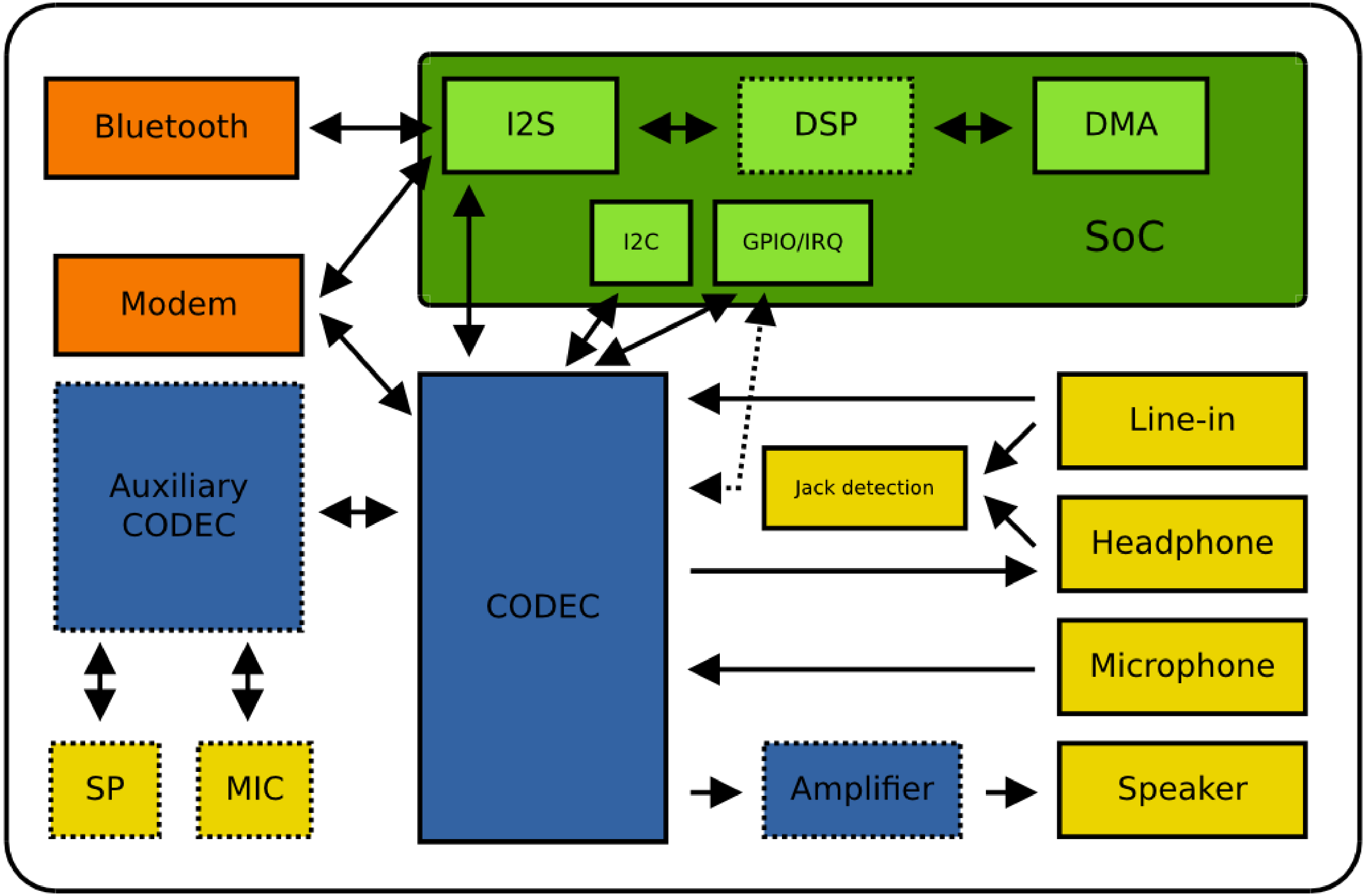
# Dynamic Audio Power Management

Lars-Peter Clausen – Analog Devices

# What is DAPM?

“Oh, it's just a graph walk, ...”

# Why DAPM?




Anatomy of a modern sound card

# Why DAPM?

- Modern sound cards consist of many independent discrete components
- Each component has functional units that can be powered independently
- Audio routing matrices get complex (1000+ functional units)

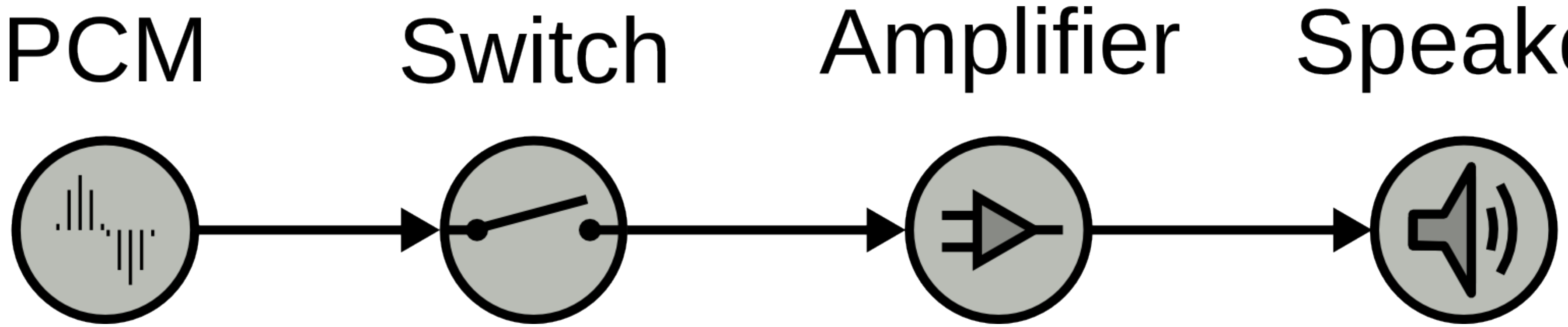
# Why DAPM?

