HIL: FTL design framework with provably-correct crash recovery

NVRAMOS 2013. 10. 25

Eyee Hyun Nam, eyeehyun.nam@sk.com

Storage Tech. Lab. SK Telecom

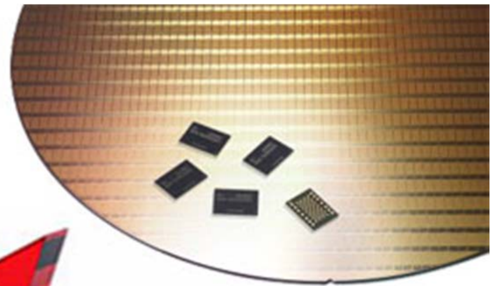
(In cooperation with Memory & Storage Architecture Lab. SNU)

Contents

- Introduction
- Motivation
- HIL Framework
- Correctness Verification
- Conclusion

Introduction

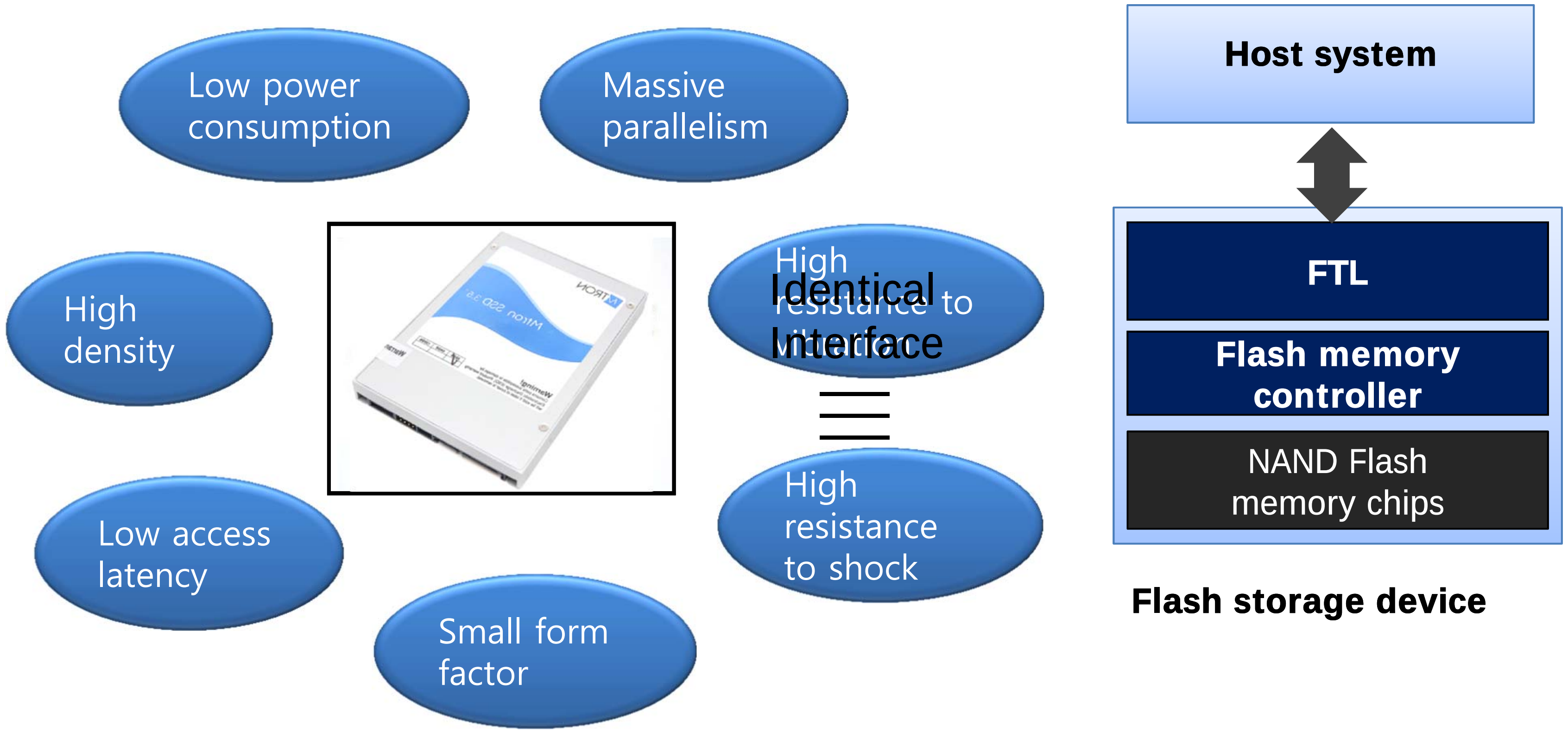
- **Flash memory is ubiquitous**



[Source: storagelook.com]

Flash Storage Device

- **Provides an interface identical to a block device, but uses flash memory as a storage medium**



Recent Trend & Our Goal

Worsening Characteristics

Performance

-Longer latency

Reliability

-Retention/Endurance

-Disturbance/Interference

-Sibling page problem

[Requirement]

Fast & Reliable Storage

Time to market, cost, & reusability

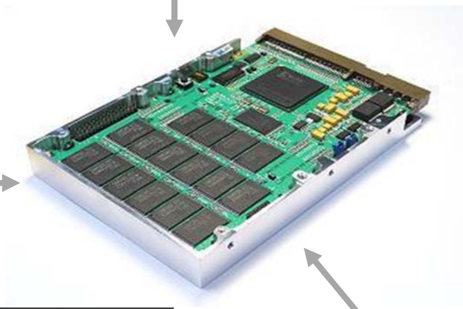
[Goal]

Maximal exploitation of diverse parallelisms

Provably correct flash management SW

Modular / extensible / compositional architecture

Flexible trade-off between performance and cost



Abundant Parallelisms

Host system and FTL

-Multi-core/Multi-threaded SW

Flash memory

-Multiple flash buses/chips

Host interface

-NCQ/TCQ/...

Increasing Diversity

Applications

-File system/DB/Virtual Memory/...

Flash memory

-ONFI/Toggle/HLnand

FTL

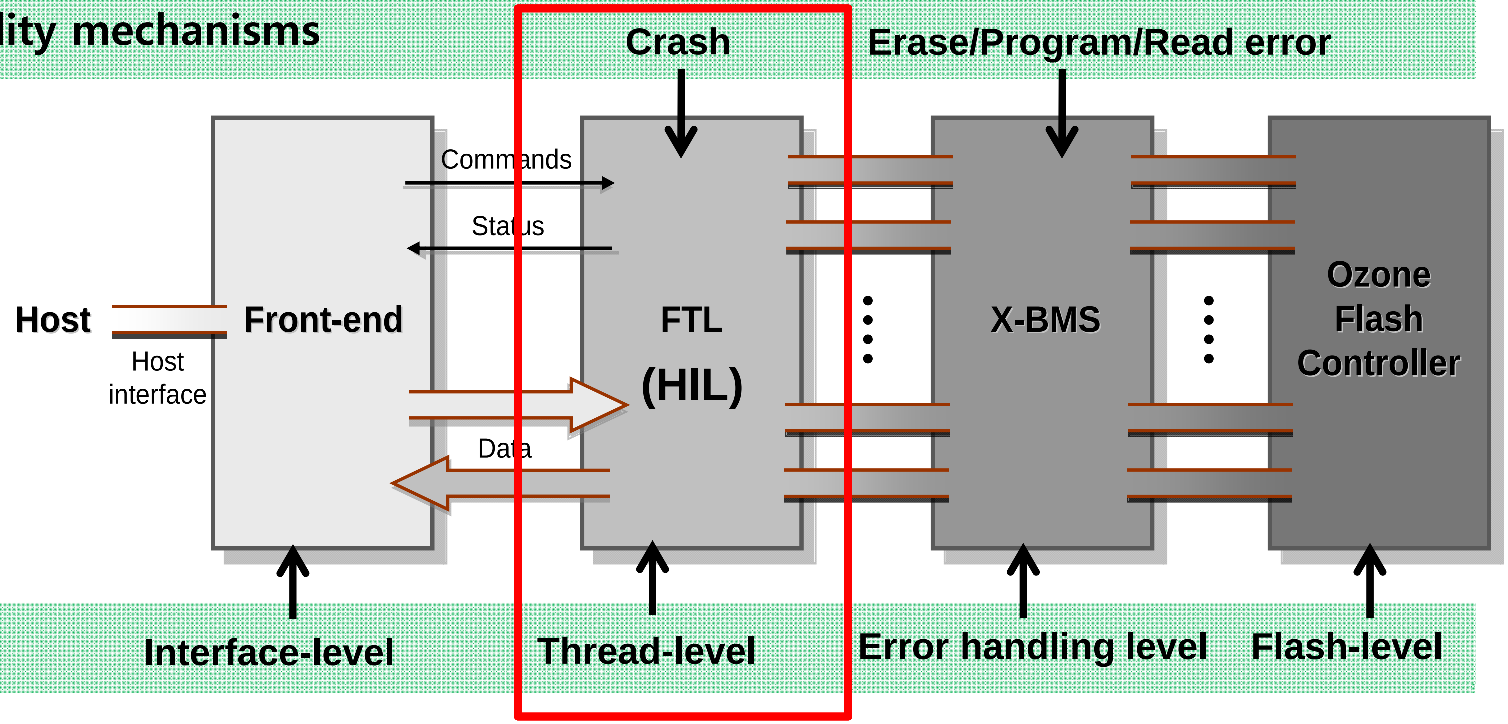
-Page-mapped/Block-mapped/Hybrid-mapped

Host interface

-SATA/PCIe/UFS/eMMC

Key Enabling Technologies

- HW/SW co-designed/co-optimized system architecture
- Packet-based interfaces
- Built-in reliability mechanisms



- Parallelisms

•Nam, E.H., Kim, S.J., Eom, H., and Min, S.L., "Ozone (O3): An Out-of-order Flash Memory Controller Architecture", *IEEE Transactions on Computers*, vol. 60, no.5, pp. 653-666, Oct. 2011.

•Yun, J. H., "X-BMS: A Provably-correct Bad Block Management Scheme for Flash Memory Based Storage Systems", Ph.D. Dissertation, 2011, SNU.

•Yun,J.H, Yoon,J.H, Nam, E.H, Min, S.L., "An Abstract Fault Model for NAND Flash Memory", *IEEE Embedded Systems Letters*, vol.4, no.4, pp.86-89, Dec. 2012.

•Y.J. Sung, "Formal verification of a compositional FTL design framework", Ph.D. Dissertation, 2013, SNU.

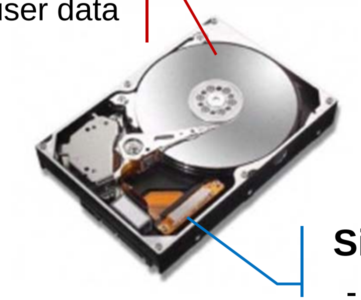
•H.S. Kim., "Design and implementation of a parallelized bad block management scheme", Ph.D. Dissertation, 2013, SNU.

Motivation (1)

- **“Flash storage is now a computer system!”**

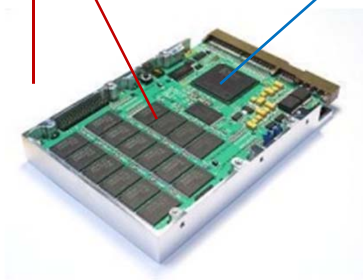
[Traditional storage system]

(mostly)
user data



Single-core processor
- buffering, I/O scheduling

User data
+
FTL metadata
- Mapping info
- Block info
- Checkpoint
⋮



Multi-core processor
- Remapping
- Garbage collection
- Wear-leveling
- Write buffering
- Host command queuing
- Interleaving (RAID)
- Crash recovery
⋮

[Flash storage system]

Motivation (1)

- **Plethora of FTLs**

HFTL

SAST SFTL MS FTL BPLRU

BFTL AFTL FAST LazyFTL

KAST

Chameleon CNFTL DFTL

LAST MNFTL

super-block scheme CFTL

Log block scheme

GFTL μ-FTL JFTL zFTL

Hydra FTL Vanilla FTL Replacement block scheme

YanusFTL

Reconfigurable FTL

...........and so on

WAFTL UFTL

[List of questions]

How do they do

- Mapping?
- Wear-leveling?
- Garbage collection?
- Write-buffering?
- Crash recovery?

