How  (and the Internet) Do It

Philip Levis
Stanford University
DoE Salishan Conference, 4/29/15

Two Basic Ideas

- "The End-to-End Argument"
- "Worse is Better"

End-to-End

END-TO-END ARGUMENTS IN SYSTEM DESIGN

J.H. Saltzer, D.P. Reed and D.D. Clark*

M.I.T. Laboratory for Computer Science

This paper presents a design principle that helps guide placement of functions among the modules of a distributed computer system. The principle, called the end-to-end argument, suggests that functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level. Examples discussed in the paper include bit error recovery, security using encryption, duplicate message suppression, recovery from system crashes, and delivery acknowledgement. Low level mechanisms to support these functions are justified only as performance enhancements.

End-to-End

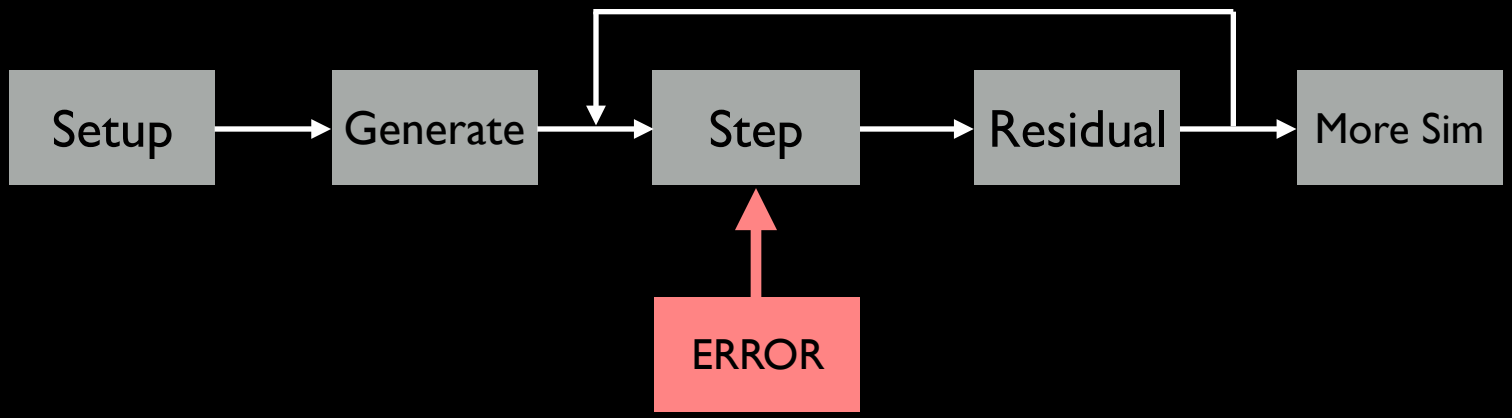
"Thus the amount of effort to put into reliability measures within the data communication system is seen to be an engineering tradeoff based on performance, rather than a requirement for correctness… the end-to-end check of the file transfer application must still be implemented no matter how reliable the communication system becomes."

