Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

CA 2253829 C 2002/07/23

(11)(21) 2 253 829

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(86) Date de dépôt PCT/PCT Filing Date: 1997/05/20
(87) Date publication PCT/PCT Publication Date: 1997/11/27
(45) Date de délivrance/Issue Date: 2002/07/23
(85) Entrée phase nationale/National Entry: 1998/11/06
(86) N° demande PCT/PCT Application No.: GB 1997/001363
(87) N° publication PCT/PCT Publication No.: 1997/044747
(30) Priorités/Priorities: 1996/05/20 (9610505.1) GB;
1996/05/22 (96303645.4) EP

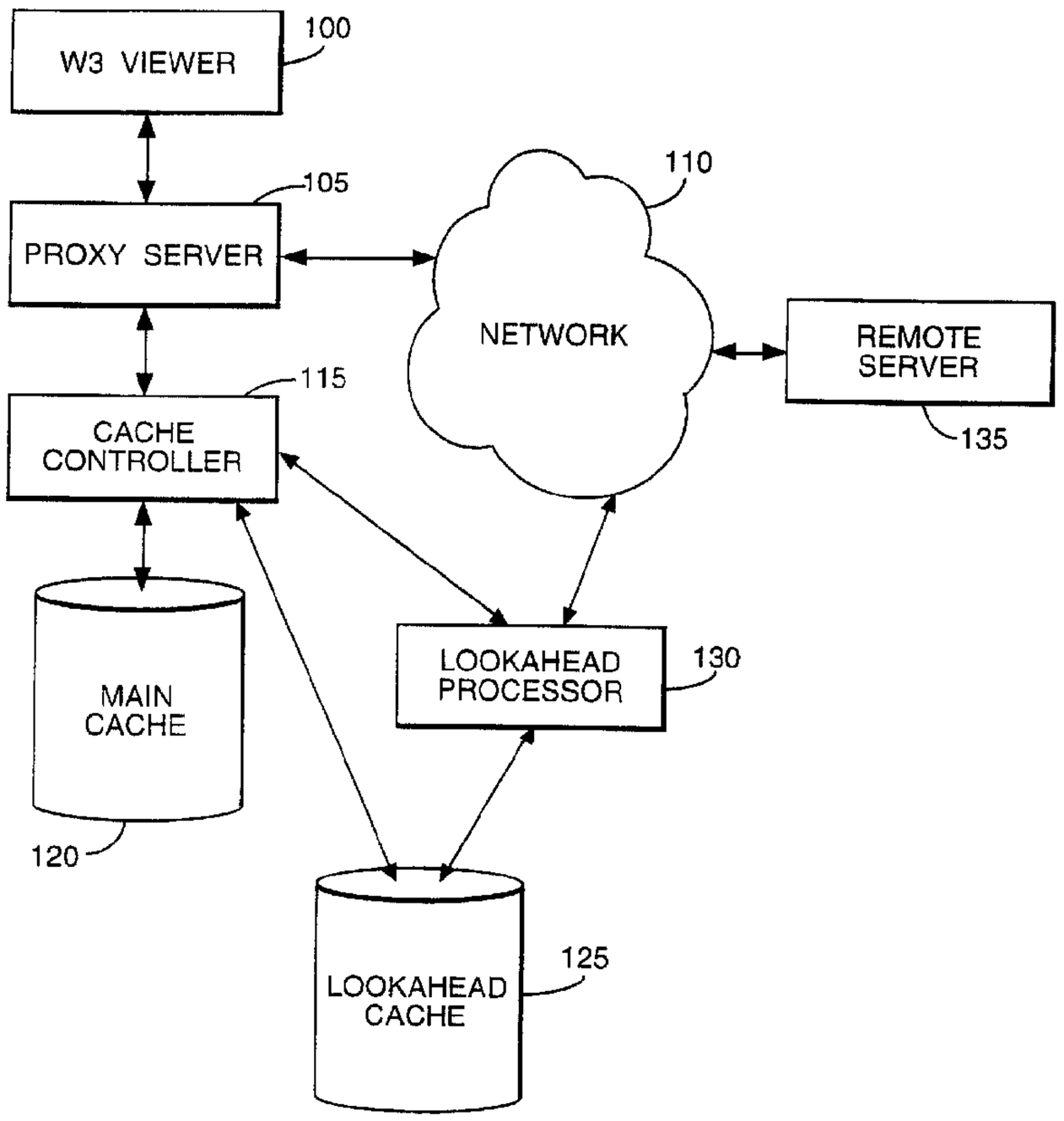
(51) Cl.Int.^{6}/Int.Cl.^{6} G06F 17/30

(72) Inventeurs/Inventors:
WEEKS, Richard, GB;
STEPHENS, Lee Michael, GB;
DAVIES, Nicholas John, GB;
REVETT, Mike Charles, GB;
FLAVIN, Phil Graeme, GB

(73) Propriétaire/Owner:
BRITISH TELECOMMUNICATIONS PUBLIC LIMITED COMPANY, GB

(74) Agent: GOWLING LAFLEUR HENDERSON LLP

(54) Titre : RECUPERATION D'INFORMATIONS DANS UNE BASE DE DONNEES CACHE
(54) Title: INFORMATION RETRIEVAL IN CACHE DATABASE



(57) Abrégé/Abstract:
The invention provides an information access system for downloading information using a communications network (110) such as the Internet. The system downloads pages at user request to local storge (120) and at the same time reviews the pages for embedded HTML links. It then also downloads the pages identified by the embedded links to a secondary, "lookahead" local cache (125). The invention, referred to as "Casper" (Cached Access to Stored Pages with Easy Retrieval), offers a reduction in accessing times to the user and also a reduction in network traffic.

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 17/30

A1

(11) International Publication Number: WO 97/44747

(43) International Publication Date: 27 November 1997 (27.11.97)

(21) International Application Number: PCT/GB97/01363

(22) International Filing Date: 20 May 1997 (20.05.97)

(30) Priority Data:

9610505.1	20 May 1996 (20.05.96)	GB
96303645.4	22 May 1996 (22.05.96)	EP

(34) Countries for which the regional or international application was filed: GB et al.

(71) **Applicant** *(for all designated States except US)*: BRITISH TELECOMMUNICATIONS PUBLIC LIMITED COMPANY [GB/GB]; 81 Newgate Street, London EC1A 7AJ (GB).

(72) **Inventors; and**

(75) **Inventors/Applicants** *(for US only)*: WEEKS, Richard [GB/GB]; 44 Glemsford Close, Felixstowe, Suffolk IP11 8UG (GB). STEPHENS, Lee, Michael [US/US]; 22 Haughley Drive, Rushmere St. Andrew, Ipswich, Suffolk IP4 5QT (US). DAVIES, Nicholas, John [GB/GB]; 10 Spindle Wood, Colchester, Essex CO4 4SX (GB). REVETT, Mike, Charles [GB/GB]; Kingston Rise, Broomheath, Woodbridge, Suffolk IP12 4DL (GB). FLAVIN, Phil, Graeme [GB/GB]; 8 Westmorland Road, Felixstowe, Suffolk IP11 9TB (GB).

(74) **Agent:** DUTTON, Erica, Lindley, Graham; BT Group Legal Services, Intellectual Property Dept., 8th floor, 120 Holborn, London EC1N 2TE (GB).

