

Distributed Systems

23. Fault Tolerance

Paul Krzyzanowski
pxk@cs.rutgers.edu

Faults

- Deviation from expected behavior
- Due to a variety of factors:
 - Hardware failure
 - Software bugs
 - Operator errors
 - Network errors/outages

Faults

- Three categories
 - transient faults
 - intermittent faults
 - permanent faults
- Any fault may be
 - Fail-silent (fail-stop)
 - Byzantine
- synchronous system vs. asynchronous system
 - E.g., IP packet versus serial port transmission

Fault Tolerance

- Fault Avoidance
 - Design a system with minimal faults
- Fault Removal
 - Validate/test a system to remove the presence of faults
- Fault Tolerance
 - Deal with faults!

Achieving fault tolerance

- Redundancy
 - information redundancy
 - Hamming codes, parity memory ECC memory
 - time redundancy
 - Timeout & retransmit
 - physical redundancy/replication
 - TMR, RAID disks, backup servers
- Replication vs. redundancy:
 - Replication:
 - multiple identical units functioning concurrently – vote on outcome
 - Redundancy:
 - One unit functioning at a time: others standing by

Availability: how much fault tolerance?

100% fault-tolerance **cannot** be achieved

- The closer we wish to get to 100%, the more expensive the system will be
- Availability: % of time that the system is functioning
 - Typically expressed as # of 9's
 - Downtime includes all time when the system is unavailable.

Availability

Class	Level	Annual Downtime
Continuous	100%	0
Six nines <small>(carrier class switches)</small>	99.9999%	30 seconds
Fault Tolerant <small>(carrier-class servers)</small>	99.999%	5 minutes
Fault Resilient	99.99%	53 minutes
High Availability	99.9%	8.3 hours
Normal availability	99-99.5%	44-87 hours

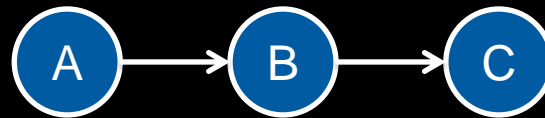
Points of failure

- Goal: avoid single points of failure
- Points of failure: A system is *k-fault tolerant* if it can withstand k faults.
 - Need $k+1$ components with silent faults
 k can fail and one will still be working
 - Need $2k+1$ components with Byzantine faults
 k can generate false replies: $k+1$ will provide a majority vote

Active replication

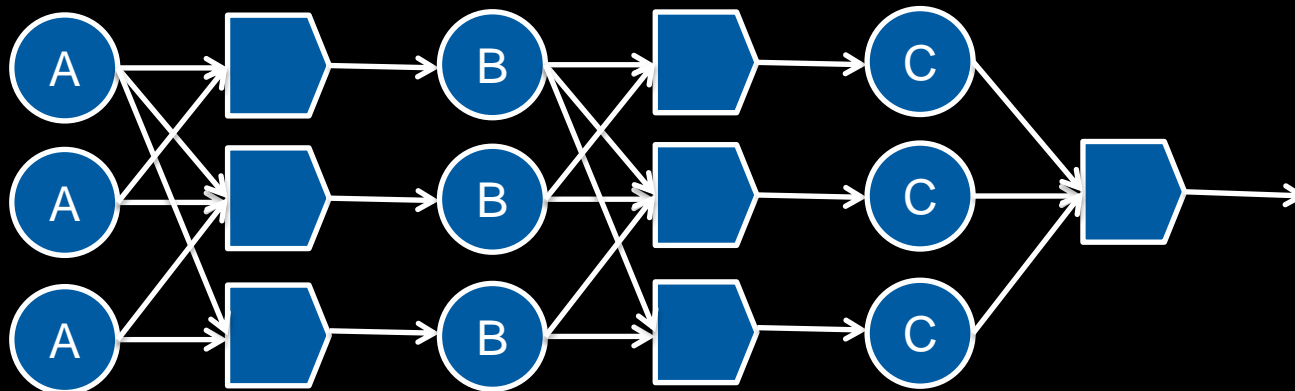
Technique for fault tolerance through physical redundancy

