

Fifty Years of Operating Systems

SIGOPS SOSP History Day

October 4, 2015

The Story

Operating systems are a major enterprise within computing. They are hosted on over a billion devices connected to the Internet. They were a \$33 billion global market in 2014. The number of distinct new operating systems each decade is growing, from 9 introduced in the 1950s to an estimated 350 introduced in the 2010s.

Operating systems became the subject of productive research in late 1950s. In 1967 operating systems research leaders organized the SOSP (symposium on operating systems principles), starting a tradition of bi-annual SOSP conferences that has continued 50 years. The early identification of operating system principles crystallized support in 1971 for operating systems to become part of the computer science core curriculum.

In October 2015, as part of SOSP-25, we celebrated 50 years of OS history. Ten speakers and a panel discussed the evolution of major segments of OS, focusing on the key insights that were eventually refined into cherished OS principles (sigops.org/sosp/sosp15/history). This is the introduction to the video record.

--- Peter J. Denning, Organizer

The Speakers

Overview of the Day

After a short summary of how the SOSP series began in 1967, EmCee Jeanna Matthews introduces the speakers. She has photos of them in their younger days when they were inventing OS principles.



Jeanna Matthews

The Founding of OS conferences



Jack Dennis

Jack Dennis launched the SOSP series in 1967. He saw an opportunity to bring out the emerging principles of operating systems and communication networks.

The Speakers

OS Foundations

Peter Denning shows the evolution of OS's from batch systems and then to time-sharing, distributed, and mobile-cloud systems. A body of significant principles evolved over time, including two of major focus in his own research, location independent addressing and locality.



Peter Denning

Protection and Security



Butler Lampson

Butler Lampson traces a long history of protection mechanisms in spite of which security remains a major problem. He considers isolation, access control, access policy, information flow control, cryptography, trust, and assurance. In the end, people dislike the inconvenience security causes.

The Speakers

System Languages and Abstraction

Barbara Liskov examines the evolution of abstractions, such as processes and software layers, to organize complex systems. Some abstractions are separate service processes invoked by RPC, others are overlaid on a user's process by monitors. Many have found their way into system programming languages. Communication is a major issue.



Barbara Liskov

File and Memory Management



Mahadev Satyanarayanan
(Satya)

Satya highlights three major themes in the search for better memory systems. Scale sought larger and faster memories that kept up with ever faster processors. Transparency hid complex physical structures behind a simple address space. Fault tolerance made memories robust with partitioning and replication. Hierarchical file systems are deeply embedded into the Internet and will not soon disappear.

The Speakers

Fault Tolerance

Ken Birman examines fault tolerance, a system's resistance to failure of memory hardware, and consistency, the system's ability to correctly reconstruct lost data from multiple copies. Enormous progress with these issues enabled the modern cloud to scale reliably to huge sizes.



Ken Birman

Virtualization



Andrew Herbert

Andrew Herbert traces the history of the OS structuring principle of virtual machines. This principle appears in virtual machine monitors, desktop hosted virtual machines, layered abstract machines, and a standard environment for program execution in distributed Unix.

The Speakers

Hardware and Architecture

Dave Patterson shows that the old debates about RISC versus CISC architecture for processors and NUMA versus clusters for parallel processing are mostly settled. Moore's law is nearly spent. For continued improvements, we look now to new memory architectures, open source instruction sets, and custom chips: a new hardware-software partnership.



Dave Patterson

Parallel Computing and the OS



Frans Kaashoek

Frans Kaashoek traces the history of parallel programming in OS, starting with parallel I/O and CPUs, then distributed systems, and then systems with multicore processors. Because software must be parallel to exploit multicore processors, the OS community is going through a rebirth of research in parallel computing.

The Speakers

The Network and the OS

Dave Clark digs through his long experience in getting network protocols (notably TCP/IP) to work efficiently with the OS. It was a long hard slog to gain deep understanding of the efficiency of each little part of the protocol software. Eventually the protocols were successfully integrated and today's OS all include the network.



Dave Clark

The Rise of Cloud Computing Systems



Jeff Dean

Jeff Dean shows that older approaches to parallel processing such as transaction processing and high-performance computing emphasized performance but did not scale well. When fault tolerance was emphasized instead, parallel systems scaled to tens of thousands of processors and millions of users without loss of performance, realizing the old dream of a computing utility available cheaply to everyone.

The Speakers

Panel: Is Achieving Security a Hopeless Quest?

Despite all the work in OS to provide protection and improve security, cyber crime has grown into a major social issue.

There seem to be no solutions to loss of data and theft of identity. Does the OS community bear a responsibility for this mess?



Margo Seltzer



Mark Miller



David Mazieres



YY Zhou

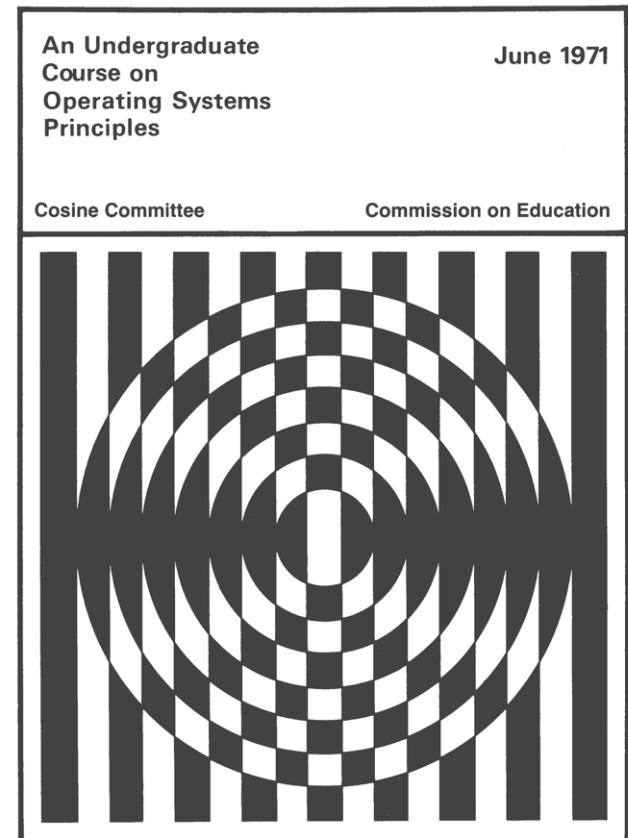
SOSP History Day Workshop

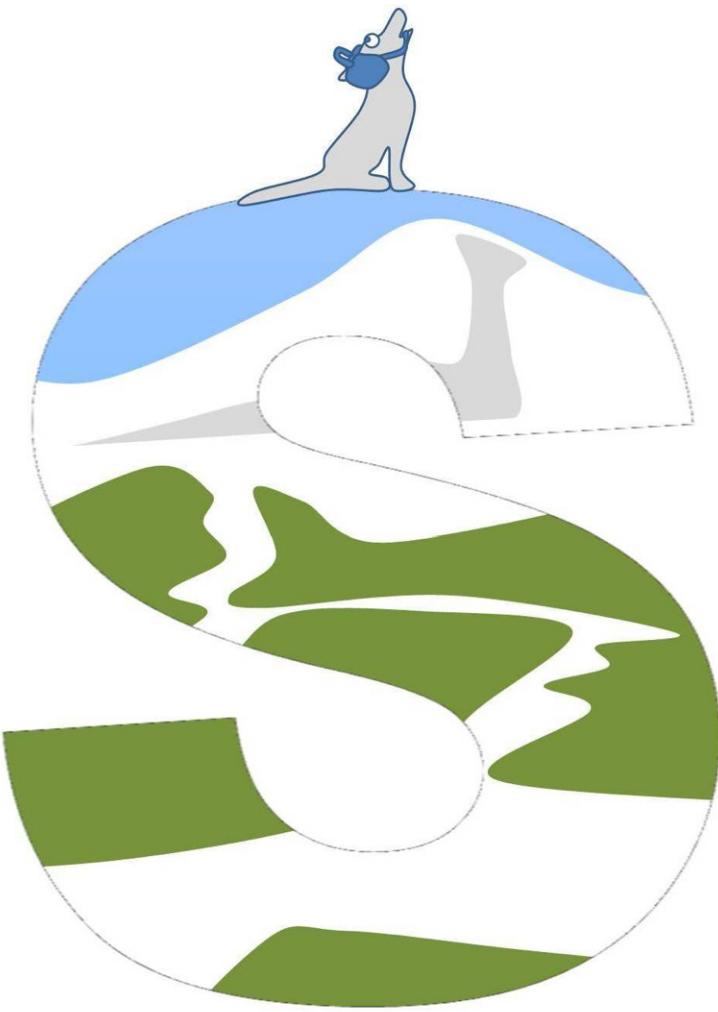
Jeanna Neefe Matthews

October 4 2015

Birth of SOSP and SIGOPS

- SOSP 2015 is SOSP-25
- Special Interest Committee on Time-Sharing (SICTIME) founded by Henriette Avram, 1965
- SOSP-1 (Gatlinburg, TN) 1967
 - SICTIME/SICCOMM sponsors
 - Special Interest Committee vs Group
- SICTIME convert to SIGOPS 1968/1969
- SOSP-2 (Princeton, NJ) 1969
- SOSP at Asilomar 3 times
 - SOSP-7 (1979), SOSP-8 (1981), SOSP-13 (1991),

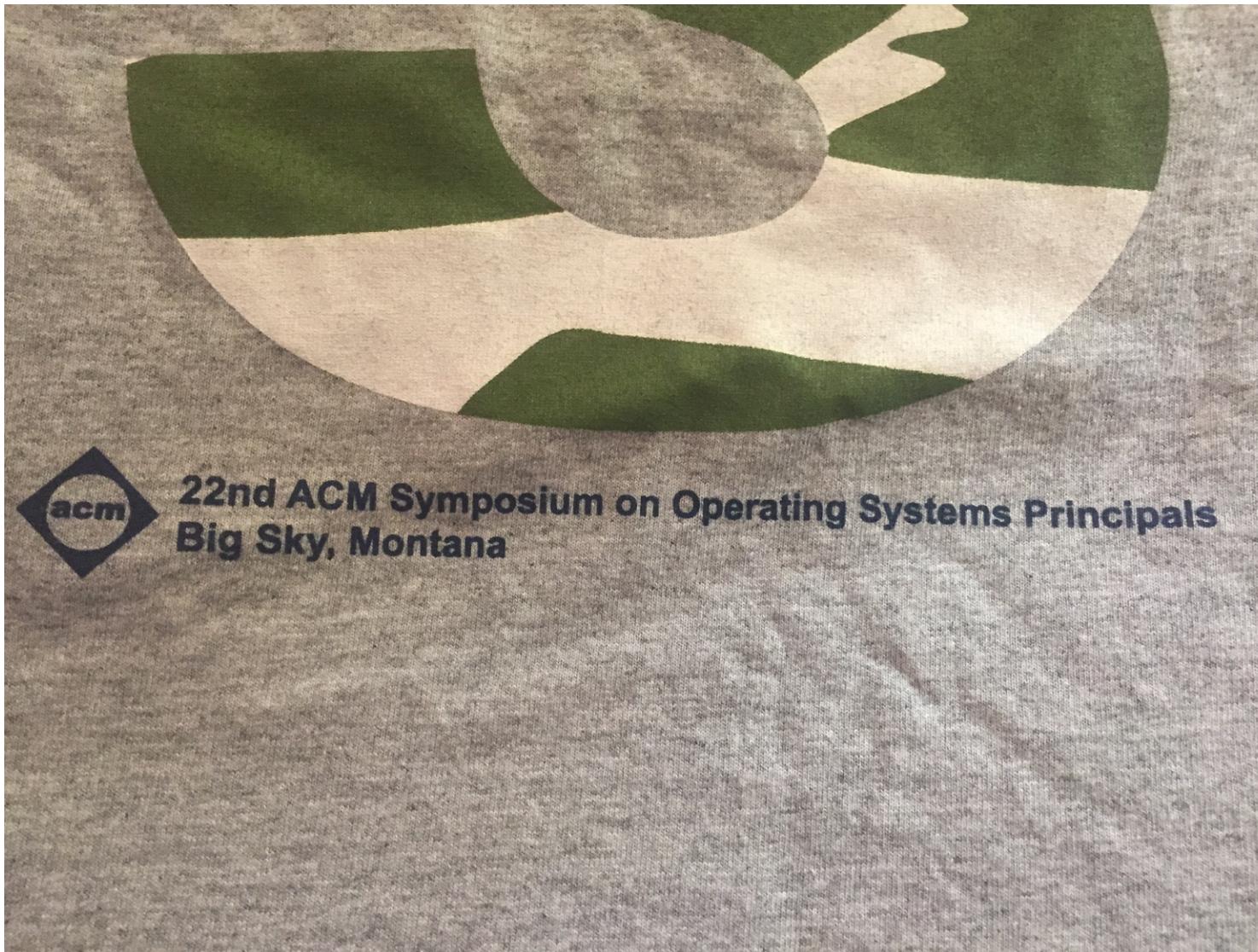




22nd ACM Symposium on Operating Systems Principles
Big Sky Resort, Montana

Andy Wang from Florida State

Principles and Principles



Jack Dennis



Research group in the 1970s

Peter Denning



CSNET executive committee at 1981 meeting in Boulder, CO



Butler Lampson



UC Berkeley Turing Award winners:
Ken Thompson, Butler Lampson, Jim
Gray and Nicklaus Wirth

Barbara Liskov



Barbara Liskov and Jack Dennis from around 1974

M. Satyanarayanan (Satya)



PhD graduation in May 1983

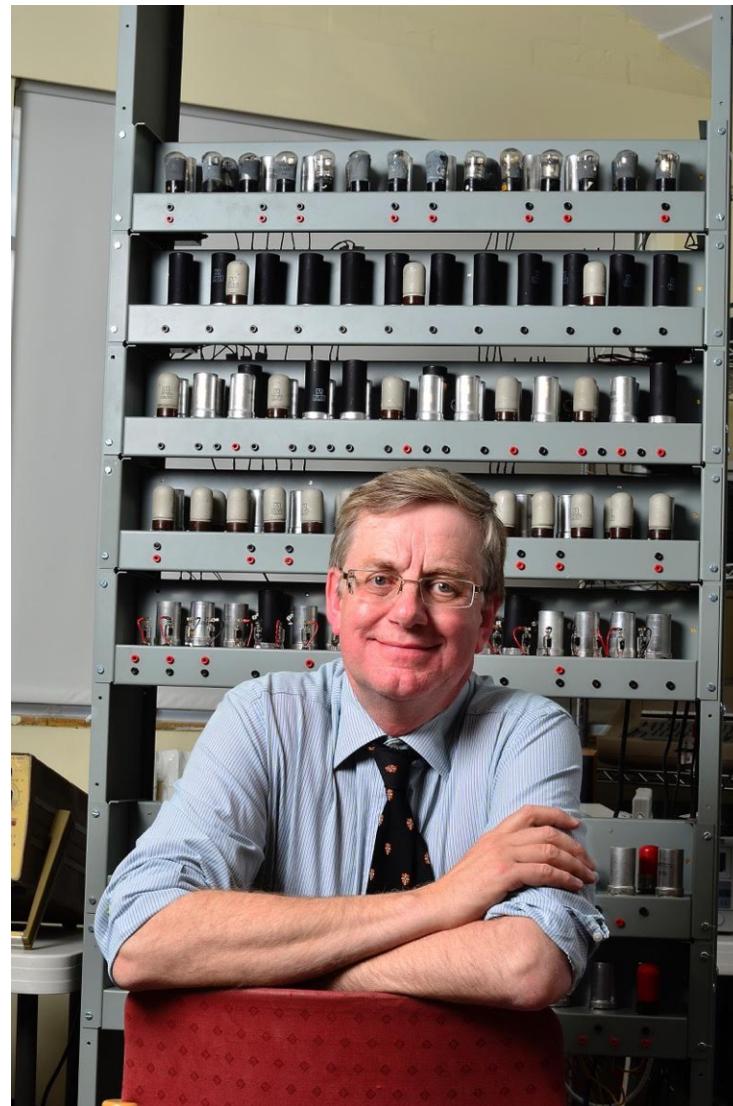


Ken Birman



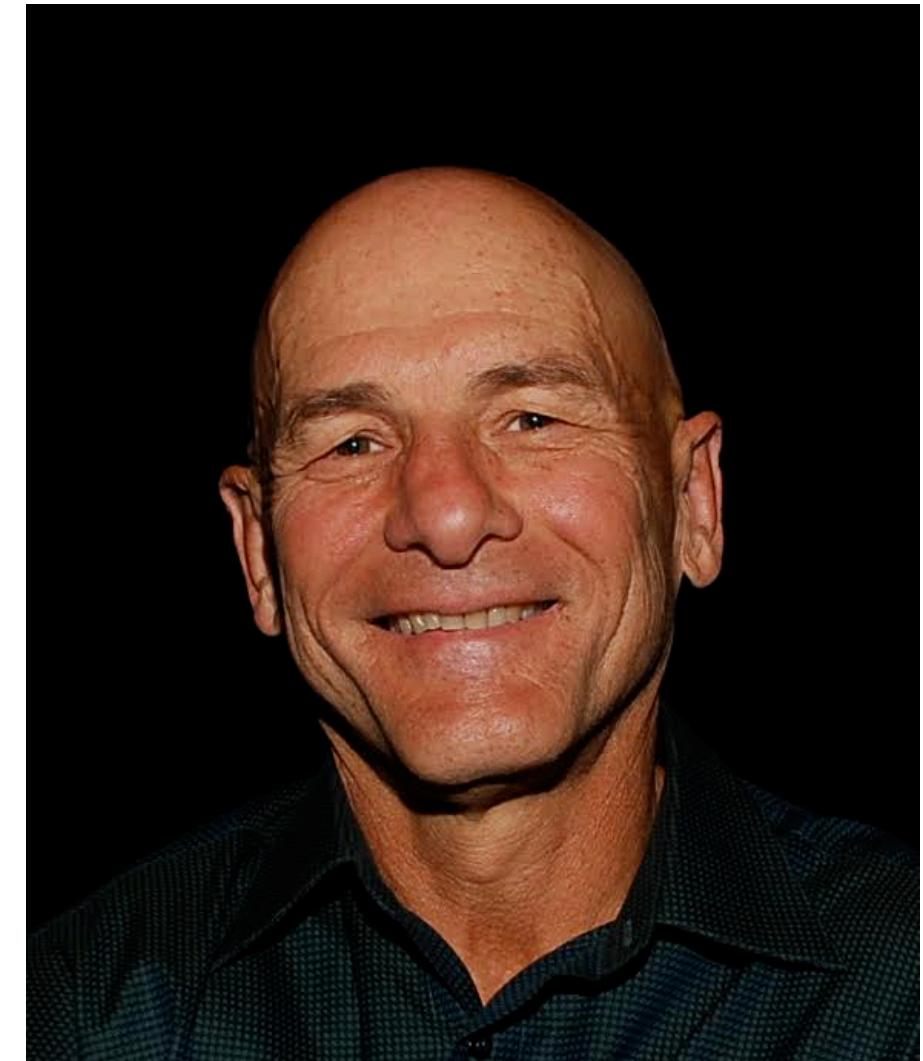
Ken Birman with Werner Vogels and
Robbert van Renesse (2004)

Andrew Herbert



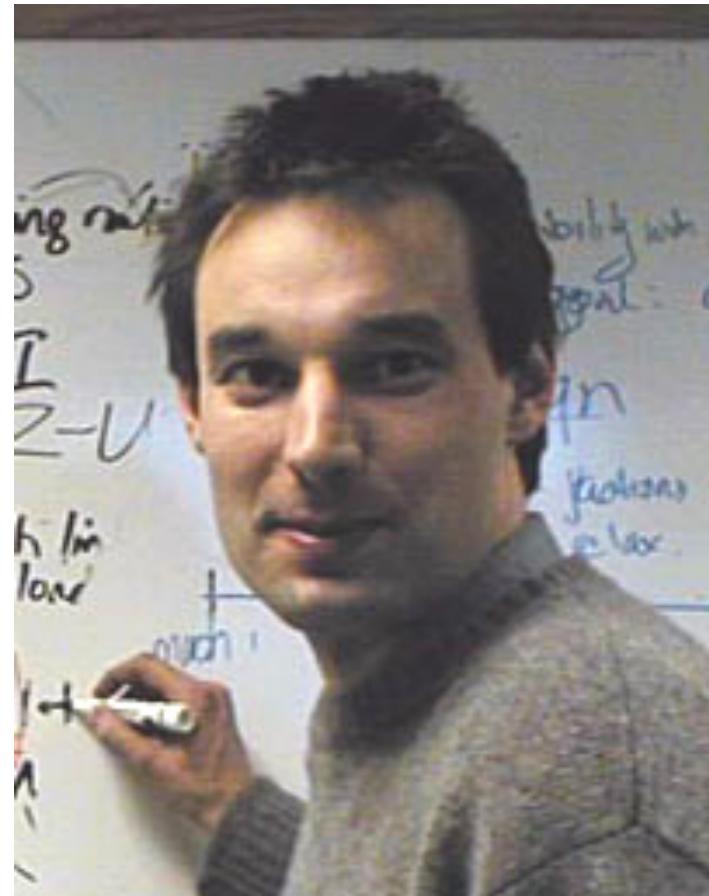
Building a replica of the Cambridge University 1469 EDSAC Computer

Dave Patterson

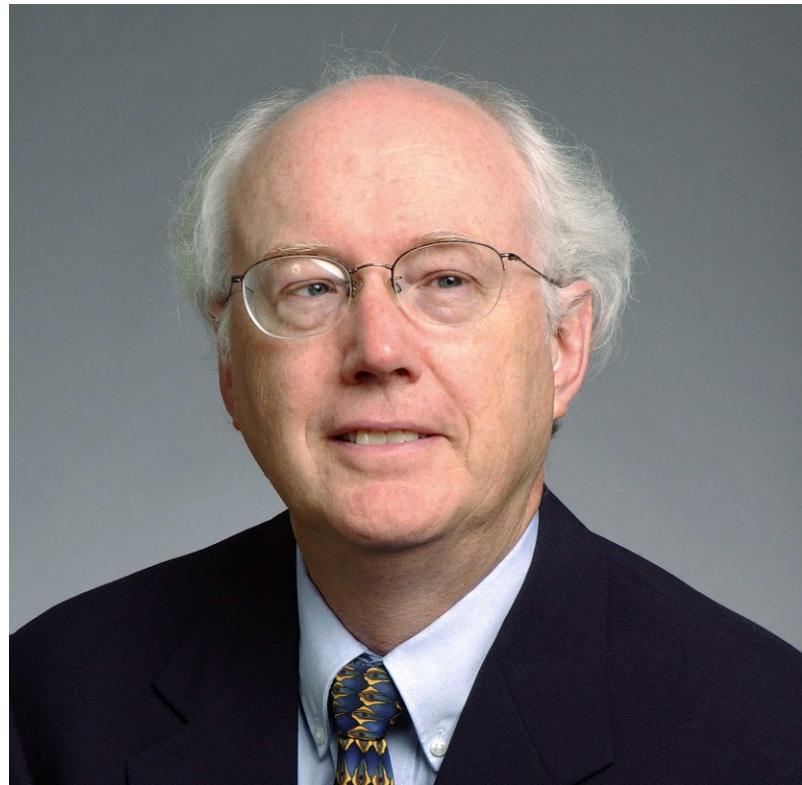


NOW Group Research
Retreat, 1995

Frans Kaashoek

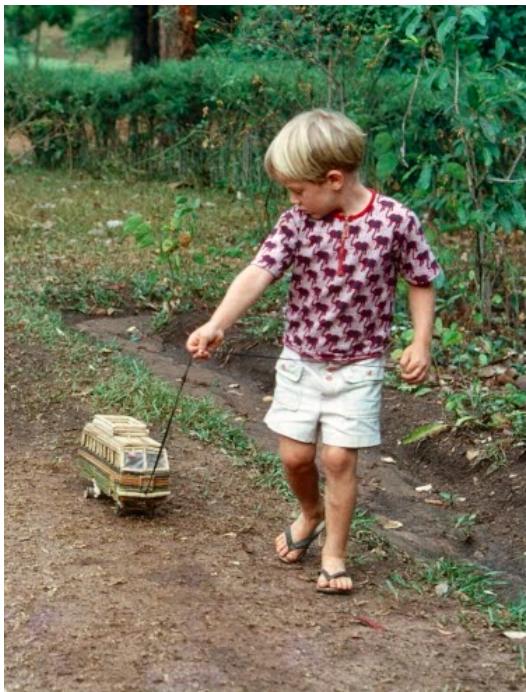


Dave Clark



Artist's conception of what a large Multics system might be

Jeff Dean



Jeff in Uganda

Panel: Is achieving security a hopeless quest?



Logistics

Impossible tasks for 25 minutes!! But we will still try very hard to stick with it

Try for 2 questions from audience after each presentation

History Evening Session

Tonight at 7 PM!

Just after opening reception

Open mic!

Bring your thoughts and memories

Thank you to the SOSP History Committee!!!

Peter Denning (chair)

Andrew Birrell

Mike Schroeder

Jeanna Matthews

Virgil Gligor

Matt Blaze

Steve Bellovin

Jack Dennis

Mark Miller

Andy Tanenbaum

Peter Neumann

Casey Henderson

Robert Watson

Fred Schneider

Ken Birman

Robbert van Renesse



Thank you to Andy Tanenbaum
for pictures of SOSPs!

OS Foundations

Peter J. Denning
October 4, 2015

Our Story

OS Principles began emerging 1960,
Grew across many generations of technologies,
And left a rich heritage in the minds, hearts, and
souls of all who use computers.

Our Story

Timelines

Personal example

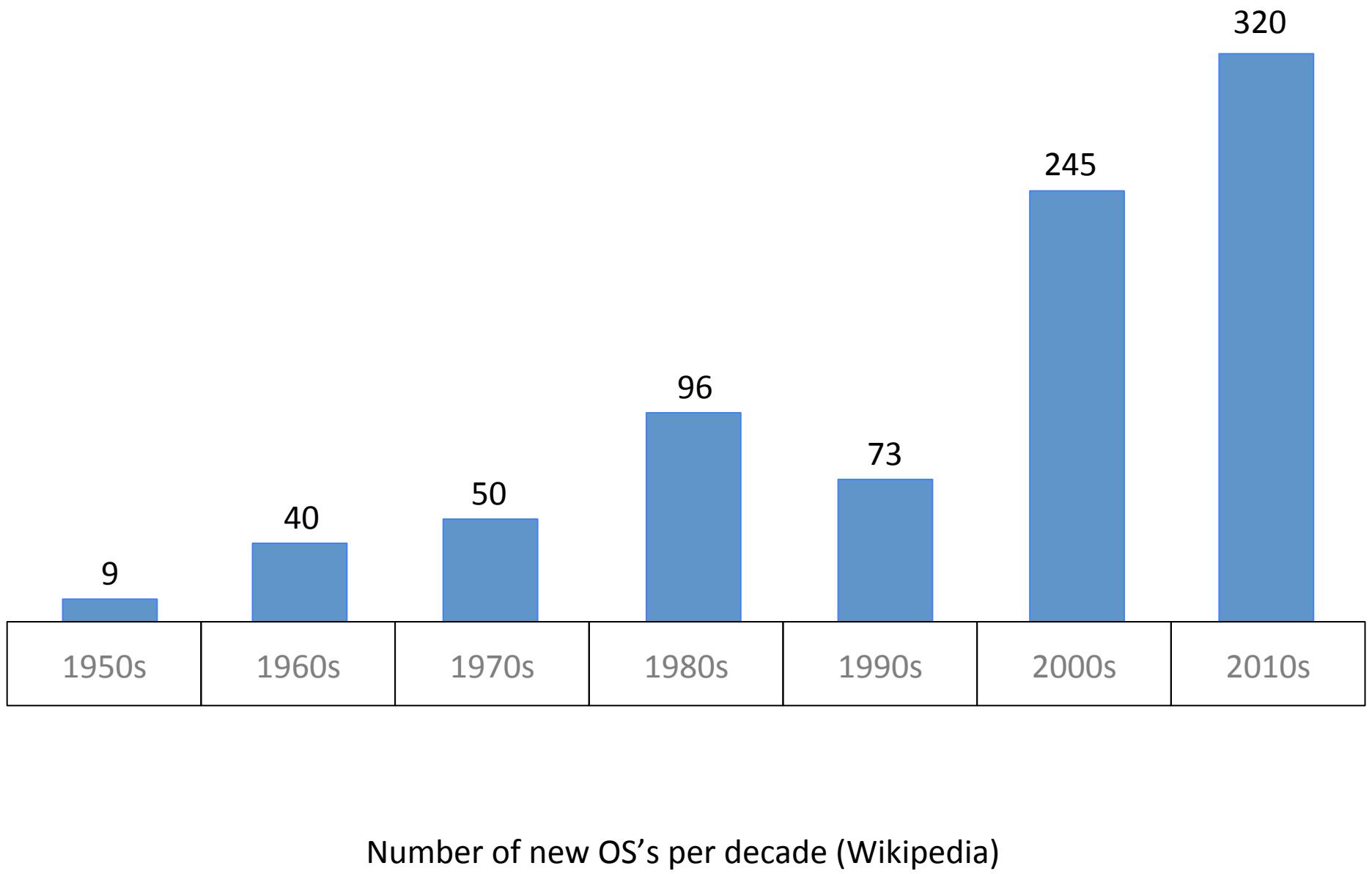
Research Lessons

OS a force on all of computing

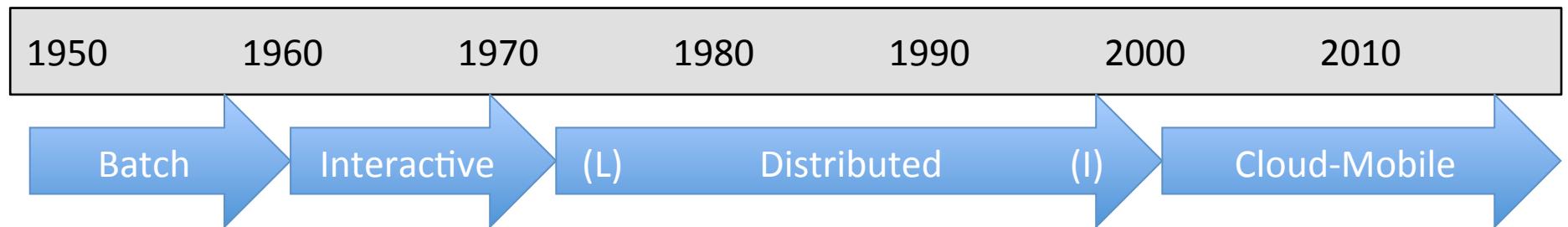
We will be surfing ...



