Using caching to speed up a batch process

Illustrated using ArchiMate and UML

After Gerben Wierda's paper

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

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

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.

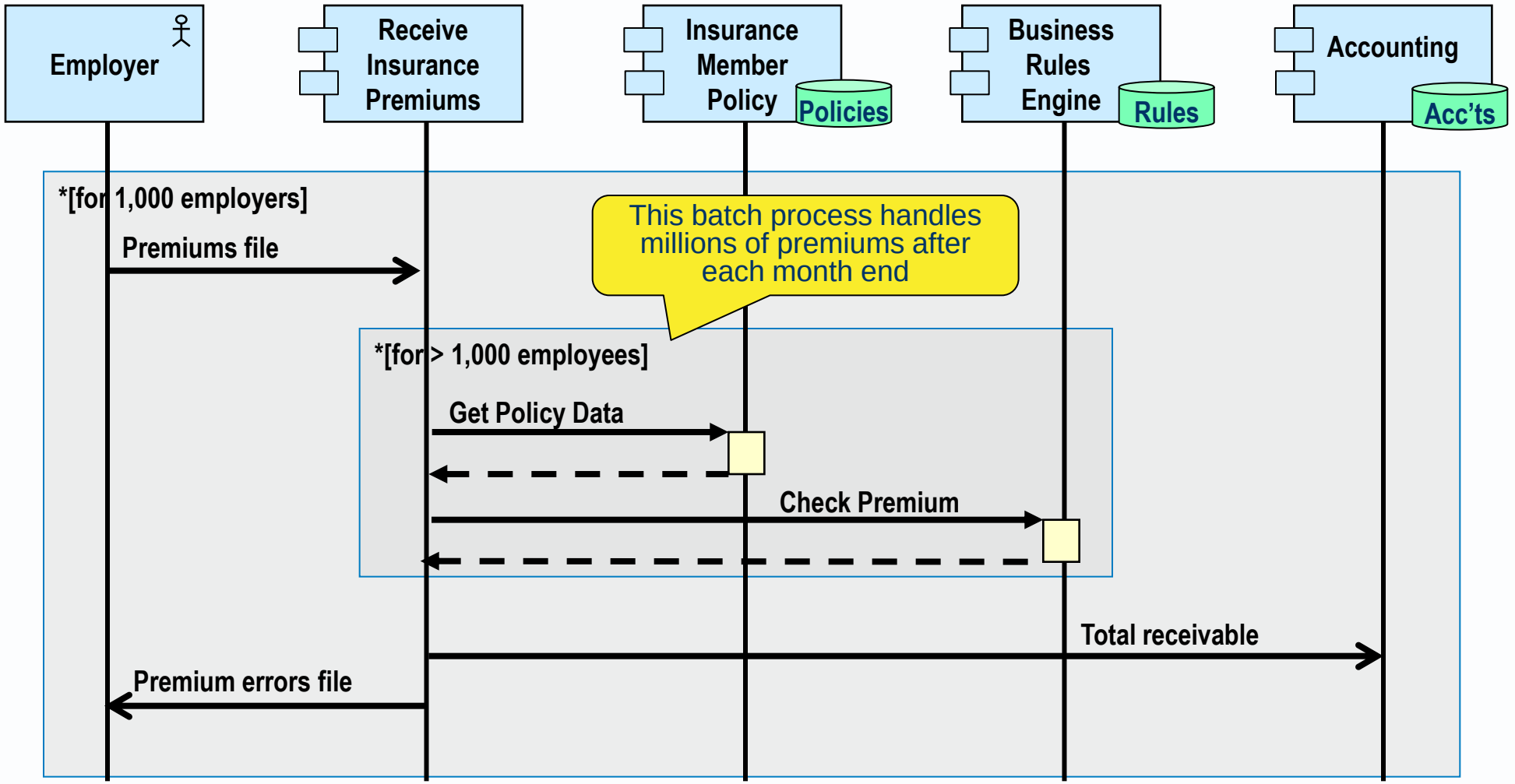
- Such principles are typically designed for
  - On-line transaction use cases
  - Flexibility to handle many customer and/or product variations