- Battery powered devices require lowest power mode
- Managing dependencies by hand is tedious and error prone

# What is DAPM?

- Models data flow and power dependencies in a directed graph
- Nodes represent functional units (called widgets)
- Edges represent connections between functional units (called routes or paths)

# What is DAPM?




Simple DAPM graph

# What are the benefits of DAPM?

- Provides a common API for audio component interoperability
- Implements efficient power management for individual components

# How does DAPM work?

- CODEC or component driver provides description of it's subsection of the graph
  - Special widgets are used for inputs and outputs
- Board driver describes connections between components as well as the audio fabric
  - Fabric includes speakers, microphones, headphone jacks, etc.
  - Information might be provided by devicetree or ACPI

# How does DAPM work?

- Each widget has a type
  - Speaker, Microphone, Amplifier, DAC, ADC, internal supply, external supply, headphone output, line-in input, line-out output, audio interface, audio interface link, mixer, mux, input pin, output pin
- Type defines how the widget behaves in the graph

# How does DAPM work?

- Detects active data paths
  - Dynamically manages the power state of functional units on those paths
  - Also manages their power dependencies
- Two phases
  - Determine target power state
  - Power sequencing

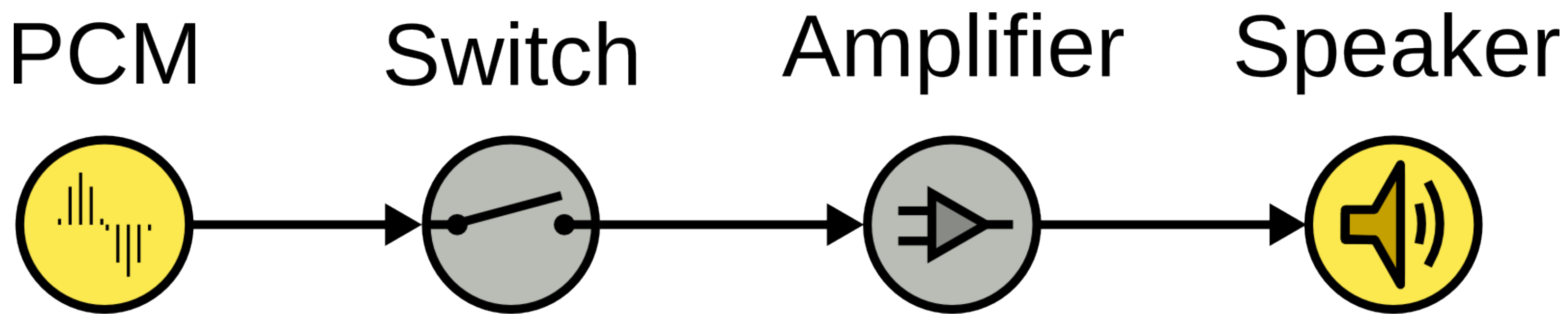
# Phase 1
# Determining Power State

# Categories of Widgets

- For finding out the power state DAPM differentiates between three different categories of widgets
  - Endpoint widgets
  - Pass-through widgets
  - Supply widgets

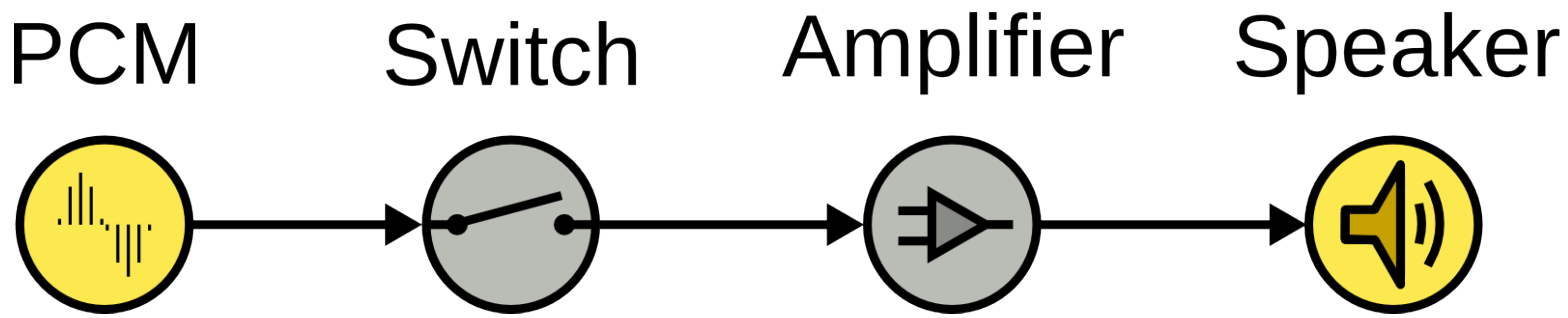
# Endpoint Widgets

- Consume or produce a signal from/into the pipeline
- Speaker, Microphone, Tone-generator, PCM device



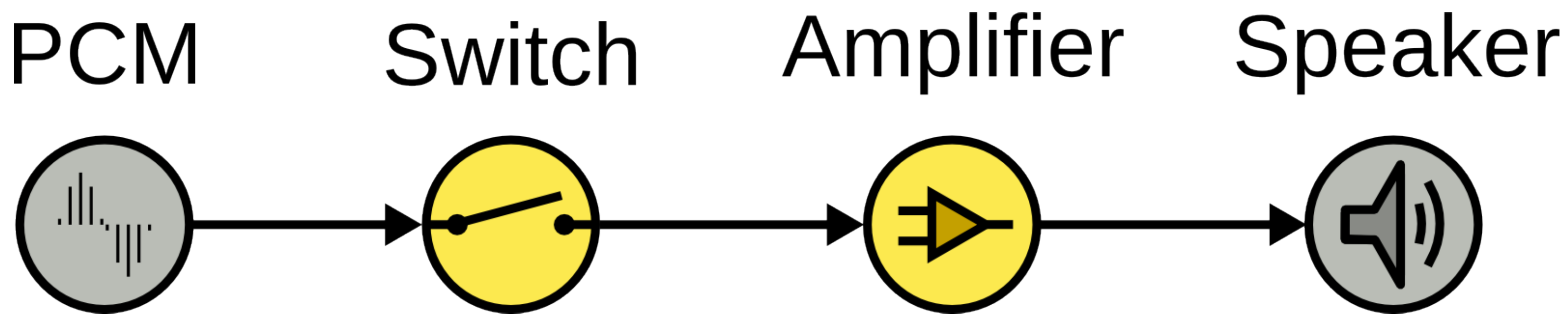
# Endpoint Widgets

- Endpoints can be active or inactive
  - This information is not available for all endpoints
- Endpoints can be marked as disconnected
  - `SOC_DAPM_PIN_SWITCH()`



