Google Cloud

Preventing Silent Data Corruption in Spanner, a Hyper-scale Database

David F. Bacon

Google Database Systems

DFT 2024 Keynote

37th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems

10 October 2024

Contributors:

Spanner: Elaine Ang, Michael Klose, Alan Li, Todd Lipcon

Hardware Platforms: Mike Fulton

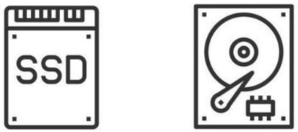
Site Reliability Engineering: Peter Hochschild

Data Corruption Monitoring: Matthew O'Neil

go/spanner-sdc-dft

What is a Hyper-scale Database?

Spanner's Scale



15 Exabytes

5 B QPS

What Drives Spanner's Scale?

Photos Play Search Maps AdWords YouTube Gmail Calendar Drive Docs Meet

Highly Diverse Workloads

Photos Play Search Maps AdWords YouTube Gmail Calendar Drive Docs Meet

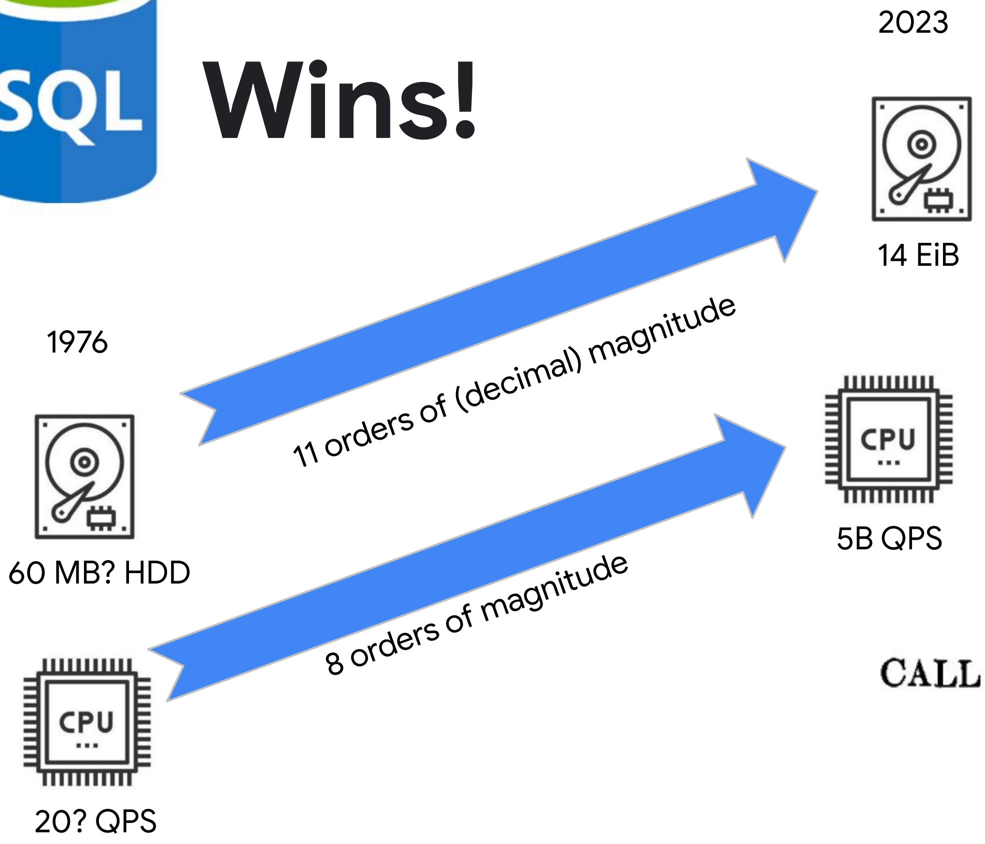
Google Cloud
Cloud Control Plane

Cloud Spanner

Zanzibar

google3
Google's internal infrastructure

Wins!




```
CALL SEQUEL('UNDERPAID(NAME, SAL) ←
        SELECT   NAME, SAL
        FROM     EMP
        WHERE    JOB = ''PROGRAMMER''
        AND      SAL < 10,000');
```

Astrahan et al: "System R: Relational Approach to Database Management" *ACM TODS 1*, 2 (June 1976)

Programmers Not So Much

```
CALL SEQUEL('UNDERPAID(NAME, SAL) ←
        SELECT   NAME, SAL
        FROM     EMP
        WHERE    JOB = ''PROGRAMMER''
        AND      SAL < 10,000');
```

1976

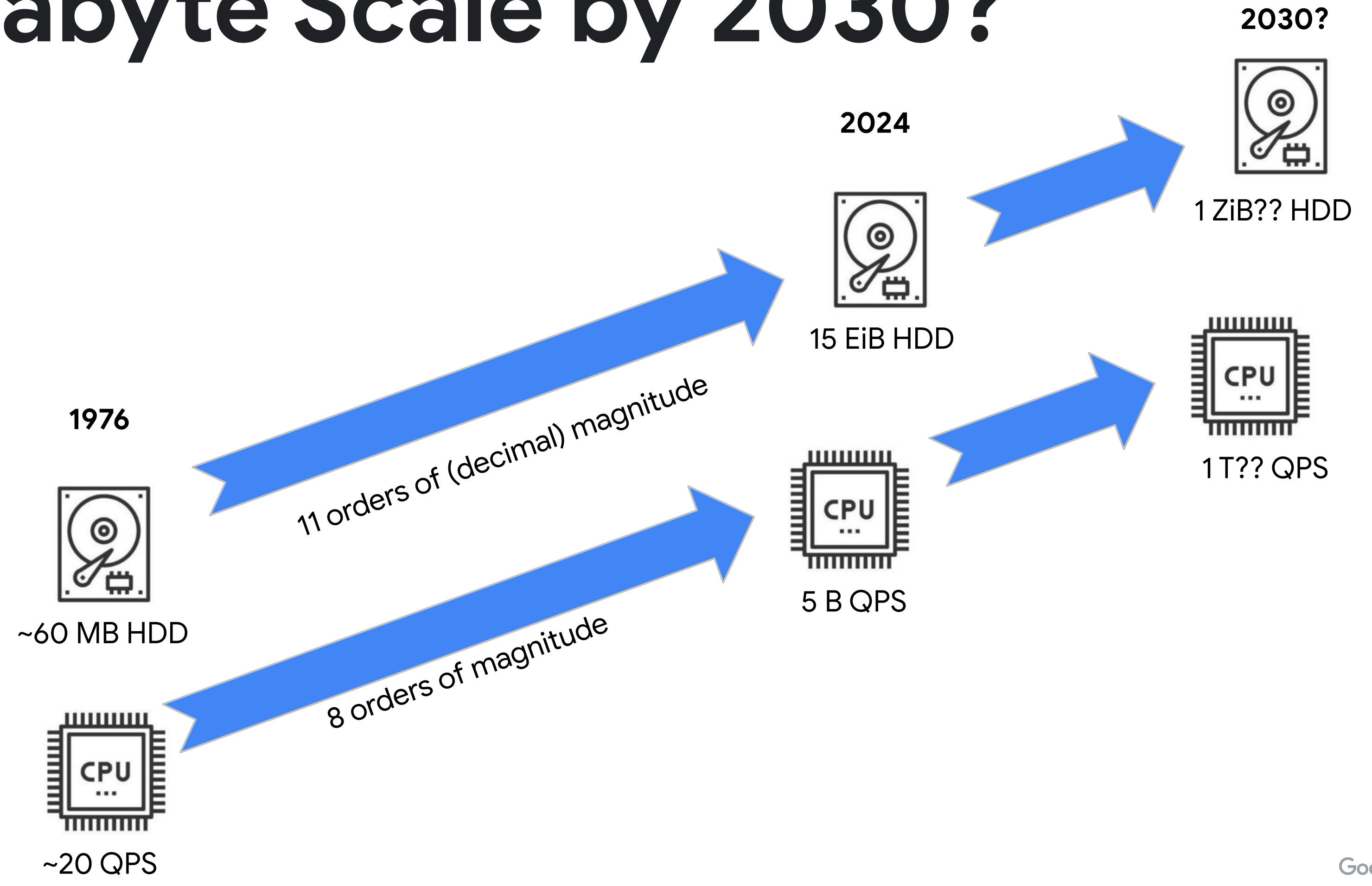
2023

< $10,000

1 order of magnitude

< $100,000

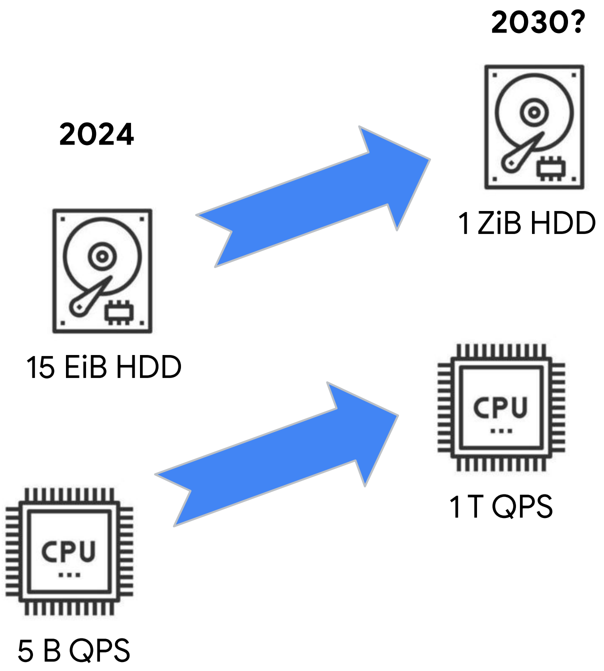
Zettabyte Scale by 2030?



