arm

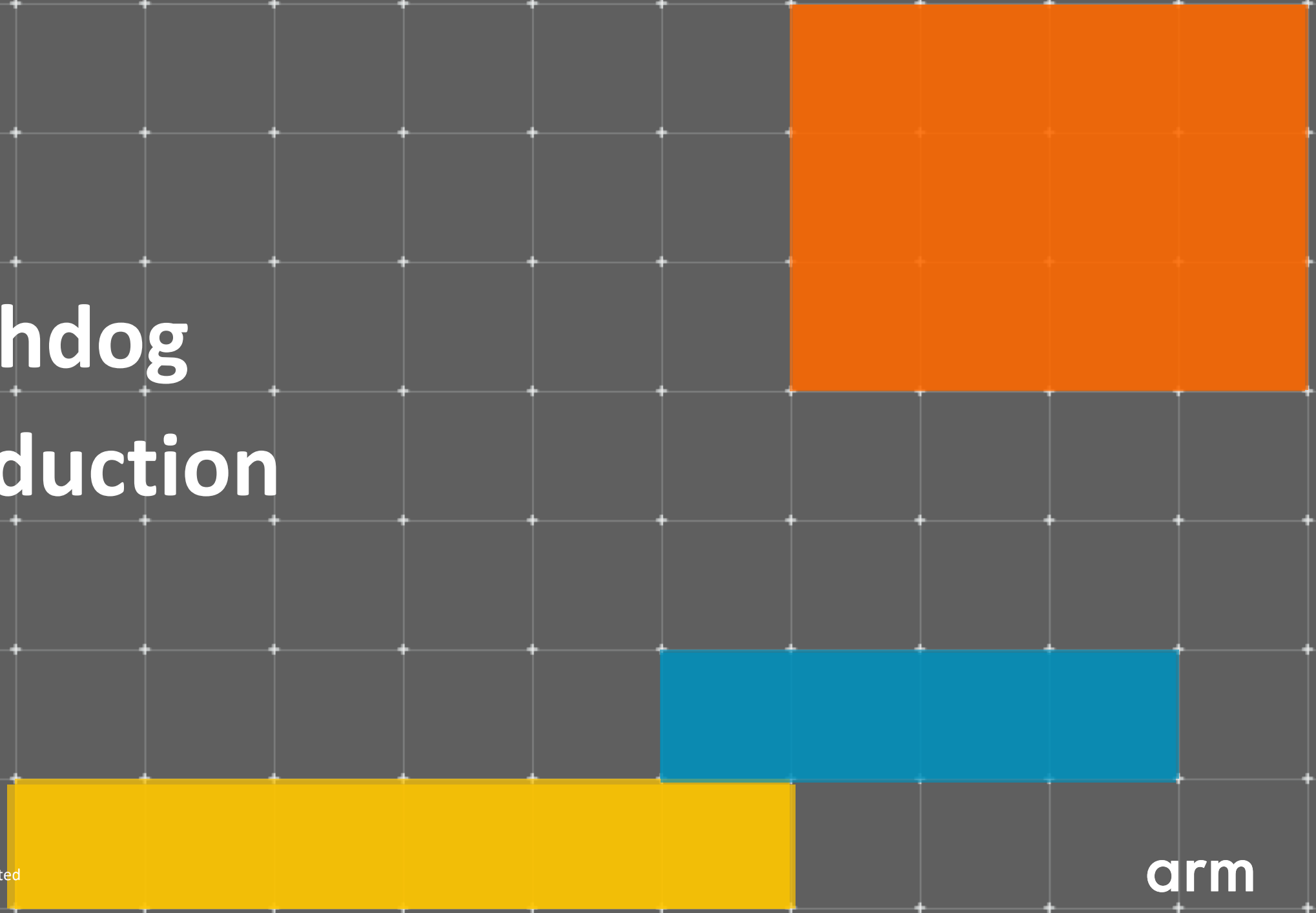
Mbed Silicon Partner Workshop

Watchdog and ResetReason

27th of March 2018

Paris, France

Watchdog Introduction

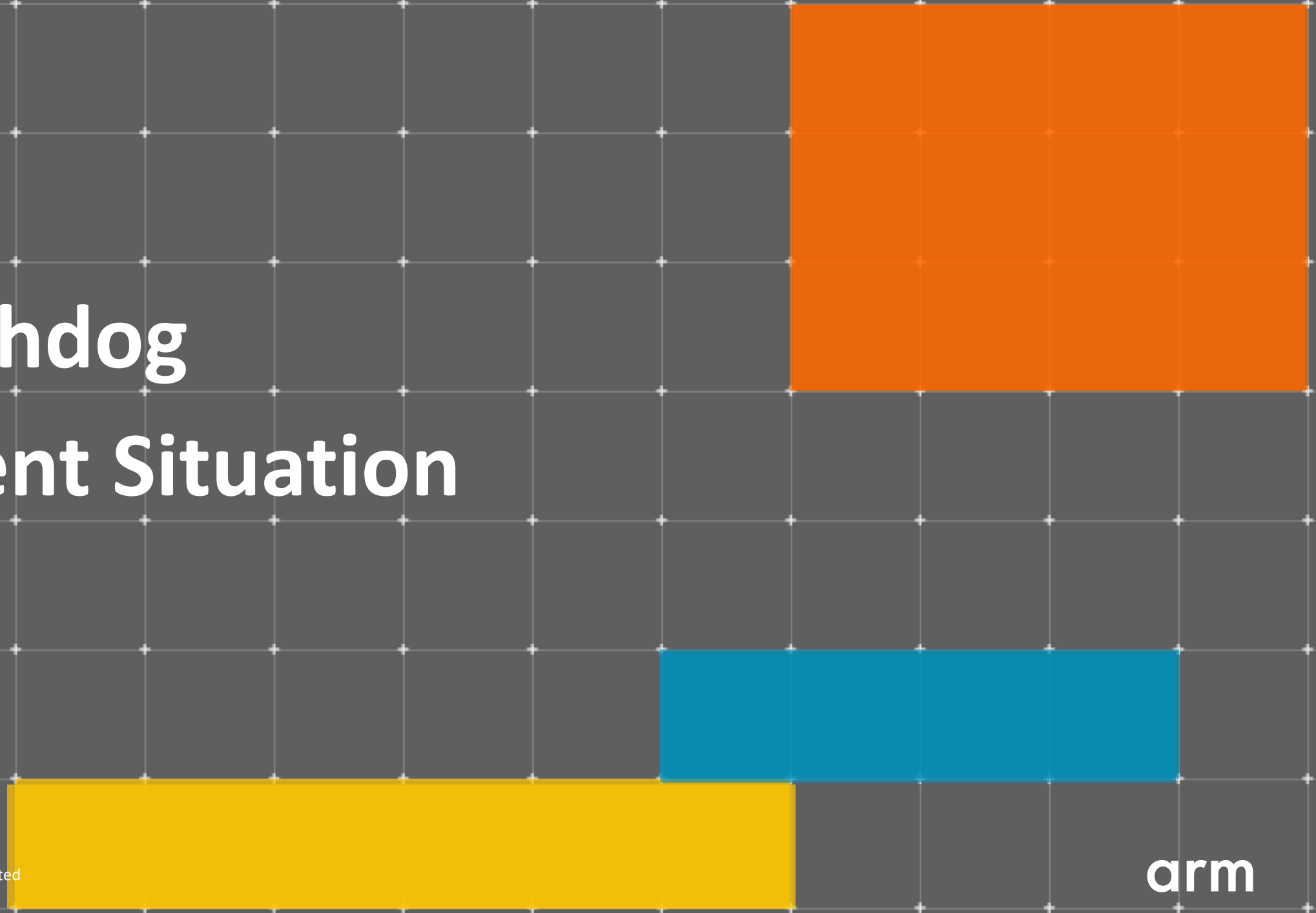


arm

Watchdog Introduction

- A watchdog timer is a dedicated piece of hardware that is used to monitor execution of a microcontroller and reset the system if it enters into an invalid state.
- The timer is configured to countdown from a specific value, upon reaching zero the system is reset.
- It must be periodically “kicked” to reset the countdown value and prevent the board from resetting.
- `Watchdog` API allows user to configure and run the Watchdog Timer in a platform-independent way.

Watchdog
Current Situation



arm

What does the new implementation do?

- Provides functions to manage a simple independent Watchdog Timer.
  - Watchdog can be started by specifying a timeout value in milliseconds.
  - Watchdog can be refresh to it’s configured reset value.
  - Watchdog can be stopped while running, depending on the hardware it's running on.
- `Watchdog` API supports timeout values between 1ms and 47 days. Actual maximum timeout is specific to each microcontroller.
- Support for single independent watchdog timer only, no support for Window Mode.

Independent mode

- Currently only Independent mode is supported by the API as it's the only mode supported across all microcontrollers.
- Independent mode is sufficient in most common use case of a Watchdog Timer.
- Windowed mode would allow adding constraints to when a watchdog could be kicked, for example kicking a watchdog twice in quick succession would trigger a reset.
- This may be useful in situation where the application is stuck in a loop that includes kicking the watchdog, but in quick succession.

Supported timeout ranges

- The `Watchdog` API supports a range of values between 1 millisecond and UINT32_MAX milliseconds (~49 days)
- The actual maximum supported by a microcontroller varies. The API provides a function to query the microcontrollers features and/or maximum value.
- No hard limit is specified because the maximum supported timeout some microcontrollers can be as low as ~34 seconds.
- Use can query the current microcontroller for it's maximum supported timeout
- Attempting to start a Watchdog timer with an invalid Watchdog timeout will return `WATCHDOG_STATUS_INVALID_ARGUMENT`.

