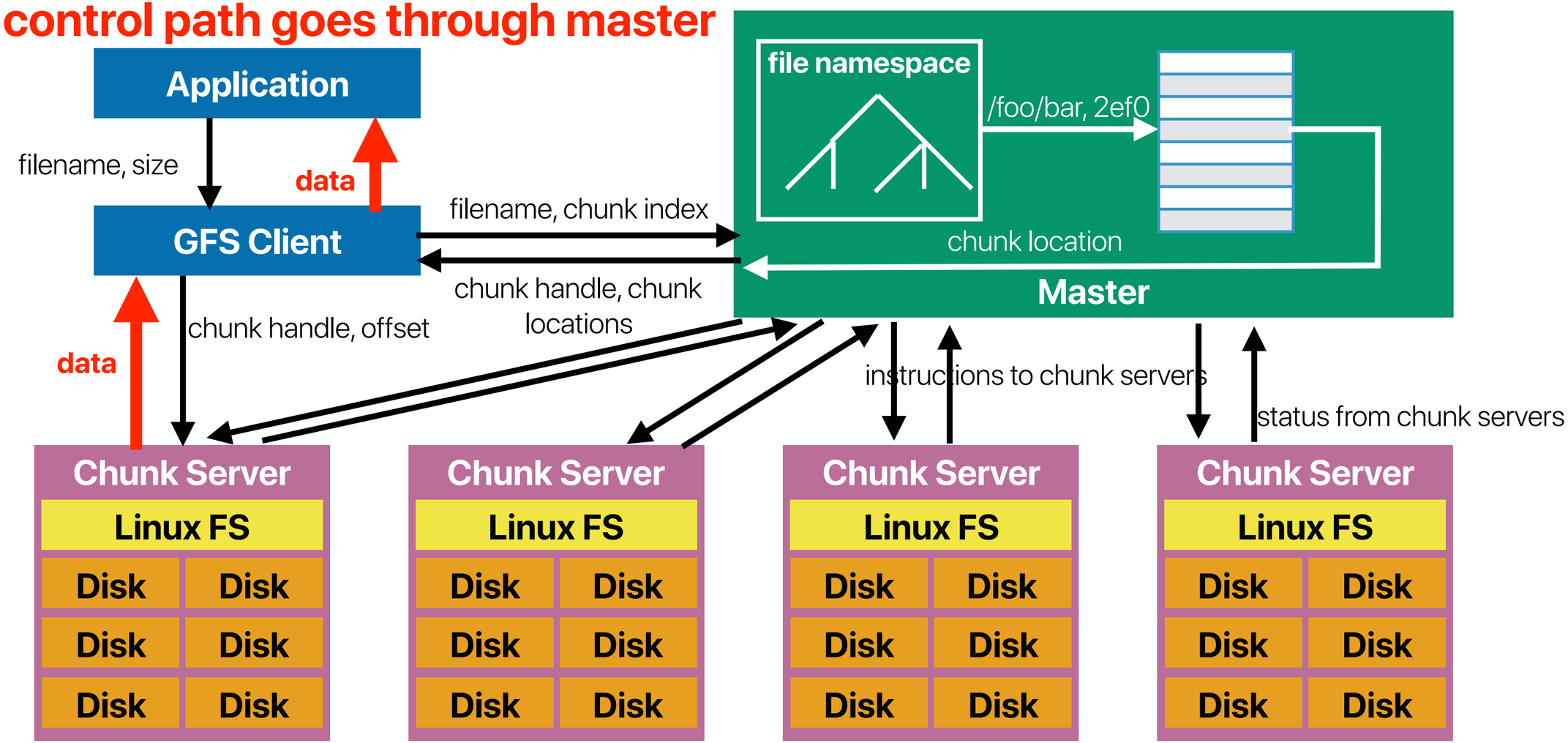
Facebook's Data Storage and Google Search Architecture

Hung-Wei Tseng

Recap: GFS architecture

decoupled data and control paths —
only control path goes through master



load balancing, replicas among chunkservers

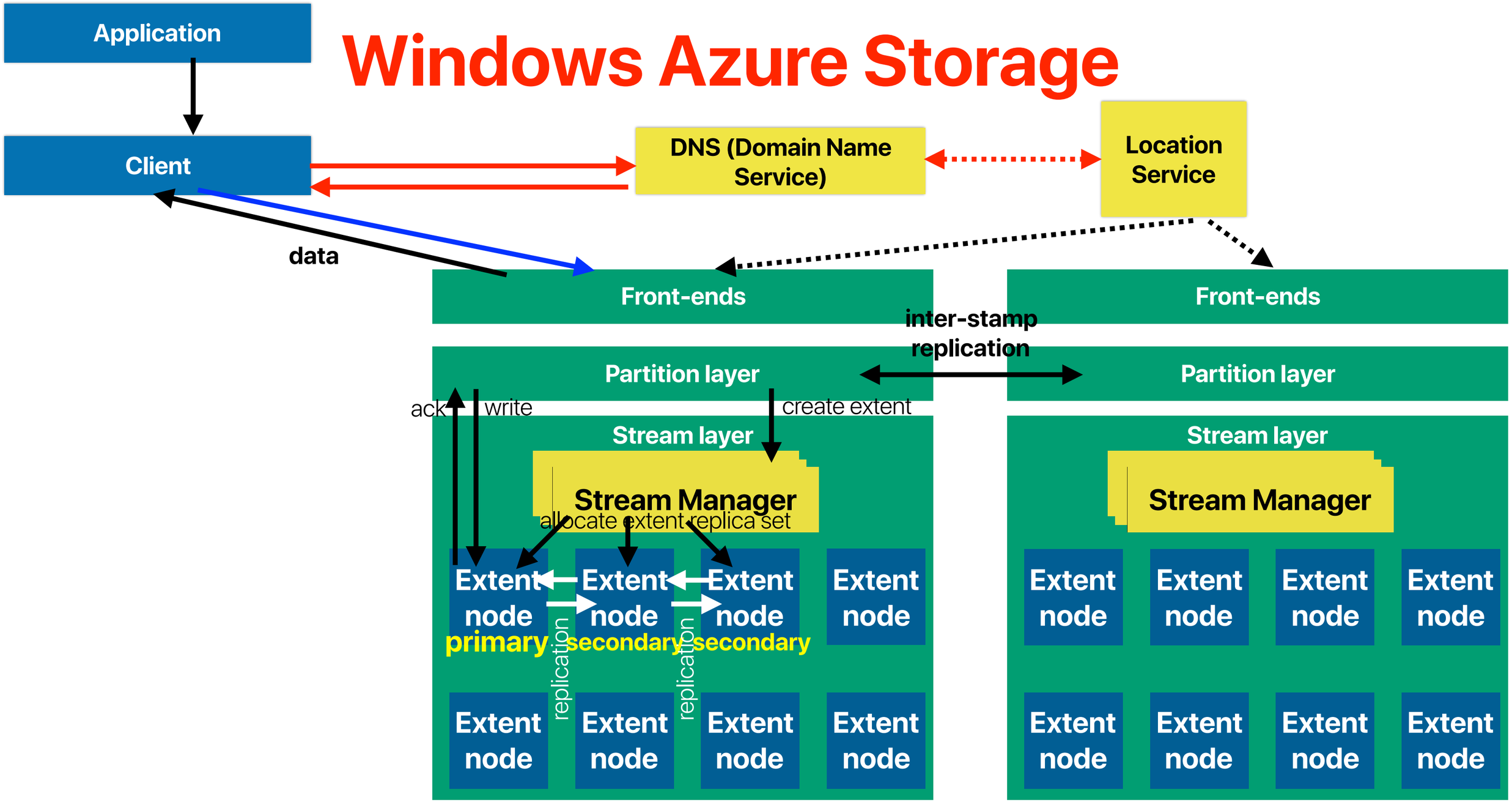
Recap: How does GFS achieve its goals?

