SOSP'15 History Day

Phil Gibbons

15-712 F15

Lecture 25

Today's Reminders

- **Today's Office Hours will be 5-5:30 pm**
  - Shortened due to conflict with CSD Faculty meeting
- **No class Friday & Monday**
  - Focus on projects & prep for midterm
- **Midterm 2 is next Wednesday**
  - Covers assigned readings from 10/28 to 11/23 + Anderson94 paper (9 papers in all)
  - Since fewer papers, a bit more in depth on each paper
- **Project Presentations 12/15 & 12/16**
  - Send email if want to meet r.e. projects before presentations
  - Final project report due midnight 12/16

SOSP History Day

October 4, 2015 — Monterey, California, USA

Introduction

Overview of the Day
Jeanna Matthews
SLIDES

The Founding of the SOSP Conferences
Jack Dennis
SLIDES

9am - 10:30am

Perspectives on OS Foundations
Peter Denning
ABSTRACT — SLIDES

Perspectives on Protection and Security
Butler Lampson
SLIDES

Perspectives on System Languages and Abstraction
Barbara Liskov
SLIDES

11am - 12:00pm





Evolution of File and Memory Management
Mahadev Satyanarayanan (Satya)
ABSTRACT — SLIDES

Evolution of Fault Tolerance
Ken Birman
ABSTRACT — SLIDES

1pm - 2:30pm

Virtualization
Andrew Herbert
PDF — SLIDES

Past and Future of Hardware and Architecture
Dave Patterson
ABSTRACT — SLIDES

Parallel Computing and the OS
Frans Kaashoek
ABSTRACT — SLIDES

3:00pm - 4pm

The Network and the OS
Dave Clark
SLIDES

The Rise of Cloud Computing Systems
Jeff Dean
ABSTRACT — SLIDES

4pm - 5pm: Panel session

Is achieving security a hopeless quest?
Margo Seltzer, Mark Miller, David Mazières, Yuanyuan Zhou





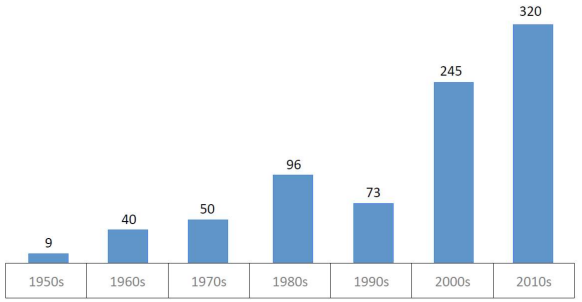


Note Concerning the Slides

- **These slides were obtained by copying and pasting from the respective speakers' original slides, and hence are copyrighted by those speakers.**
- **See http://www.sigops.org/sosp/sosp15/history/ for the complete set of slides for all speakers.**

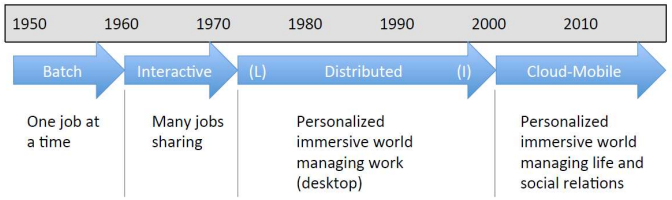
Perspectives on OS Foundations

Peter Denning

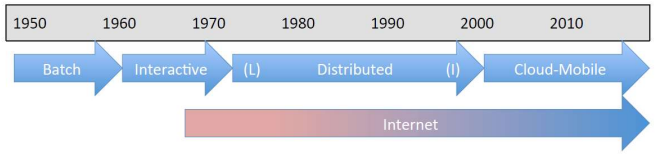


Number of new OS's per decade (Wikipedia)

