Christian Bornträger

cborntra@de.ibm.com

Execute-in-place technology for SLES9

Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

IBM *	z/VM *
VM/ESA *	zSeries *
S/390 *	

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.
Intel is a trademark of the Intel Corporation in the United States and other countries.
Linux is a registered trademark of Linus Torvalds
Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries.
Penguin (Tux) compliments of Larry Ewing.
UNIX is a registered trademark of The Open Group in the United States and other countries.

* All other products may be trademarks or registered trademarks of their respective companies.

Set up process overview

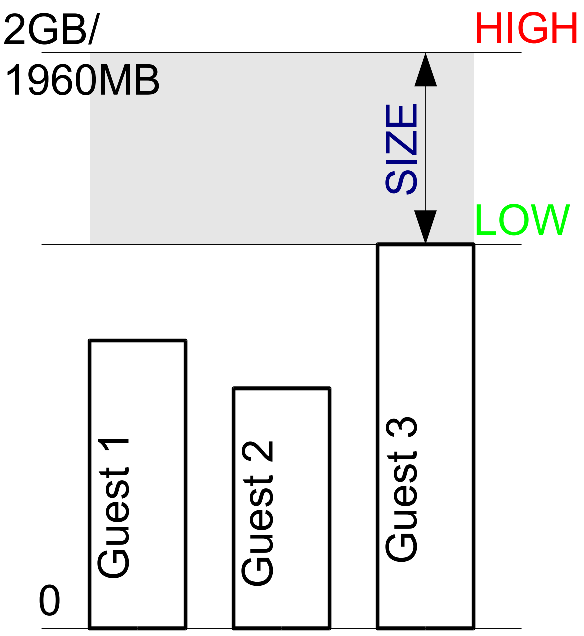
- Plan the layout of the file system
- Provide a script to over-mount shared directories on startup
- Create a file system image
- Create a DCSS from the image file
- Change the kernel parameter line
- Test the DCSS
- Activate execute in place

Documentation and scripts used in this presentation are available at

http://awlinux1.alphaworks.ibm.com/developerworks/linux390/april2004_documentation.shtml

Plan the layout of the file system

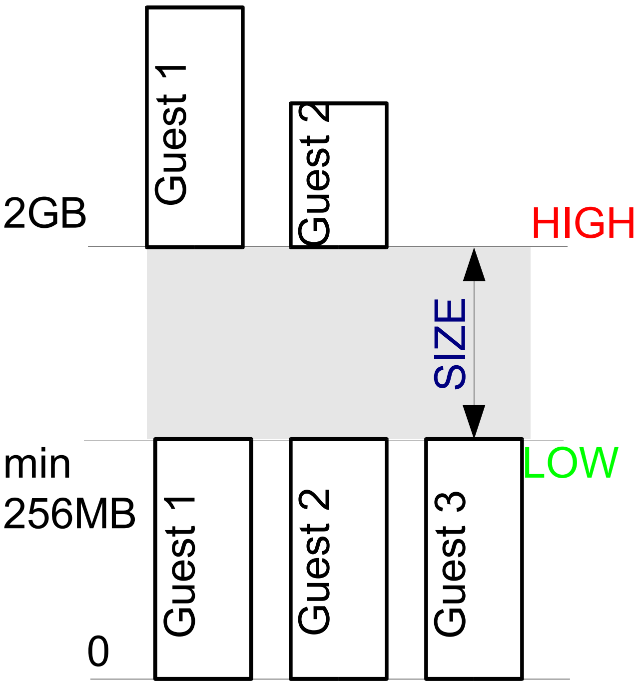
- determine the maximum size of the DCSS (SIZE)
  - lower limit of the DCSS address range (LOW)
    - may not overlap virtual guest storage
    - equals the end of virtual guest storage of the largest guest
  - upper limit of the DCSS address range (HIGH)
    - 2GB for 64-bit Linux images (z/VM limitation)
    - 1960 MB for 31-bit Linux images (Linux limitation)
  - subtract the lower limit from the upper limit to obtain the maximum possible DCSS size



SIZE = HIGH - LOW

Plan the layout of the file system

- alternative memory layout with 64bit guests allows to use more virtual guest storage
  - a minimum of 256 Mbyte below the 2GB line is required for reliable operation
  - the amount of virtual guest storage per guest is not limited by DCSS
  - the maximum size of the DCSS can be 1792 Mbyte



SIZE = HIGH - LOW

Plan the layout of the file system – which directories

- On servers, identify directories containing frequently executed files
  - issue the "**`ps`**" command on a typical server running typical workload to identify what processes are running
  - use "**`which`**" to find out where the executable files are stored
  - use "**`ldd`**" to find out what libraries are required

```
# ldd /bin/bash
                libreadline.so.4 => /lib64/libreadline.so.4 (0x0000010000021000)
                libhistory.so.4 => /lib64/libhistory.so.4 (0x0000010000063000)
                libncurses.so.5 => /lib64/libncurses.so.5 (0x000001000006c000)
                libdl.so.2 => /lib64/libdl.so.2 (0x00000100000d4000)
                libc.so.6 => /lib64/libc.so.6 (0x00000100000d8000)
                /lib/ld64.so.1 => /lib/ld64.so.1 (0x0000010000000000)
```

  - check for symbolic links

```
# find <directory> -type l -exec ls -lisa {} \;
```

Plan the layout of the file system – which directories

- Interactive systems
  - use the same rules as before
  - check the **`PATH`** environment variable for the superuser and for regular users
  - check **`/etc/ld.so.conf`** for paths containing libraries
- what NOT to share?

  - be sure not to share directories that are written to. Note that the process described here shares subdirectories as well.
  - sharing scripts and java bytecode is ineffective, they are not executed directly but interpreted
  - do not share **`/etc`** and **`/var`**
  - use the file command to check the file type

Plan the layout of the file system – segment size

- calculate space requirements
  - issue **"du -sk"** for each directory to find the space occupied by each directory (including subdirectories)
  - build the sum of the individual spaces to find the total space occupied
  - add 4KB per shared file as filesystem overhead
  - add extra space for future software updates like security fixes
  - check that the required size does not exceed the maximum DCSS size
- calculate the page frame numbers for start and end address
  - a page is 4096 bytes in size
  - DCSS needs to start on a page boundary
  - start address should be the first page frame after the virtual guest storage of the largest guest
  - end address: add start address and size, round up to next page, -1

Christian Bornträger
IBM development lab Böblingen

Example – segment planning

- 5 guests 128MB, 1 guest 256MB, 4 guests 160MB
- /bin/, /sbin/, /usr/bin/, /usr/sbin/, /lib/, and /usr/lib/ are identified for sharing
- 1. querying the size:

```
# du -skc /bin/ /sbin/ /lib/ /usr/bin/ /usr/sbin/ /usr/lib
8619    /bin
11518   /sbin
16335   /lib
29934   /usr/bin
4687    /usr/sbin
148722  /usr/lib
219813  total
```

- 219813kb of files to share
- 2. getting the amount of files

```
# find /bin/ /sbin/ /lib/ /usr/bin/ /usr/sbin/ /usr/lib | wc -l
      8929
```

- 4kb * 8929 = 35716kb as additional overhead
- You need 219813kb + 35716 kb = 255529kb
- adding some space for updates ,300MB seems a reasonable size

Example – segment planning

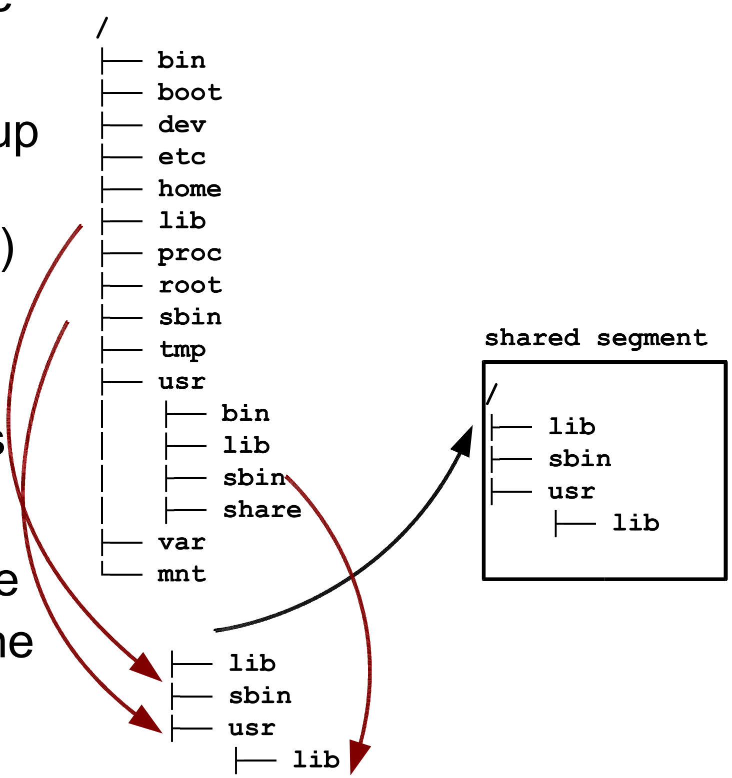
- The largest guest has 256MB of memory: largest address `0xfffffff`
- DCSS can start at 256MB ~ `0x10000000`
- considering 300MB of size: end address is 556 MB-1 ~ `0x22BFFFFF`
- size fits well between 256MB and 1960MB
- defseg command needs the address in pages (without the last 3 digits)

```
00: CP DEFSEG LINSEG1 10000-22BFF SR
00: HCPNSD440I Saved segment LINSEG1 was successfully defined in fileid 0203.
```

- now the segment is waiting to be filled
- next step is to prepare the linux guests

Provide a script to over-mount shared directories on startup

- over-mount in this context means directories are replaced by their shared copies in DCSS
- directories need to be replaced on system startup before services are being started (otherwise services and libraries will be loaded from DASD)
- the best way to do to this is running a script as initial process that mounts the DCSS, over-mounts all directories in 2 steps, and then starts the original **`/sbin/init`**
- note that **`/etc/mtab`** is not writeable at the time the script runs, therefore mount will not report the mounts later on. check **`/proc/mounts`** for a complete list of mounted filesystems
- the example script can be found in the execute-in-place Howto on IBM developerWorks



Christian Bornträger
IBM development lab Böblingen

Create a file system image (one time setup)

- get a DASD large enough to store your file system image with the size calculated earlier and prepare the dasd with a file system

```
# dasdfmt -b 4096 -d cdl -f /dev/dasdb
# fdasd -a /dev/dasdb
# mke2fs /dev/dasdb1
# mount /dev/dasdb1 /mnt
```

- create a file with the size of the planned DCSS on the newly mounted disk
- create an ext2 file system on the file

```
# dd if=/dev/zero of=/mnt/filesystem bs=1M count=300
# mke2fs -b 4096 /mnt/filesystem
/mnt/filesystem is not a block special device.
Proceed anyway? (y,n)
```

- answer the question with yes

Create a file system image

- create a mount point for the file system and mount it using the “-o loop” mount option

```
# mkdir /segment
# mount /mnt/filesystem /segment -o loop
```

