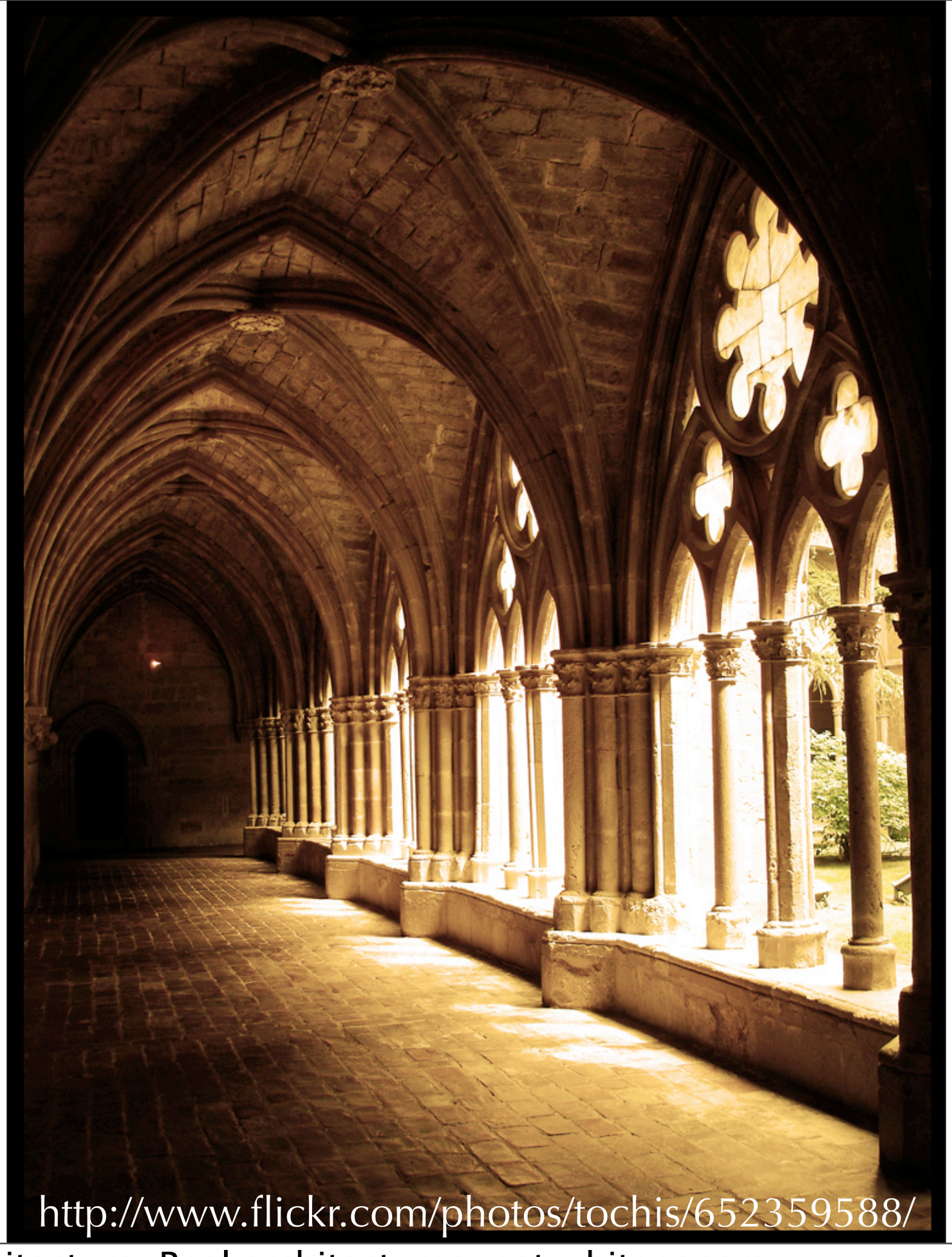
GET /Resty HTTP/1.1

Host: phillyemergingtech.com

Accept: application/vnd.etec-09.tech-talk

- Brian McCallister

Hi, I'm Brian. Talk about ning, apache, etc so they have context for my statements. Development as product, not as IT (though I have done both, I prefer being a profit center to a cost center).



About systems architecture. Real architectures, not whitepapers.

The technical stuff critical to the business which the pure business people do not understand.

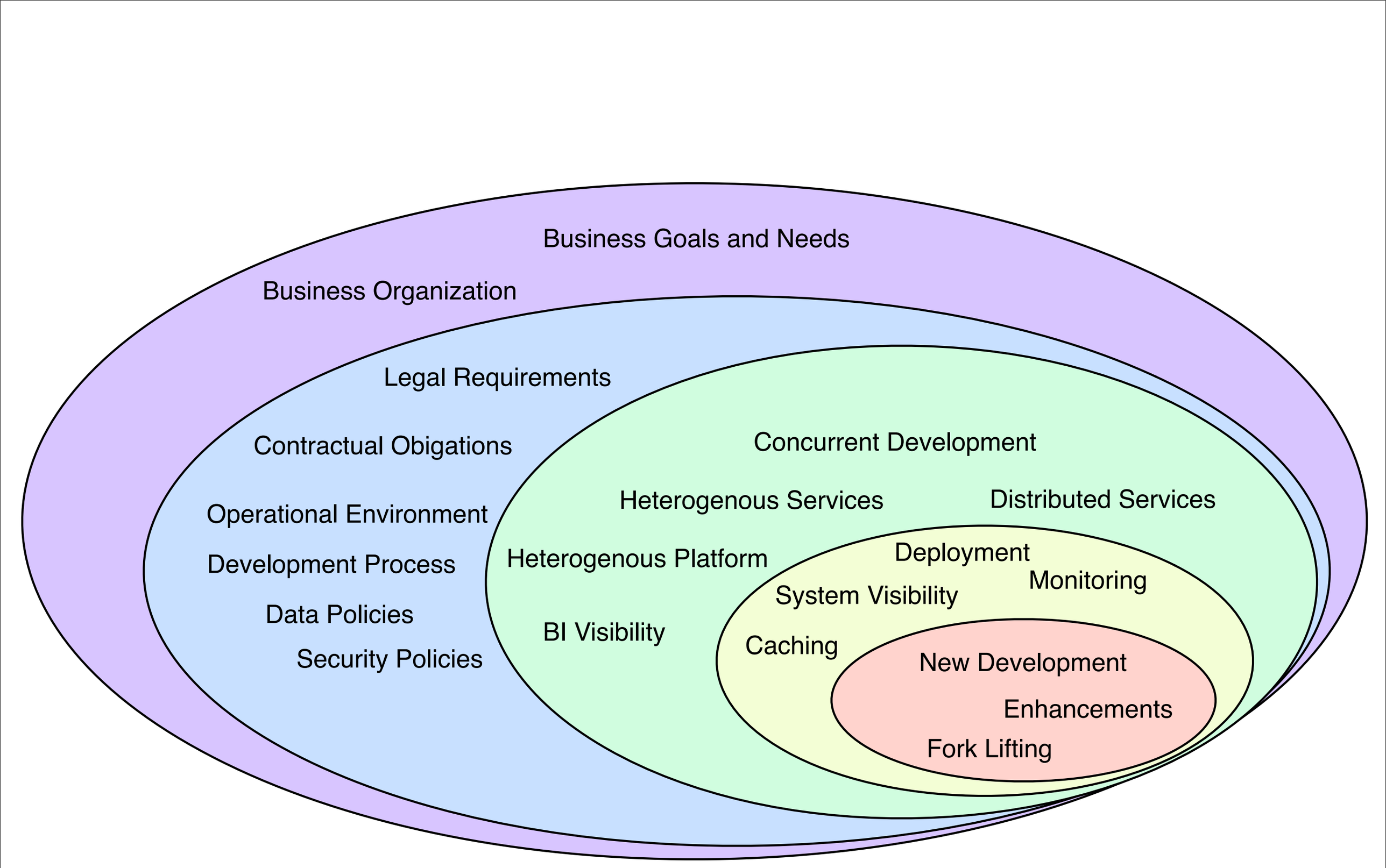
Set of principles which guide the development of components and interactions.