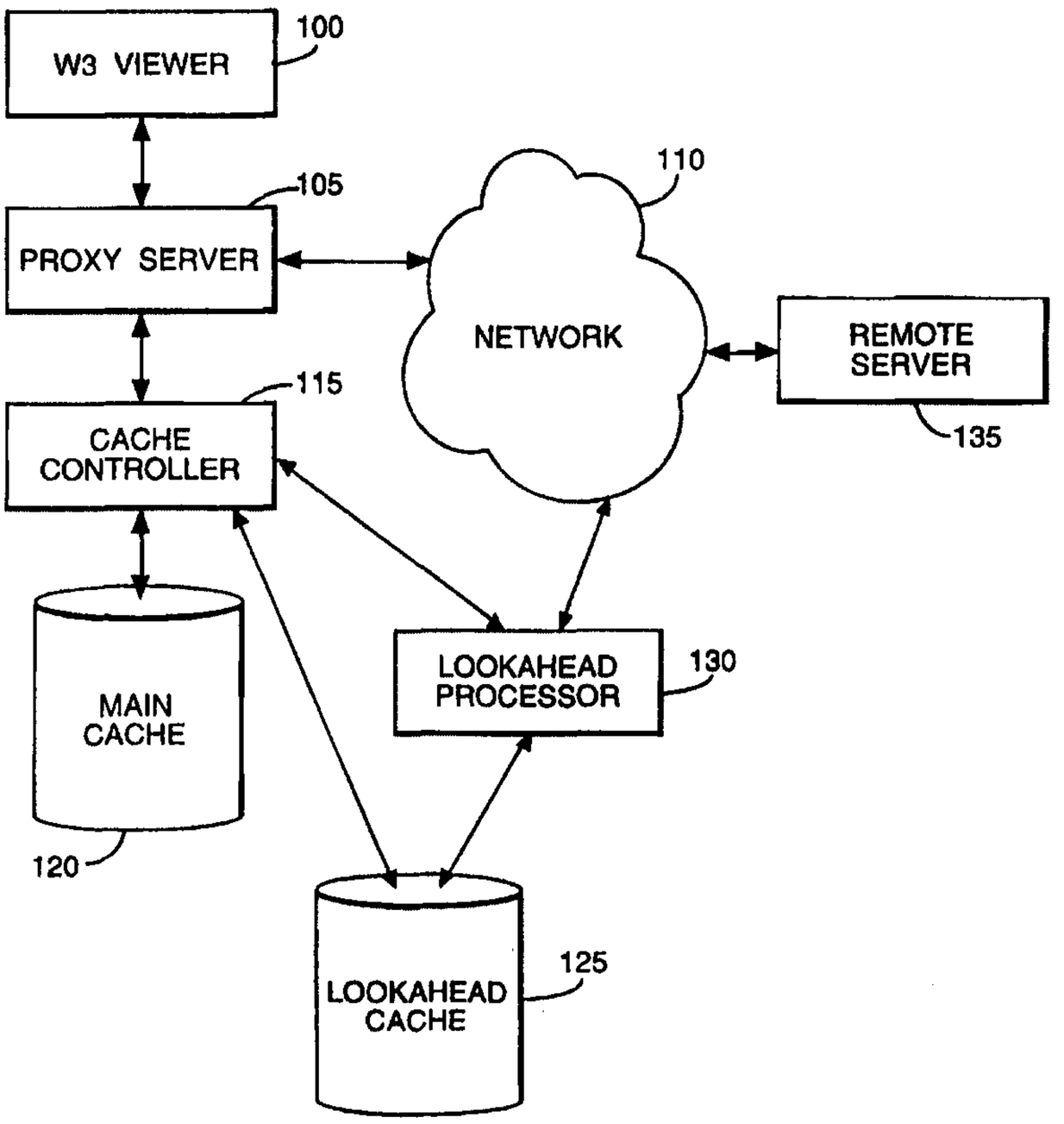
(81) **Designated States:** AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) **Title:** INFORMATION RETRIEVAL IN CACHE DATABASE

(57) **Abstract**

The invention provides an information access system for downloading information using a communications network (110) such as the Internet. The system downloads pages at user request to local storge (120) and at the same time reviews the pages for embedded HTML links. It then also downloads the pages identified by the embedded links to a secondary, "lookahead" local cache (125). The invention, referred to as "Casper" (Cached Access to Stored Pages with Easy Retrieval), offers a reduction in accessing times to the user and also a reduction in network traffic.



INFORMATION RETRIEVAL IN CACHE DATABASE

The present invention relates to methods and/or systems for accessing information by means of a communication system.

The Internet is a known communications system based on a plurality of separate communications networks connected together. It provides a rich source of information from many different providers but this very richness creates a problem in accessing specific information as there is no central monitoring and control.

In 1982, the volume of scientific, corporate and technical information was doubling every five years. By 1988, it was doubling every 2.2 years and by 1992 every 1.6 years. With the expansion of the Internet and other networks the rate of increase will continue to increase. Key to the viability of such networks will be the ability to manage the information and provide users with the information they want, when they want it.

Navigating the information available over the Internet is made possible by the use of browsers and languages such as Hypertext Markup Language (HTML). For instance, the familiar Worldwide Web (WWW) is an area of the Internet which can be browsed using Hypertext links between documents.

In co-pending international patent application number WO 96/23265 dated August 1, 1996, there is disclosed a system for accessing information, for instance by means of the Internet, based on a community of intelligent software agents which store meta information about pages on the Internet. The agent-based access system uses keyword sets to locate information of interest to a particular user. It also stores user profiles such that pages being stored by one user can be notified to another whose profile indicates potential interest.

According to a first aspect of the present invention, there is provided an information access system, for accessing information accessible by means of a communications network, the access system comprising:

a) means for downloading items of information from the network to local storage; and

b) an information processor for processing items of information to locate embedded links to other items of information accessible over the network;

wherein said information processor comprises:

i) a queuing arrangement for queuing items of information having such links embedded; and

ii) a set of concurrently activatable information acquisition units

the processor, in use, processing items of information to identify links embedded therein which identify information for downloading, and allocating identified links to respective available aquisition units, each aquisition unit then acting to download an item of information identified by a link allocated to it, from the network to the local storage.

An advantage of embodiments of the present invention is that delays in accessing information pages can be reduced. It can also be the case that network traffic is reduced since pages are only retrieved once across the network. Subsequent access by a user can be to the local storage.

Preferably, the local storage comprises more than one cache data store, a first cache data store holding items of information which have been retrieved by the system and a second cache data store holding items of information transferred out of the first cache data store when a user has requested access. The second cache data store can then be managed in a different way from the first, for instance to provide information which is accessed relatively frequently by users. The first cache data store may for instance be significantly smaller and be "pruned" to hold only information downloaded for links embedded in pages very recently requested by users.

In general, embodiments of the present invention will need monitoring and scheduling capabilities in order to control the queueing process. Otherwise, the queue may become a processing bottleneck.

According to a second aspect of the present invention, there is provided a method of storing items of information accessible by means of a communications network, which method comprises the following steps:

i) processing an item of information to locate one or more links to other items of information embedded therein, to be downloaded; and

ii) downloading said one or more other items of information by means of said network to a local data store.

An information access system according to an embodiment of the present invention will now be described, by way of example only, with reference to the accompanying Figures, wherein:

Figure 1 shows an environment in which the information access system is supported;

Figure 2 shows a flow diagram of the overall operation of the information access system;

Figure 3 shows a block diagram of the components of a lookahead processor in the information access system of Figure 1;

Figure 4 shows a flow diagram for the process that is provided by the lookahead processor of the information access sytem;

Figure 5 shows a processing queue and the relevant information acquisition units;

Figure 6 shows the queue and units of Figure 5 after a time interval has passed; and

Figure 7 shows an alternative flow diagram with respect to that of Figure 2.

The specific embodiment of the present invention described below is referred to herein as "Casper", standing for "Cached Access to Stored Pages with Easy

Retrieval". The Casper system is particularly designed for accessing WorldWide Web pages provided over the Internet Global Communications network. Other embodiments of the invention of course could however be used to access other information systems where data units ("items of information") have embedded links to other data units.

Information accessible by means of the Web is provided as pages in HTML. Within a document, strings of words or other identifiers may be highlighted. If the user, while viewing a document, selects a highlighted string of words or an identifier, and clicks on it using the mouse button, the highlighted text provides a link to another document. Clicking on the highlighted text triggers the system into calling up the relevant document over the Internet for viewing on the user's screen. It replaces the document the user was previously viewing.

In the above mentioned co-pending patent application, a system is described which can be used to store meta information about pages selected over the Internet by clicking on Hypertext links. By using keywords sets, the system alerts other interested users in a user group of a new document for which meta information has been stored.

In a Casper system, according to an embodiment of the present invention, when a page is selected, for instance to have its meta information stored, the system will automatically review that page for HyperText links to other documents available over the Internet. If there are such links in the document, and the linked documents are not already stored locally, then the Casper system enters the new page to a page processing queue. The links from the page are then allocated to page acquisition software units, as they become available. The linked document for each link in turn is then read into a local "lookahead" cache datastore.

This has the effect of putting together a local store of pages which are clearly related to pages that the user has an interest in. When the user decides to go beyond a page originally accessed, they can simply get the related pages from

local data cache, rather than calling in links on the Internet. This makes information retrieval faster for the user and reduces traffic on the Internet.

Referring to Figure1, the hardware/software environment which will support embodiments of the present invention comprises the following:

i) a WWW viewer 100 connected via a proxy server 105 to a network such as Internet 110.

ii) main and "lookahead" caches 120, 125 and an associated cache controller 115.

iii) a "lookahead" processor 130 connected between the cache controller 115 and the "lookahead" cache 125, and having direct access to the Internet 110.

The environment supporting Casper is generally of known type. The WWW viewer 100 provides browsing capability in respect of the Internet 110 via the proxy server 105. The proxy server 105 is a known type of software system which can intercept requests for information from users' browsers and process them before passing them on to the information source on the Internet. The main and lookahead caches 120, 125 are local data stores for storing WWW pages and the cache controller 115 provides an interface to them which writes and retrieves pages to and from the caches and logs their contents.

The areas where Casper differs primarily is in the provision of the second cache, the lookahead cache 125, and in the provision of the lookahead processor 130.

