DSC 102
Systems for Scalable Analytics

Arun Kumar

Topic 1: Basics of Machine Resources
Part 1: Computer Organization

Ch. 1, 2.1-2.3, 2.12, 4.1, and 5.1-5.5 of CompOrg Book
Q: What is a computer?
Q: What is a computer?

A programmable electronic device that can store, retrieve, and process digital data.
Q: What is a computer?

A programmable electronic device that can store, retrieve, and process digital data.
Q: What is a computer?

A programmable electronic device that can store, retrieve, and process digital data.

Computer science aka "Datalogy"

Peter Naur
Outline

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy

❖ Basics of Operating Systems
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management

❖ Persistent Data Storage
Outline

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy

❖ Basics of Operating Systems
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management

❖ Persistent Data Storage
Parts of a Computer

Hardware:
The electronic machinery (wires, circuits, transistors, capacitors, devices, etc.)

Software:
Programs (instructions) and data

https://www.webopedia.com/TERM/C/computer.html
Key Parts of Computer Hardware
Key Parts of Computer Hardware

- **Processor** (CPU, GPU, etc.)
  - Hardware to orchestrate and execute *instructions* to manipulate *data* as specified by a *program*
Key Parts of Computer Hardware

❖ **Processor** (CPU, GPU, etc.)
  ❖ Hardware to orchestrate and execute *instructions* to manipulate *data* as specified by a *program*

❖ **Main Memory** (aka Dynamic Random Access Memory)
  ❖ Hardware to store *data* and *programs* that allows very fast location/retrieval; byte-level *addressing* scheme
Key Parts of Computer Hardware

❖ **Processor** (CPU, GPU, etc.)
  ❖ Hardware to orchestrate and execute *instructions* to manipulate *data* as specified by a *program*

❖ **Main Memory** (aka Dynamic Random Access Memory)
  ❖ Hardware to store *data* and *programs* that allows very fast location/retrieval; byte-level *addressing* scheme

❖ **Disk** (aka secondary/persistent storage)
  ❖ Similar to memory but *persistent, slower*, and higher capacity / cost ratio; various addressing schemes
Key Parts of Computer Hardware

❖ **Processor** (CPU, GPU, etc.)
  ❖ Hardware to orchestrate and execute *instructions* to manipulate *data* as specified by a *program*

❖ **Main Memory** (aka Dynamic Random Access Memory)
  ❖ Hardware to store *data* and *programs* that allows very fast location/retrieval; byte-level *addressing* scheme

❖ **Disk** (aka secondary/persistent storage)
  ❖ Similar to memory but *persistent, slower*, and higher capacity / cost ratio; various addressing schemes

❖ **Network** interface controller (NIC)
  ❖ Hardware to send data to / retrieve data over network of interconnected computers/devices
Abstract Computer Parts and Data

- Processor
  - Control Unit
  - Arithmetic & Logic Unit
  - Registers
  - Caches

- Dynamic Random Access Memory (DRAM)

- Bus

- Input Devices

- Output Devices

- Secondary Storage (e.g., Magnetic hard disk, Flash SSD, etc.)
Abstract Computer Parts and Data

Processor
- Control Unit
- Arithmetic & Logic Unit
- Registers
- Caches

Bus

Input Devices

Output Devices

Dynamic Random Access Memory (DRAM)

Secondary Storage (e.g., Magnetic hard disk, Flash SSD, etc.)

Store; Retrieve

Store; Retrieve
Abstract Computer Parts and Data

Processor
- Control Unit
- Arithmetic & Logic Unit
- Registers
- Caches

Dynamic Random Access Memory (DRAM)

Bus

Input Devices
Output Devices
Secondary Storage (e.g., Magnetic hard disk, Flash SSD, etc.)

Input; Output; Retrieve

Store; Retrieve

Store; Retrieve
Abstract Computer Parts and Data

Processor
- Control Unit
- Arithmetic & Logic Unit
- Registers
- Caches

Bus

Dynamic Random Access Memory (DRAM)

Secondary Storage (e.g., Magnetic hard disk, Flash SSD, etc.)

Input Devices

Output Devices

Input; Output; Retrieve

Retrieve; Process

Store; Retrieve

Store; Retrieve
Key Aspects of Software
Key Aspects of Software

❖ **Instruction**

❖ A command understood by hardware; finite vocabulary for a processor: Instruction Set Architecture (ISA); bridge between hardware and software
Key Aspects of Software

❖ **Instruction**
  ❖ A command understood by hardware; finite vocabulary for a processor: Instruction Set Architecture (ISA); bridge between hardware and software

❖ **Program (aka code)**
  ❖ A collection of instructions for hardware to execute
Key Aspects of Software

❖ **Instruction**
  ❖ A command understood by hardware; finite vocabulary for a processor: Instruction Set Architecture (ISA); bridge between hardware and software

❖ **Program** (aka code)
  ❖ A collection of instructions for hardware to execute

❖ **Programming Language (PL)**
  ❖ A human-readable *formal* language to write programs; at a much higher level of *abstraction* than ISA
Key Aspects of Software

❖ **Instruction**
  ❖ A command understood by hardware; finite vocabulary for a processor: Instruction Set Architecture (ISA); bridge between hardware and software

❖ **Program** (aka code)
  ❖ A collection of instructions for hardware to execute

❖ **Programming Language (PL)**
  ❖ A human-readable *formal* language to write programs; at a much higher level of *abstraction* than ISA

❖ **Application Programming Interface (API)**
  ❖ A set of functions (“interface”) exposed by a program/set of programs for use by humans/other programs
Key Aspects of Software

❖ **Instruction**
 ❖ A command understood by hardware; finite vocabulary for a processor: Instruction Set Architecture (ISA); bridge between hardware and software

❖ **Program** (aka code)
 ❖ A collection of instructions for hardware to execute

❖ **Programming Language (PL)**
 ❖ A human-readable *formal* language to write programs; at a much higher level of *abstraction* than ISA

❖ **Application Programming Interface (API)**
 ❖ A set of functions (“interface”) exposed by a program/set of programs for use by humans/other programs

❖ **Data**
 ❖ Digital representation of *information* that is stored, processed, displayed, retrieved, or sent by a program
Main Kinds of Software
Main Kinds of Software

❖ **Firmware**
  ❖ Read-only programs “baked into” a device to offer basic hardware control functionalities
Main Kinds of Software

❖ **Firmware**
  ❖ Read-only programs “baked into” a device to offer basic hardware control functionalities

❖ **Operating System (OS)**
  ❖ Collection of interrelated programs that work as an intermediary platform/service to enable application software to use hardware more effectively/easily
  ❖ Examples: Linux, Windows, MacOS, etc.
Main Kinds of Software

❖ Firmware
  ❖ Read-only programs “baked into” a device to offer basic hardware control functionalities

❖ Operating System (OS)
  ❖ Collection of interrelated programs that work as an intermediary platform/service to enable application software to use hardware more effectively/easily
  ❖ Examples: Linux, Windows, MacOS, etc.

❖ Application Software
  ❖ A program or a collection of interrelated programs to manipulate data, typically designed for human use
  ❖ Examples: Excel, Chrome, PostgreSQL, etc.
Outline

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy
❖ Basics of Operating Systems
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management
❖ Persistent Data Storage
Q: But why bother learning such low-level computer sciencey stuff in Data Science?
Luxury of “Statisticians”/“Analysts” of Yore

Luxury of “Statisticians”/“Analysts” of Yore

- **Methods:** Sufficed to learn just math/stats, maybe some SQL

[Links]
Luxury of “Statisticians”/“Analysts” of Yore

- **Methods:** Sufficed to learn just math/stats, maybe some SQL
- **Types:** Mostly tabular (relational), maybe some time series

[Links]
Luxury of “Statisticians”/“Analysts” of Yore

- **Methods**: Sufficed to learn just math/stats, maybe some SQL
- **Types**: Mostly tabular (relational), maybe some time series
- **Scale**: Mostly small (KBs to few GBs)

Luxury of “Statisticians”/“Analysts” of Yore

- **Methods**: Sufficed to learn just math/stats, maybe some SQL
- **Types**: Mostly tabular (relational), maybe some time series
- **Scale**: Mostly small (KBs to few GBs)
- **Tools**: Simple GUIs for both analysis and deployment; maybe an R-like console

Reality of Today’s “Data Scientists”

Data acquisition
Data preparation
Feature Engineering
Training & Inference
Model Selection
Serving
Monitoring

ML/AI + Data Systems Infrastructure
Reality of Today’s “Data Scientists”

Data acquisition
Data preparation
Feature Engineering
Training & Inference
Model Selection
Serving
Monitoring

ML/AI + Data Systems Infrastructure
Reality of Today’s “Data Scientists”

Data Scientist/ML Engineer

Data acquisition
Data preparation

Feature Engineering
Training & Inference
Model Selection

Serving
Monitoring

Source → Build → Deploy

ML/AI + Data Systems Infrastructure

python TensorFlow PyTorch DASK Spark AWS
Reality of Today’s “Data Scientists”

Data acquisition
Data preparation
Feature Engineering
Training & Inference
Model Selection
Serving
Monitoring
Reality of Today’s “Data Scientists”

Data acquisition
Data preparation
Feature Engineering
Training & Inference
Model Selection
Serving
Monitoring
Why bother with these in Data Science?

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy
❖ Basics of Operating Systems
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management
❖ Persistent Data Storage
Why bother with these in Data Science?

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy

❖ Basics of Operating Systems
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management

❖ Persistent Data Storage

You will face myriad and new data types
Why bother with these in Data Science?

- Basics of Computer Organization
  - Digital Representation of Data
  - Processors and Memory Hierarchy
- Basics of Operating Systems
  - Process Management: Virtualization; Concurrency
  - Filesystem and Data Files
  - Main Memory Management
- Persistent Data Storage

You will face myriad and new data types
Compute hardware is evolving fast
Why bother with these in Data Science?

- Basics of Computer Organization
  - Digital Representation of Data
  - Processors and Memory Hierarchy
- Basics of Operating Systems
  - Process Management: Virtualization; Concurrency
  - Filesystem and Data Files
  - Main Memory Management
- Persistent Data Storage

You will face myriad and new data types

Compute hardware is evolving fast

You will need to use new methods on evolving data file formats on clusters / cloud
Why bother with these in Data Science?

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy

❖ Basics of Operating Systems
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management

❖ Persistent Data Storage

You will face myriad and new data types

Compute hardware is evolving fast

You will need to use new methods on evolving data file formats on clusters / cloud

Storage hardware is evolving fast
How much does a Statistician make?

