Dependable Systems

Fault Tolerance Patterns (II)

Dr. Peter Tröger

Source:

Hanmer, Robert S.: Patterns for Fault Tolerant Software. Wiley, 2007.

Error Recovery Patterns

Quarantine / Concentrated Recovery

- **Quarantine**
  - Prevent errors from spreading using *units of mitigation*
    - Establish barrier around the component
    - Prevent not only error spreading, but also work contribution
      - Example: Unsafe state indicator for voter result
- **Concentrated Recovery**
  - Minimize unavailability by focusing all resources on recovery activity
  - Inform *fault observer* about recovery activity, stay inside *unit of mitigation*
  - Establish *quarantine* around recovery activity

Error Handler / Restart

- **Error Handler**
  - Separate error processing code for easier maintenance
    - No conditional branches in main code, instead more generic clean-up actions
    - Final choice when no *recovery block* is usable
      - Language exception support helps in writing error handlers
- **Restart**
  - Way to resume execution when *recovery* / *escalation* is not possible
  - cold / warm restart - skip some of the initial checks, hardware vs. software restart
  - Supported by *checkpoints*

Rollback / Roll-Forward

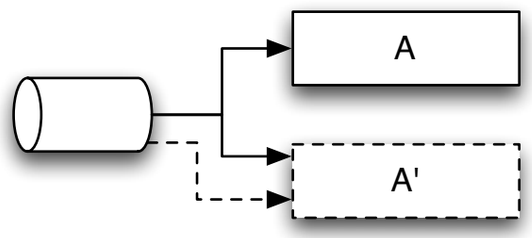
- How to resume processing after error recovery / error handler execution
- **Rollback**
    - Timing of the checkpoint / last requests decides about the rollback point
    - Consider side effects of repeated work
        - Errors might re-occur, so *limit retries*
- **Roll-Forward**
    - Resynchronization of systems tasks might be faster
    - Especially useful for event-driven stateless services
    - Demands proper damage mitigation and containment

Return to Reference Point / Limit Retries

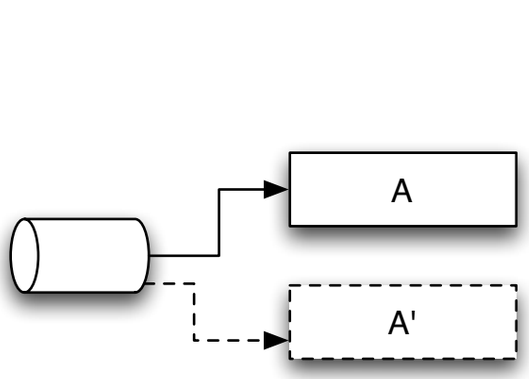
- **Return to Reference Point**
  - Scenario: *Rollback* and *Roll-Forward* are not appropriate, but *Restart* is too heavy
  - *Rollback* points are dynamic and contain error-free coherent state
  - *Reference points* are static, always available, and contain coherent state
    - Specified during design, typically in initialization, or before method return
- **Limit Retries**
  - Scenario: Faults are deterministic (latent fault -> same stimuli -> activation)
    - *Rollback* might not solve the problem when the error activation reason remains
    - Example: ‚Killer messages‘ marked as unprocessed, faulty checkpoints
- Problem: Propagation of error within itself, must be stopped by limiting retries
  - Solutions: *Safeguarding* and *roll-forward*

Failover

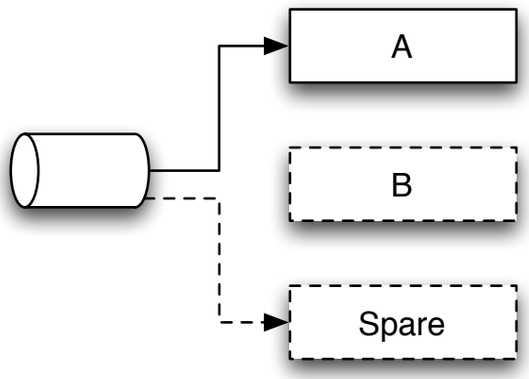
- Restoring of error-free operation in active element did not succeed
  - Switch to redundant resource, based on *replication*
  - Important factors are failover time and common data access
  - Establish *someone in charge* for steering
  - Needs proper *quarantine* for the faulty system part



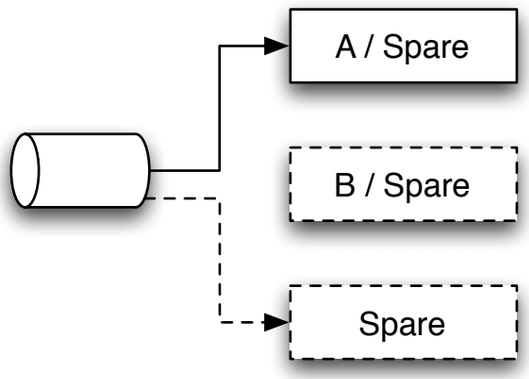
Active / Active



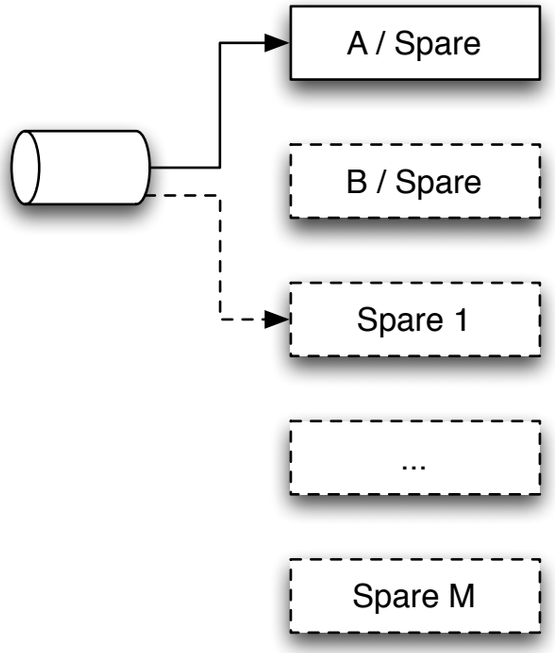
Active / Passive



N-to-1 (‚dedicated spare‘)



N+1 (‚roaming spare‘)



