CS5200 Database Management XML, JSON and Storage Devices

Ken Baclawski
Spring 2017

Outline

- Data Models
- The Hierarchical Model
  - XML Data
  - JSON Data
- Hierarchical Query Languages
  - XPath
  - HAPL
- Comparison of XML and JSON
- Storage Devices
- Assignment #8

Data Models

Data Models

- For each major data model we discuss:
  - Mathematical basis for the data structure
  - Formats for storage and serialization
  - Schema languages
  - Query and update languages
  - Primary purpose

Classification of Data Models

- Data Models and Modeling Languages
  - Hierarchical
  - Network
  - Relational
- All three models continue to be popular
  - Each with its own class of applications

Hierarchical Model

- Models data with a containment hierarchy
- The oldest data model
  - Very ancient and embedded in most cultures and social systems
- The model for books and documents
  - Division into paragraphs, sections, chapters, ...
- Early database systems used this model
- Used extensively for data interchange, such as ASN.1, XML, JSON

Data Model Features

- Mathematical basis for the data structure: infoset
- Formats for storage and serialization: XML, JSON
- Schema languages: DTD, XML Schema
- Query and update languages: XPath, HAPL
- Primary purpose: data interchange

© 2017 Ken Baclawski All Rights Reserved

Network Model

- The model for programming language data structures
  - Objects point to other objects
  - The whole structure is a directed graph
- The model for the web
  - Hypertext documents with links to other documents
- Used in early database systems but not popular
- Object database systems use this model but they have also never been popular

© 2017 Ken Baclawski All Rights Reserved

Data Model Features

- Mathematical basis for the data structure: directed graphs, semantic networks
- Formats for storage and serialization: Programming language internal storage structures, graph databases
- Schema languages: UML, MOF, RDF/S, OWL
- Query and update languages: Object-oriented programming languages, SPARQL
- Primary purpose: Programs, Data semantics

© 2017 Ken Baclawski All Rights Reserved

Relational Model

- Youngest data model
- Dominates the database industry
- Rigorous theoretical foundation in mathematical logic
  - Most database systems diverge from the theory
- Not popular outside of database systems

Data Model Features

- Mathematical basis for the data structure: FOL
- Formats for storage and serialization: Hash tables and B-trees; no standard serialization or interchange format
- Schema languages: SQL
- Query and update languages: SQL
- Primary purpose: data storage and retrieval

© 2017 Ken Baclawski All Rights Reserved

The Hierarchical Model

Theoretical Basis

- Tree structure
  - In principle, a good theoretical foundation
  - In practice, most hierarchical languages add many features and differ from one another in significant ways
  - The additional features may not have any theoretical foundation
- XML is based on the *infoset* concept
- JSON has no underlying mathematical basis
  - One can serialize an infoset in JSON, but that does not give JSON a foundation

Infoset

- The formal model includes many aspects not included here, such as comments, entity references, and so on.
- The Document Object Model (DOM) is an implementation of infosets
  - But it departs from the theory significantly

© 2017 Ken Baclawski All Rights Reserved

Infosets

- An infoset model consists of
  - nodes (shown as rectangles or ovals)
  - relationship links (shown as arrows)
- The most common types of nodes
  - Element nodes
  - Text nodes
- Kinds of relationship link
  - Parent-child link
  - Attribute link

Infosets

- Every infoset model has a root node
- For an XML document, the root node has exactly one child node
- Infosets in general can have more than one child node of the root
  - Document fragments
  - Result of a query

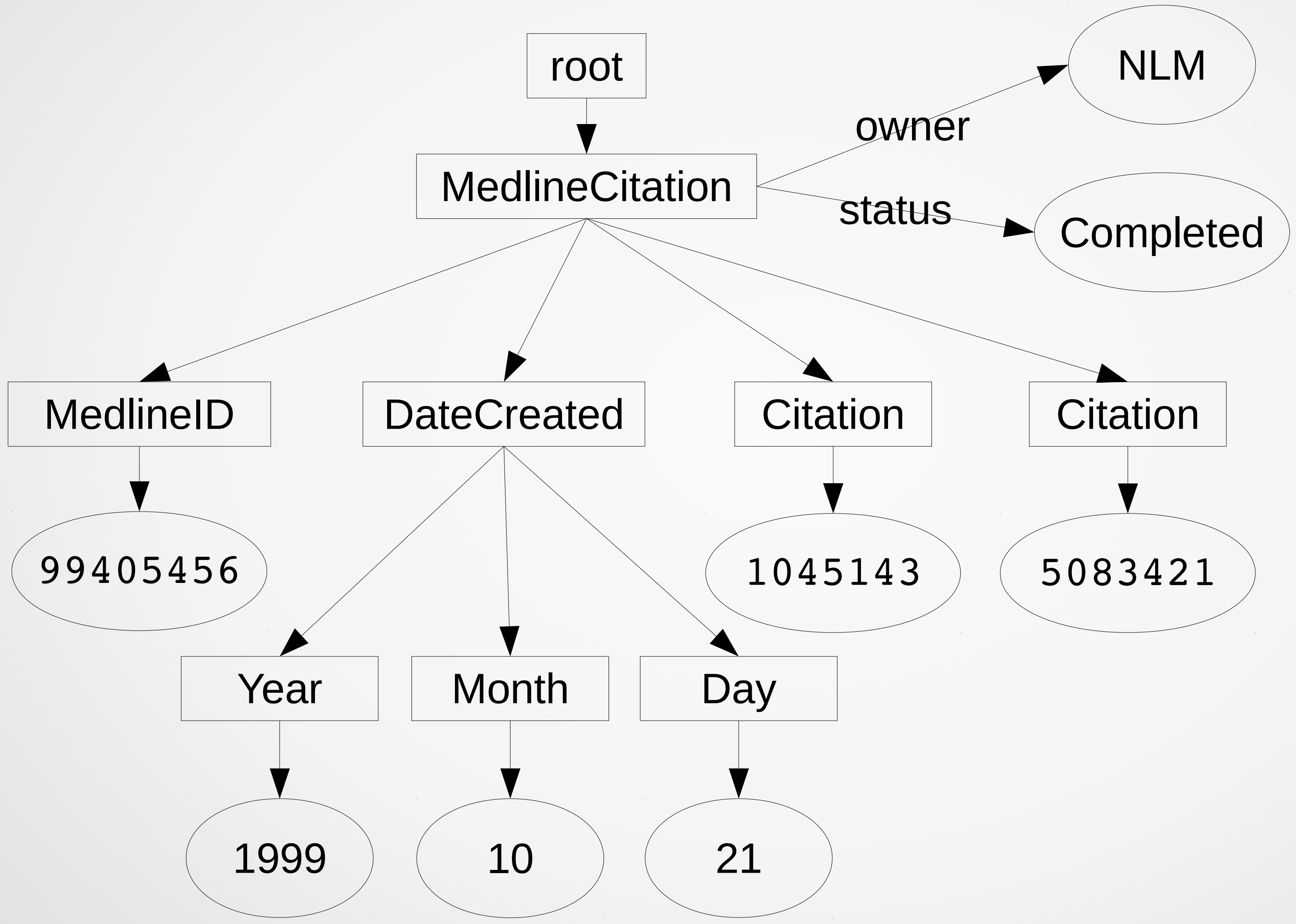
Example XML Document

- Here is a simple XML document
- The infoset is on the next slide

```
<MedlineCitation Owner="NLM" Status="Completed">
  <MedlineID>99405456</MedlineID>
  <DateCreated>
    <Year>1999</Year>
    <Month>10</Month>
    <Day>21</Day>
  </DateCreated>
  <Citation>1045143</Citation>
  <Citation>5083421</Citation>
</MedlineCitation>
```