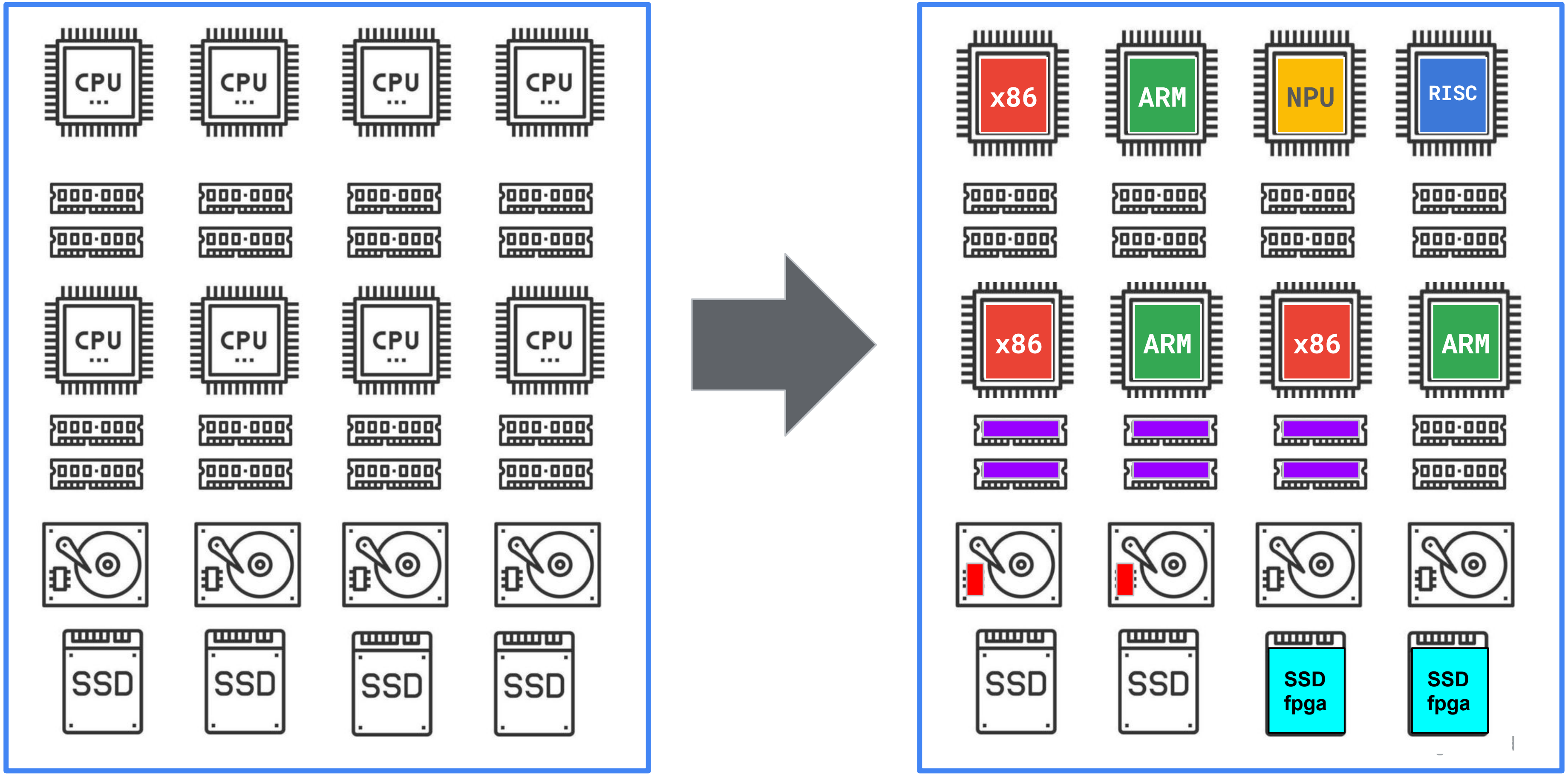
Reliability versus Scale

Reliability Must Scale

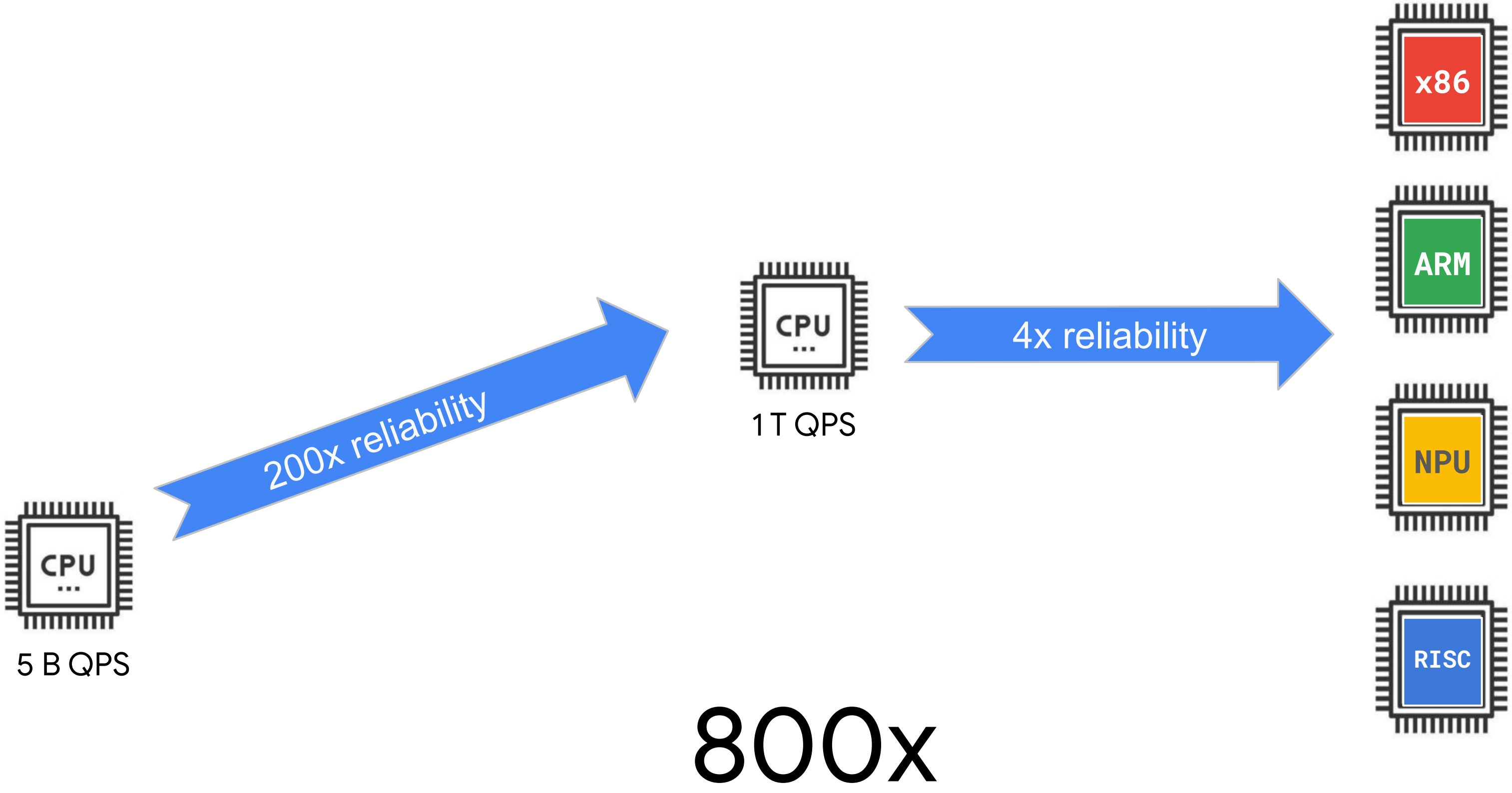
- 64x more data
  - 64x more reliable per byte
- 200x more compute
  - 200x more reliable per op



Diversity: The Key to Performance?