The tools and practices which support the principles and interactions desired.

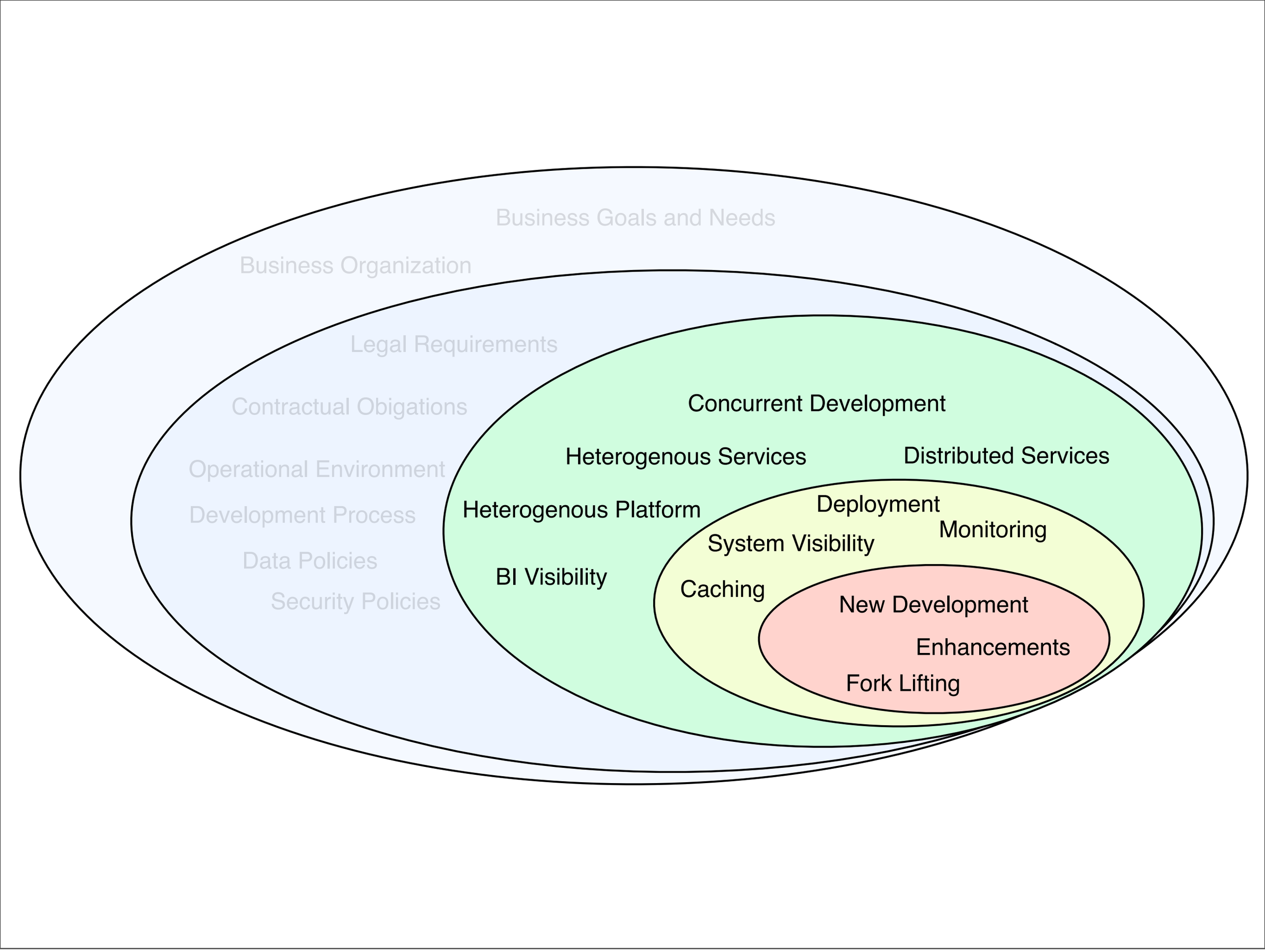
Patterns

(Alexander, not GoF)

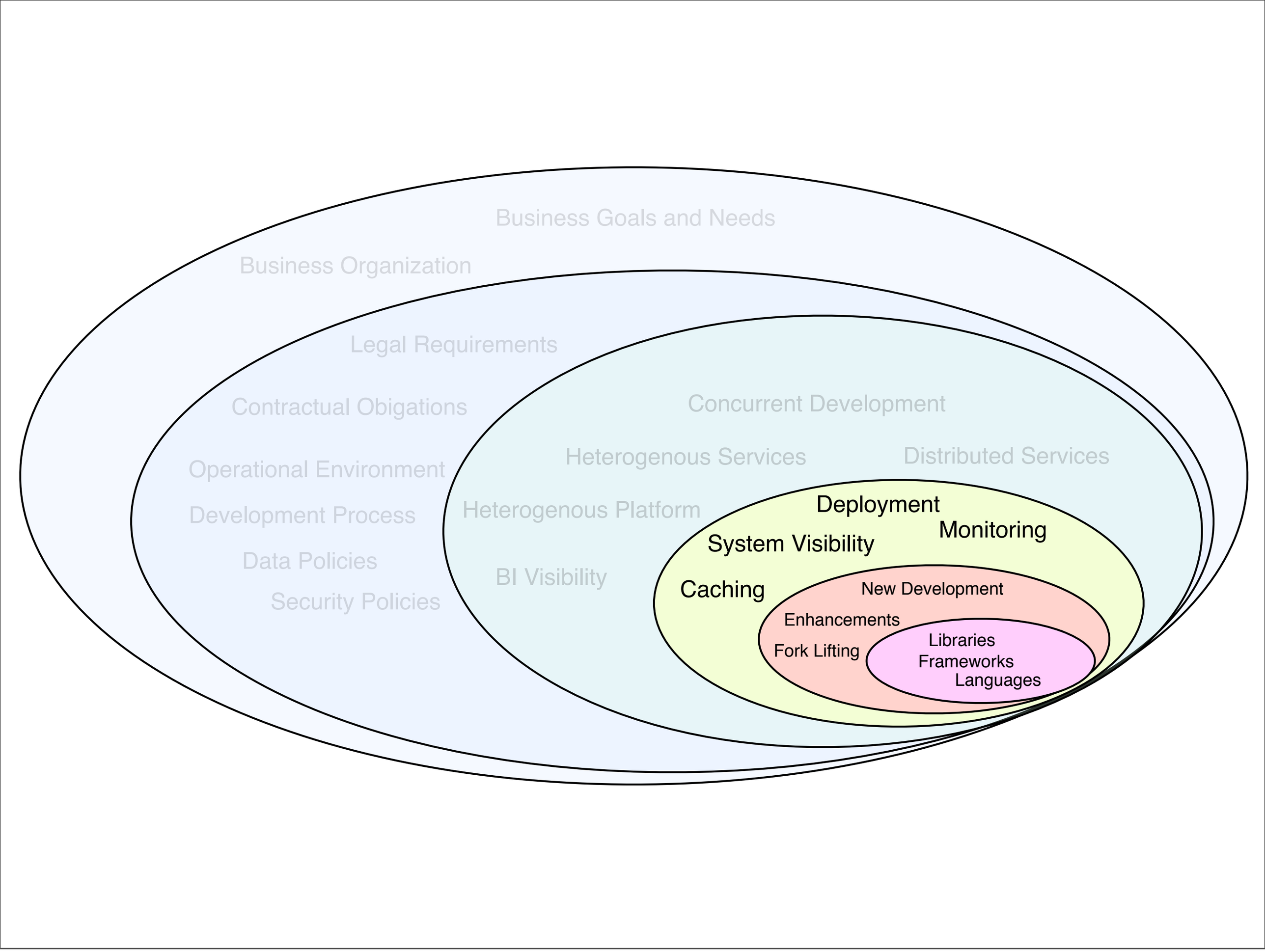
Alexander -- independent regions and distribution of towns to lighting and pictures of family.



