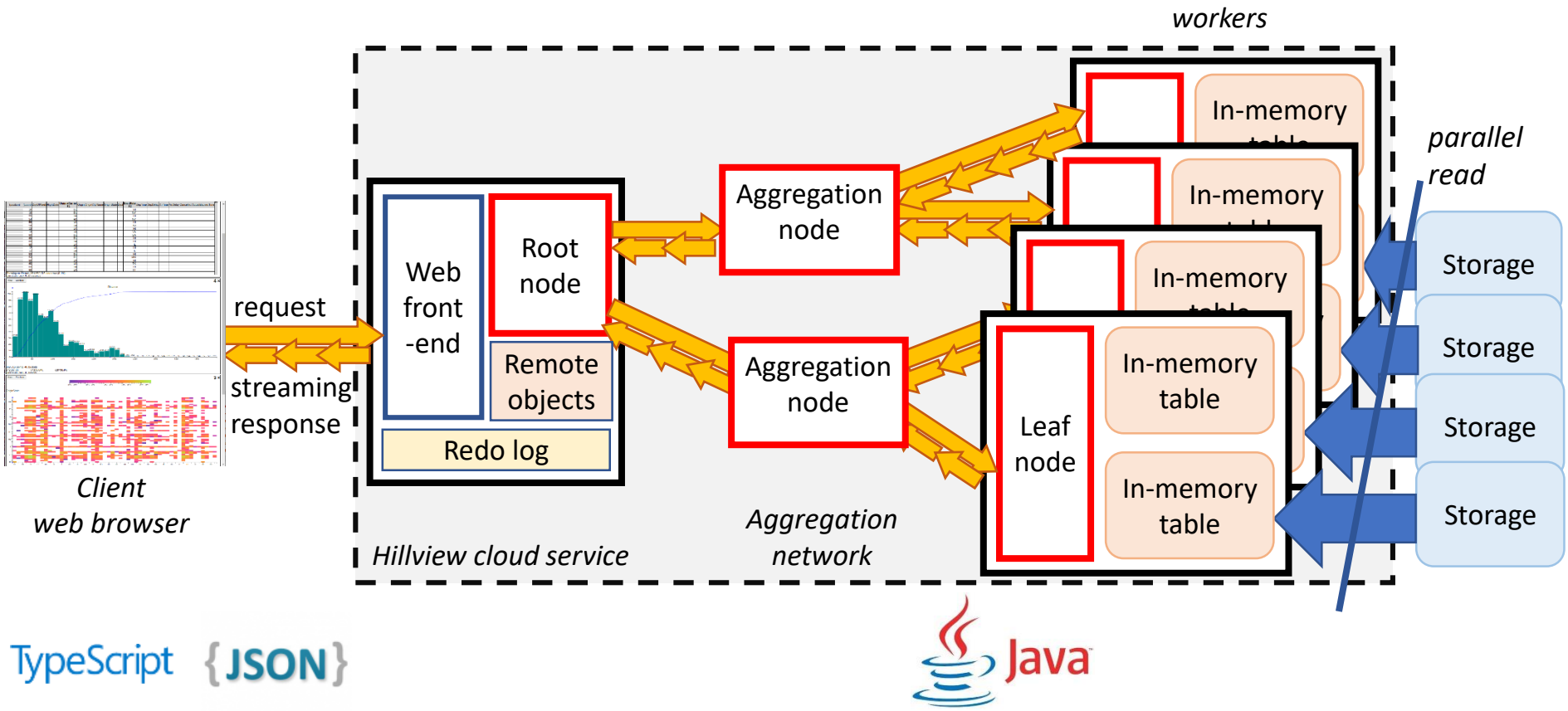
Hillview APIs

November 6, 2018

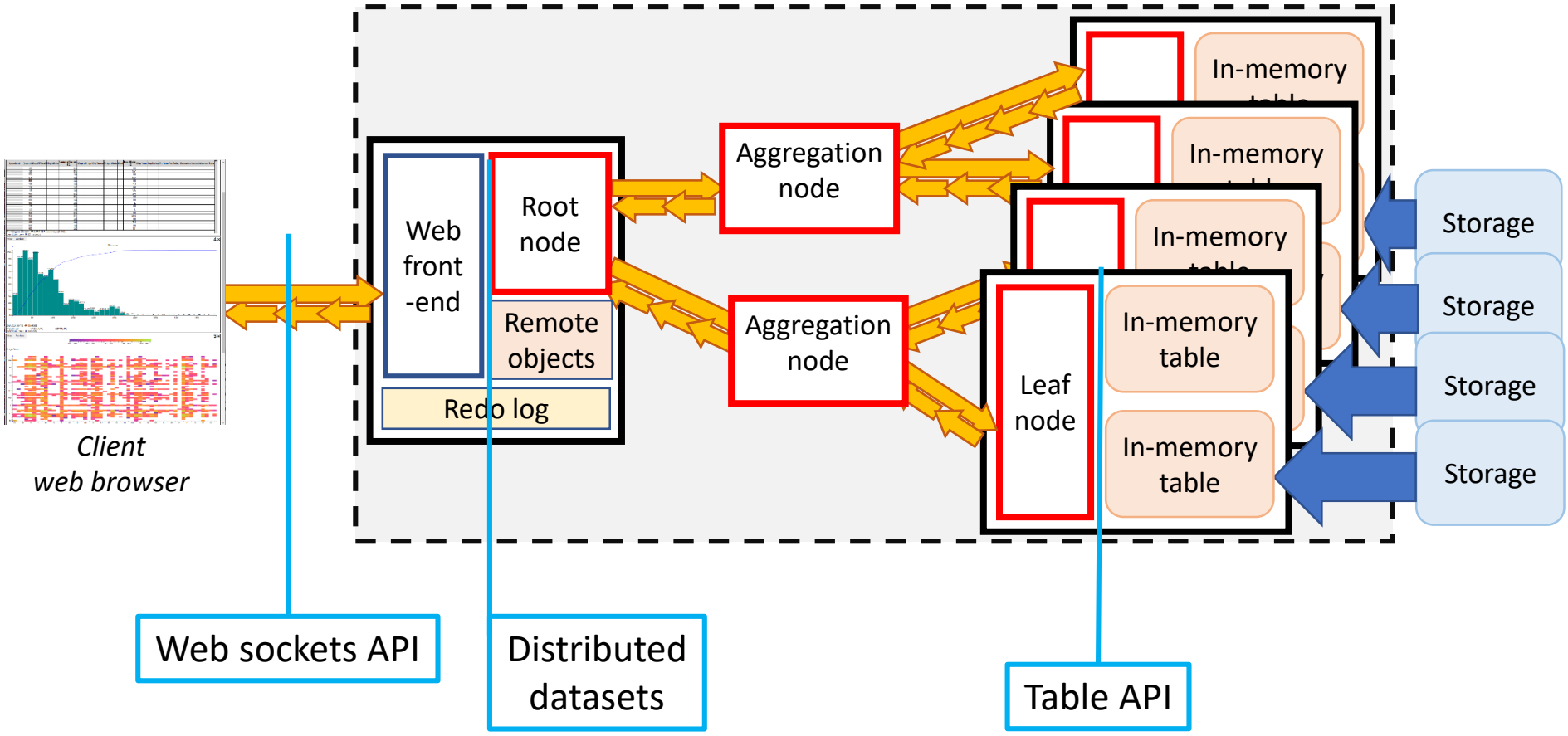
Mihai Budiu – VMWare Research

Some of the APIs presented are simplified for pedagogical reasons, but very slightly.

