# Week 12a:

# SSDs and File Systems

(NAND Flash diagrams from https://blog.silicon-power.com/index.php/guides/nand-flash-memory-technology-basics/)

CSC469

Fall 2018

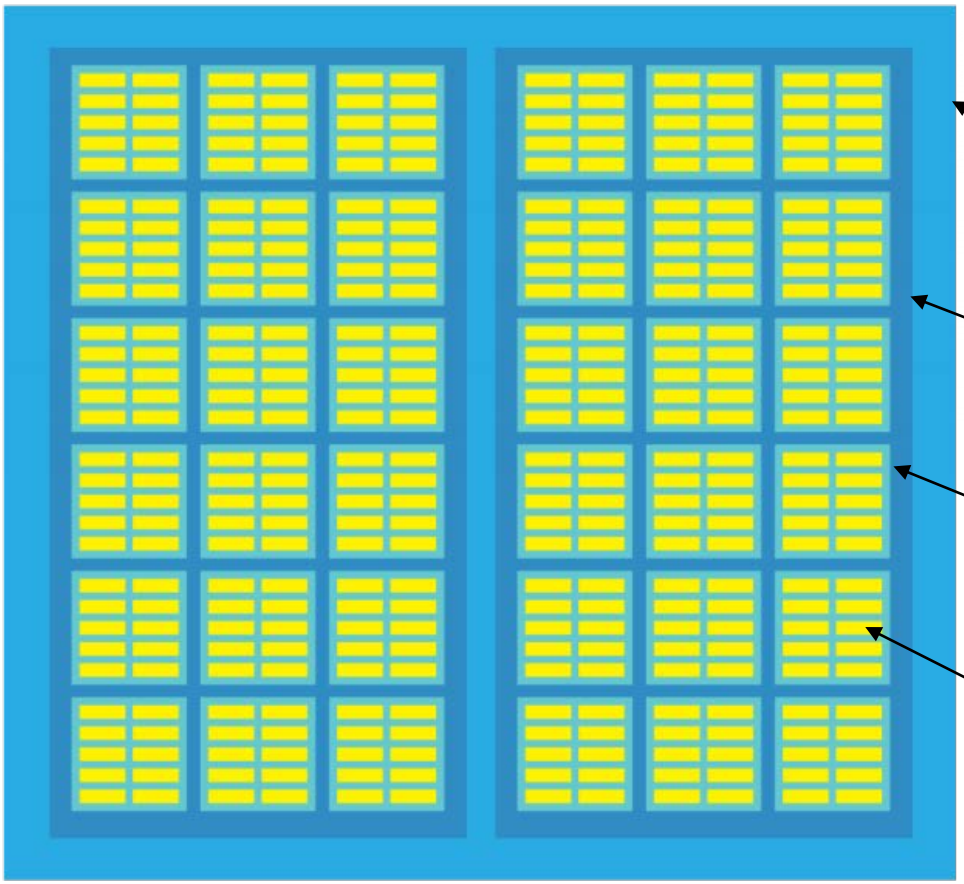
# Designing File Systems for New Storage