Watchdog API supports stopping a running Watchdog Timer

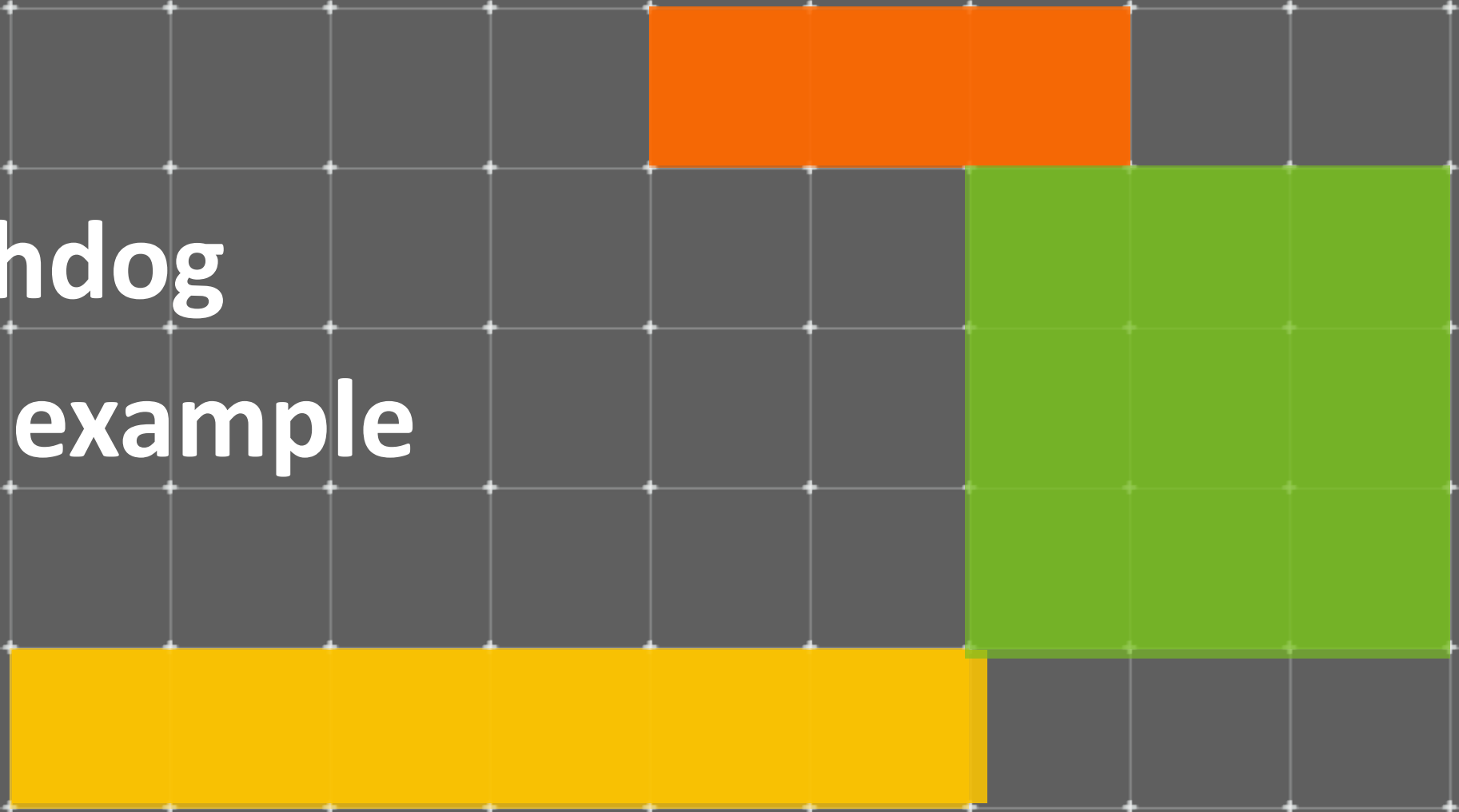
- API provides a function for stopping a running Watchdog Timer with `Watchdog::stop`
- The functionality is not supported by all boards.
- Boards that do not support stopping a running timer must return `WATCHDOG_STATUS_NOT_SUPPORTED`.

```
watchdog watchdog;
watchdog.start(5000);
const watchdog_status_t status = watchdog.stop();
if (status != WATCHDOG_STATUS_SUCCESS) {
    printf("Stopping a running timer is not supported.\r\n");
}
```

Watchdog Timer is decoupled from Deep Sleep

- Watchdog Timer will run regardless of the state of the microcontroller. This includes running whilst in sleep, deep sleep, and debug modes.
- There is currently no method of pausing the watchdog timer using the `Watchdog` API while in these modes.
- Most microcontrollers can be configured to pause during these modes, but this is not yet supported by the user facing API.

