

# Scalable vendor indifferent clusters for resilient application deployment.

an “educational rant” from Darren Starr [darrenst@uio.no](mailto:darrenst@uio.no)

# resiliency

## noun

re·sil·ien·cy ri-'zil-yən(t)-sē

## pluralresiliencies

1

: the ability of something to return to its original size and shape after being compressed or deformed : **RESILIENCE SENSE 1** Most of these rollers have to be replaced every few years when the rubber loses its *resiliency*.—Steve Rosenthal

Upholstery consisted of lightweight leather cushions; frequently there were springs beneath the cushions for added *resiliency*.—Daniel M. Mendelowitz

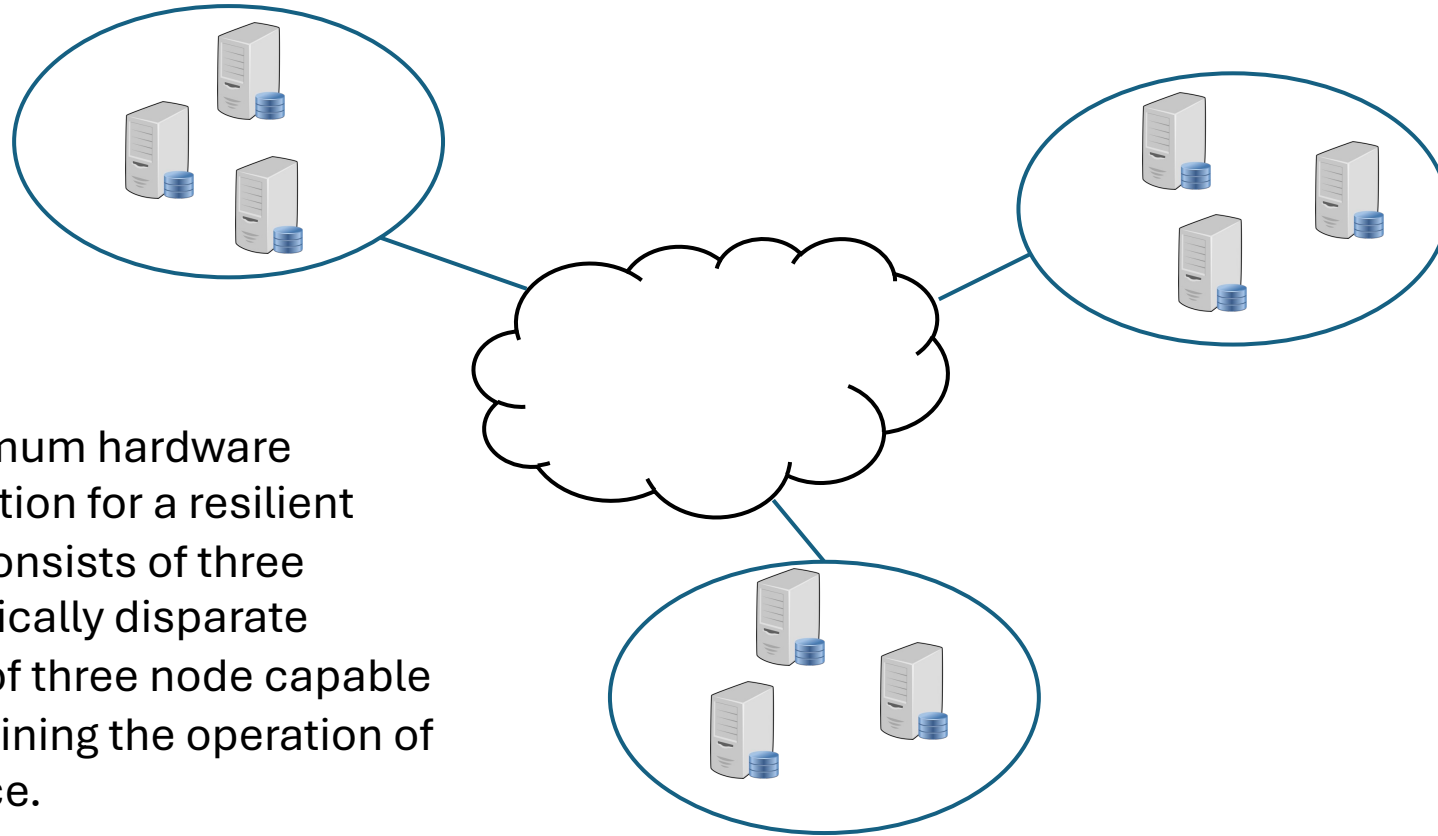
2

: an ability to recover from or adjust easily to adversity or change : **RESILIENCE SENSE 2**... the *resiliency* of nature and humans in the wake of fires, floods, earthquakes, volcanoes, and other destructive forces.—Richard K. Snow et al.

The Brewers have shown a *resiliency* through the season's ups and downs ...—*Associated Press*

