Caching approaches for content altering proxies

Duncan Martin

Mark Truran

Helen Ashman

University of Nottingham

Overview

- About caching
- Content altering proxies
- Opportunities for caching
- Storage and processing costs
- Fall back caching and delta storage
- Conclusions

Caching

- A cache is storage quicker than normal reading/writing.
- For networks data is stored ‘locally’ so that it can be quickly retrieved.
- Retrieving from the cache is quicker than retrieving over the network.

Caching example

- Image is requested.
- Image is obtained from remote resource and used, simultaneously it is placed in a cache.
- Next time the image is requested, it is obtained from the cache rather than the remote resource.

Browser caching

- Web browsers often have their own cache both on disc and in memory.
- Browser caching is not considered in this paper.

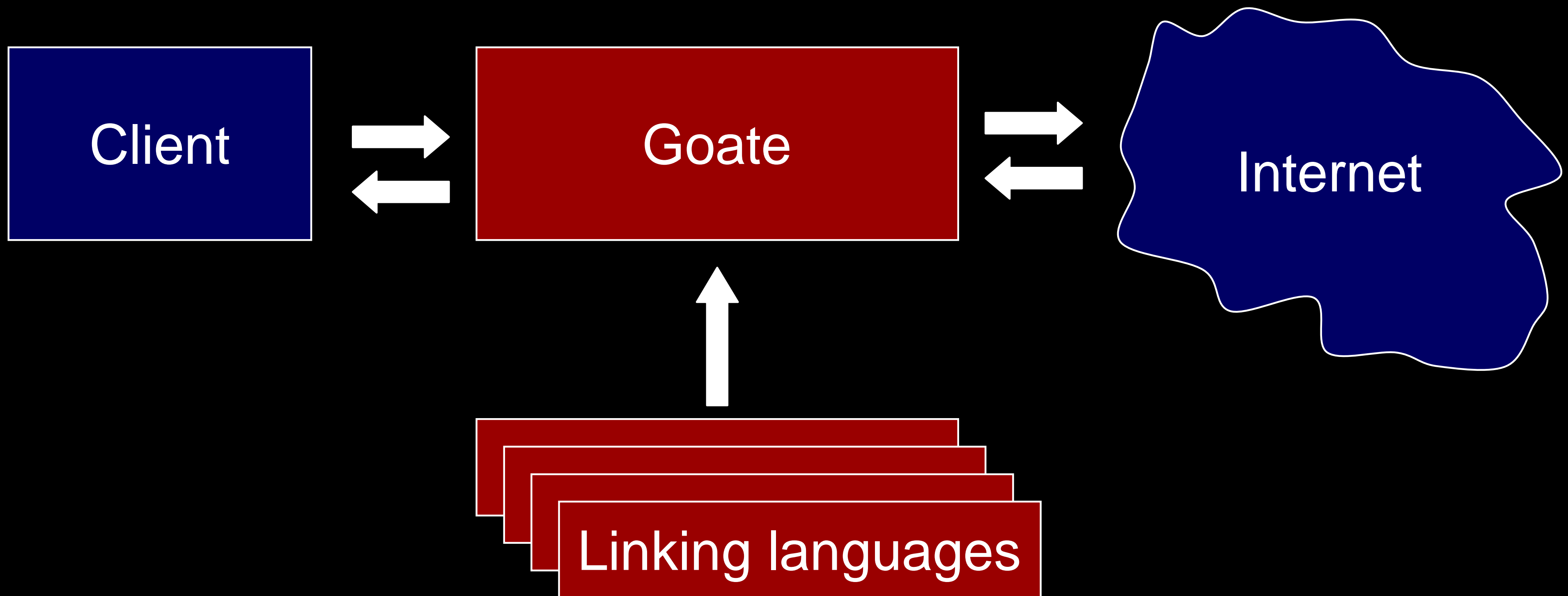
HTTP proxying

- A HTTP proxy sits between the client (browser) and network relaying requests/responses between the two.
- This is done for a number of reasons, including the ability to cache for a number of clients - hence reducing external network traffic.

Content altering proxies

- Normally HTTP proxies pass content verbatim.
- However, systems such as Goate alter the content as it passes through, adding links to the original document.

