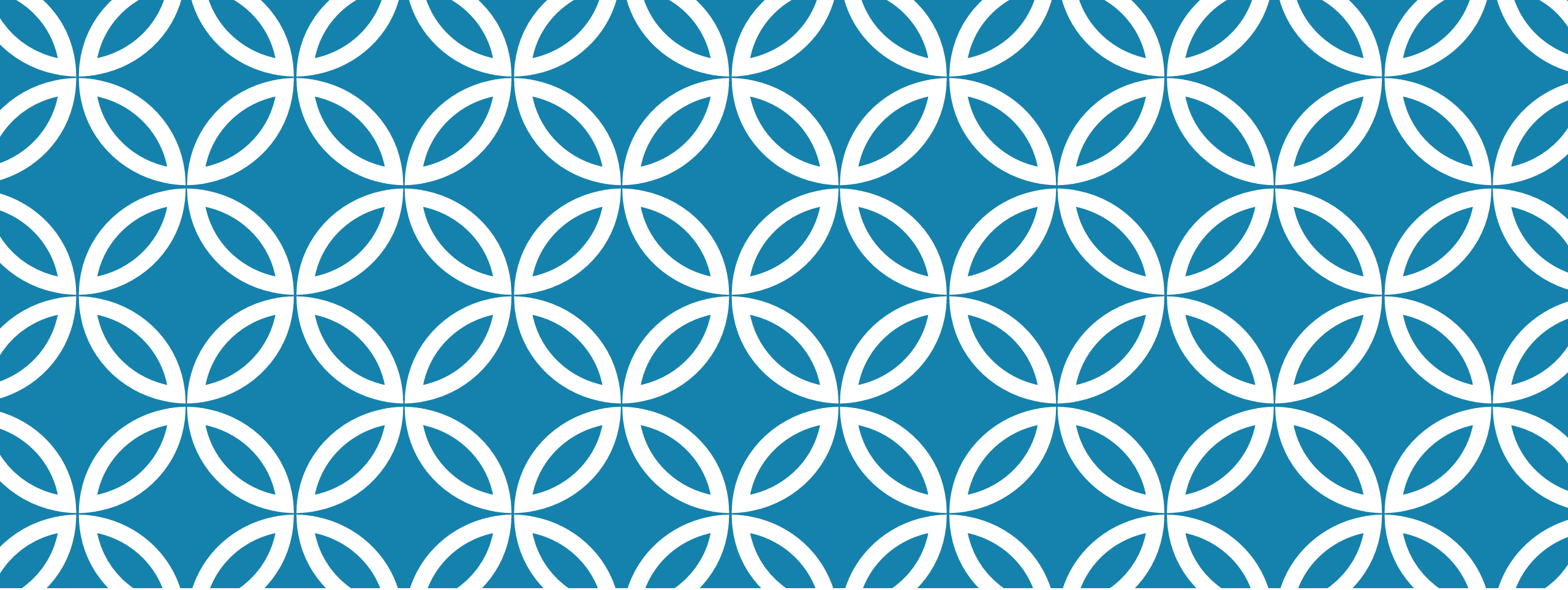


SOCIAL NETWORK GRAPHS, FACEBOOK'S TAO SERVICE

CS4414/5416
Fall 2025 Lecture 18

OUR LAST LECTURE LOOKED AT FACEBOOK'S CDN AND HOW IT DOES CACHING…

Today, we will look at the way that Facebook creates and manages its TAO database for the social networking graph. This is a central form of big data in Facebook, and most cloud platforms have a similar system.

Central to the concept is the ease of doing computing on collections. TAO is focused on rapidly updating the graph, but also on offering iterators that match closely with expected application use patterns.

WHAT IS "BIG DATA" ACTUALLY ABOUT?

Early in the cloud era, research at companies like Google and Amazon made it clear that people respond well to social networking tools and smarter advertising placement and recommendations.

The idea is simple: "People with Ken's interest find this store fantastic." "Anne really like Eileen Fisher and might want to know about this 15% off sale on spring clothing." "Sarah had a flat tire and needs new ones."

THEY HAD A LOT OF CUSTOMERS AND DATA

Web search and product search tools needed to deal with billions of web pages and hundreds of millions of products.

Billions of people use these modern platforms.

So simple ideas still involve enormous data objects that simply can't fit in memory on modern machines. And yet in-memory computing is far faster than any form of disk-based storage and computing!

WHAT ARE THE BIG DATA FILES?

Snapshot of all the web pages in the world, updated daily.

Current product data & price for every product Amazon knows about.

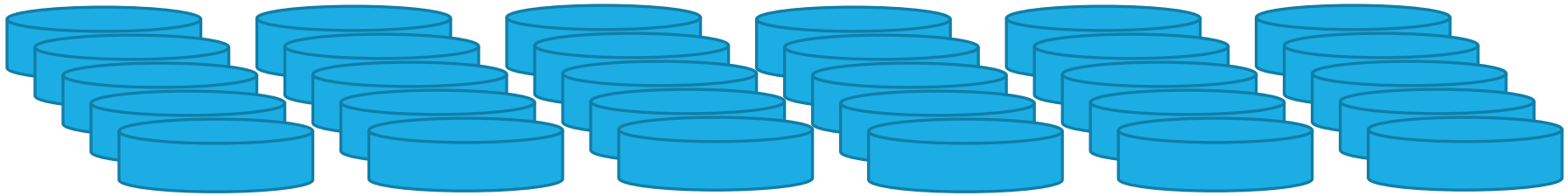
Social networking graph for all of Facebook

MANY CHALLENGES

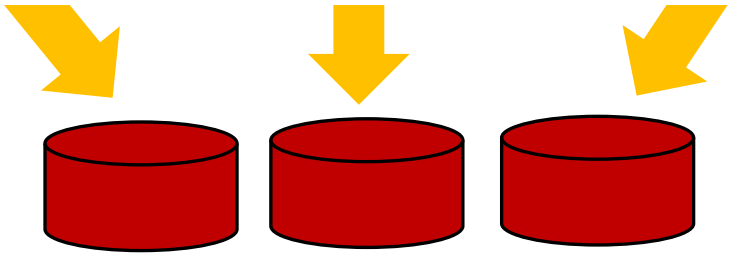
XenonStack.com

VISUALIZING THE PIPELINE

Data starts out sharded over servers



Early pipeline stages are extremely parallel: they *extract, transform, summarize*



Eventually we squeeze our results into a more useful form, like a trained machine-learning model. The first stages can run for a long time before this converges

Copy the model to wherever we plan to use it.

FOR WEB PAGES THIS IS "EASY"

The early steps mostly extract words or phrases, and summarize by doing things like counting or making lists of URLs.

The computational stages do work similar to sorting (but at a very large scale), e.g. finding the "most authoritative pages" by organizing web pages in a graph and then finding the graph nodes with highest weight for a given search.

When we create a trained machine-learning model, the output is some sort of numerical data that parameterizes a "formula" for later use (like to select ads).

WHAT ABOUT BIG DATA FOR SOCIAL NETWORKS?

Here we tend to be dealing with very large and very dynamic graphs.

The approaches used involve specialized solutions that can cope with the resulting dynamic updates.

Facebook's TAO is a great example, we'll look closely at it.

WHY DOES THIS TOPIC FIT 4414/5416?

Our fundamental goal is **highly performant computing** of the kind used in building and deploying ML systems and cloud-hosted ML services

TAO is the most efficient cloud service for big-data social networking. It *uses* ideas we've seen in the first half of the course, and also it functions as a *cloud hosted knowledge repository* widely used in new ML services.

And we access TAO through **collection-based computing** frameworks like cpplinq and Python's Pandas. These were the main focus in lecture 11

facebook

TAO

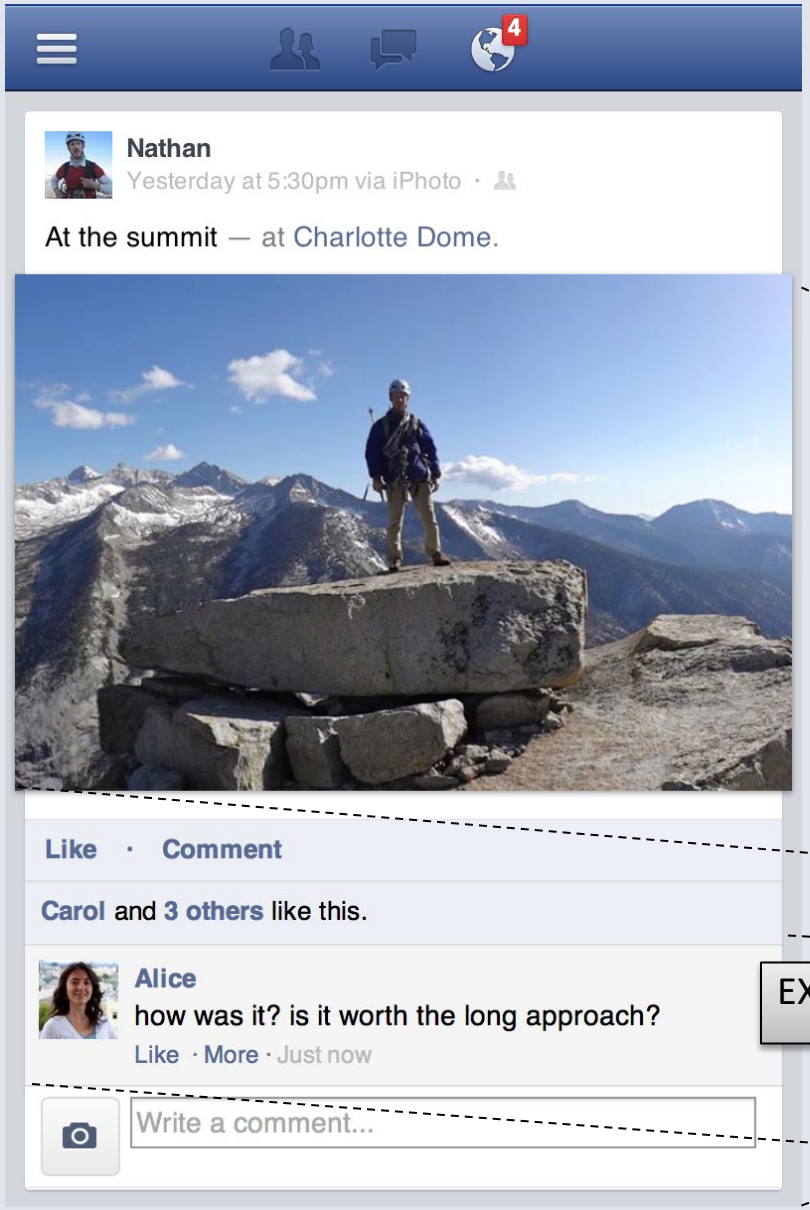
Facebook's Distributed Data Store for the Social Graph

