# Let’s merge ASI

Brendan Jackman <jackmanb@google.com>

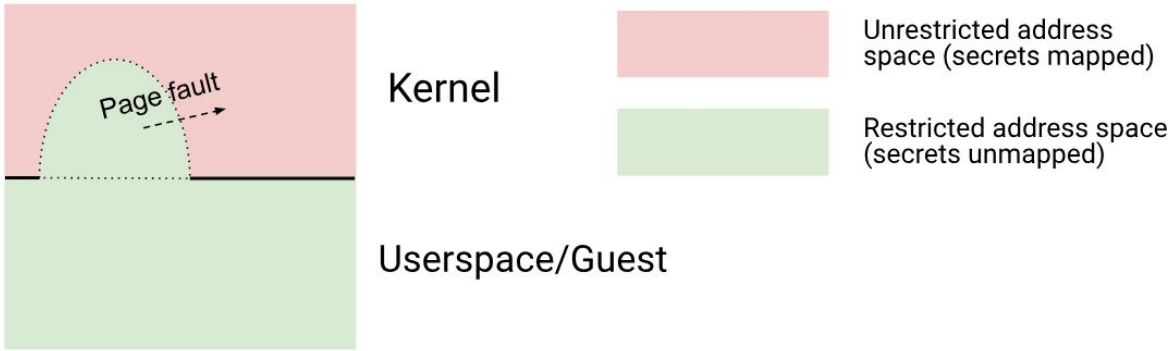
Linux Plumber’s Conference 2025

# ASI mini-refresher

For a proper intro see resources on https://linuxasi.dev/.


tl;dr:

- Run kernel without any user data mapped.
- When user data accessed, page fault, map user data again.
- CPU buffers are flushed when transitioning, breaking exploits.
- Transitions are rare so this is cheap.
- One solution fixes a wide range of vulns.



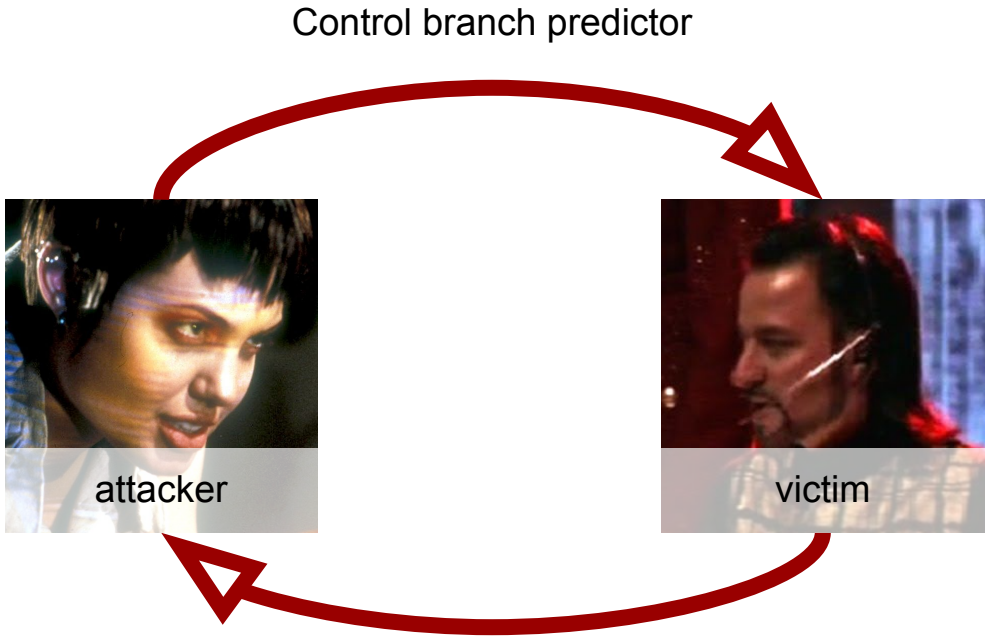
# Haven’t the CPU people cleaned up their act?

# Haven’t the CPU people cleaned up their act?

- no.

