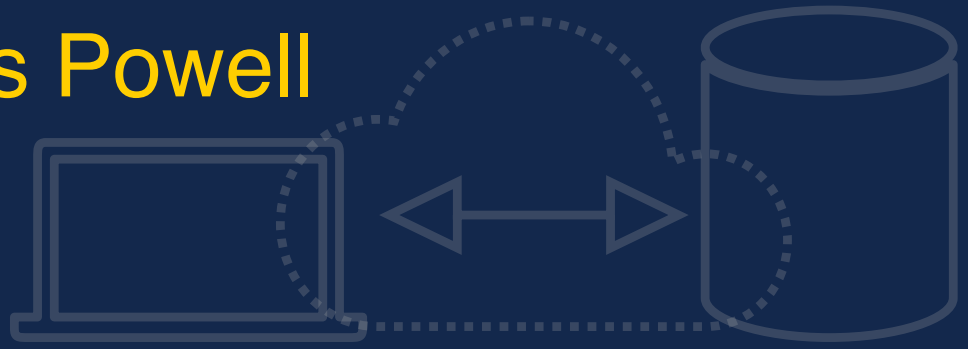
State and Session Basics

with Thomas Powell



Traditional Web Programming Review

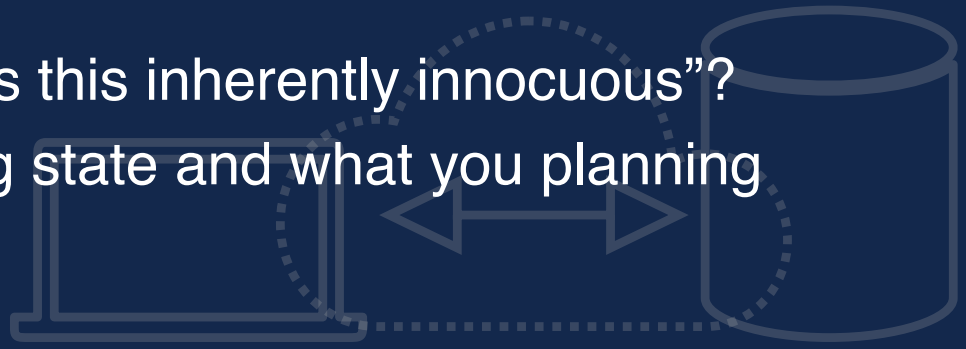
- You have to get data in (from user-agent to server)
  - Input: <form> data, URLs, HTTP headers
- Then process the data, perform some task, etc.
- You have get data out (from server to user-agent)
  - Output: Some form of data usually HTML, GIF, JPEG, etc. and the appropriate header (MIME type ,cookie, etc.)
- However, a real change will be to fix the stateless nature of HTTP if you want to do anything meaningful
  - Approaches: Sessionization via cookies, URLs, hidden fields, etc.

Stateless Implications

- Statefulness implies being able to uniquely related the activities of a particular user somehow

- In some sense to have state is to have “memory” of someone and their activities
  - “Hi Thomas welcome back to Starbucks”
    “Hey user here is the contents of your shopping cart”
  - “Wow pretty ‘interesting’ all those sites you go to…”

- Described this way the question is “Is this inherently innocuous”? Kind of depends on why your holding state and what you planning on doing with it

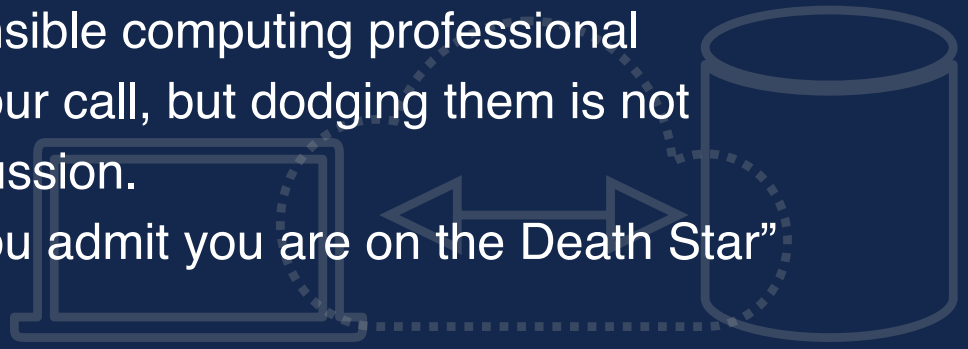


Trust Relationships Preview

- As typical humans I think we assume that when we interact with a person or organization our interaction is with that person or organization.
  - Some level of trust is assumed between the parties in an interaction.

- If trust is breached even if just in belief I should expect trouble

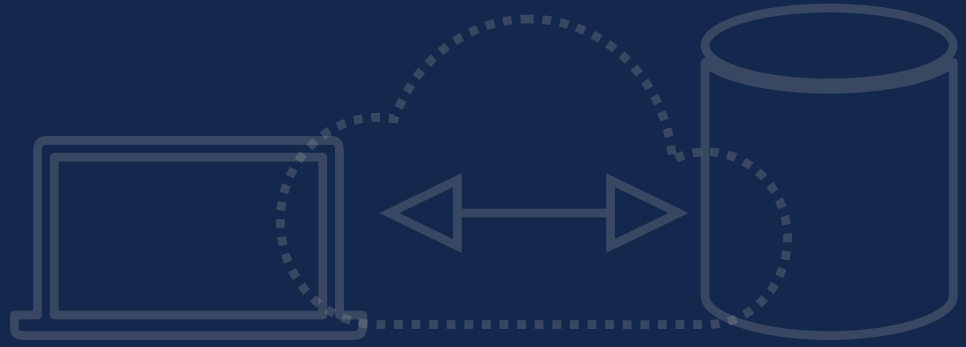
- State management inherently gets into trust relationships.
  - Keeping it technical we can focus on the how the relationship is implemented (ex. Cookies, Fingerprinting, Device Ids, etc.)
  - Discussing the transparency and values inherent to these trust relationships is required to be a responsible computing professional
  - Deciding about these ideas is 100% your call, but dodging them is not allowed so brace yourself for that discussion.
    - “It’s ok to be a Death Star Dev if you admit you are on the Death Star”



Addressing HTTP Statelessness

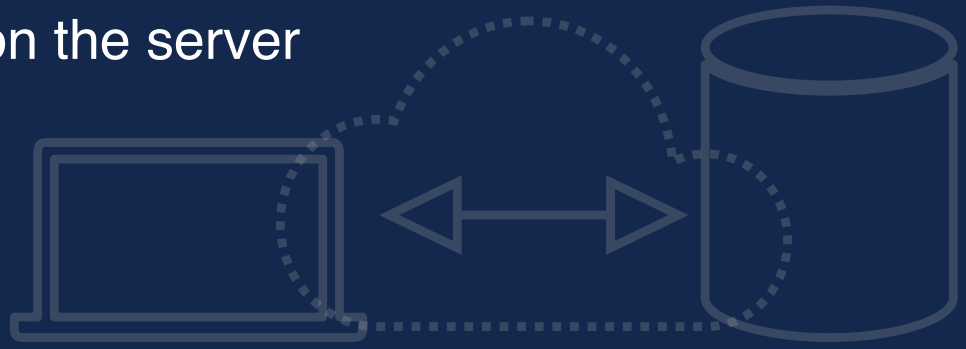
- HTTP being stateless, in other words having no memory from page view to page view, can make Web programming a hassle.

- Techniques to address state include:
  - Hidden Form fields (aka Postback systems)
  - "URL Rewriting" (aka Dirty URLs)
  - Cookies
    - Session Cookies
    - Persistent cookies
  - Dangerously - client tech



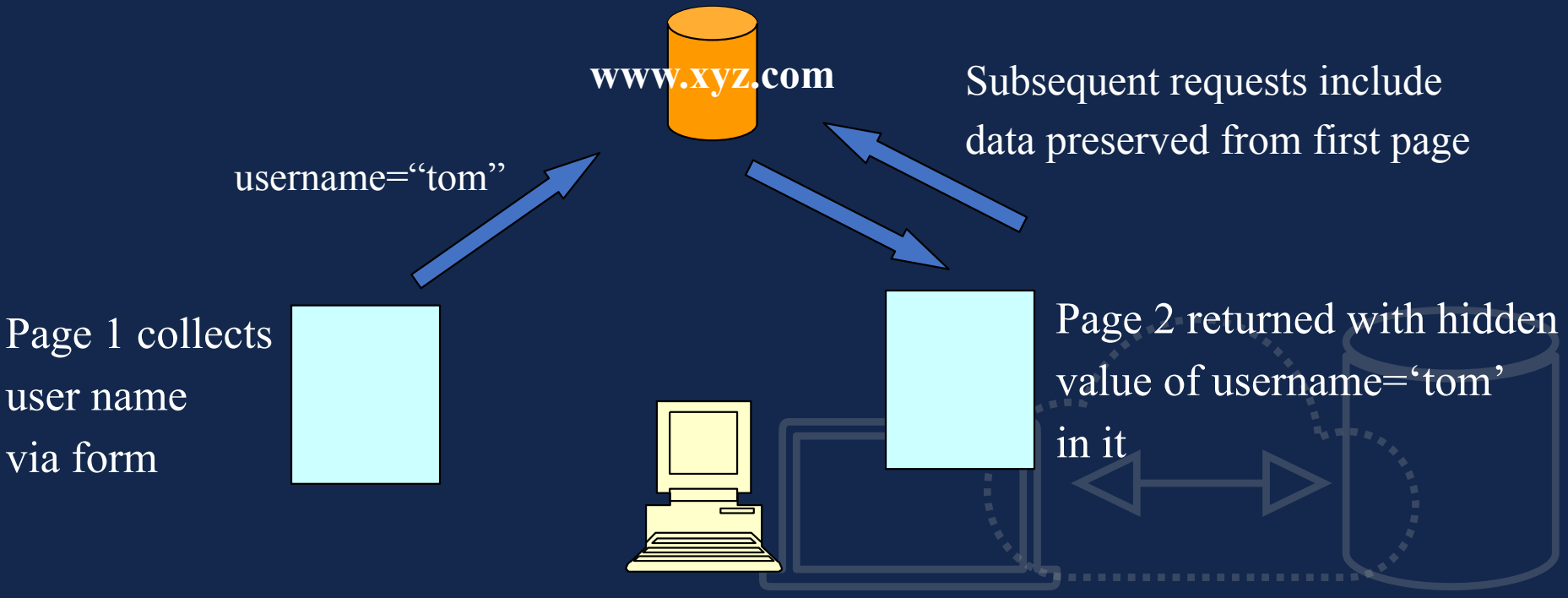
Implementing State

- Implementing state is not crazy difficult, but you do need to be careful and it is easy to do it wrong.
  - Like Authentication it is something people can do but should they!?
- Most modern programming environments such as ASP.NET, JSP, PHP etc. provide a ***session*** metaphor which will abstract away many of the details of state preservation.
  - Sessions are most often implemented as cookies which store a unique ID referencing data stored on the server



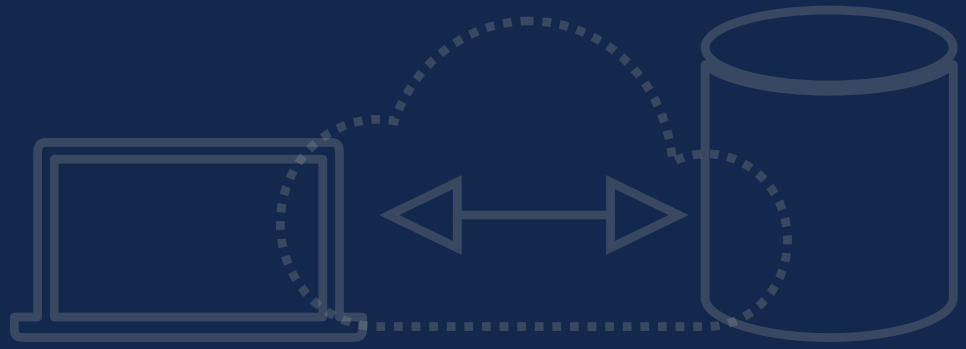
Hidden Form Fields

- Forms contain <input type="hidden" name="cartid" value="78Ccad786"> this data is passed along with submission
- Subsequent page loads are rewritten with hidden fields in them