- copy all directories you want to share into the file system image
- unmount the file system image

- a script that generates a file system image automatically can be found in the execute-in-place Howto on IBM developerWorks

Christian Bornträger
IBM development lab Böblingen

Fill a newly created DCSS using the image file

- prepare the disk containing the image file for IPL. Use the zipl command with the -s parameter along with the image file name and the DCSS start address:

```
zipl -t /mnt -s /mnt/filesystem,0x10000000
```

start address DCSS in bytes

- take down linux and IPL CMS in your guest machine
- if not already done, define the DCSS

```
00: CP DEFSEG LINSEG1 10000-22BFF SR
00: HCPNSD440I Saved segment LINSEG1 was successfully defined in fileid 0203.
```

- use "**`#cp define store <amount>`**" to define the virtual guest storage size large enough that the entire DCSS fits in
- IPL the DASD and wait for the CPU to enter disabled wait
- save the DCSS: "**`#cp saveseg <name of DCSS>`**"
- log off and back in to restore system defaults (storage)

Change the kernel parameter line

- when using the DCSS above all virtual guest storage, the kernel parameter line needs to be changed as follows:
- start up the Linux system
- add "`mem=<value>`" to the kernel parameter line, where <value> equals the end address of the DCSS
- run zipl with the new kernel parameter file
- reboot Linux
- issue "`cat /proc/cmdline`" to verify that Linux is using the new parameter

Christian Bornträger
IBM development lab Böblingen

Changing the storage configuration

- when using virtual guest storage above the 2GByte line, use CP storage configuration to create your memory setup:

```
#cp define storage config 0.256m 2g.1g
```

- result:

```
00: CP DEF STORE CONFIG 0.256M 2G.1G
00: STORAGE = 1280M
00: Storage Configuration:
00: 0.256M 2G.1G
00: Extent Specification                 Address Range
00: ----------- -----------------------------------
00: 0.256M     0000000000000000 - 000000000FFFFFFF
00: 2G.1G      0000000080000000 - 00000000BFFFFFFF
00: Storage cleared - system reset.
```


Christian Bornträger
IBM development lab Böblingen

Testing the DCSS

- if the Linux kernel has got the xip2 file system as a module, issue **`"modprobe xip2"`** to load the module into memory (optional)
- mount the xip2 file system using the mount command like **`"mount -t xip2 -o ro,memarea=<name> none <mount point>"`**
- if the segment is large, mounting might take a while
- verify that the file system has been mounted correctly by looking at **`/proc/mounts`**
- check with **`"dmesg"`**, if an error has occurred. If yes, then check:
  - mem parameter is >= end address of dcss
  - guest storage does not collide with dcss
  - no other loaded segment collides with the new segment (! segments need a megabyte boundary to other segments)
- verify that the files you copied to the file system image are accessible

Christian Bornträger
IBM development lab Böblingen

Activate execute-in-place

- unmount the file system again
- test the init script prepared earlier:
  - run the script as super user root
  - look at `/proc/mounts` to verify that
    - the file system has been mounted at the mount point created earlier
    - all directories that have been chosen earlier are over-mounted
- add “init=<full path of script>” to the kernel parameter line
- run zipl with the new kernel parameter line
- reboot Linux
- after reboot, check `/proc/mounts` to verify that
    - the file system has been mounted at its mount point
    - all chosen directories are over-mounted

Christian Bornträger
IBM development lab Böblingen

Introducing the DCSS block device driver

- DCSS block device driver is able to access a DCSS as a block device
- can be built-in or is available as a module called dcssblk.ko
- is controlled using files that are located in **`/sys/devices/dcssblk`**
- cannot be used to create new DCSSes
- motivation: XIP filesystem is readonly. DCSS block device driver can be used with ext2 to update DCSSes containing a filesystem image for XIP2:
  - the structure of a XIP filesystem equals the structure of ext2
  - ext2 needs block device to operate
- usable for other scenarios as well
- ext2 + block device driver does currently not implement execute in place!

Adding and removing DCSSes

```
echo LINSEG1 > /sys/devices/dcssblk/add
```

- add a DCSS as a block device
  - may take a while
  - driver will log a message informing about success/failure to syslog
  - a new subdirectory **/sys/devices/dcssblk/DCSSNAME/** appears

```
/sys/devices/
--- dcssblk
    --- add
    --- remove
```

```
/sys/devices/
--- dcssblk
    --- add
    --- remove
    --- LINSEG1/
        --- save
        --- shared
```

- remove the block device associated with a DCSS
  - driver will log a message informing about success/failure to syslog
  - subdirectory **/sys/devices/dcssblk/DCSSNAME/** disappears

```
echo LINSEG1 > /sys/devices/dcssblk/remove
```


Christian Bornträger
IBM development lab Böblingen

The “shared” state

- Every DCSS is by default in shared state.
  - the file /sys/devices/dcssblk/DCSSNAME/shared contains the value “1”
  - Linux uses the global copy of the DCSS
  - can be switched to non-shared mode by writing “0” to /sys/devices/dcssblk/DCSSNAME/shared when idle
- non-shared DCSSes:
  - the file /sys/devices/dcssblk/DCSSNAME/shared contains the value “0”
  - Linux uses a private copy of the DCSS
  - can be switched back to shared mode by writing “1” to /sys/devices/dcssblk/DCSSNAME/shared

The "save" state

- Linux can only save DCSSes when they are idle
- Linux remembers save requests while the DCSS is busy and saves the DCSS as soon as the DCSS becomes idle
- by default the save state is "0" meaning that the DCSS will not be saved
- by writing the value "1" to /sys/devices/dcssblk/DCSSNAME/save:
  - Linux is triggered save the DCSS immediately in case it is idle
  - Linux remembers to save the DCSS when it becomes idle
- you need Class E to write a segment!
- by writing the value "0" to /proc/dcssblk/DCSSNAME/save
  - a pending save on a DCSS that is in use can be canceled
- Linux reports all above operations in the syslog and the value in /sys/devices/dcssblk/DCSSNAME/save can be read to check if there is a pending request to save a DCSS

using the block-device node

- no partitioning
  - do not use fdasd or fdisk
- any block device and filesystem tool can be used with the DCSS block device (like mke2fs, e2fsck, etc.)
- 4096 bytes block size
  - use -b 4096 for mke2fs or other tools depending on the block size
- any file system that supports 4096 bytes block size can be used with the DCSS block device driver (like ext2, ext3, ReiserFS, GFS)
- only ext2 is compatible with xip2 and can be used to process DCSSes intended to be used with the xip2 file system
- can be used as well for doing the first time setup. Just create a segment filled with random garbage and update this segment: use defseg + saveseg to create an active segment

DCSSes as swap space

- why use a dcss for swapping?
  - fast write into z/VMs storage and swap caching when guest is memory constrained but z/VM is not
  - allows to shrink guest virtual memory size while maintaining acceptable performance for peak workloads (move overcommitment to guest level)
  - can be much faster than vdisk
  - no hypervisor calls required
- what is required?
  - the block device kernel parameter and support for mixed EW/EN segments – both included in upcoming SLES9 SP2
  - a lot of paging space assigned to z/VM
  - address space for using dcss (same as with execute in place)

DCSSes as swap space

- create an empty DCSS in CMS, first page should be EW, the rest EN

```
#cp defseg SWAPPING 20000-20000 EW 20001-6ffff EN
#cp saveseg SWAPPING
```

- in Linux initialize the segment (one time setup only!)

```
echo "SWAPPING" >/sys/devices/dcssblk/add
mkswap /dev/dcssblk0
echo 1 >/sys/devices/dcssblk/SWAPPING/save
swapon /dev/dcssblk0
```

After above setup, syslog should indicate that the segment SWAPPING was saved. In addition, /proc/swaps should mention the segment as active swap space.

DCSSes as swap space

- in order to activate swap, you need to add the following kernel parameter and re-run zipl:

```
dcssblk.segments=SWAPPING
```

- add this line to your /etc/fstab file:

```
/dev/dcssblk0        swap sw 0 0
```

After rebooting the system swap should now be activated. You can check /proc/swaps to verify this.

- In case of a mixed swap setup with both DCSS and DASD as swap target, DCSS needs to get a higher swap priority than DASD for optimal performance.

Christian Bornträger
IBM development lab Böblingen

Outlook

- SLES9 is GA
  - SLES9 includes execute-in-place
  - all new file system features like Access Control Lists
  - device node creation is made automatically by udev
- SLES9 SP2 soon
  - adds support for EW/EN mixed mode segments (swapping)
  - provides much better system messages for dcss (cleanup)
  - adds support for the dcssblk.segments parameter (swapping, root dev)
  - new “Howto” documentation on developerworks
- integration into the ext2 file system
- union mount for overlaying DCSS with normal file system
- better management and update solutions

Questions and Discussion

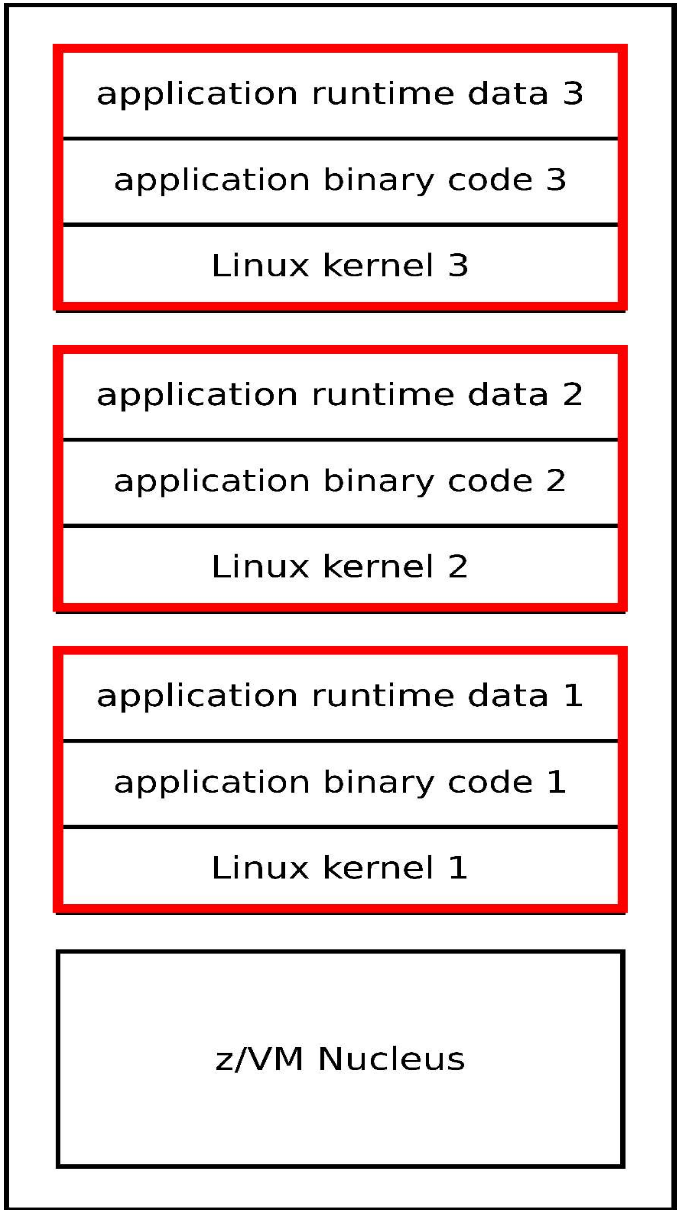
- Now
  - After this session
  - Any time during WAVV
  - Email:
    - cborntra@de.ibm.com
    - cotte@de.ibm.com
- Thank you for your attention