© 2017 Ken Baclawski All Rights Reserved

Example Infoset



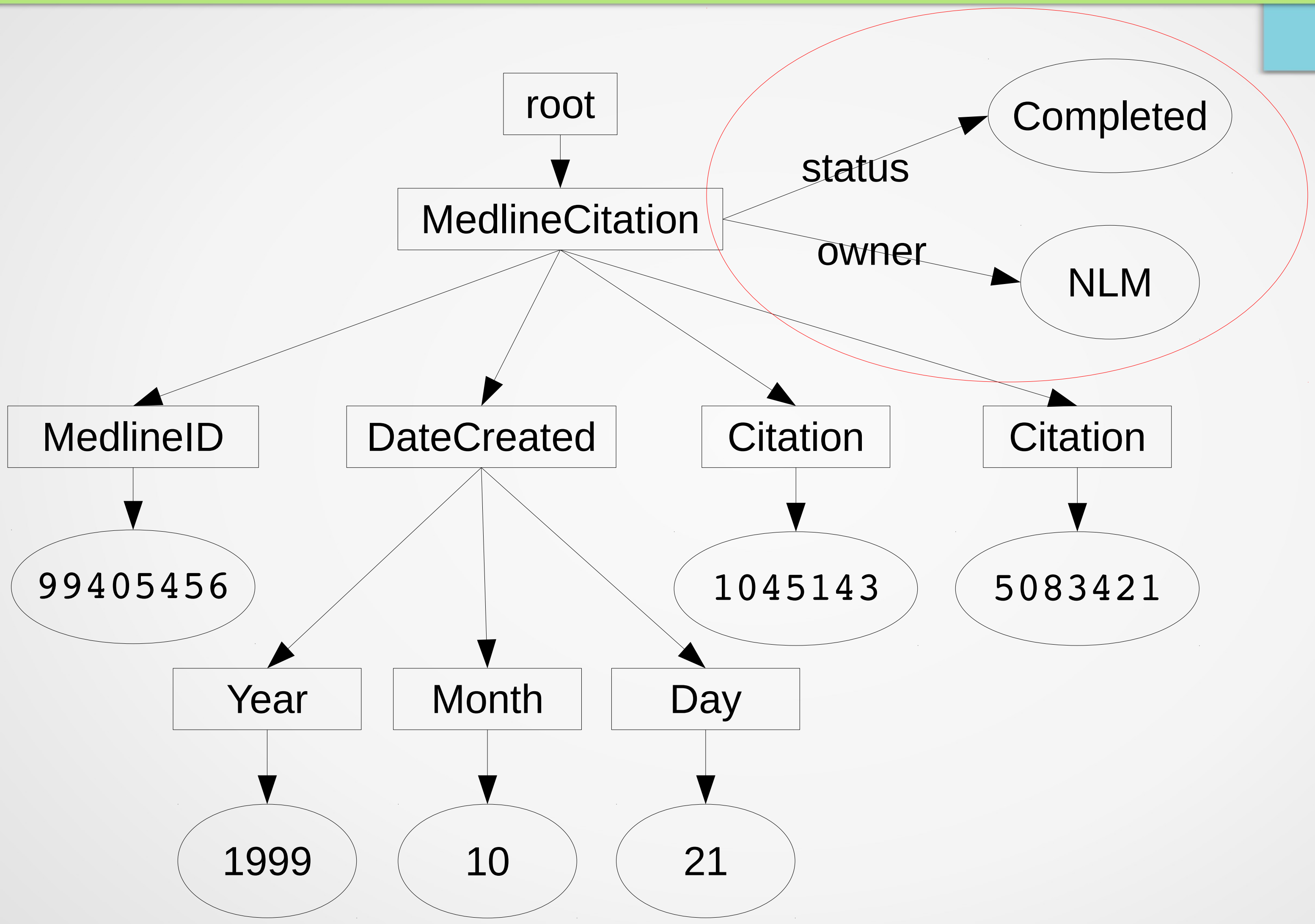
© 2017 Ken Baclawski All Rights Reserved

Meaning of an infoset

- To understand the meaning ask whether some alterations will change the meaning
- The next slide reverses the owner and status attributes
- Does this change the meaning?