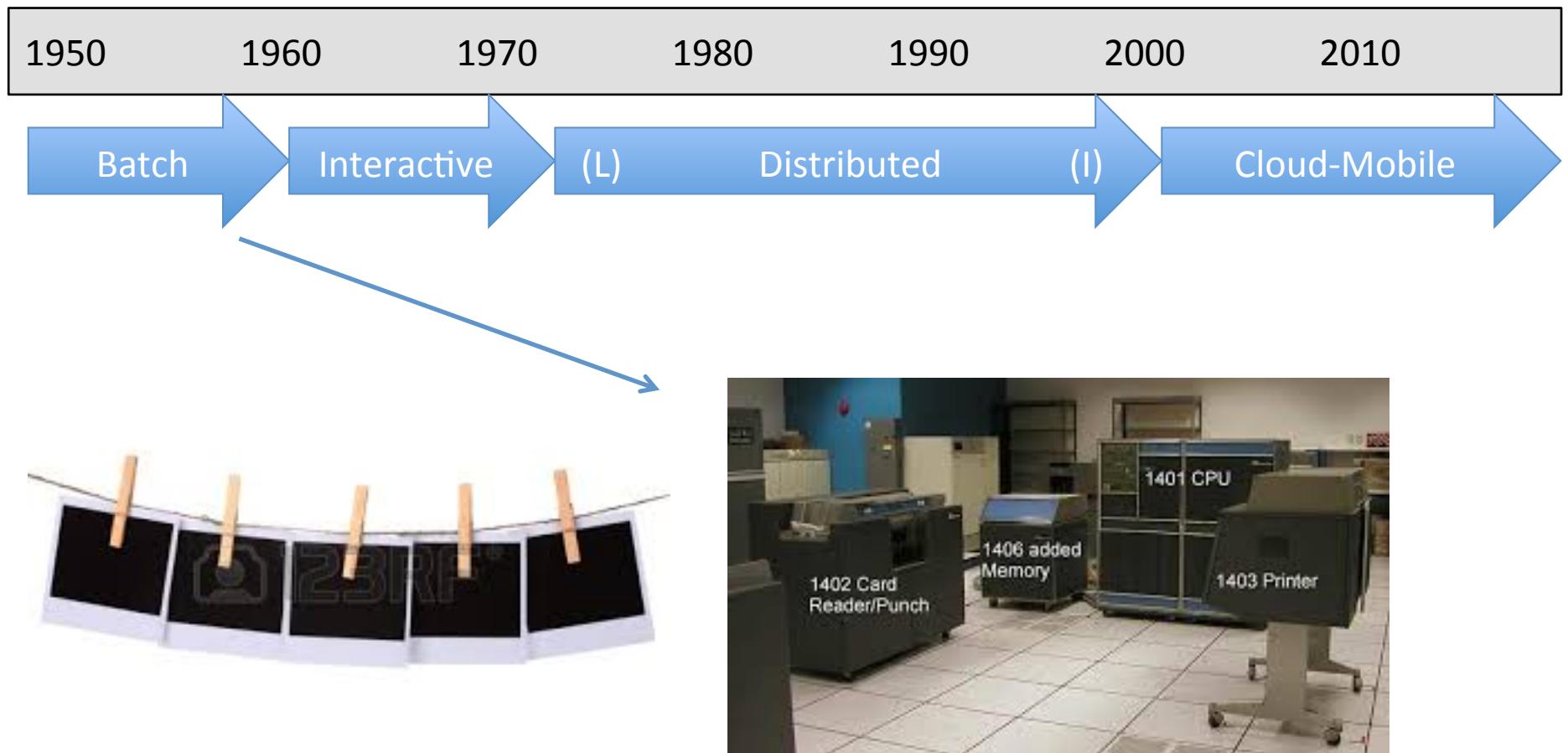
Source: public domain internet



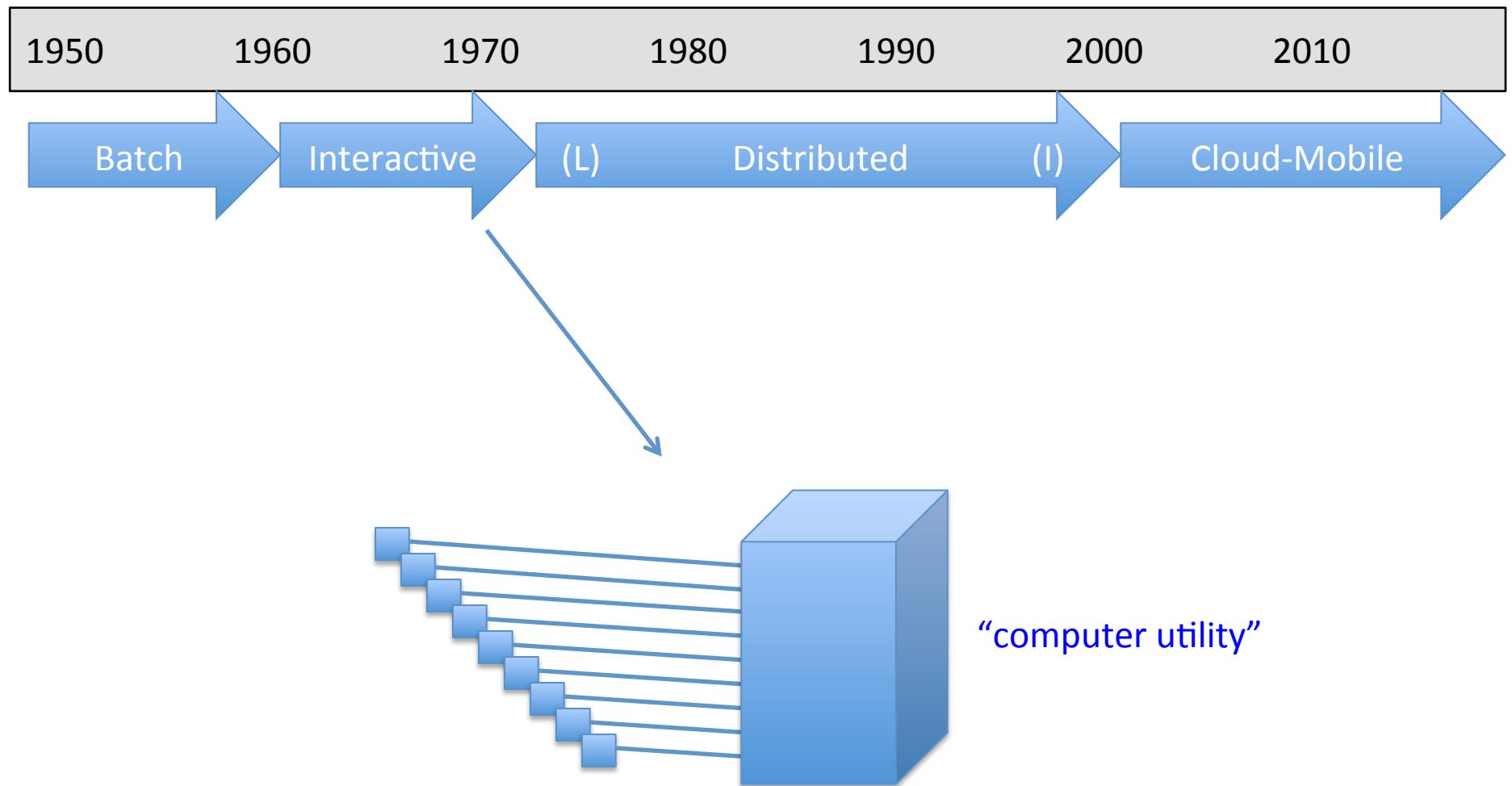
Eras of Operating Systems



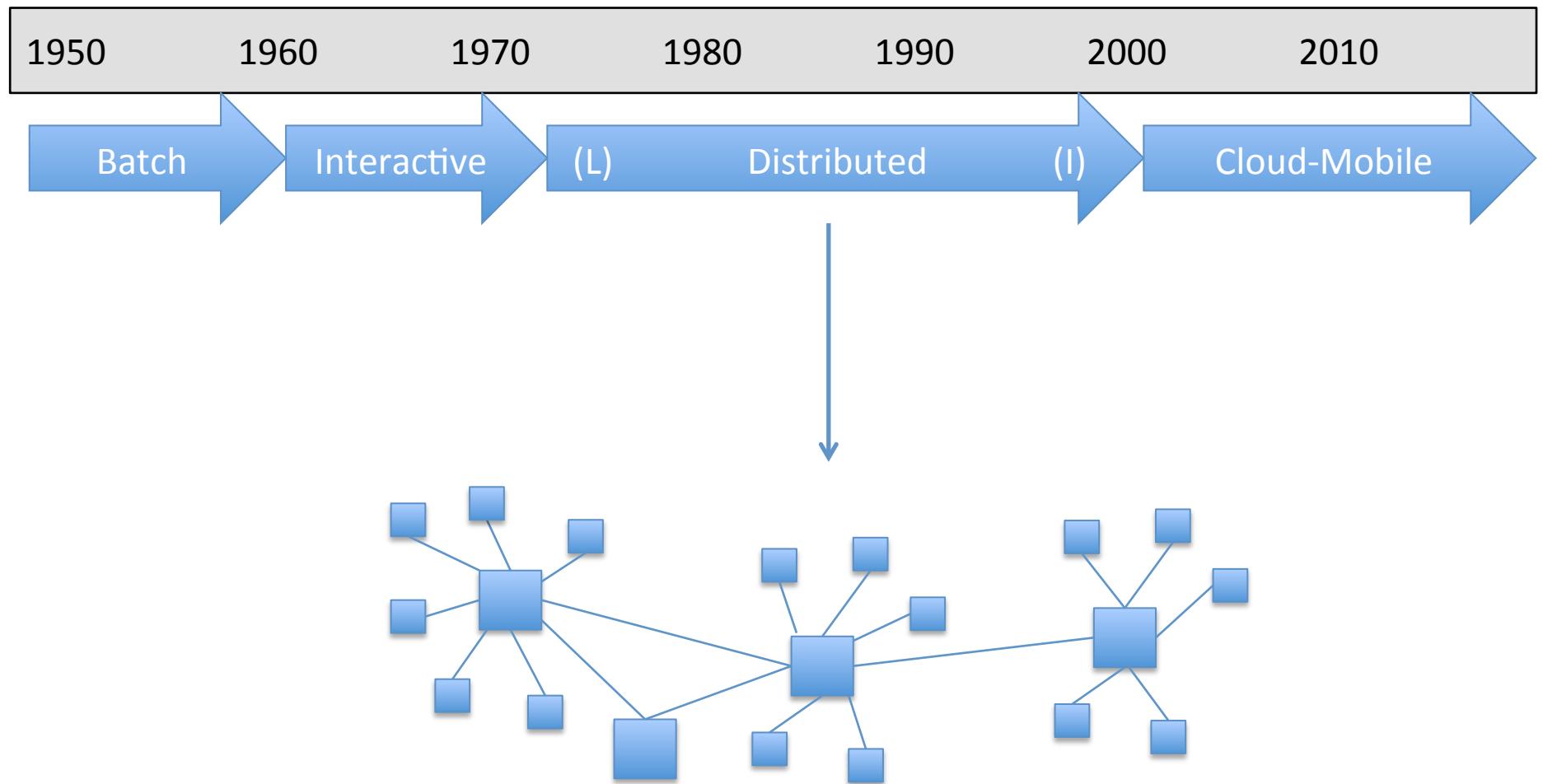
Eras of Operating Systems



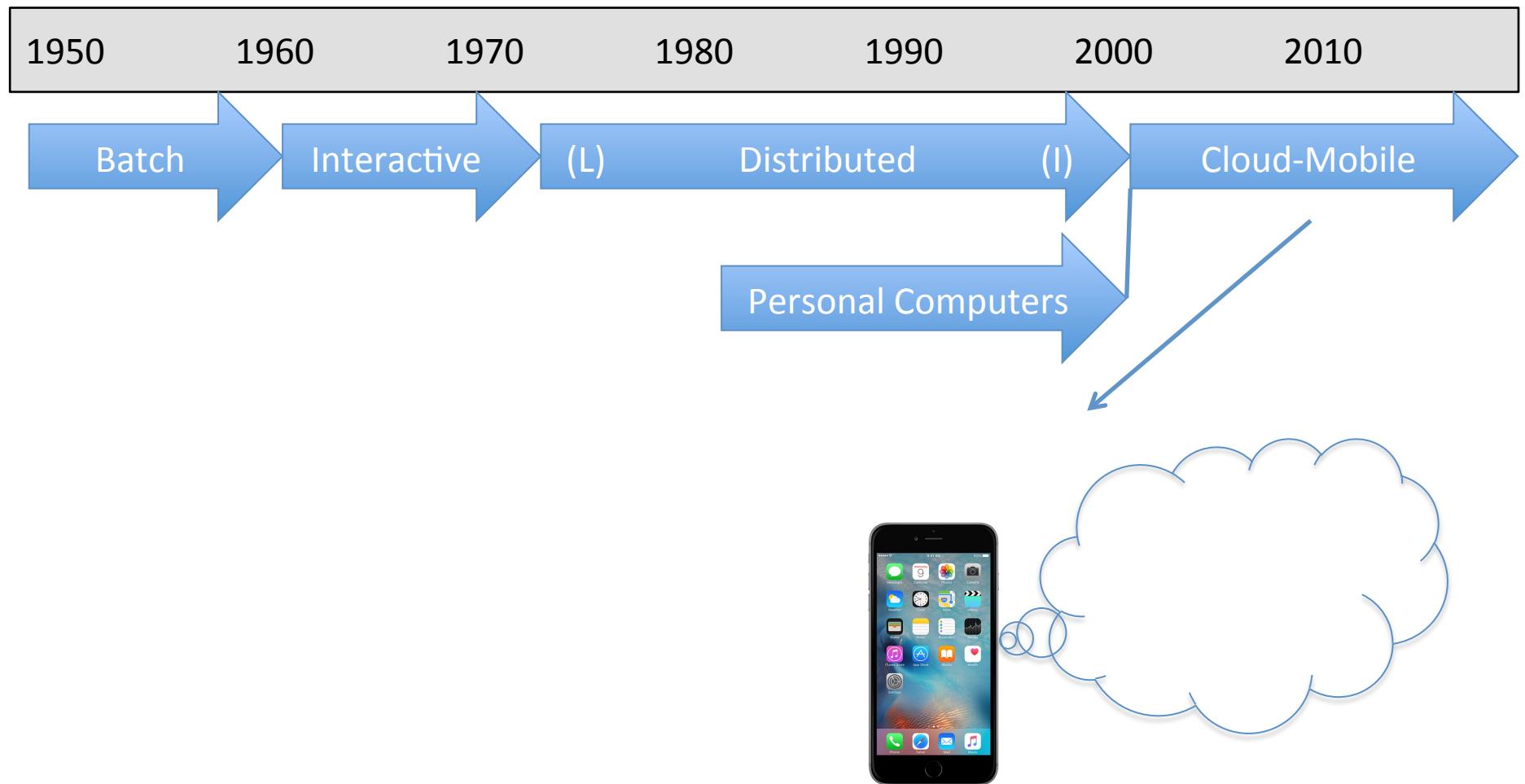
Eras of Operating Systems



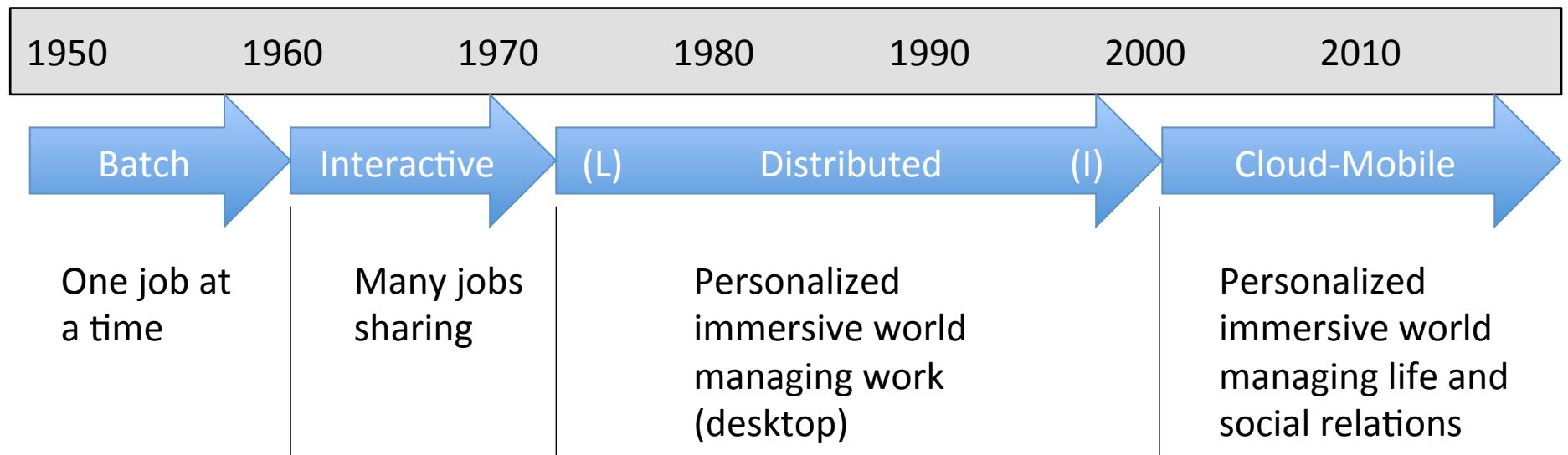
Eras of Operating Systems



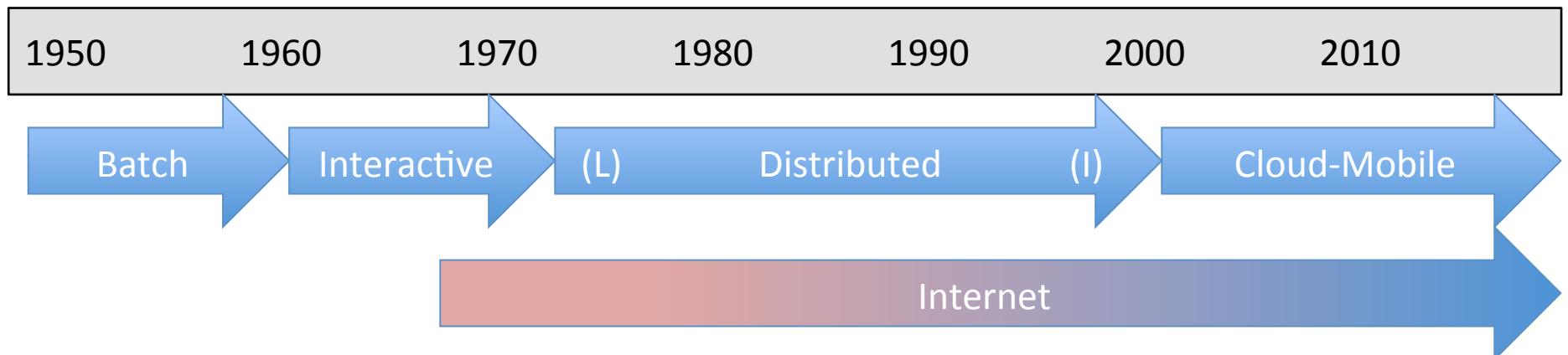
Eras of Operating Systems



Eras of Operating Systems



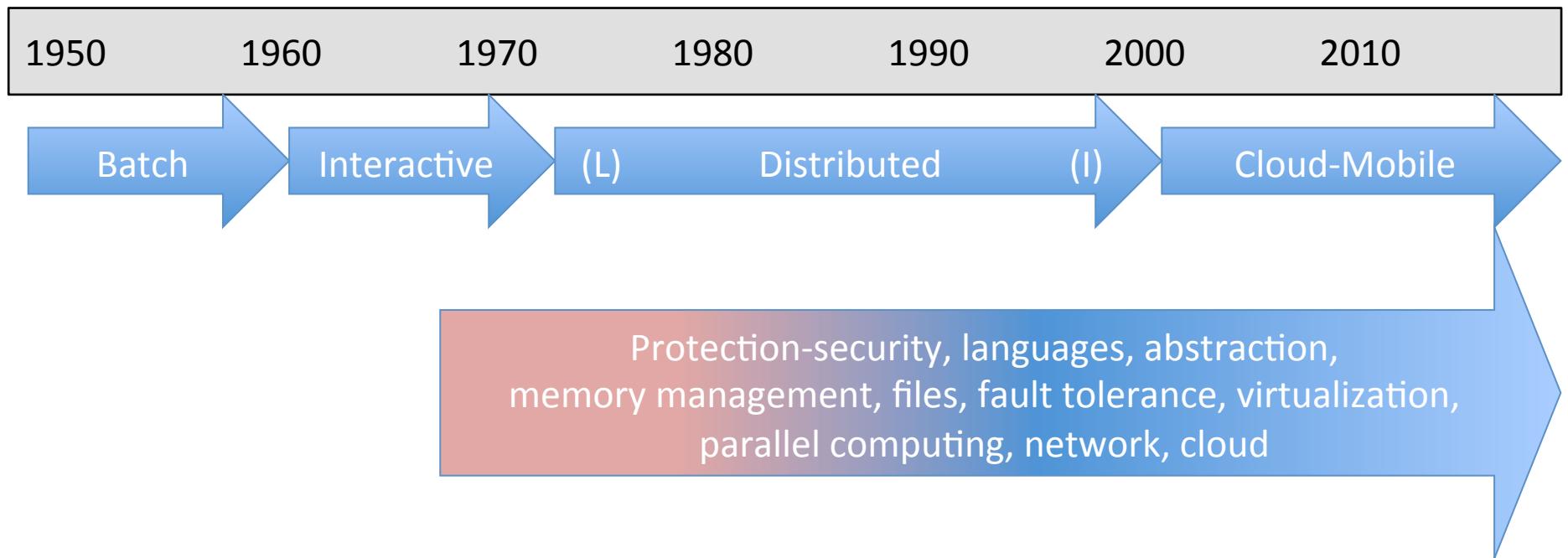
Eras of Operating Systems



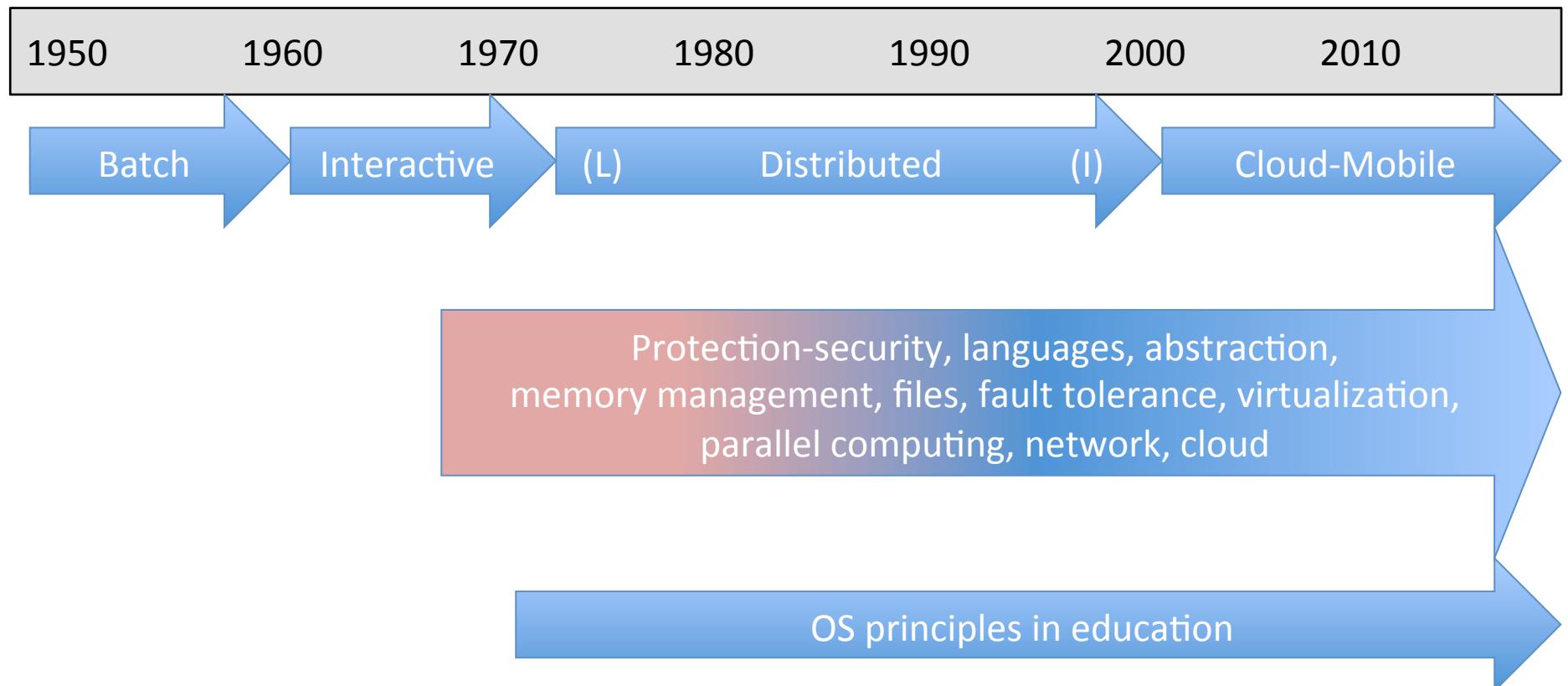
“OS interfaces with”:
TELNET
FTP
SMTP
RLOGIN
RCOPY

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

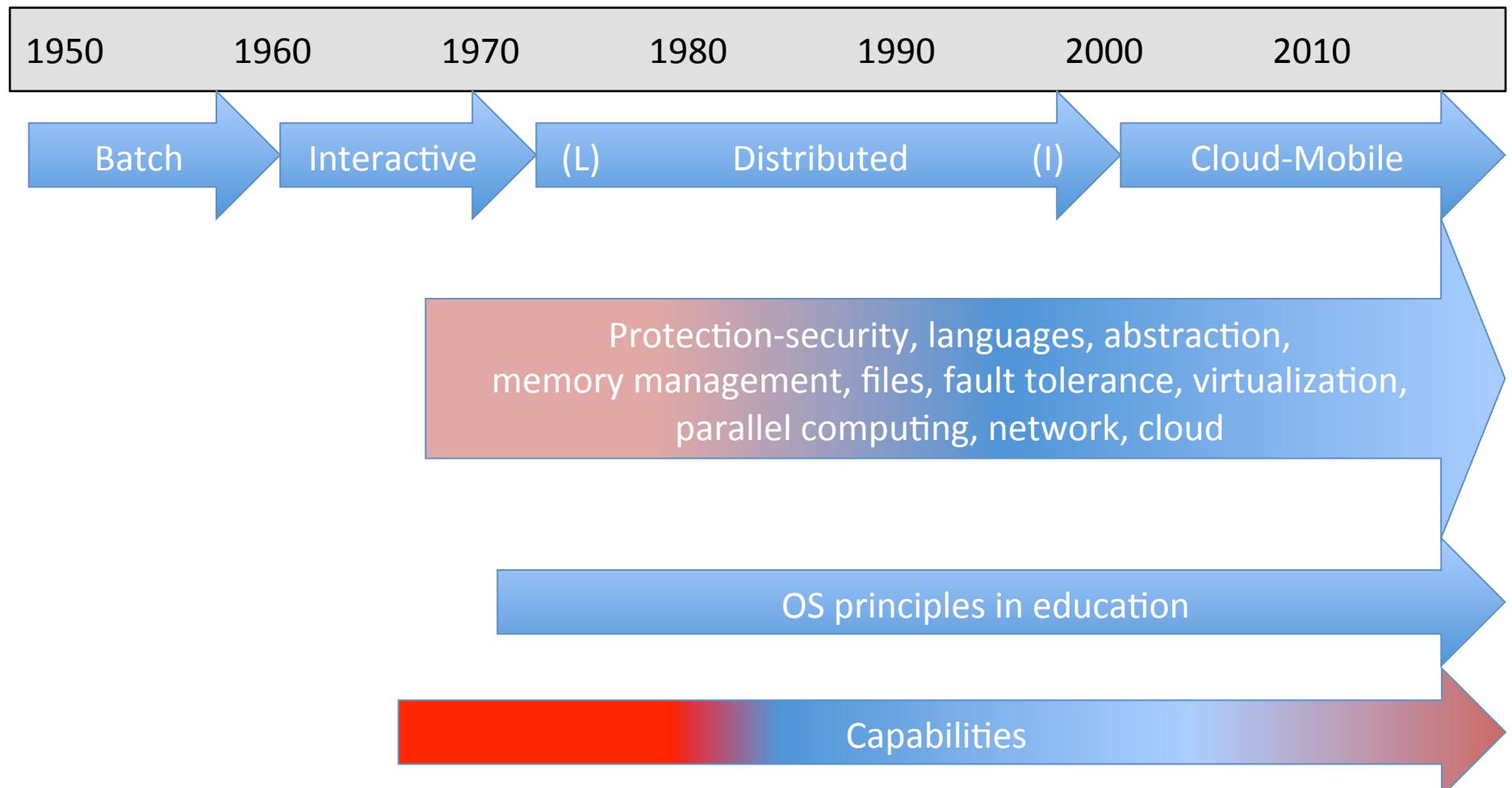
Eras of Operating Systems



Eras of Operating Systems



Eras of Operating Systems



Development of Principles

What is a computing principle?

Development of Principles

What is a computing principle?

Law

Development of Principles

What is a computing principle?

Law

$$c(t) = \min(a(t), s(t)+l)$$

$$M = (\text{spacetime}) \times (\text{throughput})$$

Mean Value Equations

Locality

Development of Principles

What is a computing principle?

Law

Statement of Design Wisdom

Development of Principles

What is a computing principle?

Law

Statement of Design Wisdom

Information hiding

Levels of abstraction

Development of Principles

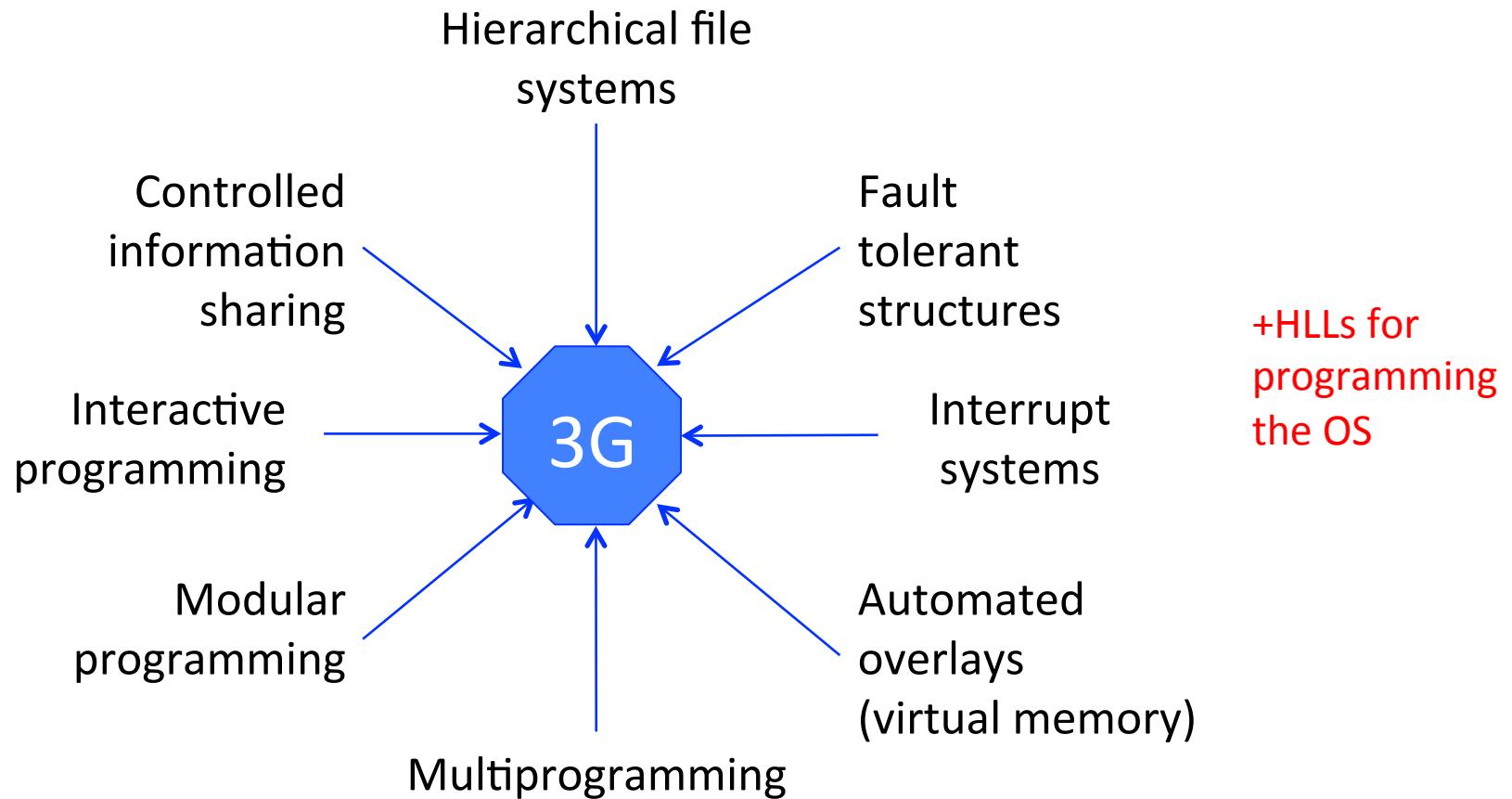
What is a computing principle?

Law

Statement of Design Wisdom

Cosmic = timeless + spaceless (Jim Gray)

Eight programming support objectives added by 1965 seeded the research



Principles govern only the recurrences part of our story.

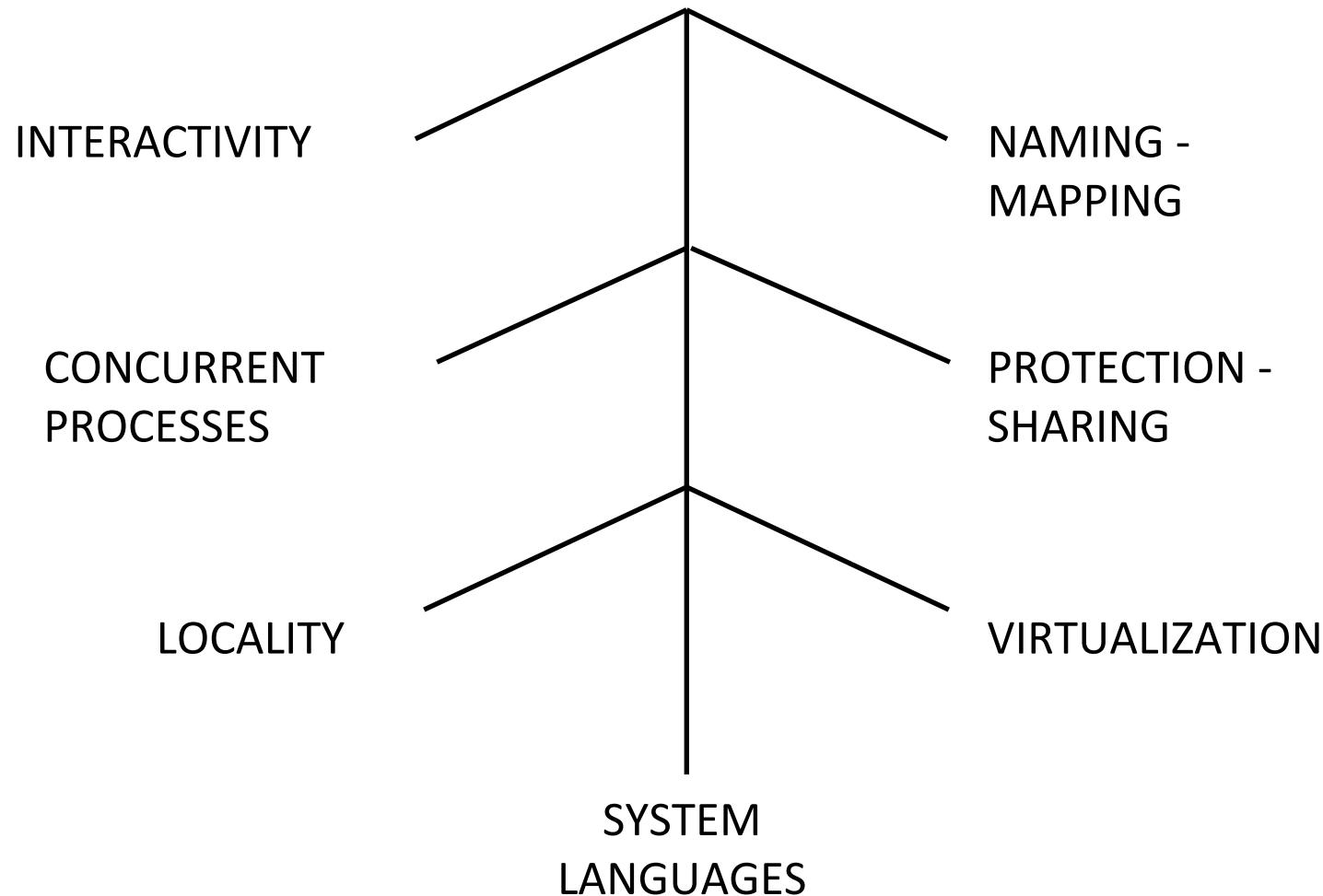
Accidents and unforeseen contingencies constantly appear.

We respond to them:

With bug fixes, patches, new designs, new apps.

With research seeking greater understanding and occasionally opening new insights and new principles.

OS Principles into CS



OS Principles into CS

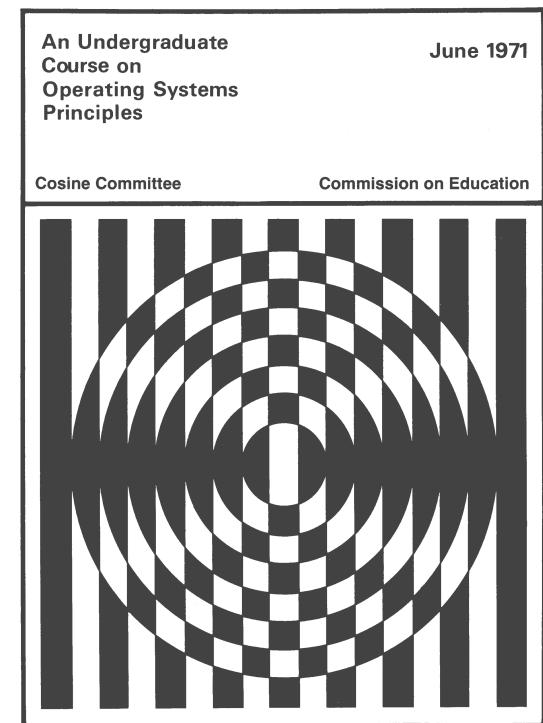
In all,

13 of 41 CS Principles (30%)

Source: greatprinciples.org

OS Principles into CS

- First non-math core course 1971
- Expanded definition
of core to include systems
- Unchallenged for 44 years



Two Cosmic Principles Revealed in Memory Management

Locality

Location independent addressing

Emerged from virtual memory research

Originally seeking to automate overlays

Locality Principle

Big Adventure

I became involved at Project MAC 1965

Many people involved (thousands!)

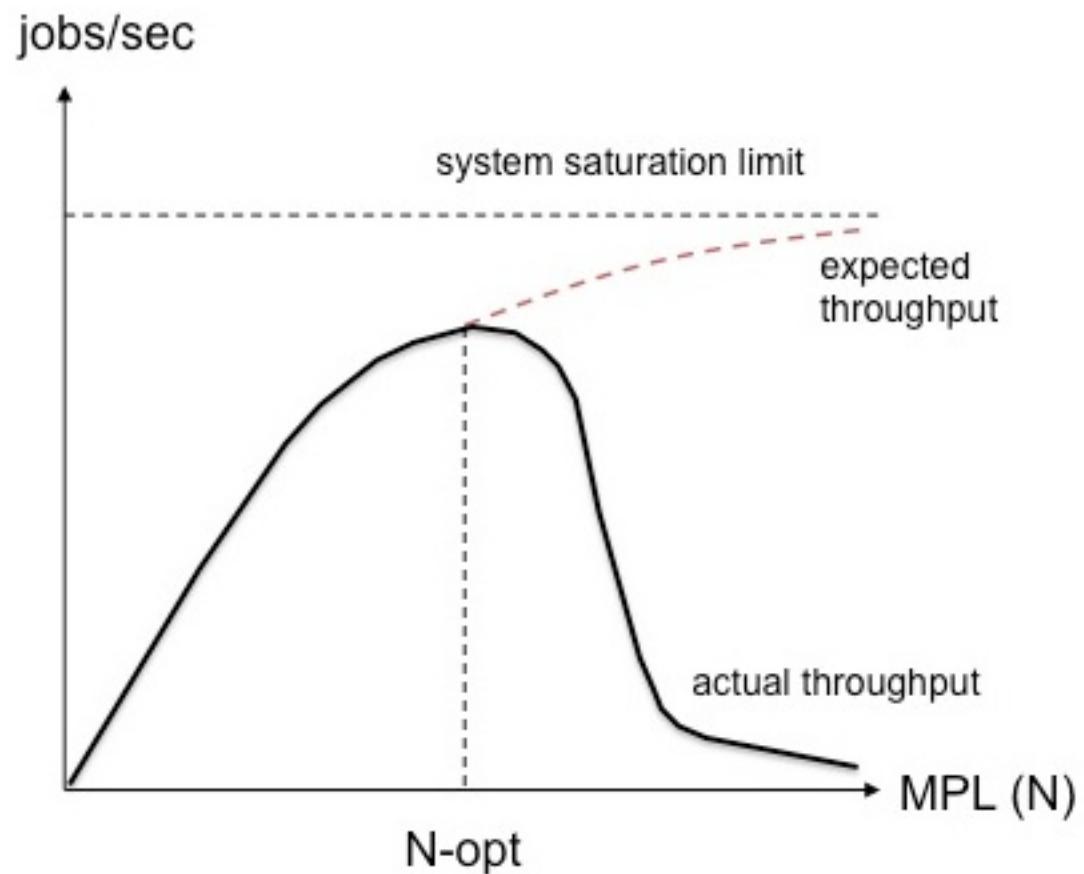
First motivations ca. 1965 performance related:

Performance of virtual memory

Multiprogramming

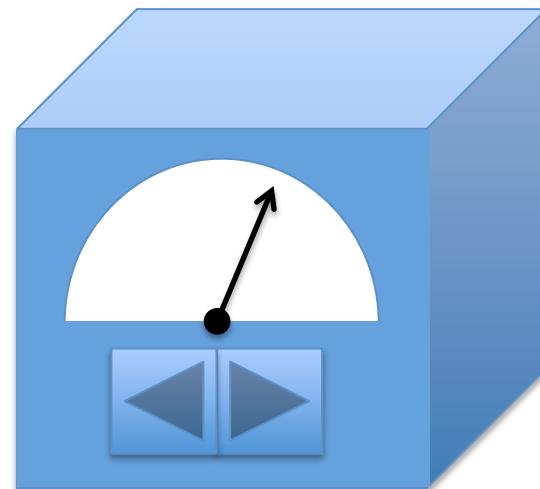
Thrashing

Thrashing

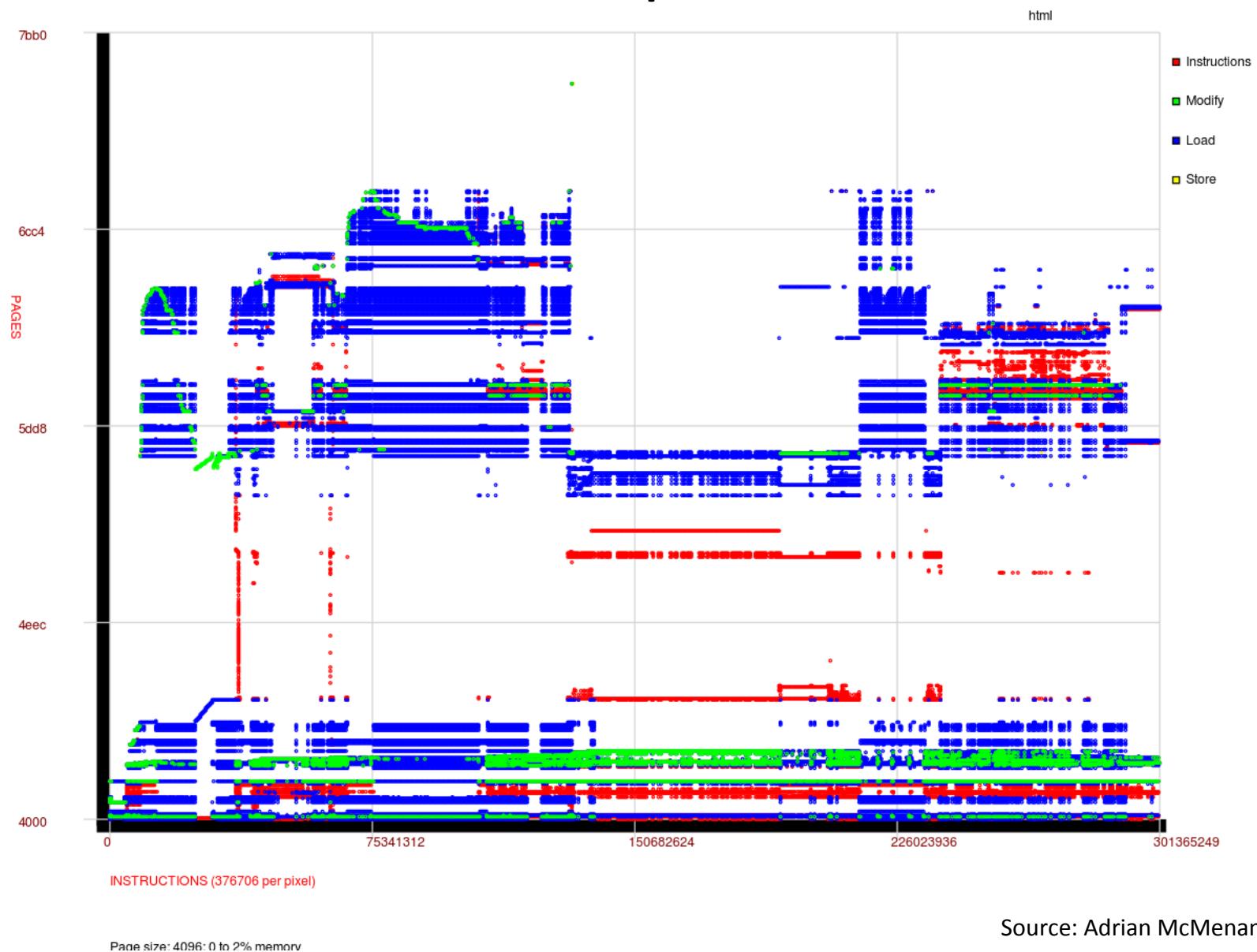


Saltzer's Challenge

Tune one parameter to
lock in optimal
performance



Reference Map – Initial Intuitions



Key insights:

- Temporal and spatial clustering (Belady, Denning 1966)
- Working sets (Denning 1966)
- Reference maps (IBM ca 1969)
- Optimality principle (Belady 1966, Prieve and Fabry 1976, Gray 1995)

Working set

Pages used in previous
virtual time window
of size T

Vertical slice in
page reference map

What We Have Learned

WS = locality set most of the time

WS policy near optimal (VMIN)

Economical implementations (WSCLOCK)

Prevents thrashing

Answers Saltzer's challenge

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.

Patterns I learned from OS Research

1. There is never certainty
2. Occasionally an insight charts a new direction
3. Technology inflection points may trigger avalanches
4. Searching for what works: building, experimenting, tinkering
5. Always in a social network
6. Theory follows practice

Summary

- Rich heritage from OS evolution

Summary

- Rich heritage from OS evolution

Batch ... Interactive ... Immersive

Summary

- Rich heritage from OS evolution
 - Batch ... Interactive ... Immersive
- Fundamental CS principles emerged from our research

Summary

- Rich heritage from OS evolution
 - Batch ... Interactive ... Immersive
- Fundamental CS principles emerged from our research
- OS drove major change to CS curriculum

Summary

- Rich heritage from OS evolution
 - Batch ... Interactive ... Immersive
- Fundamental CS principles emerged from our research
- OS drove major change to CS curriculum
- OS research a unique blend of engineering, experimenting, and modeling

Challenges Ahead

Automation of knowledge work – end of “knowledge age”

Network: space of social power, action, identities

Security and privacy in the Internet of Things

Size, complexity, scale of systems

Integrating with bio and nano tech

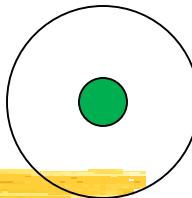
End of this story

Perspectives on Security



Butler Lampson
Microsoft Research
Symposium on Operating Systems Principles
October 4, 2015

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)

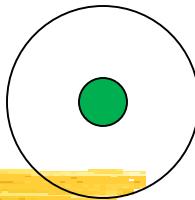
Isolation vs. sharing

- 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

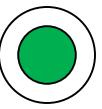


What we know how to do

- 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

<i>Winner</i>	<i>Loser</i>	<i>(in deployment, not good vs. bad)</i>
Convenience	vs. Security	
Sharing	vs. Isolation	
Bug fixes	vs. Correctness	
Policy/mechanisms	vs. Assurance	
Access control	vs. Information flow	

Timeline



Themes

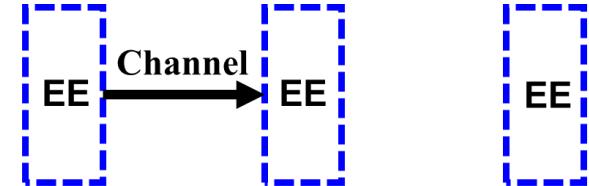
Systems

	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 ...

Foundation: Isolation



- A host isolates an execution environment
 - The basis for any security. Must trust the host
- Many ways to do it (and many bugs):



Host (CLR, kernel,
hardware, VMM, ...)

Mechanism

Java/JavaScript sandboxing

Modules/objects

Software fault isolation

Processes

Virtual machines

Limited comm (wires or crypto)

Air gaps: physical separation

Host

JVM/JS engine

language/runtime

process

OS

hypervisor

network

physics

Java 1995

Clu 1974

Wahbe et al 1993

CTSS 1962

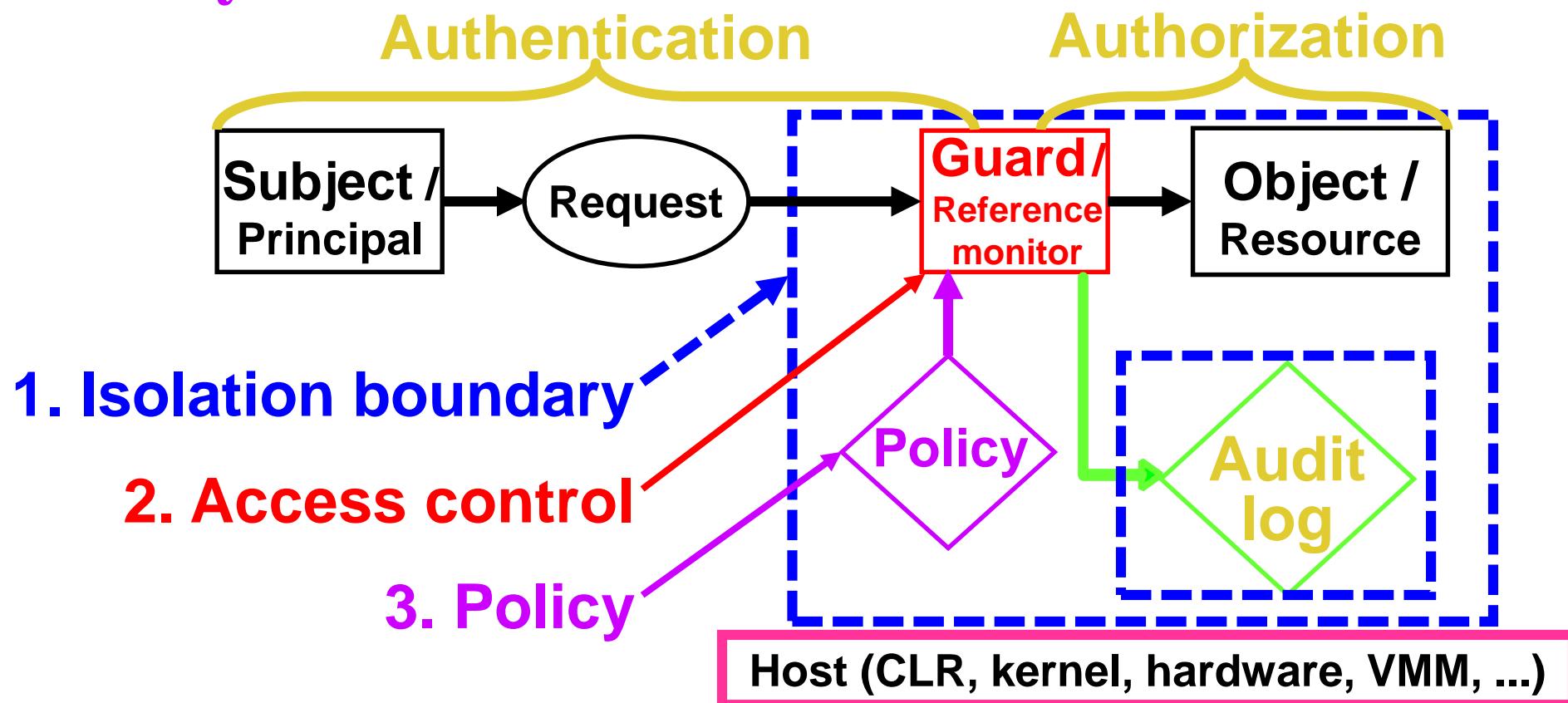
CP/40 1966

DESNC 1985

1950; Tempest ~1955

Safe Sharing: Access Control

1. **Isolation boundary** limits attacks to channels (no bugs)
2. **Access Control** for channel traffic
3. **Policy** sets the rules



Access Control: The Gold Standard

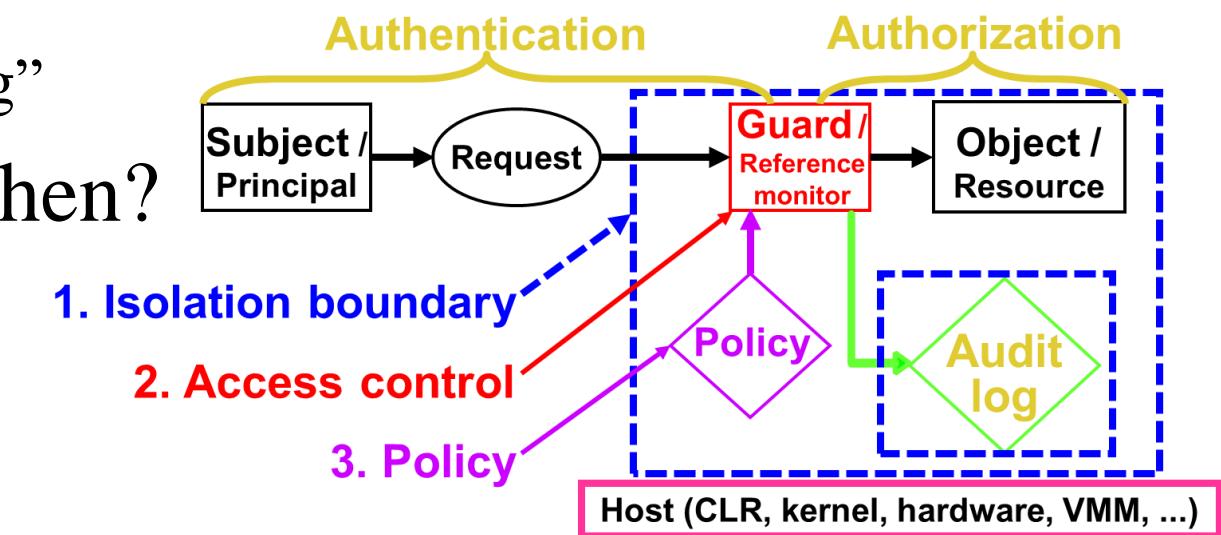
■ **Authenticate** principals: Who made a request?

