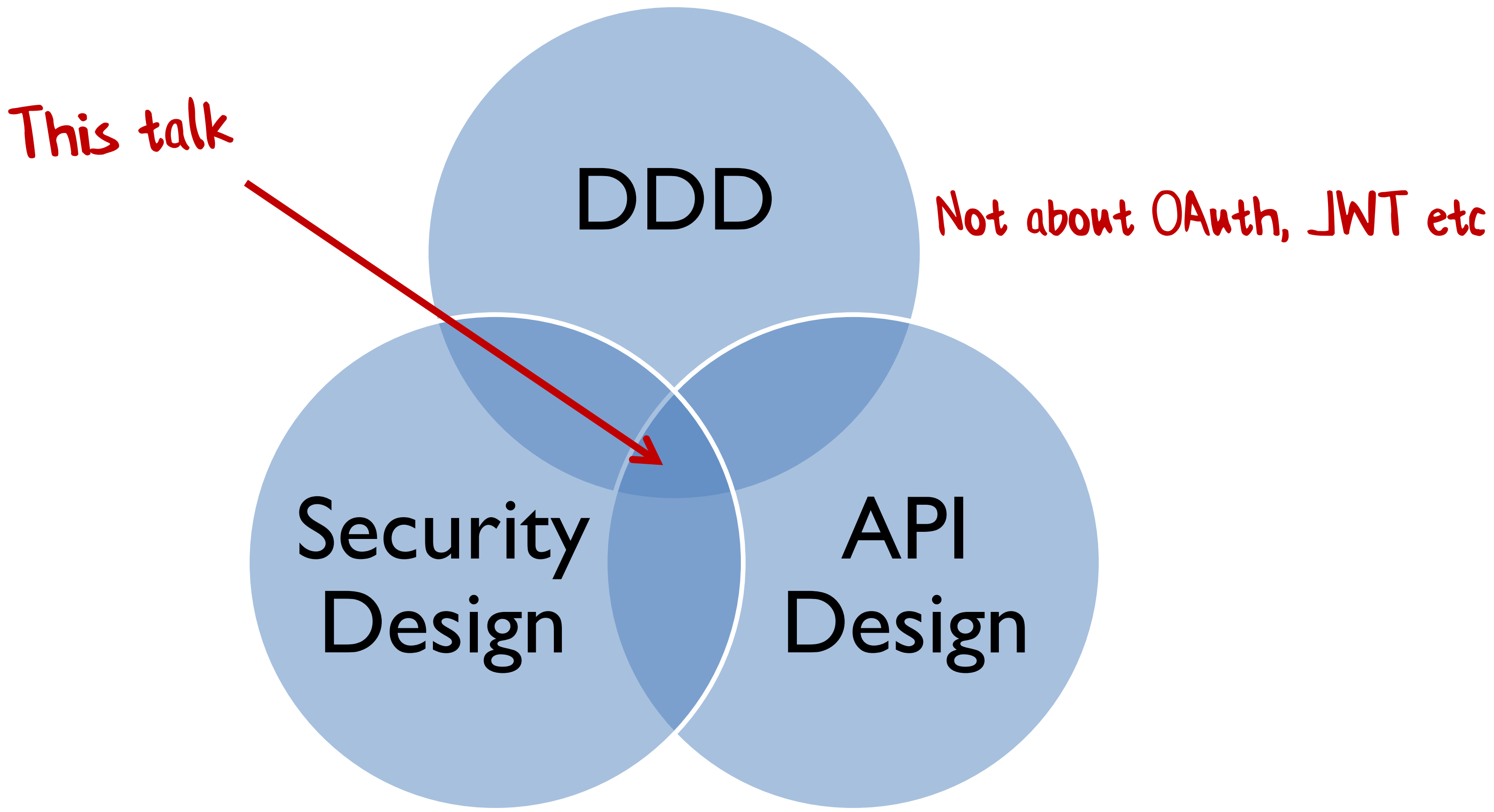
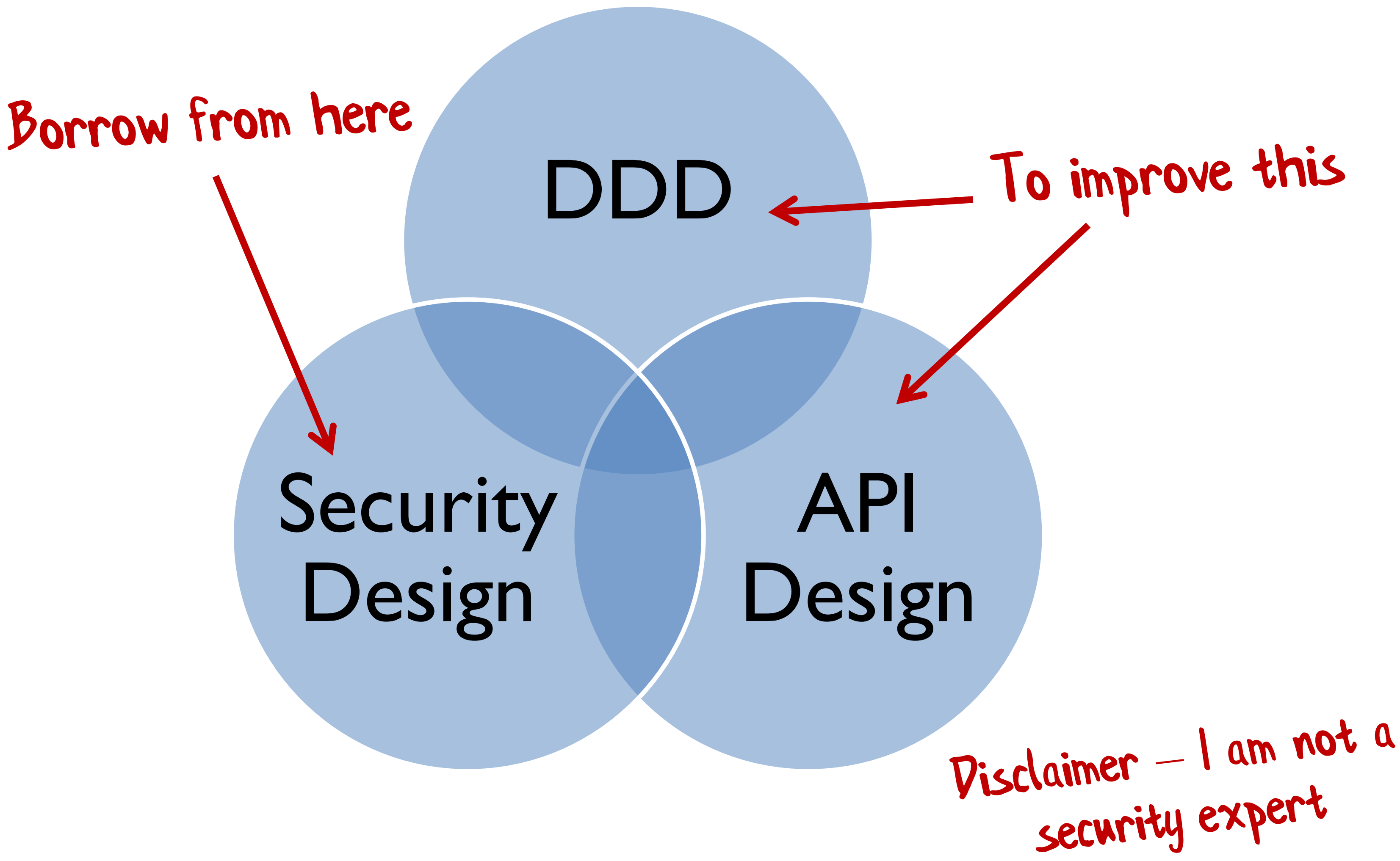
Designing with capabilities
(DDD Europe 2017)

@ScottWlaschin

fsharpforfunandprofit.com/cap

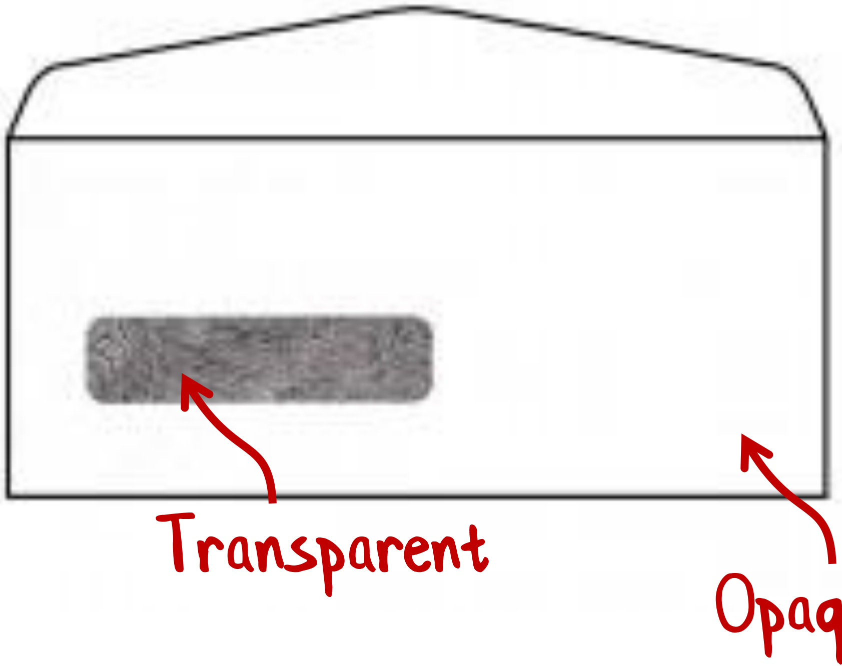

This talk
DDD
Not about OAuth, JWT etc
Security Design
API Design


Borrow from here
DDD
To improve this
Security Design
API Design
Disclaimer – I am not a security expert

Topics

- What does security have to do with design?
- Introducing capabilities
- API design with capabilities
- Design consequences of using capabilities
- Transforming capabilities for business rules
- Delegating authority using capabilities

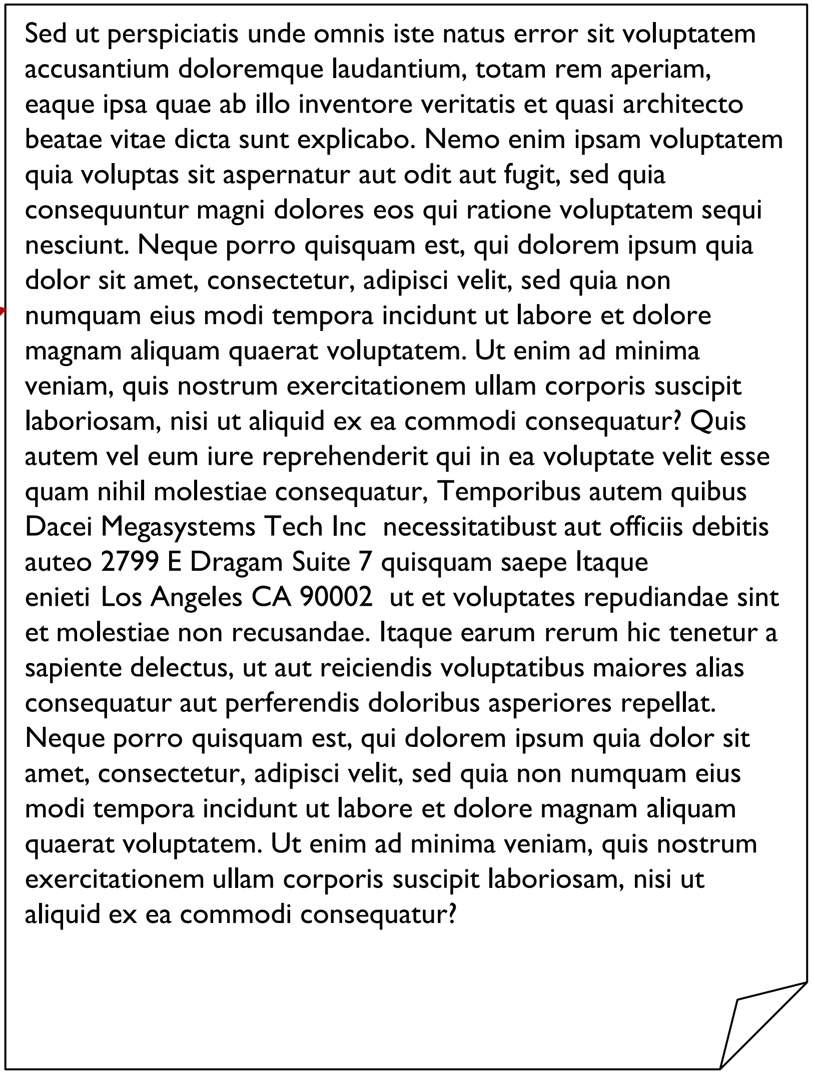
WHAT DOES SECURITY HAVE TO DO WITH DESIGN?



It's all about security, right?

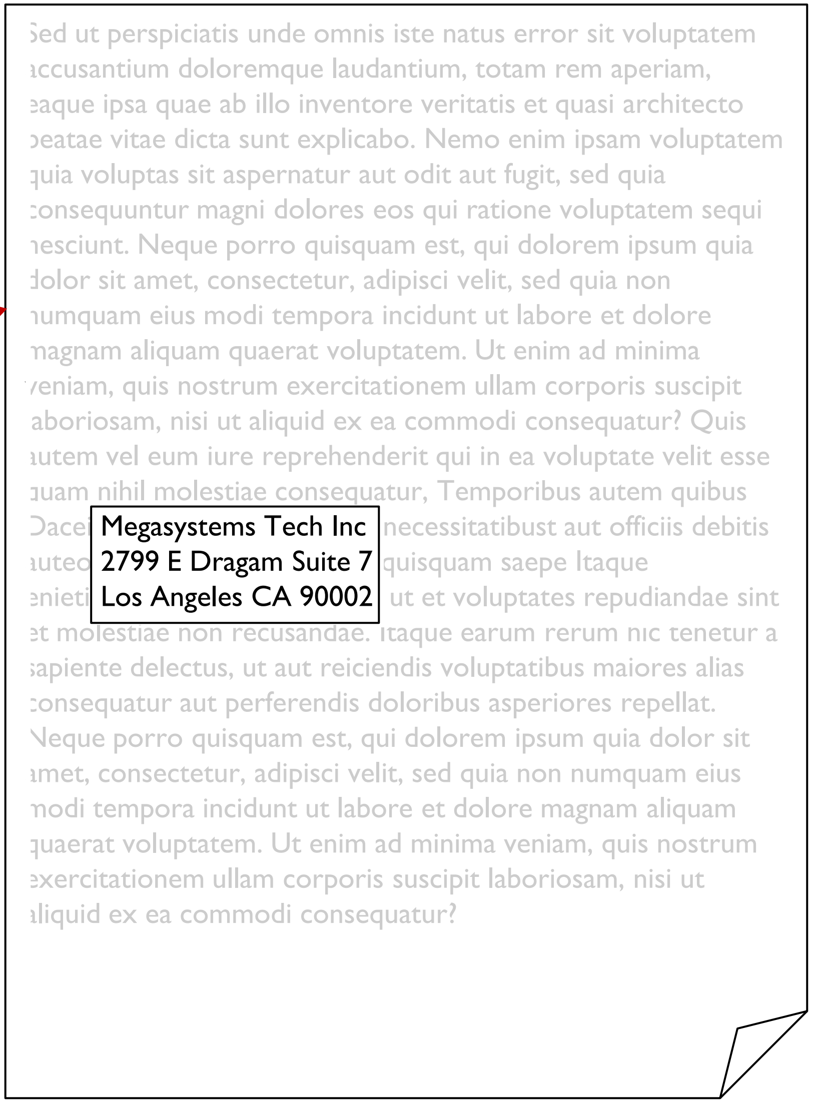
A counterexample

Please deliver this letter



Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, Temporibus autem quibus Dacei Megasystems Tech Inc necessitatibust aut officiis debitis auteo 2799 E Dragam Suite 7 quisquam saepe Itaque enieti Los Angeles CA 90002 ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut perferendis doloribus asperiores repellat. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur?

