

Flash Security: Why and How

Deneb Meketa

Flash Player
Security Engineer



My Role at Adobe



- I work on *content restrictions* – what SWFs are and aren't allowed to do
 - It's a slow, careful job, requiring a lot of delicate rules
 - I get involved in other areas of security, but my main focus is the rules – aka the model
- I also work on *best practices* – how you can design your apps to be secure
- There are other aspects of security that I won't be covering today
 - Incident response: addressing vulnerabilities found in our products or customers' sites
 - Crash-proofing: avoiding and fixing exploitable programming errors in our products
 - Digital rights management: dictating what users can and can't do with content
 - Intellectual property protection: making it difficult for others to steal your code

Two Questions I Get Asked a Lot



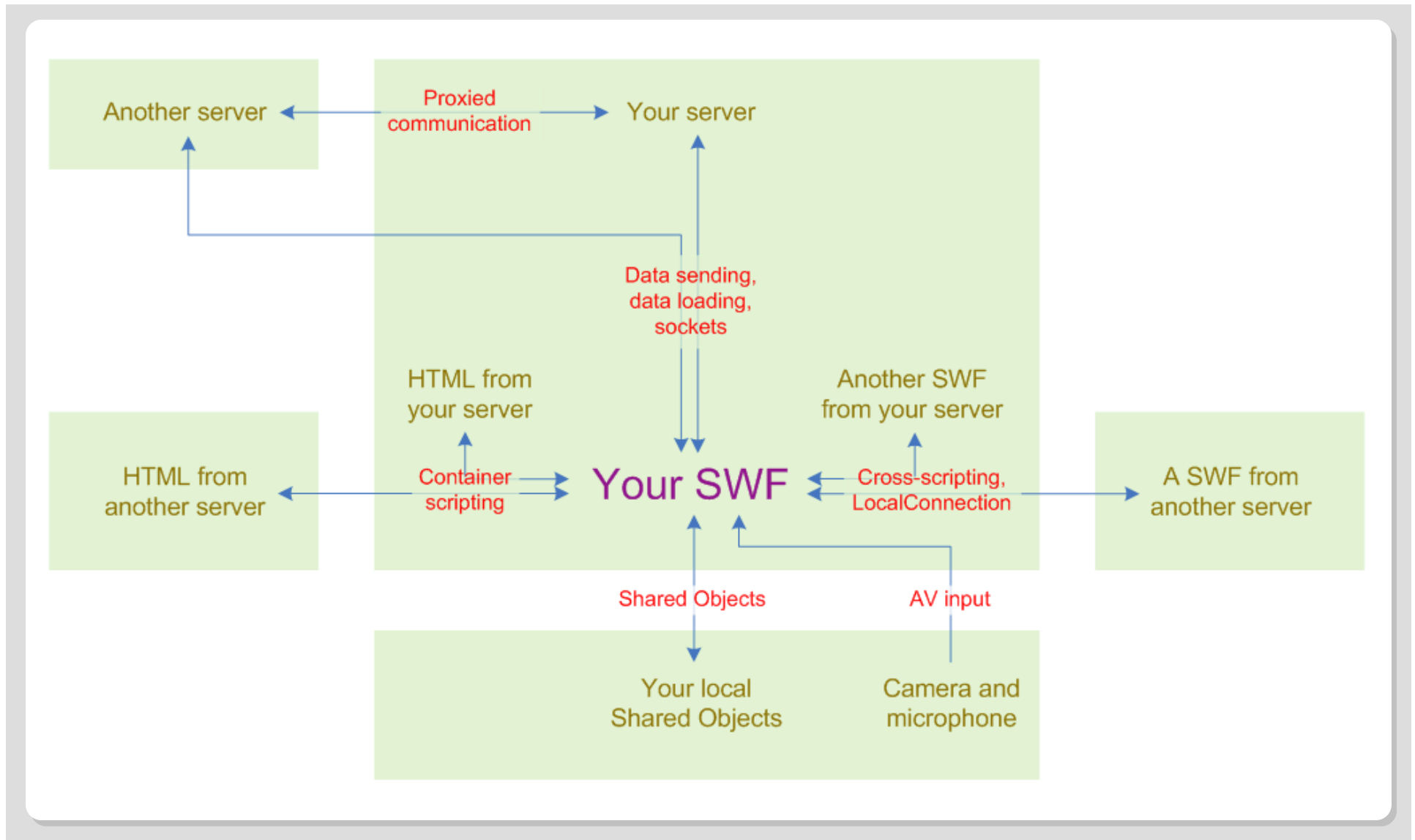
- “How can I make my Flash application secure?”
- “What is this security @\$#! doing in my way, and how can I get rid of it?”

Both Questions Have the Same Answer



- The Flash Player security model tells you which scripts can do what, and why
- There are a lot of details, but the big picture isn't too bad

Most of the Important Stuff



Today's Topics

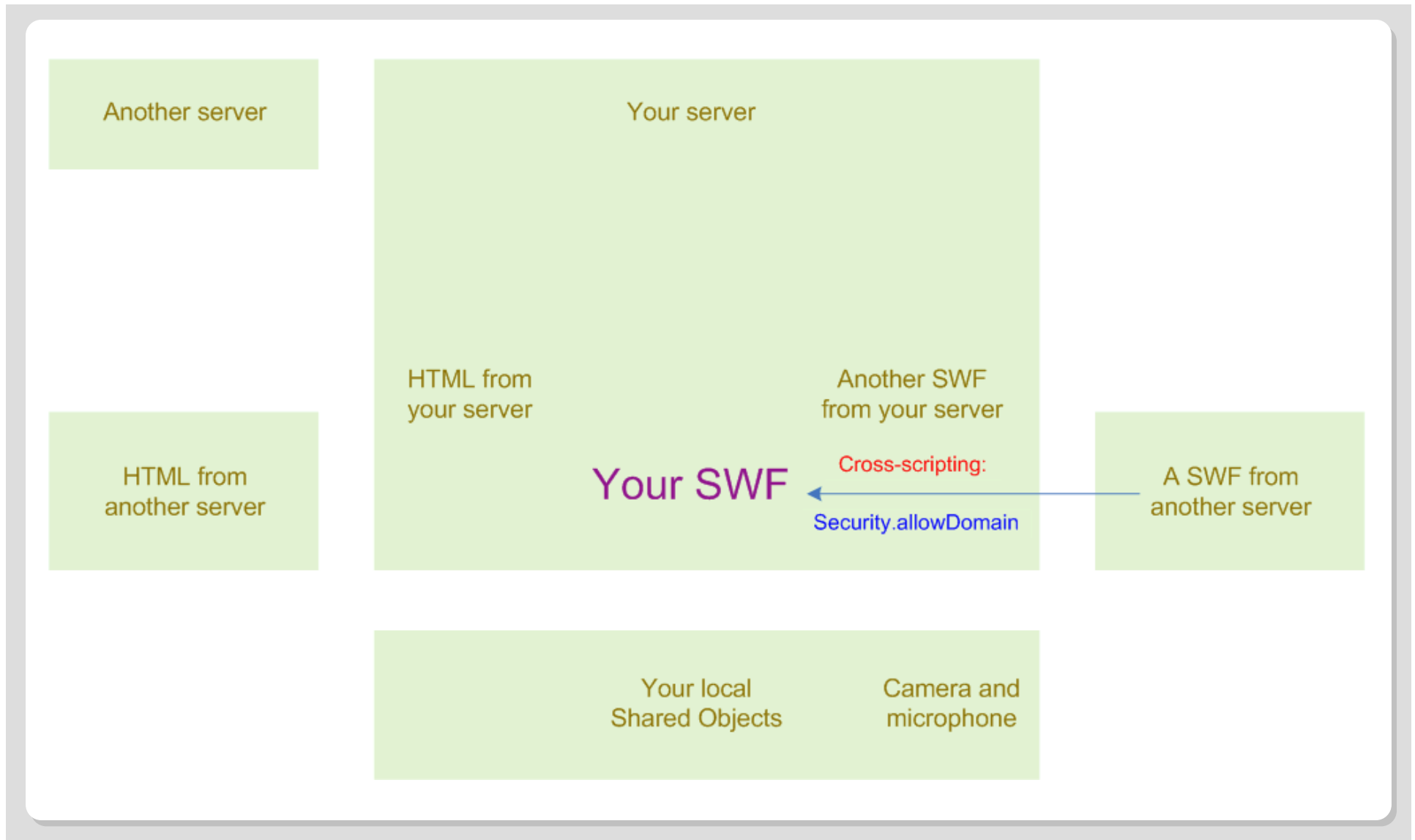


- Part 1: Client-Side Scripting
- Part 2: Client-Server Interaction
- Part 3: Local SWFs

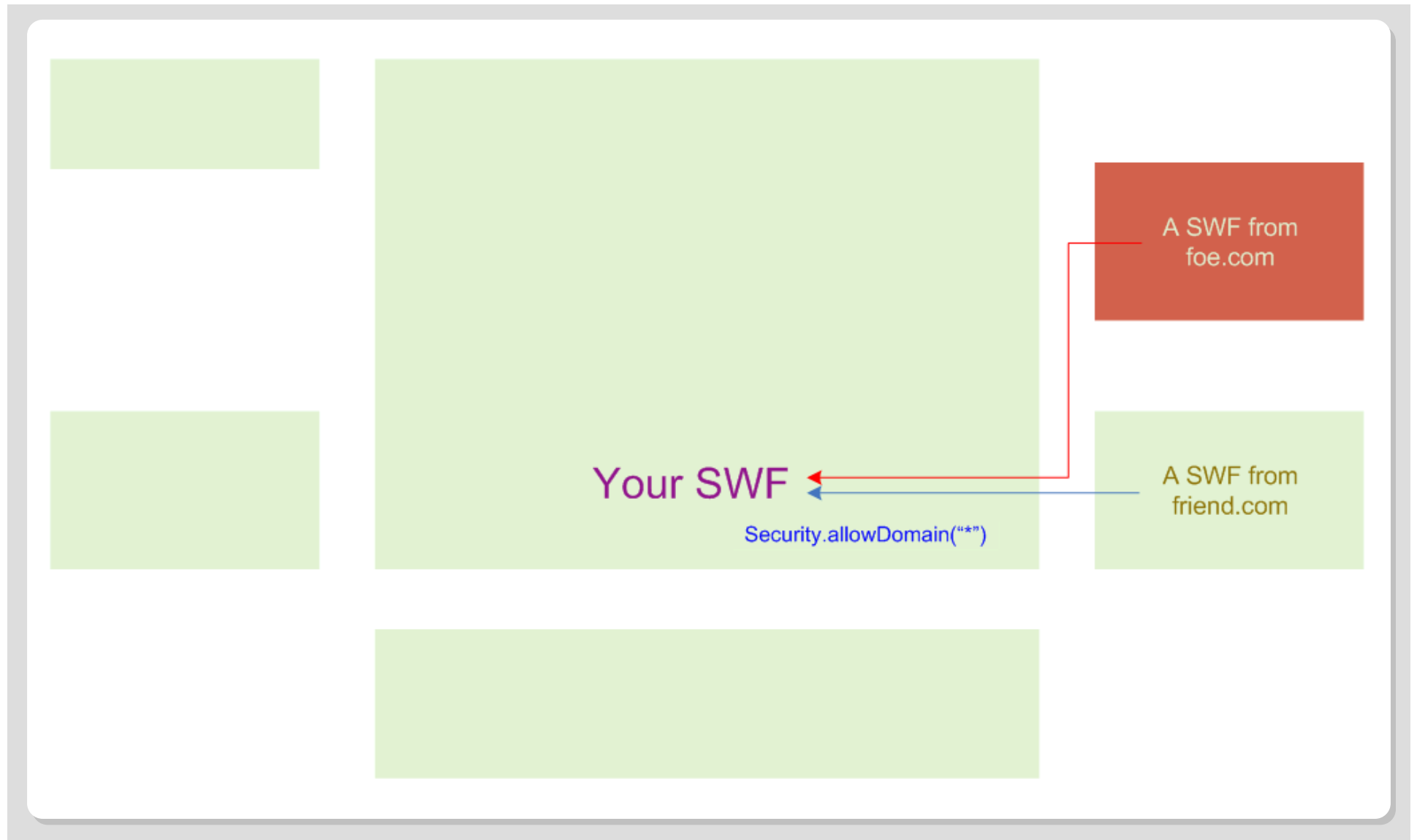
Part 1:

Client-Side Scripting

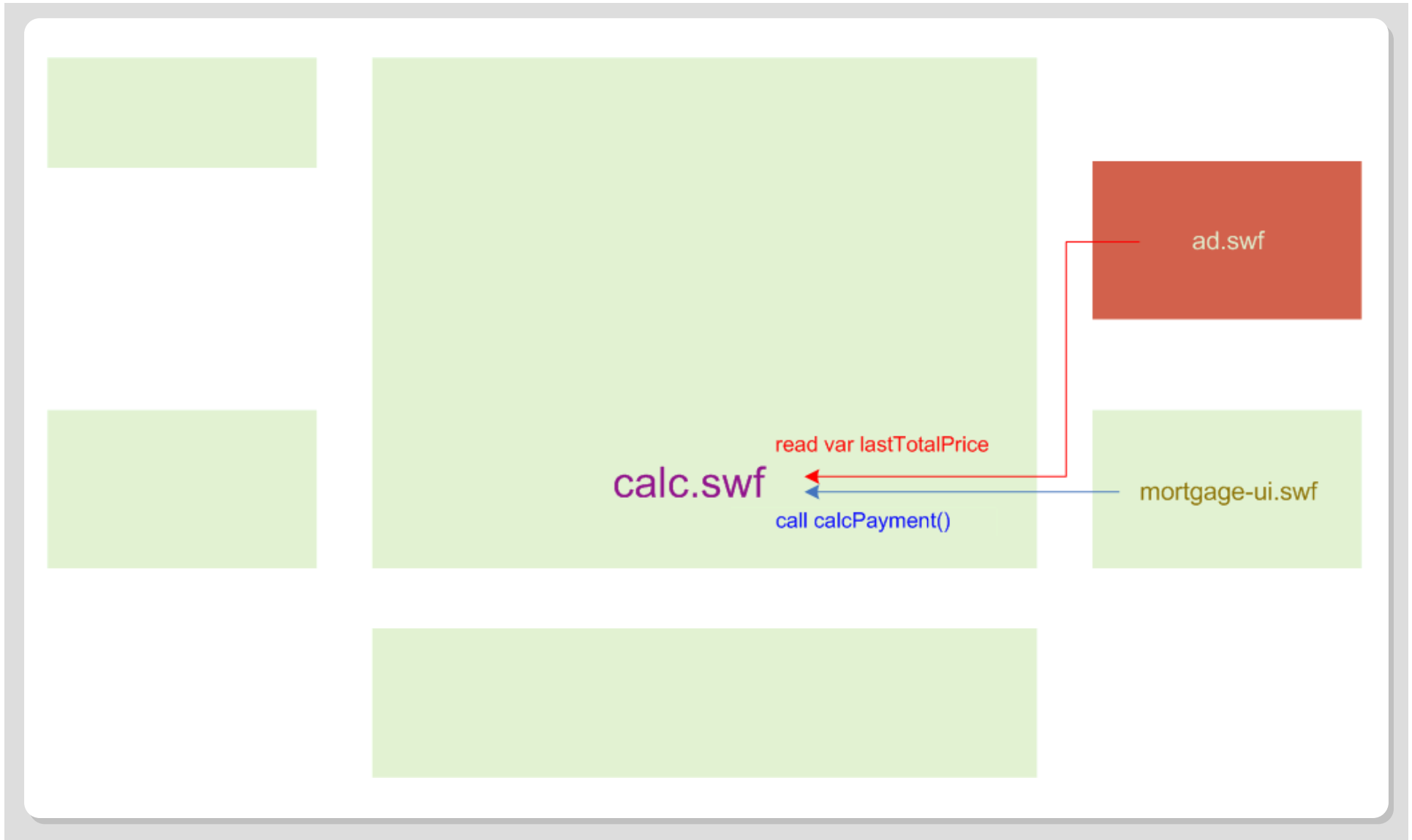
Cross-Scripting



Cross-Scripting: Wrong Way



Cross-Scripting Vulnerability

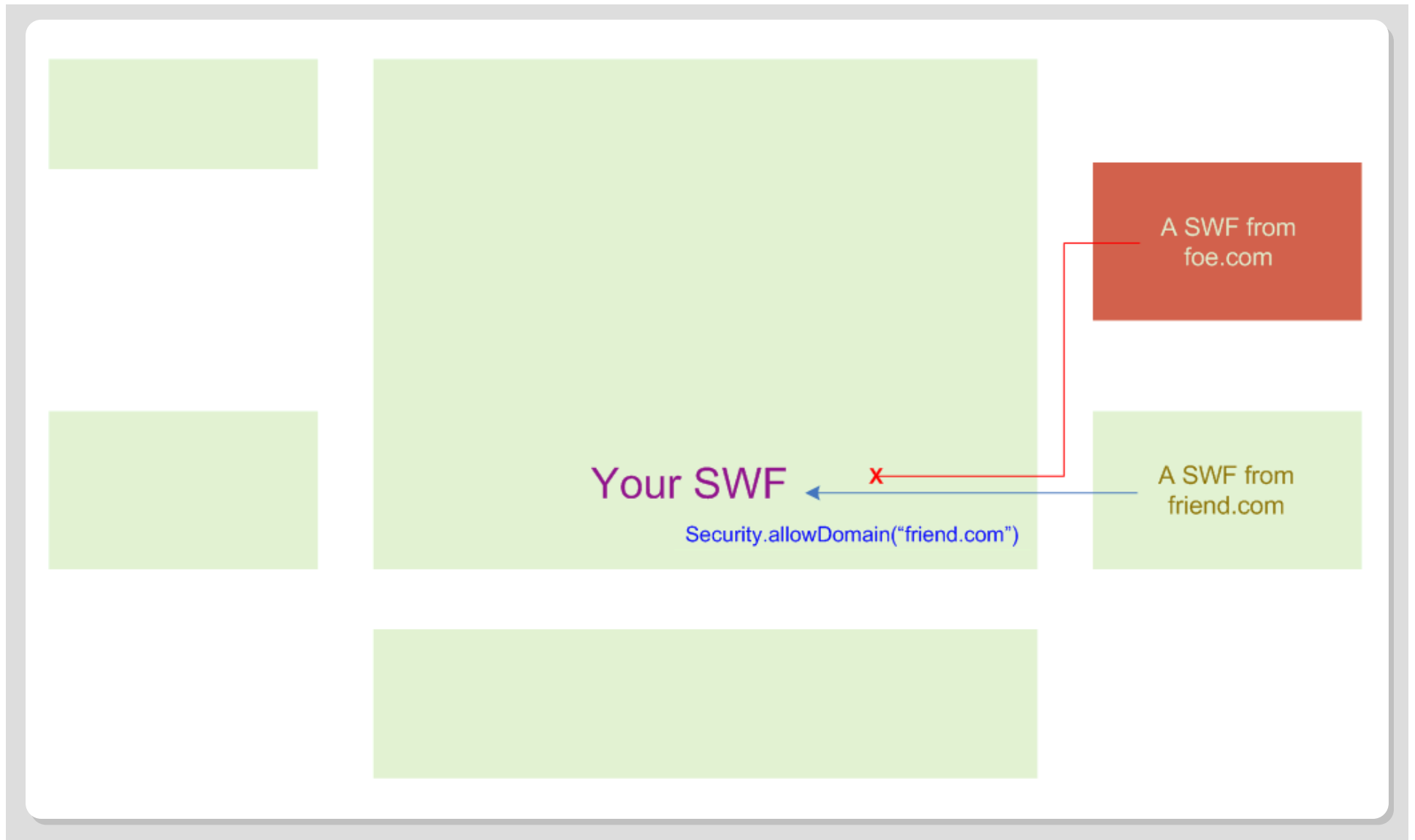


Cross-Scripting Vulnerabilities Matter!



- Internet crime is becoming big business, and there are many flavors
- Users are entrusting your content with data that may be sensitive
 - Obviously sensitive: passwords, financial data
 - More subtle: email addresses, personally identifying data, ethnicity, interests, ...
 - Some leaks might seem minor, but they can add up – e.g. scams, stalking, identity theft
 - Tampering matters to users too – purchase fraud, harassment, even game scores
- Users are not the only potential victims – your own site can be attacked
 - Purchase fraud, community disruption, UI defacement, statistical fraud, ...
- Remember: *any* SWF can load your SWF and try to script it!
 - The SWFs *you* load are the ones to be most careful of, because they appear on your site
 - But your SWF served from a different site can still look convincing

Cross-Scripting: Right Way

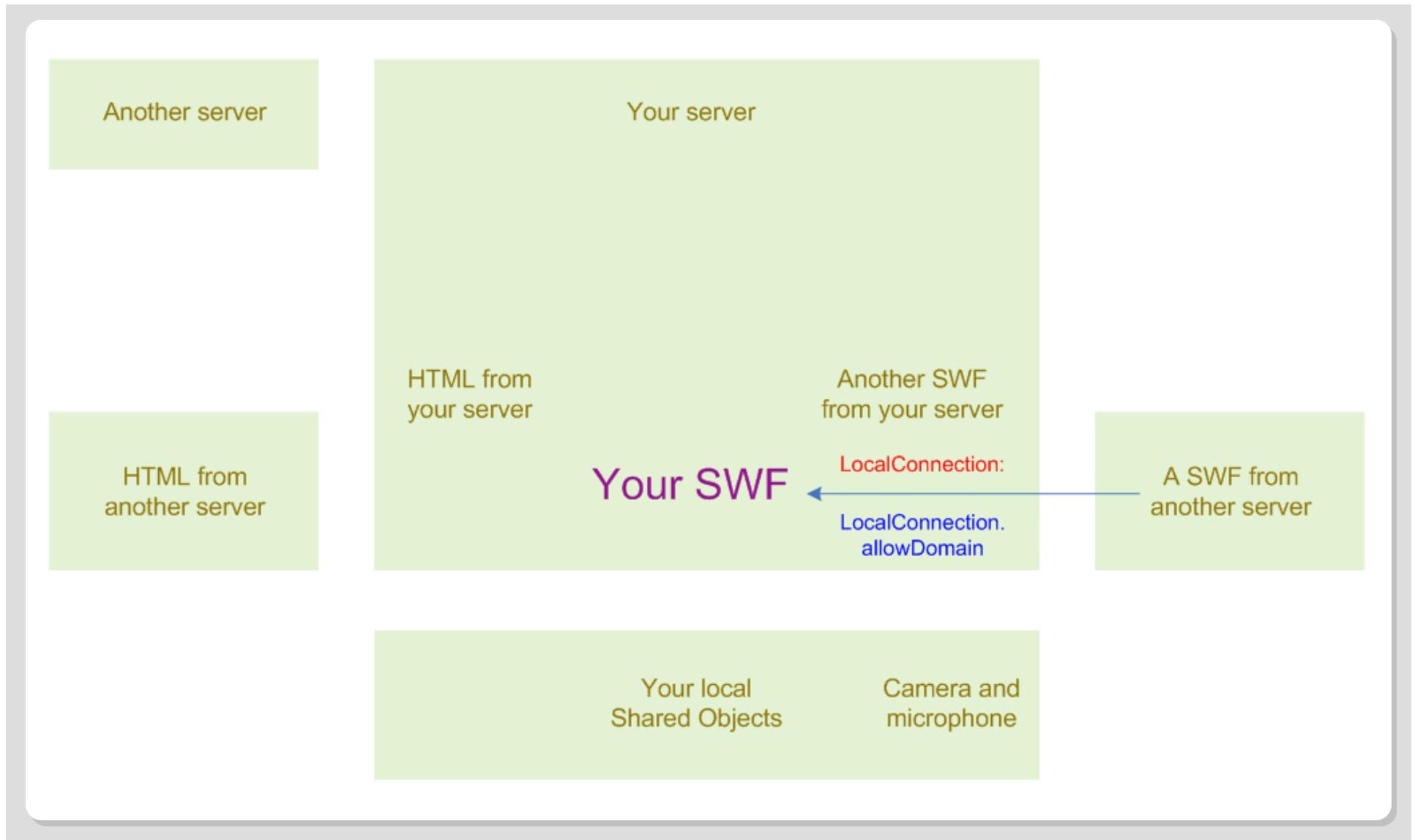


So...