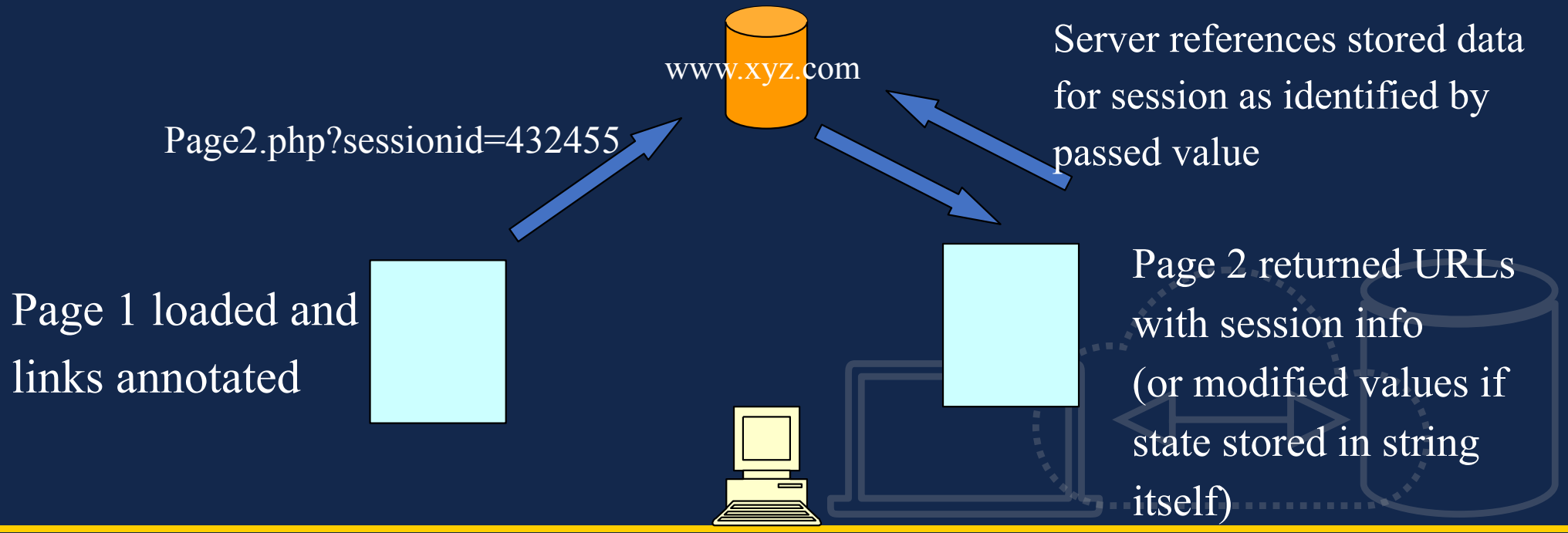
Hidden Form Field Notes

- Open for easy parameter tampering
  - Countermeasure: integrity checks
- Does not preserve state information across a user visit
- Requires that a form be used to trigger the pass back of the hidden data
  - .NET uses hidden form fields (a viewstate value) and wrap the whole page in a single giant form



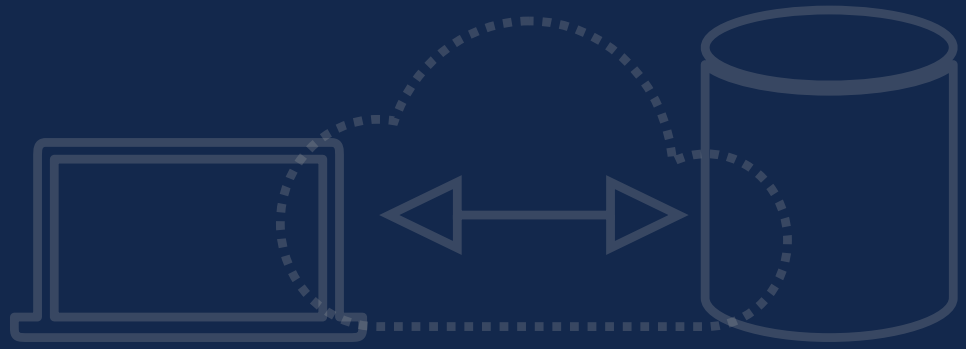
State Preservation via "Dirty URLs"

- All links are rewritten with query strings in them
  Page 1
  Page 2
- All forms pass the value via the query string as well (either direct action change or via hidden field)



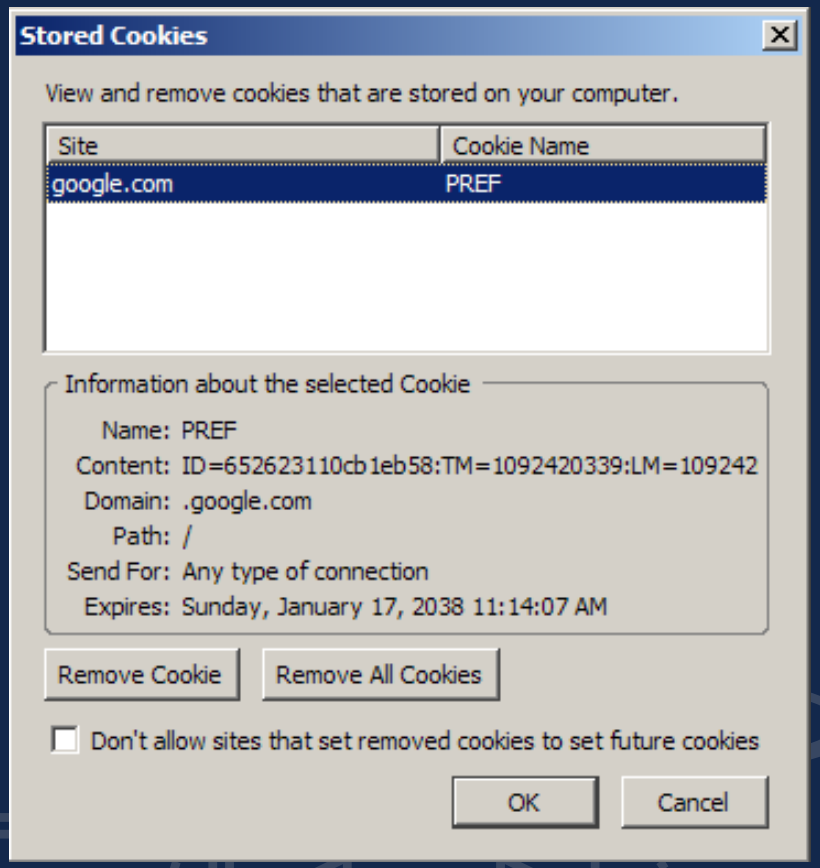
Dirty URL state management

- Open for easy tampering
  - Countermeasure: integrity checks
- Does not preserve state information across a user visit
- Does not promote URL usability, marketing, etc.
- Without dereferencing to server stored data has significant limit to the amount of data that can be saved (URL limit around 2K)

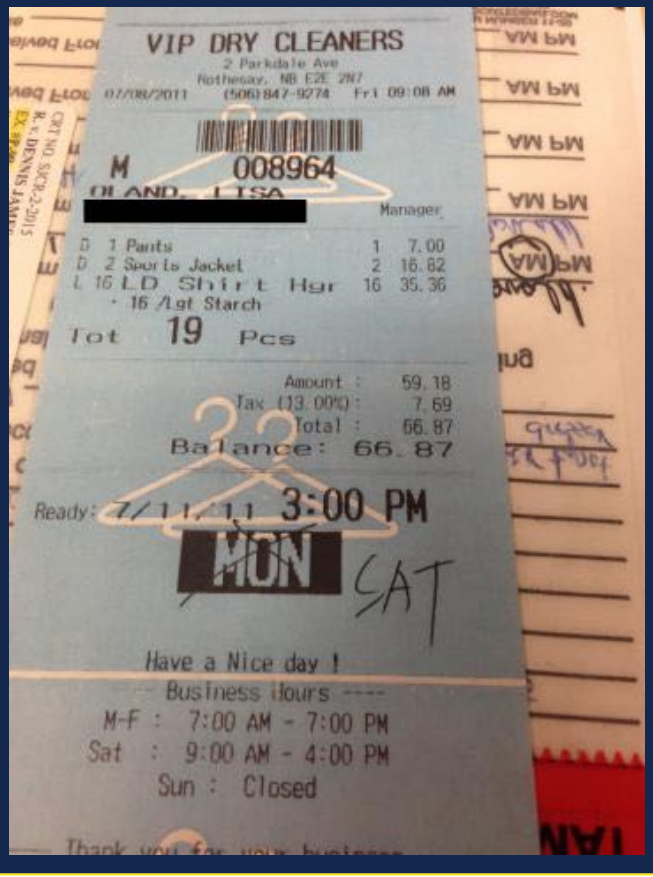


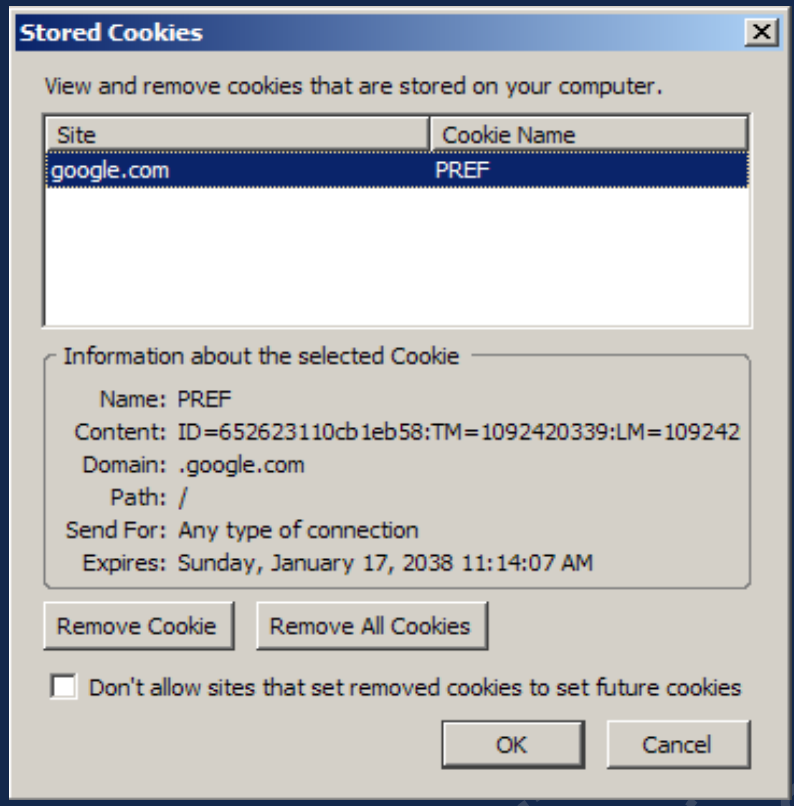
Cookies

- Cookies are simple short pieces of data stored on a user's system often in a file (cookies.txt) or a directory C:\Documents and Settings\userid\Cookies with individual cookie files
  - Example contents of a Google cookie
    - PREFID=6b476a56502ce137:TM=1088552548:LM=1088552548:S=FUe7V3k3CtQYjEeHgoogle.com/1536261887833632111634405477774429646386*
  - Notice that often cookies do not store the actual data to be tracked but an id that references data stored on a server



More Related Than You Think?





Main difference is your dry cleaner's biz model doesn't revolve around figuring you out!

There isn't a good/bad here, just understand what the goal of the biz actually is. Hint: It isn't search, giving you free photo space, etc. that is funded by / an attractor for their actual biz. We must acknowledge this to address why users behave why they do in regards to cookies and analytical tracking.

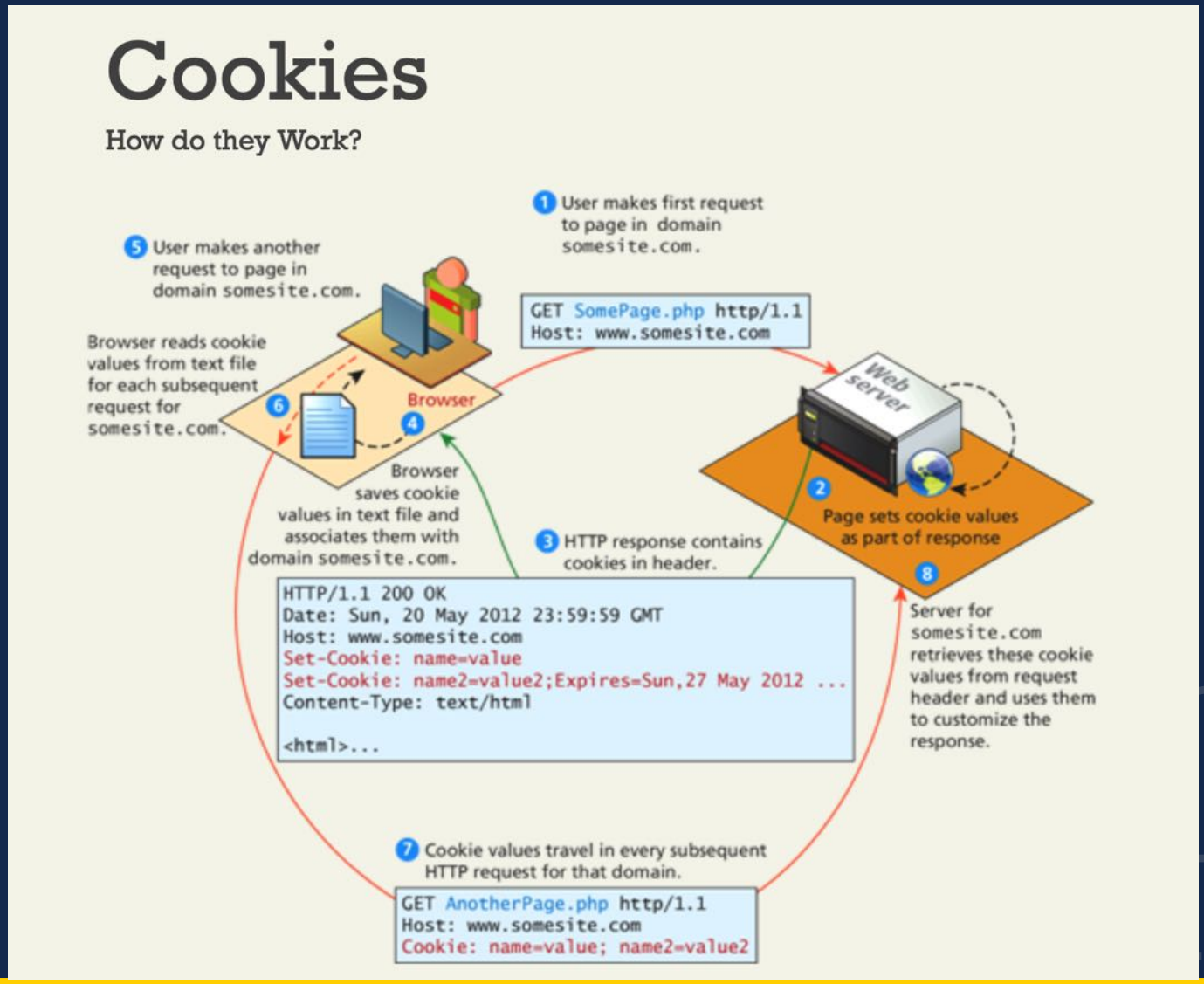
Cookies

- A cookie is typically issued to a browser via the HTTP header
  - Set-Cookie: *NAME*=*VALUE*; expires=*DATE*; path=*PATH*; domain=*DOMAIN_NAME*; secure

- Most of the time we would not set cookies manually in an HTTP response (though you could), instead the Web programming environment you employ will provide some command.

