anotheria.net
ano-tecture

Universal Architecture for Portals
by Leon Rosenberg
anotheria.net
2008-2010

TOC

- Target group
- Requirements
- Solution
- Case studies

Target group

- B2C portals.
- Service oriented portals.
- From startups to matures.
- International (or internationalized) portals.

Tweaks

- High read to write ratio.
- High traffic due to many lightweight requests.

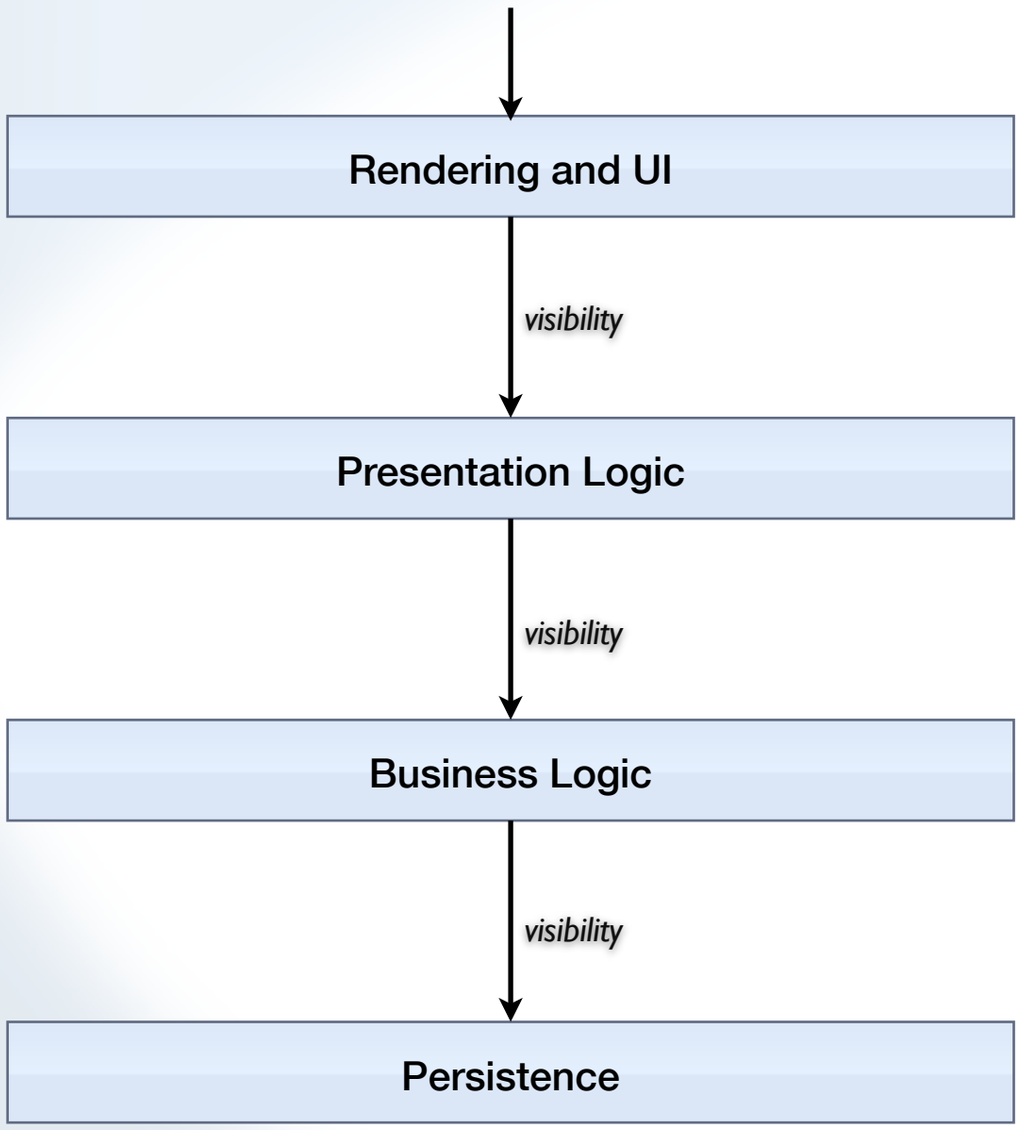
Requirements

- Scaleability.
- High availability.
- Growth support.

Solution

- The 4-Layer-n-Tier architecture.
- Clearly defined layer. Layer separation by responsibility.
- Flexible setups. 2Tier, 3Tier, 4Tier... n-Tier.

Layer Architecture



presentation rendering, markup generation, ajax-connector

presentation logic, stateful services, per-user-caching, portal-api

business logic, object-cache, single responsibility services

daos, fs-reader/writer, connectors

each layer has its own, unique responsibility. responsibilities are defined on the following slides.

Rendering and UI layer

- Produce HTML or other (js, css, json, xml) markup out of the business data for the customer.
- Parse incoming parameters.
- Support in browser flow control.

Presentation logic.

- Service / information syndication for presentation needs.
- Per user caching.
- Validation.
- Provide testable interface for the frontend logic.

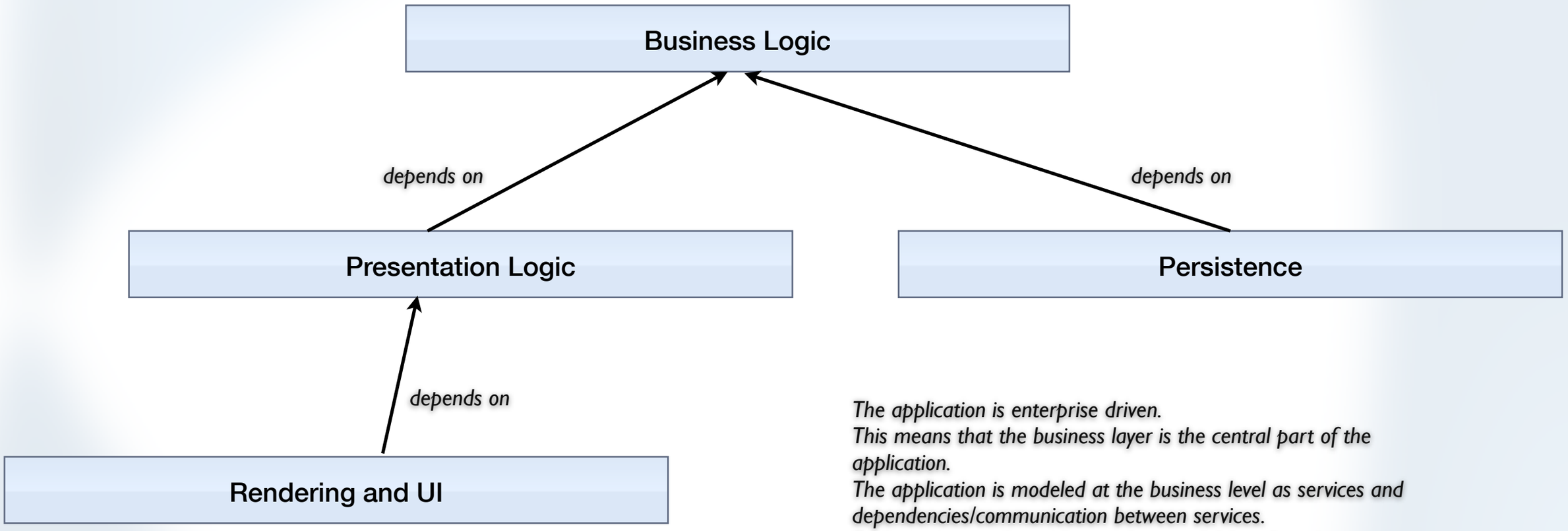
Business logic.