Watchdog
Code example



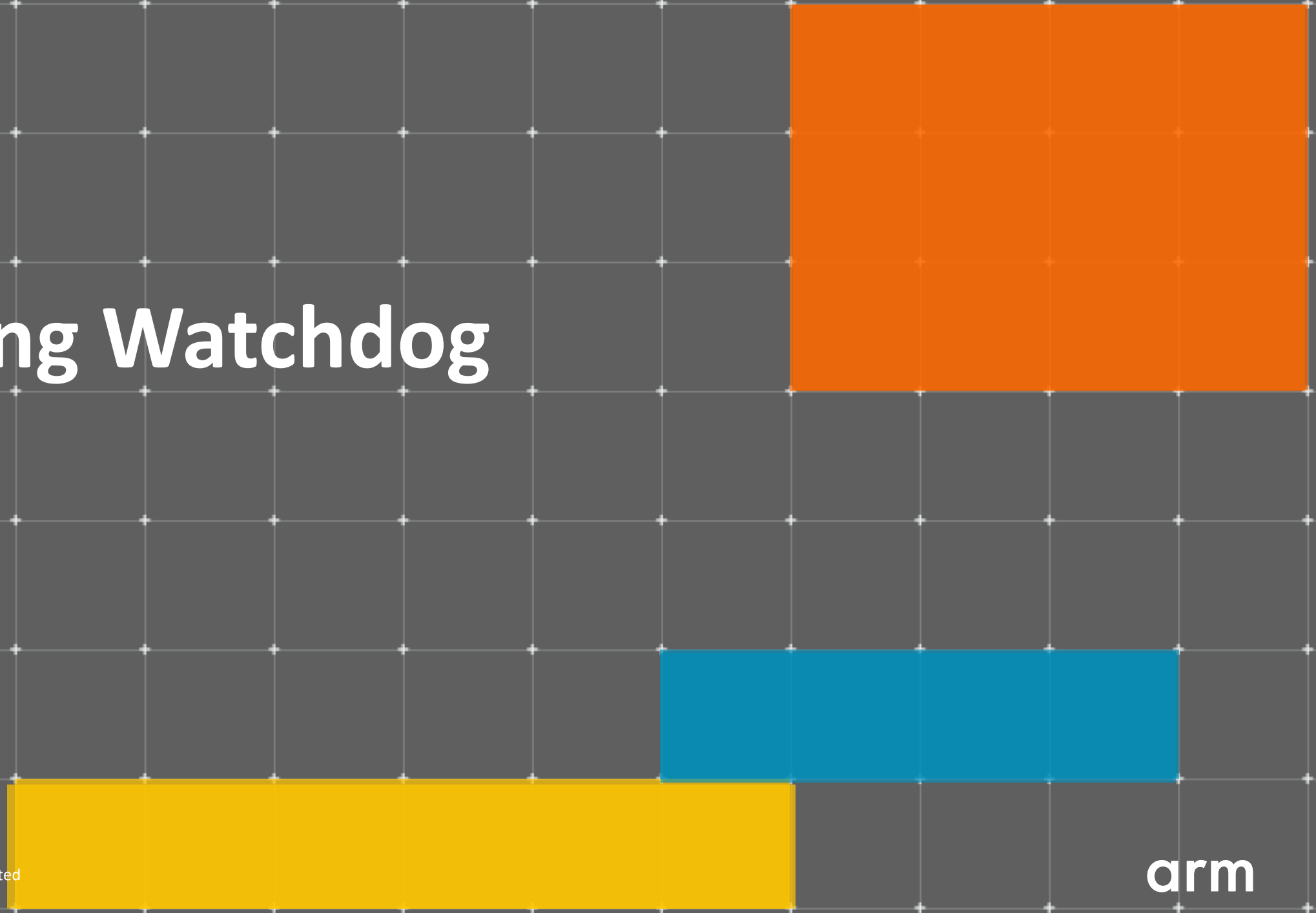
arm

Simple example

```
Watchdog watchdog;
InterruptIn interrupt(BUTTON1);

void trigger() {
  printf("Button pressed, refreshing watchdog timer\r\n");
  watchdog.kick();
}

int main() {
  printf("Starting watchdog timer with 5 second timeout.\r\n");
  watchdog.start(5000);
  interrupt.rise(&trigger);

  wait(60 * 60 * 3);
  watchdog.stop();
  printf(“Congratulations\r\n");
}
```


arm

Porting Watchdog



arm

Implementing HAL functions

- Add `WATCHDOG` to `device_has` in targets.json

  https://github.com/ARMmbed/mbed-os/blob/master/targets/targets.json

```
"LPC1768": {
    "inherits": ["LPCTarget"],
    "core": "Cortex-M3",
    "extra_labels": ["NXP", "LPC176X", "MBED_LPC1768"],
    "supported_toolchains": ["ARM", "uARM", "GCC_ARM", "GCC_CR", "IAR"],
    "detect_code": ["1010"],
    "device_has": ["ANALOGIN",..., "WATCHDOG"],
    "release_versions": ["2", "5"],
    "features": ["LWIP"],
    "device_name": "LPC1768",
    "bootloader_supported": true
},
```


arm

Implementing HAL functions

- Implement the watchdog functions in mbed target:

  https://github.com/ARMmbed/mbed-os/blob/feature-watchdog/hal/watchdog_api.h

```
watchdog_status_t hal_watchdog_init(const watchdog_config_t *config);
void hal_watchdog_kick(void);
```

`hal_watchdog_init()`

- Initialise a watchdog timer with the given configuration.
- Currently the configuration consists only of a timeout value
- Return `WATCHDOG_STATUS_OK` if watchdog timer starts successfully

`hal_watchdog_kick()`

- Refreshes the watchdog timer before it times out
- If a watchdog is not running, it does nothing

Implementing HAL functions

```
watchdog_status_t hal_watchdog_stop(void);
uint32_t hal_watchdog_get_reload_value(void);
watchdog_features_t hal_watchdog_get_platform_features(void);
```

`hal_watchdog_stop()`

- Stops the watchdog timer.
- If the platform does not support stopping this will return `WATCHDOG_STATUS_NOT_SUPPORTED`

`hal_watchdog_get_reload_value()`

- Get the configured watchdog timer refresh value

`hal_watchdog_get_platform_features()`

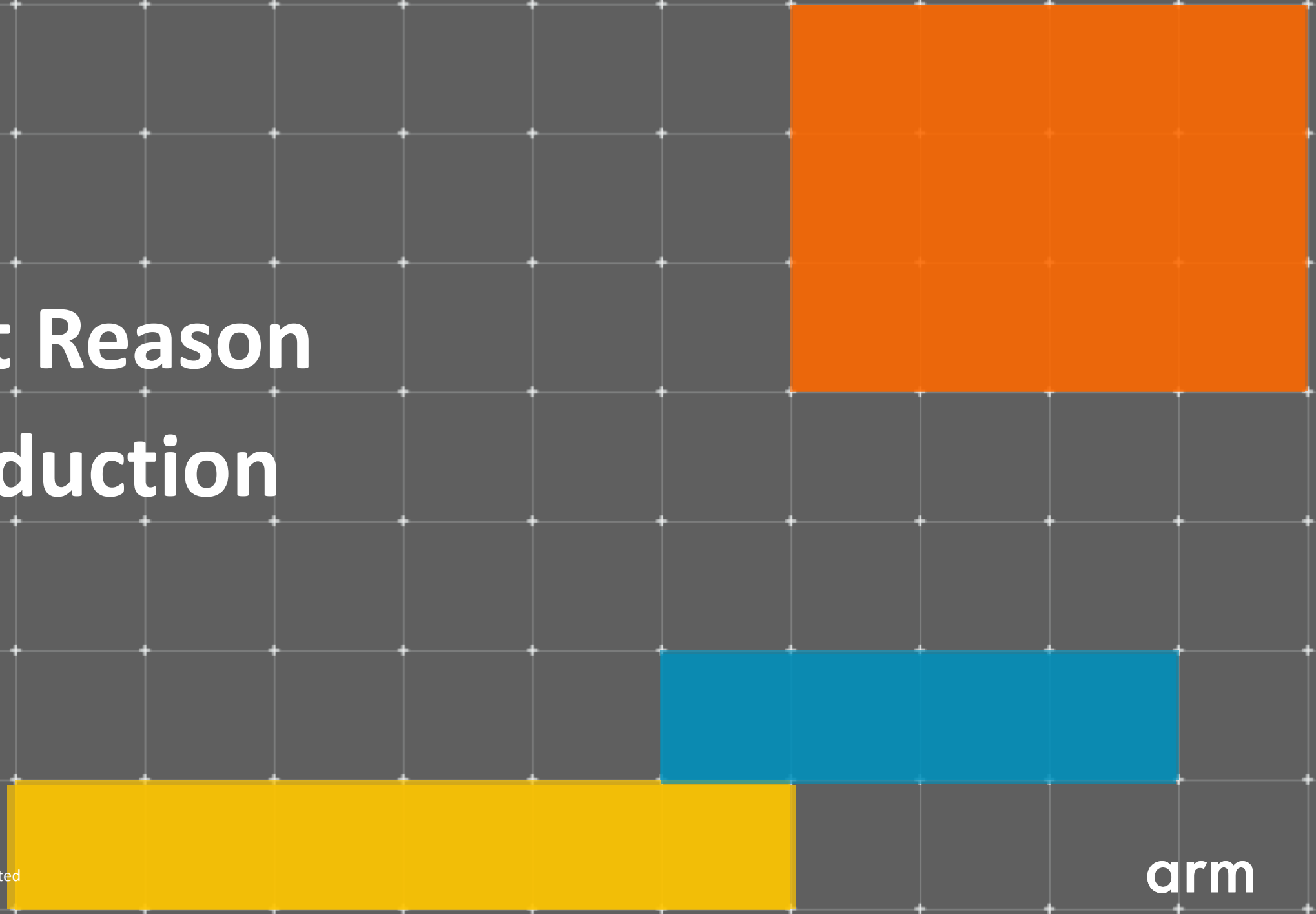
- Get information on the current platforms supported watchdog functionality
- Max timeout supported by the platform.
- Flag indicating that the platform supports stopping

