File Servers

- Basic idea is simple
  - Clients open/read/write/close files indirectly, by going through servers, who access the disk
  - A client must be matched with a server
  - Clients do not store (permanent) files; servers do
  - Clients cache parts of (or whole) files

Basic Idea---Client

Note: numbers are for this *and* next slides

(1) send request to single global channel (*open*)

(4) receive reply on client specific channel

- reply contains a server id (*openreply[clientId]*)

(5) thereafter, send requests to channel indexed by server id (*filerequest[serverId]*)

(9) receive results on a different client specific channel (*filereply[clientId]*)

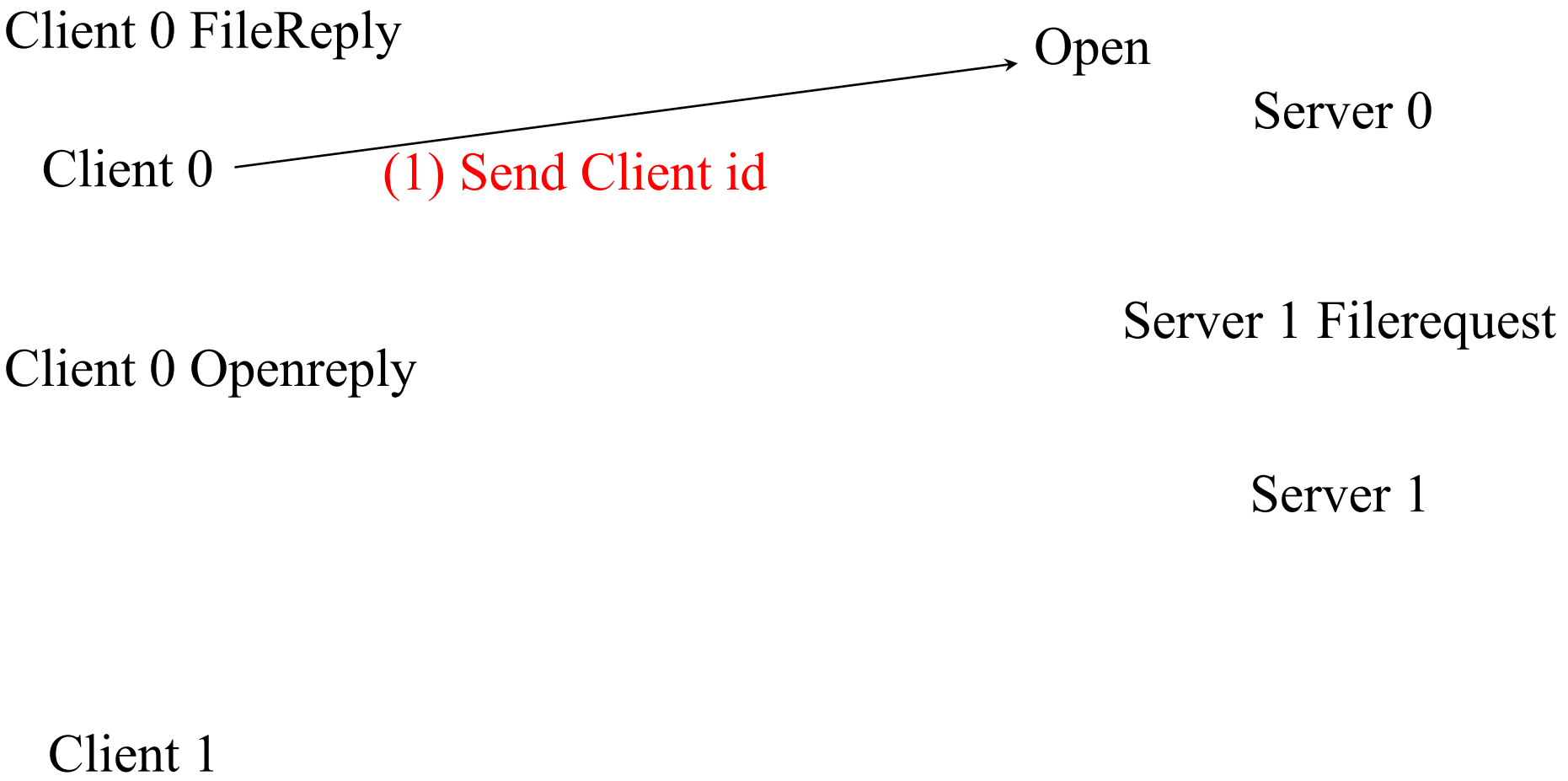
- eventually send a close message

Note: could have one channel on which servers send to clients (but extra argument marshalling)