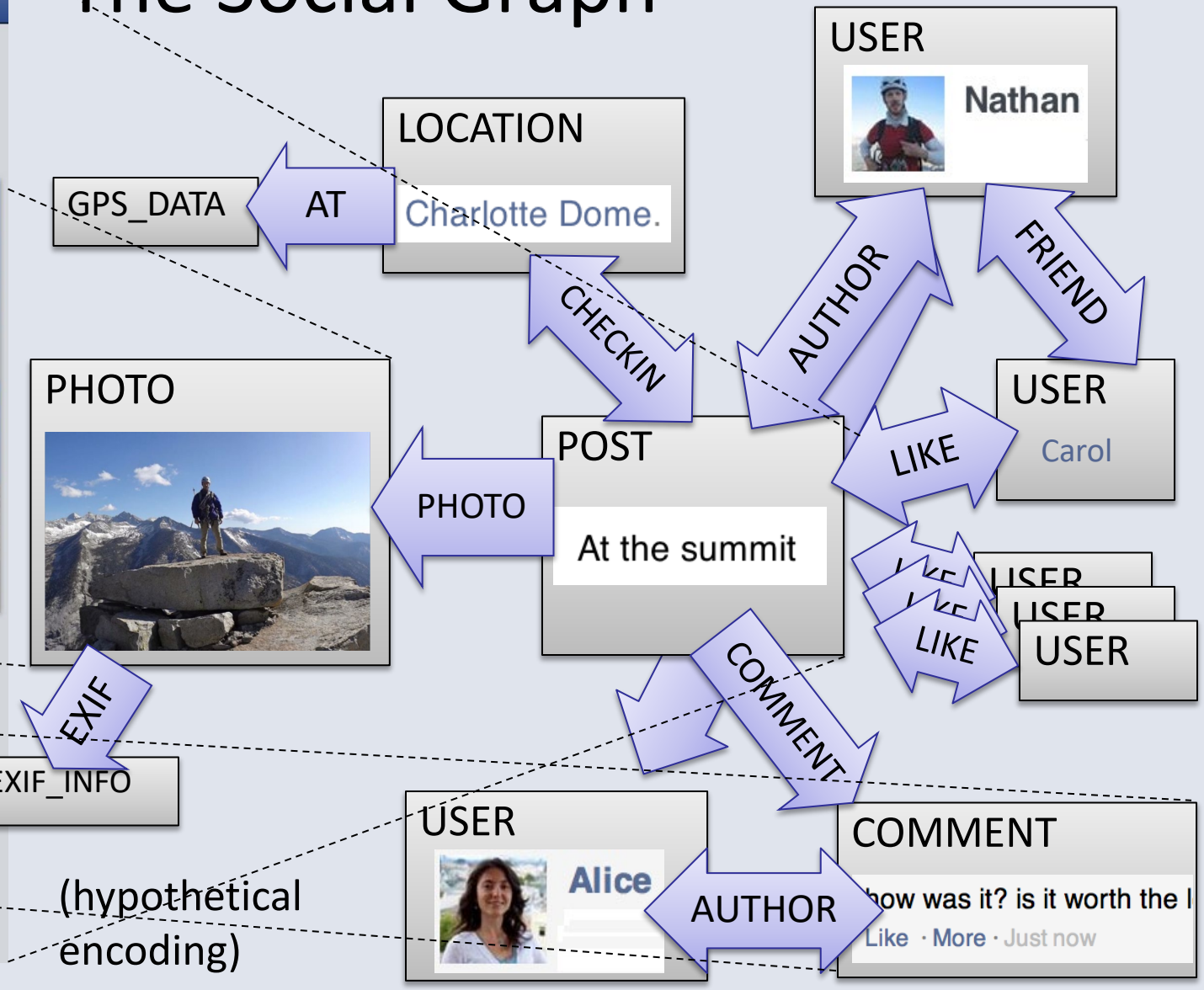
Cornell PhD who worked with Professor van Renesse. Graduated in 2010
Now one of several people with the title "Director of Engineering"
He owns the distributed systems area: the Facebook "edge"

Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, Venkat Venkataramani

Presented at USENIX ATC – June 26, 2013

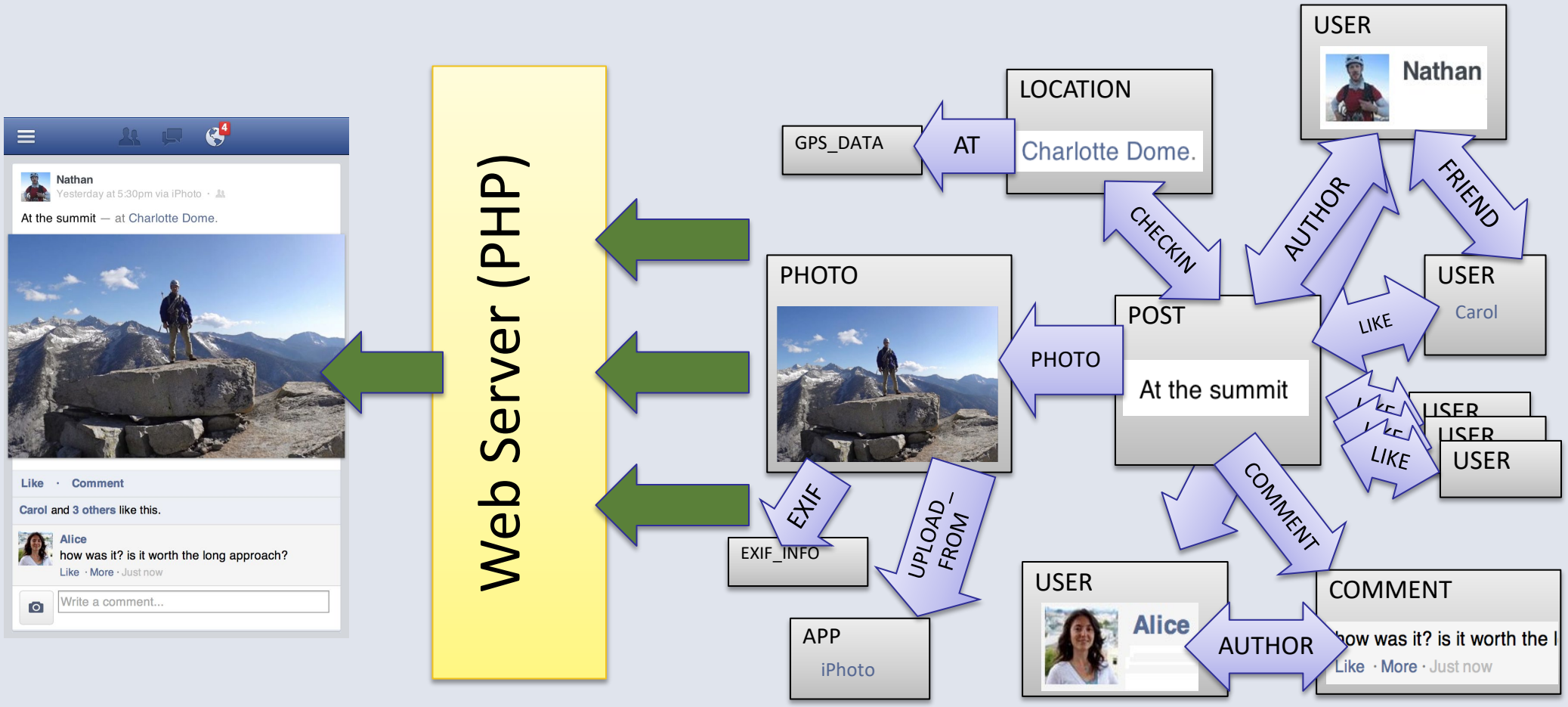
The Social Graph



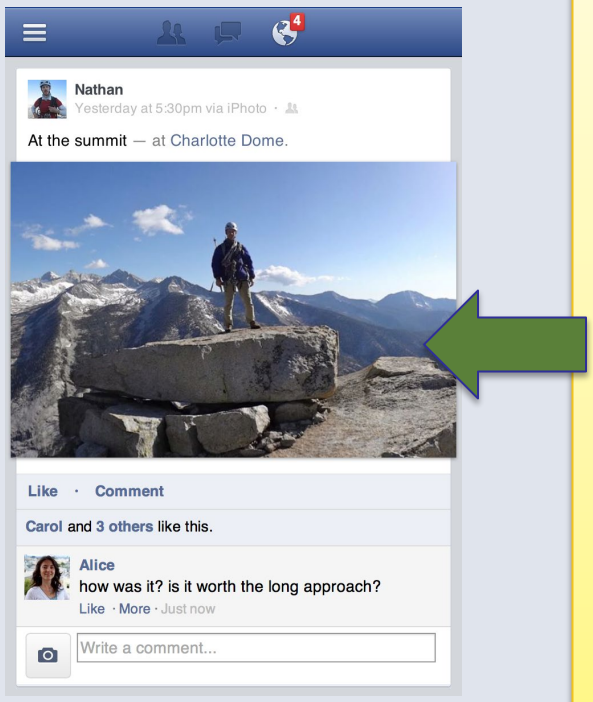


(hypothetical encoding)

