Fault Tolerance

Advanced Cloud Computing

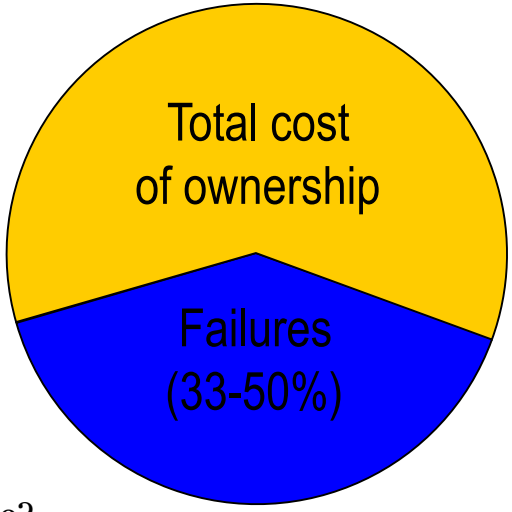
15-719/18-847b

Garth Gibson
Greg Ganger
Majd Sakr

Advanced Cloud Computing Fault Tolerance Readings

- Ref 1: “Implementing fault tolerant services using the state-machine approach: a tutorial.” Fred Schneider. ACM Computing Surveys, 1990. http://www.cs.cornell.edu/fbs/publications/smsurvey.pdf
- Ref 2: “Microreboot – A Technique for Cheap Recovery.” George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, Armando Fox. OSDI’04, 2004. https://www.usenix.org/legacy/event/osdi04/tech/full_papers/candea/candea.pdf
- Ref 3: “Disk failures in the real world: What does an MTTF of 1,000,000 hours mean too you?” FAST’07, 2007. http://www.cs.toronto.edu/~bianca/papers/fast07.pdf
- Ref 4: “Characterizing cloud computing hardware reliability.” Kashi Venkatesh Vishwanath, Nachiappan Nagappan. SOCC’10, 2010. http://doi.acm.org/10.1145/1807128.1807161

Failures are expensive



- Failures greatly impact cost of ownership
- Storage failures can be particularly expensive
  - *Unavailability* can cost millions per hour
    - How decreased response time and lowered thruput is still available?
  - *Data loss* can cost millions per 100MB
- USENIX Computer Failure Data Repository
  - 30+ clusters, 6+ sites, HPC & internet services (2000-2006)
  - > 23,000 failures; > 100,000 disk drives
- MS Datacenter paper (shortly before 2010)
  - > 100,000 machines studied for 14 months

USENIX failure data: hardware replacement logs

		Type of drive	Count	Duration
Pittsburgh Supercomputing Center	HPC1	18GB 10K RPM SCSI 36GB 10K RPM SCSI	3,400	5 yrs
Los Alamos NATIONAL LABORATORY EST.1943	HPC2	36GB 10K RPM SCSI	520	2.5 yrs
 Supercomputing X	HPC3	15K RPM SCSI 15K RPM SCSI 7.2K RPM SATA	14,208	1 yr
 Various HPCs	HPC4	250GB SATA 500GB SATA 400GB SATA	13,634	3 yrs
 Internet services Y	COM1	10K RPM SCSI	26,734	1 month
	COM2	15K RPM SCSI	39,039	1.5 yrs
	COM3	10K RPM FC-AL 10K RPM FC-AL 10K RPM FC-AL 10K RPM FC-AL	3,700	1 yr

Relative frequency of component replacements

The top ten of replaced components

HPC1	
Component	%
Hard drive	**30.6**
Memory	28.5
Misc/Unk	14.4
CPU	12.4
PCI motherboard	4.9
Controller	2.9
QSW	1.7
Power supply	1.6
MLB	1.0
SCSI BP	0.3

COM1	
Component	%
Power supply	34.8
Memory	20.1
Hard drive	**18.1**
Case	11.4
Fan	8.0
CPU	2.0
SCSI Board	0.6
NIC Card	1.2
LV Power Board	0.6
CPU heatsink	0.6

COM2	
Component	%
Hard drive	**49.1**
Motherboard	23.4
Power supply	10.1
RAID card	4.1
Memory	3.4
SCSI cable	2.2
Fan	2.2
CPU	2.2
CD-ROM	0.6
Raid Controller	0.6