Running Tests

- Watchdog validation suite:

```
$ git checkout feature-watchdog
$ mbed test -t <toolchain> -m <target> -n "tests-mbed_hal-watchdog"
```

Reset Reason Introduction

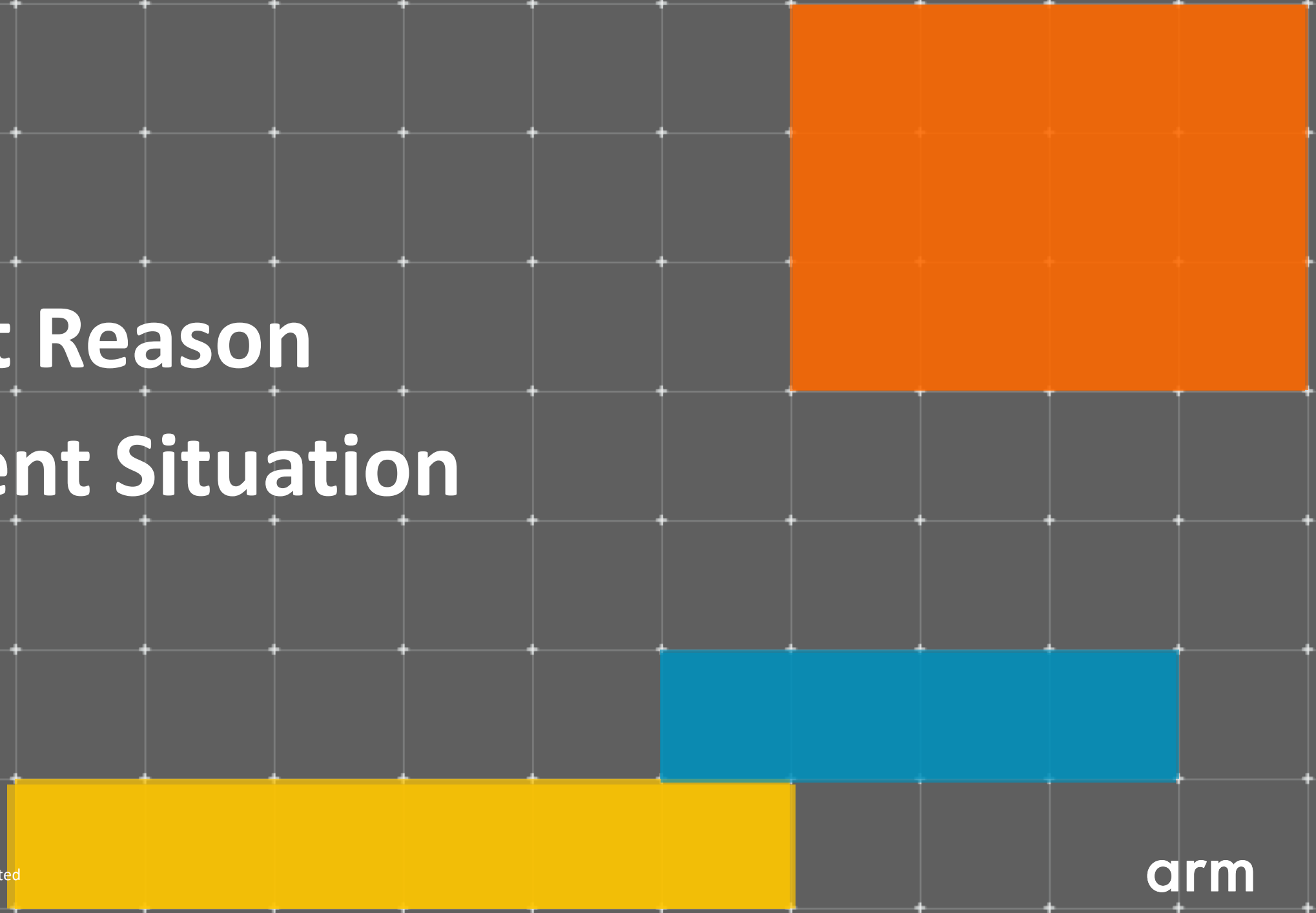


arm

What is the ResetReason API

- When a microcontroller resets, the reason for the reset is set in a bit field within a special register (usually `RESETREAS`) that is readable at runtime.
- mbedOS has previously had no platform independent method to fetch the reset reason from a microcontroller's registers.
- `ResetReason` API provides a method to do this by mapping microcontroller specific reset reasons to portable Mbed OS reset reasons.
- Used in conjunction with a Watchdog it can be used to determine if the last system reset was caused by the running Watchdog timer.