Dynamically Rendering the Graph

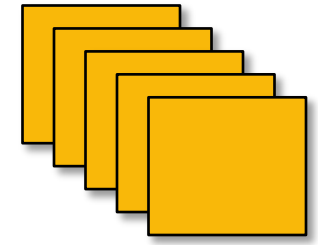


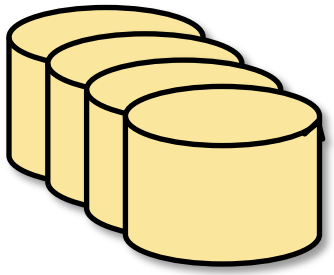
Dynamically Rendering the Graph

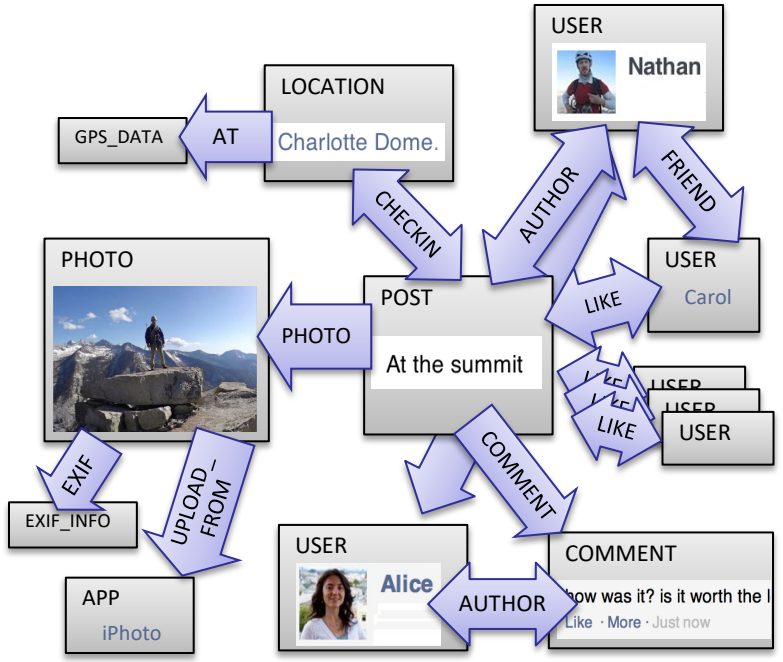


Web Server (PHP)

TAO







- 1 billion queries/second
- many petabytes of data

TAO opts for a relaxation of the SQL model

TAO applications treat the social networking graph as a form of SQL database.

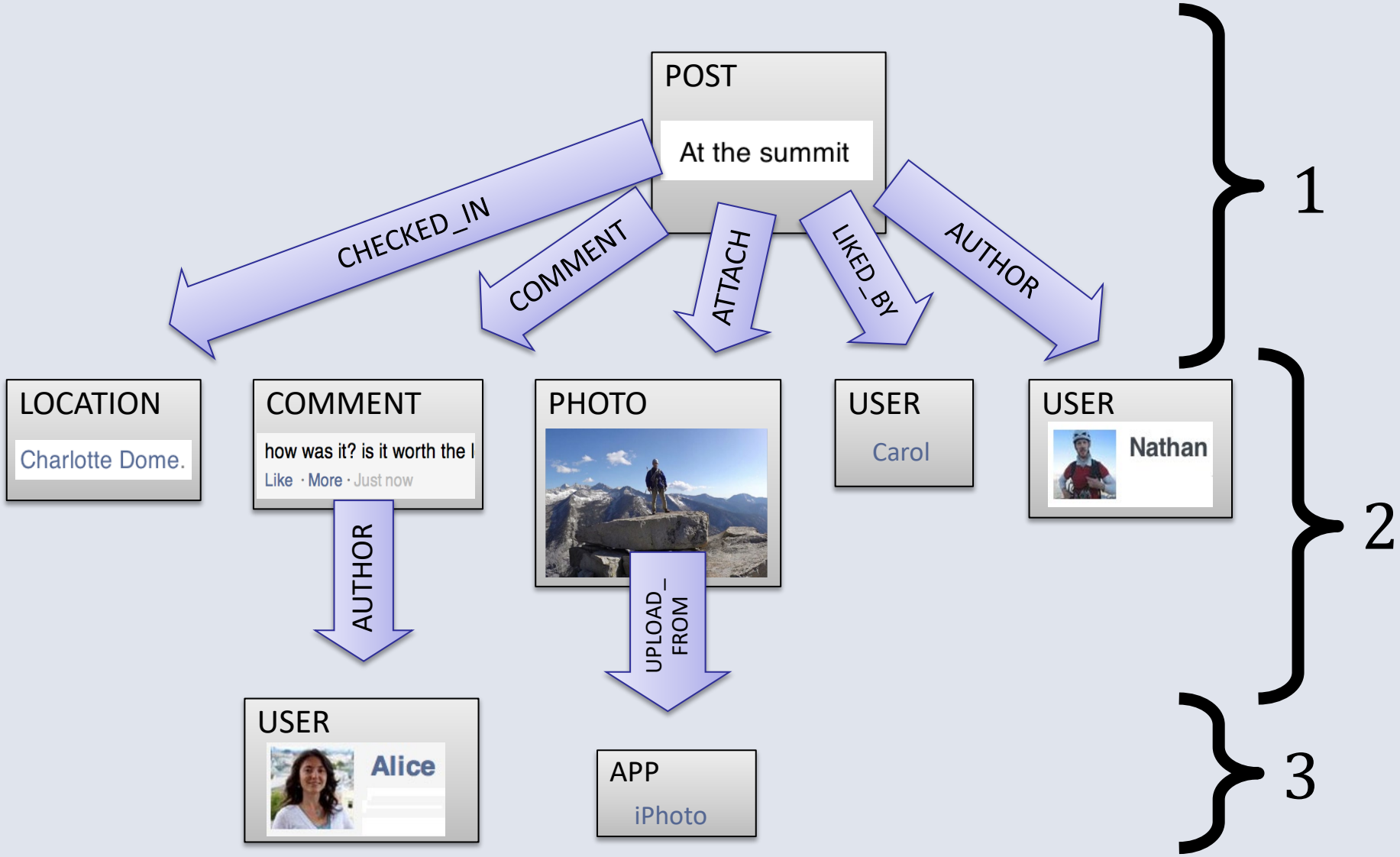
- But they don't need full SQL, and so TAO itself doesn't have to be an SQL database.
- And they don't depend on the atomicity properties typical of full SQL databases.

- In fact the *back end* of TAO actually is serializable, but it runs out of band, in a batched and high-volume way. Eventually, the system catches up and becomes consistent.
- The only edge consistency promise is that to avoid returning broken association lists, because applications find such situations hard to handle.

What Are TAO's Goals/Challenges?

- Efficiency at scale

Dynamic Resolution of Data Dependencies

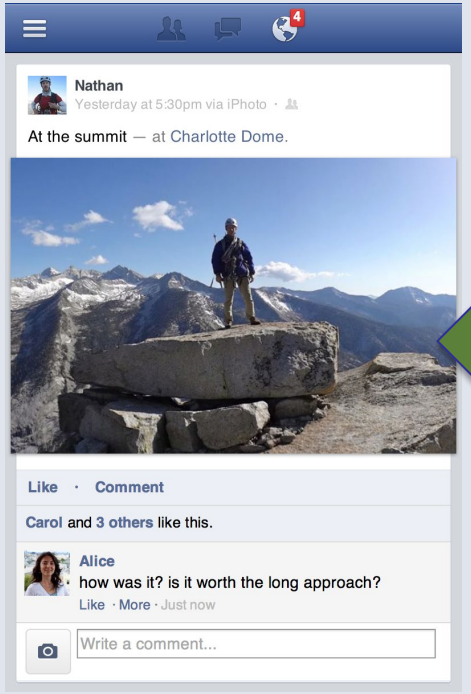


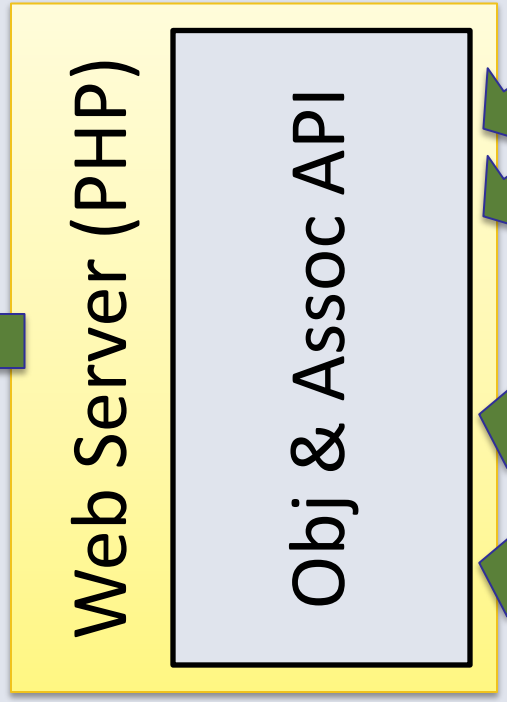
What Are TAO's Goals/Challenges?

