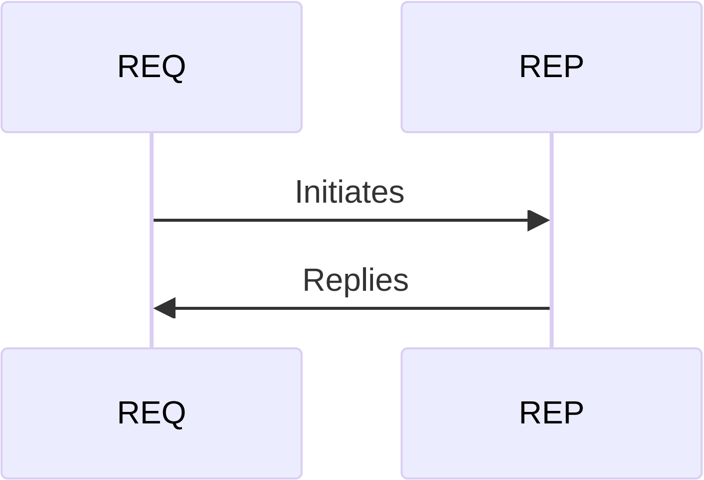
Master Thesis

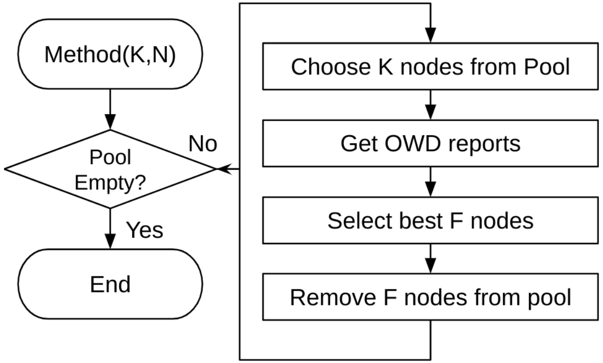
ZMQ Sockets

- ZMQ_REQ/ZMQ_REP
  - Send/Receiver order has to be respected
  - Reply remembers only last received address
- Other Sockets:
  - Push/Pull
  - Pub/Sub
  - Pair/Pair
  - Router/Dealer



Testbench and Heuristic

1. Allocate **N** VMs.
2. Run Jasper on Vanilla Setup
    1. Terminate
    2. Store Results
3. Apply Proposed Heuristic
4. Evaluate Performance



Method(K,N)

Pool Empty?