Eras of Operating Systems



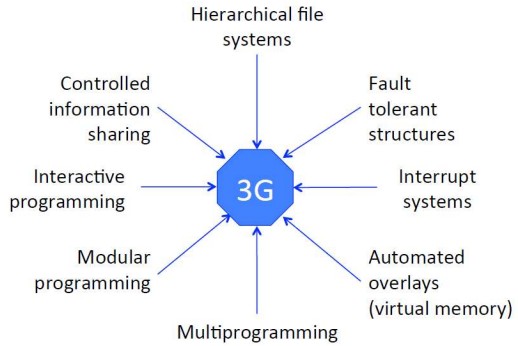
Eras of Operating Systems



"OS interfaces with":
TELNET
FTP
SMTP
RLOGIN
RCOPY

"OS integrates with":
Protocol software
IPC, RPC
Daemon processes
Client-server, X-windows
Hyperlink, URL
Browser
Search

Eight programming support objectives added by 1965 seeded the research



+HLLs for programming the OS

Two Cosmic Principles Revealed in Memory Management

Locality

Location independent addressing

Emerged from virtual memory research

Originally seeking to automate overlays

Locality Principle

Initial intuitions confirmed

All computations display locality (empirical)

All computations must display locality (theory)

Harnessing locality always pays off

- caching
- parallelizing
- performance
- no-thrashing

Location Independent Addressing

Key insights:

- Paging (U Manchester 1949)
- Virtual v. real address (ca 1959 Kilburn and Fotheringham)
- Segmentation in universal hierarchical address space (Dennis 1965)

From which flowed:

- Dynamic mapping virtual to real via page table
- Demand paging
- Replacement algorithms
- MMU and TLB mapping architecture
- Hierarchical naming systems

Huge benefits:

- Location independence
- Logical partitioning (address space isolation)
- Artificial contiguity
- Relocation
- Distributed naming authorities

Evolved into global, all-time unique addresses for digital objects anywhere in a system.

Hierarchical Internet URLs and domain names.

Now Internet is a huge virtual address space of capabilities, URLs, and DOIs with mapping via DNS and handle-servers.

Perspectives on Security
Butler Lampson

How did we get here?

- In the beginning, security was by physical isolation (1950-1963)
  - **Easy**: You bring your data, control the machine, take everything away
  - Still do this today with VMs and crypto (+ enclaves if VM host is untrusted)
- Timesharing brought the basic dilemma of security: (1963-1982)

Isolation vs. sharing

  - **Hard**: Each user wants a private machine, isolated from others
  - but users want to share data, programs and resources
- Since then, things have steadily gotten worse (1982-2015)
  - Less isolation, more sharing, no central management
  - More valuable stuff in the computers
  - Continued misguided search for perfection (following the NSA's lead)

Wisdom

- If you want security, you must be prepared for inconvenience.
  —General B.W. Chidlaw, 12 December 1954
- When it comes to security, a change is unlikely to be an improvement.
  —Doug McIlroy, ~1988
- The price of reliability is the pursuit of the utmost simplicity.
  It is a price which the very rich find most hard to pay.
  —Tony Hoare, 1980 (cf. Matthew 19:24)
- But who will watch the watchers? She'll begin with them and buy their silence.
  —Juvenal, sixth satire, ~100

What we know how to do

- Secure something simple very well
- Protect complexity by isolation and sanitization
- Stage security theatre

What we don't know how to do

- Make something complex secure
- Make something big secure if it's not isolated
- Keep something secure when it changes
- Get users to make judgments about security
- Understand privacy—fortunately not an SOSP topic

Themes

- **Goals**: Secrecy (confidentiality), integrity, availability (CIA: Ware 1970)
- **Gold standard**: Authentication, authorization, auditing (S&S 1975)
- **Principals**: People, machines, programs, … (Dennis 1966, DEC 1991)
- **Groups/roles**: make policy manageable (Multics 1968, NIST 1992)

Oppositions

(*in deployment, not good vs. bad*)

Winner		*Loser*
Convenience	vs.	Security
Sharing	vs.	Isolation
Bug fixes	vs.	Correctness
Policy/mechanisms	vs.	Assurance
Access control	vs.	Information flow