Alas

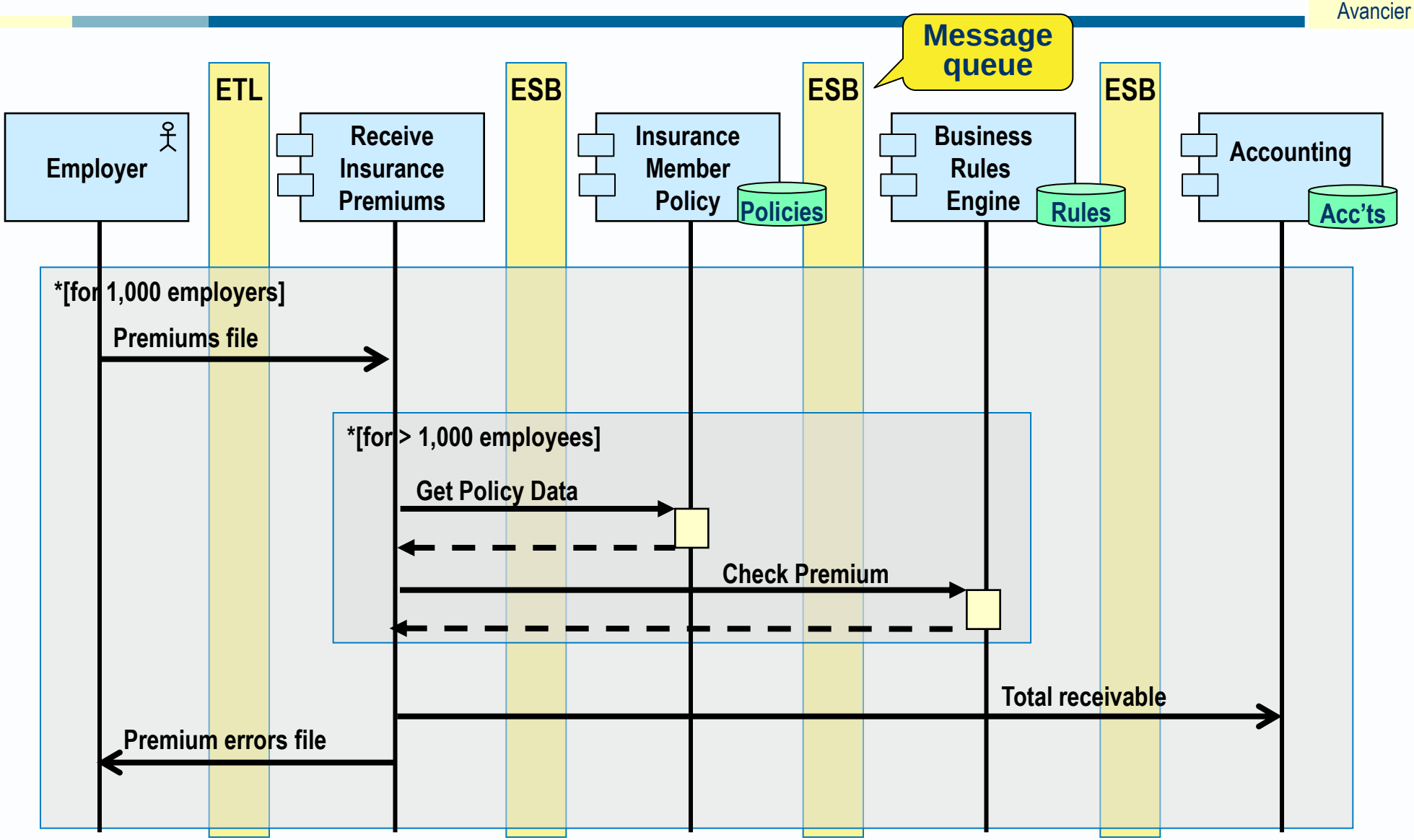
- “Alas, such models are no silver bullets.
- Projects get into serious trouble because of them.
- Three main reasons:
  - brittleness of loosely-coupled spaghetti
  - maintainability
  - performance.”

Gerben Wierda

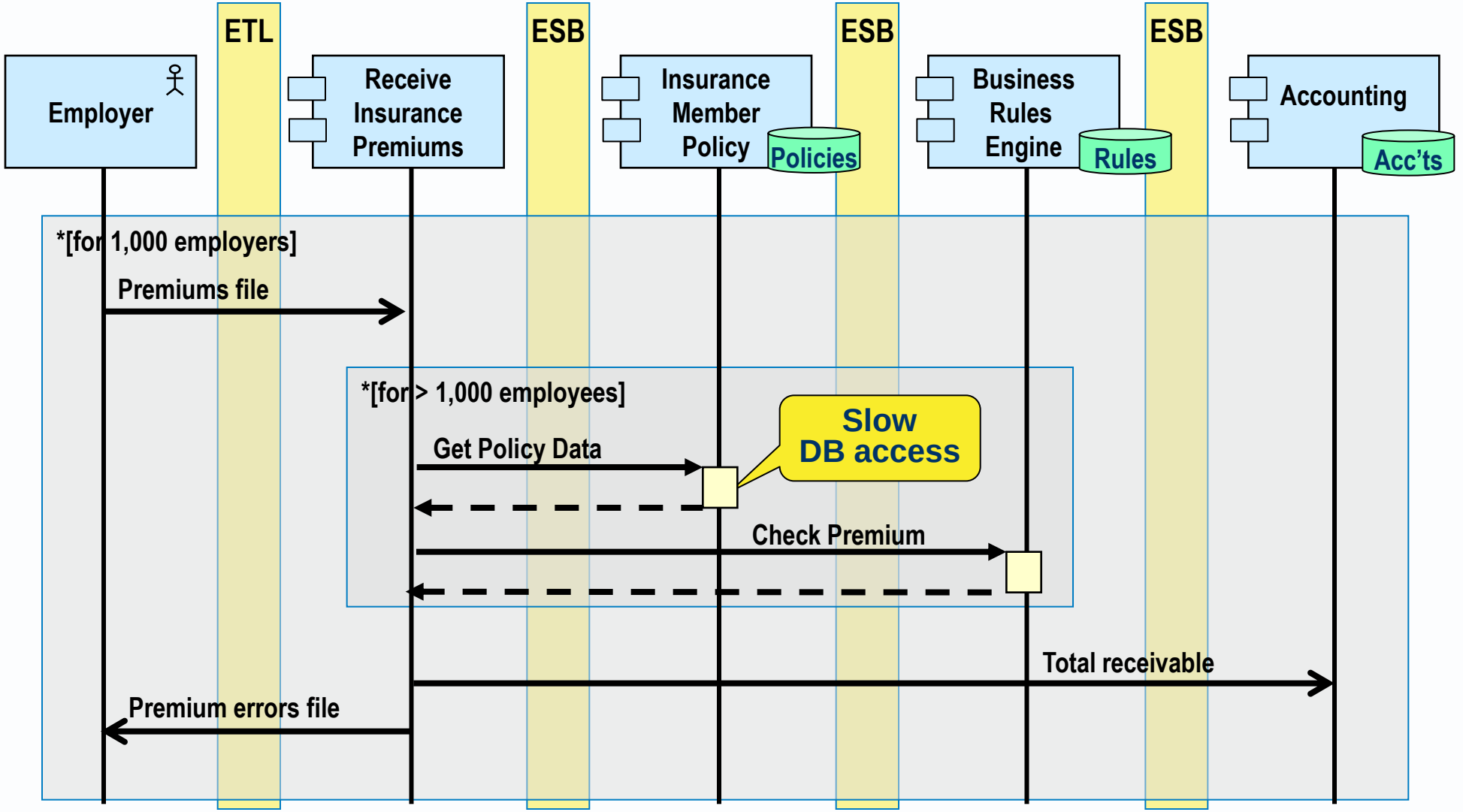
The scenario as a UML sequence diagram



First overhead – network + middleware



Second overhead – database access



How to speed up through design?

