Ist Distribution-Level Package-Management überflüssig? Welche Vorteile bringt NixOS?

Joachim Schiele
Blog:
blog.lastlog.de
Wiki:
lastlog.de
eMail:
js@lastlog.de
irc:
qknight@nixos#irc.freenode.org

2014

4:3 version

was erwartet man von software?

=> reproduzierbares verhalten

gilt das auch für
paketverwaltungssysteme

?

gilt das auch für
paketverwaltungssysteme

?

=> leider nein

paketverwaltungssysteme
sind entscheidend für
repoduzierbarkeit
von software

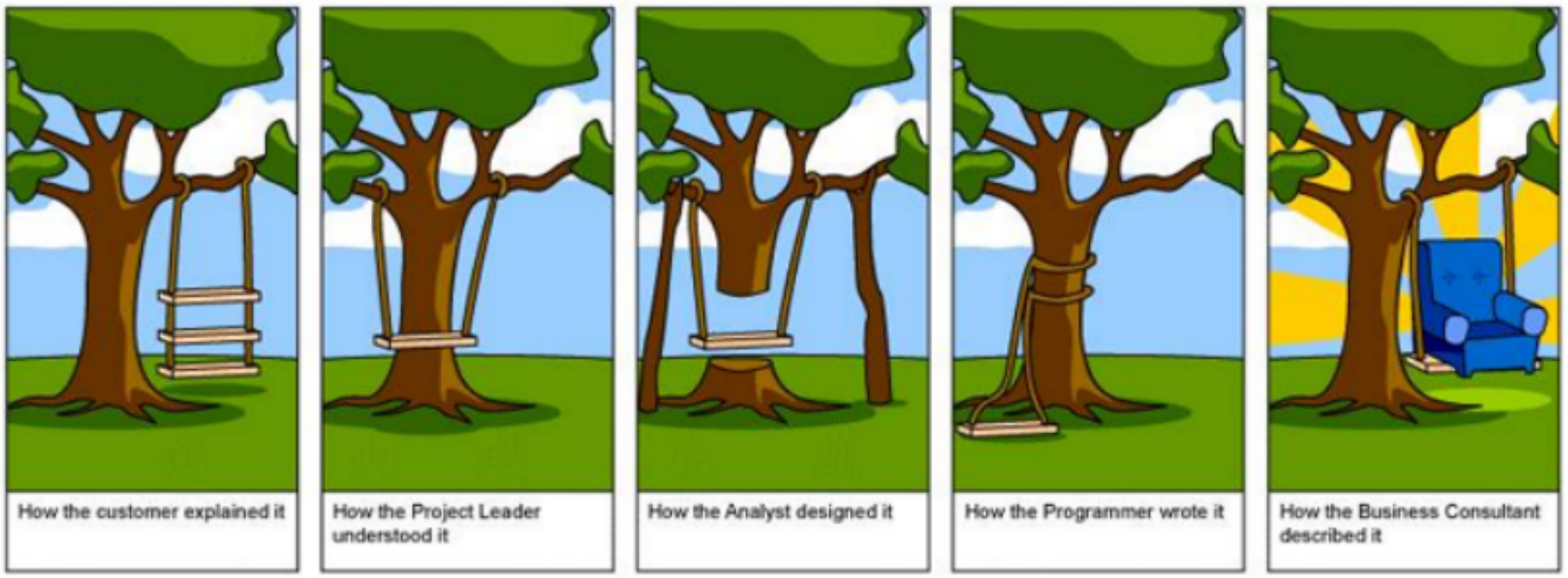
!

reproduzierbarkeit von setups

wozu soll das gut sein?

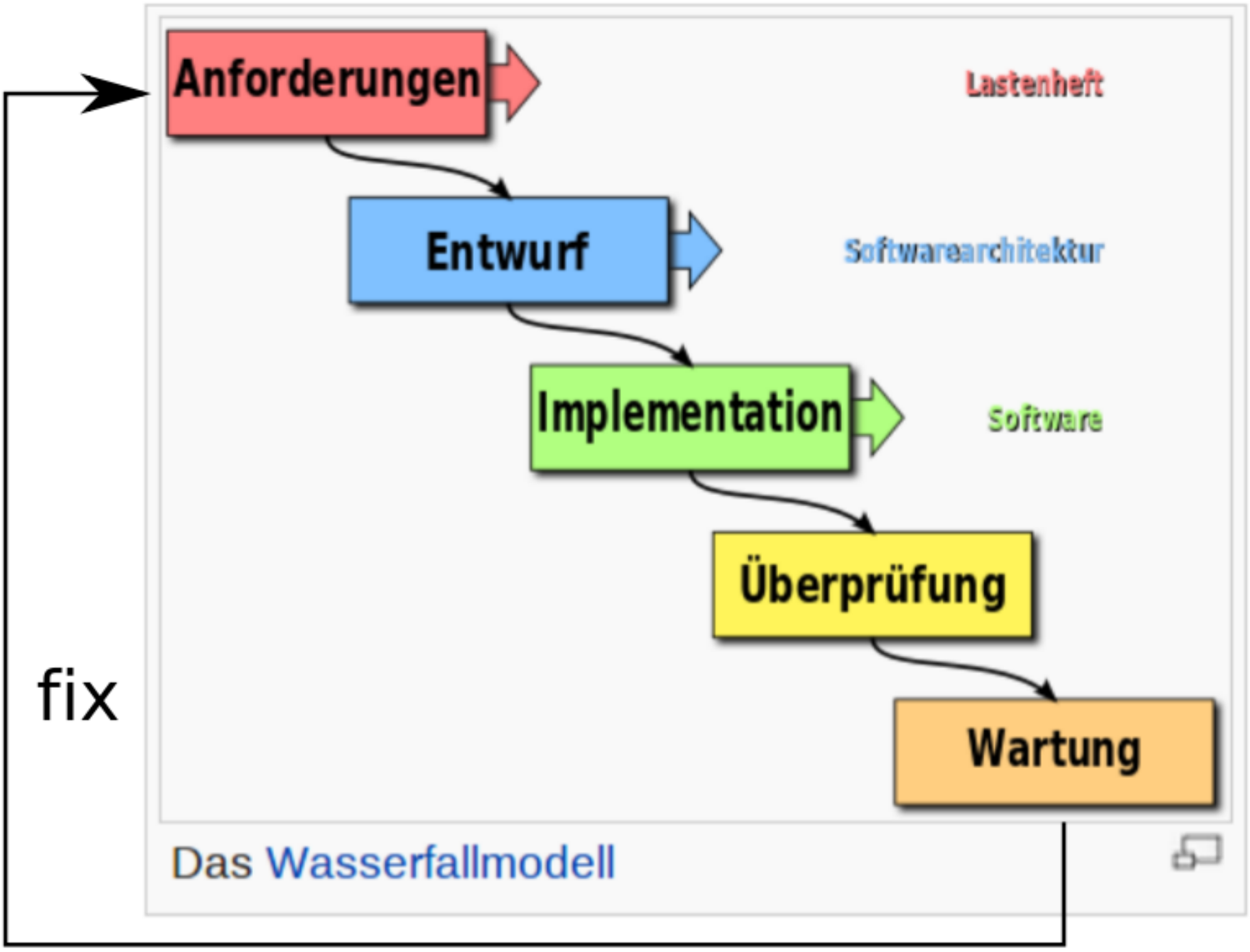
apt-get upgrade -> system tot!

z.b. für **rollbacks, CI, cooperatives**
arbeiten in der community
aka "git clone ..."



