

Integrated End-to-End Dependability in the Loris Storage Stack



David van Moolenbroek
Raja Appuswamy
Andy Tanenbaum

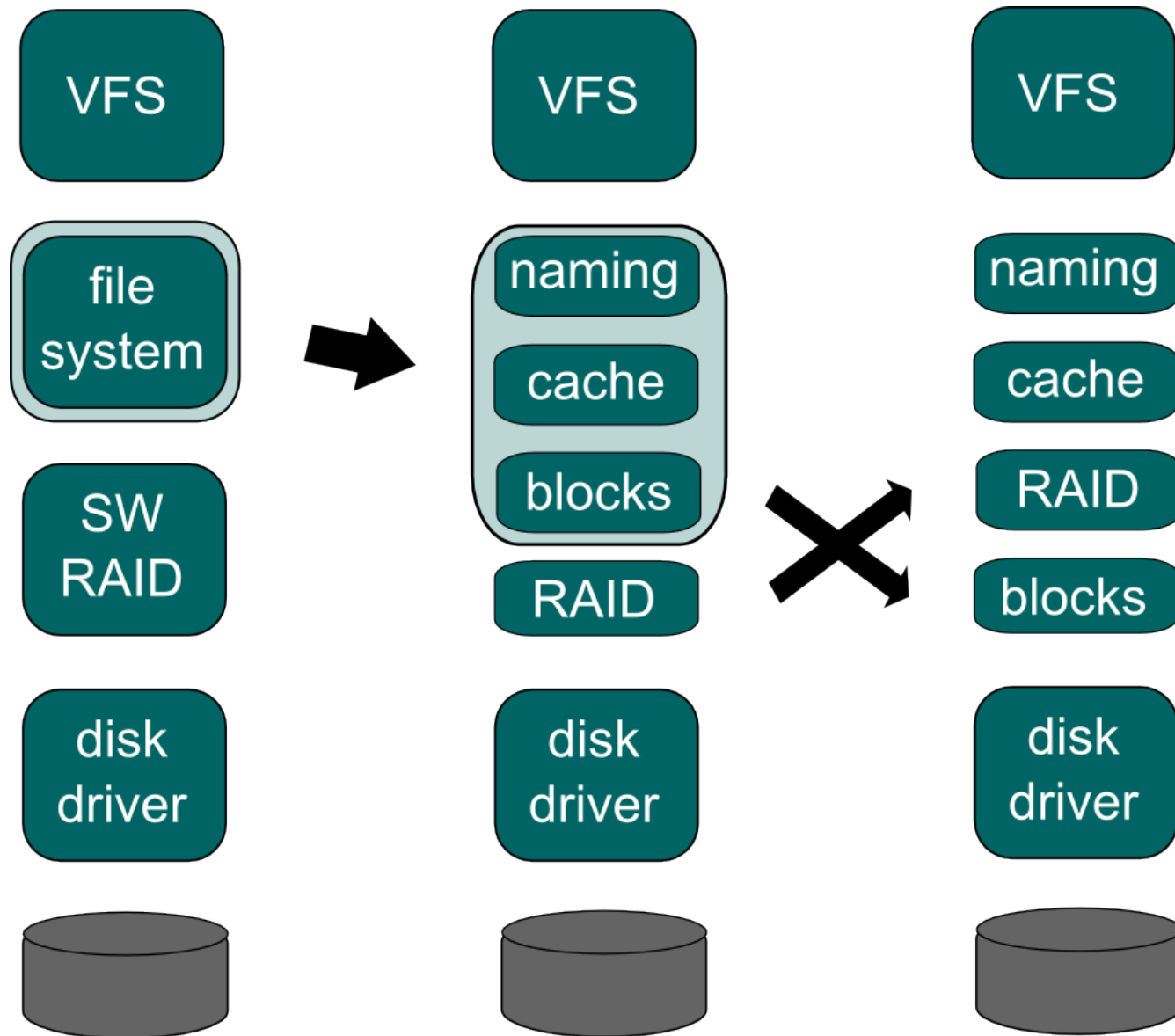


