Lecture 14.2
Map/Reduce Semantics

EN 600.320/420

Instructor: Randal Burns

14 March 2018

Department of Computer Science, *Johns Hopkins University*

Types and Domains

- Map is a transformation
  - Input domain to output domain
- Reduce is a collection
  - No domain change
- C++ implementation is based all on strings
  - User code must convert to structured types

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

The Why of Map/Reduce

- *How does programming in Map/Reduce help parallelism?*
  - Consider the count all words pseudo-code example

The Why of Map/Reduce

- *How does programming in Map/Reduce help parallelism?*
  - Consider the count all words pseudo-code example
- Potential parallelism
  - Map: on all the sites that store data
    - Or on all the sites to which you distribute data after accessing it from storage
  - Reduce: as many computers as there are terms

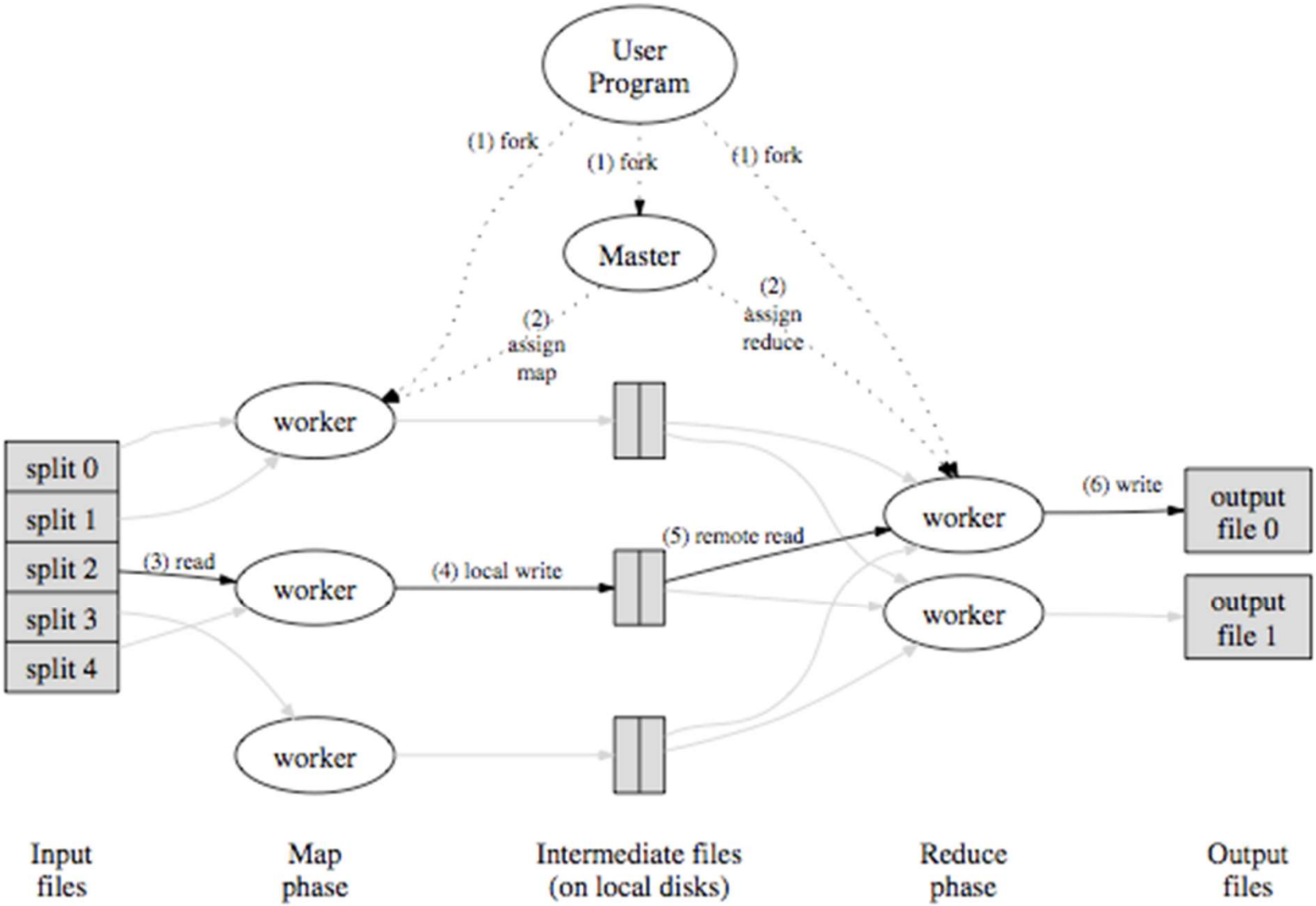
The Point

- Programs in Map/Reduce are automatically parallelized by the Map/Reduce runtime system

- No programmer interaction with parallelism
  - Except encoding problem well in Map/Reduce
- No explict knowledge of
  - # of machines
  - Location of machines
- Scale up parallelism for arbitrary configurations

- (For embarassingly parallel problems?)

Map/Reduce Runtime



Runtime Properties

- Automatically partition input data
  - 16-64 MB chunks for Google
- Create *M* map tasks: one for each chunk
  - Assign available workers (up to *M*) to tasks
- Write intermediate pairs to local (to worker) disk
- *R* reduce tasks (defined by user) read and process intermediate results
- Output is *R* files available on shared file system
- Master tracks state
  - Asssignment of *M* map tasks and *R* reduce tasks to workers
  - State and liveliness of the above