facebook

Social Networking at Scale

Sanjeev Kumar
Facebook

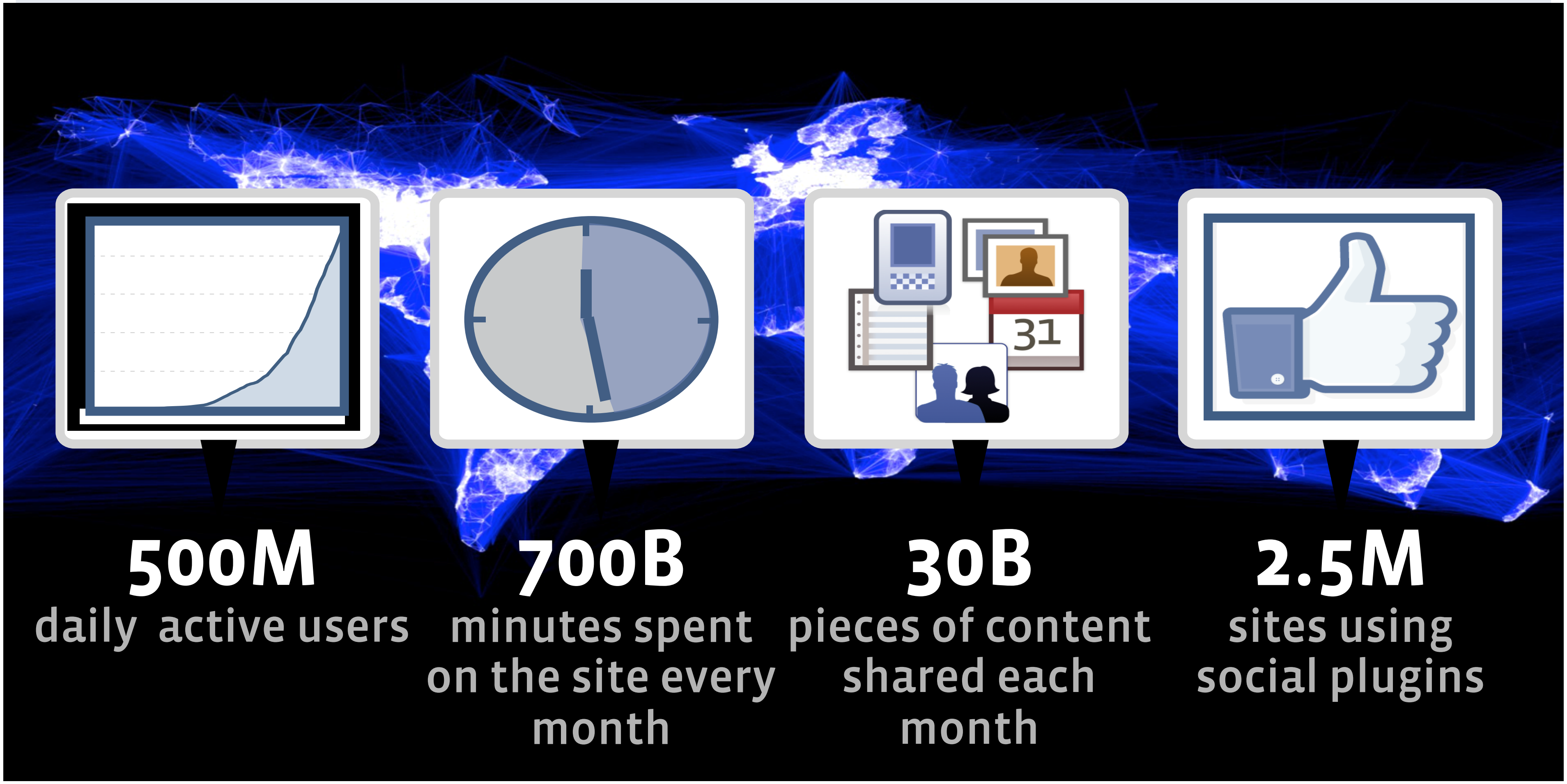
Outline

1 What makes scaling Facebook challenging?

2 Evolution of Software Architecture

3 Evolution of Datacenter Architecture

845M users worldwide



500M
daily active users

700B
minutes spent on the site every month

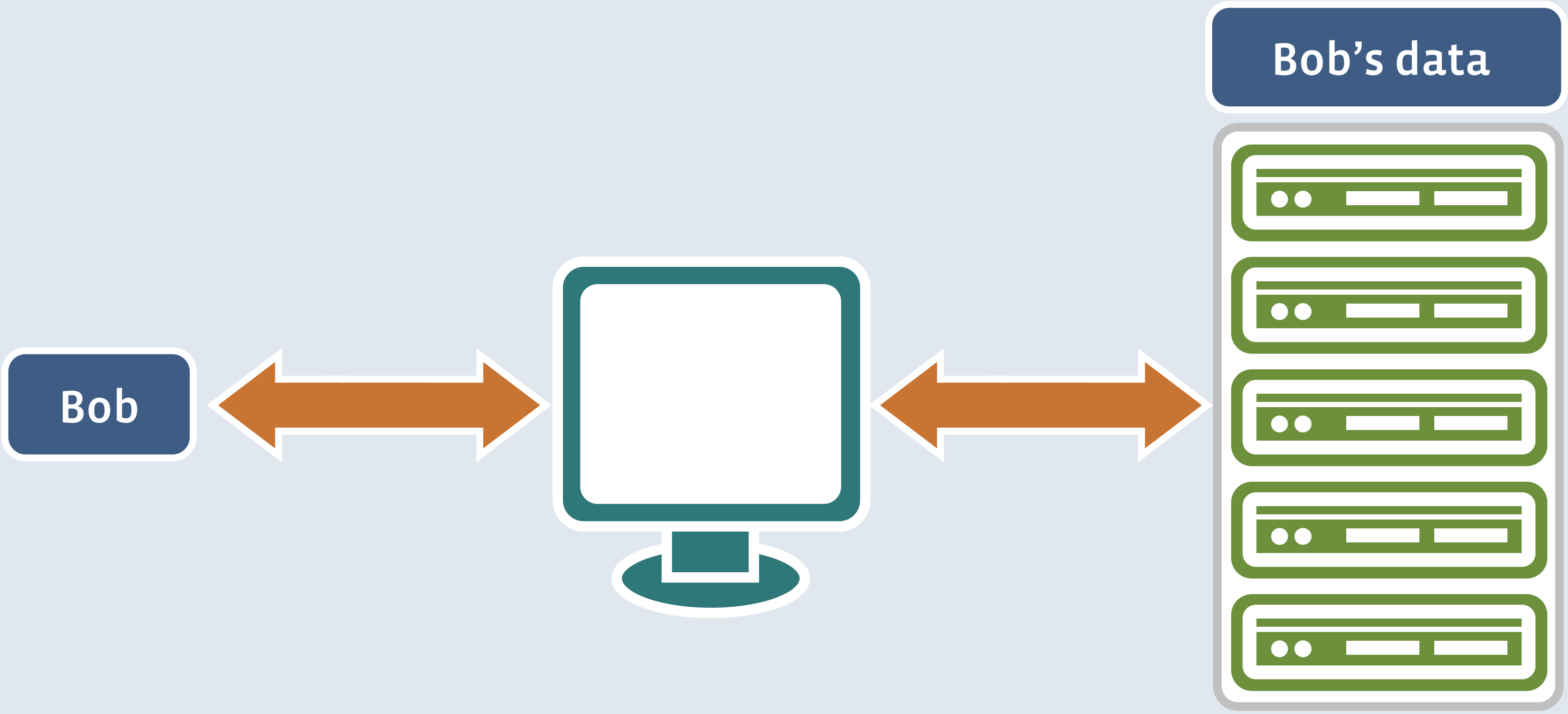
30B
pieces of content shared each month

2.5M
sites using social plugins

What makes scaling Facebook challenging?

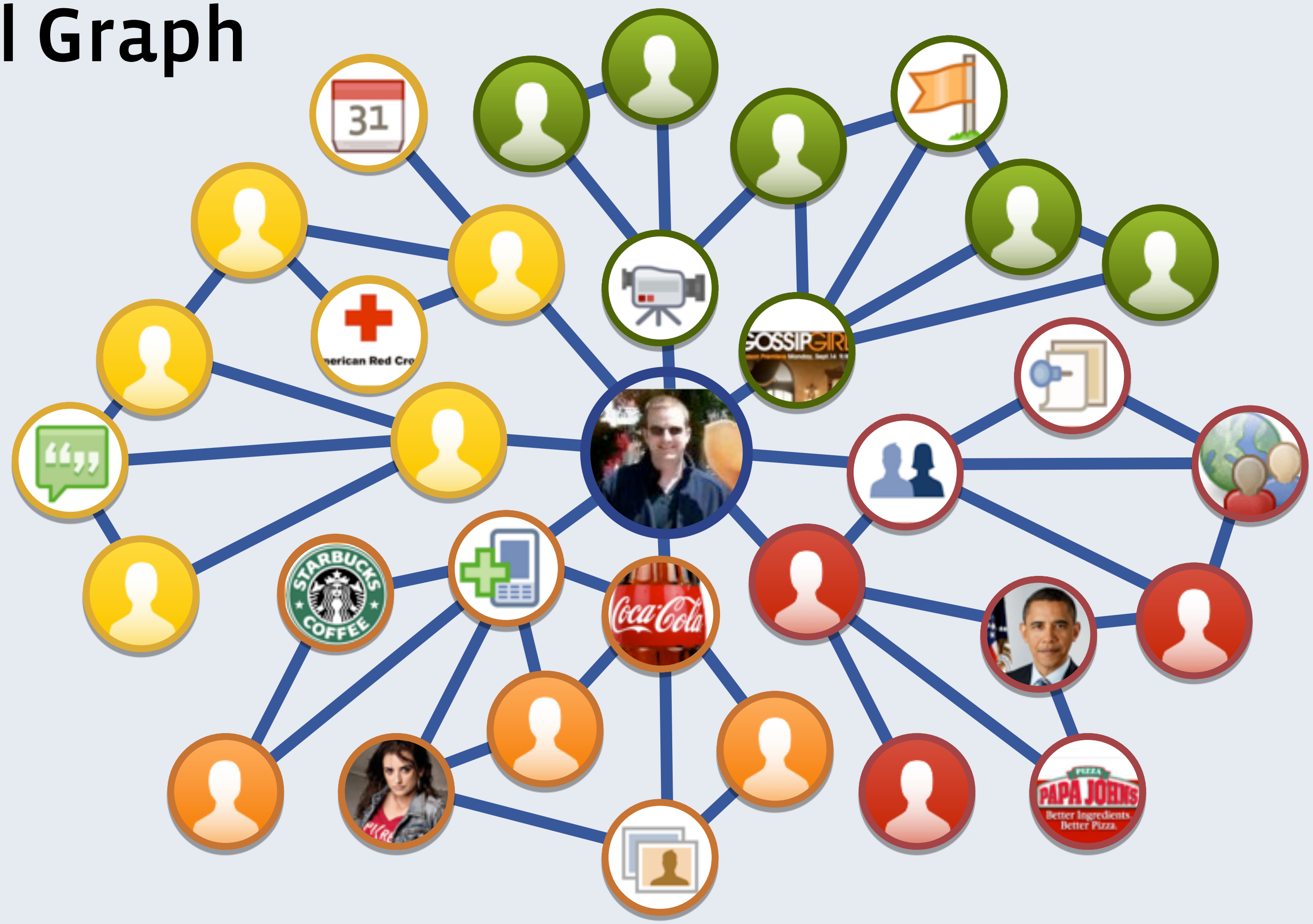
- Massive scale
- Social Graph is central to everything on the site
- Rapidly evolving product
- Complex Infrastructure

