Access Control and OS Security

CS 161: Computer Security

Prof. Anthony D. Joseph

January 29, 2014

Access Control

- Some resources (files, web pages, …) are sensitive.
- How do we limit who can access them?
- This is called the *access control* problem

Access Control Fundamentals

- *Subject* = a user, process, …
  (someone who is accessing resources)
- *Object* = a file, device, web page, …
  (a resource that can be accessed)
- *Policy* = the restrictions we'll enforce

- *access*(*S*, *O*) = true
  if subject *S* is allowed to access object *O*

Example

- *access*(Alice, Alice’s wall) = true
  access(Alice, Bob’s wall) = true
  access(Alice, Charlie’s wall) = false

- *access*(daw, /home/cs161/gradebook) = true
  access(Alice, /home/cs161/gradebook) = false

Access Control Matrix

- *access*(*S*, *O*) = true
  if subject *S* is allowed to access object *O*

	Alice’s wall	Bob’s wall	Charlie’s wall	…
Alice	true	true	false	
Bob	false	true	false	
…				

Permissions

- We can have finer-grained permissions, e.g., read, write, execute.

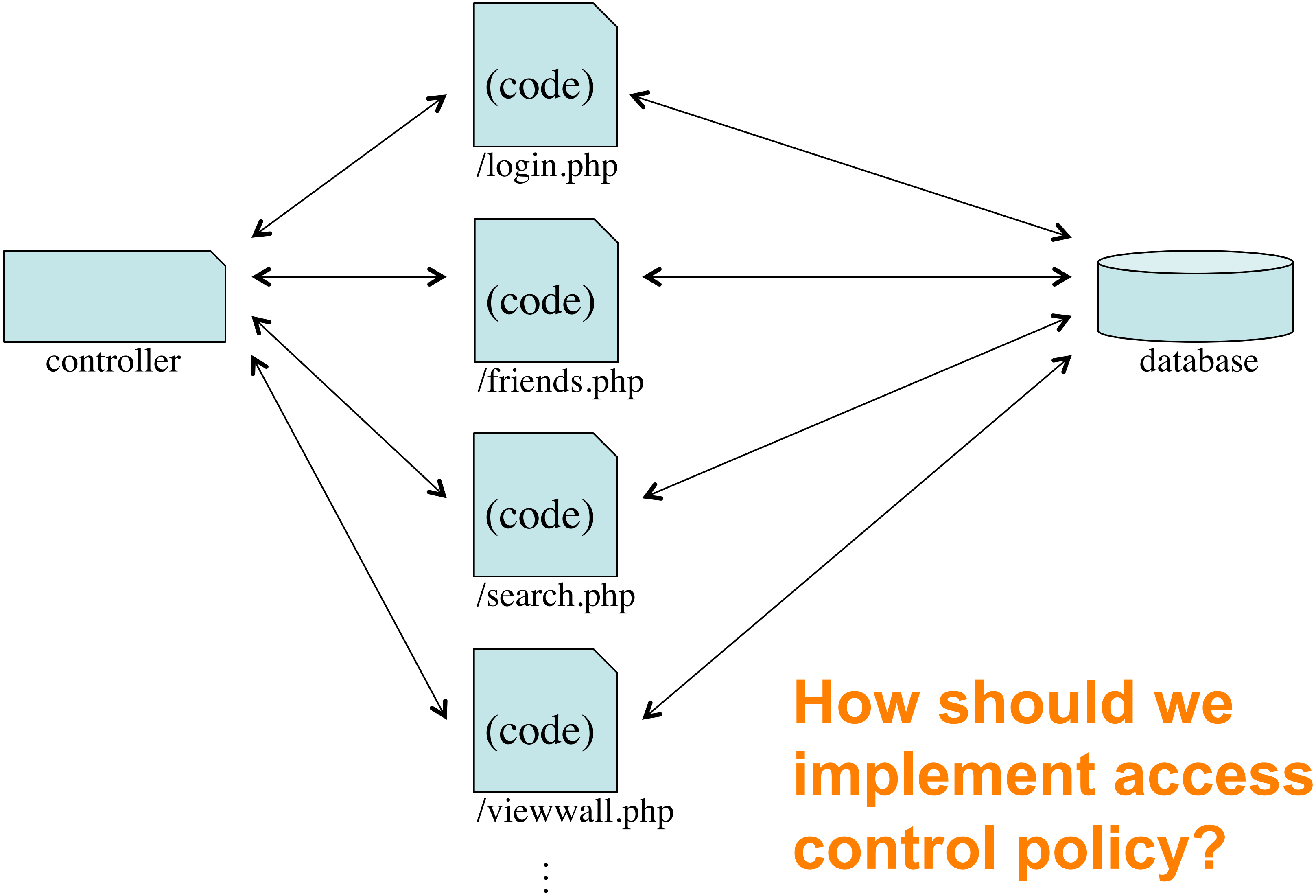
- *access*(daw, /cs161/grades/alice) = {read, write}
  access(alice, /cs161/grades/alice) = {read}
  access(bob, /cs161/grades/alice) = {}