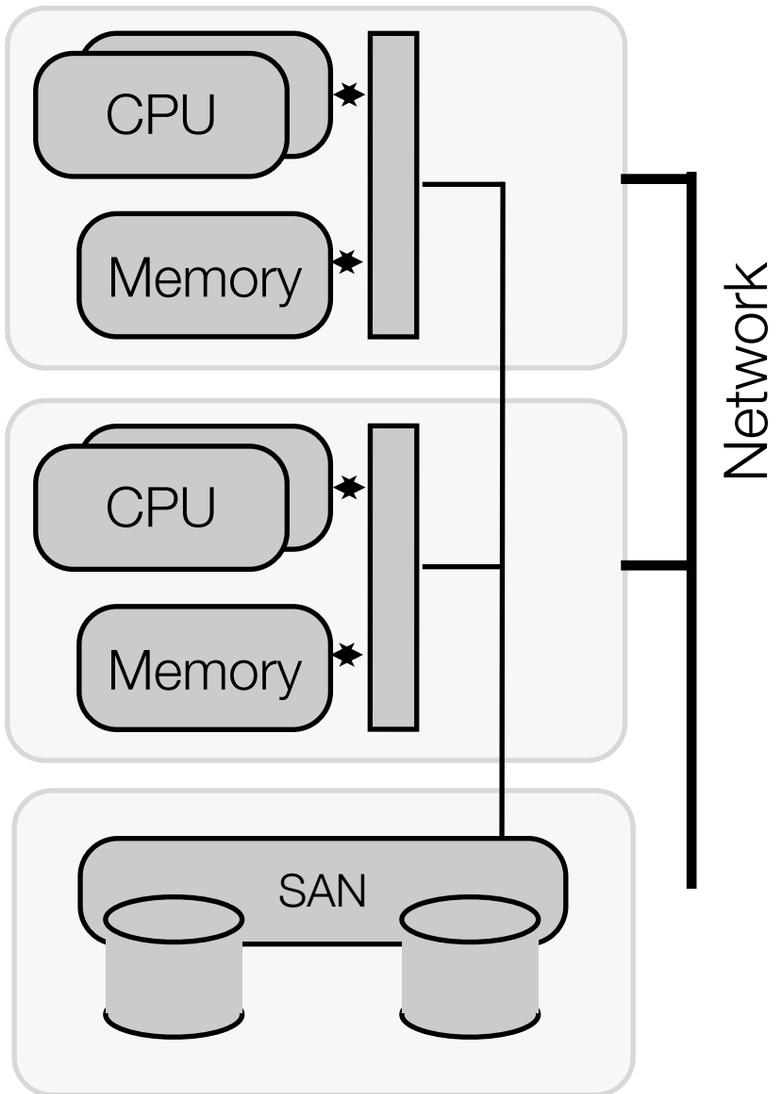
N + M

Redundancy Configurations for Failover

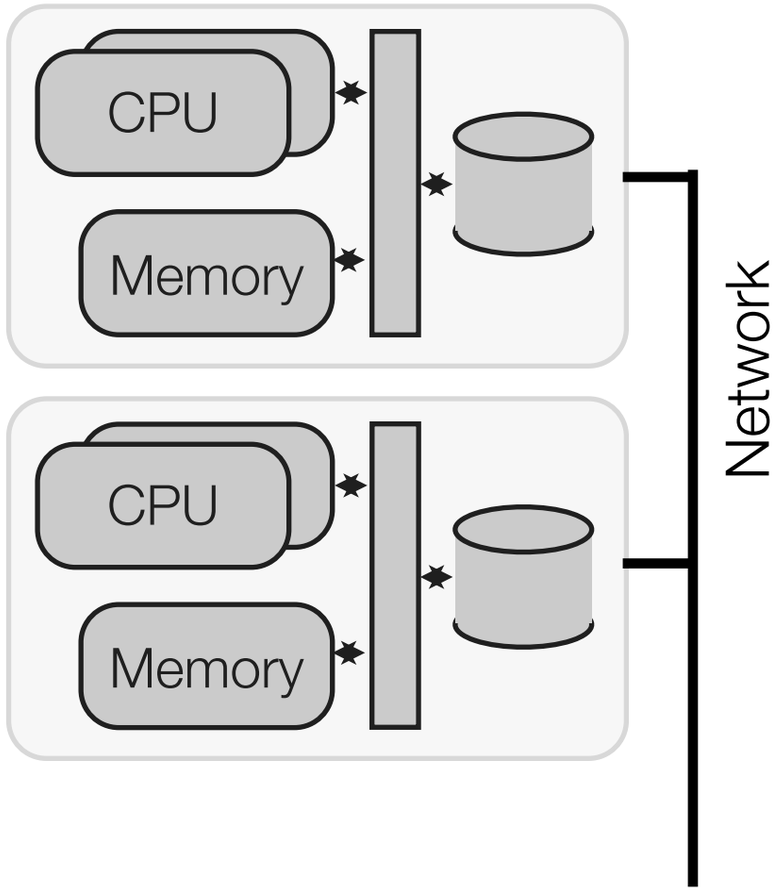
- *N-to-1* and *N+1* are special cases of *Active / Passive* with multiple services
- *Active / Active* has no downtime, but leads to degraded system performance in failover case and might demand specialized data *redundancy*
- *N-to-1* demands a fail-back step, which is not needed with *N+1*
- *Hot standby*: No ramp-up needed on failover, no service failure for the user
  - Natural property of *Active / Active* setups
  - Possible even with *Active / Passive* setting through continuous replication, stateless services or static data
- *Warm standby* resp. *log shipping*: Synchronize data block-wise on spare
- *Failover* is typically used as synonym for *Active / Passive*
- Orthogonal: *Shared Nothing* vs. *Shared Disk* data management

System-Disk Interconnect Models

Shared Disks



Shared Nothing



System-Disk Interconnect Models

	Shared Disks ‚Multi-Homing‘	Shared Nothing
Advantages	• Good availability • Good load-balancing	• Very good availability • Unlimited scalability • Low cost due to standard components
Disadvantages	• Limited scalability • Synchronization for concurrent update	• Difficult for load balancing • Difficult for performance optimization

Failover - Dual Master Problem

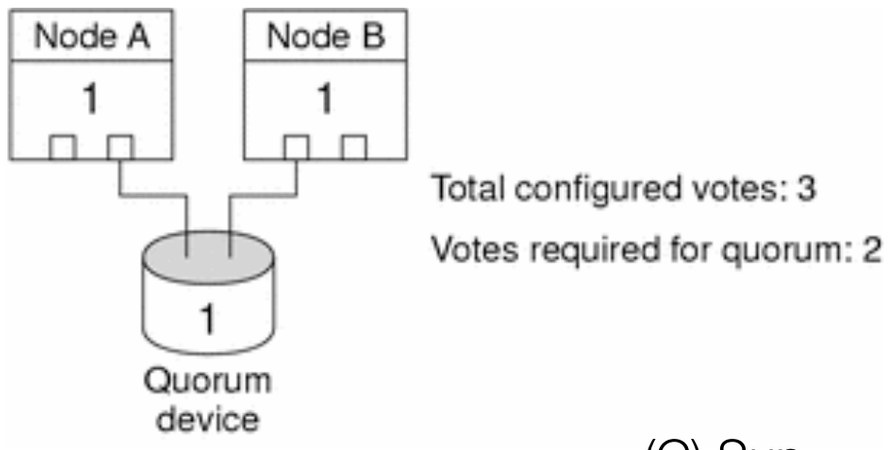
- Current active element might not relinquish control - **dual master** problem
- Typical problem in high-availability clusters
  - **Split brain** - Cluster interconnect is broken, several sub-cluster partitions start up
  - **Amnesia** - Cluster restart with outdated configuration
    - Establish **resource fencing** to let only one sub-group of the cluster work
- **Quorum** - *„The number (as a majority) of officers or members of a body that when duly assembled is legally competent to transact business“* [Merriam-Webster]
  - ‚Transact business‘ in the sense of ‚provide service‘ - only one side should operate
  - Quorum allows fencing the other sub-cluster without communication
  - Loss of quorum should lead to node suicide, if possible
- Amnesia solved by having a copy of the cluster configuration on each node

