Delayed Partial Parity Scheme for Reliable and High-Performance Flash Memory SSD

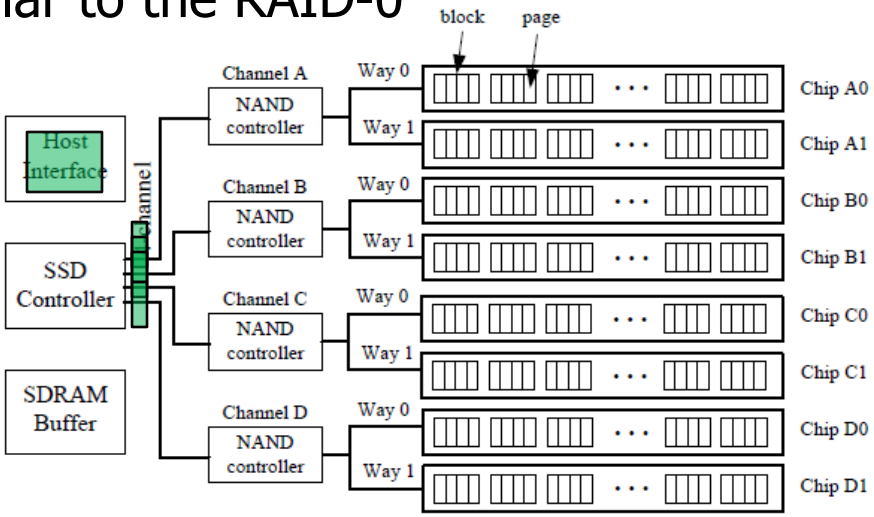
Soojun Im, Dongkun Shin

Sungkyunkwan University, S.Korea

MSST 2010

Background – SSD Architecture

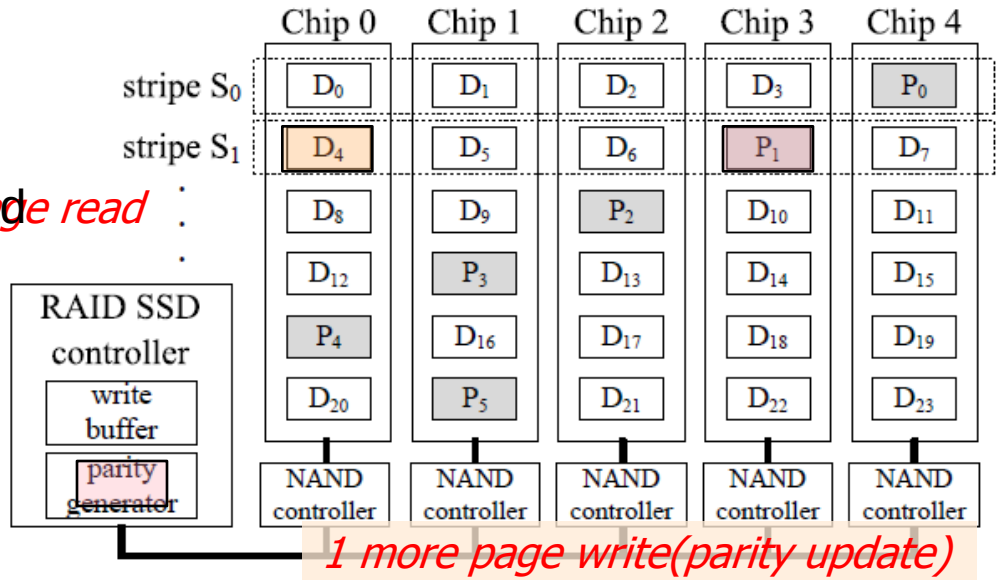
- Solid State Disk(SSD)
  - Composed of multiple flash chips
  - Replace HDDs in the mass storage market
- Multi-channel and Interleaving
  - Enhance the bandwidth of SSD
  - Channels can be operated simultaneously
    - Chips using different channels can be operated independently
    - Similar to the RAID-0