	/cs161/grades/alice
daw	read, write
alice	read
bob	-

Web security

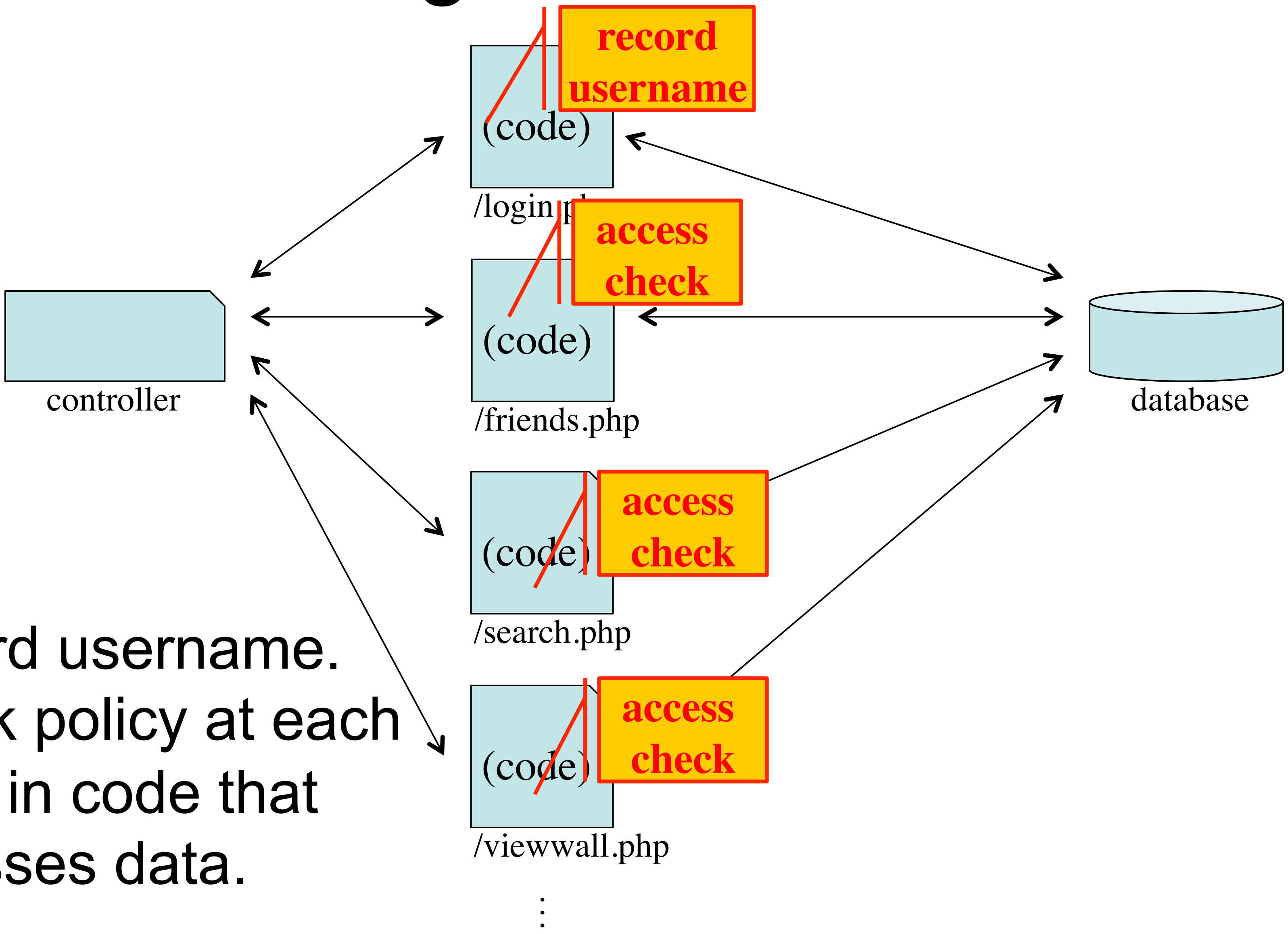
- Let’s talk about how this applies to web security…