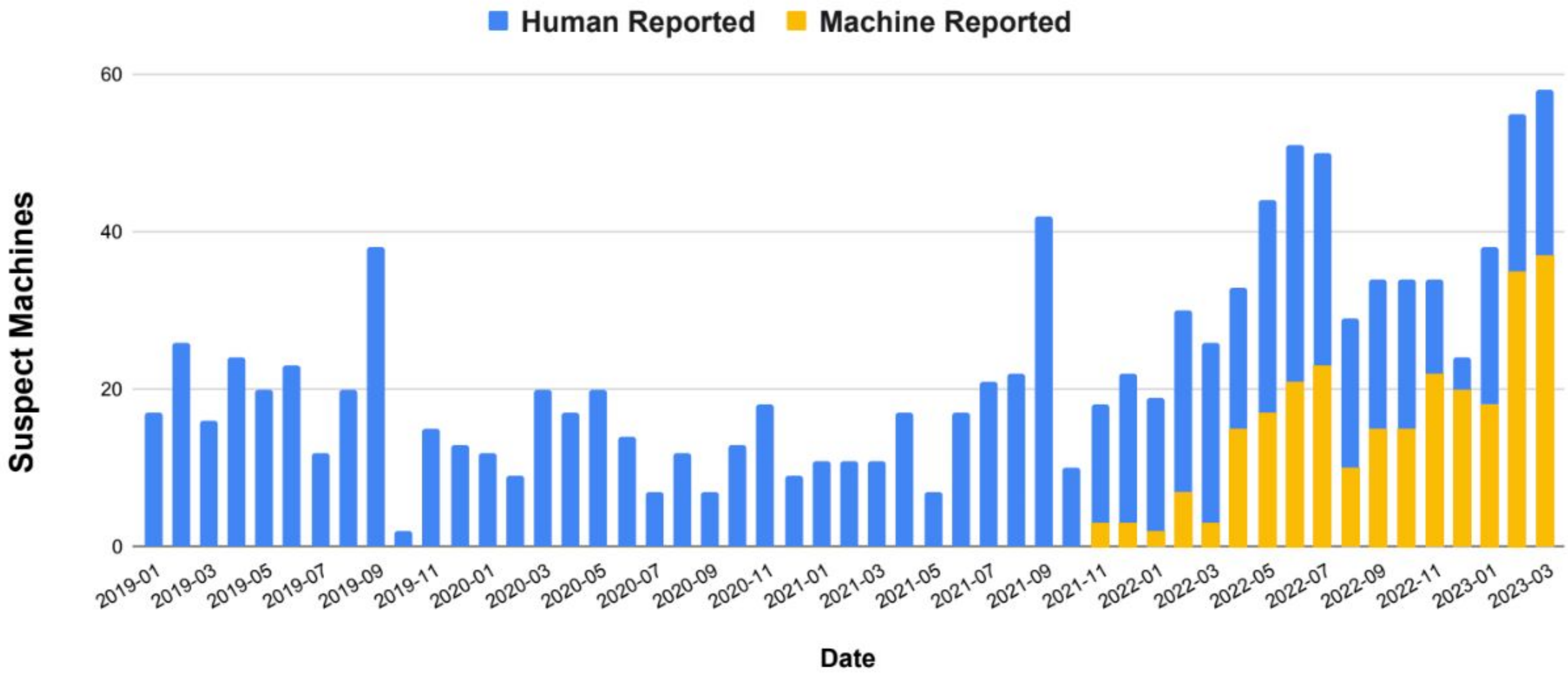


Reliability Must Scale with Diversity

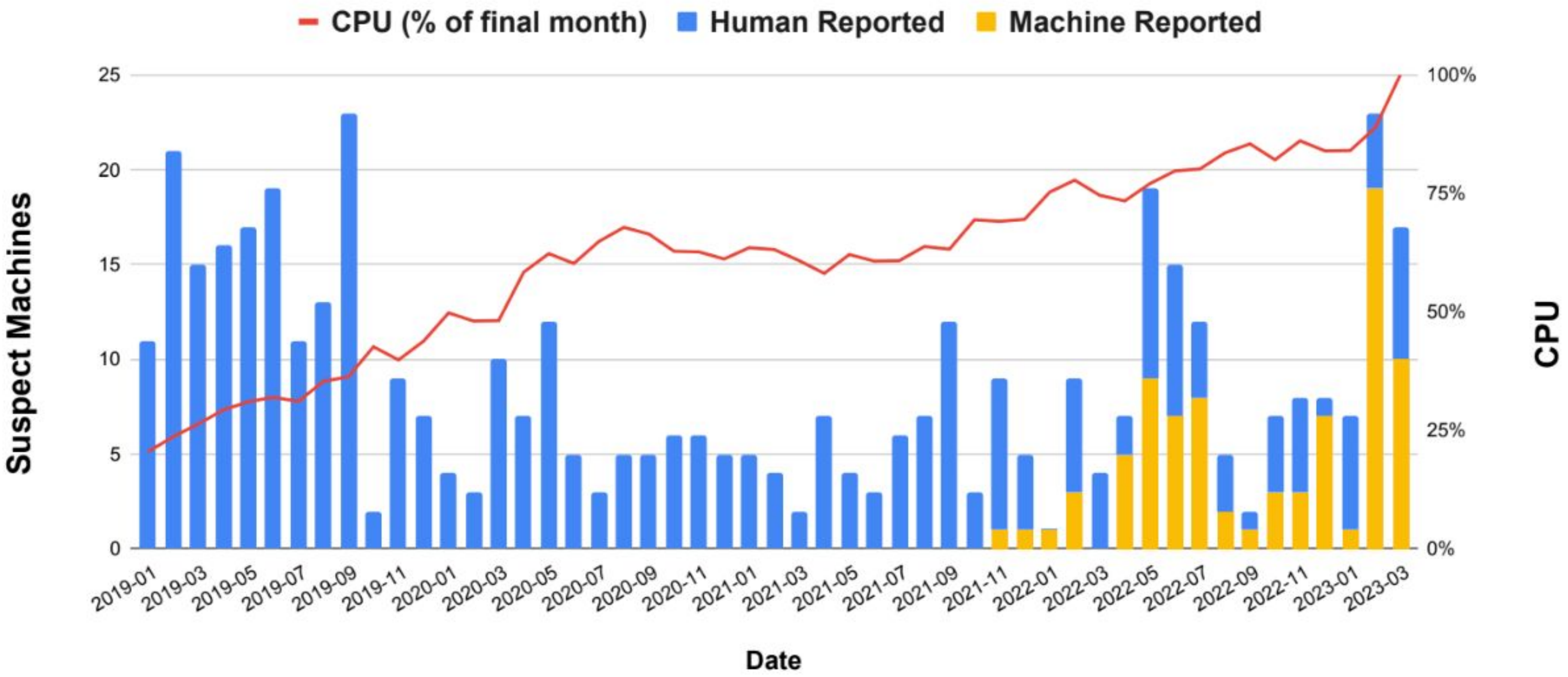


Data Corruption Monitoring

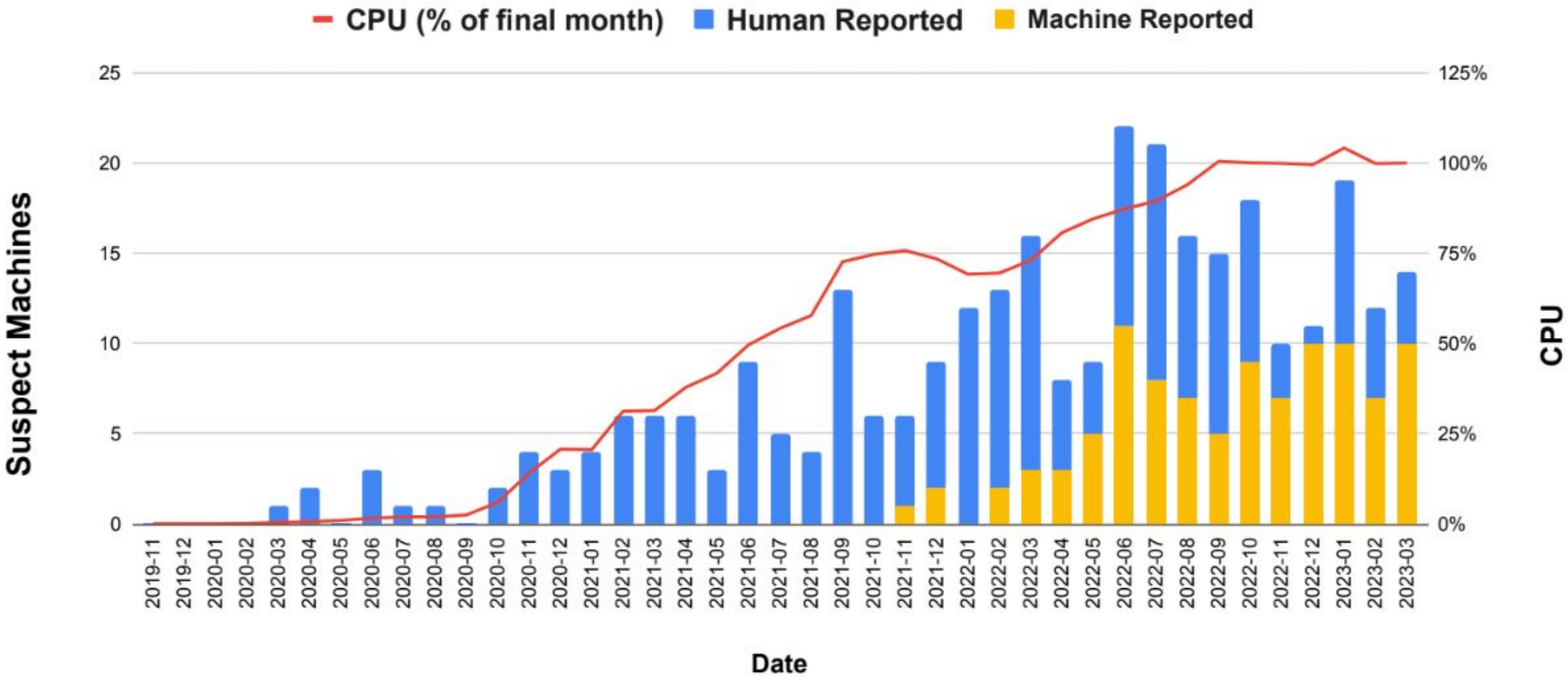
Spanner DCM Reports (4 years)



