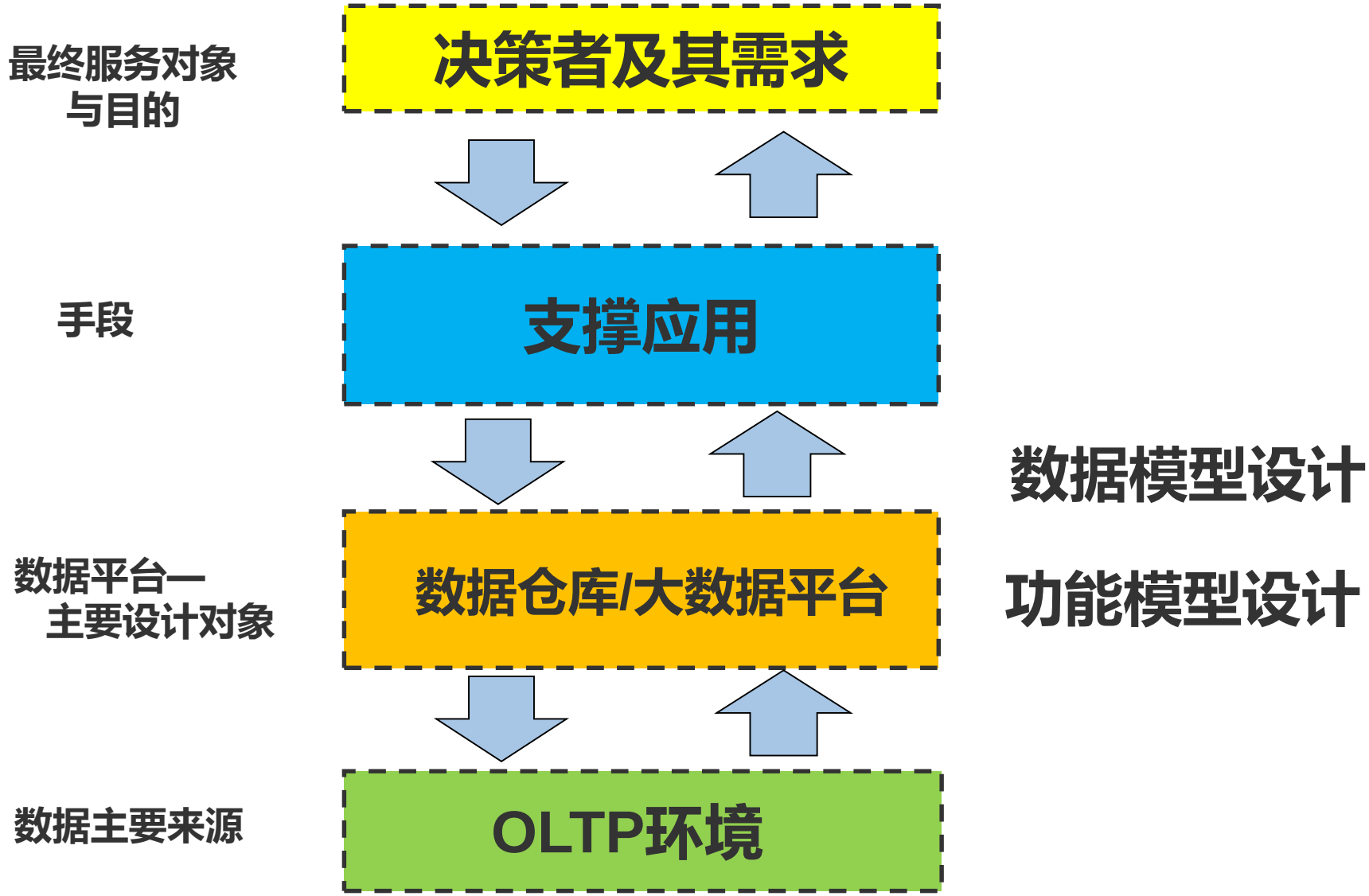
数据仓库与大数据工程

Data Warehouse and Big Data Engineering

第5部分 数据模型及设计方法

北京交通大学计算机与信息技术学院

数据仓库与大数据平台设计对象



内容提纲

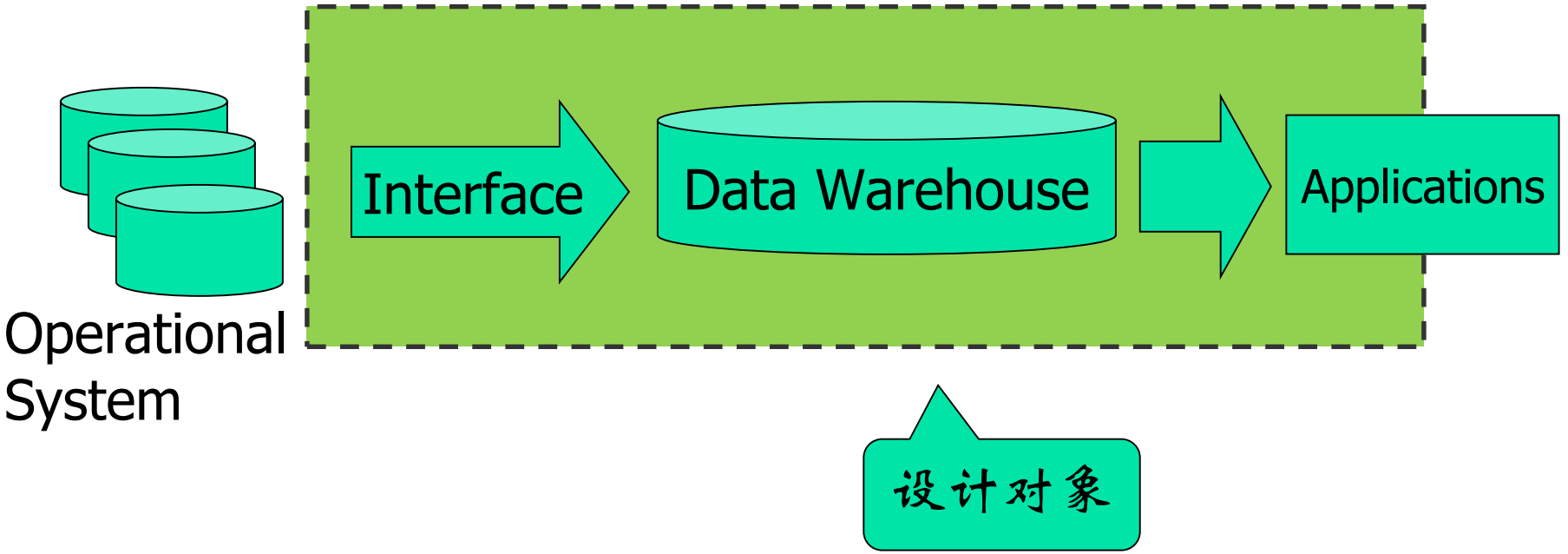
- 系统设计流程
- 数据模型基本概念
- 数据模型设计
- 数据模型设计与性能优化
- 元数据模型

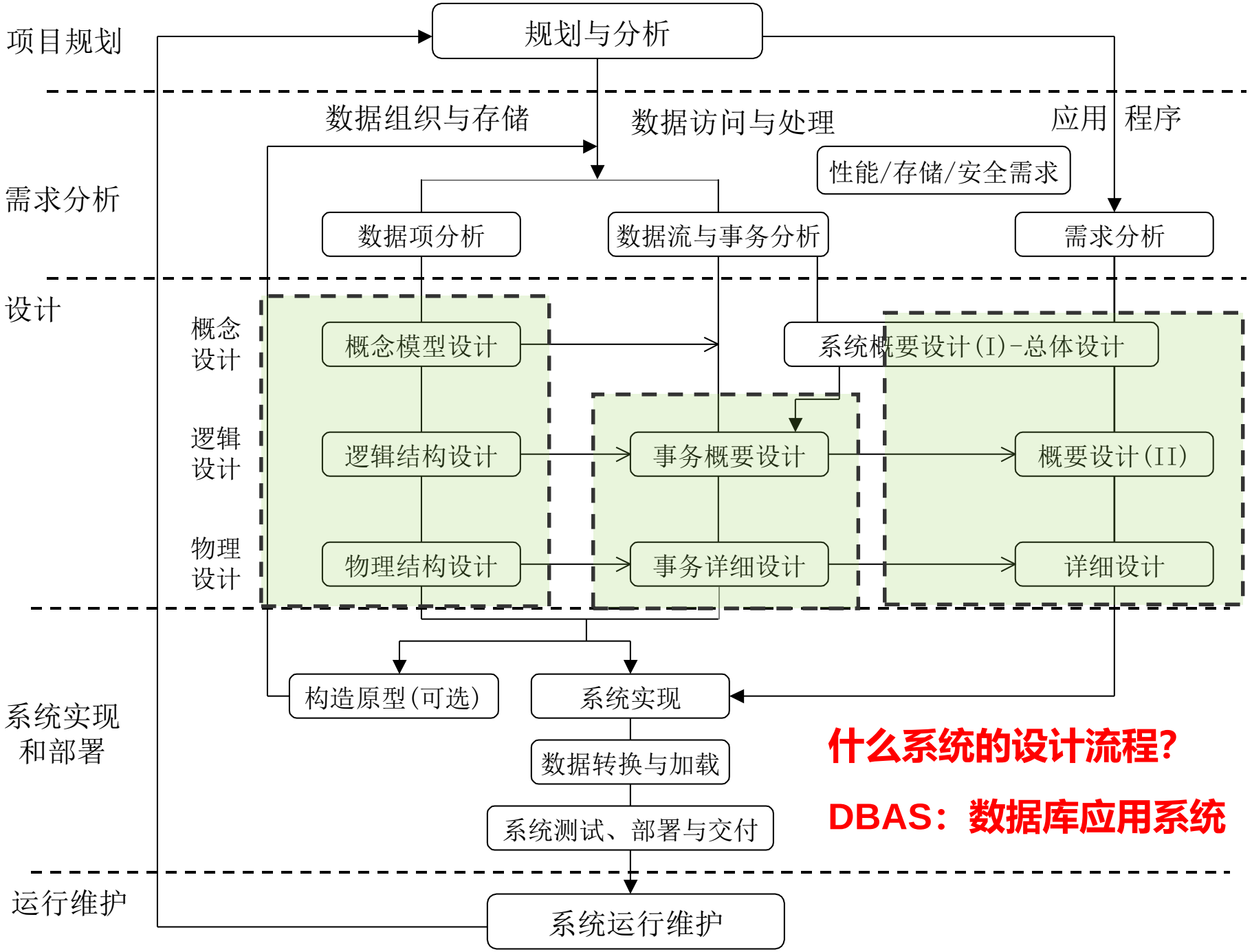
内容提纲

- 系统设计方法概述
- 数据模型基本概念
- 数据模型设计
- 数据模型设计与性能优化
- 元数据模型

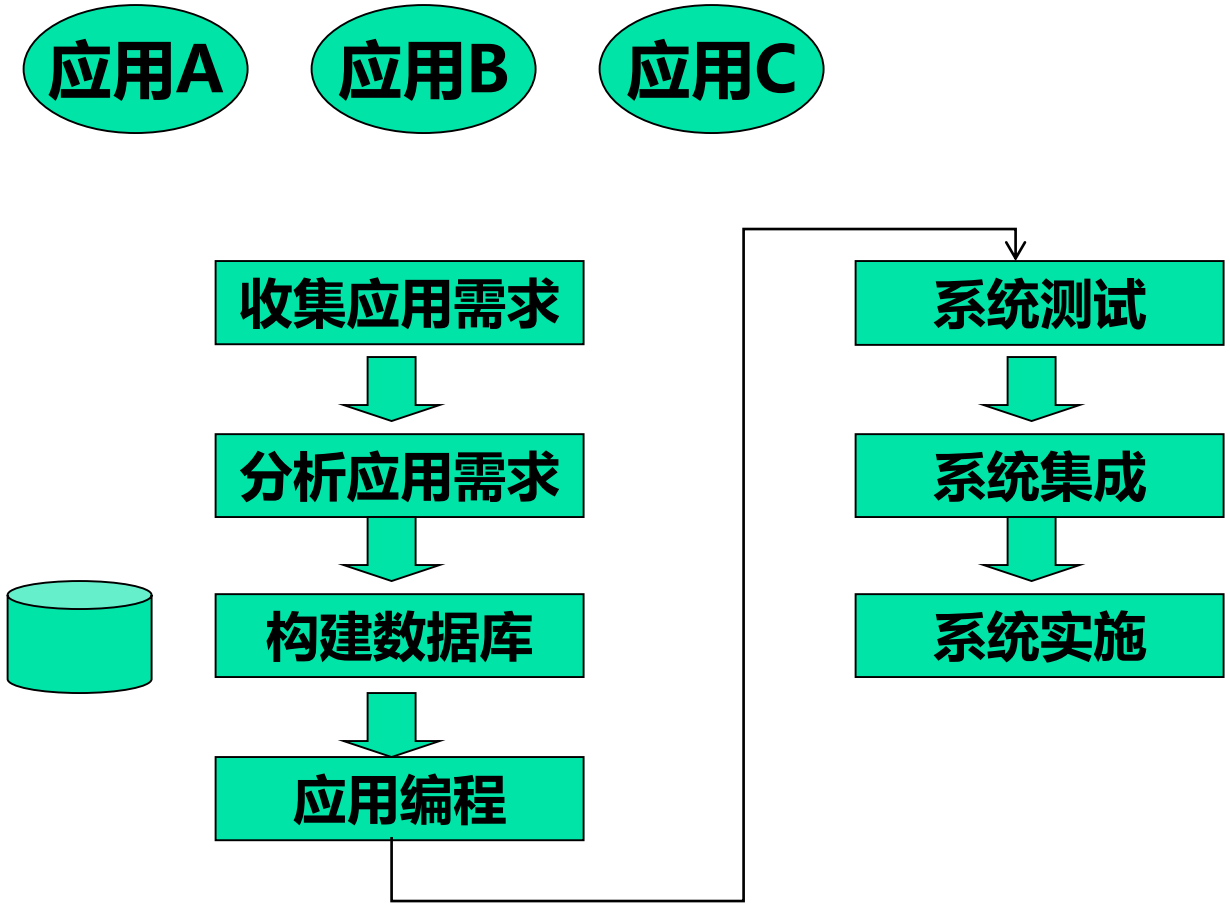
数据仓库与大数据平台设计对象

- 数据仓库与大数据平台的设计主要分成两个部分:
  - 数据模型设计
  - 功能模型设计

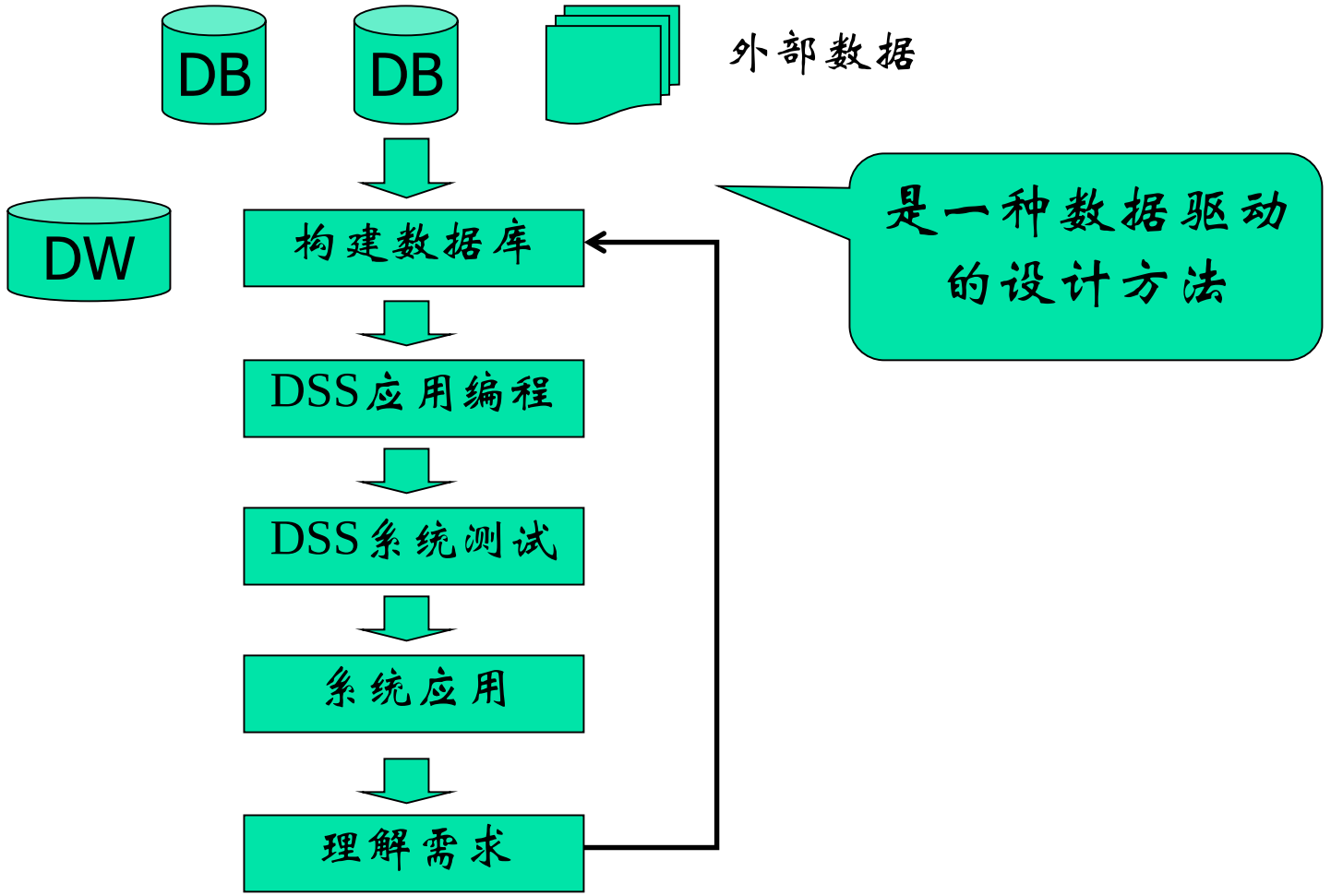


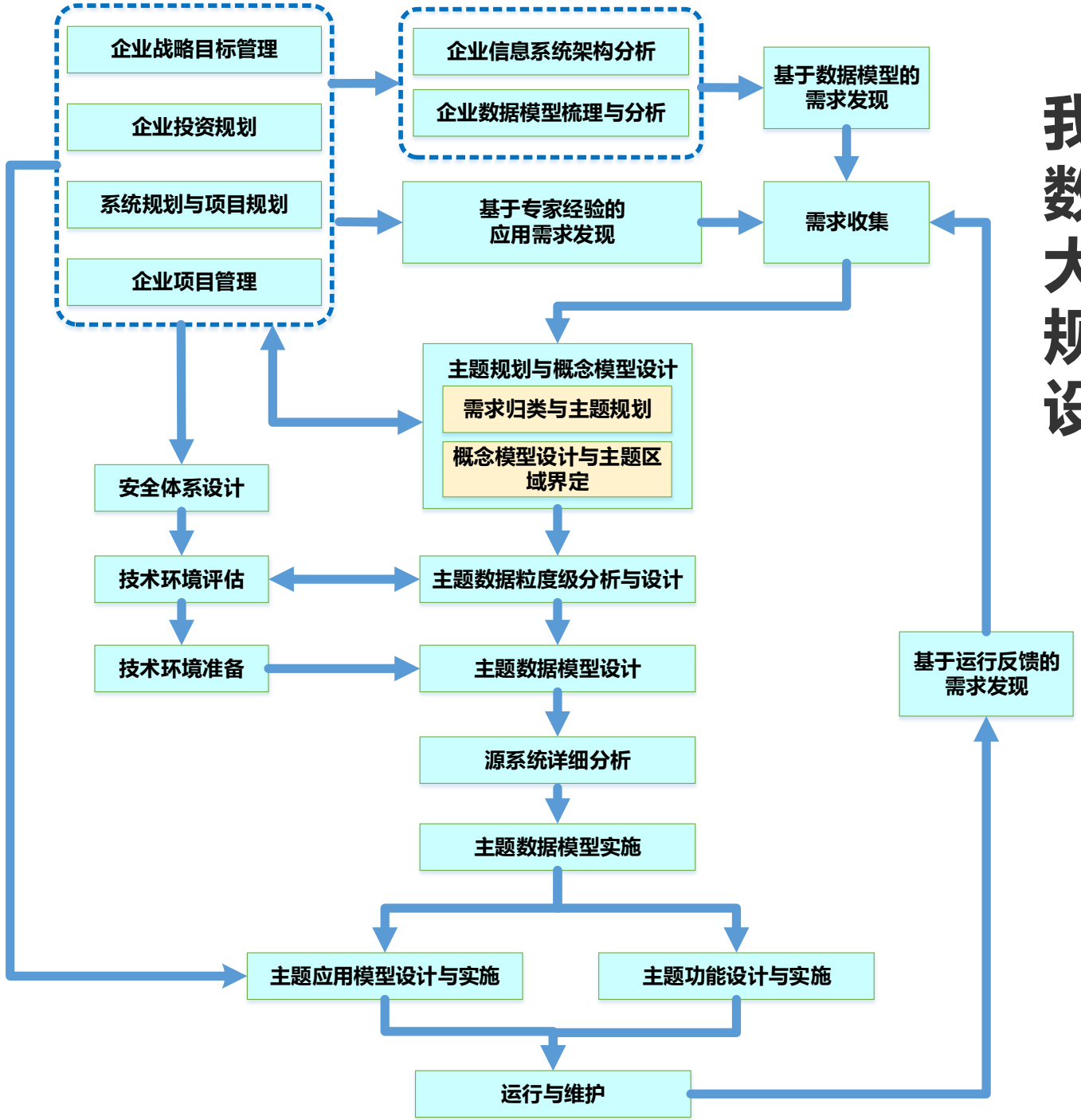

项目规划
规划与分析
数据组织与存储
数据访问与处理
应用 程序
需求分析
性能/存储/安全需求
数据项分析
数据流与事务分析
需求分析
设计
概念设计
逻辑设计
物理设计
概念模型设计
系统概要设计(I)-总体设计
逻辑结构设计
事务概要设计
概要设计(II)
物理结构设计
事务详细设计
详细设计
构造原型(可选)
系统实现
数据转换与加载
系统测试、部署与交付
系统实现和部署
运行维护
系统运行维护
什么系统的设计流程?
DBAS：数据库应用系统

瀑布式系统开发生命周期



数据仓库系统开发生命周期





我们总结的数据仓库或大数据平台规划设计建设总体过程

数据驱动设计方法的基本思路

- 1. 如果以前进行过数据库系统的建设，充分利用现有工作成果来进行系统的建设
  - 需要清楚地知道原有的数据库系统中已经有了什么，对当前系统的设计有什么影响，等等
  - 尽可能利用已有的数据、代码等，而**不是什么都从头开始**。
- 2. 不再是面向业务应用，而是组织分析型主题所题的数据，间接服务于业务应用
- 3. 核心是利用数据模型，有效地识别原有的数据库的数据和数据仓库中主题的数据的共同性。
  - 什么是数据模型?

