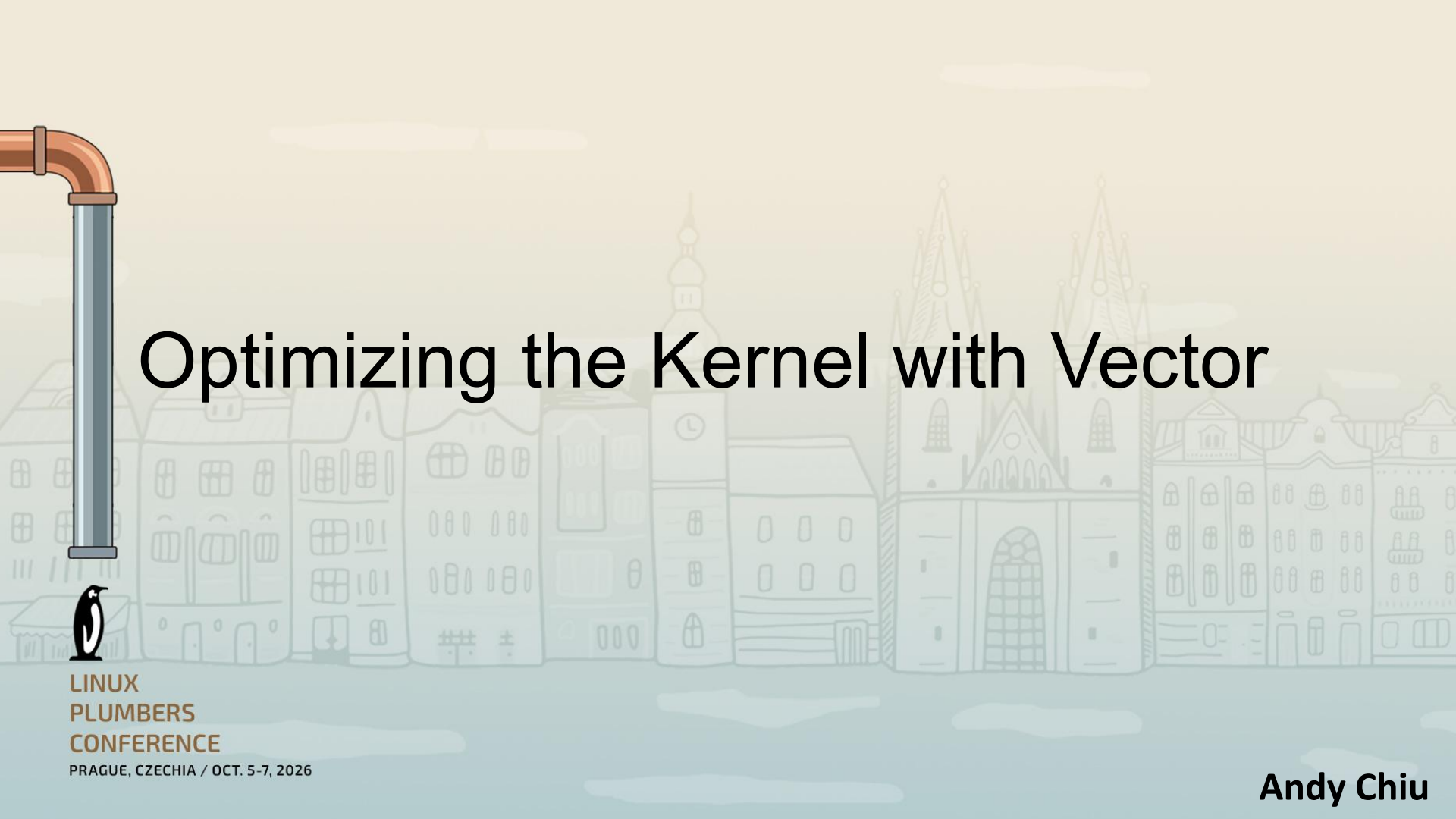




Optimizing the Kernel with Vector



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Andy Chiu

What's now

Vector is off by default, an explicit API call is required to use V:

- Managing extra context takes time/memory
- Supposedly kernel should not use fpu/vector

```
if (may_use_simd()) {           // no hardirq, no NMI
    kernel_vector_begin();      // claim V; save user V
    xor_vector(...);           // hand-written .S
    kernel_vector_end();
} else {
    xor_scalar(...);           // scalar fallback
}
```