No redundancy:



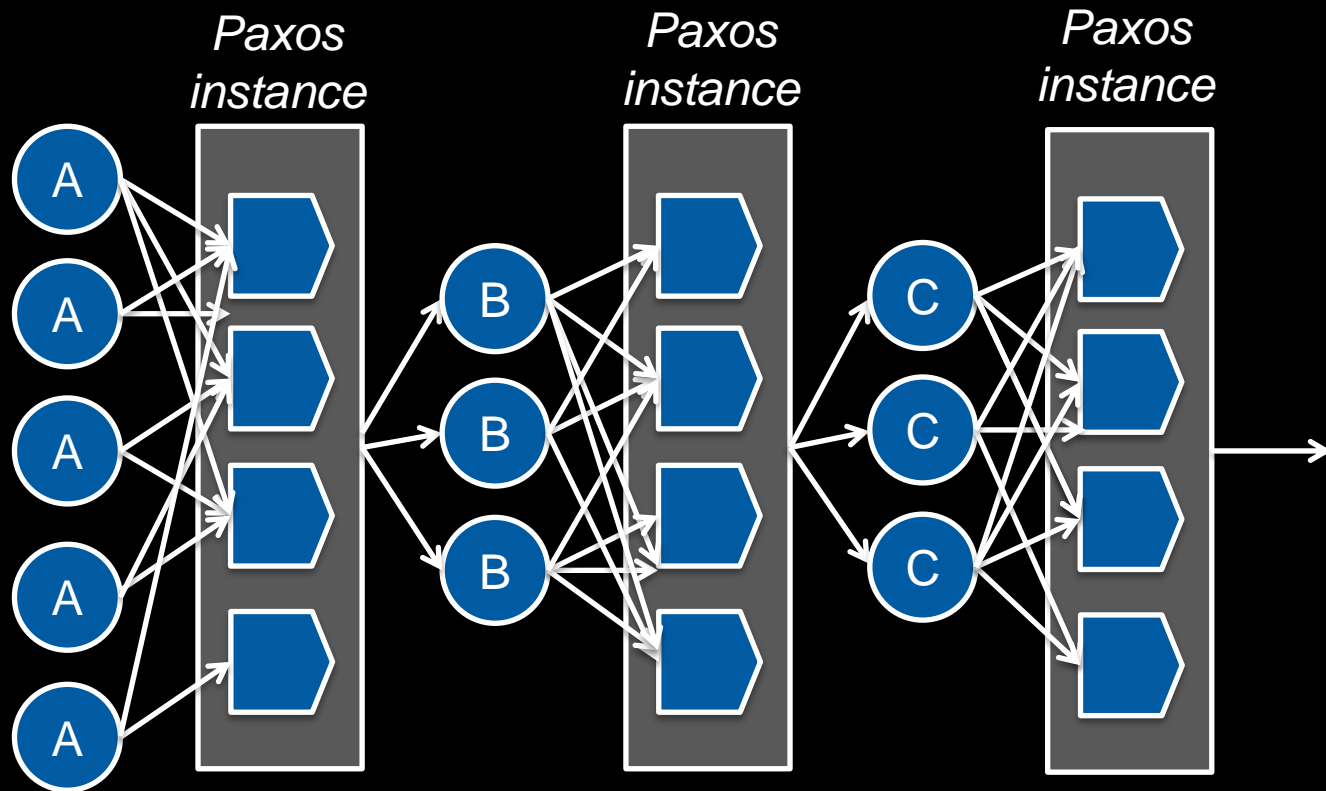
Triple Modular Redundancy (TMR):