Please deliver this letter



ed ut perspiciatis unde omnis iste natus error sit voluptatem
ccusantium doloremque laudantium, totam rem aperiam,
aque ipsa quae ab illo inventore veritatis et quasi architecto
eatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem
uia voluptas sit aspernatur aut odit aut fugit, sed quia
onsequuntur magni dolores eos qui ratione voluptatem sequi
esciunt. Neque porro quisquam est, qui dolorem ipsum quia
olor sit amet, consectetur, adipisci velit, sed quia non
umquam eius modi tempora incidunt ut labore et dolore
agnam aliquam quaerat voluptatem. Ut enim ad minima
eniam, quis nostrum exercitationem ullam corporis suscipit
aboriosam, nisi ut aliquid ex ea commodi consequatur? Quis
utem vel eum iure reprehenderit qui in ea voluptate velit esse
uam nihil molestiae consequatur, Temporibus autem quibus
ace necessitatibust aut officiis debitis
uteo quisquam saepe Itaque
nieti ut et voluptates repudiandae sint
t molestiae non recusandae. Itaque earum rerum hic tenetur a
apiente delectus, ut aut reiciendis voluptatibus maiores alias
onsequatur aut perferendis doloribus asperiores repellat.
Neque porro quisquam est, qui dolorem ipsum quia dolor sit
met, consectetur, adipisci velit, sed quia non numquam eius
odi tempora incidunt ut labore et dolore magnam aliquam
uaerat voluptatem. Ut enim ad minima veniam, quis nostrum
xercitationem ullam corporis suscipit laboriosam, nisi ut
liquid ex ea commodi consequatur?

Megasystems Tech Inc
2799 E Dragam Suite 7
Los Angeles CA 90002

It's not just about security...

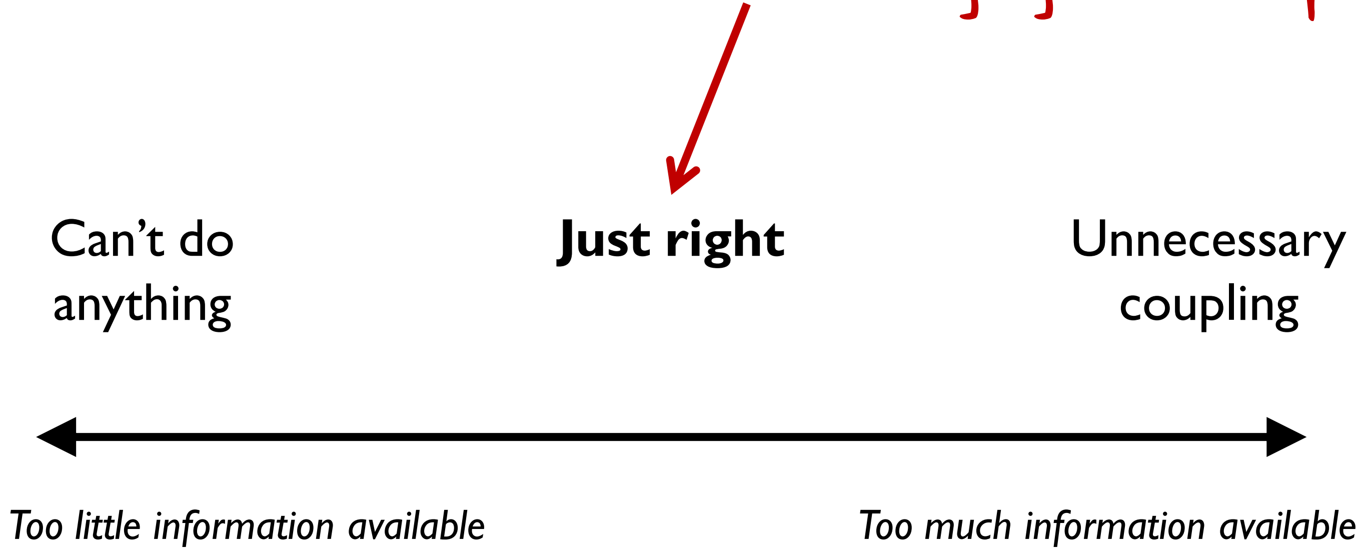
...hiding irrelevant information is good design!

David Parnas, 1971

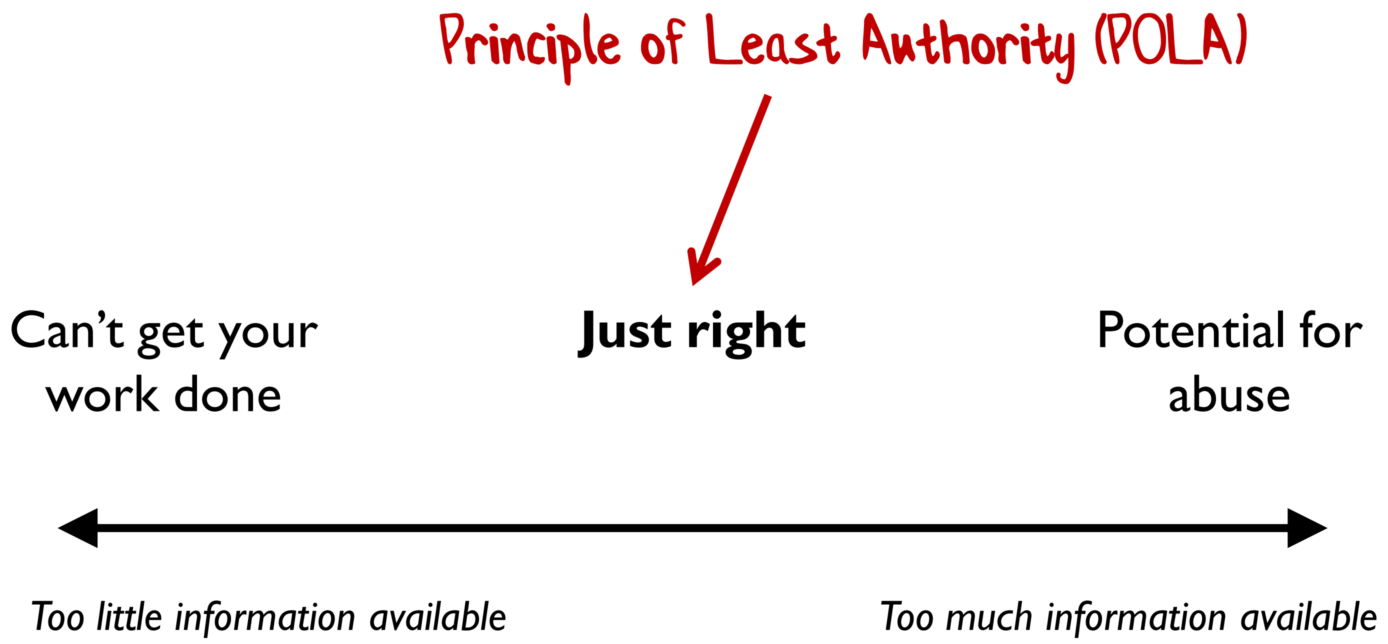
- If you make information available:
  - Programmers can’t help but make use of it
  - Even if not in best interests of the design
- Solution:
  - Don’t make information available!

Software Design Spectrum

In the large: Bounded Contexts
In the small: Interface Segregation Principle


Can’t do anything
Just right
Unnecessary coupling
Too little information available
Too much information available

Security spectrum


Principle of Least Authority (POLA)
Can't get your work done
Just right
Potential for abuse
Too little information available
Too much information available

Good Software Design

Intention-revealing interface
Minimize coupling
Make dependencies explicit

Ak.a. Minimize your surface area
(expose only desired behavior)

Good Security

Principle of Least Authority (POLA)

Ak.a. Minimize your surface area
(to reduce chance of abuse)

Good security => Good design

Good design => Good security

Security-aware design

- "Authority" = what can you do at any point?
  - Be aware of authority granted
  - Assume malicious users as a design aid!

Stupid people

Evil people

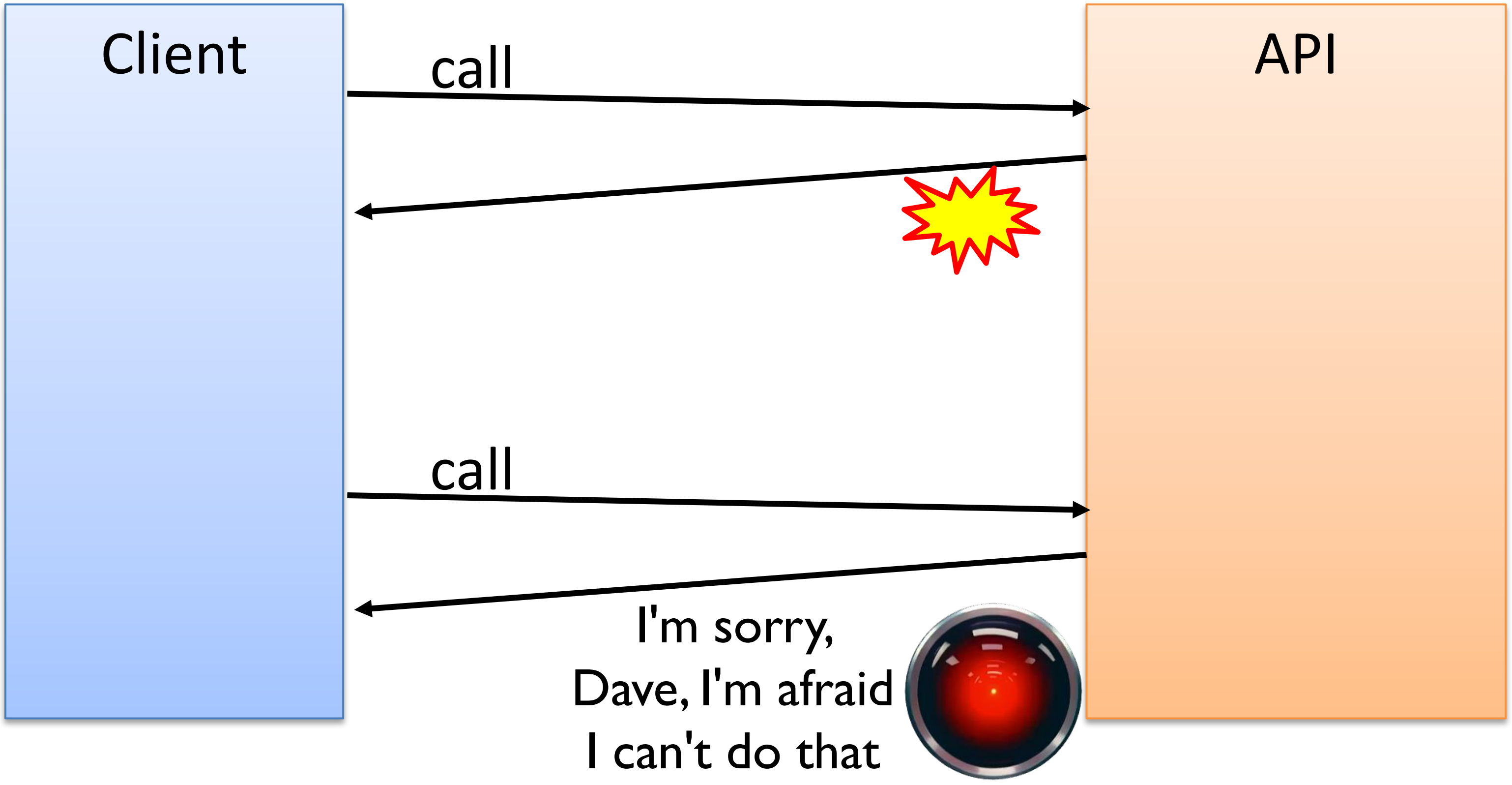
What's the difference? ☹

Security-aware design

- "Authority" = what can you do at any point?
  - Be aware of authority granted
  - Assume malicious users as a design aid!
- Use POLA as a software design guideline
  - Forces intention-revealing interface
  - Minimizes surface area & reduces coupling

INTRODUCING “CAPABILITIES”

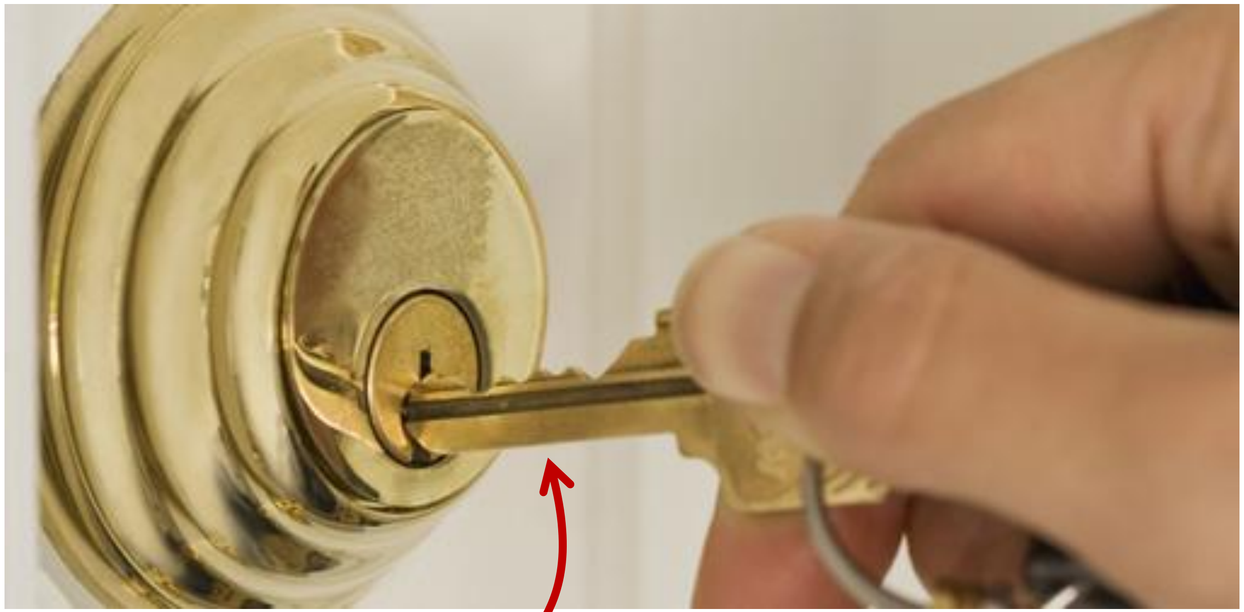
Typical API



Rather than telling me what I can't do, why not tell me what I can do?