No

Yes

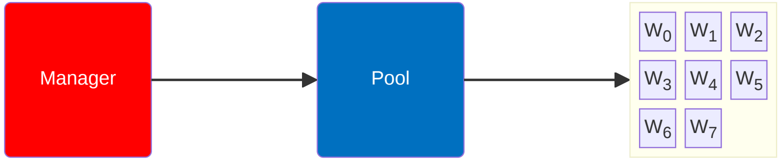
End

Choose K nodes from Pool

Get OWD reports

Select best F nodes

Remove F nodes from pool



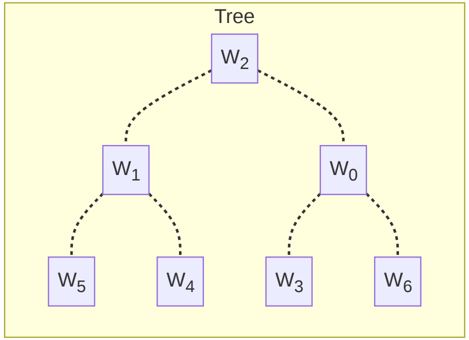
Manager

Pool

W_0 W_1 W_2

W_3 W_4 W_5

W_6 W_7



Tree

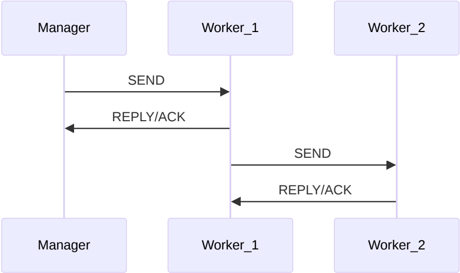
W_2

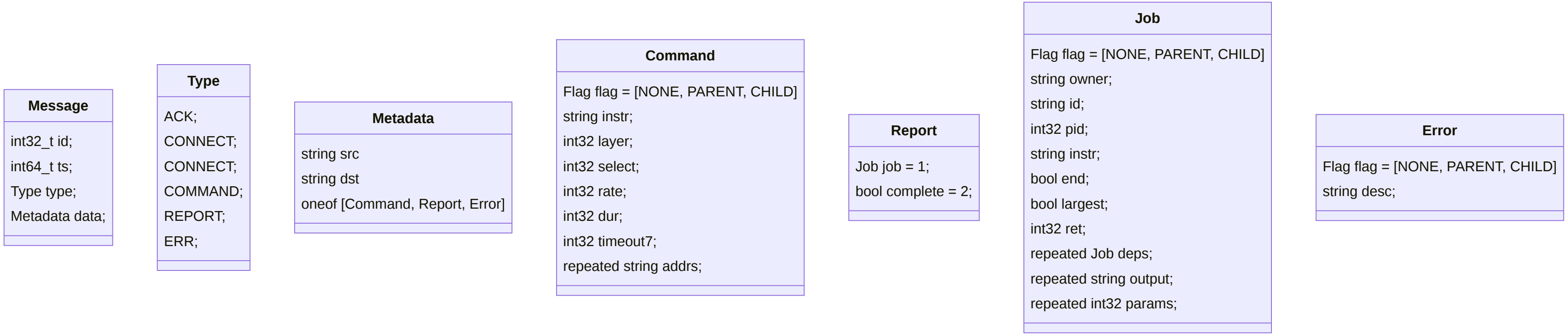
W_1 W_0

W_5 W_4 W_3 W_6

Manager x Worker: Communication

- ZMQ Sockets
- Pairwise send and reply initiated by Manager
- Manager: ZMQ_REQ
- Worker: ZMQ_REP





Message

int32_t id;
int64_t ts;
Type type;
Metadata data;

Type

ACK;
CONNECT;
CONNECT;
COMMAND;
REPORT;
ERR;

Metadata

string src
string dst
oneof [Command, Report, Error]

Command

Flag flag = [NONE, PARENT, CHILD]
string instr;
int32 layer;
int32 select;
int32 rate;
int32 dur;
int32 timeout7;
repeated string addrs;

Report

Job job = 1;
bool complete = 2;

Job

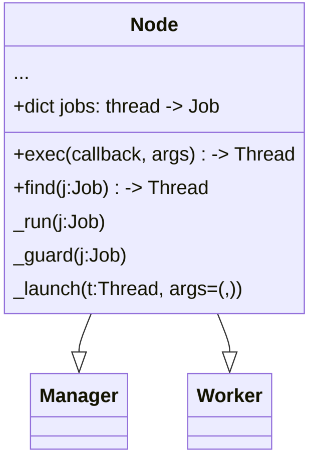
Flag flag = [NONE, PARENT, CHILD]
string owner;
string id;
int32 pid;
string instr;
bool end;
bool largest;
int32 ret;
repeated Job deps;
repeated string output;
repeated int32 params;

Error

Flag flag = [NONE, PARENT, CHILD]
string desc;

Manager x Worker: Data Structures

- Manager and Workers inherit Node Class
- Nodes:
  - own jobs, mapped via a dictionary of threads
  - are able of runnig jobs in separate threads
  - are able of guarding against job dependencies



Node

...

+dict jobs: thread -> Job

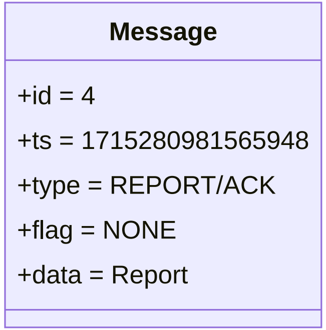
+exec(callback, args) : -> Thread

+find(j:Job) : -> Thread

_run(j:Job)

_guard(j:Job)

_launch(t:Thread, args=(,))



