OpenSolaris : fonctionnement des zones et de zfs

Bruno Bonfils, <asyd@asyd.net>
Association GUSES
http://www.guses.org/home/

Sommaire

- Présentation d'OpenSolaris
- Les zones
  - Plusieurs instances d'OpenSolaris
  - Brandz : l'émulation Linux
- ZFS
  - Comment gérer simplement son espace de stockage
  - Ne plus s'inquiéter pour ses données !

Présentation d'OpenSolaris

- En 2005, début de la mise à disposition des sources de Solaris sous le nom de code OpenSolaris (premier projet : dtrace)
- Aujourd'hui, le terme OpenSolaris couvre aussi bien les sources, que les communautés
- Mais, ce n'est pas une distribution !
  - Une distribution commerciale : Solaris 10
  - Des distributions communautaires : Belenix, Nexenta (bientôt Debian ?)

OpenSolaris

- Les communautés sont composées de personnes Sun, et de membres externes
- Une équipe structurée, ordonnée, aidée (ex : Sun Legal)
- Des projets annexes (ex : OpenGrok)
- Rejoignez le OSUG francophone !
  - http://fr.opensolaris.org/
  - http://guses.org/home/

OpenSolaris

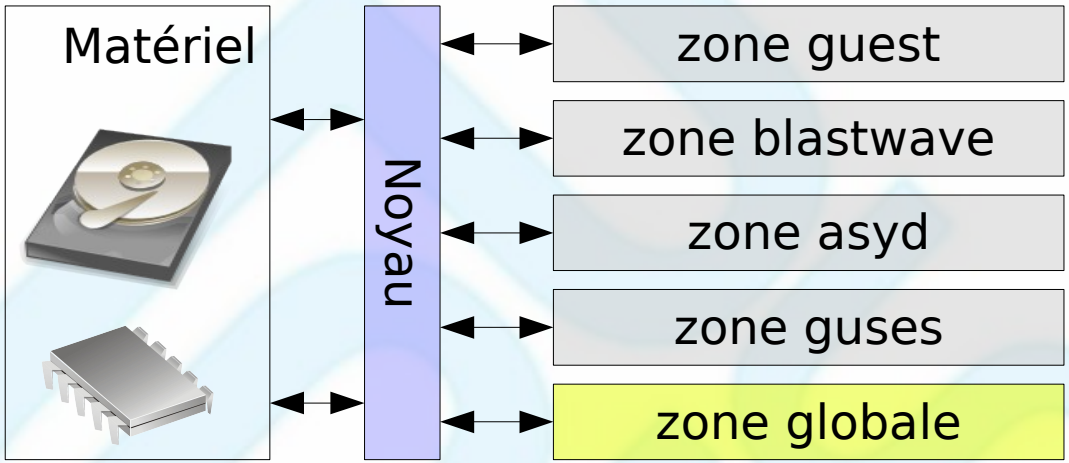
- Quelques communautés
  - Zones
  - BrandZ (zones Linux)
  - DTrace
  - Xen
  - ZFS
  - Storage
  - PowerPC
  - SMF (Gestionnaire de services)

Les zones

Les zones : introduction

- Permet plusieurs instances de l'OS
- Voir une zone « comme » :
  - un chroot
  - des systèmes de fichiers hérités ou dédiés
  - des adresses réseaux dédiées
- Partage des ressources matérielles
- Des privilèges restreints
- Zone Linux maintenant possible avec BrandZ

Les zones : vue d'ensemble



- Une seule instance du noyau
- Une et une seule zone globale
- N zones non globales (Solaris ou Linux)
- Accès à toutes les zones depuis la globale

Restriction des privilèges

- Exemple de privilèges interdits :
  - PRIV_NET_RAWACCESS (accès brut aux interfaces réseaux)
  - PRIV_PROC_ZONE (envoyer un signal vers un processus d'une autre zone)
  - PRIV_SYS_DEVICES (création de périphériques)
  - PRIV_SYS_NET_CONFIG (modification de la configuration réseau)

Les zones : création

Configuration minimale d'une zone :

```
# zonecfg -z guses
zonecfg:guses> create [-t template]
zonecfg:guses> set zonepath=/zones/guses
zonecfg:guses> set autoboot=true
zonecfg:guses> add net
zonecfg:guses:net> set physical=bge0
zonecfg:guses:net> set address=xx.xx.xx.xx
zonecfg:guses:net> end
zonecfg:guses> verify
zonecfg:guses> commit
zonecfg:guses> exit
```

Les zones : les ressources

- FS
  - Système de fichier hérité
  - Partage d'un système de fichier (lofs)
    - en lecture seule
    - en lecture / écriture
  - Système de fichier dédié
- Réseau
  - Adresse IP (choix de l'interface physique, de l'adresse)

Administration d'une zone

- Installation et configuration initiale

```
# zoneadm -z guses install
# zoneadm -z guses boot
# zlogin -C guses
```

