Project Design Document

(DPsort : Distributed, Parallel External MergeSort)

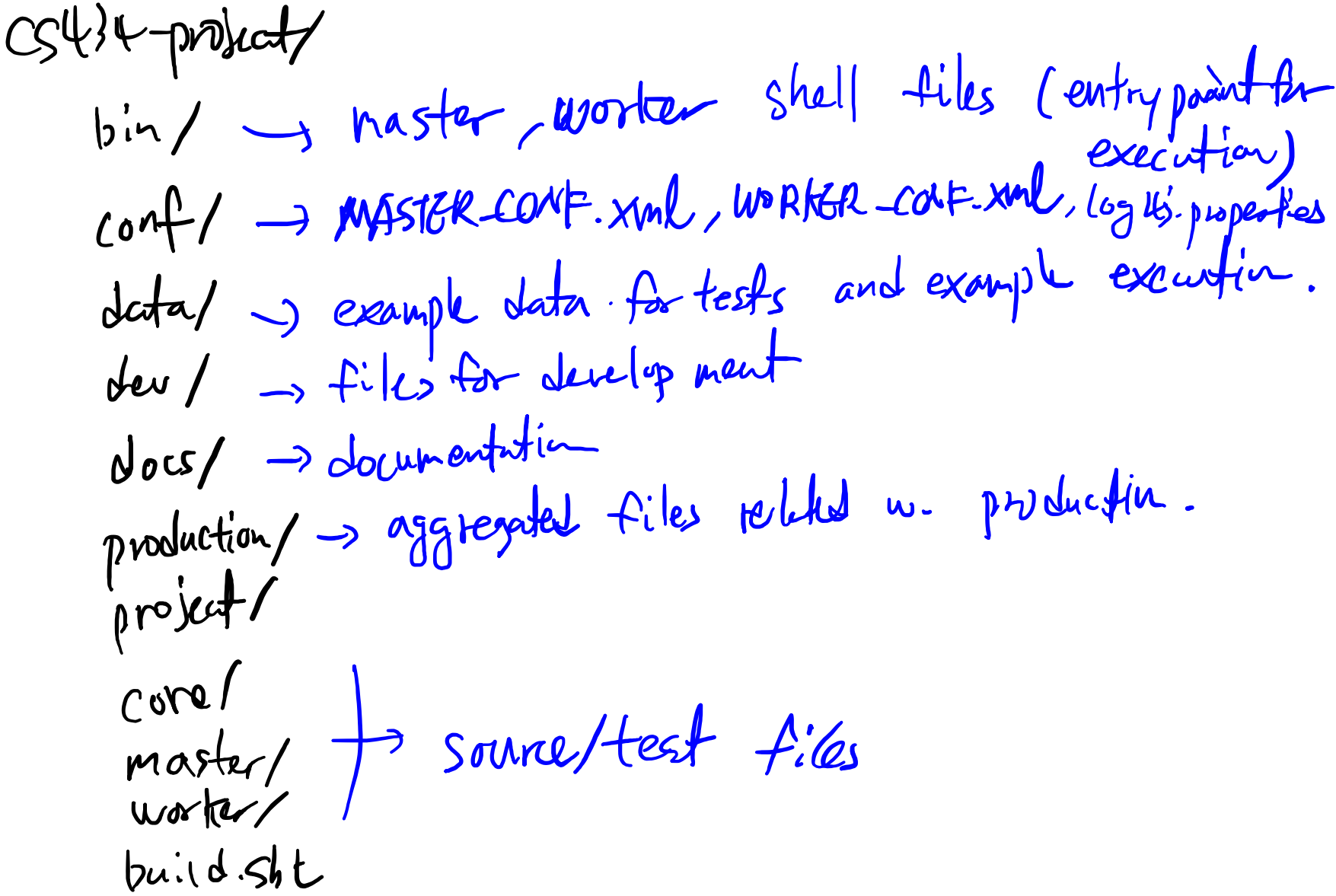
- Milestone 1 -

20180274 Jinho Ko

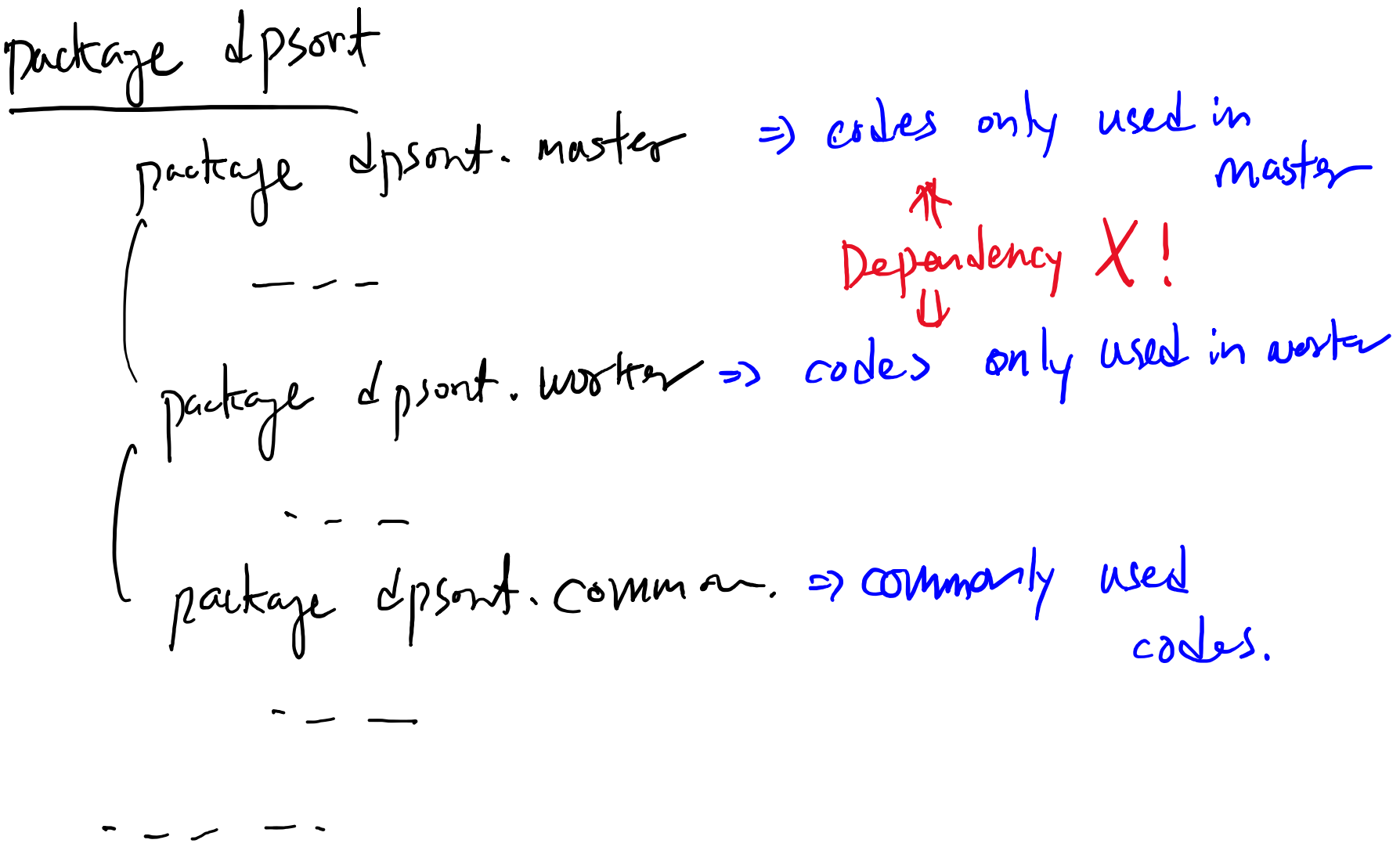
Progress Report Summary

- Finished deciding project directory structure.
- No progress in coding
  - Still need to study more on network programming
- Designed the architecture of system, especially one when parallelization becomes compulsory.

Project Directory Structure



Package Structure



Execution Behavior

(.sh)

@Master `bin/master` [#numWorkersToJoin, args...]

- parse/validate arguments + attach conf files (conf/...)
- find compiled source codes
- execute JVM.