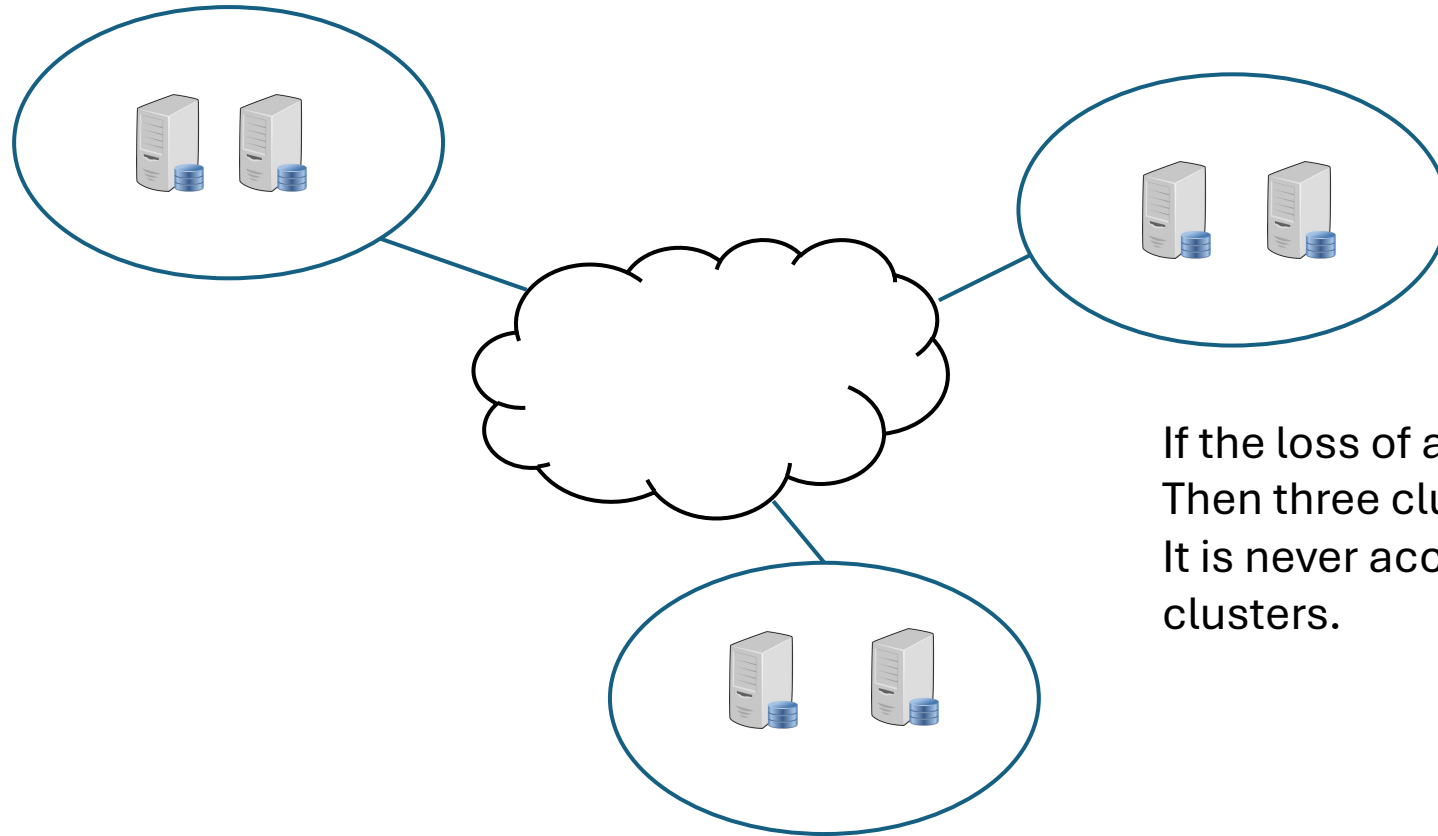
The project's goal is to develop ... cultivars with certain traits; for example, better climate *resiliency* and better tolerance for low water and low fertilizer conditions.—Julie Chao

# Darren's Rule of Threes



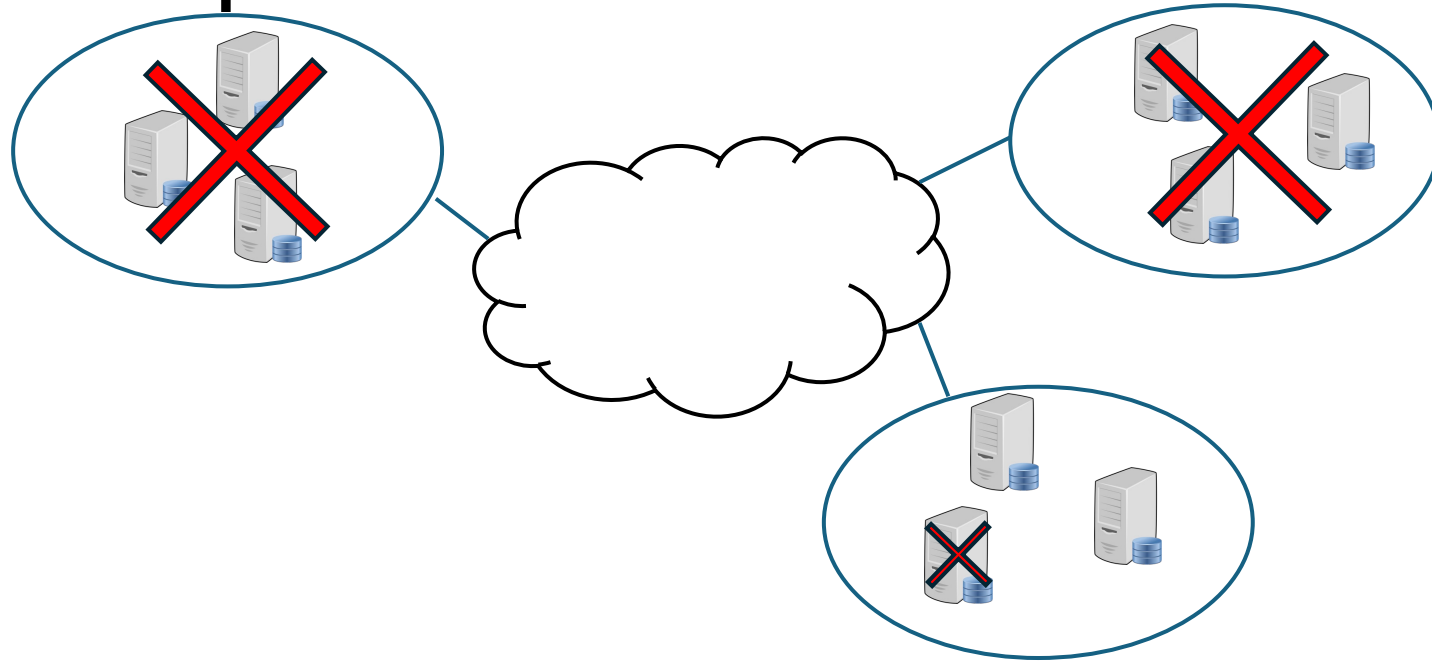
The minimum hardware configuration for a resilient service consists of three geographically disparate clusters of three node capable of maintaining the operation of the service.