内容提纲

- 系统设计流程
- **数据模型基本概念**
- 数据模型设计
- 数据模型设计与性能优化
- 元数据模型
- 平台与数据模型设计过程

基本概念

- 基本概念及其相关知识
  - 什么是模型?
  - 数据模型和相关概念
  - 数据模型的地位
  - 数据模型与信息系统中的共性

什么是模型?

- 用于**指导最终产品制造**的初步作品或结构
  - A preliminary work or construction that serves as a plan from which a final product is to be made.
- 用于**测试和完善**最终产品作品或结构
  - Such a work or construction used in testing or perfecting a final product
- 一种系统、理论或现象的**模式描述**，用于说明其已知或推导出的属性，并用于研究其进一步的特性
  - A schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics
- 用于**模仿或比较**的示范性物品或事物
  - One serving as an example to be imitated or compared.

数据模型和相关概念

数据模型

- 用于指导数据的设计、处理与存储和使用，并作为数据设计结果的检验标准的模型。
- an abstract model that organizes elements of data and standardizes how they relate to one another and to properties of the real world entities.
- A data model explicitly determines the structure of data.
- The main aim of data models is to support the development of information systems by providing the definition and format of data.

数据模型相关概念

- 数据模型的层次
  - 概念、逻辑、物理
  - 高层、中间层、低层
- 数据模型的表示
  - Entity-Relation-Diagram
  - Data structure diagram
    - A data structure diagram (DSD) is a diagram and data model used to describe conceptual data models by providing graphical notations which document entities and their relationships, and the constraints that bind them.
  - 逻辑数据模型表示
    - 关系型、网络模型、多维模型
  - 物理数据模型表示
    - 内外存数据结构

Data model topics

- Data architecture: 数据架构
  - Data architecture describes how data is processed, stored, and utilized in a given system.
  - It provides criteria for data processing operations that make it possible to design data flows and also control the flow of data in the system.
  - 总体数据架构，总体数据流图
- Data modeling：数据建模
  - the process of creating a data model by applying formal data model descriptions using data modeling techniques.

Data model topics

- Data organization
  - Another kind of data model describes how to organize data using a database management system or other data management technology.
- Data structure
  - A data structure is a way of storing data in a computer so that it can be used efficiently.

Data model topics

- Data flow diagram
  - a graphical representation of the "flow" of data through an information system
  - A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored.
  - It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model.

Data model topics

- Database Model
  - Hierarchical model
  - Network model
  - Relational model
  - Object-relation model
  - Star schema
- …

数据模型相关概念(续)

- 数据模型的通用性
  - 业务主题数据模型
    - 人和组织，订单，发票，装运，产品订购，产品…
  - 行业数据模型:
    - 电信、医疗、旅游、电子商务、制造业、保险、金融…
- 提升数据模型的通用性有什么用处
  - 数据模型的通用性决定的功能模型的通用性
  - 提升系统或产品的适用性

注意学习提升数据模型的抽象能力，提升数据模型的通用性

业务主题：
当事人模型

把人与组织抽象成同一个实体类型

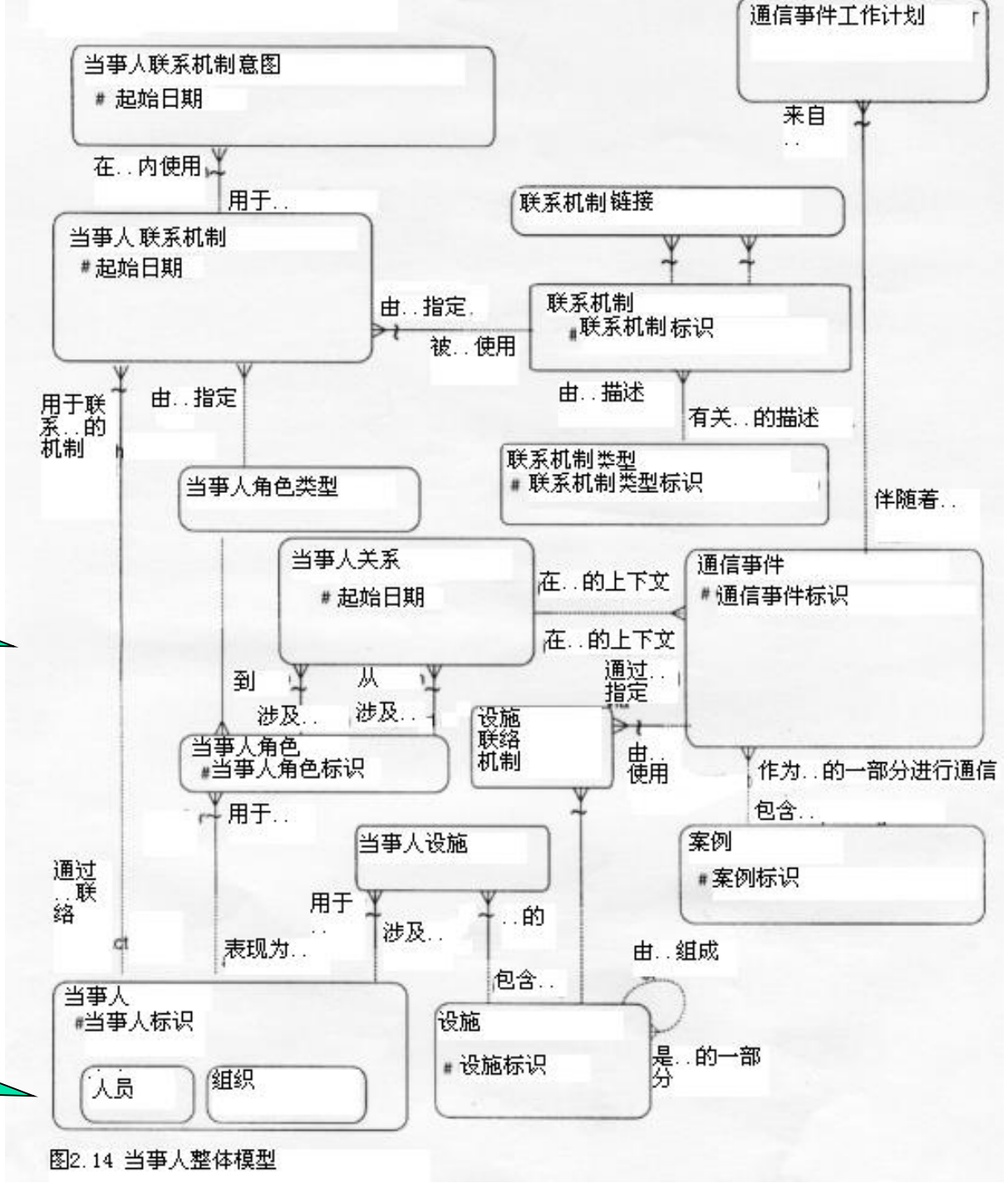
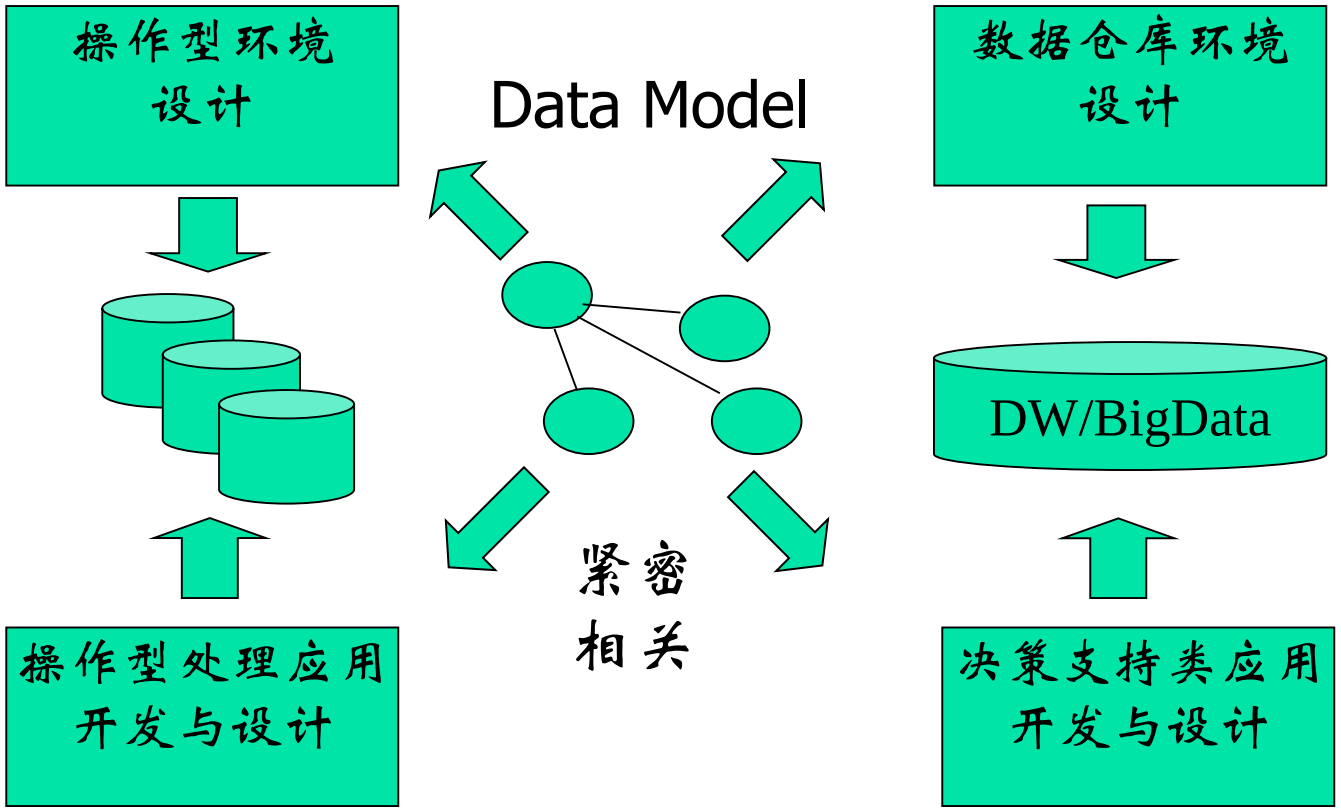


图2.14 当事人整体模型

数据模型相关概念(续)

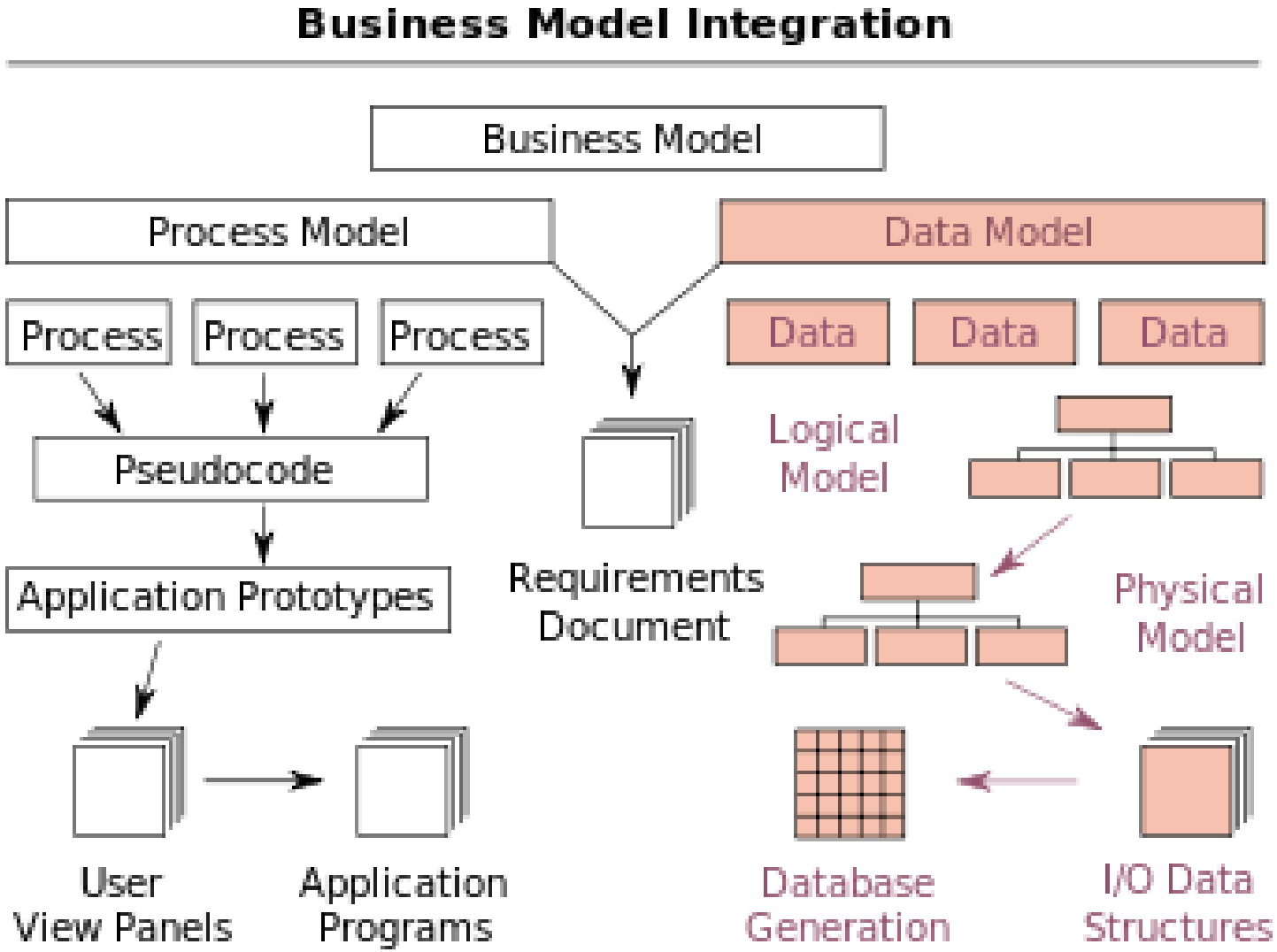
- 专用数据模型
  - 只适用于某个具体场景的数据模型
  - 在某个层面的通用数据模型基础之上，针对具体应用场景进行适当改造以后，适用了更窄的范围的数据模型

数据模型的地位



数据模型在整个系统架构中具有重要意义

业务模型的组成：数据模型+功能模型



利用数据模型可以识别两种共性

- 处理的共同性
  - 对于利用现有的工作成果是相当的重要
  - 便于建立例行化公共处理模块
  - 但是处理的变化比数据结构的变化要快得多
- 数据的共同性
  - 数据具有更大的稳定性
  - 可以根据应用，分出公用数据与独占数据
  - 通过识别数据的共同性，得到处理的共同性

信息系统环境中的模型

- 两类模型
  - 数据模型(Data models)
  - 处理模型(Process models)
- 什么是处理?
  - 在线事务处理(Online Transaction Process)
  - 在线分析处理(Online Analytical Process)
  - 其他的处理(Other Processes ……)

处理模型的组件分解

- 功能分解(Functional decomposition)
- 实体关系图(Entity-Relation Diagram)
- 数据流图(Data flow diagram)
- 结构图(Structure chart)
- 状态转移图(State transition diagram)
- HIPO 图
  - Hierarchy plus input, process, output
- 伪代码(Pseudocode)

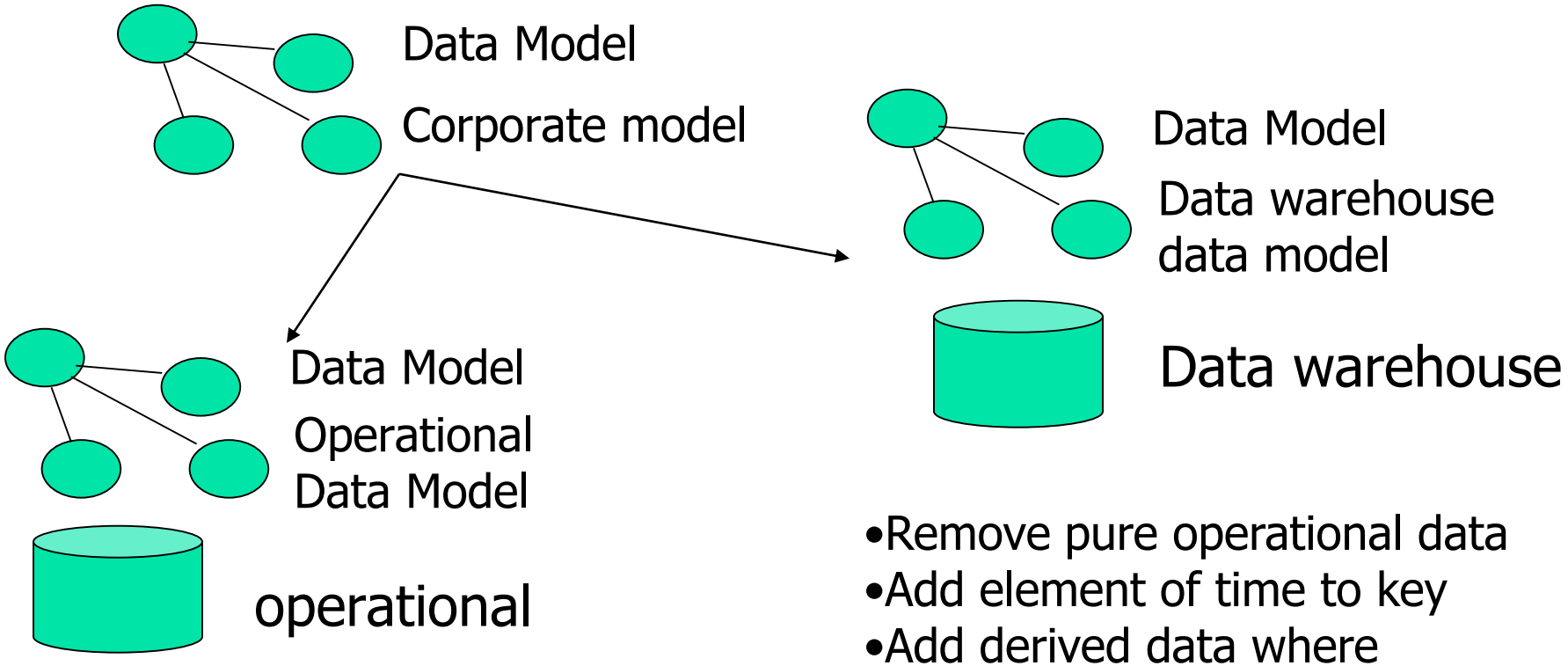
处理模型的适用性

- 对于构建以下系统很有价值
  - Data Mart
  - OLTP system
  - other common softwares
- 不适用于
  - Data warehouse
- 原因:
  - 处理模型假设已经存在一套处理的功能需求。

Corporate Data Model（企业数据模型）

- 数据模型适用于
  - The existing system environment
  - The data warehouse environment
- 整个架构中需要不同的数据模型
  - Corporate data model
  - Operational data model
  - Data warehouse data model
- Corporate data model
  - Focuses on and represents only primitive data.
  - Is the beginning point of the other two models.

不同层次的数据模型间的关联



- Operational data model equals corporate data model
- Performance factors are added prior to database design

- Remove pure operational data
- Add element of time to key
- Add derived data where appropriate
- Create artifacts of relationships
- ……

内容提纲

- 系统设计流程
- 数据模型基本概念
- **数据模型设计**
- 数据模型设计与性能优化
- 元数据模型

本小节内容

- 数据模型设计目标
- 概念模型设计
- 逻辑模型设计
  - 逻辑模型与模式分解
  - 关系型模型
  - 网络数据模型
  - 时间序列数据模型
- 物理模型设计
- 应用接口模型

数据模型设计准则

- 数据模型分层
  - 概念模型—高层模型
  - 逻辑模型—中间层模型
  - 物理模型—底层模型
- 数据模型应满足几个要求:
  - 真实地模拟现实世界
  - 范围恰当
  - 容易理解
  - 便于计算机上实现
  - 使用效率

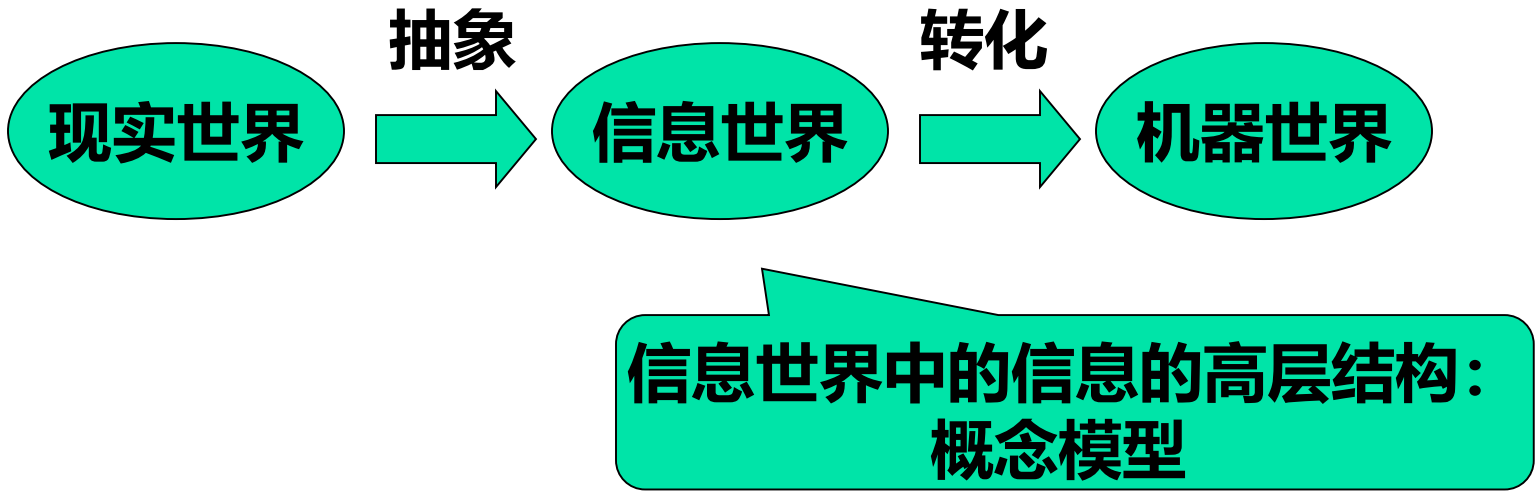
数据模型相关的三个世界或领域

(1)**现实**世界：存在于人脑之外的**客观世界**

(2)**信息**世界：人们通过**头脑**对现实世界进行**抽象**，并通过**描述工具**以实体、实体集、属性、实体标识符等形式**尽可能真实准确**地描述现实世界。

(3)**机器**世界：信息世界的信息在机器世界中的表示。以**数据形式**存储字段、记录、文件、关键码

1. 概念模型设计



- 概念模型是客观世界到机器世界的一个中间层次
- Conceptual data model
  - 也被称为Conceptual schema或Domain model

Establish the basic concepts and semantics of a given domain and help to communicate these to a wide audience of stakeholders.