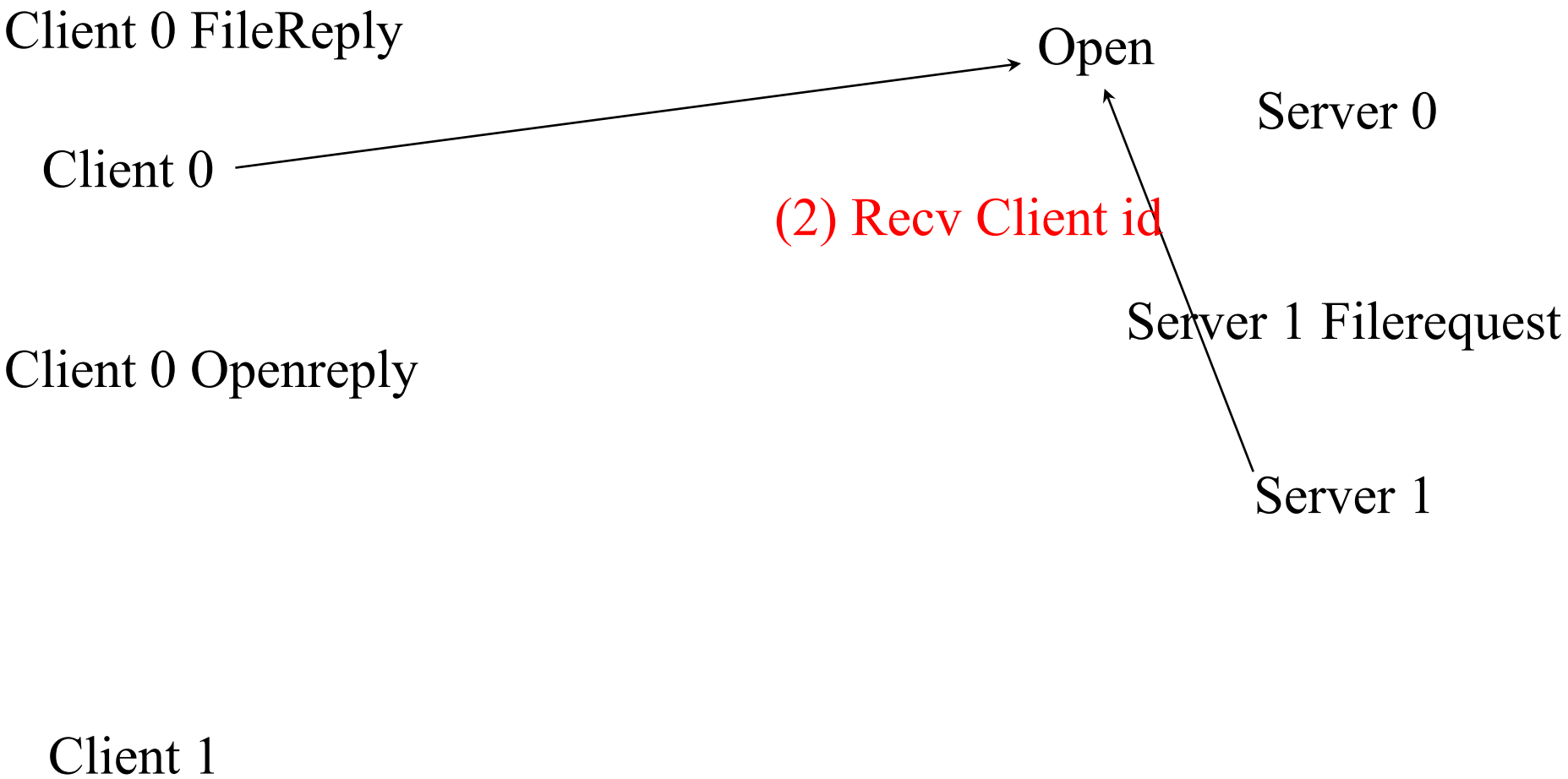
Basic Idea---Server

```
while (TRUE) {
  (2) receive client id on global channel (open)
  (3) send server id to that client's channel  (openreply[clientId])
  while close message not yet received {
      (6) recv requests on server specific channel (filerequest[servId])
      (7) process request
      (8) send reply to channel indexed by client (filereply[clientId])
   }
}
```