# Darren's Theory of Threes and Twos



If the loss of a data cluster is deemed acceptable,  
Then three clusters of two may suffice.  
It is never acceptable to have less than three  
clusters.

# System requirements minimum



- Even in a cluster of 3x3, it is mandatory that there is enough system resources to operate the service where two clusters are inoperable and one node in the remaining cluster is unavailable.

# What we lost

- Everything is an object.
- This was a core principle of all things resilient.
- Systems built on legacy mainframe technology depended on location independence which allowed all services and data to be available everywhere.



# Where we went wrong

- Mainframes were highly redundant systems with no single point of failure.
- PCs were supposed to cut the cord and let us run software by just sticking a floppy into the computer and we wouldn't even need to dial into a mainframe.
- PCs were too small, too slow, too ... meh

## Some of our biggest ideas aren't very big.



Datamaster

### Small Systems from IBM.

In business, it's often the little things that count. And you can count on IBM for a lot of them.

That is, we make a wide range of low-cost, easy-to-use small computers to help businesses of all types and sizes solve big problems.

Like inventory control, for instance. IBM small systems can tell you what's what with your widgets: How many you have on hand. How many you need to fill existing orders. Where to ship and when.

If you want to improve customer

service, we're at your service. With small systems that help your sales force match orders to inventory, figure a price and guarantee a delivery date. (All of which helps guarantee a sale.)

IBM small systems can help your marketing manager make price changes quickly. And your billing department improve cash flow by invoicing customers more efficiently.

When you write your customers, you can get letters and reports out faster.

But that's just the beginning. IBM has office



Personal Computer

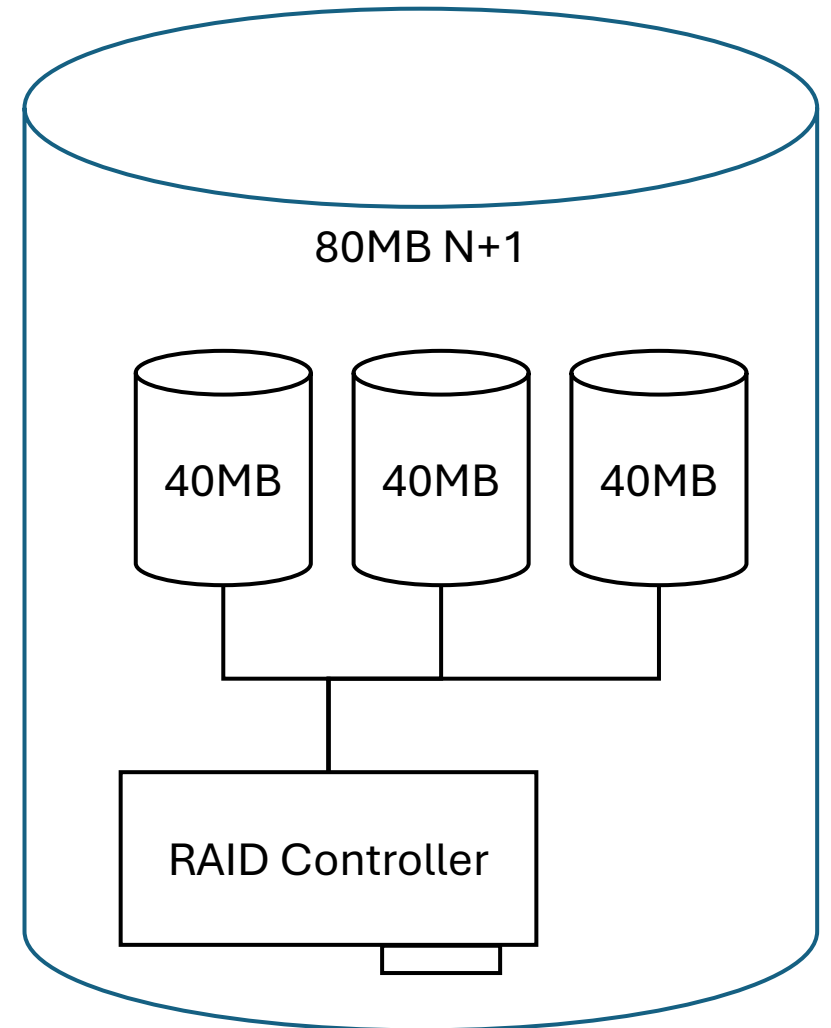
# Centralization

- PCs used a single processor to handle every task
- File systems moved into software
- Systems greatly increased in capacity to compensate
- Storage became a complete mess



