

The background of the slide features a dark, moody image of server racks in a data center. The racks are illuminated with a warm, reddish-orange light, creating a sense of depth and technology. The Virtuozzo logo is positioned in the top left corner, with a red 'z' and the word 'Virtuozzo' in white.

Virtuozzo

Painless Live Userspace Patching

<https://github.com/skinsbursky/presentations/blob/master/lup-devconf.odp>

Stanislav Kinsburskiy

Summary

- What is “Live Patching”?
- Why it's needed?
- Types of binary code
- Process state
- What can be patched?
- Binary patch creation
- When binary patch can be applied?
- Where to place new code?
- Userspace live patching in pictures
- Why is it called “painless”?

What is “Live Patching”?

- Change a piece of code in a process
- Preserve process state
- Do it safe



Why it's needed?

- Get rid of heavy services restart
- Reduce service downtime in case of critical vulnerabilities, because of:
 - No need in application restart
 - No need in migration



Types of binary code

- Statically-linked
- Dynamically-linked
 - Load-time relocation
 - Position independent code (PIC)

01100111 11111100 01111101 01111101 11011001 11001010 11101000 10011110 11101111 10100000 10010111 00100001 00010111 01000011 00011100
11100010 10011100 01100010 01011111 11010011 10001100 10001101 01110101 10010000 01011011 01110000 10111110 10110010 10110101 10011001
11001011 01001001 11100010 01011011 11000101 10001011 01111001 00111001 11001011 01001101 00000010 11100100 00011010 10010010 10000010
00100011 00011111 00001000 01011011 01001111 10100101 01111001 00111001 11001011 01001101 10000010 10000010 10000010 10000010 10000010
10111001 01010010 01100011 00000100 00001011 10011101 00011001 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
00100011 11010010 10000111 00011000 10011101 01111001 00011001 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
01000101 01010001 00100101 11110001 00110111 00100111 00011001 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
11001010 11101000 10011110 11101111 10100000 10011101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10001100 10001101 01110101 10010000 01011011 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10001011 01000011 01000111 00011110 01001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10100101 01111001 00111001 11001011 01100111 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10011000 00101000 00010110 00111010 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
01111011 01011010 10001100 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010
00100100 11010110 10101100 01011111 00011111 00001000 01011011 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10010111 00100001 00010111 01000011 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100
01110000 10111110 10110010 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010
00000010 11100100 00011010 10010010 10000000 10011101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
010001101110111 10010011 10000000 10011101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10011001 11010100 10001101 01100011 11010010 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
11100001 10100100 01001000 00110010 00000101 01010001 00100101 11110001 00110011 00000100 10011010 10101100 10101100 11101100
01100111 11111100 01111101 01111101 00000101 01010010 11101000 10011110 11101111 10100000 01011011 10010000 01011011 10010000 01011011
11100010 10011100 01100010 01011111 00000101 01010010 11101000 10011110 11101111 10100000 01011011 10010000 01011011 10010000 01011011
11001011 01001001 11100010 01011111 00000101 01010010 11101000 10011110 11101111 10100000 01011011 10010000 01011011 10010000 01011011
00100011 00011111 00001000 01011101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10111001 01010010 01100011 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100
00100011 11010010 10000111 00011000 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
01000101 01010001 00100101 11110000 00111001 11001011 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
11001010 11101000 10011110 11101111 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10001100 10001101 01110101 10010000 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100
10001011 01000011 01000111 00011110 01011011 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10100101 01111001 00111001 11001011 01100111 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10011100 00101000 00010110 00111010 10000010 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
01111011 01011010 10001100 10101010 11101100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100
00100100 11010110 10101100 01011111 00111001 01100111 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
10010111 00100001 00010111 01000011 00011100 11100011 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
01110000 10111110 10110010 10110101 10011001 11001011 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
00000010 11100100 00011010 10010010 10000010 00100011 00011111 10001101 10001101 10001101 10001101 10001101 10001101 10001101 10001101
010001101110111 10011001 10000000 01100110 10111001 01010010 01110001 00111001 01010010 01110001 00111001 01010010 01110001 00111001 01010010
10011001 11010100 10001101 01100111 00000101 01010010 00100011 11010010 10000011 00011000 10011101 01110111 01011010 10001100 10101010 11101100
11100001 10100100 01001000 00110111 00100001 01000101 01010001 00100101 11110001 00110111 00100100 11010110 10101100 01011111 00111001

Process state

- Process state is like a cards castle:
 - strongly associated
 - accurately verified
- Contains of:
 - statically allocated variables
 - dynamically allocated variables
 - stack content
- Is changing during runtime



What can be patched?

- Not any binary patch can be applied
- Fundamental Limitations:
 - Static data of different size
 - Dynamically allocated objects of different size
- Source code review is required :(



Binary patch creation

- Information about binary is required
- Patch code on function basis:
 - Reliable
 - Relatively simple
- Information about symbols required:
 - Name
 - Size
 - Address



When binary patch can be applied?

