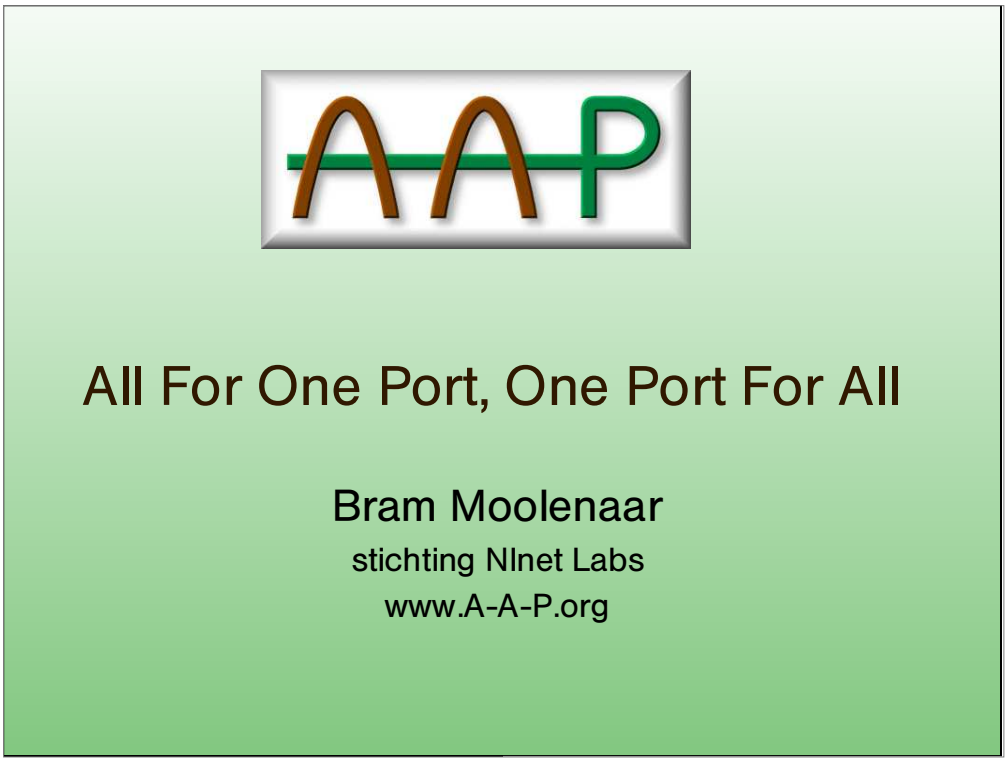
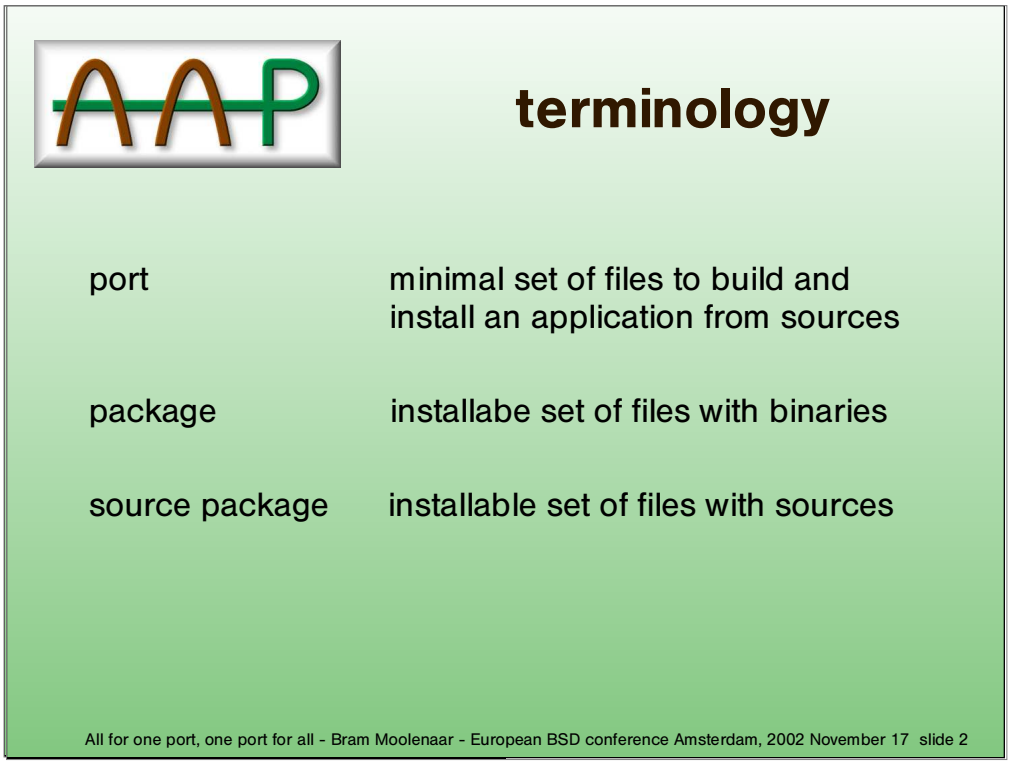


Presentation given by Bram Moolenaar at the European BSD conference, November 17 2002, Amsterdam.

You may know my name for being the main author of Vim, the editor.

My current daytime work is the A−A−P project. I'm the project leader.

For my daily work I use FreeBSD (still version 4.5, should upgrade soon).
Mostly using lots of xterms with Vim in them.



I assume I do not have to explain what the ports system is and how it works..

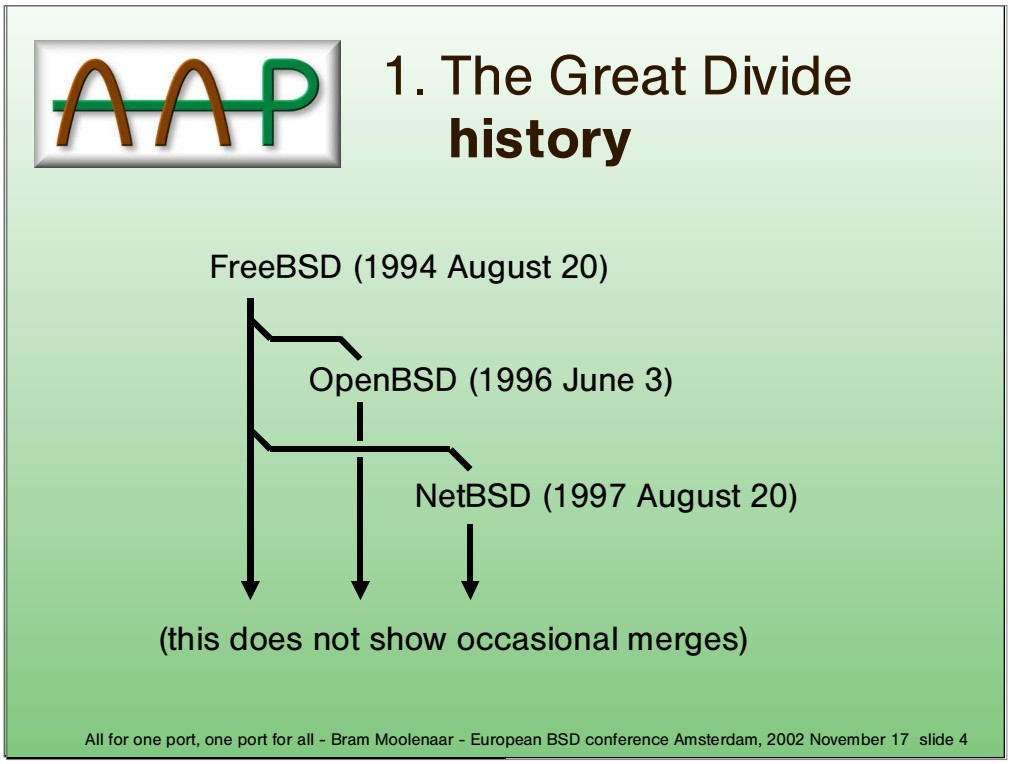
In this presentation I will be using the FreeBSD terminology.

NetBSD uses different terms, because they use "port" for using NetBSD on a different platform.

A source package is more or less a port with the used source files included.

overview

1. **The Great Divide**
2. A New Solution
3. Introducing A-A-P
4. The Port Recipe
5. Conclusions



Jordan K. Hubbard started the ports system, originally with just a few features.

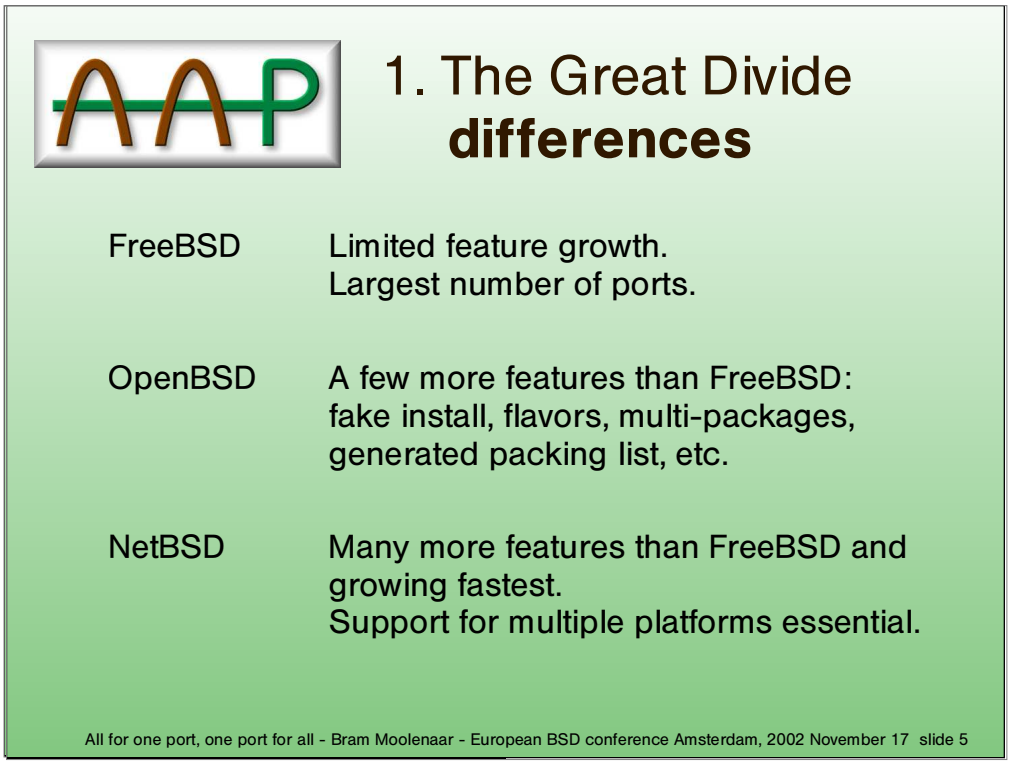
All systems still contain the credits for Jordan with the date in 1994. This clearly shows their common root.

The figure only shows the first split−off for each system. There were various exchanges of modifications in between.

The split−offs mostly happened to add features or solve problems, with the intention to feed the changes back to FreeBSD.

These days each system appears to be developed independently, without the desire to include each others modifications. It has also become more difficult to include each others changes.

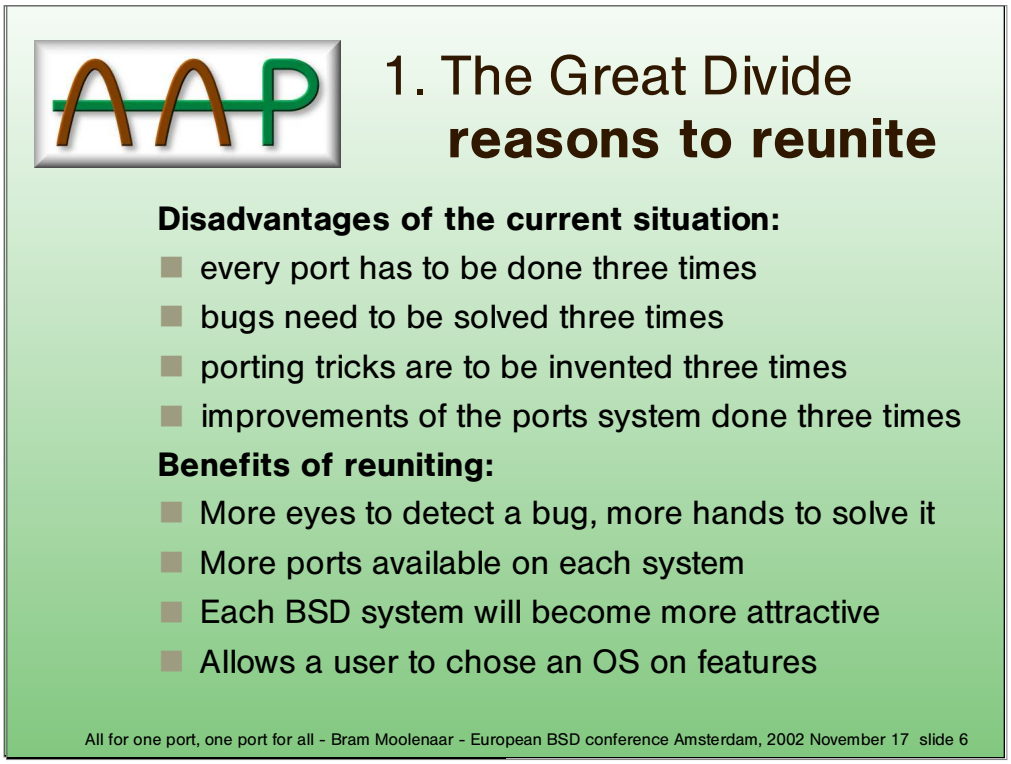
The three systems are growing further apart. Not intentional, but because of the lack of motivation to to work together. Being able to make changes directly instead of having to convince someone else to include them also plays an important role.



Ironically the OpenBSD documentation refers to the FreeBSD handbook. Thus besides the extra features it appears to be mostly the same.

Overall the differences are not that big, still using the same mechanisms.

However, the differences are big enough to make the ports files incompatible and taking over changes difficult.