Structure of a web application



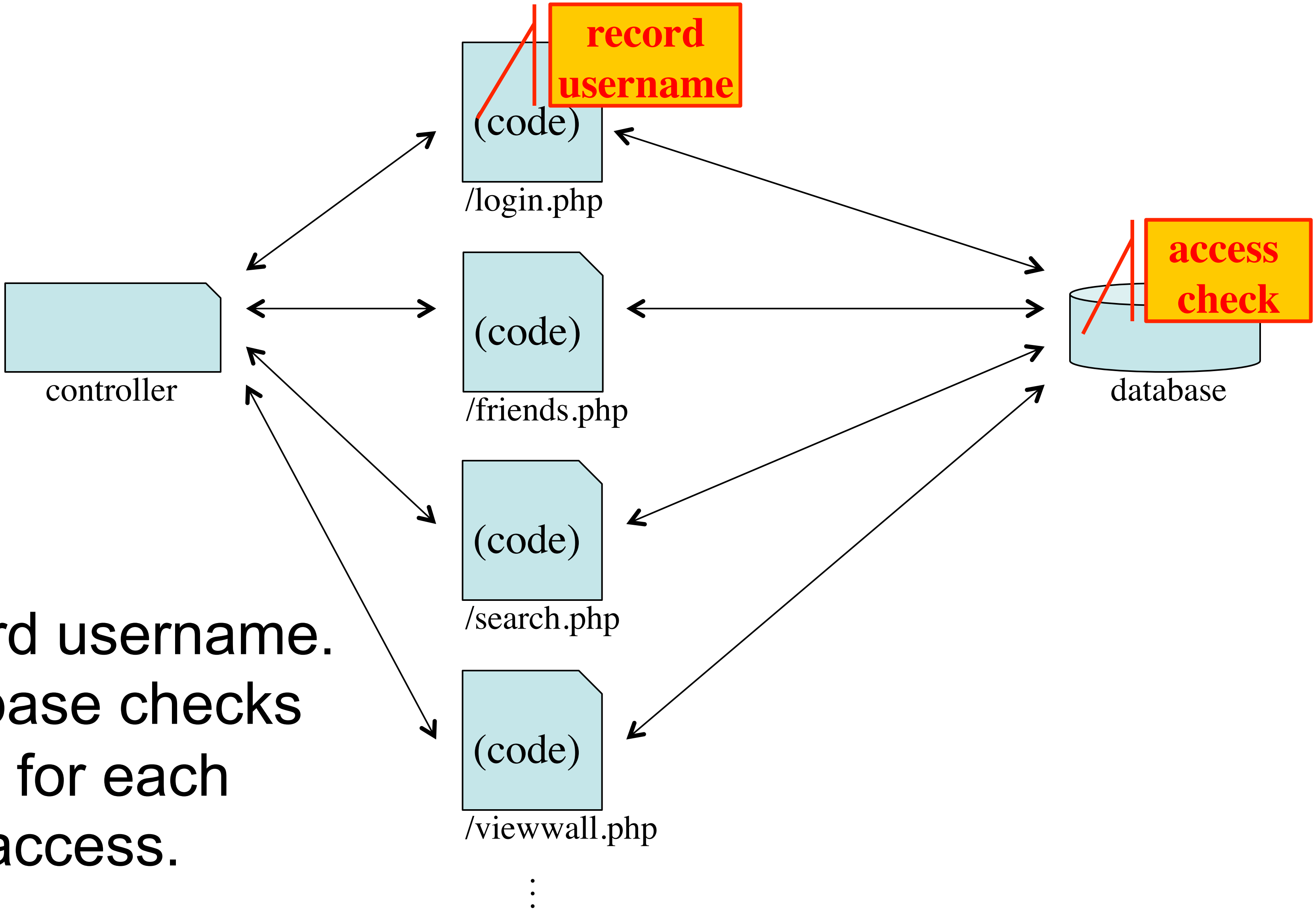
How should we implement access control policy?

Option 1: Integrated Access Control



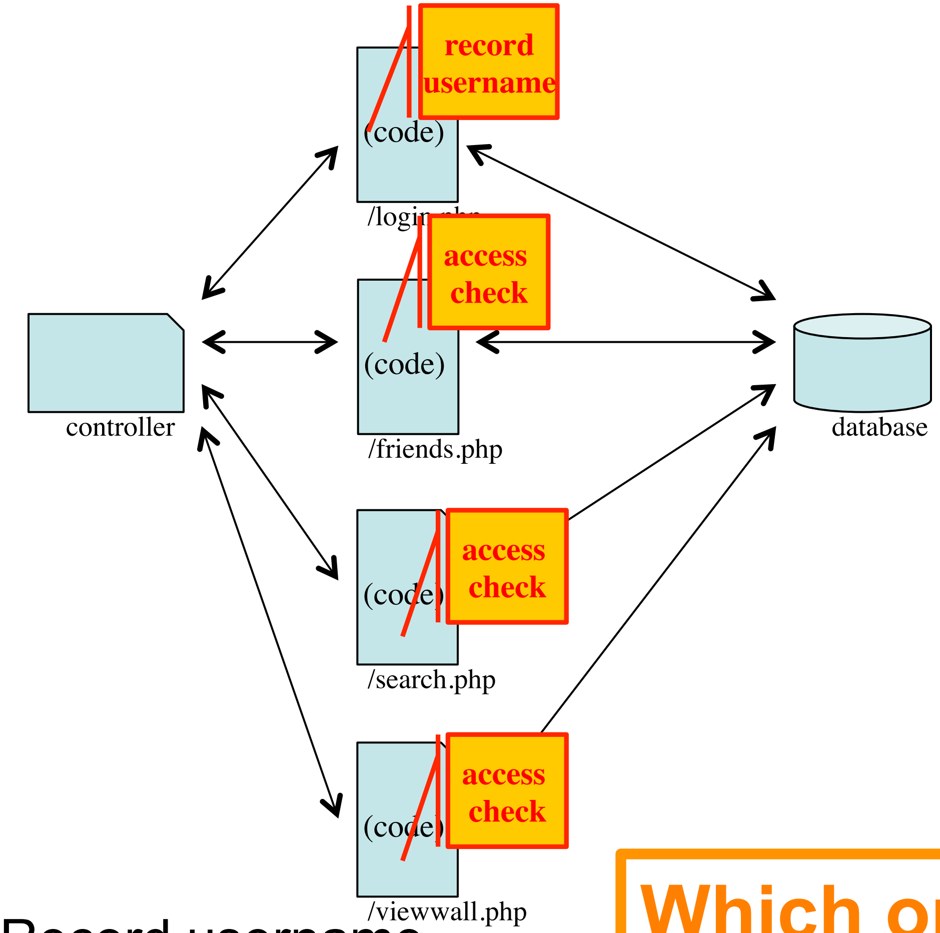
Record username. Check policy at each place in code that accesses data.