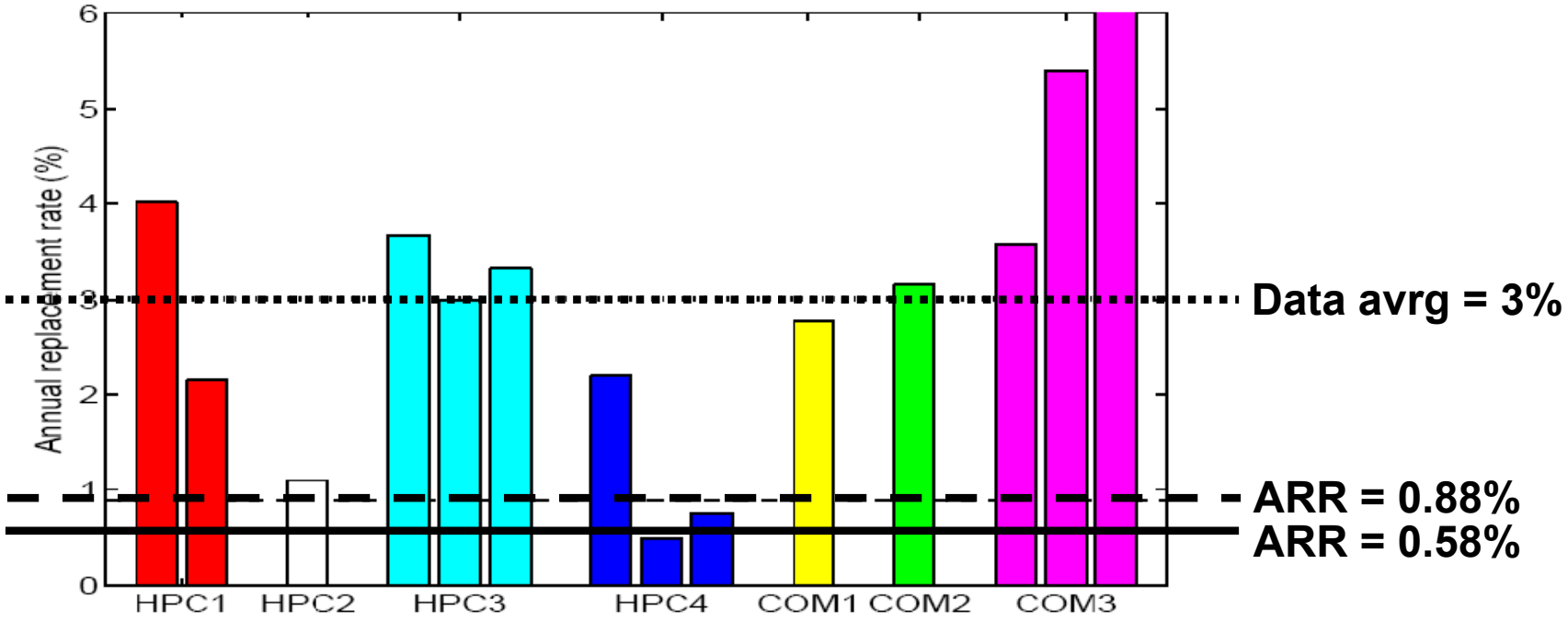
- All hardware fails, though disks failures often common

Annual disk replacement rate (ARR)

