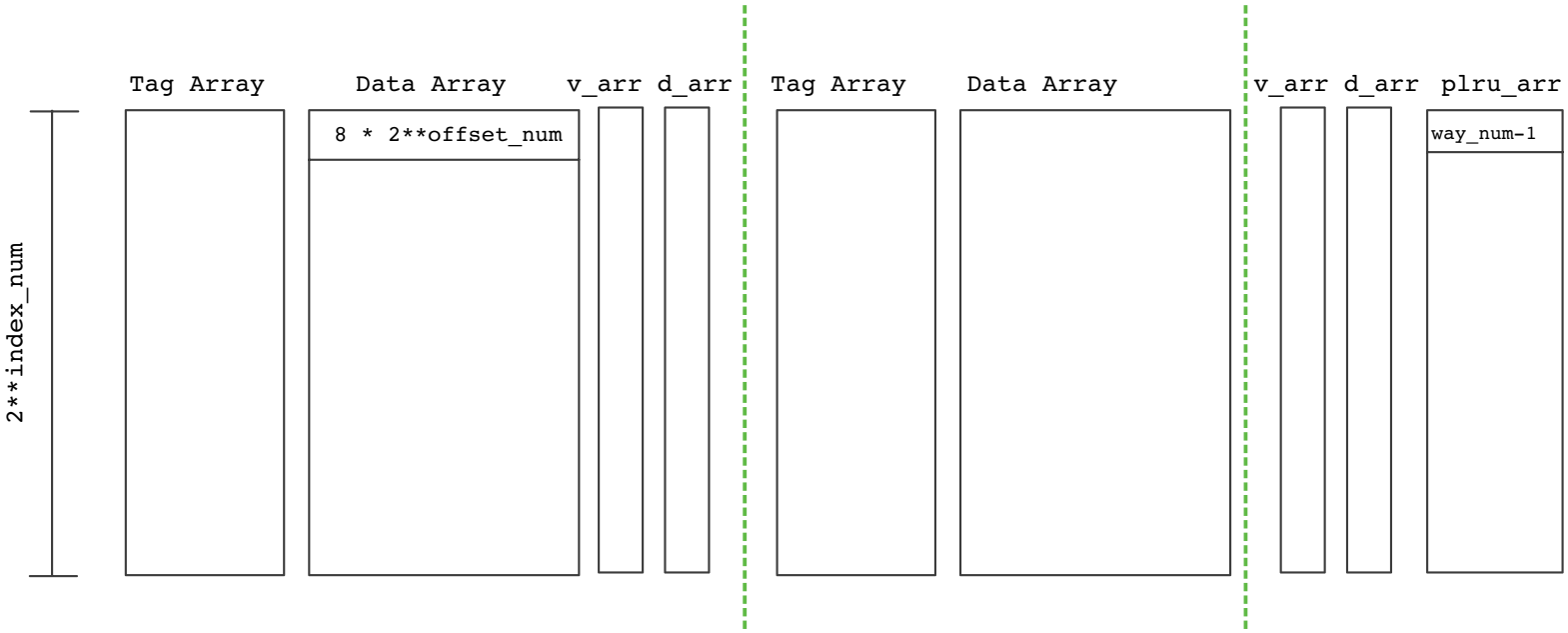
Cache Parameterization

Parameters:
way_num
index_num
offset_num

note: way_num is limited to 2, 4, and 8, for sake of extra design of PLRU logic for each case