Background

technical background

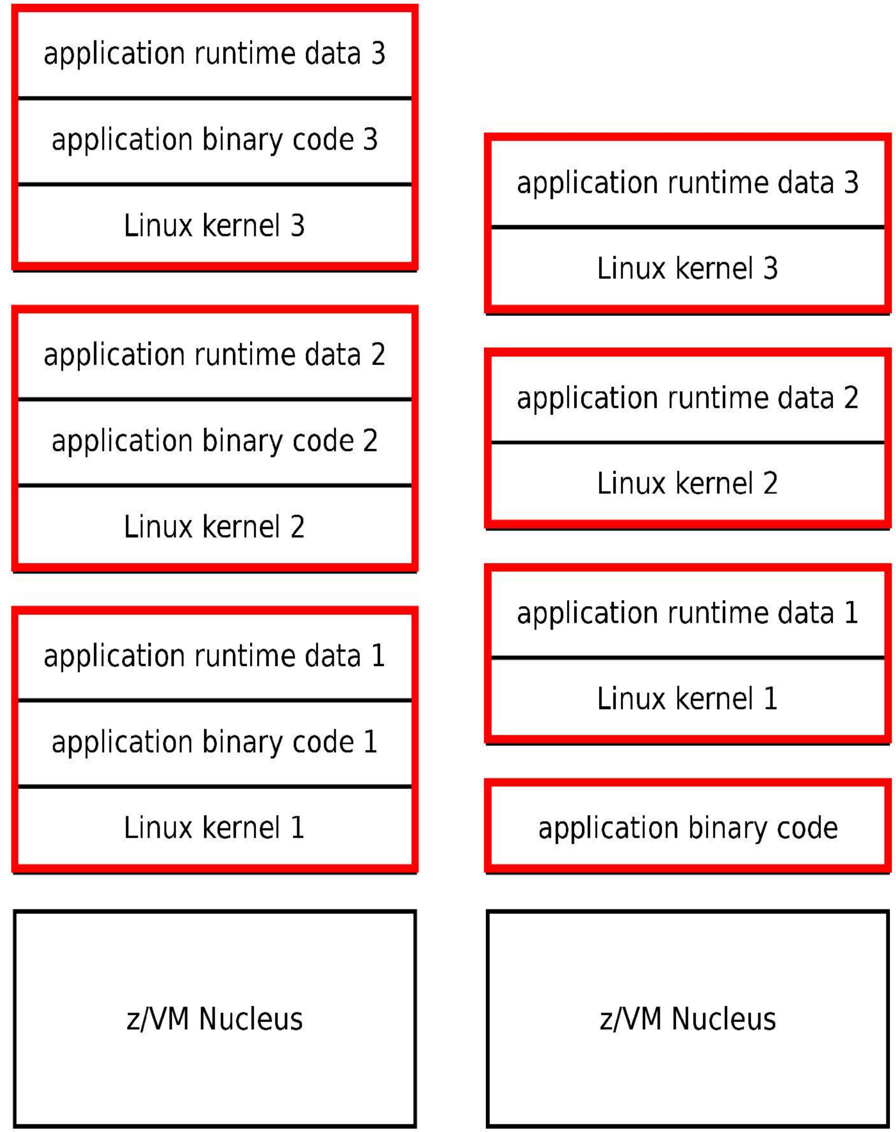
Illustration of the problem

- Linux running as guest operating system on z/VM requires a similar amount of memory to Linux on a dedicated server while it is active
- when running multiple Linux servers that run the same application, the application is present multiple times in physical memory
- storing the same data multiple times in memory is inefficient



Basic idea of the solution

- store only one copy of the application in memory
- share access to the application among multiple servers
- let z/VM load the application into memory
- restrict write access to the application for servers (security!)
- allows running the same workload with less memory consumption
- allows to run more workload with the same memory consumption as before



What is “execute in place”?

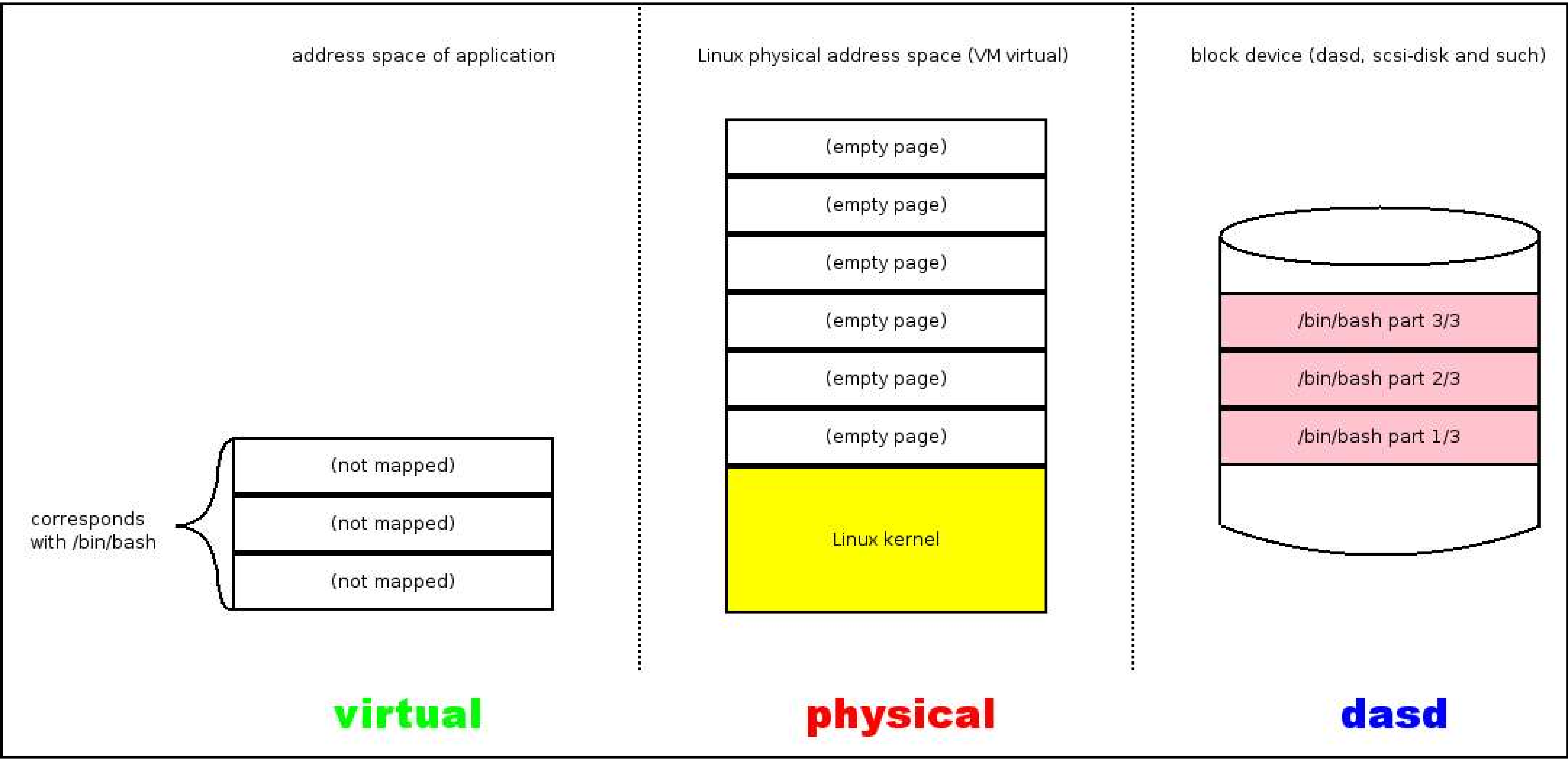
- applications are being executed directly from where they are permanently stored
- was invented for embedded systems that do not have disk drives
- applications can be run directly in flash or ROM memory
- execute in place is a popular feature of embedded operating systems such as uCLinux

Christian Bornträger
IBM development lab Böblingen

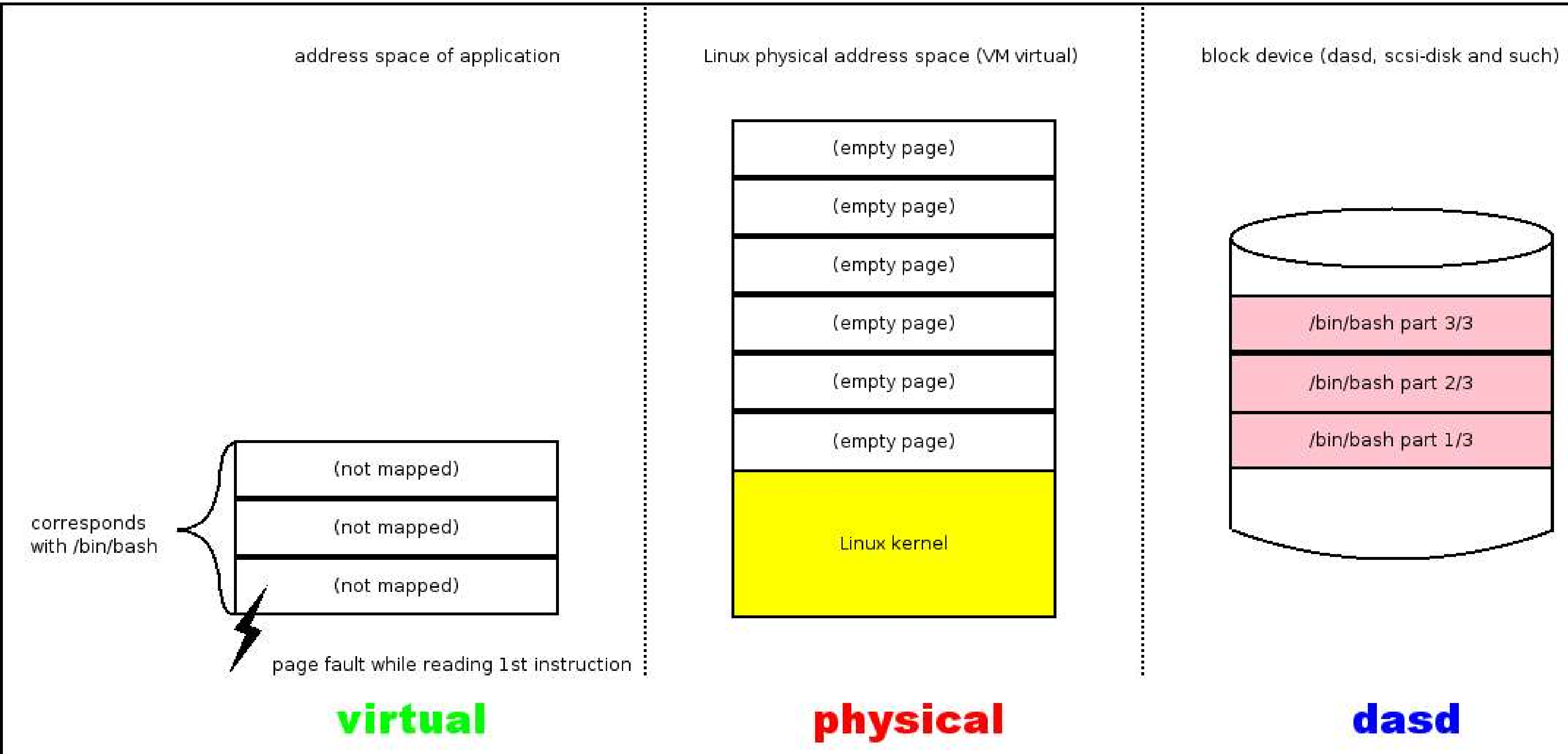
How applications are launched in Linux

- when Linux launches an application, it does not load it into memory immediately
- for application binary files and libraries, Linux:
  - remembers the position in the application address space
  - remembers the corresponding file on disk
- when the application accesses a page in memory that has not been loaded yet:
  - Linux interrupts the application
  - the page is loaded into memory
  - the application is resumed, and it retries to access the page

