THE 1000X RULE:

SCALABILITY DESIGN AT INTERNET SCALE

Justin Stottlemyer - Fellow, Storage Architecture Shutterfly

Who am I

- Fellow, Storage Architecture – Shutterfly 2010
- Technical Advisory Board – MaxiScale 2008
- Senior Manager Architecture – Rearden Commerce
- Infrastructure Architect – eBay
- Sr Operations Engineer – Facebook 2005
- Principal Engineer – Paypal
- Infrastructure Architect – eBay / Paypal
- Principal Engineer – eBay 2000

Why is designing for scale difficult ?

- First one there
- Lack of experience
- Experience can lead you astray
- Lack of Multi-Discipline Expertise
- No Functional Cross Team Communication
- Real world VS Lab Conditions
    1. Everything works in the Lab.
    2. Everything breaks in the real world.

Obstacles to Scalable Design

- Perception: It's difficult to imagine scalable systems as a whole.
  - **<u>Reality</u>**:
  - Breaking systems down to their separate components allows you to see where each will break as you scale it.

- Perception: Transitioning to a new technology is risky
  - **<u>Reality</u>**:
  - With the right architecture, you can minimize risks
  - Doing nothing guaranties failure.

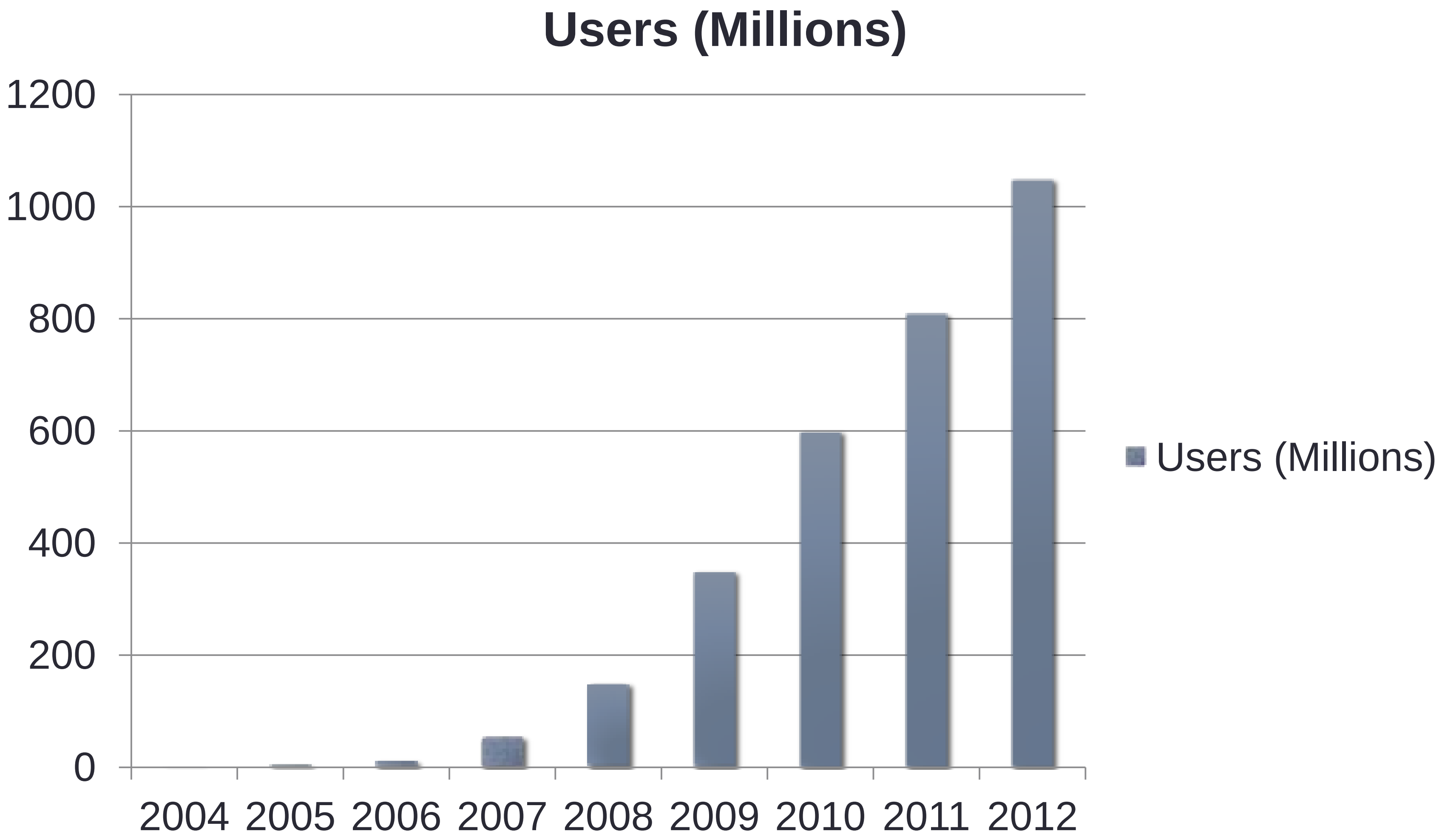
Why not 10X or 100X

- Typical 10X thinking can leave you with an existing design 10 times larger than your current one with the same cost and architectural issues.

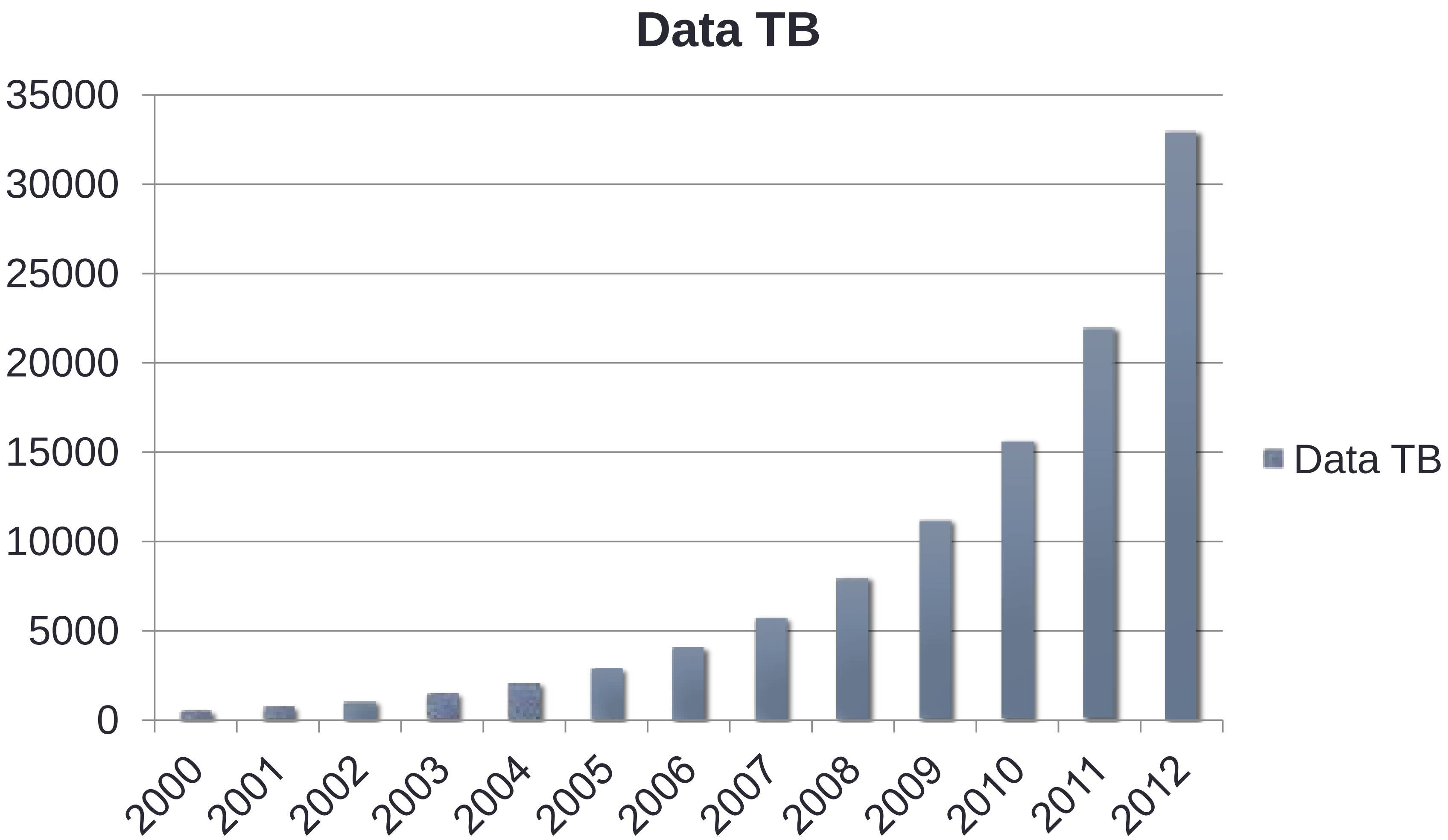
- 100X May not expose enough issues or scale for long enough.

