Projects
End-of-semester Review
cs378h
Outline/Administrivia

• Questions?
• Comments on Exam
• Project presentations x 2
• Review
  • Can someone please act as scribe?
  • Requested review content:
    • NoSQL/databases
    • ACID vs. BASE
    • Linearizability vs. Serializability
    • Spark
      • Pros/cons wrt page rank / indexing
      • Pros/cons wrt multi-core parallelism
Project Presentations

• Emily & Abby
• Ryan & Patrick
• Any last minute additions?
What is NoSQL?

- Next Generation Compute/Storage engines (databases)
  - non-relational
  - distributed
  - open-source
  - horizontally scalable
- One view: “no” \(\Rightarrow\) elide SQL/database functionality to achieve scale
- Another view: “NoSQL” is actually misleading.
  - more appropriate term is actually “Not Only SQL”
What is NoSQL?

- Next Generation Compute/Storage engines (databases)
  - non-relational
  - distributed
  - open-source
  - horizontally scalable
- One view: “no” → elide SQL/database functionality to achieve scale
- Another view: “NoSQL” is actually misleading.
  - more appropriate term is actually “Not Only SQL”

What NoSQL gives up in exchange for scale:
- Relationships between entities are non-existent
- Limited or no ACID transactions
- No standard language for queries (SQL)
- Less structured
What is NoSQL?

- Next Generation Compute/Storage engines (databases)
  - non-relational
  - distributed
  - open-source
  - horizontally scalable
- One view: “no” → elide SQL/database functionality to achieve scale
- Another view: “NoSQL” is actually misleading.
  - more appropriate term is actually “Not Only SQL”

What NoSQL gives up in exchange for scale:

- Why talk about NoSQL in concurrency class?
  - Principle
    - Most tradeoffs are a direct result of concurrency
  - Practice
    - NoSQL systems are ubiquitous
  - Relevant aspects
    - scale/performance tradeoff space
    - Correctness/programmability tradeoff space
Review: noSQL Taxonomy
Review: noSQL Taxonomy

Consistency
Review: noSQL Taxonomy
Review: noSQL Taxonomy
Review: noSQL Taxonomy

- Strong: ACID
- Eventual: BASE

Data Model

Implementation Techniques

Consistency
Review: noSQL Taxonomy

Consistency

- Atomicity
- Consistency
- Isolation
- Durability

Data Model

Implementation Techniques
Review: noSQL Taxonomy

- Strong: ACID
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- Eventual: BASE
  - Basically Available
  - Soft State
  - Eventually Consistent
Review: noSQL Taxonomy