- Providing enterprise view on the application.
- Control and manage persistence layer.
- Provide services to the presentation.
- Caching.

Persistence

- Saving and loading objects.
- Performing queries.

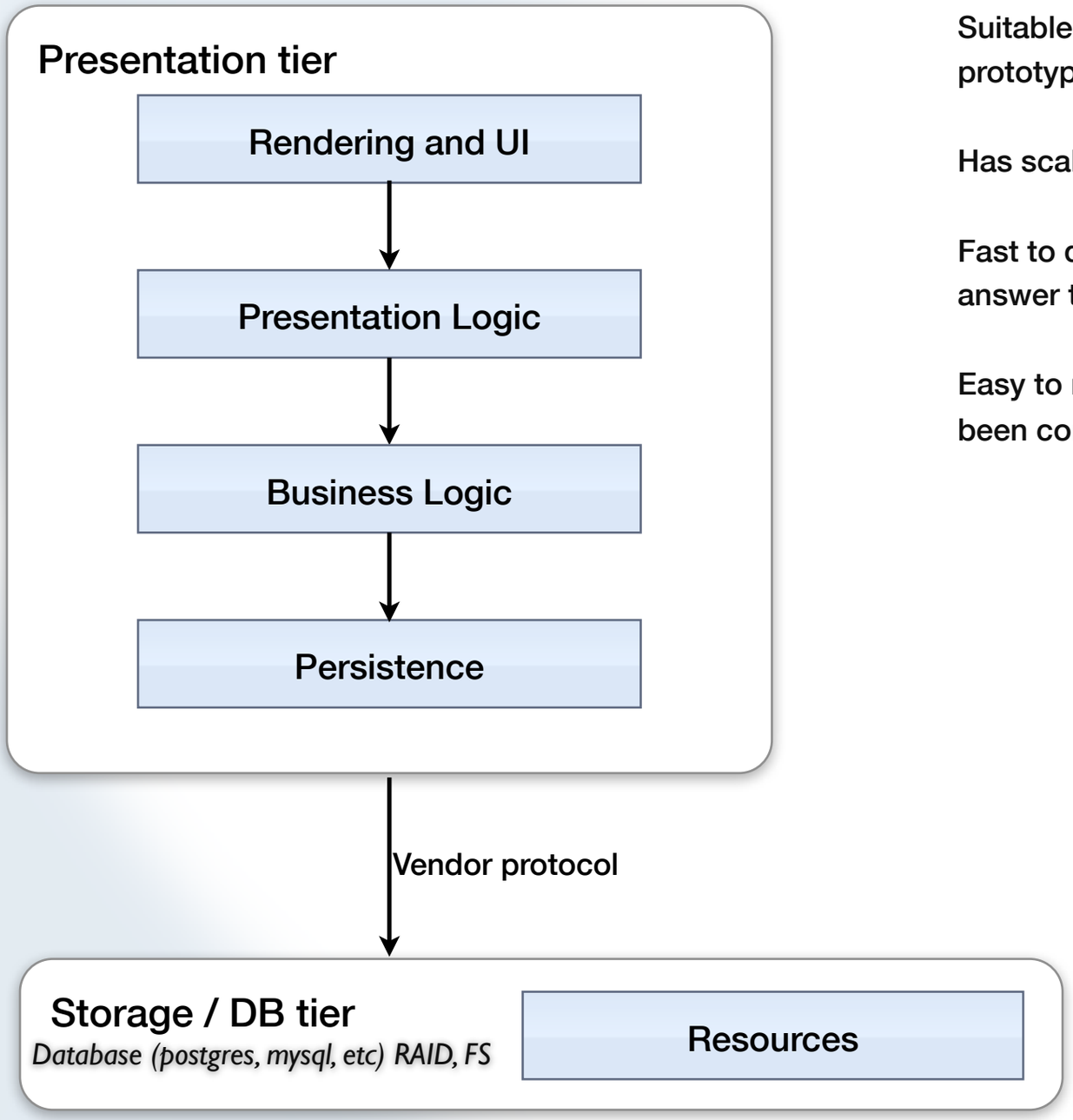
Code modeling



The application is enterprise driven.
This means that the business layer is the central part of the application.
The application is modeled at the business level as services and dependencies/communication between services.
Each service can have it's own data model, but there is no common data model which is valid for the whole application.
As a consequence - each service can have it's own database and can scale independent of other services.
Another consequence - no joins needed.