- Storage based on inexpensive disks that fail frequently — master/chunkserver/client — MapReduce is fault tolerant
- Many large files in contrast to small files for personal data — large chunk size — MapReduce aims at processing large amount of data once
- Primarily reading streams of data — large chunk size — MapReduce reads chunks of large files
- Sequential writes appending to the end of existing files — large chunk size — Output file keep growing as workers keep writing
- Must support multiple concurrent operations — flat structure — MapReduce has thousands of workers simultaneously
- Bandwidth is more critical than latency — large chunk size — MapReduce only wants to finish tasks within "reasonable" amount of time

Recap: Why Windows Azure Storage

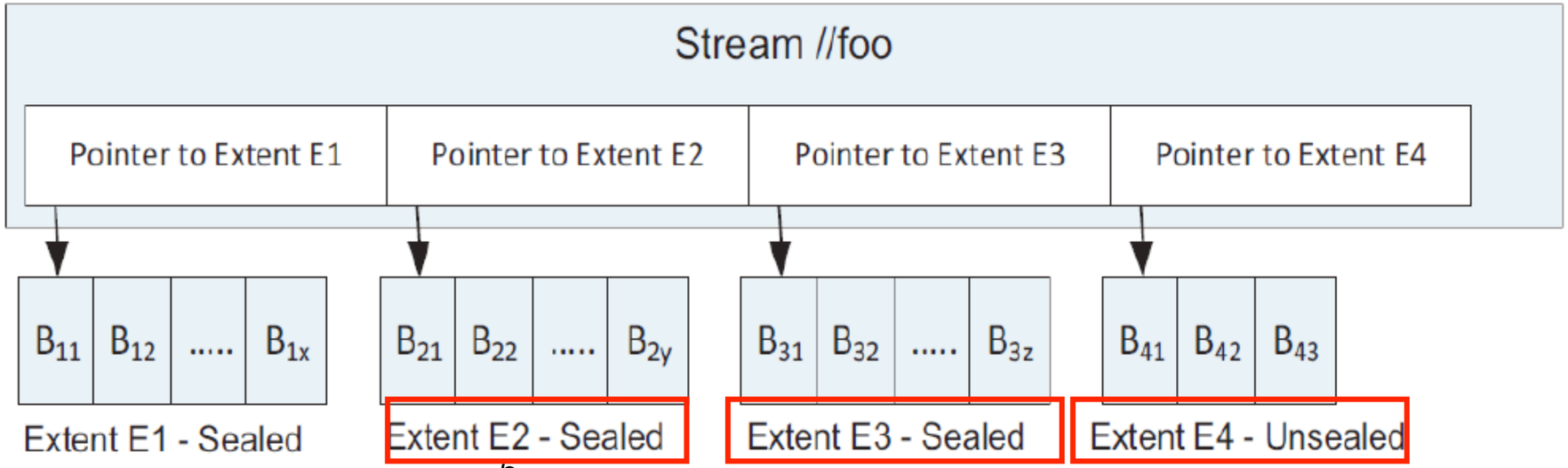
- Must tolerate many different data abstractions: blobs, tables and queues
- Learning from feedbacks in existing cloud storage
  - Strong consistency
  - Global and scalable namespace/storage
  - Disaster recovery
  - Multi-tenancy and cost of storage

Windows Azure Storage



Recap: Stream

- A stream is a collection of extents, in which an extent consists of consecutive blocks
- Each block in the stream contains a checksum to ensure the data integrity
- An update to a stream can only be appended to the end of the stream
- Two streams can share the same set of extents



Recap: Why "append-only" and "sealing"?

- In WAS, the stream is append only. The stamp will "seal" extents and extents will become immutable once sealed. How many of the following can sealing contribute to?
  - ① Must tolerate many different data abstractions: blobs, tables and queues
  - ✓ ② Strong consistency
  - ③ Global and scalable namespace/storage
  - ✓ ④ Disaster recovery
  - ✓ ⑤ Multi-tenancy and cost of storage
  - A. 1
  - B. 2
  - C. 3
  - D. 4
  - E. 5

> 2. Once an extent is sealed, any reads from any sealed replica will always see the same contents of the extent.

> **Append-only System** – Having an append-only system and sealing an extent upon failure have greatly simplified the replication protocol and handling of failure scenarios. In this

> Erasure coding sealed extents is an important optimization, given the amount of data we are storing. It reduces the cost of storing data from three full replicas within a stamp, which is three times the original data, to only 1.3x – 1.5x the original data, depending