As an architect, this is what your scope of concern is.
Stuff in outer constrains stuff in inner.



As a systems architect, this is what you can generally control, outer stuff you can just influence.



As an applications architect, you tend to go further down, but have additional constrains from that darned architecture group.

REST and HTTP

Okay, so REST -- my assertion is that it serves you well as you grow and change.

```
HEAD /get-resty-ete09.pdf HTTP/1.1
Host: skife.org
```

```
HTTP/1.1 200 OK
Date: Fri, 27 Mar 2009 04:53:35 GMT
Server: Apache
Last-Modified: Fri, 27 Mar 2009 04:52:22 GMT
ETag: c8058-61656a-466127f896d
Accept-Ranges: bytes
Content-Length: 6382954
Content-Type: application/pdf
```

Reminder of some of the features, point out resources, uniform interface

GET /get-resty-ete09.pdf HTTP/1.1
Host: skife.org
If-None-Match: c8058-61656a-466127f896d

HTTP/1.1 304 Not Modified
Date: Fri, 27 Mar 2009 05:05:05 GMT
Server: Apache
ETag: c8058-61656a-466127f896d

```
HEAD http://skife.org/get-resty-ete09.pdf HTTP/1.1
Host: skife.org
```

```
HTTP/1.1 200 OK
Date: Fri, 27 Mar 2009 15:32:40 GMT
Server: Apache
Last-Modified: Fri, 27 Mar 2009 05:17:46 GMT
ETag: 128015-618406-46612da5fd
Accept-Ranges: bytes
Content-Length: 6390790
Content-Type: application/pdf
Proxy-Connection: Keep-alive
```

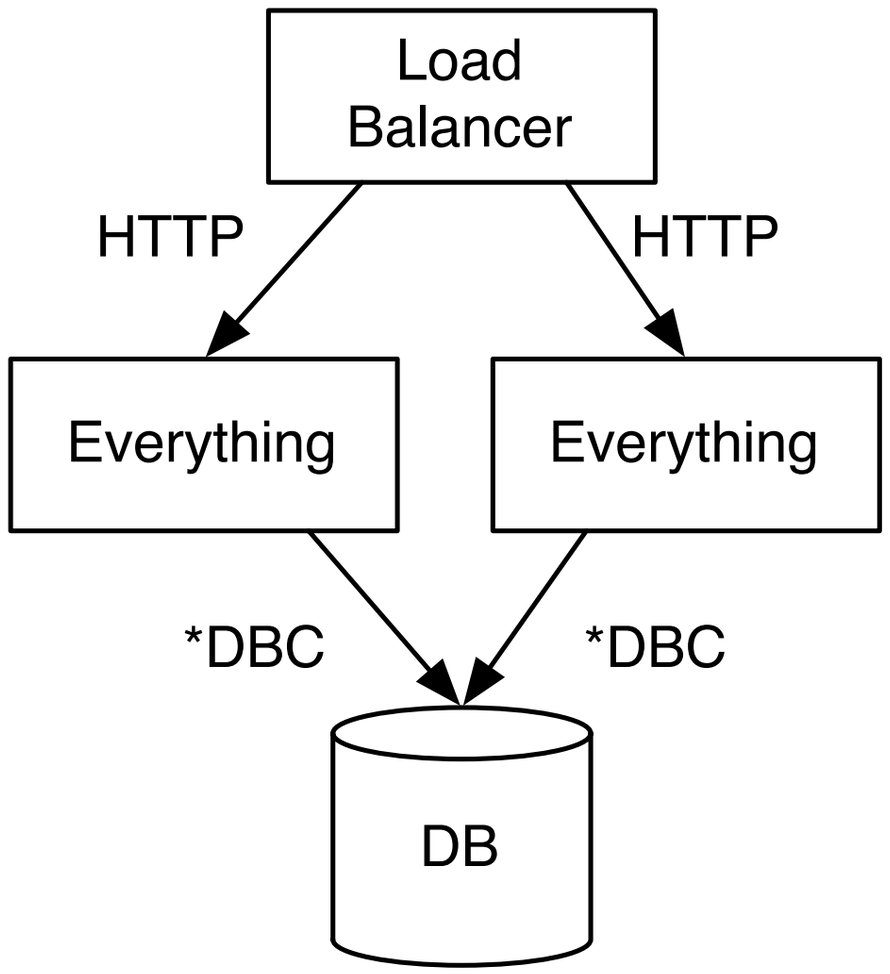
Composability, proxies, more than just client server. This proxies through Charles, a alternately transparent or traditional debugging proxy.

```
<html xmlns="http://www.w3.org/1999/xhtml"
        lang="en" xml:lang="en">
<head profile="http://gmpg.org/xfn/11">
  <link rel="alternate"
        type="application/atom+xml"
        title="Brian McCallister's Waste of Time"
        href="http://kasparov.skife.org/blog/index.atom" />

  <link rel="openid.server" href="http://openid.skife.org/" />
  <link rel="openid.delegate" href="http://openid.skife.org/" />

  <link rel="me" href="http://twitter.com/brianm" />
  <link rel="me" href="http://www.ning.com/brianm" />
  <link rel="me"
        href="http://www.linkedin.com/in/brianmccallister" />
```

