Guiding Principals

- Only Store the core people data
- Honor the authoritative sources
- Incremental integration

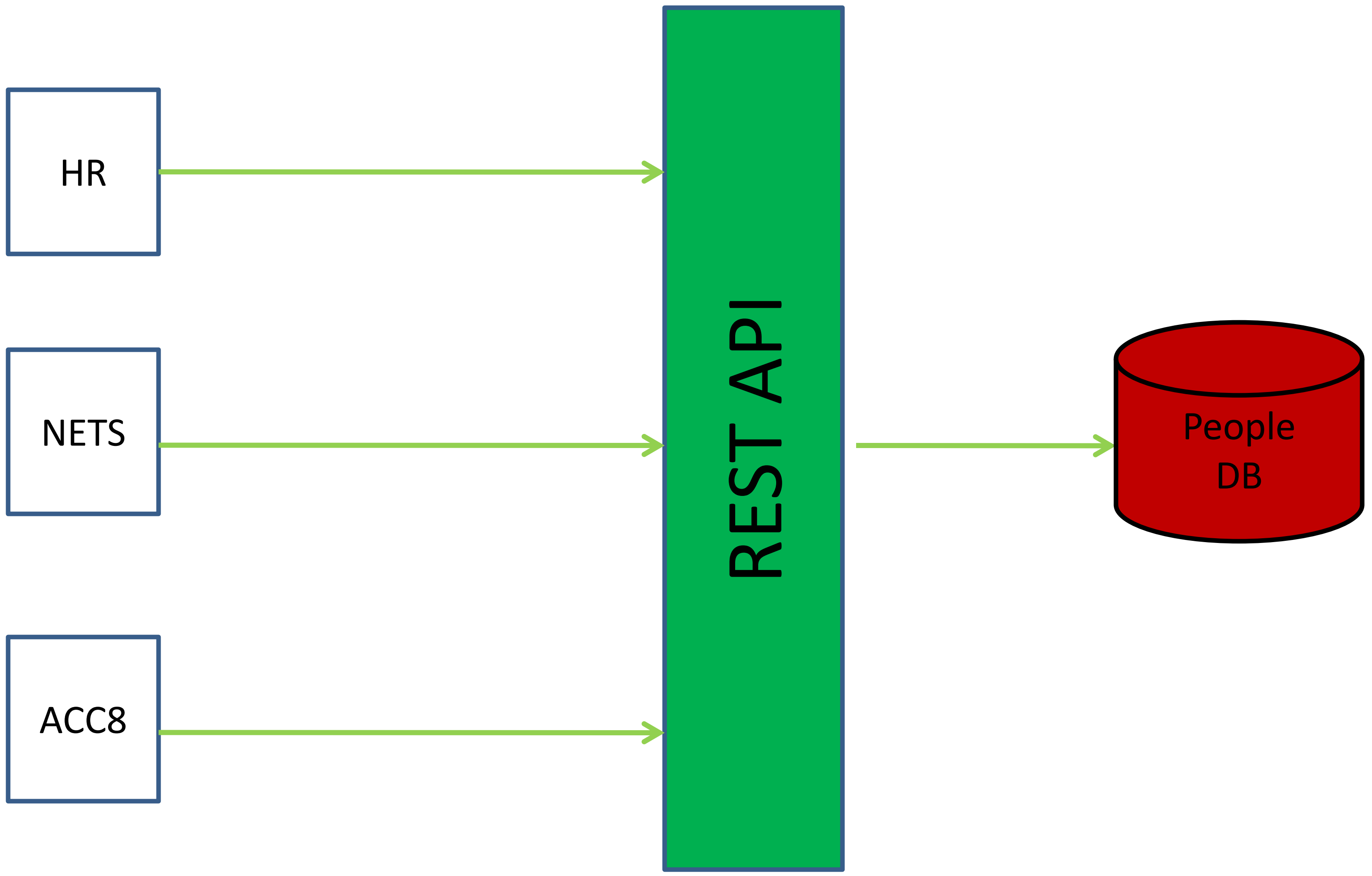
What People DB stores

- Internal Persons (staff and visitors)
  - Names, username, status
  - Positions
  - Office allocations (building, room, phone number)
- Collaborators
  - Names, username, status
  - External organization (name, address, etc)
- UCAR Organizations
- Groups
- Usernames (login, mail alias, mailman list, mailbox)

Authoritative data sources

- HR
  - Internal persons: names, position, status
  - Organization structure
- NETS
  - Internal persons: office allocations, preferred names
- CISL Allocations Database (ACC8)
  - Supercomputing and TeraGrid users
- People DB
  - upid, username (login, email alias, mailman list, mailbox)

Data sources push data to People DB



People/Group Editor

- Allow public to search UCAR employees, visitors and collaborators
- Allow user to modify their own data (e.g. home page)
- Allow system admin to add or modify people data
- Allow system admin to edit groups, email alias, mailbox, mailman lists
- GWT-based, will replace People Search app