f4: Facebook's Warm BLOB Storage System

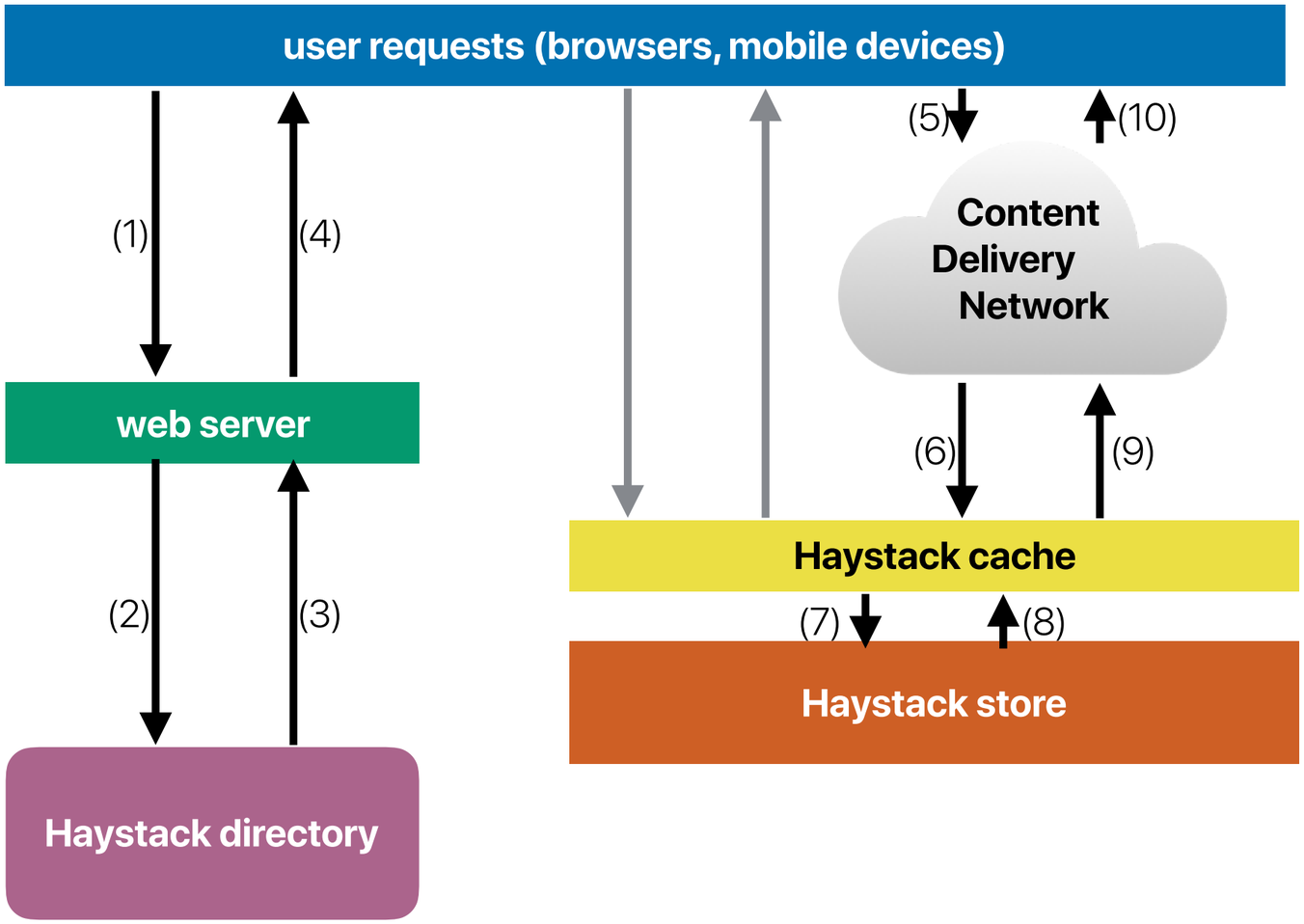
Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill,
Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar,
Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar.

FACEBOOK

The original NFS-based FB storage

- Within a data center with high-speed network, the round-trip latency of network accesses is not really a big deal
- However, the amount of metadata, especially directory metadata, is huge — cannot be cached
- As a result, each file access still requires ~ 10 inode/data requests from disks/network nodes — kill performance

Haystack

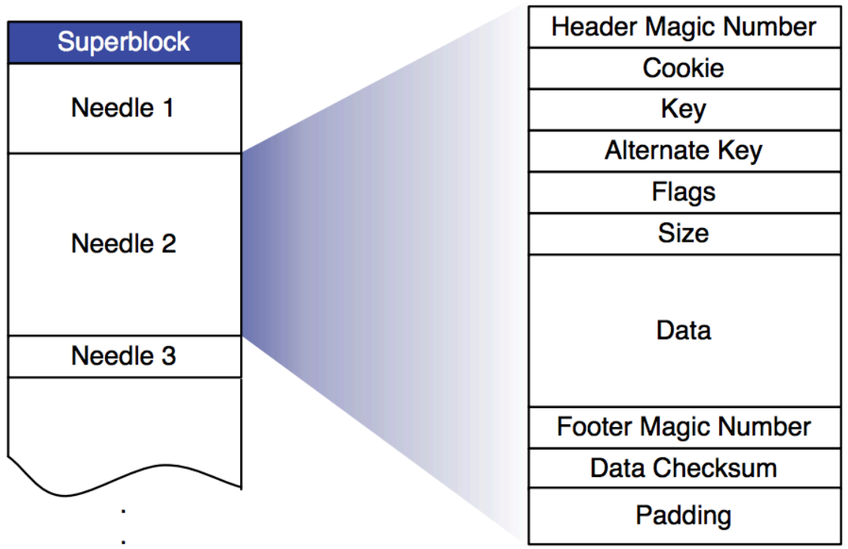


http://<CDN>/<Cache>/<Machine ID>/<Logical volume,Photo>

Finding a needle in Haystack: Facebook's photo storage. Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, Peter Vajgel. OSDI 2010

Haystack

- Each storage unit provides 10TB of usable space, using RAID-6 — 20% redundancy for parity bits
  - Each storage split into 100 physical volumes (100GB)
  - Physical volumes on different machines grouped into logical volumes
  - A photo saved to a logical volume is written to all corresponding physical volumes — **3** replicas
- Each volume is actually just a large file
  - Needle represents a photo
  - Each needle is identified through the offset
  - Sealed (the same as WAS) one the file reaches 100GB



GFS v.s. WAS

	GFS (OSDI 2003)	Facebook Haystack (OSDI 2010)	WAS (SOSP 2011)
File organizations	file chunk block	volume needle	stream extent record
System architecture	master chunkserver	directory haystack store	stream manager extent nodes
Data updates			append only updates
Consistency models	relaxed consistency		strong consistency
Data formats	files	photo/needle	multiple types of objects
Replications	intra-cluster replication	RIAD-6 & geo-replication	geo-replication
Usage of nodes	chunk server can perform both		separate computation and storage

Why FB wants f4 in addition to Haystack

- Regarding the optimization goals of f4, please identify how many the following statements is/are correct.
  - ① f4 is optimized for the throughput of accessing data
  - ② f4 is optimized for the latency of accessing data
  - ③ ✓ f4 is designed to reduce the degree of data replications in the system
  - ④ ✓ f4 is designed to tolerate various kind of failure in the system

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Storage Efficiency One of the key goals of our new system is to improve storage efficiency, i.e., reduce the effective-replication-factor while still maintaining a high degree of reliability and performance.

