Update on Ara

29/09/2021

Matteo Perotti
Matheus Cavalcante
Nils Wistoff

Professor Luca Benini
Integrated Systems Laboratory
ETH Zürich

Summary

- Prepare a **comparison** with **AraV1**
  - Kernels TP
  - PPA + Efficiency
- SW optimization
  - AXPY
  - CONV2D (3 channels)
  - Bug fixes
- HW optimization
  - Timing
  - Efficiency

SW optimization

AXPY, with intrinsics:

```
loop:
  vsetvli N, e64, m8, ta, mu
  vload v8, (a0)
  vload v16, (a1)
  vsetvli e64, m8, tu, mu
  vfmacc.vf v16, fa0, v8
  vsetvli e64, m8, ta, mu
  vstore v16, (a1)
```

SW optimization

AXPY, with intrinsics:

```
loop:
  vsetvli N, e64, m8, ta, mu
  vload v8, (a0)
  vload v16, (a1)
→ vsetvli e64, m8, tu, mu
  vfmacc.vf v16, fa0, v8
→ vsetvli e64, m8, ta, mu
  vstore v16, (a1)
```

Unwanted instructions
Performance KILLERS!

SW optimization

AXPY, with intrinsics:

```
loop:
  vsetvli N, e64, m8, ta, mu
  vload v8, (a0)
  vload v16, (a1)
→ vsetvli e64, m8, tu, mu
  vfmacc.vf v16, fa0, v8
→ vsetvli e64, m8, ta, mu
  vstore v16, (a1)
```

Unwanted instructions
Performance KILLERS!

AXPY, with assembly:

```
loop:
  vsetvli N, e64, m8, ta, mu
  vload v8, (a0)
  vload v16, (a1)
  vfmacc.vf v16, fa0, v8
  vstore v16, (a1)
```

SW optimization - Memory System

- **DAXPY** Throughput (TP) - Memory bound kernel

TP on TP_MAX	AraV1	AraV2
2 lanes	96.00%	97.34%
4 lanes	93.80%	94.80%
8 lanes	88.00%	90.14%
16 lanes	80.00%	82.05%

Comparable throughput results

SW optimization

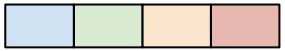
- **Conv2d 7x7, 3 channels** Throughput (TP) - Compute bound kernel

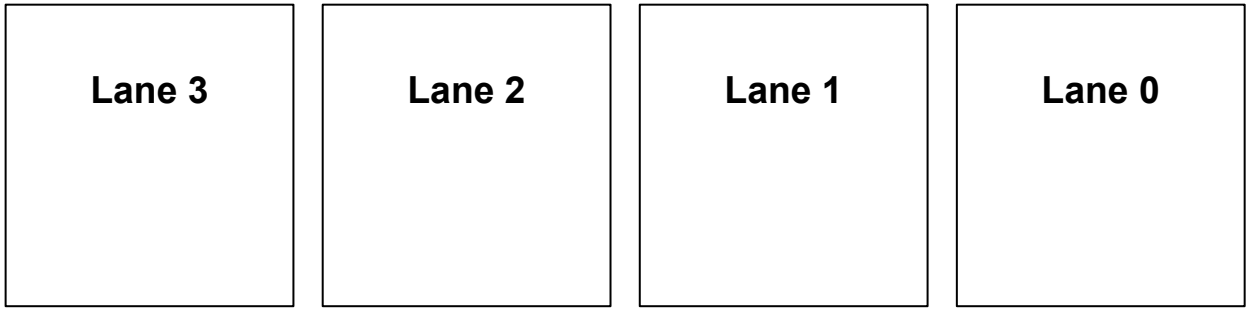
TP on TP_MAX	AraV1	AraV2
2 lanes	93.30%	96.42%
4 lanes	92.20%	95.21%
8 lanes	90.60%	91.59%
16 lanes	83.00%	82.15%