  - To issue a cookie in PHP use `setcookie(name, value);`
  - Example: `setcookie('leadstooge','moe');`
- However, not that JavaScript can also easily set cookies client-side via the `document.cookie` object

Cookies - How the Work Take 2



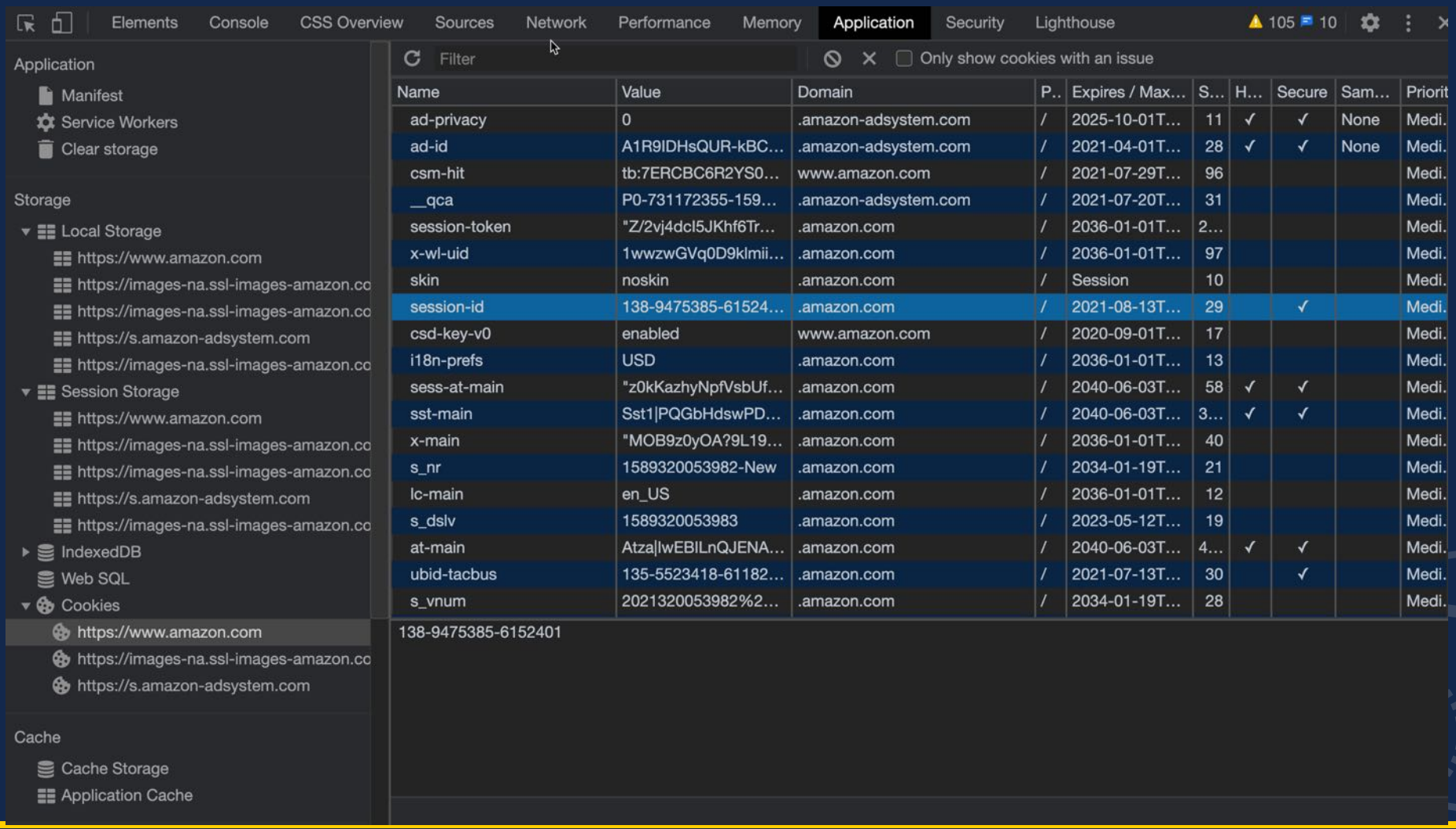
Cookie Lifetimes

- The lifetime of the cookie, in other words how long it is valid, has much to do with the value and danger of the cookie.
- **Session Cookies** (aka Memory Cookies) generally have shorter lifetimes as they should be deleted at the end of a browsing session
  - End being time based or shutting browser or tab
- **Persistent Cookies** (aka Disk Cookies) have a long lifetime and may only be cleared by a browser setting purge, some significant longer time period or by changing a machine

Cookie Data

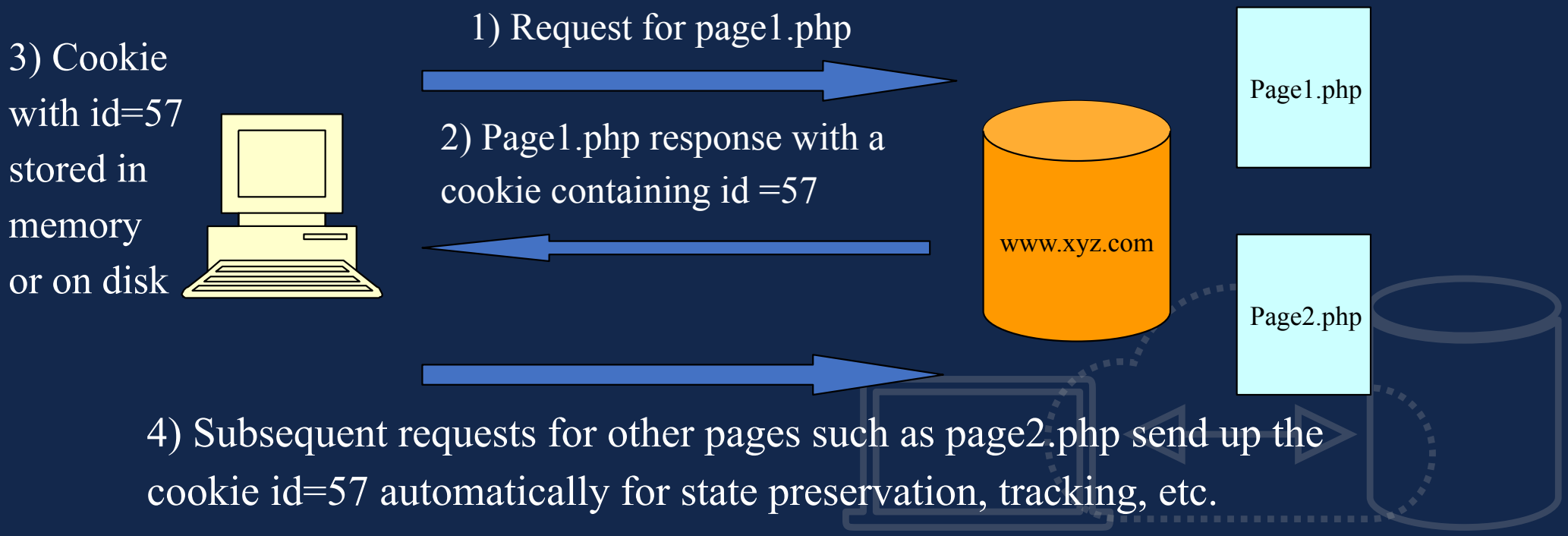
- The cookie can contain data itself or it contain some token or identifier that dereferences data
  - Cookie: name=Thomas vs Cookie: id=23dsa43c3
- Obviously storing the data itself in the cookie can be limiting for size purposes as well as potentially disclosing
- Dereferenced values such as SessionId values however are not inherently safe unless strengthened as if observed (non-SSL, logged, accessed via JS, etc.) they can be replayed by a potential intruder

Raid My Cookie Jar!



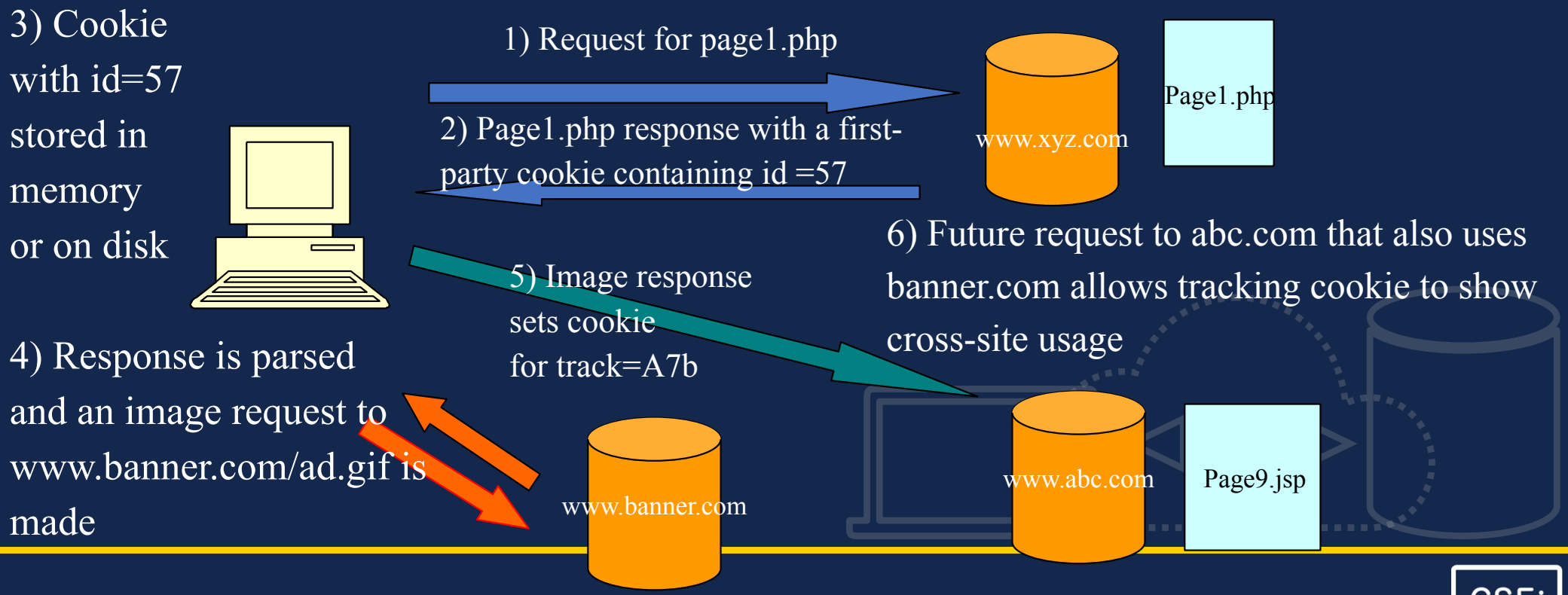
First Party Cookie Example

- Once the cookie is set it is transferred back and forth from the browser to the site for every request within the domain and path that it is set for.



Third Party Cookie Example

- Cookies can be set by any HTTP request including images, scripts, CSS, etc.
  - This is often used in the case of Third-party cookie issued from a banner ad or tracking site