概念模型

- 概念模型的功能
  - 在信息世界里对现实世界进行建模
  - 表达现实世界中的概念
  - 表达概念之间的关系
  - 表达业务领域的语义知识
- 对概念模型的要求
  - 具有很强的表达能力--能力强
  - 易于理解—容易懂
  - 简单—容易画

概念模型的定义及特点

为正确直观地反映客观**事物**及其**联系**，对所研究的信息世界，按用户观点对**数据**和**信息**建模型，称之为**概念模型**。(用于数据库设计)

特点：

（1）是独立于计算机系统的模型，完全不涉及信息在系统中的表示

（2）用于建立信息世界的数据模型，是现实世界的第一层抽象（是现实到机器的中间层）

（3）强调语义表达功能，概念简单、清晰，易于用户理解，是**用户**和**设计人员**之间交流的语言，是设计人员进行数据库设计的工具

信息世界的术语—续

- 实体集—entity set
  - 性质相同的同类实体的集合
- 联系—relationship，关系
  - 分成两类：实体内部的联系、实体之间的联系
- 实体内部的联系
  - 实体各属性之间的联系
- 实体之间的联系
  - 通常指不同实体集之间的联系

机器世界的术语

- 记录—record
  - 字段的有序集合(实体)
- 文件—file
  - 同一类记录或不同类记录集合(实体集)
- 关键码，主键，主码—key
  - 能唯一标识文件中每条记录的字段或字段集（实体标识符）
- 字段—field
  - 标记实体属性的命名单位，亦称数据项(属性)

概念模型的表示

- 概念模型的表示方法
  - ERD，Entity Relationship Diagram
    - Used to describe entities and the relationships between entities.
  - ERD 在数据库概念设计中被广泛使用
  - 为了和原有数据库的概念模型相一致，一般也常采用ERD描述数据仓库和大数据平台的概念模型。
- 良好的可操作性、形式简单
- 易于理解、便于与用户交流
- 对客观世界的描述能力较强

常用的概念模型—实体联系模型

Entity relationship model，简称ER模型，是由美籍华人陈平山于1976 年提出的。ER图提供了表示**实体、属性和联系**的方法。

(1) ER模型的三要素（三个基本语义）

A. 实体 (Entity)：表示客观事物。

B. 属性 (Attributes)：

表示客观事物的特征（属性）

C. 联系 (Relationships)：

客观事物之间的联系

ER Model

- An entity–relationship model is usually the result of **systematic analysis** to **define** and **describe** what is important to processes in **an area of a business**. It does not define the **business processes**; it only presents a **business data schema** in **graphical form**. It is usually drawn in a **graphical form** as boxes (*entities*) that are connected by lines (*relationships*) which express the associations and dependencies between entities.
- An ER model can also be expressed in a **verbal form**, for example: *one building may be divided into zero or more apartments, but one apartment can only be located in one building.*

刻划工具 –实体联系图(ER图)

ER图表示方法:

用**矩形**表示**实体**

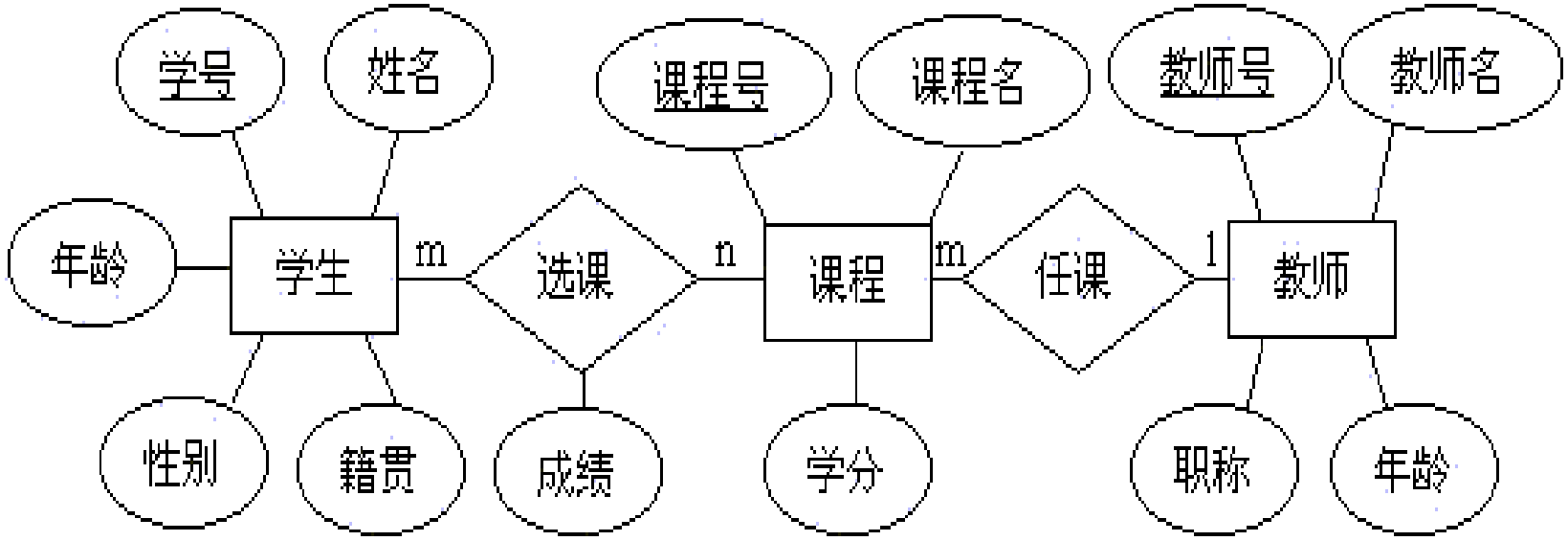
用**椭圆**表示**属性**

用**菱形**表示实体间的**联系**

属性和实体间、实体和联系间用**线段**连接(同时在线段边上标上联系的类型)

回忆数据库系统原理中的简单案例

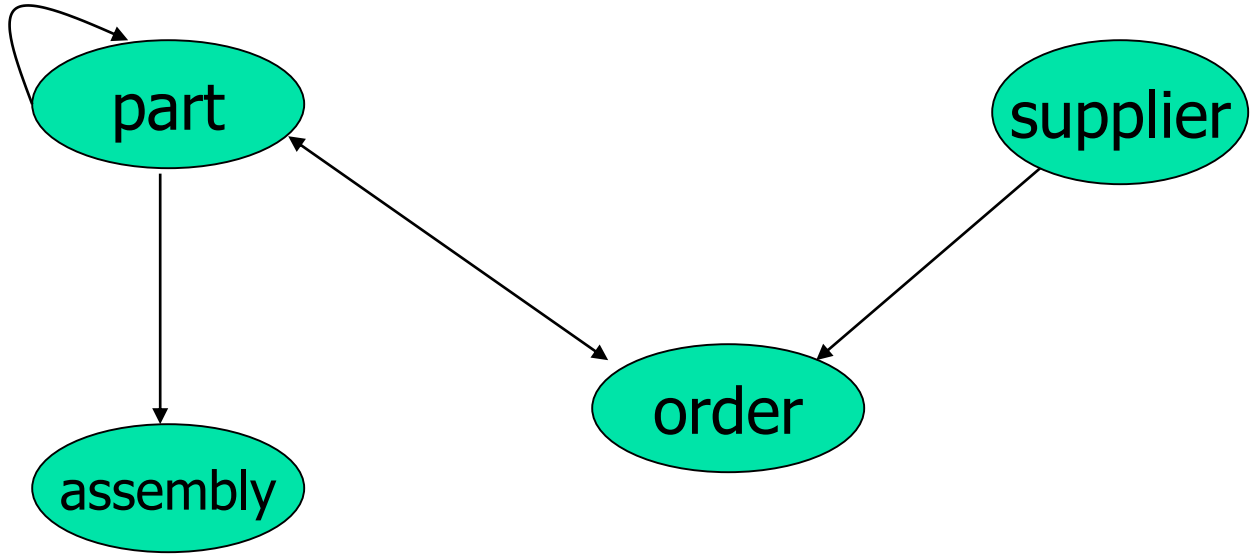
业务场景（business scenario）：假设一个学生可选多门课程，而一门课程又有多个学生选修，一个教师可讲多门课程，一门课程至多只有一个教师讲授。



扩展E-R模型

- ISA 联系—表达子类父类关系
  - 分类属性
  - 不相关约束
  - 可重叠约束
  - 完备性约束
- 基数约束
- Part-of 联系—表达组成关系

其他ER模型表示方法示例

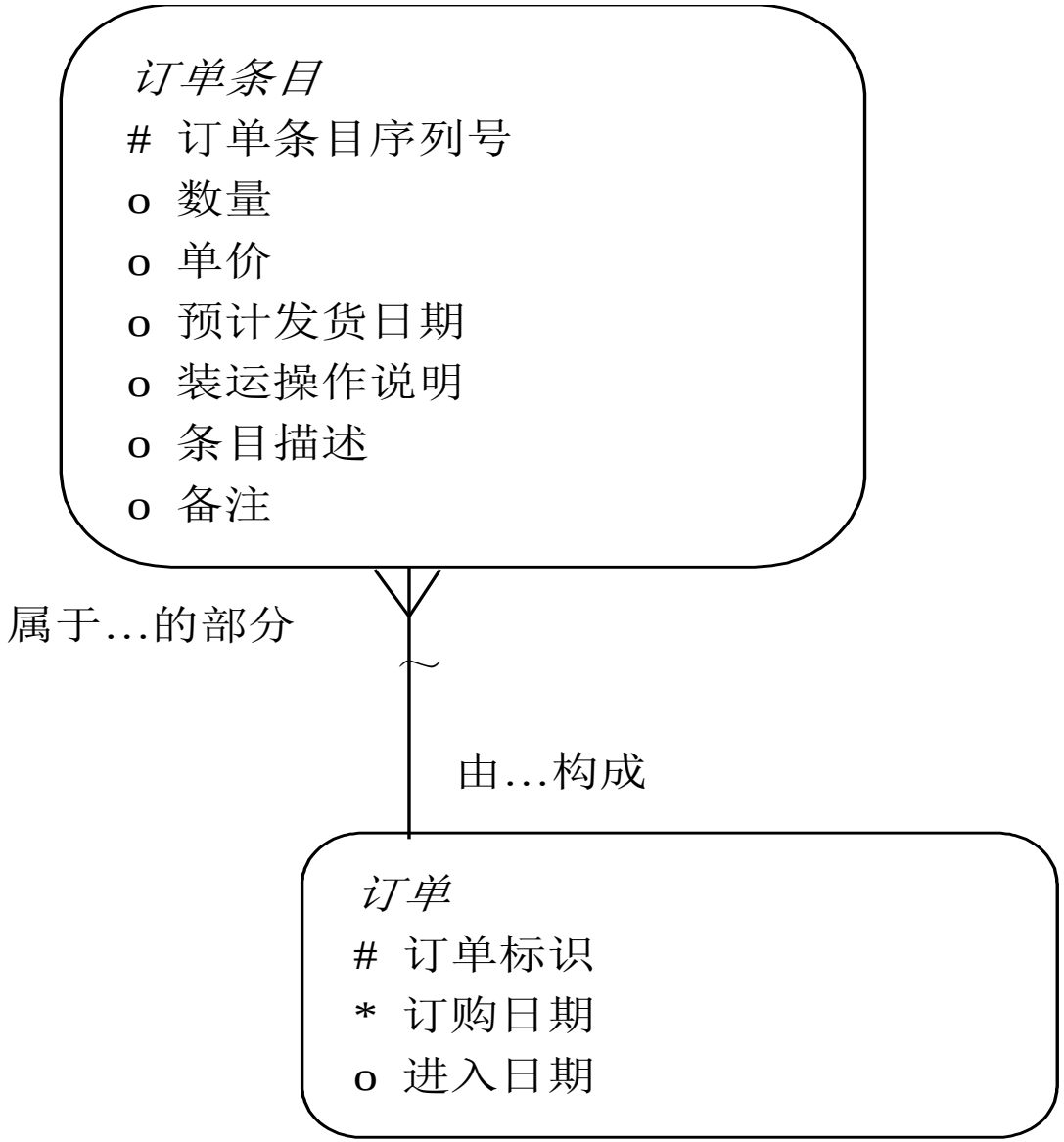


1:n relationship

1:1 relationship

m:n relationship

Entity, or major subject



ERD 表示方法，一对多关系

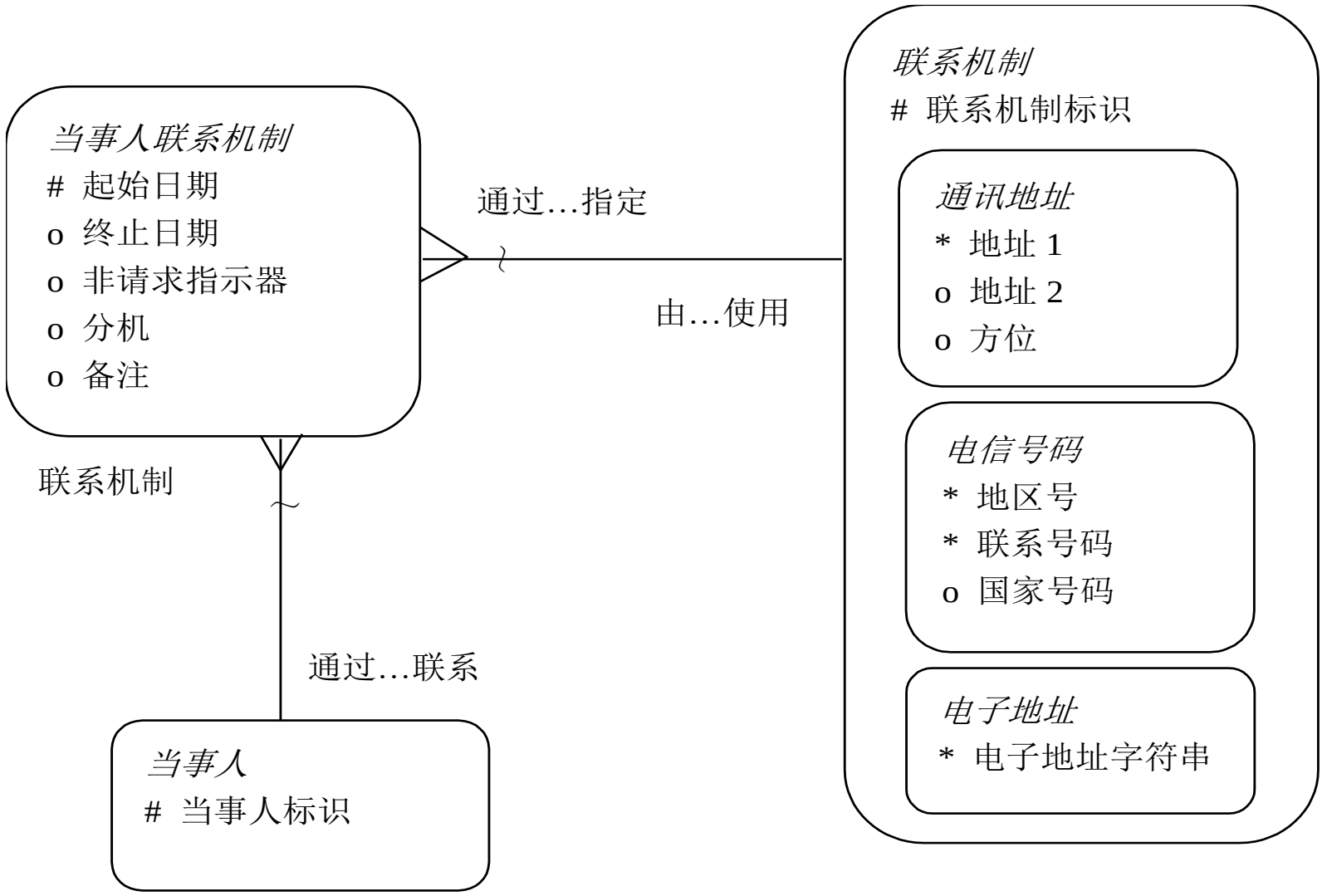

当事人联系机制
起始日期
o 终止日期
o 非请求指示器
o 分机
o 备注
通过...指定
由...使用
联系机制
通过...联系
当事人
当事人标识
联系机制
联系机制标识
通讯地址
* 地址 1
o 地址 2
o 方位
电信号码
* 地区号
* 联系号码
o 国家号码
电子地址
* 电子地址字符串

图 1.7 多对多关系

概念模型设计最重要的几个因素

- 概念模型的正确性评估
  - 是否正确地反映业务场景
  - 能否准确描述业务需求中需要记录的实体、属性及其关系
- 重要概念：**数据模型范围—集成的范围**
- Scope of Integration
- 如何确保正确与恰当的集成范围

Scope of integration 集成范围

- 集成范围
  - 哪些实体属于模型的范围
  - 哪些实体不属于模型的范围
- 集成范围定义了数据模型的范围，即企业里哪些业务相关的数据**需要进入到数据仓库或大数据平台中**。
- 这个范围必须在建模过程开始以前开展。

相关概念　SOW，scope of work

信息化范围

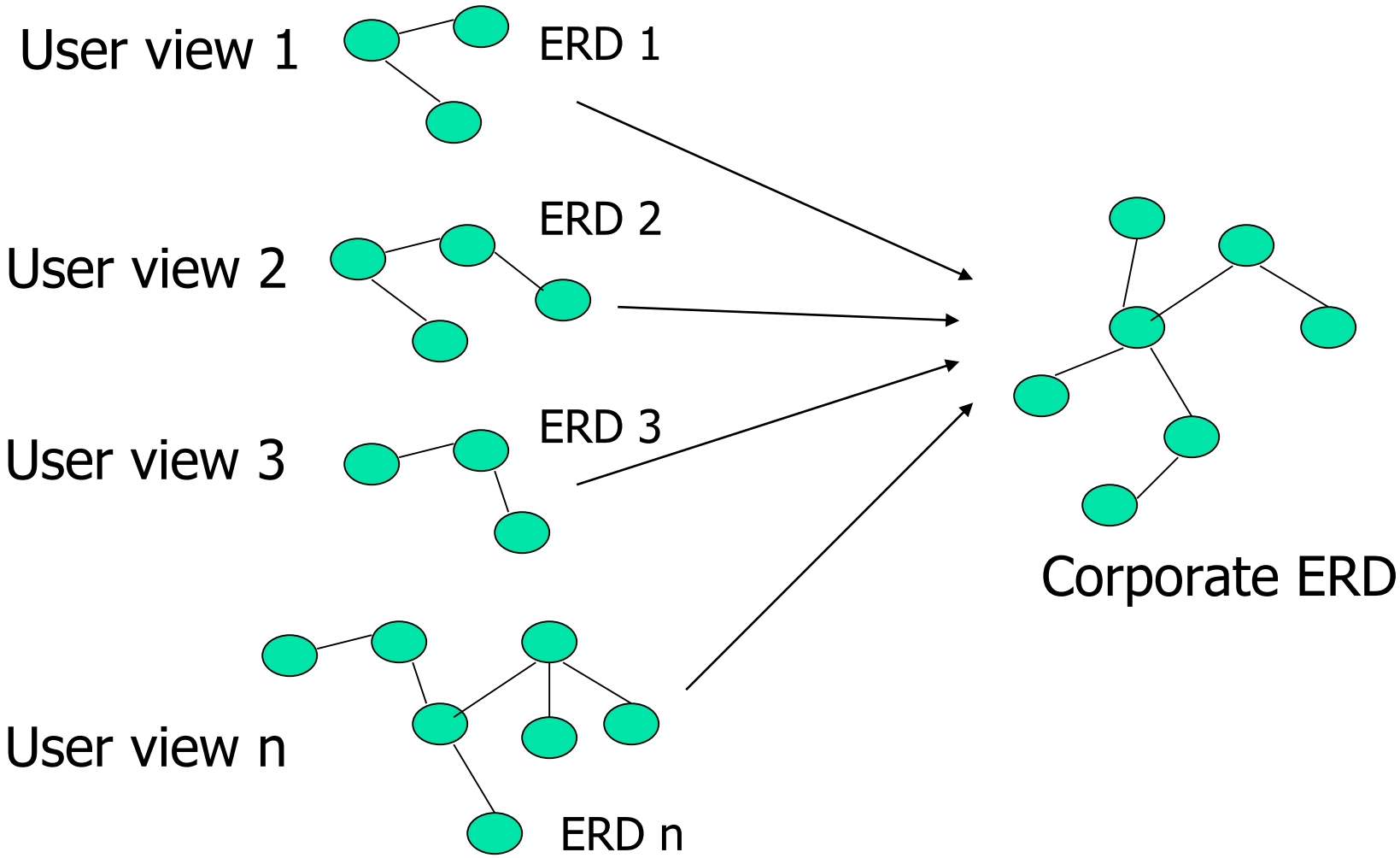
集成范围

- 集成范围必须得到以下人员的认可
  - Modeler—建模人员
  - Management—管理层
  - Ultimate user—最终用户
- 如果集成范围没有确定，建模过程会永远持续下去，无法结束。
- 问题：
  - 为什么集成的范围需要得到这些人员的认可?