⋮

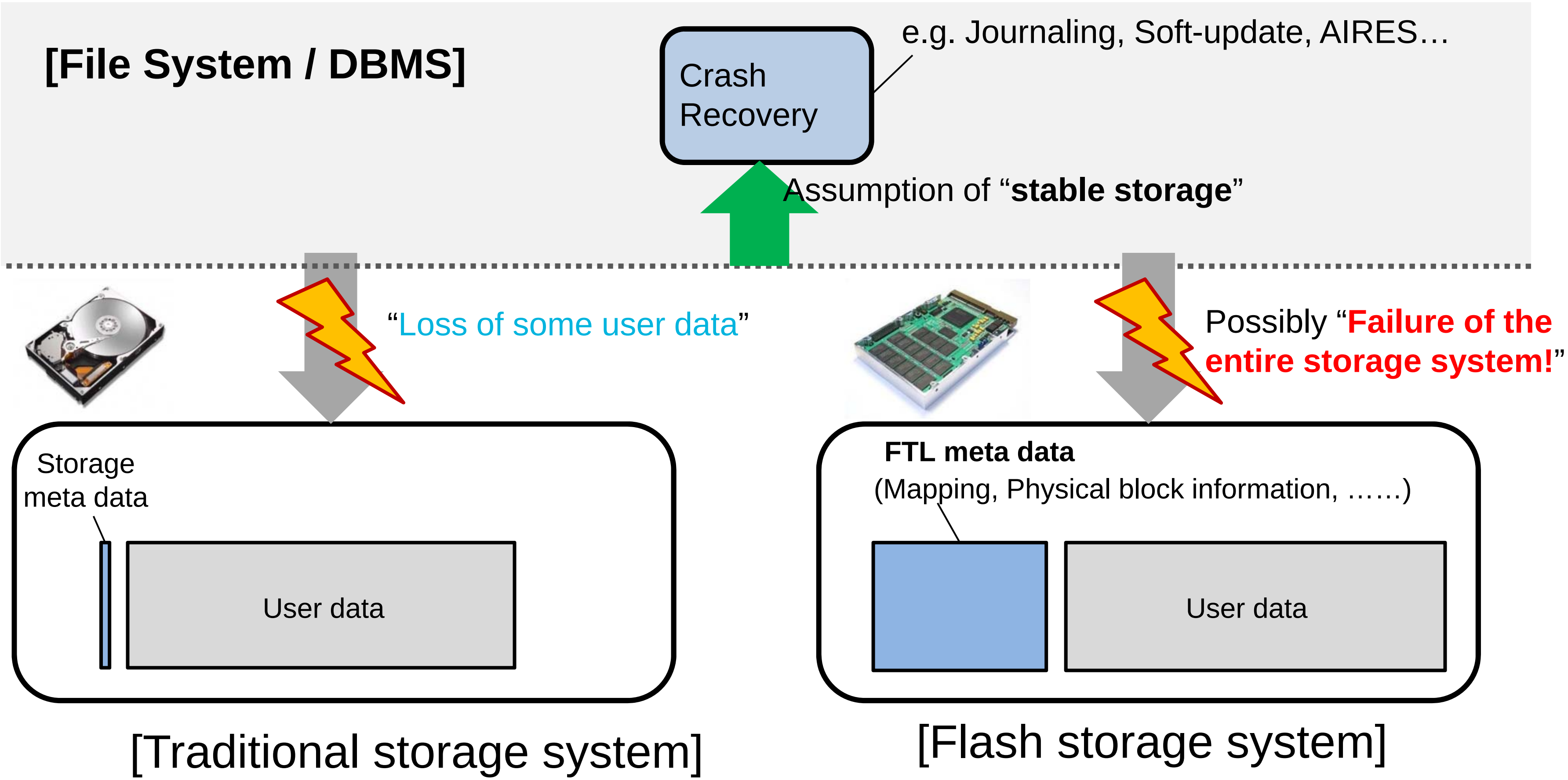
??z

?

No one relieves our worries…

Motivation (2)

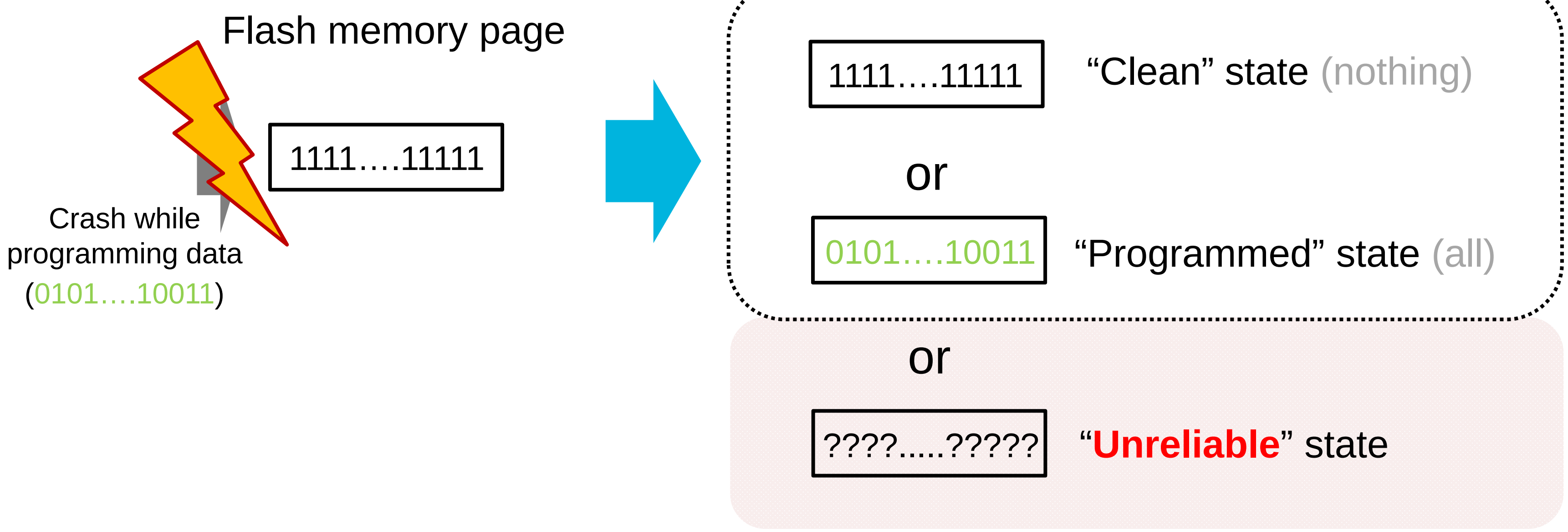
- **“Crash recovery is not only a system-software issue!”**



Motivation (2)

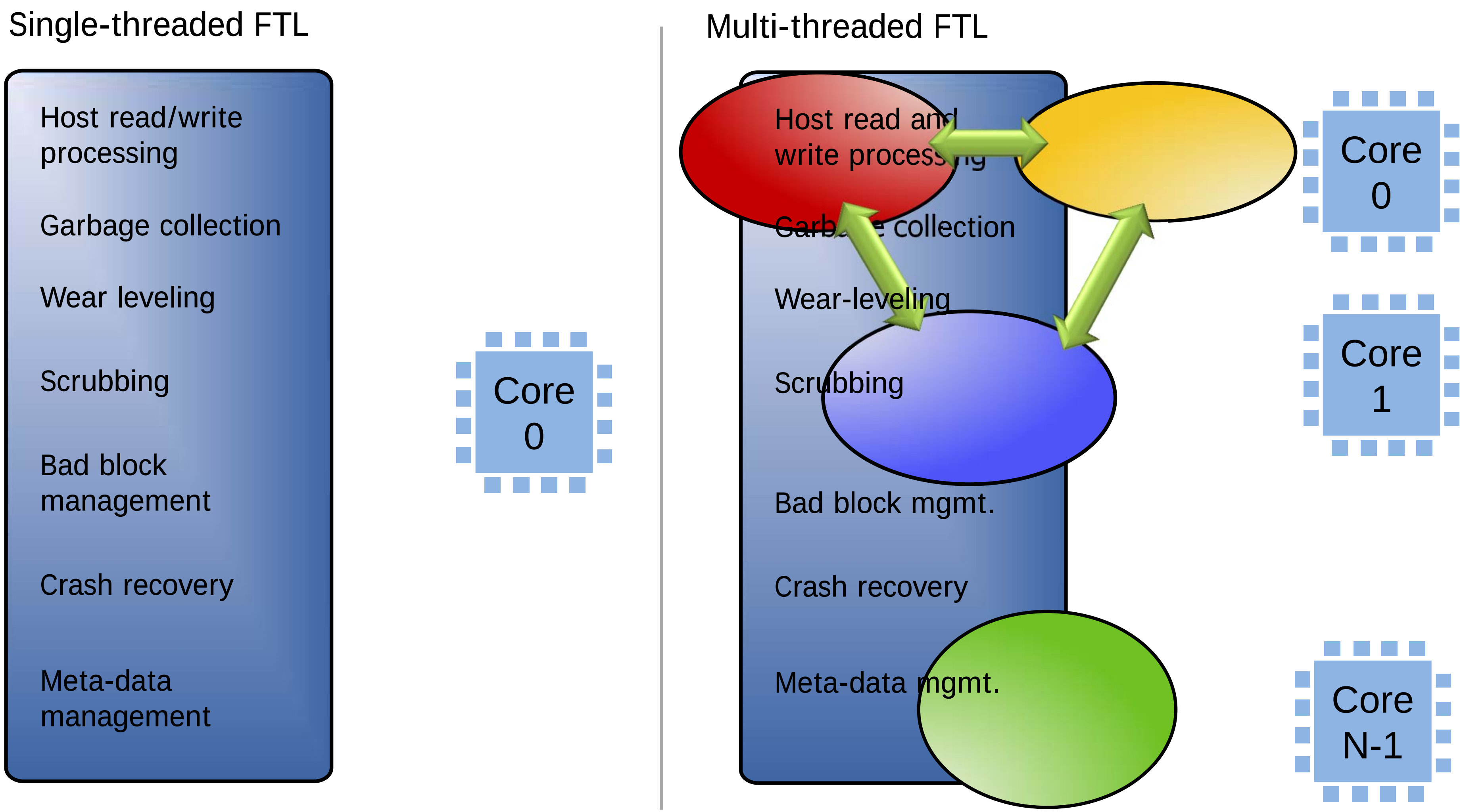
- **Challenges of crash recovery**
  - Asynchronous
  - Nested crash
  - Non-atomic page programming
  - Sibling page problem

Crash recovery of current FTLs are based on the assumption of "atomic programming"



Motivation (3)