It is known to use a caching server, the cache being a temporary store in which for instance pages most recently accessed by a group of users are held. Once the cache is full, pages are removed as more recent ones are added. In Casper, there is the main cache 120 which stores pages accessed by users. There is also however the lookahead cache 125 which stores pages which havn't themselves been accessed but which are linked to pages which have been accessed by users. This lookahead cache 125 is filled by taking the links embedded in pages requested by users and downloading the pages for those links.

These caches 120,125 then come into use when a user requests a page. Casper uses the proxy server 105 to intercept the request in order to make a check first as to whether the page is already in the main cache 120 or the lookahead cache 125. If the page is already present in one of the caches 120, 125, Casper retrieves it from the cache. Otherwise, the relevant information source on the Internet must be contacted after all and the page requested from a remote server 135.

Referring to Figure 2, a flow diagram for the above comprises the following steps:

STEP 300: the proxy server 105 of the Casper system receives a user request 300 involving a Universal Resource Locator (URL) at a remote site on the Internet such as a remote server 135. The request may be for one of several alternatives. For instance, the user may be requesting to view a page and/or may be requesting reloading of a page from the relevant remote server 135 for updating purposes. Alternatively, the user may be calling a program which runs dynamically and for which caching would be inappropriate. Where a request for a page is made, it may incorporate an indication that the cached version of the page is unacceptable and that therefore the system will have to deliver the page from the originating remote server 135.

STEP 305: in order to determine the nature of the request, and any relevant constraints, the proxy server 105 reviews the user request. The user request can contain optional constraints selected by the user, such as "reload" which means a cached version of a file is not acceptable, but it may also contain embedded constraints, for instance in relation to a URL which it contains. Casper is provided with a configuration file that can specify:

- URLs to be outlawed, for instance because they have dubious content
- URLs that themselves force a network reload, for instance because their content is known to change frequently or because the server involved is local or fast and caching is therefore inappropriate

The request review can therefore take the form of a series of checks such as the following:

- does URL force reload?
- is URL a permitted location?
- has user specified reload?
- has user otherwise specified that a cached version is unacceptable?

Depending on the outcome, the proxy server will move to either of **STEPS 310** or **330**. If the user has clicked on "reload", or the URL forces "reload", then the "cached version not acceptable" path to **STEP 310** is followed. If the URL is not a permitted location, then an "access denied" message is returned to the user and no further processing takes place. Otherwise, the "cached version acceptable" path to **STEP 330** is followed.

STEP 310: if at **STEP 305**, it is found that a cached version is not acceptable, the proxy server 105 initiates a connection with the relevant remote server 135. As seen in Figure 1, the proxy server 105 has a direct interface to the Internet 110. In **STEP 315**, the proxy server 105 checks whether the subject of the request exists. If not, in **STEPS 320, 355**, it generates and delivers a message to the user appropriately.

STEP 323: if the subject of the request exists, the proxy server 105 checks whether it should be cached. If the subject of the request exists, is an HTML page but caching is not relevant, for instance where Casper's configuration file defines the URL as a forced reload location, then the proxy server will go to **STEPS 350, 355** to trigger the lookahead processor and send the page to the user's browser. If the subject of the request exists, is an html page and caching is relevant, the proxy server 105 moves to **STEP 325** and writes the relevant page to the main cache 120. The proxy server 105 will then go to **STEPS 350, 355** as above.

It may be the case that neither caching nor lookahead processing is actually relevant, for instance in the case of a program which runs dynamically. In that case, the proxy server 105 can go to **STEPS 350, 355** as above, and the

lookahead processor 130 will not find any embedded links to process. On the other hand, an extra check can be made (not shown in Figure 2) as to whether lookahead processing is relevant. If not, then the proxy server 105 could leave out **STEPS 350, 355** and simply deliver the program to the user's browser 100.

STEP 330: if at **STEP 305** it is found that caching is relevant, that the user has not specified reload or that a cached version is unacceptable, then the proxy server 105 makes a check via the cache controller 115 whether the subject of the request, usually a Web page, is in the main cache 120. If it is, then the proxy server is ready to go to **STEP 345.**

If a requested Web page is not in the main cache 120, then the proxy server 105 makes a check via the cache controller 115 whether it is already in the lookahead cache 125. If it is, the proxy server 105 moves to **STEP 340** and transfers the page to the main cache 120 and is ready to go to **STEP 345.**

If the requested Web page is not in the lookahead cache 125 either, then the proxy server 105 moves to **STEP 310** and follows the process described above for **STEP 310.**

STEP 345: the proxy server fetches the requested page from the main cache 120 and moves to **STEP 350**, triggering the lookahead processor 130. After triggering the processor, the proxy server 105 moves to **STEP 355** where it adds Casper page wrappers to the requested page(s) and sends them to the user's browser 100.

Meanwhile, as further discussed below, the lookahead processor 130 reviews the requested page(s) to identify embedded links and to access and download to the lookahead cache 125 the pages identified by the embedded links.

Referring to Figure 3, the lookahead processor 130 comprises software processes and memory, running on a Unix machine. It comprises three asynchronous processes, a page processor 255, a slot processor 260 and a shelf processor 265.

The page processor 255 has a message input 200 for receiving messages that will for instance contain the URL of a page to be processed, an HTML parser 205 for retrieving the contents of a URL and parsing them in sufficient detail to identify all the HTML links embedded, a link assessor 210 for determining which of the identified links need to be "pre-fetched", and a page queue 215 which stores page data for processing as processing requests come in to the message input 200. The parser 205 and the link assessor 210 both have access to the main cache 120, and the link assessor 210 has access to the lookahead cache 125. There is a timer 235 associated with the page queue 215 which marks as "ignore" any links which have been waiting too long (a configurable parameter) for processing. This provides a mechanism for moderating the length of the page queue.

The slot processor 260 has the overall purpose of processing the page data in the page queue 215 as quickly as possible. To that end, it is capable of running a number of Unix sub-processes in parallel, by means of a plurality of aquisition slots 225. It is provided with a slot filler 220 which monitors the status of the slots 225 and attempts to populate any free slots from the page queue 215. Each time a free slot is newly filled, a connect processor 230 is fired off as a Unix sub-process to attempt a connection to the server indicated by a URL in an embedded link. If successful, a read processor 240 reads the relevant page from the server and passes it to a cache writer 245, part of the cache controller 115, which writes the data to the lookahead cache 125 and notifies the shelf processor 265. Both the connect and read processors 230,240 are provided with timing means 235 to avoid aquisition slots 225 being tied up inefficiently.

The shelf processor 265 limits the size of the lookahead cache 125 by removing files after a length of time. It maintains a list of files sent to the lookahead cache 125 by entering file identifiers in time slots in a data store 250 called the "timeout shelf". It is provided with a clock 235 which effectively shifts the time slots through the data store, each time slot triggering deletion of all its associated files from the lookahead cache 125 when it reaches the end of the data store.

The maximum permissible length of the page queue and the number of aquisition slots are intended to be configurable in order to meet operating conditions. To dimension a lookahead processor 130, the considerations discussed below are relevant.

Concerning page queue length, there's no serious penalty in software terms in making the page queue length arbitrarily long. Curtailing its length does however give a measure of control over system performance. The reasoning behind letting pages drop off the end of the queue is that where pages which have got to the end of the queue without being fully processed, the user is anyway likely to have lost interest by that time. There is therefore no point in processing those pages further.

A possible initial strategy would therefore be to start with a short page queue and increase its length until it only rarely overflows. However, if pages are remaining on the queue for too long, more than a minute say, this may be an indication that queue length should be reduced. Alternatively, the number of aquisition slots should be increased; see below.

The matter is complicated somewhat by the fact that there can be two different modes of operation of the lookahead cache, depending on the length of the "shelf", ie the permitted lifetime of pages in the lookahead cache.

If the only purpose of lookahead caching is to give users increased speed of response during their immediate browsing sessions, then there is no point in keeping pages in the cache for more than a few minutes. If pages are kept in the lookahead cache for considerably longer, a few days say, then the possibility arises that the cache will be able to fulfill a second demand for a page. Operating the lookahead cache in this manner would change the above argument for determining page queue length in favour of having a longer queue, since there is value in pre-fetching pages even if it takes several minutes.

With regard to the number of aquisition slots, at the simplest level then the more slots the better. If there are always free aquisition slots, pages can be fetched with the minimum of delay.

In practice, a Unix machine will place a limit on the number of sub-processes permitted. Within this limit, a large number of sub-processes, over twenty say, can be run, since none uses much cpu (central processing unit) or memory resource. Most of the sub-processes' "lives" are spent simply waiting for data to become available.

If the lookahead load is more than one machine can handle, multiple machines can of course be used.

