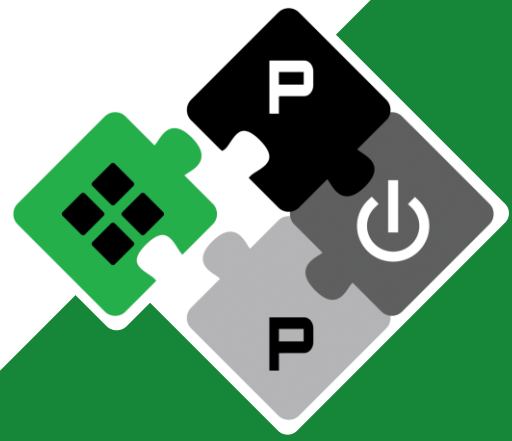
ETH zürich ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

CachePool: Many-core cluster of customizable, lightweight scalar-vector PEs for irregular L2 data-plane workloads

Integrated Systems Laboratory (ETH Zürich)

Zexin Fu, Diyou Shen zexifu, dishen@iis.ee.ethz.ch

Alessandro Vanelli-Coralli avanelli@iis.ee.ethz.ch
Luca Benini lbenini@iis.ee.ethz.ch



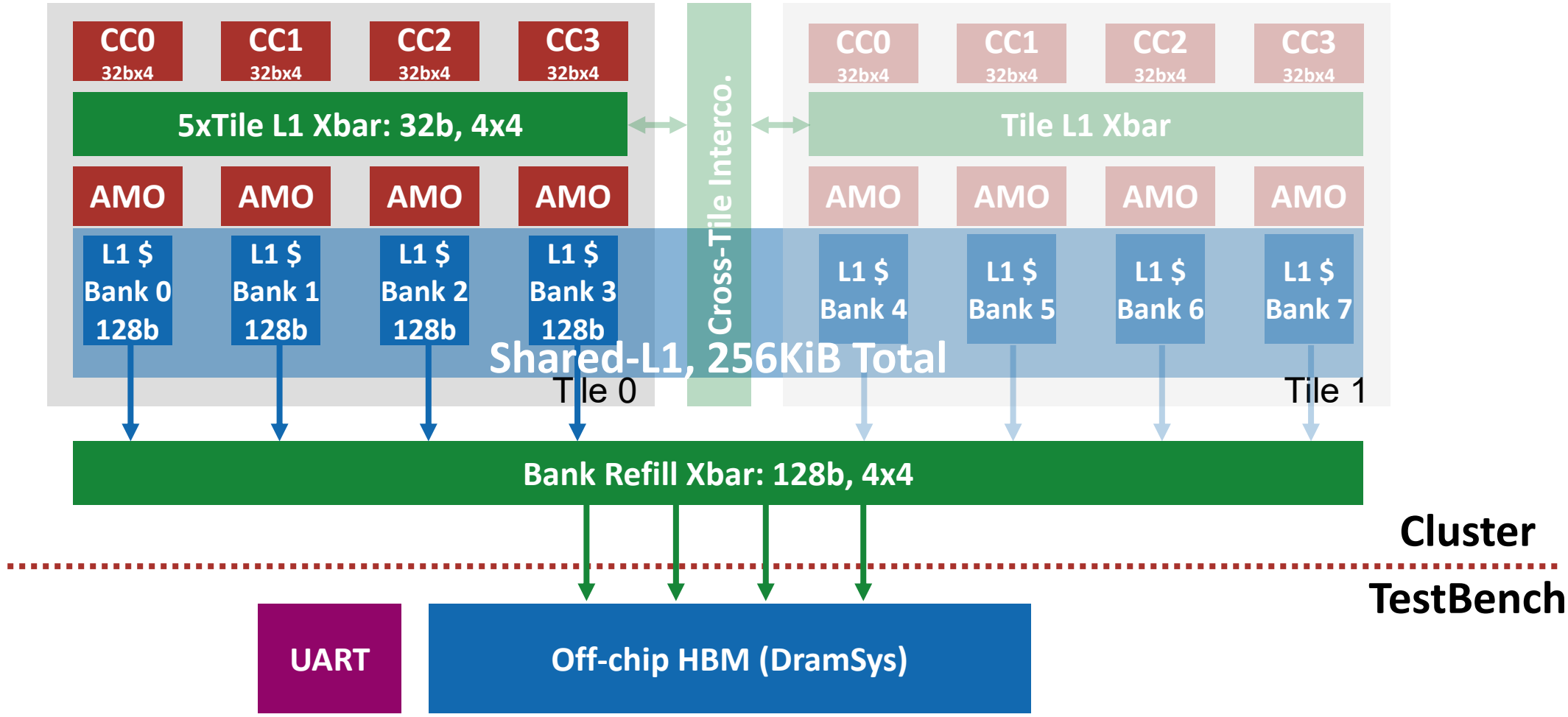
@pulp_platform

pulp-platform.org

