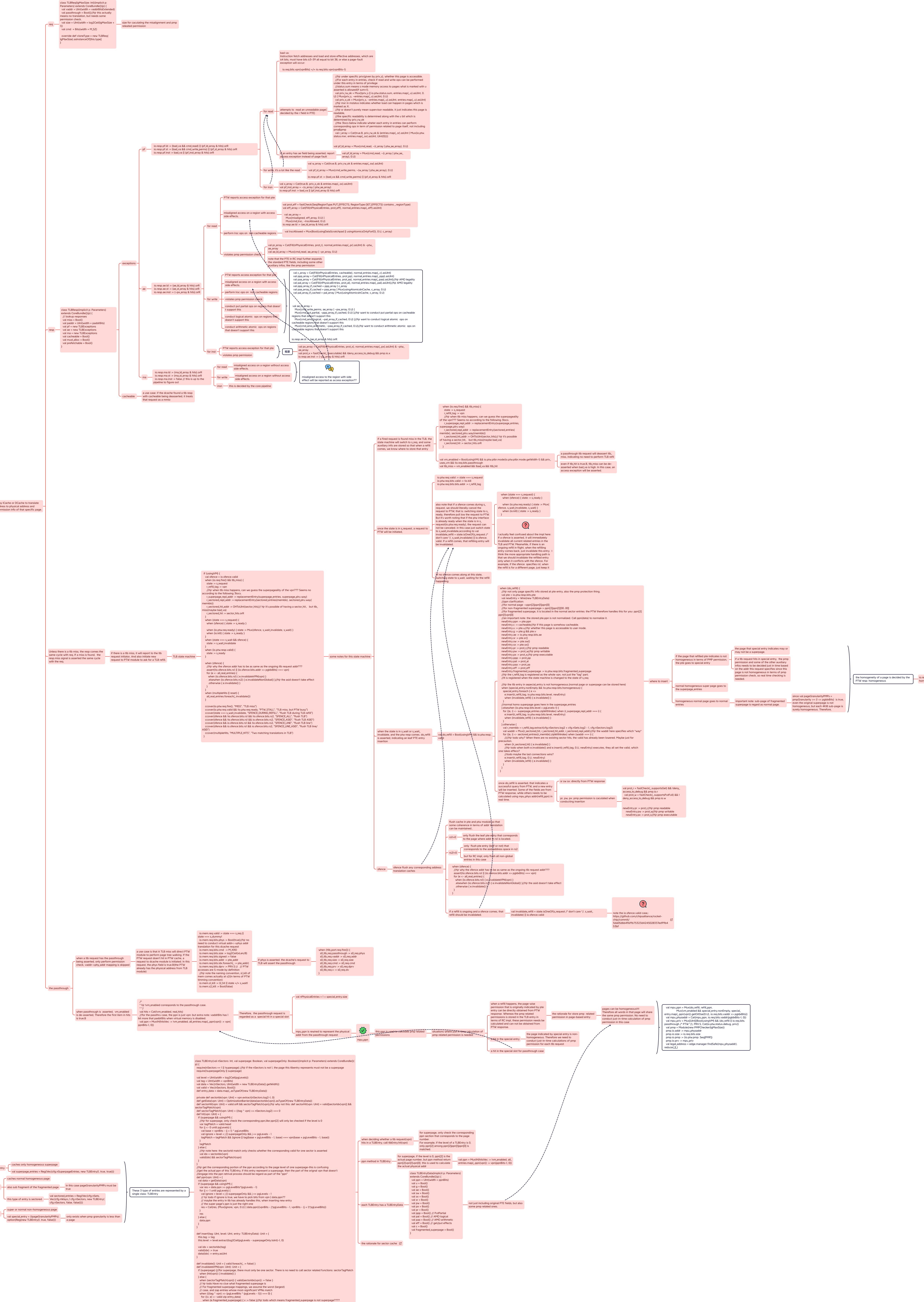


The tlb caches virtual and physical page It is used by ICache or DCache to translate mapping and corresponding permissions for ---- virtual address to physical address and a physical page. obtains permission info of that specific page.

caches normal homogeneous page cache organization — 3 kinds of entries — normal entry — also sub fragment of the fragmented page super or normal non-homogeneous page special entry

> def invalidateNonGlobal(): Unit = { for ((v, e) <- valid zip entry_data) when (!e.g) { v := false }

superpage entry



this page is accessable to user mode.					
adable	the page that special entry	indicates may or			
writable	may not be a superpage.				
xecutable	if the page that refilled pte indicates is not				
	homogeneous in terms of PMP permission, — if a tlb request hits in specia				
	the pte goes to special entry permission and some of the				
	infos needs to be decided ju on the addr this request spe				
:= io.ptw.resp.bits.fragmented_superpage	page is not homogeneous i				we can see that even if a superpage is not
as the whole vpn, not just the "tag" part;	permission check, so real til				homogeneous, but pageGranularityPMPs will
nachine is changed to the state of s_req	needed.		the homogeneity of a page is decided by the	io.requestor(i).resp.bits.homogeneous :=	pull high the homogeneous, because we
	where to insert		PTW resp: homogeneous	homogeneous pageGranularityPMPs	treat each fragment of a superpage being a
y is not homogeneous.(normal page or superpage can be stored here)	normal homogeneous super page goes to				normal homogeneous page.
& !io.ptw.resp.bits.homogeneous) {	the superpage_entries				
bits.level, newEntry)					
lidate() }		since val pageGranularityPMPs =			
	homogeneous normal page goes to normal important note: sub-page of	pmpGranularity $>= (1 << pgldxBits)$ is true,			
nere in the superpage_entries	entries superpage is regard as nor				
< pgLevels-1) {		surely homogeneous. Therefore,			
zipWithIndex) when (r_superpage_repl_addr === i) {					
bits.level, newEntry)					
lidate() }					
ct(cfg.nSectors.log2 + cfg.nSets.log2 - 1, cfg.nSectors.log2)					
r_sectored_hit_addr, r_sectored_repl_addr)//hjr the waddr here specifies which *way*					
nemldx).zipWithIndex) when (waddr === i) {					
are no existing sector hits, the valid has already been lowered. Maybe just for					
date() }					
ate() and e.insert(r_refill_tag, 0.U, newEntry) executes, they all set the valid, which					
ions wins?					
try)					
lidate() }					
sr sw sx: directly from PTW response					
ates a new entry					
are from val prot_r = fasto	Check(supportsGet) && !deny				
access_to_debug					
val prot_w = ta:	stCheck(supportsPutFull) && !				
pr, pw, px: pmp permission is caculated when deny_access_to_ conducting insertion	debug && pmp.io.w				
newEntry.pr := p	rot_r//hjr pmp readable				
	= prot_w//hjr pmp writable				
	= prot_x//hjr pmp executable				