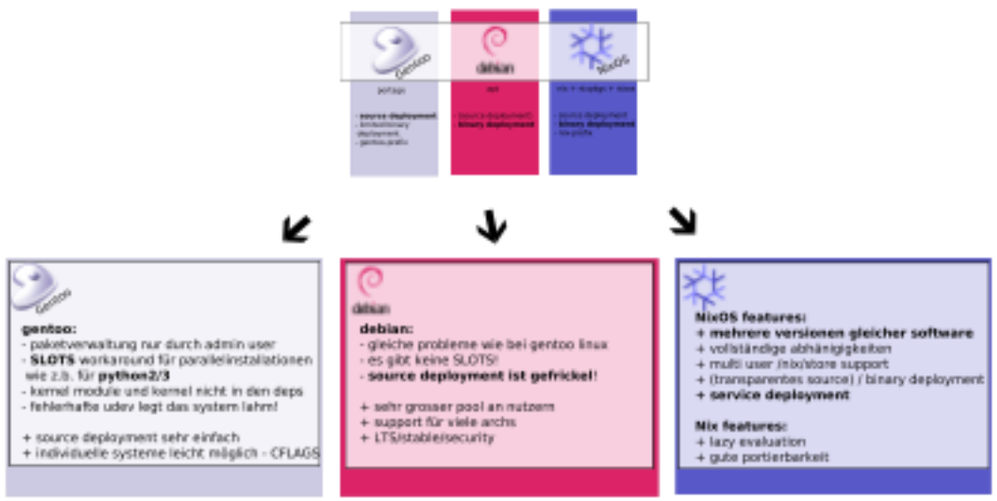
http://substance.etsmtl.ca/an-innovative-approach-to-the-development-of-an-international-software-process-lifecycle-standard-for-very-small-entities-research-paper-introduction-rpi/

software lifecycle



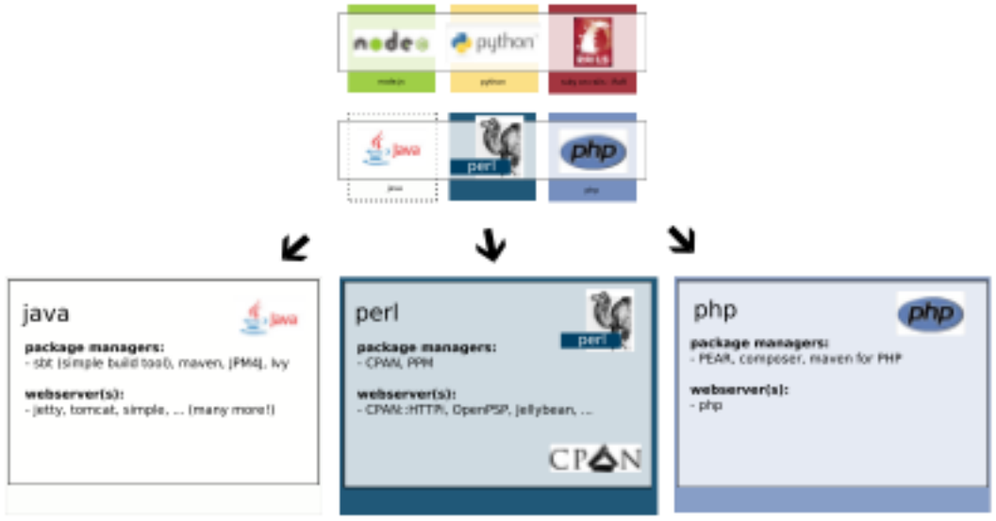
distribution-level PM
vs
language-level PM

distribution-level PM
im vergleich

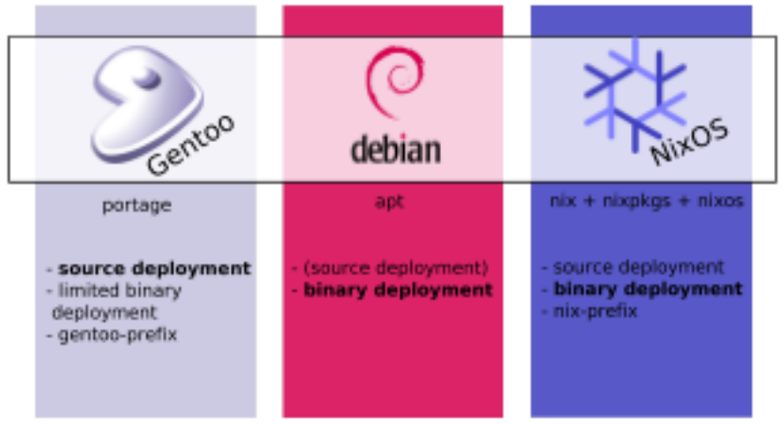


language-level PM

im vergleich



distribution-level PM im vergleich



Gentoo

gentoo:
- paketverwaltung nur durch admin user
- **SLOTS** workaround für parallelinstallationen wie z.b. für **python2/3**
- kernel module und kernel nicht in den deps
- fehlerhafte udev legt das system lahm!

+ source deployment sehr einfach
+ individuelle systeme leicht möglich - CFLAGS

debian

debian:
- gleiche probleme wie bei gentoo linux
- es gibt keine SLOTS!
- **source deployment ist gefrickel**!