- First, identify the bottlenecks in the process

1. Middleware adds network traffic
2. Middleware itself
3. IMP database structure not designed for fast access

How to speed up through design? Cache!

1. Middleware adds network traffic
   - Remove network hops
   - Cache data nearer point of use

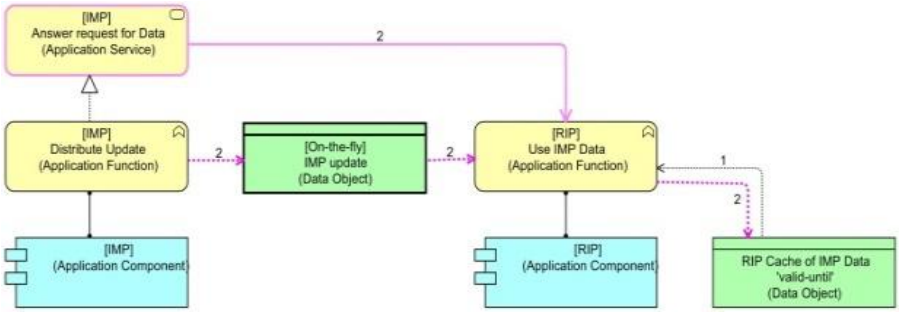
2. Middleware itself
   - Bypass the middleware
   - Cache data nearer point of use

3. IMP database structure not designed for fast access
   - Optimise the data structure for client access
   - Cache data from discs to memory or SSD

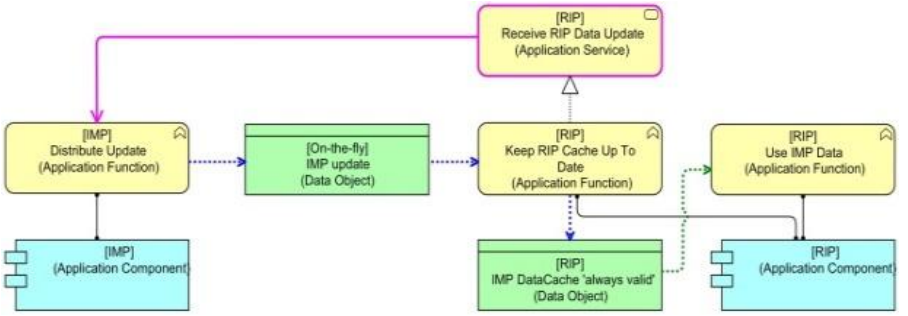
Gerben's ArchiMate diagrams of cache design options