- It takes time, effort, money to design/build/implement a new architecture, so it should last you longer than it takes to implement.

Why design for 1000x ?



Why design for 1000x ?



Storage is Central to Shutterfly's Business Model

- The Shutterfly promise: Free UNLIMITED online photo storage – FOREVER
- Shutterfly's storage situation in 2009:
  - 19 petabytes and rapidly growing
  - Growing image sizes

Keys to Successful Design

- Typically people only look at one or two dimensions, then decide between a handful of well known vendors.
- In other words $/IOP(S) and $/Gb are not the only two things to think about.
- Storage (and Application) Architecture should be taken into consideration from an end to end approach.
- Solve your problem(s)
- Design for the system to run broken (Fail in Place).

Why Move Away from RAID?

- RAID stops scaling at the Multi-Petabyte level
  - Unsustainable rebuild/recovery times
    - As drive sizes become larger, time to get back to fully redundant system increases
    - With 2,3 and 4 TB drives, time to generate parity grew from days to weeks
  - Availability / Integrity issues
    - Bad user experience/site downtime = revenue loss
    - Does it scale? For how long? Can you account for 5T Drives? 8T? 10T?
    - System Admins constantly firefighting
    - Poor internal perception of Storage team
- Asked to drastically lower costs by 66%
  - Already managing 19 PB with 3 System Admins
  - Already had industry leading pricing with multiple vendors

Data Resilience At Scale

- Object Store && Distributed Erasure Coding

  Cons:

  1. Increased Latency (first byte)
  2. Increased Metadata reliance. (In theory)
  3. Conversion from Legacy Systems

  Pros: Unparalleled

  1. Data reliability (bytes out = bytes in)
  2. Data availability (Uptime)
  3. Data resiliency (Protection from Data Loss)

  - All done with less hardware

- Checksums at every level
  - Application Level ; Scatter Gather Layer ; Local Storage Node ; Disk Layer

Storage Design Considerations

- How does the application use the storage? How does the user use the application?
  - Large block? Small block? Random? Sequential? Super Compute? B2B? B2C? Peaky traffic? Weekly, Monthly, Seasonal patterns? Transactional?
- Data Integrity Model
  - Does it scale? For how long? Can you account for 8T drives? 10T?
- Reliability / Availability / Resiliency Model
  - Can you upgrade in place with 0 downtime?
  - How long before you lose data? (MTDDL)

Architecture Design Considerations

- Stability
- Performance
- Scalability
- Modularity
- Sharding Model (Scale Up/Out)
- Failure Domain
- Maintainability
- Implementation

Performance Considerations

- Data locality (Application, Disk, Memory, Switch)
- Caching (Application, Disk, Directory, OS, Web)
- Avoid Double buffering
- Performance when degraded
- Be Asynchronous
- Be Atomic when needed
- End to End Design
- Measure Bottlenecks
- Be efficient at every level