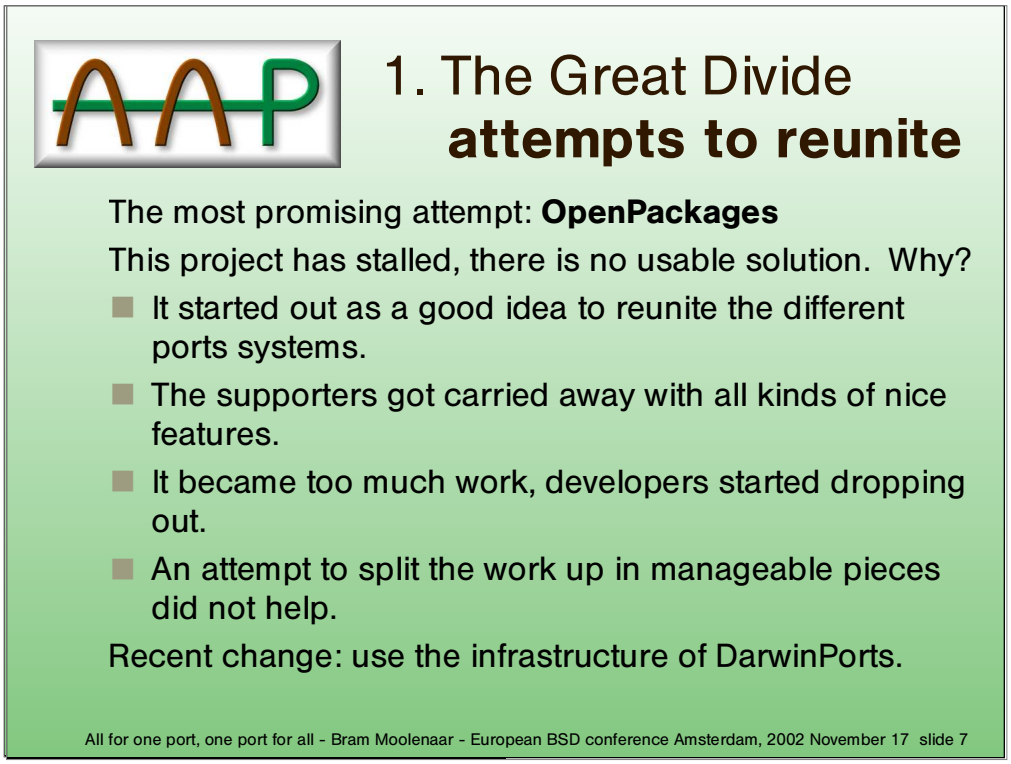


Obviously, maintaining three systems with ports is three times as much work. Since there are thousands of ports, the amount of duplicated work is significant.

Currently a user might chose one of the BSD systems because it is the only one that has a port he needs to use. If all systems have the same ports he can make his selection based on other features.

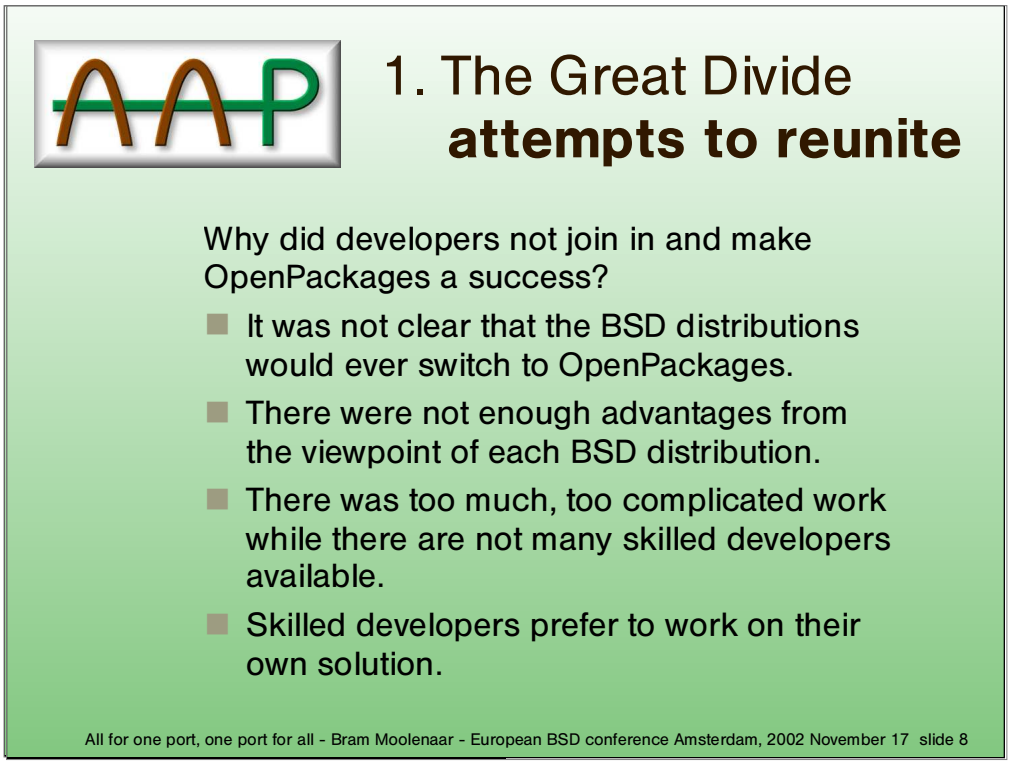
For a group of developers this may seem unimportant. Look at it from the point of view of a user. He just wants to have the port he needs to use.

One thing will not change when reuniting: testing still needs to be done for each platform.



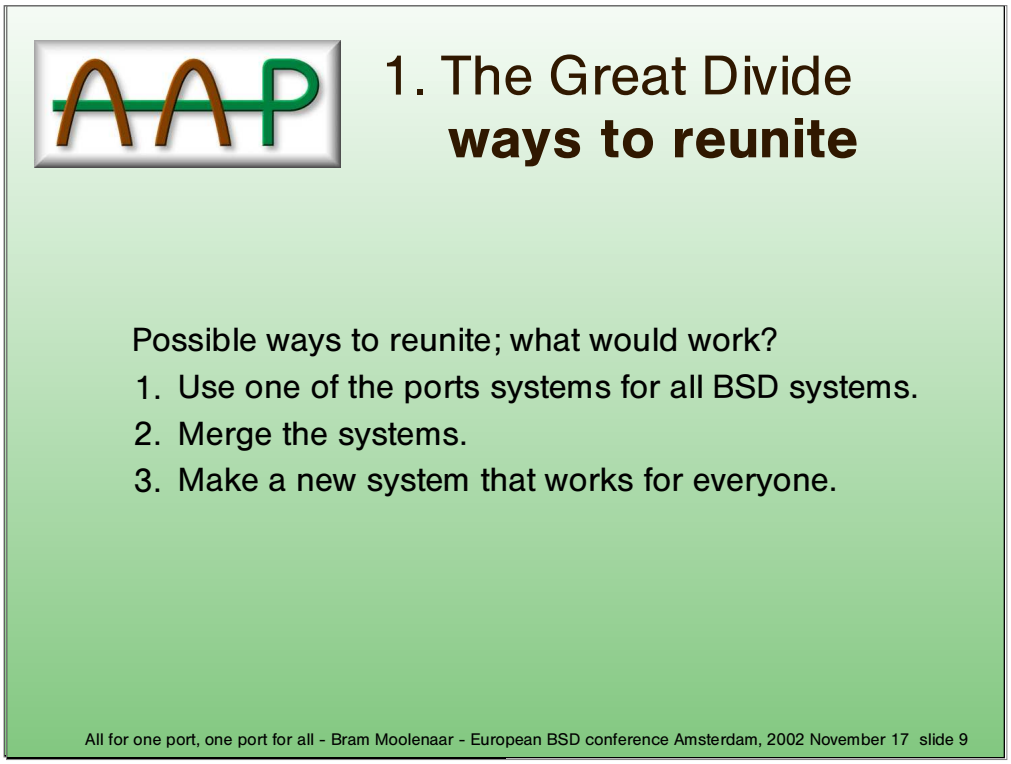
Originally this project started with the intention to make one ports system for all BSD systems.

Whether the cooperation with DarwinPorts will really work is unclear. It is not mentioned on the DarwinPorts maillist No messages in the OpenPackages−tech maillist since the announcement.



OpenPackages did sound like a good idea, so why did it not work out?

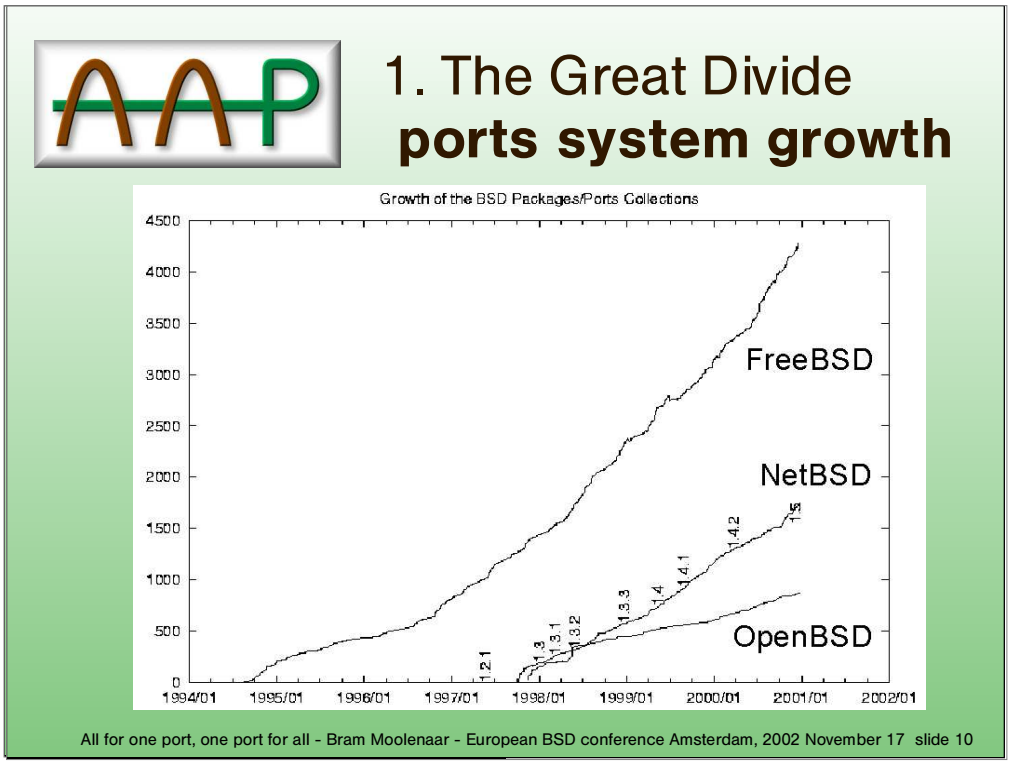
The project leader is not a developer himself, thus he had to rely on others to do the implementation.



Let's make a list of possible ways to get back to one ports system.

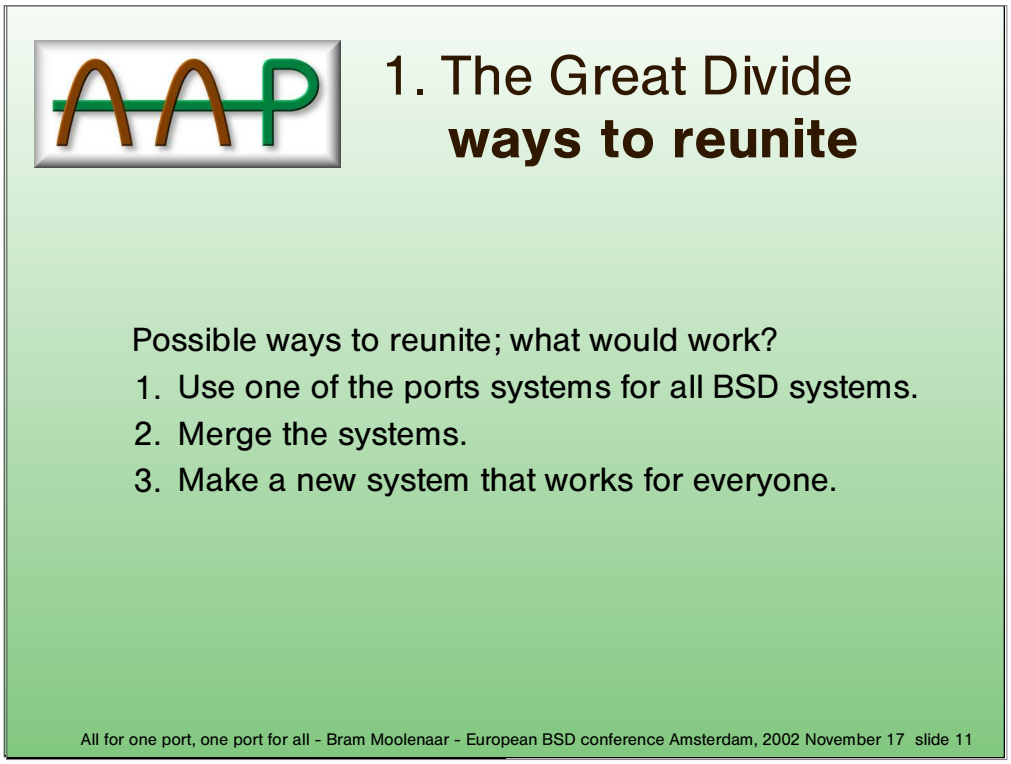
If wanted, it's always possible to take over one of the other systems. This didn't happen, and is extremely unlikely to happen in the future.

SEE NEXT SLIDE



Since FreeBSD has the smallest number of extra features, it is the most likely one to take over one of the other ports system implementation. However, since FreeBSD has the largest number of ports, the incompatibilities are a big problem.

Graph made by Hubert Feyrer in 2001.



The tendency is that merging of individual changes is decreasing. There appears to be no reason why this would turn around.