Avancier

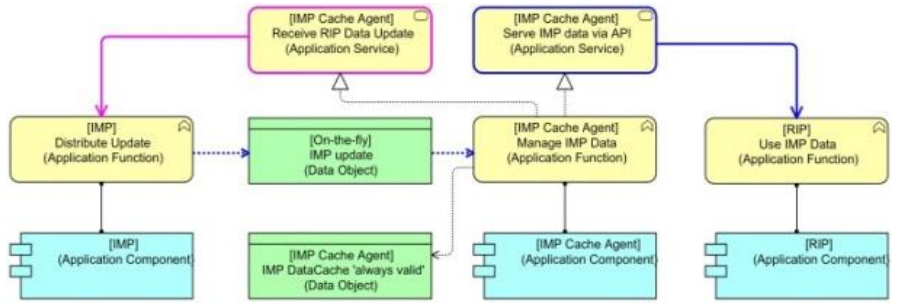
1: Client-maintained cache



2: Server-updated cache



3: Server-owned cache

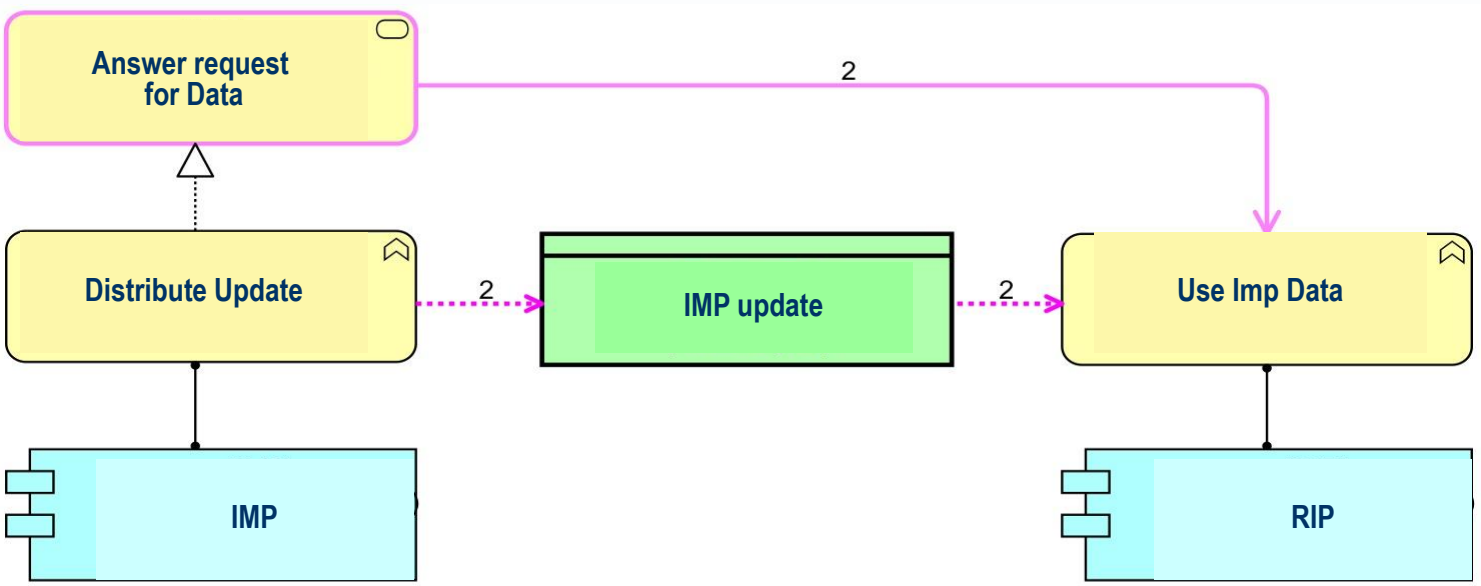


Hereafter

- All bar one service (refinement) will be copied from ArchiMate diagrams onto UML diagrams

0: Original design (no cache)

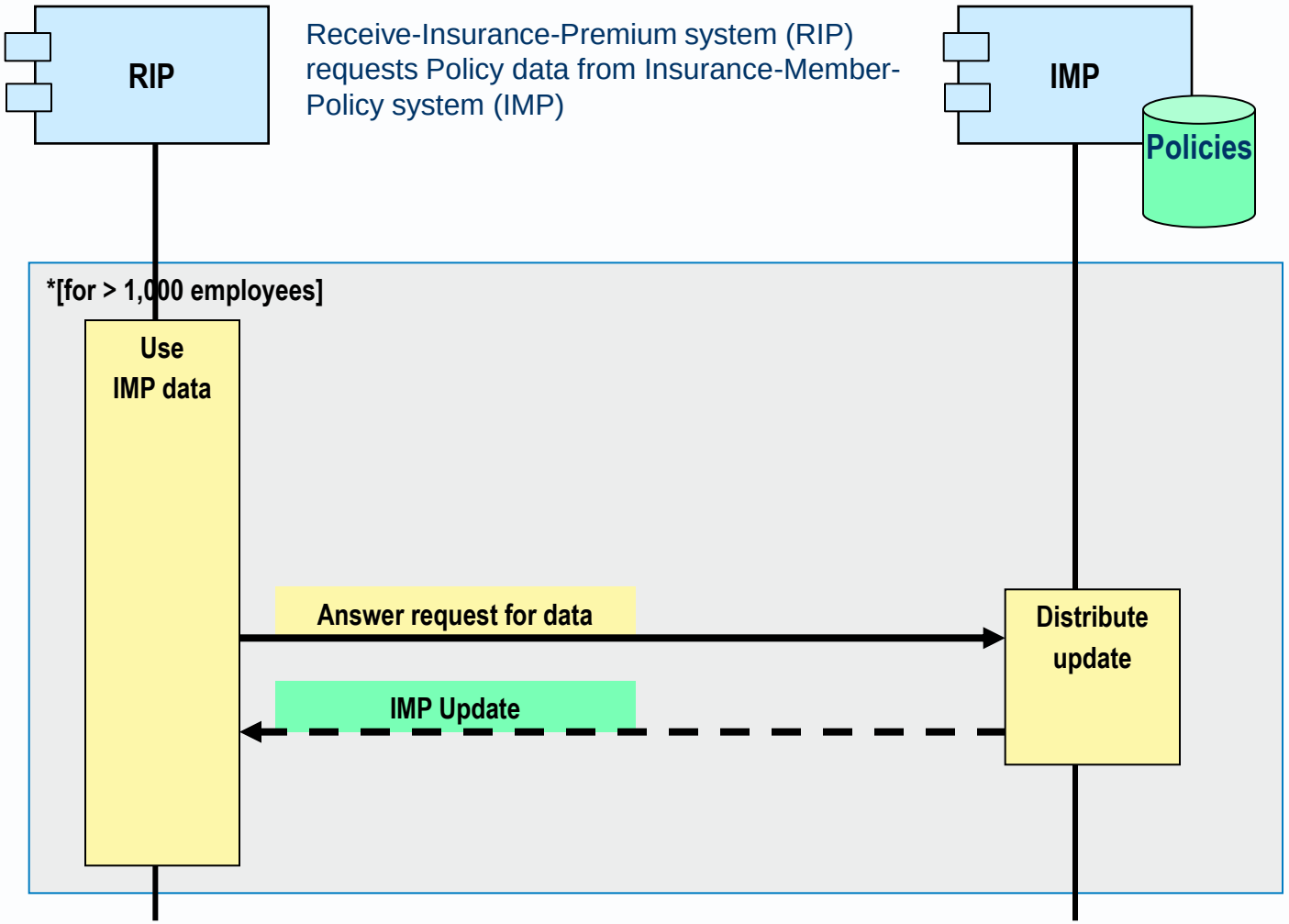
- Receive-Insurance-Premium system (RIP) requests Policy data from
- Insurance-Member-Policy system (IMP)



GW shows show data in flow as a **data object** passing from a **function** to a function
From the invoked service would be better.