Message

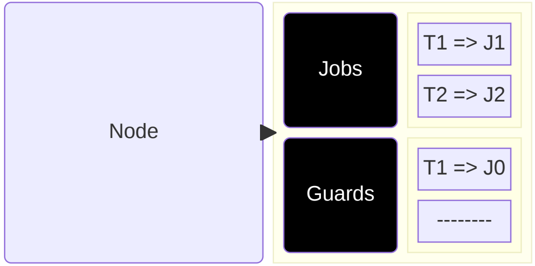
+id = 4

+ts = 1715280981565948

+type = REPORT/ACK

+flag = NONE

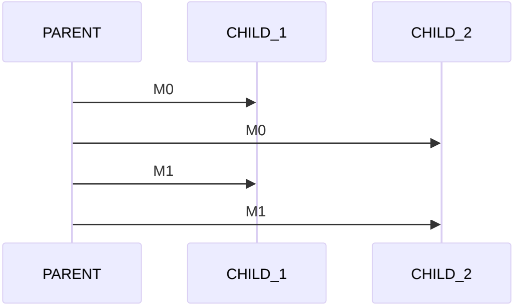
+data = Report



Parent x Child UDP

- Parent sends messages to multiple children
- Child:
  - Waits for stream start
  - Stores latency difference
  - prints to `stdout` 90% percentile latency

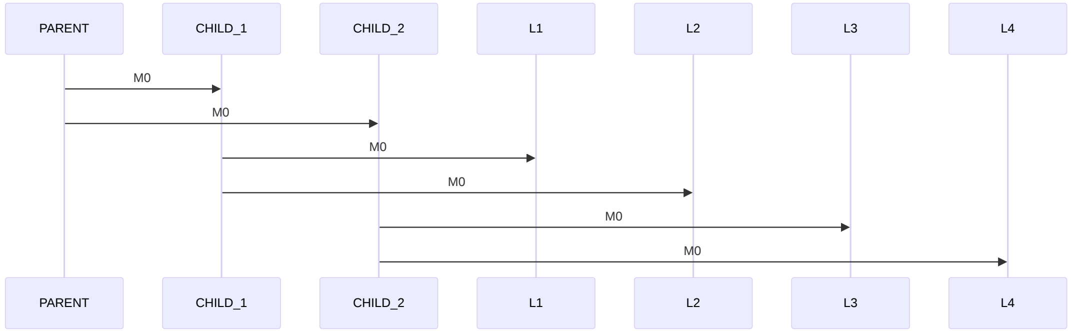
```
typedef struct MsgUDP {
    uint32_t id;
    uint64_t ts;
} MsgUDP_t;

double get_percentile(const std::vector<int64_t>& data,
                      double percentile) {
    // stuff
}
```



MCAST Tree Performance

- Root:
  - Sends messages to children
- Proxies:
  - Forwards messages from parent to children
- Leaves:
  - Stores latency difference
  - prints to `stdout` 90% percentile latency

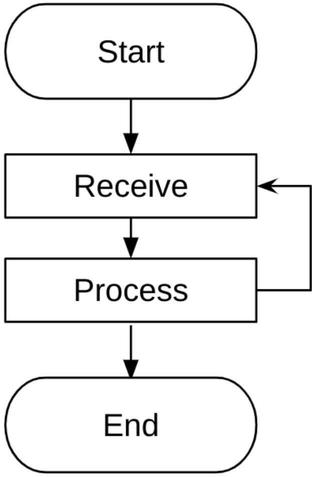
```
typedef struct MsgUDP {
    uint32_t id;
    uint64_t ts;
} MsgUDP_t;

double get_percentile(const std::vector<int64_t>& data,
                      double percentile) {
    // stuff
}
```



Worker State Machine

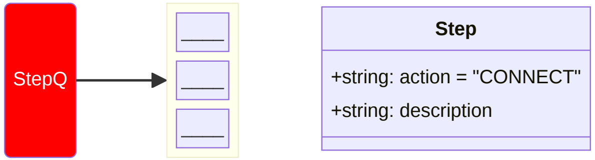
- Workers are reply sockets
- Bind and block on `recv()`
- Process message based on type

```python
while(True):
    m = self.recv_message()
    match m.type:
        case CONNECT: self.connectACK(m)
        case COMMAND: self.commandACK(m)
        case REPORT:  self.reportACK(m)
        case _:       raise RuntimeError()
```