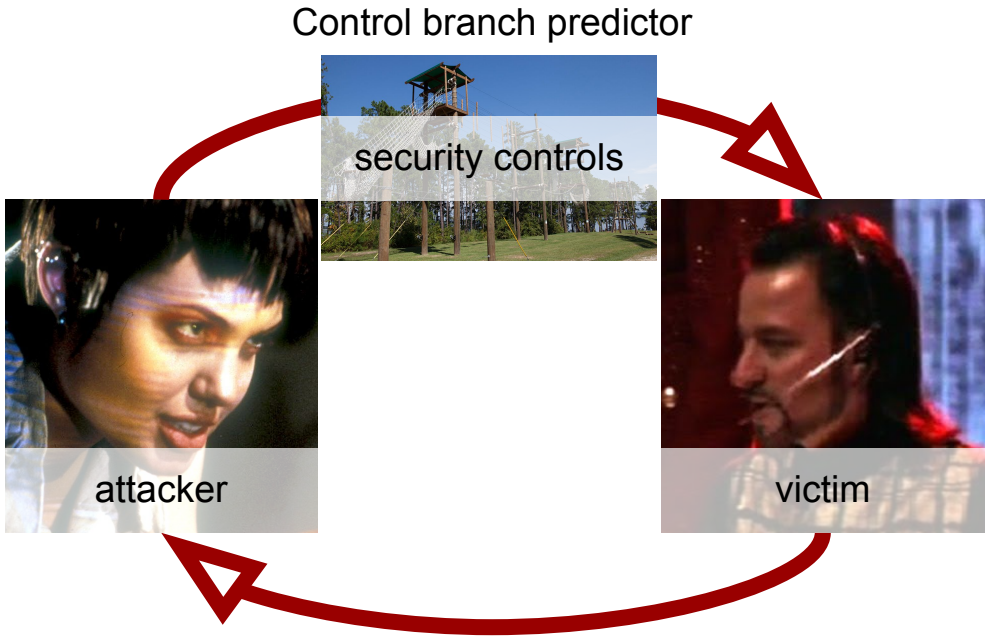
# Haven’t the CPU people cleaned up their act?

- no.



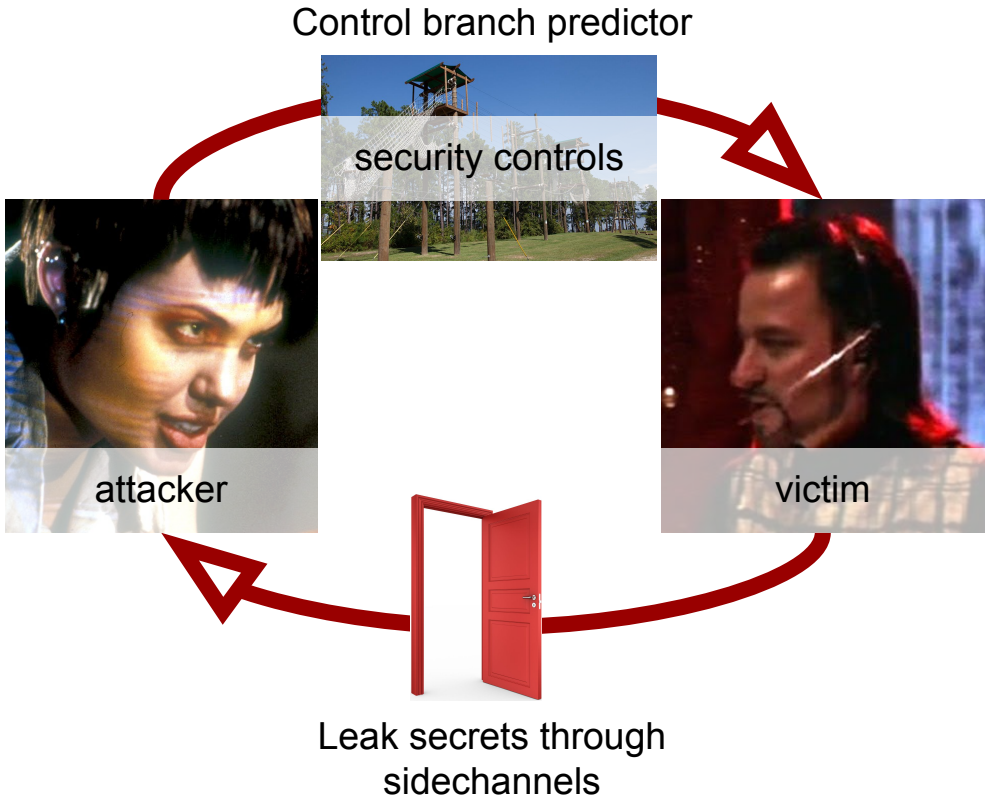
# Haven’t the CPU people cleaned up their act?

