A Vision for abapGit 2.0 and the Future of ABAP Development

Marc Bernard

October 2020

“ABAP only supports one active version of an object, so users are forced to use the latest version.”
– Unnamed ABAP Developer

Yes, but…

“We can make unlimited copies of any version of an ABAP object and users will be able to decide which version they want to use”
– The Future

abapGit 1.x – Today

Status

- abapGit is THE tool for version control and source code management
- abapGit provides a ‘standardized’ way for exporting/importing
- abapGit works with traditional y,z objects as well as with SAP/customer/partner namespaces
- But abapGit is missing

Figure 1: abapGit projects developed in y,z-custom namespace (pretty much everything on dotabap.org)

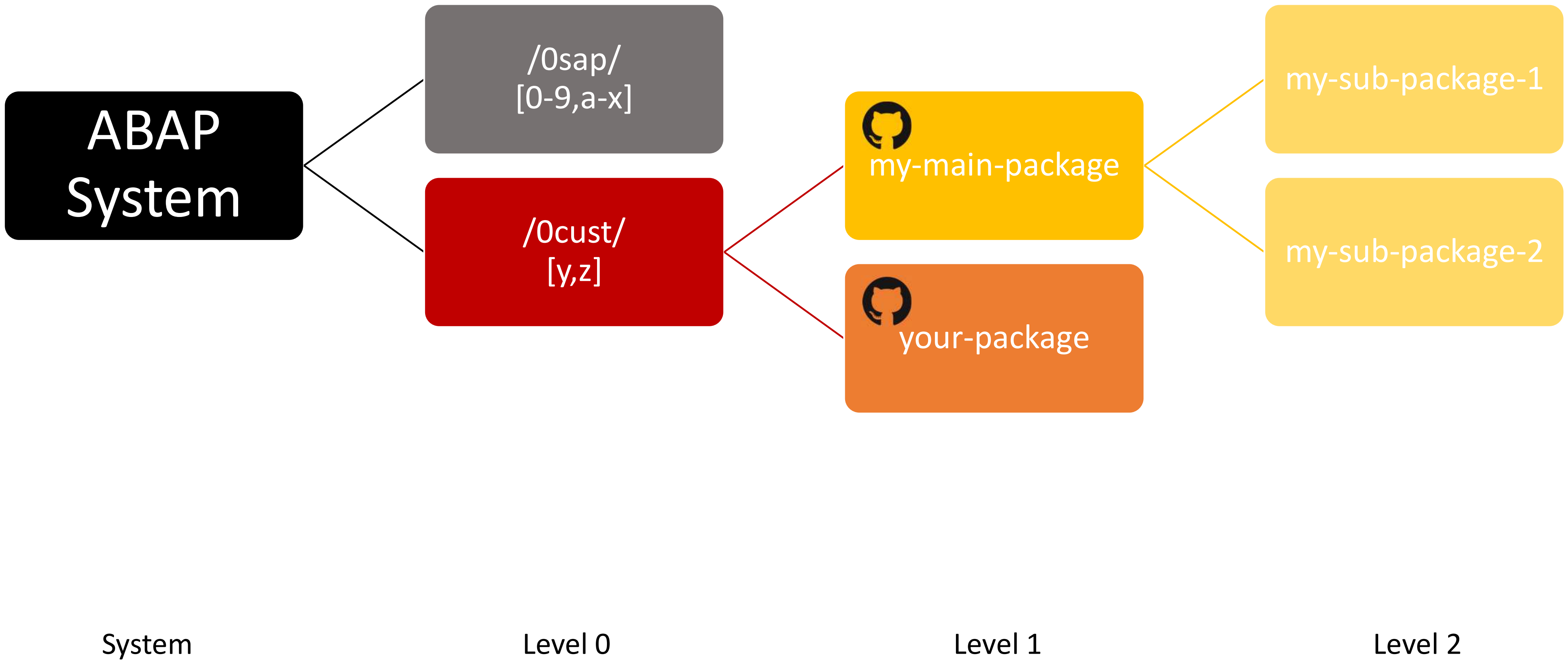
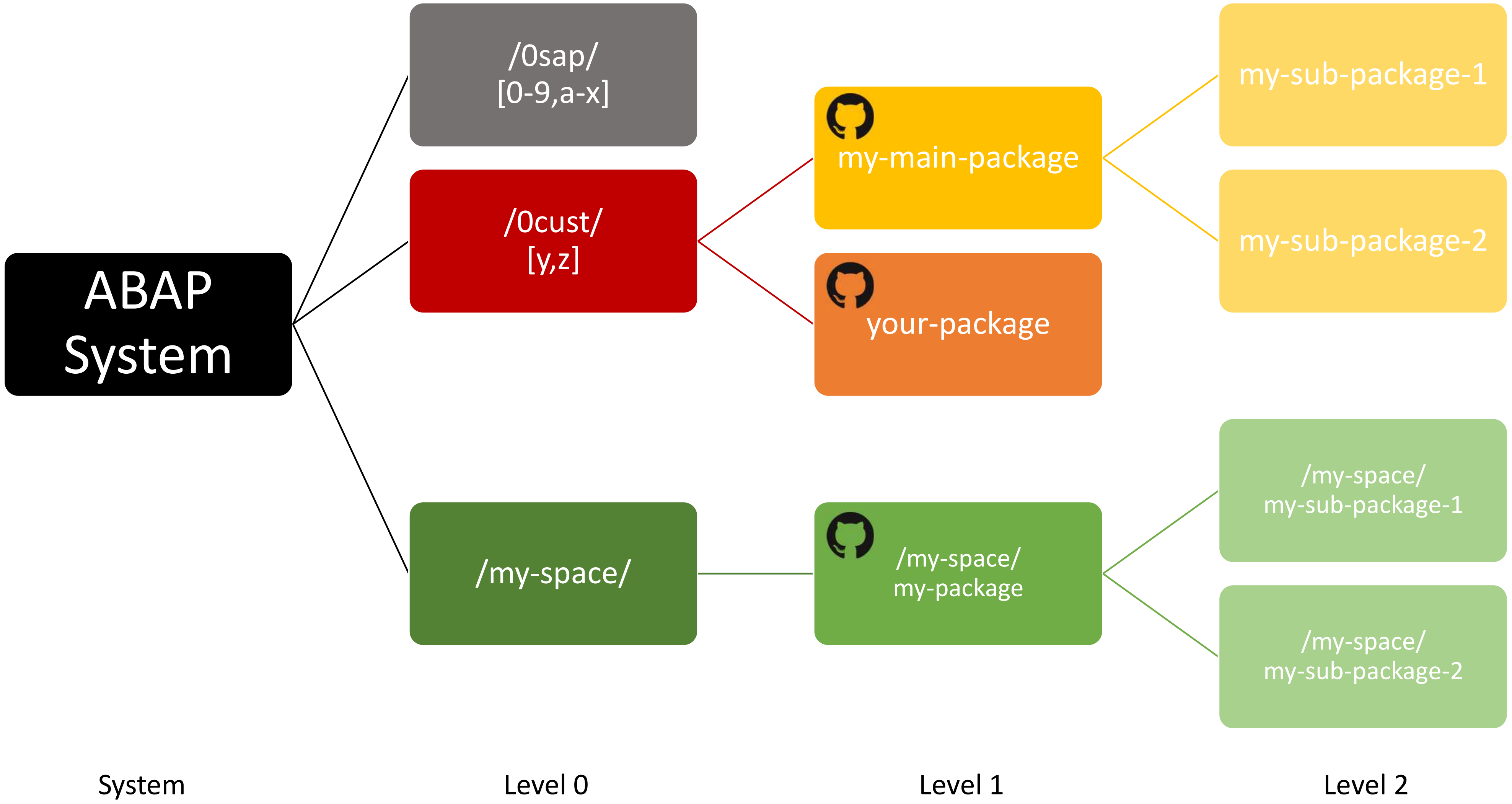