- Opération de maintenance

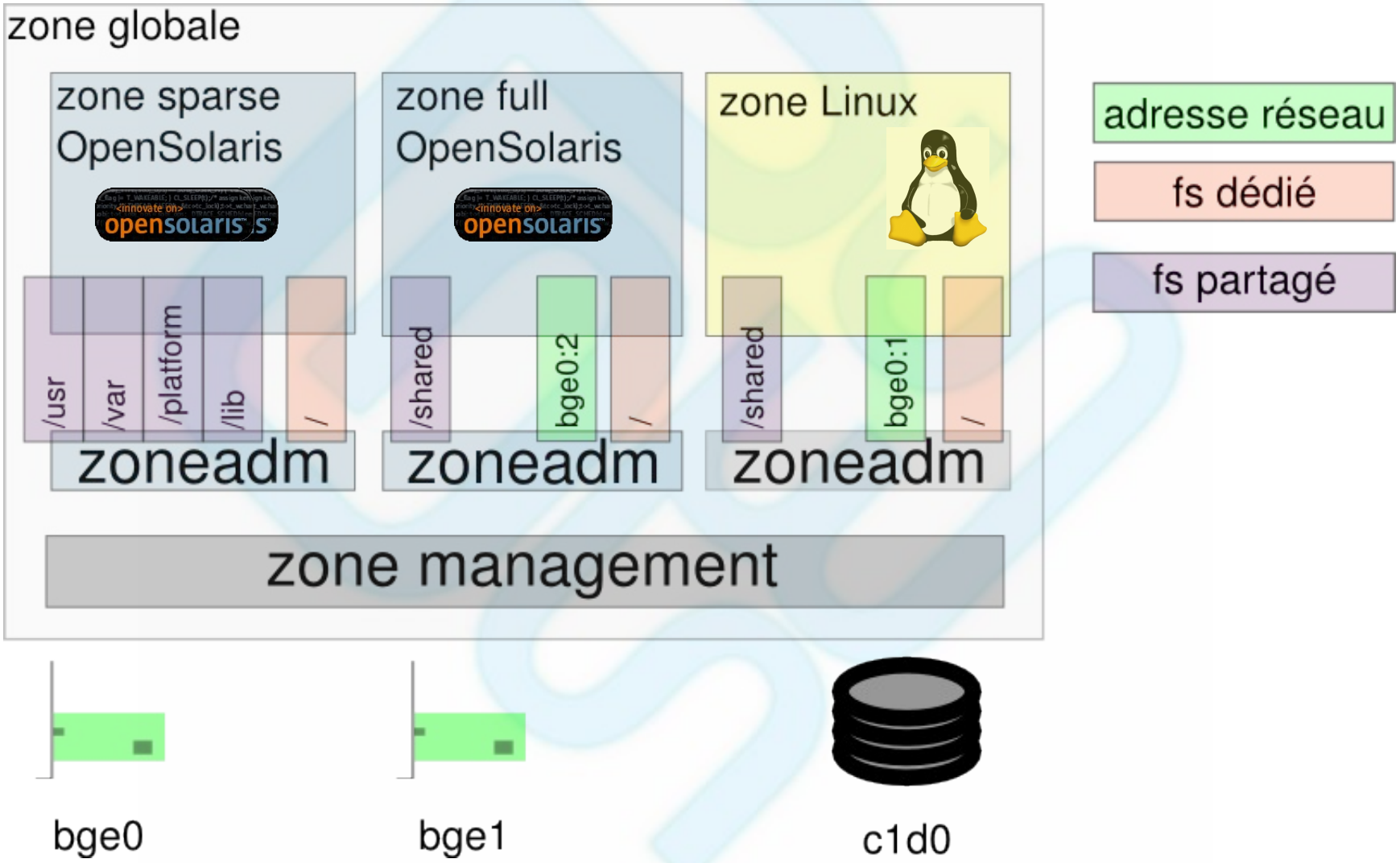
```
# zoneadm -z guses [boot|halt|reboot]
```

- Définition d'un *processor set*

```
# zonecfg -z guses 'set pool=guest'
```

Un *pset (processor set)* est un ensemble de processeurs qui seront dédiés aux éléments utilisant ce processor set (par exemple des zones). Plusieurs zones peuvent être associés au même *processor set*. La gestion des *pset* est totalement dynamique.

Exemple d'architecture



Fonctionnement des zones

- Utilisation d'un membre *zoneid* dans les structures noyaux
  - la gestion des processus
  - la pile IP
  - les services RPC (ex: NFS)
  - les IPC
- L'ensemble du noyau est développé en connaissance des zones, d'où une intégration parfaite
- Évolution en cours : crossbow

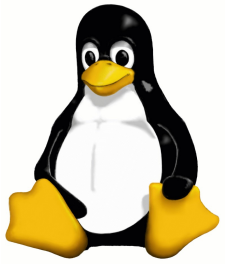
Exemple d'utilisation de zoneid

Extrait de procfs.h

```
272 typedef struct psinfo {
273         int     pr_flag;        /* process flags (DEPRECATED; do not use) */
274         int     pr_nlwp;        /* number of active lwps in the process */
275         pid_t   pr_pid;         /* unique process id */
276         pid_t   pr_ppid;        /* process id of parent */
277         pid_t   pr_pgid;        /* pid of process group leader */
278         pid_t   pr_sid;         /* session id */
279         uid_t   pr_uid;         /* real user id */
280         uid_t   pr_euid;        /* effective user id */
281         gid_t   pr_gid;         /* real group id */
282         gid_t   pr_egid;        /* effective group id */
283         uintptr_t pr_addr;      /* address of process */
284         size_t  pr_size;        /* size of process image in Kbytes */
285         size_t  pr_rssize;      /* resident set size in Kbytes */
286         size_t  pr_pad1;
[..]
304         taskid_t pr_taskid;     /* task id */
305         projid_t pr_projid;     /* project id */
306         int     pr_nzomb;       /* number of zombie lwps in the process */
307         poolid_t pr_poolid;     /* pool id */
308           zoneid_t pr_zoneid;     /* zone id */
309         id_t    pr_contract;    /* process contract */
310         int     pr_filler[1];   /* reserved for future use */
311         lwpsinfo_t pr_lwp;      /* information for representative lwp */
312 } psinfo_t;
```

```
typedef int id_t; /* A process id, */
typedef id_t zoneid_t;
```

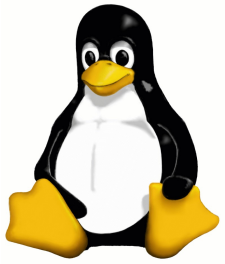
Les zones Linux : BrandZ



- **Émulation d'un noyau Linux 2.4**
- Émulation partielle du système de fichier procfs
- Installation depuis la zone globale au travers de fichiers ISO, de tarballs, ou de CDROM
- S'administre de la même façon qu'une zone (Open)Solaris

```
# zonecfg -z linux
zonecfg:linux> create -t SUNWlx
```

Fonctionnement de BrandZ



- Interception de l'interruption Linux 80
  - Code en assembleur
- Traduction des appels systèmes (lx_emulate)
  - Appel natif (mkdir)
  - Modifications mineures, notamment sur la gestion d'erreur (lx_rmdir)
  - Totalement réécrits (lx_ptrace)

lx_emulate

mappage des appels systèmes :

```
 959 static struct lx_sysent sysents[] = {
 960         {"nosys",       NULL,           NOSYS_NULL,     0},     /*  0 */
 961         {"exit",        lx_exit,        0,              1},     /*  1 */
 962         {"fork",        lx_fork,        0,              0},     /*  2 */
 963         {"read",        lx_read,        0,              3},     /*  3 */
 964         {"write",       write,          SYS_PASSTHRU,   3},     /*  4 */
[...]
 997         {"kill",        lx_kill,        0,              2},     /* 37 */
 998         {"rename",      lx_rename,      0,              2},     /* 38 */
 999         {"mkdir",       mkdir,          SYS_PASSTHRU,   2},     /* 39 */
1000         {"rmdir",       lx_rmdir,       0,              1},     /* 40 */
```

lx_rmdir :

```
 402 int
 403 lx_rmdir(uintptr_t p1)
 404 {
 405         int r;
 406
 407         r = rmdir((char *)p1);
 408         if (r < 0)
 409                 return ((errno == EEXIST) ? -ENOTEMPTY : -errno);
 410         return (0);
 411 }
```

