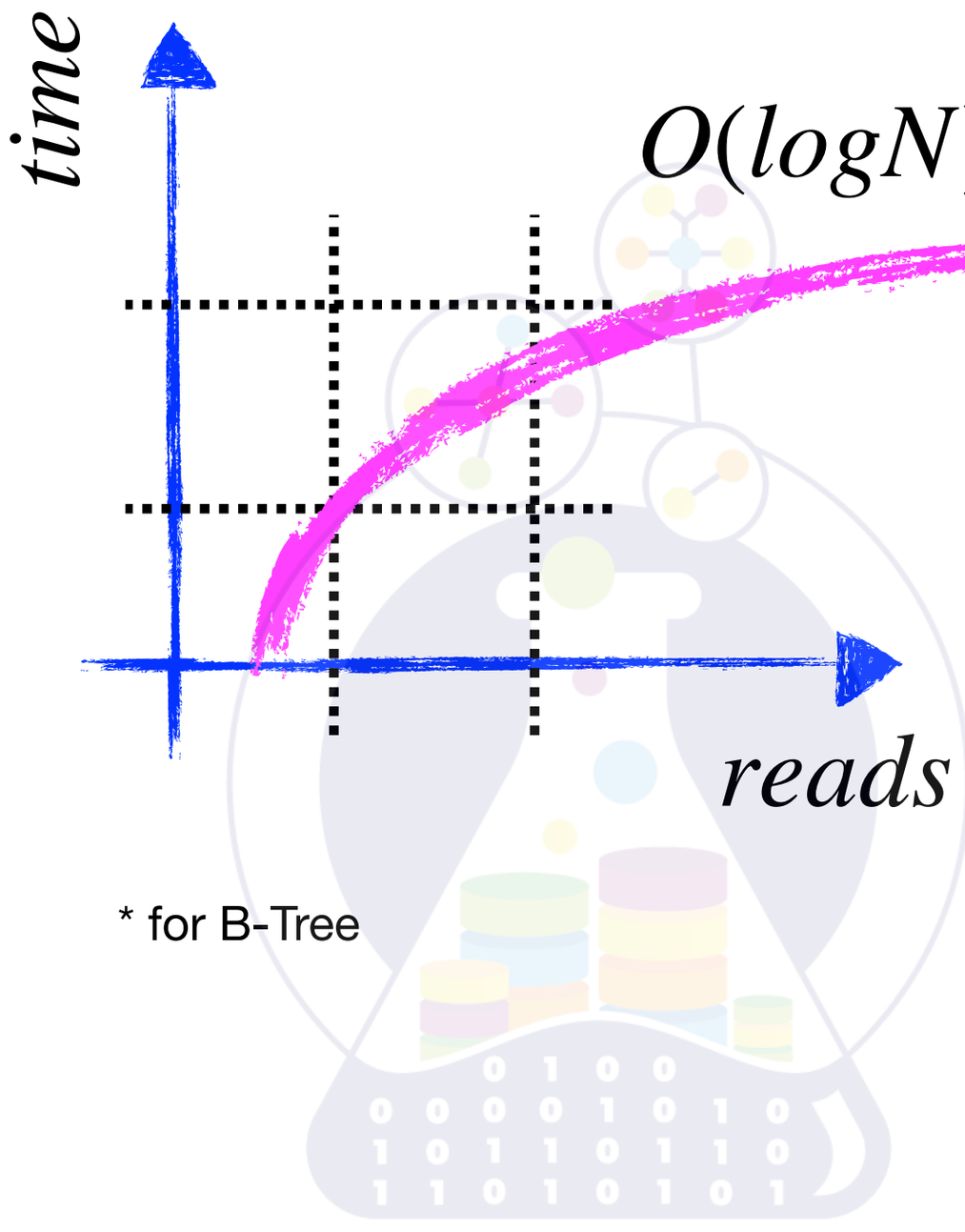
User Data



```
SELECT ID, NAME
  FROM Employee
 WHERE  ID = 100;
```



```
SELECT ID, NAME
  FROM Employee
 WHERE  ID = 100;
```