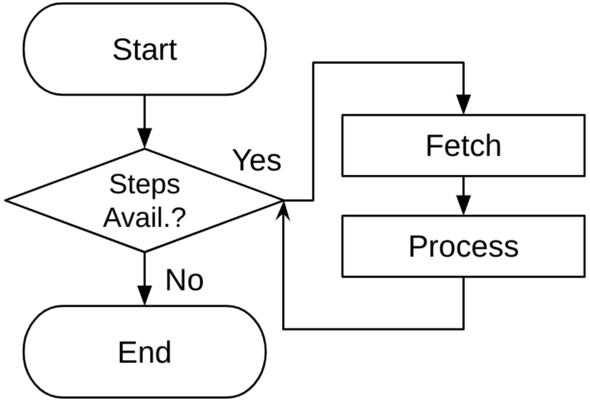


Manager State Machine

- Manager actively sends requests to workers
- Fetches steps from `step_queue`
- Process steps based on action type



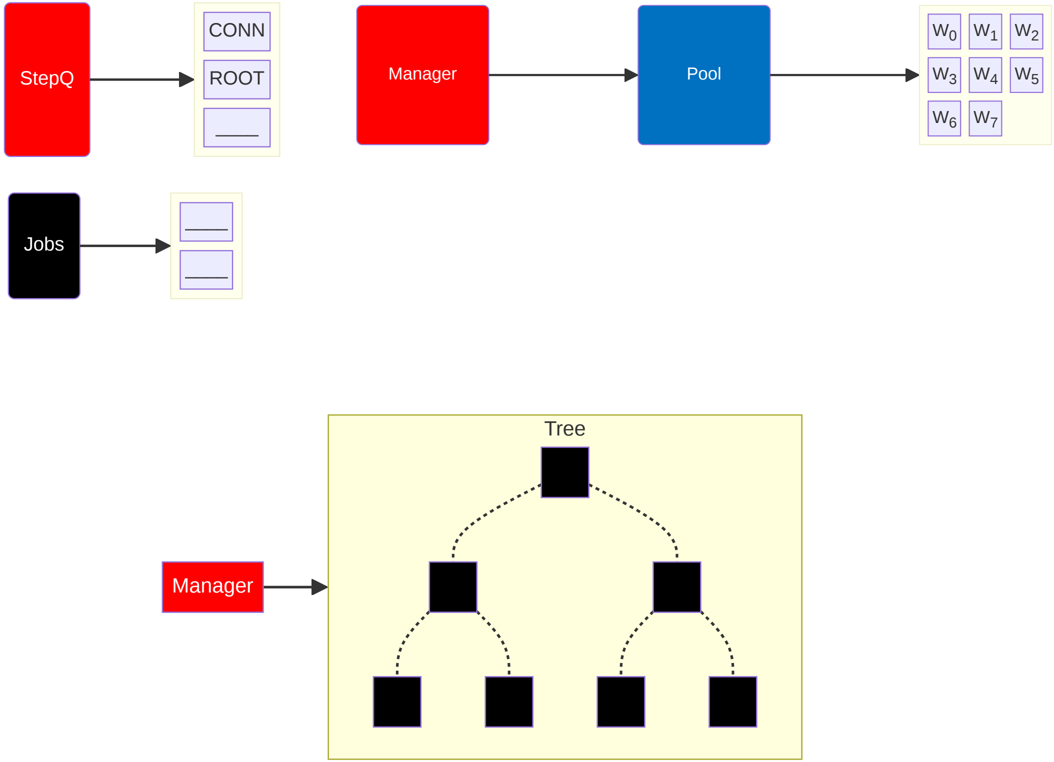
```
while(True):
    step = self.pop_step()
    if not step: break
        match step["action"]:
            case "CONNECT": self.establish()
            case "ROOT":    self.root()
            case "REPORT":  self.report()
            case _:         raise RuntimeError()
```



Manager x Worker: Workflow [Step_i = 0]

- Manager reads in YAML *script*
- Populates step queue
- Fetches first step

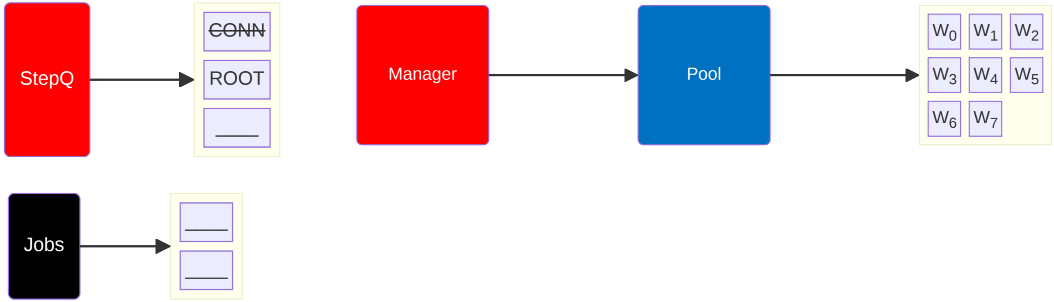
```yaml
name: DEFAULT
hyperparameter: 0.5
rate: 10
duration: 10
addrs:
  - "localhost:9091"
  - "localhost:9092"
  - "localhost:9093"
  - "localhost:9094"
  - "localhost:9095"
  - "localhost:9096"
steps:
  - action: "CONNECT"
    description: "Establish connection workers."
    data: 0
  - action: "ROOT"
    description: "Choose root among worker nodes."
    data: 0
```