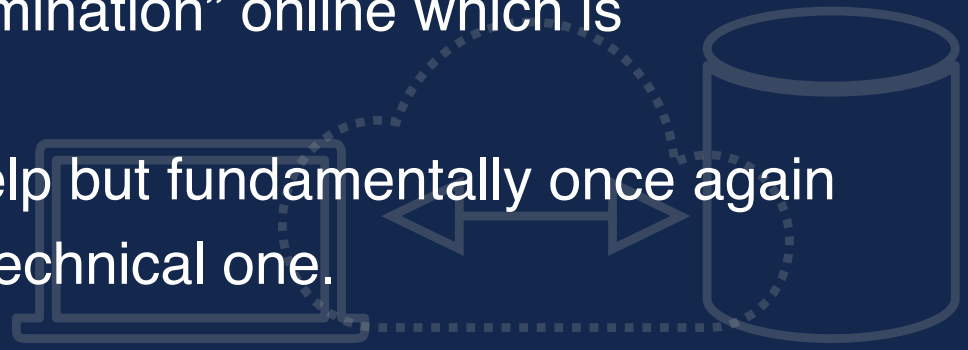


Privacy Concern with Cookies

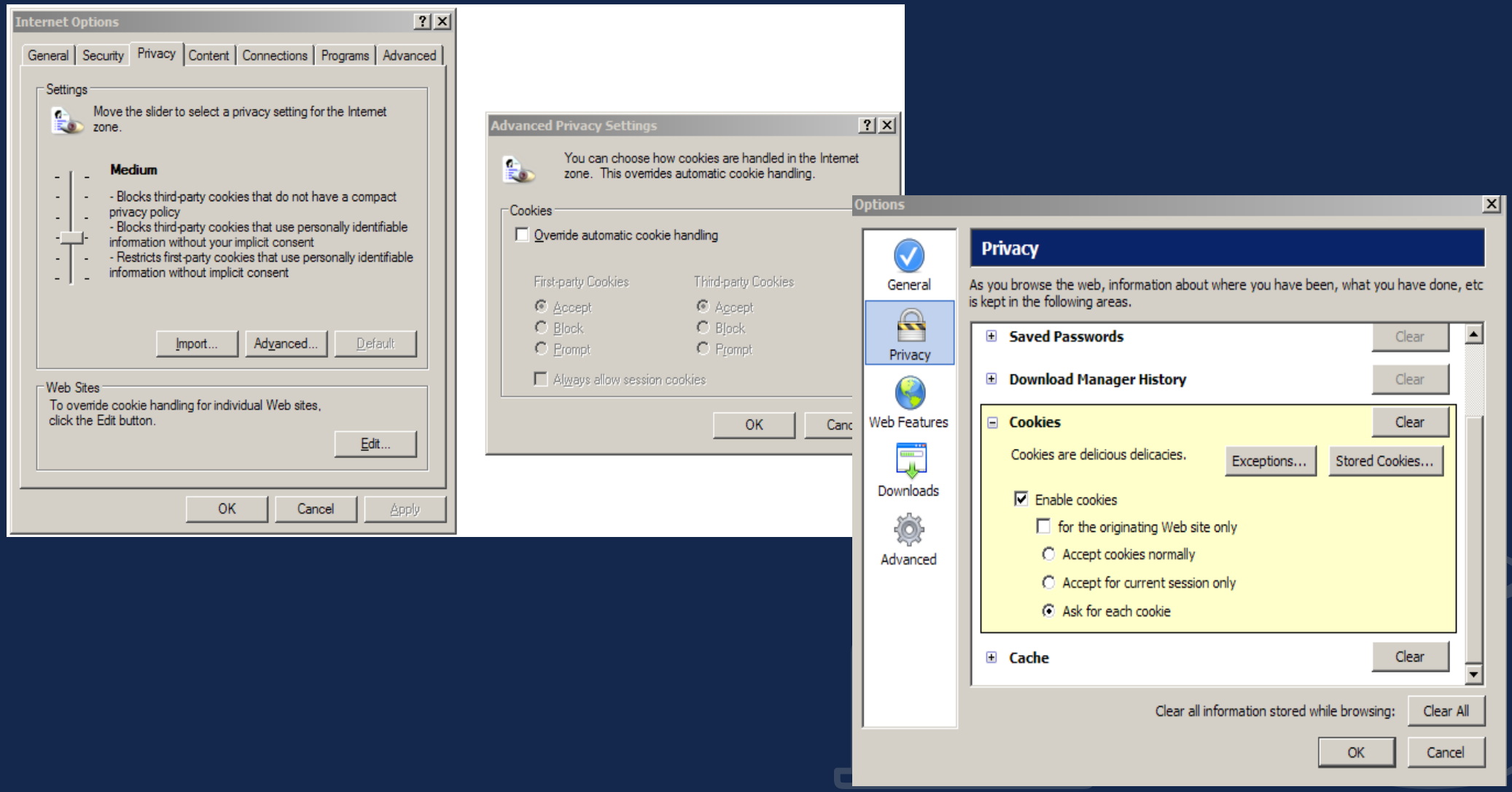
- Third party cookies represent a possibility to track user activity across a range of sites
  - Typically these sites are part of some “network” – like an ad network
- The cookies are also set often via a clear pixel GIF so the user may not see what they should be trying to block
  - These cookie setting clear pixels are dubbed “Web bugs” or just “bugs”
  - Typically they are used in a hosted Web analytics services

Privacy Concern with Cookies

- The privacy concern with cookies come from the fact that you may register at a particular site like the last example xyz.com and they then associate your personal information with the cookie-id. If the personal information is then shared to other network participating sites they can start building a detailed profile on your usage habits
  - Because of this many users want to block third party cookies, but the real problem is giving out your private information and knowing what happens to it afterwards
  - We lack “informational self-determination” online which is unfortunate
  - Browsers, tools, tech, etc. can help but fundamentally once again we see a social issue and not a technical one.



IE Cookie Related Settings



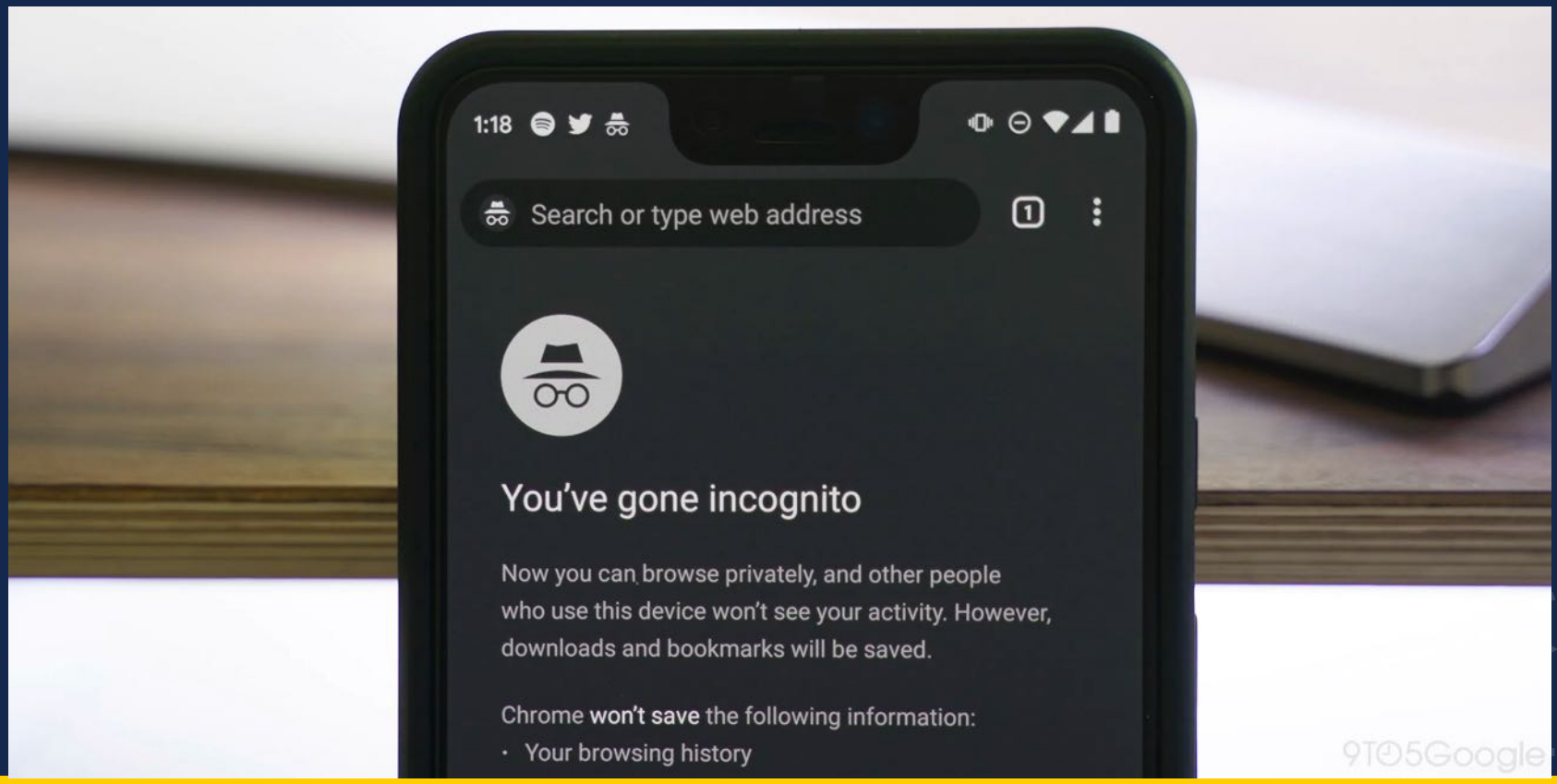

MY PRIVACY POLICY IS
COOKIES
ALWAYS ACCEPT COOKIES
imgflip.com

After Users Hear About Cookie Abuse


NO, I DON'T GIVE CONSENT TO MY PERSONAL DATA.
NO COOKIES.
NO TRACKING.
NO BROWSING HISTORY.
TOM FISH BURNE
THE LESS THIS SITE KNOWS ABOUT ME THE BETTER.
CLICK
HEY, WHY ISN'T THIS SHOPPING EXPERIENCE MORE PERSONALIZED?
© marketoonist.com

Safety or Signal!?

Time to pay attention!

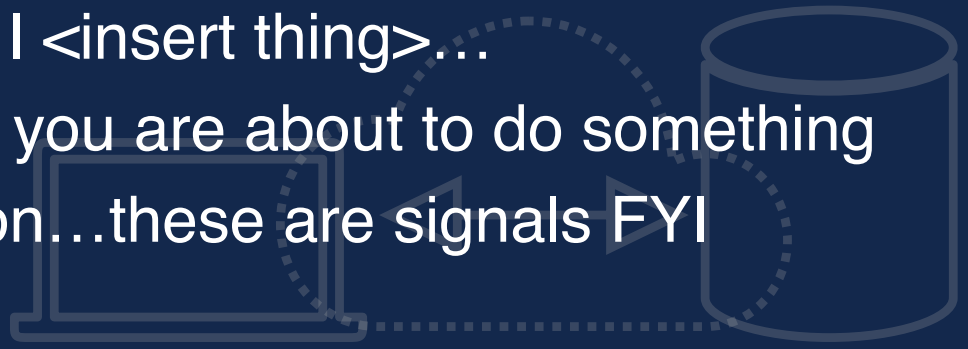


Anonymity and State

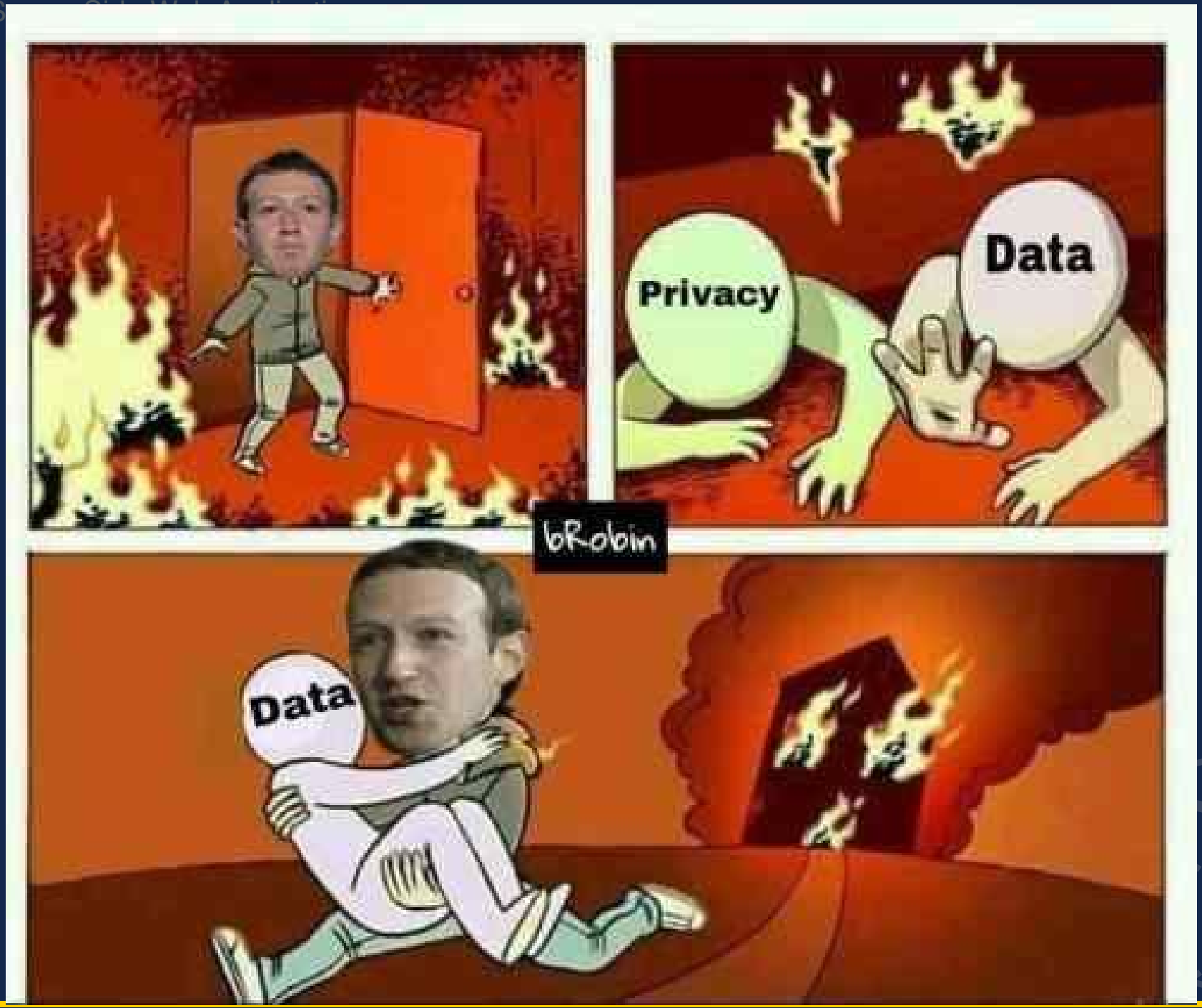
- If I had you an identifier like d3q4dasd to uniquely identify you that is not breaking your perception of anonymity on the Internet
- If however I can associate that identifier d3q4dasd with personal identifying activities such as your phone number, location, username, browser type, etc. I can start to deanonymize you for any reason I may have
- This might be ok if I feel I have given my identifying info with a particular site, but what about if they give it to someone else?