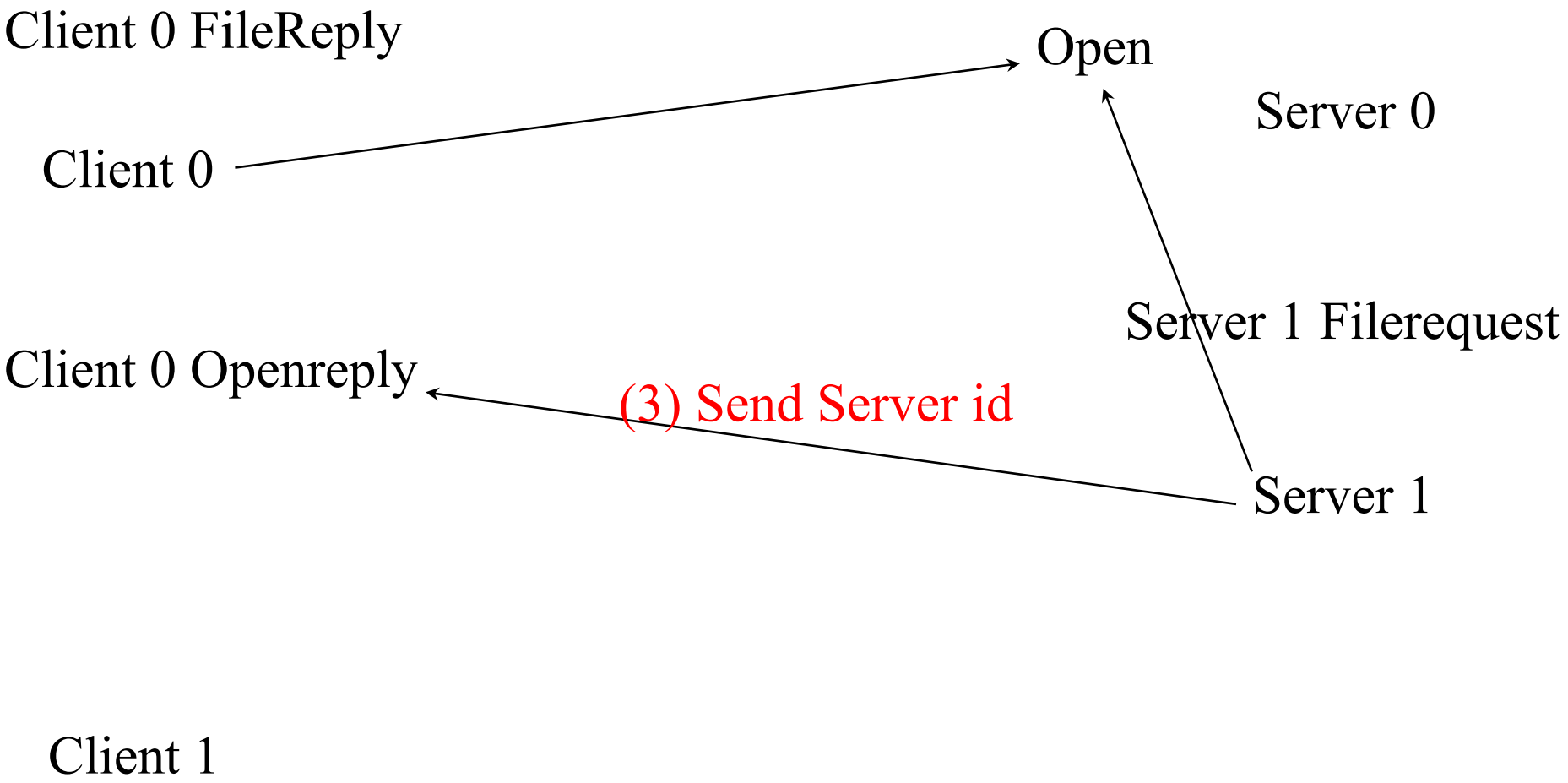
Picture of File Server, Many Channels



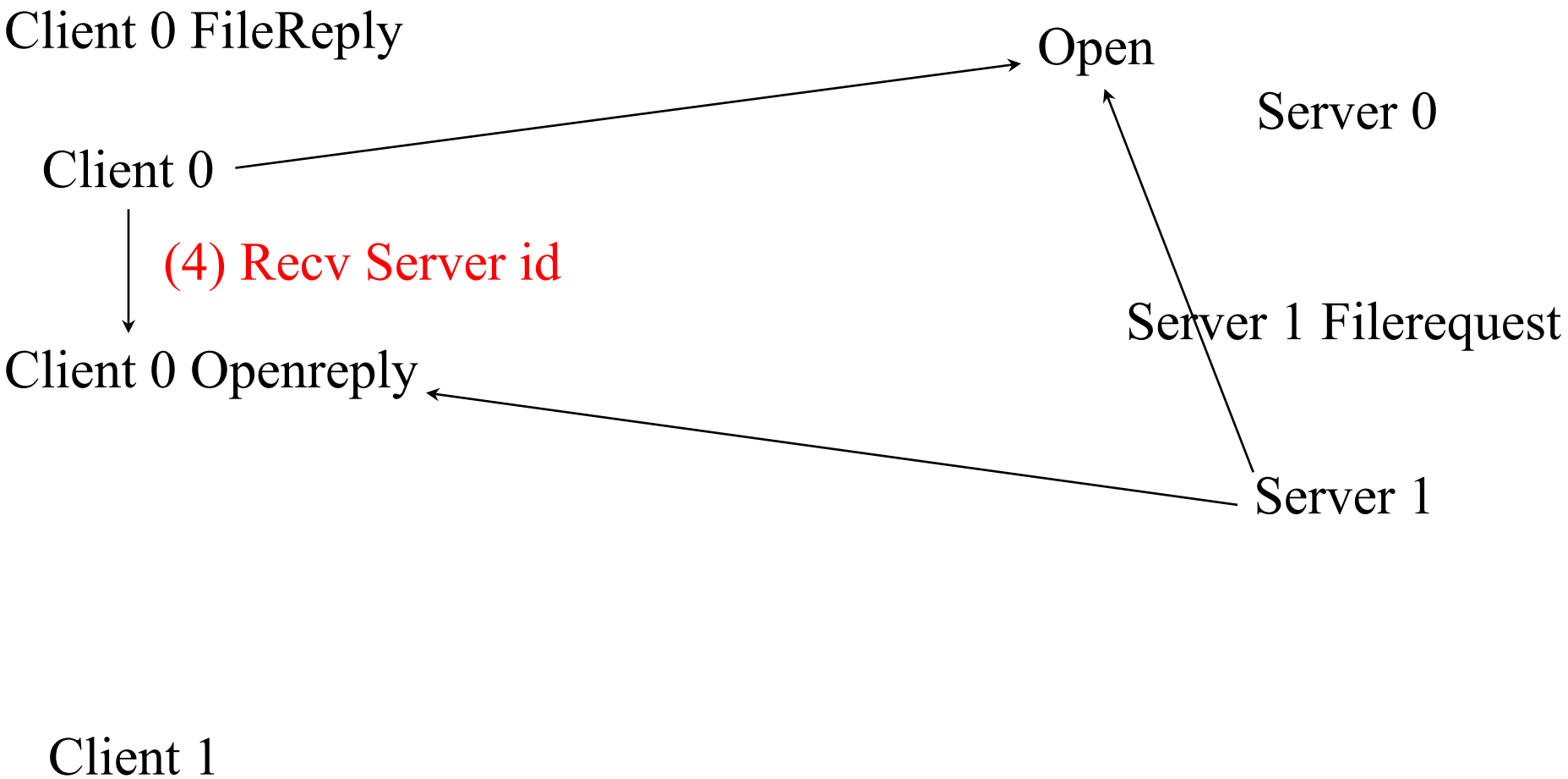
Picture of File Server, Many Channels