Correctness

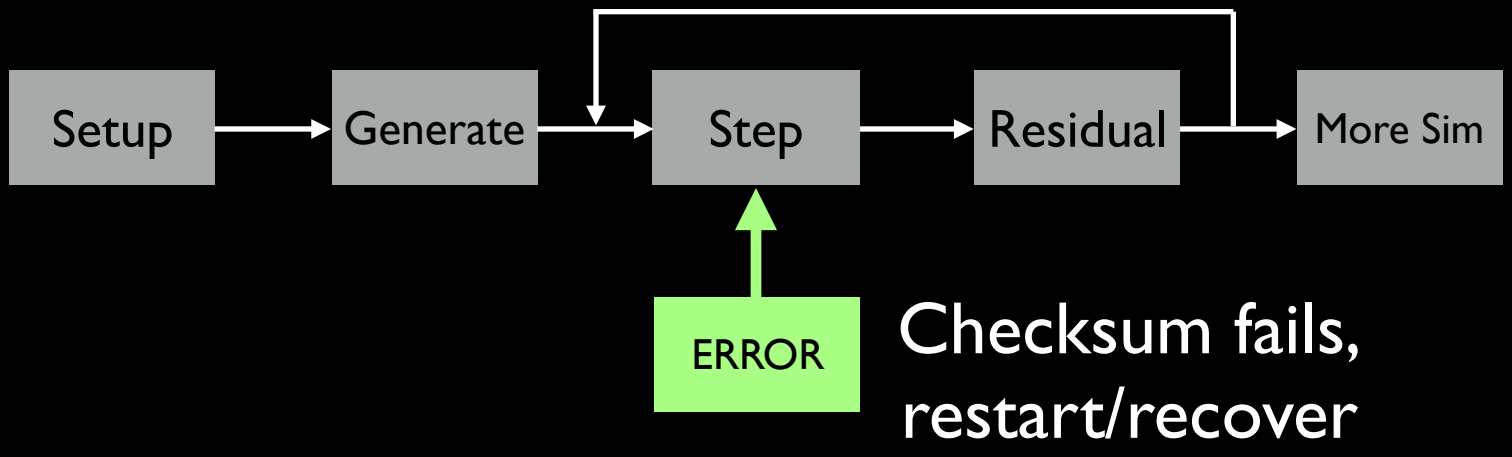
- Correctness MUST be end-to-end
- Everything else is guessing and rolling dice
- Want to be sure your simulation is "correct"?
  - You MUST have a correctness check for its end result
  - All intermediate resiliency efforts are performance improvements
- "Application" examples yesterday: solvers
  - Correctness check: mass is conserved
  - Compute mass before solve, after solve, check they are equal
  - What if the error occurs before mass is computed?