ZFS

Présentation de ZFS

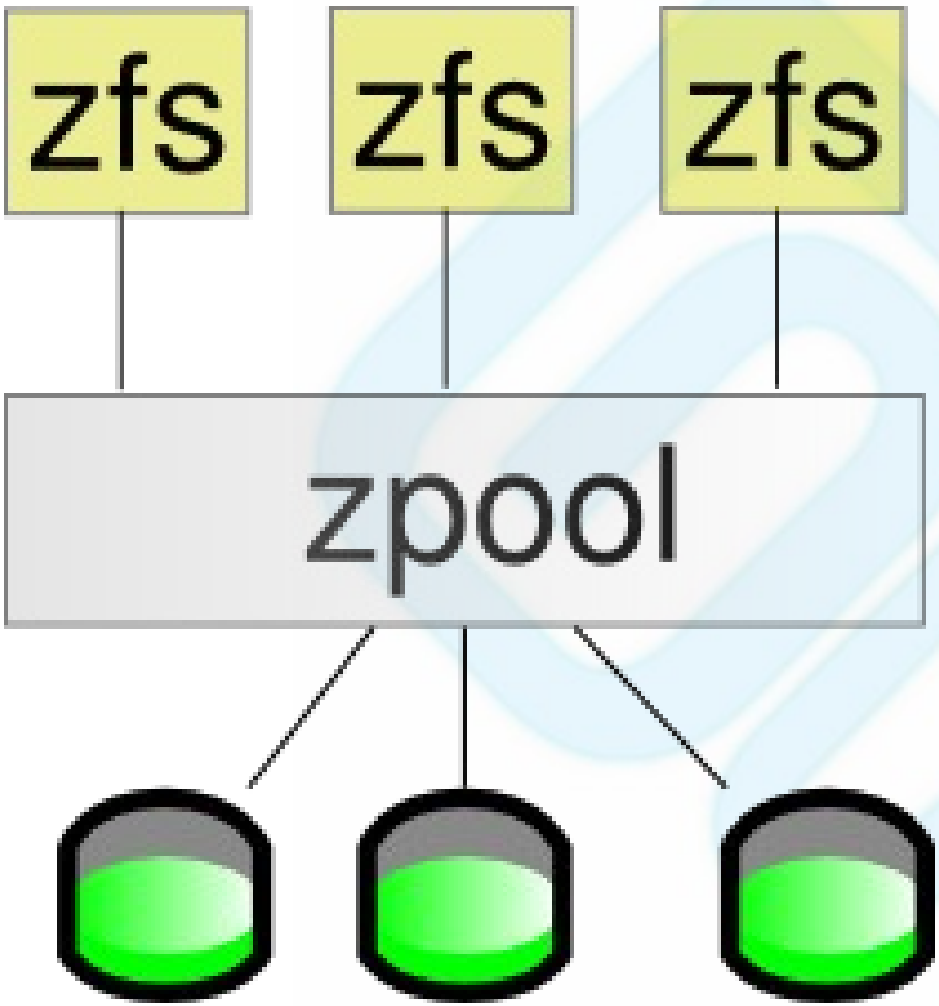
- Gérer simplement vos données
- Une facilité de sauvegarde accrue grâce aux snapshots
- Garantie la sécurité de vos données
  - miroir, raidz, raidz2
  - mécanisme de *copy on write* (ne pas modifier des blocs de données existants)
  - utilisation de *checksum*

« if 1,000 files were created every second, it would take about 9,000 years to reach the limit of the number of files »

Présentation de ZFS

- Système de fichier et gestionnaire de volume ne font qu'un
- Plusieurs vues instantanées (*snapshot*) par volume
- Plusieurs niveaux de hiérarchies possibles
- Délégation d'administration (utilisation liée aux zones)

ZFS : vue d'ensemble



- réduction, extension à chaud
- tout l'espace disponible est partagé entre les volumes
- bande passante globale
- mirror, raidz, raidz2

ZFS : Création d'un pool

- Création d'un pool en mode miroir

```
# zpool create data mirror c0t0d0 c0t1d0
```

- Administration du pool

```
# zpool status data
# zpool iostat 1
# zpool scrub
```

```
# zpool iostat
               capacity     operations    bandwidth
pool        used  avail   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
data        15.5G  59.0G      0      3  4.15K  37.3K
data        15.5G  59.0G      0      0      0      0
data        15.5G  59.0G      0    109      0    352K
```

```
# zpool status data
  pool: data
 state: ONLINE
 scrub: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        data        ONLINE       0     0     0
          c0t0d0    ONLINE       0     0     0
          c0t1d0    ONLINE       0     0     0

errors: No known data errors
```

ZFS : Création de volumes

- Création d'une hiérarchie de volumes

```
# zfs create data/zones
# zfs create data/home
# zfs create data/home/asyd
# zfs create data/home/lei
# zfs create data/home/murlock
```

- Il n'est pas nécessaire de modifier le fichier /etc/vfstab, le montage est automatique

Définition des propriétés

- Quota pour l'ensemble du volume home

```
# zfs set quota=100g data/home
```

- Quota par utilisateur

```
# zfs set quota=50g data/home/asyd
# zfs set quota=50g data/home/maria
# zfs set quota=50g data/home/murlock
```

- Réservation d'espace

```
# zfs set reversation=10g data/home/asyd
```

ZFS : Les propriétés d'un volume

- Propriétés modifiables
  - quota : taille maximum autorisée
  - reservation : réserve l'espace disque (le rend inutilisable pour les autres volumes)
  - mountpoint : point de montage
  - sharenfs : partage le volume en NFS
  - compression : active la compression à la volée
  - snapdir : répertoire des snapshots visible ou non

ZFS : les snapshots

- Manipulation de snapshots

```
# zfs snapshot data/zones/guses@20070610
# zfs send data/zones/guses@20070610 | ssh
# zfs rollback data/zones/guses@20070610
```

- Ne nécessite pas d'espace disque supplémentaire à la création. Seules les nouvelles données consommeront de l'espace disque.