Comparable throughput results

BUG fixes - VRF reshuffle

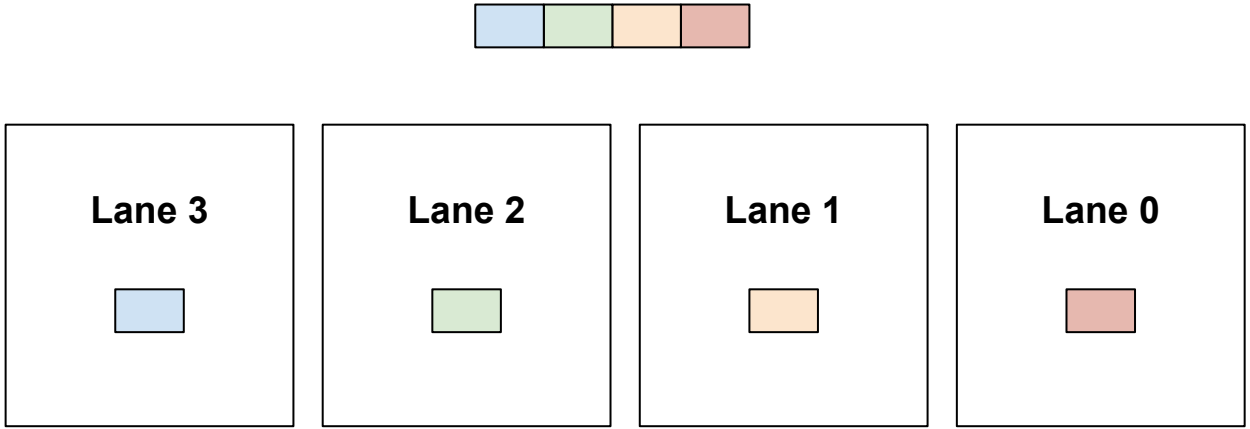
- Ara's VRF is split among the lanes
- Motivation: simple physical routing to banks (O(N) against $O(N^2)$)
- Exploit parallelism: subsequent elements go to subsequent lanes





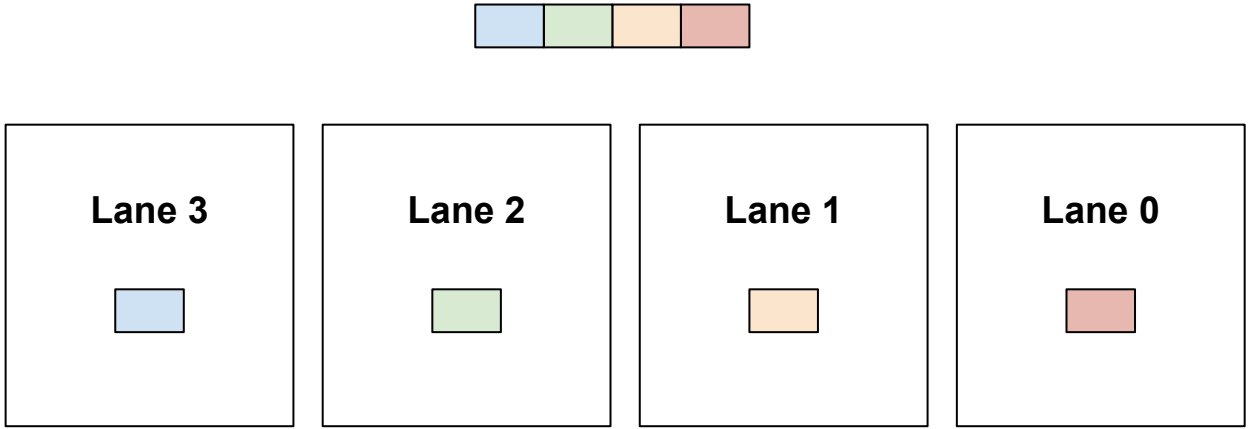
BUG fixes - VRF reshuffle

- Ara's VRF is split among the lanes
- Motivation: simple physical routing to banks (O(N) against $O(N^2)$)
- Exploit parallelism: subsequent elements go to subsequent lanes