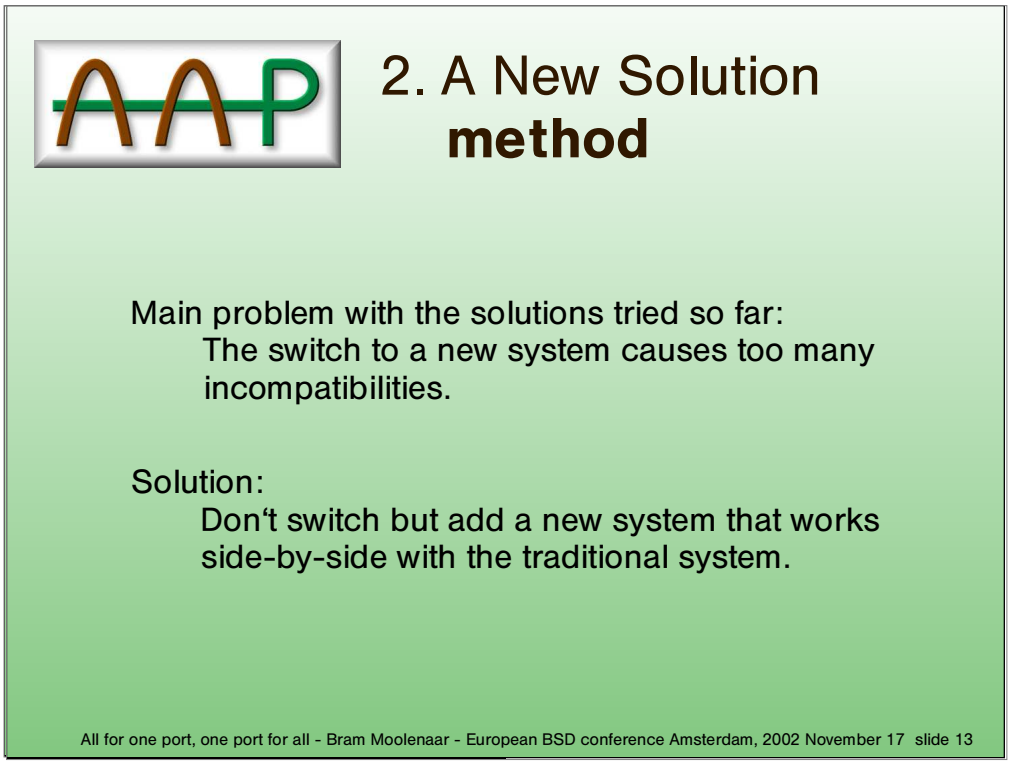
OpenPackages has shown that making a new system is unlikely to work.

Main problem here is that the port files are incompatible. Switching to another system would mean thousands of ports are suddenly unusable.

So what will work...?

overview

1. The Great Divide
2. **A New Solution**
3. Introducing A-A-P
4. The Port Recipe
5. Conclusions



This is the essential choice for the proposed solution. Other choices that follow depend on personal preferences..

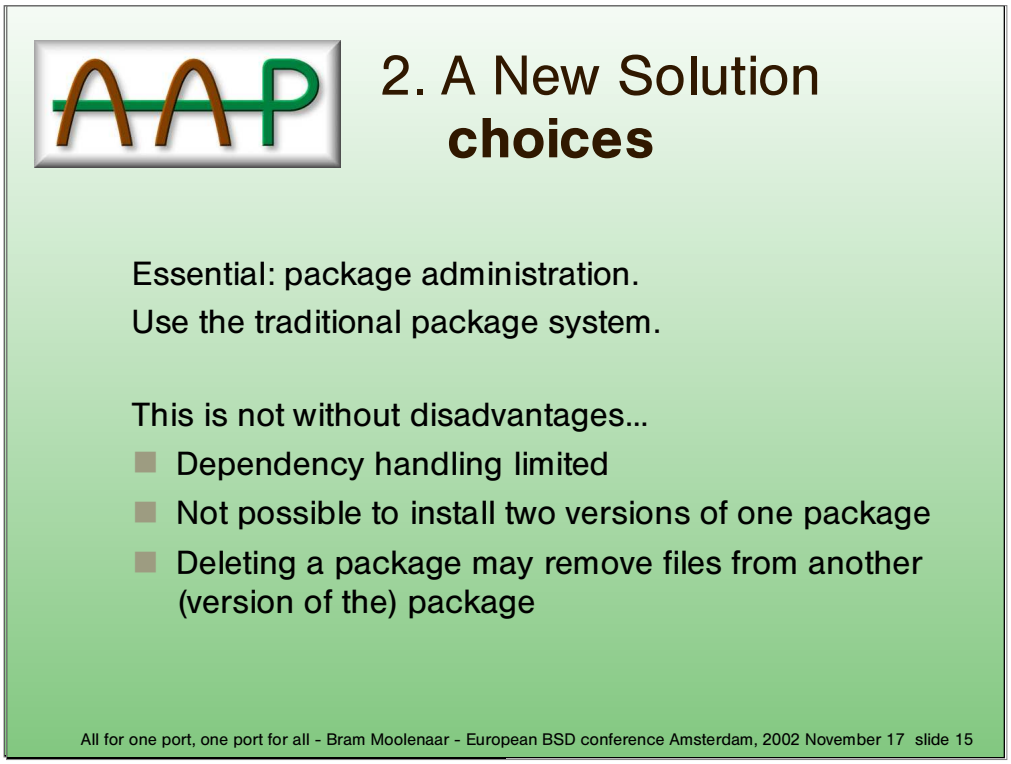
Note the terminology: traditional ports system and new ports system

The idea is that gradually more and more ports will be done for the new system. Eventually the traditional system can be discarded.

Although the port files do not need to be backwards compatible, it would be nice to use a similar structure to avoid scaring of port maintainers.

Not having to be backwards compatible leaves room for a better and more powerful implementation.

An example of what can be automized is the list of files. Especially because this may change often.



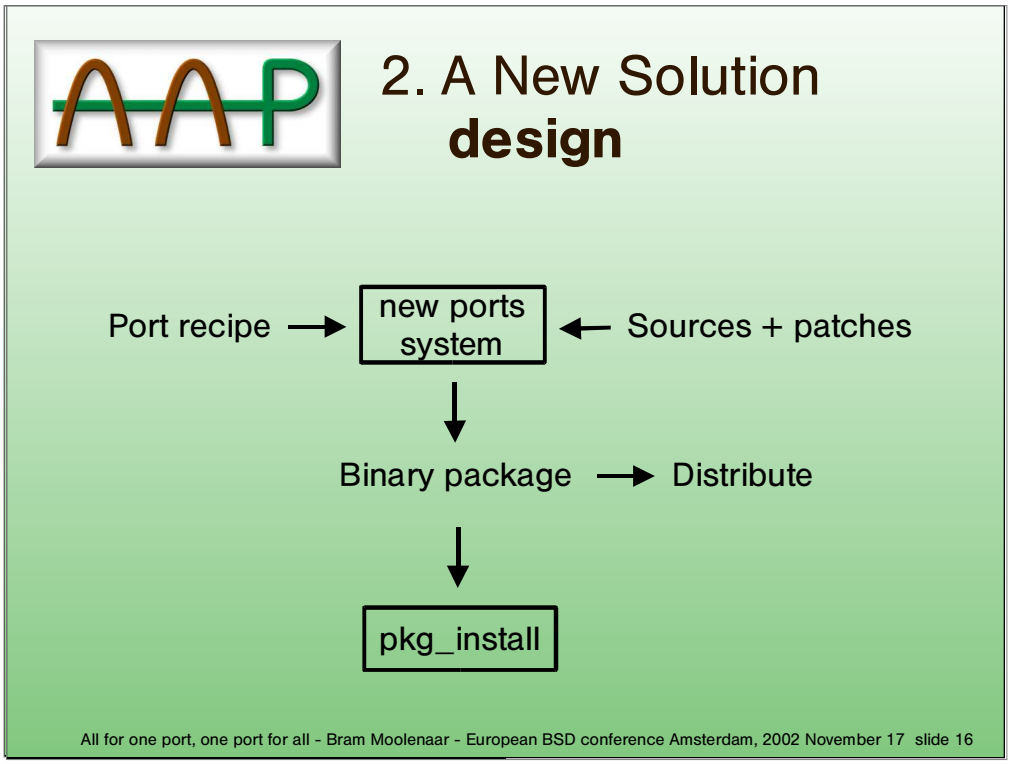
Only when the traditional package system is used is it possible to keep the registration of installed packages consistent.

The disadvantages are generic, they should be solved anyway. And these can be solved without incompatibilities with the traditional ports system.

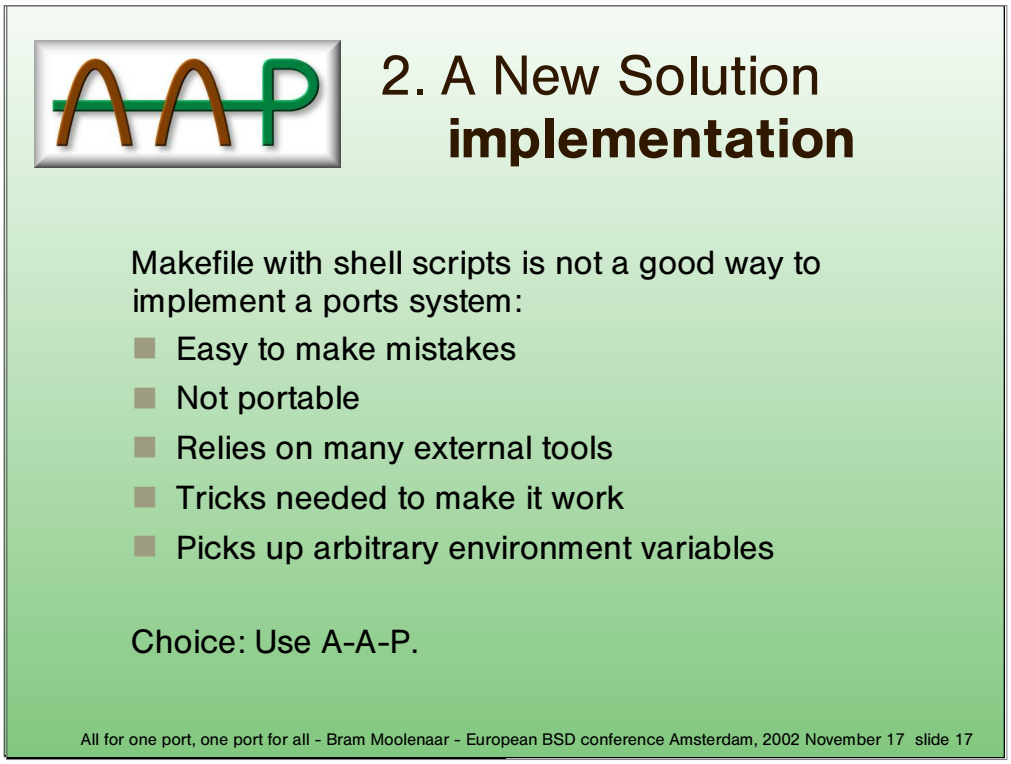
Example: Install version 1.1, install version 1.2, delete version 1.1. Files that are identical in both releases will be deleted (on FreeBSD at least).

Adding a new set of package tools is not a good alternative, since dependencies between the traditional and the new system cannot be handled.

There is no big need for a reunited package system. The new ports system can be made to work with all package systems.



The Binary package can be distributed and used just like a package created with the traditional ports system.



External tools that are standard UNIX items should be OK. But it's too easy to rely on specific versions (e.g., GNU tools intend of BSD ones).

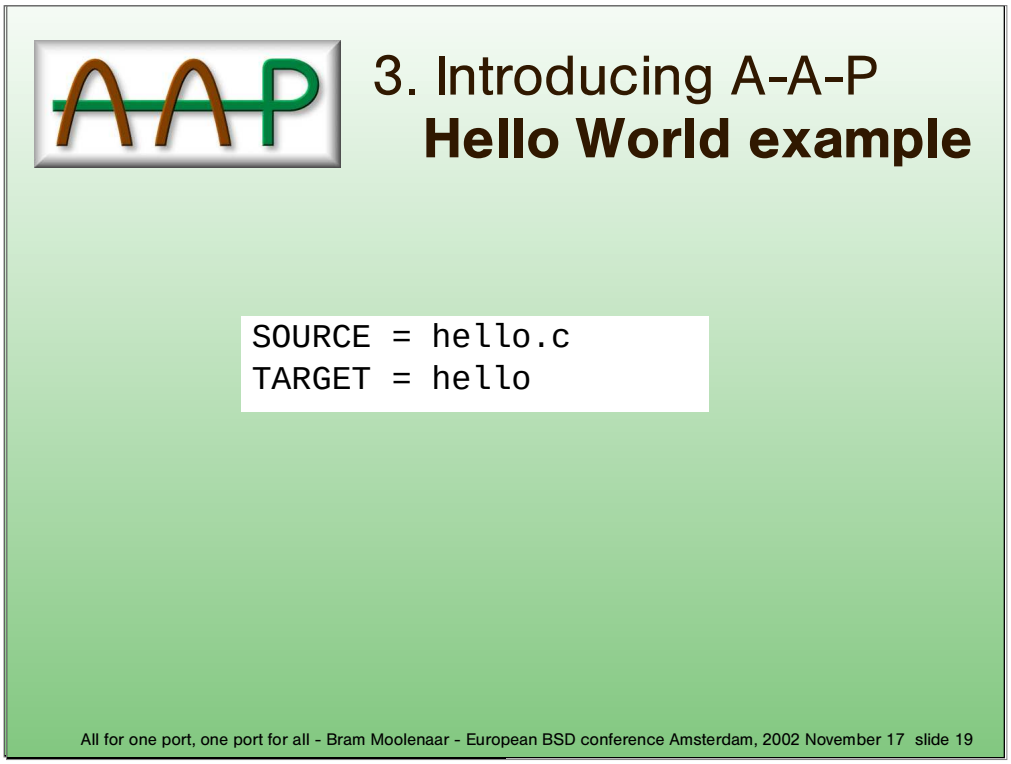
The choice for A−A−P is more or less a personal preference. There are alternatives that others might find a better choice.

To verify that these ideas are really possible, a proof of concept has been made with A−A−P.

A−A−P is a system for developing, distributing and installing software. Here we will use the build tool part of it. That's also the only part that's currently available.