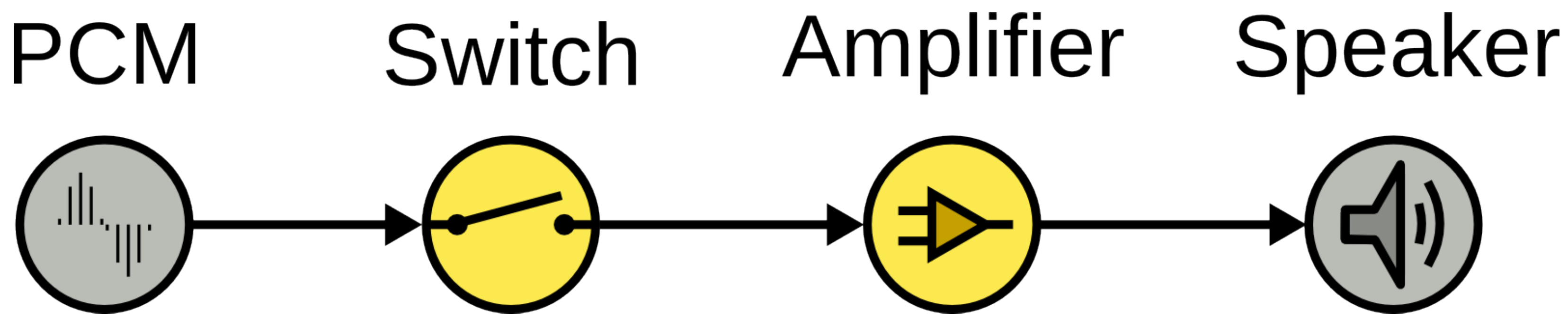
# Pass-through Widgets

- Only powered up when on a active path between two endpoints
- Amplifier, Mixer, Audio-Interface



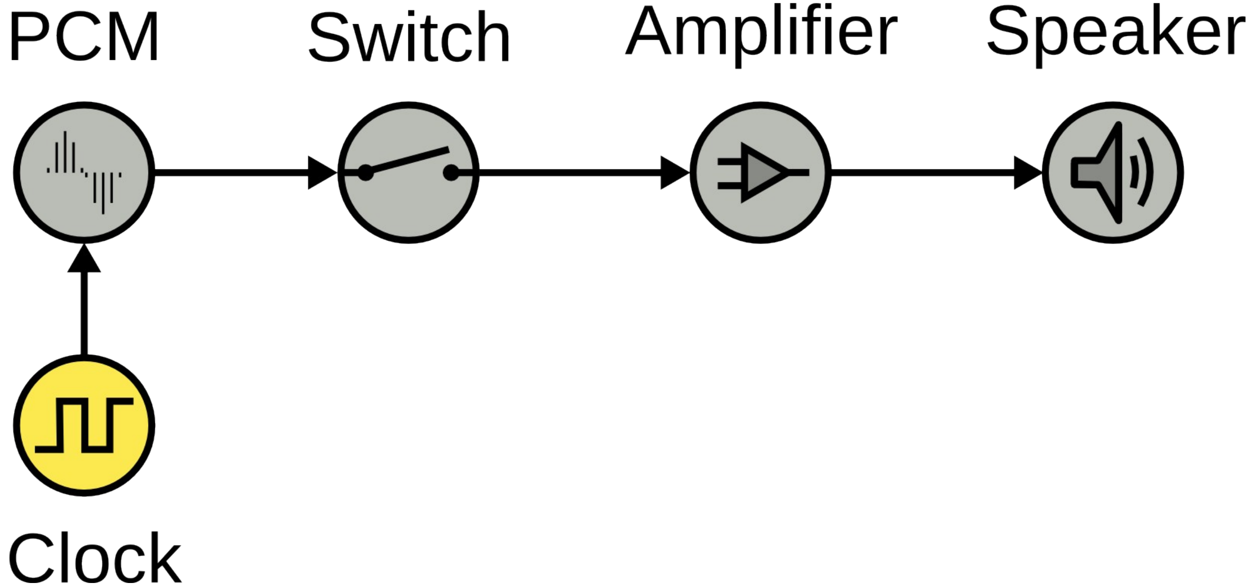
# Pass-through Widgets

- Static routing
  - All inputs contribute to all output signals
- Dynamic routing
  - Connections between inputs and output depend on state