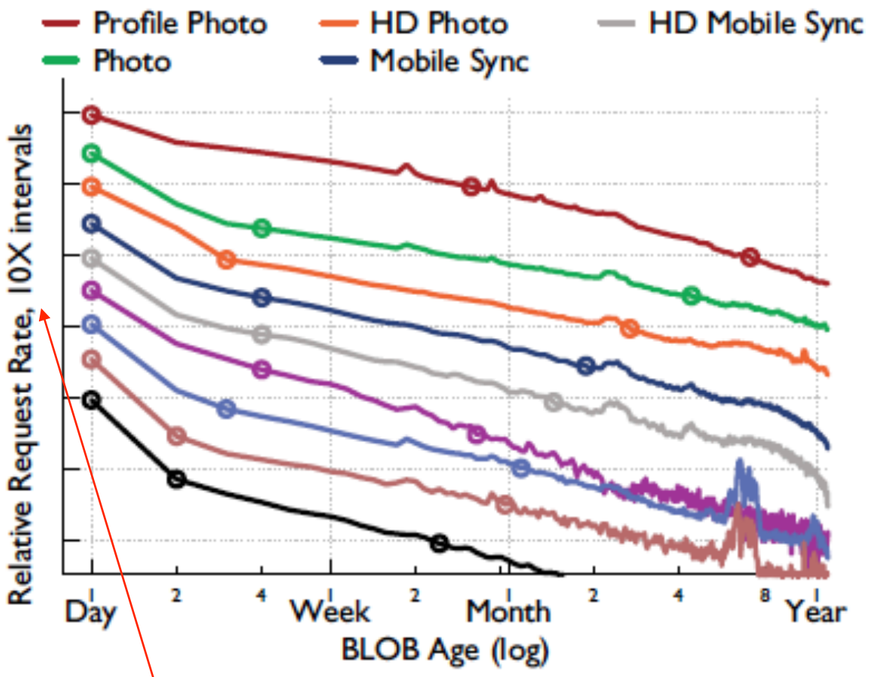
Fault Tolerance Another important goal for our storage system is fault tolerance to a hierarchy of faults to ensure we do not lose data and that storage is always available for client requests. We explicitly consider four types of failures:

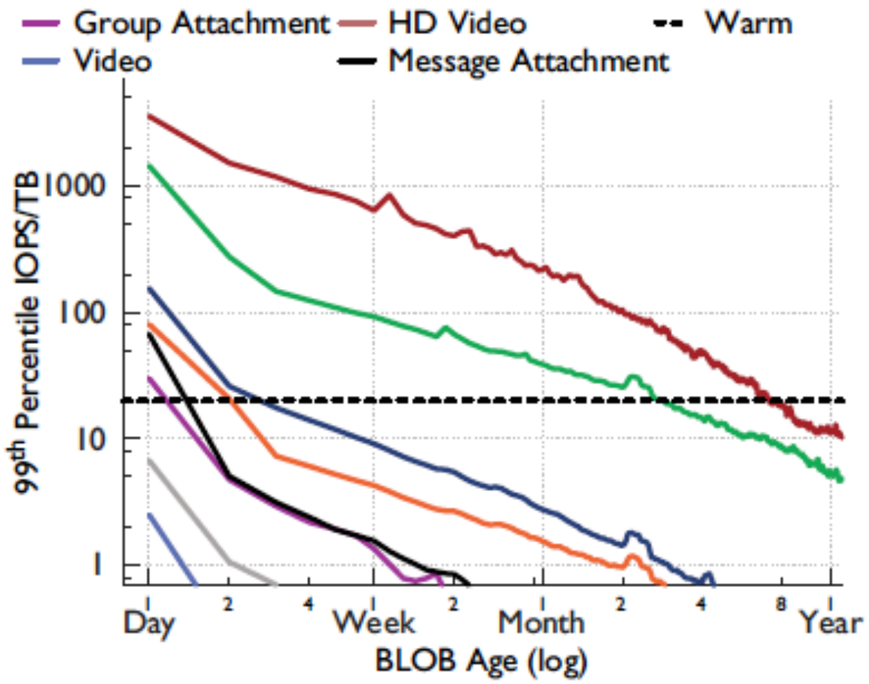
What kind of data is f4 optimized for?

- Regarding the type of data that f4 aims at, please identify how many the following statements is/are correct.
  - ① f4 is optimized for most frequently requested data in Facebook services
  - ② f4 is optimized for frequently created, deleted data
  - ③ f4 is optimized for reducing the access latency of long-term storage
  - ④ f4 is optimized for read-only data
  - ⑤ f4 is optimized for data that are not accessed very frequently

