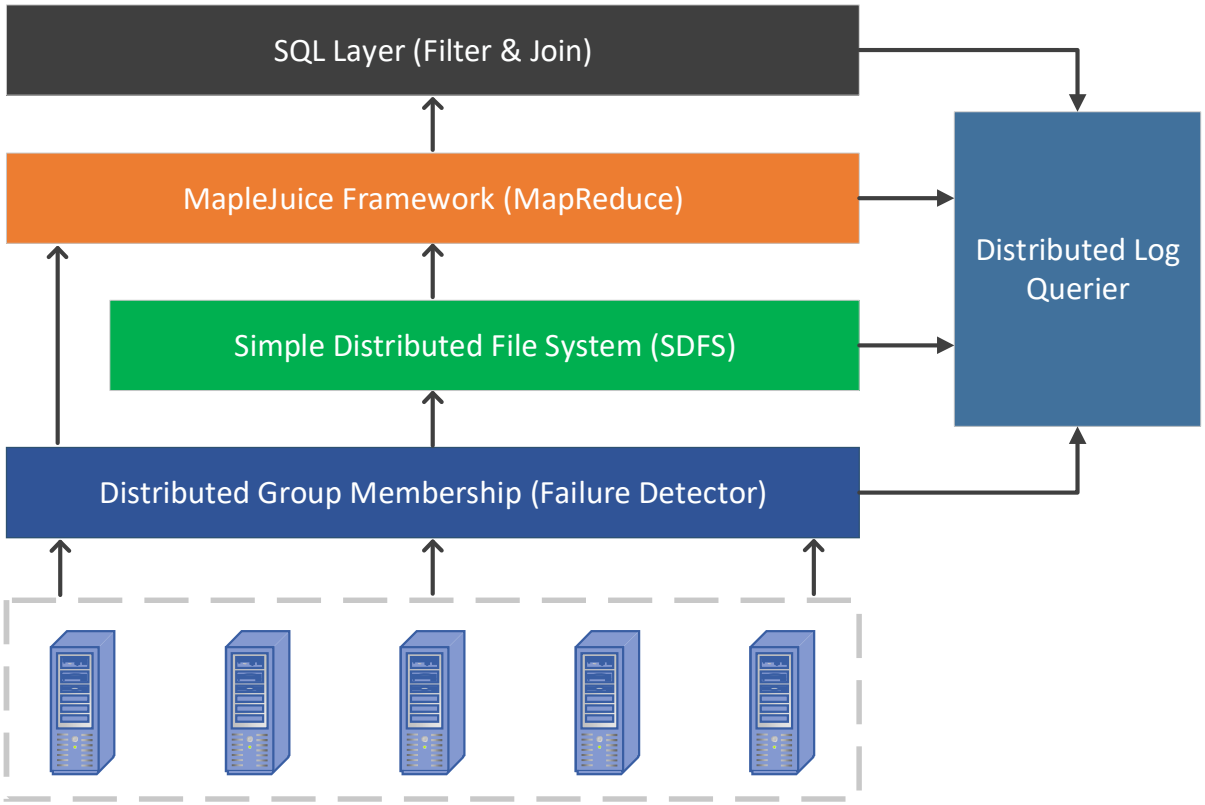
# MapleJuice

A Parallel, Distributed Batch Processing Framework

Amaan Khan

# MapleJuice High-Level Architecture




- Built the following from scratch:
  - Failure Detector
  - Distributed File System
  - MapReduce-like Framework
  - SQL Layer implemented using the MapleJuice framework
  - Distributed Log Querier used for debugging the other components

- **Timeline:** Semester-long project in our distributed systems class
- **Team:** 2-person project
- **Language:** Go

# MapleJuice: A Distributed Batch Processing Framework using the MapReduce Paradigm

- **Maple (i.e., Map):**
  - processes chunks of data in parallel. Outputs (key, value) pairs
- **Juice (i.e., Reduce):**
  - Aggregates and combines results from the Map phase.
- **Desired Properties**
  - Scalability: efficiently process massive datasets by distributing workload across many machines
  - Fault Tolerance: automatically handles failures and restarts tasks upon failures
  - Abstraction: abstracts away the distributed computing from the user