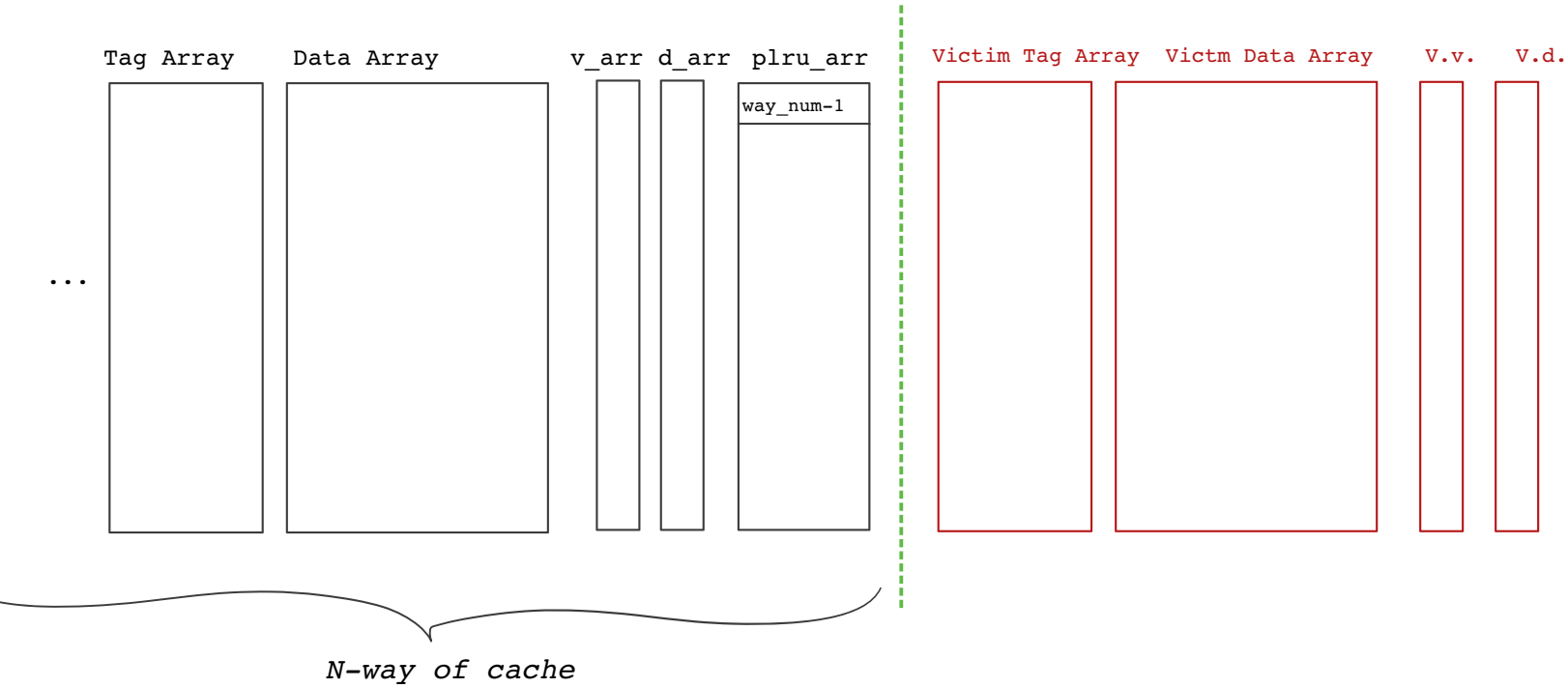
Symbolic Block Diagram (2-way is used as example)



Essential Steps for Parameterization:

1. Apply parameter in each small modules (tag array, data array, regular filp-flop, etc), such that the size of data in each array is parameterized.
2. In the cache_datapath.sv, put the parameter in the outer 'for-loop' to generate needed number of ways
3. Apply the same parameter standard to both cacheline adaptor and bus adaptor.
   A. Cache line adaptor: the number of burst collected will depend on the size of cacheline, which is determined by 'offset_num'
   B. The width of bus adaptor should also be flexible corresponding to 'offset_num' (both data width and the size of 'mem_byte_enable'
4. Design PLRU update logic for all possible "way_num".

