CS 348 Lecture 11

DBMS Architecture Overview &

Physical Data Organization

Semih Salihoğlu

Feb 10th 2022

UNIVERSITY OF WATERLOO | DSg Data Systems Group

Welcome Back!

CS 348 Diagram

User/Administrator Perspective

Primary Database Management System Features

- Data Model: Relational Model
- High Level Query Language: Relational Algebra & SQL
- Integrity Constraints
- Indexes/Views
- Transactions

Relational Database Design

- E/R Models
- Normal Forms

How To Program A DBMS (0.5-1 lecture)

- Embedded vs Dynamic SQL
- Frameworks

DBMS Architect/Implementer Perspective

- Physical Record Design
- Query Planning and Optimization
- Indexes
- Transactions

Other (Last 1/2 Lectures)

- Graph DBMSs
- MapReduce: Distributed Data Processing Systems

Announcements

- Assignment 3:
  - Out Feb 14th night (next Monday).
  - Due Feb 15th midnight
- No lecture on Tuesday (Feb 15th) to not distract you.

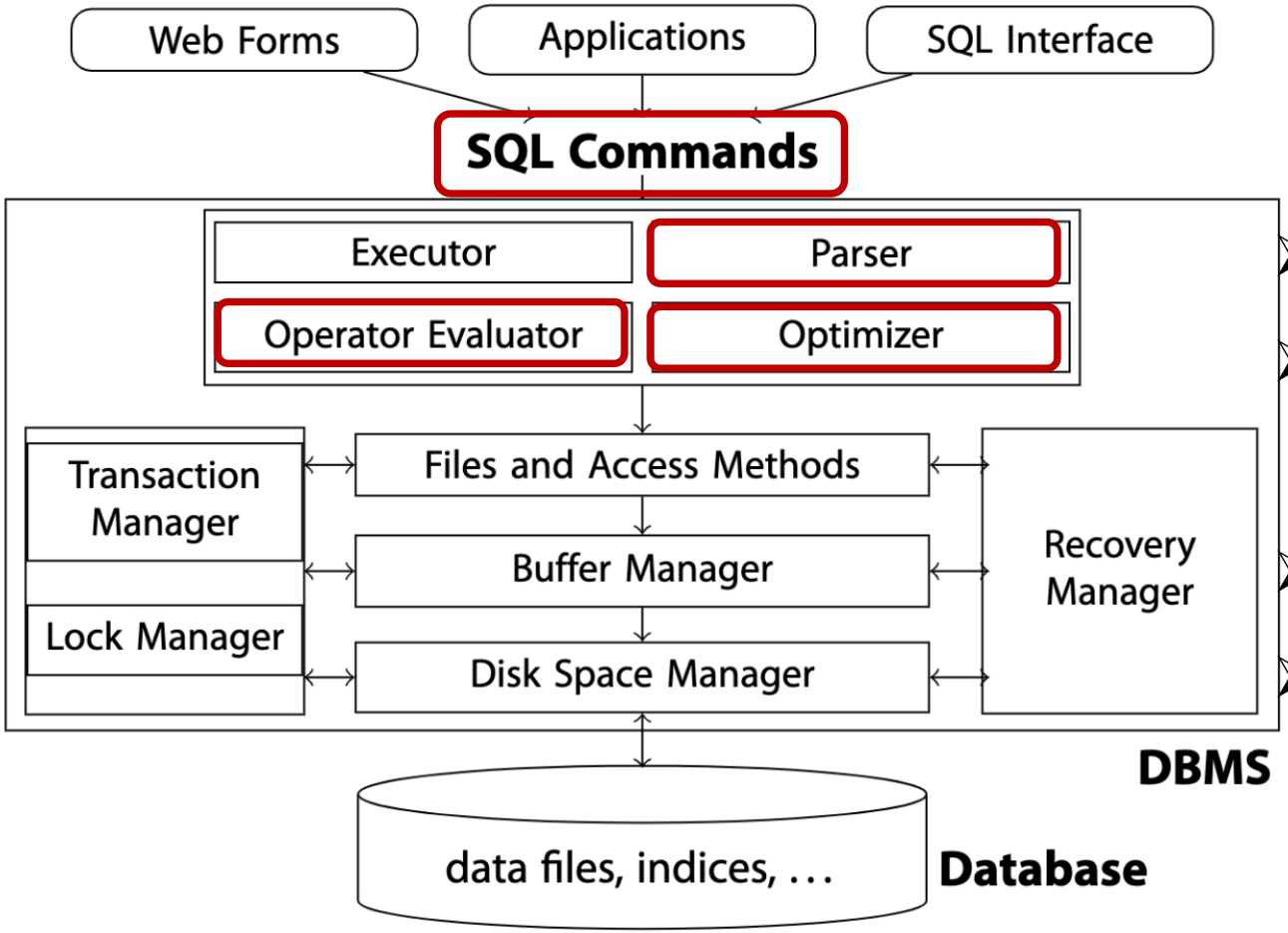
Outline For Today

1. High-level DBMS Architecture
2. Storage Manager and Physical File Organization Designs
   - Physical Operators/Storage Interface
   - Fundamental Property of Storage Devices & Potential Goals of Physical Design
   - Row-oriented Physical Design
   - Colum-oriented Physical Design
   - Hybrid (PAX) Physical Design
   - Designs for Variable-length Fields
   - Designs for NULLs

Outline For Today

1. High-level DBMS Architecture
2. Storage Manager and Physical File Organization Designs
   - Physical Operators/Storage Interface
   - Fundamental Property of Storage Devices & Potential Goals of Physical Design
   - Row-oriented Physical Design
   - Colum-oriented Physical Design
   - Hybrid (PAX) Physical Design
   - Designs for Variable-length Fields
   - Designs for NULLs

High-level DBMS Architecture Overview



- Transformations of Query's Structure
- Text
- An In-Memory Representation
- Logical Plan
- Physical Plan

- Captures overall structure. Many details are system-specific.

Figure from Database Management Systems, Ramakrishnan & Gehrke 3rd edition

Example From GraphflowDB

```
SELECT cid
FROM Customer C, Order O, Product P
WHERE C.cid = O.cid AND O.pid = P.pid
      AND P.name = BookA
```

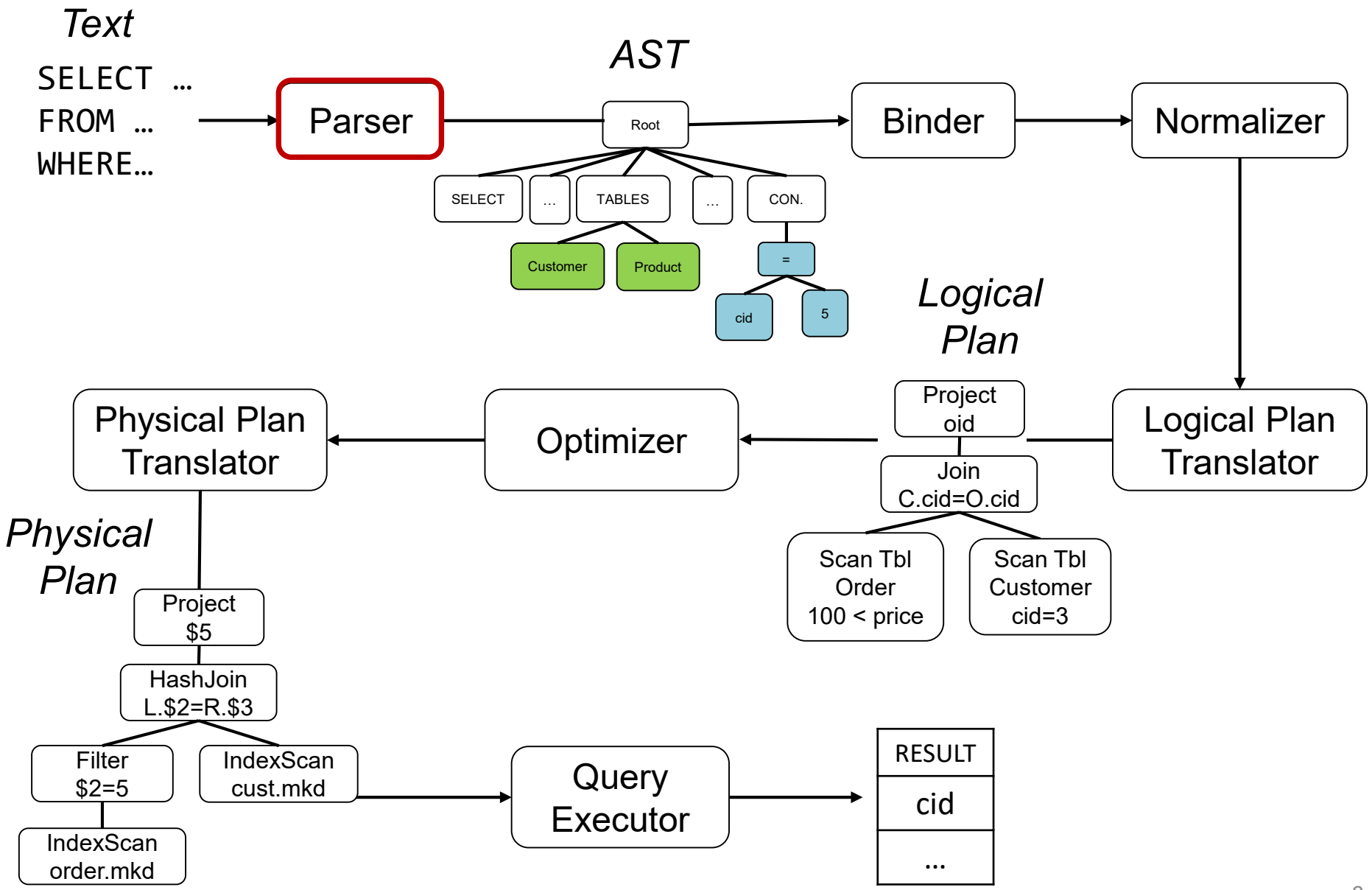
Customer	
cid	name
C1	Alice Munro
…	…

Order			
oid	cid	pid	price
O1	C1	P2	$20
…	…	…	…

Product	
pid	name
P1	WatchB
…	…

- Following example is a read-only query.
- Updates & transactions require additional steps & system components

Overview of Compilation Steps



Parser

- Parses text query into an *abstract syntax tree* based on SQL grammar
- Ex: Snippets from MySQL's SQL grammar