Reset Reason
Current Situation



arm

What does the new implementation do?

- Provides two functions for accessing the underlying reset reason.
  - `ResetReason::get()` returns an enum containing a platform-independent representation of the board specific reset reason.
  - `ResetReason::get_raw()` returns the underlying microcontroller specific reset reason represented as an unsigned 32-bit value.
- Calling either function clears the microcontrollers `RESETREAS` register after reading to keep functionality the equivalent across all platforms.
- To make the API safe to call multiple times each function will cache the returned enumeration and `RESETREAS` values before clearing.

arm

Driver caches the reset reason

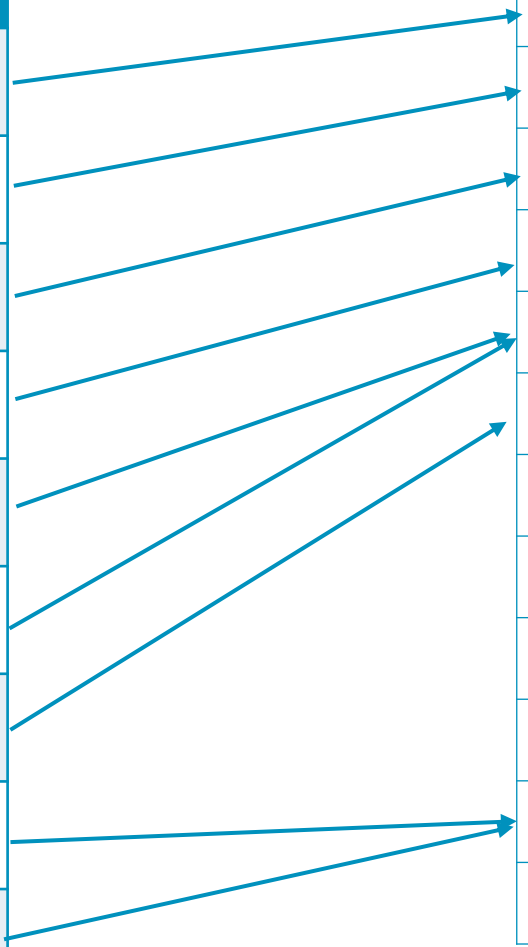
- Some microcontrollers do not clear reset reasons between resets. Reset reasons will accumulate in the `RESETREAS` bitfield and the last reason will become impossible to derive the most recent reset reason.
- To support multiple calls to the `ResetReason` API it will cache (as a static variable within the API function) the reset reason on first read before clearing the hardware register.
- The `ResetReason` enum will be cached, as well as the raw value of the register stored as a static variables in the Driver API.

Architecture

- The `ResetReason` user facing API will handle the caching and clearing of the reset reason registers.
- The HAL implementation is responsible for:
  - Mapping microcontroller specific reset reasons to Mbed OS portable reset reasons.
  - Marshalling the `RESETREAS` register into a 32-bit unsigned integer.
  - Implementing function to clear the `RESETREAS` register when called.

Enum return, per board mapping

Device Reset Reason
Hardware - Power On
Hardware - Reset Pin
Hardware – Brown Out
Software Reset
Watchdog - Independent
Watchdog - Window
Device - Core Lockup
Device - Loss of Clock
Debug - JTAG Generated Reset



mbed HAL (reset_resaon_api.h)
RESET_REASON_POWER_ON
RESET_REASON_PIN_RESET
RESET_REASON_BROWN_OUT
RESET_REASON_SOFTWARE
RESET_REASON_WATCHDOG
RESET_REASON_LOCKUP
RESET_REASON_WAKE_LOW_POWER
RESET_REASON_ACCESS_ERROR
RESET_REASON_BOOT_ERROR
RESET_REASON_MULTIPLE
RESET_REASON_PLATFORM
RESET_REASON_UNKNOWN

