Adding scalability to legacy PHP web applications

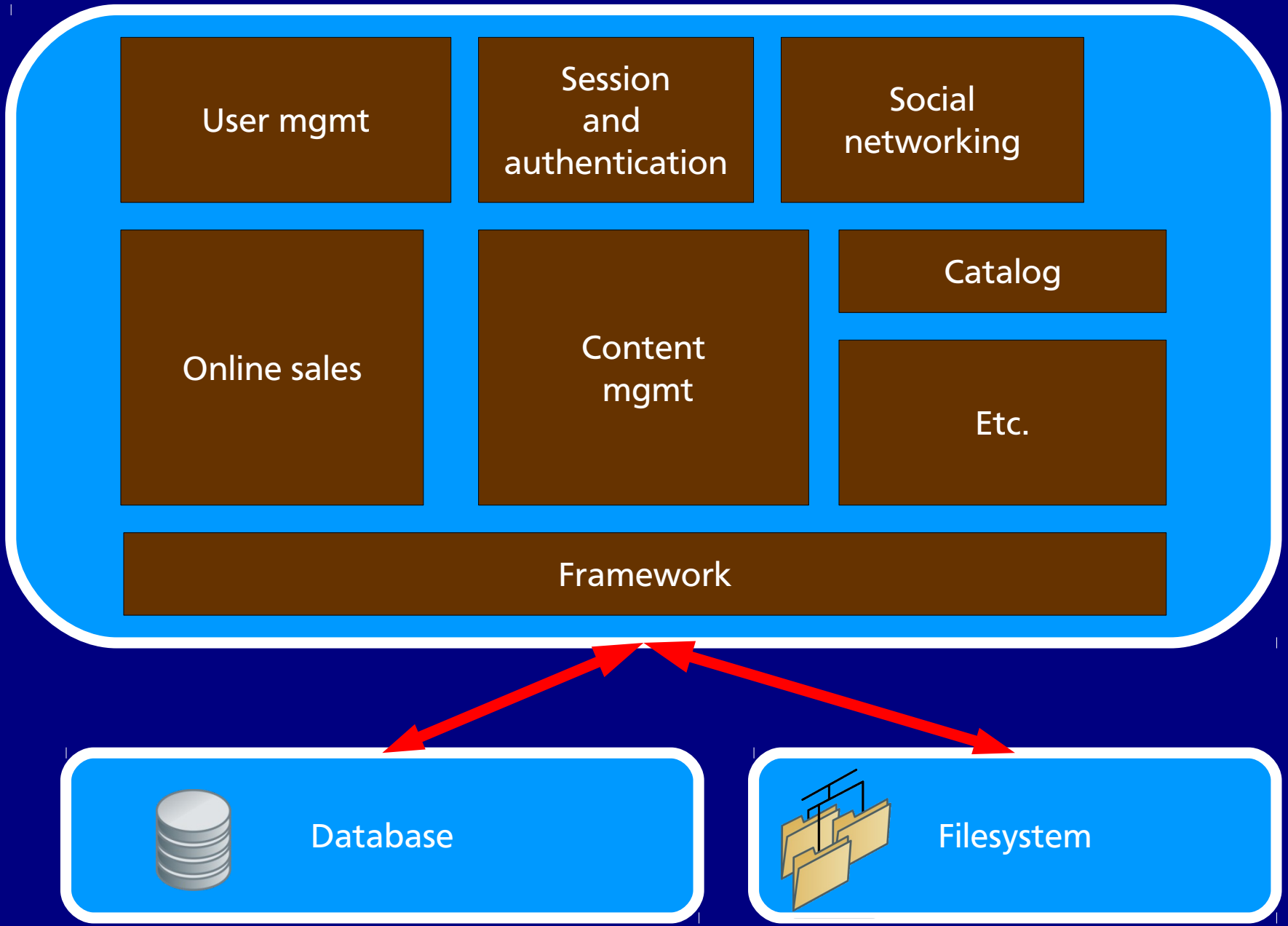
Overview

Mario Valdez-Ramirez

The scalability problems of legacy applications

- Usually were not designed with scalability in mind.
- Usually have monolithic design.
- Even modular applications use a fake separation.
  - For example, multiple modules using the same framework, communicating each other using framework functions.
  - The framework is still monolithic.

Monolithic web application



Why the scalability problem should be solved in your application

- The longer it takes to fix this, higher the cost and time.
- The problem won't go away, it will only get worst
  - as the user base grows,
  - with each new feature added.
- Application will become costlier to maintain.
  - Then one day it will be unmaintainable.
- There are software development issues that can be solved too by fixing this.