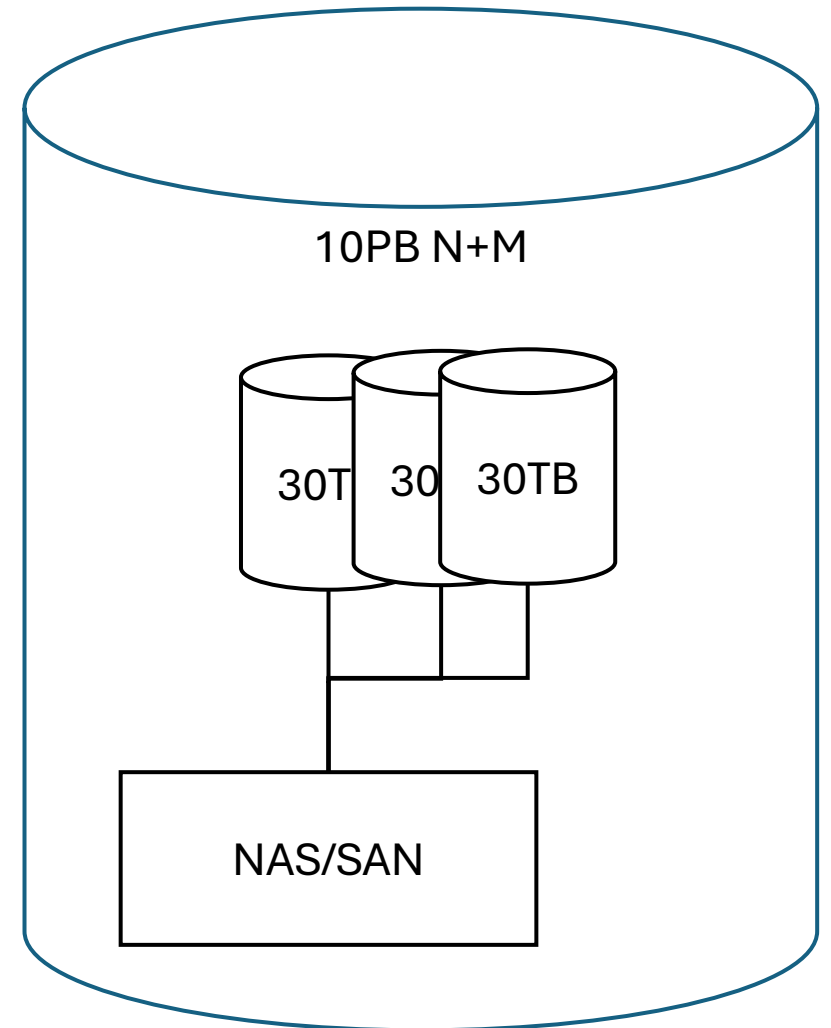
# Hard drives were small

- Everything is a file locked files into specific drives
- Larger storage needs meant needing bigger drives
- Drives were small and unreliable
- We introduced RAID



# Drives grew... count and size

- Servers still needed small storage
- Storage became plentiful
- A few drives could provide enough storage for a lot of servers
- We partitioned the storage into virtual volumes
- Storage became infinitely more complex.



# Drives got bigger, but not faster

- Server storage performance requirements grew massively
- A few drives per machine could barely provide the needed IOPS
- A few drives shared across multiple machines couldn't keep up
- Storage systems went from costing thousands of dollars to hundreds of thousands of dollars.



# Virtualization ruined the server world

- Virtualization promised lower costs by replacing 10-20 servers that each cost under a thousand dollars with a server that cost \$5000.
- It promised we could centralize everything and reduce operating costs by doing so.
- A minimum VMware resilient design costs \$150,000 in 2025
- A proper virtualized infrastructure requires network engineers, storage engineers, virtualization engineers and of course general IT.
- Virtualized IT costs are far more expensive than bare metal by 1-2 orders of magnitude.



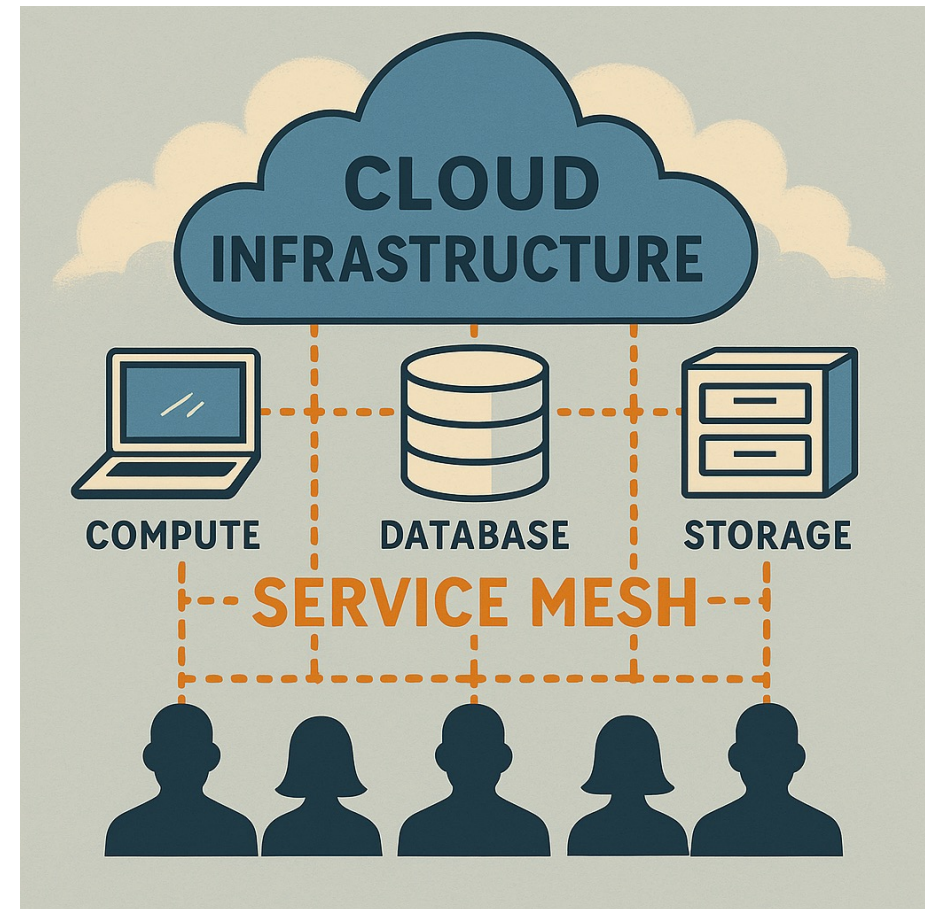
# Cloud saved everything

- A well engineered cloud environment provides storage, compute, networking and all without the need for hiring experts in each or replacing hardware every few years for no reason.
- Using PaaS or FaaS with high performance, highly redundant global databases (SQL and NoSQL), companies can use very little resources at the cost of vendor lock-in
- Due to financial market fluctuation and general corporate profiteering, costs of cloud are impossible to predict.
- Lock-in is solid, there is no way out.