Current Vector use-cases

- crypto
- raid6
- xor
- user copy (size > 768)

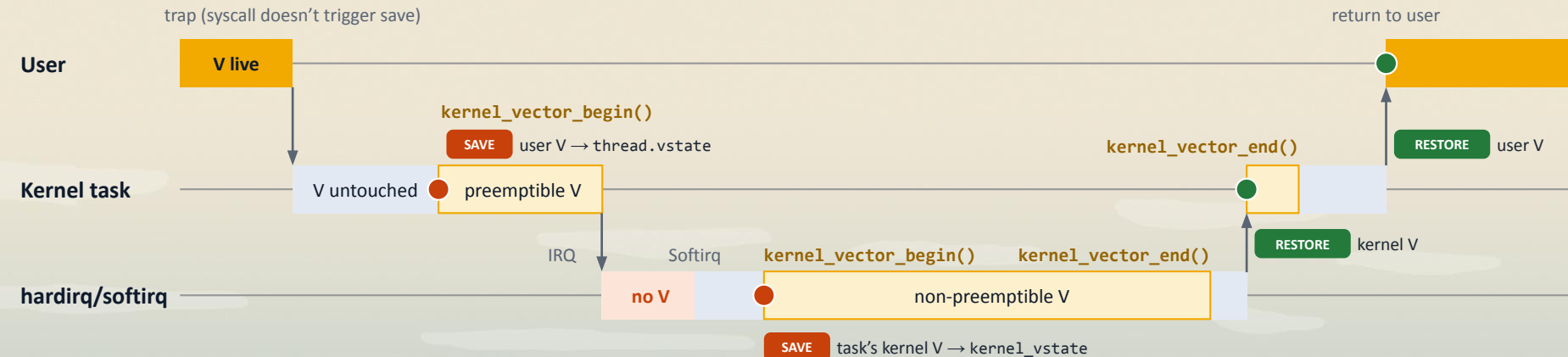


LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



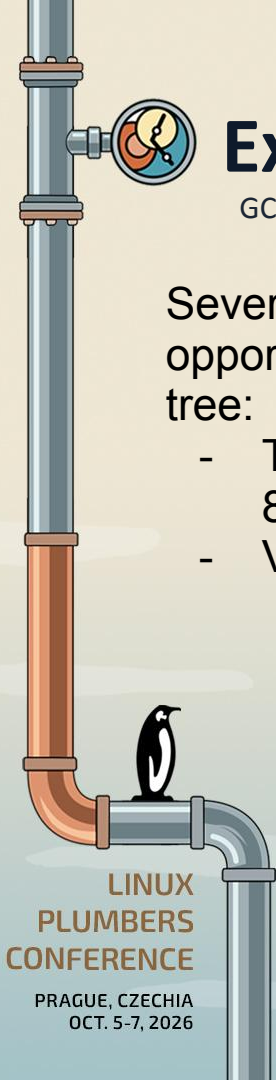
Life Cycle of Kernel-Mode Vector



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026





Examples of Vectorization in the Source Tree

GCC 16.1, -mtune=tt-ascalon-d8

Several auto-vectorizable code opportunities are available in the source tree:

- Total share of the vectorized code: 8.8/1K insn
- Vectorized reasons:
 - 27.1% SLP sequence fill
 - 24.4% (SLP/inline) block copies
 - 18.5% (SLP/inline) block set
 - 12.2% strings
 - 11.1% loops

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Loop vectorization

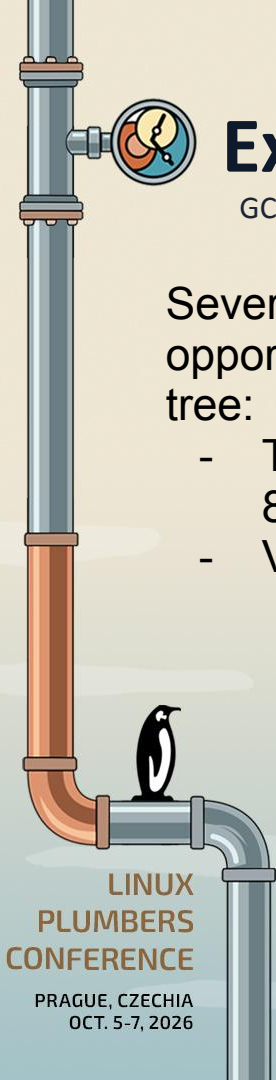
```
lib/lz4/lz4_compress.c:441, LZ4_compress_generic()
for (; accumulator >= 255; accumulator -= 255)
    *op++ = 255;
```