Examples

Architecture I (2T)



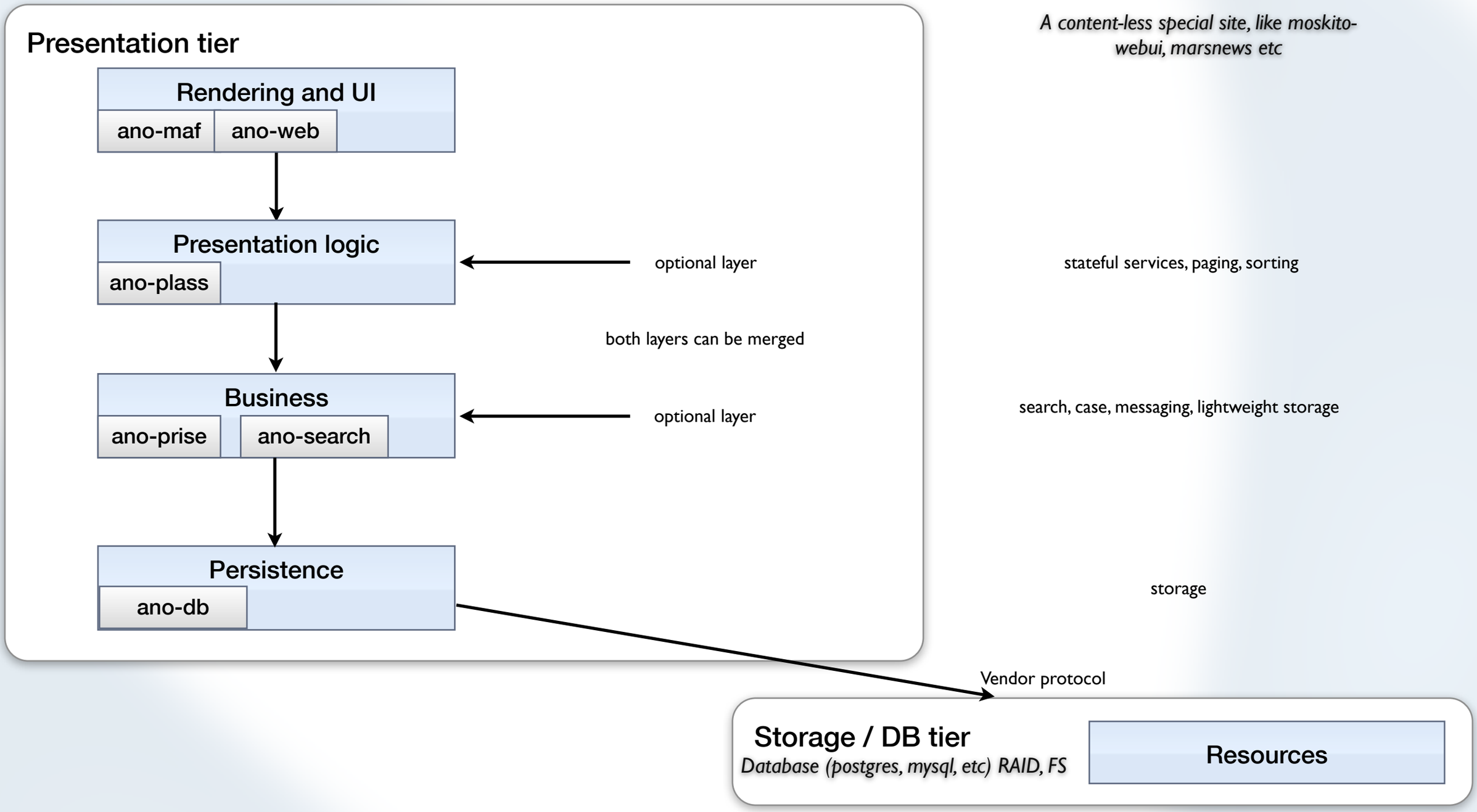
Suitable for small projects, low-traffic portals, prototypes etc.

Has scaleability limits.

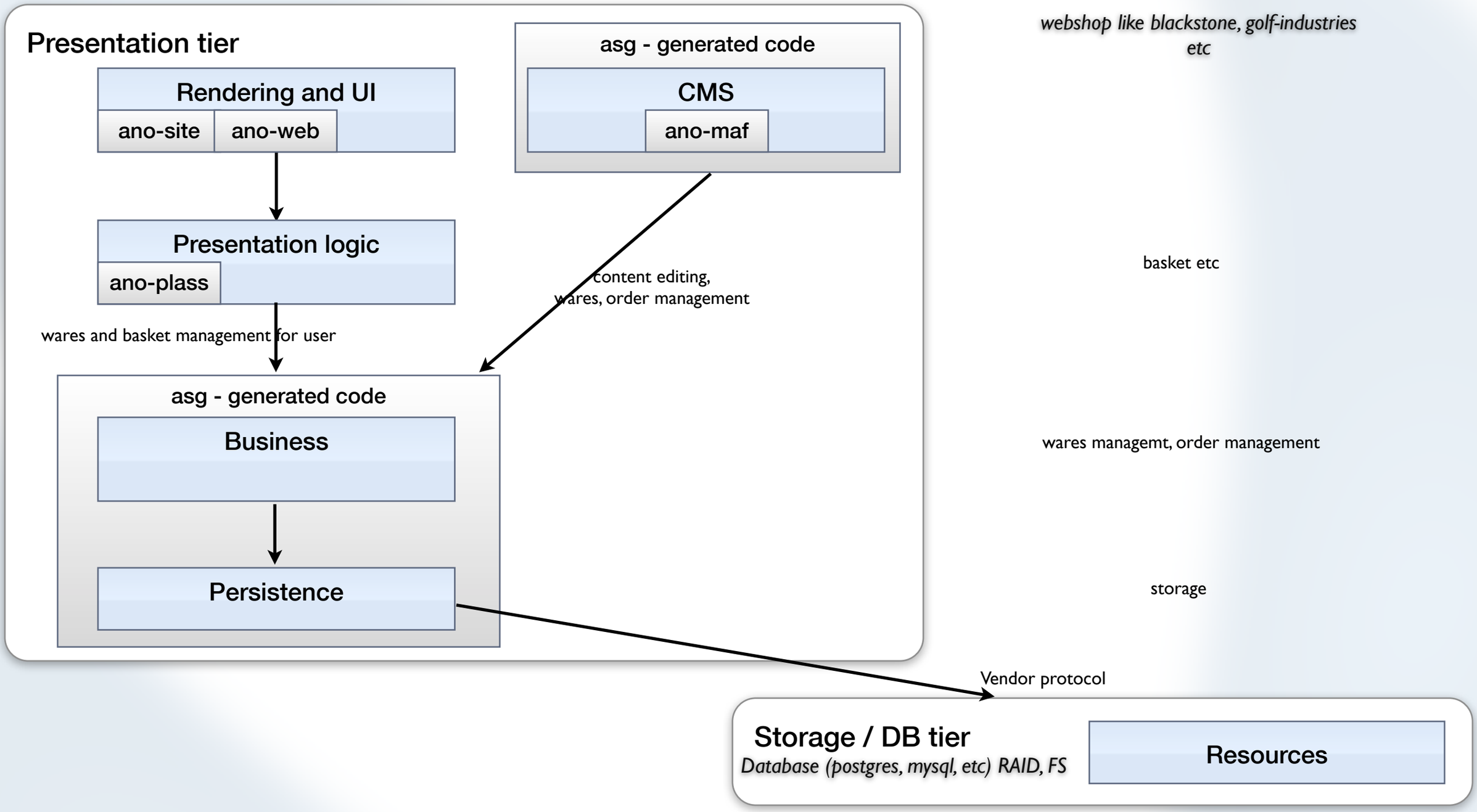
Fast to develop, easy to deploy - the java answer to LAMP.