Figure 2: abapGit also supports projects using SAP namespaces

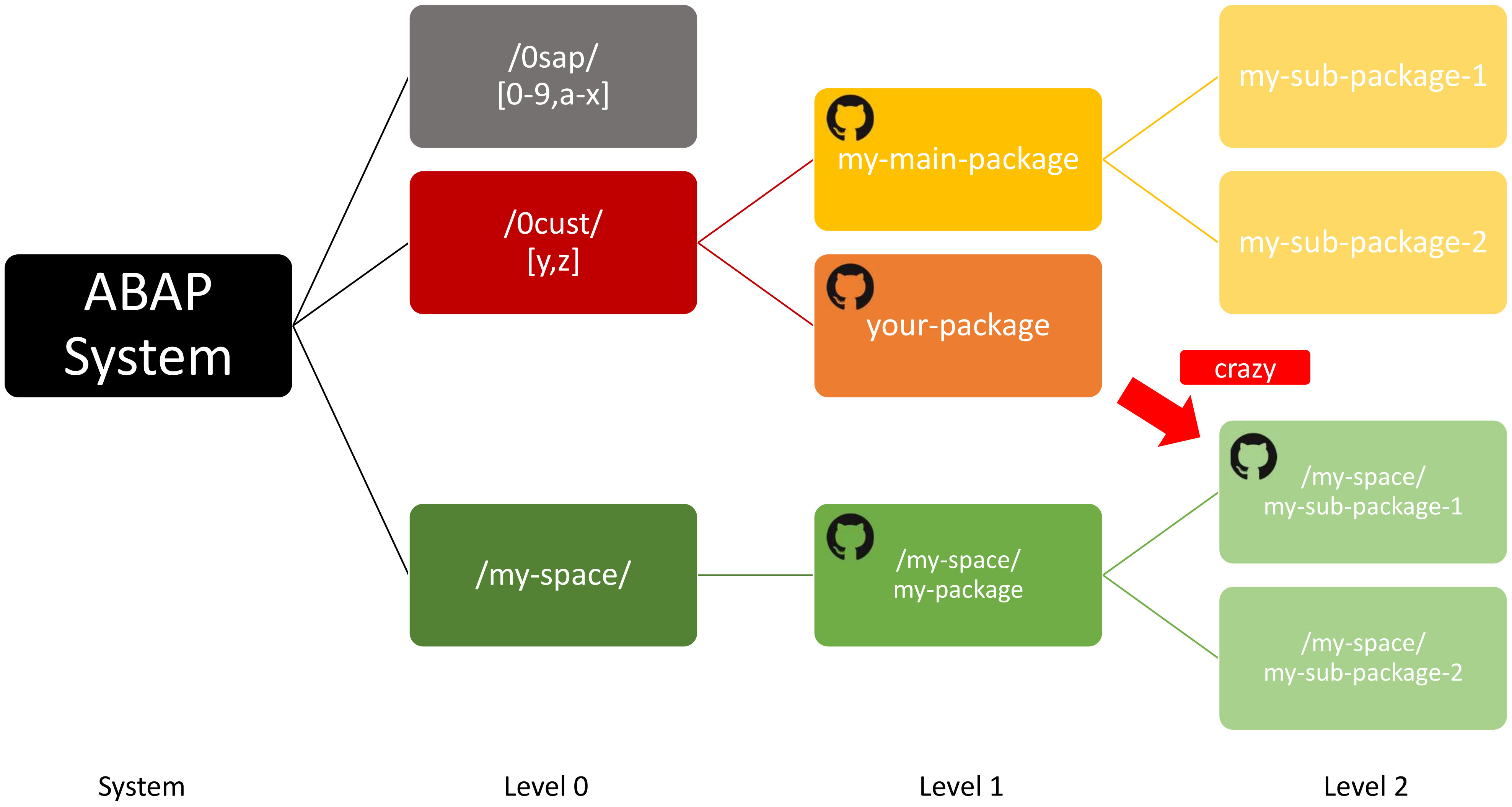


What’s Missing

- There’s only one active version of each object in ABAP
- It’s not possible to install/use different versions of the same code
- Workarounds like “if version = 1. do abc. else. do xyz. endif.” lead to unmanageable complexity

- But there is a way to have the same code multiple times…
  … by “copying and renaming” it
- This has been a requested feature for a long time…
  … but a complex task to do it in a consistent way (nothing standard)

Figure 3: Requested feature for abapGit, but OUT-OF-SCOPE due to complexity of consistently renaming objects



The World of Libraries and Packages

How do packages (libraries) work “on the internet” in general?

- Each package is developed separately and publishes versions of the code
- Each package specifies its dependencies to other packages (i.e. in package.json)
- Packages are installed into separate folders and contain a copy of the packages it depends on (for example, a copy of jquery@3.5.1)
- Therefore it’s possible that multiple versions of the same package exist on a (file) system
- A package manager keeps a catalog of published packages and versions and can evaluate and update dependencies automatically (for example, npm)

But how can we do this in ABAP?

The Idea:

Leverage abapGit and namespaces to mimic the “world of packages on the web”

Assume that /namespace/package is like a folder where you install a library. Therefore, each namespace can contain one version of a library. And with several namespaces, you can have several versions of a library in one system.