Reality of Anonymity

- The reality is that you are NOT anonymous online in fact I would posit you are more anonymous practically speaking IRL than you are online
  - Someone somewhere knows what you've done (CUE OMINOUS MUSIC)
    - Also it's logged (maybe for a very long time)
  - Well I am not that dumb! I dump my cookies. I go into incognito mode. I use a VPN. I <insert thing>…
    - Thanks for letting me know you are about to do something or a particular type of person…these are signals FYI

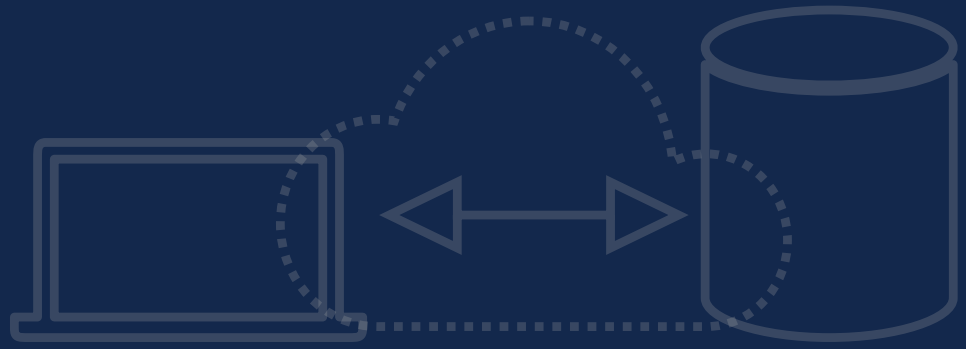


OG MEME TIME


Privacy
Data
loRobin
Data

The Reasonableness of Expecting Anonymity

- The IRL Story of Shy Thomas as his Gorilla Mask
- Morale: The expectation of being able to do many important activities without verifiable identify in IRL is very low
- Logical Conclusion: If online is now as important as IRL for many activities wouldn't the same morale follow?



Cookie Limits

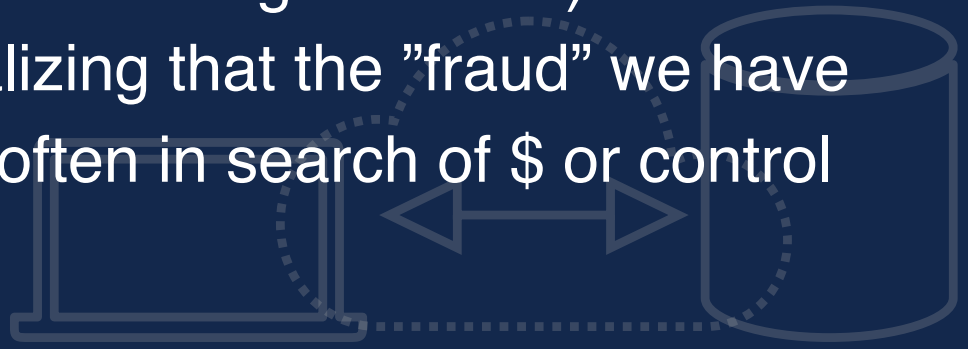
- The browser and server may put some limits on cookies
  - Commonly held that browser:
    - Can store a limited number of cookies
      - Cookies received after browser limit are deleted starting the least used
    - A single cookie can be no more than a few K in size (check)
  - Commonly held that a browser store –or- server will accept no more than X cookies for a particular host.domain combo (X likely ~ 20)
    - Though web.xyz.com and store.xyz.com are different and thus could have 20 each
  - The reality is these limits may not in fact be true as they are implementation dependent
- Commonly held development practice suggests using as few cookies per domain and only storing a sessionid or related value with the bulk of tracked information stored server side.

Session Jargon

- A user **visit** generally starts a session.
  - A token/id is generated and saved with the user
- As the visit proceed the users does things (clicks links, loads pages, etc.)
  - Various activities and data values are saved and associated with the session often being stored in a **session store** (client or server side)
- The session eventually ends when
  - The user performs some session destruction activity like a logout where code is run to clear the session
  - The user stops interacting with the site/app for an extended period of time (**Session Timeout**)
  - The user disconnects in a manner that destroys the session token (close tab/ browser for memory or wipes tokens for persistent storage)

Observation

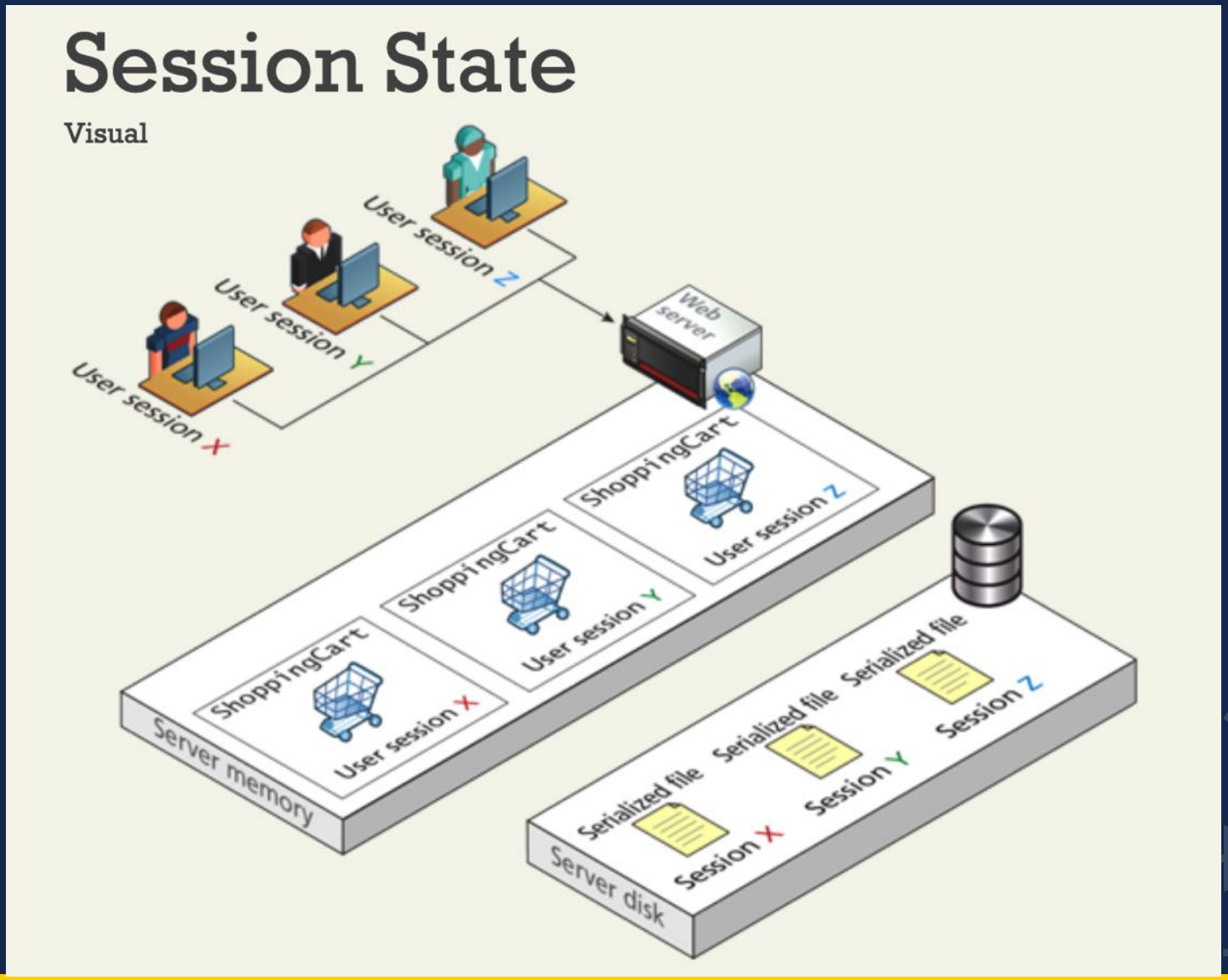
- How do I know identity online?
  - Email, profile, etc.
- How do I know who/what is real or not?
  - Ex: Catfishing
  - Ex: Socket Puppets & Concern Trolling
  - Ex: Fake People and Bots
  - Let's go take a quick look
- We are currently at a level X (X is some high number) emergency where people are realizing that the "fraud" we have been perpetrating in cyberspace often in search of $ or control has some bad implications



Sessions

- While it is possible to do all the management of most sites using cookies (or even hidden fields or dirty URLs) it is much better to use sessions
- A well implemented session management system abstracts away how data is preserved from page to page (typically in a cookie) and stores the saved information server side only referencing via a SESSIONID value in the cookie
  - You could of course build this easily enough by storing some database-id in a cookie yourself and every page load doing a query against the database reading or modifying data in regards to the user's state
- A significant benefit of sessions not storing the data in the cookie is that it helps guard against session hijacking
  - Assuming that session ids are not easily guessable
  - Even better to "brand" the session to the user somehow

Sessions


Session State
Visual
User session Z
User session Y
User session X
Web server
ShoppingCart
User session Z
ShoppingCart
User session Y
ShoppingCart
User session X
Server memory
Serialized file
Session Z
Serialized file
Session Y
Serialized file
Session X
Server disk

Session State

All modern web development environments provide some type of session state mechanism.

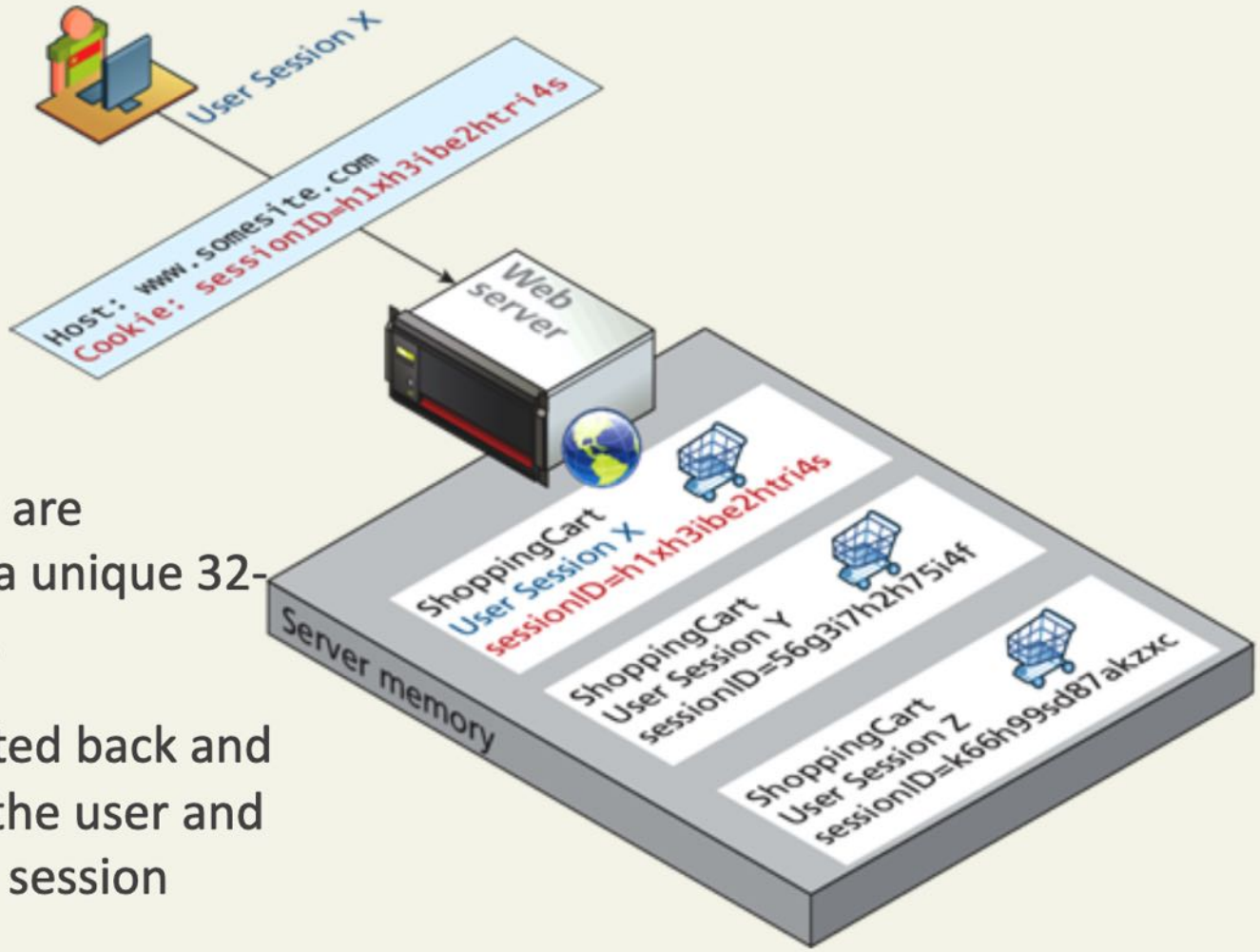
Session state is a server-based state mechanism that lets web applications store and retrieve objects of any type for each unique user session.

Session state is ideal for storing more complex objects or data structures that are associated with a user session.

- In PHP, session state is available to the via the $_SESSION variable
- Must use session_start() to enable sessions.

How does state session work?

It's magic right?



Sessions in PHP are identified with a unique 32-byte session ID.

