

## Runtime (part A) - Protocol

### Runtime



```
input()
checkSyntax()
programs.append(
progID: pc, code, variables)
start_thread(progID)
```

run filename <args>

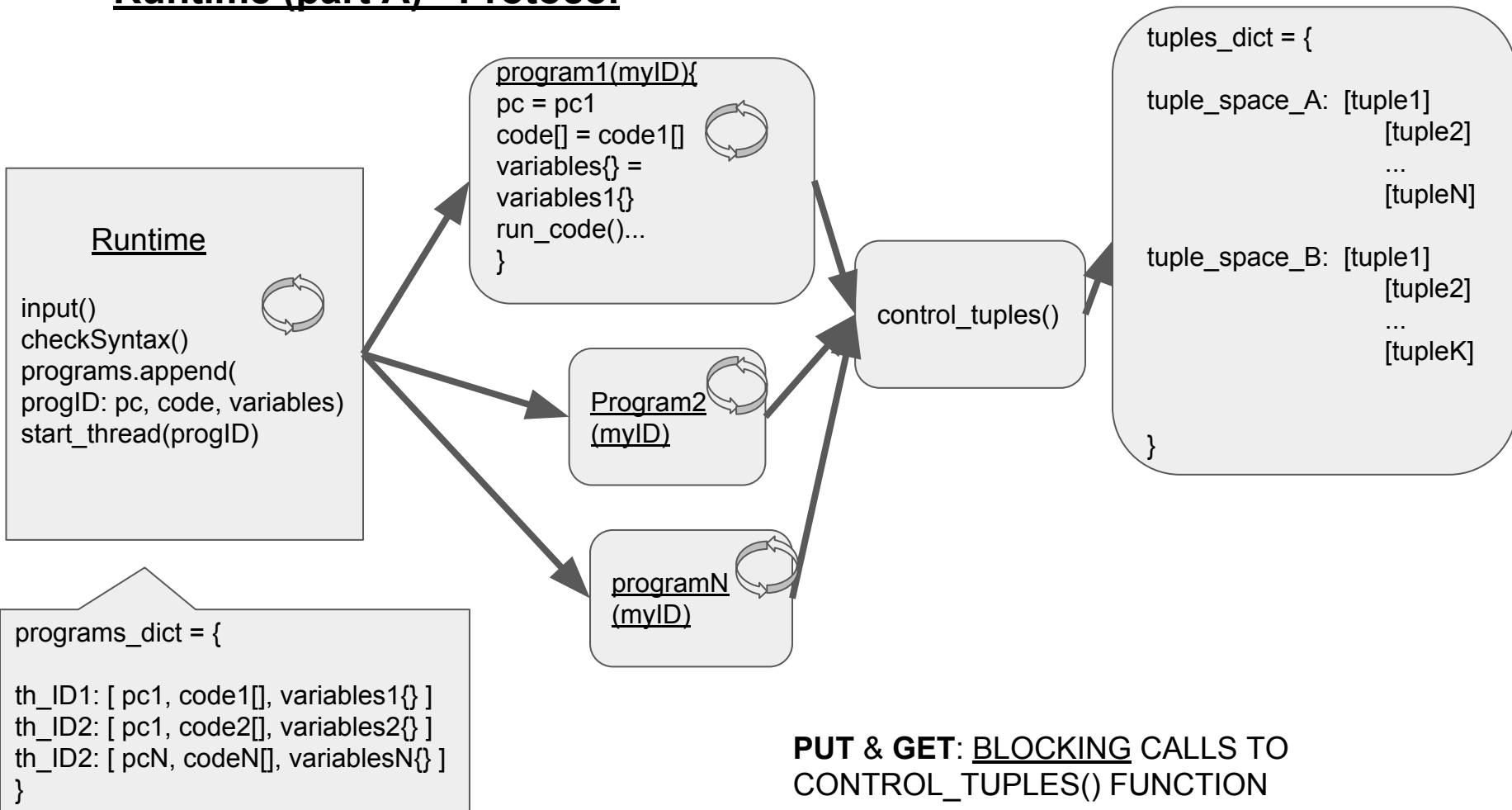
list #print programs id, args, tuples

kill filename

programs\_dict = {

```
th_ID1: [ pc1, code1[], variables1{} ]
th_ID2: [ pc1, code2[], variables2{} ]
th_ID2: [ pcN, codeN[], variablesN{} ]
}
```