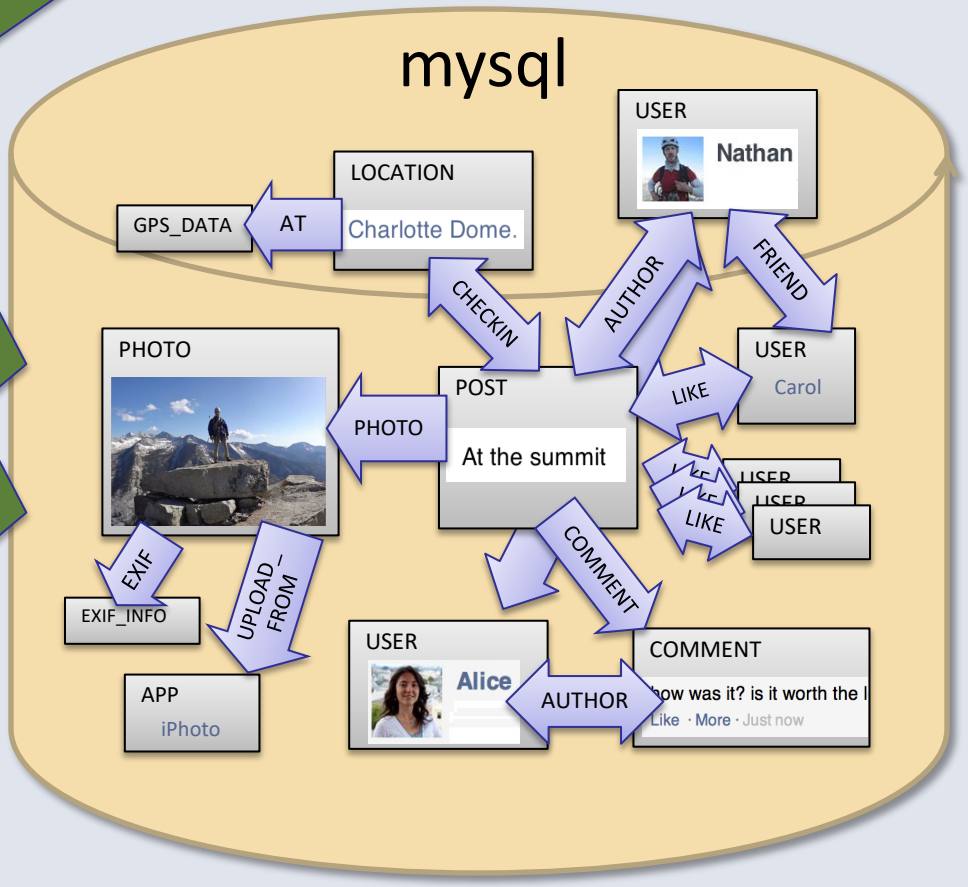
- Efficiency at scale
- Low read latency
- Timeliness of writes
- High Read Availability

Graph in “Memcache”

A distributed in-memory cache for TAO objects (nodes, edges, edge lists)





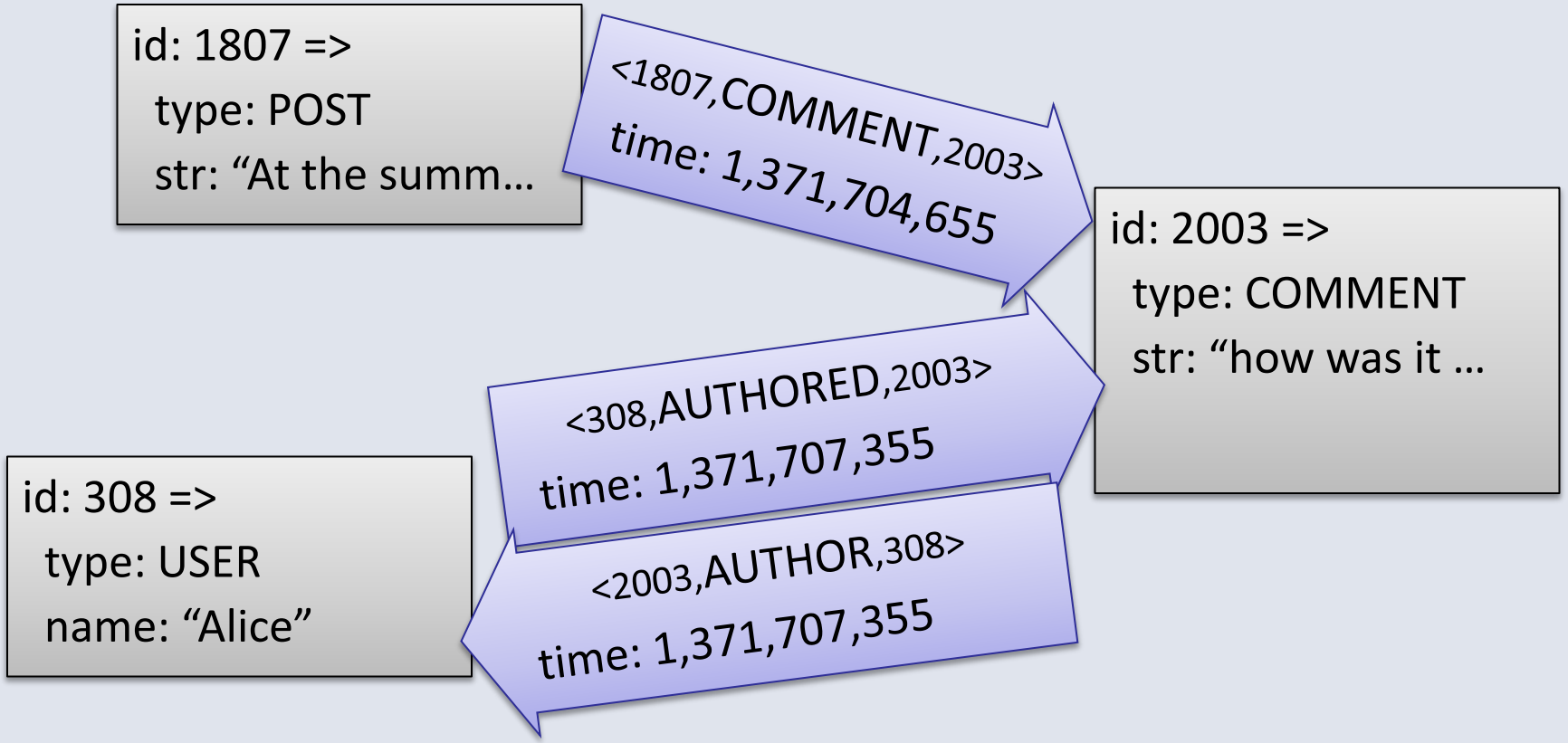


Objects = Nodes

- Identified by unique 64-bit IDs
- Typed, with a schema for fields

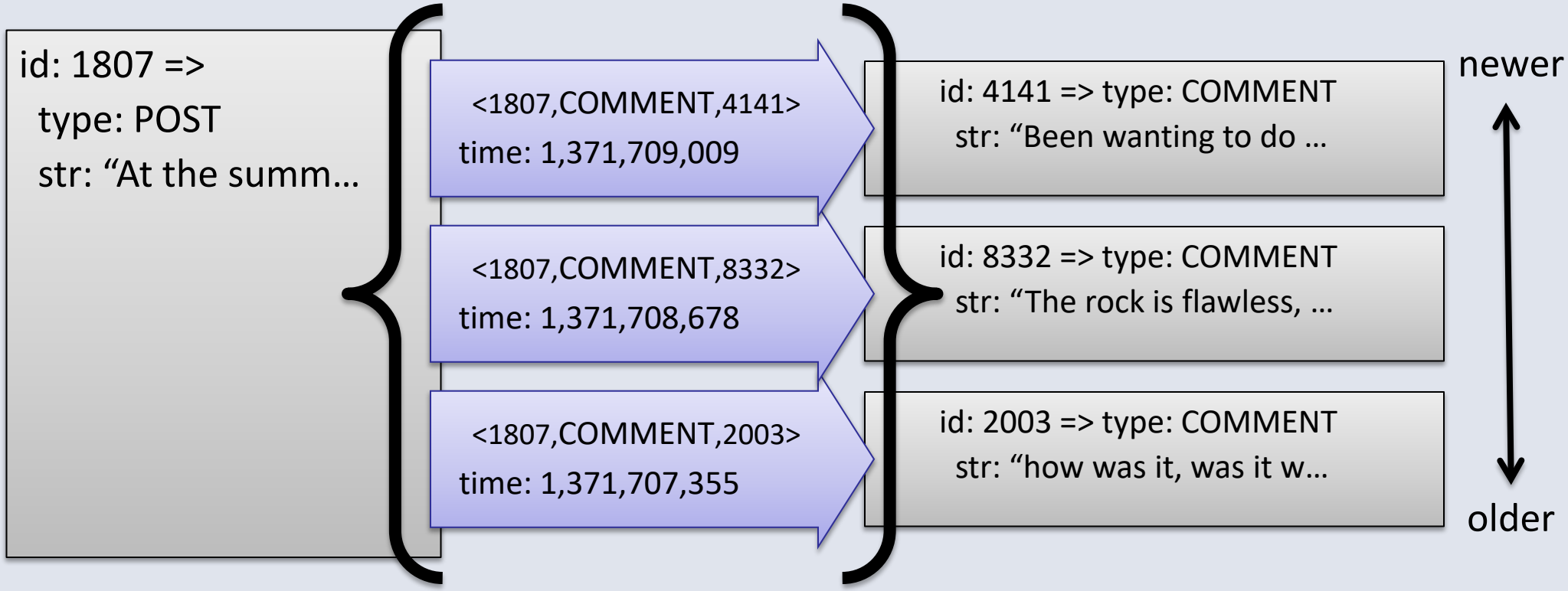
Associations = Edges

- Identified by <id1, type, id2>
- Bidirectional associations are two edges, same or different type