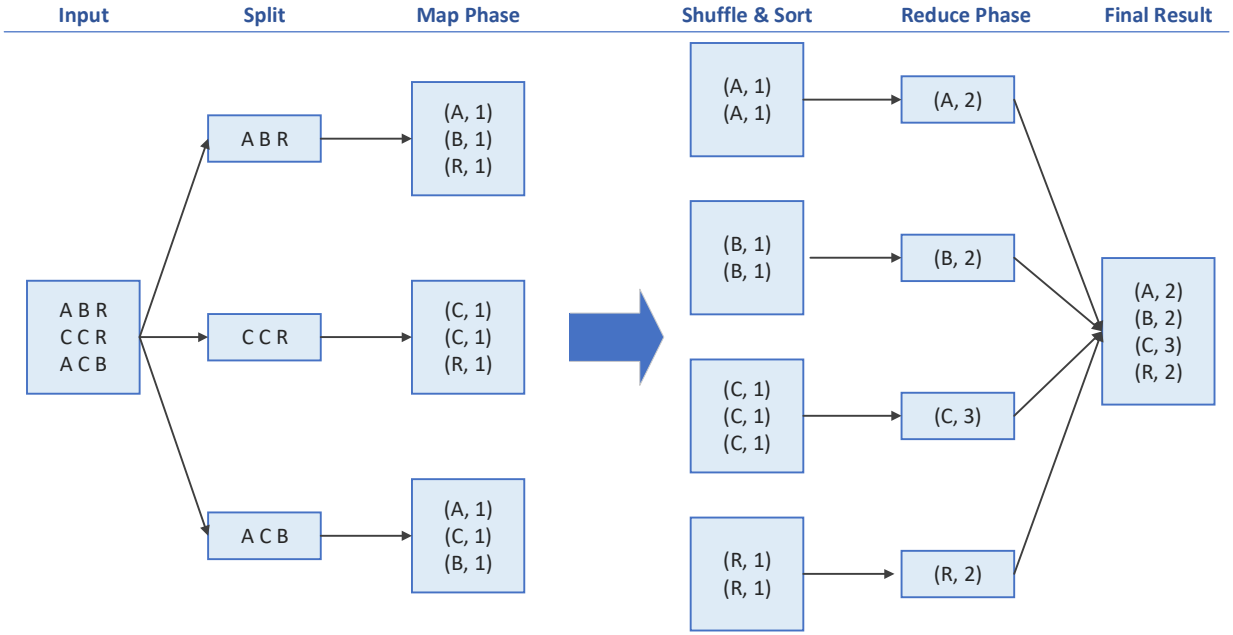


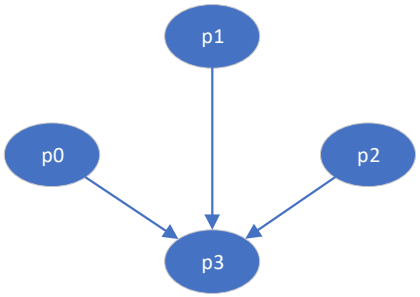

Diagram showing how MapReduce is used to gather letter frequencies from an input file

# Distributed Group Membership Service

Desirable Properties:

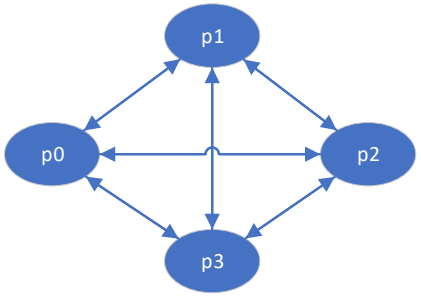
- **Completeness**: each failure is detected
- **Accuracy**: there is no mistaken detection
- **Speed**: detect failures quickly
- **Scale**: Equal Load on each member & no single point of failure

Examples of heartbeating methods for failure detection: (These are not used in MapleJuice)



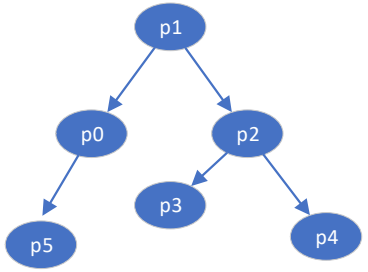

Centralized Heartbeating

(Unequal Load)




All-to-All Hearbeating

(Excessive Bandwidth)




Non-Fault Tolerant Multicast Heartbeating

(Better… but not fault tolerant)

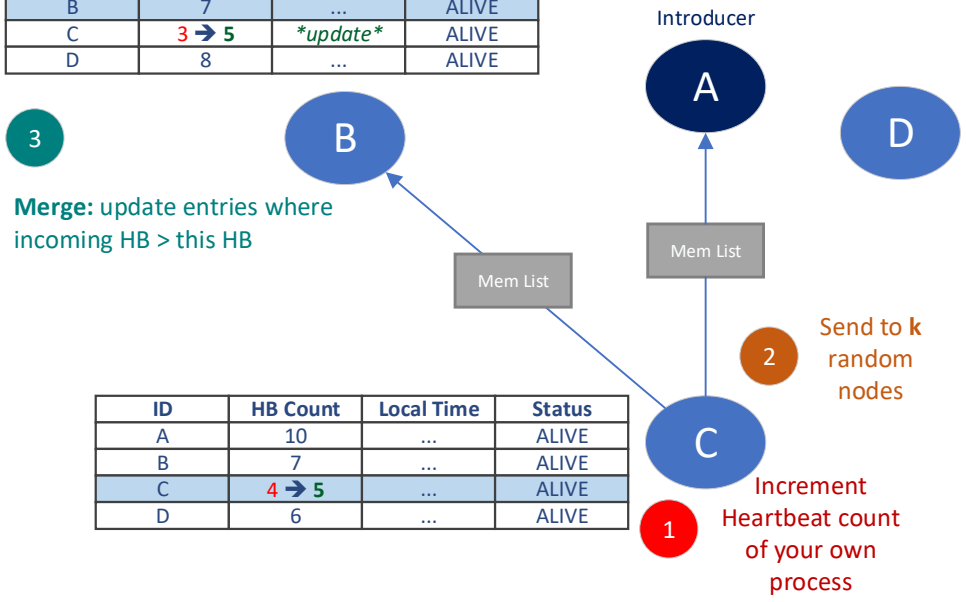
**Solution used in MapleJuice:** Gossip-Style Failure Detection

- No single point of failure, equal load, eventual consistency, inherently fault tolerant

# Gossip-Style Failure Detection Protocol (applied in MapleJuice)

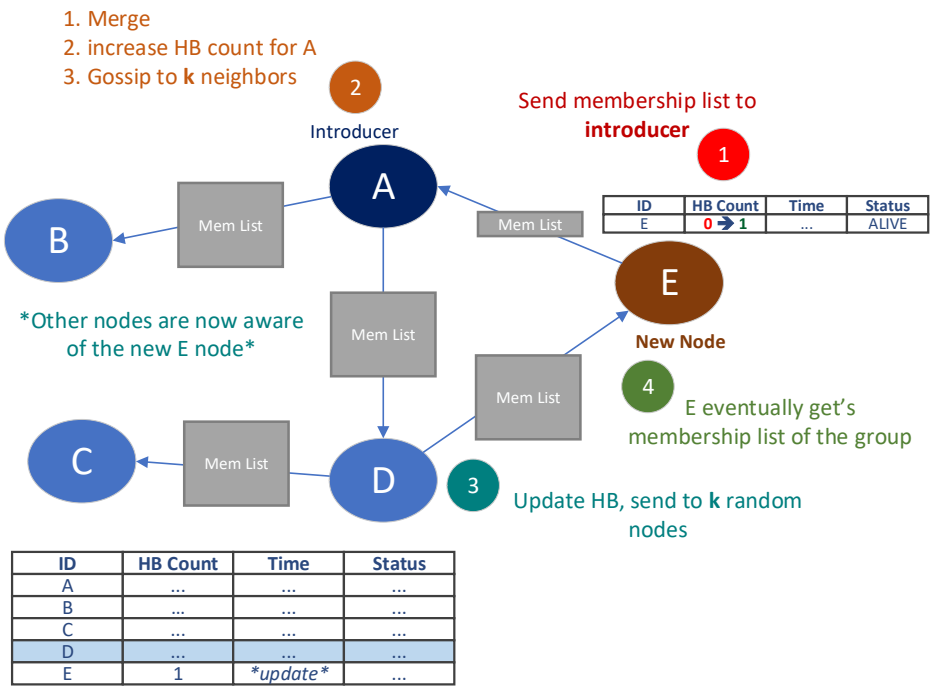
## Basic Gossiping + Merging

| ID | HB Count | Local Time | Status |
|---|---|---|---|
| A | 8 → 10 | *update* | ALIVE |
| B | 7 | ... | ALIVE |
| C | 3 → 5 | *update* | ALIVE |
| D | 8 | ... | ALIVE |