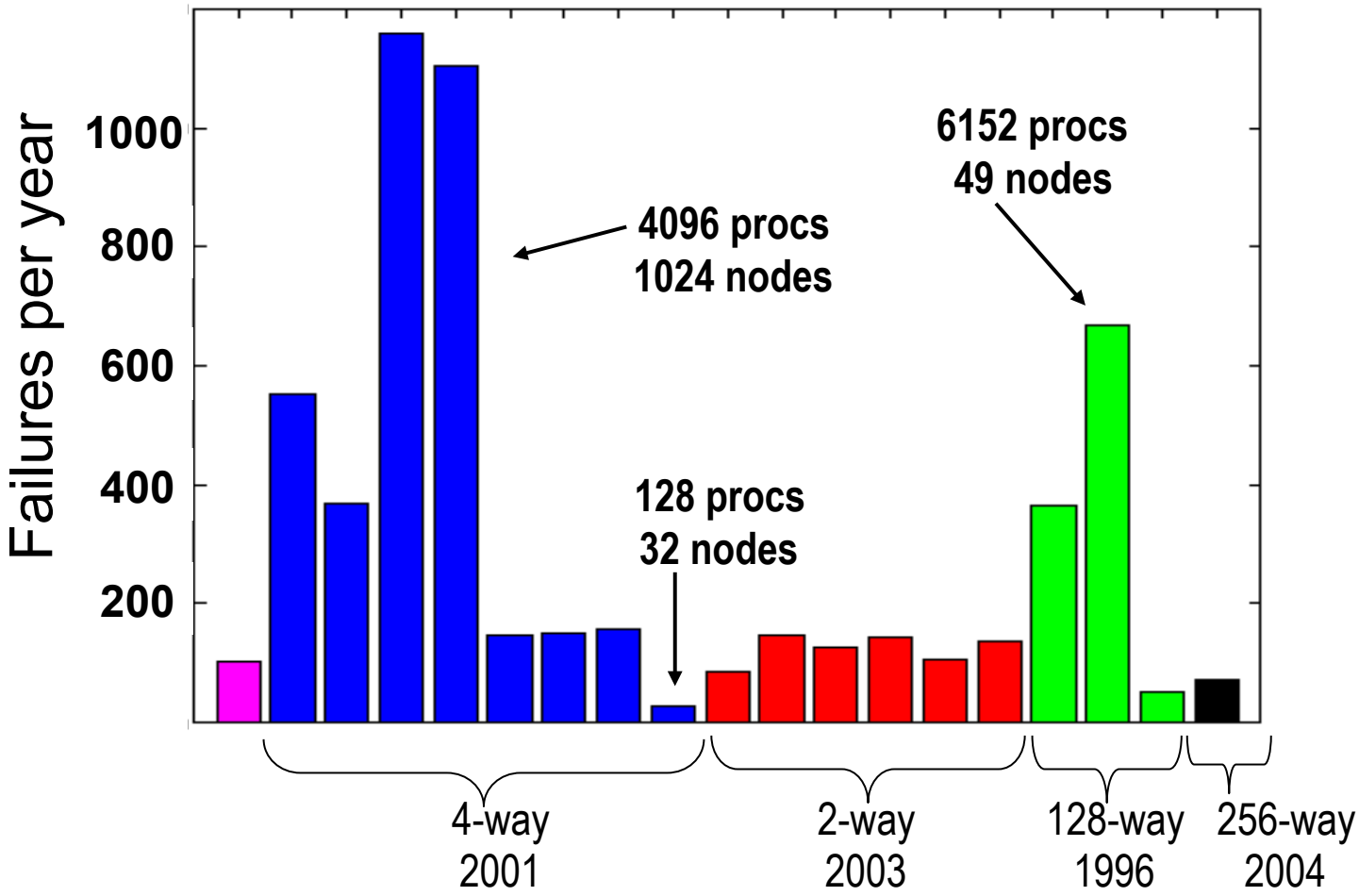
- Datasheet MTTFs are 1,000,000 to 1,500,000 hours.