+ sehr grosser pool an nutzern
+ support für viele archs
+ LTS/stable/security

NixOS features:
+ mehrere versionen gleicher software
+ vollständige abhänigigkeiten
+ multi user /nix/store support
+ (transparentes source) / binary deployment
+ service deployment

Nix features:
+ lazy evaluation
+ gute portierbarkeit

portage

- **source deployment**
- limited binary deployment
- gentoo-prefix

apt

- (source deployment)
- **binary deployment**

nix + nixpkgs + nixos

- source deployment
- **binary deployment**
- nix-prefix

gentoo:

- paketverwaltung nur durch admin user
- **SLOTS** workaround für parallelinstallationen wie z.b. für **python2/3**
- kernel module und kernel nicht in den deps
- fehlerhafte udev legt das system lahm!

+ source deployment sehr einfach
+ individuelle systeme leicht möglich - CFLAGS

debian

debian:

- gleiche probleme wie bei gentoo linux
- es gibt keine SLOTS!
- **source deployment ist gefrickel**!

+ sehr grosser pool an nutzern
+ support für viele archs
+ LTS/stable/security

NixOS features:

+ mehrere versionen gleicher software

+ vollständige abhänigigkeiten

+ multi user /nix/store support

+ (transparentes source) / binary deployment

+ service deployment

Nix features:

+ lazy evaluation

+ gute portierbarkeit

node.js

package managers:
- npm,

https://www.npmjs.org/

webserver(s):
node

python

package managers:
- PyPm, Pipy, mpgr, easy_install, ...

webserver(s) written in python:
- Chaussette, CherryPy, Gunicorn, Rocket, Spawning, Tornado, Twisted, Waitress
https://wiki.python.org/moin/WebServers

ruby (on rails)

package managers:
- RubyGems

webserver(s):
- WEBrick, mongrel,
- Phusion Passenger

language-level PM im vergleich

java

package managers:
- sbt (simple build tool), maven, JPM4J, Ivy

webserver(s):
- jetty, tomcat, simple, ... (many more!)

perl

package managers:
- CPAN, PPM

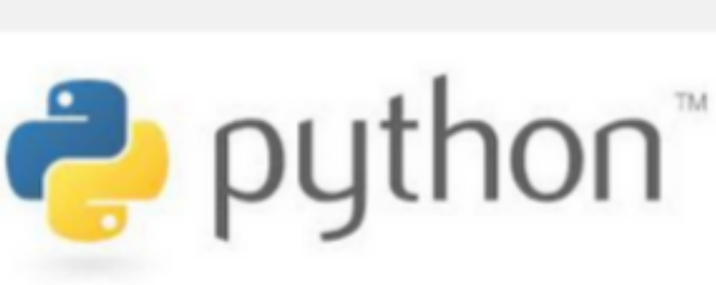
webserver(s):
- CPAN::HTTPi, OpenPSP, Jellybean, ...

php

package managers:
- PEAR, composer, maven for PHP

webserver(s):
- php

node.js



python

ruby on rails - RoR

java

perl

php

node.js

package managers:
- npm,

https://www.npmjs.org/

webserver(s):
node

python

package managers:
- PyPm, Pipy, mpgr, easy_install, ...

webserver(s) written in python:
- Chaussette, CherryPy, Gunicorn, Rocket, Spawning, Tornado, Twisted, Waitress
https://wiki.python.org/moin/WebServers

ruby (on rails)

package managers:
- RubyGems

webserver(s):
- WEBrick, mongrel,
- Phusion Passenger

java

package managers:
- sbt (simple build tool), maven, JPM4J, Ivy

webserver(s):
- jetty, tomcat, simple, ... (many more!)

perl

package managers:
- CPAN, PPM

webserver(s):
- CPAN::HTTPi, OpenPSP, Jellybean, ...

php

package managers:

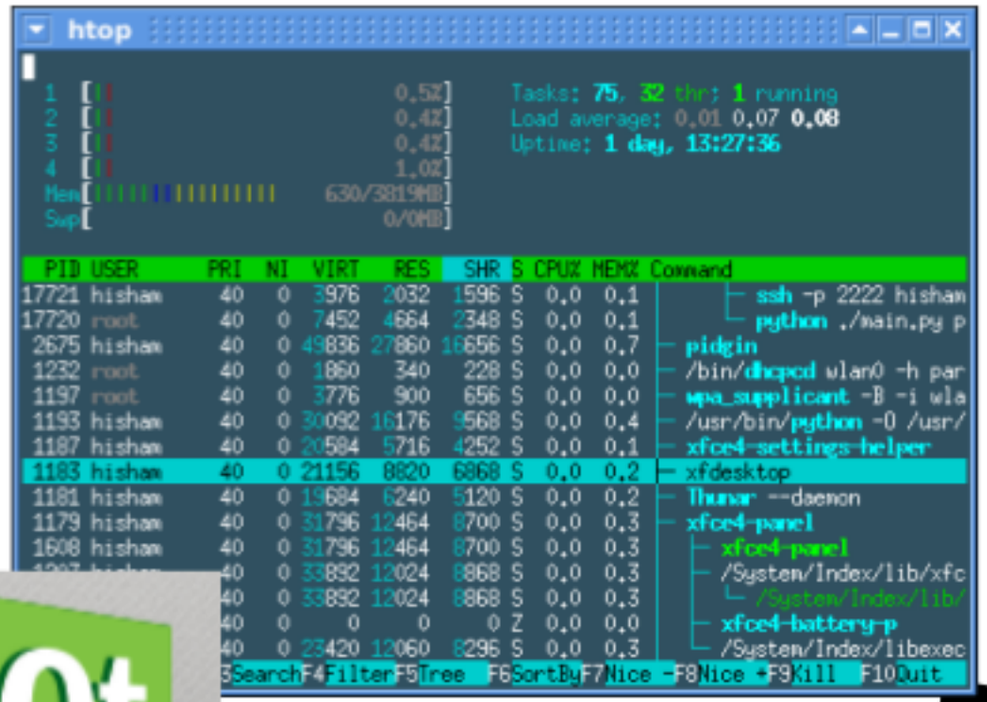
- PEAR, composer, maven for PHP

webserver(s):