arm

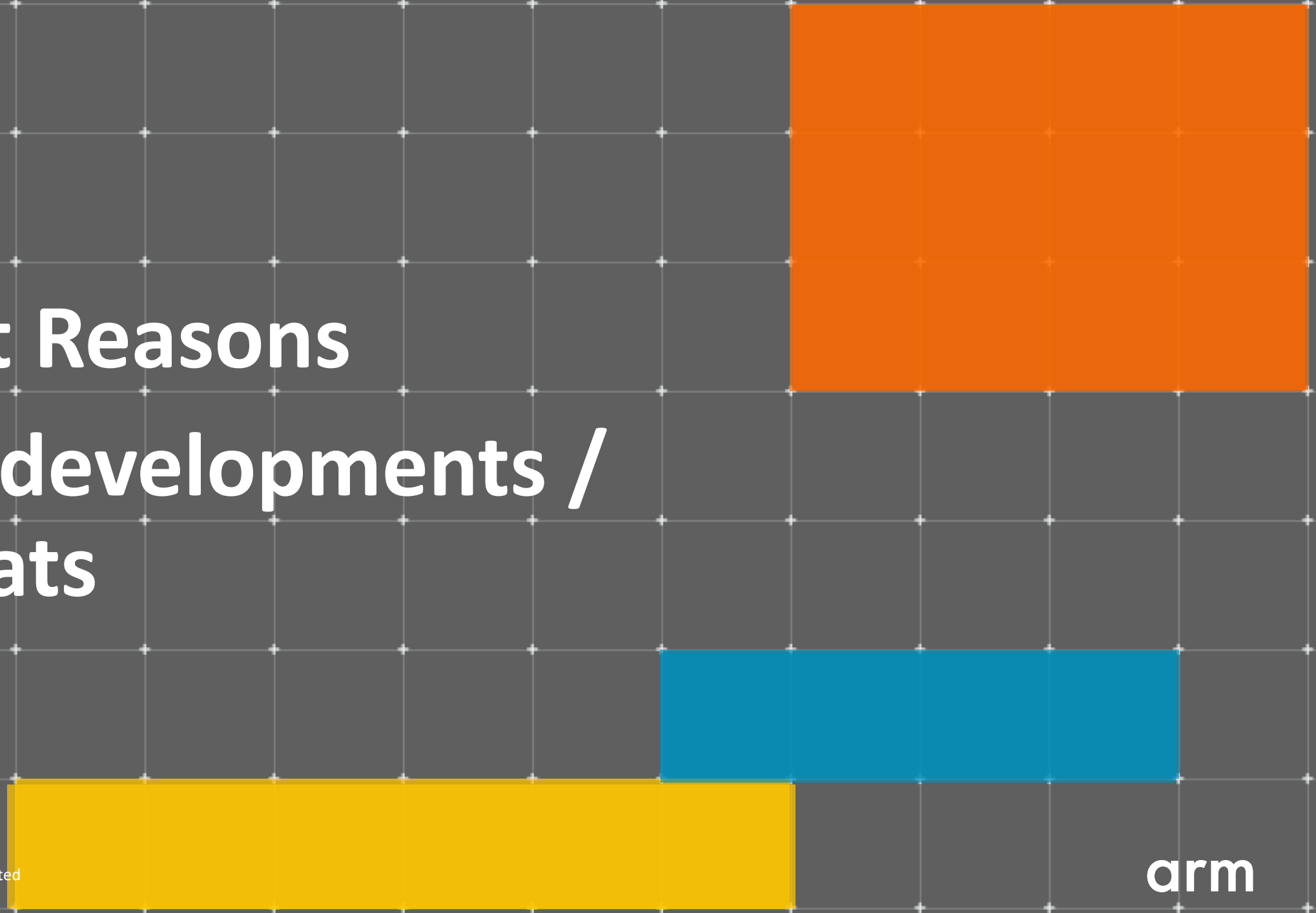
Handling unmappable reset reasons

- `ResetReason` API provides a function to fetch the microcontroller specific reset reason as a platform-independent enumeration
- If a platform has a reset reason that does not correspond to an appropriate MbedOS reset reason it will be set to `RESET_REASON_PLATFORM`
- User can call `ResetReason::get_raw()` to query the underlying hardware `RESETREAS` register, from which the user can determine the actual reset reason if they know the platform they are currently running on
- The user is not able to use the board API to get it manually, as the act of calling the `ResetReason` API will clear the underlying `RESETREAS` register.

arm

Reset Reasons

New developments / Caveats



arm

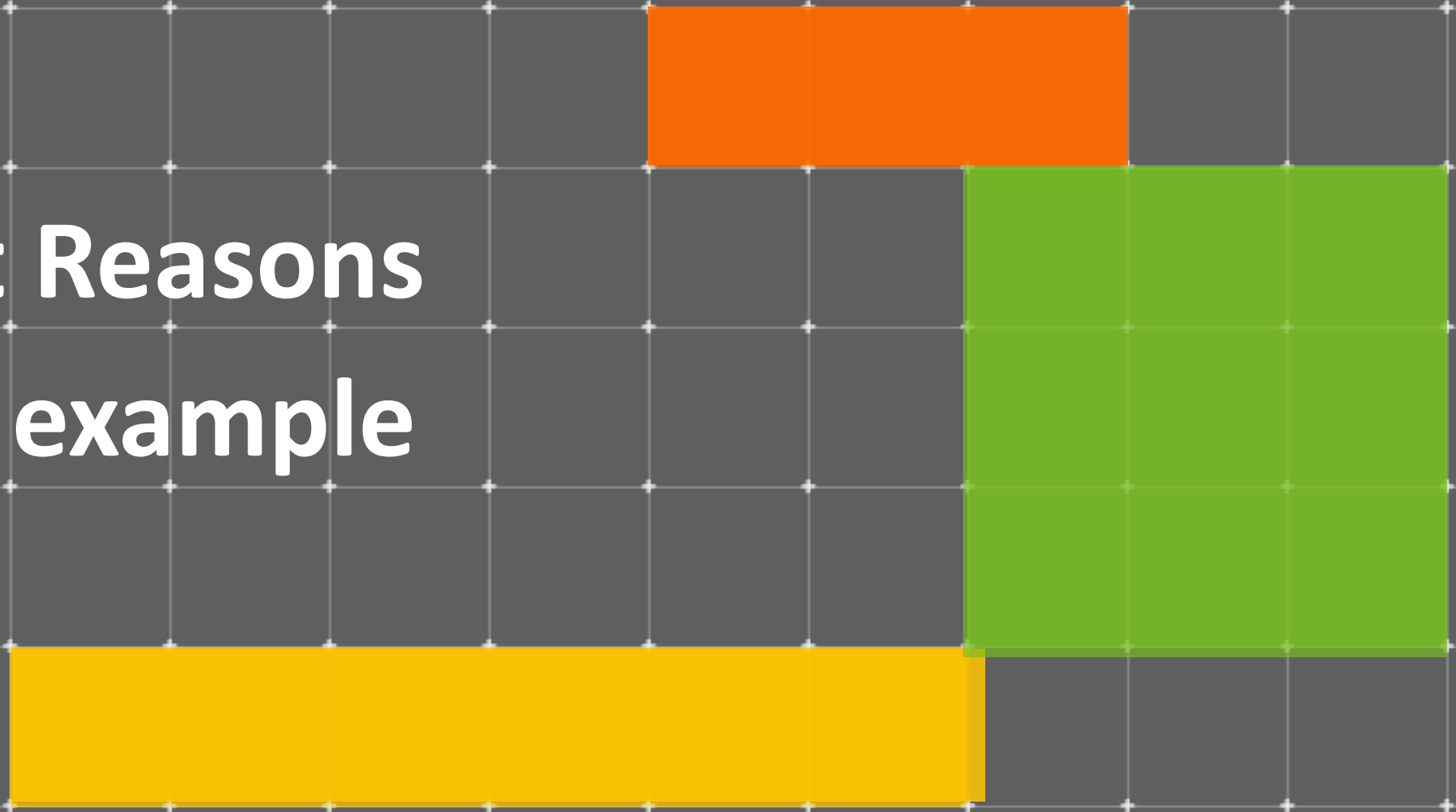
Can't currently be used by Bootloader and Application