People/Group Editor demo

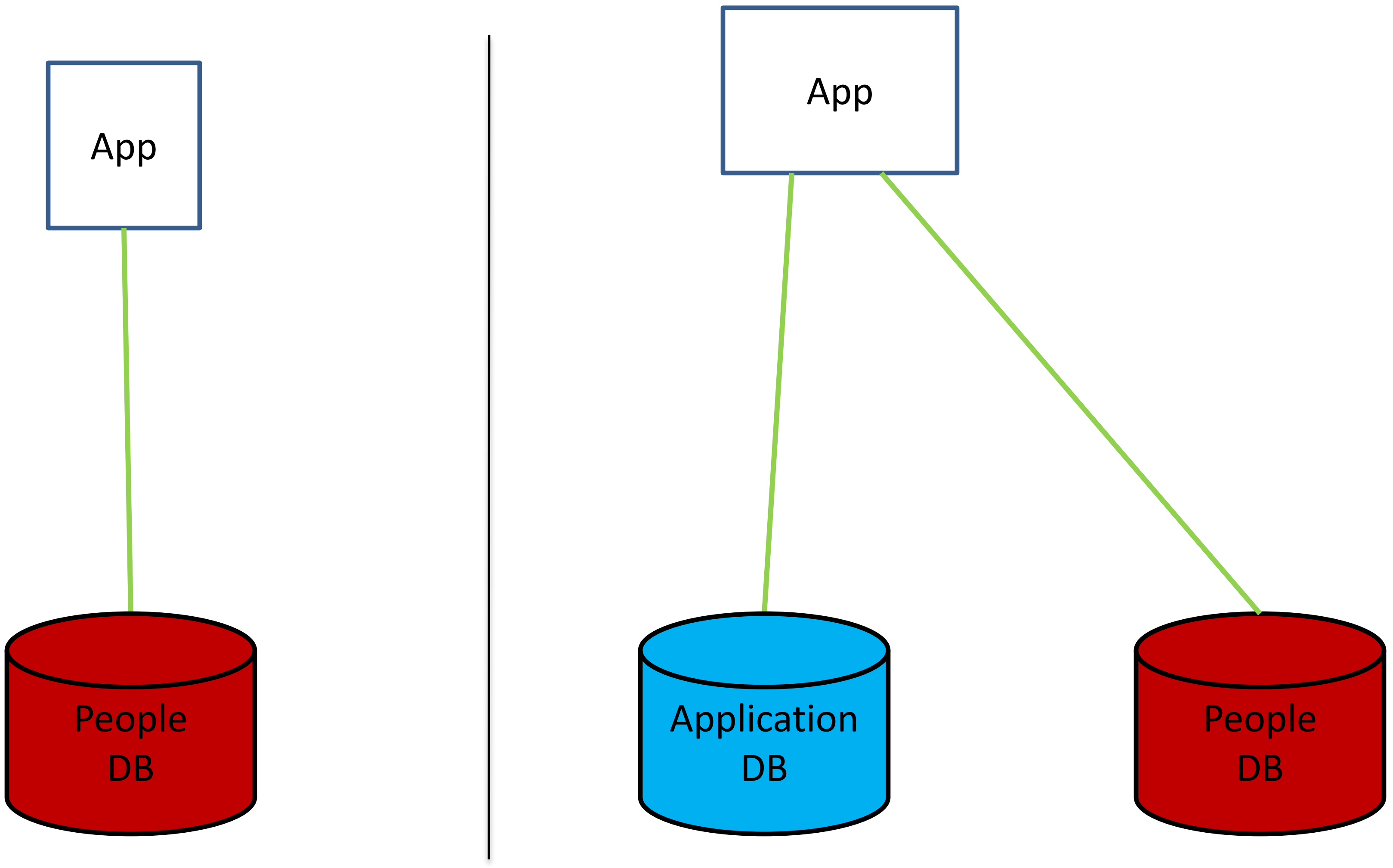
How developers can use people DB Data

- Data consumer
- Synchronize people DB data to your local DB
- Data contributer

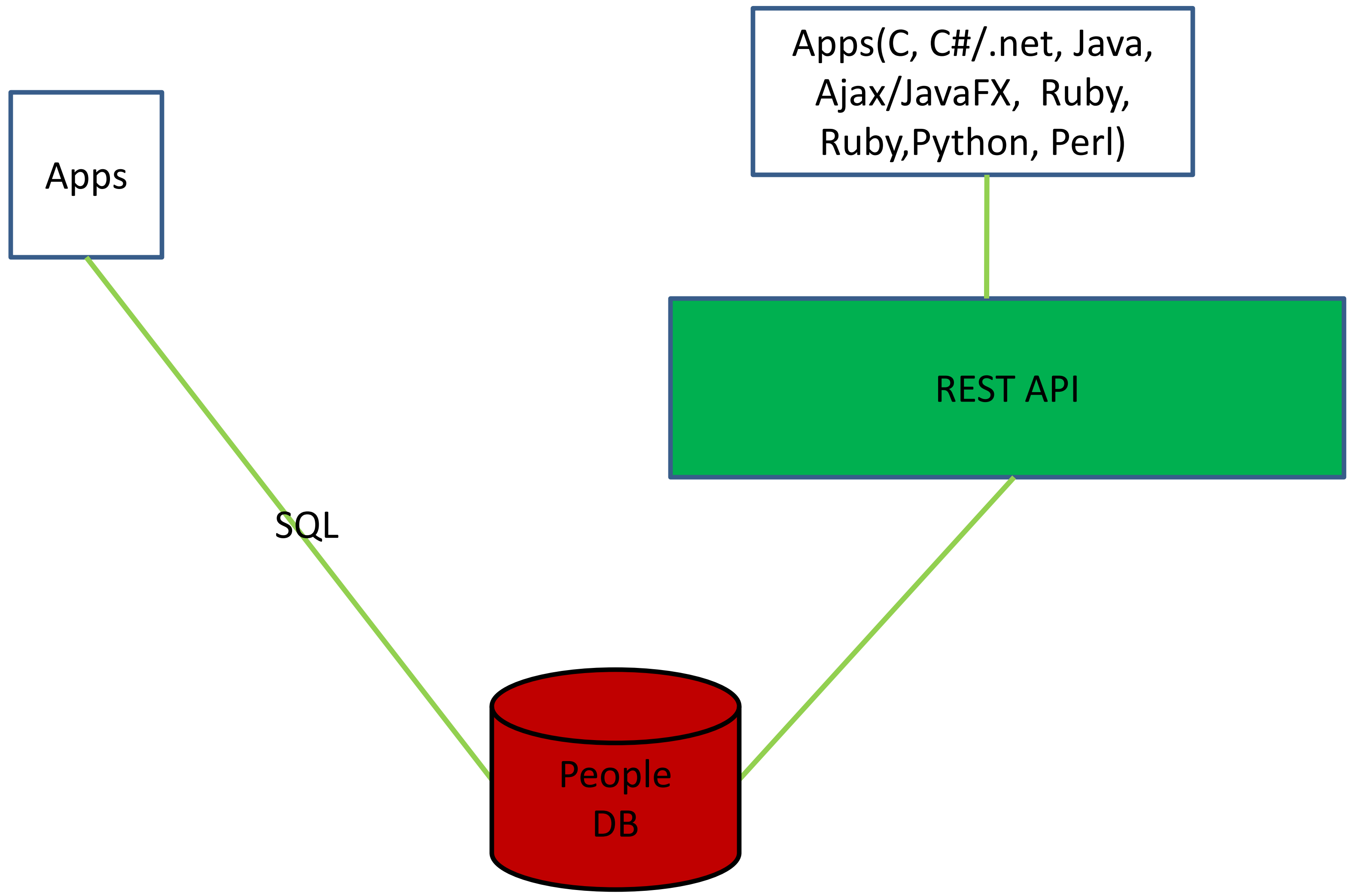
A few examples

- Use organization structure stored in people DB to create cascade drop downs
- Add people search to your site using REST API or create a local directory for your organization
- Use people data stored in People DB for their own applications or 3rd party applications
- Authorization of users for applications, content, and functionality.

Two Scenarios To Use People DB



First Scenario


Apps
Apps(C, C#/.net, Java, Ajax/JavaFX, Ruby, Ruby,Python, Perl)
REST API
SQL
People DB

It is recommended to use REST API because it is very easy and is supported by almost all languages and it is less likely you have to change if database schema changes

SQL vs. REST search internal persons

```
SELECT P.name_last, P.name_first,
P.email, P.upid,
       P.name_middle,
       P.name_suffix,
       P.nickname
FROM person P
WHERE to_lower(P.name_first) like
'%mark%' or to_lower(P.name_last)
like '%mark%' or
to_lower(P.name_middle) like
'%mark%' or
to_lower(P.name_first) like
'%mark%';
```

```
https://api.ucar.edu/people/inte
rnalPersons?name="mark"
```

SQL vs. REST
get staff detail

```
SELECT P.name_last, P.name_first,
P.email, P.upid,
       C.position_current_id,
       C.current_employee,
       C.start_date,
       C.end_date,
       C.primary_position,
       T.position_title_id,
       T.title
FROM person P,
       position_current C,
       position_title T
WHERE P.upid = C.upid
AND C.position_title_id =

       T.position_title_id
AND P.upid = 15446;
```

```
https://api.ucar.edu/people/inte
rnalPersons/296
```

SQL vs. REST
get organizational hierarchy

```
SELECT o1.acronym, o1.full_name,
       ol.code
FROM organization o1, organization
       o2, organization_level ol
WHERE o1.preorder_tree_left <
       o2.preorder_tree_left
AND o1.preorder_tree_right >
       o2.preorder_tree_right
AND o2.acronym='MMM'
AND o1.org_level_id=ol.org_level_id
ORDER by ol.code;
```

```
https://api.ucar.edu/people/orgH
ierarchy?org=MMM
```