- **"Many-core is not special any more inside SSDs"**

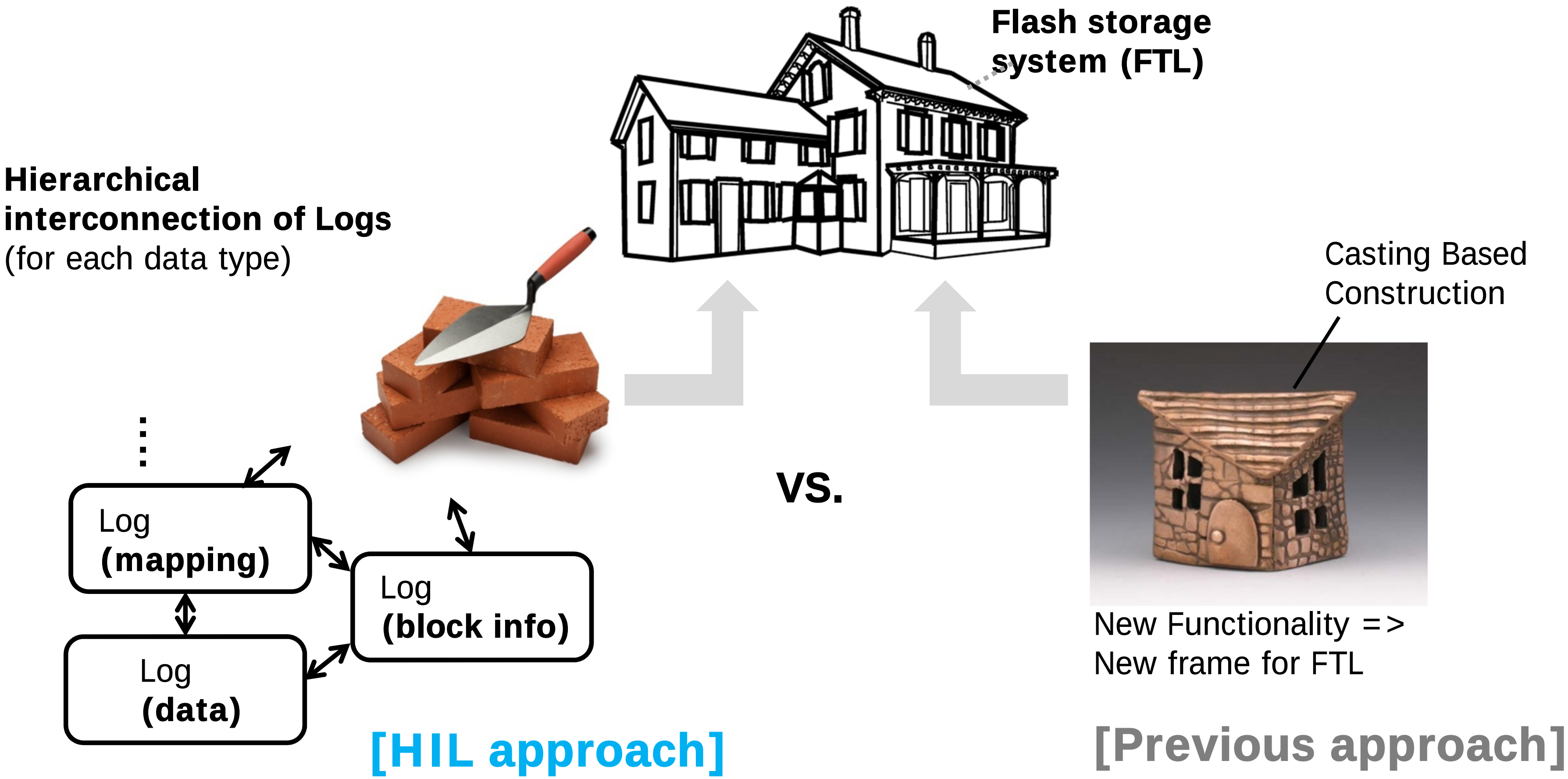


HIL framework

- **HIL (Hierarchically Interacting a set of Logs)**
  - A general FTL design framework that systematically solves crash recovery problem with following key aspects.
    - (1) Compositional construction of FTLs
    - (2) Built-in crash recovery mechanism
    - (3) Maximal exploitation of parallelisms

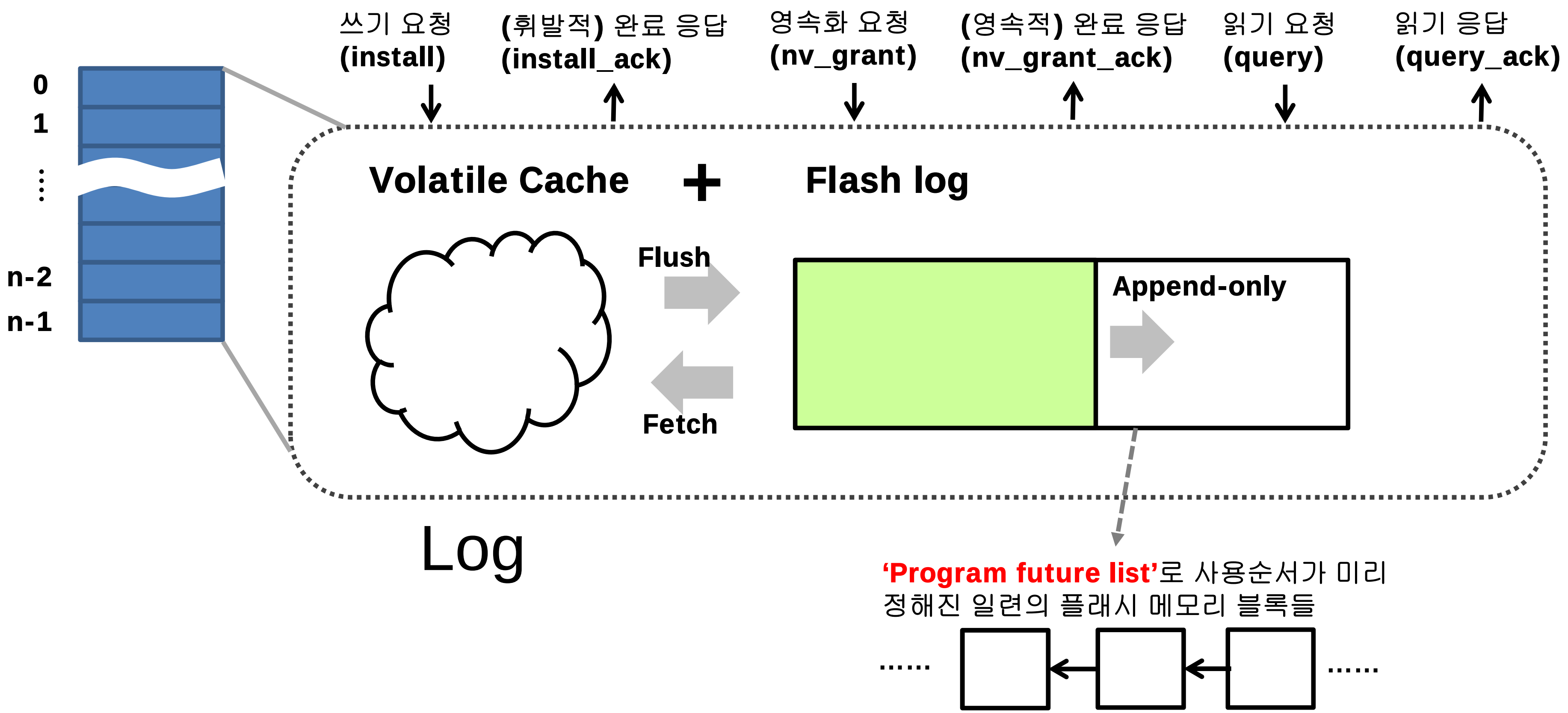
HIL: Compositionality

- **"An FTL is built with the composition of Logs"**



Log

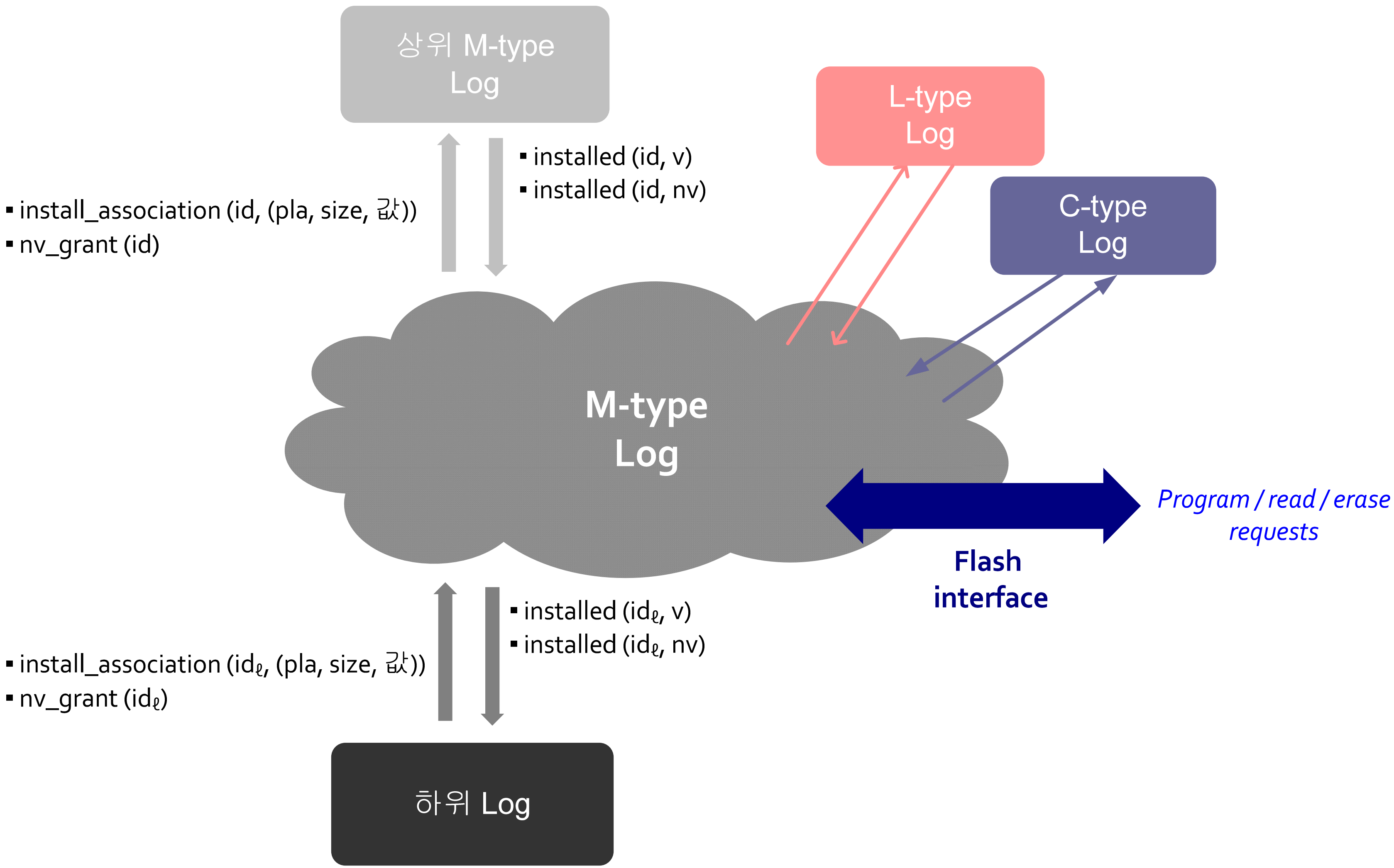
- **A building block of FTLs that provides 1) linear address space where data can be updated in-place and 2) durability of data**