Easy to migrate to 3T if the layer separation has been conducted strictly.

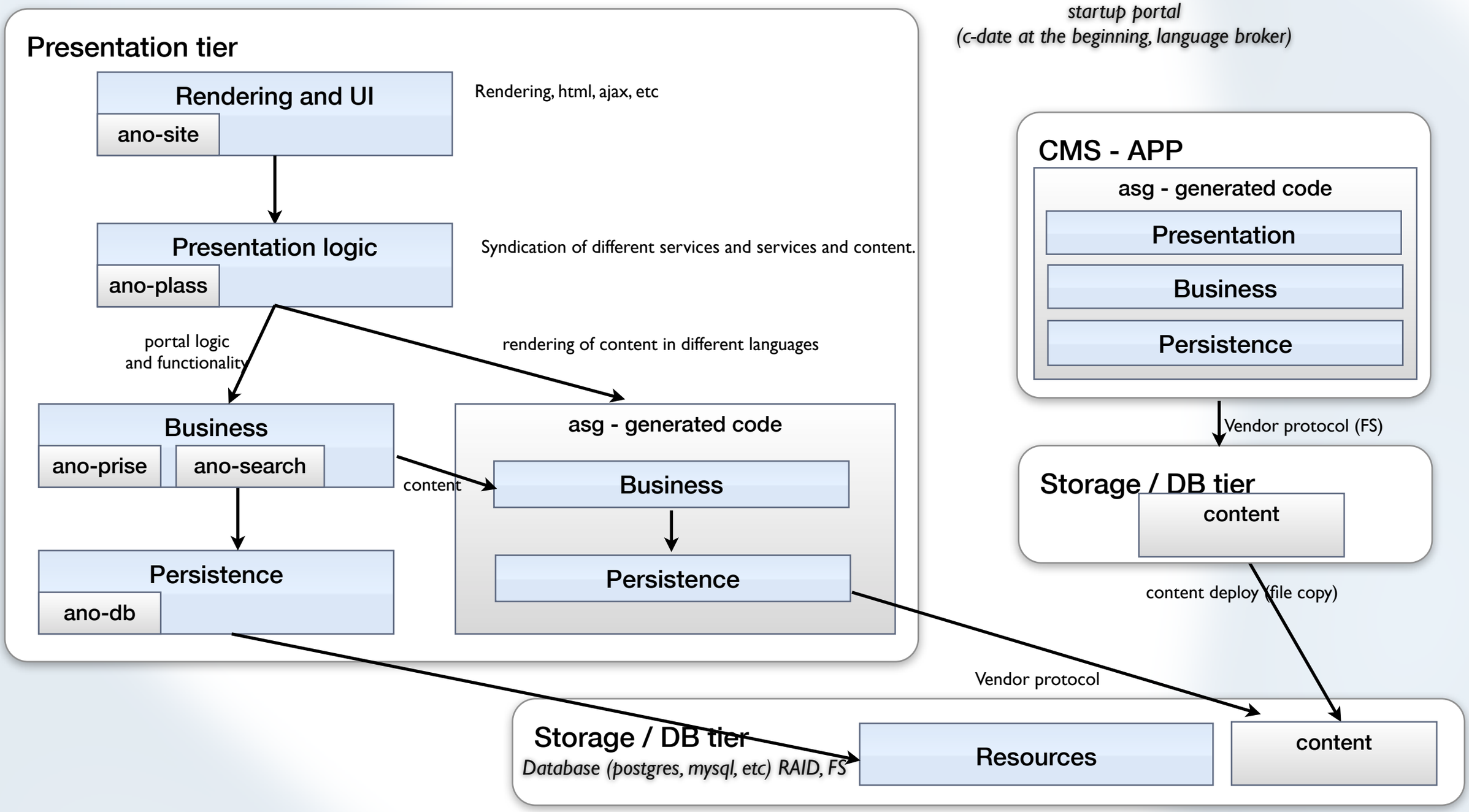
Example I : Technical Site



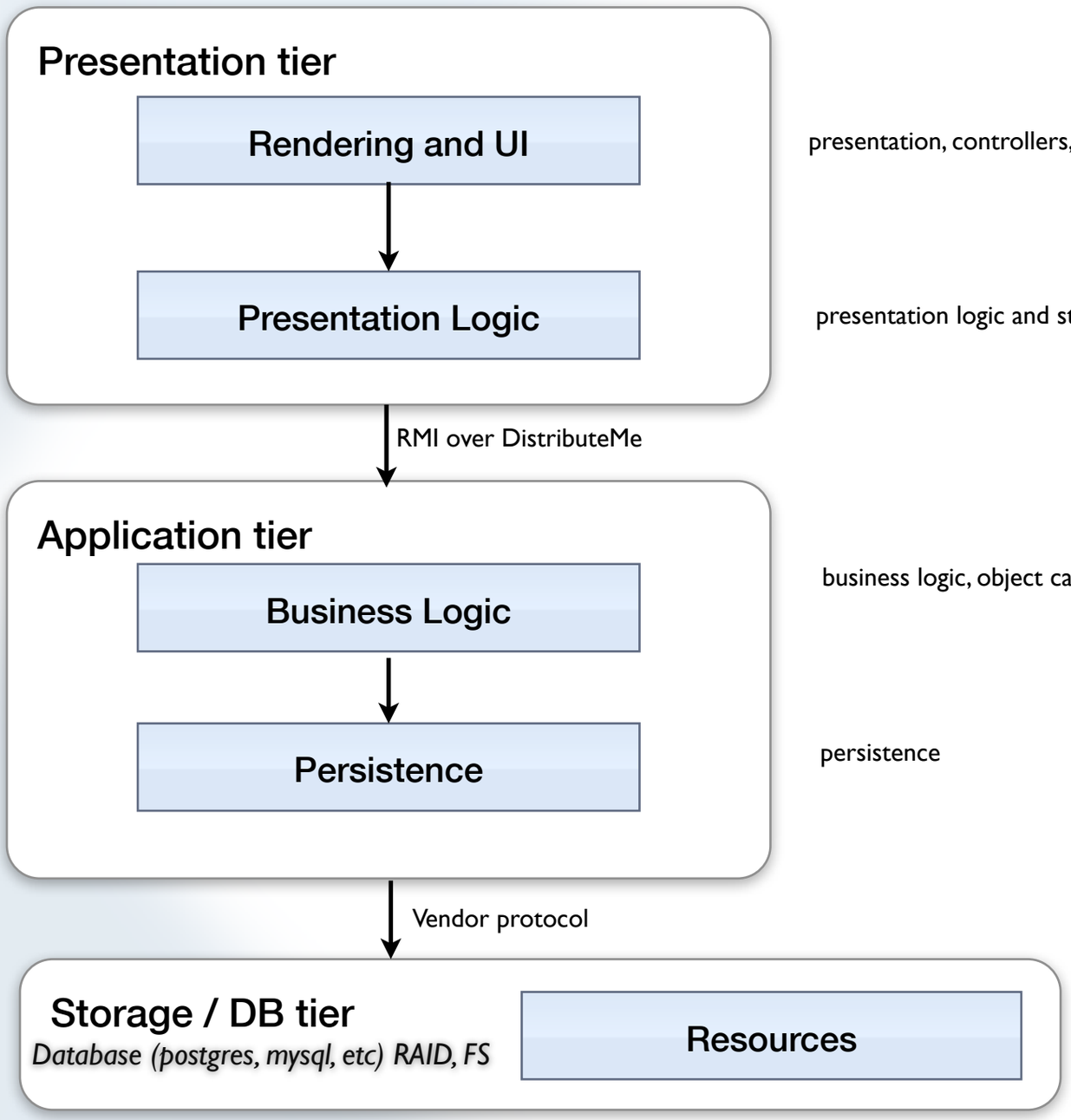
Example II : WebShop



Example III: Small Portal



Architecture II (3T)



