Nameless Writes

Remzi H. Arpaci-Dusseau
Professor @ University of Wisconsin-Madison
(+visiting professor @ EPFL)

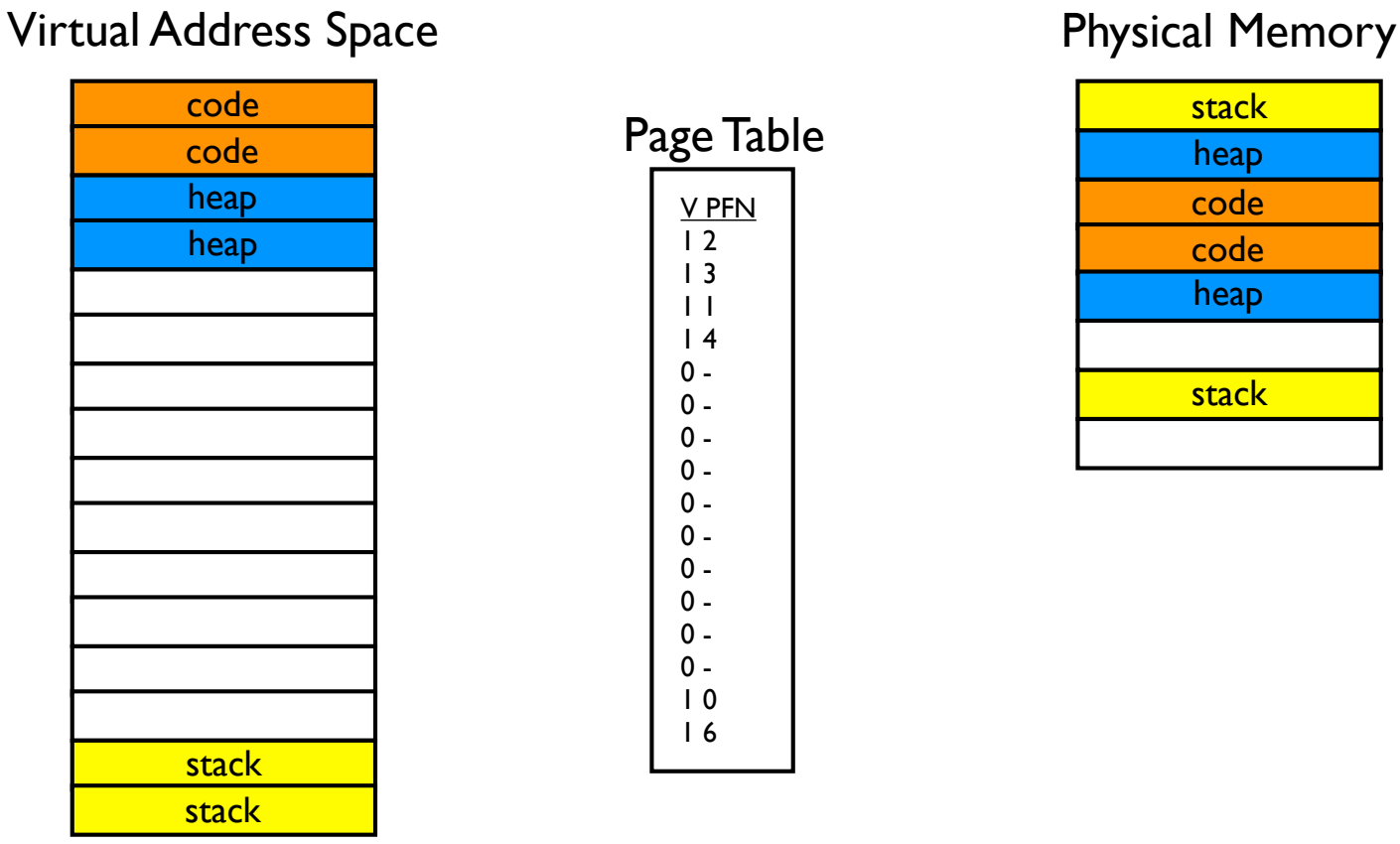
Joint work with:
Andrea C. Arpaci-Dusseau (UW, EPFL)
Vijayan Prabhakaran (MSR Silicon Valley)