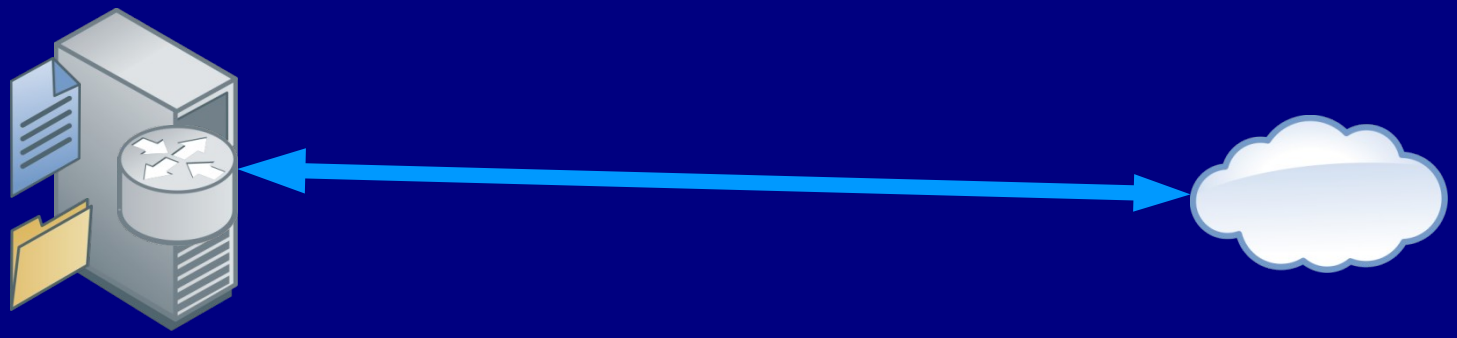
Vertical scaling

- Adding more power to the current hardware.
- Usually more disk, memory, processors and bandwidth.
- Quickly becomes very expensive.
- Does not scale very well, because of technological limitations
  - you can only go as far as the fastest processor, the largest disk, the biggest network link, etc.

Vertical scaling

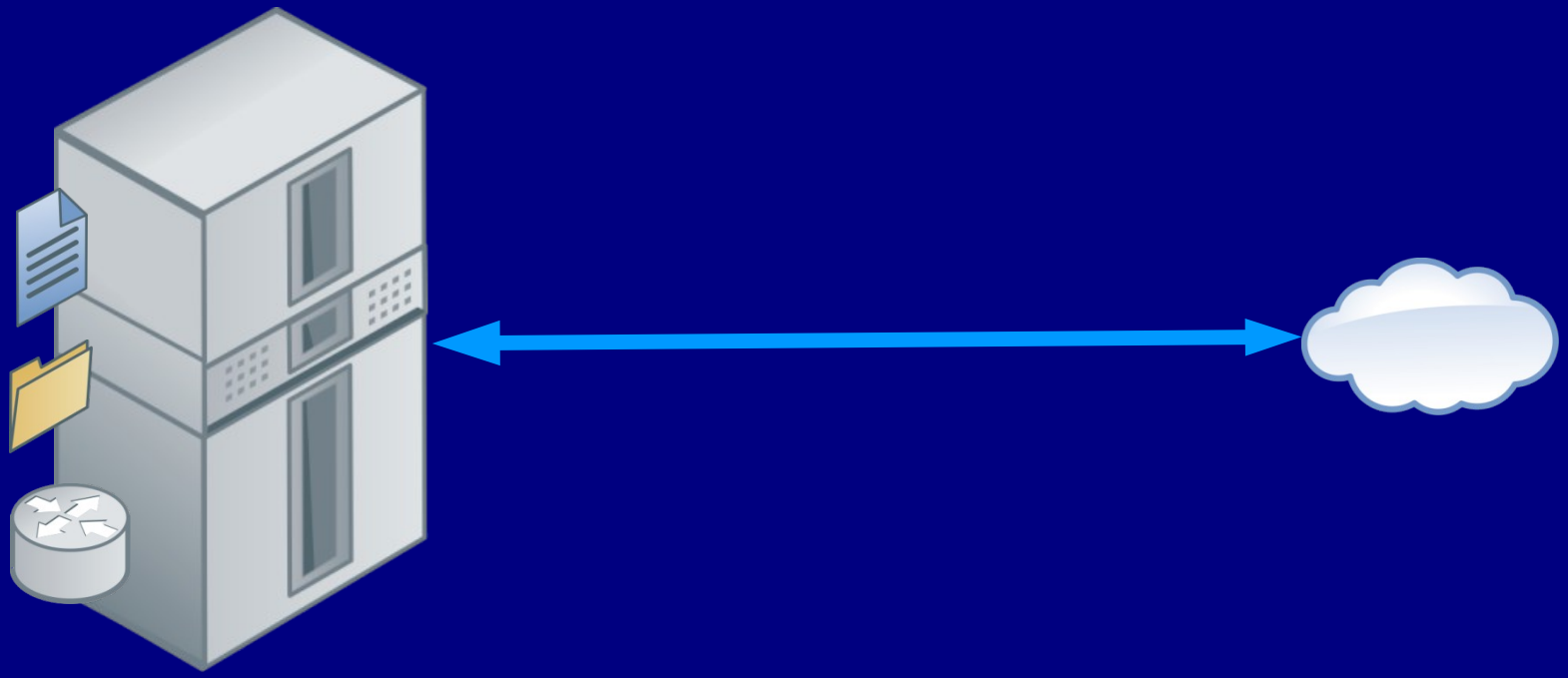
- However it is very simple to implement.
  - Does not require changes to the application.
- It is usually the first attempted solution.