- no.
- OK, well, kinda
  - Controlling branch predictors seems to be getting pretty tricky in practice. (on new CPUs)
  - But this isn’t solvable in theory.



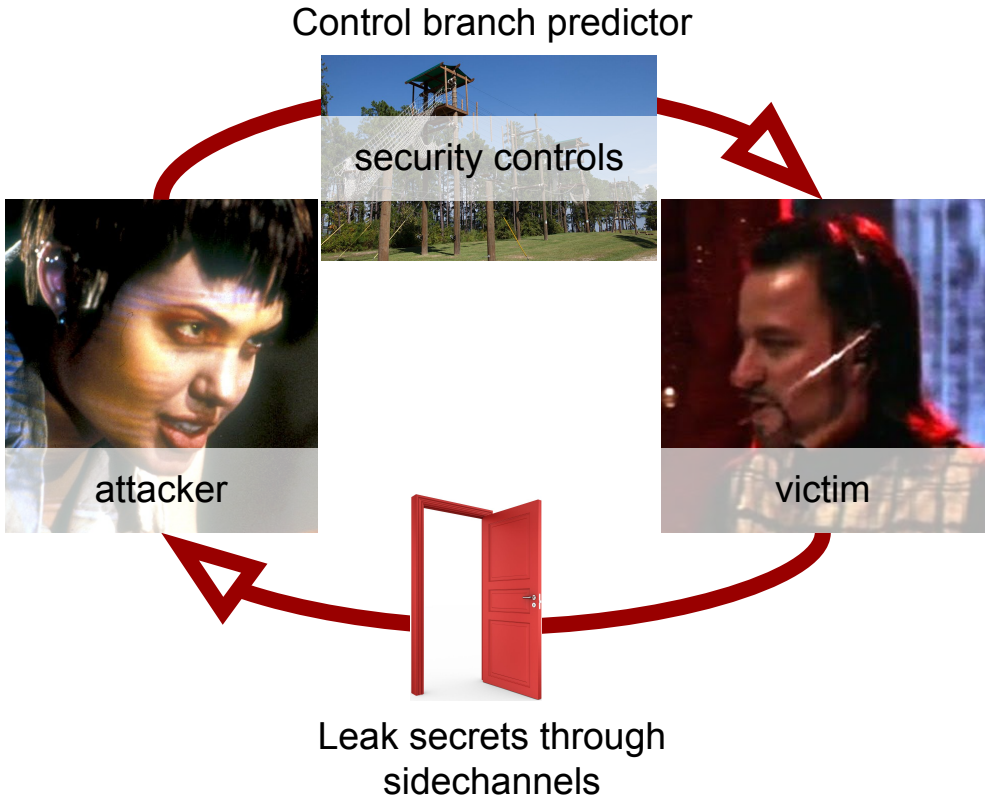
# Haven’t the CPU people cleaned up their act?

- no.
- OK, well, kinda
  - Controlling branch predictors seems to be getting pretty tricky in practice.
  - But this isn’t solvable in theory.
- Sidechannels are going nowhere



# Haven’t the CPU people cleaned up their act?

- no.
- OK, well, kinda
  - Controlling branch predictors seems to be getting pretty tricky in practice.
  - But this isn’t solvable in theory.
- Sidechannels are going nowhere
- CPU security is our problem, for good



# Can’t you just unmap KVM guest memory?

# Can’t you just unmap KVM guest memory?

- yeah!

# Can’t you ~~just~~ unmap KVM guest memory?

- yeah!
- OK, well, kinda
  - If you can unmap *all* your secrets this way
    - (Not currently possible AFAIK)
  - And your stack is fully guest_memfd-ready
    - (Spoiler alert: it isn’t)

# Can’t you ~~just~~ unmap KVM guest memory?

- yeah!

