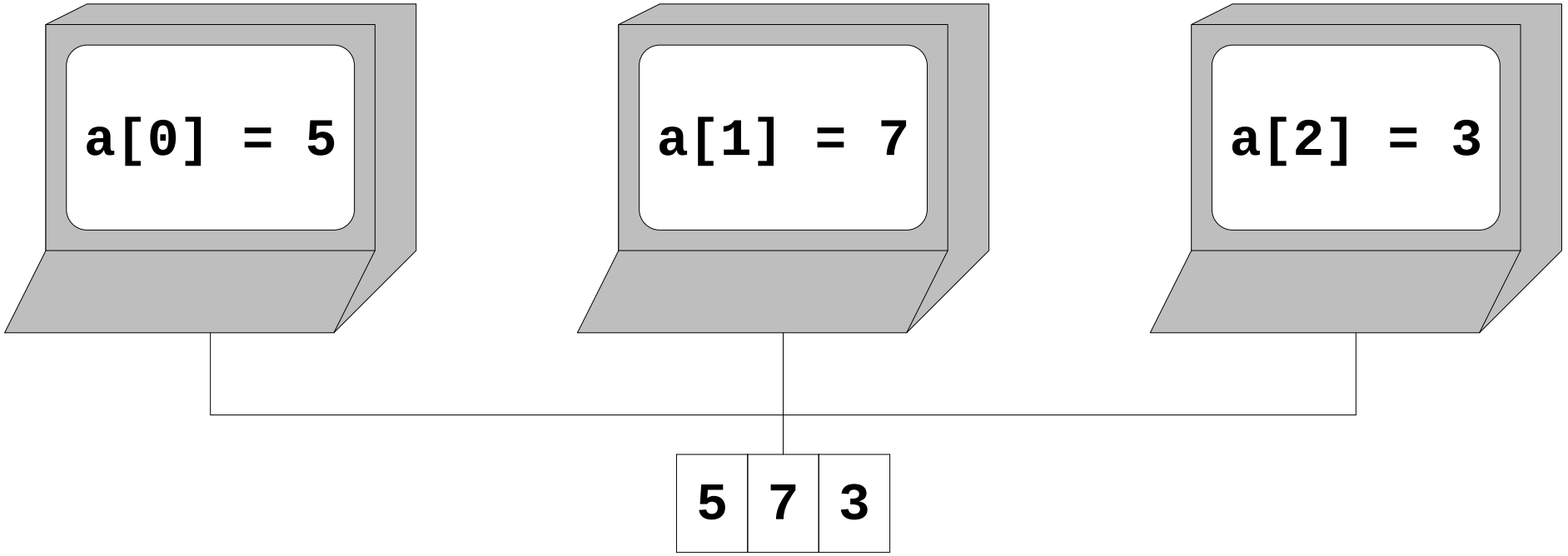
Distributed Shared Memory



see **field.c**

Distributed Shared Memory

Key problems:

- Sharing as transparently as possible
- Sharing as efficiently as possible

Sharing Changes

Most obvious idea: for each memory write or read, consult a central server

- Simple
- *Very* slow
- Difficult to make transparent

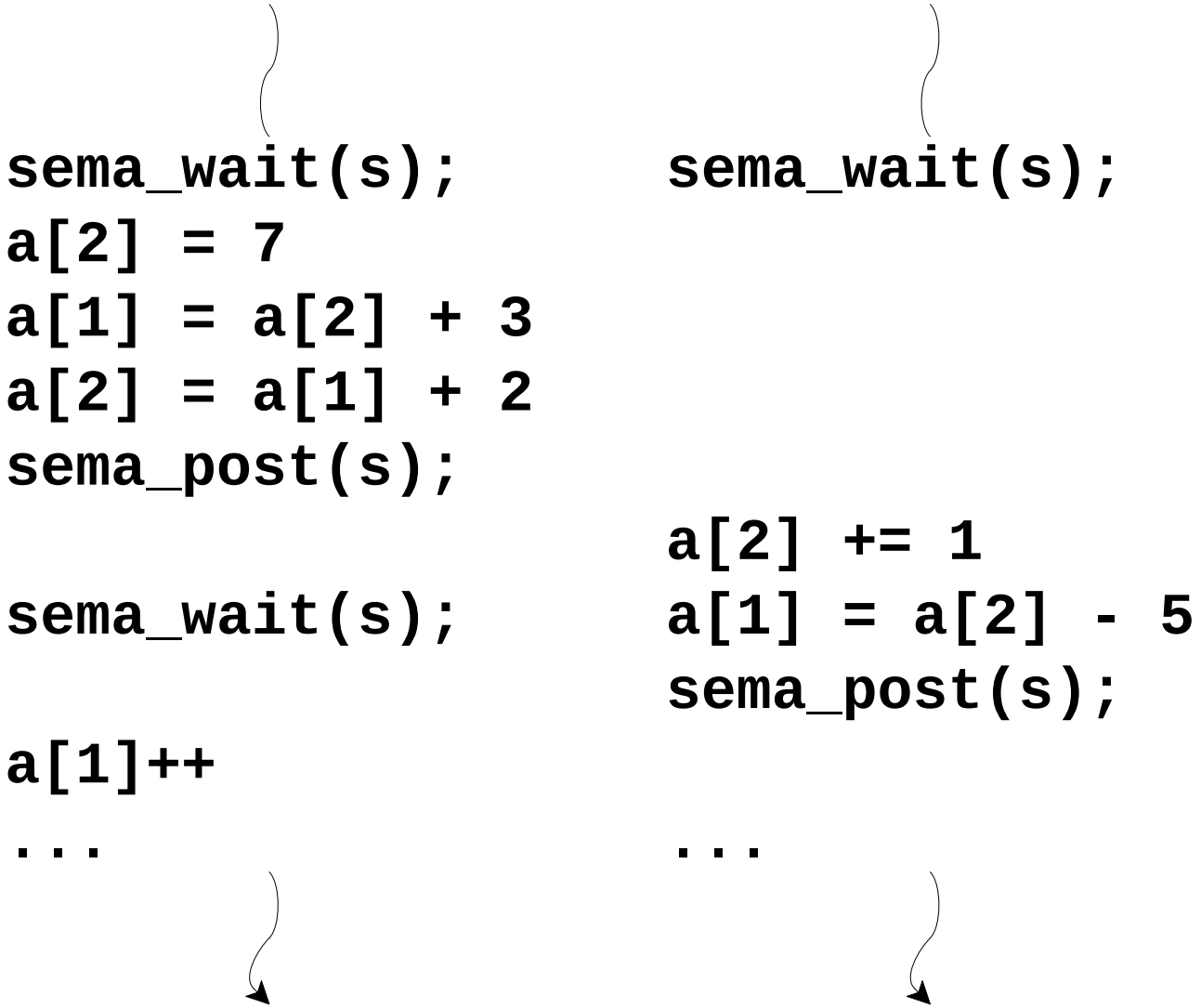
Sharing Changes

Better idea: check with central server at uses of synchonization abstractions

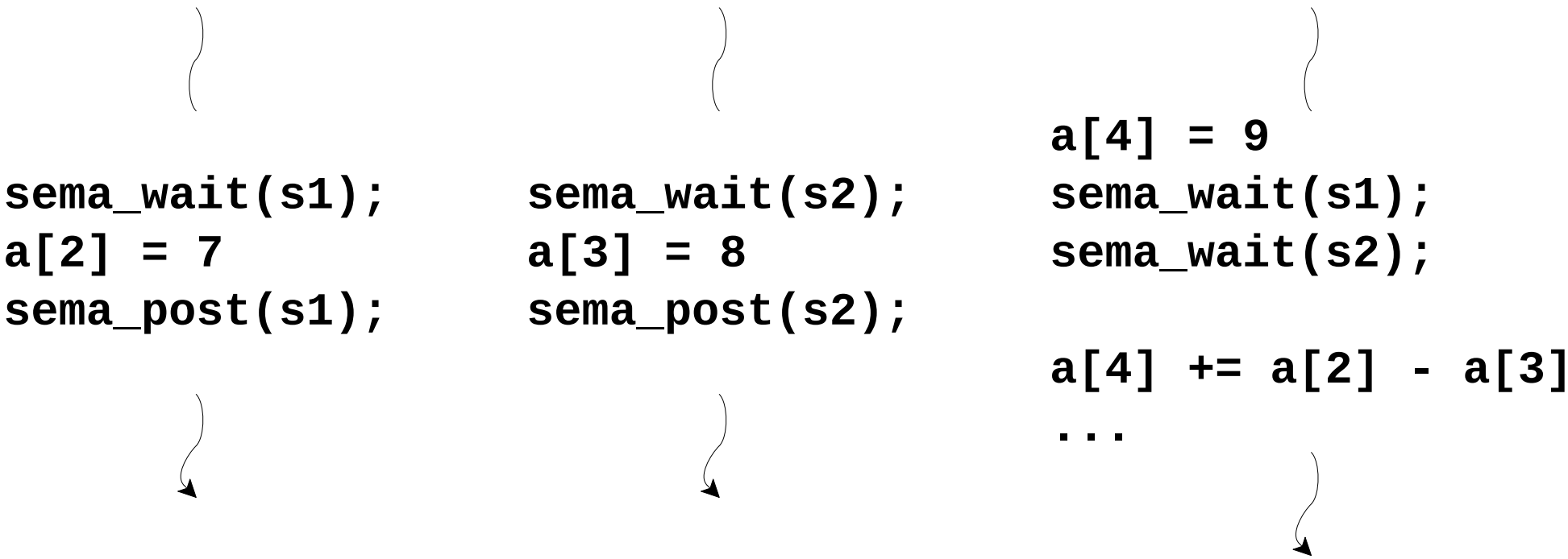
Based on the idea that changing shared data reliably requires synchronization

- Faster
- Still fairly simple
- Easier to make transparent

Sharing at Synchronization



Sharing at Synchronization



$\Rightarrow$ sites must track and communicate changes, instead of whole memory

DSM State and Changes

Server tracks:

- Sempahore state
- Current memory state
- A per-site map of changes that need to be sent

Client tracks:

- State most recently received from server
  - Can be compared to current local state to generate a change map

DSM State Communication

Client grabs a lock:

- Get state changes from the server right after

Client releases a lock:

- Send changes since last communiction right before

Mini-DSM

see **`network_dsm_client.c`**,
`network_dsm_server.c`

Improving DSM

Sending/receiving global state updates is slow and usually unnecessary

Checking all memory to detect changes is slow and usually unnecessary

Alternatives:

- Use pages and page protection to transparently detect changes and accesses

  see **`network_page_dsm_client.c`**

- Put all state in objects, and let the object accessors and mutators commuincate with the server as needed