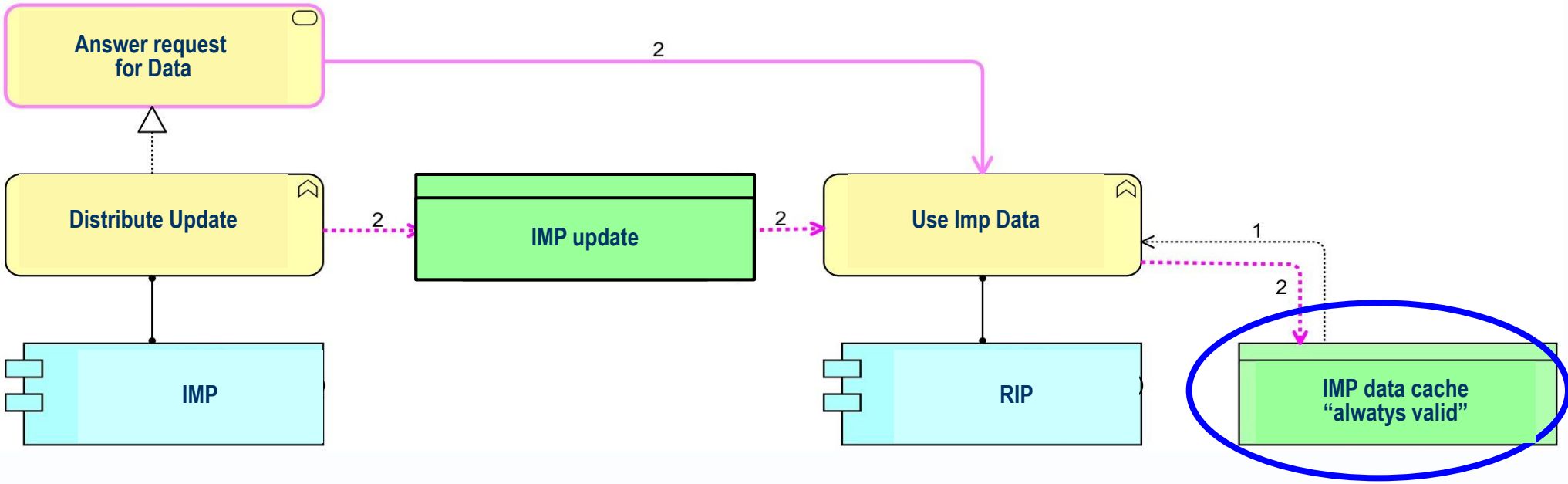
0: Original design (no cache)

(drawn to match ArchiMate diagram)

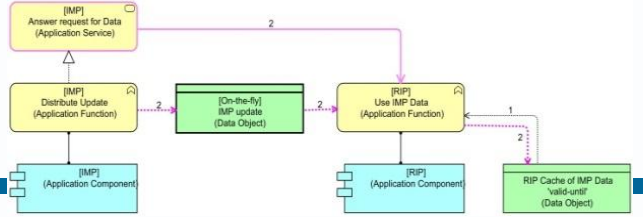


1: Classic client-maintained cache

- 1) RIP looks in the cache for data that is not marked out-of-date.
- 2) 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.



