# Flexibility v Performance
## (caching options in UML and ArchiMate diagrams)

After Gerben Wierda's paper

http://eapj.org/on-slippery-ice-20150201

(The link will break if the EAP journal move the paper.)

# Flexibility v performance trade off

**The company's enterprise architecture principles**

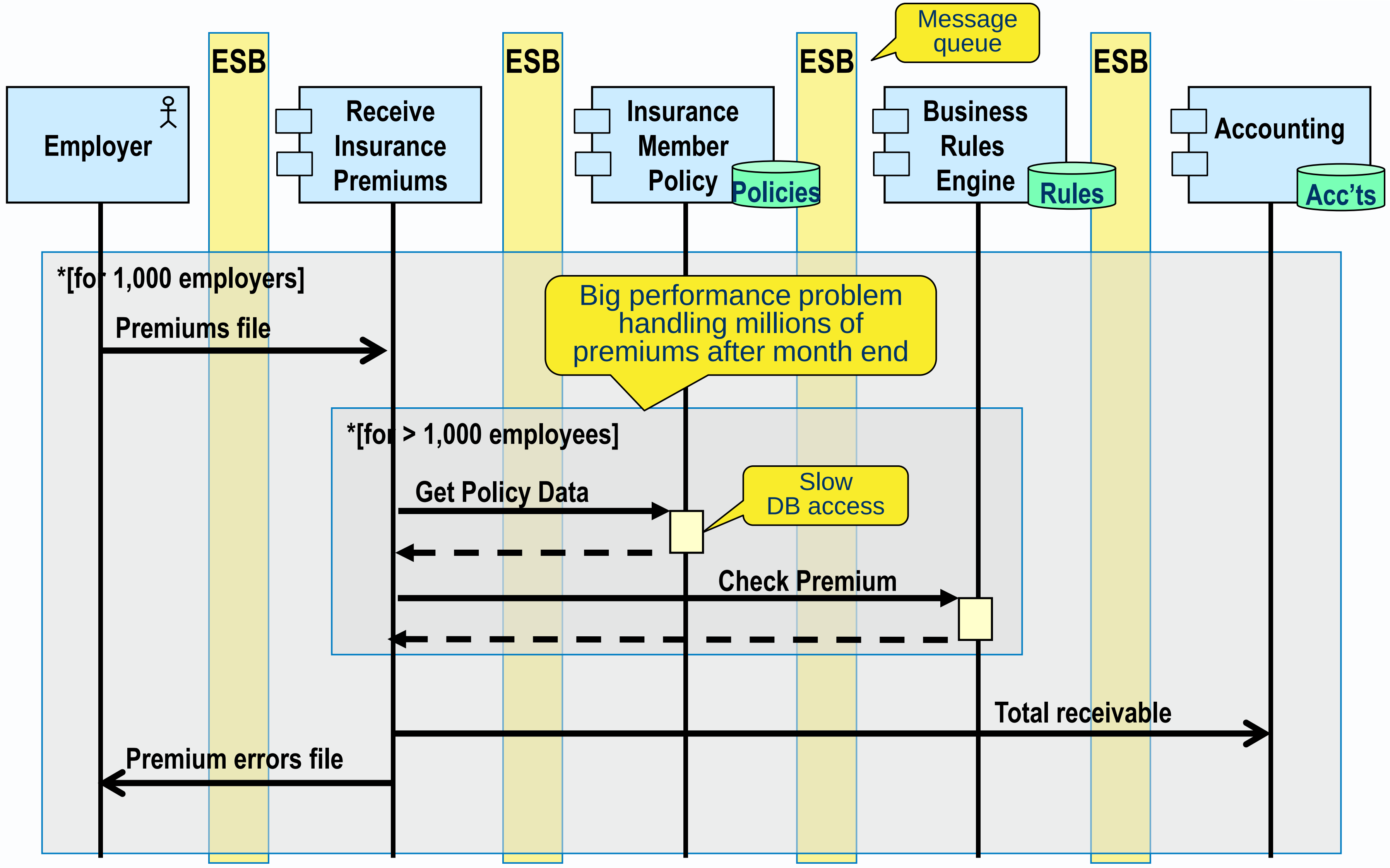
1. presentation layer is separated from application layer
2. data is owned by one business function and served to others
3. data is mastered in one application (single source of truth)
4. systems are connected via an ESB for loose-coupling
5. rules are applied by business rule engines so systems can be adapted to different policy types and different clients.

Designed for
On-line transaction use cases
Flexibility w many customer and/or product variations

- "Alas, such models are no silver bullets.
- Projects get into serious trouble because of them.
- Three main reasons:
  - brittleness of loosely-coupled spaghetti
  - maintainability
  - performance."