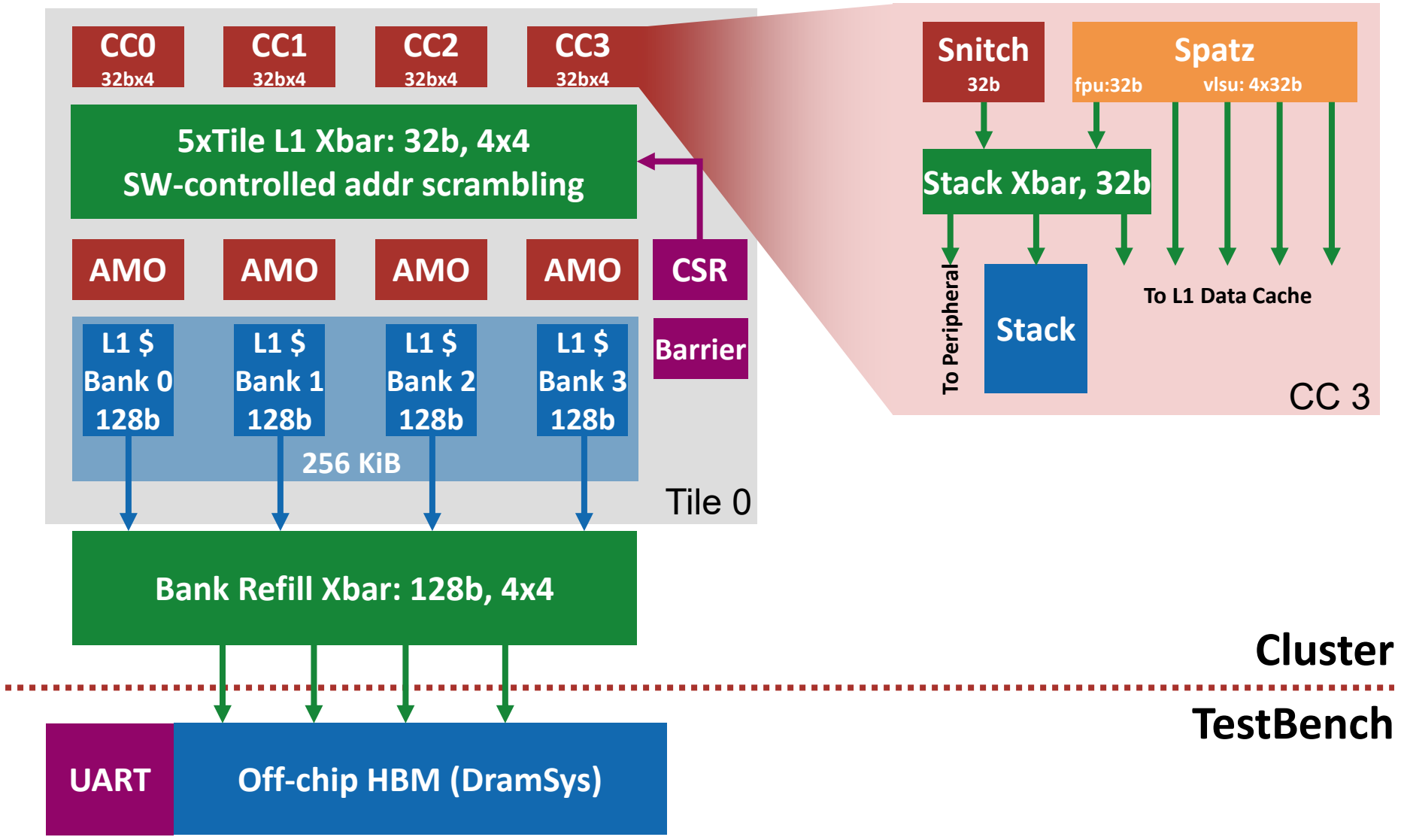
youtube.com/pulp_platform

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

Hardware Development



Hardware Development



Hardware Development

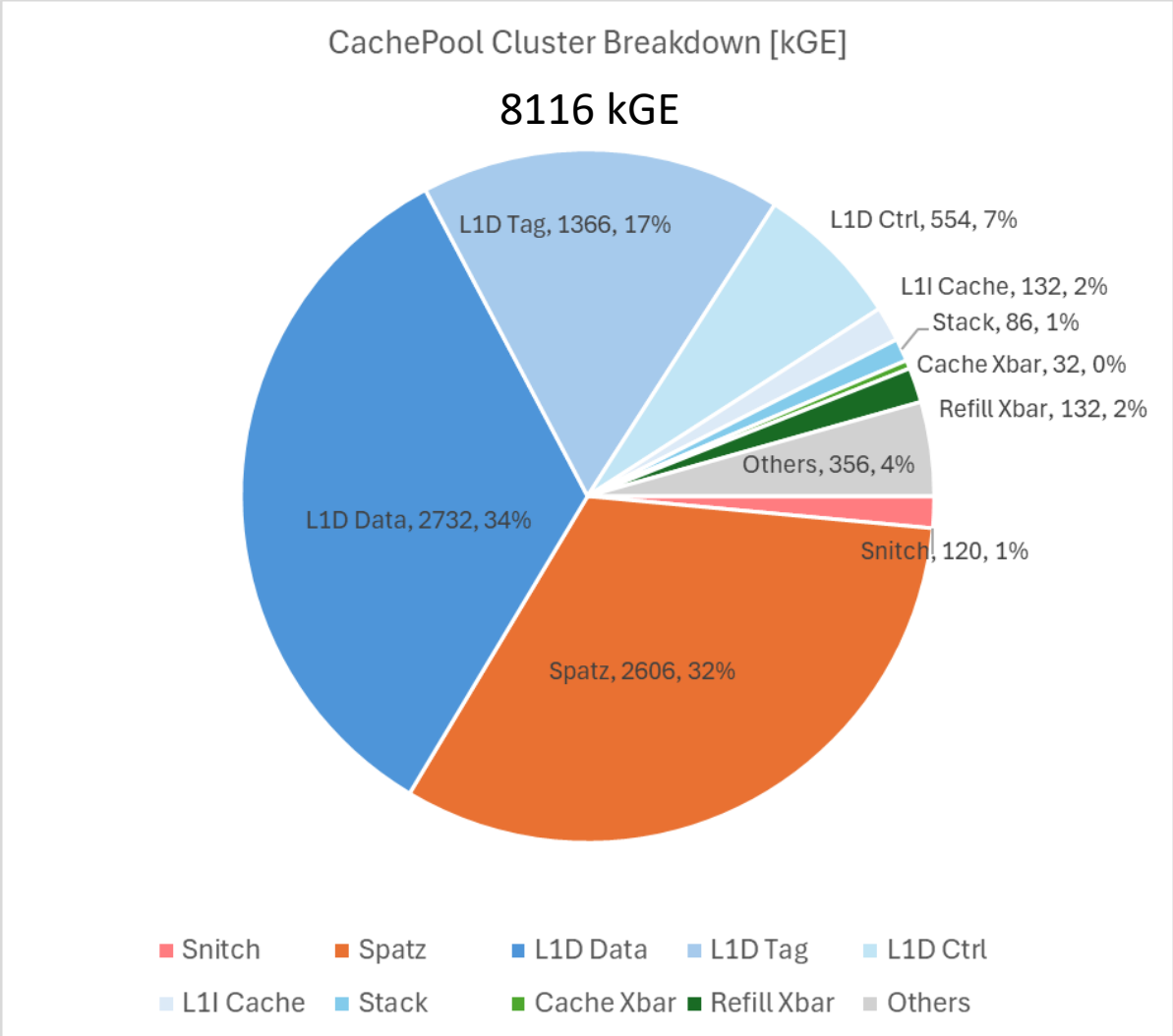
- **Add Stack overflow protection**
  - Re-route the overflowed access to data cache
  - Core ID will be added to the address to separate stack from different cores
  - Private SPM Stack currently kept for faster access
    - **Reduced** from 512x32 (2 KiB) to **128x32 (512 Bytes)** now