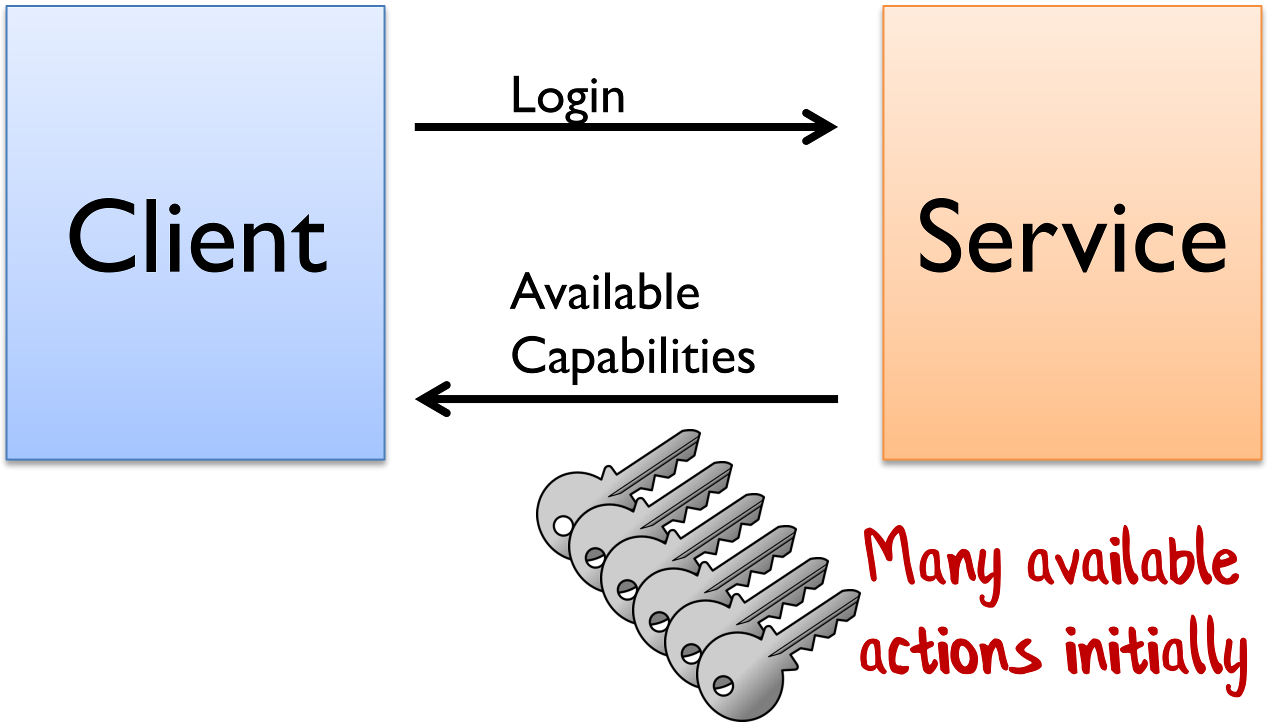
The ultimate "Intention-revealing interface"

Capability-based API

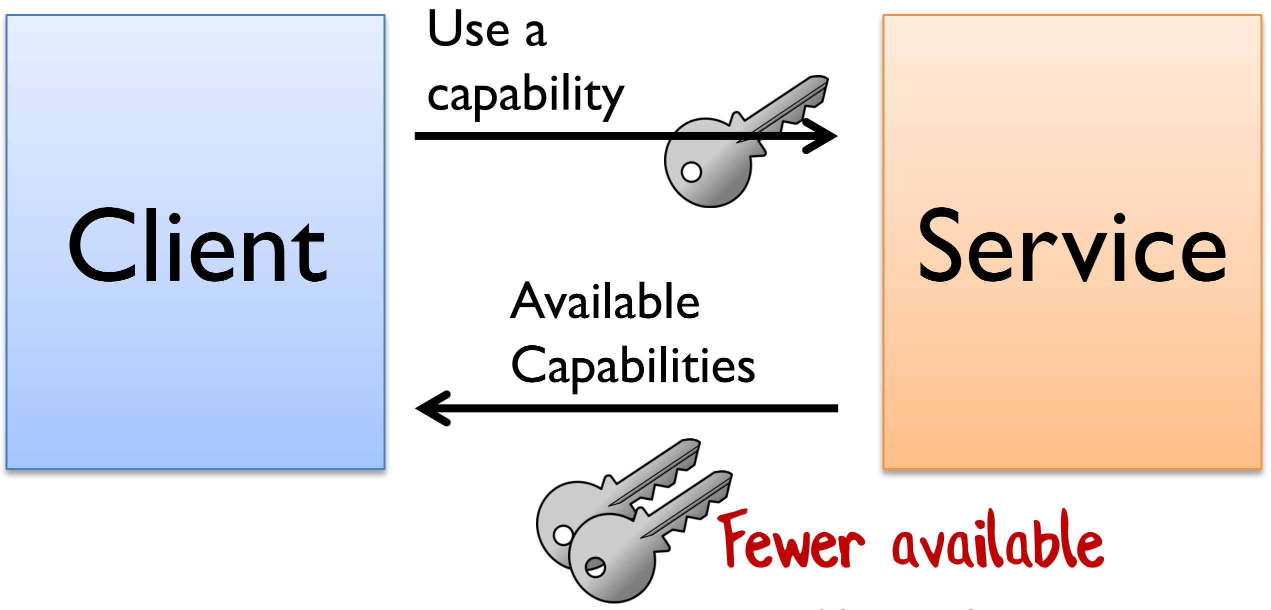


A capability

Capability-based API

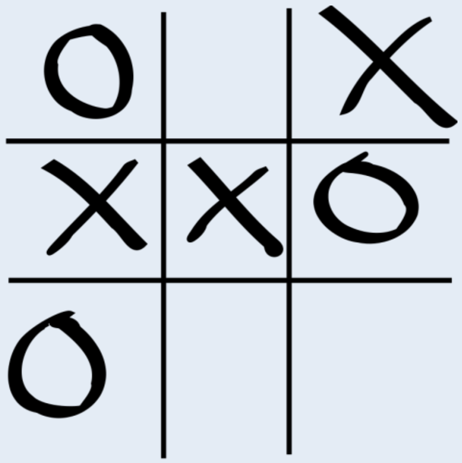


Capability-based API



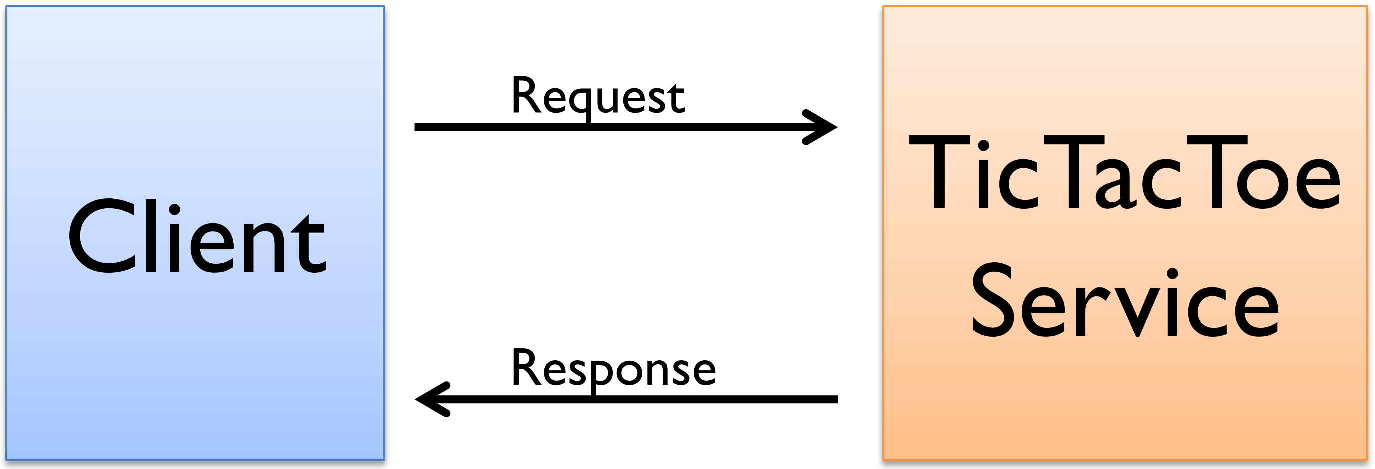
Fewer available actions in a given state

API DESIGN WITH CAPABILITIES



Tic-Tac-Toe as a service

Proper name is "Noughts and Crosses" btw
TIL: "Butter, cheese and eggs" in Dutch



Tic-Tac-Toe API (obvious version)

```
type TicTacToeRequest = {
    player: Player  // X or O
    row: Row
    col: Column
    }
```

Tic-Tac-Toe API (obvious version)

```
type TicTacToeResponse =
      | KeepPlaying
      | GameWon of Player
      | GameTied
```

"Choice" type

Demo:
Obvious Tic-Tac-Toe API

What kind of errors can happen?

- A player can play an already played move
- A player can play twice in a row
- A player can forget to check the response and keep playing

Not an intention-revealing interface

Intention-revealing interface

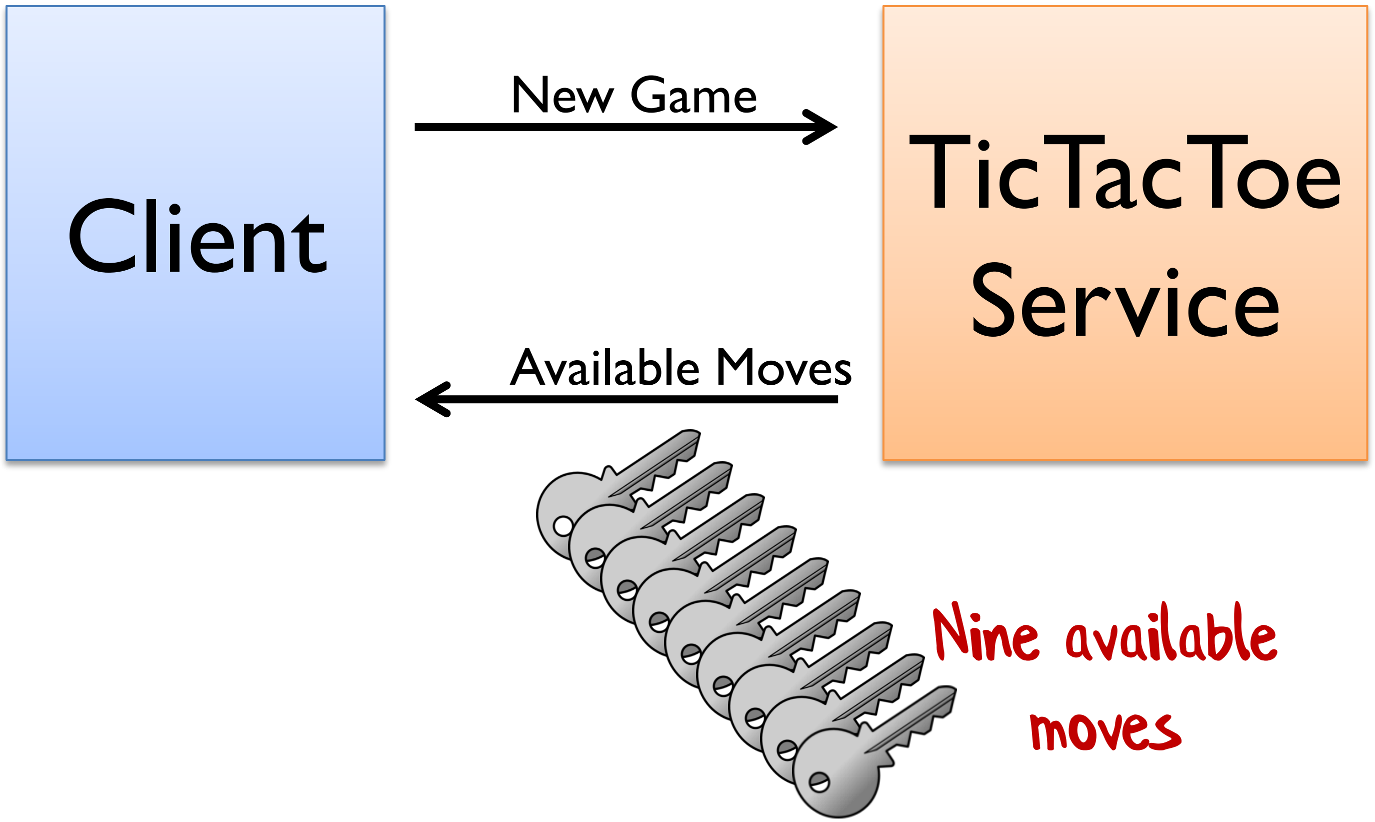
"If a developer must consider the implementation of a component in order to use it, the value of encapsulation is lost."
-- Eric Evans, DDD book

Yes, you could return errors, but...