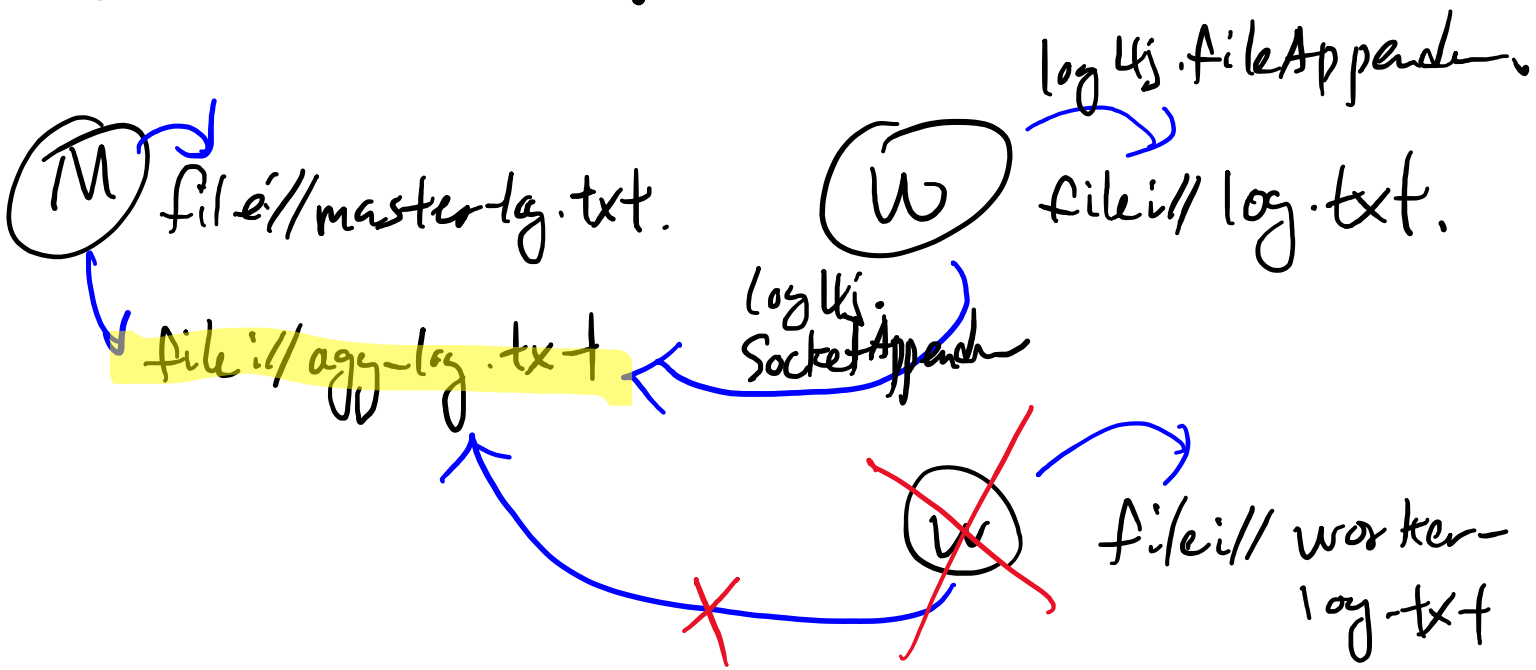
@Worker `bin/worker(.sh)` [MasterIp, port, -I ~, -O ~.]

- if not exists → err
- if not exists → mkdir.

- //
- //
- execute JVM

Log Aggregation

Since logs are gathered from multiple machines, we aggregate them to the master machine.



⇒ When network causes error, it can still save log to its local file.