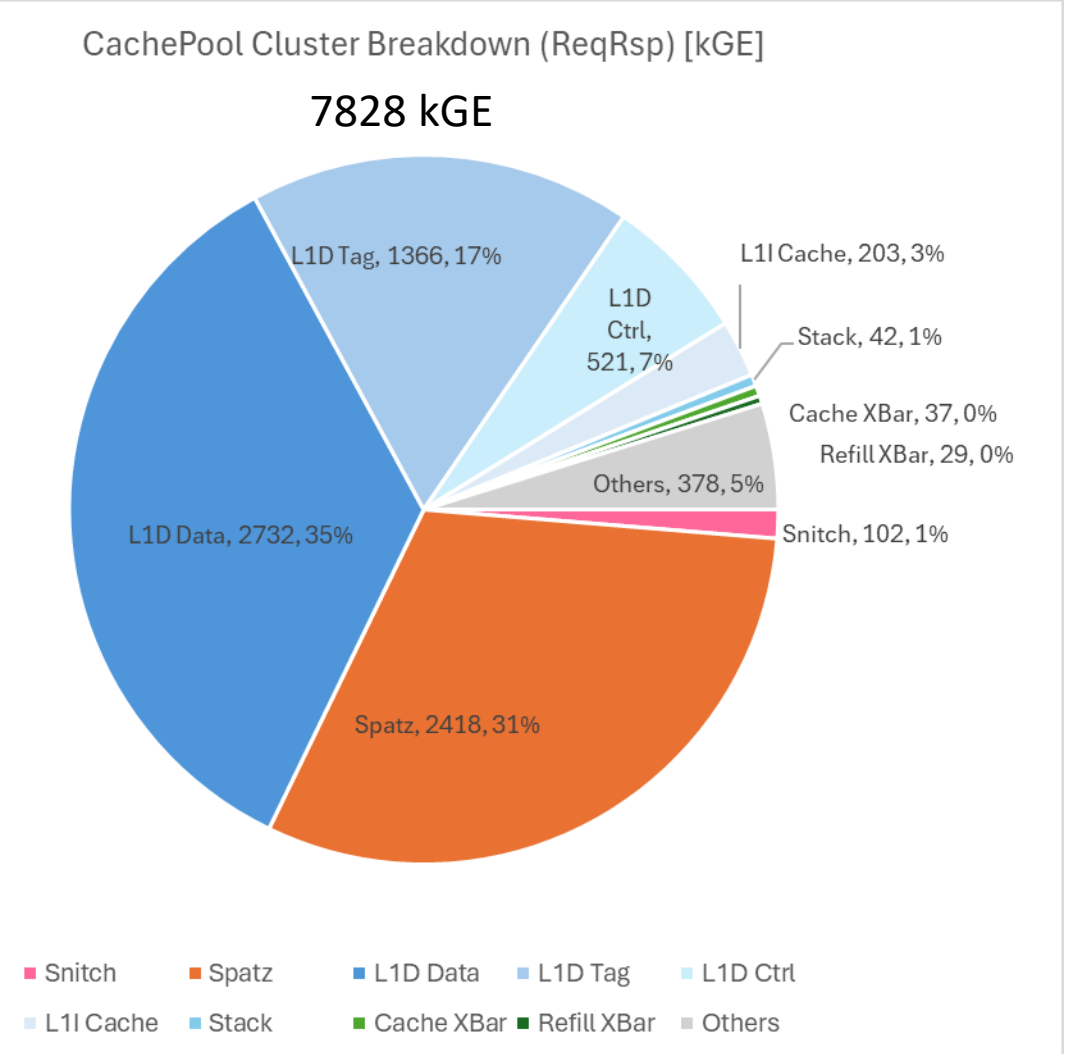
- **Interconnection reform: Change AXI to TCDM/ReqRsp**
  - Change Cache Refill Xbar to use reqrsp interface
  - Convert to AXI at cluster interface
  - BW utilization improved roughly by 4x

PPA Analysis

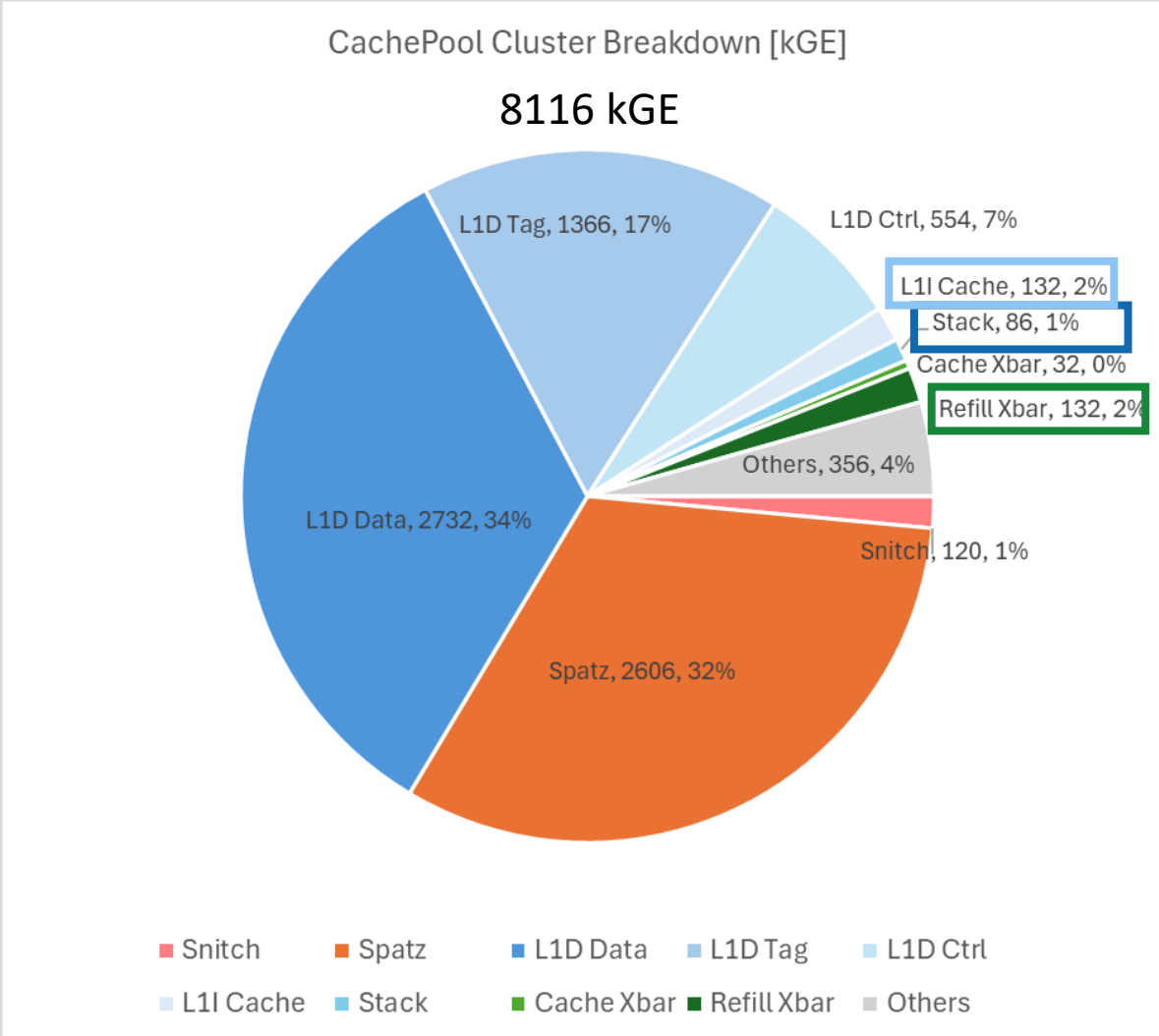
- **Execute a full run on the new cluster**
  - 12nm FinFET tech node
  - 1 GHz @ WW corner
  - Total logic area (including SRAMs): **7828 kGE** (previously **8116 kGE**)
  - Changes
    1. Refill XBar from AXI to ReqRsp
    2. ICache increased from 4KiB to 8KiB (two ways to four ways)
    3. Stack per core reduced from 2KiB to 512 Byte
    4. Spatz ROB depth from 64 to 32
    5. Further clean codes: e.g. remove unused block and modules