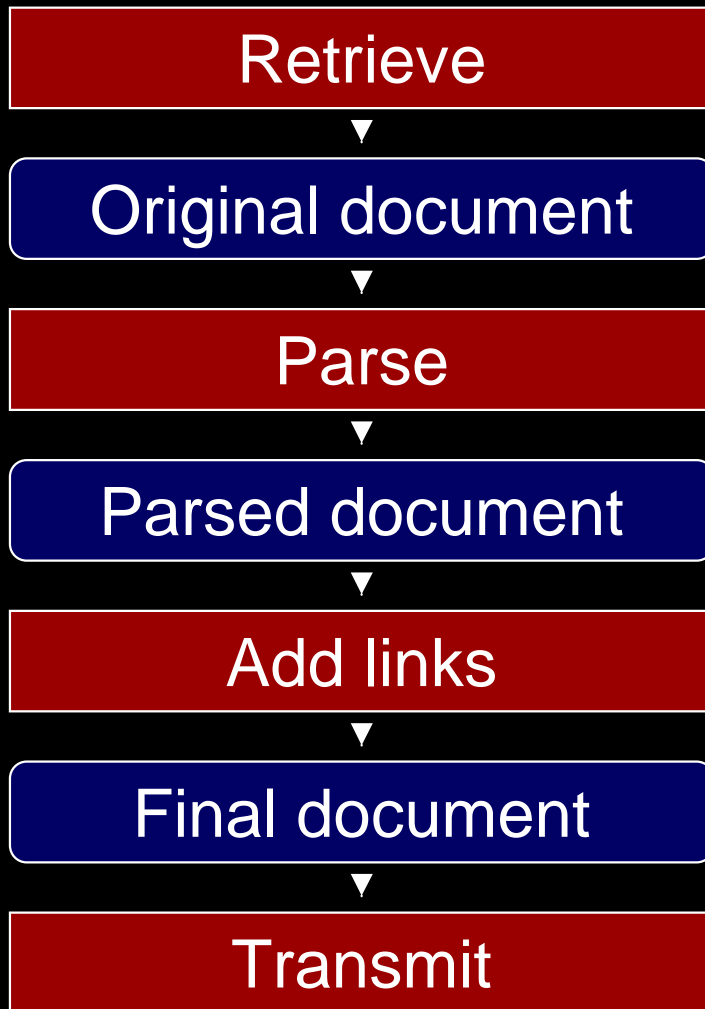
Goate



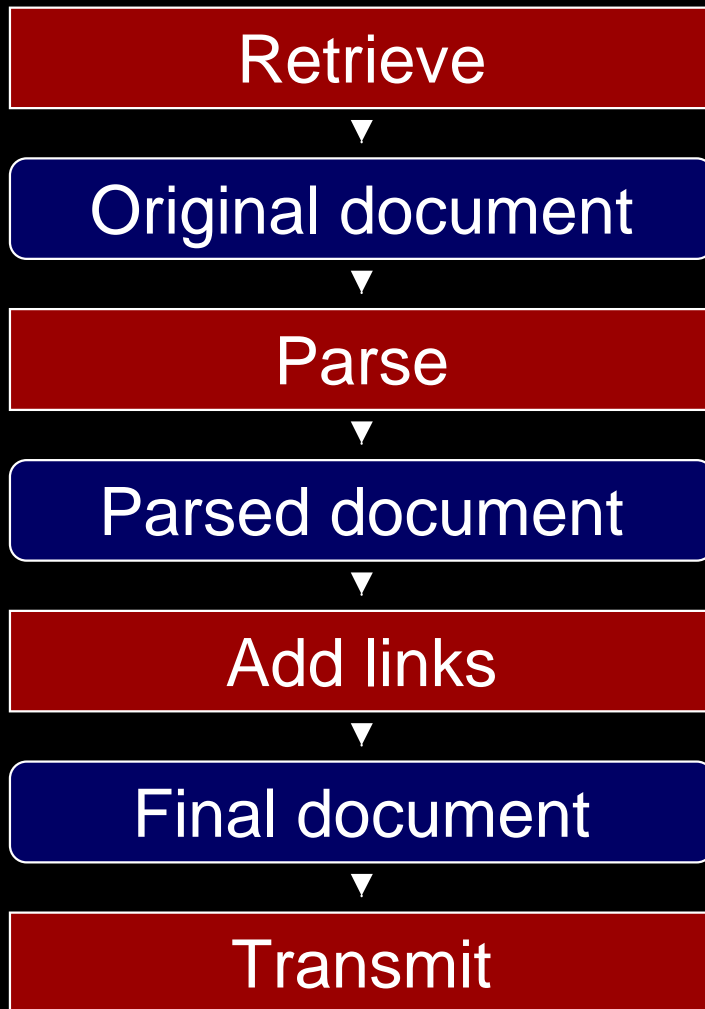
Content altering proxies

- If the proxy alters content then the benefits of traditional caching are greatly reduced.
- We consider a general content altering proxy process to be the following...

