COS3043
System Fundamentals

Lecture 9

Topics

1.	**Abstractions** **1.1 Hardware Resources** **1.2 OS Functionality** **1.3 Managing the CPU and Memory**
2.	**OS Structure** **2.1 SPIN Approach** **2.2 Exokernel Approach** **2.3 L3/L4 Micro-Kernel Approach**
3.	**Virtualization** **3.1 Intro to Virtualization** **3.2 Memory Virtualization** **3.3 CPU and Device Virtualization**
4.	**Parallelism** **4.1 Shared Memory Machines** **4.2 Synchronization** **4.3 Communication** **4.4 Scheduling**
5.	**Distributed Systems** **5.1 Definitions** **5.2 Lamport Clocks** **5.3 Latency Limit**

6.	**Distributed Object Technology** **6.1 Spring Operating System** **6.2 Java RMI** **6.3 Enterprise Java Beans**
7.	**Design and Implementation of Distributed Services** **7.1 Global Memory System** **7.2 Distributed Shared Memory** **7.3 Distributed File System**
8.	**System Recovery** **8.1 Lightweight Recoverable Virtual Memory** **8.2 Rio Vista** **8.3 Quicksilver**
9.	**Internet Scale Computing** **9.1 GiantScale Services** **9.2 Content Delivery Networks** **9.3 MapReduce**
10.	**Real-Time and Multimedia** **10.1 Persistent Temporal Streams**

List of Discussion

- Lightweight Recoverable Virtual Memory (LRVM)
- Rio Vista
- Quicksilver

Introduction

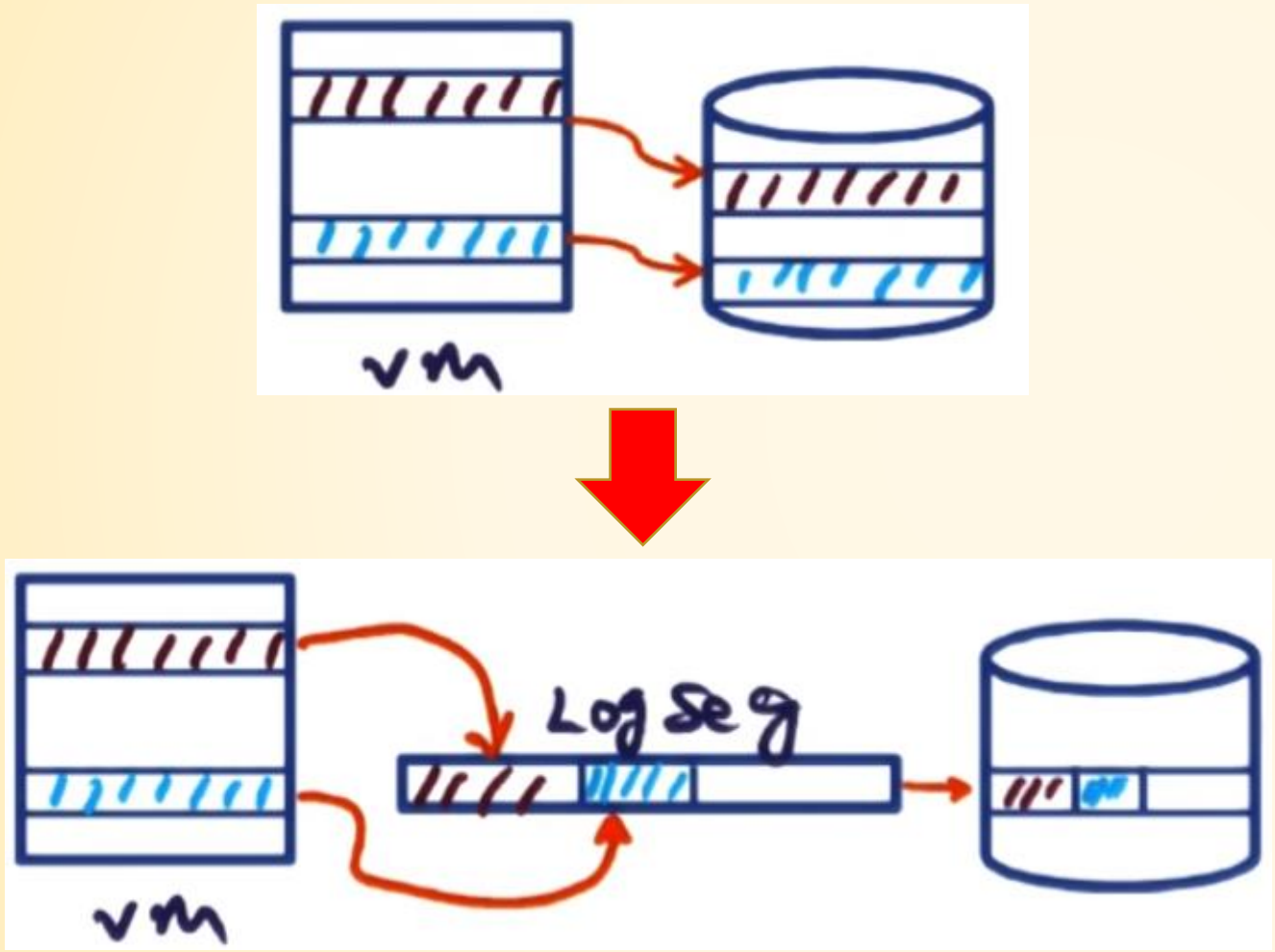
- A good system is a system that can survive crashes.
- This module is to discuss technologies/approaches to deal with system failures and recover from crashes.
- LRVM - Persistent virtual memory layer in support of system services.
- Rio Vista - Performance-conscious design of persistent memory
- Quicksilver - Making recovery a first class citizen in OS design (which is a more radical approach)

Lightweight Recoverable Virtual Memory (LRVM)

Persistence

- Why persistence?
  - Needed by OS sub-systems.
  - Example: In file system, the metadata of file.
- How?
  - Make virtual memory persistent.
- Who will use it?
  - Sub-systems designers, only if this abstraction is of **good performance** (cheap, simple to use, flexible & efficient).
- How to make it efficient?
  - Use **persistent logs** to record the changes to virtual memory.

Persistent Logs



Originally:

- Manipulation of persistent data structures in VM need to be committed into disk storage.
- That causes 2 issues:
  - Increase of I/O operations as the data are spread all over the space.
  - The writing on disk is also on different portion => increase latency.

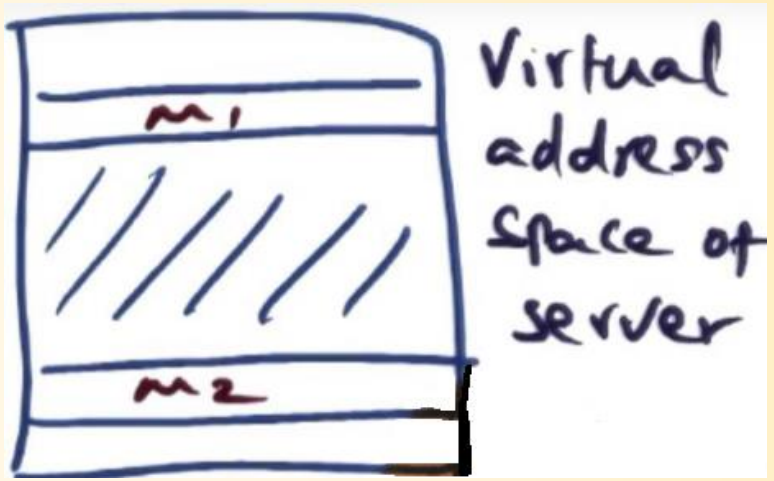
Log Segment:

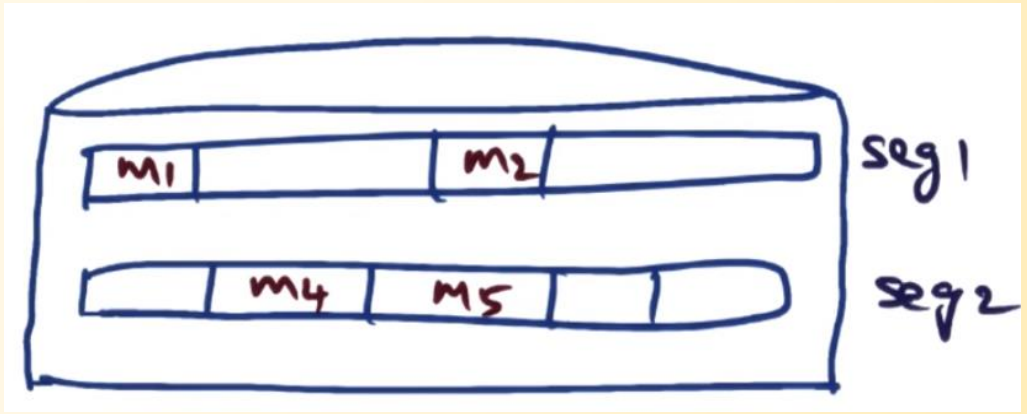
- Recording the changes associated with the manipulation of persistent data structures in VM.
- The storage on disk is stored contiguously.
- In short, convert the random writes (original operation) to sequential writes.