Gerben Wierda

# Ineffficient solution design to company principles



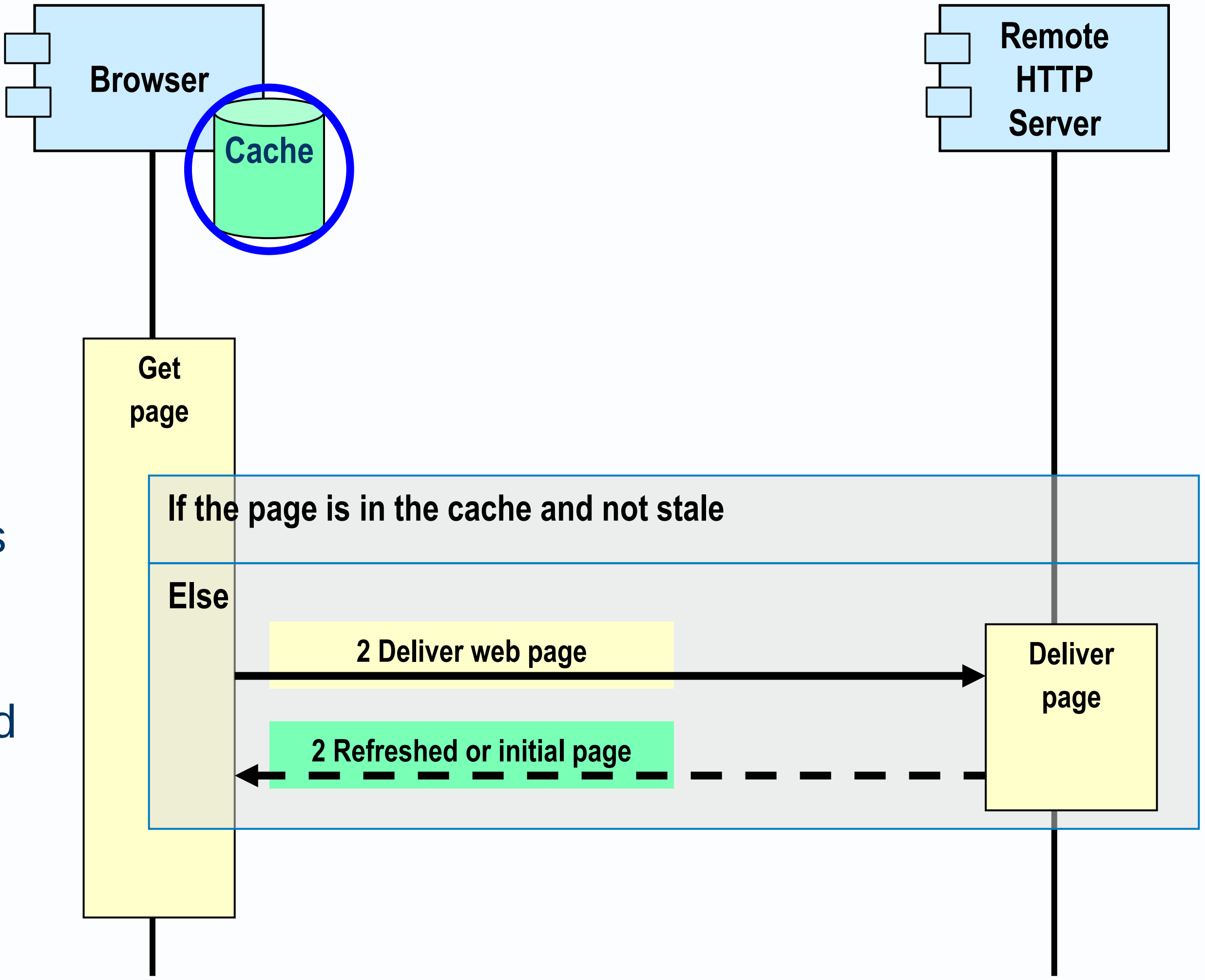
## What to do?

- Performance (when processing millions of premiums in a the first few days of a month) was unacceptable.
  - Bandwidth not the problem
  - The duration of processing steps was the problem
- Gerben explores three caching options – to follow
- What else could be done?
- More generally, how to prevent problems created by
  - naïve implementation by software architects of
  - naïve EA principles?

# “Classic client-maintained cache ”

(see appendix for ArchiMate version)