HDD vs. SSD

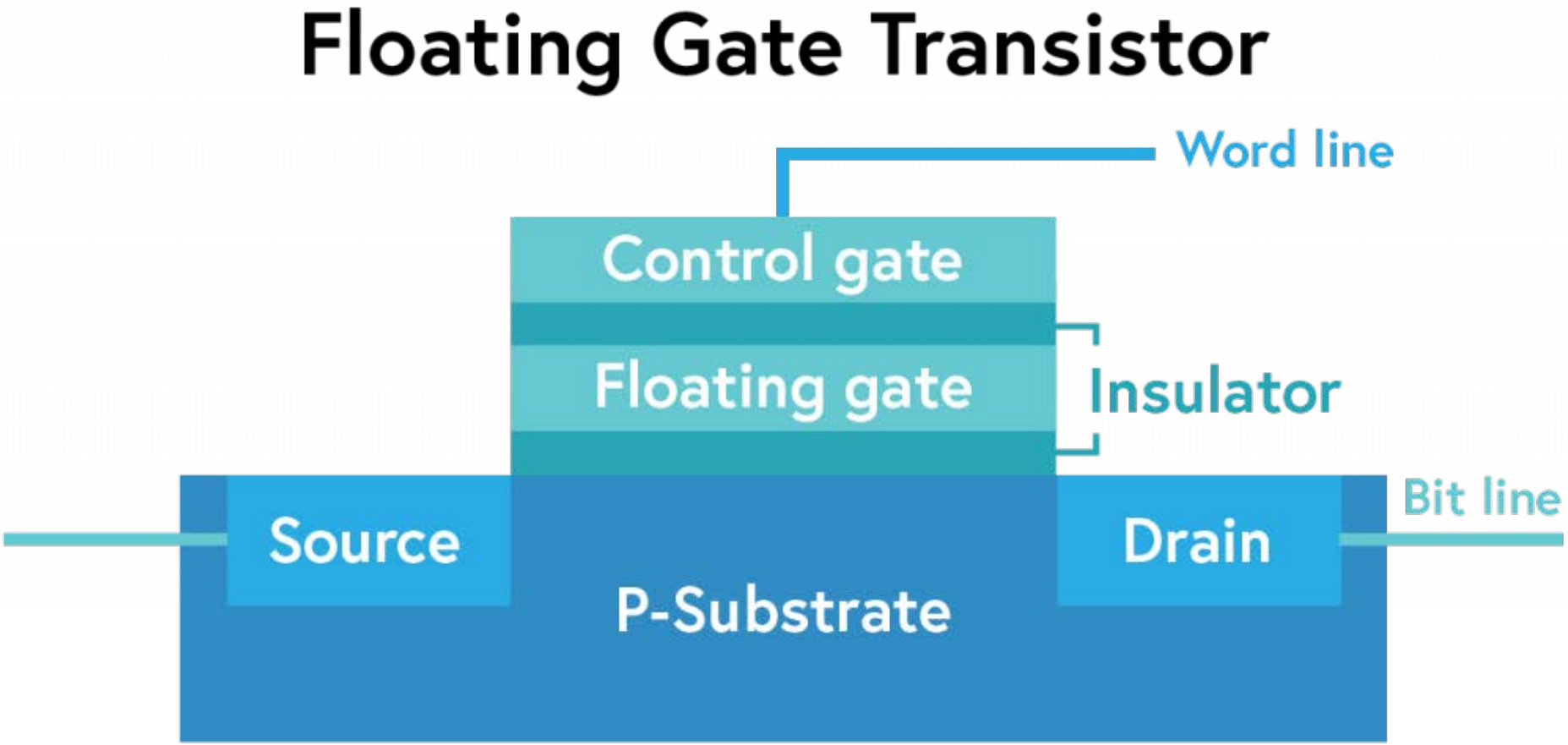

# Solid-state Storage Devices (SSDs)

- Unlike hard drives, SSDs have no mechanical parts
  - SSDs use transistors (just like DRAM), but data written to SSD is non-volatile (i.e., it persists after power loss)
  - NAND-based flash is the most popular technology, so we'll focus on it
- High-level takeaways
  1. SSDs have a higher $ cost per bit than hard drives, but better performance (no mechanical delays!)
  2. Writing to SSDs is more complicated than writing to HDD
     - This has implications for file system design