- People, but also channels, servers, programs
(encryption implements channels, so the key is a principal)

■ **Authorize** access: Who is trusted with a resource?

- *Group* principals or resources, to simplify management
 - Can define a group by a property,
e.g. “type-safe” or “safe for scripting”

■ **Audit** requests: Who did what when?



Policy: What sharing is allowed?

- The guard evaluates a function: permissions = policy(subject, object)
 - If functions are too mathematical, call it an access matrix (Lampson 1971)

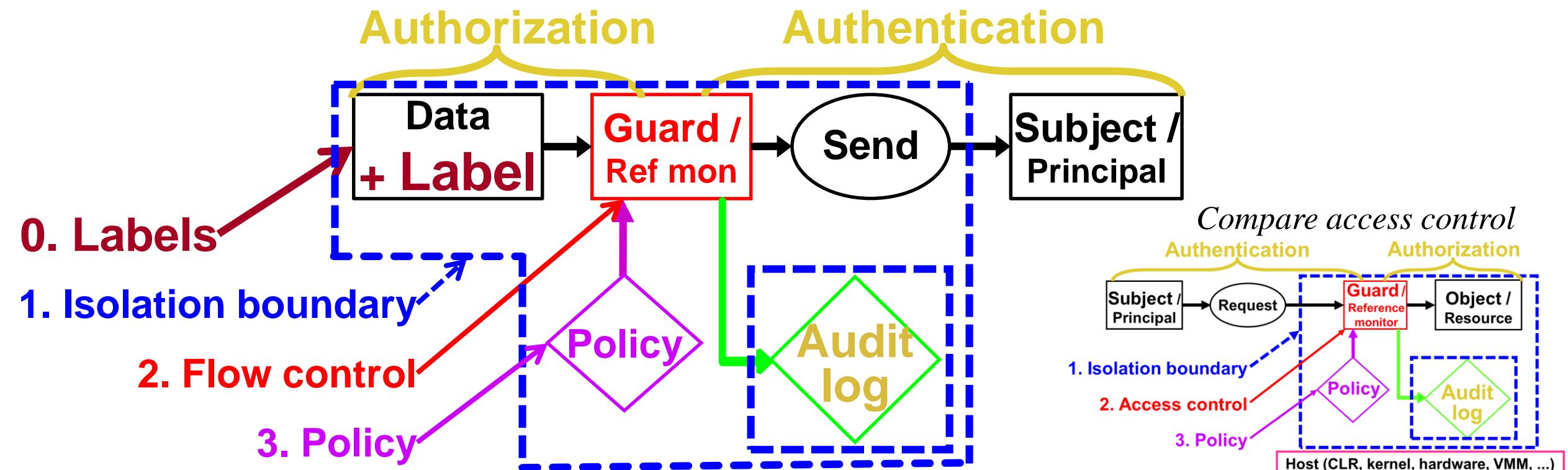
Subject/principal	Object/resource	
	File foo	Database payroll
Alice	<i>read, write</i>	<i>write paychecks</i>
Bob	<i>read</i>	-

- Permissions kept at the object are ACLs; at the subject, capabilities
 - Capabilities work for short term policy
 - File descriptors/handles in OS; objects in languages (Unix/Windows; Java, C#)
 - ACLs work for long-term policy; tell you who can access the resource
 - Groups of subjects and objects keep this manageable (Multics 1968)

Keeping Secrets: Information Flow Control

0. **Labels** on information
1. **Isolation boundary** limits flows to channels
2. **Flow control** based on labels
3. **Policy** says what flows are allowed

Adept-50 1969
Orange Book 1985



Information Flow Control

- Invented to model military classification (Adept-50 1969)
 - Label every datum: top secret/nuclear \geq top secret \geq secret
 - Labels form a lattice, and propagate: If d_1 is input to d_2 , then d_2 's label is $\geq d_1$'s
 - Enforce with access control: label subjects, containers (Bell/LaPadula 1973)
 - No read up, write down; can be dynamic or static (Adept-50; Denning 1976)
- Decentralized flow control (Myers and Liskov 1998)
 - Anyone can invent labels. If you own a label, you can declassify it
 - Can do this in a language or in an OS (Jflow 1999; HiStar 2006)
- So far, none of this has been practical
- And then there are **covert** (side) channels: timing, radiation, power ...
 - Abstractions don't keep secrets (Tempest 1955, Lampson 1972)

Who controls policy? DAC, MAC, RBAC

■ How to decide:

- Is the user or the program **malicious**? Insiders, Trojan horses
- Is the user **competent**? Policy can be tricky
- Is the user **motivated**? Security gets in the way of work and play

■ Discretionary access control (DAC) : the object's owner (CTSS 1963)

■ Mandatory access control (MAC) : an administrator (1969; 1985)

- MAC \neq flow control

■ Role based access control (RBAC): the app designer (NIST 1992)

- Administrator just populates the roles

Distributed Systems: Cryptography

- **Communicate**, so need secure channels
 - Host, secure wire, ..., but usually cryptography: General, **end to end**
 - Basic crypto functionality: mathematical magic that implements:
 - Sign with K^{-1} / verify with K : integrity
 - Seal with K / unseal with K^{-1} : secrecy
 - This gives you an **end-to-end** secure channel
 - Public key crypto: $K \neq K^{-1}$; I can sign, anyone can verify (RSA 1977)
 - **Homomorphic** crypto: compute on encrypted data (Gentry 2009)
 - This is too slow, but you can *simulate* it (CryptDB 2011)
 - Zero knowledge and **verifiable** computation (Pinocchio 2013)

Distributed Systems: Understanding Trust

- Decentralized, so have to reason about trust, justifying by proofs
- Principals: people, machines, programs, services, protocols, ...
- Accountability: principal **says** statement
 - Alice **says** read from file Foo
- Trust: principal A **speaks for** principal B DEC 1989, 1991
 - Alice **says** Bob@microsoft **speaks for** Alice
 - Microsoft **says** Key63129 **speaks for** Bob@microsoft
 - Key63129 **says** read from file Foo
 - file Foo **says** Alice **speaks for** file Foo ACL entry
 - So Foo **says** read from file Foo
 - End to end reasoning

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

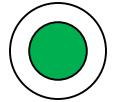
Bandaids for Bugs (Defense in Depth)

- 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 about *my* system? Configuration (Policy)

- If the code is correct, the configuration may still be wrong
 - You write the code once, but every system has its own configuration
 - It's boring, and it changes. So either it's small, or it's wrong.
 - But it's not small; there's always another feature, another plausible scenario
 - There are 12 levels of indirection in Windows printing, each with its own security
- And configuration is usually done by amateurs
 - With MAC and RBAC at least it's done by pros
- **Conflict:** want fine grained policy, but can only manage coarse grain
 - Solution (never adopted): Lower aspirations, budget for complexity





What has worked? What hasn't?

Worked ~ gotten wide adoption

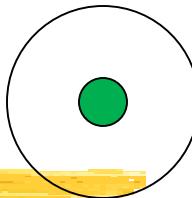
Worked

- VMs
- SSL
- Passwords
- Safe languages
- Firewalls
- Process—SDL

Failed

- “Secure systems”
- Capabilities (except short term)
- Metrics for security
- MLS/Orange book
- User education
- Intrusion detection

Why don't we have “real” security?



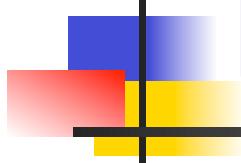
- A. People don't buy it
 - Danger is small, so it's OK to buy features instead
 - Security is expensive
 - Configuring security is a lot of work
 - Secure systems do less because they're older
 - Security is a pain
 - It stops you from doing things
 - Users have to authenticate themselves
 - Goals are 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



Perspectives on System Languages and Abstractions



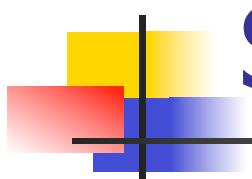
Barbara Liskov

October 2015

MIT CSAIL

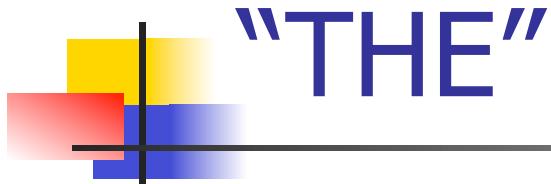
Abstractions for Structuring Systems

- The early days
- Single machine systems
- Distributed systems



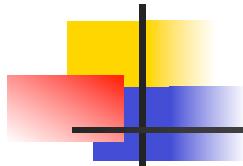
Single Machine Systems

- In the beginning: batch processing
- So:
 - Multiprogramming
 - Time sharing



“THE”

- E. W. Dijkstra, The structure of the “THE”- Multiprogramming system
 - CACM 68, SOSP 67, and **EWD 196**
- Strictly layered
- Independent users



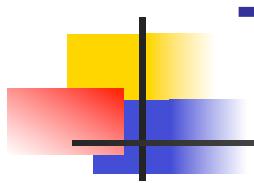
Layer 0

- Processes and **semaphores**
 - P and V operations
- Used for
 - Critical sections
 - IPC ("private" semaphores)
- No "deadly embrace"



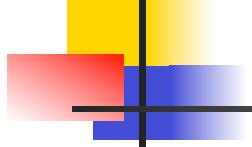
Venus

- B. Liskov, The design of the Venus operating system
 - CACM 72 and SOSP 71
- A time-sharing system
- Processes and semaphores in microcode



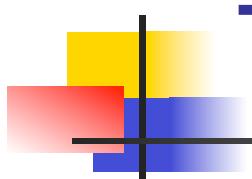
The Structure of Venus

- Resources presented through “layers of abstraction”
 - Multiple operations
 - Hidden state and resources
 - Calls ran in process of caller
- E.g., a printer requestor



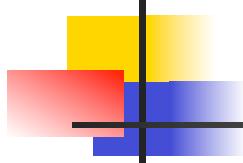
Two System Models

- Resources managed by **resource processes**
 - With IPC
- Resources managed by **user processes**
 - With abstract data types (ADTs) and procedure calls



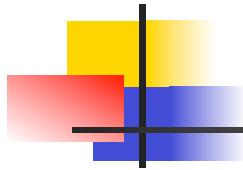
These Models are Duals

- Lauer and Needham, On the duality of operating system structures,
 - Proc. 2nd International symp. on operating systems, 78 and SIGOPS Review 79
- E.g., port == operation



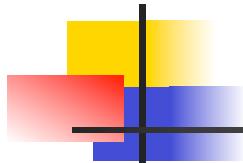
Programming Issues

- Resource process multiplexing
- User process synchronization
 - monitors
 - C. A. R. Hoare, CACM 74, Monitors: an operating system structuring concept



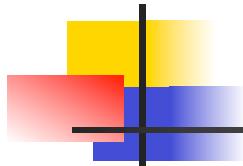
Monitors

- ADT with associated lock acquired automatically
- Plus **condition variables**
 - Wait c releases the monitor lock
 - Signal c passes the lock



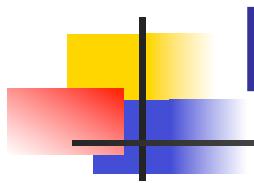
Monitors in Mesa

- Lampson and Redell, Experience with processes and monitors in Mesa
 - CACM 80 and SOSP 79
- Issues:
 - Nested monitor problem
 - “external” operations



Programming Languages

- Modula and later variants
- Concurrent Pascal
- Mesa



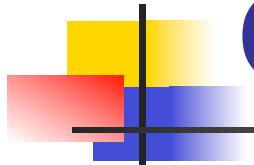
Distributed Systems

- Motivation

- Sharing on a LAN
- The dream of distributed computing

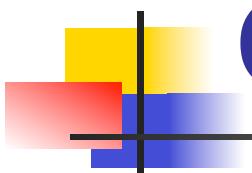
- But: how to structure?

- Clients and servers?
- Distributed heap?



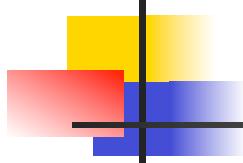
Communication is Required

- Communication is hard
 - "... construction of communicating programs was a difficult task, undertaken only by members of a select group of communication experts." (B&N, Implementing remote procedure calls, TOCS 84)



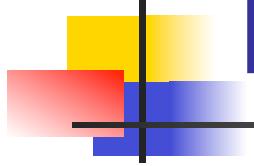
Communication Issues

- Linking requests with replies
- Format of messages
 - Heterogeneity vs. homogeneity
- Location independence
 - Local vs. remote
 - Finding/selecting remote servers



Remote Procedure Calls

- B. J. Nelson, Remote procedure call
 - Xerox Parc TR CSL-81-9
- Birrell and Nelson, Implementing remote procedure calls
 - TOCS 84 and SOSP 83



RPC Motivation

- It's clean and simple and general
 - Local and remote calls look the same
- Issues in request/reply are similar

RPC (B&N, TOCS 84)

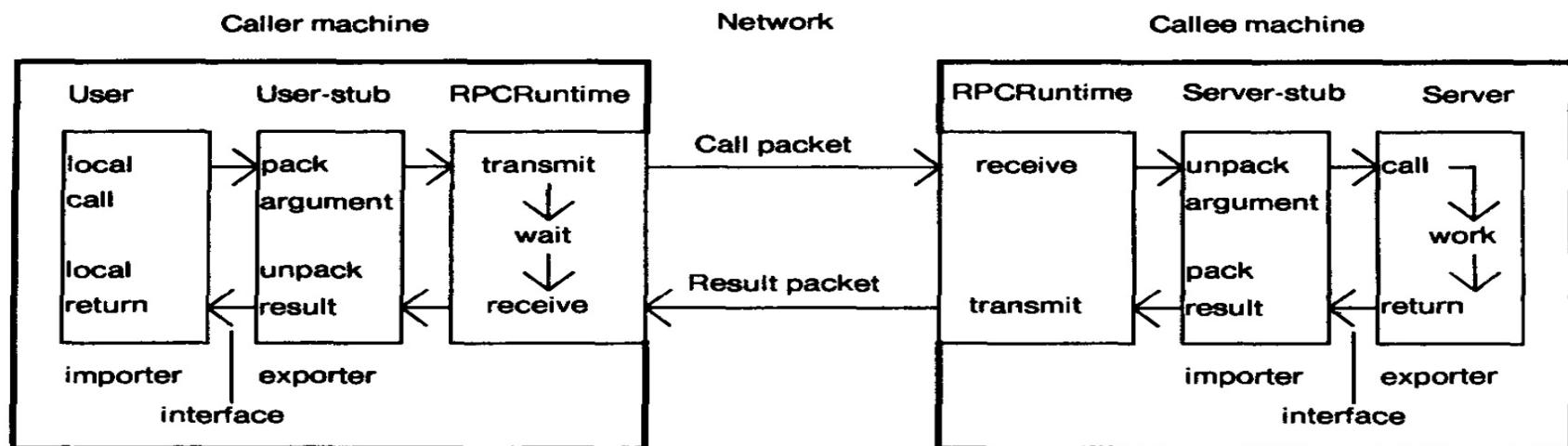
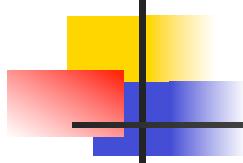
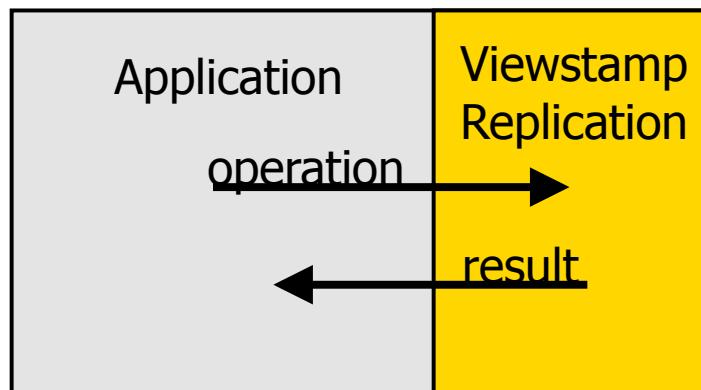


Fig. 1. The components of the system, and their interactions for a simple call.

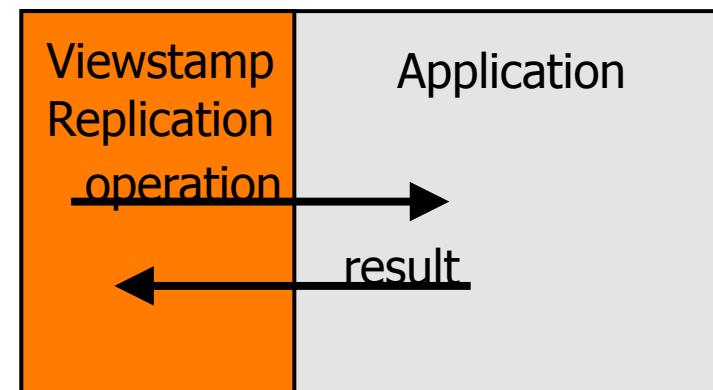


Doing More

Client



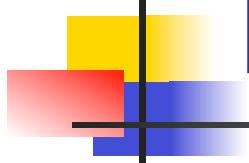
Replica





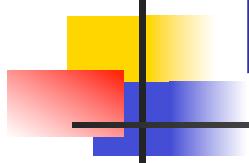
RPC Issues

- Inherent expense



RPC Issues

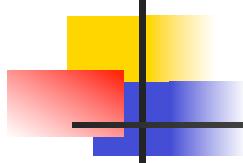
- Call/reply too constraining
 - Liskov and Shrira, Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems, PLDI 88
 - Gifford and Glasser, Remote pipes and procedures for efficient distributed communication, TOCS 88



RPC Issues

■ Semantics

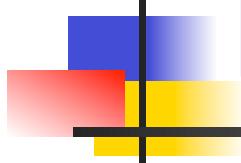
- Exactly once if reply (B&N 84)
- Exactly once (Liskov and Scheifler,
Guardians and actions: Linguistic support
for robust, distributed programs, TOCS 83)



What Next?

- Perhaps we need new abstractions?
 - Client/server with extended RPC?
- Perhaps we should be doing more language design?

Perspectives on System Languages and Abstractions



Barbara Liskov

October 2015

MIT CSAIL

Memory and File Systems

SOSP-25 Retrospective

Mahadev Satyanarayanan

School of Computer Science

Carnegie Mellon University

Four Drivers of Progress

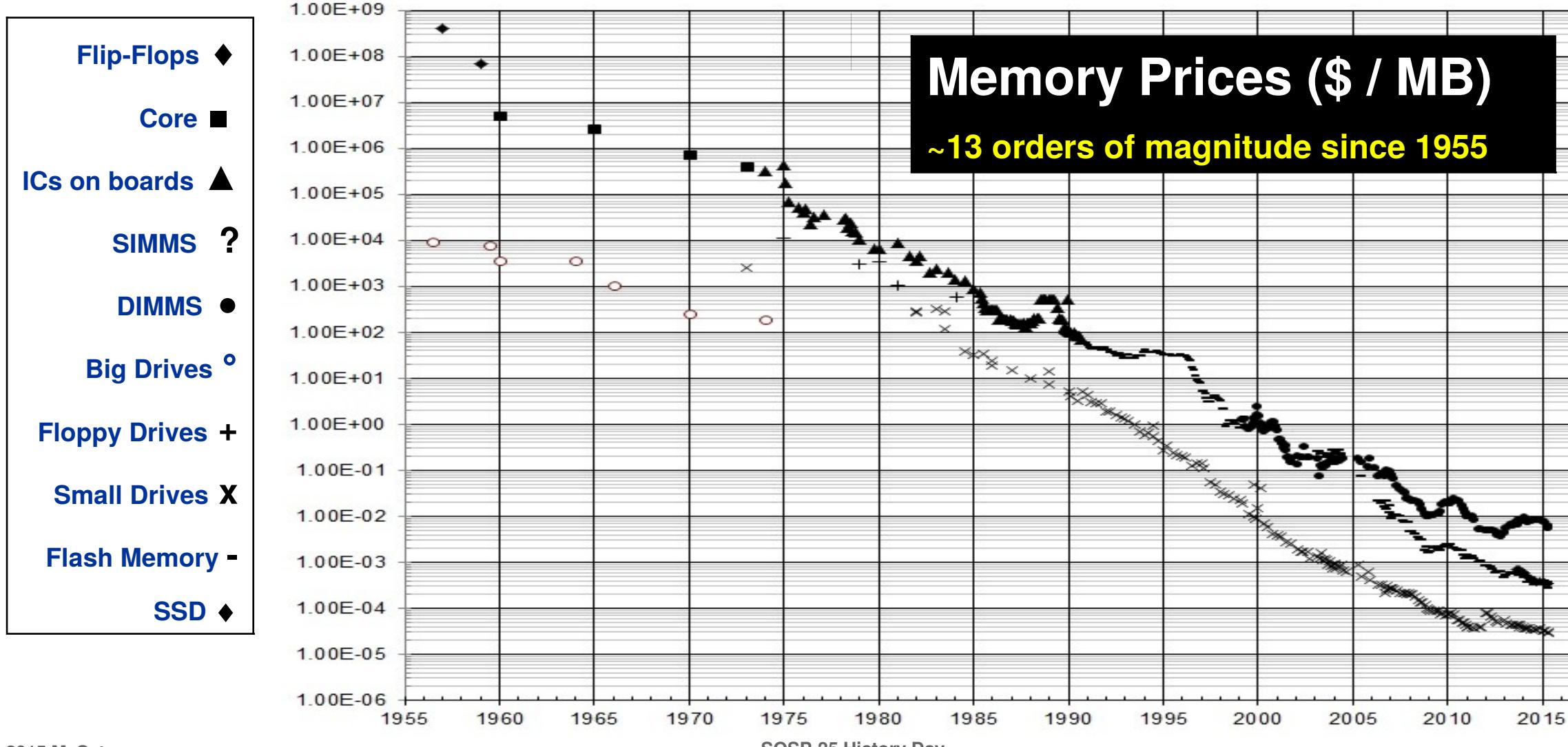
- The quest for *scale*** *from early 1950s*
- The quest for *speed*** *from early 1950s*
- The quest for *transparency*** *from early-1960s*
- The quest for *robustness*** *from mid- to late-1960s*
(both system and human errors)

Complex Interactions

The Quest for Scale

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)

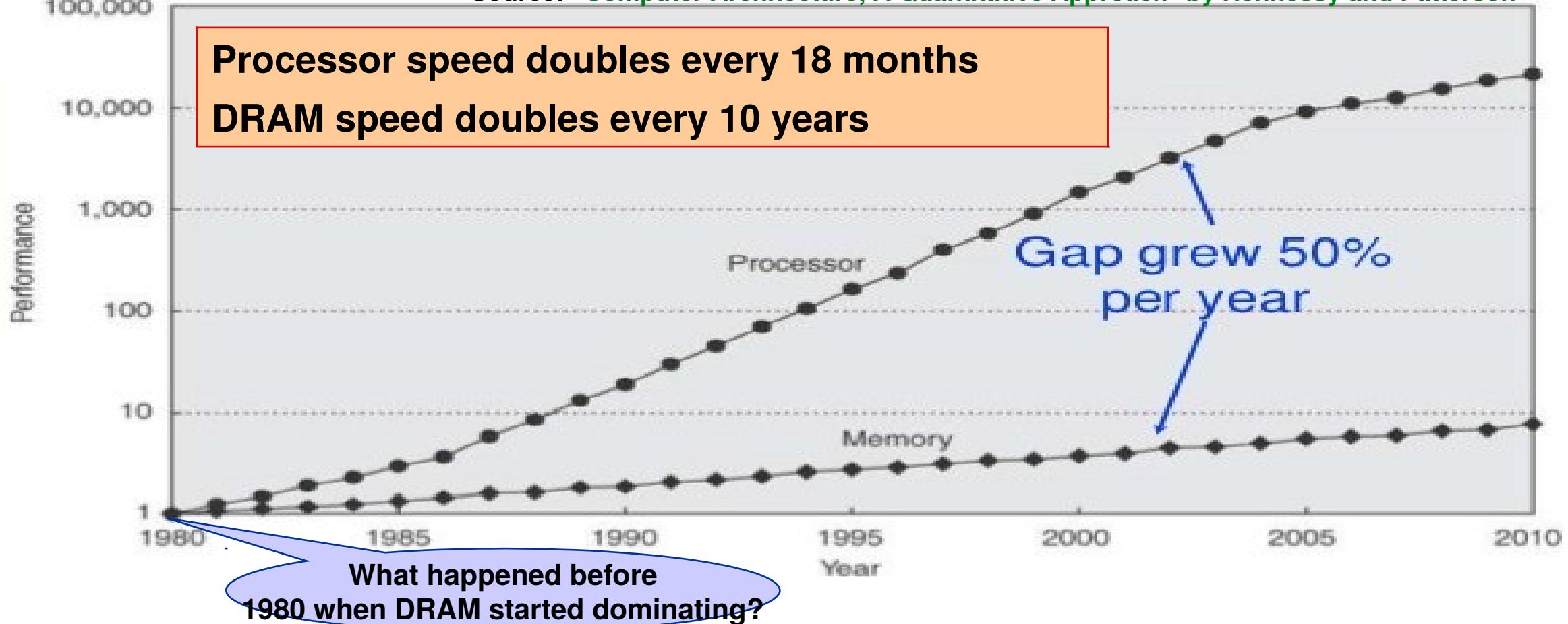
Capability-based

- **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!)

The Quest for Speed

Processor-Memory Speed Gap

Source: "Computer Architecture, A Quantitative Approach" by Hennessy and Patterson



Before 1980

IBM System/360

Source: [Wikipedia](#)

Model	Shipped	Scientific Performance (KIPS)	Commercial Performance (KIPS)	CPU Bandwidth (MB/s)	Memory Bandwidth (MB/s)	Memory Size (KB)
30	1965	10.2	29.0	1.3	0.7	8–64
40	1965	40.0	75.0	3.2	0.8	16–256
50	1965	133.0	169.0	8.0	2.0	64–512
65	1965	563.0	567.0	40.0	21.0	128–1024
75	1966	940.0	670.0	41.0	43.0	256–1024
91	1967	1900.0	1800.0	133.0	164.0	1024–4096

IBM System/370

Source: [Wikipedia](#)

Model	Shipped	Processor Cycle Time	Memory Access Time	Memory Size (KB)
155	1971	115 ns	2 ms	256–2048
165	1971	80 ns	2 ms	512–3072

Creating an Illusion of Scale and Speed

Memory hierarchies

- scale appears to be that of slower but more scalable technology
- speed appears to be that of faster but less scalable technology
- essentially probabilistic in character (worst case can be bad)

Working set characterizes the goodness of fit

Exploiting parallel data paths for increased bandwidth

- striping
- sharding
- bit-torrent, etc

Managing Data Across Levels

LRU and variants work amazingly well!

Alas, a few workloads defeat LRU

- ***purely sequential access*** → zero temporal locality
caching cannot help at all; only adds overhead
- ***purely random access*** → being smart is useless
ratio of cache size to total data size is all that matters
- **these access patterns are observed in the real world**
file scans in data mining, video/audio playback, hash-based data structures, ...

Multi-decade quest for improvement over LRU for these workloads

ARC: adaptive replacement cache (Megiddo & Modha 2003) best so far

The Quest for Transparency

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

⋮...

Some Transparency Landmarks

Caching (not overlays or other software-visible abstractions)

- *consistency of distributed caches*
- strict consistency vs. weak / eventual consistency

Shared memory in multiprocessor systems

- **UMA:** “uniform memory access” (e.g. C.mmp and many others)
- **NUMA:** “non-uniform memory access” (e.g. Cm* and many others)
- **NORMA:** “no remote memory access” (Berkeley NOW project, and others)

Distributed Shared Memory

- hot topic in 1990s; long dormant
- it is coming back! (OSDI 2012: COMET)

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

Caching is Universal



- Variable size more common
- More time for decision making
- More space for housekeeping
- More complex success criteria
- Less temporal locality
- Less spatial locality
- Higher cache advantage common

Caveat: these are “soft” differences

- Fixed size almost universal
- Fast, cheap decisions essential
- Miss ratio says it all (all misses equally bad)
- Greater temporal & spatial locality

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

Cache Consistency Strategies

emulate one-copy semantics of memory

Natural consequence of distribution + caching

Crucial dimension of transparency

Avoids changes to application software

Meets user expectations of system behavior

1. *Broadcast invalidations*
2. *Check on Use*
3. *Callbacks*
4. *Leases*
5. *Skip Scary Parts*
6. *Faith-based Caching*
7. *Pass the Buck*

Many variants over the years, but these lie at their core

The Quest for Robustness

Coping With System Failures

ECC memory

Erasure coding

RAID (including mirroring)

Bad-block mapping

Wear-leveling of flash storage

Data replication and disaster recovery

Disconnected operation

...

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?

Hot Topic Today

the death watch has begun

Hierarchical file systems are dead

Authors: [Margo Seltzer](#) [Harvard School of Engineering and Applied Sciences](#)
[Nicholas Murphy](#) [Harvard School of Engineering and Applied Sciences](#)

Published in:

- Proceeding
HotOS'09 Proceedings of the 12th conference on Hot topics in open
systems
Pages 1-1

USENIX Association Berkeley, CA, USA ©2009

[table of contents](#)

Every Page is Page One

Readers can enter anywhere. Is your content ready to receive them?

[Home](#) [Contact](#) [About](#) [The Book](#) [Publications](#) [Speaking](#) [Examples of EPPO topics](#) [!\[\]\(9068b7ca514b301ee74b1c4ea7ee5e7d_img.jpg\)](#)

The Death of Hierarchy

by [Mark Baker](#) on 2014/09/29 in Every Page is Page One, Hide and Seek, Writer vs. Reader

Hierarchy as a form of content organization is dying. A major milestone — I want to say tombstone — in its demise is the shutdown of the Yahoo directory, which will occur at the end of the year according to an article in Ars Technica, [Yahoo killing off Yahoo after 20 years of hierarchical organization](#). (Actually it seems to be offline already.)

[Take the Every Page is Page One Course!](#)

Learn to write in the Every Page is Page One style with a two-day course customized for your group's needs and using your own material for examples and exercises. Contact us for more information.

[Get the Book!](#)

The Cloud And the Death of the File System

Posted on [April 2, 2014](#)

One of the things I neglected to discuss in my eBook, [Web Development in the Cloud](#), was something that seemed so obvious to me that I simply missed including it. And that is the simple fact that if you develop web sites on the Cloud, you need to understand that the conventional file system process is dead.

Appears True at High Level

E.g. Android software focuses on Java classes and SQLite

- Android users never see a classic file system
- But, underneath the Dalvik VM, is the Linux native environment
- classic hierarchical file system continues to live on

This model may indeed become common

Will the lower layer vanish completely some day?

Not a New Viewpoint!

The Death of File Systems

by [JAKOB NIELSEN](#) on February 1, 1996

Topics: [Human Computer Interaction](#) [Predictions & Milestones](#) [Technology](#)

Summary: The file system has been a trusted part of most computers for many years, and will likely continue as such in operating systems for many more. However, several emerging trends in user interfaces indicate that the basic file-system model is inadequate to fully satisfy the needs of new users, despite the flexibility of the underlying code and data structures.

Originally published as: 145. Nielsen, J. (1996). The impending demise of file systems. IEEE Software 13, 2 (March).

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.”*

Near-Decomposability

Key property of human-created hierarchical systems (Simon 1962)

Consequence of human cognitive limitations

Allows focus on immediate neighborhood (current directory + children)

- *apparent shrinking of scale*
- valuable to exploit in achieving scalability
- exploitable in concurrency control, failure resiliency, consistency, etc.

Hierarchical file systems reflect the limitations of human cognition

- without external tools, that's the best organization for human minds
- “external tools”: e.g., SQL databases and search engines

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) →

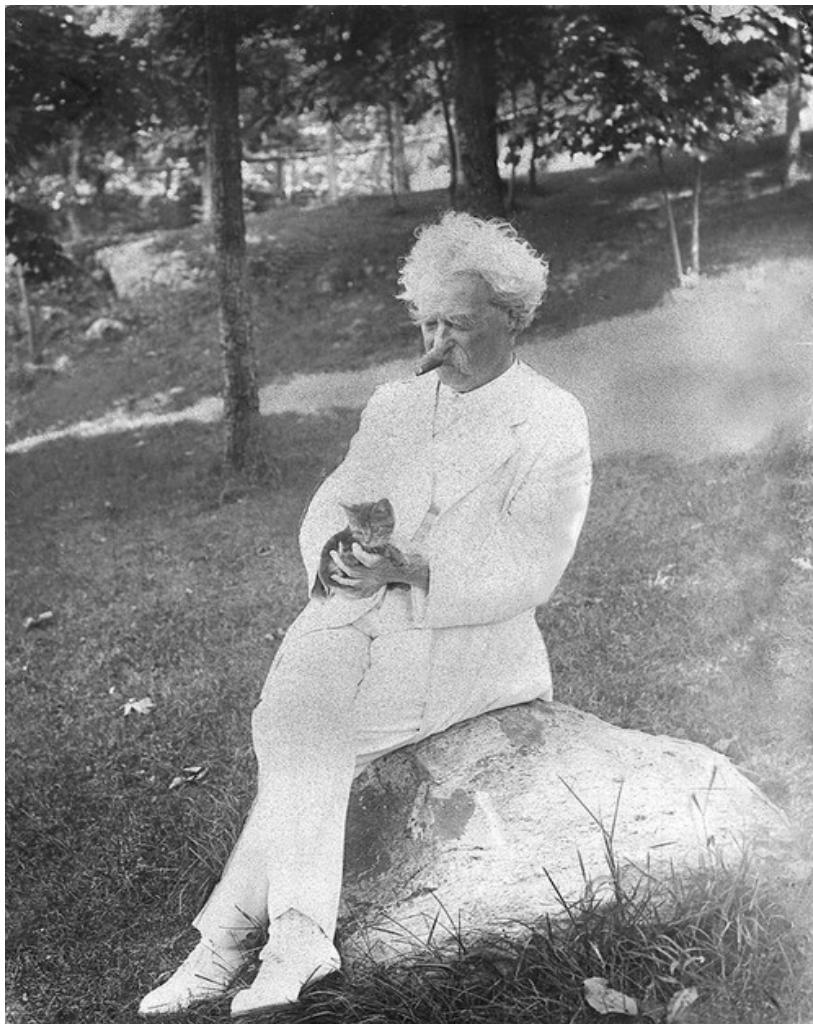
- even the features for indexing may be unclear
- manual exploration may be necessary

Need for manual exploration (even if rare) →

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

The Death of File Systems?

“... report of my death was an exaggeration”



The report of my illness
grew out of his illness, the
report of my death was
an exaggeration.

Mark Twain

—

Fault Tolerance

Ken Birman

Too many seminal concepts

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



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



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



Theory

- Consensus $\Diamond W$: consensus
- FLP



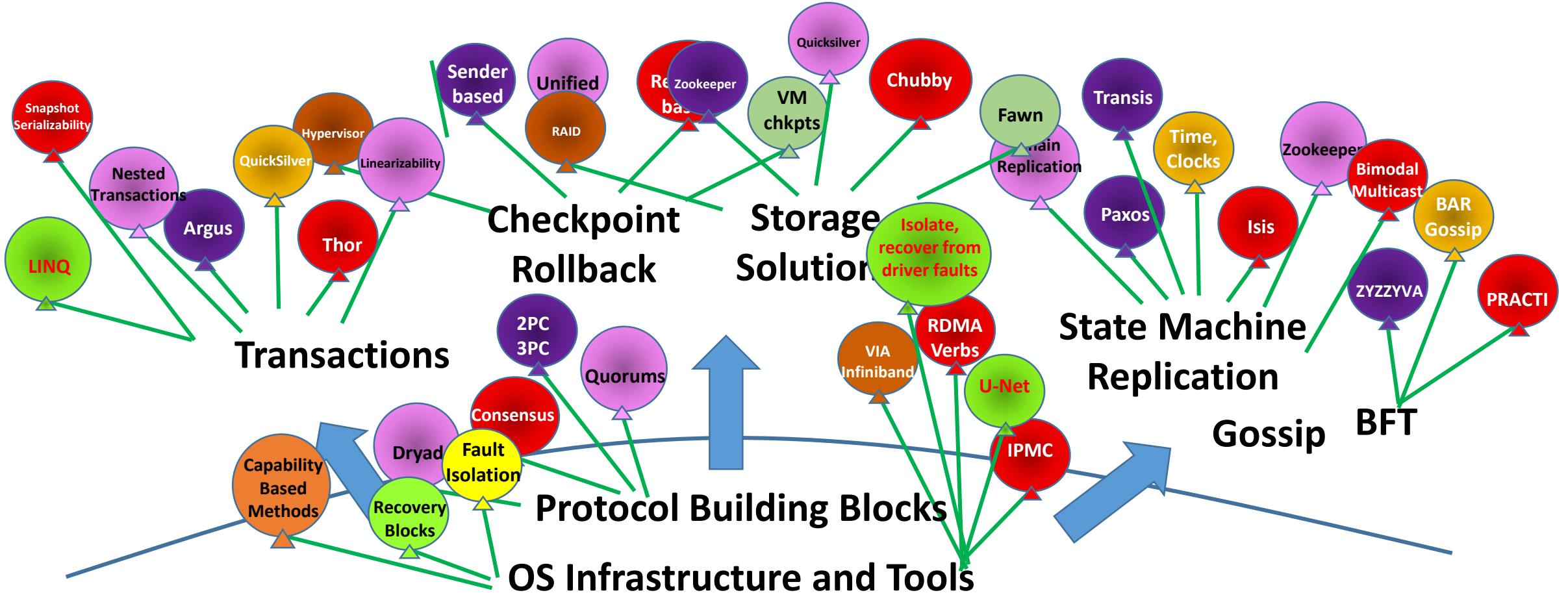
... Skepticism

- CATOCS



CAP





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

Fault-Tolerance via Replication: Rich History

- Early debate about the question itself
 - *Some believed that the OS layer is the wrong place to offer strong properties...*
- Today that debate has reemerged:
 - *Some believe that the cloud can't afford strong properties!*

Theory

Basic questions

- What sort of system are we talking about?
- What do we mean by “failure”?
- What does “tolerating” mean?

Thinking of Fault-Tolerance in terms of Safety

- Consistent State: A system-specific invariant: $\text{Pred}(S)$
- S is fault-tolerant if:
 S maintains/restores $\text{Pred}(S)$ even if something fails
- Normally, we also have *timeliness* requirements.

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

Systems

Principles from the systems side...

- Make core elements as simple as possible
 - *Pare down, optimize the critical path*
 - Captures something fundamental about systems.



Butler

- Generalized End-to-End argument:
 - Let the application layer pick its own models.
 - Limit core systems to fast, flexible building blocks.



Jerry



Dave¹⁰

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.

Gray: How do systems really fail?



Jim Gray

- Studied Tandem's “non-stop” platforms

*Failures caused by bugs, user mistakes, poor designs.
Few hardware failures, and nothing malicious.*

- Jim's advice? Focus our efforts on the real needs

Tensions

Why aren't existing OS mechanisms adequate?

Is fault-tolerance / consistency too complex or costly?

Do the needed mechanisms enable or impose models?

Do we need fault-tolerant replication?



- Not just for making systems tolerant of failures
 - Cloud computing: Provision lots of servers
 - Performance-limiting for many machine-learning systems
- So we agree, hopefully: replication is awesome!
- But is there a core OS mechanism here?

It comes down to performance and scalability

- As systems researchers, abstracted properties are...
 - Useful when designing and testing
 - Valuable tools for explaining behavior to users
 - Not obstacles: “Impossible” problems don’t scare us...
- Performance is a more fundamental challenge
 - Can fault-tolerance mechanisms be *fast*?