Failover - Dual Master Problem

- **Central arbitration** - Manual quorum, centralized server / admin sets master
- **Simple majority** - More than the half of the nodes must form a group
- **Weighted majority** - Votes for each node, group with higher vote count wins
  - Group decision is based on static data (nr. of votes, majority needed)
- **Tie-breaker** - Lightweight resolving before voting
  - Example: Ping response from common upstream router
- Whenever node connectivity changes, quorum decision happens again
- Split brain has different faces
  - Example: DRBD file system, multiple replication masters by human error or temporary connectivity lost - merging is difficult

Failover - Weighted Majority with Quorum Device

- With even node number, provide additional external vote through **quorum device**
  - Number of votes by the quorum devices should be less than node votes
    - Allows cluster to operate with failed quorum device
  - Connection scheme of the quorum device decides upon valid cases of partitioning
  - Quorum device is typically a shared disk
  - Only used when communication with other nodes fails
  - Implemented by SCSI RESERVE, Fibre Channel, or iSCSI

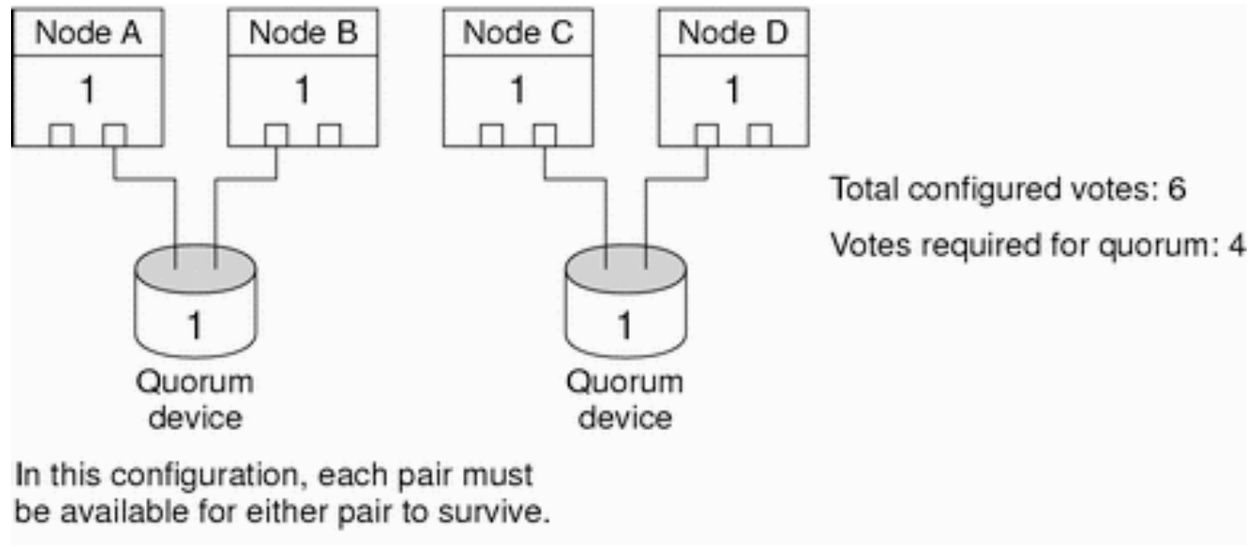


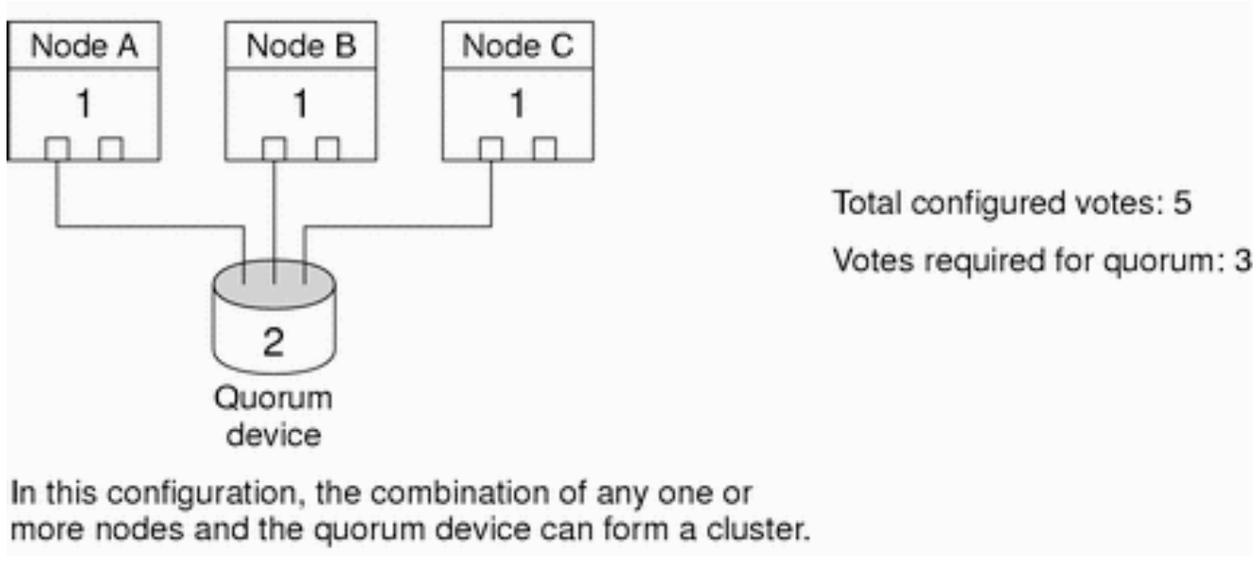
(C) Sun

SCSI Quorum Device

- Only one SCSI device can use the bus at a time - *arbitration process*
  - LUN acts as priority, so host bust adapters typically have the highest one
- SCSI commands RESERVE and RELEASE allow to lock one SCSI device for exclusive usage by another device
  - Automated release on device / bus reset
  - Periodical renewing of reservation by driver, or persistent reservation feature
- Example MS Windows Cluster Server
  - Master node acts as *defender*, renews reservation every 3 seconds
  - One node communication loss, *challenger* nodes resets the bus, waits for 7 seconds, and tries to get the reservation again