Exemple d'utilisation des snapshots

- Création d'un snapshot par jour

  @lundi, @mardi, @mercredi, etc.

- Sauvegarde différentielle

```
# zfs send -i data/home/asyd@lundi \
  data/home/asyd@mardi | ssh ..
```

- Sauvegarde complète

```
# zfs send data/home/asyd@dimanche
```

- Montage des snapshots

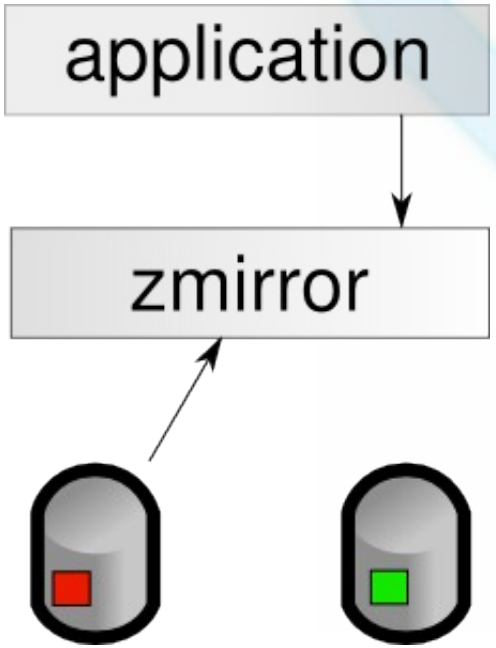
```
# zfs set snapdir=visible data/home
# ls ~/.zfs/snapdir/lundi
```

Le fonctionnement de ZFS

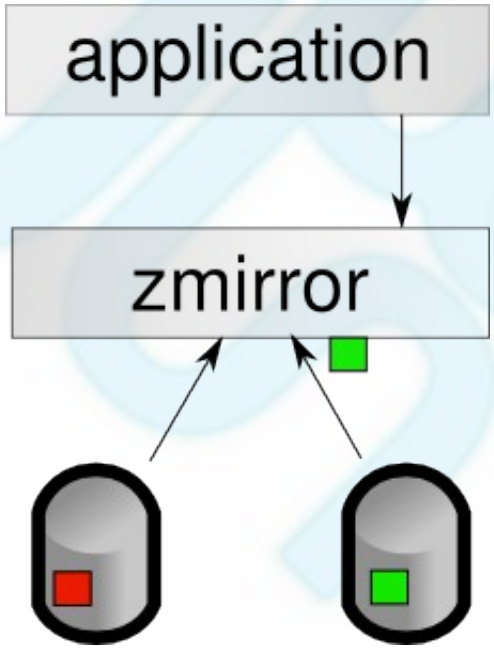
- mirror, raidz
- Architecture
- L'intégrité des données
  - CoW (Copy on Write)
  - Checksum
    - fletcher
    - sha256

ZFS : Mode miroir

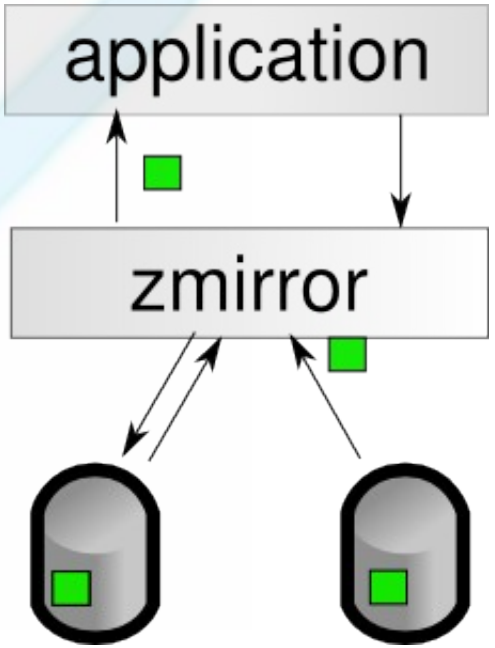
L'application génère une demande de lecture. Le premier disque est utilisé, mais une erreur est détectée par vérification du checksum.



Le deuxième disque est sollicité, le checksum est correct.



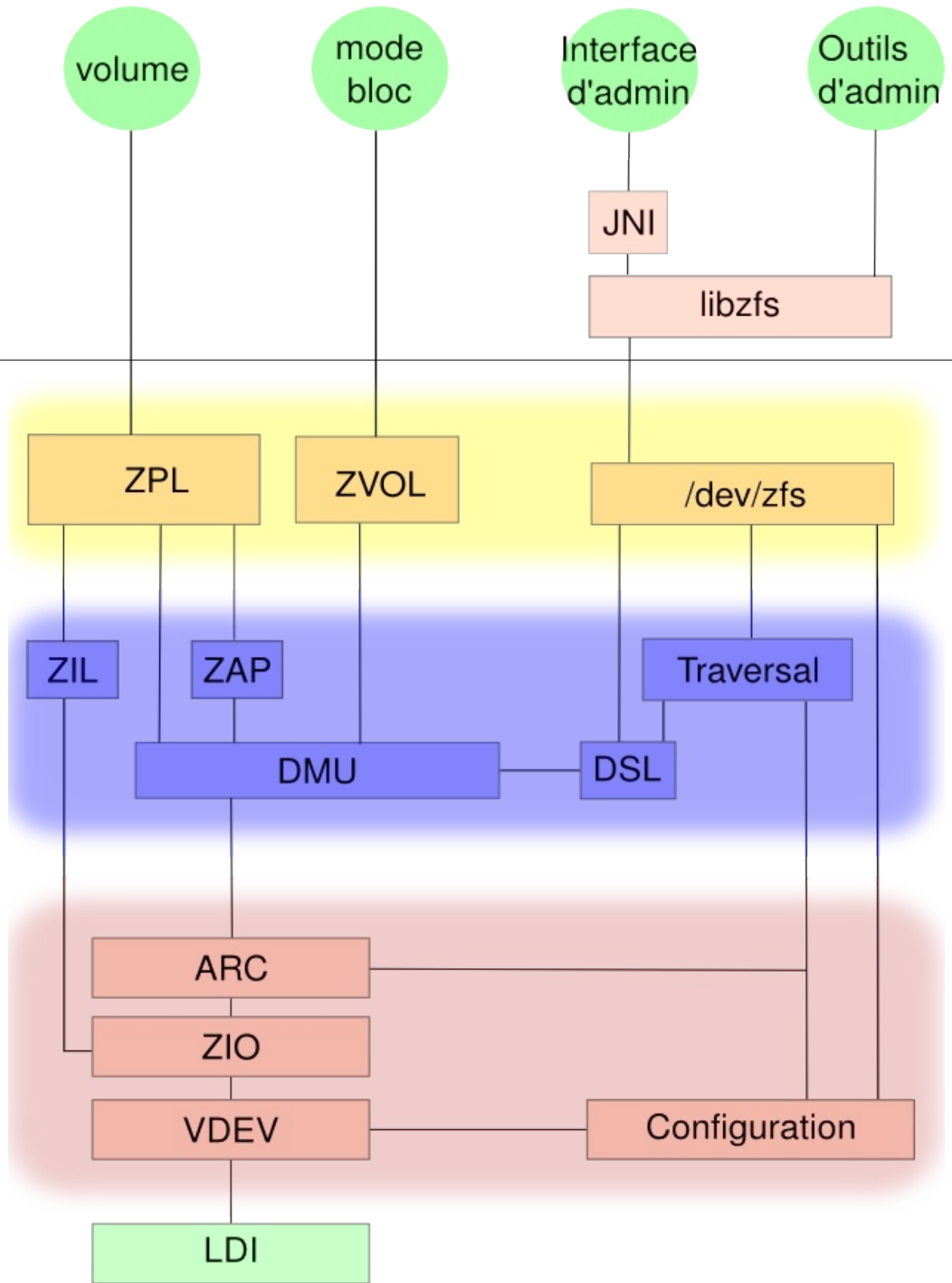
Les données correctes sont renvoyées à l'application, et le bloc du premier disque est mis à jour avec la valeur correcte.