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

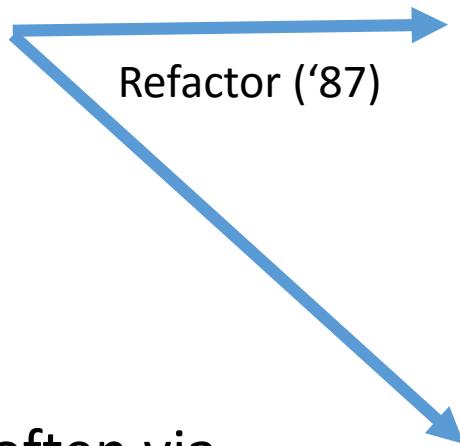
... Higher-level replication primitives?

- Isis: In 1985 used state machine replication on objects
 - Core innovation was its group membership model, which integrates membership dynamics with ordered multicast.
 - Durability tools: help application persist its state
- Paxos*: Implements state machine replication (1990)
 - A durable database of events (not an ordered multicast)
 - Runs in “quasi-static” groups.

Delays on the critical path: Isis

Original Isis Toolkit:

State machine replication of user-defined objects.
Durability was optional.



Paxos: Many optimizations, often via transformations like the Isis ones

But Paxos theory and formal methodology are very clean, elegant...

- Oracle
 - Uses quorums
 - Outputs “Views”
 - Bisimulates Paxos



- Critical Path
 - Asynchronous, pipelined
 - Flush when view changes
 - Only pay for properties used

Virtual Synchrony: Model + menu of choices
[Note: CATOCS controversy arose here...]

How does one speed such systems up?



- Start with simple, easily analyzed solution... Study the code
 - The critical paths often embody inefficiencies, like requesting total order for actions already in order, or that commute.
 - Often, synchronous events can be asynchronously pipelined
- Restructure critical paths to leverage your insights
 - Hopefully, the correctness argument still holds...

Pattern shared by Isis, Paxos, Zookeeper, Chain Replication, Zyzzyva, many others...

... 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

CATOCS: A case against consistent replication

- Too complex
- Violates End-to-End by imposing model on the user
- No matter what form of update order is supported, user won't like it
- Ordering is just too slow, won't scale



Dave Cheriton

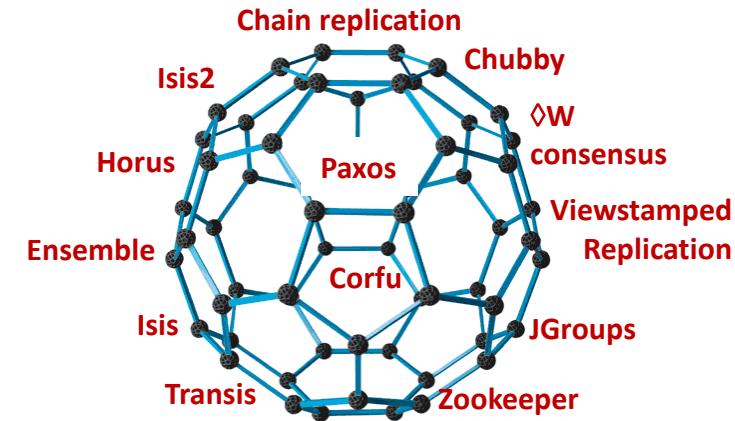


Dale Skeen

So were CATOCS claims true?

- Early replication solutions really were too slow.
 - Later ones were faster, but more complex.
- But CATOCS analysis of ordering was dubious.
- Yet... what about that missing low-level building block?
 - ... a puzzle (we'll come back to it later)

The “consensus” family...



- Can transform one to another... optimizations driven by desired properties.
- For me, durability remains puzzling
 - Is the goal durability of the application, or of its “state”?

... a few winners:

- State Machine Replication, Paxos, ACID transactions *Conceptual Tools*
-

- Chubby, Zookeeper, Corfu

Real Systems

- Primary + Warm backup... Chain Replication
-



Servers: 3-5 nodes



A cloud-hosted service could run on 5,000 nodes in each of dozens of data centers

Meanwhile, along came a cloud!

... 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



Is consistency just too costly?



Eric Brewer

- CAP: Two of {Consistency, Availability, Partition-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

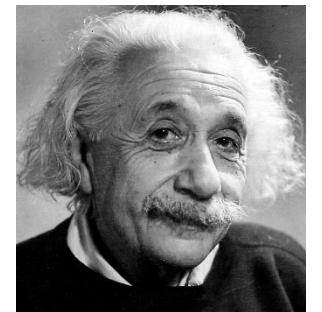
... but does CAP+BASE work?

- CAP folk theorem: “*don’t even try to achieve consistency.*”

... meaning what?

- “Anything goes”? “Bring it on?”

- Einstein: “A thing should be as simple as possible, but not simpler.”



... 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



A fascinating co-evolution



Edsger Dijkstra

- The cloud fits the need; the applications fit the cloud.
At first, fault-tolerance wasn't given much thought.
- Jim Gray : “*Why do systems fail?*”
- Today: *Why don't CAP+BASE systems fail?*
- Could we apply Dijkstra's theory of “self-stabilization” to BASE?

Future Shock: Disruption is coming

- Life and safety-critical cloud computing...
 - Smart power grid, homes, cities
 - Self-driving cars
 - Cloud-hosted banking, medical care
- *Weakened consistency won't suffice for these uses.*



Homework (due date: SOSP 2017)

- Start with a clean slate (but do learn from the past)
- Embrace a modern architecture
 - Cloud-scale systems...
 - Multicore servers with NVRAM storage
 - RDMA (like Tesla’s “insane speed” button).
- Propose a new approach to cloud-scale consistency



Future Cloud...



- The O/S has been an obstacle... even embraced inconsistency.
 - The future cloud should embrace *consistency*.
- Key: Elegance, speed, support real needs of real developers
- Need a core OS building block that works, integrated with developer tools and IDEs that are easy to use.

VIRTUALIZATION

Andrew Herbert

Cambridge University
ANSA
Microsoft Research
EDSAC Replica Project

DEFINITION (In context of SOSP)

- *Virtualization* is a property of operating systems that gives the illusion of efficiently running multiple independent computers known as *virtual machines*.
- The virtual machines be directly executed by (i.e., exactly mimic) the underlying physical machine, or they may comprise a more abstract system, parts, or all, of which are simulated by the physical machine.
- The part of the operating system that provides the virtual machine abstraction is commonly called a *virtual machine monitor* or *hypervisor*.

SOSP & VIRTUALIZATION

- Virtual Machine Monitors
- Layered Abstract Machines
- OS Process as a Virtual Environment
- Addressing Virtual Resources

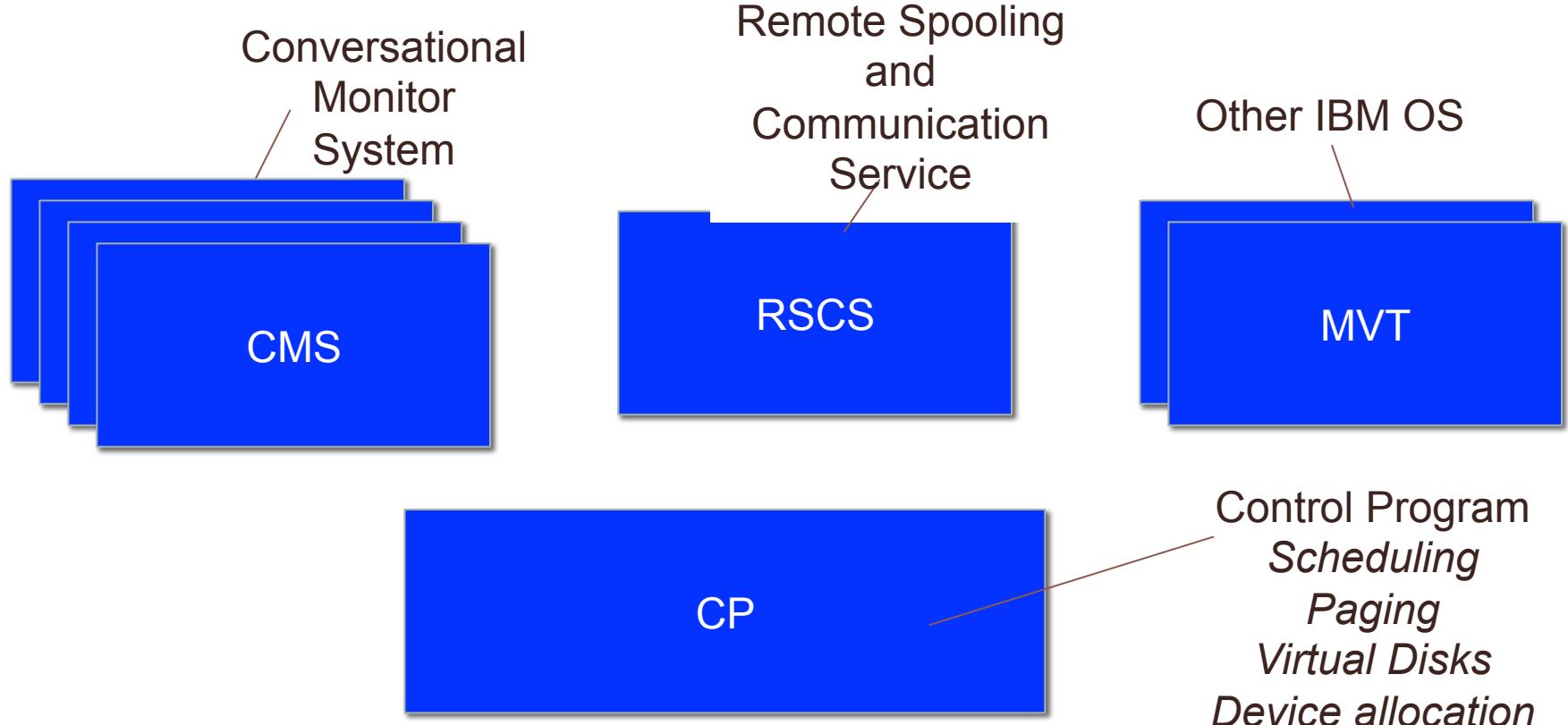
Note: referenced systems, papers are exemplars, not an exhaustive list.

VIRTUAL MACHINE MONITORS

- Origins in 1965 IBM M44/44X to explore page replacement algorithms
- M44/44X allowed each “virtual machine” to have its own paging strategy to allow measurement and comparison
- Evolved via CP/40 and CP/67 to VM/370

IBM VM/370

- Addressed three needs
 - Time-sharing computer utility
 - Running legacy applications and their operating systems alongside new applications and systems
 - Developing and debugging new operating systems using same time-sharing environment as for applications



VIRTUALIZATION APPROACHES

- Semi-formal model by Popek and Goldberg (1974)
 - Notion of *sensitive instructions* which reveal processor state
- IBM 360/370 enabled *pure* virtualization
 - Only *privileged* instructions in virtual supervisor mode have to be simulated by the VMM
- In contrast to *hybrid* virtualization
 - Some unprivileged instructions have to be simulated in virtual supervisor mode in addition
 - x86 only became pure with Intel VT-x / AMD SVM (2006)

1975-1995

1975-1995

- IBM VM/370 continues...
- Recursive virtual machines...
 - Lauer & Weyth, 1973
 - CAP (Needham & Walker, 1977)
- Then...

“SPECIAL EFFECT” VMMS

- *Hypervisor-based fault tolerance*
 - (Schneider & Bressoud 1996)
- *ReVirt: enabling intrusion analysis...*
 - (Dunlap, King, Cinar, Basrai & Chen 2002)

HOSTED VMMs (TYPE 2)

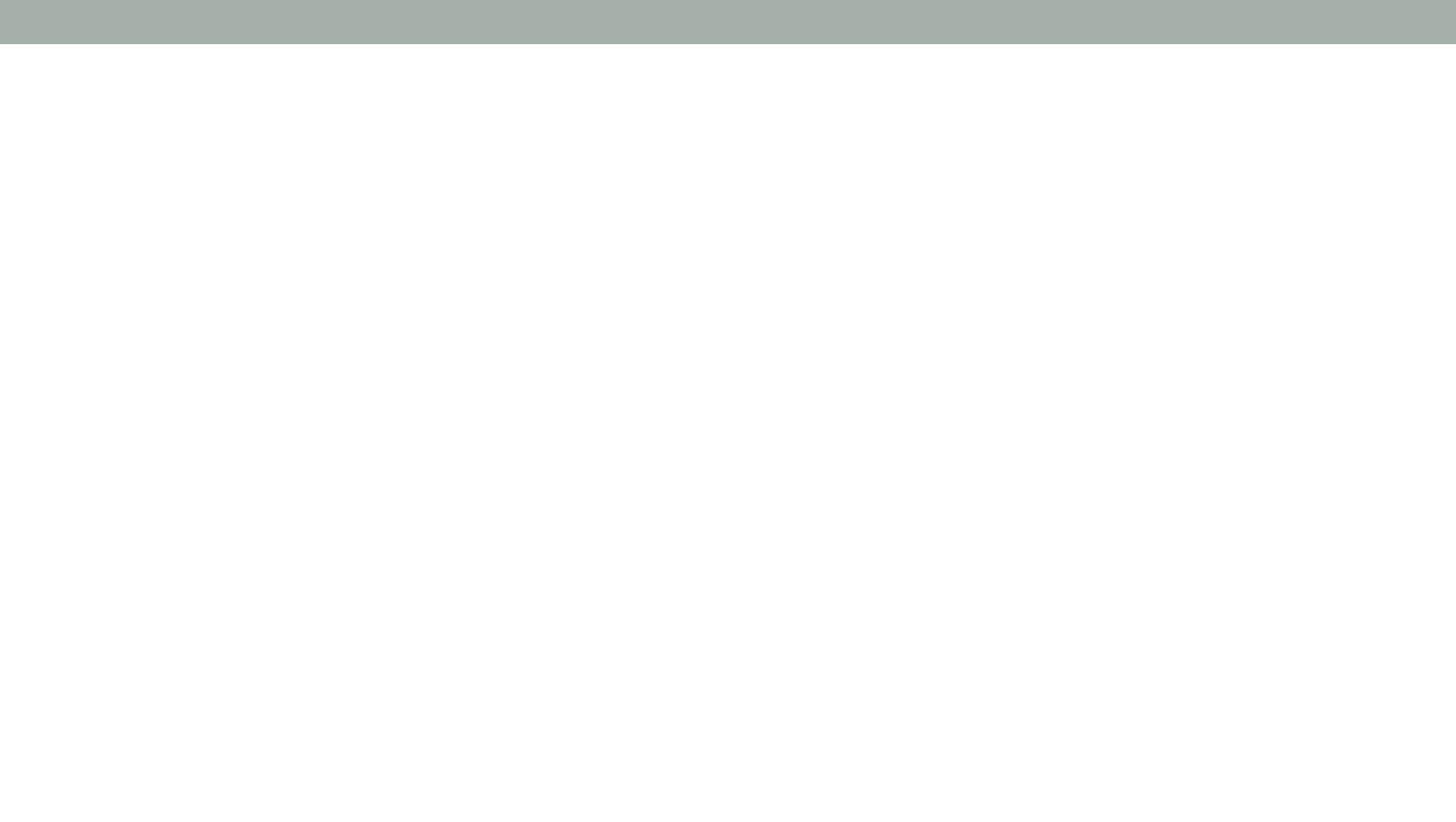
- Enable desktop OS coexistence
- VMM runs as app on host OS
 - Often with associated kernel driver
 - Uses host OS services (file, network)
 - VMs run guest OS image
- DEC-10 VMM for ITS (Galley, 1973)
- VMWare Desktop for Windows (1999)

NATIVE/BARE METAL VMMs (TYPE 1)

- DISCO (Buignon, Devine, et al. 1997)
 - *Hybrid virtualization with binary rewriting and shadowing in place of simulation*
- VMWare ESX Server (Waldspurger 2002)
 - *System-wide resource multiplexing*
 - *Ballooning, Content-based Page Sharing*
- Xen (Barham et al. 2003)
 - *Paravirtualization*

BENEFITS OF VIRTUAL MACHINES

- Desktop virtualization
 - OS coexistence
 - Desktop checkpoint/restore, migration
- Server virtualization
 - Server consolidation
 - Multi-tenancy with strong isolation
 - Statistical multiplexing of resources across multiple virtual servers
 - Management framework for server workloads



LAYERED ABSTRACT MACHINES

- Design and implement an operating system as a hierarchical series of abstract machines
- *'THE' multiprogramming system* (Dijkstra, 1967)
 - Layer 0: concurrency and interrupt handling (processes and semaphores)
 - Layer 1: automatic memory paging
 - Layer 2: operator's console
 - Layer 3: input-output
 - Layer 5: Algol batch system

WHY LAYER?

- Build and test layer by layer
- Structured (informal) proof of correctness
- 1972, Liskov, Venus
- 1980s, Comer, XINU

LAYERING FOR SECURITY

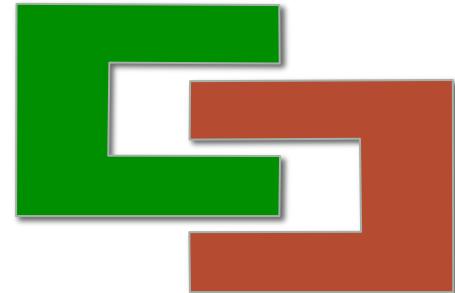
- US DoD Trusted Systems – Orange Book
 - A1 – Verified design
 - B2 – Structured protection
 - B3 – Security domains (reference monitor)
- *The foundations of a provably secure operating system (PSOS)* (Feiertag, Neumann, SRI, 1979)
 - Hierarchical Design Methodology, SPECIAL
 - 17 Nominal layers for verification, collapsed to 9 in the implementation

MULTICS KERNEL DESIGN

- 1977, MIT redesign and reimplementation of Multics to meet B2/B3 criteria (Schroeder, Clark & Saltzer, 1977)
- Added additional layers to system in order to remove code from the Multics supervisor so it could be reduced to be a reference monitor suitable for inspection
- *Type Extension*: treat each module as a *type manager*
 - Ensure type dependency graph does not contain cycles
 - Often had to split original modules into upper and lower types

LAYERS AND MODULES

- 1976, Haberman et al., FAMOS
 - *Modularization and hierarchy in a family of operating systems*
 - Set of components for building a family of (related) operating systems
 - Explored conflict between *layers* and *modules*
 - Similar model to Multics *type extension*

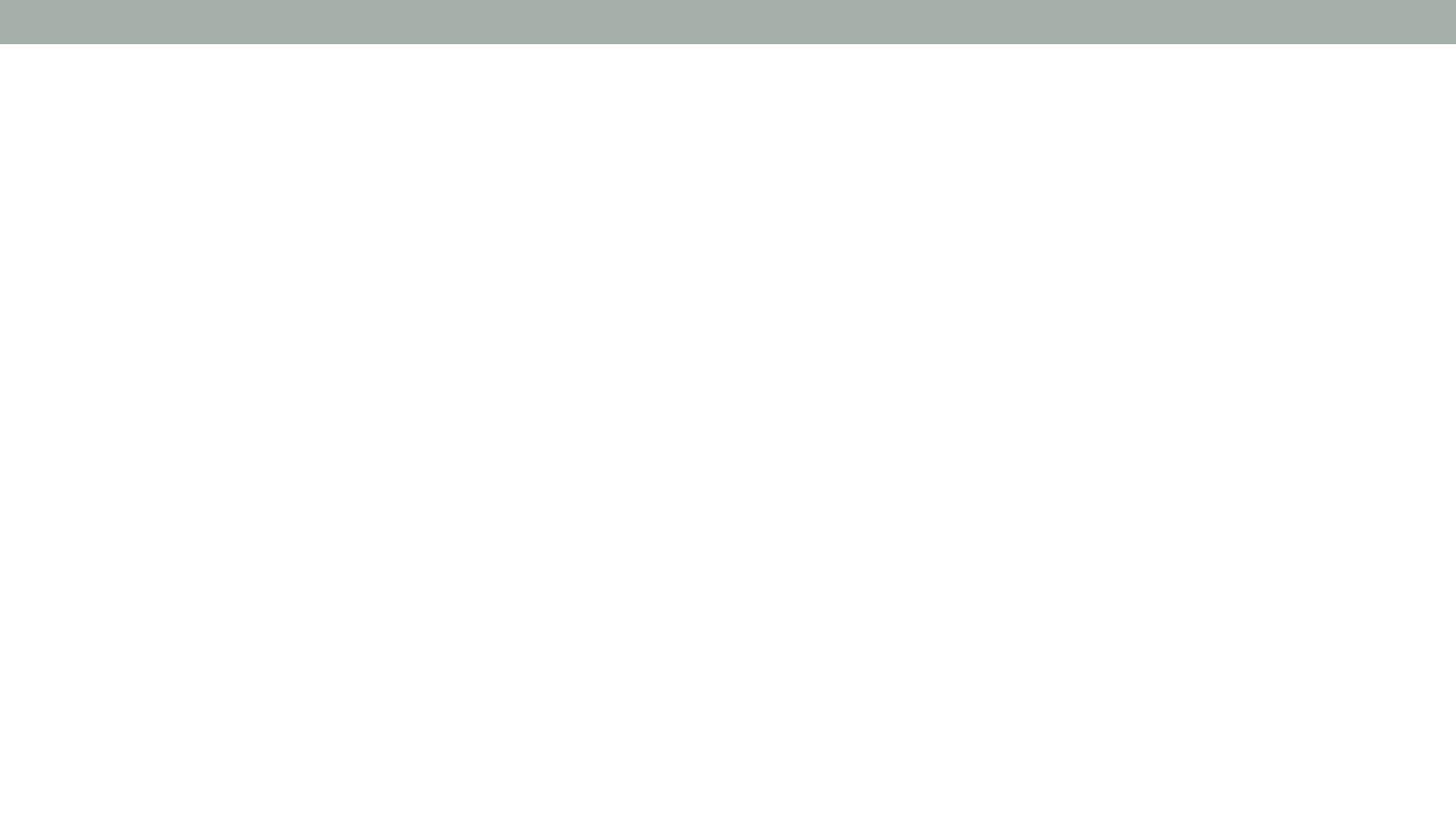


LAYERING AND NETWORKS

- Abstraction layering is not unique to systems
- Protocols defined as interacting peer entities at increasing levels of abstraction
- But Open Systems Interconnection 7-layer model regarded as overblown
- Virtualization by layering
 - Virtual Private Networks
 - Virtual LANs

OBSERVATIONS

- Most of these systems were done by people with a background in programming languages and formal specification and verification
- Some claim layering leads to inefficiency but XINU gives good evidence otherwise



VIRTUAL ENVIRONMENTS

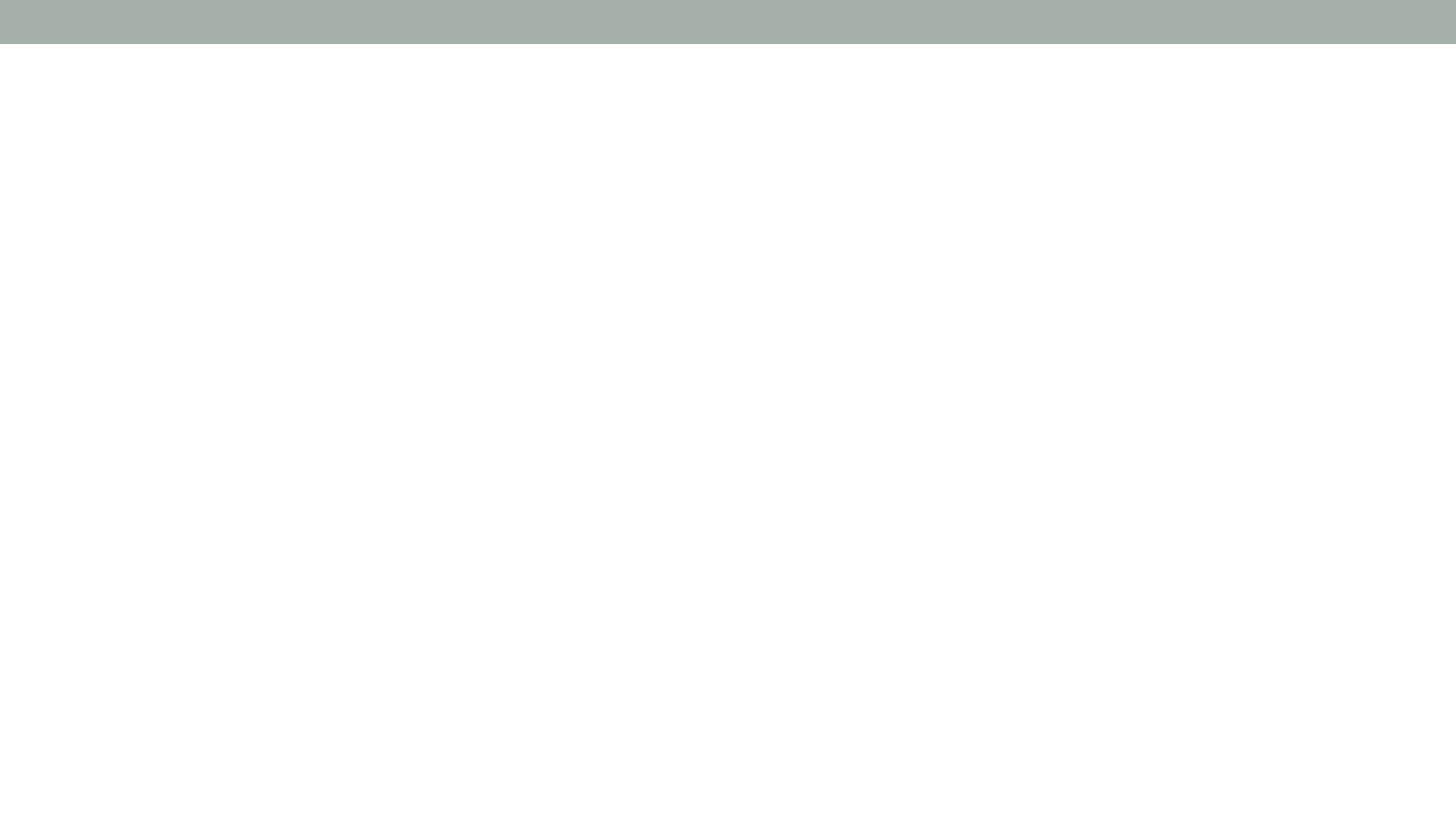
- A process is a program executing in a virtual environment (Saltzer, 1966)
- 1950s & 60s – Tyranny of the instruction set
 - Each new machine required everything to be re-written
 - Emulate older machines e.g., IBM 360 – IBM 1400
- Atlas *Extracodes*
 - Undefined instructions that execute subroutines in fast memory – e.g., supervisor calls, library functions

LIBRARIES

- 1970s Emergence of practical general purpose / systems programming languages
- 1980s Unix marked transition to the standard library becoming the virtual environment
 - High level abstract interface for applications
 - Low level concrete interface for the library
- UNIX (via Multics) gave us the byte stream as the universal *virtual device* abstraction replacing messy record-oriented structures of earlier systems

DISTRIBUTED UNIX

- Distributed *single image* implementations of UNIX virtual environment didn't last, e.g. Locus (Popek, 1981)
 - Caching/Scaling problems (see Satya on file systems)
 - Autonomy of network of workstations model outweighed benefits of centralized management



NAMING VIRTUALIZED RESOURCES

- Useful to have a uniform context [location] independent way of naming sharable [virtual] resources
- Capabilities (see Lampson for security aspects)
 - C-lists (Dennis, Van Horn 1966)
 - *Capability-based addressing* (Fabry 74)
 - *Amoeba* (Tanenbaum et al. 1986)
- Grew out of Codewords / descriptors
 - Rice R1, R2 (1959-71), B5000 (1963)
 - Tagged Architectures (Feustal 1972)

PLAN 9

- Bell Labs 1991
 - Every resource is a file
 - Name spaces are mountable, stackable, ...

ACCESSING VIRTUAL RESOURCES

- Lots of arguments about how to access virtual resources
- See Kaashoek, Liskov on threads vs events
- Custom protocol vs optimized RPC vs TCP/IP
- Active versus passive objects (OMG CORBA)
- Workstations versus Processor Banks
 - PARC, MIT Athena, CMU Andrew, Cambridge Distributed System
 - GUI windows as virtual resources, e.g. X window system

TALKING POINTS

- More mileage from using virtualization to modify operating system semantics?
- VMM ≠ TCB, especially for type 2
- Revisit layered abstract machines as a systems design and implementation principle?
- Prove systems correct by default?
 - seL4 (Klein, Elpinstone, Heiser, Andronick, ..., 2009)
 - Verve (Yang & Hawblitzel, 2010)



Past and Future Trends in Architecture and Hardware

David Patterson

pattrsn@eecs.berkeley.edu

SOSP History Day October 3, 2015



Outline

Part I - Past

50 years of Computer Architecture History:

- 1960s:
Computer Families /
Microprogramming
- 1970s: CISC
- 1980s: RISC
- 1990s: VLIW
- 2000s: NUMA vs.
Clusters

Part II – Future

HW Technology

- End of Moore's Law
- Flash vs. Disks
- Fast DRAM
- Crosspoint NVRAM
- Open ISA & RISC-V
 - Case for Open ISAs
 - Tour of RISC-V ISA
 - RISC-V Software Stack
 - RISC-V Chips

IBM Compatibility Problem in early 1960s

By early 1960's, IBM had
4 incompatible lines of computers!

701	→	7094
650	→	7074
702	→	7080
1401	→	7010



Each system had its own

- Instruction set
- I/O system and Secondary Storage:
magnetic tapes, drums and disks
- Assemblers, compilers, libraries, ...
- Market niche: business, scientific, real time, ...

⇒ *IBM System/360 – one ISA to rule them all*



IBM 360: A Computer Family

	<i>Model 30</i>	...	<i>Model 70</i>
<i>Storage</i>	8K - 64 KB		256K - 512 KB
<i>Datapath</i>	8-bit		64-bit
<i>Circuit Delay</i>	30 nsec/level		5 nsec/level
<i>Registers</i>	Main Store		Transistor Registers

The IBM 360 is why bytes are 8-bits long today!

IBM 360 instruction set architecture (ISA) completely hid the underlying technological differences between various models.

Milestone: The first true ISA designed as portable hardware-software interface!

With minor modifications it still survives today!



IBM System/360 Reference Card ("Green card")

IBM System/360 Reference Data			
MACHINE INSTRUCTIONS			
NAME	MNEMONIC	OP	FOR- OPERANDS
Add (c)	AR	1A	RR R1,R2
Add (c)	A	5A	RX R1,D2(X2,B2)
Add Decimal (c,d)	AP	FA	SS D1(L,B1),D2(L2,B2)
Add Halfword (c)	AH	4A	RX R1,D2(X2,B2)
Add Logical (c)	ALR	1E	RR R1,R2
Add Logical (c)	AL	5E	RX R1,D2(X2,B2)
AND (c)	NR	14	RR R1,R2
AND (c)	N	54	RX R1,D2(X2,B2)
AND (c)	NI	94	SI D1(B1),I2
AND (c)	NC	D4	SS D1(L,B1),D2(L2,B2)
Branch and Link	BALR	05	RR R1,R2
Branch and Link	BAL	45	RX R1,D2(X2,B2)
Branch and Store (e)	BASR	0D	RR R1,R2
Branch and Store (e)	BAS	4D	RX R1,D2(X2,B2)
Branch on Condition	BCR	07	RR M1,R2
Branch on Condition	BC	47	RX M1,D2(X2,B2)
Branch on Count	BCTR	06	RR R1,R2
Branch on Count	BCT	46	RX R1,D2(X2,B2)
Branch on Index High	BXH	86	RS R1,R3,D2(B2)
Branch on Index Low or Equal	BXLE	87	RS R1,R3,D2(B2)
Compare (c)	CR	19	RR R1,R2
Compare (c)	C	59	RX R1,D2(X2,B2)
Compare Decimal (c,d)	CP	F9	SS D1(L,B1),D2(L2,B2)
Compare Halfword (c)	CH	49	RX R1,D2(X2,B2)
Compare Logical (c)	CLR	1B	RR R1,R2
Compare Logical (c)	CL	55	RX R1,D2(X2,B2)
Compare Logical (c)	CLC	D5	SS D1(L,B1),D2(B2)
Compare Logical (c)	CLI	95	SI D1(B1),I2
Convert to Binary	CVB	4F	RX R1,D2(X2,B2)
Convert to Decimal	CVD	4E	RR R1,D2(X2,B2)
Diagnose (p)		83	SI
Divide	DR	1D	RR R1,R2
Divide	D	5D	RX R1,D2(X2,B2)
Divide Decimal (d)	DP	F'D'	SS D1(L,B1),D2(L2,B2)
Edit (c,d)	ED	DE	SS D1(L,B1),D2(B2)
Edit and Mark (c,d)	EDMK	DF	SS D1(L,B1),D2(B2)
Exclusive OR (c)	XR	17	RR R1,R2
Exclusive OR (c)	X	57	RX R1,D2(X2,B2)
Exclusive OR (c)	XI	97	SI D1(B1),I2
Exclusive OR (c)	XC	D7	SS D1(L,B1),D2(B2)
Execute	EX	44	RX R1,D2(X2,B2)
Halt I/O (c,p)	HIO	9E	SI D1(B1)
Insert Character	IC	43	RX R1,D2(X2,B2)
Insert Storage Key (a,p)	ISK	09	RR R1,R2
Load	LR	18	RR R1,R2
Load	L	50	RX R1,D2(X2,B2)
Load Address	LA	41	RX R1,D2(X2,B2)
Load and Test (c)	LTR	12	RR R1,R2
Load Complement (c)	LOR	13	RR R1,R2
Load Halfword	LH	48	RX R1,D2(X2,B2)
Load Multiple	LM	98	RS R1,R3,D2(B2)
Load Multiple Control (e,p)	LMC	B9	RS R1,R3,D2(B2)
Load Negativs (c)	LNR	11	RR R1,R2
Load Positive (c)	LPR	10	RR R1,R2
Load PSW (n,p)	LPSW	82	SI D1(B1)
Load Real Address (c,e,p)	LRA	B1	RX R1,D2(X2,B2)
Move	MVI	92	SI D1(B1),I2
Move	MVC	D2	SS D1(L,B1),D2(B2)
Move Numerics	MVN	D1	SS D1(L,B1),D2(B2)
Move with Offset	MVO	F1	SS D1(L,B1),D2(L2,B2)
Move Zones	MVZ	D3	SS D1(L,B1),D2(B2)
Multiply	MR	1C	RR R1,R2
Multiply	M	5C	RX R1,D2(X2,B2)
Multiply Decimal (d)	MP	FC	SS D1(L,B1),D2(L2,B2)
Multiply Halfword	MH	4C	RX R1,D2(X2,B2)
OR (c)	OR	16	RR R1,R2
OR (c)	O	56	RX R1,D2(X2,B2)
OR (c)	OI	96	SI D1(B1),I2

FLOATING-POINT FEATURE INSTRUCTIONS			
OR (c)	OC	D6	SS D1(L,B1),D2(B2)
Pack	PACK	F2	SS D1(L,B1),D2(L2,B2)
Read Direct (b,p)	RDD	85	SI D1(B1),I2
Set Program Mask (n)	SPM	04	RR R1
Set Storage Key (a,p)	SSK	08	RR R1,R2
Set System Mask (p)	SSM	80	SI D1(B1)
Shift Left Double (c)	SLDA	8F	RS R1,D2(B2)
Shift Left Double Logical	SLDL	8D	RS R1,D2(B2)
Shift Left Single (c)	SLA	88	RS R1,D2(B2)
Shift Left Single Logical	SLL	88	RS R1,D2(B2)
Shift Right Double (c)	SRDA	8E	RS R1,D2(B2)
Shift Right Double Logical	SRDL	8C	RS R1,D2(B2)
Shift Right Single (c)	SRA	8A	RS R1,D2(B2)
Shift Right Single Logical	SRL	88	RS R1,D2(B2)
Start I/O (c,p)	SIO	9C	SI D1(B1)
Store	ST	50	RX R1,D2(X2,B2)
Store Character	STC	42	RX R1,D2(X2,B2)
Store Halfword	STH	40	RX R1,D2(X2,B2)
Store Multiple	STM	90	RS R1,R3,D2(B2)
Store Multiple Control (e,p)	STMC	80	RS R1,R3,D2(B2)
Subtract (c)	SR	1B	RR R1,R2
Subtract (c)	S	5B	RX R1,D2(X2,B2)
Subtract Decimal (c,d)	SP	FB	SS D1(L,B1),D2(L2,B2)
Subtract Halfword (c)	SH	48	RX R1,D2(X2,B2)
Subtract Logical (c)	SLR	1F	RR R1,R2
Subtract Logical (c)	SL	5F	RX R1,D2(X2,B2)
Supervisor Call	SVC	0A	RR I
Test and Set (c)	TS	93	SI D1(B1)
Test Channel (c,p)	TCH	9F	SI D1(B1)
Test I/O (c,p)	TIO	9D	SI D1(B1)
Test under Mask (c)	TM	91	SI D1(B1),I2
Translate	TR	DC	SS D1(L,B1),D2(B2)
Translate and Test (c)	TRT	DD	SS D1(L,B1),D2(B2)
Unpack	UNPK	F3	SS D1(L,B1),D2(L2,B2)
Write Direct (b,p)	WRD	84	SI D1(B1),I2
Zero and Add (c,p)	ZAP	F8	SS D1(L,B1),D2(L2,B2)

NOTES FOR PANELS 1-3			
a. Protection feature	d. Decimal feature	e. Model 67	code is loaded
b. Direct control feature	f. New condition	g. Privileged instruction	code is loaded
c. Condition code is set	h. Extended precision	i. Floating point	j. Floating point feature

MACHINE FORMATS															
FIRST HALFWORD 1	SECOND HALFWORD 2	THIRD HALFWORD 3													
<table border="1"> <tr> <td colspan="2">REGISTER OPERAND 1</td> <td colspan="2">REGISTER OPERAND 2</td> </tr> <tr> <td>RR</td> <td>Op Code</td> <td>R1</td> <td>R2</td> </tr> <tr> <td>0</td> <td>7,8</td> <td>1112</td> <td>15</td> </tr> </table>				REGISTER OPERAND 1		REGISTER OPERAND 2		RR	Op Code	R1	R2	0	7,8	1112	15
REGISTER OPERAND 1		REGISTER OPERAND 2													
RR	Op Code	R1	R2												
0	7,8	1112	15												
<table border="1"> <tr> <td colspan="2">REGISTER OPERAND 1</td> <td colspan="2">ADDRESS OF OPERAND 2</td> </tr> <tr> <td>RR</td> <td>Op Code</td> <td>R1</td> <td>R2</td> </tr> <tr> <td>0</td> <td>7,8</td> <td>1112</td> <td>1516</td> </tr> </table>				REGISTER OPERAND 1		ADDRESS OF OPERAND 2		RR	Op Code	R1	R2	0	7,8	1112	1516
REGISTER OPERAND 1		ADDRESS OF OPERAND 2													
RR	Op Code	R1	R2												
0	7,8	1112	1516												
<table border="1"> <tr> <td colspan="2">REGISTER OPERAND 1</td> <td colspan="2">REGISTER OPERAND 3 ADDRESS OF OPERAND 2</td> </tr> <tr> <td>RX</td> <td>Op Code</td> <td>R1</td> <td>R2</td> </tr> <tr> <td>0</td> <td>7,8</td> <td>1112</td> <td>1516 1920</td> </tr> </table>				REGISTER OPERAND 1		REGISTER OPERAND 3 ADDRESS OF OPERAND 2		RX	Op Code	R1	R2	0	7,8	1112	1516 1920
REGISTER OPERAND 1		REGISTER OPERAND 3 ADDRESS OF OPERAND 2													
RX	Op Code	R1	R2												
0	7,8	1112	1516 1920												
<table border="1"> <tr> <td colspan="2">REGISTER OPERAND 1</td> <td colspan="2">ADDRESS OF OPERAND 2</td> </tr> <tr> <td>RS</td> <td>Op Code</td> <td>R1</td> <td>R2</td> </tr> <tr> <td>0</td> <td>7,8</td> <td>1112</td> <td>1516 1920</td> </tr> </table>				REGISTER OPERAND 1		ADDRESS OF OPERAND 2		RS	Op Code	R1	R2	0	7,8	1112	1516 1920
REGISTER OPERAND 1		ADDRESS OF OPERAND 2													
RS	Op Code	R1	R2												
0	7,8	1112	1516 1920												
<table border="1"> <tr> <td colspan="2">IMMEDIATE OPERAND</td> <td colspan="2">ADDRESS OF OPERAND 1</td> </tr> <tr> <td>SI</td> <td>Op Code</td> <td>I2</td> <td>B1</td> </tr> <tr> <td>0</td> <td>7,8</td> <td>1516</td> <td>1920</td> </tr> </table>				IMMEDIATE OPERAND		ADDRESS OF OPERAND 1		SI	Op Code	I2	B1	0	7,8	1516	1920
IMMEDIATE OPERAND		ADDRESS OF OPERAND 1													
SI	Op Code	I2	B1												
0	7,8	1516	1920												
<table border="1"> <tr> <td colspan="2">LENGTH OPERAND 1</td> <td colspan="2">LENGTH OPERAND 2 ADDRESS OF OPERAND 1 ADDRESS OF OPERAND 2</td> </tr> <tr> <td>SS</td> <td>Op Code</td> <td>L1</td> <td>B1</td> </tr> <tr> <td>0</td> <td>7,8</td> <td>1516</td> <td>1920 3132 3536 47</td> </tr> </table>				LENGTH OPERAND 1		LENGTH OPERAND 2 ADDRESS OF OPERAND 1 ADDRESS OF OPERAND 2		SS	Op Code	L1	B1	0	7,8	1516	1920 3132 3536 47
LENGTH OPERAND 1		LENGTH OPERAND 2 ADDRESS OF OPERAND 1 ADDRESS OF OPERAND 2													
SS	Op Code	L1	B1												
0	7,8	1516	1920 3132 3536 47												