Average Base Pay

$88,989/yr

2,398 salaries

- Low: $61K
- Average: $89K
- High: $131K

Very High Confidence
Data Scientist Salaries United States

How much does a Data Scientist make?

Industry
- All industries

Employer Size
- All company sizes

Experience
- All years of Experience

To filter salaries for Data Scientist, Sign In or Register.

Very High Confidence

$117,212/yr

Average Base Pay
- $82K Low
- $117K Average
- $167K High

18,354 salaries
How much does a Data Scientist make?

Very High Confidence

$117,212/yr

Average Base Pay
18,354 salaries

- $82K Low
- $117K Average
- $167K High

$88,989
$28,223!
Outline

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy

❖ Basics of Operating Systems
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management

❖ Persistent Data Storage
Outline

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
    ❖ Processors and Memory Hierarchy
❖ Basics of Operating Systems
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management
❖ Persistent Data Storage
Q: What is data?
Digital Representation of Data
Digital Representation of Data

❖ **Bits:** All digital data are sequences of 0 & 1 (binary digits)
❖ Amenable to high-low/off-on electromagnetism
❖ Layers of *abstraction* to interpret bit sequences
Digital Representation of Data

- **Bits:** All digital data are sequences of 0 & 1 (binary digits)
- Amenable to high-low/off-on electromagnetism
- Layers of *abstraction* to interpret bit sequences

- **Data type:** First layer of abstraction to interpret a bit sequence with a human-understandable category of information; interpretation fixed by the PL
- Example common datatypes: Boolean, Byte, Integer, “floating point” number (Float), Character, and String
Digital Representation of Data

❖ **Bits**: All digital data are sequences of 0 & 1 (binary digits)
   ❖ Amenable to high-low/off-on electromagnetism
   ❖ Layers of *abstraction* to interpret bit sequences

❖ **Data type**: First layer of abstraction to interpret a bit sequence with a human-understandable category of information; interpretation fixed by the PL
   ❖ Example common datatypes: Boolean, Byte, Integer, “floating point” number (Float), Character, and String

❖ **Data structure**: A second layer of abstraction to *organize* multiple instances of same or varied data types as a more complex object with specified properties
   ❖ Examples: Array, Linked list, Tuple, Graph, etc.
Digital Representation of Data

Data Types in Python 3

- **None** (class `NoneType`)
- **Numbers**
  - **Integral**
    - **Integer** (class `int`)
    - **Booleans** (class `bool`)
  - **Real** (class `float`)
  - **Complex** (class `complex`)
- **Set types**
  - **Sets** (class `set`)
  - **Frozen sets** (class `frozenset`)
- **Sequences**
  - **Immutable**
    - **Strings** (class `str`)
    - **Tuples** (class `tuple`)
    - **Bytes** (class `bytes`)
  - **Mutable**
    - **Lists** (class `list`)
    - **Byte Arrays** (class `bytearray`)
- **Mappings**
  - **Dictionaries** (class `dict`)
- **Callable**
  - `< Functions, Methods, Classes >`
- **Modules**
Digital Representation of Data

- The size and *interpretation* of a data type depends on PL
- A **Byte** (B; 8 bits) is typically the basic unit of data types
Digital Representation of Data

- The size and *interpretation* of a data type depends on PL
- A **Byte** (B; 8 bits) is typically the basic unit of data types
- **Boolean:**
  - Examples in data sci.: Y/N or T/F responses
  - Just 1 bit needed but actual size is almost always 1B, i.e., 7 bits are wasted! *(Q: Why?)*
Digital Representation of Data

- The size and interpretation of a data type depends on PL

- A Byte (B; 8 bits) is typically the basic unit of data types

- **Boolean:**
  - Examples in data sci.: Y/N or T/F responses
  - Just 1 bit needed but actual size is almost always 1B, i.e., 7 bits are wasted! *(Q: Why?)*

- **Integer:**
  - Examples in data science: #friends, age, #likes
  - Typically 4 bytes; many variants (short, unsigned, etc.)
  - Java *int* can represent $-2^{31}$ to $(2^{31} - 1)$; C *unsigned int* can represent 0 to $(2^{32} - 1)$; Python3 *int* is effectively unlimited length (PL magic!)
Q: How many unique data items can be represented by 3 bytes?
Q: How many unique data items can be represented by 3 bytes?

- Given k bits, we can represent $2^k$ unique data items
- 3 bytes = 24 bits => $2^{24}$ items, i.e., 16,777,216 items
- Common approximation: $2^{10}$ (i.e., 1024) ~ $10^3$ (i.e., 1000); recall kibibyte (KiB) vs kilobyte (KB) and so on
Digital Representation of Data

Q: How many unique data items can be represented by 3 bytes?

❖ Given $k$ bits, we can represent $2^k$ unique data items
❖ $3$ bytes $= 24$ bits $\Rightarrow 2^{24}$ items, i.e., $16,777,216$ items
❖ Common approximation: $2^{10}$ (i.e., $1024$) $\sim 10^3$ (i.e., $1000$); recall kibibyte (KiB) vs kilobyte (KB) and so on

Q: How many bits are needed to distinguish $97$ data items?
Q: How many unique data items can be represented by 3 bytes?

- Given k bits, we can represent $2^k$ unique data items
- 3 bytes = 24 bits $\Rightarrow$ $2^{24}$ items, i.e., 16,777,216 items
- Common approximation: $2^{10}$ (i.e., 1024) $\sim$ $10^3$ (i.e., 1000); recall kibibyte (KiB) vs kilobyte (KB) and so on

Q: How many bits are needed to distinguish 97 data items?

- For k unique items, invert the exponent to get $\log_2(k)$
- But #bits is an integer! So, we only need $\lceil \log_2(k) \rceil$
- So, we only need the next higher power of 2
- 97 $\Rightarrow$ 128 = $2^7$; so, 7 bits
Q: How to convert from decimal to binary representation?
Q: How to convert from decimal to binary representation?

1. Given decimal n, if power of 2 (say, $2^k$), put 1 at bit position k; if $k=0$, stop; else pad with trailing 0s till position 0
Q: How to convert from decimal to binary representation?

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0

2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$
Q: How to convert from decimal to binary representation?

1. Given decimal n, if power of 2 (say, $2^k$), put 1 at bit position k; if k=0, stop; else pad with trailing 0s till position 0
2. If n is not power of 2, identify the power of 2 just below n (say, $2^k$); #bits is then k; put 1 at position k
3. Reset n as n - $2^k$; return to Steps 1-2
Q: How to convert from decimal to binary representation?

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0
2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$
3. Reset $n$ as $n - 2^k$; return to Steps 1-2
4. Fill remaining positions in between with 0s
Q: How to convert from decimal to binary representation?

1. Given decimal n, if power of 2 (say, $2^k$), put 1 at bit position k; if $k=0$, stop; else pad with trailing 0s till position 0

2. If n is not power of 2, identify the power of 2 just below n (say, $2^k$); #bits is then k; put 1 at position k

3. Reset n as n - $2^k$; return to Steps 1-2

4. Fill remaining positions in between with 0s
Q: How to convert from decimal to binary representation?

1. Given decimal \( n \), if power of 2 (say, \( 2^k \)), put 1 at bit position \( k \); if \( k=0 \), stop; else pad with trailing 0s till position 0

2. If \( n \) is not power of 2, identify the power of 2 just below \( n \) (say, \( 2^k \)); 
   #bits is then \( k \); put 1 at position \( k \)

3. Reset \( n \) as \( n - 2^k \); return to Steps 1-2

4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Position/Exponent of 2</th>
<th>Power of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5(_{10})</td>
<td>7 6 5 4 3 2 1 0</td>
<td>128 64 32 16 8 4 2 1</td>
</tr>
</tbody>
</table>

5 10
**Q: How to convert from decimal to binary representation?**

1. Given decimal \( n \), if power of 2 (say, \( 2^k \)), put 1 at bit position \( k \); if \( k=0 \), stop; else pad with trailing 0s till position 0
2. If \( n \) is not power of 2, identify the power of 2 just below \( n \) (say, \( 2^k \)); #bits is then \( k \); put 1 at position \( k \)
3. Reset \( n \) as \( n - 2^k \); return to Steps 1-2
4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>5(_{10})</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Position/Exponent of 2

Power of 2

1
Q: How to convert from decimal to binary representation?

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0
2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$
3. Reset $n$ as $n - 2^k$; return to Steps 1-2
4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Position/Exponent of 2

Power of 2
**Q: How to convert from decimal to binary representation?**

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0
2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$
3. Reset $n$ as $n - 2^k$; return to Steps 1-2
4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal $n_{10}$</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_{10}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_{10}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table: Position/Exponent of 2

<table>
<thead>
<tr>
<th>Decimal</th>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_{10}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Q: How to convert from decimal to binary representation?**

1. Given decimal n, if power of 2 (say, $2^k$), put 1 at bit position k; if $k=0$, stop; else pad with trailing 0s till position 0

2. If n is not power of 2, identify the power of 2 just below n (say, $2^k$); #bits is then k; put 1 at position k

3. Reset n as n - $2^k$; return to Steps 1-2

4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$47_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Q: *How to convert from decimal to binary representation?*

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0

2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$

3. Reset $n$ as $n - 2^k$; return to Steps 1-2

4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimall</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power of 2</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5_{10}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>47_{10}</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Q: How to convert from decimal to binary representation?

1. Given decimal n, if power of 2 (say, $2^k$), put 1 at bit position k; if k=0, stop; else pad with trailing 0s till position 0
2. If n is not power of 2, identify the power of 2 just below n (say, $2^k$); #bits is then k; put 1 at position k
3. Reset n as n - $2^k$; return to Steps 1-2
4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Position/Exponent of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>7</td>
</tr>
<tr>
<td>64</td>
<td>6</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Example:

- 5 in decimal: $5 = 2^1 + 2^0 = 101_2$
- 47 in decimal: $47 = 2^5 + 2^2 = 101111_2$
Q: How to convert from decimal to binary representation?

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0

2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$

3. Reset $n$ as $n - 2^k$; return to Steps 1-2

4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_{10}</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>47_{10}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power of 2</th>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_{10}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>47_{10}</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Q: How to convert from decimal to binary representation?

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0

2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$

3. Reset $n$ as $n - 2^k$; return to Steps 1-2

4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$47_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Position/Exponent of 2

Power of 2

128 64 32 16 8 4 2 1
**Q:** How to convert from decimal to binary representation?