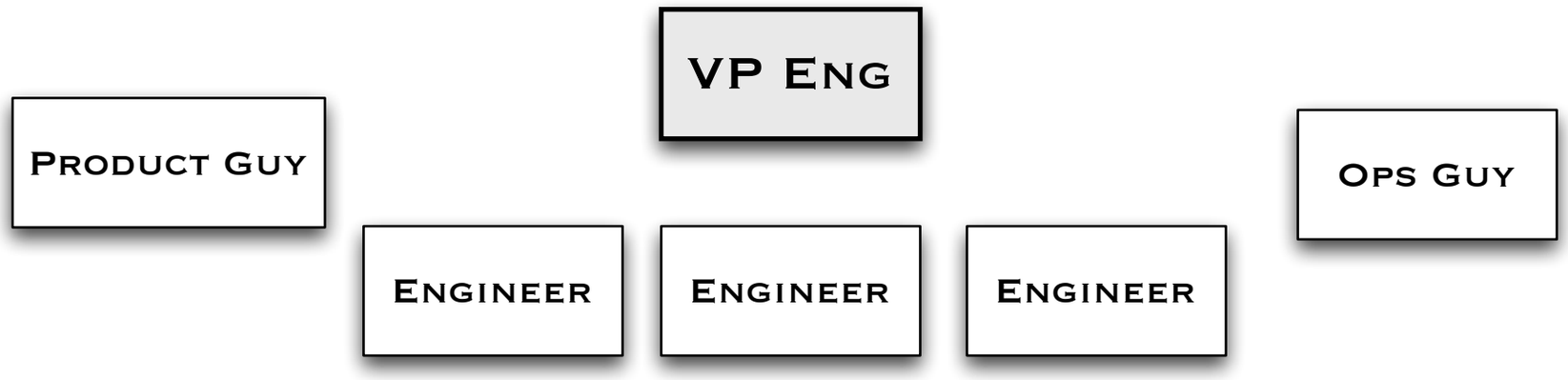
Hypermedia, resources reference other resources via that uniform interface.

The Business

All of this is in the context of what is right for the business -- and this changes, sometimes from day to day



Everything *starts* really simple.

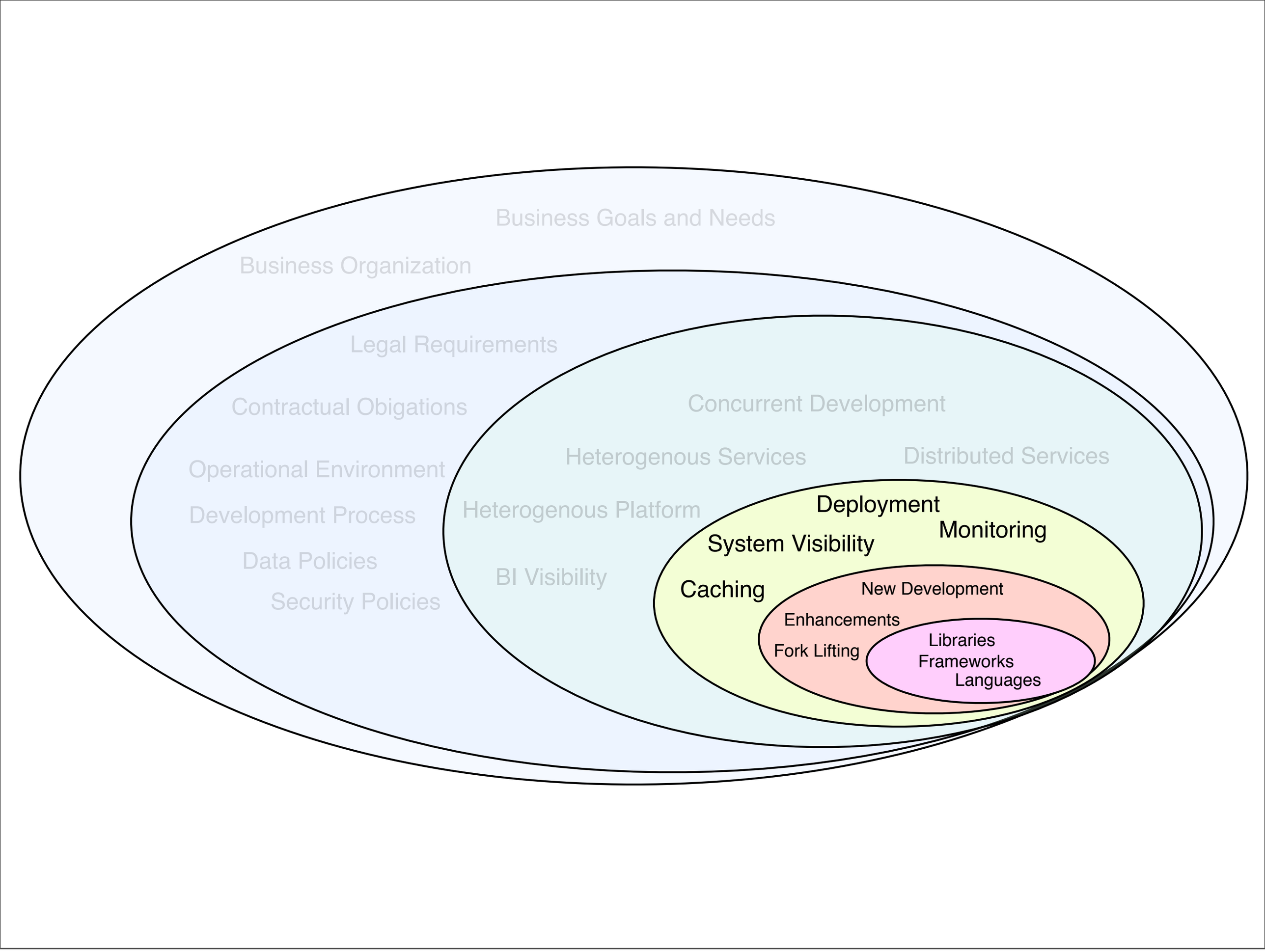


And everyone knows what everyone else is doing.