Outline

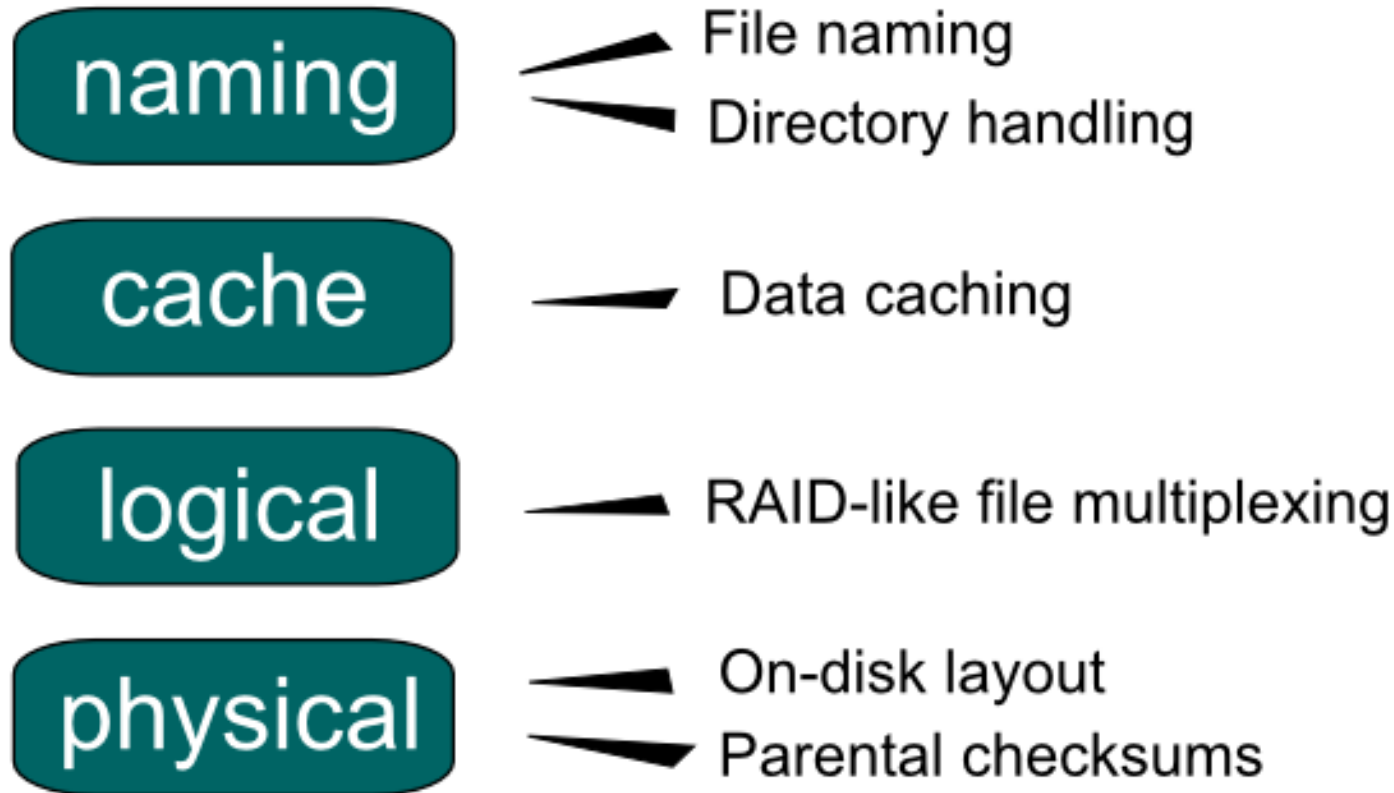
- **Context:** the Loris storage stack
- Dependability **problems**
- Our combination of **solutions**
- **Conclusion**