| ID | HB Count | Local Time | Status |
|---|---|---|---|
| A | 10 | ... | ALIVE |
| B | 7 | ... | ALIVE |
| C | 4 → 5 | ... | ALIVE |
| D | 6 | ... | ALIVE |

## How a new node joins




| ID | HB Count | Time | Status |
|---|---|---|---|
| E | 0 → 1 | ... | ALIVE |

| ID | HB Count | Time | Status |
|---|---|---|---|
| A | ... | ... | ... |
| B | ... | ... | ... |
| C | ... | ... | ... |
| D | ... | ... | ... |
| E | 1 | *update* | ... |

- **Failure Detection**: mark node as failed if its *Last Updated Time* has not been updated in **Tfail** seconds
- O(log n) dissemination
- Node Id = IP address + Timestamp

- Fail-Stop Model
- Gossip every **Tgossip**
- Detects failure in 5 seconds

# Simple Distributed File System (SDFS)

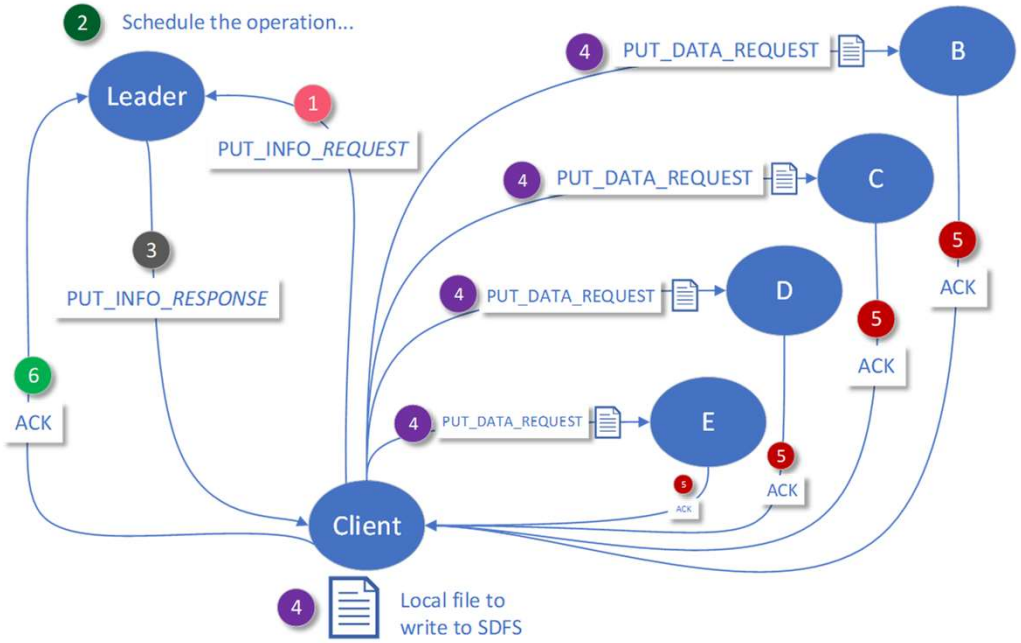
## Features & Properties of SDFS in MapleJuice Project

- Flat-file system
- Fault Tolerant up to 3 simultaneous failures
- Mutual Exclusion rules:
  - Cannot have a reader and writer simultaneously
  - Only 1 writer can write into a file
  - Up to 2 simultaneous readers can read from a file
- Starvation free rules:
  - No **writer** waits for more than 4 consecutive **reads**
  - No **reader** waits for more than 4 consecutive **writes**
- Assumes no leader failure
- Overwrites entire file on updates (cannot append)
- Strong Consistency
  - W=ALL, R=ONE
- Last writer wins policy

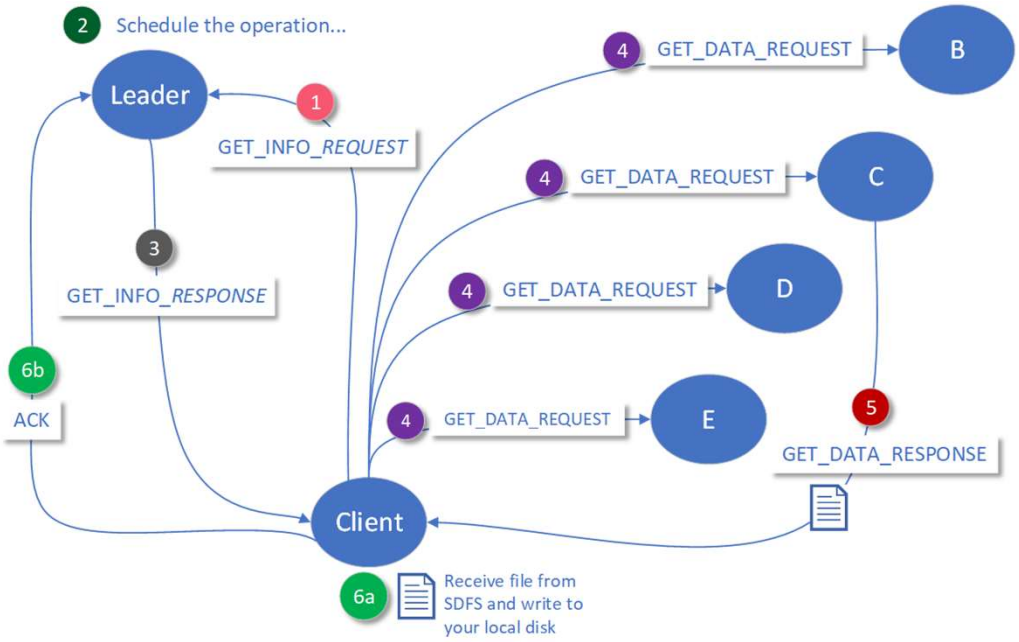
## Commands:

- **put** <local_filename> <sdfs_filename>
- **get** <sdfs_filename> <local_filename>
- **delete** <sdfs_filename>
- **ls** <sdfs_filename>
- **store**

# SDFS Write




# SDFS Read



# SDFS Re-Replication

- Re-replication is on a *per-file* basis
- Reads & writes for a file are suspended during re-replication
- If re-replication fails due to another process failure, it is reinitiated

## Legend

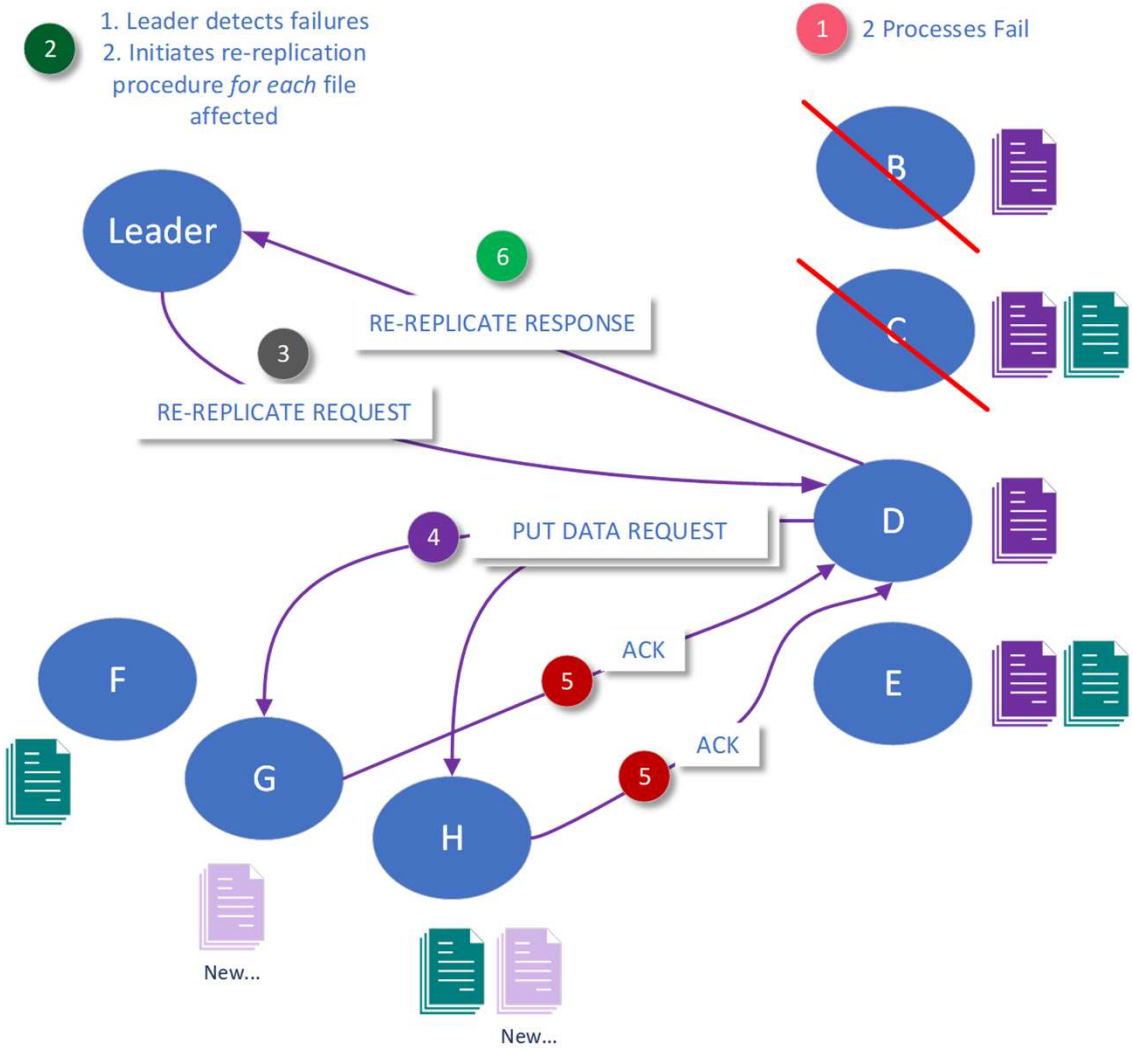
File A

New...