## Runtime (part A) - Protocol

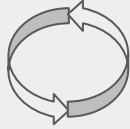


## Runtime (part A) - Implementation

```
runtime() {
```

```
    ID = 0
```

```
    While True:
```



```
        input_command()
```

```
        If instr == run:
```

```
            ID += 1
```

```
            pc = 0
```

```
            for each line in arg[0]:
```

```
                code[pc] = checkSyntax(line)
```

```
                pc += 1
```

```
            replace_labels_with_num_of_line()
```

```
            variables = {'arg[i]': value_of_arg[i]  $\forall$  i }
```

```
            programs_dict[ID] = [pc=0, code, variables)
```

```
            start_thread( ID )
```

```
        elif instr == kill:
```

```
            flagStop = True
```

```
            delete programs_dict[id]
```

```
        Elif instr == exit:
```

```
            for each ID in programs_dict:
```

```
                flagStop[ID] = True
```

```
            return
```

```
        elif instr == list:
```

```
            Print_all( ID + arguments)
```

```
    }
```

```
thread_code_implementation ( myID ){
```

```
    code = programs_dict(myID)[1]
```

```
    variables = programs_dict(myID)[2]
```

```
    pc = 0
```

```
    while pc < len(code)
```

```
        If flagStop == True:
```

```
            programs_dict[progr_id][0] = pc
```

```
            programs_dict[progr_id][2] = variables
```

```
            Exit
```

```
        if instr == 'ADD':
```

```
            ...
```

```
        else if instr == 'PUT'
```

```
            control_tuples('PUT', tuple_space, input_tuple, myID)
```

```
        else if instr == 'GET'
```

```
            control_tuples('GET', tuple_space, input_tuple, myID)
```

```
            update_variables()
```

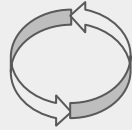
```
        else if instr == 'EXT':
```

```
            break
```

```
    delete programs_dict( myID )
```

```
    exit
```

```
}
```