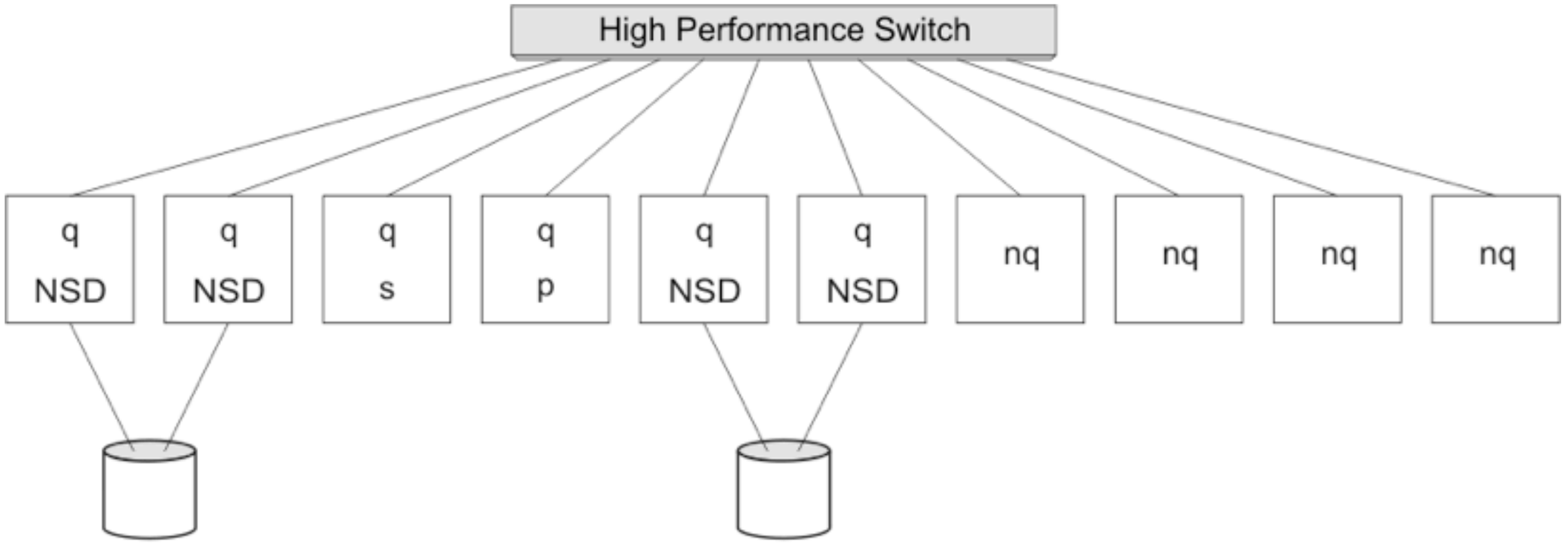
Example - Quorum in Clusters





(C) Sun

Example - GPFS Explicit Node Quorum



q - quorum node

NSD - NSD server

s - secondary cluster configuration server

p - primary cluster configuration server

nq - non-quorum node

(C) IBM

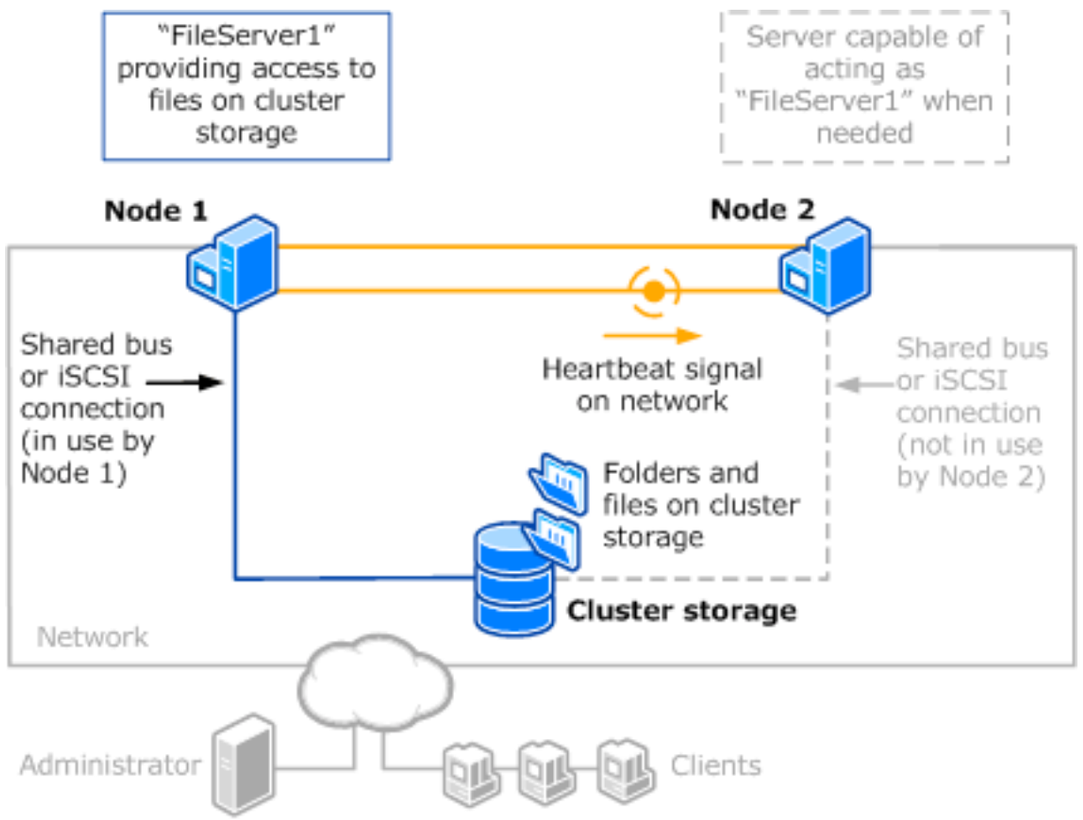
Example - Windows Server 2008 Failover Cluster