Server Design

- Persistent metadata M1, M2…, Mn.
- Normal data structures + code.

- Capability to create external data segments (disks) to back persistent data structures.
- Applications manage their persistence needs.
- Application designers' choice to use single or multiple data segments.





Recoverable Virtual Memory (RMV) Primitives

Initialization

- initialize(options)
- map(region, options)
- unmap(region)

Body of server code

- begin_xact(tid, restore_mode)
- set_range(tid, addr, size)
- end_xact(tid, commit_mode)
- abort_xact(tid)

Gc to reduce log space

- flush()
- truncate()

} done by LRVM automatically

⇓

Provided for app flexibility

Miscellaneous

- query_options(region)
- set_options(options)
- create_log(options, mode)

How RMV Primitives is used in Server

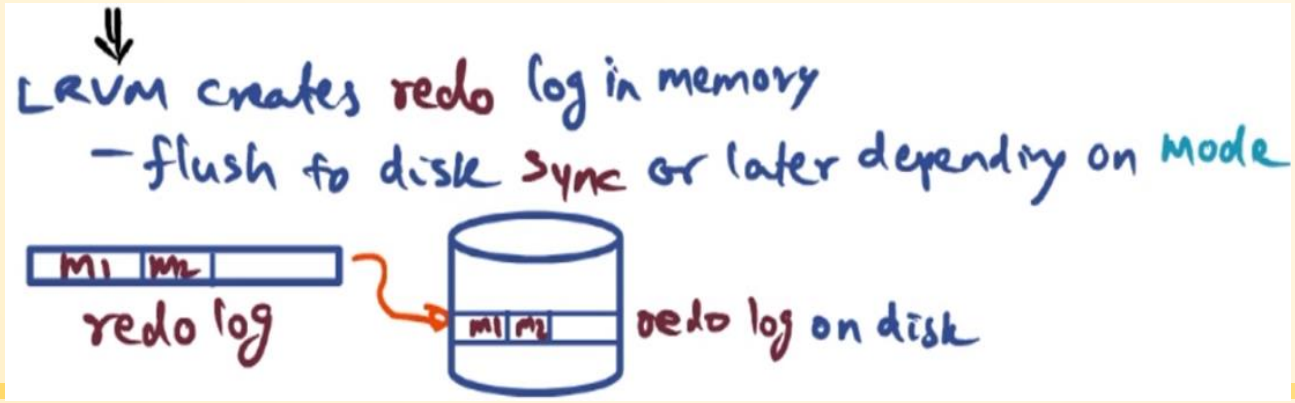
```
Initialize address space from Ext segs
        ⋮
begin_xact(tid, mode);
    set range (tid, base_addr, #bytes);
        ⋮
    write metadata m1
        ⋮
    write metadata m2
end_xact (tid, mode);
```