© 2017 Ken Baclawski All Rights Reserved

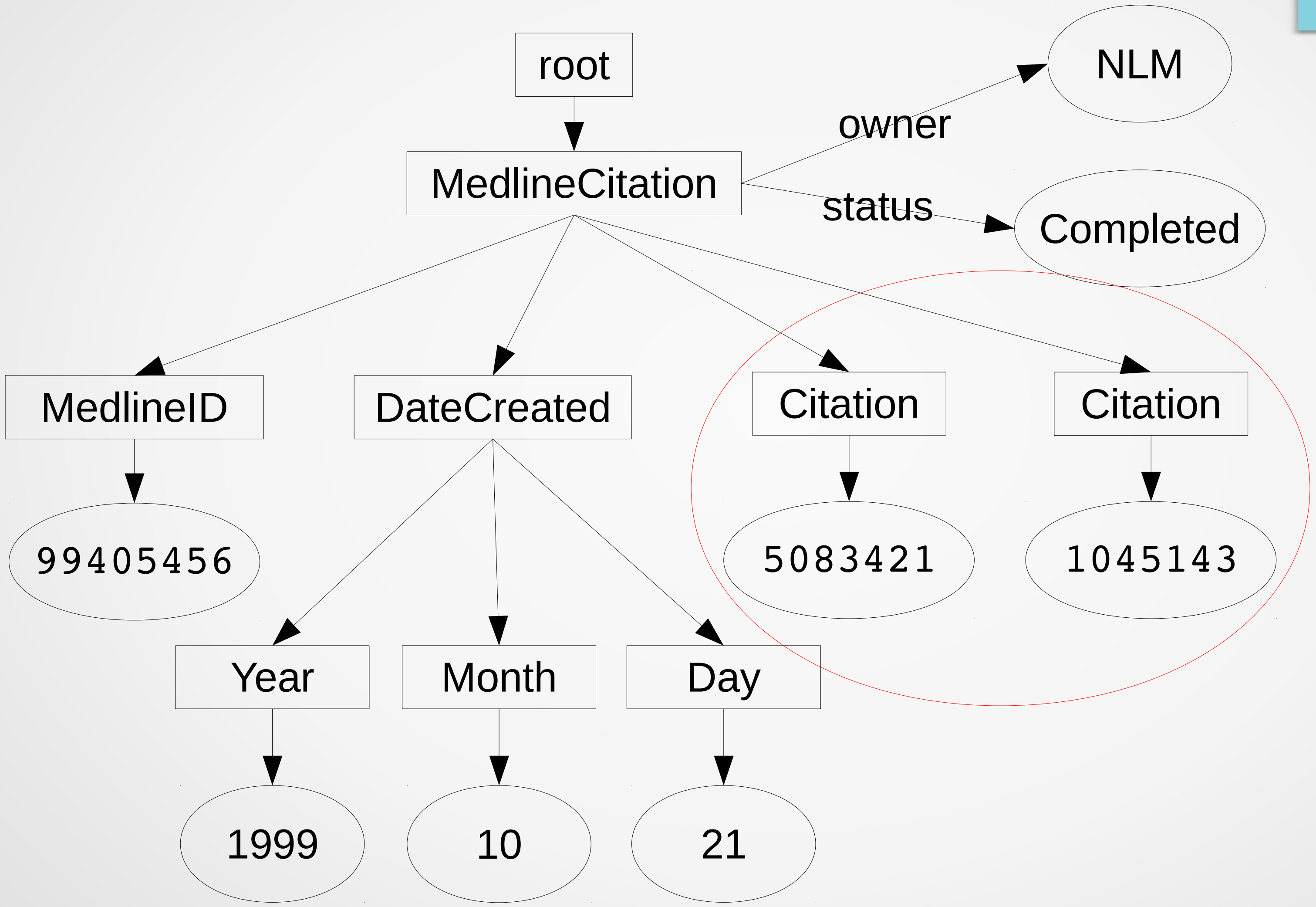
Example Infoset



Meaning of an infoset

- Reversing the order of the two attributes does **not** change the meaning
- This is consistent with the obvious meaning of the attributes of a MedlineCitation
- The order of element attributes is not significant
  - Note that DOM incorrectly keeps track of the order of element attributes
- The next slide reverses the last two child elements of MedlineCitation
- Does this change the meaning?

Example Infoset



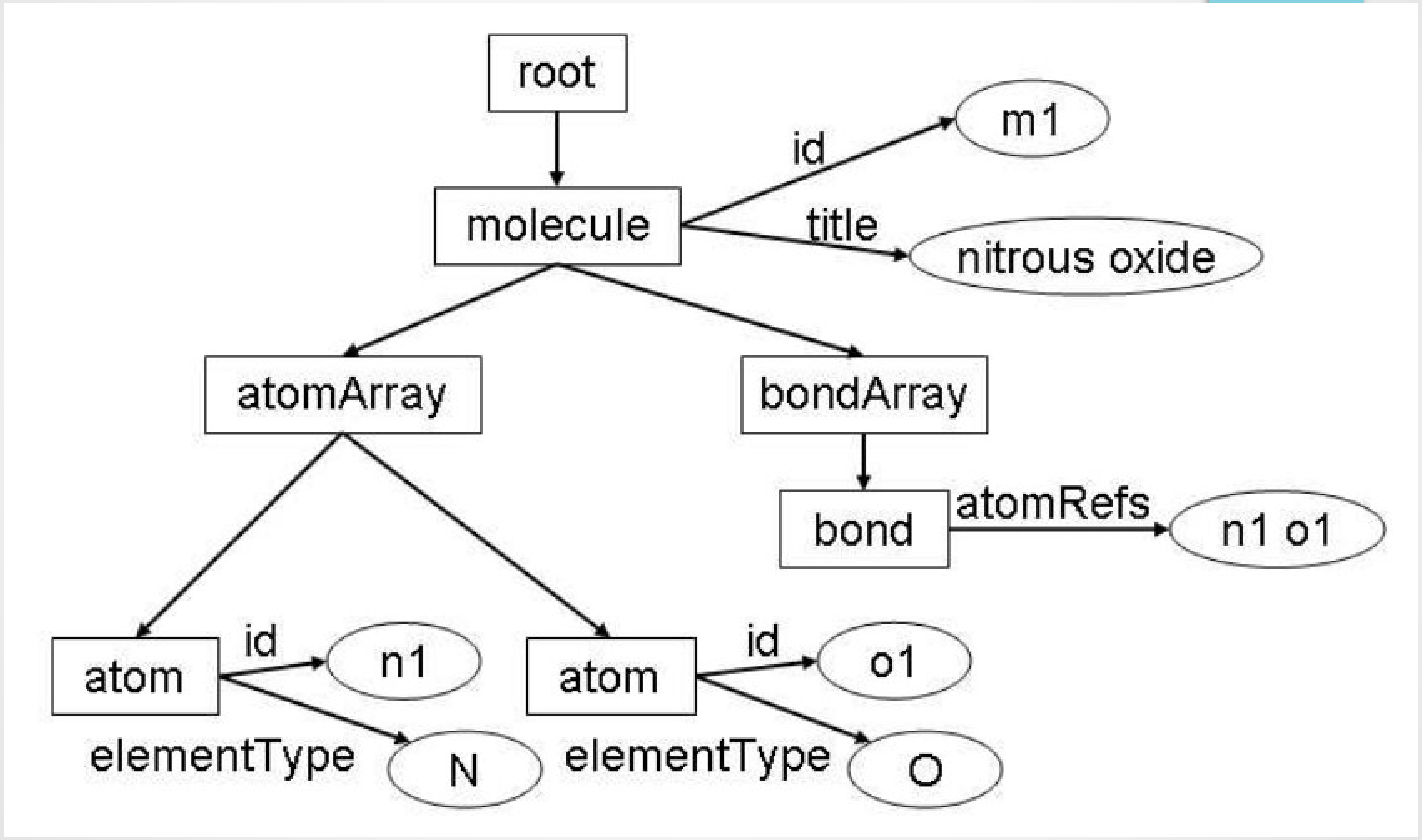
Meaning of an infoset

- Reversing the order of the two children **changes** the meaning of the XML document
- This is not consistent with the obvious notion of a Medline citation
- The order of child nodes is significant for XML
  - This reflects the origin of XML in documents
  - Important for web sites but not for most data
- XML has no mechanism for specifying that the order of child nodes is not significant

Limitations of pure hierarchies

- Hierarchies only have one relationship
  - Called parent-child or part of relationship
- Not sufficient for most kinds of data
- XML could easily represent relational data by using only XML attributes
  - Represents data
  - Does not represent uniqueness or foreign key constraints
- Consider the molecule represented on the next slide

Example Infoset



© 2017 Ken Baclawski All Rights Reserved

XML References

- The atomRefs attribute has a set of ids that are the atoms in the bond
- One cannot represent this using only parent-child relationships
  - The atoms in the bonds would be different atoms than the ones in the atom array
  - An atom could not be in two bonds