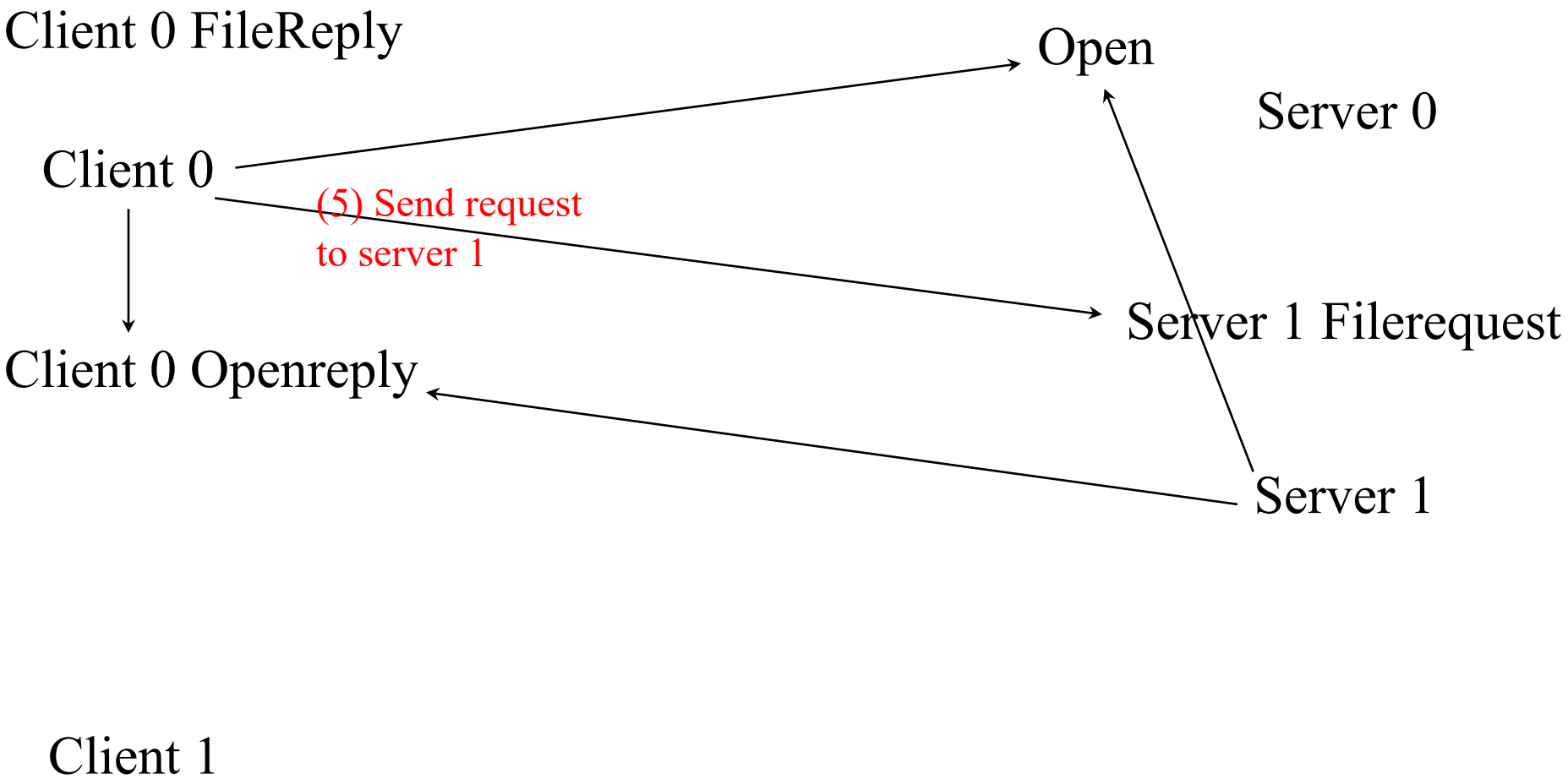
Picture of File Server, Many Channels



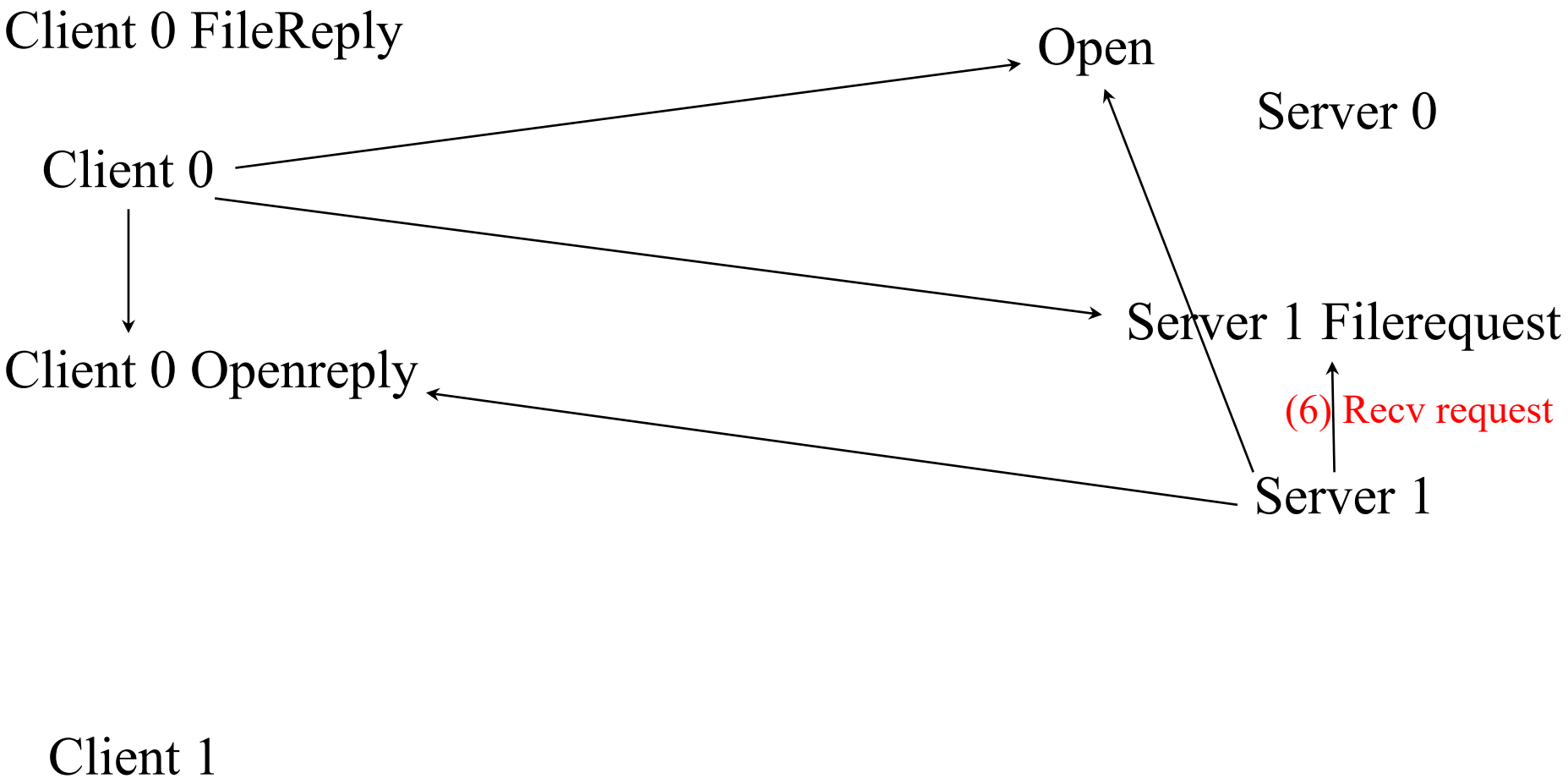
Picture of File Server, Many Channels



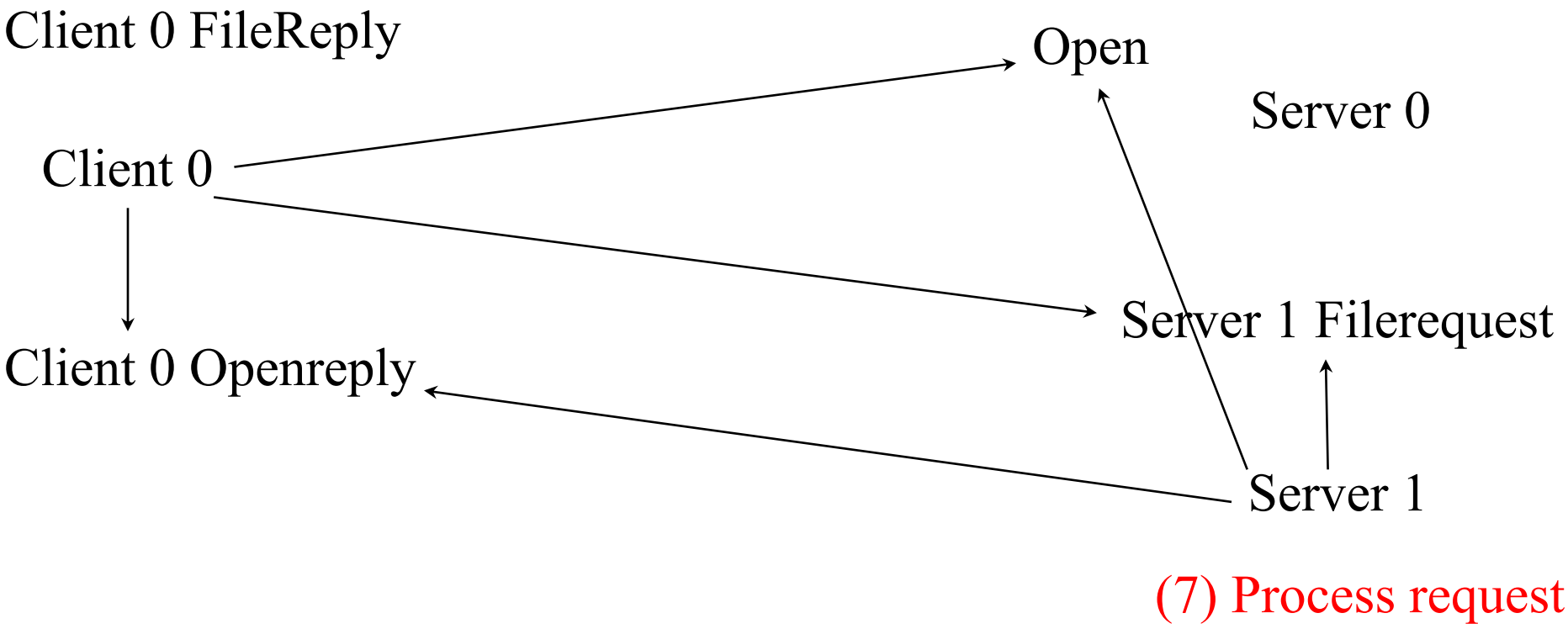