Consistency

Strong: ACID

Eventual: BASE
Review: noSQL Taxonomy

Consistency:
- Strong: ACID
- Eventual: BASE
Review: noSQL Taxonomy

- **Key Value Stores**

  - Strong: ACID
  - Eventual: BASE

  - Consistency

  - Implementation Techniques

  - Data Model
Review: noSQL Taxonomy

- Key Value Stores
- Document Stores

Consistency:
- Strong: ACID
- Eventual: BASE

Data Model

Implementation Techniques
Review: noSQL Taxonomy

- Key Value Stores
- Document Stores
- Wide-Column Stores

Strong: ACID
Eventual: BASE

Consistency

Data Model

Implementation Techniques
Review: noSQL Taxonomy

Key Value Stores

Document Stores

Wide-Column Stores

Strong: ACID

Eventual: BASE

Sharding/Partitioning

Consistency

Implementation Techniques

Data Model
Review: noSQL Taxonomy

Data Model

- Key Value Stores
- Document Stores
- Wide-Column Stores

Implementation Techniques
- Sharding/Partitioning
- Replication

Consistency
- Strong: ACID
- Eventual: BASE
Review: noSQL Taxonomy

Key Value Stores

Document Stores

Wide-Column Stores

Strong: ACID

Eventual: BASE

Consistency

Sharding/Partitioning

Replication

Storage

Data Model

Implementation Techniques
Review: noSQL Taxonomy

- Key Value Stores
- Document Stores
- Wide-Column Stores

**Consistency**
- Strong: ACID
- Eventual: BASE

**Implementation Techniques**
- Sharding/Partitioning
- Replication
- Storage
- Query Support
Review: noSQL Taxonomy

Key Value Stores

Document Stores

Wide-Column Stores

Strong: ACID

Eventual: BASE

Consistency

Sharding/Partitioning

Replication

Storage

Query Support

Implementation Techniques

• Shared-Disk
• Range-Sharding
• Hash-Sharding
• Consistent Hashing
Review: noSQL Taxonomy

- **Consistency**
  - Strong: ACID
  - Eventual: BASE

- **Data Model**
  - Key Value Stores
  - Document Stores
  - Wide-Column Stores

- **Implementation Techniques**
  - Sharding/Partitioning
  - Replication
  - Storage
  - Query Support
Review: noSQL Taxonomy

- Key Value Stores
- Document Stores
- Wide-Column Stores

Consistency:
- Strong: ACID
- Eventual: BASE

Implementation Techniques:
- Primary-Backup
- Commit-Consensus Protocol
- Sync/Async
Review: noSQL Taxonomy

- **Wide-Column Stores**
  - Consistency: Strong: ACID
  - Implementation Techniques: Sharding/Partitioning, Replication, Storage, Query Support
- **Document Stores**
  - Consistency: Eventual: BASE
  - Implementation Techniques: Sharding/Partitioning, Replication, Storage, Query Support
- **Key Value Stores**
  - Implementation Techniques: Sharding/Partitioning, Replication, Storage, Query Support
Review: noSQL Taxonomy

- Key Value Stores
- Document Stores
- Wide-Column Stores

**Consistency**
- Strong: ACID
- Eventual: BASE

**Implementation Techniques**
- Logging
- Update In Place
- Caching
- In-Memory Storage

**Data Model**

**Sharding/Partitioning**

**Replication**

**Storage**

**Query Support**
Review: noSQL Taxonomy

Key Value Stores

Document Stores

Wide-Column Stores

Strong: ACID

Eventual: BASE

Consistency

Sharding/Partitioning

Replication

Storage

Query Support

Data Model

Implementations Techniques
Review: noSQL Taxonomy

- **Key Value Stores**
  - Consistency: Strong: ACID
  - Sharding/Partitioning
  - Replication
  - Storage
  - Query Support
  - Implementation Techniques:
    - Secondary Indexing
    - Query Planning
    - Materialized Views
    - Analytics

- **Document Stores**
  - Consistency: Eventual: BASE

- **Wide-Column Stores**
  - Consistency: Eventual: BASE
Review: noSQL Taxonomy

Key Value Stores
Document Stores
Wide-Column Stores

Strong: ACID
Eventual: BASE

Consistency

Sharding/Partitioning
Replication
Storage
Query Support

Data Model
Implementation Techniques
NoSQL faux quiz:

• What is the CAP theorem? What does “PACELC” stand for and how does it relate to CAP?
• What is the difference between ACID and BASE?
• Why do NoSQL systems claim to be more horizontally scalable than RDBMSes? List some features NoSQL systems give up toward this goal?
• What is eventual consistency? Give a concrete example of how or why it causes a complex programming model (relative to a strongly consistent model).
• Compare and contrast Key-Value, Document, and Wide-column Stores
• Define and contrast the following consistency properties:
  • strong consistency, eventual consistency, consistent prefix, monotonic reads, read-my-writes, bounded staleness
NoSQL faux quiz:

• What is the CAP theorem? What does “PACELC” stand for and how does it relate to CAP?
• What is the difference between ACID and BASE?
• Why do NoSQL systems claim to be more horizontally scalable than RDMBSes? List some features NoSQL systems give up toward this goal?
• What is eventual consistency? Give a concrete example of how of why it causes a complex programming model (relative to a strongly consistent model).