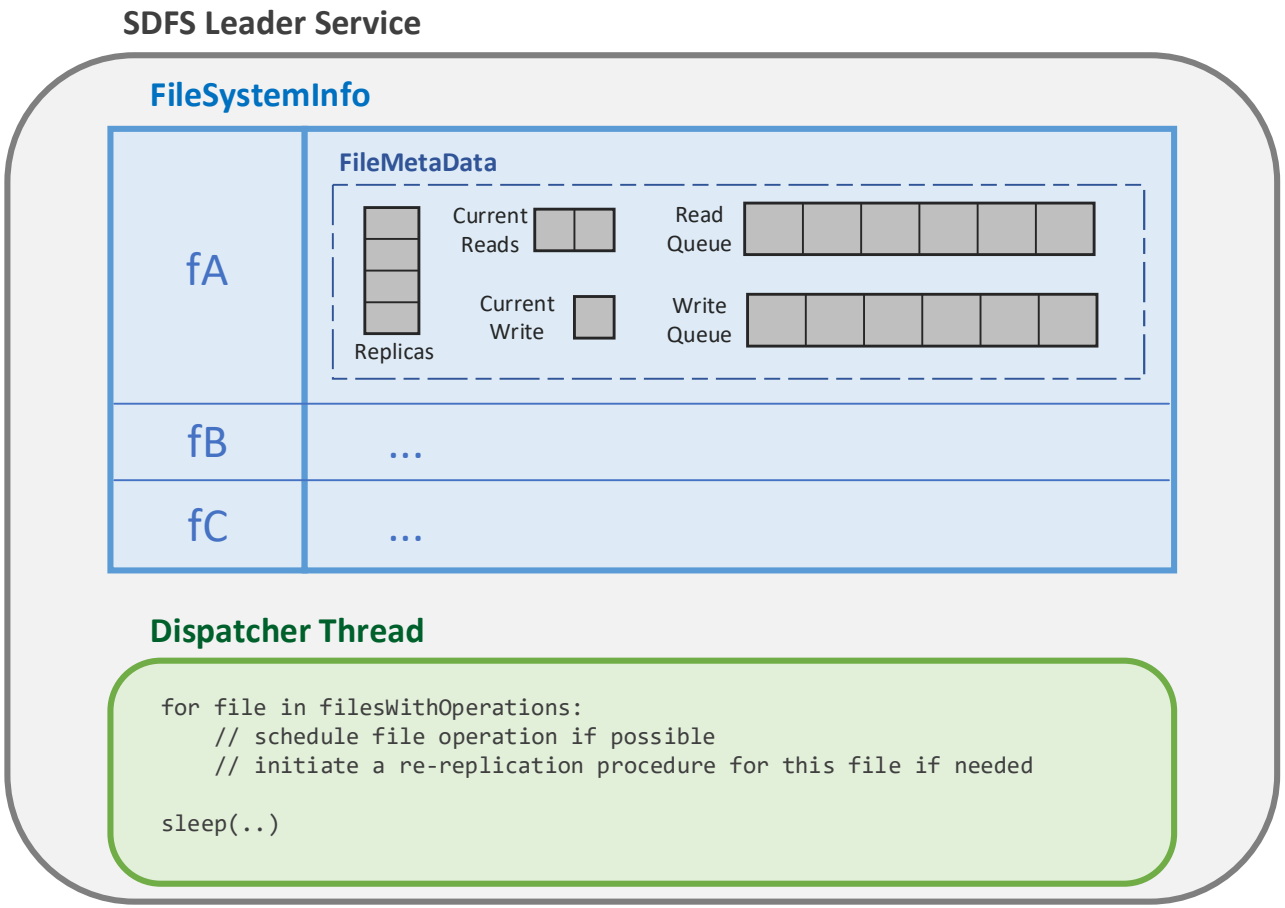
Re-Replicated File A after replication procedure is finished

File B

**Note: only showing re-replication for File A on the right*



# SDFS Leader




- The leader maintains a
  - read & write queue for each file
  - list of replicas of each file
  - extra info to ensure it is starvation free
  - any other metadata for every file in the system
- Dispatcher thread is woken up periodically to
  - schedule any file operations
  - initiate a re-replication procedure in case of a failure

# MapleJuice Framework

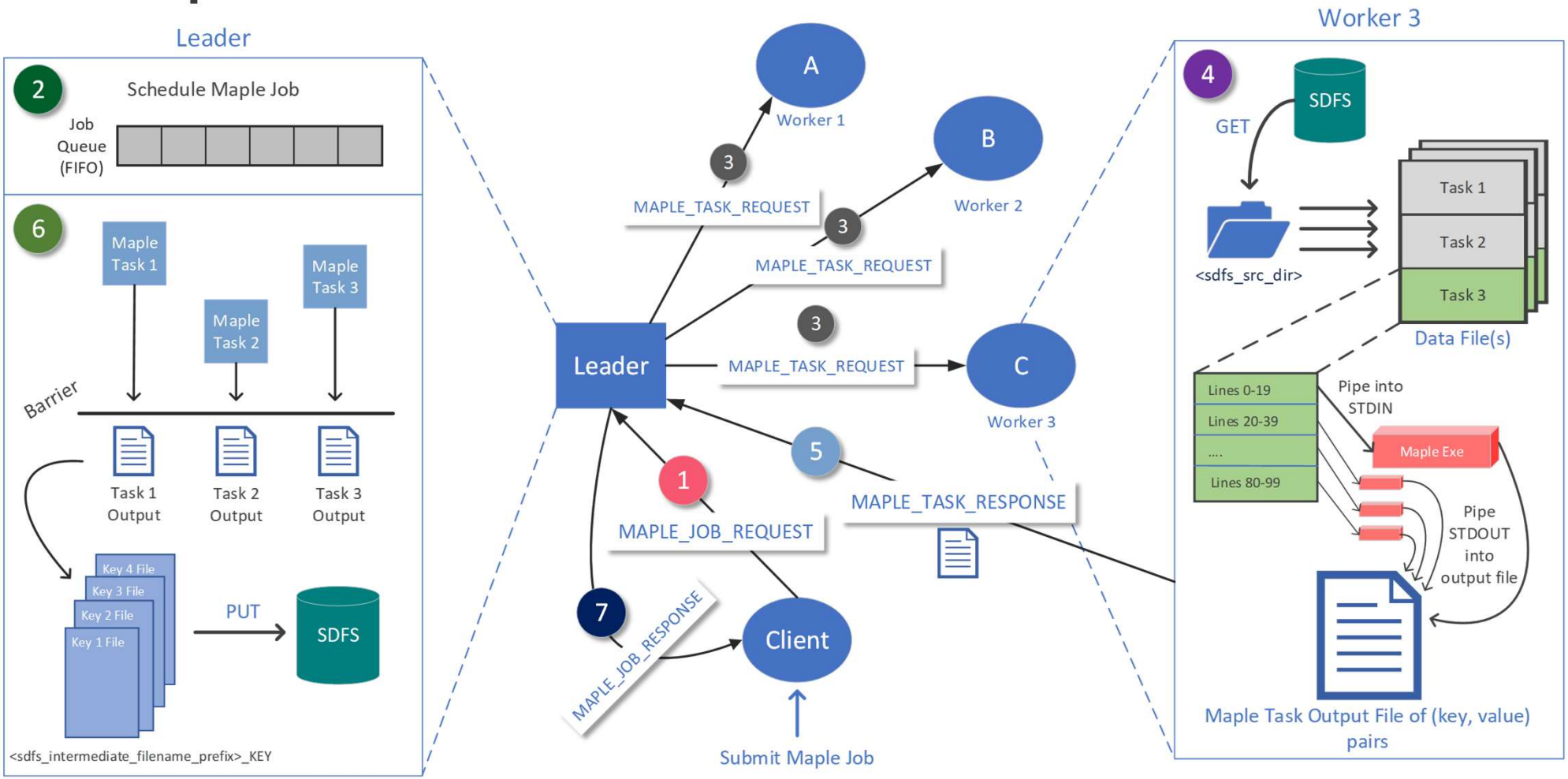
- **Maple (i.e., Map)**: Processes chunks of data in parallel outputting (key, value)
- **Juice (i.e., Reduce)**: Aggregates and combines results from the Map phase.
- **Failures & Re-replication:** Any worker nodes that fail will have it's Maple/Juice Task rescheduled & restarted on a new worker node
  - Assumes no leader failure
- User must write a **maple exe** file and **juice exe** file describing the logic that needs to be accomplished
  - Exe file must read input from STDIN and output (key, value) pairs to STDOUT