BUG fixes - VRF reshuffle

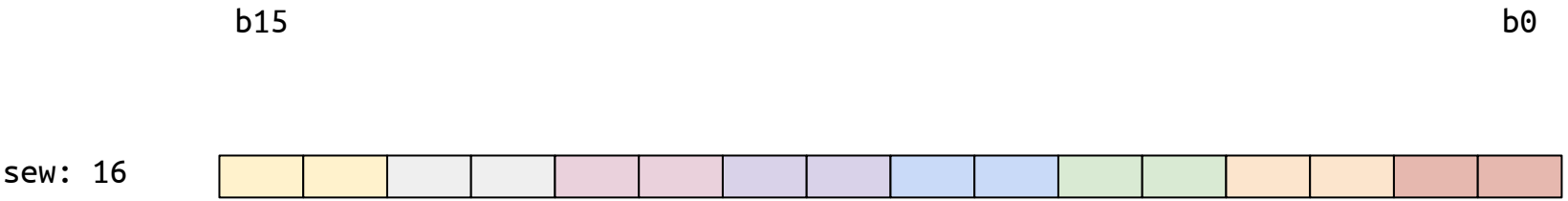
- Ara's VRF is split among the lanes
- Motivation: simple physical routing to banks (O(N) against $O(N^2)$)
- Exploit parallelism: subsequent elements go to subsequent lanes



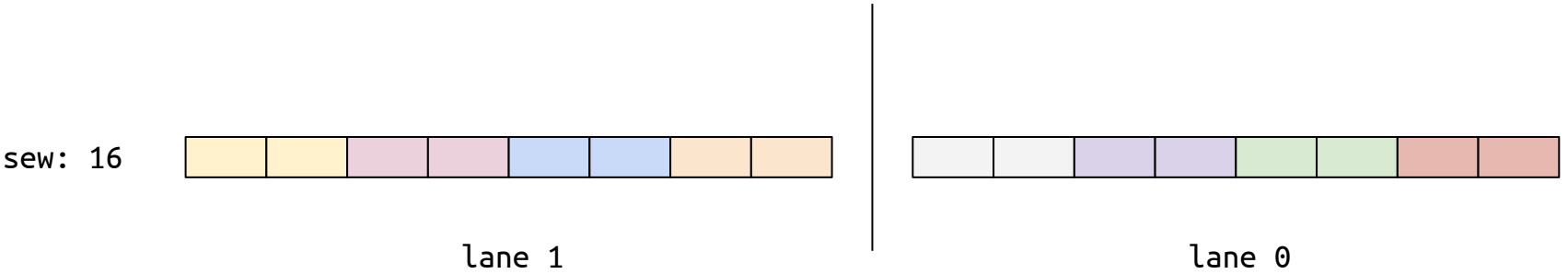
BUT: element width (SEW) can be different!

BUG fixes - VRF reshuffle

- Byte layout vector in memory

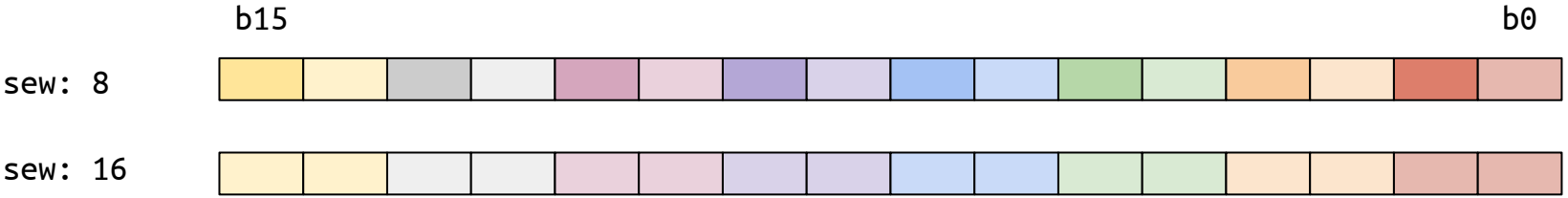


- Byte layout vector in VRF (2 lanes)