Le raidz et raidz2

- Similaire au RAID5, mais l'utilisation du COW le rend plus performant
  - Pas de « write hole » (trou d'écriture)
- Détection des anomalies (checksum)
- Le raidz2 utilise une double parité, à partir de 3 disques seulement

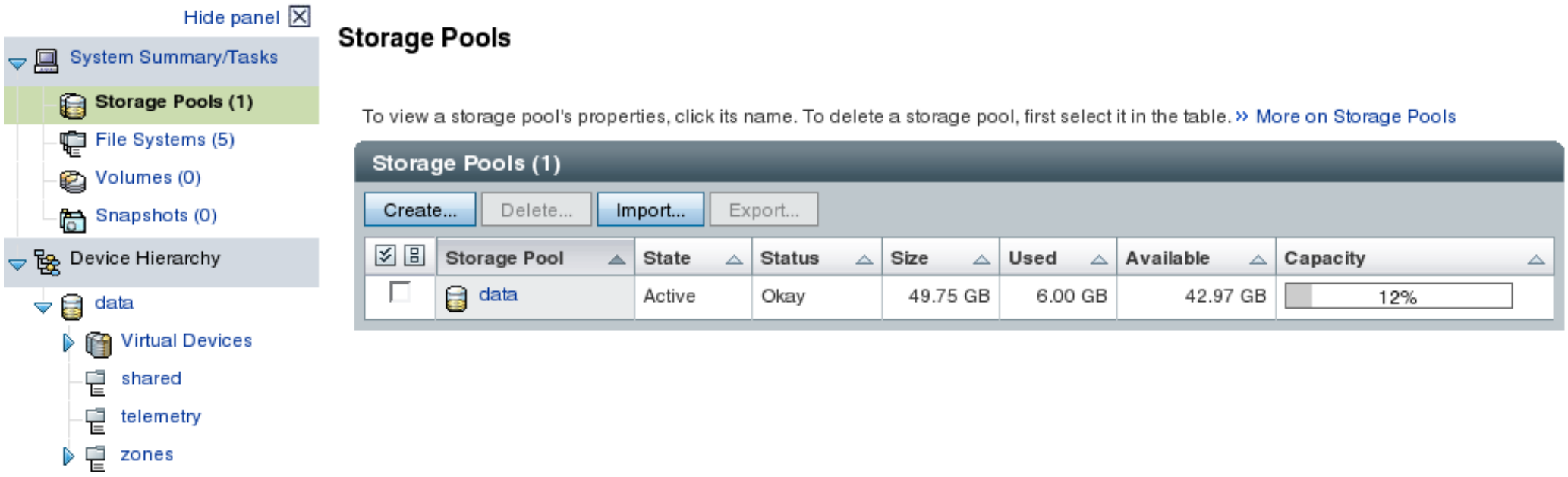
L'architecture de ZFS



Les principales couches

- ZPL (ZFS Posix Layer) interface implémentant le système de fichier
- DMU (Data Management Unit)
  - Gestion des transactions
- Vdev : Gestion bas niveau des périphériques de stockage en forme d'arbre
  - Gestion du mirror, raidz, raidz2
  - Compression, checksum