集成范围为什么需要得到这些人员的认可

- **正确性**—确保数据模型是对的
  - 正确性需要用户确认，建模人员理解
- **有用性**—领导与用户对有用性的理解
- **充分性**—每人的观点和能看到范围不同
- **经费问题**：是否有钱投入、是否足够
- **可实现性**
  - 建模人员，用户共同分析
- 问题：如何得到准确充分的视图

不同人对企业ERD的掌握程度不同



重建或设计企业ERD的方法

- 从业务背景与基础数据模型出发
  - 尽可能掌握**企业全局业务背景**
  - **业务专家**从不同的业务部门收集企业ERD
  - 跟不同的人员**聊天**掌握他们对企业ERD的认识
- 集成并重构ERD，将ERD读通，准确反应业务背景
- 研讨圈定数据仓库所涉及的ERD边界
- 将企业ERD划分成多个主题ERD模型

2. Logical Model 逻辑模型

- What's Logic?
  - The **relationship** between **elements** and between an **element** and the **whole** in a set of objects, individuals, principles, or events
- Logical Data Model
  - A logical design of a data

有关逻辑模型的一些说法

- A sound logical design should streamline the physical design process by clearly defining data structures and the relationships between them.
- Logical data model includes all required entities, attributes, key groups, and relationships that represent business information and define business rules.(这个说法体现为概念模型的细化)
- A logical data model provides all the information about the various entities and the relationships between the entities present in a database. (也是概念模型的细化)
- A logical data model represents the organization of a set of data by standardizing the people, places, things (entities) and the rules and relationships between them using a standard language and notation. It provides a conceptual abstract overview of the structure of the data. （体现数据的组织、体现对概念模型的标准化、体现对实际数据结构的抽象）

有关逻辑模型的一些说法

- Logical data modeling **does not** provide any information related to **how the structure is to be implemented** or the means (**technologies**) that are needed to implement the data structure shown. It is a **technology-independent model** of data that is developed from the initial structures identified by the conceptual model of data. Some of the information presented by a logical data model includes the following:
  - Entities
  - Attributes of entities
  - Key groups (primary keys, foreign keys)
  - **Relationships**
  - **Normalization**
- 体现对概念模型的细化，独立于物理结构

Enterprise Logical Data Model

- 企业逻辑数据模型
- The significance of enterprise logical data modeling is that it is a process-independent representation of business data.
- It also serves as a valuable data management tool and a bridge of understanding between the technical (Information Technology) and business branches of an organization. The value of logical data modeling has always been to establish the “single version of the truth” of an organization’s business functions. Or in other words, a single view of the business that is not adulterated by IT concepts and processes.
- 此处定义其实就是Corporate Data Model

逻辑数据模型定位与各类逻辑数据模型

- 定位
  - 描述数据仓库与大平台中的主题数据集的逻辑实现，即在概念模型的基础上，针对**选定的技术平台的类型**，对每个主题所对应的数据集进行进一步的细化分解与描述。
- 两大任务
  - 细化描述
  - 分解，模式分解
- 目的
  - 为后续的物理实现奠定基础

各类逻辑数据模型

- 关系模型
  - 关系型平台或类关系型平台
- 多维数据模型
  - 多维分析应用场景，多维分析引擎
- 网络数据模型
  - 针对复杂网络或图数据表示与存储问题
  - 图数据库
- 时间序列数据模型
  - 时间序列数据库
- …

任务1：细化描述

- 实体与属性命名
- 关键字
- 数据类型
- 取值范围
- 明确数据取值与业务语义间对应关系
- 数据关系具体化
- 完整性约束具体化

任务2：模式分解

- 结合物理平台的类型，进行合理的模式分解
- **Decomposition** in computer science, also known as factoring, is breaking a complex problem or system into parts that are easier to conceive, understand, program, and maintain.
- 分解目的
  - 确保数据易于管理
  - 确保数据易于操作
    - 增、删、改、查
  - 减少数据冗余
  - …

模式分解常见原则

- 根据**规范化方法**进行分解
  - Normalization，Denormalization
- 根据**独立性原则**进行分解
  - 易于管理、易于操作访问
- 根据**数据变化速度**进行分解
  - 数据变化速度快、中等、慢，将一个实体集相关属性归入不的组
- 根据**分布式与并行处理需求**进行分解
- 根据**数据保密与安全要求**进行分解

3. 逻辑模型与关系模型

- 最常见的逻辑模型的表示方法
- 关系模型
  - 传统的多数数据仓库都是以关系型数据库为基础
  - 大数据平台中HIVE也是一种变种的关系型模型
- 具有标准数据操作语言支撑，透明性好
- HIVE的出现也是因为HADOOP架构下数据管理访问的透明性需求而出现的

相关概念

- Relation
  - 关系，一个二维表，a table
- tuple, a line (record) of a table
  - 元组，记录，表中的一行
- attribute, a column
  - 属性，属性列, attribute name 属性名
- key, attribute set, a value identifies a tuple
  - 关键字，主键，主码，键码

相关概念

- domain, range of an attribute
  - 域，属性域，值域，定义域，取值范围
- component, an attribute set of a tuple
  - 分量
- schema, a description of a relation
  - *Relation_Name*(a_1,…,a_n)
  - 模式

4. Physical Model 物理模型

- 根据逻辑模型建立起来
  - 对逻辑进行扩展，引入和考虑模型的物理特性。
  - 考虑性能因素
- 逻辑模型在物理数据库上的实现
- 在关系型数据库中，物理模型看起来象是一系列的表，relational tables.

包括

- Physical access model
  - 物理存取方式
- Storage structure of data
  - 数据存储结构
- Data placement
  - 数据存放位置
- Storage allocation
  - 存储分配

需要考虑的因素

- I/O access
  - 存取时间
- Space usage
  - 空间利用率
- Maintenance cost
  - 维护代价

可以采用的技术

- Introduction of redundant data
- Partitioning
- Deriving data
- Merging tables
- Further separation of data
- Creative index

5. 另外一种三层数据模型

- The High Level Data Model
  - ERD
- The Middle Level Data Model
  - DIS，data item set, 由高层模型扩展而来。
- The Physical Data Model
  - 从中间层模型生成。
  - Tables and physical design

Data item set

- ERD中的每个实体都与一个DIS相对应
- 每个DIS中的数据项分为四个组别
  - A primary grouping of data, 基本数据组
  - A secondary grouping of data，二级数据组
  - “Type of” data 类型数据组
  - A connector, signifying the relationships of data between major subject areas
    - 联接属性组

Four basic constructs

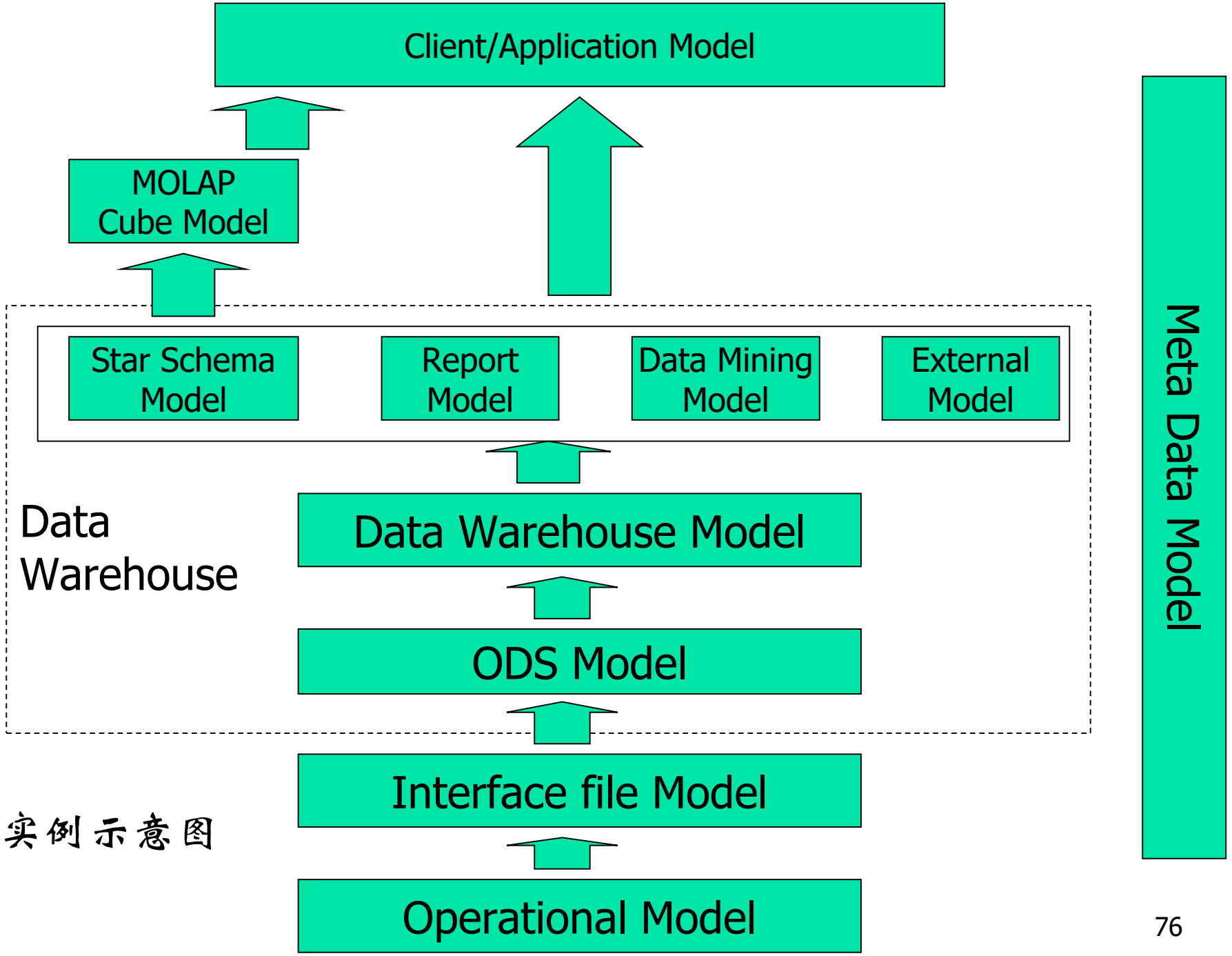
- Connector
  - 主要用于表示本主题域与其他主题域之间的联系，体现实体间的关系。
- 其余三类数据组的划分标准是基于不同程度的数据稳定性。
  - Seldom change:基本数据组
  - Sometimes change:二级数据组
  - Frequently change:类型数据组

分组好处

- 结构清晰，具有相似属性的数据被组织在一起；
- 减少冗余，如果将低数据混杂在高频数据中一起存储，将产生大量冗余。
- 例子，Customer
  - ID, 姓名，性别
  - 住址，文化程度，电话
  - 购物记录

6. 数据模型的另一种层次性

- 不同层次数据所对应的不同层次的数据模型
- 常见的层次：
  - 操作型环境中数据模型
  - 数据集成所需的缓冲层或接口层数据模型
  - ODS层数据模型
  - 核心数据仓库数据模型
  - 与应用有关的应用服务层数据模型
  - 元数据模型

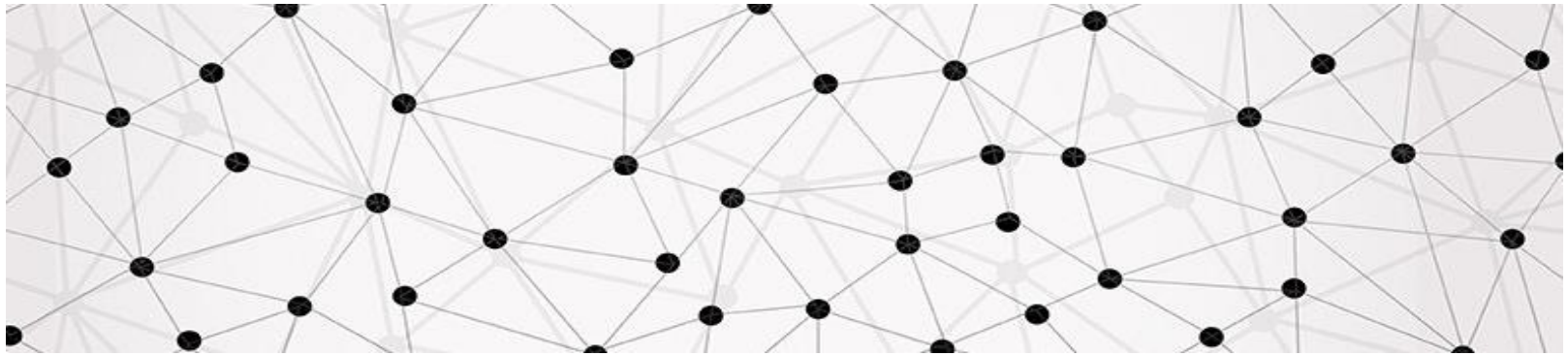

Client/Application Model
MOLAP
Cube Model
Star Schema
Model
Report
Model
Data Mining
Model
External
Model
Data
Warehouse
Data Warehouse Model
ODS Model
Interface file Model
Operational Model
Meta Data Model

实例示意图

网络（图）数据模型及平台

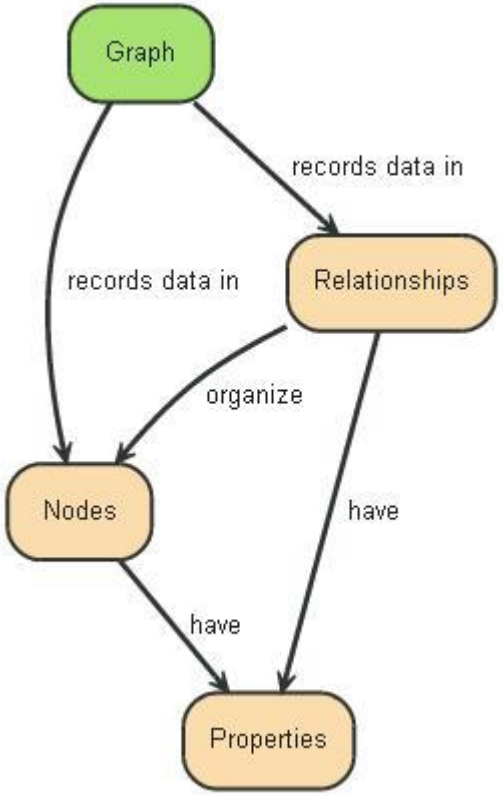
网络数据模型

- 网络数据模型也称**图形数据模型**，应用图形理论表示实体之间的关系信息。
- 最常见例子
  - 社会网络中人与人之间的关系形成社交网络
  - 计算机网络是计算机与计算机之间的关系形成的网络
  - 蛋白质相互作用网络是蛋白质之间的关系网络
  - …



网络/图由节点和边组成

- 从某种意义上讲，图就是二元关系。它利用线（称为边）或箭头（称为弧）以及连接的点（称为节点）可以形成图或网络，节点和边上可以有属性



网络数据模型的表示和分类

- 网络或图数据模型通常可以表示为：$G(V, E)$，其中V为节点的集合，E为边的集合
- 任意一条边$e \in E$可表示为(v_i, v_j)，即节点的二元组
- 多种网络或图数据模型：
  - 有向网络/无向网络
  - 有权网络/无权网络
  - 有符号网络/无符号网络
  - 异质网络/同质网络
  - 多层网络/单层网络
  - …

网络逻辑模型组成

- 节点
  - 节点属性集
- 节点集
- 边
  - 边属性集
- 边集
- 网络
  - 网络属性集
- 网络集合—图的集合

Relations and Actors

Detailed Relation facts

- Schema: $(n_i, n_j, \textit{DateTime}, a_1, a_2, \ldots, a_m)$
- *CallRecord(PhoneFrom, PhoneTo, DateTime, Duration, …)*
- *SendRecord(EmailSend, EmailRecv, DataTime, Title, MailID, …)*

Derived Relations

- Schema: $(n_i, n_j, m_1, m_2, \ldots, m_k)$
- *DateEdges(PhoneFrom, PhoneTo, Date, CallTimes, TotalDuration, …)*
- *MonthEdges(PhoneFrom, PhoneTo, Month, CallTimes, TotalDuration, …)*

Actors or Individuals

- Schema: $(ID, a_1, a_2, \ldots, a_n)$
- Users(PhoneID, IDCard, Name, Gender, PackageType, BuyTime, …)

Network or Graph

Nodes or Actors

- Schema:(ID, w_1, w_2, …, w_n)
- UserNodes(ID, *InDegree*, *OutDegree*, *ARPU*, *TimesPerWeek*, *DataUsagePerWeek*, …)

Network Edges

- Schema: （n_i, n_j, w_1, w_2, …, w_k）
- *Edges(PhoneFrom, PhoneTo, WorkingHours, Offhours, TimesPerWeek, WeekendTimes, …)*

DateTime

Networks

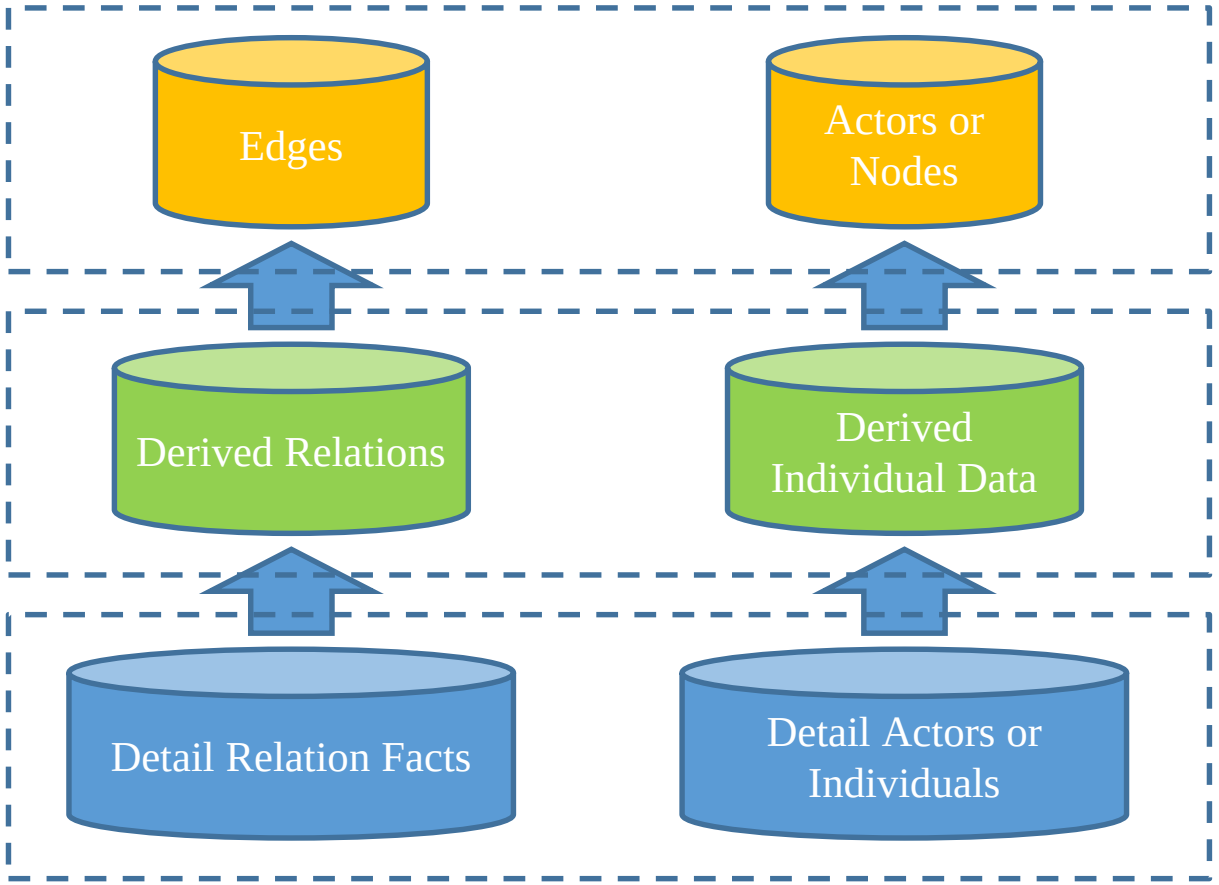
- **Graph = (N, E)**
  - N: a set of nodes, ($NodeID$, a_1, a_2, …, a_n)
  - E: a set of edges
- **Networks**
  - A set of network or graph

Considerations for Physical Models

- Database Platform
  - RDBMS or NoSQL
- Tradeoffs
  - Cost
    - Space
    - Maintenance
    - VMs
    - I/O
    - Communication
  - Performance
    - Query: online analysis
    - Application Algorithms

图数据适合用什么数据库存储

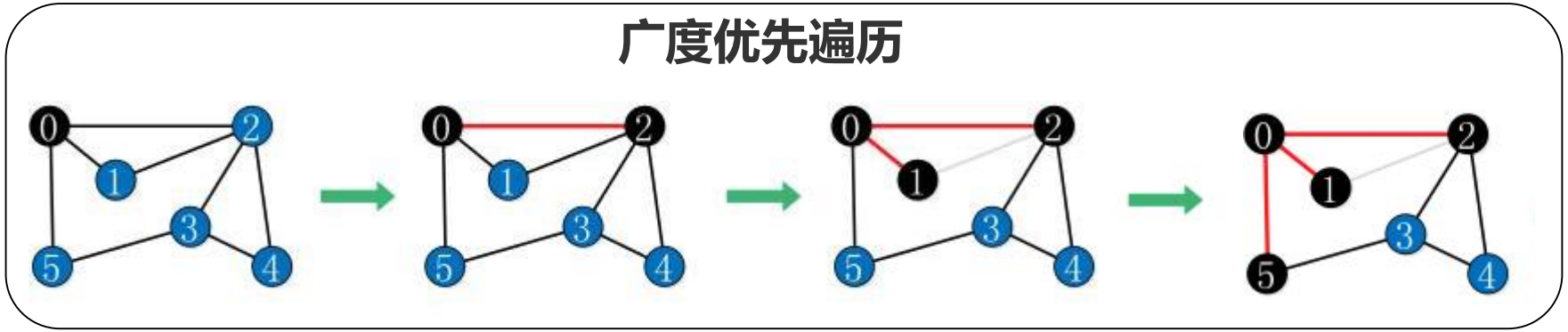
Data of Complex Networks

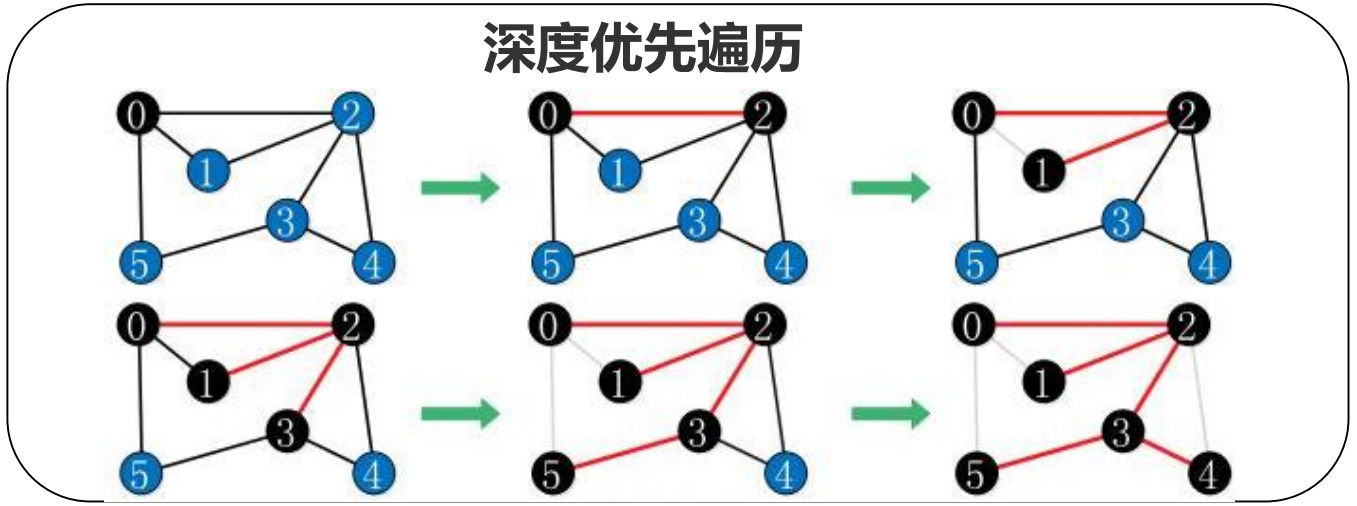


方法1—关系型数据库

- 可以实现图存储与计算，但是存在问题
- 例如，设有无向图的边集（$n_i, n_j, w_1, w_2, \ldots, w_k$），查询结点$a$出发所有的边，应该怎么查?
- 这个操作是图中从一个结点出发的单步广度遍历
- 查询n_i=a或n_j=a的边的集合，需要做两次查询，查完的以后再合并
- 或者对同一元组或做两个条件判断
- 如果涉及多步深度遍历的，查询操作非常复杂

