ETH zürich ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

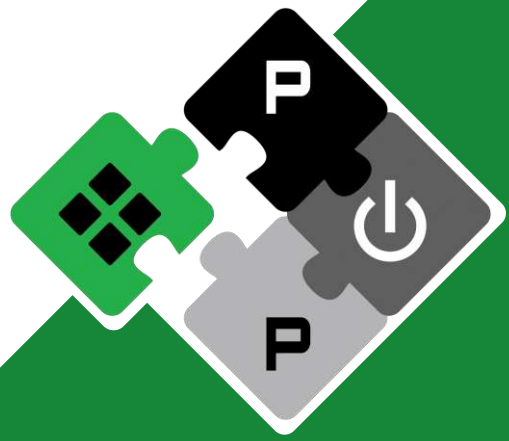
Compilers and OSes for the Future

Dreaming High-level Abstractions for the Low-level PULP Software Stack

Robert Balas balasr@iis.ee.ethz.ch
Giuseppe Tagliavini giuseppe.tagliavini@unibo.it

Contributors:

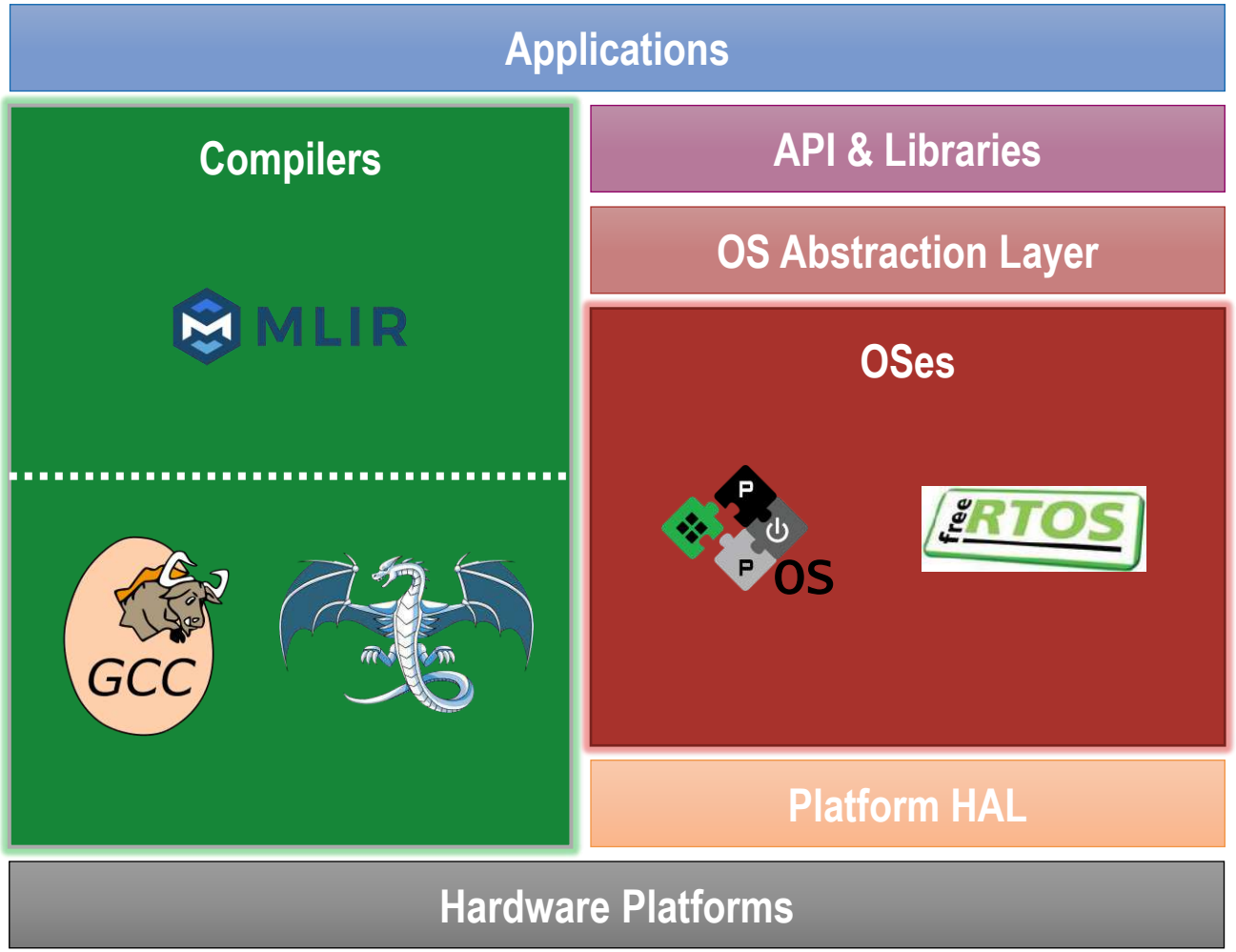
Federico Ficarelli



@pulp_platform
pulp-platform.org
youtube.com/pulp_platform

PULP Platform
Open Source Hardware, the way it should be!