# Supply Widgets

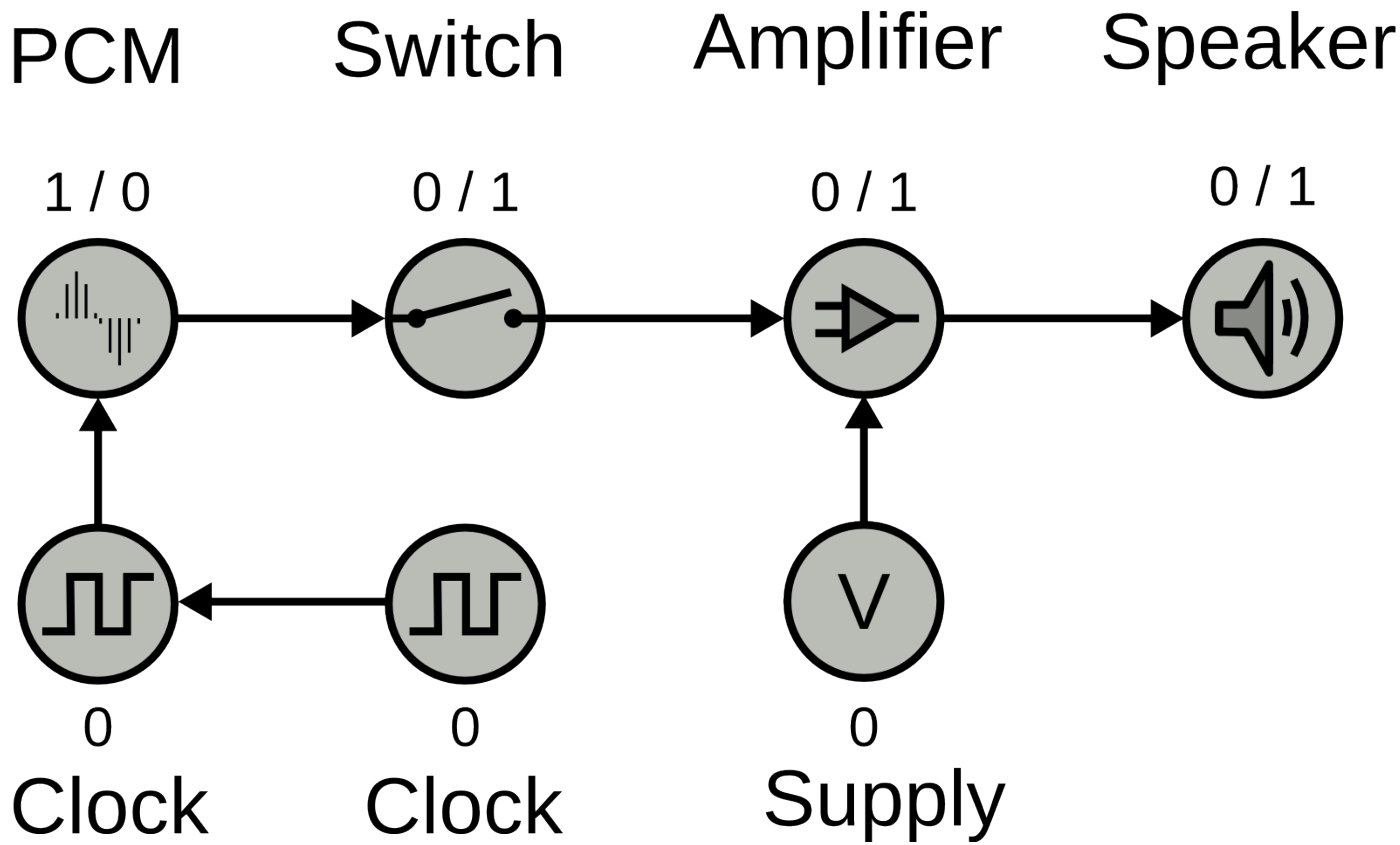
- Model resource dependencies rather than data flow relationships
- Powered up when any of the consumers is powered up
- Clock, regulator, shared enable bits



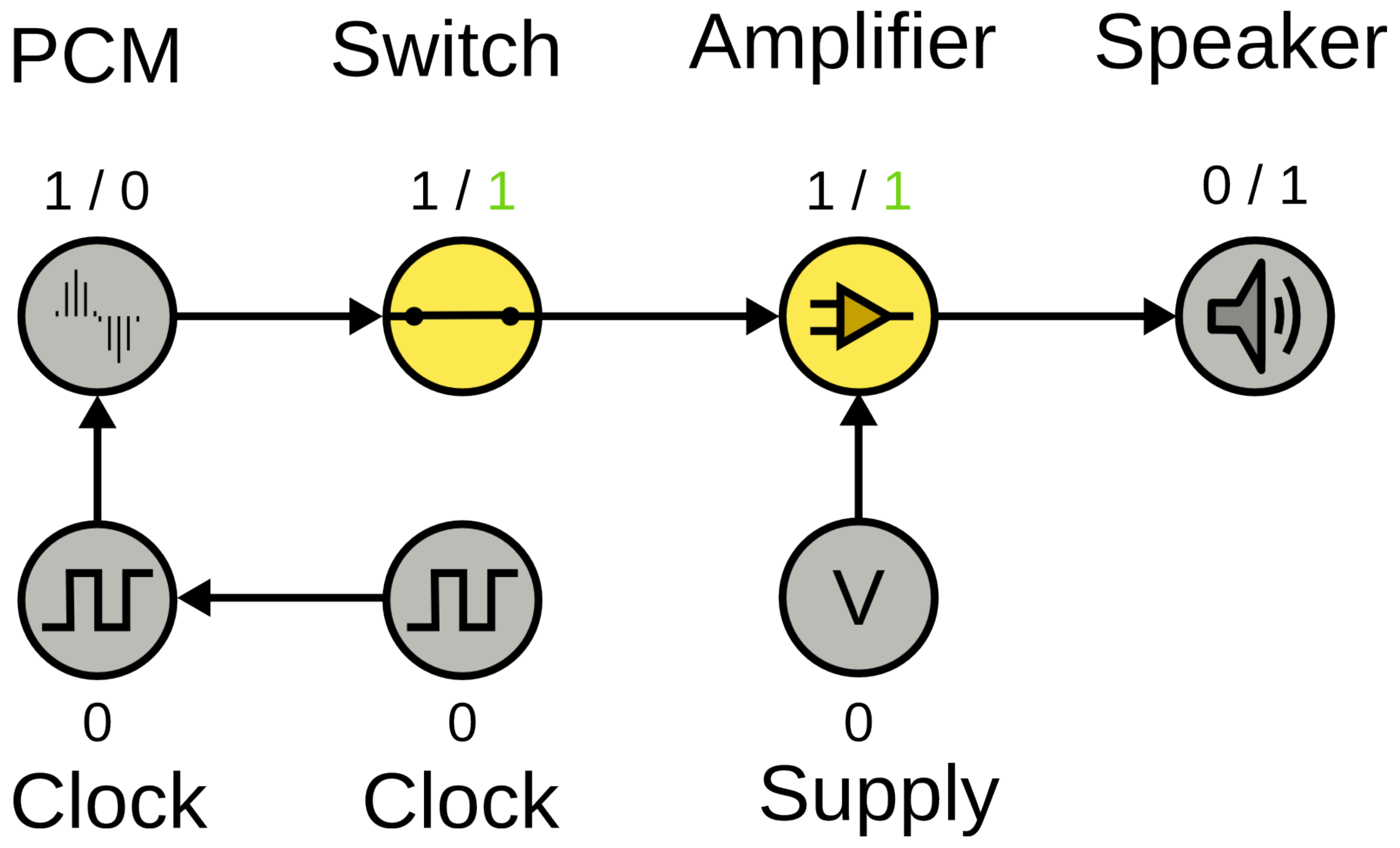
# Determining Power State

- For each widget DAPM records the number of paths to an active output and number of paths to an active input
- If the number of both connected active inputs and connected active outputs is one or more the widget is assumed powered up.