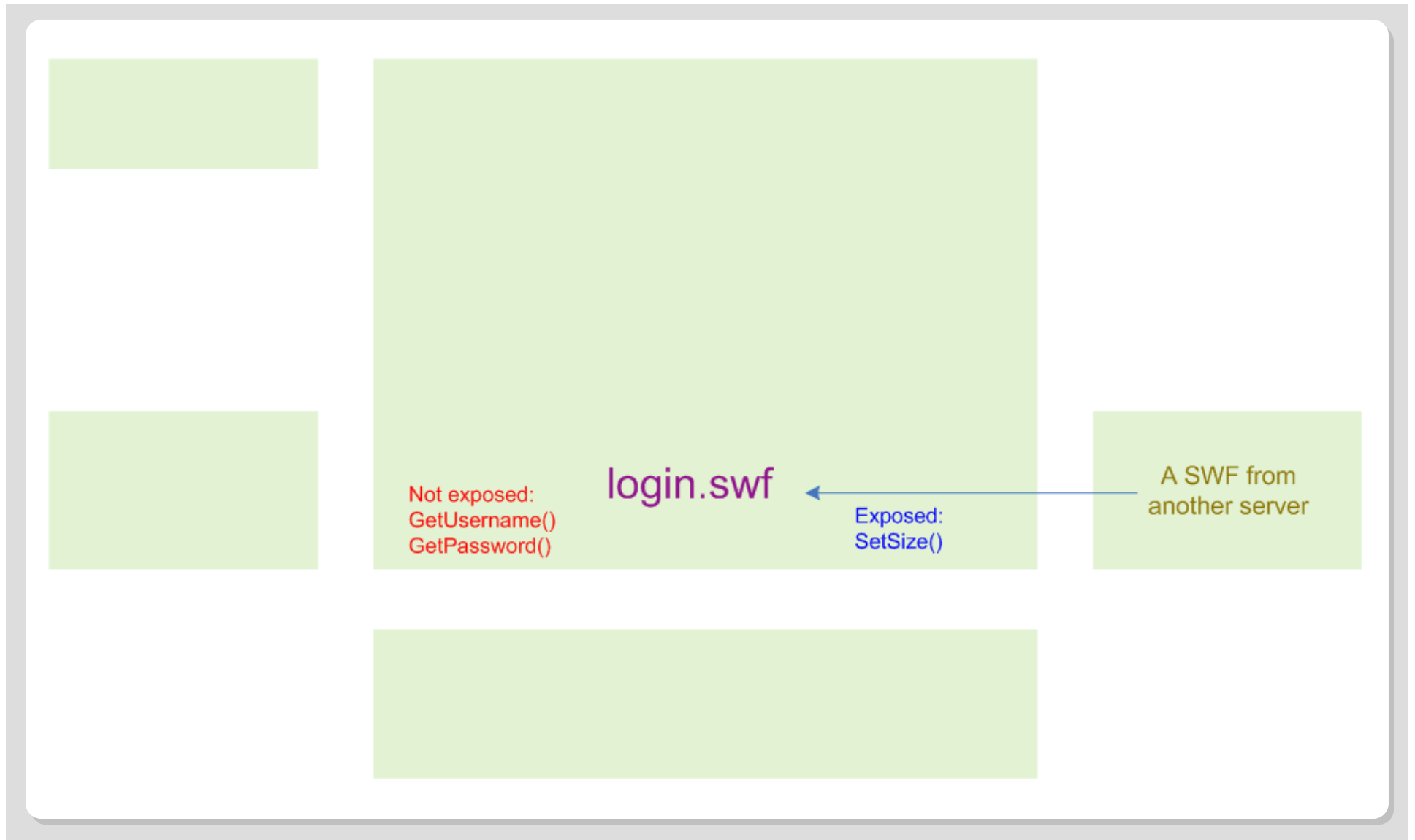
MAX

Rule 1: Use least privilege.

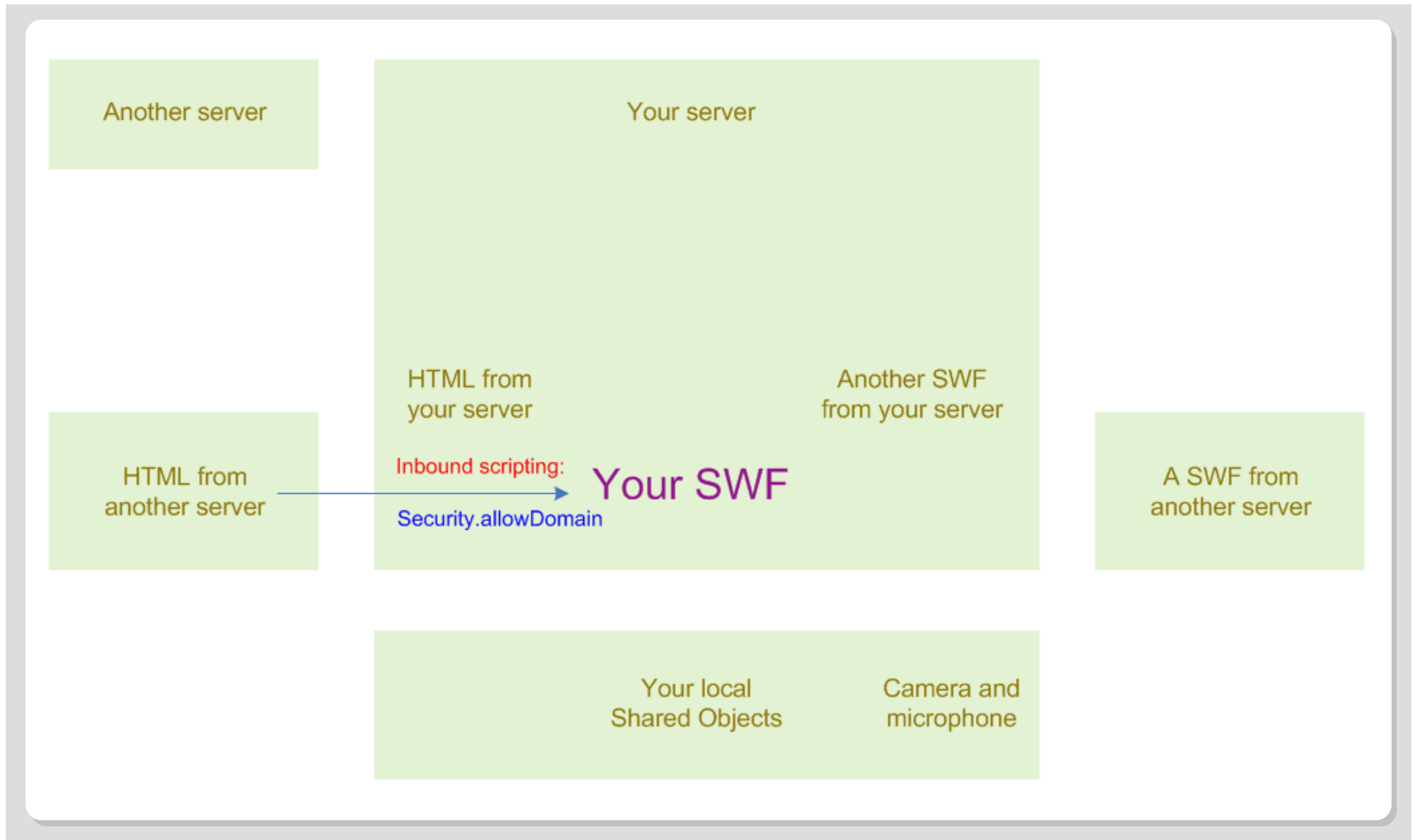
LocalConnection: a Narrow Interface



LocalConnection Example



Inbound Container Scripting



Input Validation



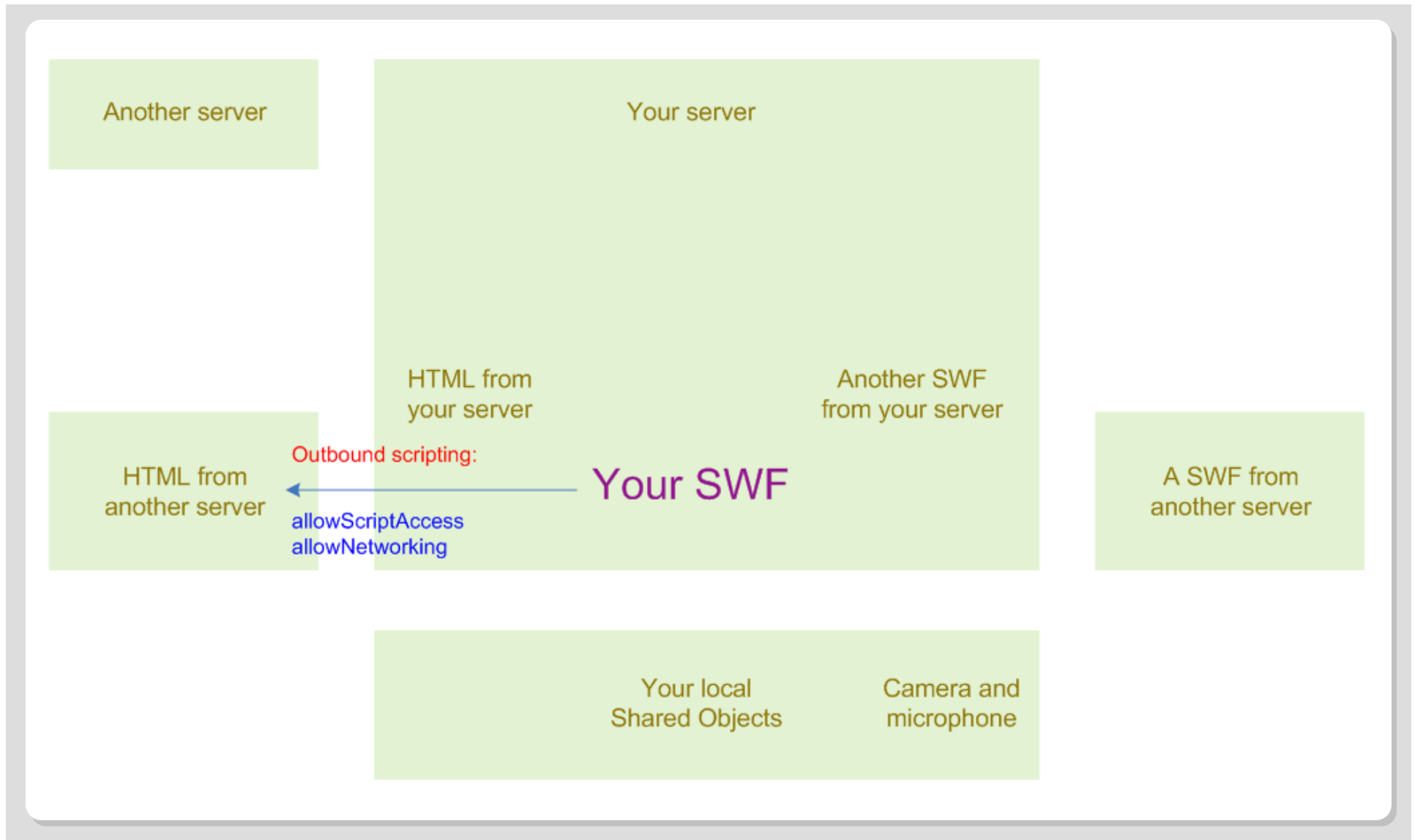
- Security.allowDomain governs which HTML files can script your SWF
- However, one thing not filtered is *parameters* (query string / FlashVars)
- Parameters are an example of untrustworthy input
 - Others: user input, data from servers
- If you take any action based on input, or pass input along, validate it!
 - Example: accept an URL as a parameter specifying an image to display
 - Without validation: your UI may now be a picture frame for porn
 - With validation: check that the URL is on your site, or build the URL yourself from a name
- Validation is even more important on the server
 - Can't ever know whether or not it's your own content that's talking to you
 - Common strategy: two-stage validation: on client for UI convenience, on server for security

So...

MAX

Rule 2: Validate input.