An alternative to multiple sub-processes would be to have multiple threads running within the lookahead processor. This approach would require less operating system overhead and possibly operate more efficiently, but could make maintenance more difficult. The lookahead processor would then be implemented all in one program, rather than in separately maintainable parts. Reliability of the lookahead processor could also be compromised: the multiple sub-process arrangement is tolerant of failure within any of the sub-processes- such failures do not affect the lookahead processor itself which must after all run reliably and continuously.

Referring to Figure 4, a flow diagram through the basic operation of the "look ahead processor" can be described as follows:

STEP 400: The proxy server 105 provides a page to the user, either from its main cache or from the Internet, and instructs the lookahead processor 130 to handle the page by transmitting a message to the processor containing the URL for the relevant page.

STEP 405: The lookahead processor 130 gets the URL's contents from the main cache 120 and parses the page's contents to determine the links to other pages.

(That is, it interprets the page's HTML to extract information about the linked pages. This of course requires programmed-in knowledge of HTML syntax.)

STEP 410: The existence of the child links so determined is tested against the main cache 120 and the lookahead cache 125.

STEP 415: If a child link is not already cached, and it is not a page or a child page that is already being processed, it is added to the page's child list.

STEP 420: If the page to be processed is determined to have any child links that need to be retrieved, the page is added to the head of the page processing queue. Each entry in the page processing queue 215 has associated with it the page's URL and, for each of the child links, a link URL and link status. This last will be selected from "ignored" if rejected by the link assessor 210, "done" if fully processed, "aborted" if timed out or a slot number if currently processing.

STEP 425: If the page processing queue 215 has grown too long, indicating a processing bottleneck, the last page in the queue is removed, and any outstanding aquisitions associated with its children are aborted.

STEP 430: An attempt is made to process each child of the new page, and any still unprocessed children of other pages in the page queue, by assigning them to inactive page acquisition slots. The slot filler 220 monitors the status of the aquisition slots 225, each of which can have a status of "free", "connecting" or "reading", and populates free slots from the page queue 215, starting with the first unprocessed link in the most recently added page.

STEP 435: Newly-filled page acquisition slots are activated and a "connect processor" 230 is fired off as a Unix sub-process as described above.

STEP 440: Any page acquisition slots that have spent too long attempting to connect, or fail for some other reason, are released and marked inactive, the slot filler 220 being notified.

STEP 445: Any page acquisition slots that have successfully connected to servers 135 are updated from a "connect" state to a " read" state. Reading is achieved by a read processor 240, using the same Unix sub-process as the connect phase.

The slot processor 130 reads the relevant child link's data from the network and passes it to a cache writer in the cache controller 115. This writes the data to the lookahead cache 125 and notifies the shelf processor 265.

STEP 450: Any page acquisition slots 225 that have spent too long attempting to read from the servers are released and marked inactive, again notifying the slot filler 220 as in the connect phase.

STEP 455: Pages that have been in the look ahead cache for too long are deemed unwanted and removed from the cache. This is practical since any files which have been accessed by a user after pre-fetching will have been transferred to the main cache 120 by a separate cache controller exercise. Although simplest to trim the lookahead cache 125 by regularly scanning for, and removing, outdated files, this approach consumes significant computing time and incurs high amounts of disc access. It is therefore preferred to use a shelf processor 265 as described above.

The above process steps repeat indefinitely, although not necessarily in the order described. Clearly, steps 440, 445, 450 and 455 in particular can be carried out at a frequency found appropriate in relation to the rest of the process and these steps are shown in dotted outline to indicate they may not be present each time that the lookahead processor 130 is instructed to handle a page.

Referring to Figure 5, a real time view of the system could be as shown. In the example, there are four pages 500 in the queue 215 and there are ten individual acquisition slots 505. Assuming that a new page (page 1) has just arrived, and that some acquisition slots 505 have just become free because processing of some children of page 4 has just completed, then some re-allocation of acquisition slots 505 is needed. Page 2 has two of its children being processed. Its other two children are ready for processing but the arrival of the new page 1 may affect what will happen to them. Page 3 has four children processing, one connecting, three already connected and already being read. Page 4 is complete, all its children having either been read or aborted because they were too slow.

The lookahead processor 130 will now assign the children of the newly arrived page 1 to slots 2, 6 and 7, and a child of page 2 to slot 8. Page 4 will be removed from the queue as it has no outstanding operations.

Referring to Figure 6, the results of the re-allocation can be seen.

Referring to Figure 7, a slightly different version of the present invention might be found preferable, at **STEPS 345, 350**, where the system bypasses the lookahead processor 130 in the event that a page is found already in the main cache. This may be preferred if all pages have already been processed by the lookahead processor before loading to the main cache 120.

As a consequence of the above variation, as shown in Figure 7, the system may move directly to trigger the lookahead processor 130, **STEP 350**, after transferring a page from the lookahead cache 125 to the main cache 120. In this case, at the same time as transferring the page to the main cache, the system will also have to supply the page to the lookahead processor 130.

It would be possible to design embodiments of the invention in object-oriented technology. For instance, at the highest level, the three processors, the page, slot and shelf processors, can be regarded as objects. Each can be an independent entity, communicating with the others via messages rather than by direct data access or modification. At a lower level, the elements of the page queue and the aquisition slots have object-like features: each is an instantiation of a prototype (a page queue element or an aquisition slot) and each has associated data and states.

As described, Casper accesses all pages linked to a selected page. It would be possible to use the principles of the co-pending patent application referenced above and to select the pages accessed according to a user profile based on interests and context for instance.

CLAIMS

1. An information access system, for accessing information accessible by means of a communications network, the access system comprising:

a) means for downloading items of information from the network to local storage; and

b) an information processor for processing items of information to locate embedded links to other items of information accessible over the network;

wherein said information processor comprises:

i) a queuing arrangement for queuing downloaded items of information having such links embedded; and

ii) a set of concurrently activatable information acquisition units for downloading items of information identified by said embedded links from the network to local storage

the processor, in use, processing items of information to locate links embedded therein which identify other items of information for downloading, queuing processed items of information, and allocating links located in the processed items of information to respective available acquisition units, each acquisition unit then acting to download an item of information identified by a link allocated to it, from the network to the local storage.

2. An information access system according to Claim 1, which further comprises an input for requests to download items of information.

3. An information access system according to Claim 2, wherein the local storage comprises at least two cache data stores, a first cache data store to which the acquisition units download identified items of information and a second cache data store for storing items of information downloaded in response to a request received at the input.

4. An information access system according to claim 3, the system further comprising means for transferring an item of information from the first cache data store to the second cache data store in response to a download request received at the input and identifying the item of information.

5. An information access system according to any one of claims 2 to 4, wherein each download request contains at least one location indicator, indicative of the location of a relevant item of information in the network, and the system further comprises a register for storing location indicators together with related constraint data, the system responding to a download request at the input by accessing the register to obtain any constraint data stored in relation to a location indicator contained in the request.

6. An information access system according to any one of claims 1 to 5, which further comprises queue length control means for deleting items of information from the queuing arrangement.

7. An information access system according to claim 6 wherein the queue length control means deletes items of information which have been in the queuing arrangement for the greatest length of time.

8. An information access system according to any one of claims 1 to 7, further comprising acquisition unit monitoring means for detecting when units become available and transferring links located in items of information from the queuing arrangement to such units.

9. An information access system according to claim 8 wherein the acquisition unit monitoring means transfers links located in items of information in a priority order, the priority order being determined by the relative lengths of time the items have been present in the queuing arrangement.

10. An information access system according to claim 9 wherein highest priority is given to links located in the item which has been in the queuing arrangement for the least amount of time.

11. An information access system according to any one of claims 1 to 10 wherein the queuing arrangement is arranged to queue each item of information by storing in respect of it:

a) an identifier for the item; and

b) an identifier for one or more links embedded in the item.

12. An information access system according to claim 11 wherein the queuing arrangement is further arranged to store status information for each of the link identifiers, relating to the download status of the link by means of an acquisition unit.

13. An information access system according to any one of claims 1 to 12, the system comprising a proxy server arranged to receive user requests from users' browsers.

14. A method of storing items of information accessible by means of a communications network, which method comprises the following steps:

i) receiving a user request to download an item of information;

ii) downloading and processing the requested item of information to locate one or more links to other items of information embedded therein, to be downloaded; and

iii) downloading said one or more other items of information by means of said network to a local data store.

15. A method according to claim 14, wherein the method is carried out by a proxy server, said proxy server being arranged to receive user requests from users' browsers.

16. A method according to either one of claims 14 or 15 wherein said processing step includes queuing the items of information by means of storing, in respect of each, an identifier for the item and an identifier for each link embedded therein.

17. A method according to any one of claims 14 to 16 wherein said processing step includes allocating queued links to respective acquisition units of a set of concurrently activatable acquisition units, and downloading the items of information associated with the allocated links by means of the acquisition units.

18. A method according to claims 16 and 17 wherein said processing step further includes storing status information in respect of each identifier for the embedded links, indicating its status in respect of downloading by means of the acquisition units.