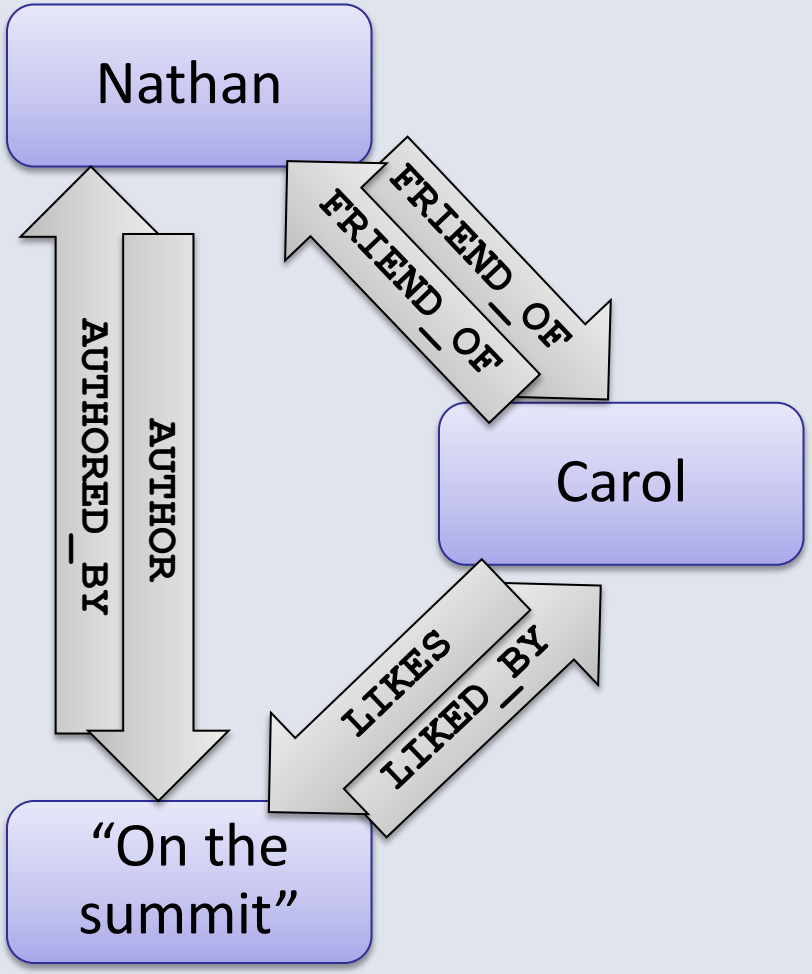
Association Lists

- <id1, type, *>
- Descending order by time
- Query sublist by position or time
- Query size of entire list



Inverse associations

- Bidirectional relationships have separate *a*→*b* and *b*→*a* edges
  - inv_type(**`LIKES`**) = **`LIKED_BY`**
  - inv_type(**`FRIEND_OF`**) = **`FRIEND_OF`**
- Forward and inverse types linked only during write
  - TAO **`assoc_add`** will update both
  - Not atomic, but failures are logged and repaired



Coding Style?

- Developers code against TAO from any language Meta supports (C++ is preferred)

- The basic style of coding centers on using SQL-style embedded code (like Pandas, cpplinq):
  - The application queries TAO for a collection of tuples such as “friends of Ken”
  - Result is an iterator object. The application uses it to filter or transform that data
  - This enables an action like “Email Ken’s friends to invite them to the party”
  - Most applications are small pieces of code performing these kinds of small tasks

Larger ML actions on TAO often involve “graphical learning” in order to support inference. For example, Joe posted that he really likes his new hiking shoes… which of Joe’s friends might be influenced and likely to buy a pair too?

But graphical learning is out of our scope

Objects and Associations API

Reads – 99.8%

- Point queries
  - **obj_get** 28.9%
  - **assoc_get** 15.7%
- Range queries
  - **assoc_range** 40.9%
  - **assoc_time_range** 2.8%
- Count queries
  - **assoc_count** 11.7%

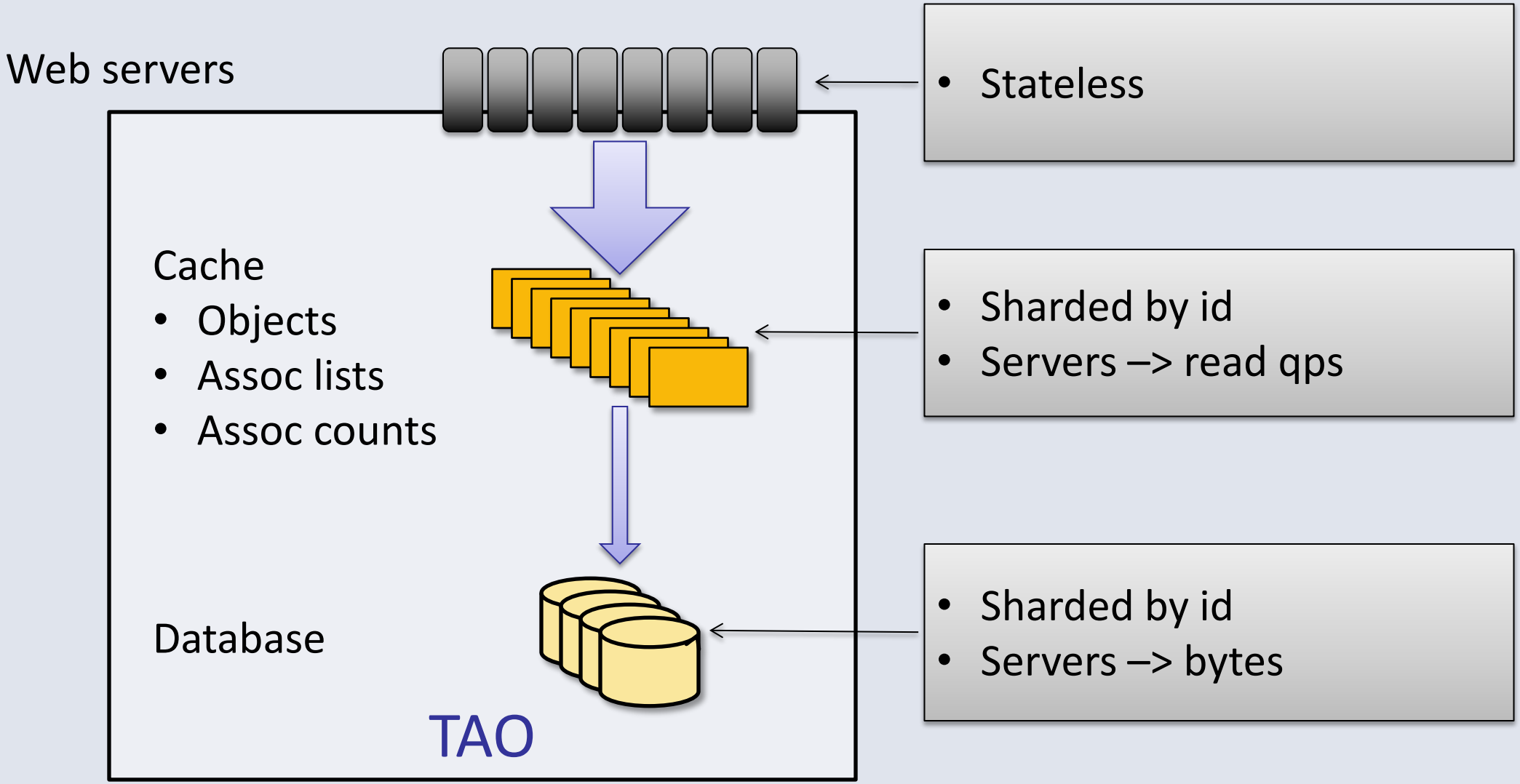
Writes – 0.2%

- Create, update, delete for objects
  - **obj_add** 16.5%
  - **obj_update** 20.7%
  - **obj_del** 2.0%
- Set and delete for associations
  - **assoc_add** 52.5%
  - **assoc_del** 8.3%

What Are TAO's Goals/Challenges?

- Efficiency at scale
- Low read latency
- Timeliness of writes
- High Read Availability