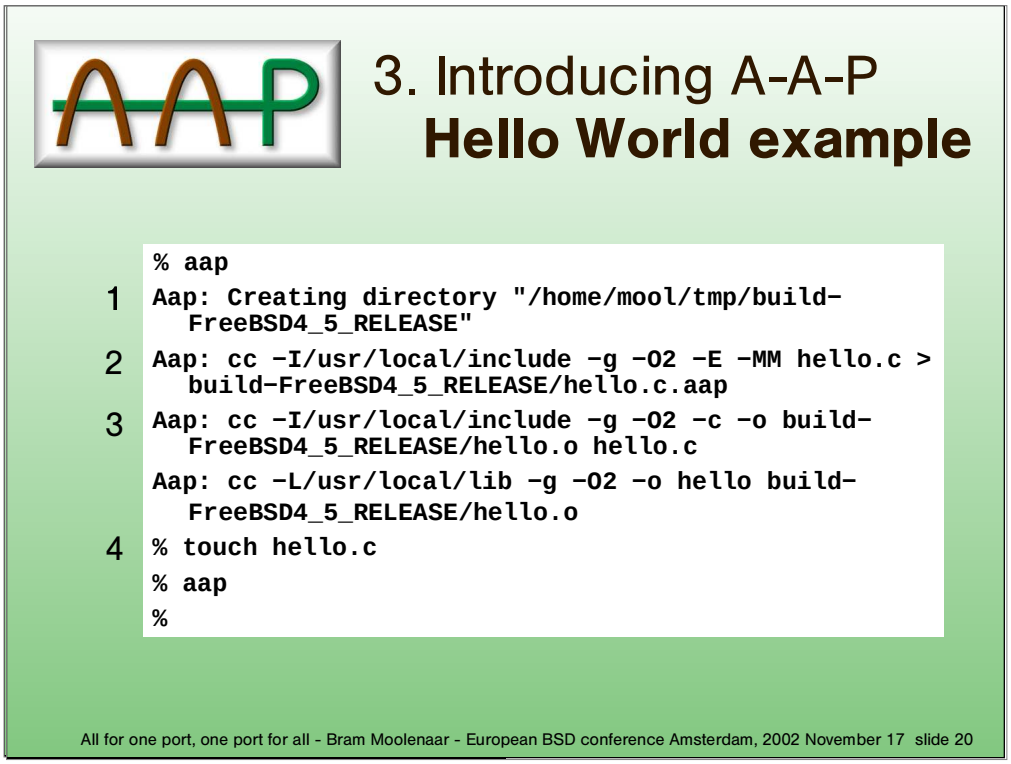
Even if you don't like my ideas for the new ports system, you might still want to use A−A−P.

I'm just giving an overview here, this is not a lecture. If you want to learn using A−A−P read the tutorial on the A−A−P web site.



Most computer languages start with an example that just prints "Hello World". This is an A−A−P recipe to compile the C version of this program.

This is really the whole thing. Store this recipe as "main.aap" and run the "aap" command. Let’s see what happens

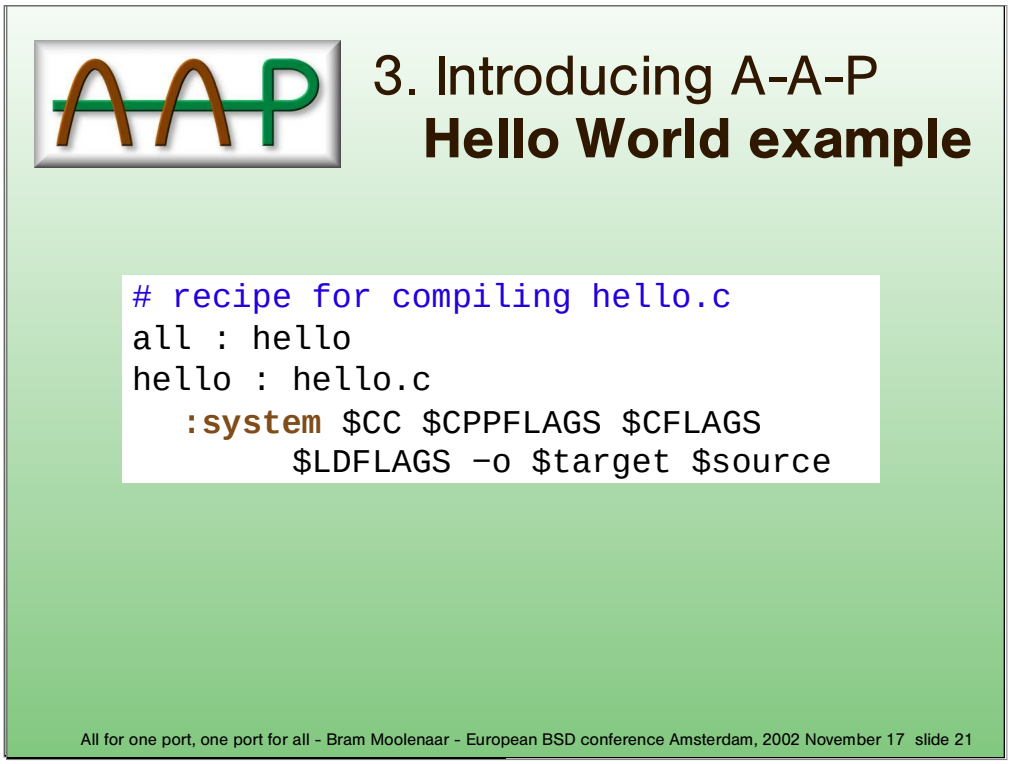


(1) First of all a directory is created to put the intermediate results in. The name is different for every platform, so that compilation for multiple platforms is automatically supported.

(2) The first invocation of "cc" obtains the dependencies on header files. A−A−P automatically detects these dependencies, which is a lot simpler and more reliable than doing this by hand.

(3) The next two invocations of "cc" compile and link the program.

(4) When touching the source file A−A−P knows it does not need to be compiled, because it uses signatures. This is more reliable than timestamps, especially when working with networked file systems, unpacking an archive or restoring an older version.



This is an example that doesn't use the automatic mechanism for SOURCE and TARGET.

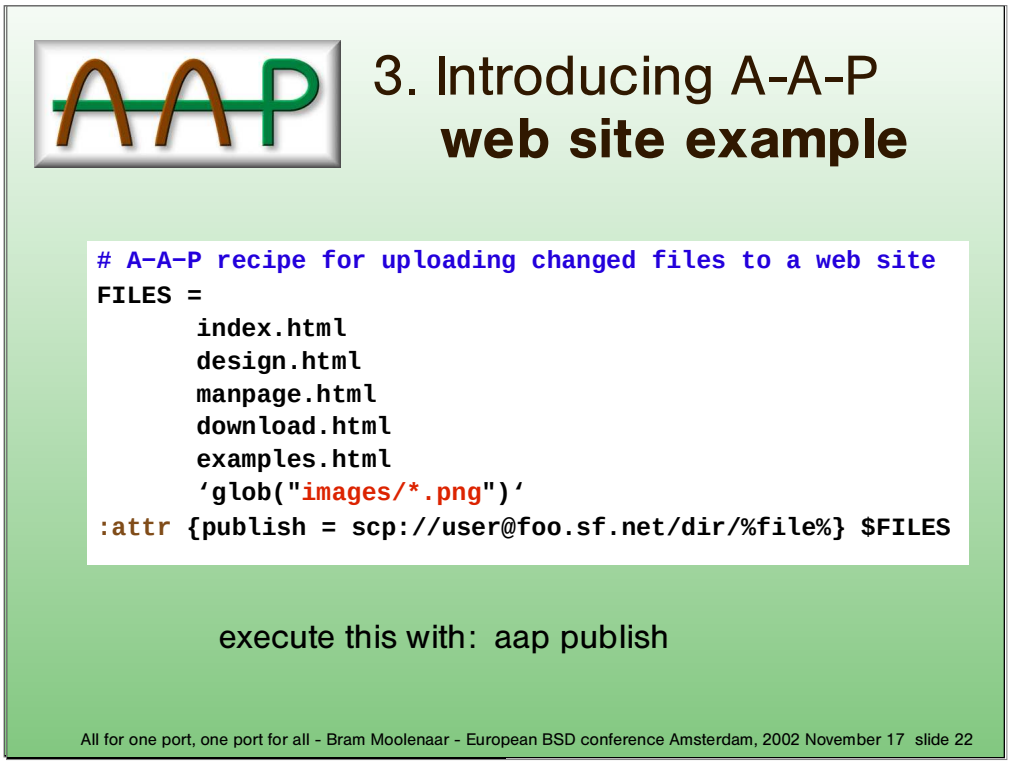
The recipe format is mostly like a Makefile, but there are a number of important differences.

The comments and dependencies are just like in a Makefile.

Build commands can be more than shell commands. A comprehensive set of often used commands is available. Therefore shell commands need to be preceded with ":system" or ":sys".

Line continuation does not require a backslash at the end of the line. The indent shows where the next command starts.

There is mostly no need to put parenthesis or braces around variable names, but it is allowed.



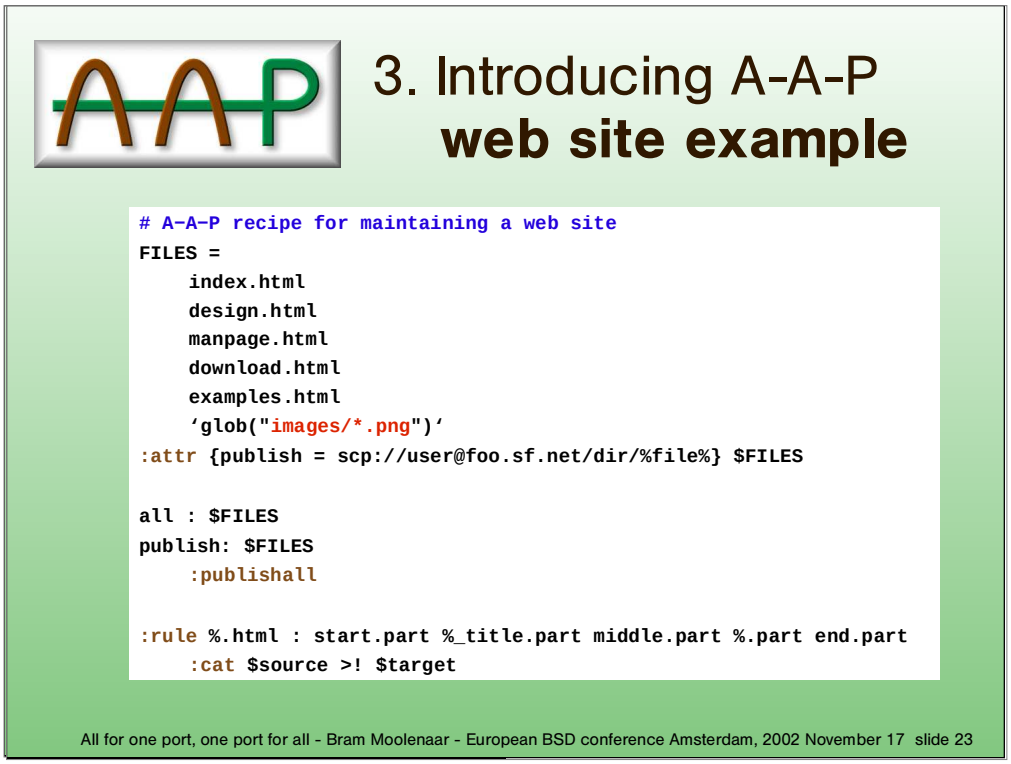
This recipe is used to upload modified files to a web site.
There are a few things here that you might find interesting

The list of file names that is to be uploaded is given one name per line. No backslashes are needed, the indent shows where the list ends. Wildcards are expanded with the Python glob() function. A Python expression is given in backticks, similar to how this is done in a shell.

NOTE: the glob() command is in backticks, but this font has a strange glyph for backticks.

The ":attr" or ":attribute" command attaches an attribute to each item in $FILES. Attributes is a generic mechanism to specify what is to be done with a file.

In the example the "publish" attribute specifies where files are to be uploaded to.



This adds generating the HTML files from parts.

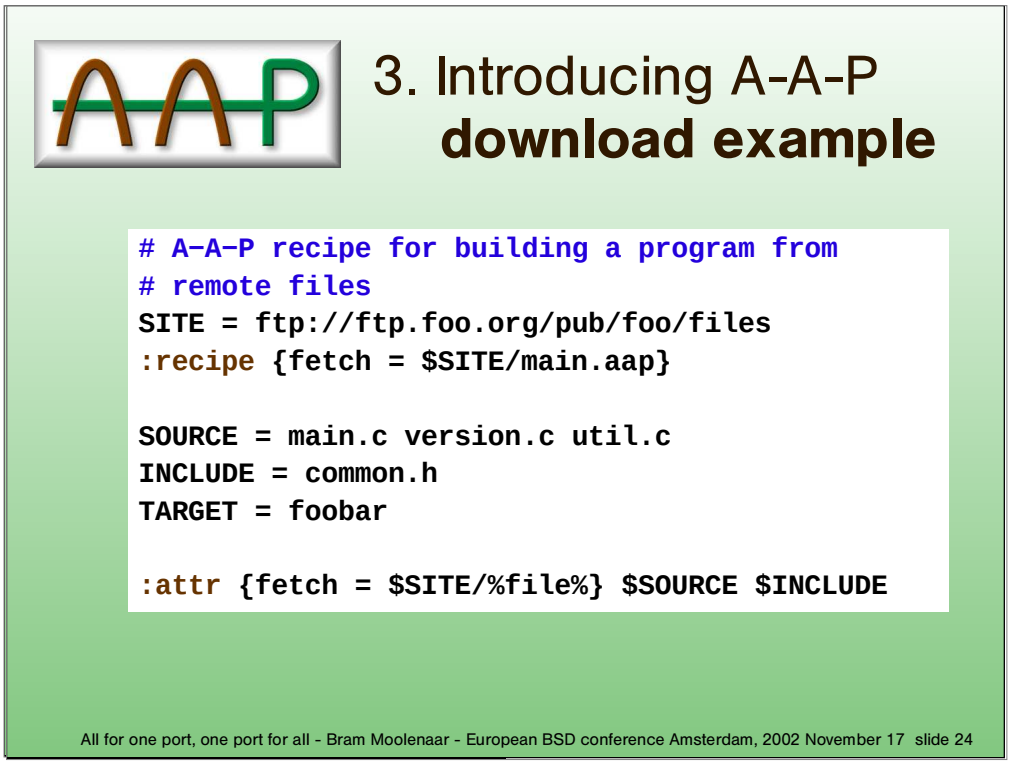
The "all" target is used to build the HTML files from five parts. The ":rule" command at the end specifies how this is done with the ":cat" command. This works just like the UNIX "cat" command and also works on non−UNIX systems.

The ":rule" command specifies a generic way to make a .html file from .part file.

$source stands for the list of sources used in the dependency.

Note the use of the percent sign, you cannot do this in a Makefile.

The ":publishall" command uploads all modified files to the URL specified with the "publish" attribute.

