CS-310 Scalable Software Architectures

Lecture 11:
Basic Architecture Design

Steve Tarzia

Last Time: Authentication

- Webservice requests are rarely open to the public.
- Each request must include an input that **authenticates** and identifies the user.
- Passwords are the most common auth mechanism.
- Email/SMS (a trusted *side channel* of communication) can be used.
- **Authentication tokens** are strings randomly generated (and stored) on the backend to verify user identity.
  - Variations include **session keys**, **cookies**, and **api keys.**
  - Often a separate microservice is dedicated to authentication (and other user management tasks, like account creation).

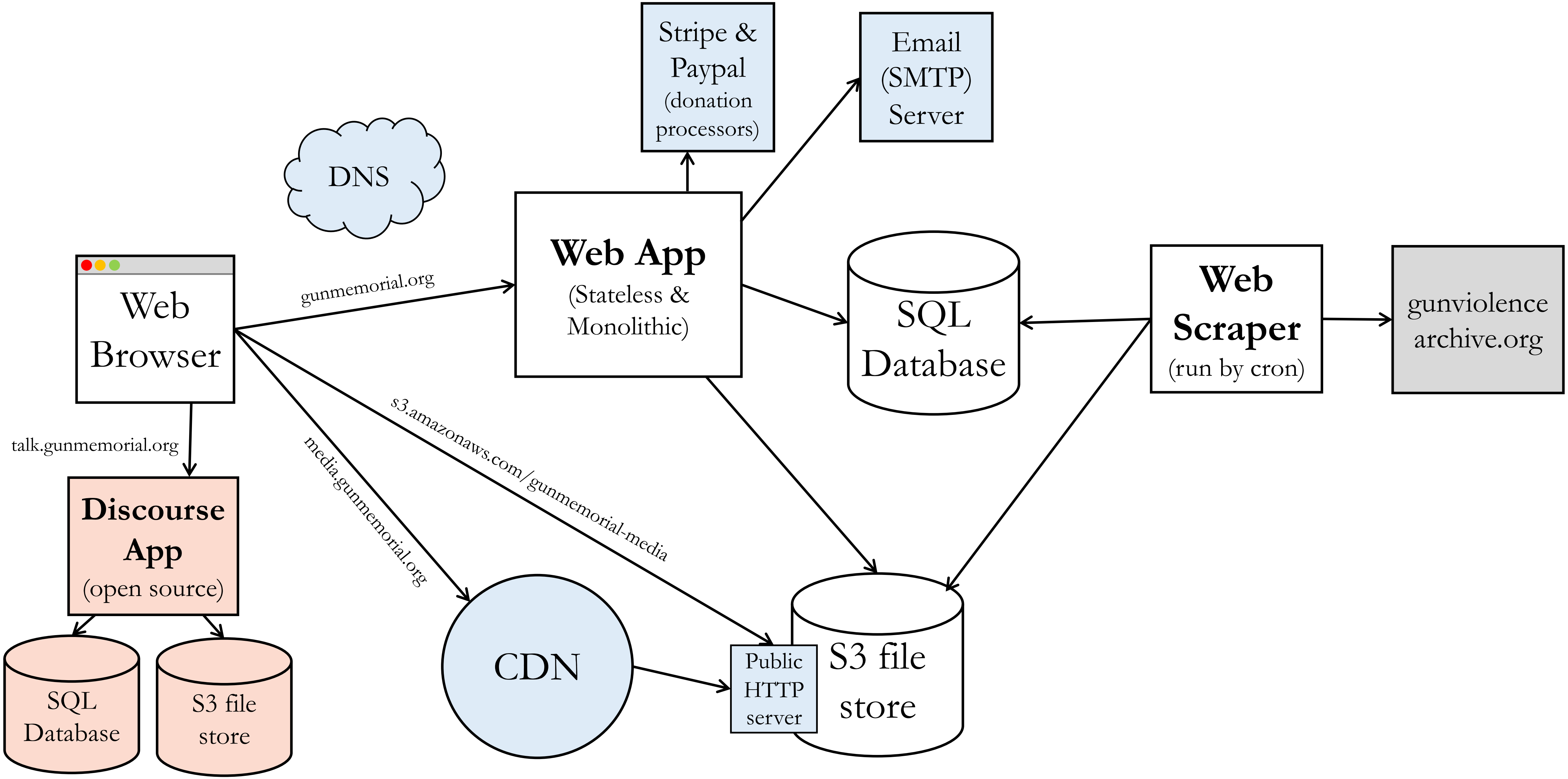
Case Study: National Gun Violence Memorial

- https://gunmemorial.org
- Java servlet w/JSP, connecting to a SQL database, with S3 for images.

AWS deployment uses these services:

- Elastic Beanstalk
- EC2: Elastic Compute Cloud (Virtual Machines)
- RDS: Relational Database Service
- CloudFront (CDN)
- Route 53 (DNS)
- Simple Email Service (SES)

NGVM architecture diagram



Monolithic web app API: **Public Pages**

HTML pages:

- GET /
- GET /[year]/[mon]/[day]/[name]
- GET /[year]/[mon]/[day]
- GET /about
- GET /search

… etc.

For full list of public pages, see:
http://gunmemorial.org/sitemap.txt
http://gunmemorial.org/sitemap.txt?startYear=2020&endYear=2020

HTML Form and JS endpoints:

- POST /doLightCandle?victim=[id]
- POST /doPublicPostPhoto
  - Body: multipart/form-data:
    - victim (int)
    - source (string)
    - contact (string)
    - mine (boolean)
    - grant (boolean)
    - sure (boolean)
    - file (binary image data)
- POST /poll/doAnswerQuestion?...
- POST /poll/doModerateQuestion?...
- POST /doDonate?stripeToken=[…]&amount=[cents]

Note that this API's design does not follow REST style. Paths specify actions, not resources.

Monolithic web app API: **Volunteers' Portal**

HTML pages

- GET /sign-in *(no cookie required, response sets a cookie)*
- GET /admin
- GET /admin/victim_edit.jsp?id=[id]
- GET /admin/photo_edit.jsp?photo=[id]
- GET /admin/moderate_photos.jsp
- GET /admin/moderate_answers.jsp
- GET /admin/victim_add.jsp

… etc.

In all these requests, require a cookie to authenticate and identify the user.