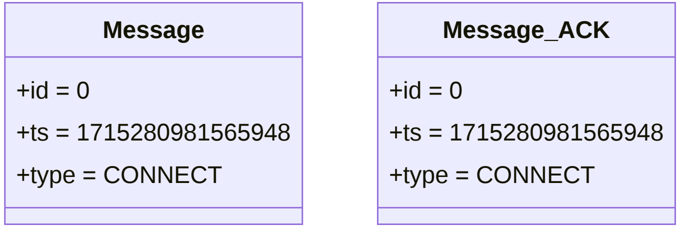


Manager x Worker: Workflow [Step_i = 1]

- ACTION: CONNECT

1. Loops through all workers
    1. Establishes connection
    2. `Send()` CONNECT Messages
    3. `Recv()` ACK Messages
    4. Disconnects





Message

+id = 0

+ts = 1715280981565948

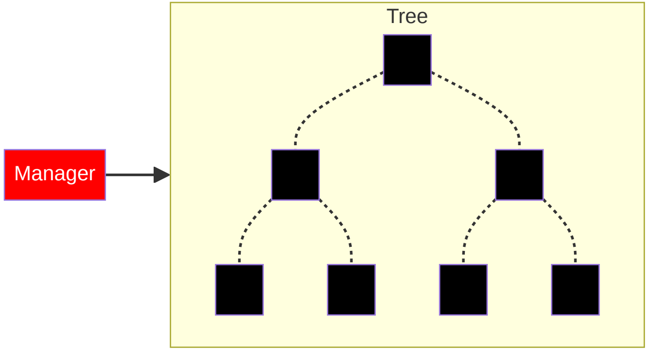
+type = CONNECT

Message_ACK

+id = 0

+ts = 1715280981565948

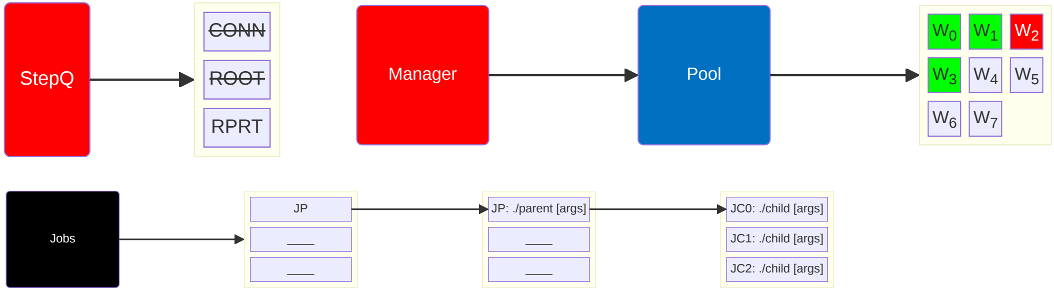
+type = CONNECT



Manager x Worker: Workflow [Step_i = 2]

- ACTION: ROOT

1. Select root from pool *(idx=2)*
2. Commands Root/Parent: `./parent <args`
    1. Commands children: `./child <args`
    2. Store their Jobs `JC`
    3. Starts Job `JP` and returns it via ACK
3. Pushes: `Step=REPORT` and stores `JP`