RAID-5 SSD

- SSD's reliability ➔ ECC (?), still a critical issue
  - MLC flash memory shows a much higher bit-error rate than can be managed with single bit-ECC
  - increase the read and write latencies
- For reliability, use redundancy in storage level
  - Implement RAID-5 SSD
  - Small write incurs significant overhead

Total parity update overhead
$= (2 * T_{read} + T_{write}) / T_{write}$



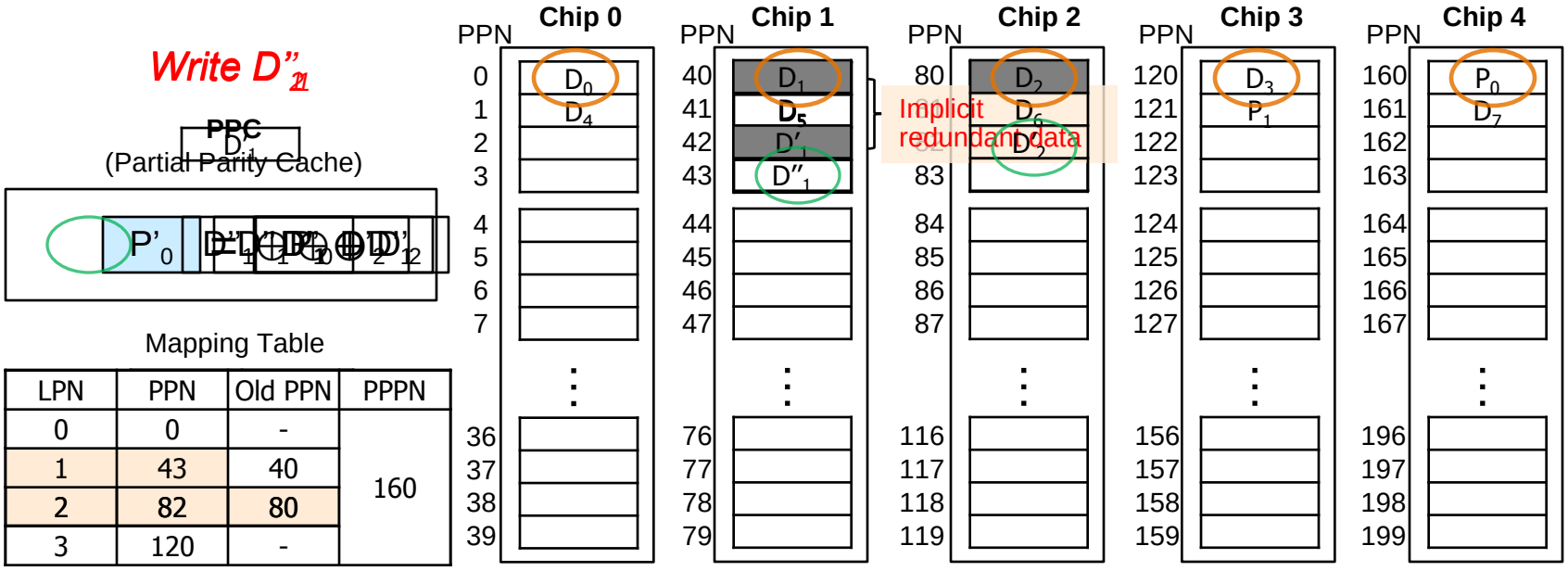
Delayed Partial Parity

- When there is an update requests, FTL generally does not erase or update old data
  - Instead, it is invalidated due to the erase-before-write constraint ➔ implicit redundant data
  - Exploit the implicit redundant data in order to reduce the parity handling overhead