HTML Form and JS endpoints:

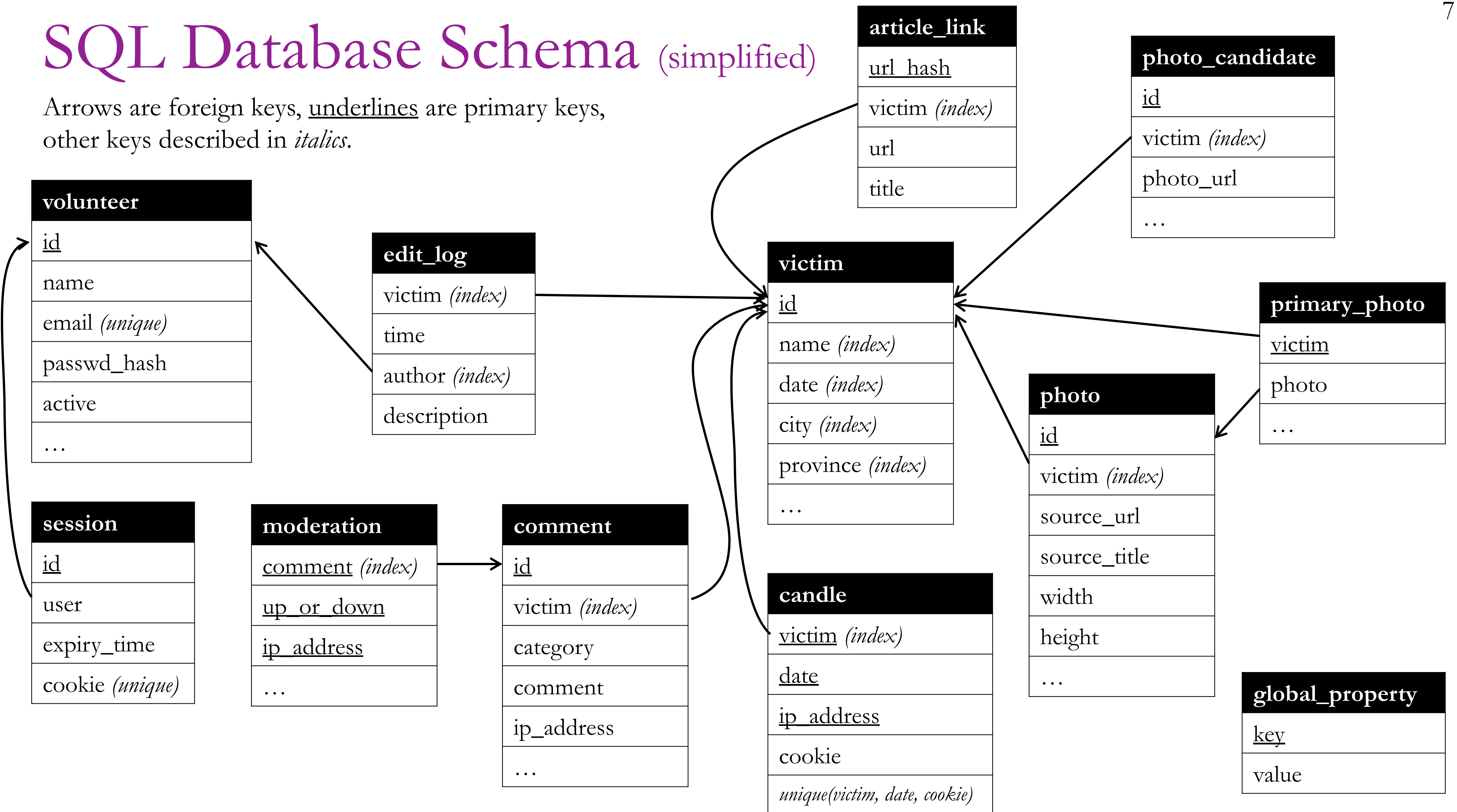
- POST /admin/doAddVictim?...
  - Query params:
    - date (YYYY-MM-DD)
    - city (string)
    - province (two-letter abbreviation)
    - name (string)
    - gender (string)
- POST /admin/doChangePassword?
- POST /admin/doChoosePhoto?
- POST /admin/doEditPhoto?
- POST /admin/doDeleteVictim?

How to rewrite this following REST design principles?

Answer: DELETE /victim/{id}

SQL Database Schema (simplified)

Arrows are foreign keys, underlines are primary keys, other keys described in *italics*.



article_link
url_hash
victim *(index)*
url
title

photo_candidate
id
victim *(index)*
photo_url
…

volunteer
id
name
email *(unique)*
passwd_hash
active
…

edit_log
victim *(index)*
time
author *(index)*
description

victim
id
name *(index)*
date *(index)*
city *(index)*
province *(index)*
…

primary_photo
victim
photo
…

photo
id
victim *(index)*
source_url
source_title
width
height
…

session
id
user
expiry_time
cookie *(unique)*

moderation
comment *(index)*
up_or_down
ip_address
…

comment
id
victim *(index)*
category
comment
ip_address
…

candle
victim *(index)*
date
ip_address
cookie
unique(victim, date, cookie)

global_property
key
value

S3 File Store details

- candidate_photo/[uuid].jpg
- photo/[photo_id].jpg
- photo_thumb/100/[photo_id].jpg
- photo_thumb/400/[photo_id].jpg
- photo_thumb/800w/[photo_id].jpg
- web_archive/[article_url_md5hash].html

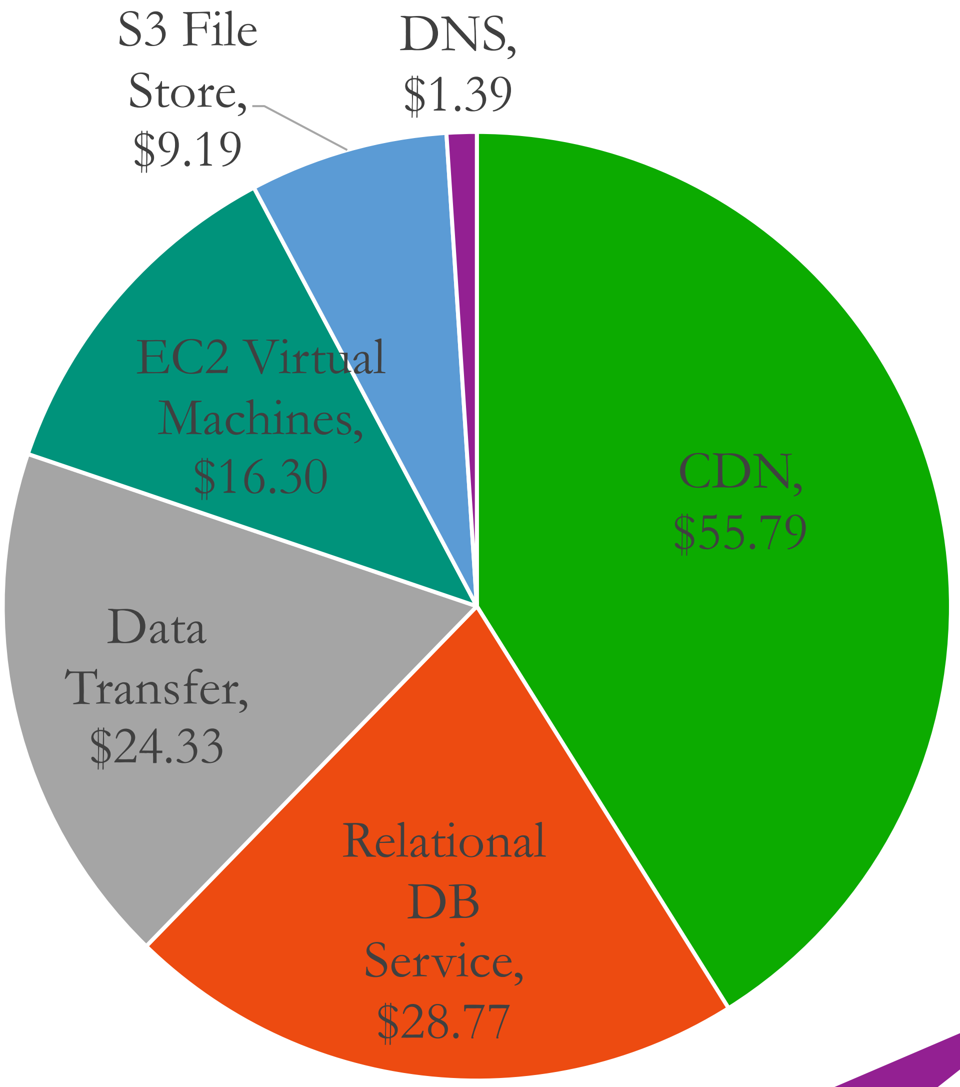
- ➢ Use a randomized uuid to prevent public scan.
- ➢ 100px-tall thumbnail
- ➢ 400px-tall thumbnail
- ➢ 800px-wide thumbnail
- ➢ Copy of news article HTML (in case original article is taken down).