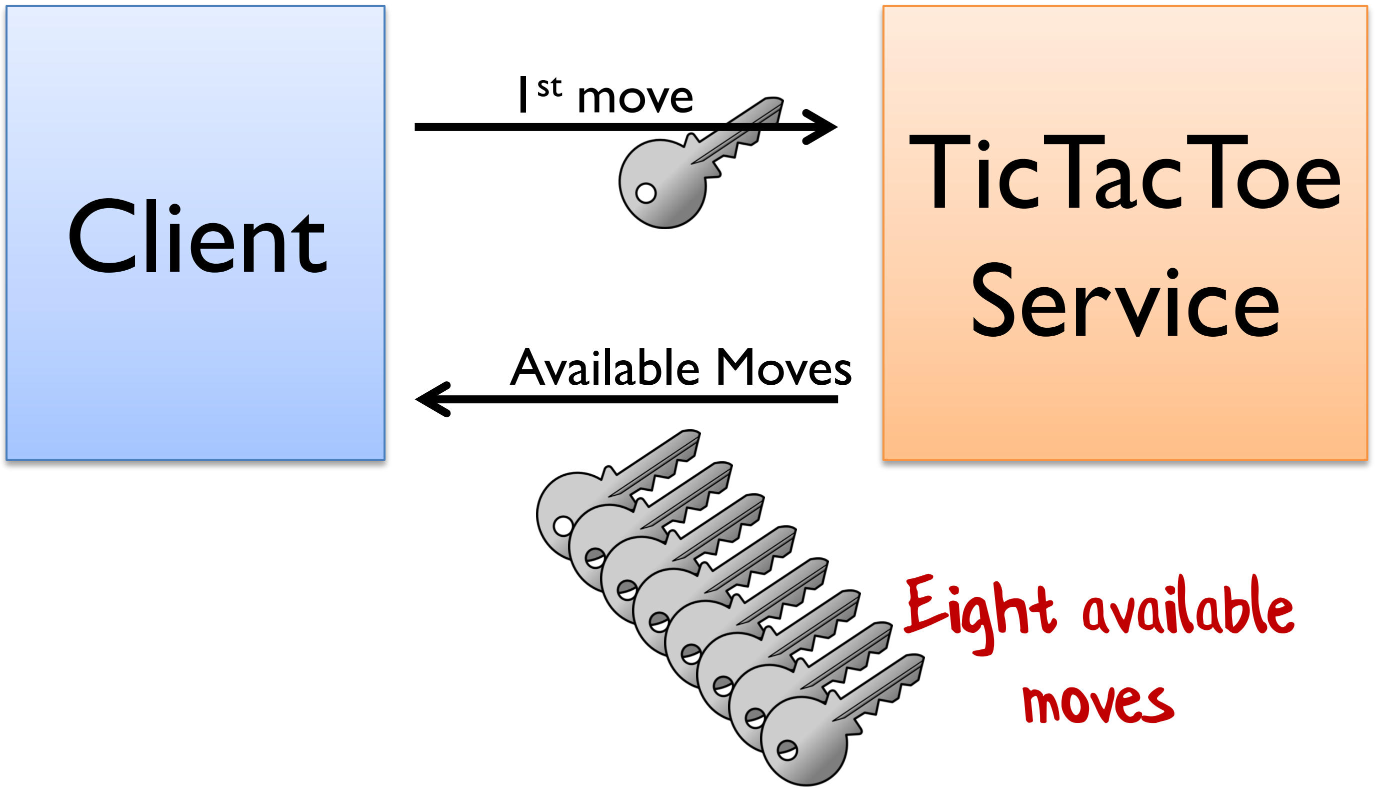
Don't let me do a bad thing and then tell me off for doing it...

"Make illegal operations unavailable"

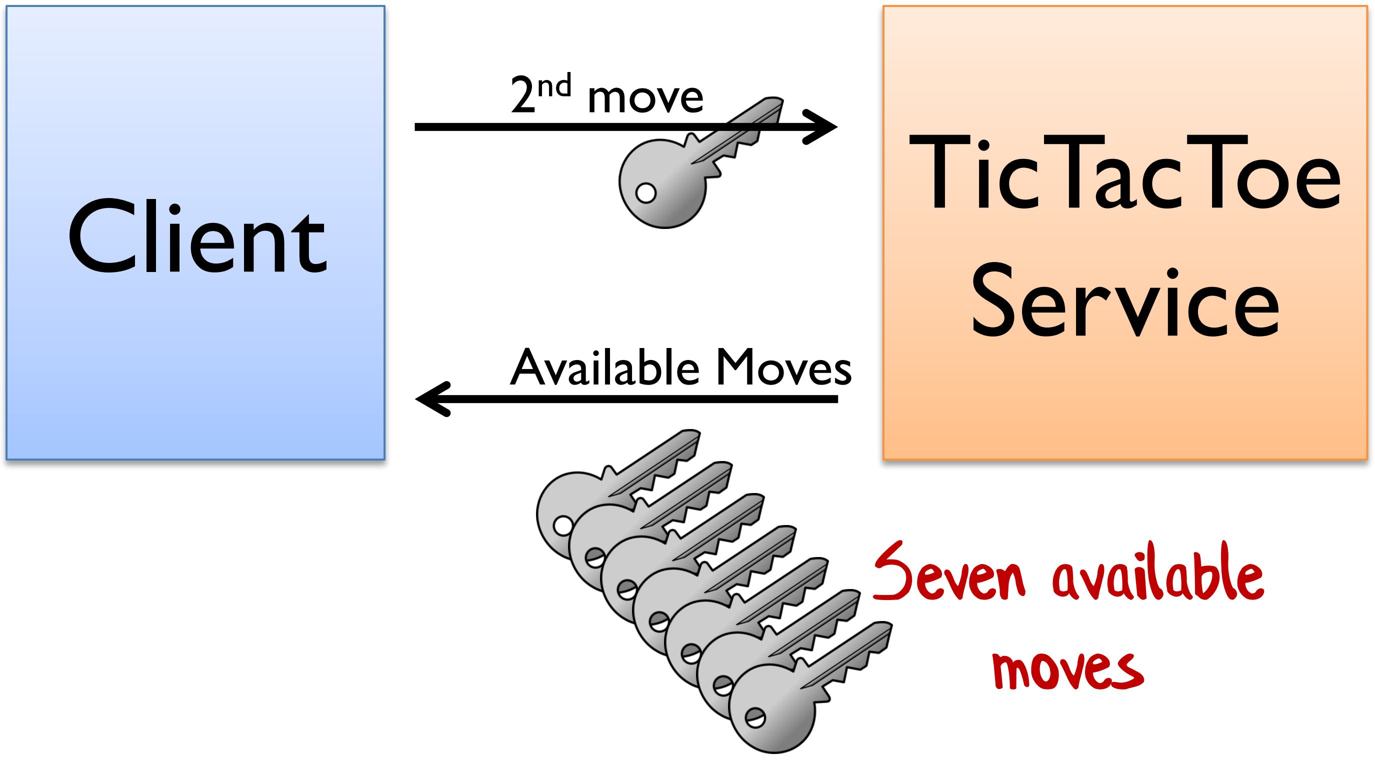
Tic-Tac-Toe service with capabilities



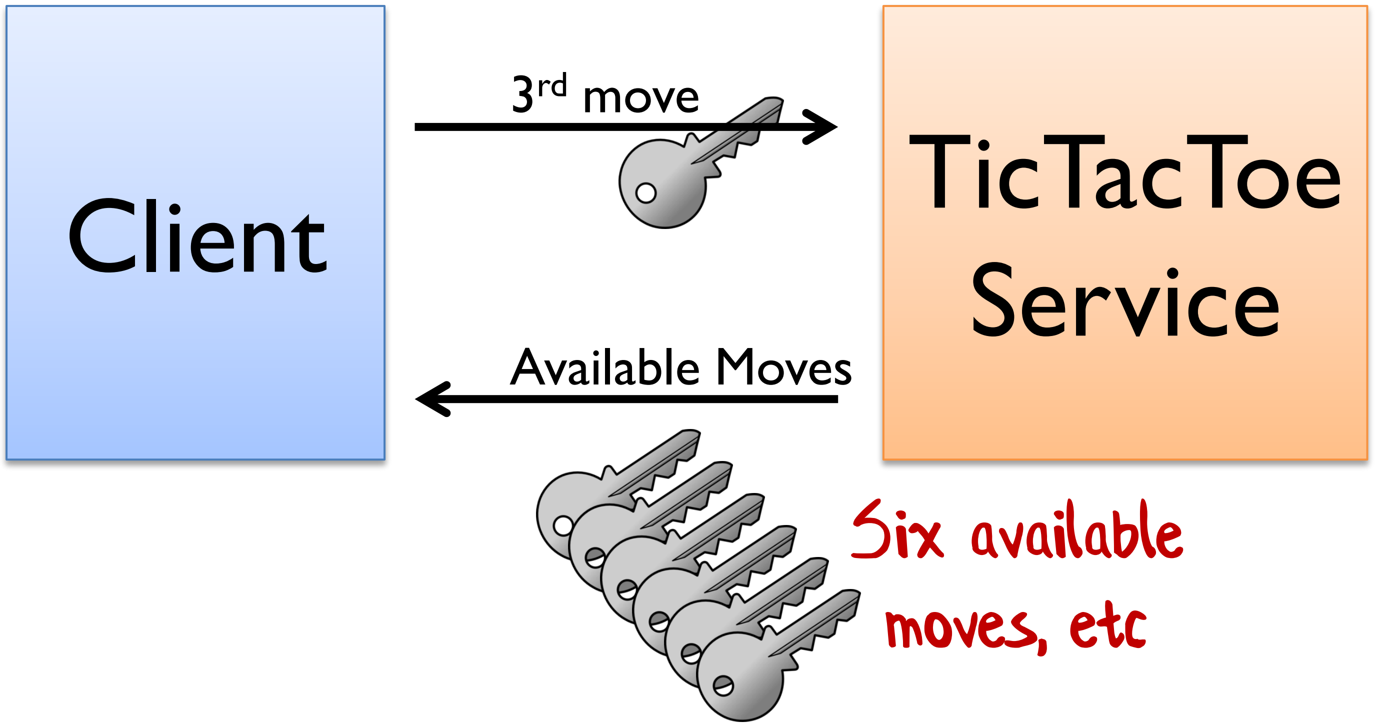
Tic-Tac-Toe service with capabilities



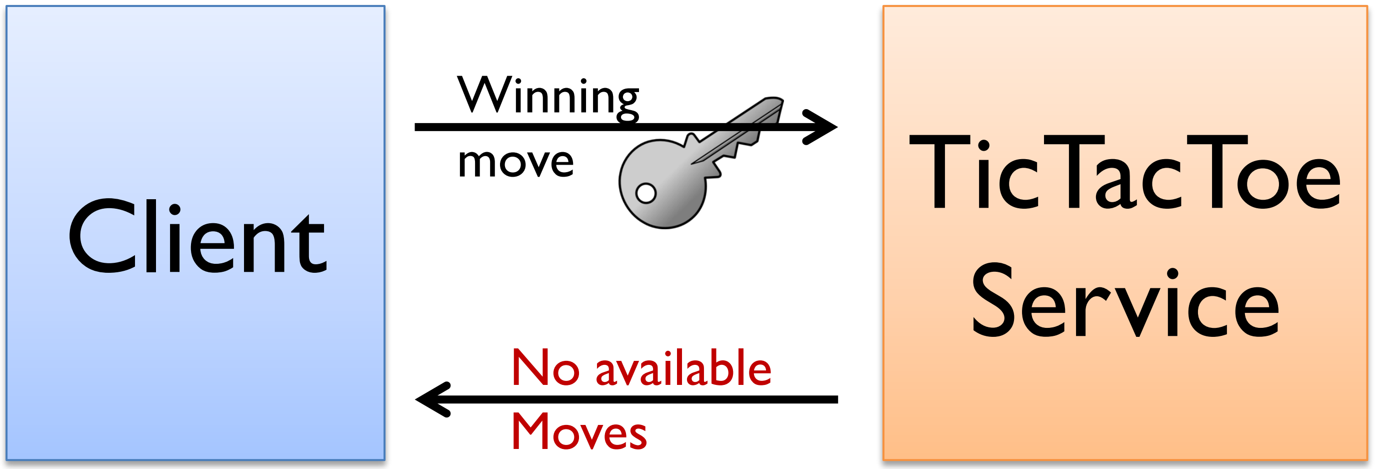
Tic-Tac-Toe service with capabilities



Tic-Tac-Toe service with capabilities



Tic-Tac-Toe service with capabilities



Tic-Tac-Toe API (cap-based version)

```
type MoveCapability =
  unit -> TicTacToeResponse
          // aka Func<TicTacToeResponse>

type TicTacToeResponse =
   | KeepPlaying of MoveCapability list
   | GameWon of Player
   | GameTied
```

Tic-Tac-Toe API (cap-based version)

```
type MoveCapability =
  unit -> TicTacToeResponse
          // aka Func<TicTacToeResponse>

type TicTacToeResponse =
   | KeepPlaying of MoveCapability list
   | GameWon of Player
   | GameTied
```

Response contains all available moves

An intention-revealing interface

Tic-Tac-Toe API (cap-based version)

```
type MoveCapability =
  unit -> TicTacToeResponse
          // aka Func<TicTacToeResponse>

type TicTacToeResponse =
   | KeepPlaying of MoveCapability list
   | GameWon of Player
   | GameTied

type InitialMoves = MoveCapability list
```

The entire API!

Where did the "request" type go?

Where's the authorization?

Demo:
Capability-based Tic-Tac-Toe

What kind of errors can happen?

- ~~A player can play an already played move~~
- ~~A player can play twice in a row~~
- ~~A player can forget to check the response and keep playing~~

All fixed now! ☺

Is this good security or good design?

RESTful done right

HATEOAS
Hypermedia As The Engine Of Application State

"A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia."

How NOT to do HATEOAS

```
POST /customers/
GET /customer/42
```

If you can guess the API
you're doing it wrong

Security problem!

Also, a design problem —
too much coupling.