# Cloud IS BETTER, just not public

- Cloud is just better
- Storage is cheap
- Compute is cheap
- Networking is cheap
- It takes good engineers
- It takes planning.



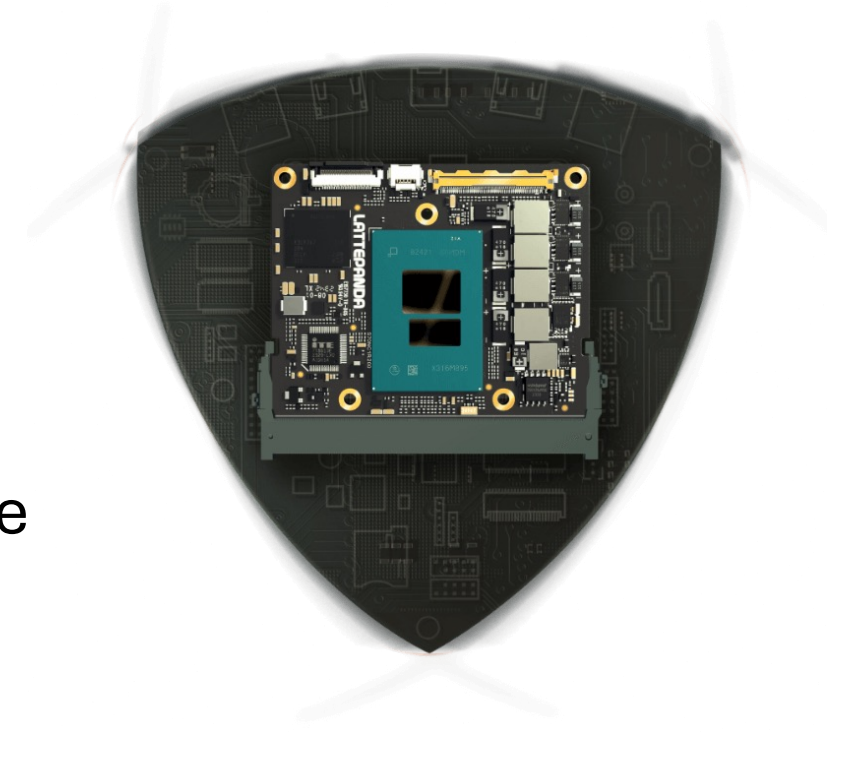
# The hardware

- We DON'T need resilient servers
- We need resilient infrastructure
- We DON'T need massive compute
- We need lots of small computers
- Enterprise drives are still better than consumer.



# Pi Scale Computing

- Servers work too, but why bother?
- Small \$100-\$500 servers can easily do the job
- Everything can be run from PoE meaning a single cable to each node.
- Debian, Ubuntu and more run on these servers.
- Options
  - Raspberry Pi 5
  - Orange Pi 5 Plus
  - Rock Pi
  - LattePanda



# Disposable Servers

- When a node fails, toss it in a box and replace it.
- If you can't fix the node later on, recycle it
- You can overprovision a rack with extra nodes of  $N+M$
- When there is no more  $M$ , service the rack
- Ignore errors until then.



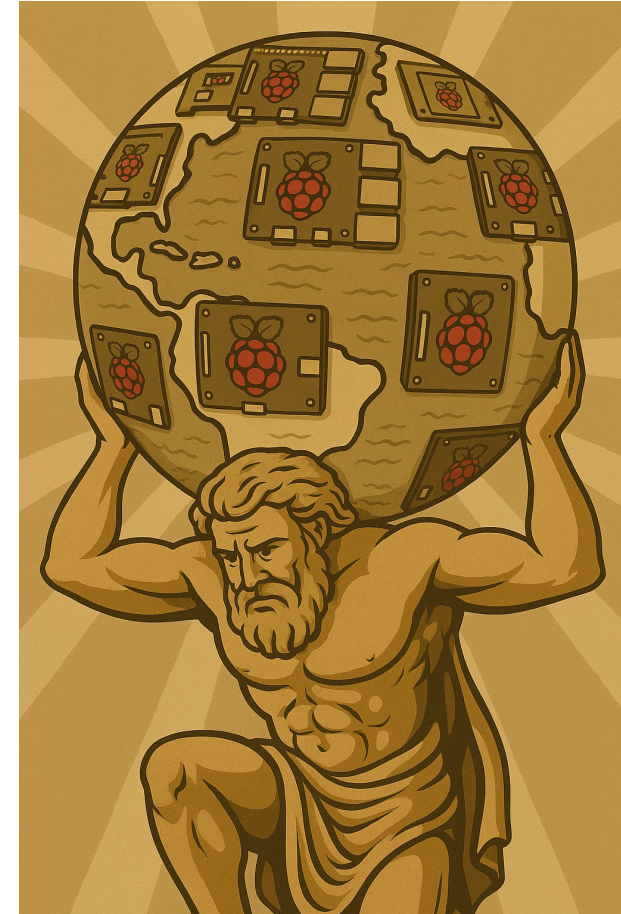
# Everything is cheap

- You don't need 10Gb Ethernet and multiple connections to support cloud storage
- You don't need super capacitors larger than what's on the drives themselves.
- You don't need managed switches (they help though)



# Exascale

- Maybe an exaggeration for now
- Adding more nodes adds more performance
- Why buy 1 when you can buy 30?