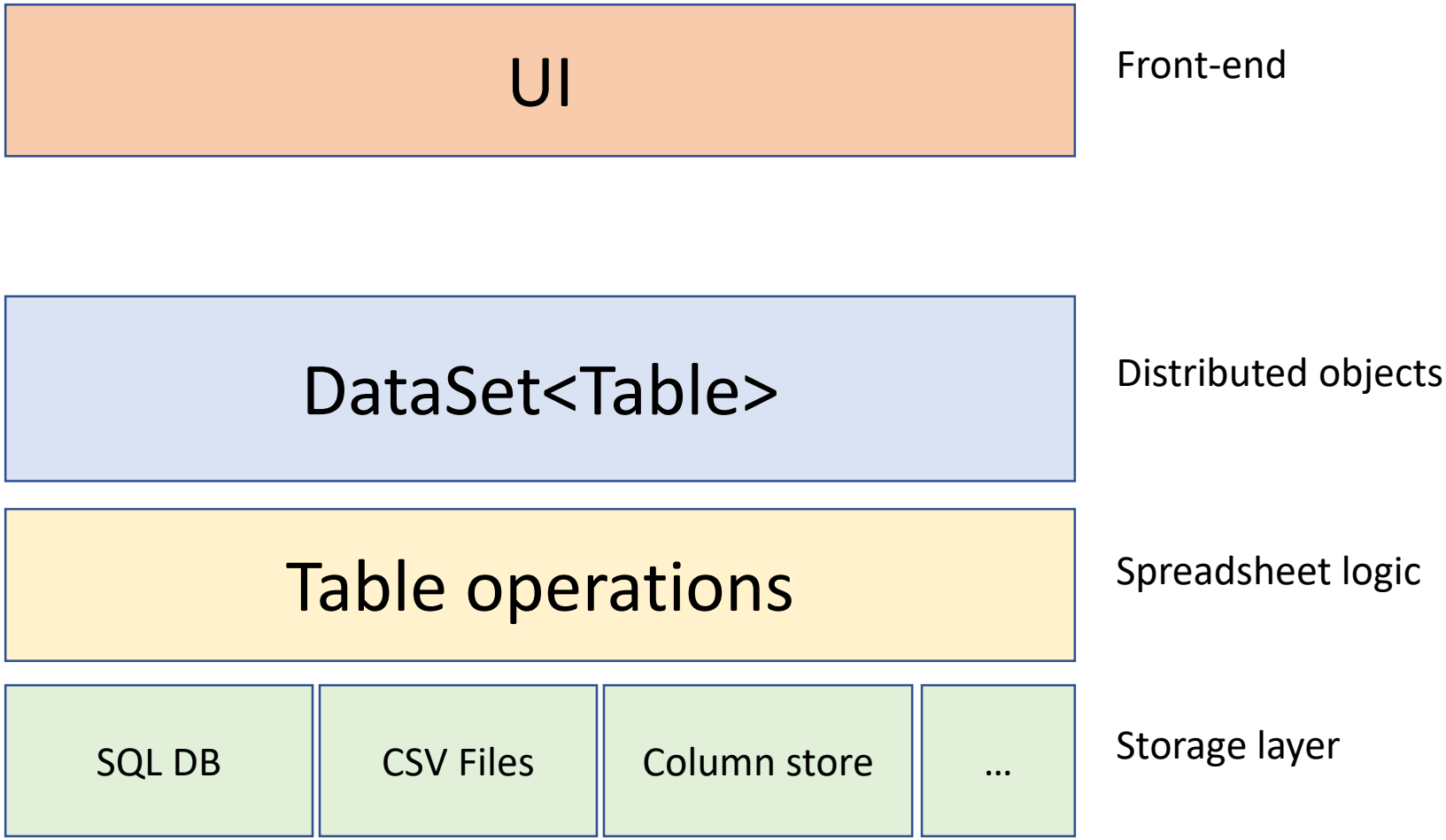
Hillview System architecture



Core APIs


Client
web browser
Web
front
-end
Root
node
Remote
objects
Redo log
Aggregation
node
Aggregation
node
In-memory
In-memory
In-memory
Leaf
node
In-memory
table
In-memory
table
Storage
Storage
Storage
Storage
Web sockets API
Distributed
datasets
Table API

Spreadsheet architecture



reactivex.io

ReactiveX Introduction Docs Languages Resources Community

ReactiveX

An API for asynchronous programming with observable streams

Choose your platform

WEB DEVELOPMENT

March 2012

acmqueue

Your Mouse is a Database

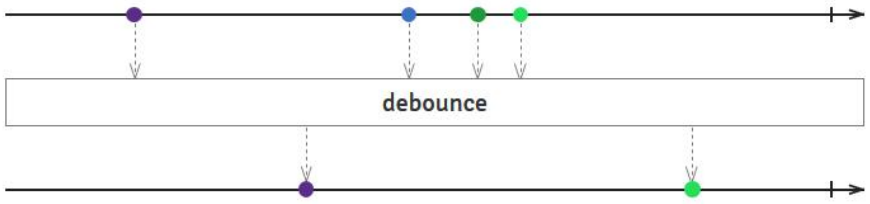
Web and mobile applications are increasingly composed of asynchronous and realtime streaming services and push notifications.

Erik Meijer

Hillview streaming APIs are based on Reactive Streams

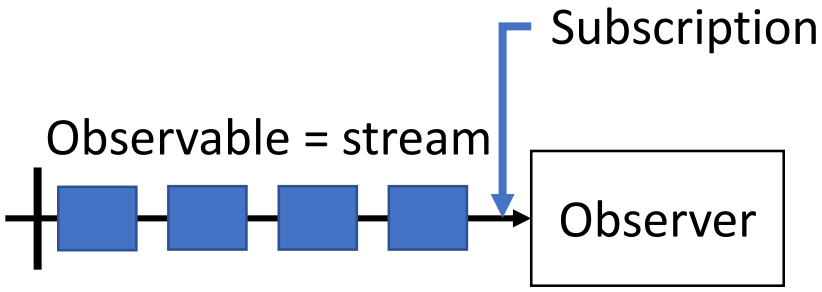
The Observer pattern done right

ReactiveX is a combination of the best ideas from
the Observer pattern, the Iterator pattern, and functional programming

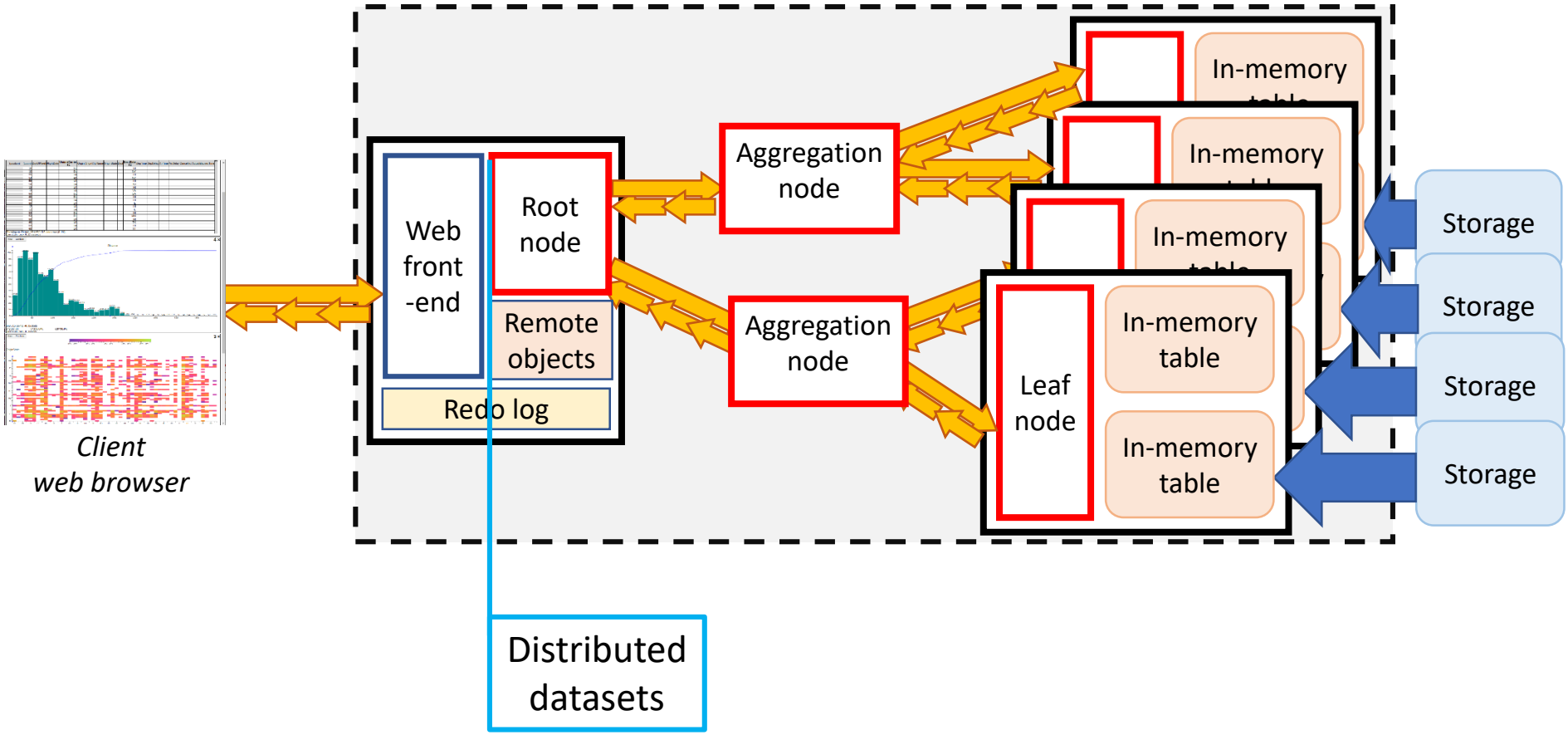


Reactive streams (ReactiveX)

```
interface Observable<T> {
  Subscription subscribe(Observer<T> observer);
}
interface Observer<T> {
  void onNext(T value);
  void onError(Throwable error);
  void onCompleted();
}
```