1. Given decimal \( n \), if power of 2 (say, \( 2^k \)), put 1 at bit position \( k \); if \( k=0 \), stop; else pad with trailing 0s till position 0

2. If \( n \) is not power of 2, identify the power of 2 just below \( n \) (say, \( 2^k \)); #bits is then \( k \); put 1 at position \( k \)

3. Reset \( n \) as \( n - 2^k \); return to Steps 1-2

4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>5(_{10})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>47(_{10})</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Position/Exponent of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
</tr>
</tbody>
</table>

**Table:**

- Decimal values: 5\(_{10}\), 47\(_{10}\)
- Binary representations:
  - 5\(_{10}\): 101\(_{2}\)
  - 47\(_{10}\): 101111\(_{2}\)
Q: How to convert from decimal to binary representation?

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0.

2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$.

3. Reset $n$ as $n - 2^k$; return to Steps 1-2.

4. Fill remaining positions in between with 0s.

### Example

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Position/Exponent of 2</th>
<th>Power of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5_{10}$</td>
<td></td>
<td>$101_2$</td>
</tr>
<tr>
<td>$47_{10}$</td>
<td></td>
<td>$11111_2$</td>
</tr>
</tbody>
</table>
### Q: How to convert from decimal to binary representation?

1. Given decimal n, if power of 2 (say, $2^k$), put 1 at bit position k; if k=0, stop; else pad with trailing 0s till position 0
2. If n is not power of 2, identify the power of 2 just below n (say, $2^k$); #bits is then k; put 1 at position k
3. Reset n as n - $2^k$; return to Steps 1-2
4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$47_{10}$</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$163_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Q: How to convert from decimal to binary representation?

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0
2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$
3. Reset $n$ as $n - 2^k$; return to Steps 1-2
4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>$5_{10}$</th>
<th>$47_{10}$</th>
<th>$163_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power of 2</td>
<td>128</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td>Position/Exponent of 2</td>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>128</td>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>163</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
**Digital Representation of Data**

**Q: How to convert from decimal to binary representation?**

1. Given decimal n, if power of 2 (say, $2^k$), put 1 at bit position k; if $k=0$, stop; else pad with trailing 0s till position 0
2. If n is not power of 2, identify the power of 2 just below n (say, $2^k$); #bits is then k; put 1 at position k
3. Reset n as n - $2^k$; return to Steps 1-2
4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>5\textsubscript{10}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>47\textsubscript{10}</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>163\textsubscript{10}</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Q: How to convert from decimal to binary representation?**

1. Given decimal n, if power of 2 (say, \(2^k\)), put 1 at bit position k; if \(k=0\), stop; else pad with trailing 0s till position 0
2. If n is not power of 2, identify the power of 2 just below n (say, \(2^k\)); #bits is then k; put 1 at position k
3. Reset n as n - \(2^k\); return to Steps 1-2
4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Position/Exponent of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>(5_{10})</td>
<td>128 64 32 16 8 4 2 1</td>
</tr>
<tr>
<td>(47_{10})</td>
<td>101 0111</td>
</tr>
<tr>
<td>(163_{10})</td>
<td>111 1</td>
</tr>
</tbody>
</table>
**Digital Representation of Data**

**Q: How to convert from decimal to binary representation?**

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0

2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$

3. Reset $n$ as $n - 2^k$; return to Steps 1-2

4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$47_{10}$</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$163_{10}$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Position/Exponent of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Power of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
</tr>
</tbody>
</table>

Q: How to convert from decimal to binary representation?

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0

2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$

3. Reset $n$ as $n - 2^k$; return to Steps 1-2

4. Fill remaining positions in between with 0s
Digital Representation of Data

Q: *How to convert from decimal to binary representation?*

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0
2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$
3. Reset $n$ as $n - 2^k$; return to Steps 1-2
4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>163_{10}</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>47_{10}</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5_{10}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Position/Exponent of 2

128 64 32 16 8 4 2 1
Q: How to convert from decimal to binary representation?

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0
2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$
3. Reset $n$ as $n - 2^k$; return to Steps 1-2
4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Position/Exponent of 2

<table>
<thead>
<tr>
<th>Power of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Position/Exponent of 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_{10}</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>47_{10}</td>
<td>1 0 1 1 1</td>
</tr>
<tr>
<td>163_{10}</td>
<td>1 0 1 0 0 0 0 1 1</td>
</tr>
<tr>
<td>16_{10}</td>
<td>1 0 1 0 0 0 0 1 1</td>
</tr>
</tbody>
</table>
**Digital Representation of Data**

**Q: How to convert from decimal to binary representation?**

1. Given decimal $n$, if power of 2 (say, $2^k$), put 1 at bit position $k$; if $k=0$, stop; else pad with trailing 0s till position 0

2. If $n$ is not power of 2, identify the power of 2 just below $n$ (say, $2^k$); #bits is then $k$; put 1 at position $k$

3. Reset $n$ as $n - 2^k$; return to Steps 1-2

4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$47_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$163_{10}$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$16_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Digital Representation of Data

Q: How to convert from decimal to binary representation?

1. Given decimal \( n \), if power of 2 (say, \( 2^k \)), put 1 at bit position \( k \); if \( k=0 \), stop; else pad with trailing 0s till position 0

2. If \( n \) is not power of 2, identify the power of 2 just below \( n \) (say, \( 2^k \)); #bits is then \( k \); put 1 at position \( k \)

3. Reset \( n \) as \( n - 2^k \); return to Steps 1-2

4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>5(_{10})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>47(_{10})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>163(_{10})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>16(_{10})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Q: How to convert from decimal to binary representation?

1. Given decimal \( n \), if power of 2 (say, \( 2^k \)), put 1 at bit position \( k \); if \( k=0 \), stop; else pad with trailing 0s till position 0

2. If \( n \) is not power of 2, identify the power of 2 just below \( n \) (say, \( 2^k \)); #bits is then \( k \); put 1 at position \( k \)

3. Reset \( n \) as \( n - 2^k \); return to Steps 1-2

4. Fill remaining positions in between with 0s

<table>
<thead>
<tr>
<th>Decimal</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 5_{10} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( 47_{10} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( 163_{10} )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( 16_{10} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Q: Binary to decimal?
Hexadecimal representation is a common stand-in for binary representation; more succinct and readable

- Base 16 instead of base 2 cuts display length by ~4x
- Digits are 0, 1, ... 9, A (10₁₀), B, ... F (15₁₀)
- From binary: combine 4 bits at a time from lowest
Digital Representation of Data

- *Hexadecimal* representation is a common stand-in for binary representation; more succinct and readable
  - Base 16 instead of base 2 cuts display length by ~4x
  - Digits are 0, 1, ... 9, A (10\textsubscript{10}), B, ... F (15\textsubscript{10})
  - From binary: combine 4 bits at a time from lowest

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5\textsubscript{10}</td>
<td>101\textsubscript{2}</td>
<td>5</td>
</tr>
<tr>
<td>47\textsubscript{10}</td>
<td>10 1111\textsubscript{2}</td>
<td>2F</td>
</tr>
<tr>
<td>163\textsubscript{10}</td>
<td>1010 0011\textsubscript{2}</td>
<td>103</td>
</tr>
<tr>
<td>16\textsubscript{10}</td>
<td>1 0000\textsubscript{2}</td>
<td>10</td>
</tr>
</tbody>
</table>
Digital Representation of Data

- **Hexadecimal** representation is a common stand-in for binary representation; more succinct and readable
  - Base 16 instead of base 2 cuts display length by ~4x
  - Digits are 0, 1, ... 9, A (10\text{10}), B, ... F (15\text{10})
  - From binary: combine 4 bits at a time from lowest

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5\text{10}</td>
<td>101\text{2}</td>
<td>5\text{16}</td>
</tr>
<tr>
<td>47\text{10}</td>
<td>10 1111\text{2}</td>
<td></td>
</tr>
<tr>
<td>163\text{10}</td>
<td>1010 0011\text{2}</td>
<td></td>
</tr>
<tr>
<td>16\text{10}</td>
<td>1 0000\text{2}</td>
<td></td>
</tr>
</tbody>
</table>
Hexadecimal representation is a common stand-in for binary representation; more succinct and readable

- Base 16 instead of base 2 cuts display length by ~4x
- Digits are 0, 1, ... 9, A ($10_{10}$), B, ... F ($15_{10}$)
- From binary: combine 4 bits at a time from lowest

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5_{10}$</td>
<td>101$_2$</td>
<td>$5_{16}$</td>
</tr>
<tr>
<td>$47_{10}$</td>
<td>10 1111$_2$</td>
<td>$F_{16}$</td>
</tr>
<tr>
<td>$163_{10}$</td>
<td>1010 0011$_2$</td>
<td></td>
</tr>
<tr>
<td>$16_{10}$</td>
<td>1 0000$_2$</td>
<td></td>
</tr>
</tbody>
</table>
Digital Representation of Data

- *Hexadecimal* representation is a common stand-in for binary representation; more succinct and readable
  - Base 16 instead of base 2 cuts display length by ~4x
  - Digits are 0, 1, ... 9, A (10_{10}), B, ... F (15_{10})
  - From binary: combine 4 bits at a time from lowest

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_{10}</td>
<td>101_{2}</td>
<td>5_{16}</td>
</tr>
<tr>
<td>47_{10}</td>
<td>101111_{2}</td>
<td>2F_{16}</td>
</tr>
<tr>
<td>163_{10}</td>
<td>1010 0011_{2}</td>
<td></td>
</tr>
<tr>
<td>16_{10}</td>
<td>10000_{2}</td>
<td></td>
</tr>
</tbody>
</table>
Hexadecimal representation is a common stand-in for binary representation; more succinct and readable

- Base 16 instead of base 2 cuts display length by ~4x
- Digits are 0, 1, ... 9, A ($10_{10}$), B, ... F ($15_{10}$)
- From binary: combine 4 bits at a time from lowest

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5&lt;sub&gt;10&lt;/sub&gt;</td>
<td>101&lt;sub&gt;2&lt;/sub&gt;</td>
<td>5&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>47&lt;sub&gt;10&lt;/sub&gt;</td>
<td>10 1111&lt;sub&gt;2&lt;/sub&gt;</td>
<td>2 F&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>163&lt;sub&gt;10&lt;/sub&gt;</td>
<td>1010 0011&lt;sub&gt;2&lt;/sub&gt;</td>
<td>3&lt;sub&gt;16&lt;/sub&gt;</td>
</tr>
<tr>
<td>16&lt;sub&gt;10&lt;/sub&gt;</td>
<td>1 00000&lt;sub&gt;2&lt;/sub&gt;</td>
<td></td>
</tr>
</tbody>
</table>
**Digital Representation of Data**