LRVM creates undo record

; // contained in range

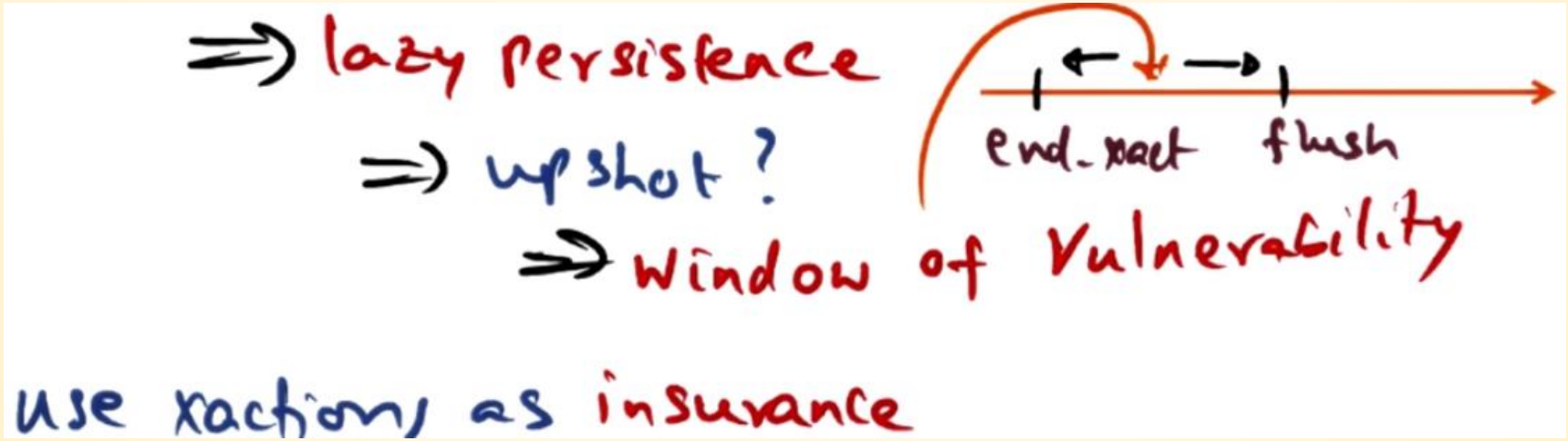
} // contained in range

// or this can be abort

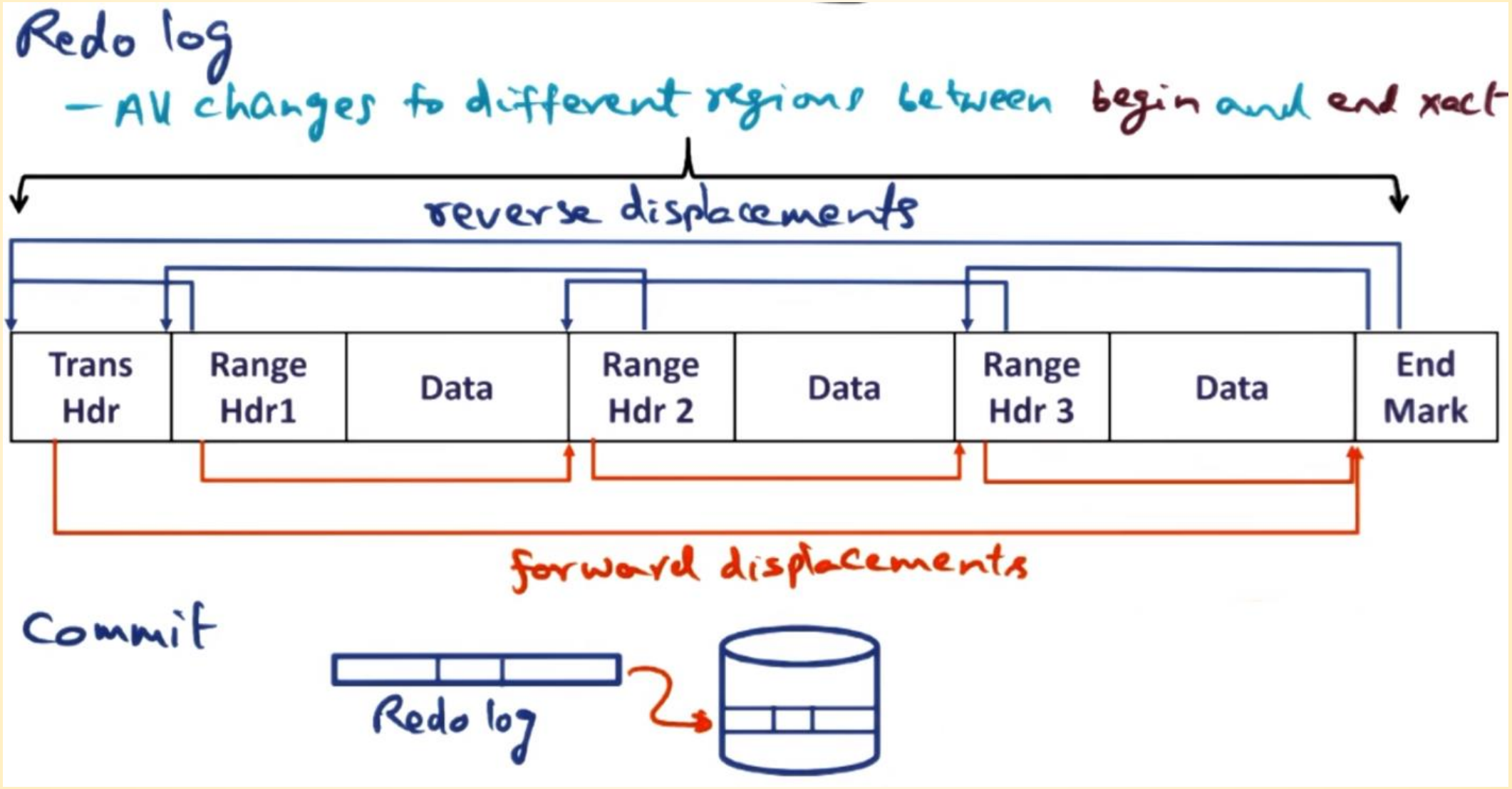


Transactions Optimizations

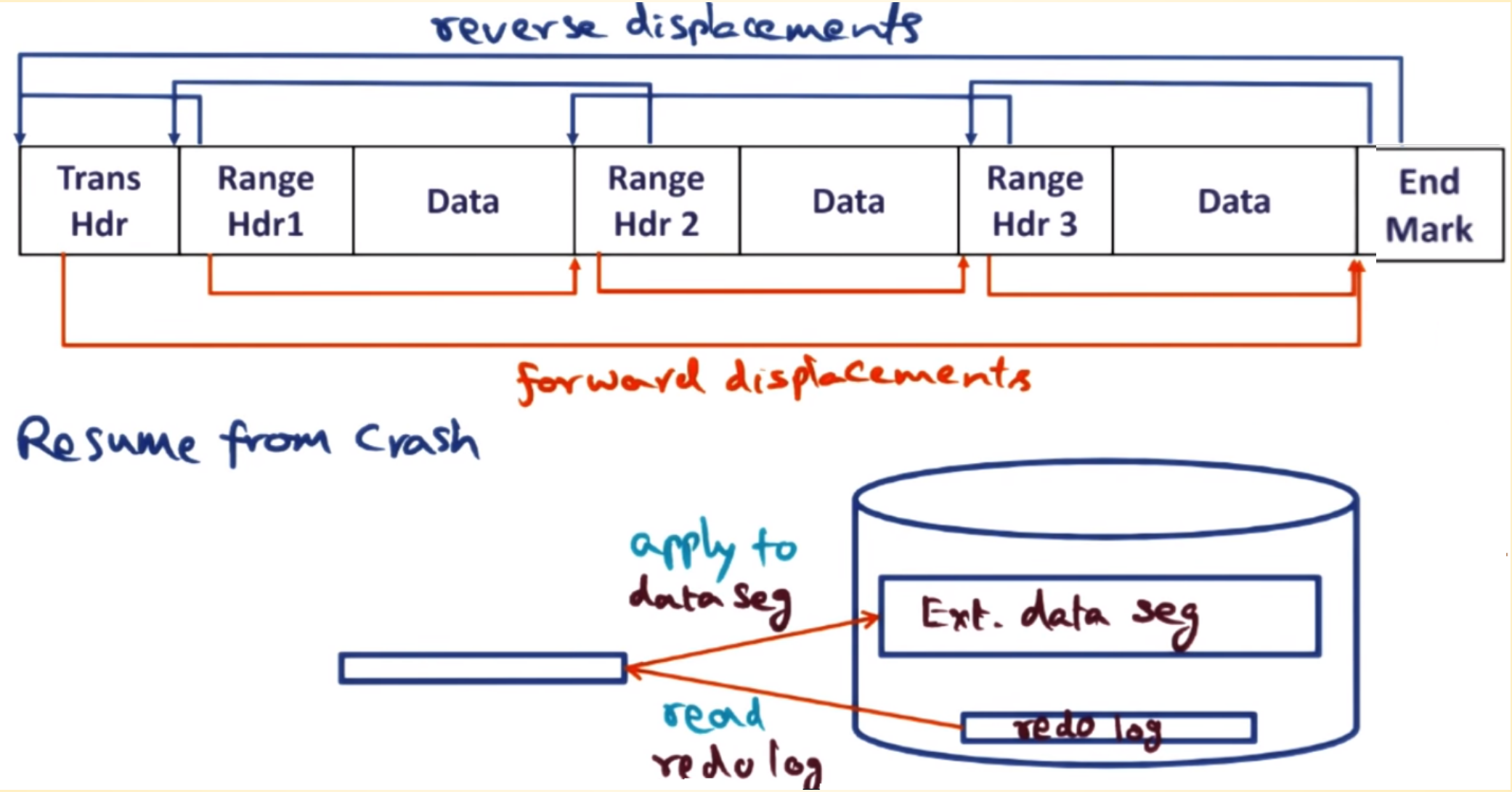
- No-restore mode in begin_xact()
  - ➢No need to create in-memory undo records
- No-flush mode in end_xact()
  - ➢No need to do synchronously flush redo logs to disk



Implementation



Crash Recovery



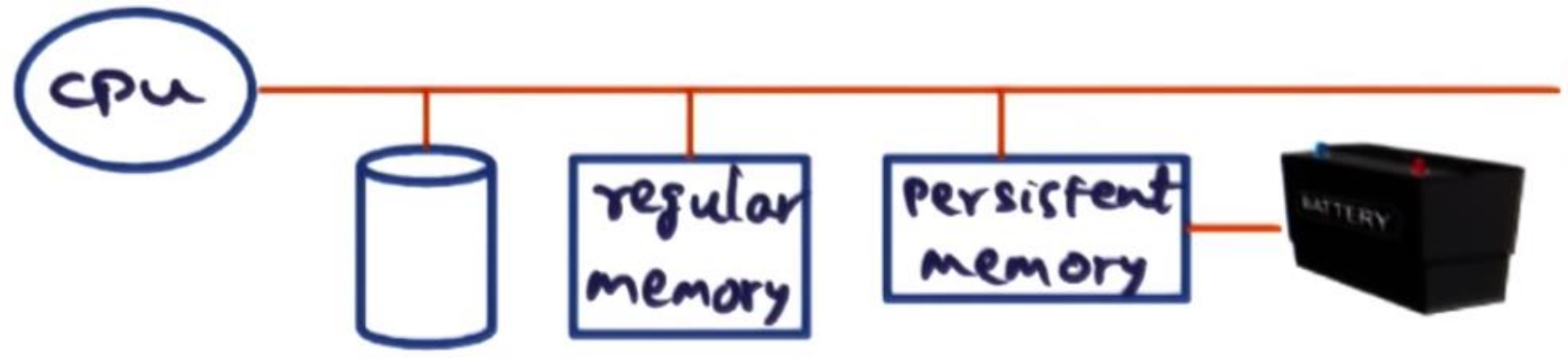
Rio Vista

System Crash

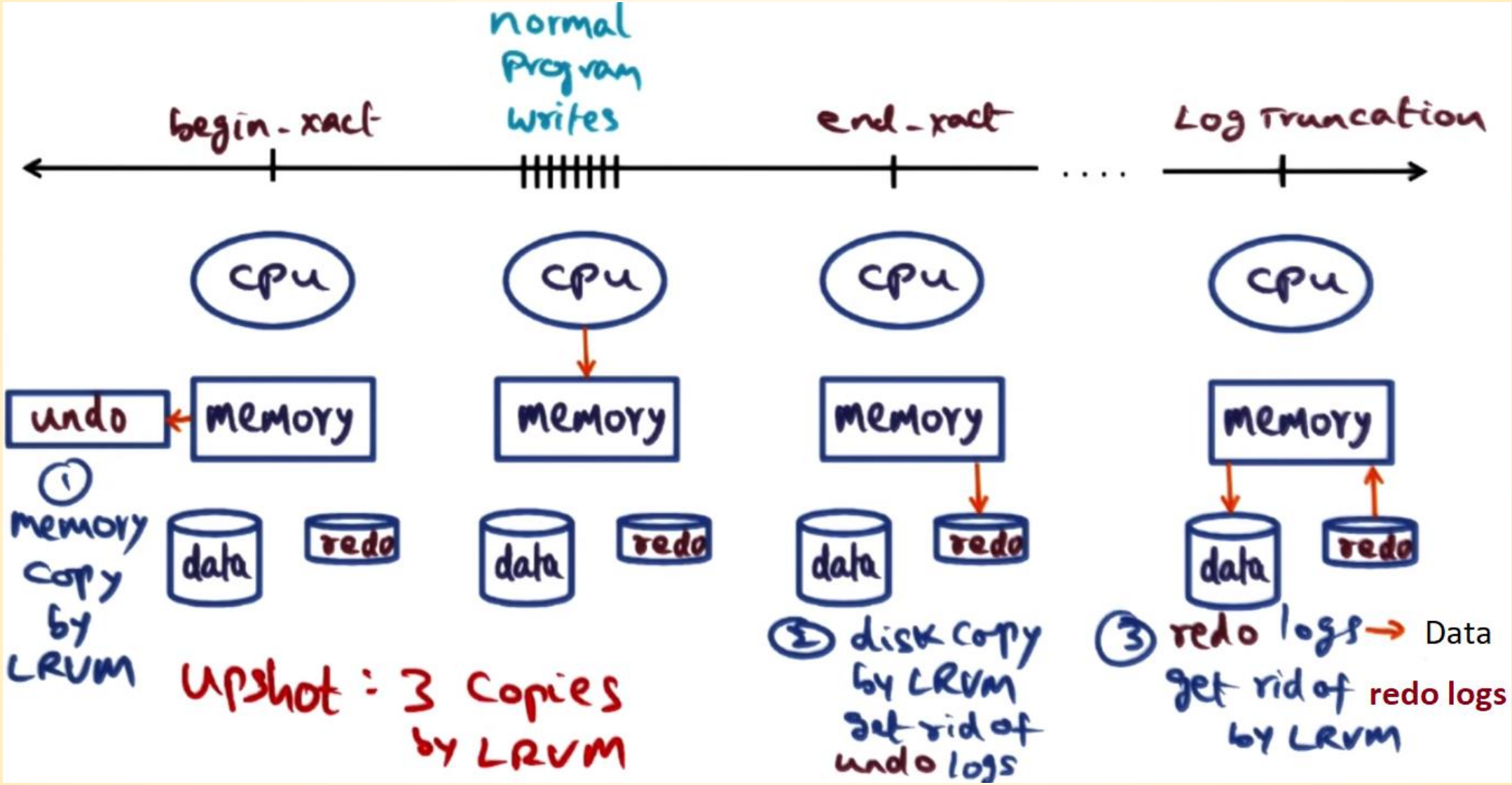
- Two problems concerning failure:
  - Power Failure