SDC Reports vs CPU Growth (CPU A)

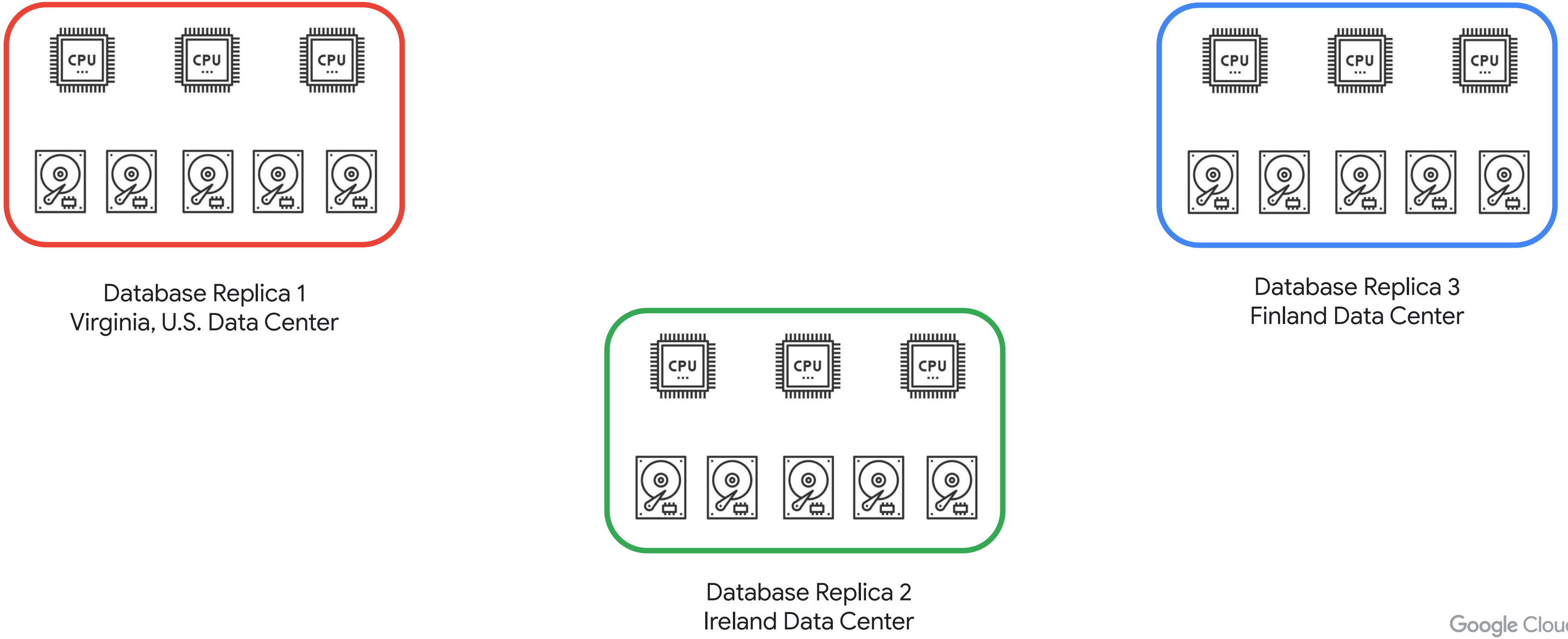


SDC Reports vs CPU Growth (CPU B)