Avancier

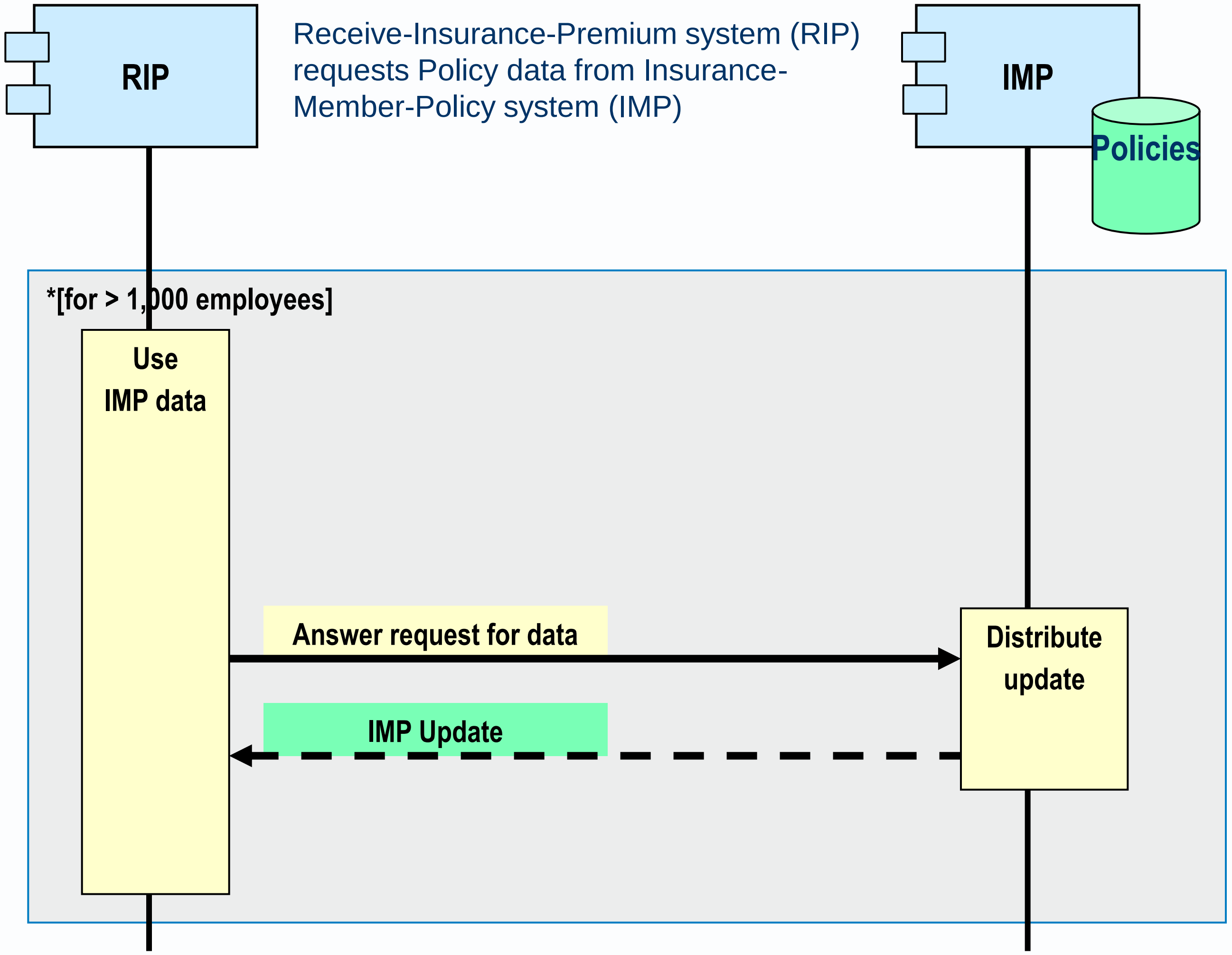


1. Browser looks first in cache
2. Page retrieved
3. Page cached

# "Original design (no cache) "

(see appendix for ArchiMate version)

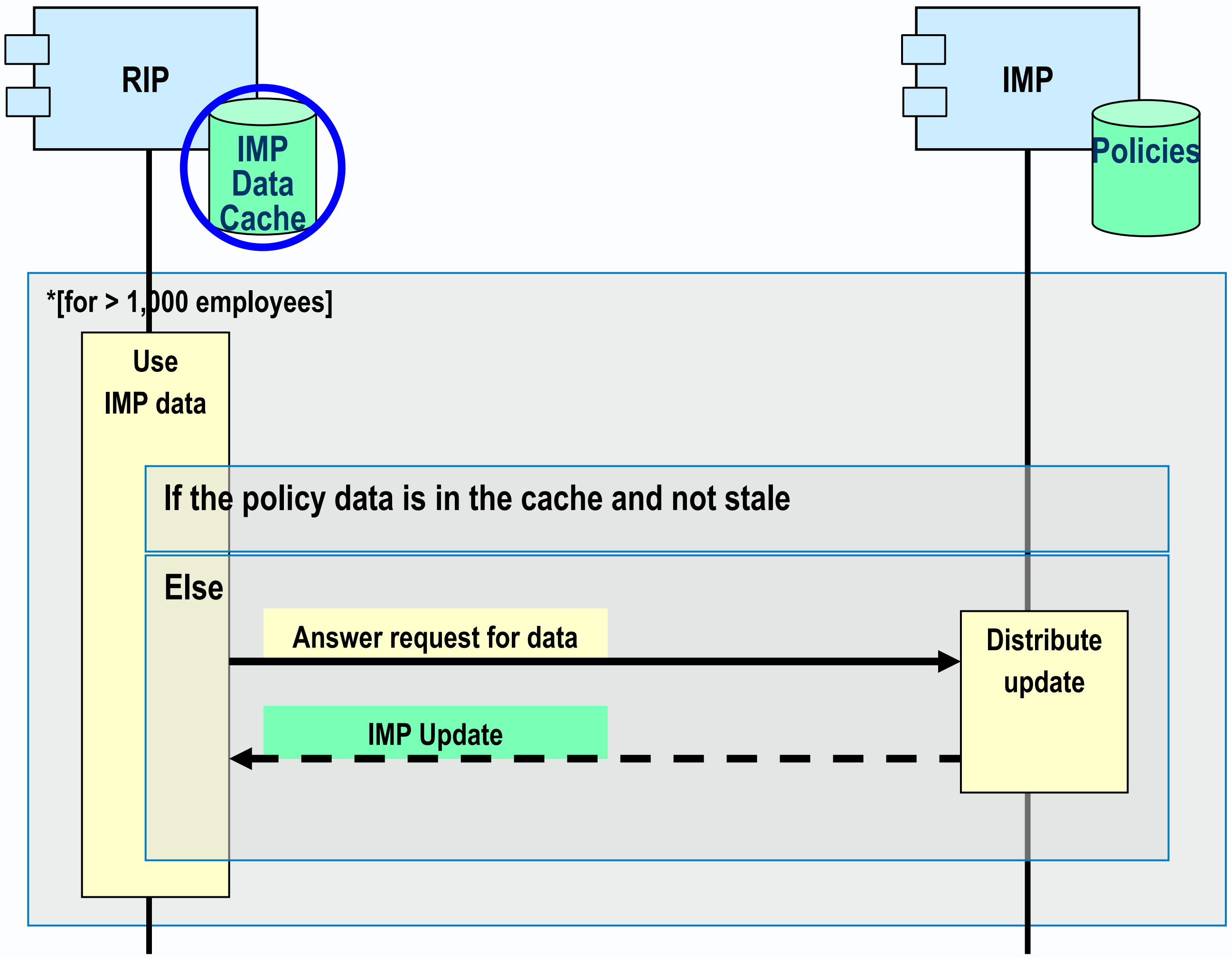
Receive-Insurance-Premium system (RIP) requests Policy data from Insurance-Member-Policy system (IMP)



# “Classic client-maintained cache ”

(see appendix for ArchiMate version)

Avancier



RIP first looks in the cache for data that is not marked out-of-date.