Traditional websites



Horizontally scalable

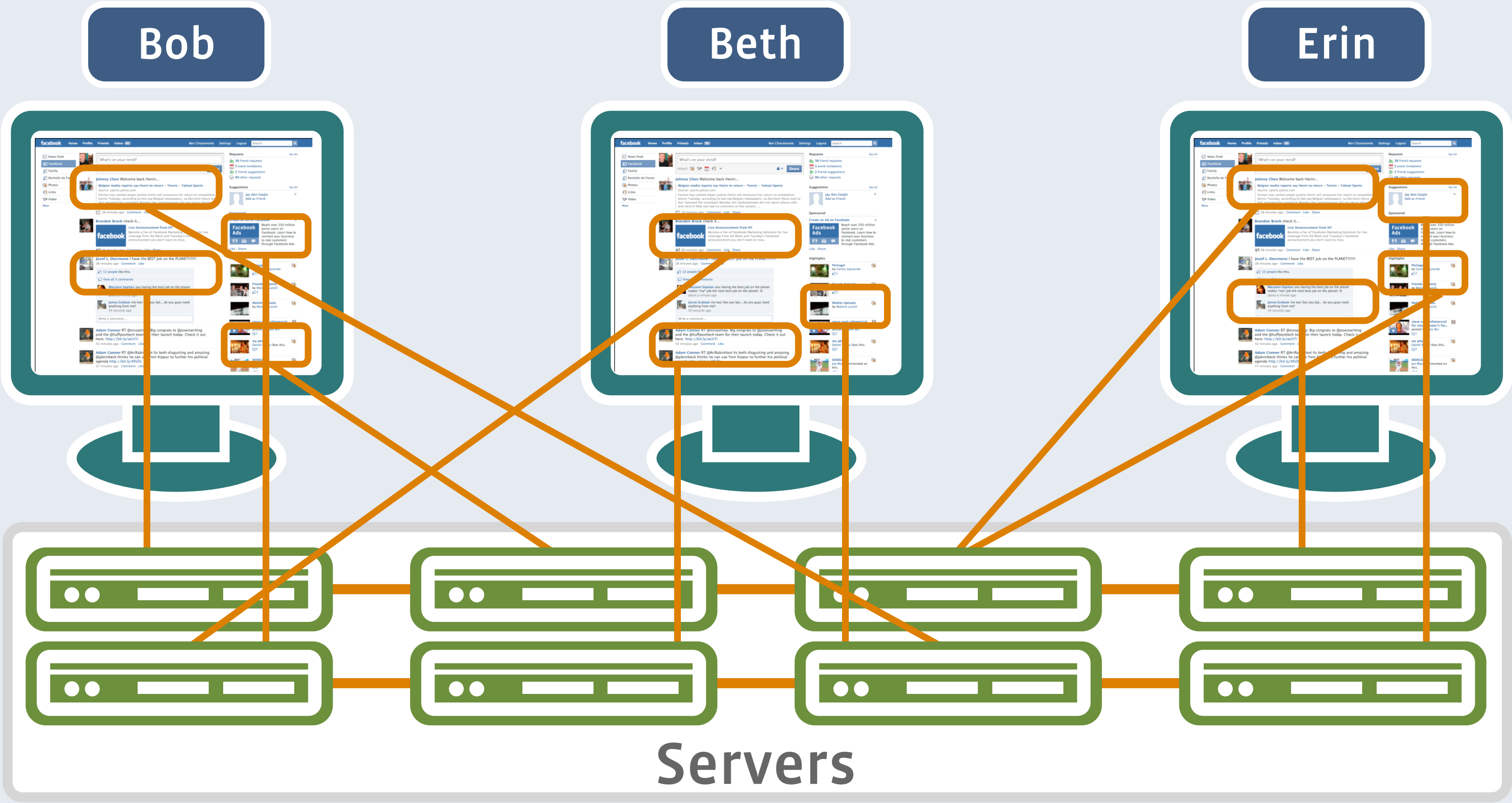
Social Graph



People are only one dimension of the social graph

Facebook: The data is interconnected

Common operation: Query the social graph



Social Graph Cont’d

- Highly connected
  - 4.74 average degree-of-separation between users on Facebook
  - Made denser by our connections to places, interests, etc.
- Examples of Queries on Social Graph
  - What are the most interesting updates from my connections?
  - Who are my connections in real-life who I am not connected to on Facebook?
  - What are the most relevant events tonight near me and related to my interests? Or that my friends are going to?

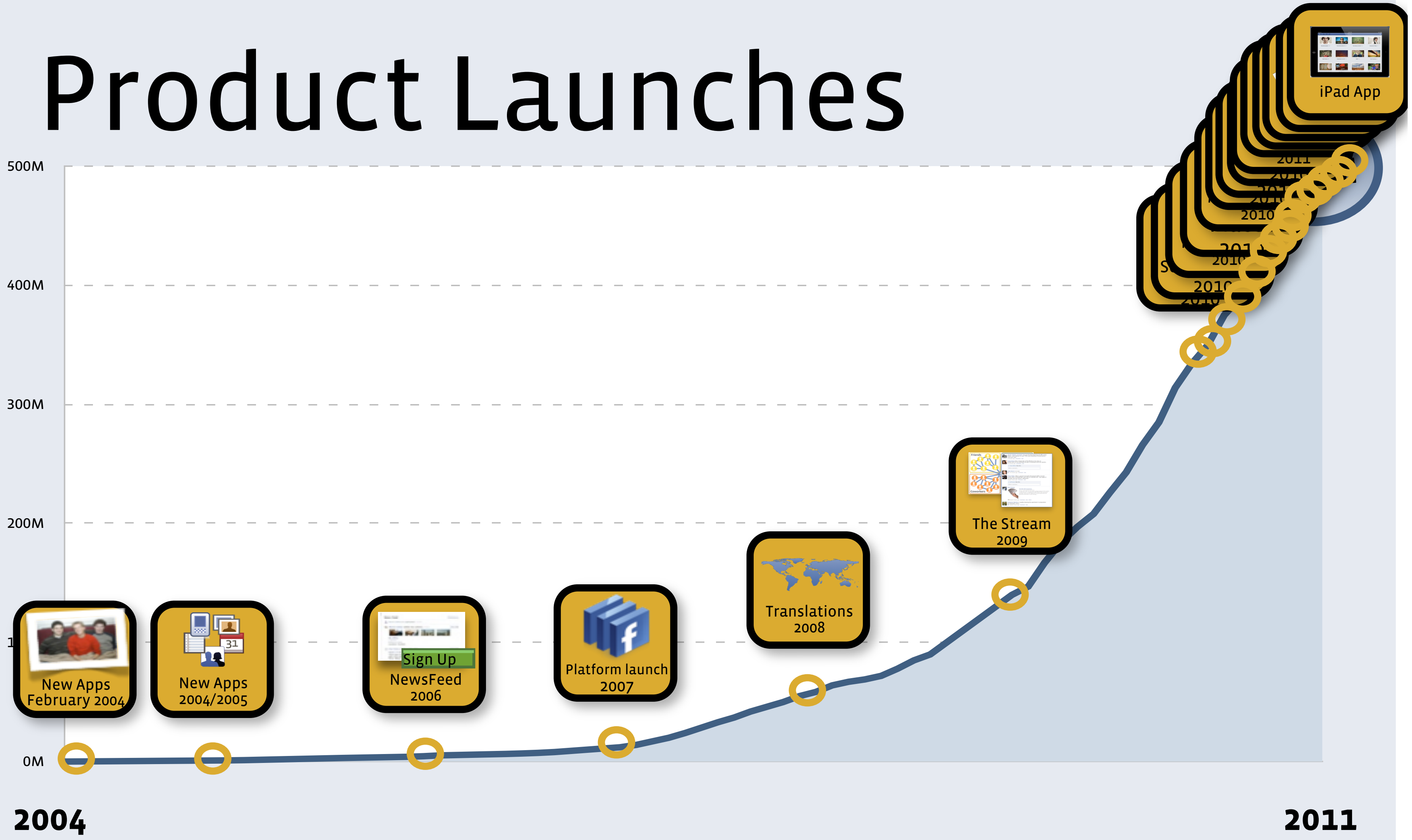
Social Graph Cont'd

- **System Implications of Social Graph**
  - Expensive to query
  - Difficult to partition
  - Highly customized for each user
  - Large working sets (Fat tail)

What makes scaling Facebook challenging?

- Massive scale
- Social Graph: Querying is expensive at every level
- Rapidly evolving product
- Complex Infrastructure

Product Launches



Rapidly evolving product

- Facebook is a platform
  - External developers are innovating as well
- One integrated product
  - Changes in one part have major implications on other parts
    - For e.g. Timeline surfaces some of the older photos
- **System Implications**
  - Build for flexibility (avoid premature optimizations)
  - Revisit design tradeoffs (they might have changed)

What makes scaling Facebook challenging?

- Massive scale
- Social Graph: Querying is expensive at every level
- Rapidly evolving product
- Complex Infrastructure

Complex infrastructure

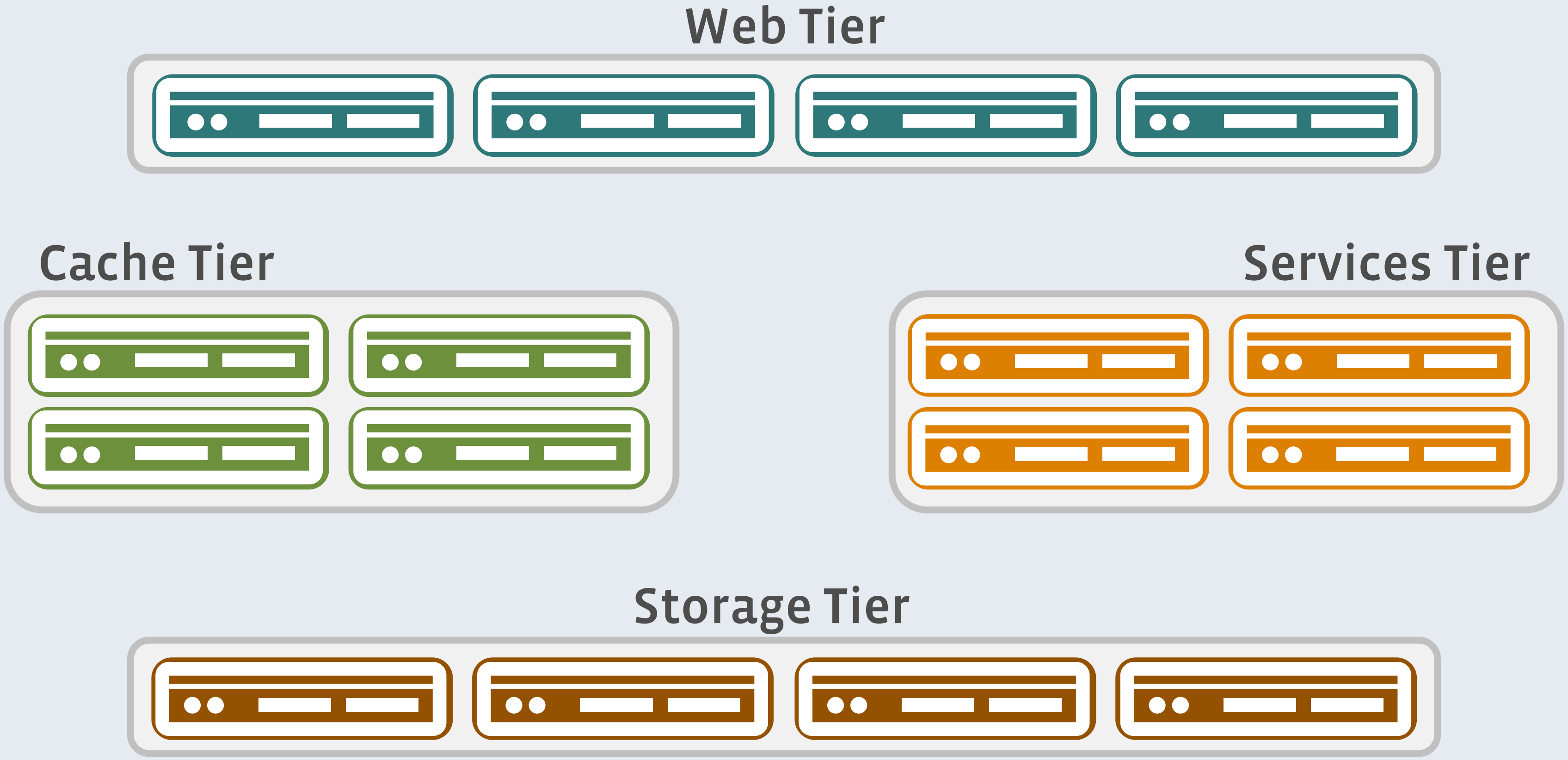
- Large number of Software components
  - Multiple Storage systems
  - Multiple Caching Systems
  - 100s of specialized services
- Often deploy cutting-edge hardware
  - At our scale, we are early adopters of new hardware
- Failure is routine
- **Systems implications**
  - Keep things as simple as possible

Outline

1. What makes scaling Facebook challenging?
2. Evolution of Software Architecture
3. Evolution of Datacenter Architecture