XML References

- XML has a notion of an id
  - Supposed to be unique within the document
  - Not usually enforced
- XML also has a notion of a reference to an id
  - Does not have any notion of a type
  - Cannot specify that a reference must be to only elements with certain names
  - Seldom used and seldom enforced

The DTD Schema Language

- There are several hierarchical schema definition languages even for XML
- Here is the Medline Citation schema in DTD format

```
<!ELEMENT MedlineCitation
  (MedlineID, DateCreated, Citation*)>
<!ATTLIST MedlineCitation
  Owner  CDATA                  "NLM"
  Status (Incomplete|Completed) #REQUIRED>
<!ELEMENT MedlineID (#PCDATA)>
<!ELEMENT DateCreated (Year, Month, Day)>
<!ELEMENT Year (#PCDATA)>
<!ELEMENT Month (#PCDATA)>
<!ELEMENT Day (#PCDATA)>
<!ELEMENT Citation (#PCDATA)>
```

© 2017 Ken Baclawski All Rights Reserved

XML Schema

- XML Schema is many more features than DTD
- One can convert from a DTD to XML Schema with the `dtd2xsd.pl` tool
- The next slides show the translations of the Medline Citation DTD
- In practice, the translation is only the first draft
  - Need to specify the namespaces
  - Need to adjust the constraints
- Unfortunately, it does not add very much to XML

XML Schema Example Part 1

```
<schema
  xmlns='http://www.w3.org/2001/XMLSchema'
  targetNamespace='http://www.w3.org/namespace/'
  xmlns:t='http://www.w3.org/namespace/'>

 <element name='MedlineCitation'>
  <complexType>
   <sequence>
    <element ref='t:MedlineID'/>
    <element ref='t:DateCreated'/>
    <element ref='t:Citation' minOccurs='0' maxOccurs='unbounded'/>
   </sequence>
   <attribute name='Owner' type='string' use='default' value='NLM'/>
   <attribute name='Status' use='required'>
    <simpleType>
     <restriction base='string'>
      <enumeration value='Incomplete'/>
      <enumeration value='Completed'/>
     </restriction>
    </simpleType>
   </attribute>
  </complexType>
 </element>
```

© 2017 Ken Baclawski All Rights Reserved

XML Schema Example Part 2

```
<element name='MedlineID'>
 <complexType mixed='true'>
 </complexType>
</element>

<element name='DateCreated'>
 <complexType>
  <sequence>
   <element ref='t:Year'/>
   <element ref='t:Month'/>
   <element ref='t:Day'/>
  </sequence>
 </complexType>
</element>
```

```
 <element name='Year'>
  <complexType mixed='true'>
  </complexType>
 </element>

 <element name='Month'>
  <complexType mixed='true'>
  </complexType>
 </element>

 <element name='Day'>
  <complexType mixed='true'>
  </complexType>
 </element>

 <element name='Citation'>
  <complexType mixed='true'>
  </complexType>
 </element>
</schema>
```

© 2017 Ken Baclawski All Rights Reserved

JSON Data

JavaScript Object Notation

- Increasingly popular data language
- Originally from JavaScript
- JSON is not as well supported as XML
  - Schema languages
  - Datatype definition and enforcement
  - Validation
  - Transformation tools
  - Query languages

Example JSON Expression

```
{
  'MedlineCitation': {
    'Owner': 'NLM',
    'Status': 'Completed',
    'MedlineID': 99405456,
    'DateCreated': {
      'Year': 1999,
      'Month': 10,
      'Day': 21
    },
    'Citation': [
      1045143,
      5083421
    ]
  }
}
```

This is intended to convey the same data as the XML document.

Hierarchical Query Languages

© 2017 Ken Baclawski All Rights Reserved

Hierarchical Query Languages

- There are very many hierarchical query languages
  - XQuery
  - XPath
  - XSLT (which uses XPath)
  - HAPL
  - Very many others see HAPL Documentation
- We will focus on XPath and HAPL

© 2017 Ken Baclawski All Rights Reserved

XPath

- A query looks like a file path
- It consists of a sequence of *location steps* separated by slashes
- A location step has three parts
  - Axis
  - Node test
  - Optional predicates
- The syntax is axis::node test[predicate][predicate]...
- An initial slash denotes navigation from the root

XPath Data Types

- XPath results are one of these:
  - Sequence of nodes
  - String
  - Double-precision number
  - Boolean
- HAPL has two additional types
  - Union of results (highest type)
  - Database null (lowest type)

© 2017 Ken Baclawski All Rights Reserved

XPath Axes

- ancestor
- ancestor-or-self
- attribute
- child (default)
- descendant
- descendant-or-self
- following
- following-sibling
- namespace
- parent
- preceding
- preceding-sibling
- self

© 2017 Ken Baclawski All Rights Reserved

XPath Abbreviations

- Child is the default and most common
- Descendant-or-self is “//”
- Parent is “..” but this is a whole location step not an axis
  - Not available in HAPL
- Self is “.” but this is a whole location step not an axis
  - Not available in HAPL
- Attribute is “@” which is a true abbreviation of an axis but once one is at at attribute one cannot have any child elements
- No other axes are available in HAPL

XPath Node Test

- Usually this is an element name
- It can also be a wildcard, either * or namespace:*
- Another possibility is to restrict to a type of node such as text() to constrain to a text node

XPath Predicates

- These are boolean expressions
- Within the expression one can use arithmetic and comparison operations
- There are many functions
- Most importantly, one can have XPath queries
- An expression in brackets will be converted to boolean if necessary
  - The most important case of such a conversion is converting the result of an XPath query
  - This is equivalent to an exists subquery

XPath Functions

- Node set (or actually sequence) functions
- String functions
- Number functions
- Logical (boolean) functions
- Implicit conversions
- For details, see HAPL Documentation

© 2017 Ken Baclawski All Rights Reserved

XPath Comparisons

- XPath equality and inequality comparisons can only be described as “bizarre”
- It is possible for two objects to be equal to each other but not to be less than or equal to each other!
- Each of the XPath data types has a sort order, but it is not always possible to test that two objects of the same type are in the same order
- See HAPL Documentation

© 2017 Ken Baclawski All Rights Reserved

XPath Conversions

- Normally, conversions are from a “higher” type to a “lower” type as shown in HAPL Documentation
- However, XPath violates this condition at times

© 2017 Ken Baclawski All Rights Reserved

HAPL Semantics

- HAPL is available with two different semantics
  - XPath semantics matches the XPath 1.0 specification
  - Javascript semantics fixes the peculiar features of XPath and is compatible with Javascript
    - Index numbering starts at 0 not 1
    - Comparisons are consistent with Javascript
    - Conversions are consistent with Javascript
- HAPL queries can be applied to either XML documents or JSON expressions.