Single server, multiple jobs



Web server,
file server
database server

Single server, multiple jobs, vertical scaling



Web server,
file server
database server

Horizontal scaling

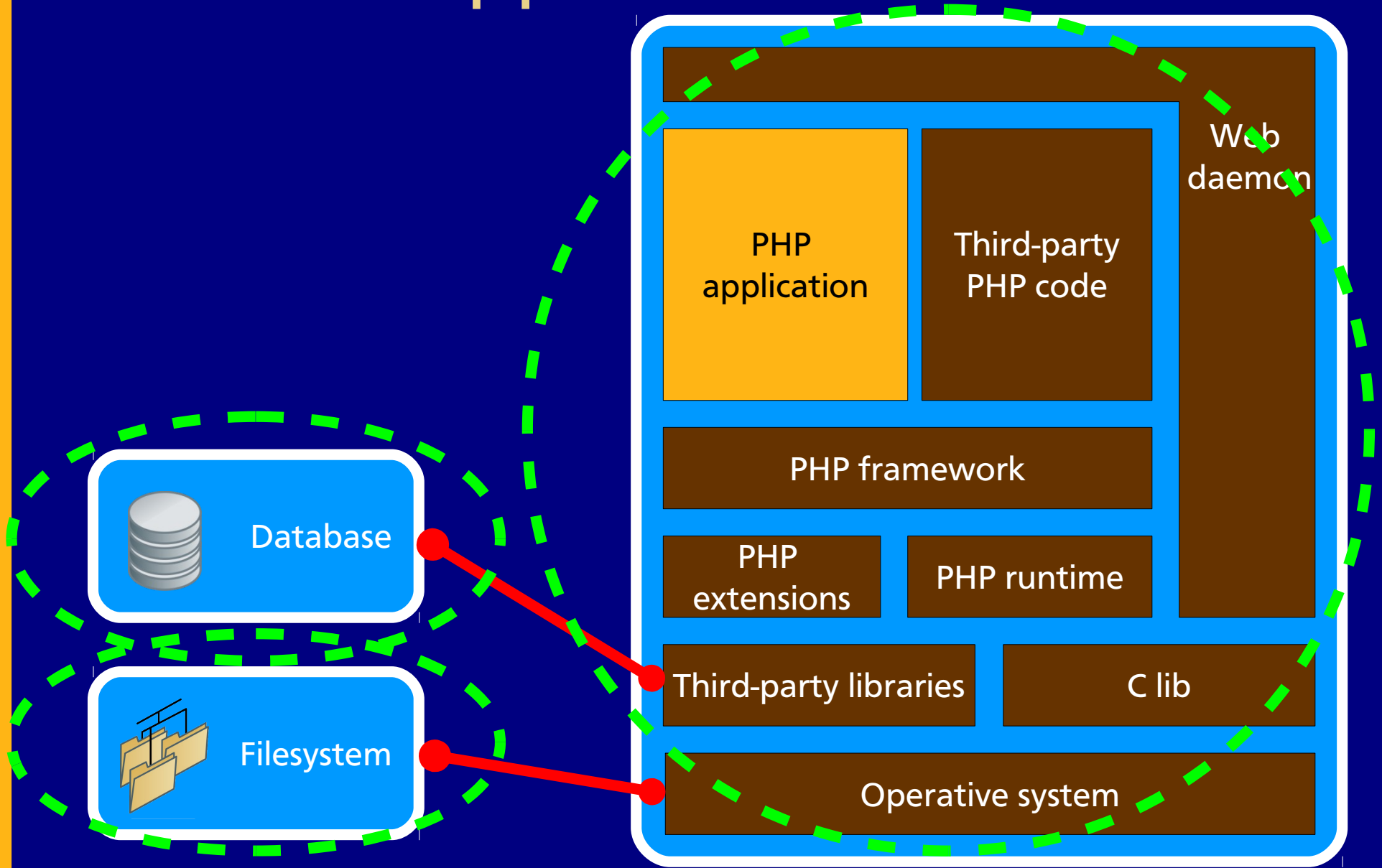
- Adding more nodes (servers).
- Usually requires splitting the job done by the server and the application.
- Allows to use cheaper resources.
- Requires changes to the application.
- Increases complexity.
- Scales better than vertical scaling.
- Can be used to build redundant solutions.

How to start fixing a legacy application?

Easy fixes

- Leverage separation already present in any PHP application.
- Separate
  - database,
  - data files,
  - PHP scripts.
- Different servers for each.

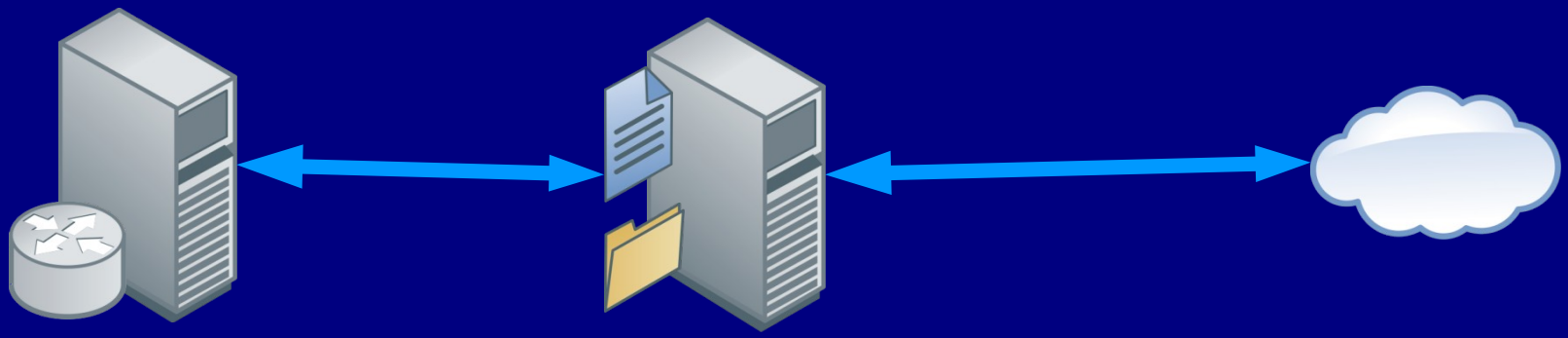
Easy fixes, separations presents in most PHP applications



Easy fixes

- Two servers, one for PHP/files and one for database.
- Three servers, one for PHP, one for database, one for files.
- Move static (or even public dynamic files) to another server (or CDN).
- Use multiple servers for PHP application.
  - Use a load balancer.