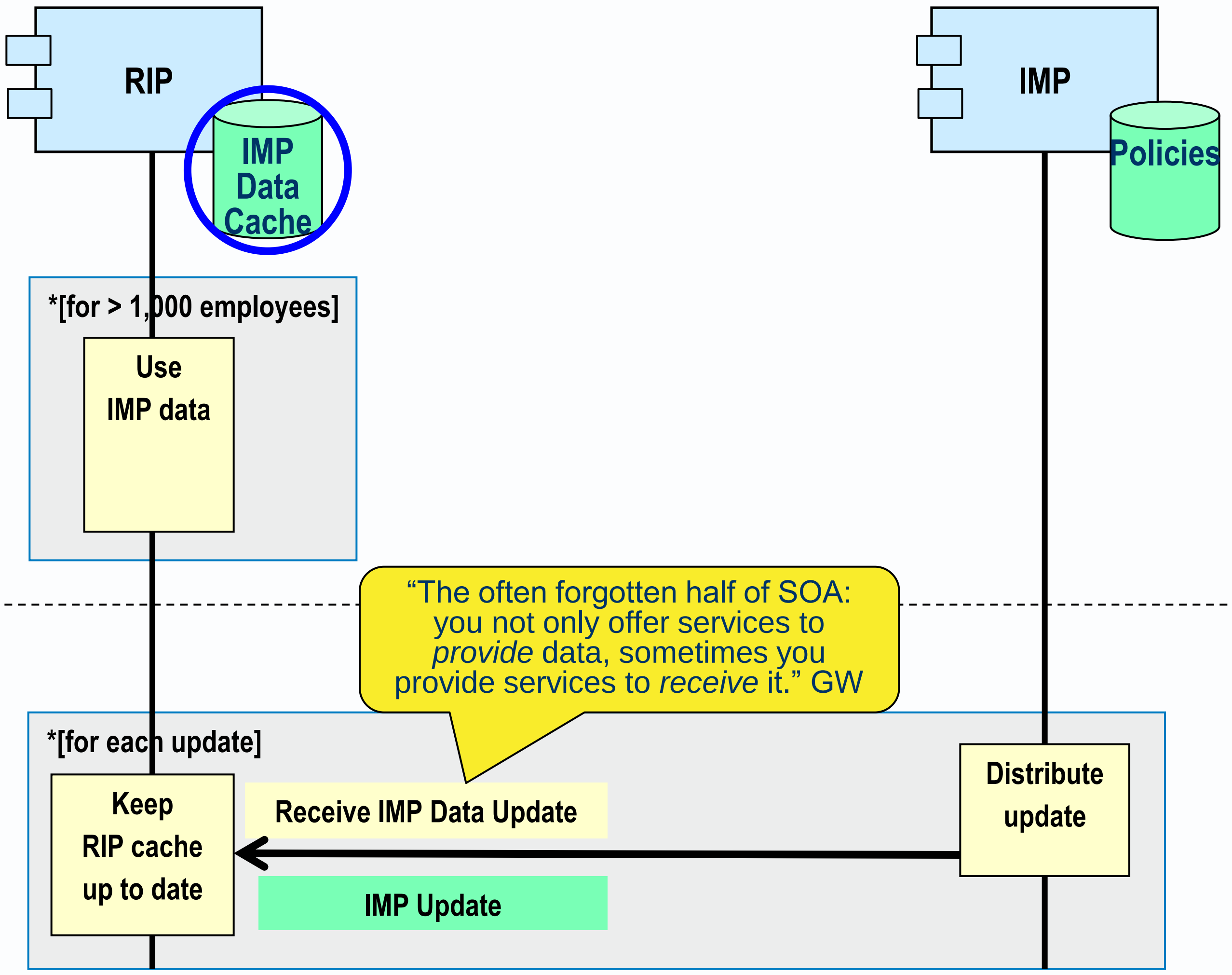
If the data needs to be refreshed, RIP uses the service from IMP and receives the data

The problem is that we might be using data that is out of date.

# “Server-updated cache ”

(see appendix for ArchiMate version)

Avancier



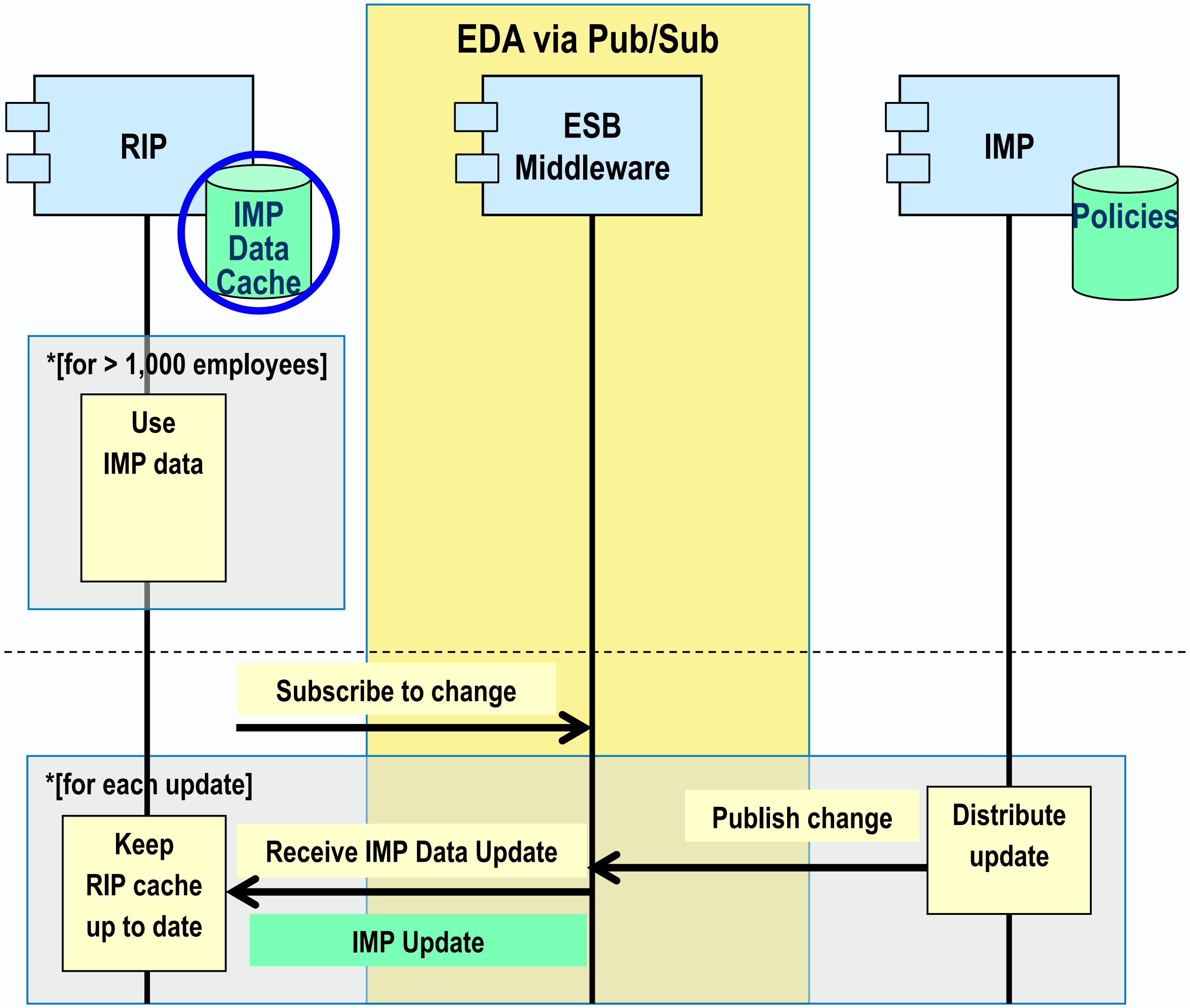
RIP
IMP Data Cache
IMP
Policies
*[for > 1,000 employees]
Use IMP data
*[for each update]
Keep RIP cache up to date
Receive IMP Data Update
IMP Update
Distribute update
“The often forgotten half of SOA: you not only offer services to *provide* data, sometimes you provide services to *receive* it.” GW