End To End

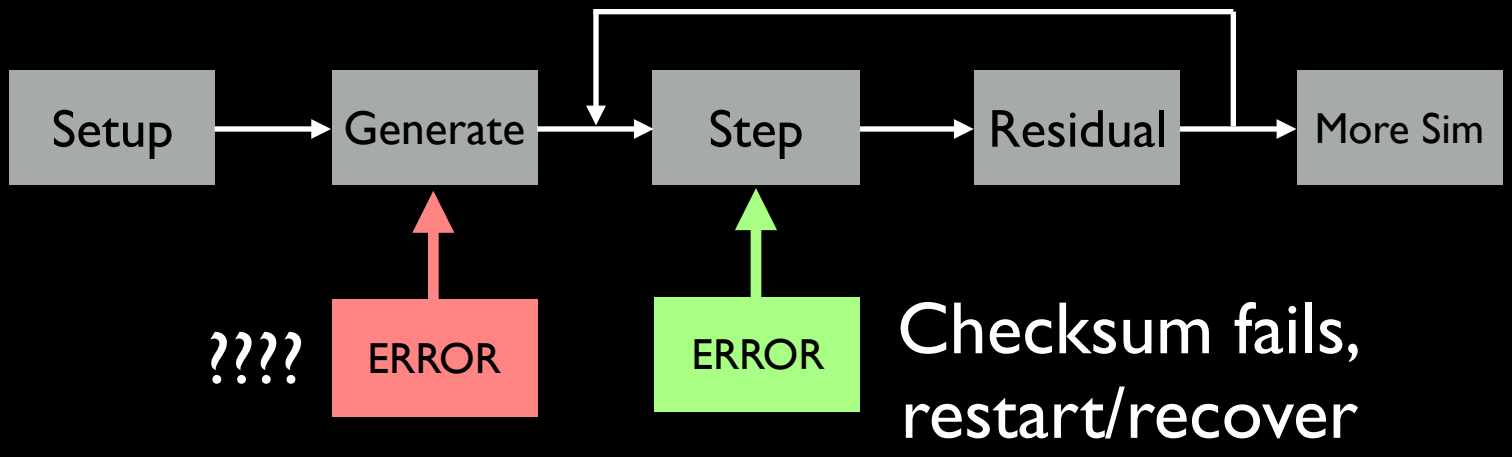
End To End



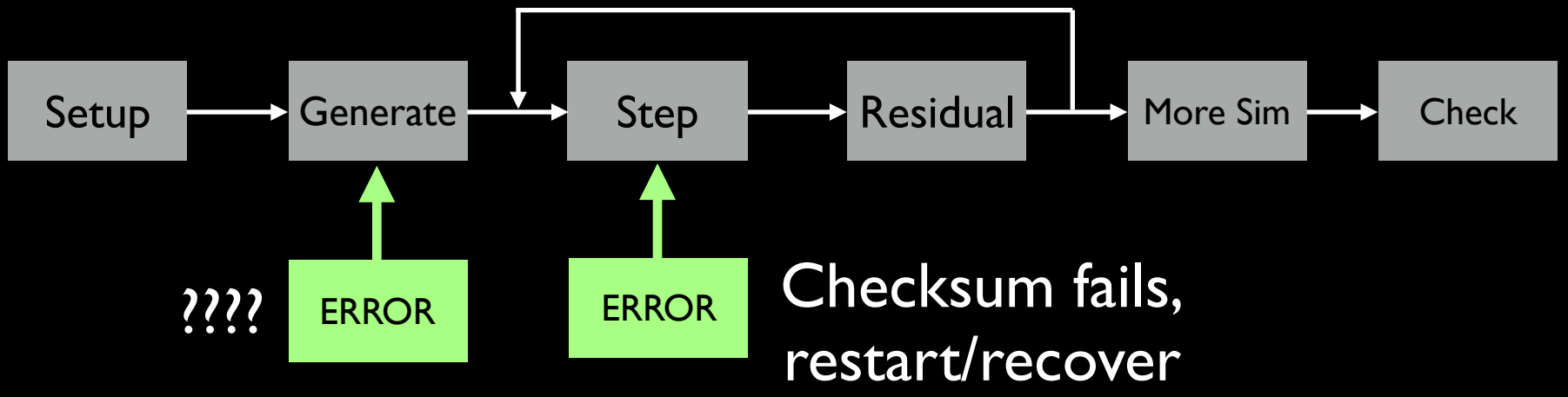
End To End


Setup
Generate
Step
Residual
More Sim
ERROR
Checksum fails,
restart/recover

End To End


Setup
Generate
Step
Residual
More Sim
????
ERROR
ERROR
Checksum fails,
restart/recover

End To End


Setup
Generate
Step
Residual
More Sim
Check
????
ERROR
ERROR
Checksum fails,
restart/recover

How To Think About It

- Correctness is an end-to-end property
  - Do these simulation results make sense?
  - Physical world experiments have errors, why are simulations different?
- Add intermediate checks if and only if their benefit is greater than their cost
  - Cost of matrix checksums < performance benefit of partial restarts
- Break a system into layers, define what correctness is for each layer (but provide no guarantees, just best effort)
  - Anyone who guarantees anything is lying: it's all probabilities
  - Many layers won't need checks

Example: HTTPS

Sometimes errors occur: reload!

Application Layer: HTTP	No checks!!
Session Layer: TLS	Message authentication code (strong)
Transport Layer: TCP	Segment checksum (weak)
Network Layer: IP	Header checksum (weak)
Link Layer: Ethernet	Spreading, CRC (strong)
Physical Layer: 1000BaseT	Tons of errors, high coding (strong)

Example: HTTPS

Sometimes errors occur: reload!

Application Layer: HTTP	No checks!!
Session Layer: TLS	Message authentication code (strong)
Transport Layer: TCP	Segment checksum (weak)
Network Layer: IP	Header checksum (weak)
Link Layer: Ethernet	Spreading, CRC (strong)
Physical Layer: 1000BaseT	Tons of errors, high coding (strong)