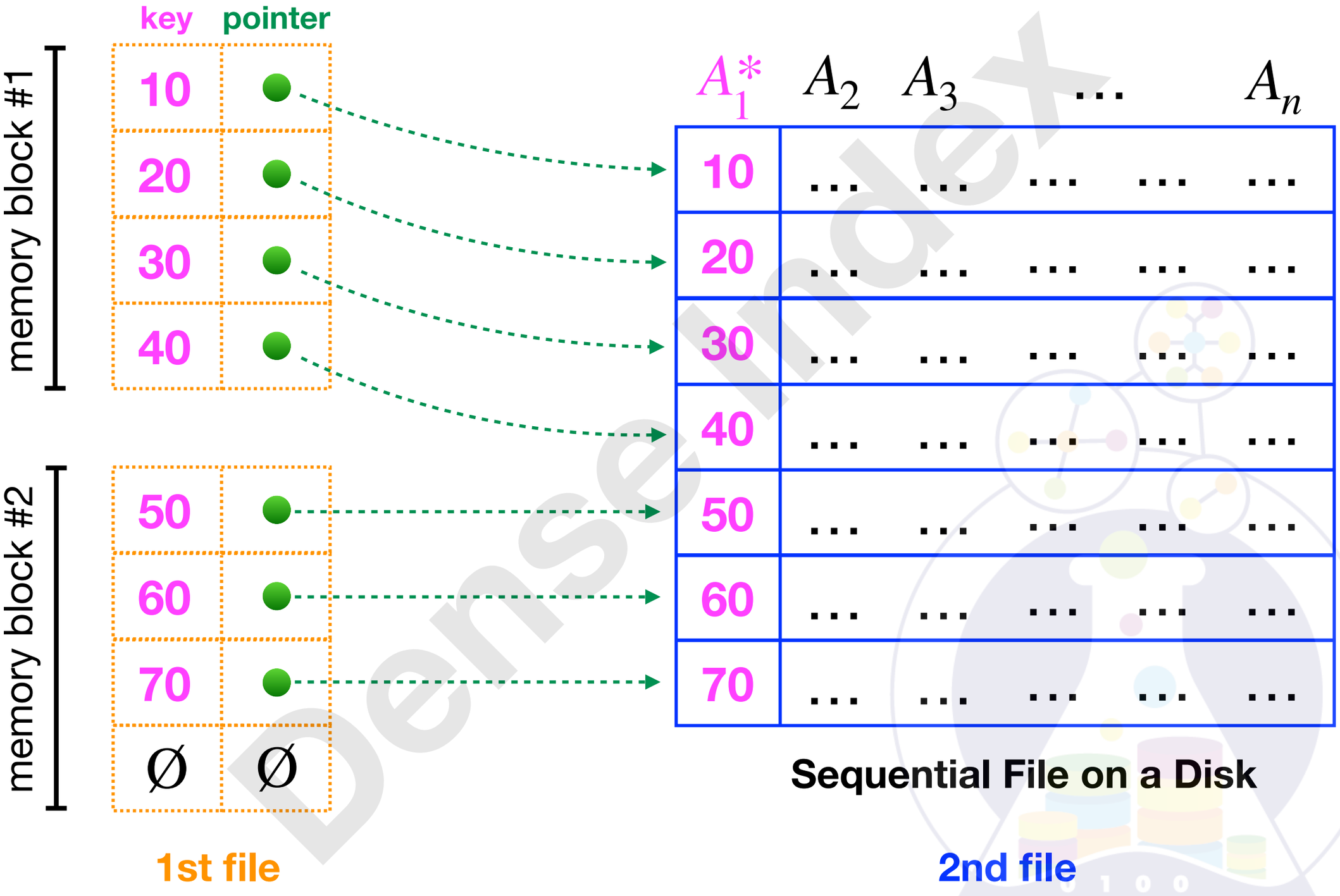


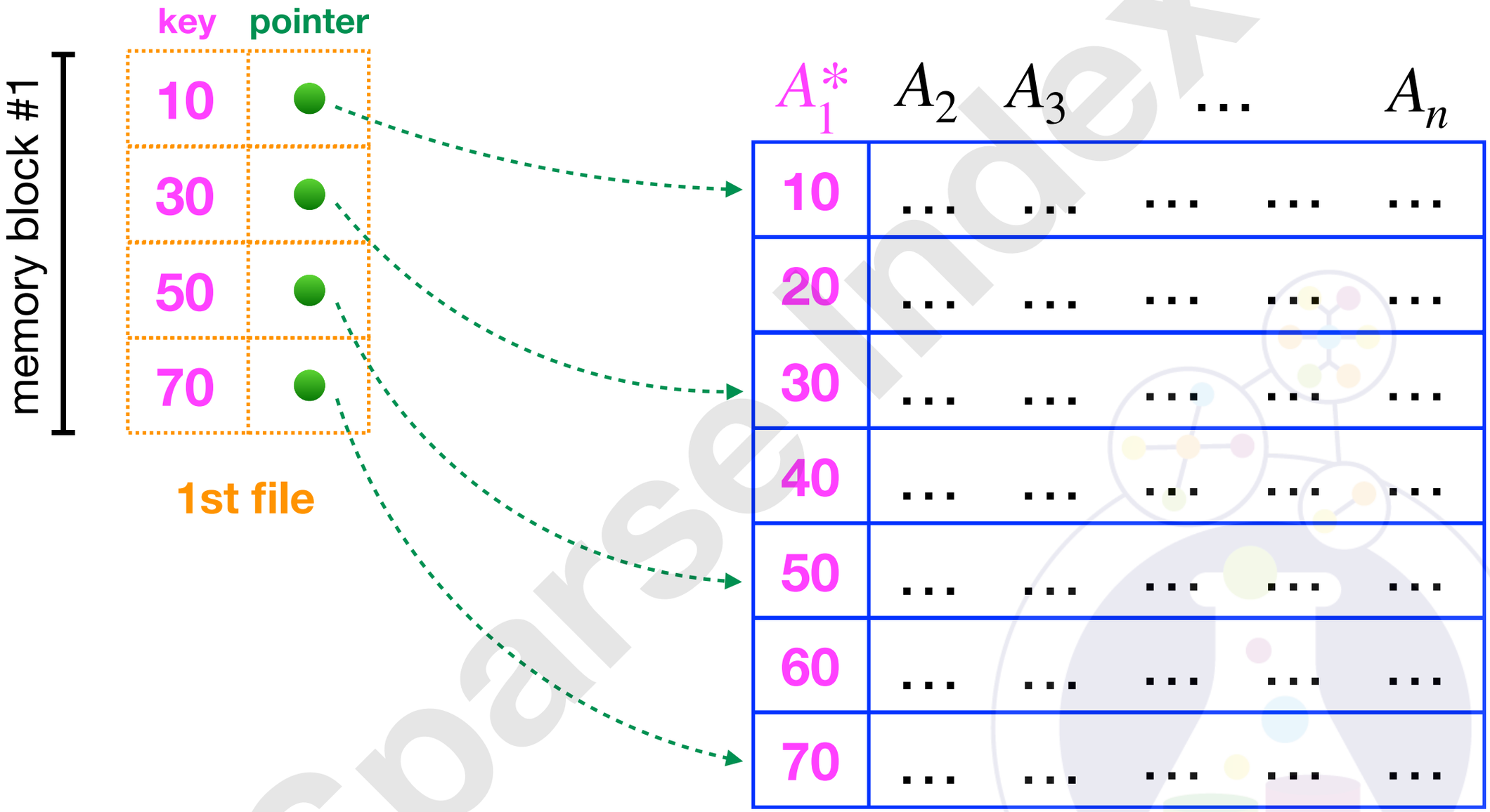
* for B-Tree

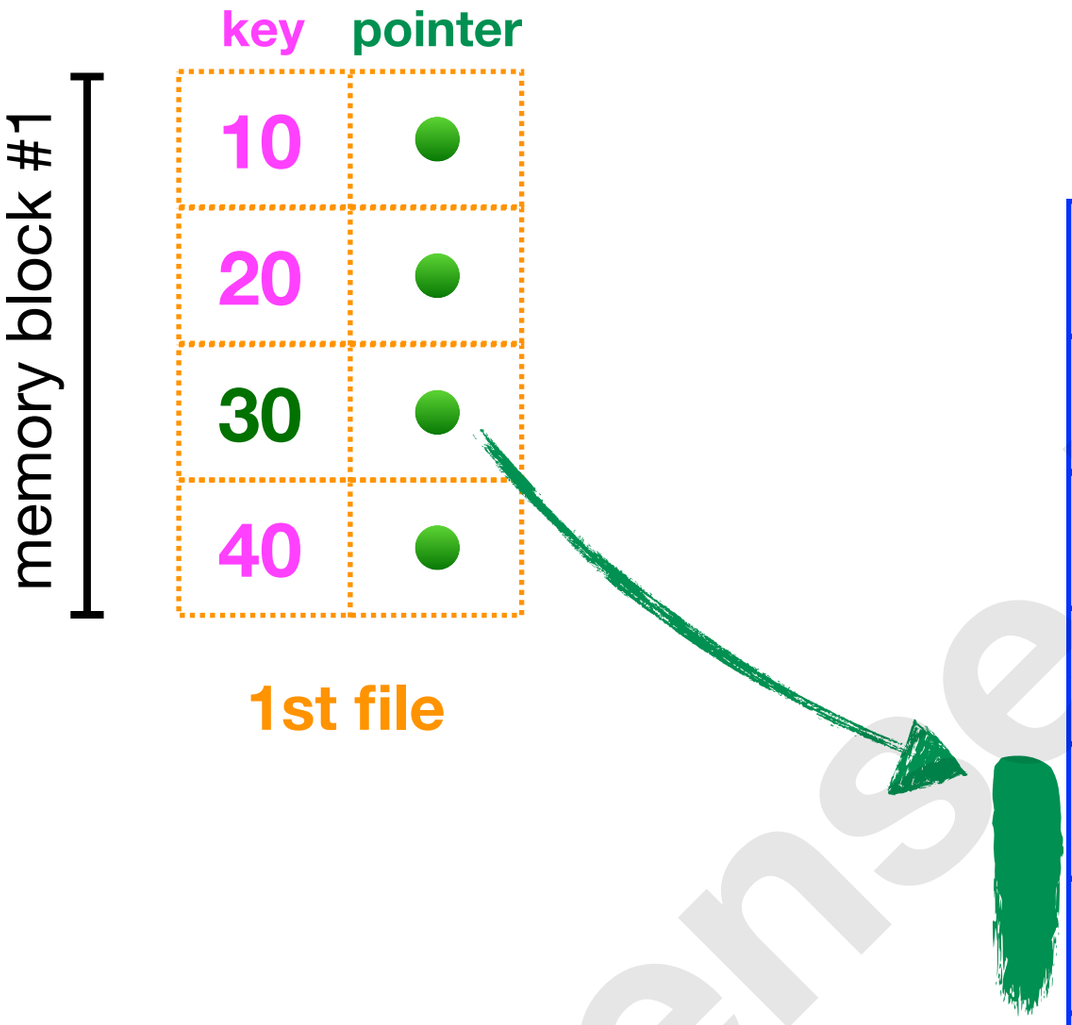
Index

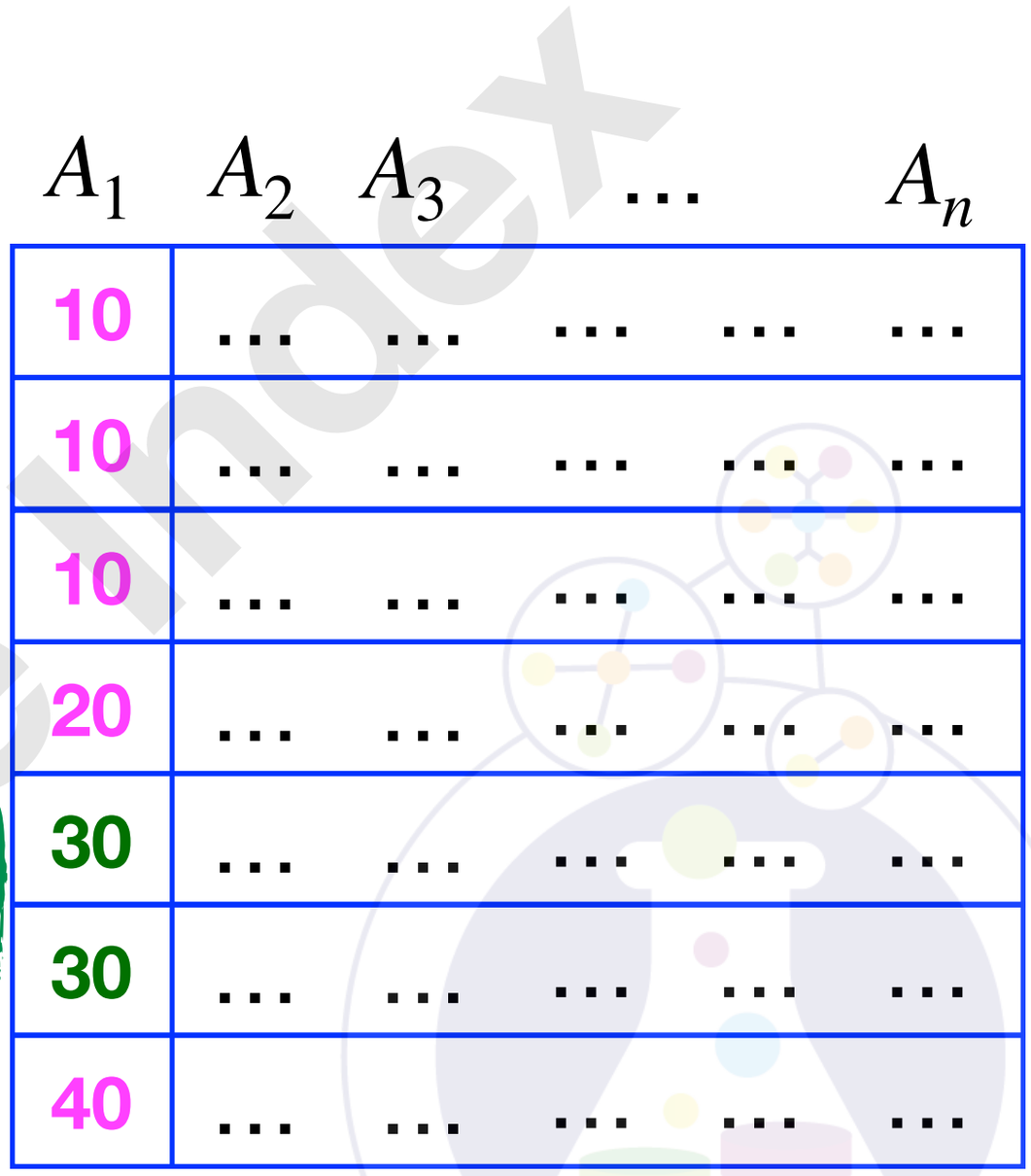
Searched values

Values address(es) ~ pages





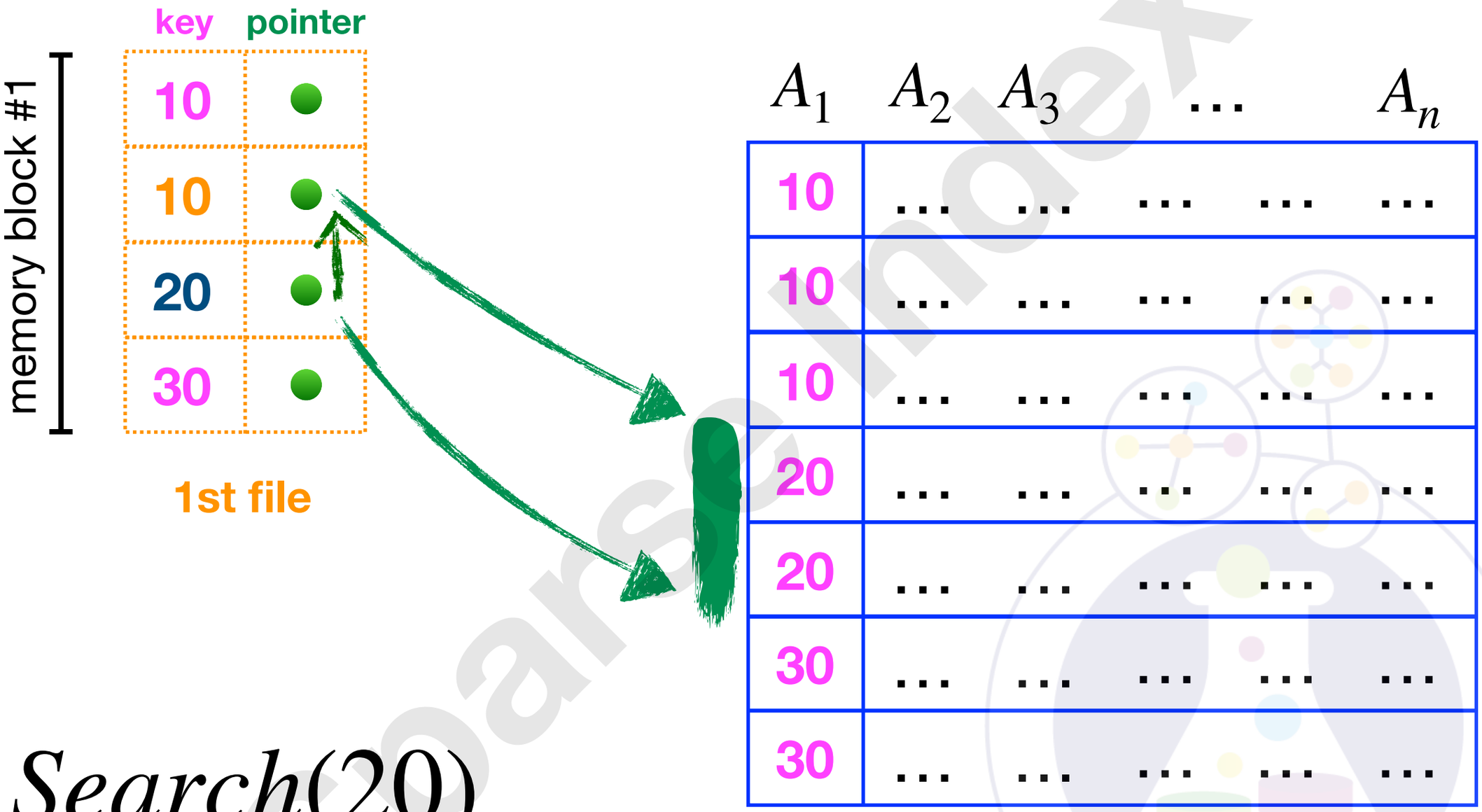




Sequential File on a Disk

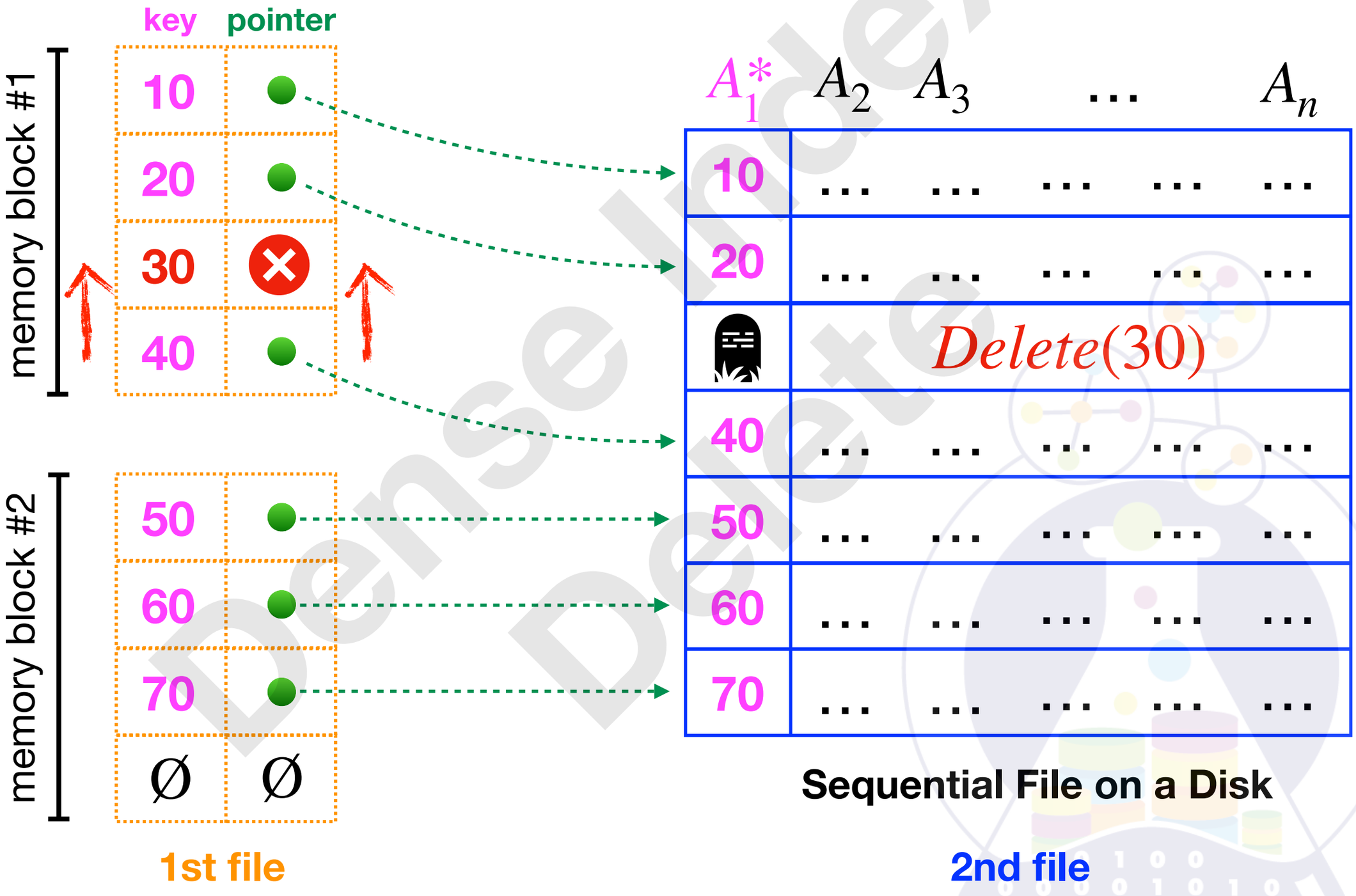
2nd file

Search(30)