- The caching of the `ResetReason` register does not persist between running the bootloader and running the application.
- If the bootloader uses the `ResetReason` API, the API will not be useable from the Application as the registers will be cleared but no cached value will exist.
- In this situation the `ResetReason` API will return `RESET_REASON_UNKNOWN` as no bits will be set in the `RESETREAS` register.
- This may be solved in future by storing the cached value in a less volatile location.

Multiple set reset reasons

- The `ResetReason` API will clear `RESETREAS` registers automatically due to some platforms persisting reasons between reboots
- If the `ResetReason` API is not called during runtime the register will not be cleared
- For platforms that persist reset reasons this may cause multiple flags to be set at boot
- In this case the HAL implementation should return `MBED_RESET_REASON_MULTIPLE`

Reset Reasons Code example



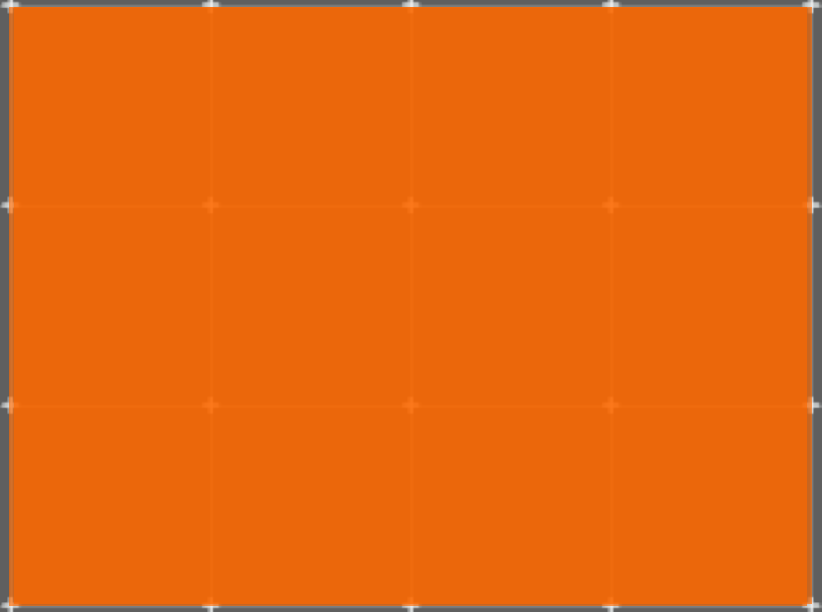
arm

ResetReason API – Example Use

Real life example – cont.

```
const reset_reason_t reason = ResetReason::get();

if (reason == RESET_REASON_WATCHDOG) {
    printf("Watchdog reset\r\n");
    rollback();
}
```

Porting ResetReason



arm

Implementing HAL functions

- Add `RESETREASON` to `device_has` in targets.json

https://github.com/ARMmbed/mbed-os/blob/master/targets/targets.json

```
"LPC1768": {
        "inherits": ["LPCTarget"],
        "core": "Cortex-M3",
        "extra_labels": ["NXP", "LPC176X", "MBED_LPC1768"],
        "supported_toolchains": ["ARM", ..., "GCC_CR", "IAR"],
        "detect_code": ["1010"],
        "device_has": ["ANALOGIN", ..., "RESET_REASON"],
        "release_versions": ["2", "5"],
        "features": ["LWIP"],
        "device_name": "LPC1768",
        "bootloader_supported": true
    },
```


arm

Implementing HAL functions

- Implement the `ResetReason` functions in mbed target:

  https://github.com/ARMmbed/mbed-os/blob/feature-watchdog/hal/reset_reason_api.h

```
reset_reason_t hal_reset_reason_get(void);
uint32_t hal_reset_reason_get_raw(void);
```

`hal_reset_reason_get()`

- Fetch the reset reason for the last system reset.
- Map the platform specific reasons to a corresponding mbedOS reset reason and return it.
- If multiple reset reasons are set, return RESET_REASON_MULTIPLE.

`hal_reset_reason_get_raw()`

- Return the raw platform specific reset reason register value

arm

Implementing HAL functions

```
void hal_reset_reason_clear(void);
```

`hal_reset_reason_clear()`

- Clear the reset reason from registers
- If the platform does not persist reset reasons between reboots this does not require any action.

Running Tests

ResetReason validation suite:

```
$ git checkout feature-watchdog
$ mbed test -t <toolchain> -m <target> -n \
    "tests-mbed_hal-reset_reason"
```

Hands-on workshop

- Watchdog
  - Use branch - `feature-watchdog`
  - Porting - https://github.com/ARMmbed/mbed-sip-workshop-2018q1/blob/master/Watchdog.md

- ResetReason
  - Use branch - `feature-watchdog`
  - Porting - https://github.com/ARMmbed/mbed-sip-workshop-2018q1/blob/master/ResetReason.md

- Workshop materials - https://github.com/ARMmbed/mbed-sip-workshop-2018q1

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
감사합니다
धन्यवाद