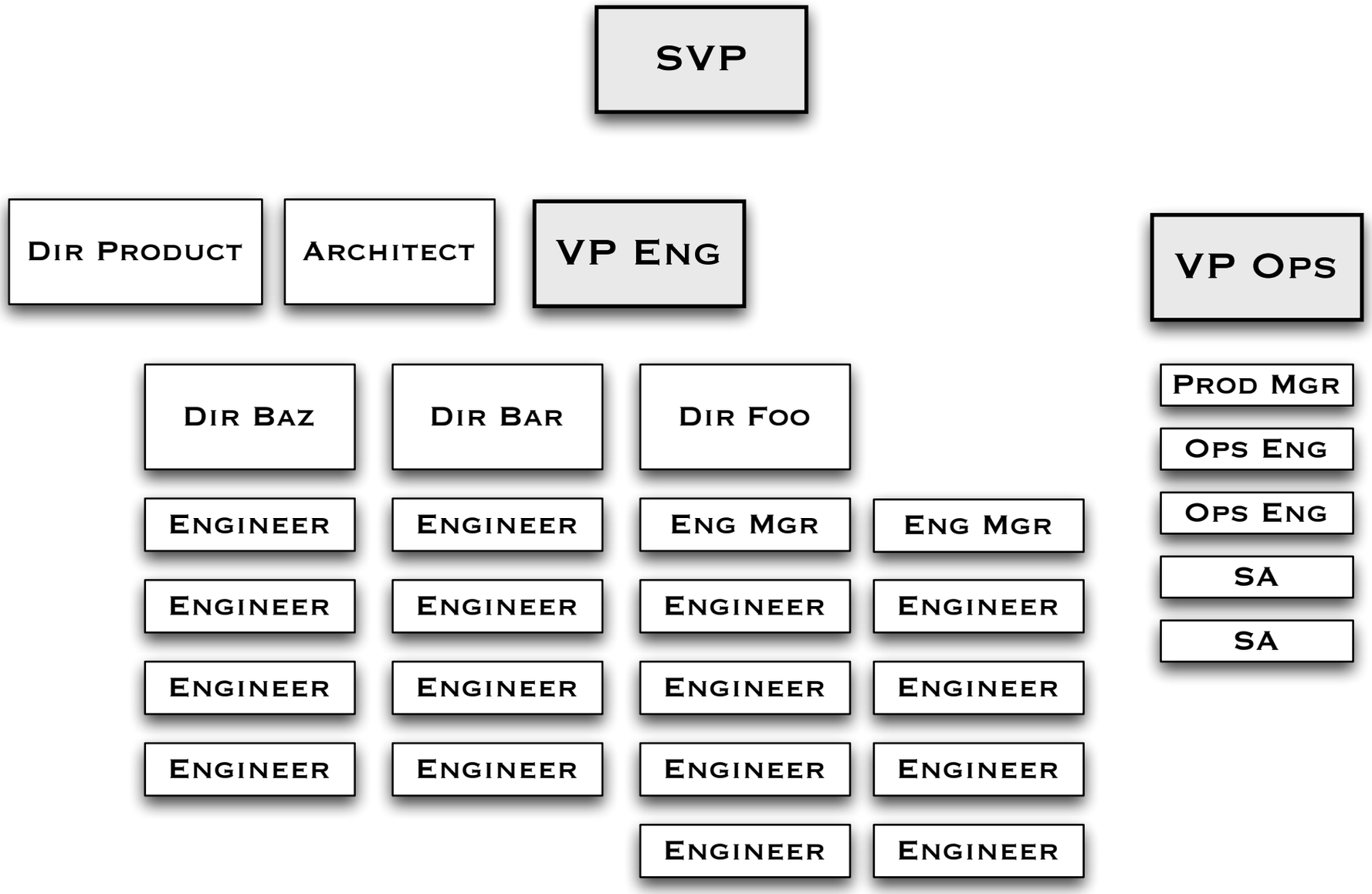
Time is not on our side

(cc) http://www.flickr.com/photos/dipster1/1403240351/

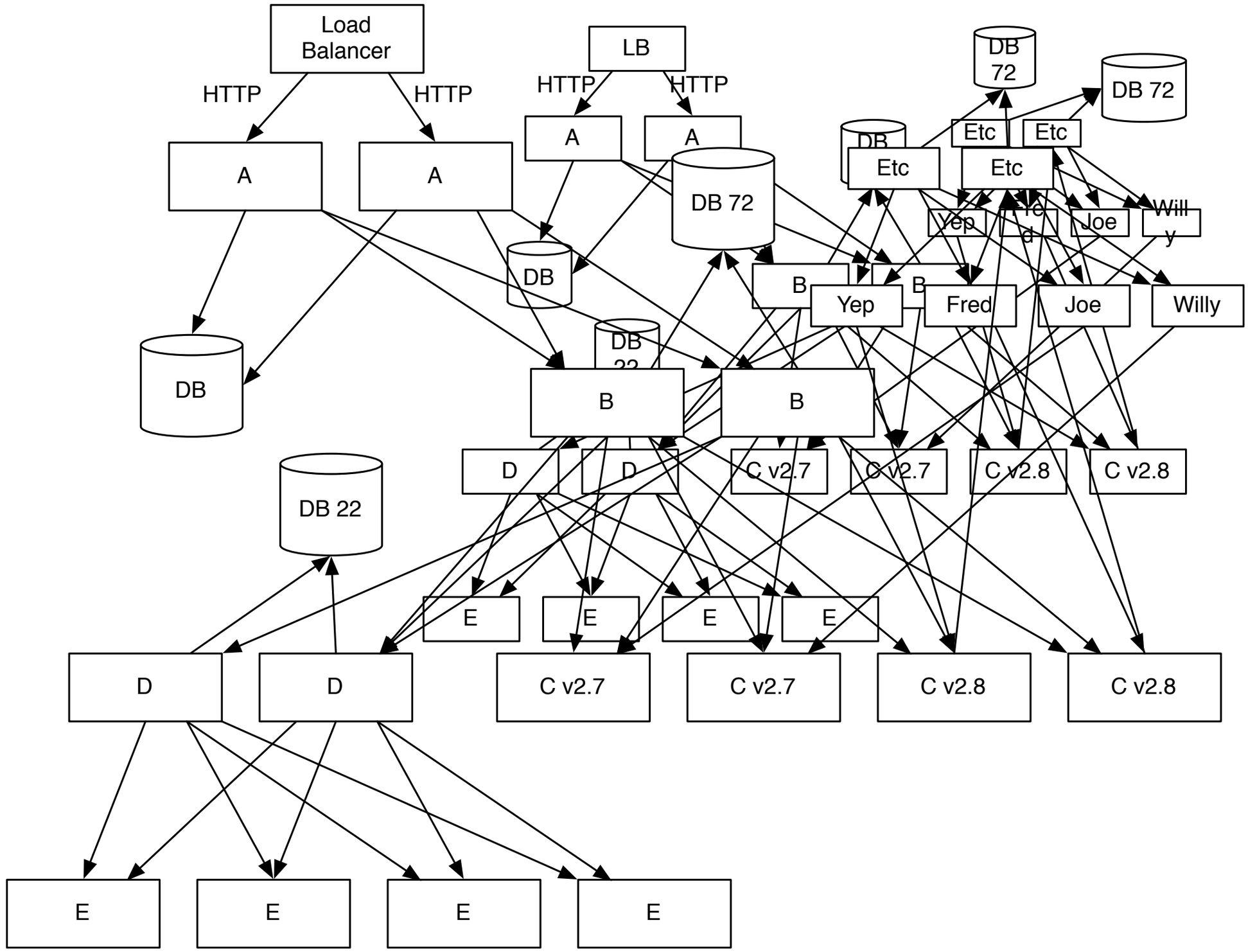
We want existing, easy to use tools. Spending time writing things not core to your business is generally very, very wrong. You want maximum leverage, here.