Option 2: Centralized Enforcement



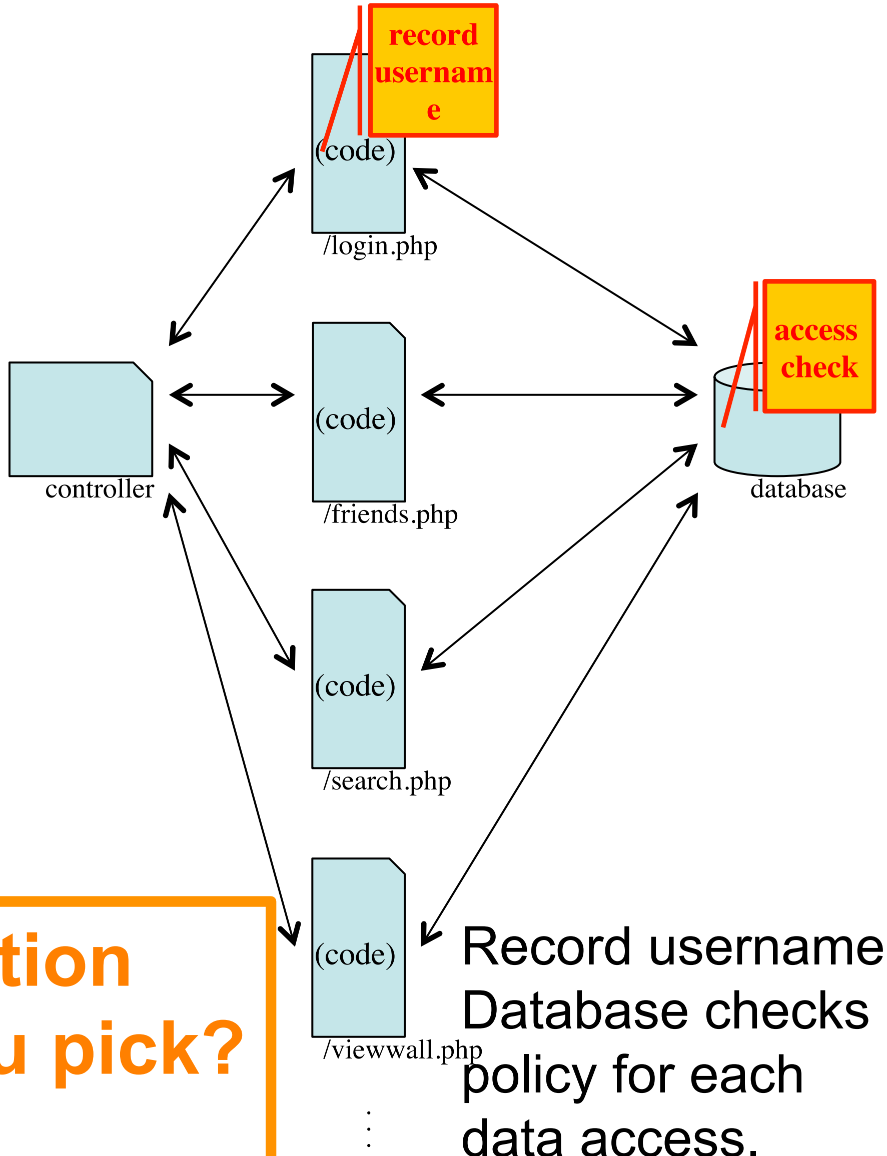
Record username.
Database checks policy for each data access.

Option 1: Integrated Access Control



Record username. Check policy at each place in code that accesses data.

Option 2: Centralized Enforcement



Record username. Database checks policy for each data access.

Which option would you pick? Discuss.

Analysis

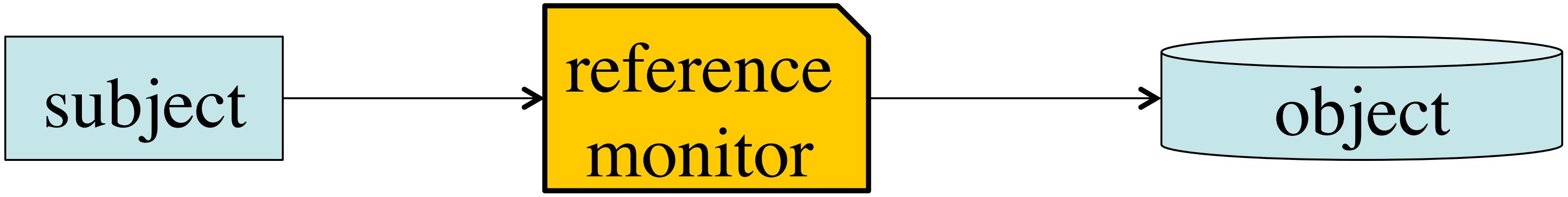
- Centralized enforcement might be less prone to error
  - All accesses are vectored through a central chokepoint, which checks access
  - If you have to add checks to each piece of code that accesses data, it’s easy to forget a check (and app will work fine in normal usage, until someone tries to access something they shouldn’t)
- Integrated checks are occasionally more flexible