How to do HATEOAS

```
POST /81f2300b618137d21d /
GET /da3f93e69b98
```

You can only know what URLs to use by parsing the page

Each of these URLs is a capability

Tic-Tac-Toe HATEOAS

```
[
{ "move": "Play (Left, Top)",
  "rel": "Left Top",
  "href": "/move/ec03def5-7ea8-4ac3-baf7-b290582cd3f2" },
{ "move": "Play (Left, Middle)",
  "rel": "Left Middle",
  "href": "/move/d4532ca0-4e61-4fae-bbb1-fc11d4e173df" },
{ "move": "Play (Left, Bottom)",
  "rel": "Left Bottom",
  "href": "/move/fe1bfa98-e77b-4331-b99b-22850d35d39e" }
...
]
```

capabilities

An intention-revealing interface

Demo: Tic-Tac-Toe HATEOAS

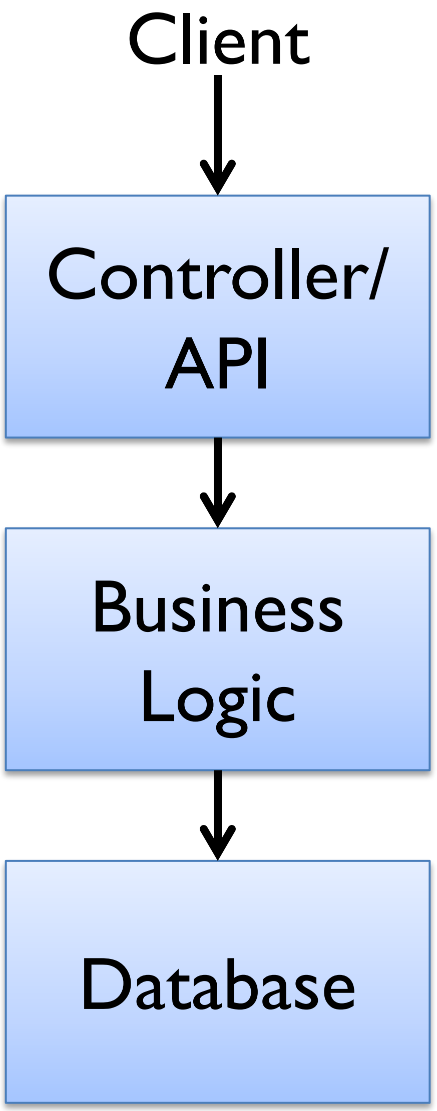
Good security => Good design

Good design => Good security

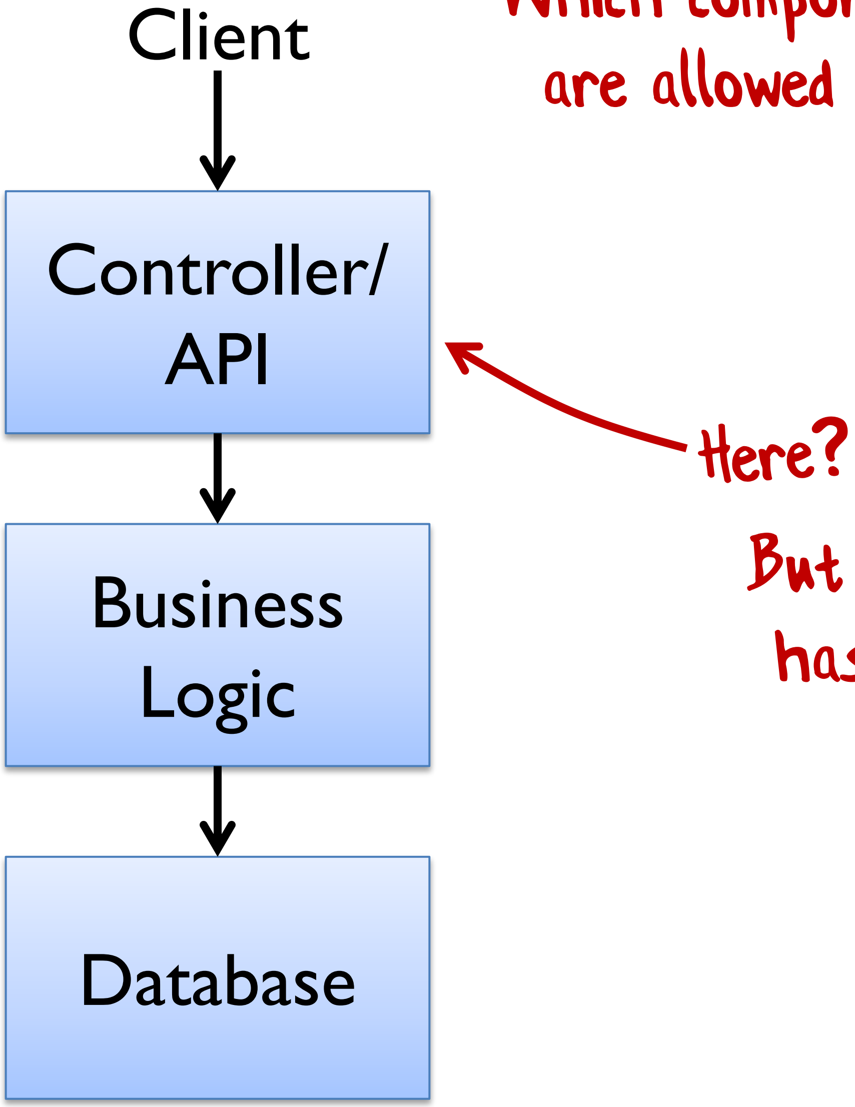
DESIGN CONSEQUENCES OF USING CAPABILITIES

Not just for APIs -- use these design techniques inside a bounded context too

Example:
Read a customer from a database



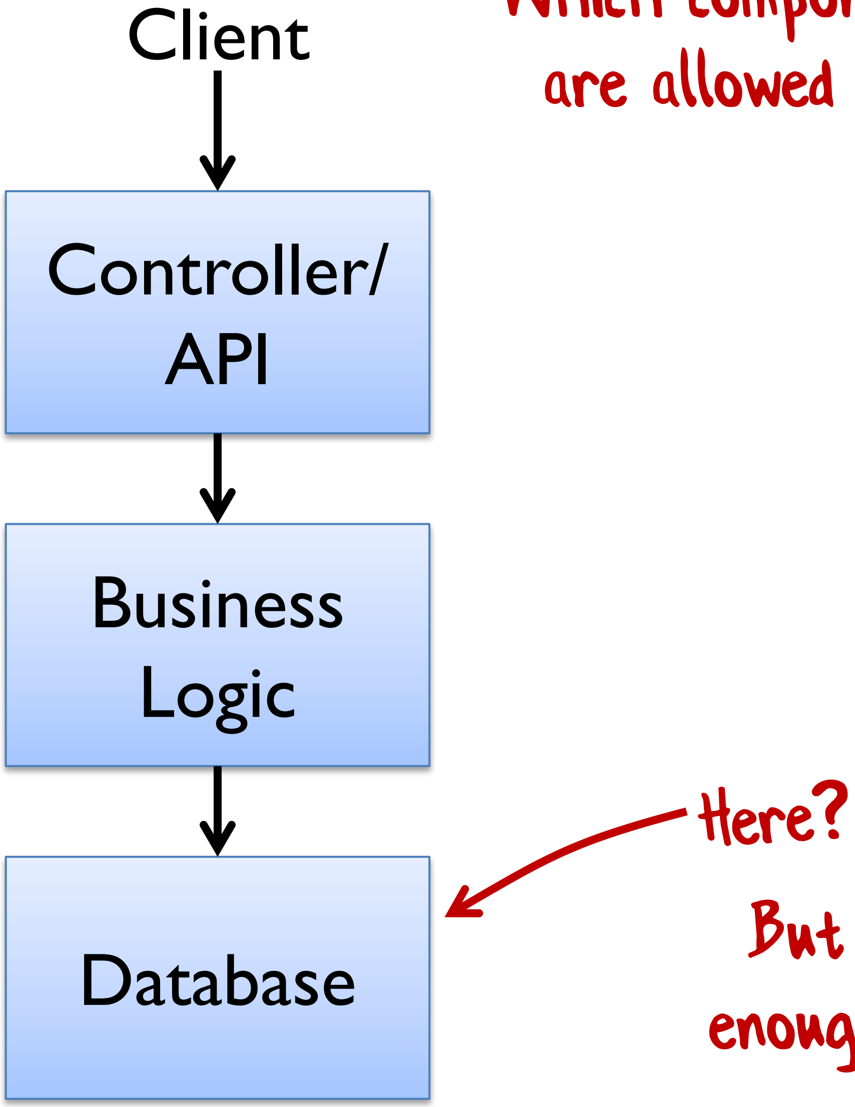
Could also be Onion architecture or Ports and Adapters -- not important



Which component decides whether you are allowed to read the customer?

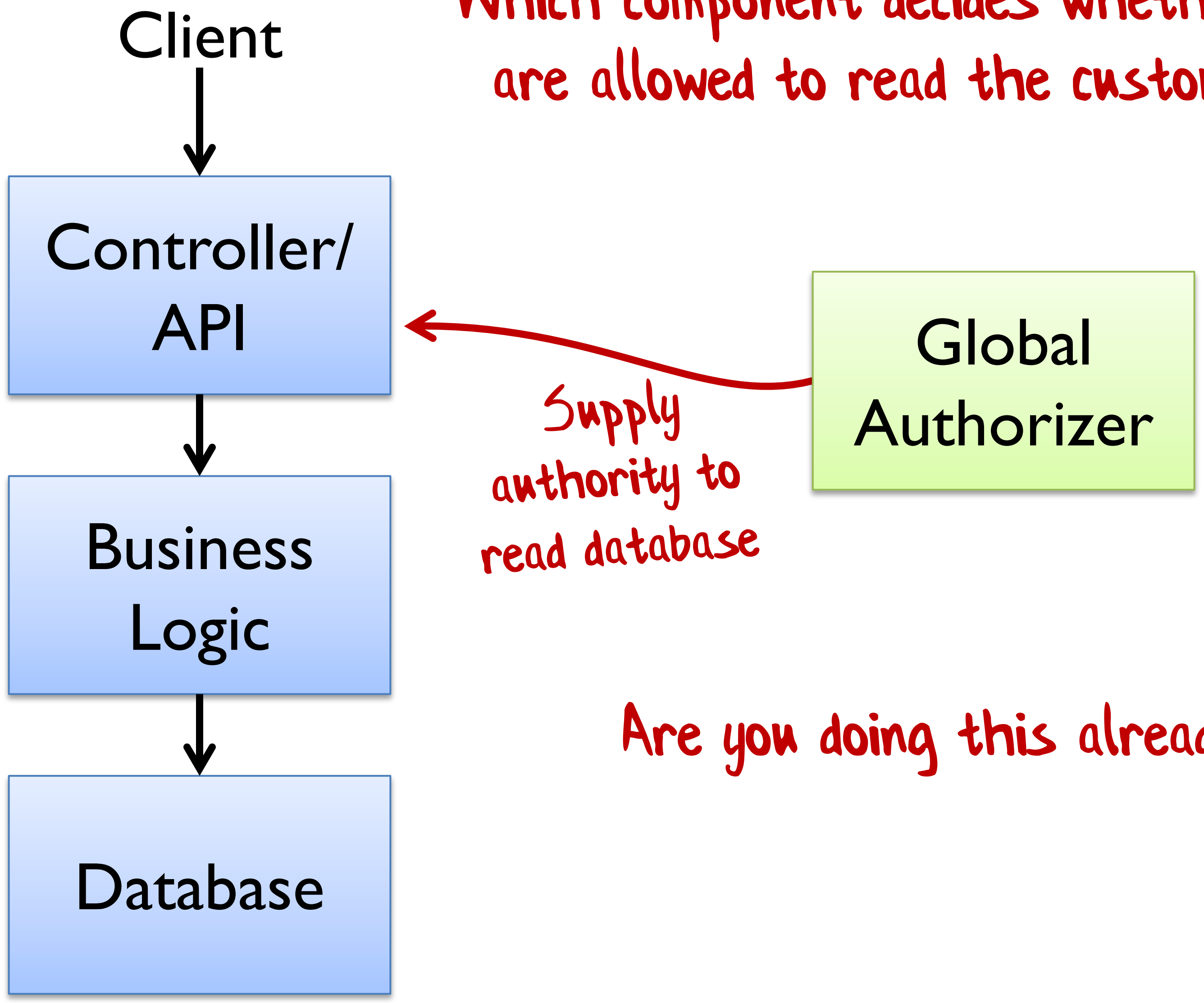
But then any other path has complete access to the database ☹

Which component decides whether you are allowed to read the customer?



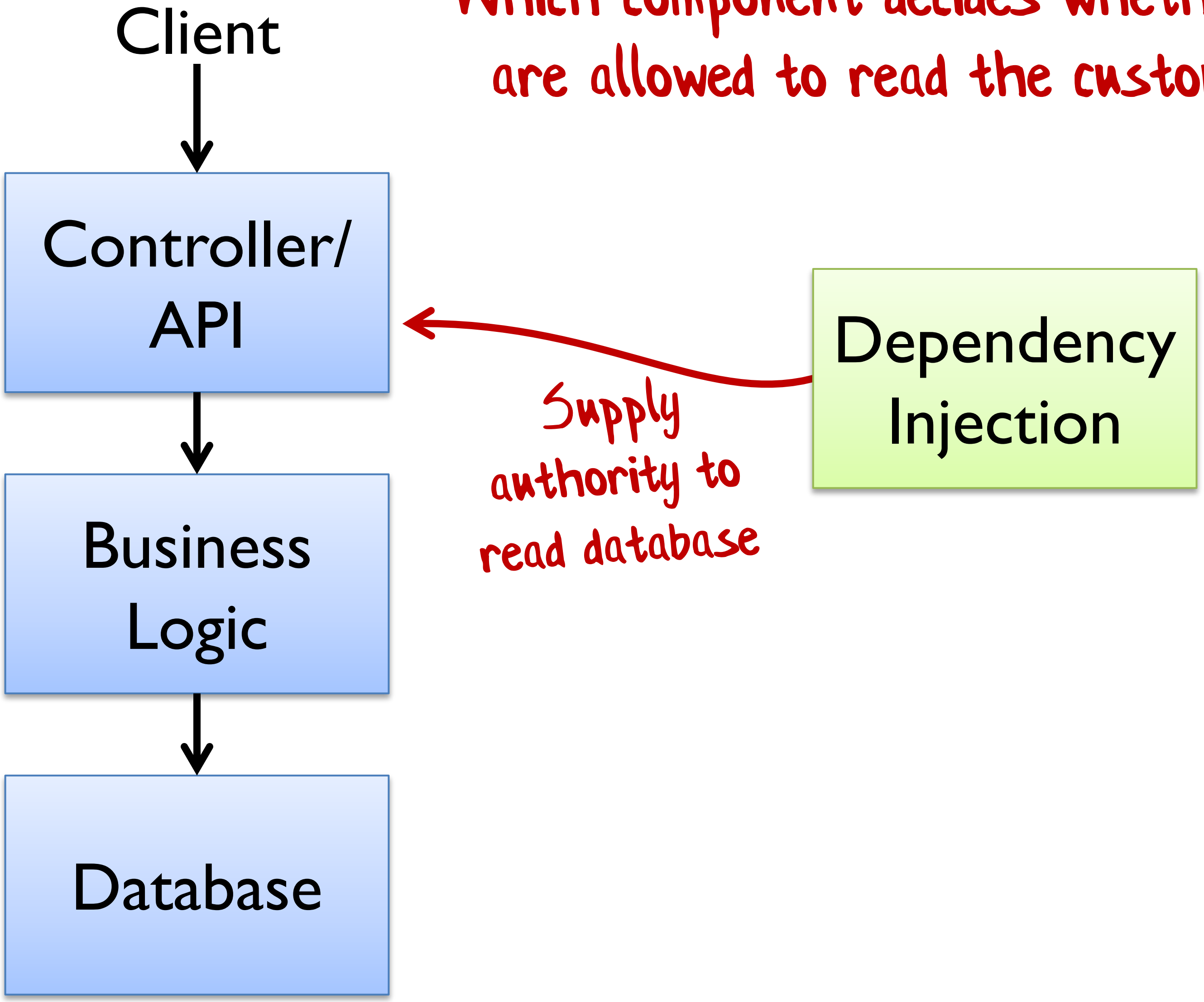
But then it doesn't have enough context to decide ☹

Which component decides whether you are allowed to read the customer?