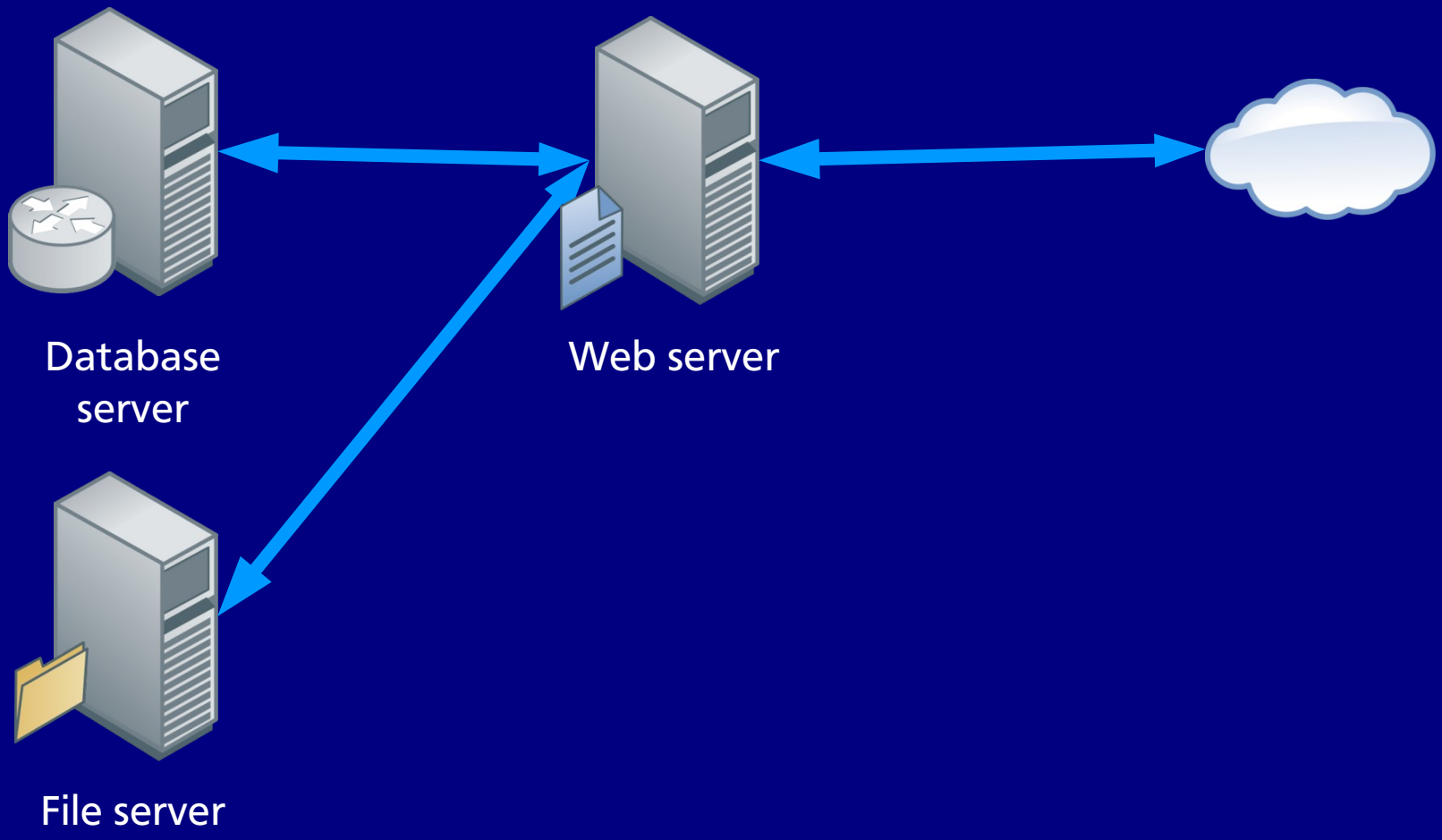
Multiple servers, different jobs



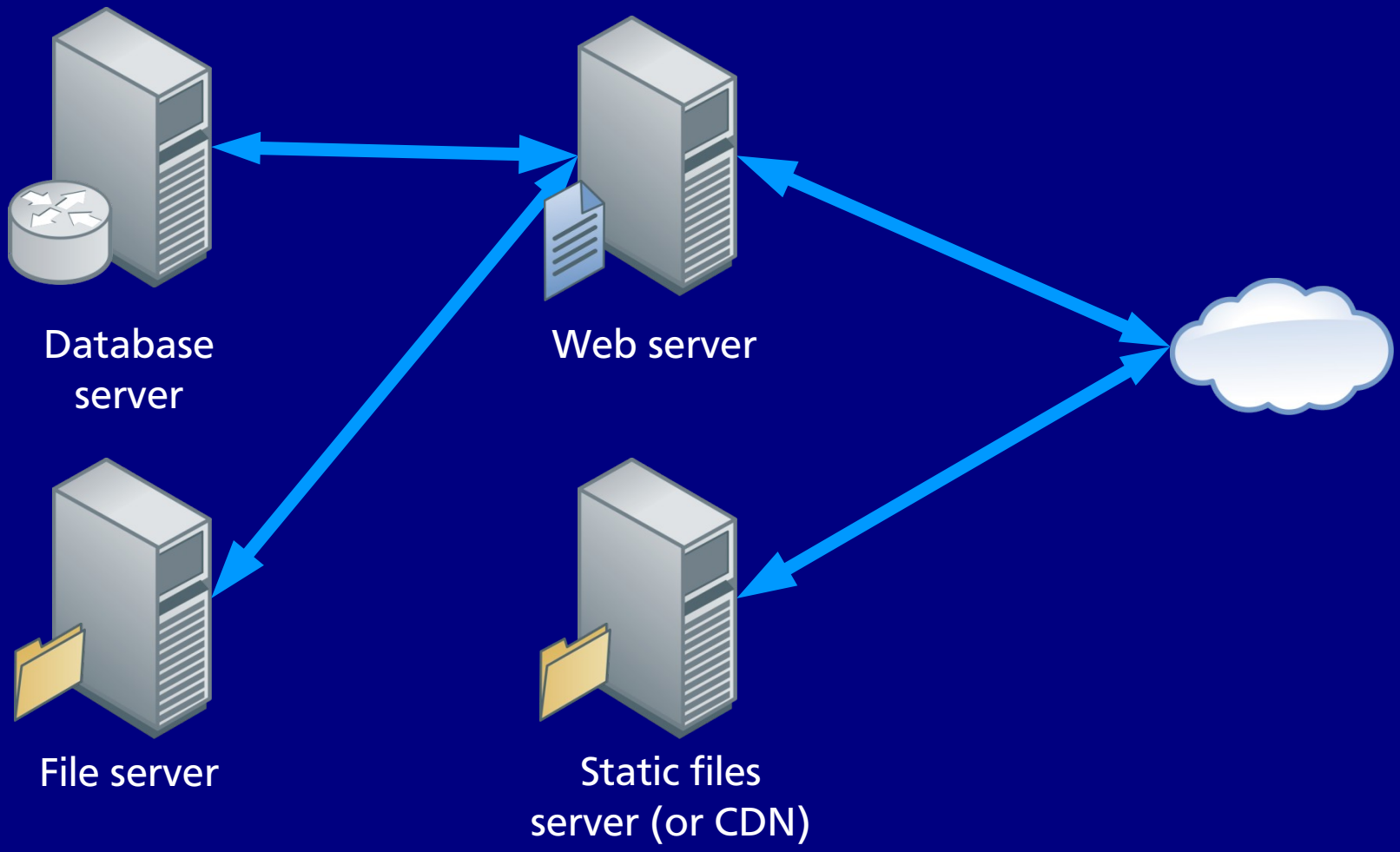
Database server

Web server, file server

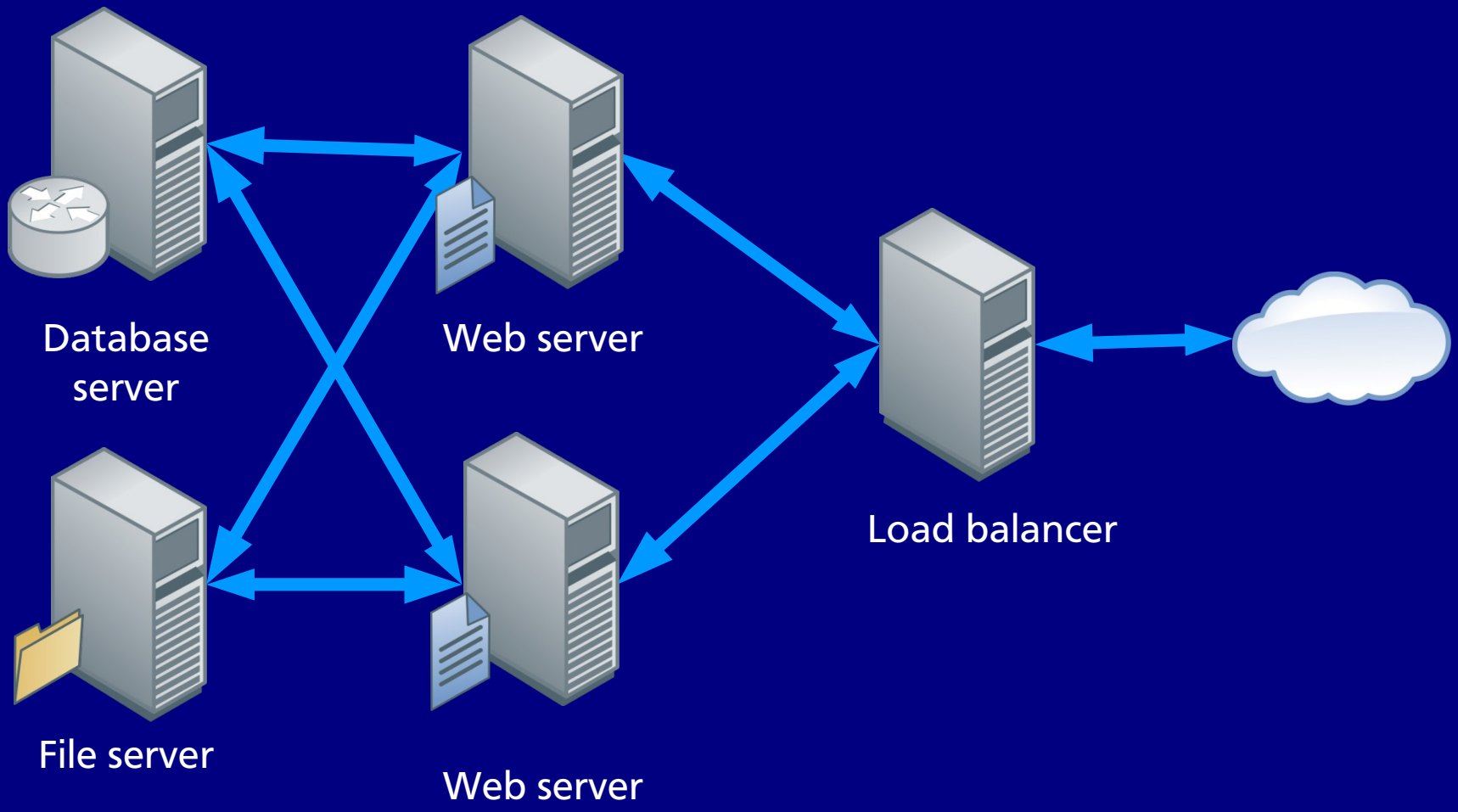
Multiple servers, different jobs



Multiple servers, different jobs



Multiple servers, different jobs



Advantages of these easy fixes

- Require minor changes to the web application.
- Relatively cheap
  - usually cheaper than using one very large server.
- Easy for most programmers to implement
  - mostly changes to session and file management.
