In this class, “caching” will always mean “on-demand caching”

True outside this class as well

(What DropBox does is not called “caching”)

E2E Latency Really Affects Business

"Being fast really matters…half a second delay caused a 20% drop in traffic. and it killed user satisfaction"

- Marissa Mayer @ Web 2.0 (2008)

"…a 400 millisecond delay resulted in a -0.59% change in searches/user",

[i.e. Google *would lose 8 million searches per day - they'd serve up many millions fewer online adverts*]

- Jake Brutlag, Google Search (2009)

"…for Amazon every 100 ms increase in load times decreased sales with 1%"

- Andy King, book author

"…when 50% of traffic was redirected to our edges preliminary results showed a 5.9% increase in click-thru rates"

- Andy Lientz, Partner GPM, BingEdge (2013)

Dilemma

Businesses want to know your every click and keystroke

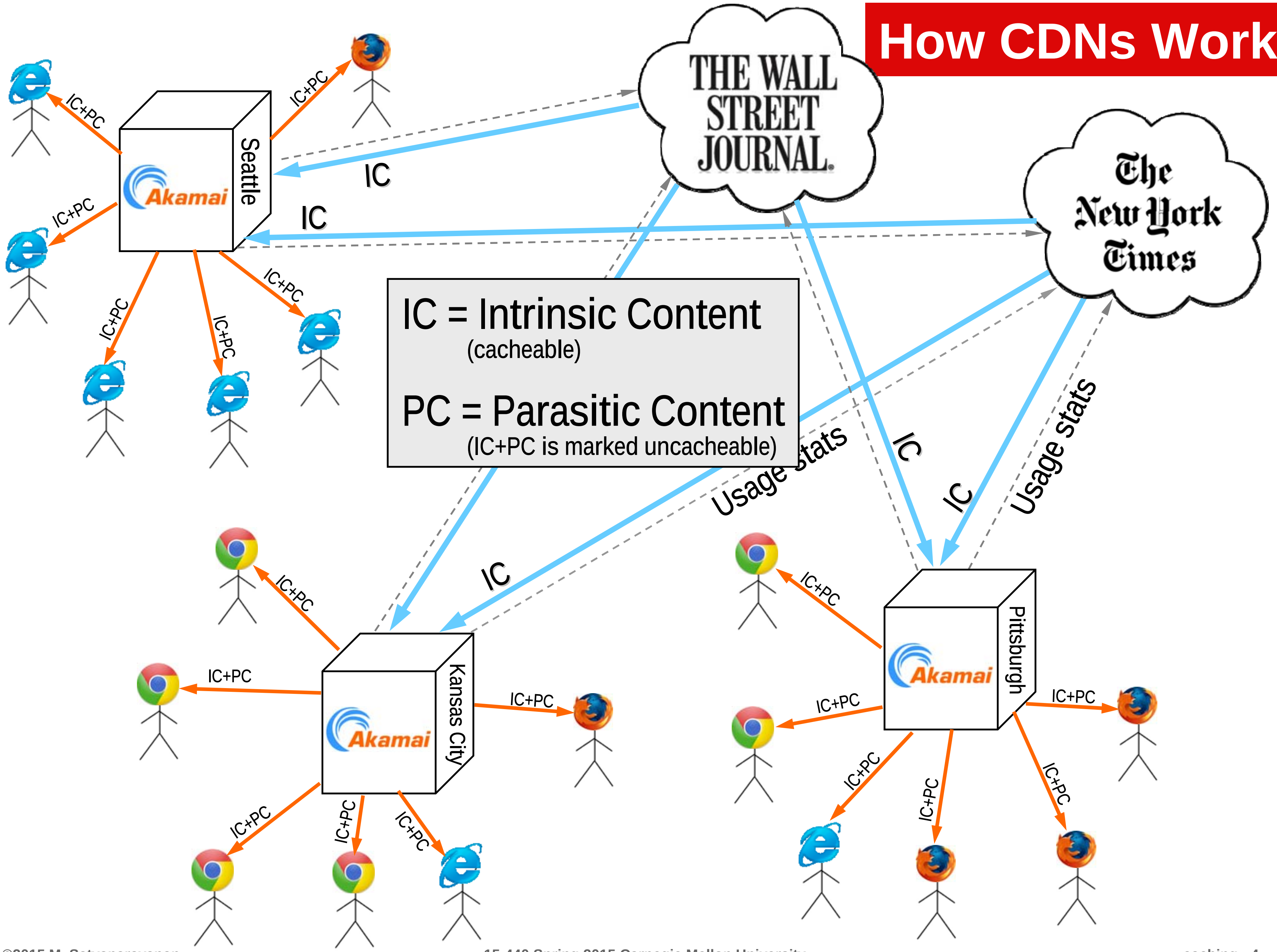
- deep, intimate knowledge of user
- client caching hides this knowledge from the server
- so server marks pages as "uncacheable"
  this is often a lie, because the content is really cacheable
- however, the lack of caching hurts latency and user experience