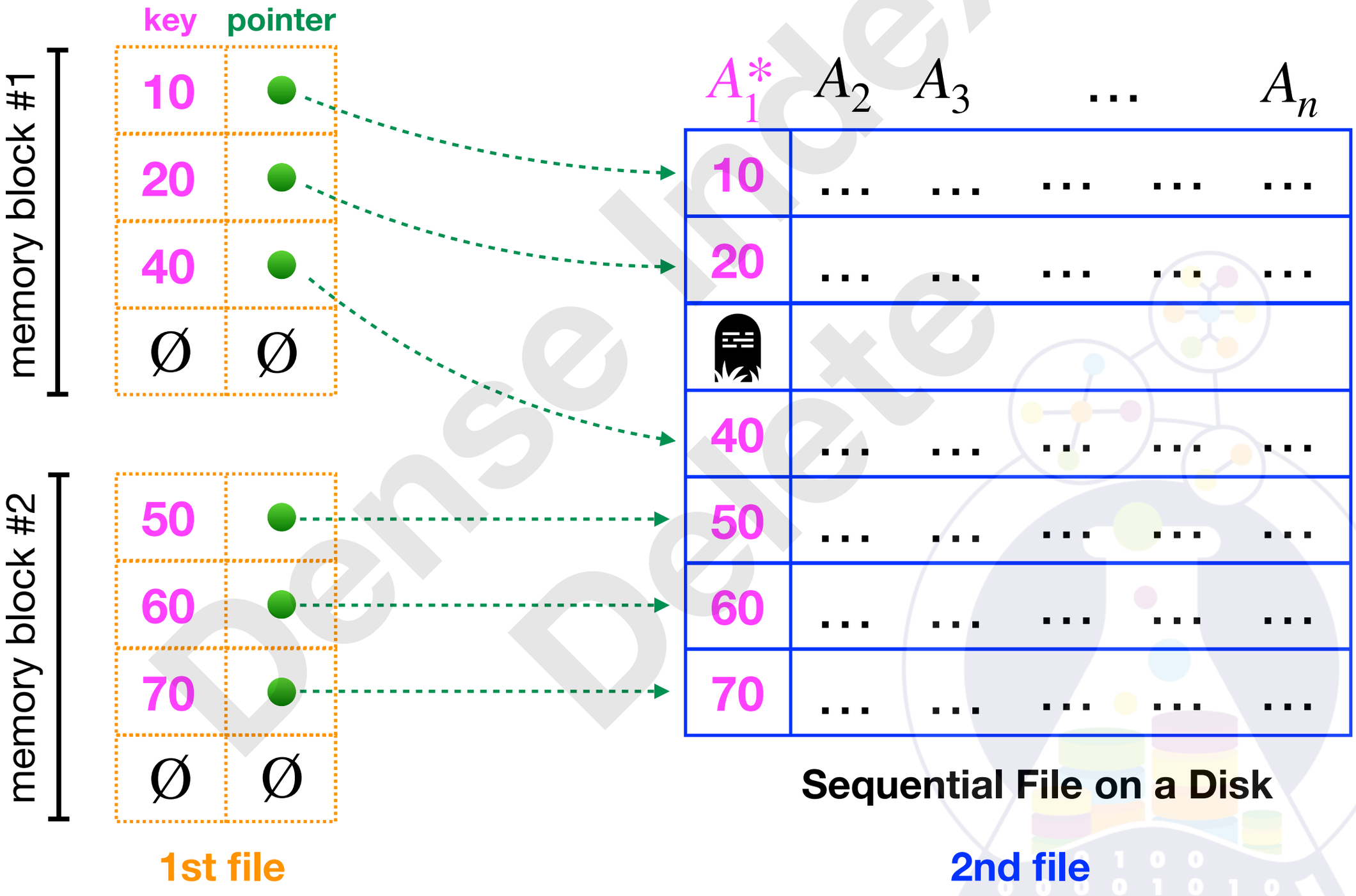


Search(20)