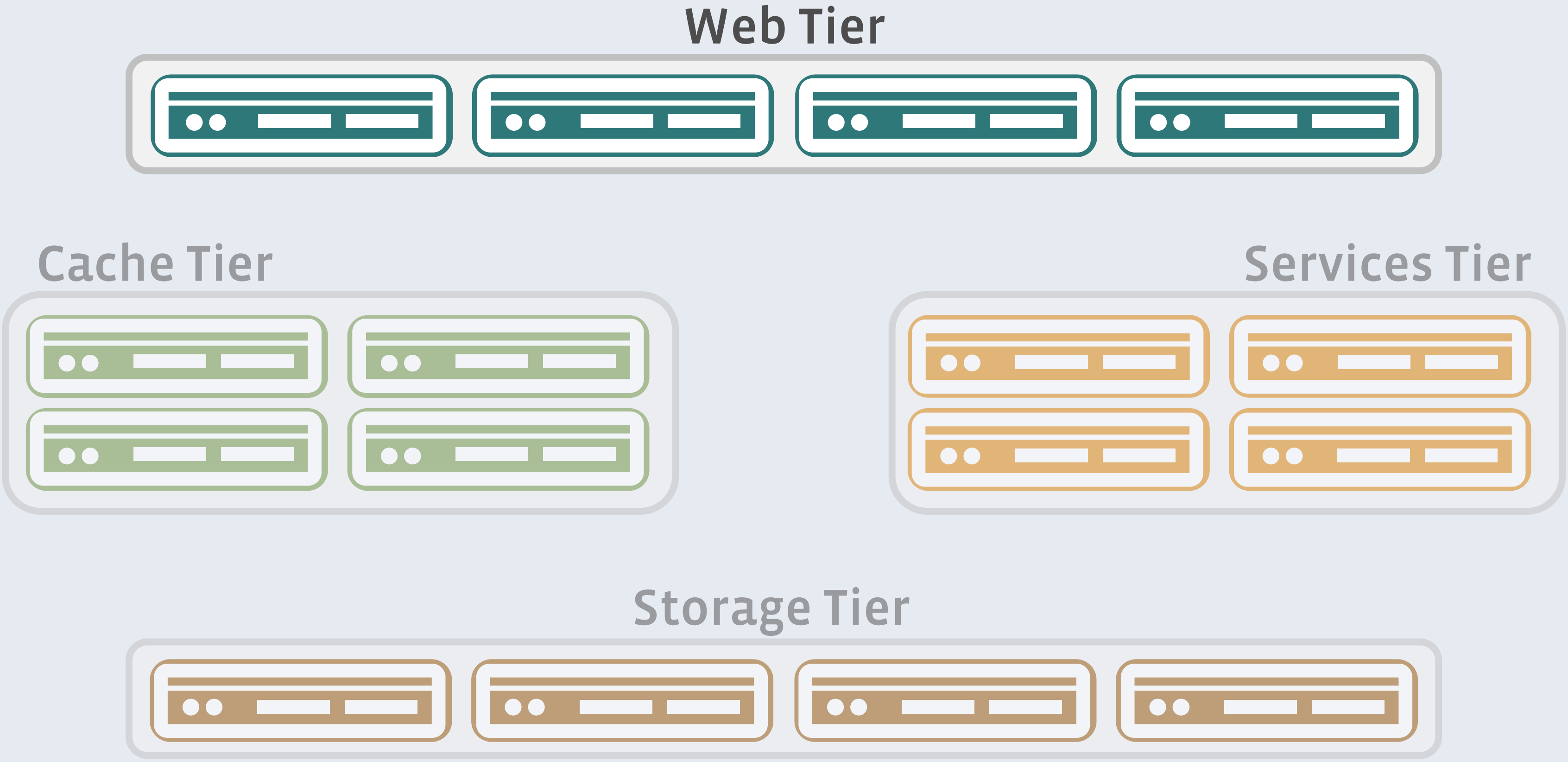
Evolution of the Software Architecture

Evolution of each of these 4 tiers



Evolution of the Software Architecture

Evolution of Web Tier

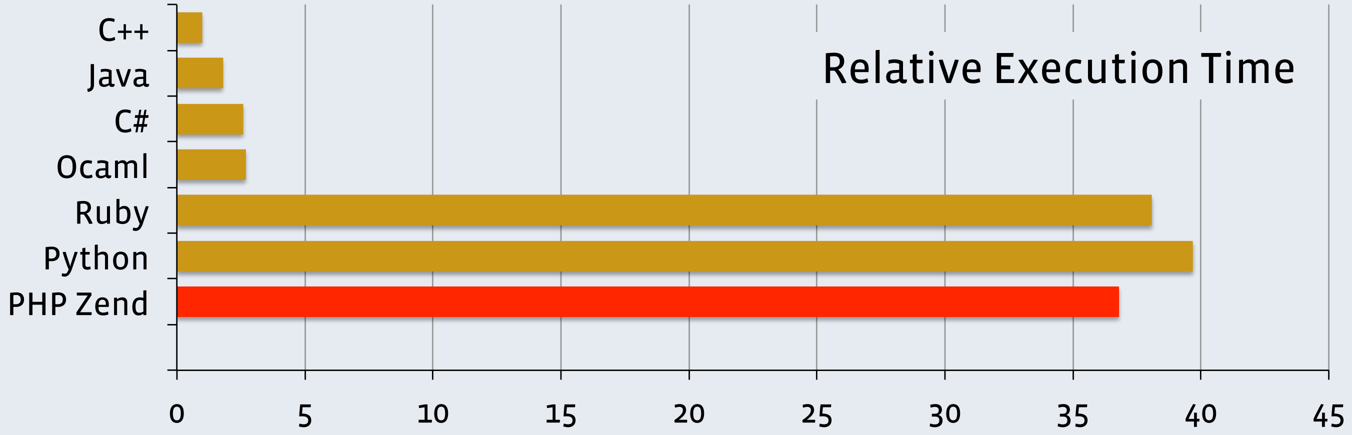


Web Tier

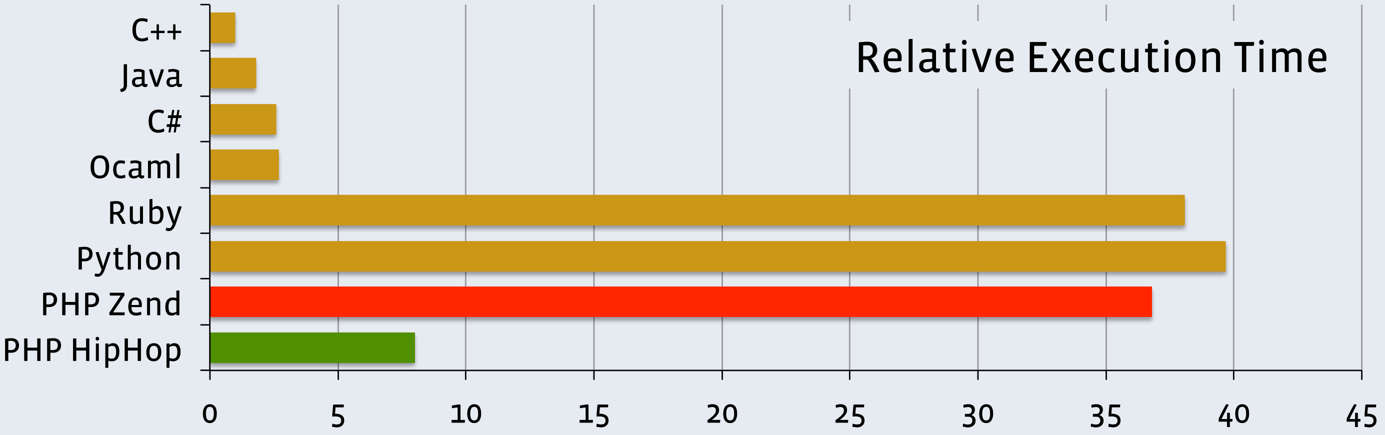
- **Stateless** request processing
  - **Gather Data**: from storage tiers
  - **Transform**: Ranking (for Relevance) and Filtering (for Privacy)
  - **Presentation**: Generate HTML
- Runs PHP code
  - Widely used for web development
  - Dynamically typed scripting language
- Integrated product ➔ One single source tree for all the entire code
  - Same “binary” on every web tier box
- **Scalability**: Efficiently process each request

Generation 1: Zend Interpreter for PHP

- Reasonably fast (for an interpreter)
- Rapid development
  - Don’t have to recompile during testing
- **But:** at scale, performance matters

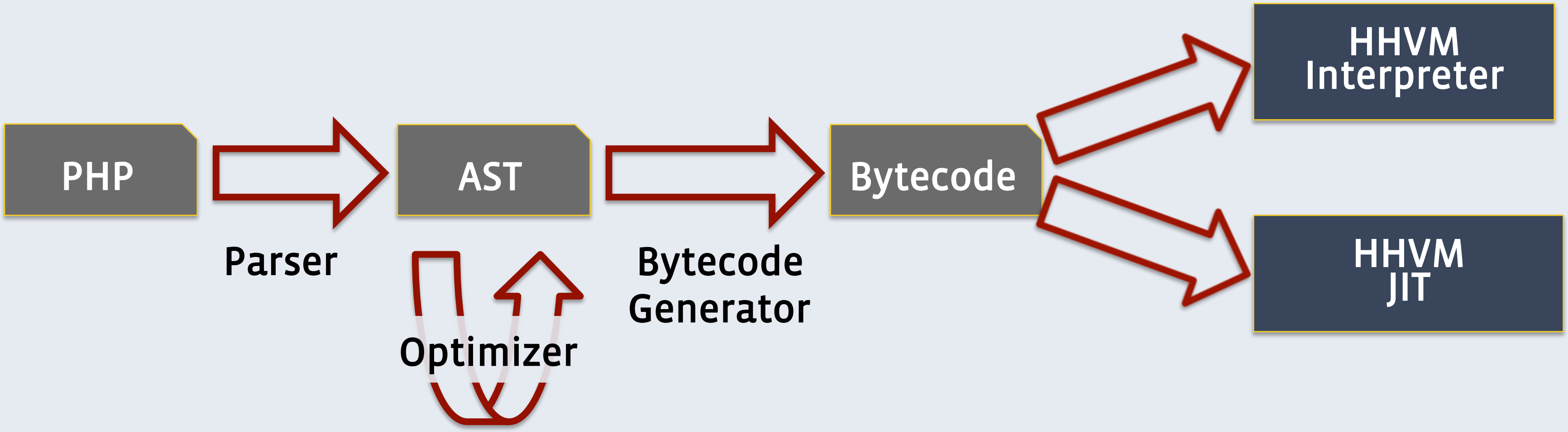


Generation 2: HipHop Compiler for PHP



- Technically challenging, Impressive gains, Still room for improvement
- **But:** takes time to compile (slows down development)
  - Solution: HipHop interpreter
    - **But:** Interpreter and compiler sometimes disagree
    - Performance Gains are slowing. Can we improve performance further?

Generation 3: HipHop Virtual Machine



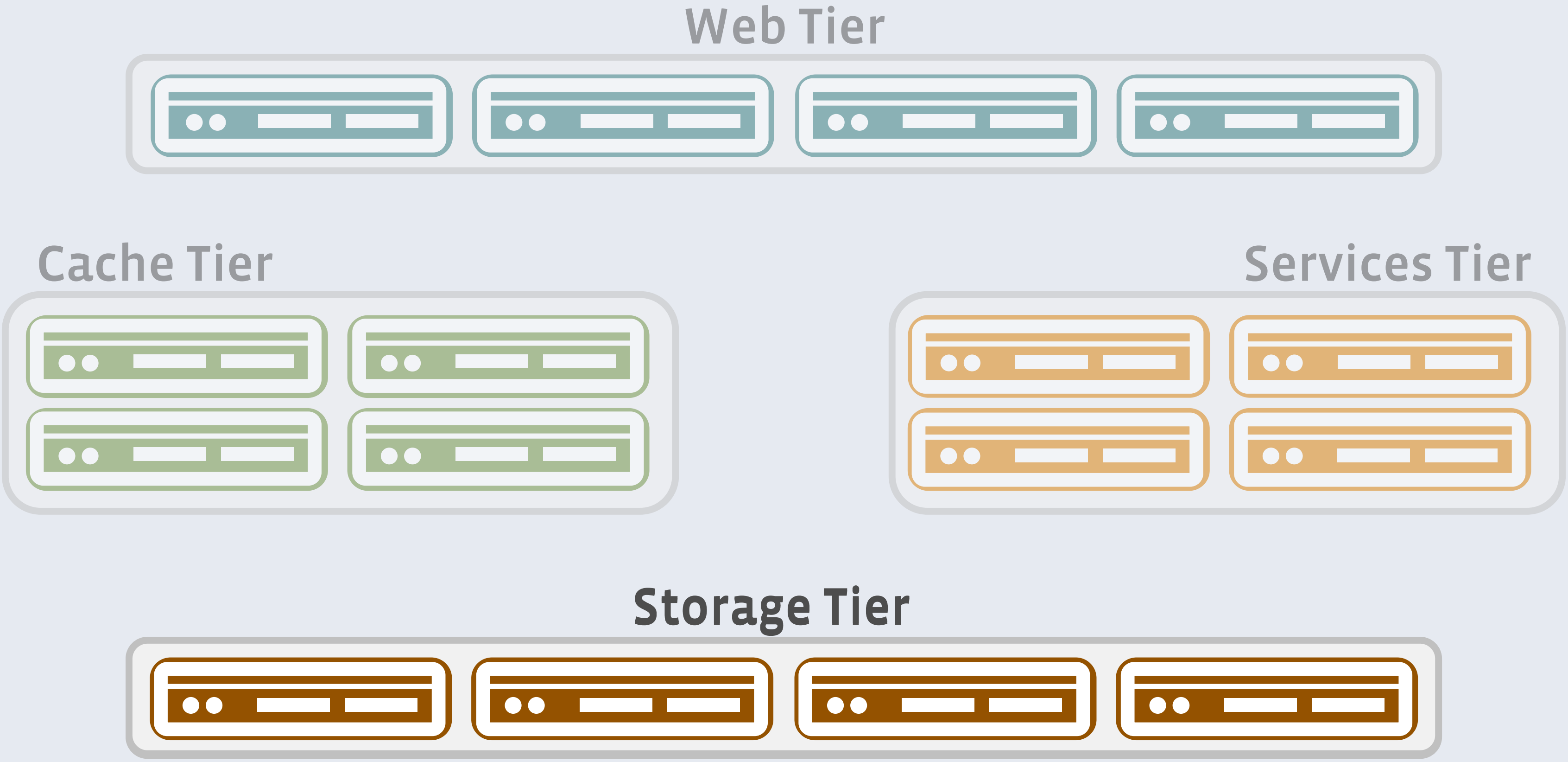
- Best of both worlds
  - Common path, well-specified bytecode semantics
  - Potential performance upside from dynamic specialization
- Work-In-Progress

Web Tier Facts

- Execution time only a small factor in user-perceived performance
  - Can potentially use less powerful processors
  - Throughput matters more than latency (True for other tiers as well)
- Memory management (allocation/free) is a significant remaining cost
  - Copy-on-Write in HipHop implementation
- Poor Instruction Cache Performance
  - Partly due to the one massive binary
- Web load predictable in aggregate
  - Can use less dynamic techniques to save power
  - Potentially even turn off machines. Failure rates is an open question?