Outbound Container Scripting

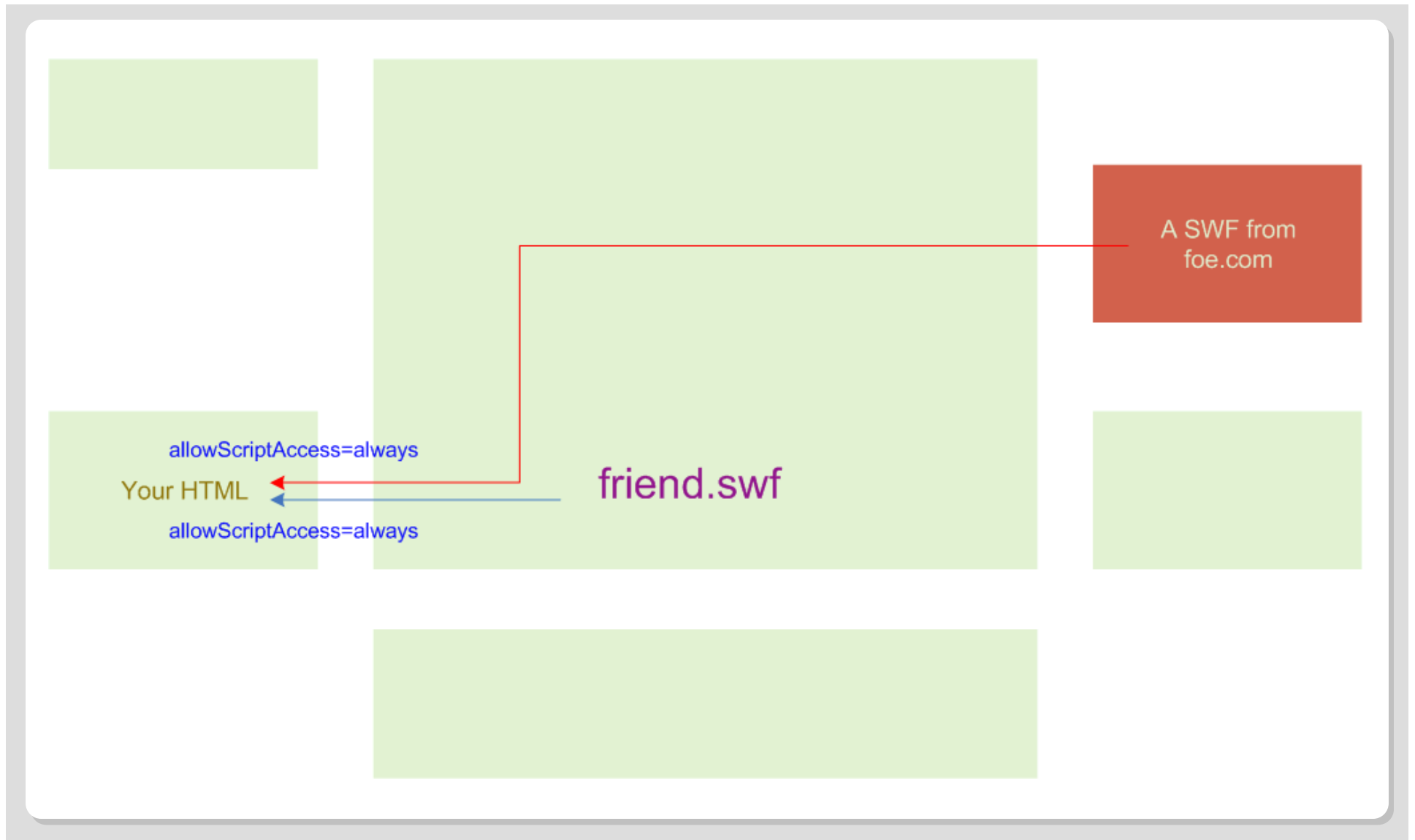


Outbound Scripting Permissions

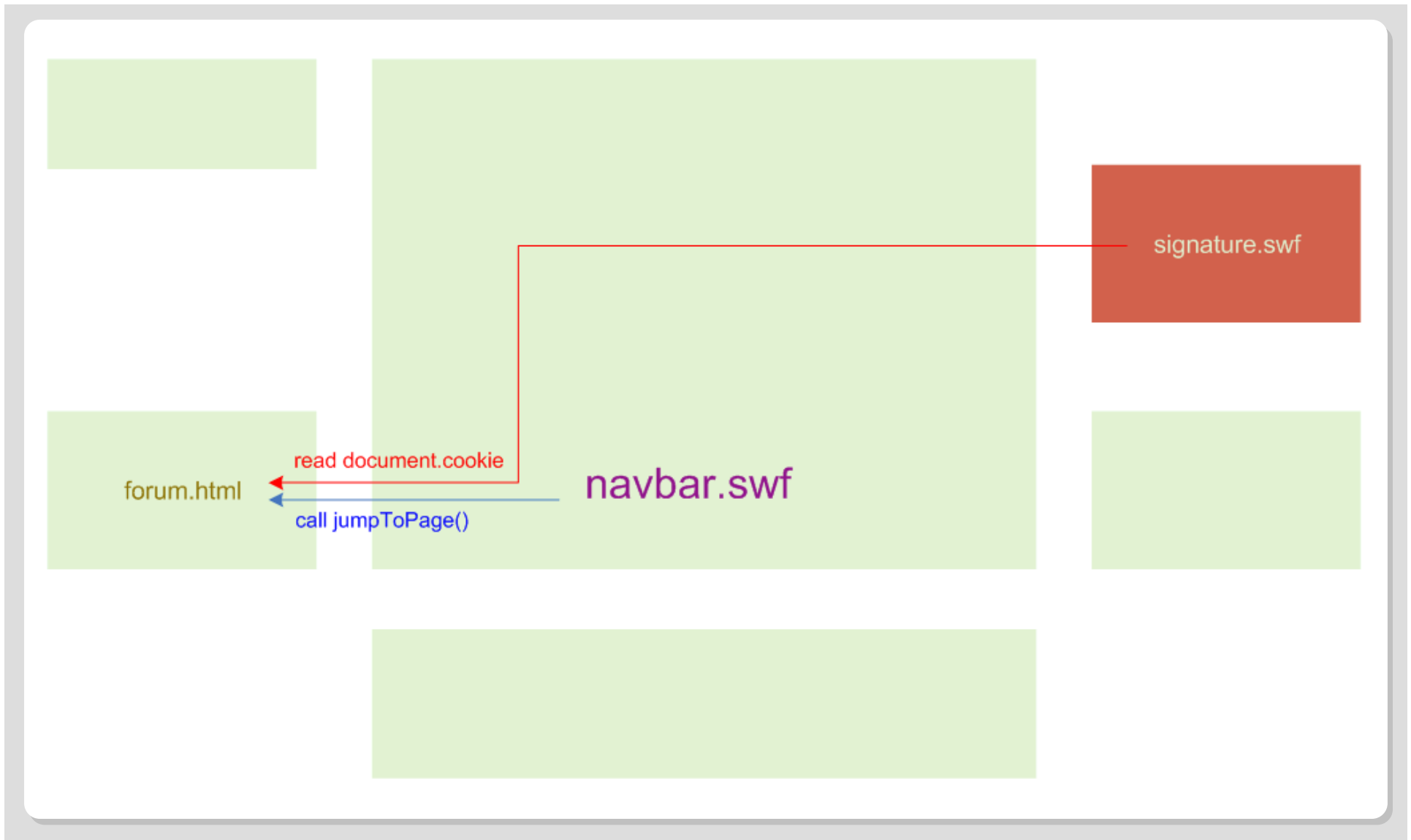


Permission	Outbound scripting?	Browser URL navigation?	Networking?
allowScriptAccess = always	✓	✓	✓
allowScriptAccess = sameDomain	if SWF and HTML domains match	✓	✓
allowScriptAccess = never	✗	✓	✓
allowNetworking = internal	✗	✗	✓
allowNetworking = none	✗	✗	✗

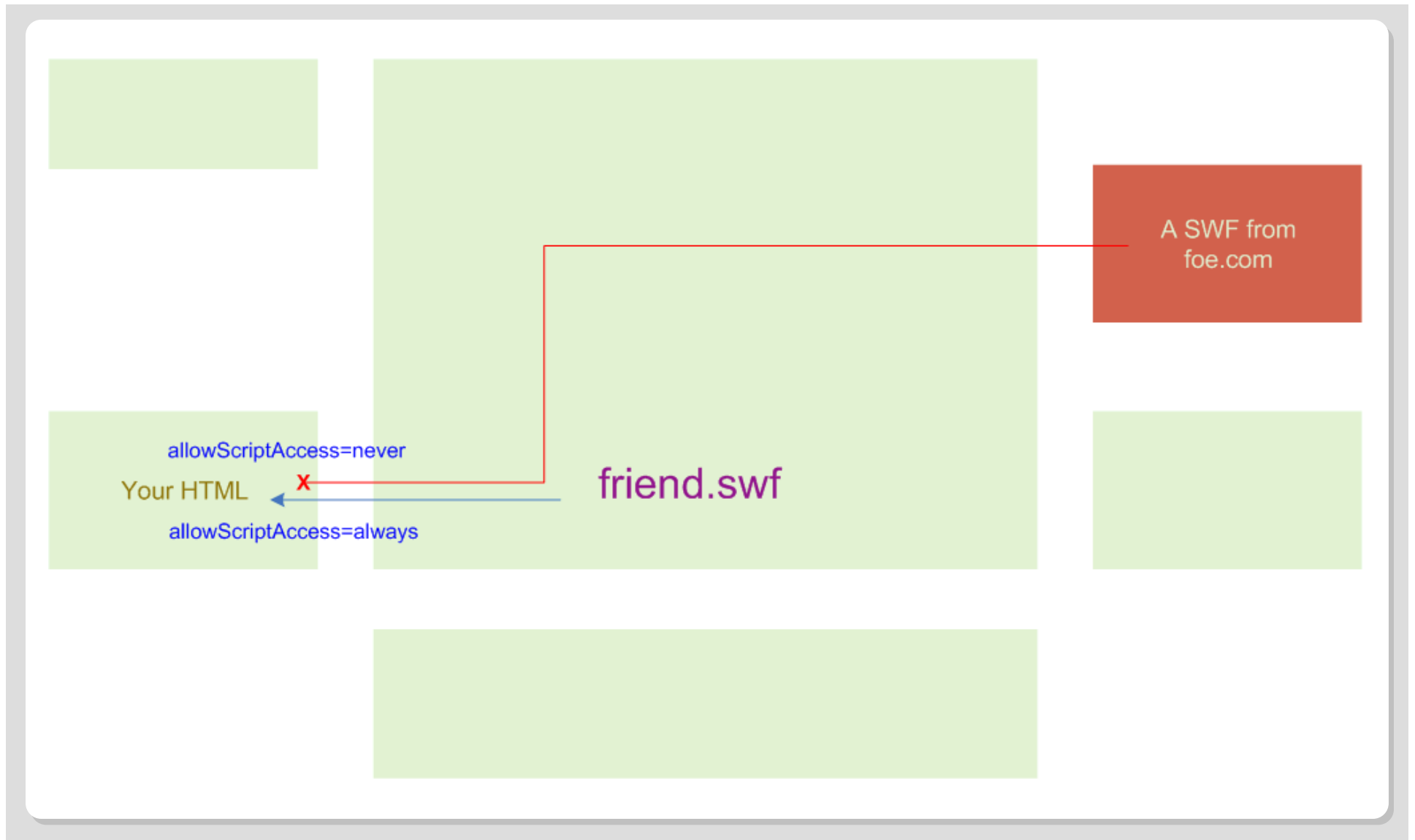
Outbound Scripting: Wrong Way



Outbound Scripting Vulnerability



Outbound Scripting: Right Way



Other Secured Client-Side Resources



- Camera and microphone input
 - User must interactively authorize
- Local shared objects
 - Like HTML cookies, but larger and more structured –persistent data associated with a SWF
 - By default, each SWF URL gets its own "jar" of shared objects that don't overlap with others
 - You can elect (using the localPath parameter to getLocal()) to use a jar that is more general, applying to some shorter prefix of your URL – this allows you to share objects between SWFs – the limit to this generalization is the root of your domain
 - Local shared objects aren't fully reliable or always available
 - Users may delete them or limit their size
 - Users may decide not to permit shared objects for SWFs from domains other than the one that is showing the browser's address bar ("third-party data", a tracking concern)