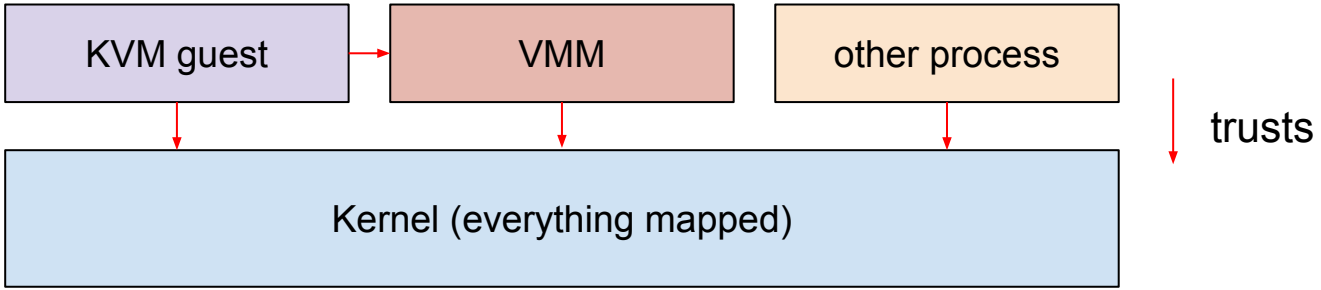
- OK, well, kinda
  - If you can unmap *all* your secrets this way
    - (Not currently possible AFAIK)
  - And your stack is fully guest_memfd-ready
    - (Spoiler alert: it isn’t)

- I ❤️ this effort
- (Something like) this is the endgame for cloud providers
- But cloud providers aren’t the only Linux users

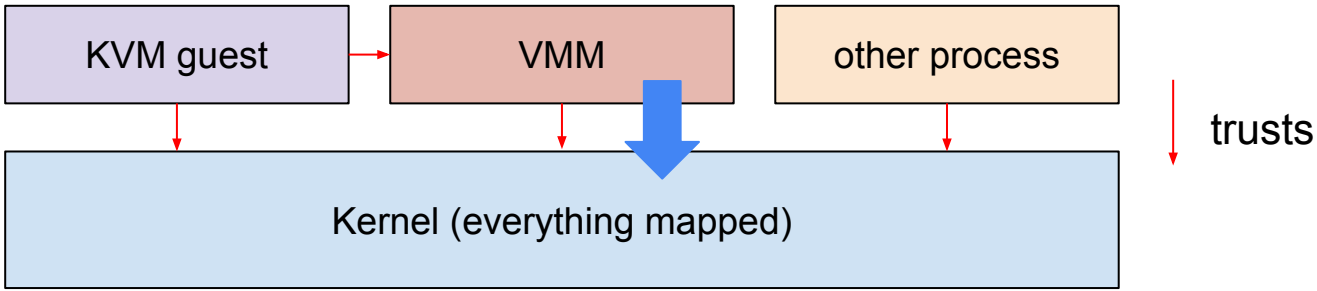
# How ASI pre-empted VMSCAPE

ASI has a “domain” model”

When you enter a domain, what do you need to do to secure the CPU?



# Example: asi_exit()



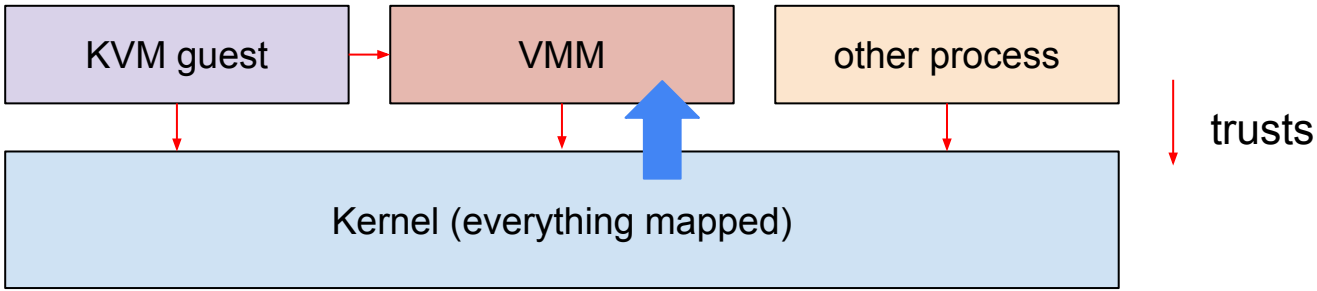

Servicing a syscall, get an ASI #PF, need to switch to full address space

- User process controls branch predictor, may attack kernel.
- ~~Sidechannels contain process' data. Kernel may leak it~~.