**Commands:**

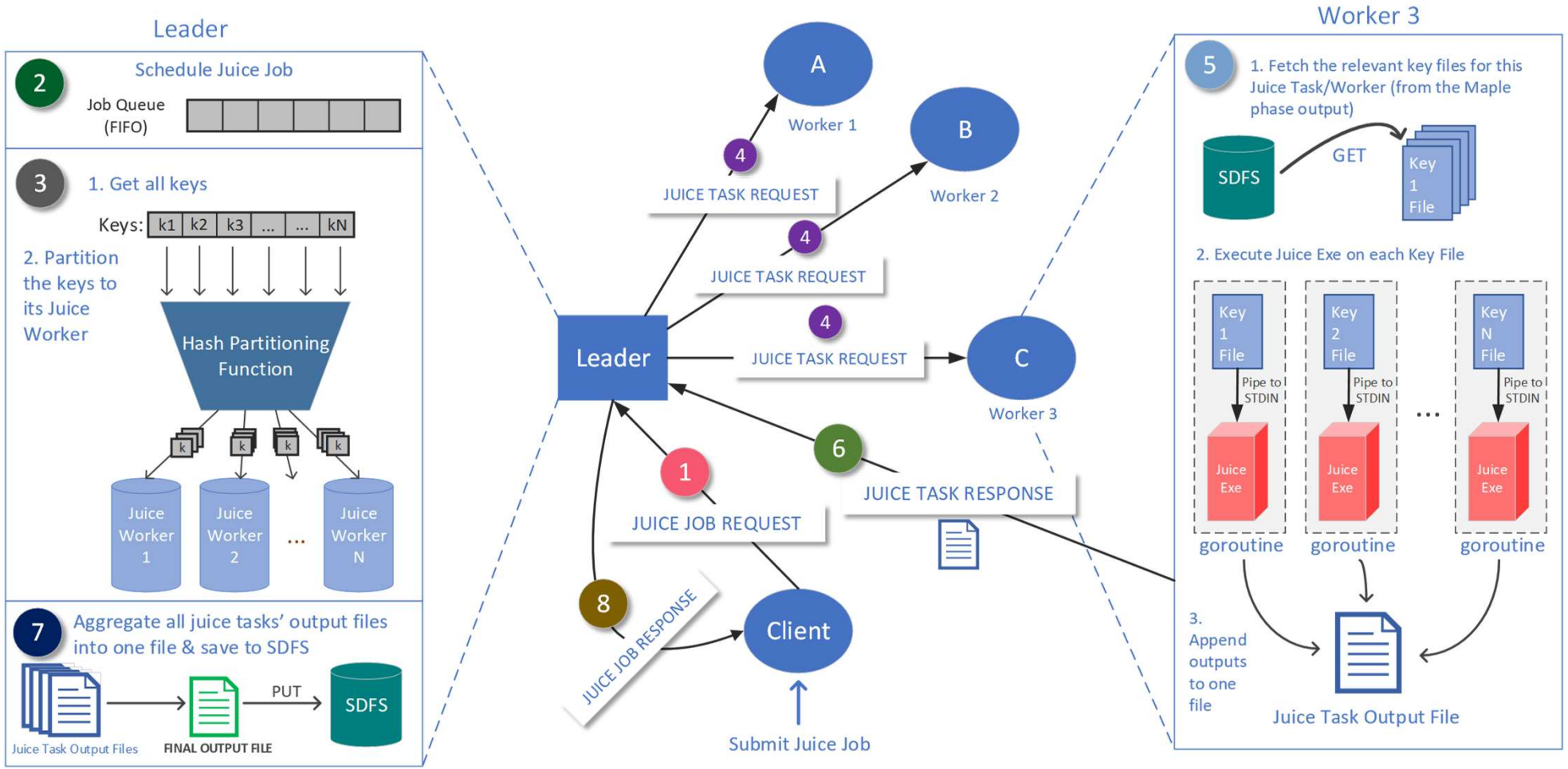
---

```
maple <maple_exe> <num_maples> <sdfs_intermediate_filename_prefix> <sdfs_src_directory>
juice <juice_exe> <num_juices> <sdfs_intermediate_filename_prefix> <sdfs_dest_filename> delete_input={0,1}
```