Threefold component replication to detect and correct a single component failure



Active replication: State Machine Sync

Use a distributed consensus algorithm to synchronize intermediate outputs



Primary backup

- One server does all the work
- When it fails, backup takes over
 - Backup may ping primary with *are you alive* messages
- Simpler design: no need for multicast
- Works poorly with Byzantine faults
- Recovery may be time-consuming and/or complex

Examples of Fault Tolerance

Example: ECC memory

- Memory chips designed with Hamming code logic
- Most implementations *correct* single bit errors in a memory location and *detect* multiple bit errors.
- Example of **information redundancy**

Example: Failover via DNS SRV

- Goal: allow multiple machines (with unique IP addresses in possibly different locations) to be represented by one hostname
- Instead of using DNS to resolve a hostname to one IP address, use DNS to look up SRV records for that name.
 - Each record will have a priority, weight, and server name
 - Use the priority to pick one of several servers
 - Use the weight to pick servers of the same priority (for load balancing)
 - Then, once you picked a server, use DNS to look up its address
- Commonly used in voice-over-IP systems to pick a SIP server/proxy
- MX records (mail servers) take the same approach: use DNS to find several mail servers and pick one that works
- Example of **physical redundancy**

Example: DNS with device monitoring

- Custom DNS server that returns an IP address of an available machine by monitoring the liveness of a set of equivalent machines
 - Akamai approach (Akamai has more criteria than this)

Example: TCP retransmission

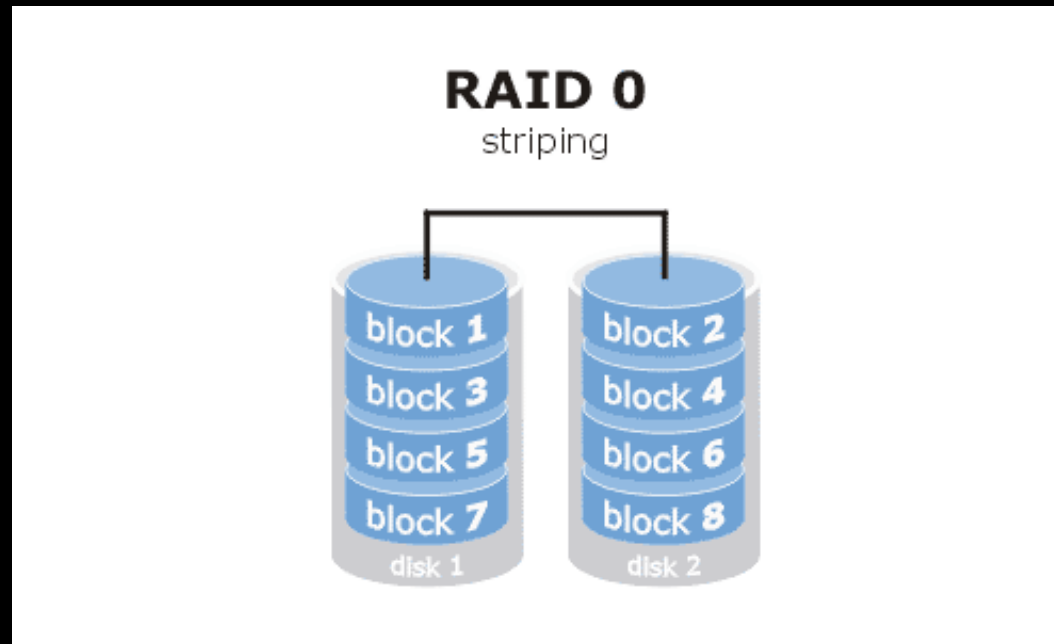
- Sender requires ack from a receiver for each packet
- If the ack is not received in a certain amount of time, the sender retransmits the packet
- Example of **time redundancy**

Example: RAID 1 (disk mirroring)

- RAID = redundant array of independent disks
- RAID 1: disk mirroring
 - All data that is written to one disk is also written to a second disk
 - A block of data can be read from either disk
 - If one disk goes out of service, the remaining disk will still have the data
- Example of **physical redundancy**