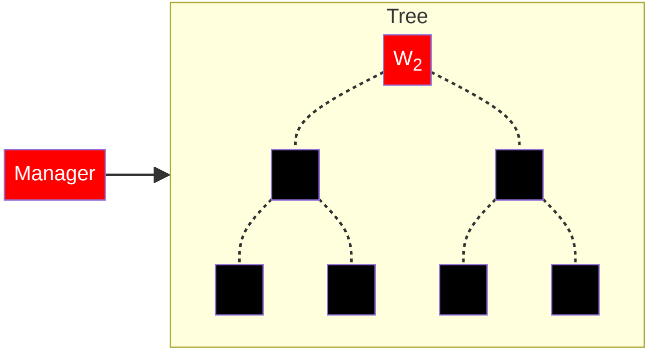


Message_P
+id = 1
+ts = 1715280981565948
+type = COMMAND
+flag = PARENT
+data = CommandP

Message_C
+id = 1
+ts = 1715280981565948
+type = COMMAND
+flag = PARENT
+data = CommandC

Message_P_ACK
+id = 1
+ts = 1715280981565948
+type = ACK
+data = JP

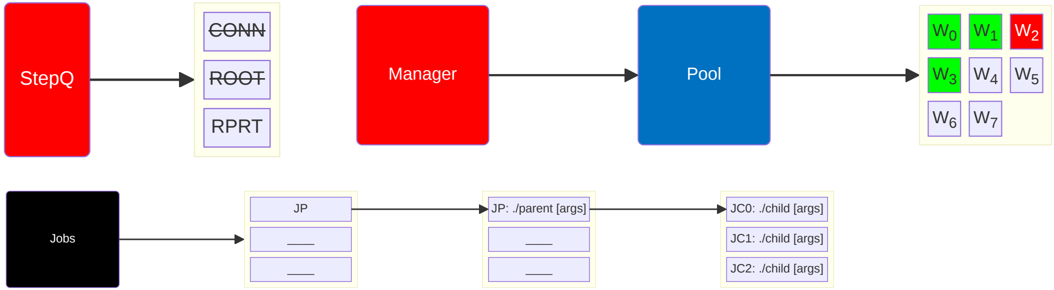
Message_C_ACK
+id = 1
+ts = 1715280981565948
+type = ACK
+data = JC



Manager x Worker: Workflow [Step_i = 2.1]

- ACTION: ROOT

1. Select root from pool *(idx=2)*
2. Commands Root/Parent: `./parent <args`
   1. Commands children: `./child <args`
   2. Store their Jobs `JC`
   3. Starts Job `JP` and returns it via ACK
3. Pushes: `Step=REPORT` and stores `JP`



Message_P

+id = 1
+ts = 1715280981565948
+type = COMMAND
+flag = PARENT
+data = CommandP

Message_C

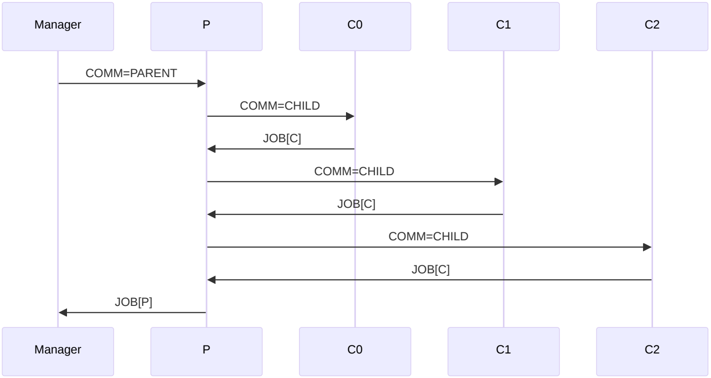
+id = 1
+ts = 1715280981565948
+type = COMMAND
+flag = PARENT
+data = CommandC

Message_P_ACK

+id = 1
+ts = 1715280981565948
+type = ACK
+data = JP

Message_C_ACK

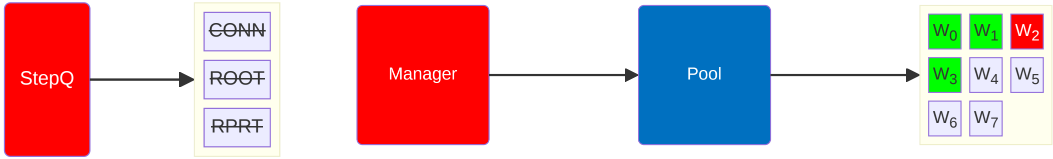
+id = 1
+ts = 1715280981565948
+type = ACK
+data = JC