  A. 1
  B. 2
  C. 3
  D. 4
  E. 5

"Temperature" of data



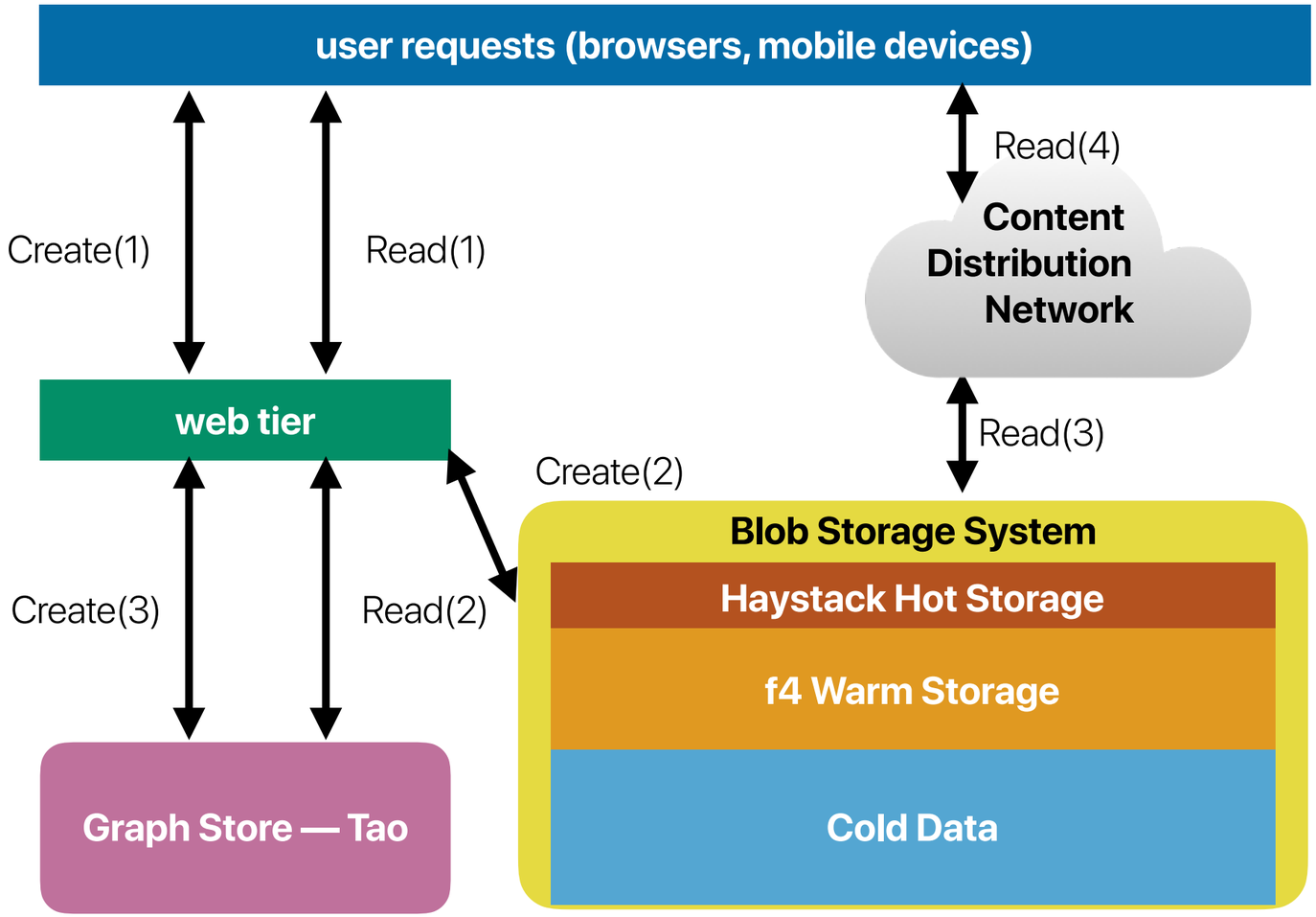


log scale — not encouraged to graph like this if you're writing a technical document or scientific paper

“Temperature” of data

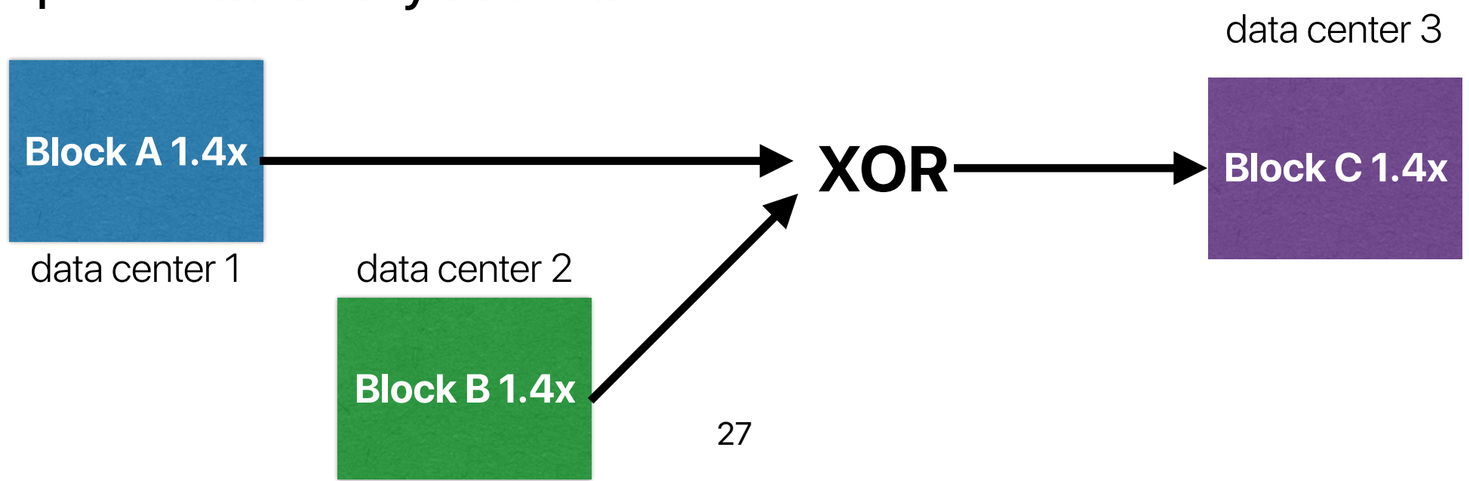
	Hot	Warm	Cold
Access Frequency	Most frequent	Less frequent	Rare
Pattern	Created often, delete often	Not so frequently read Not so frequently deleted Maybe read-only	Long-term storage, usually takes hours to retrieve
Size		65PB in 2014 and growing rapidly	

Facebook storage architecture



Storage efficiency

- Reed-Solomon erasure coding
  - Strips: 10GB data + 4GB parity — 1.4x space efficiency
  - One volume contains 10 strips
- XOR Geo-replication
  - Use XOR to reduce overhead further (e.g., Azure makes full copies)
  - Block A in DC1 + block B in DC2 -> parity block P in DC3
  - Any two blocks can be used to generate the third
  - 1.5x space efficiency
- 1.4*1.5 = 2.1x space efficiency in total