19. A method according to any one of claims 14 to 18, for use with an information access system according to Claim 2, wherein the local storage comprises at least two cache data stores, a first cache data store to which the acquisition units download identified items of information and a second cache data store for storing items of information downloaded in response to a request received at the input, the method comprising transferring an item of information from the first cache data store to the second cache data store in response to a download request received at the input in respect of the item of information.

Fig.1.

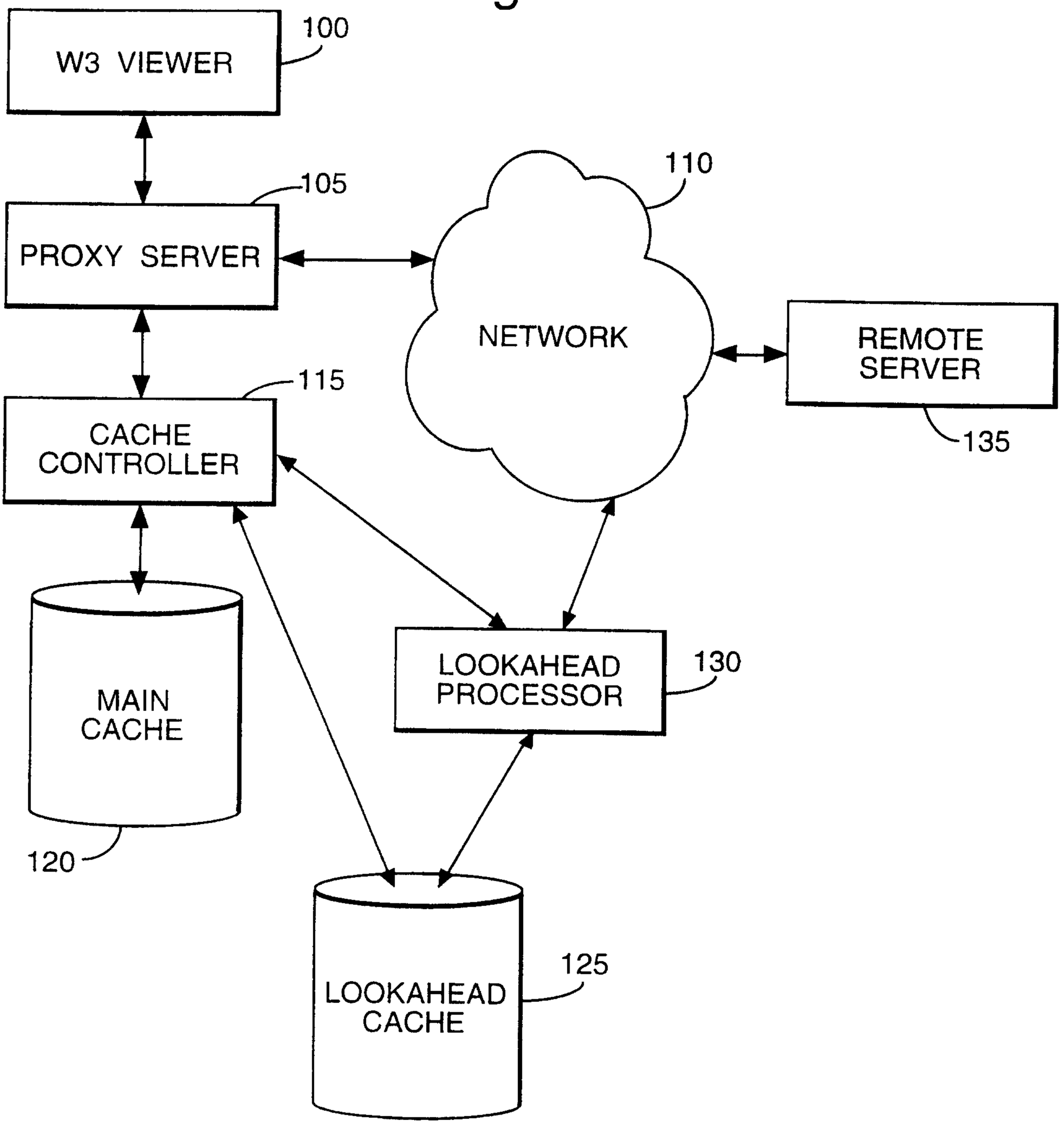

W3 VIEWER
100
PROXY SERVER
105
NETWORK
110
REMOTE SERVER
135
CACHE CONTROLLER
115
MAIN CACHE
120
LOOKAHEAD PROCESSOR
130
LOOKAHEAD CACHE
125

Fig.2.

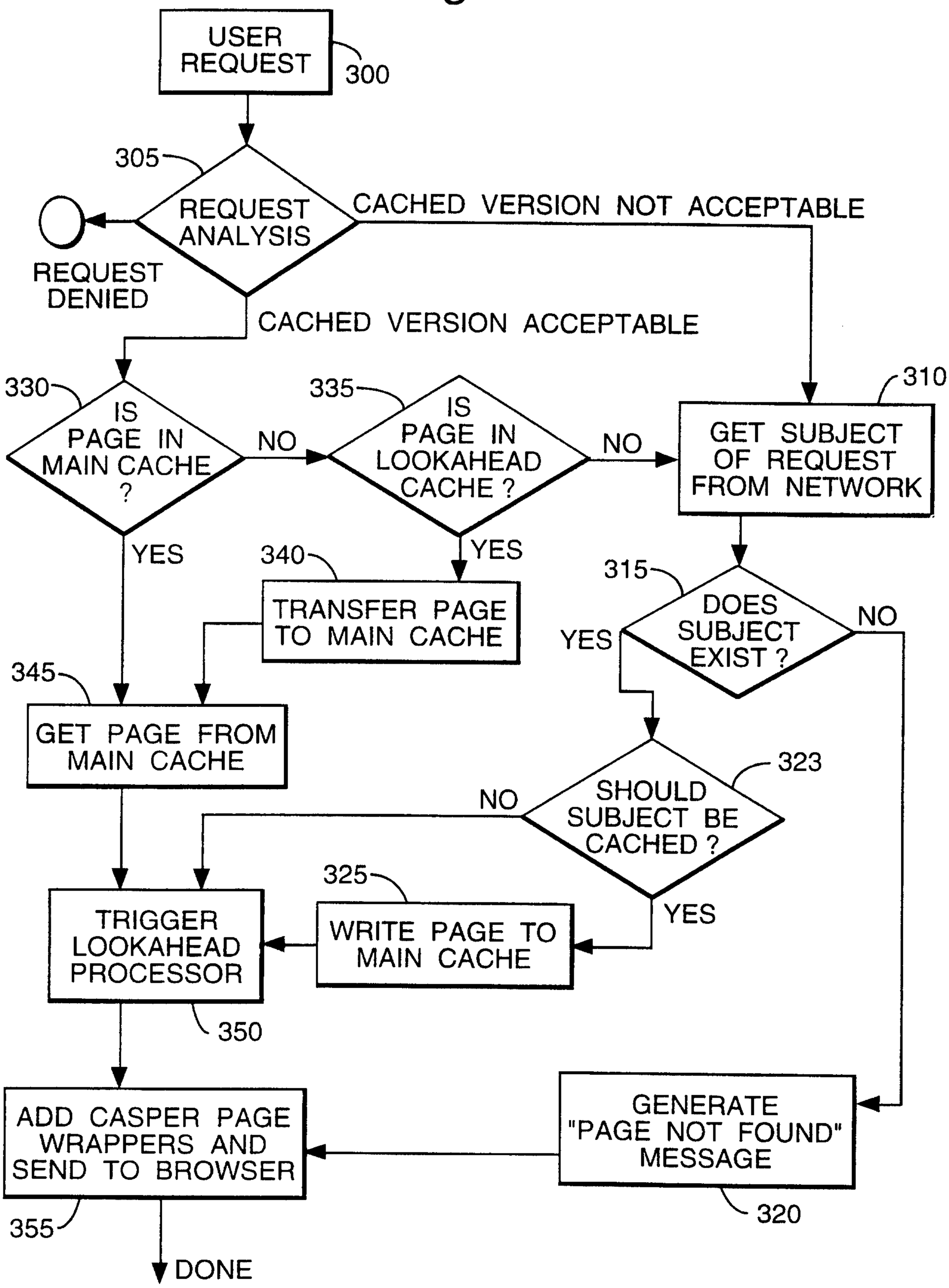

USER REQUEST
300
305
REQUEST ANALYSIS
REQUEST DENIED
CACHED VERSION NOT ACCEPTABLE
CACHED VERSION ACCEPTABLE
330
IS PAGE IN MAIN CACHE ?
NO
335
IS PAGE IN LOOKAHEAD CACHE ?
NO
310
GET SUBJECT OF REQUEST FROM NETWORK
YES
YES
340
TRANSFER PAGE TO MAIN CACHE
315
DOES SUBJECT EXIST ?
YES
NO
345
GET PAGE FROM MAIN CACHE
323
SHOULD SUBJECT BE CACHED ?
NO
YES
325
WRITE PAGE TO MAIN CACHE
TRIGGER LOOKAHEAD PROCESSOR
350
GENERATE "PAGE NOT FOUND" MESSAGE
320
ADD CASPER PAGE WRAPPERS AND SEND TO BROWSER
355
DONE

Fig.3.