- *Hexadecimal* representation is a common stand-in for binary representation; more succinct and readable
  - Base 16 instead of base 2 cuts display length by ~4x
  - Digits are 0, 1, ... 9, A \( (10_{10}) \), B, ... F \( (15_{10}) \)
  - From binary: combine 4 bits at a time from lowest

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5(_{10})</td>
<td>101(_{2})</td>
<td>5(_{16})</td>
</tr>
<tr>
<td>47(_{10})</td>
<td>10 1111(_{2})</td>
<td>2 F(_{16})</td>
</tr>
<tr>
<td>163(_{10})</td>
<td>1010 0011(_{2})</td>
<td>A3(_{16})</td>
</tr>
<tr>
<td>16(_{10})</td>
<td>1 00000(_{2})</td>
<td></td>
</tr>
</tbody>
</table>
Digital Representation of Data

- **Hexadecimal** representation is a common stand-in for binary representation; more succinct and readable
  - Base 16 instead of base 2 cuts display length by \(~4\times\)
  - Digits are 0, 1, ... 9, A (\(10_{10}\)), B, ... F (\(15_{10}\))
  - From binary: combine 4 bits at a time from lowest

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5(_{10})</td>
<td>101(_{2})</td>
<td>5(_{16})</td>
</tr>
<tr>
<td>47(_{10})</td>
<td>10 1111(_{2})</td>
<td>2 F(_{16})</td>
</tr>
<tr>
<td>163(_{10})</td>
<td>1010 0011(_{2})</td>
<td>A3(_{16})</td>
</tr>
<tr>
<td>16(_{10})</td>
<td>1 0000(_{2})</td>
<td>0(_{16})</td>
</tr>
</tbody>
</table>
Digital Representation of Data

- *Hexadecimal* representation is a common stand-in for binary representation; more succinct and readable
  - Base 16 instead of base 2 cuts display length by ~4x
  - Digits are 0, 1, ... 9, A (10₁₀), B, ... F (15₁₀)
  - From binary: combine 4 bits at a time from lowest

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5₁₀</td>
<td>101₂</td>
<td>5₁₆</td>
</tr>
<tr>
<td>47₁₀</td>
<td>10 1111₂</td>
<td>2 F₁₆</td>
</tr>
<tr>
<td>163₁₀</td>
<td>1010 0011₂</td>
<td>A3₁₆</td>
</tr>
<tr>
<td>16₁₀</td>
<td>1 0000₂</td>
<td>10₁₆</td>
</tr>
</tbody>
</table>
Digital Representation of Data

- *Hexadecimal* representation is a common stand-in for binary representation; more succinct and readable
- Base 16 instead of base 2 cuts display length by ~4x
- Digits are 0, 1, ..., 9, A (10\text{_{10}}), B, ..., F (15\text{_{10}})
- From binary: combine 4 bits at a time from lowest

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5\text{_{10}}</td>
<td>101\text{_{2}}</td>
<td>5\text{_{16}}</td>
</tr>
<tr>
<td>47\text{_{10}}</td>
<td>10 1111\text{_{2}}</td>
<td>2 F\text{_{16}}</td>
</tr>
<tr>
<td>163\text{_{10}}</td>
<td>1010 0011\text{_{2}}</td>
<td>A3\text{_{16}}</td>
</tr>
<tr>
<td>16\text{_{10}}</td>
<td>1 0000\text{_{2}}</td>
<td>10\text{_{16}}</td>
</tr>
</tbody>
</table>

Alternative notations: 0xA3 or A3\text{_{H}}
Digital Representation of Data

❖ **Float:**
  ❖ Examples in data sci.: salary, scores, model weights
  ❖ IEEE-754 single-precision format is 4B long; double-precision format is 8B long
  ❖ Java and C *float* is single; Python *float* is double!
Digital Representation of Data

❖ **Float:**
  ❖ Examples in data sci.: salary, scores, model weights
  ❖ IEEE-754 single-precision format is 4B long; double-precision format is 8B long
  ❖ Java and C *float* is single; Python *float* is double!

---

Using Automatic Mixed Precision for Major Deep Learning Frameworks
Digital Representation of Data

❖ **Float:**
❖ Standard IEEE format for single (aka binary32):
Digital Representation of Data

❖ **Float:**
❖ **Standard IEEE format for single (aka binary32):**

![Diagram showing the IEEE single-precision floating-point format](image)
Digital Representation of Data

- **Float:**
- **Standard IEEE format for single (aka binary32):**

\[ (-1)^{\text{sign}} \times 2^{\text{exponent} - 127} \times (1 + \sum_{i=1}^{23} b_{23-i}2^{-i}) \]
Digital Representation of Data

- **Float:**
  - Standard IEEE format for single (aka binary32):

\[
(-1)^{sign} \times 2^{exponent-127} \times (1 + \sum_{i=1}^{23} b_{23-i}2^{-i})
\]

\[
(-1)^0 \times 2^{124-127} \times (1 + 1 \cdot 2^{-2})
\]
Digital Representation of Data

❖ **Float:**
❖ Standard IEEE format for single (aka binary32):

\[
(\frac{-1}{2})^{\text{sign}} \times 2^{\text{exponent}-127} \times \left(1 + \sum_{i=1}^{23} b_{23-i}2^{-i}\right)
\]

\[
(\frac{-1}{2})^0 \times 2^{124-127} \times (1 + 1 \cdot 2^{-2}) = (1/8) \times (1 + (1/4)) = 0.15625
\]
Digital Representation of Data

❖ **Float:**

❖ **Standard IEEE format for single (aka binary32):**

\[
(-1)^{sign} \times 2^{exponent-127} \times (1 + \sum_{i=1}^{23} b_{23-i}2^{-i})
\]

\[
(-1)^0 \times 2^{124-127} \times (1 + 1 \cdot 2^{-2}) = (1/8) \times (1 + (1/4)) = 0.15625
\]

(NB: Converting decimal reals/fractions to float is NOT in syllabus!)
Due to representation imprecision issues, floating point arithmetic (addition and multiplication) is not associative!
Digital Representation of Data

- Due to representation imprecision issues, floating point arithmetic (addition and multiplication) is not associative!
Digital Representation of Data

- Due to representation imprecision issues, floating point arithmetic (addition and multiplication) is not associative!

  ```python
  (base) arun@Arun's-MacBook-Pro:~ $ python
  [Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
  Type "help", "copyright", "credits" or "license" for more information.
  >>> 0.1 + 0.2
  0.30000000000000004
  >>> (0.1 + 0.2) + 0.7
  1.0
  >>> 0.1 + (0.2 + 0.7)
  0.9999999999999999
  >>>
  ```

- In binary32, special encodings recognized:
  - Exponent 0xFF and fraction 0 is +/- “Infinity”
  - Exponent 0xFF and fraction <> 0 is “NaN”
  - Max is ~ 3.4 x 10^{38}; min +ve is ~ 1.4 x 10^{-45}
Digital Representation of Data
More float standards: double-precision (float64; 8B) and half-precision (float16; 2B); different #bits for exponent, fraction
Digital Representation of Data

- More float standards: double-precision (float64; 8B) and half-precision (float16; 2B); different #bits for exponent, fraction
- Float16 is now common for *deep learning* parameters:
  - Native support in PyTorch, TensorFlow, etc.; APIs also exist for weight quantization/rounding post training
  - NVIDIA Deep Learning SDK support mixed-precision training; 2-3x speedup with similar accuracy!

More float standards: double-precision (float64; 8B) and half-precision (float16; 2B); different #bits for exponent, fraction

Float16 is now common for deep learning parameters:
- Native support in PyTorch, TensorFlow, etc.; APIs also exist for weight quantization/rounding post training
- NVIDIA Deep Learning SDK support mixed-precision training; 2-3x speedup with similar accuracy!

New processor hardware (FPGAs, ASICs, etc.) enable arbitrary precision, even 1-bit (!), but accuracy is lower

Peer Instruction Activity

(Switch slides)
Digital Representation of Data

- Representing **Character (char)** and **String**:
  - Letters, numerals, punctuations, etc.
Digital Representation of Data

- Representing **Character (char)** and **String**:
  - Letters, numerals, punctuations, etc.
  - A string is typically just a variable-sized array of char
Digital Representation of Data

- Representing **Character (char)** and **String**:
  - Letters, numerals, punctuations, etc.
  - A string is typically just a variable-sized array of char
  - C `char` is 1B; Java char is 2B; Python does not have a char type (use `str` or `bytes`)


Digital Representation of Data

❖ Representing **Character (char)** and **String**:
  ❖ Letters, numerals, punctuations, etc.
  ❖ A string is typically just a variable-sized array of char
  ❖ C `char` is 1B; Java char is 2B; Python does not have a char type (use `str` or `bytes`)
  ❖ American Standard Code for Information Interchange (ASCII) for encoding characters; initially 7-bit; later extended to 8-bit
  ❖ Examples: ‘A’ is 61, ‘a’ is 97, ‘@’ is 64, ‘!’ is 33, etc.
Digital Representation of Data

❖ Representing **Character (char)** and **String**:
  ❖ Letters, numerals, punctuations, etc.
  ❖ A string is typically just a variable-sized array of char
  ❖ C `char` is 1B; Java char is 2B; Python does not have a char type (use `str` or `bytes`)
  ❖ American Standard Code for Information Interchange (**ASCII**) for encoding characters; initially 7-bit; later extended to 8-bit
    ❖ Examples: ‘A’ is 61, ‘a’ is 97, ‘@’ is 64, ‘!’ is 33, etc.
  ❖ **Unicode UTF-8** is now common, subsumes ASCII; 4B for ~1.1 million “code points” incl. many other language scripts, math symbols, emojis, etc. 😊
Digital Representation of Data
Digital Representation of Data

- All digital objects are *collections* of basic data types (bytes, integers, floats, and characters)
Digital Representation of Data

- All digital objects are *collections* of basic data types (bytes, integers, floats, and characters)
- SQL dates/timestamp: string (w/ known format)
Digital Representation of Data

- All digital objects are *collections* of basic data types (bytes, integers, floats, and characters)
- SQL dates/timestamp: string (w/ known format)
- ML feature vector: *array* of floats (w/ known length)
Digital Representation of Data

❖ All digital objects are collections of basic data types (bytes, integers, floats, and characters)
❖ SQL dates/timestamp: string (w/ known format)
❖ ML feature vector: array of floats (w/ known length)
❖ Neural network weights: set of multi-dimensional arrays (matrices or tensors) of floats (w/ known dimensions)
Digital Representation of Data