# Basics – Physical Organization

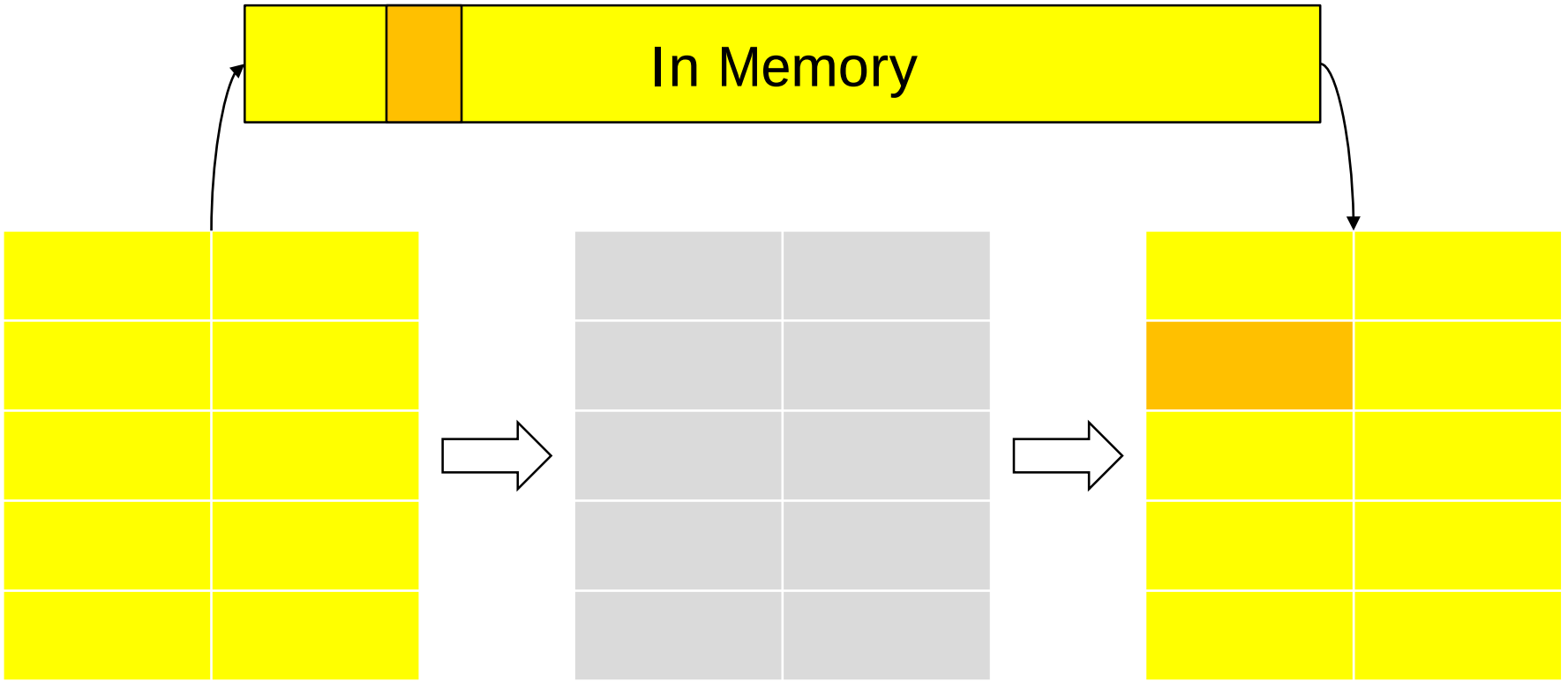


# At the cell level…



# Issue #1: Write unit != Erase unit

- After a write, a page must be erased before it can be re-written.
- But… Can only erase entire blocks at one time!



