

Using DT overlays to manage a collection of related boards

Douglas Anderson (Google)
Chen-Yu Tsai (Google)



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Problem statement

- Devices ship with a single "OS Image" that supports several related (but slightly different) boards.
- The bootloader has a non-DT way to ID what hardware it's running on.
 - EEPROM, special region of flash, compiled in, etc.
- Even though the "OS Image" includes support for several different boards, Linux needs to be provided just one device tree. The bootloader is responsible for providing it.
- The boards supported by one image are highly related. They all use the same SoC and are based on the same reference design.



Problem statement (continued)

- To support this efficiently, people use device tree overlays.
 - Image has 1-2 base device trees and a pile of DT overlays.
 - Bootloader picks the correct overlays and combines before giving to Linux (or other HLOS).
- Downstream bootloaders have various ways that they support this, but everyone comes up with their own scheme.
- Upstream doesn't like downstream solutions, resulting in problems trying to upstream device tree files.
- Goal: agree on a scheme that upstream accepts that can meet downstream needs.



LINUX

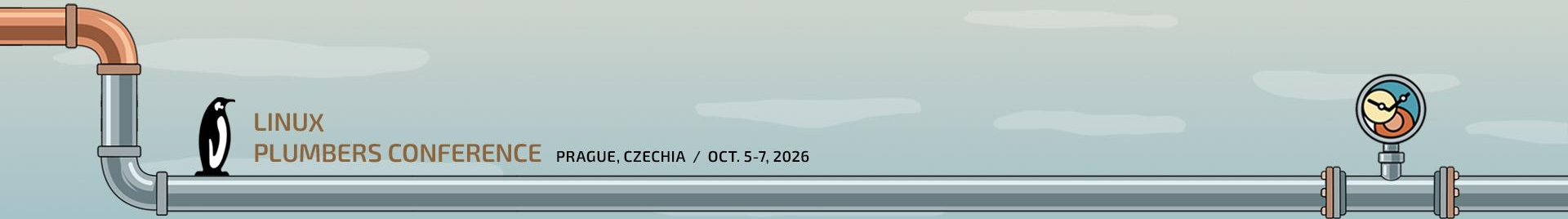
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Aside: Why not just ship N full DTBs?

- Lots of boards / revisions supported and it ends up taking up lots of space.
- Combinatorial explosion. If we can use overlays for specific components we can avoid $N \times M$ full DTBs.



Q: What does the base DT represent?

- There is some belief upstream that the base DT (.dts / .dtb) needs to be a full (bootable) board. In this case, overlays are only for "add-on" components that are not truly needed to boot.
- Specifically: can the base DT represent the SoC and then we use overlays to implement the boards?
- If the base DT can represent the SoC (🎉), great!
- If the base DT can't represent the SoC, what are the exact requirements? Does the base DT need to represent a bootable board?
 - What if we want to share base DT among boards, one which only has UFS and one only eMMC?



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Q: What does the base DT represent?

- Option 1: Base DTB can represent just the SoC (or an unbootable common subset), and boards are implemented via overlays.
- Option 2: Base DTB must represent a full, bootable board (e.g., reference board).
 - If so, how do we share a base DTB among boards with mutually exclusive components (e.g., UFS-only vs. eMMC-only)?
 - Overall, limits the ways we can use overlays.

<insert audience discussion here>



LINUX

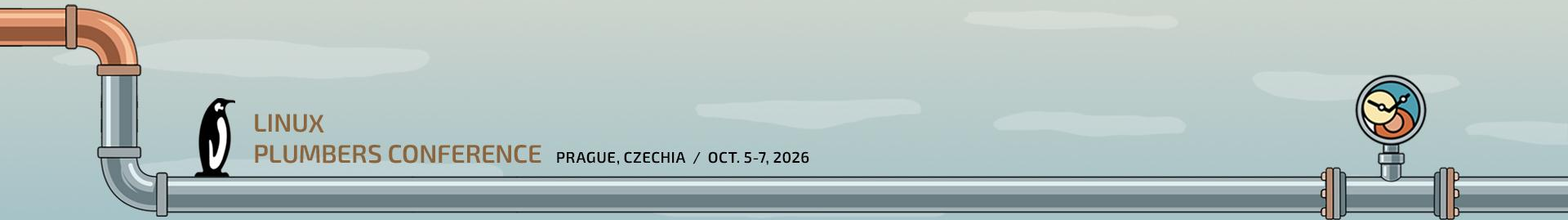
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Q: How does bootloader pick overlays?