❖ All digital objects are *collections* of basic data types (bytes, integers, floats, and characters)
  ❖ SQL dates/timestamp: string (w/ known format)
  ❖ ML feature vector: *array* of floats (w/ known length)
  ❖ Neural network weights: *set* of multi-dimensional *arrays* (matrices or tensors) of floats (w/ known dimensions)
  ❖ Graph: an *abstract data type* (ADT) with *set* of vertices (say, integers) and *set* of edges (*pair* of integers)
Digital Representation of Data

- All digital objects are *collections* of basic data types (bytes, integers, floats, and characters)
  - SQL dates/timestamp: string (w/ known format)
  - ML feature vector: *array* of floats (w/ known length)
  - Neural network weights: *set* of multi-dimensional *arrays* (matrices or tensors) of floats (w/ known dimensions)
  - Graph: an *abstract data type* (ADT) with *set* of vertices (say, integers) and *set* of edges (*pair* of integers)
  - Program in PL, SQL query: string (w/ grammar)
Digital Representation of Data

- All digital objects are *collections* of basic data types (bytes, integers, floats, and characters)
  - SQL dates/timestamp: string (w/ known format)
  - ML feature vector: *array* of floats (w/ known length)
  - Neural network weights: *set* of multi-dimensional *arrays* (matrices or tensors) of floats (w/ known dimensions)
  - Graph: an *abstract data type* (ADT) with *set* of vertices (say, integers) and *set* of edges (*pair* of integers)
  - Program in PL, SQL query: string (w/ grammar)
  - Other data structures or digital objects?
Digital Representation of Data
Digital Representation of Data

❖ **Serialization and Deserialization:**
❖ A data structure often needs to be persisted (stored in a file) or transmitted over a network
Digital Representation of Data

- **Serialization and Deserialization:**
  - A data structure often needs to be persisted (stored in a file) or transmitted over a network.
  - Serialization is the process of converting a data structure (or program objects in general) into a neat sequence of bytes that can be exactly recovered; deserialization is the reverse, i.e., bytes to data structure.
Digital Representation of Data

- **Serialization and Deserialization:**
  - A data structure often needs to be persisted (stored in a file) or transmitted over a network.
  - Serialization is the process of converting a data structure (or program objects in general) into a neat sequence of bytes that can be exactly recovered; deserialization is the reverse, i.e., bytes to data structure.
  - Serializing bytes and characters/strings is trivial.
Digital Representation of Data

- **Serialization and Deserialization:**
  - A data structure often needs to be persisted (stored in a file) or transmitted over a network.
  - Serialization is the process of converting a data structure (or program objects in general) into a neat sequence of bytes that can be exactly recovered; deserialization is the reverse, i.e., bytes to data structure.
  - Serializing bytes and characters/strings is trivial.
  - 2 alternatives for serializing integers/floats:
Digital Representation of Data

- **Serialization and Deserialization:**
  - A data structure often needs to be persisted (stored in a file) or transmitted over a network.
  - Serialization is the process of converting a data structure (or program objects in general) into a neat sequence of bytes that can be exactly recovered; deserialization is the reverse, i.e., bytes to data structure.
  - Serializing bytes and characters/strings is trivial.
  - 2 alternatives for serializing integers/floats:
    - As *byte stream* (aka “binary type” in SQL)
Digital Representation of Data

- **Serialization and Deserialization:**
  - A data structure often needs to be persisted (stored in a file) or transmitted over a network.
  - Serialization is the process of converting a data structure (or program objects in general) into a neat sequence of bytes that can be exactly recovered; deserialization is the reverse, i.e., bytes to data structure.
  - Serializing bytes and characters/strings is trivial.
  - 2 alternatives for serializing integers/floats:
    - As *byte stream* (aka “binary type” in SQL)
    - As *string*, e.g., 4B integer 5 -> 2B string as “5”
Digital Representation of Data

- **Serialization and Deserialization:**
  - A data structure often needs to be persisted (stored in a file) or transmitted over a network.
  - Serialization is the process of converting a data structure (or program objects in general) into a neat sequence of bytes that can be exactly recovered; deserialization is the reverse, i.e., bytes to data structure.
  - Serializing bytes and characters/strings is trivial.
  - 2 alternatives for serializing integers/floats:
    - As *byte stream* (aka “binary type” in SQL).
    - As *string*, e.g., 4B integer 5 -> 2B string as “5”.
    - String ser. common in data science (CSV, TSV, etc.)
Review Questions
Review Questions

❖ What is the difference between data and code?
Review Questions

❖ What is the difference between data and code?
❖ What kind of software is TensorFlow? Linux?
Review Questions

❖ What is the difference between data and code?
❖ What kind of software is TensorFlow? Linux?
❖ Why do computers use binary digits?
Review Questions

❖ What is the difference between data and code?
❖ What kind of software is TensorFlow? Linux?
❖ Why do computers use binary digits?
❖ How many integers can you represent with 5 bits?
Review Questions

❖ What is the difference between data and code?
❖ What kind of software is TensorFlow? Linux?
❖ Why do computers use binary digits?
❖ How many integers can you represent with 5 bits?
❖ How many bits do you need to represent 5 integers?
Review Questions

❖ What is the difference between data and code?
❖ What kind of software is TensorFlow? Linux?
❖ Why do computers use binary digits?
❖ How many integers can you represent with 5 bits?
❖ How many bits do you need to represent 5 integers?
❖ What is the hexadecimal for $20_{10}$?
Review Questions

- What is the difference between data and code?
- What kind of software is TensorFlow? Linux?
- Why do computers use binary digits?
- How many integers can you represent with 5 bits?
- How many bits do you need to represent 5 integers?
- What is the hexadecimal for $20_{10}$?
- Why do we need a float standard?
Review Questions

❖ What is the difference between data and code?
❖ What kind of software is TensorFlow? Linux?
❖ Why do computers use binary digits?
❖ How many integers can you represent with 5 bits?
❖ How many bits do you need to represent 5 integers?
❖ What is the hexadecimal for \(20_{10}\)?
❖ Why do we need a float standard?
❖ Why should a data scientist know about float formats?
Review Questions

❖ What is the difference between data and code?
❖ What kind of software is TensorFlow? Linux?
❖ Why do computers use binary digits?
❖ How many integers can you represent with 5 bits?
❖ How many bits do you need to represent 5 integers?
❖ What is the hexadecimal for $20_{10}$?
❖ Why do we need a float standard?
❖ Why should a data scientist know about float formats?
❖ What does “lower precision” mean for a float weight in DL?
Review Questions

❖ What is the difference between data and code?
❖ What kind of software is TensorFlow? Linux?
❖ Why do computers use binary digits?
❖ How many integers can you represent with 5 bits?
❖ How many bits do you need to represent 5 integers?
❖ What is the hexadecimal for $20_{10}$?
❖ Why do we need a float standard?
❖ Why should a data scientist know about float formats?
❖ What does “lower precision” mean for a float weight in DL?
❖ Why is serialization needed?
Review Questions

❖ What is the difference between data and code?
❖ What kind of software is TensorFlow? Linux?
❖ Why do computers use binary digits?
❖ How many integers can you represent with 5 bits?
❖ How many bits do you need to represent 5 integers?
❖ What is the hexadecimal for $20_{10}$?
❖ Why do we need a float standard?
❖ Why should a data scientist know about float formats?
❖ What does “lower precision” mean for a float weight in DL?
❖ Why is serialization needed?
❖ Is code a string? Is a string code?
Review Questions

❖ What is the difference between data and code?
❖ What kind of software is TensorFlow? Linux?
❖ Why do computers use binary digits?
❖ How many integers can you represent with 5 bits?
❖ How many bits do you need to represent 5 integers?
❖ What is the hexadecimal for $20_{10}$?
❖ Why do we need a float standard?
❖ Why should a data scientist know about float formats?
❖ What does “lower precision” mean for a float weight in DL?
❖ Why is serialization needed?
❖ Is code a string? Is a string code?
❖ Is reality a computer simulation? :)
Peer Instruction Activity

(Switch slides)
Outline

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy
❖ Basics of Operating Systems
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management
❖ Persistent Data Storage
Outline

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy
❖ Basics of Operating Systems
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management
❖ Persistent Data Storage
Basics of Processors
Basics of Processors

❖ **Processor**: Hardware to orchestrate and *execute instructions* to *manipulate data* as specified by a program
❖ Examples: CPU, GPU, FPGA, TPU, embedded, etc.
Basics of Processors

❖ **Processor**: Hardware to orchestrate and *execute instructions* to *manipulate data* as specified by a program
❖ Examples: CPU, GPU, FPGA, TPU, embedded, etc.
❖ **ISA**: The vocabulary of commands of a processor
Basics of Processors

- **Processor**: Hardware to orchestrate and *execute instructions* to *manipulate data* as specified by a program
  - Examples: CPU, GPU, FPGA, TPU, embedded, etc.
- **ISA**: The vocabulary of commands of a processor

Program in PL
Basics of Processors

❖ **Processor**: Hardware to orchestrate and *execute instructions* to *manipulate data* as specified by a program

❖ Examples: CPU, GPU, FPGA, TPU, embedded, etc.

❖ **ISA**: The vocabulary of commands of a processor

Program in PL

**Compile/Interpret**

Program in Assembly Language
Basics of Processors

❖ **Processor**: Hardware to orchestrate and *execute instructions* to *manipulate data* as specified by a program

❖ Examples: CPU, GPU, FPGA, TPU, embedded, etc.

❖ **ISA**: The vocabulary of commands of a processor

```
Program in PL
Compile/Interpret
Program in Assembly Language
Assemble
Machine code tied to ISA
```
Basics of Processors

- **Processor**: Hardware to orchestrate and *execute instructions* to *manipulate data* as specified by a program
- **Examples**: CPU, GPU, FPGA, TPU, embedded, etc.
- **ISA**: The vocabulary of commands of a processor

Program in PL

Compile/Interpret

Program in Assembly Language

*Assemble*

Machine code tied to ISA

*Run on processor*
Basics of Processors

- **Processor**: Hardware to orchestrate and *execute instructions* to *manipulate data* as specified by a program
  - Examples: CPU, GPU, FPGA, TPU, embedded, etc.
- **ISA**: The vocabulary of commands of a processor

```
Program in PL
Compile/Interpret
Program in Assembly Language
Assemble
Machine code tied to ISA
Run on processor
```
Abstract Computer Parts and Data

- Processor
  - Control Unit
  - Arithmetic & Logic Unit
  - Registers
  - Caches

Bus

- Input Devices
- Output Devices
- Secondary Storage (e.g., Magnetic hard disk, Flash SSD, etc.)