Picture of File Server, Many Channels



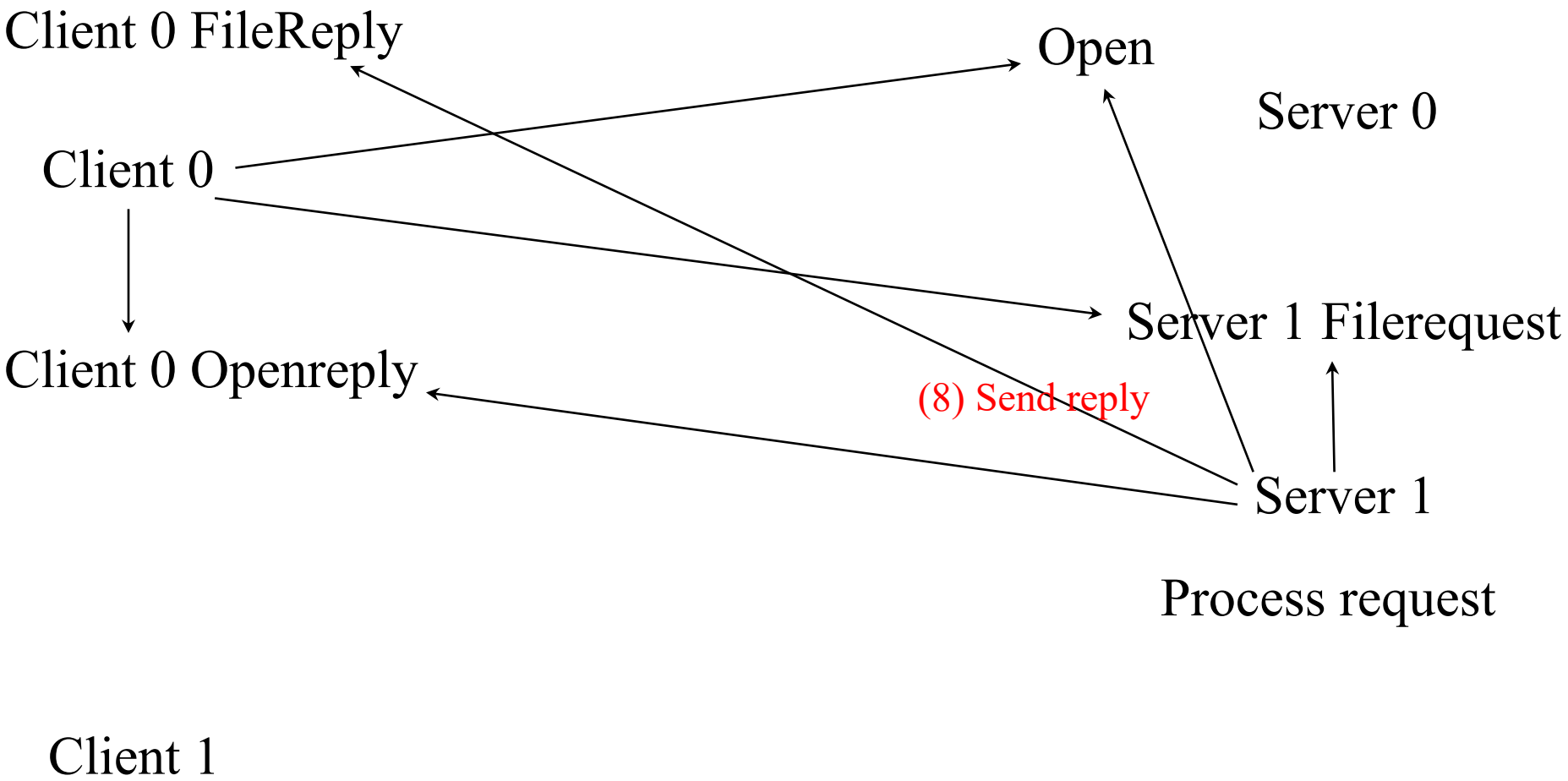
Picture of File Server, Many Channels



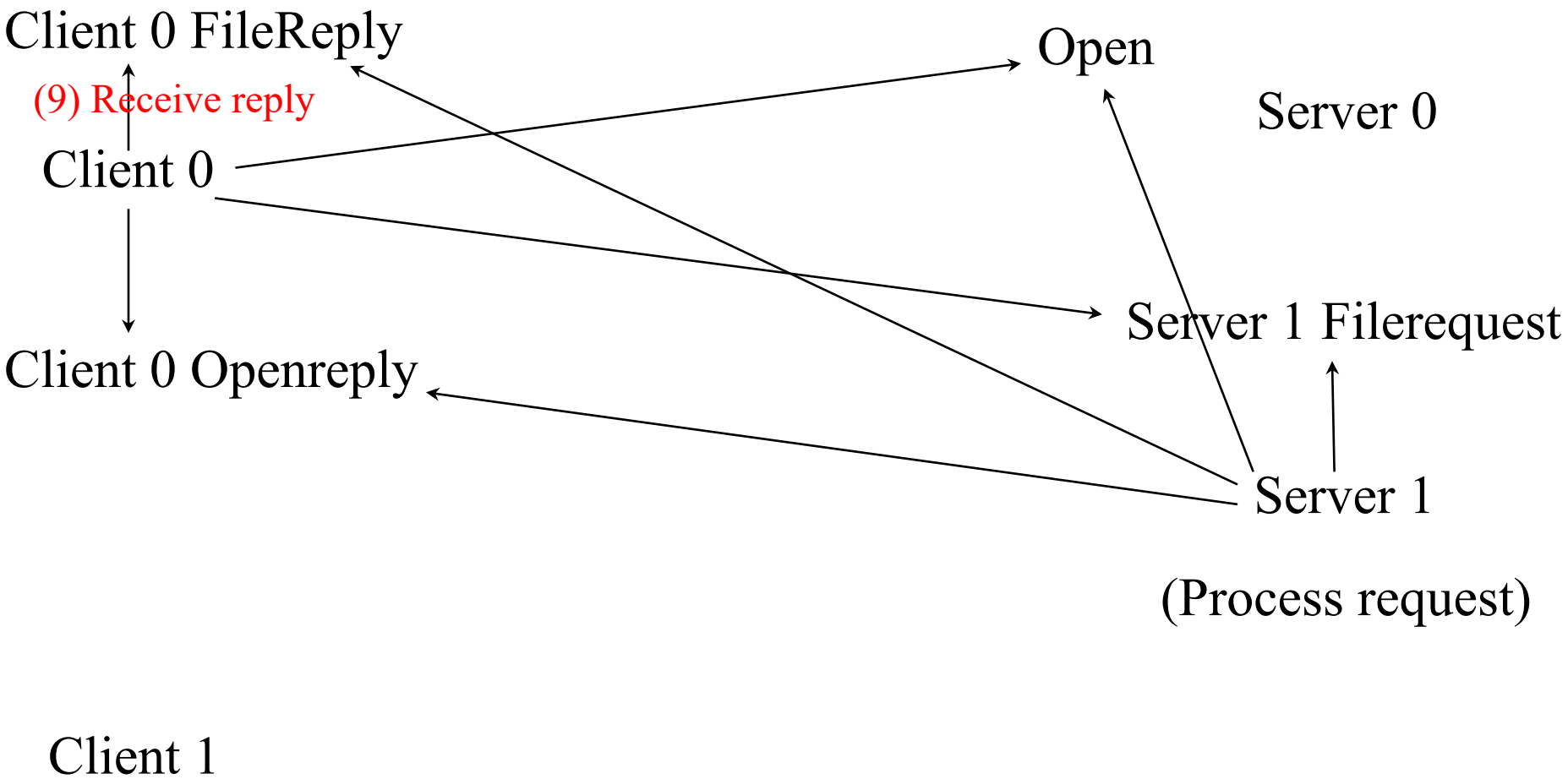