(don't care, process trusts kernel!)

Conclusion: **need to flush branch predictor**.

# Example: asi_enter(VMM)

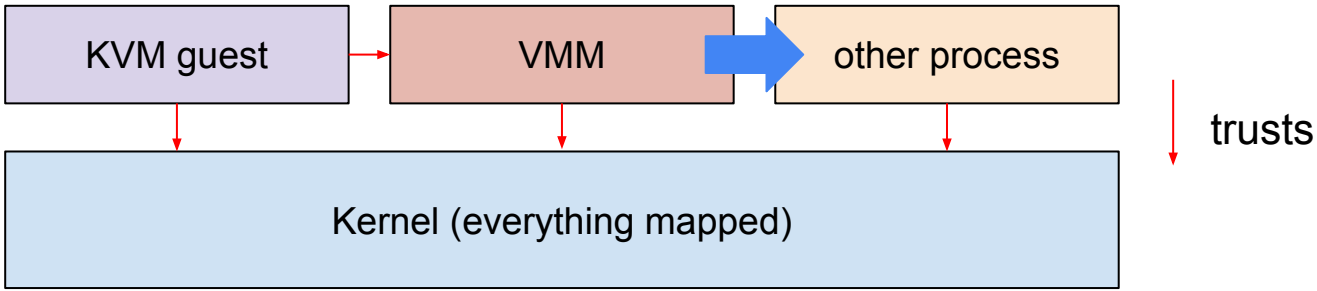



Returning to userspace

- ~~Kernel controls branch predictor, may attack userspace.~~
- Sidechannels may contain arbitrary data. Process may leak it.

Conclusion: **need to flush sidechannels**.

# Example: asi_enter(other process)



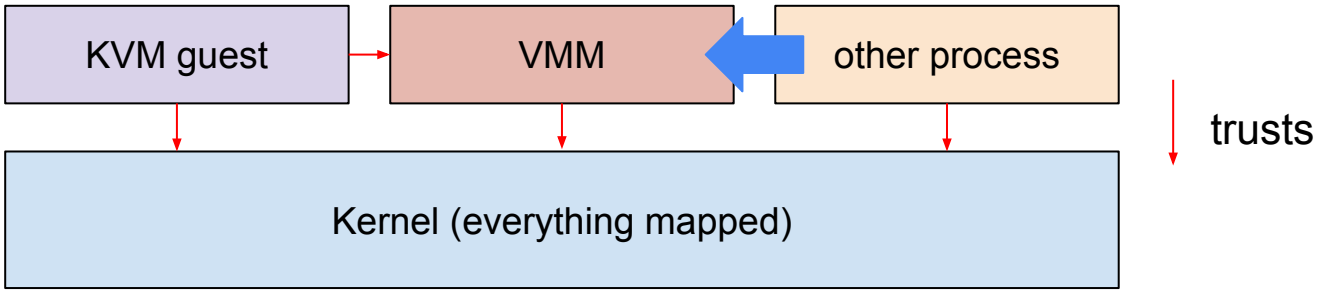

Switching processes without a #PF*

- VMM controls branch predictor
- Sidechannels contain process' data

Conclusion: **flush BP** (protect other process from BVMM) **AND sidechannels** (protect VMM from other process)

*Initial patchset always asi_exit()s is context switch. But that's a temporary simplification.