- Process external stop/resume is essential
- Not at any moment of time:
 - Process can execute the code to patch
 - Code to patch can be referenced in the call stack
- Stack unwinding is essential:
 - Need to catch the process outside old code
 - Or at least outside functions to patch

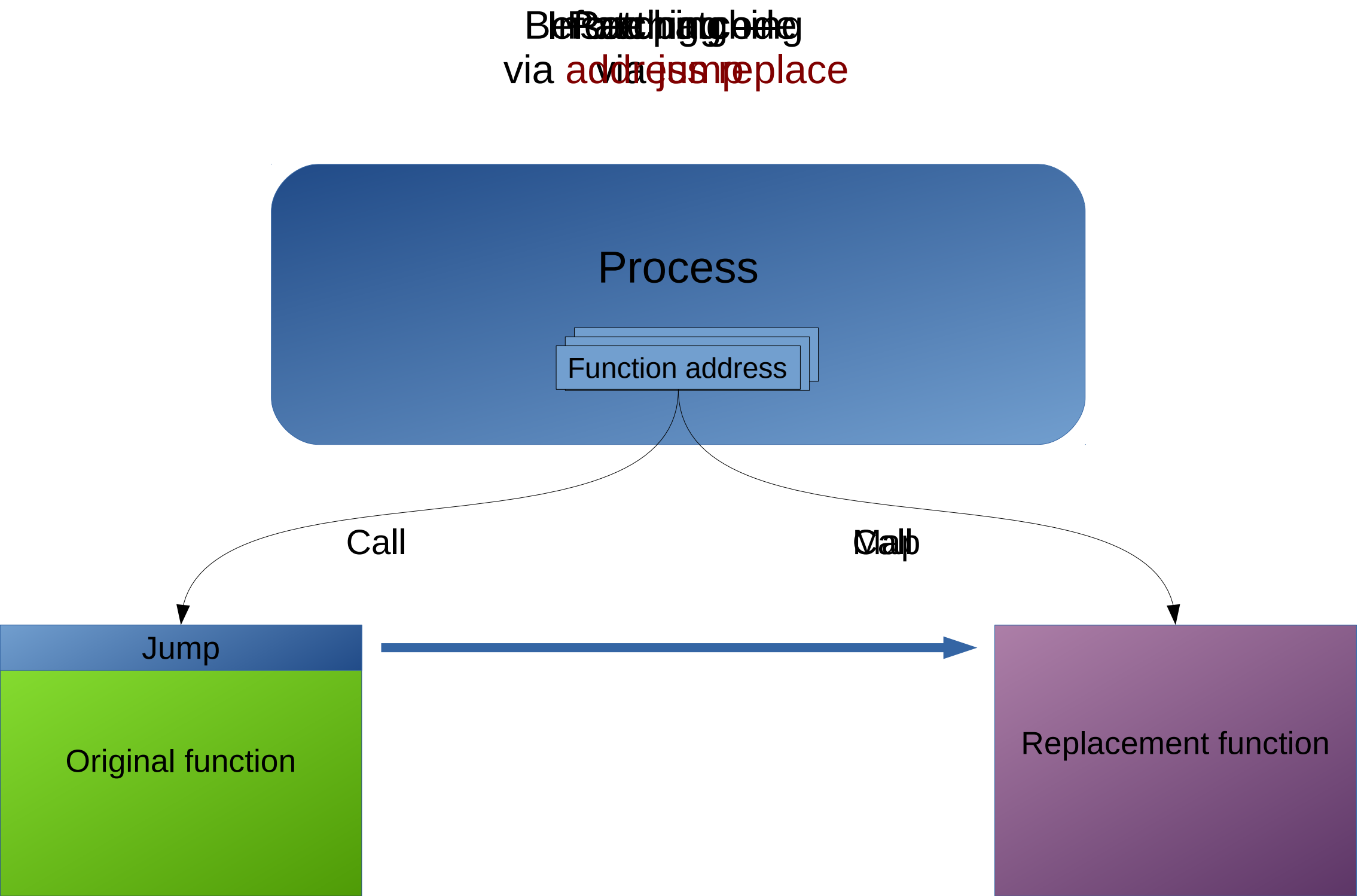


Where to place new code?

- Has to be placed into process address space:
 - Either as binary blob
 - Or as a file mapping
- Static code has to be “fixed” in place
- PIC code can be mapped as is, but:
 - Initialization required: external and global symbols
 - Corresponding data has to be copied
- Need to redirect execution to the new code



Userspace live patching in pictures



Why is it called “painless”?

- Live patching can't be painless :)
- No kernel changes!
- Binary patch creation can be based on ELF parsing
- Libcompel can be used for:
 - Task stop
 - Task resume
 - Code insertion
 - Source: <https://github.com/xemul/criu/tree/criu-dev>
- Libunwind can be used for stack unwinding
 - Source: <http://git.savannah.gnu.org/cgit/libunwind.git>



The background of the slide is a photograph of a server room aisle, viewed from a low angle looking down the center. The aisle is flanked by tall server racks on both sides. The floor is covered with a metal grating. The ceiling has recessed lighting. A semi-transparent red overlay covers the right half of the image, while the left half is dark. The text is overlaid on the dark left side.

Virtuozzo

Thanks for your
attention.
Q&A.

<https://github.com/skinsbursky/presentations/blob/master/lup-devconf.odp>