Interface d'administration web

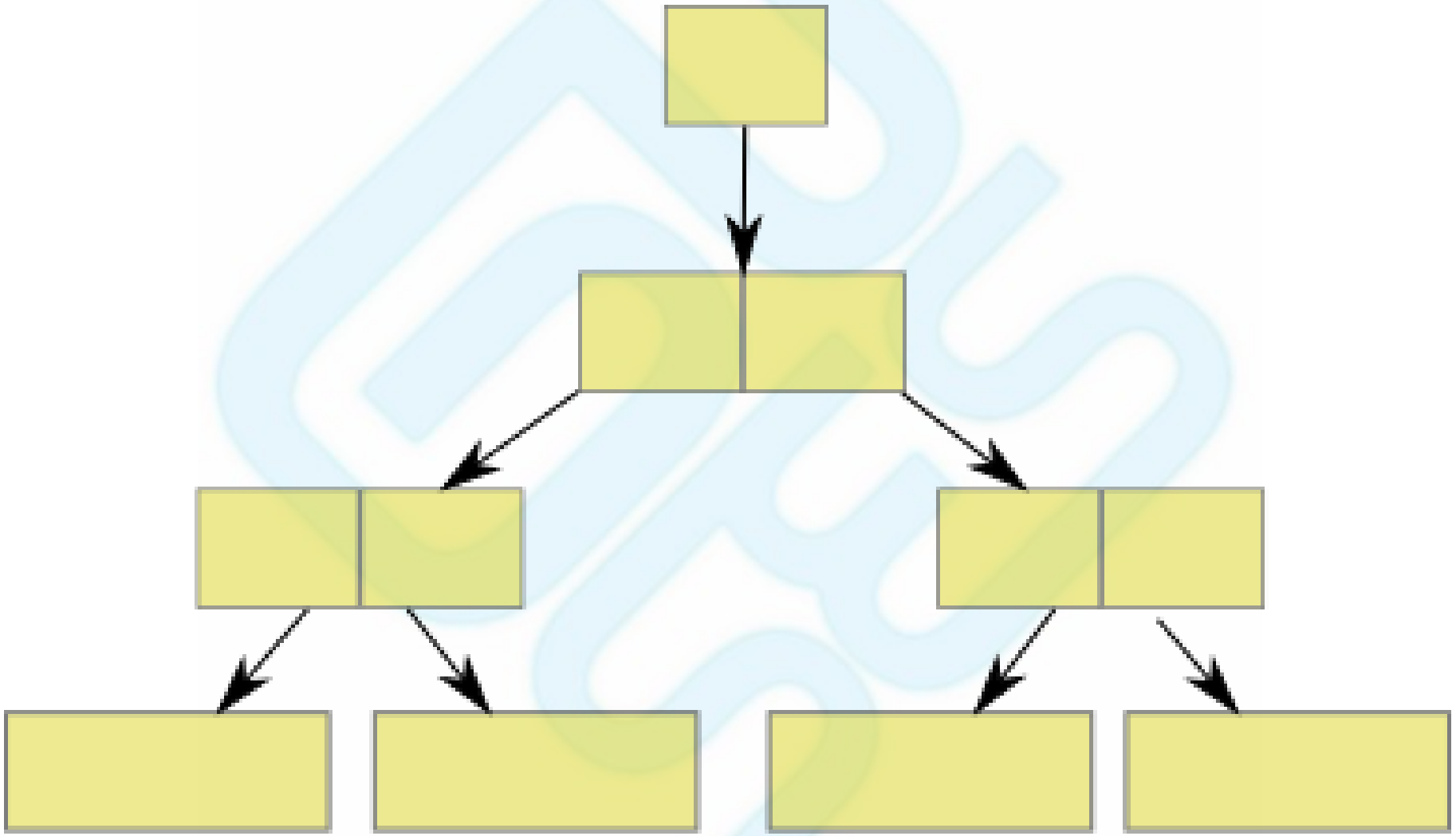


Intégrité des données

- Le COW en action !
  -
  -

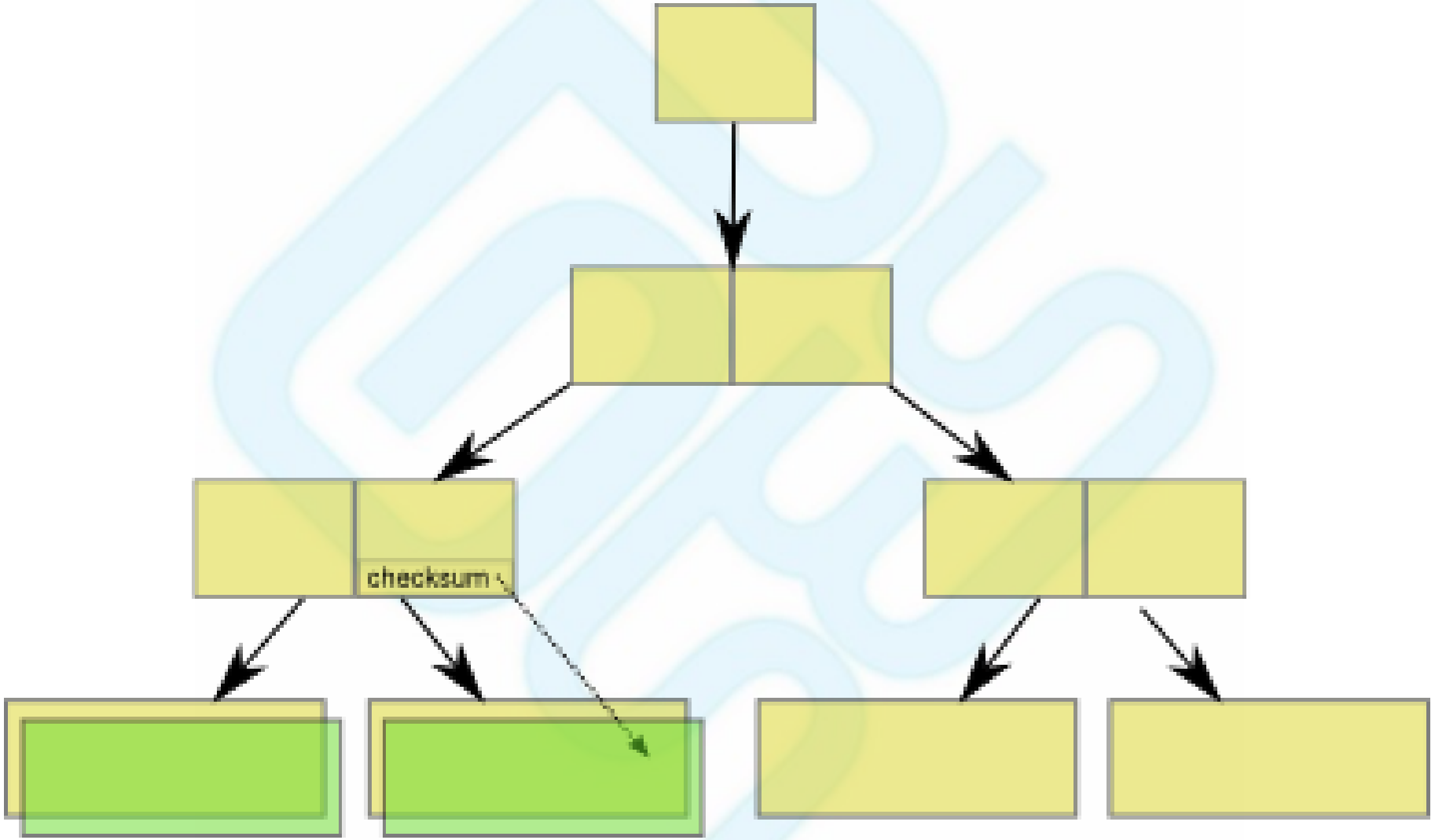
Modification d'un fichier existant

Intégrité des données



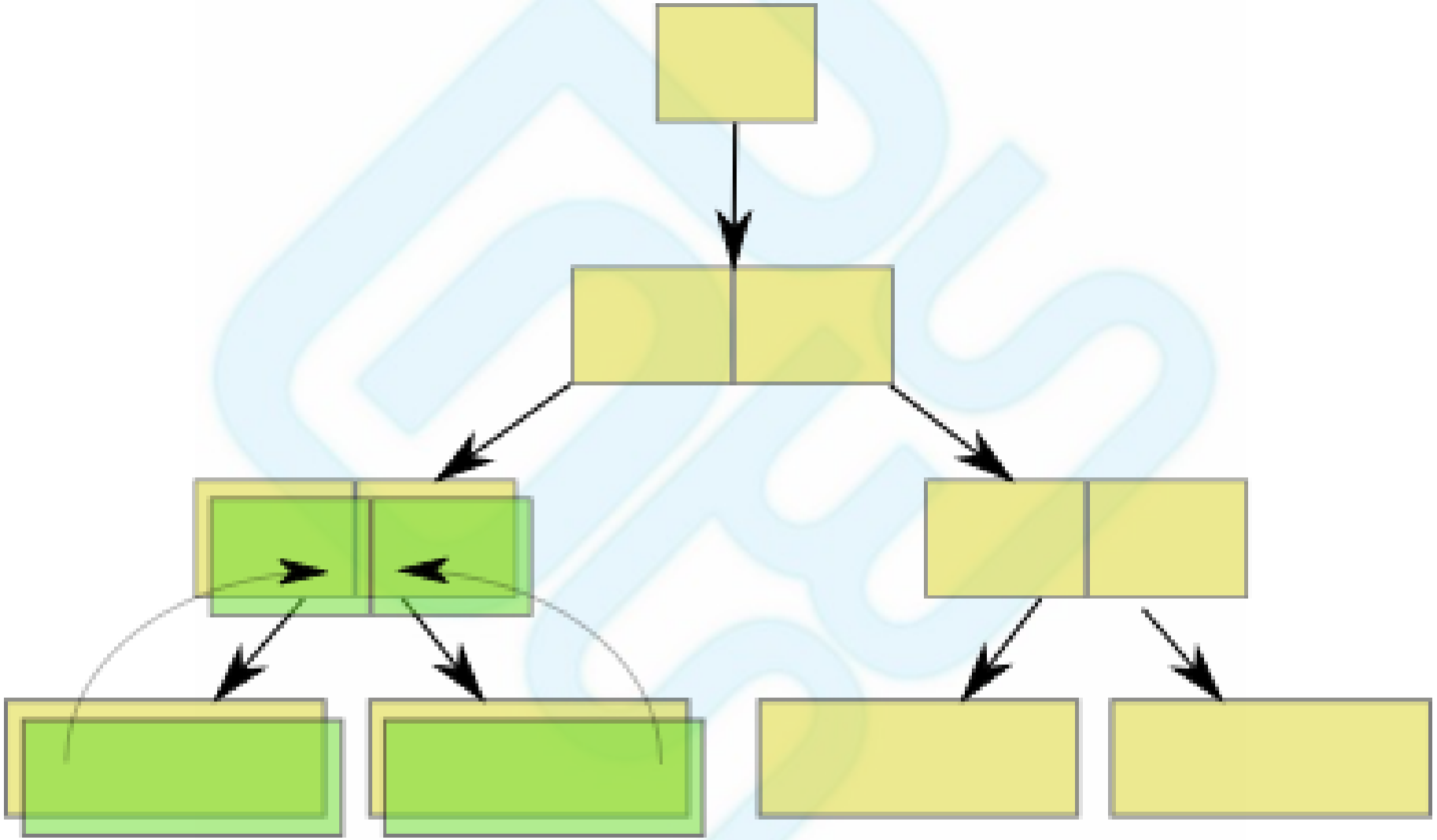
Les carrés jaunes représentent les données d'un fichier existant.