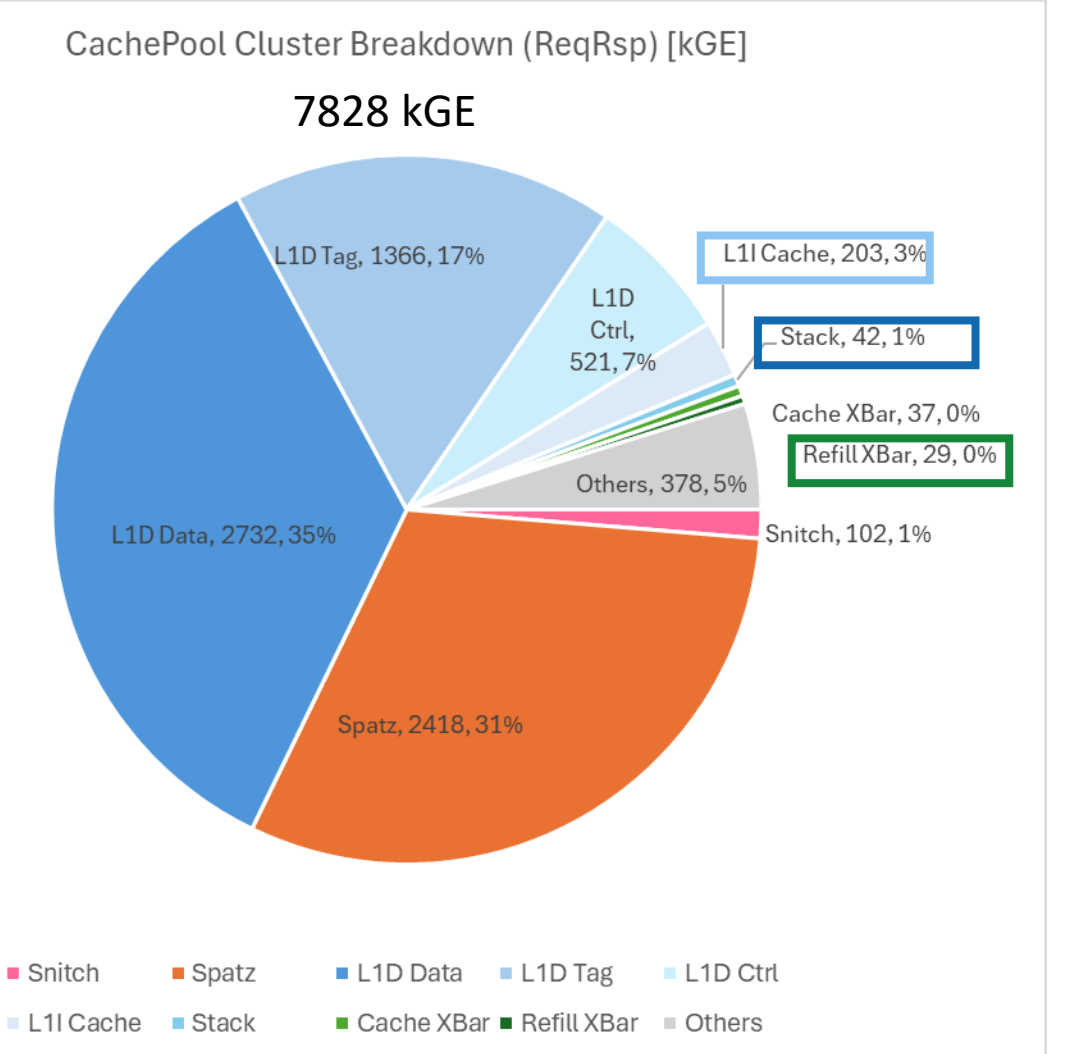
PPA Analysis





PPA Analysis



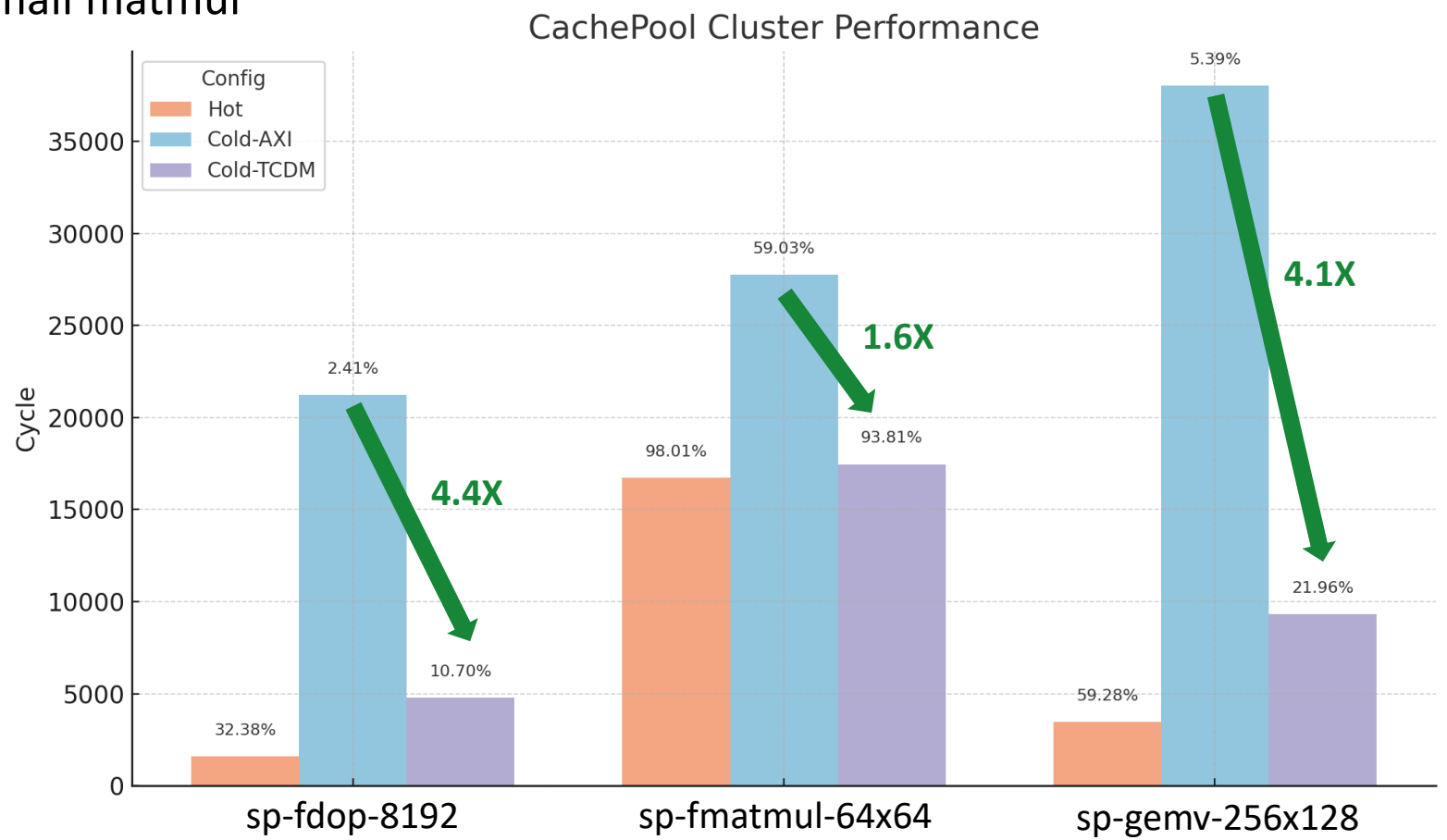


Hardware Development

- **Completed**
  - Solve the BW bottleneck due to AXI implementation
  - Add stack overflow protection
- **WIP**
  - Exploration on the influence on cacheline width
    - DRAM has an access granularity of **512b** (64 Bytes), but our cacheline and interco are **128b** (16B)
    - Currently explore to switch cacheline to 512b while keeping the 128b interconnection
    - Send read burst on request and forward four 128b data one-by-one
    - This serves as a **prefetching** of four next-lines (each core still access 128b per cycle)
- **Todo**
  - Add a configuration without FPU and do PPA analysis

Software Analysis – Vector Kernels

- **Performance improved for mem-bound kernels**
  - ~4 times same as BW improvements
  - Remove BW bottleneck on small matmul



