Daley, R.C., and Dennis, J.B.
Virtual Memory, Processes, and Sharing in MULTICS
Communications of the ACM, Vol. 11, No. 5, May 1968, pp. 306-312.

① Effectively serve the computing needs of a large community of users with diverse interests, operating from remote terminals mainly. √√ Awesome!, √√

+ large virtual memory √
+ symbolic names for procedures
+ sharing

② √√ Address Space

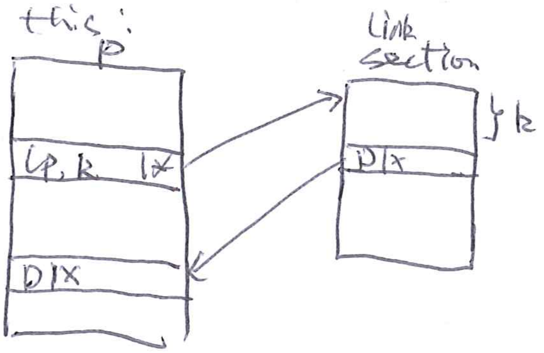
- Segment #, Word # "Generalized Address"
- Segments organized in directory structure (need not be different than files)

Dynamic Linking

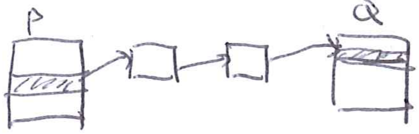
- Using just symbolic segment name / word # in code segments, at runtime these symbols are resolved lazily and physical segment/word kept in another segment for the process called "linkage section"

③ How does linkage section work?

~~Sike~~ Since the procedure segment is pure (readonly), we ~~cannot~~ cannot put ~~its ad~~ ~~CP~~ Linkage section address in procedure segment, so we have a special register lp (link pointer) to point the linkage section like this:



Things get more interesting when another linkage section ~~is~~ involved

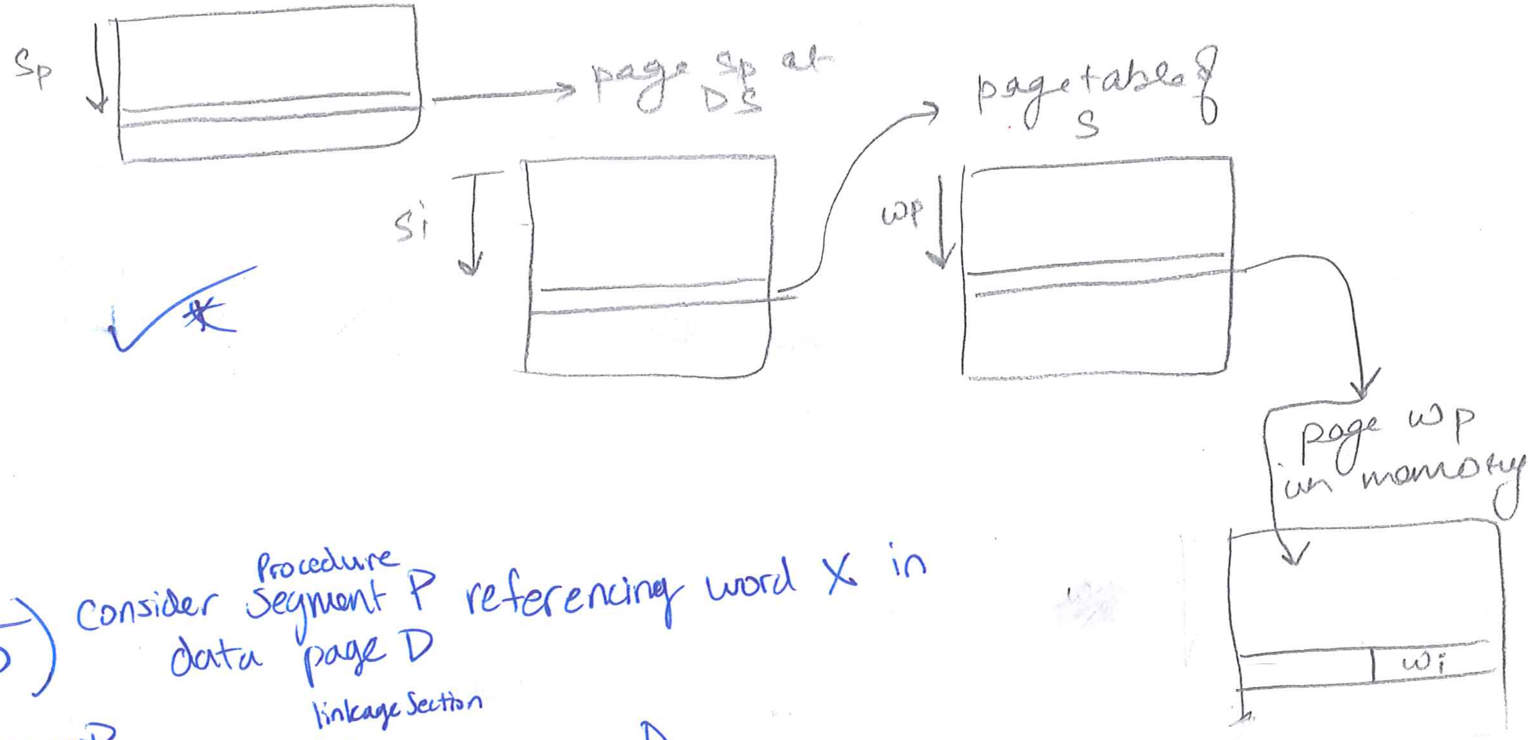


Paging Implementation :

1024 words/page ⇒ 2^{10} word = 10 bit address

Sp	Si	Wp	Wi
4	10	8	10

PBR at page table of DS



5) Consider Procedure Segment P referencing word X in data page D

⑥ PIO: high-level language implementation

on-line reconfiguration, large virtual memory with segments, paging and generalized addresses

First hierarchical file system, Dynamic linking and function call by name

Shared memory multiprocessor

Security and rings 24/7 Computing service.

Daley, R.C., and Dennis, J.B.
Virtual Memory, Processes, and Sharing in MULTICS
Communications of the ACM, Vol. 11, No. 5, May 1968, pp. 306-312.

√ √ - Multics is an OS built to serve a large group of users working
☆ from remote machines (Multiplexed Information and Computing Service)

↳ Goals

1. provide each user with a large machine independent virtual memory space (avoid worrying about memory management)
2. Allow users to invoke procedures while only knowing symbolic names (programming generality)
3. Allow/permit data sharing among users (more effienct)

2.) Major contribution was the idea of dynamic linking using symbolic names for shared libraries. It allows a single memory copy of the library to be used ☆ across multiple programs, It also allows the library to be updated ☆ without recompiling the user's program since the linking is done at runtime.

3) Technique:
Process stand in one-one correspondence with address space/virtual memories

Generalized address

segment #	word #
14bit	18 bit

Each word is 36 bits.

This is a location independent way to address information
Segments are associated with symbolic name searched in a directory structure.
Has Procedure and Data segment.

Instruction format in

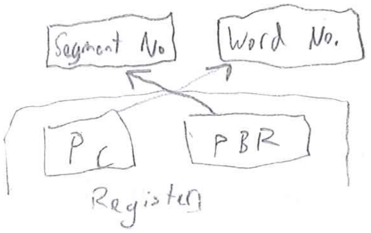
segment tag	address	op code	ext flag	add mode.

'its' addressing mode used to access other segment
Descriptor segment stores information about segments access by a process, the location of which is in DBR

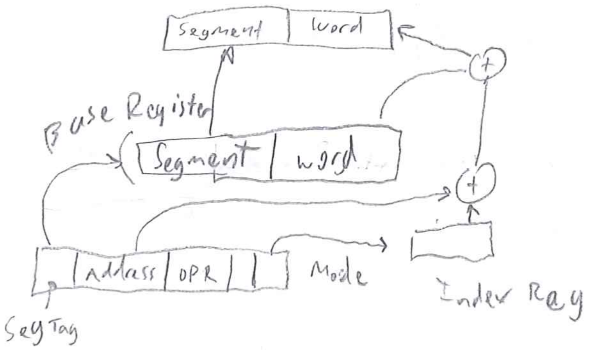
Linkage section stores just data for linking segments to a procedure.
Symbol table stores associations between sym word names & word # of a segment.

④ Generalized Address Formation

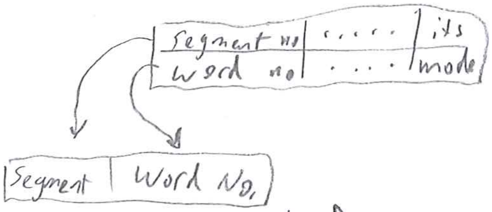
Instruction Fetch



Data Access

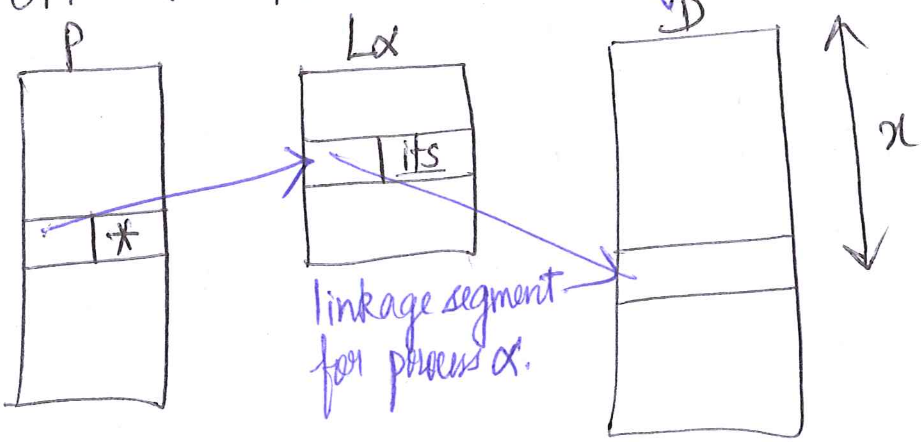


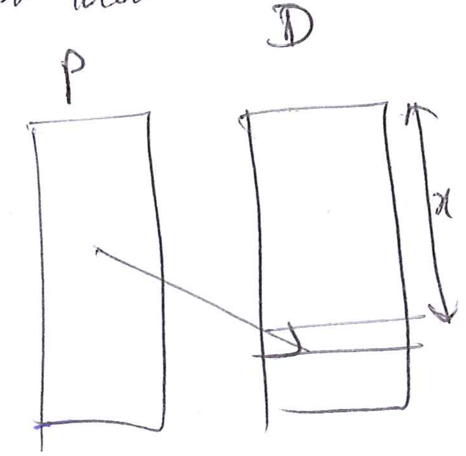
Indirect Addressing — Same method to make generalized address as data access. Then use g.a. to retrieve two 36-bit words which make new generalized address



⑤ Procedure segment P - makes reference to a word at location x in data segment D.