Files have read-only public access at:

- https://s3.amazonaws.com/gunmemorial-media/...
- https://media.gunmemorial.org/...

- ➢ Served from Virginia.
- ➢ Using CDN (costs more).

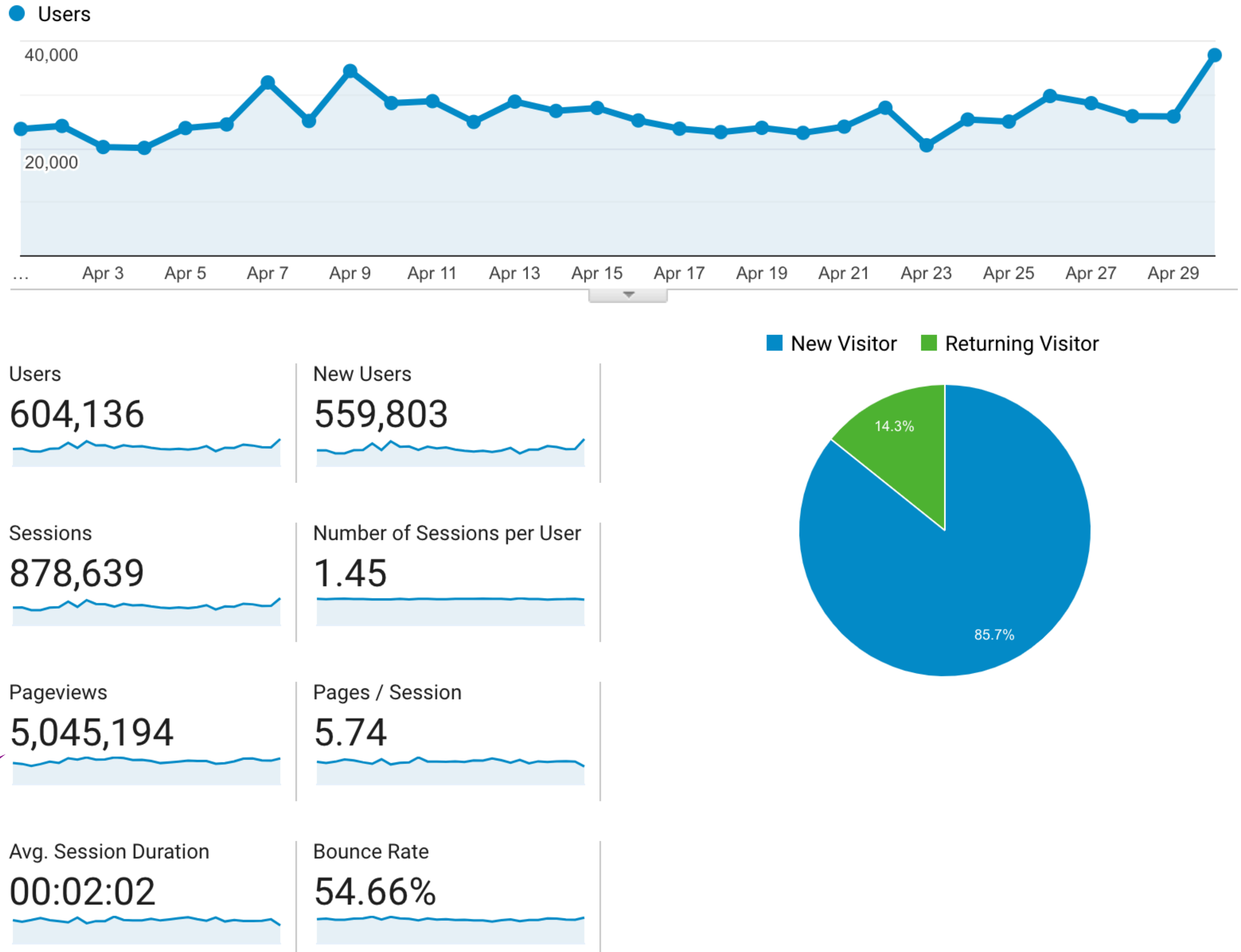
April 2020 monthly operating cost ($136 total)