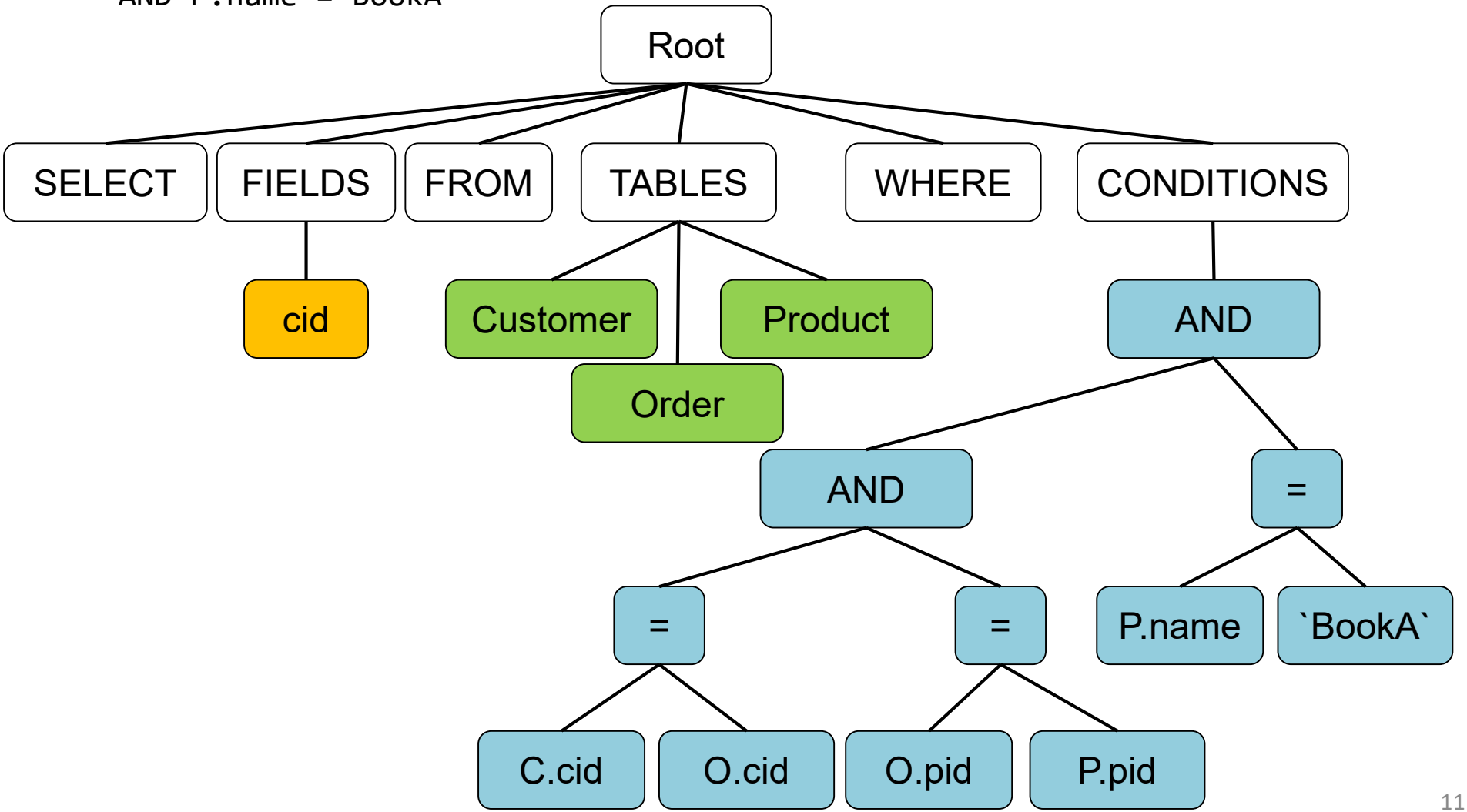
```
parser grammar MySqlParser;

options { tokenVocab=MySqlLexer; }


// Top Level Description

root
    : sqlStatements? (MINUS MINUS)? EOF
    ;

sqlStatements
    : (sqlStatement (MINUS MINUS)? SEMI? | emptyStatement)*
    (sqlStatement ((MINUS MINUS)? SEMI)? | emptyStatement)
    ;

sqlStatement
    : ddlStatement | dmlStatement | transactionStatement
    | replicationStatement | preparedStatement
    | administrationStatement | utilityStatement
    ;
```

```
selectStatement
    : querySpecification lockClause?
    | queryExpression lockClause?
    | querySpecificationNointo unionStatement+
        (
          UNION unionType=(ALL | DISTINCT)?
          (querySpecification | queryExpression)
        )?
        orderByClause? limitClause? lockClause?
    | queryExpressionNointo unionParenthesis+
        (
          UNION unionType=(ALL | DISTINCT)?
          queryExpression
        )?
        orderByClause? limitClause? lockClause?
    ;
```

```
querySpecification
    : SELECT selectSpec* selectElements selectIntoExpression?
      fromClause? groupByClause? havingClause? orderByClause? limitClause?
    | SELECT selectSpec* selectElements
    fromClause? groupByClause? havingClause? orderByClause? limitClause? selectIntoExpression?
    ;
```

Parser

```
SELECT cid
FROM Customer C, Order O, Product P
WHERE C.cid = O.cid AND O.pid = P.pid
      AND P.name = BookA
```



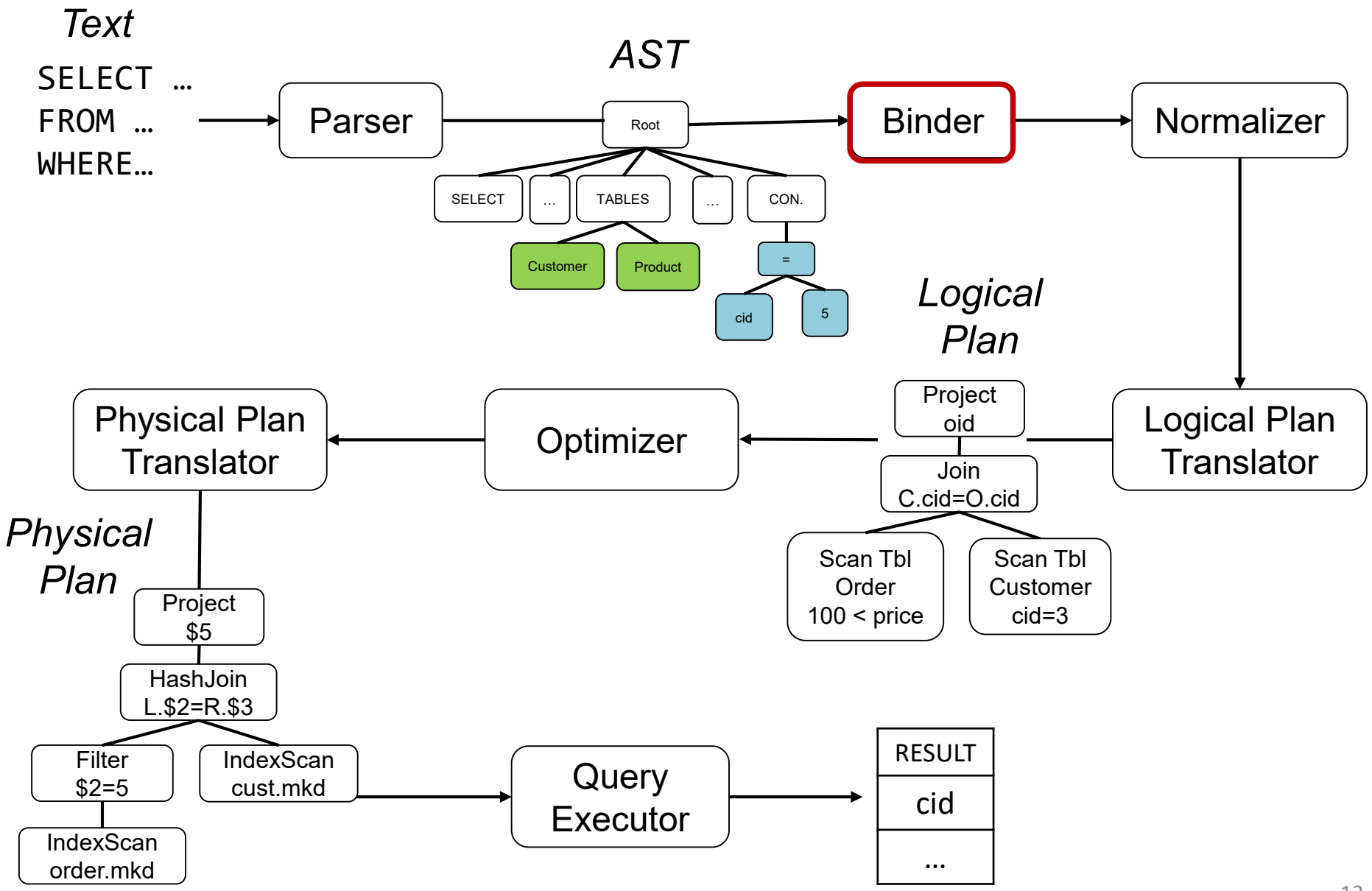
Parser

- Catches syntax errors

```
FROM Customer C, Order O, Product P
SELECT cid
WHERE C.cid = O.cid AND O.pid = P.pid
      AND P.name = BookA
```

42601 syntax error at or near "from"

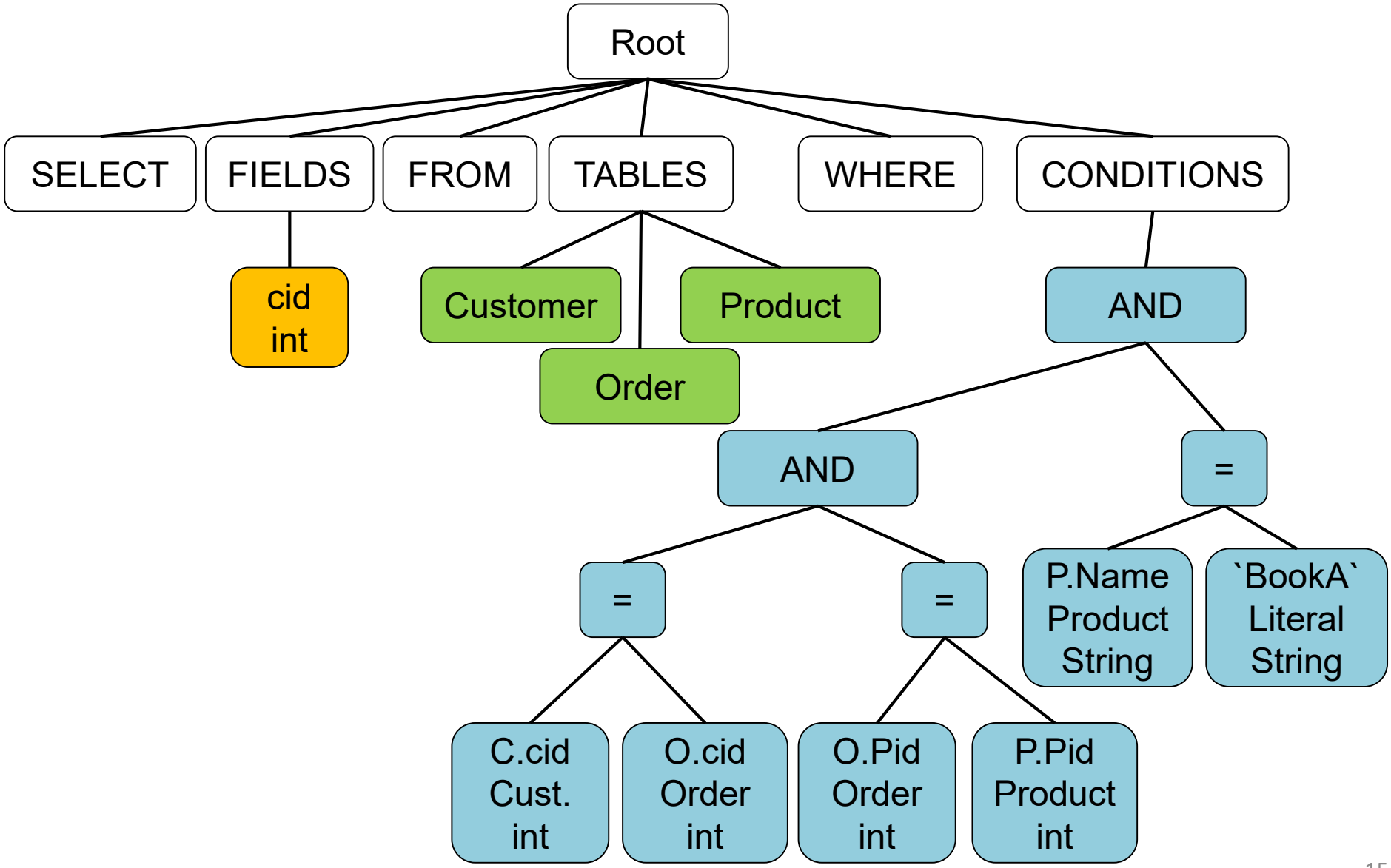
Overview of Compilation Steps



Binder

- Binds type information to columns and expressions
- Uses a system component called *Database Catalog*
  - Keeps “metadata” about database tables and indexes
  - Ex Metadata: Column names and types, constraints, etc.