Retrieve; Process

Store; Retrieve

Dynamic Random Access Memory (DRAM)
Abstract Computer Parts and Data

- Processor
  - Control Unit
  - Arithmetic & Logic Unit
  - Registers
  - Caches

- Bus
  - Input Devices
  - Output Devices
  - Secondary Storage (e.g., Magnetic hard disk, Flash SSD, etc.)

- Dynamic Random Access Memory (DRAM)

Activities:
- Retrieve; Process
- Store; Retrieve
- Input; Output; Retrieve
Q: How does a processor execute machine code?
Q: How does a processor execute machine code?

❖ Most common approach: load-store architecture
Q: How does a processor execute machine code?

- Most common approach: **load-store architecture**
- **Registers**: Tiny local memory ("scratch space") on proc. into which instructions and data are copied
Q: How does a processor execute machine code?

❖ Most common approach: **load-store architecture**
❖ **Registers**: Tiny local memory ("scratch space") on proc. into which instructions and data are copied
❖ ISA specifies bit length/format of machine code commands
**Q:** How does a processor execute machine code?

- Most common approach: **load-store architecture**
- **Registers:** Tiny local memory ("scratch space") on proc. into which instructions and data are copied
- ISA specifies bit length/format of machine code commands
- ISA has several commands to manipulate register contents
Basics of Processors

Q: *How does a processor execute machine code?*
Q: How does a processor execute machine code?

- Types of ISA commands to manipulate register contents:
  - **Memory access**: load (copy bytes from a DRAM address to register); store (reverse); put constant
  - **Arithmetic & logic** on data items in registers: add/multiply/ etc.; bitwise ops; compare, etc.; handled by ALU
  - **Control flow** (branch, call, etc.); handled by CU
**Q:** How does a processor execute machine code?

- Types of ISA commands to manipulate register contents:
  - **Memory access:** load (copy bytes from a DRAM address to register); **store** (reverse); put constant
  - **Arithmetic & logic** on data items in registers: add/multiply/etc.; bitwise ops; compare, etc.; handled by ALU
  - **Control flow** (branch, call, etc.); handled by CU
  - **Caches:** Small local memory to buffer instructions/data
Basics of Processors

Q: How does a processor execute machine code?

- Types of ISA commands to manipulate register contents:
  - Memory access: load (copy bytes from a DRAM address to register); store (reverse); put constant
  - Arithmetic & logic on data items in registers: add/multiply/etc.; bitwise ops; compare, etc.; handled by ALU
  - Control flow (branch, call, etc.); handled by CU
  - Caches: Small local memory to buffer instructions/data

If interested in more details: https://www.youtube.com/watch?v=cNN_tTXABUA
Q: How fast can a processor process a program?
Q: How fast can a processor process a program?

- Modern CPUs can run millions of instructions per second!
- ISA tells \texttt{clock cycles} per instruction
- CPU’s \texttt{clock rate} helps map that to runtime (ns)
Q: How fast can a processor process a program?

- Modern CPUs can run millions of instructions per second!
  - ISA tells \textit{clock cycles} per instruction
  - CPU’s \textit{clock rate} helps map that to runtime (ns)
- Alas, most programs do not keep CPU always busy:
  - Memory access commands \textbf{stall} the processor; ALU and CU are \textit{idle} during DRAM-register transfer
  - Worse, data may not be in DRAM—wait for (disk) I/O!
  - So, actual \textit{runtime} of program may be OOM higher than what clock rate calculation suggests
Q: How fast can a processor process a program?

- Modern CPUs can run millions of instructions per second!
  - ISA tells \textit{clock cycles} per instruction
  - CPU’s \textit{clock rate} helps map that to runtime (ns)
- Alas, most programs do not keep CPU always busy:
  - Memory access commands \textbf{stall} the processor; ALU and CU are \textit{idle} during DRAM-register transfer
  - Worse, data may not be in DRAM—wait for (disk) I/O!
  - So, actual \textit{runtime} of program may be OOM higher than what clock rate calculation suggests

\textbf{Key Principle:} Optimizing access to DRAM and use of processor caches is critical for processor performance!
Processor-Optimized Math Libraries
Processor-Optimized Math Libraries

Q: Would you like to write ML code in a cache-aware manner? :)
Q: Would you like to write ML code in a cache-aware manner? :)  

- Matrices/tensors are ubiquitous in statistics/ML/DL programs
Processor-Optimized Math Libraries

Q: Would you like to write ML code in a cache-aware manner? :)

❖ Matrices/tensors are ubiquitous in statistics/ML/DL programs
❖ Processor-optimized libraries for matrix/tensor arithmetic (linear algebra) studied for decades
❖ They reduce memory stalls and increase parallelism (more on parallelism later) automatically:
  ❖ Multi-core CPUs: BLAS/LAPACK (C), Eigen (C++), la4j (Java), NumPy/SciPy (Python; can wrap BLAS)
  ❖ GPUs: cuBLAS, cuSPARSE, cuDNN, cuDF, cuGraph
Processor-Optimized Math Libraries

Q: Would you like to write ML code in a cache-aware manner? :)

❖ Matrices/tensors are ubiquitous in statistics/ML/DL programs
❖ Processor-optimized libraries for matrix/tensor arithmetic (linear algebra) studied for decades
❖ They reduce memory stalls and increase parallelism (more on parallelism later) automatically:
  ❖ Multi-core CPUs: BLAS/LAPACK (C), Eigen (C++), la4j (Java), NumPy/SciPy (Python; can wrap BLAS)
  ❖ GPUs: cuBLAS, cuSPARSE, cuDNN, cuDF, cuGraph

If interested, some benchmark empirical comparisons:
https://medium.com/datathings/benchmarking-blas-libraries-b57fb1c6dc7
https://github.com/andre-wojtowicz/blas-benchmarks
Memory/Storage Hierarchy

- CPU
- Cache
- Main Memory
- Magnetic Hard Disk Drive (HDD)
Memory/Storage Hierarchy

- CPU
- Cache
- Main Memory
- Magnetic Hard Disk Drive (HDD)

- Access Cycles: $10^7 - 10^8$
- Cycles: 100s
- Cycles: 1000s
Memory/Storage Hierarchy

- CPU
- Cache
- Main Memory
- Magnetic Hard Disk Drive (HDD)

Access Speed

- 10^7 - 10^8
- 100s
- Cycles

- Memory/Storage Hierarchy

- Access Speed

- 10^7 - 10^8
- 100s
- Cycles
Memory/Storage Hierarchy

Access Speed

~100GB/s

~10GB/s

~200MB/s

Magnetic Hard Disk Drive (HDD)

Main Memory

Cache

CPU

ACCESS CYCLES

107 - 10^8

100s

10s
Memory/Storage Hierarchy

- **Access Speed**
  - CPU Cache: ~100GB/s
  - Main Memory: ~10GB/s
  - Magnetic Hard Disk Drive (HDD): ~200MB/s

- **Cycles**
  - 10^7 - 10^8

- **Capacity**
  - Access to cycles: 100s
Memory/Storage Hierarchy

- CPU Cache
- Main Memory
- Magnetic Hard Disk Drive (HDD)

Access Speed:
- ~100GB/s
- ~10GB/s
- ~200MB/s

Cycles:
- 10^7 - 10^8
- 100s
- 10^10 - 10^11

Price

Capacity
Memory/Storage Hierarchy

- **CPU Cache**: Access Speed ~100GB/s, Access Cycles 100s, Capacity ~10GBs, Price ~$2/MB
- **Main Memory**: Access Speed ~10GB/s, Access Cycles 10s, Capacity ~10TBs, Price ~$5/GB
- **Magnetic Hard Disk Drive (HDD)**: Access Speed ~200MB/s, Access Cycles ~10^7 to ~10^8, Capacity ~10TBs, Price ~$30/TB
Memory/Storage Hierarchy

CPU Cache

Main Memory

Flash Storage

Magnetic Hard Disk Drive (HDD)

Access Speed

- ~200MB/s
- ~GB/s
- ~10GB/s
- ~100GB/s

Cycles

- 10^7 - 10^8
- 10^5 - 10^6
- 100s
- 10s

Price

- ~$2/MB
- ~$5/GB
- ~$200/TB
- ~$30/TB
- ~10TBs
- ~10GBs
- ~MBs
Memory/Storage Hierarchy

- **Flash Storage**
  - Access Speed: ~100GB/s
  - Cycles: 10ns
  - Price: ~$2/MB
  - Capacity: ~10GBs
  - Access Speed: ~10GB/s
  - Price: ~$5/GB
  - Capacity: ~10TBs
  - Price: ~$30/TB
  - Access Speed: ~200MB/s
  - Cycles: 10^8
  - Price: ~$200/TB

- **Magnetic Hard Disk Drive (HDD)**
  - Access Speed: ~50MB/s
  - Cycles: 10^7 - 10^8
  - Price: ~$10/TB
  - Capacity: ~PBs

- **Main Memory (Cache)**
  - Access Speed: ~10GB/s
  - Cycles: 10^5 - 10^6
  - Price: ~$5/GB
  - Capacity: ~10TBs

- **CPU**
  - Access Speed: ~100GB/s
  - Cycles: 10ns
  - Price: ~$2/MB
  - Capacity: ~10GBs
  - Access Speed: ~10GB/s
  - Price: ~$5/GB
  - Capacity: ~10TBs
  - Price: ~$200/TB

- **Tape**
  - Access Speed: ~200MB/s
  - Cycles: 10^8
  - Price: ~$10/TB
  - Capacity: ~PBs
Memory/Storage Hierarchy

- **CPU Cache**
  - Access Speed: ~100GB/s
  - Price: ~$2/MB

- **Main Memory**
  - Access Speed: ~10GB/s
  - Price: ~$5/GB

- **Non-Volatile RAM**
  - Access Speed: ~200MB/s
  - Price: ~$200/TB

- **Flash Storage**
  - Access Speed: ~50MB/s
  - Price: ~$30/TB

- **Magnetic Hard Disk Drive (HDD)**
  - Access Speed: ~10TBs
  - Price: ~$10/TB

- **Tape**
  - Access Speed: ~PBs
  - Price: ~$10/TB
Q: What does this program do when run with ‘python’? (Assume tmp.csv is in current working directory)

```python
import pandas as p
m = p.read_csv('tmp.csv', header=None)
s = m.sum().sum()
print(s)
```

```
1,2,3
4,5,6
```
Memory Hierarchy in Action