abapGit 2.x – The Future

Figure 4: abapGit – AS IS assuming repos use SAP namespaces

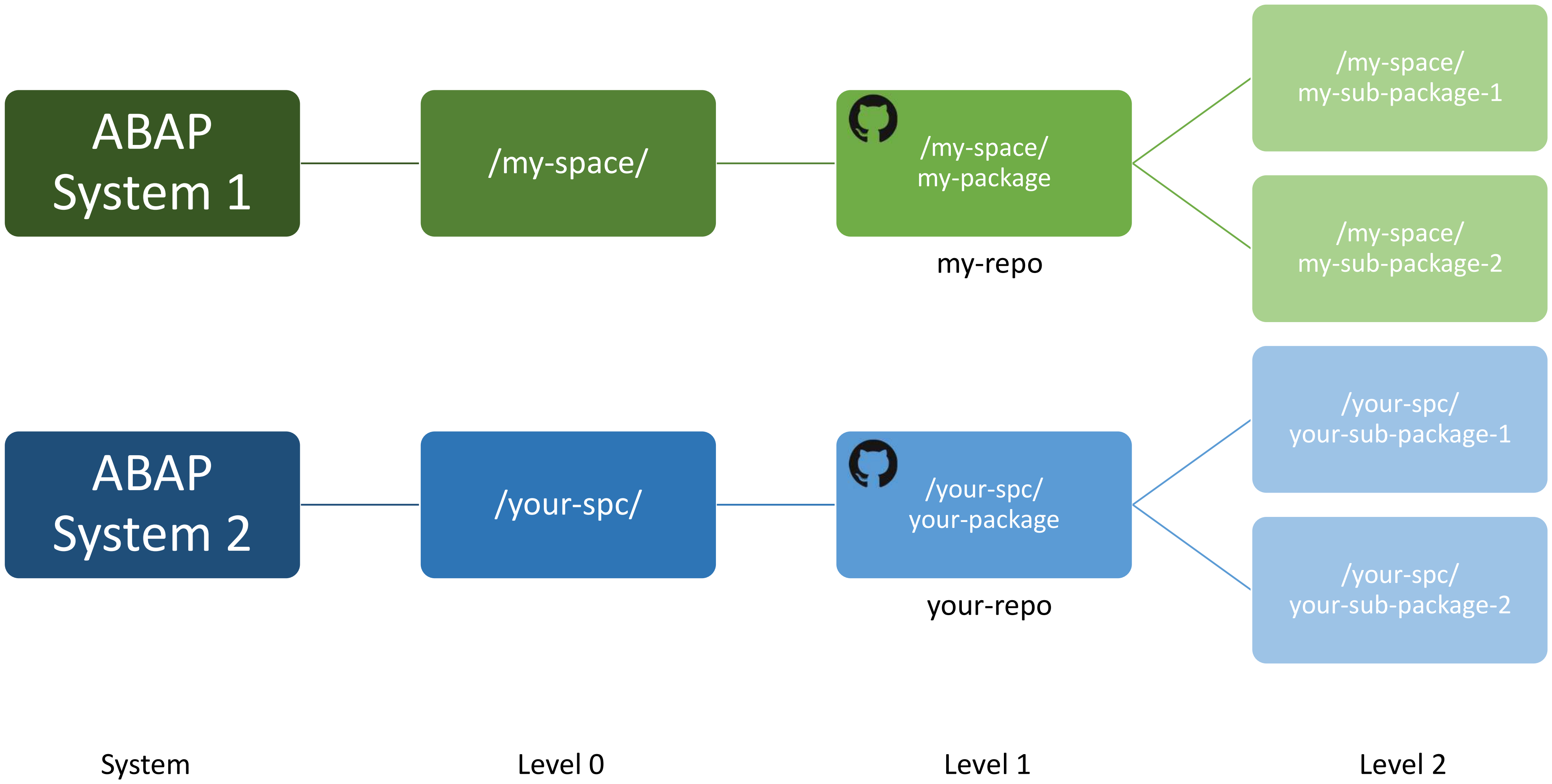


Figure 5: Ideal solution with **multi-level namespaces** (folders) and **long technical names** (but not possible in ABAP)