⇒ Expected annual replacement rate (ARR): 0.58 - 0.88 %

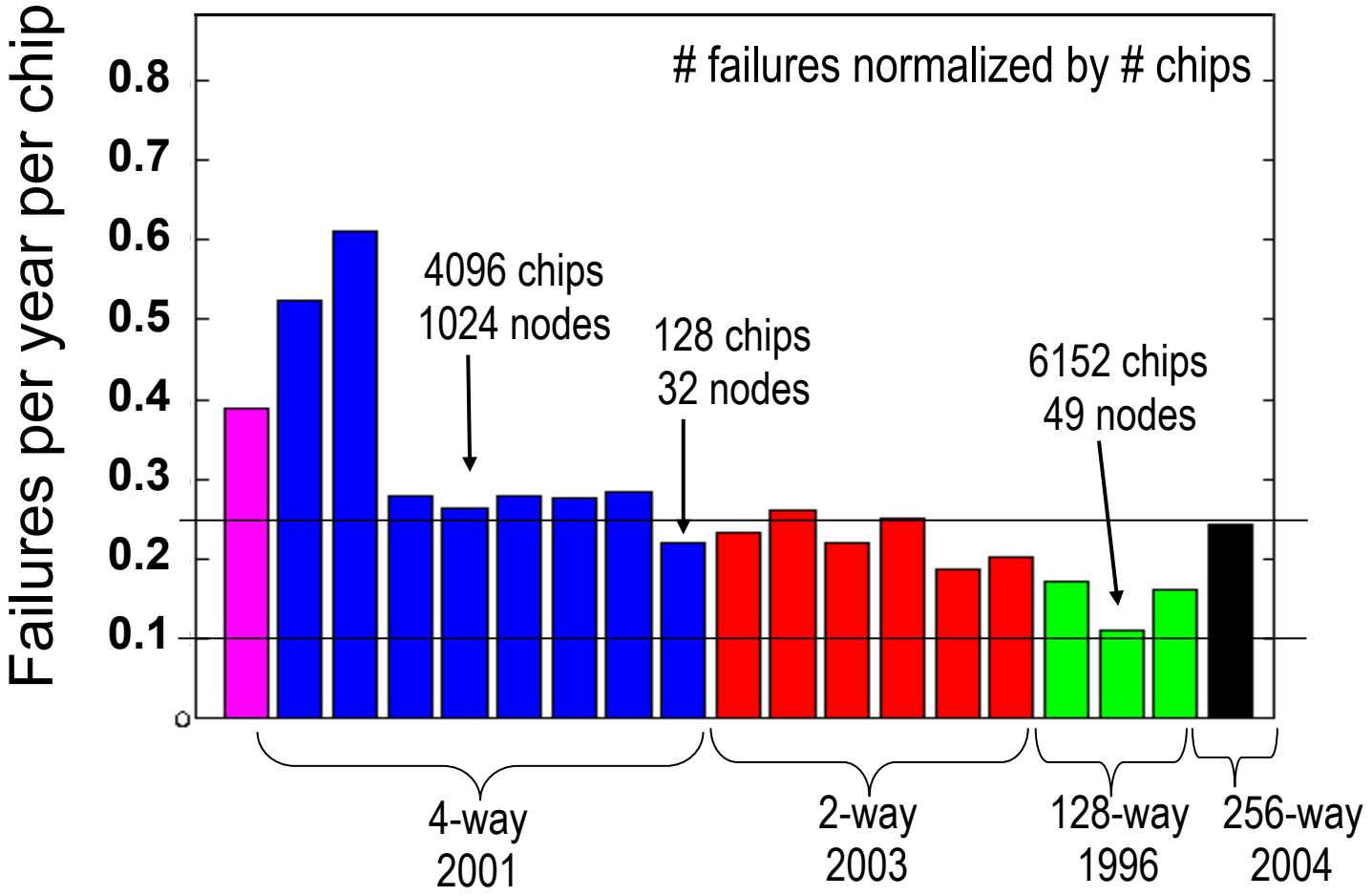
- Vendor sees “no fault found” for about 50% of returns
- Customer also breaks SLA/warranty terms: heat, moisture, vibration, workload