Timeline

	Themes	Systems
1960s	**Timesharing**; ACLs; access control matrix; VMs; passwords; capabilities; domains; gates	CTSS; Multics; CP/CMS; Cal TSS; Adept-50; Plessey 250
1970s	**TS**; LANs/Internet (e/e security); public key; multi-level sec.; ADTs/objects; least privilege; Trojans; isolation by crypto; amplification; undecidability	Unix; VMS; VM/370; IBM RACF; Clu; Hydra; Cambridge CAP
1980s	**Workstations; client/server**; Orange Book; global authentication; Clark and Wilson	A1 VMS; SecureID; Morris worm; IX
1990s	**PCs; Web**; sandboxes; Java security; crypto export; decentralized information flow; Common Criteria; biometrics; RBAC; BAN; SFI; SET	Browsers; SSL; NT; Linux; PGP; Taos
2000s	**Web; JavaScript**; buffer overflows; DDoS	TPM; LSM; SELinux; seL4; HiStar
2010s	**Web; big data**; enclaves; homomorphic crypto	Singularity; CryptDB; Ironclad ...

Does it actually work? Assurance (Correctness)

- Keep it simple—Trusted Computing Base (TCB) (Rushby 1981)
  - One way is a security kernel: apps are not in the TCB. Works for sharing hardware
- Ideally, you **verify**: prove that a system satisfies its security spec
  - This means that *every* behavior of the system is allowed by the spec
    - Not the same as proving that it does everything in the manual
  - Today in seL4, Ironclad, … First tried in Gypsy (late 1970s)
  - What if the spec is wrong? Keep it simple
- Usually verifying is too hard, so you **certify** instead
  - Through some "independent" agency. Alas, process trumps substance
    - First by DoD for Orange Book, later international Common Criteria (1985, 1999)
- Or you can verify **some** properties: isolation, memory/type safety
- Or you can apply bandaids

Bandaids for Bugs (Defense in Depth)

- No guarantees, but at least the bad guy has to work harder
  - **Firewalls** to keep intruders out, look for suspicious traffic (DEC 1988)
  - **Signature** hacks to detect malware (~1990)
  - **Memory safety** hacks to catch writes outside array bounds (Phrack 1996)
  - **Intrusion detection** hacks to look for anomalous behavior (SRI 1986)
  - **Control Flow Integrity** to block jumps not in the normal flow (MSR 2005)
  - **Taint tracking** to keep unsanitized input away from execution (CMU 2005)
  - **Process** to enforce use of the tools (MS SDL 2004)
- "I don't have to outrun the bear; I just have to outrun you."
  - These are not bad things, but they are hacks

What has worked? What hasn't?

Worked ~ gotten wide adoption

Worked	Failed
VMs	"Secure systems"
SSL	Capabilities (except short term)
Passwords	Metrics for security
Safe languages	MLS/Orange book
Firewalls	User education
Process—SDL	Intrusion detection

Why don't we have "real" security?

- **A**. People don't buy it
  - Danger is small, so it's OK to buy features instead
  - Security is expensive
    - Configuring security is a lot of work
    - Secure systems do less because they're older
  - Security is a pain
    - It stops you from doing things
    - Users have to authenticate themselves
  - Goals are unrealistic, ignoring technical feasibility and user behavior
- **B**. Systems are complicated, so they have bugs
  - Especially the configuration

What next?

- Lower aspirations. In the real world, good security is a bank vault
  - Hardly any computer systems have anything like this
  - We only know how to make simple things secure
- Access control doesn't work—40 years of experience says so
  - Basic problem: its job is to say "No"
    - This stops people from doing their work, and then they relax the access control
    - usually too much, but no one notices until there's a disaster
- Retroactive security: focus on things that actually happened
  - rather than all the many things that *might* happen
  - Real world security is retroactive
    - Burglars are stopped by fear of **jail**, not by locks
    - The financial system's security depends on **undo**, not on vaults

