Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Parallel Programming Concepts

MapReduce

Frank Feinbube

Source:
MapReduce: Simplied Data Processing on Large Clusters; Dean et. Al.

Examples for Parallel Programming Support

	Task-Parallel Programming Model	Data-Parallel Programming Model	Actor Programming Model	Functional Programming Model	PGAS / DSM Programming Model
Shared Memory System	OpenMP, Threading Libs, Linda, Ada, Cilk	OpenMP, PLINQ, HPF	Scala, Erlang	Lisp, Clojure, Haskell, Scala, Erlang	-
Distributed Memory System	*Socket communication, MPI, PVM, JXTA, MapReduce, CSP channels*				-
Hybrid System	-	OpenCL	-	-	Unified Parallel C, Titanium, Fortress, X10,Chapel

MapReduce

- Programming model + associated implementation
- Processing and generating large data sets

- **Map:**
  - key/value pair → intermediate key/value pairs

- **Reduce:**
  - merge all intermediate values associated with the same intermediate key

- Origin: Lisp

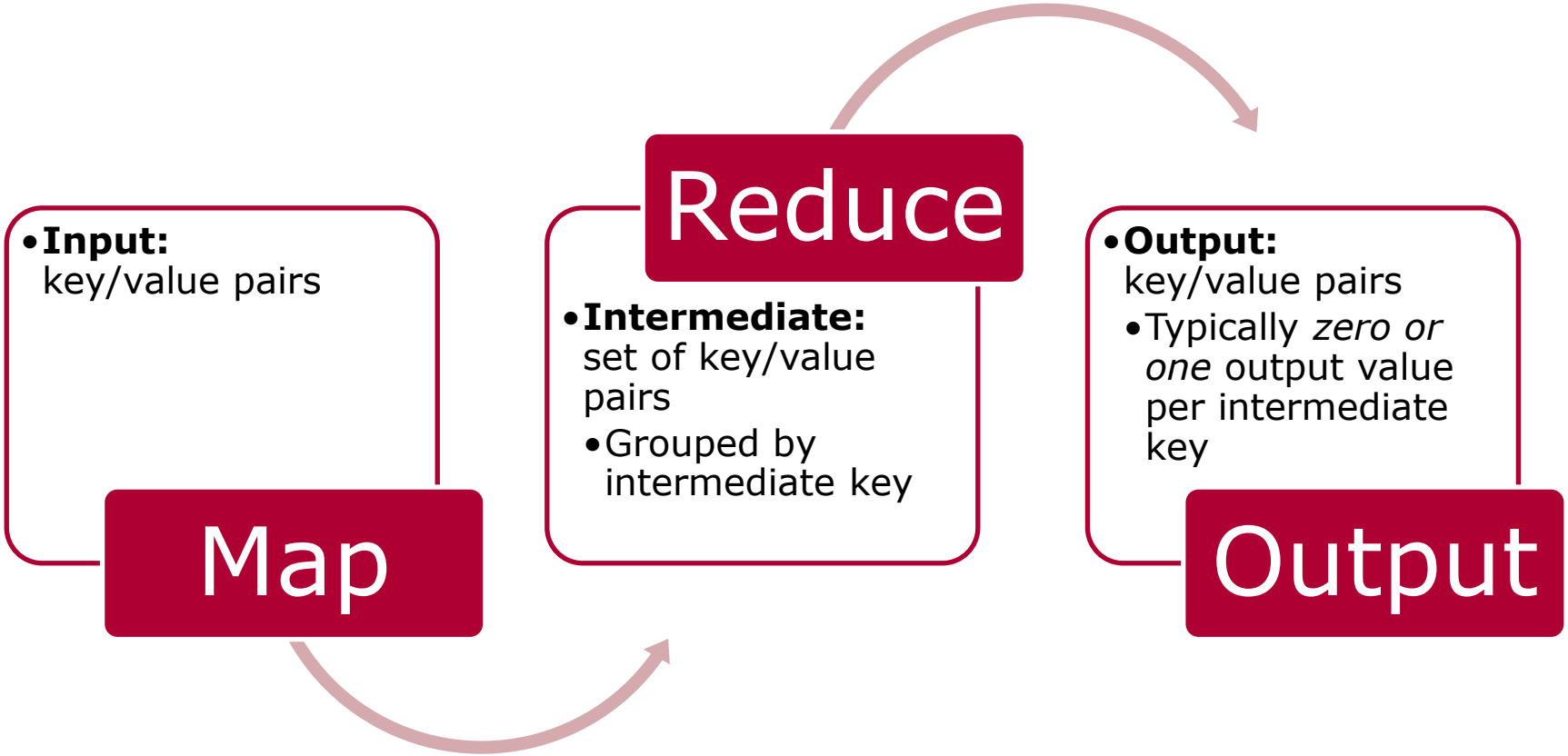
Run-time system

Automated parallelization and distribution

- Partitioning the input data
- Scheduling the program‘s execution across a set of machines
- Handling machine failures
- Managing the required inter-machine communication

➢ Programmers do not have to think about parallel and distributed system specifics