37k pageviews per $ cost

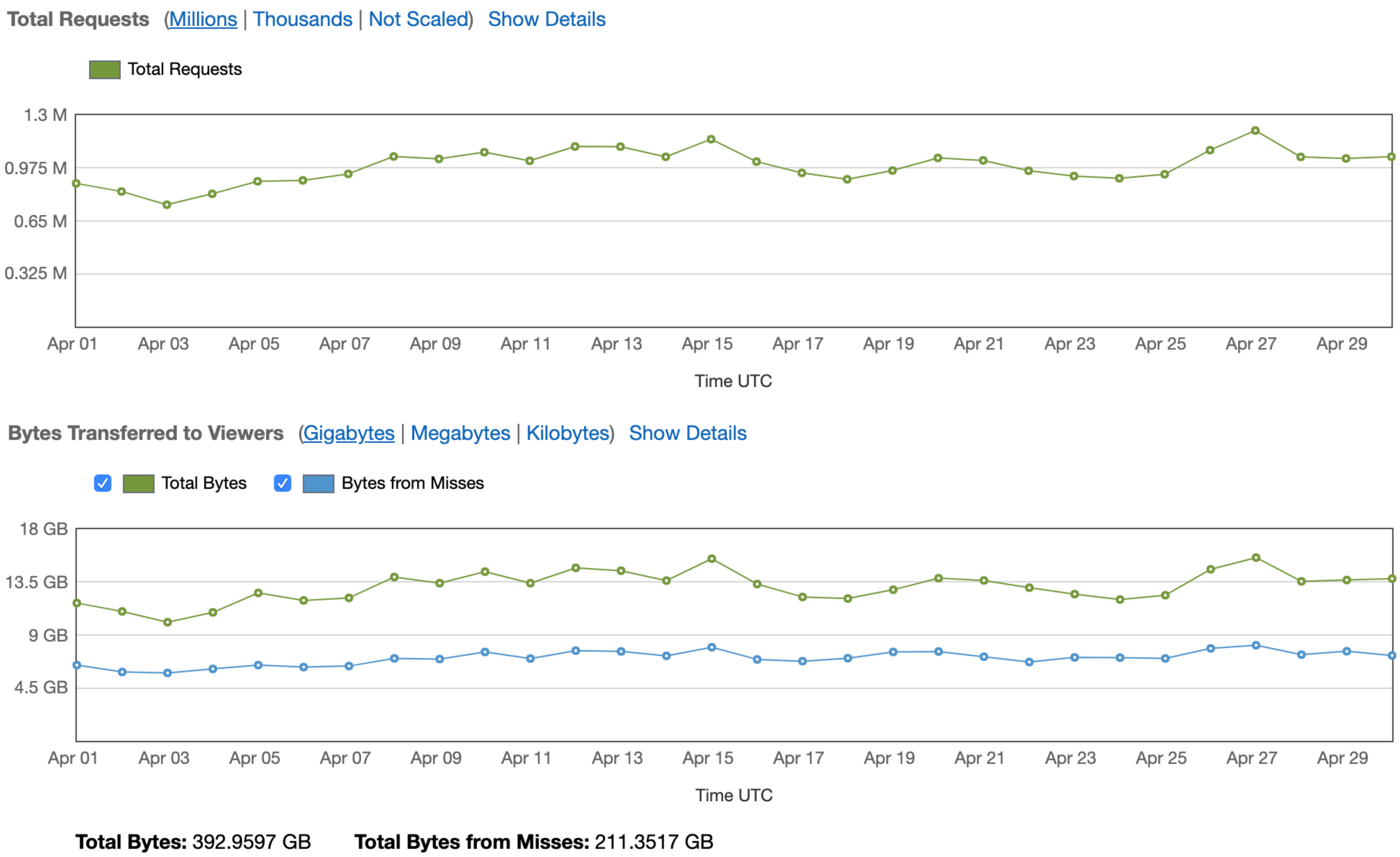
Traffic: (from Google Analytics).

Typically about 150 users on the site at any given time.



Users 604,136	New Users 559,803
Sessions 878,639	Number of Sessions per User 1.45
Pageviews 5,045,194	Pages / Session 5.74
Avg. Session Duration 00:02:02	Bounce Rate 54.66%

CDN statistics in April



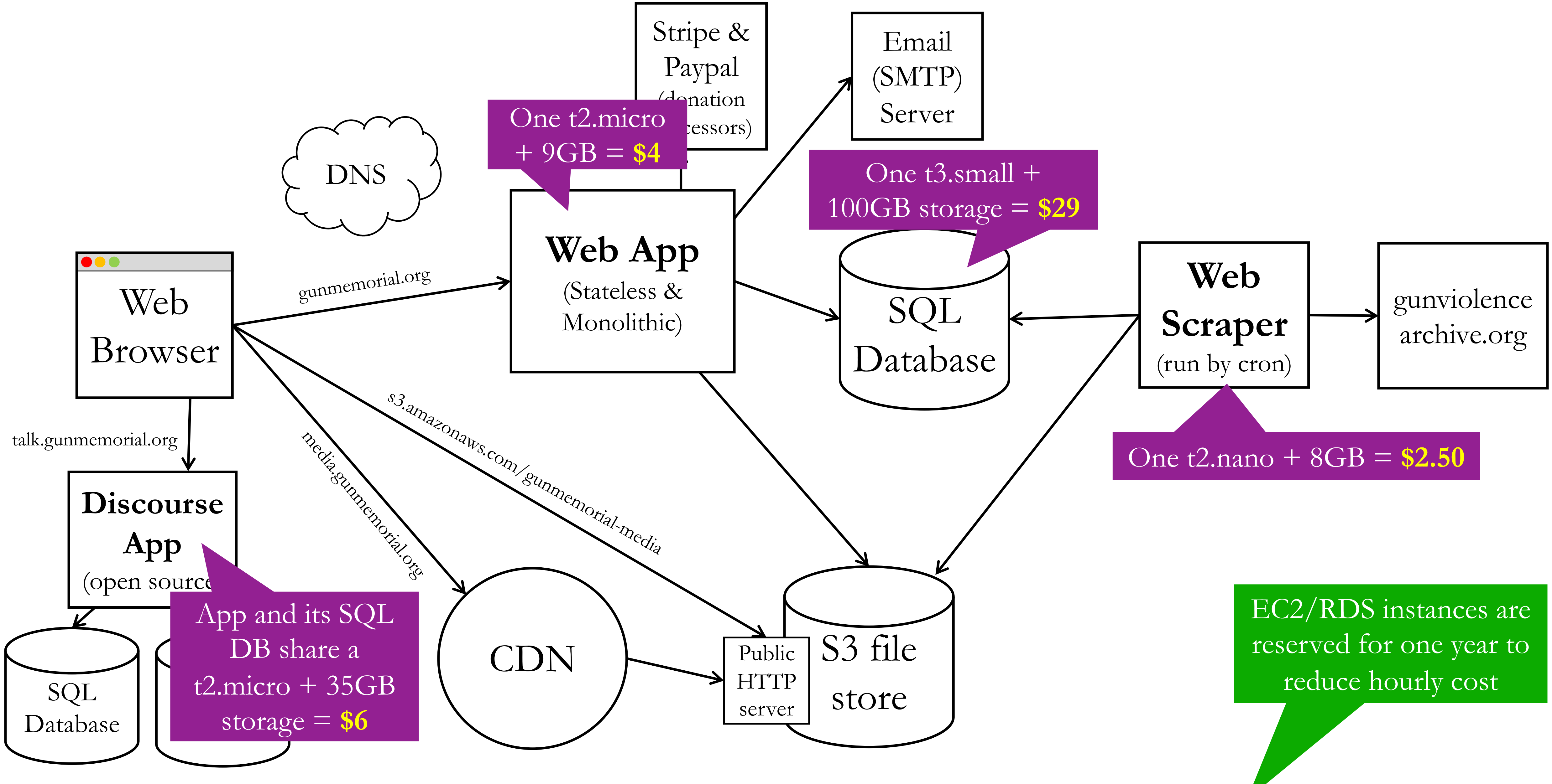
Total Requests (Millions | Thousands | Not Scaled) Show Details