Nonexistent Flash Player Features

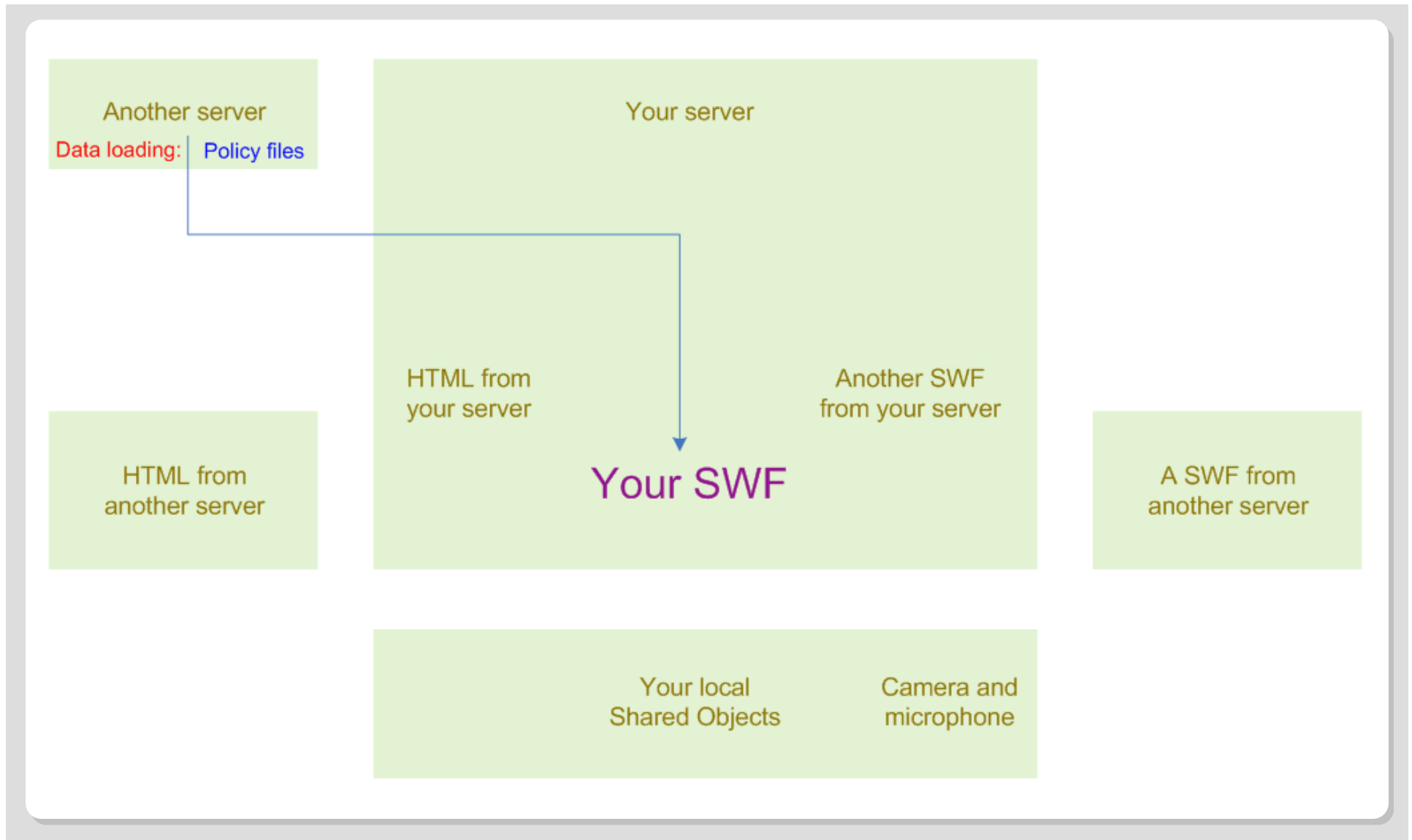


- Much of Flash Player's security comes from simply not implementing certain dangerous abilities
 - Writing files to disk
 - Reading directory contents
 - Invoking local programs
- Many of these features are present in Adobe AIR, which allows Flash content to behave more like a local application

Part 2:

Client-Server Interaction

Data Loading



Data Loading Operations



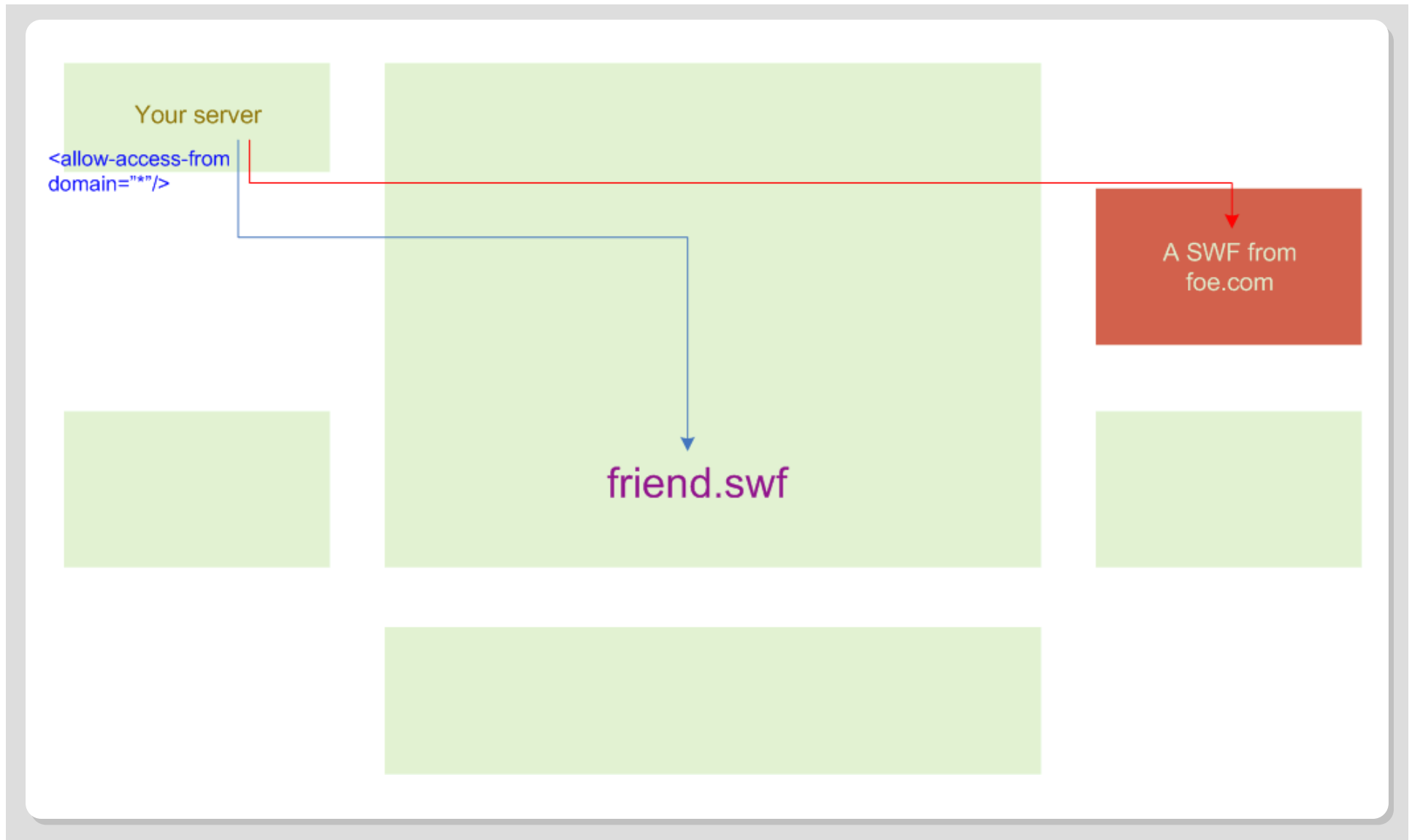
- Protocols
 - HTTP, HTTPS
 - FTP
- AS3
 - Direct loading (async): `URLLoader.load()`, `URLStream.load()`
 - Image extraction (sync): `Loader.content`, `BitmapData.draw()`
 - Sound data extraction (sync): `Sound.id3`, `SoundMixer.computeSpectrum()`
- AS1 / AS2
 - Direct loading (async): `loadVariables()`, `LoadVars.load()`, `XML.load()`, ...
 - Image extraction (sync): `BitmapData.draw()`
 - Sound data extraction (sync): `Sound.id3`

Why Does Data Loading Require Permission?

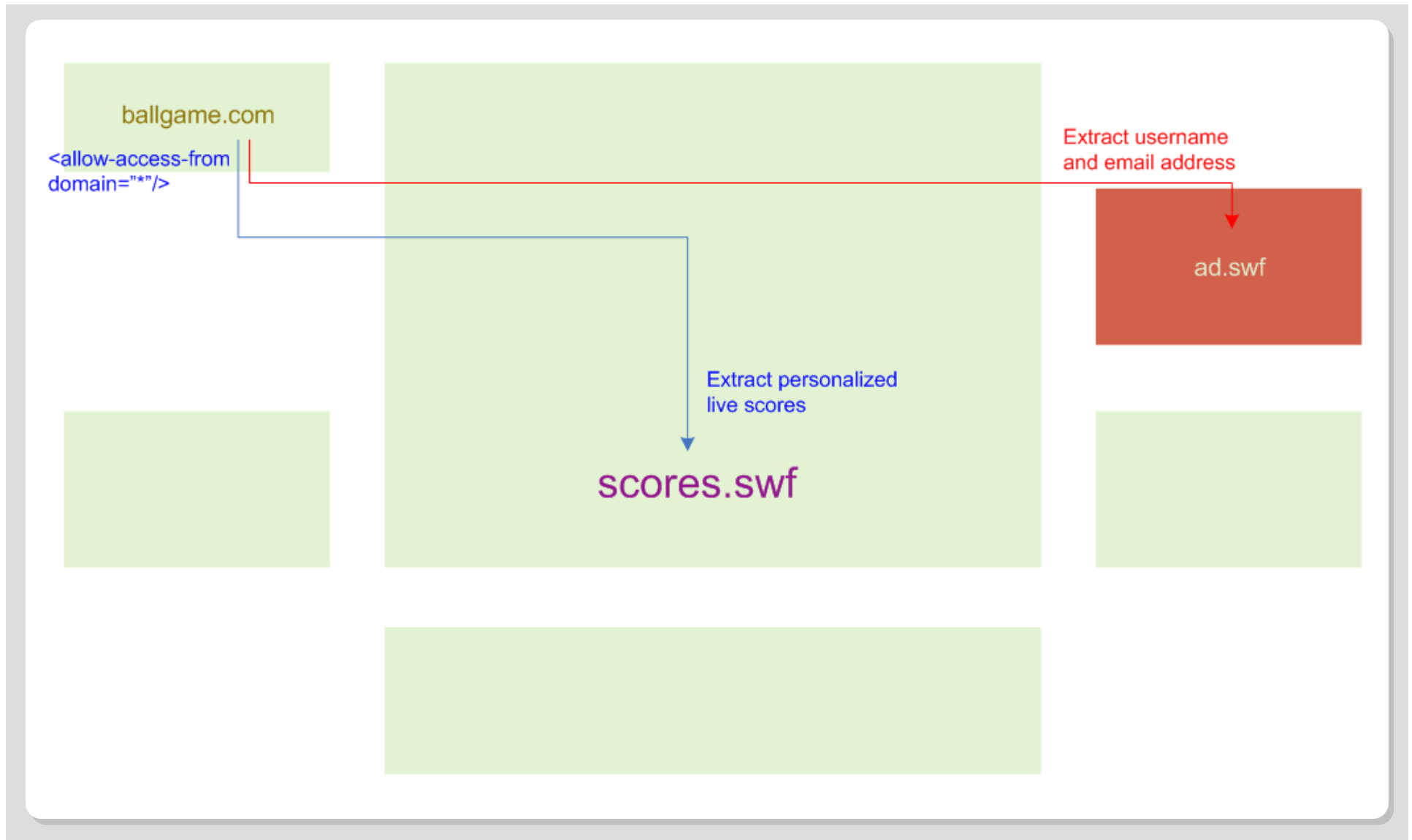


- Isn't this just public data from the Internet that we're loading?
 - Not necessarily – all Flash Player knows is that it's data the *user* can access
 - Might be from a private server behind a firewall (e.g. corporate secrets, personal files)
 - Might be from a public server, but require a user login (e.g. online financial data)
- Flash is currently the only client to allow direct cross-domain data loading
 - Browsers are rumored to be looking into similar abilities, but don't yet allow it
 - There's a reason browsers don't allow it yet: it's hard to secure!