Biertan fortified church, Romania

Jail

Lock

Memory and File Systems

M. Satyanarayanan

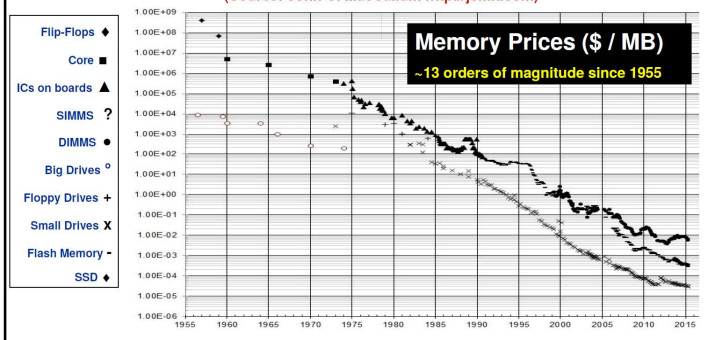
Four Drivers of Progress

The quest for ***scale***	*from early 1950s*
The quest for ***speed***	*from early 1950s*
The quest for ***transparency***	*from early-1960s*
The quest for ***robustness*** (both system and human errors)	*from mid- to late-1960s*

Complex Interactions

Cost of Memory & Storage

(Source: John C. MacCallum http://jcmit.com)



Naming and Addressability

Consistently too few bits in addressing (12-bit, 16-bit, 18-bit, 32-bit, …)
re-learned in DOS/ Win3.1 (memory extenders); hopefully 64 bits will last us a while

Semantic addressing
hierarchical name spaces, SQL, search engines

Content Addressable Storage (aka deduplication)
Venti (late 1990s), LBFS (early 2000s), many others since,
continuing concerns regarding collisions (Val Henson)

Capability-based

- **short term (seconds, minutes, hours lifetime)**
  can be viewed as a form of caching expensive/cumbersome access checks
- **long term (infinite life)**
  Hydra on C.mmp (mid 1970s) pushed this concept to the limit
  Intel iAPX 432 (3 papers in SOSP 1981!)

Transparency

"Indistinguishable from original abstraction"

- *no application changes:* programs behave as expected
- *no unpleasant surprises for users:* good user experience
- importance increases as hardware to human cost ratio shifts

Hugely important in industry, less important in academic research

Achieved by interposing new functionality at widely-used interfaces

- memory abstraction (hardware caches)
- POSIX distributed file systems
- x86 virtual machines

A Brief History of Caching

Demand paging was first known use of caching idea *(1961)*

John Fotheringham, CACM, 1961, pp 435-436

Dynamic Storage Allocation in the Atlas Computer, Including an Automatic Use of a Backing Store*

John Fotheringham

Ferranti Electric, Inc., Plainview, New York

1. Introduction

This paper is concerned with the method of address interpretation in the Atlas computer. The Atlas has been designed and built as a joint exercise between the Computer Departments of Manchester University and of Ferranti Ltd., and the ideas and concepts described in this

characters within a word for certain special functions, and the leading digit of these three is also used for identifying the half-word operand for 24-bit functions such as index register operations. The remaining 21 bits address a word, giving a range of over two million words; this range is divided into halves of which the first half is the main

Hardware caches *(1968)*

"Structural Aspects of the System/360 Model 85, Part II: The Cache,"
J. S. Liptay, IBM Systems Journal, Vol. 7, No. 1, 1968

Distributed file systems *(~1983)*

- *AFS, NFS, Sprite, Coda*

Web caching *(mid 1990s)*

- *SQUID, Akamai (CDNs)*

Virtual machine state caching *(early 2000s)*

- *Internet Suspend/Resume, Collective, Olive*

Key-Value caches *(mid 2000s)*

- REDIS

The Importance of Demand Fetch

Assumes ability to detect read operations
- ability to detect cache misses
- ability to interpose cache logic
- result is *total transparency*

In a file system this requires OS support
- distributed file systems (e.g AFS, Coda, …)
- FUSE interface