Context: the Loris storage stack (1)



Context: the Loris storage stack (2)



Context: the Loris storage stack (3)

- The new arrangement offers better reliability
 - Specifically: protection against **disk device failures**
 - Parental checksumming + file-level RAID
- Implemented on **MINIX3 microkernel OS**
 - Each module is a separate userspace process



Dependability problems

- [**x**] Disk device problems
- [] Whole-system failures
 - Recover to a globally consistent, recent state
- [] Software bugs
 - Modeled as black-box process crashes
 - Recover transparently, or otherwise limit the damage
- [] Memory corruption
 - Focus on large, important memory areas
 - Detect corruption; recover when possible



Our combination of solutions

- Simple restart
- Execution environments
- Checkpoints and logging
- Checksumming memory

App

VFS

Naming

Cache

Logical

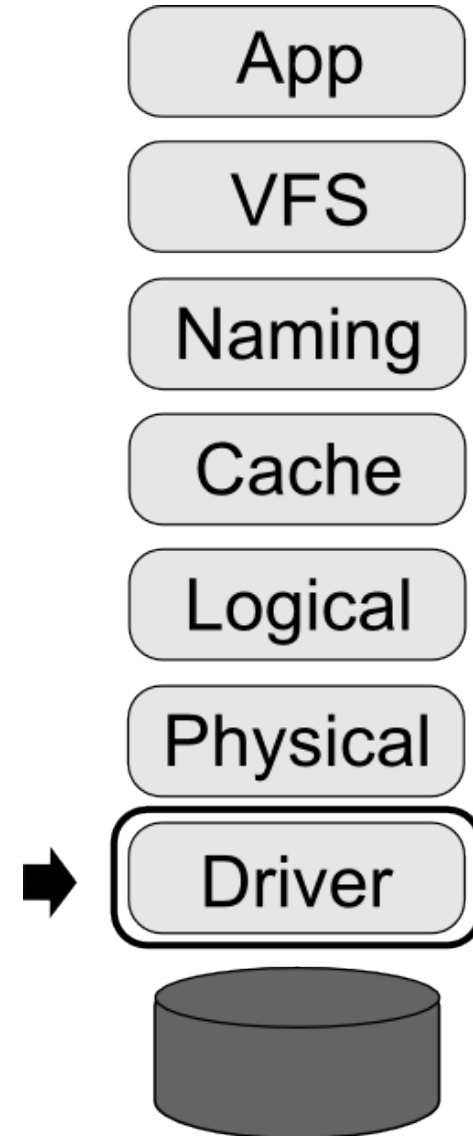
Physical

Driver



Lowest (driver) layer

- Software bugs only
- Simple restart



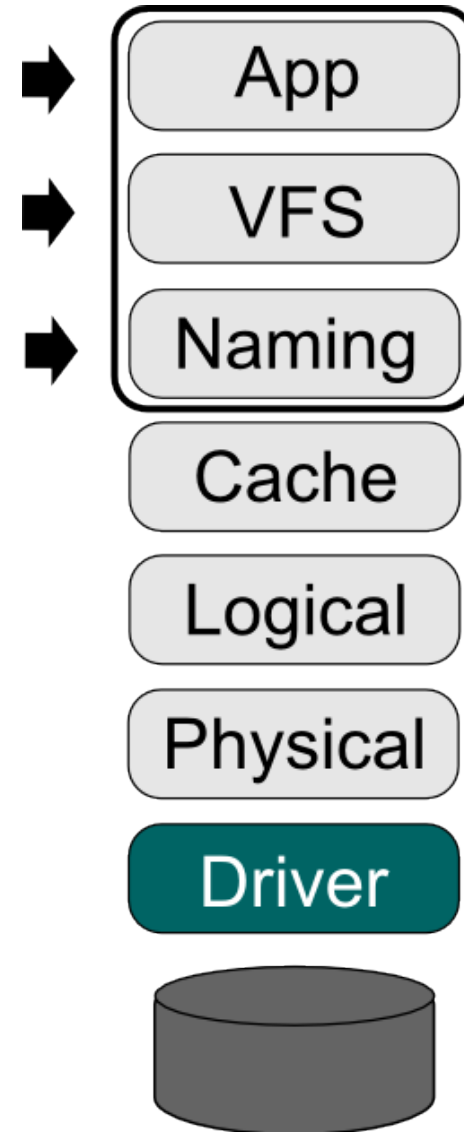