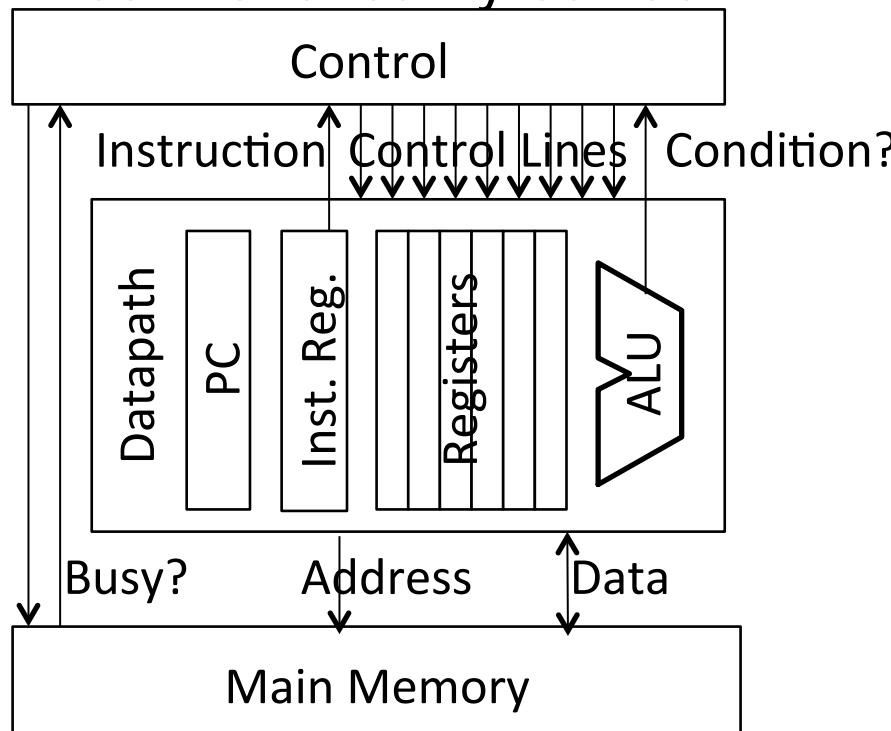
SUMMARY OF CONSTANTS

TYPE	IMPLIED LENGTH, BYTES	ALIGNMENT	FORMAT	TRUNCATION/PADDING
C	-	byte	characters	right
X	-	byte	hexadecimal digits	left
B	-	byte	binary digits	left
F	4	word	fixed-point binary	left
H	2	halfword	fixed-point binary	left
E	4	word	short floating-point	right
D	8	doubleword	long floating-point	right
L	16	doubleword	extended floating-point	right
P	-	byte	packed decimal	left
Z	-	byte	zoned decimal	left
A	4	word	value of address	left
Y	2	halfword	value of address	left
S	2	halfword	address in base-displacement form	-
V	4	word	externally defined address value	left
Q*	4	word	symbol naming a DXD or DSECT	left

*OS only

Control versus Datapath

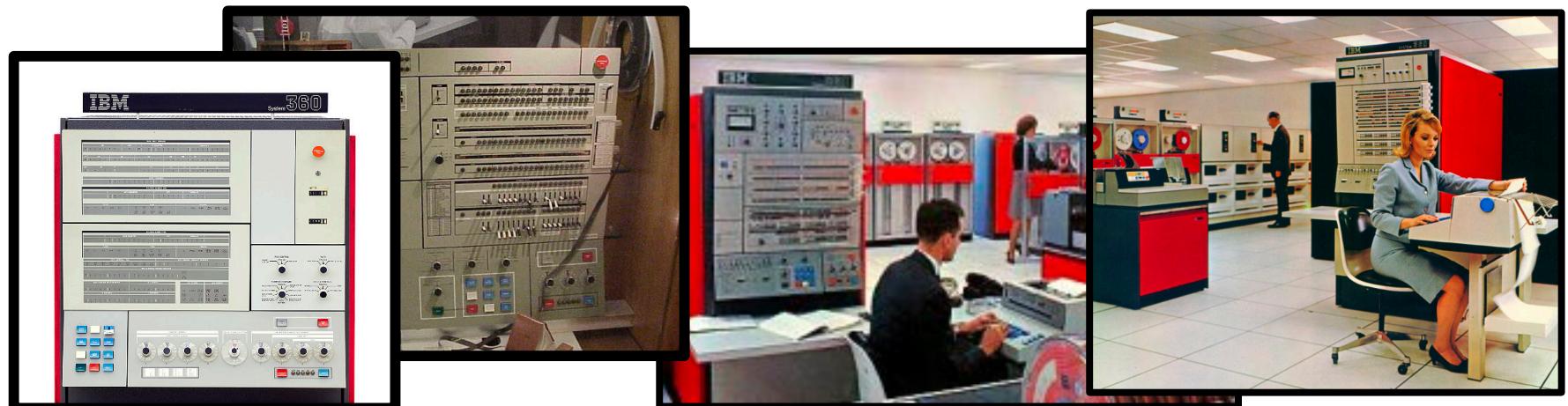
- Processor designs can be split between *datapath*, where numbers are stored and arithmetic operations computed, and *control*, which sequences operations on datapath
- Biggest challenge for early computer designers was getting control circuitry correct



- Maurice Wilkes invented the idea of *microprogramming* to design the control unit of a processor, 1958
 - Logic expensive compared to ROM or RAM
 - ROM cheaper than RAM
 - ROM much faster than RAM

Microprogramming in IBM 360

Model	M30	M40	M50	M65
Datapath width	8 bits	16 bits	32 bits	64 bits
Microcode size	4k x 50	4k x 52	2.75k x 85	2.75k x 87
Clock cycle time (ROM)	750 ns	625 ns	500 ns	200 ns
Main memory cycle time	1500 ns	2500 ns	2000 ns	750 ns
Annual rental fee (1964 \$)	\$48,000	\$54,000	\$115,000	\$270,000
Annual rental fee (2015 \$)	\$570,000	\$650,000	\$1,400,000	\$3,200,000





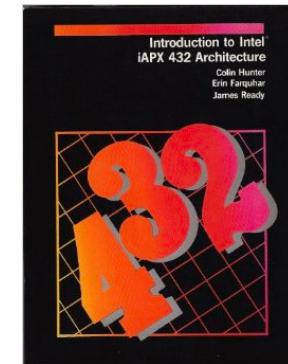
IC technology, Microcode, and CISC

- Logic, RAM, ROM all implemented using MOS transistors
- Semiconductor RAM ≈ same speed as ROM
- With Moore's Law, memory for control store could grow
- Allowed more complicated instruction sets (CISC)
- Minicomputer (TTL server)
Example:
Digital Equipment
VAX ISA in 1978



Microprocessor Evolution

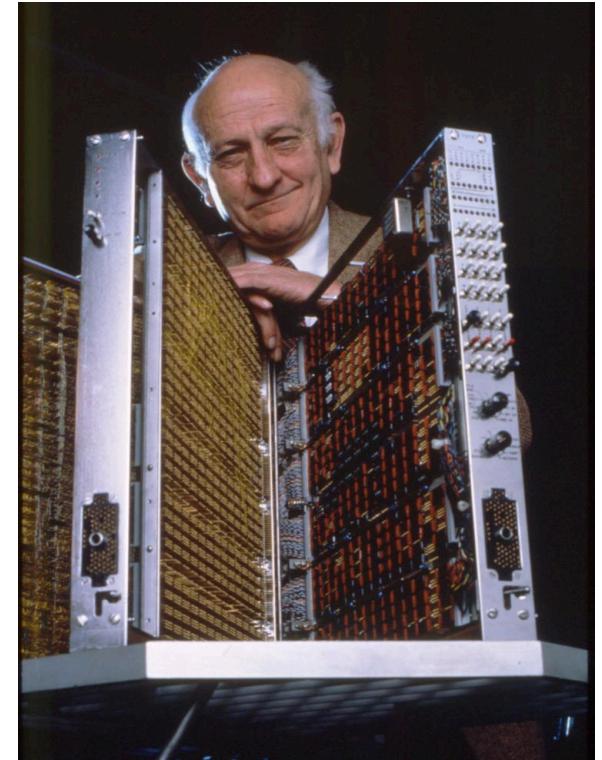
- Rapid progress in 1970s, fueled by advances in MOS technology, imitated minicomputers and mainframe ISAs
- Intel i432
 - Most ambitious 1970s micro
 - started in 1975 - released 1981
 - 32-bit capability-based object-oriented architecture
 - Instructions variable number of bits long
 - Heavily microcoded
 - Severe performance, complexity, and usability problems
- Intel 8086 (1978, 8MHz, 29,000 transistors)
 - “Stopgap” 16-bit processor, architected in 10 weeks
 - Extended accumulator architecture
 - Assembly-compatible with 8080
 - 20-bit addressing through segmented addressing scheme
- IBM PC uses Intel 8088 for 8-bit bus
 (and Motorola 68000 was late)
 - Estimated sales of 250,000; 100,000,000s sold



Analyzing Microcoded Machines

1980s

- John Cocke and group at IBM
 - Working on a simple pipelined processor, 801 minicomputer (ECL server), and advanced compilers inside IBM
 - Ported experimental PL.8 compiler to IBM 370, only used simple register-register and load/store instructions similar to 801
 - Code ran faster than other existing compilers that used all 370 instructions!
 - Up to 6 MIPS whereas 2 MIPS considered good before
- Emer and Clark at DEC
 - Found 20% of VAX instructions responsible for 60% of microcode, but only account for 0.2% of execution time!
- Patterson 1979 sabbatical at DEC
 - VAX microcode bugs ⇒ field repair, but field-repairable chips don't make sense





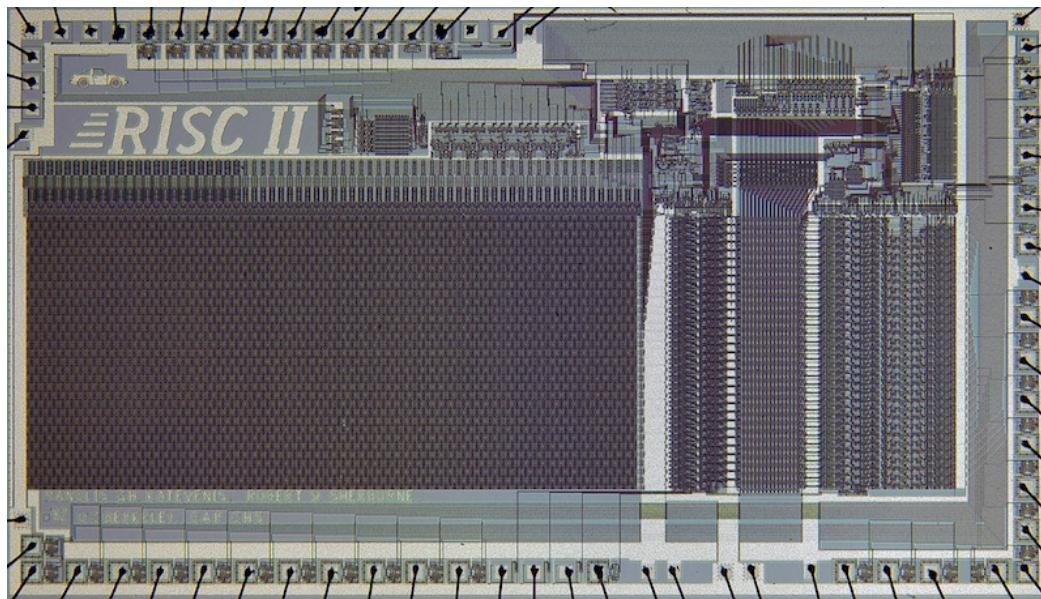
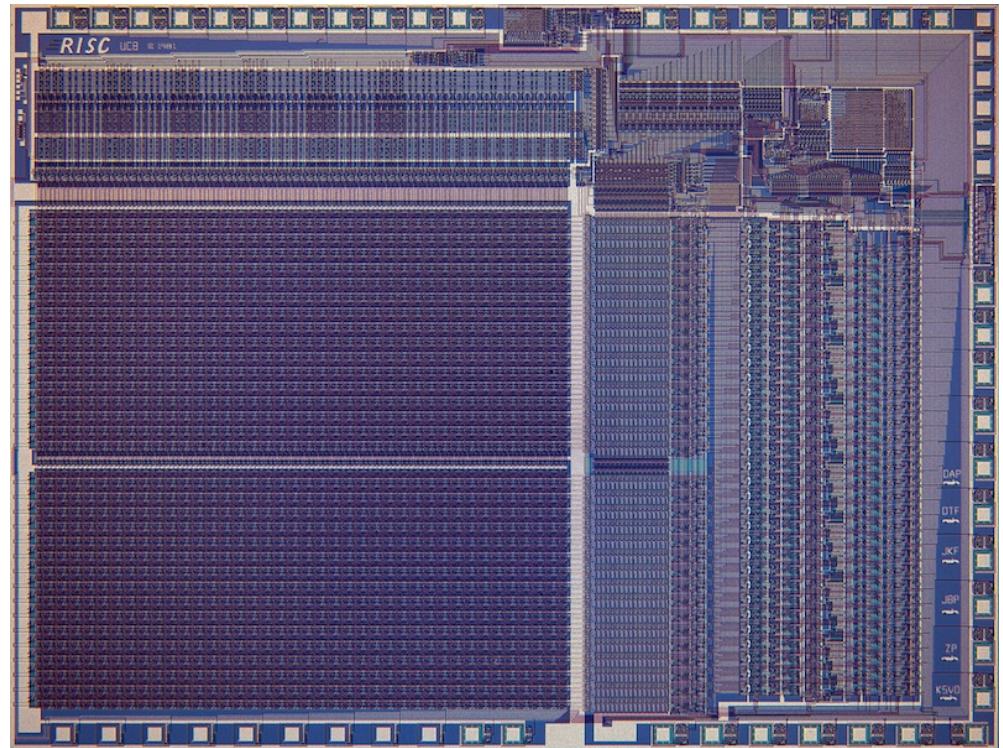
From CISC to RISC

- Use fast RAM to build fast instruction *cache* of user-visible instructions, not fixed hardware microroutines
 - Contents of fast instruction memory change to fit what application needs right now
- Simple ISA => hardwired pipelined implementation
 - Compiled code only used a few CISC instructions
 - Simpler encoding allowed pipelined implementations
- Further benefit with integration
 - In early '80s, could finally fit 32-bit datapath + small caches on a single chip
 - No chip crossings in common case allows faster operation



Berkeley RISC Chips

RISC-I (1982) Contains 44,420 transistors, fabbed in 5 μm NMOS, with a die area of 77 mm^2 , ran at 1 MHz.



RISC-II (1983) contains 40,760 transistors, was fabbed in 3 μm NMOS, ran at 3 MHz, and the size is 60 mm^2

Stanford built some too...

IEEE MILESTONE IN ELECTRICAL ENGINEERING AND COMPUTING

First RISC (Reduced Instruction-Set Computing) Microprocessor
1980-1982

UC Berkeley students designed and built the first VLSI reduced instruction-set computer in 1981. The simplified instructions of RISC-I reduced the hardware for instruction decode and control, which enabled a flat 32-bit address space, a large set of registers, and pipelined execution. A good match to C programs and the Unix operating system, RISC-I influenced instruction sets widely used today, including those for game consoles, smartphones and tablets.

February 2015

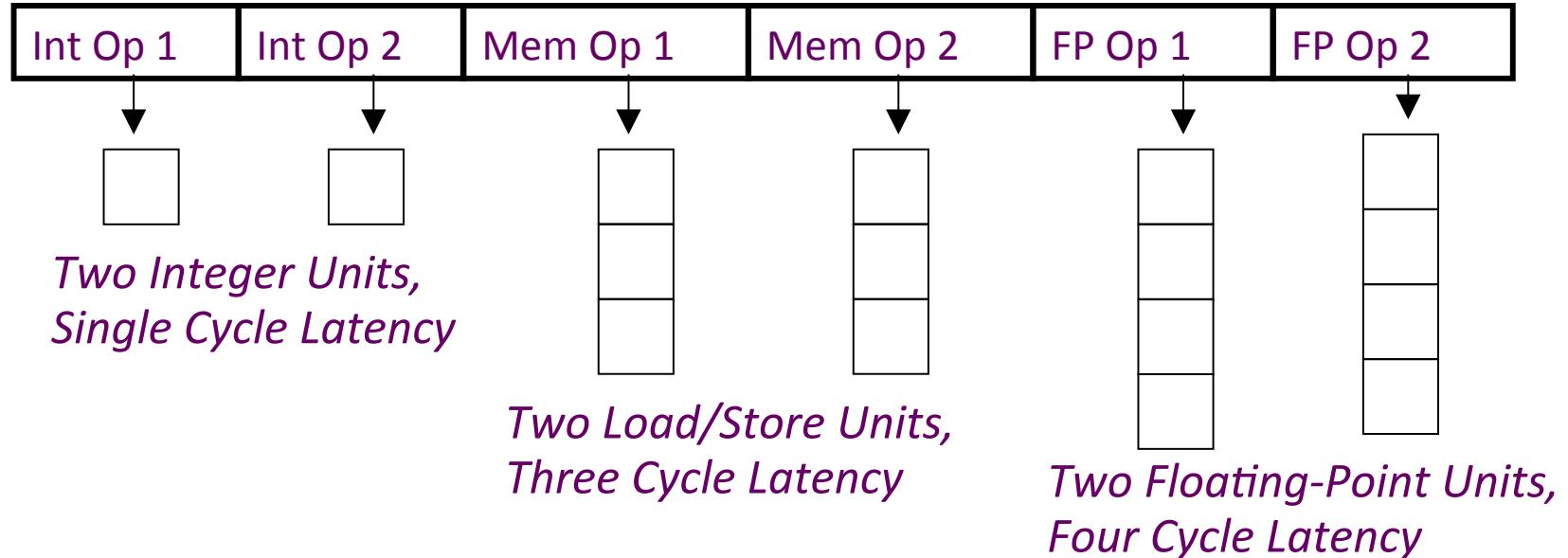




CISC vs. RISC Today

- PC Era
- Hardware translates x86 instructions into internal RISC instructions
- Then use any RISC technique inside MPU
- > 350M / year !
- x86 ISA eventually dominates servers as well as desktops
- PostPC Era: Client/Cloud
- IP in SoC vs. MPU
- Value die area, energy as much as performance
- > 16B / year in 2014!
- 98% RISC Processors:
 - 12.0B ARM (Advanced RISC Machine)
 - 2.0B Tensilica
 - 1.5B ARC (Argonaut RISC Core)
 - 0.8B MIPS

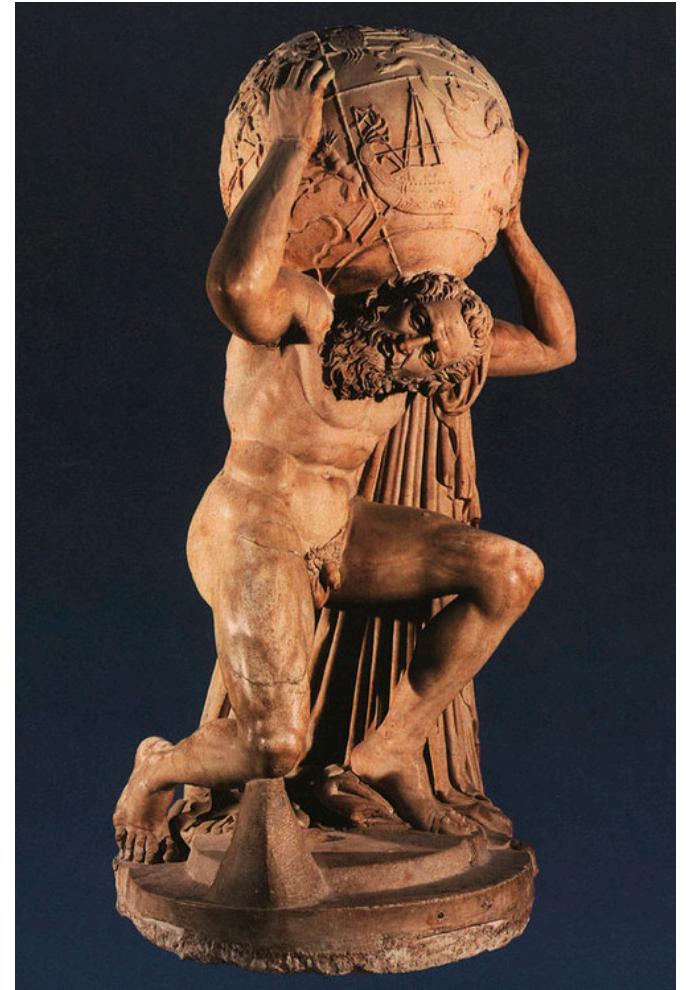
VLIW: Very Long Instruction Word



- Multiple operations packed into one instruction
- Each operation slot is for a fixed function
- Constant operation latencies are specified
- Architecture requires guarantee of:
 - Parallelism within an instruction => no cross-operation RAW check
 - No data use before data ready => no data interlocks

VLIW Compiler Responsibilities

- Schedule operations to maximize parallel execution
- Guarantees intra-instruction parallelism
- Schedule to avoid data hazards (no interlocks)
 - Typically separates operations with explicit NOPs



Loop Unrolling

```
for (i=0; i<N; i++)  
    B[i] = A[i] + C;
```

Unroll inner loop to perform 4 iterations at once

```
for (i=0; i<N; i+=4)  
{  
    B[i]      = A[i] + C;  
    B[i+1] = A[i+1] + C;  
    B[i+2] = A[i+2] + C;  
    B[i+3] = A[i+3] + C;  
}
```

Scheduling Loop Unrolled Code

Unroll 4 ways

```
loop: fld f1, 0(x1)
      fld f2, 8(x1)
      fld f3, 16(x1)
      fld f4, 24(x1)
      add x1, 32
      fadd f5, f0, f1
      fadd f6, f0, f2
      fadd f7, f0, f3
      fadd f8, f0, f4
      fsd f5, 0(x2)
      fsd f6, 8(x2)
      fsd f7, 16(x2)
      fsd f8, 24(x2)
      add x2, 32
      bne x1, x3, loop
```

Schedule →

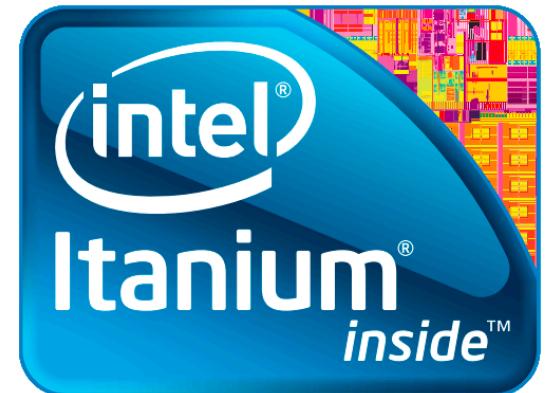
	Int1	Int 2	M1	M2	FP+	FPx
loop:			fld f1			
			fld f2			
			fld f3			
	add x1		fld f4		fadd f5	
					fadd f6	
					fadd f7	
					fadd f8	
			fsd f5			
			fsd f6			
			fsd f7			
	add x2	bne	fsd f8			

How many FLOPS/cycle?

$$4 \text{ fadds} / 11 \text{ cycles} = 0.36$$

Intel Itanium, EPIC IA-64

- EPIC is the style of architecture
 - “Explicitly Parallel Instruction Computing”
 - A binary object-code-compatible VLIW
 - Developed jointly with HP
- IA-64 was Intel’s chosen 64b ISA successor to 32b x86
 - IA-64 = Intel Architecture 64-bit
 - AMD wouldn’t be able to make, unlike x86
- Intel Merced was first Itanium implementation
 - 1st customer shipment expected 1997 (actually 2001)
 - McKinley, 2nd implementation, 180 nm, shipped in 2002
 - Poulson, most recent, 8 cores, 32 nm, shipped in 2012



VLIW Issues and an “EPIC Failure”

- Unpredictable branches
- Variable memory latency (unpredictable cache misses)
- Code size explosion
- Compiler complexity: *“The Itanium approach...was supposed to be so terrific—until it turned out that the wished-for compilers were basically impossible to write.”*
 - Donald Knuth, Stanford
- Columnist Ashlee Vance noted delays and under performance of Itanium *“turned the product into a joke in the chip industry”*



Itanium => “Itanic” (like infamous ship *Titanic*) **20**



2000s: How Should We Build Scalable Multiprocessors?

1. Shared Memory with "Non Uniform Memory Access" time (NUMA) using loads and stores
 - Distributed directory remembers sharing for coherency and consistency
 - DASH/FLASH projects at Stanford (1992-2000)
2. Message passing Cluster with separate address space per processor using RPC (or MPI)
 - Collection of independent computers connected by LAN switches to provide a common service
 - Network of Workstations project at Berkeley (1993-1998)



SGI Origin 2000 NUMA vs. Sun Enterprise 10000 SMP

- A pure NUMA
- Scales up to 2048 CPUs
- Scalable bandwidth is crucial to Origin
- Designed for scientific computation
- A pure UMA
- Up to 64 CPUs
- \$4.7M = 64 CPUs, 64 GB SDRAM memory, 868 18GB disk, 12X CD, 1yr service
- Designed for commercial processing



NUMA Advantages

- Ease of programming when communication patterns are complex or vary dynamically during execution
- Ability to develop apps using familiar SMP model
- Lower communication overhead, better use of BW for small items due to implicit communication
- HW-controlled caching to reduce remote communication by caching of all data



Cluster Drawbacks

- Cost of administering a cluster of N machines
 - ~ administering N independent machines
 - vs. cost of administering a shared address space N processors multiprocessor ~ administering 1 big machine
- Clusters usually connected using I/O bus, whereas multiprocessors usually connected on memory bus
- Cluster of N machines has N independent memories and N copies of OS and code, but a shared address multi-processor allows 1 program to use almost all memory



Cluster Advantages

- Error isolation: separate address space limits contamination of error
- Repair: Easier to replace a machine without bringing down the system
- Scale: easier to expand the system
- Cost: Large scale machine has low volume => fewer machines to spread development costs vs. leverage high volume off-the-shelf switches and computers
- Inktomi first then Amazon, AOL, Google, Hotmail, WebTV, Yahoo ... relied on clusters of PCs to provide services used by millions of people every day

Review: Networking

- Clusters +: fault isolation and repair, scaling, cost
- Clusters -: maintenance, network interface performance, memory efficiency
- Google as cluster example:
 - scaling (6000 PCs, 1 petabyte storage)
 - fault isolation (2 failures per day yet available)
 - repair (replace failures weekly/repair offline)
 - Maintenance: 8 people for 6000 PCs
- Cell phone as portable network device
 - # Handsets >> # PCs
 - Universal mobile interface?
- Is future services built on Google-like clusters delivered to gadgets like cell phone handset?

Outline

Part I - Past

50 years of Computer Architecture History:

- 1960s:
Computer Families /
Microprogramming
- 1970s: CISC
- 1980s: RISC
- 1990s: VLIW
- 2000s: NUMA vs.
Clusters

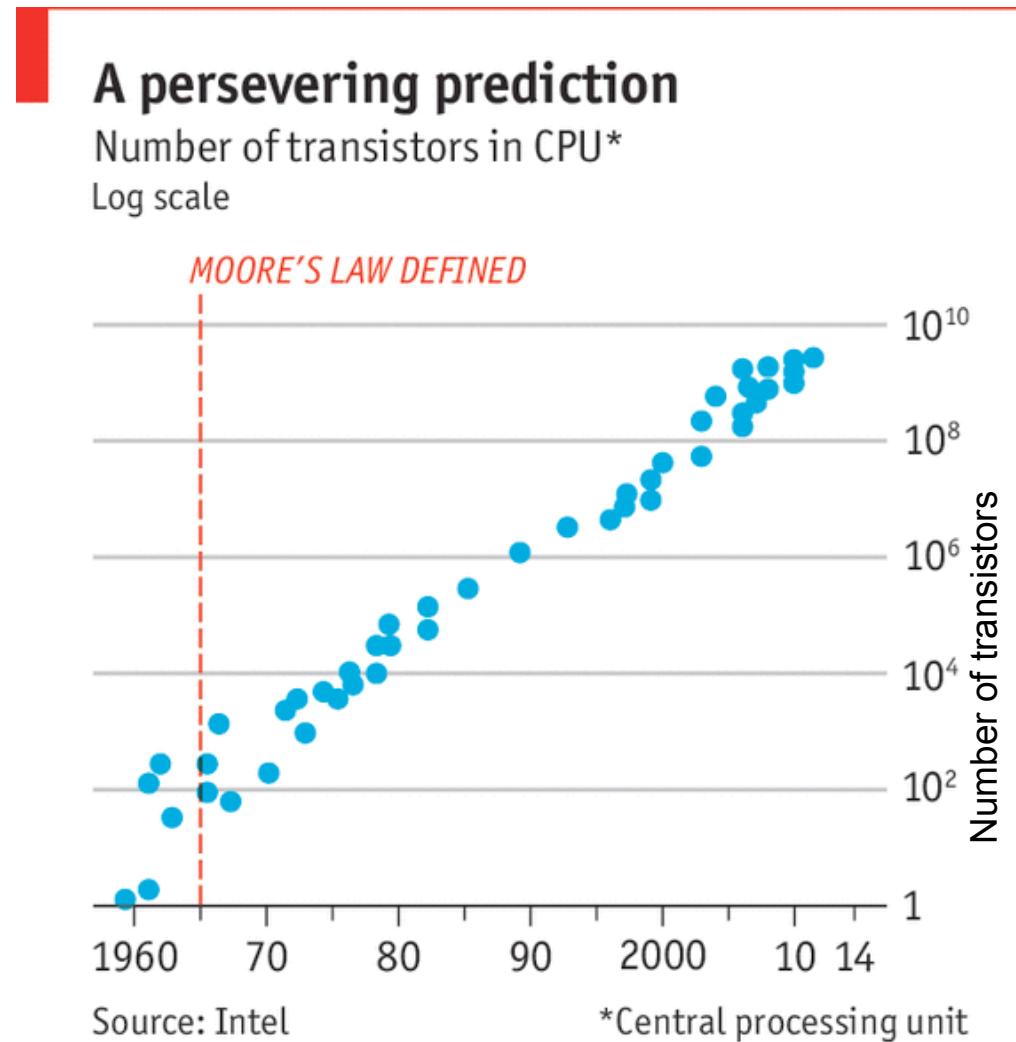
Part II – Future

HW Technology

- End of Moore's Law
- Flash vs. Disks
- Fast DRAM
- Crosspoint NVRAM
- Open ISA / RISC-V
 - Case for Open ISAs
 - Tour of RISC-V ISA
 - RISC-V Software Stack
 - RISC-V Chips

Moore's Law Slowing Down

- Stated 50 years ago by Gordon Moore
 - Number of transistors on microchip double every **1-2 years**
 - Today **2.5-3? years**





CPU Performance Improvement

- Number of cores: +18-20%
- Per core performance: +10%
- Aggregate improvement: +30-32%



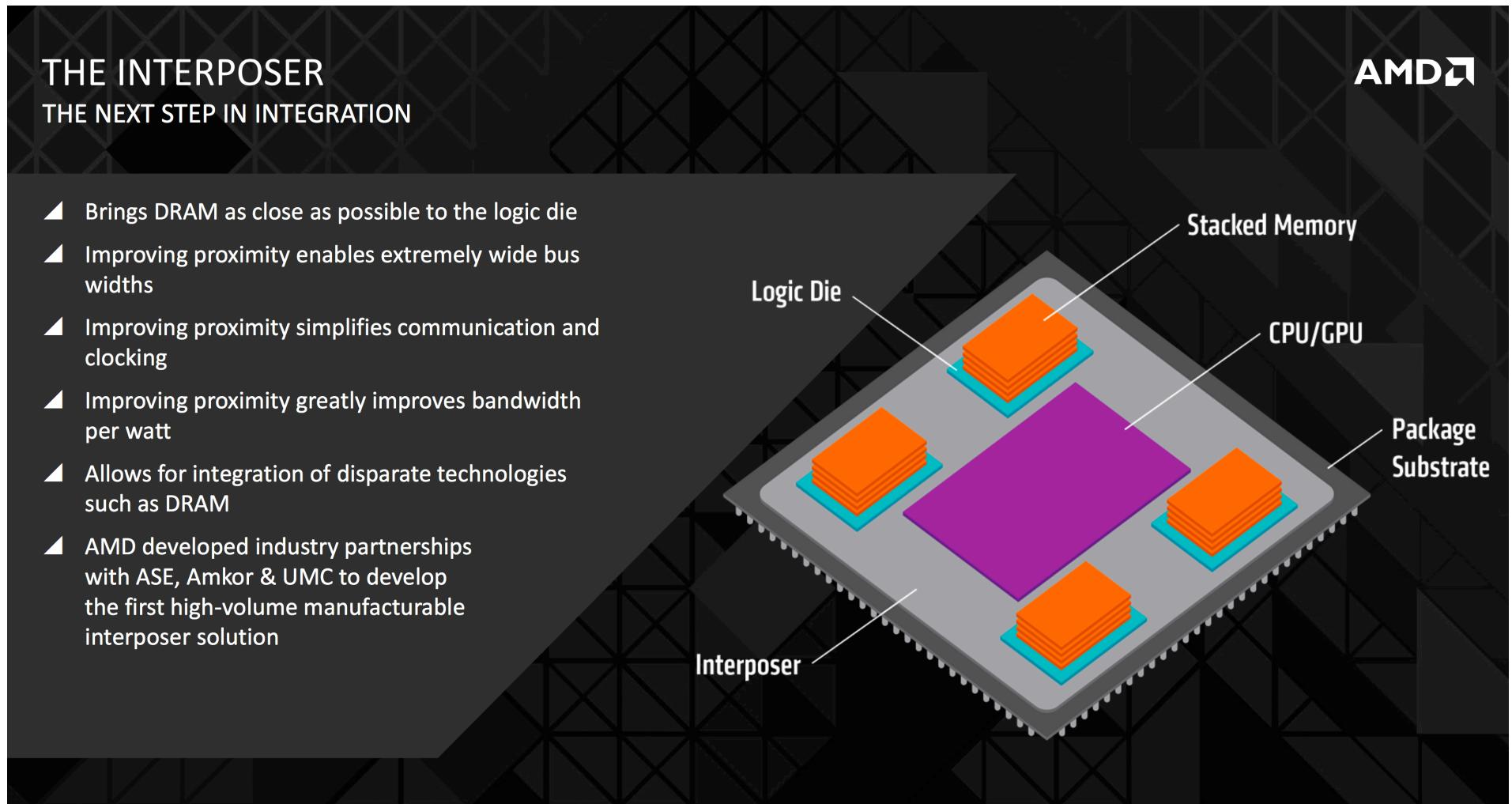
Memory Price/Byte Evolution

- 1990-2000: **-54%** per year
- 2000-2010: **-51%** per year
- 2010-2015: **-32%** per year
- (<http://www.jcmit.com/memoryprice.htm>)

High Bandwidth Memory

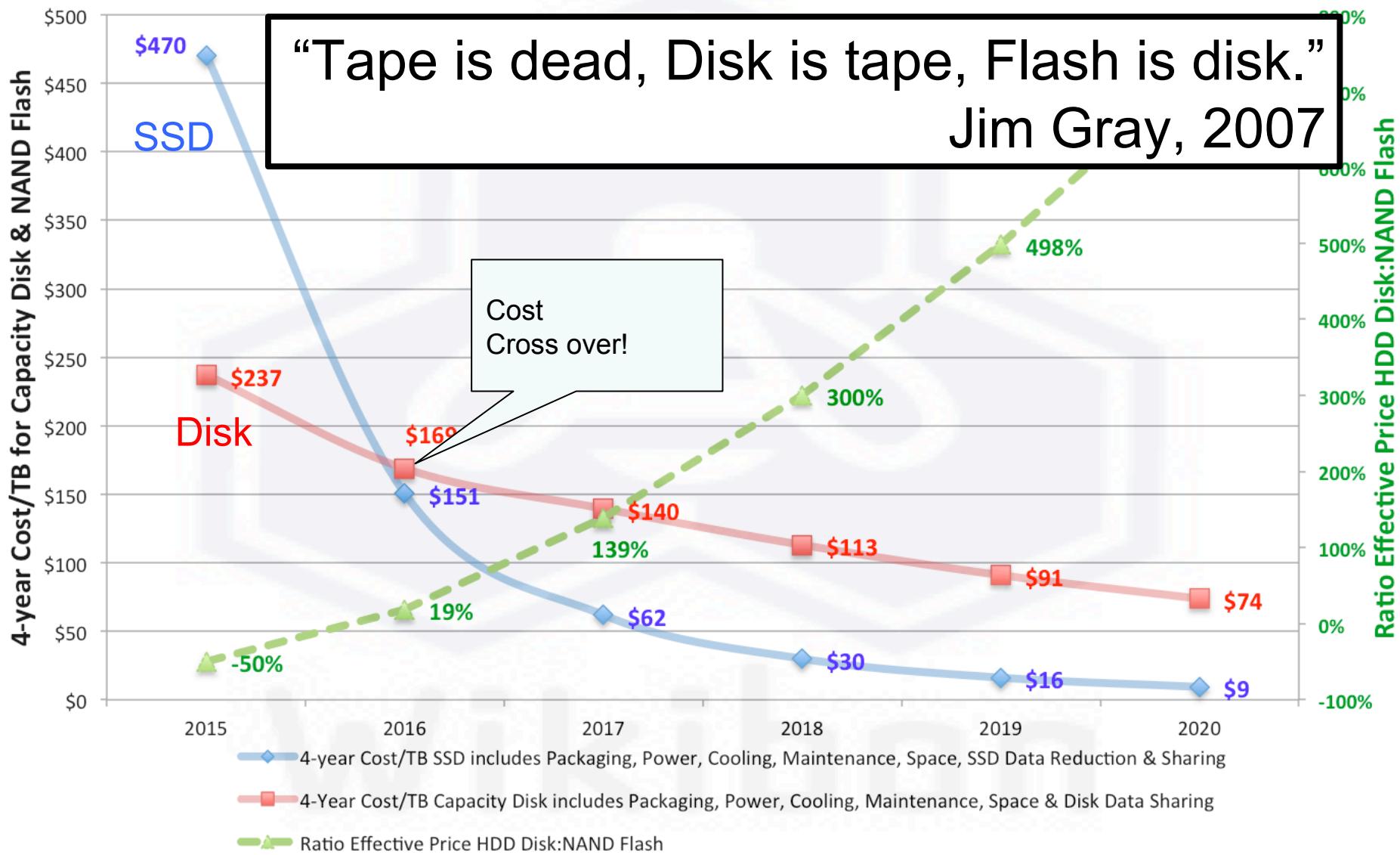
THE INTERPOSER
THE NEXT STEP IN INTEGRATION

AMD



- ▲ Brings DRAM as close as possible to the logic die
- ▲ Improving proximity enables extremely wide bus widths
- ▲ Improving proximity simplifies communication and clocking
- ▲ Improving proximity greatly improves bandwidth per watt
- ▲ Allows for integration of disparate technologies such as DRAM
- ▲ AMD developed industry partnerships with ASE, Amkor & UMC to develop the first high-volume manufacturable interposer solution

Projection 2015-2020 of Capacity Disk & Scale-out Capacity NAND Flash



Source: © Wikibon 2015. 4-Year Cost/TB Magnetic Disk & SSD, including Packaging, Power, Maintenance, Space, Data Reduction & Data Sharing



3D XPoint Technology

- Developed by Intel and Micron
 - Announced July 28, 2015!
- Exceptional characteristics:
 - Non-volatile memory
 - 1000x more resilient than SSDs
 - 8-10x density of DRAM
 - Performance in DRAM ballpark!
 - 2-3x slower reads, 4x-6x slower writes



Future Memory Hierarchy Deeper

- Storage hierarchy gets more and more complex:
 - L1 cache
 - L2 cache
 - L3 cache
 - Fast DRAM (on interposer with CPU)
 - 3D XPoint based storage
 - SSD
 - (HDD)
- Need to design software to take advantage of this hierarchy

Consensus on ISAs Today



- Not CISC: no new commercial CISC ISAs in 30+ years
- Not VLIW: Despite several attempts,
VLIW has failed in general-purpose computing arena
 - Complex VLIW architectures close to in-order superscalar in complexity, no real advantage on large complex apps
 - Although some VLIWs successful in embedded DSP market
(Simpler VLIWs, more constrained, friendlier code)
- RISC! Widespread agreement (still) that RISC principles are best for general purpose ISA



So...

If there is widespread agreement on ISA principles ...

Why isn't there a free, open, industry-standard ISA?





ISAs Should Be Free and Open

While ISAs may be proprietary for historical or business reasons, there is no good technical reason for the lack of free, open ISAs:

- It's not an error of omission
- Nor is it because the companies do most of the software development
- Neither do companies exclusively have the experience needed to design a competent ISA
- Nor are the most popular ISAs wonderful ISAs
- Neither can only companies verify ISA compatibility
- Nor does it protect you from patent lawsuits
- Finally, proprietary ISAs are not guaranteed to last, and many actually disappear



Why Open ISA Now?

1. Switch from microprocessors of PC Era to IP in SoC of PostPC Era
 - ⇒ Can offer designs (as ARM does) without offering chips (as Intel does)
2. Ending of Moore's Law
 - ⇒ Cost/performance/energy advance via architectural innovation vs. semiconductor process improvements
 - ⇒ Renaissance for domain specific coprocessor (e.g., image processor, DSP, GPU, ...)
 - ⇒ Want a minimal, open ISA to run standard software with domain specific coprocessors



RISC-V Origin Story

- In 2010, after many years and many projects using MIPS, SPARC, and x86 as basis of research, time to look at ISA for next set of projects
- x86 and ARM obvious choices, but complex ISAs and serious IP issues
- MIPS64 – not enough opcodes left if try to extend
- So we started “3-month project” in summer 2010 to develop our own clean-slate ISA
- Four years later, we released frozen base user spec
 - Also many tape outs and research publications
- Why are Outsiders complaining about changes to RISC-V in Berkeley classes???