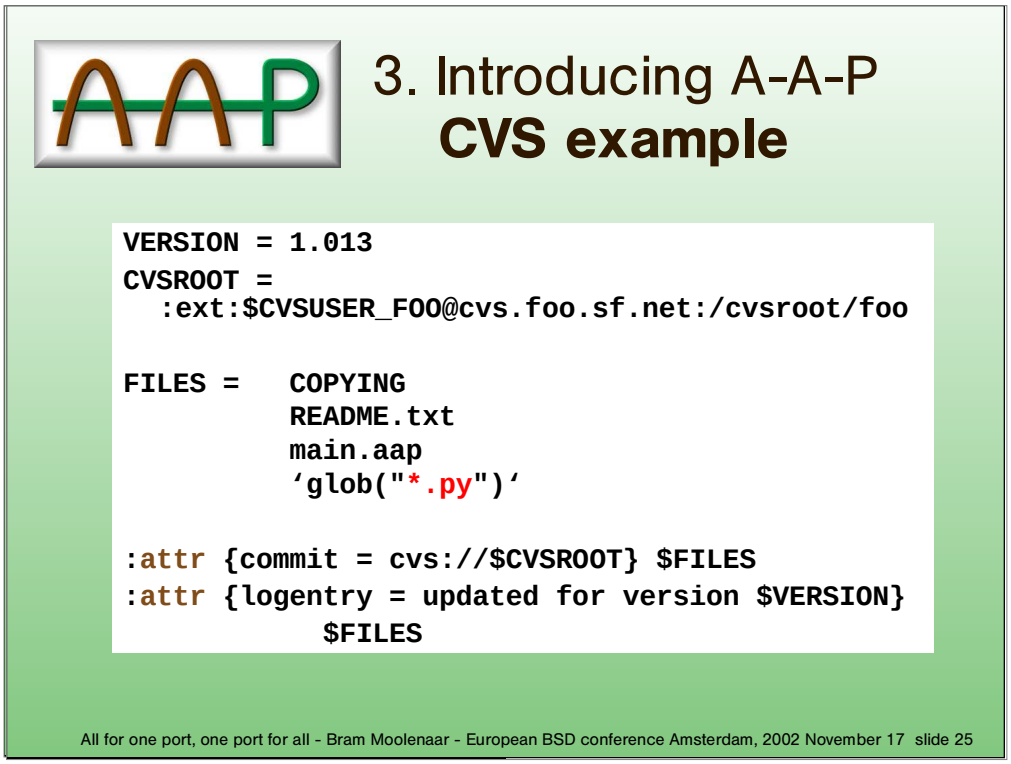


This recipe is used to download individual files for an application.

The ":recipe" command is used to obtain a new version of the recipe itself. This makes it possible to modify the recipe and let the user download it without typing the URL.

The build commands are automatically generated from SOURCE and TARGET.

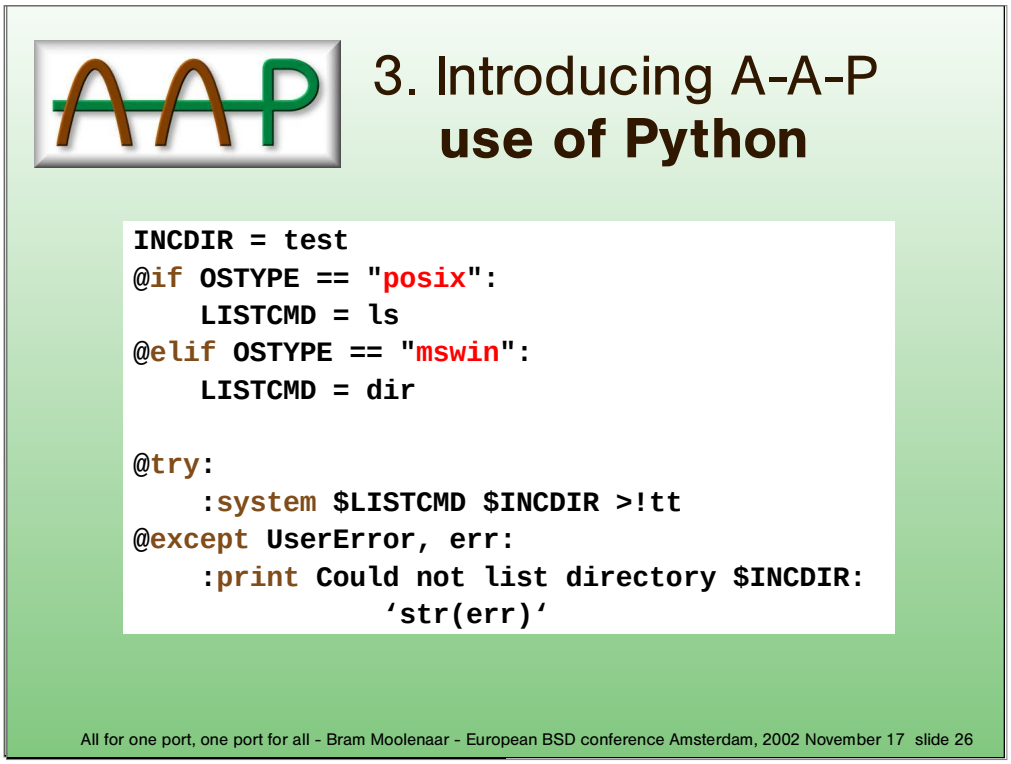
Since the URL is given as an attribute to the files, A−A−P will know where to download them from.



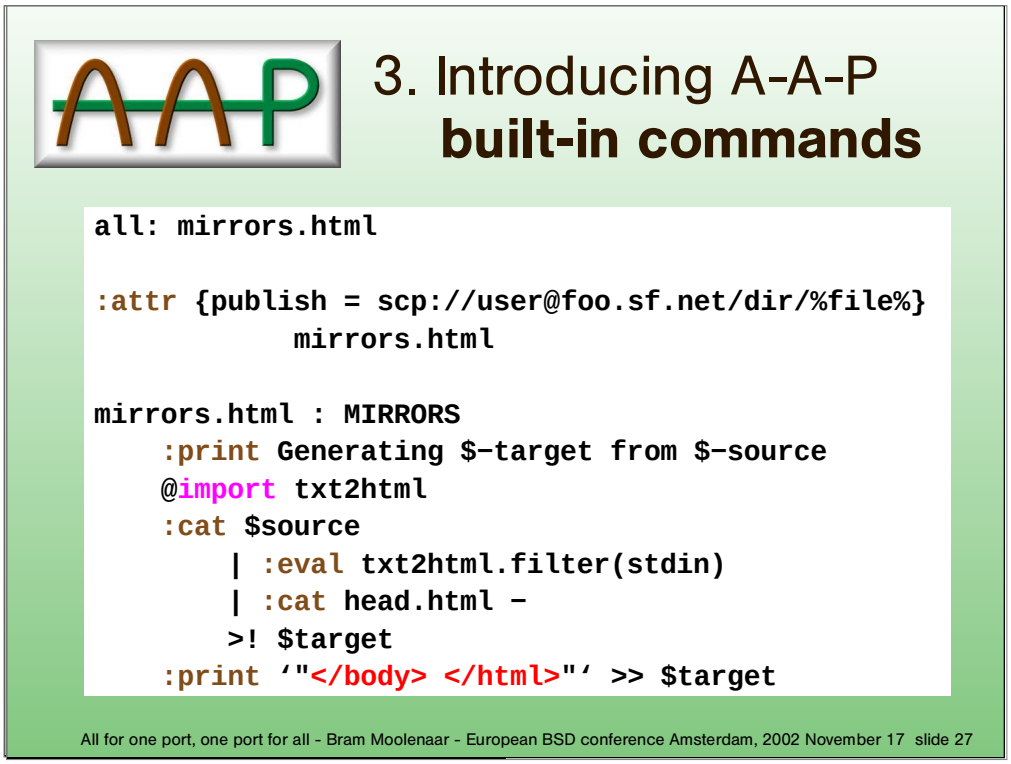
I will not try explain everything here.

Using CVS is like another way to upload and download files. The CVSROOT is specified like a URL. This is especially useful for people who don't know CVS very well.

CVS can be used for downloading only, this is a simple way of getting the latest version of an application.



When you need to add flow control, expressions or something complicated, Python script can be used. It mixes with the other items in a recipe.



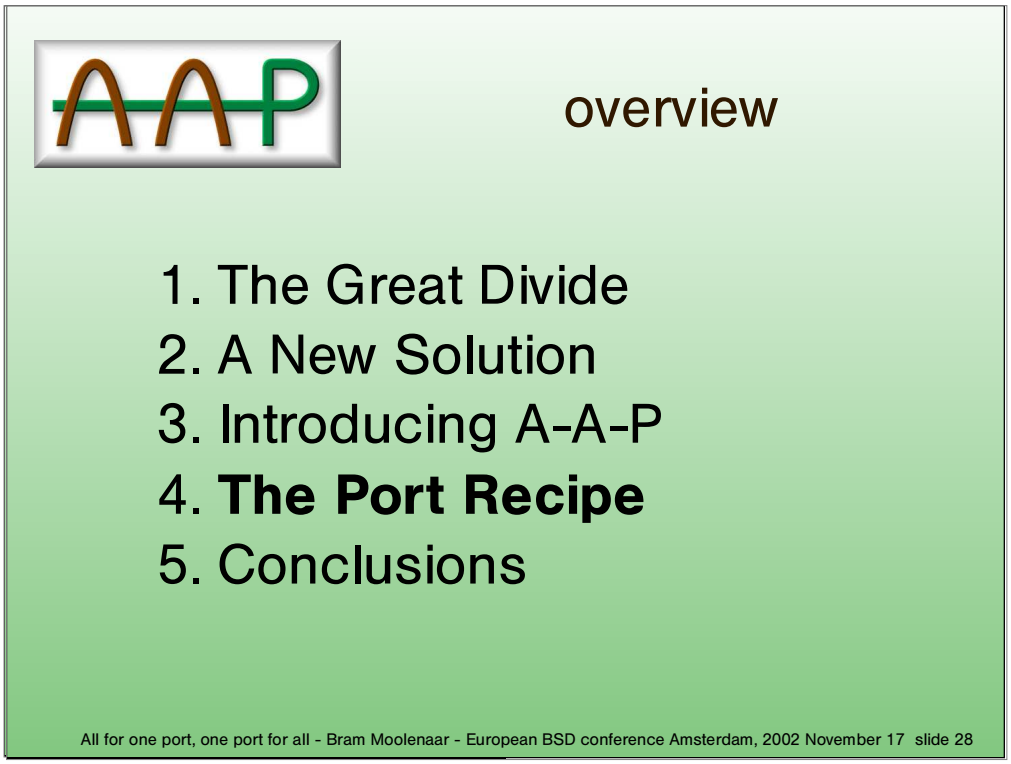
There are many built−in commands to use for common tasks. This avoids using shell commands, which are not portable.

This example shows how an HTML file is generated from a plain text file.

The "@import txt2html" command imports a Python module. The "filter()" function it defines is used in an ":eval" command.

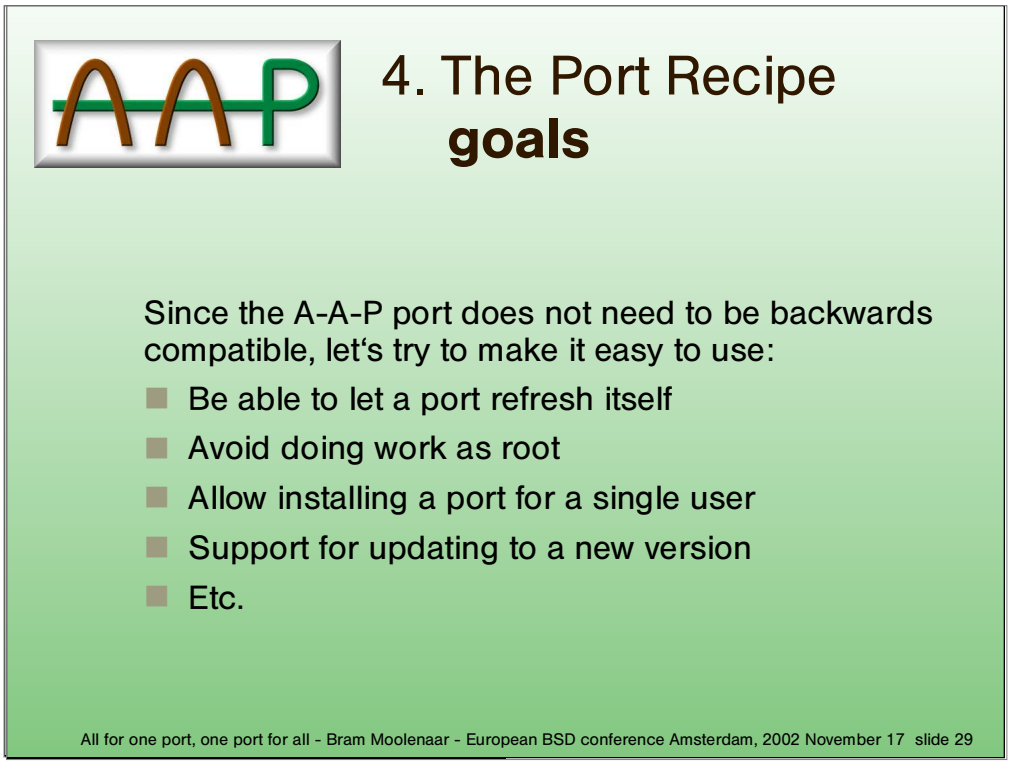
Note that a pipe is supported, much like it is in a shell.

This is just to show what is possible with an A−A−P recipe, I will skip the details here.



So far you have seen the generic items in an A−A−P recipe. It is a super−make tool.

Now lets see how the A−A−P recipe can be used for a port. This is more or less like a Makefile is used for a traditional port.