Complete mediation

- The principle: complete mediation
- Ensure that all access to data is mediated by something that checks access control policy.
  - In other words: the access checks can’t be bypassed

Reference monitor

- A reference monitor is responsible for mediating all access to data



- Subject cannot access data directly; operations must go through the reference monitor, which checks whether they're OK

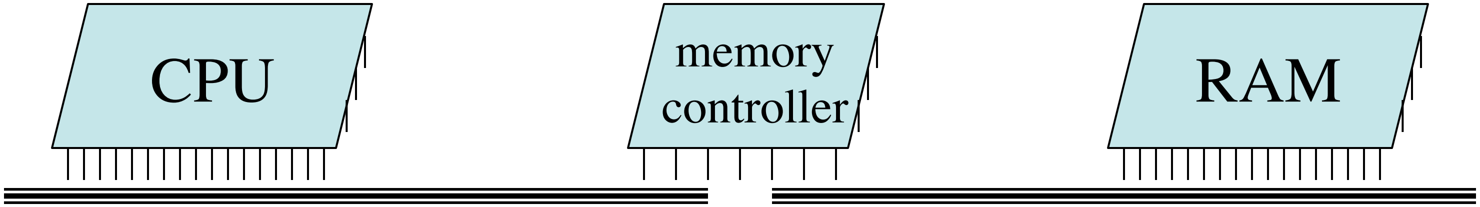
Criteria for a reference monitor

Ideally, a reference monitor should be:

- Unbypassable: all accesses go through the reference monitor
- Tamper-resistant: attacker cannot subvert or take control of the reference monitor (e.g., no code injection)
- Verifiable: reference monitor should be simple enough that it’s unlikely to have bugs

Example: OS memory protection

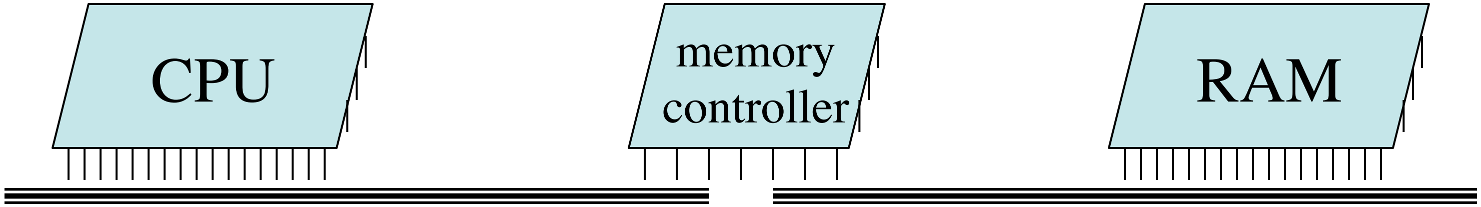
- All memory accesses are mediated by memory controller, which enforces limits on what memory each process can access



Unbypassable? ✔

Example: OS memory protection

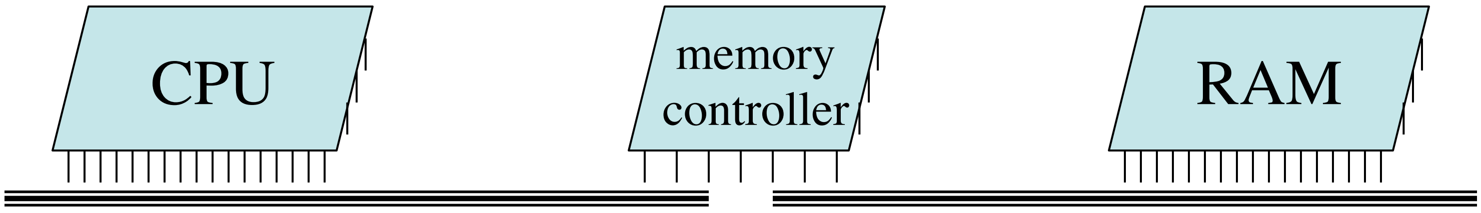
- All memory accesses are mediated by memory controller, which enforces limits on what memory each process can access



Tamper-resistant? ✔

Example: OS memory protection

- All memory accesses are mediated by memory controller, which enforces limits on what memory each process can access



Verifiable? ✔

TCB

- More broadly, the trusted computing base (TCB) is the subset of the system that has to be correct, for some security goal to be achieved
  - Example: the TCB for enforcing file access permissions includes the OS kernel and filesystem drivers
- Ideally, TCBs should be unbypassable, tamper-resistant, and verifiable