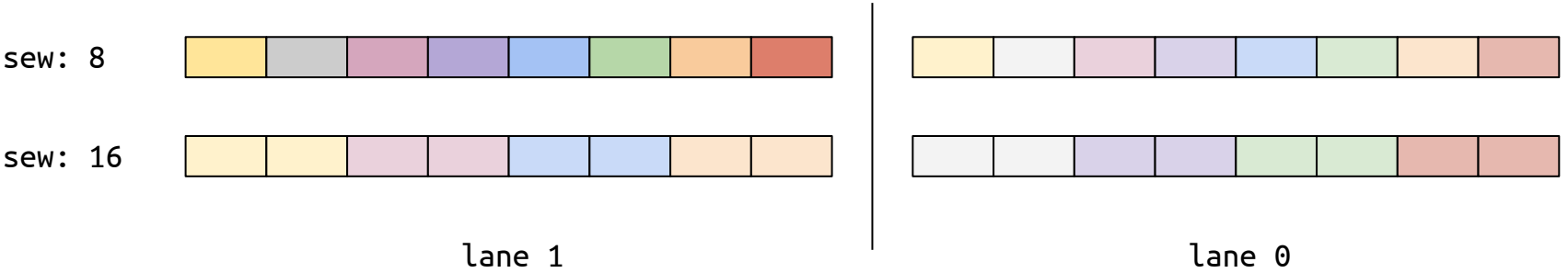


BUG fixes - VRF reshuffle

- Byte layout vector in memory



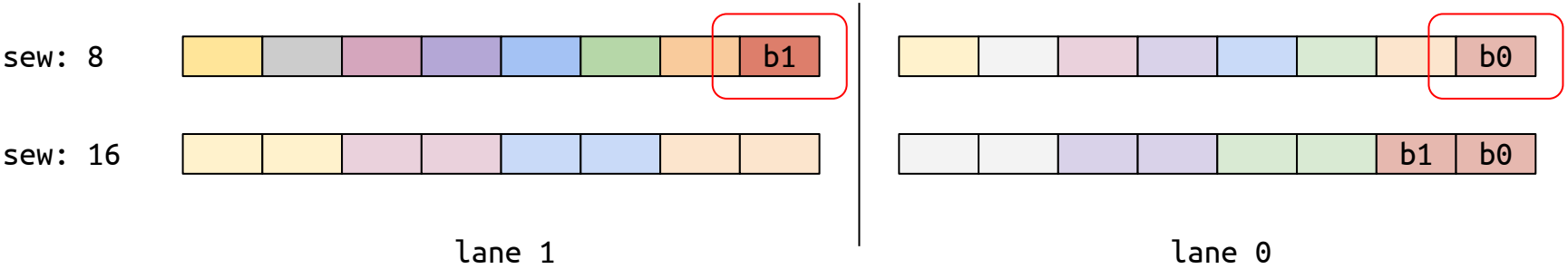
- Byte layout vector in VRF (2 lanes)



BUG fixes - VRF reshuffle

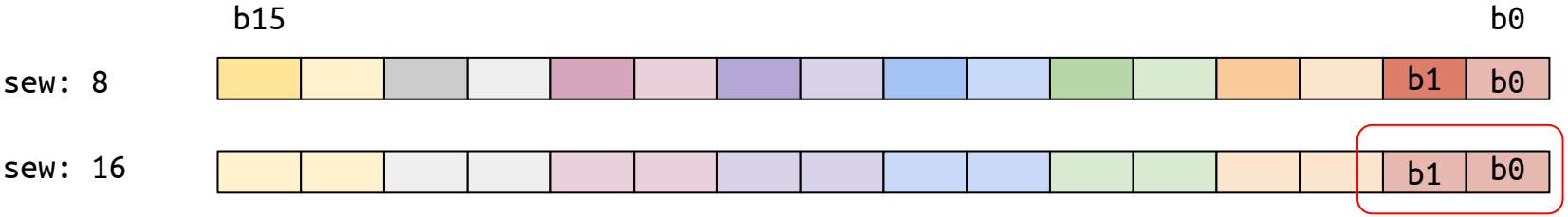
- Byte layout vector in memory

- Byte layout vector in VRF (2 lanes)