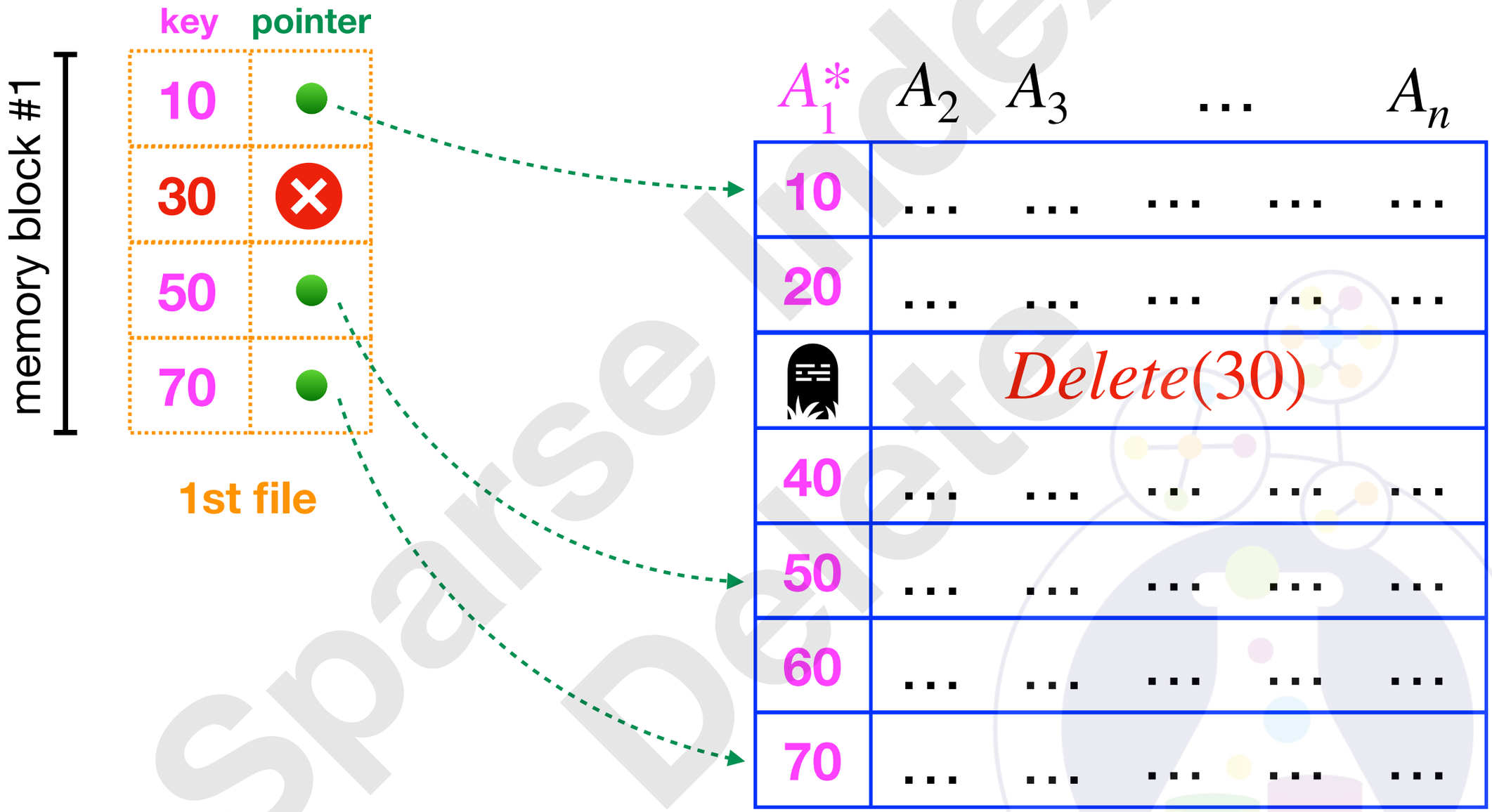
Phase #1



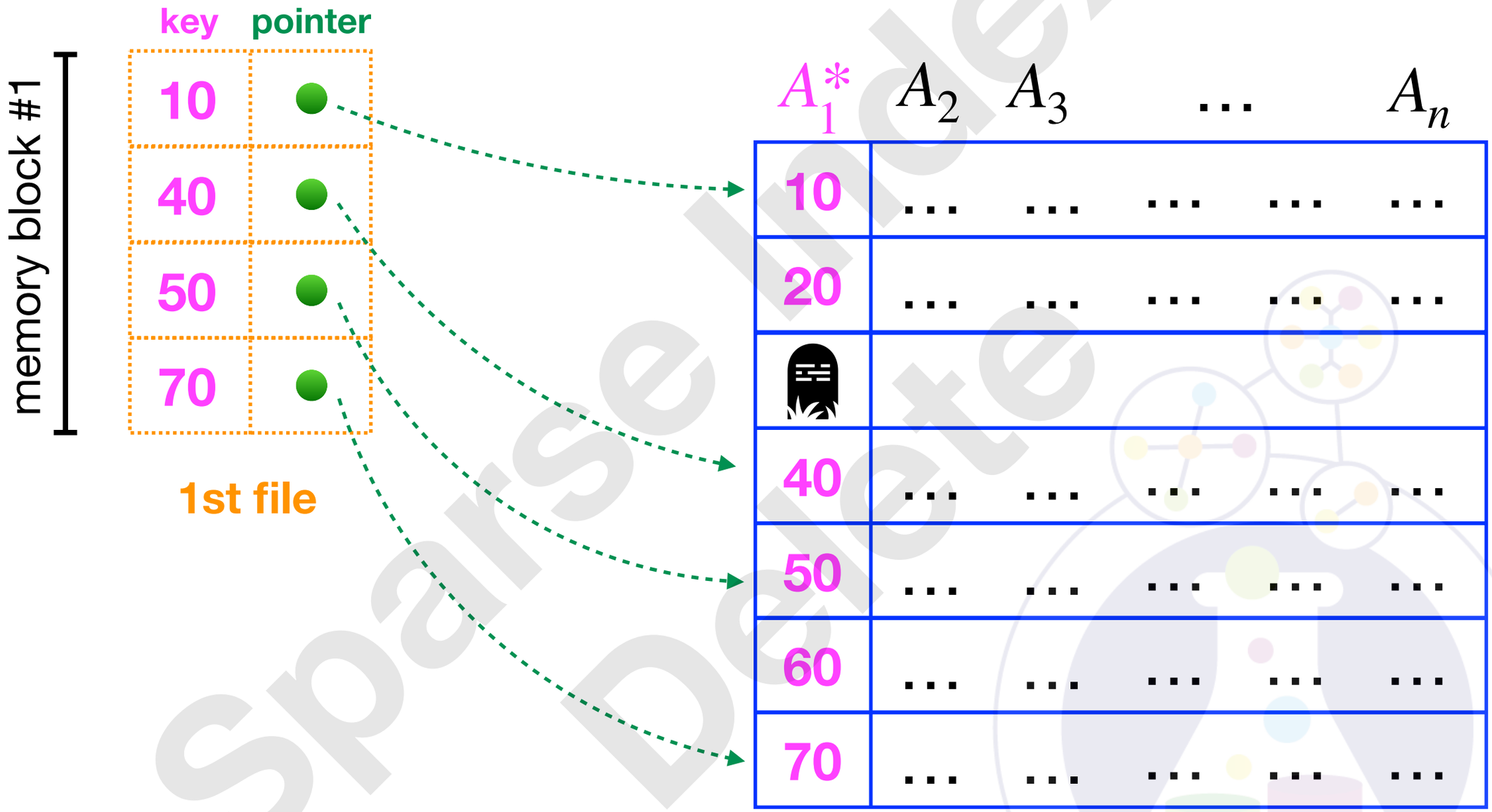
Phase #2



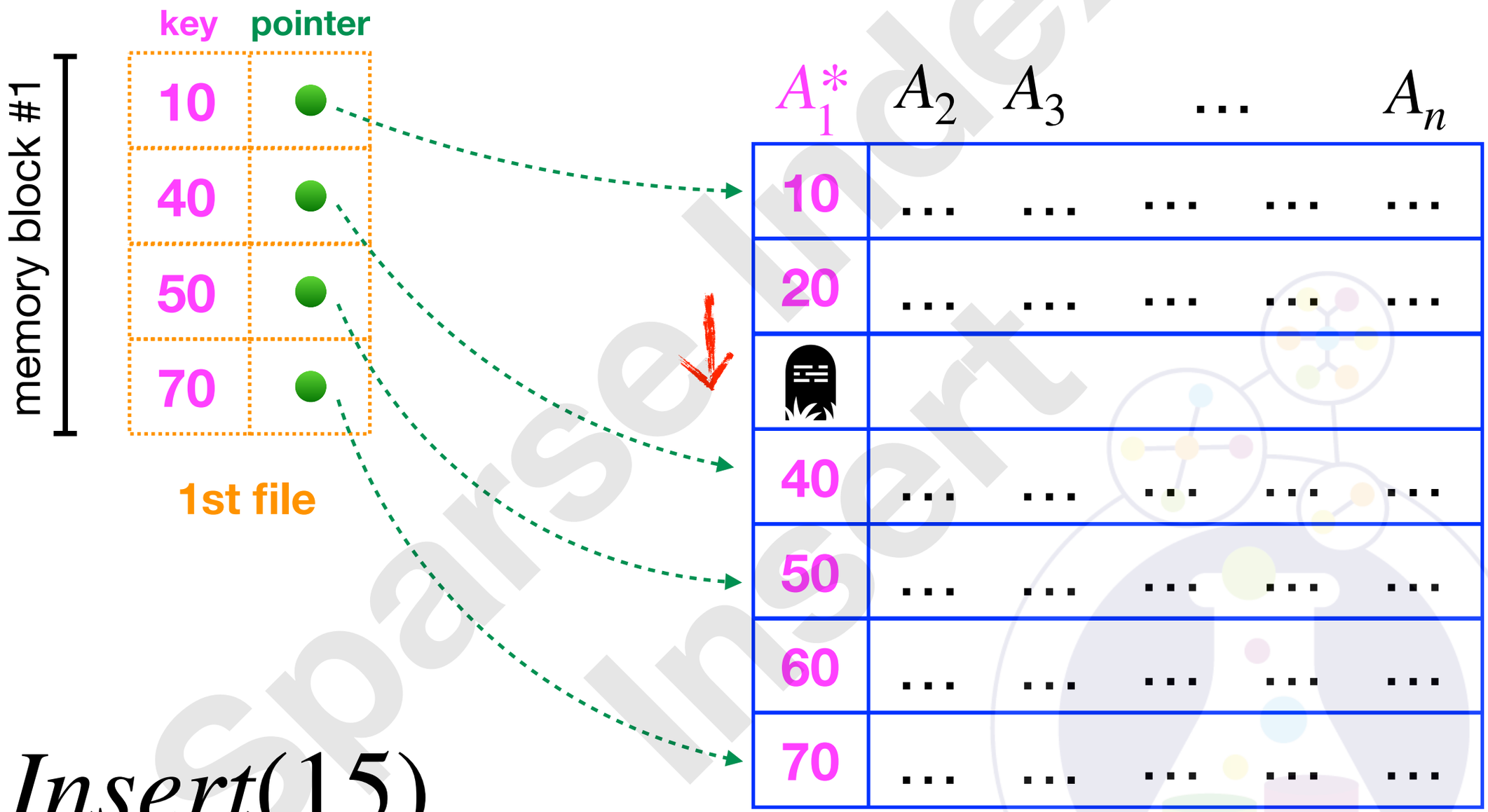
Phase #1



Phase #2

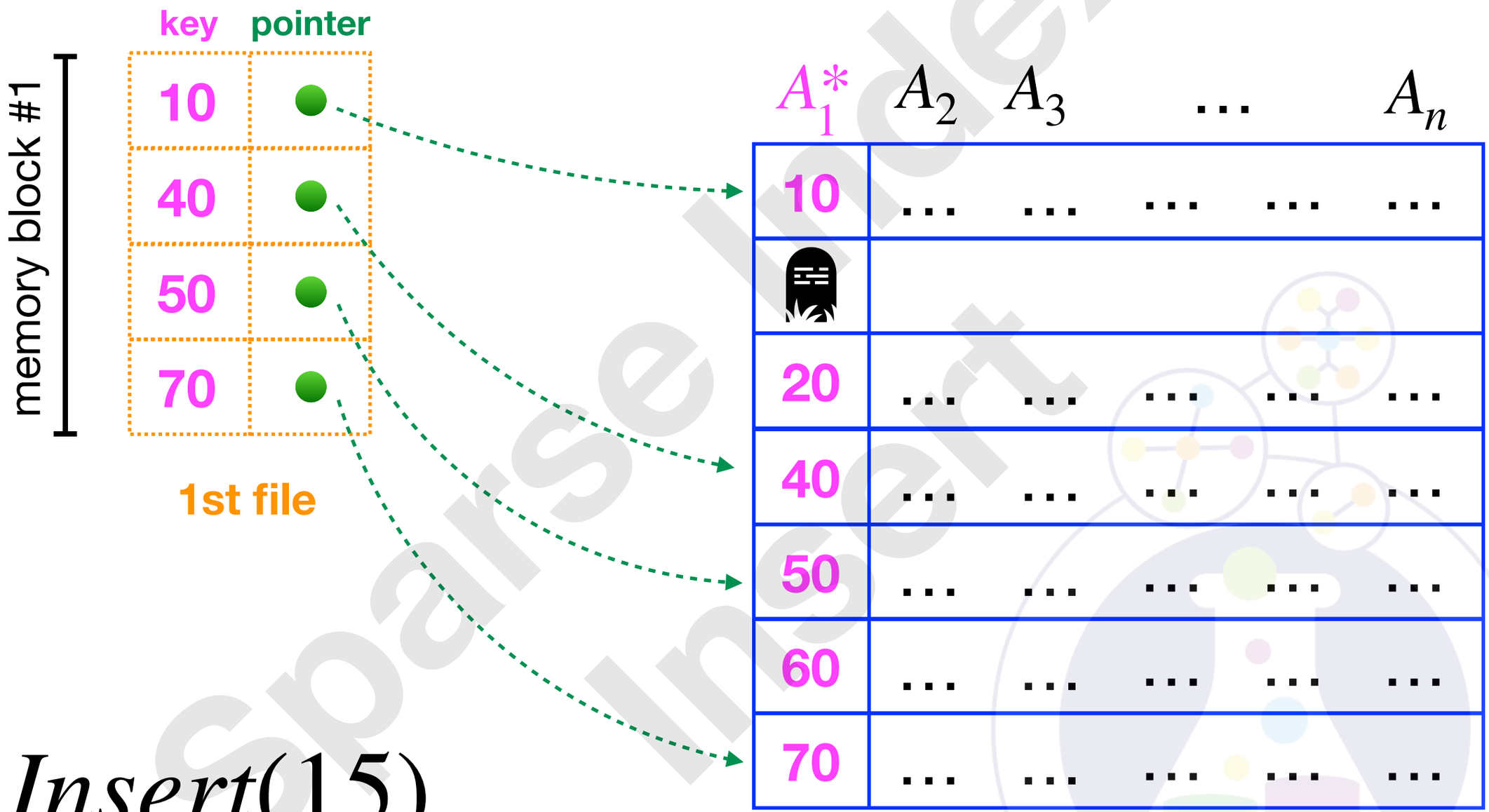


Phase #1



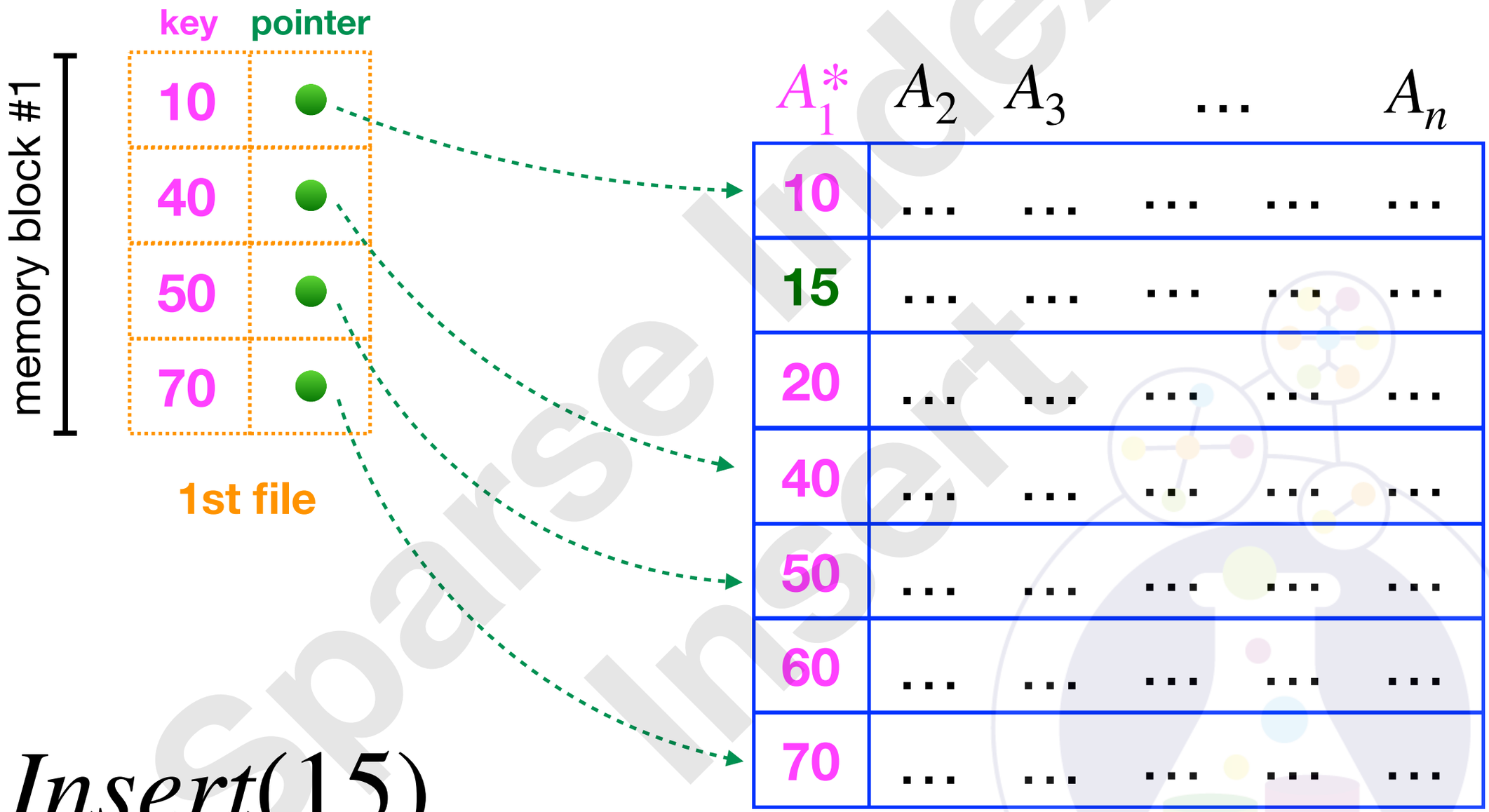
Insert(15)

Phase #1.1



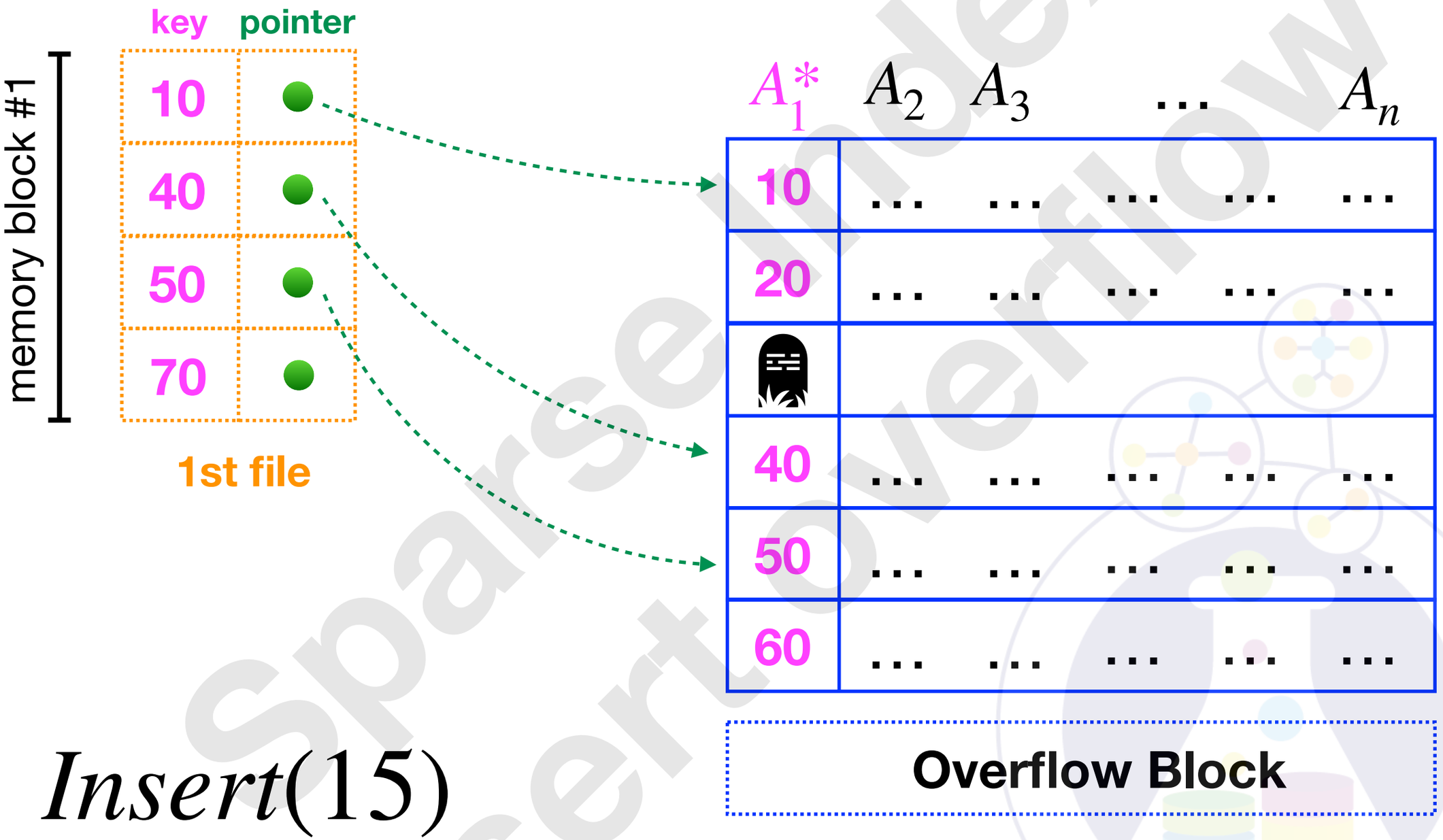
Insert(15)

Phase #2



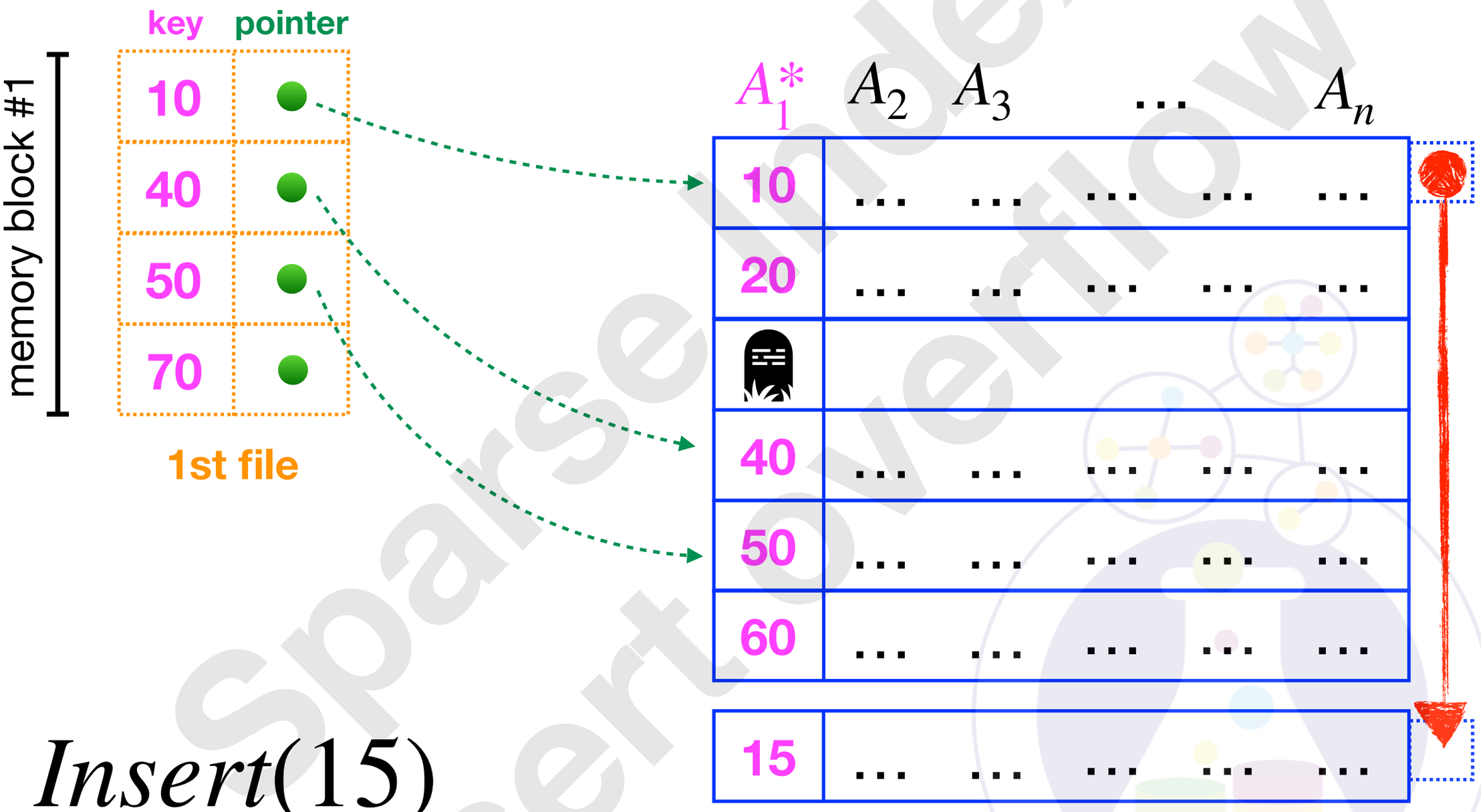
Insert(15)