Our architectural effort is focused here, right now. Our big concerns architectural concerns are django, rails, or jersey, keeping downtime measured in hours instead of days, and cranking out features. Visibility (sniff) monitor (http ping), deploy (vlad, capistrano, copy tarball, change symlink, whatever)



Complicating matters, we have been hiring to keep up with growth and iterate on features faster. We have an engineering team mostly focused on non-product features, in fact! One of the teams wants to use haskell, and makes a good case for it.



Of course the system has grown, at this point no one actually understands all of it, though various people believe they do.

“Anarchic scalability refers to the need for architectural elements to continue operating when they are subjected to an unanticipated load, or when given malformed or maliciously constructed data...” -Roy

Uniform Interface

GET, POST, and those others

```
{
  "self" : "http://network-search/1.0?q=clojure",
  "from" : 0,
  "to" : 99,
  "total" : 1,
  "results" : [ {
    "url" : "http://orlambda.ning.com/",
    "metadata" : "http://app/application/2.7/orlambda"
  } ]
}
```

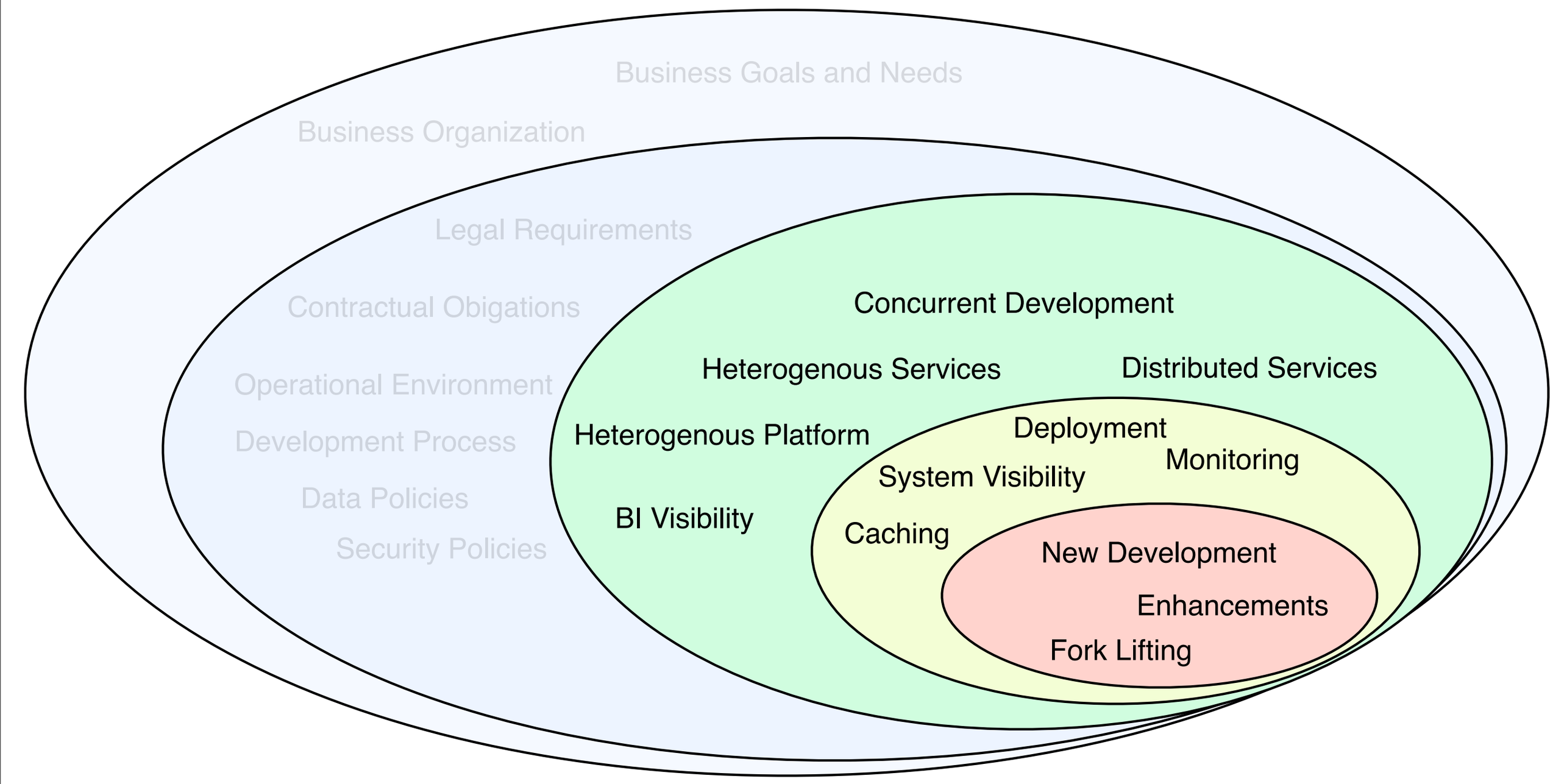

Hypermedia

Hypermedia doesn't mean HTML -- is media linking to other media. Interfaces for the links are all the same (uniform interface). JSON, Thrift, ProtoBuff, Atom, all are valid forms of hypermedia. Spend more time designing media types, less designing URIs.