- With the load balancing option, it is very easy to add more web servers.
- With proper monitoring the load balancing option provides some redundancy.

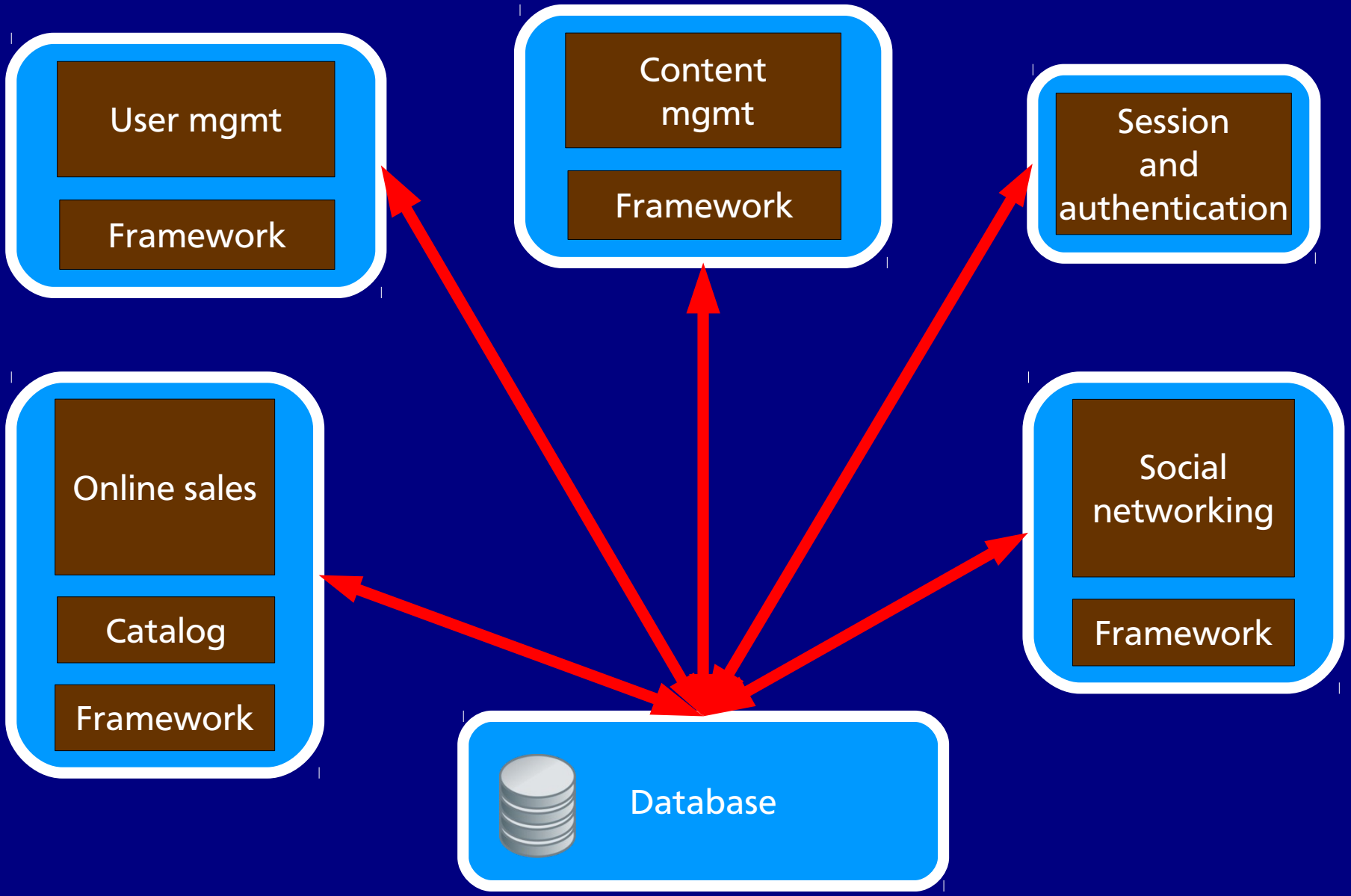
Disadvantages of these easy fixes

- There are still potential bottlenecks and single failure points:
  - The database server.
    - A very common bottleneck for transactional applications.
    - It can be replaced with a DB cluster.
  - The file server.
    - It can be replaced with a SAN or distributed filesystem.
  - The load balancer.
    - It can also be replaced with multiple load balancers.

Harder fixes.

- Code splitting (module separation)
  - Split the application as simpler modules.
  - Each module is independent.
  - For example, one module for user management, another for content management, another for online sales, another for social networking, etc.
  - Any communication among modules is done through the database (database is shared).
    - There are alternatives, like web services.
  - Each module must be really independent, down to the framework and third-party libraries (if used).