• Compare and contrast Key-Value, Document, and Wide-column Stores
• Define and contrast the following consistency properties:
  • strong consistency, eventual consistency, consistent prefix, monotonic reads, read-my-writes, bounded staleness
## Consistency

<table>
<thead>
<tr>
<th>col</th>
<th>col</th>
<th>col₂</th>
<th>...</th>
<th>colₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consistency
### Consistency

<table>
<thead>
<tr>
<th>col</th>
<th>col</th>
<th>col_2</th>
<th>...</th>
<th>col_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Partitions

- [Blue](#)
- [Green](#)
- [Gray](#)
- [Orange](#)

**Key Concepts**

- Key Value Stores
- Document Stores
- Data Model
Consistency

<table>
<thead>
<tr>
<th>col</th>
<th>col</th>
<th>col₂</th>
<th>...</th>
<th>colₖ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Consistency
Consistency

- Clients perform reads and writes
Consistency

- Clients perform reads and writes
- Data is replicated among a set of servers
Consistency

- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
Consistency

- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes
Consistency

- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes
- How to keep data in sync?
Consistency

- Clients perform reads and writes
- Data is replicated among a set of servers
- Writes must be performed at all servers
- Reads return the result of one or more past writes
- How to keep data in sync?

Consistency ≠ Correctess
- Consistency: no internal contradictions
- Correct: higher-level property
- Inconsistency → code does wrong things
Consistency Spectrum

- **Eventual Consistency**
  - If writes to a key stop, all replicas of key will converge
  - Originally from Amazon’s Dynamo and LinkedIn’s Voldemort systems

- **Strict:**
  - Absolute time ordering of all shared accesses, reads always return last write

- **Linearizability:**
  - Each operation is visible (or available) to all other clients in real-time order

- **Sequential Consistency** [Lamport]:
  - “… the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.”
  - After the fact, find a “reasonable” ordering of the operations (can re-order operations) that obeys sanity (consistency) at all clients, and across clients.

- **ACID properties**

---

**BASE:**
- **Basically Available**
- **Soft State**
- **Eventually Consistent**

---

**Faster reads and writes**

**More consistency**

Eventual → Strong (e.g., Sequential)
Consistency Spectrum

- **Eventual Consistency**
  - If writes to a key stop, all replicas of key will converge
  - Originally from Amazon’s Dynamo and LinkedIn’s Voldemort systems

**BASE:**
- Basically Available
- Soft State
- Eventually Consistent

- **Strict:**
  - Absolute time ordering of all shared accesses, reads always return last write

- **Linearizability:**
  - Each operation is visible (or available) to all other clients in real-time order

- **Sequential Consistency** [Lamport]:
  - “… the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.
  - After the fact, find a “reasonable” ordering of the operations (can re-order operations) that obeys sanity (consistency) at all clients, and across clients.

- **ACID properties**

---

**Faster reads and writes**

**More consistency**

**Eventual** → **Strong**

Eventual: BASE

Strong: ACID (e.g., Sequential)
Sequential Consistency

- weaker than strict/strong consistency
  - All operations are executed in some sequential order
  - each process issues operations in program order
    - Any valid interleaving is allowed
    - All agree on the same interleaving
    - Each process preserves its program order
Sequential Consistency

• weaker than strict/strong consistency
  • All operations are executed in some sequential order
  • each process issues operations in program order
    • Any valid interleaving is allowed
    • All agree on the same interleaving
    • Each process preserves its program order

• Why is this weaker than strict/strong?
Sequential Consistency

• weaker than strict/strong consistency
  • All operations are executed in some sequential order
  • each process issues operations in program order
    • Any valid interleaving is allowed
    • All agree on the same interleaving
    • Each process preserves its program order

Why is this weaker than strict/strong?