Evolution of the Software Architecture

Evolution of Storage Tier



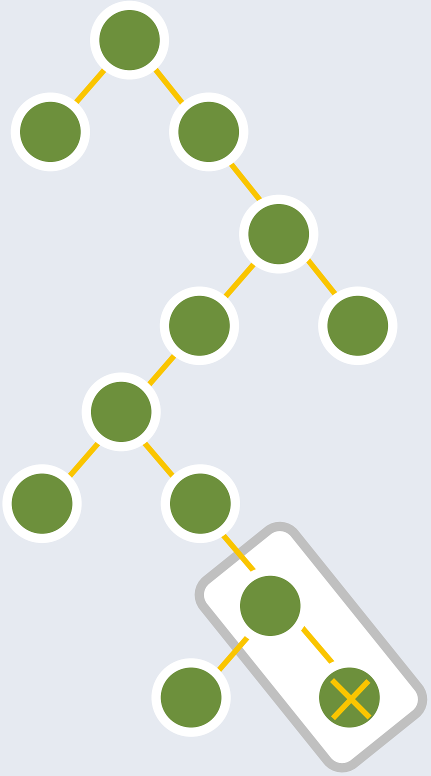
Evolution of a Storage Tier

- Multiple storage systems at Facebook
  - MySQL
  - HBase (NoSQL)
  - **Haystack (for BLOBS)** ←
- Case Study: BLOB storage
  - BLOB: Binary Large Objects (Photos, Videos, Email attachments, etc.)
    - Large files, No updates/appends, Sequential reads
  - **More than 100 petabytes**
  - **250 million photos uploaded per day**

Generation 1: Commercial Filers

- New Photos Product
- First build it the easy way
  - Commercial Storage Tier + HTTP server
  - Each Photo is stored as a separate file
- Quickly up and running
  - Reliably Store and Serve Photos
- **But**: Inefficient
  - Limited by IO rate and not storage density
  - Average 10 IOs to serve each photo
  - Wasted IO to traverse the directory structure

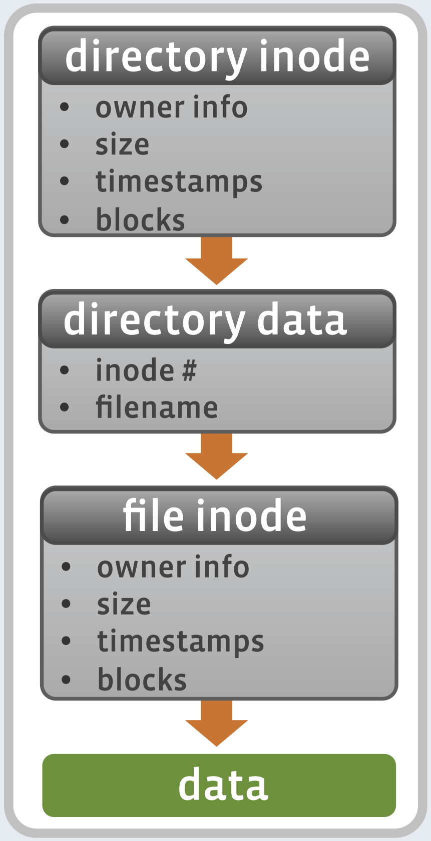
NFS Storage



Generation 2: Gen 1 Optimized

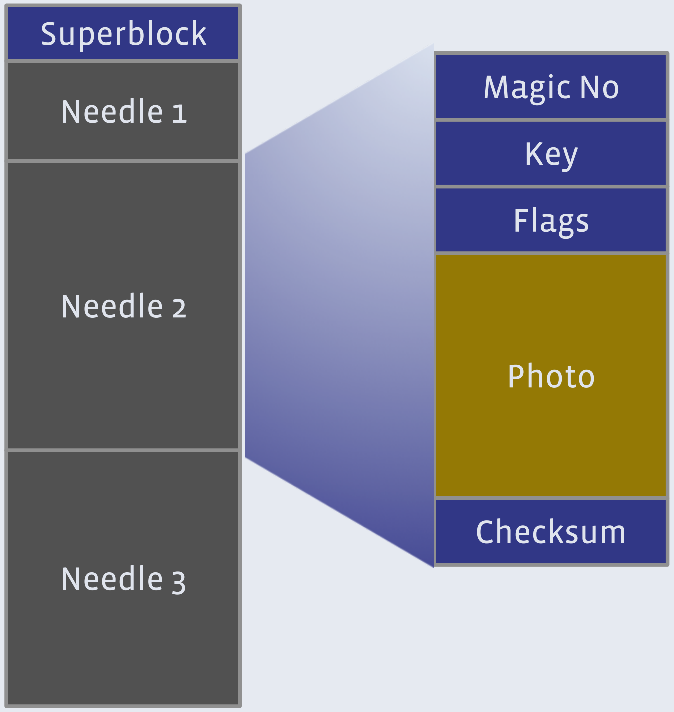
- Optimization Example:
  - Cache NFS handles to reduce wasted IO operations
- Reduce the number of IO operations per photo by 3X
- **But**:
  - **Still expensive**: High end storage boxes
  - **Still inefficient**: Still IO bound and wasting IOs

NFS Storage Optimized



Generation 3: Haystack [OSDI'10]

- Custom Solution
  - Commodity Storage Hardware
  - Optimized for 1 IO operation per request
    - **File system on top of a file system**
    - Compact Index in memory
    - Metadata and data laid out contiguously
- Efficient from IO perspective
- **But**:
  - Problem has changed now



Single Disk IO to read/write a photo

Generation 4: Tiered Storage

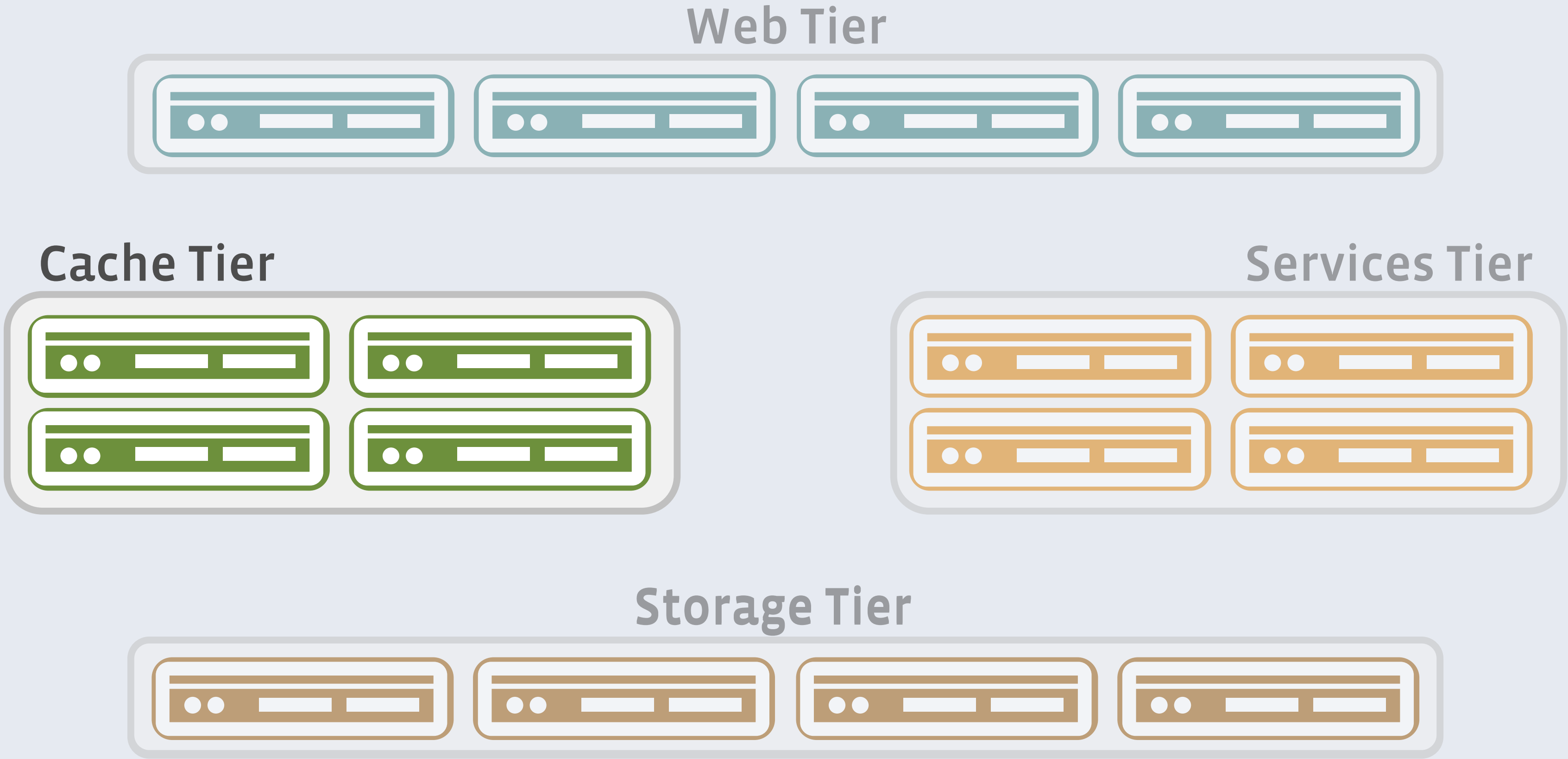
- Usage characteristics
  - Fat tail of accesses: everyone has friends ☺
  - A large fraction of the tier is no longer IO limited (new)
    - **Storing efficiency** matters much more than **serving efficiency**
- Approach: Tiered Storage
    - Last layer optimized for **storage efficiency** and **durability**
    - Fronted by caching tier optimized for **serving efficiency**
- Working-In-Progress