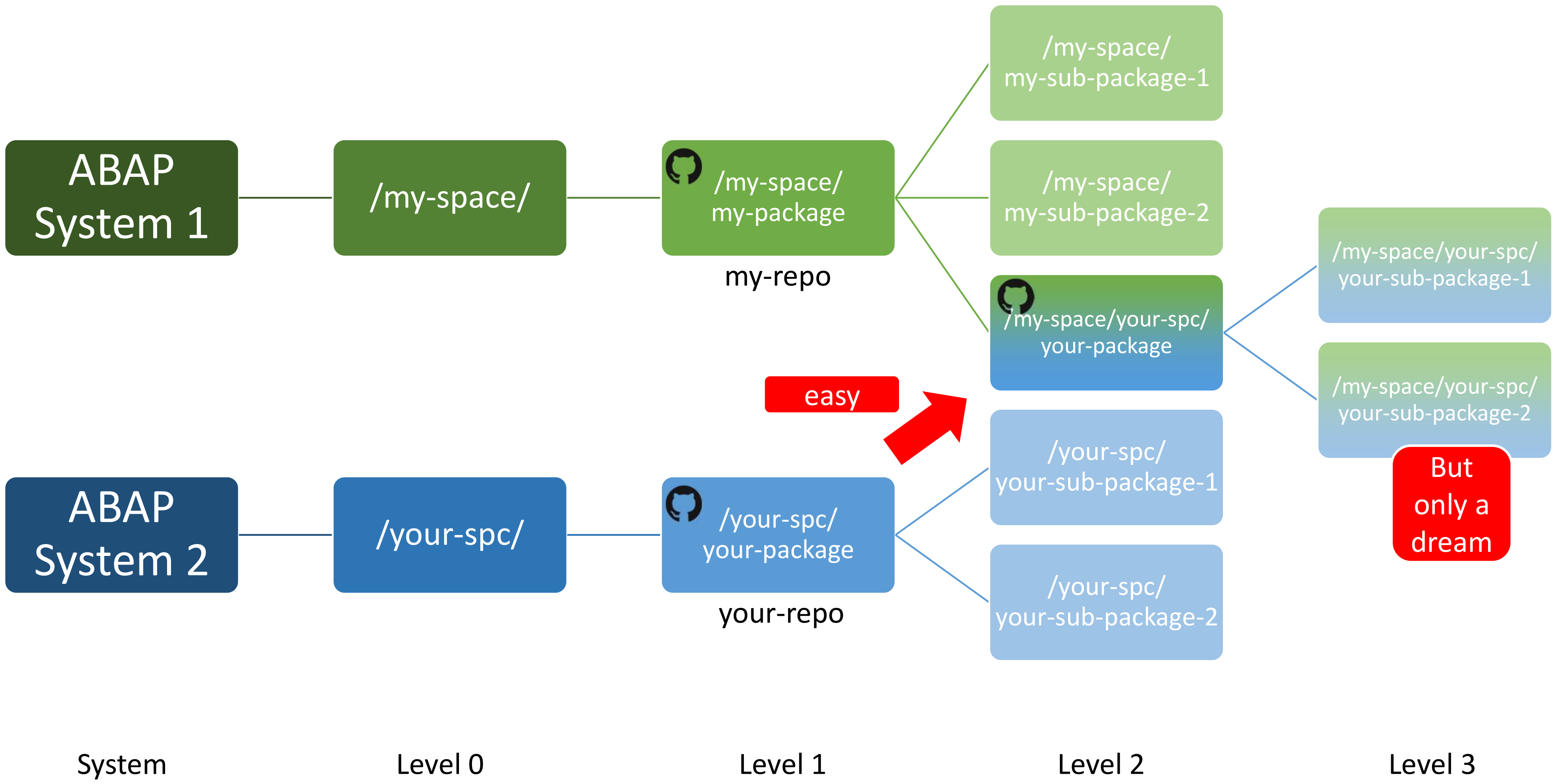
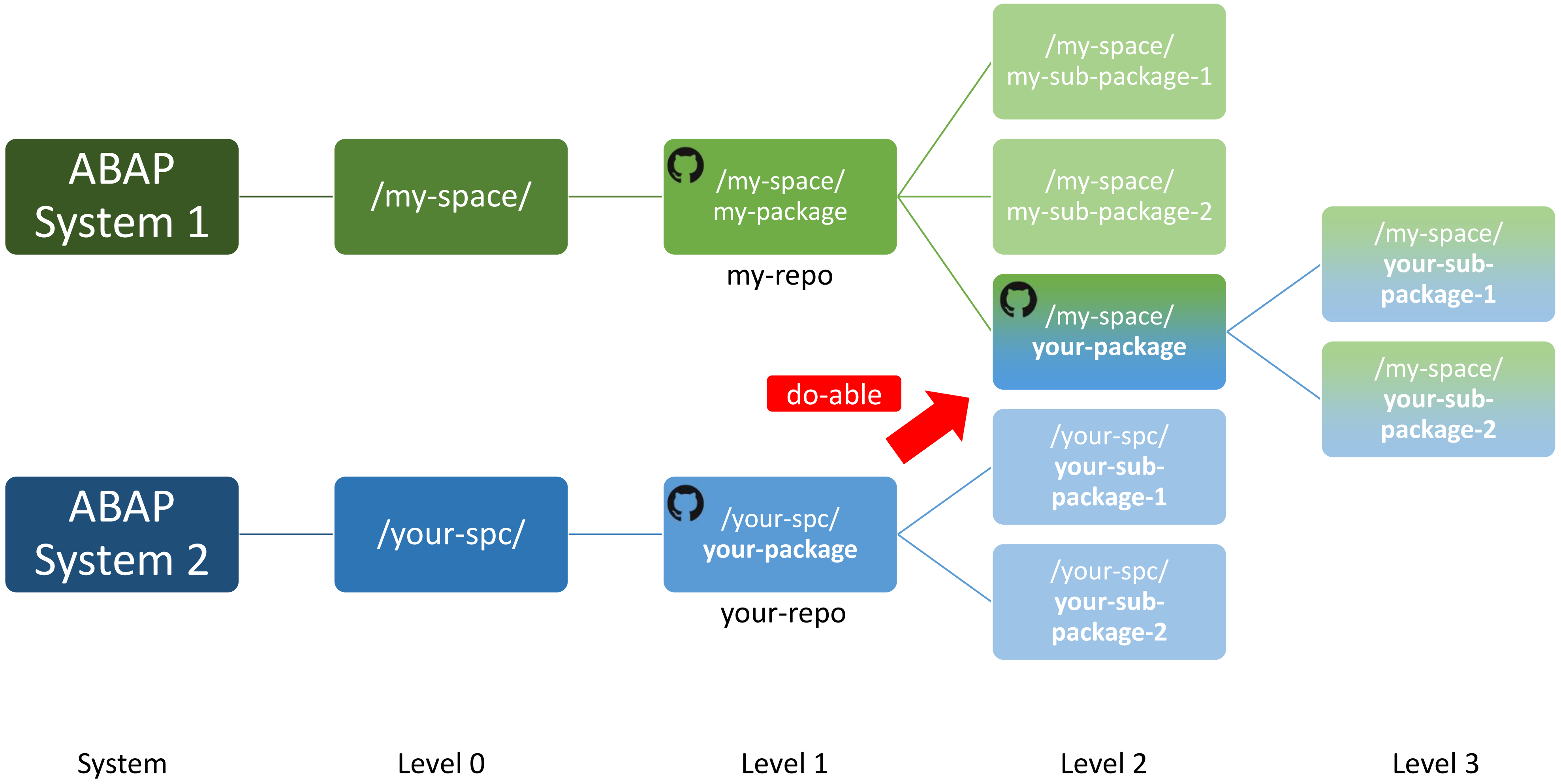