Delayed parity update

Delays the parity data update and stores it on a special device called a partial parity cache (PPC)

generated with only partial data of stripe

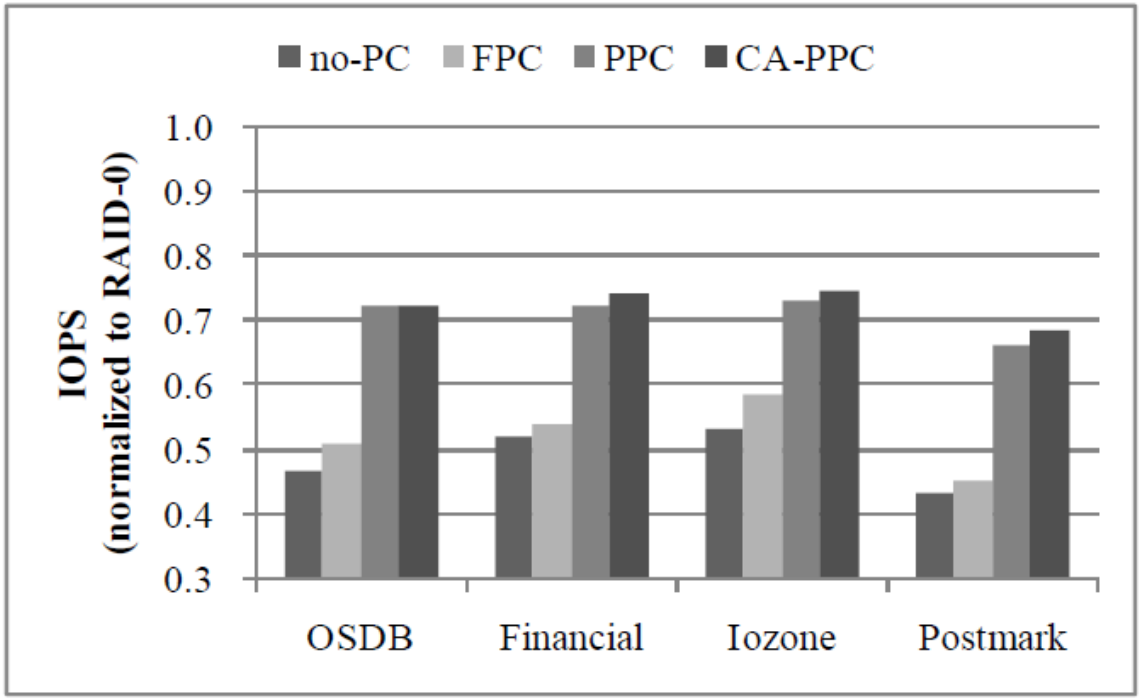


Mapping Table

LPN	PPN	Old PPN	PPPN
0	0	-	160
1	43	40	
2	82	80	
3	120	-	

IOPS

- Avg. values of the normalized IOPS are 0.48, 0.51, 0.70 and 0.72
- PPC improves the performance by 46% and 37% on average compared to those of no-PC and FPC, respectively

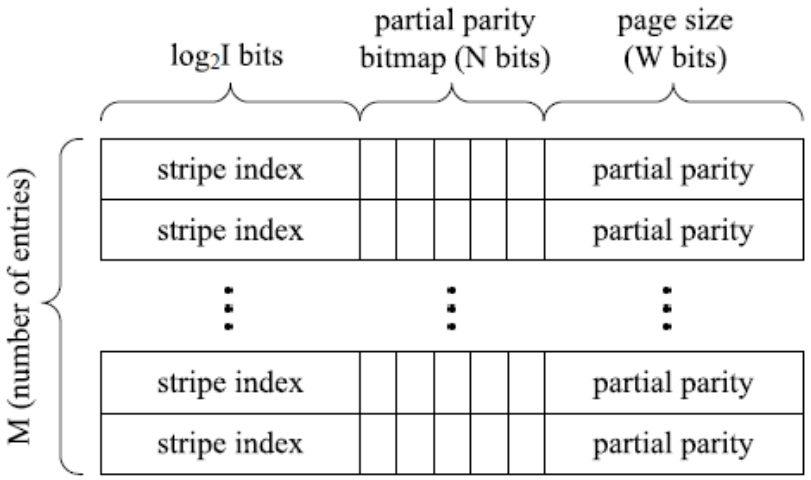


Conclusion

- Efficient RAID techniques for reliable flash memory SSDs
  - The delayed parity update technique
    - Reduces the number of flash memory write operations
  - The partial parity caching technique
    - Exploits the implicit redundant data of flash memory
    - Reduce the number of read operations required to calculate the parity
    - Reduce GC overheads in flash memory