Indirection

"All problems in computer science can be solved by another level of indirection"

- usually attributed to Butler Lampson

Example: Virtual Memory



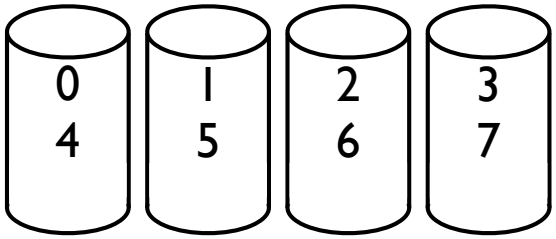
Problems?

- Too **big**, too **slow**

Another example: RAID

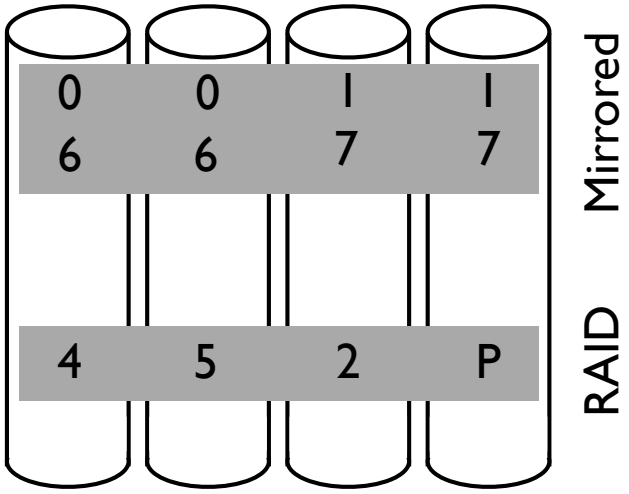
Early RAIDs: Simple indirection

- Fixed mapping avoids need for indirection table



More sophisticated RAID, more sophisticated mappings

- e.g., AutoRAID



Too Much of a Good Thing?

Virtual Machine Monitors

VMMs: Another layer, beneath OS

- Consolidation, multi-platform support, many other reasons

But the cost of indirection grows

Example: Virtual Memory (again)

- **Double Indirection:**
  Virtual to Physical to Machine

Many Examples

VMMs and Memory

File System and RAID

File System and Disk (a little)

File System and RAID and Disk

File System and Flash FTL

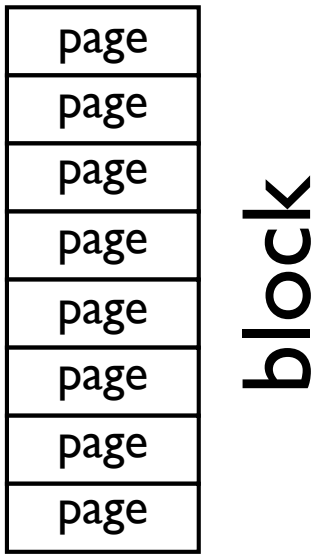
Today's Focus:
Flash

Flash FTL

Flash Translation Layer (FTL)

- Turns **read-erase/program** into **read-write**
- Allows for **wear leveling**

Background



Flash organized into **blocks**

Each block contains some **pages**

Problem:

- To program a page, must erase block first
- Even worse: Erase is costly (ms not us)

Implication: Simple mapping performs poorly

- Would turn each write into erase/program

Solution: Use Indirection

Solution: Borrow log-structuring ideas

- Organize flash into a **log**
- Erase an "active" block
- Direct all writes to active block
- Record mapping in **indirection table (i-table)**

Useful for Wear Too

Wear-leveling problem

- Too many erase-program cycles will render block unreadable (can't differentiate ones from zeroes)

Indirection helps here too

- Balance write load across blocks
- Might have to **migrate** blocks from live but not-often-used block for leveling

Problems

Cost of Indirection

Too big

- i-table (naive): one mapping per page
- i-table (hybrid): one per page for some, one per block for most
- Either way: **MB (or GB) of memory**, just for mapping information

Too slow

- Could be a problem too (if i-table doesn't fit in memory)

So What Can We Do?