Modest RISC-V Goal

Become an industry-standard ISA for
all computing devices





RISC-V Base Plus Standard Extensions

- Three base integer ISAs, one per address width
 - RV32I, RV64I, RV128I
 - Minimal: <50 hardware instructions needed
- Modular: Standard extensions
 - M: Integer multiply/divide
 - A: Atomic memory operations (AMOs + LR/SC)
 - F: Single-precision floating-point
 - D: Double-precision floating-point
 - Q: Quad-precision floating-point
 - C: Compressed instruction encoding (16b and 32b)
- Reserved opcode space for SoC unique instructions
- All the above in fairly standard RISC encoding



RV32I

Base Integer Instructions: RV32I, RV						
Category	Name	Fmt	RV32I Base			
Loads	Load Byte	I	LB	rd,rs1,imm		
	Load Halfword	I	LH	rd,rs1,imm		
	Load Word	I	LW	rd,rs1,imm		
	Load Byte Unsigned	I	LBU	rd,rs1,imm		
	Load Half Unsigned	I	LHU	rd,rs1,imm		
Stores	Store Byte	S	SB	rs1,rs2,imm		
	Store Halfword	S	SH	rs1,rs2,imm		
	Store Word	S	SW	rs1,rs2,imm		
Shifts	Shift Left	R	SLL	rd,rs1,rs2		
	Shift Left Immediate	I	SLLI	rd,rs1,shamt		
	Shift Right	R	SRL	rd,rs1,rs2		
	Shift Right Immediate	I	SRLI	rd,rs1,shamt		
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2		
	Shift Right Arith Imm	I	SRAI	rd,rs1,shamt		
Arithmetic	ADD	R	ADD	rd,rs1,rs2		
	ADD Immediate	I	ADDI	rd,rs1,imm		
	SUBtract	R	SUB	rd,rs1,rs2		
	Load Upper Imm	U	LUI	rd,imm		
	Add Upper Imm to PC	U	AUIPC	rd,imm		
Logical	XOR	R	XOR	rd,rs1,rs2		
	XOR Immediate	I	XORI	rd,rs1,imm		
	OR	R	OR	rd,rs1,rs2		
	OR Immediate	I	ORI	rd,rs1,imm		
	AND	R	AND	rd,rs1,rs2		
	AND Immediate	I	ANDI	rd,rs1,imm		
Compare	Set <	R	SLT	rd,rs1,rs2		
	Set < Immediate	I	SLTI	rd,rs1,imm		
	Set < Unsigned	R	SLTU	rd,rs1,rs2		
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm		
Branches	Branch =	SB	BEQ	rs1,rs2,imm		
	Branch ≠	SB	BNE	rs1,rs2,imm		
	Branch <	SB	BLT	rs1,rs2,imm		
	Branch ≥	SB	BGE	rs1,rs2,imm		
	Branch < Unsigned	SB	BLTU	rs1,rs2,imm		
	Branch ≥ Unsigned	SB	BGEU	rs1,rs2,imm		
Jump & Link	J&L	UJ	JAL	rd,imm		
	Jump & Link Register	UJ	JALR	rd,rs1,imm		
Synch	Synch thread	I	FENCE			
	Synch Instr & Data	I	FENCE.I			
System	System CALL	I	SCALL			
	System BREAK	I	SBREAK			
Counters	ReaD CYCLE	I	RDCYCLE	rd		
	ReaD CYCLE upper Half	I	RDCYCLES	rd		
	ReaD TIME	I	RDTIME	rd		
	ReaD TIME upper Half	I	RDTIMES	rd		
	ReaD INSTR RETired	I	RDINSTRRET	rd		
	ReaD INSTR upper Half	I	RDINSTRTH	rd		

32-bit Instruction Formats

	31	30	25 24	21	20	19	15 14	12 11	
R			funct7	rs2	rs1	funct3	rd	opcode	
I			imm[11:0]		rs1	funct3	rd	opcode	
S			imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	
SB			imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1] imm[11]	opcode
U			imm[31:12]				rd	opcode	
UJ			imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode	

+ 12
for
64I
/128I

14
Privileged

+ 8 for M
+ 11 for A

+ 30 for C

+ 34
for F, D, Q

+ 4 for
64M
/128M
+ 11 for
64A
/128A
+ 6 for
64F/
128F,
64D/
128D,
64Q/
128Q
42



RV32I / RV64I / RV128I + C, M, A, F, D,& Q RISC-V “Green Card”

Base Integer Instructions: RV32I, RV64I, and RV128I							RV Privileged Instructions							Optional Multiply-Divide Instruction Extension: RVM						
Category	Name	Fmt	RV32I Base			+RV{64,128}	Category	Name	RV mnemonic			Category	Name	Fmt	RV32M (Multiply-Divide)			+RV{64,128}		
Loads	Load Byte	I	LB	rd,rs1,imm			CSR Access	Atomic R/W	CSRRW	rd,csr,rs1		Multiply	MUL	rd,rs1,rs2	MUL{W D}	rd,rs1,rs2				
	Load Halfword	I	LH	rd,rs1,imm				Atomic Read & Set Bit	CSRRS	rd,csr,rs1			MULH	rd,rs1,rs2						
	Load Word	I	LW	rd,rs1,imm	L{D Q}	rd,rs1,imm		Atomic Read & Clear Bit	CSRRC	rd,csr,rs1			MULHSU	rd,rs1,rs2						
	Load Byte Unsigned	I	LBU	rd,rs1,imm				Atomic R/W Imm	CSRRWI	rd,csr,imm			MULHU	rd,rs1,rs2						
	Load Half Unsigned	I	LHU	rd,rs1,imm	L{W D}U	rd,rs1,imm		Atomic Read & Set Bit Imm	CSRRCI	rd,csr,imm			DIV	rd,rs1,rs2	DIV{W D}	rd,rs1,rs2				
Stores	Store Byte	S	SB	rs1,rs2,imm			Change Level	Env Call	ECAL			DIVU	rd,rs1,rs2							
	Store Halfword	S	SH	rs1,rs2,imm			Environment Breakpoint	EBREAK				REM	rd,rs1,rs2	REM{W D}	rd,rs1,rs2					
	Store Word	S	SW	rs1,rs2,imm	S{D O}	rs1,rs2,imm		Environment Return	ERET			REMU	rd,rs1,rs2	REMU{W D}	rd,rs1,rs2					
Shifts	Shift Left	R	SLL	rd,rs1,rs2	SLL{W D}	rd,rs1,rs2	Trap Redirect	to Supervisor	MRTS			Optional Atomic Instruction Extension: RVA			+RV{64,128}					
	Shift Left Immediate	I	SLLI	rd,rs1,shamt	SLLI{W D}	rd,rs1,shamt	Redirect Trap to Hypervisor	MRTTH	MRTS			RV32A (Atomic)			+RV{64,128}					
	Shift Right	R	SRL	rd,rs1,rs2	SRL{W D}	rd,rs1,rs2	Hypervisor Trap to Supervisor	HRTS				Load			LR.W					
	Shift Right Immediate	I	SRLI	rd,rs1,shamt	SRLI{W D}	rd,rs1,shamt	Wait for Interrupt	WFI				Store			SC.{D Q}					
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2	SRA{W D}	rd,rs1,rs2						Swap			AMOSWAP.W					
	Shift Right Arith Imm	I	SRAI	rd,rs1,shamt	SRAI{W D}	rd,rs1,shamt						Add			AMOADD.D Q					
Arithmetic	ADD	R	ADD	rd,rs1,rs2	ADD{W D}	rd,rs1,rs2	MMU	Supervisor FENCE	SFENCE.VM	rs1		Logical			AMOXOR.W					
	ADD Immediate	I	ADDI	rd,rs1,imm	ADDI{W D}	rd,rs1,imm						AND			AMOAND.D Q					
	SUBtract	R	SUB	rd,rs1,rs2	SUB{W D}	rd,rs1,rs2						OR			AMOOR.D Q					
	Load Upper Imm	U	LUI	rd,imm								Min/Max			AMOMIN.W					
	Add Upper Imm to PC	U	AUIPC	rd,imm								MAXimum			AMOMAX.W					
Logical	XOR	R	XOR	rd,rs1,rs2	C.LW	rd',rs1',imm						MINimum Unsigned			AMOMINU.D Q					
	XOR Immediate	I	XORI	rd,rs1,imm	Load Word SP	CI						MAXimum Unsigned			AMOMAXU.D Q					
	OR	R	OR	rd,rs1,rs2	Load Double SP	CL						Three Optional Floating-Point Instruction Extensions: RVF, RVD, & RVO			+RV{64,128}					
	OR Immediate	I	ORI	rd,rs1,imm	Load Quad SP	CI						Move			FMV.{H S}.X					
	AND	R	AND	rd,rs1,rs2								Move to Integer			FMV.X.{H S}					
	AND Immediate	I	ANDI	rd,rs1,imm								Convert			FCVT.{H S D Q}.W					
Compare	Set <	R	SLT	rd,rs1,rs2	CSW	rs1',rs2',imm						Convert from Int			FCVT.{H S D Q}.{L T}					
	Set < Immediate	I	SLTI	rd,rs1,imm	CSWSW	rs2,imm						Convert from Int Unsigned			FCVT.{H S D Q}.{WU}					
	Set < Unsigned	R	SLTU	rd,rs1,rs2	CSD	rs1',rs2',imm						Convert to Int			FCVT.W.{H S D Q}					
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm	CSSDSP	rs2,imm						Convert to Int Unsigned			FCVT.WU.{H S D Q}					
Branches	Branch =	SB	BEQ	rs1,rs2,imm	CSQ	rs1',rs2',imm						Load			FL{W,D,Q}					
	Branch ≠	SB	BNE	rs1,rs2,imm	CSSQSP	rs2,imm						Store			S{W,D,Q}					
	Branch <	SB	BLT	rs1,rs2,imm								Arithmetic			FADD.{S D Q}					
	Branch ≥	SB	BGE	rs1,rs2,imm								SUBtract			FSUB.{S D Q}					
	Branch < Unsigned	SB	BLTU	rs1,rs2,imm								MULtiply			FMUL.{S D Q}					
	Branch ≥ Unsigned	SB	BGEU	rs1,rs2,imm								DIVide			FDIV.{S D Q}					
Jump & Link	J&L	UJ	JAL	rd,imm	CR	ADD	rd,rd,rs1					SQuare Root			FSQRT.{S D Q}					
	Jump & Link Register	UJ	JALR	rd,rs1,imm	CR	ADDW	rd,rd,imm					Mul-Add			FMADD.{S D Q}					
Synch	Synch thread	I	FENCE		CI	ADDI	rd,imm					Mul-SUBtract			FMSUB.{S D Q}					
	Synch Instr & Data	I	FENCE.I		CI	ADDIW	rd,rd,imm					Negative Mul-Subtract			FNNSUB.{S D Q}					
System	System CALL	I	SCALL		CI	ADDI16SP	x0,imm					Negative Multi-Add			FNNAADD.{S D Q}					
	System BREAK	I	SBREAK		CI	ADDI4SPN	rd',imm					Sign Inject			FSGNJ.{S D Q}					
Counters	Read CYCLE	I	RDCYCLE	rd	CI	funct4	rd,rs1					Negative SIGN source			FSGNJN.{S D Q}					
	ReaD CYCLE upper Half	I	RDCYCLEH	rd	CI	funct3	rd,imm					Xor SIGN source			FSGNJK.{S D Q}					
	Read TIME	I	RDTIME	rd	CI	funct3	rd,imm					Min/Max			FMIN.{S D Q}					
	ReaD TIME upper Half	I	RDTIMEH	rd	CI	funct3	rd,imm					MAXimum			FMAX.{S D Q}					
	ReaD INSTR RETired	I	RDINSTRET	rd	CI	funct3	rd,imm					Compare			FEQ.{S D Q}					
	ReaD INSTR upper Half	I	RDINSTRETH	rd	CI	funct3	rd,imm					Compare Float <			FLT.{S D Q}					
32-bit Instruction Formats																				
R		31	30	25 24	21 20	19	15 14	12 11	8	7	6	0	CR	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	16-bit (RVC) Instruction Formats					
I				funct7	rs2	rs1	funct3	rd	opcode				CR	funct4	rd,rs1	rs2	op			
S				imm[11:0]	rs1	rs1	funct3	rd	opcode				CI	funct3	imm	rd,rs1	imm	op		
B				imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode				CSS	funct3	imm	rs2	op			
U				imm[12] imm[10:5]	rs2	rs1	funct3	imm[4:1] imm[11]	opcode				CTW	funct3	imm	rd'	op			
				imm[31:12]	rd	opcode							CL	funct3	imm	rs1'	imm	rd'	op	
				imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode				CS	funct3	imm	rs1'	imm	rs2'	op	
				imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode				CB	funct3	offset	rs1'	offset	op		
				imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode				CJ	funct3	jump target			op		

Configuration	Read Status	R	FRCSR	rd
	Read Rounding Mode	R	FRRM	rd
	Read Flags	R	FRFLAGS	rd
	Swap Status Reg	R	FCSR	rd,rs1
	Swap Rounding Mode	R	FSRM	rd,rs1
	Swap Flags	R	FSFLAGS	rd,rs1
	Swap Rounding Mode Imm	I	FSRMI	rd,imm
	Swap Flags Imm	I	FSFLAGSI	rd,imm



- **Documentation**

- User-Level ISA Spec v2.0
(Released 5/6/14)
- Privileged ISA Spec v1.7
(Released 5/9/15)
- Compressed Instr. v1.7
(Released 5/29/15)

- **Software Tools**

- GCC/glibc/GDB
- LLVM/Clang
- Linux
- Yocto
- Verification Suite

- **Hardware Tools**

- Zynq FPGA Infrastructure
- Chisel
- Interfaces to ARM buses
- Debugger interface (underway)

- **Hardware Implementations**

- Rocket Chip Generator
 - RV64G single-issue in-order pipe
- Zscale Chip Generator
- Zscale core also in Verilog
- Sodor Processor Collection

- **Software Implementations**

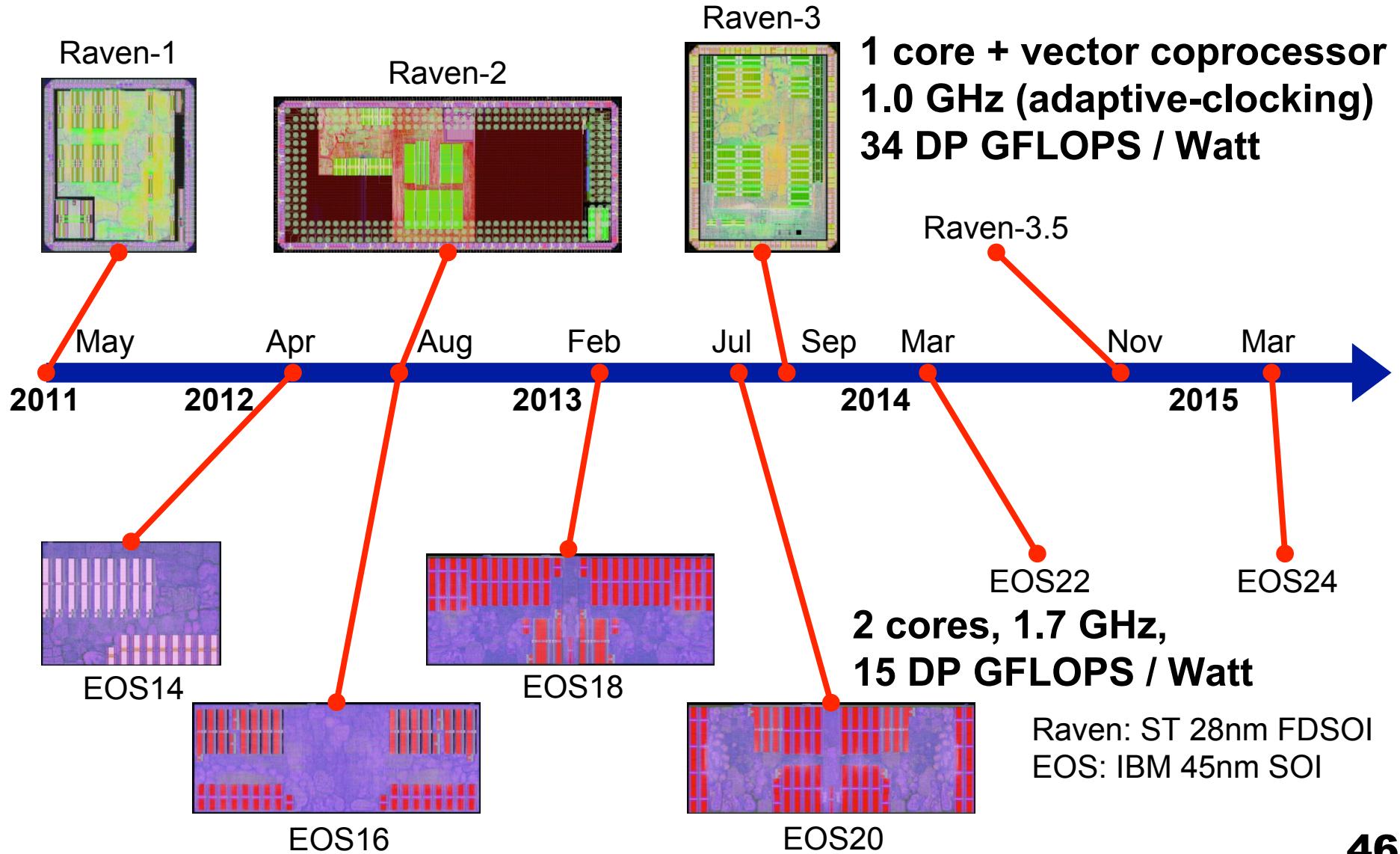
- ANGEL, JavaScript ISA Sim.
- Spike, In-house ISA Sim.
- QEMU ISA Sim.



RISC-V as Customizable Computer using FPGAs

- \$250 Zed FPGA board ⇒ working computer with full SW stack to customize as desired in ≈1 hour @ 50 – 100 MHz
- ≈1 minute on real hardware processor ⇒ ≈1 hour of FPGA vs ≈1 month on SW simulator
- 32 node FPGA cluster for ≈\$10,000

Four 28nm & Six 45nm RISC-V Chips taped out so far





Cost for 100 2x2mm 28 nm dies?

designlines SoC

Blog

Agile Design for Hardware, Part II

David Patterson and Borivoje Nikolić,
UC Berkeley

7/30/2015 07:00 AM EDT

12 comments post a comment



In the second of a three-part series, two Berkeley professors suggest its time to apply Agile design techniques to hardware.

We asked readers of [Part I](#) to guess the cost of a prototype run of 28 nm chips, as Agile development relies on a sequence of interim prototypes versus the One Big Tapeout of the traditional Waterfall process. Here are the results:

Prototype Category	Reader Average	Reader St. Dev.	Actual
Smallest die	2.3 x 2.3 mm	1.1 x 1.1 mm	1.57 x 1.57 mm
Fewest dies	190	280	80 to 100
Avg. cost / untested die	\$690	\$470	\$300 to \$375
Total cost	\$170,000	\$250,000	\$30,000

\$30,000!
Any project can
afford to build
hardware!

See “Is Agile
Development Feasible
for Hardware? Part II,”
by David Patterson and
Borivoje Nikolić, *EE
Times*, 8/1/2015



RISC-V Beyond Berkeley

- Adopted as “standard ISA” for **India**
 - IIT-Madras building 6 different open-source cores, from microcontrollers to servers (\$80M)
- **LowRISC** project based in Cambridge, UK producing open-source RISC-V based SoCs
 - Led by a founder of Raspberry Pi, privately funded
 - Adding capability-based security
 - Make and distribute ≈200,000 LowRISC SoCs
- **U. Maryland** research: Privacy preserving processor*

*Liu, Chang, Austin Harris, Martin Maas, Michael Hicks, Mohit Tiwari, and Elaine Shi. "GhostRider: A hardware-software system for memory trace oblivious computation." In *Proc. Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2015. Best paper award.



RISC-V Big Ideas: An ISA for SoCs

- Base of <50 RISC instrs run can full SW stack
 - Just need to get simple ISA working
- Optional standard extensions to include or omit
 - Save area/energy by using only what needed
- Reserved opcodes to tailor SoC to apps
 - Secret sauce per SoC yet run SW stack
- Free ISA: \$0, 0 paperwork, anyone can use
 - vs. if lucky, 6+ months negotiation + royalty
- Foundation will evolve RISC-V slowly for technical reasons determined by votes
 - vs. fast for business & technical reasons



Learning More about RISC-V

- Sign up for mailing lists/twitter at riscv.org to get announcements
- 1st RISC-V workshop was January 14-15 in Monterey
 - Slides & videos: riscv.org/workshop-jan2015.html
 - Sold out: 144 (33 companies & 14 universities)
- 2nd RISC-V workshop was June 29-30 at UC Berkeley
 - Slides & videos: riscv.org/workshop-jun2015.html
 - Sold out: 120 (30 companies & 20 universities)
- 3rd RISC-V workshop Jan 5-6 at Oracle Redwood City
 - Free to academics & RISC-V sponsors; \$149 others
 - Will likely sell out too, so sign up soon
 - Sign up www.regonline.com/riscvworkshop3

Outline

Part I - Past

50 years of Computer Architecture History:

- 1960s:
Computer Families /
Microprogramming
- 1970s: CISC
- 1980s: RISC
- 1990s: VLIW
- 2000s: NUMA vs.
Clusters

Part II – Future

HW Technology

- End of Moore's Law
- Flash vs. Disks
- Fast DRAM
- Crosspoint NVRAM
- Open ISA & RISC-V
 - Case for Open ISAs
 - Tour of RISC-V ISA
 - RISC-V Software Stack
 - RISC-V Chips

Questions?



BACKUP SLIDES



RISC-V ISA vs. ARMv8 ISA

Category	RISC-V	ARMv8	ARM/RISC
Year announced	2011	2011	--
Address sizes	32 / 64 / 128	32 / 64	--
Instruction formats	6 / 12 [†]	53	4X-8X
Data addressing modes	1	8	8X
Instructions	177 [†]	1,070	6X
Min number instructions to run Linux, gcc, LLVM	57	359	6X
Backend gcc compiler size	10K LOC	47K LOC	5X
Backend LLVM compiler size	10K LOC	22K LOC	2X
ISA manual size	181 pages	5,428 pages	30X

MIPS manual 700 pages
80x86 manual 3,600 pages

[†]With optional Compressed RISC-V ISA extension



And it's still growing! ARM v8.1

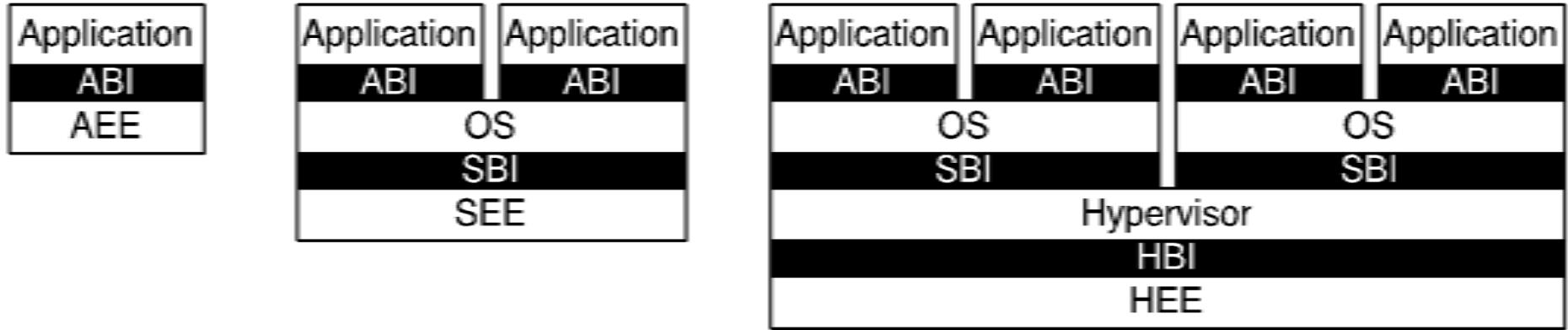
- “The ARM architecture, in line with other processor architectures, is evolving with time. ARMv8.1 is the first set of changes ...”*
 - Add a set of atomic read-write instructions
 - Add a set of load & store instruction limited to configurable address regions
 - More SIMD and scalar Multiply-Add instructions
 - “Signed Saturating Rounding Doubling Multiply Accumulate/Subtract, Returning High Half”
 - Add a new protection mode
 - Add a dirty bit for virtual address translation
 - Expand Virtual Machine ID register
- ...



*[“The ARMv8-A architecture and its ongoing development,” by David Bash, 12/2/2014](#)



RISC-V Privileged Architecture



- Application communicates with Application Execution Environment (AEE) via Application Binary Interface (ABI)
 - ABI: user ISA + calls to AEE
- OS communicates via Supervisor Execution Environment (SEE) via System Binary Interface (SBI)
 - SBI: user ISA + privileged ISA + calls to SEE
- Hypervisor communicates via Hypervisor Binary Interface (HBI) to Hypervisor Execution Environment (HEE)
- All levels of ISA designed to support virtualization



RISC-V Foundation

- Mission statement

“to standardize, protect, and promote the free and open RISC-V instruction set architecture and its hardware and software ecosystem for use in all computing devices.”

- Established 7/31/2015 as a 501(c)(6) foundation
- Rick O'Connor is Executive Director
- Currently recruiting “founding” member companies
 - 7 signed up so far; to be revealed at workshop



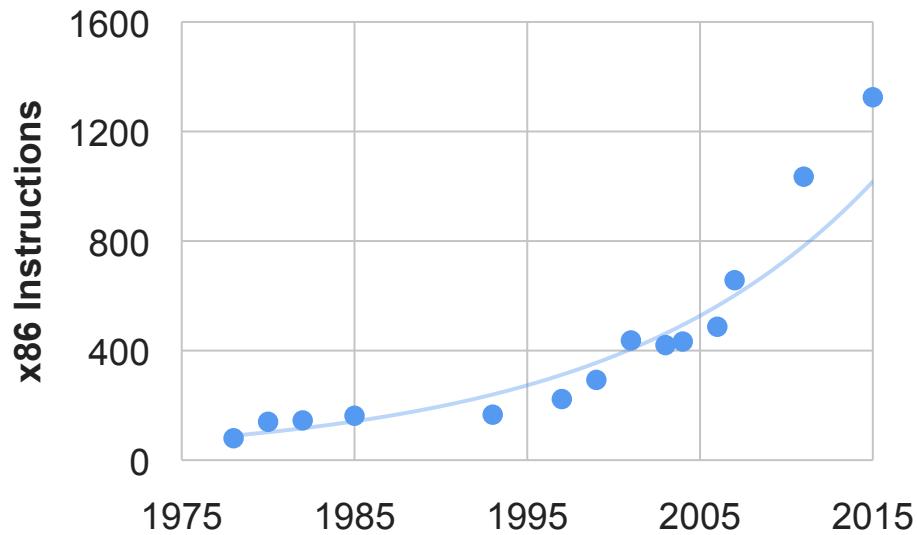
SSDs vs. HDDs

- SSDs will soon become cheaper than HDDs
- Transition from HDDs to SSDs will accelerate
 - Already most instances in Amazon Web Service have SSDs
- Going forward we can assume SSD-only clusters

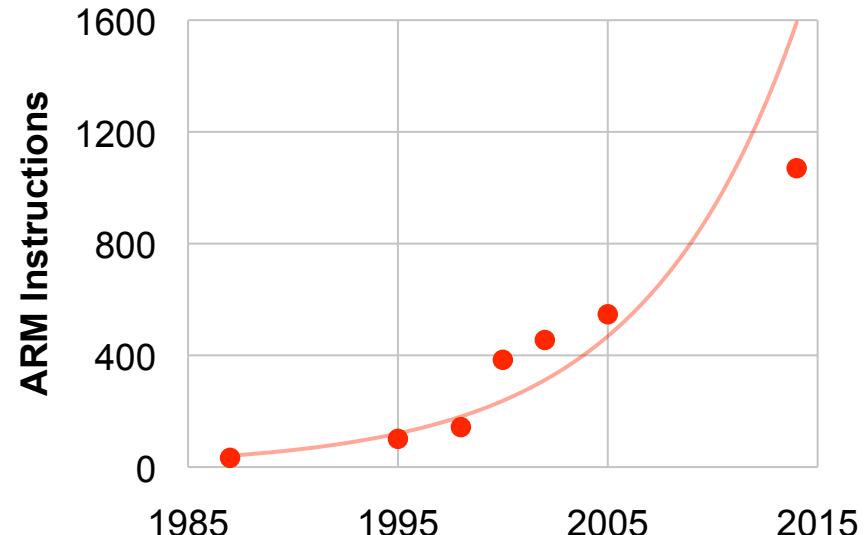
“Tape is dead, Disk is tape, Flash is disk.”

Jim Gray, 2007

Evolution of Proprietary ISAs by company for business & technical reasons



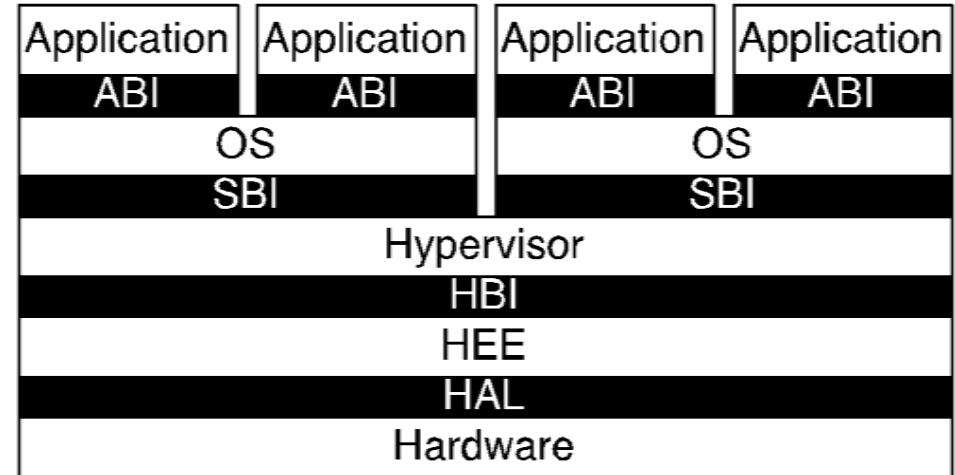
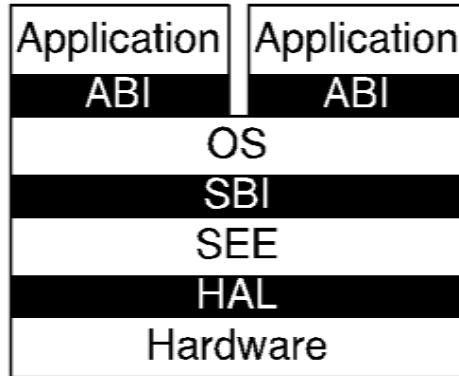
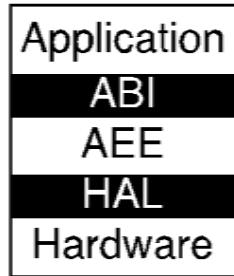
2 new x86 instructions
per month for 38 years



2 new ARM instructions
per month for 28 years



RISC-V Hardware Abstraction Layer



- HW requires more features beyond system ISA to support execution environments
- Separate features for HW platform from EE in HAL
 - Execution environments communicate with HW platforms via Hardware Abstraction Layer (HAL)
 - Details of execution environment and hardware platforms isolated from OS/Hypervisor ports

Four Supervisor Architectures

- Mbare
 - Bare metal, no translation or protection
- Mbb
 - Base and bounds protection
- Sv32
 - Demand-paged 32-bit VA space
- Sv39
 - Demand-paged 39-bit VA space
- Sv48
 - Demand-paged 48-bit VA space
- Page sizes: 4 KB, 2 MB, 1 GB
- Designed to support current popular operating systems
- Draft spec released May 7, 2015 for feedback





“Iron Law” of Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Clock cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Clock cycle}}$$

- Instructions per program depends on source code, compiler technology, and ISA
- Clock cycles per instructions (CPI) depends on ISA and underlying microarchitecture
- Time per clock cycle depends upon the microarchitecture and base technology
- RISC executes more instructions per program, but many fewer clock cycles per instruction (CPI) \Rightarrow RISC faster than CISC



RISC-V ISA and Patents?

- Patents last 20 years, ISAs since 1950s
⇒ patent ISA quirks
- MIPS sued Lexra ISA clone for load/store word left/right (unaligned data)
 - US patent 4,814,976 (expired 2006)
- ≈35 RISC ISAs ≤1995
- 100 expired RISC patents
 - ≈25 expire in 2016 ...
- 100% coverage RISC-V?
 - Genealogy poster?

Year	Research / Commercial RISC ISA
1980	IBM 801
1981	Berkeley RISC-I, RISC-II
1982	Stanford MIPS
1983	Pyramid Technology 90X
1984	Berkeley SOAR ("RISC-III")
1985	ARMv1, MIPS I, Alliant FX(vector), Convex C1(vector)
1986	Sun SPARC v7, HP PA-RISC, IBM RT-PC
1987	Berkeley SPUR (SMP) ("RISC-IV")
1988	AMD 29000, Intel i960, Motorola 88000
1989	Intel i860 (SIMD), National CompactRISC
1990	DLX, IBM POWER, Sun SPARC v8, MIPS II
1991	MIPS III (64b address), Hitachi SH-1
1992	IBM PowerPC, ARMv6, DEC Alpha (64b), SH-2
1993	IBM POWER2, Sun SPARC v9 (64b), SH-3
1994	ARM Thumb (16b instr), HP PA-RISC (SIMD)
1995	MIPS16e (16b instr)



RISC-V

Instruction Set Lineage

2015	1981	1984	1984	1987	1988	1990	1990	1992	1992	1992	1994
RISC V	RISC I RISC II	SOAR	Intel i960	ARMv2	SPUR	DLX	SPARCv8	DEC Alpha	MIPS III	IBM PowerPC	MIPS IV
LUI	LDHI					LHI	STHI		LUI		LUI
AUIPC				ADD ²							
JAL	CALL	BAL	BL	JUMP/CALL	JAL	JMPL		JAL	BL	JAL	
JALR	CALL	BAL	BL	JUMP/REGISTER	JALR	JMPL		JALR	BLR	JALR	
BEQ	JMPR	SKIP+CALL	BE	BEQ	CMP_BRANCH_LIKELY	BEQ	BICC	BEQ	BEQ	BEQ	BEQ
BNE	JMPR	SKIP+CALL	BNE	BNE	CMP_BRANCH_LIKELY	BNE	BICC	BNE	BNE	BNE	BNE
BLT	JMPR	SKIP+CALL	BL	BLT	CMP_BRANCH_LIKELY		BICC	BLT		BLT	
BGE	JMPR	SKIP+CALL	BGE	BGE	CMP_BRANCH_LIKELY	I	BICC	BGE		BCE	
BLTU	JMPR	SKIP+CALL			CMP_BRANCH_LIKELY	I			I	BLT	
BGEU	JMPR	SKIP+CALL			CMP_BRANCH_LIKELY	I			I	BGE	
LB	LDBS		LDIB	LDRB		LB	LDSB		LB	LBZ	LB
LH	LDS	LOADC	LDIS			LH	LDSH	LDL	LH	LHZ	LH
LW	LDL	LOAD	LD	LDRB	LOAD_32	LW	LD	LDQ	LW	LWZ	LW
LBU	LDBU		LDOB			LBU	LDUB		LBU	LBU	
LHU	LDSU		LDOS			LHU	LDUH		LHU	LHA	LHU
SB	STB		STIB	STRB		SB	STB		SB	STB	SB
SH	STS		STIS			SH	STH	STL	SH	STH	SH
SW	STL	STORE	ST	STR	STORE_32	SW	ST	STQ	SW	STW	SW
ADDI	ADD ¹	ADD		ADD	ADD	ADDI	ADD	ADD	ADDI	ADDI	ADDI
SLTI						SLTI			SLTI	SLTI	SLTI
SLTIU									SLTIU		SLTIU
XORI	XOR	XOR		EOR	XOR	XORI	XOR	XORI	XORI	XORI	XORI
ORI	OR	OR		OR	OR	OR	OR	BIS	ORI	ORI	ORI
ANDI	AND	AND		AND	AND	ANDI	AND	AND	ANDI	ANDI	ANDI
SLLI	SLL	SLA		LSL	SLL	SLLI	SLL		SLW		
SRLI	SRL	SRL		LSR	SRL	SRLI	SRL		SRW		
SRAI	SRA	SRA		ASR	SRA	SRAI	SRA		SRAWI		
ADD	ADD	ADD	ADDI	ADD	ADD	ADD	ADD	ADD	ADDI	ADDI	ADD
SUB	SUB/SUBR	SUB	SUBI	SUB	SUBTRACT	SUB	SUB	SUB	SUB	SUB	SUB
SLL	SLL	SLA	SHLI	LSL	SLL	SLL	SLL	SLL	SLW	SLL	SLL
SLT						SLT			SLT		SLT
SLTU									SLTU		SLTU
XOR	XOR	XOR	XOR	EOR	XOR	XOR	XOR	XOR	XORI	XOR	XOR
SRL	SRL	SRL	SHRO	LSR	SRL	SRL	SRL	SRL	SRW	SRL	SRL
SRA	SRA	SRA	SHRI	ASR	SRA	SRA	SRA	SRA	SRAW	SRA	SRA
OR	OR	OR	OR	OR	OR	OR	OR	BIS	ORI	ORI	ORI
AND	AND	AND	AND	AND	AND	AND	AND	AND	ANDI	AND	AND
FENCE								MB	SYNC	SYNC	SYNC
FENCE.I								CALL_PAL_IMB		ISYNC	
SCALL		TRAP	CALLS		CALL_KERNEL	TRAP	TRAP		SYSCALL	SC	SYSCALL
SBREAK			RET		RETURN_KERNEL	RFE	RETT			RFI	
RDCYCLE							RDASR	RPCC			
RDCYCLES											
RDTIME							RDASR				
RDINSTRET							RDASR				
RDINSTRETH											
MUL			MULI	MUL		MULT	SMUL	MUL	MULT ³	MULLW	MULT ³
MULH							SMUL		MULT	MULHW	MULT
MULHSU											
MULHU							UMUL	UMULH	MULTU	MULHWU	MULTU
DIV		DIVI			DIV	SDIV	SDIV		DIV	DIVV	DIV
DIVU		DIVO			DIVU	UDIV	UDIV		DIVU	DIVWU	DIVU
REMU		REMO									
LR.W							LDSTUB	LDI_I	LL	LWARX	LL
SC.W							LDSTUB	STL_C	SC	STWCX	SC
AMOSWAP.W		ATADD					SWAP				
AMOADD.W											
AMOXOR.W											
AMOAND.W											
AMOOR.W											
AMOMIN.W											
AMOMAX.W											
AMOMINU.W											
AMOMAXU.W											
FLW			LDF	LOAD_SINGLE	LF	LDF	LDS	LWC1	LFS	LWC1	
FSW			STF	STORE_SINGLE	SF	STF	STS	SWC1	STFS	SWC1	
FMADD.S									FMADDS	MADD.S	
FMSUB.S									FMSUBS	MSUB.S	
FNMSUB.S									FNMSUBS	NMSUB.S	
FNMADD.S									FNMADDS	NMADD.S	
FADD.S		ADDR	ADF	FADD	ADDF	FADDs	ADDs	ADD.S	FADDs	ADD.S	
FSUB.S			SUF	FSUB	SUBF	FSUBs	SUBS	SUB.S	FSUBs	SUB.S	
FMUL.S		MULR	MUF	FMUL	MULTF	FMULs	MUL.S	FMLS	FMULS	MUL.S	
FDIV.S		DIVR	DVF	FDIV	DIVF	FDIVs	DIVS	DIVLS	FDIVS	DIVLS	
FSQRT.S		SQRTR	SQT			FSQRTs		SQRT.S		SQRT.S	
FGSNJ.S		CPYSR ⁶						CPYS			
FGSNJS.S		CPYRSR ⁶		FNEGATE				CPYSN			
FGSNXJS.S											

Parallelism and operating systems

M. Frans Kaashoek

MIT CSAIL

With input from Eddie Kohler, Butler Lampson, Robert Morris, Jerry Saltzer, and Joel Emer

Parallelism is a major theme at SOSP/OSDI

Real problem in practice, from day 1

Parallel programming is either:

- **a cakewalk:** No sharing between computations
- **a struggle:** Sharing between computations
 - ▶ race conditions
 - ▶ deadly embrace
 - ▶ priority inversion
 - ▶ lock contention
 - ▶ ...

SOSP/OSDI is mostly about avoiding struggling for programmers

Parallelism is a major theme before SOSP

An example: Stretch [IBM TR 1960]:

Several forms of
parallelism

- User-generated parallelism
- I/O parallelism
- Instruction-level parallelism

MULTIPROGRAMMING STRETCH: FEASIBILITY CONSIDERATIONS

by

E. F. Codd
E. S. Lowry
E. McDonough
C. A. Scalzi

ABSTRACT

The tendency towards increased parallelism in computers is noted. Exploitation of this parallelism presents a number of new problems in machine design and in programming systems. Minimum requirements for successful concurrent execution of several independent problem programs are discussed. These requirements are met in the Stretch system by a carefully balanced combination of built-in logic and programmed logic. Techniques are described which place the burden of the programmed logic on system programs (supervisory program and compiler) rather than on problem programs.

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

In reality phases overlap and changes happened gradually

Trend: More programmers must deal with parallelism

Talk is **not** comprehensive

Phase 1: Time sharing

Many users, one computer

- Often 1 processor



[IBM 7094, 1962]

Standard approach: batch processing

Run one program to completion, then run next

A pain for interactive debugging [SJCC 1962]:

In part, this effect has been due to the fact that as elementary problems become mastered on the computer, more complex problems immediately become of interest. As a result, larger and more complicated programs are written to take advantage of larger and faster computers. This process inevitably leads to more programming errors and a longer period of time required for debugging. Using current batch monitor techniques, as is done on most large computers, each program bug usually requires several hours to eliminate, if not a complete day. The only alternative presently available is for the programmer to attempt to debug directly at the computer, a process which is grossly wasteful of computer time and hampered seriously by the poor console communication usually available. Even if a typewriter is the console, there are usually lacking the sophisticated query and response programs which are vitally necessary to allow effective interaction. Thus, what is desired is to drastically increase the rate of interaction between the programmer and the computer without large economic loss and also to make each interaction more meaningful by extensive and complex system programming to assist in the man-computer communication.

Time-sliced at 8-hour shifts [<http://www.multicians.org/thvv/7094.html>]:

IBM had been very generous to MIT in the fifties and sixties, donating or discounting its biggest scientific computers. When a new top of the line 36-bit scientific machine came out, MIT expected to get one. In the early sixties, the deal was that MIT got one 8-hour shift, all the other New England colleges and universities got a shift, and the third shift was available to IBM for its own use. One use IBM made of its share was yacht handicapping: the President of IBM raced big yachts on Long Island Sound, and these boats were assigned handicap points by a complicated formula. There was a special job deck kept at the MIT Computation Center, and if a request came in to run it, operators were to stop whatever was running on the machine and do the yacht handicapping job immediately.

