



Distributed Systems

Winter Term 2025/26

Roland Wismüller
Universität Siegen
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: January 8, 2026



Distributed Systems

Winter Term 2025/26

11 Distributed Shared Memory



Contents

- ➔ Introduction
- ➔ Design alternatives

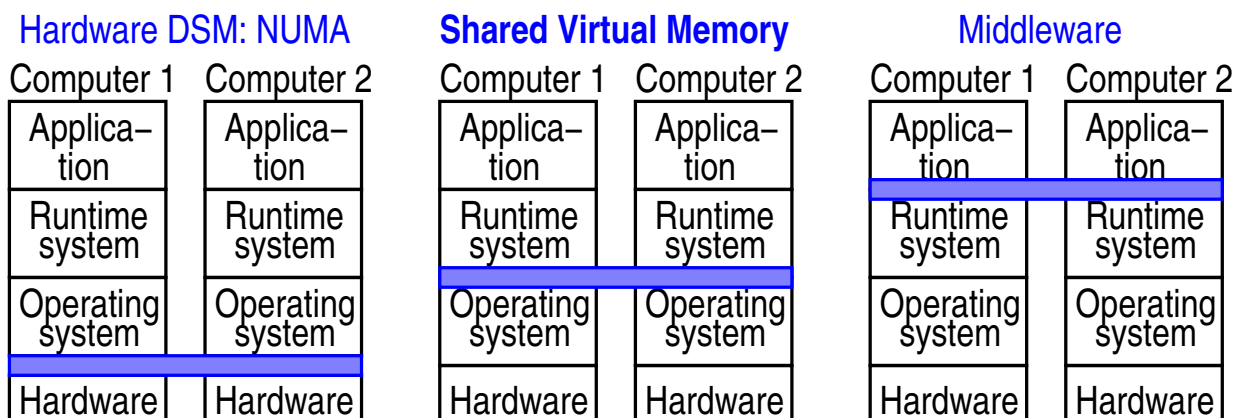
Literature

- ➔ Colouris, Dollimore, Kindberg: Kap. 16.1-16.3

11 Distributed Shared Memory ...



- ➔ Goal: shared memory in distributed systems
- ➔ Basic technique considered here:
 - page-based memory management on the nodes
 - on demand: loading pages over the network
 - if necessary replication of pages to increase performance
- ➔ Differentiation:





Design alternatives

- ➔ Structure of the shared memory:
 - ➔ byte-oriented (distributed shared memory pages)
 - ➔ object-oriented (distributed shared objects)
 - ➔ e.g., Orca
 - ➔ immutable data (distributed shared container)
 - ➔ operations: read, add, remove
 - ➔ e.g., Linda Tuple Space, JavaSpaces
- ➔ Granularity (for page-based methods):
 - ➔ when changing a byte: transmission of entire page
 - ➔ with large pages: more efficient communication, less administrative effort, more false sharing



Design alternatives ...

- ➔ Consistency model: mostly sequential or release consistency
- ➔ Consistency protocol: usually local write protocol
 - ➔ i.e., writable memory page migrated to accessing process
 - ➔ with or without replication for read accesses
 - ➔ client initiated replication, i.e., reader requests copy
 - ➔ usually only one writer per page
 - ➔ mostly invalidation protocols (with push model)
 - ➔ update protocols only if write accesses can be buffered (e.g. with release consistency)

Notes for slide 311:

If write accesses cannot be buffered, we would not only have to send a multicast message for each write access, which would be expensive, but we would also have to be able to detect each individual write access. To do this, we can proceed as follows:

- ➔ The relevant page is write-protected.
- ➔ A write access triggers a page fault; the OS then gains control.
- ➔ In order for the process to execute the access afterwards, the write protection must be disabled (i.e. the page is given write access).
- ➔ However, in order to be able to detect subsequent write accesses, the OS must switch the write protection on again immediately after the access.
- ➔ This requires a trace mode (usually available) in the processor that interrupts the process immediately after executing the next instruction.

However, this procedure is very expensive.

If write accesses can be buffered, only the first write access must be detected. It is not necessary to reactivate write protection using trace mode. In addition, fewer updates have to be sent.

311-1

11 Distributed Shared Memory ...



Design alternatives ...

- ➔ Management of copies
 - ➔ mostly: at any time either multiple readers or one writer
 - ➔ each page has an owner
 - ➔ writer or one of the readers (last writer)
 - ➔ manages a list of processes with copies of the page
 - ➔ before write access: process requests current copy
- ➔ Finding the owner of a page:
 - ➔ central manager
 - ➔ manages owners, forwards requests
 - ➔ distributed manager with fixed distribution
 - ➔ fixed mapping: page → manager

Notes for slide 312 (Management of copies):

Notation:

- ➔ $owner(S)$ = owner of page S (needed by each process)
- ➔ $copyset(S)$ = set of nodes that have copies of S (needed only by $owner(S)$)

When a read request is made to a page S by a process P , the following happens if P does not have a copy of S :

- ➔ the MMU generates a page fault
- ➔ the OS requests a read copy of S from $owner(S)$
- ➔ if the page S is writable at $owner(S)$: remove write permissions
- ➔ $owner(S)$ sends S to P 's node
- ➔ $copyset(S) := copyset(S) \cup \{P\}$
- ➔ if the page S arrives at P 's node, the OS sets the page to non-writable and lets P repeat the aborted access

312-1

When a process P requests to write to a page S , the following happens if P does not have a writable copy of S :

- ➔ the MMU generates an exception (page fault or protection violation)
- ➔ the OS is requesting a writable copy of S from $owner(S)$
- ➔ $owner(S)$ then invalidates all copies of the page stored on nodes in $copyset(S)$ and sends S to P 's node
- ➔ $owner(S) := P, copyset(S) := \{P\}$
- ➔ if the page S arrives at P 's node, the OS sets the page to writable and lets P repeat the aborted access

312-2

Design alternatives ...