Binder


Root
SELECT
FIELDS
FROM
TABLES
WHERE
CONDITIONS
cid
int
Customer
Product
Order
AND
AND
=
=
=
P.Name
Product
String
`BookA`
Literal
String
C.cid
Cust.
int
O.cid
Order
int
O.Pid
Order
int
P.Pid
Product
int

Binder

- Catches type errors

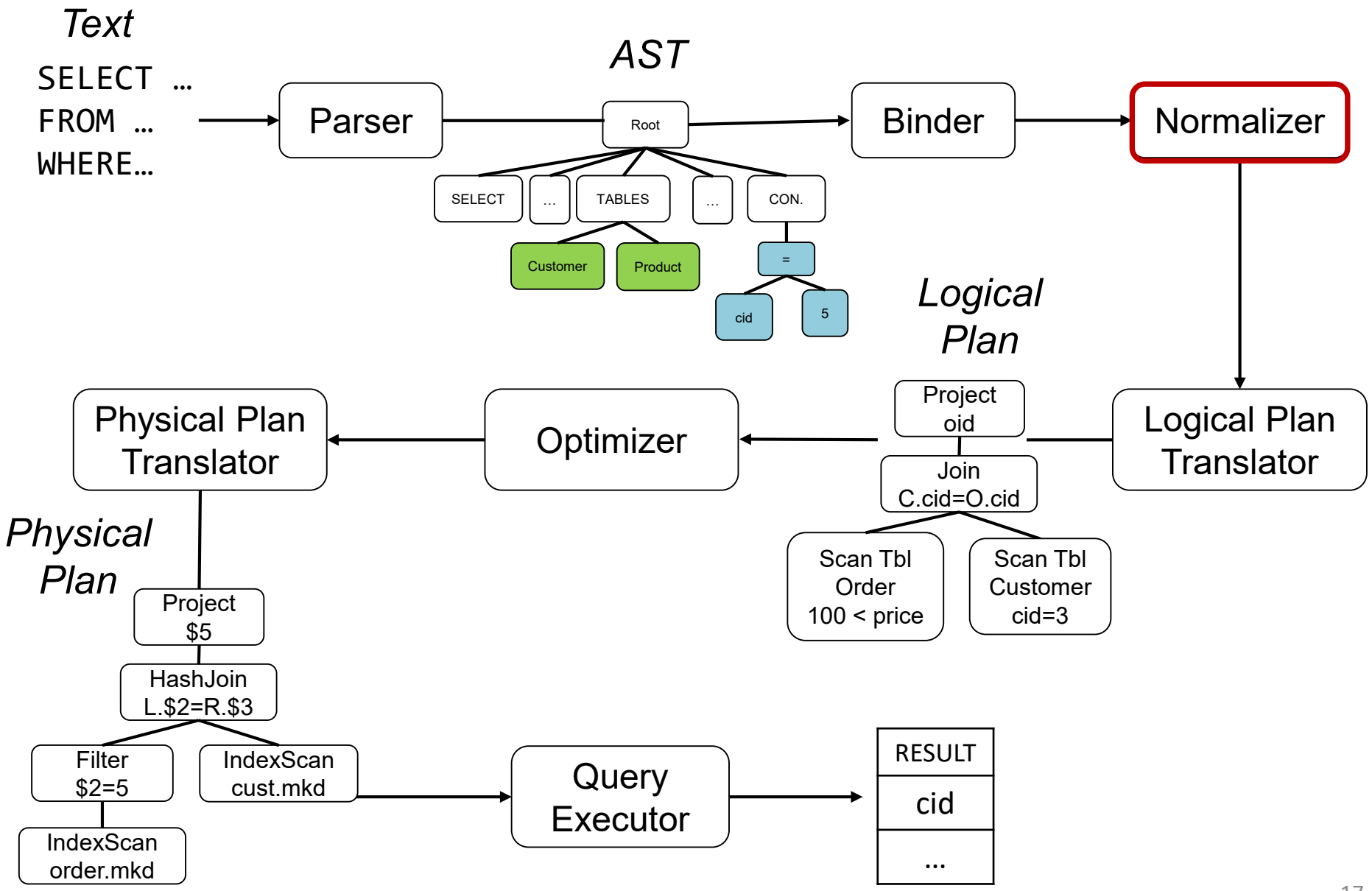
```
SELECT cid
FROM Customer C, Order O, Product P
WHERE C.cid AND O.pid = P.pid
      AND P.name = BookA
```

42804 argument of WHERE must be type boolean, not type integer

```
SELECT cid
FROM Customer C, Order O, Product P
WHERE C.cid + false = O.cid AND O.pid = P.pid
      AND P.name = BookA
```

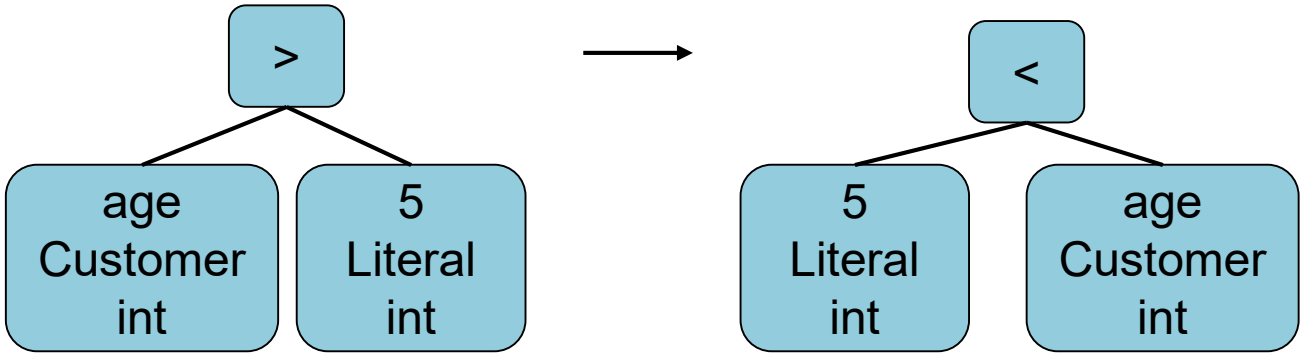
42883 operator does not exist: integer + boolean

Overview of Compilation Steps

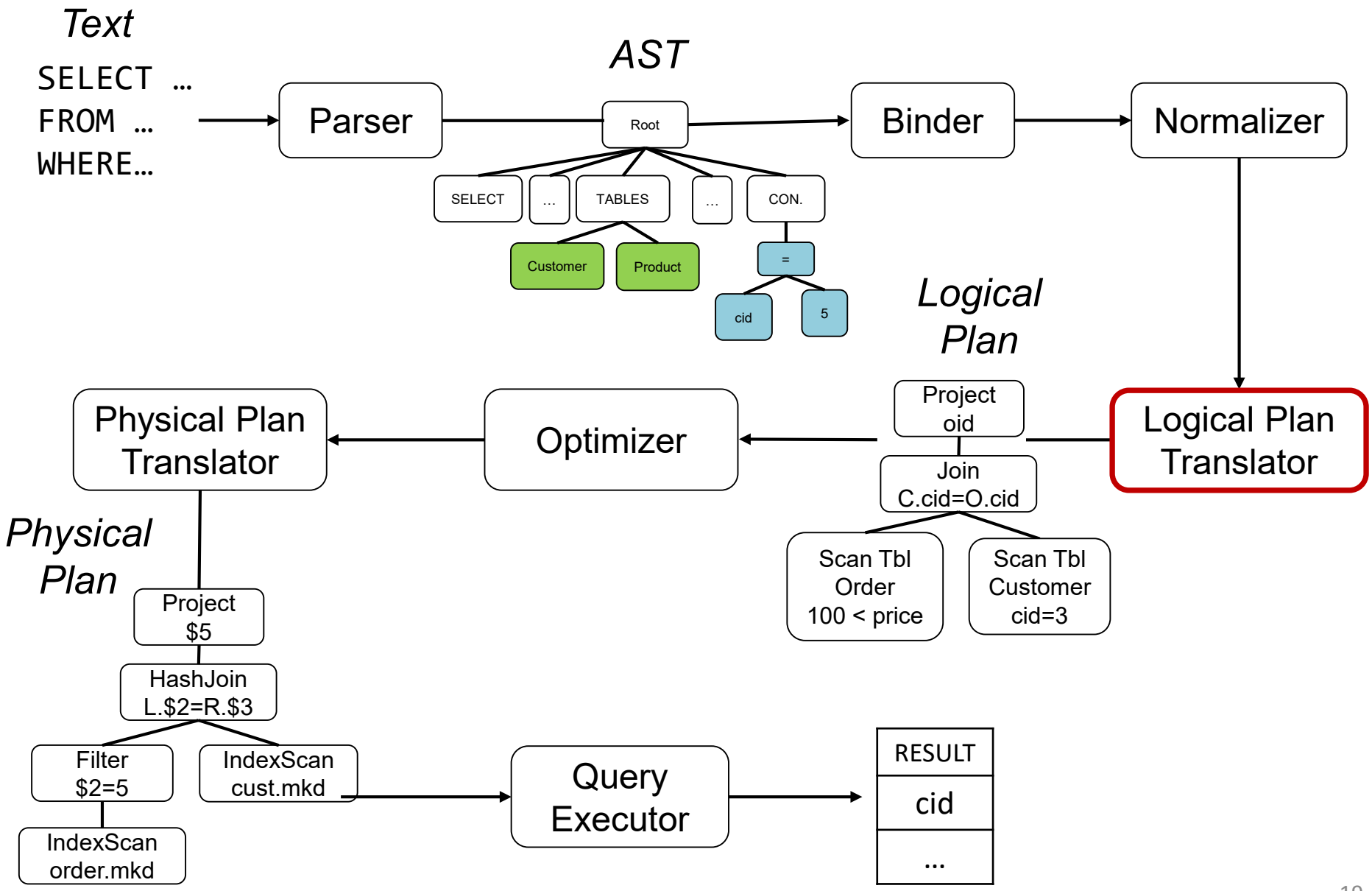


Normalizer

- Normalizes the ast according to several rules
- Examples:
  - Convert all comparisons to < or <=.
  - Perform all constant arithmetic: 5 + 10 => 15
  - Simplify bool expressions that are guaranteed to be true or false.
  - Convert WHERE expression to Conjunctive Normal Form (ANDs of ORs)
  - etc.