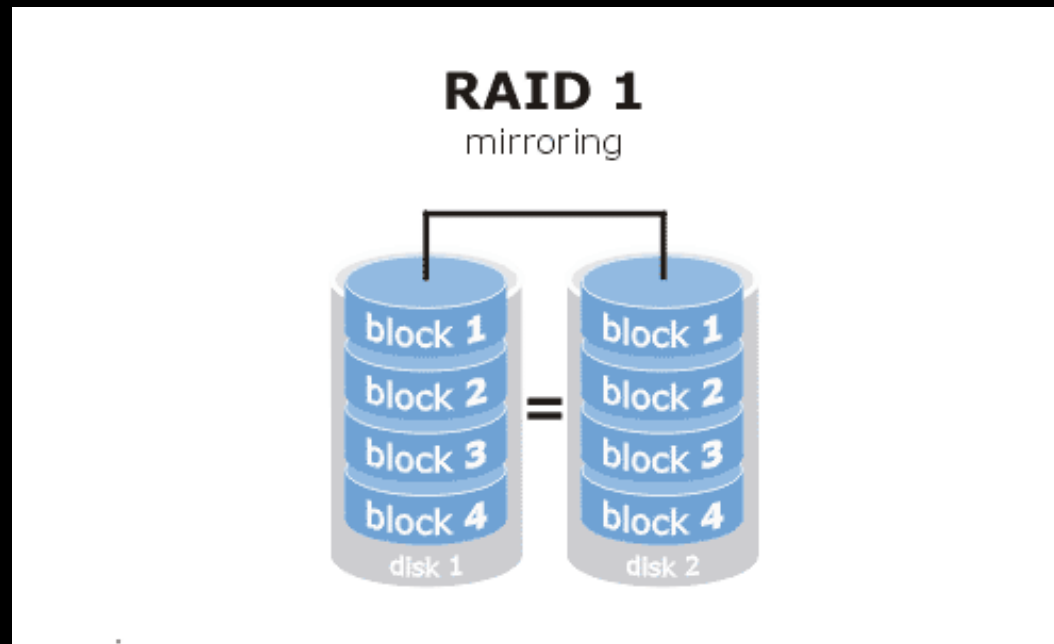
RAID 0: Performance

- **Striping**
- **Advantages:**
 - Performance
 - All storage capacity can be used
- **Disadvantage:**
 - Not fault tolerant



RAID 1: HA

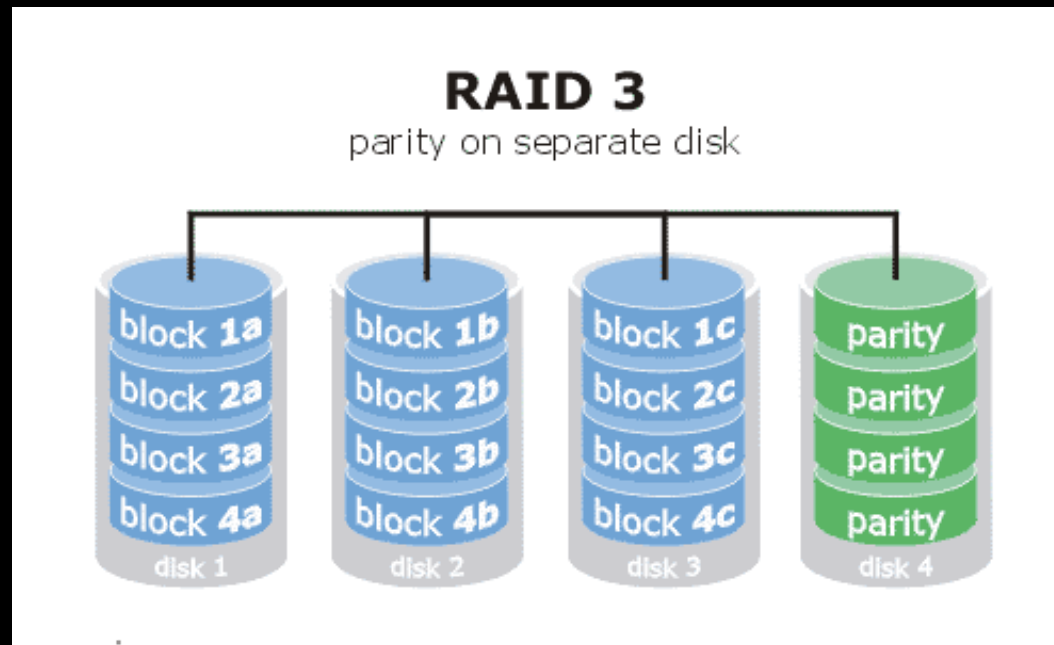
- **Mirroring**
- **Advantages:**
 - Double read speed
 - No rebuild necessary if a disk fails: just copy
- **Disadvantage:**
 - Only half the space
- **Physical Redundancy**