Bytes Transferred to Viewers (Gigabytes | Megabytes | Kilobytes) Show Details

☑ 🟩 Total Bytes ☑ 🟦 Bytes from Misses

Total Bytes: 392.9597 GB **Total Bytes from Misses:** 211.3517 GB

Deployment sizing and monthly costs



One t2.micro + 9GB = **$4**

One t3.small + 100GB storage = **$29**

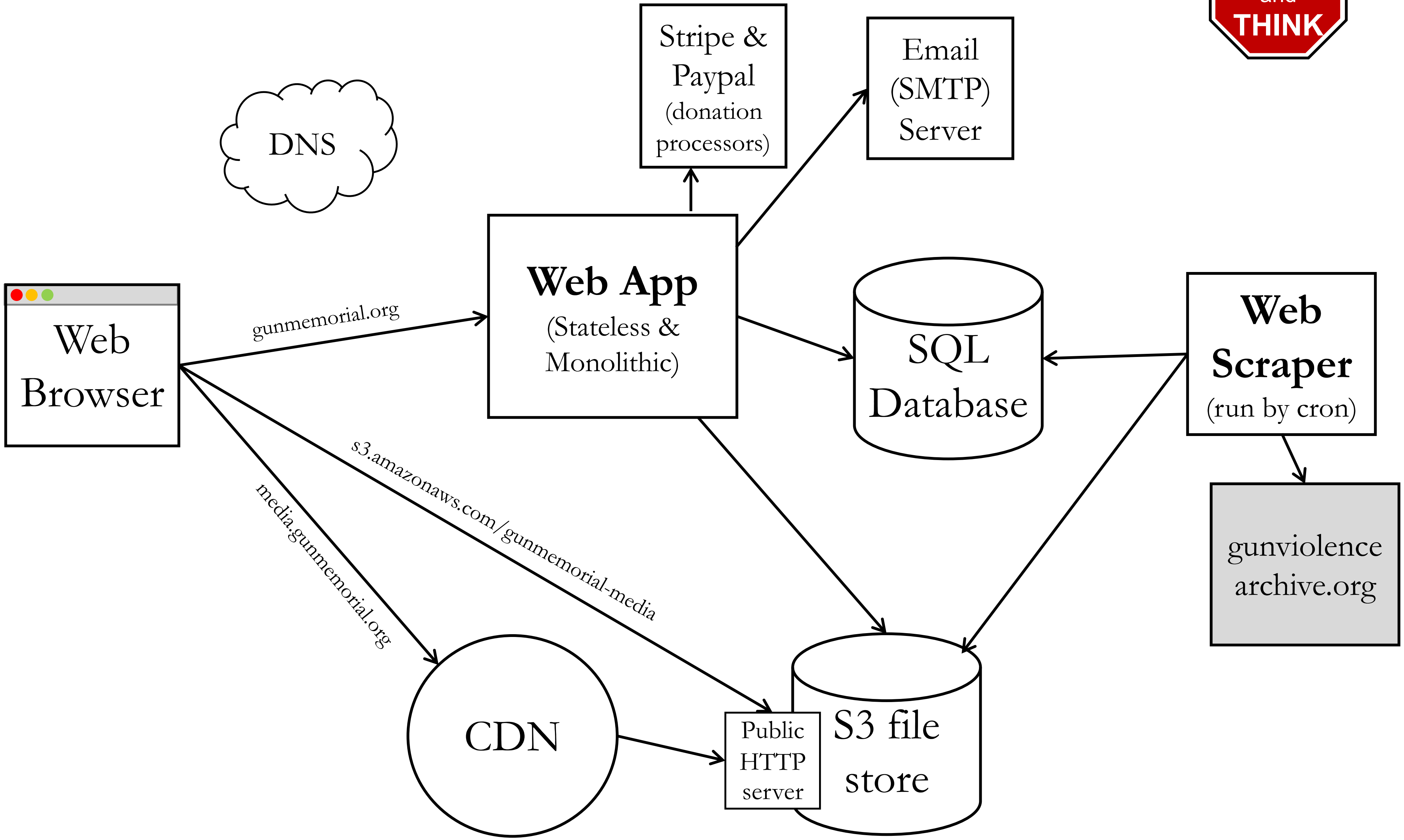
One t2.nano + 8GB = **$2.50**

App and its SQL DB share a t2.micro + 35GB storage = **$6**

EC2/RDS instances are reserved for one year to reduce hourly cost

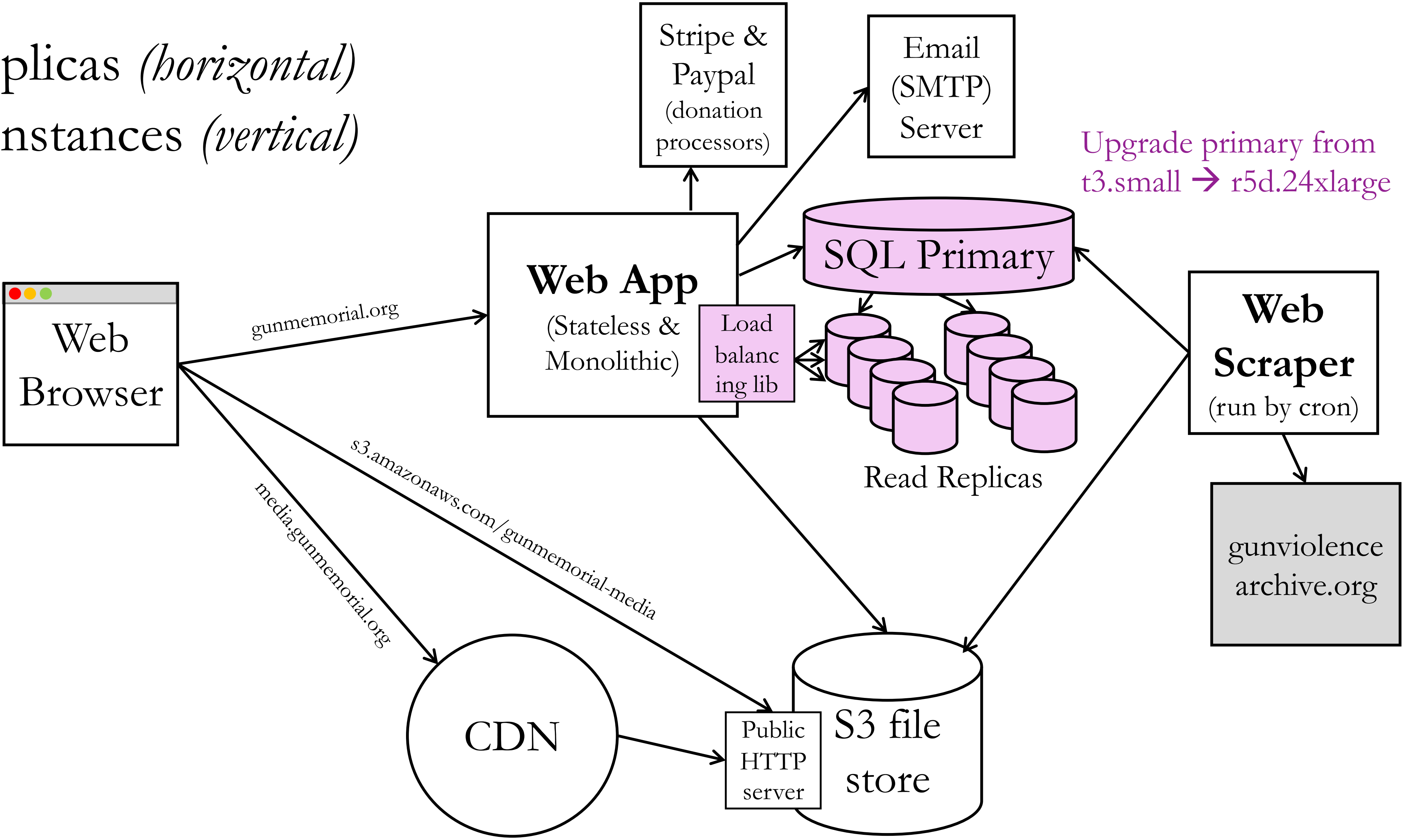
Scaling up to 200x traffic (equal to cnn.com)