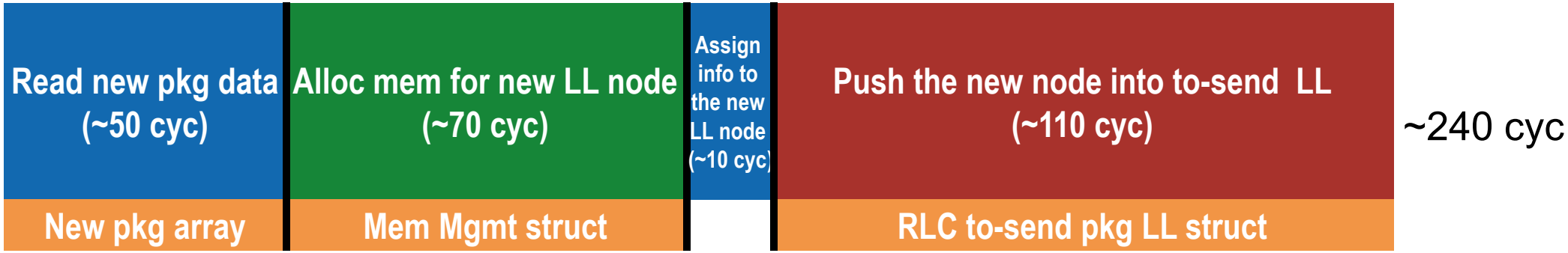
Software Analysis - RLC

- **Kernel Development**
  - Added 1350 Byte version
  - Added header insertion
    - Snitch (scalar core) prepare the header and inform Spatz (vector core) to insert
    - Use vector slide operation to move in the header
- **Performance**
  - The entire vector part (consumer), including sliding, loading and storing can be finished in ~320 cycles on average

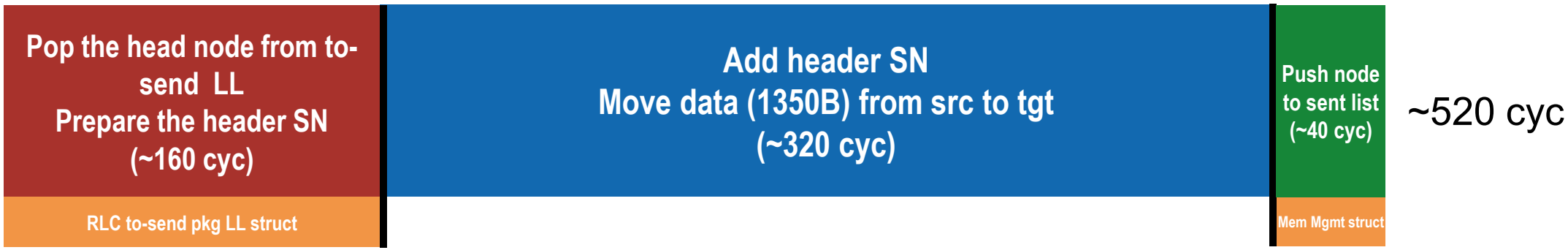
```
// Load
// 1. Medium smaller vector chunks, leading 336 bytes
avl = 84; // load 84 words, 336 bytes
asm volatile("vsetvli %0, %1, e32, m8, ta, ma" : "=r"(vl) : "r"(avl));
asm volatile("vle32.v v16, (%0)" :: "r"(s32));

// 2. Big unrolled chunks, rest 1024 bytes
avl = 128; // load 128 words, 512 bytes per vle, 1024 bytes in total
asm volatile("vsetvli %0, %1, e32, m8, ta, ma" : "=r"(vl) : "r"(avl));

// the 84 is the first 84 bytes, the 4 byte is to overlap that part with the first load,
// so that we can right shift the first load with 1 word (4 bytes) to add RLC header
asm volatile("vle32.v v0,  (%0)" :: "r"(s32 + 84 -1 + 0*elems_per_vreg));
asm volatile("vle32.v v8,  (%0)" :: "r"(s32 + 84 -1 + 1*elems_per_vreg));

// Add header
// 3. Right shift the first load by 1 word (4 bytes) to add RLC header
asm volatile("vslideup.vi v24, v16, 1"); // right shift by 1 word (4 bytes)

// 4. Load a scalar value (SN) into a vector register
asm volatile("vmv.s.x v24, %0" :: "r"(SN)); // move scalar SN into v24
// asm volatile("vor.vv v16, v16, v24"); // bitwise OR, put the SN as the header

// Store
// 5. Medium smaller vector chunks, leading 336 bytes
avl = 84; // store 84 words, 336 bytes
asm volatile("vsetvli %0, %1, e32, m8, ta, ma" : "=r"(vl) : "r"(avl));
asm volatile("vse32.v v24, (%0)" :: "r"(d32));

// 6. Big unrolled chunks, rest 1024 bytes
avl = 128; // store 128 words, 512 bytes per vle, 1024 bytes in total
asm volatile("vsetvli %0, %1, e32, m8, ta, ma" : "=r"(vl) : "r"(avl));
// no need to left shift 1 word here, as we already right shifted the first load
asm volatile("vse32.v v0,  (%0)" :: "r"(d32 + 84 + 0*elems_per_vreg));
asm volatile("vse32.v v8,  (%0)" :: "r"(d32 + 84 + 1*elems_per_vreg));
```