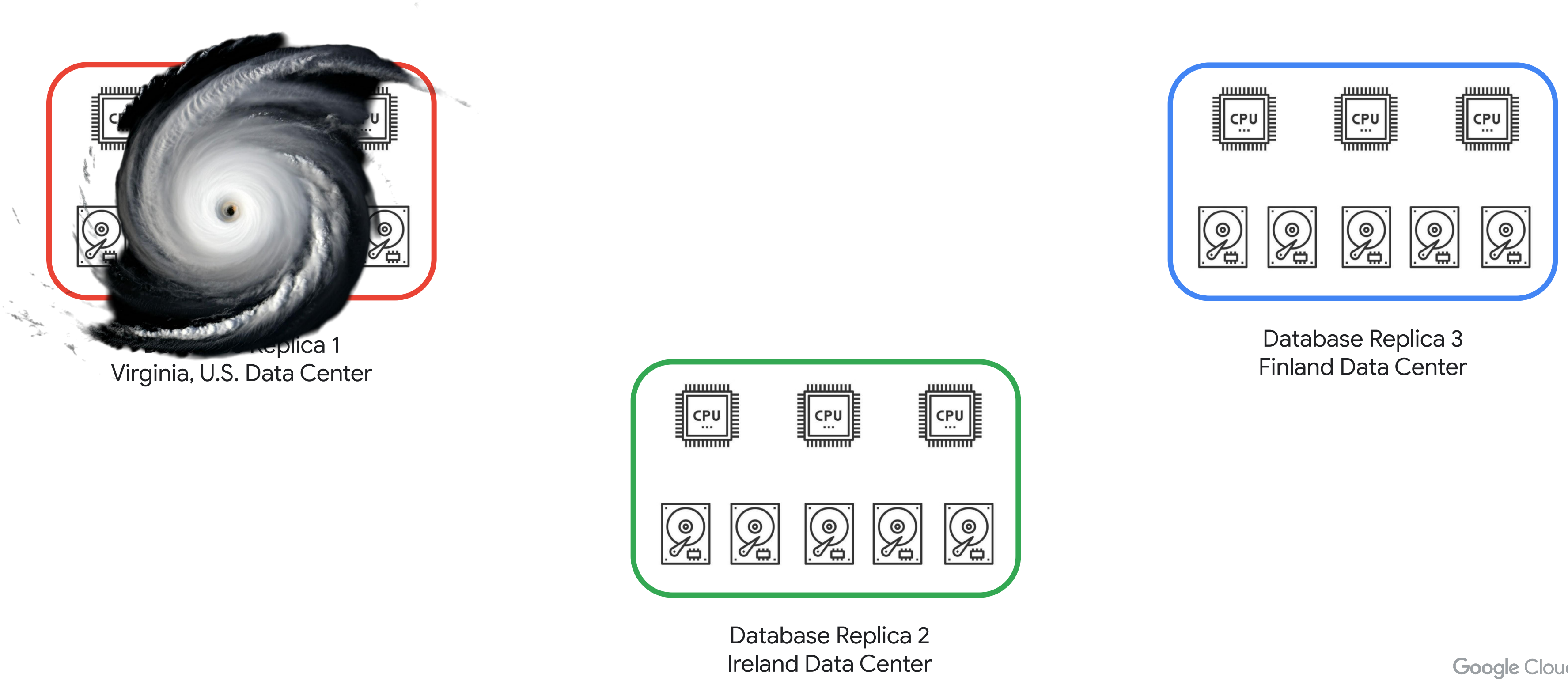


Spanner’s Resilience Architecture

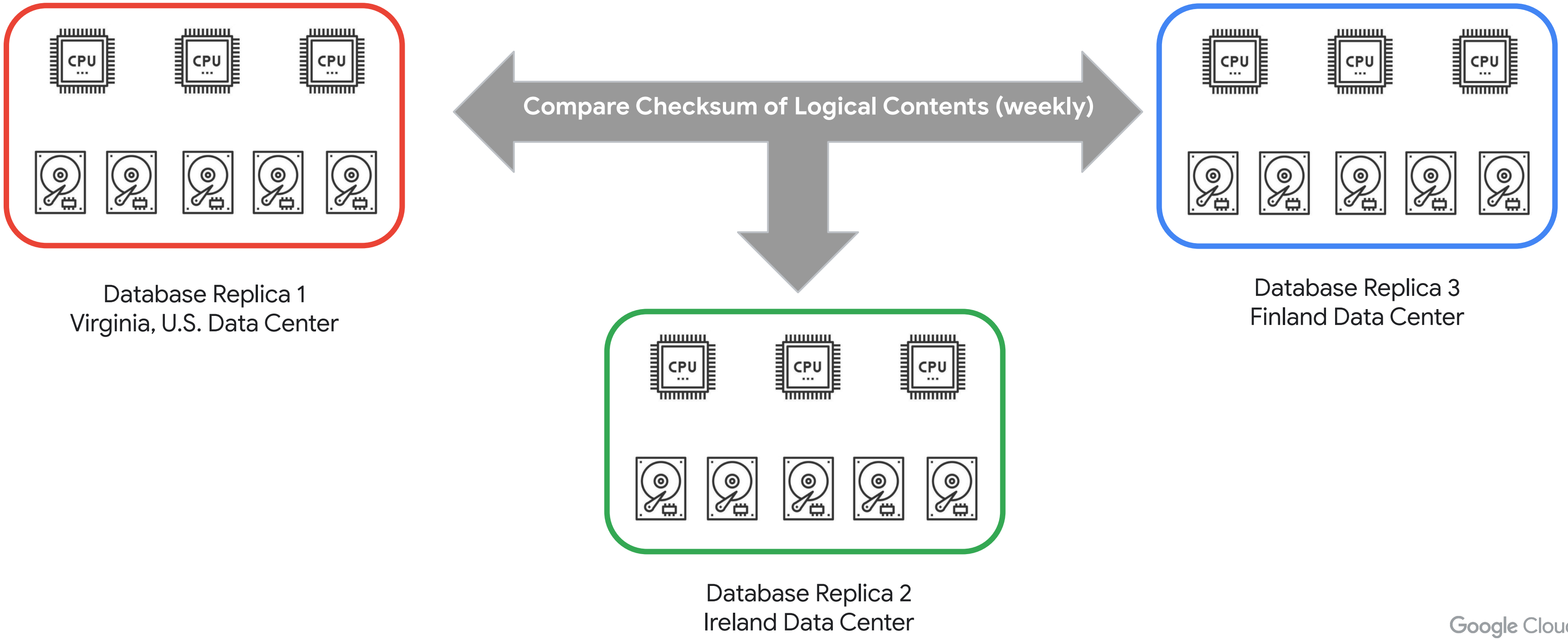
Geographic Replication



Resilient to Failure (with Quorum)