OPR <D> | [x]. — assembly language.





One linkage segment per process.

⑥ Pros — + Dealt with all possible virtual memory concepts earlier than most systems. (1960)
+ Had a huge impact on UNIX VM design
+ Dynamic linking → standard in today's system

Cons — Too complex for old processors which were slow, multiple lookups.

2

Superpages.

Juan Navarro, Sitaram Iyer, Peter Druschel, and Alan Cox
Practical, transparent operating system support for superpages
Proceedings of the 5th Symposium on Operating Systems Design and Implementation
Boston, Massachusetts, USA December 9–11, 2002

✓

① TLB misses are expensive. Over time the ratio of TLB coverage to main memory size has decreased. Increasing the size of pages increases coverage, but results in enlarged application footprints, increased memory requirements and ✓ paging traffic. So it's not a good idea to use large pages if the memory is not going to be used. Allowing multiple page sizes would help. ✓

② They propose a superpage management system which balances the competing needs for allocation, fragmentation control, promotion, demotion, and eviction.

Allocation – they use a reservation-based scheme (rather than relocation-based)

✓ Frag Control – their system prefers preempting reservations over refusing allocations

promotion – they wait until regions are fully populated

demotions – Use speculative demotion ✓

eviction – they demote clean superpages before writing

③ Specific Techniques:-

(i) Reservation list :- Each reservation is kept in a reservation list which tell us the maximum superpage size which is still not allocated in that reservation.

↳ used for allocation by buddy allocator

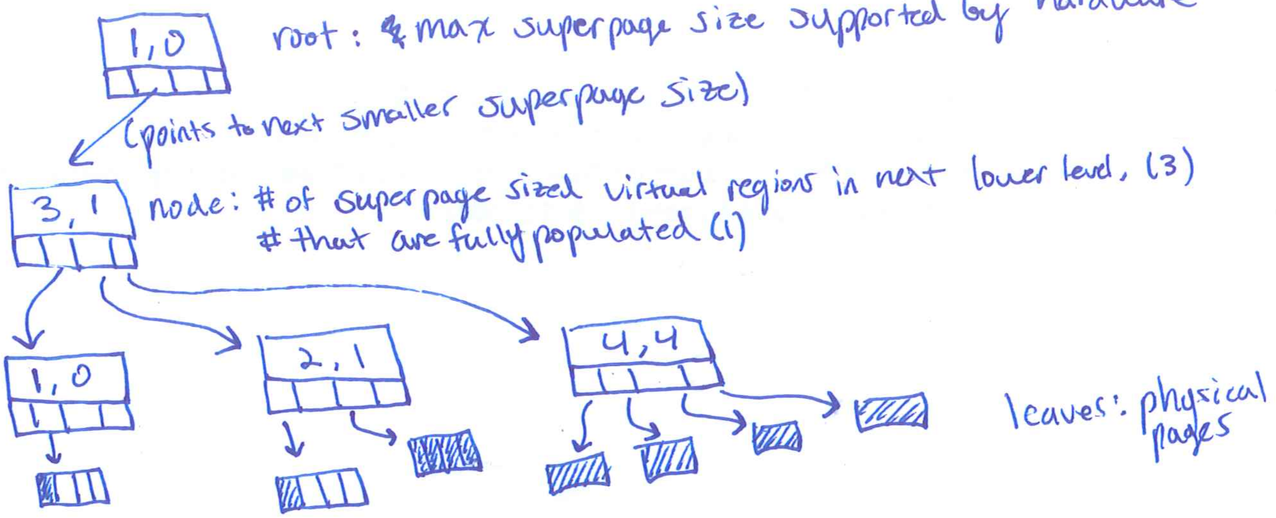
✓ (ii) Population Map:- It is associated with each memory object to keep track of allocates & contiinous free space.

↳ used for promotion, overlap detection,

(iii) Cache, Inactive & Active list:- use by Page replacement daemon to evict pages upon memory pressure in a contiguity aware manner.

(iv) Wired Clustering :- Try to cluster Kernel (System) pages in a contigous manner to avoid memory fragmentation.

4. Population Map



root: ~~&~~ max superpage size supported by hardware
(points to next smaller superpage size)

node: # of superpage sized virtual regions in next lower level, (3)
that are fully populated (1)

leaves: physical pages

5. Behavior over time

→ Clean state develops fragmentation within 15 minutes. All contiguous memory regions > 64KB were used up.

(Figure 5)

→ Two schemes to get contiguity

Cache
(recovers ~ 9%)
- cached pages are coalesced!

Daemon
(recovers ~ 43%)
- page replacement is done in a way such that it is contagious ?
- page clustering is wired
- activated when contiguity low

6. pros - less TLB misses, more TLB reachability.

cons - contiguity management not great

Carl A. Waldspurger
Memory Resource Management in VMware ESX Server
In Proc. Fifth Symposium on Operating Systems Design and Implementation (OSDI '02), Dec. 2002

problem/motivation

Target: No modification to the OS kernel ☆, but make the memory ~~manage ment~~ management better for VMM.

✓ ① influence the guest OS mem allocation policy from bottom up. OS thinks it has ~~sm~~ X MB of physical mem, but it doesn't.

✓ ② share memory across VMs, so we have more free machine mem for other uses.

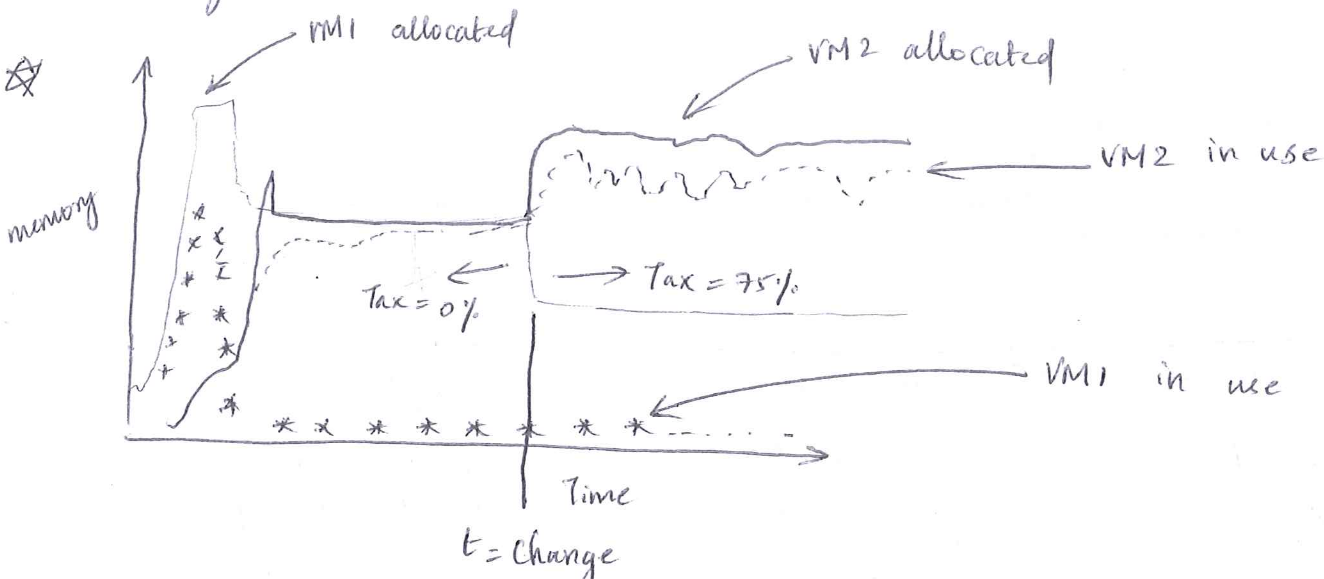
(2.) ✓ Ballooning: install a kernel module in guest that claims memory when VMM decides to reduce guest's allocation. This allows guest OS to use its knowledge of its memory use to make good decisions about swapping, paging, etc., avoiding duplicating efforts in VMM.

③ ESX implemented transparent page sharing between VMs w/o modifying the guest OS. It randomly samples pages and computes a hash on the page content. If the hash collides in the hash table, then the contents are exhaustively compared before eliminating the redundancy. Redundant pages maintain reference counts and use COW technique. *

if no match, keeps the hash as a hint; if the hash has a match later, have to recompute hash to ensure it didn't change. ✓

④ Idle memory tax

VM1 and VM2 started with same shares

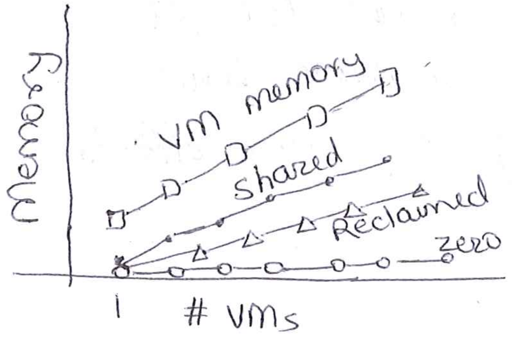


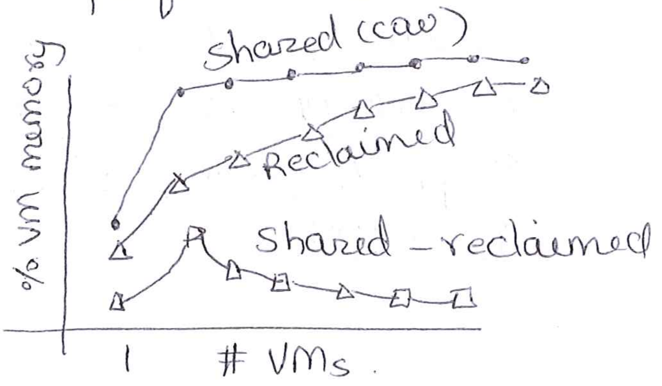
At time t = change, tax was was set to 0.75 so idle memory will be taxed and so shares per page will be changed.

Pure fair share alloc ~~ba~~ when t < change

VM2 gets more memory when t ≥ change

⑤ Effect of page sharing on performance.





Sharing with single VM ?

- Copies of zero pages.