Nothing is said about “most recent write”
Linearizability vs. Serializability

http://www.bailis.org/blog/linearizability-versus-serializability/
Linearizability vs. Serializability

- Linearizability assumes sequential consistency and
  - If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  - Stronger than sequential consistency

http://www.bailis.org/blog/linearizability-versus-serializability/
Linearizability vs. Serializability

• Linearizability assumes sequential consistency and
  • If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  • Stronger than sequential consistency

• Difference between linearizability and serializability?
  • Granularity: reads/writes versus transactions
Linearizability vs. Serializability

- Linearizability assumes sequential consistency and
  - If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  - Stronger than sequential consistency
- Difference between linearizability and serializability?
  - Granularity: reads/writes versus transactions

Linearizability:

http://www.bailis.org/blog/linearizability-versus-serializability/
Linearizability vs. Serializability

• Linearizability assumes sequential consistency and
  • If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  • Stronger than sequential consistency
• Difference between linearizability and serializability?
  • Granularity: reads/writes versus transactions

Linearizability:
• Single-operation, single-object, real-time order

http://www.bailis.org/blog/linearizability-versus-serializability/
Linearizability vs. Serializability

• Linearizability assumes sequential consistency and
  • If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  • Stronger than sequential consistency
• Difference between linearizability and serializability?
  • Granularity: reads/writes versus transactions

Linearizability:
• Single-operation, single-object, real-time order
• Talks about order of ops on single object (e.g. atomic register)

http://www.bailis.org/blog/linearizability-versus-serializability/
Linearizability vs. Serializability

• Linearizability assumes sequential consistency and
  • If $TS(x) < TS(y)$ then OP(x) should precede OP(y) in the sequence
  • Stronger than sequential consistency
• Difference between linearizability and serializability?
  • Granularity: reads/writes versus transactions

Linearizability:
• Single-operation, single-object, real-time order
• Talks about order of ops on single object (e.g. atomic register)
• Ops should appear instantaneous, reflect real time order

http://www.bailis.org/blog/linearizability-versus-serializability/
Linearizability vs. Serializability

- Linearizability assumes sequential consistency *and*
  - If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  - Stronger than sequential consistency

- Difference between linearizability and serializability?
  - Granularity: reads/writes versus transactions

Linearizability:
- Single-operation, single-object, real-time order
- Talks about order of ops on single object (e.g. atomic register)
- Ops should appear instantaneous, reflect real time order

Serializability:
Linearizability vs. Serializability

• Linearizability assumes sequential consistency *and*
  • If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  • Stronger than sequential consistency
• Difference between linearizability and serializability?
  • Granularity: reads/writes versus transactions

Linearizability:
• Single-operation, single-object, real-time order
• Talks about order of ops on single object (e.g. atomic register)
• Ops should appear instantaneous, reflect real time order

Serializable:
• Talks about groups of 1 or more ops on one or more objects

http://www.bailis.org/blog/linearizability-versus-serializability/
Linearizability vs. Serializability

- Linearizability assumes sequential consistency *and*
  - If $TS(x) < TS(y)$ then $OP(x)$ should precede $OP(y)$ in the sequence
  - Stronger than sequential consistency

- Difference between linearizability and serializability?
  - Granularity: reads/writes versus transactions

Linearizability:
- Single-operation, single-object, real-time order
- Talks about order of ops on single object (e.g. atomic register)
- Ops should appear instantaneous, reflect real time order

Serializability:
- Talks about groups of 1 or more ops on one or more objects
- Txns over multiple items equivalent to serial order of txns

http://www.bailis.org/blog/linearizability-versus-serializability/
Linearizability vs. Serializability

• Linearizability assumes sequential consistency and
  • If TS(x) < TS(y) then OP(x) should precede OP(y) in the sequence
  • Stronger than sequential consistency
• Difference between linearizability and serializability?
  • Granularity: reads/writes versus transactions

Linearizability:
  • Single-operation, single-object, real-time order
  • Talks about order of ops on single object (e.g. atomic register)
  • Ops should appear instantaneous, reflect real time order