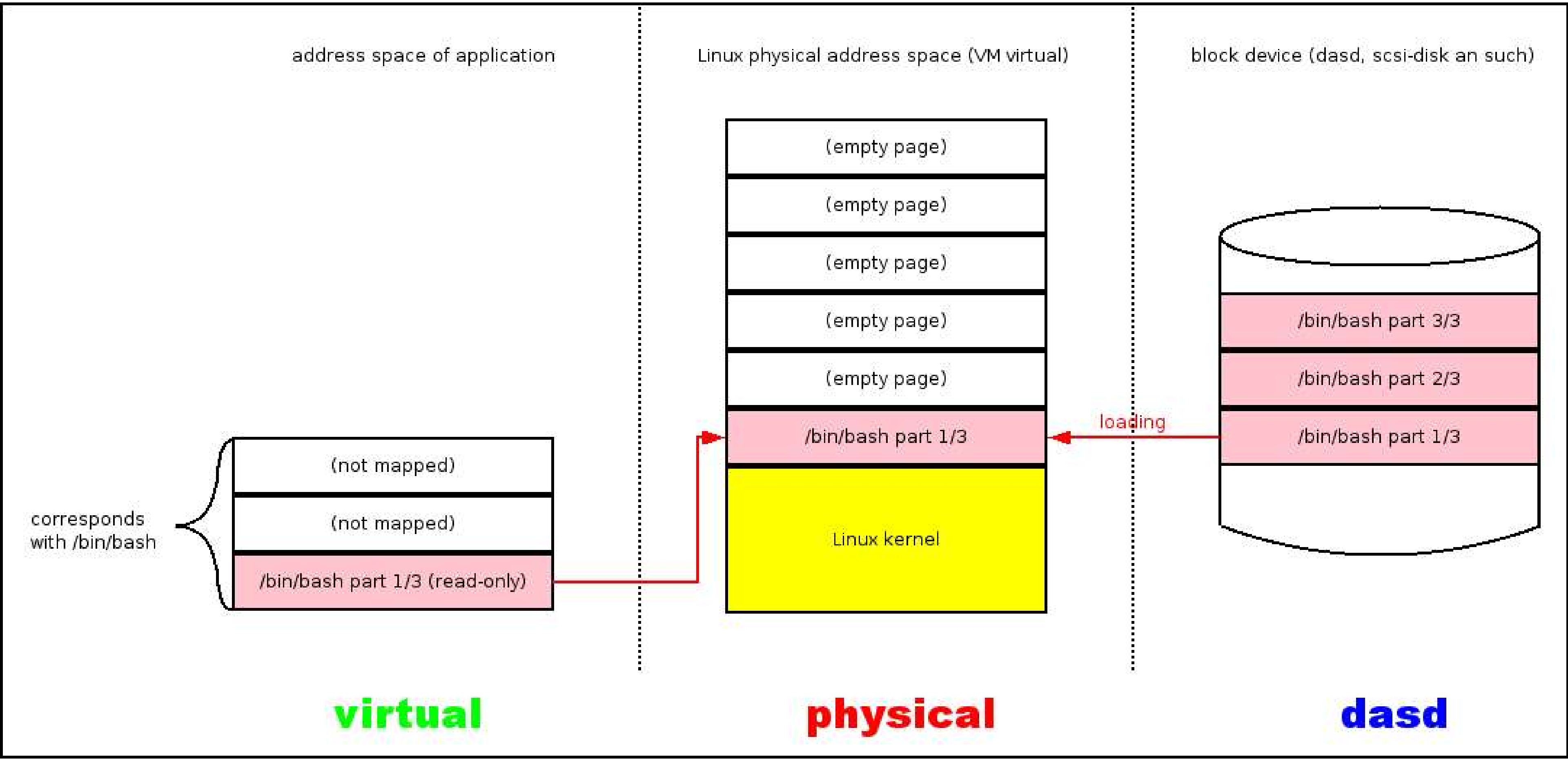
How applications are launched in Linux



How applications are launched in Linux

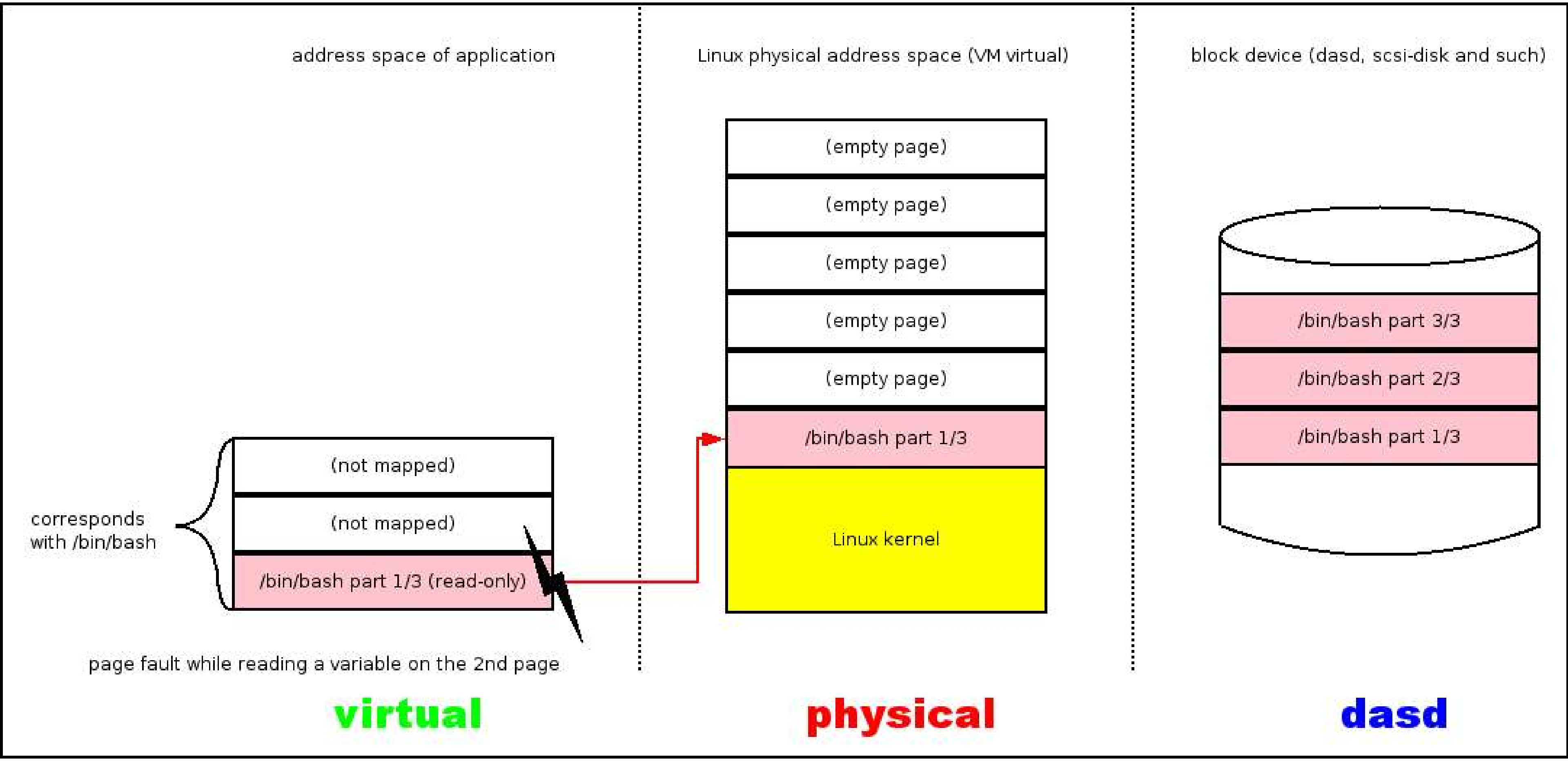


How applications are launched in Linux



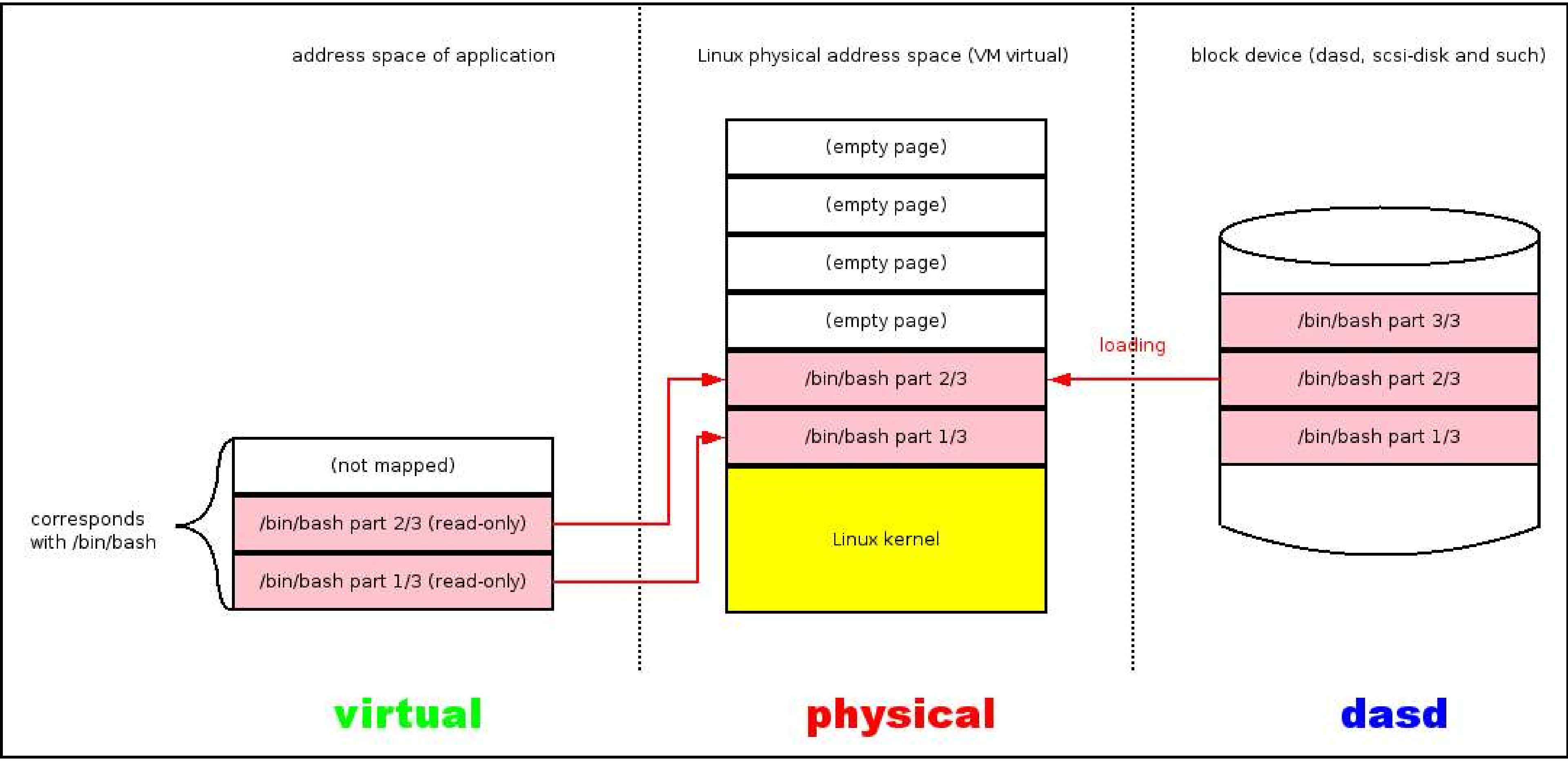
Christian Bornträger
IBM development lab Böblingen

