Scaling Facebook

The presentation addresses three subjects:

1) What are the difficulties inherent in scaling Facebook?

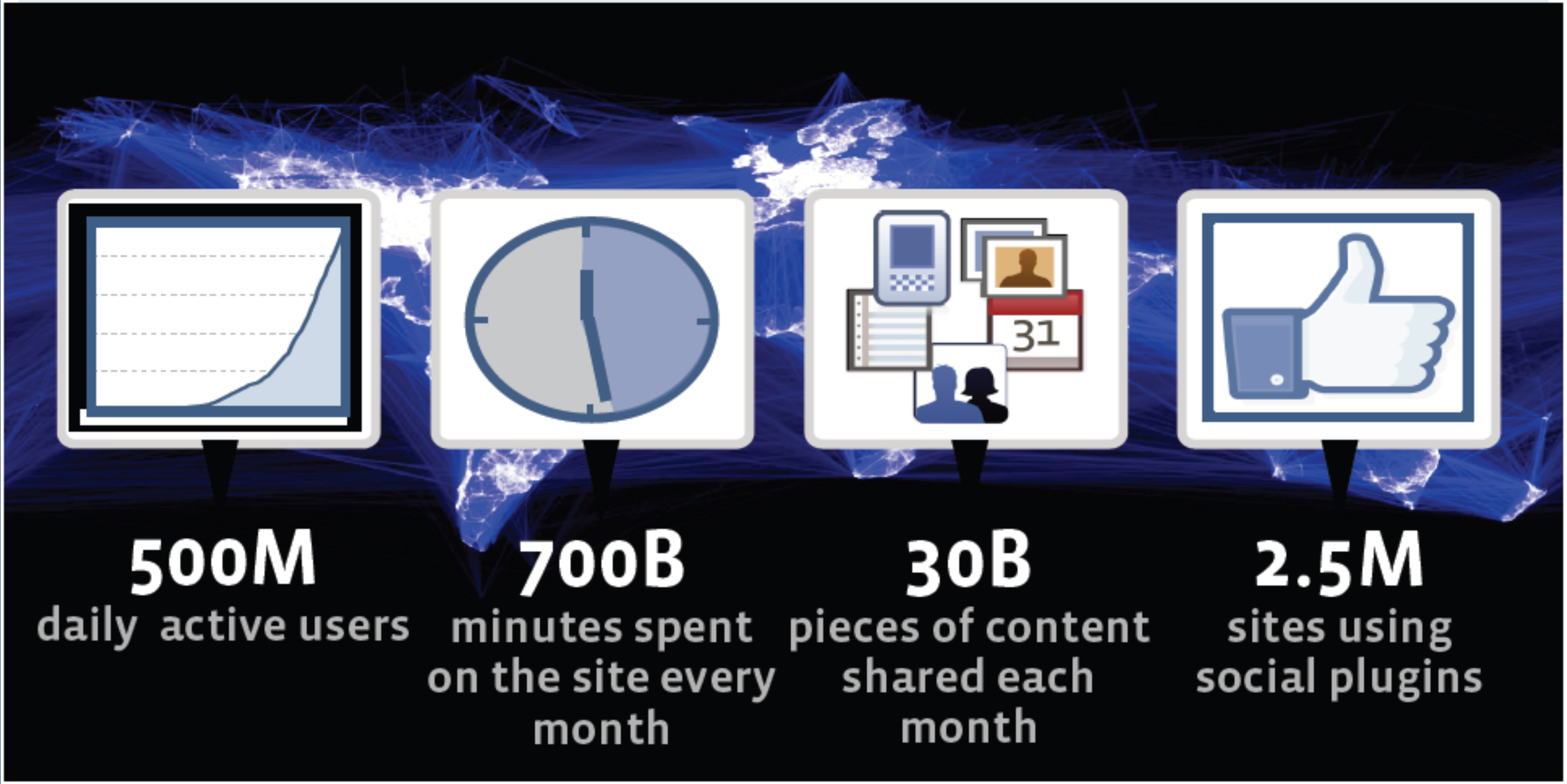
2) How has Facebook's <u>software</u> evolved to meet this growing need?

3) How has Facebook's <u>datacenter</u> evolved to meet this growing need?

- Most of the presentation deals with the first two questions (we'll ignore the third since it isn't central to the class).