Intégrité des données



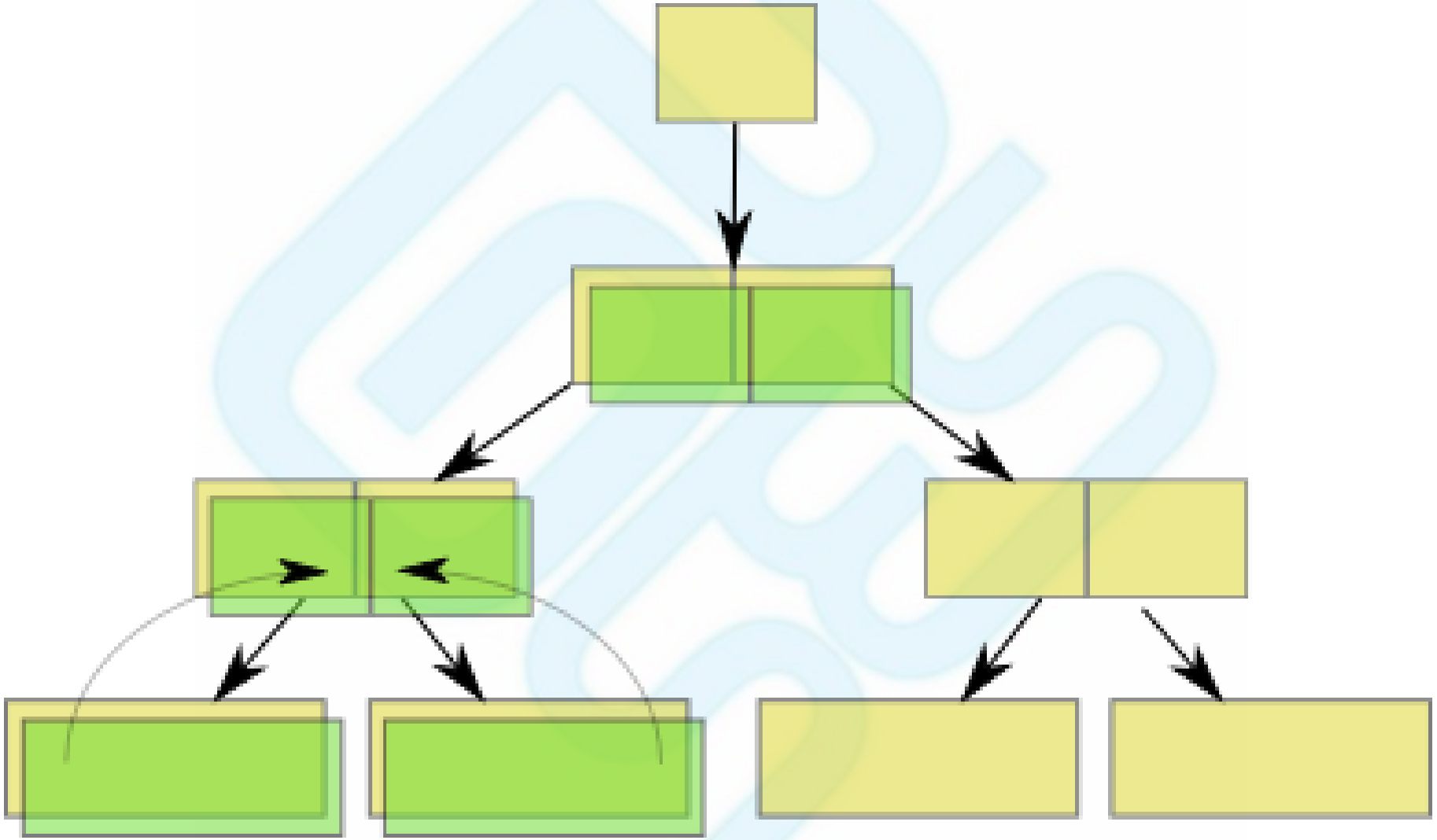
Une application génère une modification du fichier existant. Cette modification porte sur deux blocs de données, mais est partielle sur le second bloc. On copie donc les données depuis le bloc existant, on vérifie son checksum, et on apporte les modifications sur le nouveau bloc.