# Issue #2: Limited Endurance

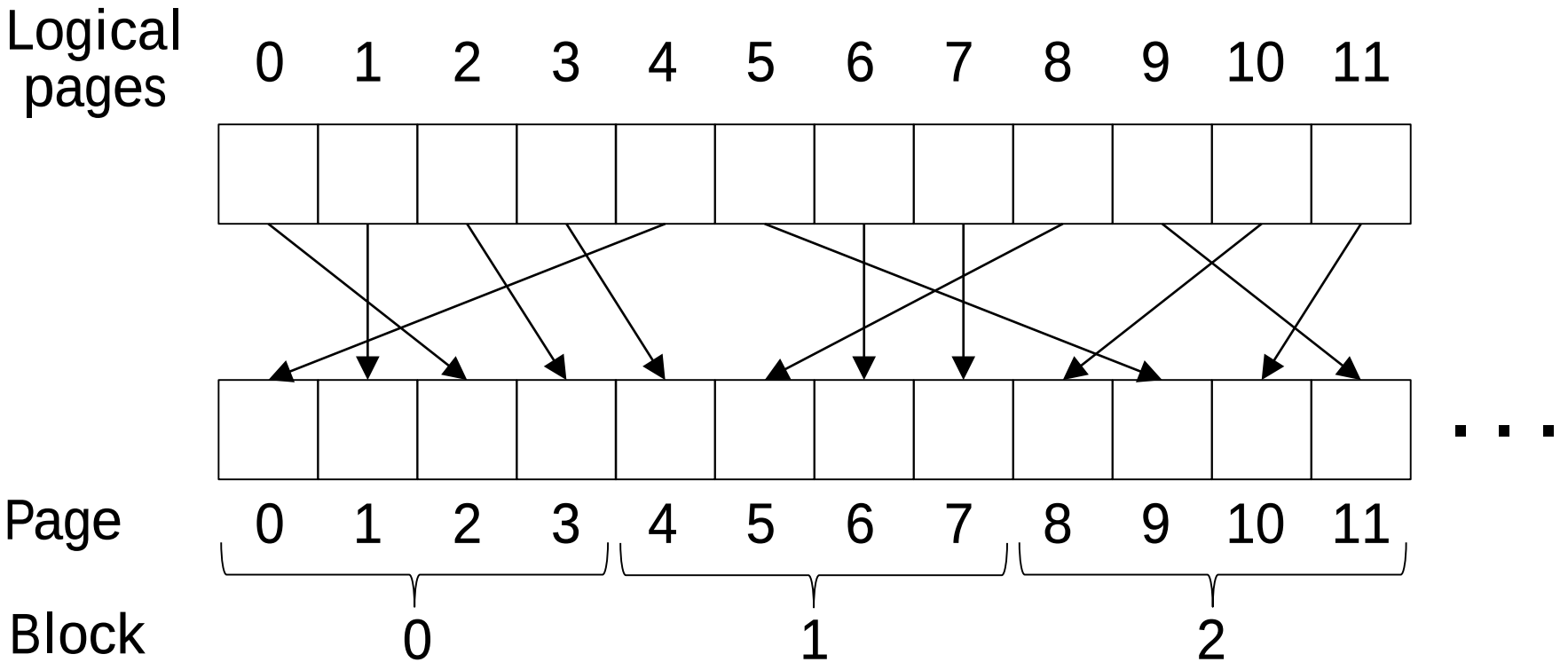
- Forcing electrons across the insulating layer is destructive
- After some number of program/erase (P/E) cycles, the block “wears out”. Charge leaks.
  - 1 bit per cell (SLC): 50-100K P/E cycles
  - 2 bits per cell (MLC): ~10-18K P/E cycles
  - 3 bits per cell (TLC): <5K P/E cycles
- Solution: Wear-leveling

# Flash Translation Layer (FTL)

- Goal 1: Translate reads/writes to logical blocks into reads/erases/programs on physical pages+blocks
  - Allows SSDs to export the simple “block interface” that hard disks have traditionally exported
  - Hides write-induced copying and garbage collection from applications
- Goal 2: Reduce write amplification (i.e., the amount of extra copying needed to deal with block-level erases)
- Goal 3: Implement wear leveling (i.e., distribute writes equally to all blocks, to avoid fast failures of a “hot” block)
- FTL is typically implemented in hardware in the SSD, but is implemented in software for some SSDs

# FTL Approach: Log-based mapping

- Basic idea: Treat the physical blocks like a log
  - Send data in each page-to-write to the end of the log
  - Maintain a mapping between logical pages and the corresponding physical pages in the SSD