Are you doing this already?

Which component decides whether you are allowed to read the customer?



```
public class CustomerController : ApiController
{
    readonly ICustomerDb _db;
    public CustomerController(ICustomerDb db)
    {
        _db = db;
    }

    [Route("customers/{customerId}")]
    [HttpGet]
    public IHttpActionResult Get(int customerId)
    {
        var custId = new CustomerId(customerId);
        var cust = _db.GetProfile(custId);
        var dto = DtoConverter.CustomerToDto(cust);
        return Ok(dto);
    }
```

Inject authority to access db

Use the authority

How much authority do you really need?

```
public interface ICustomerDb
{
  CustomerProfile GetProfile(CustomerId id);
  void UpdateProfile(CustomerId id, CustomerProfile cust);

  void CreateAccount(CustomerId id, CustomerProfile cust);
  void DeleteAccount(CustomerId id);

  void UpdateLoginEmail(CustomerId id, string email);
  void UpdatePassword(CustomerId id, string password);
}
```

Too much authority!

The only authority I need

How much authority do you really need?

```
public interface ICustomerDb
{
  CustomerProfile GetProfile(CustomerId id);
~~  void UpdateProfile(CustomerId id, CustomerProfile cust);~~

~~  void CreateAccount(CustomerId id, CustomerProfile cust);~~
~~  void DeleteAccount(CustomerId id);~~

~~  void UpdateLoginEmail(CustomerId id, string email);~~
~~  void UpdatePassword(CustomerId id, string password);~~
}
```

How much authority do you really need?

```
public interface ICustomerDb
{
  CustomerProfile GetProfile(CustomerId id);
}
```

The only authority
I need for this use-case

A whole interface
for one method?

How much authority do you really need?

```
Func<CustomerId,CustomerProfile>
```

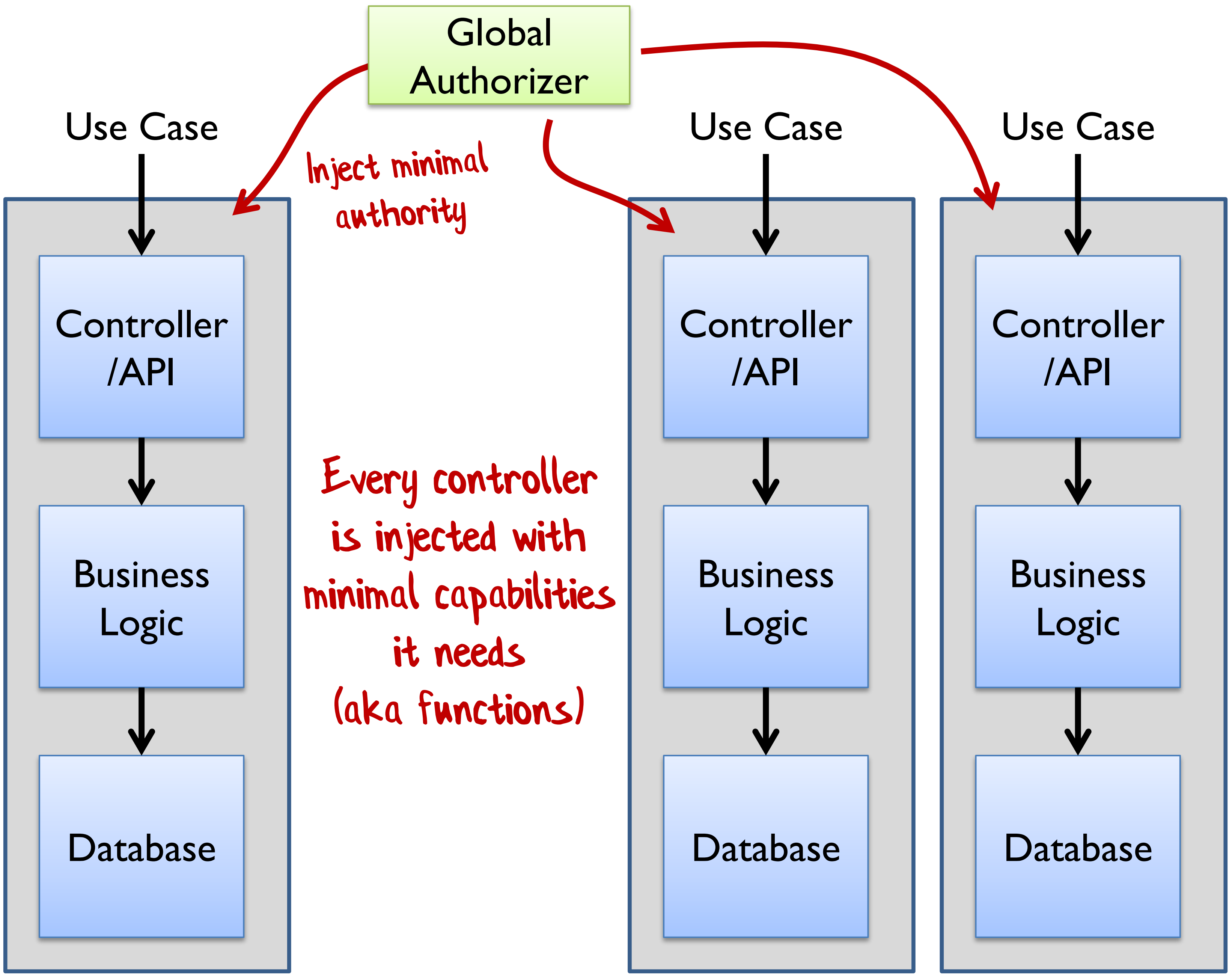
A single method interface is just a function!

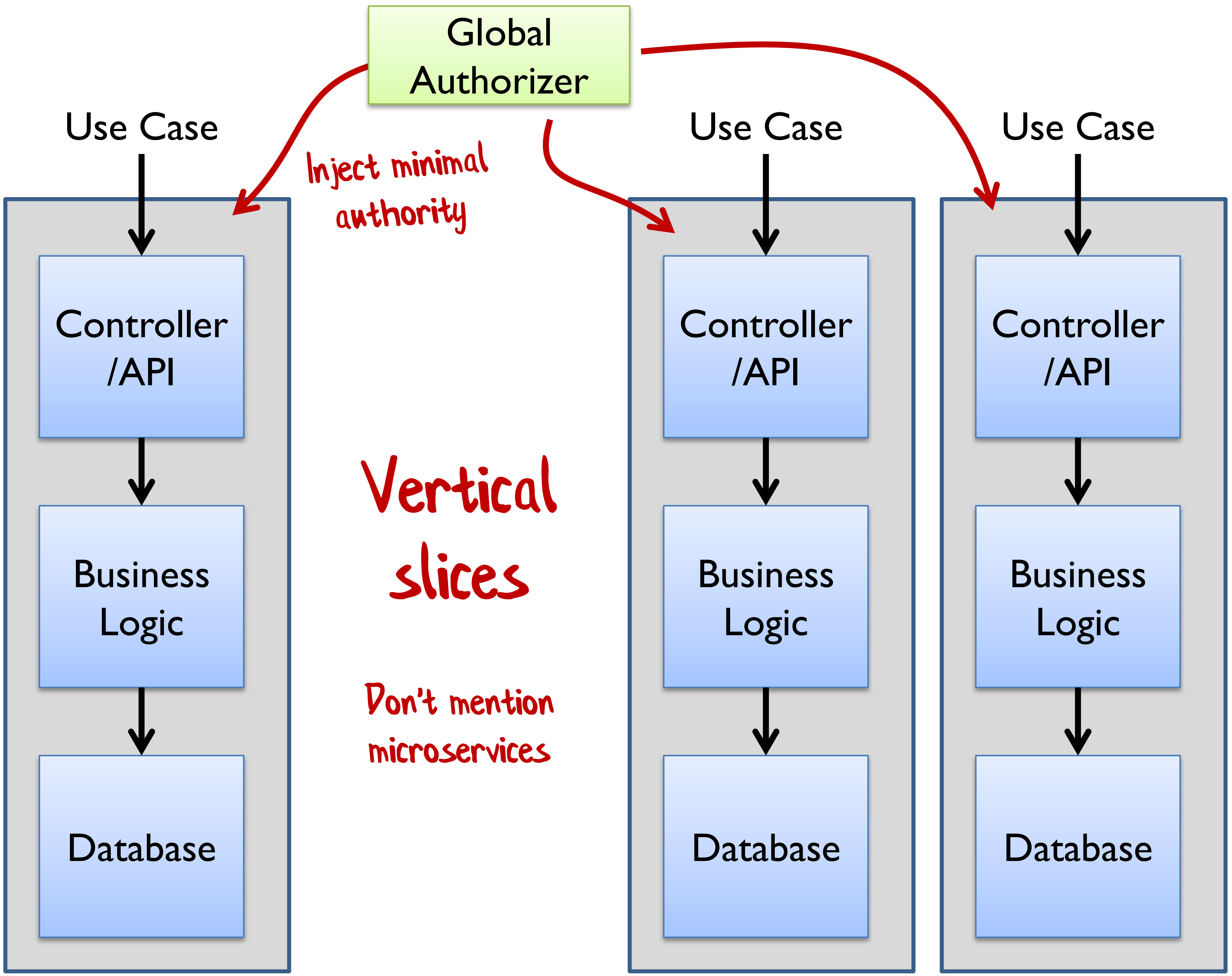
```
public class CustomerController : ApiController
{
    Func<CustomerId,CustomerProfile> _readCust;
    public CustomerController(Func<..> readCust)
    {
        _readCust = readCust;
    }

    [Route("customers/{customerId}")]
    [HttpGet]
    public IHttpActionResult Get(int customerId)
    {
        var custId = new CustomerId(customerId);
        var cust = _readCust(custId);
        var dto = DtoConverter.CustomerToDto(cust);
        return Ok(dto);
    }
```

Inject authority

Use the authority


Global Authorizer
Use Case
Use Case
Use Case
Inject minimal authority
Controller /API
Controller /API
Controller /API
Business Logic
Business Logic
Business Logic
Database
Database
Database
Every controller is injected with minimal capabilities it needs (aka functions)


Global Authorizer
Inject minimal authority
Use Case
Controller /API
Business Logic
Database
Use Case
Controller /API
Business Logic
Database
Use Case
Controller /API
Business Logic
Database
Vertical slices
Don't mention microservices

But wait, there's more!

Should we be allowed to
access ANY customer?

We need more fine-grained control

```
public class CustomerController : ApiController
{

  [Route("customers/{customerId}")]
  [HttpGet]
  public IHttpActionResult Get(int customerId)
  {
    var custId = new CustomerId(customerId);
    var readCust = authorizer.GetReadCustCap(custId);
    if (readCust != null)
    {
      var cust = readCust();
      var dto = DtoConverter.CustomerToDto(cust);
      return Ok(dto);
    }
    else
      // return error
  }
```

Attempt to get capability for this customer

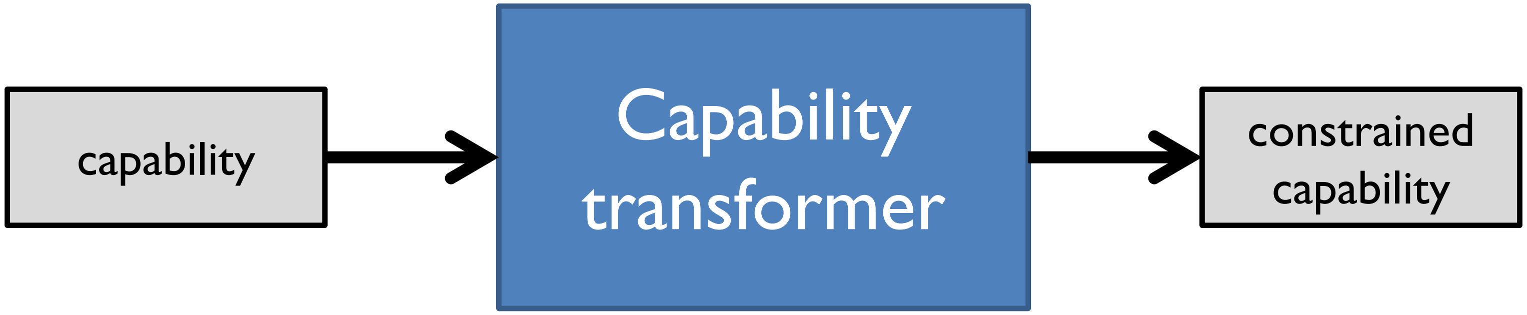
Check whether the capability exists

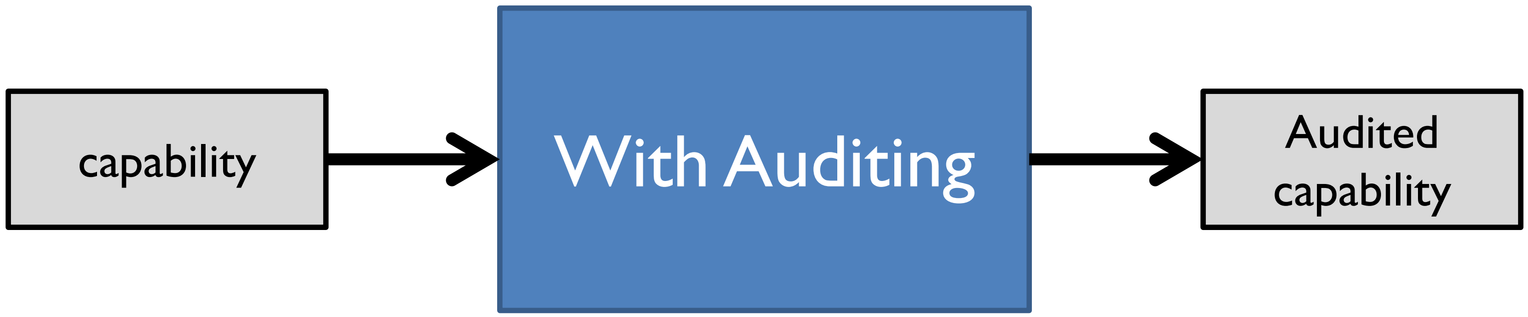
Don't need to pass in customer id

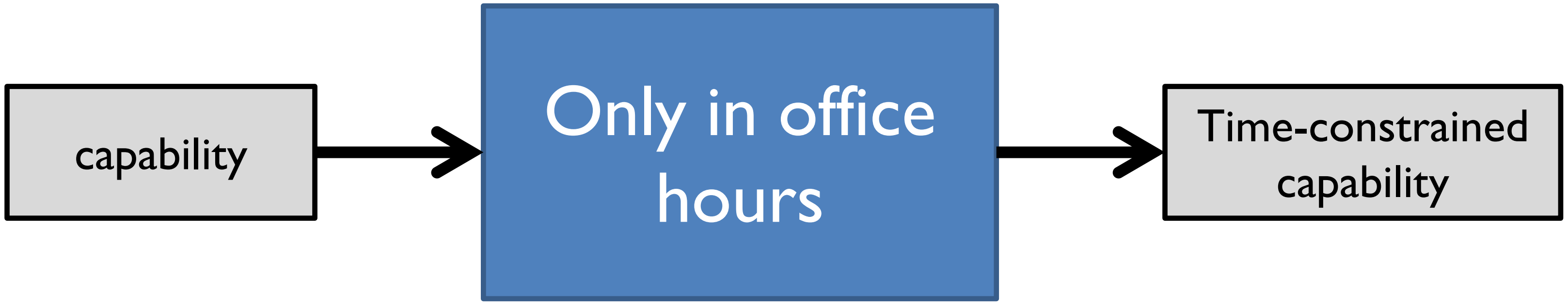
TRANSFORMING CAPABILITIES FOR BUSINESS RULES