Can businesses benefit from caching without giving up control?

Solution: Content Delivery Networks (CDNs)

- effectively third-party caching sites trusted by businesses
- late binding of parasitic content by caching site
- pioneered by Akamai in the late 1990s
- many examples now
  CloudFront (Google), Windows Azure CDN, Streamzilla, …

How CDNs Work



IC = Intrinsic Content
(cacheable)

PC = Parasitic Content
(IC+PC is marked uncacheable)

Recap: Key Questions

1. *What data should you cache and when?*

   Fetch policy

2. *How do updates get propagated?*

   Update propagation policy

3. *What old data do you throw out to free up space?*

   Cache replacement policy

Real-World Complications

Reality 1: ***Cost of remote data access often not uniform***

- **often takes the form *Ax + B* for *x* bytes**
- **may be more complex as *x* gets larger (e.g. TCP file transfer)**

Reality 2: ***"Nearby" objects often accessed soon after object access***

- **another empirical observation about real systems in real use**
- **referred to as *"spatial locality of reference"* or *"spatial locality"***

Reality 3: ***Remote data more coarsely addressable than local***

- **typically a scalability tradeoff at next level of memory hierarchy**
  same number of address bits can cover larger volume of data
  e.g. cache line width, page size, whole file, tape mount
- **"wholesale" versus "retail"**

Combining observations ⇒ *fetch more than you need on miss*

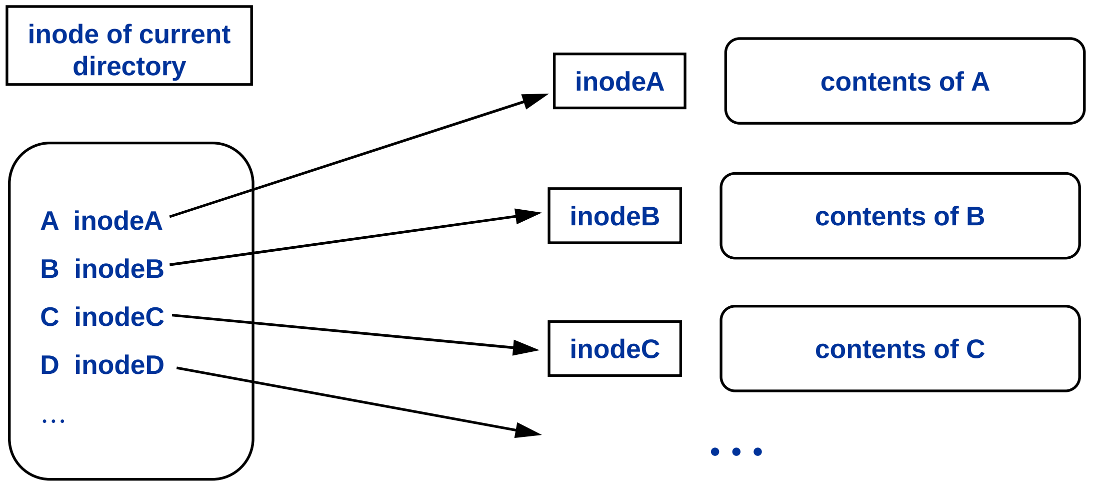
- **effectively amortizes cost of fetch**
- **assumption sometimes violated**
- **extra data fetched is then useless (may even hurt)**
  can be viewed as crude form of prefetching