    Can we throw some hardware at problem and make it disappear? (ex: use UPS power supply)
  - Software Crash

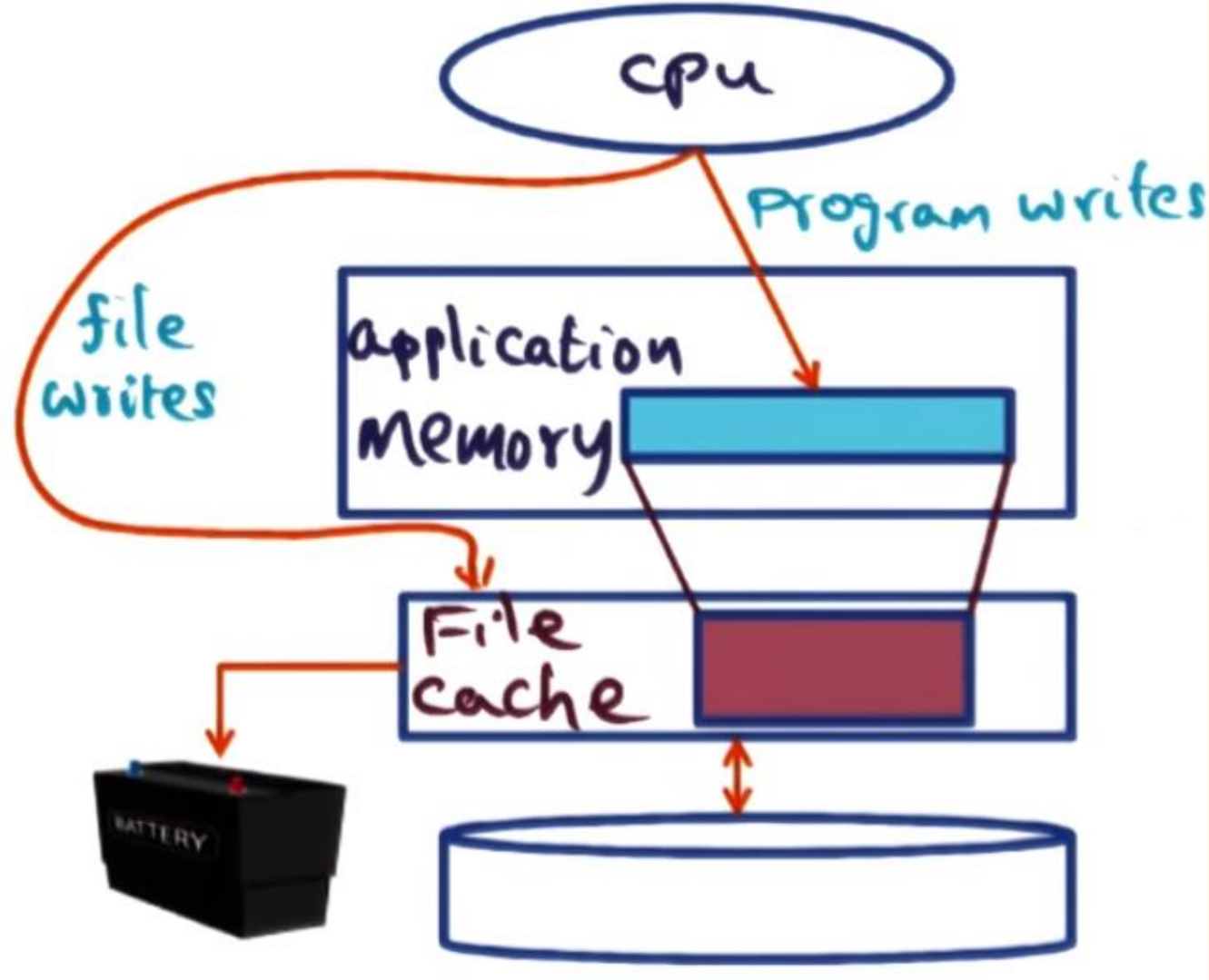
    Reserve a portion of main memory that survives the crashes



LRVM Revisited



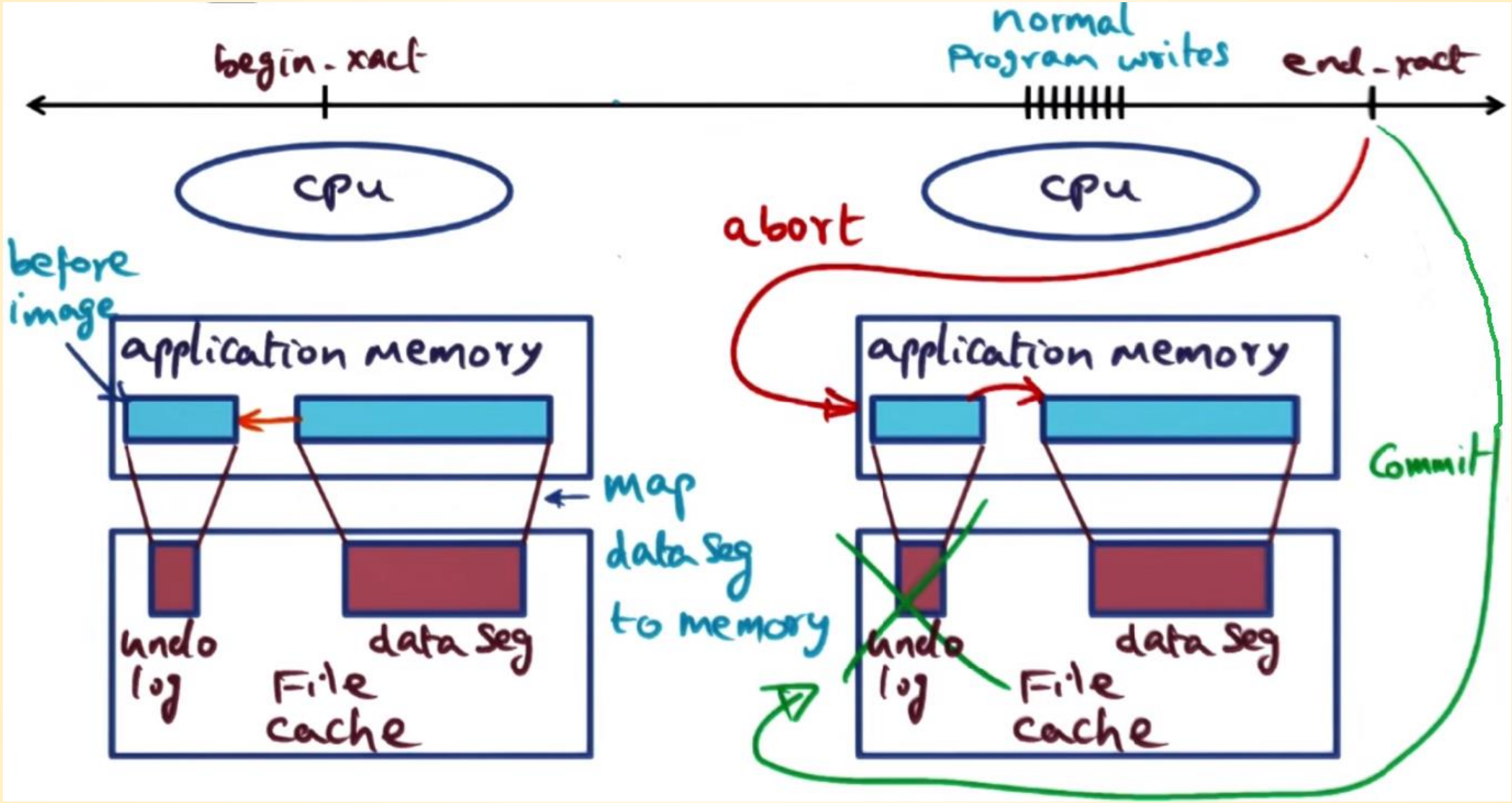
Rio File Cache



Two types of write:

- File writes to file cache.
- Program writes to memory which is mapped to cache using MMAP => normal application memory becomes persistent.