REST Client Code - Java

```
public String invokeRESTGet(String url)  {
    try {
        URL u = new URL(url);
        HttpsURLConnection uc = (HttpsURLConnection) u.openConnection();
        uc.setRequestMethod("GET");
        uc.setRequestProperty("Content-Type", "application/json");
        uc.setDoOutput(false);

        int status = uc.getResponseCode();
        if (status != 200)  {
            //handle HTTP errors
        }
        InputStream in = uc.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(in));
        String buffer = br.readLine();
        return buffer;
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

REST Client Code - PHP

```
<?php

$response =
http_get("https://api.ucar.edu/people/internalPersons?name=ma
rk" , $info);
$body = http_parse_message($response)->body;
$persons = json_decode($body);
foreach ($persons as $person) {
        foreach ($person as $key => $value) {
                echo nl2br("$key: $value \n");
        }
        echo nl2br("\n");
}
?>
```

REST Client Code - Python

```
#!/usr/bin/python
import httplib2
import demjson

url='https://api.ucar.edu/people/internalPersons?name=bill'
http=httplib2.Http()

headers = {'Content-type': 'application/json'}
response, content = http.request(url, 'GET', headers=headers)
status = response.status
if status == 200:
		persons=demjson.decode(content)
		for i in range(len(persons)):
				for key, value in persons[i].items():
						print "%s=%s" % (key, value)
				print
if status == 399:
		print content
if status == 500:
		print 'Server error'
```

REST Client Code - Perl

```perl
#!/usr/bin/perl
use LWP::UserAgent;
use JSON;

$url='https://localhost:8443/internalPersons?name=bill';
$ua = LWP::UserAgent->new;
$ua->agent("MyApp/0.1 ");
my $req = HTTP::Request->new(GET => $url);
$req->content_type('application/json');
my $res = $ua->request($req);

if ($res->is_success) {
           #print $res->content;
           $respData=from_json($res->content);
           #use Data::Dumper; die(Dumper($respData));
           foreach my $hash (@{$respData}) {
                      #use Data::Dumper; print Dumper($hash);
                      while( my($key, $value) = each %{$hash} ){
                                 print "$key: $value\n";
                      }
                      print "\n";
           }
}
elsif ($res->code eq "399" )
{
           print $res->content, "\n";
}
else
{
           print $res->status_line, "\n";
}
```

REST Client Code - Ruby

```
#!/usr/bin/ruby
require 'rubygems'
require 'rest-open-uri'
require 'json'

url='https://bluegrass:8443/internalPersons?name=bill'
begin
      persons = JSON.parse(open(url, :method => :get, 'Content-Type' => "application/json“,:ssl_verify => false).read)
      persons.each do
           |p|
           p.each do
              |key, value|
              puts "#{key}: #{value}"
           end
       puts
end