Partial Parity Cache

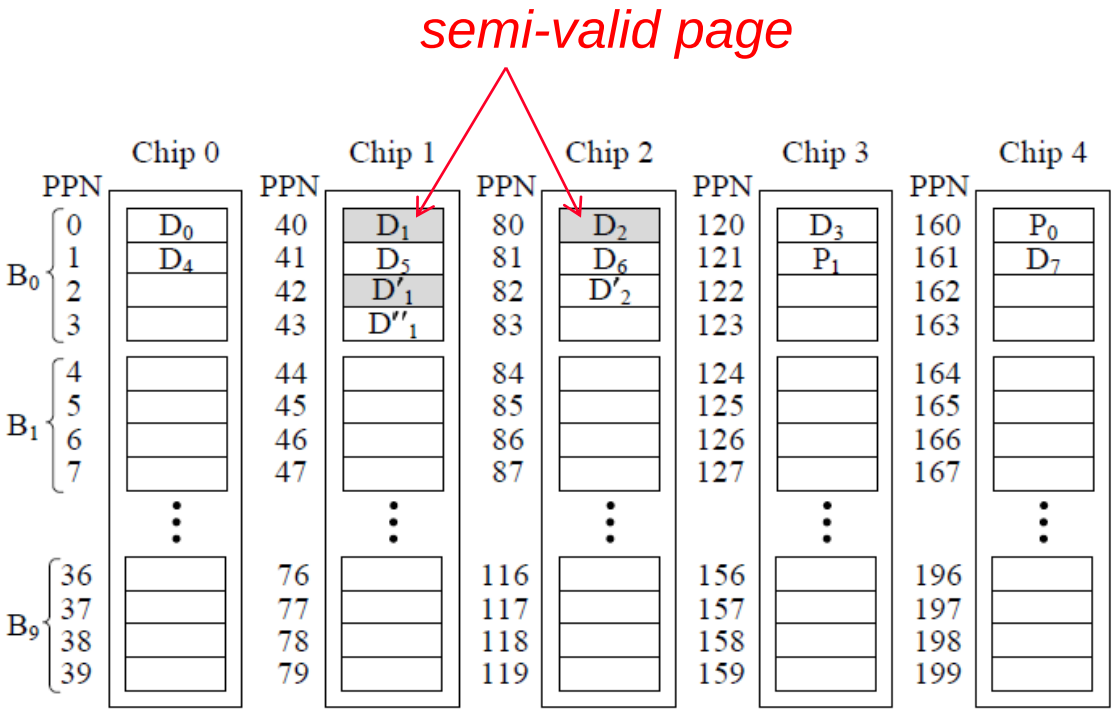
- Stores the delayed parities temporarily
- NVRAM in order to avoid losing the parities stored in the PPC at sudden power failure
- Uncommitted stripe whose up-to-date parity is not written to a flash chip
- Bitmap: the data indices associated with the partial parity
- Size of PPC: $M(\log_2 I + N + W)$ bits



Main Idea

- Delayed parity update
  - Delays the parity data update and stores it on a special device called a partial parity cache (PPC)
    - generated with only partial data of stripe