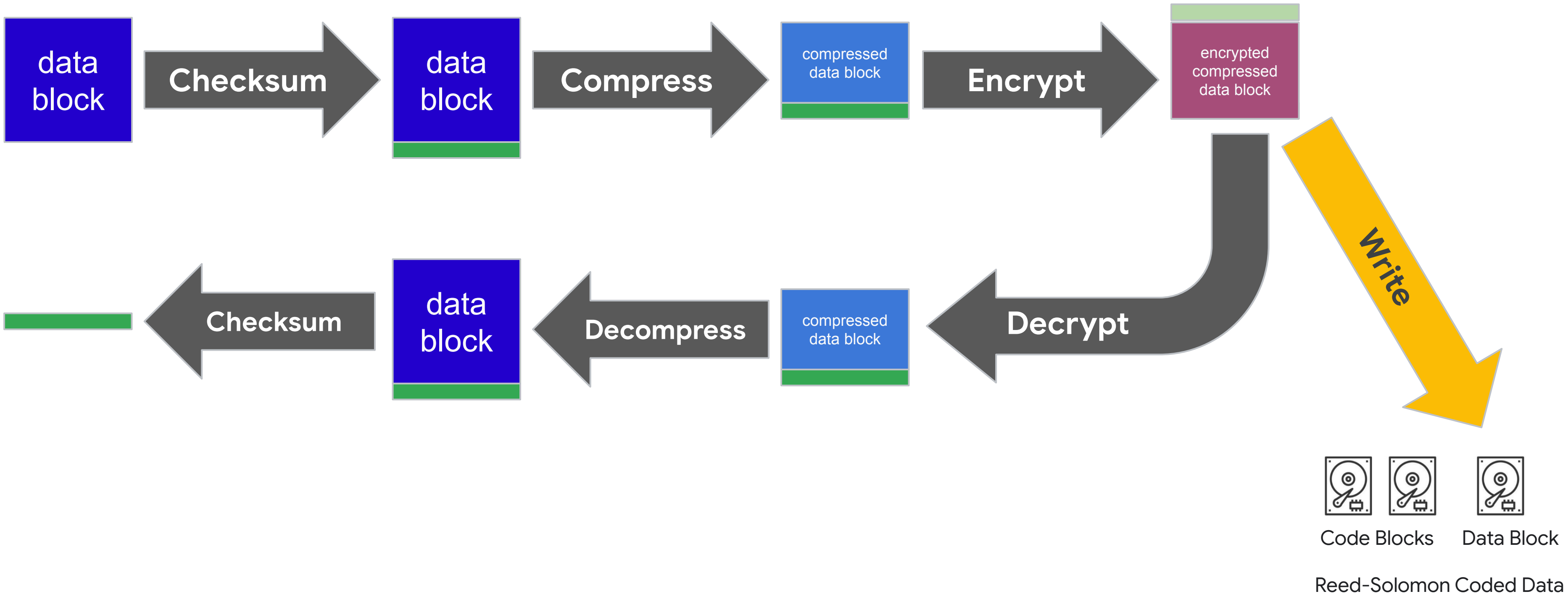


Ensuring Replica Consistency



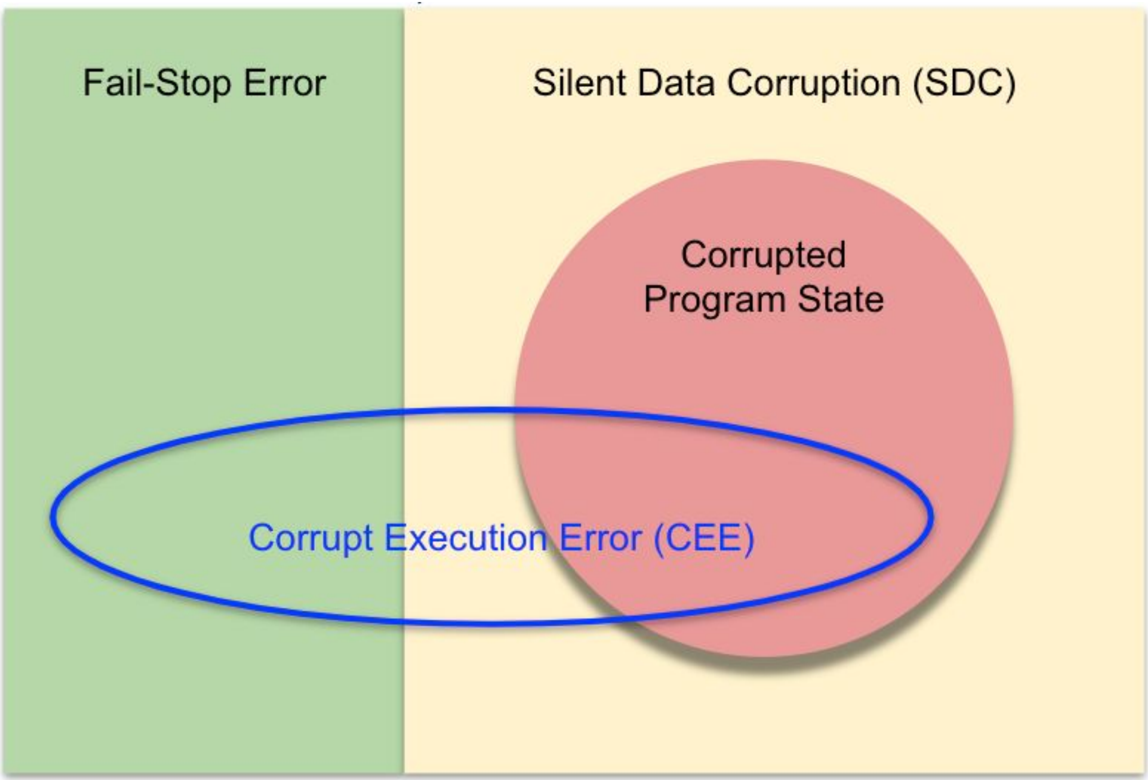
Writing the Data

Protecting Writes Against Corruption

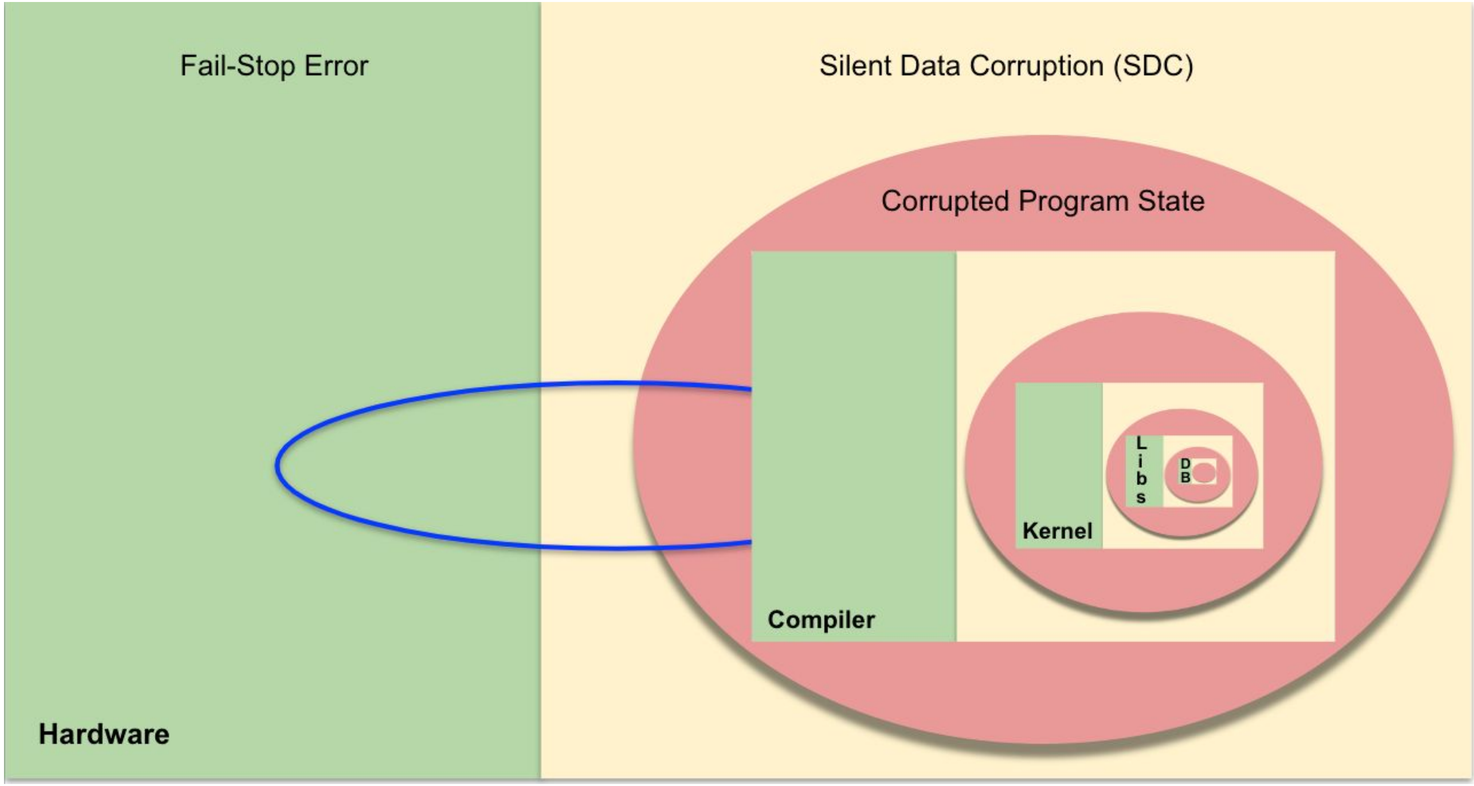


What *is* Silent Data Corruption?

How We Used to Think About It

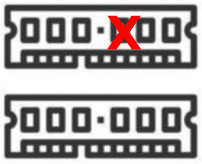


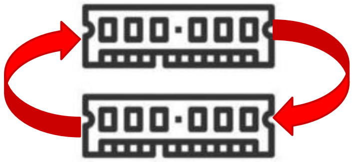
How We Think About It Now

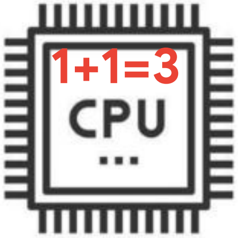


War Stories

Hardware: Memory Errors and CEE







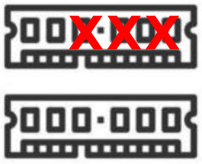
Bit Flip
(we use ECC)

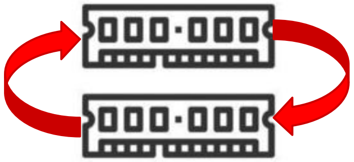
Memory Swap
DIMM Row Count Error

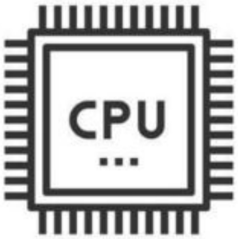
Corrupt Execution Error
Movedir Crash Loop
Mail Corruption
spanner-machines-of-death
(882 as of October 2024)

Markoff, "Tiny Chips, Big Headaches", NY Times, 2022.
Bacon, "Detection and Prevention of Silent Data Corruption in an Exabyte-scale Database System", IEEE Workshop on Silicon Errors in Logic – System Effects (2022).
Hochschild et al, "Cores that Don't Count" Workshop on Hot Topics in Operating Systems (HotOS 2021).

Compiler



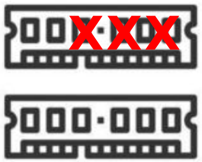




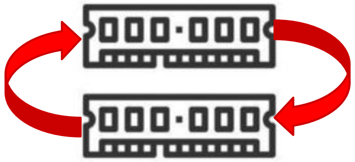
Torn Atomics

std::atomic<int64>
crosses cache line

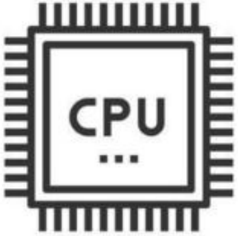
Kernel



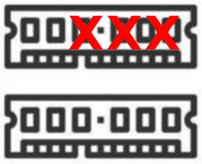
AMD Context Switch Race

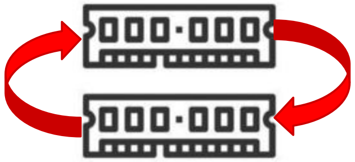


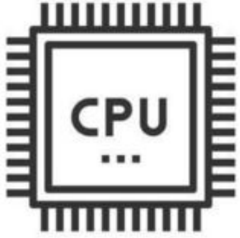
NVRAM Page Table Corruption



Libraries







`tcmalloc Experiment`

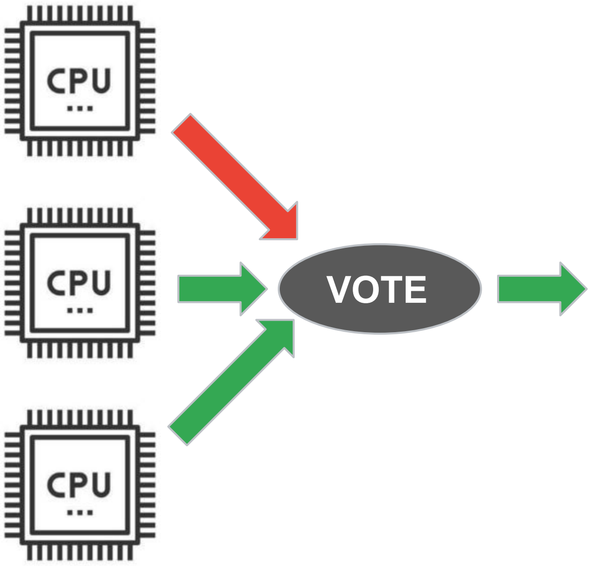
`std::sort stomp`

What's Different About Hardware Bugs?

- Software bugs have an application pattern that emerges over time
  - e.g. always happens in objects of size 432
- Hardware bugs are largely only confined to hardware models
  - Hardware-specific software bugs can muddy the waters
- They create a "fog of war"
  - Delays response to both hardware and software bugs
  - Only applications with a very low bug rate can distinguish

Prevention Techniques

n-Modular Redundancy?