© 2017 Ken Baclawski All Rights Reserved

Comparison of XML and JSON

© 2017 Ken Baclawski All Rights Reserved

Hierarchical Data Languages

- Differences between hierarchical data languages
- Nodes are labeled or unlabeled
- Parent-child relationship is labeled or unlabeled
- Children of a node are ordered or unordered

	XML	JSON
Nodes	named	unnamed
Parent-Child Relationship	unlabeled	labeled
Children of a Node	ordered	unordered
Additional Structure	labeled attributes	ordered arrays

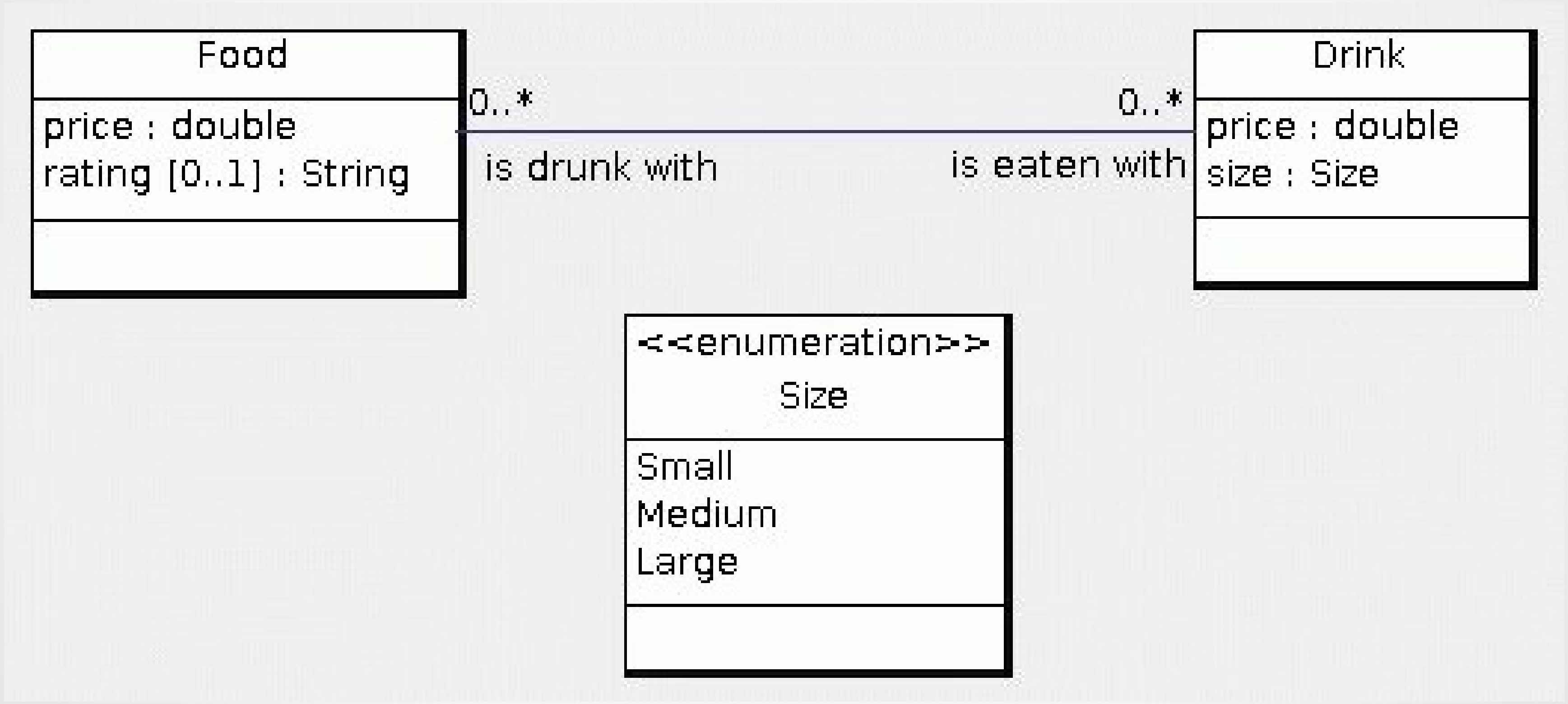
© 2017 Ken Baclawski All Rights Reserved

Restaurant Example

- UML Data Model
- SQL Schema
- XML DTD
- XML Schema
- Example data in XML
- Example data in JSON
- Example Query
- HAPL query for XML data
- HAPL query for JSON data

© 2017 Ken Baclawski All Rights Reserved

Restaurant Data Model



© 2017 Ken Baclawski All Rights Reserved

Restaurant SQL Schema

```
create table Food (
  id int primary key,
  price double not null check(price >= 0),
  rating varchar(31)
);
create table Drink (
  id int primary key,
  price double not null check(price >= 0),
  size enum('Small', 'Medium', 'Large') not null
);
create table Compatibility (
  isEatenWith int,
  isDrunkWith int,
  primary key(isEatenWith, isDrunkWith),
  foreign key(isEatenWith) references Drink(id)
    on update cascade on delete cascade,
  foreign key(isDrunkWith) references Food(id)
    on update cascade on delete cascade
);
```

© 2017 Ken Baclawski All Rights Reserved

Restaurant DTD

```
<!ELEMENT Restaurant (Food*,Drink*)>
<!ELEMENT Food (price,isEatenWith*)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT isEatenWith EMPTY>
<!ELEMENT Drink (#PCDATA)>
<!ATTLIST Food
        name        CDATA #REQUIRED
        rating      CDATA #IMPLIED>
<!ATTLIST Drink
        id          ID    #REQUIRED
        size        CDATA #REQUIRED
        price       CDATA #REQUIRED>
<!ATTLIST isEatenWith
        ref         IDREF #REQUIRED>
```

© 2017 Ken Baclawski All Rights Reserved

Restaurant XML Schema Part 1

```
<schema
  xmlns='http://www.w3.org/2001/XMLSchema'
  targetNamespace='http://kenb.ccs.neu.edu/restaurant/'
  xmlns:r='http://kenb.ccs.neu.edu/restaurant/'>

 <simpleType name='Size'>
  <restriction base='string'>
   <enumeration value='Small'/>
   <enumeration value='Medium'/>
   <enumeration value='Large'/>
  </restriction>
 </simpleType>

 <simpleType name="nonNegativeDouble">
  <restriction base="double">
   <minInclusive value="0"/>
  </restriction>
 </simpleType>
```

© 2017 Ken Baclawski All Rights Reserved

Restaurant XML Schema Part 2

```
<element name='Restaurant'>
 <complexType>
  <sequence>
   <element ref='r:Food' minOccurs='0' maxOccurs='unbounded'/>
   <element ref='r:Drink' minOccurs='0' maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>

<element name='Food'>
 <complexType>
  <sequence>
   <element ref='r:price'/>
   <element ref='r:isEatenWith' minOccurs='0' maxOccurs='unbounded'/>
  </sequence>
  <attribute name='name' type='string' use='required'/>
  <attribute name='rating' type='string' use='optional'/>
 </complexType>
</element>
```

