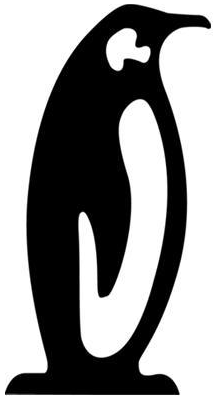


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Transitioning Android from ashmem to memfd

Isaac Manjarres <isaacmanjarres@google.com>

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Historical context

- Ashmem is used in Android to share memory regions across processes using file descriptors (fds).
- Ashmem was removed from staging in Linux kernel v5.18 in 2022 in favor of memfd.
- Android has made efforts to incrementally migrate to memfd (e.g. Joel Fernandes' addition of memfd support in the ASharedMemory interface).

Motivation for deprecating ashmem

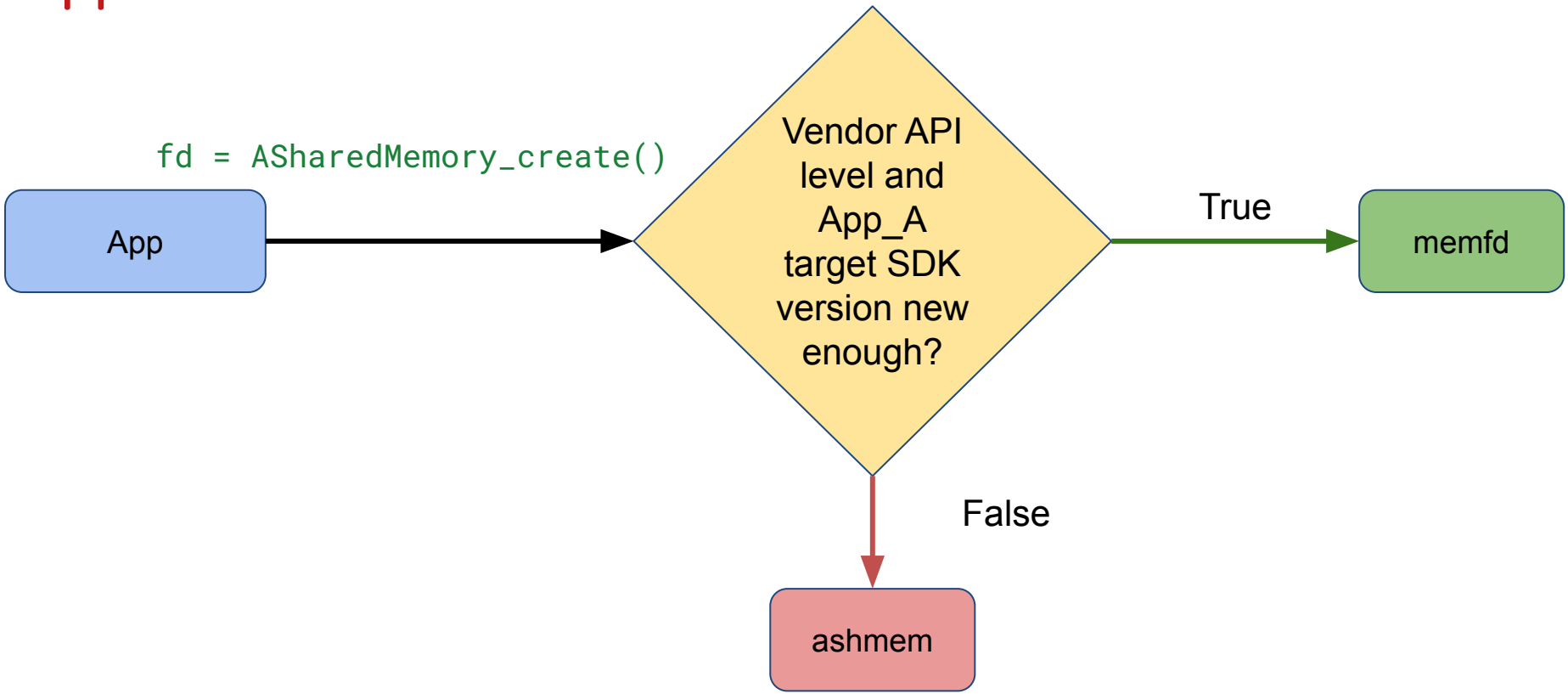
Feature	Ashmem	Memfd
Interface	fd-based; `ioctl()` commands, and `mmap()`.	fd-based; `ftruncate()`, `mmap()`, other syscalls.
Unpinning memory regions	Deprecated; no notable performance benefits.	Not supported.
Technical debt for Android	100%	0%
Testing	Android	Upstream Linux testing
SEPolicy	Overly permissive	Fine-grained