Programming Model



Example: Wordcount

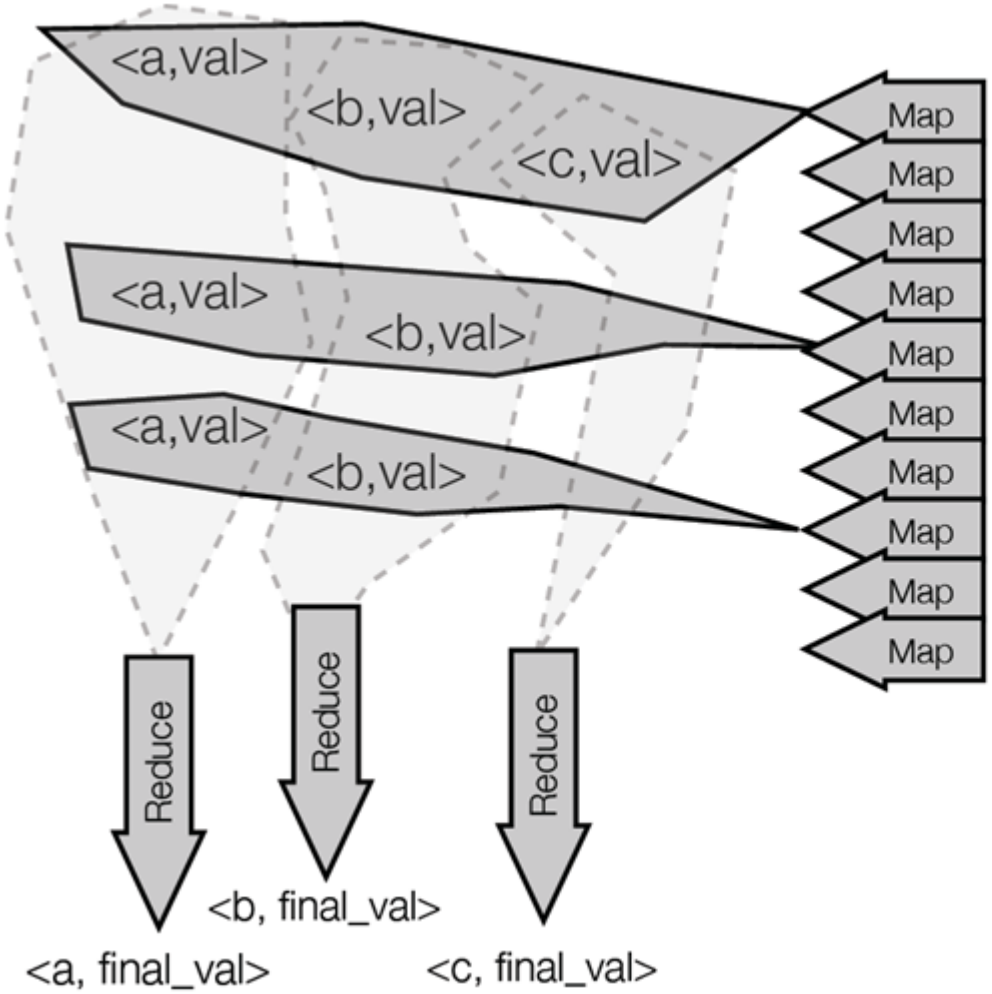
Counting the number of occurences of each word in a large collection of documents:

```
map(String key, String value):
  for each word w in value:
    EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

+ mapreduce specification object

Example: Wordcount



NAME	DATA
doc1	abcbcab
doc2	acbcabac
doc3	aaa
doc4	bcabab
doc5	aababaa
doc6	cabcba
doc7	cbabcabc
doc8	aabbabab
doc9	bcabcbac
doc10	bcbac

Types

Type Specification

```
map    (k1,v1)         → list(k2,v2)
reduce (k2,list(v2)) → list(v2)
```

Example

Work items: ID, binary content → character, number of occurences