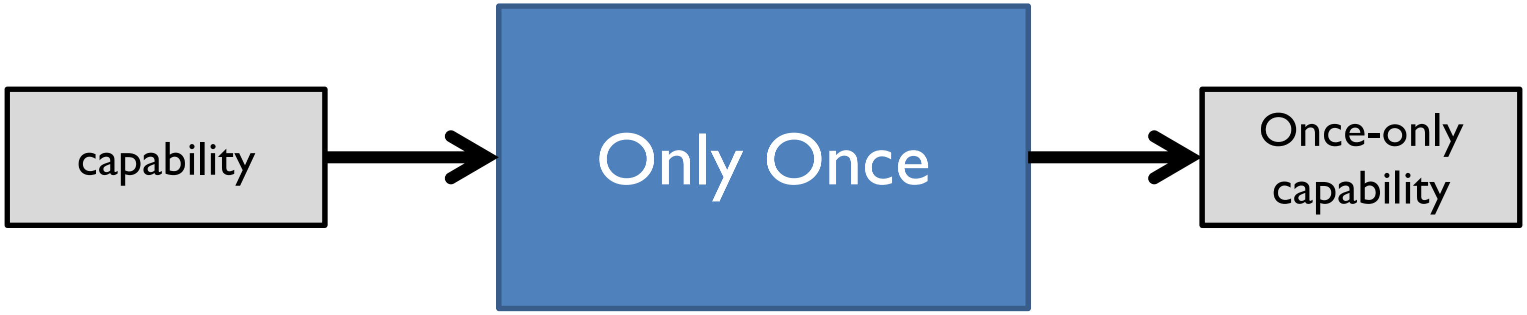
Capabilities are functions...

...so can be transformed to implement business rules




capability
With Auditing
Audited capability

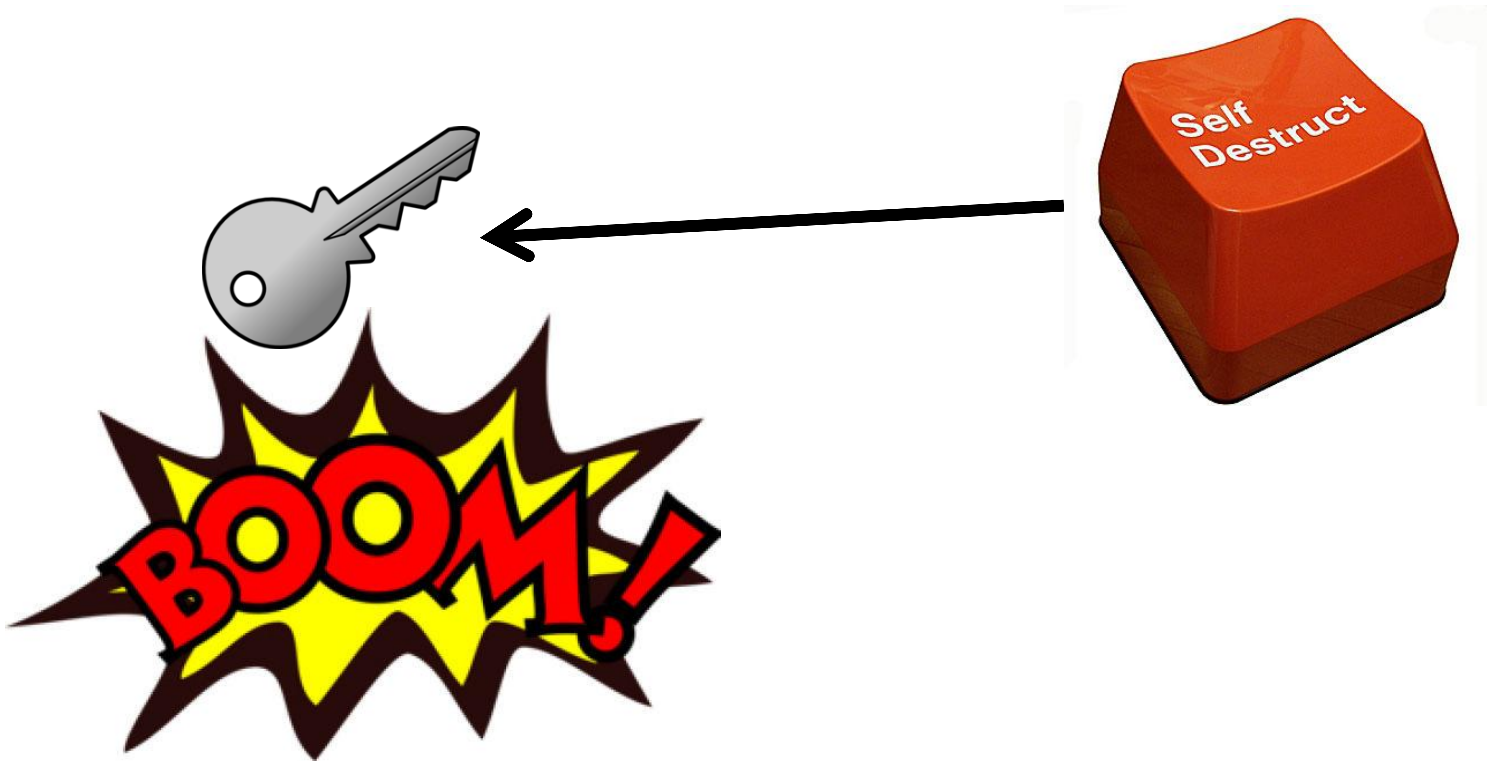

capability
Only in office hours
Time-constrained capability

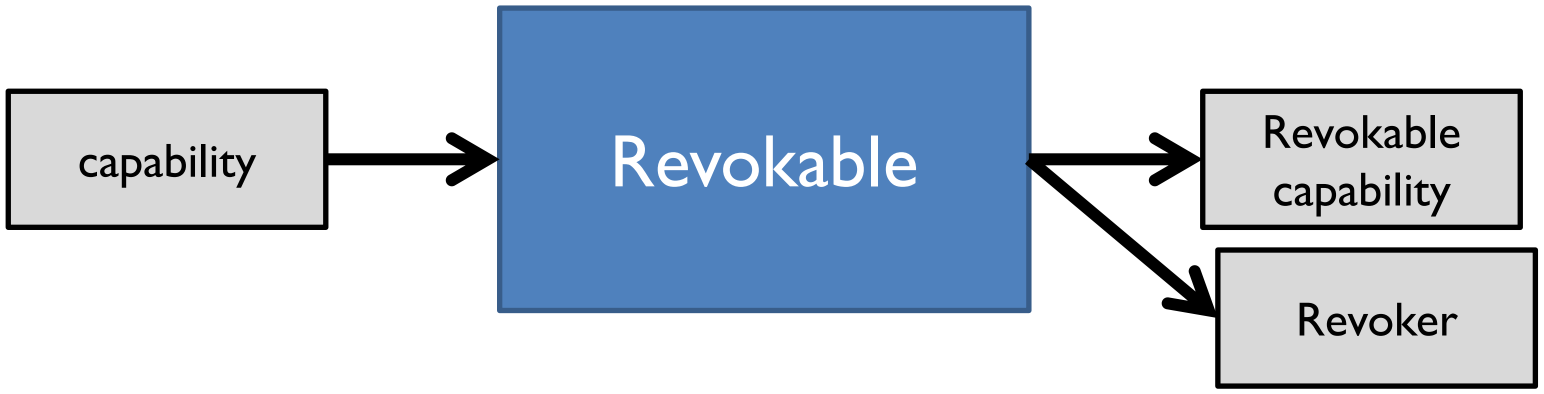

capability
Only Once
Once-only capability

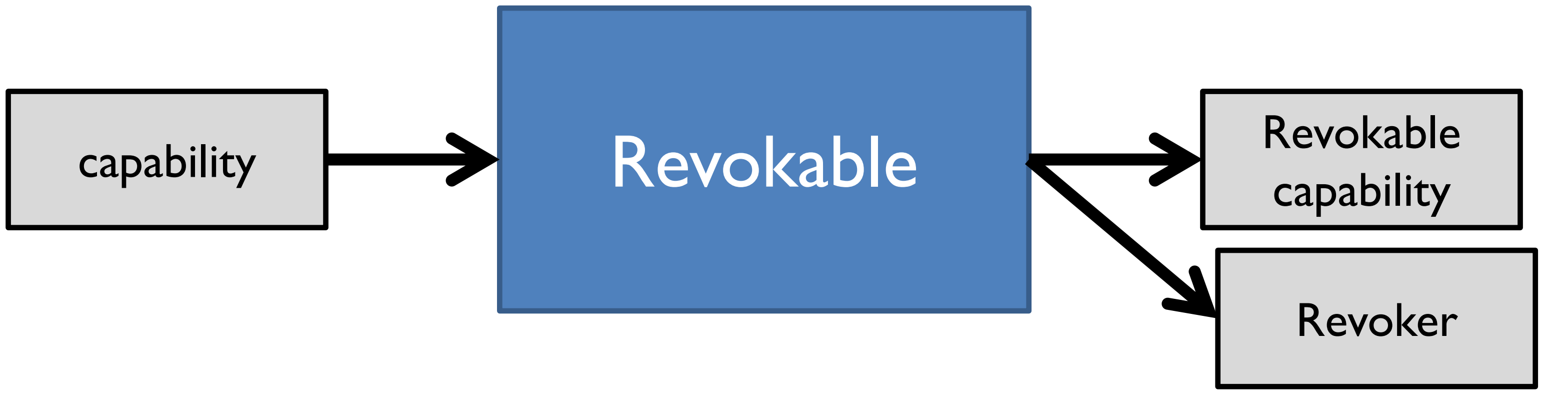
How to revoke access in a cap-based system?

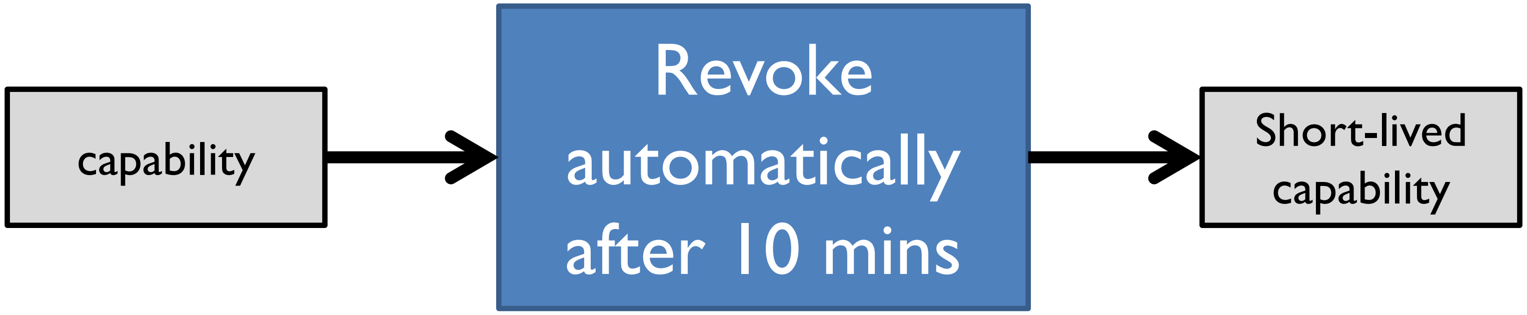
It's hard to revoke physical keys in the real world...

But this is software!




capability
Revokable
Revokable capability
Revoker


capability
Revokable
Revokable capability
Revoker


capability
Revoke automatically after 10 mins
Short-lived capability

Demo: Transforming Capabilities

DELEGATING AUTHORITY USING CAPABILITIES

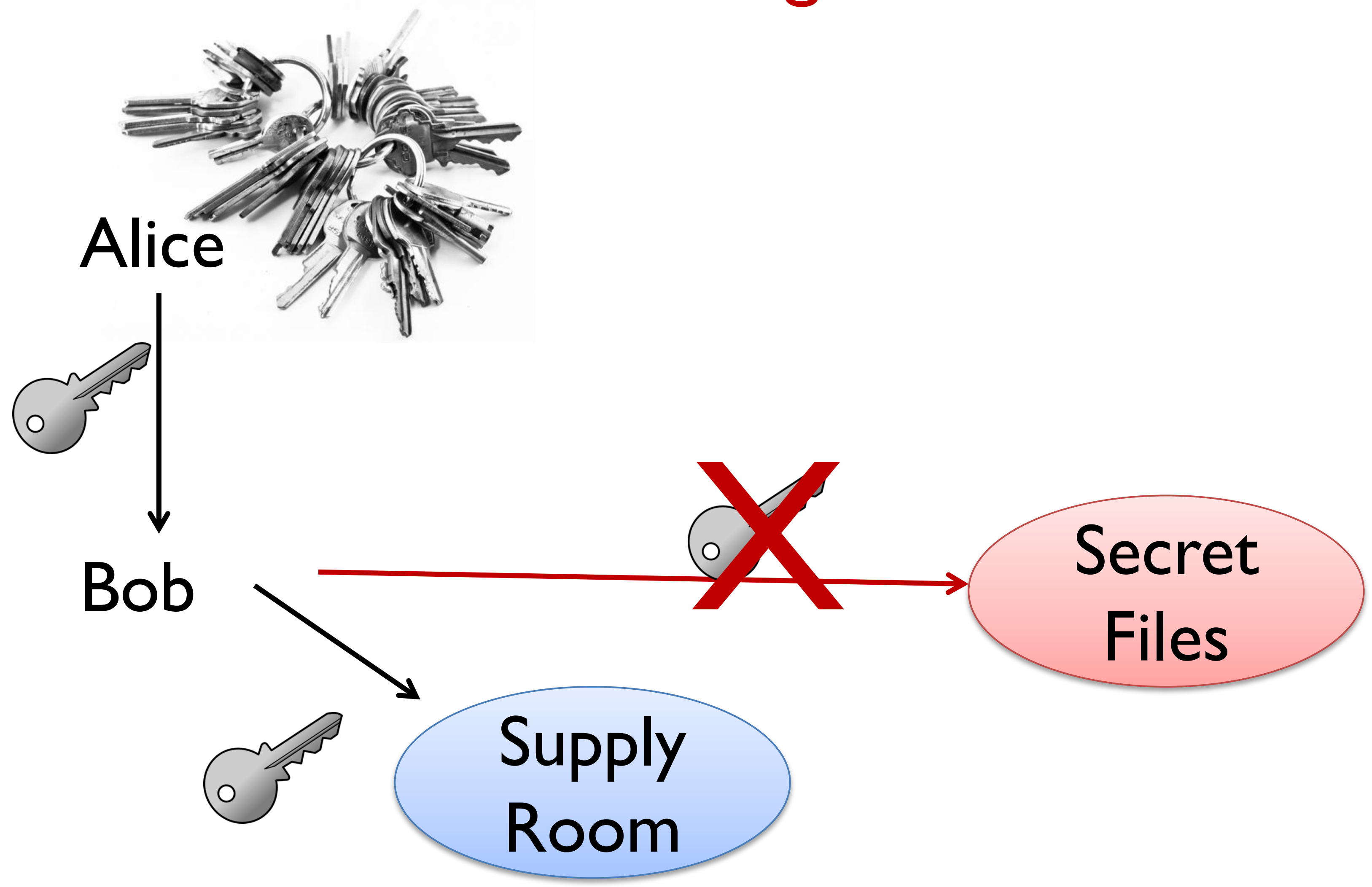
Reasons for access control

- **Prevent** any access at all.
- **Limit** access to some things only.
- **Revoke** access when you are no longer allowed.
- **Grant** and delegate access to some subset of things.