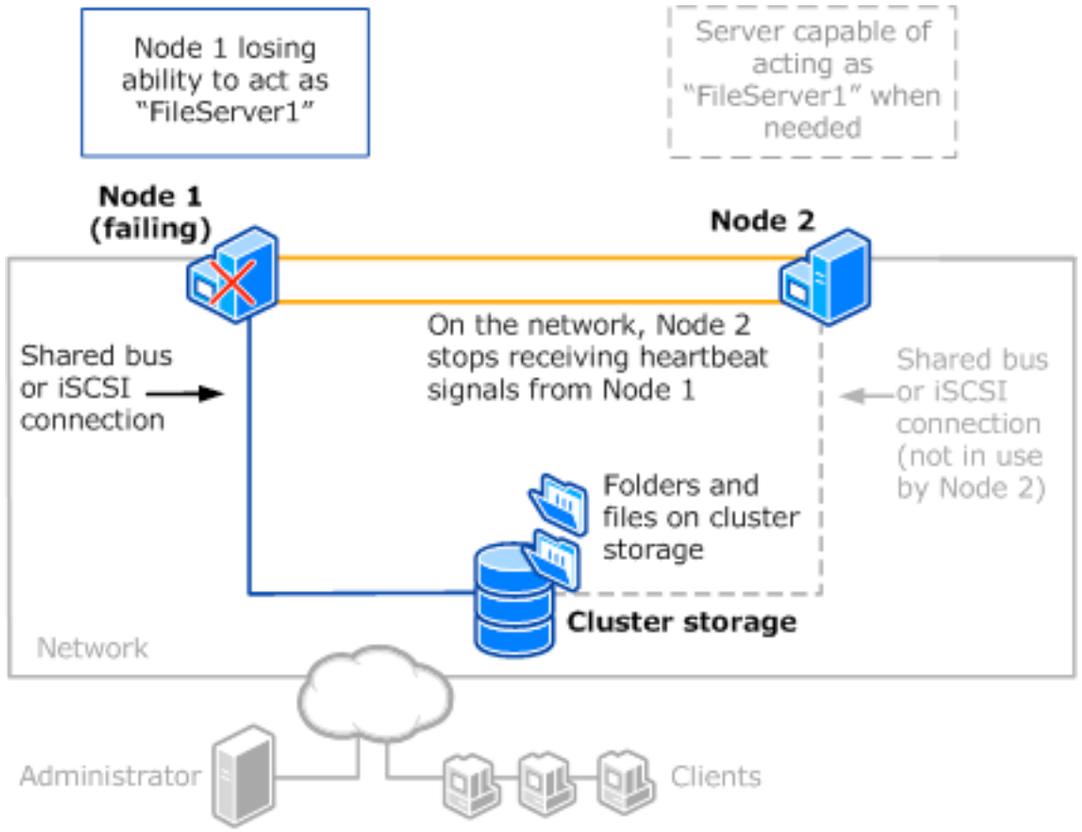
- Voting elements: Nodes, disk witness, file share witness (= tie-breaker)
- Quorum modes
  - *Node majority*, for odd node number
  - *Node and disk majority*, for even node number with shared storage
  - *Node and file share majority*, for even node number in multi-site cluster
  - *No majority: disk only*, disk-based quorum as in Windows Server 2003
- File share / disk contains information about most recent cluster configuration
- Disk mode: Hardware must offer persistent arbitration - single node gains physical control and defends it (e.g. SCSI reserve and release)
- File share mode: Active node keeps open file lock on the share

Example - Windows Server 2008 Failover Cluster

- Permanent point-to-point heartbeat surveillance on each node
- Process of achieving quorum
  - As the node comes up, determine if other cluster members can be contacted
  - Members compare their membership view on the cluster and agree on one
  - Member collection determines if it has quorum
    - Without enough votes, it is waited for more members to appear
    - With quorum attended, resources and applications are brought into service

Example - Windows Failover File Server



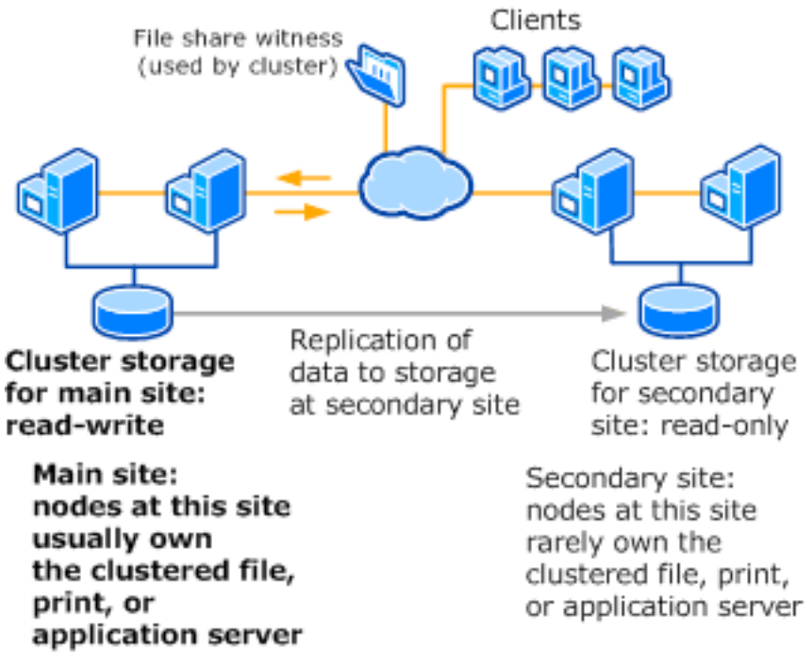


(C) Microsoft

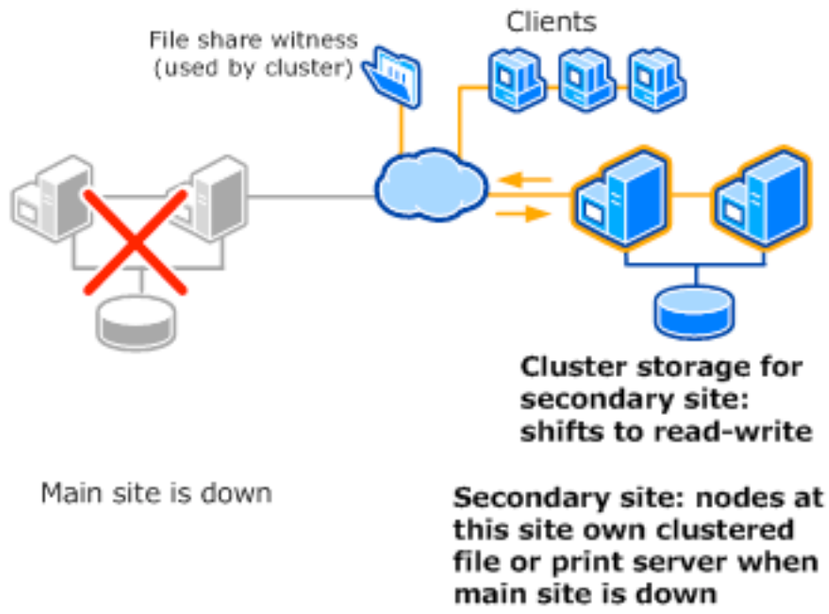
Example - Windows Multi-Site Clustered File Server