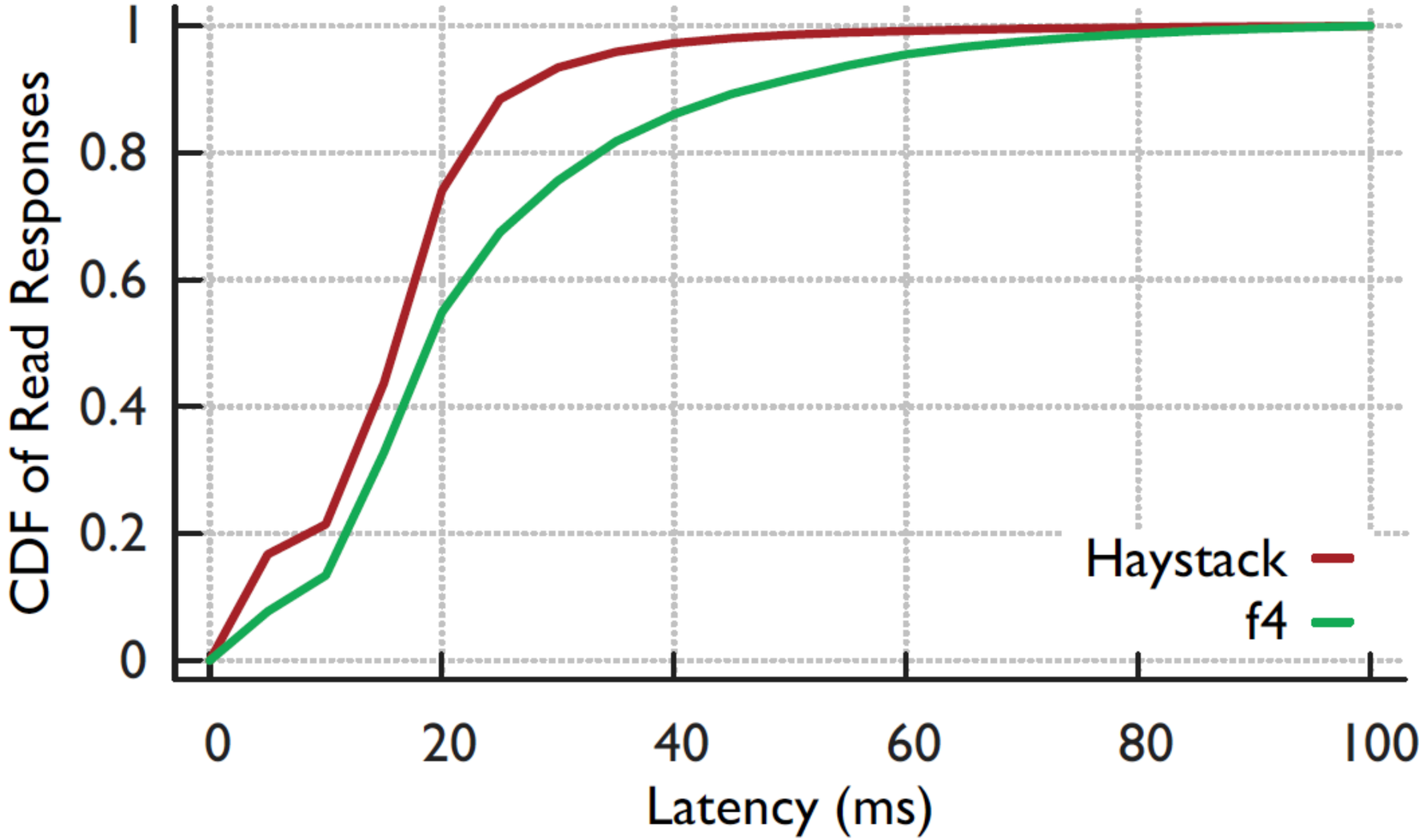
Fault tolerance

- 1%-2% HDD fail in a year
  - replicate data across multiple disks
  - Use erasure coding for storage efficiency
    - n blocks -> n + k blocks, can tolerate k simultaneous failures
    - higher cost for recovering data when there is a failure
- Host failures (periodically)
  - replicate coded blocks on different hosts
- Rack failures (multiple times/year)
  - replicate coded blocks on different racks
- Datacenter failures (rare, but catastrophic)
  - replicate blocks across data centers
  - use XOR to reduce overhead further (e.g., Azure makes full copies)
    - block A in DC1 + block B in DC2 -> parity block P in DC3
    - any two blocks can be used to generate the third
- Index files
  - use normal triple replication (tiny, little benefit in coding them)

What happens if fault occurs?

- Drive fails
  - Reconstruct blocks on another drive
  - Heavy disk, Network, CPU operation
  - one in background
- During failure, may need to reconstruct data online
  - rebuilder node reads BLOB from data + parity, reconstructs
  - only reads + reconstructs the BLOB (40KB), not the entire block (1GB)

Performance of f4



Cells

- Each cell contains 14 racks of 15 hosts, each host contains 30 4TB H.D.Ds.
- A unit of acquisition, deployment
- Storage for a set of volumes
- Similar to the idea of stamps

Web search for a planet: The Google cluster architecture

Luiz Andre Barroso, Jeffery Dean ; Urs Holzle
Google

Google search architecture

- How many of the following fulfill the design agenda of the Google search architecture described in this paper?
  - ① Reduce the hardware cost by using commodity-class and unreliable PCs
  - ② Use RAID to provide efficiency and reliability
  - ③ Use replication for better request throughput and availability
  - ④ Optimize for the peak performance

Also reliability and fault-tolerance

— replica, replica, replica

— for performance per dollar

A. 0
B. 1
C. 2
D. 3
E. 4

- *Price/performance beats peak performance.* We purchase the CPU generation that currently gives the best performance per unit price, not the CPUs that give the best absolute performance.
- *Using commodity PCs reduces the cost of computation.* As a result, we can afford to use more computational resources per query, employ more expensive techniques in our ranking algorithm, or search a larger index of documents.

- *Software reliability.* We eschew fault-tolerant hardware features such as redundant power supplies, a redundant array of inexpensive disks (RAID), and high-quality components, instead focusing on tolerating failures in software.
- *Use replication for better request throughput and availability.* Because machines are inherently unreliable, we replicate each of our internal services across many machines. Because we already replicate services across multiple machines to obtain sufficient capacity, this type of fault tolerance almost comes for free.

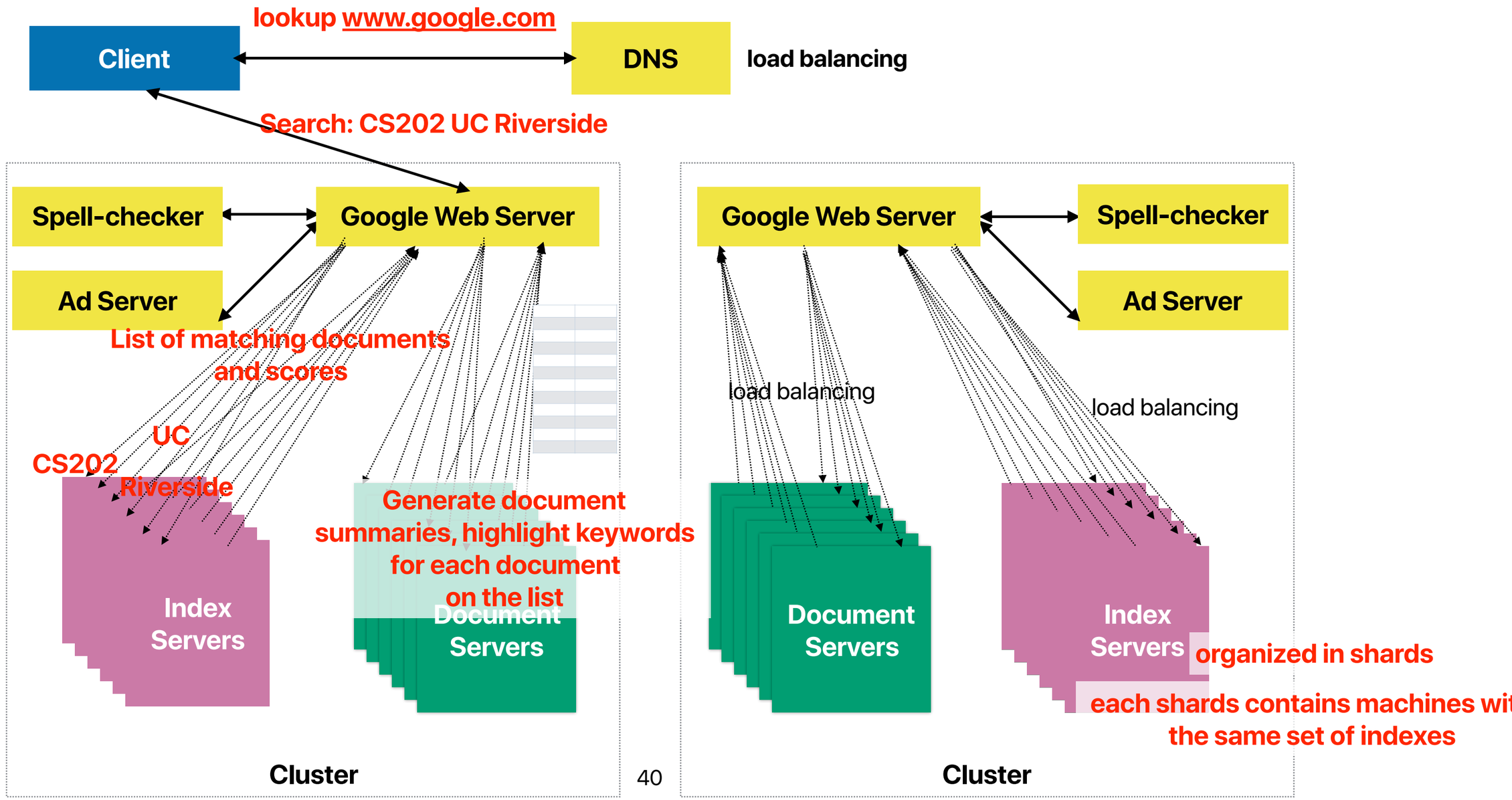
Why Google Search Architecture?