Systems like DropBox cannot do this
- lack of OS support simplifies implementation
- improves portability of code across OSes
- DropBox needs complete replicas everywhere
  (aka "sync solution")

Without OS intercept

1. Even viewing one small file requires whole replica
2. Every update has to be propagated everywhere

Coping With Human Error

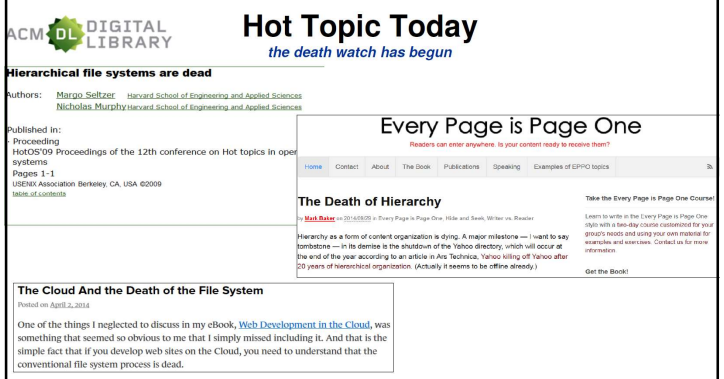
Use of separate address spaces (threads vs. processes)

Easy retrospection of file systems by users
- periodic read-only snapshots (AFS)
- Apple Time Machine, Elephant File System, …

Why is memory distinct from file system?

Single level stores have been proposed in the past
- but separation offers enhanced robustness
- well-formed open / read / write / close unlikely to be accidental
- contrast with wild memory write

Are Classic File Systems Dead?



Why are File Systems Hierarchical?

Ken Thompson made radical changes in creating Unix
- why was the Unix file system so conventional and hierarchical?
- mere sentiment? lack of imagination?

"The Architecture of Complexity"
Herbert A. Simon, *Proceedings of the American Philosophical Society*, Vol. 106, No. 6., Dec. 12, 1962, pp. 467-482.

"Empirically, a large proportion of the complex systems we observe in nature exhibit hierarchic structure. On theoretical grounds we could expect complex systems to be hierarchies in a world in which complexity had to evolve from simplicity. In their dynamics, hierarchies have a property, ***near-decomposability,*** *that greatly simplifies their behavior."*

How Hierarchy Helps

Hierarchical file systems conflate search and access

- well-matched to limitations of human cognition,
- locality is an emergent property (temporal and spatial)
- locality is precious performance-wise for direct human exploration of data

Retrospective use of old unstructured data (e.g., decades later) →

- even the features for indexing may be unclear
- manual exploration may be necessary

Need for manual exploration (even if rare) →

- hierarchical file systems will not disappear
- but the hierarchical nature may remain deeply buried

Fault Tolerance
Ken Birman

Lorenzo Alvisi's Byzantine twin wants you to use 2f+1 replicas

Too many seminal concepts

- Process pairs, primary-backup
- 2PC and 3PC, Quorums
- Atomic Transactions
- State machine replication
- RAID storage solutions

- Checkpoints, Message Logging
- Byzantine Agreement
- Gossip protocols
- Virtual synchrony model
- Paxos
- Zookeeper

Theory

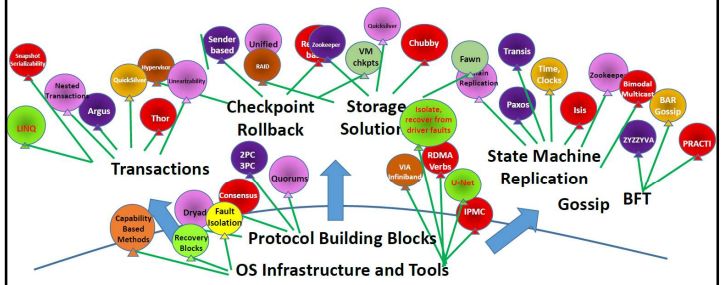
- Consensus
- FLP

◊W: consensus + oracle

… Skepticism

- CATOCS

CAP



- Too much for 25 minutes…
- Focus on state machine replication with crash failures