Distributed Dataset API



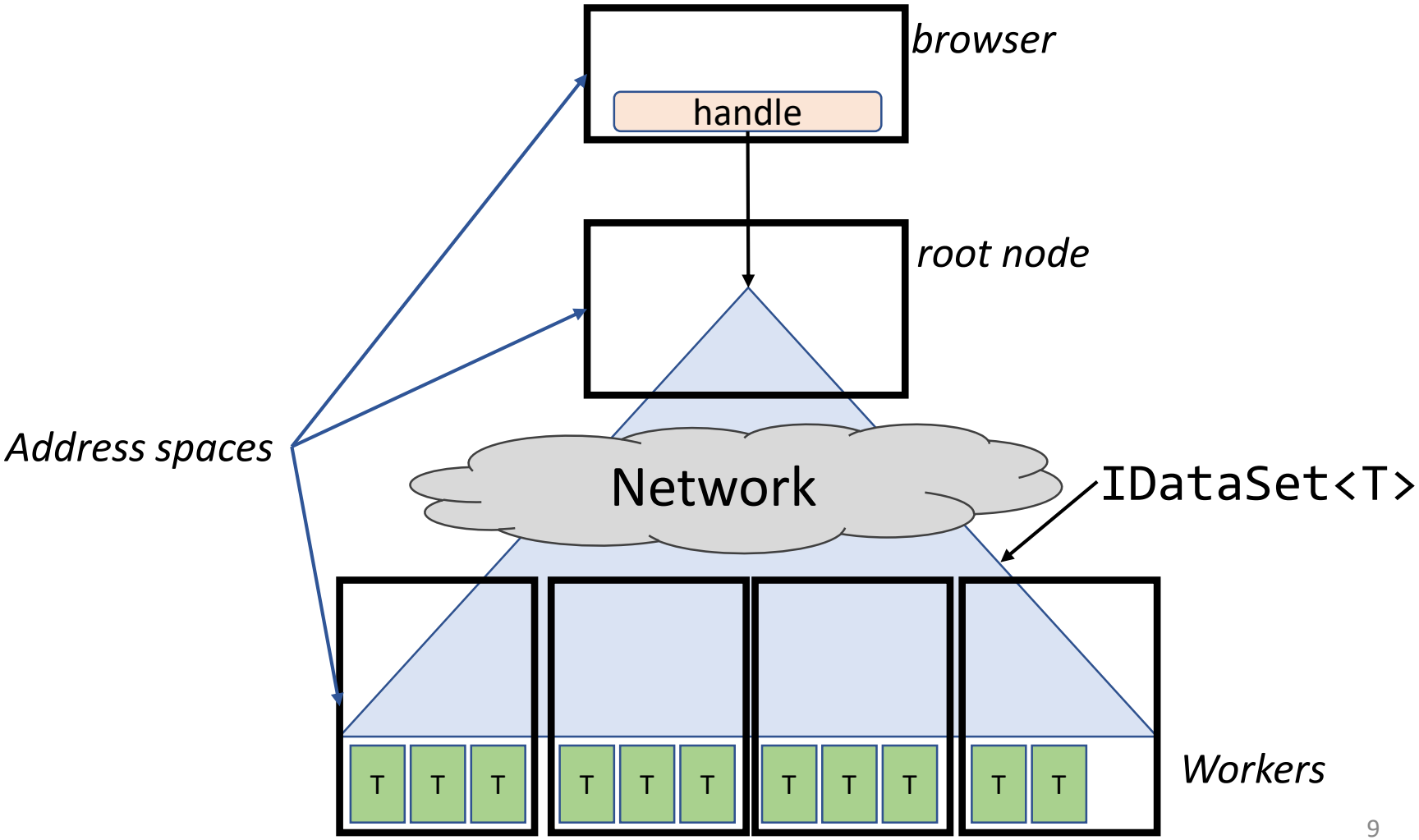
IDataSet<T>

Think of IDataSet<T> as a Collection<T>

- Distributed
- Immutable
- Collection

In general T is not Serializable

DataSets span multiple address spaces



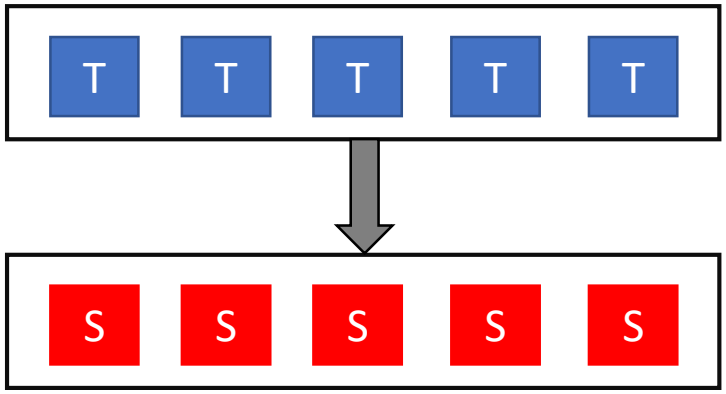
IDataSet Core High-level API

```
interface IDataSet<T> {
      R sketch(Aggregator<T,R> sk);
      IDataSet<S> map(Mapper<T, S> map);
      IDataSet<Pair<T,S>> zip(IDataSet<S>);
}
```

This is the high-level idea; refined on the following slides.

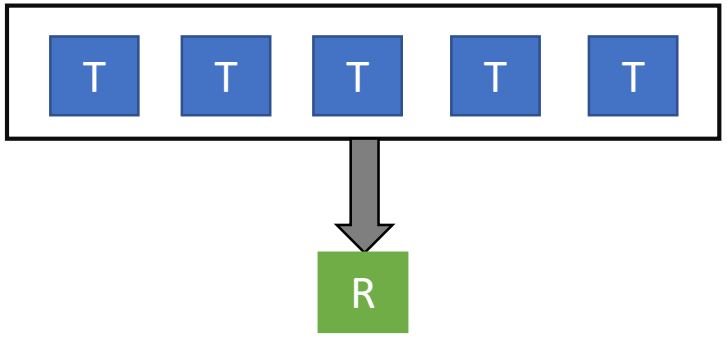
https://github.com/vmware/hillview/blob/master/platform/src/main/java/org/hillview/dataset/api/

Map<T, S>

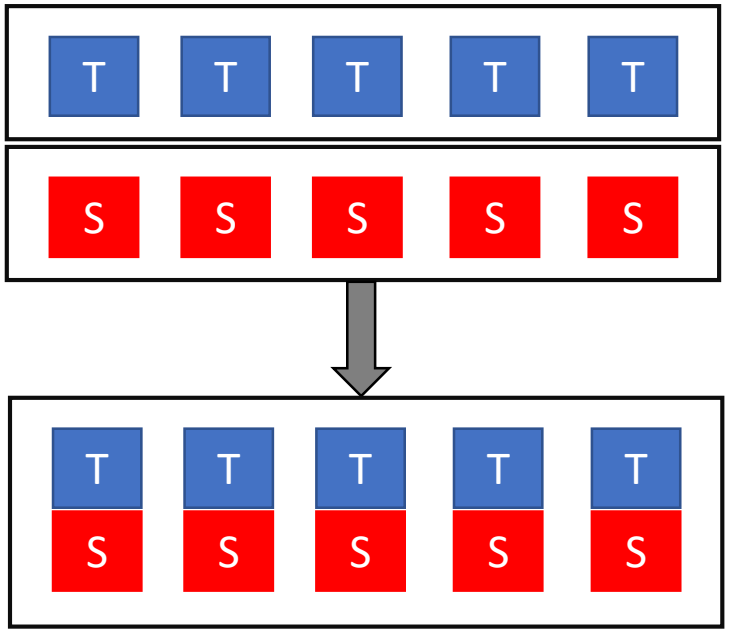


Sketch<T, R>

- Aggregation with associative and commutative function



Zip<T, S>



Collections of Partitions

#	Name	Type	Allows missing
1	DayOfWeek	Integer	true
2	FlightDate	Date	true
3	UniqueCarrier	Category	true
4	Origin	String	true
5	OriginCityName	String	true
6	OriginState	Category	true
7	Dest	Category	true
8	DestState	Category	true
9	DepTime	Integer	true
10	DepDelay	Double	true
11	ArrTime	Integer	true
12	ArrDelay	Double	true
13	Cancelled	Double	true
14	ActualElapsedTime	Double	true
15	Distance	Double	true

#	Name	Type	Allows missing
1	DayOfWeek	Integer	true
2	FlightDate	Date	true
3	UniqueCarrier	Category	true
4	Origin	String	true
5	OriginCityName	String	true
6	OriginState	Category	true
7	Dest	Category	true
8	DestState	Category	true
9	DepTime	Integer	true
10	DepDelay	Double	true
11	ArrTime	Integer	true
12	ArrDelay	Double	true
13	Cancelled	Double	true
14	ActualElapsedTime	Double	true
15	Distance	Double	true

#	Name	Type	Allows missing
1	DayOfWeek	Integer	true
2	FlightDate	Date	true
3	UniqueCarrier	Category	true
4	Origin	String	true
5	OriginCityName	String	true
6	OriginState	Category	true
7	Dest	Category	true
8	DestState	Category	true
9	DepTime	Integer	true
10	DepDelay	Double	true
11	ArrTime	Integer	true
12	ArrDelay	Double	true
13	Cancelled	Double	true
14	ActualElapsedTime	Double	true
15	Distance	Double	true

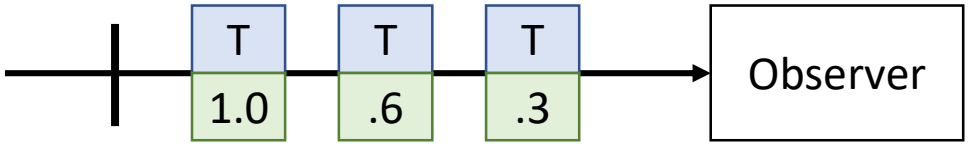
IDataSet<DataPartition>

Values in an IDataSet tend to be complex objects, e.g., whole tables.

Partial results = progress reporting

```
interface PartialResult<T> {
    T      data;
    double done;  // percent, in [0,1]
}
```

```
Observable<PartialResult<T>>
```



I will abbreviate Observable<PartialResult<T>> to OPR<T>.