Vista - RVM library on top of Rio File Cache



Crash Recovery

- Treat it like abort:
  - Recover old image from undo log => survives crashes since it is on Rio file cache.
- Crash during crash recovery?
  - Idempotency of recovery => no problem.

Rio Vista Simplicity

- 700+ lines of code in Vista:
  - 10k lines in LRVM.
- Why?
  - No redo logs or truncation code.
  - Checkpointing and recovery code simplified.
- Outcome
  - Simple like LRVM but with much improved performance.

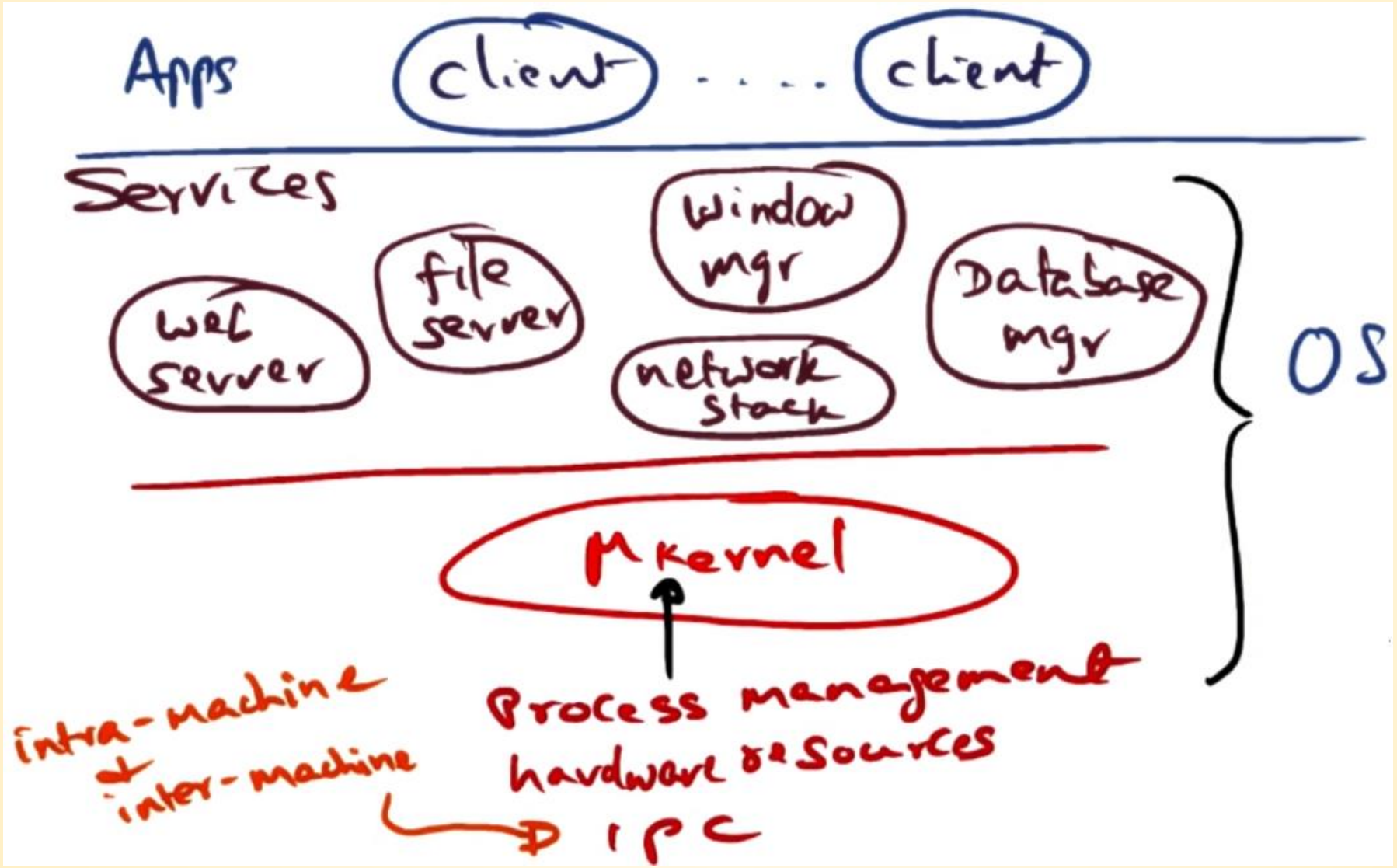
Quicksilver

Introduction

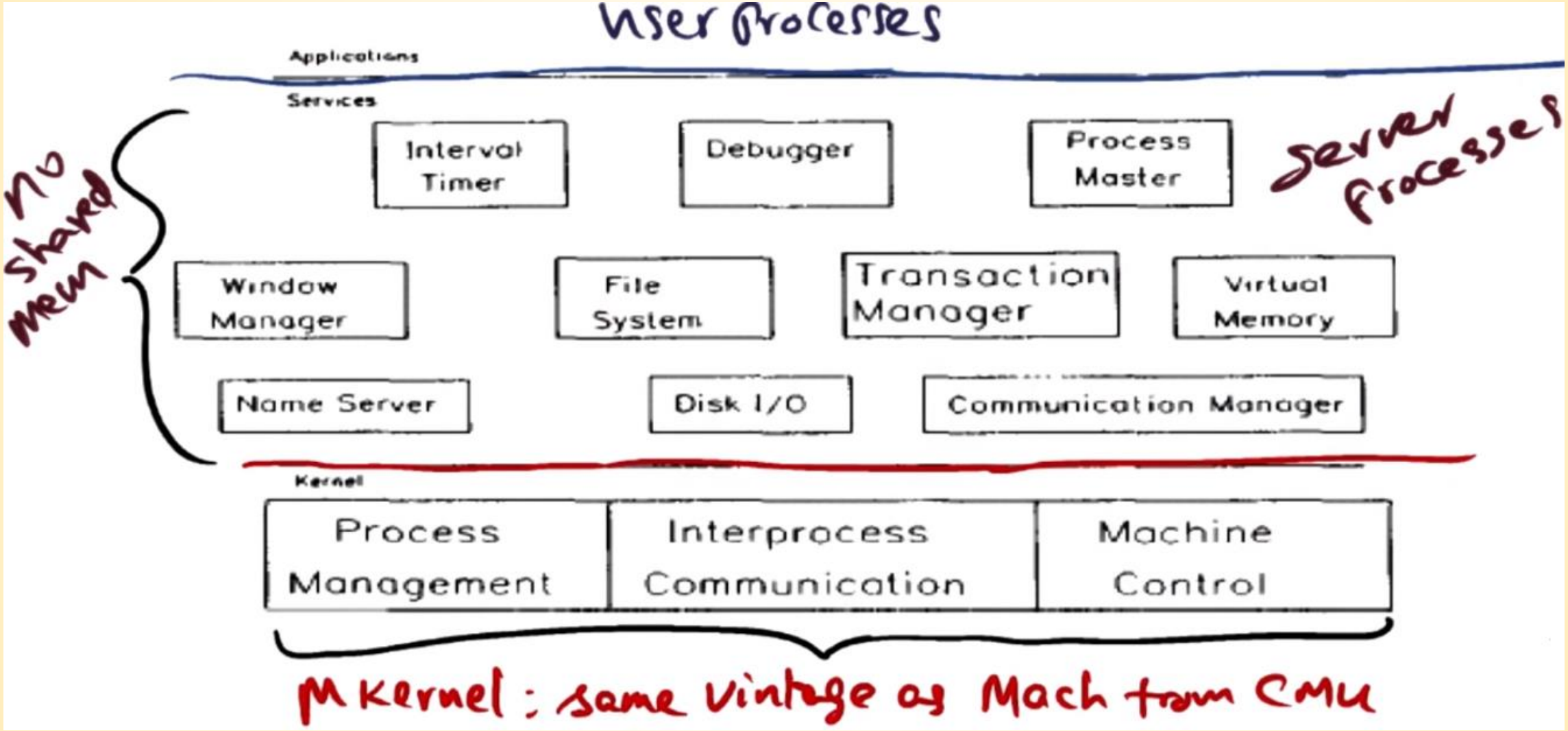
"Making recovery a first class citizen in OS design."