# Garbage Collection (GC)

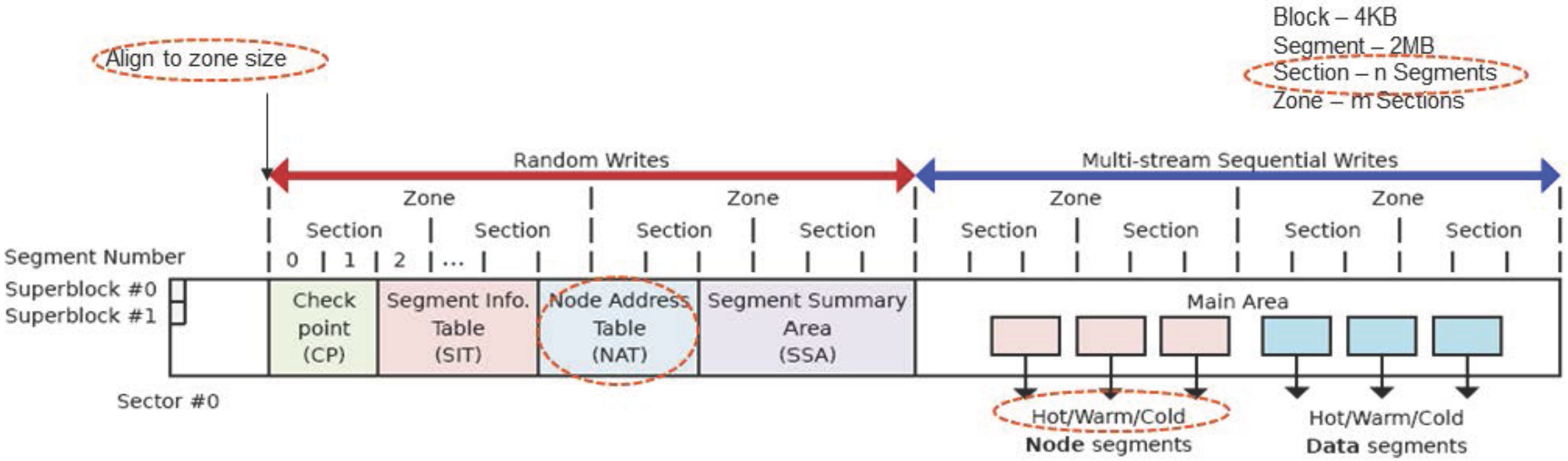
- Requires extra read and write traffic
- Overprovisioning makes GC less painful
  - FTL exposes a logical page space that is smaller than the physical page space
  - By keeping extra, “hidden” pages around, the FTL tries to defer GC to a background task (thus removing GC from critical path of a write)
- FTL will also occasionally shuffle live (i.e., non-garbage) blocks that never get overwritten
  - Enforces wear leveling


# F2FS: Flash-friendly File System

- Flash-friendly on-disk layout
- Cost-effective index structure
- Multi-head logging
  - Separates data and metadata into segments with similar usage characteristics (hot/cold)
  - Takes advantage of parallel SSD operations
- Adaptive logging
  - Reduces cleaning overhead