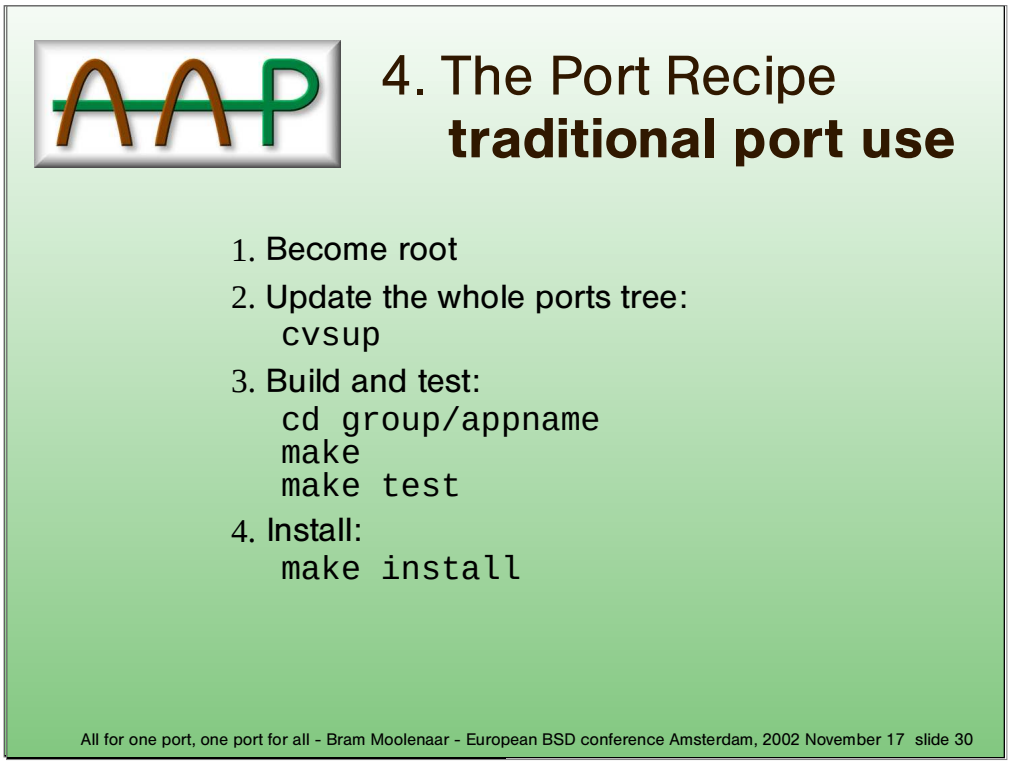
Not having the restriction to be backwards compatible gives us quite a few possabilities.

Not having to update the whole ports tree before installing one is a big advantage. For me it takes over half an hour to run cvsup, and that's with a fast internet connection.

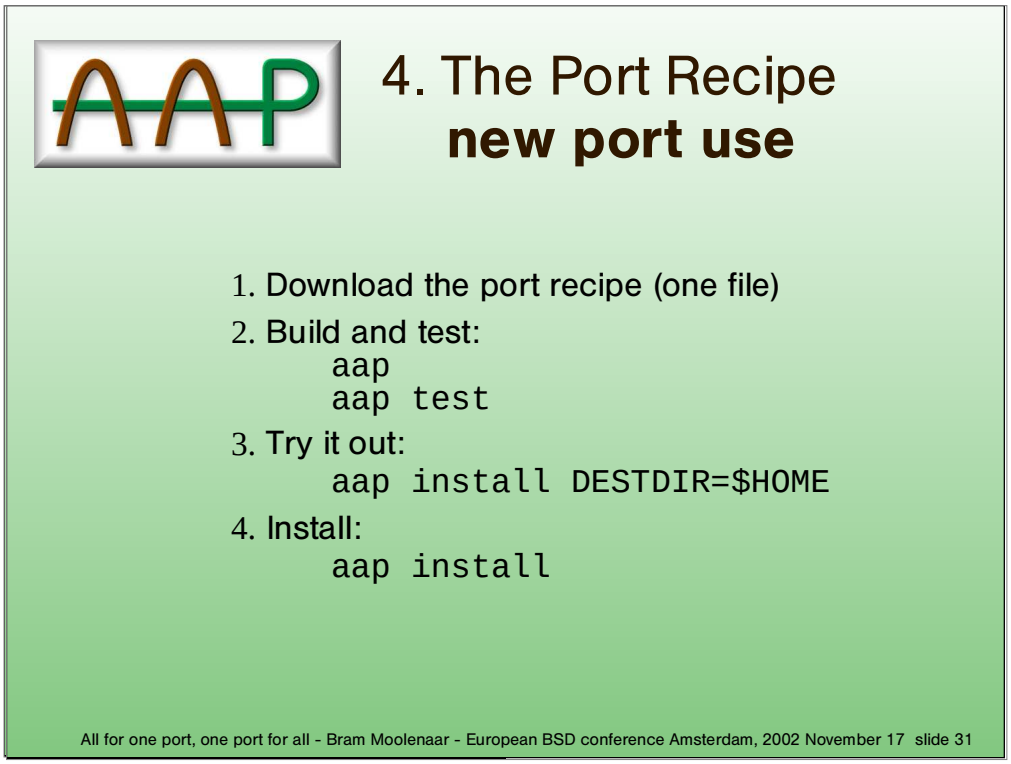
Security becomes more important every day. To avoid building a port while being root helps a bit.

Installing for a single user is required for trying it out and when the system administrator is not cooperative.

Some applications are updated very often. Being able to update a port in a simple way is very much desired. The dependencies often make it complicated though.



This is how you mostly install a traditional port.



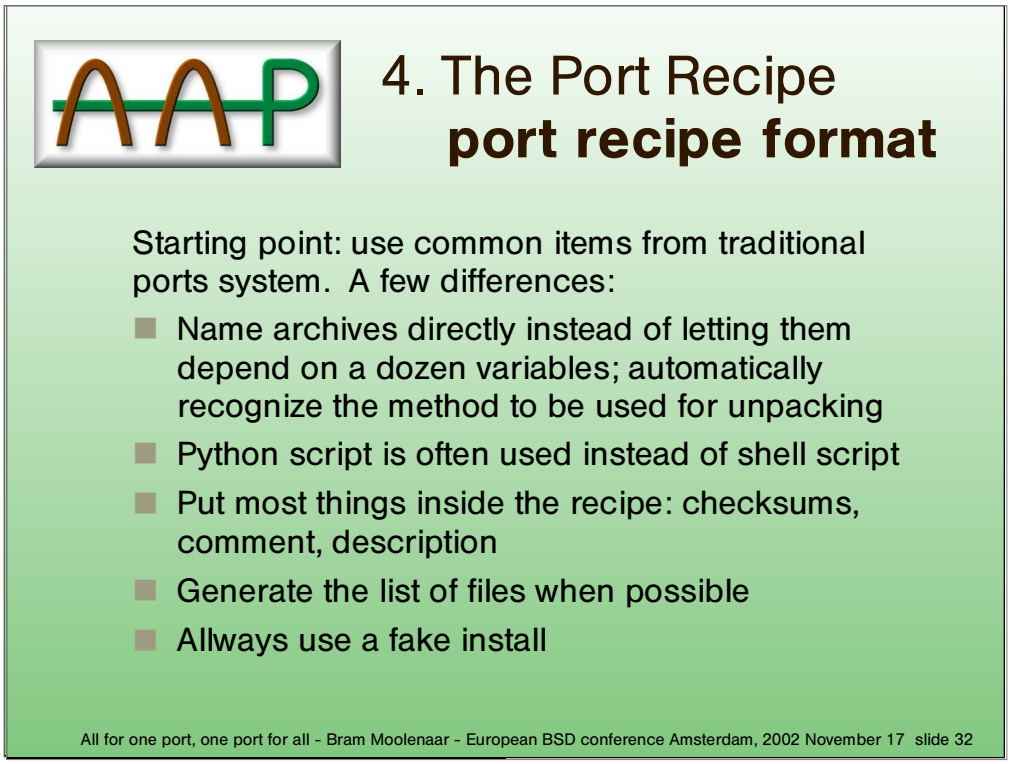
This is how you normally install an A−A−P port.

Usually the system maintainer would have his own directory with the port tree somewhere. He does not need to be root to use it.

If the port has dependencies, the port recipes for those packages may be automatically downloaded as well.

The "try out" step is very important on a multi−user system. You don’t want to break tools that are being used.

Note that there is no step to become root. In the final install step you will be prompted to enter the root password. It is only needed once, also when several dependencies are to be installed. This is done in a separate shell, so that the building isn’t done by root.



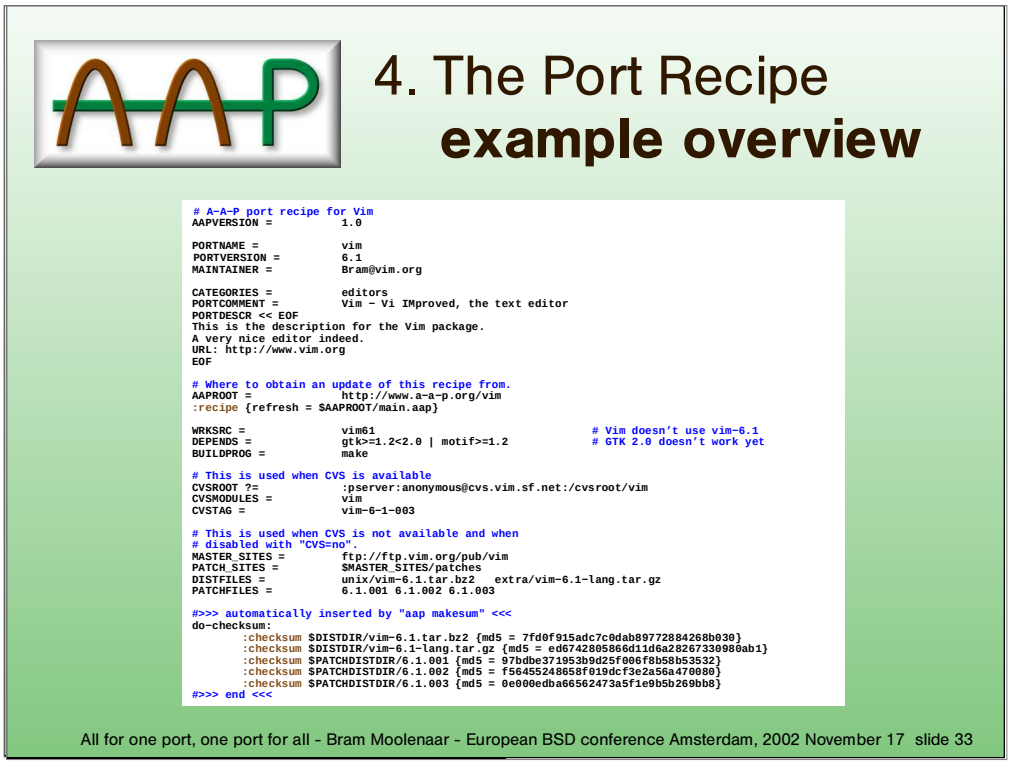
Using useful items from the traditional ports system has the advantage that it's easier to switch to the new system.

Shell script can still be used when it's needed.

The fake install is required to be able to generate a binary package without overwriting files on the system.

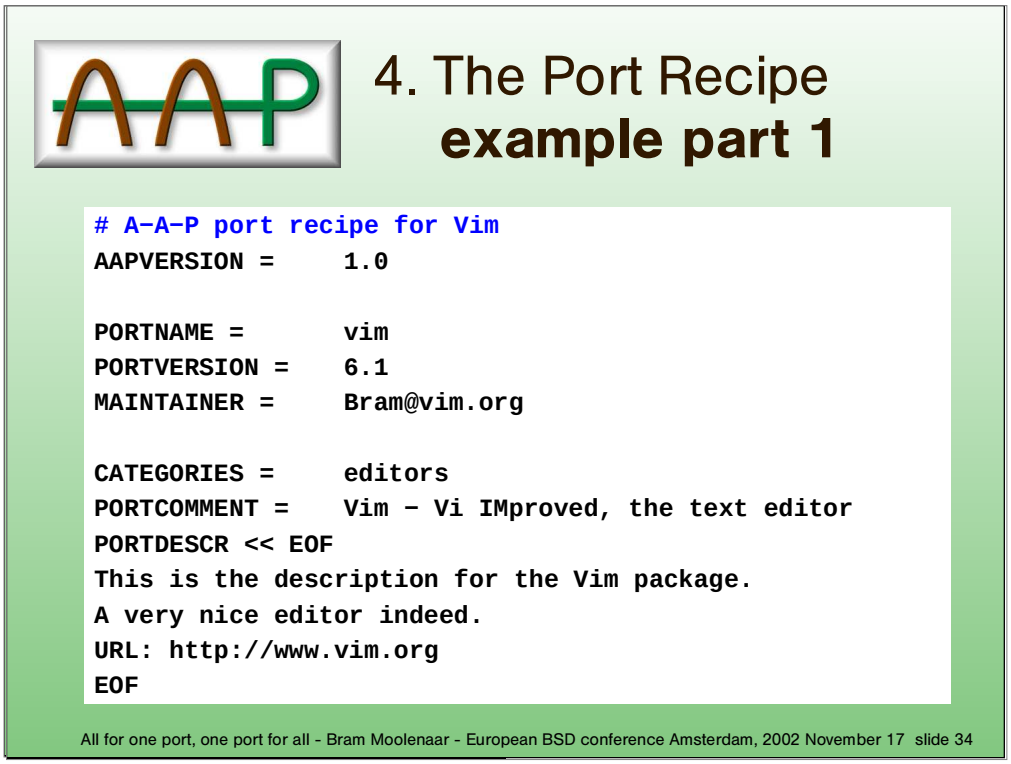
Example: Vim contains hundreds of syntax files, therefore it is vey convenient to generate the list of files automatically.

Always using a fake install might have some problems, perhaps it should be allowed to install in another way in specific situations.



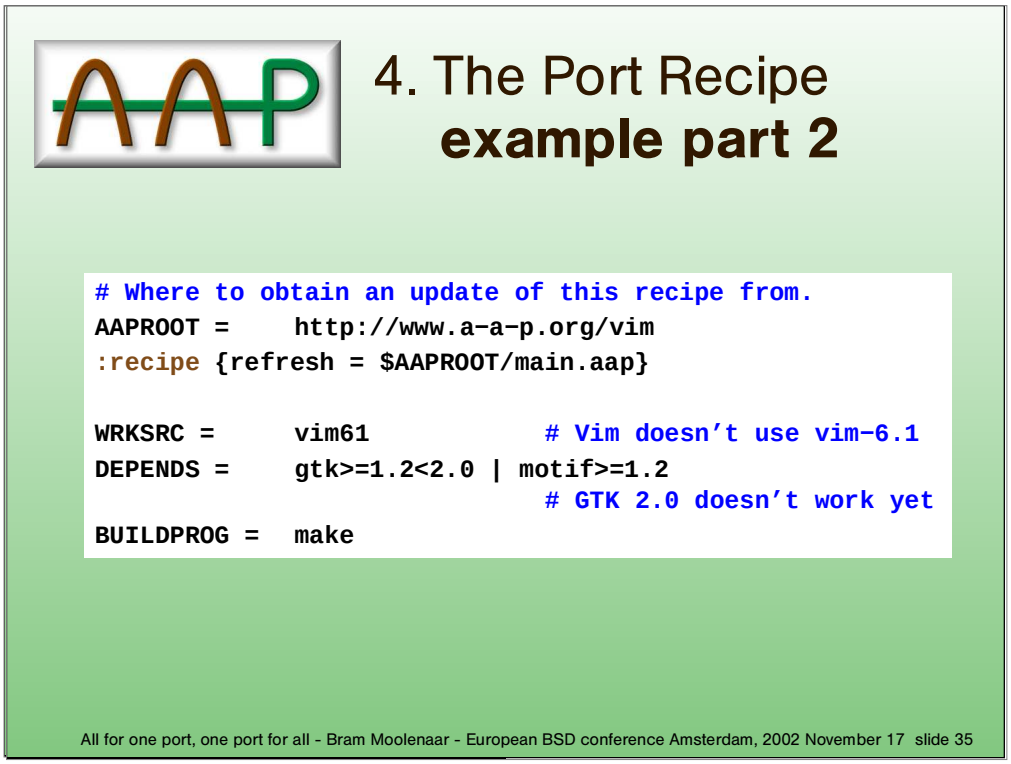
Most things in the port recipe look just like the traditional port Makefile. It is a bit longer, because of including the description and checksums.