- php

systemprogrammierung, scripting

- tools
- GUI



webservices
benötigt **webserver** wie:
- apache/lighttpd/nginx

oder

language spezifischen webserver

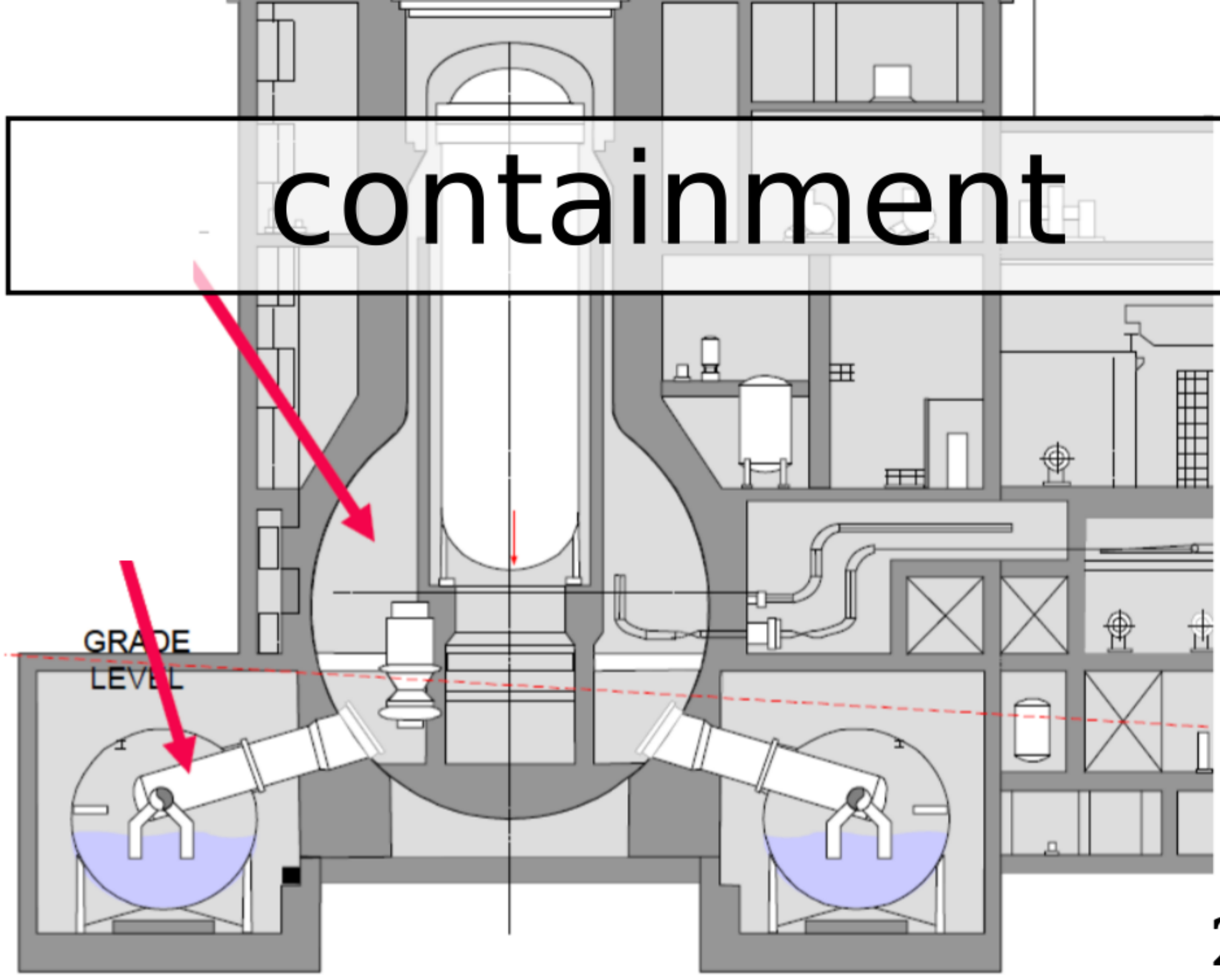
PM-usage

diskussion PMs

probleme:
- DL(distribution-level)-PM **reden nicht mit** LL(language-level)-PM
- admins bauen sich eigene deployment scripte bzw. wiederholen das deployment von hand
- sicherheitsprobleme: **bundled software**
- deployment issues: **dependencies**

... updates

containment



containerization

lösungsansätze

LXC - Linux Containers

virtualisierung

lösungsansätze

configuration management

SALTSTACK

Puppet

Disnix

NixOps

4 typen von paketverwaltungen

4

- installierte software gut erweiterbar!
- updates/upgrades geplant
- **stateless store!**
- deklarative konfiguration der services!
  d.h.: konfiguration wird on-the-fly generiert!
- in NixOS wird das system 'quasi'
  von auserhalb gebaut

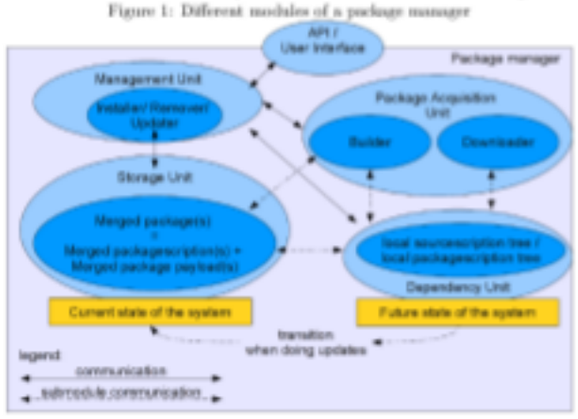
3

- installierte software gut erweiterbar!
- updates/upgrades geplant
- **stateful store!**

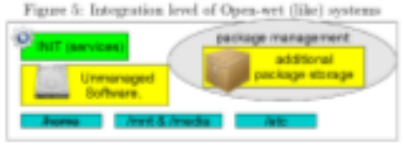
1

- installierte software nicht erweiterbar
- nur konfigurierbar
- updates ersetzen komplette systemimage

aufbau einer paketverwaltung



2



- installierte software begrenzt erweiterbar
- updates ersetzen komplette systemimage