BLOB Storage Facts

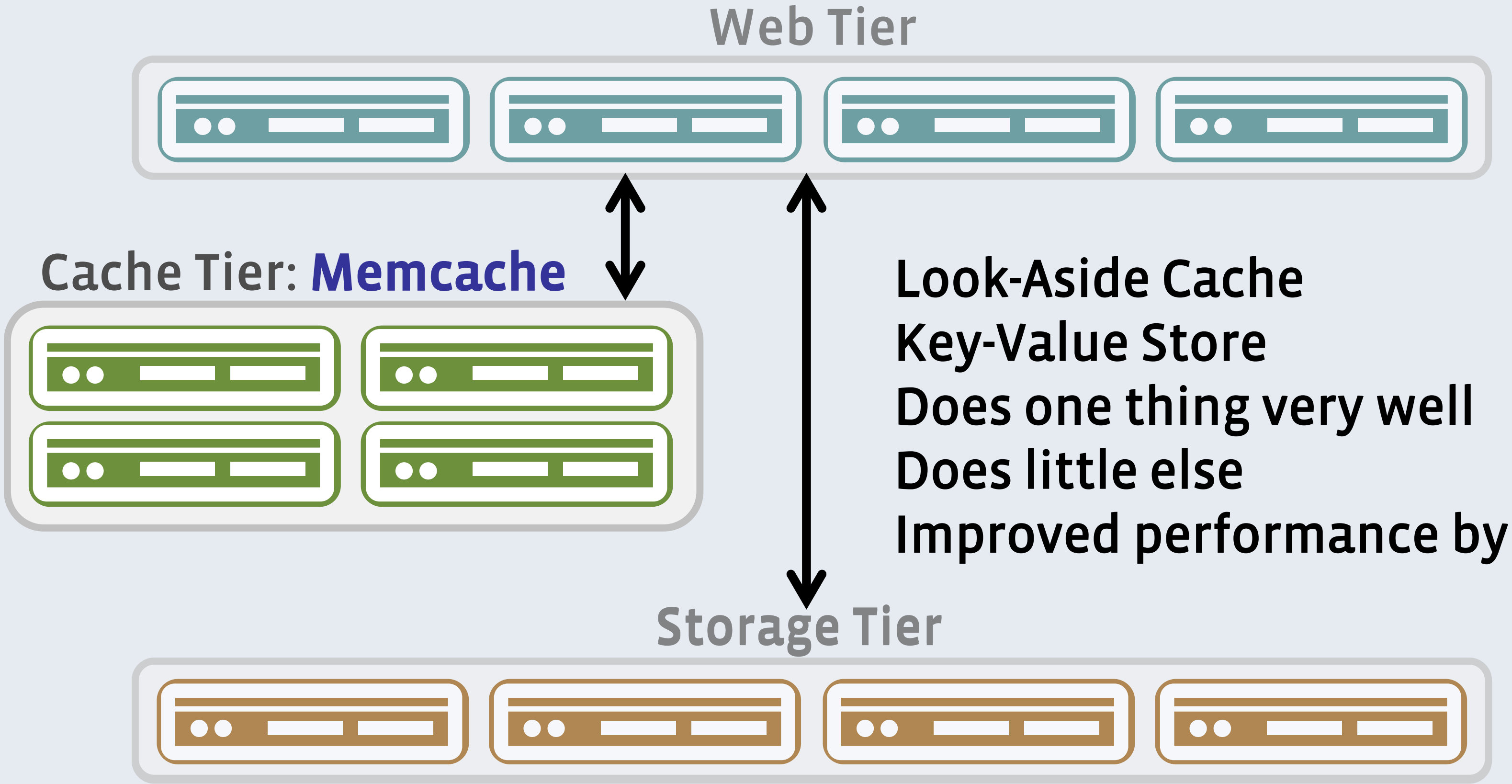
- Hot and Warm data. Little cold data.
- Low CPU utilization
  - Single digit percentages
- Fixed memory need
  - Enough for the index
  - Little use for anything more
- Next generation will use denser storage systems
  - Do we even bother with hardware raid?
  - Details to be publicly released soon

Evolution of the Software Architecture

Evolution of Cache Tier



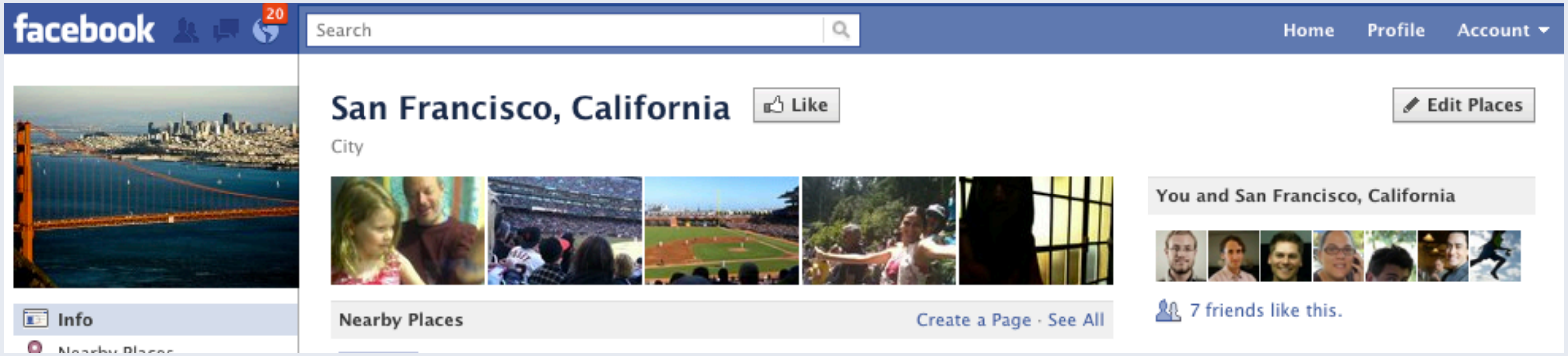
First few Generations: Memcache



Look-Aside Cache
Key-Value Store
Does one thing very well
Does little else
Improved performance by 10X

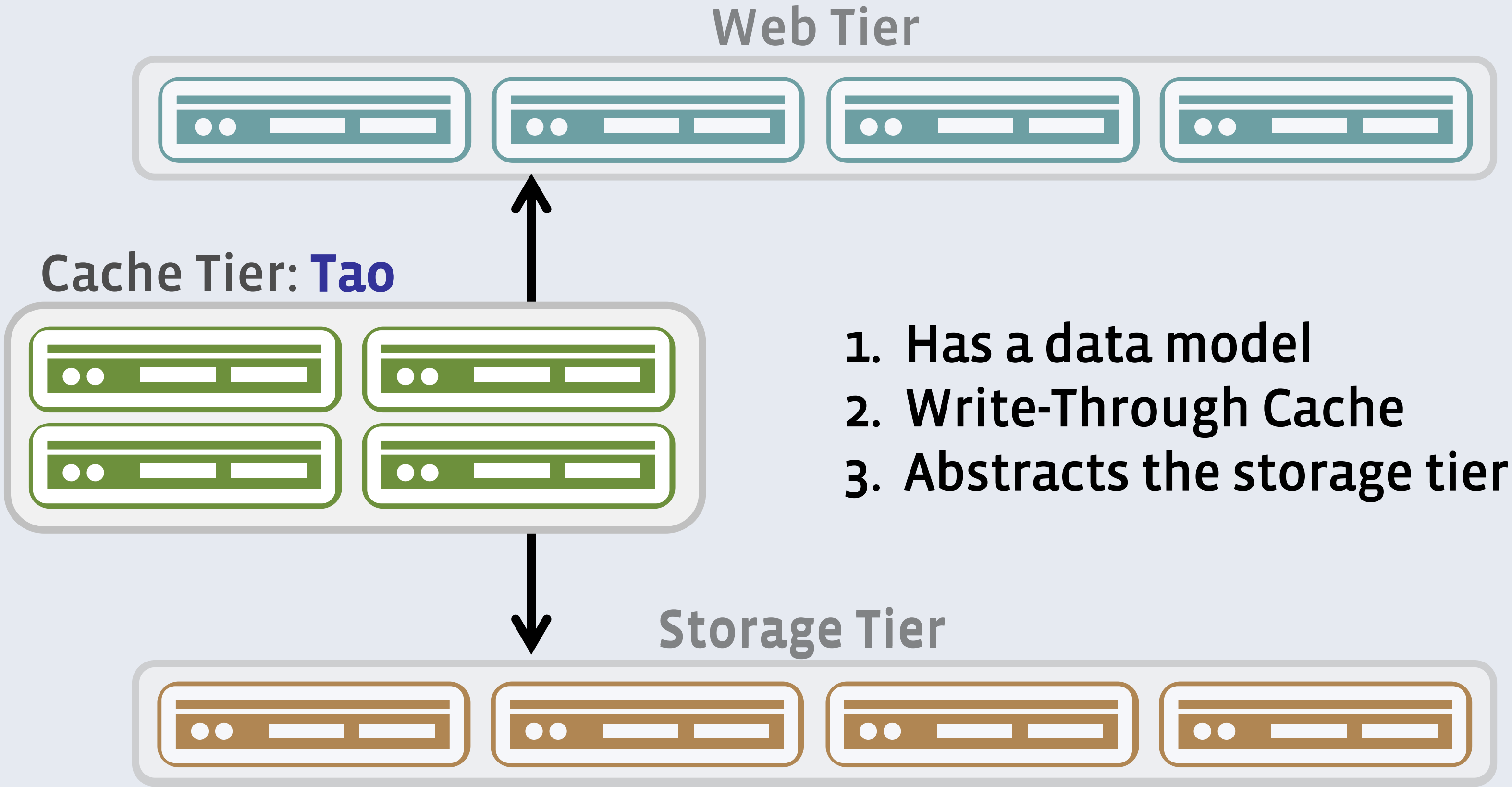
Memcache limitations

- “Values” are opaque
  - End up moving huge amounts of data across the network



- Storage hierarchy exposed to web tier
  - Harder to explore alternative storage solutions
  - Harder to keep consistent
  - Harder to protect the storage tier from thundering herds

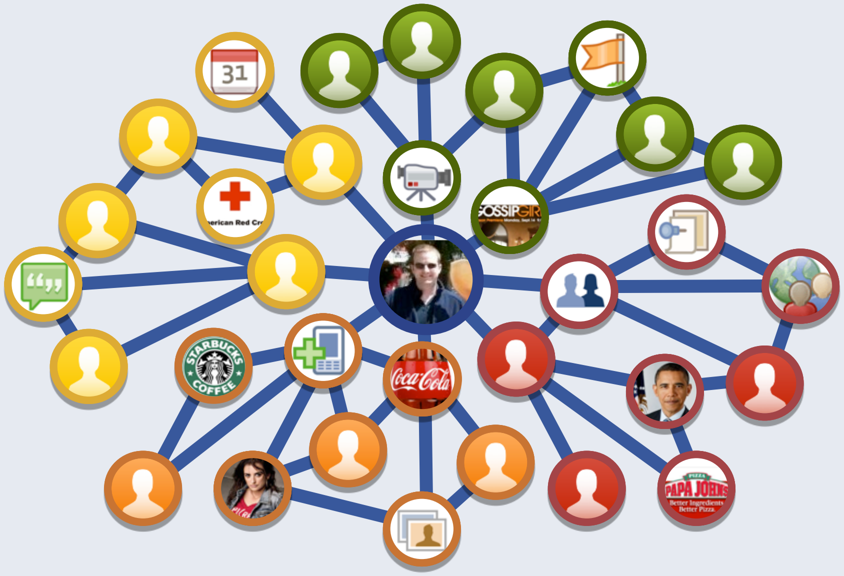
Alternative Caching Tier: Tao



1. Has a data model
2. Write-Through Cache
3. Abstracts the storage tier

Tao Cont'd

- Data Model
  - Objects (Nodes)
  - Associations (edges)
  - Have "type" and data
- Simple graph operations on them
  - Efficient: Content-aware
    - Can be performed on the caching tier
- In production for a couple of years
  - Serving a big portion of data accesses



Tao opens up possibilities

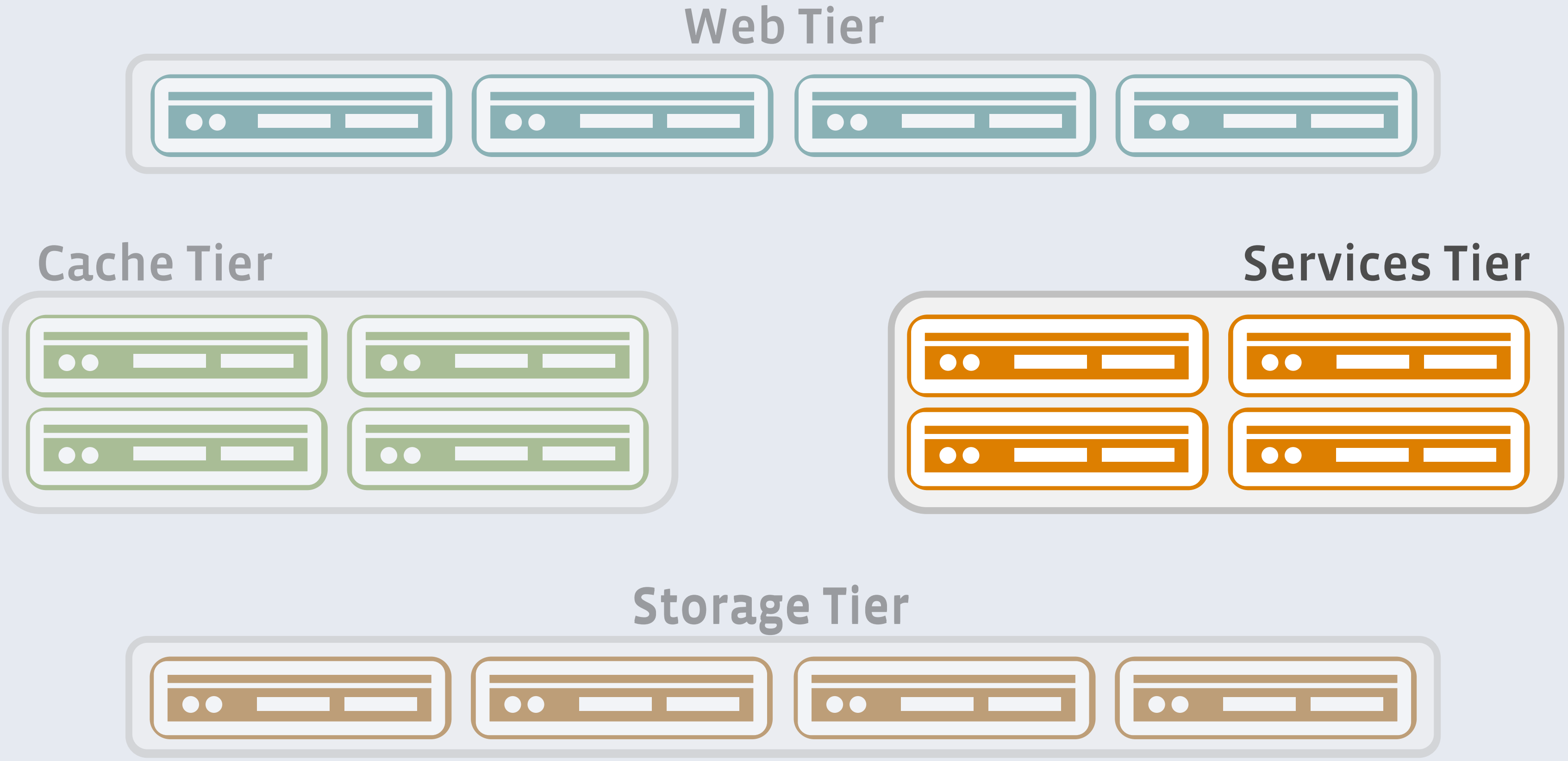
- Alternate storage systems
  - Multiple storage systems
    - To accommodate different use case (access patterns)
- Even more powerful Graph operations
- Multi-Tiered caching