Don't worry if you can't read this, I'll explain each part.



Most things in the port recipe look just like the traditional port Makefile.

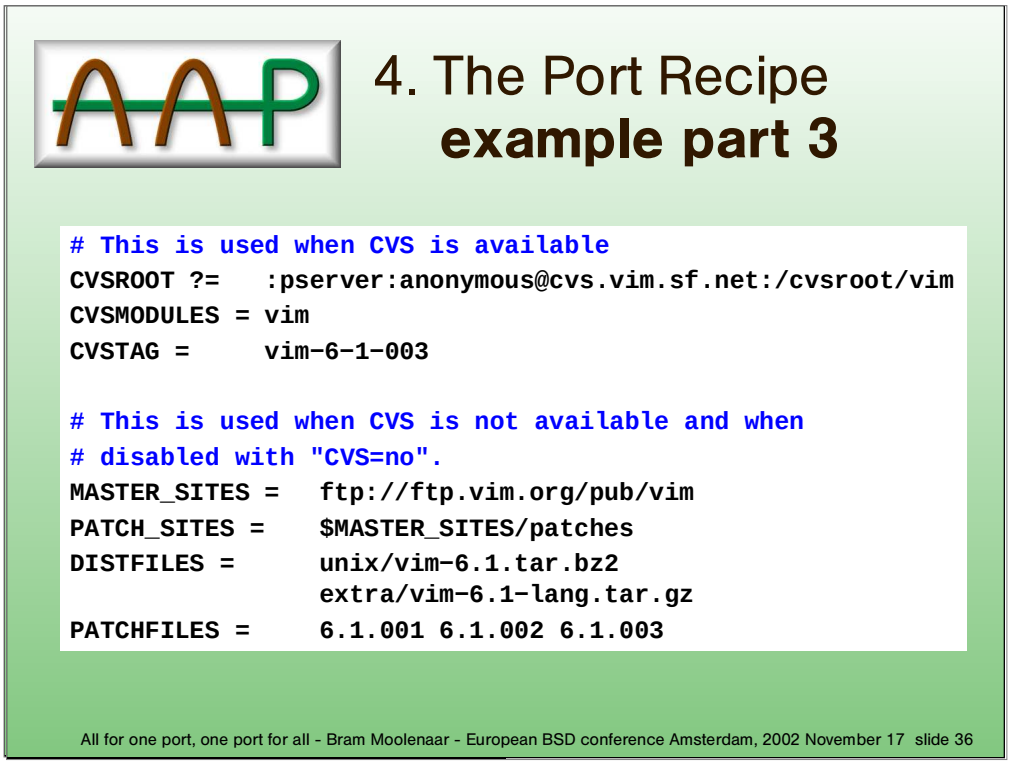
The comment and description is given right here. That's easier than using a separate file for this. It avoids the inefficiency caused by a large number of small files in the port collection.



The ":recipe" command is used to obtain a fresh copy of the recipe when desired. Thus the recipe knows where to download itself from.

The dependencies specify which items this port depends on. It can be a list of specific versions or use ranges. More about the implementation of this further on.

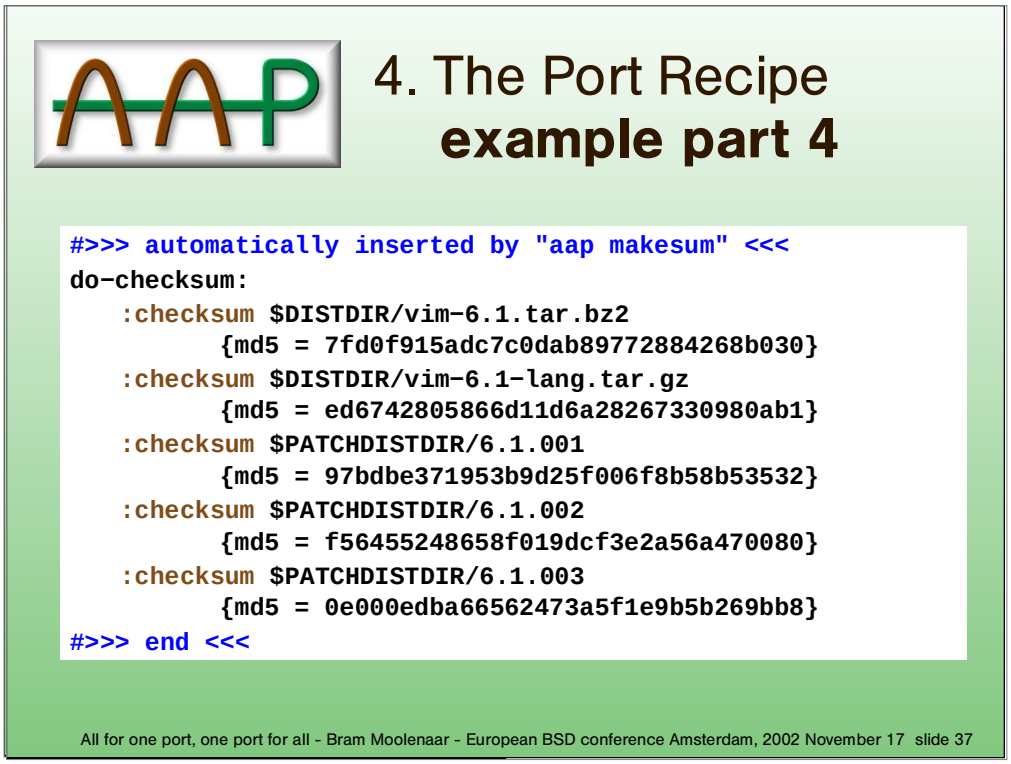
It is possible to specify any build command, including configure, without the need to overrule the default dependency.



This port can be obtained through CVS or using archives and patches.

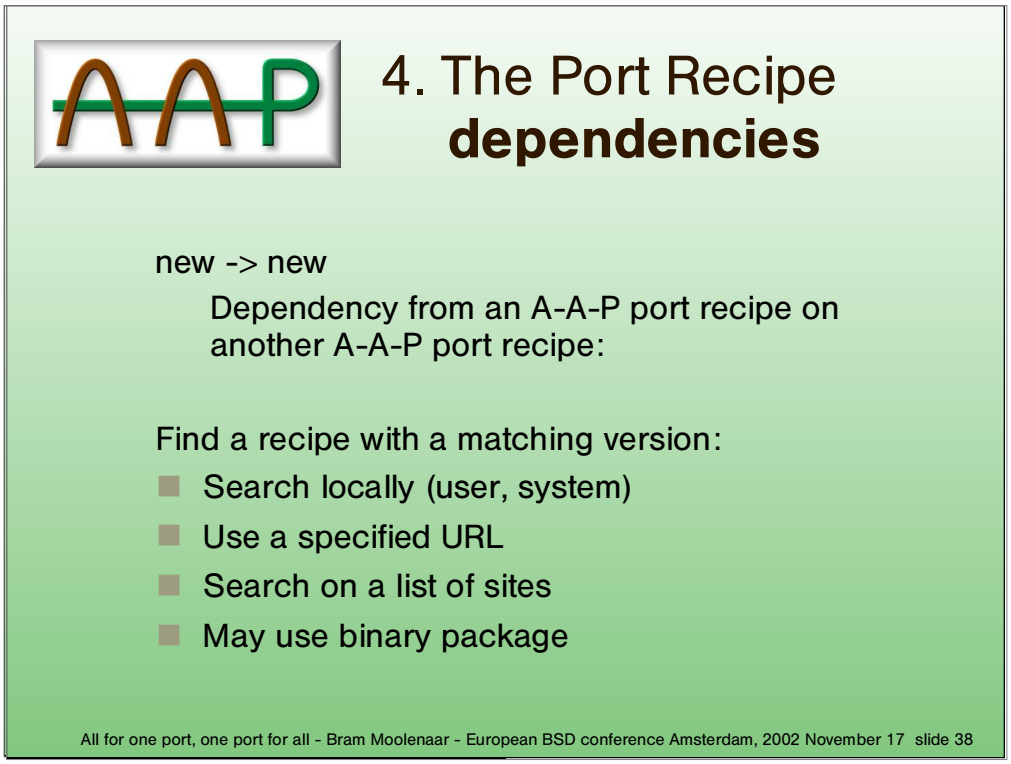
Obtaining files through CVS is possible, but you can leave it out if you don't like it. For stable ports it's probably better not to use CVS, for the latest version it may be the only way to get the files.

Note that two archives are specified, with a different extention. This is much simpler than using various variables to form the archive name.



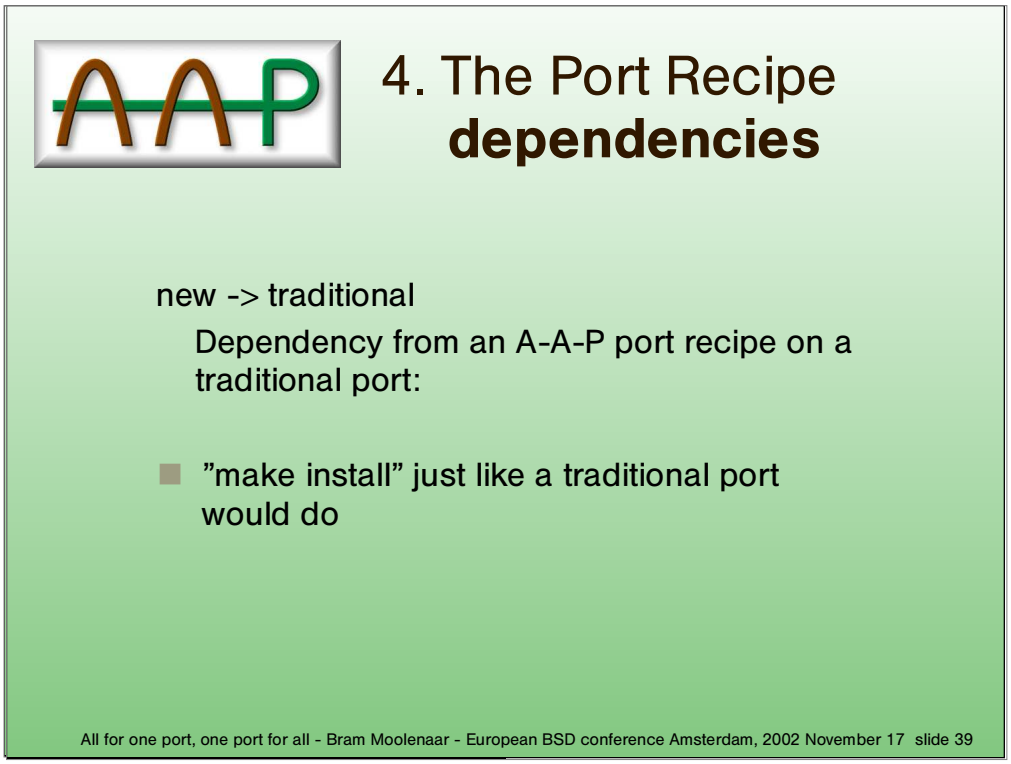
These lines are automatically added or replaced when using "aap makesum".

It’s nice to have it in one file, so that there can’t be any mistake in using the wrong checksum file. It can be in a separate recipe if needed.

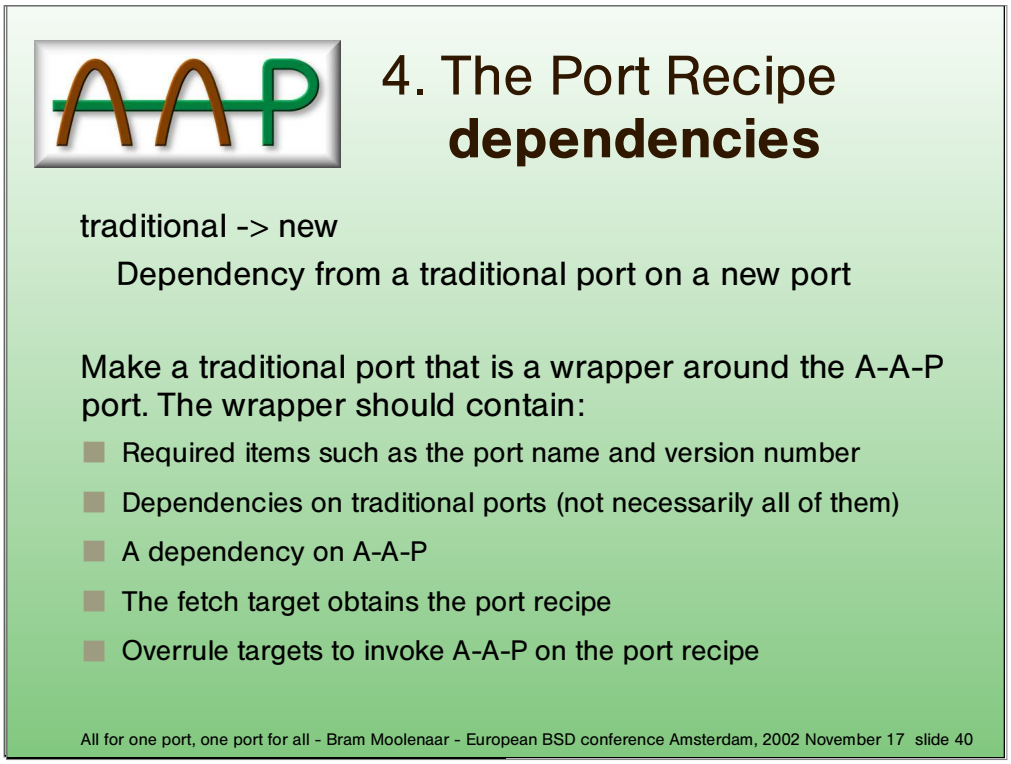


There are three kinds of dependencies for a new port.