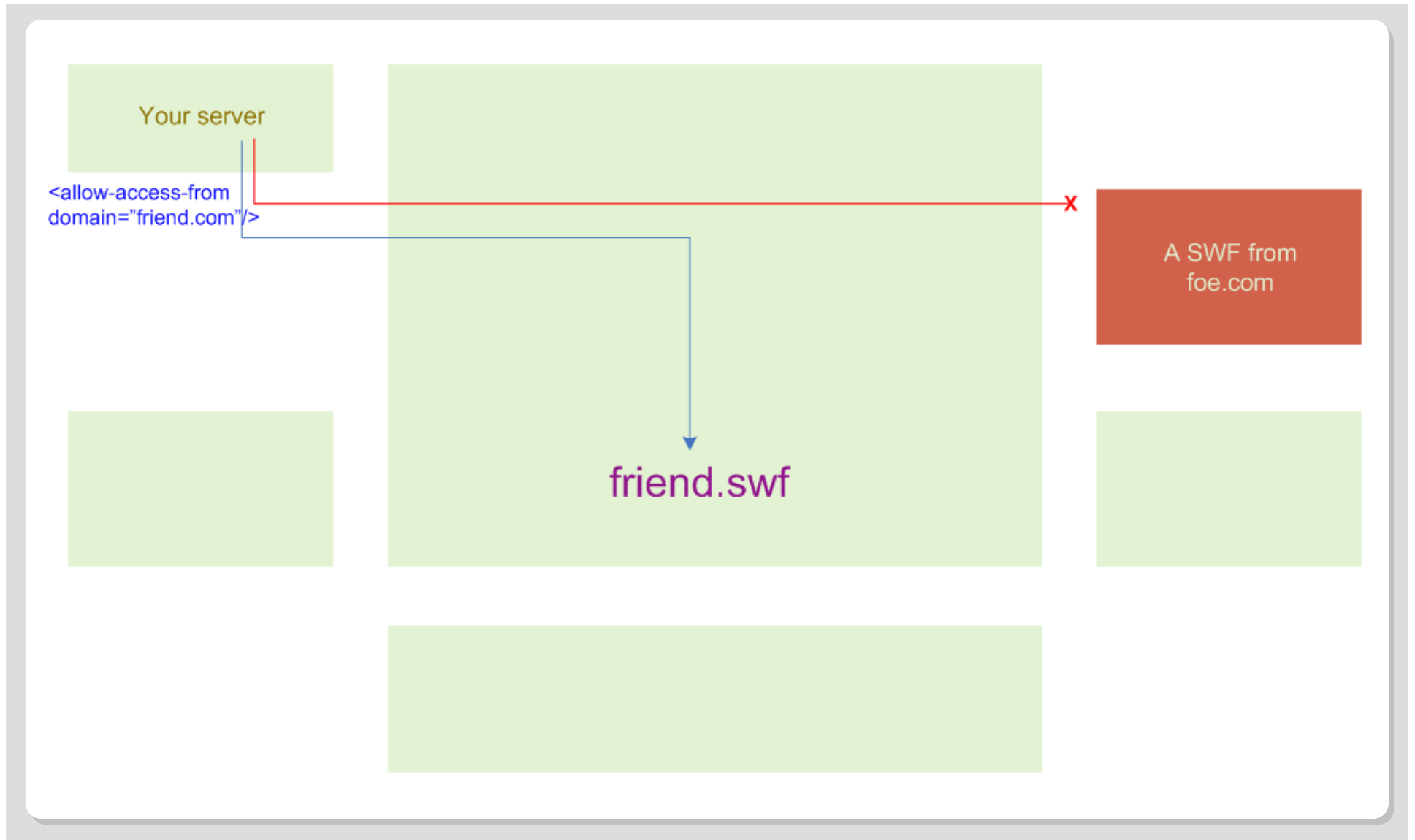
Data Loading: Wrong Way



Data Loading Vulnerability



Data Loading: Right Way



Policy File Syntax



- The *contents* of a policy file determine *who* may access data
- To permit access by SWFs from all locations:

```
<cross-domain-policy>  
  <allow-access-from domain="*" />  
</cross-domain-policy>
```

- To permit access only by SWFs from certain domains:

```
<cross-domain-policy>  
  <allow-access-from domain="friend.com" />  
  <allow-access-from domain="www.friend.com" />  
  <allow-access-from domain="*.anotherfriend.com" />  
  <allow-access-from domain="42.127.12.88" />  
</cross-domain-policy>
```

- Flash Player interprets domains by name, not by IP address
 - So you should list every way you expect an accessing SWF's domain to be specified
 - The partial-wildcard syntax "*.domain.com" is useful for this purpose
 - You can list literal IP addresses, but this only covers SWFs loaded explicitly by IP address

Policy File Scope



- The *location* of a policy file determines *what* may be accessed
- The default location, which affects the entire server, is `/crossdomain.xml`
 - Flash Player checks this location automatically
- Any other location gives access only to files at the same level or deeper
 - For example, `http://foo.com/subdir/crossdomain.xml`
 - Gives access to `http://foo.com/subdir/file.txt`
 - Gives access to `http://foo.com/subdir/deeper/path/something.cgi`
 - Does not give access to `http://foo.com/elsewhere/wally.xml`

When Is It Safe to Allow Access With a Policy File?



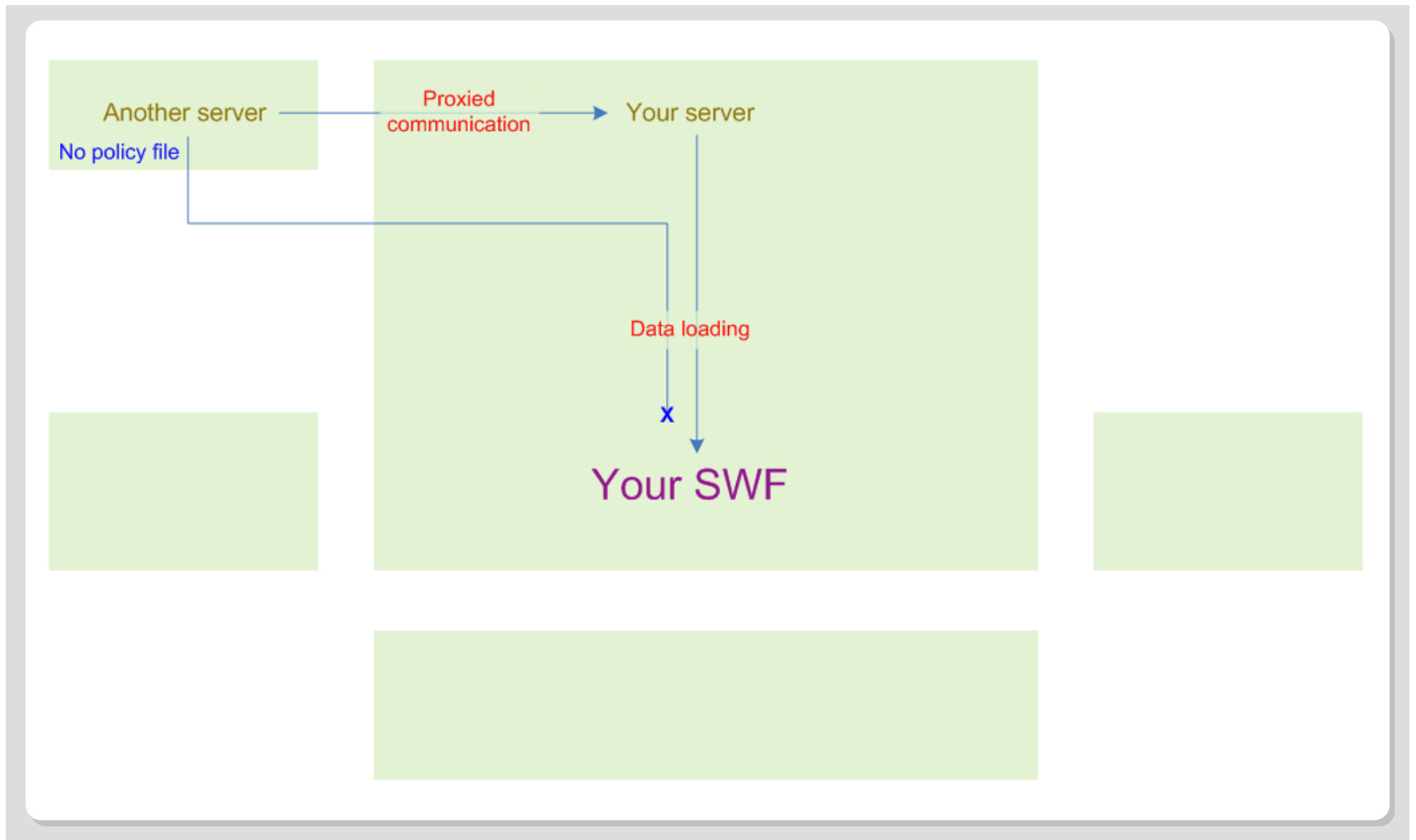
- If your *entire server* serves only completely public data, "*" at root is fine
 - If you handle user logins, "*" is usually a bad idea
 - If you are behind a firewall, "*" is usually is a bad idea
- If you only need to give access to some data, don't use a root policy file
 - Put a policy file at a lower spot in your directory hierarchy
- If you only need to permit access by certain sites, enumerate them
 - Rather than allowing "*"
- If you're not sure, you can always deny access and require proxying!

Using Policy Files: Client Side



- To use the default location, `/crossdomain.xml`, no need to do anything
 - Flash Player automatically checks this location when you request data from another domain
- To use a non-default location, call `Security.loadPolicyFile("policyFileURL")`
 - AS1 / AS2: `System.security.loadPolicyFile`
 - Call *before* making any data-loading requests that depend on that policy file
- To use any policy file for a *synchronous* operation (data extraction):
 - Call `loadPolicyFile()` if the policy file isn't in the default location
 - Set the `checkPolicyFile` flag when you first load the media file
 - `LoaderContext.checkPolicyFile`, `Sound.checkPolicyFile`, `NetStream.checkPolicyFile`, ...
 - This tells the player to resolve policy files before loading media
 - Perform your synchronous operation
 - If you've set `checkPolicyFile`, the player can authorize access synchronously
 - Otherwise, you'll get an exception