- The demand of performing search queries efficiently
  - Each query reads hundreds of MBs of data
  - Support the peak traffic would require expensive supercomputers or high-end servers
- We need a **cost-effective** approach to address this demand
  - Google search is compare against “AltaVista” search engine that uses DEC's high-performance alpha-based multiprocessor systems
  - AltaVista is later acquired by Yahoo! and you know the later story…

What Google proposes?

- Using commodity-class / unreliable PCs
- Provide reliability in software rather in hardware
- Target the best aggregate request throughput, not peak server response time

Google query-serving architecture



Replication is the key

- Scalability: simply add more replicas, the service capacity can improve
- Availability: even though one machine fails, another replica to take over

What kind of processors Google search needs

- If we are designing a processor just for Google search or similar type of applications, how many of the following targets/features would fulfill the demand?
  - ① Can execute many instructions from the same process/thread simultaneously
  - ② Can execute many processes/threads simultaneously
  - ③ Can predict branch outcome accurately
  - ④ Have very large cache capacity
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

given how little ILP our application yields, and shorter pipelines would reduce or eliminate branch mispredict penalties. The avail-

prediction logic. In essence, there isn't that much exploitable instruction-level parallelism (ILP) in the workload. Our measurements suggest that the level of aggressive out-of-order, speculative execution present in modern processors is already beyond the point of diminishing performance returns for such programs.

For such workloads, a memory system with a relatively modest sized L2 cache, short L2 cache and memory latencies, and longer (perhaps 128 byte) cache lines is likely to be the most effective.

end ones. Exploiting such abundant thread-level parallelism at the microarchitecture level appears equally promising. Both simultaneous multithreading (SMT) and chip multiprocessor (CMP) architectures target thread-level parallelism and should improve the performance of many of our servers. Some early

Hardware

- Processor
  - Index search has little ILPs — doesn't need complex cores
  - Index search can be highly parallelized — processors with thread-level parallelism would be a good fit (e.g. Simultaneous Multithreading, SMT and Chip Multicprocessor, CMP)
  - Branch predictor matters
- Memory: Good spatial locality. Moderate cache size will suffice
- Storage: No SCSI, No RAID — not worth it
- Power: is an issue, but only $1,500/mo operating bill vs $7,700 capital expense

Will their architecture work for other things?

As mentioned earlier, our infrastructure consists of a massively large cluster of inexpensive desktop-class machines, as opposed to a smaller number of large-scale shared-memory machines. Large shared-memory machines are most useful when the computation-to-communication ratio is low; communication patterns or data partitioning are dynamic or hard to predict; or when total cost of ownership dwarfs hardware costs (due to management overhead and software licensing prices). In those situations they justify their high price tags.

At Google's scale, some limits of massive server parallelism do become apparent, such as the limited cooling capacity of commercial data centers and the less-than-optimal fit of current CPUs for throughput-oriented applications. Nevertheless, using inexpensive PCs to handle Google's large-scale computations has drastically increased the amount of computation we can afford to spend per query, thus helping to improve the Internet search experience of tens of millions of users. MICRO