Streaming Dataset API

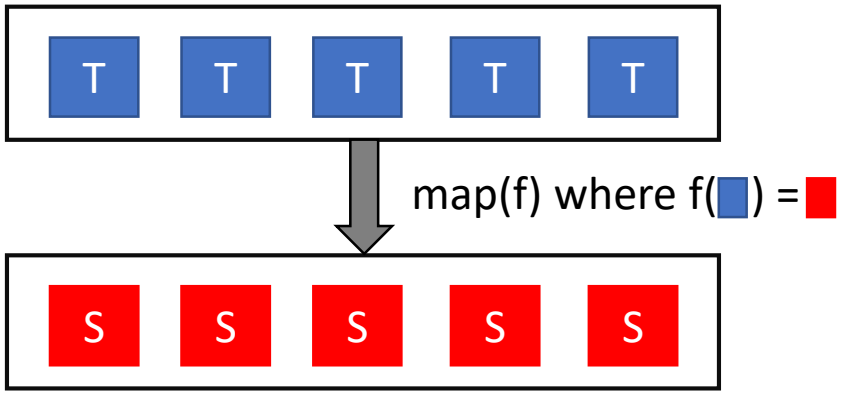
```
interface IDataSet<T> {
        OPR<R> sketch(ISketch<T,R> sk);
        OPR<IDataSet<S>> map(IMap<T, S> map);
        OPR<IDataSet<S>> flatmap(IMap<T, List<S>> map);
        OPR<IDataSet<Pair<T,S>>> zip(IDataSet<S>);
        OPR<ControlMessage.StatusList> manage(ControlMessage m);
}
```


OPR<T> is Observable<PartialResult<T>>

Note: IDataSet does **not** provide an iteration API!

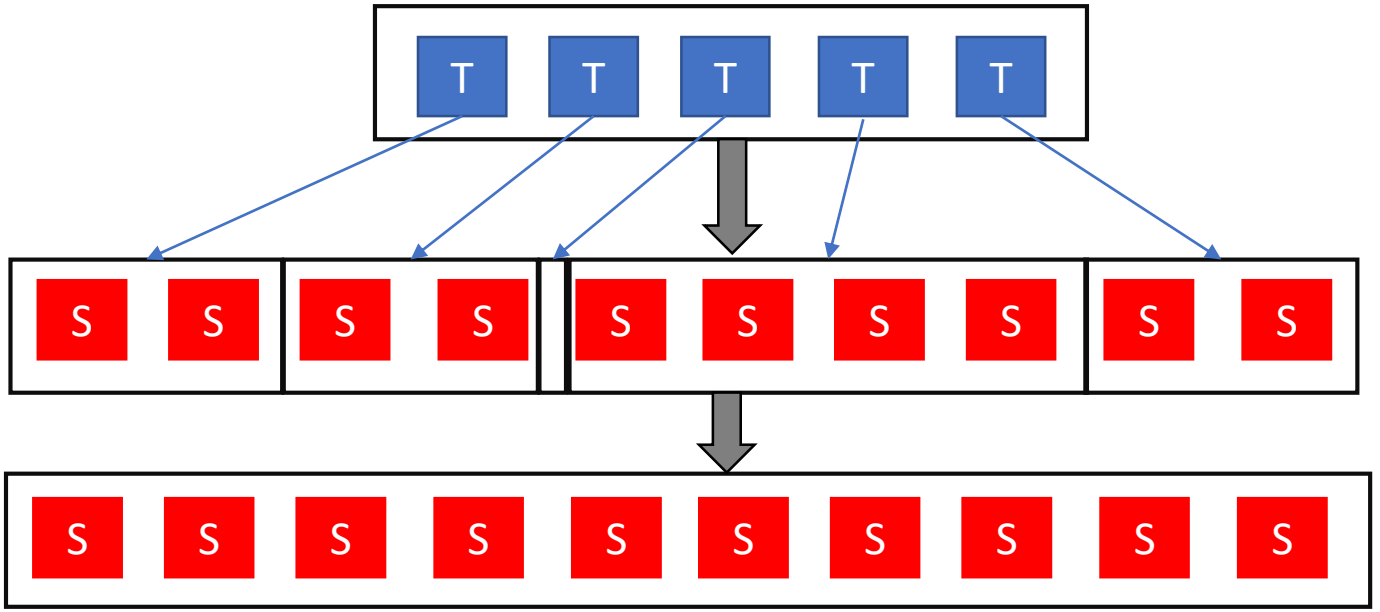
Mappers

```
public interface IMap<T, S>
        extends Serializable {
    S apply(T data);
}
interface IDataSet<T> {
    OPR<IDataSet<S>> map(IMap<T, S> map);
}
```



Flatmap

```
interface IDataSet<T> {
      OPR<IDataSet<S>> flatmap(
            IMap<T, List<S>> map);
}
```

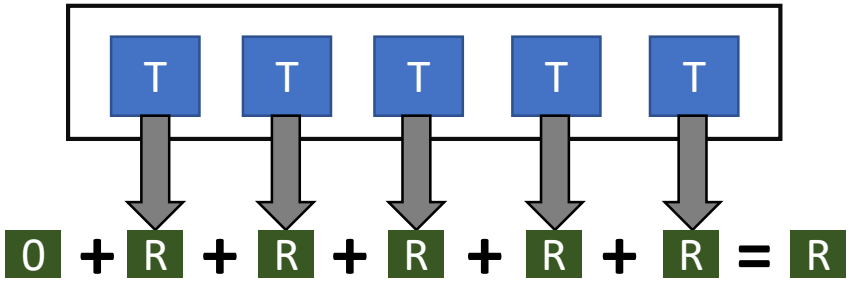


Aggregations (sketches)

```
interface IMonoid<R> {
        R zero();
        R add(R left, R right);
}


interface ISketch<T,R> extends IMonoid<R> implements Serializable {
    R sketch(T data);
}


interface IDataSet<T> {
        OPR<R> sketch(ISketch<T,R> sk);
}
```

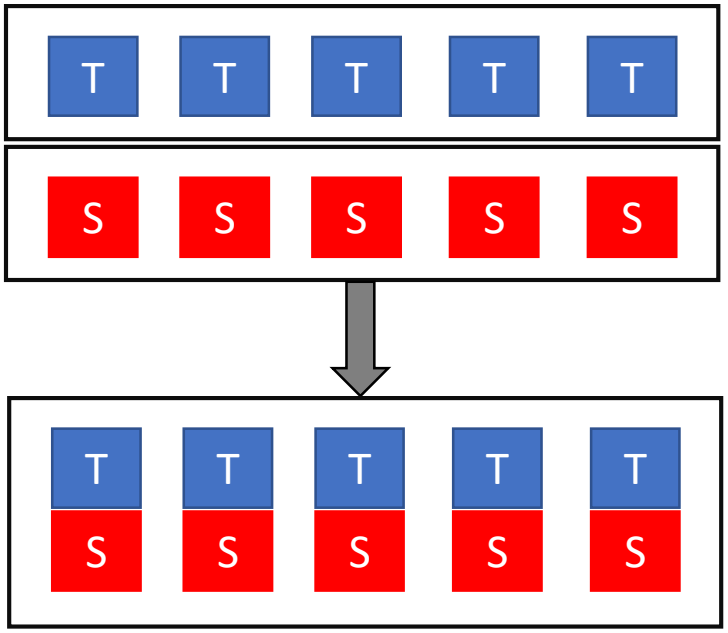


R must be serializable and "small"

Making 2 datasets into 1

```
interface IDataSet<T> {
    OPR<IDataSet<Pair<T,S>>> zip(IDataSet<S>);
}
```

This is useful to support binary operations between datasets, e.g., set intersection. Both datasets must have the same "shape".



Management API

- A variety of operations for monitoring and benchmarking
  - Find Java heap memory use on each worker
  - Ping each worker
  - Change memoization (of query results)
  - Delete memoized results
  - Delete cached datasets