RIP uses only the local cache which is kept up to date by IMP.

Every time IMP updates data, it uses a service from RIP and sends it the updated data. RIP immediately stores it in the cache.

## Server-updated cache via ESB

Avancier



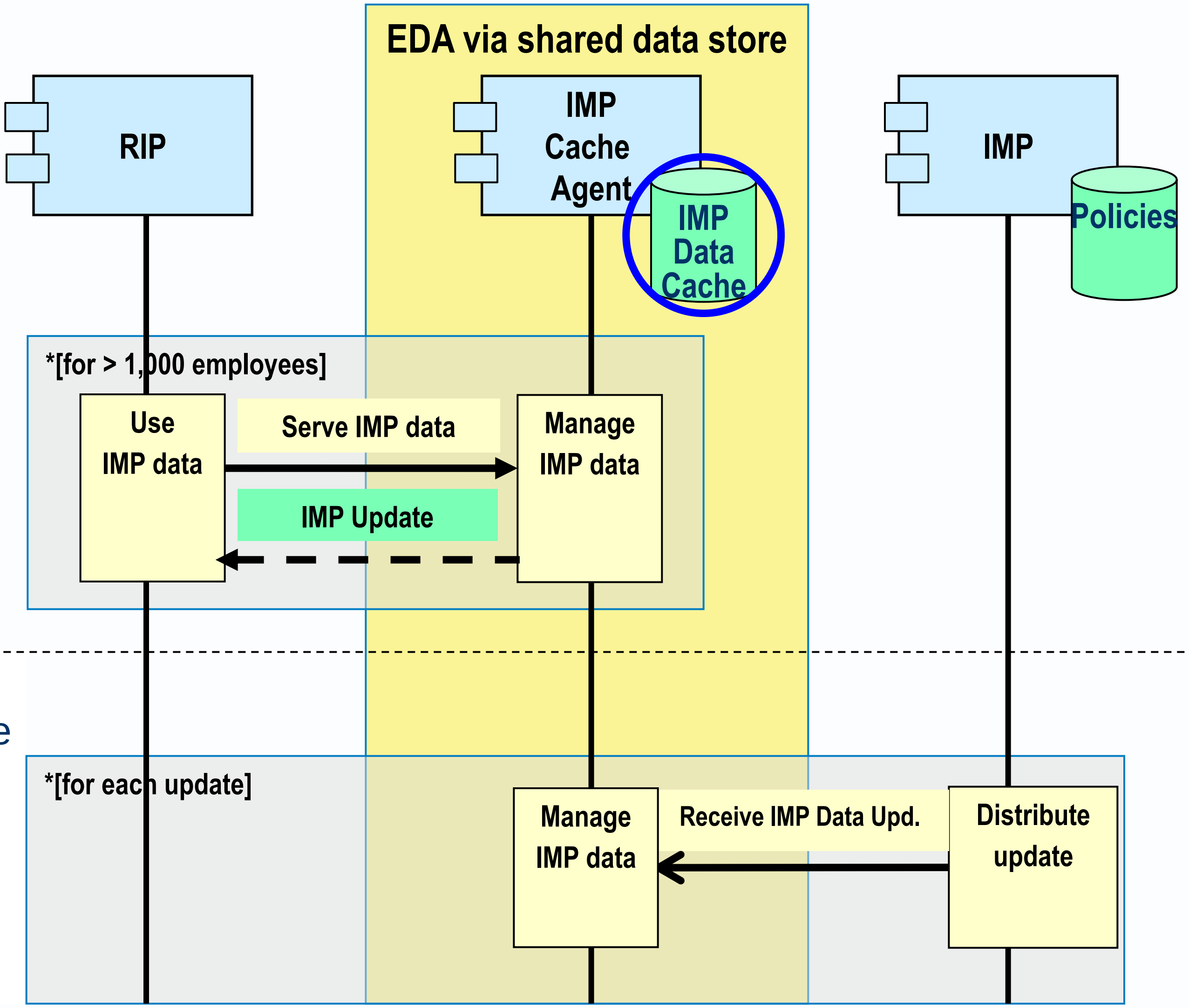
RIP uses only the local cache which is kept up to date by IMP.

IMP publishes changes on the ESB. RIP subscribes to them, storing them locally and using them as needed.

# "Server-owned cache"

(see appendix for ArchiMate version)

Avancier



3rd app manages the cache made available to RIP via API

It runs on the environment where the cache must be maintained

## There is no silver bullet

- Distributed programming and OO design patterns
  - have created far more complex programs than earlier.