```
map    (long,byte[])          → list(char,int)
reduce (char,list(int))       → list(int)
```

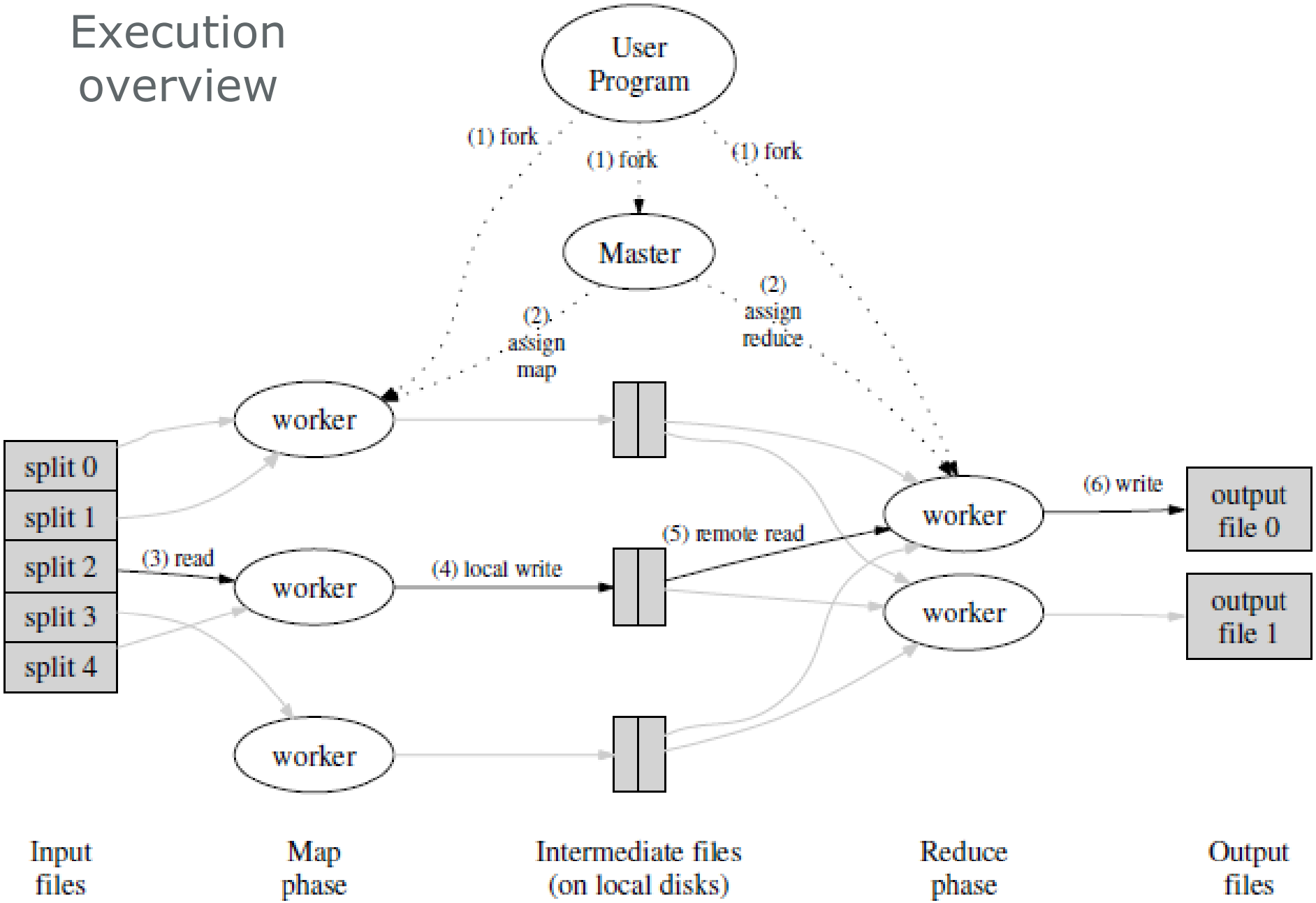
More Examples

Example	Map	Reduce
Distributed Grep	Emits a line, if it matches the pattern	Emit unchanged
Count of URL access frequency	Processes logs of requests: `<URL,1>`	Add values per URL: `<URL, total count>`
Reverse web-link graph	`<target,source>`, if link is found in source	`<target,list(source)>`
Term-vector per host (list of most important words)	`<hostname,term vector>` for each input document	Add all term vectors together: `<hostname, term vector>`