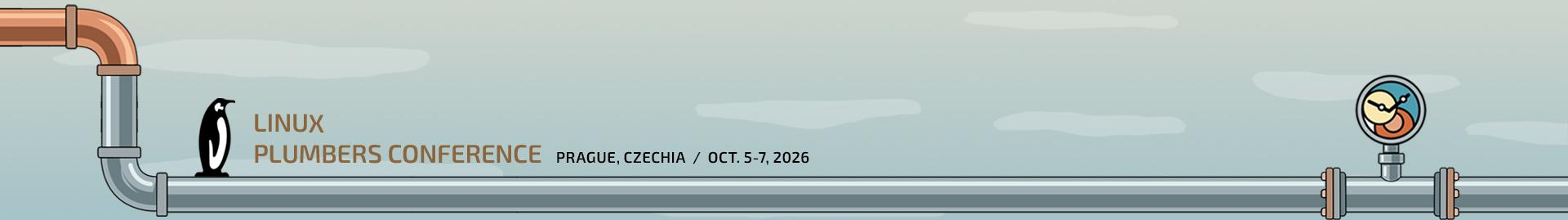
- OS Image has DTs and overlays for a pile of related devices.
- Bootloader knows which device it's on.
- How does the bootloader pick the right DTs and overlays?



Q: How does bootloader pick overlays?

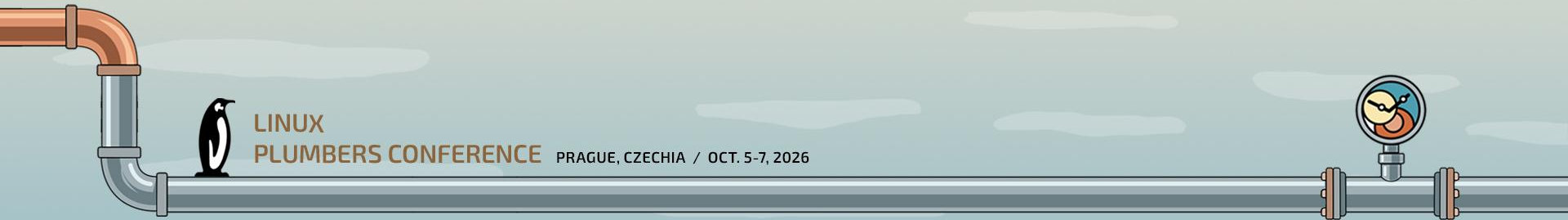
Two real options:

1. Maintain a parallel, out-of-tree file associating each overlay with the properties that the bootloader needs.
 - a. You can't truly boot upstream without this magic file.
 - b. Doesn't feel like a full solution.
2. Add properties to each overlay describing it and the bootloader uses those properties to help pick.
 - a. Seems like we should do this.



Aside: what the bootloader actually sees

- For performance reasons, the bootloader doesn't parse all overlays.
- When an OS image is created, dtbo files are bundled with metadata that talks about them.
- Tool doing the bundling could either get the metadata from a parallel file or could get the metadata straight from an overlay.



Q: How does bootloader pick overlays?

- The docs actually say it's OK to put properties in the overlays for this purpose! [Documentation/dt-object-internal.txt](#) (in dtc):
 - /* various properties for loader use; i.e. part id etc. */
- During review of trying to add such a property:
 - *adding such new properties is fine, but they must come for more than one user and be universal across these users.*
- During the last several years, nobody has ever achieved the dream of universality.
- Where are we left: no upstream solution, everyone downstream reinvents the wheel with no upstream review.



LINUX

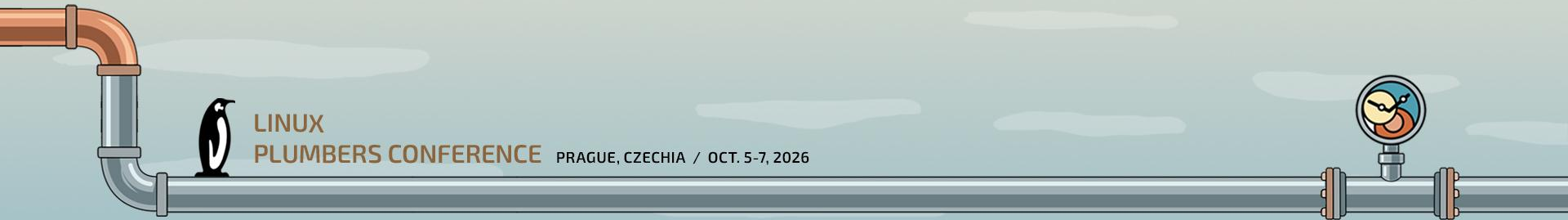
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Q: How does bootloader pick overlays?