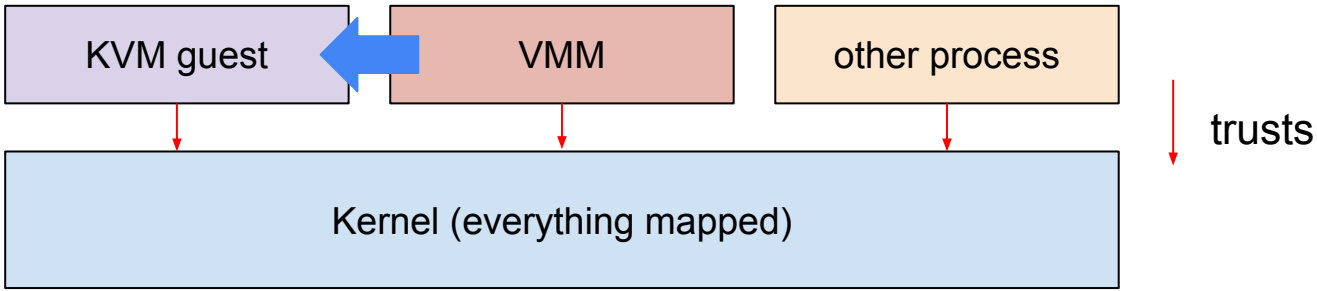
# Example: asi_enter(VMM)




Back again, same flushes

*Initial patchset always asi_exit()s is context switch. But that's a temporary simplification.

# Example: asi_enter(KVM)



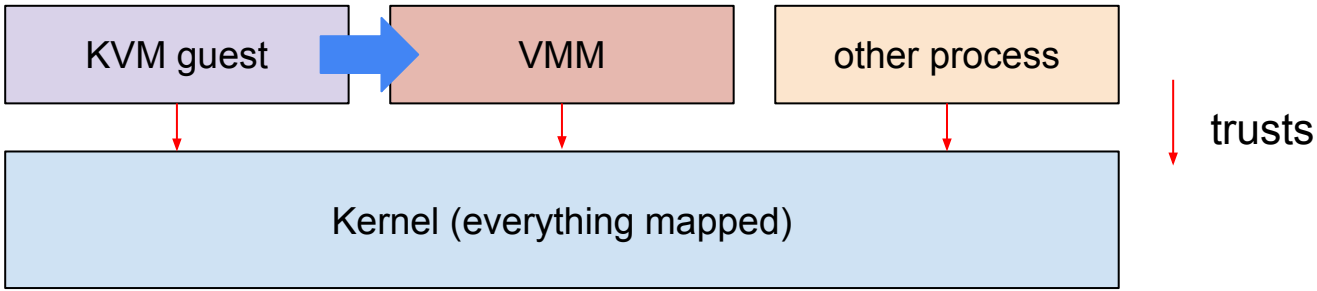

Entering VM guest

- ~~VMM controls branch predictor~~
- Sidechannels contain VMM data*

Conclusion: **flush sidechannels**

*Actually, VMM might not care. Ideally there are no secrets here. Should be configurable.

# Example: asi_enter(VMM)

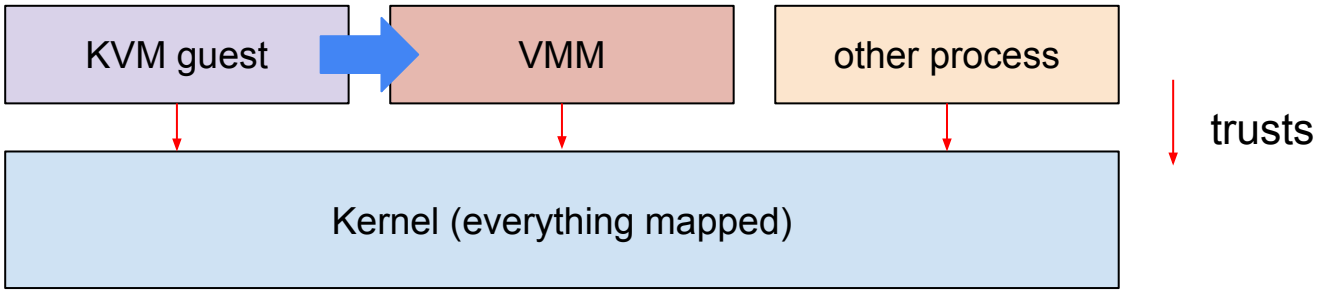



Entering returning to VMM

- VM guest controls the branch predictor
- ~~Sidechannels contain guest data~~

Conclusion: **flush branch predictor**

# Example: asi_enter(VMM)




Entering returning to VMM

- VM guest controls the branch predictor
- ~~Sidechannels contain guest data~~

Conclusion: **flush branch predictor**

**Oops, just mitigated VMSCAPE**

# ASI lets you think systematically

- This “domain model is actually simple and obvious
- It was obvious that VMSCAPE would be possible
- Obvious mitigations are too expensive if every kernel entry is a domain transition
- With ASI that’s not the case any more.