System failure rate highly variable

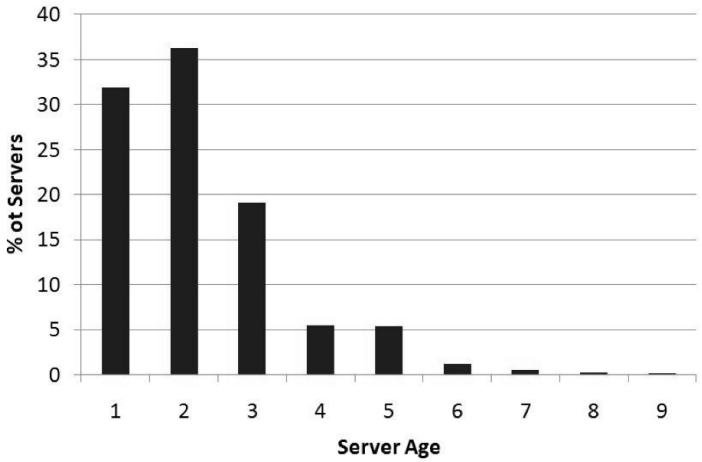


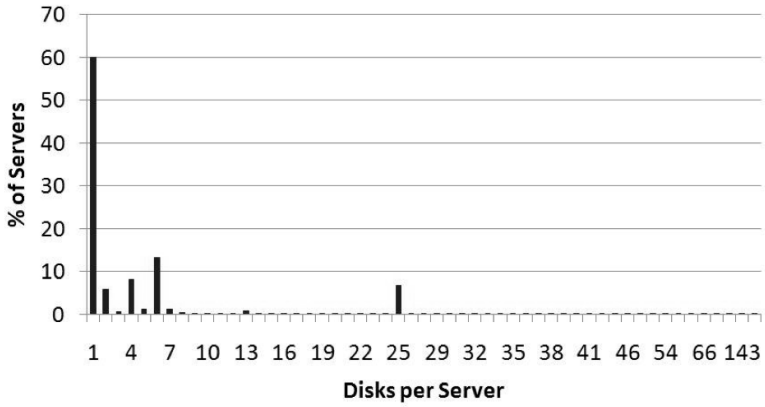
Best model: failures track # of processor chips

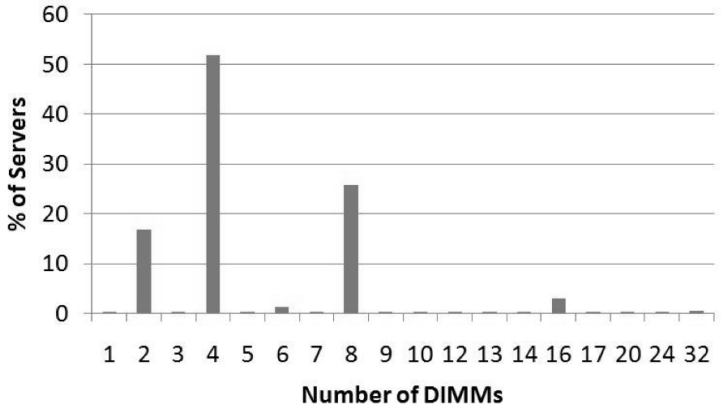


Microsoft datacenter systems

- Over 100,000 machines
- Unreported socket/chip count, usage
- Studied 14 months of hardware logs
- 8% machine hardware fails annually
  - 50% repaired once, 85% < 4 times
  - 78% failures caused by hard disks
    - 2.7% disks repaired per year
  - 5% were RAID controllers
  - 3% were memory DIMM failures
  - 20% repeat failures within 1 day
    - 50% repeat failures within 2 weeks







How does (software) handle failures? Redundant storage and repeatable computation

- Media failure (fail-stop)
  - IO device code bugs, disk HW failures: loss of (durable) disk info
- System failure (fail-stop)
  - DB bug, OS fault, HW failure:

    wipe out volatile memory but durable memory (disk) survives
- Short, non-shared, deterministic programs (most of them)
  - OS, framework or user destroys partial changes, then reruns program
    - Builds on external storage independently protected (RAID/Replicas)
- Long running, non-shared, deterministic programs (simulation, ETL)
  - Periodic stop and checkpoint state to durable, independent, protected storage
    - Components/tasks may checkpoint independently (less synchronization)
  - On failure, isolate failed component/system, then restart from checkpoint
    - Dependent components/systems are waiting & can trigger failure detection

How does software handle soft failures?

