API+Node	1. Provide services to another system
Reactor	2. Respond to multiple overlapping requests
Express	3. Allow and restrict user access
OWASP	4. Protect my business from harm (exploits)
OWASP	5. Protect user data from being harvested
State + DB	6. Make changes persistent
API calls	7. Consume services from another system
Architecture	8. Modularize my application
Next term…	9. Display information from a source
	10. Synch information shown on different views
	11. Maximize responsiveness
	12. Adapt to different devices and screen sizes

Last week

1. Inside the box – Node.js and ES2015
    1. Promises (and Async/Await)
2. State and statelessness
3. Information Security

```
// Normal callback usage => PYRAMID OF DOOOOOOOOM
asyncOperation(function(data){
    // Do some processing with `data`
    anotherAsync(function(data2){
        // Some more processing with `data2`
        yetAnotherAsync(function(){
            // Yay we're finished!
        });
    });
});


// Let's look at using promises
asyncOperation()
.then(function(data){
    // Do some processing with `data`
    return anotherAsync();
})
.then(function(data2){
    // Some more processing with `data2`
    return yetAnotherAsync();
})
.then(function(){
    // Yay we're finished!
});
```

http://www.telerik.com/blogs/what-is-the-point-of-promises

Injection

Any time an application uses **an interpreter of any type** there is a danger of introducing an injection vulnerability.

When a web application passes information from an **HTTP request** through as part of an external request, it must be carefully scrubbed

- why?

SQL injection is a particularly widespread and dangerous form of injection…

Authentication etc.

Definitions

- Authentication: establish claimed identity
- Authorisation: establish permission to act
- Authentication *precedes* authorisation

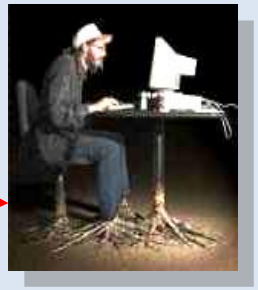
Why authenticate?

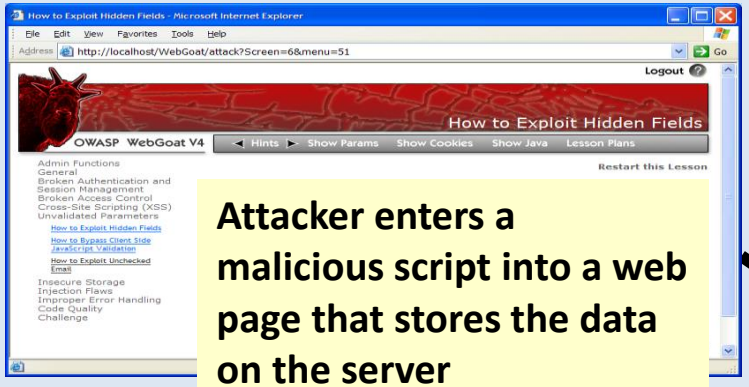
How can we authenticate?

Three factors…

Cross-Site Scripting (XSS)

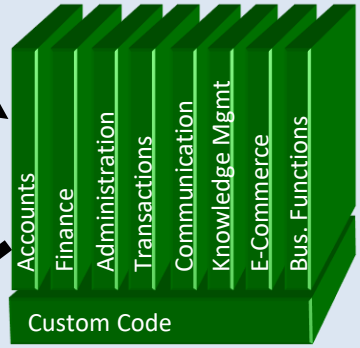
1 **Attacker sets the trap – update my profile**



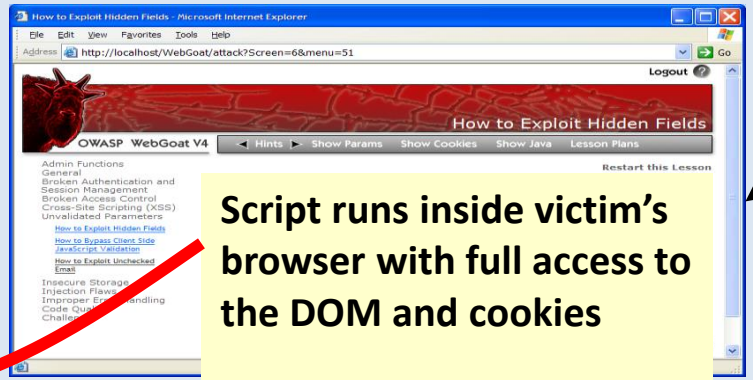


Attacker enters a malicious script into a web page that stores the data on the server

Application with stored XSS vulnerability



2 **Victim views page – sees attacker profile**



Script runs inside victim's browser with full access to the DOM and cookies

3 **Script silently sends attacker Victim's session cookie**

https://owasp.org

Other approaches (not prioritised)

Hash username and password

Require users to change their passwords regularly

Use multi-factor authentication

- Username & password
- Code sent by phone

Salt the username and password

- Add additional elements to the ID information

Use HTTPS (HTTP + TLS)

This week

How do I...