Overview of Compilation Steps

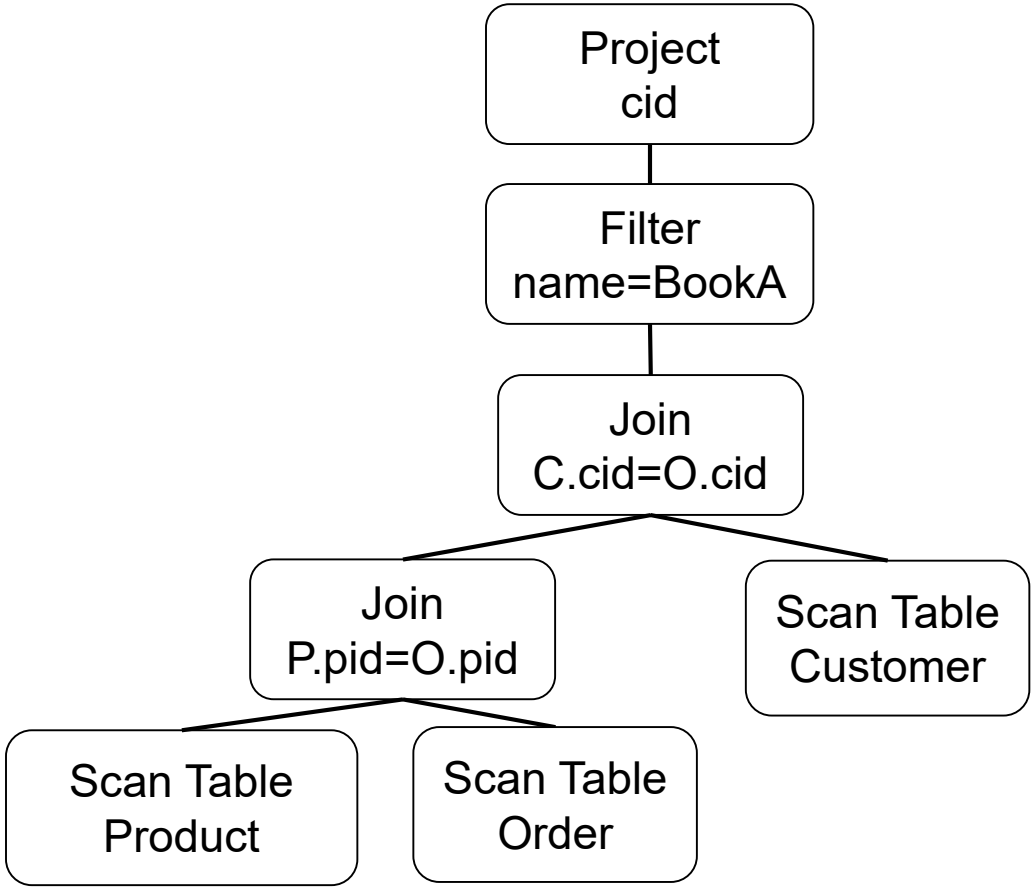


Logical Plan Translator

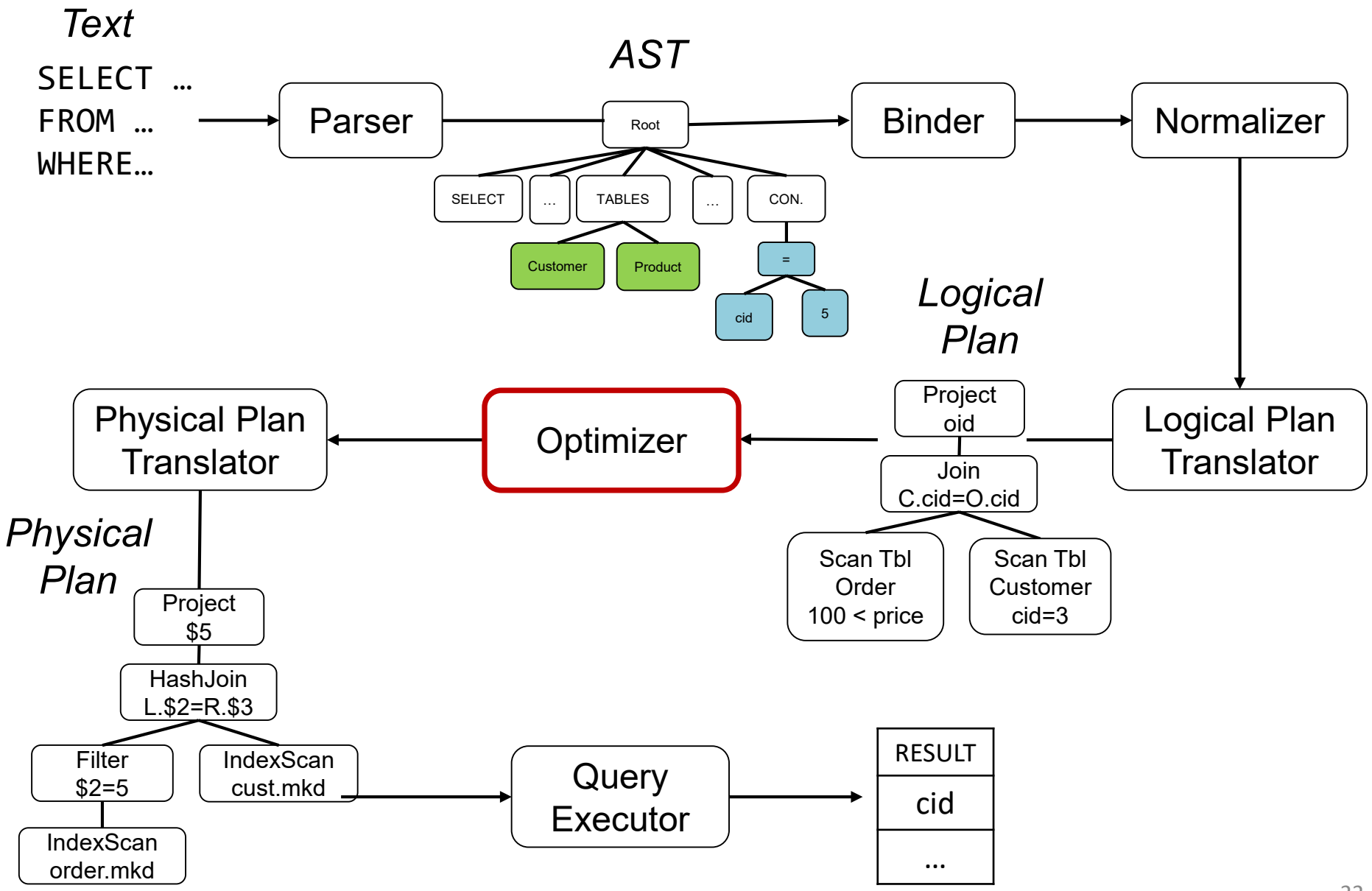
- Maps normalized ast or other in-memory representation to an initial (unoptimized) logical query plan
- Logical Query Plan: Consists of “high-level” operators:
  - Table Scan, Filter, Projection, Join, Group By-And Aggregate, etc.
  - Many but not all operators are based on relational algebra but correspond to SQL clauses
  - Some are not: e.g: LIMIT, SKIP, Order By, etc.

Logical Plan Translator

```
SELECT cid
FROM Customer C, Order O, Product P
WHERE C.cid = O.cid AND O.pid = P.pid
      AND P.name = BookA
```