6. Pros - Transparent, no changes on guest OS.
 - Well defined problem and solutions..
 - Idle memory tax configurable at run-time.

cons - Ballooning subject to guest OS policies. Can fail.
 - Page sharing may not always yield benefits. ~~Any sharing data~~ ~~to~~ Sharing needs to be re-established after reboot.
 -

Carl A. Waldspurger
Memory Resource Management in VMware ESX Server
In Proc. Fifth Symposium on Operating Systems Design and Implementation (OSDI '02), Dec. 2002

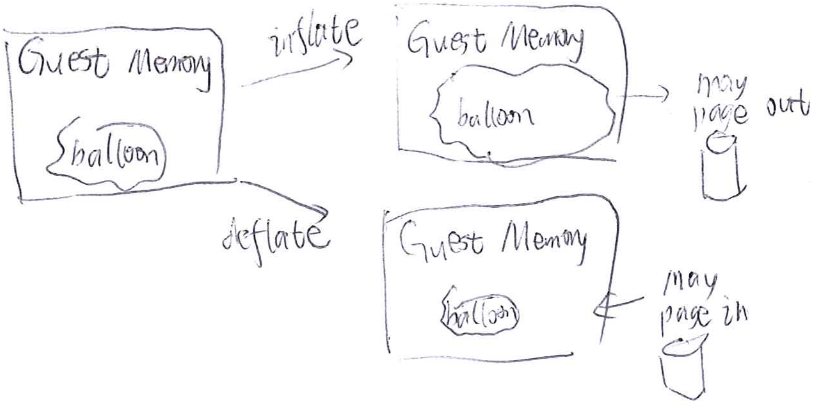
1) Individual servers are typically under-utilized so it is preferred to group them together as VM's to maximize hardware usage. VMware's goal is to provide the most efficient multiplexing of resources between VMs by using ballooning, sharing, and memory taxing. ☆★ ✓

2) As mentioned above, VMWare used methods to help manage resources across VMs without significant change to the VMs being run like inserting a pseudo-driver to 'balloon' and gather memory that VMware could page out, sharing identical memory across VMs, and ensuring that no VM proportionally used too many resources. By doing so, they greatly improved the efficiency of running multiple VMs of commodity OSes. ✓

③ They support memory ~~[illegible]~~ sharing by doing content-based identical page detection, and do it without having to interpose on bcopy or IO accesses like disco. They do a periodic scan of memory to look for matching pages.

```
For each page: hash it, look in hash table
        if match, do a full check
            if still a match, update pmap to share COW-style
        if no match
            ~~[illegible]~~ store hash as hint, and ~~[illegible]~~ do not mark COW
```

④



⑤ Fig. 7 (didn't draw here) Studies the effect of imposing tax on idle memory. The initial tax rate is 0, resulting in a pure Share-based allocation. Then the tax rate is increased to 0.75, causing memory to be reclaimed from one idle system & reallocate to another active system.

⑥ VMWARE ESX is similar to Disco but does not require modification except installing balloon driver.
It is much easier to install, does not require kernel changes.
This affects how page sharing is done.
In Disco pages from the same source are shared, while in VMware, pages are shared based on their contents. Because it is time consuming to check all pages, this is done randomly or on page out. In VMware, pages from the same source wouldn't be identified and automatically shared like Disco. However VMWARE can share more pages overall once they are identified.

5

Butler W. Lampson, David D. Redell
Experiences with Processes and Monitors in Mesa
Communications of the ACM, 23 2, February 1980, pp. 105-117.

①

Monitors and condition variables, implementation of synchronization primitives in Mesa not addressed by C.A.R Hoare

the semantics of nested monitor calls; the various ways of defining the meaning of WAIT; priority scheduling; handling of timeouts, aborts and other exceptional conditions; interactions with process creation and destruction. monitoring large numbers of small objects.

✓✓

A Notify is regarded as a hint to a waiting process ☆

Whenever a process enters a monitor, its priority is temporarily increased to the monitor's priority

② In monitor ~~pag~~paper, a waiting process has ~~to~~ resumed ~~so~~ immediately after being signaled. Mesa allows the ~~so~~ waiting process to ~~b~~ resume at ~~ag~~ any ~~the~~ time that is convenient. This allows us to implement priority scheduling, time out, abort...

✓

The idea is implemented as condition variable ✓. like condvar.wait(k), condvar.notify(). ~~The~~ The waiting process with the lowest k ~~will~~ will resume first.

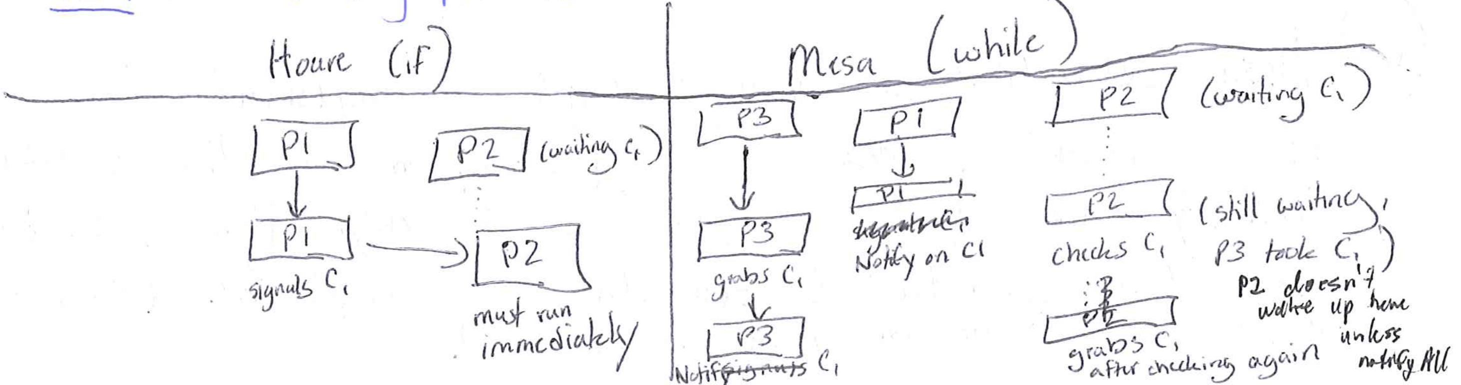
③ - Also talks about scheduling. in some apps it's good to use a priority scheduling discipline for allocating processors to processes.

✓

- More detail abt c.v. when one proc. establish a condition, it notifies ~~the~~ corresponding c.v. A NOTIFY is regarded as a hint to a waiting process.

④

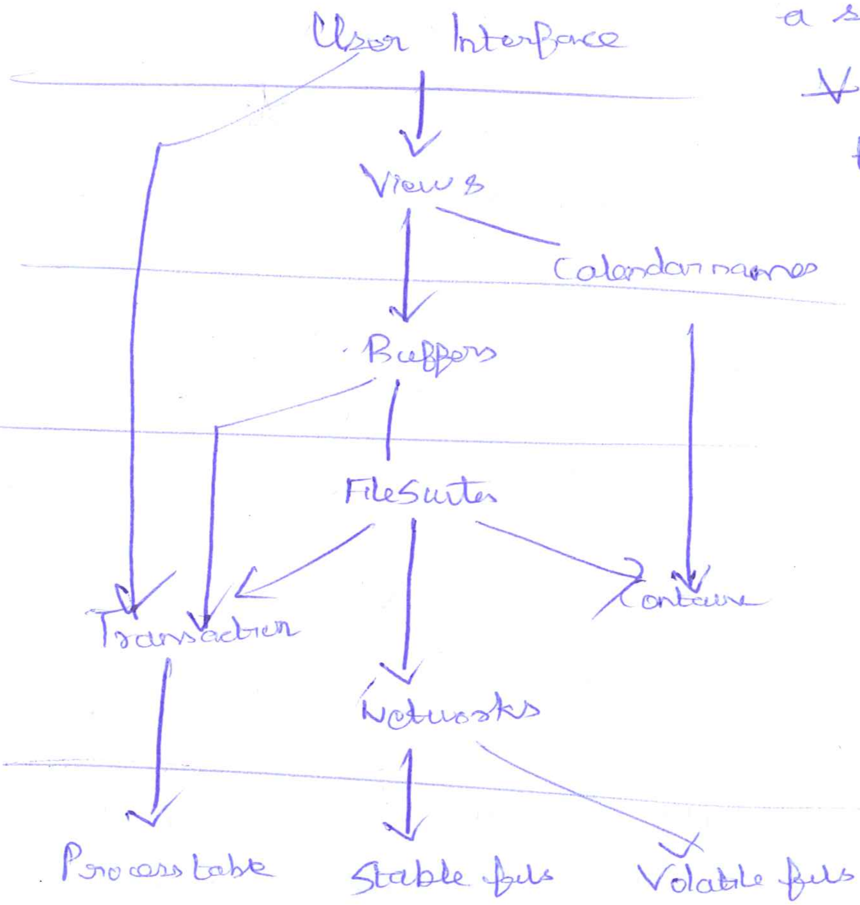
✓



5) Eg Violet A distributed Calendar system
Supports replicated data files and provides a display interface to the calendar system.

UI :
Display - keep display consistent with views from user & dB
Keyboard : user change
Datachanges : DB

FileSuites : Construct a single replicated file from a set of representatives.
R+W > N
∀ File suite, a monitor tracks the representatives & version number.
File suite created, FORKs detaches enquiry
Read quorum ✓
if not the wait on Crowd Larger
For Writes, a write quorum is required
If not, fork UPDATE process to write to all repres



If read quorum is present, but not write quorum, invoke COPY process to copy to a new repres.

6. Mesa semantics are much simpler to implement than Hoare semantics. Mesa allows pre-emption and other scheduler flexibility since signals are regarded as hints only. However, reasoning about Mesa semantics is somewhat more complicated. (Signalling a process doesn't guarantee it runs immediately).

Anderson, T., Bershad, B., Lazowska, E., and Levy, H.
Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism
ACM Transactions on Computer Systems, Vol. 10, No. 1, February 1992, pp. 53-79.

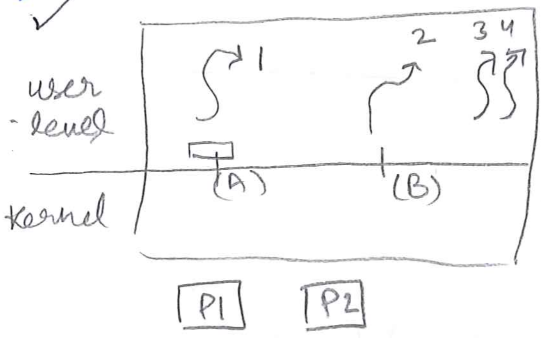
✓✓✓ 1. Threading and parallelism was a tricky topic due to the fact threads were divided into user-level threads (lower overhead, more manageable, but less knowledge of kernel activity such as I/O) and kernel-level threads (had detailed knowledge of resources, but much higher overhead). Bershad et. al. wanted to create a solution that had the performance of user-level threads but could use knowledge of resources (what CPUs were idle, etc.) in cases where the kernel was involved (blocking on I/O, etc.)

✓ 2. The big idea is to use the advantages of kernel + user threads by providing scheduler activations to address space.

Kernel - provide virtual multiprocessors to users, let them run what they want

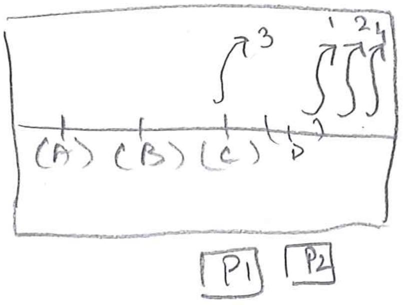
User - run on scheduler activations, and ask for more/less as needed.