- Naïve interpretations of SOA
  - have created more complex application landscapes
  - what Gerben Wierda calls “*loosely-coupled spaghetti*”

- EA principles tend to favour
  - flexibility over simplicity and performance
- And tend to gloss over the clash between
  - loose coupling and integrity

## The need for proper Solution Architects

- An enterprise needs Solution Architects to work between Enterprise Architects and Software Architects

- Solution architects must
  - Analyse critical use cases looking at
    - Performance – response/cycle time and throughput
    - Availability
    - Recoverability
    - Integrity
    - Security
    - Scalability
    - Maintainability
    - Etc.
  - Design up-front to meet NFRS
  - Present for approval to authorities

## Some principles for solution architecture

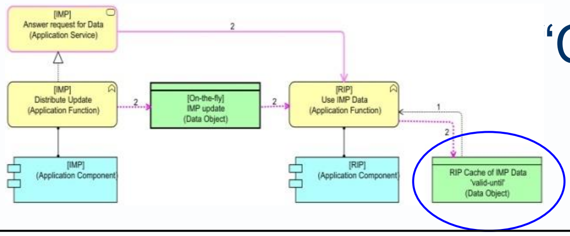
Avancier

1. KISS
2. There is no single way to design systems
3. There are always trade offs
4. Vendors, fashion and principles may push one design pattern, but an architect must be mindful of the opposite
5. Don’t overcomplicate the design to meet fanciful throughput and concurrency numbers that stakeholders agree are very unlikely in the next 5 years (YAGNI and the money will follow)
6. Beware the design of data processing has affects business processes in the wider human activity system (e.g. data store separation impacts integrity)
7. An architect must understand design patterns, trade offs and their impacts on the wider business system.

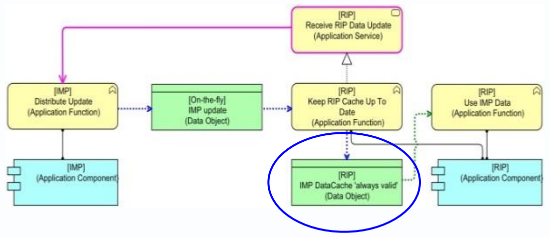
## APPENDIX: Gerben’s ArchiMate diagrams of the same options

Avancier

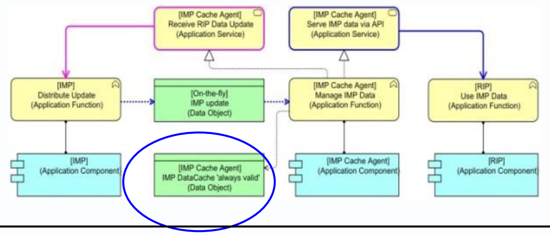
- The content of the earlier UML-style sequence diagrams is copied from the ArchiMate diagrams on the following slides
- Including all application, function and service names



‘Classic client-maintained cache”



“Server-updated cache”

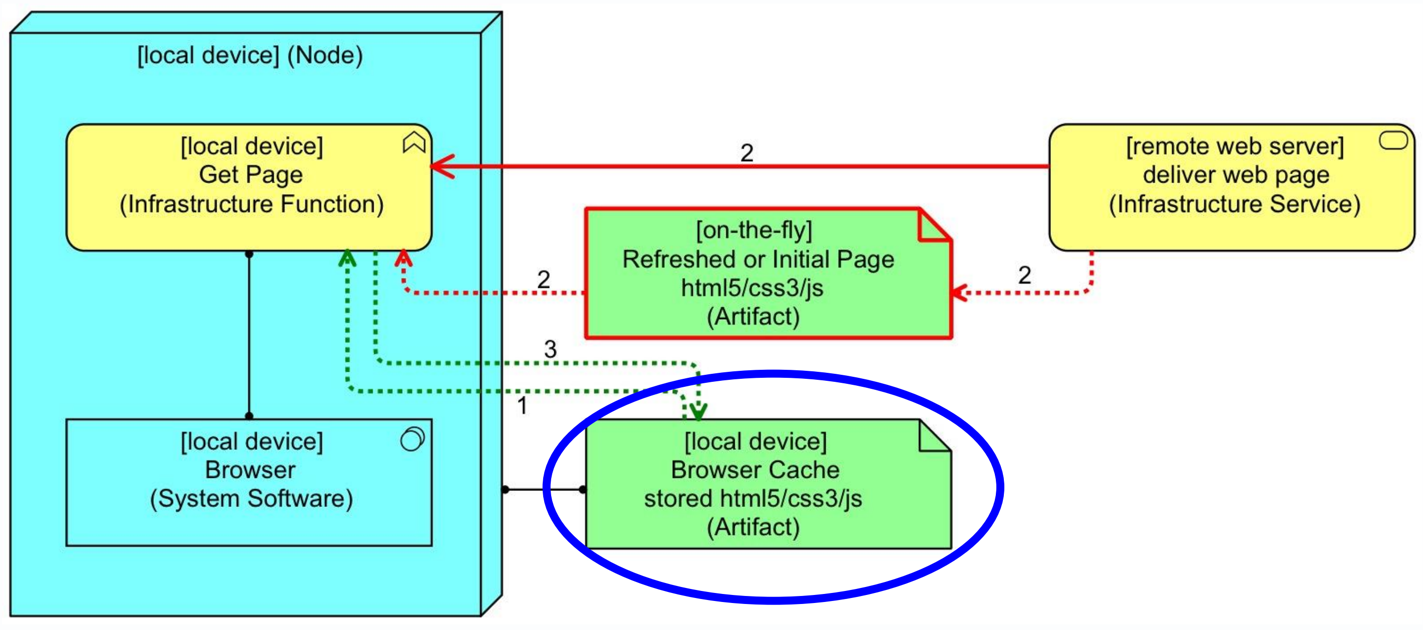


“Server-owned cache”