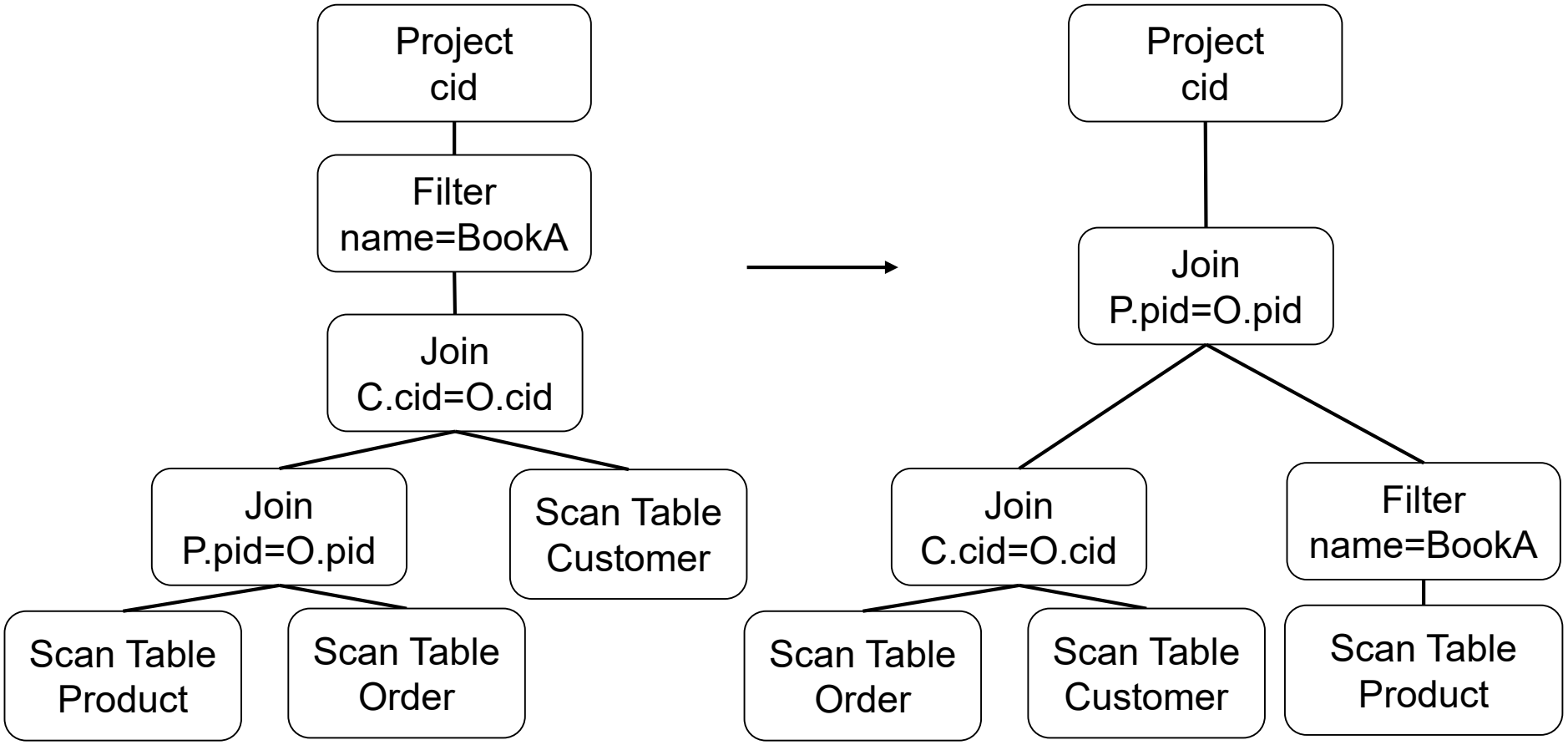
Overview of Compilation Steps



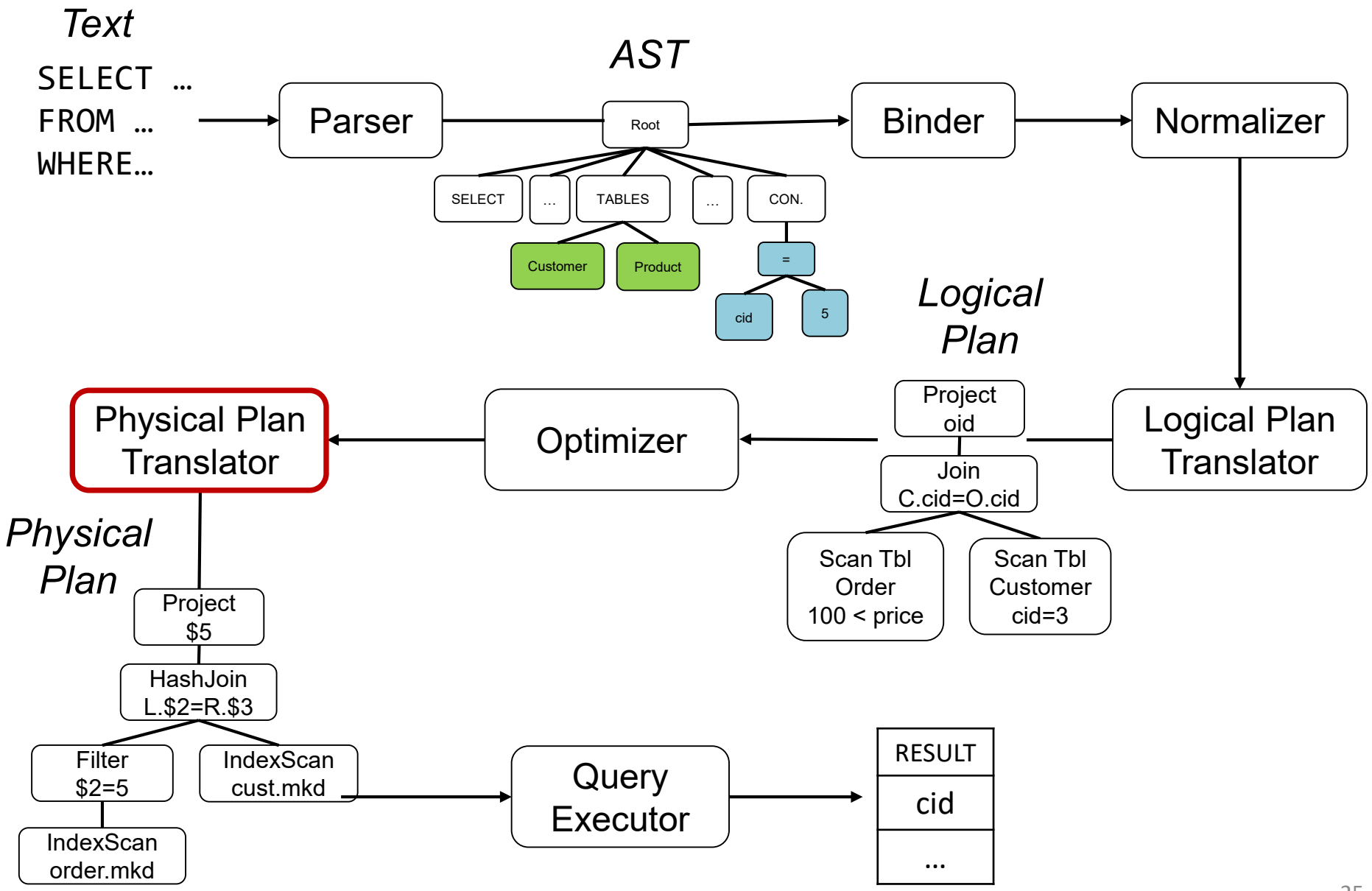
Optimizer

- Optimizes initial logical plan using a set of rules and estimated costs
- Example Optimizations:
  - Pushing filters down in the query plan
  - Pushing projections down
  - Changing the order of the joins
  - Replacing sub-plans with views
  - Identifying and reusing common sub-plans
  - etc..

Example Optimization: Filter Pushdown



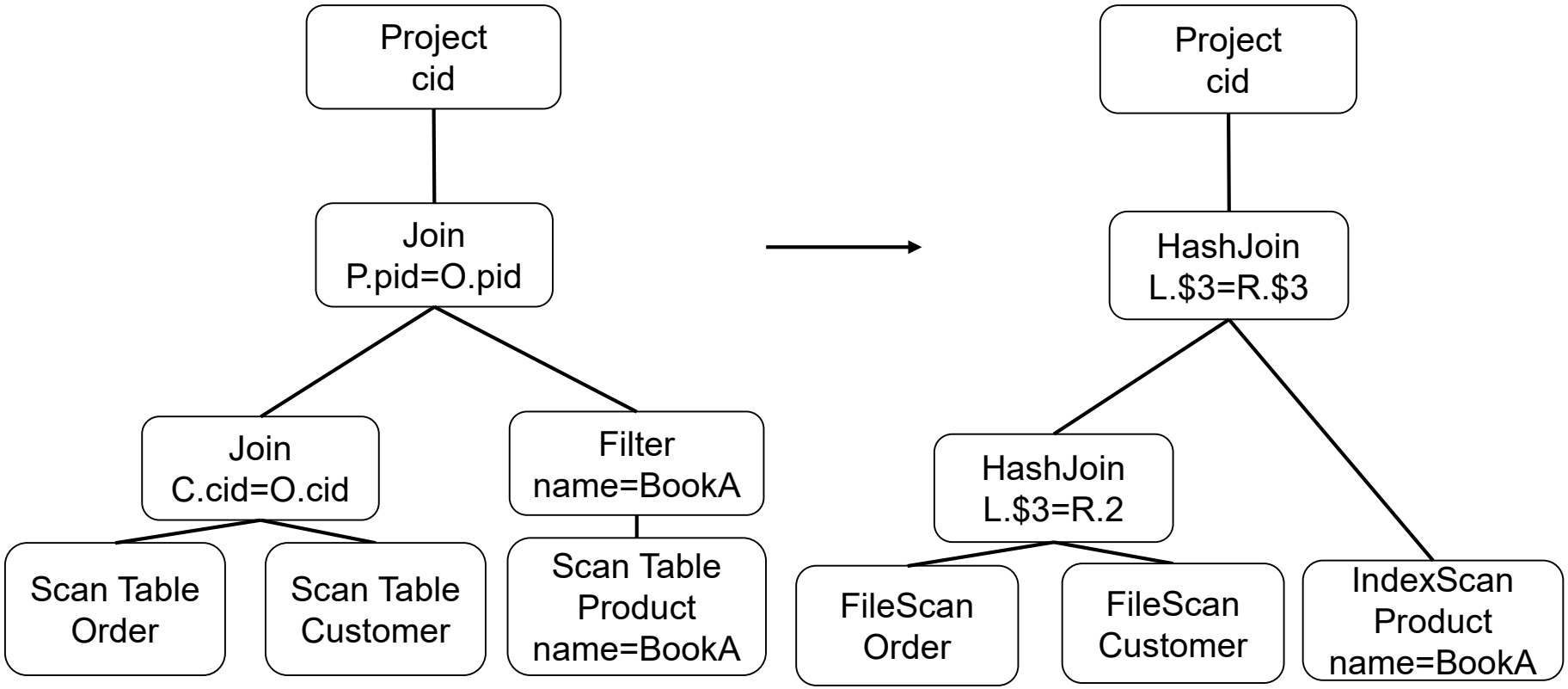
Overview of Compilation Steps



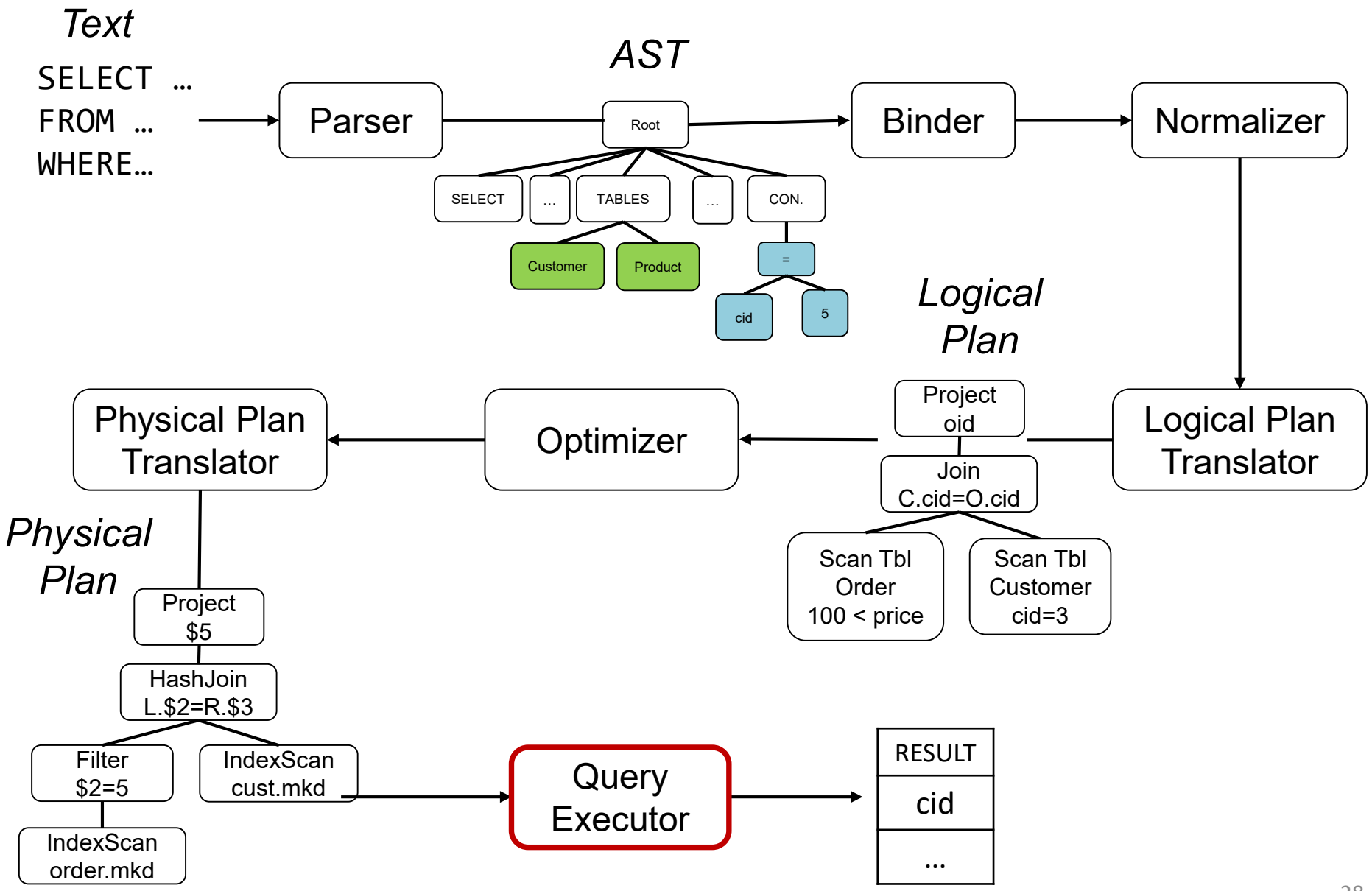
Physical Plan Translator

- Translates logical operators into physical operators
- Logical operators: placeholders to describe the overall query processing algorithm
- Physical operators: classes in the implementation language of DBMS (e.g. C++) that have functions that operate on in-memory tuples
- Each logical op might be implemented with multiple physical ops
  - Scan Table => FileScan or IndexFileScan
  - Join => Hash Join, Merge Join, Index Nested Loop Join, etc.
- Might implicitly do some optimizations

Physical Plan Translator



Overview of Compilation Steps



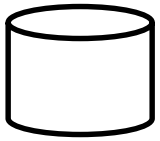
Query Executor

- Executes physical plans
- Implements a task scheduling mechanism:
  - Can have a threadpool of “workers”
  - Workers execute physical plans in parallel and coordinate

Outline For Today

1. High-level DBMS Architecture
2. Storage Manager and Physical File Organization Designs
   - Physical Operators/Storage Interface
   - Fundamental Property of Storage Devices & Potential Goals of Physical Design
   - Row-oriented Physical Design
   - Colum-oriented Physical Design
   - Hybrid (PAX) Physical Design
   - Designs for Variable-length Fields
   - Designs for NULLs