PULP Software Stack: Focus on Compilers and OSes



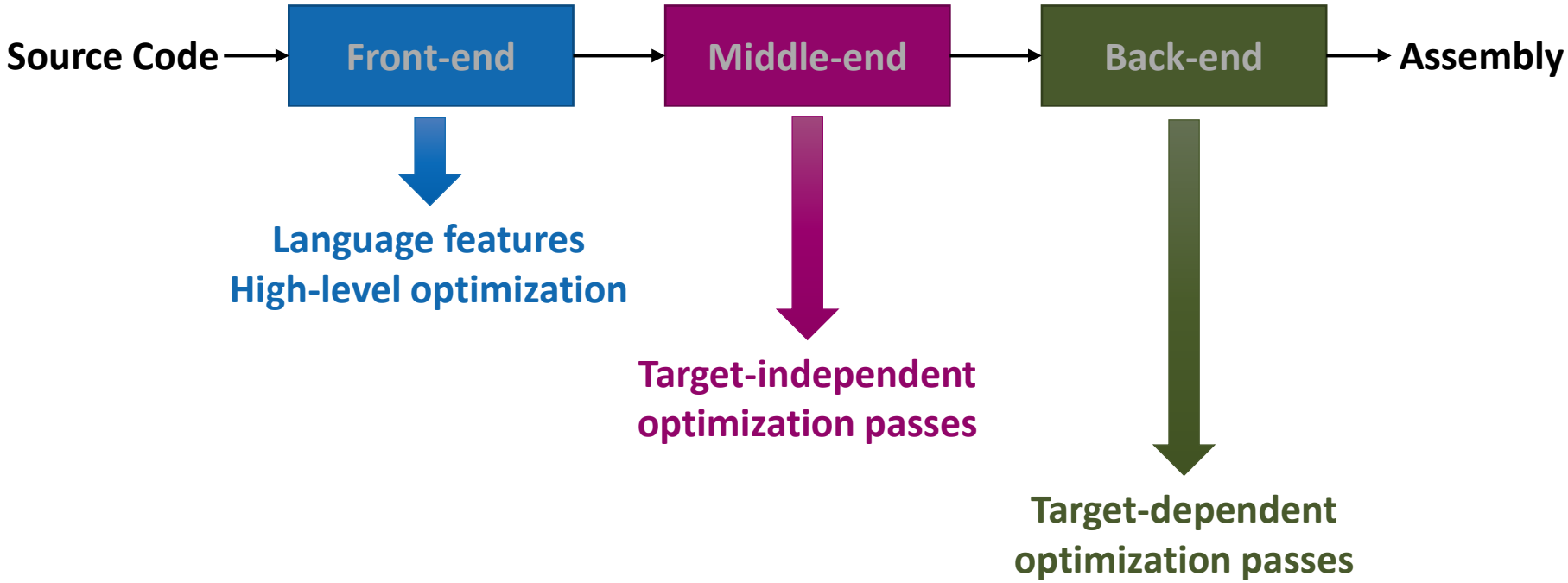
https://github.com/pulp-platform/pulp-sdk

https://github.com/pulp-platform/pulp-freertos

https://github.com/pulp-platform/riscv-gnu-toolchain

https://github.com/pulp-platform/llvm-project

Compiler Optimization: A quick overview



Front-End example: OpenMP lowering (LLVM)

C code

```
#pragma omp parallel for schedule(static, 2)
for (int i = 0; i < N; i++)
    c[i] += fn(a[i], b[i]);
```

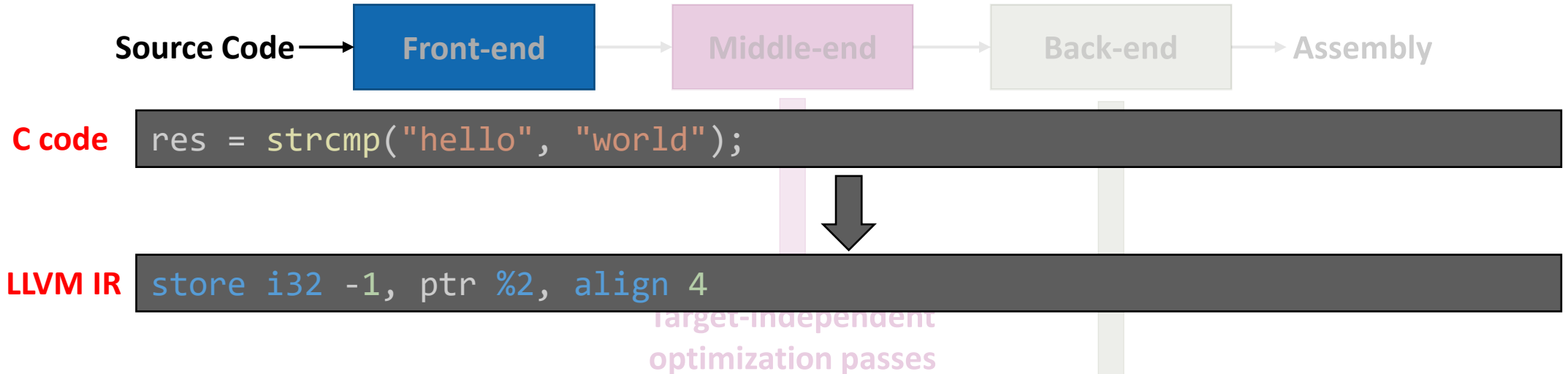
Target-independent

LLVM IR

```
call void @__kmpc_for_static_init_4(ptr nonnull @1, i32 %7, i32 33,
    ptr nonnull %6, ptr nonnull %3, ptr nonnull %4, ptr nonnull %5,
    i32 1, i32 2)
```

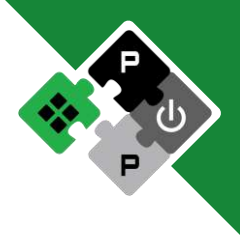
Compilers lower OpenMP directive by means of **function outlining**
If we want to extend this mechanism, we can define **new directives**
→ **front-end passes + runtime library**

Front-End example: Reducing language patterns (LLVM)



We can infer properties on the Abstract Syntax Tree (AST)
Matching AST patterns to find opportunities for code optimization
The progressive lowering approach provided by MLIR is better for complex cases

Middle-End Example: Recognizing builtin patterns (LLVM)