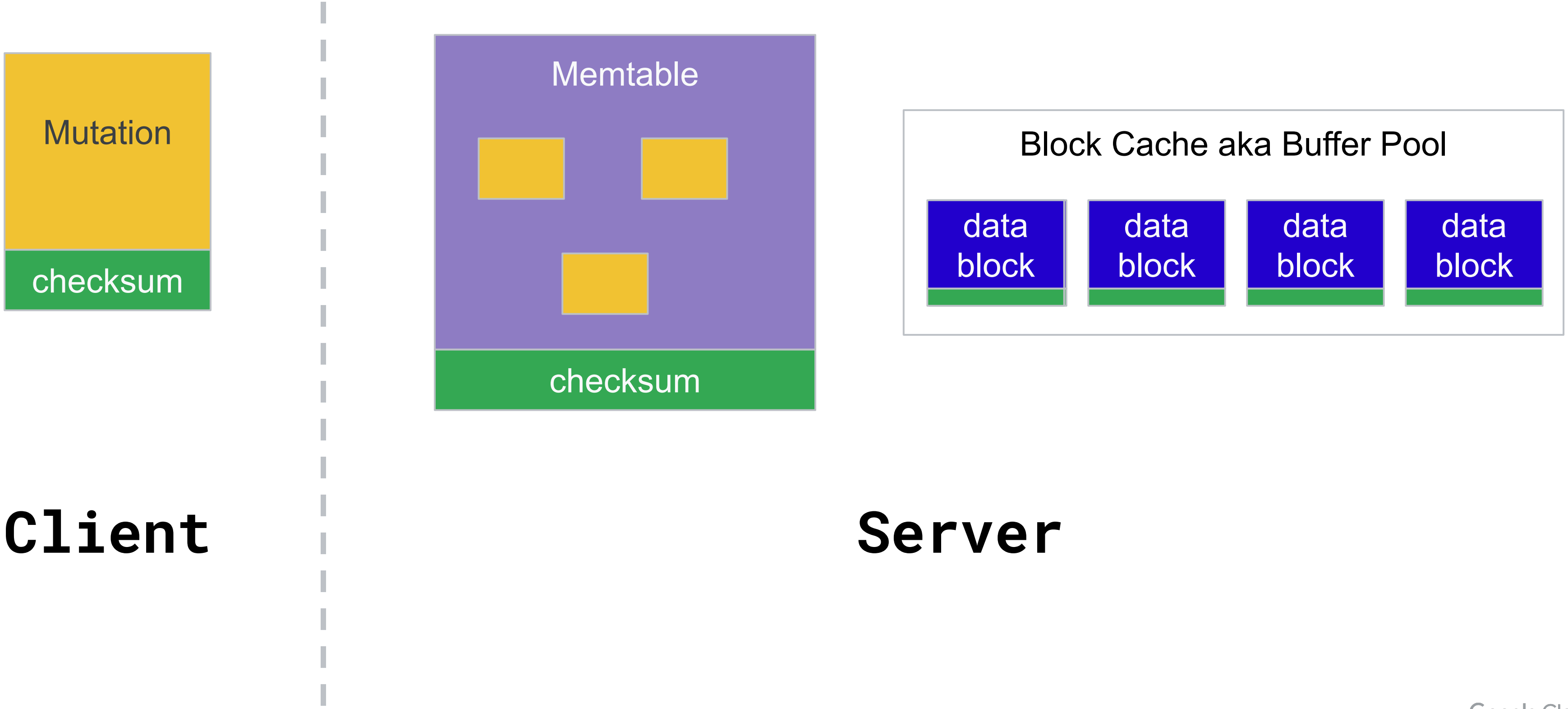
Triple Modular Redundancy

Reliable, “Simple”, Expensive, Incomplete

Hardware Data Protection?

- ARM Memory Tagging Extension (MTE)
  - Costly in RAM & CPU, inflexible
  - Improvements coming to make it practical; ETA ~3 years.
- Fine-grained access control (in space and time)
  - Intel MPK - relatively coarse, but (?) fast
- True capability systems
  - ARM CHERI
  - 128 bit pointers – too big or just natural evolution?
  - Doesn’t protect against many hardware errors

SDC Prevention: Redundant Data



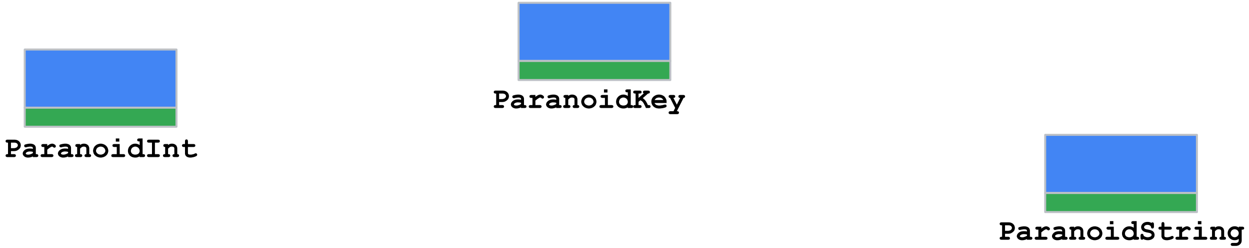
Checksumming CPU Costs

Component	Checksum Function	Spanner CPU
Block Cache	CRC32C	0.066%
Writes	Various	0.150%
Memtable	Incremental Xor	0.128%
Compaction	CRC32C	0.506%
Audit	MurmurHash	0.043%
Total	—	0.893%

Tinfoil Hat:

A Library for Detecting Corruption

ParanoidVariable

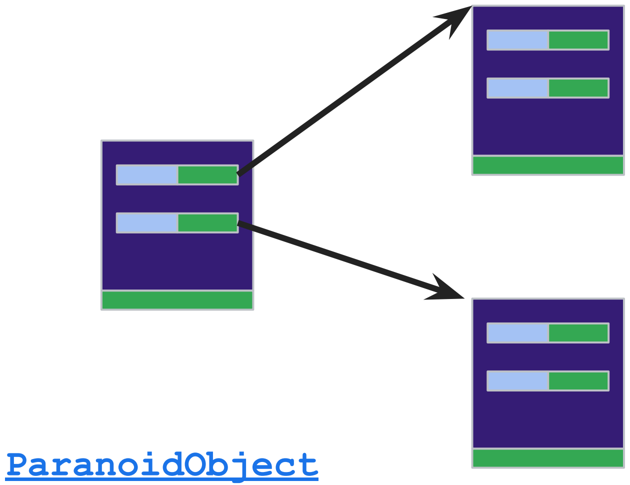


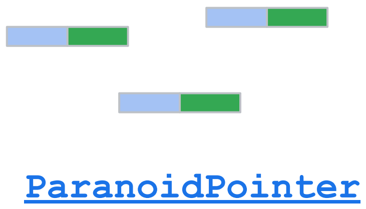
- A single value along with a (very cheap) checksum
- Checked on every access
- Used for “high blast radius” values (e.g. # records in file)

Tinfoil Hat pointer and unique_ptr

- C++ Smart pointers
- Verify type- and architecture-mandated zero bits
  - i.e. low bits and high bits didn't get stomped
- Usually only checked on destruction
- Cheap and widely deployable

ParanoidObject and ParanoidPointer





- ParanoidObject stores a hash of its immutable data
- ParanoidPointer stores some bits of that hash
- Can check on access or destruction

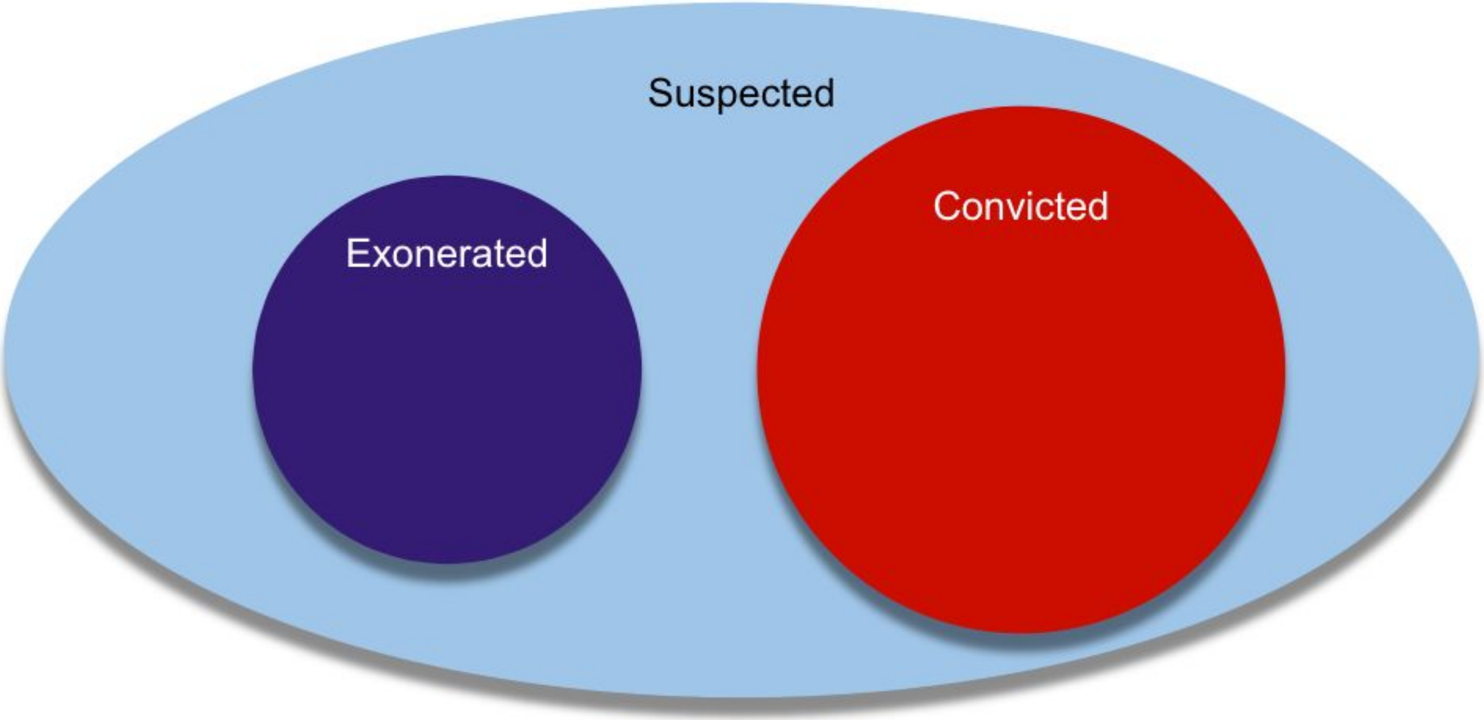
Tinfoil Hat Detectors Cost

Type	Checksum Type	Checksum Functor	Spanner CPU
`ParanoidInt<uint64>`	uint64	Xor with constant	$< 0.001\%$
ParanoidTimestamp	uint64	Xor with constant	$< 0.001\%$
ParanoidString	uint32	Scrambled CRC32C	$< 0.001\%$
ParanoidProto<CryptInfo>	uint64	Proto library checksum	0.0096%
ParanoidKey	uint64	Sum of bytes Xored with constant	0.0830%
Total	—	—	0.0936%