Persistent Data Storage

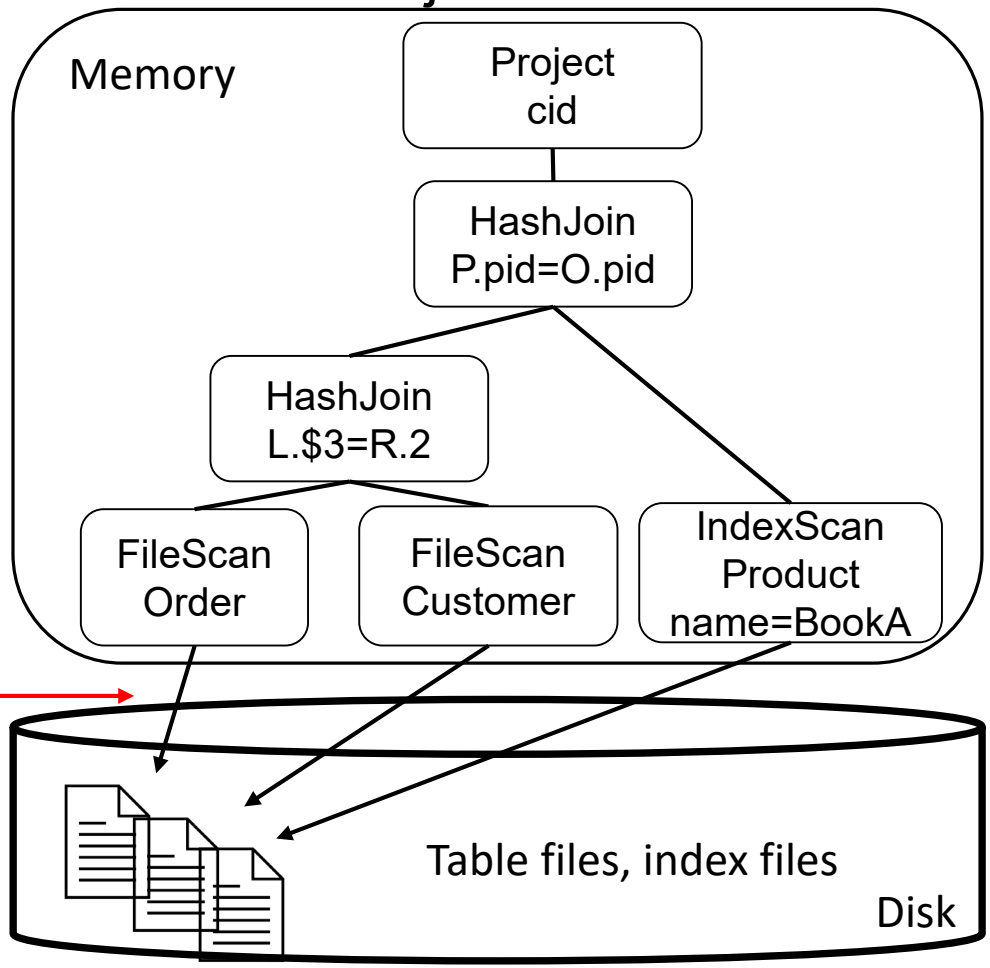
- DBMSs store data persistently on durable storage devices
  - Hard disks or flash disks or more recently non-volatile memory
  - Hence our drawing of DBMSs as disks 
- But query processing is computation and all computation ultimately happens in memory.

Physical Operator/Storage Interface

- RDBMSs store data in table/index files on disk (indices next lecture)
- Several physical operators access data in files
  - Most importantly scans but sometimes also joins

Recall: Physical operators: classes in the implementation language of DBMS (e.g. C++) that have functions that operate on in-memory tuples

File accesses happen via classes implemented in *StorageManager* component (aka FileManager, FileAccessMethods etc.)



Physical Operator/Storage Interface

```
Class ScanTable : public PhysicalOperator {

string tableFilename;
DBFileHandle* dbFH; // Interface to access storage
int nextRecordID;
// Assume parent operator has access to outTuple as well
Tuple* outTuple;

void init() {
   dbFH = new DBFileHandle(tableFilename); // e.g., order.mkd
   nextRecordID = 0;}

// each operator in the tree calls its child to get next Tuple
Tuple* getNextTuple() {
  if (nextRecordId < dbFH.maxRecordID) {
    dbFH.readRecord(&outTuple, nextRecordID);
    nextRecordID++;
    return &tuple;
  }
}}
```

will show a sample pseudocode later

File Access Functions

- Functions such as readRecord in DBFileHandle-like classes that form the operator/storage interface know about physical data organizations (and operators don't):
  1. Record layout in files
  2. Page/Block layout of records in files
- Other Possible operator/storage interface functions:
  - readRecords(int beginRecordID, int endRecordID)
  - readRecordsInNextPage(int pageID) (see next slide)
  - etc…

Question 1: Why is there another unit of data called page/block?
Question 2: Potential goals of DBMS implementors when designing physical layouts?
Question 3: Common record and page layouts?

Outline For Today

1. High-level DBMS Architecture
2. Storage Manager and Physical File Organization Designs
   - Physical Operators/Storage Interface
   - Fundamental Property of Storage Devices & Potential Goals of Physical Design
   - Row-oriented Physical Design
   - Colum-oriented Physical Design
   - Hybrid (PAX) Physical Design
   - Designs for Variable-length Fields
   - Designs for NULLs

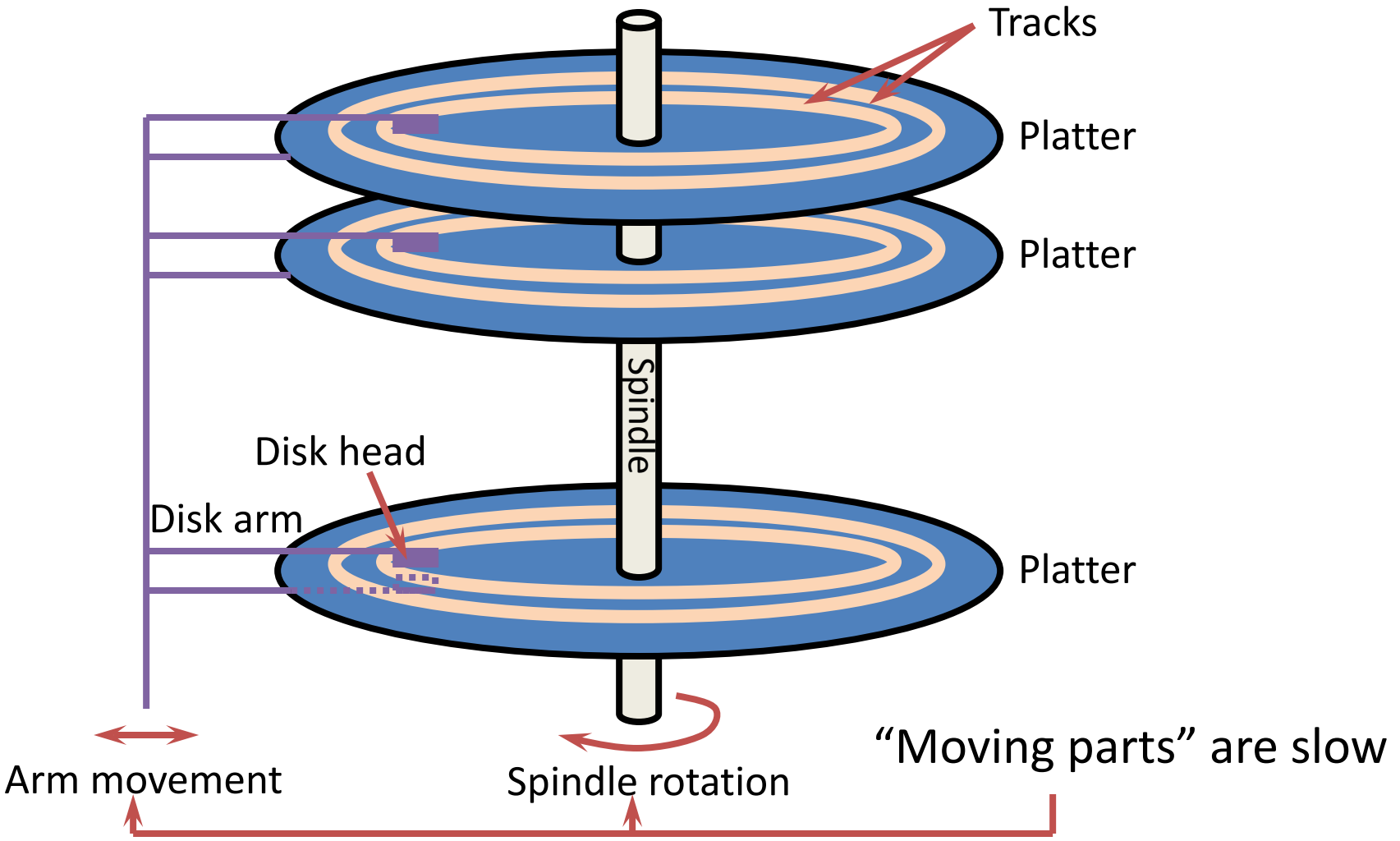
Fundamental Property of Storage Devices

Random reads is slow! Sequential reads is fast!

Take this very seriously!

- Holds for RAM, magnetic disks, or flash disks.
- Examples: Read 1GB of 64bit ints randomly in RAM vs sequentially and you might get ~10x difference.
  - Exercise: Test this and let me know the exact answer.
- Worse diff. depending on read size in hard/flash disks (e.g. ~1000x)
- Question 1: Why is there another unit of data called page/block?
- Important Consequences:
  - OS reads/writes data from/to disk in pages (e.g., 4K)
  - DBMS should also read/write data in pages (multiple of OS page sizes, 4K, 8K, 12K etc. but not 2K)

Why Random Reads Are Slow on Disks?



Why Random Reads Are Slow for RAM?

What Every Programmer Should Know About Memory

Ulrich Drepper
Red Hat, Inc.
drepper@redhat.com

November 21, 2007

Abstract

As CPU cores become both faster and more numerous, the limiting factor for most programs is now, and will be for some time, memory access. Hardware designers have come up with ever more sophisticated memory handling and acceleration techniques–such as CPU caches–but these cannot work optimally without some help from the programmer. Unfortunately, neither the structure nor the cost of using the memory subsystem of a computer or the caches on CPUs is well understood by most programmers. This paper explains the structure of memory subsystems in use on modern commodity hardware, illustrating why CPU caches were developed, how they work, and what programs should do to achieve optimal performance by utilizing them.

1 Introduction

In the early days computers were much simpler. The various components of a system, such as the CPU, memory, mass storage, and network interfaces, were developed together and, as a result, were quite balanced in their performance. For example, the memory and network interfaces were not (much) faster than the CPU at providing data.

This situation changed once the basic structure of computers stabilized and hardware developers concentrated on optimizing individual subsystems. Suddenly the performance of some components of the computer fell significantly behind and bottlenecks developed. This was especially true for mass storage and memory subsystems which, for cost reasons, improved more slowly relative to other components.

The slowness of mass storage has mostly been dealt with using software techniques: operating systems keep most often used (and most likely to be used) data in main memory, which can be accessed at a rate orders of magnitude faster than the hard disk. Cache storage was added to the storage devices themselves, which requires no changes in the operating system to increase performance.[1] For the purposes of this paper, we will not go into more details of software optimizations for the mass storage access.

Unlike storage subsystems, removing the main memory as a bottleneck has proven much more difficult and almost all solutions require changes to the hardware. Today these changes mainly come in the following forms:

- RAM hardware design (speed and parallelism).
- Memory controller designs.
- CPU caches.
- Direct memory access (DMA) for devices.

For the most part, this document will deal with CPU caches and some effects of memory controller design. In the process of exploring these topics, we will explore DMA and bring it into the larger picture. However, we will start with an overview of the design for today's commodity hardware. This is a prerequisite to understanding the problems and the limitations of efficiently using memory subsystems. We will also learn about, in some detail, the different types of RAM and illustrate why these differences still exist.

This document is in no way all inclusive and final. It is limited to commodity hardware and further limited to a subset of that hardware. Also, many topics will be discussed in just enough detail for the goals of this paper. For such topics, readers are recommended to find more detailed documentation.