- Micro-reboot applies restart to long running system software components
  - Components must be transactional, store state externally, restart from external state, be loosely coupled, support locks that expire and requests that are retry-able
- Concurrent, shared data (database) multi-application systems
  - ACID transactions and write-ahead logging governing all shared state
    - Builds on external state independently protected
- Concurrent, shared-nothing replicated systems (maybe no external state)
  - Replicated state machines driven by coordinating replica changes

Transactions

- Multiple users manipulating shared data safely
  - Users == application processes, assume to be less experienced/skilled
- ACID properties of a transaction (a "user" interaction with DB)
  - Atomicity: a specific transformation is done all or nothing
    - All partial changes must be tentative til one committing change
  - Consistency: users make only (application defined) correct changes
  - Isolation: partial changes not visible to other user's code (less complex)
  - Durability: changes survive subsequent failures
    - Basic notion is storage redundancy/RAID, but can be process redundancy
- AID provided by database system, C (mostly) by programmer
  - DB is consistent iff contents result only from successful transactions
  - Integrity constraints (partial consistency) may be enforced by DB
    - Replica consistency is an important special case (later)

Isolation: Two-Phase Locking (2PL)

- Simplify for user: think only about running one transaction at a time
- Assuming well-formed/consistent transactions seeking isolation
  - Simple locking: Hold a (shared) lock to read & exclusive lock to write
    - Can fail to provide isolation (if transactions interleave mutation/locking)
- 2PL: acquire no lock after releasing any
  - Sufficient to insure isolation
  - Strict 2PL: release no lock before committing, avoids cascading aborts
  - Locks held a long time increase blocking; decrease concurrency
- Optimistic methods don't take/hold locks but may abort & retry
  - Record all variables touched and check for conflicts on commit
  - Faster if conflict is rare, but risks livelock if not

Failure Types

- Media failure (fail-stop)
  - IO device code bugs, disk HW failures: loss of disk info
  - Rare events, “hours” to recover from checkpoints & audit logs
- System failure (fail-stop)
  - DB bug, OS fault, HW failure:
    wipe out volatile memory but durable memory (disk) survives
  - Infrequent events, “minutes” to recover
- Transaction failure
  - Code aborts, based on input/database inconsistency
    [sometimes programmer is just escaping complex corner cases in code]
  - Mechanical aborts caused by concurrency control solutions to isolation
  - Frequent events, “instant” recovery needed

Recoverable Database System Model

- Log changes durably before database changes durable
  - Write-ahead logging
  - Once a committed transaction has been logged separately, multiple changes to database can be serialized & retry will "REDO" work
- REDO: repeat completed transaction on old DB data
  - Partial system or total media failure
- UNDO: rollback aborted transaction
  - Transaction or system failure
  - Only if uncommitted transaction allowed to change durable media