Key Idea: Turn

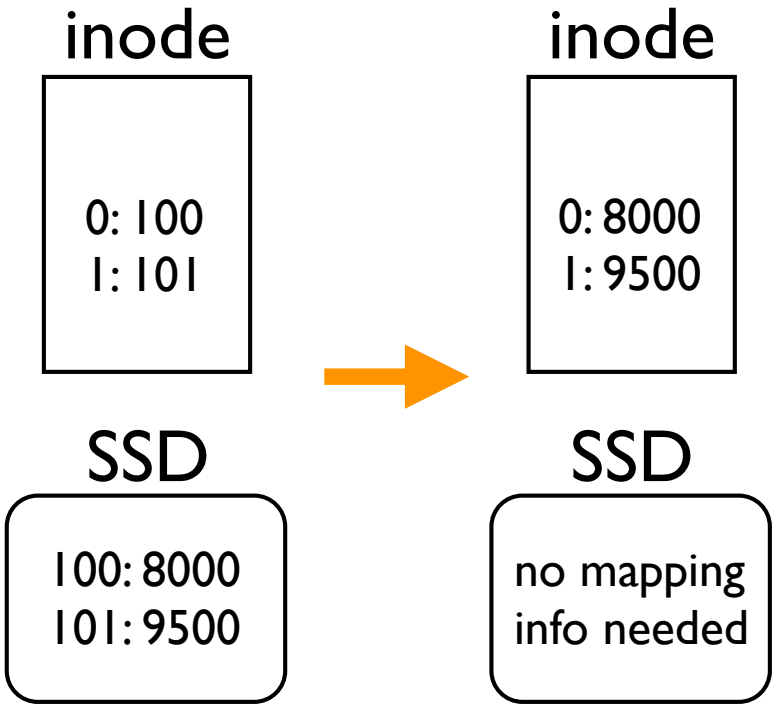
Double Indirection

To Our Advantage

Leverage: Double Indirection

Double indirection example

- FS: virtual offset (in file) to logical block (on dev)
- Flash: logical block to physical page



Can we remove one level of the indirection?

- Generically called **de-indirection**

Our "Solution": Nameless Writes

Nameless Writes

Usual interface:

- write(address, data): return OK/FAIL

Nameless interface

- write(data): return address, OK/FAIL

Device chooses where to write block,

and returns **physical address** to client (FS)

Simple Example

Structures dirtied: inode (I), data (D)

Usual approach

- D is allocated to address A(D)
- I is at fixed location [A(D) inside]
- Write them out whenever (depending on FS)

Nameless approach

- Nameless write of D, returns A(D)
- Update inode I with A(D)

What About Wear?

Problem: Wear-leveling

- Wear-leveling algorithm still might need to move blocks

Solution: **Renaming callback**

- Device upcalls into client, informs that device has moved block at address X to new location: addressY
- Client (FS) must take action as needed

Key Features

Removes FTL indirection

- No more indirection table;
  assumed that client tracks locations

Device retains control

- For performance, still log-structured
- For reliability, still does wear leveling

But, Lots of Problems

File system must delay allocation decision

File system must be able to write out blocks in certain order

File system must be able to handle callback

Sometimes need a "known location"

Device must be willing to expose its physical nature

(many more; your thoughts/complaints go here)

Other Ways To Do This?

Could remove FTL ("file-system only")

- Buggy FS might do poor wear leveling
- Device is better at managing its detailed performance characteristics

Could do it in device ("device only")

- Hard to do while device is mounted

Could consider alternate interfaces

- e.g., inform device of pointers

Conclusions

Nameless Writes

Addresses overheads of FTL indirection

- Enables little or no mapping info
- Device controls low-level decisions

But, some pain points

- Integrating into existing/new file systems
- Will devices expose physical names?

General approach of **de-indirection**

- Likely more widely applicable

Indirection: Reprise

"All problems in computer science can be solved by another level of indirection"
- usually attributed to Butler Lampson

Lampson attributes it to David Wheeler

And Wheeler usually added:
"but that usually will create another problem"