STOP and THINK



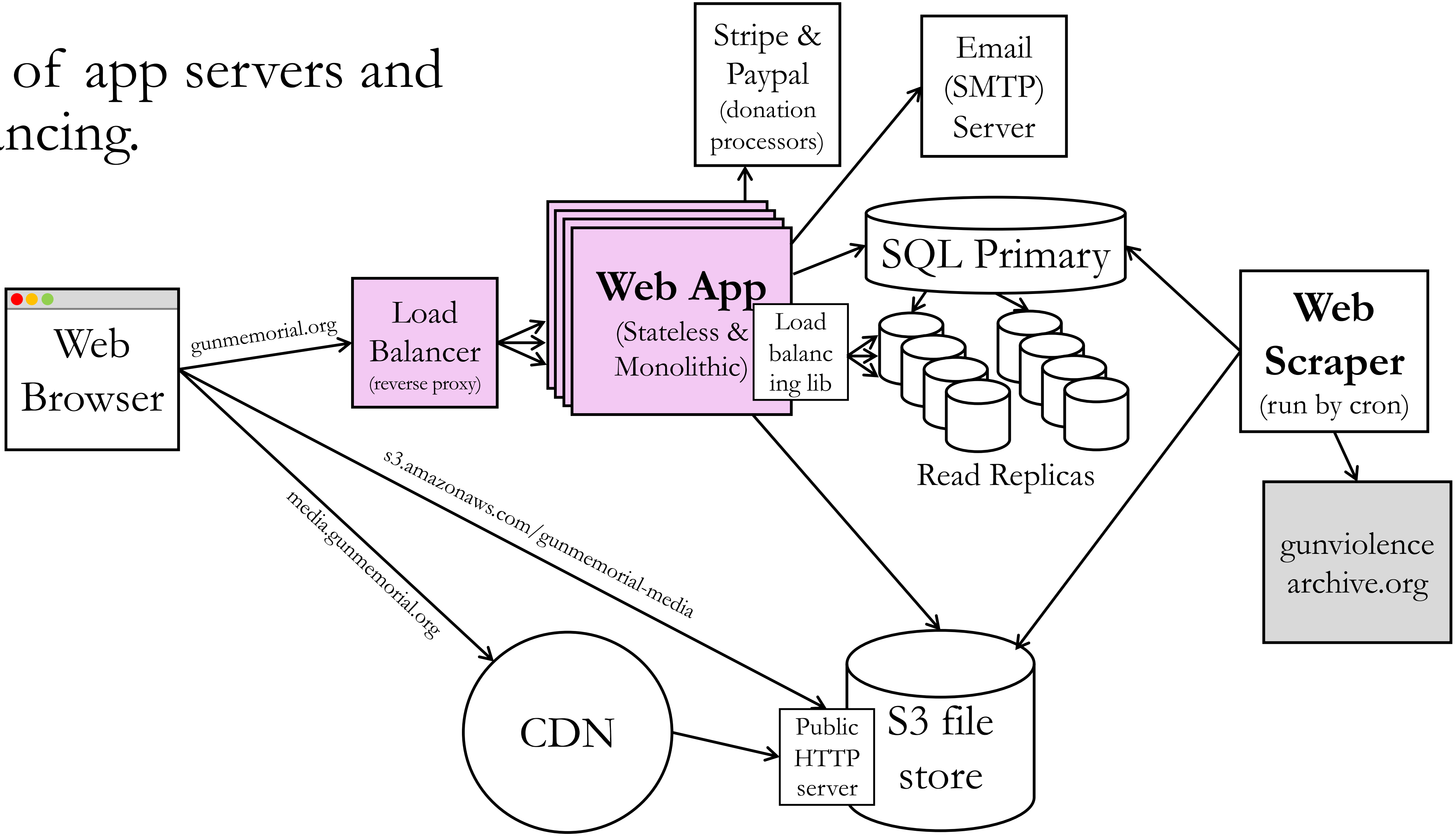
Database scaling

- Add read-replicas *(horizontal)*
- Use bigger instances *(vertical)*



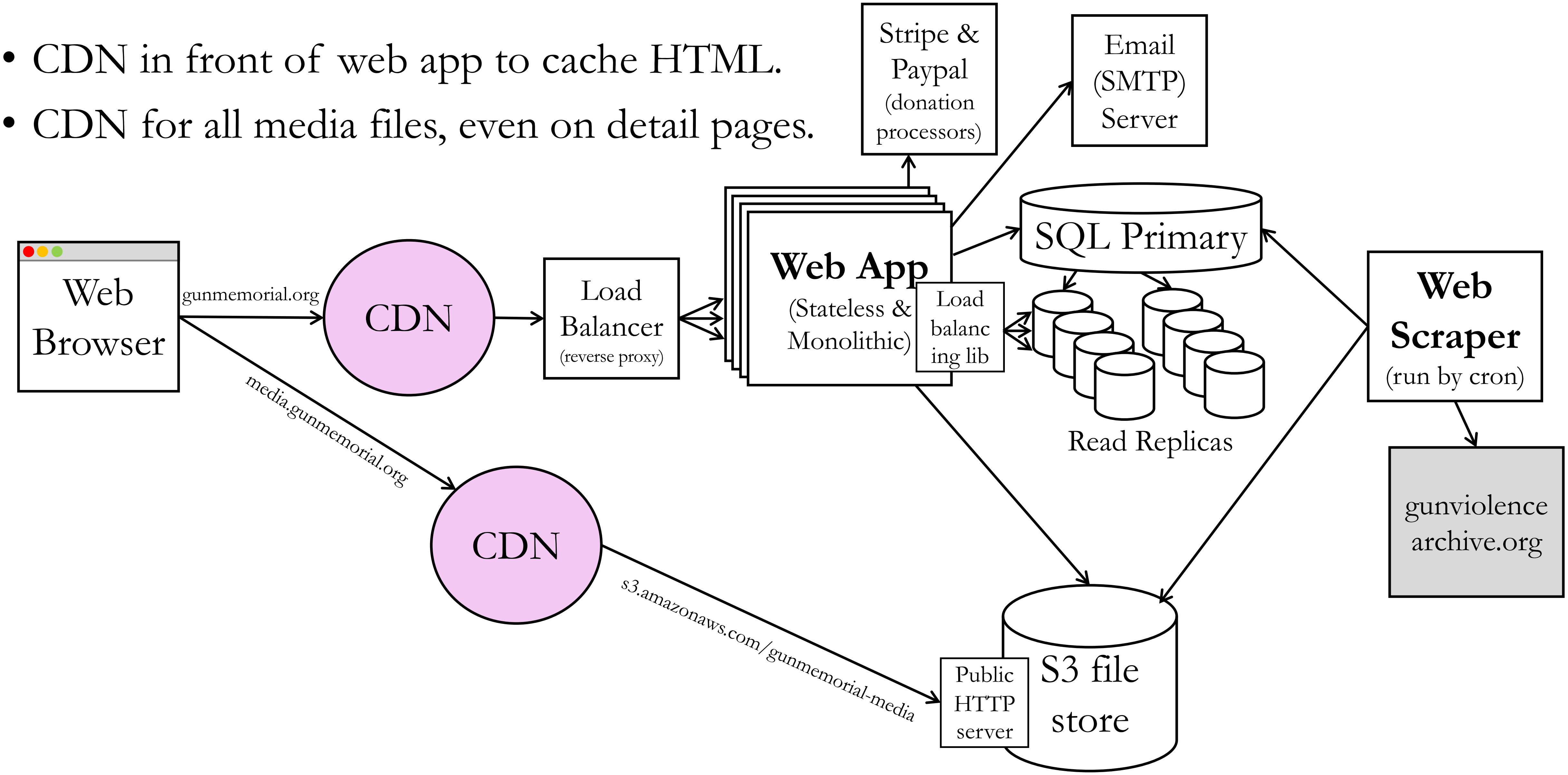
App scaling