In-Memory Table API

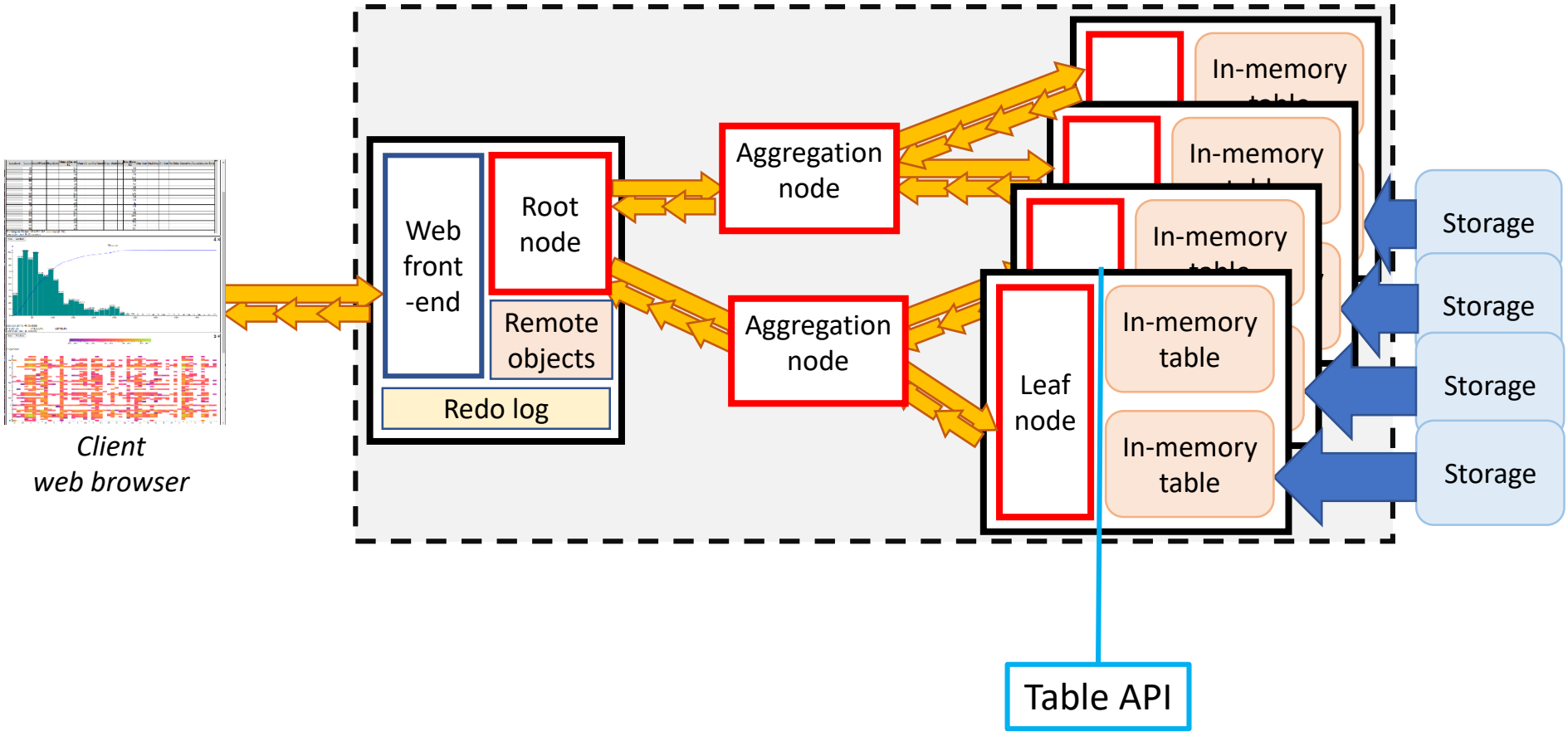


Table data storage

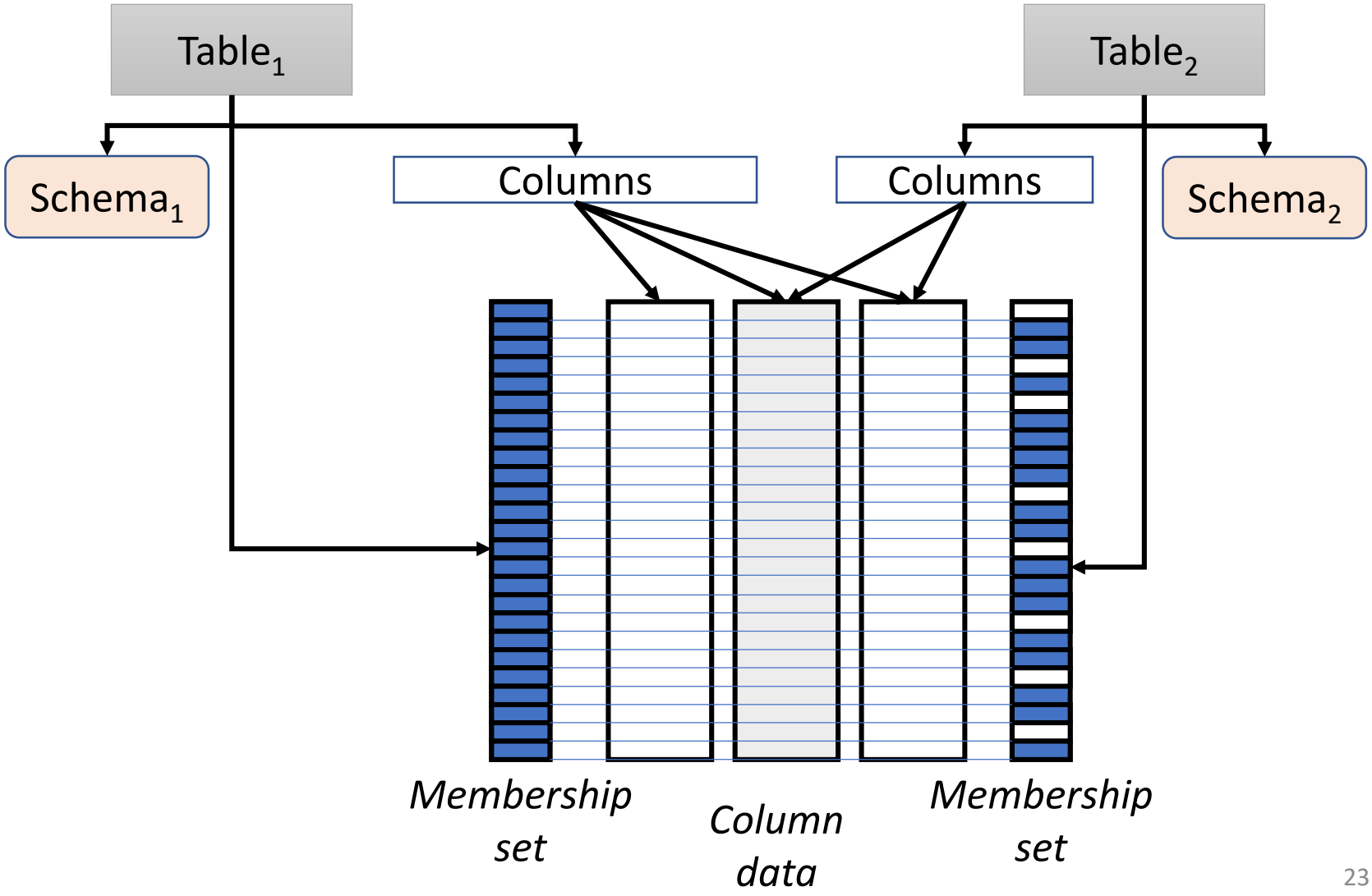


Table Classes

- `Schema`: data schema
- `IColumn`: data access APIs
- `IMembershipSet`: iteration and sampling over rows
- `IRow`: row-level API
- `Table`: a list of columns

https://github.com/vmware/hillview/tree/master/platform/src/main/java/org/hillview/table/api

Schema

```
public class Schema implements
     Serializable, IJson {
      HashMap<String, ColumnDescription> columns;
}
```

Name, data type

IColumn

```
public interface IColumn {
    ColumnDescription getDescription();
    int sizeInRows();

    boolean isMissing(int rowIndex);
    double getDouble(int rowIndex);
    int getInt(int rowIndex);
    String getString(int rowIndex);
    Instant getDate(int rowIndex);
    Duration getDuration(int rowIndex);
}
```

Only 1 should work for a given column.

IRow

```
public interface IRow extends Map<String, Object> {
    int columnCount();
    List<String> getColumnNames();
    boolean isMissing(String colName);


    Object getObject(String colName);
    Instant getDate(String colName);
    Duration getDuration(String colName);
    String getString(String colName);
    int getInt(String colName);
    double getDouble(String colName);
}
```

Only 1 should work for a given column.

Iterating over rows

```
public interface IRowIterator {
    // Returns -1 when iteration is completed;
    // else it returns the index of the next row.
    int getNextRow();
}
```

Each Table is limited to 2^{31} rows.
Not a problem, since a dataset can have lots of Tables, even on one machine.

MembershipSet

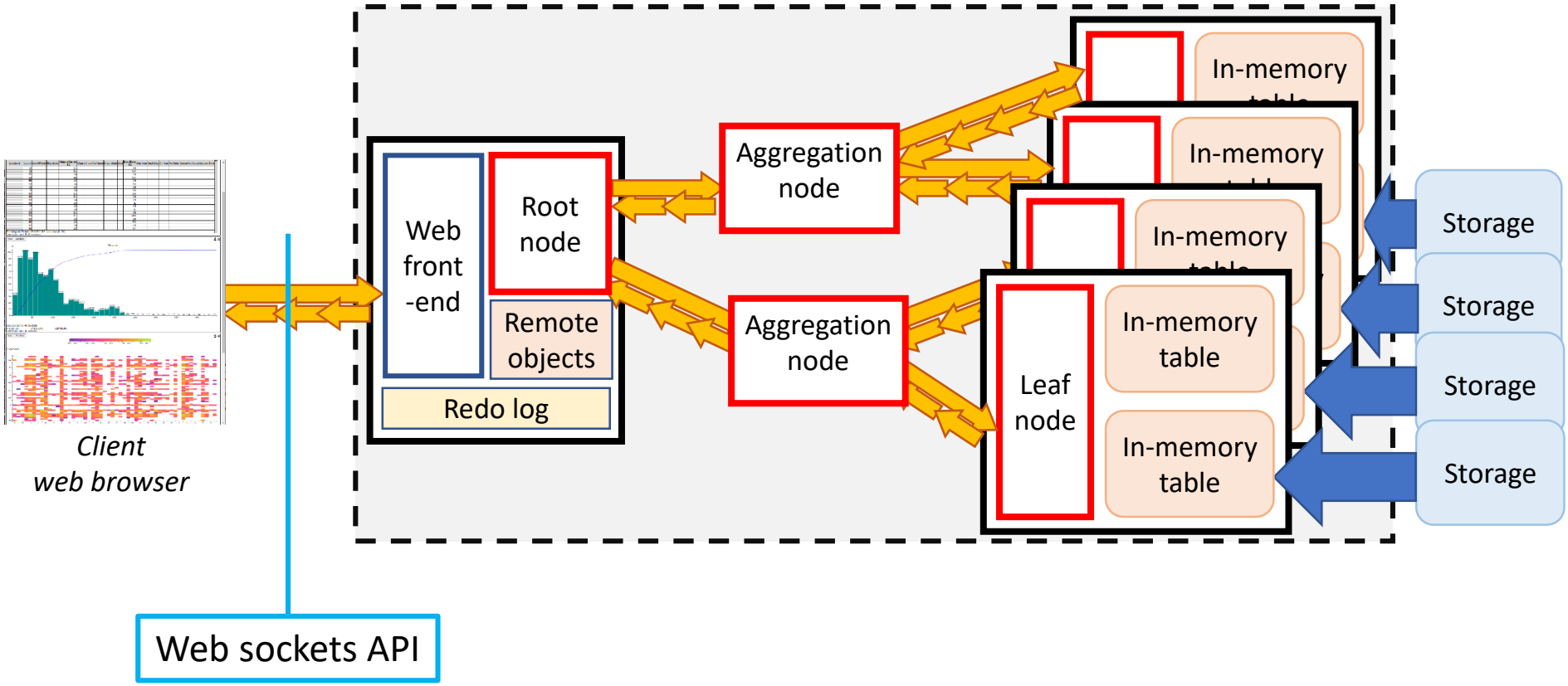
```
public interface IMembershipSet {
    int getSize();
    IRowIterator getIterator();
    boolean isMember(int rowIndex);

    IRowIterator getIteratorOverSample(double rate);
    IMembershipSet sample(double rate);
    IMembershipSet union(IMembershipSet other);
}
```