Proxying When You Can't Get Client Access

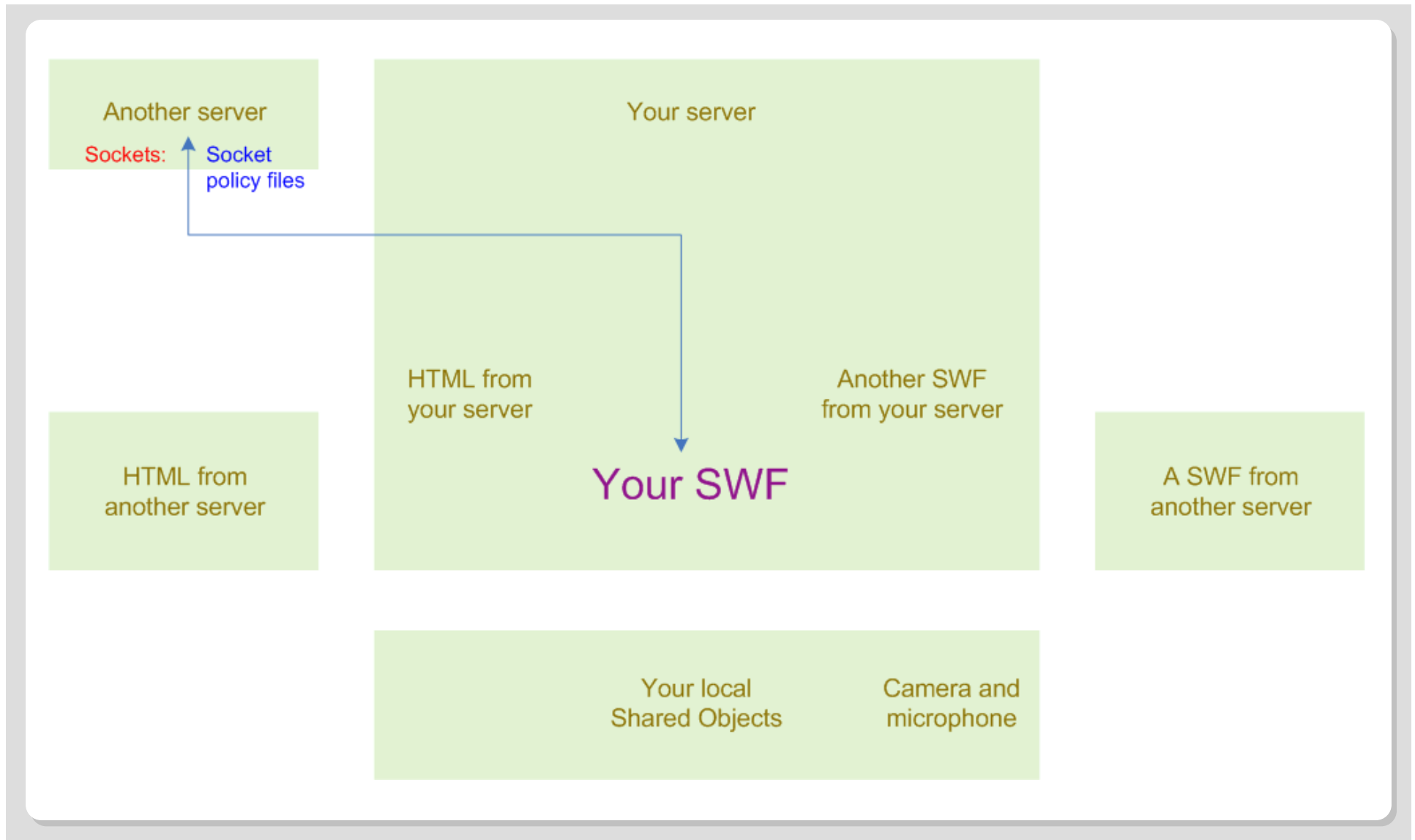


A Brief Detour Into Server-Side Security



- Let's say you're writing a proxying script as shown in the previous slide
- Remember that *anyone* can connect to this script and request *anything*
 - Problems: liability for illegal data, extra bandwidth charges, snooping inside your server farm, ...
- You only want to allow your *own* SWFs to use the script
 - Partial solution: check Referer header and verify it's from your own domain
 - This problem can't be completely solved
- You should be able to determine *what* the client will need
 - Don't accept arbitrary URLs; accept resource names and map them to URLs on the server
 - Validate your input, and accept only what you expect to be requested
- Server security is never done, and you can go further if there are problems
 - Watch total usage, and investigate overages
 - Look for botnet machines that make loads of requests, and block them

Sockets



Socket Policy Files



- Similar to URL policy files, but transmitted directly over sockets
 - Player sends ASCII `<policy-file-request/>` plus null byte
 - Server responds by sending ASCII policy file plus null byte; player then disconnects
- Can choose a port two different ways
 - Use same port as main connection – requires socket server that understands policy files
 - Use another port with a dedicated policy server (hope to offer software on adobe.com soon)
 - Player checks same port by default
 - Clients use other ports with `LoadPolicyFile("xml socket: //server.com: 333")`
- Unlike URL policy files, scope of socket policy files is stated in file contents

```
<cross-domain-policy>  
  <allow-access-from domain="friend.com" to-ports="555,666,777-999" />  
</cross-domain-policy>
```

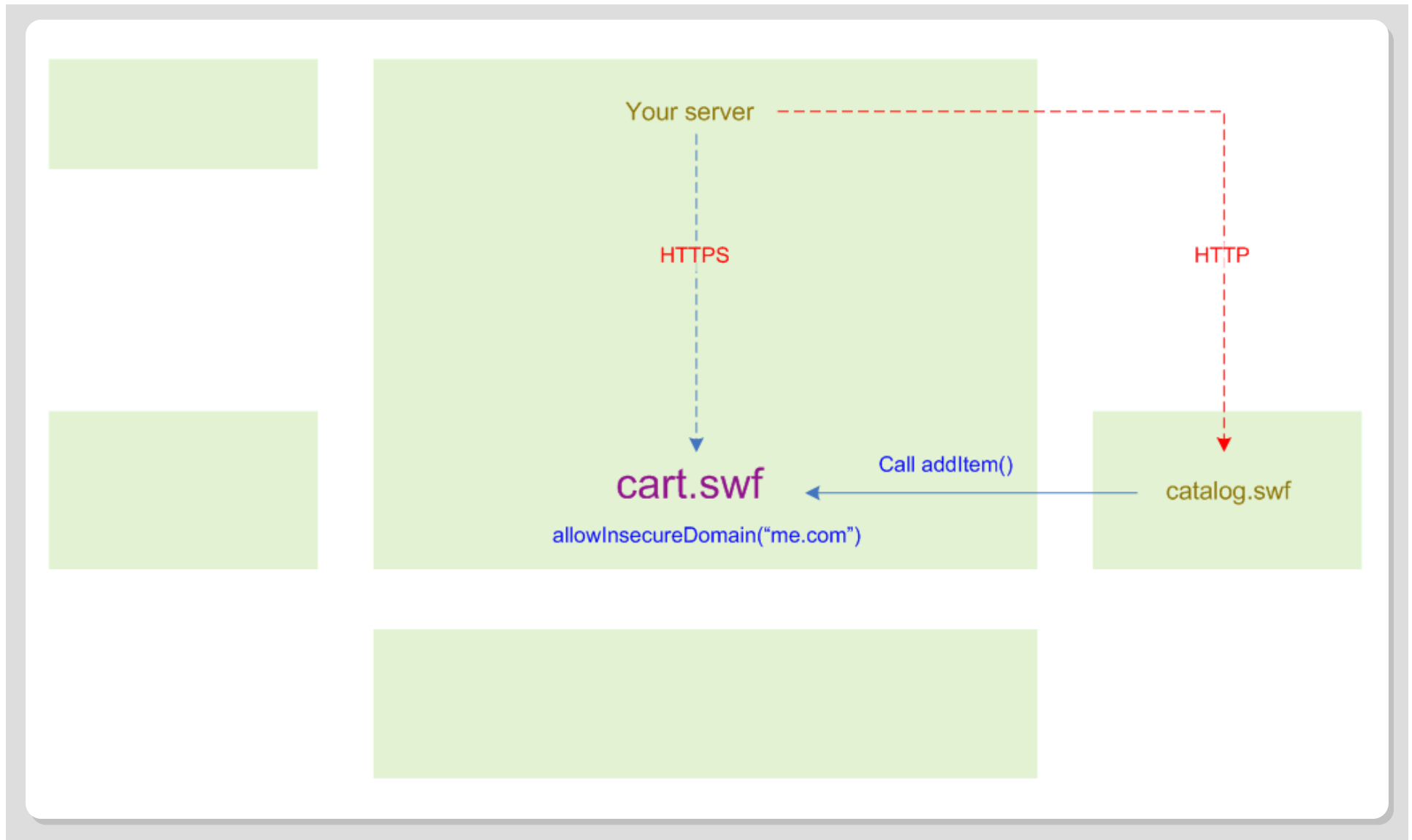
- Socket policy files from ports ≥ 1024 may only authorize access ≥ 1024

HTTPS: 3 Protections

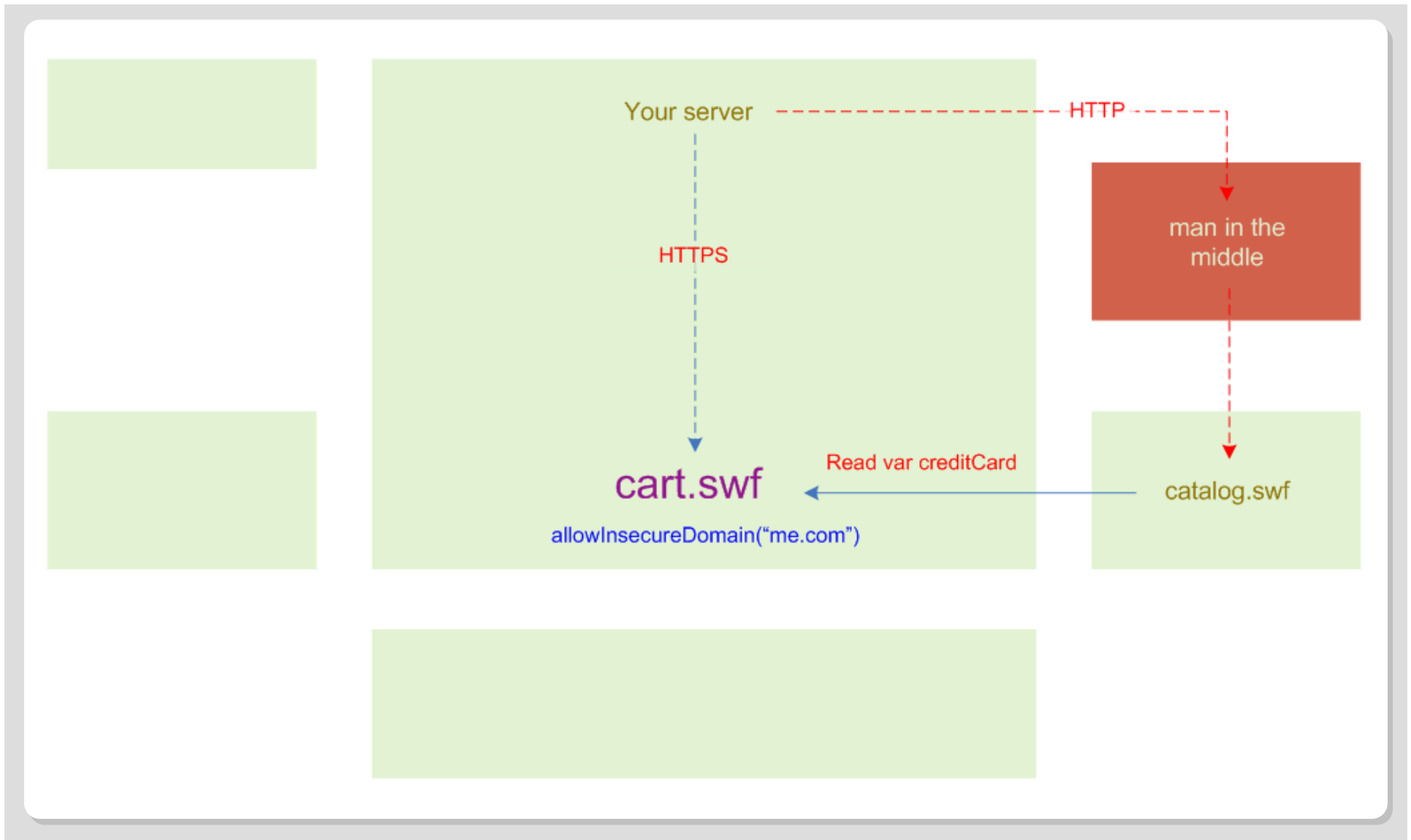


- Authentication: makes it harder for others to impersonate your site
- Encryption: makes it harder for eavesdroppers to observe transactions on the wire
- Tamper resistance: makes it harder for others to alter data in transit