- Add lots of app servers and load balancing.

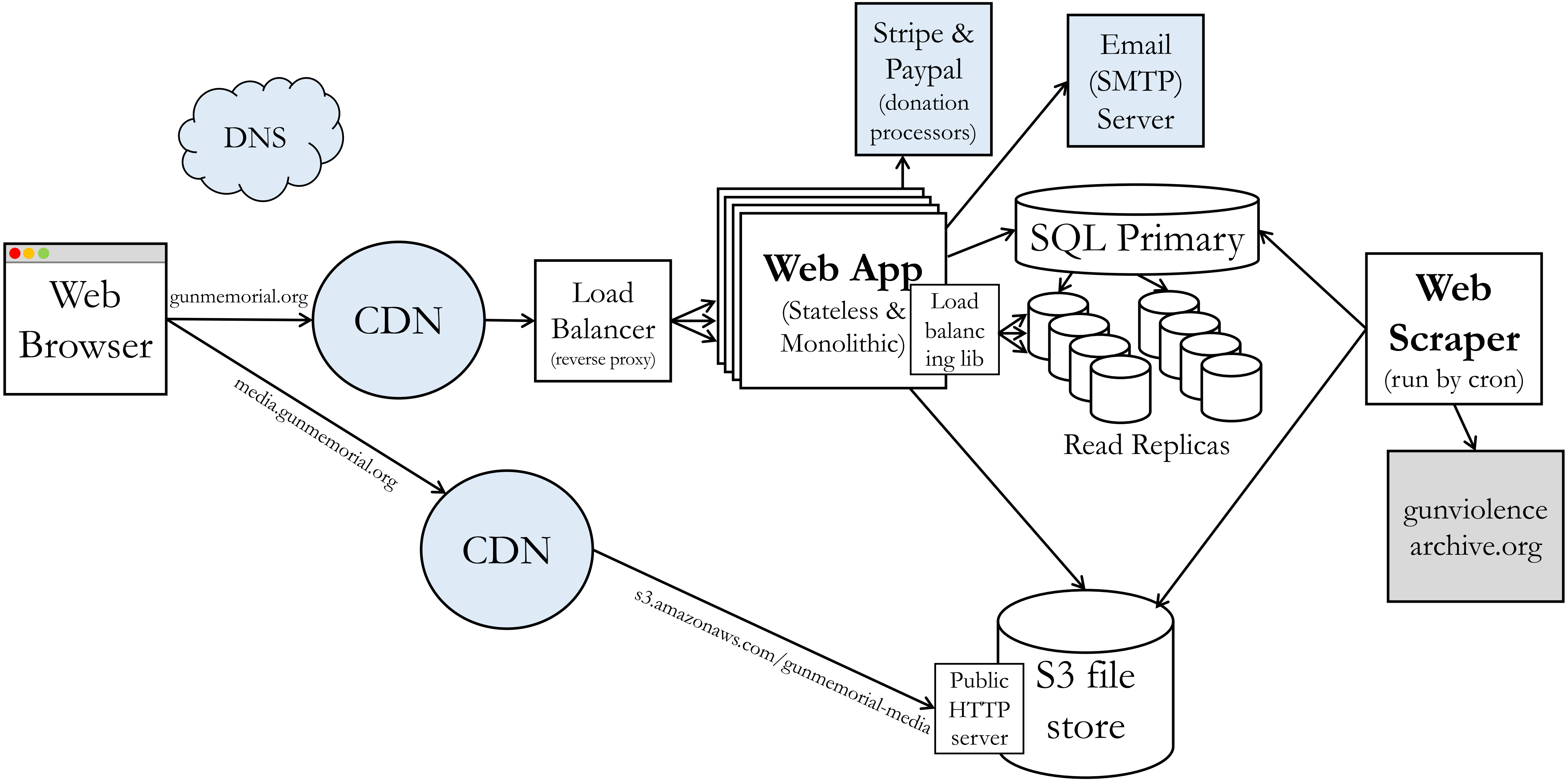


More front-end caching

- CDN in front of web app to cache HTML.
- CDN for all media files, even on detail pages.