为什么关系型数据库不太适用于网络数据





- 关系数据库在处理这两类操作时效率非常低下
- 关系数据库难以对网络数据进行划分从而进行分布式处理

网络/图数据库

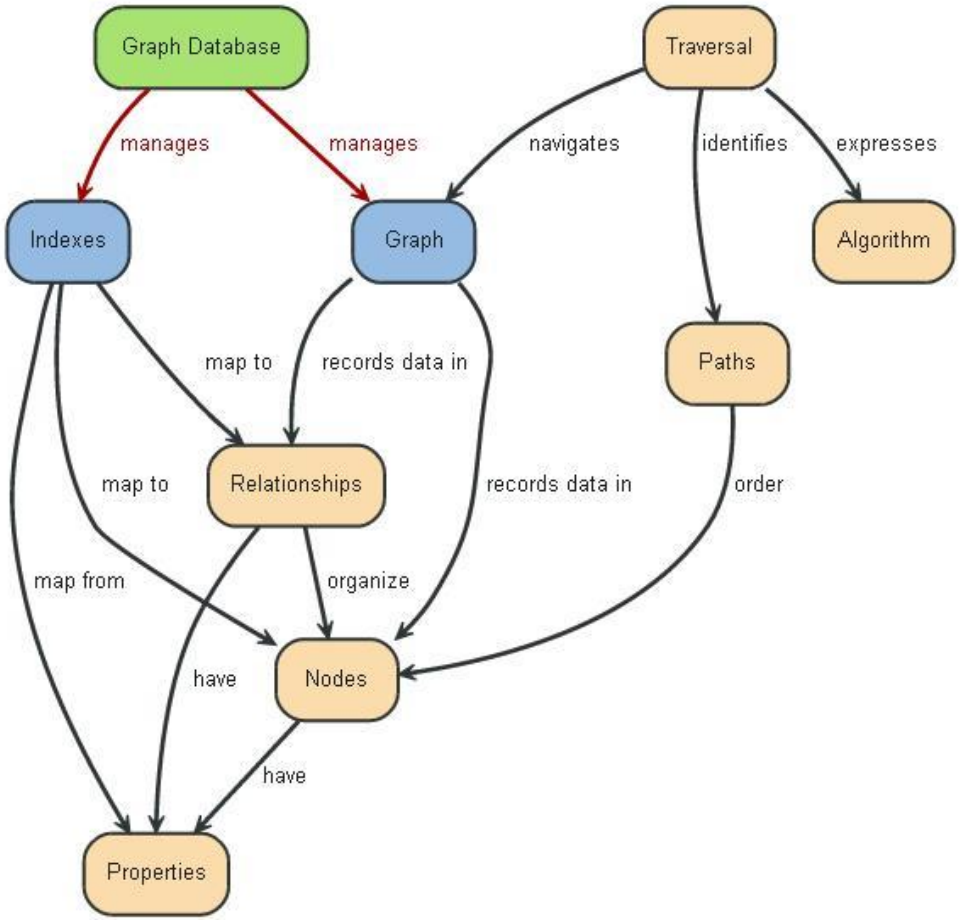
- 图形数据库（Graph Database）是四大非关系型（ NoSQL ）数据库之一
- NoSQL数据库的四大分类:
  - 键值存储数据库
    - 例如：Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB
  - 列存储数据库
    - 例如：Cassandra, HBase, Riak
  - 文档型数据库
    - 例如：CouchDB, MongoDb
  - 图形（网络）数据库
    - 例如：Neo4J, InfoGrid, Infinite Graph.

图数据库VS关系型数据库

- 图数据库与关系型数据库相比主要优点在于：
  - 数据量较大时，图数据库具有更快的数据库操作
  - 数据更直观，相应的SQL语句也更好写
  - 更灵活。不管有什么新的数据需要存储，都是一律的节点和边，只需要考虑节点属性和边属性

一种流行的图数据库——Neo4j

- 面向网络的数据库
- 高性能的图引擎
- 嵌入式、基于磁盘、具备事务特性的Java持久化引擎



Neo4j的核心概念

- （1） Nodes（节点）
- （2） Relationships（关系）
- （3） Properties（属性）
- 图的基本单位主要是节点和关系，他们都可以包含属性，一个节点就是一行数据，一个关系也是一行数据，里面的属性就是数据库里面的row里面的字段。

Neo4j的核心概念

- **（4） Labels（标签）**
  - 标签通过形容一种角色或者给节点加上一种类型，一个节点可以有多个类型，通过类型区分一类节点，这样在查询时候可以更加方便和高效，除此之外标签在给属性建立索引或者约束时候也会用到
- **（5） Traversal（遍历）**
- **（6） Paths（路径）**
  - 查询时候通常是遍历图谱然后找到路径，在遍历时通常会有一个开始节点，然后根据cpyher提供的查询语句，遍历相关路径上的节点和关系，从而得到最终的结果。

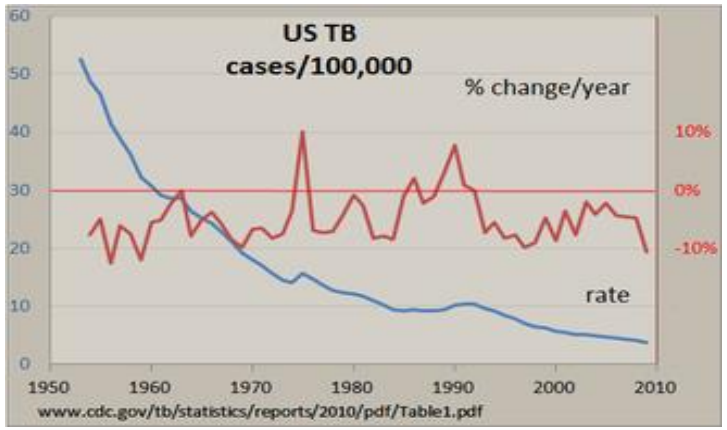
七种常见的图数据库对比

	Neo4j	Infinite Graph	DEX	InfoGrid	HyperGraphDB	Trinity	Allegro Graph
说明文档质量	好	好	一般	差	好	差	好
便携性如何	好	差	好	好	好	差	差
是否支持Java	是	是	是	是	是	否	是
是否免费	是	<1M	<1M	是	是	否	<50M
是否支持图属性	是	是	是	是	是	是	RDF
是否支持超图	否	否	否	否	是	是	否

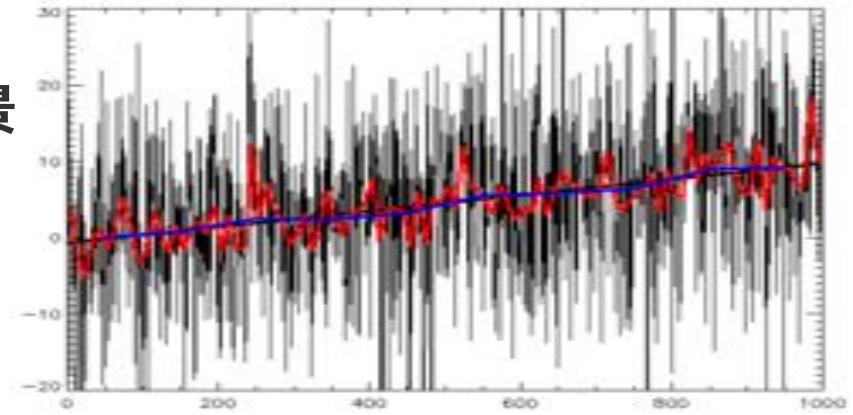
时间序列数据模型

为什么要有时间序列数据模型?

- 需要持续观测并记录观测对象的状态，加以利用
  - 气温、噪音、流量、文字流、语音流、…
- A time series **is a series of data points** indexed (or listed or graphed) **in time order**. Most commonly, a time series is a sequence taken at successive **equally spaced points** in time. Thus it is a sequence of **discrete-time data.**
- Time series data have a natural temporal ordering.



主要应用场景
预测
异常发现
原因分析



Motivation of time series analysis

- **Forecasting**
  - statistics, econometrics, quantitative finance, seismology, meteorology, and geophysics
- **signal detection and estimation**
  - signal processing, control engineering and communication engineering
- **clustering, classification, query by content, anomaly detection as well as forecasting**
  - data mining, pattern recognition and machine learning

时间序列数据模型

- 时间序列数据模型可以分为两部分，分别是series 和data points
- series
  - 唯一标识符
- data points
  - 点的数组，每个点由时间戳和值组成
  - 数据点有两种模型，一种是时间戳和值分别为一个数组，另一种是时间戳和值的一个数组，前者对应于列存储，后者对应行存储。
  - 在现有数据库基础之上构建TSDB的时候，大多使用行存储，专门构建的TSDB，大多使用列存储。

时间序列数据库

- 时间序列数据库
  - 就是存放时间序列的数据库，并且要支持时序数据的快速写入、持久化、多维度的聚合查询等基本功能
- 与传统数据库相比，传统数据库仅仅记录了数据的当前值
- 时序数据库则记录了所有的历史数据，同时时序数据的查询也总是会带上时间作为过滤条件。

时序数据库的一些基本概念

- metric
  - 度量，相当于关系型数据库中的table
- data point
  - 数据点，相当于关系型数据库中的row
- timestamp
  - 时间戳，代表数据点产生的时间
- field
  - 度量下的不同字段。
- tag
  - 标签，或者附加信息

时序数据库遇到的挑战

- 时序数据库是为了解决海量数据场景而设计的，因此，需要解决一下几个问题：
  - 时序数据的写入：如何支持每秒钟上千万上亿数据点的写入
  - 时序数据的读取：如何支持在秒级对上亿数据的分组聚合运算
  - 成本敏感：由海量数据存储带来的是成本问题。如何更低成本的存储这些数据，将成为时序数据库需要解决的重中之重。

时间序列数据库的类型

- 几乎任何数据库都可以存储时间序列数据，但是不同的数据库能支持的查询类型并不相同。按照能够支持的查询类型，时间序列数据库可以分为两类：
  - 第一类表结构为：[metric_name] [timestamp][value]
    - 优点：这种模式容易优化，可以做到非常快
    - 缺点：无法快速响应变化，存储膨胀
  - 第二类表结构为：[timestamp][d1][d2]..[dn][v1][v2]..[vn]
    - 优点：优化的查询方式不限于查询原始数据，而是可以组合查询条件并且做聚合计算。
  - 第一种结构仅仅可以提供原始数据的查询，而第二种结构不仅支持原始数据的查询，也支持对原始数据的聚合能力。

时序数据库的类型（续）

- 时序数据库按照底层技术的不同可以划分为三类：
  - 直接基于文件的简单存储：RRD Tool，Graphite Whisper。这类工具附属于监控告警工具，底层没有一个正规的数据库引擎。只是简单的有一个二进制的文件结构。
  - 基于K/V数据库构建：opentsdb（基于hbase），blueflood，kairosDB（基于cassandra），influxdb，prometheus（基于leveldb）
  - 基于关系型数据库构建：mysql，postgresql 都可以用来保存时间序列数据

时间序列数据库发展里程碑

- 1999/07/16 RRD Tool First release
- 2009/12/30 Graphite 0.9.5
- 2011/12/23 OpenTSDB 1.0.0
- 2013/05/24 KairosDB 1.0.0-beta
- 2013/10/24 InfluxDB 0.0.1
- 2014/08/25 Heroic 0.3.0
- 2017/03/27 TimescaleDB 0.0.1-beta

HDFS、HBase、Hive简介

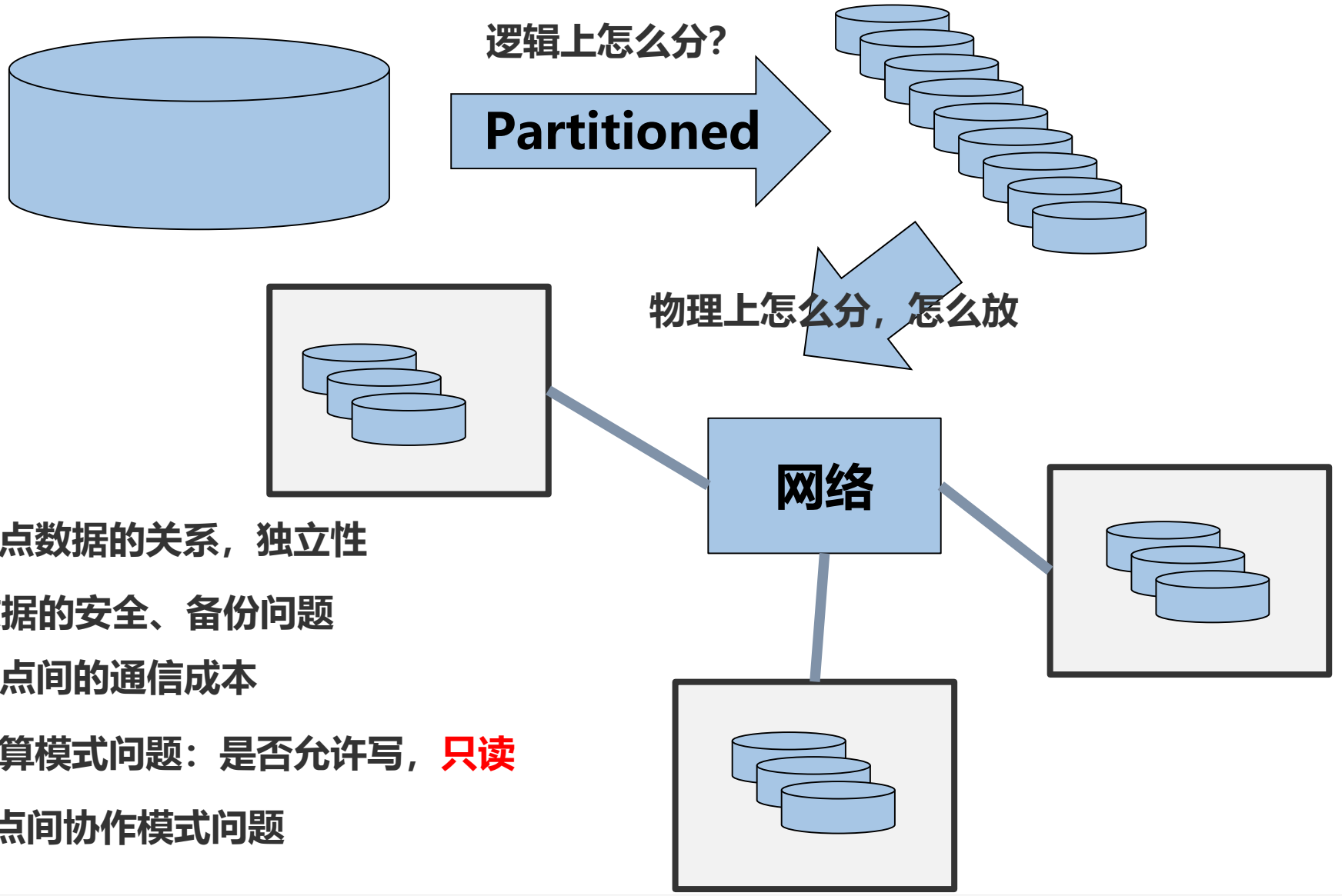
内容提纲

- HDFS、HBase、Hive由来
- HDFS简介
- HBase简介
- Hive简介

背景

- 企业数据仓库和关系型数据库擅长处理结构化数据，并且可以存储大量的数据，要实现高性能大批量数据处理，存在许多问题
  - 实现并行、大批量数据计算成本昂贵
  - 机器升级的难度成指数上涨
  - 对数据结构化的要求限制了可处理的数据种类
  - 面对海量异构数据时敏捷性
- 问题
  - 能否由多台廉价的机器组成的集群，实现大规模数据集的敏捷处理

数据分区与并行计算



节点数据的关系，独立性

数据的安全、备份问题

结点间的通信成本

计算模式问题：是否允许写，只读

节点间协作模式问题

Google的探索

- Google针对只读的大规模数据并行计算解决方案问题进行了一系列的探索。
- 起因：Google要爬取全球的网页然后对其进行排名(PageRank)而衍生出的问题
- 核心思想类似于分治，提出经典解决方案：
  - GFS：Google File System
  - 分布式文件系统，解决的大数据的存储问题
  - 提出著名的BigTable模型，一种NoSQL数据库，将数据存在一张大表之中，用空间换性能
  - 提出经典的计算架构: MapReduce
- 问题：分布式并行计算模式，其中的挑战有哪些?

HDFS的由来

- Apache利用GFS的设计思想，开源设计了HDFS
- 使得不同机器之间繁琐通信对用户实现透明化。
  - 并行计算需要服务器之间进行大量繁琐的通信
- HDFS特点
  - 一个逻辑上实现使用集群和使用一台机器一样
  - 物理上实现多机协同工作的分布式文件系统。
  - 具有高容错性的特点
  - 被设计用来部署在低廉的硬件上
  - 提供高吞吐量，适合超大数据集的应用程序

回顾数据库出现的原因及HDFS存在的问题

- 数据库出现的主要原因
  - 程序与数据的分离
  - 提高数据独立性
  - 实现数据的有效共享
  - 减低程序员编写程序管理与维护数据负担
- HDFS存在的问题
  - 缺少**数据库的概念**，虽然实现了逻辑上像操作单机文件系统那样方便。但HDFS依旧存在着**大量的数据管理工作**。
  - 只有**文件引擎**，纯文本文件，缺少**数据建模、维护与管理能力**
  - 程序员负担太重，**门槛过高**
  - 缺少**透明性支持**

HBase的由来

- Apache基于Google的BigTable思想，实现了开源的面向列式存储的**分布式数据库HBase**。
- 使得Hadoop 在这些年崛起成为拥有着**高性能，高稳定，可管理的**大数据应用平台。
- HBase采用的是**Key/Value的存储方式**，这意味着，即使随着数据量增大，也几乎不会导致查询的性能下降。
- 有Key就可方便建**索引**，因此，HBase具有**高性能的随机读取能力**
- 然而，如此复杂的存储结构和分布式的存储方式带来的代价就是：哪怕只是存储少量数据，**它也不会很快**。
- 所以有人总结：“**HBase并不快，只是当数据量很大的时候它慢的不明显**”。

关于Key/Value

- 关系型模型中的元组实质上是Key/Value
  - Key是一组**主属性的集合**
  - Value是1到多个属性的值构成的集合，每个属性单独存放
  - 数据总体按行存放
- Hbase的Key/Value总体概念类似
  - Key仍然是实体的主键
  - Value是一个列族，其中有一个或多个列，每一个列所对应的值可以有多个版本

CAP theorem & ACID

- ACID: **Atomicity**, **Consistency**, **Isolation**, **Durability**
- It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees
  - **Consistency**： Every read receives the most recent write or an error
  - **Availability**： Every request receives a (non-error) response – without guarantee that it contains the most recent write
  - **Partition tolerance**： The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes
- In other words, the CAP theorem states that in the presence of a network partition, one has to choose between **consistency** and **availability**.

Explanation

- **No distributed system is safe from network failures, thus network partitioning generally has to be tolerated. In the presence of a partition, one is then left with two options: consistency or availability.**
  - **When choosing consistency over availability, the system will return an error or a time-out if particular information cannot be guaranteed to be up to date due to network partitioning.**
  - **When choosing availability over consistency, the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up to date due to network partitioning.**
- In the absence of network failure – that is, when the distributed system is running normally – both availability and consistency can be satisfied.