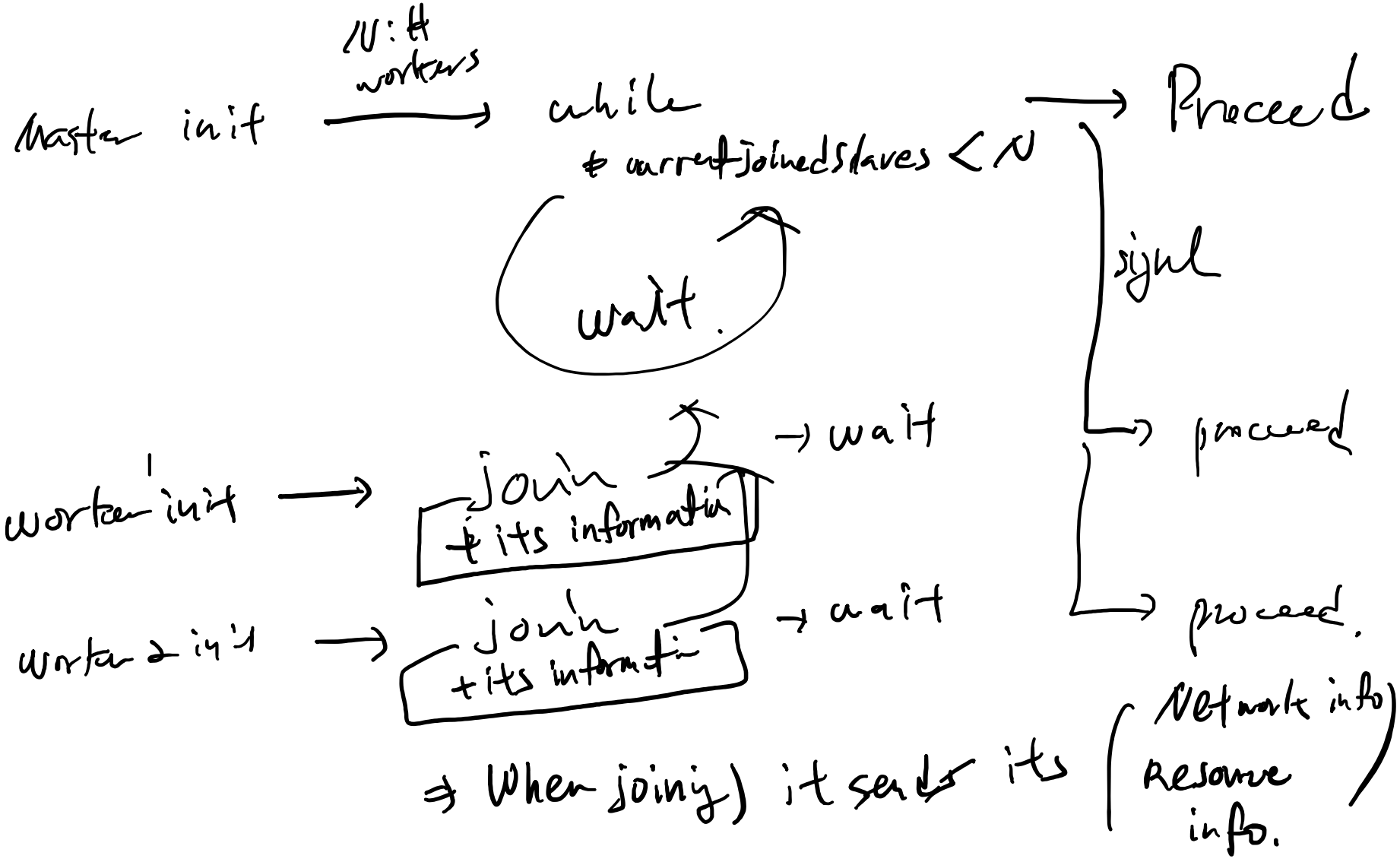
Conf files

common: file directory / memory buffer
includes. logging / Data format
Timeout

- MASTER_CONF.xml
  - → common
  - → master-only (Parallism behavior, Port range, ...)
    - ↳ # partitions
- WORKER_CONF=.xml. ⇒ can vary depending on each worker.
  - → common
  - → worker-only (Hardware specs (RAM, CORES, ...)
    - ...

log4j.properties.