Privilege separation

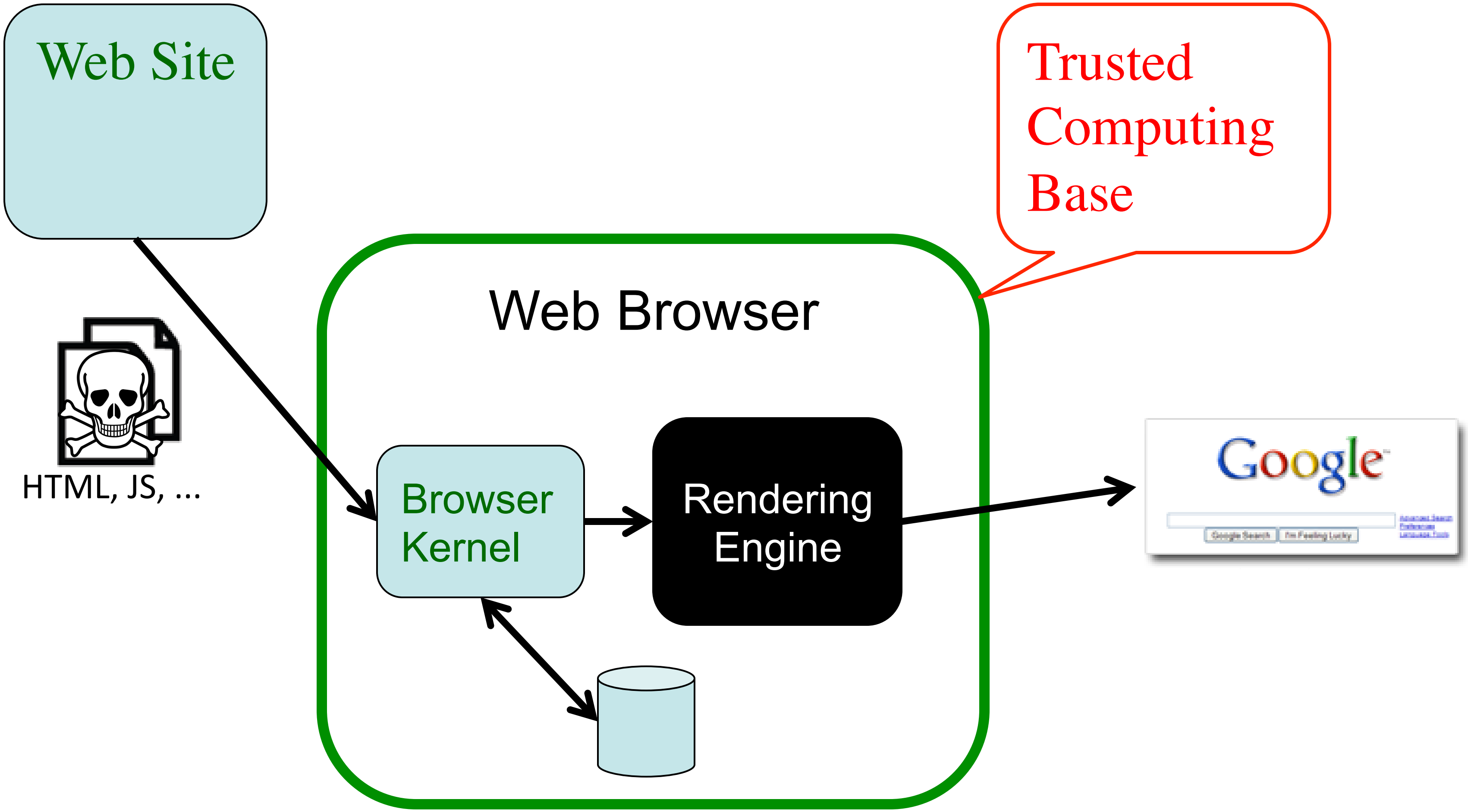
- How can we use these ideas to improve the security of software, so security bugs are less likely to be catastrophic?

Privilege separation

- How can we use these ideas to improve the security of software, so security bugs are less likely to be catastrophic?