# Current challenge is “patchset feng shui”

- Nobody wants to review `[PATCH 0/100] Address Space Isolation`
- I posted `[PATCH 00/21] mm: ASI direct map management`
- Nobody wants to merge code that doesn’t do anything
- Need to find ways to break it down into things that deliver incremental value
- Ideas:
  - Refactor L1TF/VMSCAPE/SpectreV2 mitigations to resemble ASI “domain model”?
    - I tried it, it works, but it feels like “churn” without value. Not really a cleanup.
  - Refactor pagetable code to make ASI additions easy?
    - Yeah. So far haven’t found a route forward here but plenty of room for exploration
  - Help out the guest_memfd unmapping feature?
    - This is the most exciting thing at the moment - immediate usecases for my code.

# `freetype_t` and `__GFP_UNMAPPED`

- Page allocator implements mapping/unmapping for ASI
- I implement this at the pageblock level
- Introduce `freetype_t`
  - Just a generalisation of migratetype

- `guest_memfd` unmapping folks having a hard time unmapping efficiently
- I can help!!!!
- Introduce `__GFP_UNMAPPED` - “gimme pages that aren’t in physmap”
- This can then be easily adapted to support ASI’s needs
  - (`__GFP_SENSITIVE` - “gimme pages that are protected by ASI”)
- What about `__GFP_UNMAPPED | __GFP_ZERO`

# `freetype_t` and `__GFP_UNMAPPED`

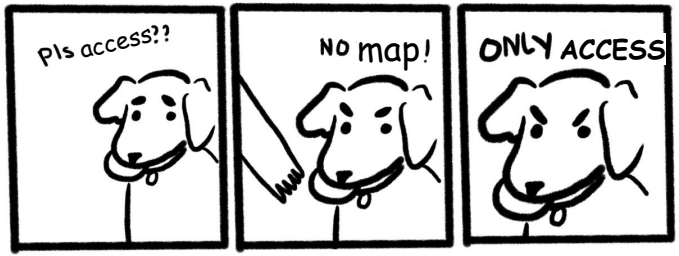
- Page allocator implements mapping/unmapping for ASI
- I implement this at the pageblock level
- Introduce `freetype_t`
  - Just a generalisation of migratetype

# The mermap (ephemeral mapping - f.k.a “ephmap”)

Want to remove stuff from the direct map etc.

But then we want to access it e.g.

- For __GFP_ZERO
- For a read() syscall
- For a user page’s COW




Solution: map it very briefly

- logically CPU-local (no shootdown on unmap)
- physically mm-local (other process can’t leak it)

Similar to kmap_local_page()

# mermap vs kmap_local_page()

| kmap_local_page() | mermap_get() |
| --- | --- |
| disable migration, use per-CPU region | disable migration, use per-CPU region |
| unmap without shootdown | unmap without shootdown |
| mapped by **global kernel pagetable** | mapped **locally to mm** |
| map **single page** | allocate & **map arbitrary sizes** |
| mostly **has to succeed** | mostly **allowed to fail** |

ringbuf "allocator"

exception: mermap_get_reserved() always succeeds to map a single base page, requires IRQs off

# Current idea for patch evolution

[PATCH 0/20]