(C) Microsoft

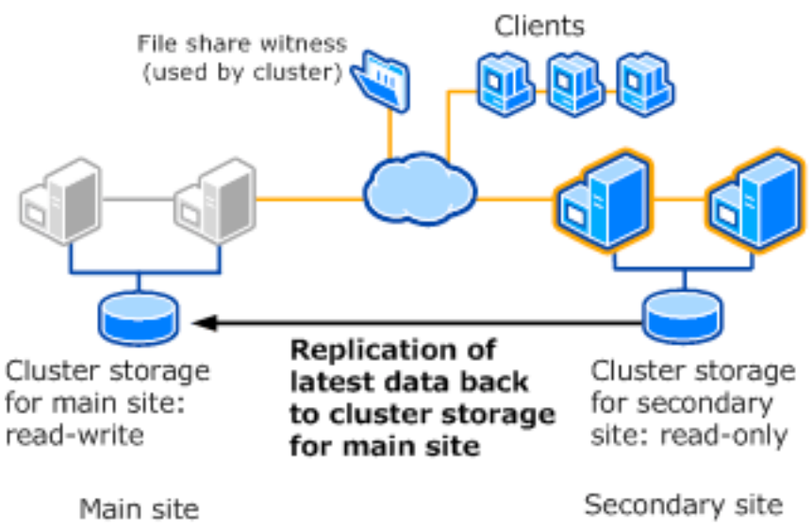
Normal Conditions



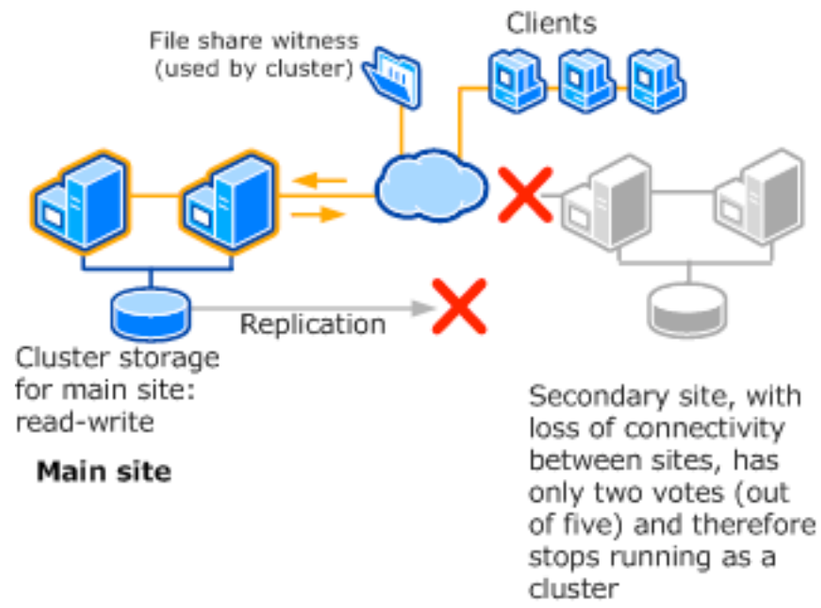
Main Site Gone



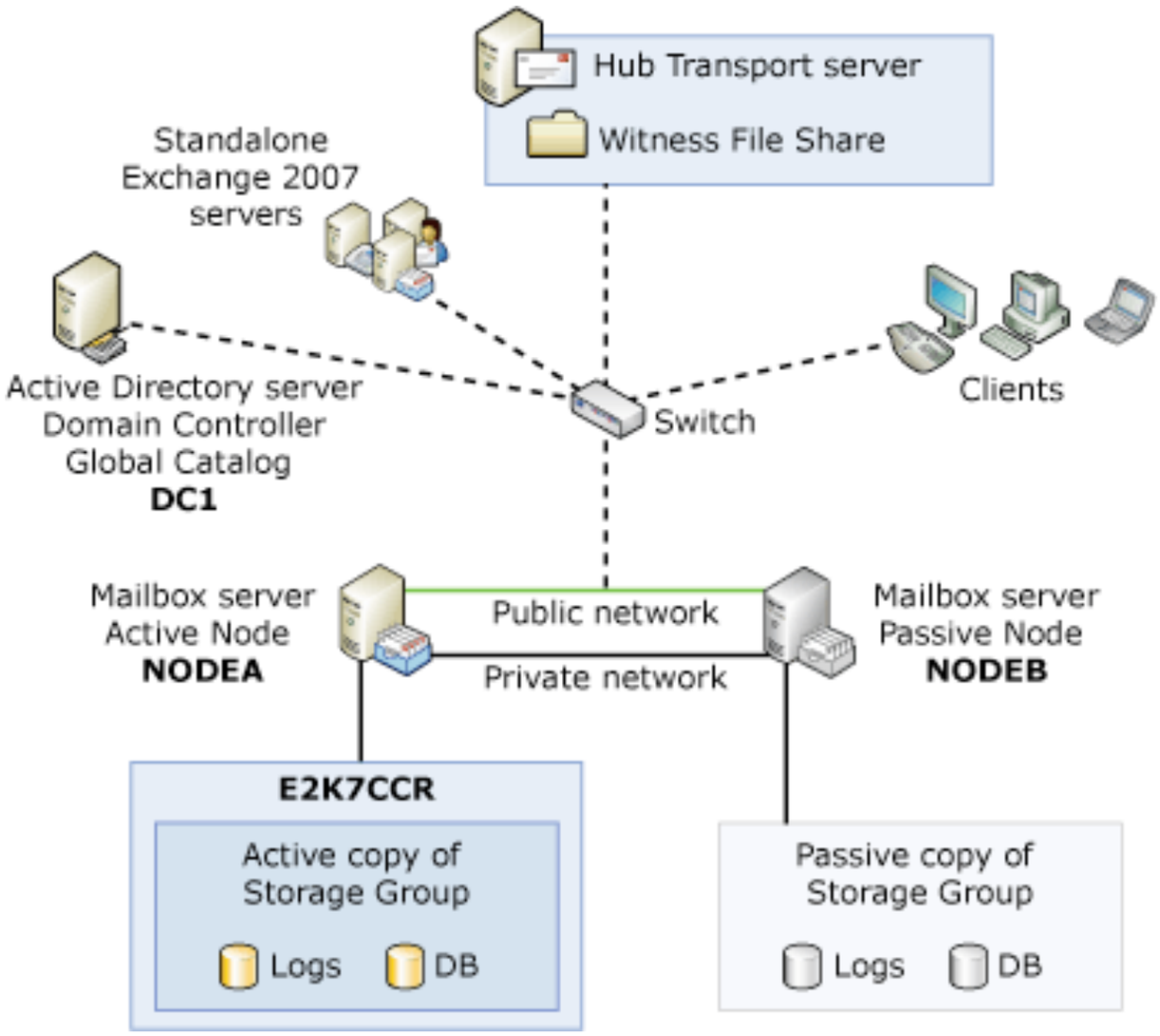
Main Site Back



Communication Lost
Split Brain



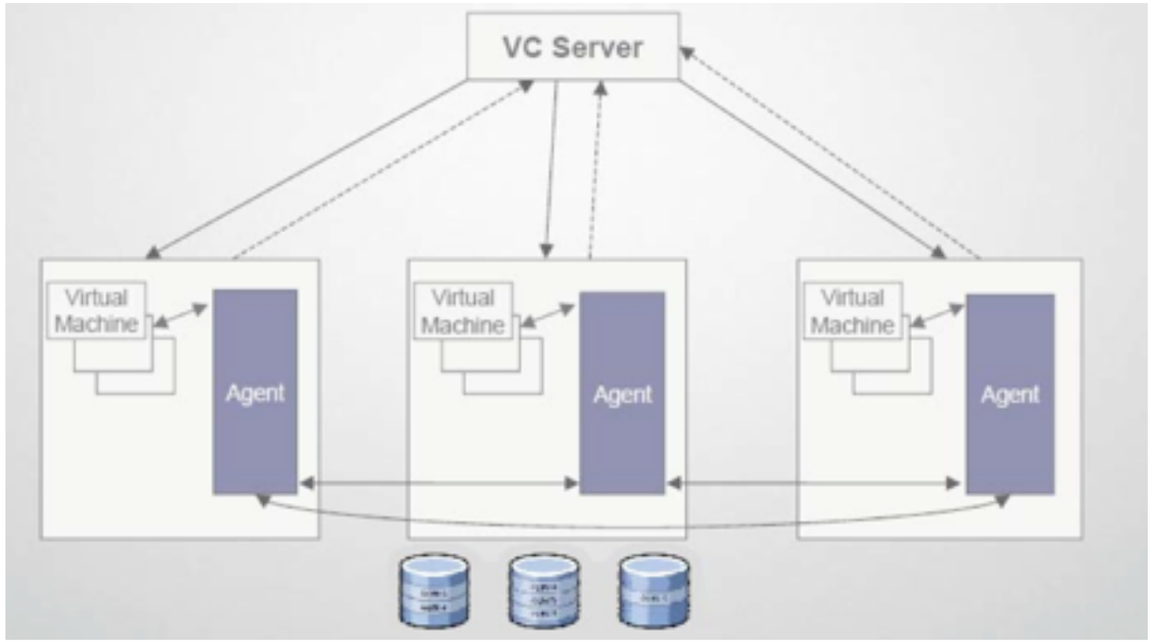
Example - Exchange 2007 Clustering



- Hub Transport Server allows messaging about the witness file share
- Witness share checked:
  - When a cluster node comes up and only one cluster node is available.
  - When a previously reachable node cannot be contacted.
  - When a cluster node leaves the cluster (release lock).
  - Periodically for validation purposes.

Example - VMWare HA Split Brain Situation

- Even number of hosts run virtual machine images, stored on iSCSI / NFS
- Virtual machine image is protected by file lock with timeout
- Single host running a VM looses overall network connectivity
  - With some configuration, VM will continue to run in this case
  - Other hosts restart the VM (due to lost external reachability)
- Primary host gets connectivity back