Spatial & Temporal Locality

Temporal and spatial locality are very different properties

- caching implementations often tightly combine these assumptions
- one can exist without the other
  - spatial without temporal: linear scan of huge file
  - temporal without spatial: tight loop accessing just one object

For example, consider typical implementation of "rm -f *"



shell expands "*" into list

loop iterates through list

- stat object
- unlink object

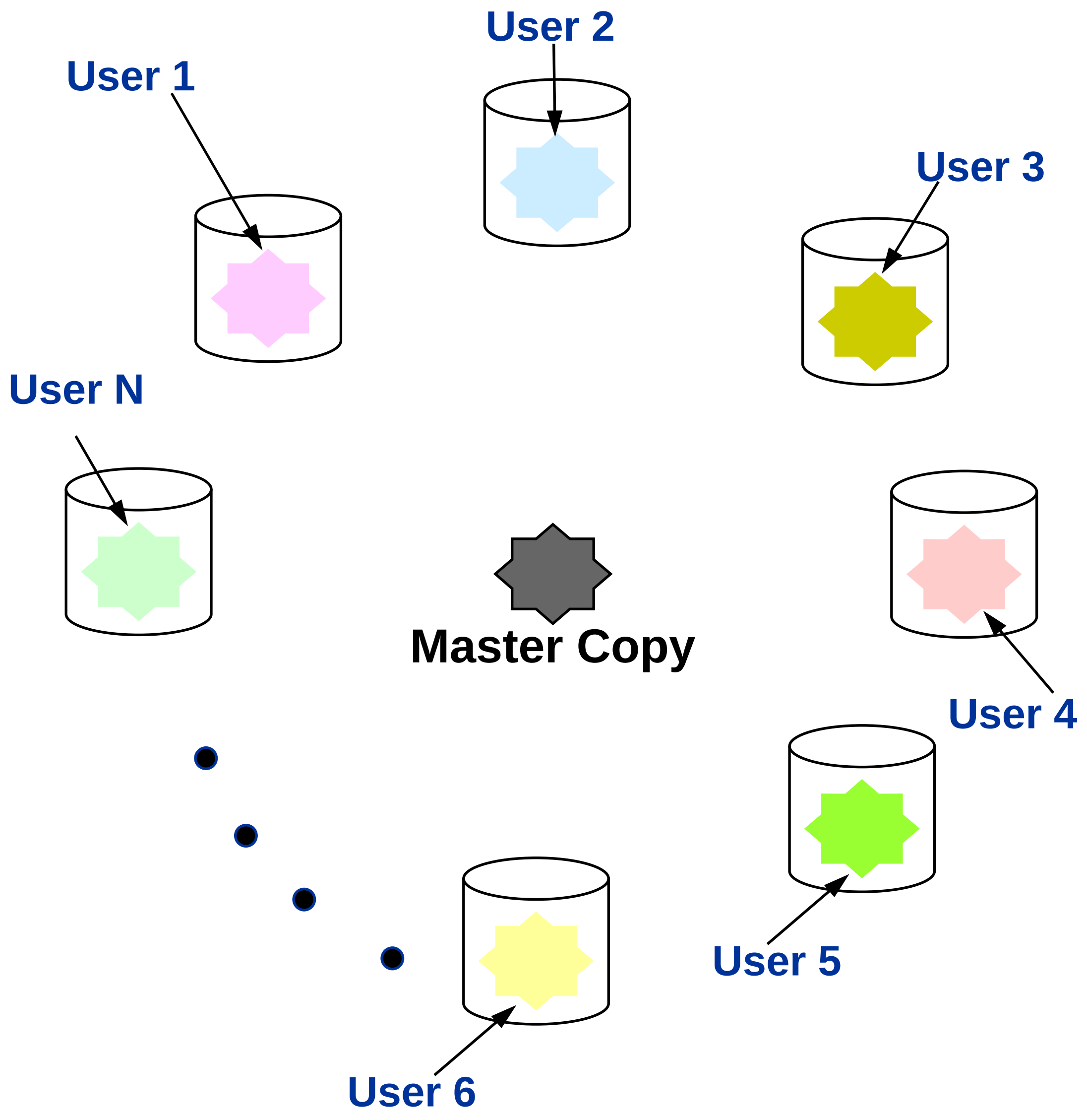
parent directory exhibits temporal locality