Serializability:
  • Talks about groups of 1 or more ops on one or more objects
  • Txns over multiple items equivalent to serial order of txns
  • Only requires *some* equivalent serial order

http://www.bailis.org/blog/linearizability-versus-serializability/
Linearizability vs. Serializability

- Linearizability assumes sequential consistency and
  - If $TS(x) < TS(y)$ then $OP(x)$ should precede $OP(y)$ in the sequence
  - Stronger than sequential consistency
- Difference between linearizability and serializability?
  - Granularity: reads/writes versus transactions

**Linearizability:**
- Single-operation, single-object, real-time order
- Talks about order of ops on single object (e.g. atomic register)
- Ops should appear instantaneous, reflect real time order

**Serializability:**
- Talks about groups of 1 or more ops on one or more objects
- Txns over multiple items equivalent to serial order of txns
- Only requires *some* equivalent serial order

Serializability + Linearizability == “Strict Serializability”
- Txn order equivalent to some serial order *that respects real time order*
- Linearizability: degenerate case of Strict Ser: txns are single op single object

Causal consistency
Causal consistency

- Causally related writes seen by all processes in same order.
Causal consistency

• Causally related writes seen by all processes in same order.
  • *Causally*?
Causal consistency

• Causally related writes seen by all processes in the same order.

  • Causally?

Causal:
If a write produces a value that causes another write, they are causally related.

```plaintext
X = 1
if(X > 0) {
    Y = 1
}
```

Causal consistency → all see X=1, Y=1 in same order.
Causal consistency

• Causally related writes seen by all processes in same order.
  • *Causally?*
Causal consistency

- Causally related writes seen by all processes in same order.
  - Causally?
  - Concurrent writes may be seen in different orders on different machines
Causal consistency

• Causally related writes seen by all processes in same order.
  • Causally?
  • Concurrent writes may be seen in different orders on different machines

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1: W(x)a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2: R(x)a W(x)b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3:</td>
<td>R(x)b R(x)a</td>
<td></td>
</tr>
<tr>
<td>P4:</td>
<td>R(x)a R(x)b</td>
<td>(a)</td>
</tr>
</tbody>
</table>
Causal consistency

- Causally related writes seen by all processes in same order.
  - \textit{Causally}?
  - \textit{Concurrent} writes may be seen in different orders on different machines

<table>
<thead>
<tr>
<th></th>
<th>W(x)a</th>
<th>W(x)b</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2:</td>
<td>R(x)a</td>
<td>W(x)b</td>
</tr>
<tr>
<td>P3:</td>
<td>R(x)b</td>
<td>R(x)a</td>
</tr>
<tr>
<td>P4:</td>
<td>R(x)a</td>
<td>R(x)b</td>
</tr>
</tbody>
</table>

(a)

Not permitted
Causal consistency

• Causally related writes seen by all processes in same order.
  • *Causally*
  • *Concurrent* writes may be seen in different orders on different machines

<table>
<thead>
<tr>
<th>P1: W(x)a</th>
<th>P2:</th>
<th>R(x)a</th>
<th>W(x)b</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3:</td>
<td></td>
<td>R(x)b</td>
<td>R(x)a</td>
</tr>
<tr>
<td>P4:</td>
<td></td>
<td>R(x)a</td>
<td>R(x)b</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>P1: W(x)a</th>
<th>P2:</th>
<th>W(x)b</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3:</td>
<td></td>
<td>R(x)b</td>
</tr>
<tr>
<td>P4:</td>
<td></td>
<td>R(x)a</td>
</tr>
</tbody>
</table>

(b)

Not permitted
Causal consistency

- Causally related writes seen by all processes in same order.
  - Causally?
  - Concurrent writes may be seen in different orders on different machines