1: Classic client-maintained cache

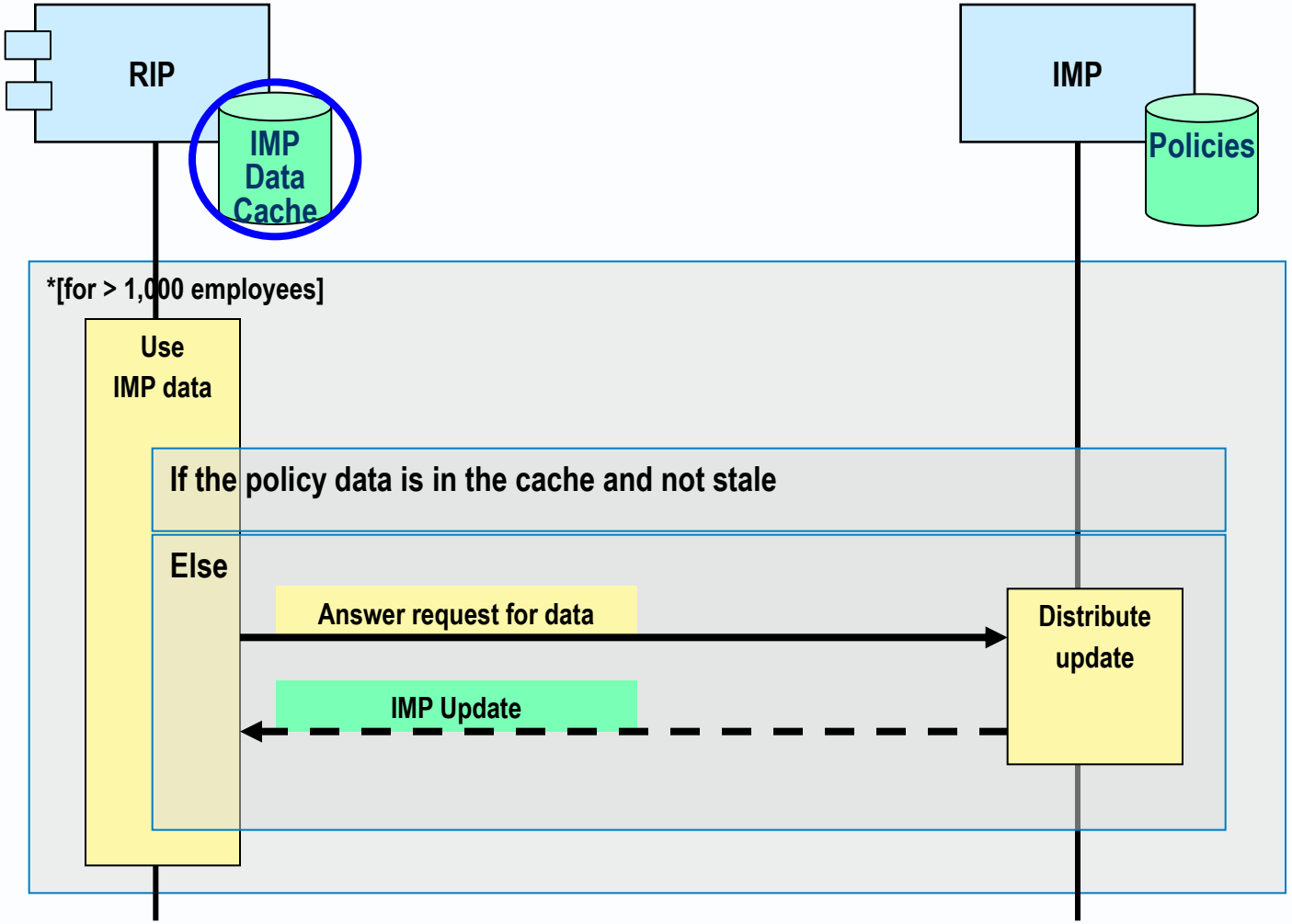


Avancier

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

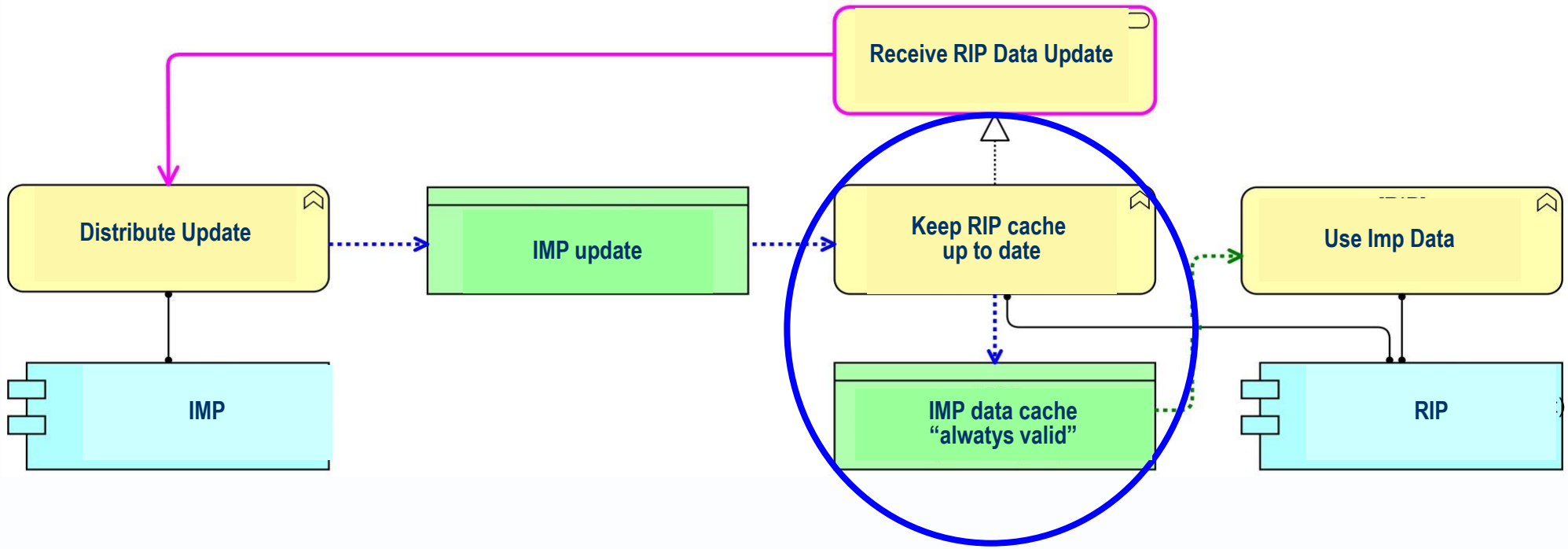
If the data needs to be refreshed, RIP uses the service from IMP and receives the data

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

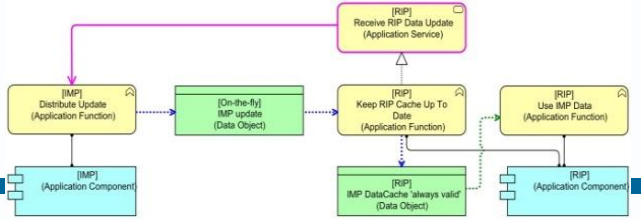


2: Server-updated cache

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.