Partial Parity Creation & Updating



① Write D′1
② Write D′2
③ Write D″1

PPC

after writing D′1	$\tilde{P}_0 = D'_1$
after writing D′2	$\tilde{P}_0 = D'_1 \oplus D'_2$
after writing D″1	$\tilde{P}_0 = D''_1 \oplus D'_2 = D''_1 \oplus \tilde{P}_0 \oplus D'_1$

mapping table

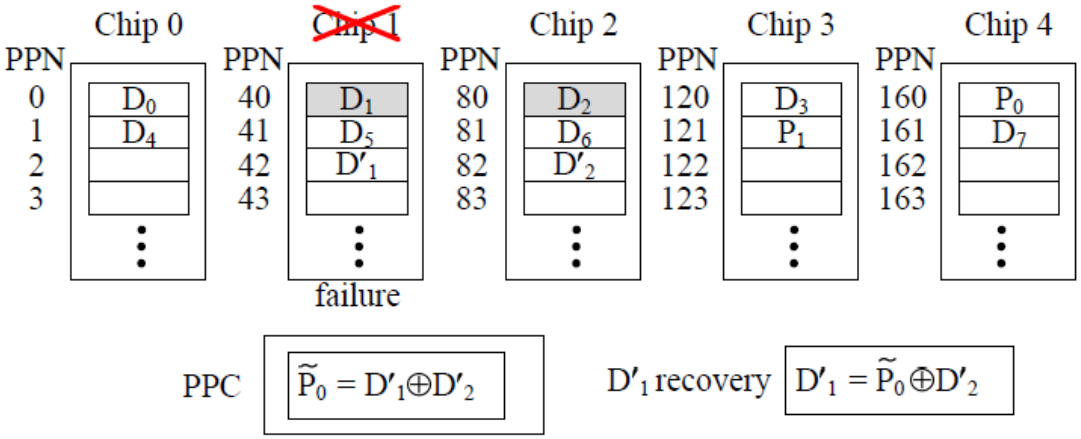
LPN	PPN	Old PPN	PPPN
0	0	-	160
1	43	40	
2	82	80	
3	120	-	

Partial Parity Commit

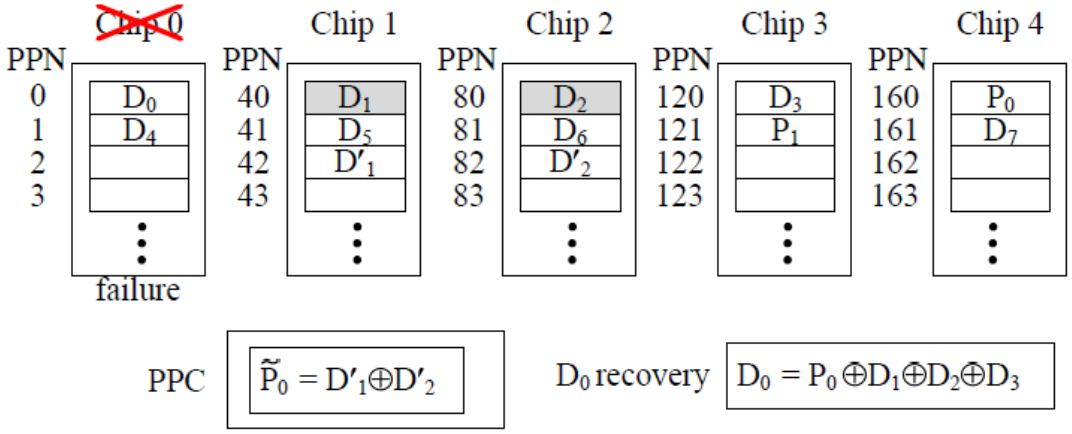
- Two kinds of parity commits
  - Replacement Commit
    - When there is no free space in the PPC for a new partial parity, one of partial parities should be replaced
  - GC Commit
    - Before the GC erases the semi-valid pages of the uncommitted stripe, the corresponding delayed parity should be committed
- Before the commit of partial parity, RAID controller should first build the full parity