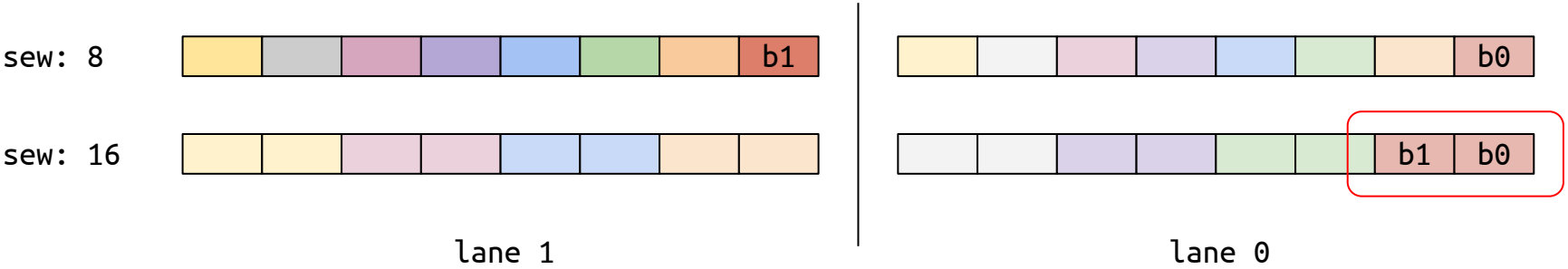


BUG fixes - VRF reshuffle

- Byte layout vector in memory

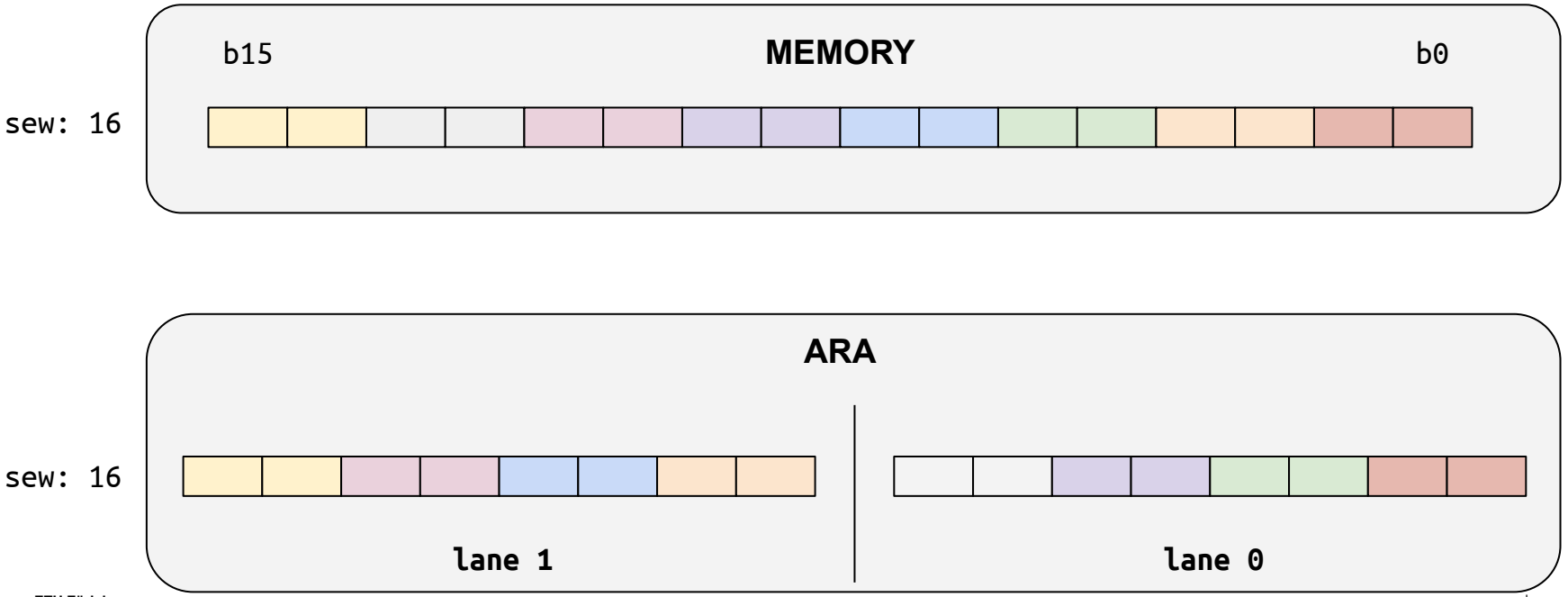


- Byte layout vector in VRF (2 lanes)