When it comes to operating-system-specific details and solutions, the text exclusively describes Linux. At no time will it contain any information about other OSes. The author has no interest in discussing the implications for other OSes. If the reader thinks s/he has to use a different OS they have to go to their vendors and demand they write documents similar to this one.

One last comment before the start. The text contains a number of occurrences of the term "usually" and other, similar qualifiers. The technology discussed here exists

[1] Changes are needed, however, to guarantee data integrity when using storage device caches.

Fundamentally because of better CPU cache utilization, i.e., L1/L2 caches in modern CPUs

Latency Numbers Every Programmer Should Know

```
Latency Comparison Numbers
--------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns                      14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns                      20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy             3,000   ns        3 us
Send 1K bytes over 1 Gbps network       10,000   ns       10 us
Read 4K randomly from SSD*             150,000   ns      150 us          ~1GB/sec SSD
Read 1 MB sequentially from memory     250,000   ns      250 us
Round trip within same datacenter      500,000   ns      500 us
Read 1 MB sequentially from SSD*     1,000,000   ns    1,000 us    1 ms  ~1GB/sec SSD, 4X memory
Disk seek                           10,000,000   ns   10,000 us   10 ms  20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns   20,000 us   20 ms  80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns  150,000 us  150 ms

Notes
-----
1 ns = 10^-9 seconds
1 us = 10^-6 seconds = 1,000 ns
1 ms = 10^-3 seconds = 1,000 us = 1,000,000 ns

Credit
------
By Jeff Dean:               http://research.google.com/people/jeff/
Originally by Peter Norvig: http://norvig.com/21-days.html#answers
```

Q2: Potential Goals of Physical Data Design

1. Minimize I/O!
   i. Design files to pack records/columns into few number of pages
   ii. Keep related data close in storage devices!
   iii. Cache pages of data that is likely to be used
2. Maximize sequential reads from storage devices!
3. Simplicity/speed of accessing values
   - E.g., don't compress using a complex compression scheme

These are general guidelines and not necessarily independent.

Q3: Common Record and Page Layouts

1. Row-oriented
2. Column-oriented
3. Hybrid (PAX) Row & Column-oriented Fixed-length Values
4. Designs for Variable-length Values
5. Designs for NULL Values

Several Simplification Assumptions

- Each field/column is of constant size
- No compression
- Assume files are broken into pages of size H (e.g., 4K)
- Assume each field and entire rows of each table easily fit into a page << H (e.g., fields ~10s of bytes rows 100s of bytes)
  - Assume each row takes T many bytes

Outline For Today

1. High-level DBMS Architecture
2. Storage Manager and Physical File Organization Designs
   - Physical Operators/Storage Interface
   - Fundamental Property of Storage Devices & Potential Goals of Physical Design
   - Row-oriented Physical Design
   - Colum-oriented Physical Design
   - Hybrid (PAX) Physical Design
   - Designs for Variable-length Fields
   - Designs for NULLs

Row-Oriented N-ary Storage Model (NSM)

- Each page contains ~ H/T many rows and all fields of each row

Customer		
cid : int64 (8 bytes)	name : string (60 chars)	isGoldMember : bool (1 byte)

101	Alice Munro\0\0\0….\0	1	201	Carl Sagan\0\0\0…\0	0
103	Bob \0\0\0….\0	1	…	…	…

(Example Disk Page, e.g., in file customer.mkd)

- Note: because records are fixed length, can read any field with simple arithmetic (*not a consequence of NSM*)
  - E.g: 5th records name field at offsets: 4*(8+60+1) + 8 = 284 to 344

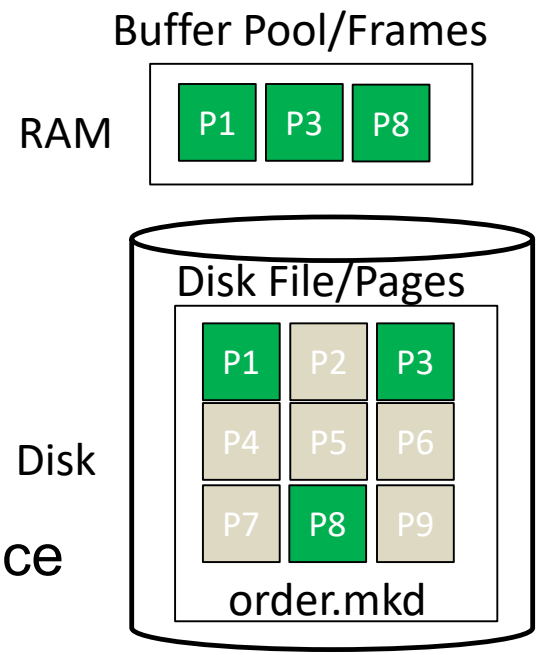
Pros/Cons of NSM

- Pro: Good for queries that access multiple or all fields of tuples
- Con: If queries access a single field, e.g., counting numGoldMembers, “effective read size” can be small
  - E.g. 1/69 bytes read is useful, in columnar storage you can do 69x less I/O to do the same count.
- Con: For some workloads higher # I/O => lower (buffer) cache utilization (next slide)

Buffer Manager & Ex Op/Storage Interface Code

- DBMSs have a *buffer manager* (BM) component that keeps a set of file pages in memory to reduce I/O.
- Goal of BM: keep pages that are likely to be accessed in memory
  - Implement a caching alg, e.g., Least-Recently Used (LRU)

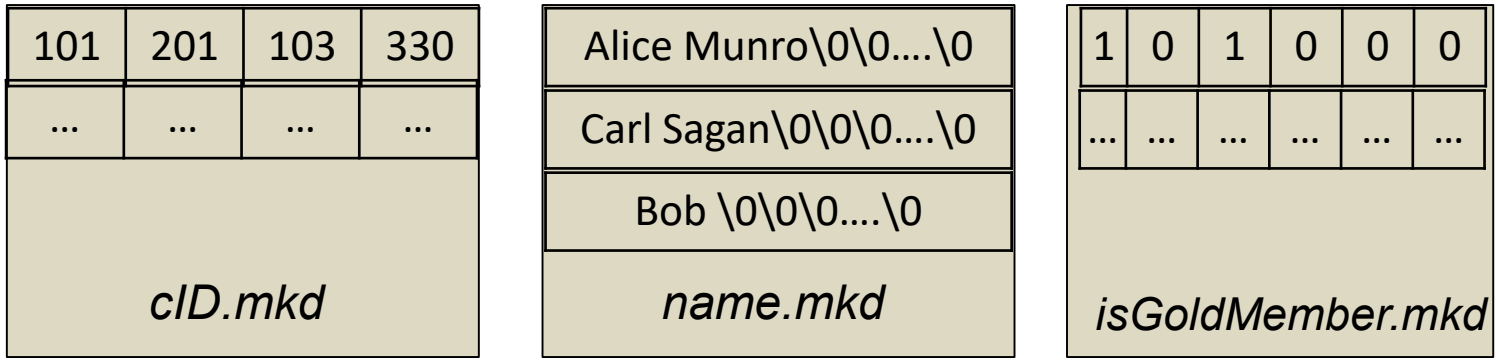
```
class DatabaseFileHandle : {
string filename; // e.g., order.mkd
int tupleLen; // e.g., 69
BufferManager bm;
void readRecord(Tuple* outTuple, recordID) {
   (pageIdx, offset)= calculatePageInfo(recordID)
   Frame* frame = bm.pinPage(pageIdx);
   memcpy(outTuple, frame>bytes[offset], tupleLen)
   bm.unpin(frame); }}
```



- Pin: put disk page into buffer pool until further notice
  - possibly evicts an existing page
- Unpin: can now remove the page from buffer pool if needed

Column-Oriented Storage Design

- One file for each field of table (~H/field-size rows worth of data):



- Pros: Good for queries that access few columns
  - Can reduce I/O significantly
  - Better sequential reads when pages are in memory

```
Select count(*)
from customer
where isGoldMember
```

- Pros: Easier to apply compression because each page is very "homogenous"
- Cons: Bad for queries that access all columns (1 I/O in row-oriented storage could be done with # cols I/O).

PAX Hybrid Storage Design

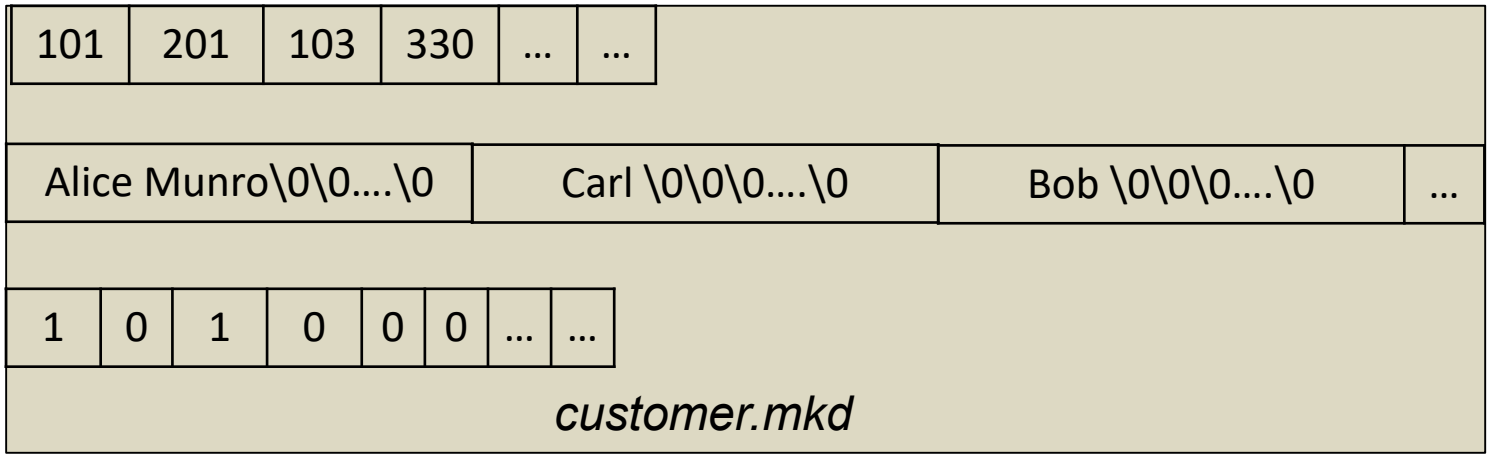
- PAX: Partition Attributes Across (Ailamaki et al. VLDB 2001)
  - All fields of a row in the same page
  - But pages internally organized by columns


Weaving Relations for Cache Performance

Anastassia Ailamaki[1] David J. DeWitt Mark D. Hill Marios Skounakis

Abstract

1 Introduction



- Better sequential reads for queries that read a single column
- Gives the advantage of having all fields of a row in the same page
- But still more I/O than pure columnar storage for some queries

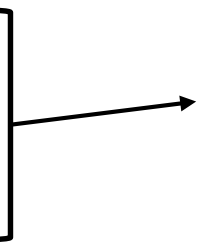
Outline For Today

1. High-level DBMS Architecture
2. Storage Manager and Physical File Organization Designs
   - Physical Operators/Storage Interface
   - Fundamental Property of Storage Devices & Potential Goals of Physical Design
   - Row-oriented Physical Design
   - Colum-oriented Physical Design
   - Hybrid (PAX) Physical Design
   - Designs for Variable-length Fields
   - Designs for NULLs

Variable-length Values

- Pages are often more complex than examples in these slides
- One complexity: variable-length fields
- Assume row-oriented NSM storage
- Approaches to encode var-length fields, e.g., var-len strings, in pages