# Determining Power State



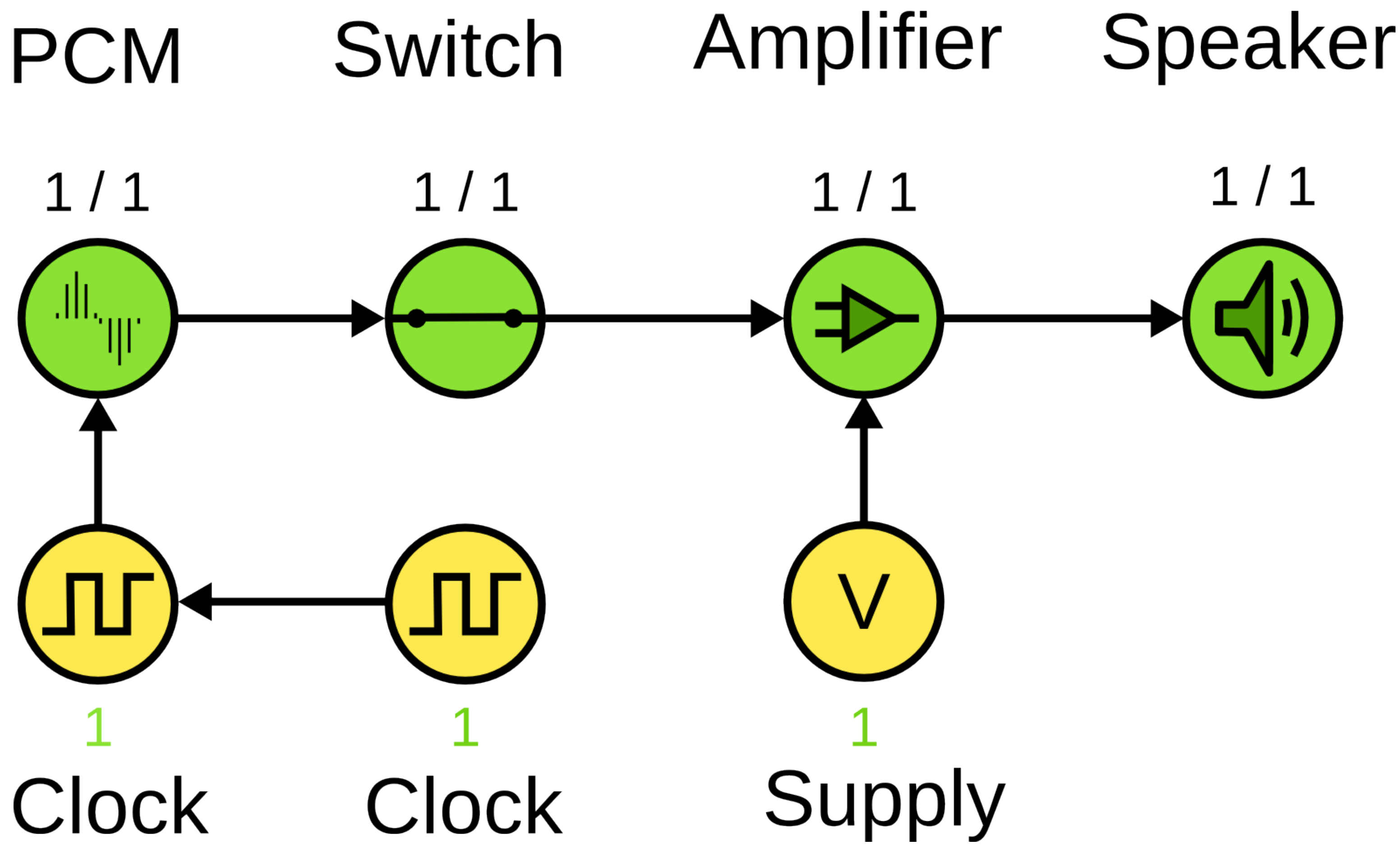
# Determining Power State



# Determining Power State

- Source endpoint widgets are assumed powered up if they are active and there is a path to a active sink endpoint widget
- Sink endpoint widgets are assumed powered up if they are active and there is a path to a active source endpoint widget