# Rip and Replace – Never Fix

- All systems leave trash behind
- It takes less time to replace a system than repair it
- We repair the scripts, not the installation.
- Maintenance windows should completely reinstall all systems.
- Automation tools are everything.



# Operating System

- Linux is at the heart of it all
- Declarative distributions allow full configuration at build time eliminating the need for configuration through Ansible for example
- Boards that can network boot allow reinstalling the operating system on every boot.
- The entire operating system can be managed via CI/CD



# Operating System Security

- Declarative operating systems can't be tampered with
- There is no ability to write to the boot storages as they are static images
- If you can't change stuff, neither can the bad guys.



# Container based everything

- Kubernetes is a beast and a hog
- It has become the defacto standard for more or less everything.
- Kubernetes itself is a a facilitator for a framework of components that are built and hardened by external providers.
- K8s is a disgusting mess that grows worse every day
- Long live K8s.



# Down with virtual machines

- Virtual machines use too many resources
- They aren't maintainable
- They shift the constant IT nightmare to a virtualized environment.
- Changes are rarely reproducible
- They perform quite poorly



# Containers are maintainable.

- We rapidly reproduce container environments.
- Everything is declarative
- All real work can be done before deployment
- K8s is elastic
- K8s is “efficient”
- K8s is “resilient”
- K8s is a pain in the ass if the controllers fail.
- Overall declarative linux and k8s is a DR dream.



# Rollback is critical

- Easy deployment is always good
- Easy rollback is better
- Being able to revert an entire system with little or no data loss is ideal.



# Container-based infrastructures are alive

- Given the declarative nature of container-based infrastructures, the “deployment” declarations can allow changes to be deployed in production.
- Updates to the deployment remove what shouldn't be there
- They also remove what should but replace it with new stuff.



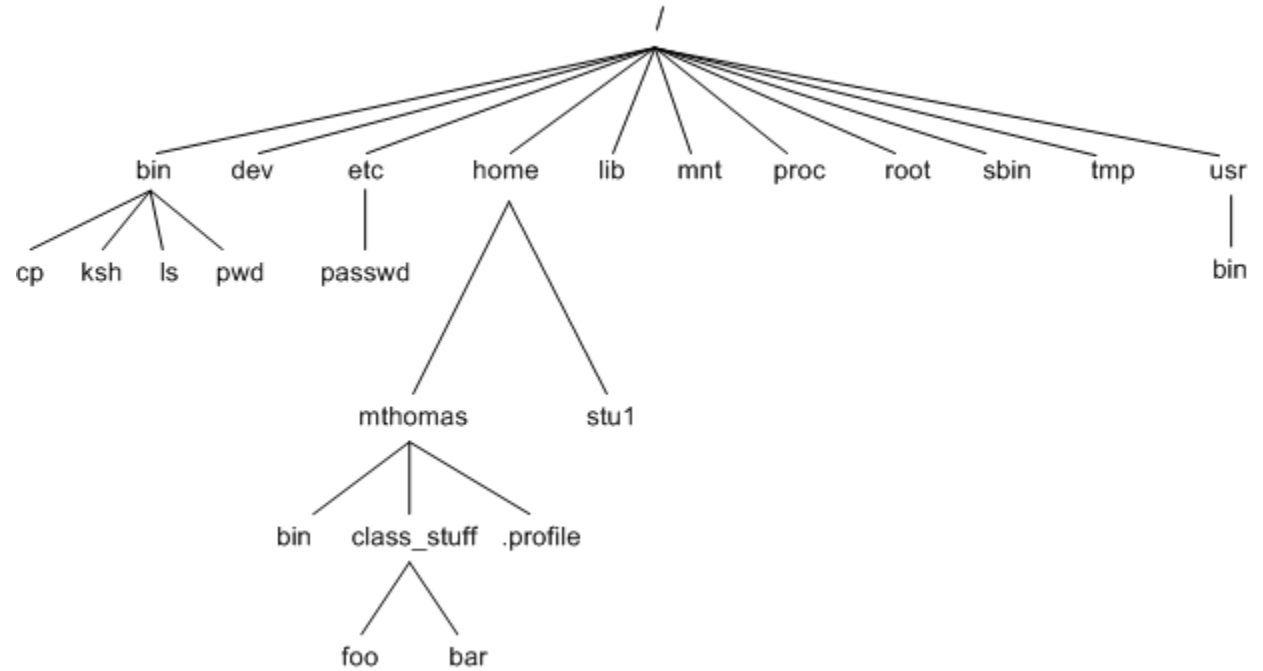
# Everything is an object



- Storing data on a mainframe meant storing an object.
- An object is basically a database entry.
- Can be a S3 style file object tied to a location independent identifier (kinda like an inode)
- Can be a record in an object database.

# Everything is a file is not an alternative

- These are two totally different paradigms.
- One approaches storage of files in specific locations
- One approaches storage of information as objects.