Simple restart

- Microkernel: we can **restart** crashed processes
- Simple restart only works if process is *stateless*
 - That is: it can recover its state without any help
- In the Loris stack, this only applies to **drivers**



Upper layers

- Software bugs only
- Execution environments



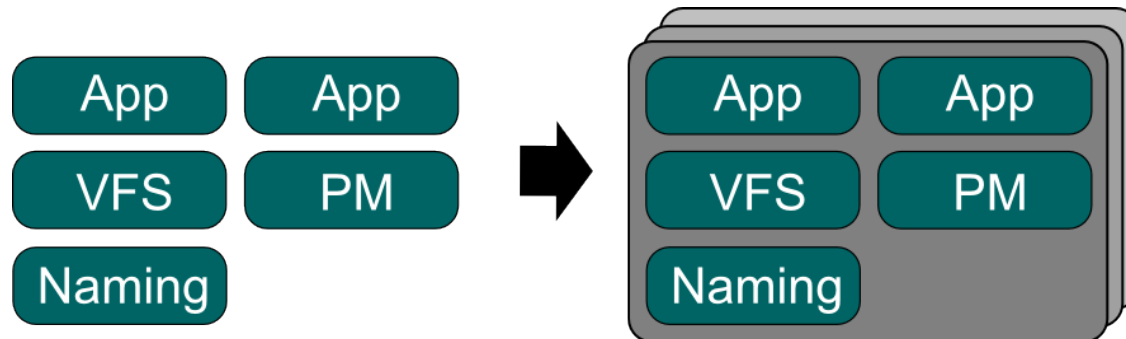
Execution Environments (1)

- VFS is highly *stateful*, and interacts with apps
 - Impossible to recover from crash transparently
 - This applies to *all* user-interfacing servers
- If we cannot recover, at least limit the damage!
 - VFS is shared by all applications
 - However, most applications *do not share VFS state*
 - So.. why not have multiple isolated VFS instances?



Execution Environments (2)

- The result: *Multiserver Execution Environments*
 - Applications and supporting servers bundled



- One base system; multiple isolated MEEs
 - Base system maintains shared resources
 - A crash in a MEE affects *only that MEE*



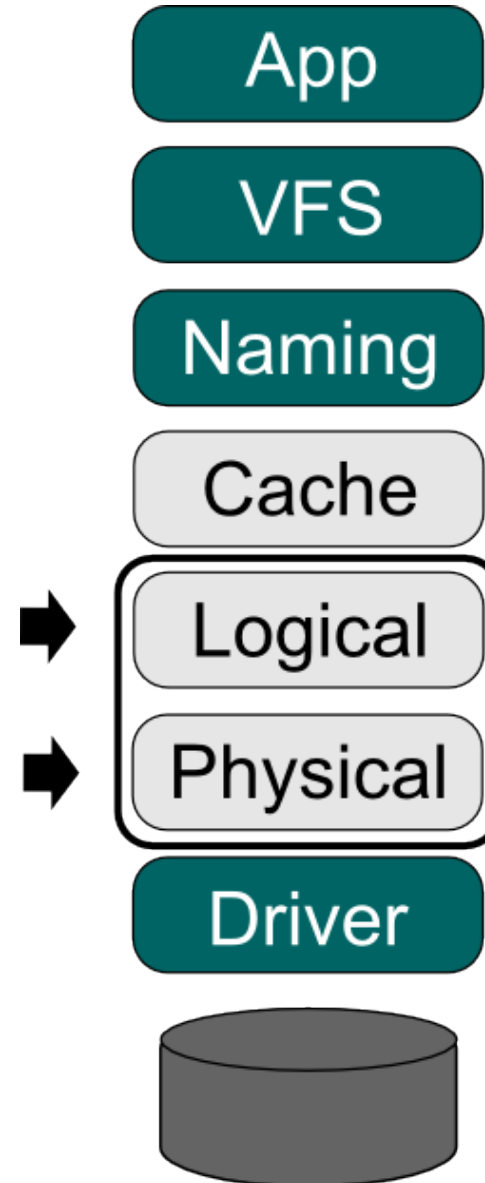
Execution Environments (3)