Picture of File Server, Many Channels



Picture of File Server, Many Channels



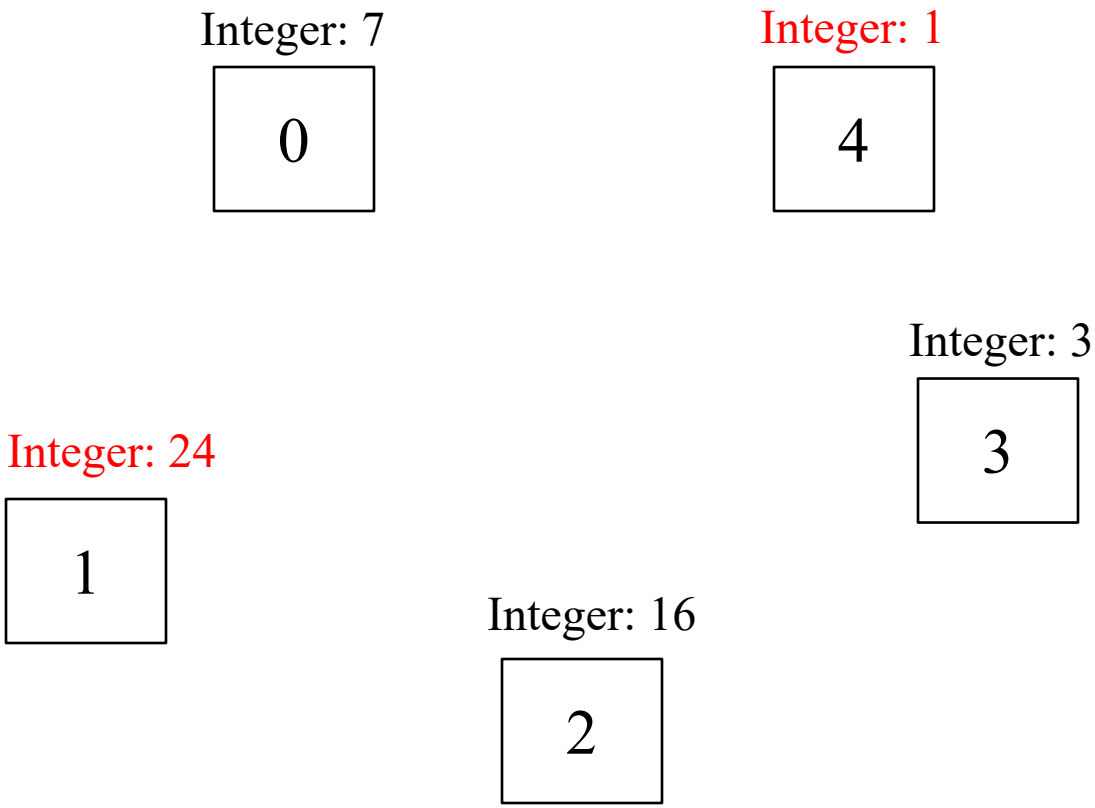
Picture of File Server, Many Channels



Notes on File Server

- Our solution used *conversational continuity*
  - Client talks to same server throughout the lifetime of the file
    - Advantage: caching on server side