Tables

```
class Table {
    HashMap<String, IColumn> columns;
    Schema schema;
    IMembershipSet members;
    IColumnLoader columnLoader;
    List<IColumn> getLoadedColumns(
        List<String> columns);
}
```

Core APIs


Client
web browser
Web sockets API
Web
front
-end
Root
node
Remote
objects
Redo log
Aggregation
node
Aggregation
node
Leaf
node
In-memory
table
In-memory
table
In-memory
table
In-memory
table
Storage
Storage
Storage
Storage

Client-server messages

- Data serialized as JSON in both directions
- Using GSON to deserialize in to Java objects
- Using web sockets to communicate
- Client => Server
  - Invoke a method in a Java Class
- Server => Client
  - An Observable (stream) of replies

Client => Server

```
{
  objectId:"2b8cd056-…",
  method:"getDataRanges1D",
  arguments:[
     {
       cd:{
         name:"FlightDate",
         kind:"Date"
       },
       seed:0,
       stringsToSample:1253
     }
   ],
   "requestId":37
}
```

JSON

```
class RangeArgs {
  ColumnDescription cd;
  long seed;
  int stringsToSample;
}

@HillviewRpc
public void getDataRanges1D(
   RpcRequest req,
   RpcRequestContext context) {
   RangeArgs[] args = req.parseArgs(
      RangeArgs[].class);
   …
}
```

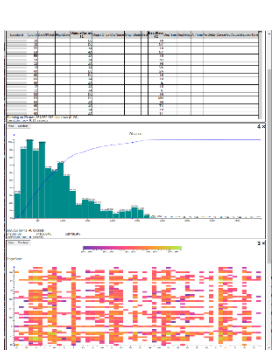
Java

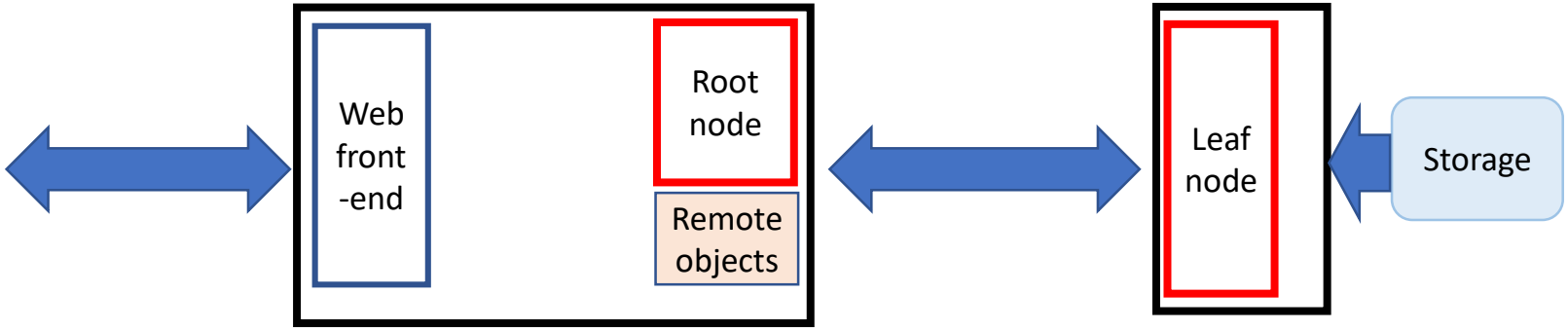
Server => Client

- Client is written in TypeScript
- When calling a remote method, client supplies an Observer that can consume the results
- The results are a stream of JSON objects

```
abstract class Receiver<T> implements
    Rx.Observer<PartialResult<T>>
```

Flow for loading data





initial.findFiles

initial.flatMap(findFiles)

files = initial.scan_fs()

files.getFileSize

files.sketch(fileSize)

size = files.size()

files.loadTable

files.map(loadFiles)

table = files.parse()

table.getSchema

table.sketch(getSchema)

table.getSchema()

tableView.display(
schema)

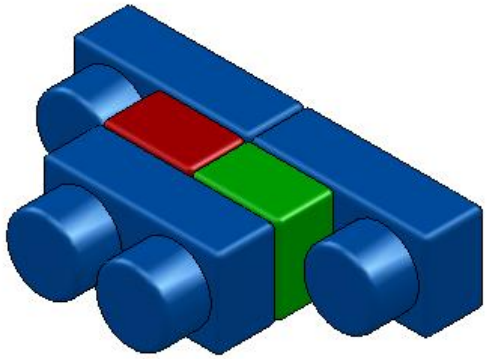
JavaScript objects

IDataSets

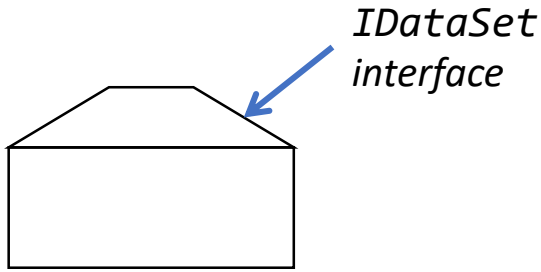
Java objects

Backup slides

Dataset objects

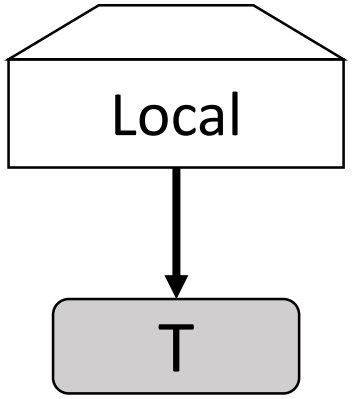


- Implement `IDataSet<T>`
- Identical interfaces on top and bottom
- Can be stacked arbitrarily
- Modular construction of distributed systems



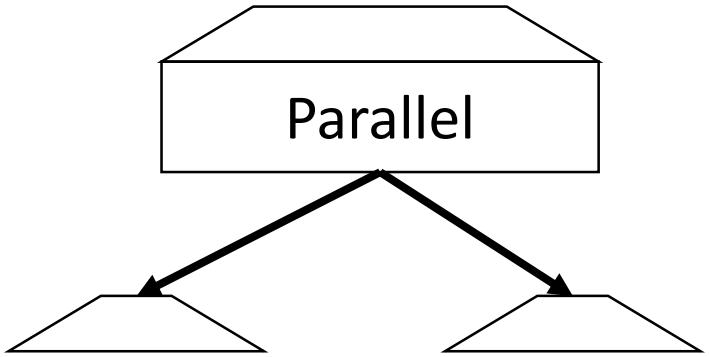
`LocalDataset<T>`

- Contains a reference to an object of type T in the same address space
- Directly executes operations (`map`, `sketch`, `zip`) on object



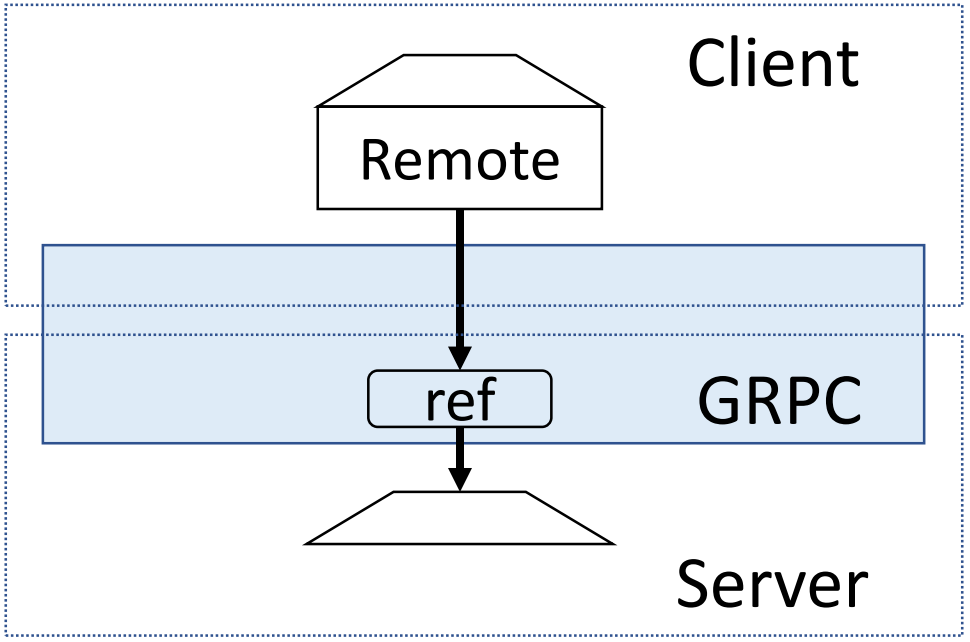
`ParallelDataset<T>`

- Has a number of children of type `IDataSet<T>`
- Dispatches operations to all children
  - `sketch` adds the results of children