Phase #1



Insert(15)

Phase #2



Insert(15)

```
SELECT *
  FROM Employee
 WHERE ID = 1
```

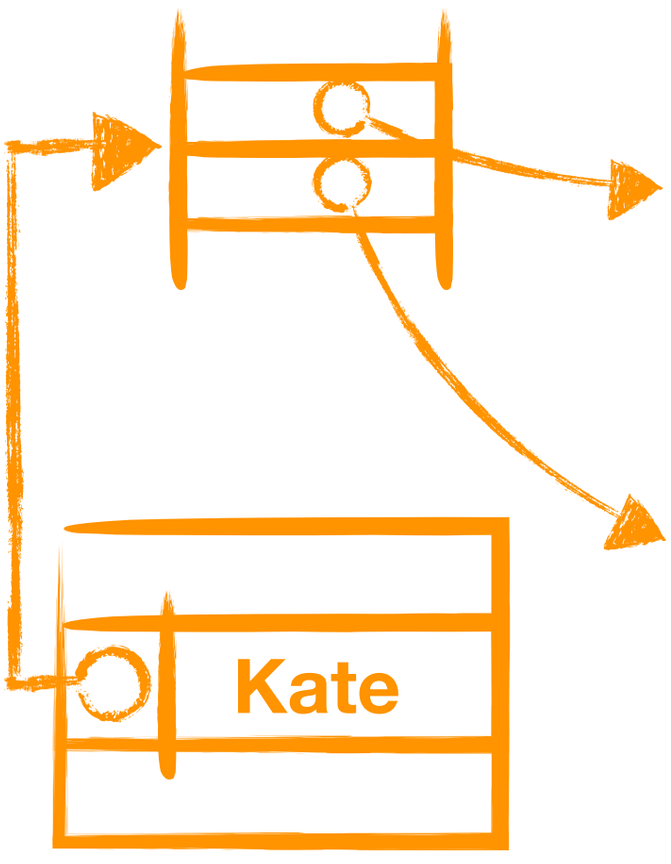
Primary Index (sparse / dense)

ID*	Name
1	Bulat
2	Kate
3	Damir
…	…
1000000	Ivan

Secondary Index (dense only)

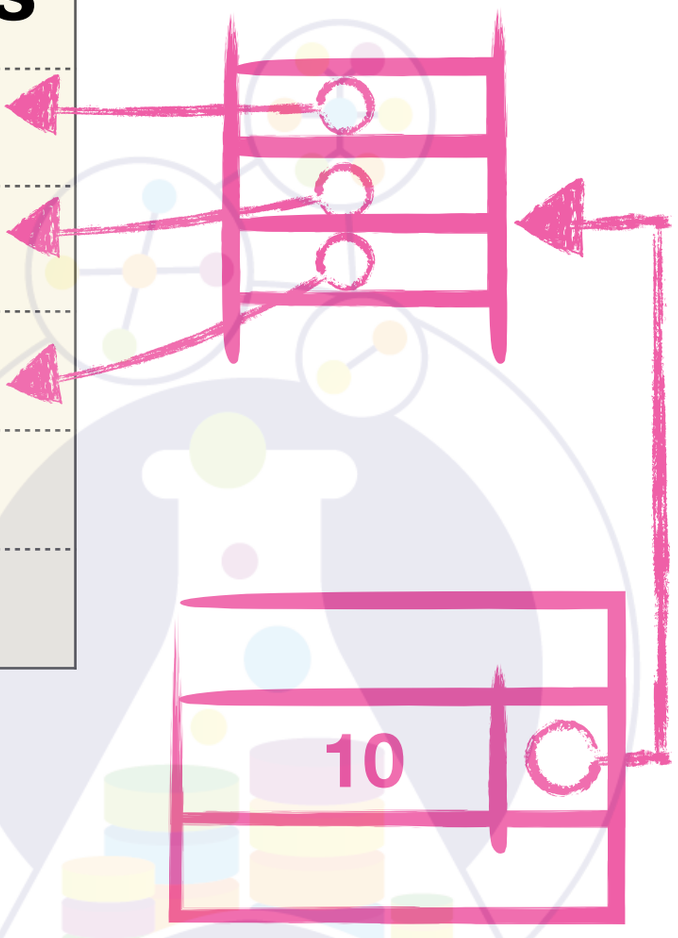
```
SELECT *
  FROM Employee
 WHERE Name = ‘Kate’
```

```
SELECT *
  FROM Employee
 WHERE Name = ‘Kate’ AND Status = 10
```

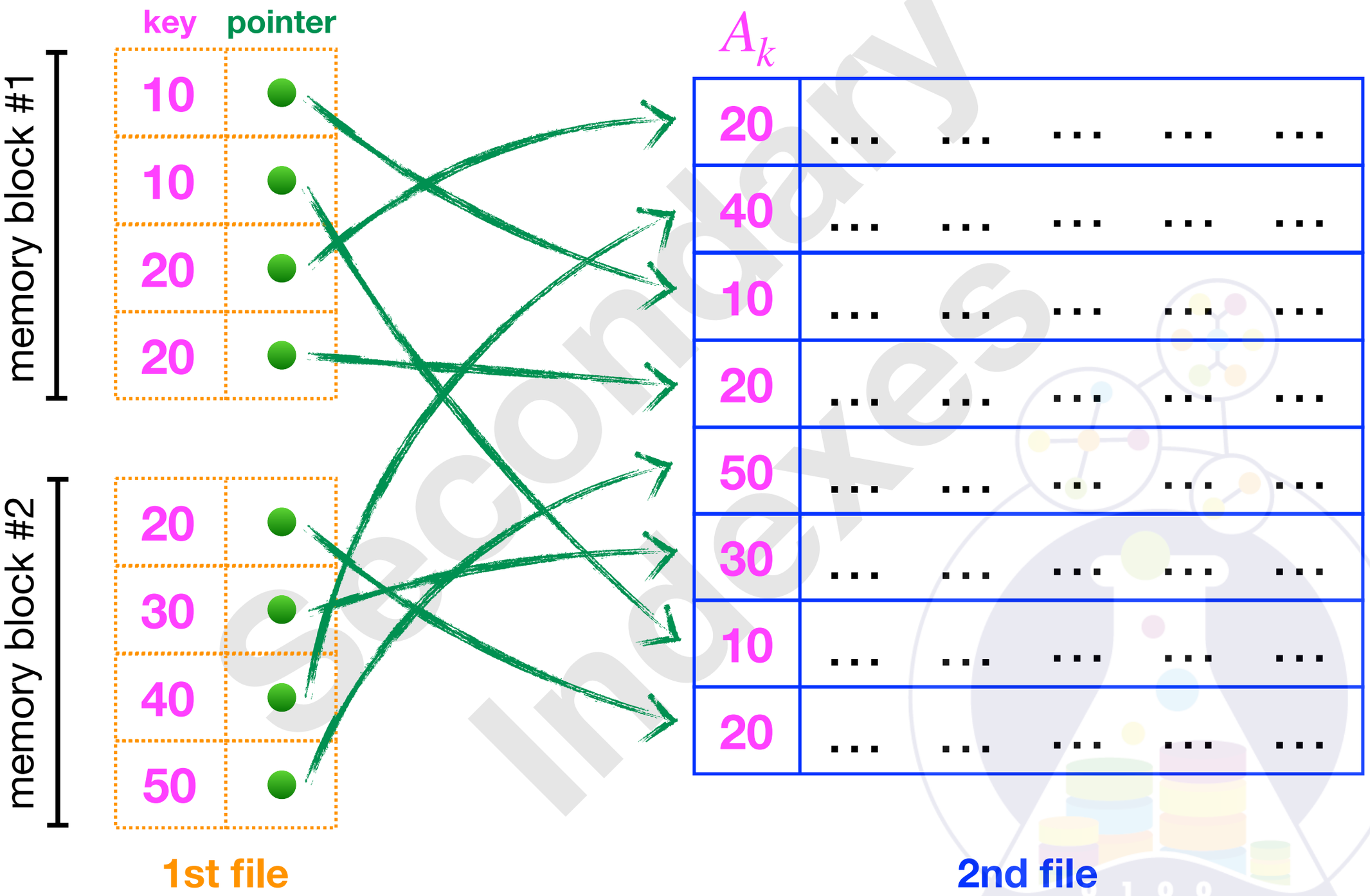


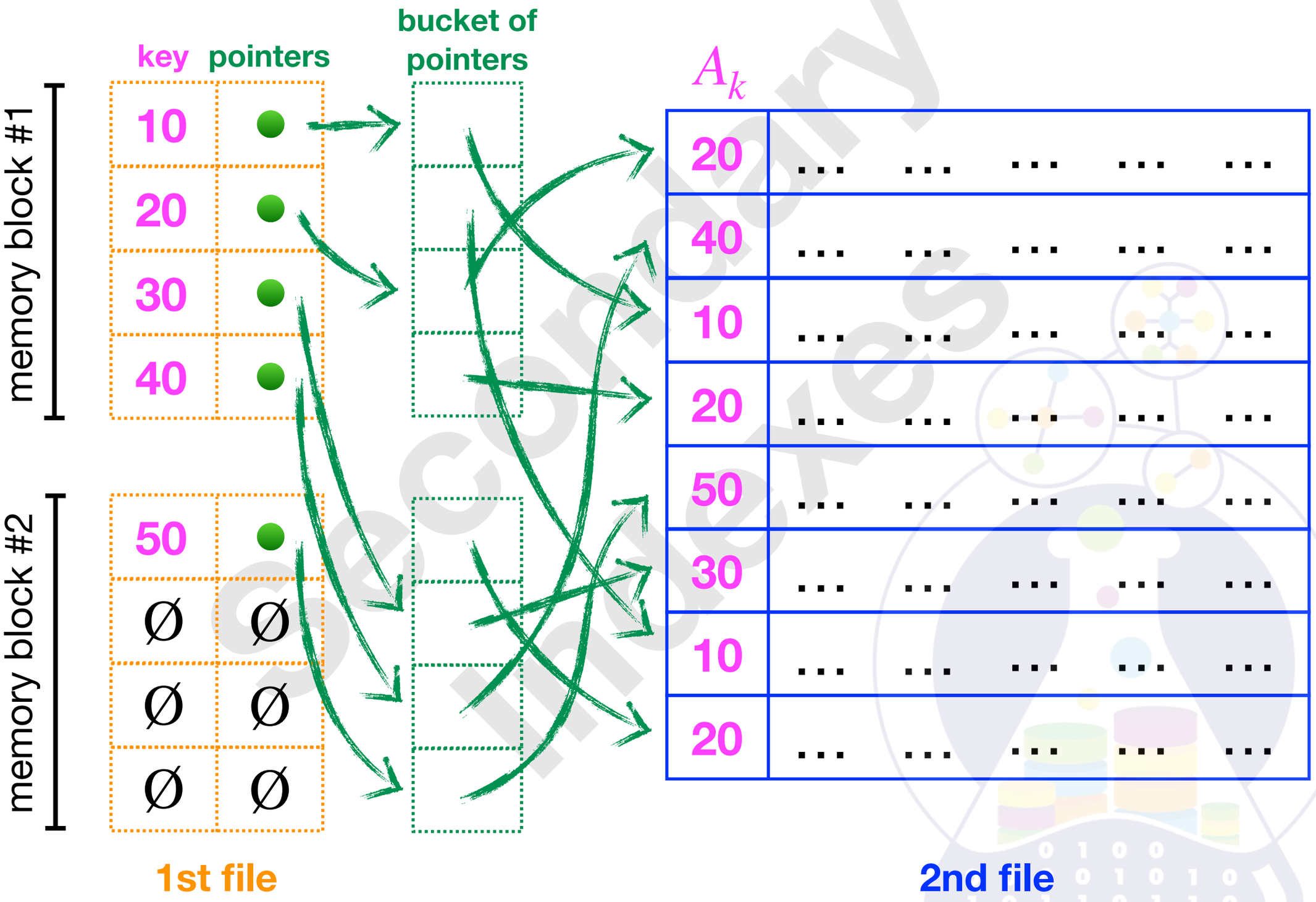
index for “Name”

ID*	Name	Status
1	Bulat	10
2	**Kate**	**10**
3	Damir	10
...	...	...
1000000	Kate	20



index for “Status”