SLP: straight-line code

```
include/linux/list.h:237, list_del()
__list_del_entry(entry);
entry->next = LIST_POISON1;
entry->prev = LIST_POISON2;
```

Block copy: memcpy / memmove

```
include/linux/skbuff.h:4620, skb_data_move()
/* skb_data_move() */
memmove(skb->data, skb->data - len, n);
/* vlan_remove_tag(): n = 2 * ETH_ALEN */
```



Examples of Vectorization in the Source Tree

GCC 16.1, -mtune=tt-ascalon-d8

Several auto-vectorizable code opportunities are available in the source tree:

- Total share of the vectorized code: 8.8/1K insn
- Vectorized reasons:
 - 27.1% SLP sequence fill
 - 24.4% (SLP/inline) block copies
 - 18.5% (SLP/inline) block set
 - 12.2% strings
 - 11.1% loops

Loop vectorization

```
lib/lz4/lz4_compress.c:441, LZ4_compress_generic()
for (; accumulator >= 255; accumulator -= 255)
    *op++ = 255;
```

SLP: straight-line code

```
include/linux/list.h:237, list_del()
__list_del_entry(entry);
entry->next = LIST_POISON1;
entry->prev = LIST_POISON2;
```

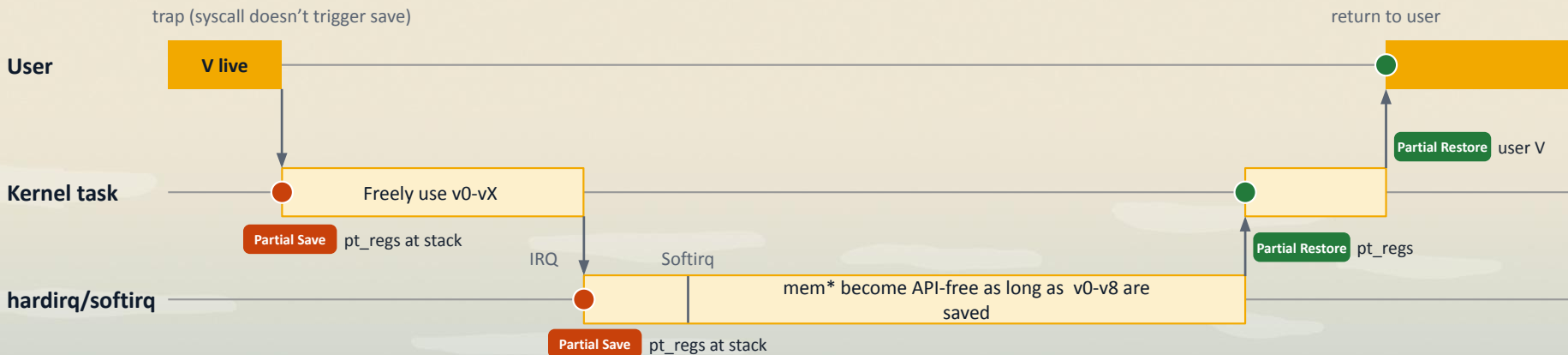
Block copy: memcpy / memmove

```
vmlinux __skb_vlan_pop (inlined)
ld      a5,0xc8(s1)      # skb->data
addi    a2,a5,-4         # skb->data - VLAN_HLEN
vsetivli zero,12,e8,m1,ta,ma
vle8.v  v1,(a2)          # load all 12 bytes
vse8.v  v1,(a5)          # then store
```