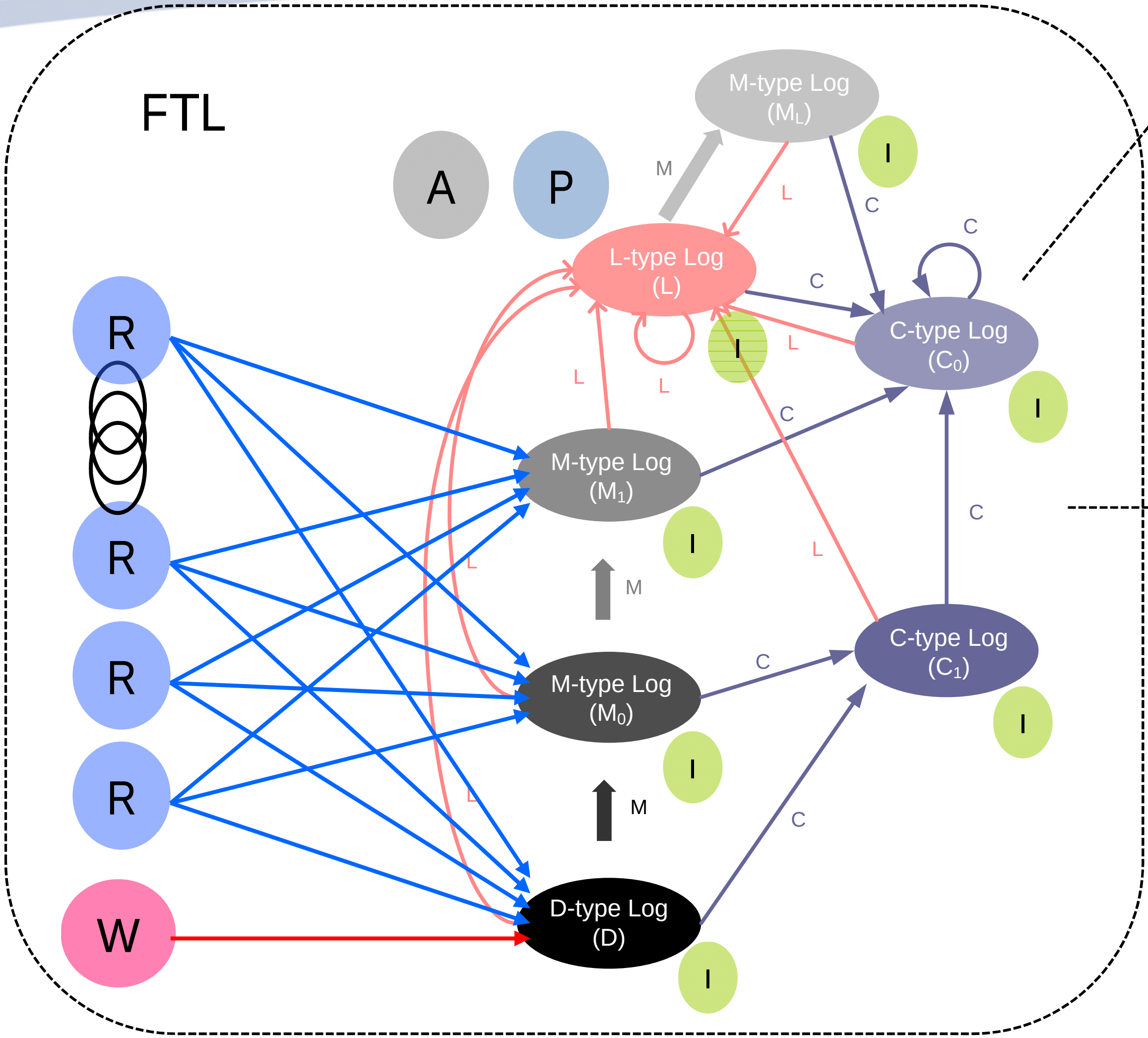
Types of Logs

- **D-type Log (for user data)**
- **M-type Log (for mapping information)**
- **L-type Log (for liveness information)**
- **C-type Log (for checkpoint information)**
- **W-type Log (for non-volatile write buffering)**

Example: A more detailed picture of M-type log



Compositional Construction of an FTL



Interconnection topology

: Diverse mapping scheme

- Block mapping
- Page mapping
- Hybrid mapping

Pluggable

Garbage collection policy

Wear-leveling policy

Buffer mgmt. policy

R : Read Processing Unit

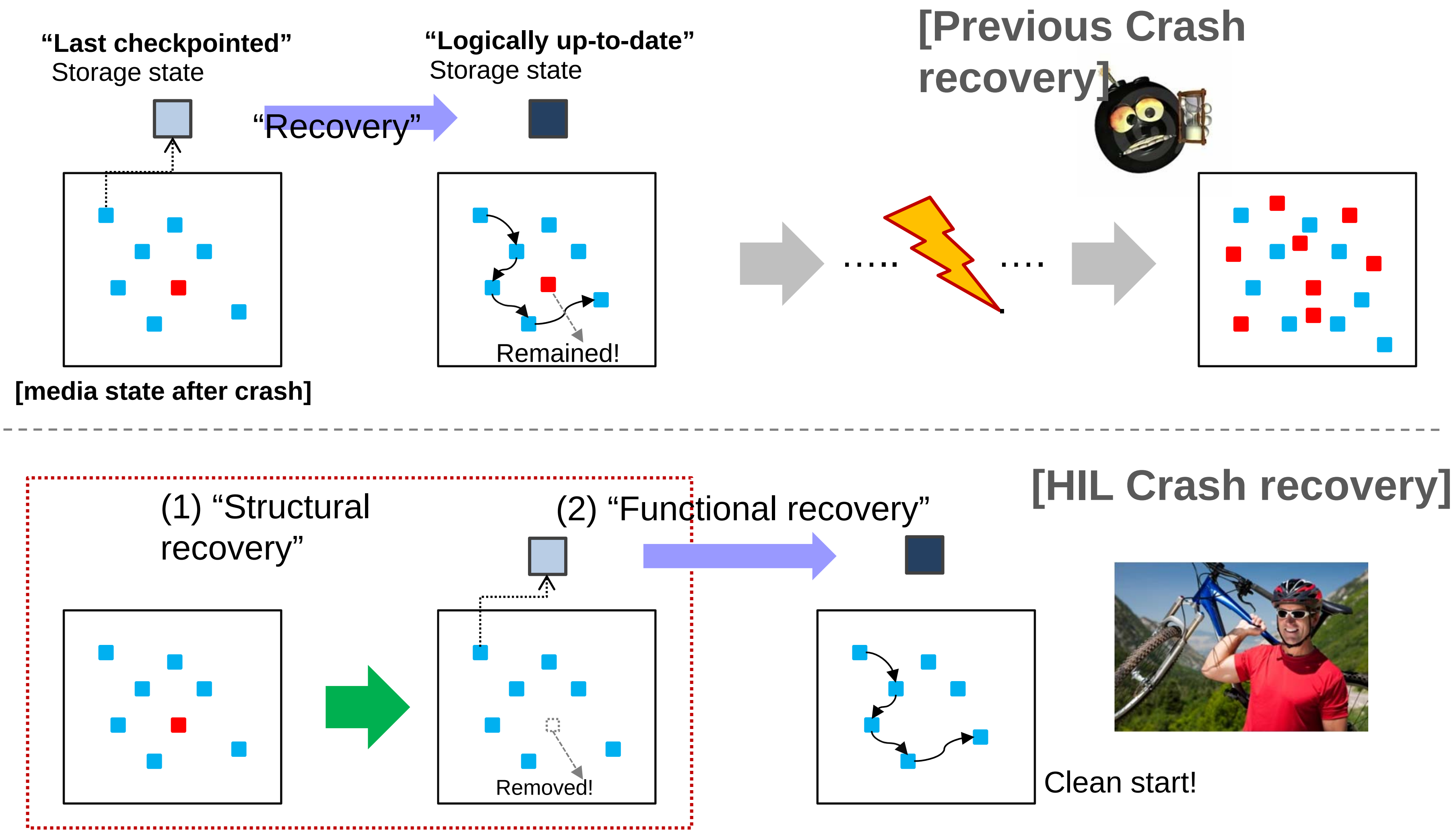
W : Write Processing Unit

P : Profiler

A : Free block allocator

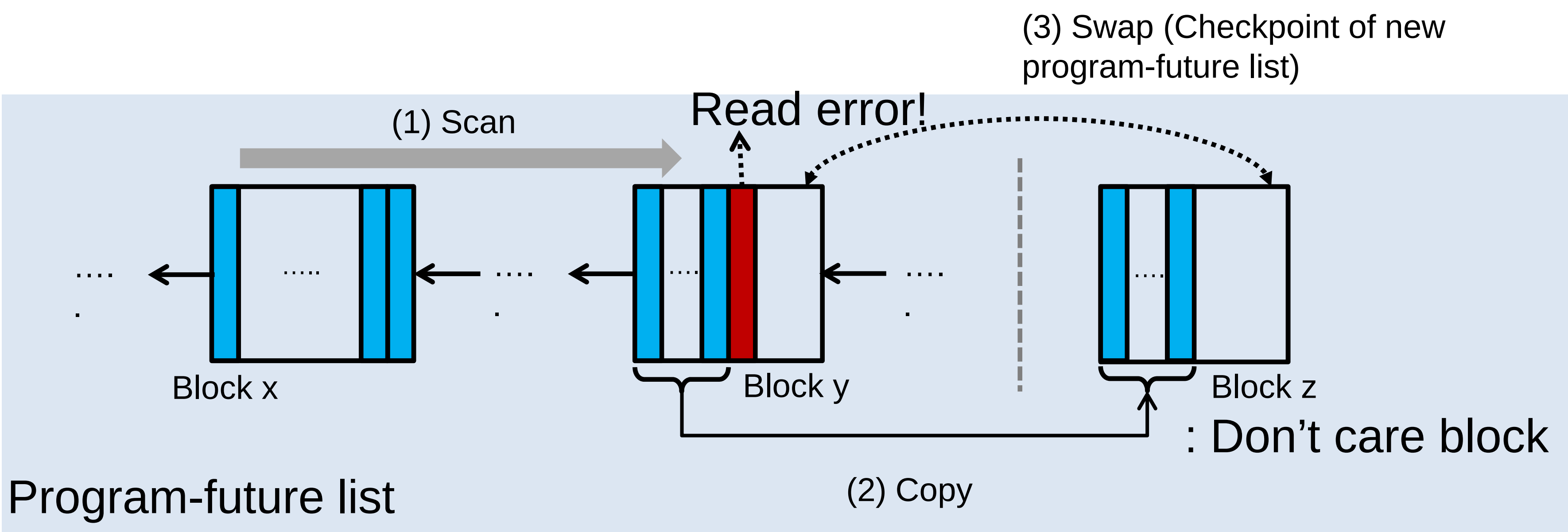
"Each Log has its own ***Write buffering scheme*** and ***built-in Crash recovery mechanism***"