<table>
<thead>
<tr>
<th>Process</th>
<th>Action 1</th>
<th>Action 2</th>
<th>Action 3</th>
<th>Action 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1:</td>
<td>W(x)a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2:</td>
<td>R(x)a</td>
<td>W(x)b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3:</td>
<td></td>
<td>R(x)b</td>
<td>R(x)a</td>
<td></td>
</tr>
<tr>
<td>P4:</td>
<td></td>
<td>R(x)a</td>
<td>R(x)b</td>
<td></td>
</tr>
</tbody>
</table>

(a) Not permitted

<table>
<thead>
<tr>
<th>Process</th>
<th>Action 1</th>
<th>Action 2</th>
<th>Action 3</th>
<th>Action 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1:</td>
<td>W(x)a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2:</td>
<td></td>
<td>W(x)b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3:</td>
<td></td>
<td>R(x)b</td>
<td>R(x)a</td>
<td></td>
</tr>
<tr>
<td>P4:</td>
<td></td>
<td>R(x)a</td>
<td>R(x)b</td>
<td></td>
</tr>
</tbody>
</table>

(b) Permitted
Dataflow
Dataflow

• MR is a *dataflow* engine
Dataflow

• MR is a *dataflow* engine
Dataflow

• MR is a *dataflow* engine
• So are Lots of others
  • Dryad
  • DryadLINQ
  • Dandelion
  • CIEL
  • GraphChi/PowerGraph/Pregel
  • Spark
Dataflow

• MR is a **dataflow** engine
• So are Lots of others
  • Dryad
  • DryadLINQ
  • Dandelion
  • CIEL
  • GraphChi/PowerGraph/Pregel
  • Spark
Spark faux quiz (5 min, any 2):

• What is the difference between transformations and actions in Spark?
• Spark supports a persist API. When should a programmer want to use it? When should she [not] use use the “RELIABLE” flag?
• Compare and contrast fault tolerance guarantees of Spark to those of MapReduce. How are[n’t] the mechanisms different?
• Is Spark a good system for indexing the web? For computing page rank over a web index? Why [not]?
• List aspects of Spark’s design that help/hinder multi-core parallelism relative to MapReduce. If the issue is orthogonal, explain why.
Collections and Iterators

class Collection<T> : IEnumerable<T>;

```csharp
// Diagram of Collection class

```
class Collection<T> : IEnumerable<T>;

public interface IEnumerable<T>  {
    IEnumerator<T> GetEnumerator();
}
Collections and Iterators

class Collection<T> : IEnumerable<T>;

public interface IEnumerable<T> {
    IEnumerator<T> GetEnumerator();
}

public interface IEnumerator<T> {
    T Current { get; }
    bool MoveNext();
    void Reset();
}
Collections and Iterators

class Collection<T> : IEnumerable<T>;

public interface IEnumerable<T> {
    IEnumerator<T> GetEnumerator();
}

public interface IEnumerator<T> {
    T Current { get; }
    bool MoveNext();
    void Reset();
}
DryadLINQ Data Model

Partition

.Net objects

Collection
DryadLINQ = LINQ + Dryad

Collection<T> collection;
bool IsLegal(Key k);
string Hash(Key);

var results = from c in collection
               where IsLegal(c.key)
               select new { Hash(c.key), c.value};
DryadLINQ = LINQ + Dryad

```csharp
Collection<T> collection;

bool IsLegal(Key k);
string Hash(Key);

var results = from c in collection
                where IsLegal(c.key)
                select new { Hash(c.key), c.value};
```

**Diagram:**
- Data flows to the `collection` node.
- The `collection` node processes the input data through a LINQ query.
- The processed data is then stored in the `results` node.
DryadLINQ = LINQ + Dryad

```csharp
Collection<T> collection;

bool IsLegal(Key k);
string Hash(Key);

var results = from c in collection
              where IsLegal(c.key)
              select new { Hash(c.key), c.value};
```