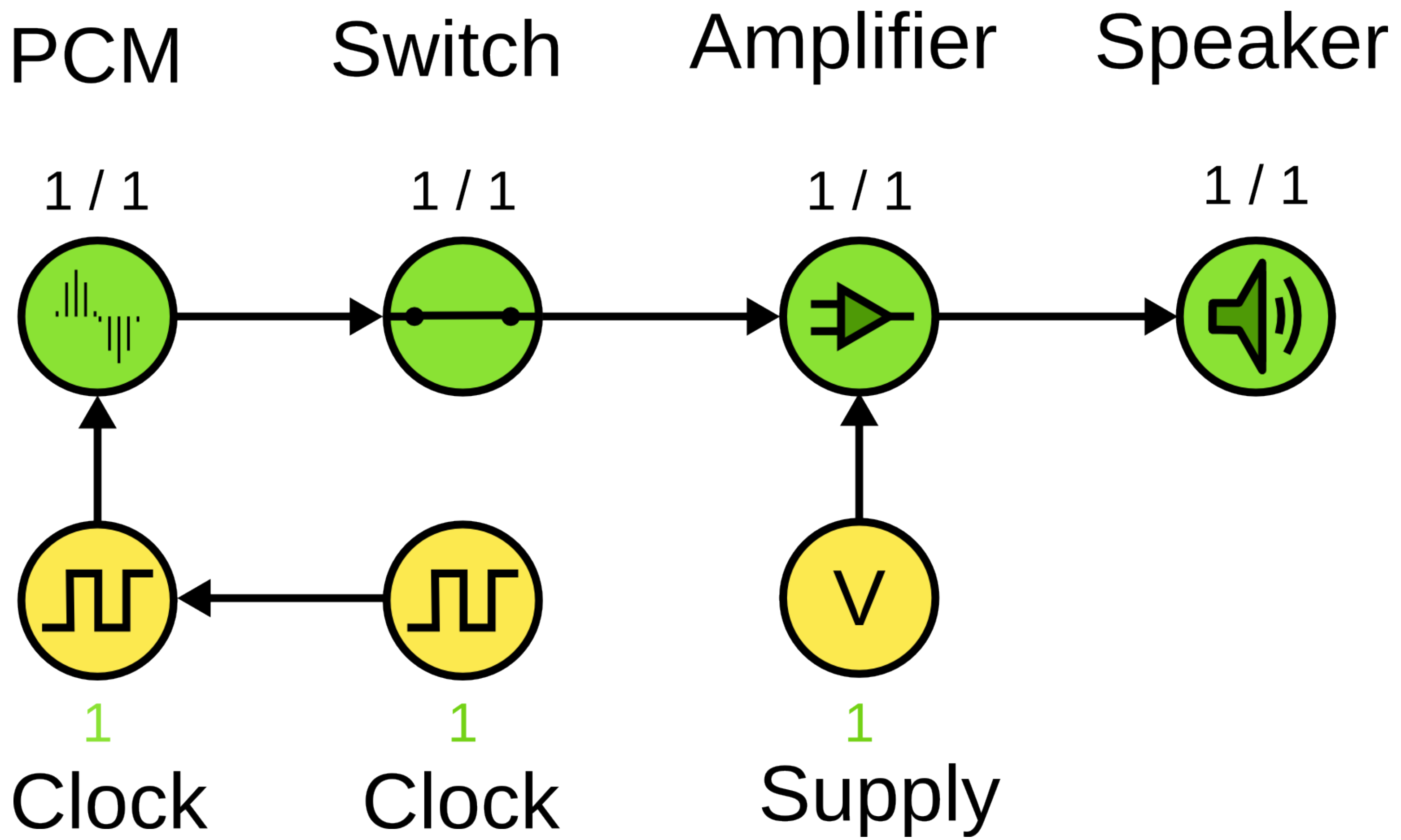
# Determining Power State



# Determining Power State

- Supply widgets are assumed powered up if there is a path to an powered-up widget

# Determining Power State



# Phase 2
# Power Sequencing

# Power Sequencing

- Once the new state has been determined DAPM makes a diff to the current state and schedules the required changes
- Changes are performed in a certain order depending on widget type
  - Minimizes audio click/pop noises

# Powering Sequene

1. Power-down all newly disabled widgets
2. Perform routing changes (if any)
3. Power-up all newly enabled widgets

# Sequencing Order

- Each widget type has a sequence ID
  - Widgets of similar type have the same sequence number
- Power-up sequence order is not the reverse power-down sequence order
- Each widget can have a sub-sequence ID
  - For ordering within the same sequence

# Sequencing Order

- Power updates are order by
  - Widget type sequence ID
  - Widget sub-sequence ID
  - IO register access
  - DAPM context (device)

# Applying Power Changes

- DAPM has the concept of register mapped IO built-in
  - Widget specifies register offset, a mask and a value for the on state and off state
- Per widget callbacks are also available
  - For external supplies
  - For widgets internal widgets that require a more complex on/off register write sequence

# Register Update Coalescing

- Multiple updates to the same register in the same sub-sequence are coalesced into a single update
- Reduces the number of IO operations
  - Important for slow buses like I2C

# Dynamic Graph Changes

# Dynamic Graph Changes

- DAPM has support for dynamic graph changes
- After each change the power state of the graph is re-evaluated

# Dynamic Graph Changes

- Enable/disable (add/remove) a edge in the graph
  - Dynamic routing changes
- Enable/disable a endpoint node in the graph
- Starting/Stopping a playback or capture stream
- Hot-plug/-unplug of components
  - Poorly supported at the moment

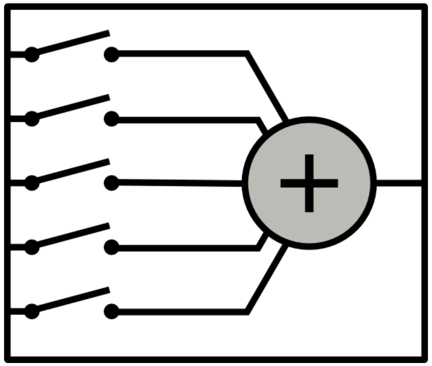
# Dynamic Routing Changes

- DAPM has built-in support for common types of dynamic routing changes
  - Mixers, Mux, Demux
- Driver can implement their own dynamic routing when necessary
  - Typically used when different operating modes require different routing

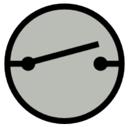
# Mixer

- Has multiple input paths that can be independently enabled/disabled
- Output is the sum of all inputs
- Exported to userspace using multiple boolean ALSA controls

Mixer

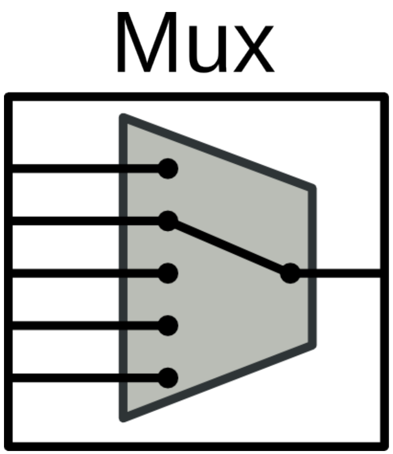


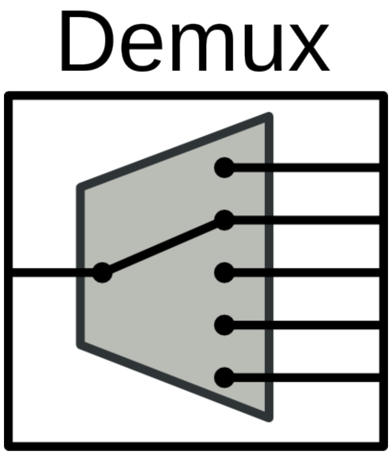

Switch



# Mux/Demux

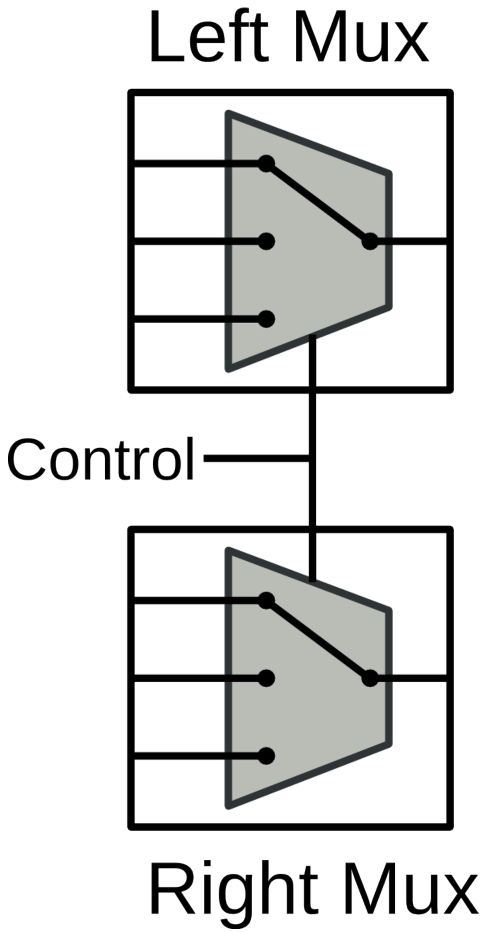
- Mux: Routes one of multiple inputs to a single output
- Demux: Routes one input to exactly one of multiple outputs
- Exported to userspace using a single enum control