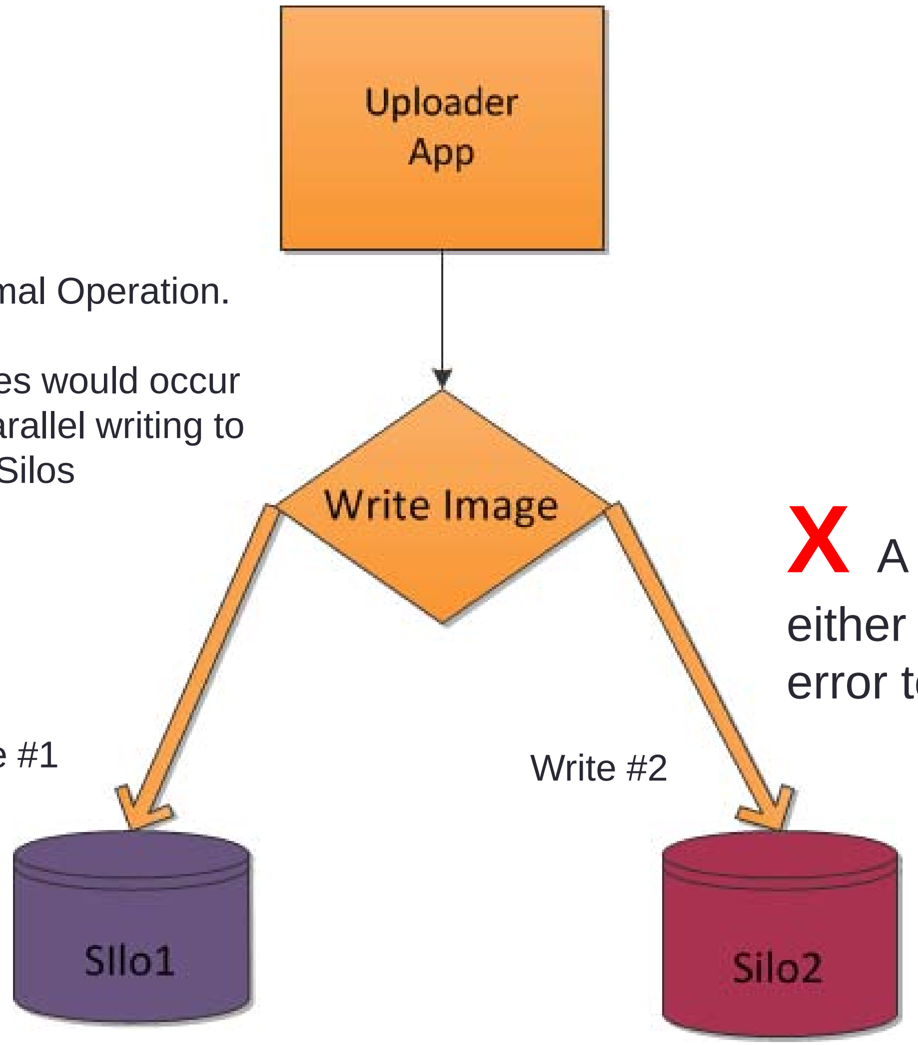
Implementing Scalable Storage

1. Implement new architecture to take advantage of storage technology innovations (including object storage) with long term scalability
2. Update metadata system to support object store
3. Guarantee metadata consistency
4. Complete application integration and testing