Source Code → Front-end → **Middle-end** → Back-end → Assembly

C code

```
#define N 100
for (int i = 0; i < N; i++)
    c[i] = 0;
```

LLVM IR

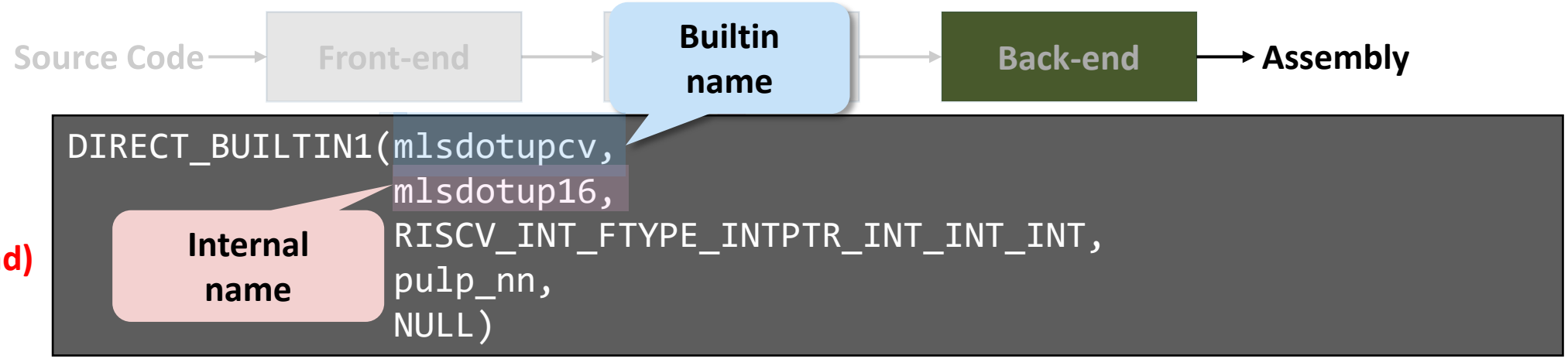
```
for.body:                                         ; preds = %for.cond
  %1 = load i32, ptr %i, align 4
  %arrayidx = getelementptr inbounds [0 x i32], ptr @c, i32 0, i32 %1
  store i32 0, ptr %arrayidx, align 4
  br label %for.inc
```

LoopIdiomRecognizePass

LLVM IR

```
tail call void
@llvm.memset.p0.i32(ptr noundef nonnull align 4 dereferenceable(400) @c,
    i8 0, i32 400, i1 false)
```

Back-End Example: Defining target-specific intrinsics (GCC)



Builtins are associated with an instruction pattern (see next slide)

Other infos: function signature, ISA extension, parameter constraints

Back-End Example: Defining target-specific intrinsics (GCC)

Internal name (pattern)

MD file (backend)

```
(define_insn "mlsdotup<VMODESALLINT:mode>"
[
  (unspec:SI [(post_inc:SI (match_operand:SI 1 "register_operand"
                "+r"))] UNSPEC_MLSDOT)
  (parallel[
    (set (match_operand:SI 0 "register_operand" "=r")
      (plus:SI
        (plus:SI
          (mult:SI
            (zero_extend:SI (vec_select:HI (unspec:V2HI
...
```

RTL template

Middle-end instructions are expanded into RTL functional representations

RTL enables back-end optimizations → more powerful than inline ASM!!!

Back-End Example: Defining target-specific intrinsics (GCC)

MD file (backend)

```
(define_insn "mlsdotup<VMODESALLINT:mode>"
[
 ...
]
"((Pulp_Cpu==PULP_NN) && !TARGET_MASK_NOVECT)"
"pv.mlsdotup.<allint_vec_size>.%4 \t%0,%1,%2\t"
[(set_attr "type" "arith")
 (set_attr "mode" "SI")]
)
```