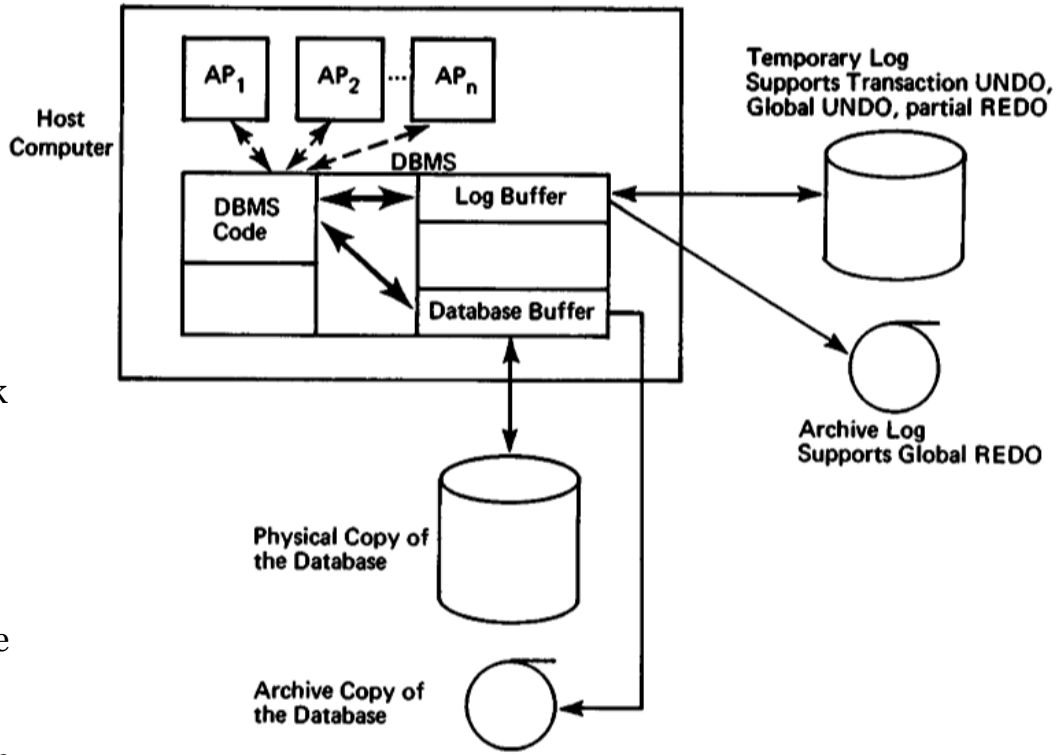


Figure 4. Storage hierarchy of a DBMS during normal mode of operation.

Replicated State Machines

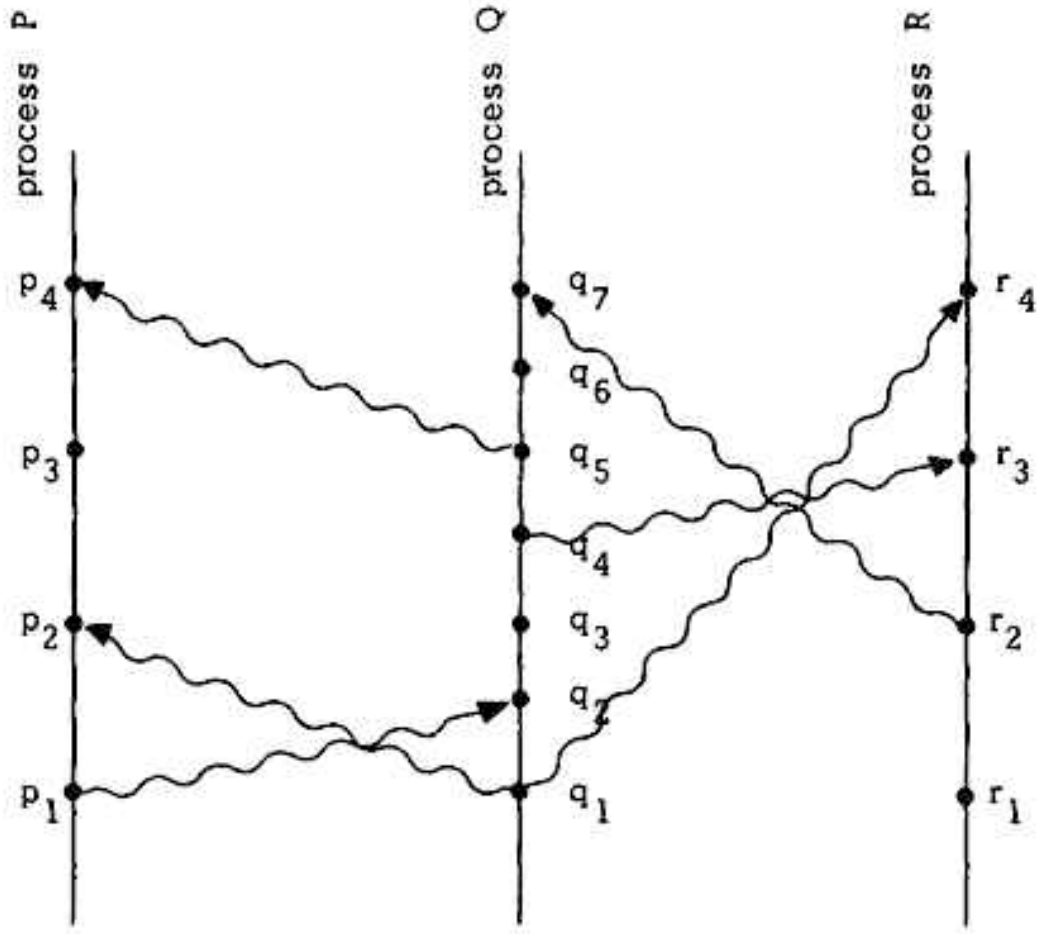
- A state machine is code and data that acts predictably and deterministically to input commands
- All non-faulty replicas of a service started with the same state and executing the same commands produce the same state & output
  - If failures are simple random “fail-stop”, 1 surviving replica is sufficient
  - If failures are malicious deceivers, non-faulty survivors must win a vote
    - Need 2t+1 replicas to survive t malicious (Byzantine) failures
- Common tools for replicated state machines
  - Part-time parliament or PAXOS [Lamport89], ZooKeeper, RAFT, ….