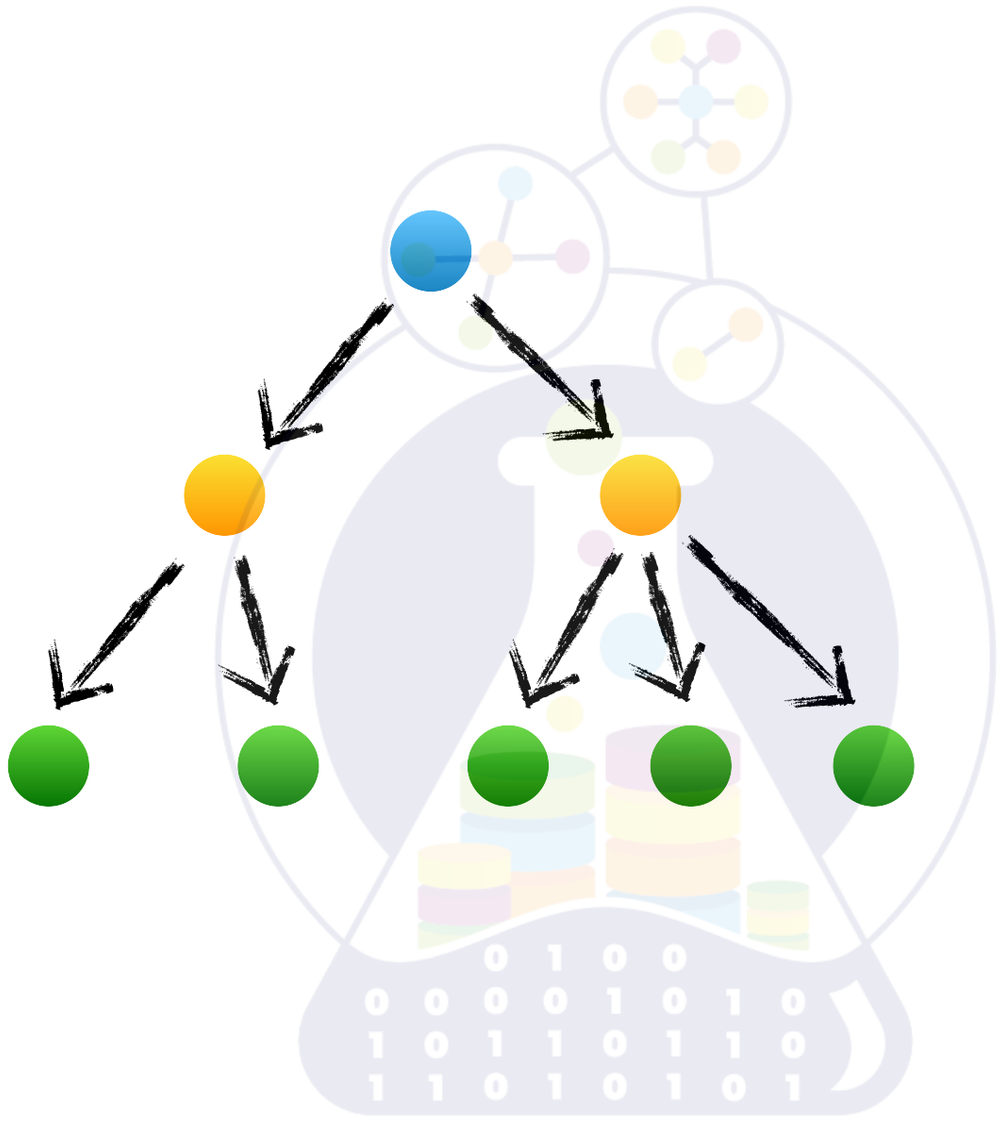




Balanced Tree - all paths from **root** to **leaves** are equal

Typical Balanced Tree has 3 levels

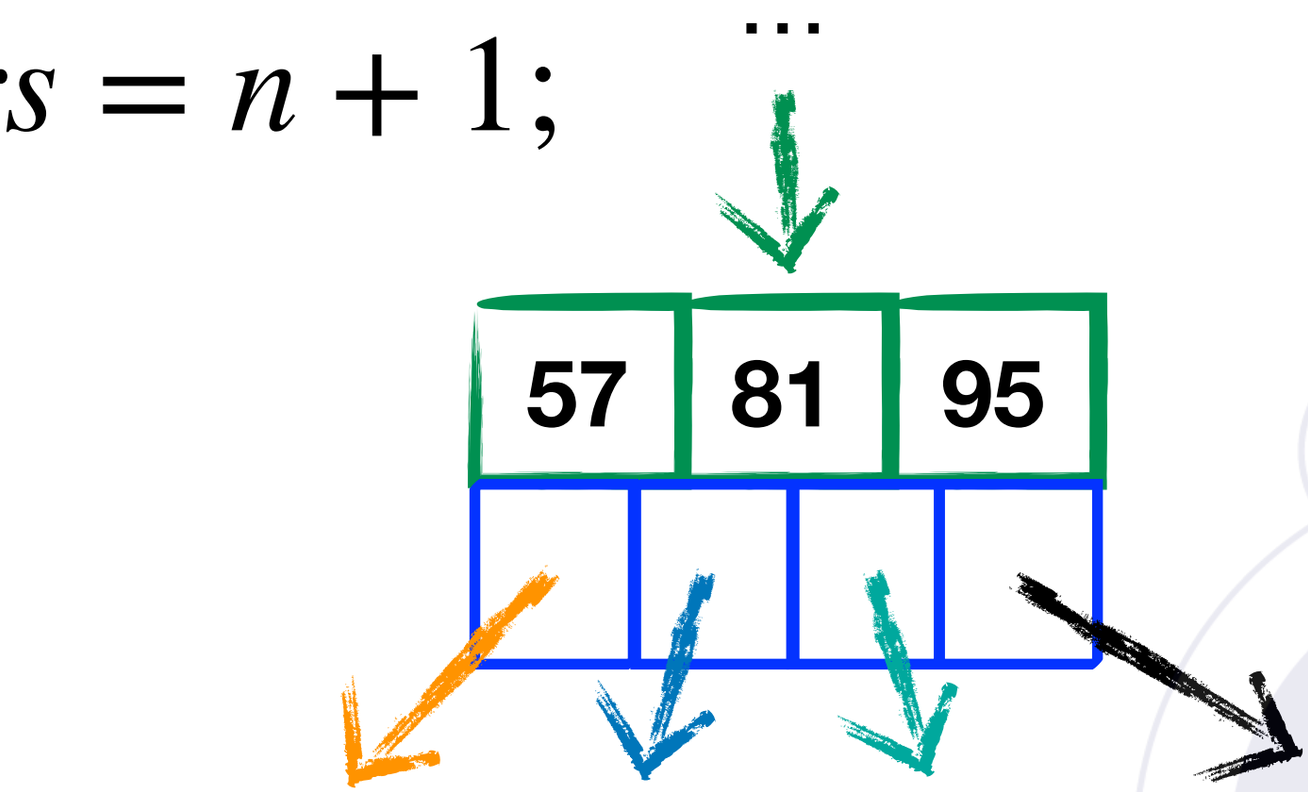
- Root
- Middle nodes
- Leaves



- Middle nodes

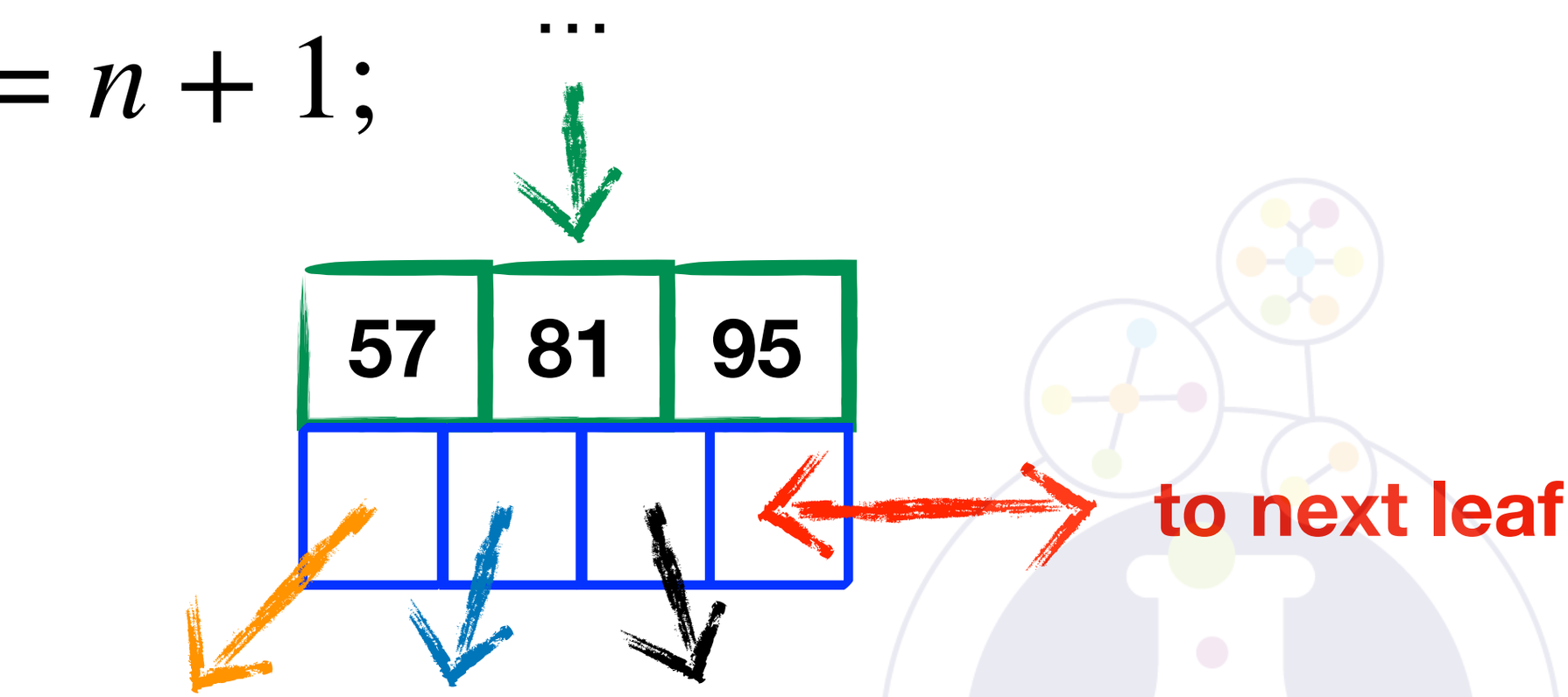
$n = 3;$

$pointers = n + 1;$

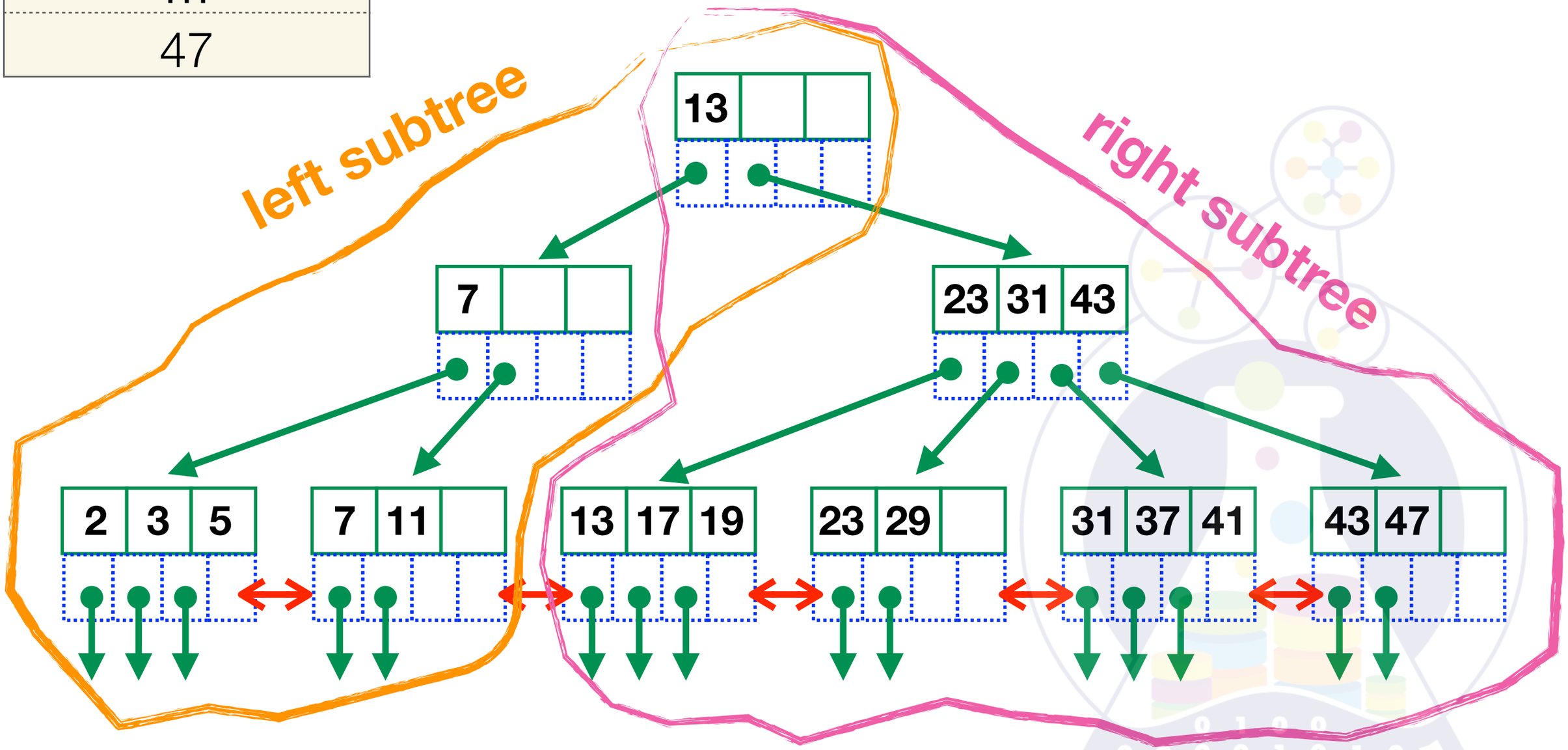


$n = 3;$

$pointers = n + 1;$



PrimeNumber*
3
1
5
...
47



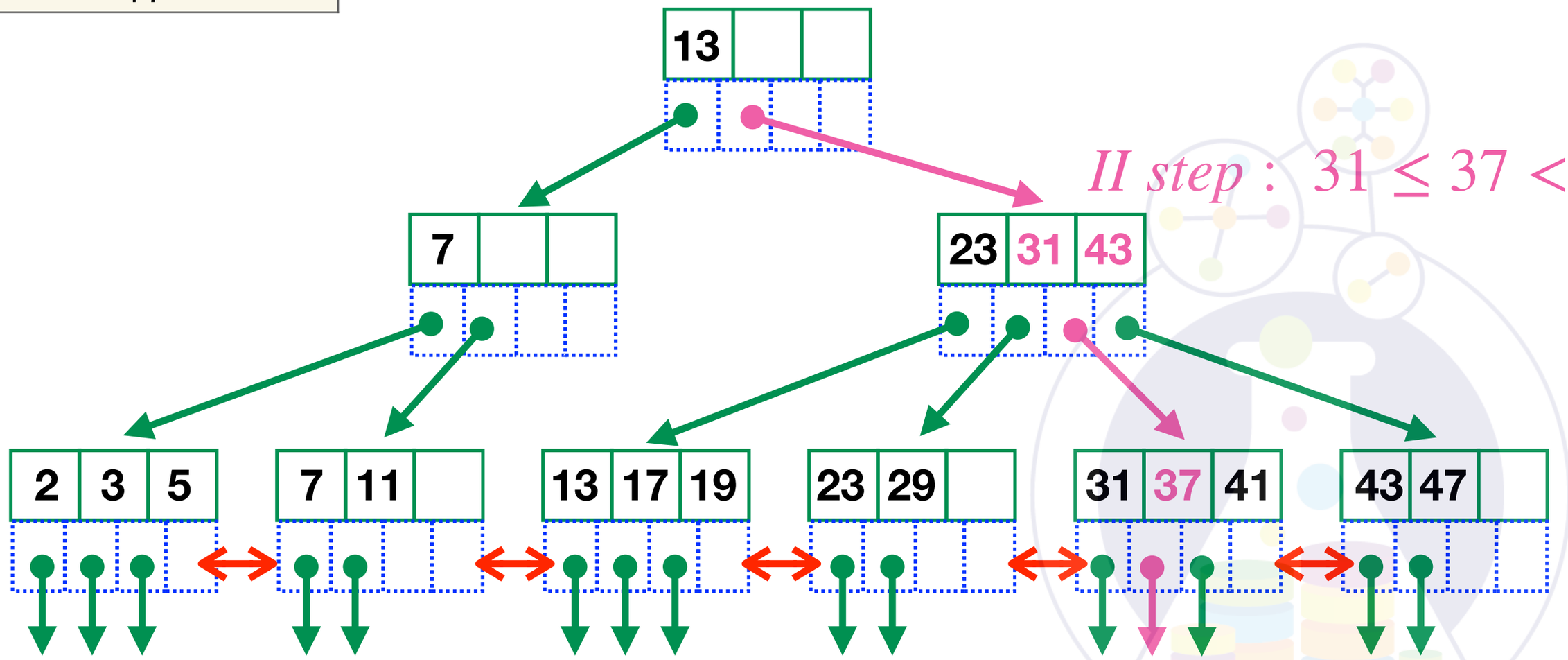
PrimeNumber*
3
1
5
…
47

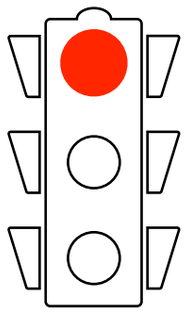
Search(37)

I step : 13 ≤ 37

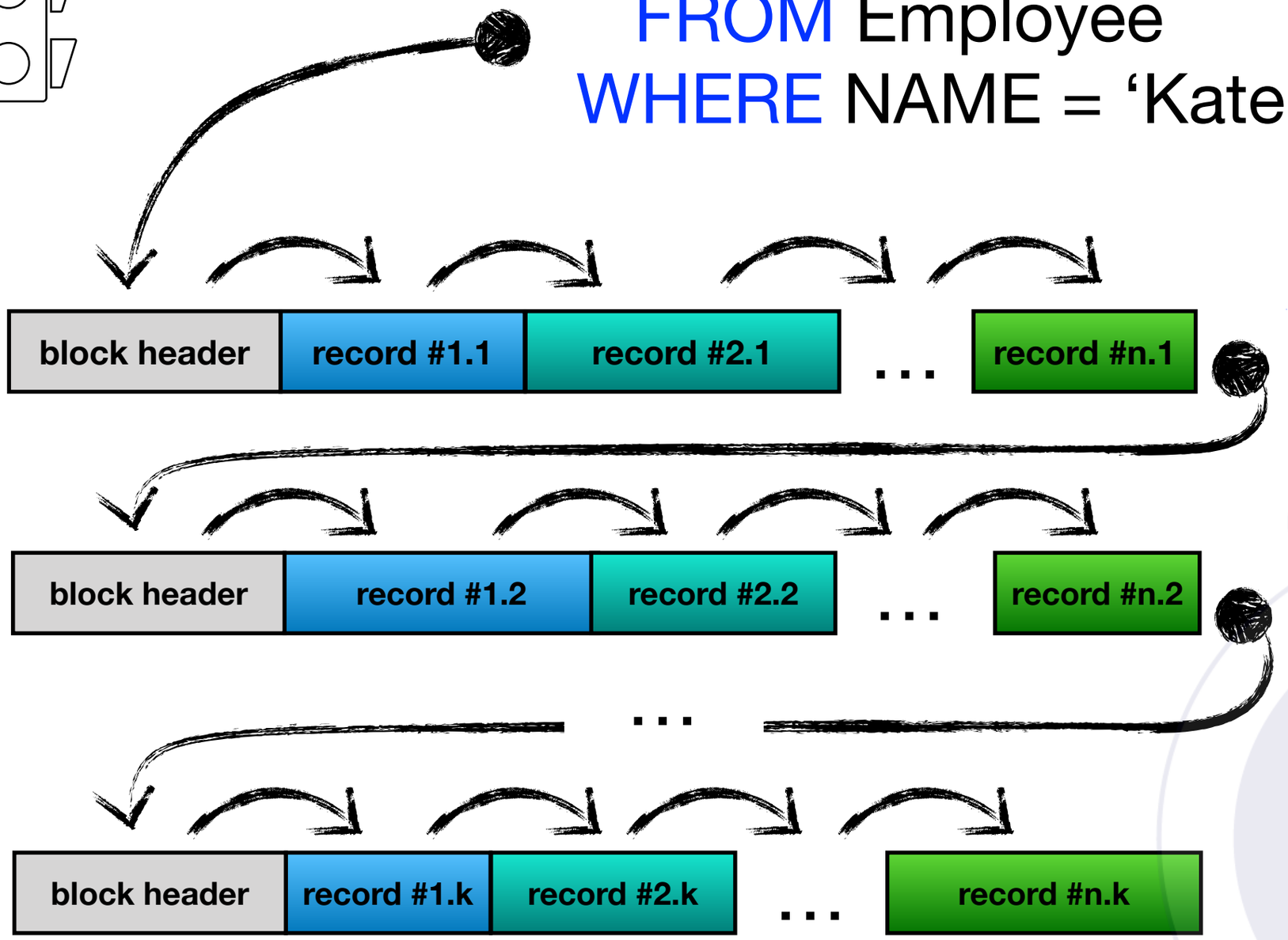
II step : 31 ≤ 37 < 43