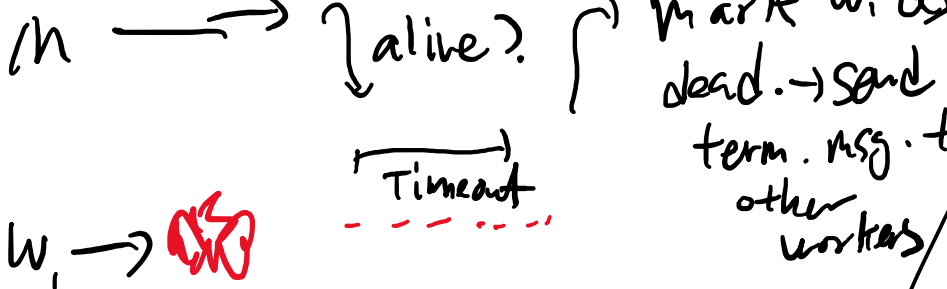
Worker Joining Behavior



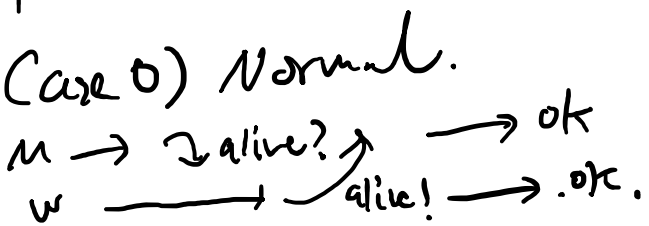
Heartbeat

- Idea: master never gets to know if worker is running right unless worker sends its heartbeat.
- Why heartbeat? : memory overflow / system lag due to excessive operation.
  - ↳ almost all cause of error.

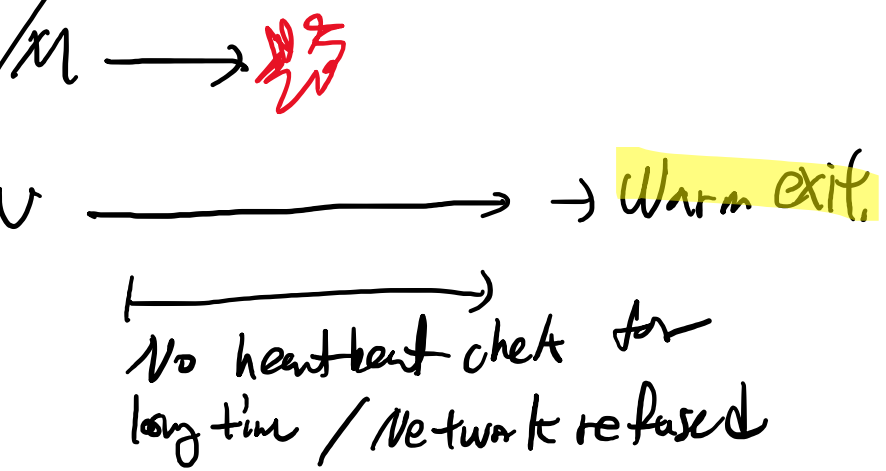
Case 1) worker is missing.



Case 0) Normal.