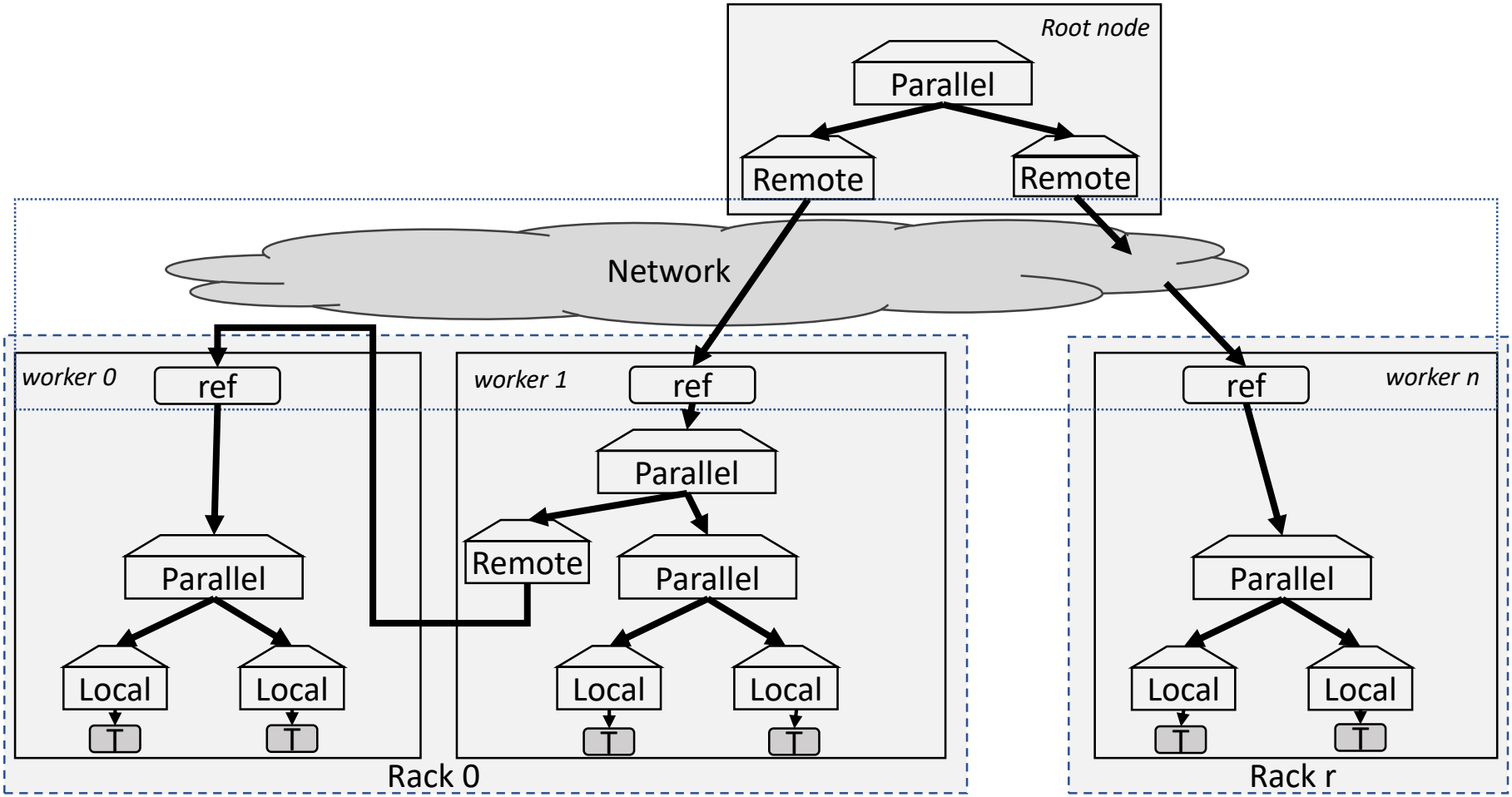


RemoteDataset<T>

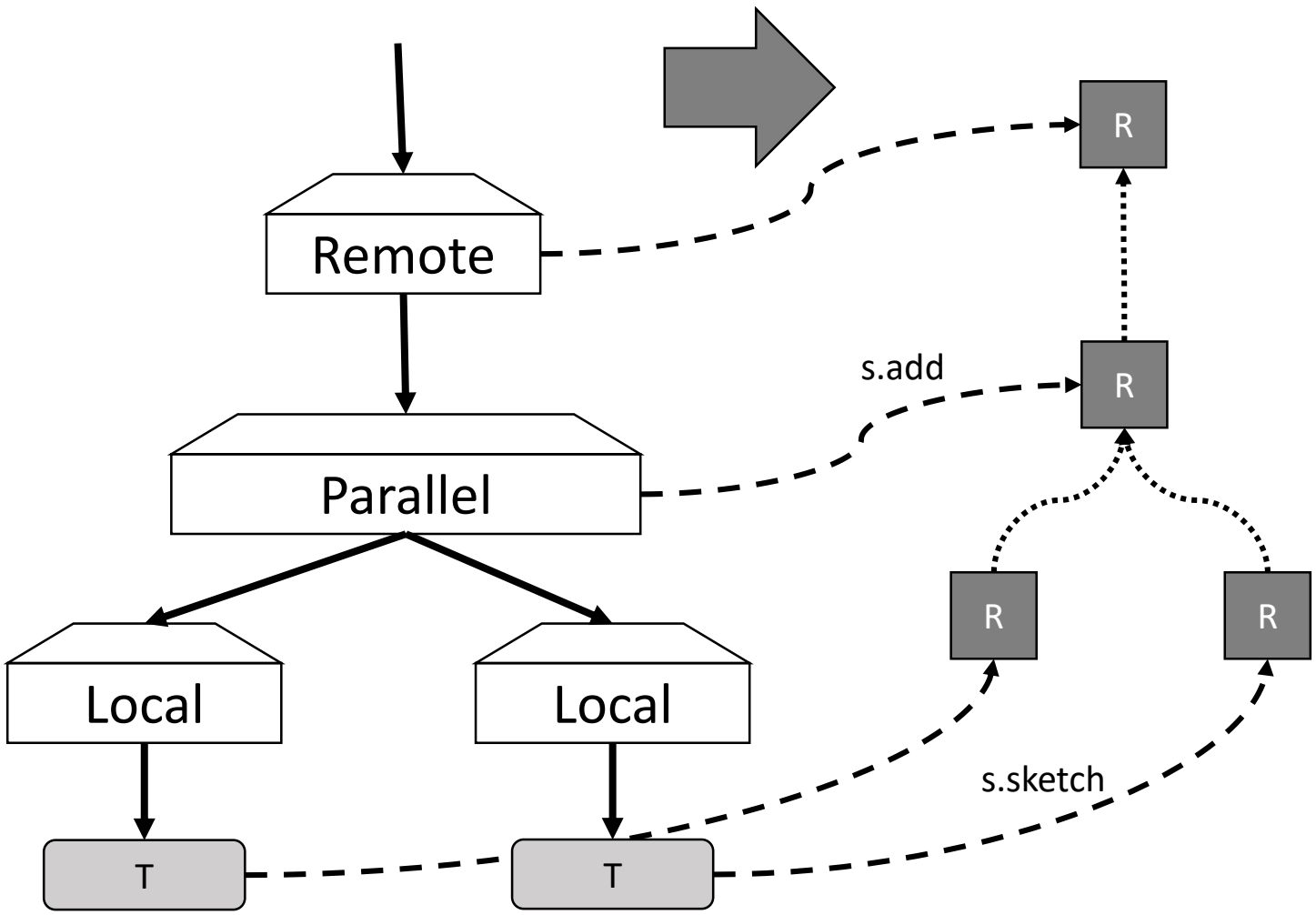
- Has a reference to an IDataSet<T> in another address space
- The only component that deals with the network
- Built on top of GRPC