Process flow



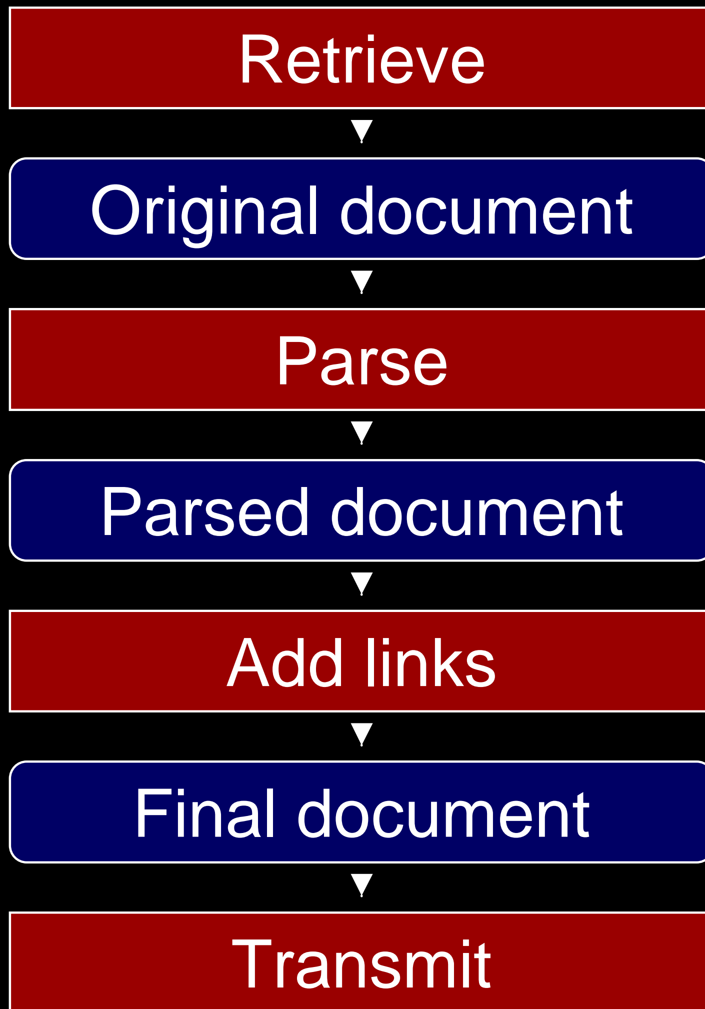
Retrieve



The first stage is to retrieve the document from the remote resource.

This gives us the original, unaltered, document.

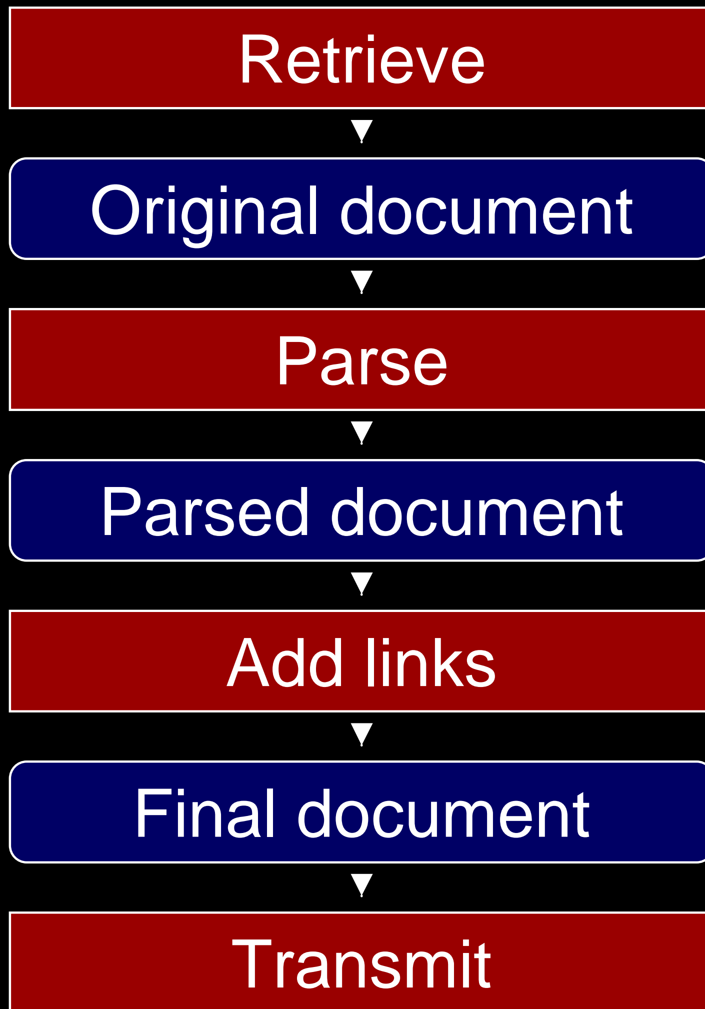
Parse



The document is parsed and stored in some structured, internal format.