Data

Collection

results

Query plan (Dryad job)
DryadLINQ = LINQ + Dryad

```
Collection<T> collection;
bool IsLegal(Key k);
string Hash(Key);

var results = from c in collection
where IsLegal(c.key)
select new { Hash(c.key), c.value};
```
Language Summary
Language Summary

Where
Language Summary

Where
Language Summary

Where
Select
Language Summary

Where

Select
Language Summary

Where
Select
GroupBy
Language Summary

Where
Select
GroupBy
Language Summary

Where
Select
GroupBy
OrderBy
Language Summary

Where
Select
GroupBy
OrderBy
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Materialize
Language Summary

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Materialize
public static IQueryable<Pair> Histogram(IQueryable<LineRecord> input, int k) {
    var words = input.SelectMany(x => x.line.Split(' '));
    var groups = words.GroupBy(x => x);
    var counts = groups.Select(x => new Pair(x.Key, x.Count()));
    var ordered = counts.OrderByDescending(x => x.count);
    var top = ordered.Take(k);
    return top;
}
Iterative Computations: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page’s rank to
   \[ \sum_{i \in \text{neighbors}} \frac{\text{rank}_i}{|\text{neighbors}_i|} \]

\[ \text{links} = \text{RDD of (url, neighbors) pairs} \]
\[ \text{ranks} = \text{RDD of (url, rank) pairs} \]

for (i <= 1 to ITERATIONS) {
    ranks = links.join(ranks).flatMap {
        (url, (links, rank)) =>
        links.map(dest => (dest, rank/links.size))
    }.reduceByKey(_ + _)
}
# RDD Operations

## Transformations
(Define a new RDD)

- map
- filter
- sample
- union
- groupByKey
- reduceByKey
- join
- persist/cache
- ...

## Parallel operations
(Return a result to driver)

- reduce
- collect
- count
- save
- lookupKey
- ...
- ...
 RDD Operations

<table>
<thead>
<tr>
<th>Transformations</th>
<th>Parallel operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(define a new RDD)</td>
<td>(return a result to driver)</td>
</tr>
<tr>
<td>map</td>
<td>reduce</td>
</tr>
<tr>
<td>filter</td>
<td>collect</td>
</tr>
<tr>
<td>sample</td>
<td>count</td>
</tr>
<tr>
<td>union</td>
<td>save</td>
</tr>
<tr>
<td>groupByKey</td>
<td>lookupKey</td>
</tr>
<tr>
<td>reduceByKey</td>
<td>...</td>
</tr>
<tr>
<td>join</td>
<td></td>
</tr>
<tr>
<td>persist/cache</td>
<td></td>
</tr>
</tbody>
</table>

Where
Select
GroupBy
OrderBy
Aggregate
Join
Apply
Materialize
RDD Fault Tolerance

• RDDs maintain *lineage* information that can be used to reconstruct lost partitions

• Ex:
  
  ```scala
  cachedMsgs = textFile(...).filter(_.contains("error"))
  .map(_.split('t')(2))
  .persist()
  ```
## RDDs vs Distributed Shared Memory

<table>
<thead>
<tr>
<th>Concern</th>
<th>RDDs</th>
<th>Distrib. Shared Mem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads</td>
<td>Fine-grained</td>
<td>Fine-grained</td>
</tr>
<tr>
<td>Writes</td>
<td>Bulk transformations</td>
<td>Fine-grained</td>
</tr>
<tr>
<td>Consistency</td>
<td>Trivial (immutable)</td>
<td>Up to app / runtime</td>
</tr>
<tr>
<td>Fault recovery</td>
<td>Fine-grained and low-overhead using lineage</td>
<td>Requires checkpoints and program rollback</td>
</tr>
<tr>
<td>Straggler mitigation</td>
<td>Possible using speculative execution</td>
<td>Difficult</td>
</tr>
<tr>
<td>Work placement</td>
<td>Automatic based on data locality</td>
<td>Up to app (but runtime aims for transparency)</td>
</tr>
</tbody>
</table>