Code splitting



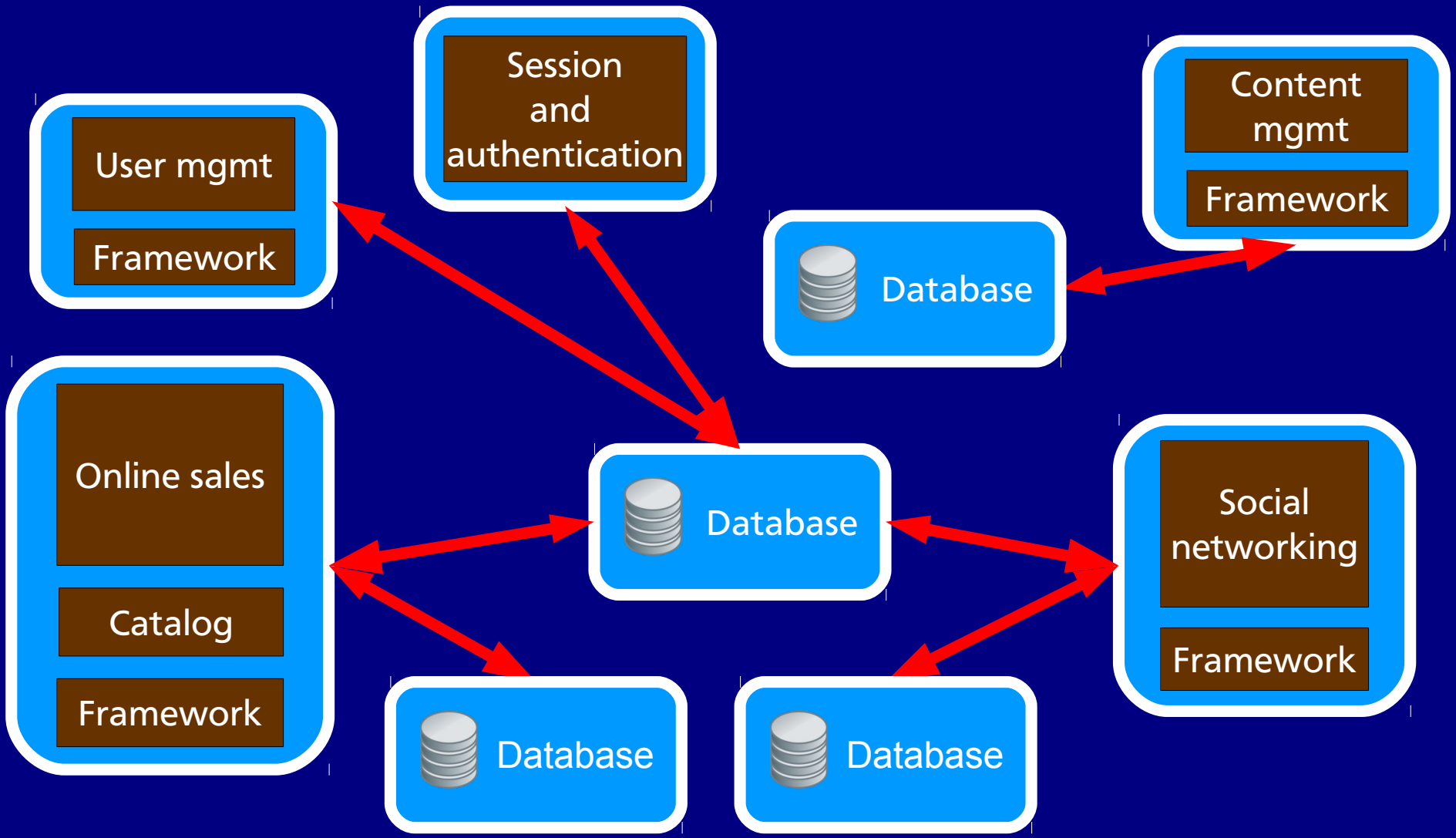
Advantages of code splitting

- Each module is simpler to maintain.
- Each module can be hosted in a different server.
- Each module can be build using different frameworks, even different languages and use different OS.
- Remove most licensing issues as each module shares only the data, not the code of other modules.

Harder fixes

- Code+data splitting
  - Split the application but also the data (multiple databases).
  - For example, the social networking data may be separated from the online sales data, and the content management data from the user management data.
  - Modules still share some databases (for example, the users database).
  - Easier to implement if the application is already split in modules.