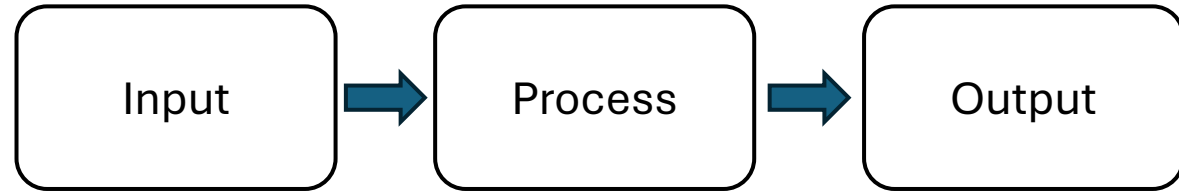
# The real battle – Application vs Transaction System



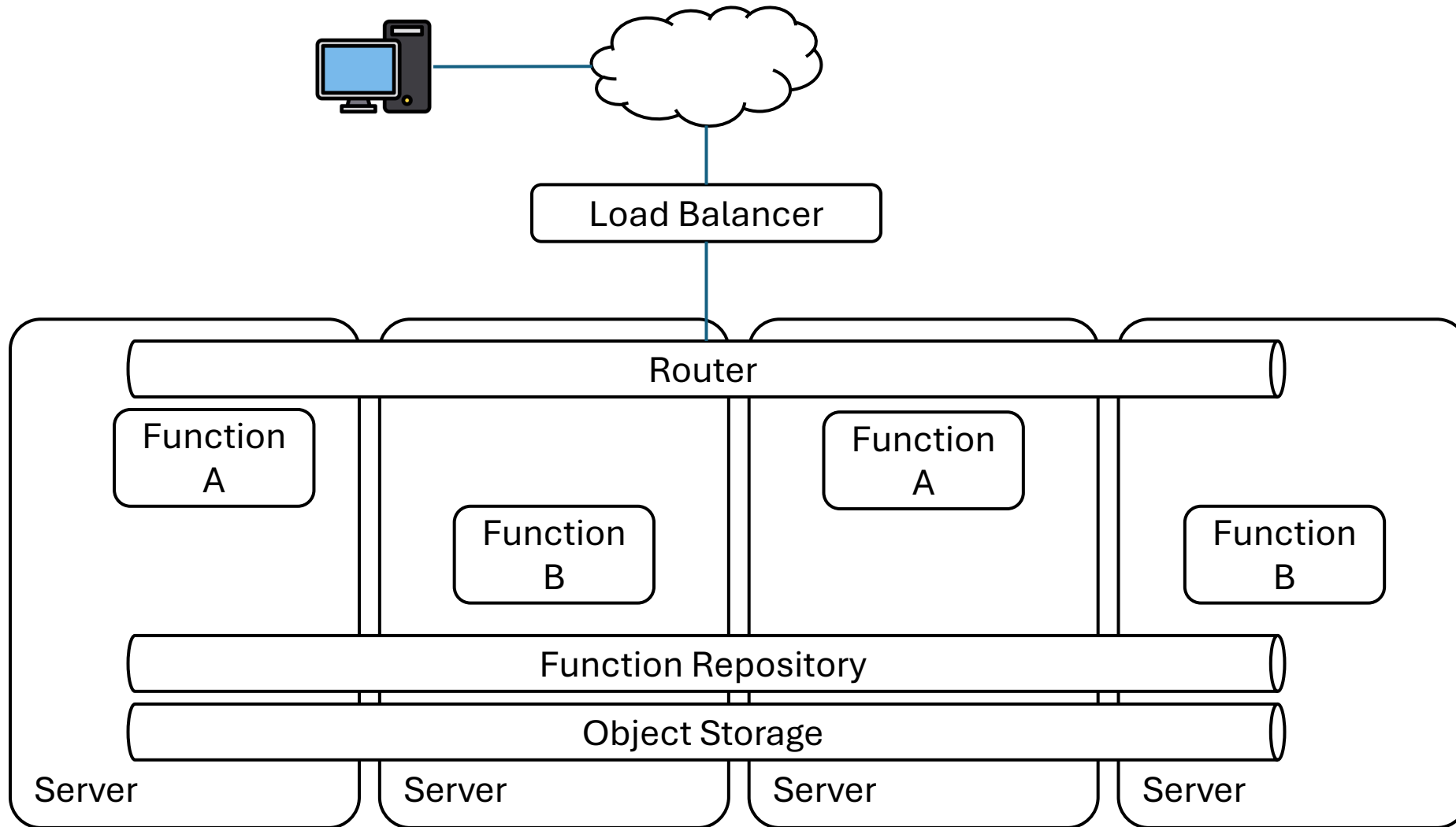
- Programs are long running tools. Things like SSH sessions
- Transactions are like REST APIs where a function is called, processes something and returns a result as an atomic item.

# What is a transaction

- A transaction is a program which receives input.
- Acts upon it
- Provides output
- It's a REST API



# On-Line Transaction Processing System OLTP



# Everything is a transaction

- Transactions can be triggered by routed user events
- They can also be triggered by scheduled events
- The code running the transactions can be unloaded when done.

