Livepatch visibility at Scale
Takes 45+ days to install a new kernel to all machines. KLP helps roll out changes.
Iterate on acculumative hotfixes (~4 hr build time)
Health check metrics monitor new kernel+klp
How to determine if a klp is in effect?

1. Add --hotfix<version> to kernel version -- problems w/netconsole
2. crashdump parsing
3. parse sysfs (need functions for all languages)

Visibility of perf impact? (steve: ftrace args better than ftrace regs) Add a per-cpu counter to struct klp_func? (steve: static branch for the counter)
Visibility of transition failures

KLP for Clang LTO Kernel
KLP at Meta uses kpatch fork, enhanced for PGO and LTO
LTO compilation workflow moves objtool to run against vmlinux.o, not individual object files as kpatch build expects.
Current LTO kpatch build takes more than 10 hours
Workaround with linker flag (--lto-obj-path) vmlinux.o.thinlto.o[0-9]+ => foo.o -- compare these with kpatch-build instead
PGO support not upstream for kernel, kpatch likewise
(Request for a single livepatch build tool)

Kbuild support for klp-relocation generation
KLP is special, needs to reference non-included local symbols, non-exported, etc.
Kernel module loader supports livepatch module ELF formation for relocations (so called klp-relocations)
klp-convert - proposed tool to generate klp-relocations in kernel build
New minimal version proposed - based on klp-convert v7, KLP_RELOC_SYMBOL macro aliases variable to .klp.sym.rela.lp_object.sym_object.sym_name,sympos
Why is it still not upstreamed - safety concerns (will selftests make it look too easy?)

Make livepatch callbacks, shadow variables, and states work together
Relationship between shadow variables and patch / states is vague. How to manage their lifetime?
Proposal - callbacks only called upon new state introduction/removal. Garbage collect shadow variables if new patch doesn't support their associated state.

Rust MC

First MC after Rust support merged in mainline

Klint: Compile-time Detection of Atomic Context Violations for Kernel Rust Code

klint is designed for multiple uses.

Safe Rust code cannot have UBs. Deadlocks are memory-safe, but when it comes to schedule() used in rcu_read_lock() critical section, it causes safety issues in PREEMPT_NONE kernel.

Possible solutions:

* make sleep function unsafe. But it's a bad idea due to ergonomic
* token types.
* dynamic checking.
* just ignore

Klint tries to achieve soundness, no runtime cost and ergonomic in the same time.

Q: What's the build cost of klint

A: Compared to other part of compile process, the cost is very little.

<u>pin-init: Solving Address Stability in Rust</u>

Initialization issues may be thought easy to prevent, the fact is not.

All types are moveable. Address stablity in Rust is handled by `Pin<_>` API.

Q: Why field projection is needed when there already exist a few pin-projection library.

Answer (By Benno & Miguel): We are carefully choosing the third party crates in kernel, and we now don't have `syn` in kernel, so it's tricky to use these pin-projection crates.

<u>Coccinelle for Rust</u>