presentation, controllers, action, markup

presentation logic and stateful services, per user caching

business logic, object caching

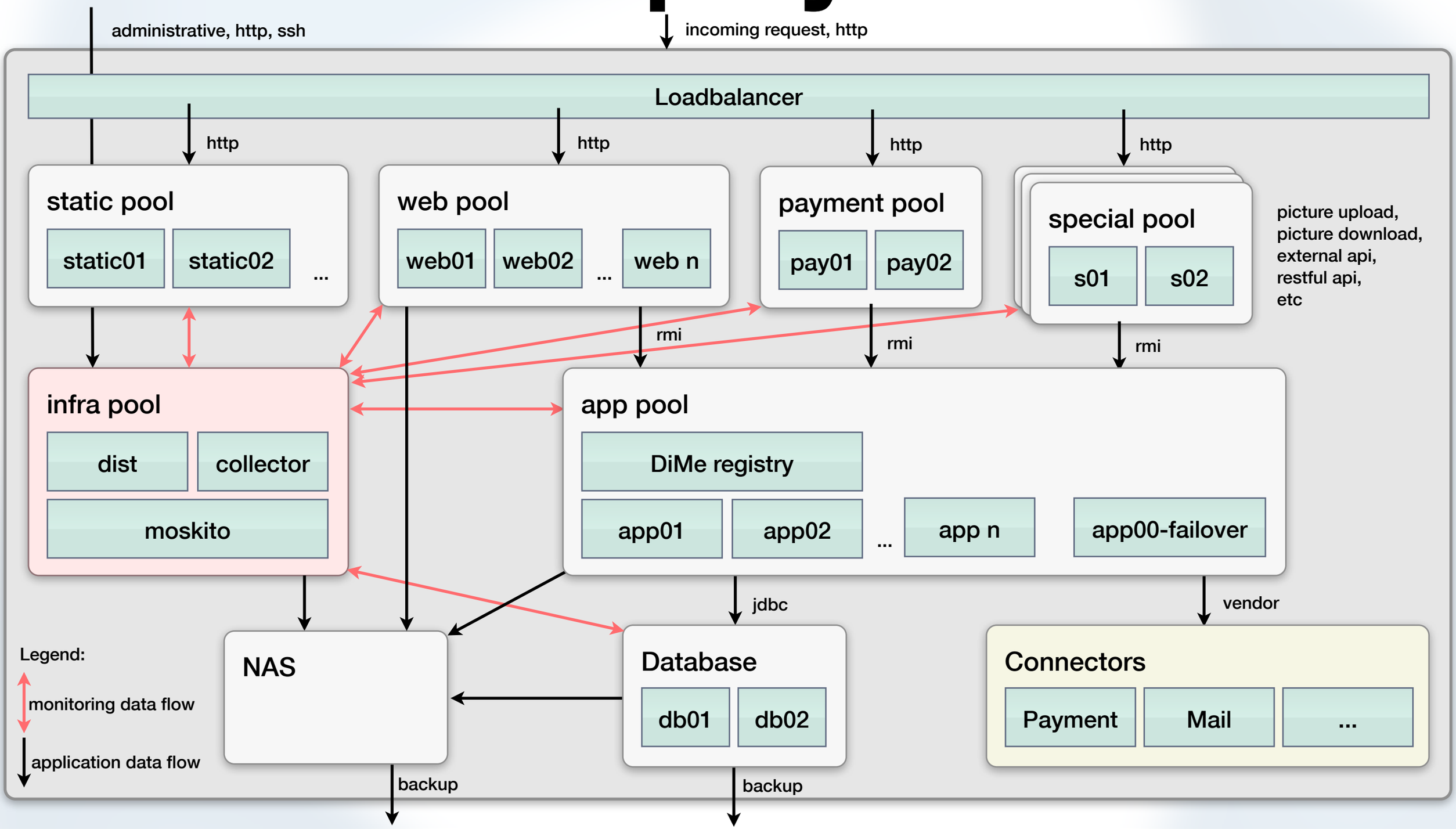
persistence

Suitable for mid range projects, mid/high-traffic portals.

Scales better.

Performs!