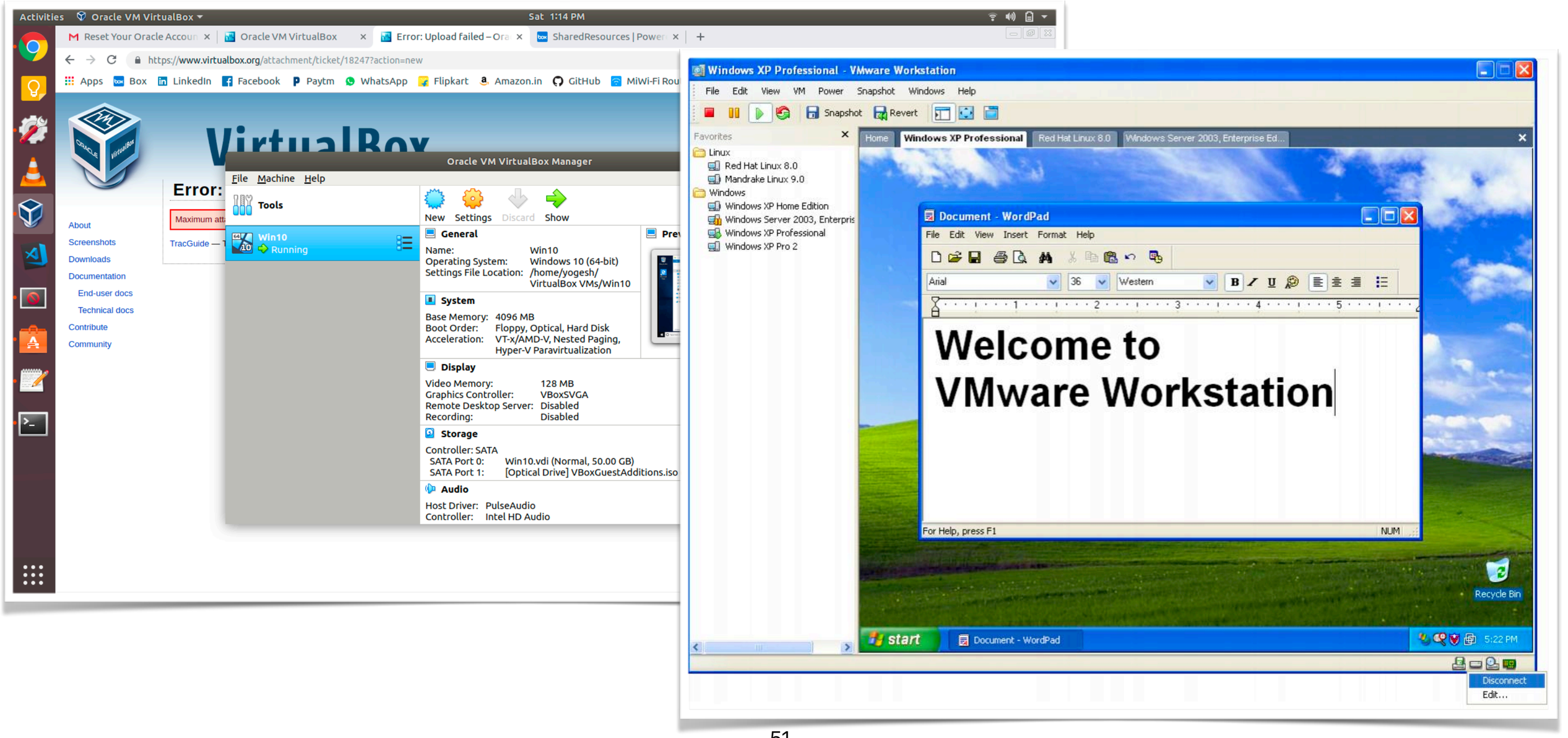
At first sight, it might appear that there are few applications that share Google's characteristics, because there are few services that require many thousands of servers and petabytes of storage. However, many applications share the essential traits that allow for a PC-based cluster architecture. As long as an application orientation focuses on the price/performance and can run on servers that have no private state (so servers can be replicated), it might benefit from using a similar architecture. Common examples include high-volume Web servers or application servers that are computationally intensive but essentially stateless. All of these applications have plenty of request-level parallelism, a characteristic exploitable by running individual requests on separate servers. In fact, larger Web sites already commonly use such architectures.

Metrics we care about data center design

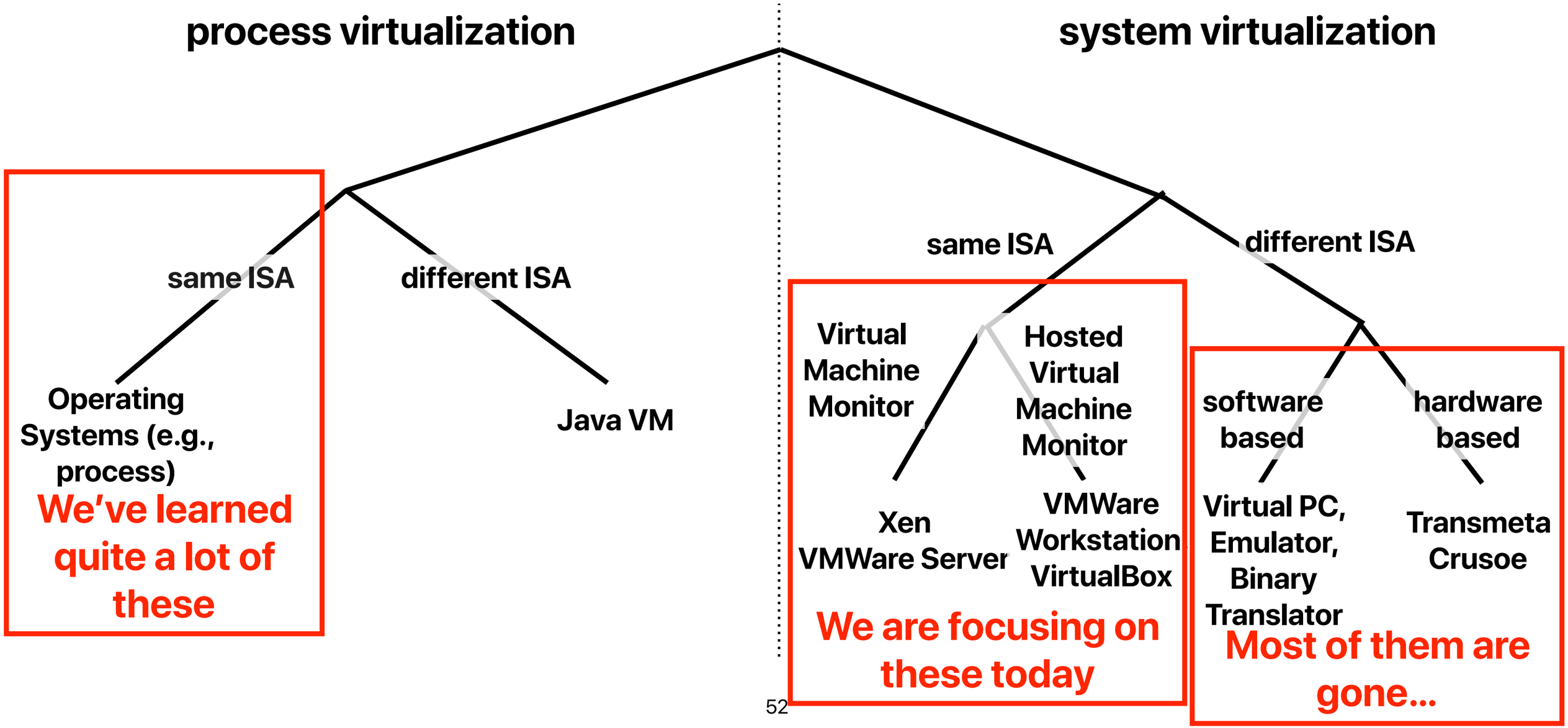
- Costs — machine architecture, distributed system architecture, replication strategies
- Power — machine architecture
- Energy — machine architecture
- Space-efficiency — erasure coding, replication, distributed
- Throughput — replication, distributed
- Reliability — replication

Virtual machines

Virtual machines



Taxonomy of virtualization



Announcement

- iEVAL
  - We highly value your opinions
  - Submit your screenshot of confirmation, equivalent to a full-credit reading quiz
- Check your grades on iLearn as soon as possible
  - We drop 2 of your lowest reading quizzes
  - We allow 4 absences through out the whole quarter
  - Midterm grade is up. One week regrading policy applies — check the website regarding how to initiate that
  - “Weighted Total” is your current total.