Cache Tier Facts

- Memcache
  - Low CPU utilization
  - Little use for Flash since it is bottlenecked on network
- Tao
  - Much higher CPU load
  - Will continue to increase as it supports more complex operations
  - Could use Flash in a multi-tiered cache hierarchy

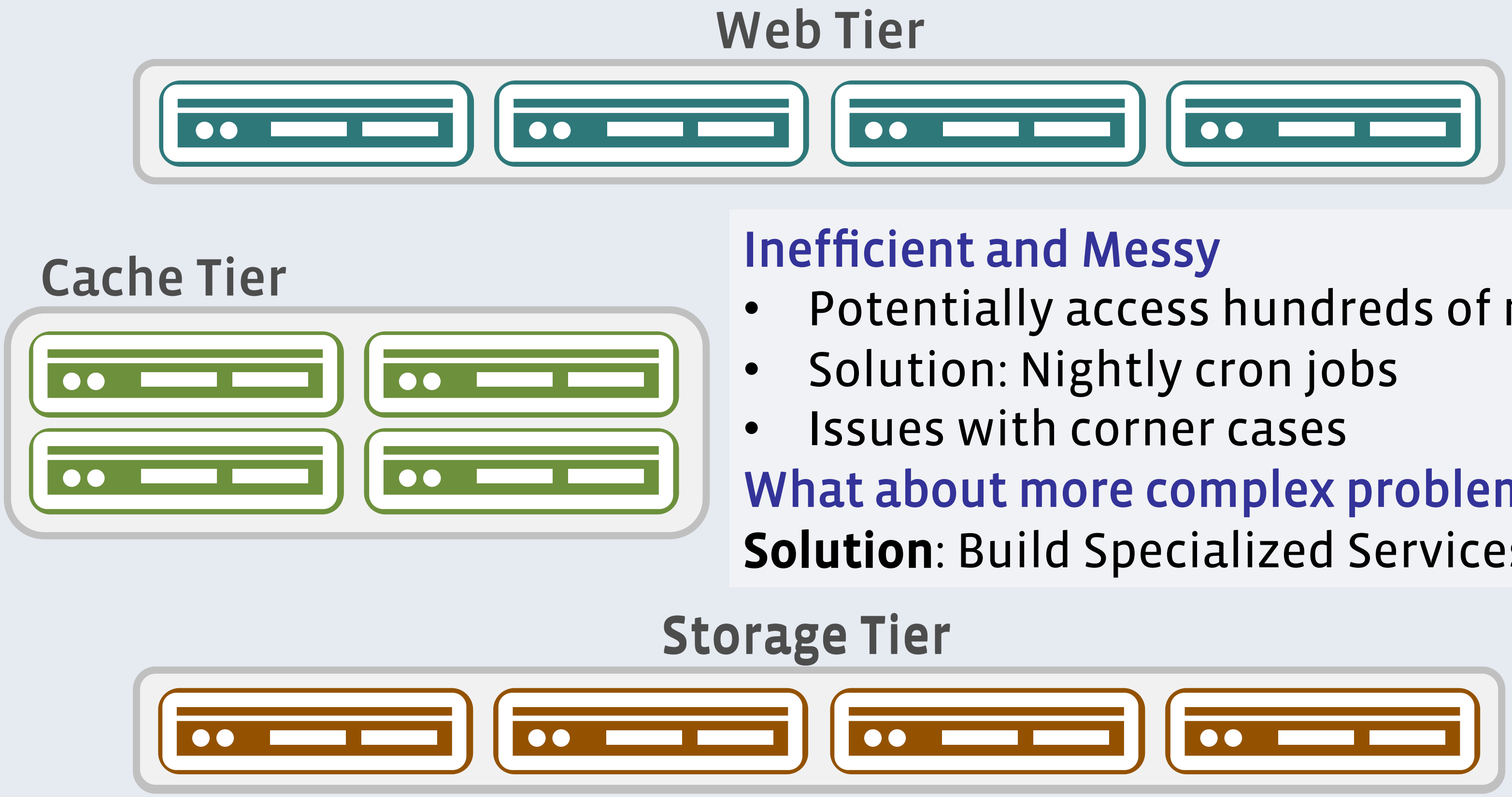
Evolution of the Software Architecture

Evolution of Services Tier



Life before Services

Example: Wish your friend a Happy Birthday



Inefficient and Messy

- Potentially access hundreds of machines
- Solution: Nightly cron jobs
- Issues with corner cases

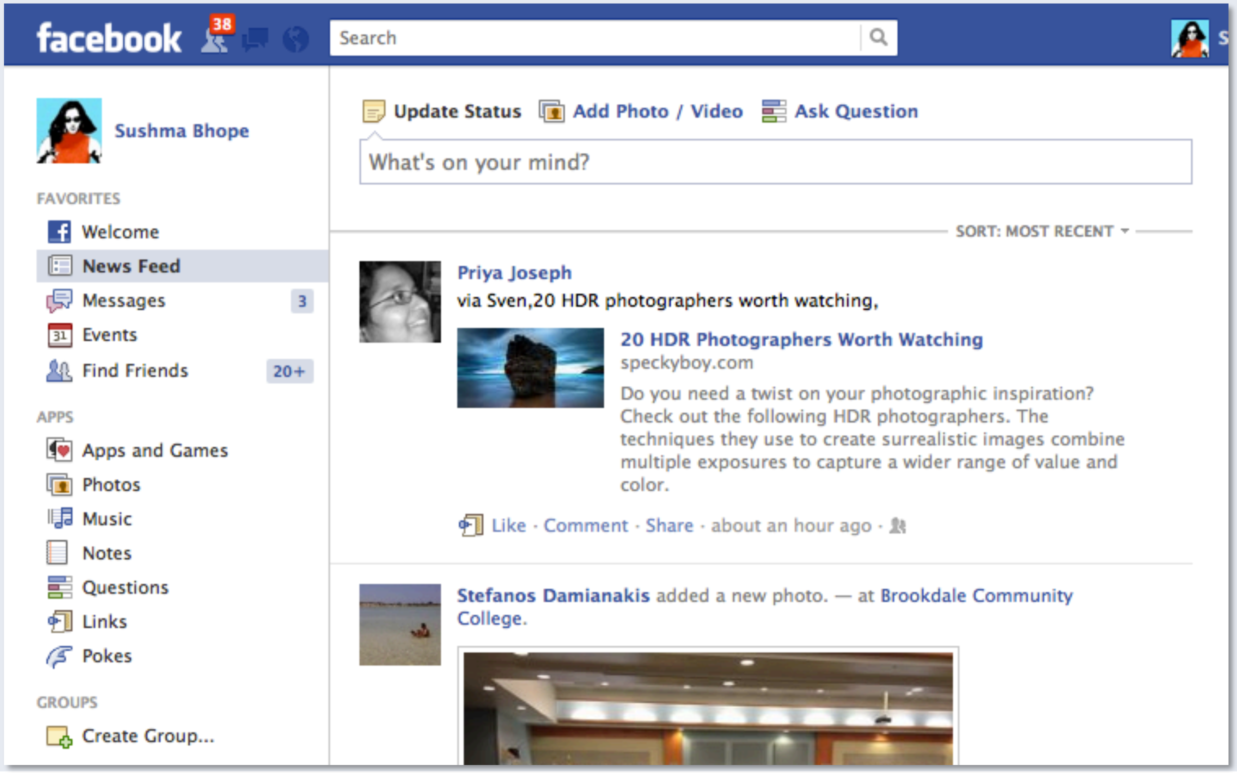
What about more complex problems?

Solution: Build Specialized Services

A more complex service: News Feed

Aggregation of your friends' activity

One of many (100s) services at Facebook



Priya Joseph

via Sven,20 HDR photographers worth watching,

20 HDR Photographers Worth Watching

speckyboy.com

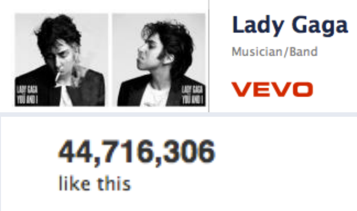
Do you need a twist on your photographic inspiration? Check out the following HDR photographers. The techniques they use to create surrealistic images combine multiple exposures to capture a wider range of value and color.

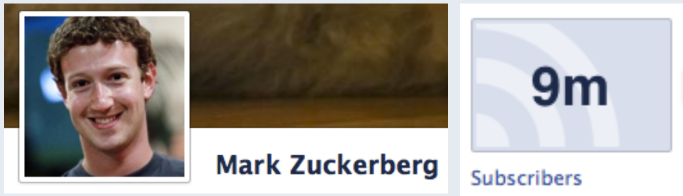
Like · Comment · Share · about an hour ago ·

Stefanos Damianakis added a new photo. — at Brookdale Community College.

News Feed Product characteristics

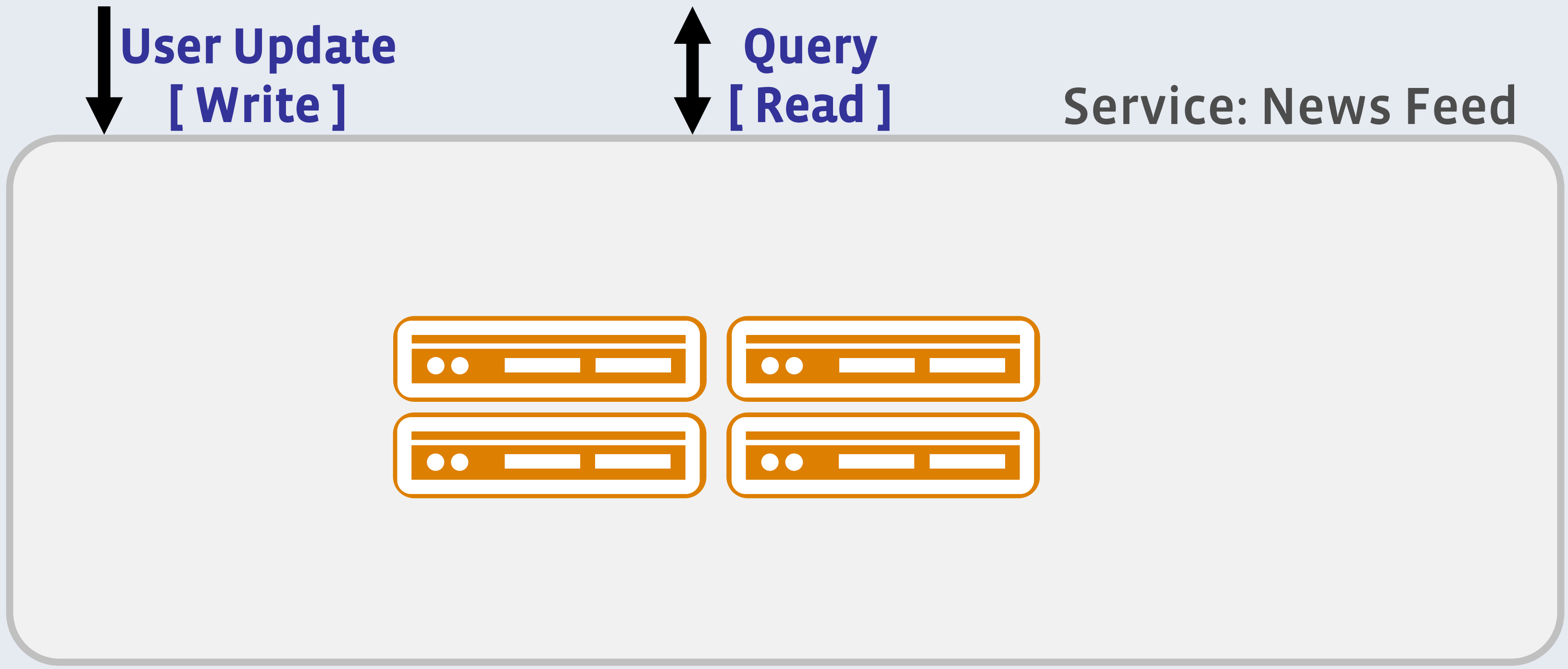
- Real-time distribution
  - Along edges on the Social Graph
- **Writer** can potentially broadcast to very large audience





- **Reader** wants different & dynamic ways to filter data
  - Average user **has 1000s of stories per day** from friends/pages
  - Friend list, Recency, Aggregation, Ranking, *etc.*

News Feed Service

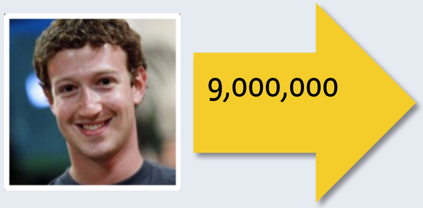


- **Build and maintain an index**: Distributed
- **Rank**: Multiple ranking algorithms

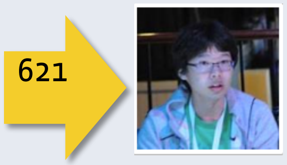
Two approaches: Push vs. Pull

- **Push approach**
  - Distribute actions by **reader**
  - Write broadcasts, read one location
- **Pull approach**
  - Distribute actions by **writer**
  - Write one location, read gathers

- **Pull model is preferred because**
  - More dynamic: Easier to iterate
  - “In a social graph, the number of incoming edges is much smaller than the outgoing ones.”