不同的策略

- Database systems designed with traditional ACID guarantees in mind such as RDBMS choose consistency over availability(一致性高于可用性),
- whereas systems designed around the BASE philosophy, common in the NoSQL movement for example, choose availability over consistency（可用性高于一致性)

HBase的缺点

- 凡事都不可能只有优点而没有缺点。
- 很多业务场景中，统计分析指标可能都是**新定义的**。
- HDFS上的文件是纯文本文件，没有表的一些概念，即使文件存储格式经过良好的设计，进行复杂的MapReduce编程依然不是良好的办法，是程序员巨大负担。
- 数据分析是HBase的弱项，因为对于HBase乃至整个NoSQL生态圈来说，基本上都是**不支持表关联**的，实现RDBMS类似的连接很麻烦。
- 当你想实现group by 或者order by的时候，你会发现，也需要写很多的MapReduce代码来实现。

HIVE的由来

- 为了简化设计实现负担，降低工程难度，提高工程效率，Hive应运而生。它由Facebook开源贡献
  - 底层依旧是HDFS
  - 多了**表的相关概念**
  - 增加**逻辑视图**，减少模型重构的复杂度
  - 独有的**HQL**，类似于关系型数据库的SQL语句，可以减轻学习负担
- 实际上HQL是**转化成MapReduce任务**在HDFS上执行的。

Nosql与RDBMS对比

RDBMS

- 高度组织化结构化数据
- 结构化查询语言（SQL）（SQL)
- 数据和关系都存储在单独的表中。
- 数据操纵语言，数据定义语言
- 严格的一致性
- 基础事务

Nosql

- 非结构或半结构化数据
- 没有声明性查询语言
- 没有预定义的模式
- 键 - 值对存储，列存储，文档存储，图数据库
- 最终一致性，而非ACID属性
- 非结构化和不可预知的数据
- CAP定理
- 高性能，高可用性和可伸缩性

ACID	BASE
原子性(Atomicity)	基本可用(Basically Available)
一致性(Consistency)	软状态/柔性事务(Soft state)
隔离性(Isolation)	最终一致性 (Eventual consistency)
持久性 (Durable)	

采用混合架构的原因

- 关系型数据库对结构化数据具有良好的组织管理，满足查询的实时需求
- Nosql能够存储处理非结构化数据与关系型数据库互补
- 数据不断增多，只能依赖多机器组成的集群上的分布式环境
- 满足实时随机读取的HBase和满足方便多维分析的Hive在应用场景上互补

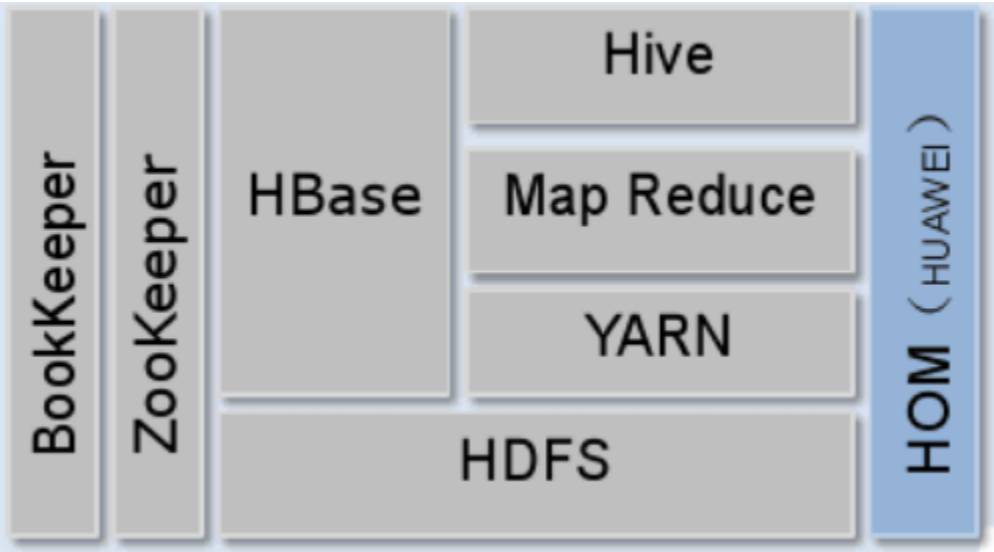
内容提纲

- HDFS、HBase、Hive由来
- HDFS简介
- HBase简介
- Hive简介

HDFS简介

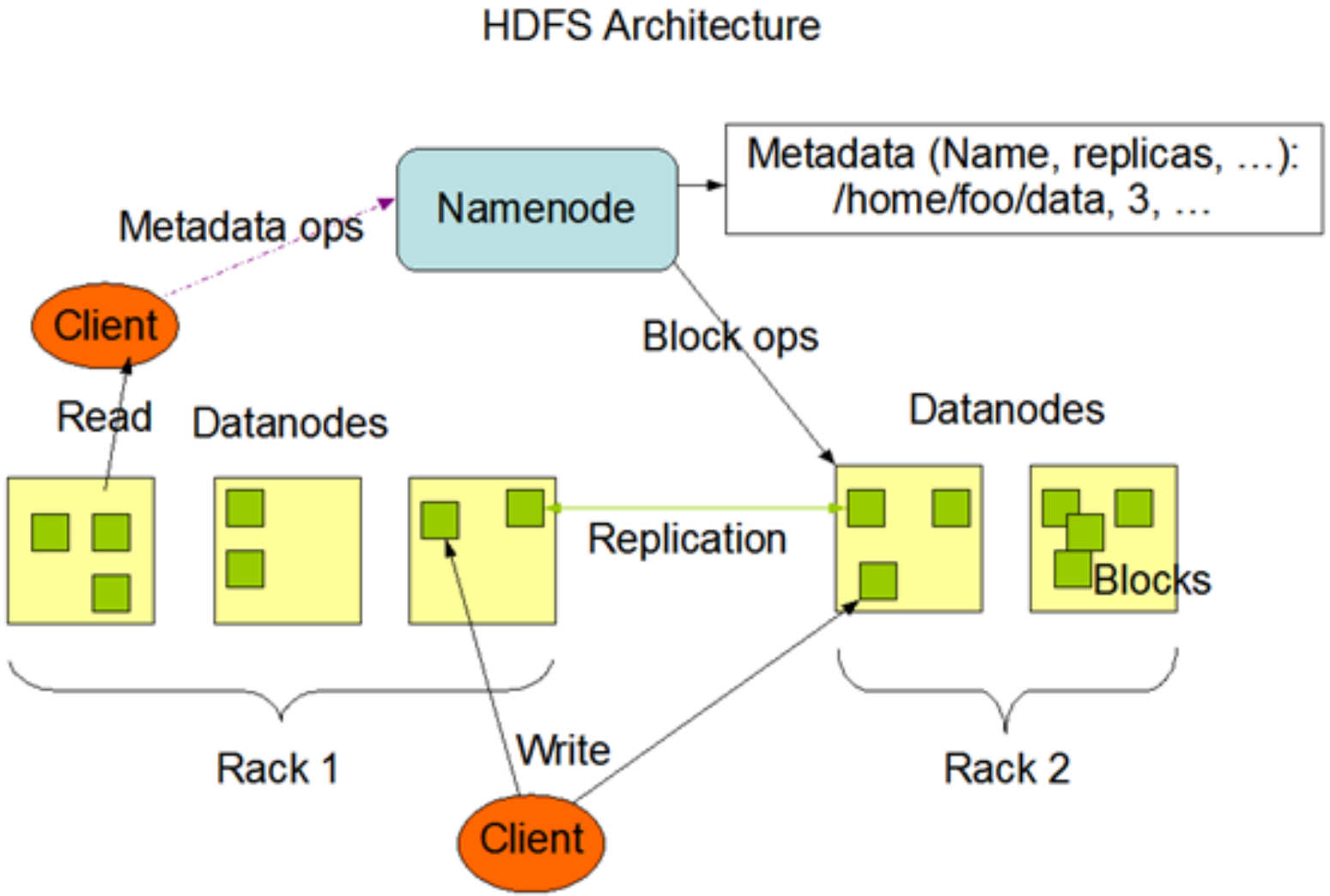
Hadoop分布式文件系统(HDFS)是可以在低端通用硬件上运行的分布式**文件系统**。它与现有的分布式文件系统有很多共同点，但又有明显的区别。

HDFS是一个高度容错性的系统，适合部署在廉价的机器上。HDFS能提供**高吞吐量**的数据访问，非常适合大规模数据集上的应用。

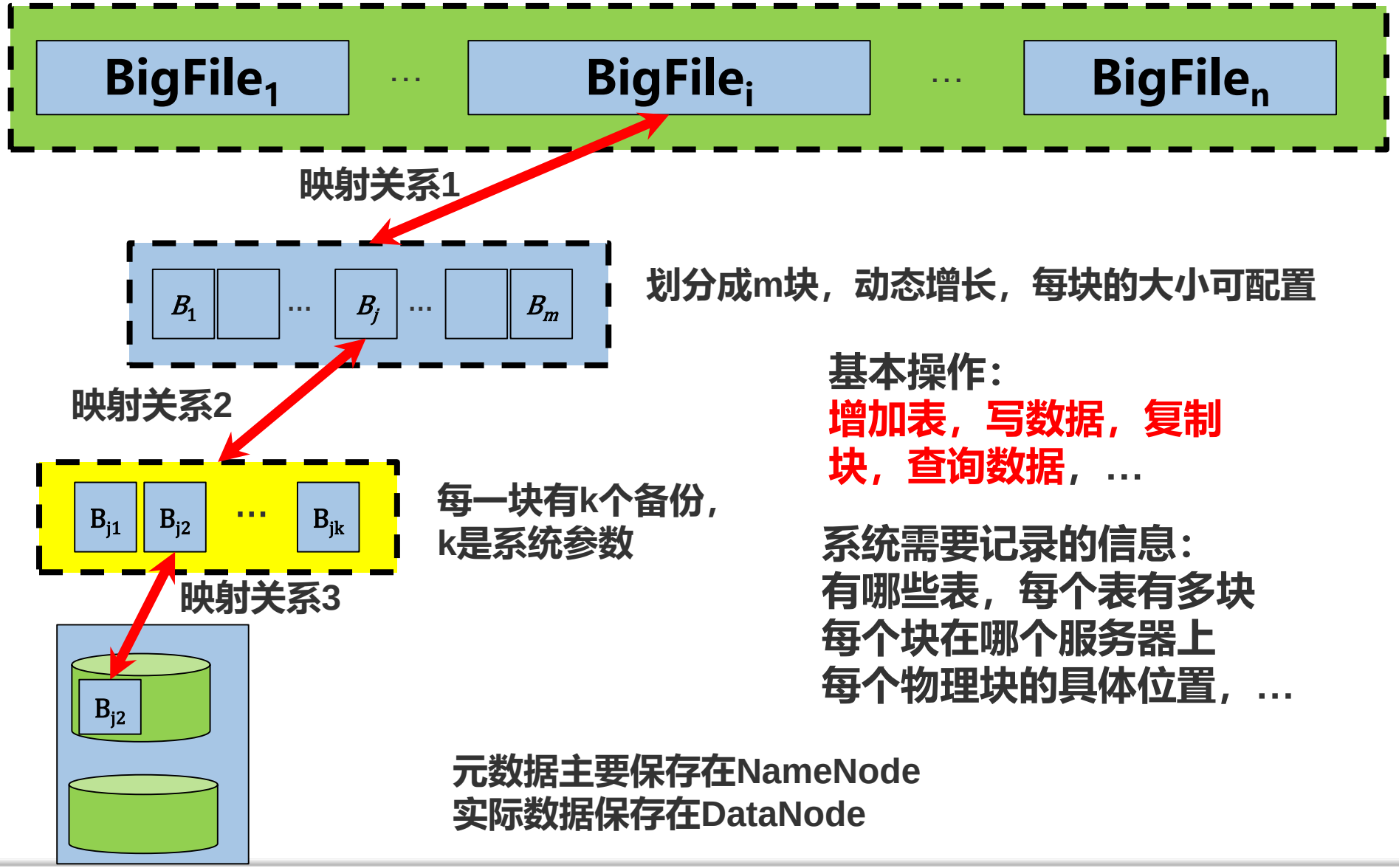


如何实现高吞吐量

HDFS系统架构


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

理解各类关系



基本操作：
增加表，写数据，复制块，查询数据，…

系统需要记录的信息：
有哪些表，每个表有多块
每个块在哪个服务器上
每个物理块的具体位置，…

元数据主要保存在NameNode
实际数据保存在DataNode

NameNode功能

- 负责管理文件系统的**命名空间、集群配置信息**
- 维护着整个文件系统
  - 文件目录树和文件根目录的元信息，
  - 管理**文件与block**之间的映射关系，block与DataNode之间的映射关系
  - 因此NameNode**内存大小**决定了集群的容量
  - 负责数据块的复制
- 接收用户的操作请求
- 处理事务日志：记录文件生成，删除等

DataNode功能

- DataNode是提供真实文件数据的存储服务，将数据块(Block)存储在本地文件系统中，实现HDFS的大部分容错机制
  - 保存数据块的**元数据**，Meta-data
  - 周期性地将所有**存在的数据块信息**发送给NameNode
    - 包括数据块的属性，即数据块属于哪 个文件
    - 数据块 ID ， 修改时间等
  - NameNode中的 **DataNode和数据块的映射关系**就是通过系统启动时DataNode的数据块报告建立的

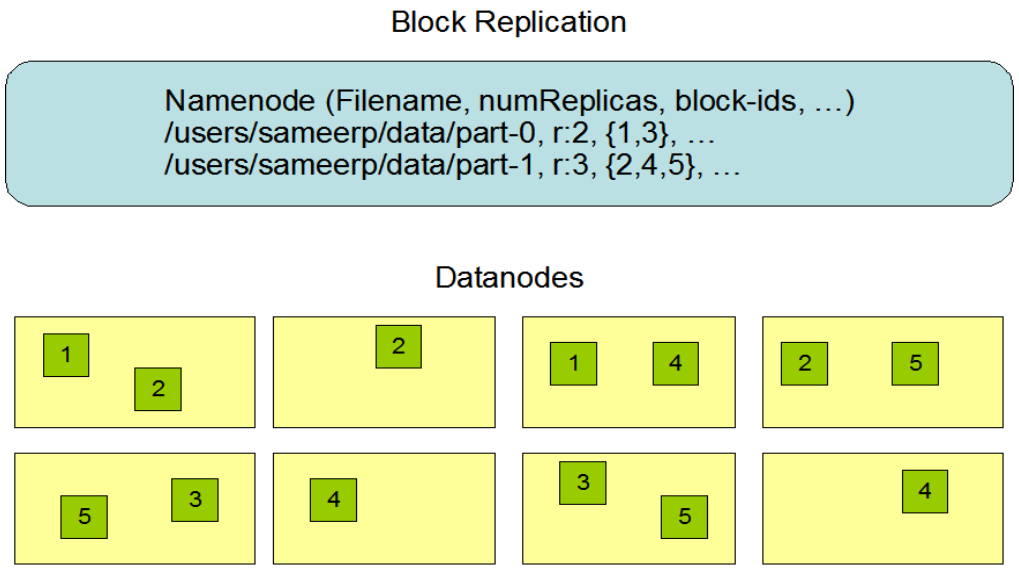
文件系统命名空间

- HDFS命名空间层次结构与现有大多数的文件系统相似:
  - HDFS支持传统的分层文件组织。
  - 用户或者应用可以在这些文件目录下创建子目录并存储文件数据。
  - 用户可以创建、删除、移动或重命名文件。
- NameNode负责维护文件系统的命名空间，记录对命名空间及其属性所有的改动。
- 用户可以在NameNode中指定数据文件有多少份备份，文件的备份数量称为replication。

数据的冗余备份机制

HDFS能实现以高可靠的方式在一个大集群中进行跨节点数据存储。文件被划分为块（block）的集合，按顺序存储。除了最后一个block其他的size都相等。这些block通过冗余备份来增强容错性，而备份数量和size都可以在配置文件中指定。

NameNode对block的复制有绝对的决策权。它会从各节点中周期性地接收心跳（Heartbeat）信号和块报告（Blockreport）信息。Heartbeat表明该Data Node还可以正常工作，而Blockreport则包含了一个该节点所有block的列表。



标准的namenode目录结构（物理模型）

```
├── current
│   ├── VERSION
│   ├── edits_0000000000000000001-0000000000000000007
│   ├── edits_0000000000000000008-0000000000000000015
│   ├── edits_0000000000000000016-0000000000000000022
│   ├── edits_0000000000000000023-0000000000000000029
│   ├── edits_0000000000000000030-0000000000000000030
│   ├── edits_0000000000000000031-0000000000000000031
│   ├── edits_inprogress_0000000000000000032
│   ├── fsimage_0000000000000000030
│   ├── fsimage_0000000000000000030.md5
│   ├── fsimage_0000000000000000031
│   ├── fsimage_0000000000000000031.md5
│   └── seen_txid
└── in_use.lock
```

元数据及日志信息

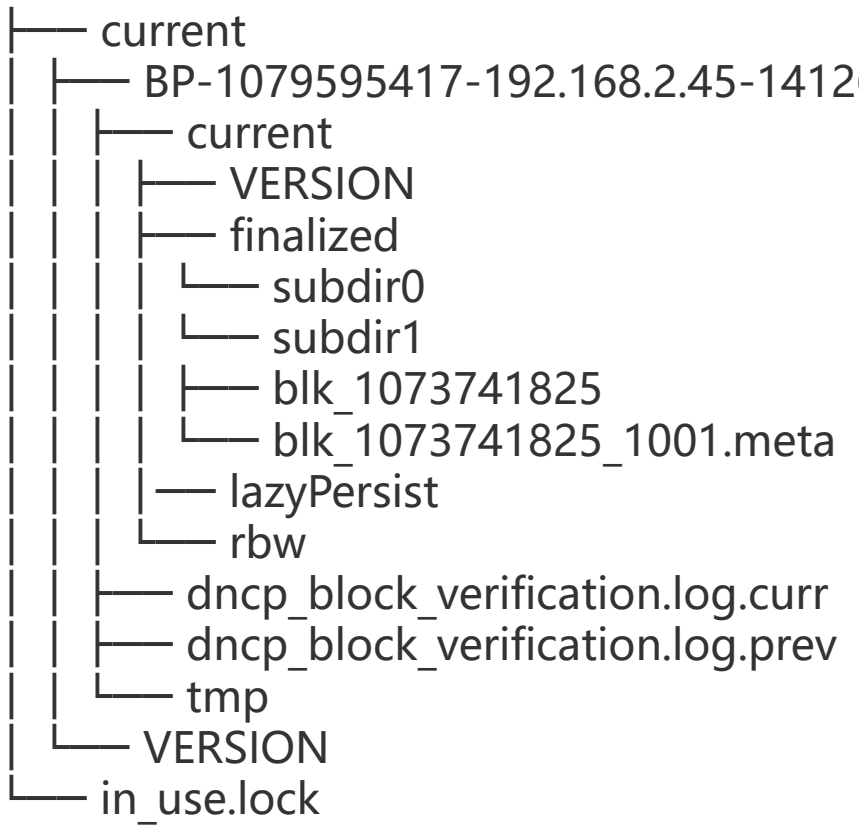
1、version
版本信息
2、edits_start transaction ID-end transaction ID
已经结束的 edit log segments
3、edits_inprogress__start transaction ID
当前正在被追加的edit log
4、fsimage_end transaction ID
每次checkpoing（合并所有edits到一个fsimage的过程）产生的最终的fsimage，同时会生成一个.md5的文件用来对文件做完整性校验
5、seen_txid
保存最近一次fsimage或者edits_inprogress的transaction ID。
6、in_use.lock
防止一台机器同时启动多个Namenode进程导致目录数据不一致

- **edits_start transaction ID-end transaction ID** – These are finalized (unmodifiable) edit log segments. Each of these files contains all of the edit log transactions in the range defined by the file name's through . In an HA deployment, the standby can only read up through the finalized log segments. It will not be up-to-date with the current edit log in progress (described next). However, when an HA failover happens, the failover finalizes the current log segment so that it's completely caught up before switching to active.
- **edits_inprogress__start transaction ID** – This is the current edit log in progress. All transactions starting from are in this file, and all new incoming transactions will get appended to this file. HDFS pre-allocates space in this file in 1 MB chunks for efficiency, and then fills it with incoming transactions. You'll probably see this file's size as a multiple of 1 MB. When HDFS finalizes the log segment, it truncates the unused portion of the space that doesn't contain any transactions, so the finalized file's space will shrink down.

概念解释

- **fsimage** – An fsimage file contains the complete state of the file system at a point in time. Every file system modification is assigned a unique, monotonically increasing transaction ID. An fsimage file represents the file system state after all modifications up to a specific transaction ID.
- **edits** – An edits file is a log that lists each file system change (file creation, deletion or modification) that was made after the most recent fsimage.
- Checkpointing is the process of merging the content of the most recent fsimage with all edits applied after that fsimage is merged in order to create a new fsimage. Checkpointing is triggered automatically by configuration policies or manually by HDFS administration commands.