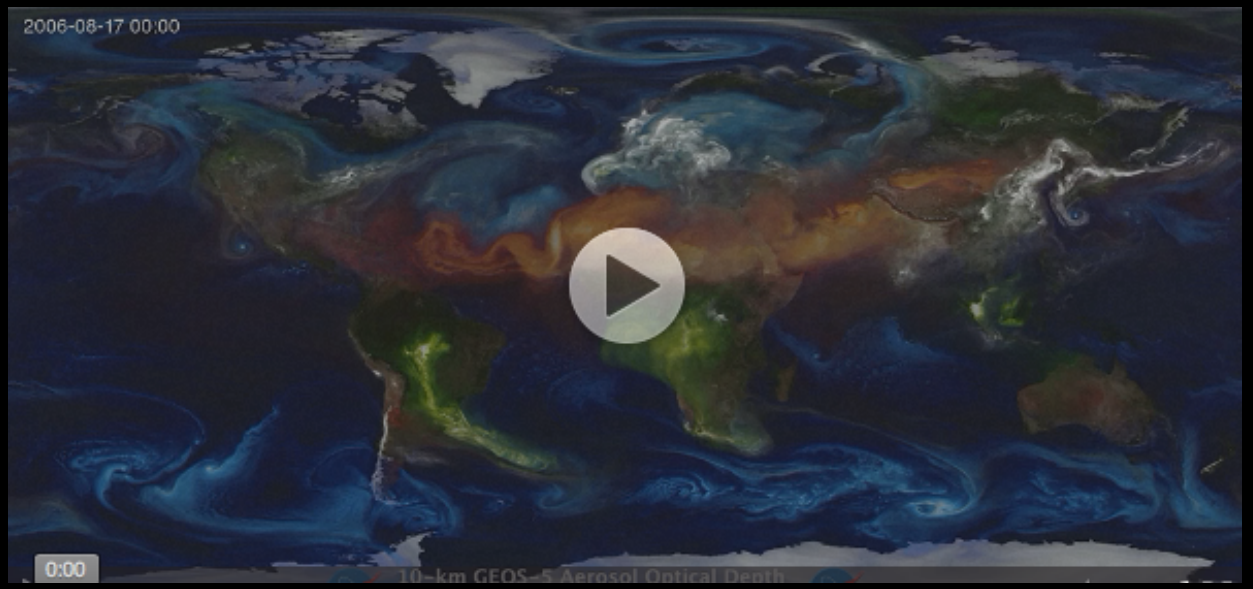
Example: HTTPS

Sometimes errors occur: reload!

Wireless networks see **many** more errors than wired ones: you can't control the medium. So WiFi uses single-hop acknowledgements (Ethernet does not). The cost of waiting for an acknowledgement after each frame (10 µs) is less than the cost of waiting to detect a loss end-to-end (100ms).

Network Layer: IP	Header checksum (weak)
Link Layer: WiFi	Spreading, CRC, retransmit (strong!)
Physical Layer: 1000BaseT	Tons of errors, high coding (strong)

Ultimate End-to-End



The scientists detected the error.
The system worked.

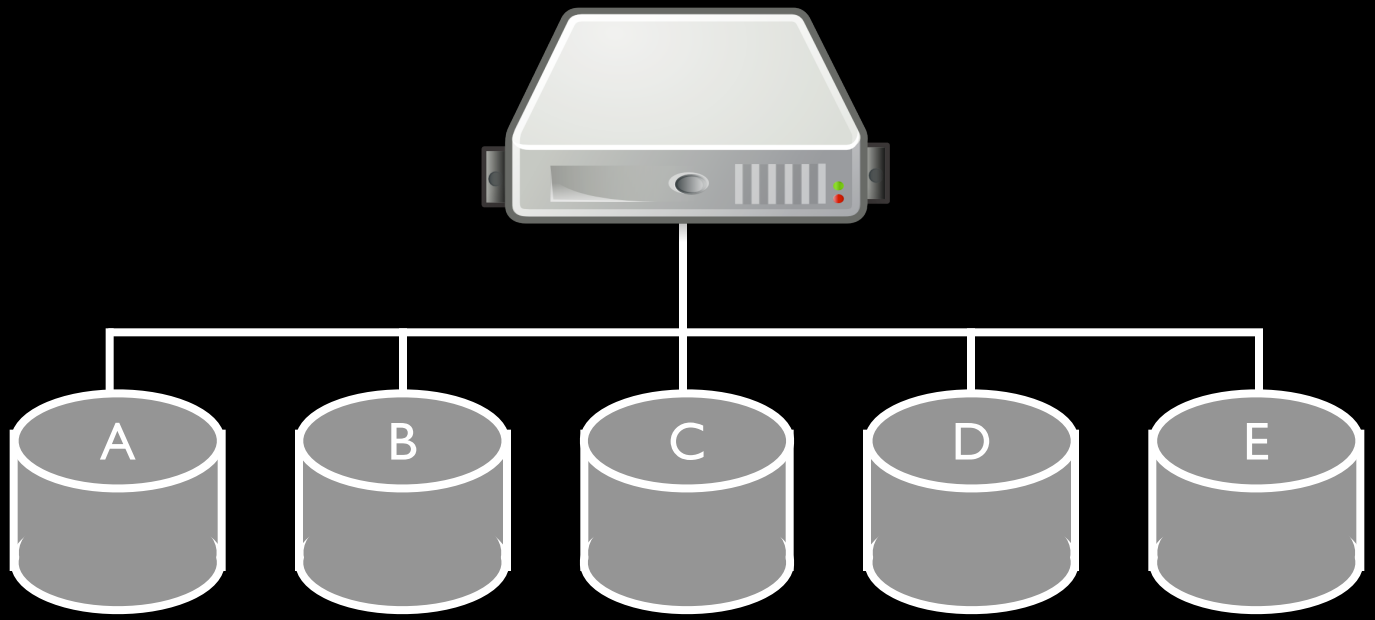
“Worse is Better”