How applications are launched in Linux

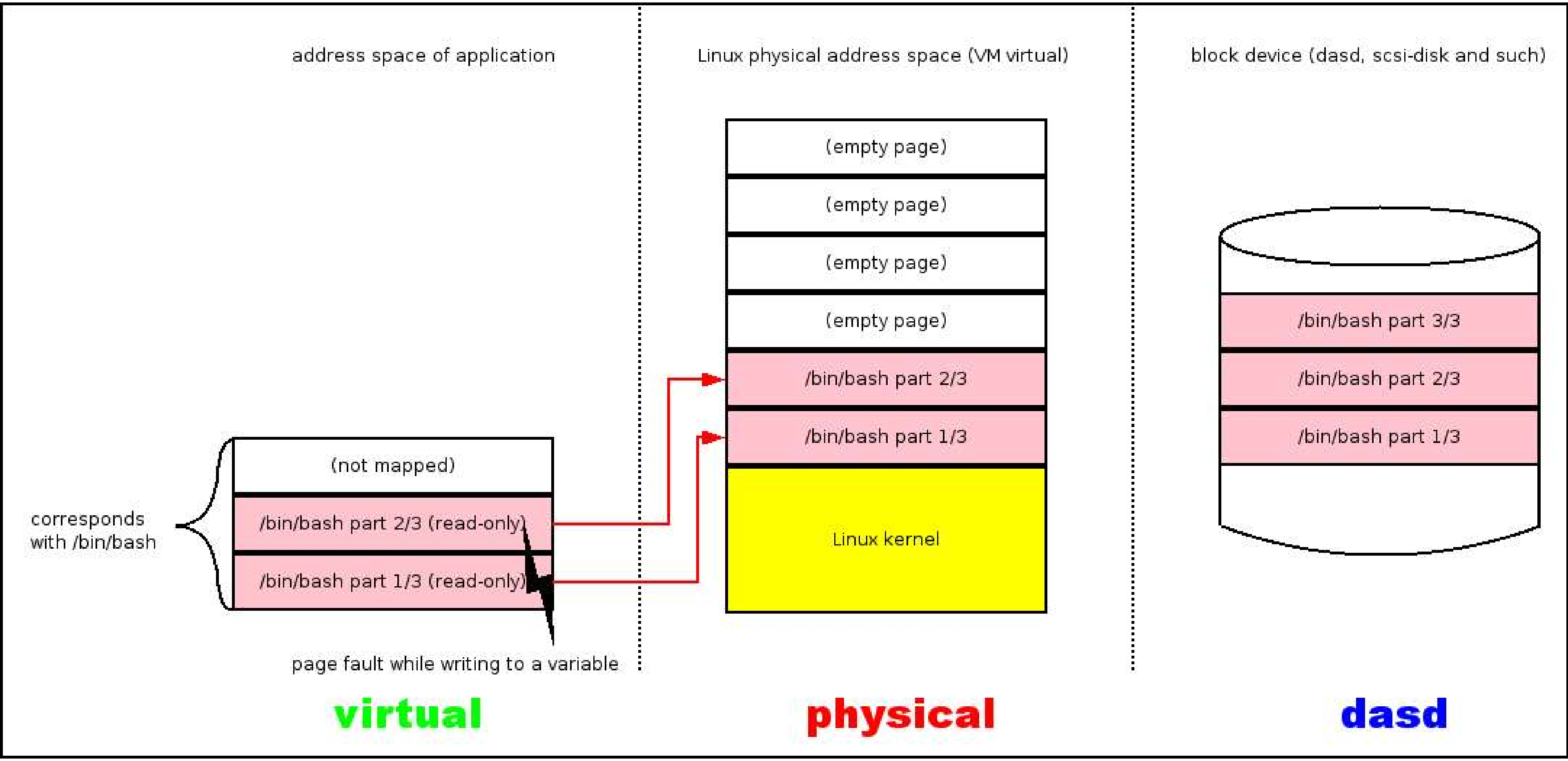


Christian Bornträger
IBM development lab Böblingen

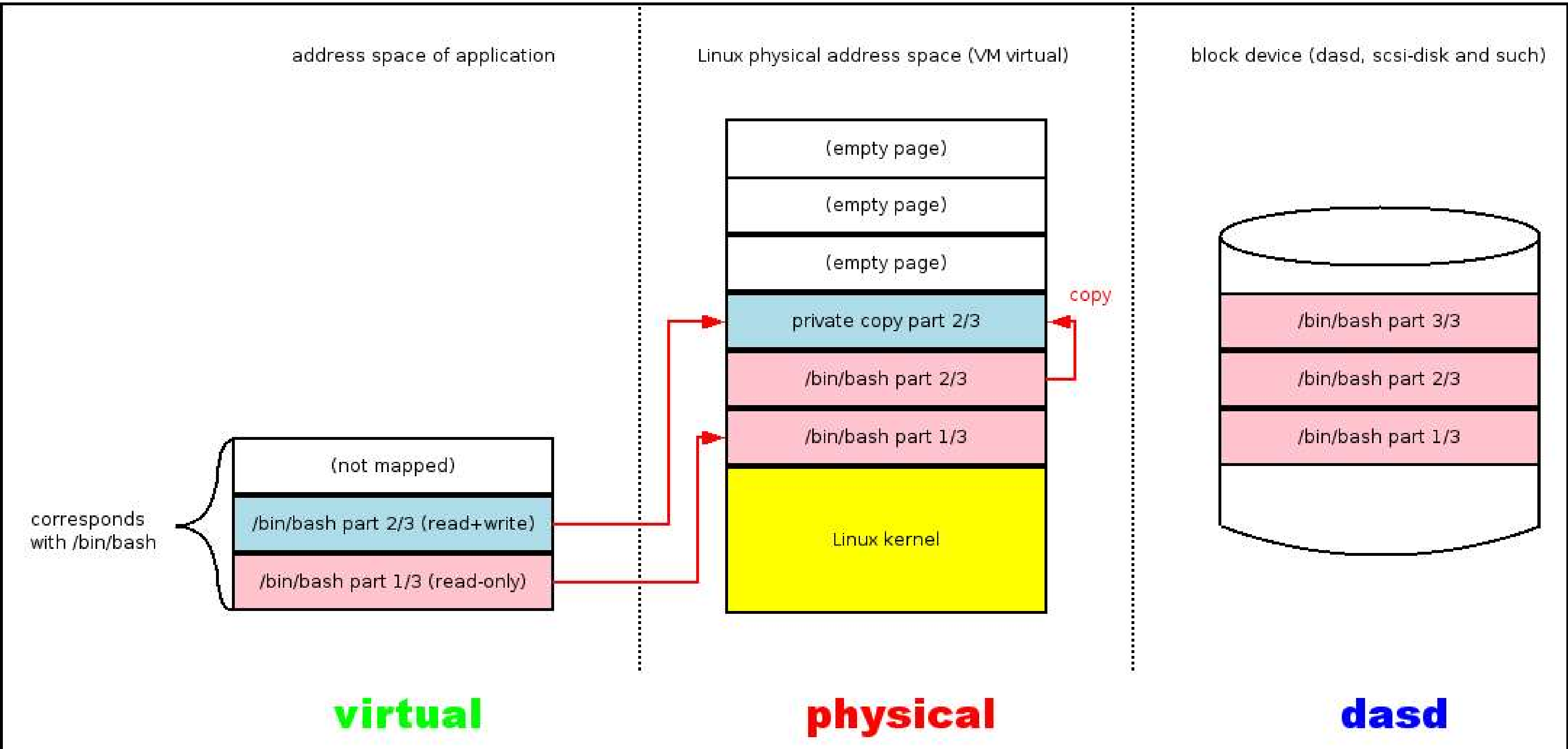
How applications are launched in Linux



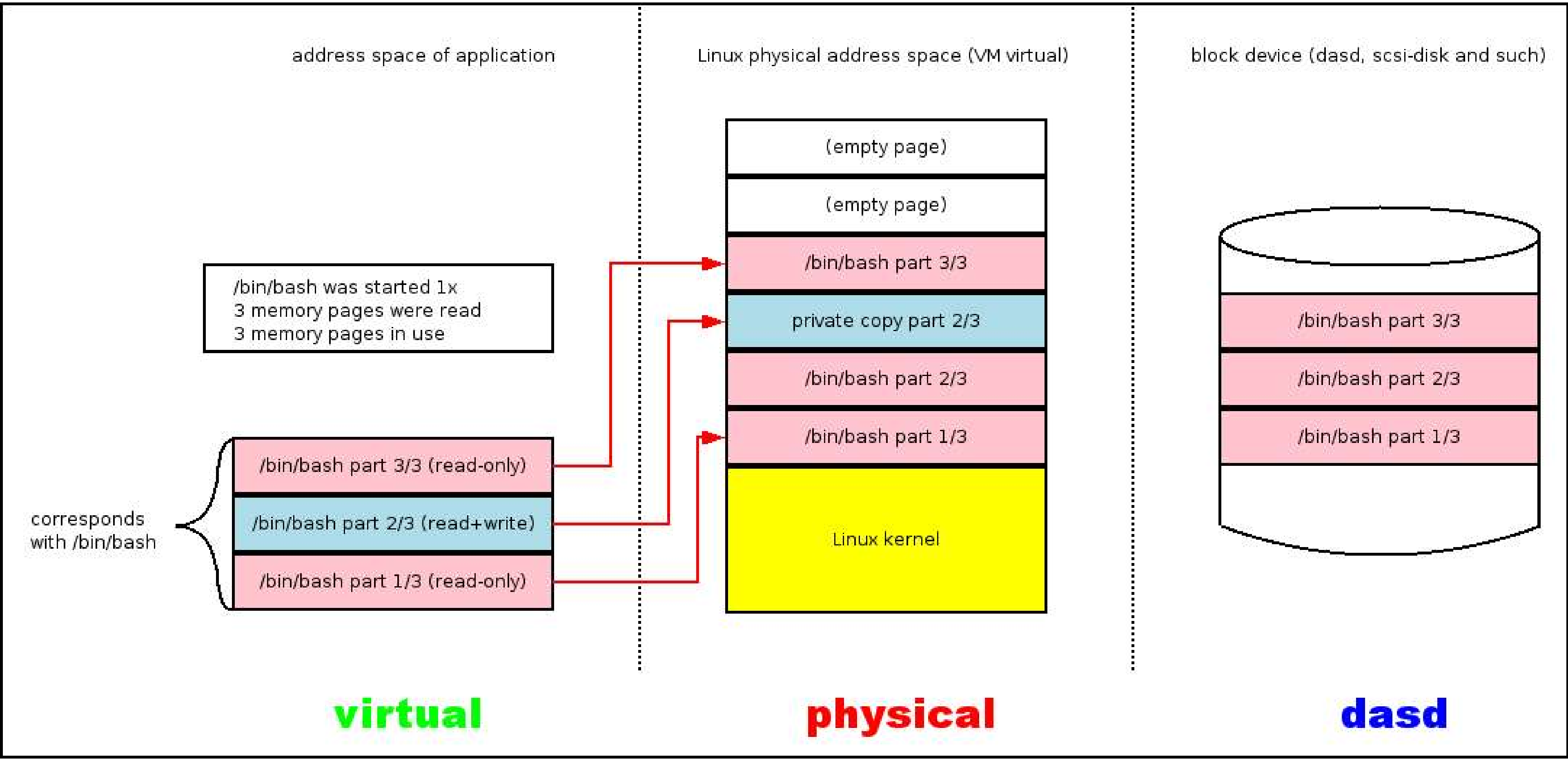
How applications are launched in Linux



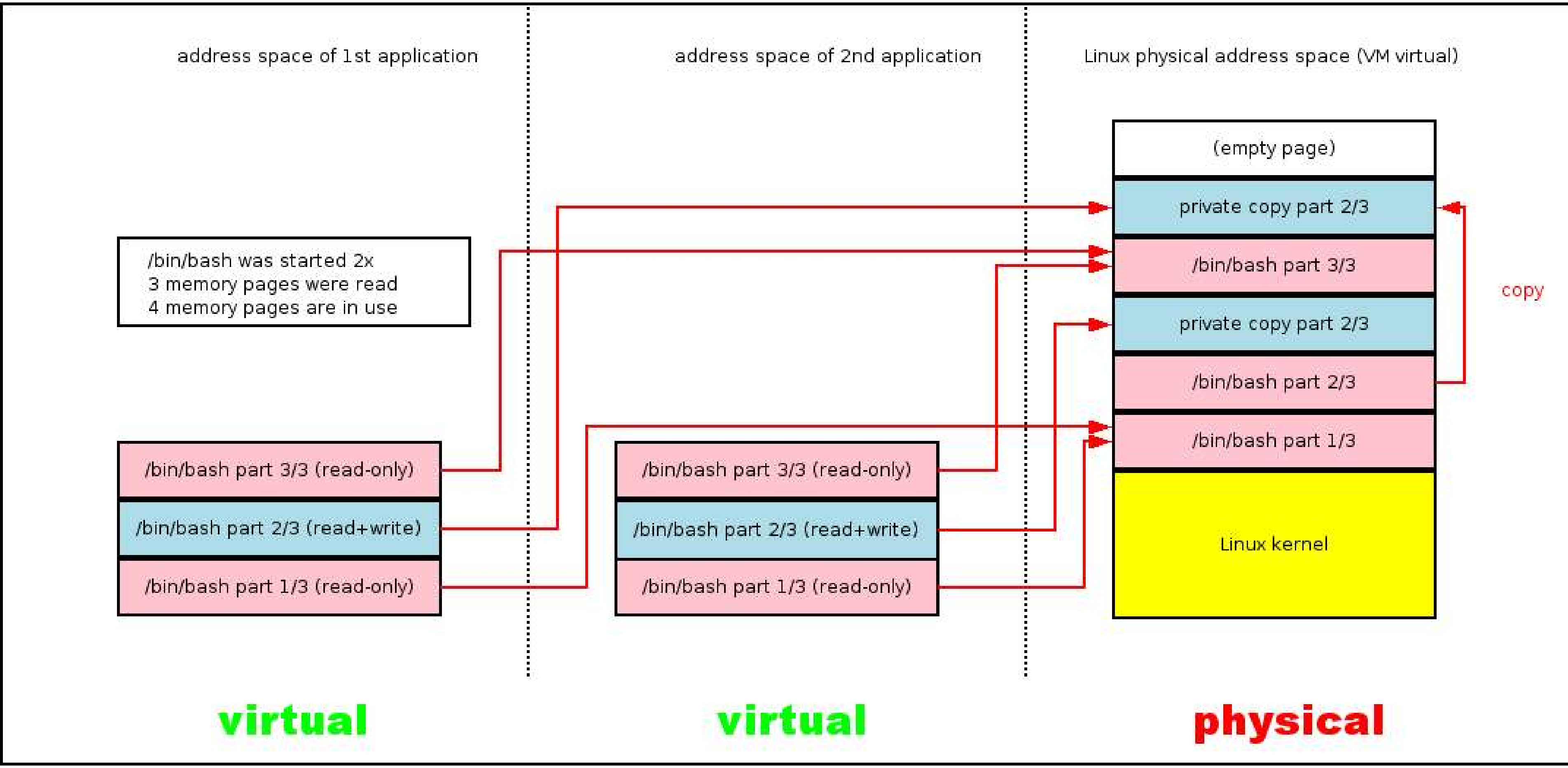
How applications are launched in Linux



How applications are launched in Linux