RAID 3: HA

- Separate parity disk
- Advantages:
 - Very fast reads
 - High efficiency: low ratio of parity/data
- Disadvantages:
 - Slow random I/O performance
 - Only one I/O at a time

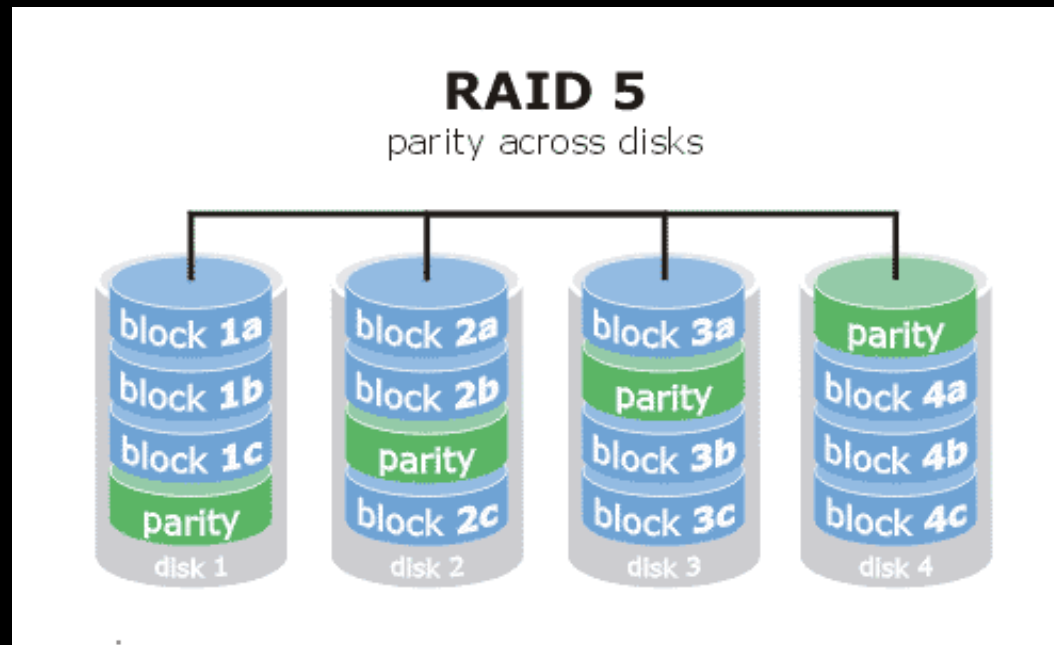
Physical +
Information redundancy



RAID 5

- Interleaved parity
- Advantages:
 - Very fast reads
 - High efficiency: low ratio of parity/data
- Disadvantage:
 - Slower writes
 - Complex controller

Physical +
Information redundancy



RAID 1+0

- Combine mirroring and striping
 - Striping across a set of disks
 - Mirroring of the entire set onto another set

The End