Bus

CPU

CU

ALU

Registers

Caches

Retrieve; Process

Monitor

Store; Retrieve

DRAM

Store; Retrieve

Disk

tmp.py

tmp.csv

Monitor
Memory Hierarchy in Action

Rough sequence of events when program is executed

- Bus
  - CPU: CU, ALU
  - Caches
  - Registers
  - Monitor

- Disk: tmp.py, tmp.csv
  - Store; Retrieve

- DRAM
  - Store; Retrieve
Memory Hierarchy in Action

Rough sequence of events when program is executed

- **CPU**
  - CU
  - ALU
  - Registers
  - Caches

- **Bus**

- **Store; Retrieve**
  - DRAM

- **Monitor**

- **I/O for code**
  - Disk
  - tmp.py
  - tmp.csv
Memory Hierarchy in Action

Rough sequence of events when program is executed

- **CPU**
  - CU
  - ALU
  - Registers
  - Caches

- **Bus**
  - **DRAM**
    - Store; Retrieve
  - **Disk**
    - I/O for code
    - tmp.py
    - tmp.csv
  - **Monitor**
    - Store; Retrieve
Memory Hierarchy in Action

Rough sequence of events when program is executed

1. Retrieve; Process
2. CPU
   - CU
   - ALU
   - Registers
   - Caches
3. Bus
4. Commands interpreted
5. Monitor
6. I/O for code
7. Disk
   - tmp.py
   - tmp.csv
8. Store; Retrieve
   - DRAM
9. Store; Retrieve
Memory Hierarchy in Action

Rough sequence of events when program is executed

- **CPU**
  - CU
  - ALU
  - Registers
  - Caches

- **Bus**
  - Commands interpreted

- **Monitor**

- **DRAM**
  - Store; Retrieve

- **I/O for code**
  - Disk
  - tmp.py

- **I/O for data**
  - Disk
  - tmp.csv
Rough sequence of events when program is executed
Memory Hierarchy in Action

Rough sequence of events when program is executed

- **CPU**
  - CU (Control Unit)
  - ALU (Arithmetic Logic Unit)
  - Registers
  - Caches

Arithmetic done within CPU

- **DRAM**
- **Disk**
  - tmp.py
  - tmp.csv

Store; Retrieve

Commands interpreted

I/O for code

Monitor

I/O for data
Memory Hierarchy in Action

Rough sequence of events when program is executed

- CPU
  - CU
  - ALU
  - Registers
  - Caches

- Bus
- DRAM
- Disk

Arithmetic done within CPU

Store; Retrieve

Commands interpreted

I/O for Display
- I/O for code
- I/O for data

Monitor

'21'

'21'

tmp.csv

tmp.py

tmp.csv
Concepts of Memory Management
Caching: Buffering a copy of bytes (instructions and/or data) from a lower level at a higher level to exploit locality
Concepts of Memory Management

❖ **Caching:** Buffering a copy of bytes (instructions and/or data) from a lower level at a higher level to exploit locality

❖ **Prefetching:** Preemptively retrieving bytes (typically data) from addresses not explicitly asked yet by program
Concepts of Memory Management

❖ **Caching**: Buffering a copy of bytes (instructions and/or data) from a lower level at a higher level to exploit locality

❖ **Prefetching**: Preemptively retrieving bytes (typically data) from addresses not explicitly asked yet by program

❖ **Spill/Miss/Fault**: Data needed for program is not yet available at a higher level; need to get it from lower level
Concepts of Memory Management

❖ **Caching**: Buffering a copy of bytes (instructions and/or data) from a lower level at a higher level to exploit locality

❖ **Prefetching**: Preemptively retrieving bytes (typically data) from addresses not explicitly asked yet by program

❖ **Spill/Miss/Fault**: Data needed for program is not yet available at a higher level; need to get it from lower level

❖ **Register Spill** (register to cache); **Cache Miss** (cache to main memory); **“Page” Fault** (main memory to disk)
Concepts of Memory Management

❖ **Caching:** Buffering a copy of bytes (instructions and/or data) from a lower level at a higher level to exploit locality

❖ **Prefetching:** Preemptively retrieving bytes (typically data) from addresses not explicitly asked yet by program

❖ **Spill/Miss/Fault:** Data needed for program is not yet available at a higher level; need to get it from lower level
  - **Register Spill** (register to cache); **Cache Miss** (cache to main memory); “**Page**” **Fault** (main memory to disk)

❖ **Hit:** Data needed is already available at higher level
Concepts of Memory Management

❖ **Caching**: Buffering a copy of bytes (instructions and/or data) from a lower level at a higher level to exploit locality

❖ **Prefetching**: Preemptively retrieving bytes (typically data) from addresses not explicitly asked yet by program

❖ **Spill/Miss/Fault**: Data needed for program is not yet available at a higher level; need to get it from lower level
  ❖ **Register Spill** (register to cache); **Cache Miss** (cache to main memory); “Page” **Fault** (main memory to disk)

❖ **Hit**: Data needed is already available at higher level

❖ **Cache Replacement Policy**: When new data needs to be loaded to higher level, which old data to evict to make room? Many policies exist with different properties
Memory/Storage Hierarchy
Memory/Storage Hierarchy

- Typical desktop computer today ($700):
  - 1 TB magnetic hard disk (SATA HDD); 32 GB DRAM
  - 3.4 GHz CPU; 4 cores; 8MB cache
Memory/Storage Hierarchy

- Typical desktop computer today ($700):
  - 1 TB magnetic hard disk (SATA HDD); 32 GB DRAM
  - 3.4 GHz CPU; 4 cores; 8MB cache
- High-end enterprise rack server for RDBMSs ($8,000):
  - 12 TB Persistent memory; 6 TB DRAM
  - 3.8 GHz CPU; 28-core per proc.; 38MB cache
Memory/Storage Hierarchy

- Typical desktop computer today ($700):
  - 1 TB magnetic hard disk (SATA HDD); 32 GB DRAM
  - 3.4 GHz CPU; 4 cores; 8MB cache

- High-end enterprise rack server for RDBMSs ($8,000):
  - 12 TB Persistent memory; 6 TB DRAM
  - 3.8 GHz CPU; 28-core per proc.; 38MB cache

- Renting on Amazon Web Services (AWS):
  - EC2 m5.large: 2-core, 8GiB: $0.096 / hour
  - EC2 m5.24xlarge: 96-core, 384 GiB, $4.608 per hour
  - EBS general SSD: $0.10 per GB-month
  - S3 store / read: $0.023 / 0.013-0.023 per GB-month
Review Questions

❖ What is an ISA?
Review Questions

❖ What is an ISA?
❖ What are the 3 main kinds of commands in an ISA?
Review Questions

❖ What is an ISA?
❖ What are the 3 main kinds of commands in an ISA?
❖ Why do CPUs have both registers and caches?
Review Questions

❖ What is an ISA?
❖ What are the 3 main kinds of commands in an ISA?
❖ Why do CPUs have both registers and caches?
❖ Why is it typically impossible for data processing programs to achieve 100% processor utilization?
Review Questions

❖ What is an ISA?
❖ What are the 3 main kinds of commands in an ISA?
❖ Why do CPUs have both registers and caches?
❖ Why is it typically impossible for data processing programs to achieve 100% processor utilization?
❖ Which of these memory hierarchy layers is the most expensive: CPU cache, DRAM, flash disks, or magnetic hard disks?
Review Questions

❖ What is an ISA?
❖ What are the 3 main kinds of commands in an ISA?
❖ Why do CPUs have both registers and caches?
❖ Why is it typically impossible for data processing programs to achieve 100% processor utilization?
❖ Which of these memory hierarchy layers is the most expensive: CPU cache, DRAM, flash disks, or magnetic hard disks?
❖ Which of the above layers is the slowest for data access?
Review Questions

❖ What is an ISA?
❖ What are the 3 main kinds of commands in an ISA?
❖ Why do CPUs have both registers and caches?
❖ Why is it typically impossible for data processing programs to achieve 100% processor utilization?
❖ Which of these memory hierarchy layers is the most expensive: CPU cache, DRAM, flash disks, or magnetic hard disks?
❖ Which of the above layers is the slowest for data access?
❖ Which library helps ML users avoid need for writing GPU cache-aware computations?
Memory Hierarchy in PA0
Memory Hierarchy in PA0

- Pandas DataFrame needs data to fit entirely in DRAM
Memory Hierarchy in PA0

- Pandas DataFrame needs data to fit entirely in DRAM
- **Dask DataFrame** automatically manages Disk vs DRAM for you
  - Full data sits on Disk, brought to DRAM upon compute()
  - Dask stages out computations using Pandas
Memory Hierarchy in PA0

- **Pandas DataFrame** needs data to fit entirely in DRAM
- **Dask DataFrame** automatically manages Disk vs DRAM for you
  - Full data sits on Disk, brought to DRAM upon compute()
  - Dask stages out computations using Pandas

**Tradeoff:** Dask may throw memory configuration issues. :(
Outline

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy
❖ Basics of Operating Systems (OS)
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management
❖ Persistent Data Storage
Outline

❖ Basics of Computer Organization
  ❖ Digital Representation of Data
  ❖ Processors and Memory Hierarchy

❖ Basics of Operating Systems (OS)
  ❖ Process Management: Virtualization; Concurrency
  ❖ Filesystem and Data Files
  ❖ Main Memory Management

❖ Persistent Data Storage
Optional/Advanced Extra Reading
(NOT included for the exams)
Key Principle: Locality of Reference

Carefully handling/optimizing access to DRAM and use of processor caches is critical for processor performance!
Key Principle: Locality of Reference

Carefully handling/optimizing access to DRAM and use of processor caches is critical for processor performance!

Due to OOM access latency gaps across memory hierarchy, optimizing access to lower levels and careful use of higher levels is critical for overall system performance!
Key Principle: Locality of Reference

Carefully handling/optimizing access to DRAM and use of processor caches is critical for processor performance!

Due to OOM access latency gaps across memory hierarchy, optimizing access to lower levels and careful use of higher levels is critical for overall system performance!

- **Locality of Reference:** Many programs tends to access DRAM locations in a somewhat *predictable* manner
Carefully handling/optimizing access to DRAM and use of processor caches is critical for processor performance!

Due to OOM access latency gaps across memory hierarchy, optimizing access to lower levels and careful use of higher levels is critical for overall system performance!

- **Locality of Reference**: Many programs tend to access DRAM locations in a somewhat *predictable* manner
- **Spatial**: Nearby locations will be accessed soon
Key Principle: Locality of Reference

Carefully handling/optimizing access to DRAM and use of processor caches is critical for processor performance!

Due to OOM access latency gaps across memory hierarchy, optimizing access to lower levels and careful use of higher levels is critical for overall system performance!