Agreement & Ordering

- Decompose Replicated State Machine protocol into two:
  - Agreement – deliver every request to all non-faulty machines
  - Ordering – ensure the same order of execution at all non-faulty machines

Concurrency & “Happens Before”

- Two events are not concurrent if one “happens before” the other
- Eg. P1 happens before R3 but P2 and R4 may be concurrent
- Replicated state machine wants same order of changes at all replicas

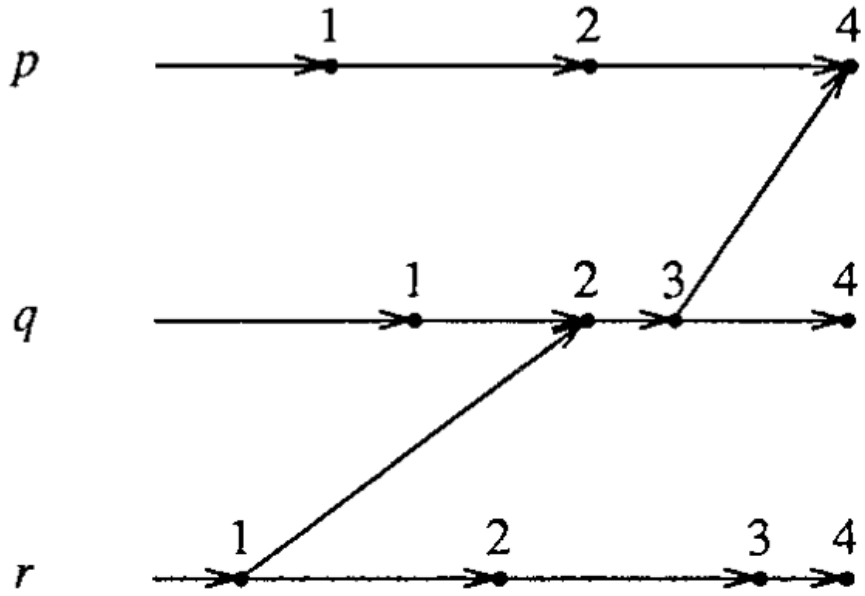


Agreement & Ordering

- Decompose Replicated State Machine protocol into two:
  - Agreement – deliver every request to all non-faulty machines
    - A coordinator/client specifies a request & the rest agree
  - Ordering – ensure the same order of execution at all non-faulty machines
    - Assign identifiers to requests and execute in identifier order
    - Use a clock – three kinds: logical, real-time, server generated
    - Client sends a logical clock with every message, or
    - Every machine has & sends real-time clocks, or
    - Servers/replicas negotiate a clock/identifier for order

Logical clocks

- Every machine maintains a counter for its (orderable) events
- When a message arrives, carrying the sender's counter the receiver advances its counter past the sender's
  - C' = max(C, msg-C) + 1
  - Resolve ties by adding machine/thread ID as lower order bits
- Defines a total order that that is consistent with "happens before"



Replicated State Machines using Logical Clocks

- In order to decide what request to execute next, need to know that no request with a lower logical clock may arrive in future
- Require messages between two machines arrive in order (e.g. TCP)
- Delay execution at a replica until it has heard a larger logical clock from all non-faulty machines
  - The requests being held all happened before the latest messages, so a smallest identifier can be selected and executed (following total order)
- Waiting for later messages is undesirable
  - Forces heartbeat messages, and significant latency
  - Real-time clocks can fix this if clock skew is smaller than message delivery
  - Replicas can negotiate an order by communicating among themselves
    - At the cost of extra messaging

Next up

- Latency and the tyranny of stragglers ☺
- More on Failure