HIL: Crash Recovery



HIL: Crash Recovery

- **Structural recovery of each Log level**



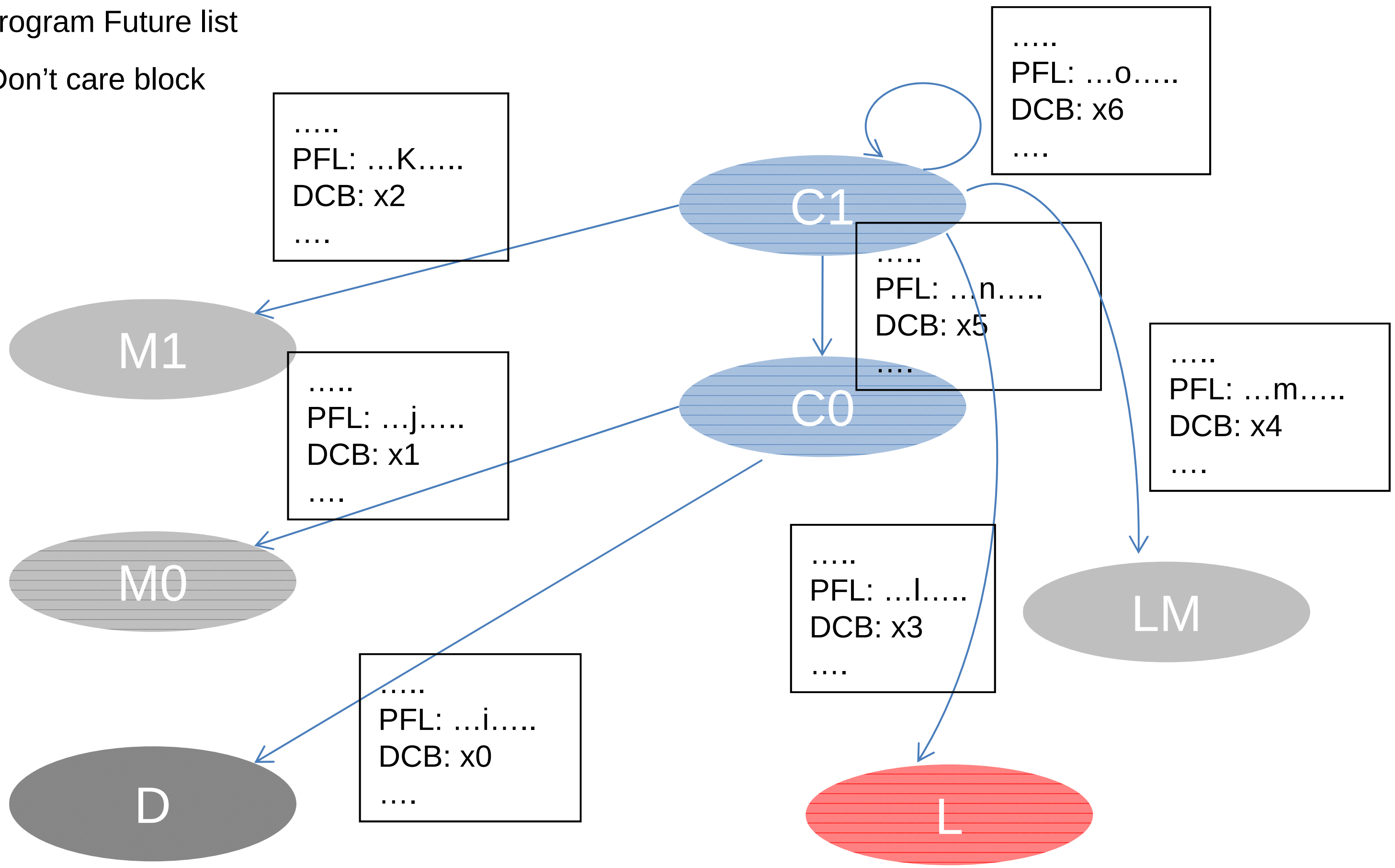
HIL: Crash Recovery

- **Structural recovery of storage device level**
  - Top down propagation of checkpoint info.
  - Local processing
    - Identifying crash frontier
    - Copying valid data and shadowing
  - Bottom up update of checkpoint info.
  - Atomic commit
  - Top down broadcasting of the completion of atomic commit