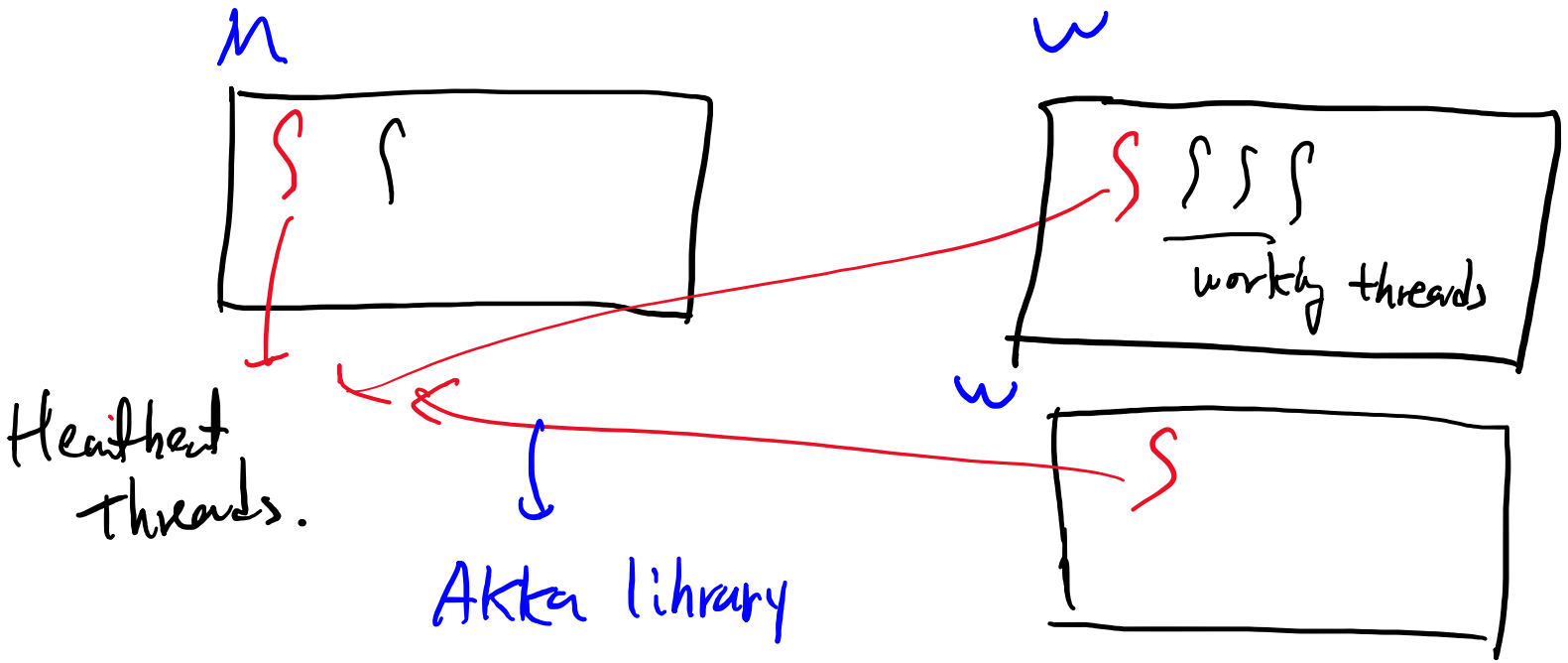


Case 2) master is missing.



Heartbeat (con't)

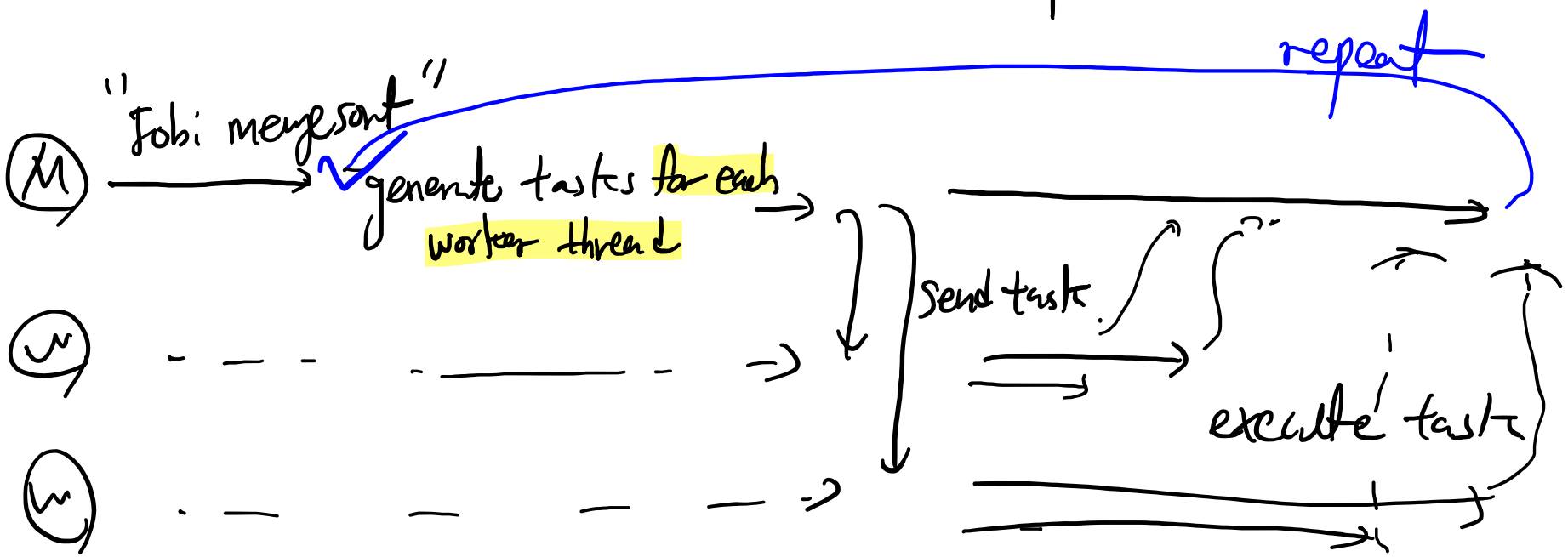
- Heartbeat "should" run on seperate thread.
- The thread also should have enough opportunity to be executed, otherwise, possibility for timeout.