Interacting Peers Picture



Interacting Peers

- Each process has an integer
  - Goal: find max and min integers
- Approaches
  - Centralized: use coordinator
  - Symmetric: every process sends value to every other
  - Ring: form a circle; send values around
  - Tree: create a binary (in general, n-ary) tree

Interacting Peers

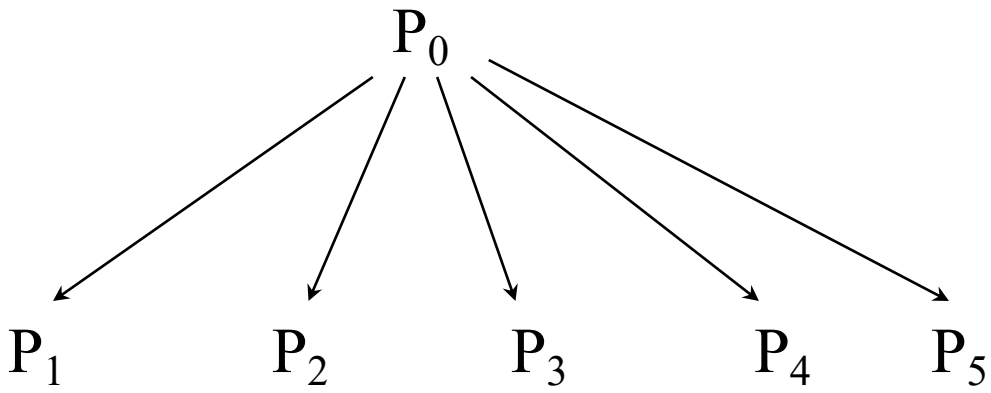
- Each process has an integer
  - Goal: find max and min integers
- Approaches
  - Centralized: use coordinator
    - 2 * (P-1) messages and a bottleneck
  - Symmetric: every process sends value to every other
    - P * (P-1) messages (but easy to program)
  - Ring: form a circle; send values around
    - 2 * (P-1) messages; no bottleneck, but sequential
  - Tree: create a binary (in general, n-ary) tree
    - 2 * (P-1) messages; less bottleneck, but log(P) steps

Interacting Peers

- How to know which implementation to choose?
- Can use analytical models
  - **LogP** model most widely used model
  - L (latency), o (overhead), g (gap), P (number of cores)
    - Allows mostly architecture-independent analysis of parallel algorithms

LogP Model applied to broadcast

(Note: broadcast is half of a barrier.)

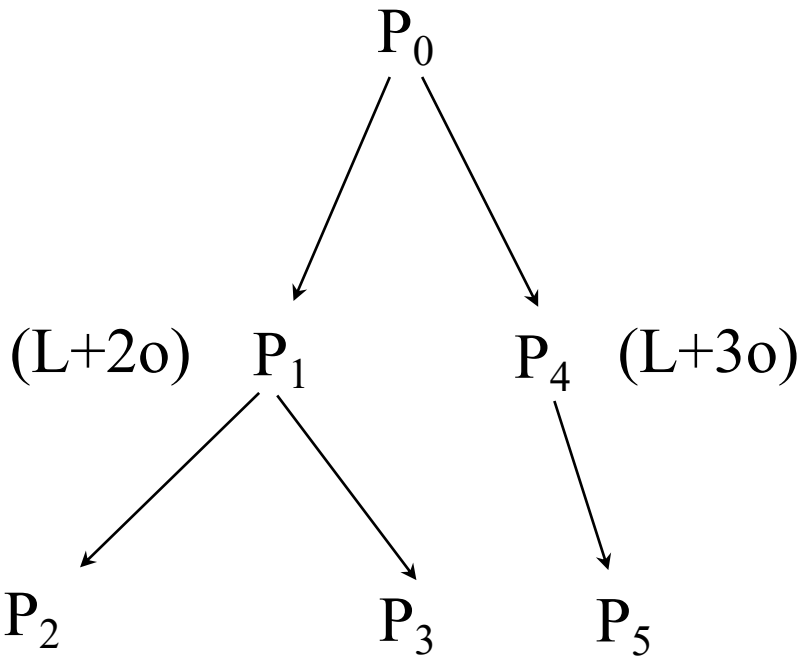


Received at time: L+2o L+3o L+4o L+5o L+6o

Broadcast completion time is L+6o

LogP Model applied to broadcast

(Note: broadcast is half of a barrier.)



Received at time: 2L+4o 2L+5o 2L+5o

Broadcast completion time is 2L+5o (compared to L+6o on previous slide).
Which is better depends on the values of L and o.

Interacting Peers

- Generalizations
  - Example: all-to-all
    - Implementations are not at all clear here
    - Must worry about contention
    - May want intelligent scheduling