1. Consume services from another system?
2. Make things persistent?
3. Modularize my application?

Consuming services

http/https modules

Callback/event based

Lots of alternatives, e.g.:

- request
- request-promise
- request-promise-native


Pet

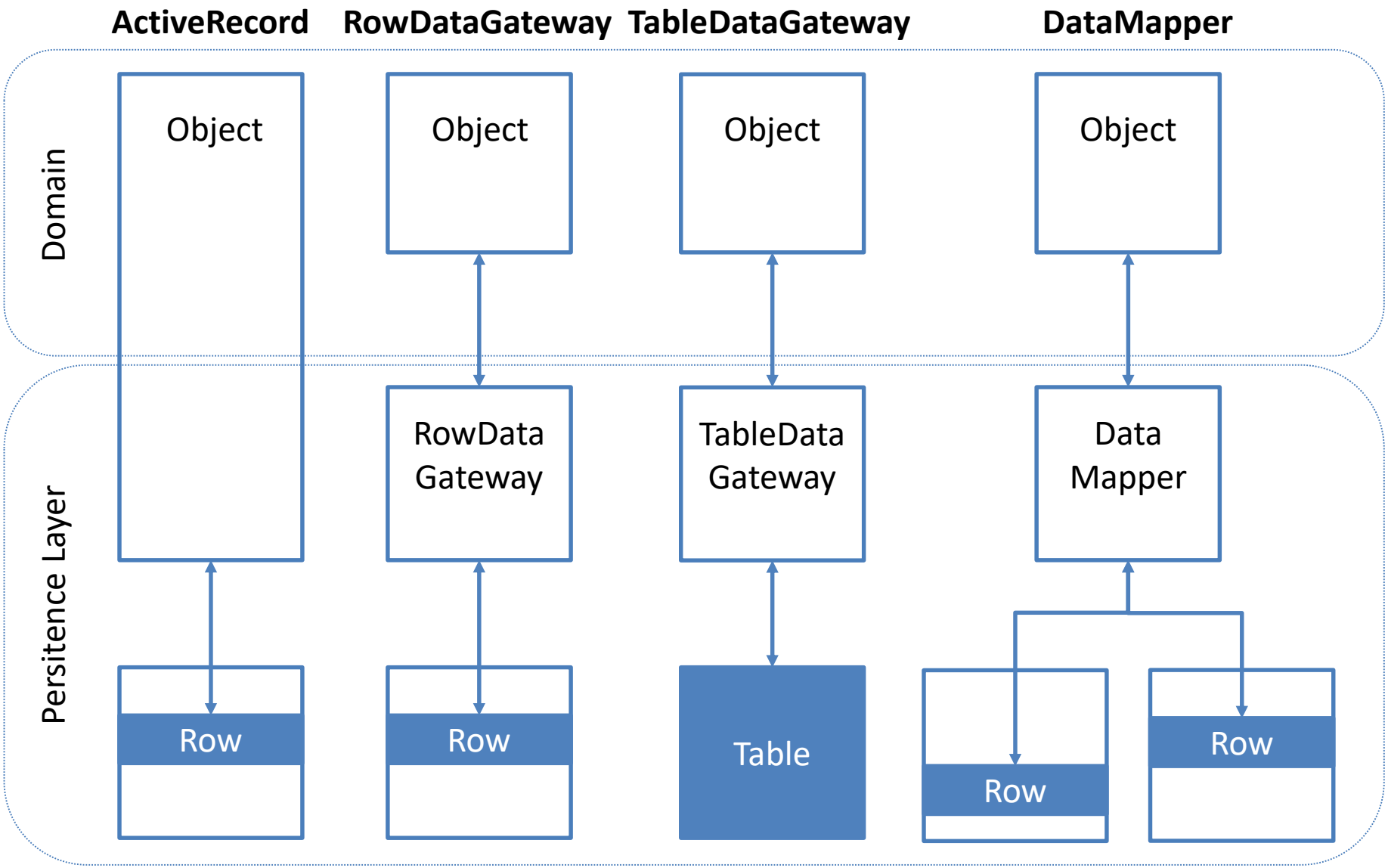
PERSISTENCE (I)