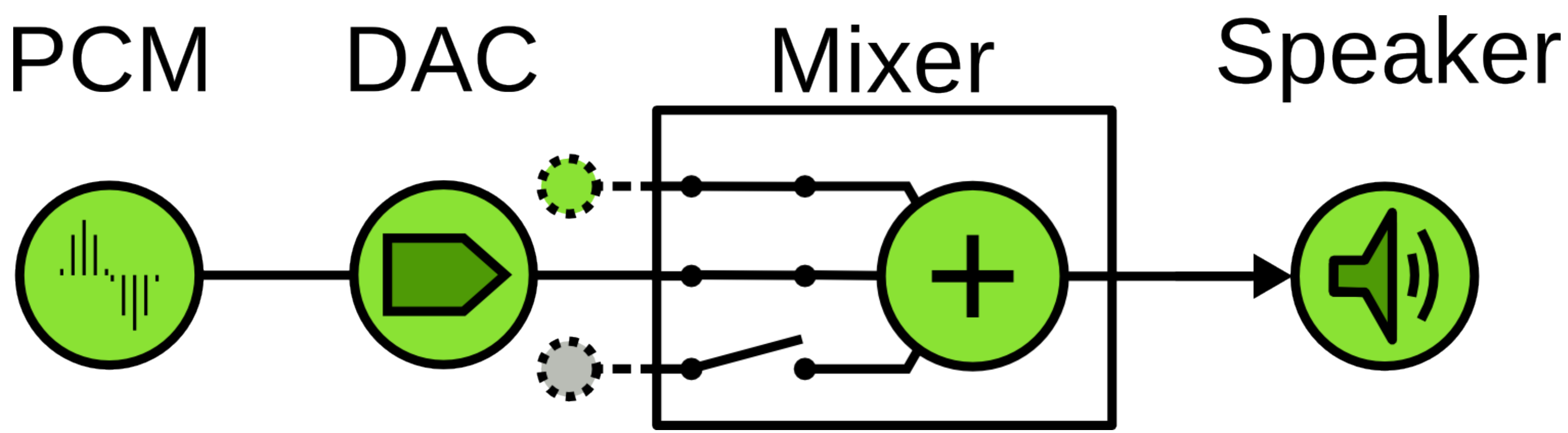
# Shared Mixers/Muxes

- Allow to model independent data flow paths with shared control path
  - E.g. left and right path of a stereo signal
- In the driver pass the same struct snd_kcontrol_new to all controlled mixers/muxes



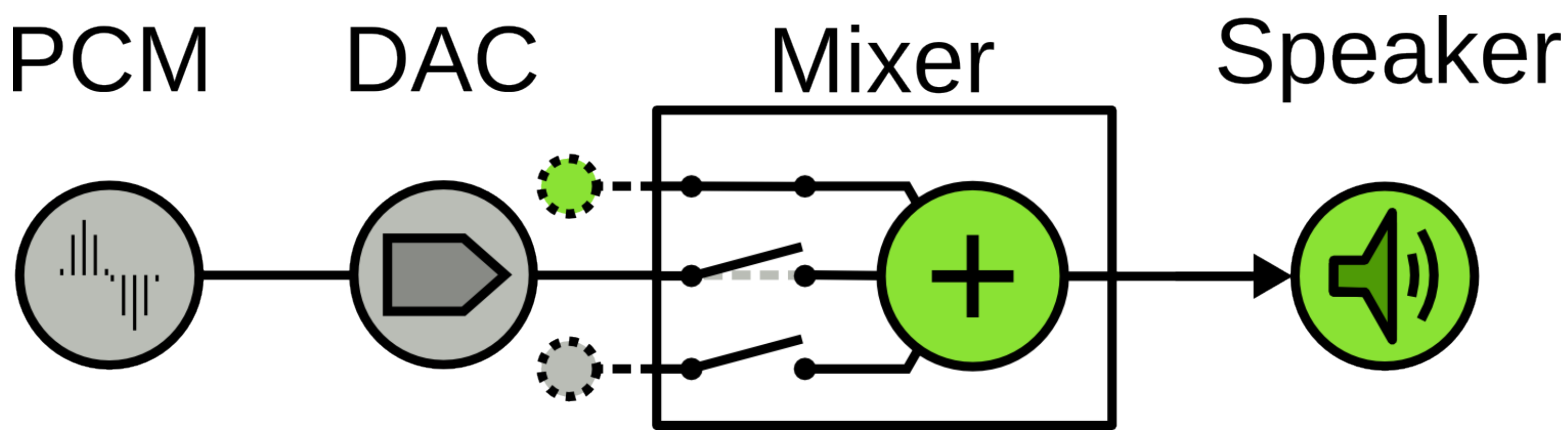
# Auto-mute Mixers

- Automatically mutes/disables the input to a mixer source is powered down
- Useful when the source outputs a invalid or undefined signal when powered down