III step



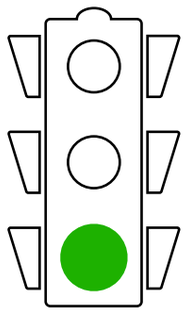


```
SELECT ID, NAME
  FROM Employee
 WHERE NAME = 'Kate'
```

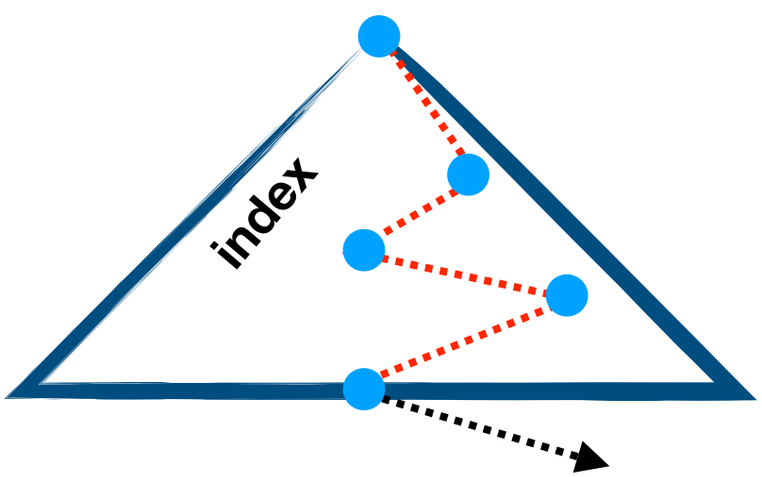


ID*	Name
1	Ivan
2	Anna
...	...
1000000	Kate

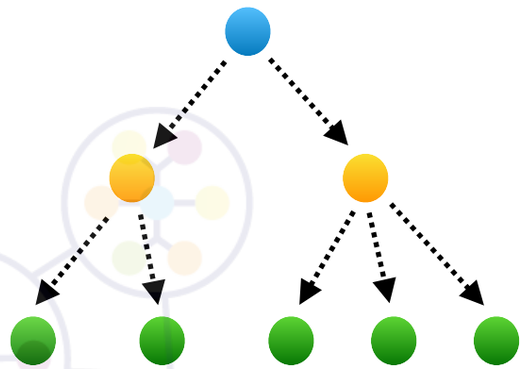
SEQUENTIAL SCANNING



```
SELECT ID, NAME
  FROM Employee
 WHERE NAME = 'Kate'
```

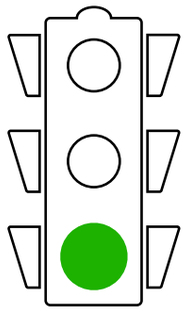


Key = ‘Kate’, TID = (block=7, offset=2)

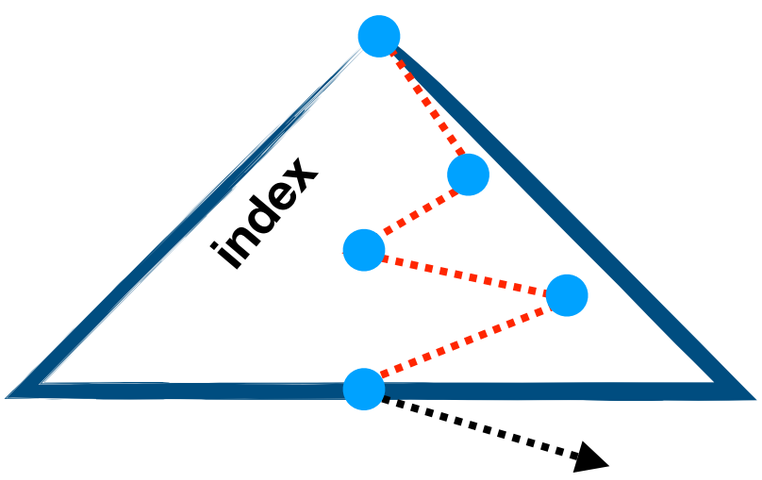


ID*	Name
1	Ivan
2	Anna
...	...
1000000	Kate

INDEX SCANNING

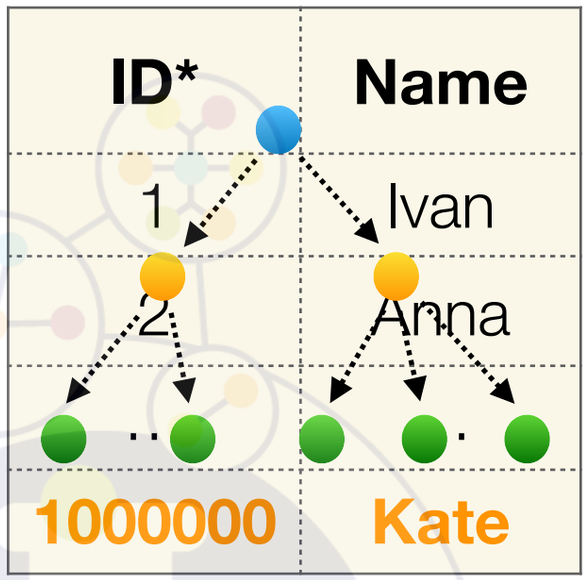


```
SELECT ID, NAME
  FROM Employee
 WHERE NAME = 'Kate'
```