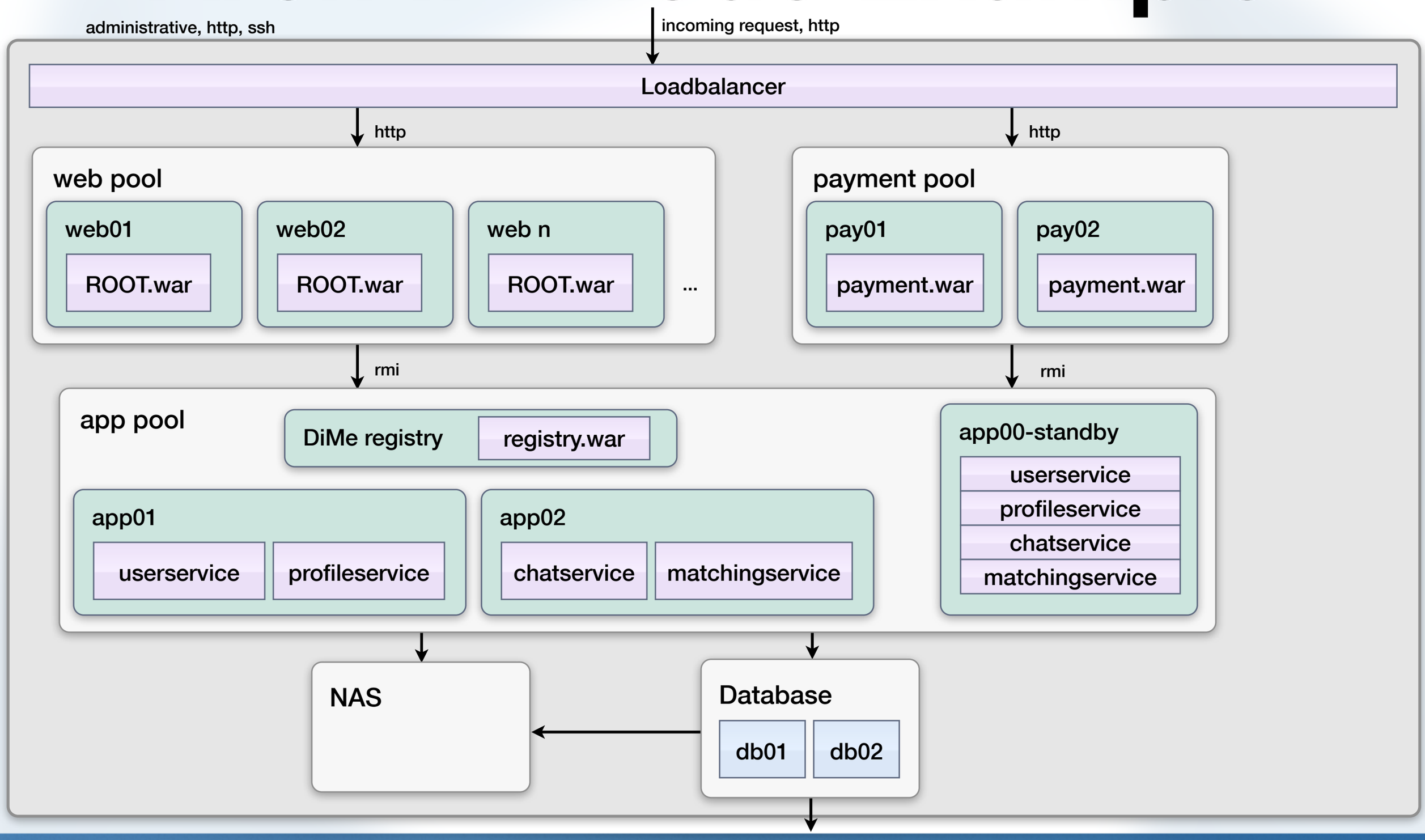
Arch II - Deployment View



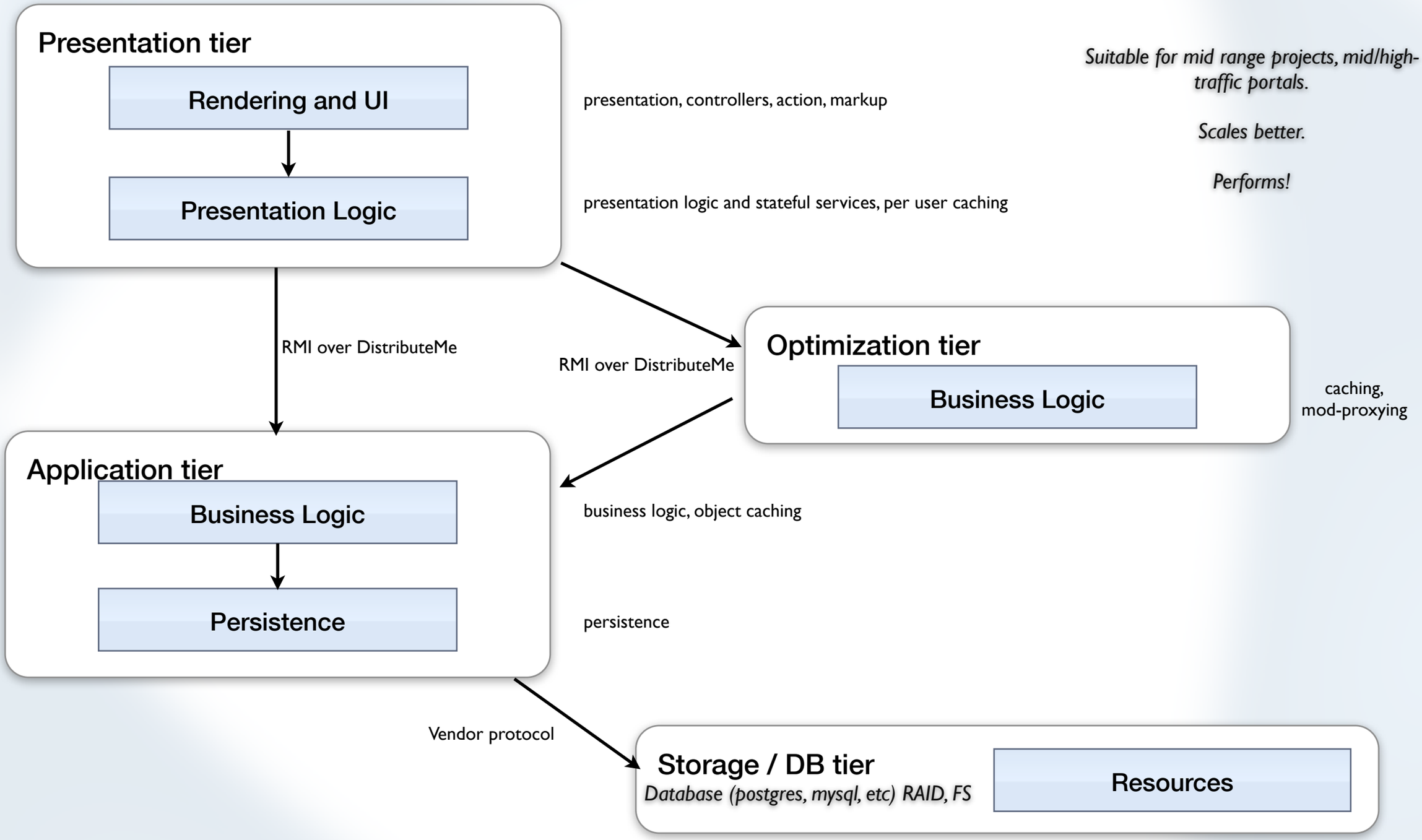
Architecture II - Nodes

- Presentation-tier nodes (web, pay, ext, photo etc) are web-applications (wars) running in tomcat and completely equal to each other (web01 == web02, pay01==pay02, pay01!=web01). They scale (almost) indefinitely.
- Application-tier nodes are java processes (services) running in separate java vms and accessible via RMI (Corba, Soap etc). Each process (service instance) has its own, unique goal and responsibility.