aufbau einer paketverwaltung

Figure 1: Different modules of a package manager

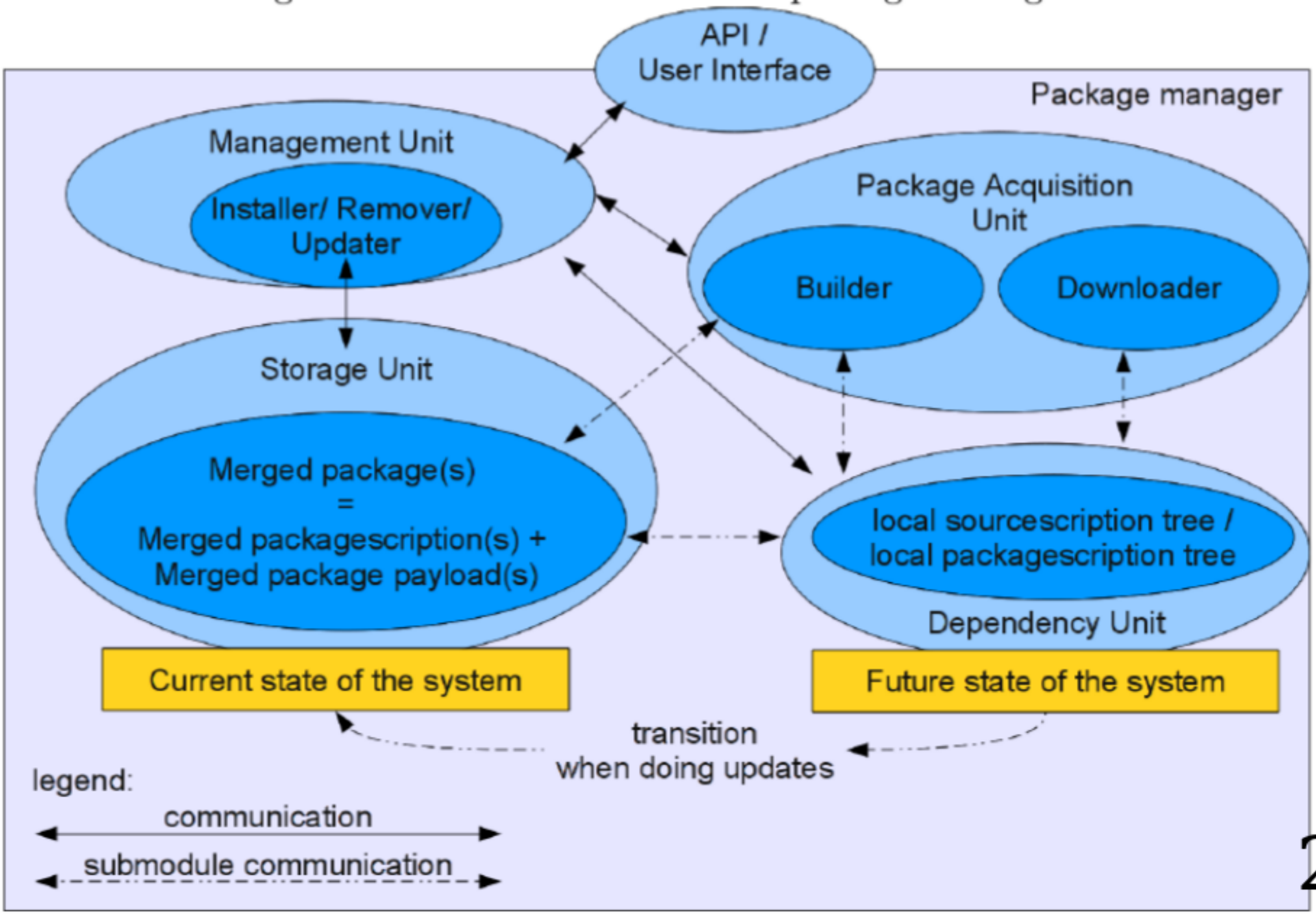
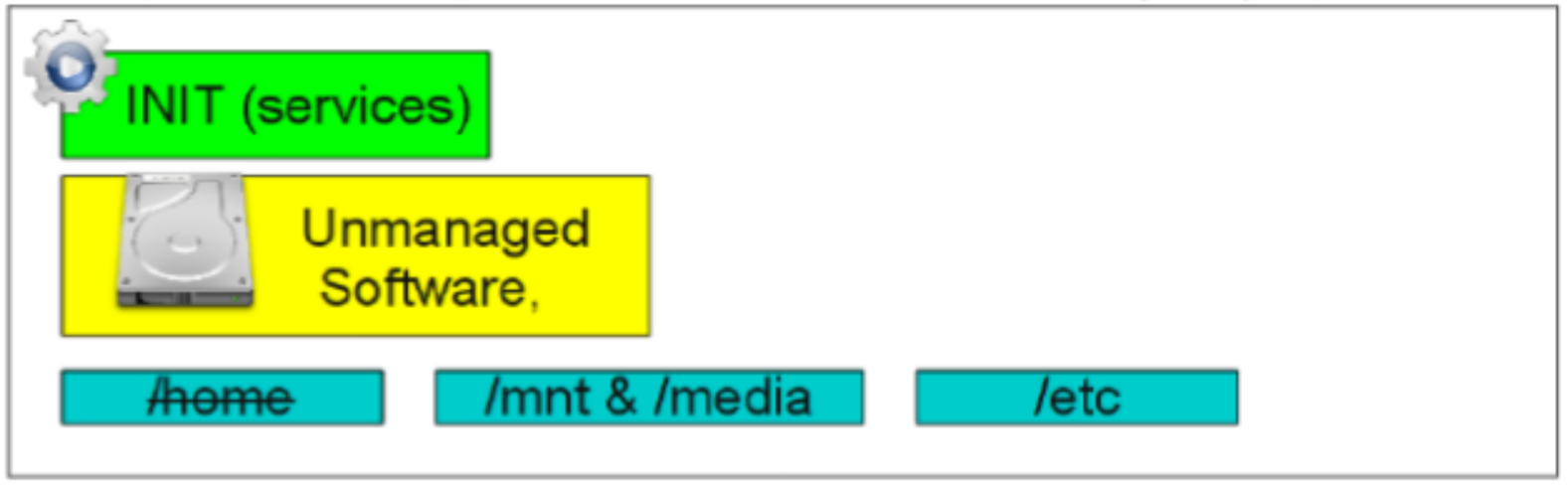
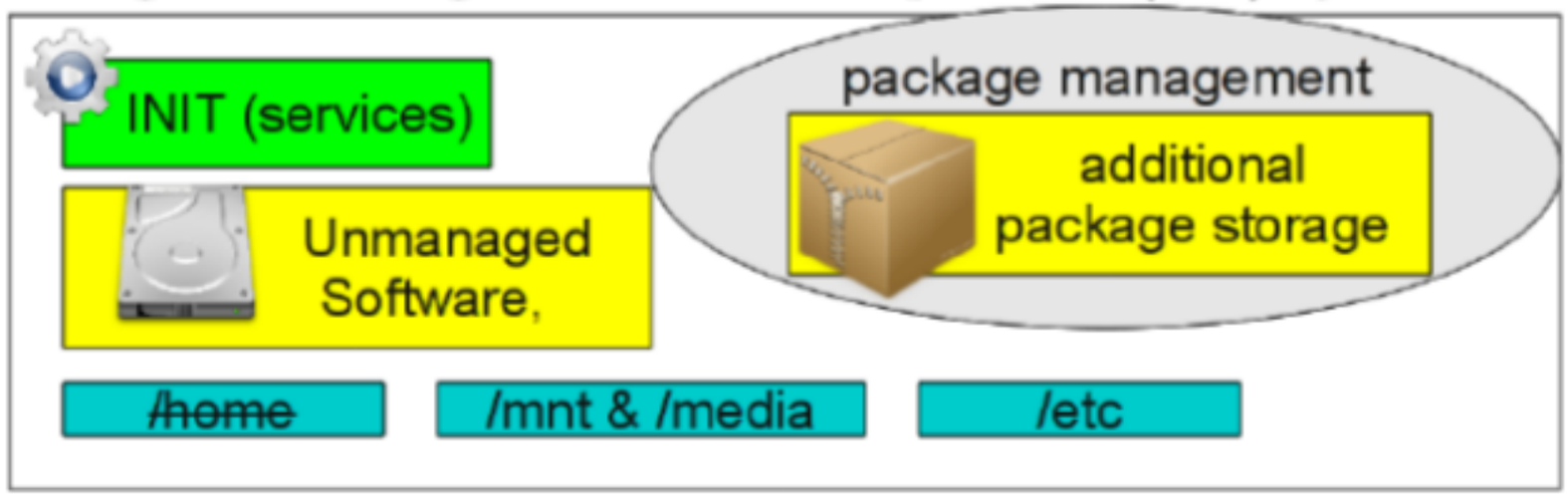