This is transmitted back and forth between the user and the server via a session cookie

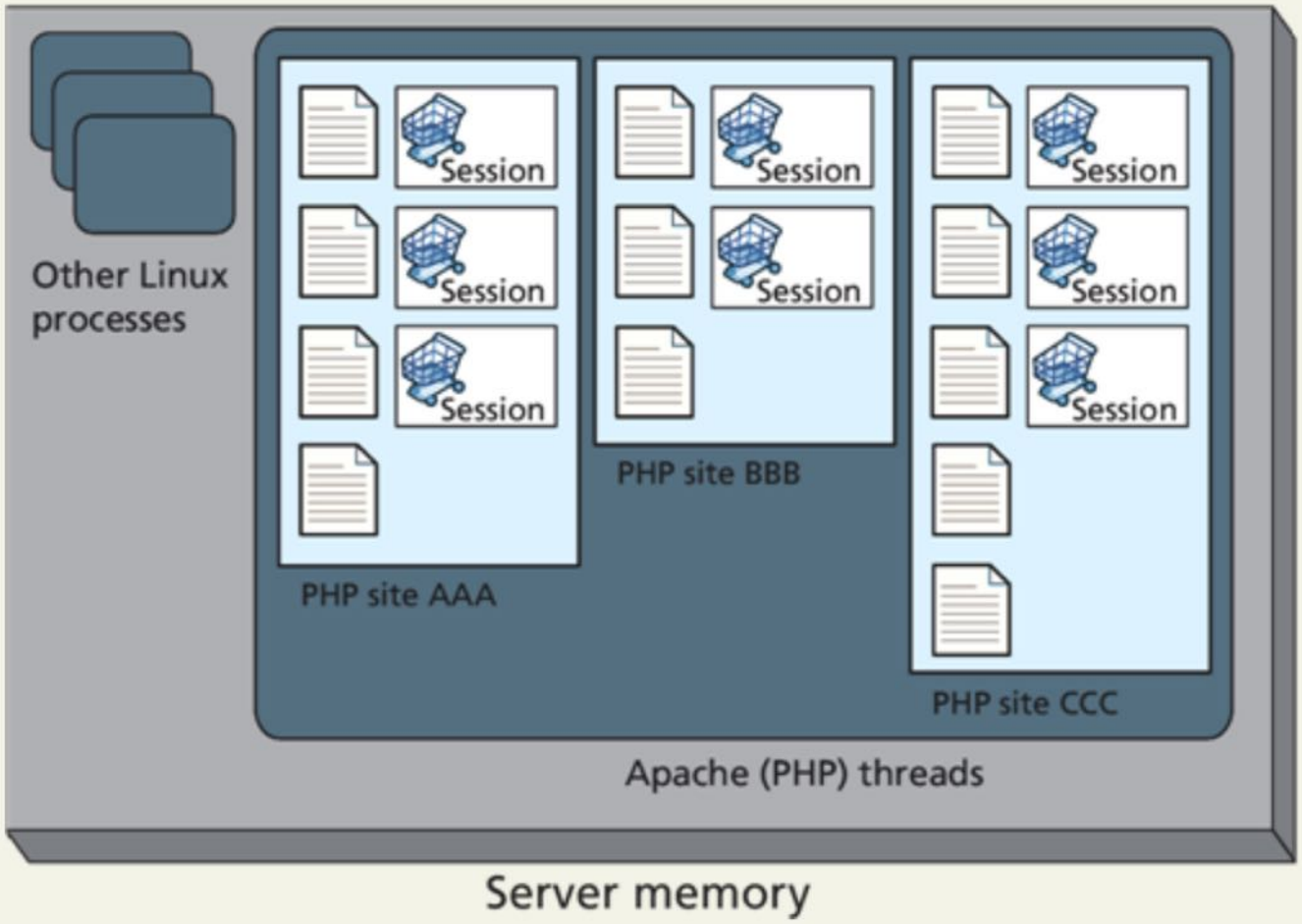
How does state session work?

It's magic right?

- For a brand new session, PHP assigns an initially empty dictionary-style collection that can be used to hold any state values for this session.
- When the request processing is finished, the session state is saved to some type of state storage mechanism, called a session state provider
- When a new request is received for an already existing session, the session's dictionary collection is filled with the previously saved session data from the session state provider.

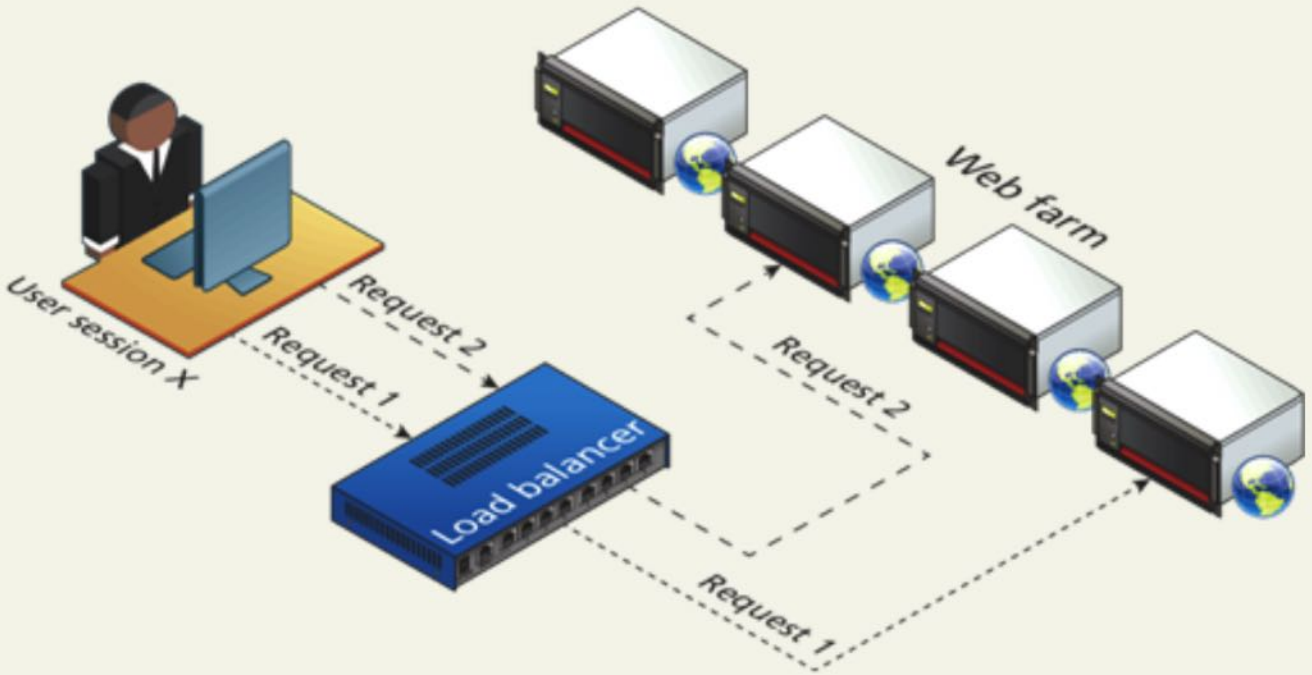
Session Storage

Applications and Server Memory



Session Storage

Web Farm Sessions: Visualizing the problem



Session Storage

Visualizing the problem

There are effectively two categories of solution to this problem.

1. Configure the load balancer to be "session aware" and relate all requests using a session to the same server.

2. Use a shared location to store sessions, either in a database, memcache, or some other shared session state mechanism

Session Storage

Visualizing the problem

There are effectively two categories of solution to this problem.

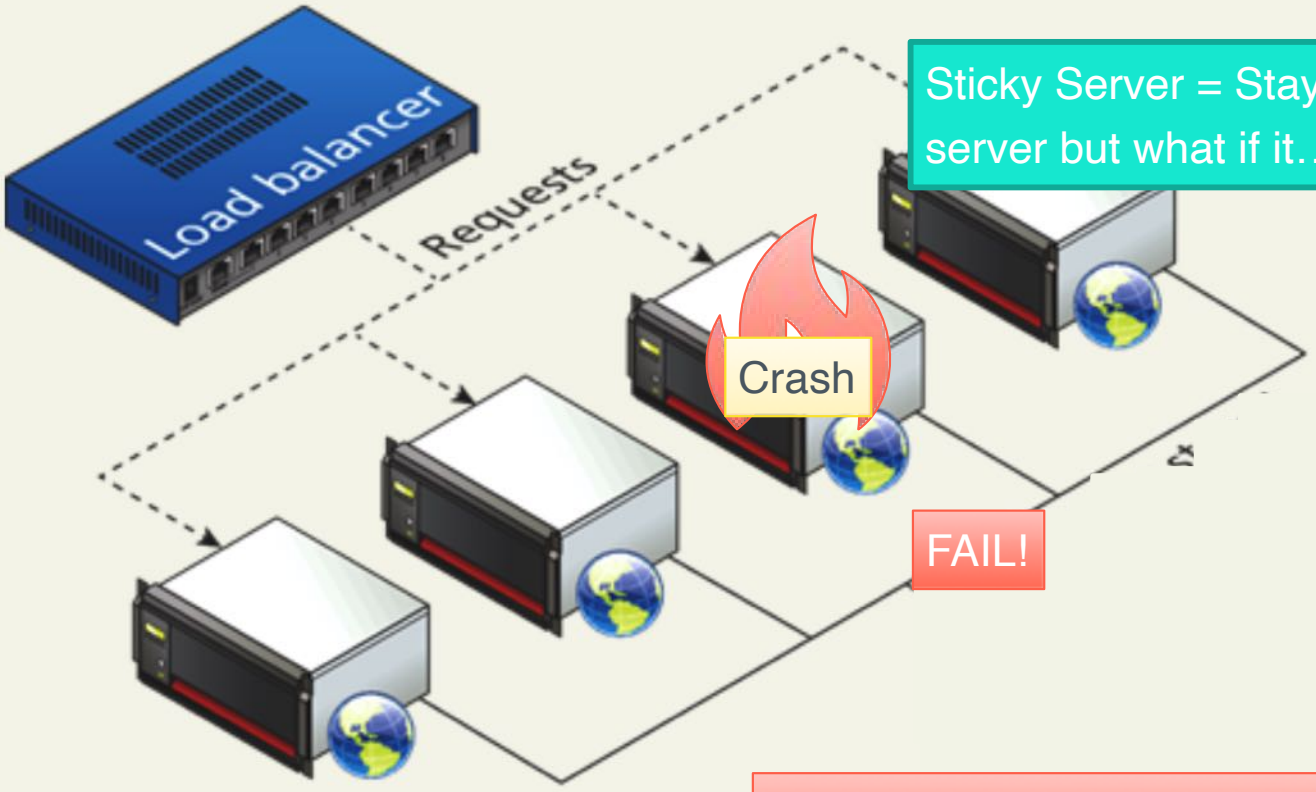
1. Configure the load balancer to be "session aware" and relate all requests using a session to the same server.

2. Use a shared location to store sessions, either in a database, memcache, or some other shared session state mechanism

Session Storage

Simple Solution with “sticky server” concept

Upon Access Balance and Stay on a Sticky Server



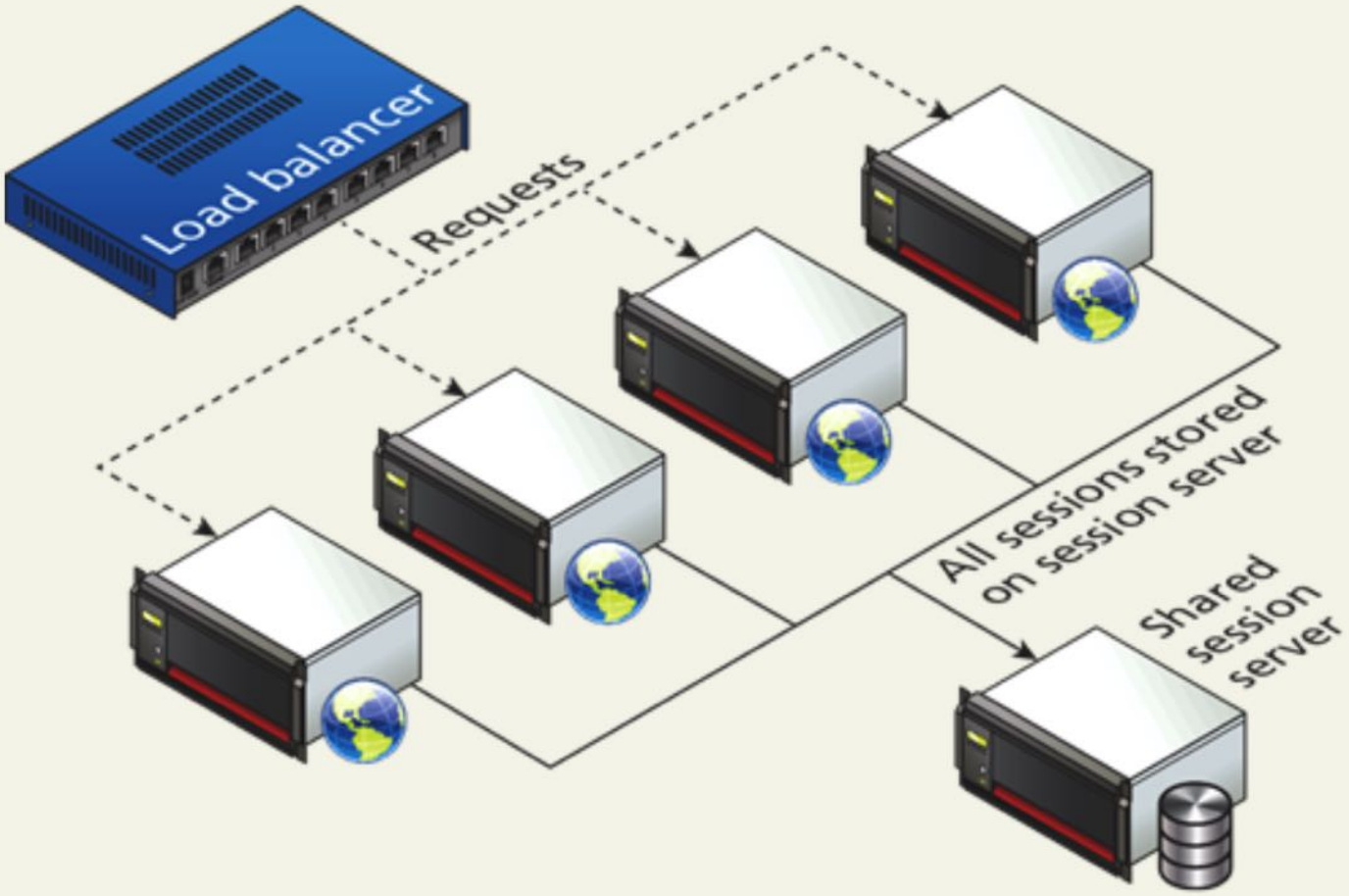
Sticky Server = Stay only on this server but what if it…crashes!?