directory entries exhibit spatial locality

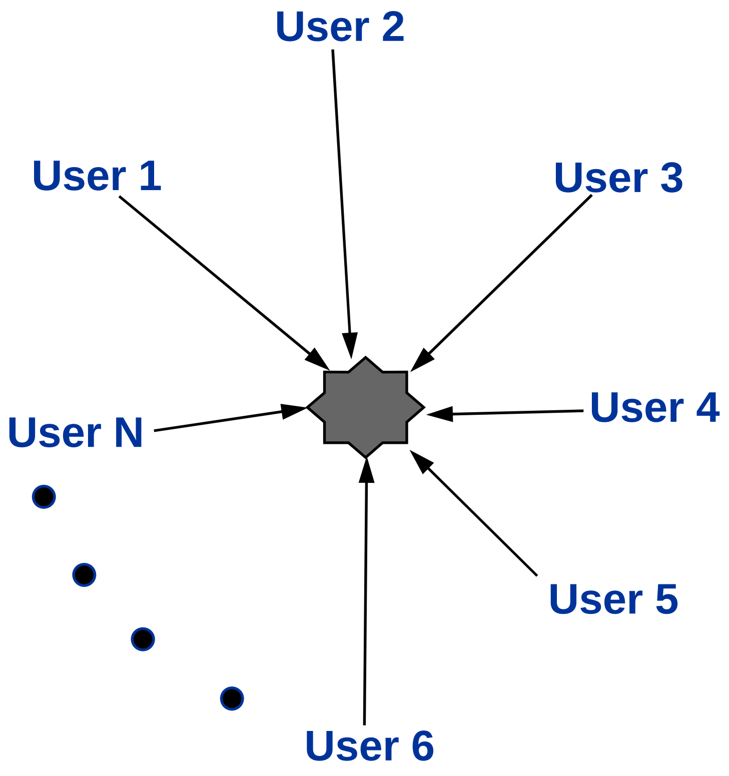
Update Propagation

(aka “Cache Consistency”)

Caching Reality



Desired Illusion



One-copy semantics

What Makes This Hard?

Physical master copy may not exist

- **hosts track who has most recent copy**
- **more likely in P2P scenarios than client-server scenarios**
- **also common in multiprocessor hardware caches**

Network may break between some users and master copy

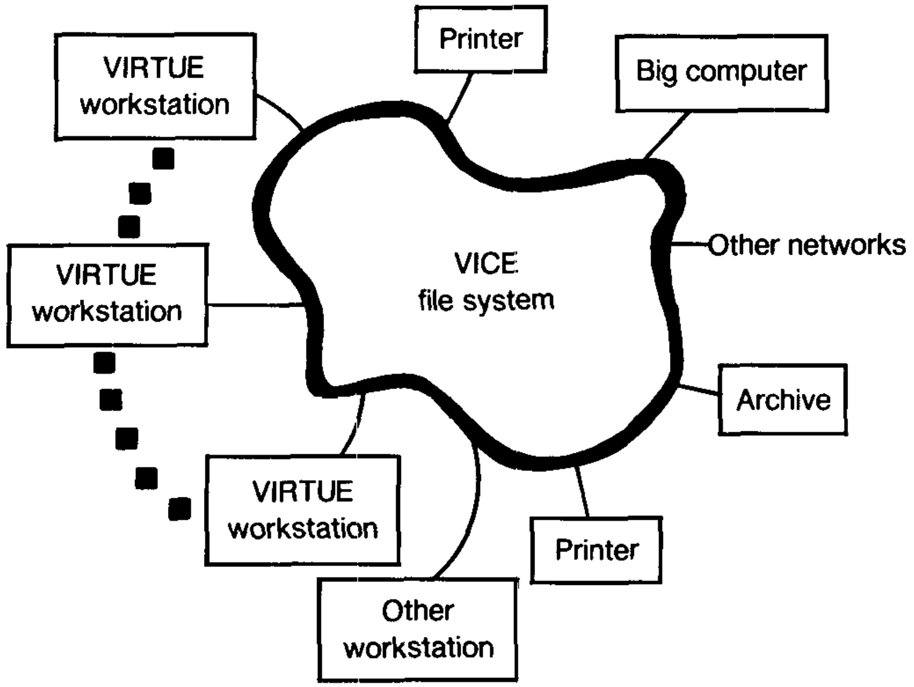
- **disconnected sites see no further updates**
- **other sites don’t see updates by disconnected site**