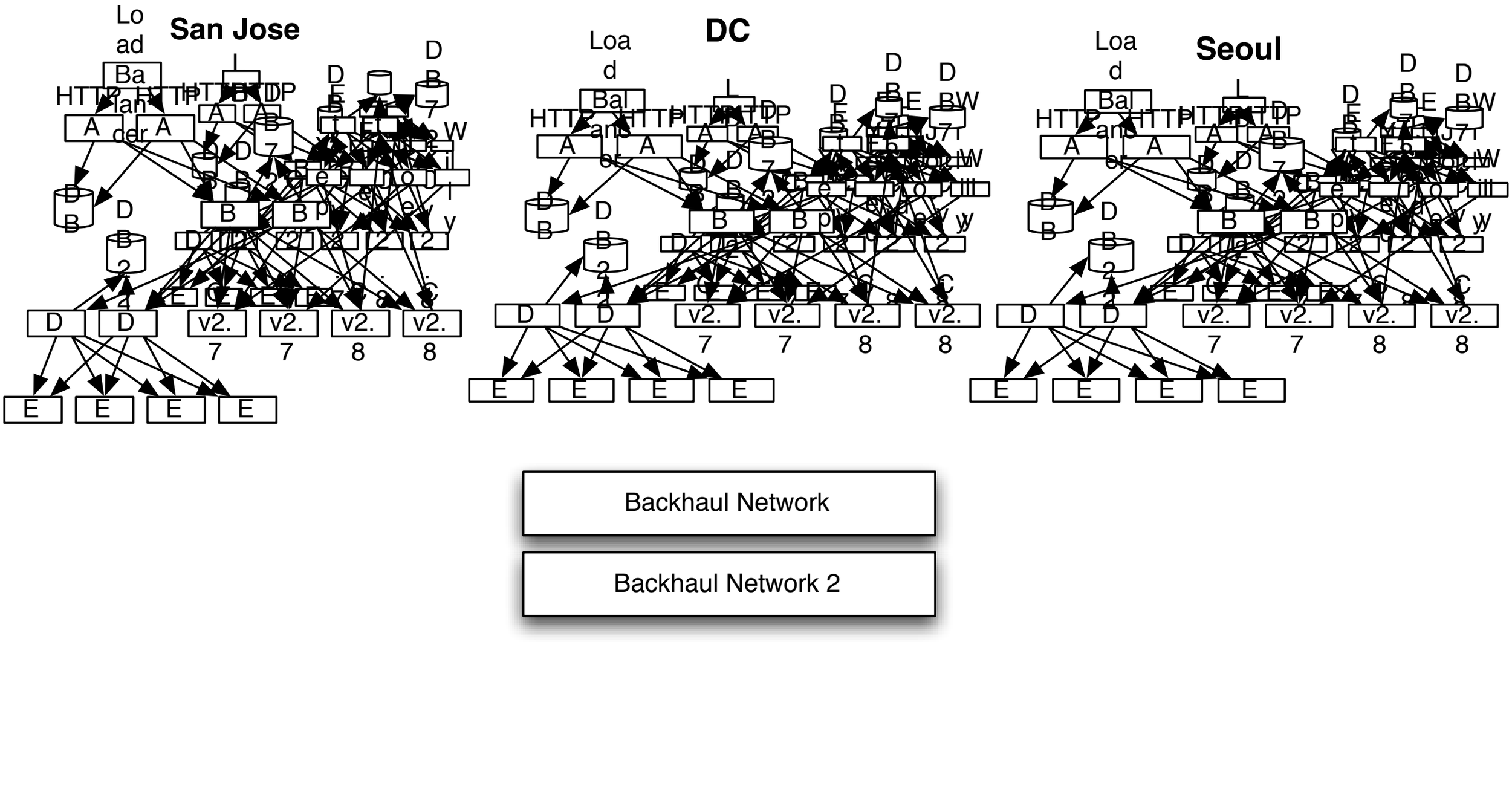
~~Ids~~

URIs

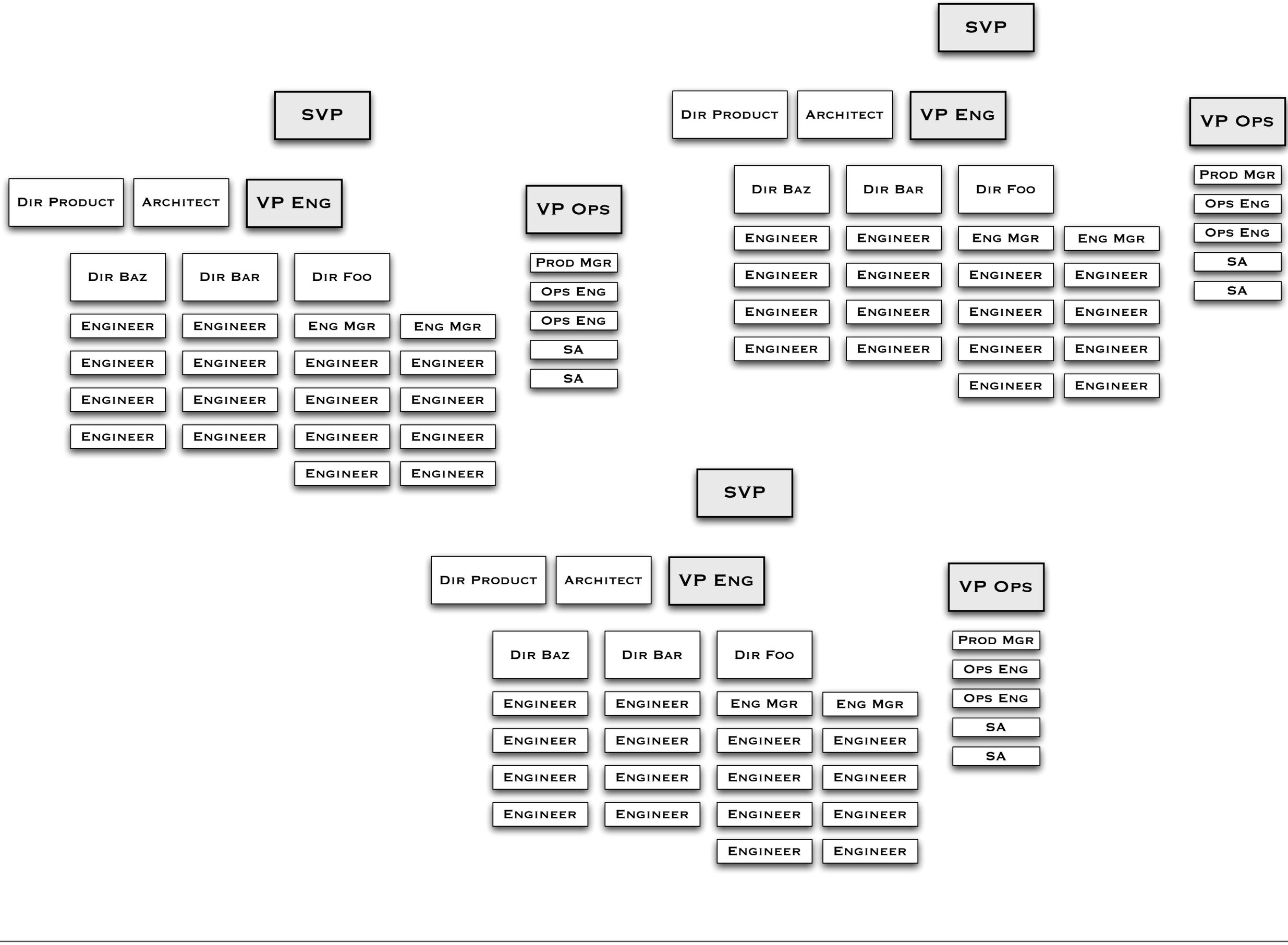
If it is shared between components, make it a URL.



Our architectural concerns now expand to include nasty things like the business wanting to track signup conversion rates, the team that insists they need haskell on freebsd, and the fact that we have four teams on different schedules.



At this point your infrastructure *is* an internet.



Then they reorganized engineering by business unit. Multiple products, possibly multiple markets.