Figure 4: Integration level of Fritz!Box (like) systems



- installierte software nicht erweiterbar
- nur konfigurierbar
- updates ersetzen komplette systemimage

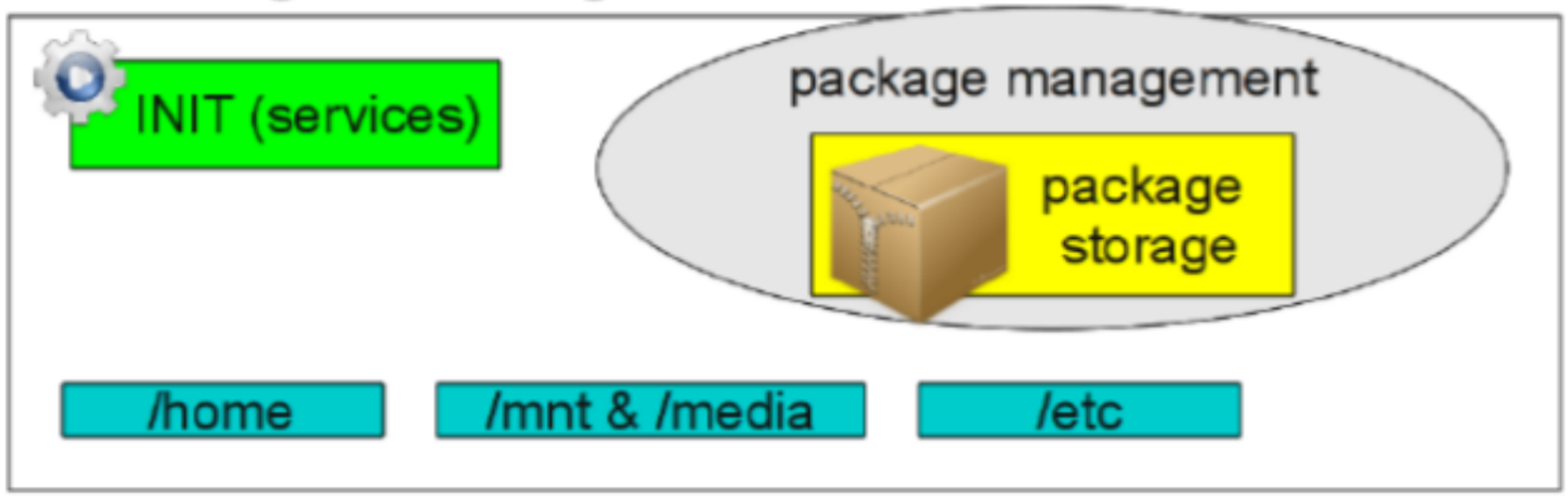
Figure 5: Integration level of Open-wrt (like) systems



2

- installierte software begrenzt erweiterbar
- updates ersetzen komplette systemimage

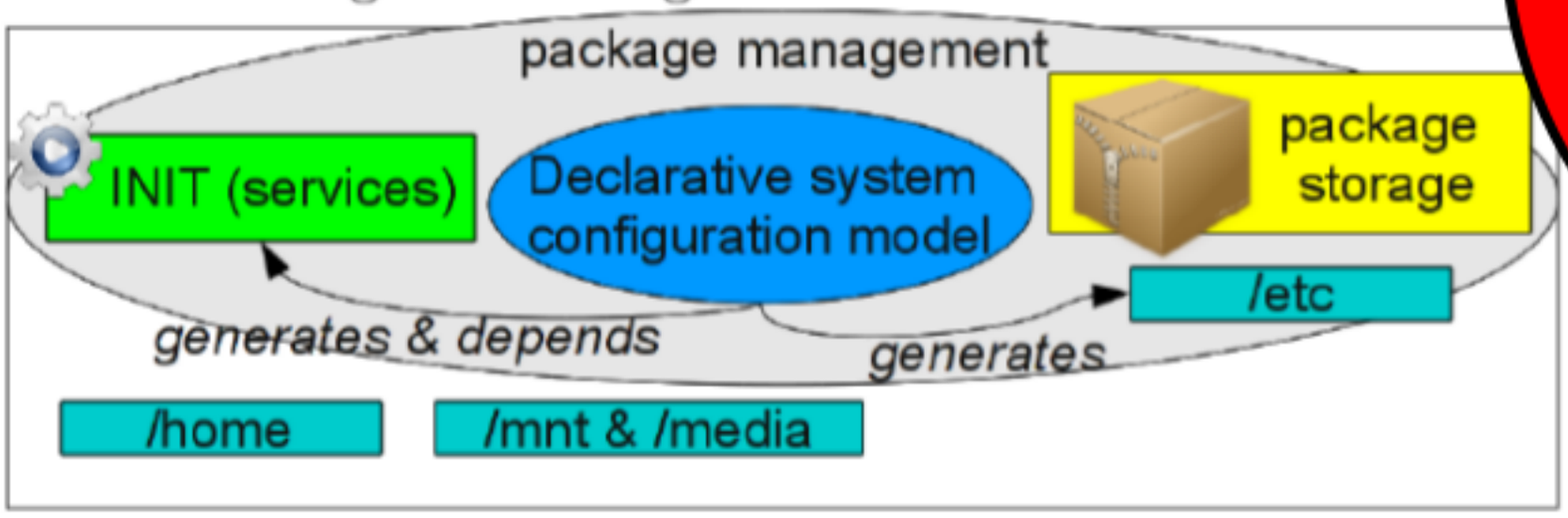
Figure 6: Integration level of Gentoo Linux