Principles from the theory side…

- FLP: Protocols strong enough to solve asynchronous consensus cannot guarantee liveness (progress under all conditions).
- If running a highly available database with network partition, conflicting transactions induce inconsistencies (CAP theorem).
- Need 3f+1 replicas to overcome Byzantine faults

Principles from the systems side…


Butler

- Make core elements as simple as possible
  - *Pare down, optimize the critical path*
  - Captures something fundamental about systems.
- Generalized End-to-End argument:
  - Let the application layer pick its own models.
  - Limit core systems to fast, flexible building blocks.

Jerry


Dave

B. Lampson. Hints for computer system design. *ACM Operating Systems Rev.* 1983.
J. Saltzer/D. Reed/D. Clark. End-To-End Arguments in System Design. 1984.

Tensions

Why aren't existing OS mechanisms adequate?

Is fault-tolerance / consistency too complex or costly?

Do the needed mechanisms enable or impose models?

Existing core OS support: Inadequate

- IP multicast just doesn't work…
  - Amazon AWS disables IPMC and tunnels over TCP
- TCP is the main option, but it has some issues:
  - No support for reliable transfer to multiple receivers
  - Uncoordinated model for breaking connections on failure
  - Byte stream model is mismatched to RDMA

… Real systems informed by sound theory

- Isis: Widely adopted during the 1995-2005 period
  - French ATC system, US Navy AEGIS, NYSE...
- Paxos: Very wide uptake 2005-now
  - Locking, file replication, HA databases…
  - Clean methodology and theory appeal to designers
  - Corfu is the purest Paxos solution: robust logging

… Cloud rebellion: "Just say no!"

Werner Vogels

- State Machine Replication, Paxos, ACID transactions — *Conceptual Tools*
- Chubby, Zookeeper, Corfu — *Real Systems*
- Primary + Warm backup… Chain Replication
- Dynamo: Eventual consistency (BASE), NoSQL KVS — ACID

Is consistency just too costly?

Eric Brewer

- CAP: Two of {**C**onsistency, **A**vailability, **P**artition-Tolerance}
  - Widely cited by systems that cache or replicate data
  - Relaxed consistency eliminates blocking on the critical path
  - CAP theorem: proved for a WAN partition of an H/A database
- BASE (eBay, Amazon)
  - Start with a transactional design, but then weaken atomicity
  - Eventually sense inconsistencies and repair them

… but does CAP+BASE work?

- ~~CAP folk theorem: *"don't even try to achieve consistency*~~."

- CAP + BASE are successful *for a reason:*
  - In the applications that dominate today's cloud, stale cache reads have negative utility but don't cause safety violations.
  - In effect a *redefinition*, not a rejection, of consistency

Parallelism and Operating Systems

Frans Kaashoek

Three types of parallelism in operating systems

1. **User parallelism**
   - Users working concurrently with computer
2. **I/O concurrency**
   - Overlap computation with I/O to keep a processor busy
3. **Multiprocessors parallelism**
   - Exploit several processors to speedup tasks

The first two may involve only **1** processor

This talk: 4 phases in OS parallelism

Phases	Period	Focus
Time sharing	60s/70s	Introduction of many ideas for parallelism
Client/server	80s/90s	I/O concurrency inside servers
SMPs	90s/2000s	Multiprocessor kernels and servers
Multicore	2005s-now	All software parallel

Phases represent major changes in commodity hardware

The debate: events versus threads

Handle I/O concurrency with event handlers
- Simple: no races, etc.
- Fast: No extra stacks, no locks

High-performance Web servers use events
Javascript uses events

The response: Why Events Are A Bad Idea [HotOS IX]
- Must break up long-running code paths
- "Stack ripping"
- No support for multiprocessor parallelism

Why Threads Are A Bad Idea (for most purposes)

John Ousterhout
Sun Microsystems Laboratories

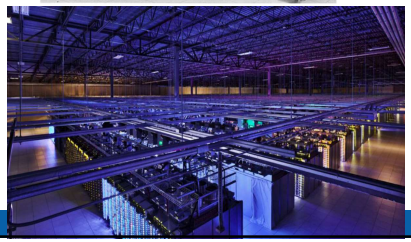
[Keynote at USENIX 1995]

Should You Abandon Threads?