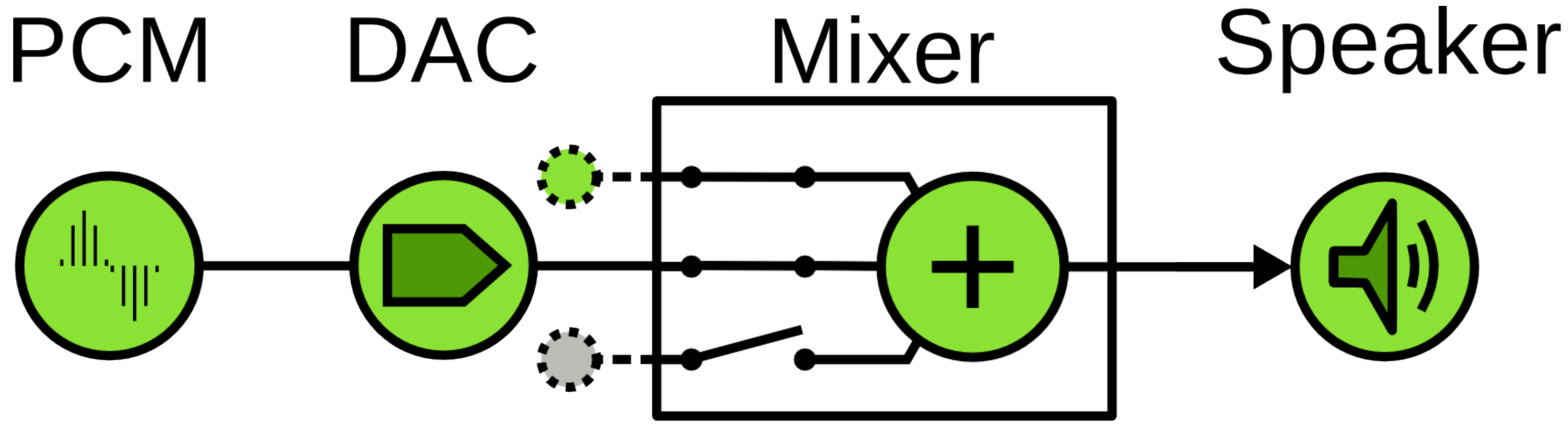
# Auto-mute Mixers

- When the source stops the switch is automatically opened
- Switch state is still reported as closed to userspace applications



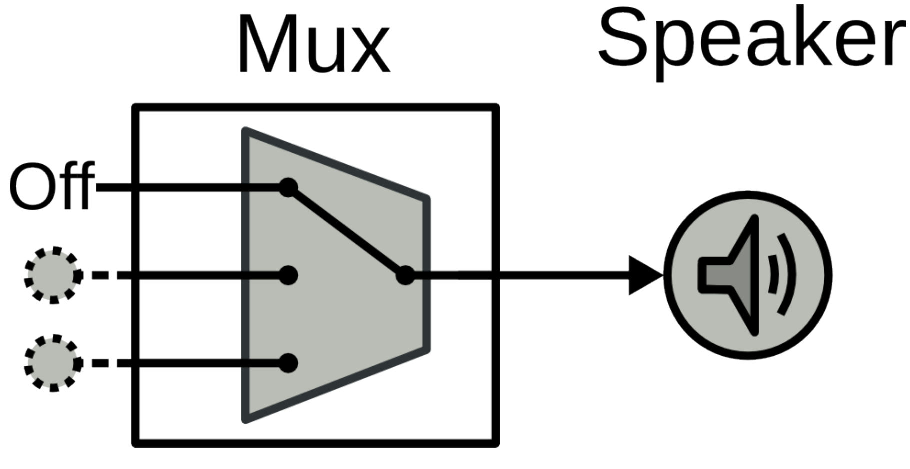
# Auto-mute Mixers

- When the source resumes the switch is set back to the userspace provided setting



# Auto-disable Mux

- When the selected source is powered down the mux switches to a special off state
- Useful when the source output is undefined or invalid when powered off
- Useful when the mux has no dedicated power-down control



# Future

# Future - DXPM

- Using DAPM not only for audio
  - E.g. video processing pipelines
- Allows to model complex power relationships
- Doesn't suffer problems of classical power runtime power management
  - E.g. DAPM can handle cyclic dependencies
  - Finer grained resolution
- DAPM core algorithm is not audio specific

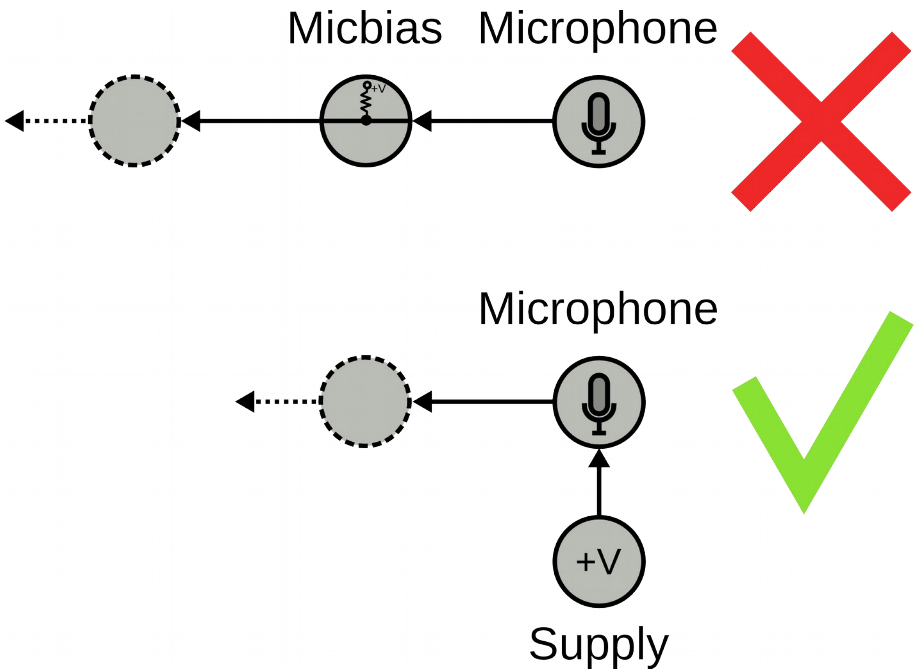
# Q/A

# Thanks

# Bonus Slides

# Micbias Widget

- Conceptually broken
- Don't use them
- Use supply widgets instead



# Jack Detection

- DAPM has jack detection integration
- Automatically disables endpoint when nothing is connected

# Suspend/Resume

- During system suspend all endpoints are marked as disconnected
  - Unless the are marked to ignore suspend

# Runtime Suspend/Resume

- DAPM integrates nicely with runtime PM
- Runtime PM is enabled when at least one widget is enabled
- Runtime PM is disabled when all widgets are disabled
- Don't access the same hardware state from DAPM and runtime PM

# Pre/Post widgets

- Pre/Post widgets are special virtual widgets
- Callbacks are executed each time the DAPM sequencing runs
- Don't need to be connected anywhere