标准的datanode目录结构（物理模型）



rbw: replica being written

1、BP-random integer-NameNode-IP address-creation time
BP代表BlockPool的意思，就是上面Namenode的VERSION中的集群唯一blockpoolID

2、 version
版本信息

3、finalized/rbw目录
这两个目录都是用于实际存储HDFS BLOCK的数据，里面包含许多block_xx文件以及相应的.meta文件，.meta文件包含了checksum信息

4、dncp_block_verification.log
该文件用于追踪每个block最后修改后的checksum值，该文件会定期滚动，滚动后会移到.prev文件

5、in_use.lock
防止一台机器同时启动多个Datanode进程导致目录数据不一致

设计理念

1、存储超大文件

运行在HDFS上的应用程序使用大数据集--几百MB、GB甚至TB级别的文件。

2、流式数据访问

HDFS主要是为批量处理而设计的，而不是为普通用户的交互使用。运行在HDFS上的应用程序一般采用流式的访问它们的数据集，强调的是数据访问的高吞吐量而不是数据访问的低延迟时间。

HDFS存储的数据集作为hadoop的分析对象。在数据集生成后，可以在此数据集上进行各种长时分析。这类分析可能会涉及该数据集的大部分数据甚至全部数据，因此读取整个数据集的时间延迟比读取第一条记录的时间延迟更重要。

设计理念

3、运行在普通廉价的服务器上

硬件错误是正常现象而不是异常。HDFS实例由成百上千个服务器组成，每个都存储着文件系统的一部分数据，这就会有大量的组成部分，而每个组成部分出故障的可能性都很大，这意味着HDFS总有一些服务器是不能工作的。因此，检测错误并快速自动恢复就成了HDFS的核心设计目标。

4、简单一致性模型

HDFS的应用程序需要对文件实行一次性写，多次读的访问模式。文件一旦建立后写入，文件就不需要再更改了。这样的假定简化了数据一致性问题并使高数据吞吐量成为可能。

设计理念

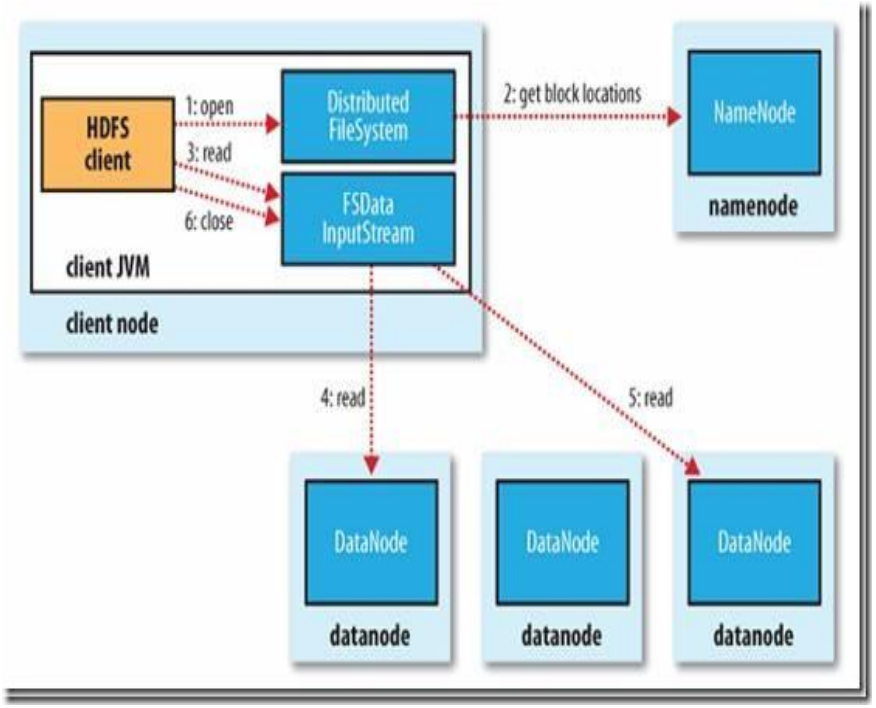
5、移动计算环境比移动数据划算

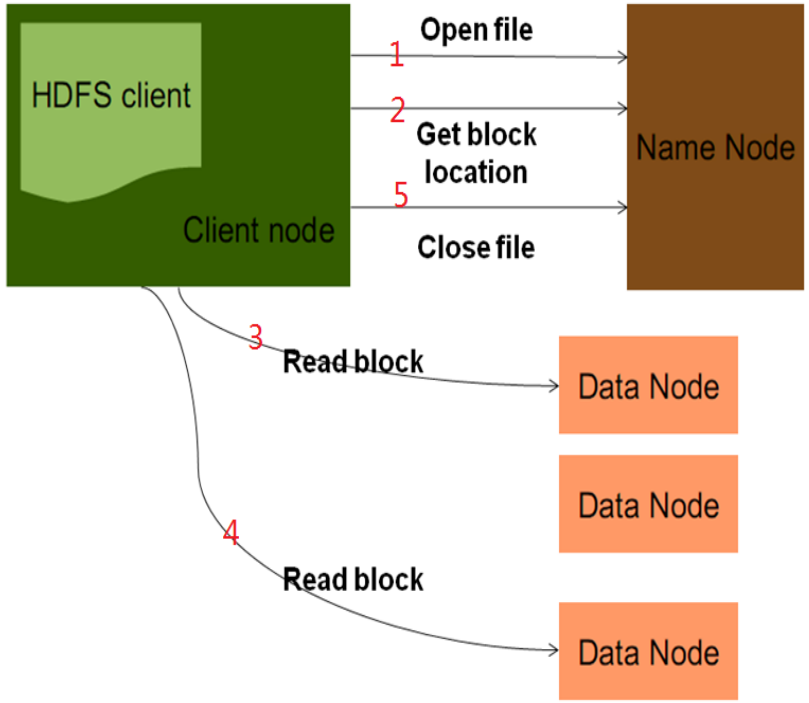
如果就在数据的旁边来执行对这些数据的操作，那么程序所使用的设备就会很高效。这当文件相当巨大的时候就尤其正确。还可以减少网络的拥塞和提高系统的吞吐量。HDFS提供了程序接口以便把他们自己移动到数据存储的地方执行

6、跨硬件和软件平台的移动

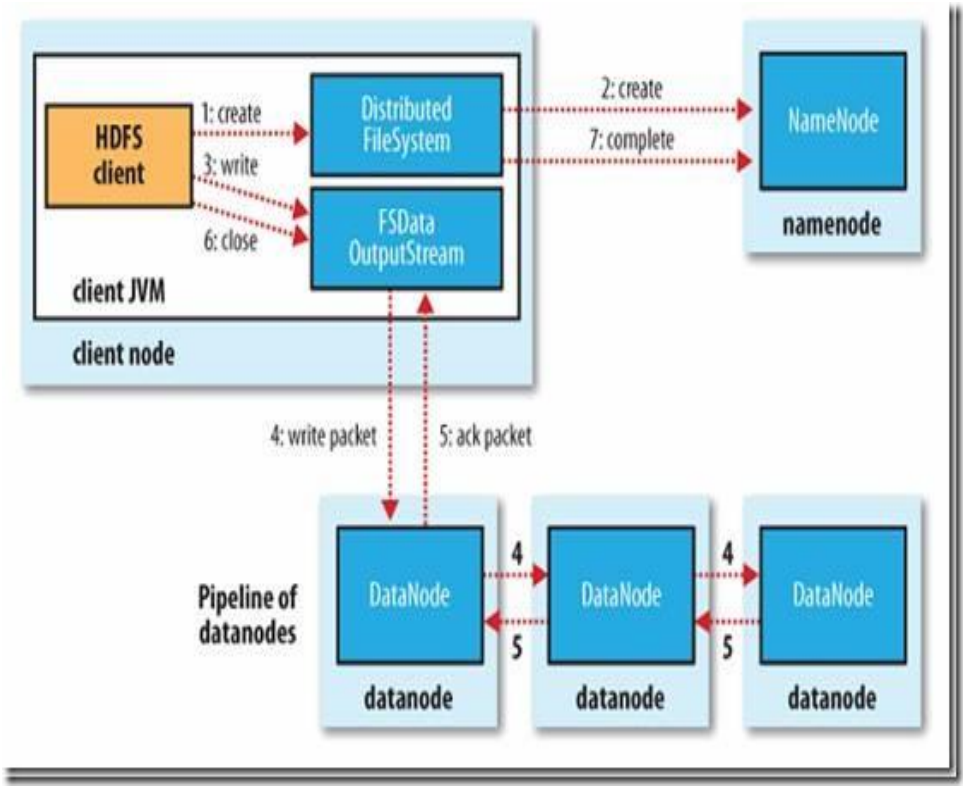
HDFS设计为容易的从一个平台移动到另一个平台。这有助于HDFS被采用做为一个大程序集合的工作平台。

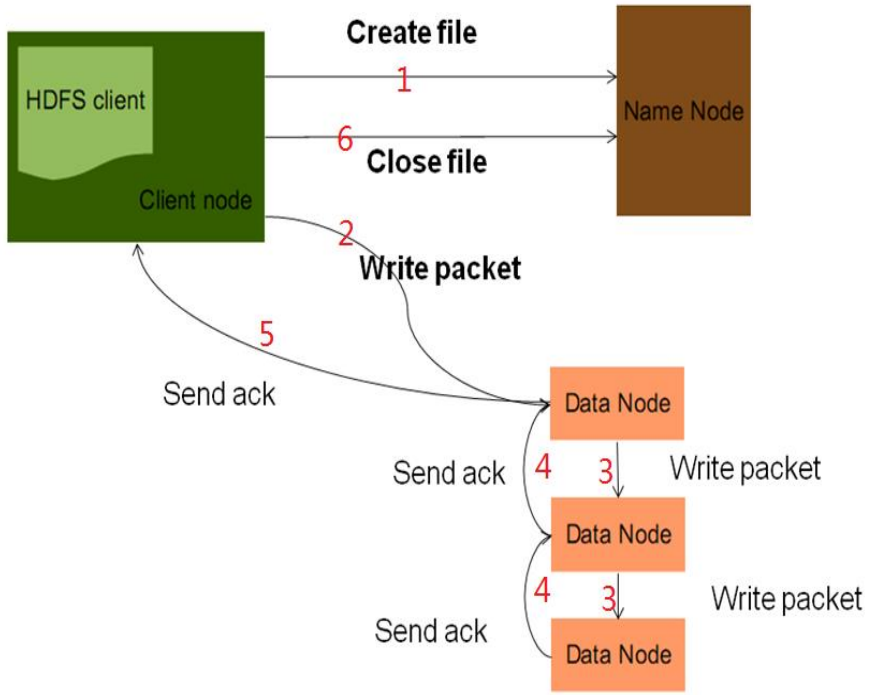
HDFS读文件


HDFS client
1: open
3: read
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: get block locations
NameNode
namenode
4: read
5: read
DataNode
DataNode
DataNode
datanode
datanode
datanode


HDFS client
Client node
1
Open file
2
Get block location
5
Close file
Name Node
3
Read block
4
Read block
Data Node
Data Node
Data Node

HDFS写文件


HDFS client
1: create
3: write
6: close
Distributed FileSystem
FSData OutputStream
client JVM
client node
2: create
7: complete
NameNode
namenode
4: write packet
5: ack packet
Pipeline of datanodes
DataNode
DataNode
DataNode
4
5
4
5
datanode
datanode
datanode


HDFS client
Client node
Create file
1
6
Close file
Name Node
2
Write packet
5
Send ack
Data Node
Send ack
4
3
Write packet
Data Node
4
3
Write packet
Send ack
Data Node

内容提纲

- HDFS、HBase、Hive由来
- HDFS简介
- **HBase简介**
- Hive简介

HBase简介

- HBase是Apache的Hadoop项目的子项目。
- 建立在Hadoop文件系统之上的分布式面向列的数据库
- HBase利用了Hadoop的文件系统（HDFS）提供的容错能力。适合存储海量非结构化数据或半结构化数据、具备
  - 高可靠性、高性能、可灵活扩展伸缩、支持实时数据读写

When Should I Use HBase?

- HBase isn't suitable for every problem.
- First, make sure you **have enough data.** If you have hundreds of millions or billions of rows, then HBase is a good candidate. If you only have a few thousand/million rows, then using a traditional RDBMS might be a better choice due to the fact that all of your data might wind up on a single node (or two) and the rest of the cluster may be sitting idle.
- Second, make sure you **can live without all the extra features** that an RDBMS provides (e.g., typed columns, secondary indexes, transactions, advanced query languages, etc.) An application built against an RDBMS cannot be "ported" to HBase by simply changing a JDBC driver, for example. Consider moving from an RDBMS to HBase **as a complete redesign** as opposed to a port.
- Third, make sure you have **enough hardware**. Even HDFS doesn't do well with anything less than 5 DataNodes (due to things such as HDFS block replication which has a default of 3), plus a NameNode.

What Is The Difference Between HBase and Hadoop/HDFS?

- HDFS is a distributed file system that is well suited for the storage of large files. Its documentation states that it is not, however, a general purpose file system, and does not provide fast individual record lookups in files. HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables.
- This can sometimes be a point of conceptual confusion. HBase internally puts your data in indexed "StoreFiles" that exist on HDFS for high-speed lookups.

HBase基础设计思想

- HBase是一个面向列的数据库，在表中它由行排序。
- 表模式定义只能列族，也就是键值对。一个表有多个列族，每一个列族可以有任意数量的列。
- 后续列的值连续地存储在磁盘上。表中的每个单元格值都具有时间戳。
- 对于Null值的存储是不占用任何空间的
- 最适合使用HBase存储的数据：**非常稀疏的数据**
- 所有数据库更新操作都有时间戳。HBase对每个数据单元，只存储**指定个数的最新版本**。

HBase的特点

行式数据库	列式数据库
适用于联机事务处理（OLTP）	适用于在线分析处理（OLAP）
这样的数据库被设计为小数目的行和列	面向列的数据库设计的巨大表

1.HBase线性可扩展
2.表大，一个表有上亿行，上百万行
3.稀疏，一行数据不是所有列都有数据，空值不占空间
4.事务性数据只存在于单行以内
5.每一列数据类型相似，高效压缩存储

Data Model of HBase

- **Table**
  - An HBase table consists of multiple rows
- **Row**
  - A row in HBase consists of a row key and one or more columns with values associated with them. Rows are sorted alphabetically by the row key as they are stored. For this reason, the design of the row key is very important. The goal is to store data in such a way that related rows are near each other.
- **Column**
  - A column in HBase consists of a column family and a column qualifier, which are delimited by a : (colon) character.

Data Model of HBase

Column Family

- Column families physically colocate a set of columns and their values, often for performance reasons. Each column family has a set of storage properties, such as whether its values should be cached in memory, how its data is compressed or its row keys are encoded, and others. Each row in a table has the same column families, though a given row might not store anything in a given column family.

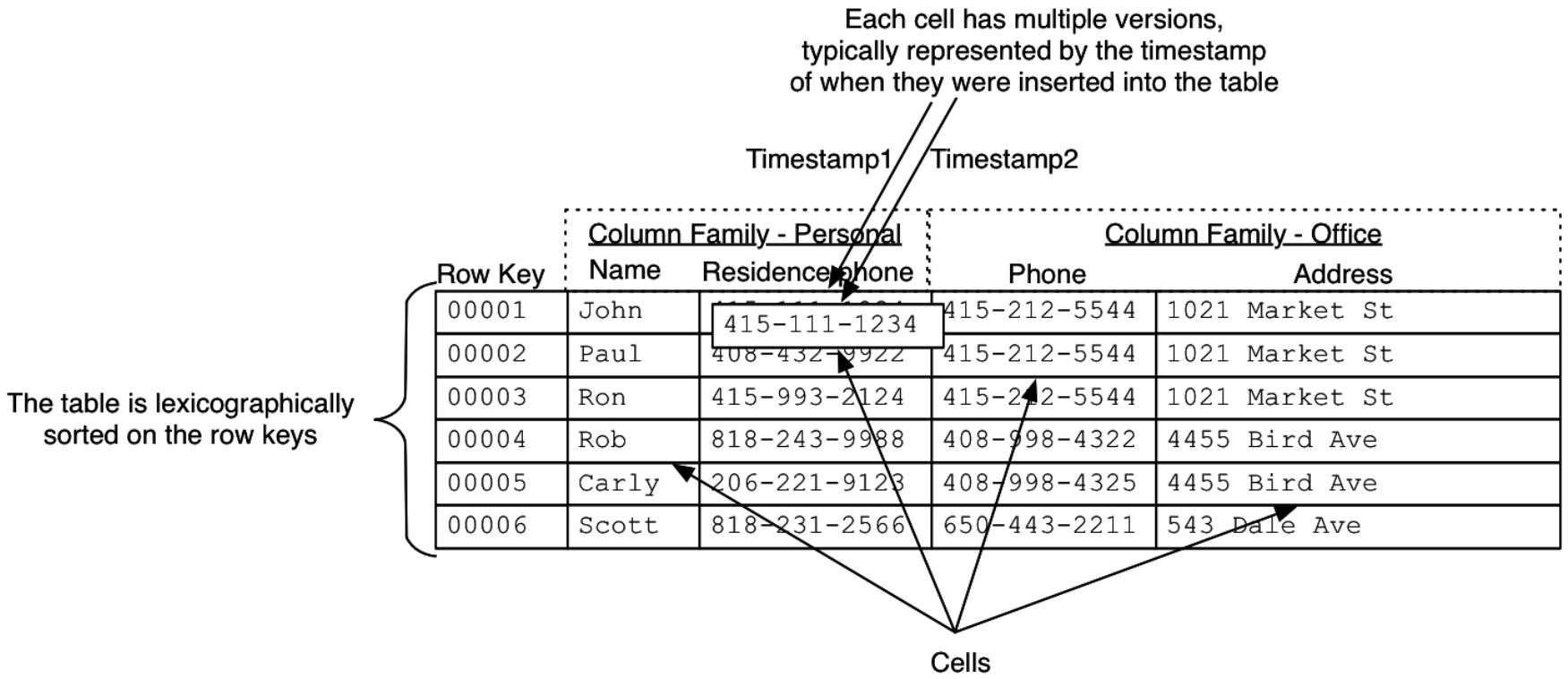
Column Qualifier

- **A column qualifier is added to a column family to provide the index for a given piece of data. Given a column family content, a column qualifier might be content:html, and another might be content:pdf. Though column families are fixed at table creation, column qualifiers are mutable and may differ greatly between rows.**

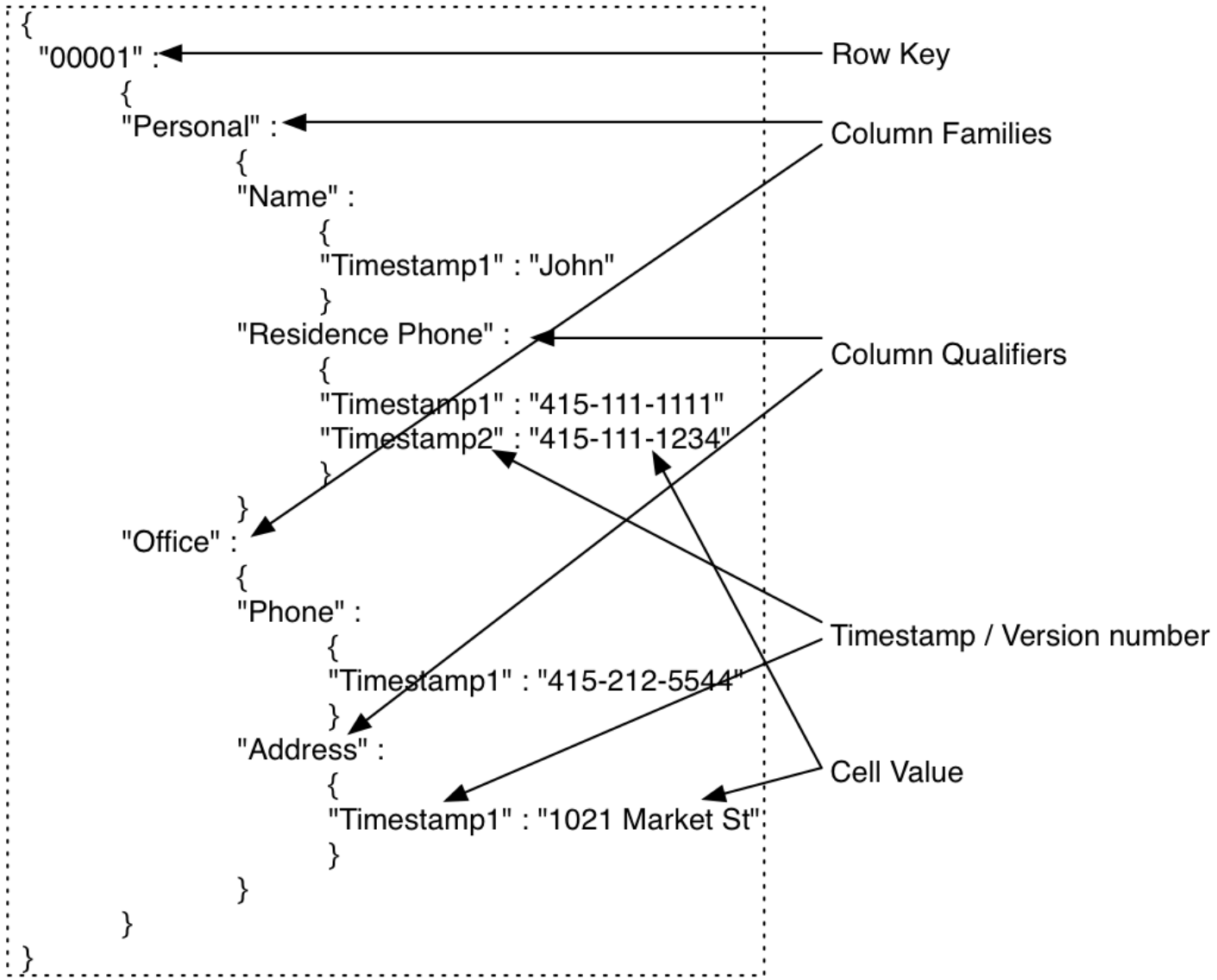
Data Model of HBase

- **Cell**
  - A cell is a combination of row, column family, and column qualifier, and contains a value and a timestamp, which represents the value's version
- **Timestamp**
  - A timestamp is written alongside each value, and is the identifier for a given version of a value. By default, the timestamp represents the time on the RegionServer when the data was written, but you can specify a different timestamp value when you put data into the cell

Example


Each cell has multiple versions,
typically represented by the timestamp
of when they were inserted into the table
Timestamp1
Timestamp2
Column Family - Personal
Column Family - Office
Row Key
Name
Residence phone
Phone
Address
00001 John 415-111-1234 415-212-5544 1021 Market St
00002 Paul 408-432-9922 415-212-5544 1021 Market St
00003 Ron 415-993-2124 415-212-5544 1021 Market St
00004 Rob 818-243-9988 408-998-4322 4455 Bird Ave
00005 Carly 206-221-9123 408-998-4325 4455 Bird Ave
00006 Scott 818-231-2566 650-443-2211 543 Dale Ave
The table is lexicographically
sorted on the row keys
Cells