ABAP System 1
/my-space/
/my-space/ my-package
my-repo
/my-space/ my-sub-package-1
/my-space/ my-sub-package-2
/my-space/your-spc/ your-package
/my-space/your-spc/ your-sub-package-1
/my-space/your-spc/ your-sub-package-2
But only a dream
easy
ABAP System 2
/your-spc/
/your-spc/ your-package
your-repo
/your-spc/ your-sub-package-1
/your-spc/ your-sub-package-2
System
Level 0
Level 1
Level 2
Level 3

Figure 6: Do-able solution with single-level namespace


ABAP System 1
/my-space/
/my-space/ my-package
my-repo
/my-space/ my-sub-package-1
/my-space/ my-sub-package-2
/my-space/ your-package
/my-space/ your-sub-package-1
/my-space/ your-sub-package-2
do-able
ABAP System 2
/your-spc/
/your-spc/ your-package
your-repo
/your-spc/ your-sub-package-1
/your-spc/ your-sub-package-2
System
Level 0
Level 1
Level 2
Level 3

How shall it work in abapGit?

When creating a new repo, you specify a target namespace and package that is different from the namespace used in the repo.

During a pull, abapGit would rename all files (i.e. change the namespace in the filename) and do a global search & replace of the namespaces in all files. This should work well since /namespace/ is an easily identifiable pattern in .abap and .xml files.

abapGit would set the new repo to "write-protected remote" (i.e. pulls are allowed but commits to the remote are not).

Example

You want to develop an application and reuse the “libDemo Package”

1) libDemo is developed in namespace /libdemo/

   As usual developers of libDemo have the most recent version on their system. Published versions are available as git release tags.

2) Your application is developed in namespace /appdemo/

   You want to use a specific version of libDemo in your project (not necessarily the latest).