Challenges and proposed solutions

Challenge: upgrading devices

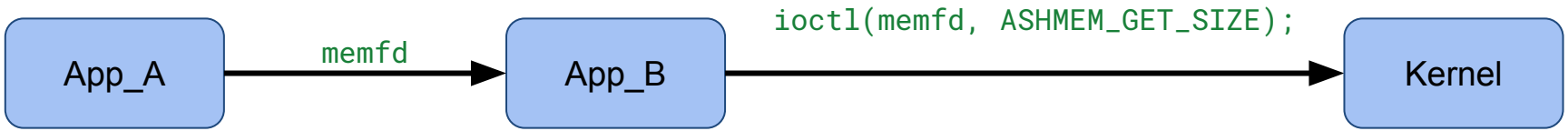
- Memfd usage may necessitate updating vendor sepolicy and seccomp policies.
- Cannot assume that vendor software upgrades at the same time as Android version upgrade with Project Treble.

Solution: use memfd on launching devices with newer applications

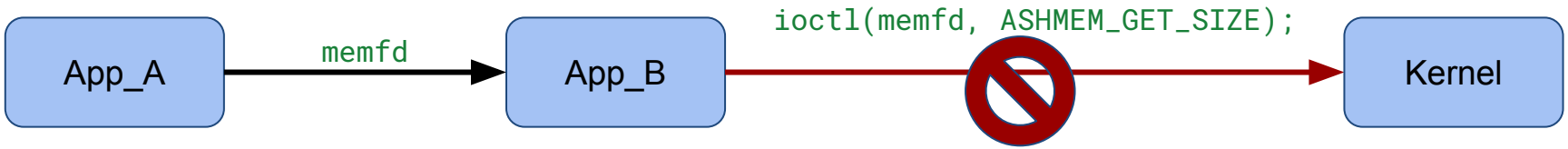


Challenge: assumptions about shared memory fds

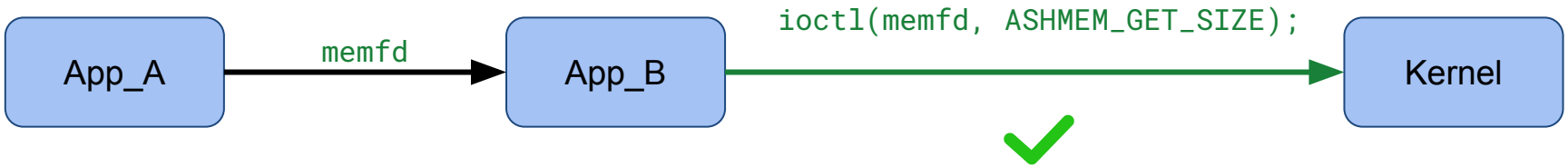
Challenge: assumptions about shared memory fds



Challenge: assumptions about shared memory fds



Solution: support ashmem ioctl commands for memfd



Note: SEPolicy would enforce that this is only supported for applications with older target SDK versions.

Challenge: SELinux and memfd handling

```
# memfds inherit the backing filesystem's type: tmpfs
allow process_A tmpfs:file { read write getattr map };
```

Challenge: SELinux and memfd handling

```
# Policy for sharing memory between process_A and process_B

type_transition process_A tmpfs:file process_A_tmpfs;
allow process_A process_A_tmpfs:file { getattr read write map };

type_transition process_B tmpfs:file process_B_tmpfs;
allow process_B process_B_tmpfs:file { getattr read write map };

allow process_A process_B_tmpfs:file { getattr read write map };
allow process_B process_A_tmpfs:file { getattr read write map };
```