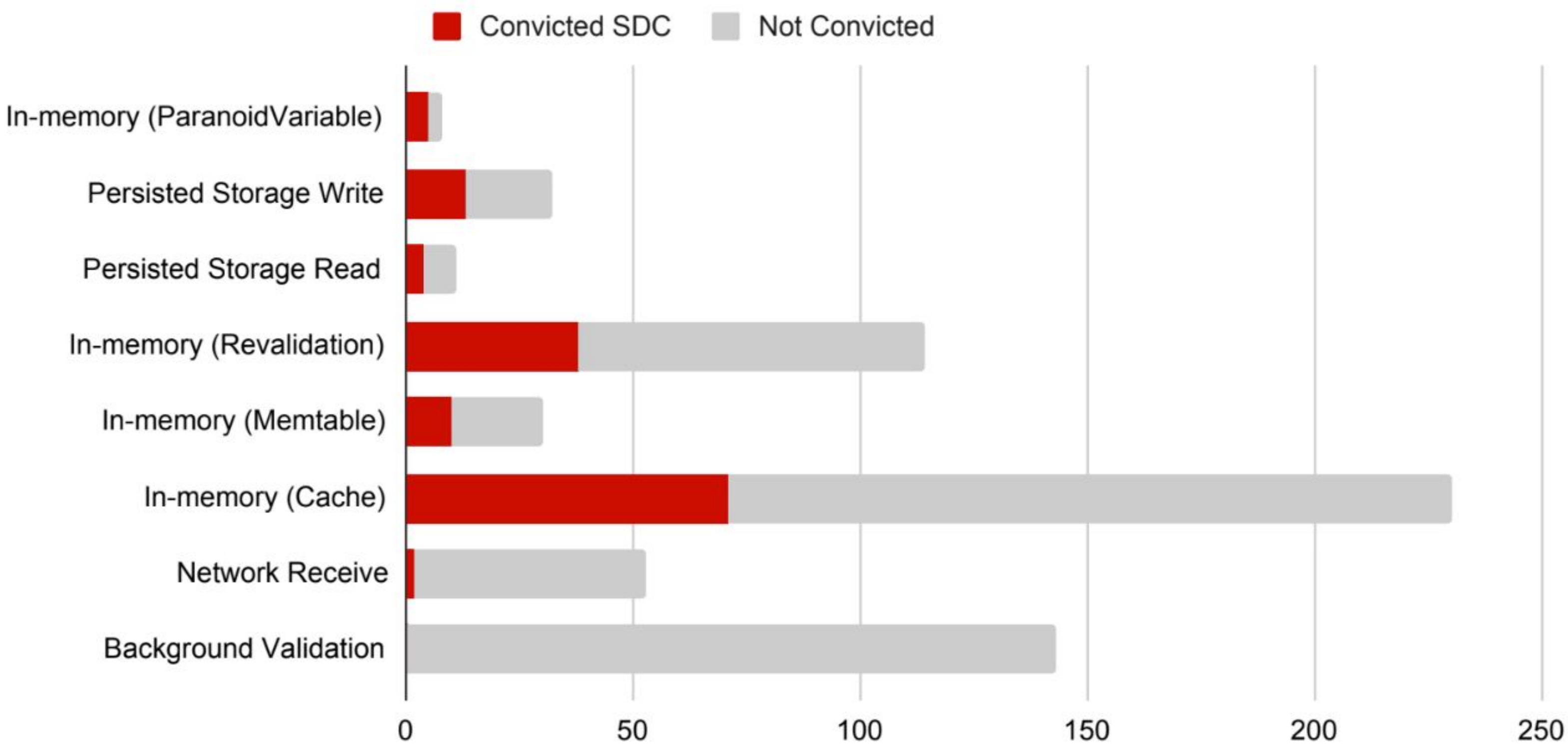
Type	Checksum	Spanner CPU
`ParanoidObject<CacheEntry>`	uint16	0.039%
ParanoidObject<SimpleArena>	uint8	0.011%
Total	—	0.050%

Measuring Corruption Detectors

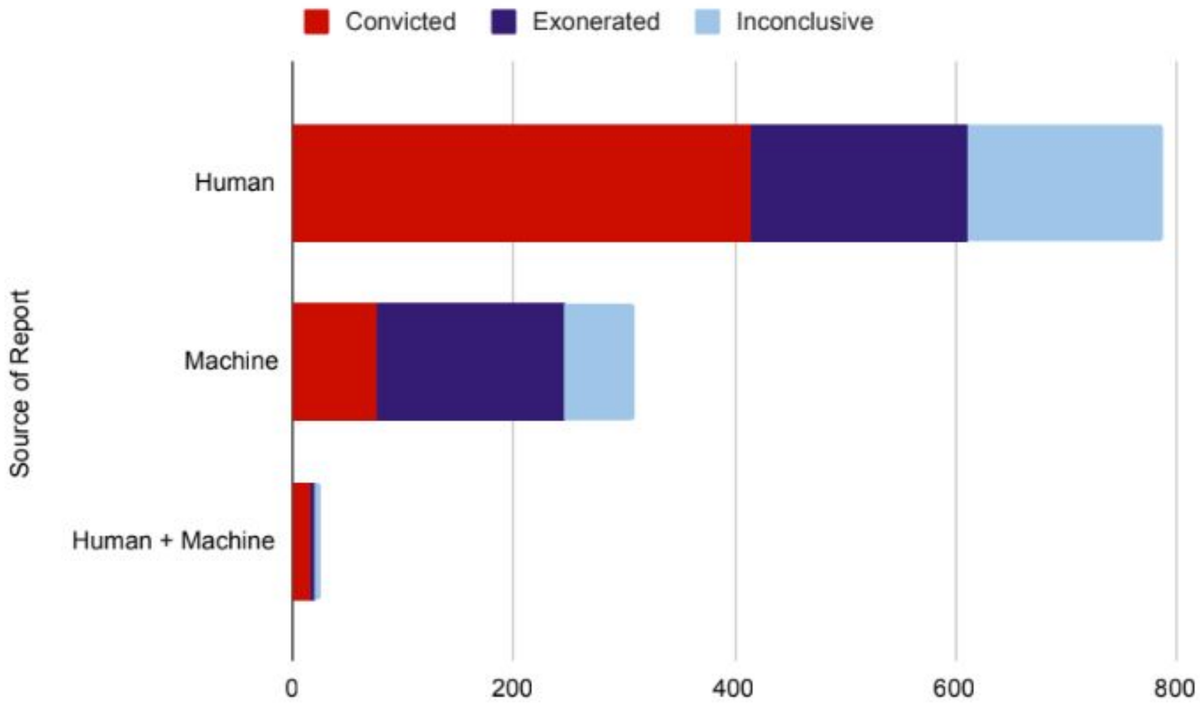
What Happens to a Detection Event?



Machine Conviction Rate by D



Conviction Rates: Human v. Machine



Conclusions

From Protecting the Data to Protecting the Fleet

- Can we make rare corruptions fail fast?
  - Leverage available redundancy across entire fleet
- Treat every byte of redundancy in every address space as an opportunity
- Turn the fleet into a big corruption sensor array

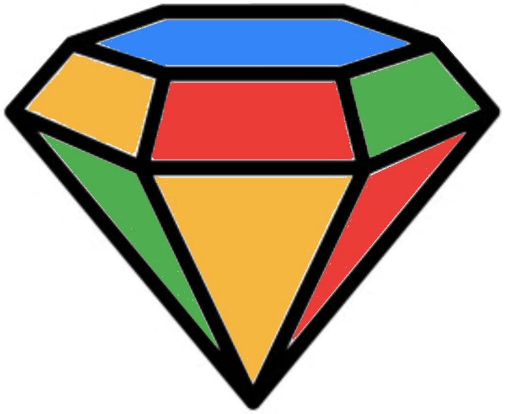
What's Next

- Insert profile-driven checks into production code
    - Tinfoil Hat checks
    - C++ Undefined Behavior (UBSAN) checks
- Based on these experiences, consider hardware support

Conclusions

- We must continue to improve reliability just to stand still
  - Software techniques are stopping the flood
- We plan for hardware SDC getting worse at each node
- Grand Challenge: Read-side corruptions
  - No good answers so far
  - No way to even measure it

Questions?



David F. Bacon
dfb@google.com