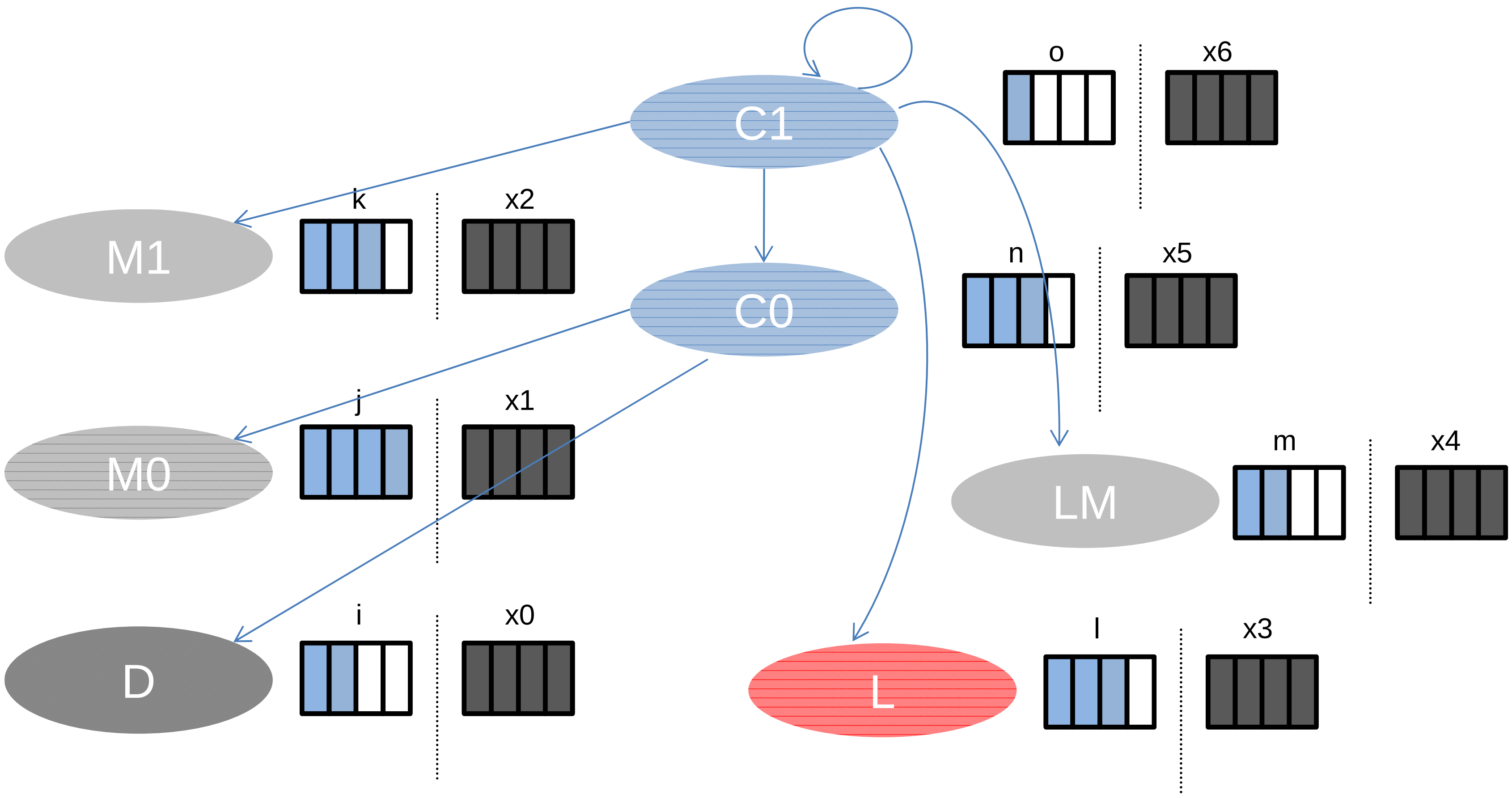
Top down propagation of checkpoint info

PFL: Program Future list

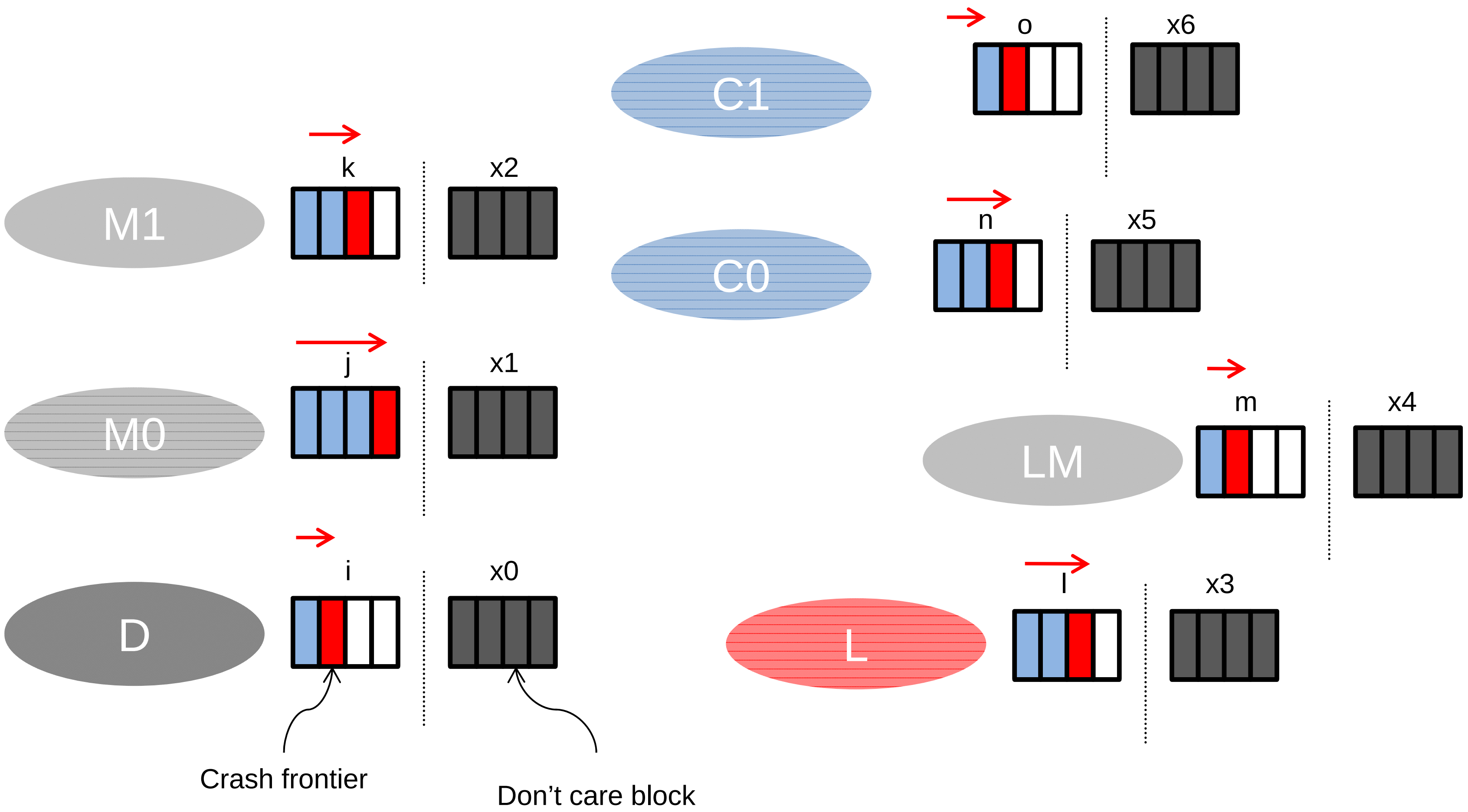
DCB: Don't care block



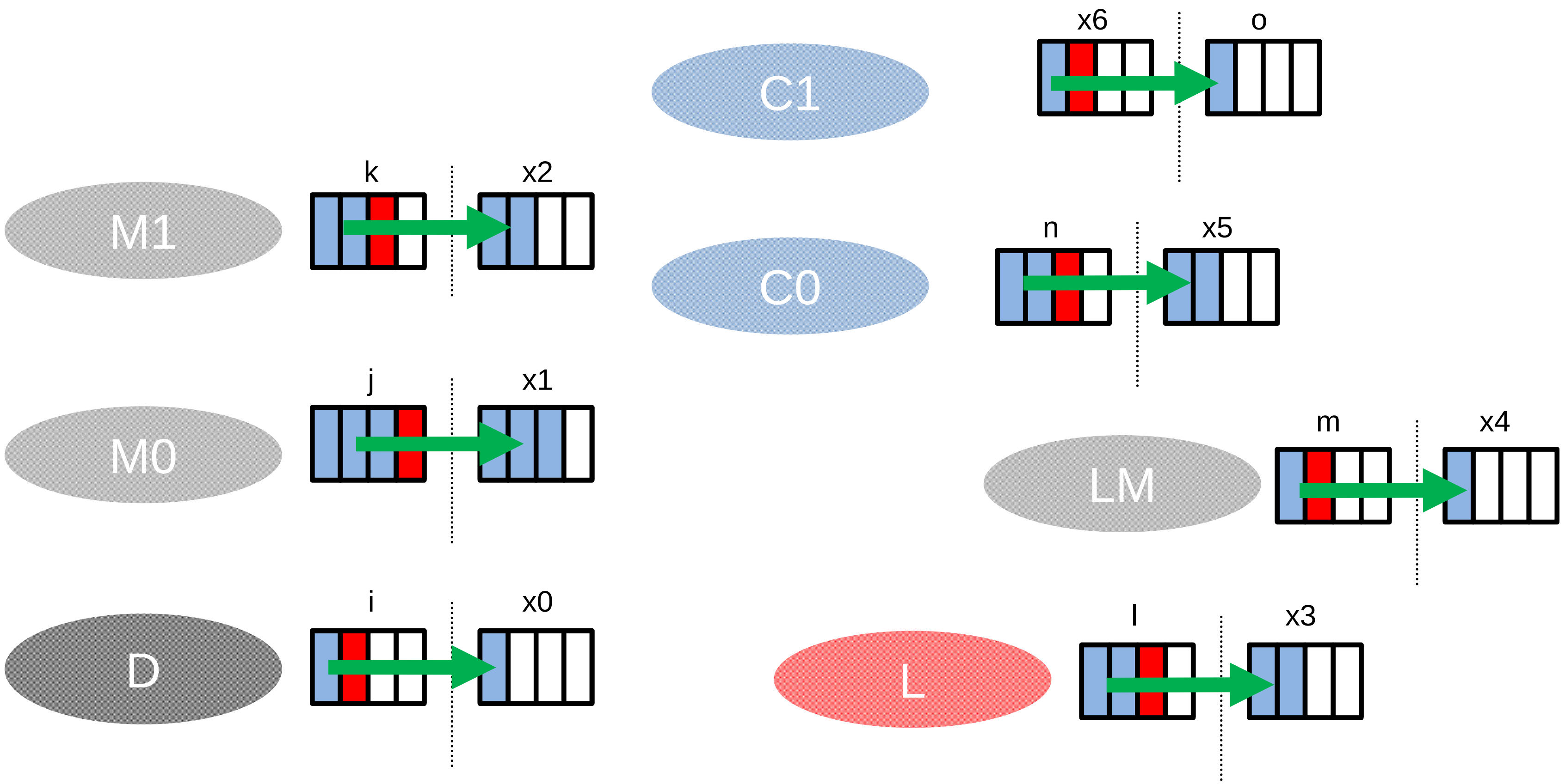
Top down propagation of checkpoint info



Local processing – Identifying crash frontier



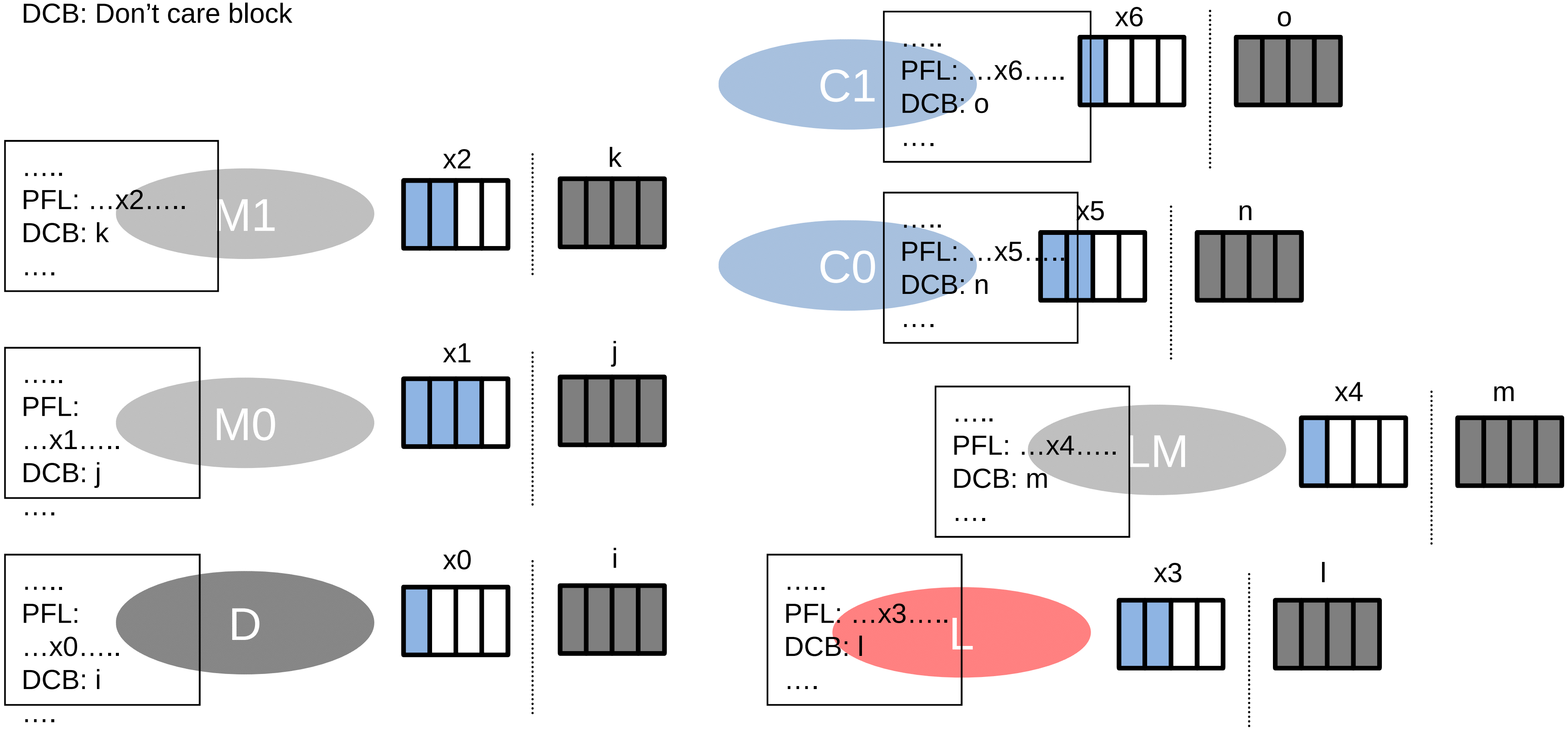
Local processing – Copying valid data and shadowing