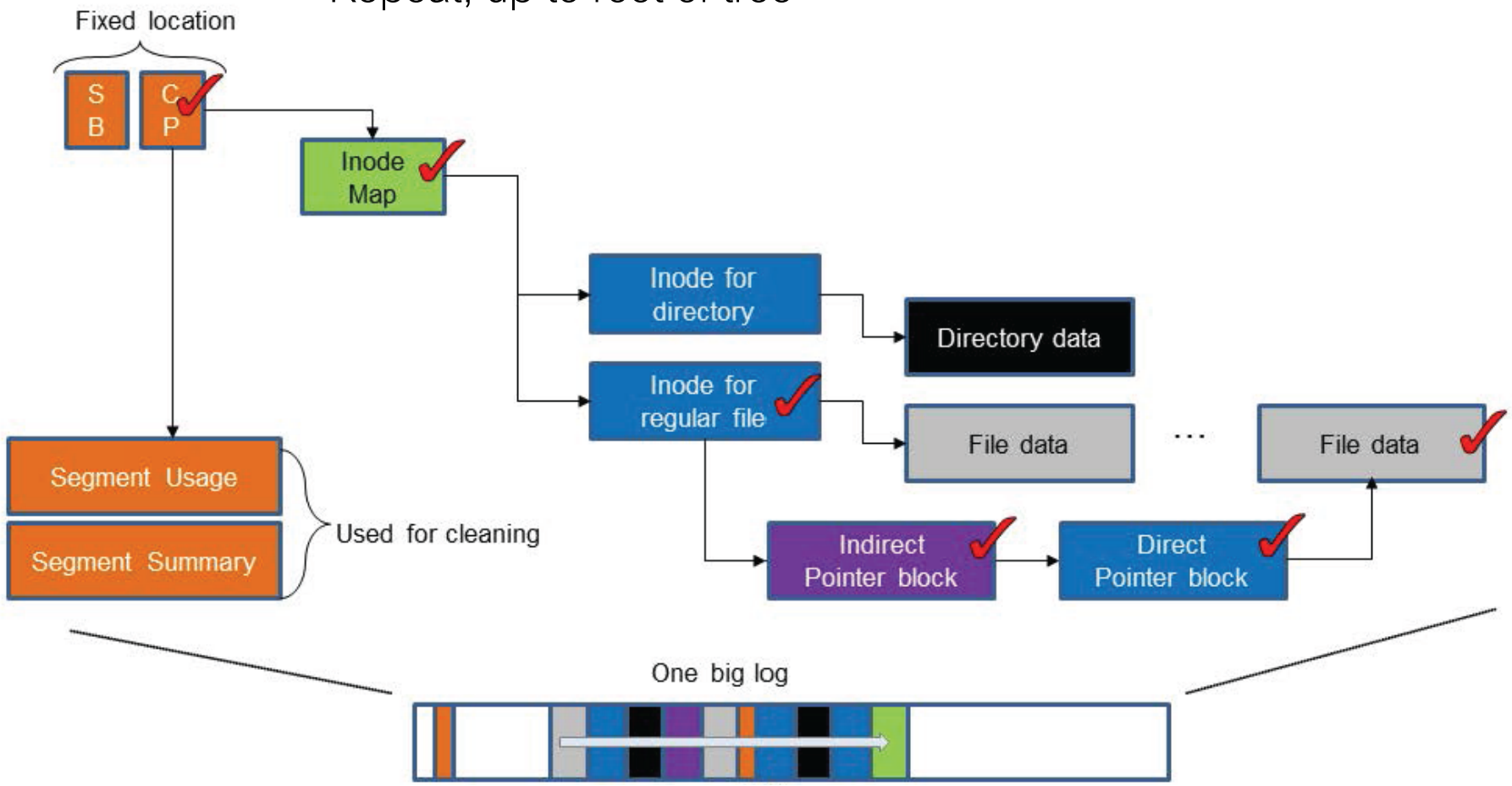
# Flash-friendly On-disk Layout

- Log-structured, but with some update-in-place regions
  - Takes advantage of underlying FTL on SSD
- Section size is power-of-two segments
  - By default, 1 section per zone
- Zones correspond to sets in FTL set-associative mapping