Independent Scaling by Separating Roles

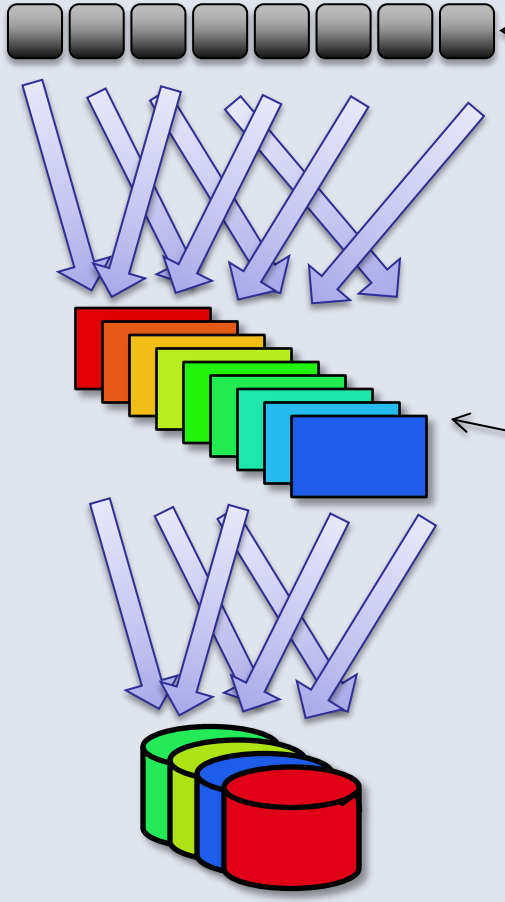


Subdividing the Data Center

Web servers

Cache

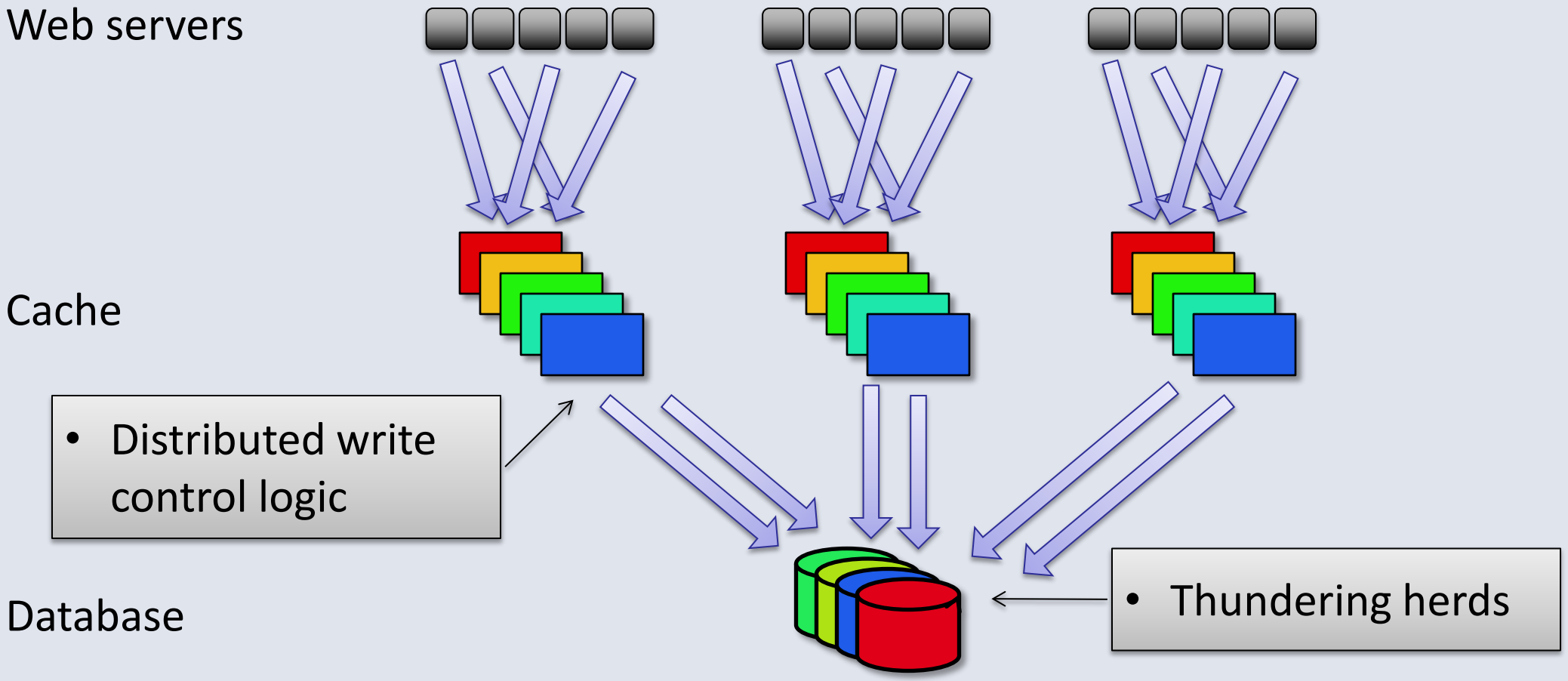
Database



- Inefficient failure detection
- Many switch traversals

- Many open sockets
- Lots of hot spots

Subdividing the Data Center



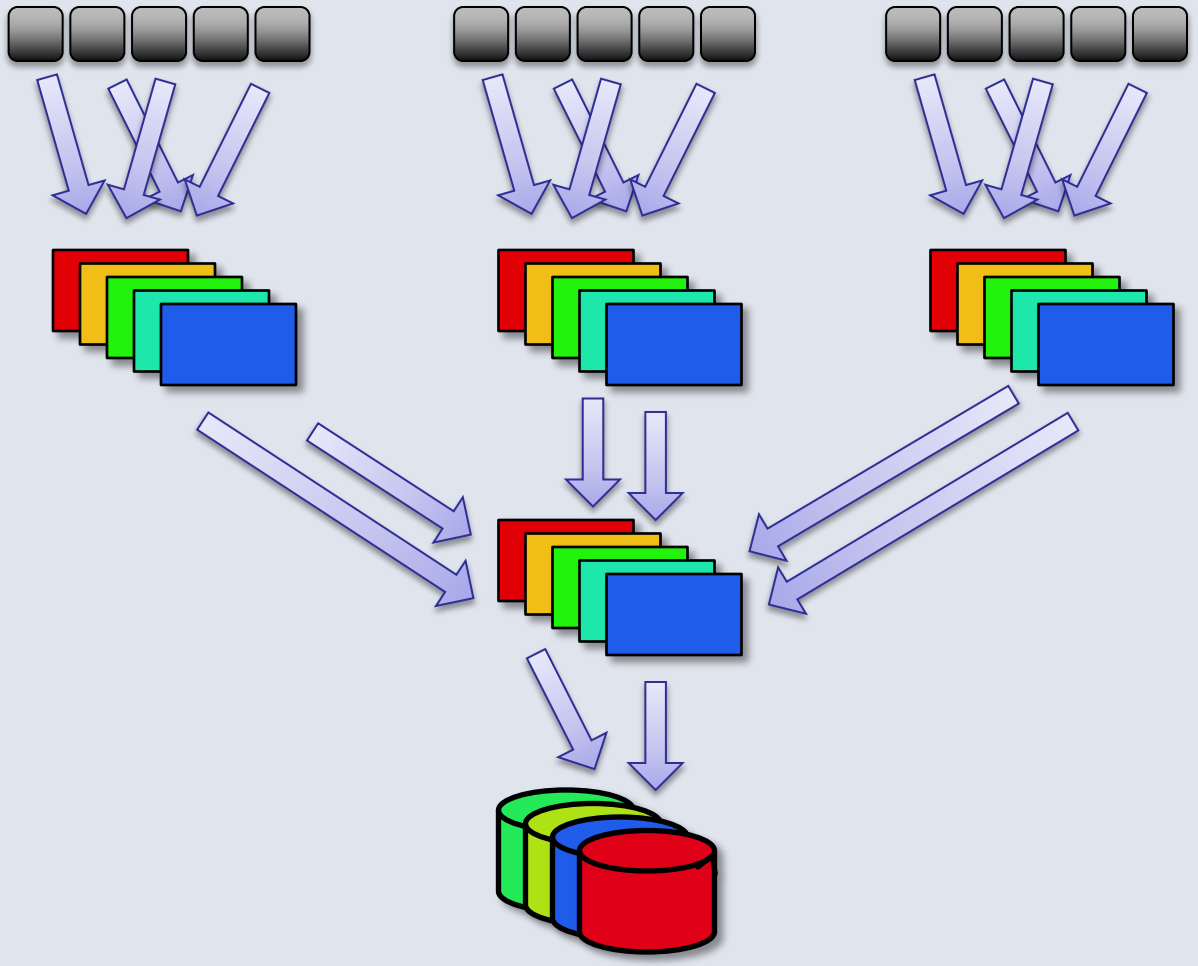
Follower and Leader Caches

Web servers

Follower cache

Leader cache

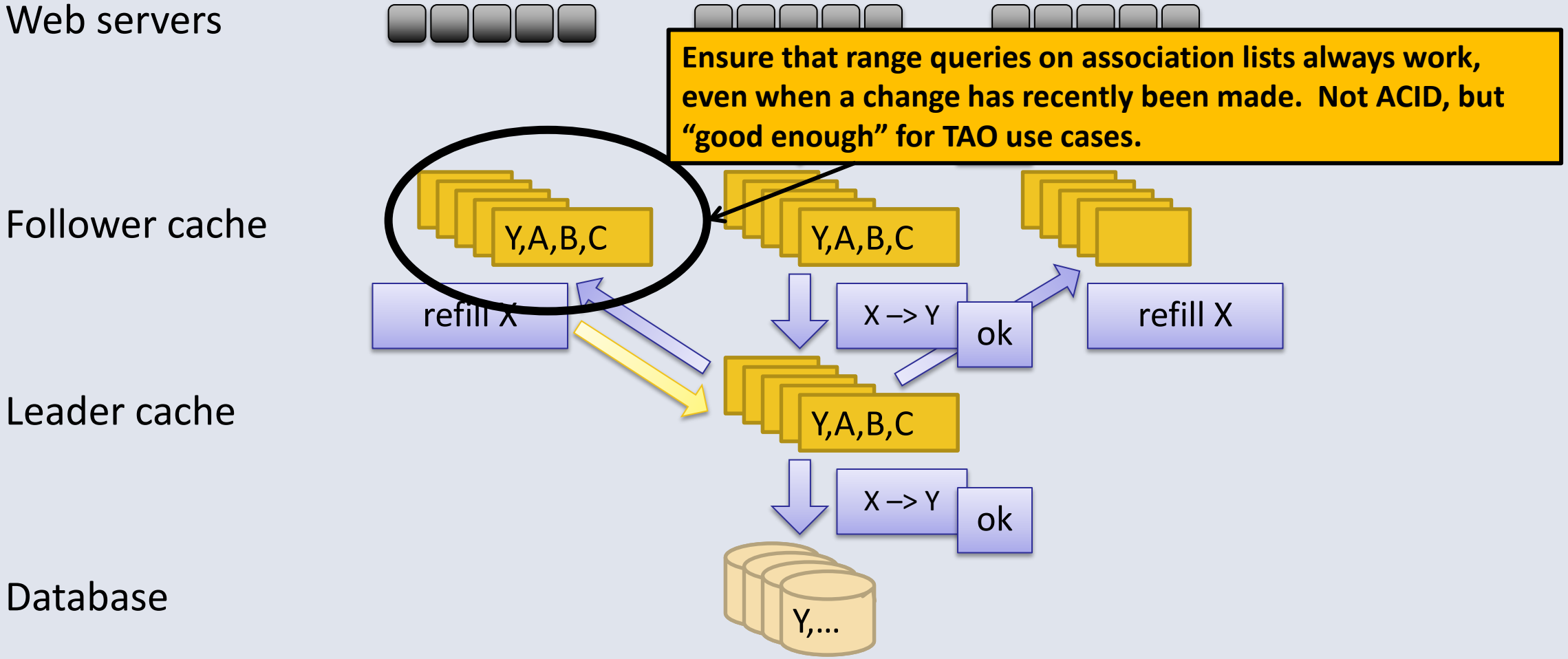
Database



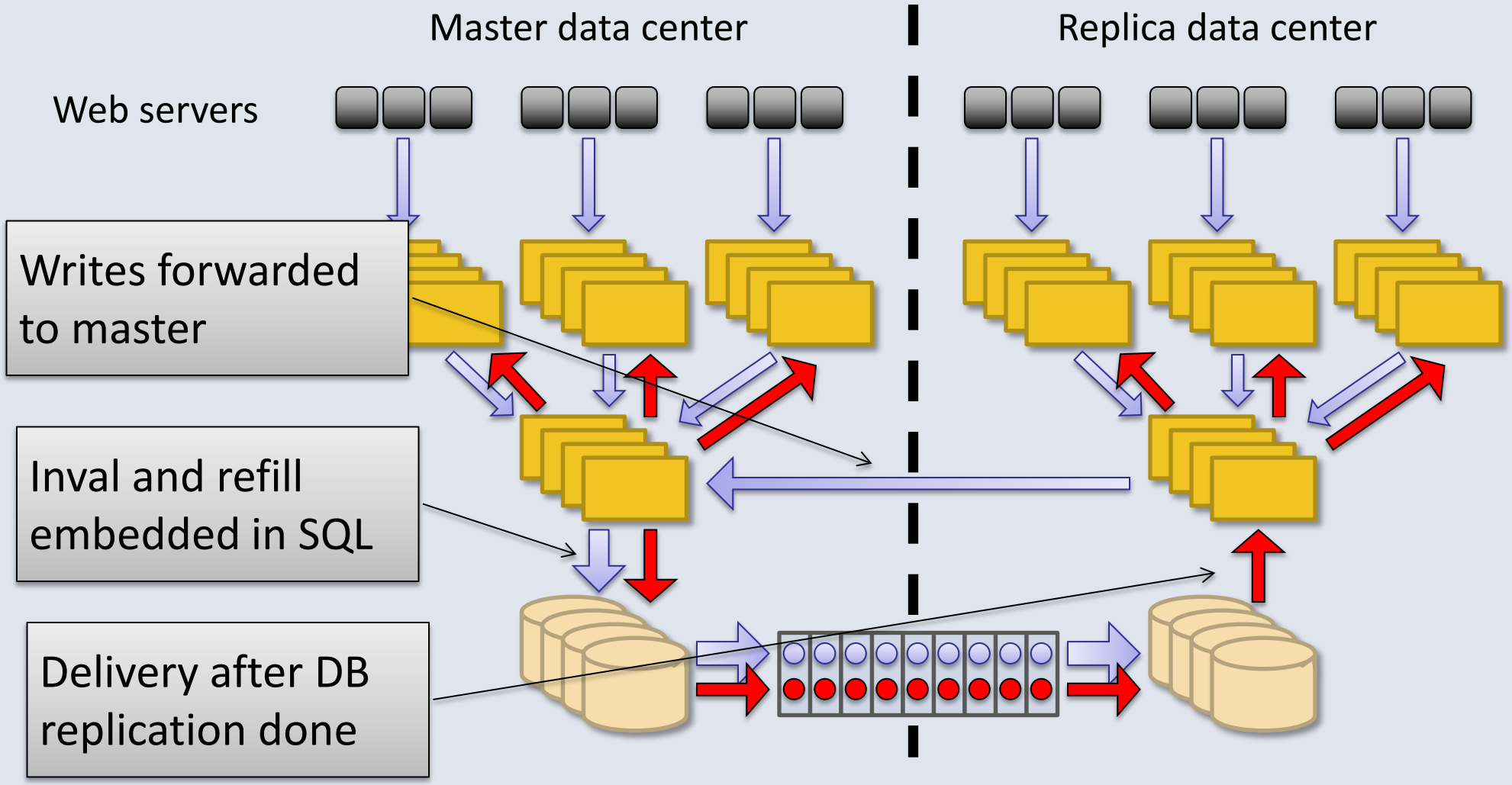
What Are TAO's Goals/Challenges?

- Efficiency at scale
- Low read latency
- Timeliness of writes
- High Read Availability

Write-through Caching – Association Lists