Intégrité des données



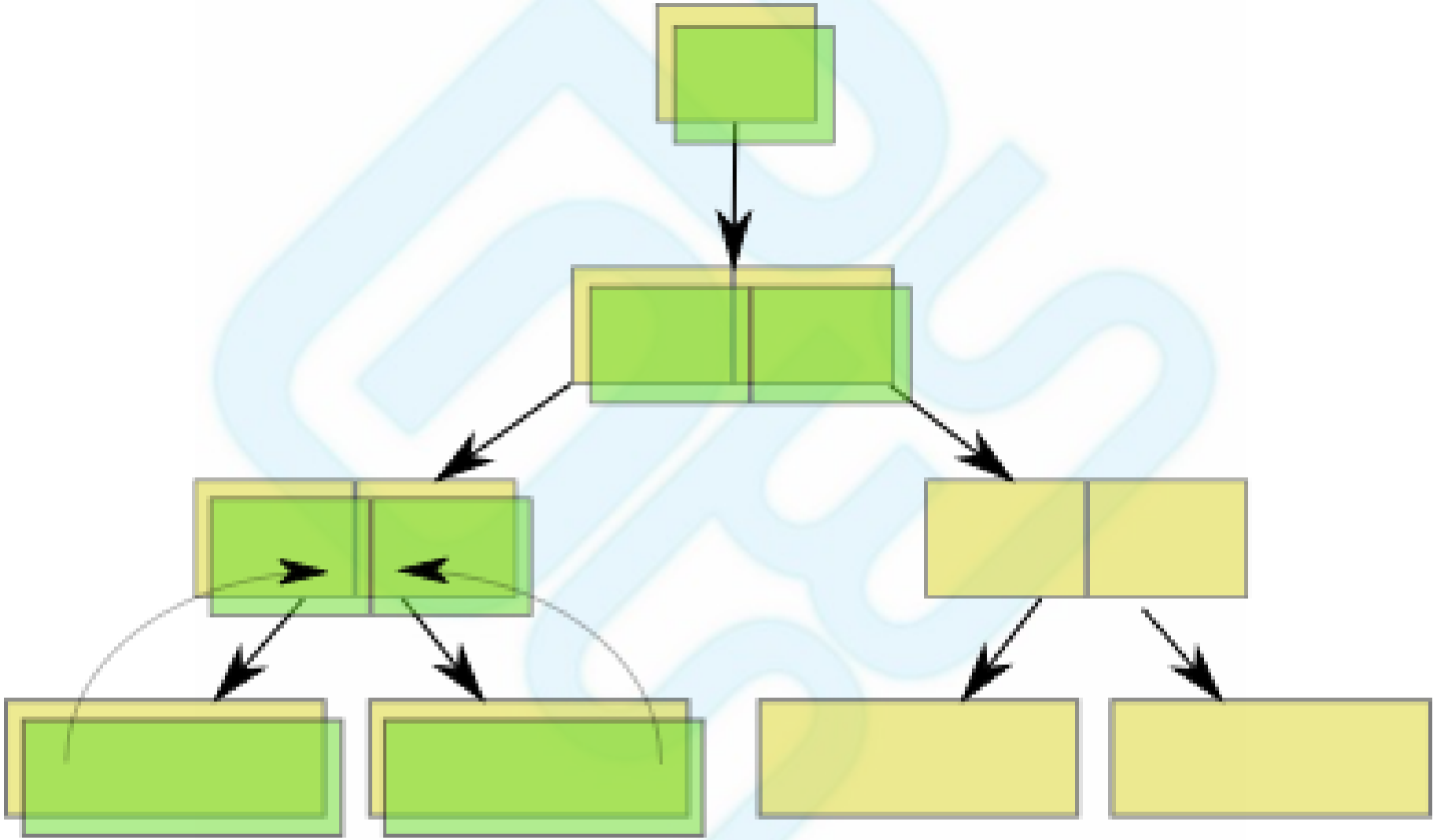
De nouveaux blocs de références (block pointers) sont créés, on y insère les références vers les blocs de données ainsi que leurs checksums.

Intégrité des données



On continue de remonter.

Intégrité des données



L'uberblock (équivalent du superblock) est modifié. S'il n'y a pas de snapshots, les anciens blocs sont libérés. Il est donc moins coûteux (en terme de CPU) d'utiliser des snapshots !

Un mot sur l'optimisation ZFS

- Utilisation d'un cache de type ARC (Adaptive Replacement Cache)
  - Utilisation de deux listes au lieu d'une (récemment et fréquemment versus récemment)
- lzjb, l'algorithme de compression utilisé par zfs a été développé spécifiquement pour ZFS, par Jeff Bonwick
  - Utilisation possible de gzip dans OpenSolaris

Démonstration

Fin

- Questions / réponses
- Site OpenSolaris
- La communauté zones
- La communauté BrandZ
- La communauté ZFS
- Le OpenSolaris User Group francophone
- Solaris Internals, Seconde édition