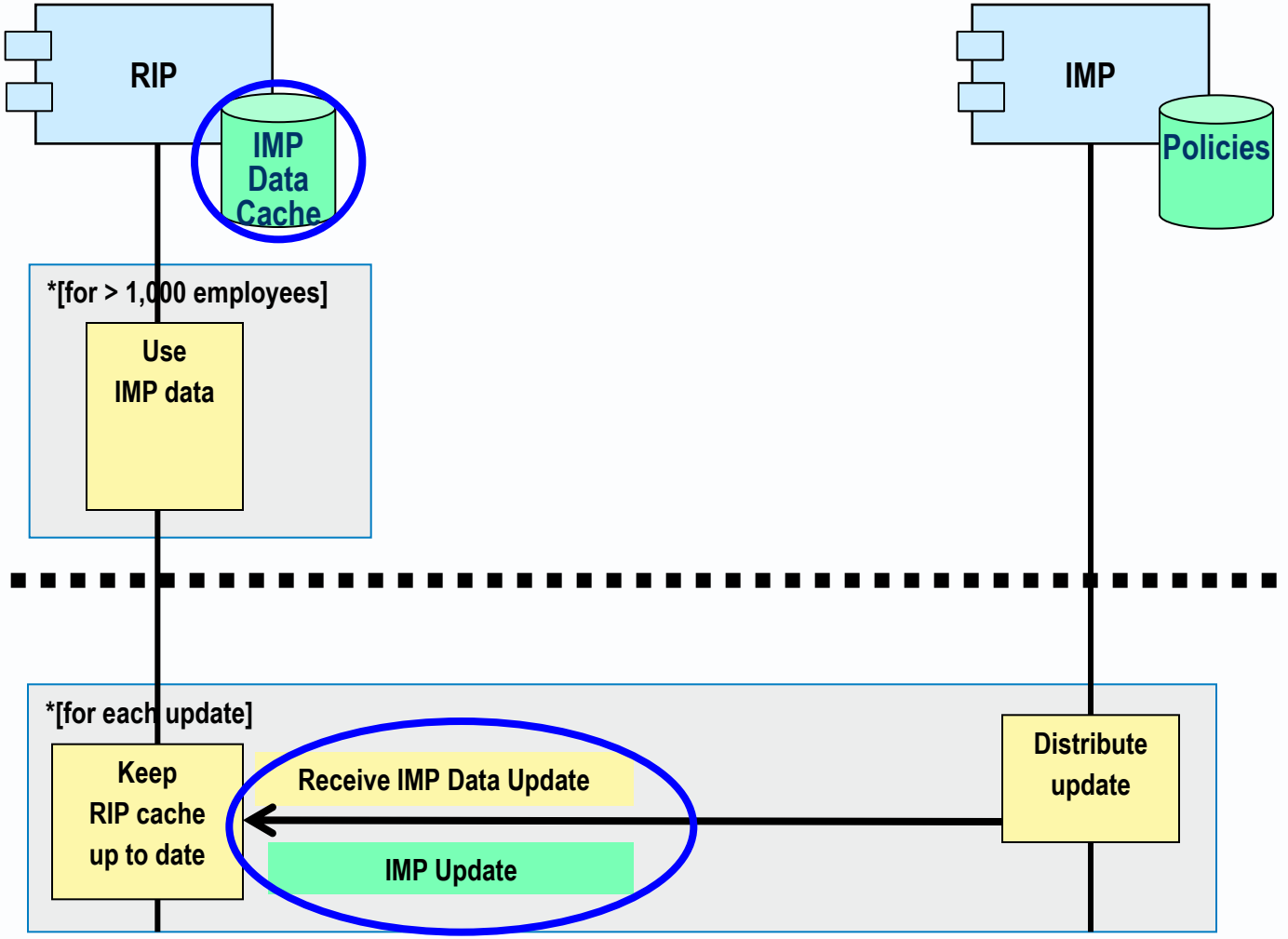
2: Server-updated cache



Avancier

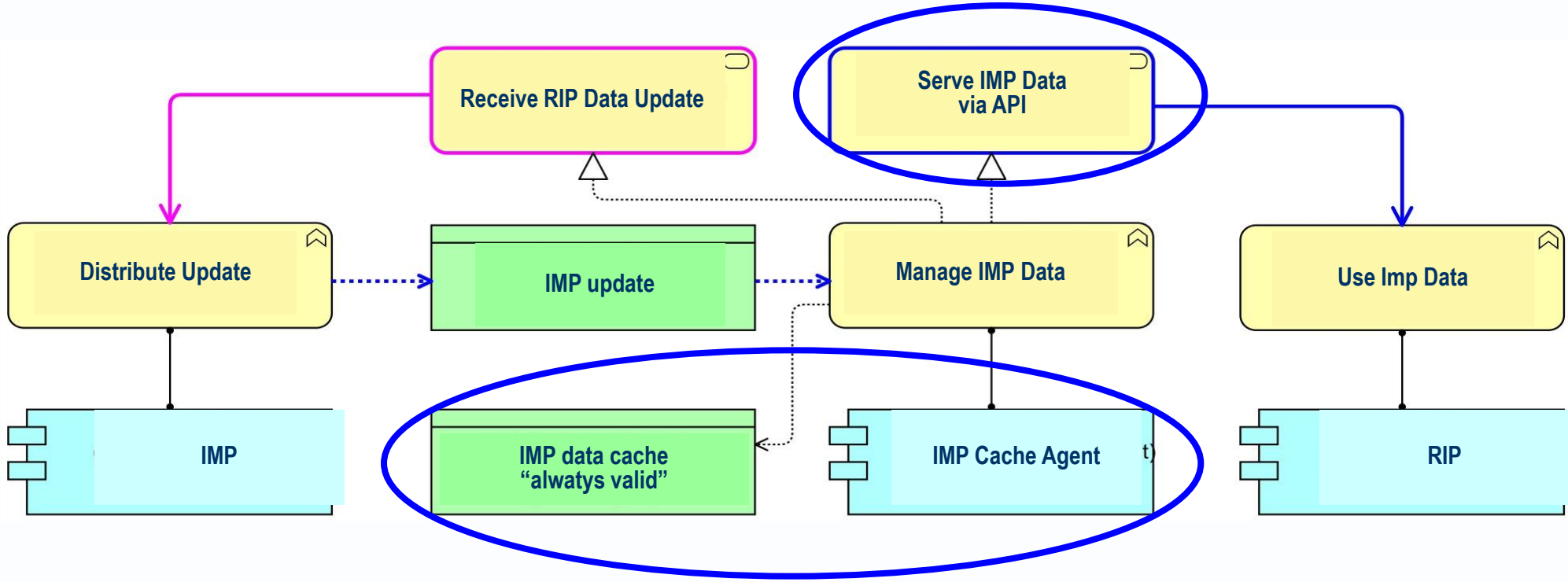
RIP uses only the local cache which is kept up to date by IMP.