- **Locality of Reference**: Many programs tend to access DRAM locations in a somewhat predictable manner
- **Spatial**: Nearby locations will be accessed soon
- **Temporal**: Same locations accessed again soon
Key Principle: Locality of Reference

Carefully handling/optimizing access to DRAM and use of processor caches is critical for processor performance!

Due to OOM access latency gaps across memory hierarchy, optimizing access to lower levels and careful use of higher levels is critical for overall system performance!

❖ **Locality of Reference:** Many programs tends to access DRAM locations in a somewhat *predictable* manner
  ❖ **Spatial:** Nearby locations will be accessed soon
  ❖ **Temporal:** Same locations accessed again soon
❖ Locality can be exploited to reduce runtimes using *caching* and/or *prefetching* across all levels in the hierarchy
Locality of Reference for Data
Locality of Reference for Data

❖ **Data Layout:**
❖ The *order* in which data items of a complex data structure/ADT are laid out in memory/disk
Locality of Reference for Data

❖ **Data Layout:**
  ❖ The *order* in which data items of a complex data structure/ADT are laid out in memory/disk

❖ **Data Access Pattern** (of a program on a data object):
   ❖ The *order* in which a program has to access items of a complex data structure/ADT in memory
Locality of Reference for Data

- **Data Layout:**
  - The *order* in which data items of a complex data structure/ADT are laid out in memory/disk

- **Data Access Pattern** (of a program on a data object):
  - The *order* in which a program has to access items of a complex data structure/ADT in memory

- **Hardware Efficiency** (of a program):
  - How close *actual execution runtime* is to best possible runtime given the proc. clock rate and ISA
Locality of Reference for Data

❖ **Data Layout:**
  ❖ The *order* in which data items of a complex data structure/ADT are laid out in memory/disk

❖ **Data Access Pattern** (of a program on a data object):
  ❖ The *order* in which a program has to access items of a complex data structure/ADT in memory

❖ **Hardware Efficiency** (of a program):
  ❖ How close *actual execution runtime* is to best possible runtime given the proc. clock rate and ISA
  ❖ Improved with careful data layout of all data objects used by a program based on its data access patterns
Locality of Reference for Data

❖ **Data Layout:**
  ❖ The *order* in which data items of a complex data structure/ADT are laid out in memory/disk

❖ **Data Access Pattern** (of a program on a data object):
  ❖ The *order* in which a program has to access items of a complex data structure/ADT in memory

❖ **Hardware Efficiency** (of a program):
  ❖ How close *actual execution runtime* is to best possible runtime given the proc. clock rate and ISA
  ❖ Improved with careful data layout of all data objects used by a program based on its data access patterns

❖ **Key Principle:** Raise cache hits; reduce memory stalls!
Locality of Reference in Data Science

- Common example: matrix multiplication (>1m cells each)
- Suppose data layout in DRAM is in **row-major** order

\[ C_{n \times m} = A_{n \times p} \cdot B_{p \times m} \]
Locality of Reference in Data Science

- Common example: matrix multiplication (>1m cells each)
- Suppose data layout in DRAM is in \textit{row-major} order

\[
C_{n \times m} = A_{n \times p} \times B_{p \times m}
\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Caches</td>
</tr>
</tbody>
</table>
Locality of Reference in Data Science

- Common example: matrix multiplication (>1m cells each)
- Suppose data layout in DRAM is in **row-major** order

\[ C_{n \times m} = A_{n \times p} B_{p \times m} \]

for \( i = 1 \) to \( n \)
    for \( j = 1 \) to \( m \)
        for \( k = 1 \) to \( p \)
            \[ C[i][j] += A[i][k] * B[k][j] \]


Caches
Locality of Reference in Data Science

- Common example: matrix multiplication (>1m cells each)
- Suppose data layout in DRAM is in **row-major** order

\[
C_{n \times m} = A_{n \times p} \times B_{p \times m}
\]

Not too hardware-efficient

```
for i = 1 to n
  for j = 1 to m
    for k = 1 to p
      C[i][j] += A[i][k] * B[k][j]
```
Locality of Reference in Data Science

❖ Common example: matrix multiplication (>1m cells each)
❖ Suppose data layout in DRAM is in **row-major** order

\[ C_{n \times m} = A_{n \times p} B_{p \times m} \]

Not too hardware-efficient
❖ Prefetching+caching means full row based on innermost loop is brought to proc. cache
Locality of Reference in Data Science

- Common example: matrix multiplication (>1m cells each)
- Suppose data layout in DRAM is in **row-major** order

\[
C_{n \times m} = A_{n \times p} B_{p \times m}
\]

```plaintext
for i = 1 to n
  for j = 1 to m
    for k = 1 to p
      C[i][j] += A[i][k] * B[k][j]
```

- Not too hardware-efficient
- Prefetching+caching means full row based on innermost loop is brought to proc. cache
- A[i][.] Hits but B[k][j] Misses
Locality of Reference in Data Science

- Common example: matrix multiplication (>1m cells each)
- Suppose data layout in DRAM is in **row-major** order

\[ C_{n \times m} = A_{n \times p} B_{p \times m} \]

for \( i = 1 \) to \( n \)
for \( j = 1 \) to \( m \)
for \( k = 1 \) to \( p \)
\[ C[i][j] += A[i][k] \times B[k][j] \]

- Not too hardware-efficient
- Prefetching+caching means full row based on innermost loop is brought to proc. cache
- \( A[i][.] \) Hits but \( B[k][j] \) Misses
- So each * op is a stall! :(
Locality of Reference in Data Science

- Common example: matrix multiplication (>1m cells each)
- Suppose data layout in DRAM is in **row-major** order

\[
C_{n \times m} = A_{n \times p} \cdot B_{p \times m}
\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Caches</td>
<td>A[1;]</td>
<td>B[1;]</td>
<td>C[1;]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Caches A[1;]. B[1;]. C[1;]
Locality of Reference in Data Science

❖ Common example: matrix multiplication (>1m cells each)
❖ Suppose data layout in DRAM is in **row-major** order

\[
C_{n \times m} = A_{n \times p} \ B_{p \times m}
\]

for i = 1 to n  
  for k = 1 to p  
    for j = 1 to m  
      \[ C[i][j] += A[i][k] \times B[k][j] \]
Locality of Reference in Data Science

- Common example: matrix multiplication (>1m cells each)
- Suppose data layout in DRAM is in **row-major** order

\[ C_{n \times m} = A_{n \times p} \cdot B_{p \times m} \]

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Caches</td>
<td>A[1;]. B[1;]. C[1;]</td>
</tr>
</tbody>
</table>

```
for i = 1 to n
    for k = 1 to p
        for j = 1 to m
            C[i][j] += A[i][k] * B[k][j]
```
Locality of Reference in Data Science

❖ Common example: matrix multiplication (>1m cells each)
❖ Suppose data layout in DRAM is in **row-major** order

\[ C_{n \times m} = A_{n \times p} \cdot B_{p \times m} \]

- **DRAM**

- **Caches**
  - \( A[1:] \), \( B[1:] \), \( C[1:] \)

for i = 1 to n
  for k = 1 to p
    for j = 1 to m
      \( C[i][j] \) += \( A[i][k] \) * \( B[k][j] \)

❖ *Logically equivalent* computation but different order of ops!
❖ \( C[i][] \) and \( B[k][] \) Hits
Locality of Reference in Data Science

❖ Common example: matrix multiplication (>1m cells each)
❖ Suppose data layout in DRAM is in **row-major** order

\[
C_{n \times m} = A_{n \times p} \cdot B_{p \times m}
\]

- Logically equivalent computation but different order of ops!
- \(C[i][.]\) and \(B[k][.]\) Hits
- \(A[i][k]\) also Hit (unaffected by \(j\))
Locality of Reference in Data Science

❖ Common example: matrix multiplication (>1m cells each)
❖ Suppose data layout in DRAM is in row-major order

\[ C_{n \times m} = A_{n \times p} \cdot B_{p \times m} \]

for \( i = 1 \) to \( n \)
for \( k = 1 \) to \( p \)
for \( j = 1 \) to \( m \)

\[ C[i][j] = A[i][k] \cdot B[k][j] \]

❖ Logically equivalent computation but different order of ops!
❖ \( C[i][.] \) and \( B[k][.] \) Hits
❖ \( A[i][k] \) also Hit (unaffected by \( j \))
❖ Orders of magnitude fewer stalls!
❖ Lot more hardware-efficient
Locality of Reference in Data Science

- Common example: matrix multiplication (>1m cells each)
- Suppose data layout in DRAM is in **row-major** order

\[ C_{n \times m} = A_{n \times p} \cdot B_{p \times m} \]
Common example: matrix multiplication (>1m cells each)

Suppose data layout in DRAM is in row-major order

\[ C_{n \times m} = A_{n \times p} \cdot B_{p \times m} \]

for i = 1 to n
  for j = 1 to m
    for k = 1 to p
      C[i][j] += A[i][k] * B[k][j]
Locality of Reference in Data Science

- Common example: matrix multiplication (>1m cells each)
- Suppose data layout in DRAM is in row-major order

\[ C_{n \times m} = A_{n \times p} \cdot B_{p \times m} \]

for i = 1 to n
  for j = 1 to m
    for k = 1 to p
      C[i][j] += A[i][k] * B[k][j]

Rewrite

for i = 1 to n
  for k = 1 to p
    for j = 1 to m
      C[i][j] += A[i][k] * B[k][j]
Locality of Reference in Data Science

❖ Common example: matrix multiplication (>1m cells each)
❖ Suppose data layout in DRAM is in **row-major** order

\[ C_{n \times m} = A_{n \times p} \cdot B_{p \times m} \]

for i = 1 to n
   for j = 1 to m
      for k = 1 to p
         C[i][j] += A[i][k] * B[k][j]

Although the math is the same and gives the same results ("logically equivalent"), the physical properties of program execution are vastly different.

for i = 1 to n
   for k = 1 to p
      for j = 1 to m
         C[i][j] += A[i][k] * B[k][j]
Locality of Reference in Data Science

❖ Common example: matrix multiplication (>1m cells each)
❖ Suppose data layout in DRAM is in row-major order

\[ C_{n \times m} = A_{n \times p} B_{p \times m} \]

```plaintext
for i = 1 to n
  for j = 1 to m
    for k = 1 to p
      C[i][j] += A[i][k] * B[k][j]
```

❖ Although the math is the same and gives the same results ("logically equivalent"), the physical properties of program execution are vastly different
❖ Commonly used in compiler optimization and later on, also in query optimization

```plaintext
for i = 1 to n
  for k = 1 to p
    for j = 1 to m
      C[i][j] += A[i][k] * B[k][j]
```