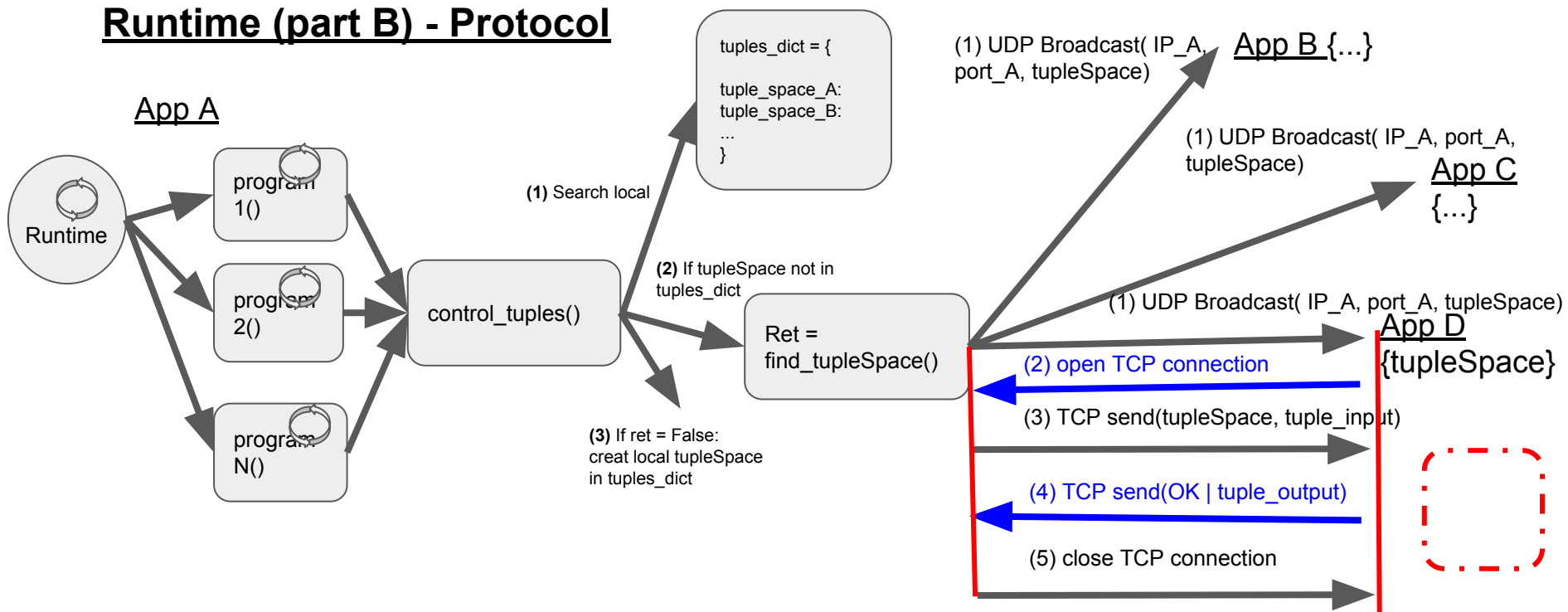


**PUT & GET: BLOCKING CALLS TO  
CONTROL\_TUPLES() FUNCTION**

## Runtime (part A) - Implementation

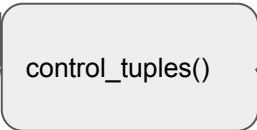
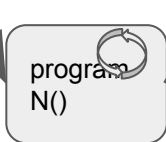
```
control_tuples( instr, tuple_space, input_tuple, th_ID ){  
    if tuple_space not in tuples_dict:  
        #check if system is distributed  
        if configuration.flagDistributed == True:  
            ret = find_tuple_broadcast(instr, tuple_space,  
input_tuple)  
            return  
        else:  
            tuples_dict[tuple_space] = []  
  
    if instr == 'PUT':  
        #put tuple in tuples_dict  
        event_tuple.set() #notify wait program  
        event_tuple.clear()  
    Elif instr == 'GET':  
        If tuple not in tuples_dict:  
            event.wait()  
        Else:  
            tuples_dict[tuple_space].pop()  
}
```

# Runtime (part B) - Protocol



# Runtime (part B) - Protocol

## App A



(1) Search local

```
tuples_dict = {  
  tuple_space_A:  
  tuple_space_B:  
  ...  
}
```

(2) If tupleSpace not in  
tuples\_dict

Ret =  
find\_tupleSpace()

(3) If ret = False:  
creat local tupleSpace  
in tuples\_dict

```
tuple_over_TCP() {  
  connect_with_sender()  
  tupleSpace, tuple_input = packet  
  control_tuples(tupleSpace, tuple_in )  
}
```

```
receiver_UDP(){  
  While True:  
    rcv_upd_packets()  
    If tupleSpace in local:  
      create_thread()  
}
```

(1) UDP Broadcast( IP\_A,  
port\_A, tupleSpace)

App B {...}

(1) UDP Broadcast( IP\_A, port\_A,  
tupleSpace)

App C  
{...}

(1) UDP Broadcast( IP\_A, port\_A, tupleSpace)

App D  
{tupleSpace}

(2) open TCP connection

(3) TCP send(tupleSpace, tuple\_input)

(4) TCP send(OK | tuple\_output)

(5) close TCP connection



```
find_tuple_broadcast(instr, tuple_space, input_tuple):  
    #create udp socket for bradcast the tuple  
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  
    s.setsockopt(socket.SOL_SOCKET,  
socket.SO_BROADCAST, 1)  
    s.sendto(packet,(broadcast_IP,udp_port))  
  
    s.close()  
    send_tuple_tcp(ip_addr, port, instr, tuple_space, input_tuple)  
  
}
```

```
send_tuple_tcp(ip_addr, port, instr, tuple_space,  
input_tuple):  
    #create tcp socket for sending tuple  
    s = socket.socket(socket.AF_INET,  
socket.SOCK_STREAM)  
    s.bind((TCP_IP, TCP_PORT))  
    packet = (instr + tuple_space +input_tuple)  
    s.sendto(packet)  
  
    while True:  
        data = conn.recv(2048)  
        if data != None:  
            if b'OK' in data:  
                return data  
            elif b'OK' not in data:  
                return 0  
        else:  
            return 0  
  
}
```

## Runtime (part C) - Protocol-Directory

```
#create tcp communication for info about ip and  
port of hosts for tcp communication
```

```
hosts = {}  
s = socket.socket(socket.AF_INET,  
socket.SOCK_STREAM) s.bind((ipAddr, 0))  
port = s.getsockname()[1]
```

```
s.listen(100)
```

```
while True:
```

```
    connection, host_addr = s.accept()
```

```
    data = connection.recv(256)
```

```
    If data == 'List':
```

```
        connection.send(hosts)
```

```
    elif data == 'Join':
```

```
        #data = [ip, port] of hosts
```