Restaurant XML Schema Part 3

```
 <element name='price' type='r:nonNegativeDouble'/>

 <element name='isEatenWith'>
  <complexType>
   <attribute name='ref' type='IDREF' use='required'/>
  </complexType>
 </element>

 <element name='Drink'>
  <complexType mixed='true'>
   <attribute name='id' type='ID' use='required'/>
   <attribute name='size' type='r:Size' use='required'/>
   <attribute name='price' type='r:nonNegativeDouble' use='required'/>
  </complexType>
 </element>

</schema>
```

© 2017 Ken Baclawski All Rights Reserved

Restaurant XML Data

```
<Restaurant xmlns='http://kenb.ccs.neu.edu/restaurant/'>
  <Food name="Steak" rating="excellent">
    <price>20</price>
    <isEatenWith ref="d1"/>
    <isEatenWith ref="d3"/>
  </Food>
  <Food name="Pizza">
    <price>8.50</price>
    <isEatenWith ref="d1"/>
    <isEatenWith ref="d2"/>
  </Food>
  <Drink id="d1" size="Medium" price="1.25">Coca Cola</Drink>
  <Drink id="d2" size="Medium" price="1.30">Pepsi Cola</Drink>
  <Drink id="d3" size="Small" price="2.50">Coffee</Drink>
</Restaurant>
```

© 2017 Ken Baclawski All Rights Reserved

Restaurant JSON Data Part 1

```
{
  "Restaurant": {
    "Foods": [
      {
        "Food": {
          "name": "Steak",
          "rating": "excellent",
          "price": 20,
          "isEatenWith": [
            {
              "ref": "d1"
            },
            {
              "ref": "d3"
            }
          ]
        }
      },
```

```
      {
        "Food": {
          "name": "Pizza",
          "price": 8.50,
          "isEatenWith": [
            {
              "ref": "d1"
            },
            {
              "ref": "d2"
            }
          ]
        }
      }
    ],
```

© 2017 Ken Baclawski All Rights Reserved

Restaurant JSON Data Part 2

```
{
    "Drinks": [
      {
        "name": "Coca Cola",
        "price": 1.25,
        "id": "d1",
        "size": "Medium"
      },
      {
        "name": "Pepsi Cola",
        "price": 1.30,
        "id": "d2",
        "size": "Medium"
      },
```

```
      {
        "name": "Coffee",
        "price": 2.50,
        "id": "d3",
        "size": "Small"
      }
    ]
  }
}
```

Query on XML Data

- Compute the average price of a small drink
- The list of all drink elements is `//Drink`
- The size of a drink is `@size`
- To restrict to small drink elements, use `[@size='Small']`
- The price of a drink is `@price`
- Compute the `sum` and divide by the `count`
- The division operator is `div`

```
sum(//Drink[@size='Small']/@price) div
count(//Drink[@size='Small']/@price)
```

© 2017 Ken Baclawski All Rights Reserved

Query on JSON Data

- Compute the average price of a small drink
- The set of all drink arrays is `//Drinks`
- The elements of an array are obtained with a wildcard
- The size of a drink is `size`
- To restrict to small drink elements, use `[size='Small']`
- The price of a drink is `price`
- Compute the `sum` and divide by the `count`
- The division operator is `div`

```
sum(//Drinks/*[size='Small']/price) div
count(//Drinks/*[size='Small']/price)
```

Max Price Query

- Find the prices of foods that are good or excellent and eaten with drink 'd3'.
- Write your answer on a piece of paper.

Storage Devices

© 2017 Ken Baclawski All Rights Reserved

Data Storage

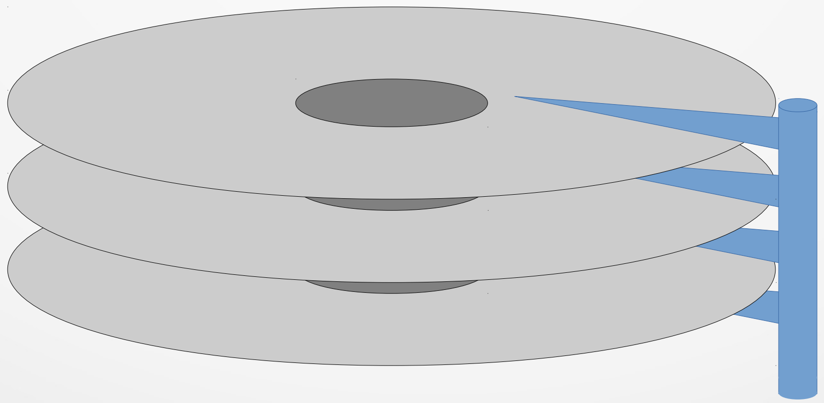
- Primary Storage
  - Primarily semiconductor
- Secondary Storage
  - Hard disk drives (HDD)
  - Solid state drives (SSD)
- Tertiary and Off-line storage
  - HDD
  - Tape drives

© 2017 Ken Baclawski All Rights Reserved

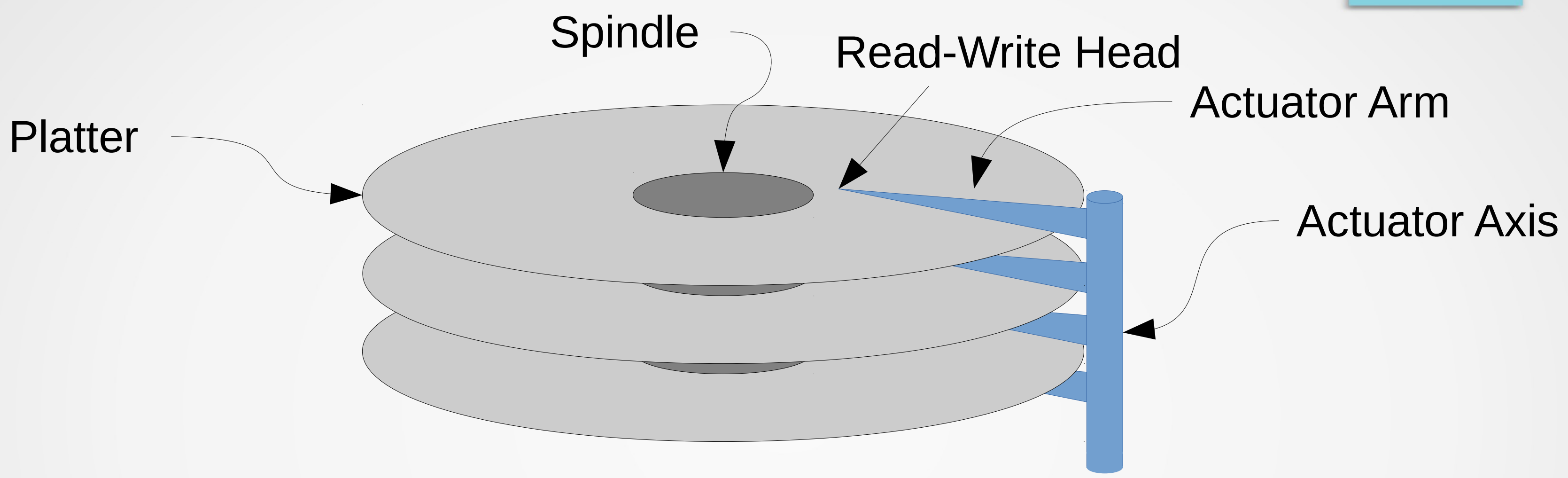
Hard Disk Drives

History

- Invented by IBM in 1956
- Current design introduced by IBM in 1961
- Impressive improvements in storage density, speed, reliability, cost, power requirements
  – No change in the basic design