Execution-related Terminologies

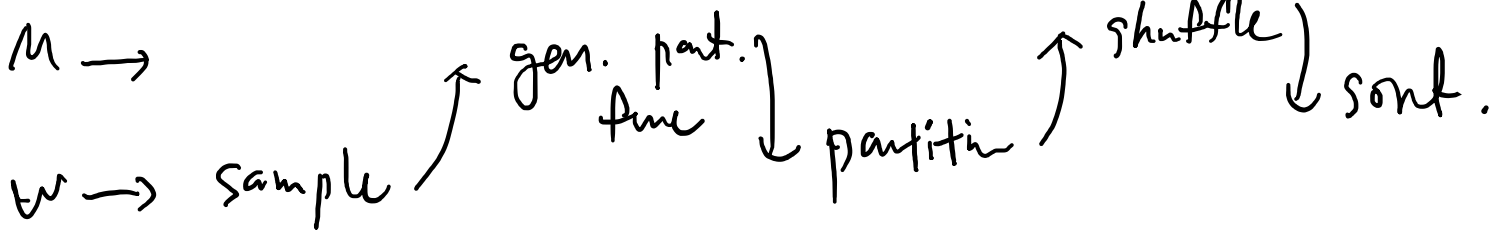
concept referenced from Apache Spark

- Job: "external merge sort"
- Task: minimum unit of execution allocated to each worker thread (& partition)



Passing Tasks

Idea 1) Concurrency Control.



(+) simplest to implement

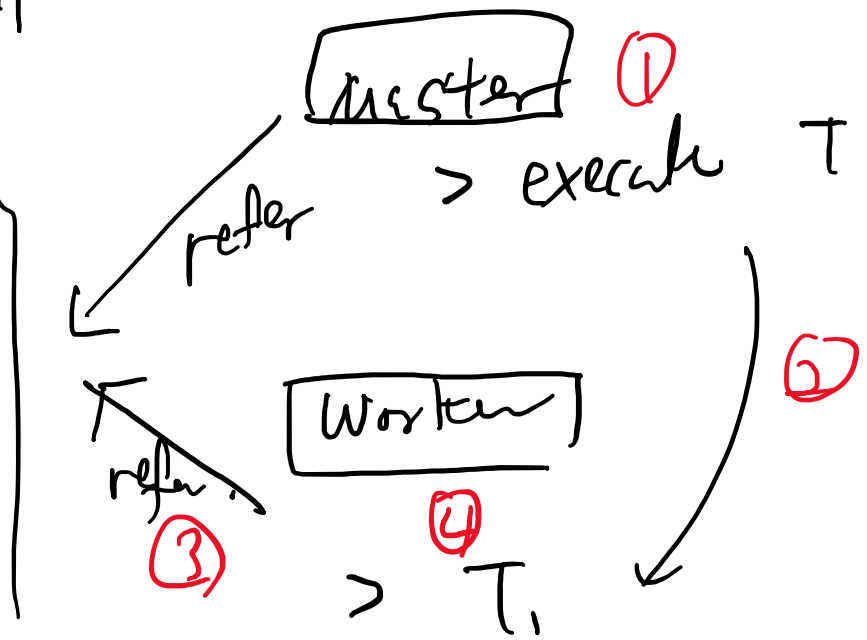
(-) hard to revise/upgrade code.
(change in execution flow.