Arch II - Node Example



Architecture III (nT)



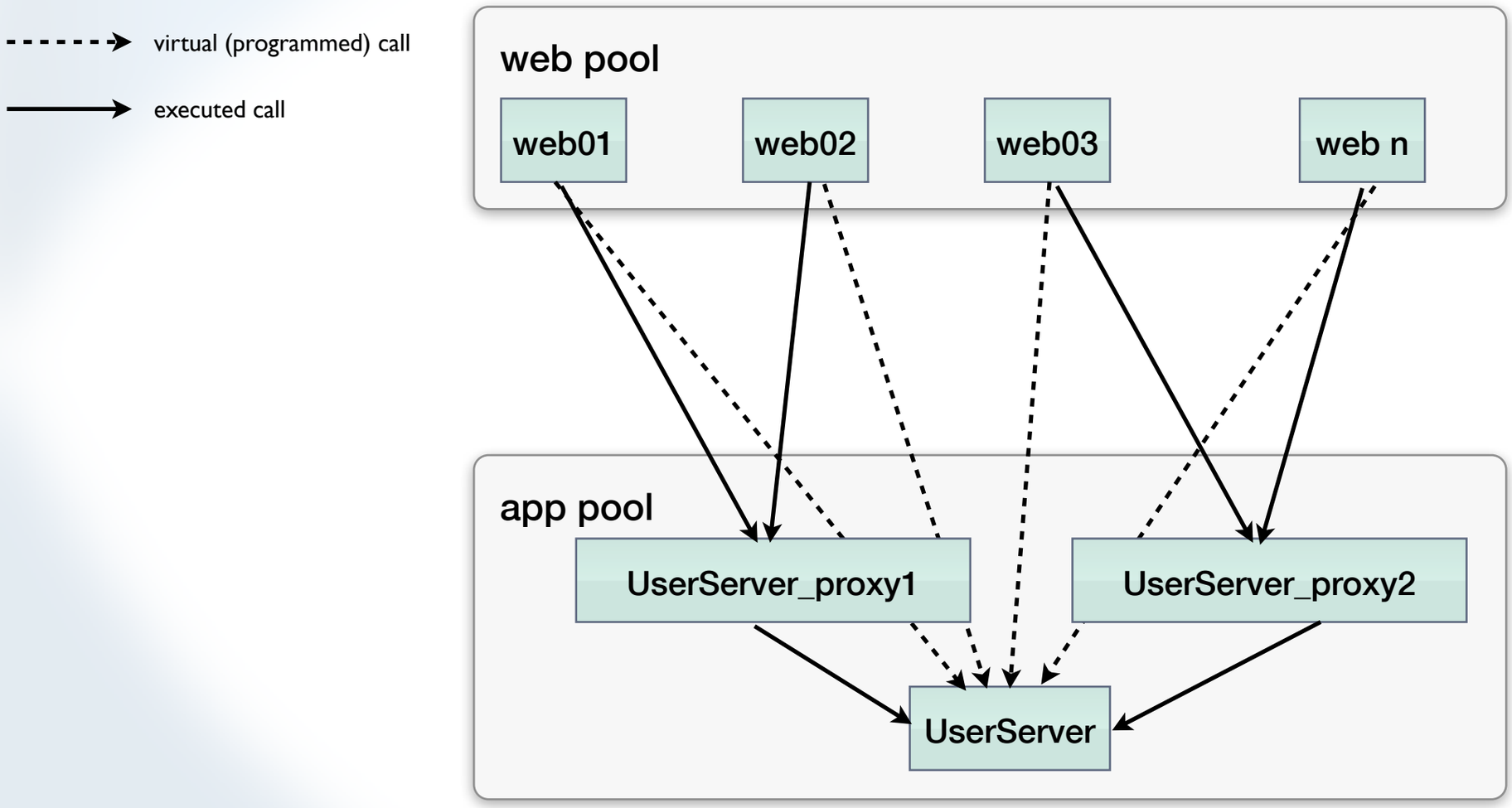
Optimization Tier

- Additional Tier between presentation and application tiers.
- Main responsibility: reduce load on application tier.
- Transparent (invisible) to the presentation layer, at least at code level.

Optimization tier

- Performs (mod) caching.
- Increases robustness.
- Helps controlling scaleability factors (aka bottlenecks).