- ➔ Finding the owner of a page ...:
 - ➔ multicast instead of manager
 - ➔ problem: concurrent requests
 - ➔ solution: totally ordered multicast, vector time stamps
 - ➔ dynamically distributed manager
 - ➔ every process knows a likely owner
 - ➔ this node forwards the request if necessary
 - ➔ the likely owner is updated,
 - ➔ when a process transfers the ownership property
 - ➔ upon receipt of an invalidation message
 - ➔ upon receipt of a requested read-only page
 - ➔ when a request is forwarded (to the requestor)

Notes for slide 313 (concurrent requests):

The following situation may occur, for example:

- ➔ node N1 and N2 concurrently request the same page, the owner is O.
 - ➔ the request from N1 goes to N2 and O (among others), the request from N2 goes to N1 and O (among others).
- ➔ O is sending the page to N1.
- ➔ So N1 would have to process N2's request, but not before he actually has the page. I.e., N1 would have to buffer the request.
- ➔ On the other hand, N2 should ignore the request from N1, since it was already answered by O.
- ➔ But for N1 and N2, the situation is completely the same when the requests arrive, so how are they supposed to decide what to do?

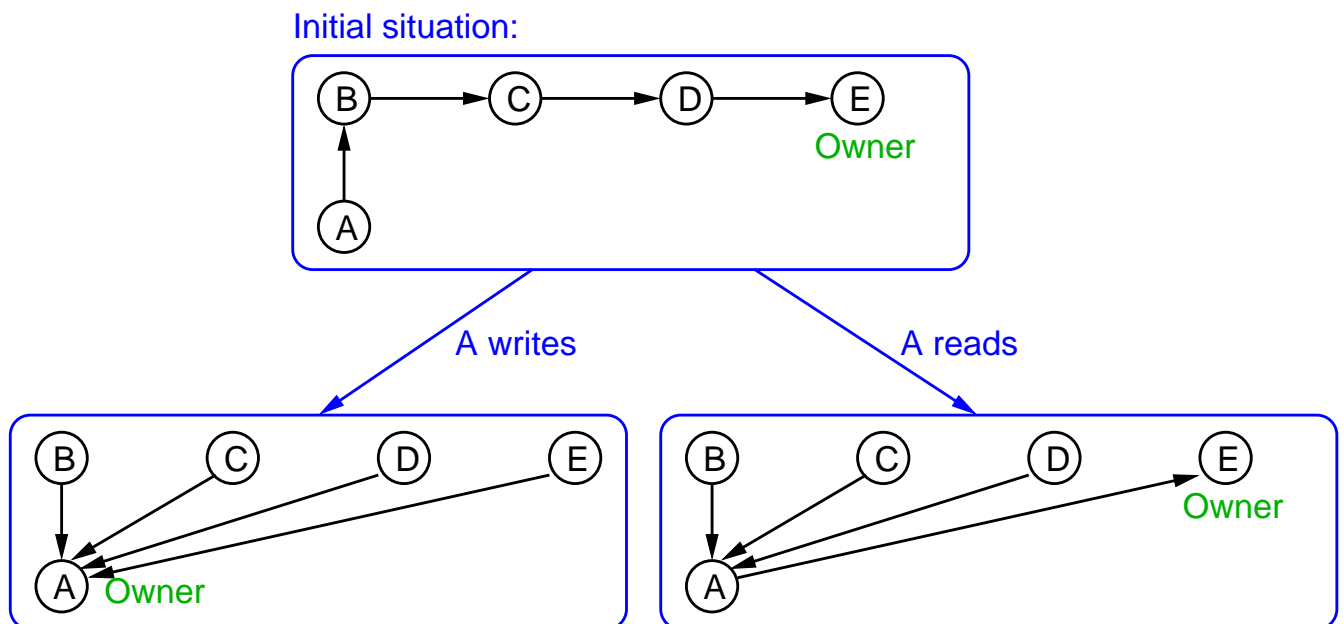
Notes for slide 313 (dynamically distributed manager):

Rationales for updating the probable owner:

- when a process A transfers the ownership to process B:
then B is the new owner of the page; A updates its reference.
- if process A receives an invalidation message from process B:
then B must be the owner; A updates its reference.
- when process A gets a requested read-only page from process B:
then B must be the owner; A updates its reference.
- when process A forwards a request initiated by process B for a page it does not own:
then A updates its reference to process B, since it is likely (if it is a write request, even certain) that process B will soon become the owner.

313-1

Example of updating the probable owner:



314-1

Design alternatives ...

- ➔ Problems: e.g., thrashing, especially due to false sharing
 - ➔ simple remedy:
 - ➔ a page can be migrated again only after a certain period of time
 - ➔ TreadMarks: multiple writer protocol
 - ➔ release consistency; when released, only the changed parts of the page are transferred
 - ➔ changes are then “merged”
 - ➔ in case of conflicts: result is non-deterministic