- Proposal: accept and embrace that we don't need a universal solution.
- Allow different "loaders" to define their own schema.
- Upstream the bindings for the different "loaders" so they can be reviewed by upstream and overlays can be validated.
- If multiple loaders can use the same bindings, great. If there are different loaders with different needs, that's OK.
- Having structured and documented data describing each overlay is a win.



Q: How does bootloader pick overlays?

- Each loader could be assigned a unique node name and document their bindings.
- One loader could be for the Android "mkdtimg" and documents / requires board-id and board-rev. Could offer optional "custom1" through "custom4" to match Android DTBO format.
- Other loaders can have totally different requirements and that's OK.

```
/ {  
  loaders {  
    mkdtimg {  
      board-id = <0x1234>;  
      board-rev = <0x5678>;  
    };  
    other-loader {  
      something-else = <0xaaaa>;  
    };  
  };  
};
```



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Q: How does bootloader pick overlays?

- Note: "loaders" is in the root of the overlay. In other words, this:
 - `/ { loaders { ... }; };`
- Not:
 - `&{/} { loaders { ... }; };`
- This means it automatically is stripped when an overlay is merged into the base.

```
/ {  
  loaders {  
    mkdting {  
      board-id = <0x1234>;  
      board-rev = <0x5678>;  
    };  
    other-loader {  
      something-else = <0xaaaa>;  
    };  
  };  
};
```



Q: How does bootloader pick overlays?

- Are we agreed on putting loader metadata inside the .dtso / .dtbo rather than out-of-tree parallel files?
- Are we OK dropping the "universal across all users" requirement and allowing per-loader schemas under "/"? AKA:
 - `/ { loaders { <loader-name> { ... }; }; }`

<insert audience discussion here>



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



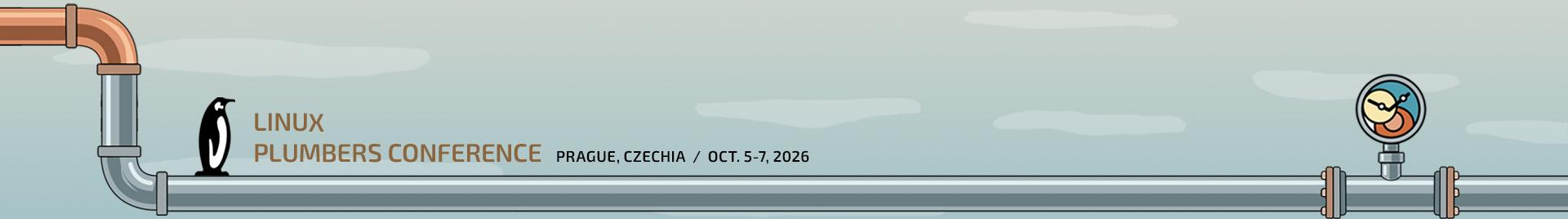
Q: What does top-level "compatible" mean?

- Ignoring overlays, top-level compatible strings often list several levels.
- If just the board and SoC, maybe:
 - "boardvendor,myboard-rev1", "boardvendor,myboard", "socvendor,mysoc-rev1", "socvendor,mysoc";
- If we also add the reference board into the compatible list:
 - "boardvendor,myboard-rev1", "boardvendor,myboard", "referencevendor,reference-codename", "socvendor,mysoc-rev1", "socvendor,mysoc";



Q: What does top-level "compatible" mean?

- Today, when overlays are applied:
 - If the overlay has a top-level compatible, it replaces the one in the base.
 - If the overlay doesn't have a top-level compatible, the base top-level compatible remains.



Q: What does top-level "compatible" mean?

- If we want to replicate the old compatible strings but use overlays, each level of overlays needs to duplicate / add to the compatible strings from the level above it.
 - If dtb is the SoC, just has "SoC" compatible.
 - If there's an overlay for the reference board, has both SoC and reference board compatible.
 - The overlay for the actual board has to duplicate all of them.
- Is this what we want? Some other solution? Some type of merging?