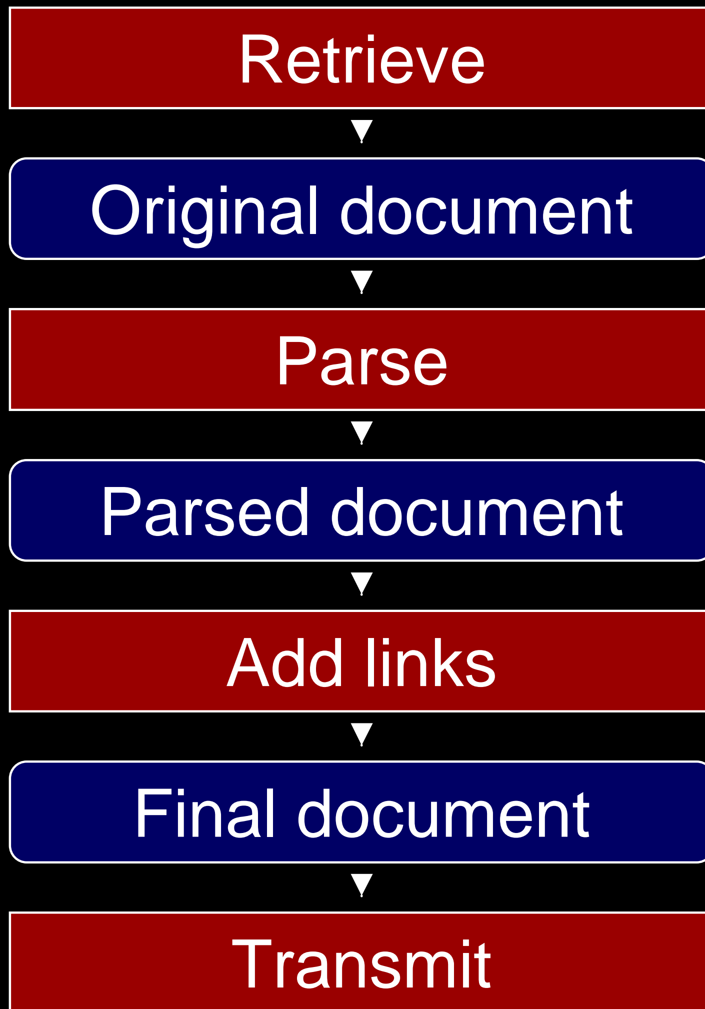
For example, an XML document may be stored as a tree.