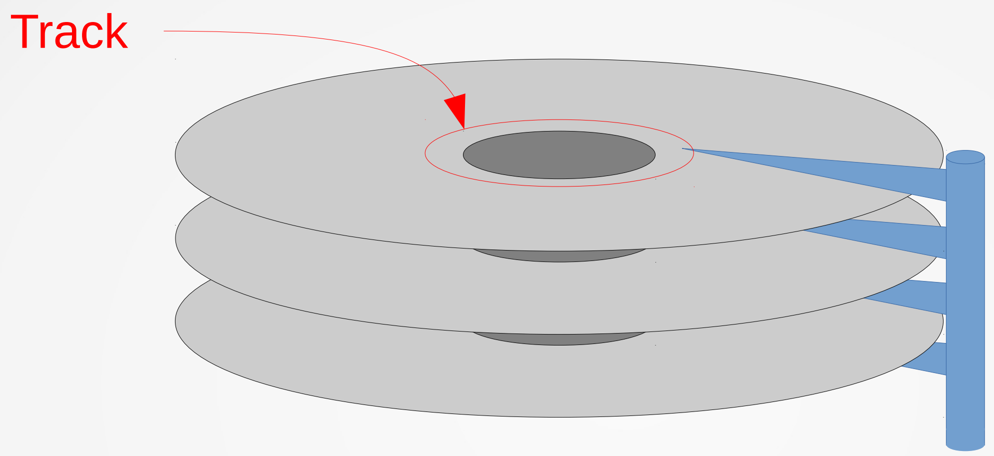
Hard Disk Drives



This HDD has 3 *platters* and 6 *surfaces* so there are 6 actuator arms (two of which face both up and down), each with a read-write head. The actuator moves the actuator arms by turning the actuator axis.   The actuator is not shown. The motor that spins the disks is under the spindle and is also not shown.

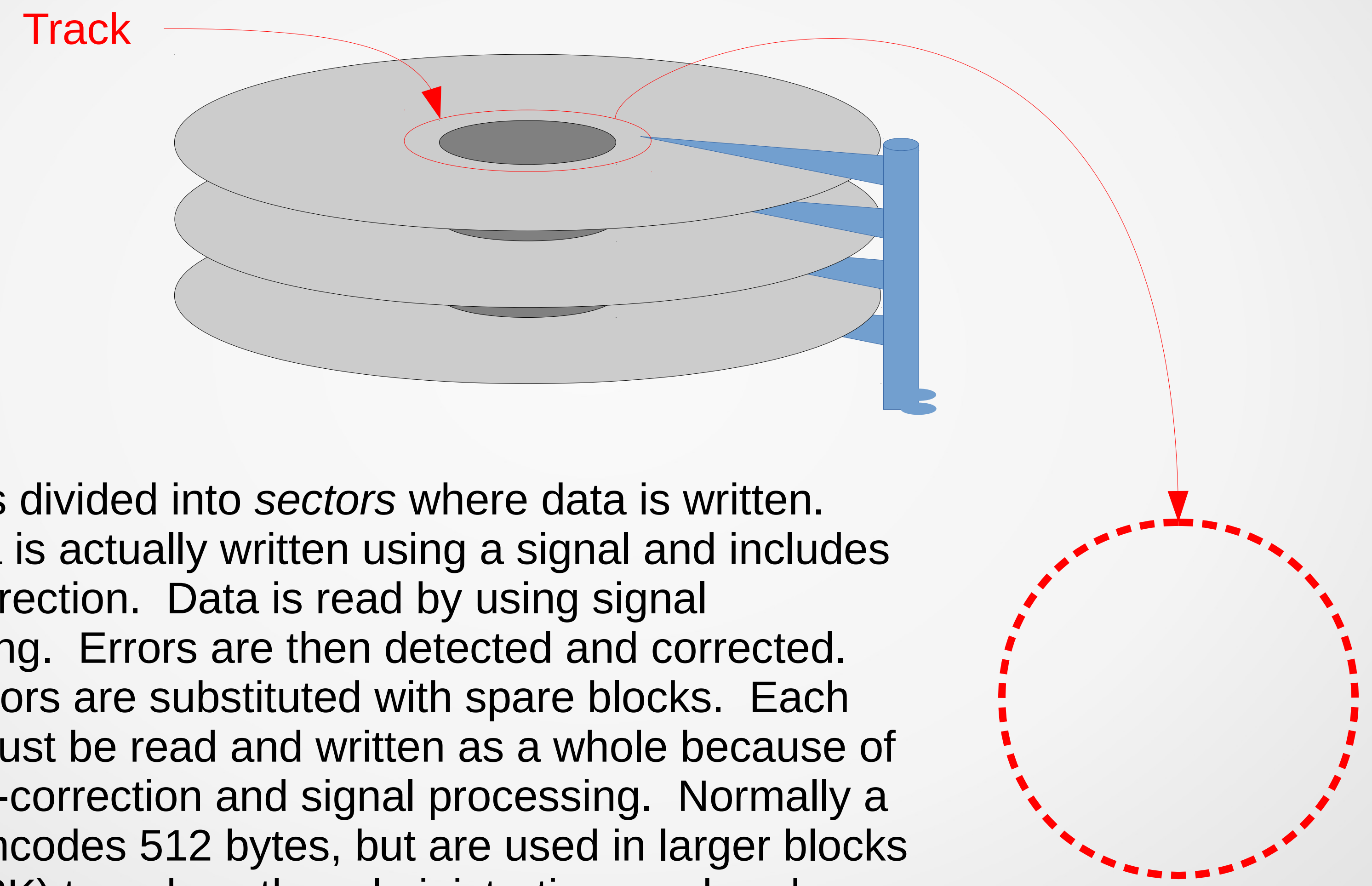
© 2017 Ken Baclawski All Rights Reserved

Hard Disk Drives



A *track* is the path traced by one read-write head on a single surface (side of one platter) while the actuator arm is in a single position.