Figure 7: Development of libdemo package

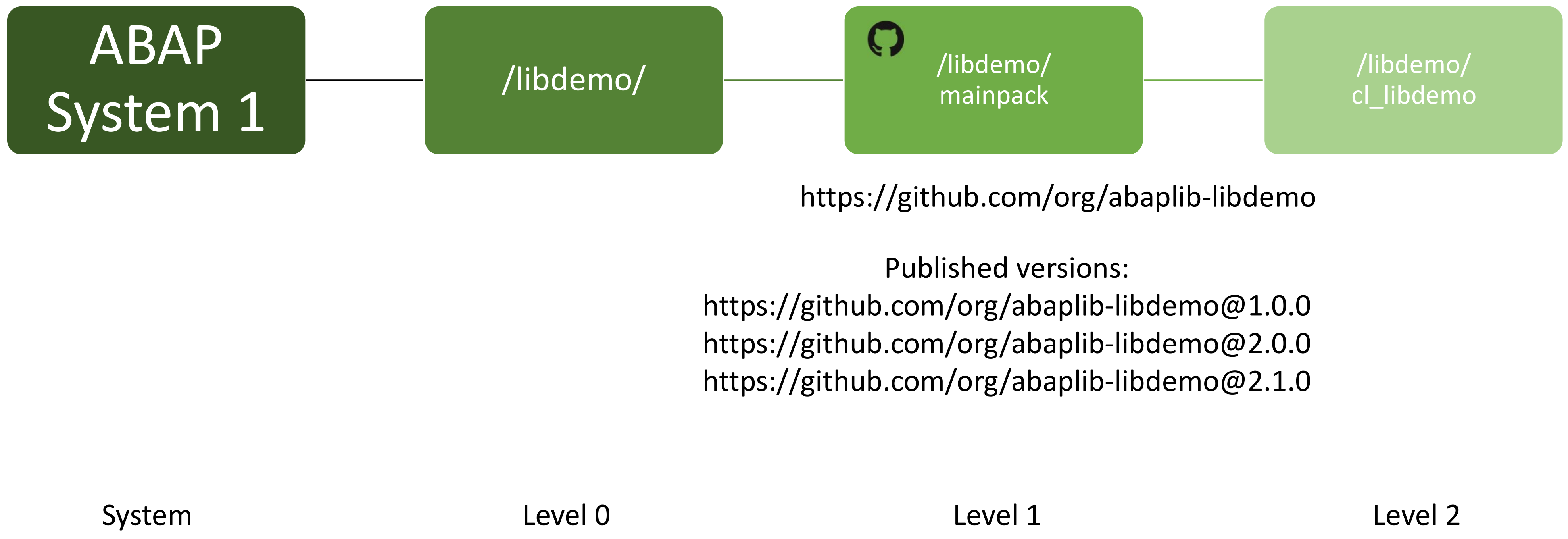

ABAP System 1
/libdemo/
/libdemo/ mainpack
/libdemo/ cl_libdemo
https://github.com/org/abaplib-libdemo
Published versions:
https://github.com/org/abaplib-libdemo@1.0.0