☆✓ 3. I/O Request Completion:

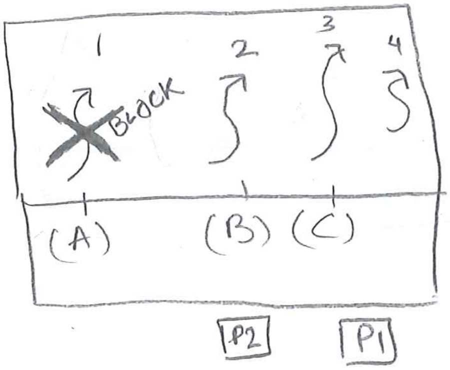


Amazing ☆! Wow!

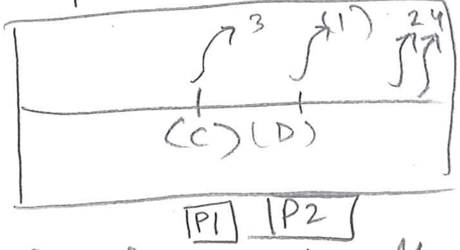
- application gets 2 processors P1, P2
- P1 upcalls (A) which runs thread 1
- P2 " (B) " " " 2



- I/O now complete. Kernel needs to inform user-level about this.
- Kernel preempts P2 and sends upcall.

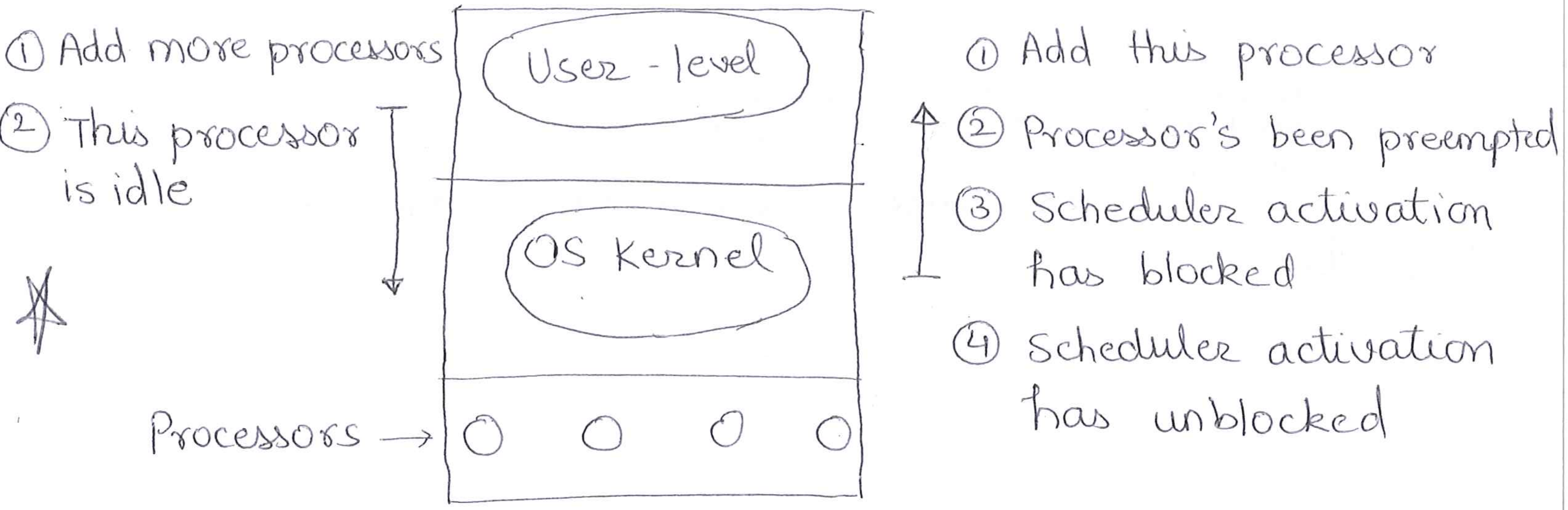


- Thread 1 blocks in kernel
- Kernel needs to tell this to user-level
- Kernel takes P1 and upcalls via C.



- Previous upcall takes thread 1 and runs it.

④ Scheduler Activation Upcall Points.



✓

5) It is possible for a user level thread to be preempted while in a critical section and holding the lock.

They use a recovery-based solution. When the thread is notified of being preempted, system checks if it is in critical section. If so, thread is continued temporarily via user-level context switch. When thread exits critical section it relinquishes control and is added to ready list.

⑥ Pros

\+ Combines the performance of user-level threads with the functionality of kernel threads

Cons

?

8

Waldspurger, C.A. and Weihl, W.E.
Lottery Scheduling: Flexible Proportional-Share Resource Mangement
Proceedings of the First Symposium on Operating Systems Design and Implementation, Monterey CA, November 1994, pp. 1-11.

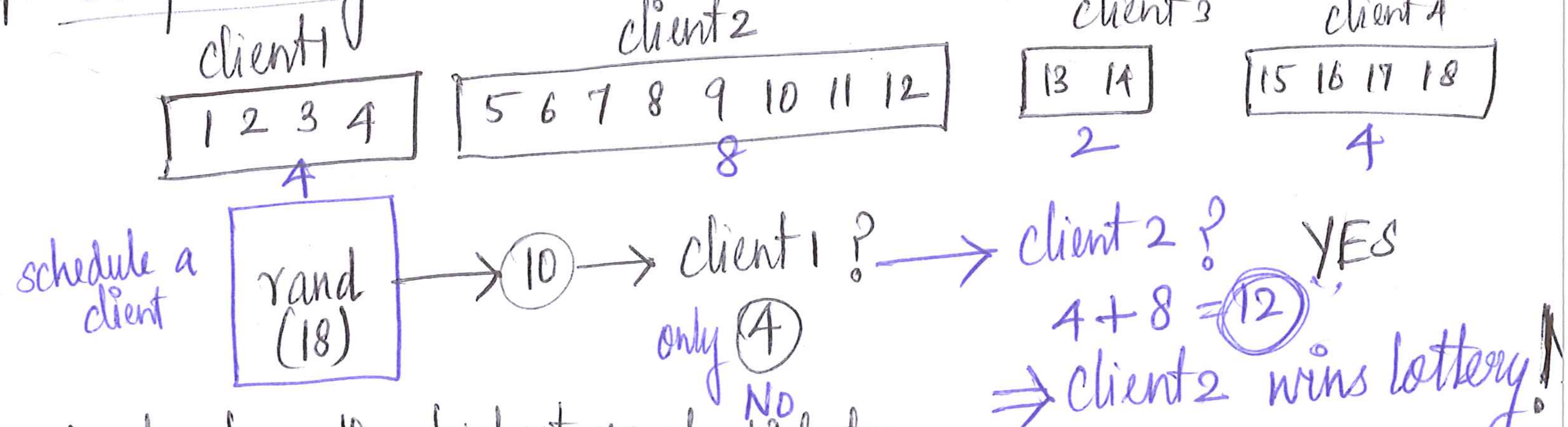
1.) Alternative to complicated hierarchy, which is hard to control and complex to reason about in addition to scheduling significant overhead (at least lottery scheduling is considered efficient). Also wanted a model that provided more fine-grained control over resource allocation in a modular way.

Proportional - share resource management ✓ ✓

2) MAJOR RESULT :
→ Introduced tickets - abstract, relative, uniform
→ If more tickets, then you are the lottery winner ??
→ Lottery winner gets resources in proportion to tickets held.
→ Use MLFQ. Does probabilistic fair distribution - ✓

3) Compensation Tickets - a way to try to make up for unused time.
☆ if a client consumes fraction f of it's allocated quanta, it's ticket value inflates by 1/f until it runs again.
The problem here is that a client cannot win lotteries they are not part of. If they are asleep for a long time, winning once more will not make up for that lost time.

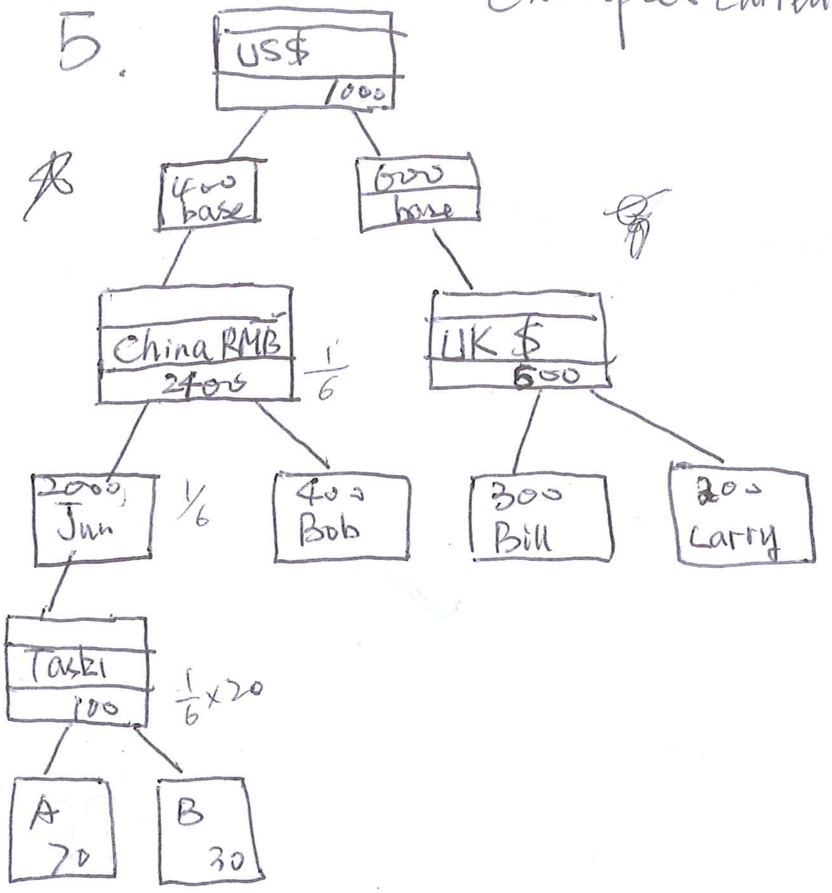
4. Sample lottery. Total tickets = 18



* client 2 has the highest no. of tickets.

Example: currency.

5.



What's B's Base currency?

$30 \times \frac{1}{6} \times 20 = 100$?

[illegible]

~~2000~~

$1000 \cdot \frac{3}{5} \cdot \frac{5}{6} \cdot \frac{2}{5} =$

Larry's currency in base

6. pros: - conceptually simple & easy to implement.
- can be generalized to manage diverse resourses & add to existing OS to improve control over resourse management.

cons: - overheads are for sure, but not too much comparing to Mach.