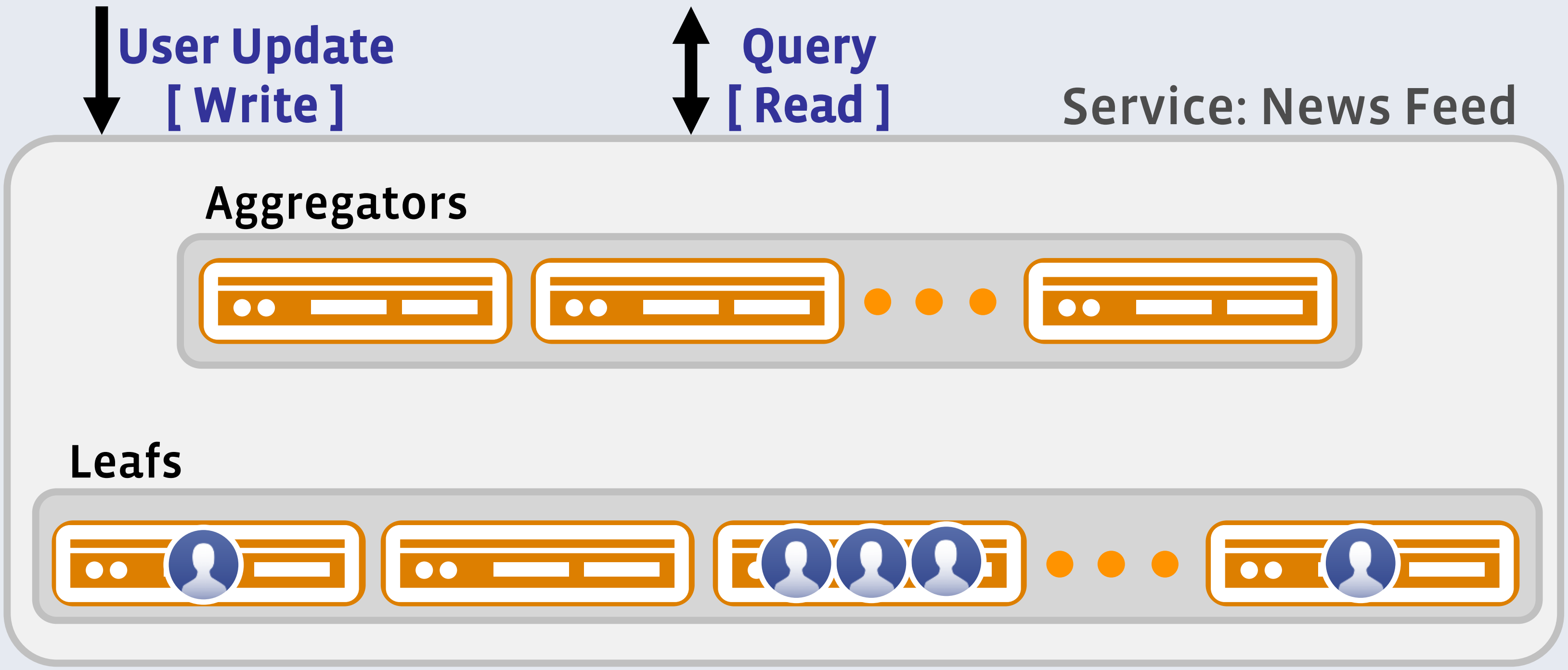


9,000,000



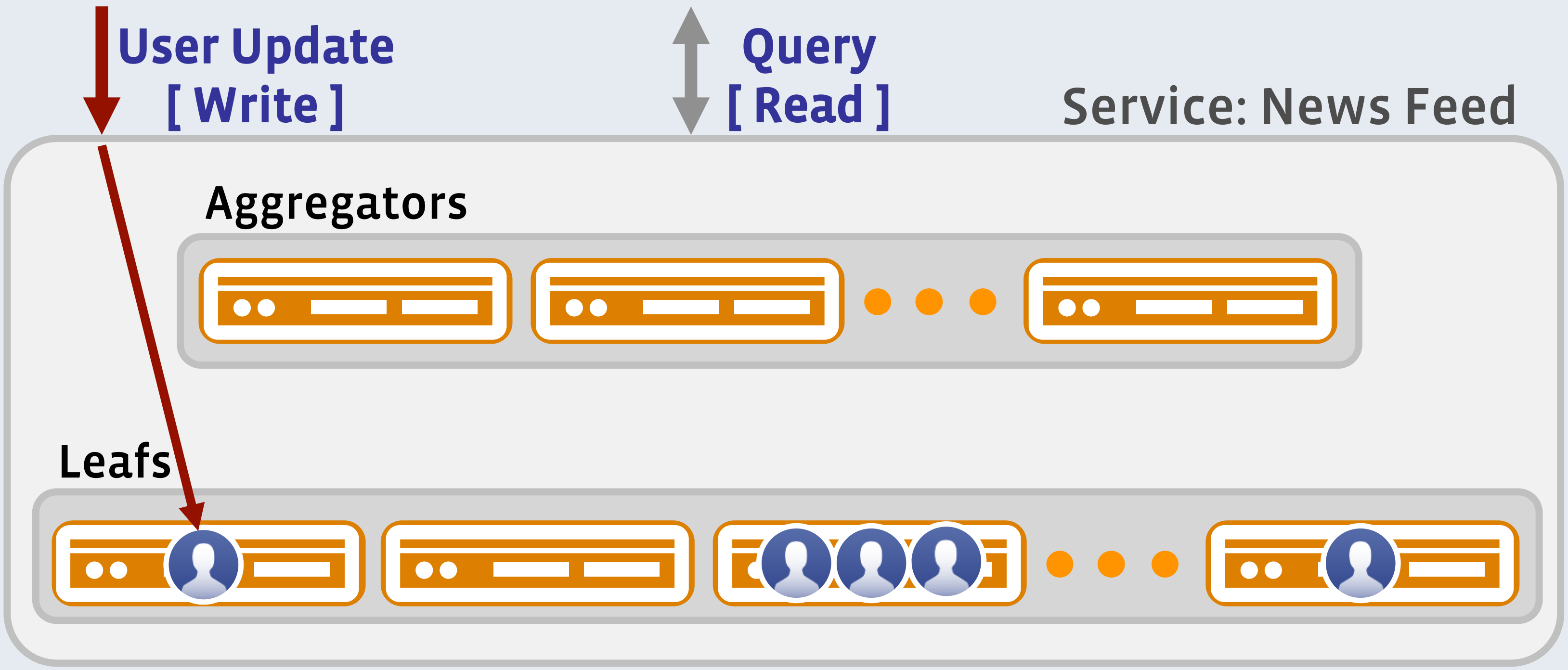
621

News Feed Service: Big Picture



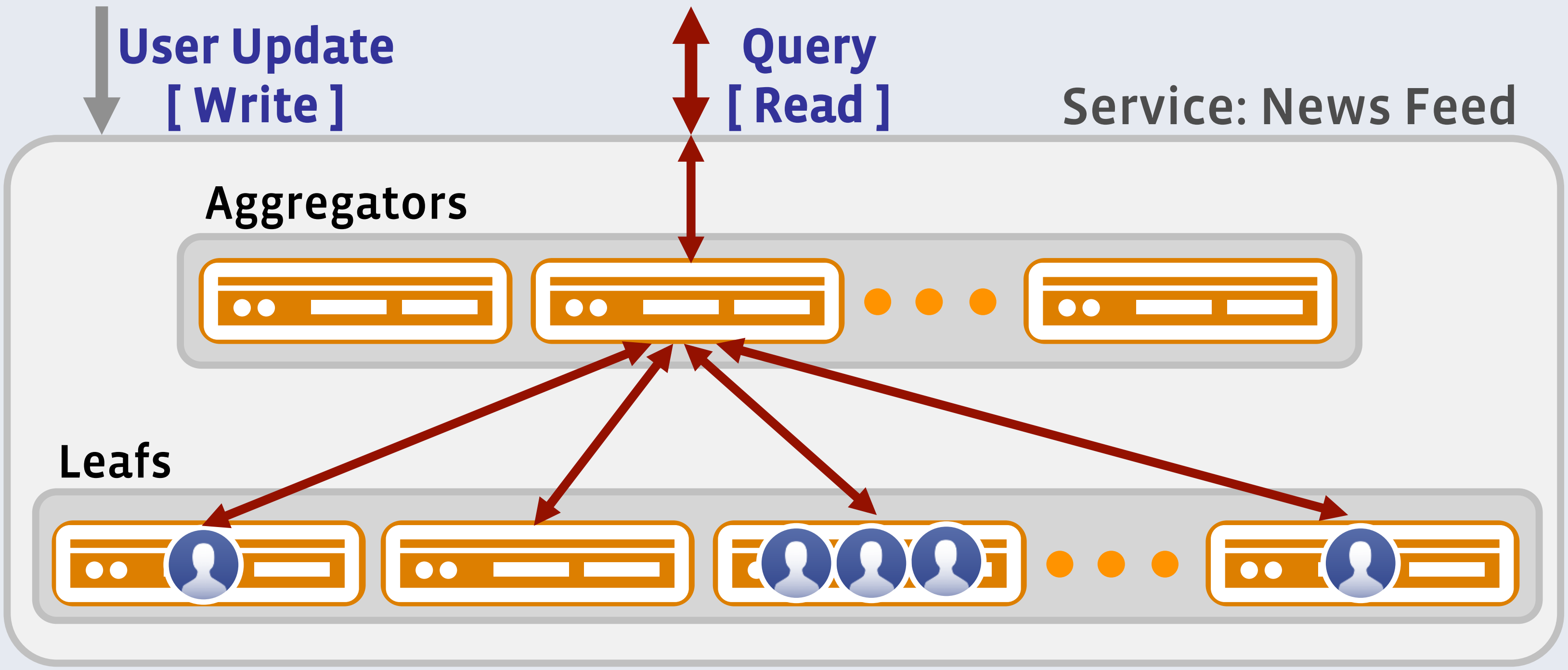
- **Pull Model**
  - **Leafs**: One copy of the entire index. Stored in memory (**Soft state**)
  - **Aggregators**: Aggregate results on the read path (**Stateless**)