- installierte software gut erweiterbar!
- updates/upgrades geplant
- **stateful store!**

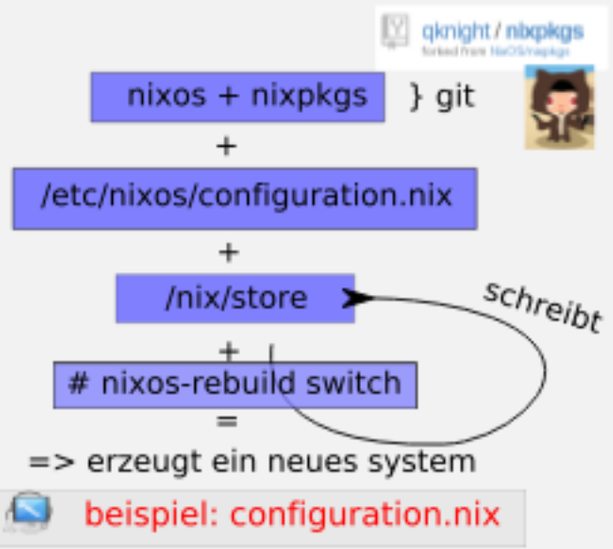
4

Figure 7: Integration level of Nix OS



- installierte software gut erweiterbar!
- updates/upgrades geplant
- **stateless store!**
- deklarative konfiguration der services!
  d.h.: konfiguration wird on-the-fly generiert!
- in NixOS wird das system 'quasi'
  von auserhalb gebaut

nixos aufbau



- Nix ist bekannt welche abhängigkeiten wirklich benötigt werden
- mit dem befehl "**nix-copy-closure**" kann man software kopieren!
- NixOS binary deployment lädt NAR files (analog zu deb) aus dem netz und installiert diese nach /nix/store/...

ACHTUNG: es gilt
deserialize(serialize(storepath)) == deserialize(NAR)

evopedia/default.nix

```
{stdenv, fetchurl, bzip2, qt4, libXtst}:

stdenv.mkDerivation rec {
  name = "evopedia-0.4.2";
  src = fetchurl {
    url = "http://lastlog.de/misc/${name}.tar.gz";
    sha256 = "88a06afc0bb4ad3a5723ea8ac12cb3210240d17ff98f55094b604c3edef3bf9";
  };
  configurePhase = ''
    qmake PREFIX=$out
  '';
  buildInputs = [ bzip2 qt4 libXtst ];
  meta = {
    description = "Offline Wikipedia Viewer";
    homepage = http://www.evopedia.info;
    license = "GPLv3+";
    maintainers = with stdenv.lib.maintainers; [viric];
    platforms = with stdenv.lib.platforms; linux;
  };
}
```

hash & storepaths

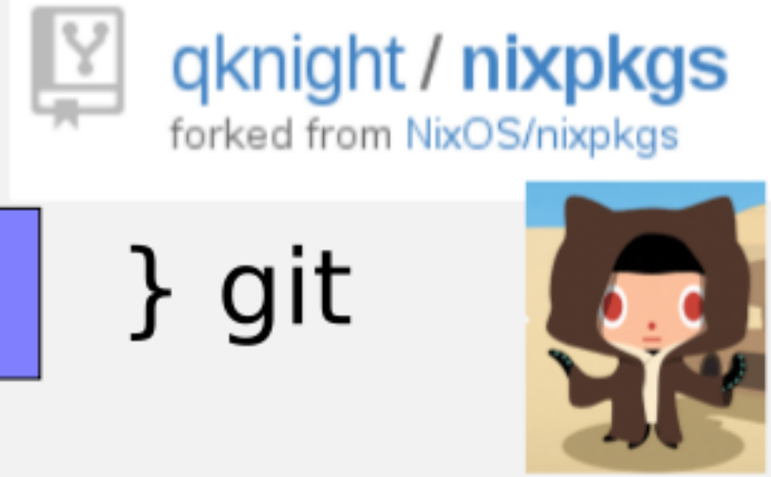
beim "source/binary deployment" gilt:
store hash (evopedia.nix) =
hash(**aller abhängigkeiten**) **&**
hash(**des evopedia.nix inhalts**) **&**
hash(**des evopedia-1.2.3.tgz downloads**)

beispiel **store hash**:
/nix/store/zzsx8izyzscfk7ag8arzvivfcw6hcgd4-evopedia-1.2.3/

frage:
ergibt sich hieraus ein nachteil?

environments & profile

- **alle software** in NixOS (nix expressions) werden **eingekapselt**
- für **c/c++** programme ist dies mit **RPATH** meist erledigt
- **python script** bekommen einen **wrapper** vorgeschalten
- ähnlich ist es für andere script-sprachen
- es **existiert nur /bin/sh** (zeigt auf bash)
  - kein #!/bin/bash
  - kein #!/usr/bin/python