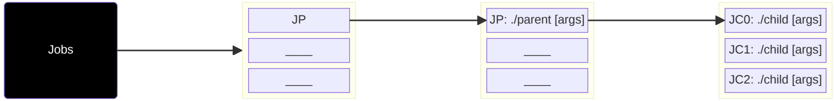


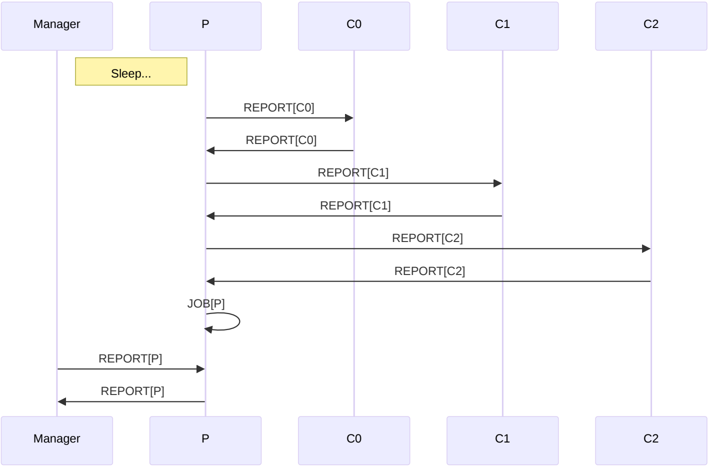
Manager x Worker: Workflow [Step_i = 3.0]

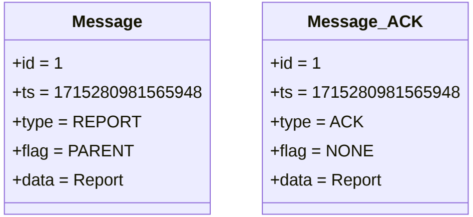
- ACTION: REPORT

1. Pops next report
2. Sleeps until trigger timestamp
3. Probes report on pending job to owner
    1. Parent also probes for reports
    2. Parent aggregates results and reports









Message

+id = 1
+ts = 1715280981565948
+type = REPORT
+flag = PARENT
+data = Report

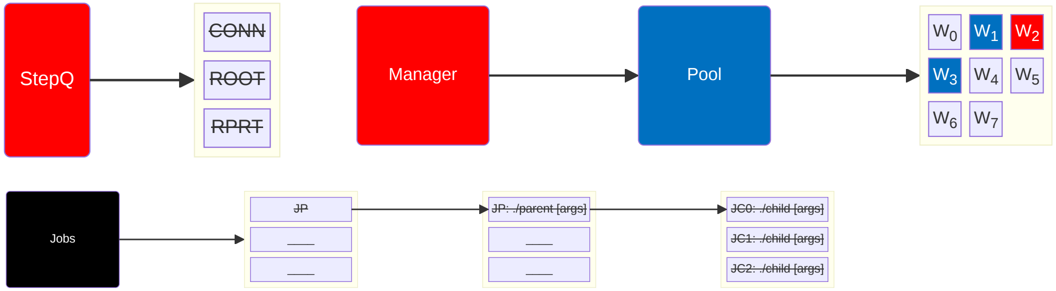
Message_ACK

+id = 1
+ts = 1715280981565948
+type = ACK
+flag = NONE
+data = Report

Manager x Worker: Workflow [Step_i = 3.1]

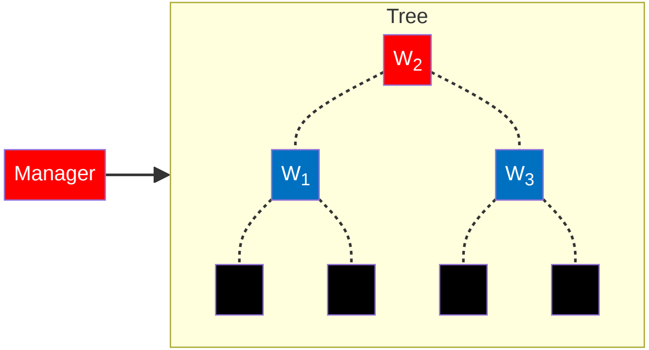
- ACTION: REPORT

1. Pops next report
2. Sleeps until trigger timestamp
3. Probes report on pending job to owner
    1. Parent also probes for reports
    2. Parent aggregates results and reports