## Classic client-maintained cache

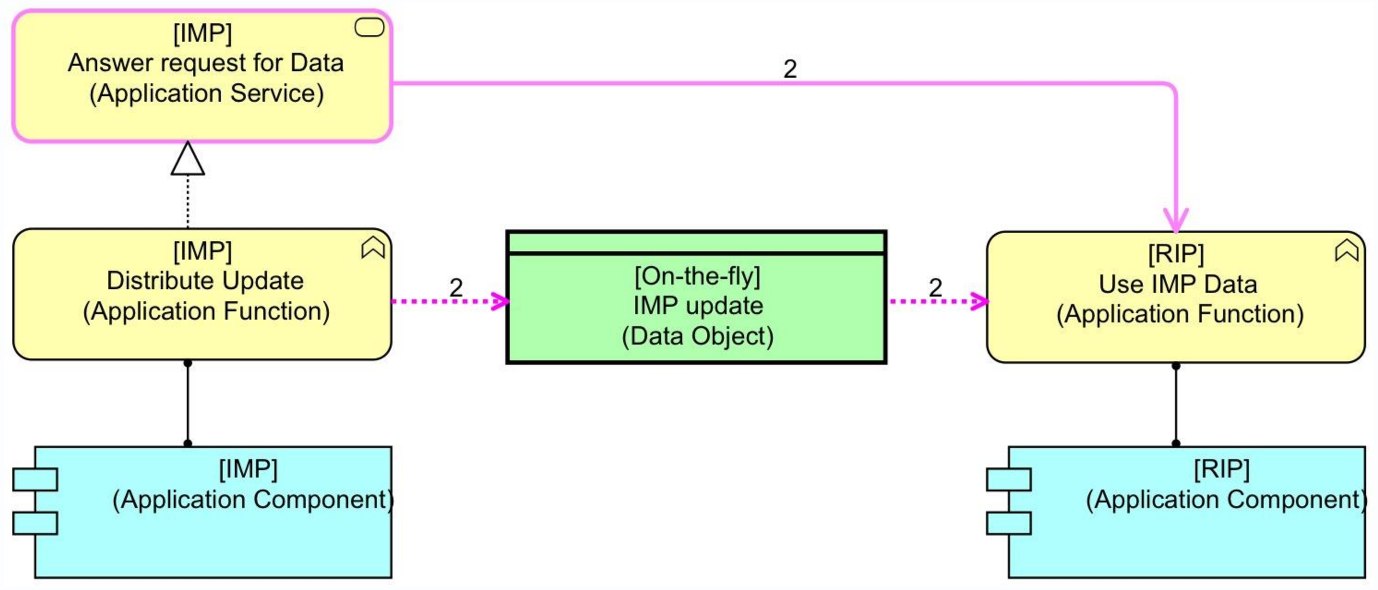
Avancier

1) Browser looks first in cache
2) Page retrieved
3) Page cached



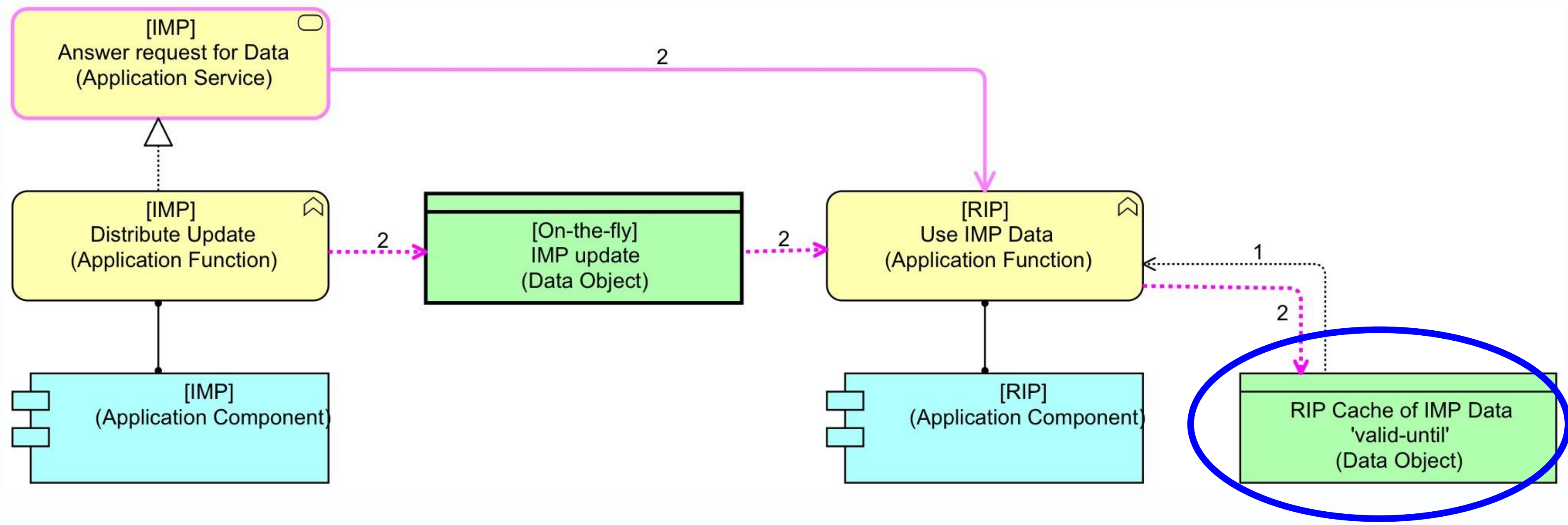
## Original design (no cache)

- Receive-Insurance-Premium system (RIP) requests Policy data from
- Insurance-Member-Policy system (IMP)



## Classic client-maintained cache solution

- RIP first looks in the cache for data  that is not marked out-of-date.
- If data needs to be refreshed, RIP uses the service from IMP and receives the data
- The problem is that we might be using data that is out of date.