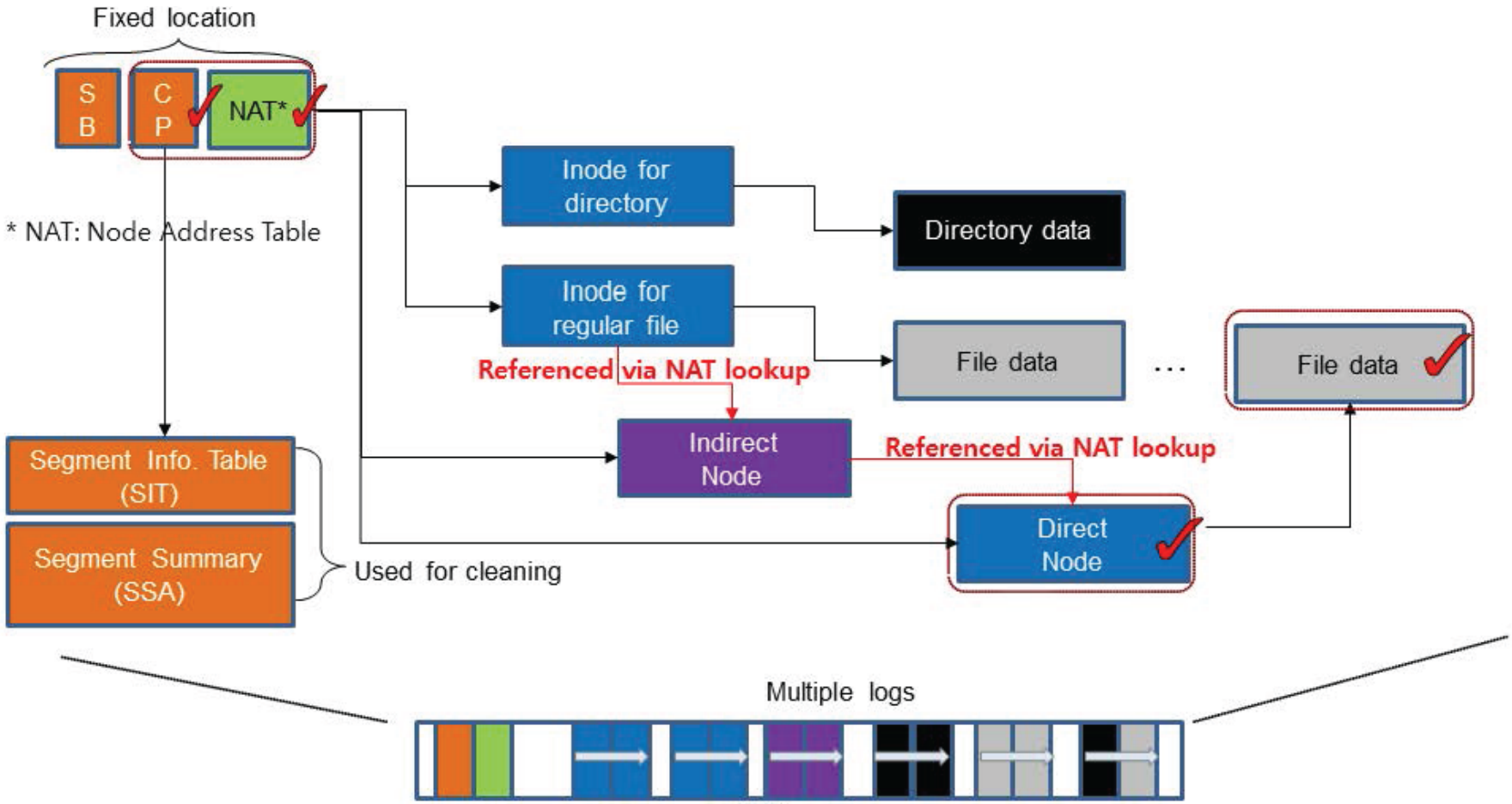
# Indexing: Wandering Tree Issue

- Each write to a block changes its location
- Requires update to block that points to it
  - Repeat, up to root of tree



# F2FS Indexing

- Node Address Table (NAT) stores location of block
  - Other index structures store only offset in NAT
  - Cuts off update propagation at one level



# Summary

- File system design considers storage characteristics
- SSDs are internally very different from HDDs
  - Well-suited for log-structured FS
- Both perform better with large sequential writes though
  - Random writes on SSDs make more work for FTL
    - Triggers garbage collection, high overheads
    - Increases *write amplification*, reduces life expectancy