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

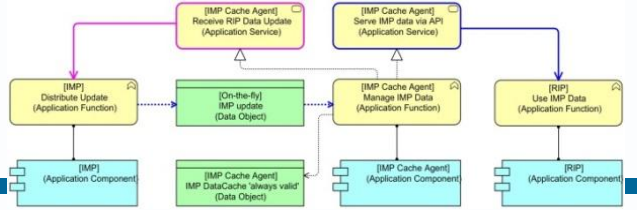


3: Server-owned cache

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



3: Server-owned cache v1



Avancier

An agent sits on every node where data is needed fast

RIP

IMP Cache Agent

IMP Data Cache

IMP

Policies

- RIP reads the cache via the API

*[for > 1,000 employees]

Use IMP data

Serve IMP data

IMP Update

Manage IMP data

- Each agent subscribes to receive

*[for each update]

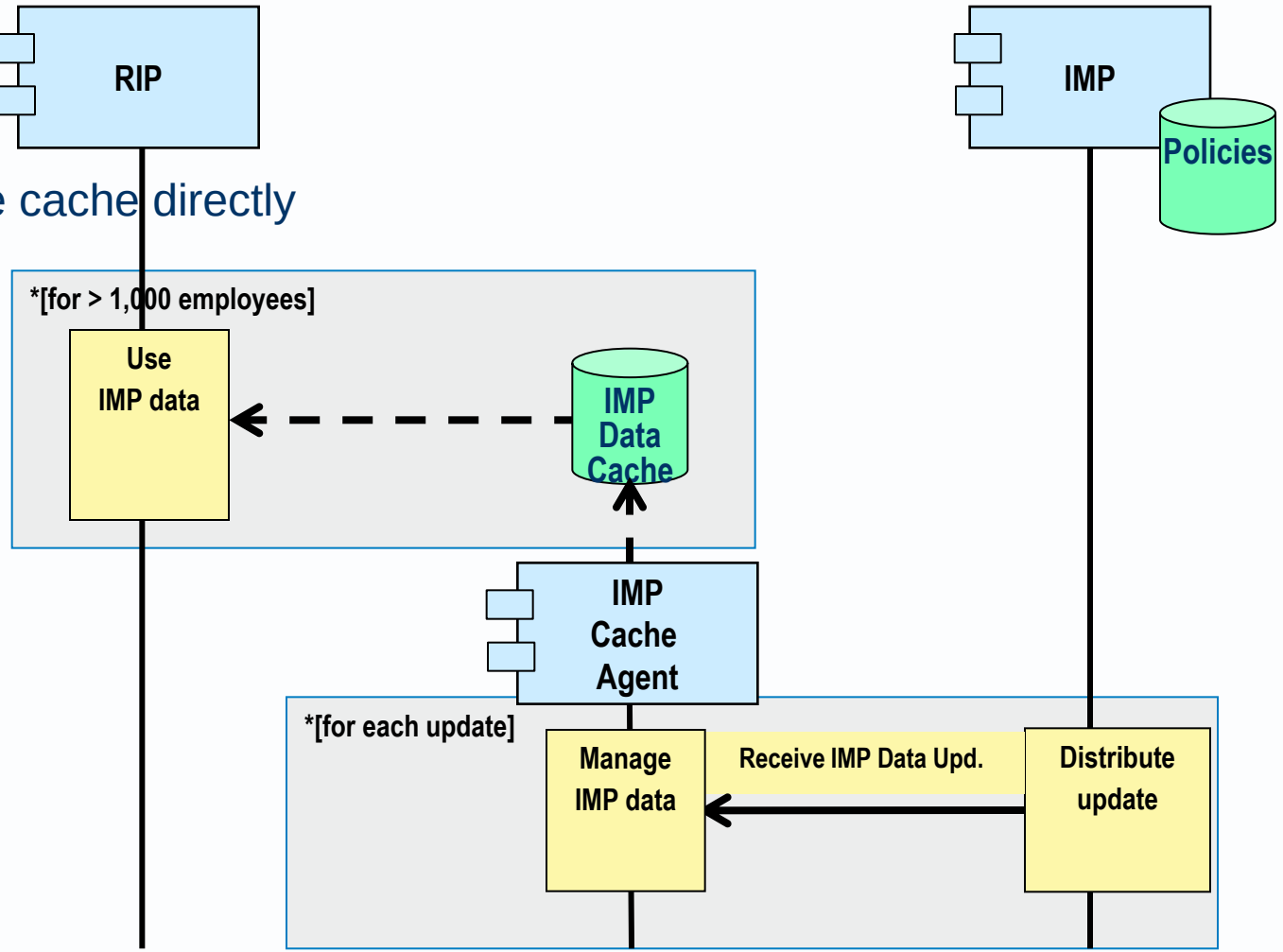
Manage IMP data

Receive IMP Data Upd.

Distribute update

3: Server-owned cache v2

- RIP reads the cache directly



4: Shared data space

- Not likely to work in this case, but a general pattern