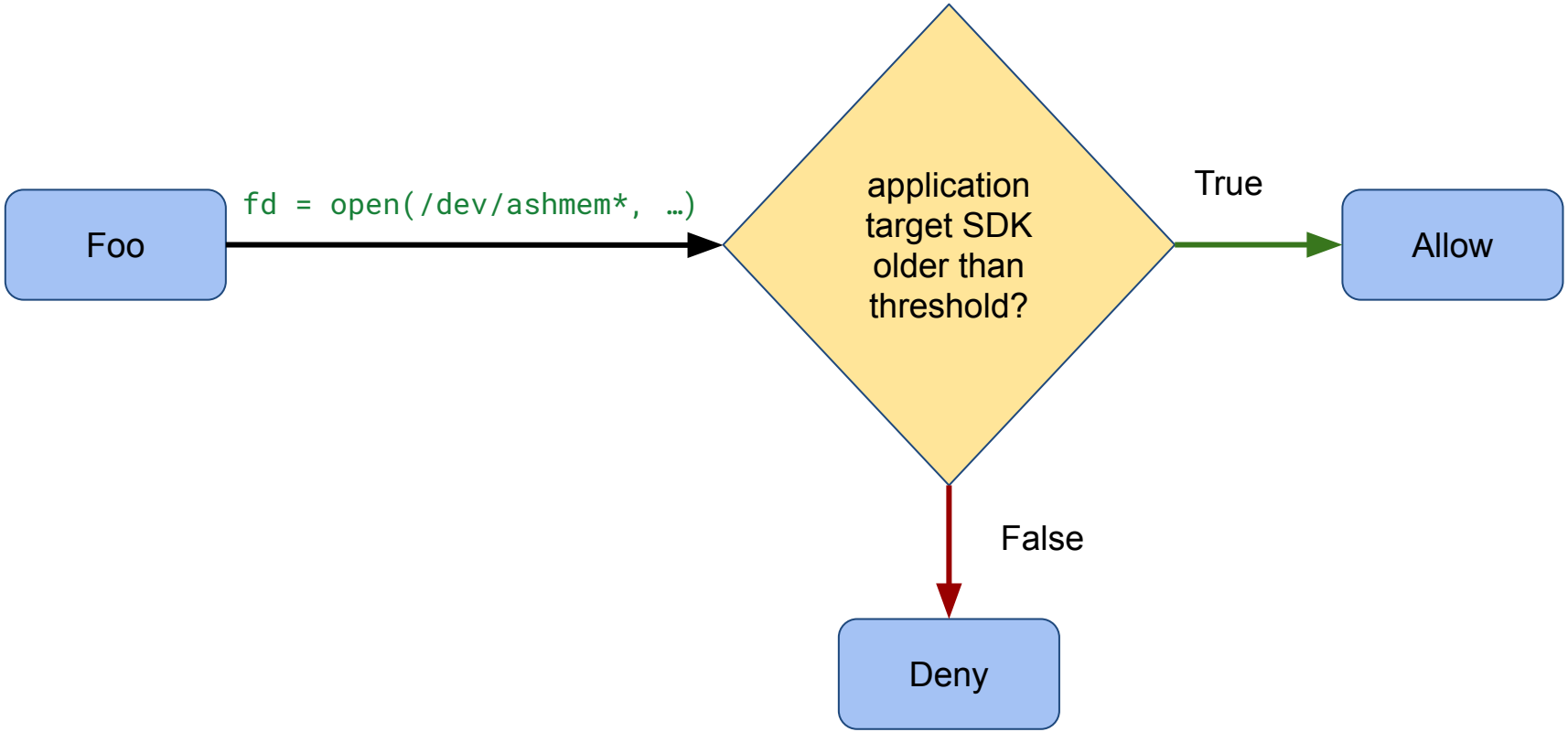
Solution: use improved SELinux handling for memfd

- Memfds can now inherit security context of the allocating process and are `memfd_file` class objects if SEPolicy advertises `memfd_class` support.
- Only enable `memfd_class` policy capability on launching devices.

```
# Improved policy for sharing between process_A and process_B

allow process_A process_B:memfd_file { getattr read write map };
allow process_B process_A:memfd_file { getattr read write map };
```

Use SEPolicy to restrict ashmem allocations



Proposal summary

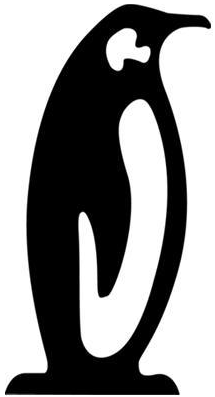
- Launching devices allocate shared memory via memfd if an application is new enough.
- Launching devices enable `memfd_class` support in SEPolicy and implement fine-grained SEPolicy for shared memory usecases.
- Support for ashmem ioctl commands is supported for older applications.
- Ashmem allocations are denied for newer applications.
- Ashmem can be removed once all supported devices and applications are new enough.

Discussion

- Feedback on the proposed migration plan?
- Is there a cleaner way to enable the `memfd_class` capability rather than requiring devices to enable it?
- Is anyone dependent on the ashmem unpinning feature?
- How can we make the transition to memfd easier?
- Are there any thoughts on how to extend the transition to memfd to upgrading devices?

Thank you!

Isaac Manjarres <isaacmanjarres@google.com>



東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025