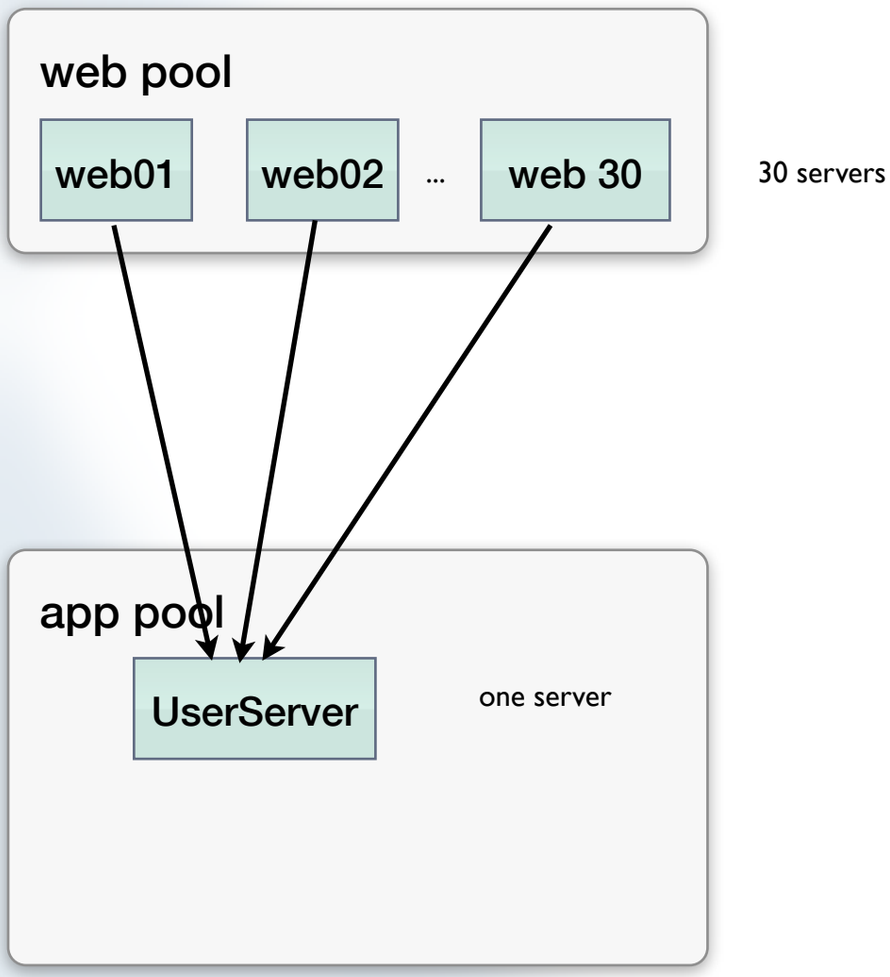
Mod caching / Robustness



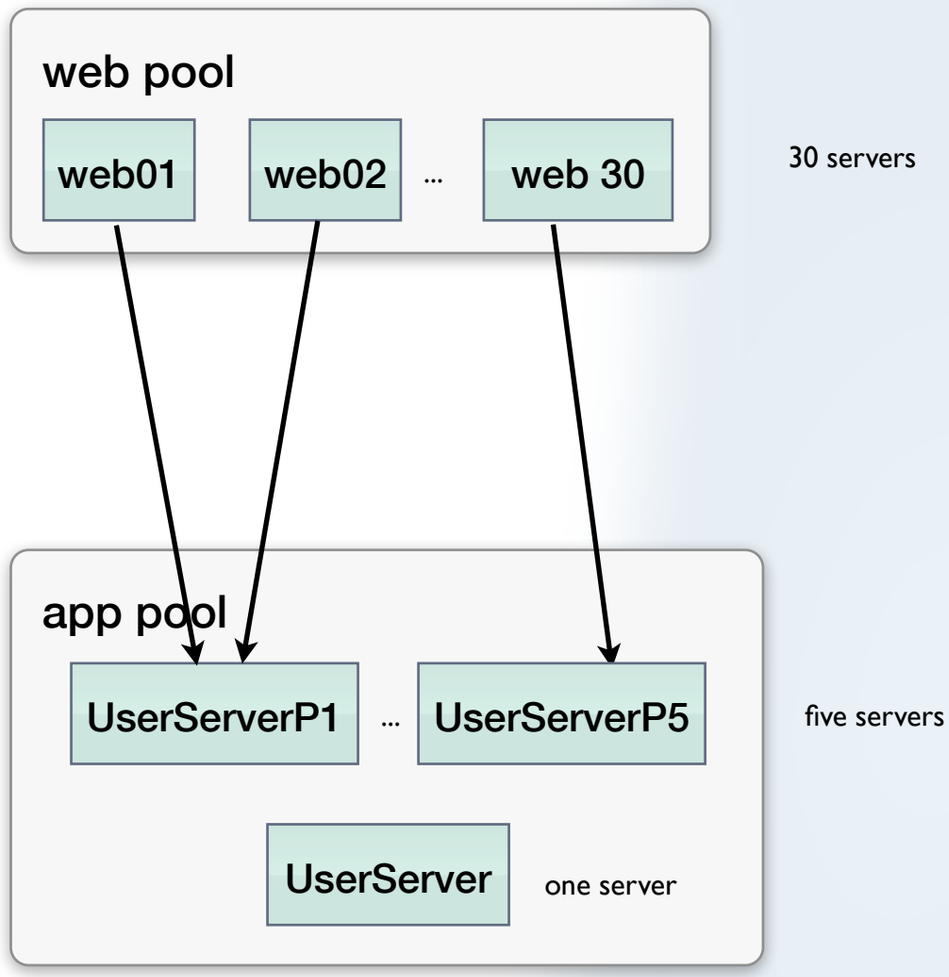
mod-ing is lineary dividing calls on proxies based on call context to achieve most linear distribution.
Two most common approaches are
a) mod by parameter (i.e. userid)
This method allows a proxy to perform a 100% cache of a subset of the data.
b) mod by source (i.e. servername)
This method reduces the consequences of a component failure and reduces the number of affected nodes
See the Performance Tuning for Portals presentation for details on caching and moding.

Scaleability factors

Classical approach,
factors 1:30



Optimized approach,
factors 1:5:30



...

Scaleability factor

- Number of affected components/nodes in case of a short outtage of a component/node.
- Low scaleability factors indicate robust systems which can deal with short peaks and outtage. However the system still should be able to handle the regular load.
- Classic approach with factor 1:30 means that a short outtage (for example a gc run) in the userserver will affect 30 components.

Scaleability factor (II)

- Optimized scaleability factor 1:5:30 means that, a short outtage in a userserver proxy affects only 6 webservers and a short outtage in the userserver itself only affects the 5 userservice proxies which should be capable of handling the short outtage internally through caching.

Architecture IV (nT)

- Scaleability on the front side is reached by the nTier architecture and optimization layer introduced by architecture III.
- However all deployment models contains a single db.
- The goal is now to add an additional separation in the db tier.
- If you modeled properly - this is a matter of deployment, not code.

Arch IV - Node Example