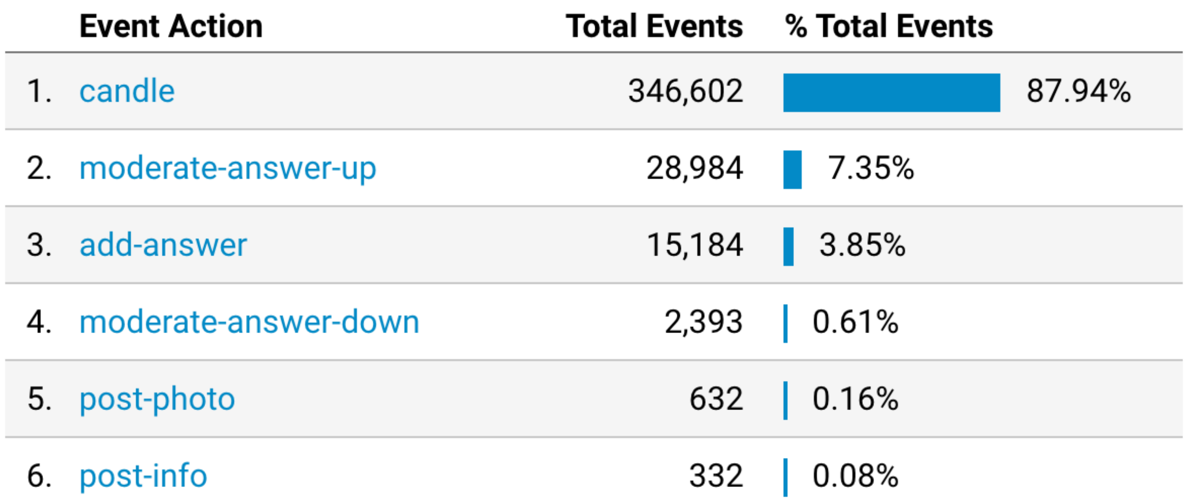


Final scalable design



Can a single SQL database handle the write load?

Month of April UI events (leading to DB writes):

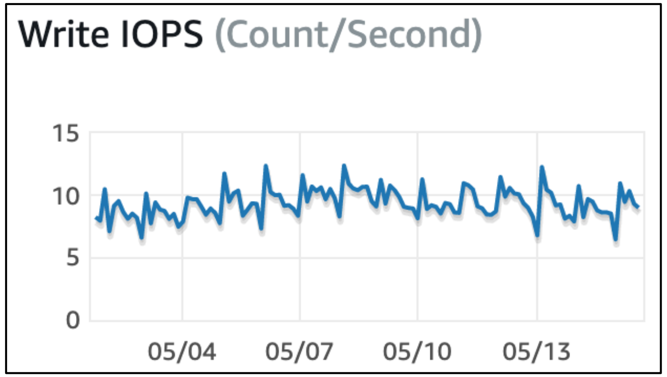


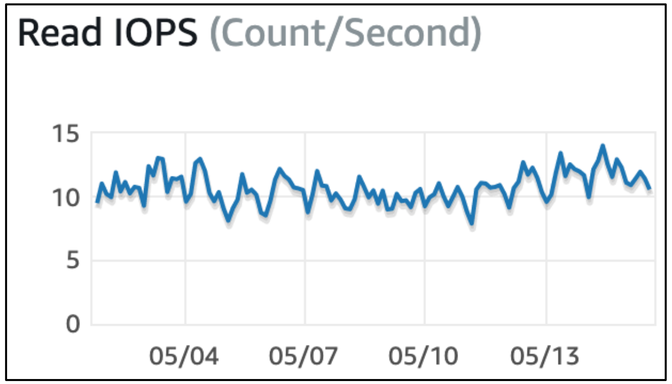
	Event Action	Total Events	% Total Events
1.	candle	346,602	87.94%
2.	moderate-answer-up	28,984	7.35%
3.	add-answer	15,184	3.85%
4.	moderate-answer-down	2,393	0.61%
5.	post-photo	632	0.16%
6.	post-info	332	0.08%

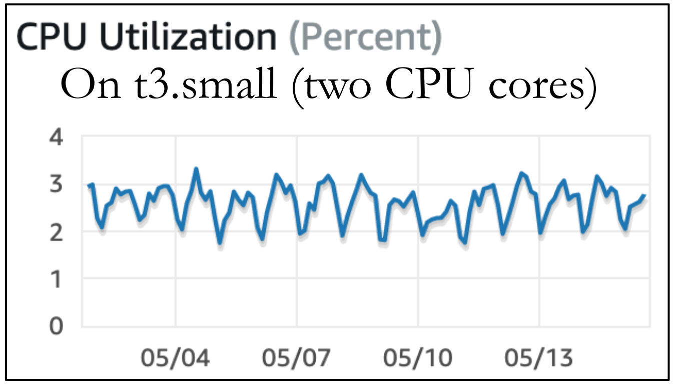
There are also DB writes to add new victims to the database, but this negligible and does not scale with traffic. Visitor actions are the main concern for scaling.

- At 200x the load, we'd expect about 400k×200 = 80M events/month
- 80M/month × 1 month/2.6M sec ≅ **30 DB writes per second**
- This is definitely achievable:
  - Magnetic disk can do ~100 IOPS
  - SSD can do > 5,000 IOPS [ref.]
- But this is just a theoretical projection. It's better to look at the load in practice…

Empirical scaling analysis *(real traffic on t3.small)*


Write IOPS (Count/Second)


Read IOPS (Count/Second)


CPU Utilization (Percent)

On t3.small (two CPU cores)

- Data at left is from two weeks in May 2020, running the database on a t3.small instance.
- Remember, our goal is to scale traffic by 200x.
- AWS allows DB instances with up to 32k IOPS.
  - Can a single machine's storage handle 200x the load?
  - **Yes!** 200x more load would be just 2k IOPS.
- The biggest DB instance available (r5.24xlarge) has 96 CPU cores instead of just two.
  - Can a single machine's CPU handle 200x the load?
  - **Yes!** Two CPU cores can handle 30x more load. 48x more CPU cores might handle 1,400x the load.

NGVM is easy to scale. Why?

STOP and THINK

- Traffic is mostly reads.
- Visitors are not logged in.
    - There are no personal recommendations or user behavior models.
    - Each user gets the same HTML, and responses can be cached in CDN.
- Effects of visitor actions (lighting candles, leaving comments) need not be visible immediately to other visitors. Caching is possible.
- Users don't interact directly with each other. No user notifications.
- Memorial pages are independent of each other.
- Data size does not scale with traffic (number of memorial pages is fixed). Legacy.com would be more difficult to scale.
- Writes don't involve any transactions.

Recap

- Showed NVGM architecture design case study.
- It's another article publishing system, so arch is similar to Wikipedia.
  - Caching and load balancers on frontend,
  - Stateless app,
  - SQL DB with read-replicas.
- S3 file store was used for large media files (photos).