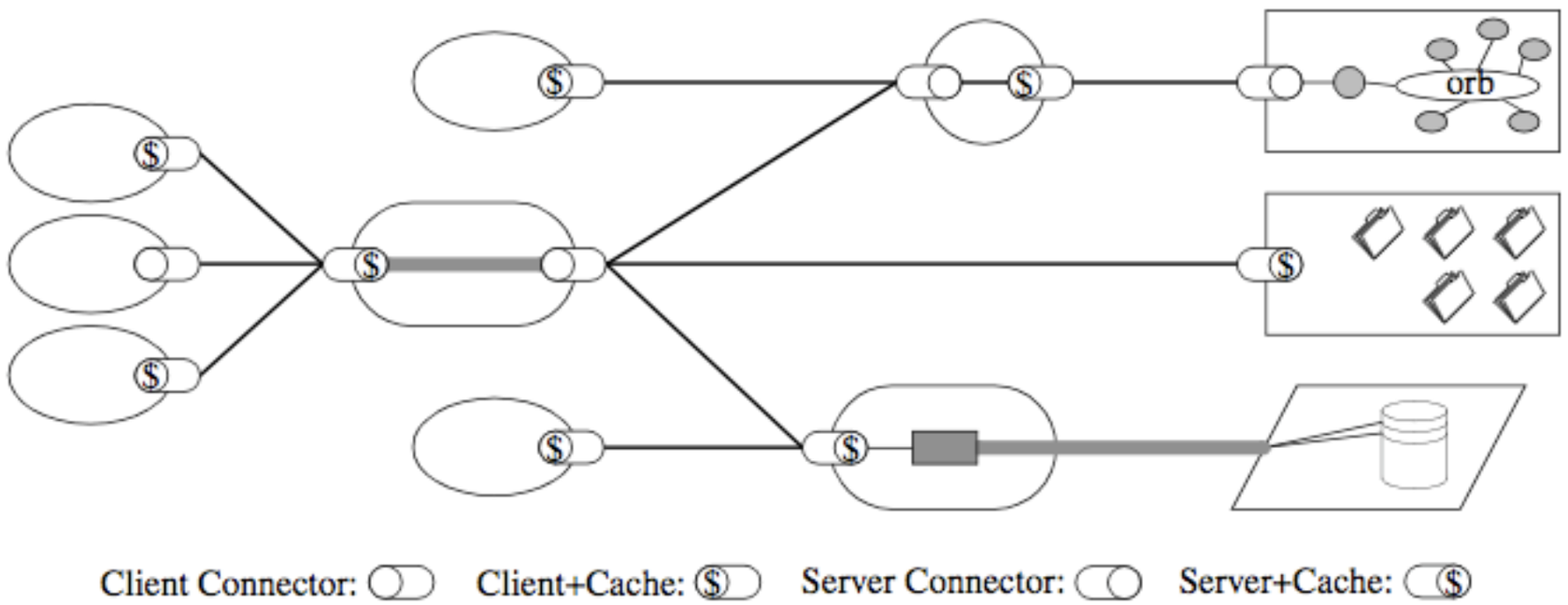


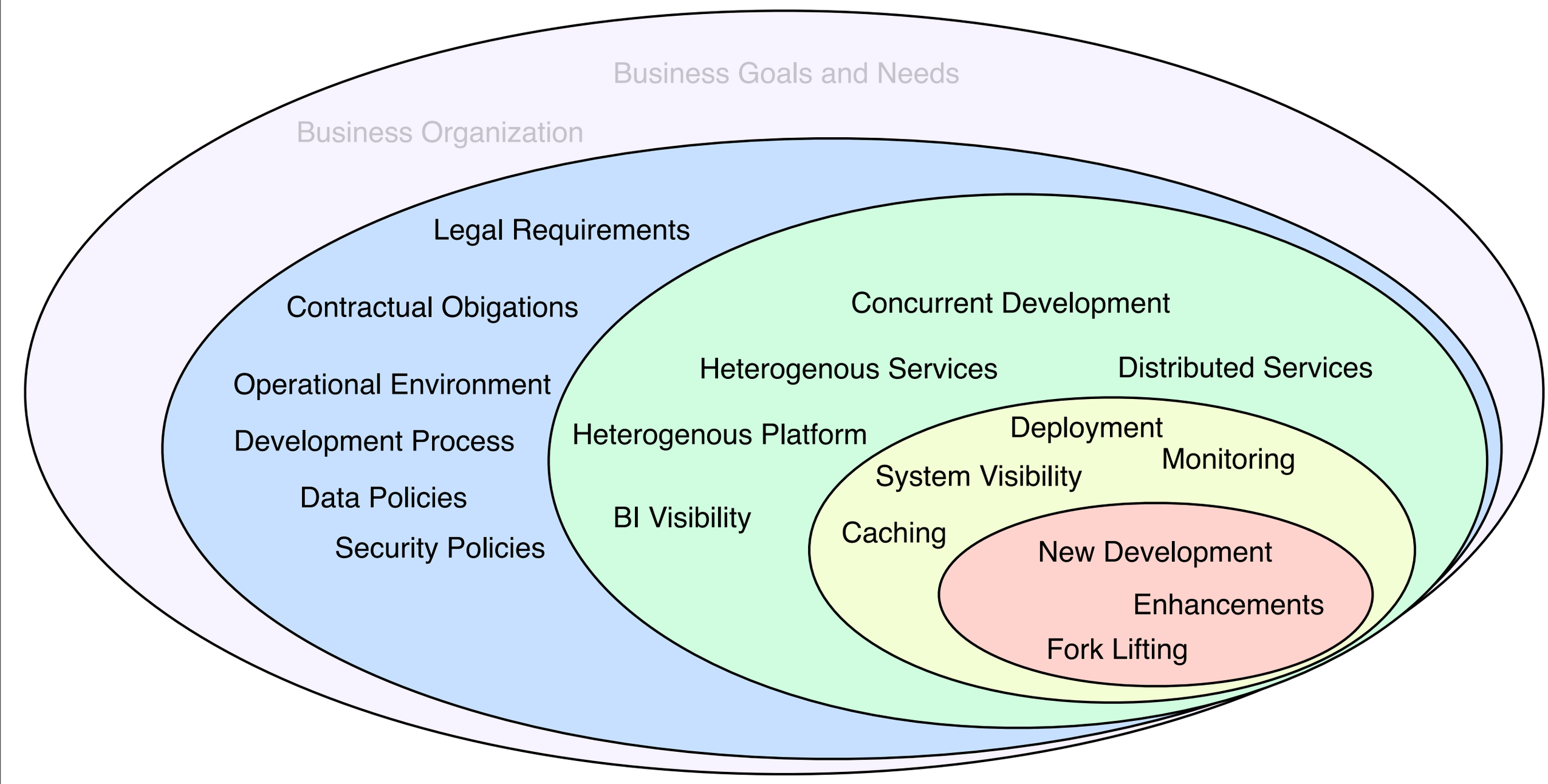
Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

(c) Roy Fielding

At this point the peer-to-peer aspects of the *protocol* save your ass.

http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html

Caching is your friend



You have no control over these, still, but there are *significant* constraints caused by the blue stuff now, and you spend a LOT of time worrying about them.

(c) Brian McCallister

A major theme here is that things change. Decisions made early on have major consequences. Your architecture needs the flexibility and extensibility to adapt as the business grows. An analogy I like to use is why folks use more ruby than haskell. When the tool is “done” there are about the same number of lines of code, the difference is that the ruby one got longer as you worked, and the haskell one got shorter until it finally compiled.

GET /Resty HTTP/1.1

Host: phillyemergingtech.com
Accept: application/vnd.etec-09.tech-talk
Location: http://skife.org/get-resty-ete09.pdf

Brian McCallister
http://skife.org
brianm@skife.org

Thank you!

Thank You, Sponsors

MANNING
PUBLICATIONS

Thank them too!