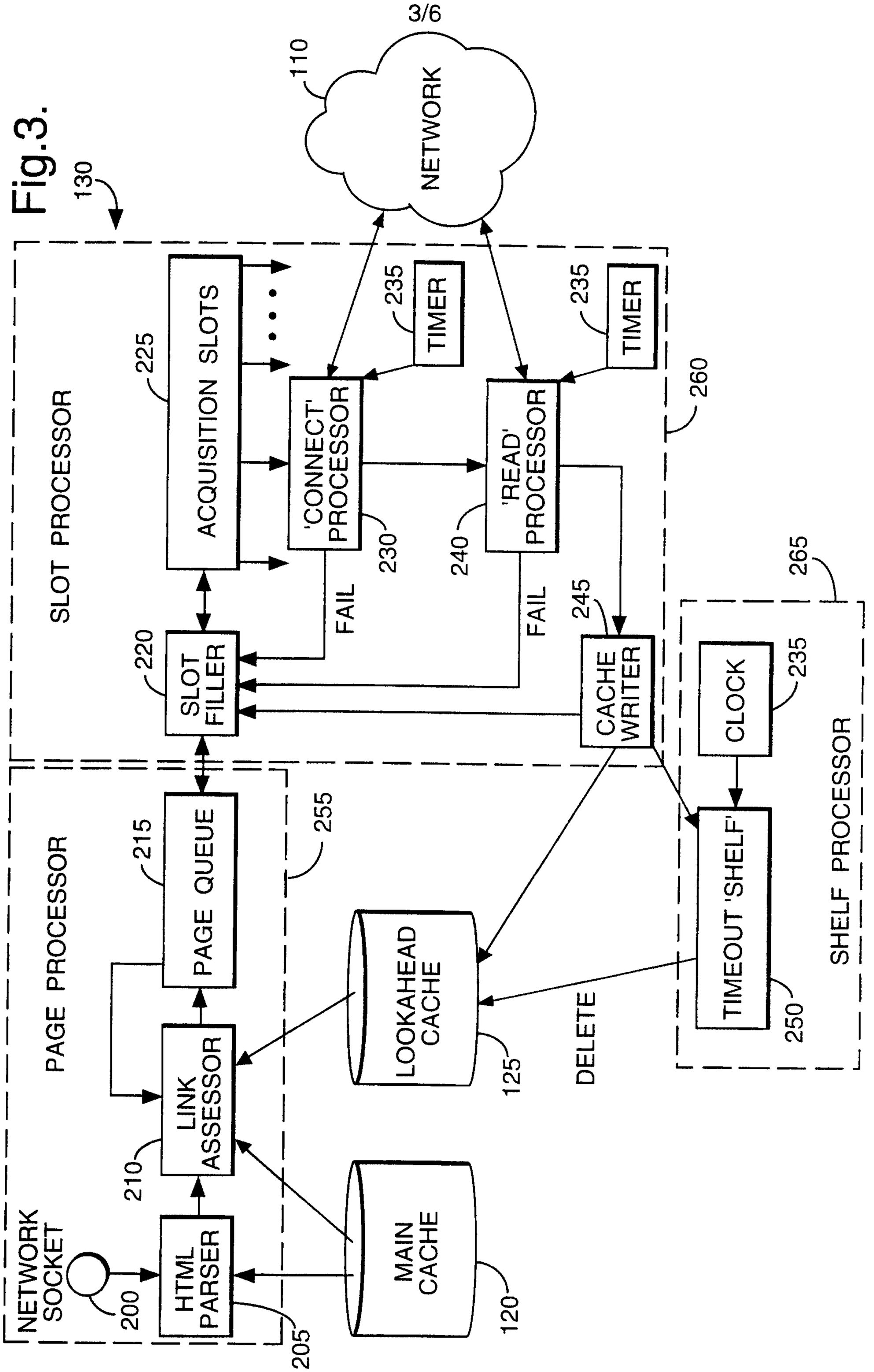

NETWORK
110
130
SLOT PROCESSOR
ACQUISITION SLOTS
225
TIMER
235
TIMER
235
'CONNECT' PROCESSOR
230
'READ' PROCESSOR
240
260
265
FAIL
FAIL
245
SLOT FILLER
220
CACHE WRITER
CLOCK
235
PAGE PROCESSOR
PAGE QUEUE
215
255
LOOKAHEAD CACHE
125
DELETE
TIMEOUT 'SHELF'
250
SHELF PROCESSOR
LINK ASSESSOR
210
NETWORK SOCKET
200
HTML PARSER
205
MAIN CACHE
120

Fig.4.