Old Architecture

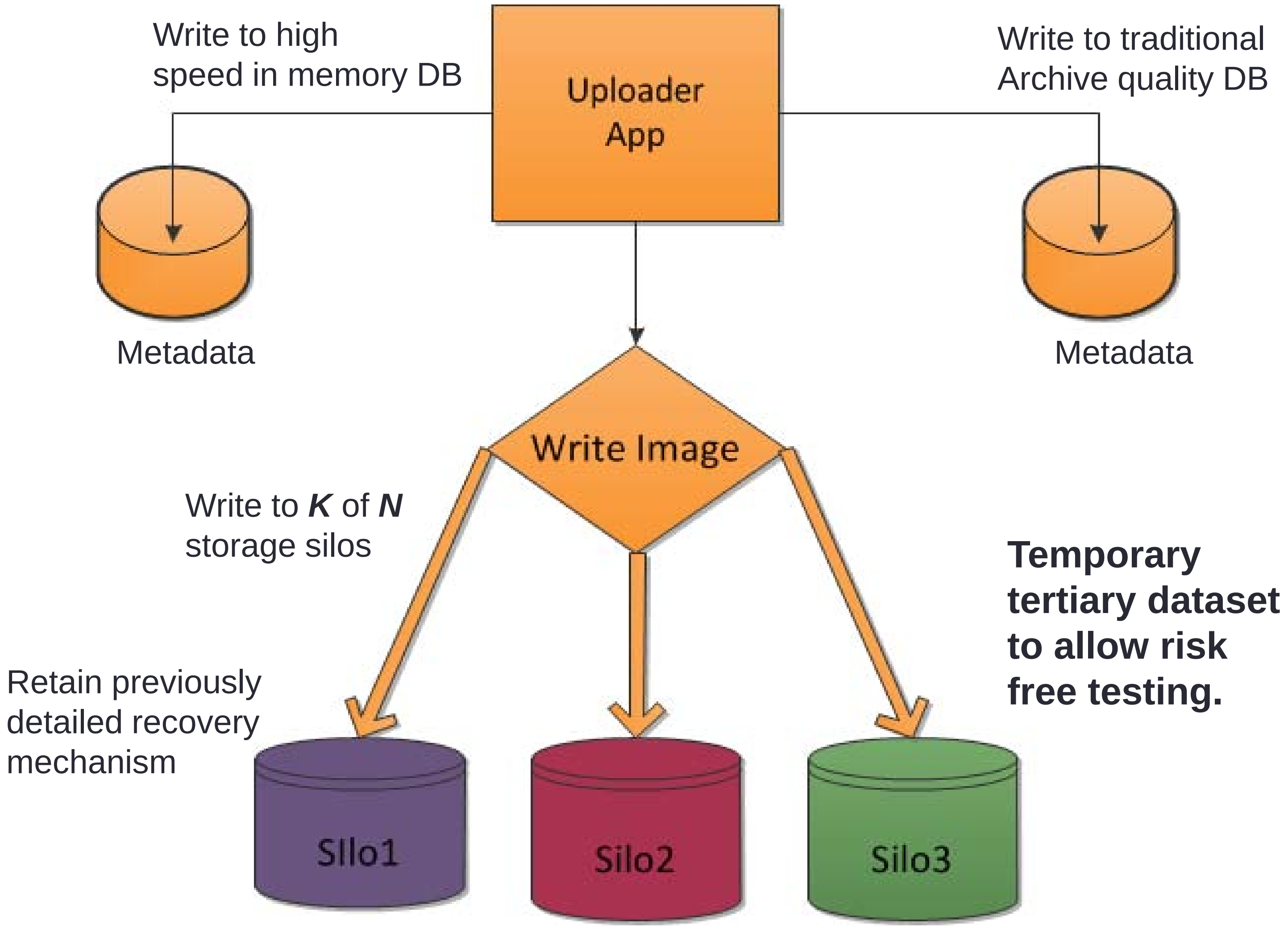
Normal Operation.