- Conventionally, performance and reliability can't really co-exist at the same time.
- But for Quicksilver, its major notion is if a system is robust on recovery, while without scarifying the performance, then perhaps that's the way to go.

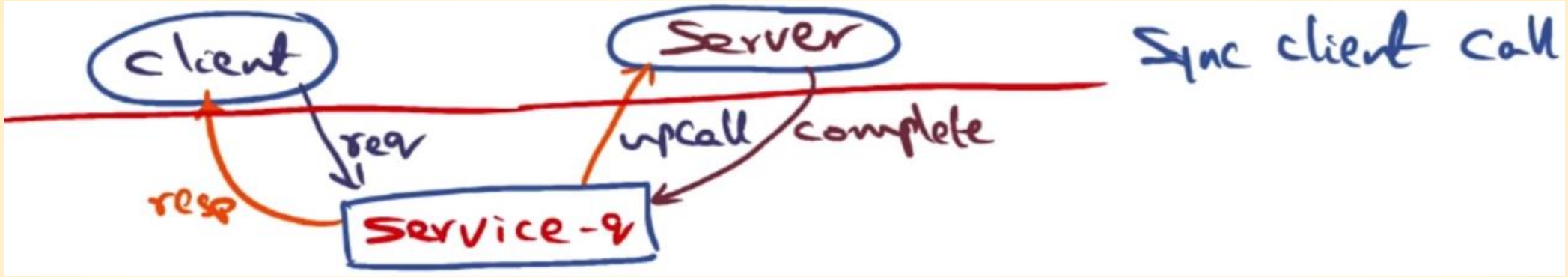
Distributed System Structure

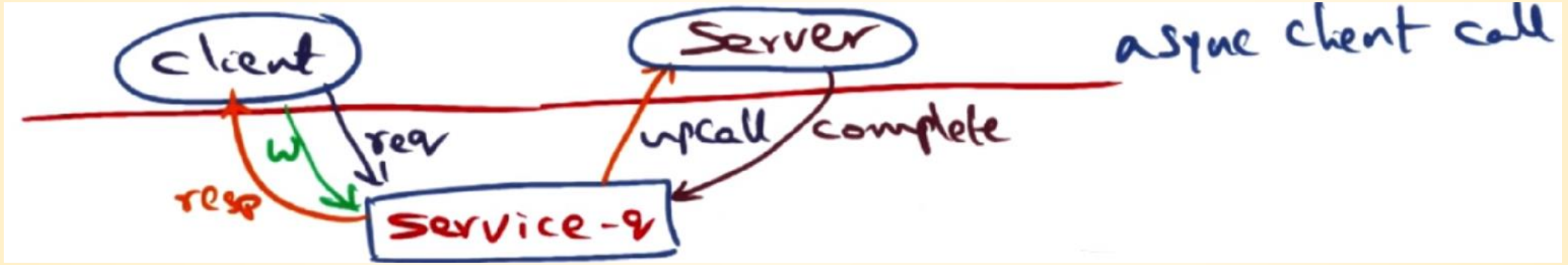


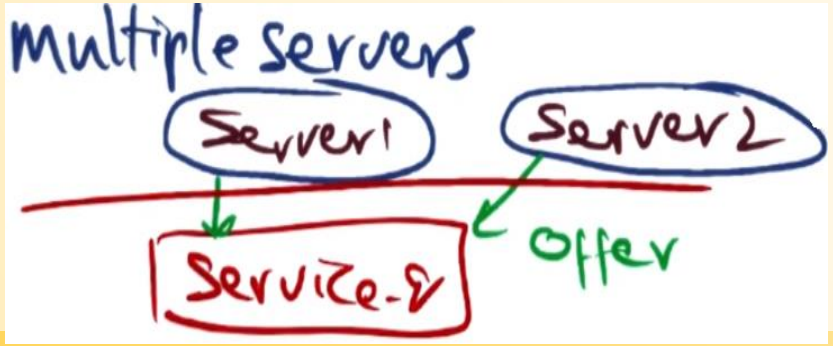
Quicksilver Architecture

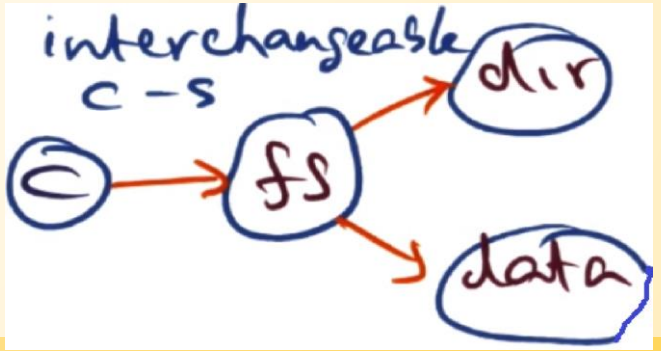


IPC: Fundamental to System Services