- Simplicity is the most important consideration in a design.
- It is slightly better to be simple than correct.
- It is better to drop those parts of the design that deal with less common circumstances than to introduce either implementational complexity or inconsistency.
- Completeness must sacrificed whenever implementation simplicity is jeopardized.

Reliable from Unreliable

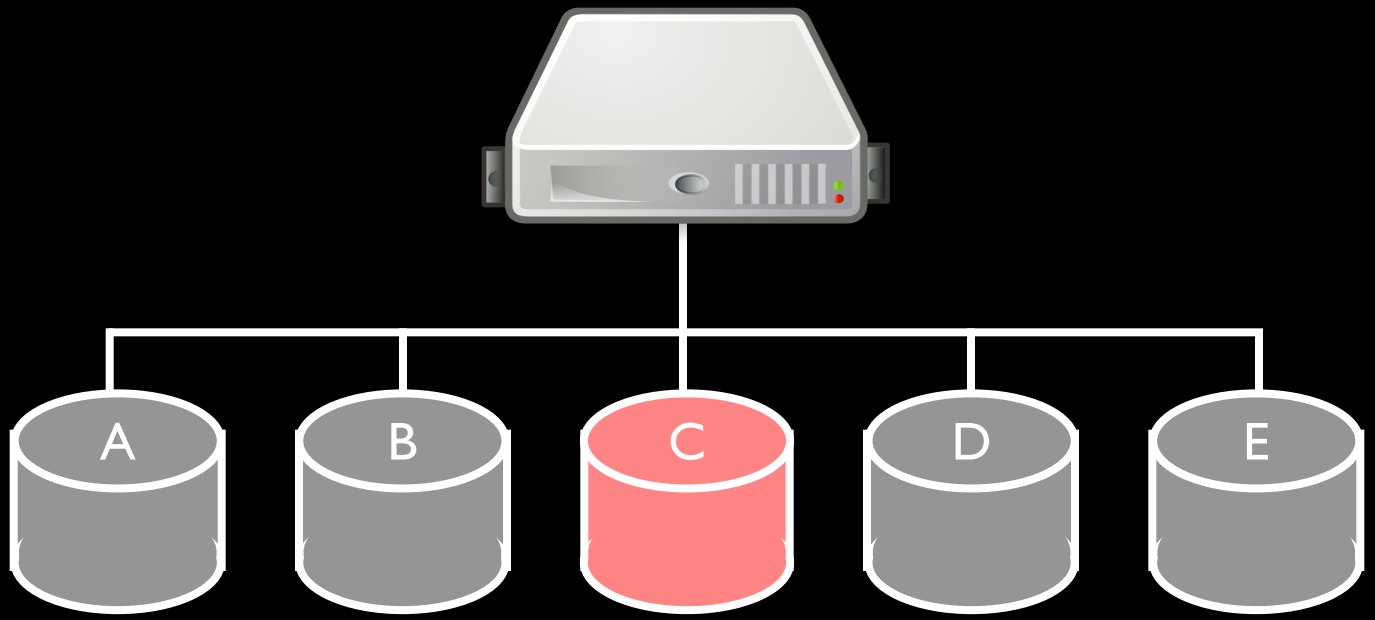
- Large, highly reliable things are expensive
- Make something large and reliable out of a bunch of smaller, unreliable parts
  - RAID: redundant array of inexpensive disks
  - GFS: cross-node storage
  - MapReduce
- Add a bit of redundancy, embrace and accept failure
- Use end-to-end checks!