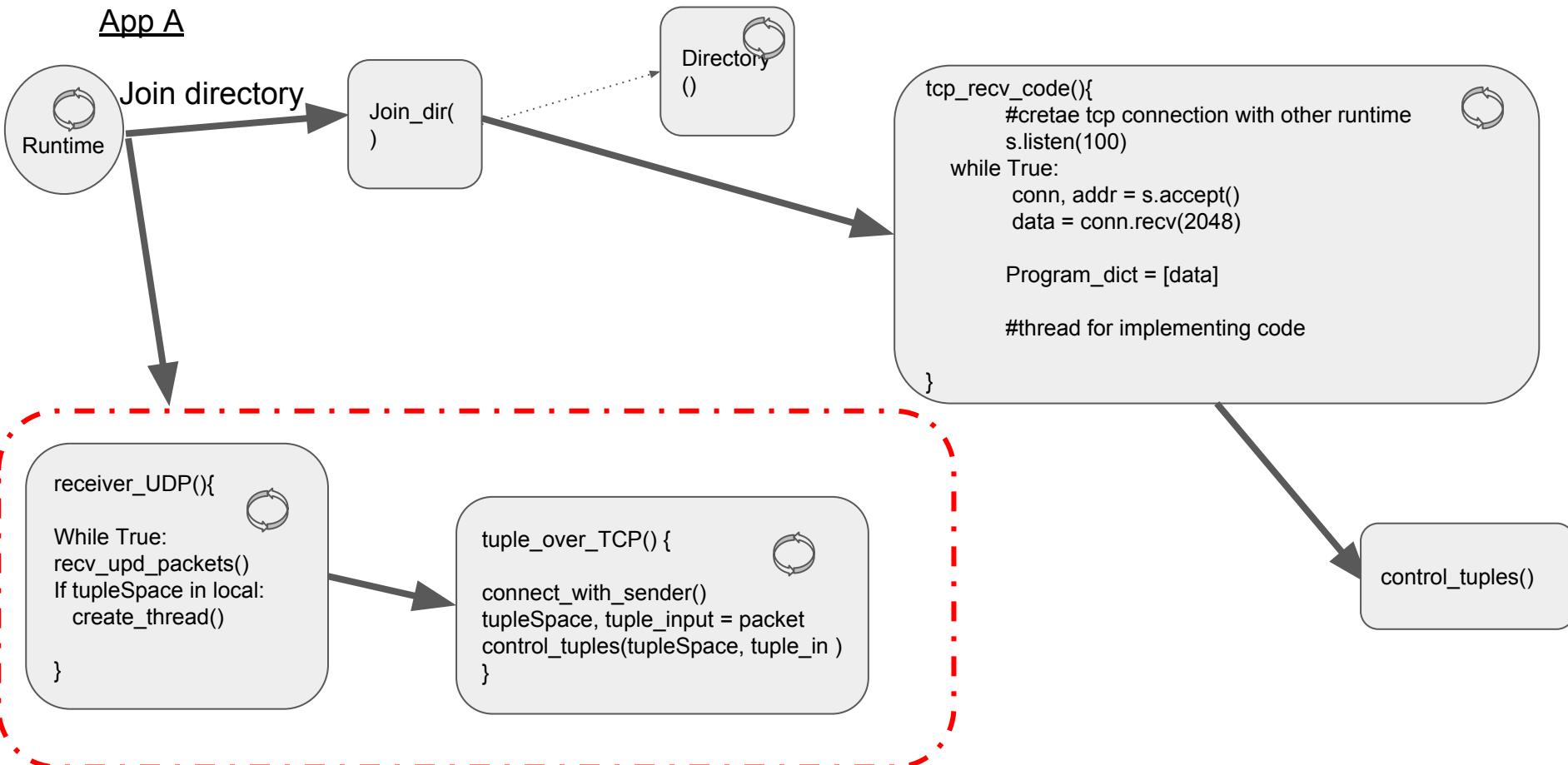
```
        hosts[runtime] = [ip_addr, port]
```

```
        connection.send('OK')
```

```
hosts = {}  
hosts = {runtime1: [ip1, port1],  
         'runtime2': [ip2, port2],  
         ....  
         }
```



# Runtime (part C) - Protocol-Directory



## Runtime (part C) - Protocol

App A

Runtime

run filename <args>:

control\_tuples()

Migrate <id> <ip> <port>

```
#terminate current id
flagStop[progr_id] = True
th=tcp_send_code_thread(progr_id,
ip_addr, port)
```

```
kill <id>:
    #check if program exists in
    program_dicts
    flagStop[id] = True
    r = programs_dict.pop(progr_id)
```

shutdown

```
tcp_send_code(progr_id, ip_addr, port) {
    sock = socket(AF_INET, SOCK_STREAM)
    sock.settimeout(None)
    sock.connect((ip_addr, port))

    #create packet: program_dict[progr_id]
    sock.send(packet)

    sock.close()

    programs_dict.pop(progr_id, None)
    return
}
```

**Shutdown:**

```
#connect with directory for getting list of hosts  
#search inlist if exists a different runtime
```

```
#if exists:
```

```
for i in programs_dict:
```

```
    th = tcp_send_code_thread(progr_id, new_ipaddr,  
new_port)  
    th.start()
```

```
programs_dict.clear()
```