1. mermap
2. freetype_t
3. __GFP_UNMAPPED

mermap supports `__GFP_UNMAPPED | GFP_ZERO`

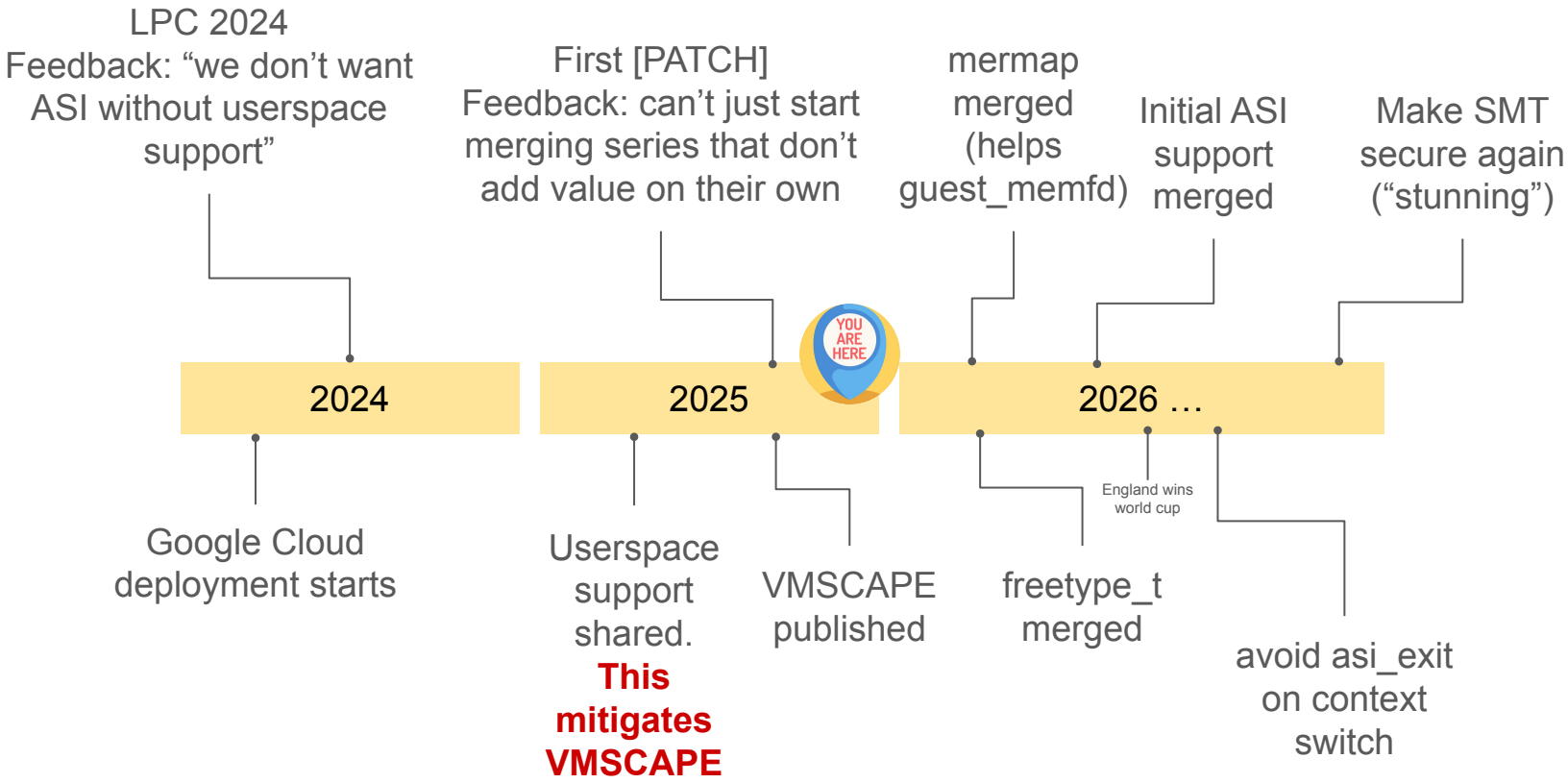
`__GFP_UNMAPPED` supports `GUEST_MEMFD_FLAG_NO_DIRECT_MAP`

4. Build nonsensitive address space
5. Introduce ASI API
6. Support sandboxing KVM + userspace

[PATCH 0/40]

7. Avoid asi_exit on context switch
8. Squash more asi_exits….
9. Make ASI default / start removing other mitigations
10. Support SMT protection (“stunning”)
11. Add loads of annoying control knobs
12. Support arm64…

# Timeline


LPC 2024
Feedback: “we don’t want ASI without userspace support”
First [PATCH]
Feedback: can’t just start merging series that don’t add value on their own
mermap merged (helps guest_memfd)
Initial ASI support merged
Make SMT secure again (“stunning”)
YOU ARE HERE
2024
2025
2026 …
Google Cloud deployment starts
Userspace support shared.
This mitigates VMSCAPE
VMSCAPE published
England wins world cup
freetype_t merged
avoid asi_exit on context switch