Systems research enables application development by defining and implementing abstractions:

- **Operating systems** provide a stable, consistent interface to unreliable, inconsistent hardware.
- **Networks** provide a robust data transfer interface to constantly evolving communications infrastructure.
- **Databases** provide a declarative interface to complex software that stores and retrieves information efficiently.
- **Distributed systems** provide a unified interface to a cluster of multiple machines.

A unifying property of effective systems:

*Hide complexity, but retain flexibility.*

**Unix**

**Example: The Unix Operating System**

Essential features of the Unix operating system (and variants):

- **Portability:** The same operating system on different hardware.
- **Multi-Tasking:** Many processes run concurrently on a machine.
- **Plain Text:** Data is stored and shared in text format.
- **Modularity:** Small tools are composed flexibly via pipes.

"We should have some ways of coupling programs like [a] garden hose – screw in another segment when it becomes necessary to massage data in another way," Doug McIlroy in 1964.

**Python Programs in a Unix Environment**

- **sys.stdin** and **sys.stdout** values provide access to the Unix standard streams as files.

A Python file has an interface that supports iteration, read, and write methods.

Using these "files" takes advantage of the operating system text processing abstraction.

The **input** and **print** functions also read from standard input and write to standard output.

**Big Data**

**Big Data Examples**

- **Facebook's daily logs:** 60 Terabytes (60,000 Gigabytes)
- **1,000 genomes project:** 200 Terabytes
- **Google web index:** 10+ Petabytes (10,000,000 Gigabytes)

Time to read 1 Terabyte from disk: 3 hours (100 Megabytes/second)

Typical hardware for big data applications:

- Consumer-grade hard disks and processors
- Independent computers are stored in racks
- Concerns: networking, heat, power, monitoring

When using many computers, some will fail.
Apache Spark

Apache Spark is a data processing system that provides a simple interface for large data:
- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Supports common UNIX operations: sort, distinct, uniq in UNIX, count, pipe
- Supports common sequence operations: map, filter, reduce
- Supports common database operations: join, union, intersection

All of these operations can be performed on RDDs that are partitioned across machines.

Apache Spark Execution Model

Processing is defined centrally but executed remotely:
- A Resilient Distributed Dataset (RDD) is distributed in partitions to worker nodes.
- A driver program defines transformations and actions on an RDD.
- A cluster manager assigns tasks to individual worker nodes to carry them out.
- Worker nodes perform computation & communicate values to each other.
- Final results are communicated back to the driver program.

Apache Spark Interface

The SparkContext gives access to the cluster manager:

```python
click >>>
```

A SparkContext can be constructed from the lines of a text file:
```
A.textFile('shakespeare.txt')
```

The `sortBy` transformation and `take` action are methods:
```
(x, y).sortBy(take(2))
```

MapReduce

MapReduce is an important early distributed processing system developed at Google.

Generic application structure that happened to capture many common data processing tasks:

1. Map phase: Apply a mapper function to all inputs, emitting intermediate key-value pairs.
2. Reduce phase: For each intermediate key, apply a reducer function to accumulate all values associated with that key.

MapReduce Evaluation Model

Map phase:
- The mapper yields zero or more key-value pairs for each input.

Reduce phase:
- The reducer yields zero or more values, each associated with that intermediate key.
**MapReduce Evaluation Model**

Google MapReduce
Is a Big Data framework
For batch processing

**Reduce phase**: For each intermediate key, apply a reducer function to accumulate all values associated with that key

- All key-value pairs with the same key are processed together
- The reducer yields zero or more values, each associated with that intermediate key

```
| a: 4 | reducer | a: 6 | i: 2 |
| a: 1 |         | a: 5 |     |
| a: 3 |         |      |     |
```

**MapReduce Applications on Apache Spark**

Key-value pairs are just two-element Python tuples

<table>
<thead>
<tr>
<th>Call Expression</th>
<th>Data</th>
<th>fn Input</th>
<th>fn Output</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data.flatMap(fn)</td>
<td>Values</td>
<td>One value</td>
<td>Zero or more key-value pairs</td>
<td>All key-value pairs returned by calls to fn</td>
</tr>
<tr>
<td>data.reduceByKey(fn)</td>
<td>Key-value pairs</td>
<td>Two values</td>
<td>One value</td>
<td>One key-value pair for each unique key</td>
</tr>
</tbody>
</table>