Intense read- and write-sharing across sites

- **generates huge amount of cache propagation traffic**
- **interconnect becomes bottleneck for cache access**
- **neither writer-bias (write-back) nor reader-bias (write-through) helps**
- **caches effectively useless**

Caching From the Cloud

Earliest insights: CMU circa 1986



from
"Andrew: a Distributed Personal Computing Environment"
Morris, J.H., Satyanarayanan, M., Conner, M.H., Howard, J.H., Rosenthal, D.S., Smith, F.D.
Communications of the ACM, March 1986

The amoebalike structure in the middle, called VICE, is a collection of communication and computational resources serving as the backbone of a user community. Individual workstations, called VIRTUEs, are attached to VICE and provide users with the computational cycles needed for actual work as well as a sophisticated user–machine interface. (VICE stands for "Vast, Integrated Communications Environment"; VIRTUE, for "Virtue is reached through UNIX® and EMACS.")

User mobility is supported: A user can walk to any workstation in the system and access any file in the shared name space. A user's workstation is personal only in the sense that he owns it.

System administration is easier: Operations staff can focus on the relatively small number of servers, ignoring the more numerous and physically dispersed clients. Adding a new workstation involves merely connecting it to the network and assigning it an address.

from
"Scalable, Secure and Highly-Available File Access"
Satyanarayanan, M.,
IEEE Computer, May 1990

Optional Readings

- “Scale and performance in a distributed file system”
  Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., and West, M. J.
  ACM Transactions on Computer Systems Volume 6, Number 1, Feb. 1988
- “Leases: an efficient fault-tolerant mechanism for distributed file cache consistency”
  Gray, C. and Cheriton, D.
  In Proceedings of the Twelfth ACM Symposium on Operating Systems Principles, Litchfield Park, AZ, 1989

1. Broadcast Invalidations

Basic Idea

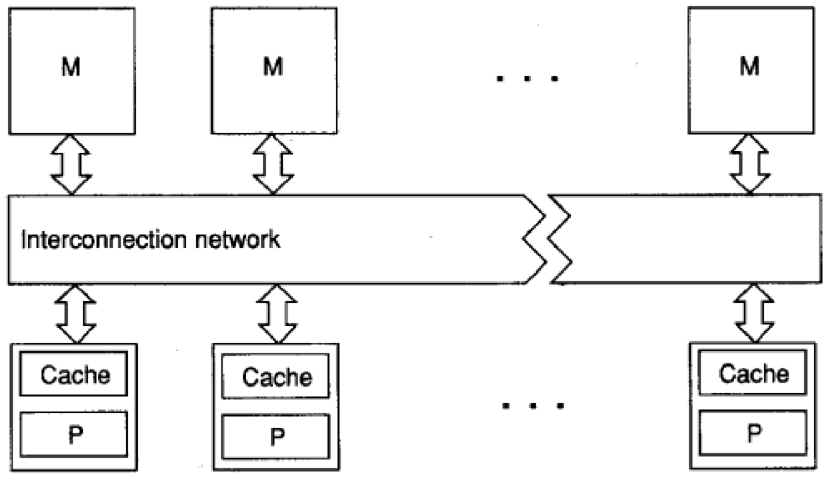
Every potential caching site notified on every update