- Each MEE has its own file namespace
 - Private and shared files
- The naming layer is also part of the MEE
 - But we can do even better for naming layer crashes
- The cache layer must be outside the MEE
 - The only way to maintain consistency



Lower layers

- Integrated solution for:
 - Whole-system failures
 - Software bugs
- Checkpoints and logging



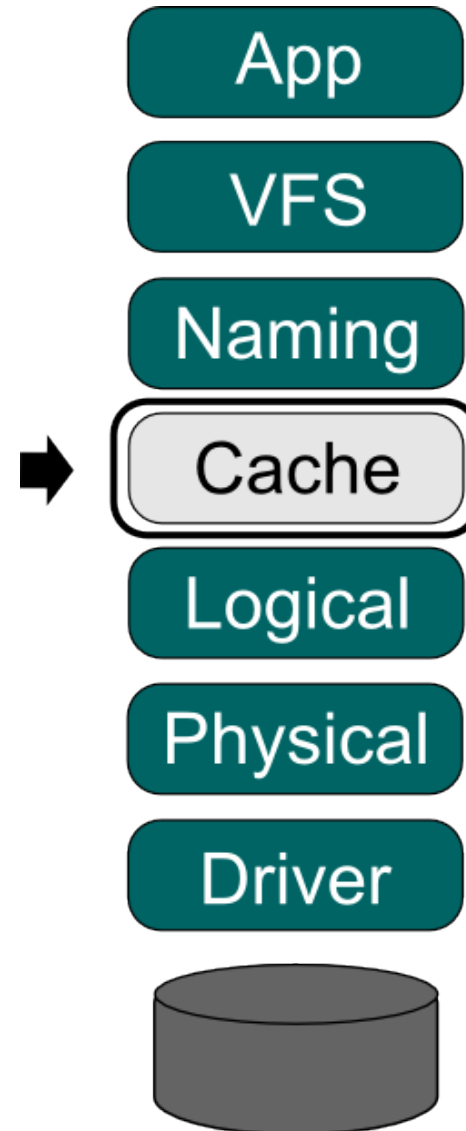
Checkpoints and logging

- Basic idea:
 - Every so many seconds, the stack takes a globally consistent **checkpoint** on the storage device(s)
 - The cache layer keeps an in-memory **log** of operations performed since the last checkpoint
 - For **system** crash recovery: reload the checkpoint
 - For **process** crash recovery: reload the checkpoint, then replay the log
- Main problem: global consistency



The middle (cache) layer

- Integrated solution for:
 - Software bugs
 - Memory corruption
- Checksumming memory



Checksumming memory

- The cache is a special case
 - Memory corruption: cache uses most memory
 - Software bugs: we cannot provide any *guarantees*
- The solution: more **checksumming!**
 - Already have/need checksums in physical layer
 - Can reuse these to detect cache memory corruption
 - Can reuse them to validate dirty pages for *best-effort* process crash recovery, too
- Tradeoff doing this *lazily*



Conclusion

- We believe that for improving storage stack dependability, the full picture is important
- We presented our plan for the entire Loris stack
- More research is needed on each of the parts



EOF

Questions?