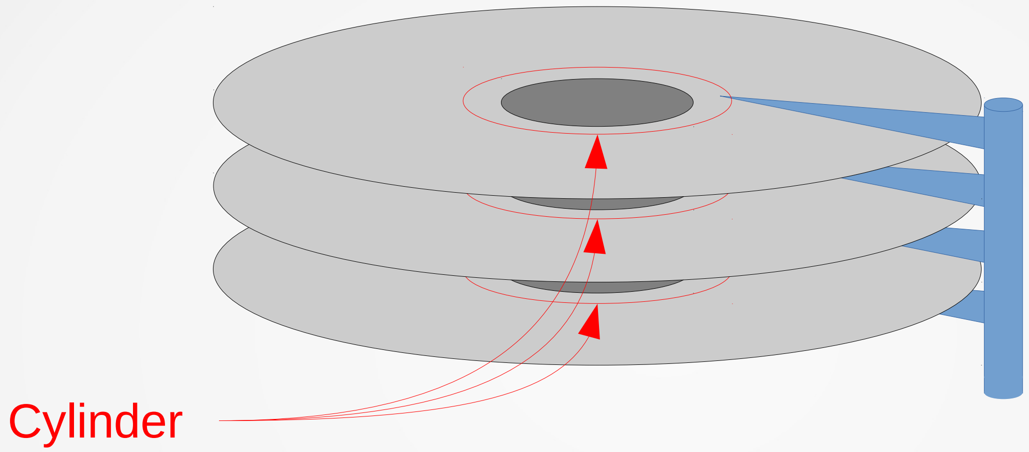
Hard Disk Drives



A track is divided into *sectors* where data is written. The data is actually written using a signal and includes error-correction. Data is read by using signal processing. Errors are then detected and corrected. Bad sectors are substituted with spare blocks. Each sector must be read and written as a whole because of the error-correction and signal processing. Normally a sector encodes 512 bytes, but are used in larger blocks (4K to 32K) to reduce the administrative overhead.

© 2017 Ken Baclawski All Rights Reserved

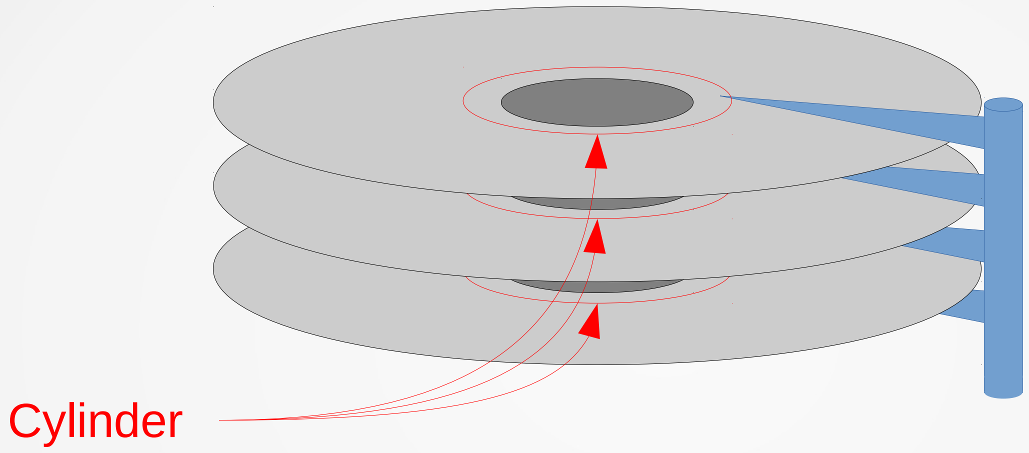
Hard Disk Drives



A *cylinder* is the set of all tracks traced by all of the read-write heads when the actuator arms are in a single position.

© 2017 Ken Baclawski All Rights Reserved

Hard Disk Drives



A *seek* is the movement of the actuator axis from one cylinder to another cylinder. The drive waits until the sector is under the read-write head. This is called the *latency period*. The *access time* is the sum of the seek time and latency.

© 2017 Ken Baclawski All Rights Reserved

Solid State Drives

Solid State Drives

- Semiconductor technology
- Not closely related to the technology used by processor cores and RAM
- Derived from Read-Only Memory
- SSD products have capacities almost comparable to HDD
- Costs about 8x HDD with same capacity
  - This factor is for commodity SSD
  - Other types of SSD are much more expensive

© 2017 Ken Baclawski All Rights Reserved

History of Read-Only Memory

- Fuses
  - Used as early as 1864
  - A thin wire that vaporizes if voltage is too high
  - Original purpose was to protect devices
- The normal (default) state of a fuse is conducting
  - Represents binary 1
- The other state of a fuse is “blown” or non-conducting
  - Represents binary 0

History of Read-Only Memory

- Read-Only Memory (ROM)
  - An addressable grid of wires called a *diode array*
  - The intersection of two wires is either connected or not
  - Connection pattern is fixed when manufactured
  - Used in earliest computers and still used today
- Programmable ROM (PROM)
  - Invented in 1956.
  - Connection pattern is programmed when installed
  - The connections are blown with a high-voltage pulse
  - Programming was called *burning*

History of Read-Only Memory

- Erasable PROM (EPROM)
  - Invented in 1971
  - Instead of cutting a wire, a capacitor (called a floating gate) holds a charge that changes the state
  - Connections are burned with a high-voltage pulse that charges the capacitor
  - Ultraviolet light causes the capacitors to discharge
  - This *erases* the EPROM
  - Erasing is slow and erases the entire EPROM
  - Erasing damages the EPROM slightly

History of Read-Only Memory

- Electrically Erasable PROM (EEPROM)
  - Invented in 1977
  - Erases with a special electric circuit
  - Exposure to ultraviolet light is no longer needed
  - Erasing damages the EEPROM
- Block EEPROM (Flash)
  - Invented in the 1980s (several variants)
  - Acronym was getting unwieldy
  - Same as EEPROM but erases in blocks
  - Greatly simplifies the circuitry

© 2017 Ken Baclawski All Rights Reserved

Characteristics of Flash Memory

- Most common technology is NAND
- Erasing is done in large blocks (16 KB to 512 KB)
  - Erased state is still binary 1
  - Relatively slow process
  - Damages the device slightly
- Reading and writing is done by pages (512 bytes to 4 KB)
  - Writing can only change bits from 1 to 0
  - Rewriting in place is not possible

© 2017 Ken Baclawski All Rights Reserved

Characteristics of Flash Memory

- Limitations
  - Gradual deterioration over time
  - Blocks fail due to damage from erasing
  - Commodity devices support around 1000 erasures
  - Complex wear-leveling algorithms are used
  - Bad blocks must be managed
  - Even reading has damaging effects
- Improvements
  - Flash technologies exist that deteriorate more slowly
  - Unfortunately, these devices are much more expensive

© 2017 Ken Baclawski All Rights Reserved

Comparison of SSD and HDD

- Advantages of SSD
  - Much faster random access time
  - Higher bandwidth in some cases
  - Lighter, more rugged, less noisy
  - Half to third the power requirement

- Disadvantages of SSD
  - More expensive
  - Deteriorates
    - Intensive use can lead to failure in a month or so
  - Some applications have lower performance
  - Erasing can be noisy
  - More susceptible to power outages and spikes
  - Lower long-term storage capability

Tape Drives

Tape Drives

- Still in use for off-line storage
- Capacities are high (185 TB tapes are now available)
- Access times are very slow
- Tape drives are expensive
- Tapes are only slightly cheaper than an HDD with the same capacity
- Competition from HDD devices is reducing the market for tape drives

Assignment #8

© 2017 Ken Baclawski All Rights Reserved