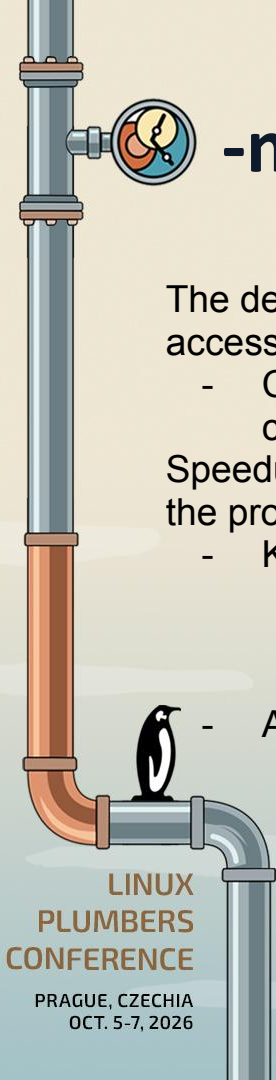
LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Partial Save of V regs at the Trap Boundary



VLEN	v0-v7	v0-v15	v0-v31
128	128 B	256 B	512 B
256	256 B	512 B	1 KiB
512	512 B	1 KiB	2 KiB
1024	1 KiB	2 KiB	4 KiB



-mtune: Vectorizing in a Sub-Optimal Way

The default mtune assumes slow unaligned access:

- Codegen optimizes unknown alignment byte copies to byte-aligned vector copy

Speedup when compiled with NONPORTABLE and the proper mtune:

- K3 (generic-ooo):
 - Geomean +4.98% of 24 system benchmarks (pts)
 - +34.8% zram-lz4 improvement
- Ascalon-X (tt-ascalon-d8):
 - +90.6% on zram-lz4
 - +35.4% on zram-zstd
 - +7.4/+7.2% on btrfs-create/unlink

Codegen for byte copies

GCC 16, copy8 test

COPY8: -mtune=generic, V off

```
lbu  a5,0(a1)
sb   a5,0(a0)
lbu  a5,1(a1) # ... 8x lbu + 8x sb: 18 insns
```

COPY8: V on

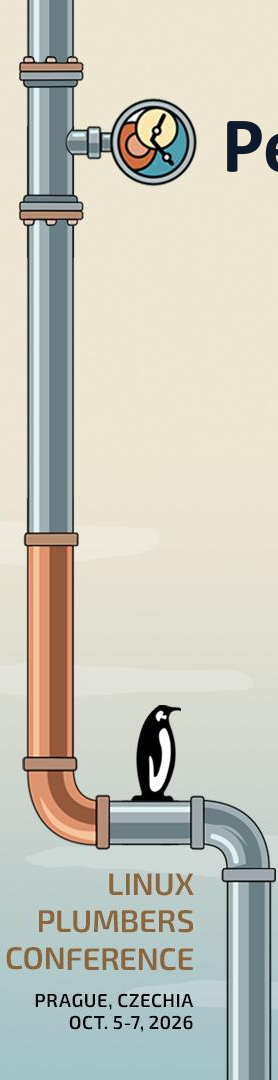
```
vsetivli zero,8,e8,mf2,ta,ma
vle8.v v1,(a1)
vse8.v v1,(a0) # 3 insns
```

COPY8: -mtune=generic-ooo

```
ld   a5,0(a1)
sd   a5,0(a0)
```

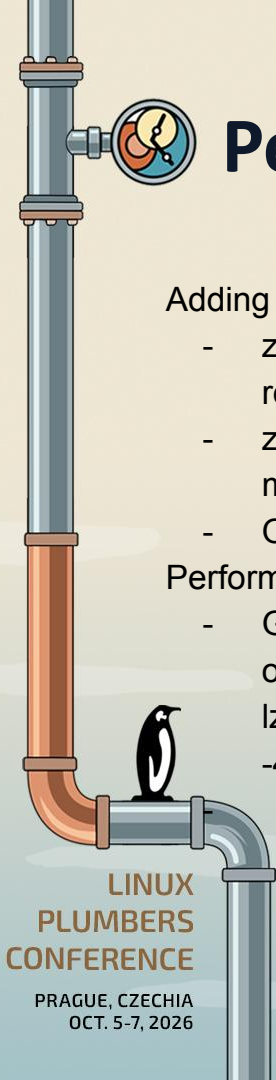
LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