- **No check to verify caching site actually contains object**
- **Notification includes specific object being invalidated**
- **Effectively broadcast of address being modified**

At each cache site, next reference to object will cause a miss

Original Use

- **Small-scale multiprocessors in early 1970's**
- **"snoopy cache" multiprocessor variants of 1980s-1990s**



Strengths

- very strict emulation of “one-copy semantics”
- no race conditions if updater blocked until all caches invalidated
- simple to implement

Limitations

- wasted traffic if no readers elsewhere
- updating process blocked until invalidation complete
- not a scalable design

2. Check on Use

Basic Idea

- *reader checks master copy before each use*
  - conditional fetch, if cache copy stale
- has to be done at coarse granularity (e.g. entire file or large block)
  - otherwise every read is slowed down excessively
- at whole file granularity $\rightarrow$ "*session semantics*"
  - open {read | write}* close $\rightarrow$ "session"
  - approximation to strict one-copy semantics at bit-granularity

Original Use

- AFS-1 (circa 1983)

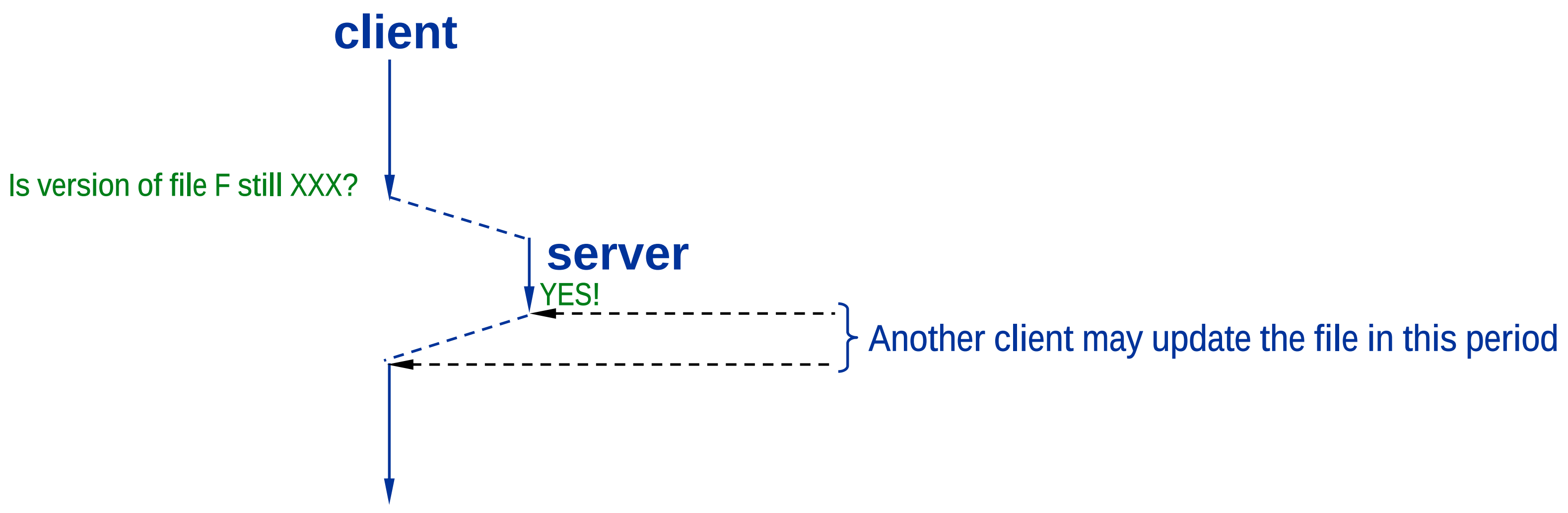
Advantages

- strict consistency at coarse granularity
- easy to implement, no server state
- servers don't need to know of caching sites

Disadvantages

- **slows read access on loaded servers & high-latency networks**
- **check is almost always success → frivolous traffic**
- **load on network and server**

"Up to date" is relative to network latency


client
Is version of file F still XXX?
server
YES!
Another client may update the file in this period