Long lived state

Data access patterns (Fowler)

- ActiveRead
- DataMapper
- TableDataListener
- TableModule

- Other patterns
  - DataAccessObject (J2EE)


ActiveRecord
RowDataGateway
TableDataGateway
DataMapper
Domain
Persitence Layer
Object
Object
Object
Object
RowData Gateway
TableData Gateway
Data Mapper
Row
Row
Table
Row
Row

DISTRIBUTION/ARCHITECTURE

https://www.flickr.com/photos/bk1bennett/6334043117

What's ACID?

notes from Martin Kleppmann's *amazing* "Designing Data-Intensive Applications" book

ACID is about safety guarantees for database transactions.

Atomicity

not about concurrent writes (that's "isolation")

Consistency

super overloaded term. This sense of "consistency" is actually an application property not a DB property.

not linearizability

not as in "eventual consistency"

About preserving application invariants like "every sale gets an invoice".

Isolation

Isolation is about preventing race conditions like this.

Some isolation levels:

- serializability
- snapshot isolation
- read committed

Durability

Perfect durability doesn't exist.

Can involve:

- write-ahead log (usually)
- replication

https://drawings.jvns.ca/drawings/acid.svg

https://drawings.jvns.ca/drawings/acid.svg

Brewer's conjecture, or the CAP theorem

Consistency, Availability and Partition tolerance. Choose two.

Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News* 33, 2 (June 2002), 51-59. DOI=http://dx.doi.org/10.1145/564585.564601

~~*Consistency, Availability and Partition tolerance. Choose two.*~~

Partition => **A**vailability or **C**onsistency
Else (normal) => **L**atency or **C**onsistency

Possibility of Network Partitions => not(Availability and Consistency)

if you have a network that may drop messages, *then* you cannot have both availability and consistency, you must choose one

https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
http://blog.cloudera.com/blog/2010/04/cap-confusion-problems-with-partition-tolerance/
D. Abadi, “Consistency Tradeoffs in Modern Distributed Database System Design..”, 2012

Eventual consistency is a widely used term that can have many meanings. Here it is defined as the strongest property provided by all systems that claim to provide it: namely, writes to one data center will eventually appear at other data centers, and if all data centers have received the same set of writes, they will have the same values for all data.

Strong consistency (linearizability): a total order exists over all operations in the system (one definition…)

Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. 2014. Don't Settle for Eventual Consistency. *Queue* 12, 3, pages 30 (March 2014), 16 pages. DOI=http://dx.doi.org/10.1145/2602649.2610533

A proof that predates the CAP theorem by 14 years shows that it is impossible to guarantee low latency and provide strong consistency at the same time.

Lipton, R. J., Sandberg, J. S. 1988. PRAM: a scalable shared memory. Technical Report TR-180-88. Princeton University, Department of Computer Science.

Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. 2014. Don't Settle for Eventual Consistency. *Queue* 12, 3, pages 30 (March 2014), 16 pages. DOI=http://dx.doi.org/10.1145/2602649.2610533