RAID



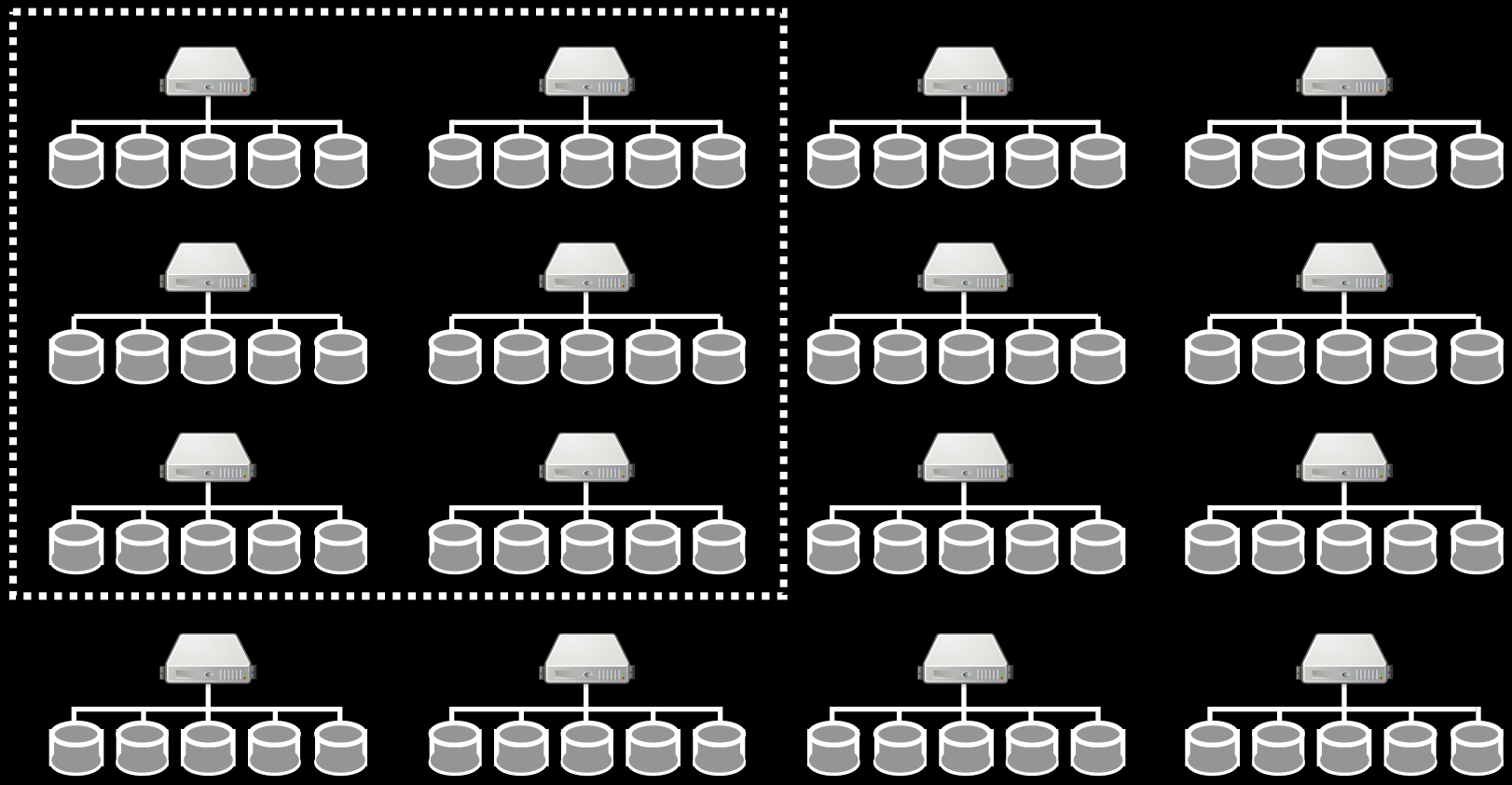
$$A = B \oplus C \oplus D \oplus E$$

RAID



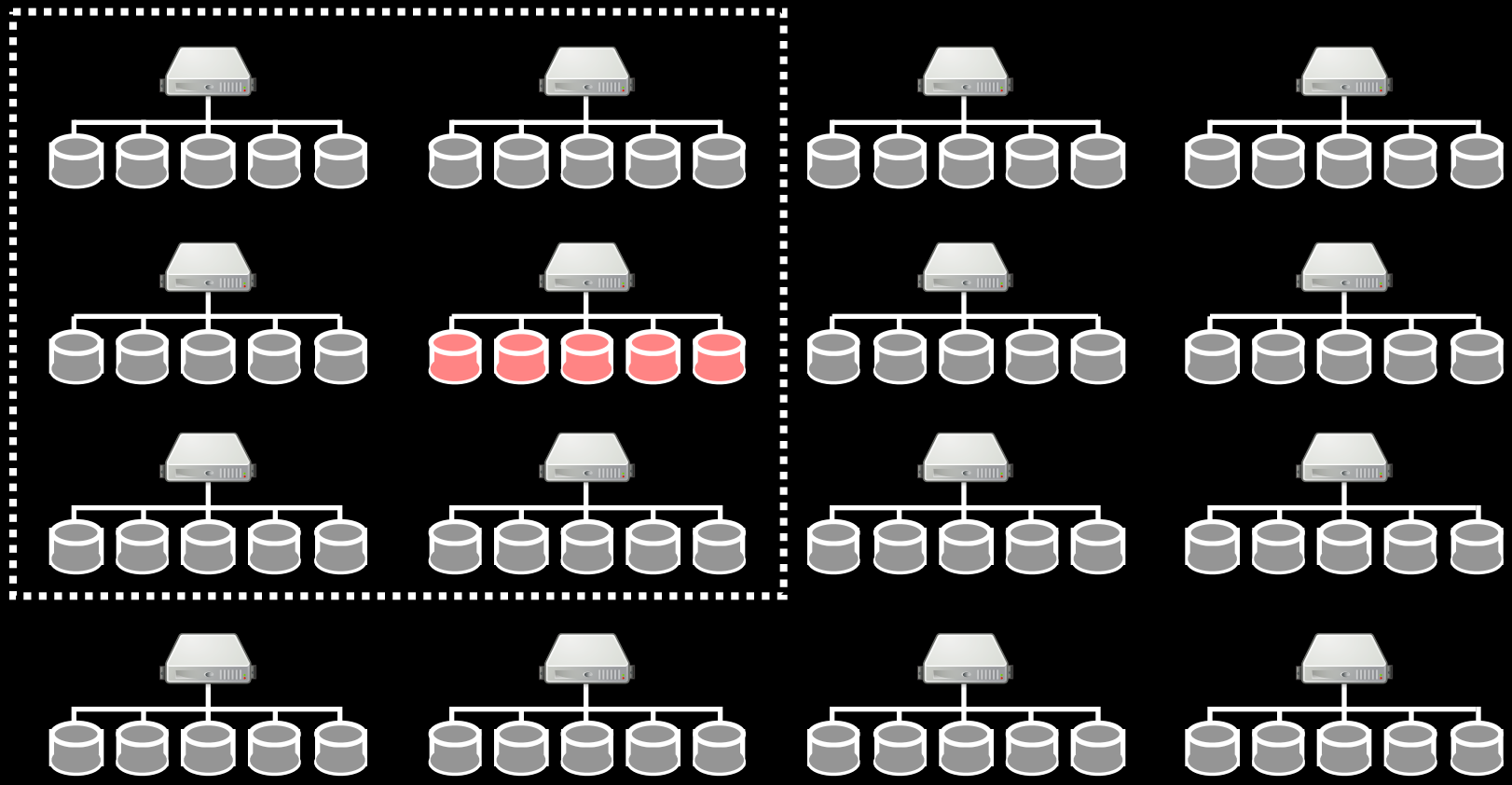
$$A = B \oplus C \oplus D \oplus E$$

Next Level Up



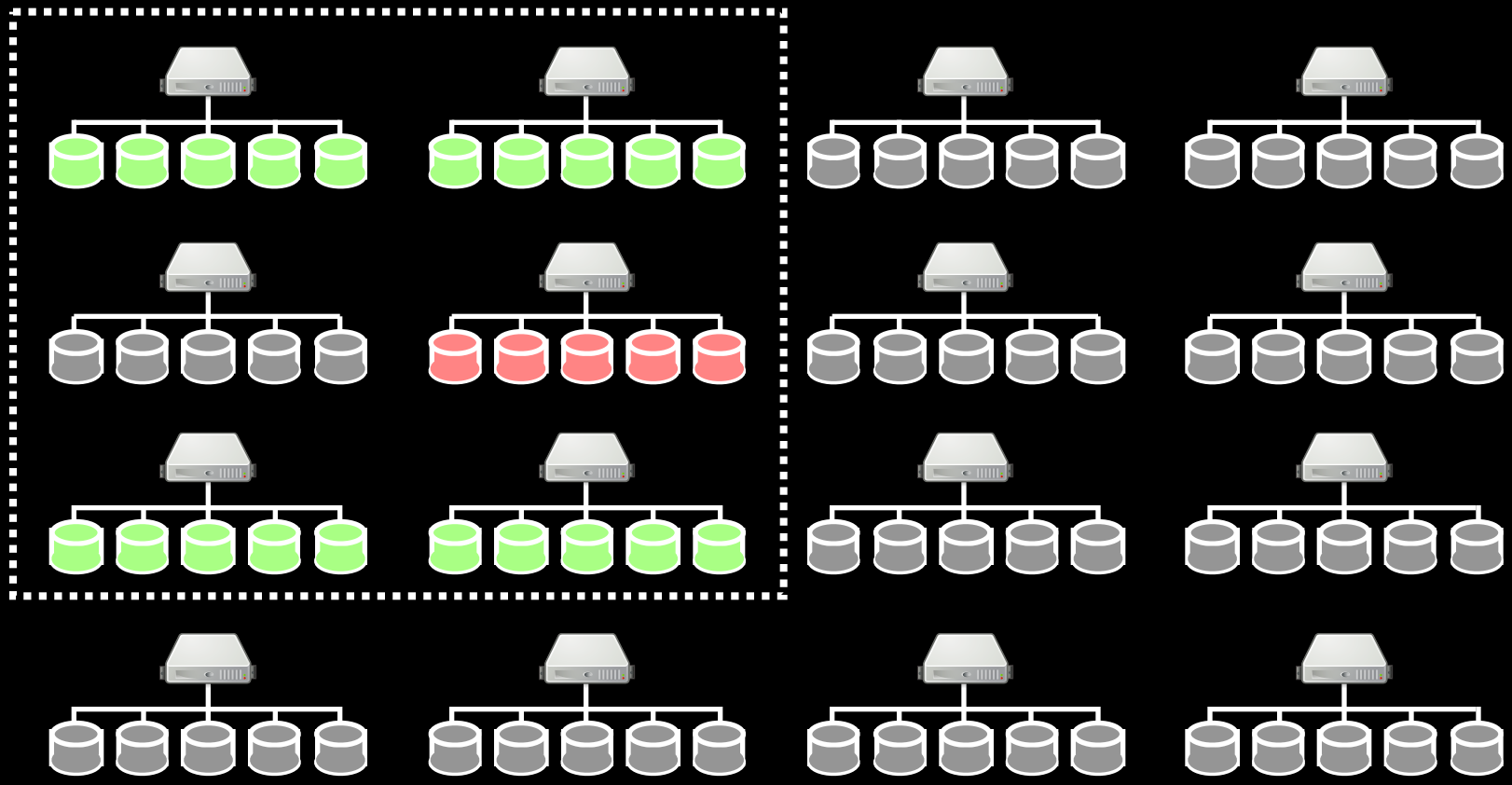
Reed-Solomon code data across nodes

Next Level Up



Reed-Solomon code data across nodes

Next Level Up



Reed-Solomon code data across nodes

Rubber Hits the Road

- The basic principles for building resilient/fault-tolerant systems are well known, well established, and demonstrated over the past 40 years
- How to apply these principles to HPC workloads is not!
  - Many good ideas presented this week
- How do they fit together into an overall framework?

A Cultural Shift

>

Everywhere


Powered by Diesel
03
RedBull
MDV
android
MDV
MOHR DAVIDOW VENTURES

>


CATERPILLAR
25

Some Papers

- End-to-End Arguments in System Design
- The Rise of "Worse is Better"
- Hints for Computer System Design
  - "There are two problems with the end-to-end strategy…"
- Examples
  - A Case for Redundant Arrays of Inexpensive Disks
  - MapReduce: Simplified Data Processing on Large Clusters
  - Resilient Distributed Data Sets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
  - Availability in Globally Distributed Storage Systems