https://github.com/org/abaplib-libdemo@2.0.0
https://github.com/org/abaplib-libdemo@2.1.0
System
Level 0
Level 1
Level 2

Figure 8: Development of your application in /appdemo/mainpack

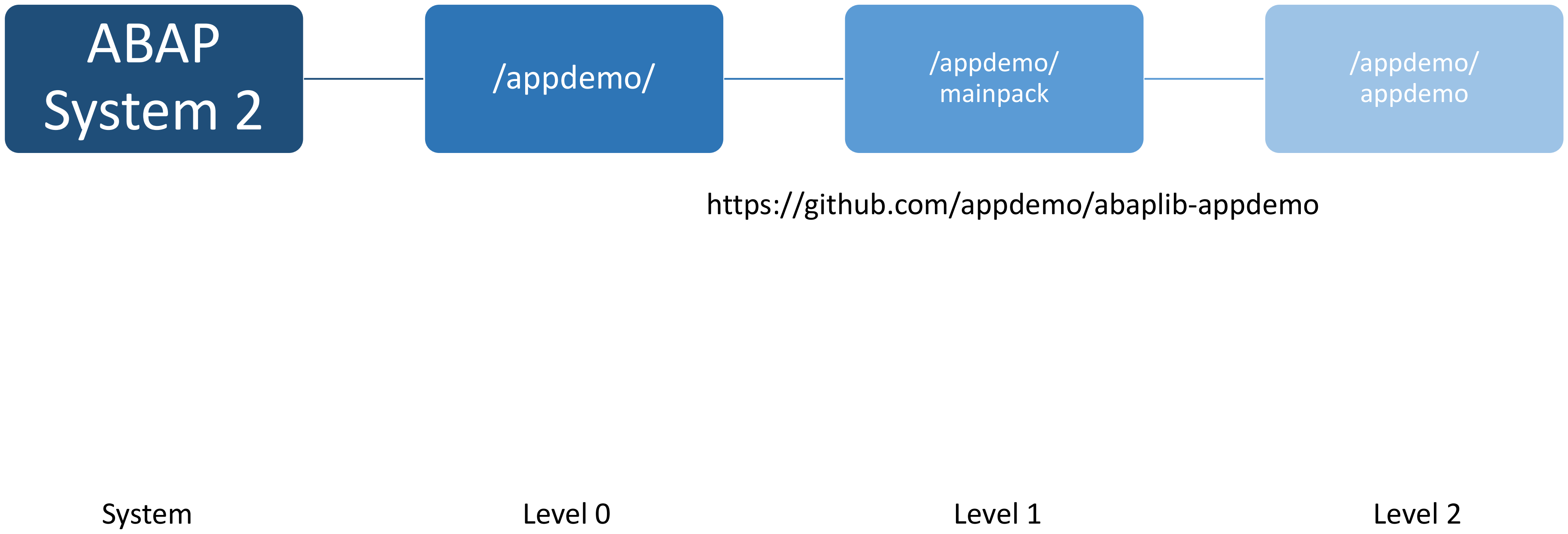
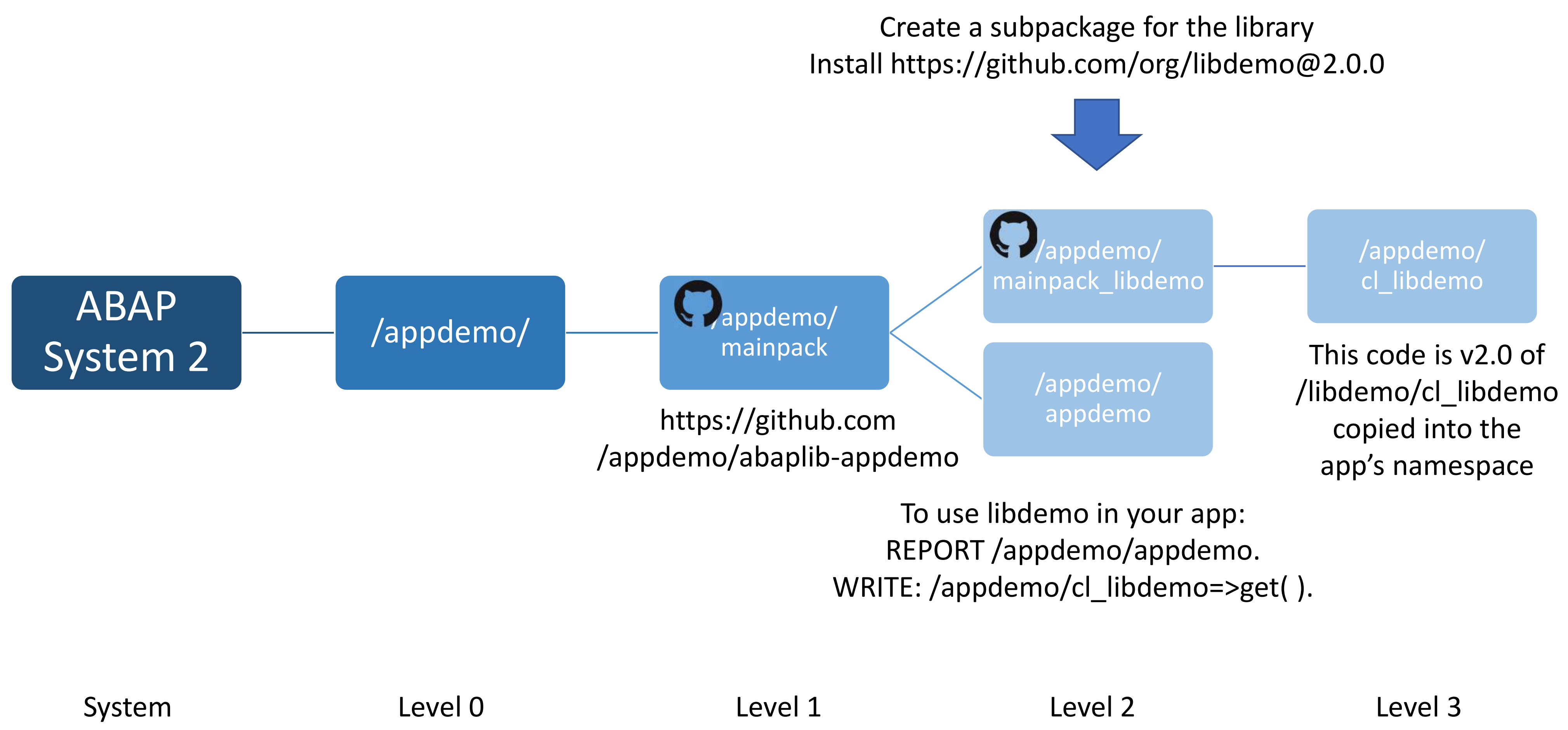