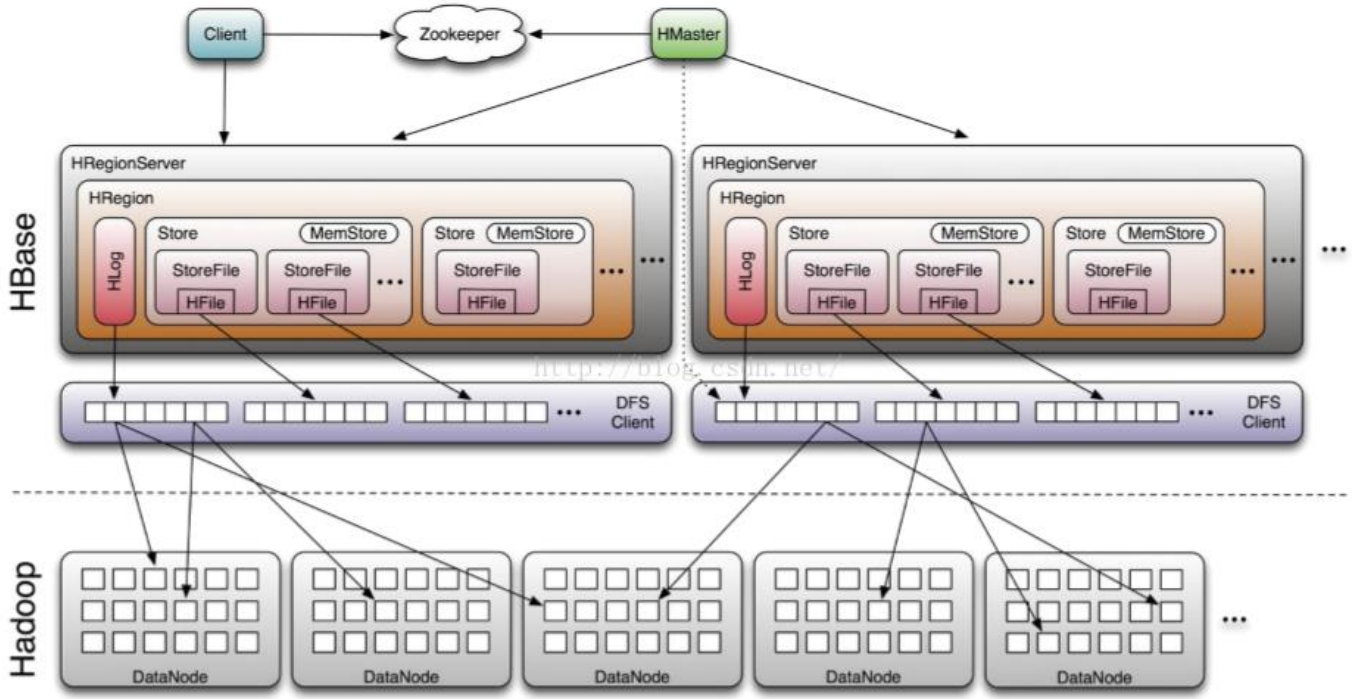
Multidimensional map



HBase逻辑模型

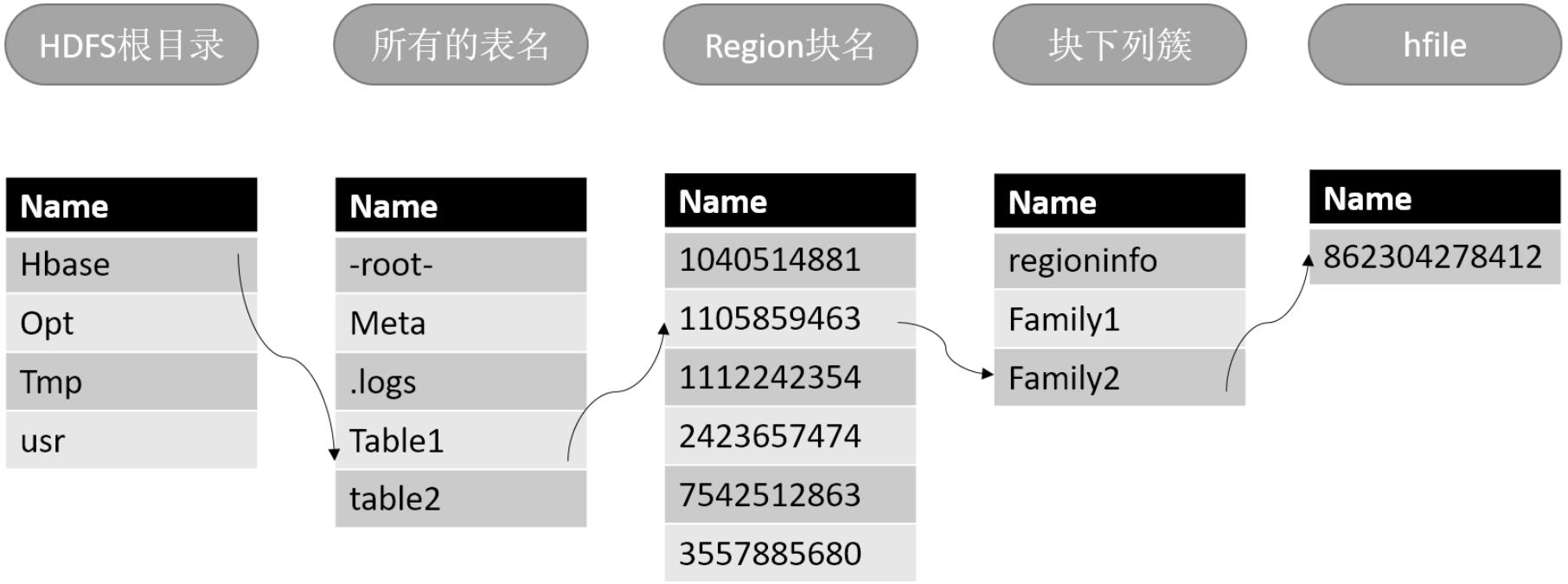
Row Key	**Time Stamp**	**Column** *"contents:"*	**Column** *"anchor:"*		**Column** *"mime:"*
"com.cnn.www"	t9		"anchor:cnnsi.com"	"CNN"	
	t8		"anchor:my.look.ca"	"CNN.com"	
	t6	"<html>..."			"text/html"
	t5	"<html>..."			
	t3	"<html>..."			

HBase系统架构



- Hmaster存储元数据，告诉客户端数据在哪里
- 每个节点对应一个HRegionServer。下有好几个Hregion（对应table的一个region）。
- Hregion有许多store（对应table的列簇）。每个store包含一个memstore（内存缓冲区）和许多storefile和hfile（列）。
- 每个hregion包含一个hlog（日志保持数据一致性）

HBase物理存储模型



HBase的优缺点

◆优点

1. 列可以动态增加，并且列为空就不存储数据,节省存储空间
2. Hbase自动切分数据，使得数据存储自动具有水平scalability
3. Hbase可以提供高并发读写操作的支持
4. 有时间戳，适合有关时间的查询
5. 最近的数据可能还在memstore，加上多级索引方式，io小或无，查询速度快。

◆缺点

1. 不能支持条件查询，只支持按照Row key来查询
2. 暂时不能支持Master server的故障切换,当Master宕机后,整个存储系统就会挂掉

HBase数据模型设计的注意事项

1. Rowkey设计最重要（RowKey的结构该如何设置，而RowKey中又该包含什么样的信息），设计的好坏直接影响程序和HBase交互的效率和数据存储的性能
2. 每一行应该包含什么信息，涉及检索速度
3. 建表预分区，分区数量和分区策略都要考虑，避免某些region过热，其余过冷
4. 一般情况下列簇尽量不要超过2个，同列簇内列多一点没关系，因为列没有不存储，列簇是要在底层建立文件的，而且列可活动添加，列簇建表的时候要指定。
5. 每个单元中存储多少个版本信息
6. 以纵向扩张为主设计的表结构能快速简单的获取数据，但牺牲了一定的原子性；而以横向扩张为主设计的表结构，也就是列族中有很多列
7. HBase并不支持事务，所尽量在一次API请求操作中获取到结果（不要设计涉及修改的表结构）
8. 列标识(Column Qualifier)名字的长度和列族名字的长度都会影响I/O的读写性能和发送给客户端的数据量，所以它们的命名应该简洁

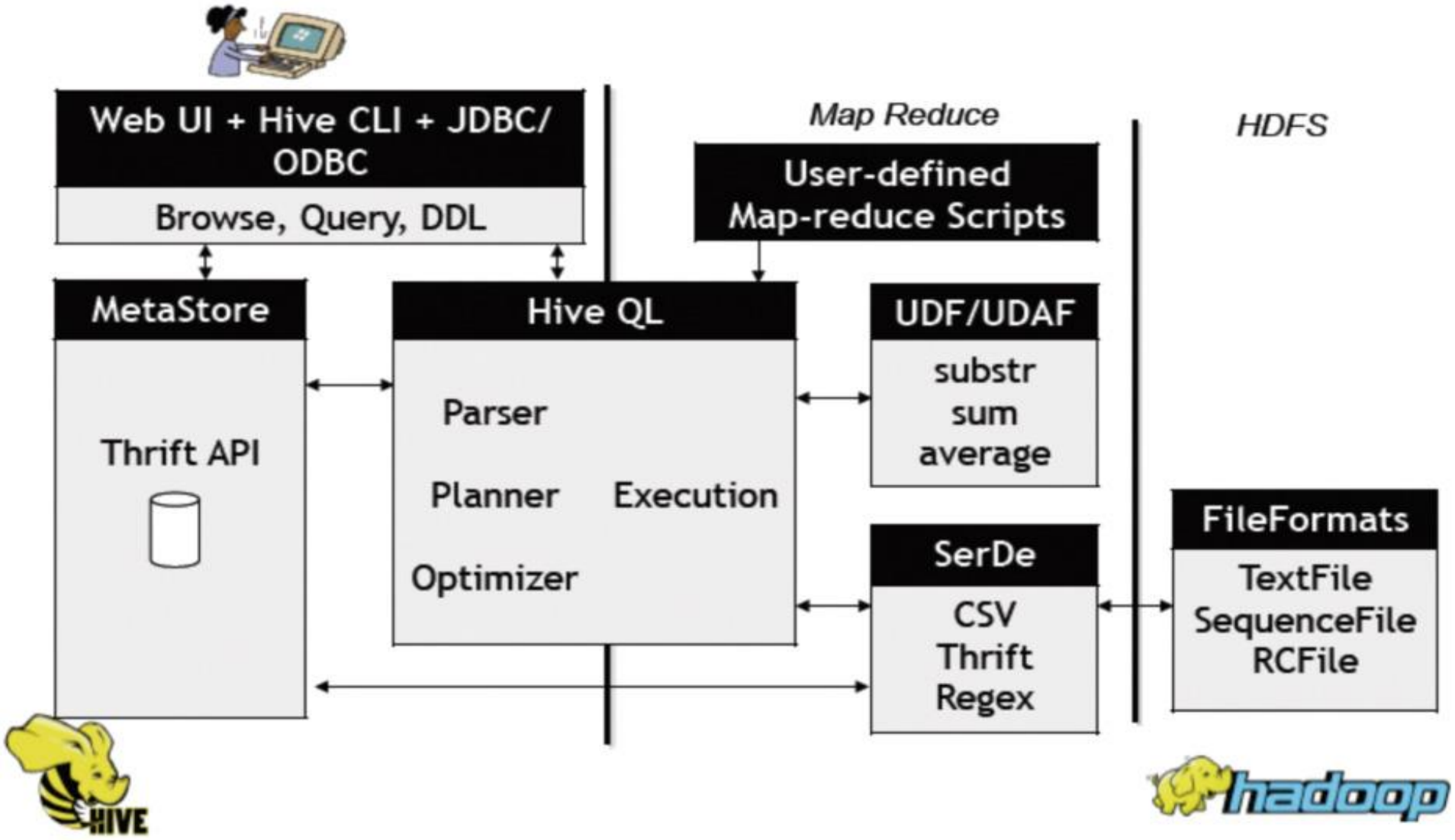
内容提纲

- HDFS、HBase、Hive由来
- HDFS简介
- HBase简介
- Hive简介

Hive简介

Hive由Facebook实现并开源，是建立在Hadoop之上的数据仓库解决方案，支持将结构化的数据文件映射为一张表，提供HQL（Hive SQL）实现方便高效的数据查询，底层数据存储在HDFS上。Hive的本质是将HQL转换为MapReduce程序去执行，使不熟悉MapReduce的用户很方便地利用HQL进行数据ETL操作。

Hive系统架构

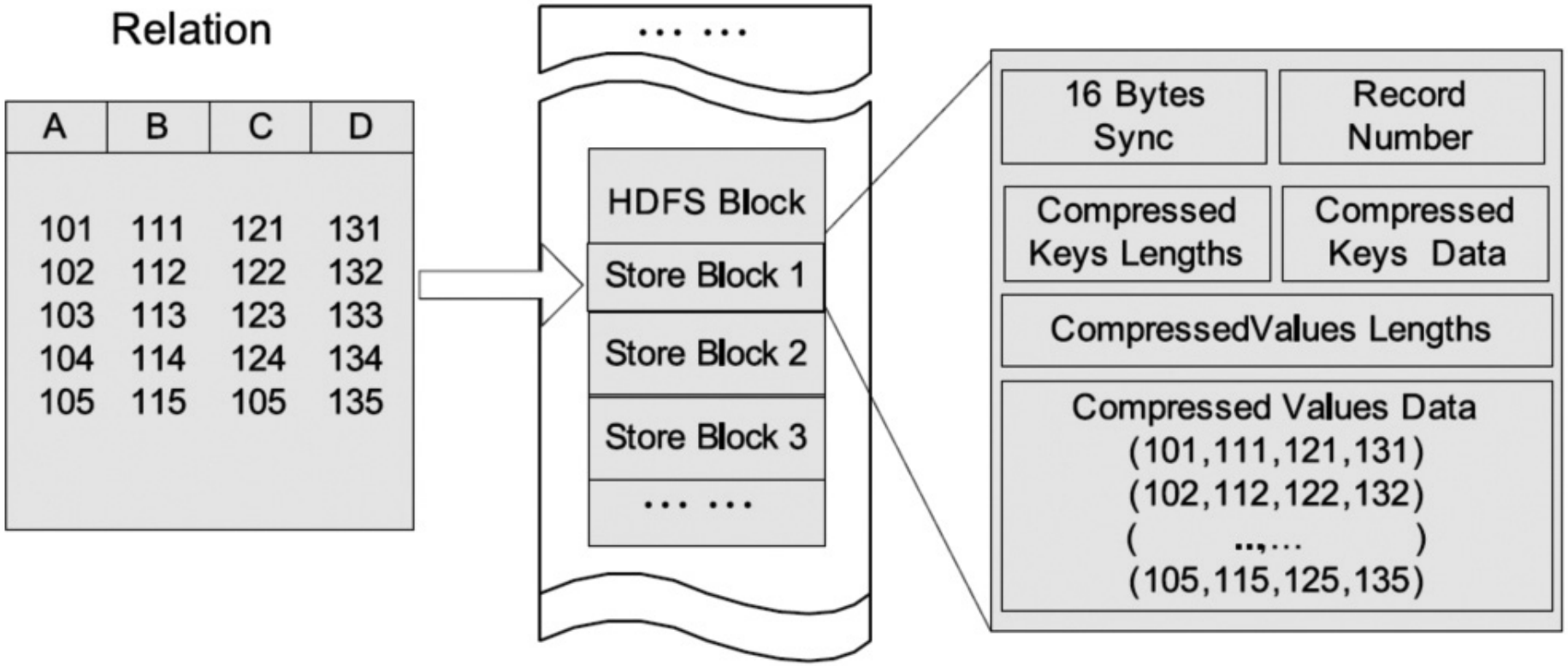

Web UI + Hive CLI + JDBC/ ODBC
Browse, Query, DDL
Map Reduce
User-defined Map-reduce Scripts
HDFS
MetaStore
Thrift API
Hive QL
Parser
Planner
Execution
Optimizer
UDF/UDAF
substr
sum
average
SerDe
CSV
Thrift
Regex
FileFormats
TextFile
SequenceFile
RCFile
HIVE
hadoop

Hive逻辑存储模型

逻辑结构和MySql等关系型数据库一样，如下图：

Name	Age	Id
mike	11	2177867
jack	56	9080973

Hive物理存储模型


Relation
A
B
C
D
101 111 121 131
102 112 122 132
103 113 123 133
104 114 124 134
105 115 105 135
HDFS Block
Store Block 1
Store Block 2
Store Block 3
16 Bytes Sync
Record Number
Compressed Keys Lengths
Compressed Keys Data
CompressedValues Lengths
Compressed Values Data
(101,111,121,131)
(102,112,122,132)
(105,115,125,135)

Hive数据存储

➢ 元数据：基于 RDBMS 存储

◆Hive将元数据存储在数据库中，连接到这些数据库(mysql, derby)的模式分三种：单用户模式、多用户模式、远程服务器模式。

◆元数据包括Database、表名、表的列及类型、 存储空间、分区、表数据所在目录等

➢ 其它数据：基于 HDFS 存储

Hive中的表

1、内部表（Internal Table）

与传统数据库中的表类似，在Hive中每一个Table都有对应的目录存取其数据。假设表在HDFS上的存储路径为/warehouse/test，则表test的所有数据都会存储在该路径下。当删除表时，其元数据和表中的数据都会被删除。

例 句： create table test_inner_table (key string)

Hive中的表

2、外部表（External Table）

与内部表不同，外部表在创建过程中，并不会移动数据到数据仓库的目录中，只是与外部数据建立一个链接。删除外部表时，只会删除该外部表的元数据，外部数据不会删除。因此如果想在数据加载时跳过导入数据过程，推荐采用外部表。

例句：　create external table test_external_table (key string)

Hive中的表

3、分区表（Partition Table ）

在Hive中表的一个Partition对应表下的一个子目录，所有Partition的数据都存储在对应子目录中。
假设test表包含date和lang两个partition，则对应于date=20140428, lang=zh 的HDFS子目录 为 /warehouse/test/date=20140428/lang=zh。

例 句：
create table test_partition_table (key string) partitioned by (dt string)。

Hive中的表

4、桶表（Bucket Table）

选择表的某列通过Hash算法将表横向切分成不同的文件存储，具备这类属性的表称之为Bucket Table。
如果业务计算需采用Map 并行方式，推荐创建 Bucket Table 。

例 句： create table test_bucket_table (key string) clustered by (key) into 20 buckets

视图（View）

- 与传统数据库视图类似，只读，基于基本表创建。
- 如果基本表改变，增加数据或增加列不会影响视图的呈现；但如果删除列或删除数据，会影响视图的呈现。
- 基于基本表创建视图时，如果不指定视图的列，视图的列由select语句生成。
- 例句：create view test_view as select * from test_inner_table。

Hive表设计原则

✓在创建表时，先分析字段的内容，对应关系型数据库的类型选择与之相匹配的类型。

✓Hive分为内部表和外部表，对于不做经常变更的表数据，可以参考外部表创建模式。如果业务计算需采用Map并行方式，推荐创建桶表。

Hive和RDBMS的对比

	Hive	RDBMS
查询语言	HQL	SQL
数据存储	HDFS	本地文件系统
执行	MapReduce	非MapReduce
数据处理规模	大	小
索引	简单索引	复杂索引
事务	不支持	支持
执行延迟	高	低
应用场景	海量数据查询	实时查询
可扩展性	好	差
数据加载模式	读时模式（快）	写时模式（慢）
数据操作	数据覆盖追加	行级更新删除

Hive Vs HBase

Hive主要是为简化编写MapReduce程序而生，而HBase是为查询而生。

通过Hive的存储接口，Hive和HBase可以整合使用。

Hive通过与HBase集成后能够实现快速的查询。

同时，由于HBase不支持类SQL语句，通过与Hive集成，Hive为HBase提供SQL语法解析的外壳。

	Hive	HBase
存储方式	行存	列存
执行延迟	高	低
数据格式	结构化的	非结构化的
应用场景	面向分析，append-only批处理	面向查询，行级更新
表类型	纯逻辑表（不存数据）	物理表（存储数据）
查询效率	低	高
SQL	提供HQL	不支持，NoSQL
索引	简单索引	简单索引
事务	不支持	仅支持单行事务
行更新	不支持	支持
数据库类型	关系型	非关系型

数据模型与性能优化设计

性能优化考虑的主要因素

- 时间
  - 不同类别操作需要耗费的时间
- 空间
  - 存储空间，外存，内存
- 成本
  - 维护成本、运行成本

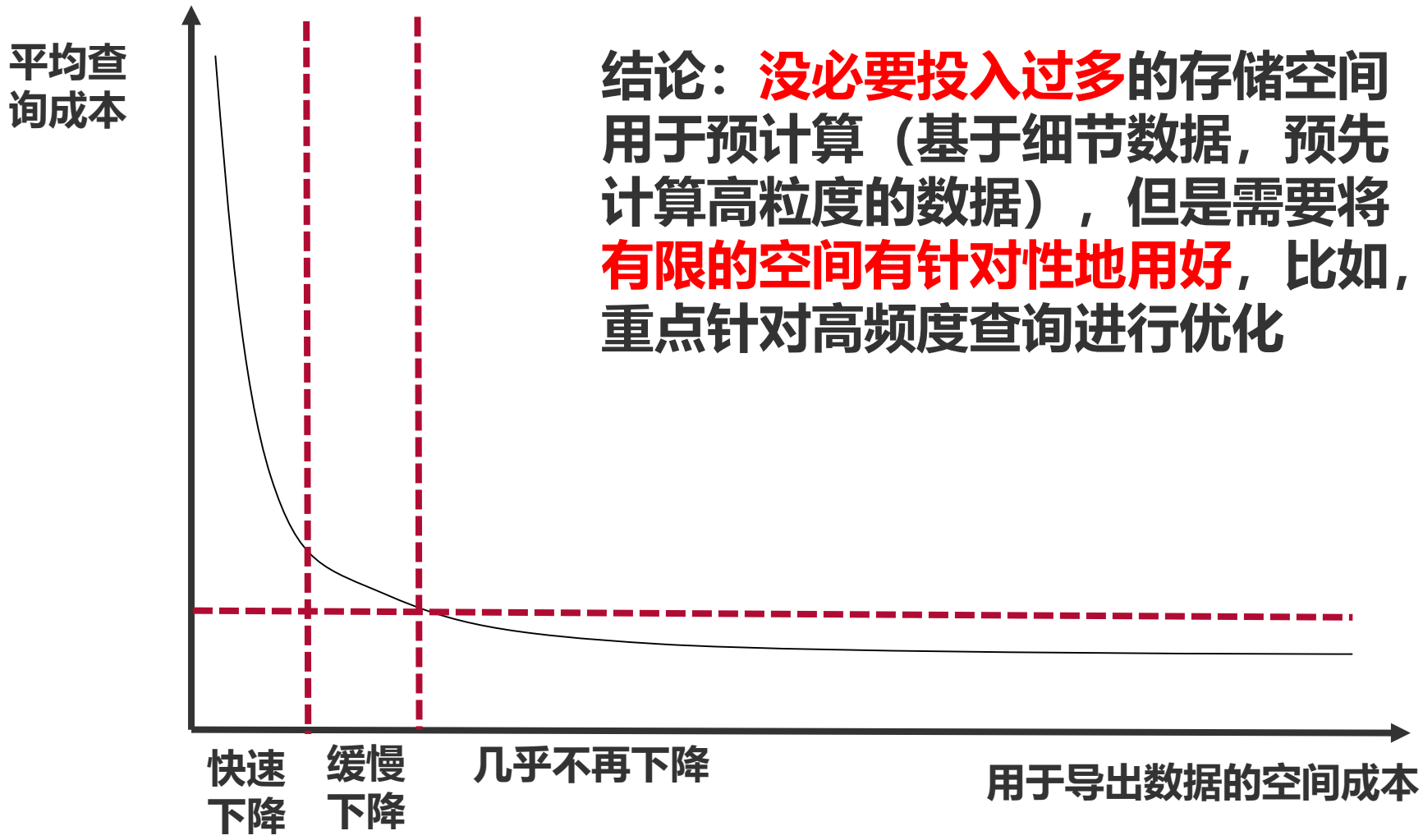
影响系统性能的主要因素

- CPU
  - 处理机核数及性能
- 磁盘I/O
  - 磁盘速度
- 内存
  - 内存速度与大小
- 网络通信
  - 网络带宽：背板、端口、通道

数据体系粒度级设计

- 粒度级设计问题
  - 设计有哪些输入参数?
  - 设计输出是什么?
- 输出
  - 要保存哪些细节数据，要保存多少时间
  - 需要保存哪些粒度级别的数据，保存多少时间
- 输入
  - 系统整体预计的查询模式
    - (查询，预计查询频率，响应时间要求)
  - 系统具有的内外存空间
  - 其他硬件环境

多粒度数据设计中的经典空间与时间关系



结论：**没必要投入过多**的存储空间用于预计算（基于细节数据，预先计算高粒度的数据），但是需要将**有限的空间有针对性地用好**，比如，重点针对高频度查询进行优化

维护成本与空间成本成正比

数据划分及其策略

- 文件分区
- 模式分解
  - 横向分解、纵向分解
- Hbase
  - 列族、分区
- HIVE
  - 分区、分桶
- RDBMS
  - 分区

本部分结束！