Software Analysis - RLC

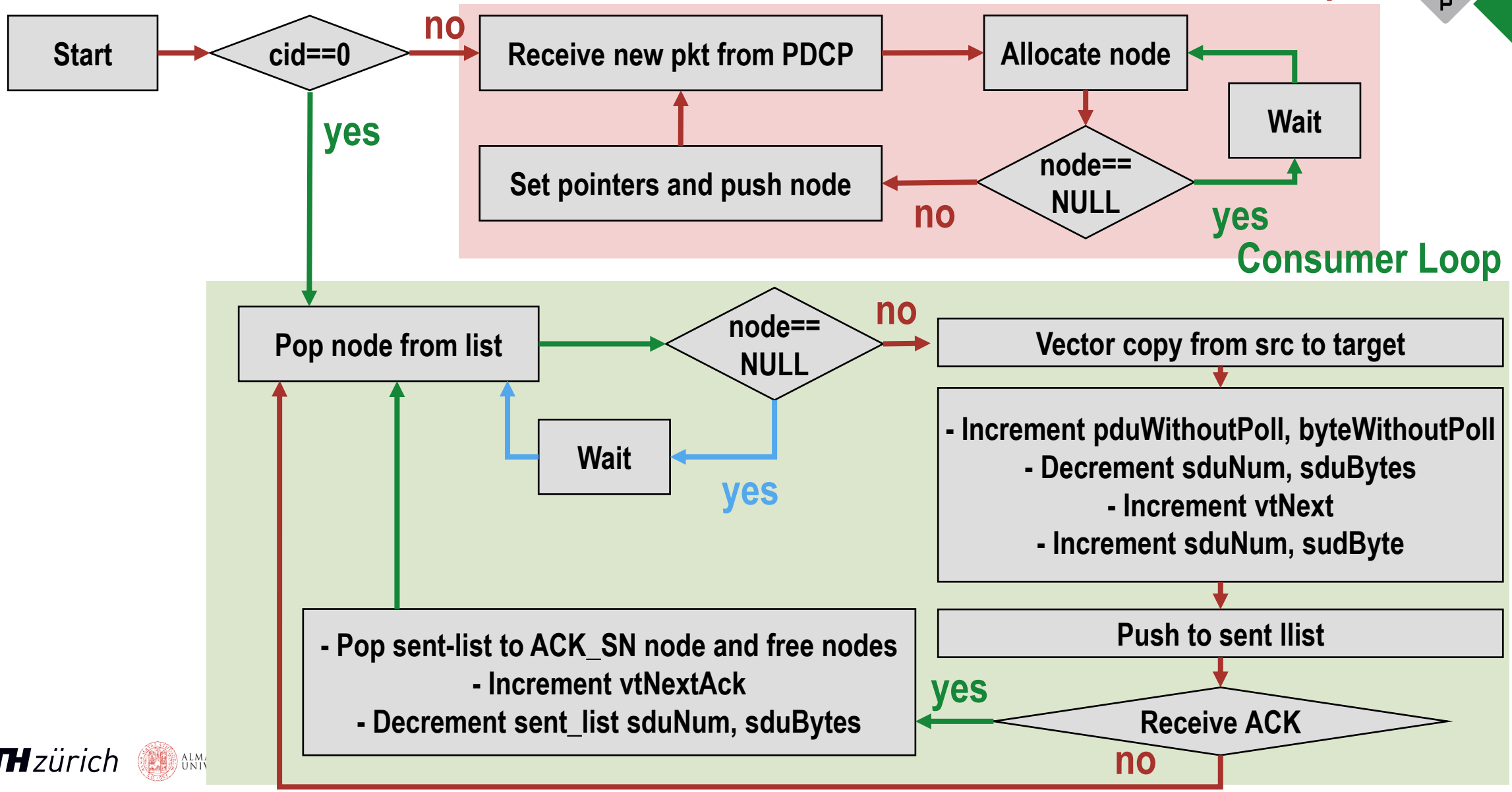
- **Producer: Receive new pkg from PDCP layer, add its info into to-send LL**



- **Consumer: Pop the pkg info from the LL, move pkg data (transmit to UE)**

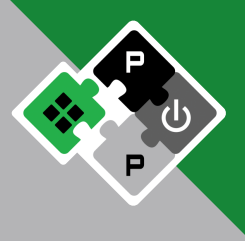


Software Analysis – RLC Block Diagram



Discussions

- **Holiday & Summer School Plans**
  - Zexin is on holiday until August 8th
  - Diyou will attend Summer School from August 4th to 8th
- **Work Package Submission**
  - Cache controller open-source is finally in progress now
    - Will switch to open-sourced version once it's done



Thank you!