nixos + nixpkgs } git

+

/etc/nixos/configuration.nix

+

/nix/store

schreibt

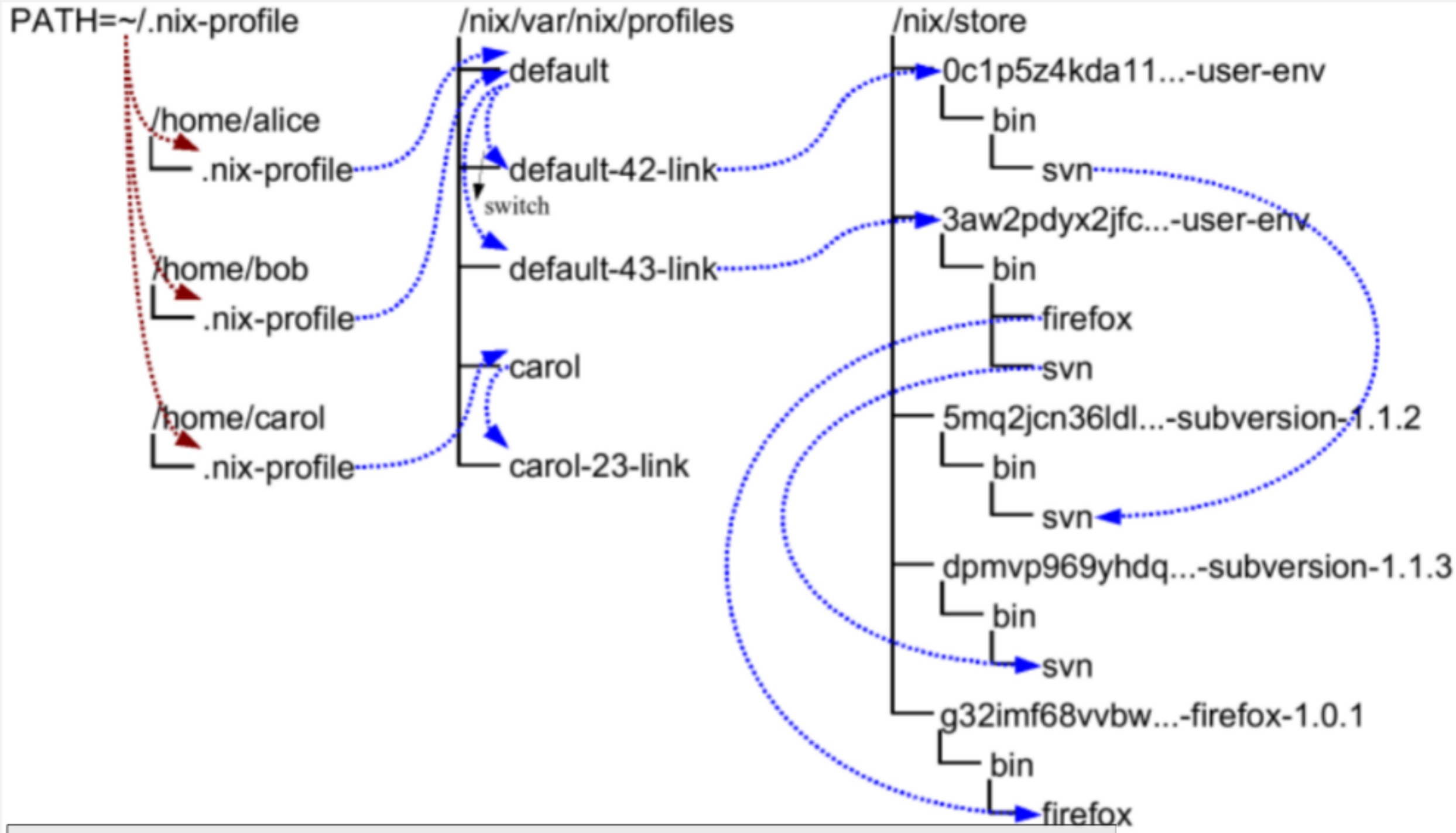
+

nixos-rebuild switch

=

=> erzeugt ein neues system

beispiel: configuration.nix



beispiel:
/nix/store/*-glibc-2.12.2/

- Nix ist bekannt welche abhängigkeiten wirklich benötigt werden
- mit dem befehl "**nix-copy-closure**" kann man software kopieren!
- NixOS binary deployment lädt NAR files (analog zu deb) aus dem netz und installiert diese nach /nix/store/...

ACHTUNG: es gilt
deserialize(serialize(storepath)) ==
deserialize(NAR)

evopedia/default.nix

```
{stdenv, fetchurl, bzip2, qt4, libX11}:

stdenv.mkDerivation rec {
  name = "evopedia-0.4.2";
  src = fetchurl {
    url = "http://lastlog.de/misc/${name}.tar.gz";
    sha256 = "88a06a0c0bb4ad3a5723ea9ac12cb33102484d17ff86f55094b504c3edef3bf9";
  };
  configurePhase = ''
    qmake PREFIX=$out
  '';
  buildInputs = [ bzip2 qt4 libX11 ];
  meta = {
    description = "Offline Wikipedia Viewer";
    homepage = http://www.evopedia.info;
    license = "GPLv3+";
    maintainers = with stdenv.lib.maintainers; [viric];
    platforms = with stdenv.lib.platforms; linux;
  };
}
```

hash & storepaths

beim "source/binary deployment" gilt:
store hash (evopedia.nix) =
hash(**aller abhängigkeiten**) **&**
hash(**des evopedia.nix inhalts**) **&**
hash(**des evopedia-1.2.3.tgz downloads**)

beispiel **store hash**:
/nix/store/zzsx8izyzscfk7ag8arzvivfcw6hcgd4-evopedia-1.2.3/

frage:
ergibt sich hieraus ein nachteil?

environments & profile

- **alle software** in NixOS (nix expressions) werden **eingekapselt**
- für **c/c++** programme ist dies mit **RPATH** meist erledigt
- **python script** bekommen einen **wrapper** vorgeschalten
- ähnlich ist es für andere script-sprachen
- es **existiert nur /bin/sh** (zeigt auf bash)
  - kein #!/bin/bash
  - kein #!/usr/bin/python

service deployment

NixOS beschreibt auch services deklarativ

ein service benötigt:
- eine **nix expression (in nixpkgs)**
- eine **nix service expression**

beispiel:
cntlm.nix

impurities

einfluss hat:

- **hardware** x86 / x64 (arch)
- **timestamps** der compiler
- **parallelität** (make -j4)
- **dynamischer linker** sucht in /lib

beispiel:
https://nixos.org/wiki/Nix_impurities

systemprofile

NixOS unterscheidet zwischen:
- dem systemprofil
- den nutzerprofilen (mehrere)

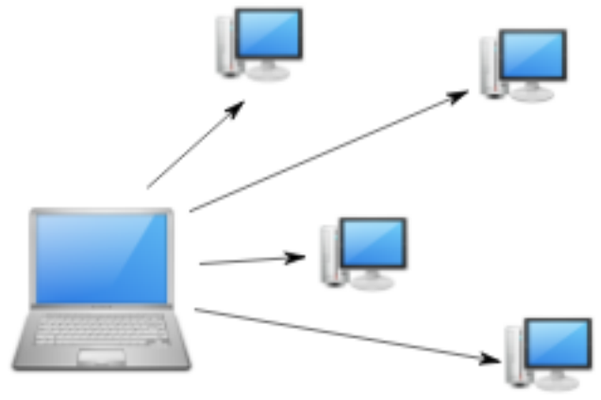
wer software schreibt will sicher tools wie:
- make / qmake
- qt4 libs

uvm

beispiel:
load-env-srmg

Disnix/nixOPs - verteiltes deployment

- zentrale stelle verwaltet
  weitere rechner
- Nix garantiert auch verteilt
  rollbacks/rollouts



- nutzung für:
  - komplexe setups
  - unit-tests

§ **rechtlicher hinweis:**
teile dieser präsentation wurden aus dem internet ausgeliehen, werden aber nach beendigung zurückgegeben!

?

fragen

---> 

shutdown