# Maple Job Execution



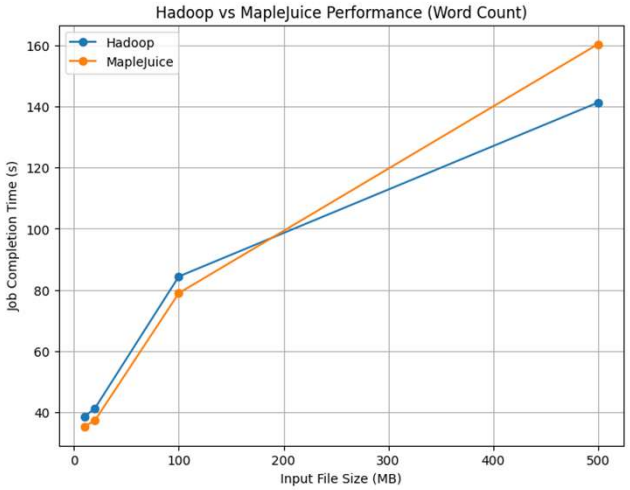
# Juice Job Execution

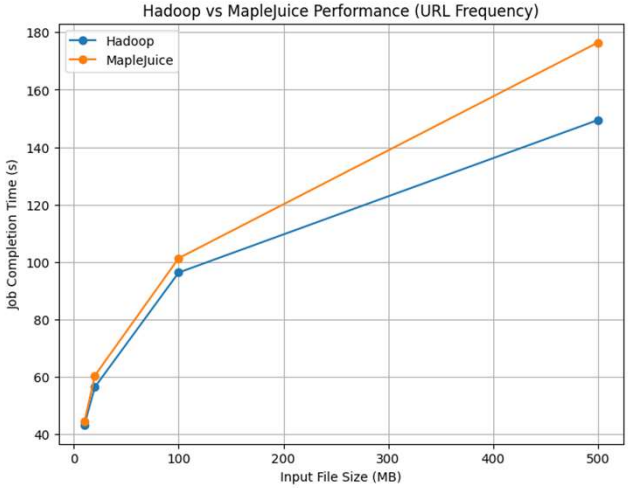


# Performance Results

# MapleJuice vs Hadoop

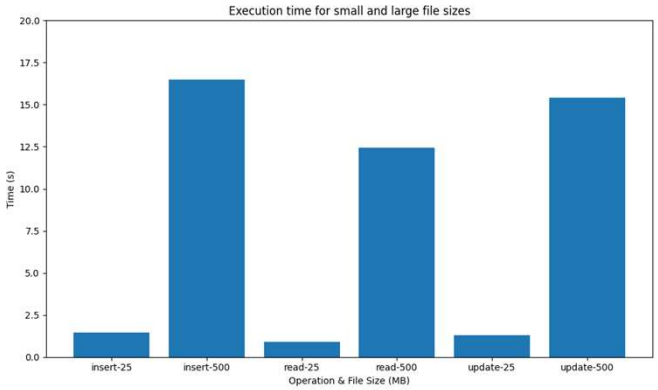
- Word Count is a subset from Gutenberg Books Dataset and URL Access Percentage is from a Web Caching Dataset
- MapleJuice had slightly better results than Hadoop when file sizes were small.
- Configured Hadoop & MapleJuice to have a replication factor of 3
- MapleJuice had 9 workers for all runs
- Hadoop automatically decides the number of Map/Reduce tasks to achieve best results



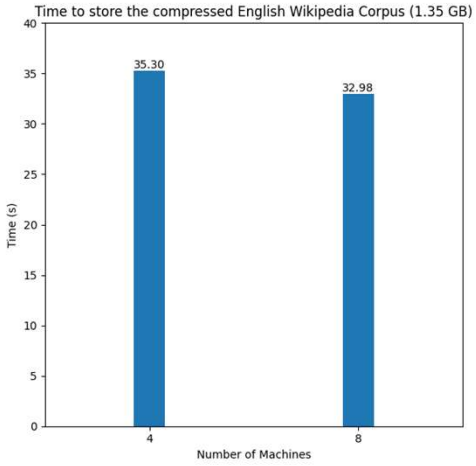



# SDFS Graphs & Performance

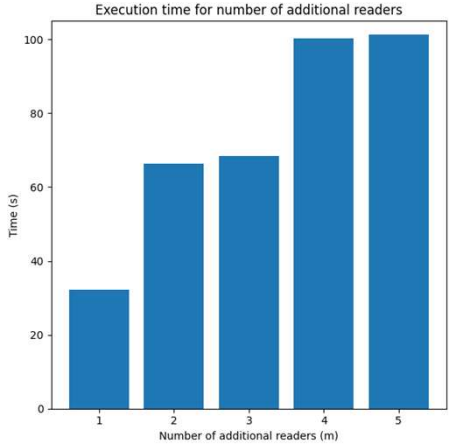
- Measured execution time for PUT, GET, and overwriting a file with a PUT
- 25MB and 500MB file sizes




- Execution time of storing the entire compressed English Wikipedia Corpus (1.35 GB) while varying the number of machines
- Shows how the speed is independent of the number of machines since # replicas = 4



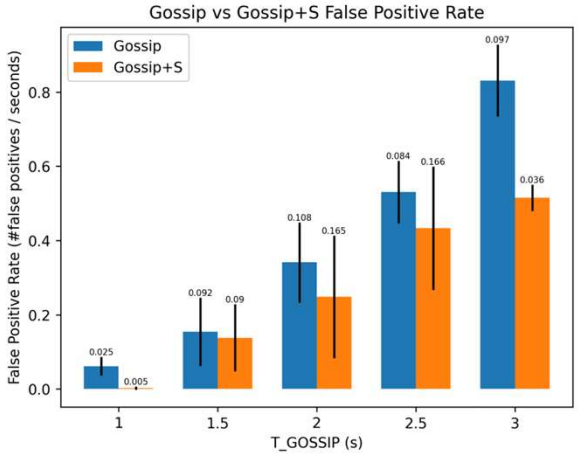

- Measured the execution time for a read with *m* waiting readers for a file that takes ~30 seconds to read w/ 0 waiting readers

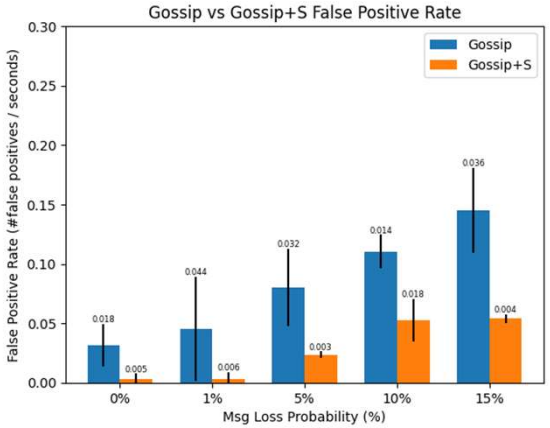


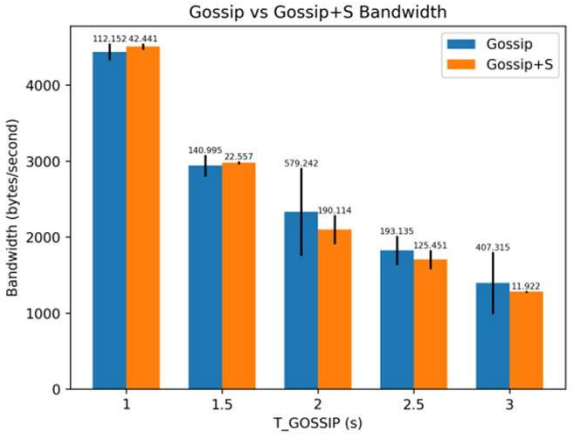
# Improvement: Gossip + Suspicion Mechanism