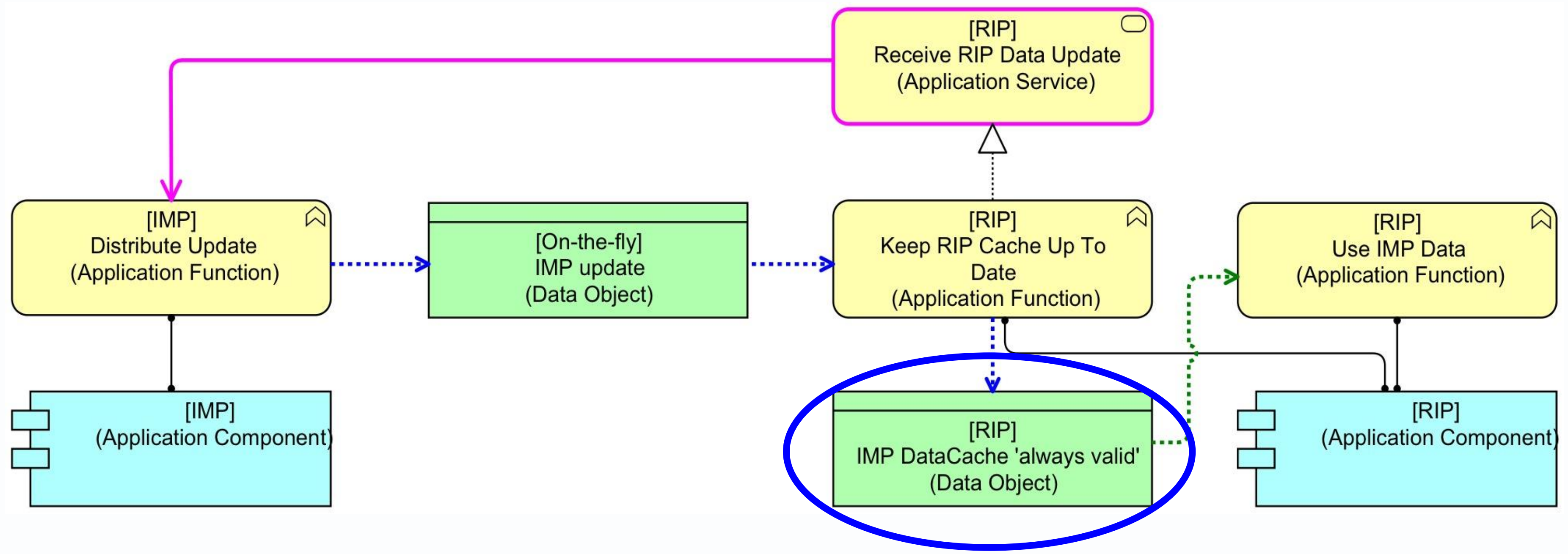


## Server-updated cache

RIP uses only the local cache which is kept up to date by IMP

Every time IMP updates data, it uses a service from RIP and sends it the updated data.

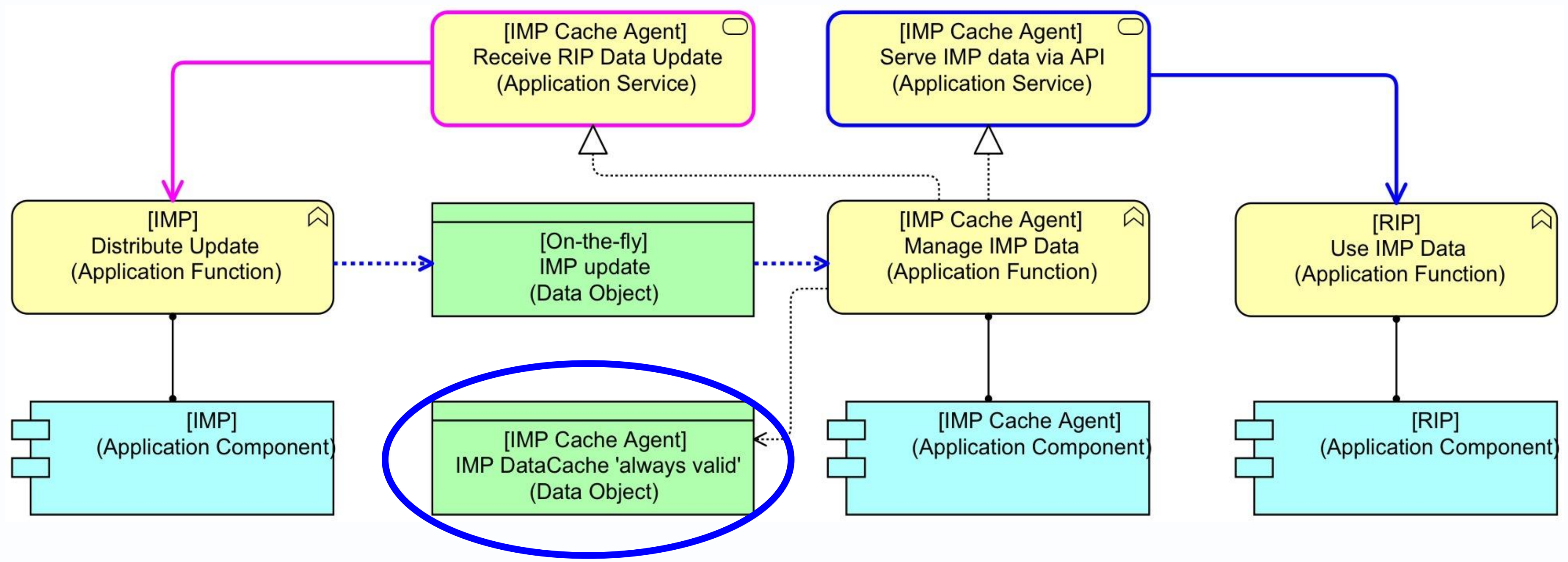
RIP immediately stores it in the cache.



## Server-owned cache

Avancier

- 3rd application manages the cache made available to RIP via API
- It runs on the environment where the cache must be maintained


[IMP Cache Agent] Receive RIP Data Update (Application Service)
[IMP Cache Agent] Serve IMP data via API (Application Service)
[IMP] Distribute Update (Application Function)
[On-the-fly] IMP update (Data Object)
[IMP Cache Agent] Manage IMP Data (Application Function)
[RIP] Use IMP Data (Application Function)
[IMP] (Application Component)
[IMP Cache Agent] IMP DataCache 'always valid' (Data Object)
[IMP Cache Agent] (Application Component)
[RIP] (Application Component)