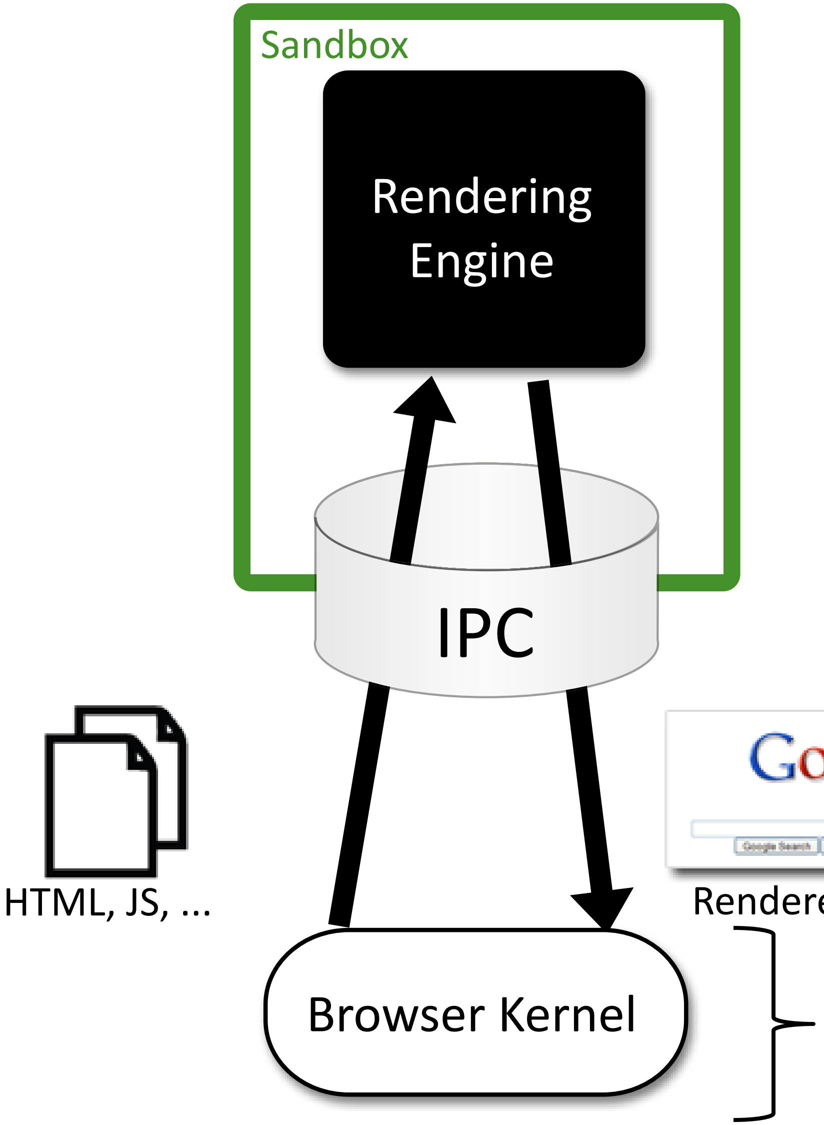
- Answer: privilege separation. Architect the software so it has a separate, small TCB.
  - Then any bugs outside the TCB will not be catastrophic

Web browser



"Drive-by malware": malicious web page exploits a browser bug to read/write local files or infect them with a virus

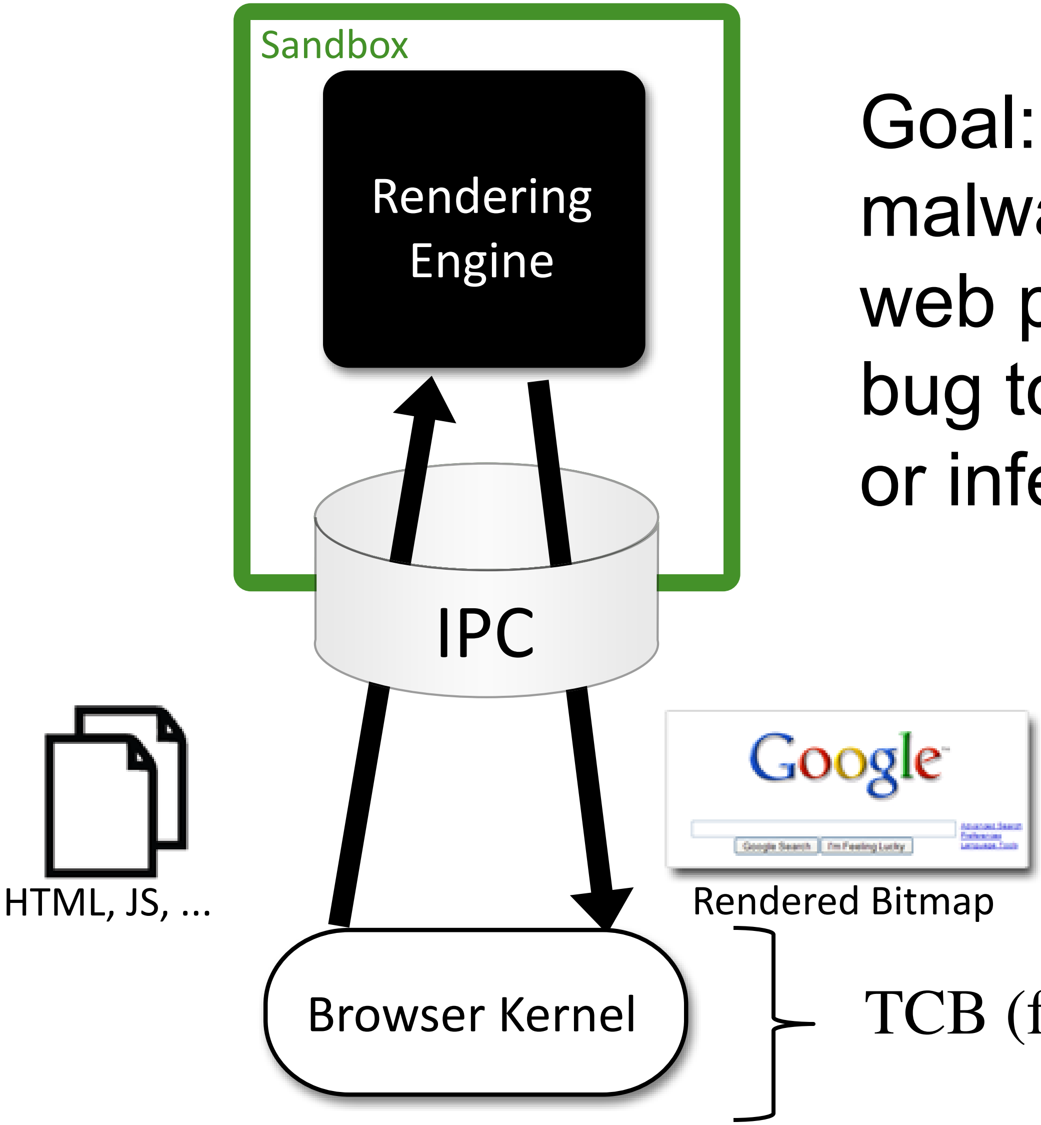
The Chrome browser



Goal: prevent “drive-by malware”, where a malicious web page exploits a browser bug to read/write local files or infect them with a virus