Chip Failure Recovery

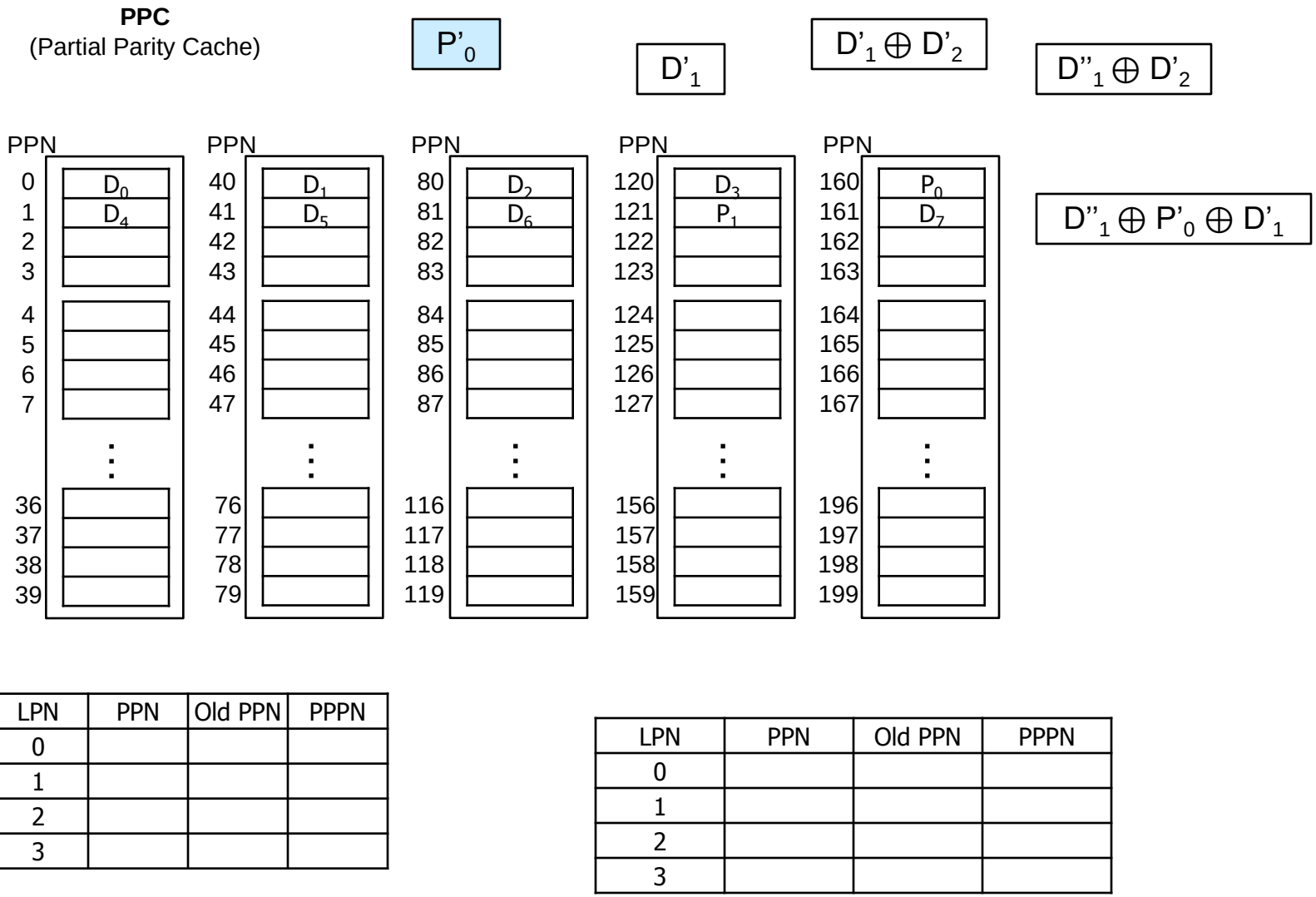
- When a partial parity is associated with the failed page



- When no partial parity is associated with the failed page



LPN	PPN	Old PPN	PPPN
0	0	-	160
1	43	40	
2	82	80	
3	120	-	



LPN	PPN	Old PPN	PPPN
0			
1			
2			
3			

LPN	PPN	Old PPN	PPPN
0			
1			
2			
3			