Waldspurger, C.A. and Weihl, W.E.
Lottery Scheduling: Flexible Proportional-Share Resource Mangement
Proceedings of the First Symposium on Operating Systems Design and Implementation, Monterey CA, November 1994, pp. 1-11.

i) Motivation

This work aims at developing a new resource allocation mechanism where the execution rates of various computations are proportional to the relative shares allocated to the computations. A currency abstraction is developed to define and allocate these shares and the paper shows that this abstraction can be used to manage diverse resources.

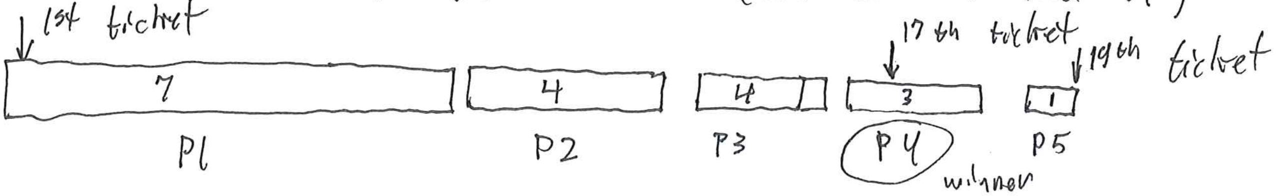
② →

③ Technique:

- clients are allotted lottery tickets.
- Resource allocation is done by holding a lottery.
- Provision for ticket inflation (useful among mutually trusting clients), ticket transfers (to ensure progress in case one client blocks on something), compensation tickets (to ensure fairness in case of I/O etc).
- Ticket currencies. (to convert tickets to base units)

④ Lotteries are implemented by picking (?. a number u.a.r. between 1 and total # of tickets held by runnable processes. Processes are arranged in decreasing order of tickets held to minimize linear search time:

random # is, e.g., 17 (between 1 and 19)



5. Consider a system that starts with two tasks, A and B. Out of 1000 Basecurr A has 400 base currency, B has 600. A wants to spin-off four subtasks, $A_1 - A_4$, and use its own currency to manage their priority - call it Acurr. Out of 100 , Acurr, A_1 has 10, A_2 and A_3 each have 20, and A_4 has 50. Then the overall system divides up scheduling as follows:

- B gets to run 60% of the time (600 base currency)
- A_1 gets to run 4% of the time $(\frac{10\ \text{Acurr}}{100\ \text{Acurr}} * 400$ base Curr = 40 base curr)
- A_2 gets to run 8% of the time $(\frac{20\ \text{Acurr}}{100\ \text{Acurr}} * 400$ base curr = 80 base curr)
- A_3 gets to run 8% of the time "
- A_4 gets to run 20% of the time $(\frac{50\ \text{Acurr}}{100\ \text{Acurr}} * 400$ base curr = 200 base curr)

✓

6.) Pros:
- Starvation no longer exists in traditional sense
- Cute
- Allows for different types of tickets to control proportion of resource share at different levels of the process hierarchy

Cons:
- Non-deterministic (by its very nature)
- I/O not handled as well.
  b/c real-time, short tasks not served well
- Fair over-time but some processes need instant attention

Banga, G., Druschel, P,. Mogul, J.
Resource Containers: A New Facility for Resource Management in Server Systems
Proceedings of the Third Symposium on Operating System Design and Implementation
(OSDI-III), New Orleans, LA, February, 1999, 45-58.

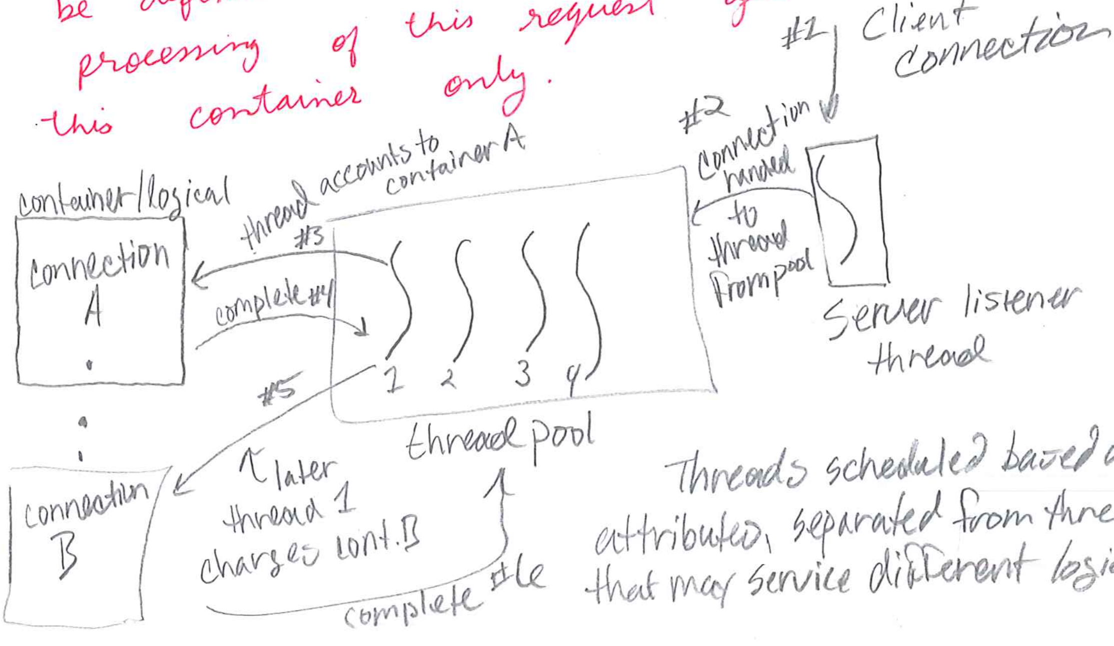
1) modern computing relies heavily on servers, which don't manage their resources well. They conflate Protection Domains
* ✓ and resource principals, and do not appropriate time spent in the kernel as resource usage. This leads to the problem of improper resource accounting, degrading performance.

② ⓐ Allow explicit and fine-grained control over resource consumption at all levels (user, kernel) in the system. ✓

ⓑ Resource containers allow to perform resource allocation on a per-activity basis, instead of OS-process basis.
?

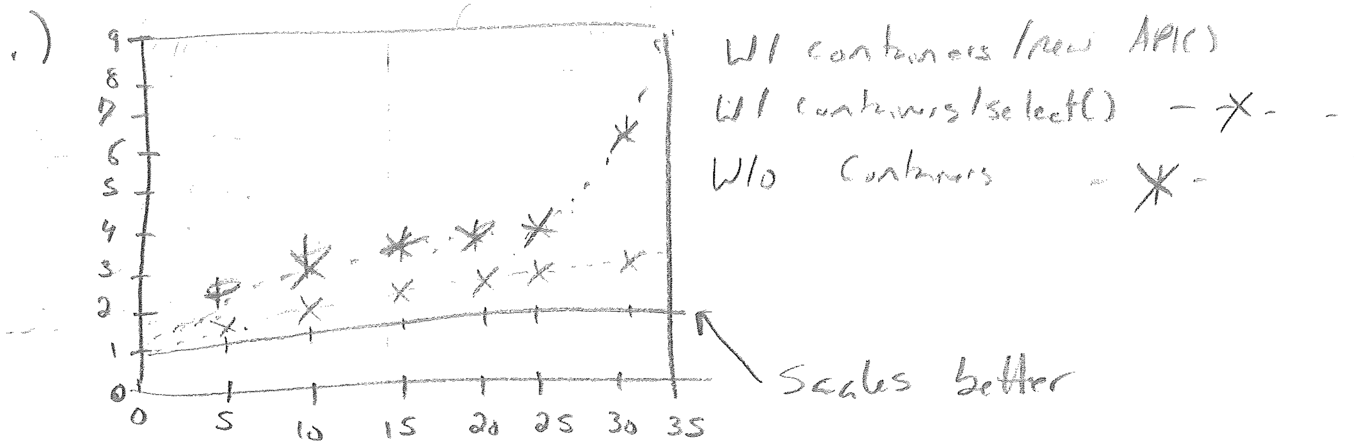
3 A new abstraction, resource container, as the resource principal. Activities, threads, processes ~~bing~~ bind to a resource container. All processing is charged to that container. Any processing for a thread is charged to ✓ the resource container it is bound ~~to~~ at the moment. Resource containes can be defined at any level. For excample a client request can be defined to have a resource container. All processing of this request gets charged to this container only.

4.)



Threads scheduled based on container attributes, separated from thread/process that may service different logical operation.

5.)



6) This system supports hierarchical containers. Assume there are 2 containers at Level 1 and they share 60% & 40% of the resources. The second container in turn can have 2 Child containers that each get 20% of the resources.

✓

The Lottery scheduling achieves the same effect using the currency abstraction. Each task can define its own currency equivalent to the total value of the base currency allocated to it and share it with its sub tasks.

Michael Mesnier, Feng Chen, Tian Luo, Jason B. Akers
Differentiated Storage Services
SOSP '11, October 23-26, 2011, Cascais, Portugal

1. MOTIVATION/PROBLEM STATEMENT: ✓✓

Different classes of data need different levels of service. The computer system makes attempts to obtain some form of differentiated services through intelligent allocation, but semantic information is lost in the block layer. The objective is to develop a mechanism for classifying I/O so that different classes of data be handled with different storage policies.

2. The approach uses classification of data that can be treated differently by the storage system, without modifying the block interface. The filesystem Classifies I/O for a certain performance policy. the class is transfered to the storage system using 5 bits in the SCSI command. The storage device is in charge of enforcing the policy. ✓

3. The interesting technique is the ability to relay relevant information down to the block level with 5 bits of space that are already present in the SCSI standard. This information can also be used to perform more intelligent caching. (selective allocation/selective eviction)

④ How the implementation works:

FS classifies the inode buffer
↓ bh→b_class
OS block layer uses this when generating Block I/O request
↓ bio→bi_class
Class from BIO is copied to SCSI CDB
scpnt→cmnd[6]
(5-bit vendor-specific Group Number field)

⑤ Classifying data for general purpose storage (ext3).

Ext3 Class	Class ID	Priority
Superblock	1	0
Bitmap	2	0
Inode	3	0
Indirect	4	0
Directory	5	0
Journal	6	0
File ≤ 4K	7	1
File ≤ 16K	8	2
⋮	⋮	

Ext3 classifies data as metadata types or data blocks for files of varying sizes.

The device-layer chooses to cache metadata with higher priority, then small files, medium files, etc...
(Each priority could have its own LRU, e.g.)

⑥ **Pros:-** Computer System don't have to try hard to manage blocks in ~~den~~ storage system and storage system don't have to know about the complete semantic of FS.

- Different storage system can provide different policy for each classification. for eg. RAID can use different policy of cache magement & local hard disk can use some other policy.
- Backward compatible

Cons:-
- Computer system & storage system has to agree upon I/o classification & policy.
- Not standardized yet
- Require some changes in OS & file system, also in storage system.

Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B.
Design and Implementation of the Sun Network Filesystem
Proceedings of the Summer 1985 USENIX Conference, Portland OR, June 1985, pp. 119-130.

① Motivation

a) To enable sharing of filesystem resources in a network of non-homogeneous machines

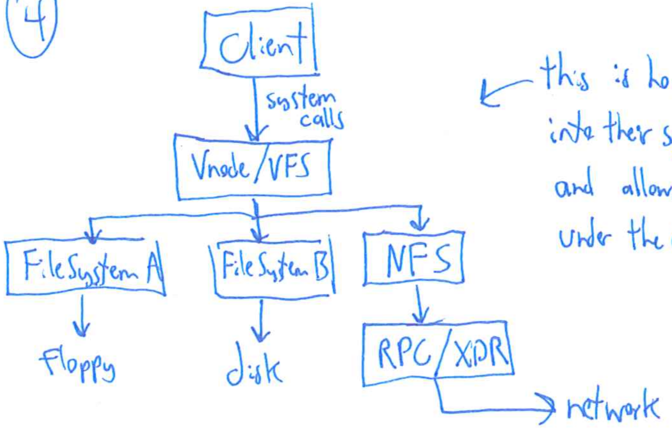
b) Provide UNIX-like interfaces to operate on remote files.

② Result / Contribution / Approach

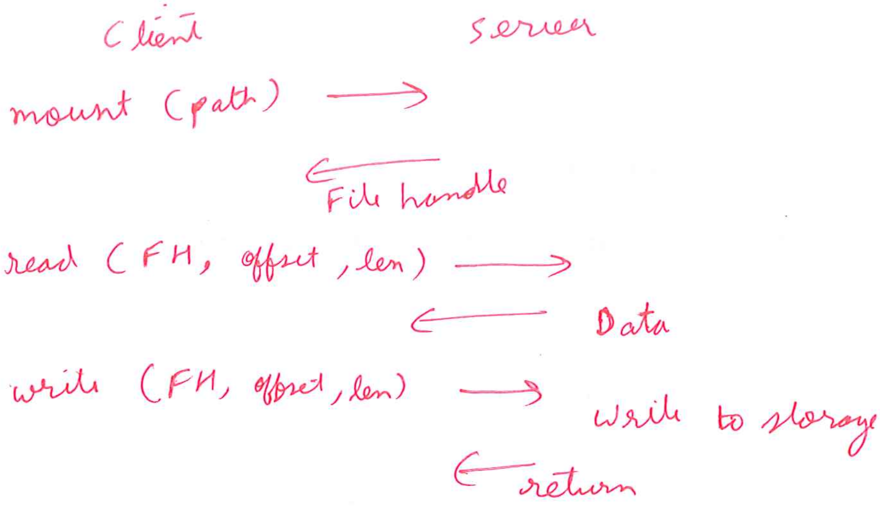
- The approach/contribution of NFS was to design a system that utilized a stateless protocol. This enabled/resulted in a simpler to reason about protocol and made crash recovery simple and fast (simply reboot)
- Another major contribution was the VFS and vnode which allowed transparent access to remote files (i.e. it did not change the structure/semantics of path names)

③ Vnodes allowed for transparency. This allowed all file systems and nodes to be treated in a uniform way. Vnodes contain a pointer back to its parent VFS and a pointer to a mounted-on VFS. Any node can be a mount point.

④



this is how they layered VFS into their system. It allows multiple FS's to sit at the backend, and allows NFS to be able to pretend it's just like the local FS's, even though under the covers it's doing network I/O



~~Server~~
Important Points.
No state
All communication
using File handle
only.

6. Pros: Comparatively simple, stateless on server, worked with diskless machines, matched UNIX file semantics

Cons: Poor performance due to synchronous block-level I/O rather than caching, no remote file locking runs risk of interleaved data on writes, poor scalability

Comparisons: AFS provided better performance for large file I/O and scaled much better; however it could not run on diskless machines well and was significantly more complex.

AFS

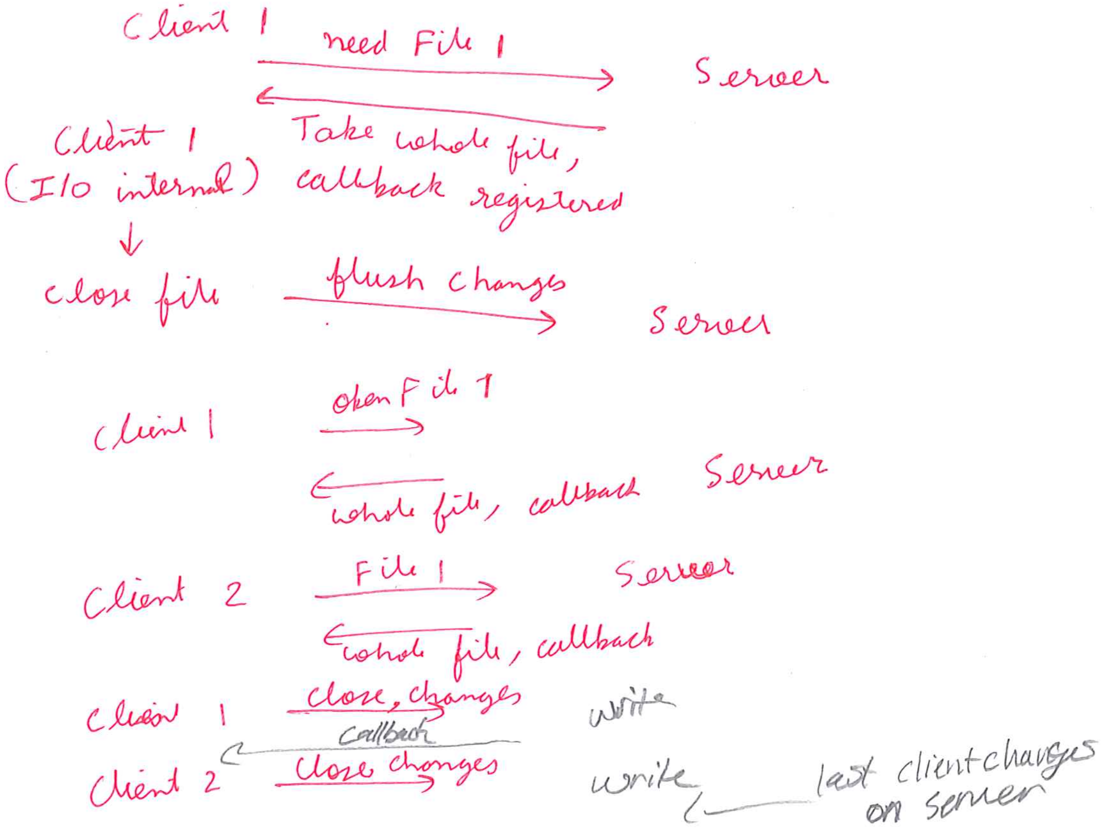
Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., and West, M.J.
Scale and Performance in a Distributed File System
ACM Transactions on Computer Systems, Vol. 6, No. 1, February 1988, pp. 51-81.

①. Large scale affects performance and administrative/day-to-day operations in a distributed file system. ∴ AFS's motivation is to scale a distributed file system. ok

2. Came up with a distributed FS that scaled well. Used caching with files, more efficient path naming, improved server process communication, and a change in the low-level representation.

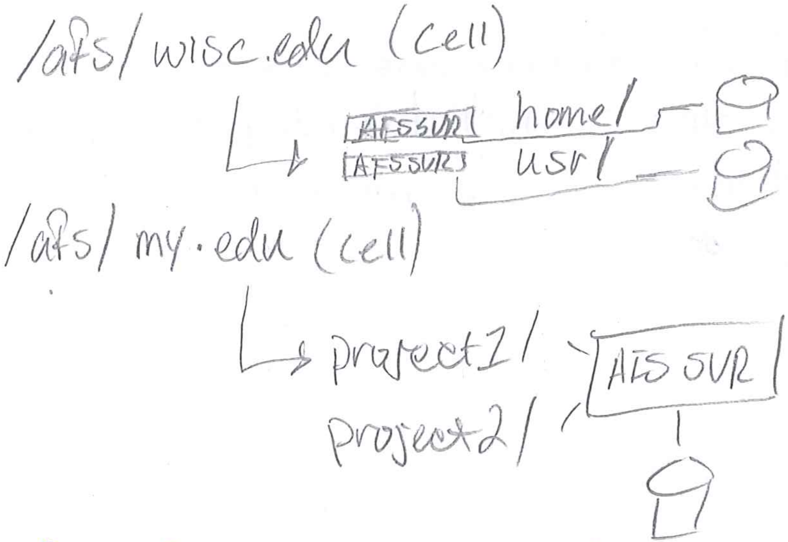
3. They use a stateful design to read data to reduce accesses to server. The first time a file is read, the server is contacted. Then a callback is registered. As long as the callback is not broken, a clientA can use the local copy. When another clientB modifies the file and writes it to server, the server breaks the callback with client A.

4.5.



5.) see 4

4.) Diagram Global namespace, distributed filesystem.



6) Good for doing many operations on files + reducing network overhead.

Bad on small reads/writes, and files larger than disk.

Overall, the state on the server became an important facet on distributed FS's; it does enough good that it's better than stateless.

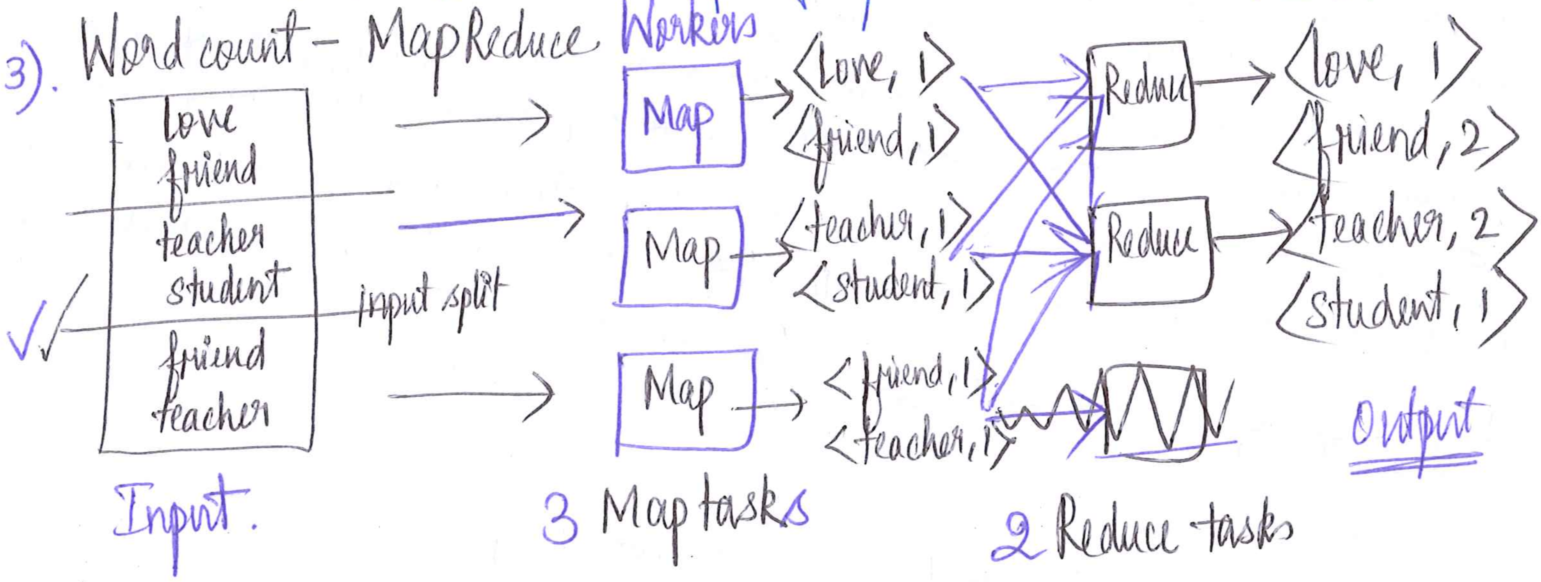
Problem Statement?