StepQ
~~CONN~~
~~ROOT~~
~~RPRT~~
Manager
Pool
W_0 W_1 W_2
W_3 W_4 W_5
W_6 W_7
Jobs
~~JP~~
____
____
~~JP: ./parent [args]~~
____
____
~~JC0: ./child [args]~~
~~JC1: ./child [args]~~
~~JC2: ./child [args]~~

Message
+id = 1
+ts = 1715280981565948
+type = REPORT
+flag = PARENT
+data = Report

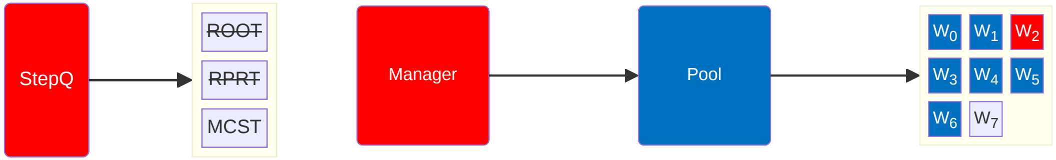
Message_ACK
+id = 1
+ts = 1715280981565948
+type = ACK
+flag = NONE
+data = Report

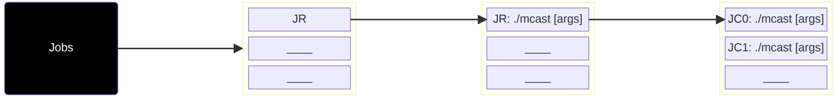

Manager
Tree
W_2
W_1
W_3

Manager x Worker: Workflow [Step_i = 4.0]

- ACTION: MCAST

1. Tree complete
2. Manager trickles down `mcast` job
3. Reports are handled between workers



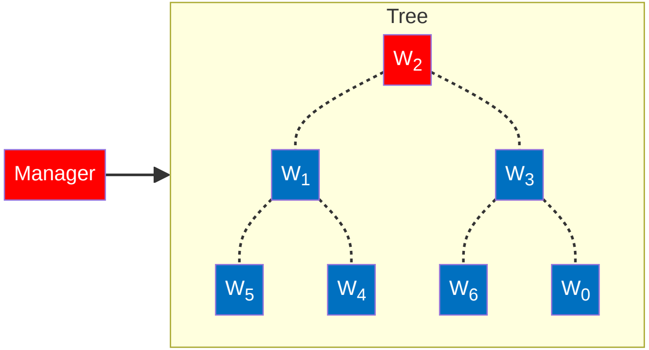


Message

+id = 1
+ts = 1715280981565948
+type = REPORT
+flag = MCAST
+data = Report

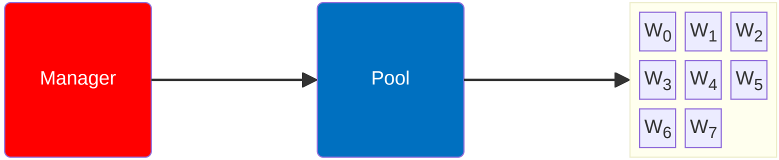
Message_ACK

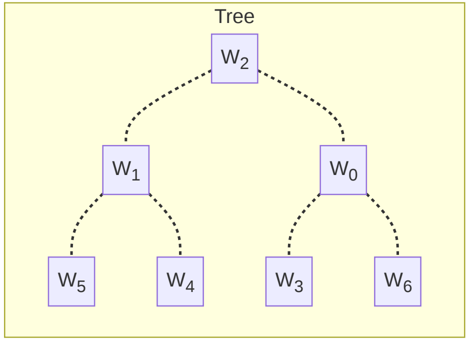
+id = 1
+ts = 1715280981565948
+type = ACK
+flag = NONE
+data = Report

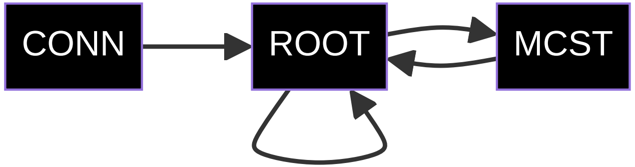


Summary

1. Establish connection to workers
2. Do for [`best` , `worst` , `random`] trees:
    1. Choose `root`
    2. Until Tree is complete
        1. Start `parent x child` jobs
        2. Probe for results
        3. When done: modify `Tree` and `Pool` accordingly
    3. Store results







Draw

Draw