Conditions

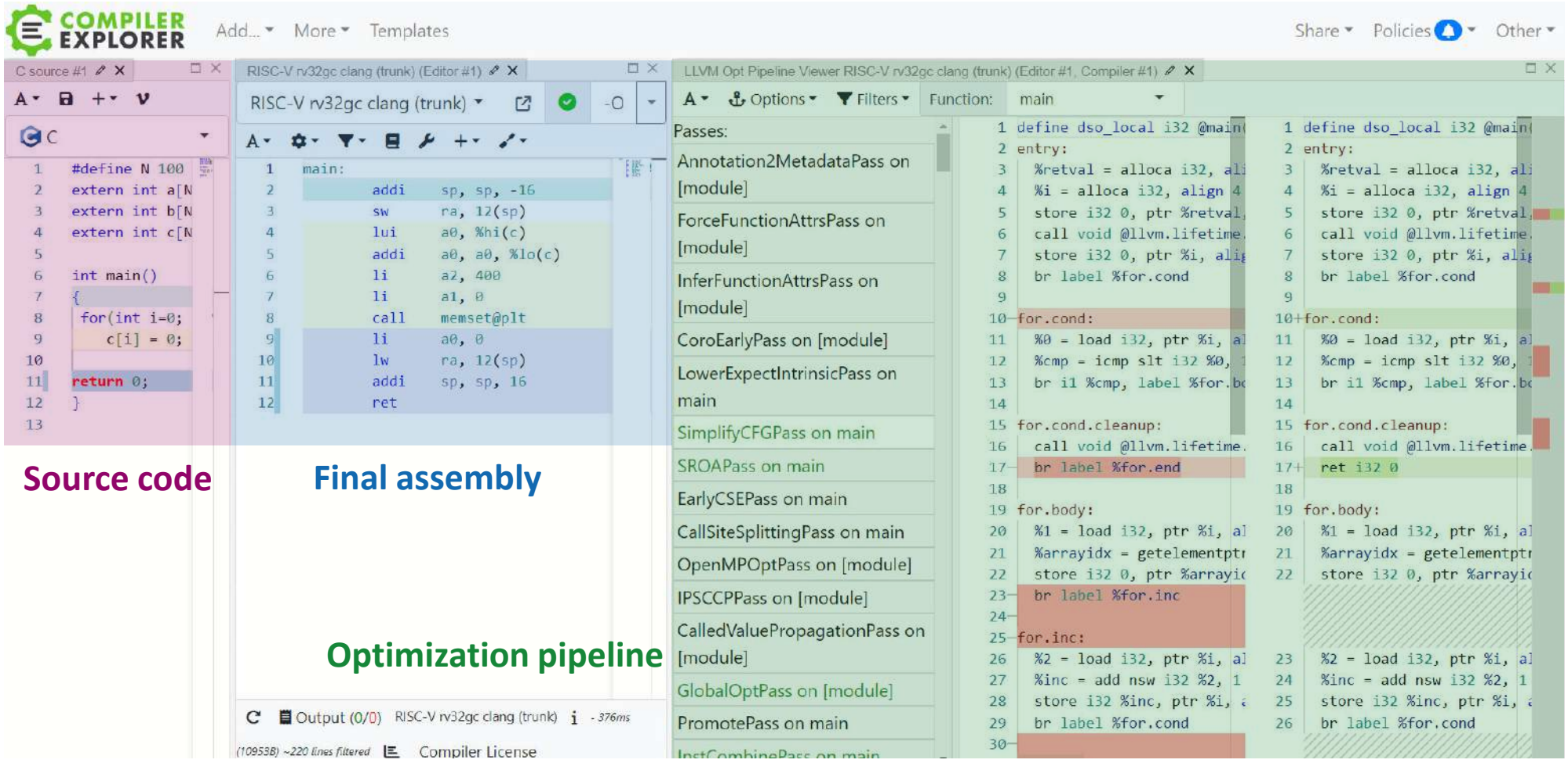
Output template

Assembly generation follows all back-end transformations and optimization passes

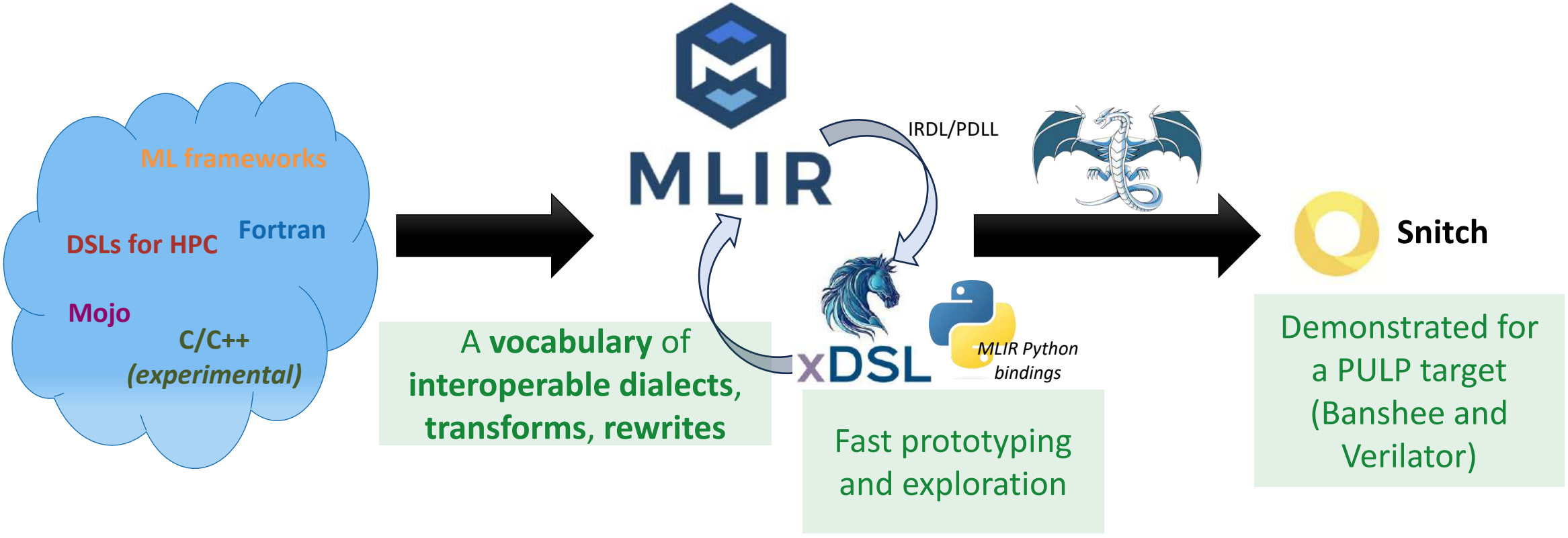
The matching RTL template may be different by the one used for expansion

How to quickly understand the compiler flow?

- We can use Compiler Explorer (https://godbolt.org/) to follow the compiler flow



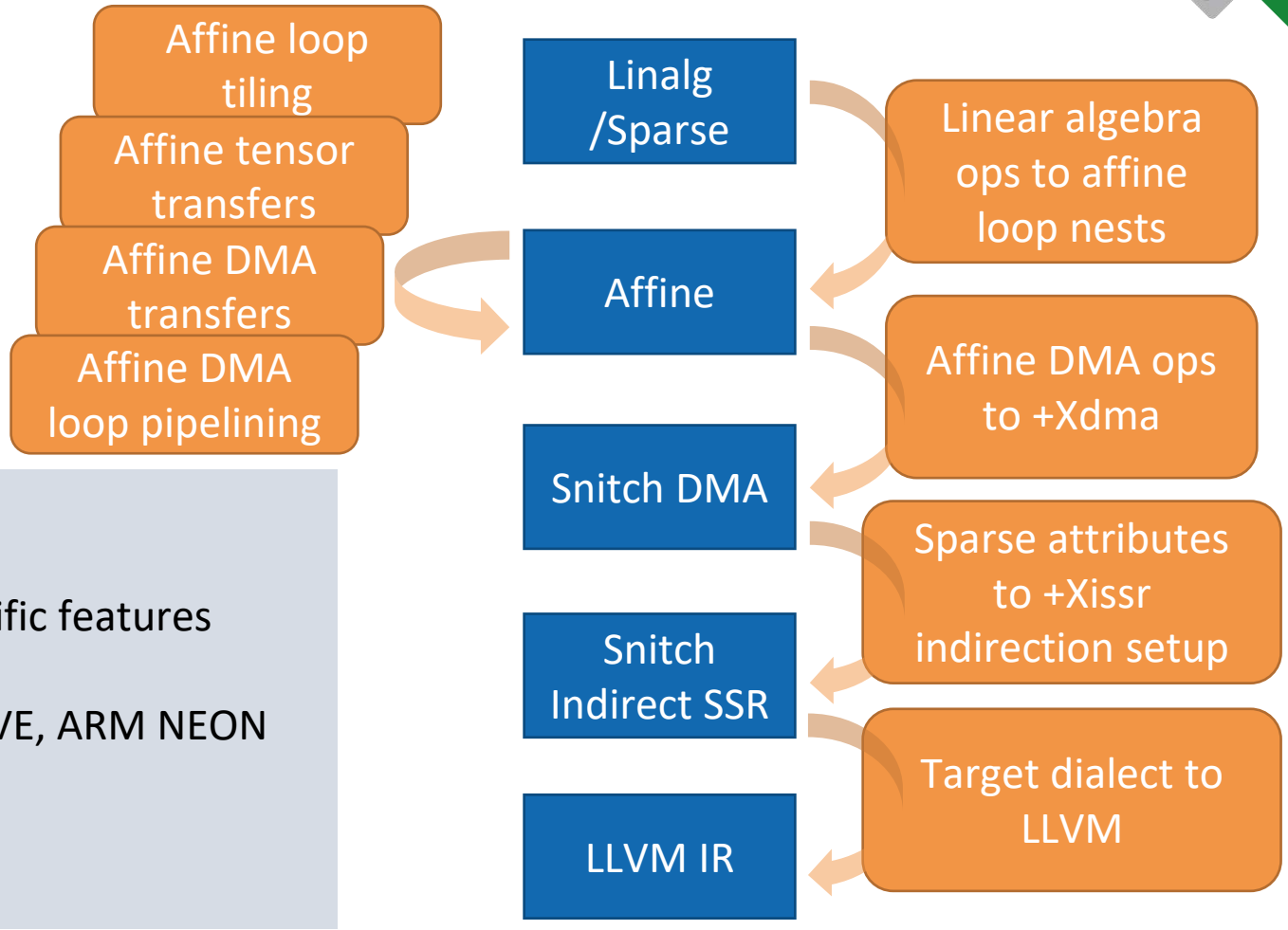
Increasing the abstraction level of compiler toolchains