ATM example

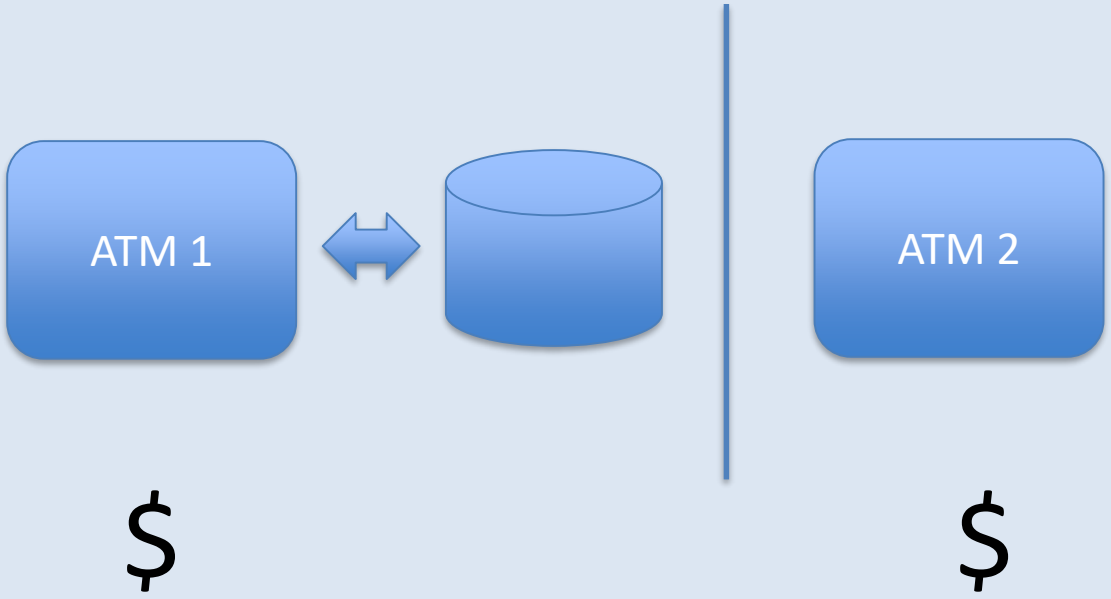
Availability or Consistency when Partitioned?

$ $

Brewer - https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

ATM example

Availability or Consistency when Partitioned?



Brewer - https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

ATM example

Availability or Consistency when Partitioned?

Availability! More money.

Operations:

- Deposit
- Withdrawal
- Check balance

Brewer - https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

ATM example

Availability or Consistency when Partitioned?

Availability! More money.

Operations:

- Deposit
- **Withdrawal – can change invariants (balance>0)**
- Check balance

Brewer - https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

PERSISTENCE (I)

Key/Value

Document

Graph

Time-series/Event

Key/Value

Document

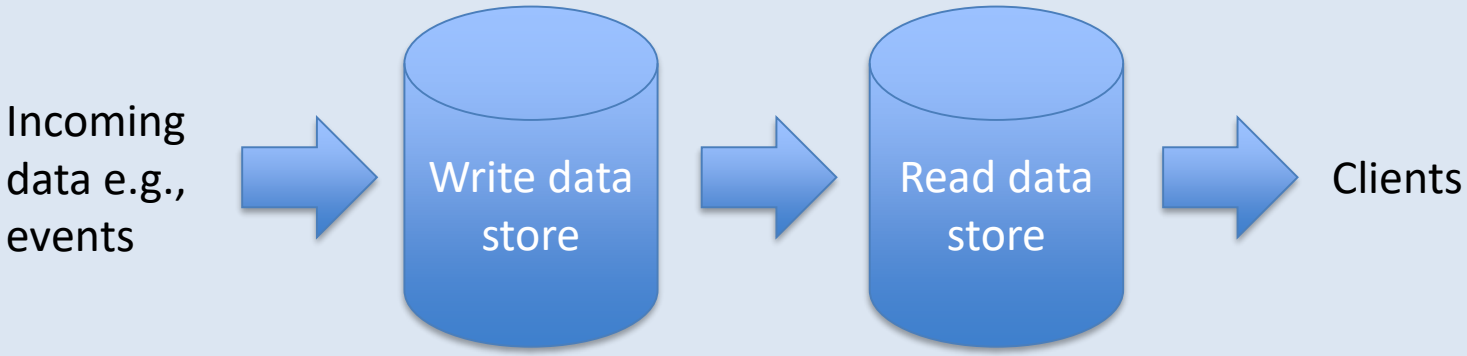
Graph

Time-series

CQRS Pattern and Event Sourcing

CQRS: Command and Query Responsibility Segregation

Often used in combination with EventSourcing



State timescales

Individual HTTP request (stateless)

Business transaction

Session

Preferences

Record state


Some cattle
https://www.flickr.com/photos/mbowler/61713384

Further reading

- Course wiki – Fundamentals of Distributed Computing (CAP etc)

Next week

- Monday – Assignment 1 due
- Thursday - Mid-semester test (1 hour)

Next term

- Front-end (client-side), with Vue.js
- Assignment 2 – client using API

ACID, CAP: not the same

ACID

Atomicity

Consistency (preserve invariants)

Isolation

Durability

CAP

Yes, a Good Thing

Consistency (looks like a single-copy)

Partition+Availability?

Availability+Consistency?