Time-sharing: exploit user parallelism

The basic technique for a time-sharing system is to have many persons simultaneously using the computer through typewriter consoles with a time-sharing supervisor program sequentially running each user program in a short burst or quantum of computation. This sequence, which in the most straightforward case is a simple round-robin, should occur often enough so that each user program which is kept in the high-speed memory is run for a quantum at least once during each approximate human reaction time (~.2 seconds). In this way, each user sees a computer fully responsive to even single key strokes each of which may require only trivial computation; in the non-trivial cases, the user sees a gradual reduction of the response time which is proportional to the complexity of the response calculation, the slowness of the computer, and the total number of active users. It should be clear, however, that if there are n users actively requesting service at one time, each user will only see on the average 1/n of the effective computer speed. During the period of high interaction rates while debugging programs, this should not be a hindrance since ordinarily the required amount of computation needed for each debugging computer response is small compared to the ultimate production need.

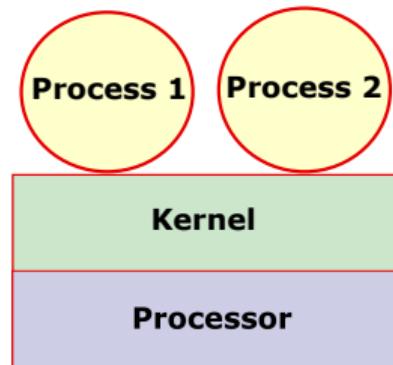
CTSS [SJCC 1962]

Youtube: "ctss wgbh" [<https://www.youtube.com/watch?v=Q07PhW5sCEk>, 1963]

Many programs: an opportunity for I/O parallelism

Multiprogramming [Stretch 1960, CTSS 1962]:

- On I/O, kernel switches to another program
- Later kernel resumes original program
- Benefit: higher processor utilization



Kernel developers deal with I/O concurrency

supervisor < 5K 36-bit-words

Programmers write sequential code

$q = 16 \text{ m.s.}$ (based on 1% switching overhead)

$w_q = 120 \text{ words}$ (based on one IBM 1301 model
2 disc unit without seek or latency
times included)

$t_r \leq 8Nf \text{ sec.}$ (based on programs of (32k)f
words)

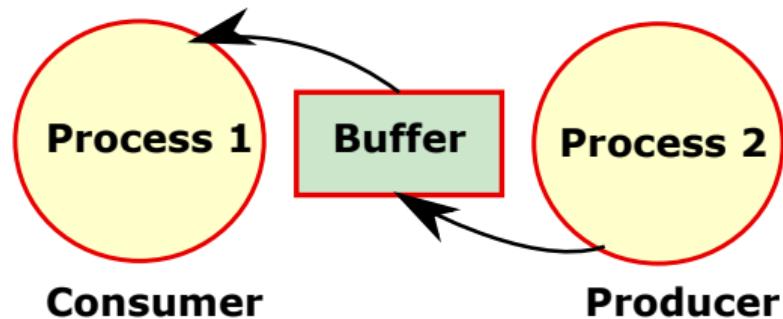
$t_a \leq \log_2 (1000/N)$ (based on $t_u = 16 \text{ sec.}$)

$t_o \leq 8$ (based on a maximum program size of
32K words)

Challenge: atomicity and coordination

Example: the THE operating system [EWD123 1965, SOSP 1967]

- Technische Hogeschool Eindhoven (THE)
- OS organized as many “sequential” processes
 - ▶ A driver is a sequential process



The THE solution: semaphores

Finally I should like to thank the members of the program committee who asked for more information on the synchronizing primitives and some justification of my claim to be able to prove logical soundness a priori. In answer to this request the appendix has been added, of which I hope that it gives the desired information and justification.

Appendix

The Synchronizing Primitives.

Explicit mutual synchronization of parallel sequential processes is implemented via so-called "semaphores". They are special purpose integer variables allocated in the universe in which the processes are embedded, they are initialized (with the value 0 or 1) before the parallel processes themselves are started. After this initialization the parallel processes will access the semaphores only via two very specific operations, the so-called synchronizing primitives. For historical reasons they are called the P-operation and the V-operation.

[The “THE” multiprogramming system, First SOSP]

The THE solution: semaphores

Still in practice today

```
28 #include <linux/compiler.h>
29 #include <linux/kernel.h>
30 #include <linux/export.h>
31 #include <linux/sched.h>
32 #include <linux/semaphore.h>
33 #include <linux/spinlock.h>
34 #include <linux/ftrace.h>
35
36 static __attribute__((noinline)) void __down(struct semaphore *sem);
37 static __attribute__((noinline)) int __down_interruptible(struct semaphore *sem);
38 static __attribute__((noinline)) int __down_killable(struct semaphore *sem);
39 static __attribute__((noinline)) int __down_timeout(struct semaphore *sem, long timeout);
40 static __attribute__((noinline)) void __up(struct semaphore *sem);
41
42 /**
43  * down - acquire the semaphore
44  * @sem: the semaphore to be acquired
45  *
46  * Acquires the semaphore. If no more tasks are allowed to acquire the
47  * semaphore, calling this function will put the task to sleep until the
48  * semaphore is released.
49 *
```

P & V?

passing (P) and release (V) [EWD35]

kan duren. We geven dit aan met een P (van Passering); vooruitlopend op latere behoeften representeren we de statement "SX:= true" door "V(SX) -met de V van Vrijgave. (Deze terminologie is ontleend aan het spoorwegwezen: in een eerder stadium heetten de gemeenschappelijke logische variabelen "Seinpalen" en als hun naam met een S begint, dan is dat nog een reminiscentie daaraan.) De text van de programma's

portmanteau try to reduce (P) and increase (V) [EWD51]

1.1. De operatie V ("Verhoog").

De operatie V kan betrekking hebben op een willekeurig aantal verschillende seinpalen, dus bv. "V(S1,S2,S3)". Als deze operatie in een van de machines voorkomt

EWD51 - 2

-in ons voorbeeld moeten dan S1, S2 en S3 voor deze machine toegankelijke seinpalen zijn- dan is het effect, dat alle opgegeven seinpalen in één ondeelbare handeling met 1 verhoogd worden.

1.2. De operatie P ("Prolaag").

Time-sharing and multiprocessor parallelism

Early computers with several processors

- For example, Burroughs B5000 [1961]

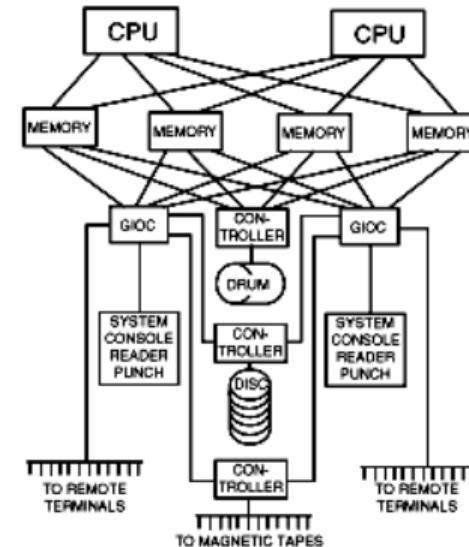
Much attention paid to parallelism:

- Amdahl's law for speedup [AFIPS 1967]
- Traffic control in Multics [Saltzer PhD thesis, 1966]
- Deadlock detection
- Locking ordering
- ...

i.e., Most ideas that you will find in an intro OS text

Serious parallel applications

- E.g., Multics Relational Database Store
 - Ran on 6-processor computer at Ford



[GE 645, Multics Overview 1965]

Time-sharing on minicomputers: just I/O parallelism

Minicomputers had only one processor

Multiprocessor parallelism de-emphasized

- Other communities develop processor parallelism further (e.g., DBs).



For example: Unix [SOSP 1973]

- Unix kernel implementation specialized for uniprocessors
- User programs are sequential
 - ▶ Pipelines enable easy-to-use user-level producer/consumer

```
$ cat todo.txt | sort | uniq | wc  
273      1361      8983
```

\$

To put my strongest concerns in a nutshell:

1. We should have some ways of coupling programs like garden hoses--screw in another segment when it becomes ~~then~~ it becomes necessary to massage data in another way.

This is the way of IO also.

[McIlroy 1964]

Phase 2: Client/server computing

Computers inexpensive enough to give each user her own

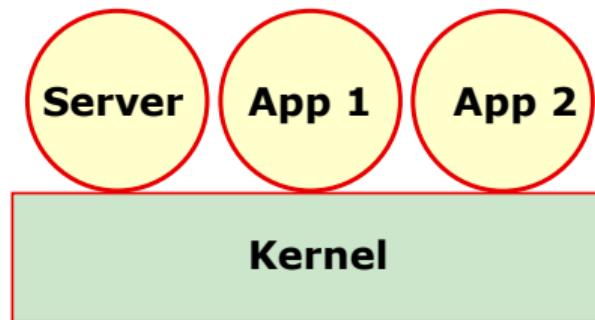
Local-area networks and servers allow users to collaborate



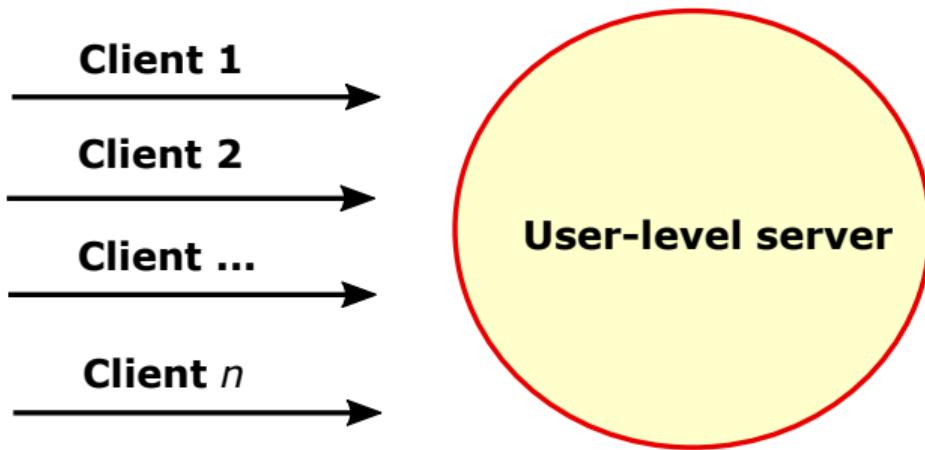
[Alto, Xerox PARC, 1975]

Goal: wide range of services

Idea: allow non-kernel programmers to implement services by supporting servers at user level

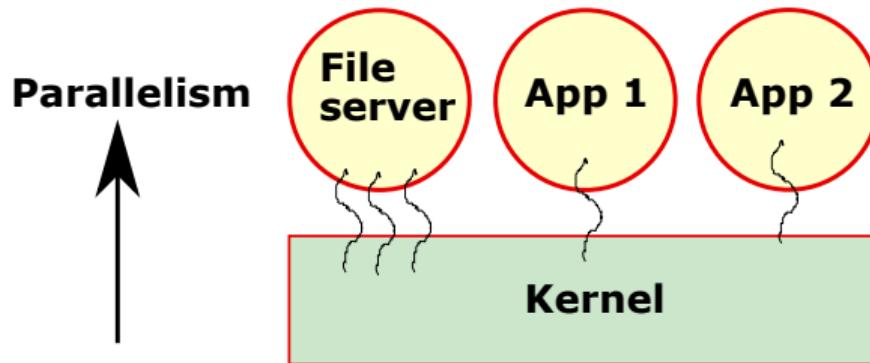


Challenge: user-level servers must exploit I/O concurrency



Some of the requests involve expensive I/O

Solution: Make concurrency available to servers



Kernel exposes interface for server developers

- Threads
- Locks
- Condition variables
- ...

Result: many high-impact ideas

New operating systems (Accent [SOSP 1981]/Mach [SOSP 1987], Topaz/Taos, V [SOSP 1983], etc.)

- Support for multithreaded servers encourages microkernel design

Much impact: e.g., Pthreads [POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995)]

- Supported now by many widely-used operating systems

New programming languages (Mesa [SOSP 1979] , Modula2+, etc.)

- If you have multithreaded programs, you want automatic garbage collection
- Other nice features too (e.g., monitors, continuations)
- Influenced Java, Go, ...

Programming with threads

An introduction to programming with threads [Birrell tutorial 1989]

This is fairly straightforward, but there are still some subtleties. Notice that when a consumer returns from the call of "Wait" his first action after re-locking the mutex is to check once more whether the linked list is empty. This is an example of the following general pattern, which I strongly recommend for all your uses of condition variables.

```
WHILE NOT expression DO Thread.Wait(m,c) END;
```

Case study: Cedar and GVX window system [SOSP 1993]:

- Many threads
- Written over a 10 year period, 2.5M LoC

Design patterns:

Table 4. Static Counts
Cedar

		GVX	
Defer work	108	31%	77
Pumps			33%
General pumps	48	14%	33
Slack processes	7	2%	2
Sleepers	67	19%	15
Oneshots	25	7%	11
Deadlock avoid	35	10%	6
Task rejuvenate	11	3%	0
Serializers	5	1%	7
Encapsulated fork	14	4%	5
Concurrency			2%
exploiters	3	1%	0
Unknown or other ²	25	7%	78
TOTAL	348	100%	234
			100%

Bugs:

```
WHILE NOT (condition) DO WAIT cv END  
not the  
IF NOT (condition) THEN WAIT cv
```

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?

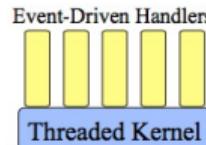
▫ **No:** important for high-end servers (e.g. databases).

▫ **But, avoid threads wherever possible:**

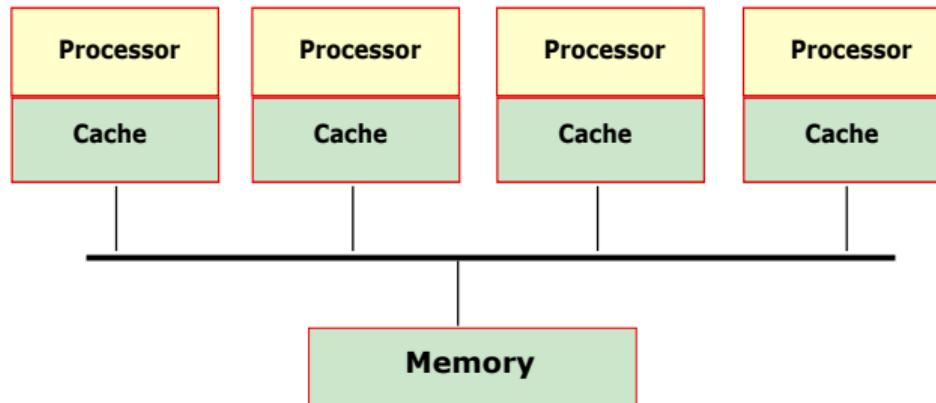
- Use events, not threads, for GUIs, distributed systems, low-end servers.

- Only use threads where true CPU concurrency is needed.

- Where threads needed, isolate usage in threaded application kernel: keep most of code single-threaded.



Phase 3: Shared-memory multiprocessors (SMPs)



Mid 90s: inexpensive x86s multiprocessors showed up with 2-4 processors

Kernel and server developers had take multiprocessor parallelism seriously

- E.g., Big Kernel Lock (BKL)
- E.g., Events **and** threads

Much research on large-scale multiprocessors in phase 3

Scalable NUMA multiprocessors: BBN Butterfly, Sequent, SGI, Sun, Thinking Machines, ...

Many papers on scalable operating systems:

- Scalable locks [TOCS 1991]
- Efficient user-level threading [SOSP 1991]
- NUMA memory management [ASPLOS 1996]
- Read-copy update (RCU) [PDCS 1998, OSDI 1999]
- Scalable virtual machines monitor [SOSP 1997]
- ...



[VU, Tanenbaum, 1987]

Uniprocessor performance keeps doubling in phase 3

No real need for expensive parallel machine

IS PARALLEL COMPUTING DEAD?

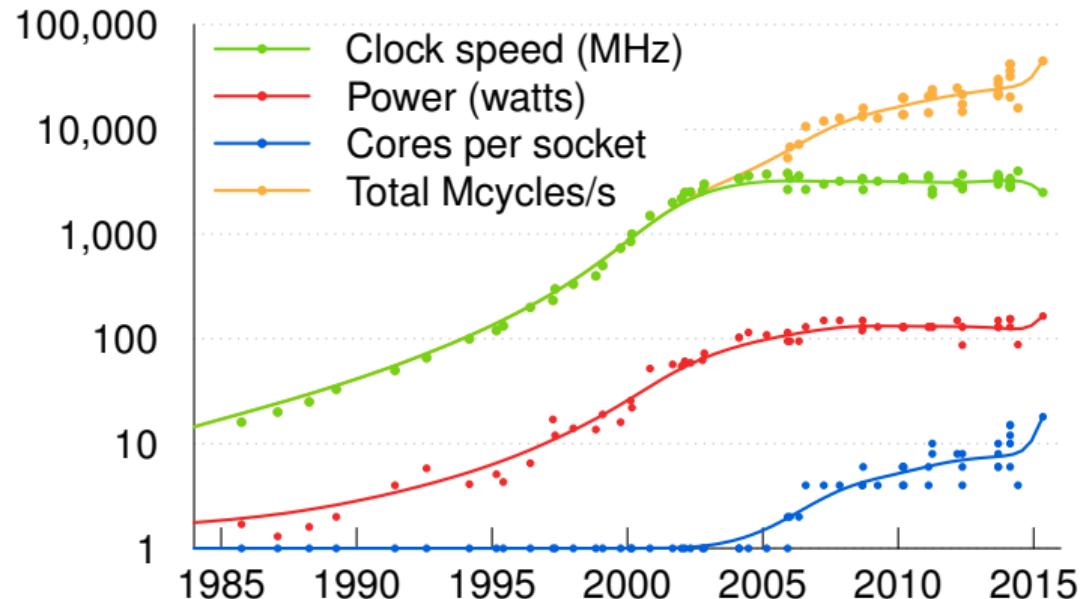
Ken Kennedy, Director, CRPC

The announcement that Thinking Machines would seek Chapter 11 bankruptcy protection, although not unexpected, sent shock waves through the high-performance computing community. Coupled with the well-publicized problems of Kendall Square Research and the rumored problems of Intel Supercomputer Systems Division, this event has led many people to question the long-term viability of the parallel computing industry and even parallel computing itself. Meanwhile, the dramatic strides in the performance of scientific workstations continues to squeeze the market for parallel supercomputing. On several recent occasions, I have been asked whether parallel computing will soon be relegated to the trash heap reserved for promising technologies that never quite make it. Washington certainly seems to be looking in the other direction--agency program managers, if they talk of high-performance computing at all, seem to view it as a small and relatively unimportant subcomponent of the National Information Infrastructure.

Is parallel computing really dead? At the very least, it is undergoing a major transition. With the

[<http://www.crpc.rice.edu/newsletters/oct94/director.html>]

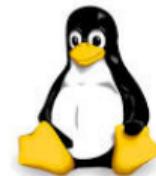
Phase 4: multicore processors



Achieving performance on commodity hardware requires exploiting parallelism

Scalable operating systems return from the dead

Several parallel computing companies switch to Linux



Linux Support for NUMA Hardware

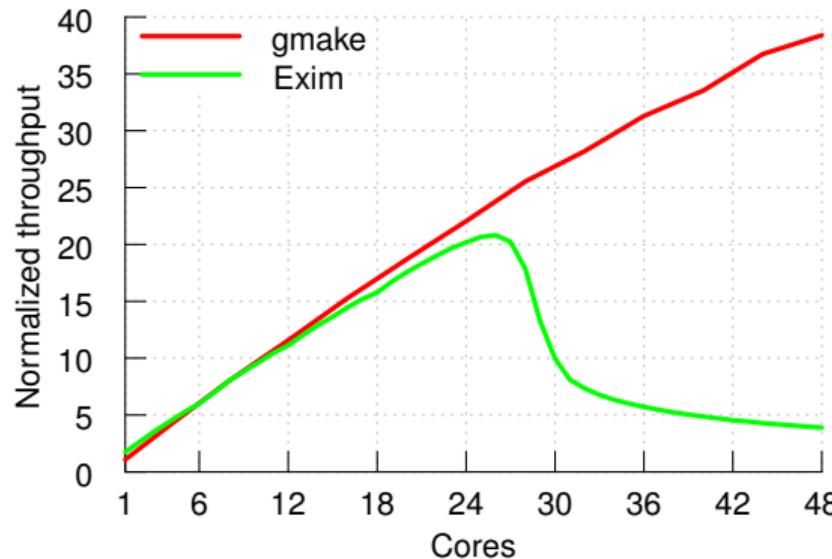
Large count multiprocessors are being built with non-uniform memory access (NUMA) times - access times that are dependent upon where within the machine a piece of memory physically resides. For optimal performance, the kernel needs to be aware of where memory is located, and keep memory used as close as possible to the user of the memory. Examples of NUMA machines include the NEC Azusa, the IBM x440 and the IBM NUMAQ.

The 2.5 Linux kernel includes many enhancements in support of NUMA machines. Data structures and macros are provided within the kernel for determining the layout of the memory and processors on the system. These enable the VM subsystem to make decisions on the optimal placement of memory for processes. This topology information is also exported to user-space via sysfs.

In addition to items that have been incorporated into the 2.5 Linux kernel, there are NUMA features that have been developed that continue to be supported as patchsets. These include NUMA enhancements to the scheduler, multipath I/O and a user-level API that provides user control over the allocation of resources in respect to NUMA nodes.

This page provides links to information about the various Linux on NUMA projects. Discussions related to Linux on NUMA take place on the lse-tech mailing list and on the linux kernel mailing list.

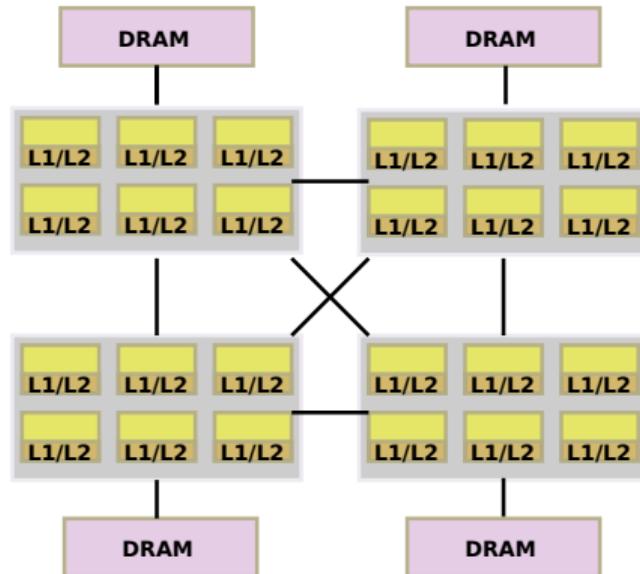
Many applications scale well on multicore processors



But, more applications stress parallelism in operating systems

- Some tickle new scalability bottlenecks
- Exim contends on a *single* reference counter in Linux [OSDI 2010, SOSP 2013]

Cache-line fetches are expensive



Read cache line written by
another core: expensive!
100–10000 cycles
(contention)

For reference, a creat system
call costs 2.5K cycles

Avoiding cache-line sharing is challenging

Consider read-write lock

```
struct read_write_lock {  
    int count;           // -1, write mode; > 0, read mode  
    list_head waiters;  
    spinlock wait_lock;  
}
```

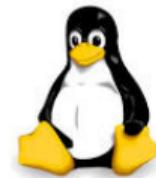
Problem: to acquire lock in **read** mode requires **modifying** count

- Fetching a remote cache line is expensive
- Many readers can cause performance collapse

Read-copy update (RCU) becomes popular

Readers read shared data without holding **any** lock

- Mark enter/exit read section in per-core data structure



Writer makes changes available to readers using an atomic instruction

- Free node when all readers have left read section

Lots of struggling to scale software [Recent OSDI/SOSP papers]

What will phase 4 mean for OS community?

What will commodity hardware look like?

- 1000s of unreliable cores?
- Many heterogeneous cores?
- No cache-coherent shared memory?



Barrelfish [SOSP 2009]

How to avoid struggling for programmers?

- Exploit transactional memory [ISCA 1993]?
- Develop frameworks for specific domains?
 - ▶ MapReduce [OSDI 2004], ..., GraphX [OSDI 2014], ...
- Develop principles that make systems scalable by design? [SOSP 2013]



Stepping back: some observations

SOSP/OSDI papers had tremendous impact

- Many ideas can be found in today's operating systems and programming languages

Processes/threads have been good for managing computations

- OS/X 10.10.5 launches 1158 threads, 308 processes on 4-core iMac at boot

Shared memory and locks have worked well for concurrency and parallelism

Events vs. threads – have both?

Rewriting OSes to make them more scalable has worked surprisingly well (so far)

- From big kernel lock to fine-grained parallelism

Summary

Parallelism has moved up the software stack driven by changes in commodity hardware

- More and more programmers are writing parallel code

Today: to achieve performance on commodity hardware
programmers **must** use parallelism

- Phase 1: time sharing (foundational ideas)
- Phase 2: client/server (concurrent servers)
- Phase 3: SMPs (parallel kernels and servers)
- Phase 4: multicore (all applications parallel)

Summary

Parallelism has moved up the software stack driven by changes in commodity hardware

- More and more programmers are writing parallel code

Today: to achieve performance on commodity hardware
programmers **must** use parallelism

Prediction: Many more SOSP/OSDI papers on parallelism

- Phase 1: time sharing (foundational ideas)
- Phase 2: client/server (concurrent servers)
- Phase 3: SMPs (parallel kernels and servers)
- Phase 4: multicore (all applications parallel)

The network and the OS

David Clark
MIT CSAIL
October, 2015

From the specific to the cosmic

- Early issues were pragmatic and “mechanical”.
 - How to structure and position the code that implemented the protocols.
 - Performance.
- Later issues were more fundamental:
 - What does it mean for a machine to be connected to the rest of the world?
 - Security, availability

Structure

- To understand the issues of structure, must understand what is distinctive about implementing network protocols.
 - Start there, then look at implications for the OS.

What is different about net I/O?

- Variable size units (packets and application data).
- Malformed content and size.
- Internet connected heterogeneous machines over heterogeneous networks.
 - First (and in some sense only) goal was interoperation.
 - Byte order, 9 bit bytes, etc.
- Unpredictable arrival/transmission.
- Must be processed to demultiplex.
 - Trustworthy processing.

A 1986 perspective

Our state of understanding in 1986:

- A slide of mine from the time.

There was deep confusion as to how to move from protocol specification to protocol implementation.

SOME GENERAL OBSERVATIONS ABOUT PROTOCOLS

- ~THEY ARE DIFFICULT TO UNDERSTAND AND CODE.
- ~THE IMPLEMENTATIONS ARE OFTEN VERY LARGE.
- ~THEY DO NOT PERFORM VERY WELL.

WHAT IS THE CAUSE OF THESE PROBLEMS?

- ~THE PROTOCOL DESIGN?
- ~THE PROTOCOL IMPLEMENTATION?
- ~SOMETHING ELSE?

Implementing a protocol

- The stages in our understanding. What was the challenge?
 - Implementing the state machine.
 - Marshalling the packet fields.
 - Dealing with errors.
 - Processing 32 bit numbers.
 - Copying the data.
 - Dealing with congestion control.
 - Dispatching the packet to correct connection.
 - Dealing with layers

Where to put the software?

Protocol in the OS?

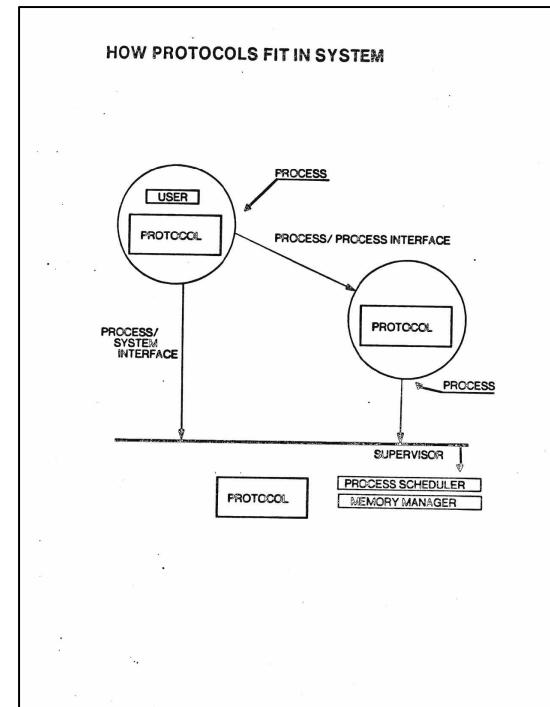
- Low overhead.
- Nasty programming environment.
- Run all the code at interrupt time?

Protocol in the application process?

- No asynchrony.
- Easy invocation.

Protocol in a separate process?

- High cost to invoke.
- Asynchronous execution.



Waiting for events

- Protocols have an odd (by the thinking of the day) structure.
 - They wait for multiple events.
 - A user event, a network event, a timer event.
- Many interprocess scheduling mechanisms required the waiting process to wait on one event.

Performance

- We had to learn the relative cost of different actions.
 - Processing a header.
 - Scheduling a process/thread.
 - Setting a timer.
 - Taking an interrupt.
 - Copying the data.
 - Dispatching the packet.

Protocols can be simple

Implementation of TCP
input routine for Xerox
Alto.

It fit on one page.
– It does call subroutines...

```
tcp.bcp1          30-Apr-81 11:48:22
Page 2

//-----  
let tcpReceive(soc,pbi) be  
//-----  
  
[  
let ip = lv pbi>>INBPI.INHeader  
let itp = ip + (ip>>INHeader.ihl lshift 1)  
let idp = ip + (itp>>dataOffset lshift 1) // offset in words  
let idataLng = (ip>>INHeader.totalLength - (ip>>INHeader.ihl lshift 2)) -  
    (ip>>dataOffset lshift 2)  
  
// compute incoming tcp checksum  
  
unless INCompareForeignPort(pbi,INsoc.foreignPort) return  
if itp>>f.rst eq 1 then [ cleanup("reset");return ]  
if itp>>f.syn eq 1 then [ inc itp>>sn = (itp>>sn + 1)  
    DoubleIncrement(lv itp)>>1 ]  
test opening pr 1 //this code updates things based on incoming ack value  
ifso if itp>>f.ack eq 1 then [  
    let diff = otp>>sn2 + odataLng - itp>>ack2  
    if diff eq -1 & otp>>f.fin then  
        [ otp>>sn2 = otp>>sn2 + 1;diff = 0;otp>>f.fin = 0;  
        if otp>>urg > closing + 1; if closing eq 3 then Wl("Closed")]  
    if diff ls 0 then [ DMove(lv otp)>>ack,lv itp>>ack )  
        DMovet(lv otp)>>ack,lv itp>>sn); send = true; return ]  
    if otp>>f.urg eq 1 then  
        [ otp>>urg = otp>>urg + odataLng + diff  
        if otp>>urg ls 0 then otp>>f.urg = 0  
    ]  
    if diff eq -otp>>f.fin then sendCount = 0  
    if diff ls odataLng then  
        [ for i = 0 to diff - 1 do  
            otp>>brf = otp>>brf + (odataLng - diff))  
            otp>>sn2 = otp>>sn2 + odataLng - diff;odataLng = diff;  
        ]  
    ]  
ifnot [  
    if (itp>>f & 22b) ne 22b then [ error(1);return ] //must have Syn and Ack  
    if (itp>>ack2 ne 1 then [ error(2);return ] // bad ack value  
    otp>>sn2 = 1; send = true  
    otp>>f.ack = 30h // ack and eol  
    DMovet(lv otp)>>ack,lv itp>>sn)  
    opening = 3  
    Wl("Open")  
]  
  
// next line is:diff = otp>>ack - itp>>sn  
let diff = DoubleDifference(lv otp)>>ack,lv itp>>sn)  
if diff ls 0 then [ Ws("X");return ] // packet out of sequence  
Ws("Y")  
  
if itp>>f.fin eq 1 then  
    [ if closing eq 0 then [ otp>>f.fin = 1;closing = closing + 1]  
        send = true  
        closing = closing + 1;if closing eq 3 then Wl("Closed")]  
  
if idataLng gr 0 then  
    [ for i = diff to idataLng - 1 do  
        tcpProcessByte(idp>>i)  
        send = true  
        otp>>window = otp>>window - idataLng + diff  
    ]  
if diff ls idataLng then  
// next lines are:otp>>ack = itp>>sn + idataLng + itp>>f.fin  
[  
    DoubleIncrement(lv itp)>>sn,idataLng + itp>>f.fin  
    DMovet(lv otp)>>ack,lv itp>>sn)  
]  
return
```

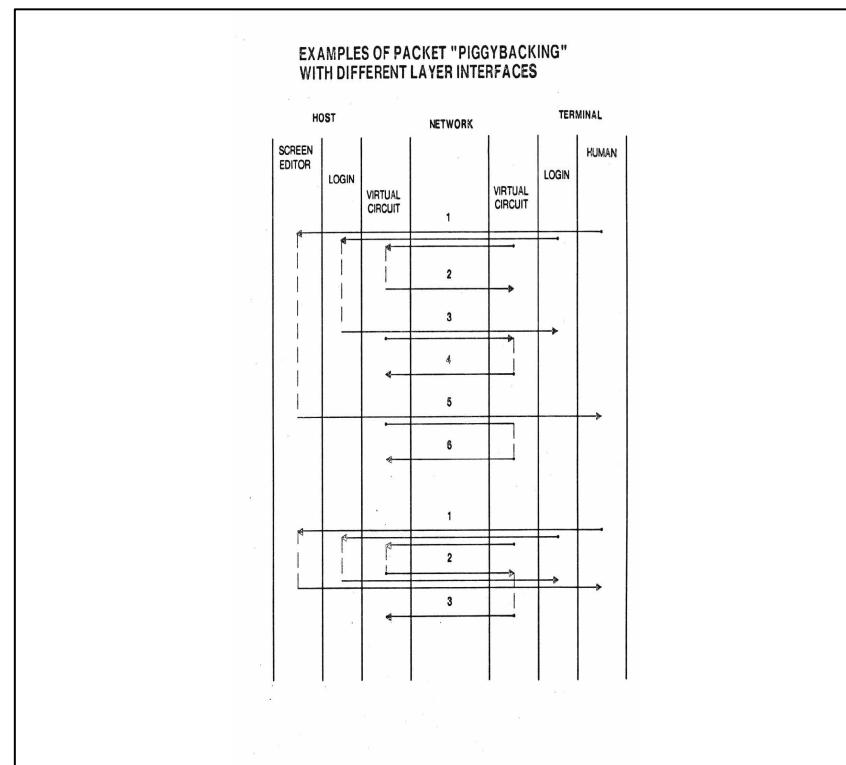
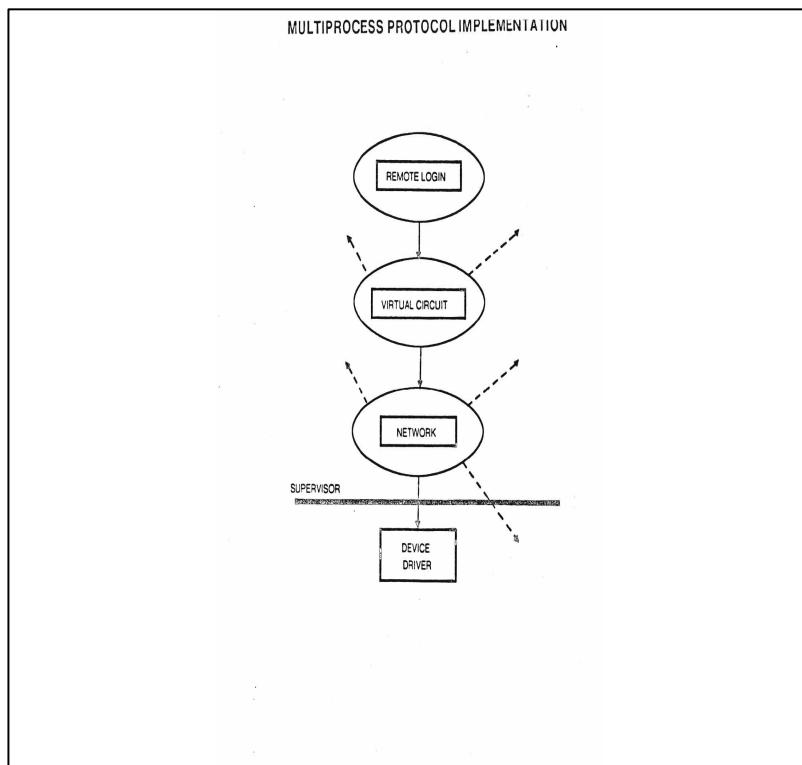
Layers of protocol

- Link, IP, TCP, app.
- How should the code be structured?
 - Obvious (but bad) idea: structure a layer as a process.
 - Why? It takes (much) longer to schedule a process than process a packet.
- Layering is a device for specification, not code structure.

An example--TRIPOS

- TRIPOS (Cambridge University) was wonderful little OS that used processes for most system functions. (The micro-kernel philosophy.)
 - Interprocess communication by pointer, not copy.
 - highly efficient.
 - Network code structured as three processes.
 - Network, transport, remote login.
 - 54 process wakeups to exchange a character.
 - Recoding as one process: 10x smaller, 10x faster

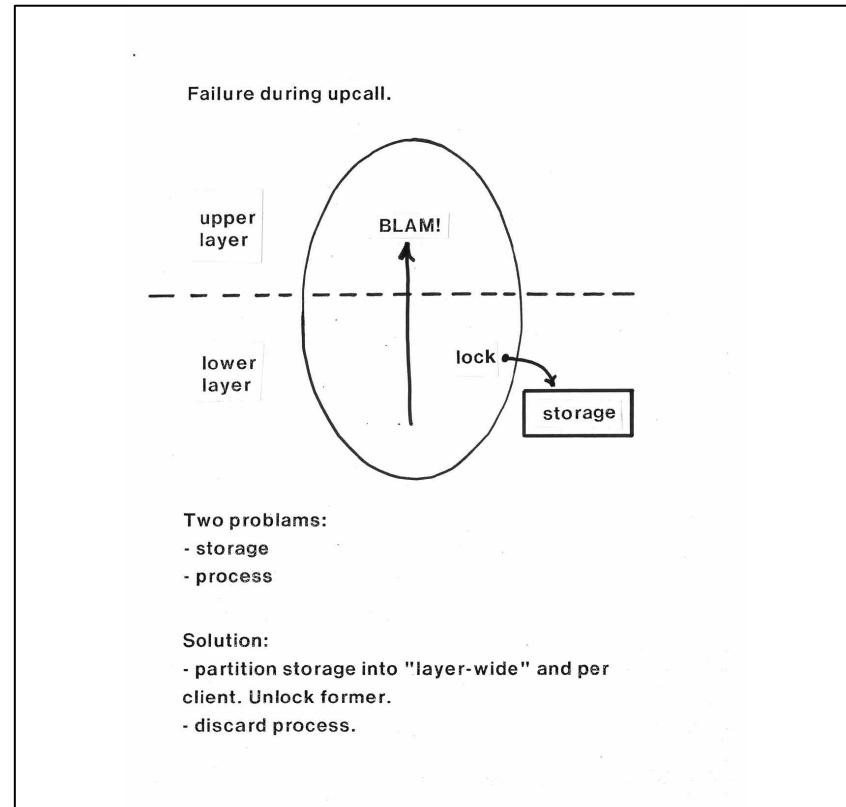
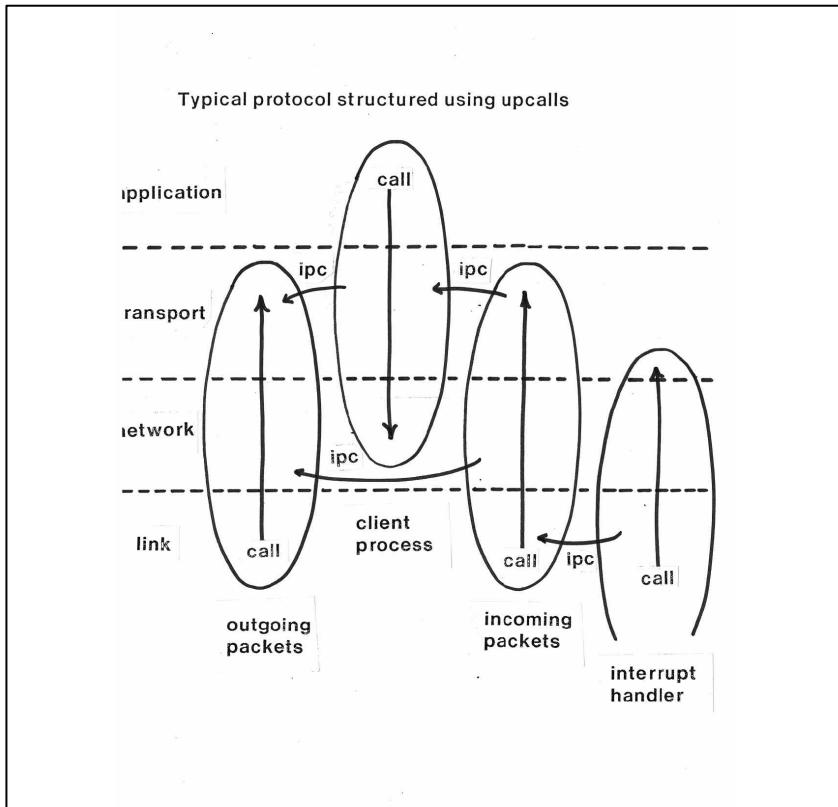
The consequence of processes



Emerging ideas

- “The Structuring of Systems Using Upcalls”
 - David Clark, SOSP, 1985
- “Layered Multiplexing Considered Harmful”
 - David Tennenhouse, First International Workshop on High Speed Networking, 1989

Some pictures of upcalls



Fixing other performance problems

- G. Varghese and T. Lauck. Hashed and hierarchical timing wheels: data structures for the efficient implementation of a timer facility. In *Proceedings of the eleventh ACM Symposium on Operating systems principles* (SOSP '87). ACM, New York, NY, USA,

Packet processing

- Clark, D.D.; Romkey, J.; Salwen, H., "An analysis of TCP processing overhead," in *Local Computer Networks, 1988., Proceedings of the 13th Conference on*, vol., no., pp.284-291, 10-12 Oct 1988
- TCP packet receipt:
 - Sender of data: 191-235 instructions
 - Receiver of data, 186 instructions.
 - Set a timer: 35 (used timing wheel algorithm)
 - Internet protocol: ~60

A range of topics

- Early issues were performance
- Network software design
- Homogeneity
- Co-processing
- Small machines
 - From Alto, PC, (to IoT).
- Parallel machines
- Alternative network semantics
- High-level implications of connectivity to the world
 - Security, availability, etc.
- Virtual networks and virtual computers
- Speed of light

The recurring structural issue

- Networks have a distinct set of issues to solve.
 - Resource allocation, security, managing delivery.
- But they do not know what they are being used for. (The end to end model).
 - What is core and what is overlay?
- TCP persists because we found no other general service model.
 - The alternative is to push to the app the implementation of the desire semantics. (UDP.)
 - But then app designer is implementing the protocol. See earlier part of talk.
 - Is the protocol (e.g., transport) a core service?
- The net cannot trust the host, the OS cannot trust the app, the app cannot trust any of them, and the resulting system should have some sort of reliability.