TCB (for this property)

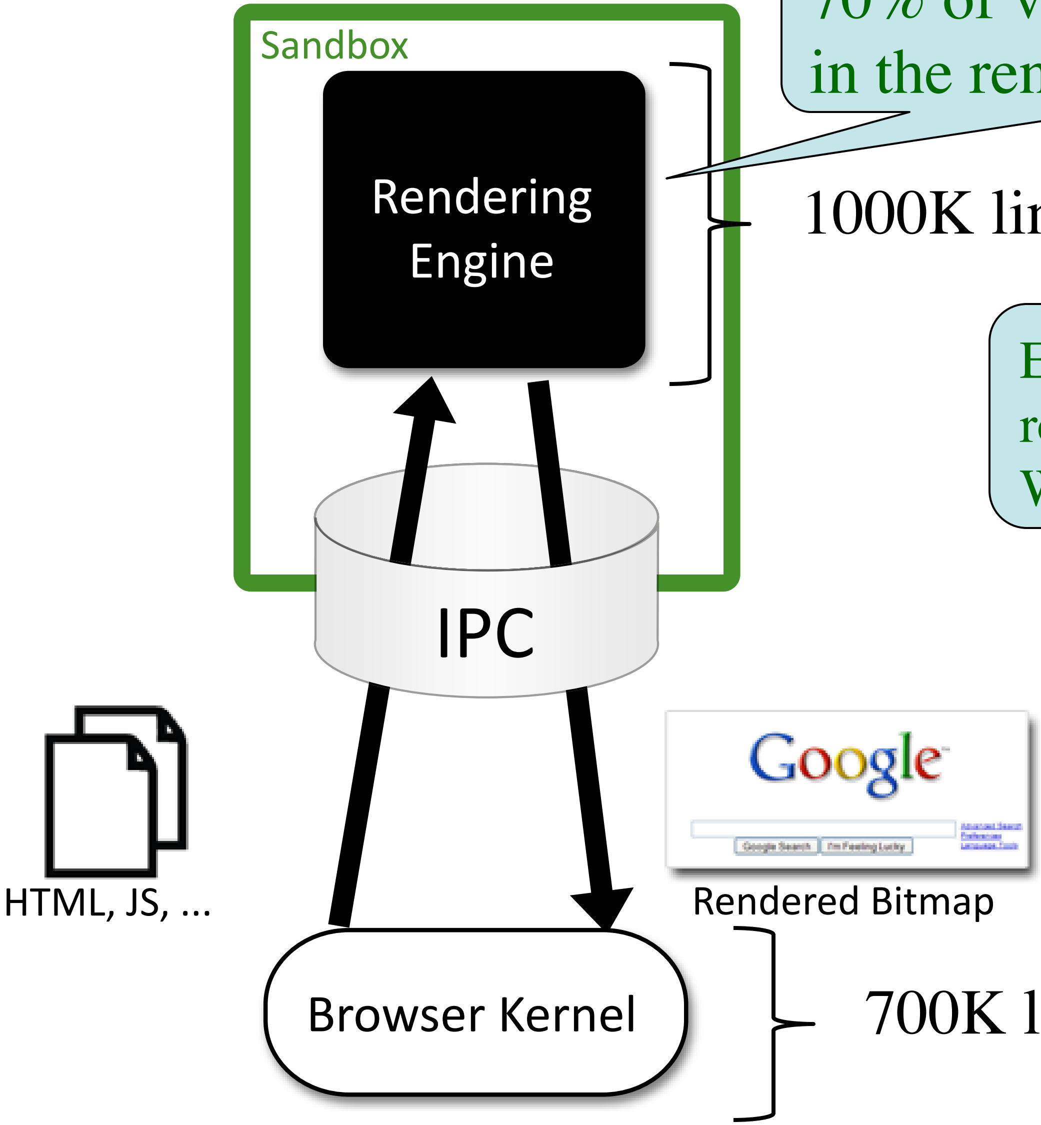
The Chrome browser



TCB (for this property)

Goal: prevent “drive-by malware”, where a malicious web page exploits a browser bug to read/write local files or infect them with a virus

The Chrome browser



70% of vulnerabilities are in the rendering engine.

1000K lines of code

Example: PNG, WMF, GDI+ rendering vulnerabilities in Windows OS

700K lines of code

Summary

- Access control is a key part of security.
- Privilege separation makes systems more robust: it helps reduce the impact of security bugs in your code.
- Architect your system to make the TCB unbypassable, tamper-resistant, and verifiable (small).

Coming Up …

- Friday guest lecture:
  Malware
- Homework 0 due Friday
- C review session, Saturday, February 1st, 2-4pm, 306 Soda