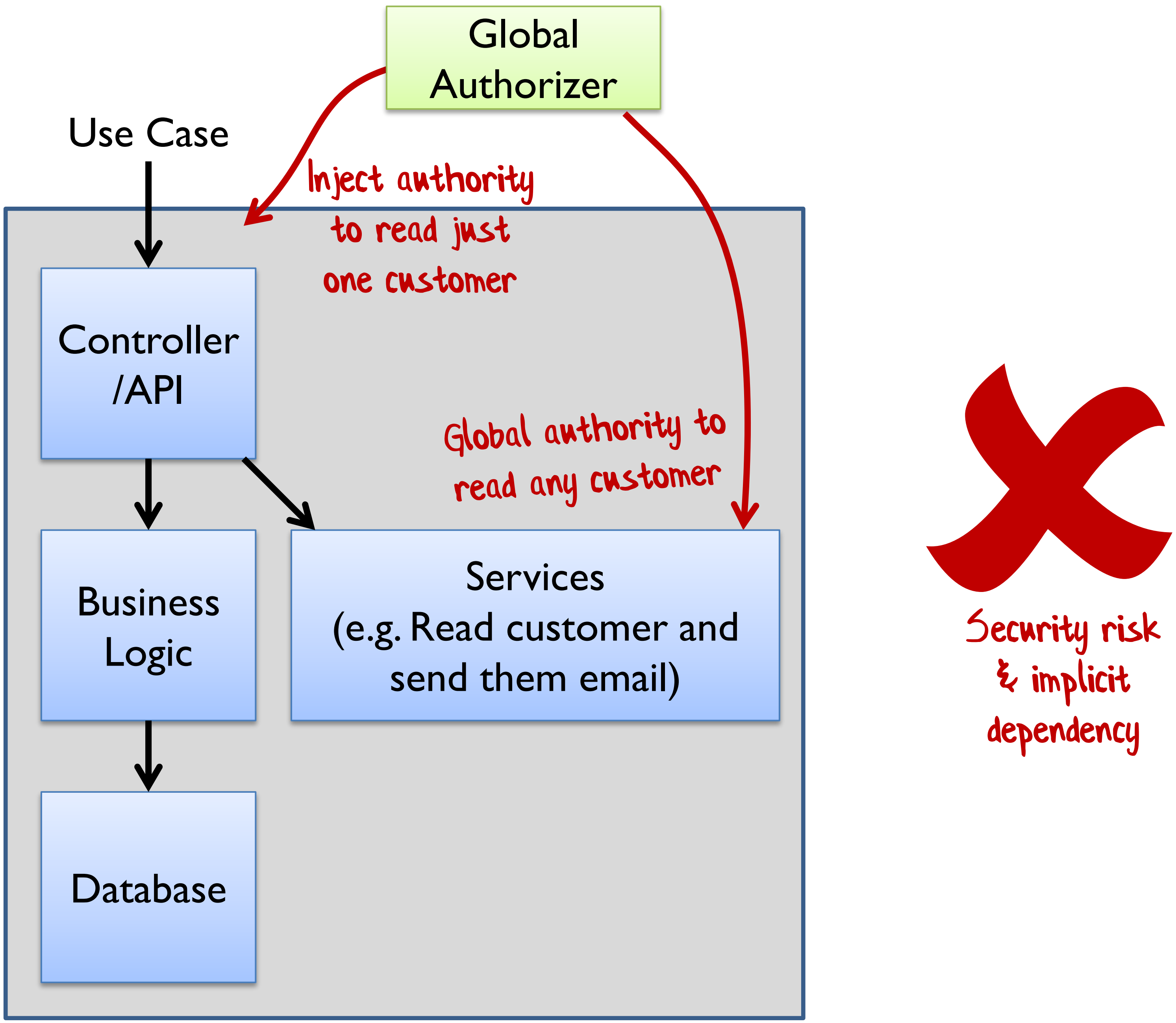
It's not always about saying no!

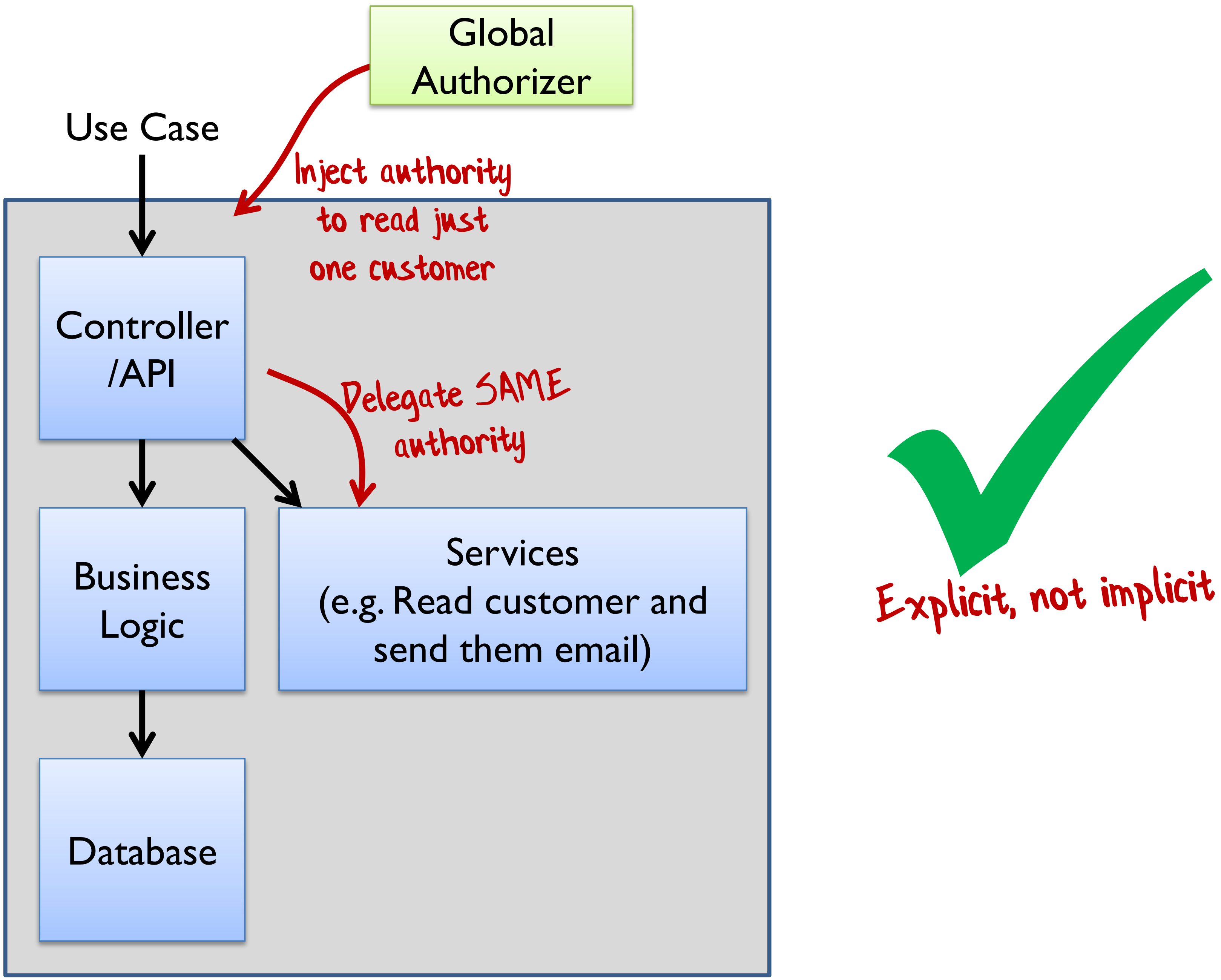
A set of capabilities

Capabilities support decentralized delegation

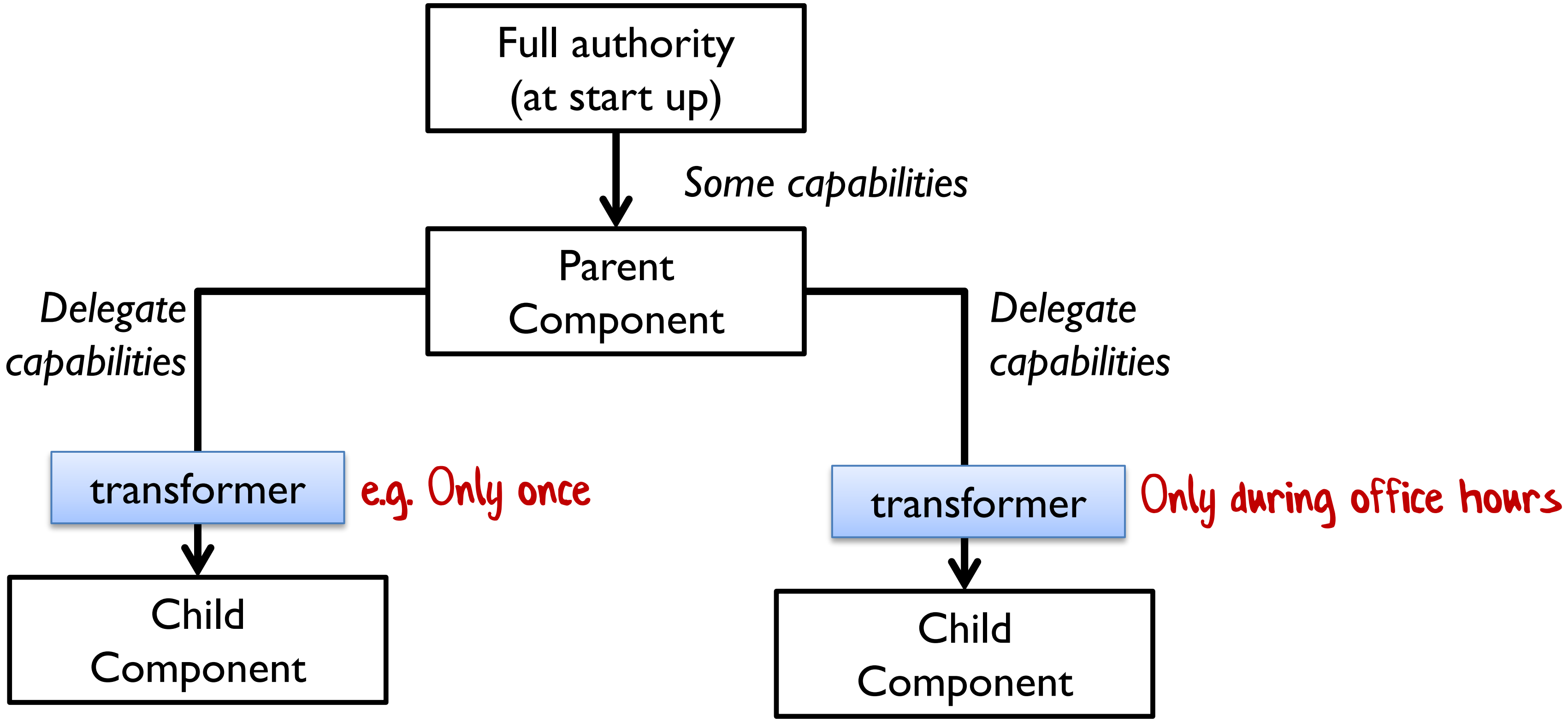


Delegation of authority example


Global Authorizer
Use Case
Inject authority to read just one customer
Controller /API
Global authority to read any customer
Business Logic
Services (e.g. Read customer and send them email)
Database
Security risk & implicit dependency


Global Authorizer
Use Case
Inject authority to read just one customer
Controller /API
Delegate SAME authority
Business Logic
Services (e.g. Read customer and send them email)
Database
Explicit, not implicit

Delegation of capabilities



CONCLUSION

Common questions

- Is this overkill? Is it worth it?
  - It depends....
  - Useful as a thought experiment
- How does this relate to DDD?
  - Intention-revealing interfaces
  - Map commands from event storming to capabilities

Common questions

- Are you saying that *all* external IO should be passed around as capabilities?
  - Yes! You should never access any ambient authority.
  - You should be doing this anyway for mocking.
- How do you pass these capabilities around?
  - Dependency injection or equivalent

Common questions

- Won’t there be too many parameters?
  - Less than you think!
  - Counter force to growth of interfaces
  - Encourages vertical slices (per use-case)
- Can’t this be bypassed by reflection or other backdoors?
  - Yes. This is really all about design not about total security.

Summary

- **Good security → good design**
  - Bonus: get a modular architecture!
- **Use POLA as a design principle**
  - Don’t trust other people to do the right thing
  - Don’t force other people to read the documentation!
- **Intention revealing interfaces**
  - Don't force the client to know the business rules
  - Make interfaces more dynamic
  - Change the available capabilities when context changes

Thanks!

Contact me

@ScottWlaschin

fsharpforfunandprofit.com/cap

Slides and video here

F# consulting

fsharpWorks

More F# at fsharp.org

fsharp.org