Passing Tasks (con't)

Idea 2) Use task type ID.

a dpsort.common.

```
{ T1 : executeSort()
  T2 : executeMerge()
  :
  :
}
```



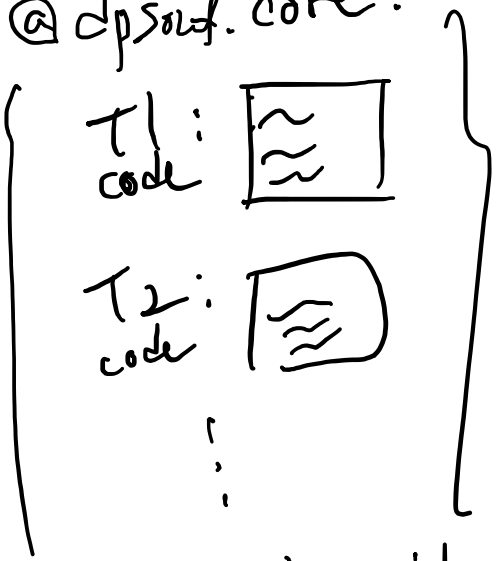
(+) much easier to upgrade code → by modifying Task types

(−) code length ↑

Passing Tasks (con't)

Idea 3) Worker doesn't know about the code

@dpsuf.core.



Master.
- > val serCode = serialize(code)
- > execute serCode

Worker
- > fetch code (+ data)
- > execute code.

- ✓ short worker-side codes
- (+) ✓ robust.
  - ✓ can be easily optimized / upgraded.
- (-) might be tricky to implement.

Passing Tasks : Decision

✓ Considering between Idea 2 & Idea 3.

✓ ③은 하지 않으면 보이도 수행함.
code serialization이 필요.

하지만, 이것도 partition function은 broadcast하려면 serialization이 필요한 수도 있음.

Serialization

Objects should be serialized in order to be passed via networking.

- Data: shuffle / sample data
- Code: generated partition function.

Resource Allocation

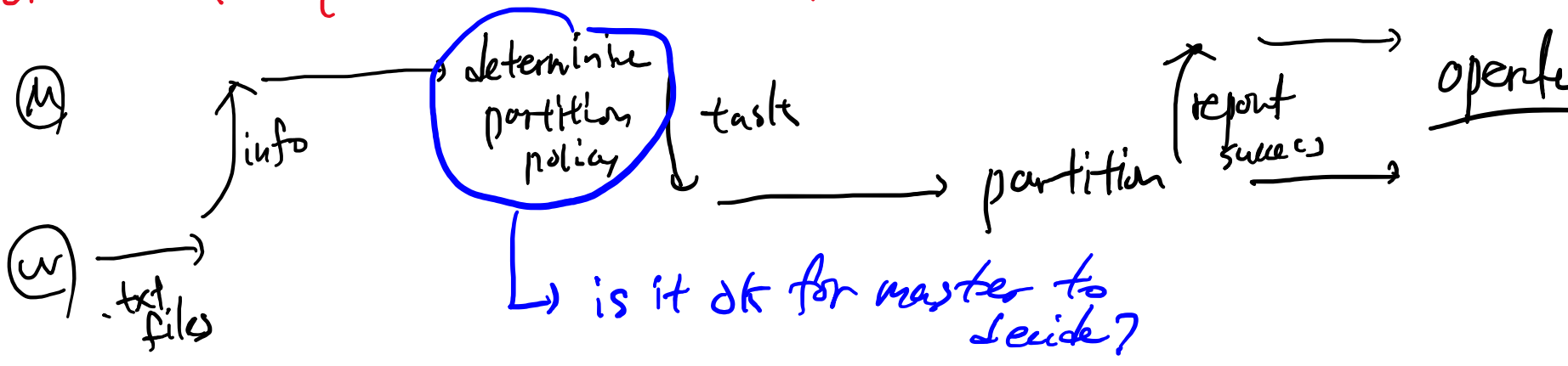
v Multithreading. ⇒ 1 partition / thread. golden rule

⇒ Expect equi-memory distribution to each threads.
(::sort is simple!)

Threads ↑ : (+) can scale
(−) Disk IO ↑

Threads ↓ : (+) Disk IO ↓
(−) might not scale.

#Partition policy → not decided yet. (consider file size, #threads, memory).



Networking

- Use Protobuf (gRPC) for networking
- Need to study more about networking + async programming!

Shuffle

- Might be the most resource-consuming task in whole job
- Might need to consider assigning a thread only for shuffle behavior.
  - Consider case when Partition / shuffle / sort overlaps.

Merge Sort

- When executing merge sort, it is obvious that the whole data won't scale.
- Thus we need mechanism to spill data to disk. (It is differenting from simply writing data directly to file!)
  - A typical DBMS task.
  - Any simple library to use in Scala?