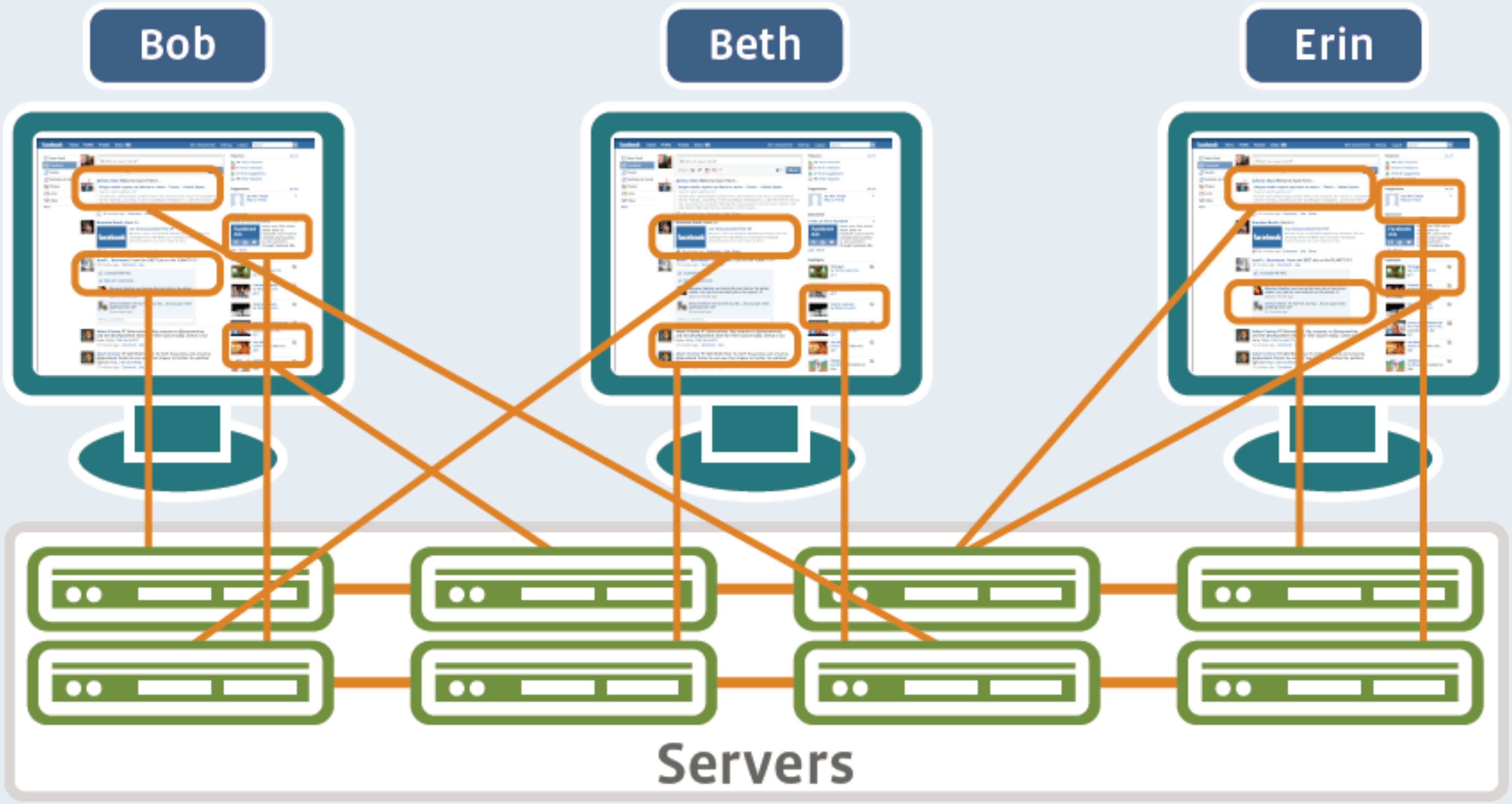
1) What are the difficulties inherent in scaling Facebook?

845M users worldwide



Facebook: The data is interconnected

Common operation: Query the social graph



Additional difficulties:

- There are millions of third party applications/software that interface with Facebook with varying degrees of dependence
- Complex infrastructure – there exists a large variety of storage systems, caching systems and specialized services.

2) How has Facebook's <u>software</u> evolved to meet this growing need?

Facebook’s software can be divided into four broad categories:
- Web
- Services
- Cache
- Storage

Let’s see what optimizations/enhancements have been added at each level!

Web:

Challenge: Servers run on PHP with Zend interpreter -> slow performance.

Solution: Iterative improvements to the PHP “interpreter”:

- ➢ Step 1: HipHop compiler (which an accompanying interpreter).
- ➢ Step 2: HipHop VM, that supports both a compiler and JIT.

Remaining challenges:

- Memory management (partial solution: Copy-On-Write using HipHop)
- Poor instruction cache performance (single massive binary)

Storage:

The presentation provides a case study covering the storage of BLOBS (binary large objects).

Challenge: Storing and retrieving large objects efficiently.

Solution:

- Step 1: Commercial storage, each photo is stored separately. Reliable but inefficient.
- Step 2: Added a caching layer.
- Step 3: A customized hardware storage system that was optimized for 1 IO operation per request (compact index in memory, with metadata and data stored adjacently).
- Step 4: Previous enhancements emphasized retrieval rather than storage. Storing new items efficiently is now the primary focus for future improvements.

Cache:

Initially, Facebook relied primarily on Memcache for caching. Memcache suffered from two primary issues:

- Values are opaque (which prevents optimizations that could reduce network traffic)
- Memcache internals are directly exposed to web layer, which introduces various problems

Tao is an alternative to Memcache that comes with different tradeoffs. For example, it’s better suited to handle graph operations, though it also creates a greater CPU load

Services:

Specialized services are often required when ad-hoc solutions are inefficient. The case study discussed in the presentation deals with the News Feed service.

Challenge: Very large audiences (outgoing), filtering (incoming).

Solution: A “pull” approach, i.e., the writer writes to a single location, and the reader gathers information from multiple locations, and applies filters and ranking algorithms before presenting them. Preferable since this is more flexible and because “…the number of incoming edges is much smaller than the outgoing ones”.

(continued)

- Index (pointing to data) is sharded between various leaves (=nodes).
- Leaves are organized into sets. Each set contains the entire index (redundancy, availability).
- Readers query the leaves within a set for new relevant information, then aggregate and rank the resulting data.

Bottom line: Writes are cheap, reads are expensive.

Discussion

3) How has Facebook’s datacenter evolved to meet this growing need?

Essentially, this is the plan:

- Facebook tries to simplify things by maintaining a relatively small number of “classes” of machines (Web, Services, Cache, Storage)

- Rather than relying on leased datacenters, Facebook is aiming to develop custom datacenters that support these specific classes of machines. This should optimize their performance, and decrease power consumption