- **No: important for high-end servers (e.g. databases).**
- **But, avoid threads wherever possible:**
  - Use events, not threads, for GUIs, distributed systems, low-end servers.
  - Only use threads where true CPU concurrency is needed.
  - Where threads needed, isolate usage in threaded application kernel: keep most of code single-threaded.

The Rise of Cloud Computing Systems

Jeff Dean



Typical first year for a new Google cluster (circa 2006)

~1 **network rewiring** (rolling ~5% of machines down over 2-day span)
~20 **rack failures** (40-80 machines instantly disappear, 1-6 hours to get back)
~5 **racks go wonky** (40-80 machines see 50% packetloss)
~8 **network maintenances** (4 might cause ~30-min random connectivity losses)
~12 **router reloads** (takes out DNS and external vips for a couple minutes)
~3 **router failures** (have to immediately pull traffic for an hour)
~**dozens of minor 30-second blips** for DNS
~1000 **individual machine failures**
~**thousands of hard drive failures**
slow disks, bad memory, misconfigured machines, flaky machines, etc.
Long distance links: **wild dogs, sharks, dead horses, drunken hunters, etc.**

Reliability Must Come From Software

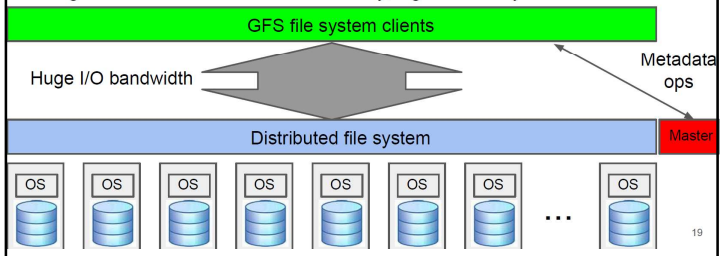
A Series of Steps,
All With Common Theme:

Provide Higher-Level View Than
"Large Collection of Individual Machines"

Self-manage and self-repair as much as possible

Google File System (Ghemawat, Gobioff, & Leung, SOSP'03)

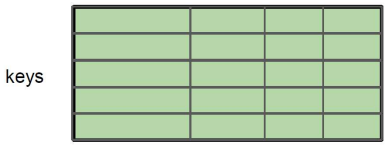
- Centralized master manages metadata
- 1000s of clients read/write directly to/from 1000s of disk serving processes
- Files chunks of 64 MB, each replicated on 3 different servers
- High fault tolerance + automatic recovery, high availability



Successful design pattern:

Centralized master for metadata/control, with thousands of workers and thousands of clients

Many Applications Need To Update Structured State
With Low-Latency and Large Scale



TBs to 100s of PBs of data

10^6, 10^8, or more reqs/sec

Desires:

- Spread across many machines, **grow and shrink** automatically
- **Handle machine failures** quickly and transparently
- Often prefer **low latency and high performance** over consistency

Distributed Semi-Structured Storage Systems

- BigTable [Google: Chang *et al.* OSDI 2006]
  - higher-level storage system built on top of distributed file system (GFS)
  - data model: rows, columns, timestamps
  - no cross-row consistency guarantees
  - state managed in small pieces (tablets)
  - recovery fast (10s or 100s of machines each recover state of one tablet)
- Dynamo [Amazon: DeCandia *et al.*, 2007]
  - versioning + app-assisted conflict resolution
- Spanner [Google: Corbett *et al.*, 2012]
  - wide-area distribution, supports both strong and weak consistency

Successful design pattern:

Give each machine hundreds or thousands of units of work or state

Helps with:
dynamic capacity sizing
load balancing
faster failure recovery