Example: Progressive lowering for Snitch

Main goal

Leverage **double buffering** (i.e., orchestrating DMA to/from scratchpad memory) and **affine/indirect SSRs** by means of **MLIR lowering passes**



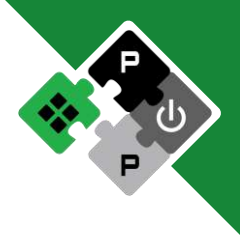
Target dialects

- Lowest level dialects in MLIR
- Map higher level operations to target-specific features
- Last MLIR lowering step towards LLVM IR
- Examples: Intel AMX, Intel AVX512, ARM SVE, ARM NEON
- For Snitch:
  - **MLIR → +Xissr setup instructions**
  - **MLIR → +Xdma operations**

Main challenges

- Find an **efficient lowering** strategy in a **huge search space**
- **Adopt the right dialect at the right abstraction level**

Lowering for Scratchpad Memory 1/4



```
func.func @matmul(%A: memref<256x256xf32>,
                  %B: memref<256x256xf32>,
                  %C: memref<256x256xf32>) {
  linalg.matmul ins(%A, %B:
                    memref<256x256xf32>,
                    memref<256x256xf32>)
                outs(%C:
                    memref<256x256xf32>)
  return
}
```

Starting point: naive 2D matrix multiplication using builtin **linalg dialect**

```
func.func @matmul(%arg0: memref<256x256xf32>,
                  %arg1: memref<256x256xf32>,
                  %arg2: memref<256x256xf32>) {
    affine.for %arg3 = 0 to 256 {
      affine.for %arg4 = 0 to 256 {
        affine.for %arg5 = 0 to 256 {
          %0 = affine.load %arg0[%arg3, %arg5]
          %1 = affine.load %arg1[%arg5, %arg4]
          %2 = affine.load %arg2[%arg3, %arg4]
          %3 = arith.mulf %0, %1 : f32
          %4 = arith.addf %2, %3 : f32
          affine.store %4, %arg2[%arg3, %arg4]
        }
      }
    }
    return
  }
```

Linalg to Affine: lower to **affine** operations and **arithmetic operations** (textbook N^3)

Lowering for Scratchpad Memory 2/4

```
#map = affine_map<(d0, d1) -> (d0)>

func.func @matmul(%arg0: memref<256x256xf32>, %arg1:
memref<256x256xf32>,
                  %arg2: memref<256x256xf32>) {
  [...]
  %alloc = memref.alloc() : memref<1x256xf32, 1>
  %alloc_8 = memref.alloc() : memref<1xi32>
  [...]
  affine.for %arg3 = 0 to 256 {
    affine.dma_start %arg0[%arg3, 0], %alloc[%c0_6, %c0_6],
                     %alloc_8[%c0_6], %c256_5
    affine.dma_wait %alloc_8[%c0_6], %c256_5
    affine.for %arg4 = 0 to 256 {
      affine.dma_start %arg1[0, %arg4], %alloc_9[%c0_4, %c0_4],
                       %alloc_10[%c0_4], %c256, %c256_2, %c1_3
      affine.dma_wait %alloc_10[%c0_4], %c256
      affine.for %arg5 = 0 to 256 {
        [...]
```

Scratchpad buffers

DMA async tags

Affine data copy: generate explicit copying for affine memory operations into **local, statically sized buffers**

Introduce **sync copies** using **async DMA ops** to/from scratchpad buffers

Lowering for Scratchpad Memory 3/4

```
#map = affine_map<(d0) -> (d0 - 1)>
func.func @matmul(%arg0: memref<256x256xf32>, %arg1: memref<256x256xf32>, %arg2: memref<256x256xf32>) {
  // [...]
  %alloc_13 = memref.alloc() : memref<2x1x256xf32, 1>
  %alloc_14 = memref.alloc() : memref<2x1xi32>
  affine.dma_start %arg0[%c0, 0], %alloc_13[%c0 mod 2, 0, 0], %alloc_14[%c0 mod 2, 0], %c256_9
  affine.for %arg3 = 1 to 256 {
    affine.dma_start %arg0[%arg3, 0], %alloc_13[%arg3 mod 2, 0, 0], %alloc_14[%arg3 mod 2, 0], %c256_9
    // [...]
    %10 = affine.apply #map(%arg3)
    affine.dma_wait %alloc_14[%10 mod 2, 0], %c256_9
    // [...]
    affine.dma_start %arg1[0, %c0_1], %alloc_17[%c0_1 mod 2, 0, 0], %alloc_18[%c0_1 mod 2, 0], %c256_5, %c256_6, %c1_7
    affine.for %arg4 = 1 to 256 {
      affine.dma_start %arg1[0, %arg4], %alloc_17[%arg4 mod 2, 0, 0], %alloc_18[%arg4 mod 2, 0], %c256_5, %c256_6, %c1_7
      // [...]
      affine.dma_wait %alloc_18[%20 mod 2, 0], %c256_5
      // [...]
      affine.for %arg5 = 0 to 256 {
        %26 = affine.load %alloc_13[%10 mod 2, 0, %arg5]
        %27 = affine.load %alloc_17[%20 mod 2, %arg5, 0]
        %28 = affine.load %arg2[%10, %20]
        %29 = arith.mulf %26, %27 : f32
        %30 = arith.addf %28, %29 : f32
        affine.store %30, %arg2[%10, %20]
      }
    }
  // [...]
```