 rescue OpenURI::HTTPError => e
     response=e.io
     response_code = response.status[0].to_i
     puts response_code
     if response_code ==  399
           puts response.read
     elsif response_code == 500
           puts 'Server error‘
     else
            raise e
end
end
```

Search Internal Staff

https://api.ucar.edu/people/internalPersons?name=mark

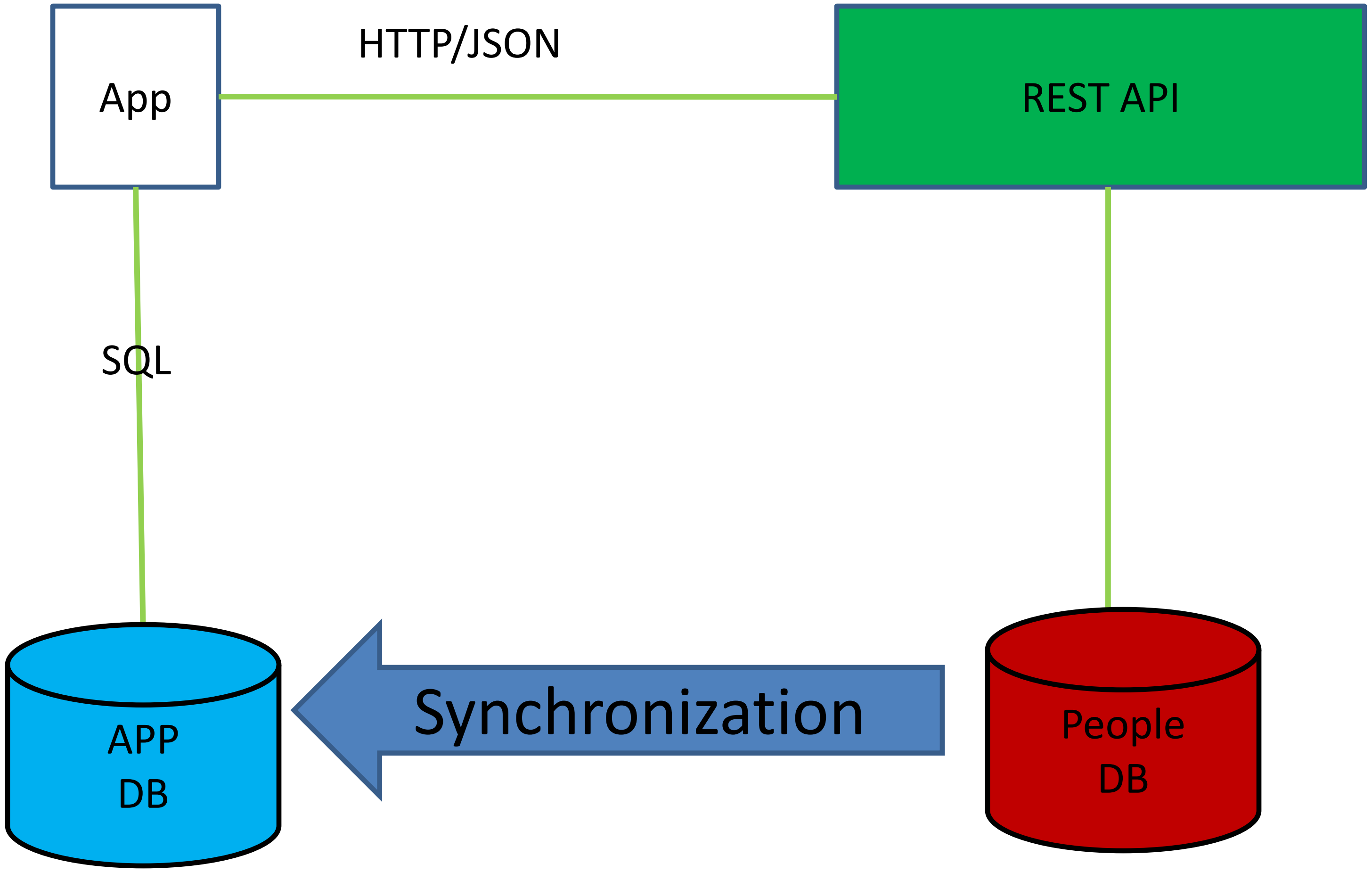
```
[
 - {
      upid: 194,
      firstName: "Mark”,
      lastName: "Lord”,
      middleName: "”,
      nameSuffix: "”,
      nickname: "”,
      username: "”,
      email: "lord@ucar.edu",
   },
 - {
      upid: 296,
      firstName: "Mark”,
      lastName: "Stobbs”,
      middleName: "”,
      nameSuffix: "”,
      nickname: "”,
      username: "”,
      email: "mstobbs@ucar.edu",
   },
 - {
      upid: 906,
      firstName: "Mark”,
      lastName: "Miesch”,
      middleName: "”,
      nameSuffix: "”,
      nickname: "”,
      username: "”,
      email: "miesch@ucar.edu",
   }
]
```

Get Internal Staff Detail

https://api.ucar.edu/people/internalPersons/296