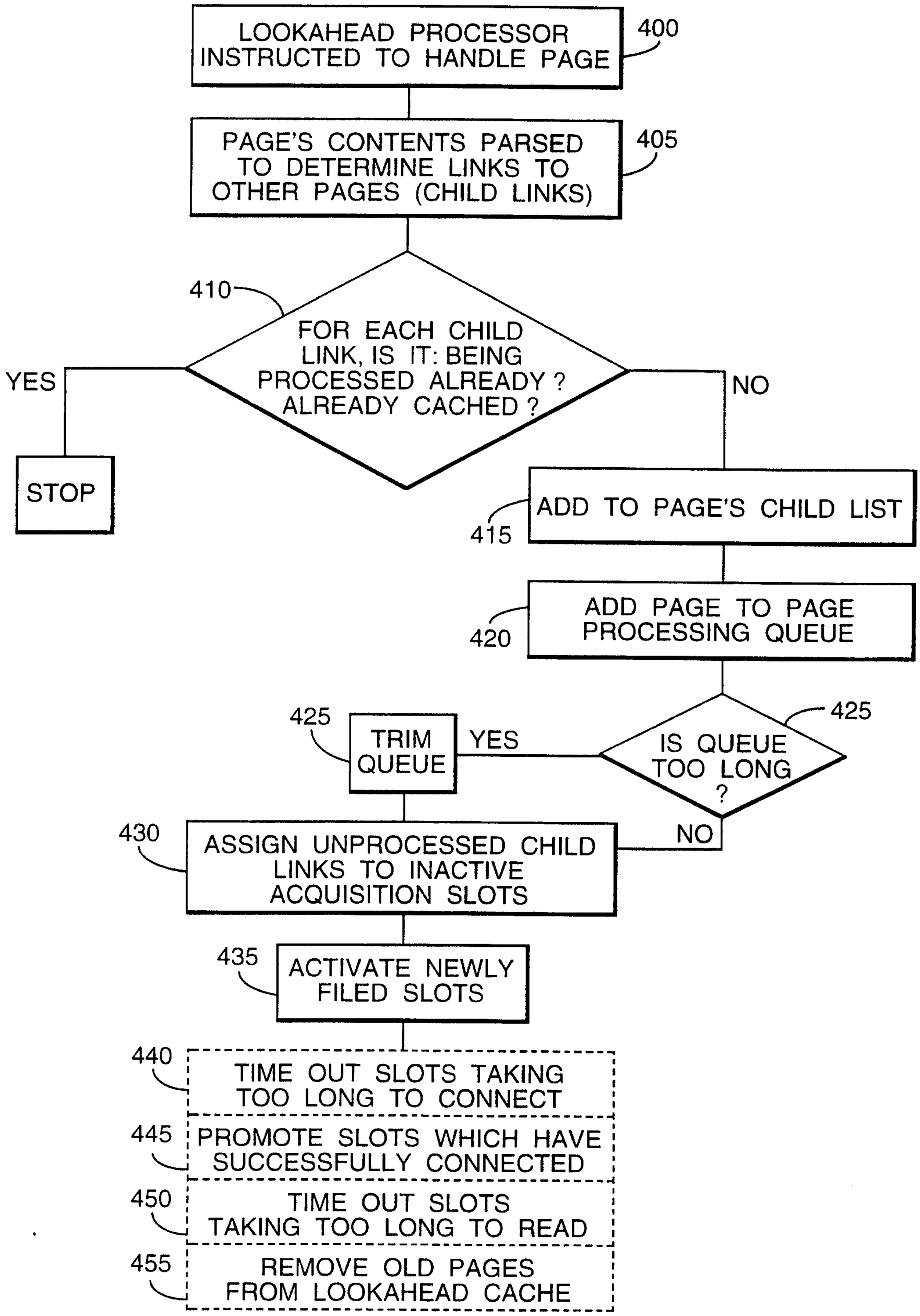
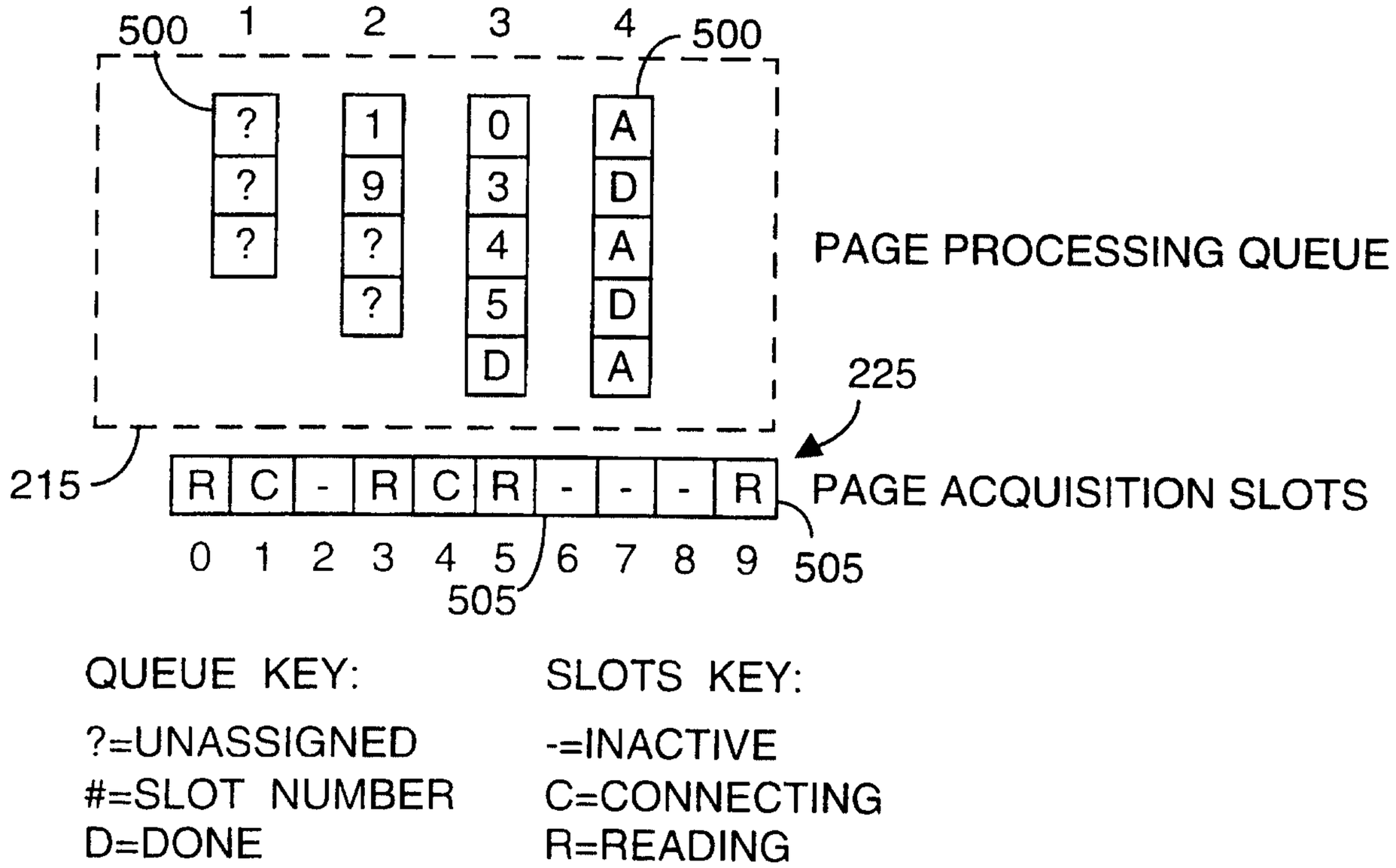

LOOKAHEAD PROCESSOR INSTRUCTED TO HANDLE PAGE
400
PAGE'S CONTENTS PARSED TO DETERMINE LINKS TO OTHER PAGES (CHILD LINKS)
405
410
FOR EACH CHILD LINK, IS IT: BEING PROCESSED ALREADY ? ALREADY CACHED ?
YES
STOP
NO
415
ADD TO PAGE'S CHILD LIST
420
ADD PAGE TO PAGE PROCESSING QUEUE
425
IS QUEUE TOO LONG ?
425
TRIM QUEUE
YES
NO
430
ASSIGN UNPROCESSED CHILD LINKS TO INACTIVE ACQUISITION SLOTS
435
ACTIVATE NEWLY FILED SLOTS
440
TIME OUT SLOTS TAKING TOO LONG TO CONNECT
445
PROMOTE SLOTS WHICH HAVE SUCCESSFULLY CONNECTED
450
TIME OUT SLOTS TAKING TOO LONG TO READ
455
REMOVE OLD PAGES FROM LOOKAHEAD CACHE

Fig.5.



QUEUE KEY:

?=UNASSIGNED
#=SLOT NUMBER
D=DONE
A=ABORTED

SLOTS KEY:

-=INACTIVE
C=CONNECTING
R=READING

Fig.6.