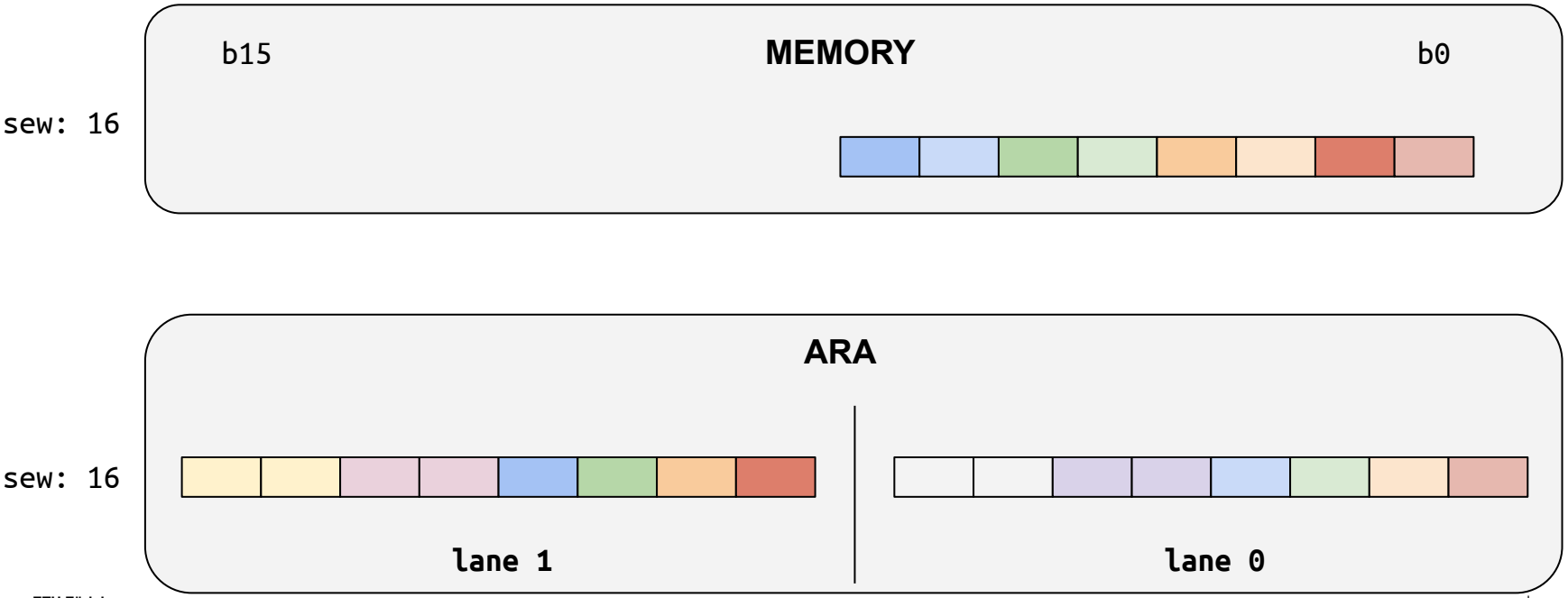
BUG fixes - VRF reshuffle

- Load 1: 8 elements, VSEW = 2 Byte



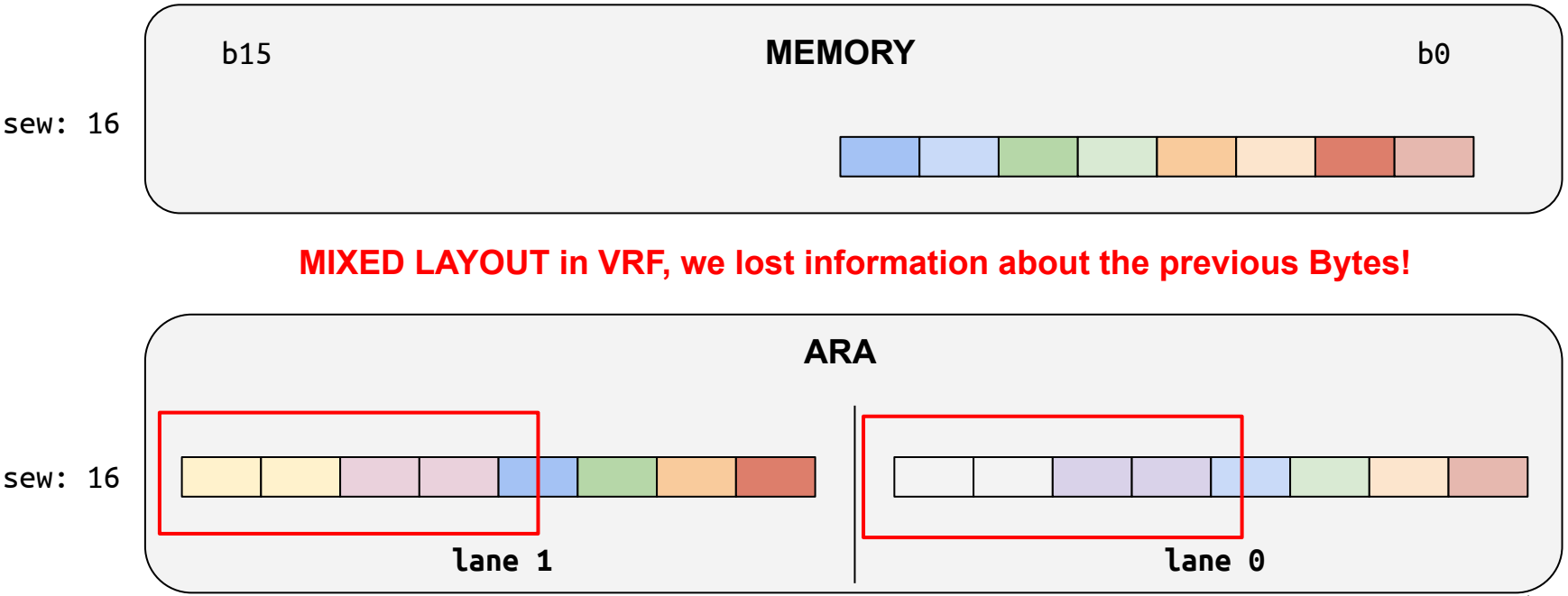
BUG fixes - VRF reshuffle

- Load 2: 8 elements, VSEW = 1 Byte



BUG fixes - VRF reshuffle

- Load 2: 8 elements, VSEW = 1 Byte



BUG fixes - VRF reshuffle

- To be compliant to RVV 1.0 we cannot lose that information!

- Solution: a vector slide operation (stride == 0)

Re-shuffle the VRF Bytes when VSEW changes in a vector register

BUG fixes - VRF reshuffle

- To be compliant to RVV 1.0 we cannot lose that information!

- Solution: a vector slide operation (stride == 0)

Re-shuffle the VRF Bytes when VSEW changes in a vector register

This is a costly operation:

- Avoid reshuffling when the register is written for the 1st time
- Avoid reshuffling when the whole register is written
- Keep different registers for different VSEW

New feature

- Indexed memory operations (scatter/gather)

- WIP:
  - Testing
  - Benchmark
  - Frequency optimization

- Benchmark sparse workloads
  - Hard to convert to bare metal:
    https://github.com/RALC88/riscv-vectorized-benchmark-suite/tree/rvv-1.0/_canneal/src
  - We will use smaller kernels

HW optimization

- **The lane reaches 1 GHz**
- **Frequency optimization for the whole System**

  - Reproduce environment of AraV1 paper
  - Cut Ara-CVA6 memory consistency signal paths
  - Disable CVA6 OS-related TLB checks and PMPs
  - Cut path Addrgen - Sequencer
  - Simplify Mask Unit handshake

Worst corner frequency:
670 MHz

Worst corner frequency:
850 MHz

HW optimization

- **The lane reaches 1 GHz**
- **Frequency optimization for the whole System**

  - Reproduce environment of AraV1 paper
  - Cut Ara-CVA6 memory consistency signal paths
  - Disable CVA6 OS-related TLB checks and PMPs
  - Cut path Addrgen - Sequencer
  - Simplify Mask Unit handshake

Worst corner frequency:
670 MHz

Worst corner frequency:
850 MHz

WIP: merging the last bug fixes and features

Further

- Merge bug fixes and features
- Starting point for optimization
- Optimize frequency and efficiency
- Freeze RTL