Local processing – Copying valid data and shadowing

PFL: Program Future list

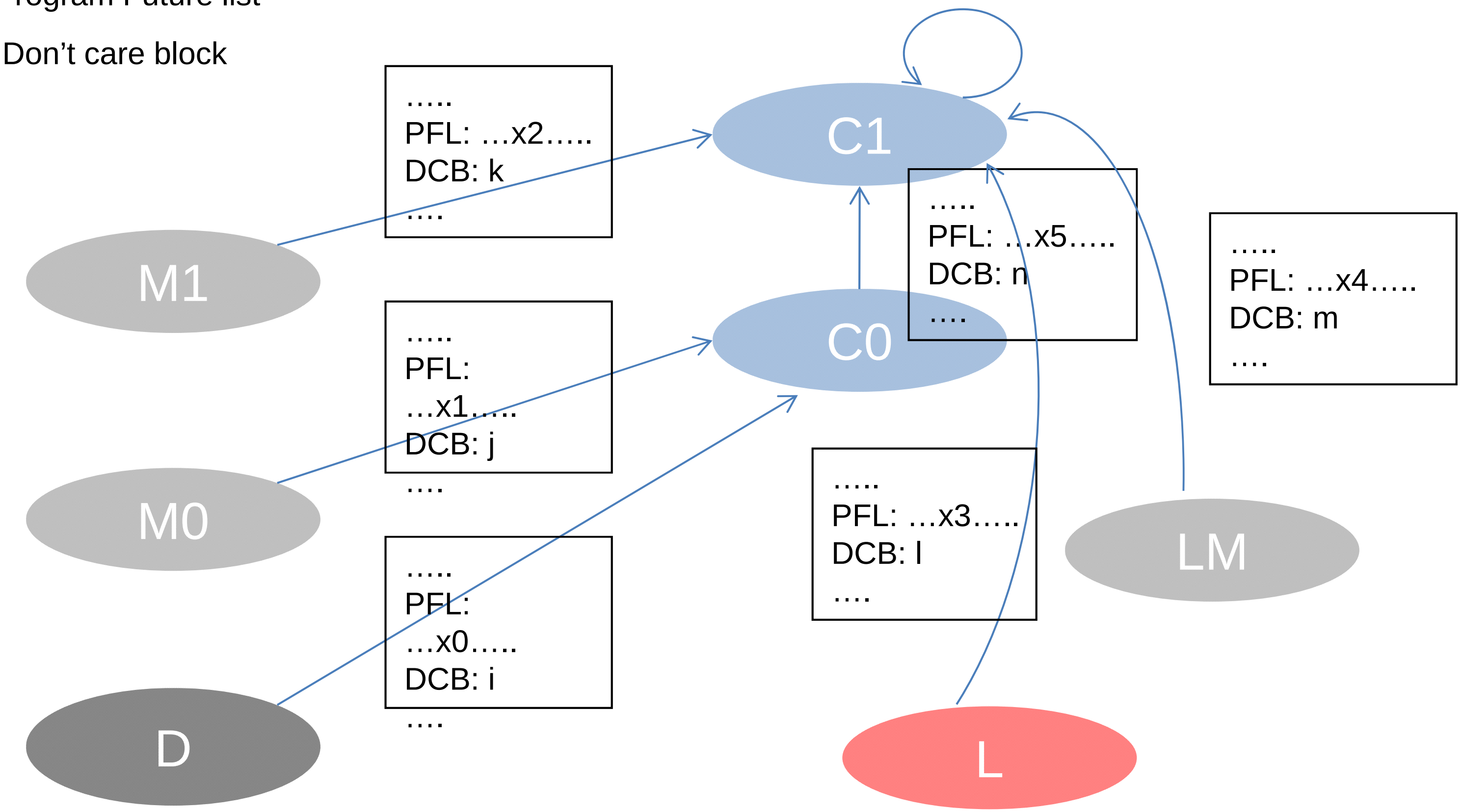
DCB: Don't care block



Bottom up update of checkpoint info

PFL: Program Future list

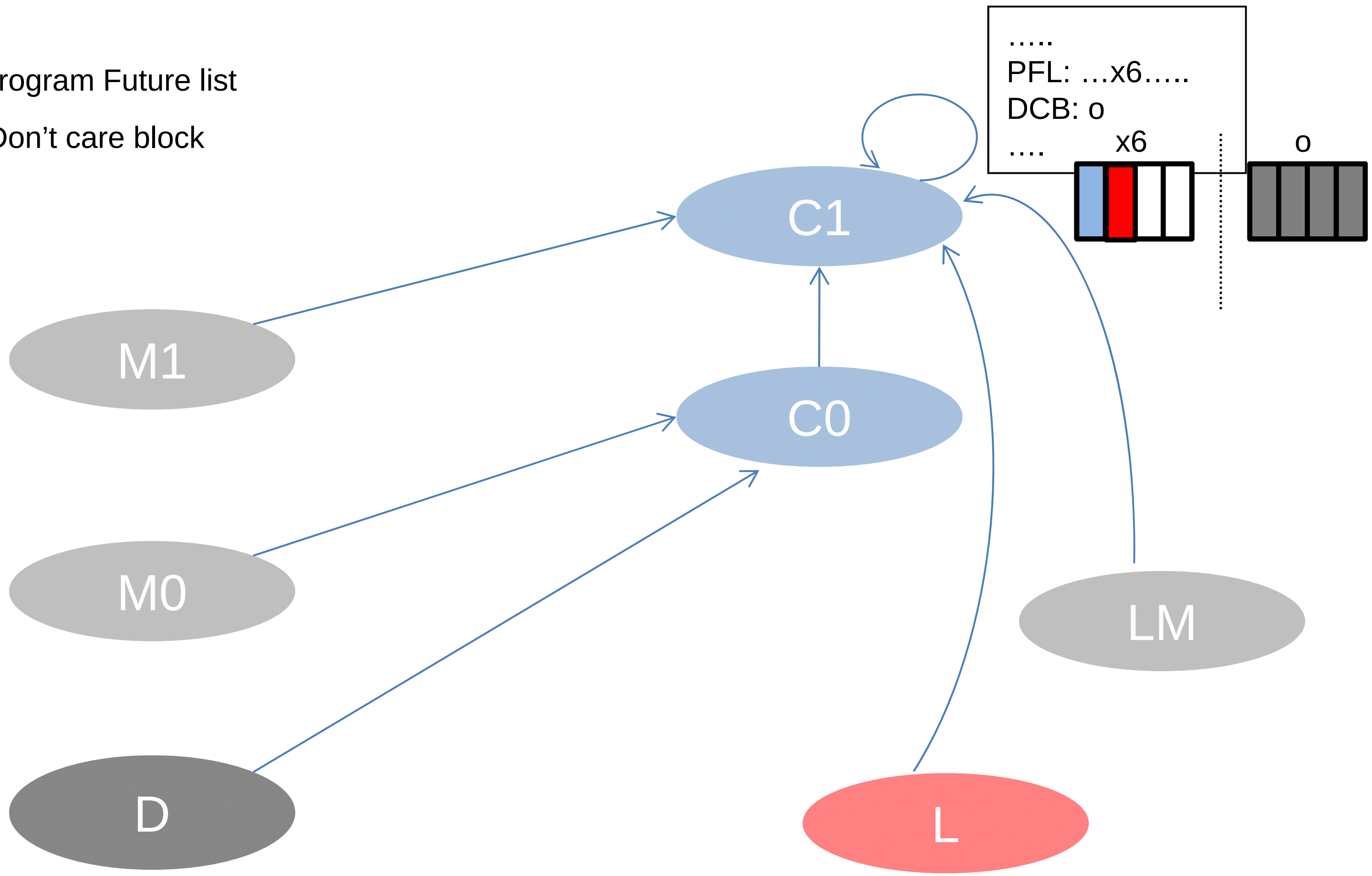
DCB: Don’t care block



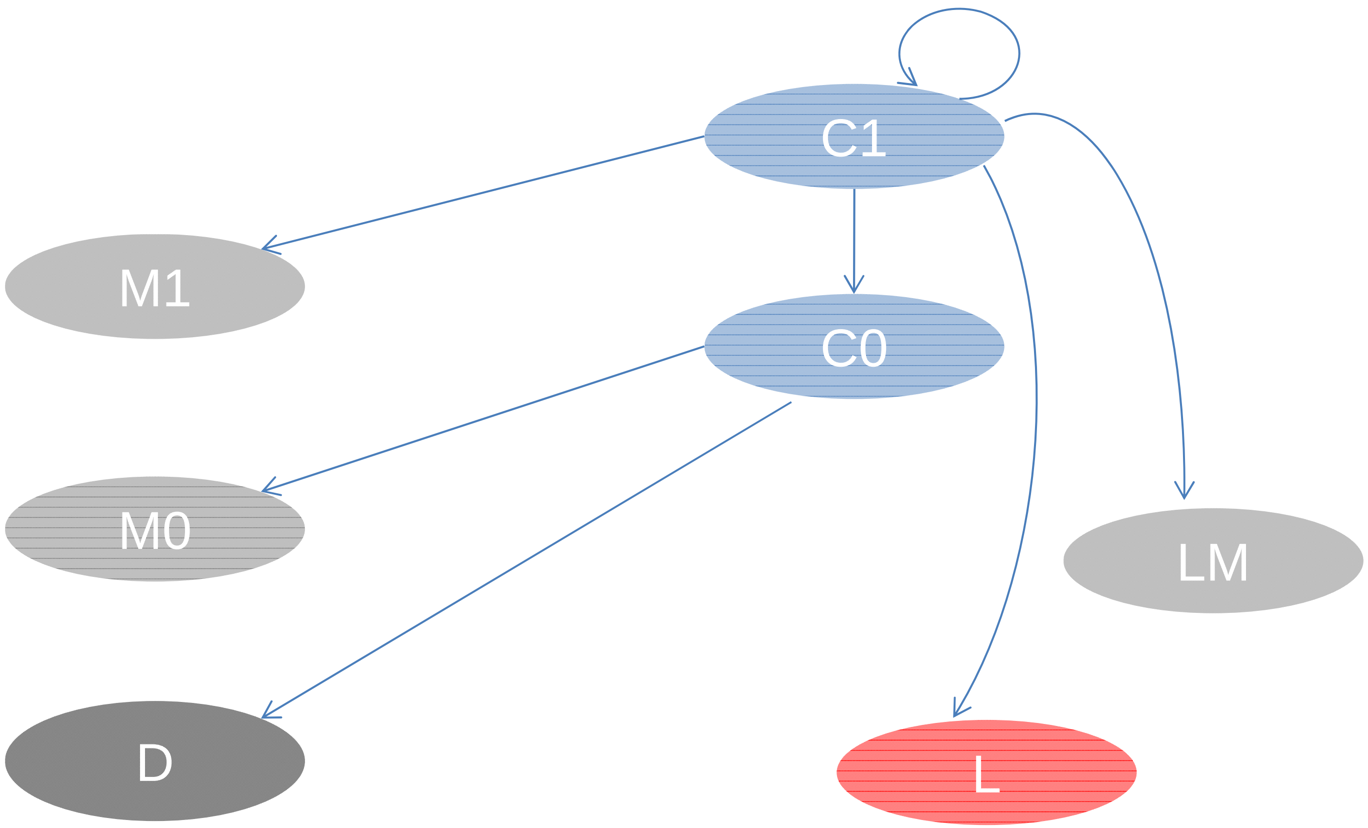
Atomic commit

PFL: Program Future list

DCB: Don't care block

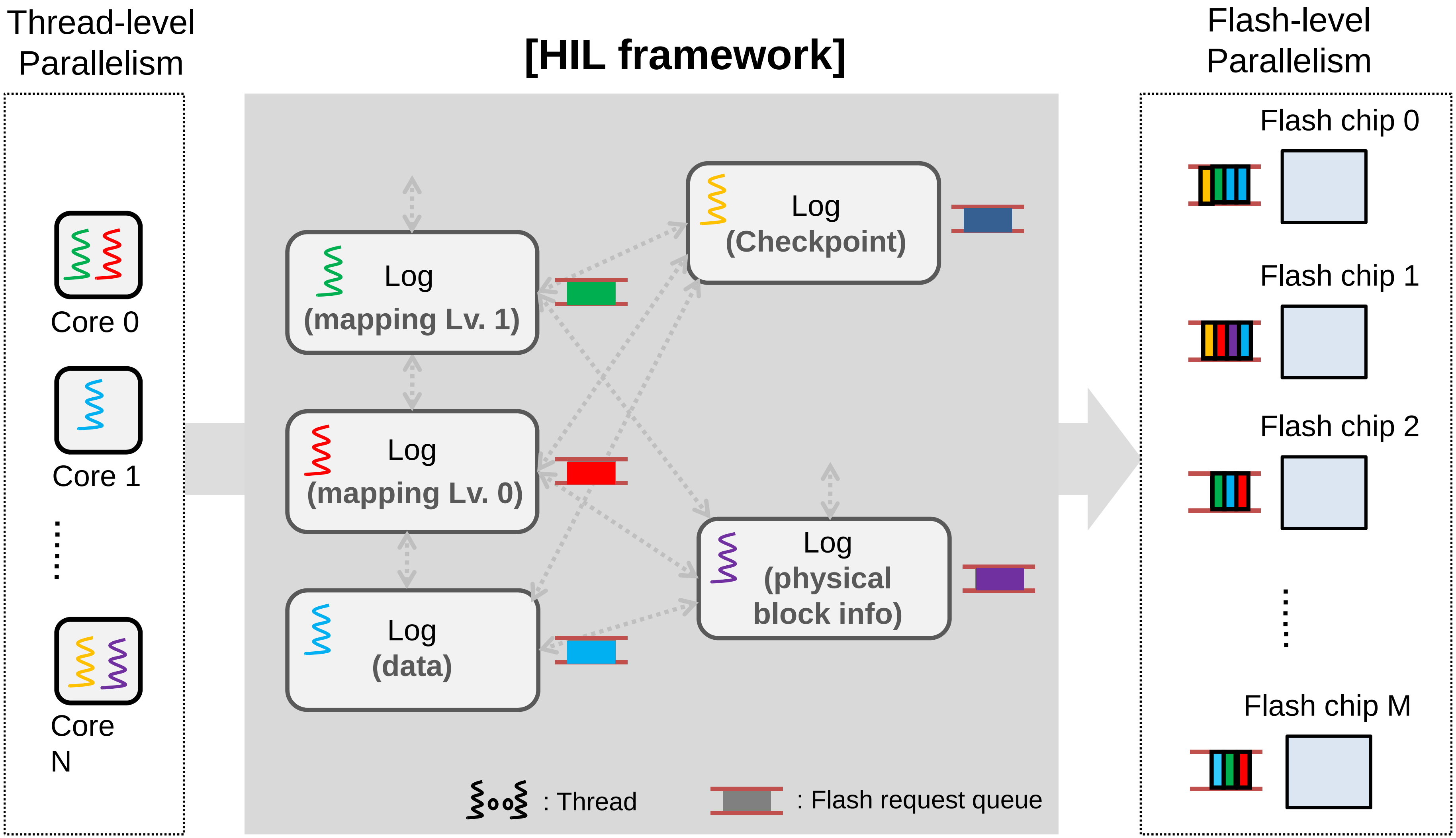


Top down broadcasting of the completion of the atomic commit



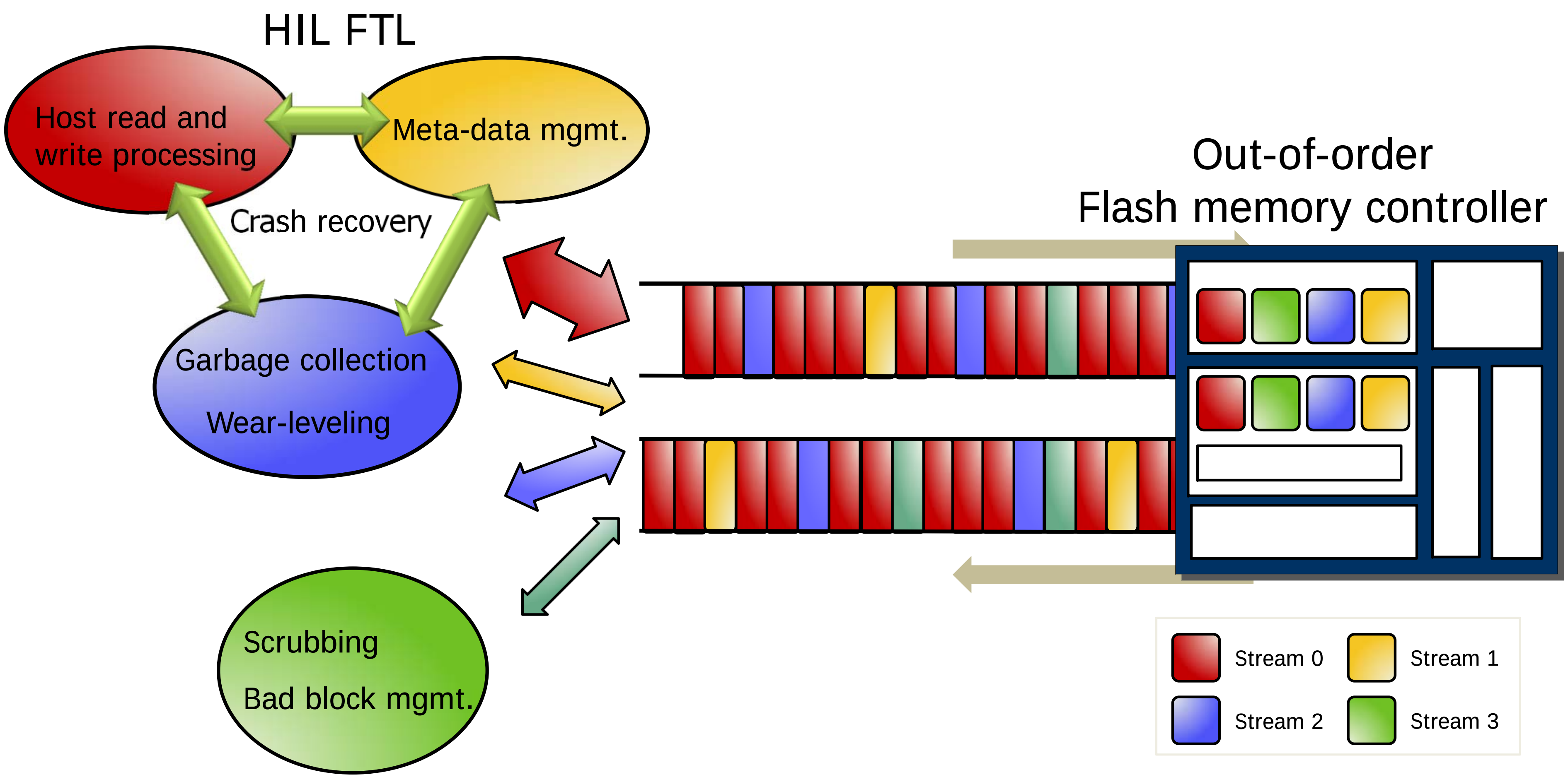
Ready to process functional recovery

HIL: Parallelism Exploitation



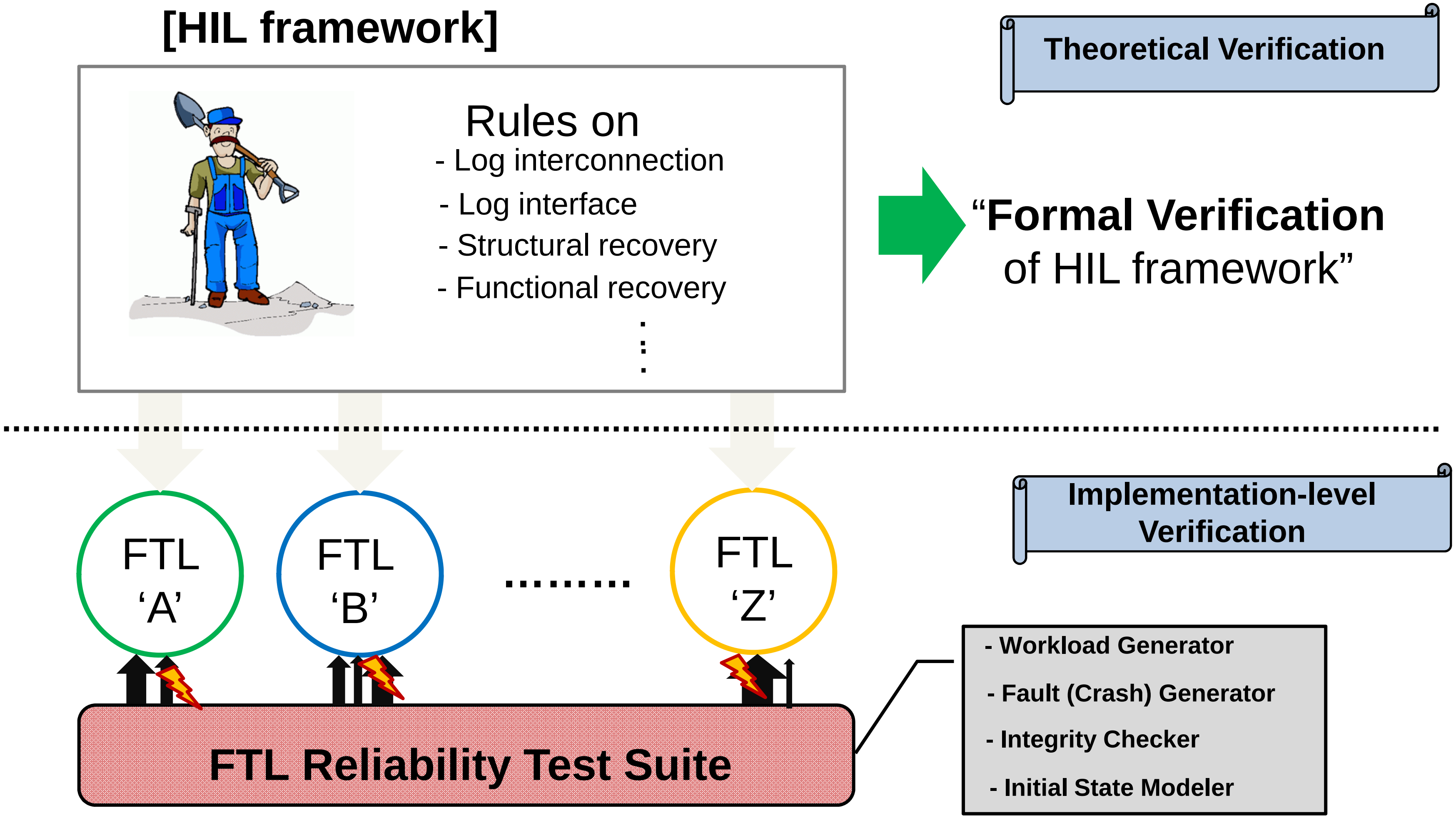
HIL: Parallelism Exploitation

- **Multiple streams of flash operations**
  - Seamless integration with out-of-order flash controller



•Nam, E.H., Kim, S.J., Eom, H., and Min, S.L., "Ozone (O3): An Out-of-order Flash Memory Controller Architecture", *IEEE Transactions on Computers*, vol. 60, no.5, pp. 653-666, Oct. 2011.

Correctness Verification



Formal Verification of HIL

[Defining Correctness Criteria]

=> Theorem to prove

A storage system is correct if read command for any logical page ***p*** is always responded with the data value ***v***, which is most recent data version of the logical page ***p***

[Theorem proving]

For $i = 0$,

- $nv\_link_0(p, v)$ became durable before the crash (**by rule 5**)
- $nv\_link_0(p, v)$ will be read correctly during structural recovery (by the definition of the durablity of nv_link)
- If $nv\_link_0(p, v)$ is in the crash frontier block, it will eventually be moved to don't care block even with repeated crashes (by the idempotence of structural recovery)
- For k=0, $nv\_link_k(p, v) \in flash\_log_k$

$\exists i\ (1 \leq i \leq n)$,

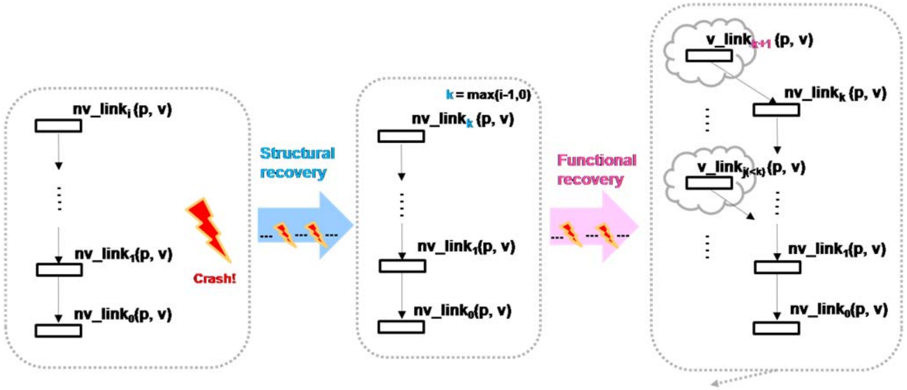
- $\forall k\ (0 \leq k \leq i-1)$, $nv\_link_k(p, v)$ became durable before the crash (**by rule 3**)
- $\forall k\ (0 \leq k \leq i-1)$, $nv\_link_k(p, v)$ will be read correctly during structural recovery (by the definition of the durability of nv_link)