News Feed Service: Writes



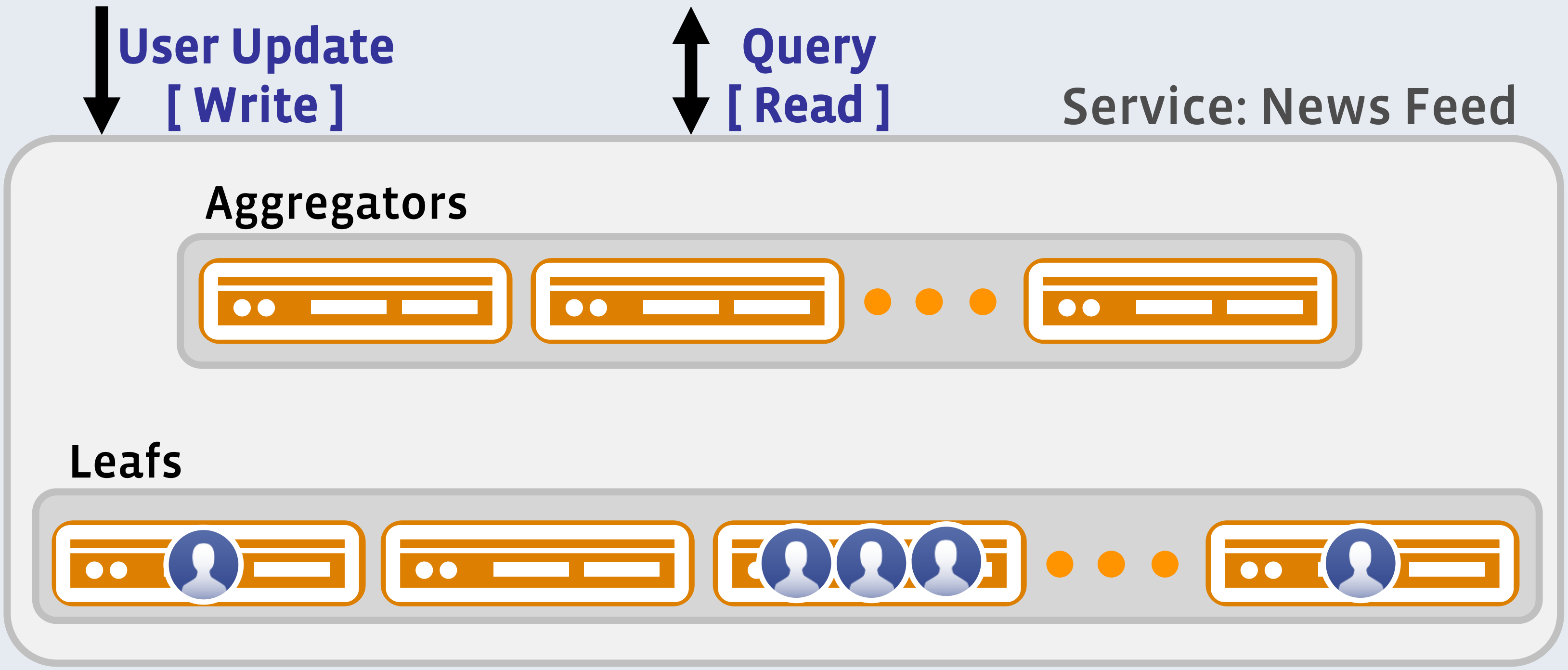
- On User update (Write)
  - Index sharded by Writer
  - Need to update one leaf

News Feed Service: Reads



- On Query (Read)
  - Query all leafs
  - Then do aggregation/ranking

News Feed Service: Scalability



- 1000s of machines
  - **Leafs**: Multiple sets. Each set (10s of machines) has the entire index
  - **Aggregators**: Stateless. Scale with load.

News Feed Service: Reliability

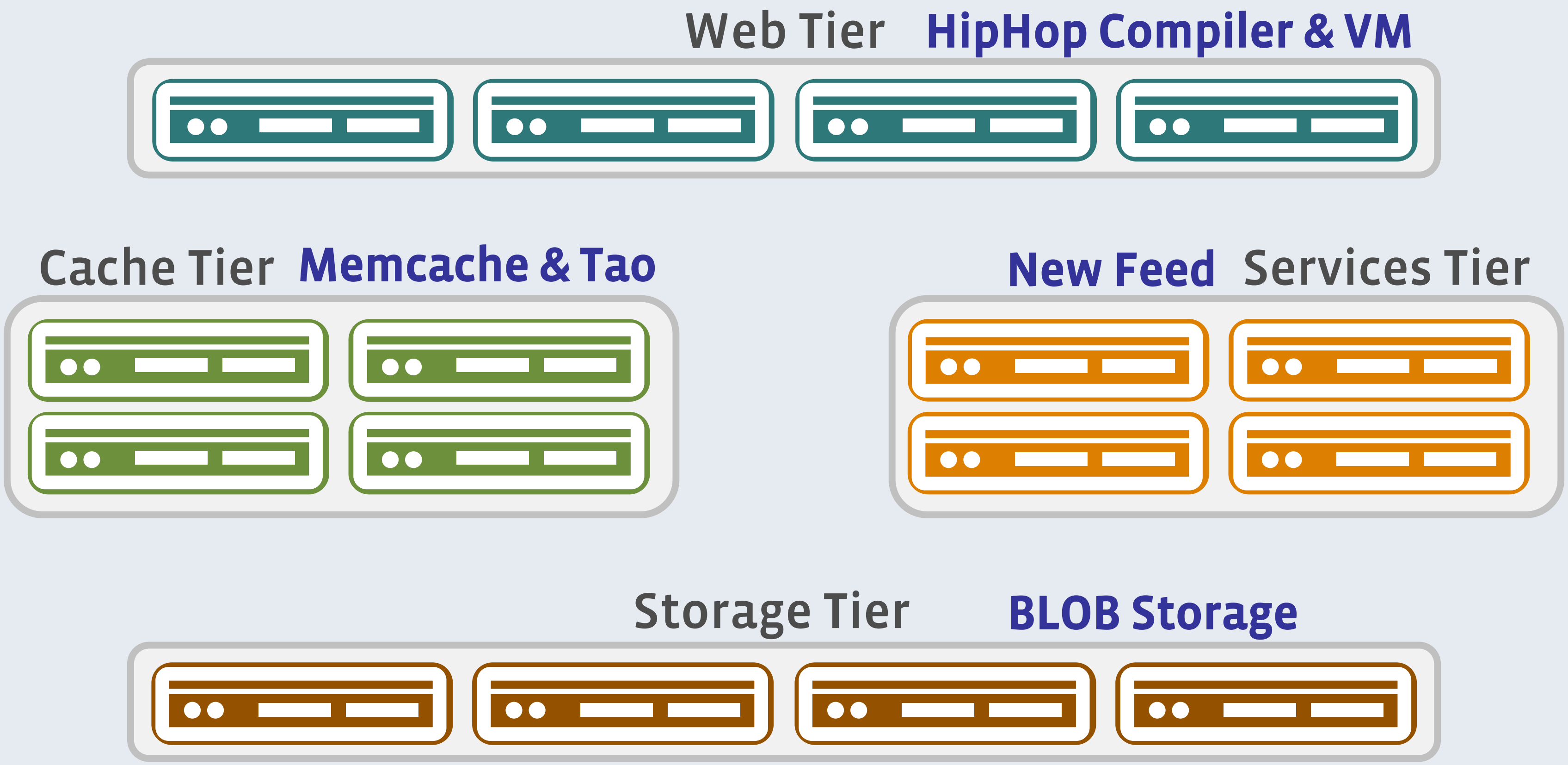
- Dealing with (daily) failures
  - Large number of failure types
    - Hardware/software
    - Servers/Networks
    - Intermittent/Permanent
    - Local/Global
- Keep the software architecture simple
  - Stateless components are a plus
- For example, on read requests:
  - If a **leaf** is inaccessible, failover the request to a different set
  - If an **aggregator** is inaccessible, just pick another

New Feed Service Facts

- Number of leafs dominate the number of aggregators
  - Reads are more expensive than writes
  - Every read (query) involves **one** aggregator and **every** leaf in the set
- Very high network load between aggregator and leafs
  - Important to keep a full leaf set within a single rack on machines
  - Uses Flash on leafs to ensure this

Evolution of the Software Architecture

Summary



Web Tier HipHop Compiler & VM

Cache Tier Memcache & Tao

New Feed Services Tier

Storage Tier BLOB Storage

Outline

1 What makes scaling Facebook challenging?

2 Evolution of Software Architecture

3 Evolution of Datacenter Architecture

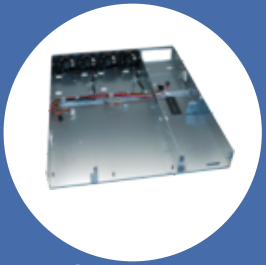
Recall: Characteristics of Facebook

- Massive Scale
- Social Graph
  - Expensive to query
  - Hard to partition
  - Large working set (Fat tail)
- Product is rapidly evolving
- Hardware failures are routine

Implications

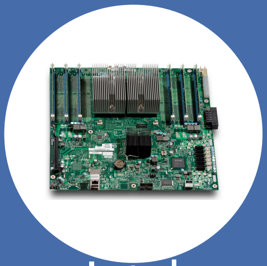
- **On Datacenters**
  - Small number of massive datacenters (currently 4)
- **On Servers**
  - Minimize the “classes” (**single digit**) of machines deployed
    - Web Tier, Cache Tier, Storage Tier, and a couple of special configurations
- **Started with**
  - Leased datacenters + Standard server configurations from vendors
- **Moving to**
  - Custom built datacenters + custom servers
  - Continue to rely on a small number of machine “classes”

Servers



Server Chassis

AMD Motherboard



Intel Motherboard

Power Supply

Battery Cabinet

Triplet Rack

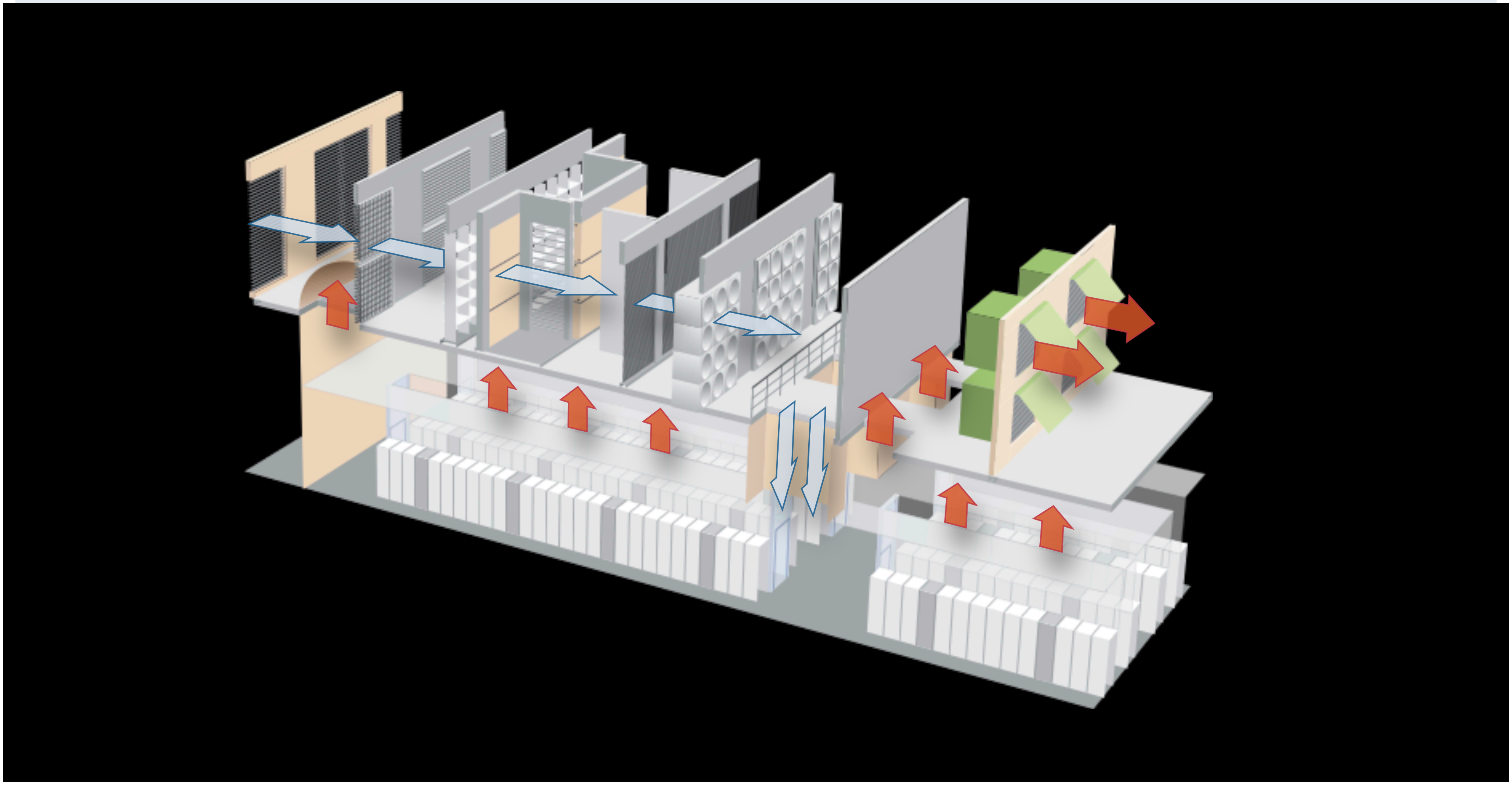
Data Center

Electrical

Mechanical

WV
EU

Evaporative cooling system



Open Compute

- Custom datacenters & servers
- Minimizes power loss
  - **POE of 1.07**
- Vanity Free design
  - **Designed for ease of operations**
- Designs are open-sourced
  - More on the way

OPEN
Compute Project

Outline

1 What makes scaling Facebook challenging?

2 Evolution of Software Architecture

3 Evolution of Datacenter Architecture

Questions?

facebook