Asynchronous DB Replication



What Are TAO's Goals/Challenges?

- Efficiency at scale
- Low read latency
- Timeliness of writes
- High Read Availability

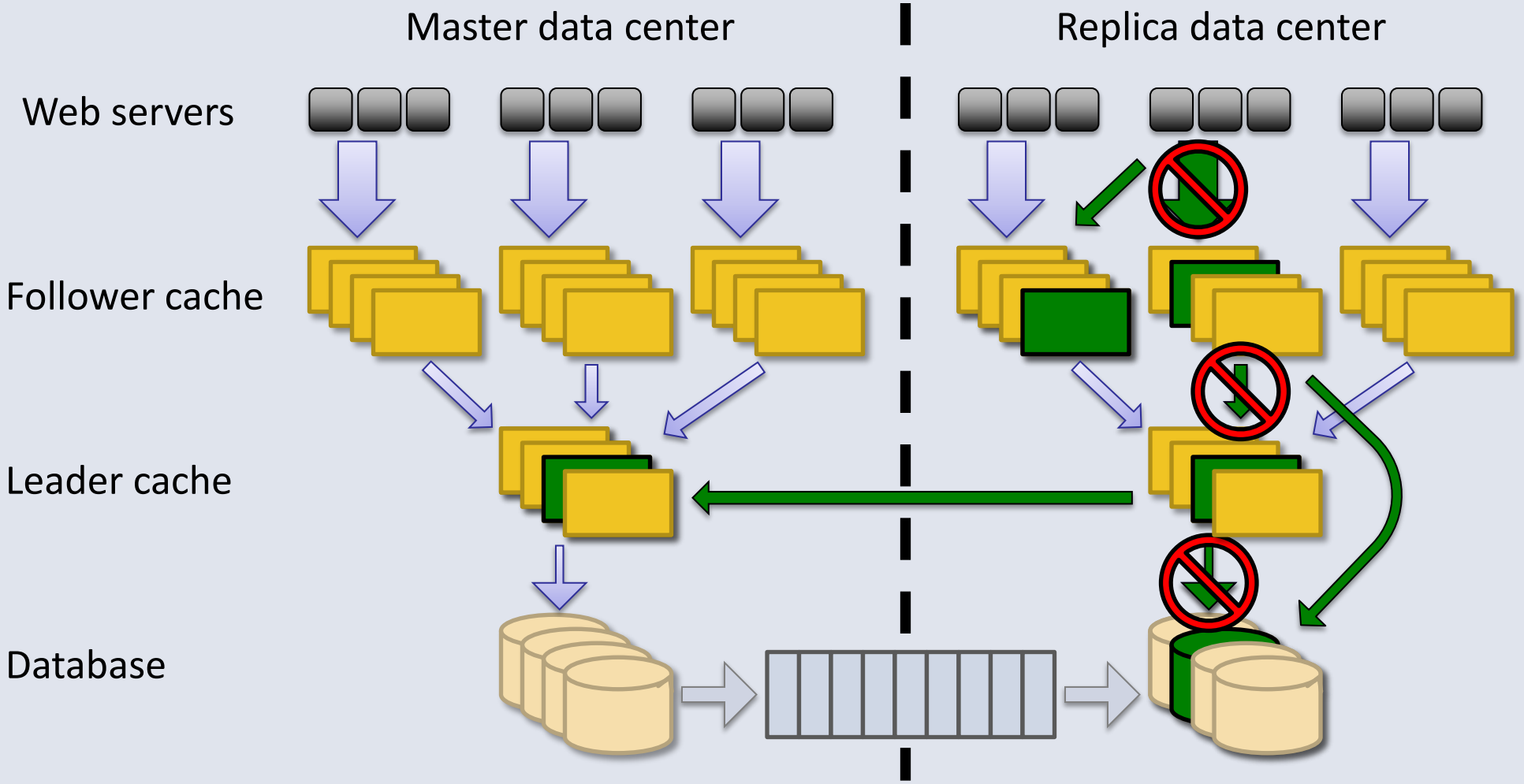
Key Ideas

- TAO has a “normal operations” pathway that offers pretty good properties, very similar to full database transactions.

- But they also have backup pathways for almost everything, to try to preserve updates (like unfollow, or unfriend, or friend, or like) even if connectivity to some portion of the system is disrupted.

- This gives a kind of self-repairing form of fault tolerance. It doesn’t promise perfect transactional atomicity model, yet is pretty close to that.