The Rise of Cloud Computing Systems

Jeff Dean
Google, Inc.

(Describing the work of thousands of people!)



Utility computing: Corbató & Vyssotsky, "Introduction and Overview of the Multics system", AFIPS Conference, 1965.







How Did We Get to Where We Are?

Prior to mid 1990s: Distributed systems emphasized:

- modest-scale systems in a single site (Grapevine, many others), as well as
- widely distributed, decentralized systems (DNS)

Adjacent fields

High Performance Computing:

Heavy focus on performance, but not on fault-tolerance

Transactional processing systems/database systems:

Strong emphasis on structured data, consistency

Limited focus on very large scale, especially at low cost

Caveats

Very broad set of areas:

Can't possibly cover all relevant work

Focus on few important areas, systems, and trends

Will describe context behind systems with which I am most familiar

What caused the need for such large systems?

Very **resource-intensive interactive services** like **search** were key drivers

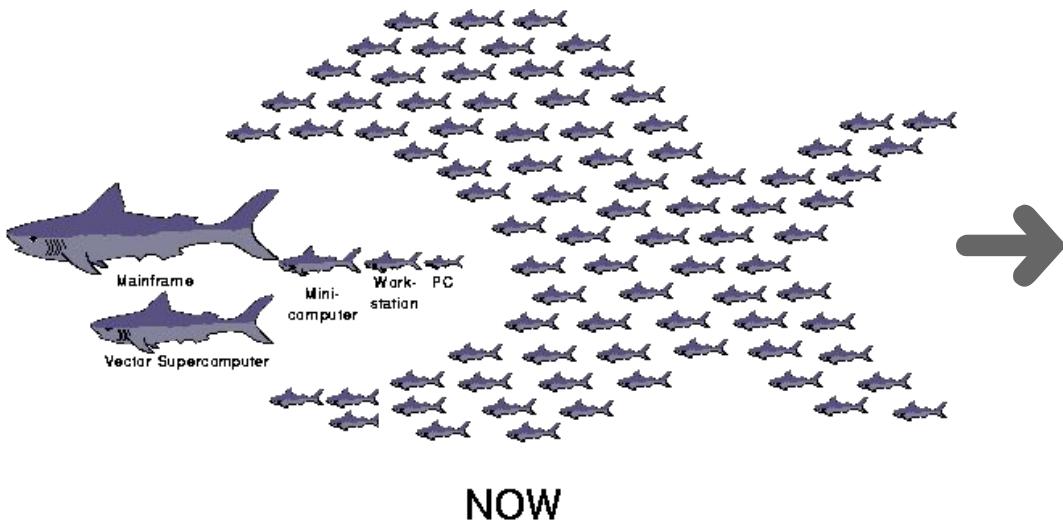
Growth of web

- ... from millions to hundreds of billions of pages
- ... and need to index it all,
- ... and search it millions and then billions of times per day
- ... with sub-second latencies



Inktomi®

The Berkeley NOW Project

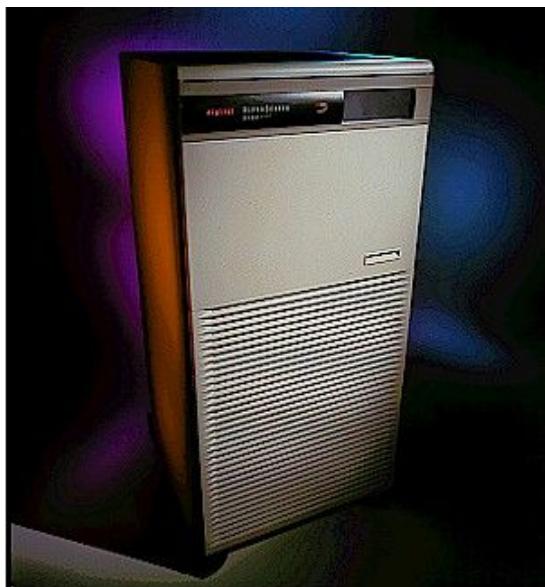


A Case for Networks of Workstations: NOW, Anderson, Culler, & Patterson. IEEE Micro, 1995

Cluster-Based Scalable Network Services, Fox, Gribble, Chawathe, Brewer, & Gauthier, SOSP 1997.

My Vantage Point

Joined DEC WRL in 1996 around

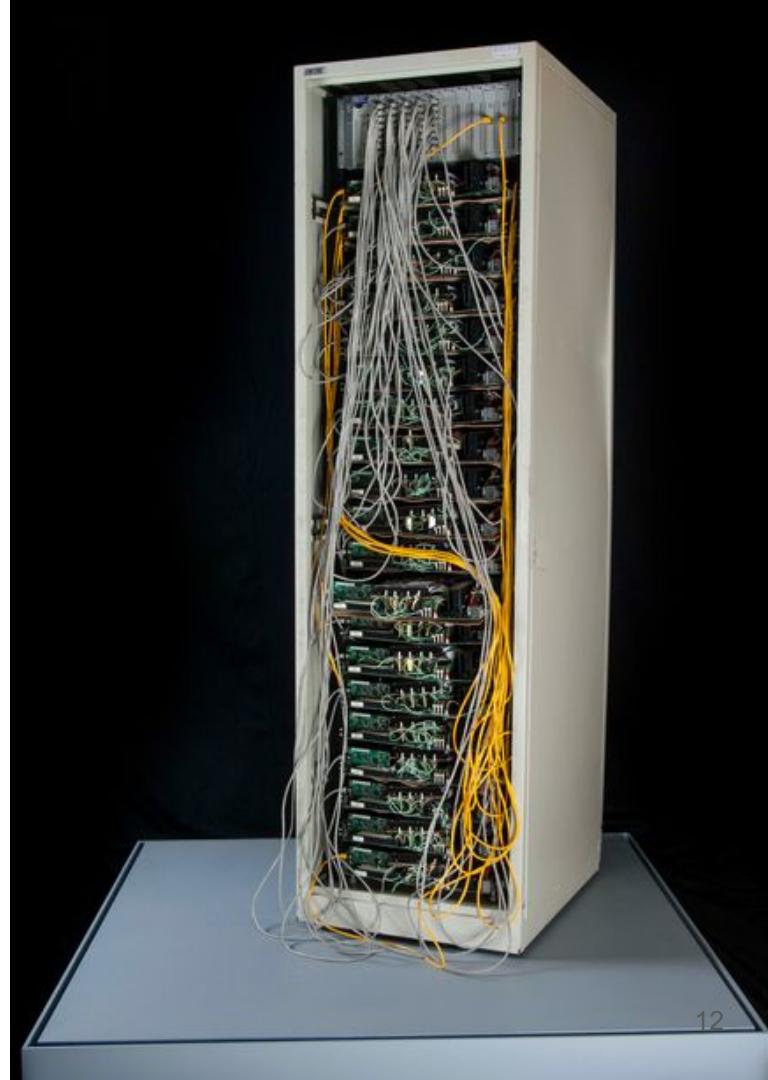


My vantage point, continued:
Google, circa 1999

Early Google tenet:
Commodity PCs give high perf/\$

Commodity components **even better!**

Aside: use of cork can land your computing platform in the Smithsonian



At Modest Scale: Treat as Separate Machines

```
for m in a7 a8 a9 a10 a12 a13 a14 a16 a17 a18  
a19 a20 a21 a22 a23 a24; do ssh -n $m "cd  
/root/google; for j in ``seq $i ${$i+3}``; do  
j2=`printf %02d $j`; f=`echo '$files' | sed  
s/bucket00/bucket$j2/g`; fgrun bin/buildindex  
$f; done' & i=${$i+4}; done
```

What happened to poor old a11 and a15?

At Larger Scale: Becomes Untenable



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

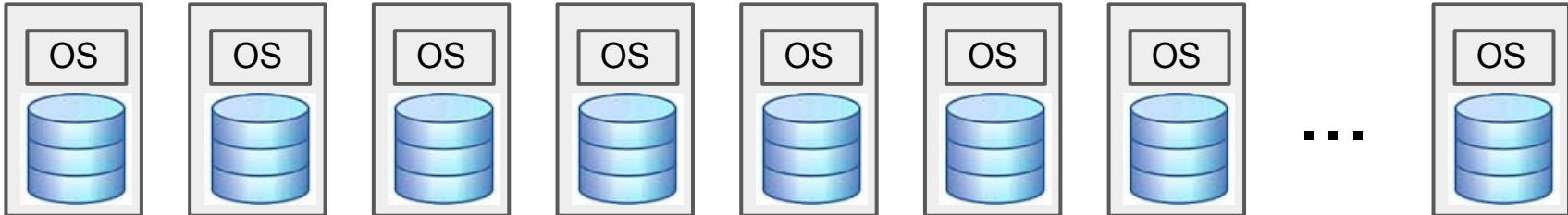
- ~1 **network rewiring** (rolling ~5% of machines down over 2-day span)
- ~20 **rack failures** (40-80 machines instantly disappear, 1-6 hours to get back)
- ~5 **racks go wonky** (40-80 machines see 50% packetloss)
- ~8 **network maintenances** (4 might cause ~30-min random connectivity losses)
- ~12 **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

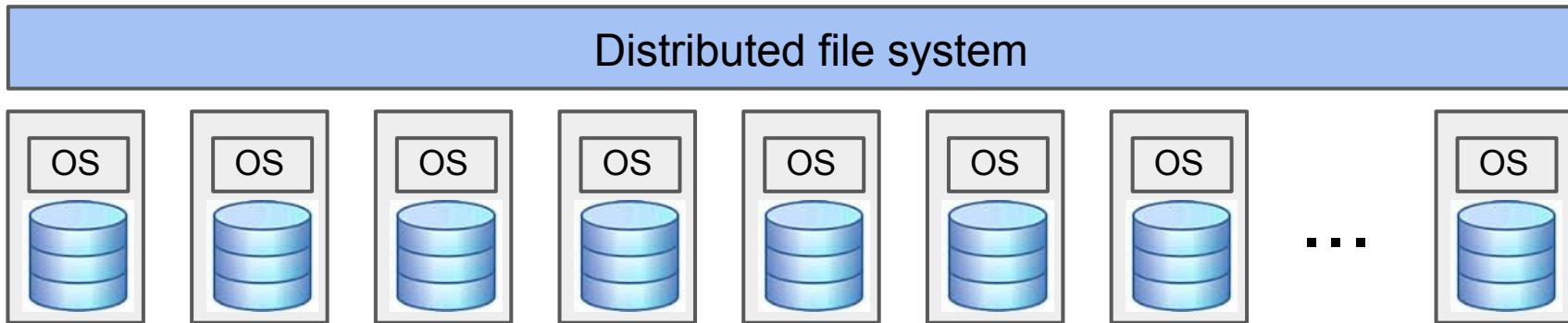
A Series of Steps, All With Common Theme:

Provide Higher-Level View Than
“Large Collection of Individual Machines”

Self-manage and self-repair as much as possible



First Step: Abstract Away Individual Disks



Long History of Distributed File Systems

Xerox Alto (1973), NFS (1984), many others:
File servers, distributed clients

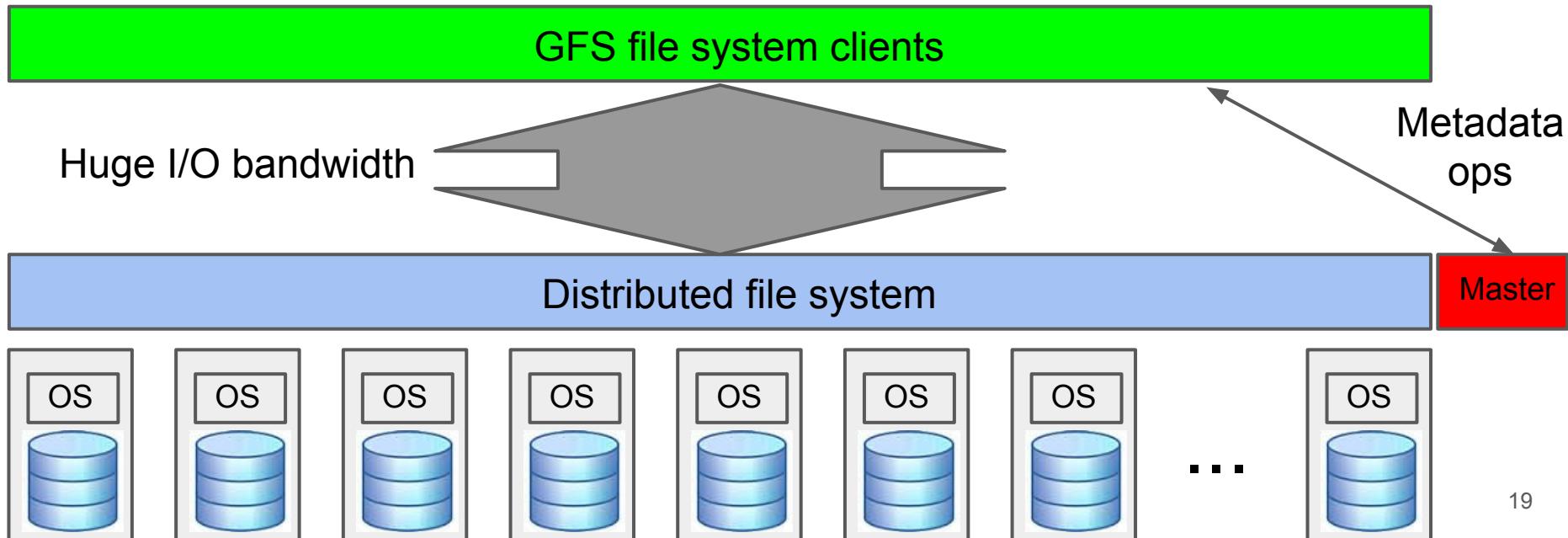
AFS (Howard et al. '88):
1000s of clients, whole file caching, weakly consistent

xFS (Anderson et al. '95):
completely decentralized

Petal (Lee & Thekkath, '95), Frangipani (Thekkath et al., '96):
distributed virtual disks, plus file system on top of Petal

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



Disks in datacenter basically self-managing



Successful design pattern:

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

Once you can store data, then you want to be able to process it efficiently

Large datasets implies need for highly parallel computation

One important building block:
Scheduling jobs with 100s or 1000s of tasks

Multiple Approaches

- Virtual machines
- “Containers”: akin to a VM, but at the process level, not whole OS

Virtual Machines

- Early work done by MIT and IBM in 1960s
 - Give separate users their own executing copy of OS
- Reinvigorated by Bugnion, Rosenblum *et al.* in late 1990s
 - simplify effective utilization of multiprocessor machines
 - allows consolidation of servers

Raw VMs: key abstraction now offered by cloud service providers

Cluster Scheduling Systems

- **Goal: Place containers or VMs on physical machines**
 - handle resource requirements, constraints
 - run multiple tasks per machine for efficiency
 - handle machine failures

Similar problem to earlier HPC scheduling and distributed workstation cluster scheduling systems

e.g. Condor [Litzkow, Livny & Mutkow, '88]

Many Such Systems

- Proprietary:
 - **Borg** [Google: Verma *et al.*, published 2015, in use since 2004]
(unpublished predecessor by Liang, Dean, Sercinoglu, *et al.* in use since 2002)
 - **Autopilot** [Microsoft: Isaard *et al.*, 2007]
 - **Tupperware** [Facebook, Narayanan slide deck, 2014]
 - **Fuxi** [Alibaba: Zhang *et al.*, 2014]
- Open source:
 - **Hadoop Yarn**
 - **Apache Mesos** [Hindman *et al.*, 2011]
 - **Apache Aurora** [2014]
 - **Kubernetes** [2014]

Tension: Multiplexing resources & performance isolation

- Sharing machines across completely different jobs and tenants necessary for effective utilization
 - But leads to unpredictable performance blips
- Isolating while still sharing
 - Memory “ballooning” [Waldspurger, OSDI 2002]
 - Linux containers
 - ...
- Controlling tail latency very important [Dean & Barroso, 2013]
 - Especially in large fan-out systems

Higher-Level Computation Frameworks

Give programmer a **high-level abstraction** for computation

**Map computation automatically
onto a large cluster of machines**

MapReduce

[Dean & Ghemawat, OSDI 2004]

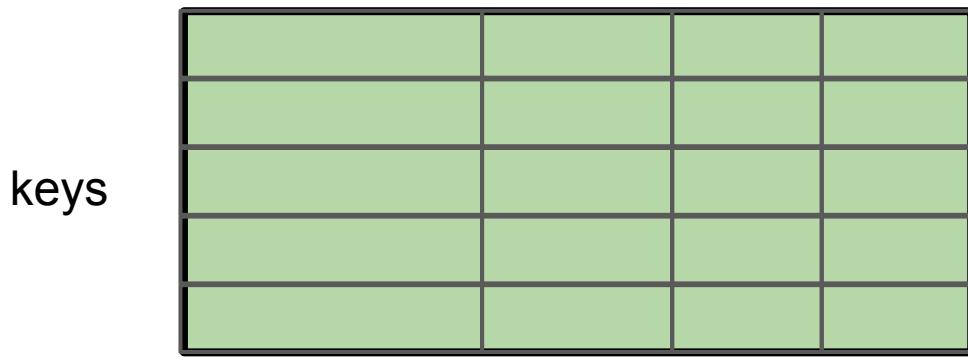
- simple Map and Reduce abstraction
- **hides messy details** of locality, scheduling, fault tolerance, dealing with slow machines, etc. in its implementation
- **makes it very easy** to do very wide variety of large-scale computations

Hadoop - open source version of MapReduce

Succession of Higher-Level Computation Systems

- **Dryad** [Isard *et al.*, 2007] - general dataflow graphs
- **Sawzall** [Pike *et al.* 2005], **PIG** [Olston *et al.* 2008],
DryadLinq [Yu *et al.* 2008], **Flume** [Chambers *et al.* 2010]
 - higher-level languages/systems using MapReduce/Hadoop/Dryad as underlying execution engine
- **Pregel** [Malewicz *et al.*, 2010] - graph computations
- **Spark** [Zaharia *et al.*, 2010] - in-memory working sets
- ...

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

The Public Cloud

Making these systems available to developers
everywhere

Cloud Service Providers

- Make computing resources available on demand
 - through a growing set of simple APIs
 - leverages economies of scale of large datacenters
 - ... for anyone with a credit card
 - ... at a large scale, if desired

Cloud Service Providers

Amazon: Queue API in 2004, EC2 launched in 2006

Google: AppEngine in 2005, other services starting in 2008

Microsoft: Azure launched in 2008.

Millions of customers using these services

Shift towards these services is accelerating

Comprehensiveness of APIs increasing over time

So where are we?

A diagram illustrating a distributed system architecture. At the top, a blue cloud contains the text: "Amazon Web Services, Google Cloud Platform, Microsoft Azure". Below it is a yellow starburst containing "BigTable, Dynamo, Spanner". To the left is a green starburst containing "MapReduce, Dryad, Pregel, ...". To the right is a red starburst containing "Powerful web services". A green horizontal bar labeled "Cluster Scheduling System" spans the middle. Below it is a blue horizontal bar labeled "Distributed file system". At the bottom are several server icons, each containing an "OS" label above a blue cylinder icon.

Amazon Web Services,
Google Cloud Platform,
Microsoft Azure

Powerful
web services

BigTable,
Dynamo,
Spanner

MapReduce,
Dryad, Pregel, ...

Cluster Scheduling System

Distributed file system



...



What's next?

- **Abstractions for interactive services** with 100s of subsystems
 - less configuration, much more automated operation, self-tuning, ...
- **Systems to handle greater heterogeneity**
 - e.g. automatically split computation between mobile device and datacenters

Thanks for listening!

Thanks to
Ken Birman, Eric Brewer, Peter Denning,
Sanjay Ghemawat, and Andrew Herbert for
comments on this presentation

Reminiscences on the 25th SOSP's History Day Workshop

Peter G. Neumann, Senior Principal Scientist,
SRI International Computer Science Laboratory

Close to 400 people joined us for the 25th SOSP History Day workshop. It was a spectacular event with something for everyone. The older generations had the opportunity to see how ideas they grew up with interacted with other lines of ideas that evolved into present systems. The middle generation has the opportunity to see where the foundations of their areas came from and to meet some of the people who built those foundations. The newest generation had the opportunity to see the rich heritage of ideas, principles, and concerns they have inherited from 50 years of prior work by thousands of researchers. We also scheduled a less formal evening session to allow anyone among the conference participants to reminisce about operating system. Here are a few thoughts on the entire event, including contributed text from some of the participants.

We owe enormous thanks to Peter Denning for creating this event and to Ken Birman for his close coordination with PJD from the very beginning. The organizing committee for History Day was phenomenally responsive during the planning stages. Not only did they help choose the topics and the speakers, but they also helped the speakers by reviewing draft slide sets. The committee included three authors who were at SOSP 1 in Gatlinburg (Denning, Jack Dennis, Butler Lampson) and two authors from SOSP 2 in Princeton (Jerry Saltzer and myself -- I was also at SOSP 1), with the rest from later years. Jeanna Matthews, past chair of SIGOPS, was emcee for the day; with her deft hand at time management, all the speakers kept to the allotted 25 minutes and allowed time for questions from the audience.

The History Day slides and paper abstracts are on the History Day website: <http://sigops.org/sosp/sosp15/history/> They are diverse, but comprehensive. In addition, copies of all past SOSP proceedings are accessible online: <http://sigops.org/sosp/sosp15/archive> Younger researchers especially are encouraged to look them up, together with the cited references, all of which represent a gold mine of treasures that are likely to contain nuggets useful for addressing today's problems.

Peter Denning spoke about the emergence of OS principle and showed several time lines of their evolution. He delved into memory management, just one timeline that unfolded the principles of location-independent addressing and locality. He was a leading researcher in those areas through the 1970s. Both areas were motivated by virtual memory -- which was a newly alluring but controversial technology at the time. The early concerns were whether the automation of mapping could be made transparent, and whether the automation of paging would perform well. A common principle across all implementations of virtual memory over the years is location independent addressing, and the methods of mapping addresses to the physical locations of objects. This principle was found in the original virtual memory, which had a contiguous address space made of pages, and is present in today's Internet, which provides a huge address space made of URLs, DOIs, and functionality. Performance was the other concern for early virtual

memory because the speed gap between a main memory access and a disk address was 10,000 or more; even a few page faults hurt performance. The locality principle emerged from years of study of paging algorithms, multiprogramming, and thrashing, where it was harnessed to measure working sets, avoid thrashing, and optimize system throughput. It is harnessed today in all levels of systems, including the many layers of cache built into chips and memory control systems, the platforms for powering cloud computing, and in the Internet itself to cache pages near their frequent users and avoid bottlenecks at popular servers.

Butler Lampson spoke of the evolution of principles and thought on protection and security, emphasizing how isolation is at the core of security but is sacrificed in the mania to share. For many years the principles that would lead to more secure systems took back seat to features that users found more convenient. He summarized the core principles of security with the Golden Rule of the three AUs: AUthentication, AUthorization, and AUditing. (I found Butler's speech and the entire event to be AUspicious, AUgmentational, AUthoritative, AUGuring a better future, and overall AUsome.) In the final session, Butler insisted that we all need to be concerned about the growing cybercrime problems that have been exacerbated by our computer systems and networks. We cannot solve these problems alone, but we can contribute to solutions. He also maintained that users almost always choose convenience over security, and are unwilling to acknowledge that user attitude is the elephant in the room.

Butler wondered how many people would want to buy a car that contains ten logic bombs in its embedded computers that could go off at any random time and cause an accident. That is possible now, and no one is doing anything to stop it. In the early years of the SOSP there was a lot of concern about protection and security. In 1975 Jerry Saltzer and Mike Schroeder distilled eight principles from all the discussion and research into a paper that became a classic -- still read today by students of security. They did not mention a 9th principle, which appears in the Saltzer-Kaashoek book, Principles of Computer System Design (2009) -- minimizing the part of the system that must be trusted. That principle was formulated during the design of Multics (around 1965) and was part of its ring protection structure. The principle of minimal trust has been one of my guides for over 50 years.

In the committee for the COSINE-8 report proposing a core course on operating systems principles (1971), Butler took the protection topic. He proposed to represent access rights as a function *Rights=f(principal,object)*. Other committee members thought this too abstract and suggested he display his function as a matrix. He acceded, and thus was born the access matrix model for representing the security policy of a system. Sometimes committees can be helpful in charting directions that shape history.

Barbara Liskov spoke about abstractions developed within the systems community to make it easier for implementers to organize system code; this work underlies the way systems are structured today. Two different approaches emerged from early work on how to structure systems that ran on a single computer. In one, described in Dijkstra's paper on the THE system, each individual resource (e.g., a printer) is controlled by a specific

"owner" process and code running in other processes use the resource by communicating with the owner using IPC (inter-process communication). In the other (described in Liskov's paper on the Venus system), resources are encapsulated inside modules and processes call operations provided by the module to use the resource; a popular way to control concurrency in this approach was provided by using monitors. Later work by Lauer and Needham showed that these two approaches are logically (but not necessarily operationally) the same, leaving each system developer free to choose the one that best fits the given architecture and application.

In the area of distributed computing, some form of IPC is needed and early work addressed the form this communication should take. Concerns included how to relate requests and replies, communicate arguments and results, allow servers to advertise their availability, and allow clients to select servers of interest. RPCs (remote procedure calls) that address these issues were first described in a paper by Birrell and Nelson. RPCs can be generalized beyond the call/return paradigm, e.g., to allow a client to make a sequence of calls that will be delivered in order, without the client having to wait for a reply from one call before making another. Generalized RPC between clients and servers is the approach used for all online applications today.

Edsger W. Dijkstra, our beloved colleague of the early years, was mentioned frequently throughout the day. His work on hierarchies of abstractions for operating systems, along with related work for programming by David Parnas, was hugely influential. His SOSP 1967 paper on the THE system inspired Barbara's early research on the use of abstraction levels to simplify systems and make them more resistant to errors. She noted that THE system was said to be free of deadlocks -- because function invocations were restricted to downward and upward calls in a process tree, which cannot allow a circular wait. Incidentally, Nico Habermann (Dijkstra's student and later colleague) told me that there had been unexpected deadlocks *within* a single layer, where cooperating processes could wind up in circular loops waiting for each other to send signals. Thus the design was "almost but not quite perfect."

Dijkstra himself added to the allure of the deadlock story at SOSP 1967. When it was his turn to present his paper, he said he would not speak about the THE system -- we could read that for ourselves -- but would instead speak of the problem of "deadly embrace", about which he was being asked in the hallways. This was a completely impromptu change of speaking plans. He sketched out a diagram on the chalkboard showing the state space of cooking stew and pudding in a kitchen with a single burner and single beater that both recipes needed. With his diagram he showed how certain (but not all) action sequences in the kitchen could lead to deadlock. To the surprise of many, there was an unsafe region of the state space such that if the joint progress path of the two recipes wandered into it, a deadlock at some future time was unavoidable. This diagram made such an impression that it wound up in many operating systems textbooks and inspired an entire line of research into deadlocks and preventing them. Two versions of it recreated subsequently are included here.

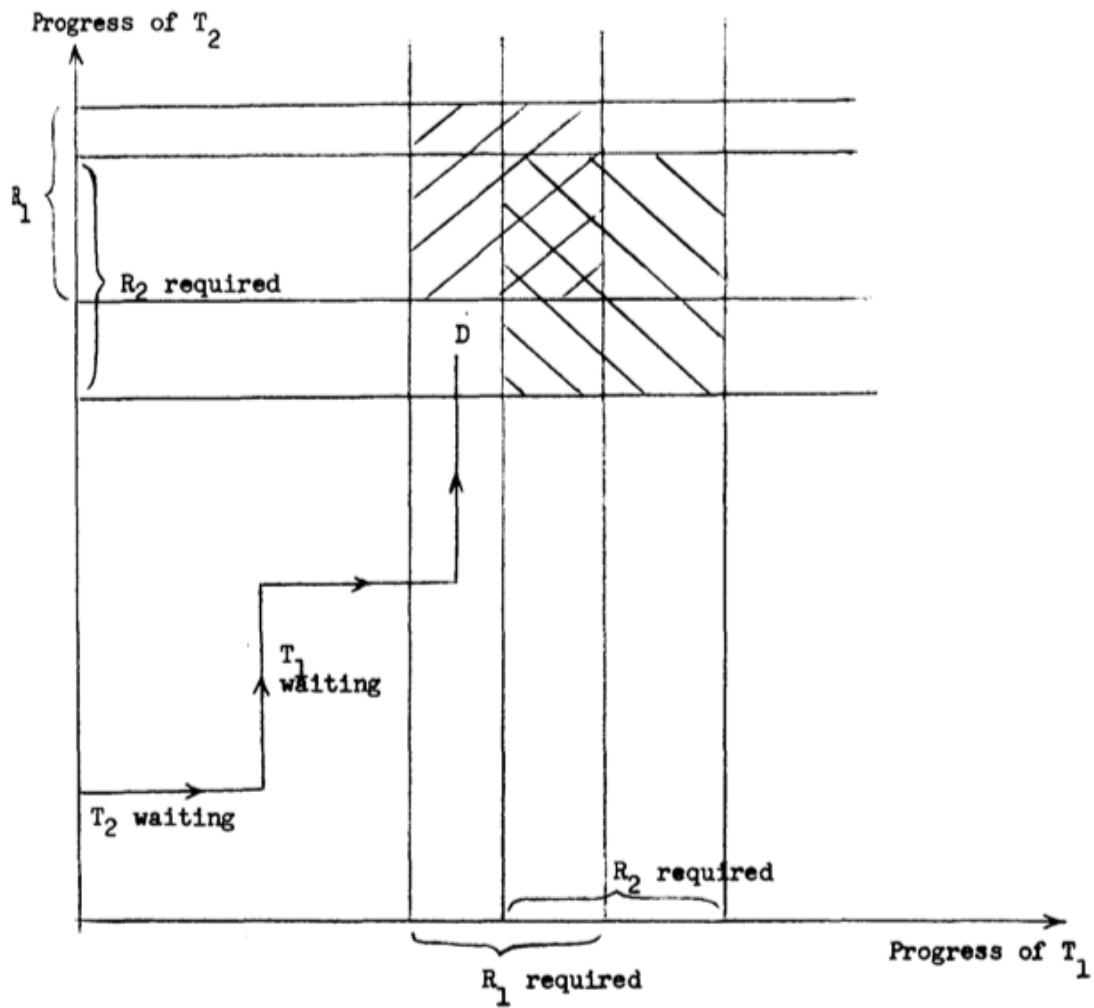
FIG. 2. Joint progress of tasks T_1 , T_2 .

Figure source: *System Deadlocks*, Coffman, Elphick, Shoshani, CSURV, June 1971.

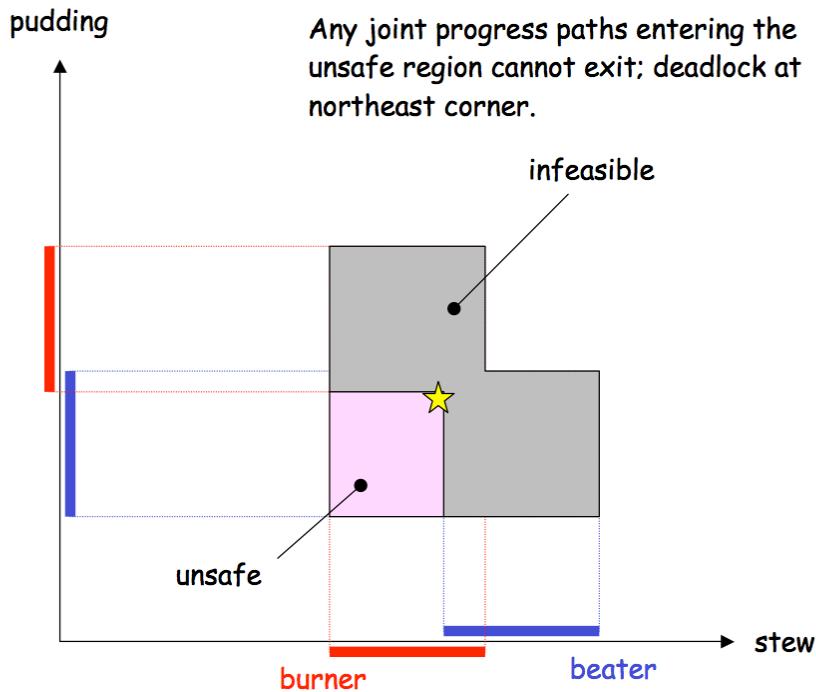


Figure source: Peter J. Denning, 2002, diagram for his students

I recall a WG2.3 meeting in Santa Cruz in the later 1970s when Edsger and I walked back through the redwoods after lunch. He was working for Burroughs at the time. I asked him what he was teaching Burroughs programmers about developing operating systems. (They wrote in a proprietary higher-level language, BALGOL; together with the PL/I subset and its Multics compiler -- developed by Doug McIlroy and Bob Morris -- they were the primary early adopters of higher-level languages for OS development.) Edsger's response was quite characteristic: "I don't do that. If I cannot write a program on the back of an envelope and prove it is correct, I have no interest in it." I do not know how many Burroughs programmers took his advice. I do know that his philosophy of building correctness into a system by design became very influential within the OS community.

In his comprehensive discussion of the evolution of memory and file systems, Mahadev Satyanarayanan (Satya) focused on the quests for scale, performance, transparency, and robustness at differing points in history, as well as gaining almost 13 orders of magnitude reduction in the cost per bit. As Bob Daley's co-designer of the Multics hierarchical file system with ACLs to help manage access (Fall Joint Computer Conference, 1965), I was particularly intrigued by Satya's comments refuting predictions that such file systems would soon be extinct! (Satya pointed out the origin of file-system hierarchicalization stems from Herb Simon in 1962.)

Ken Birman discussed the history of fault-tolerance and the CATOCS/CAP controversies (CATOCS = Causally and Totally Ordered Communication Support. This topic is rather complicated, and deserves considerably more detail from Ken than could include in his slides. However, his talk was a poignant reminder that we really need trustworthy systems that can help enforce a wide range of total-system trustworthiness properties.

Andrew Herbert spoke about the origins and evolution of virtualization systems. From beginnings as a testbed for exploring segmentation and paging algorithms (e.g., in Multics), early IBM work led to the successful and long-lasting mainframe operating system VM/370. In addition to timesharing, VM/370 had two other important capabilities: it allowed multiple IBM operating systems to be run on the same machine and it enabled operating systems to be developed using the same tools and environment as for applications. The virtual machine concept remained in the background of OS research until it reappeared in the early 21st century, first as a means to enable different desktop operating systems to coexist on the same personal computer and then as a means to enable server consolidation. Today, virtual machines are a key component of cloud computing and the means by which server workloads are managed. Virtual machine research reported at SOSP has included hybrid virtualization, paravirtualization, virtual machine monitor resource management, virtualization for fault tolerance and for misuse detection and monitoring. Andrew also looked at the relationship between virtualization and abstraction (going back to Dijkstra's THE operating system) and the development of the concept of uniform location-independent names for virtualized resources.

Dave Clark spoke about the early days of networking. In the 1970s, the early implementations of TCP/IP were dreadfully slow. Dave and his colleagues spent years understanding in detail all the sources of inefficiency and one by one worked them out. When they were done TCP/IP was tightly integrated into the operating system and was very efficient. Without that work, the Internet would not have taken off. This is one of the lessons of history. It often takes a lot of engineering work to transform a great idea (such as TCP/IP) into a working system with acceptable performance. We tend to remember the people who discovered the idea but not the many engineers who figured out how to make it work.

Dave Patterson spoke about the evolution of computer hardware and how gross differences between instruction sets of machines was a major impediment to interoperability. The first attempts to standardize machines around instruction sets were with the IBM 360 and evolved into machines with very complex instruction sets (CISC). Beginning with John Cocke, researchers found that simplifying instructions around an execution pipeline significantly improved performance and gave birth to the RISC revolution in the early 1980s. Some of the great ideas of earlier eras, such as capability addressing, were lost in the RISC revolution. Today there is an effort to produce open instruction sets that all chip makers could use, and some researchers are aiming to include capability addressing in that architecture. I think many participants came to a new appreciation of hardware they did not have before.

Frans Kaashoek's talk divided research on parallelism in operating systems in 4 periods. He summarized foundational ideas for parallel programming, covering three types of parallelism in operating systems: user-generated parallelism, I/O parallelism, and processor parallelism. With the arrival of commodity small-scale multiprocessors, the OS community "rediscovered" the importance of processor parallelism and contributed techniques to scale operating systems to large number of processors. These techniques found their way in today's mainstream operating systems largely because of the emerging ubiquity of multicore processors. Frans observed a "rebirth" of research in parallel computing as a result. Given the advent of cloud computing and the presence of frequently compromised operating systems, Frans's talk was especially timely as a cry for future research.

Jeff Dean described the development of systems that are going to be essential for cloud computing systems. These systems are going to have to address trustworthiness much more systemically, incorporating security, reliability, distributed transactional consistency, high-performance computing, and lots more. My own view of cloud computing is that has the potential to combine the worst aspects of outsourcing to untrustworthy or unaccountable third parties, and will introduce enormous risks unless major improvements in trustworthy operating systems and networking are achieved.
Caveat Emptor!

The day ended with a panel looking at the future of system support for security, with Jeanna Matthews as the moderator, and David Mazieres, Mark Miller, Margo Seltzer and YY Zhou. The panel raised many diverse points, and was evidently engaged the audience.

Aside from Clark and Patterson, the other speakers focused mostly on software. They all called for constant attention to the principle that operating systems be designed holistically, with system architectures coordinating hardware, software, and even programming languages around user practices and concerns. Peter Denning mentioned this in his overview of evolution of the purposes of operating systems -- the current-generation OSs present an immersive environment in which users manage their life, work, and social networks. However, total-system properties such as human safety and system survivability in the face of accidental, intentional, and environmental disturbances were not mentioned, yet must also be systemically addressed. The same is true for hardware that can more effectively enforce what is needed for the operation systems and applications.

Bob Morris's 19 September 1988 quote before the National Research Council Computer Science and Telecommunications Board is even more relevant today, with the potentials for crime and abuse in the extended Internet. (Note that the 'T' in CSTB later became 'Technology'.)

"To a first approximation, every computer in the world is connected with every other computer."

Capability-based hardware and operating systems seem to be having a renaissance. In 1971, Butler Lampson discussed the intuitive notion of an access-control matrix, with a row for each subject and a column for each object. In that model, each column represents an access control list for an object (as introduced in Multics, and noted above), and each row represents capabilities for accesses permitted to that subject. In his talk, Lampson noted that short-term capabilities (access tickets with an expiry of a few minutes) are much more useful than long-term capabilities (access tickets that do not expire, causing revocation and auditing problems). History is replete with many attempts to design and implement various types of capability-based systems -- including include Lampson's Berkeley system, Plessey 250, PSOS, Hydra, KeyKos, EROS, the E-system, Cambridge's CAP, and most recently NICTA's seL4 and the SRI-University of Cambridge CHERI (Capability Hardware Enhanced RISC Instructions) system, with MIPS-64-based hardware running FreeBSD. Attendees involved in capability-based systems also included Andrew Herbert, Alan Karp, and Mark Miller, and myself (in addition to Robert Watson on the committee, who was unable to attend). This is a diverse community, but generally quite optimistic about the potential role of capabilities in the future.

In the evening event, Ken Birman took a moment to remind us of Jim Gray (1944-2008), who received the ACM Turing Award in 1998 and was a passionate participant in the ACM systems community. Although his roots were primarily in the transactional and database systems communities, Jim nonetheless made a point of attending SOSP and often spoke about fault-tolerance scalability of consistency mechanisms. Jim's unique mix of technical wisdom and personal warmth, and his style of outreach and engagement with the new generation of researchers, is truly inspirational.

Many other individuals have contributed significantly to our history and the current state of the art. For example, other authors from those first two SOSPs included Brian Randell, Earl Van Horn, and Bob Daley from 1967, and Roger Needham, Brian Kernighan, Bill Wulf, Elliott Organick, and George Mealy from 1969 -- to name just a few. Ken Thompson and Dennis Ritchie were authors in 1971. Also, work by many other people was mentioned. For example, Andrew Herbert noted the importance of the work of Roger Needham, Peter Denning noted the early work on memory management and locality by Les Belady at IBM, Ken Birman noted the recursive recovery-block work led by Brian Randell, and several people mentioned the long-time contributions of Les Lamport on distributed algorithms. Everyone named here is an important part of our history, as are many other authors not explicitly mentioned.

All in all, the 25th SOSP History Day was remarkably valuable for everyone, old and young. Clearly, it would be a major task to do justice to everyone who has contributed to this history, although that is way beyond the intended scope of my brief remarks. However, I hope that the SOSP History Day can inspire a comprehensive project, perhaps beginning with a collection of oral histories akin to what has been done in the computer security community -- where over 300 people were interviewed by the Babbage Institute of the University of Minnesota, under the aegis of NSF funding. I also hope that some of the attendees of the 25th SOSP History Day will be around to organize a 50th SOSP History Day in 2065.

PGN, 23 October 2015