# How does it scale?

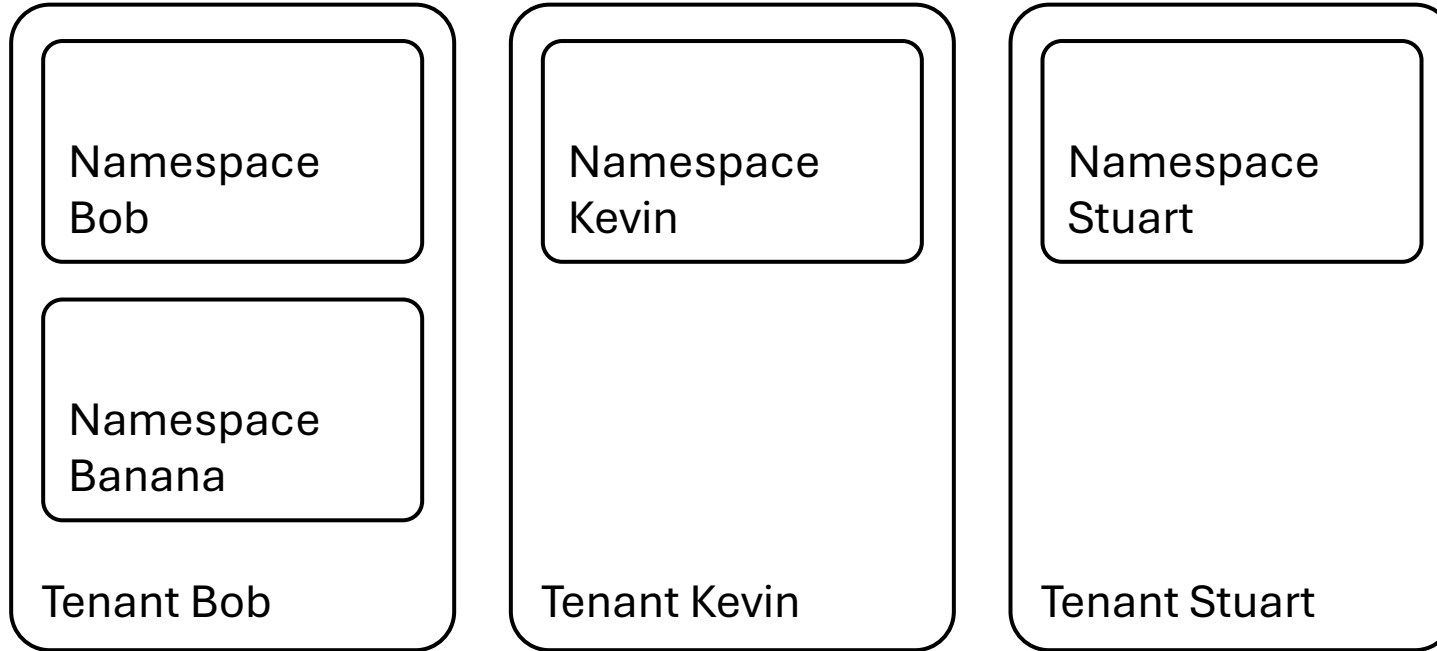


- All storage understood what an object is
- Whether in RAM, disk, tape, even punch card, data could be read and written by using a system wide object identifier.
- Each CPU in the system shown is about 10Mhz and 10MB of RAM.
- Four CPUs could operate hundreds of bank branches, thousands of minibanks, millions of paper checks being processed, millions of bank-to-bank transfers, hundreds of thousands of pages printed

# Requirements for a OLTP Cluster

- Resilience Five-Nines uptime minimum.
- Easy scalability
- Location independence
- Multitenancy
- Online upgrades
- Easy disaster recovery

# Multitenancy



- Namespaces are the key
- While some services are replication (ingress and routing), most services are entirely isolated between namespaces
- Users can belong to one or more namespace.

# Schema-less vs Schema

- NoSQL or document databases allow free form schemas
- Schema-less development can be tricky as garbage collection is not always possible
- Schema based struggles with rolling back. New data follows new schemas. Older versions of the software may fail to be compatible.
- Before moving forward, a rollback strategy should be formulated.
- Schema based is easier to unit and integration test for rollbacks
- Schema-less can easily leave trash behind.

# Load Balancer



- Always knows the state of each server
- Spreads the work loads to the nearest cluster and nearest server
- Operates at an IP/IPv6 level (l3, internetworking layer)

# Home of the code



- Programmers don't upload to the cluster.
- They check-in to GIT and register the functions with the cluster which pulls the code from the git, compiles it, then runs it.

# Router



**NGINX Ingress Controller®**

- Routes incoming transactions to services based on their URLs

# Certificate Management



- Applies for certificates that ties ingress controllers to certificates for automatic generation and renewal.

# Serverless Function management



**Knative**

- Allows registration of functions to URLs
- Checks out the latest versions of the functions and deploys them to nodes within a cluster.
- Elastically scales capacity based on load, location and other metrics.
- Lets developers write transactions rather than full systems
- Atomic upgrades of functionality.

# Static content delivery



- Delivering static pages can be done easily using a standard web server instance.
- Another solution is to make a function which reads the page from the object storage.

# Service Mesh



- Visible, secure, scalable, reliable communication via the mesh
- No components talk directly with other components.
- All traffic is routed through a highly monitorable, securable and visible mesh

# Message Passing



- Passing messages between services is fine.
- Not common in a transaction processing system
- Great for real-time communication

# Memory caching

## Valkey

- As data, services, and everything else are location independent, session variables need a place to be stored.
- Valkey (a successor/alternative to Redis) provides high performance in-memory object storage

# Table storage



- Distributed database storage for traditional PostgreSQL databases

# Object Storage



- NoSQL database for object storage.
- Great for objects under 1MB in size
- Suitable for larger objects as well
- Fully querably
- Ability to act upon structured (JSON) and unstructured data  
“server-side”

# File/Object Storage



- For large scale file storage
- Provides and S3 API for full distributed (similar to CERN dCache) data.
- Great for massive image storage
- Great for AI training data
- File sizes don't matter.
- Pretty good for data providence.

# Software Defined Networking



- Let important as service meshes deal with most inter-service communication
- Provides full multitenant networking
- Integrates with external networking systems via BGP
- Makes IP addressing entirely irrelevant
- Can do IPv4 or IPv6, but not relevant in a containers environment.

# Graph based monitoring



Grafana



Prometheus

- For graphic counter based information such as bandwidth, CPU, memory consumption and more.

# Text based monitoring



- For storing and centralizing system logs.

# Bonus Section – Non-OLTP

- Not everything is a database or application
- Sometimes we need a jump host for example
- SSH can be a long running transaction
- Systems like Guacamole can be used but are not ideal

# Jumphost example