Figure 9: Create package /appdemo/mainpack_libdemo and install the libDemo v2.0 repo into it.
abapGit shall adjust namespace automatically. (note: pushing back to org/libDemo repo must be prevented)
Now your repo will include the version 2.0 of libDemo. When there's a newer version of libDemo, simply pull again.


Create a subpackage for the library
Install https://github.com/org/libdemo@2.0.0
ABAP System 2
/appdemo/
/appdemo/ mainpack
https://github.com /appdemo/abaplib-appdemo
/appdemo/ mainpack_libdemo
/appdemo/ appdemo
/appdemo/ cl_libdemo
This code is v2.0 of /libdemo/cl_libdemo copied into the app's namespace
System
Level 0
Level 1
Level 2
Level 3

To use libdemo in your app:

```
REPORT /appdemo/appdemo.
WRITE: /appdemo/cl_libdemo=>get( ).
```

Introducing “abapLibs” – ABAP Libraries

An “abapLib” is a specialization of an abapGit repo (=a mode/setting chosen when creating the repo and enforced during development)

- Must follow a standard for repo versioning (package.json)
- Must be developed in a namespace
- Therefore only object types are supported that support namespaces
- Max length of object names shall be limited so that all target namespaces are possible (i.e. regular max length - 10)
- Recommendation: Object names should contain name of library to avoid name conflicts

The more you think about it…

- You don’t want to use the latest development version of an abapLib (repo) in your project
- You want to decide which version of the abapLib you need and keep it stable.
- But you eventually want to update the abapLib and test it with your project
- You also might want to modify the code of the abapLib in your system (or as a fork)
- Therefore, copying all objects of the abapLib into your namespace absolutely makes the most sense.

Imagine

How much time could you save if you could reuse “abapLibs” in your projects?

What if many developers would build “abapLibs” and make them available as open source?

What if each “abapLib” comes with a package.json and you could run a “dependabot” for your project?

What if SAP would restructure SAP Basis into “abapLibs” and with independent release cycles and well-defined dependencies?

What if SAP would allow for multi-level namespaces and long technical names for everything? … ok, ok, I’ll stop dreaming