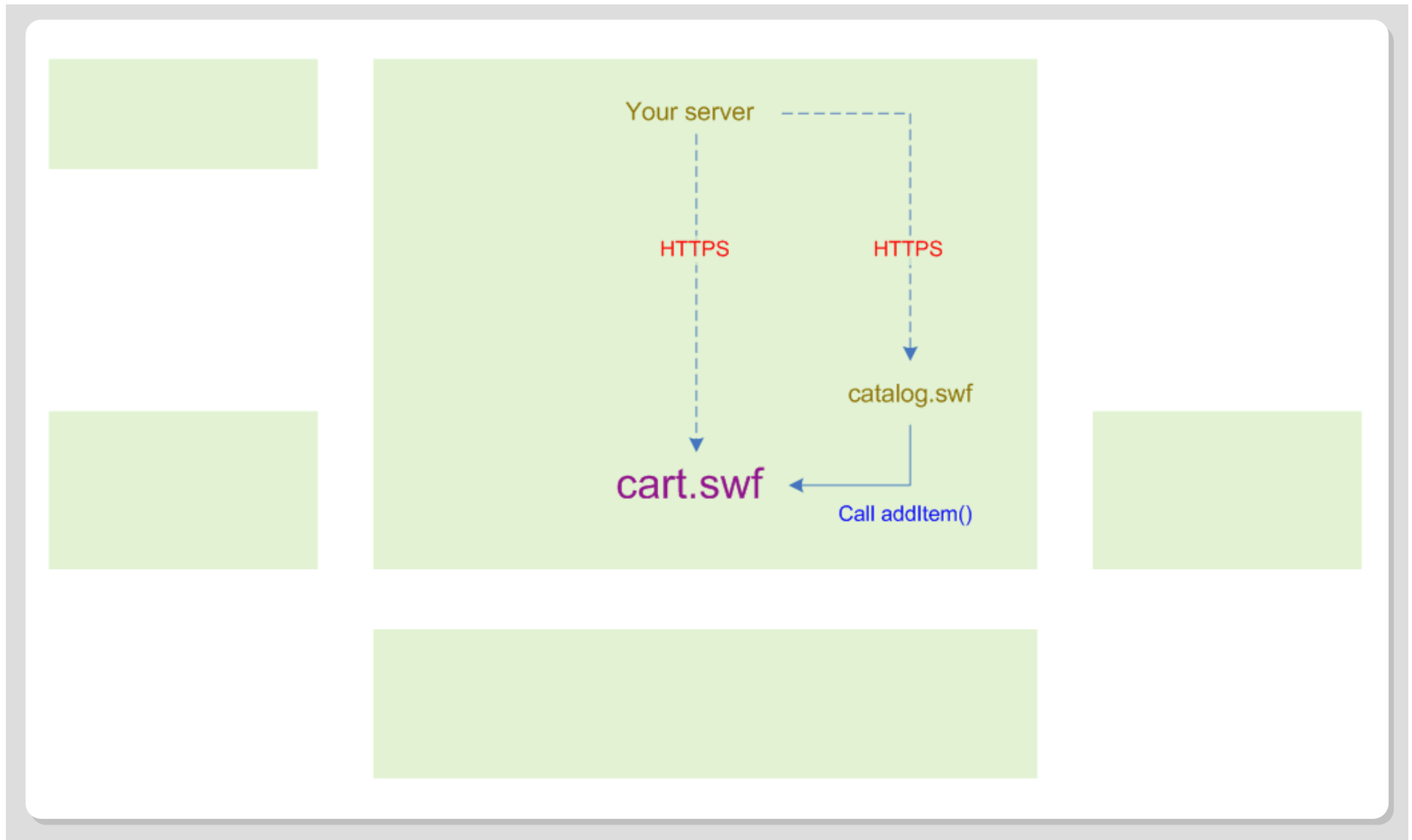
HTTPS: Wrong Way



HTTPS Vulnerability



HTTPS: Right Way



Avoiding HTTP/HTTPS Mixtures



- Flash Player places HTTP and HTTPS content into separate sandboxes, even when from the same domain
 - Applies to SWF and HTML content, and also to resources loaded as data
- HTTPS content may load or script HTTP content from its own domain
 - But it's not a good idea, since you can't trust what you're calling
- HTTP content may only load or script HTTPS content with permission
 - An even less good idea – avoid granting these permissions – deliberately scary names
 - Granting HTTP content permission into an HTTPS SWF: `allowInsecureDomain()`
 - Granting HTTP content permission to load HTTPS data: `<allow-access-from secure="false"/>`
- One more reason to avoid mixtures: browsers will complain about them
 - Padlock icon often won't display
 - Users may see "mixed-content" warnings

So...



Rule 3: Deploy HTTPS consistently.

Part 3:
Local SWFs

When Are Local SWFs Useful?



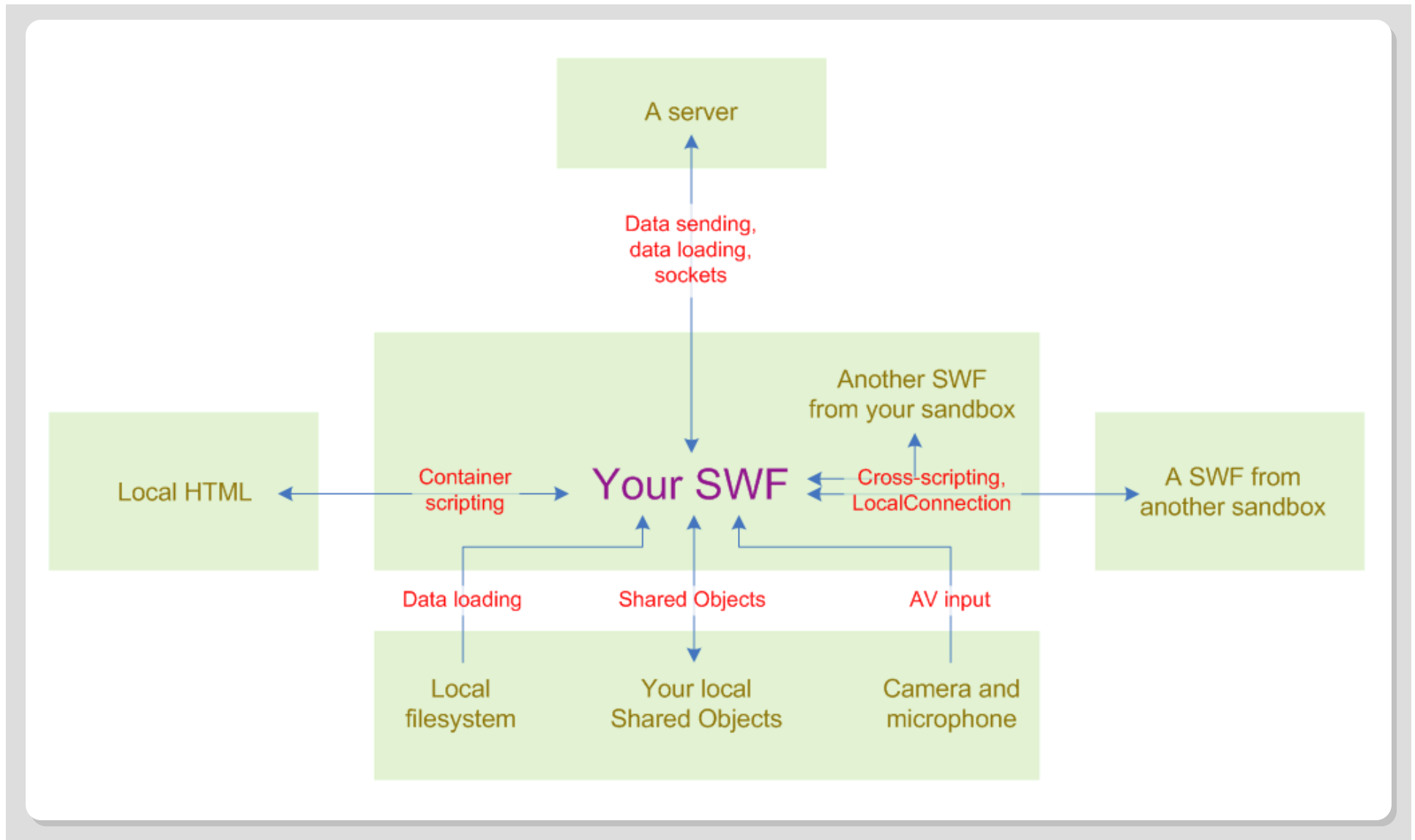
- All SWFs begin life as local as you create and revise them
 - Local security can be a bit of a pain at this time
 - If you will deploy on a server, local files do not properly simulate your runtime security
 - You should create a test server if you plan to deploy on a server
 - This is especially important for setting up demos for your customers
- You may create SWFs for local deployment
 - CD distributions, help systems, offline libraries
 - Downloadable games, viral marketing
 - Apps for unconnected or semi-connected environments
 - Local portions of hybrid offline/online apps
- If you're planning a local deployment, definitely consider targeting AIR
 - AIR security can be much less restricted at runtime – like traditional desktop apps
 - Check out Lucas Adamski's AIR Security session to learn more

So...

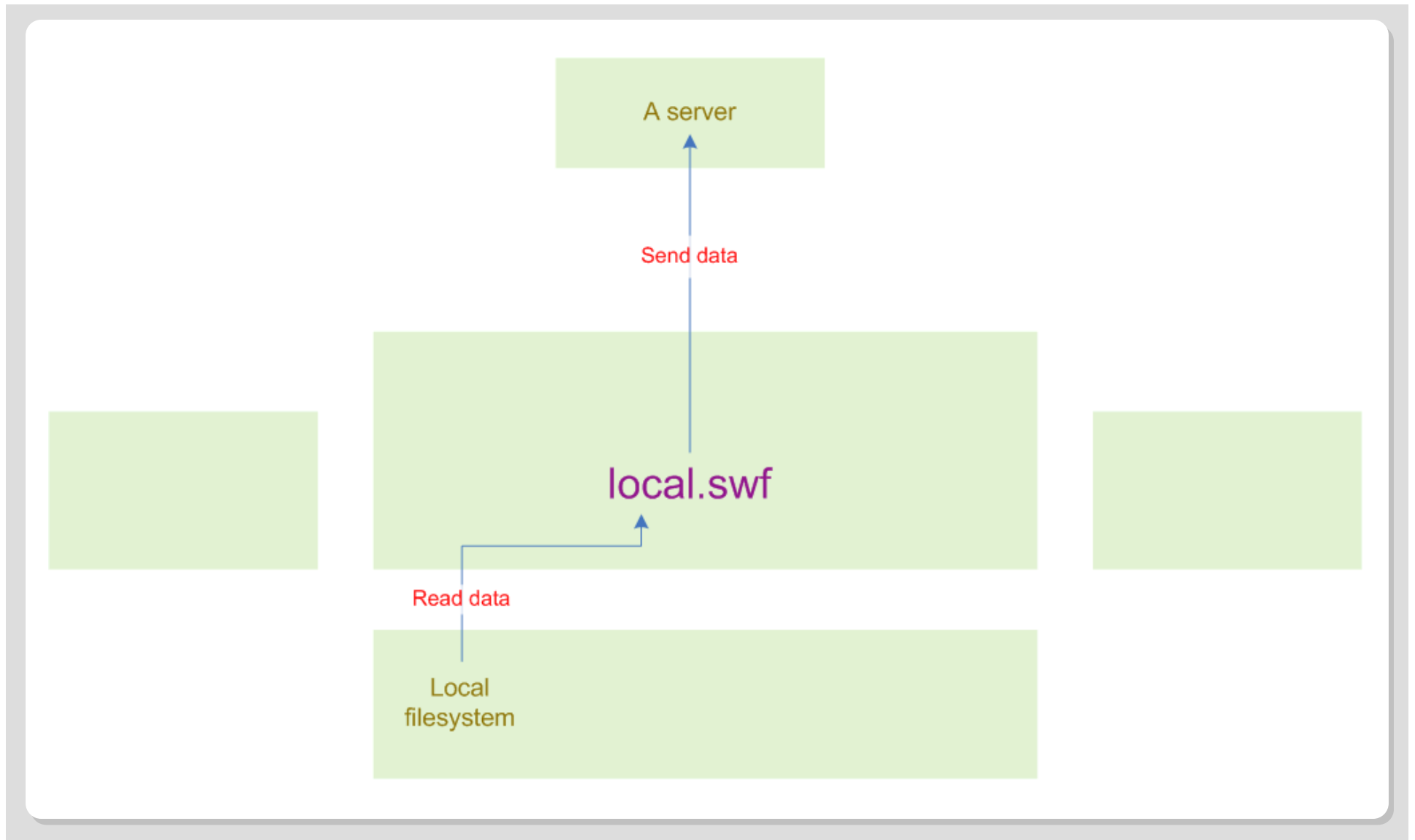
MAX

Rule 4: Prototype early.

The Big Picture for Local SWFs



Why Do We Need Local Security?



Local Sandbox Comparison



Sandbox	Read local files?	Communicate with servers?	Communicate with HTML?	How hard to achieve?
Local-with-filesystem	✓	✗	✗	Easy
Local-with-network	✗	✓	✗	Easy
Local trusted	✓	✓	✓	Harder

That's about it.

If You Remember Nothing Else...



- Rule 1: Use least privilege.
- Rule 2: Validate input.
- Rule 3: Deploy HTTPS consistently.
- Rule 4: Prototype early.

- Flash Player Security page, with many good links:

<http://www.adobe.com/products/flashplayer/security/>

- Click the "Resources for Developers" tab
- Especially see the Flash Player Security chapter of *Programming ActionScript 3.0*
- See *Security Changes in Flash Player 8* for a detailed introduction to local security
- Colin Moock's *Essential ActionScript 3.0* has a great chapter on security

<http://www.oreilly.com/catalog/9780596526948/>

Q & A