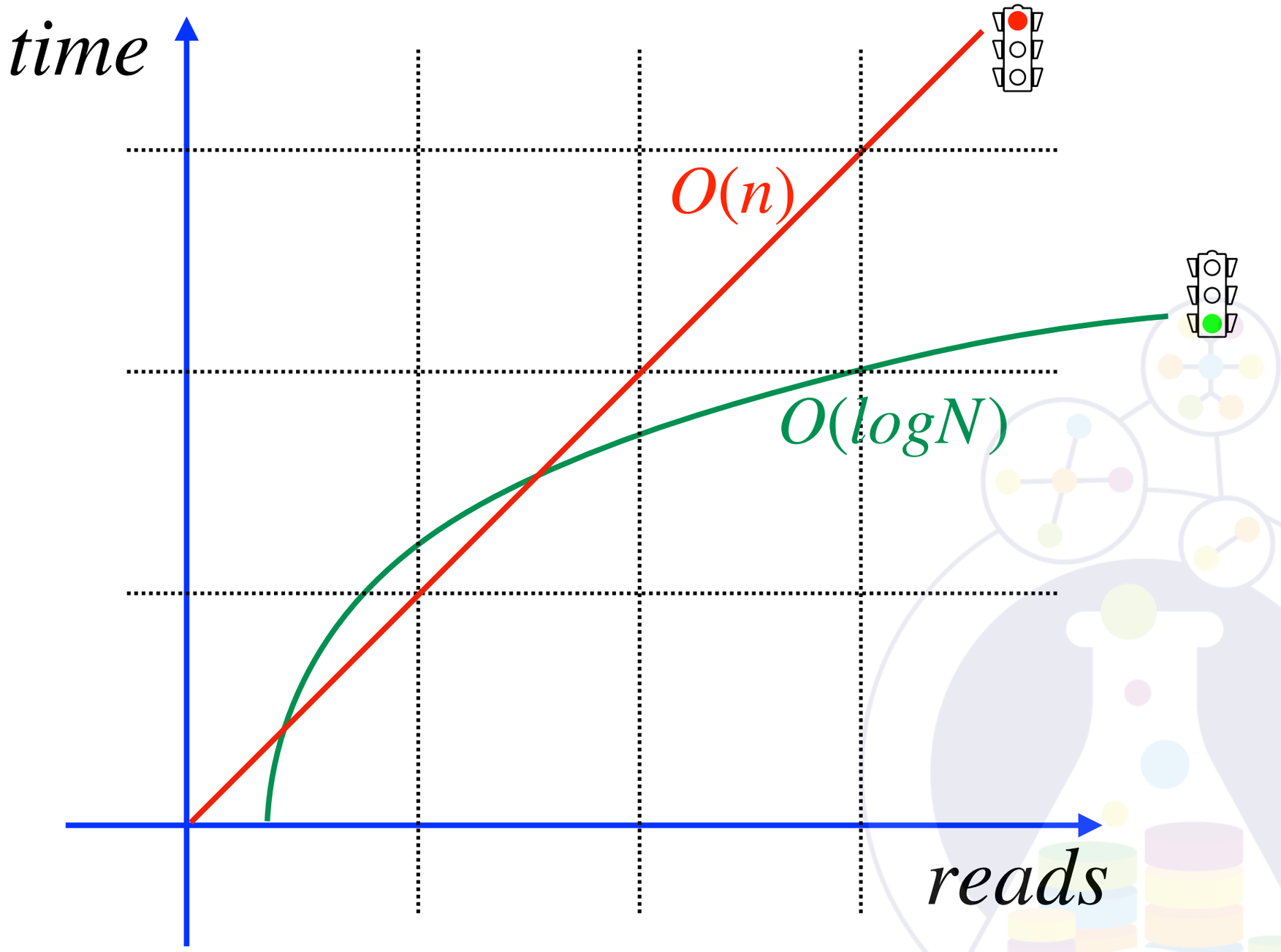


Key = ‘Kate’, TID = (**block=7, offset=2**)

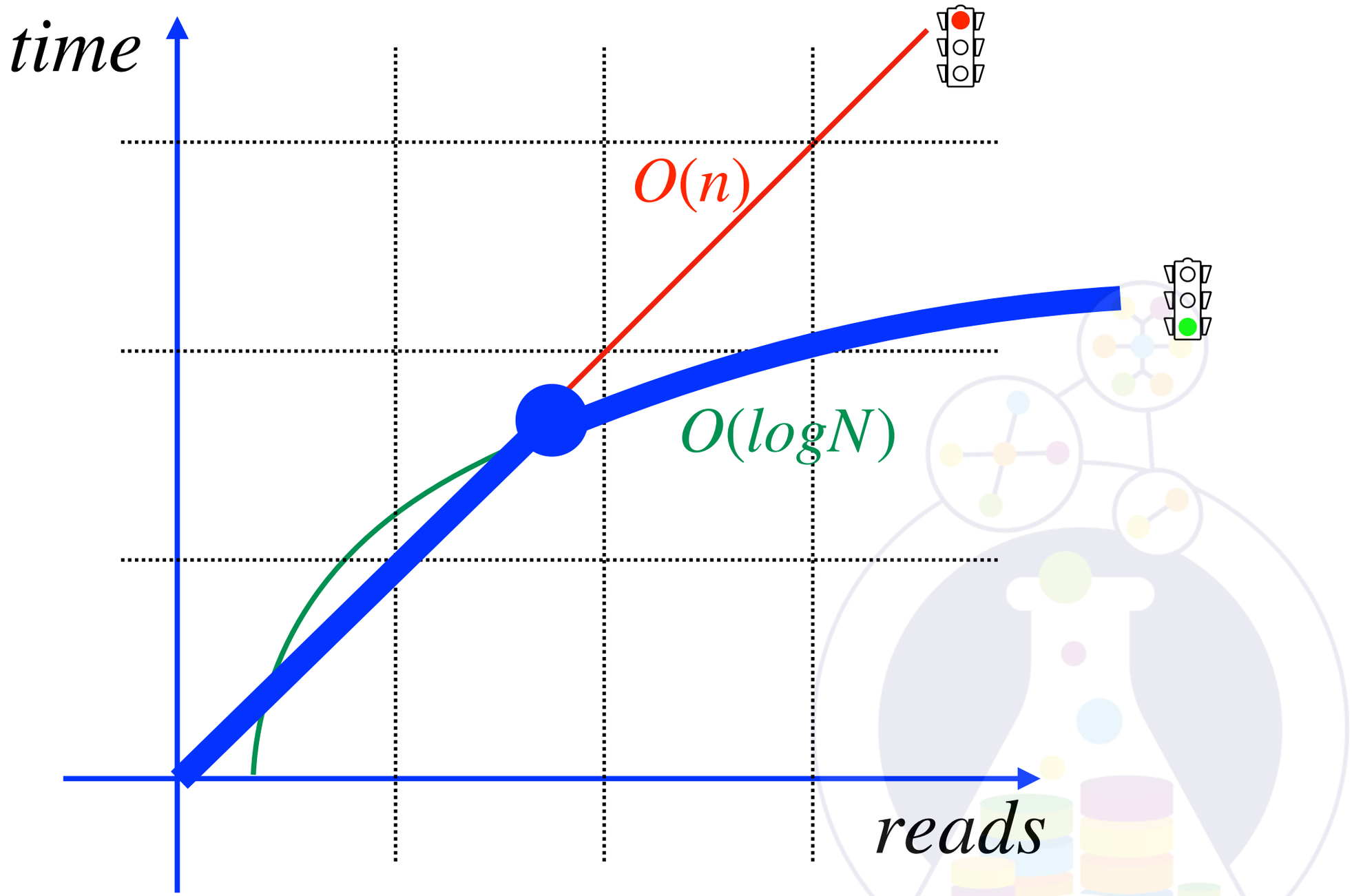
\+ ID = 1000000



INDEX ONLY SCANNING

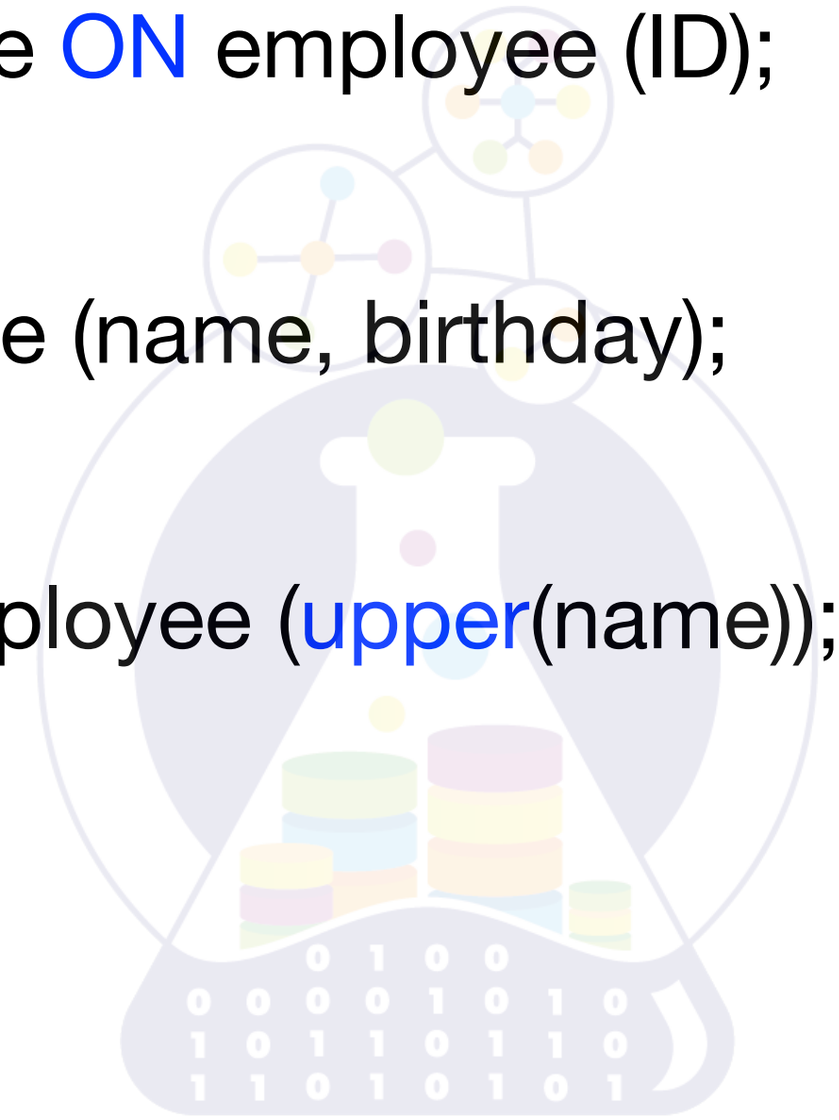


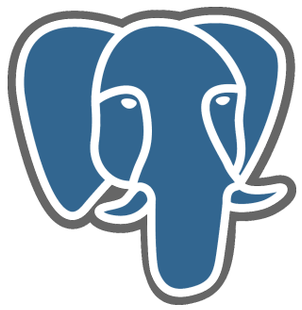
Why is the index not being used?



ORACLE

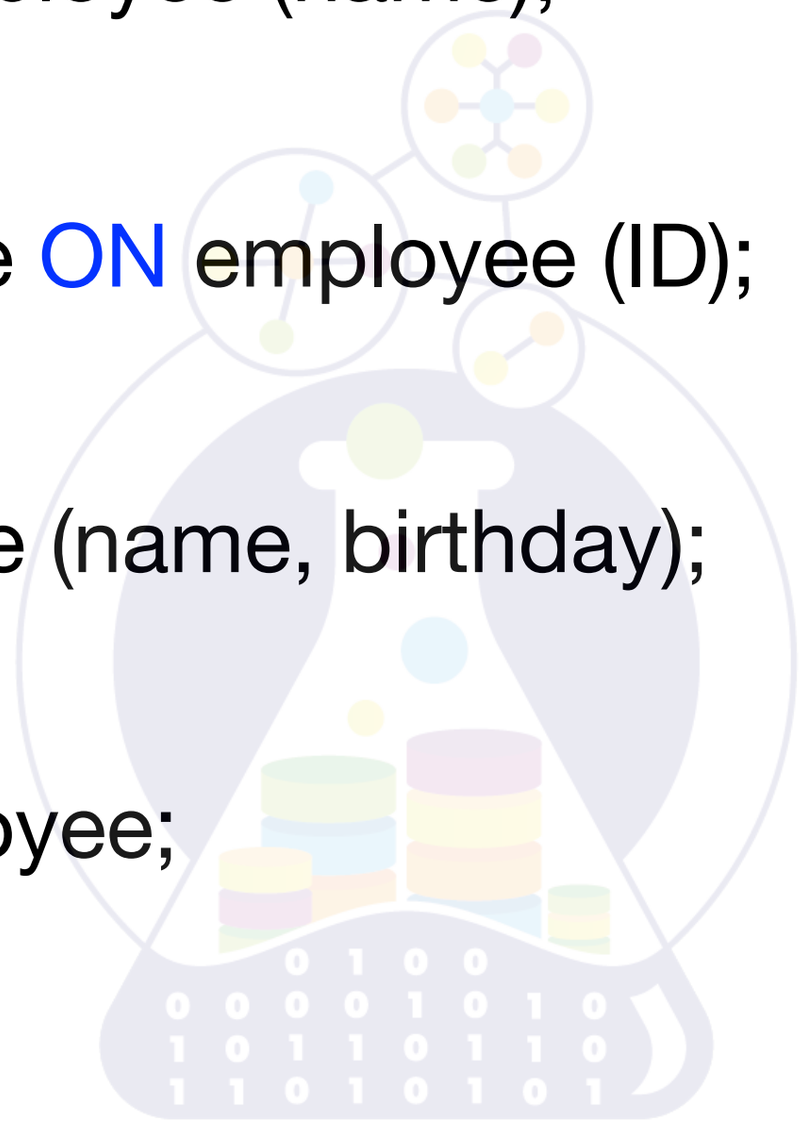
```
CREATE INDEX idx_employee ON employee (name);

CREATE UNIQUE INDEX uk_employee ON employee (ID);

CREATE INDEX idx_emp ON employee (name, birthday);

CREATE INDEX idx_employee ON employee (upper(name));

DROP INDEX idx_employee;
```





```
CREATE INDEX idx_employee ON employee (name);

CREATE UNIQUE INDEX uk_employee ON employee (ID);

CREATE INDEX idx_emp ON employee (name, birthday);

CREATE INDEX idx_employee ON employee (upper(name))
WHERE name != ‘Ivan’;

DROP INDEX idx_employee;
```

```
CREATE INDEX idx_employee ON employee (name);

CREATE UNIQUE INDEX uk_employee ON employee (ID);

CREATE INDEX idx_emp ON employee (name, birthday);

DROP INDEX idx_employee ON employee;
```



```
SELECT *
   FROM Employee e, Tasks t
WHERE e.ID = t.EMPLOYEE_ID
```

```
SELECT ID
   FROM Employee
 WHERE Name LIKE ‘Ivanov%’
```

```
SELECT ID, NAME
   FROM Employee
ORDER BY ID DESC
```

```
SELECT NAME, count(*), avg(POINTS)
   FROM Employee
GROUP BY NAME DESC
```

COMMIT;

Azat Yakupov