1. Delimiters
2. Field Offsets

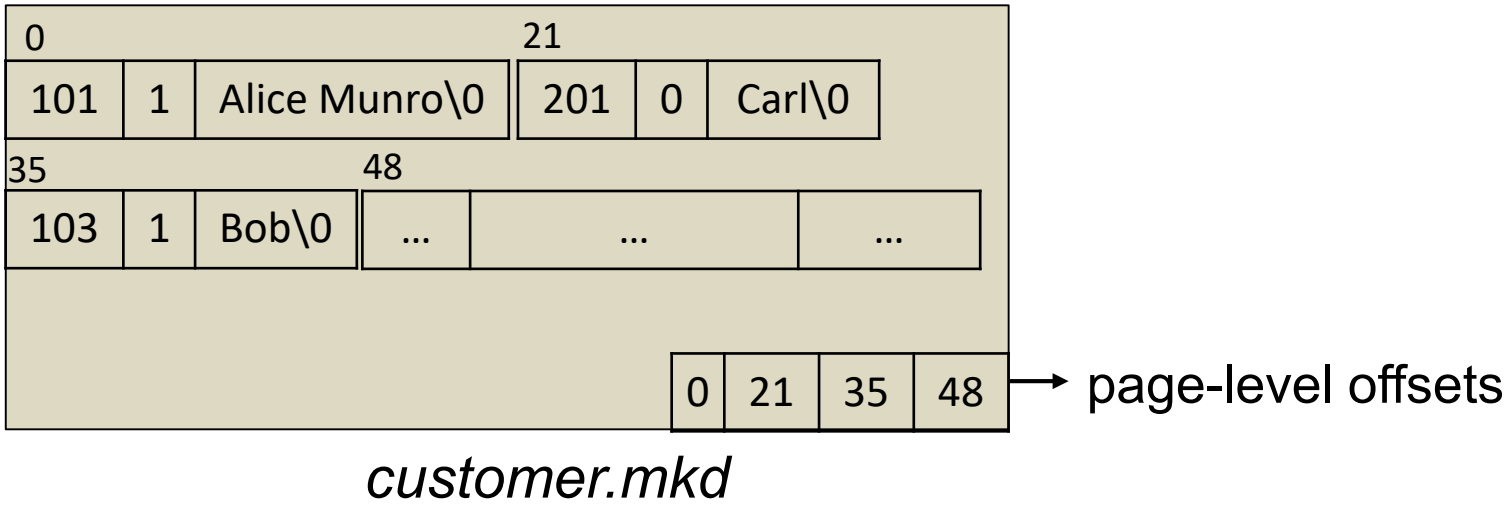


Both:
(i) put first fixed-len fields and then var-len fields when encoding each row
(ii) require storing page-level offset of each row

3. Pointers to Overflow Pages

Delimiter Approach

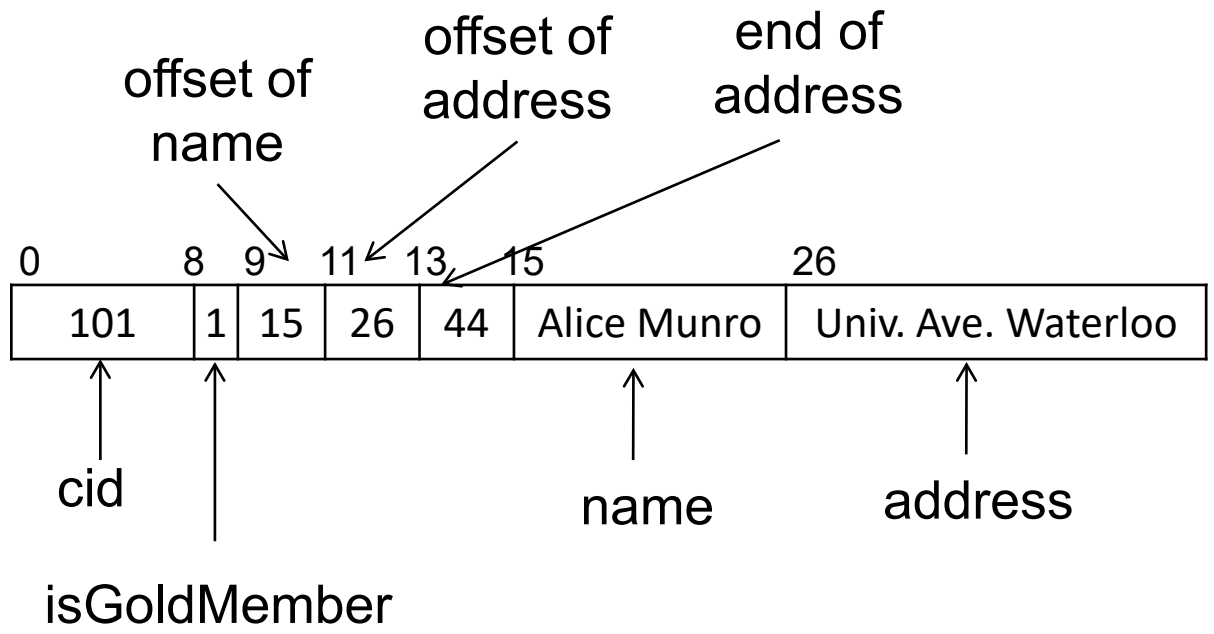
- Example Delimiter: \0 character



- Why are page offsets at the end and fields in the beginning of pages?
  - Can grow them separately without sliding data within pages.
- Advantage of storing fixed-len fields and then var-len fields:
  - Can read fixed-len fields w/ page-level offsets +arithmetic
  - e.g., isGoldMember of rowID 2: offset[2] + 8 = 35+8=43.
  - no need to scan over var-len fields

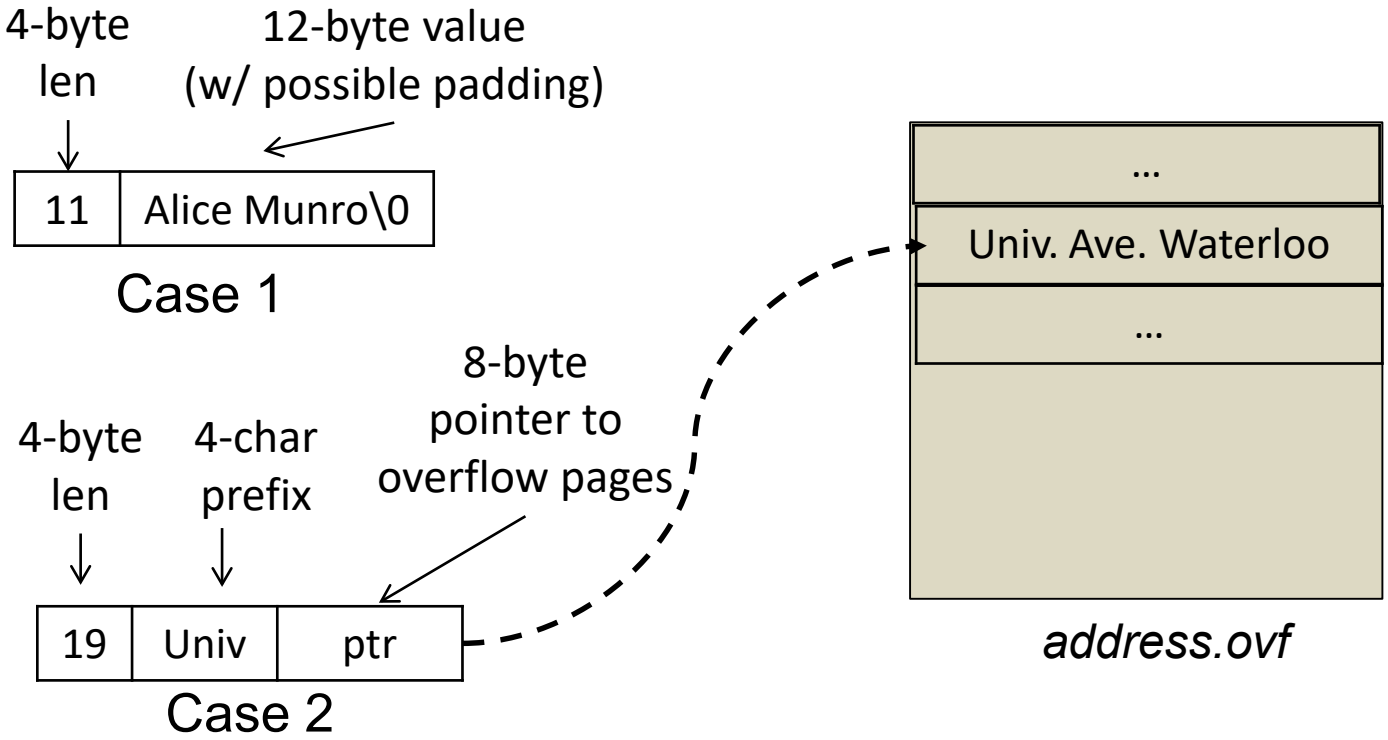
Field Offsets Approach

- Suppose there is a second var-len field called address
- Page structure stays same except each row would be encoded as:



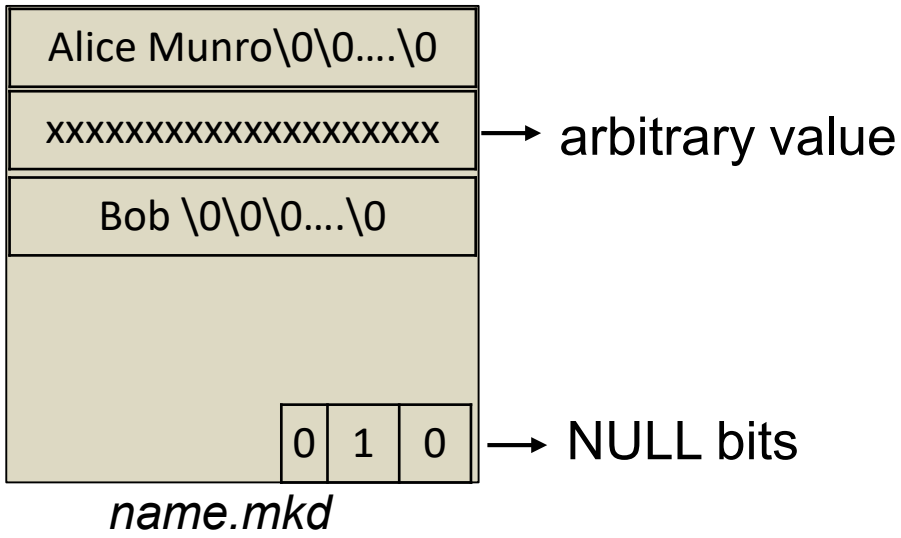
Overflow Pages

- Keep variable length fields, in particular strings, fixed length
- Any overflow points to separate overflow pages
- E.g., Strings in Graphflow (originally by Thomas Neumann)
  - 16 bytes fixed part with 2 possible cases
  - Pro: original pages are simpler; Con: More I/O for reading large strings



NULL Values

- Several possible approaches. E.g:
  - Store special NULL values for each data type as regular values
    - E.g.: -2^63 for 8-byte int
  - Store NULL bits in the page
    - Assume columnar storage and (fixed-len) name can be NULL
    - 1 means NULL 0 means NULL
  - Other alternatives are possible



Summary of Physical Data Design Alternatives

- High-level: row-oriented vs columnar but row-oriented can be hybrid (PAX)
- Variable-length fields and fields that can be NULL require more sophisticated design
- For any physical design we showed, other alternatives are possible
- In practice physical design is principled art!
- Many tradeoff, choices, and complications exist:
  - Deletions of tuples (can leave gaps in pages or shift tuples)
  - Fields larger than page sizes
  - Tuple versions (some systems, e.g., Postgres, support multiple version of tuples so users can keep track of how each record changed)
  - Compression
  - Some crazy ideas put rows from multiple tables into same pages
  - Many other designs