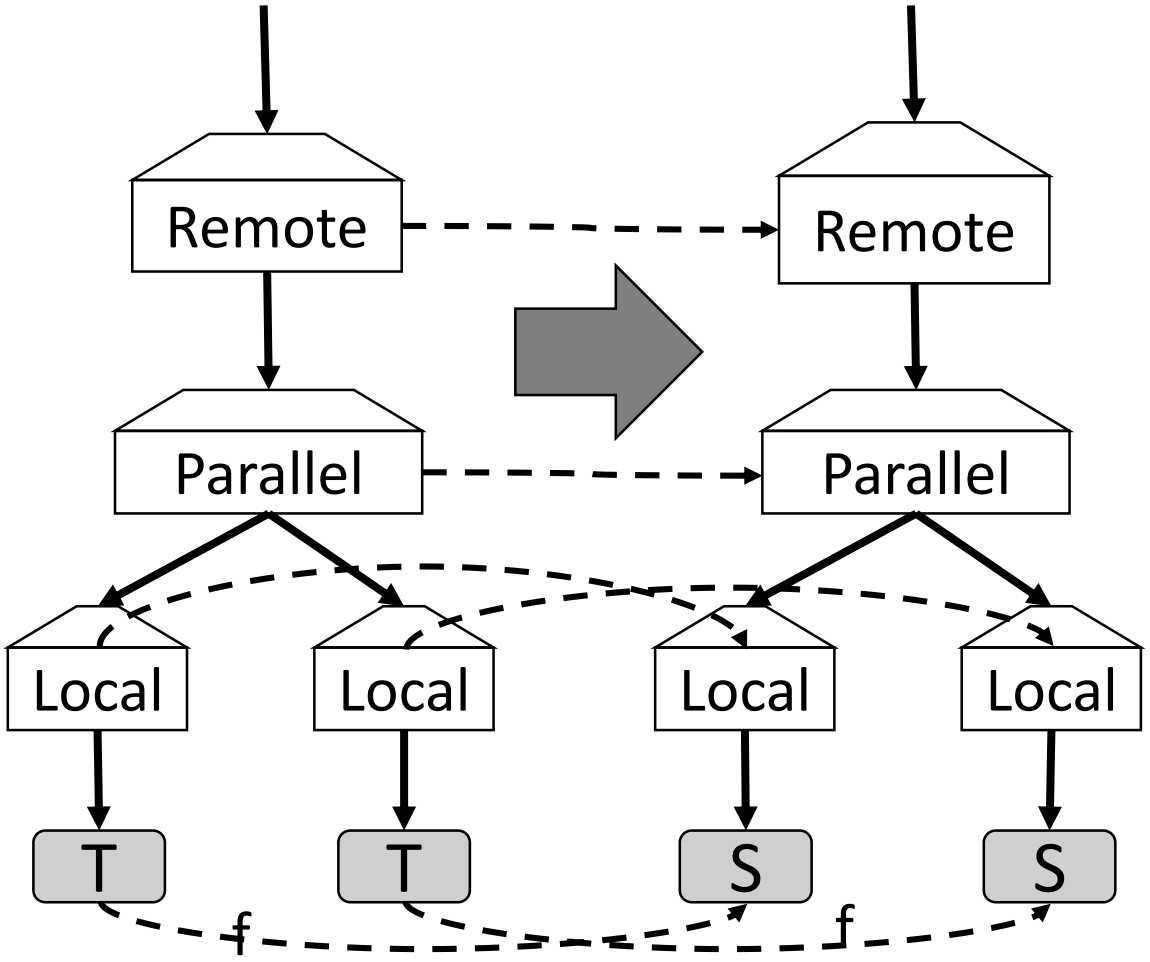
A distributed dataset



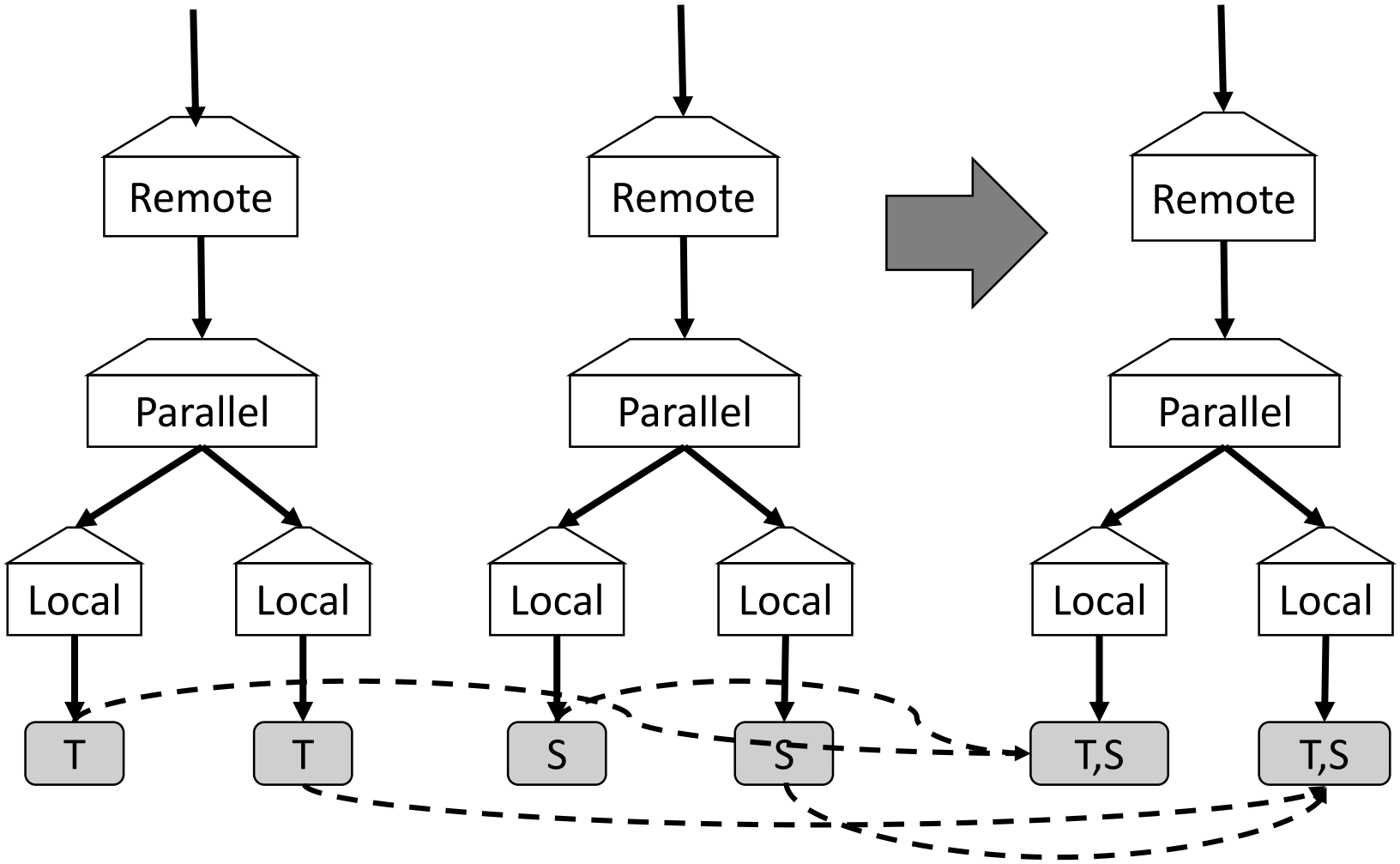
sketch(s)



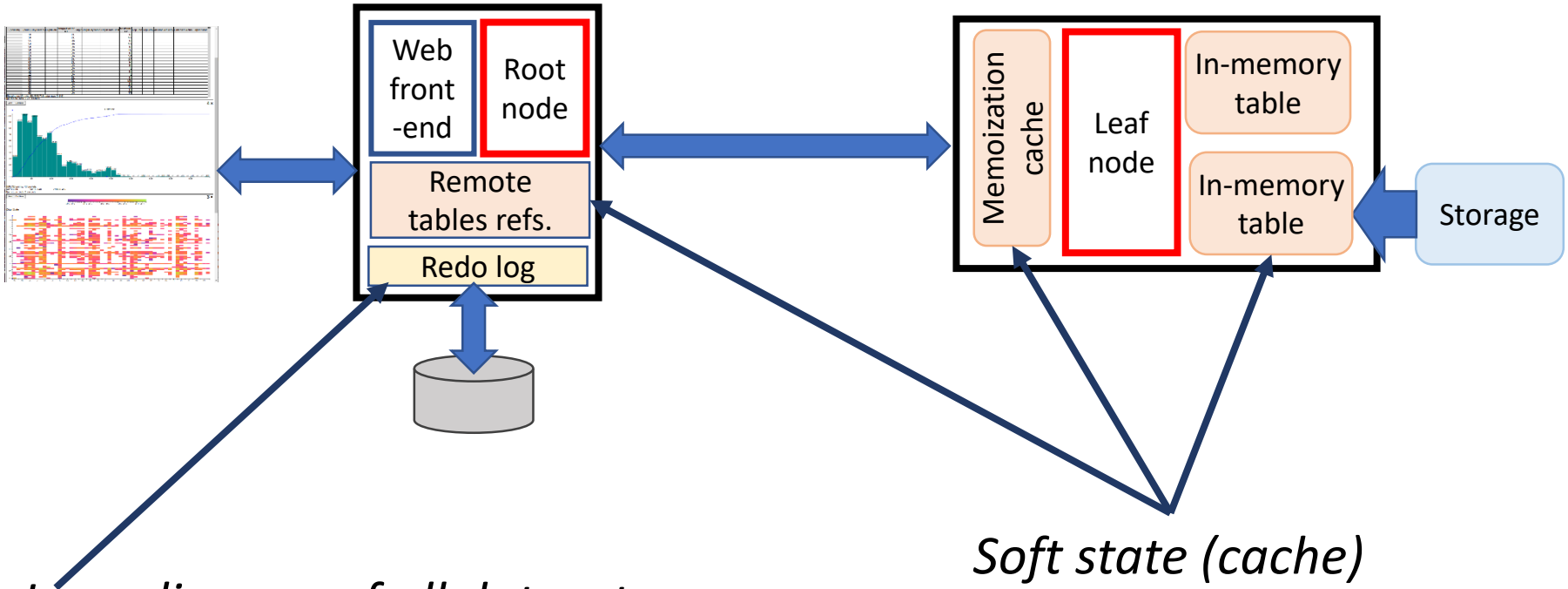
map(f)


Remote
Remote
Parallel
Parallel
Local
Local
Local
Local
T
T
S
S
f
f

zip


Remote
Parallel
Local
Local
T
T
Remote
Parallel
Local
Local
S
S
Remote
Parallel
Local
Local
T,S
T,S

Distributed Memory Management



- *Log = lineage of all datasets*
- *Log = JSON messages received from client*
- *Replaying the log reconstructs all soft-state*
- *Log can be replayed as needed*