Writes would occur in parallel writing to two Silos

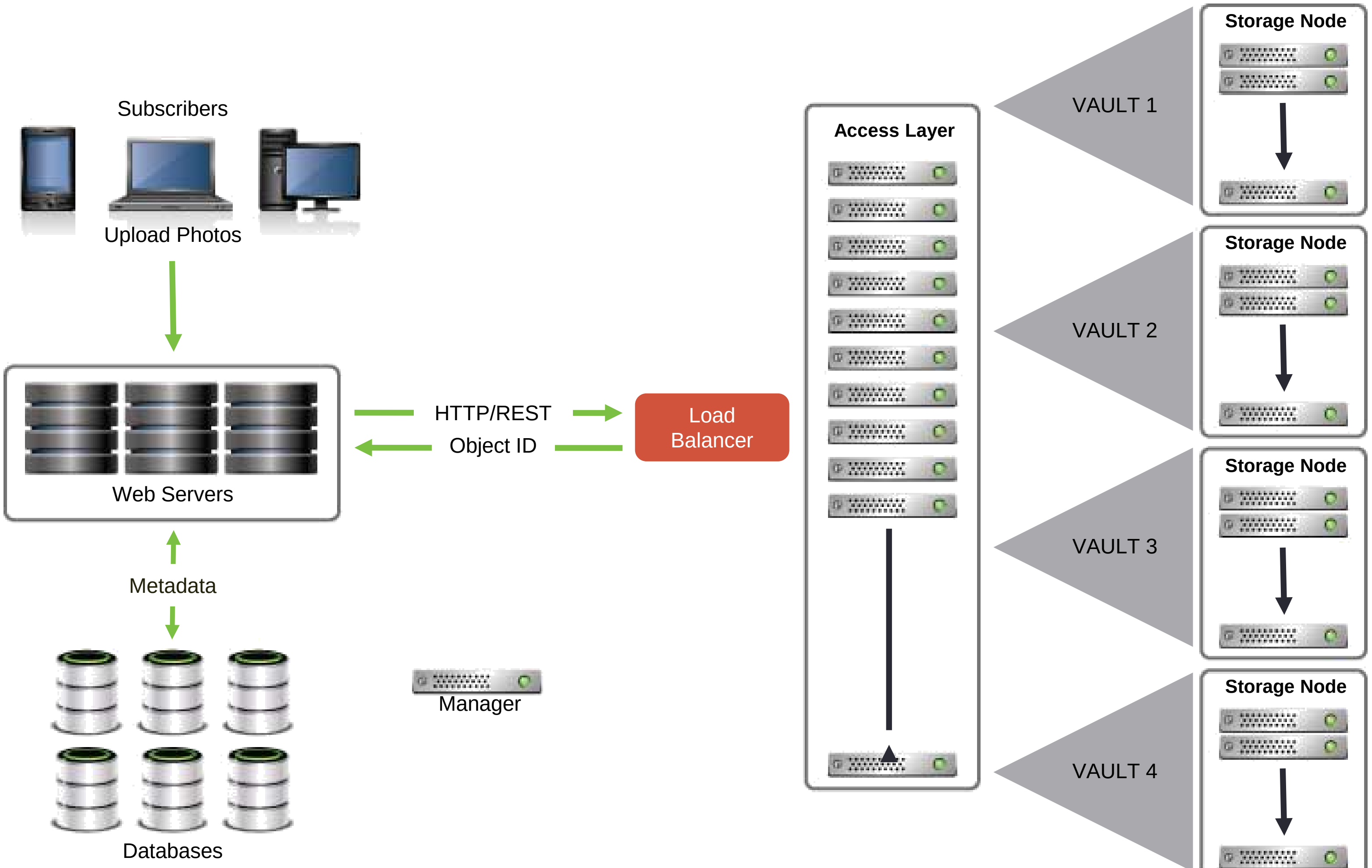


X A failure writing to either silo results in an error to the user

New Architecture



Object Storage Workflow



EOL Old Gear & Data moves

- How to EOL old equipment?
  - Relying on Vendor Strategy
  - Drop in newer replacement gear or drives (let it heal)
  - Home Grown

- How to move mass amounts of Data.
- Mass parallelism
- Consistency Checkpoints
- Sharding Strategy

- Kodak Data migration in 82 days
- 5.5 Billion images
- 8.8 PB

Results of Transition to Object Storage

- Object storage system has been in full production for more than two years
  - Primary archive for more than 18 billion images– over 80PB and growing
- No system downtime or data loss despite upgrading, extending and physically moving the system.
- Storage costs reduced by more than 60%
- Future-proof storage that can easily scale to exabytes

Key Takeaways

- Migrating architectures from traditional systems is doable.
- Plan for scale, fail in place, and resilience.
- Planning for Flexibility in the application architecture allows you to test and implement new technology with little to no risk.
- Take big risks intelligently.
- Designing a system for 1000X scalability doesn’t mean you need to buy it or build it all today.