Victim Cache

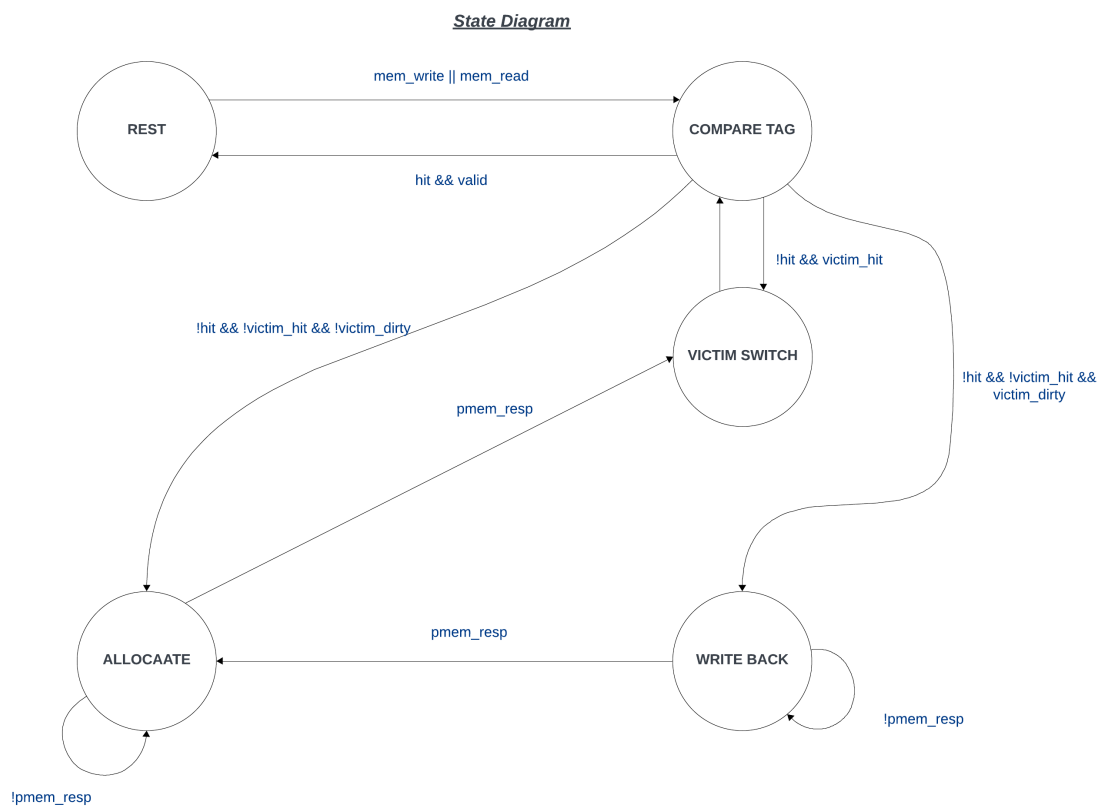


Description:

This concept of victim cache provides the regular cache another layer of buffer before any write-back takes place. So when we instantiate the cache datapath, we will always instantiate victIm tag, data, valid, and dirty arrays.

When a cache miss happens, we will need to further check if the desired data and tag lies within the victim buffer.

A. If it is in victim cache, then we switch this with the PLRU cacheline in regular cache and return a hit.
B. If it's not in victim cache, then we pull out the cacheline from memory (or maybe L2 cache). Use the desired cacheline to evict and replace the PLRU cacheline, and use the evicted cacheline to replace the one in victim cache.



State Explaination

	Description	Transition	Action
REST	Cache wait the signal of 'mem_read' or 'mem_write' from CPU, stay rest	mem_read || mem_write -> COMPARE TAG	

COMPARE TAG	First level check the hit/miss condition happen in cache:
	• If successful write/read can be done in regular cache blocks in this stage, return back to rest
	• If the tag is not found in regular cache but found in victim cache, go to switch the values in next stage
	• If both regular and victimcache miss and the current victim cacheline is dirty, start the process of write back the victim cacheline
	• If both regular and victimcache miss and the current victim cacheline is clean, directly start to fetch/write new data with pmem and ready to load into victim cache
		(hit && valid) -> REST
		
		(!hit && victim_hit) -> VICTIM SWITCH
		
		(!hit && !victim_hit && victim_dirty) -> WRITE BACK
		
		(!hit && !victim_hit && !victim_dirty) -> ALLOCATE
			if (hit):
			load_plru()
			if(mem_read):
			set_data_out(hit_array)
			mem_resp = '1
			elif(mem_write):
			set_data_in(cpu)
			set_data_out(hit_array)
			load_dirty()
			load_data()
			mem_resp = '1

WRITE BACK	Write back the data stored in the victim cachel entry	!pmem_resp -> WRITE BACK pmem_resp -> ALLOCATE	set_data_out(victim_array) pmem_write = 1
ALLOCATE	Fetch data from memory and ready to be stored in victim cache first	!pmem_resp -> ALLOGATE pmem_resp -> VICTIM SWITCH	pmem_read = '1 set_victim_data_in(memory) load_victim_dirty() load_victim_valid() load_victim_data() load_victim_tag()

VICTIM SWITCH	• Switch value between the PLRU entry of regular cache entry and victim cache entry. This can take place in two scenarios:
	◦ miss in regular cache but hit in victim cache, directly switch current value
	◦ miss in both, and the new value from memory is first loaded in victim cache, not it's the time to switch a PLRU entry in regular cache and the new fetched value in victim cache
		Always go to compare tag after this stage	set_victim_data_in(evicted_plru);
			load_victim_tag()
			load_victim_dirty()
			load_victim_valid()
			
			set_data_in(victim);
			load_tag()
			load_dirty()
			load_valid()