- $\forall k\ (0 \leq k \leq i-1)$, If $nv\_link_k(p, v)$ is in the crash frontier block, it will eventually be moved to don't care block even with repeated crashes (by the idempotence of structural recovery)
- $\forall k\ (0 \leq k \leq i-1)$, $nv\_link_k(p, v) \in flash\_log_k$

Therefore, $\forall k\ (0 \leq k \leq \max(i-1,0))$ $nv\_link_k(p, v) \in flash\_log_k$

[Formal description of HIL framework]

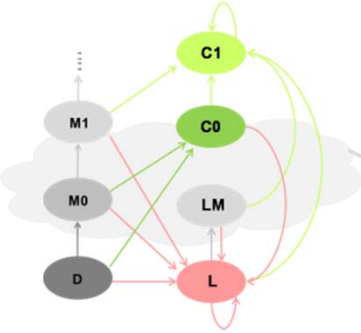
- Rule 1
  - $\exists i\ (0 \leq i \leq n)$, $v\_link_i(p, v)$ is removed from the $cache_i$ only after (1) $v\_link_{i+1}(p, v)$ is installed in the $cache_{i+1}$ (2) or when it is replaced by $v\_link_i(p, v')$ where v' is more recent data version of the logical page p, (3) or when a crash occurred
    - $v\_link_{n+1}(p, v)$ are not removed from $cache_{n+1}$ by the condition (1)
- Rule 2
  - $\exists i\ (0 \leq i \leq n-1)$, $nv\_link_i(p, v)$ is removed from redo_set of log i ($redo\_set_i$) only after $nv\_link_{i+1}(p, v)$ becomes durable in the log i+1
    - $Redo\_set_n \equiv NV\_set_n$,which means that Log n is redone from the start of log during recovery

⋮



•Y.J. Sung, "Formal verification of a compositional FTL design framework", Ph.D. Dissertation, 2013, SNU.

Implementation Verification



HIL : Page-mapped FTL prototype implementation

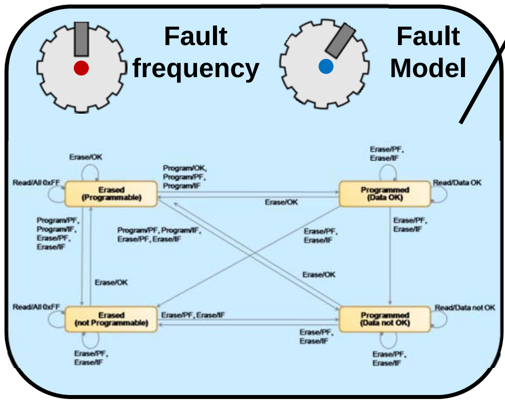
.....

FTL Reliability Test Suite

Verification under practical environment

- Virtex 5 FPGA
- 128MB SDRAM
- 8 Channel Flash modules
- Ethernet
- UART

In-house prototype platform



Flash memory simulator

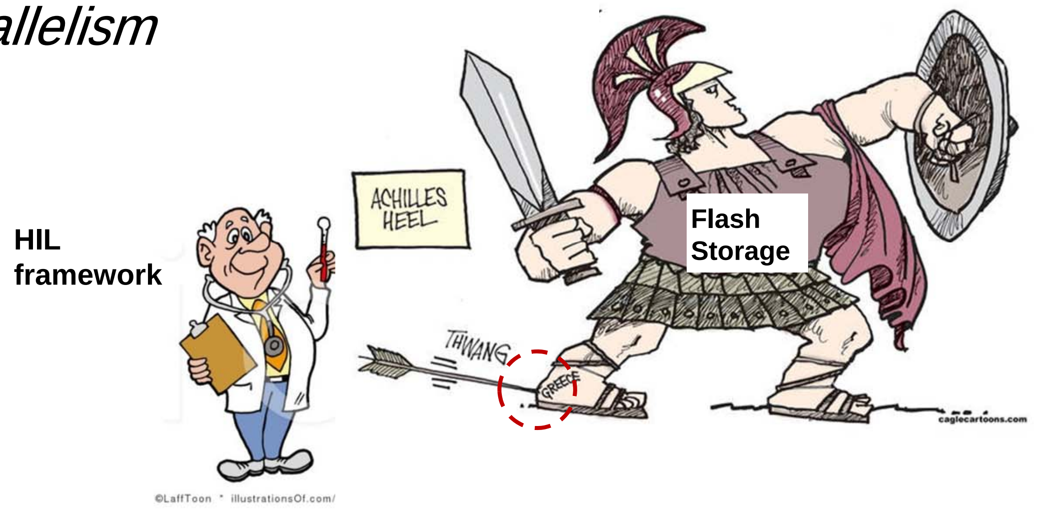
More Fault-intensive & various Test scenarios

Conclusions

- **Thesis statement**

  "HIL framework heals the Achilles' heel of flash storage systems, which is characterized by following key aspects"

  - *Compositional construction of FTLs*
  - *Built-in Crash Recovery mechanism*
  - *Maximal exploitation of parallelism*



Thank you & Questions ?