Inverted index	Parse document, emit `<word, document ID>`	Sort and emit `<word,list(document ID)>`
Distributed sort	Extract keys from records: `<key,record>`	Emit unchanged (done by ordering properties)

Google Implementation of MapReduce

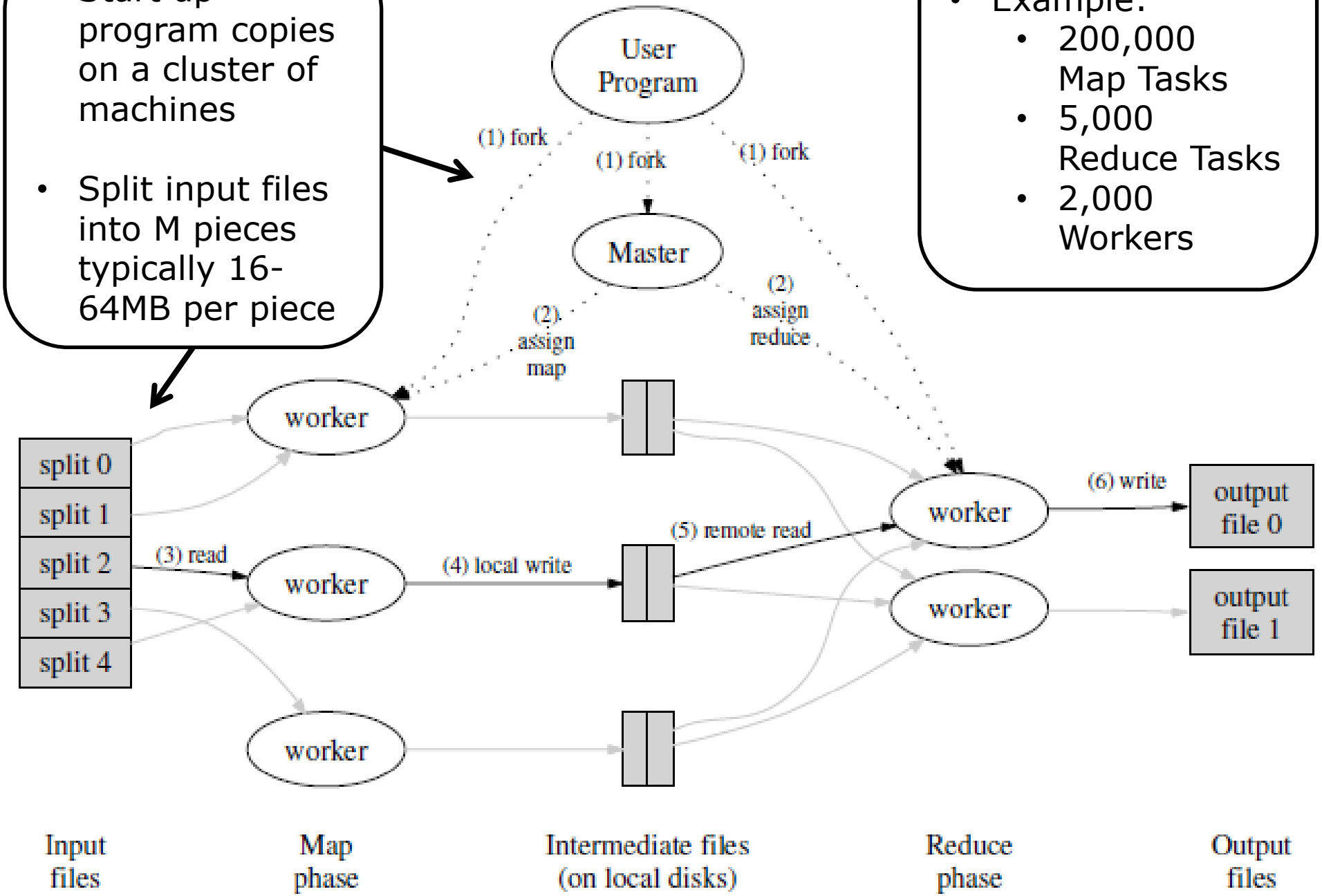
- Large cluster of standard PCs with local disks
  - x86, Ethernet: 100 Mbit/s to 1 Gbit/s, 2-4GB RAM, IDE
  - Custom global file system
    - Replication for availability and reliability
  - Job scheduling system
    - Set of tasks to set of machines
  - Machine failures are common (large number of machines)

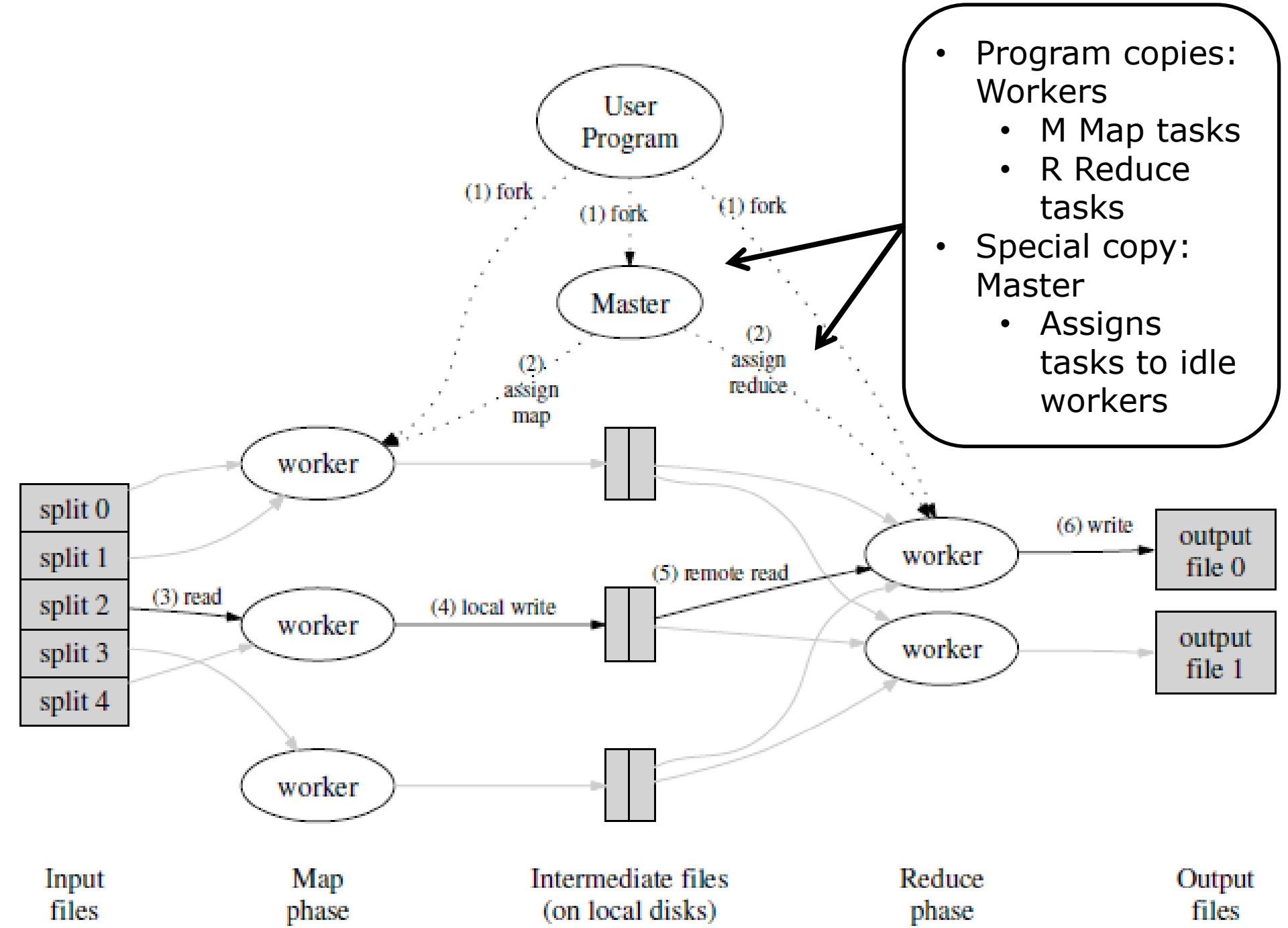
Execution overview