- Inspired by the SWIM Protocol*
- Reduces false positives
- Mark node as *suspicious* before marking as *failed*
  - Other nodes may correct your *suspicious* claim before you incorrectly mark it as failed

*Further details in Appendix



# Future Improvements

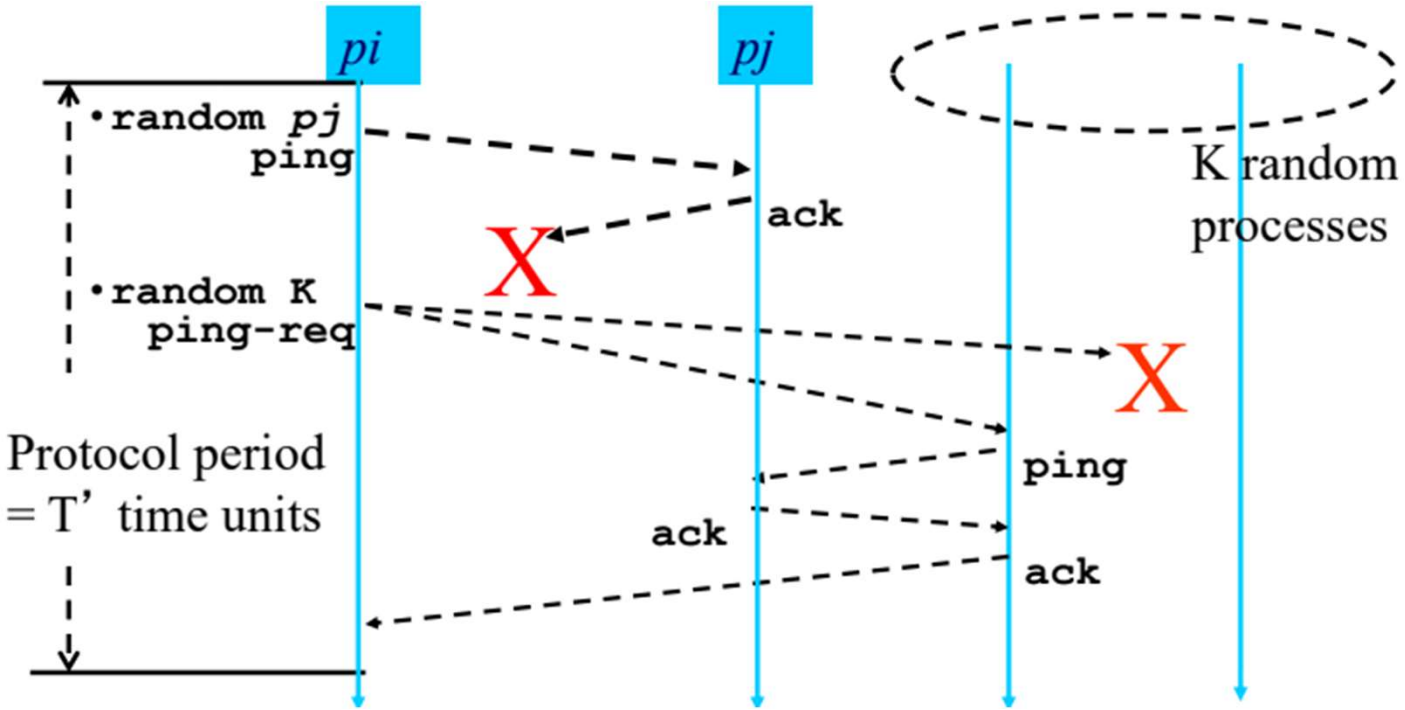
- **General**
  - Handle leader failure & leader election (Implement a consensus protocol like Raft or Paxos)
  - Remote Procedure Calls (RPCs)
- **Failure Detector**
  - SWIM Protocol for failure detection
- **SDFS**
  - Implement sharding
  - Implement pipelined writes (similar to how HDFS performs writes)
  - Quorum consistency
  - Improving reads by picking the fastest read instead
- **MapleJuice Phases**
  - Speculative Execution
  - Different partitioning schemes in Juice Phase
    - Range Partitioning
    - LEEN (Locality-aware and fairness-aware key partitioning
  - More intelligent assignment of maple/juice tasks
  - Store Maple's intermediate output to the local disk and perform a remote read instead of saving it to SDFS

# Lessons Learned & Challenges

- Scalability challenges
- Fault tolerance
- Design and performance optimizations used in existing distributed systems like Hadoop, Cassandra, etc.
- Testing and debugging distributed systems & networking applications
- Working effectively in a team environment
- Remote Procedure Calls (RPC)
- Acquired new skills
  - Go Programming Language, networking, Docker

# Appendix

# SWIM (Scalable Weakly Consistent Infection-Style Process Group Membership)




- Constant load per member

Source: Indranil Gupta, CS 425 at University of Illinois Urbana-Champaign

# Distributed Log Querier

- **Challenge**: debugging distributed systems
  - Debugging using log files, which are very large and on separate machines
- **Solution:** developed a distributed log querier that runs grep queries remotely on each machine and sends back the output concatenated in a readable manner

# SQL Filter

- Works on CSV Datasets

```
SELECT ALL FROM Dataset WHERE <regex condition>

Maple(record):
  if record matches regex expression:
     output (key=NULL, value=record)

Juice(key, values):
     for record in key:
          output(record, NULL)
```