How applications are launched in Linux

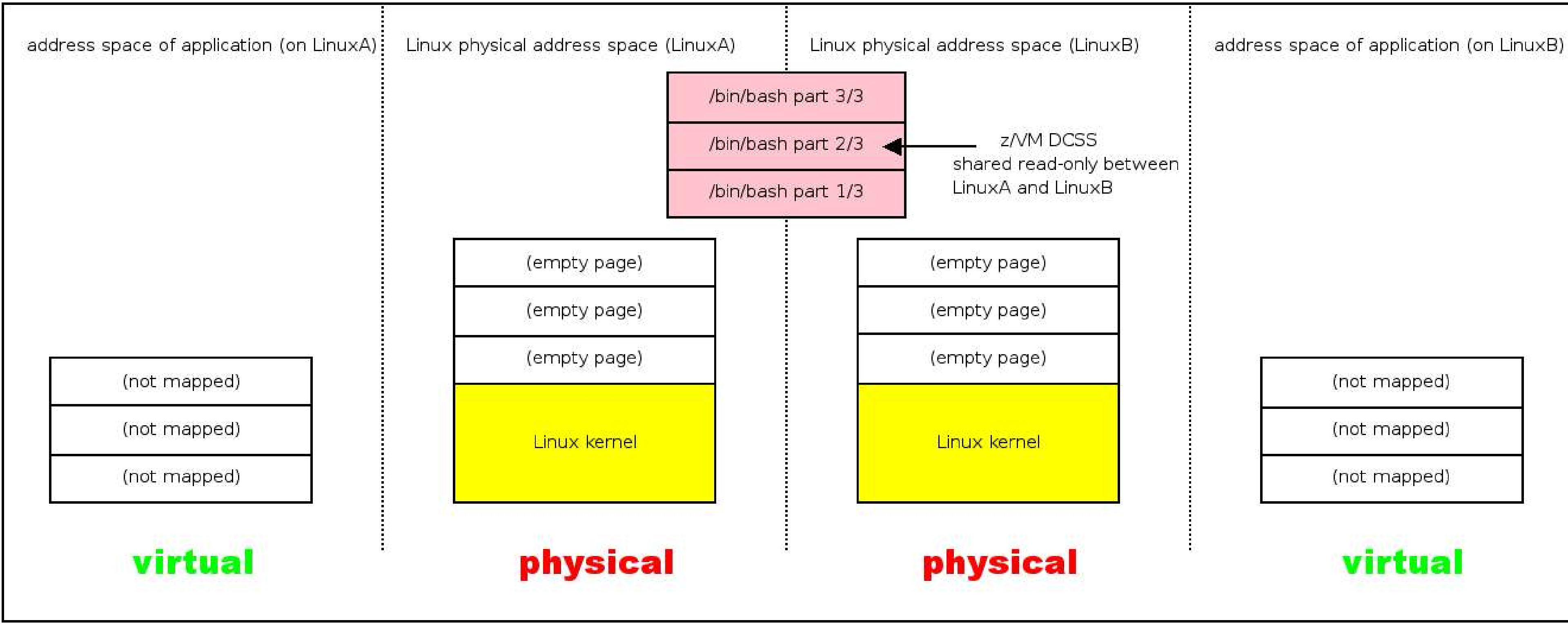


Christian Bornträger
IBM development lab Böblingen

Goals for “execute in place” with Linux on z/VM

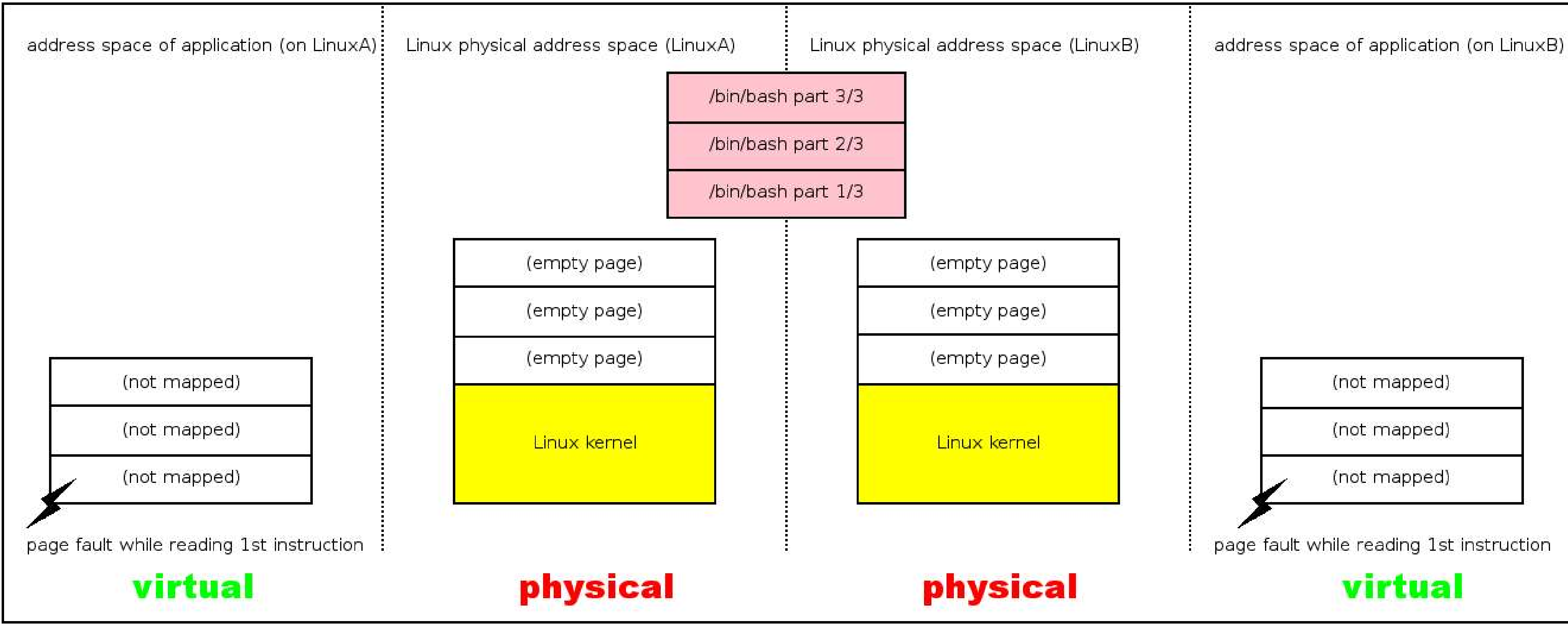
- improve resource usage in a virtual environment
- reduce memory requirements
- reduce I/O bandwidth requirements
- keep individual servers separated for security reasons