Additional dimension on async tags

Affine pipeline data transfers: overlap non-blocking DMA operations in a loop with computations through **double buffering**

Lowering for Scratchpad Memory 4/4

```
#map = affine_map<(d0) -> (d0 - 1)>
func.func @matmul(%arg0: memref<256x256xf32>, %arg1: memref<256x256xf32>, %arg2: memref<256x256xf32>) {
  // [...]
  %alloc_13 = memref.alloc() : memref<2x1x256xf32, 1>
  %alloc_14 = memref.alloc() : memref<2x1xi32>
  snitch.dma_start %arg0[%c0, 0], %alloc_13[%c0 mod 2, 0, 0], %alloc_14[%c0 mod 2, 0], %c256_9
  affine.for %arg3 = 1 to 256 {
    snitch.dma_start %arg0[%arg3, 0], %alloc_13[%arg3 mod 2, 0, 0], %alloc_14[%arg3 mod 2, 0], %c256_9
    // [...]
    %10 = affine.apply #map(%arg3)
    snitch.dma_wait %alloc_14[%10 mod 2, 0], %c256_9
    // [...]
    snitch.dma_start %arg1[0, %c0_1], %alloc_17[%c0_1 mod 2, 0, 0], %alloc_18[%c0_1 mod 2, 0], %c256_5, %c256_6, %c1_7
    affine.for %arg4 = 1 to 256 {
      snitch.dma_start %arg1[0, %arg4], %alloc_17[%arg4 mod 2, 0, 0], %alloc_18[%arg4 mod 2, 0], %c256_5, %c256_6, %c1_7
      // [...]
      snitch.dma_wait %alloc_18[%20
      // [...]
      affine.for %arg5 = 0 to 256 {
        %26 = affine.load %alloc_13[%10 mod 2, 0, %arg5]
        %27 = affine.load %alloc_17[%20 mod 2, %arg5, 0]
        %28 = affine.load %arg2[%10, %20]
        %29 = arith.mulf %26, %27 : f32
        %30 = arith.addf %28, %29 : f32
        affine.store %30, %arg2[%10, %20]
      }
    }
   // [...]
```

Lowered to runtime calls/inline dm* setup

Lowered to runtime call/inline dmstati busy wait

Partial lowering to a custom dialect that can be lowered directly to `llvm` dialect

Lowering for SSRs

```
func.func @memcpy(%A: memref<256xi32>, %B:
memref<256xi32>) {
  %lb = index.constant 0
  %ub = = index.constant 256
  %step = = index.constant 1
  scf.for %iv = %lb to %ub step %step ({
    %v = memref.load %A[%iv] : memref<256xi32>
    memref.store %v, %B[iv] : memref<256xi32>
  }
}
```

Starting point: structured/affine loop nests on memory buffers

```
func.func @memcpy(%A: memref<256xi32>, %B: memref<256xi32>) {
  %lb = index.constant 0
  %ub = index.constant 256
  %step = index.constant 1
  %0 = index.constant 0
  %1 = index.constant 0
  snitch.ssr_set_dimension_bound(%0, %ub) {dimension = 0 : i32}
  snitch.ssr_set_dimension_stride(%0, %step) {dimension = 0 : i32}
  %isrc = memref.extract_aligned_pointer_as_index(%A)
  %src = arith.index_cast %isrc : index to i32
  snitch.ssr_set_dimension_source(%0, %src) {dimension = 0 : i32}
  %idst = memref.extract_aligned_pointer_as_index(%B)
  snitch.ssr_set_dimension_bound(%1, %ub) {dimension = 0 : i32}
  snitch.ssr_set_dimension_stride(%1, %step) {dimension = 0 : i32}
  %dst = arith.index_cast %idst : index to i32
  snitch.ssr_set_dimension_destination(%1, %dst) {dimension = 0 : i32}
  snitch.ssr_enable()
  scf.for %iv = %lb to %ub step %step ({
    %v = memref.load %A[%iv] : memref<256xi32>
    memref.store %v, %B[iv] : memref<256xi32>
  }
  snitch.ssr_disable()
}
```