  - Takes back the virtual machine image file since it has the according processes
- Results in a ping-pong effect - solved by automatically shutting down the VM running on the primary host

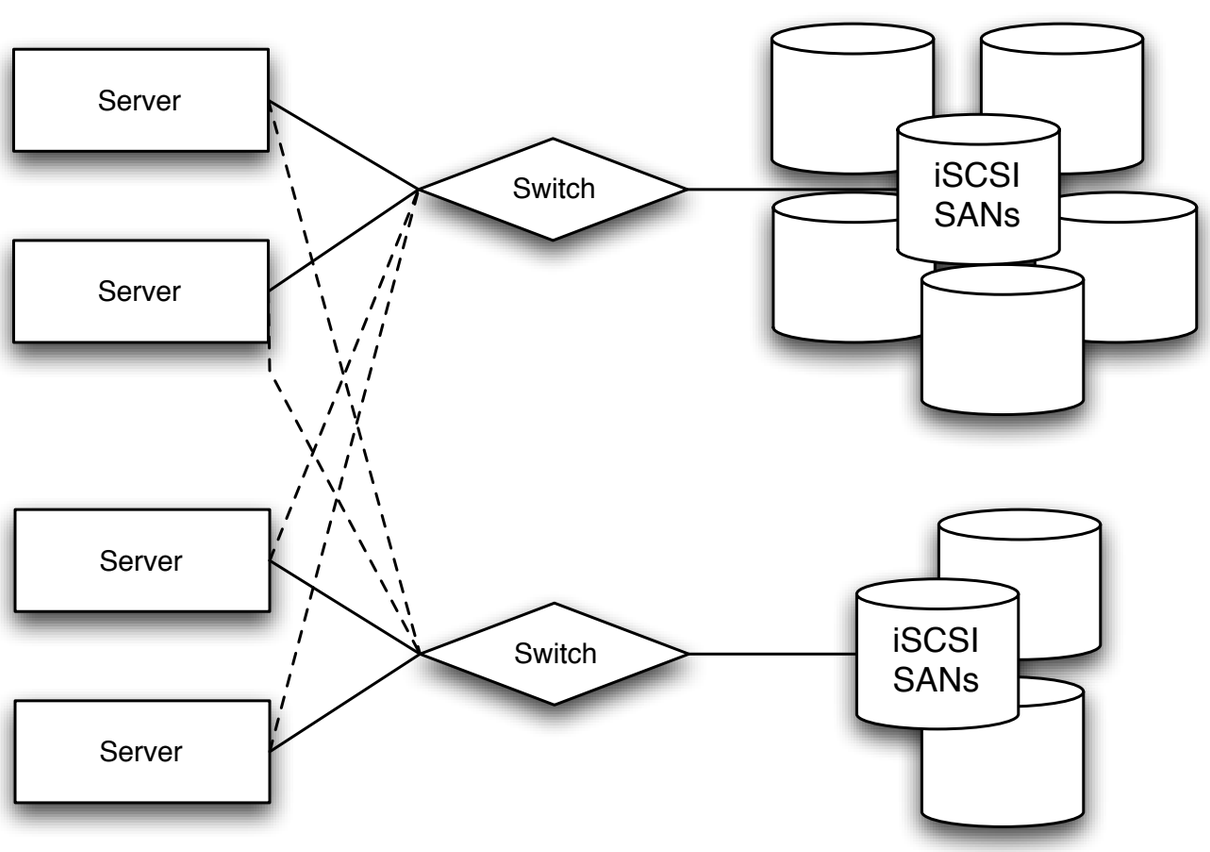


Checkpoint

- Avoid loss of results during recovery by saving global state information
  - Focus on long duration data that is hard to achieve
  - Checkpoint data consistency and checkpointing interval are relevant
- The „snapshot“ problem - how to achieve global consistency ?
  - Global state == local states + messages
  - Snapshot algorithms: Determine past, consistent, global state
  - Chandy & Lamport (1985) landmark paper
    - Relies on flushing principle of FIFO communication channels
    - Control messages ‚push out‘ pending messages

Remote Storage

- Storage location for checkpoints is relevant in failover / rollback case
  - Should not be the single point of failure
  - Pattern is good decision point for level of redundancy needed
- Real-world application: iSCSI Multi-Homing