- Start up program copies on a cluster of machines
- Split input files into M pieces typically 16-64MB per piece

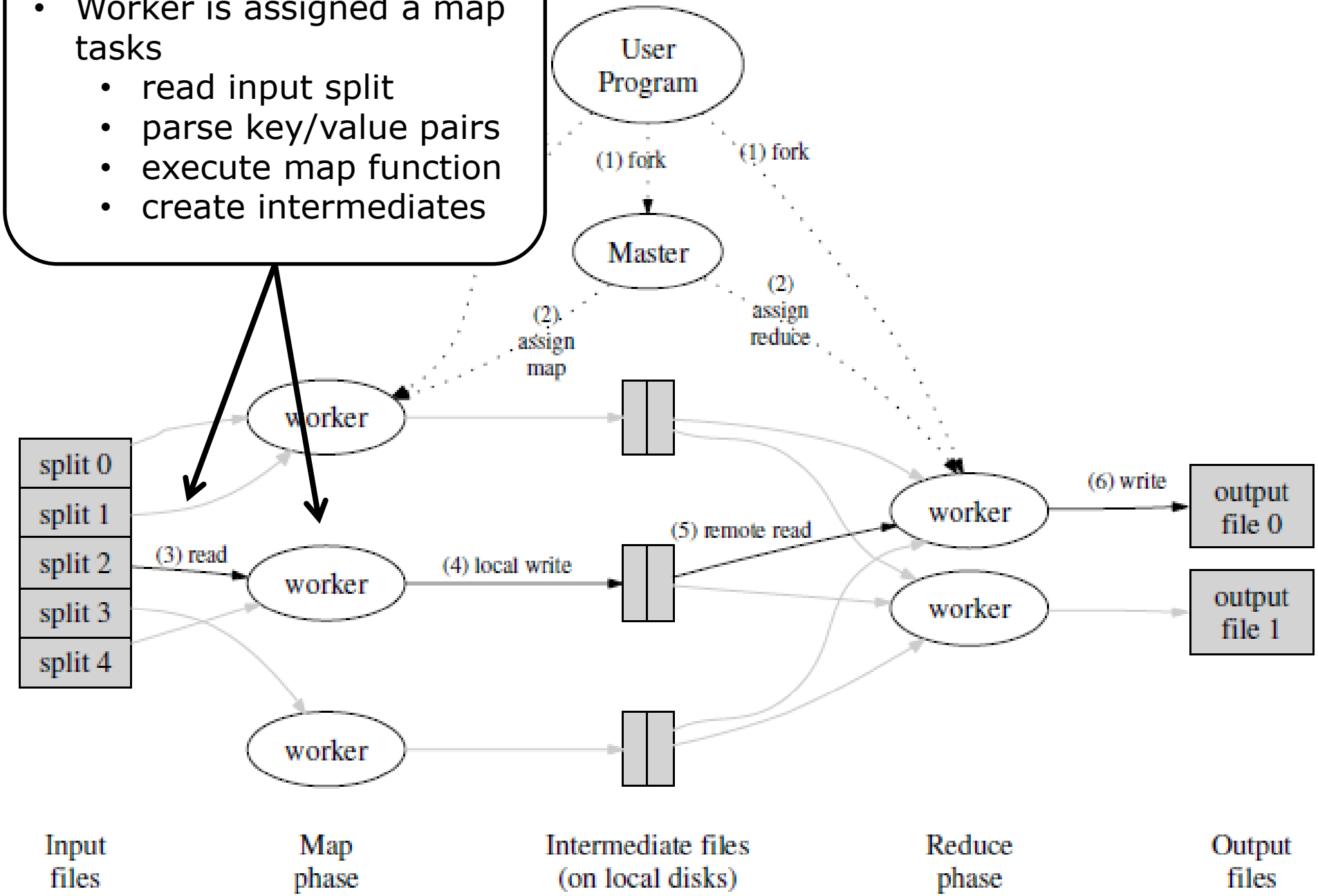
- Example:
  - 200,000 Map Tasks
  - 5,000 Reduce Tasks
  - 2,000 Workers



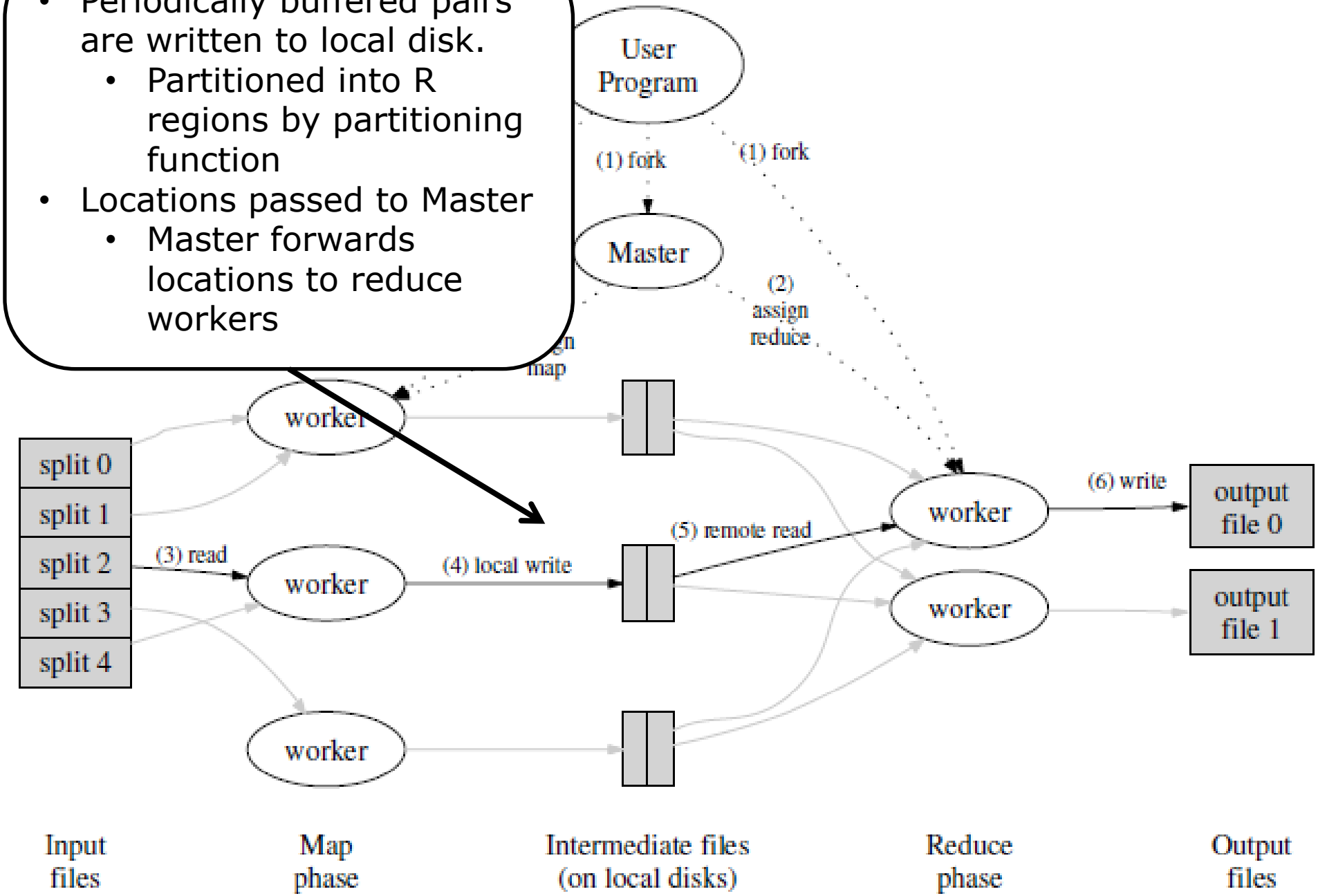


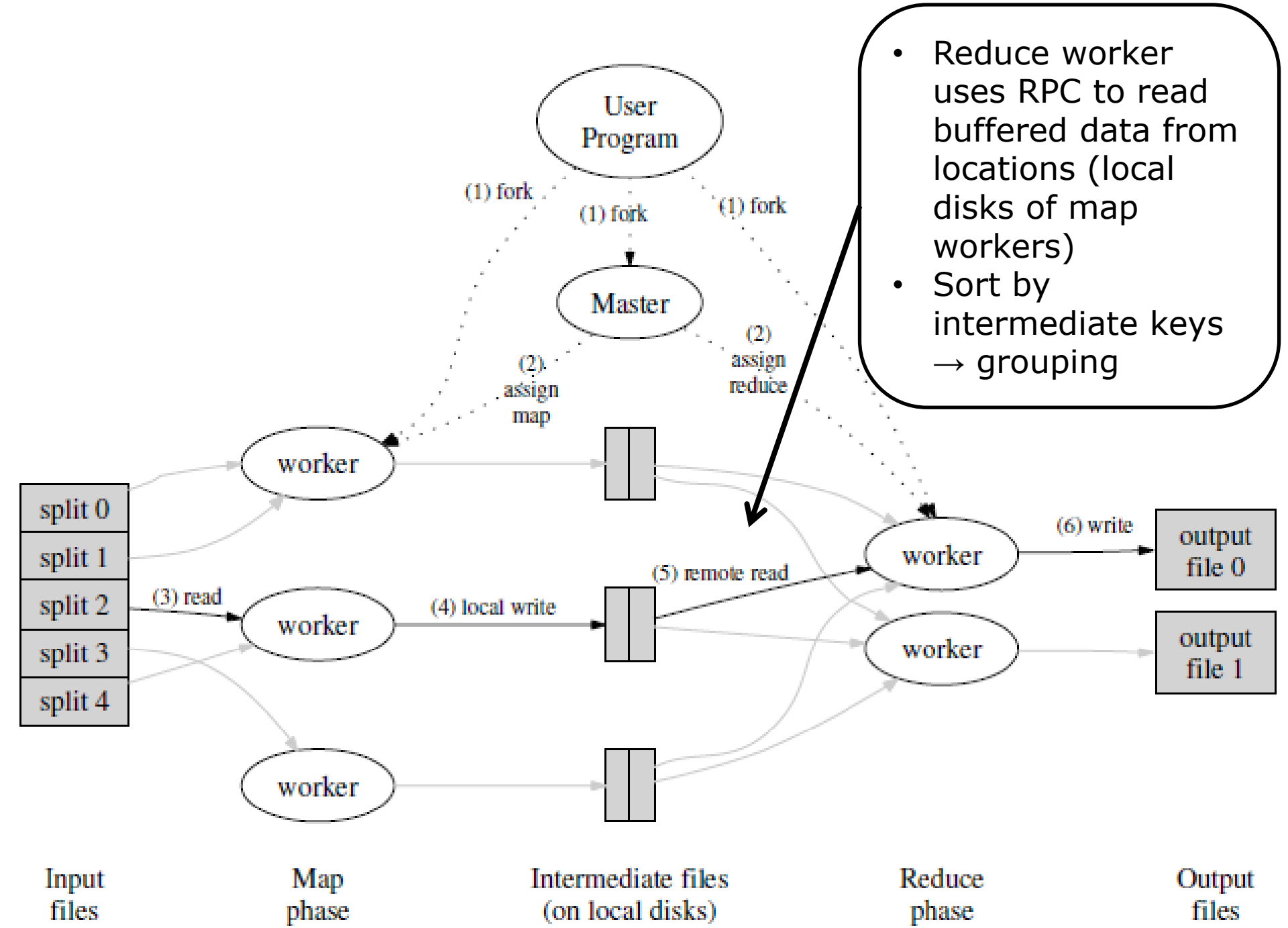
- Program copies: Workers
  - M Map tasks
  - R Reduce tasks
- Special copy: Master
  - Assigns tasks to idle workers

- Worker is assigned a map tasks
  - read input split
  - parse key/value pairs
  - execute map function
  - create intermediates



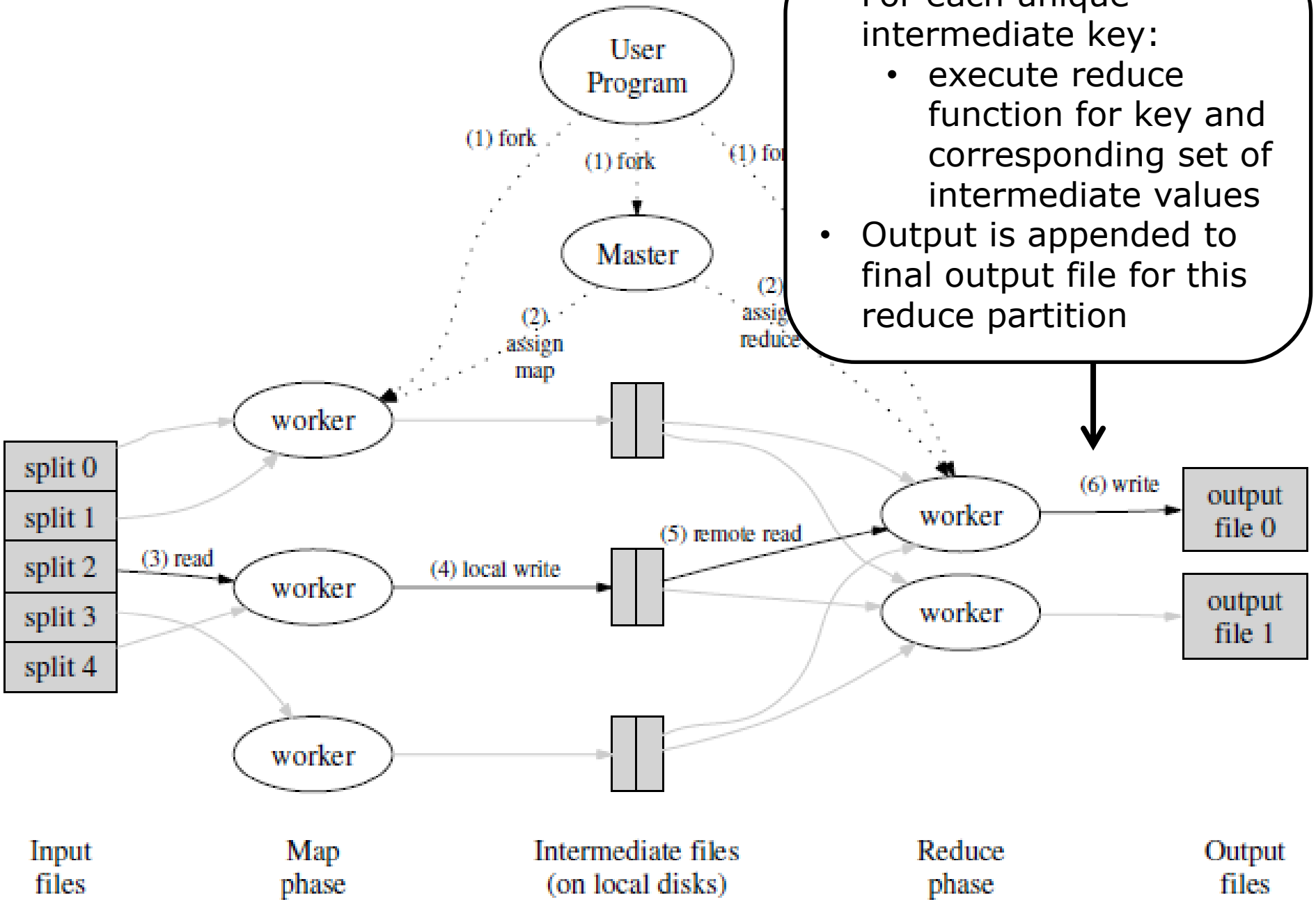
- Periodically buffered pairs are written to local disk.