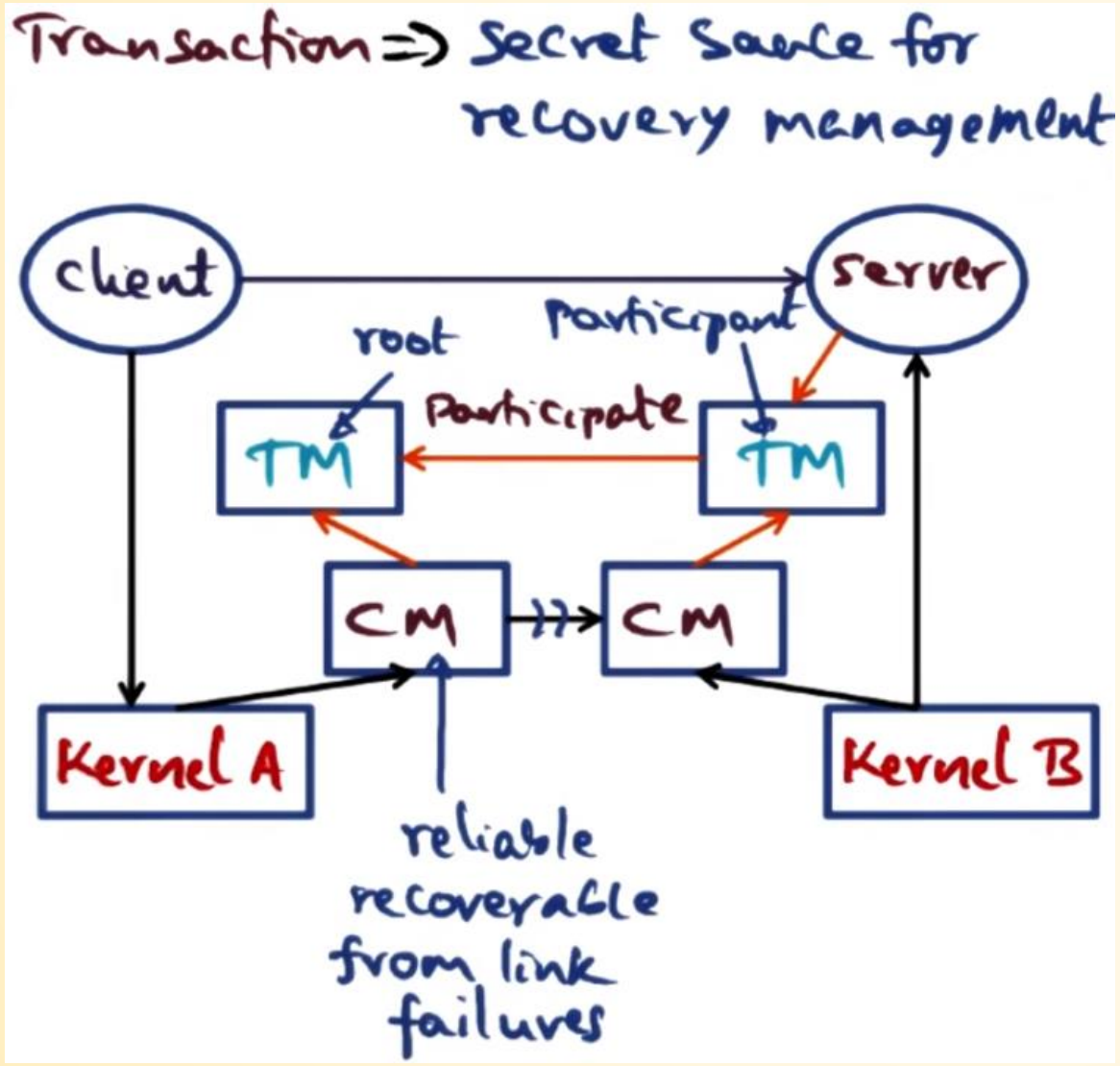


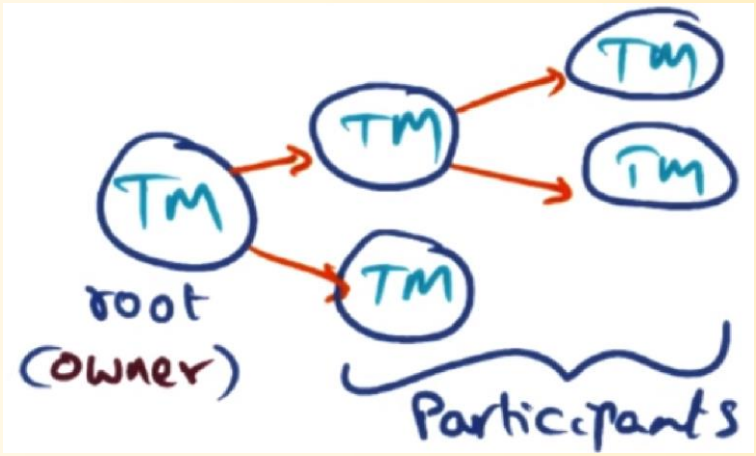


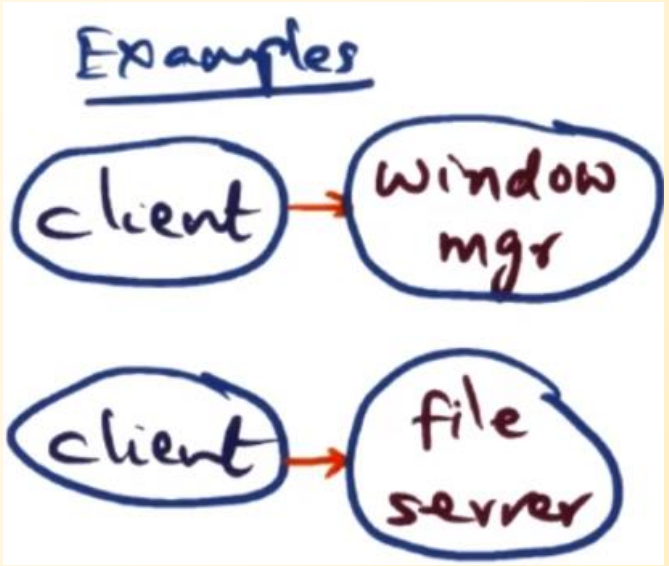




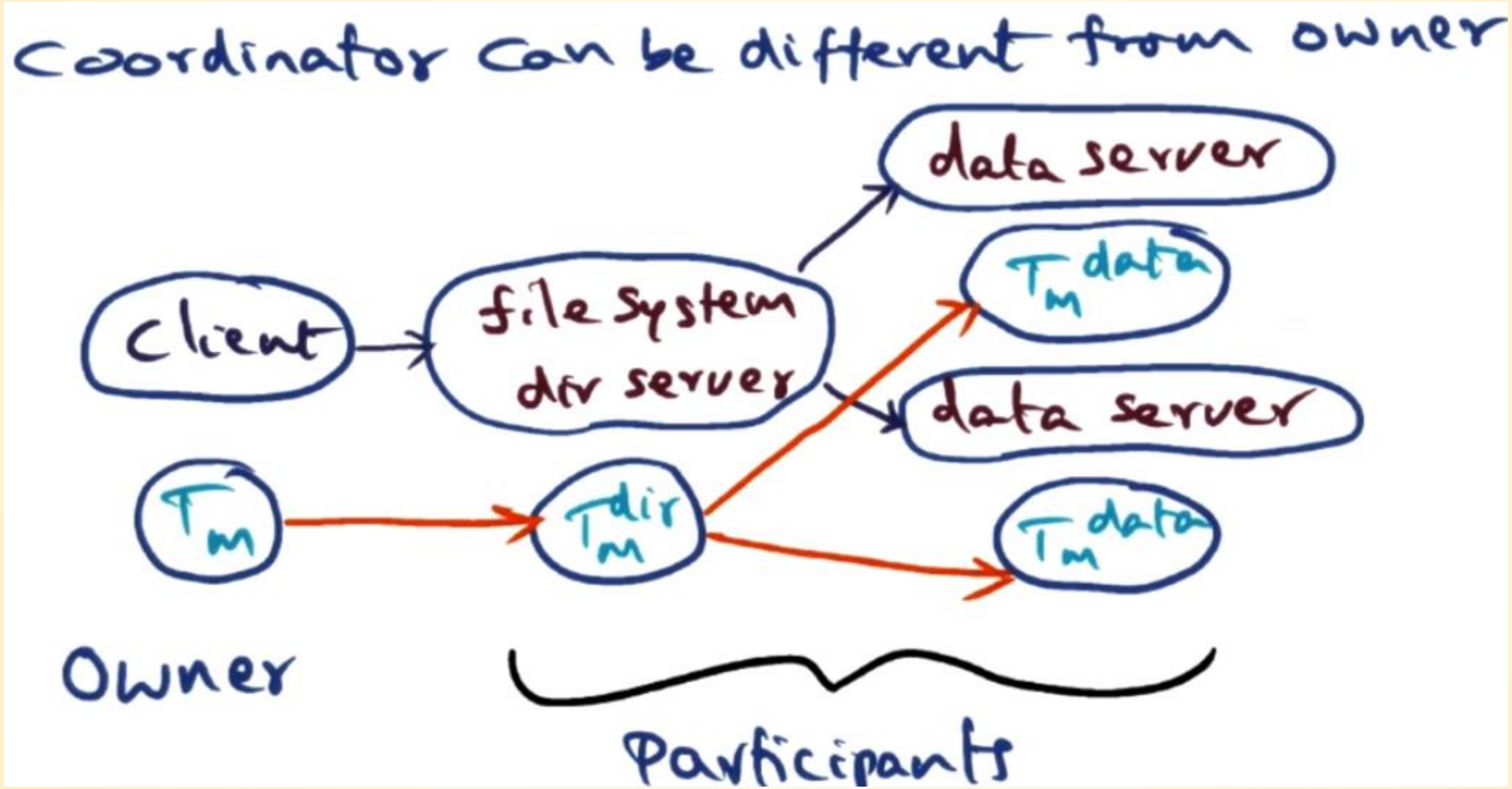
Bundling Distributed IPC + Transactions



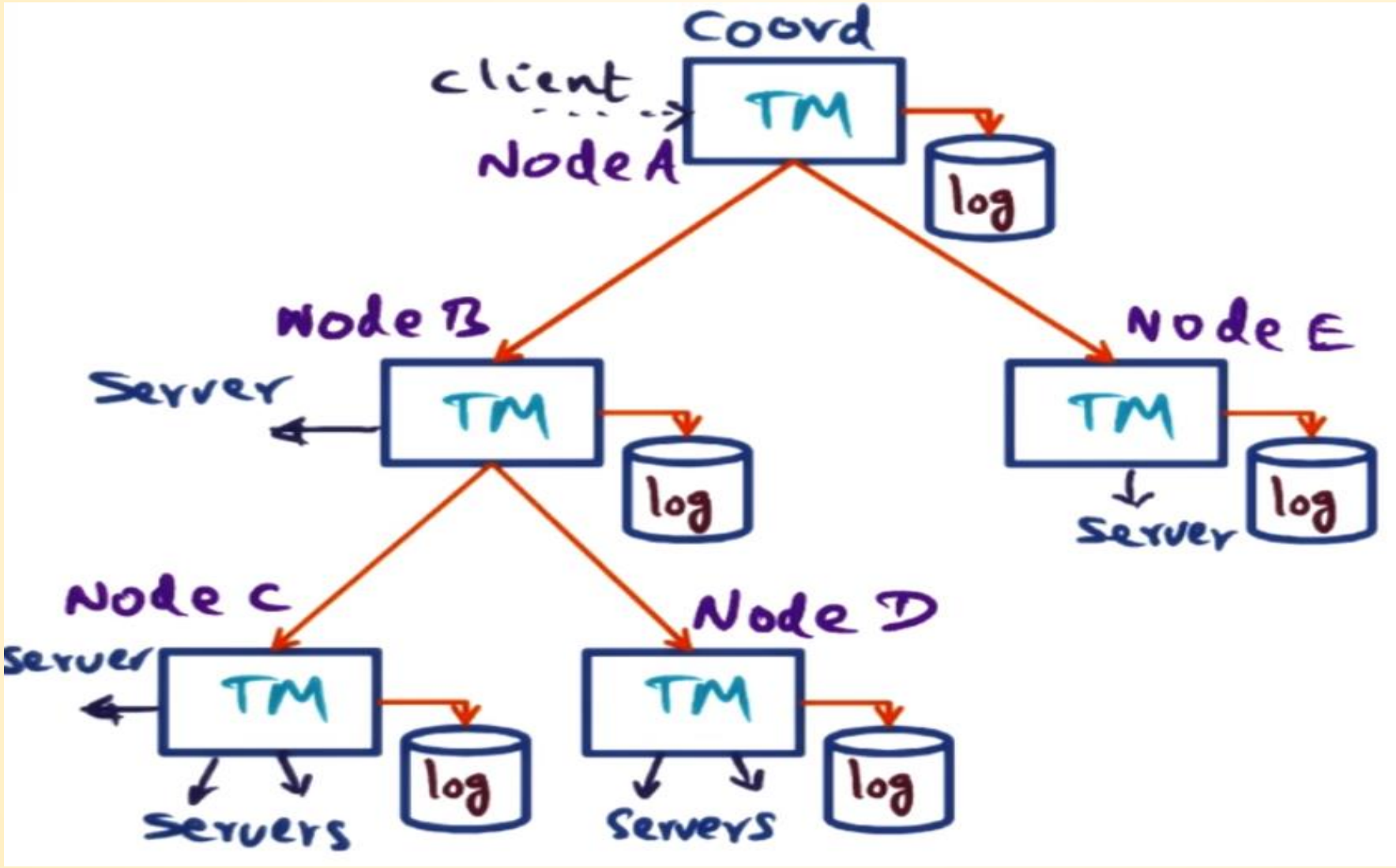




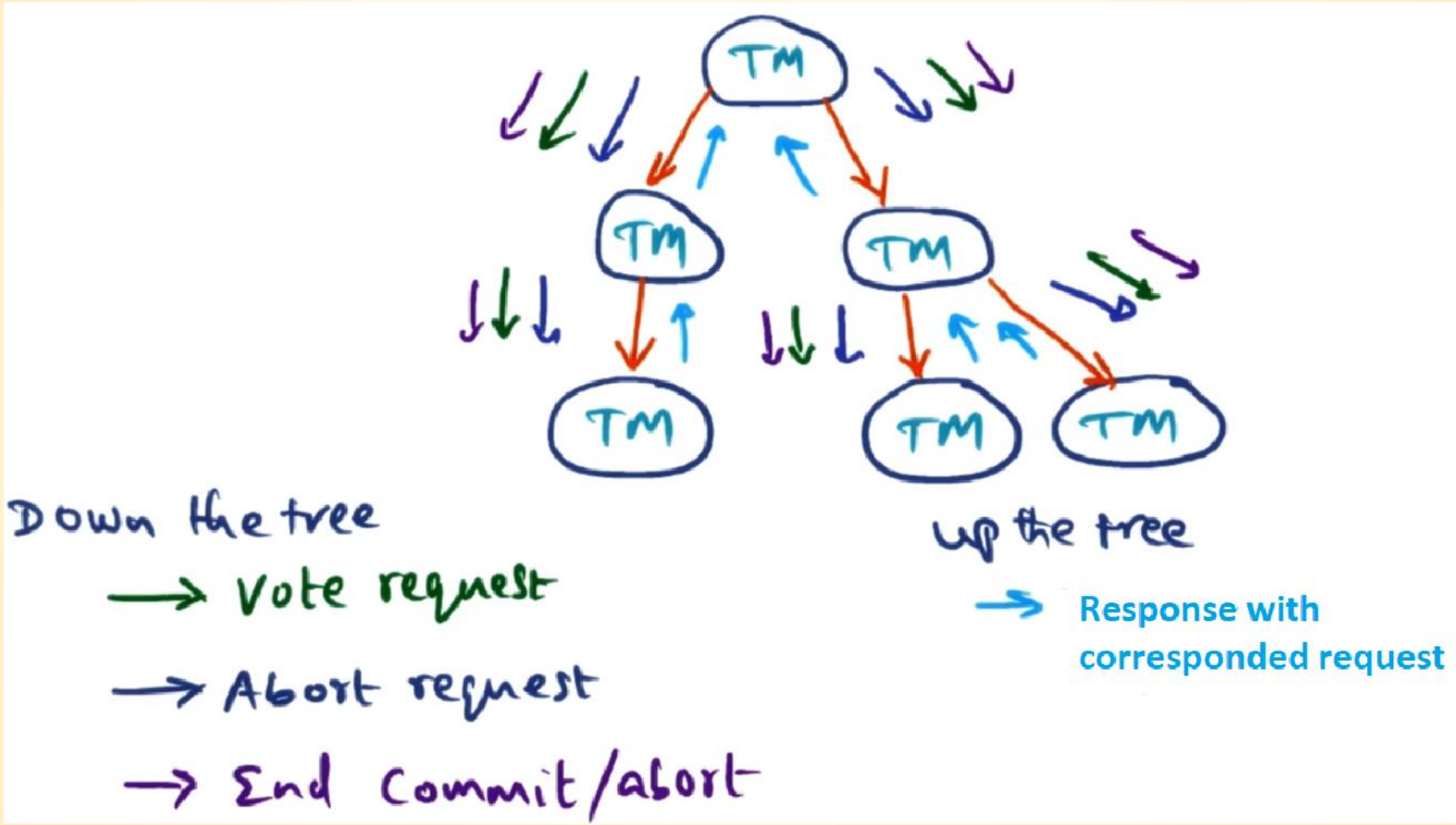
Transactions Management



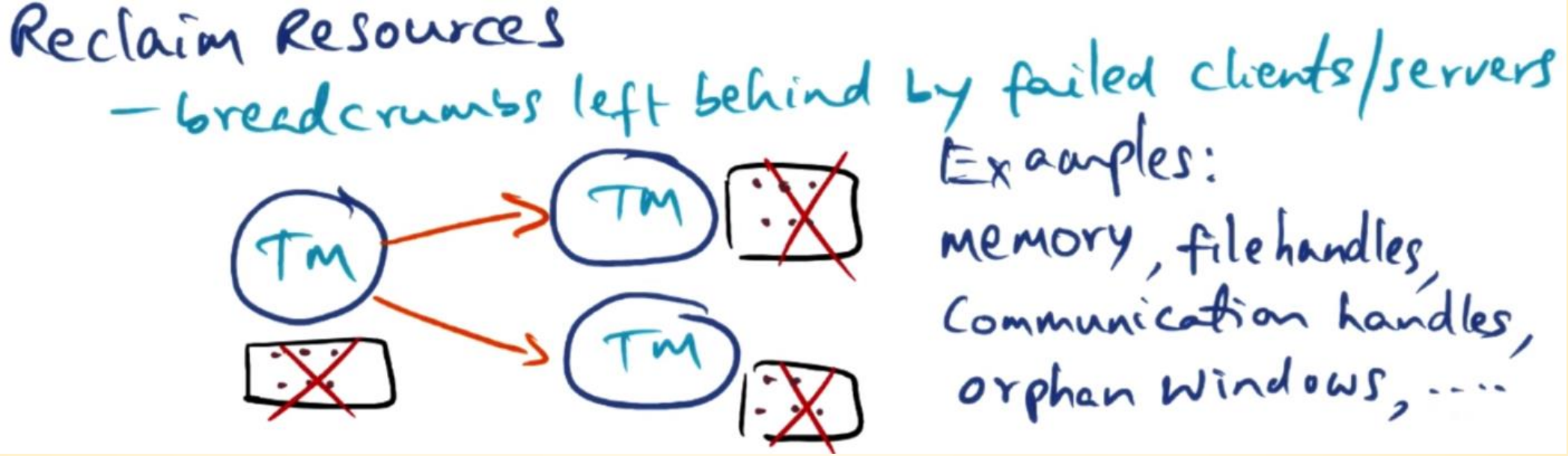
Distributed Transactions



Commit Initiated by Coordinator



Outcome of Bundling IPC + Recovery



No Extra communication for recovery.

Only mechanism in OS, policy is up to each service.

- Services can simply ignore these mechanisms if not needed, or
- Use low-overhead mechanisms for simple services, or
- Use weighty mechanisms for services such as FS.

QuickSilver Implementation Notes

Log maintenance

- TMs write log records for recovering persistent state
- Frequency of “log force” impacts performance.

Services have to be careful choosing mechanisms that commensurate with their recovery requirements.