Decentralized state can be a problem

Session Storage

Centralized State

Shared location with memcache



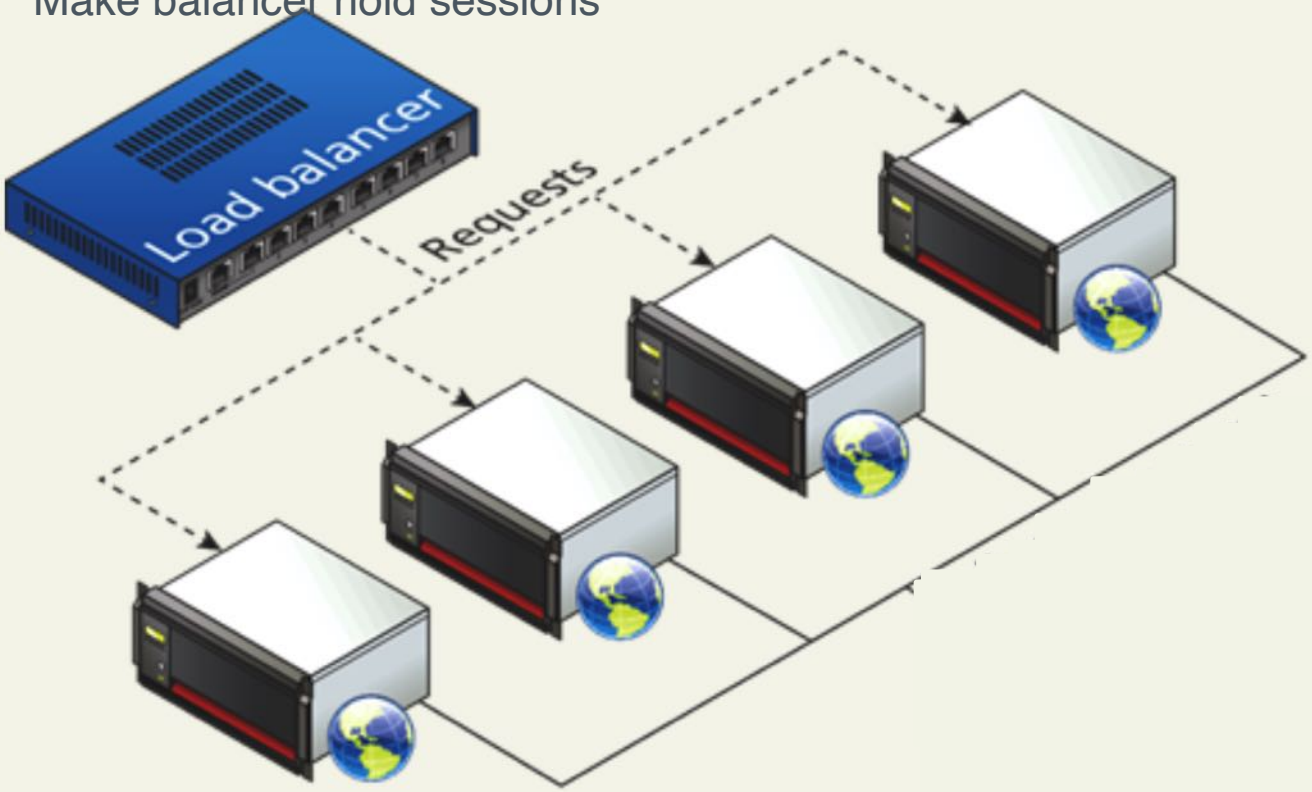
Session Storage

Simple Solution with balance storage

Centralized State

Do not confuse with JS API of same name!

Make balancer hold sessions

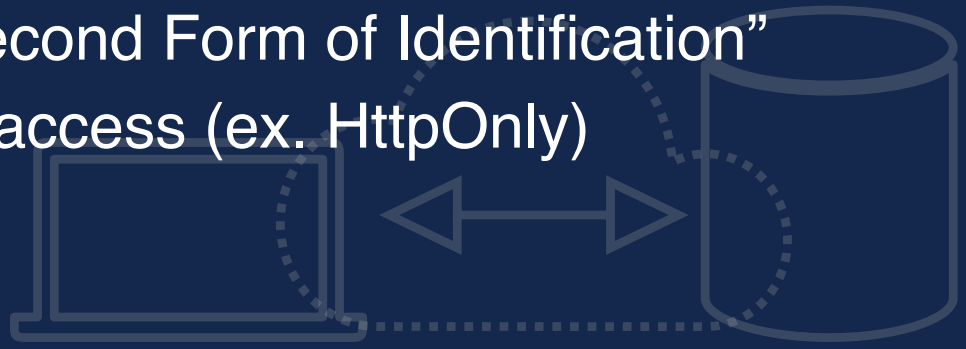


Note: Same as memcached just less programatic

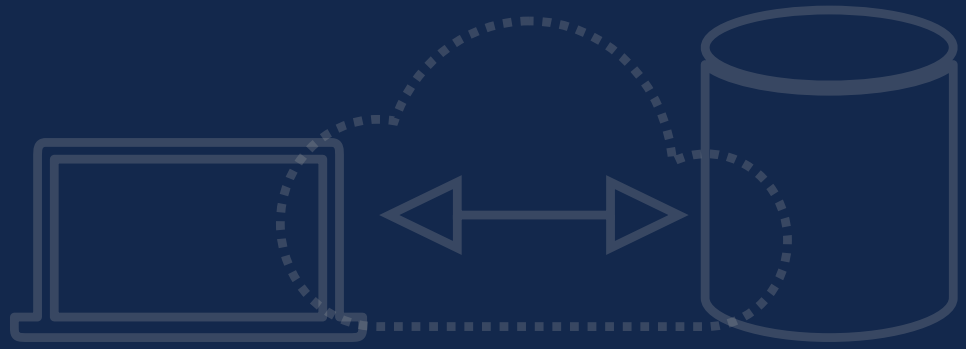
Session Hijack

- If an adversary can get access to a session token (ex: cookie, query string, access to storage) they can potentially hijack a live session or even replay it

- Countermeasures to hijack are numerous:
  - Short Session Lifetimes
  - Transmission Encryption (SSL)
  - Multiple Factors to relate the session to (Time, Location, Browser type, etc.) - Think “Second Form of Identification”
  - Various restrictions on Cookie access (ex. HttpOnly)



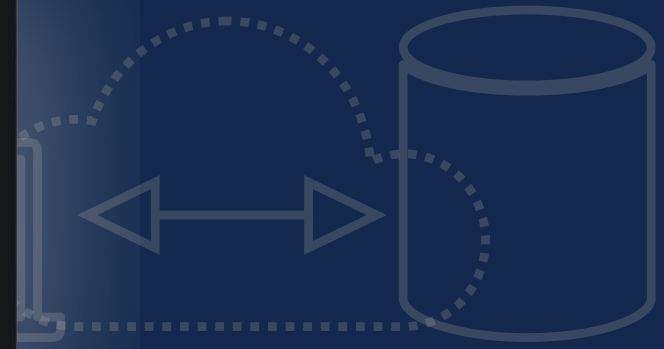
Client Side State



localStorage

- A JavaScript API that allows you to save information with larger capacity persistently in the browser
  - `localStorage.setItem('key', 'value')`
  - `localStorage.getItem('key')`
  - `localStorage.removeItem('key');`
  - `localStorage.clear();`

```
localStorage.setItem('secret', 'ohnoes');
localStorage.setItem('secret2','ohnoz');
localStorage.getItem('secret'); // 'ohnoes'
localStorage.getItem('moar');   // null
localStorage.removeItem('secret2');
localStorage.clear();   // all items removed
```

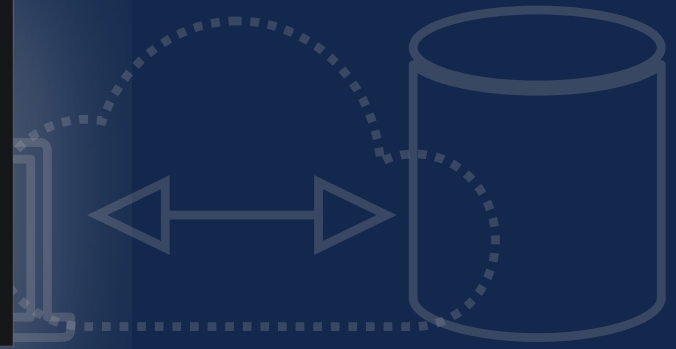


sessionStorage

- A JavaScript API that allows you to save information with for a “session” of a window or tab

  - `sessionStorage.setItem('key', 'value')`
  - `sessionStorage.getItem('key')`
  - `sessionStorage.removeItem('key');`
  - `sessionStorage.clear();`

```
sessionStorage.setItem('secret', 'ohnoes');
sessionStorage.setItem('secret2','ohnoz');
sessionStorage.getItem('secret'); // 'ohnoes'
sessionStorage.getItem('moar');   // null
sessionStorage.removeItem('secret2');
sessionStorage.clear();   // all items removed
```



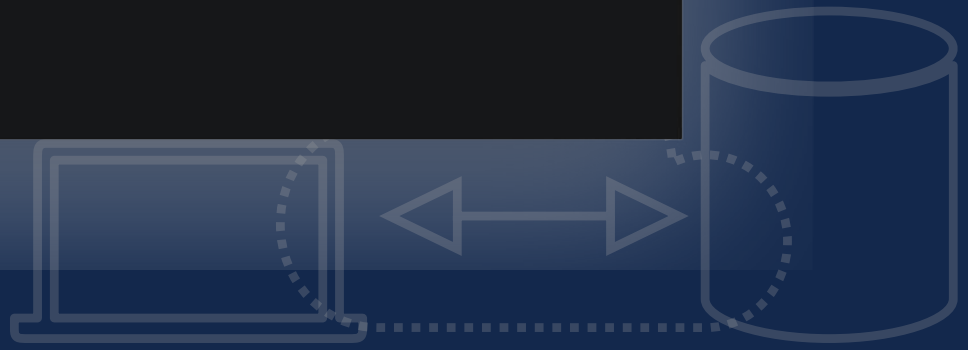
Storage Dump

- You can also loop over either storage object and get the keys in can you want to dump the data without knowing the keys

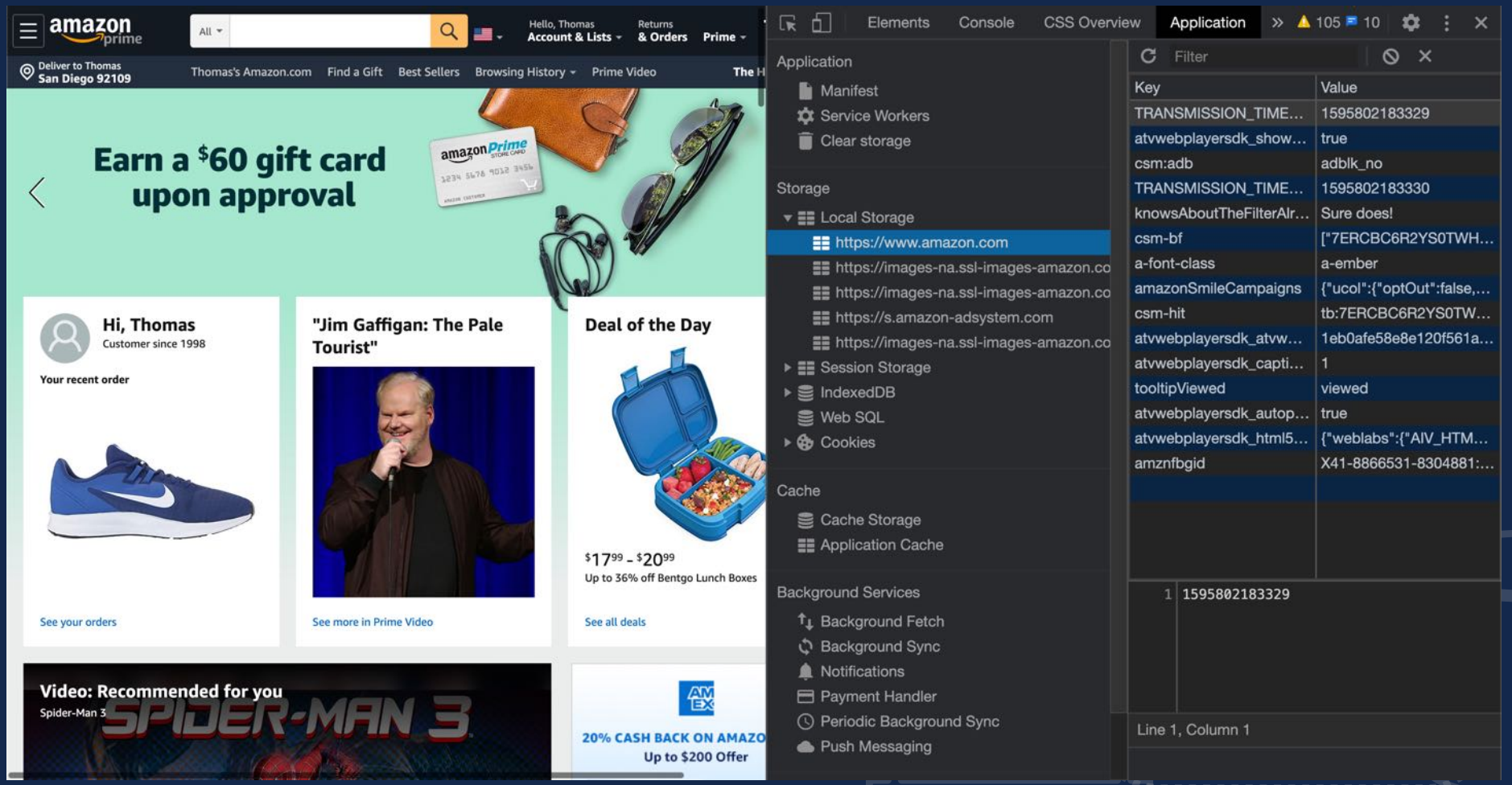
```
let result = {};

for (let i = 0; i < localStorage.length; i++) {
    result[localStorage.key(i)] = localStorage.getItem(localStorage.key(i));
}

console.dir(result);
```