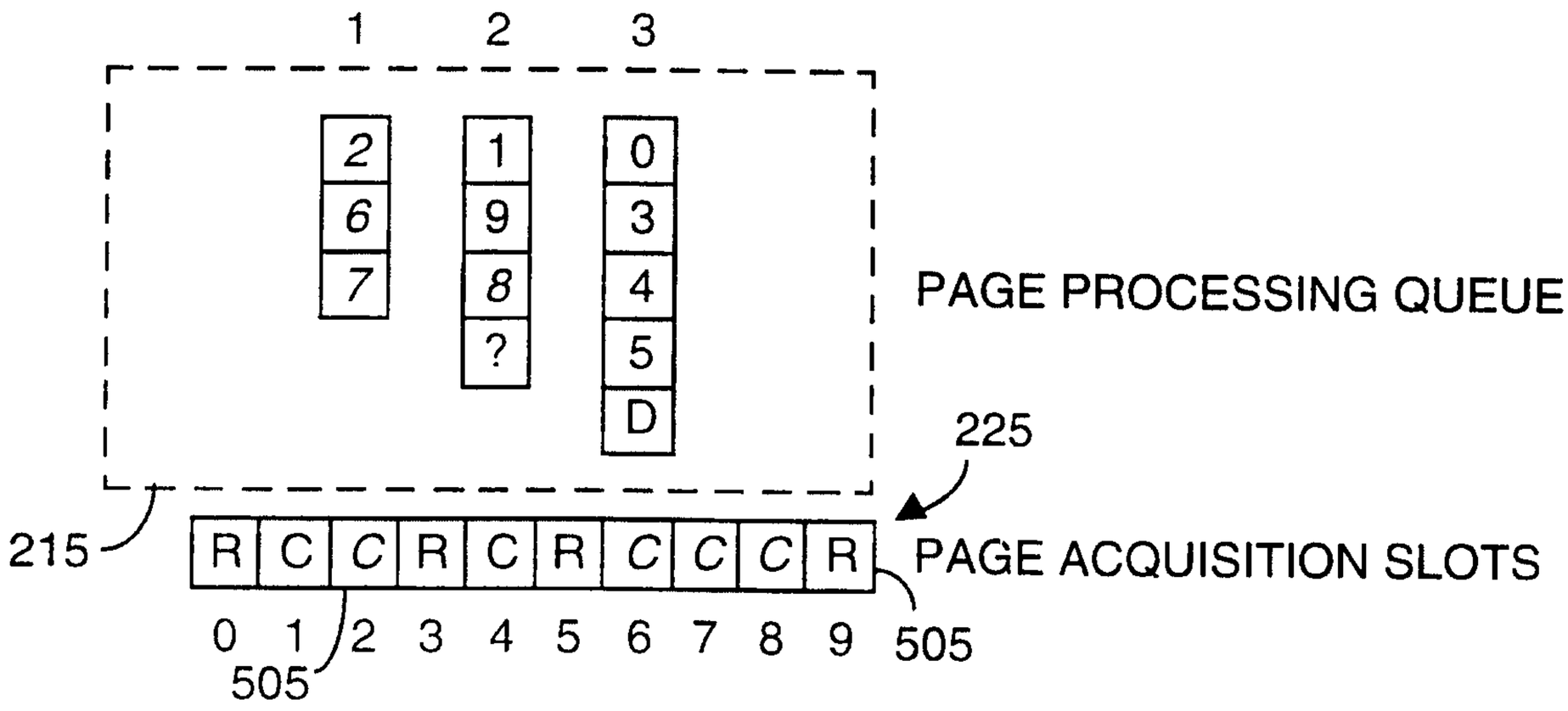
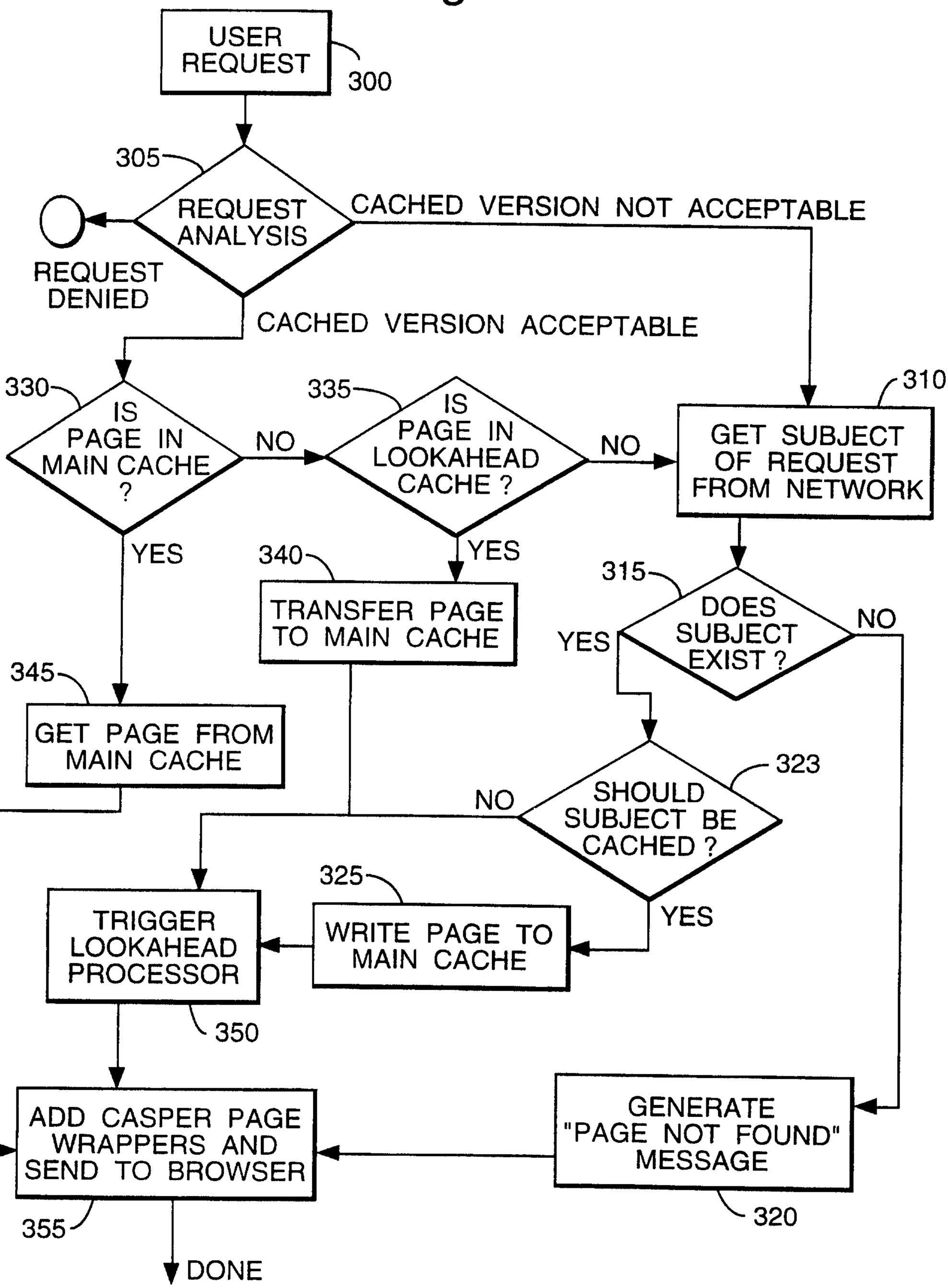
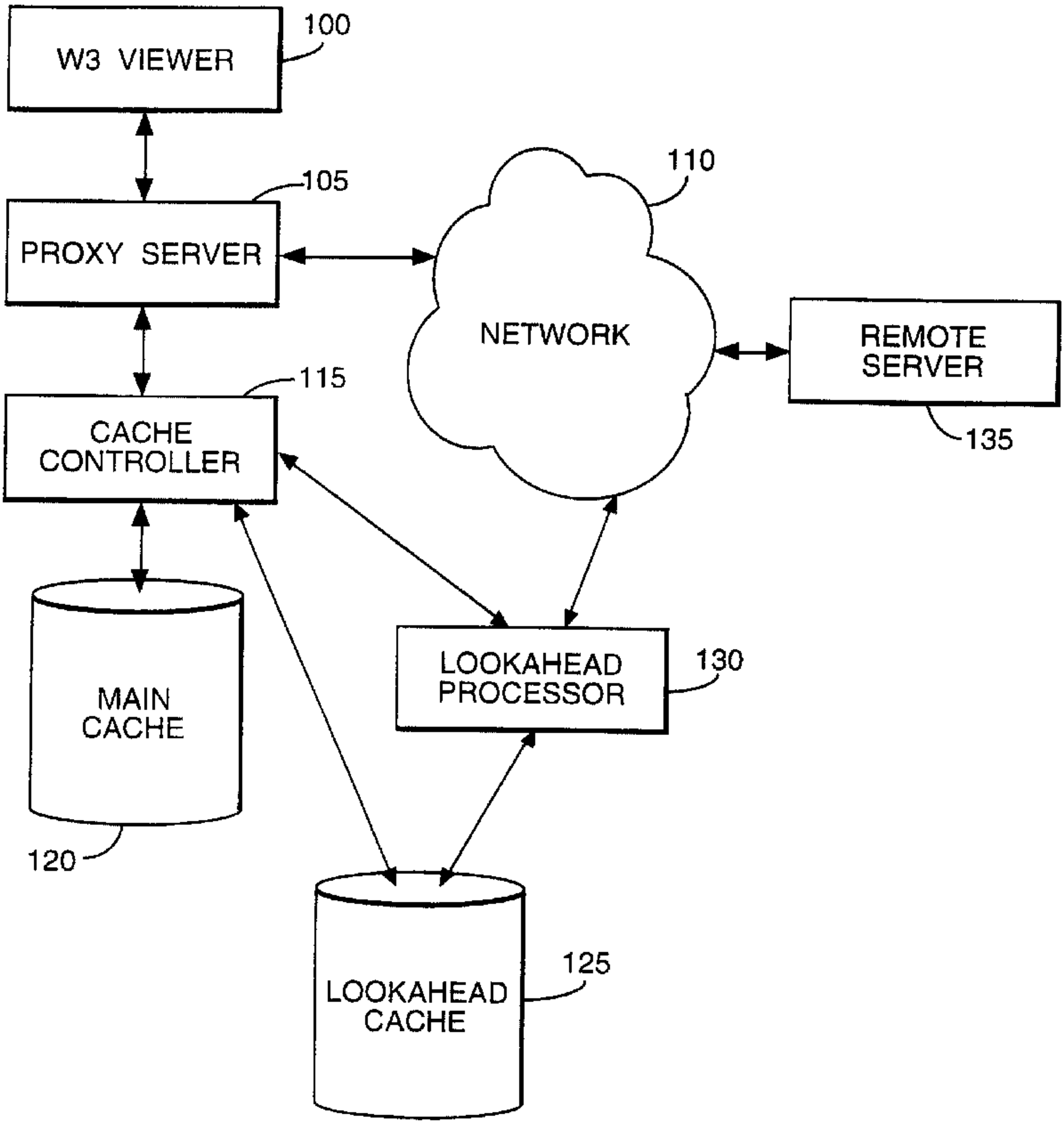


Fig.7.


USER REQUEST
300
305
REQUEST ANALYSIS
REQUEST DENIED
CACHED VERSION NOT ACCEPTABLE
CACHED VERSION ACCEPTABLE
330
IS PAGE IN MAIN CACHE ?
NO
335
IS PAGE IN LOOKAHEAD CACHE ?
NO
310
GET SUBJECT OF REQUEST FROM NETWORK
YES
YES
340
TRANSFER PAGE TO MAIN CACHE
315
DOES SUBJECT EXIST ?
YES
NO
345
GET PAGE FROM MAIN CACHE
323
SHOULD SUBJECT BE CACHED ?
NO
YES
325
TRIGGER LOOKAHEAD PROCESSOR
WRITE PAGE TO MAIN CACHE
350
ADD CASPER PAGE WRAPPERS AND SEND TO BROWSER
GENERATE "PAGE NOT FOUND" MESSAGE
320
355
DONE


W3 VIEWER
100
PROXY SERVER
105
110
NETWORK
REMOTE SERVER
135
CACHE CONTROLLER
115
MAIN CACHE
120
LOOKAHEAD PROCESSOR
130
LOOKAHEAD CACHE
125