```
{
upid: 296,
firstName: "Mark″,
lastName: "Stobbs″,
middleName: "″,
nameSuffix: "″,
nickname: "″,
username: "″,
email: "mstobbs@ucar.edu",
- positions: [
     - {
          startDate: "2009-10-04″,
          endDate: "″,
          title: "SOFT ENG/PROG III″,
          type: "Employee″,
          organization: "OSD″,
          supervisorUpid: 7344,
          isPrimary: true,
          hostContact: "″,
     }
],
- officeAllocations: [
     - {
          buildingName: "ML″,
          roomNumber: "17L″,
          officePhoneNumber: "303-497-
          1238″,
     }
   ]
}
```

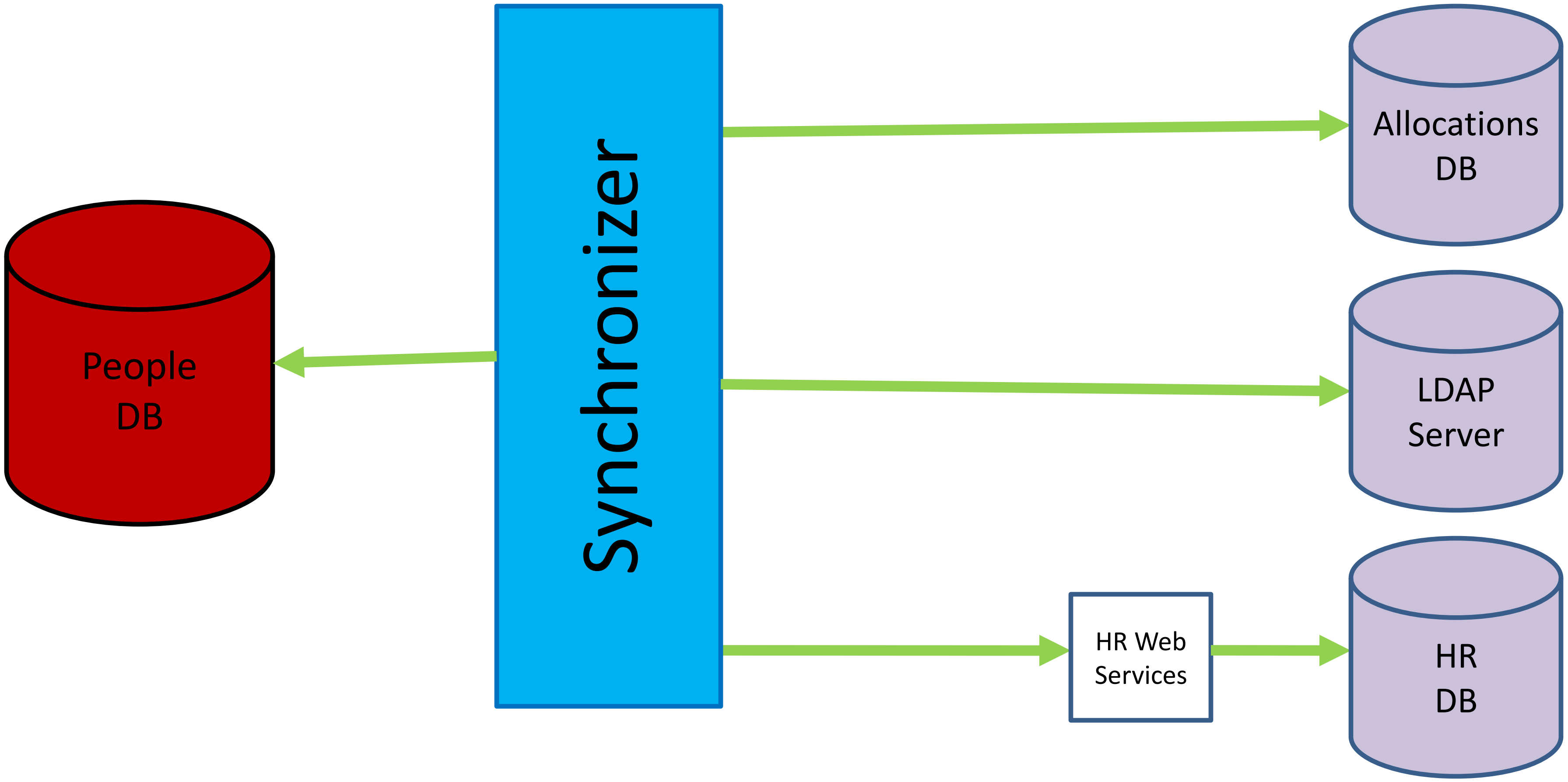
Second Scenario



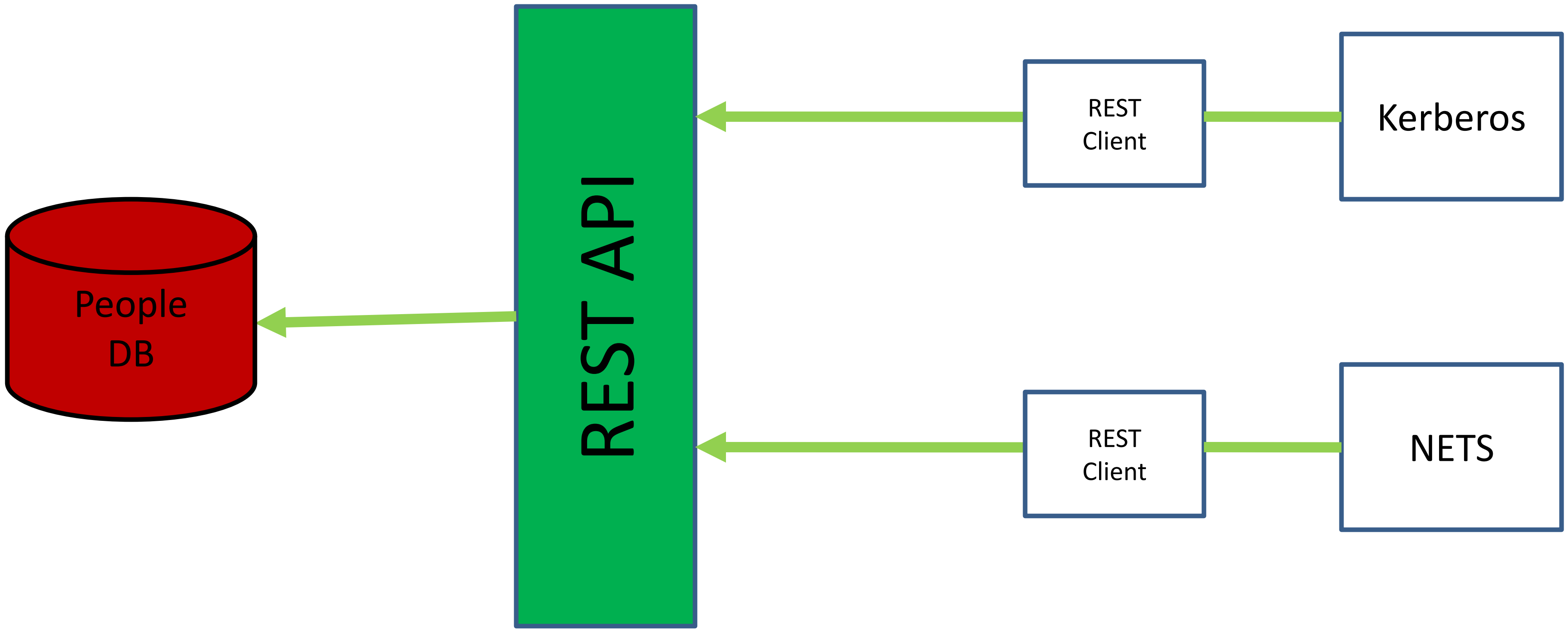
How synchronizer works

- Mapping from people db to your db
- Only synchronize the changes from last time using timestamp
- Real time push or scheduled
- Implement your own custom pull synchronizer

Synchronization Strategies-Push



Synchronization Strategies - pull



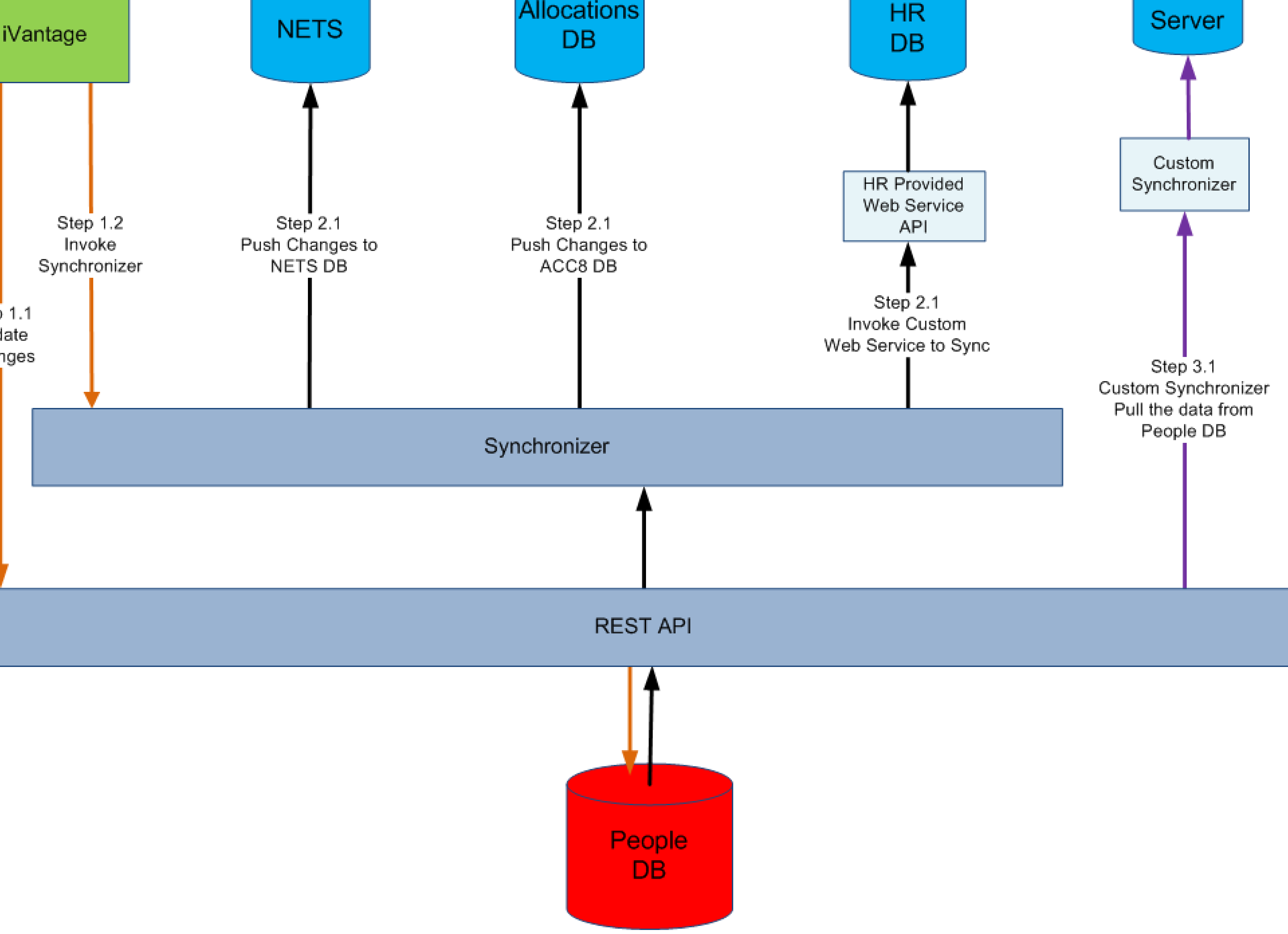
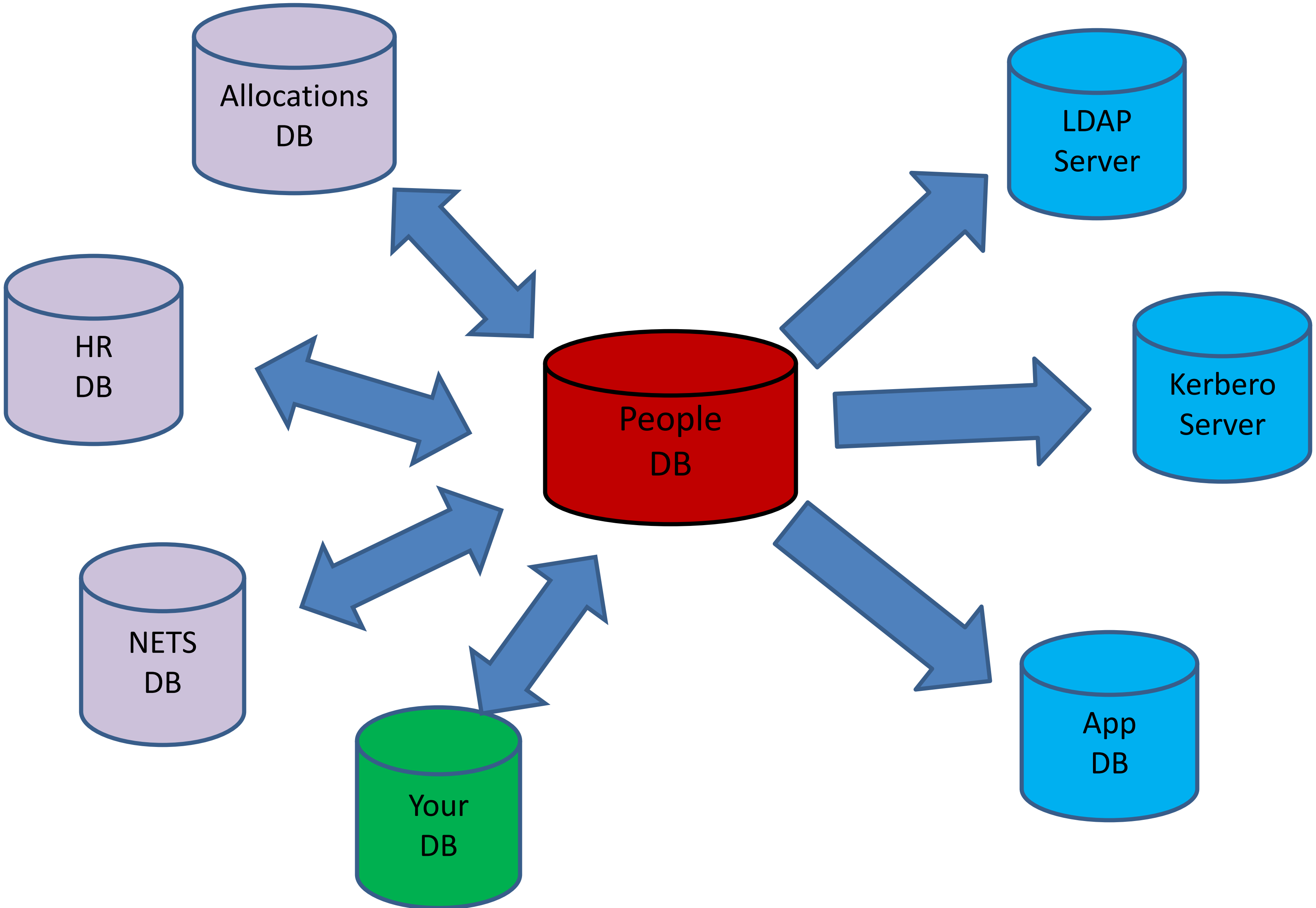

iVantage
NETS
Allocations DB
HR DB
Server
HR Provided Web Service API
Custom Synchronizer
Step 1.2 Invoke Synchronizer
Step 2.1 Push Changes to NETS DB
Step 2.1 Push Changes to ACC8 DB
Step 2.1 Invoke Custom Web Service to Sync
Step 3.1 Custom Synchronizer Pull the data from People DB
1.1
date
nges
Synchronizer
REST API
People DB

Figure 1. Add/modify the people data in central people DB and sync the changes to

Join the federation



Documentation & Code Examples

- https://wiki.ucar.edu/display/weg/People+DB+1.0