How “execute in place” works

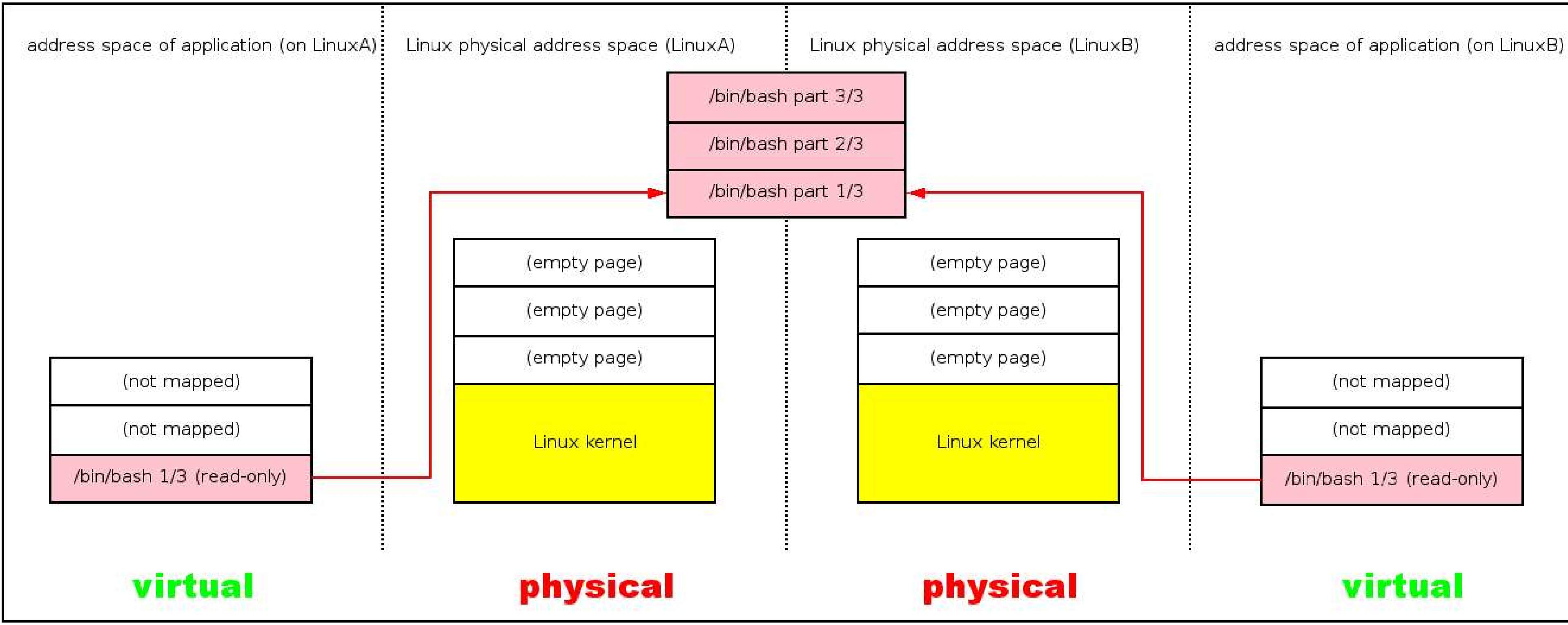


Christian Bornträger
IBM development lab Böblingen

How "execute in place" works

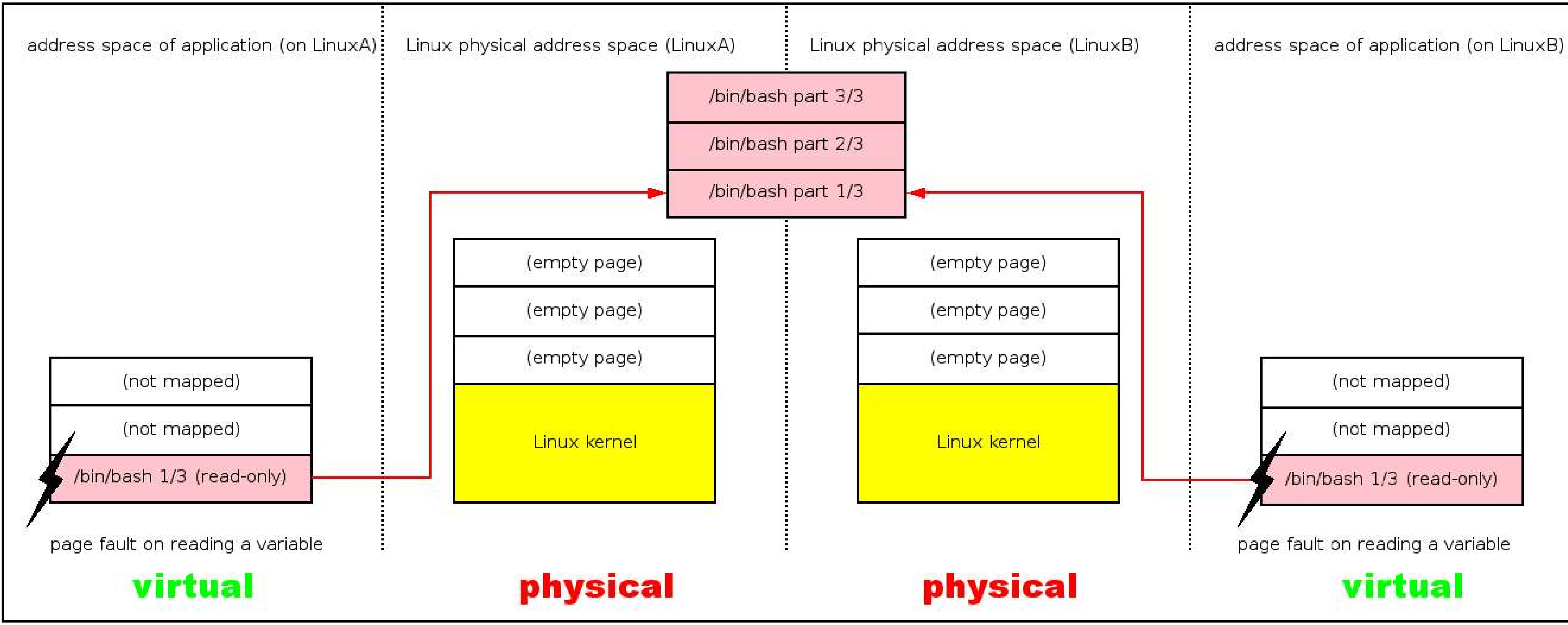


How “execute in place” works

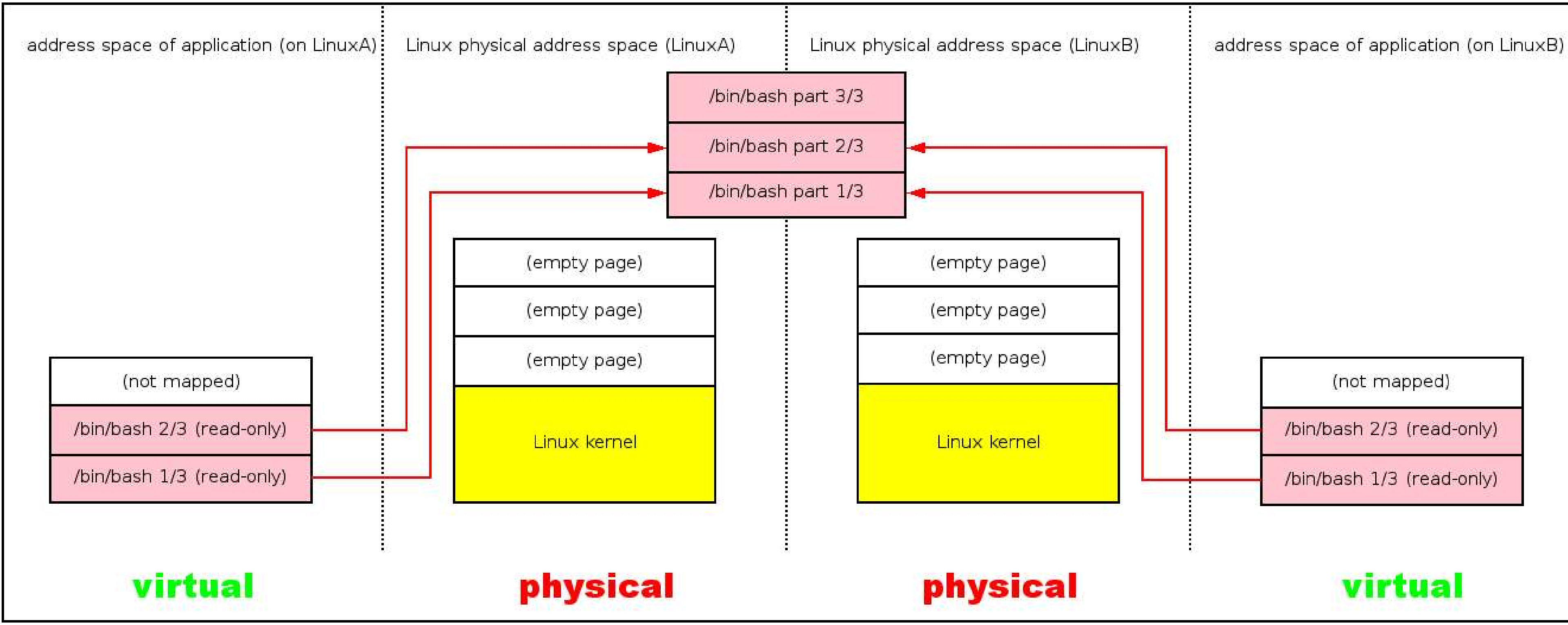


Christian Bornträger
IBM development lab Böblingen

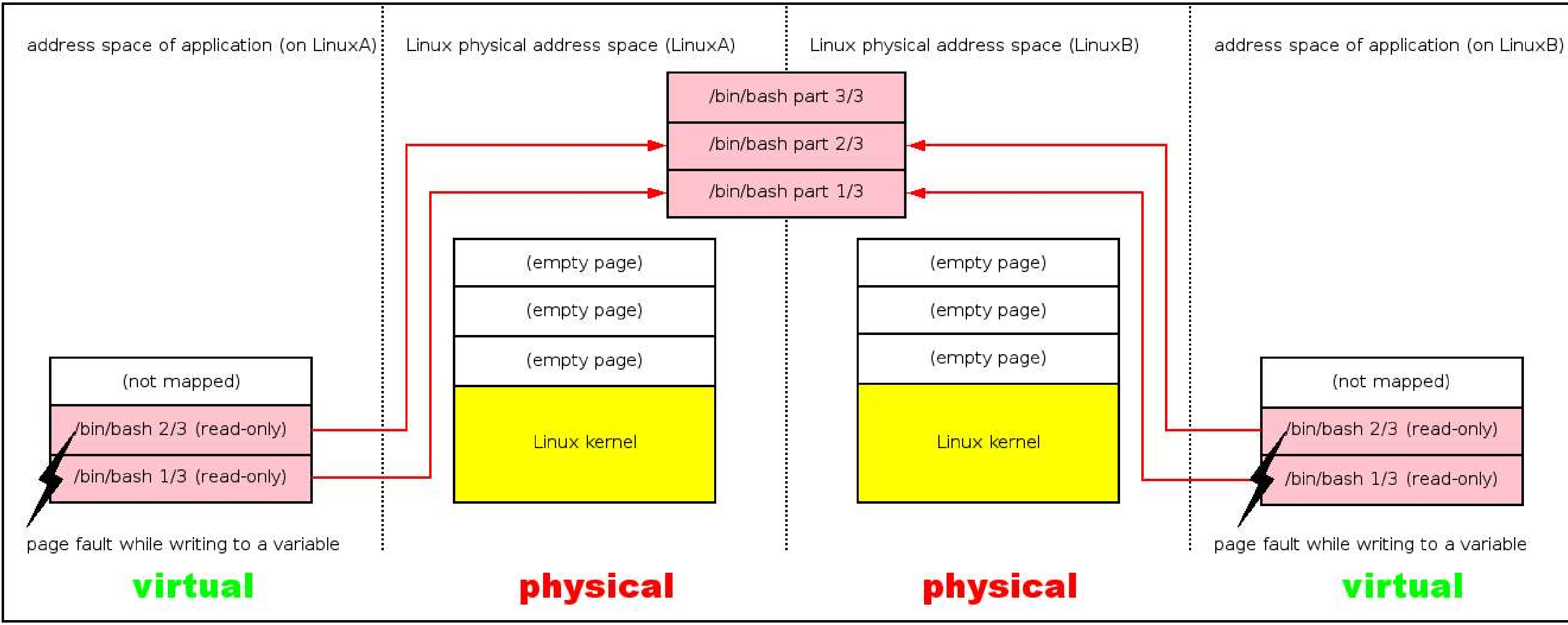
How “execute in place” works



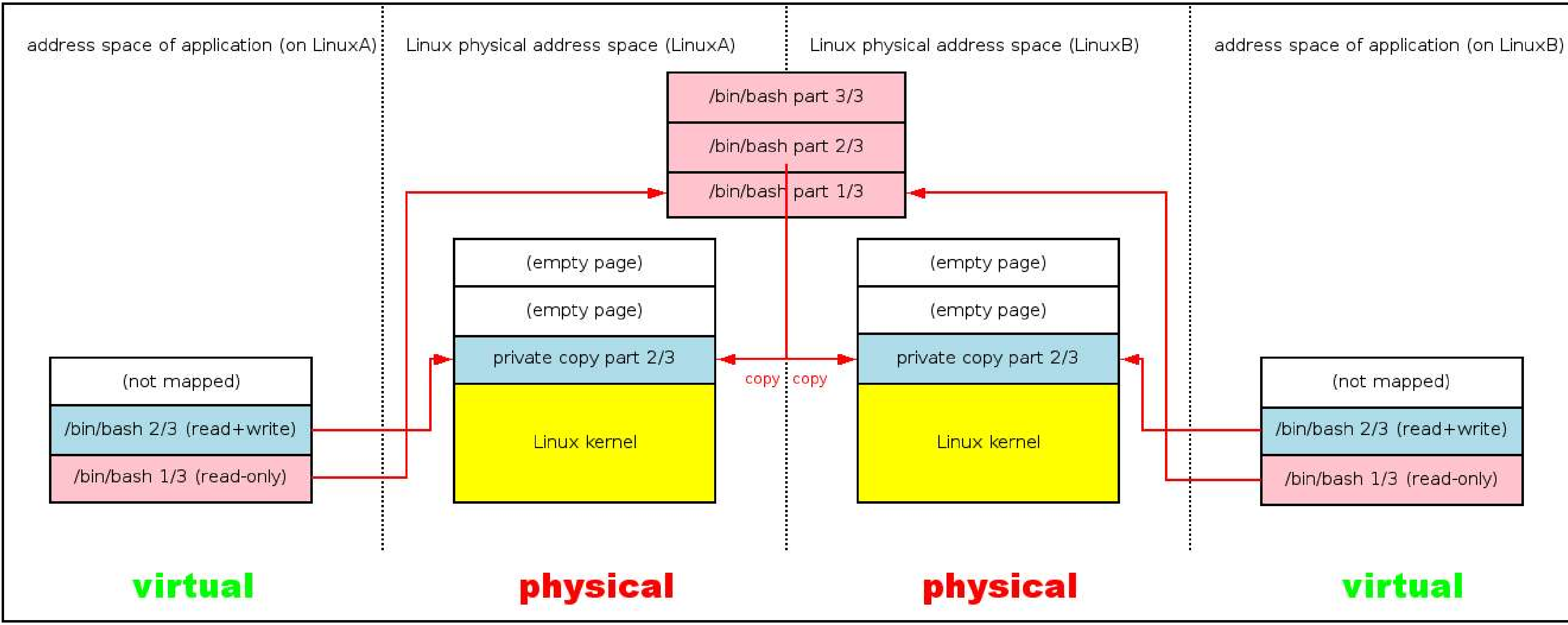
How “execute in place” works



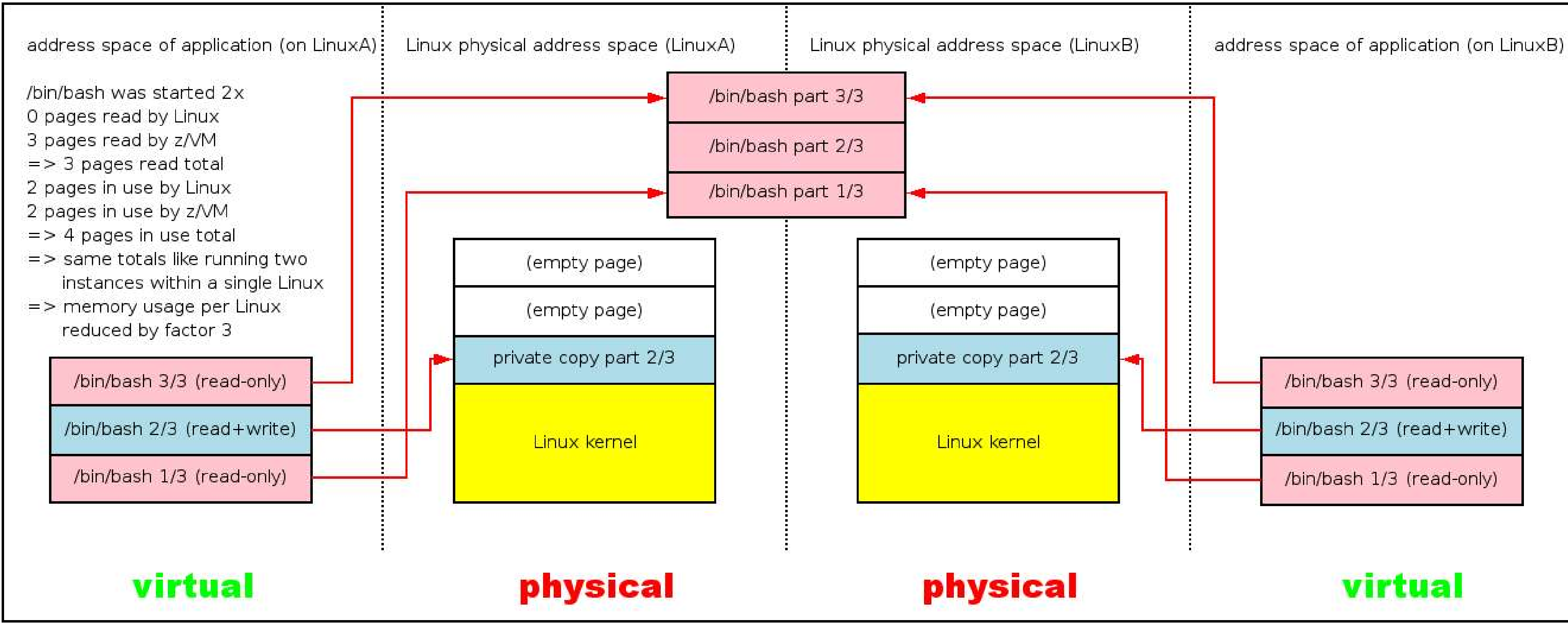
How “execute in place” works



How “execute in place” works



How “execute in place” works



Characteristics of Linux on z/VM with execute in place

- implemented as filesystem type “xip2”
- can only be used read-only, writing is not supported
- the filesystem layout is compatible with ext2
- requires z/VM DCSS as storage
- does not need write access to the DCSS, individual servers remain strictly separated in memory (security!)
- the DCSS block device driver can be used with ext2 to write data to the DCSS

Characteristics of Linux on z/VM with execute in place

- no changes to the operating system outside the filesystem module
- no changes to applications, existing applications can benefit without modifications
- provides major improvement for memory consumption of servers
- allows to run more active servers with given resources
- source code is available at http://www.linuxvm.org/Patches
- is available as update to SuSE Linux Enterprise Server Version 8
- is integrated in SuSE Linux Enterprise Server Version 9
  - new ext2 features supported (dirindex, acl...)
  - remove memory restriction to 2GByte