Jeffrey Dean and Sanjay Ghemawat
MapReduce: Simplified Data Processing on Large Clusters
OSDI'04: Sixth Symposium on Operating System Design and Implementation,
San Francisco, CA, December, 2004.

✓ ★ ① Motivation – Provide programmers an abstraction to write parallel programs to deal with huge data sets without worrying about underlying difficulties like loadbalancing, failures, data distribution etc. Also come up with an implementation for use inside Google.

② ✓ Major Result: MapReduce programming model
"a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations" + "~~and~~ an implementation that acheives high performance on PCs."

3). Word count – MapReduce



✓ Steps:

4. a) MapReduce library in user program 'word count' splits the input into M pieces (in our case 3) and starts many copies of program on a cluster of machines

B)

b) One of them is master. There are M map tasks & R 'B' reduce task 'R' is specified by the user. (2 in the above examples

c) In 'Map' task, the program builds a word count for all lines in its split,

✓ d) These word counts are partitioned into R regions by the user given partitioning function.

e) Reduce workers are notified by master & each reduce worker reads the corresponding split

f) It then applies the reduce function, summing the word count of portion from all map workers. And appends the output to a file

(5) :- Example :- Sorting :-

✓ Mapper will emit a <key, record> for each input where key is the property on which we have to sort the records. Thus the task of mapper is very easy.

~~The~~ Also, reducer don't have to do anything, they will just emit the same <key, value> pair which they receive.

The main thing here is done by the ~~Map reduce~~ framework itself which partition the output (of mapper) into sorted intermediate data before giving it to reducer.

(6) Pro + Cons ✓

\+ Allows programmers to quickly implement massively parallel programs without worry underlying details of parallelism

\+ makes the code ~~easy~~ easy to reason about. Only have a Map and Reduce Function to debug

\-

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
The Google File System
19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003.

① Motivation / Problem Statement

Motivated by various observation of kind of workload they have to perform & type of hardware they run on:-

(i) failures are a norm
(ii) Files are typically very large (GBs)
✓ (iii) Reads are sequential (from start to end)
(iv) Files mutate normally by appends only (random writes are less)

2. Major Result / Contribubilion

They developed a distributed file system that is being used in Google data centers.

Approach / Idea

A master node executes all namespace operations. All other
✓ nodes store data as chunks of fiscid size. They are not involved with name space operations. Master directs clients
✓ to appropriate chunk server node. File I/O happens between the ~~master~~ chunk server and client.
For performance Master maintains all data structures in memory.
Master has a redundant Shadow server which activates as a read / only master in case of failures.

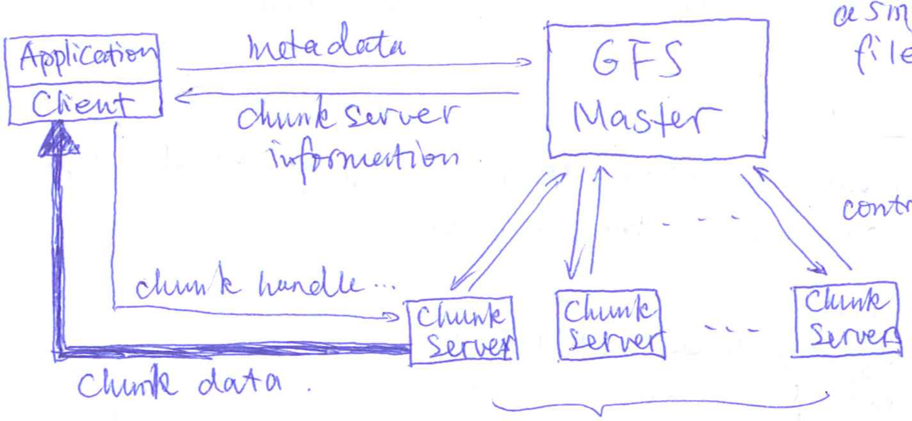
3) Techniques / Interesting ideas

✓ i) The FS design is closely tailored to work efficiently for the workloads typically observed in Google's environment.

ii) Since appends are common, GFS allows atomic appends without guarantee for the actual order of concurrent appends

(iii) Each chunk is broken up into 64k blocks and the blocks are checksummed.

④ GFS Architecture:



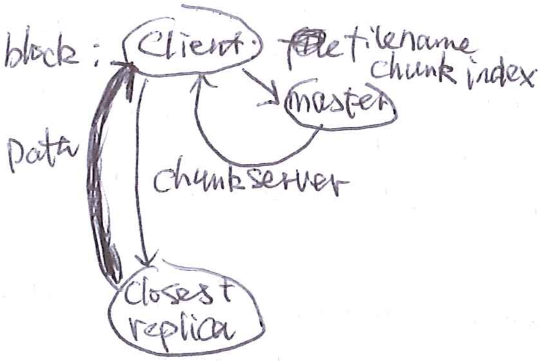
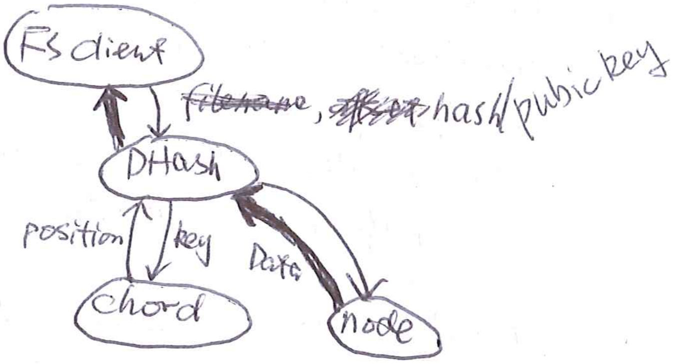
a single master maintains all file system metadata.

control msgs. exchange: instructions, server states.

multiple chunk servers could be accessed by multiple clients.

⑤

⑥ GFS Vs. CFS.

	GFS	~~GFS~~ CFS
Block size:	64MB	8KB
Lookup a block:		
replica:	3 replica, across racks	N replica following the desinated node.
scale:	claster	global

Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica
Wide-area cooperative storage with CFS
In the Proceedings of the 18th ACM Symposium on Operating Systems Principles
(SOSP '01), Chateau Lake Louise, Banff, Canada. October 2001

① Motivation: Existing peer-to-peer systems have shown in practice to be robust, to reasonably balance system load, and to harness machines in the system for storage making them useful when they would otherwise be idle.

Problem: Developing a P2P file system with these above benefits.

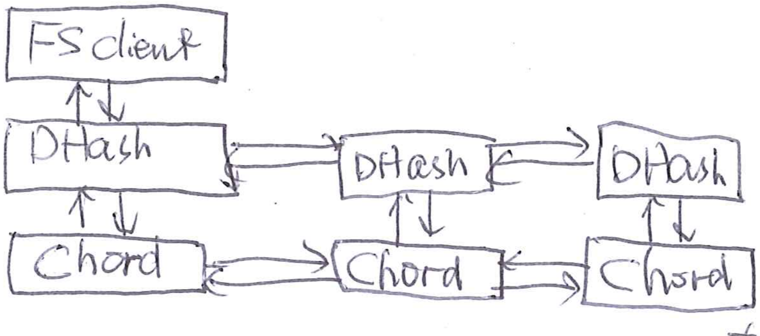
② CFS is implemented using the SFS file system toolkit runs on multiple OS.

CFS server provides a distributed hash table (DHash) for block storage

CFS client interpret DHash blocks as a file system. DHash distributes and caches blocks at a fine granularity to achieve load balance, uses replication for robustness and ↓ latency with server selection.

③ Chord uses a ring of servers representing the hash space. Each server is responsible for all blocks that hash to values between the server's designated hash value and the previous server's hash. Blocks are duplicated "forward" around the circle, so server failures result in minimal block movement.

④



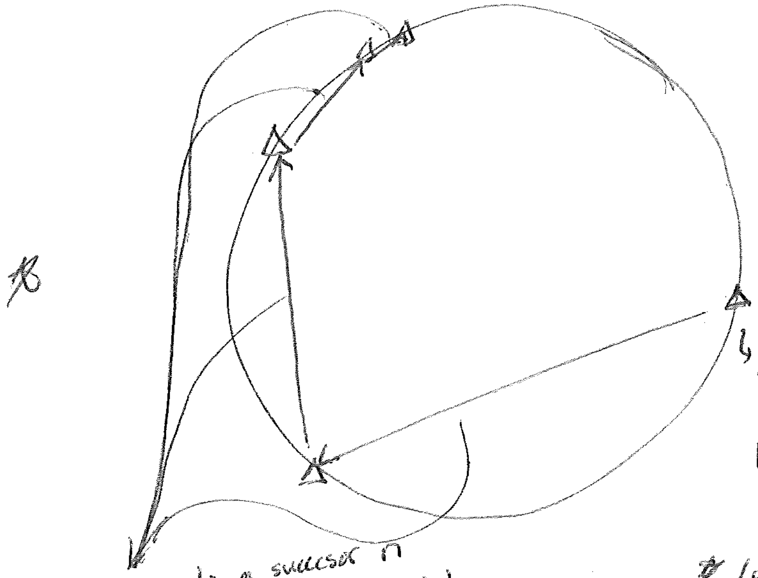
Below Chord; you only see blocks (8K?).

DHash is in charge of replicating blocks, distribute them across nodes.

Chord is the consistent hashing service which builds the "ring". provided a key, it will find the position for you.

FS Client is in charge of interpreting the block; metadata, file data, ...