Performance of the Vectorized Kernel

Benchmark	Vector Density	Major Contributor	Gain over mtune
lz4 (zram)	37.3	loop (77%)	+15.1%, +1.0% on ASC-X -7.5%, +9.4% on K3 (compress, decompress)
btrfs (fsmeta)	11.7	block ops (13% + 70%) (inline/hand-written)	+5.2%, +8.4% on ASC-X +6.9%, +3.7% on K3 (create, unlink)
xfs (fsmeta)	26.2	block ops (57% + 21%)	+2.5%, 5.0% on ASC-X Not Significant on K3
nginx	18.3	handwritten block ops (74%)	0.9 to +2.2% on ASC-S 0.3 to +1.9% on K3 80B-32KB serving sizes



Performance of the Vectorized Kernel

Adding more registers gives diminishing returns:

- zram-zstd gives 2.4% more improvement with 16 regs on ASC-X.
- zram-lz4 compress on K3 regresses less with more regs.
- Other benchmarks give less than 1% differences.

Performance gain depends on workload and hardware:

- Geomean gain for K3 on the system benchmarks over -mtune is 0.1%, worst regressions are lz4_compress (-7.5%), and hackbench, -2.51%, -4.06% (process, thread)

Benchmark	gain over mtune with 8 regs
lz4 (zram)	+15.1%, +1.0% on ASC-X -7.5%, +9.4% on K3 (compress, decompress)
btrfs (fsmeta)	+5.2%, +8.4% on ASC-X +6.9%, +3.7% on K3 (create, unlink)
xfs (fsmeta)	+2.5%, 5.0% on ASC-X Not Significant on K3
nginx	0.9 to +2.2% on ASC-S 0.3 to +1.9% on K3 80B-32KB serving sizes

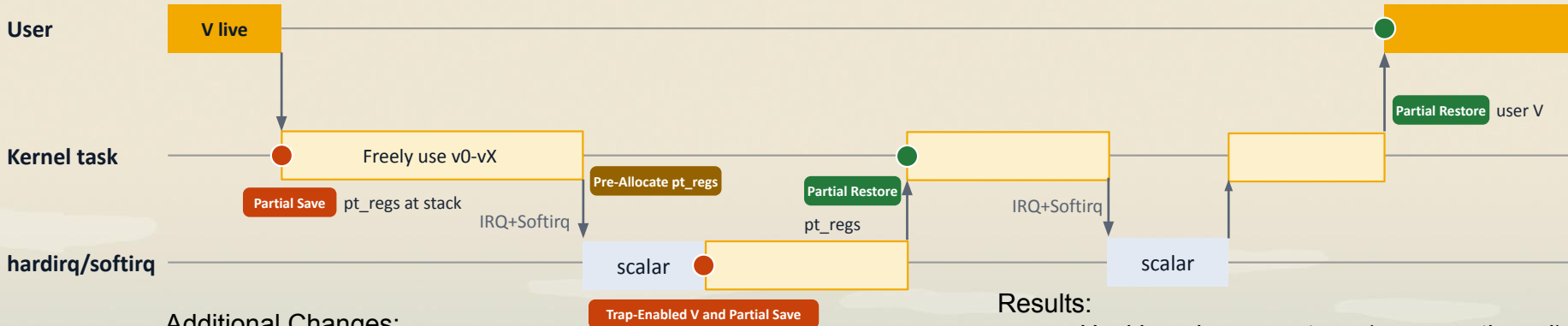
LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Proposed Solution: Trap-Enabled V in irq Context

trap (syscall doesn't trigger save)

return to user



Additional Changes:

- mem* become context-aware
- Add -mstringop-strategy=scalar to prevent vectorizing fixed sizes block ops in irq
- Files explicitly compiled without V:
 - arch/riscv/kernel/{*vector.o,traps.o,irq.o}
 - kernel/entry/common.o

Results:

- Hackbench over -mtune (process, thread): (+4.2%,+3.4%)

cyclic on idle	-mtune	autovec-8 r	autovec-8 r-noirq
p99.99990	8	10	8
p99.999990	70	91	92
max	373	343	348



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Discussion

Misaligned accesses:

- Do we remove `-mstrict-align` from the default (which depends on NONPORTABLE)?
- `-mtune=generic-ooo` looks like a good start, should we go further and open up vendor specific `-mtune`?

Autovectorization:

- Is this a right thing to approach? What are the performance metrics we are looking for as the gatekeeper

Better `-mstringop-strategy` for inlining/calling vectorized code.

- Enablement model: kernel-wise or per-module

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026