Storage Availability

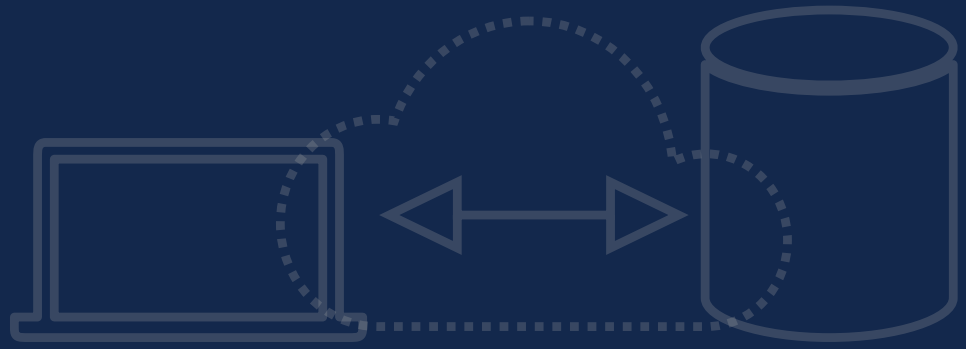


Storage Differences and Warnings

- Some might view that `localStorage` corresponds to a persistent cookie and `sessionStorage` to a session cookie.
  - Difference
    - More storage available
    - Session cookies are longer lived than `sessionStorage` because they relate to the domain and not the window
    - Cookies may be limited to script access, no current way to limit script access (includes 3rd parties!)

JS storage Access Restrictions

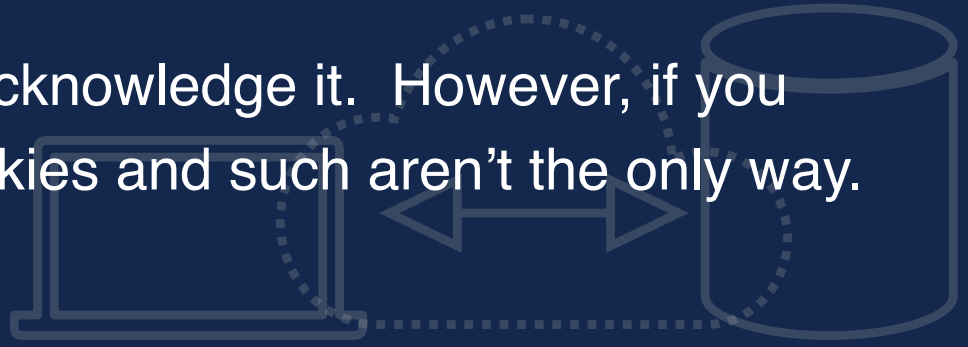
- It is limited to the particular domain it is issued with and that includes the protocol so http and https would be different
- However, beyond this it is easily accessed by ANY script running in the domain



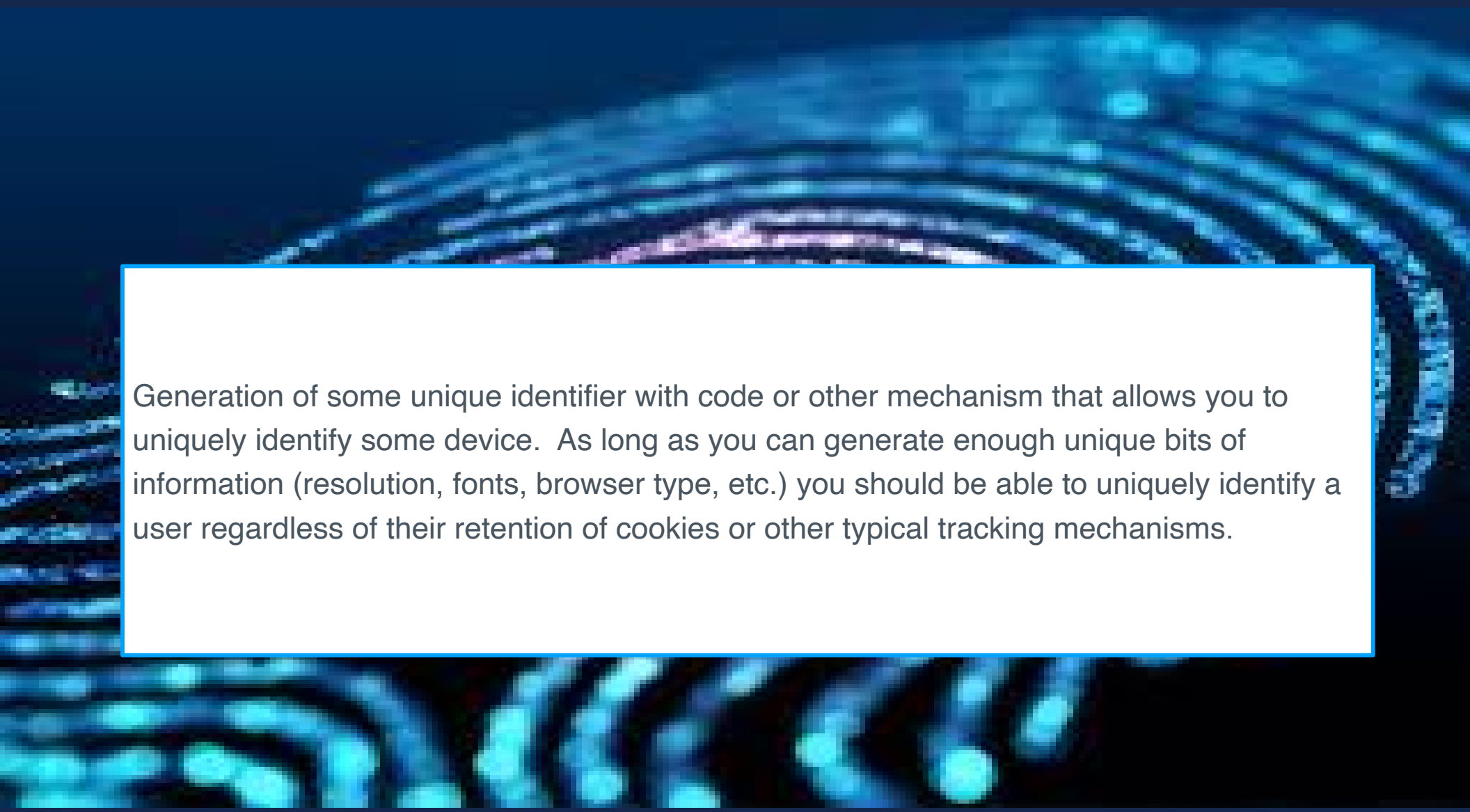
Some other important ideas

- Reassociating cookies
  - Why your blank Vons club card doesn't save you as much as you think
- Using persistence, browser specific or more standard HTML5 persistence – we can abstract this - careful here
- Tricks using if-modified-science values for Web bugs
- Tracking with JavaScript, fingerprinting and simple or esoteric ideas like frames or window.name

- TL;DR tracking is real. You should acknowledge it. However, if you are going to worry, be consistent cookies and such aren't the only way.

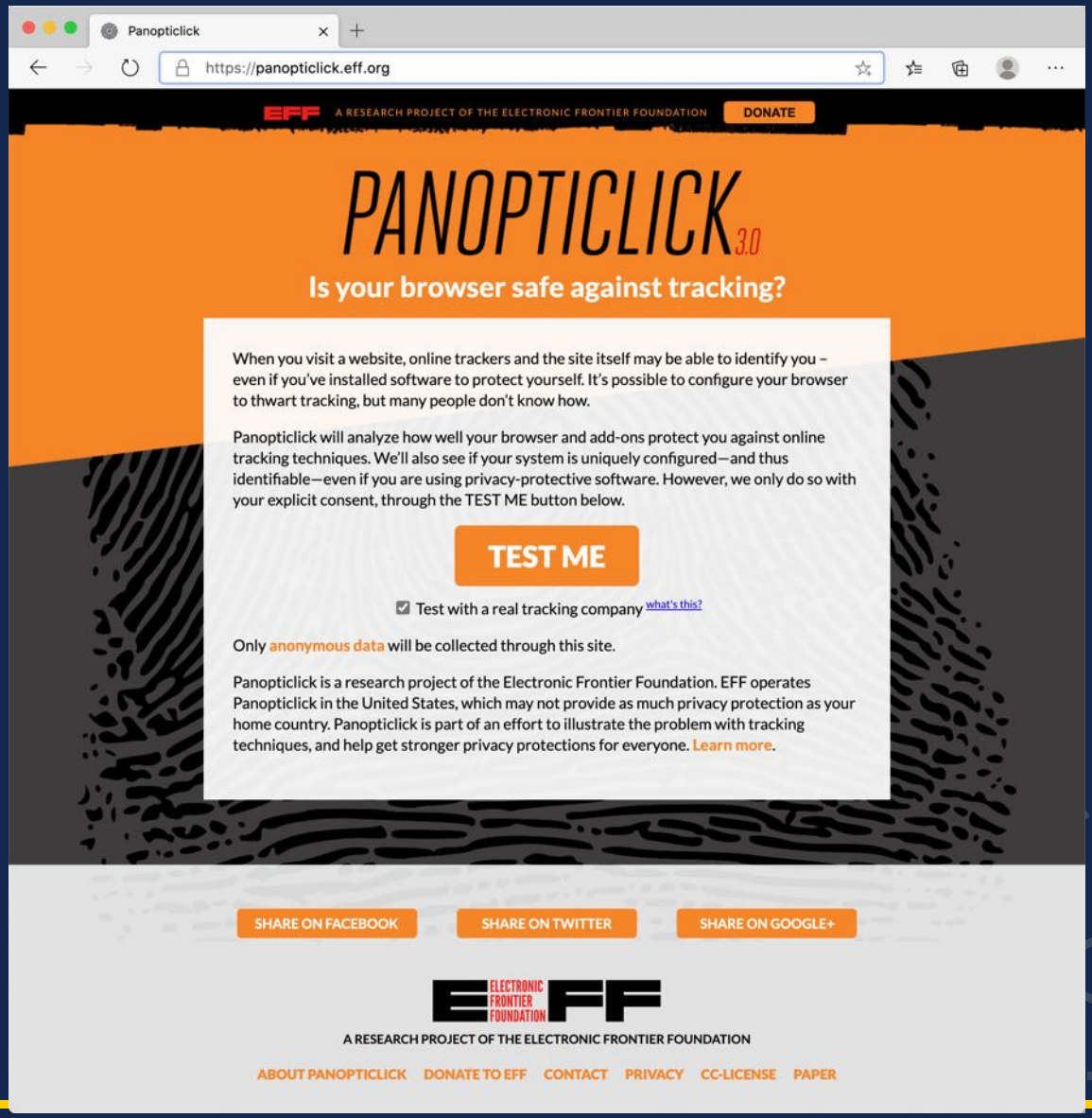


Fingerprinting



Generation of some unique identifier with code or other mechanism that allows you to uniquely identify some device. As long as you can generate enough unique bits of information (resolution, fonts, browser type, etc.) you should be able to uniquely identify a user regardless of their retention of cookies or other typical tracking mechanisms.

Fingerprinting Demo



Remember a face and telling the difference is this nefarious?

ME

NOPE - NOT ME

Identification isn't the problem

- In our real world we are ok with our uniqueness and being recognized is only a problem when we don't want to be or we assume we are "private"

- False Assumption - Internet Privacy
- IRL Difference - Perfect Memory
- Trust Bust - Retention and abuse of the identification
  - 3rd party identification happening without knowledge
  - Permeance of the identification
    - "Right to be forgotten" & Benefits of Imperfect Memory

Summary

- Solving HTTP statelessness is required for analytical tracking as well as common web development tasks
- Following the “Law of Three” we can “solve” state via
  - URL line - query string sessionid
  - Message body - hidden form field “postback”
  - Headers - the most common approach with tokens
- JavaScript based solutions have their place as well
- We need to be very aware of the intersection between sessionization and security/privacy. We will likely see this a few more times in the course.