Loops to streams: lower to SSR setup/teardown

Operating System Landscape

Bare-metal

pulp-runtime

pulp-freertos

Real-Time OS

FreeRTOS

RTIC

pulpos

General-Purpose OS

Linux

FreeBSD

Operating System Landscape

Bare-metal

pulp-runtime

pulp-freertos

Real-Time OS

FreeRTOS

RTIC

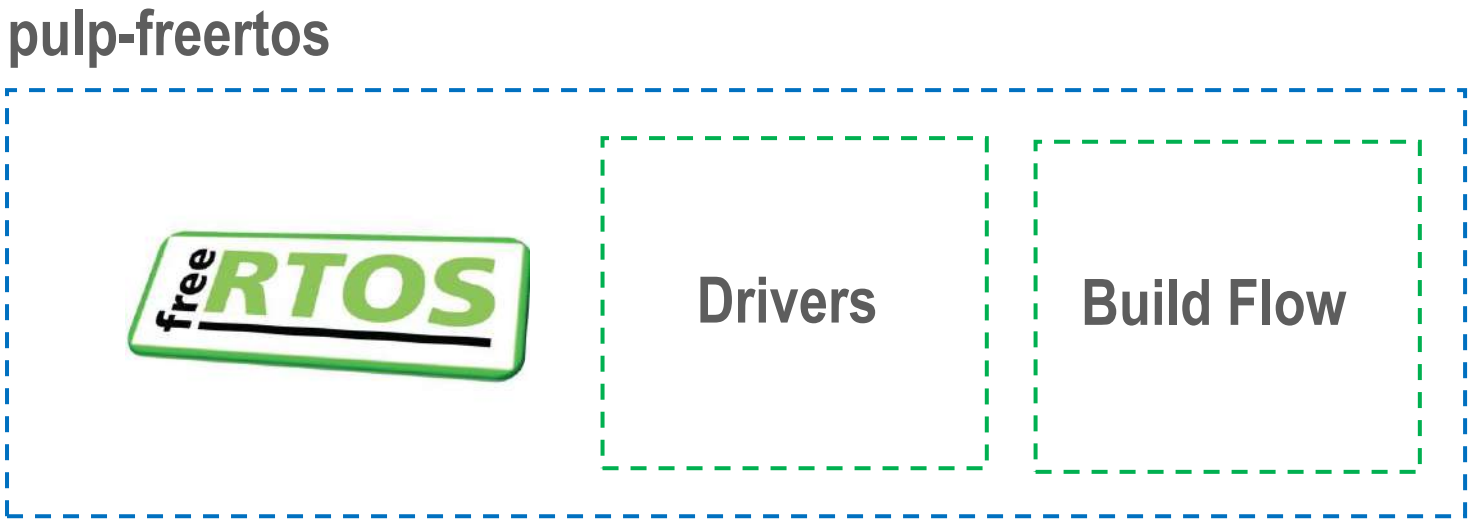
pulpos

General-Purpose OS

Linux

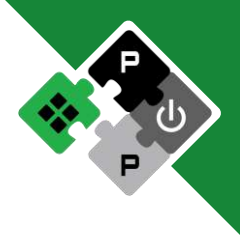
FreeBSD

PULP FreeRTOS Distribution



FreeRTOS itself is just a scheduling kernel!

FreeRTOS Primitives



- **Scheduling**
  - Pre-emptive and Cooperative
  - Systick
  - SMP and AMP
- **Message Passing**
  - Queues
  - Message Buffer
- **Synchronization**
  - Mutex and Semaphores
  - Task notifications
- **Asynchronous Events**
  - RTOS and non-RTOS interrupts

FreeRTOS Example

- **Scheduling**
  - Pre-emptive and Cooperative
  - Systick
  - SMP and AMP
- **Message Passing**
  - Queues
  - Message Buffer
- **Synchronization**
  - Mutex and Semaphores
  - Task notifications
- **Asynchronous Events**
  - RTOS and non-RTOS interrupts

```
static void hello_task(void *params) {
  printf(«hello world!»);
}

int main(void) {
  prvSetupHardware();
  xTaskCreate(hello_task, ..., PRIO, ...);
  vTaskStartScheduler();
  for(;;);
}
```

FreeRTOS Example – Periodic

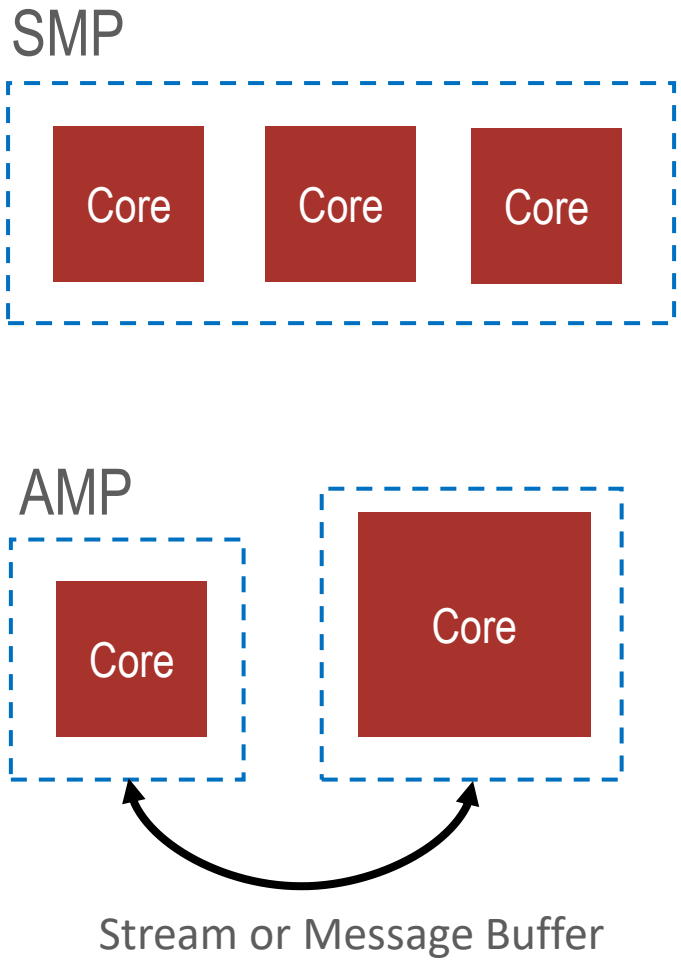
- **Scheduling**
  - Pre-emptive and Cooperative
  - Systick
  - SMP and AMP
- **Message Passing**
  - Queues
  - Message Buffer
- **Synchronization**
  - Mutex and Semaphores
  - Task notifications
- **Asynchronous Events**
  - RTOS and non-RTOS interrupts


```
static void hello_task(void *params) {
  TickType_t last_wake_time;
  last_wake_time = xTaskGetTickCount();
  for (;;) {
    vTaskDelayUntil(&last_wake_time, 10);
    printf(«hello world!»);
  }
}

int main(void) {
  prvSetupHardware();
  xTaskCreate(hello_task, ..., PRIO, ...);
  vTaskStartScheduler();
  for(;;);
}
```

FreeRTOS

- **Scheduling**
  - Pre-emptive and Cooperative
  - Systick
  - SMP and AMP
- **Message Passing**
  - Queues
  - Message Buffer
- **Synchronization**
  - Mutex and Semaphores
  - Task notifications
- **Asynchronous Events**
  - RTOS and non-RTOS interrupts



FreeRTOS Example - Queues

- Scheduling
  - Pre-emptive and Cooperative
  - Systick
  - SMP and AMP
- **Message Passing**
  - Queues
  - Message Buffer
- Synchronization
  - Mutex and Semaphores
  - Task notifications
- Asynchronous Events
  - RTOS and non-RTOS interrupts

```
static void snd_task(void *params) {
  TickType_t last_wake_time;
  last_wake_time = xTaskGetTickCount();
  for (;;) {
    vTaskDelayUntil(&last_wake_time, 10);
    xQueueSend(queue, 123, ...);
  }
}
static void rcv_task(void *params) {
  int value;
  for (;;) {
    xQueueReceive(queue, &value, ...);
  }
}
int main(void) {
  [...]
  queue = xQueueCreate(len, sizeof(int));
  [...]
}
```