Using a binary package for a dependency greatly speeds up the install. It can only be done when the dependencies do not require tuning the way the port is build (which is mostly true).



This is quite simple, but it does require becoming root.

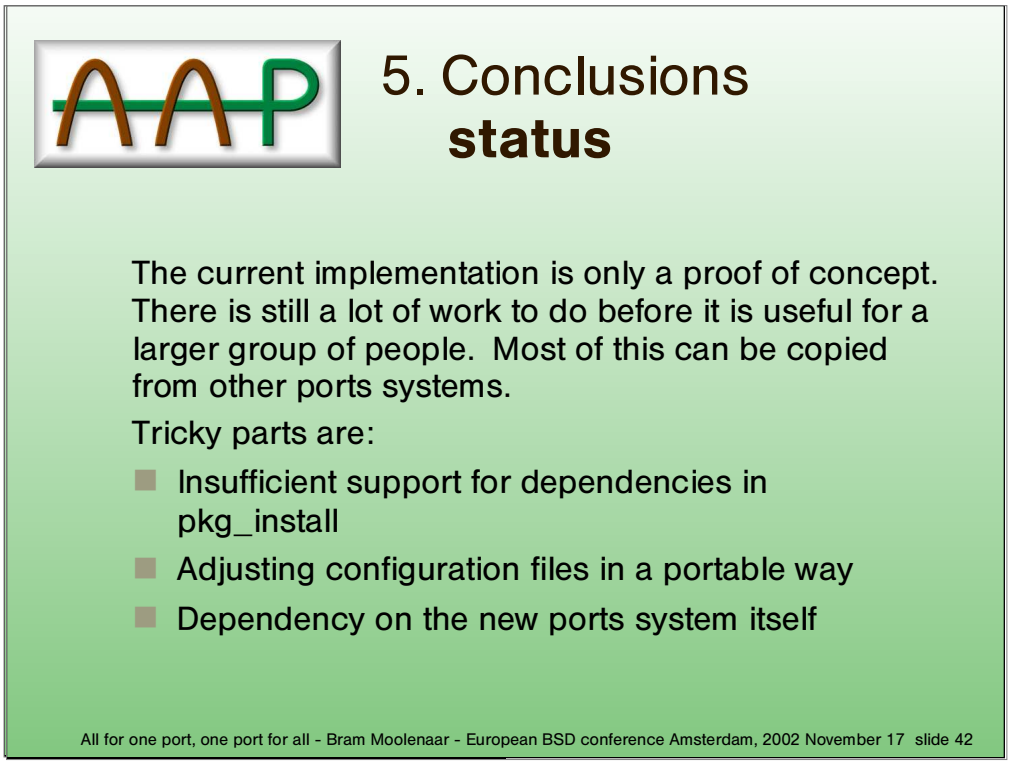


This is a bit clumsy, perhaps it would be possible to generate the wrapper port.

Alternatively, the traditional ports system could be expanded with functionality to handle a dependency on a new port itself.

overview

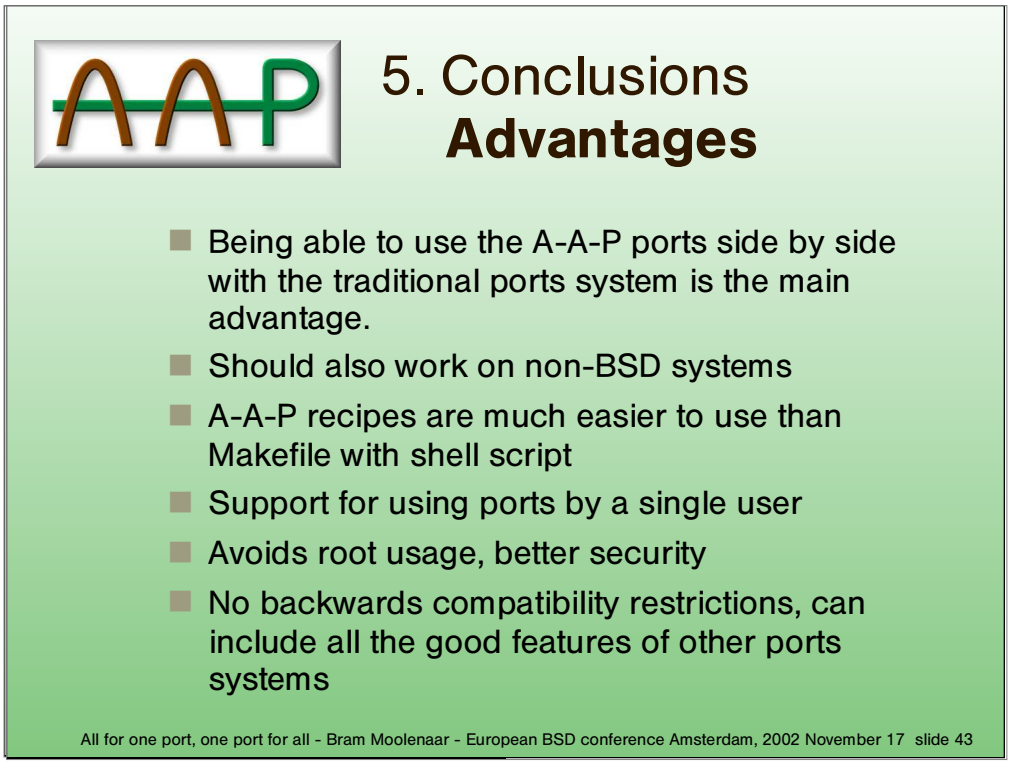
1. The Great Divide
2. A New Solution
3. Introducing A-A-P
4. The Port Recipe
5. **Conclusions**



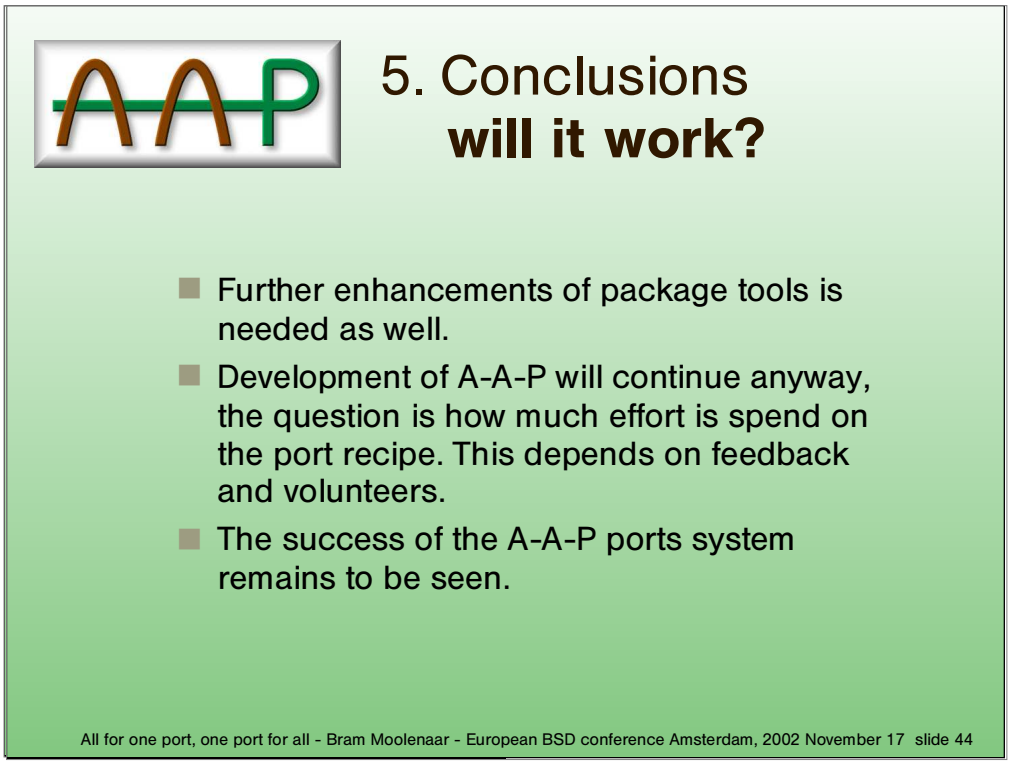
Most of the port recipe works. Main part that is missing is the dependency handling. This could be solved by adding a pre−install shell script that handles the dependencies. Not a very nice solution though.

The adjustment of configuration files is to be done in a binary package. This should probably be a shell script. An alternative is to use an A−A−P recipe and add a dependency on the A−A−P package.

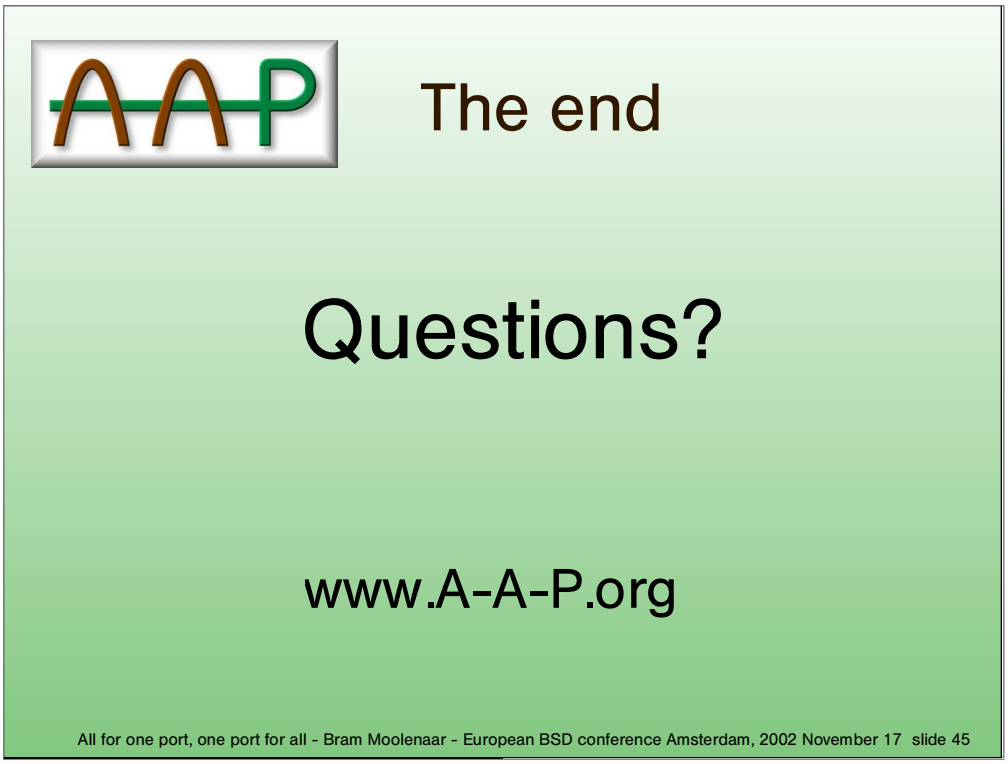
When fetching individual port recipes, some of them may depend on another version of the ports system files. In the traditional ports system this is solved by the brute force method of updating everything. For A−A−P a smarter solution is needed.



It will be possible to use the A−A−P ports system side by side with a traditional ports system. Gradually more A−A−P ports can be made.

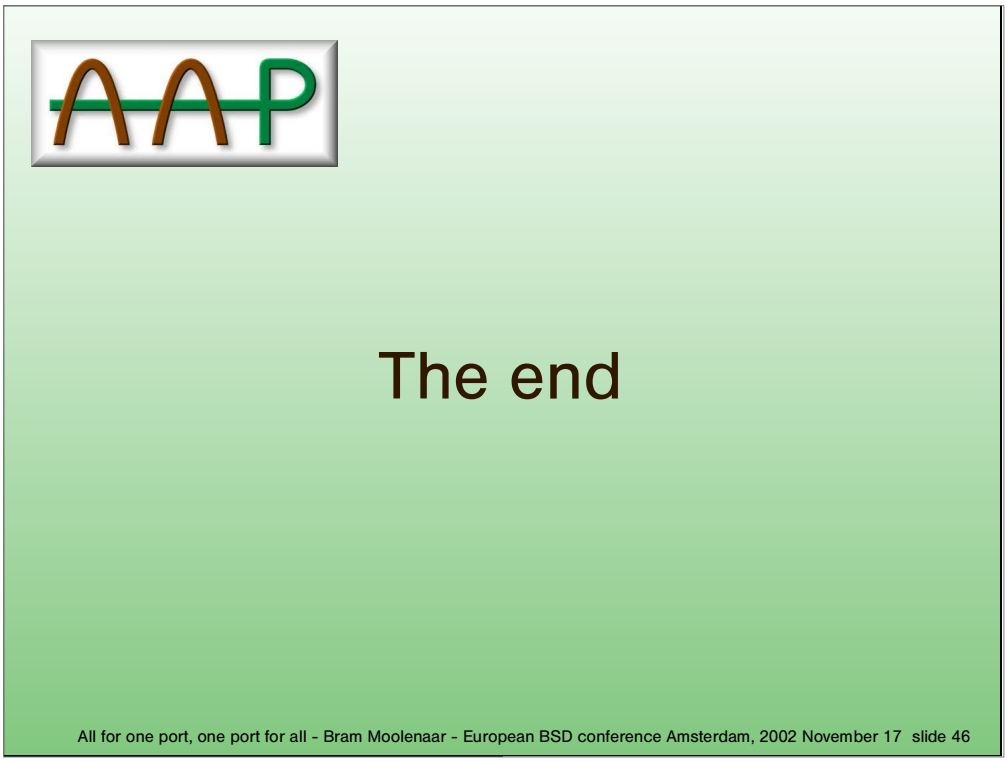


The development of A−A−P will be concentrated on the generic items. Only when there is sufficient interest for the port recipe will work be done on it. Help from others will be needed, I cannot do this all by myself.



Whether the A−A−P recipe will be successful or not, I hope this presentation has inspired you to think of ways to use one port for all systems.

I will be around for more information.


The end
All for one port, one port for all - Bram Moolenaar - European BSD conference Amsterdam, 2002 November 17 slide 46

.