Individuals Decide Timing / Data Reset

- **Individuals Decide Timing**
  - **Independent checkpoints**: Counter-approach to global checkpoints
    - Each process takes a **dynamic local snapshot** when it needs to
    - Consistency establishment overhead at recovery time vs. global checkpoint overhead during operation
- **Data Reset**
  - Recover from an uncorrectable data error by taking / computing initial values and approximate value
  - Relationship to *return to reference point* pattern - data reset is often a correlated activity

Error Mitigation Patterns

Overload Toolboxes

- Handle overload situation with too many requests for the system
- Each resource class needs dedicated overload treatment
  - Memory: Exhaustion hinders new request from entering the system
  - CPU: Overload slows overall processing down
    - Patterns: *Fresh work before stale, share the load, shed load*
  - Tangible resources: Processing demands exclusive system resources
    - Network ports, shared storage, devices, ...
    - Patterns: *Queue for resources, equitable resource allocation*
- Consider user demands
  - Patterns: *Fresh work before stale, finish work in progress*

Deferrable Work / Equitable Resource Allocation

- **Deferrable Work**
  - High load: Shed incoming work vs. shed routine maintenance workload
  - Make routine work (only relevant in error case) deferrable
- **Equitable Resource Allocation**
  - Scenario: Handling of many requests for a set of resources, some of them are rare
  - Request-level handling would render some resources unnecessarily idle
  - Solution: Pool similar requests, allocate resources to pools
  - Additional bookkeeping needed for managing the requests and their related resource demands
  - Might lead to priority-inversion scenario

Expansive / Protective Automatic Controls

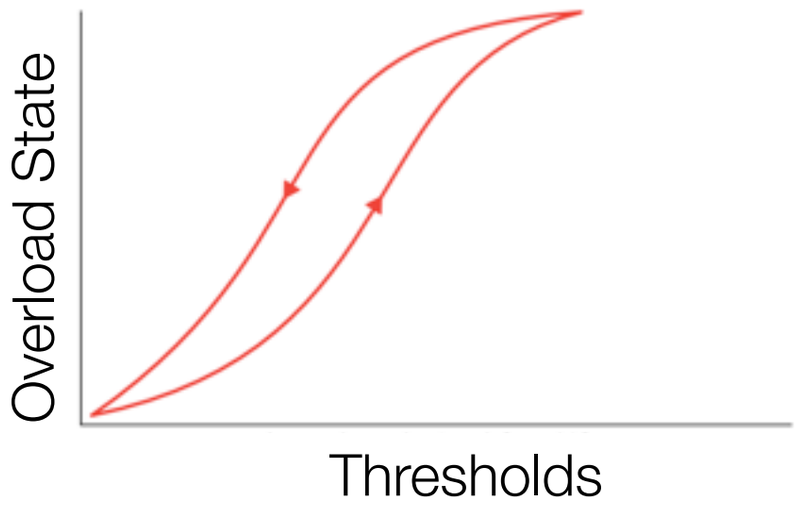
- **Expansive Automatic Controls**
  - Design some resources into the system only used in case of overload
    - Example: No 100% CPU utilization in normal operation of HA clusters
    - Example: Dynamic Offloaded Work - Cloud Computing
      - Increases request processing overhead, so take only as temporary solution
- **Protective Automatic Controls**
  - Overload options: Shed internal work, shed incoming load, do nothing
  - Put restrictions on how much work the system accepts while still functioning
  - System throughput can drop due to contention, but should not drop to zero

Shed Load

- Throw away a minority of requests to serve the majority
  - As early as possible, to minimize resource consumption
  - Rejection method to be considered, e.g. do not send acknowledgements
- Example: ICMP
  - Type 3: Destination Unreachable - not time-out on client host
  - Type 4: Source Quench - typically only between routers, also used by mail servers
  - Type 11: Time Exceeded - through congestion (or circular packets)
- Example: HTPP 5XX error codes
  - 503: Service Unavailable
- Specialized case: *Shed work at periphery*

Slow It Down

- Handle overload cases and avoid saturation by multi-step *escalation*
  - Restrict request processing with increasing severity per level
  - Goal: Slow things down until the system can catch up with the load
  - Feedback system, demands dedicated resources for the controller part
- Add **hysterisis effect** to prevent oscillation for level changes
  - Different trigger values to enter / leave and escalation level



Finish Work in Progress / Fresh Work Before Stale

- **Finish Work in Progress**
  - What to process, what to reject ?
    - Best case is labeling of requests: „new“ vs. „continuation“
    - ‚In progress‘ does not mean *queued for resources*
      - Can lead to oscillation when system is „starving“ for new requests after cleanup
        - Solution: Let small portion of new requests through
- **Fresh Work Before Stale**
  - If requester gives up, *final handler* and retry eats up more resources
  - Perform LIFO queue handling or non-queueing for premium requesters

Marked Data

- Data error detected, but no recovery option available, error mitigation is acceptable
- Data should be *quarantined* - do not use it, do not derive actions from it
- Example: IEEE ‚Not a Number' (NaN)
  - Result of division by zero, square root of -1, ...
  - IEEE 754-1985: Standard representation for binary floating point numbers
  - Rules for computation when operand is NaN - typically result is again NaN
    - Options: Assume default value, skip operation, mark result as erroneous

Fault Treatment Patterns

Let Sleeping Dogs Lie / Reintegration

- **Let Sleeping Dogs Lie**
  - Treating faults by system change can introduce new faults
    - Known latent fault: Risk of reoccurrence, damage assessment possible
    - Potential new fault: Additional risk of miss-applied correction, no damage assessment possible for accidentally added faults
- **Reintegration**
  - Different steps needed to reintegrate repaired component
    - Take off *riding over transient* and *isolation* lists
    - Watch new component for a while: **hardening / soaking / trailing**
    - Follow deterministic procedure, use as *standby* if possible

Reproducible Error / Small Patches / Revise Procedure

- **Reproducible Error**
  - Apply stimuli again under *quarantine* in order to prove fix
    - Can be automated (regression test)
    - Compare system output with *golden unit* output
- **Small Patches**
  - Design system update as small as possible
- **Revise Procedure**
  - When predetermined procedures contributed to failure duration, fix them

Root Cause Analysis

- Error was mitigated or processed - find the original cause in the *dependability chain*
- Peeling back the layers
- Specialized topic in quality methodologies
  - Must be performed systematically as an investigation
    - Establish sequence of events / timeline
- Example: 5 Whys
  - Originally developed by Toyota for manufacturing
  - Limit number of dive-in‘s to avoid tracing the chain of causality
- Many more approaches (later in the course)

Root Cause Analysis - 5 Whys [Six Sigma]

- The Washington Monument was disintegrating
  - Why? Use of harsh chemicals.
  - Why? To clean pigeon poop.
  - Why so many pigeons? They eat spiders and there are a lot of spiders at monument.
  - Why so many spiders? They eat gnats and lots of gnats at monument.
  - Why so many gnats? They are attracted to the light at dusk.
  - Solution: Turn on the lights at a later time.