  - Partitioned into R regions by partitioning function
- Locations passed to Master
  - Master forwards locations to reduce workers

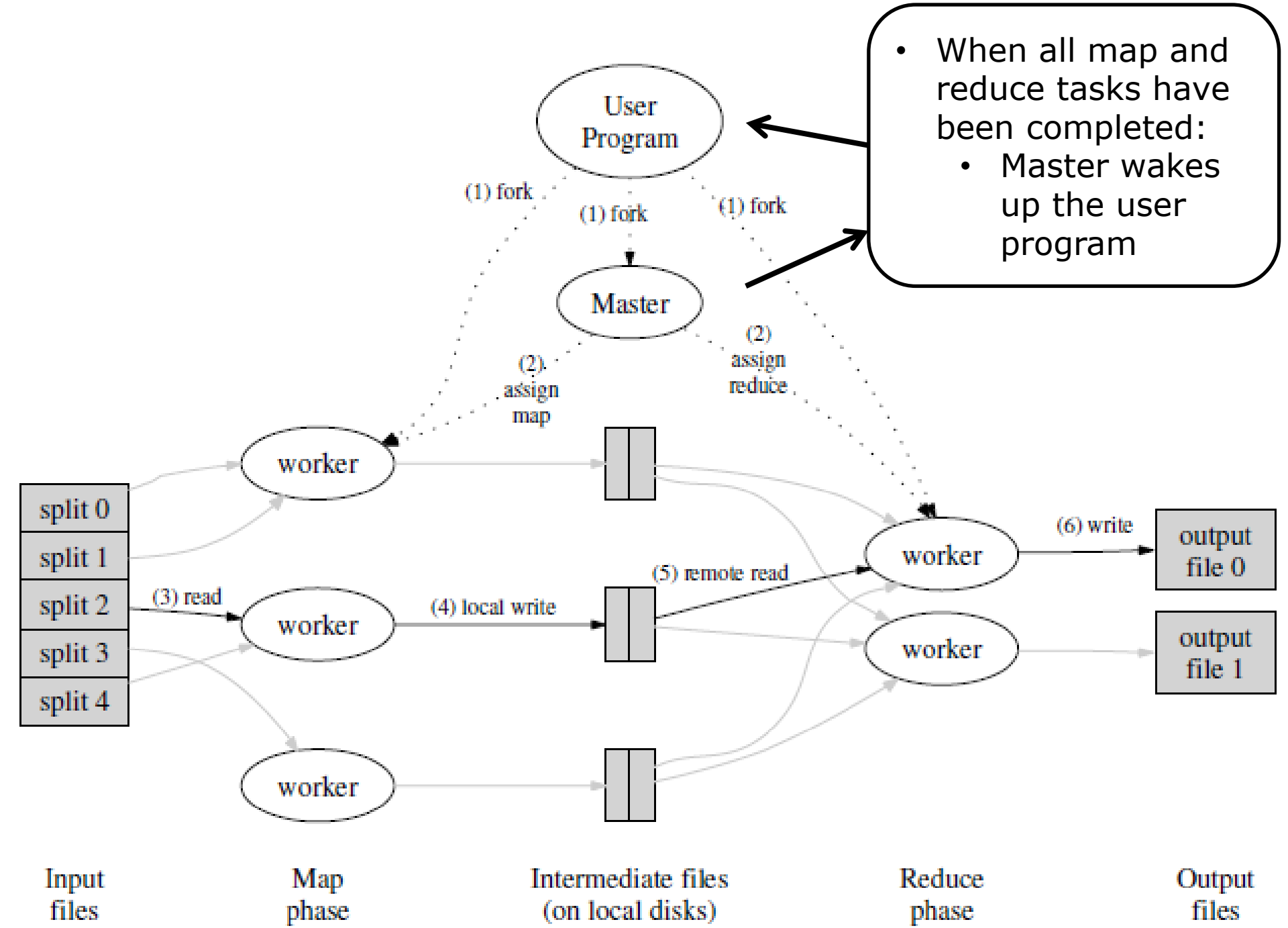




- Reduce worker uses RPC to read buffered data from locations (local disks of map workers)
- Sort by intermediate keys → grouping

- For each unique intermediate key:
  - execute reduce function for key and corresponding set of intermediate values
- Output is appended to final output file for this reduce partition





- When all map and reduce tasks have been completed:
  - Master wakes up the user program

Specific Google properties

- Network bottleneck in Google cluster
  - Master tries to use locality information about the input data, which is stored in the distributed file system
  - For large MapReduce tasks, most input data is read locally
- **Fault tolerance**
  - Periodic heartbeat between master and workers
  - For a failed worker, re-execute completed and in-progress map tasks (of this particular worker)
  - For a failed master, MapReduce is aborted → user has to re-execute
  - Span backup tasks (cloned workers, same task) when MapReduce is close to completion, to compensate faulty (delaying) workers

Refinements

Refinement	Description
Partitioning Function	User functions for data partitioning are possible (`hash(key) mod R` is default)
Ordering Guarantees	Intermediate key/value pairs are ordered inc.
Combiner Function	Partial merging of local data (like reduce)
Input and Output Types	Some standard formats; user can specify more
Side-Effects	Additional files have to be addressed by the user
Skipping Bad Records	Ignore records with deterministic crashes (configurable)
Local Execution	Special MapReduce library for sequential execution
Status Information	Master runs an internal HTTP server for diagnosis
Counters	Count occurences of various events; user defined