Code+data splitting



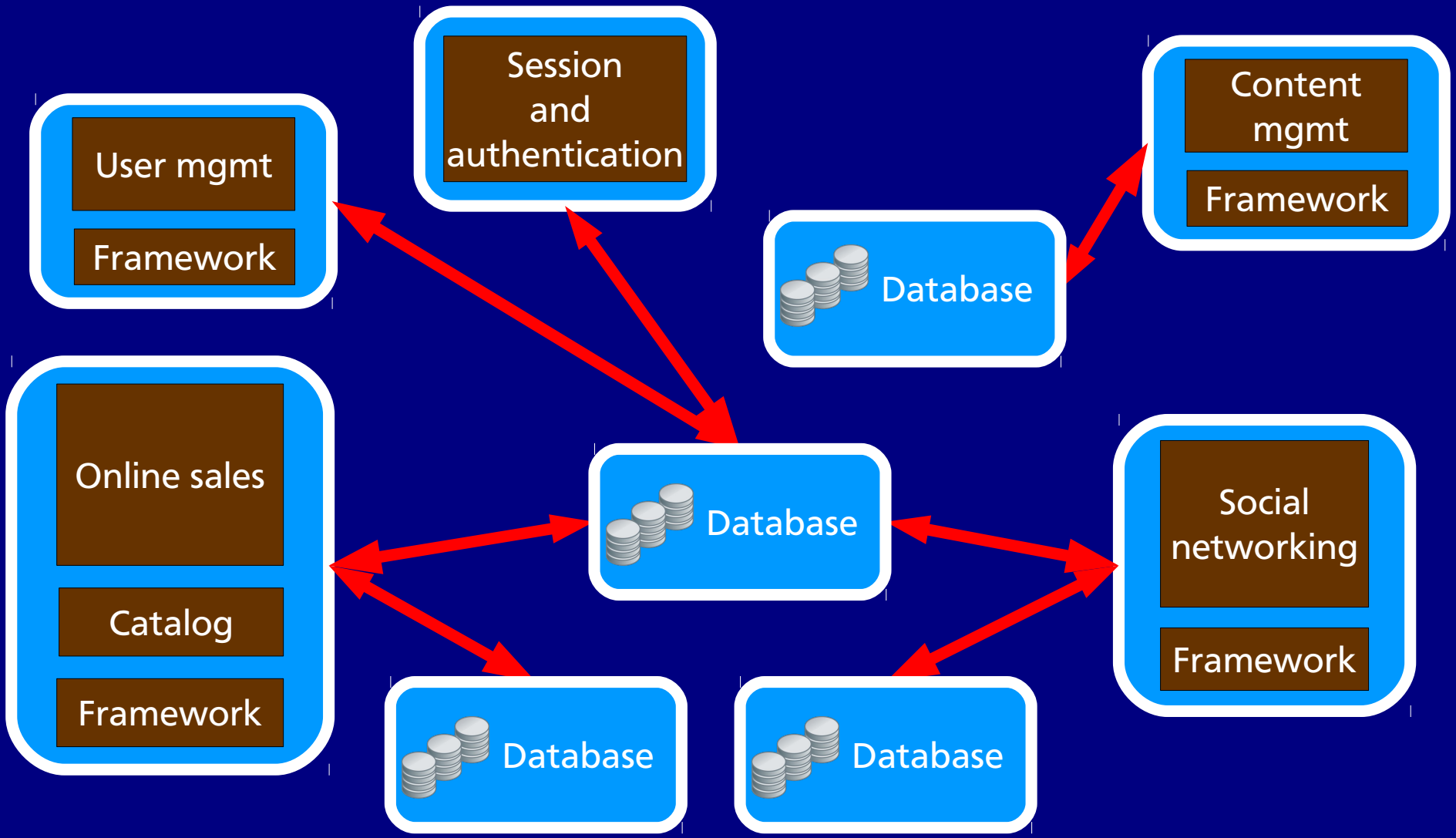
Advantages of code/data splitting

- All the advantages of code splitting plus:
  - Data is stored in multiple databases so,
    - it can be hosted in different servers,
    - and each database can use a different RDBMS,
    - or even use a different kind of DB (like NoSQL).

Harder fixes.

- Code+data splitting + data *sharding*
  - Split the application, and data vertically but also split the data horizontally.
  - For example, the content data can be split in multiple databases according to category, or content provider, or the user data according to geographic location.
  - Only make sense with large datasets, with logically separated data.

Code+data splitting+sharding



Advantages of code/data splitting + data sharding

- All the advantages of code/data splitting plus:
  - Data is stored in multiple databases, not only per-module, but per other criteria (client, category, region, etc.).
  - It allows for greater scalability and performance, as long as queries don't require data from multiple shards.
  - Useful for multitenant applications.
  - For data requiring high separation between clients (for security, compliance, privacy), the sharding is done per client.

New opportunities for code/data split applications

New opportunities for code/data split applications

- Software development can be split easily among teams.
  - Each team can develop using whatever language, framework and platform they know.
  - Increase resilience of the project, as each developer is easier to replace affecting temporarily only one module.
  - Potentially more cost-effective development.
  - Allow to outsource the development among multiple providers.

New opportunities for code/data split applications

- Software development is less risky.
  - Easier to fix bugs as they span single modules.
  - New features can be deployed without affecting other modules.
  - Allow to test new solutions (platform, methodology, technique, language, etc.) without risking the whole application.
  - Allow to integrate third-party applications without licensing issues.

Other useful solutions not covered

- Caching
  - specially PHP caching, including clustered caches like Memcached.
- Database clusters
  - master-slave, master-master, clustered RDBMS and cloud databases.
- Cloud/PaaS scalability
  - however most PaaS providers require the application have already all the easy fixes implemented to leverage their platform.

How to start?

How to start?

- Start with the easy fixes.
  - One step at a time.
- Plan and implement module splitting.
  - Divide by two. Then by four. Then continue.
- Any new feature or module should be implemented independently.

How to start? Easy fixes

- Start implementing the easy fixes in the main application.
  - Usually only the database, session and file management functions require rewriting.
  - If your DB, session and file handling code is not inside a single class or group of functions, first refactor, then split.
    - Spaghetti code should be first untangled, then split.

How to start? Module splitting

- Use the documented API of your application and start planning how to separate its functions per module.
  - If you don't have a defined API, define it, then refactor.
  - If you have one but is undocumented, document it.
- Decide what data is shared among all modules.
- Then refactor.
  - Never refactor without architectural planning!
- This job is better suited for your current development team.
  - It requires deep knowledge about your application.

How to start? New modules

- Any new feature or module, should be started as a fully separated module, down to the framework.
  - Design with share-nothing code in mind.
  - Remember each module should be independent.
  - Share data, not code.
- New modules are good opportunities to test new platforms, languages, frameworks and development teams.

Questions?