⑤ CFS Lookup



user provides key → k

↳ Each Chord node uses successor list and finger table to perform lookups

Each jump to a successor n moves strictly closer to the data requested by the key client supplied

- Lookups with only successor list requires an average of N/2 message exchanges
- Lookups with finger table closer to O(logN)
  ↳ Finger table contains m entries. The ith entry in table is ~~a node~~ first node that succeeds n by at least 2^{i-1} in the circle. Accelerates jumping

⑥ Pros

- Scales well (finger tables). #RPC per query grows very slowly w.r.t. #servers
- Seemingly robust to server exits/crashes
- Read-only, only publisher may change their data blocks

Cons

- Read-only, not as general purpose as NFS, AFS

Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, and Nickolai Zeldovich
An Analysis of Linux Scalability to Many Cores
In Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI), Vancouver, Canada, October 2010

1. Motivation and Problem Statement

There is a general sense in the community that traditional linusc kernel designs don't scale well on multicore processors. The authors aimed at analysing this ~~on~~ quantitatively. They also proposed solutions for bottlenecks. Their aim was to establish ~~these things~~ the limits of linusc multicore scalability for upto 48 cores. They are aware ~~of~~ that higher number of cores may bring new bottlenecks to light.

2. Approach - 1. Show scalability problems in linux kernel with applications that are known not scale well and applications that are kernel intensive.

2. Use common parallel programming techniques to solve those issues.

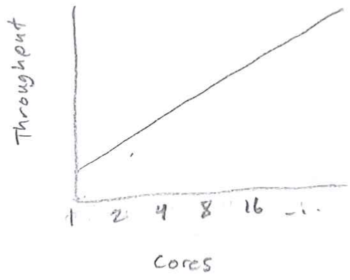
Techniques - sloppy counters, lock free comparison, per core ds, Eliminating false sharing

3. For their analysis, they made sure to run a variety of commodity programs meant to scale well and/or are kernel-intensive in an in-memory tmpfs file system to avoid disk bottlenecks from dominating the results

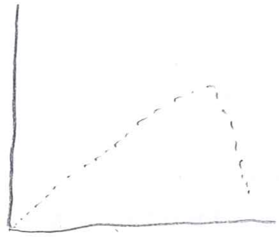
The aforementioned 'sloppy counter' is an answer to the issue of bottlenecks due to reference-counted resources; this shared count must be accessed by all cores, and so became a bottleneck even with atomic ops (since changes are serialized); sloppy counters keep reference counts on each core that can usually be used instead of going to the shared reference count.

This is O.K. (sloppy) b/c the exact count of references doesn't matter to each thread/cpu. Only when all counts are 0 do we care (free resource).

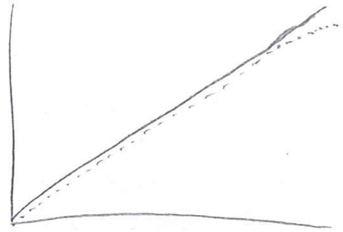
④ Ideal



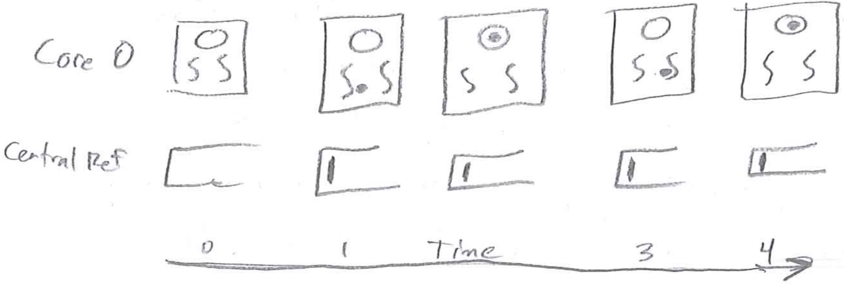
Discovered



After Careful Targeting of Bottlenecks



Sloppy Counters



Core 0 acquires a reference from central counter at time 1. At time 2 it is using it (inc local sloppy counter). At time 4 a process on same core may use this reference again without going to central counter.

⑤ Among the bottlenecks, they found problems with usage of 3 kernel data structures per-superblock list of open files, table of mount points, pool of free packet buffers. These big structures were problematic due to lock contention. To fix this, they refactored the data structures to minimize lock contention in the common case. (fine grain locks)

⑥ Pro: Solved a number of bottlenecks, the easy to fix ones anyway.

Cons: Always another problem is discovered after fixing one issue.
Some problems require significant refactoring of the app/kernel (the paper didn't address these)

Austin T. Clements, M. Frans Kaashoek, Nickolai Zeldovich, Robert T. Morris, and Eddie Kohler
The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors.
In Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP), Farmington, PA, November 2013.

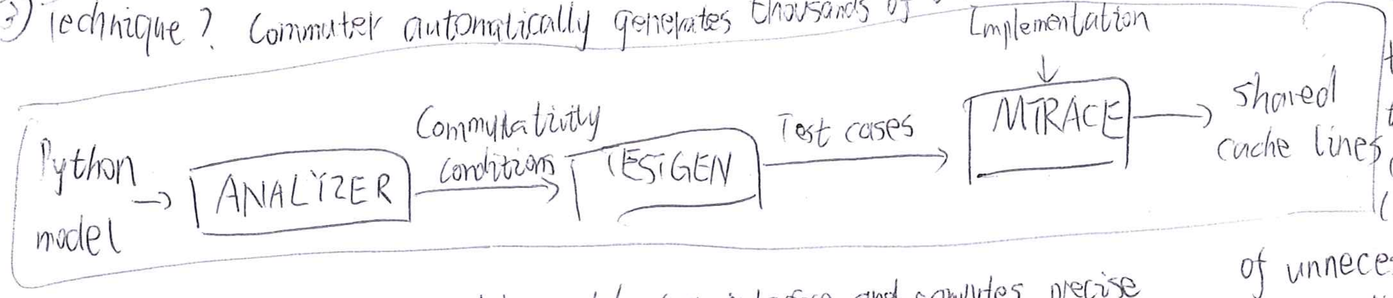
① Motivation: Establish whether opportunities for scalability exist by examining " software interfaces.
The Scalable Commutivity Rule: "whenever interface operations <u>commute</u>, they can be implemented in a way that scales."

✓

② <u>Approach</u> :- ~~the~~ analyze system interface using <u>SIM</u> to find scalability bottlenecks. Then try to ~~make~~ ~~this~~ remove those bottleneck using various approaches (choose any FD, sloppy counter).

<u>Contribution</u> : (i) Commutativity Rule → "whenever interface operations commute, they can be implemented in a way that scales".
(ii) COMMUTER.

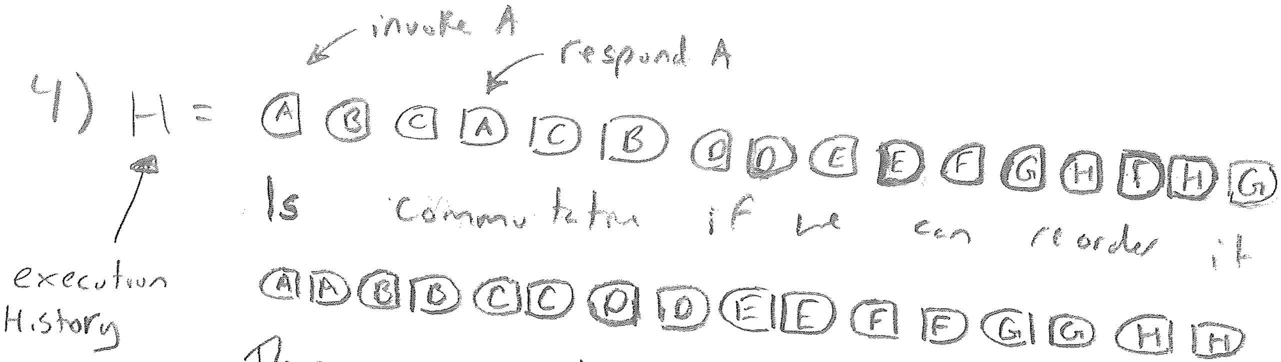
③ Technique ? Commuter automatically generates thousands of test cases for your interface and tests your implementation to root out the exact causes and locations of unnecessary scalability-limiting sharing



① Analyzer takes a symbolic model of an interface and computes precise conditions ~~and generates con~~ under which that interface's operations commute

②

4) H =



Is commutation if we can reorder it like...

This means the thread tasks are independent.

⑥ Pros :

1. Embed Scalability in the software right from designing interfaces.
2. Established a formally provable "scalable commutativity rule".

Cons :

1. Changes to the POSIX calls require a new kernel implementation. This reduces usability.

Xi Wang, Nickolai Zeldovich, M. Frans Kaashoek, Armando Solar-Lezama
Towards Optimization-Safe Systems: Analyzing the Impact of Undefined Behavior
In Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP),
Farmington, PA, November 2013.

1) MOTIVATION & PROBLEM STATEMENT:

✓ → Compilation made certain portions of the code disappear. (∵ optimization of code).

✓ → Condition checks inserted by programmers are "undefined" behavior ~~and~~ as per the language standards and will be 'optimized away' by the compiler, causing security concerns.

→ The blame shift game b/w compilers designers and programmers

2) New static checker STACK. - Identifies unstable code. ✓

- model for unstable code & approach to identify it.
- well-defined program (using this can eliminate unreachable code & simplifying unnecessary comp).

③ Technique: ☆

Given the 'well defined program' formalism $\Delta(x) = \bigvee_{\text{for } e \in P} R_e(x) \rightarrow \neg U_e(x)$ ☆

phase i) Run an optimizer without the assumption of $\Delta(x)$

phase ii) Run the optimizer with the assumption of $\Delta(x)$

The differences between the results in phase ii) and phase i) gives the unstable code. Two kinds of optimizers are described
1. Sliminator
2. Simplifier

✓

Something interesting - If the optimizer performs poor in phase i) then there is chance of false +ves and if phase ii) optimization is not accurate then false -ves are possible

④. Figure 4 of paper :- It shows which compiler discard which kind of undefined behaviour in which optimization level.

There are many things which this table suggests

(i) Almost every compiler optimize/discard some kind of undefined behaviour

(ii) Usually the optimization on undefined behaviour are done in O2 level

(iii) Increasing version no. of the same compiler becomes more aggressive towards undefined behaviour optimization.

(iv) Optimization exploit undefined behaviour from library functions also.

⑤ Example of unstable code:

```
char *buf = ...;
char *buf_end = ...;
unsigned int len = ...;
if (buf + len < buf)
    return;        /* intent is to catch overflow in
                      ptr arithmetic. But ptr overflow
                      is undefined, so the gcc optimizer assumes
                      the condition is always false and
                      therefore discards the if statement */
```

6) Other similar systems detected such bugs by preparing a variety of test environments. KLEE generates ~~test~~ cases by symbolic execution. Compilers provide optimization flags but not enough. Checkers directly target undefined behavior but Stack finds dead code because of undefined behavior

PROS : Working system applied to real software like kernel, Postgres, kerberos. Can scale

CONS : Incomplete list of covered undefined code

Approximation can lead to missing of unstable code