FreeRTOS Example - Semaphores

- Scheduling
  - Pre-emptive and Cooperative
  - Systick
  - SMP and AMP
- **Message Passing**
  - Queues
  - Message Buffer
- **Synchronization**
  - Mutex and Semaphores
  - Task notifications
- Asynchronous Events
  - RTOS and non-RTOS interrupts


```
static void snd_task(void *params) {
  TickType_t last_wake_time;
  last_wake_time = xTaskGetTickCount();
  for (;;) {
    vTaskDelayUntil(&last_wake_time, 10);
    xQueueSend(queue, 123, ...);
  }
}
static void rcv_task(void *params) {
  int value;
  for (;;) {
    xQueueReceive(queue, &value, ...);
  }
}
int main(void) {
  [...]
  queue = xQueueCreate(len, sizeof(int));
  [...]
}
```

FreeRTOS Example - Interrupts

- **Scheduling**
  - Pre-emptive and Cooperative
  - Systick
  - SMP and AMP
- **Message Passing**
  - Queues
  - Message Buffer
- **Synchronization**
  - Mutex and Semaphores
  - Task notifications
- **Asynchronous Events**
  - RTOS and non-RTOS interrupts

```
void some_handler(void) {
  [...]
}

int main(void) {
  irq_set_lvl_and_prio(IRQ_ID, 4, 0);
  irq_set_handler(IRQ_ID, some_handler);
  [...]
  for (;;);
}
```

regular C functio

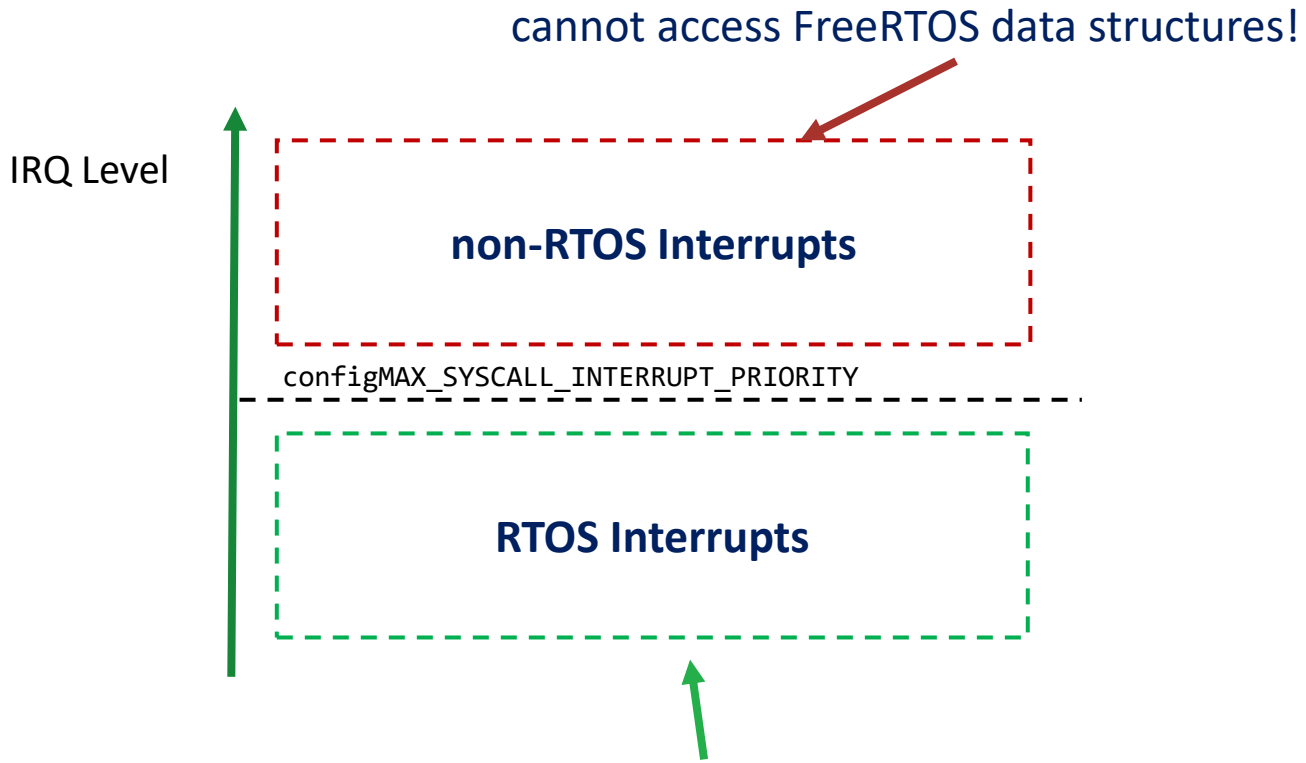
priority

level

interrupt ID of physical pin to CLIC

FreeRTOS and CLIC

- **Scheduling**
  - Pre-emptive and Cooperative
  - Systick
  - SMP and AMP
- **Message Passing**
  - Queues
  - Message Buffer
- **Synchronization**
  - Mutex and Semaphores
  - Task notifications
- **Asynchronous Events**
  - RTOS and non-RTOS interrupts



for interrupt controllers that support level based pre-emption

I want to tweak various things

- **Look into `FreeRTOSConfig.h`**

```
// features
#define configUSE_MUTEXES 1
#define configUSE_RECURSIVE_MUTEXES 1
#define configUSE_TIMERS 1
#define configUSE_NEWLIB_REENTRANT 1
[...]
// settings
#define configTOTAL_HEAP_SIZE (16 * 1024)
#define configTIMER_TASK_PRIORITY 1
#define configMAX_SYSCALL_INTERRUPT_...
[...]
// hooks
#define configUSE_MALLOC_FAILED_HOOK 0
#define configUSE_TICK_HOOK 0
#define configUSE_IDLE_HOOK 0
```

FreeRTOS

- **Flexible lightweight open-source RTOS**
- **Simple and powerful API**
- **Works on PULPissimo, PULP, ControlPULP, CVA6...**

github.com/pulp-platform/pulp-freertos

PULP Platform

Open Source Hardware, the way it should be!

Robert Balas balasr@iis.ee.ethz.ch
Giuseppe Tagliavini giuseppe.tagliavini@unibo.it

Institut für Integrierte Systeme – ETH Zürich
Gloriastrasse 35
Zürich, Switzerland

DEI – Universitá di Bologna
Viale del Risorgimento 2
Bologna, Italy

ETH zürich ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

@pulp_platform

pulp-platform.org

youtube.com/pulp_platform