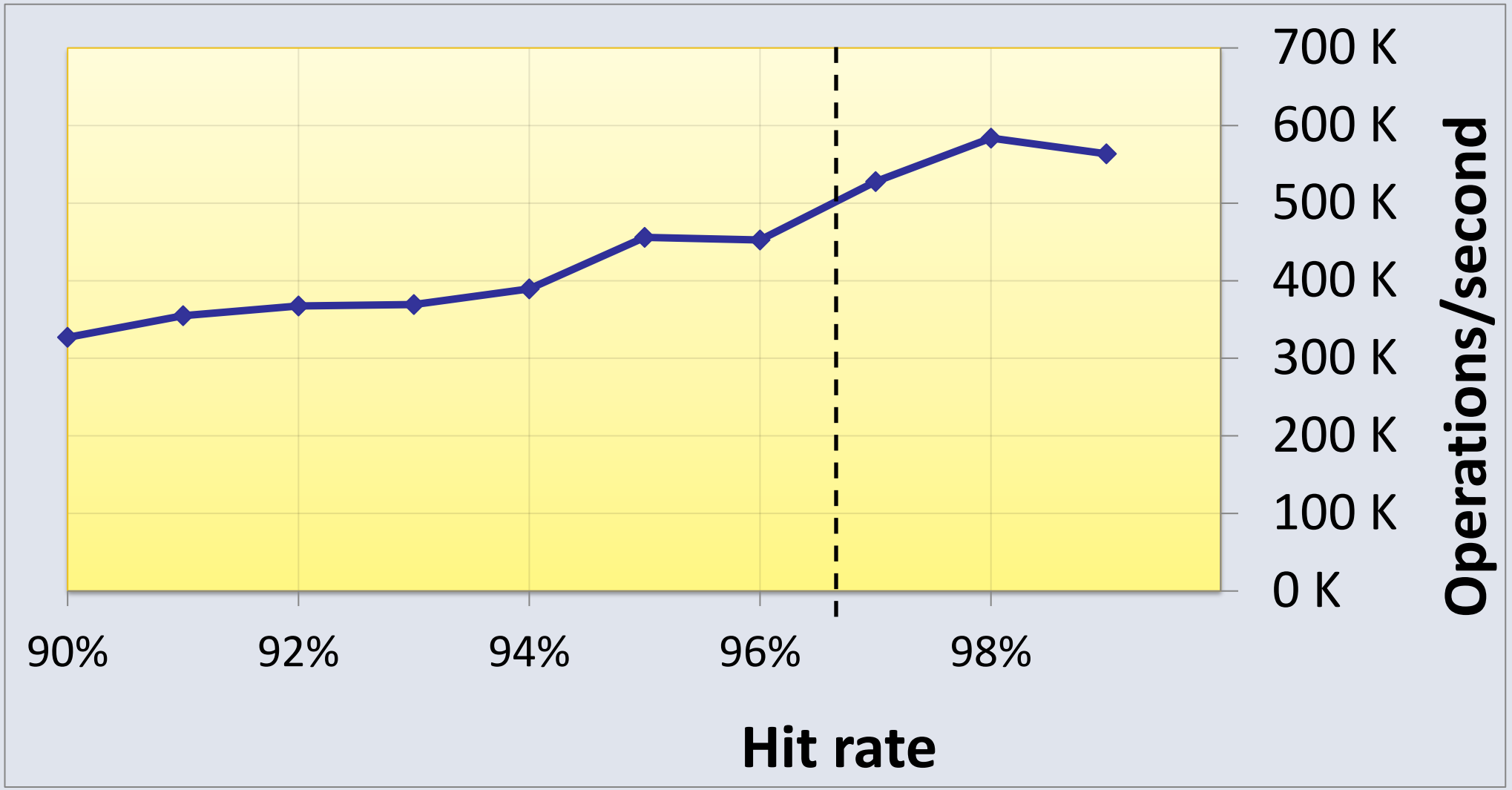
Improving Availability: Read Failover



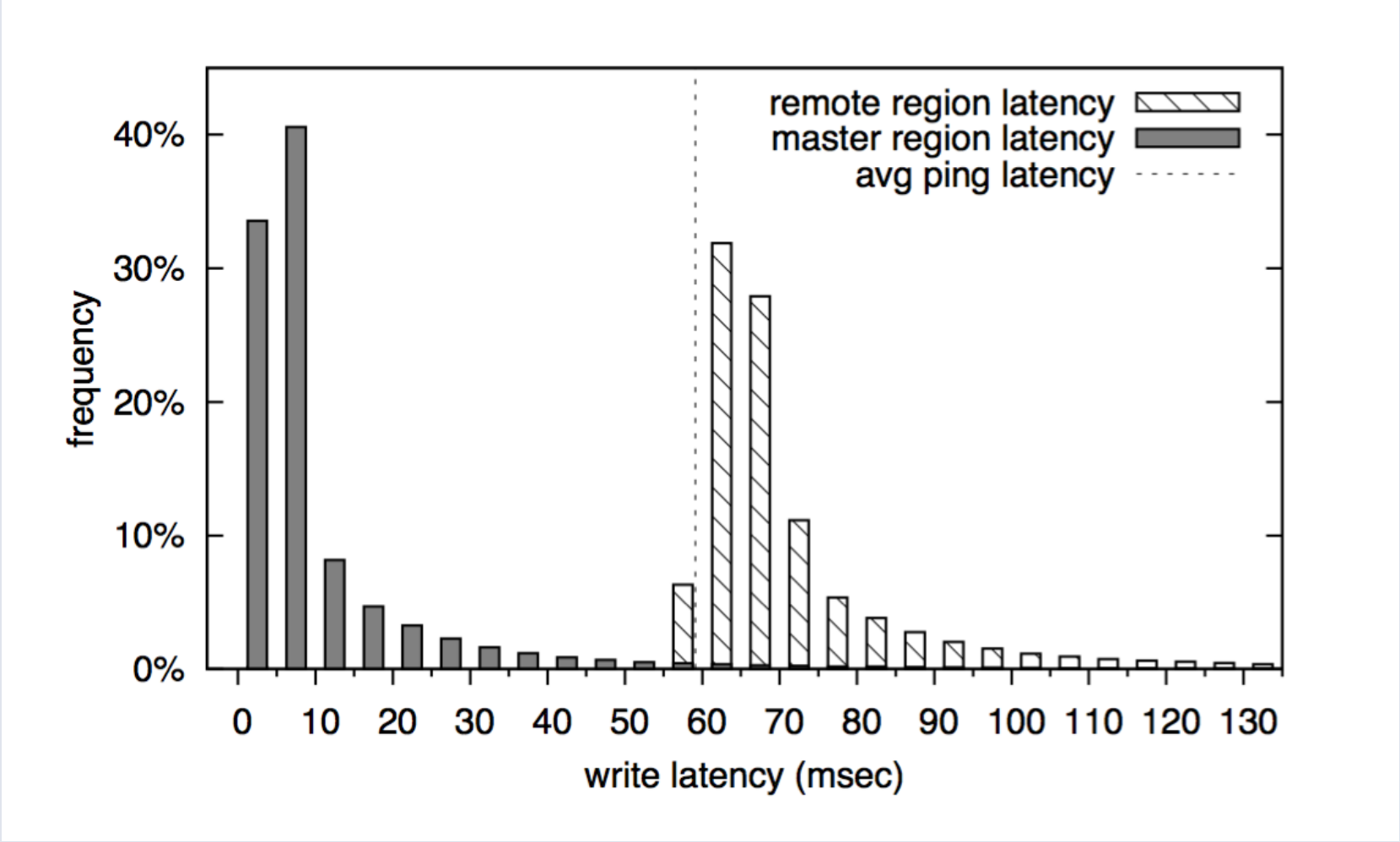
TAO Summary

Efficiency at scale Read latency	• Separate cache and DB • Graph-specific caching • Subdivide data centers
Write timeliness	• Write-through cache • Asynchronous replication
Read availability	• Alternate data sources

Single-server Peak Observed Capacity



Write latency



More In the Paper

- The role of association time in optimizing cache hit rates
- Optimized graph-specific data structures
- Write failover
- Failure recovery
- Workload characterization

Not in the paper but worthwhile to be aware of:

- **Attaching lambdas to TAO, to customize it (EventLoopProxies).** Topic is too big a stretch for us today but is worth knowing about: TAO, like many services in the cloud, can be customized with code that will run when some matching event occurs.

REMINDER: WHY DID WE LOOK AT TAO?

TAO is a best of breed, vendor-supplied cloud service.

- Designed really smartly for speed, with many ideas such as using a graph representation oriented around lists (not relational tables), using a consistency model more like consistent cuts than SQL atomicity, …
- TAO itself is coded in C++. Its interface to the applications is inspired by C++ lambdas.
- Many applications access it by connecting and then retrieving iterators, convenient for Pandas and cpplinq.hpp users
- TAO offers a good "segway" to think about ML "as a service" in clouds

WHAT HAVE WE LEARNED?

We know how to connect to a datacenter over a network

We've seen two examples of major infrastructure services (and others, like the file system, in past lectures)

Now we'll shift focus and start to think about how to *add new ML services* to these cloud ecosystems – ones that already have big services like TAO that we can easily customize with lambdas and compute on using iterators

CONCEPT OF A LAMBDA… IN A DATA CENTER

We know all about lambdas in C++. Often they allow us to take some existing piece of code and give it a customized "action", like a readers and writers template that we give a read action and write action.

Datacenter leaders realized that programming at cloud scale is too hard for normal people to really do from scratch!

So they designed more and more services, and all of them allow you to provide lambdas: plug-ins to customize the service.

MANY APPLICATIONS AT META (AND ELSEWHERE) ARE LAMBDAS!

You as the developer code (or vibe) to create these customizations

Then you (or your vibe partner) will deploy the lambdas by registering them with the appropriate existing cloud services

- Could be a KV store, TAO, the BLOB store, etc
- Your registration action "plugs the lambda in" on some event path
- Now when that kind of event occurs, your logic will be triggered

UPCOMING LECTURES WILL DIVE IN ON THESE IDEAS

First we will learn about the extensible microservices concept and how clouds use it

Then how we build fault-tolerance in availability zone deployments

And then we can dive in on big-data mining for ML, for a few lectures

SELF-TEST

TAO is oriented around a “collections” model, and its APIs focus on iterators.

Consider some typical Facebook tasks, such as identifying friends of a person who share their love for whitewater rafting and highlighting Harry and Sally’s whitewater kayak adventure on their feeds.

How would you code that sort of logic using TAO together with the kinds of collection-oriented computing features discussed in Lecture 11?

SELF-TEST

There are many ways to represent graphs, including the format TAO adopts but also as a matrix in which each (i,j) cell is a Boolean, with 1 if there is an edge from node i to node j, and 0 if not.

In modern ML we often deal with large models, but when TAO was created we had search and social network graphs but not modern ML.

If you were joining the TAO team today, would you advocate for reimplementing TAO as a matrix using this representation? Why or why not? Give one example of a task more easily done using a Boolean graph representation, and one of a task that becomes hard to do.

SELF-TEST

TAO is using a single data structure for posts, likes, liked-by, friend associations, unfriending, comments, etc.

Another option would have been to store each kind of information separately, in some form of relational database table.

List pros and cons of a tabular representation compared against the graph employed in TAO.

Revisit the question about computing using TAO iterators. Would the same tasks be easier or harder to code with a relational database model? Would the actual code run faster, slower, or the same?