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Q: What does top-level "compatible" mean?

- Maybe we can reduce what the top-level compatible represents?
- We're not trying to squeeze the whole DT into that top-level compatible. If we were, we're back to board.c files.
- Maybe just put the final level without the pile of other strings?
 - "boardvendor,myboard-rev1"
- Info about what SoC we're on feels like it belongs elsewhere in the DT, not jammed into the top-level compatible.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Aside: where is top-level compatible used?

- Used by some bootloaders to pick a device-tree to use.
 - Used in DT schema validation.
 - Used to identify which SoC we're on.
 - It's a last resort, mostly in hacks.
-
- Of the above, the biggest problem is places where we need to identify which SoC we're on. If we can find a better place to put this for new device trees, it would help a lot.
 - Even the name of the reference board (if relevant) feels like it could go elsewhere.



Q: What does top-level "compatible" mean?

- Option 1 (Status Quo): Each level of overlay duplicates and adds to the compatible strings from the level above it (board-rev, board, ref-board, soc).
- Option 2 (Reduce top-level compatible): Top-level compatible only lists the final board; SoC (and reference board) identity moves elsewhere in the DT (e.g., /soc node).
- Option 3 (Merge strings): Add some mechanism to prepend/merge compatible strings when applying overlays?

<insert audience discussion here>



LINUX
PLUMBERS CONFERENCE

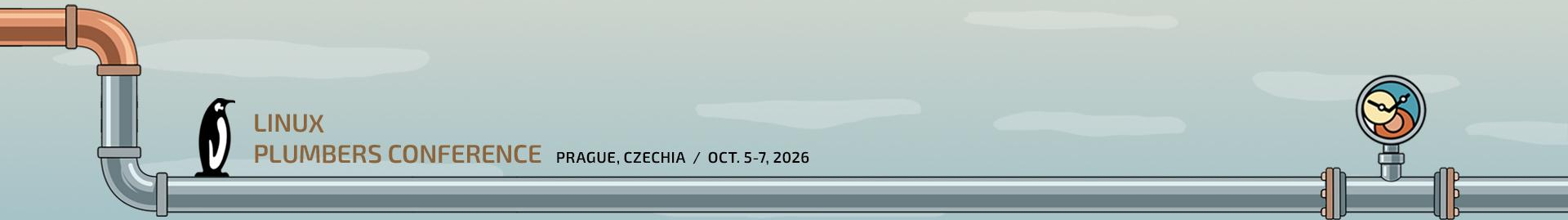
PRAGUE, CZECHIA / OCT. 5-7, 2026



Q: How do we validate all this?

- Does base DTB pass validation on its own?
- Maybe we mark it as "incomplete" and then in the final level of overlays we mark it as "complete".
- Schema validator can know not to try to validate an incomplete DT.
- To validate "loaders", validation would need to run on dtbo files directly (and just validate the "loaders" node), which would be new work.

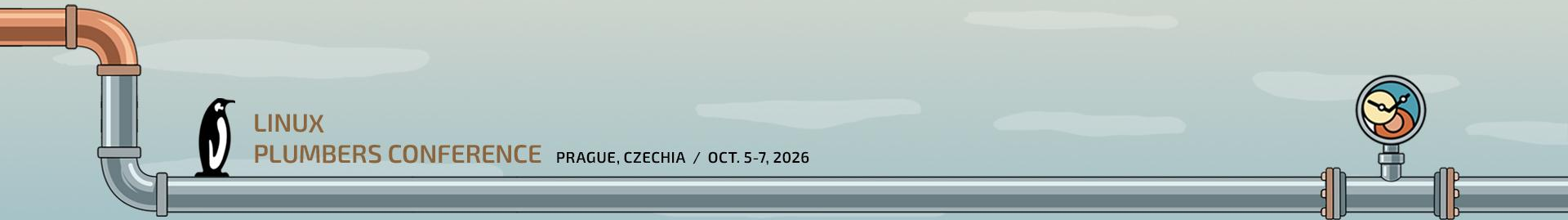
<insert audience discussion here>



Q: Is there a promised ABI?

- For anyone using overlays in this way, the base device tree and overlays will be VERY tightly coupled.
- I would assert that no ABI is promised here. All overlays must be in-tree so they can be validated together with the base.

<insert audience discussion here>



Conclusion

- Hopefully we made some progress? ;-)