Add links



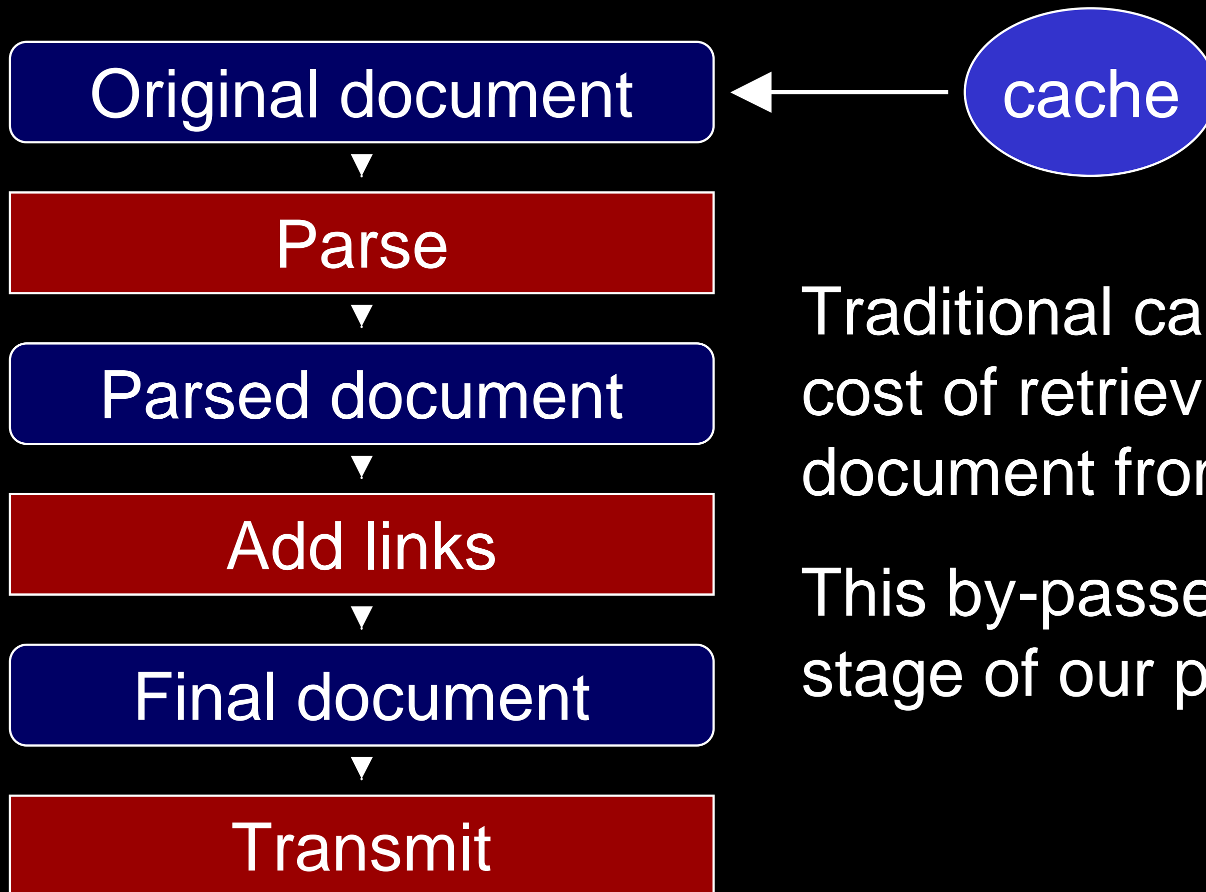
Links are added to the document from whatever source this system supports.

Transmit



The final version of the document is sent to the client.

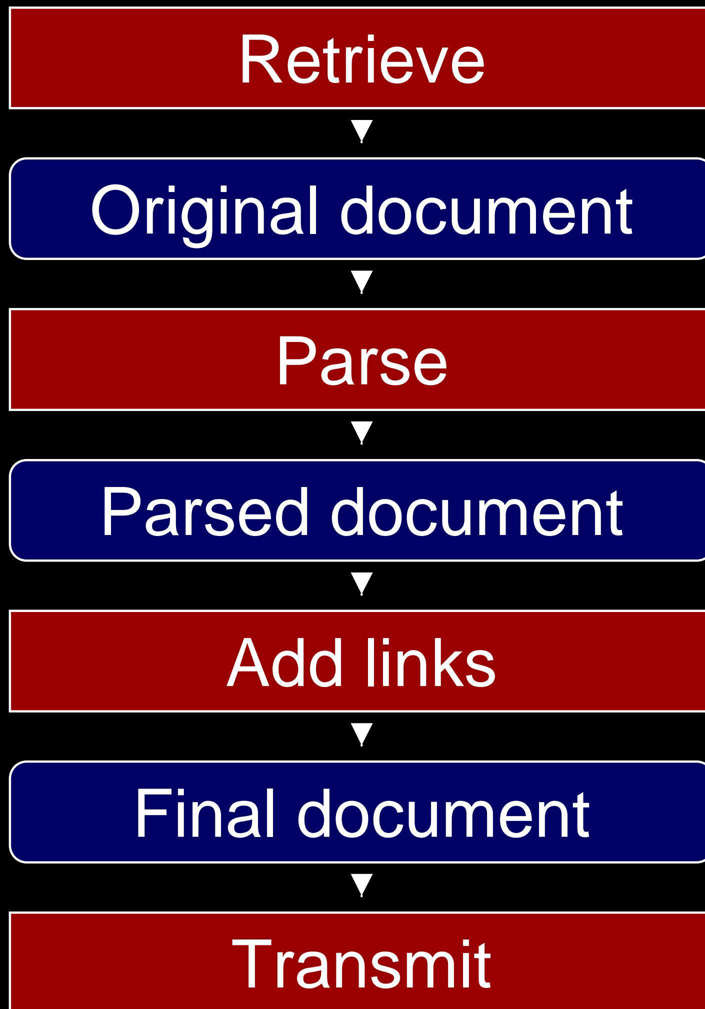
Traditional caching



Traditional caching saves the cost of retrieving the document from the network.

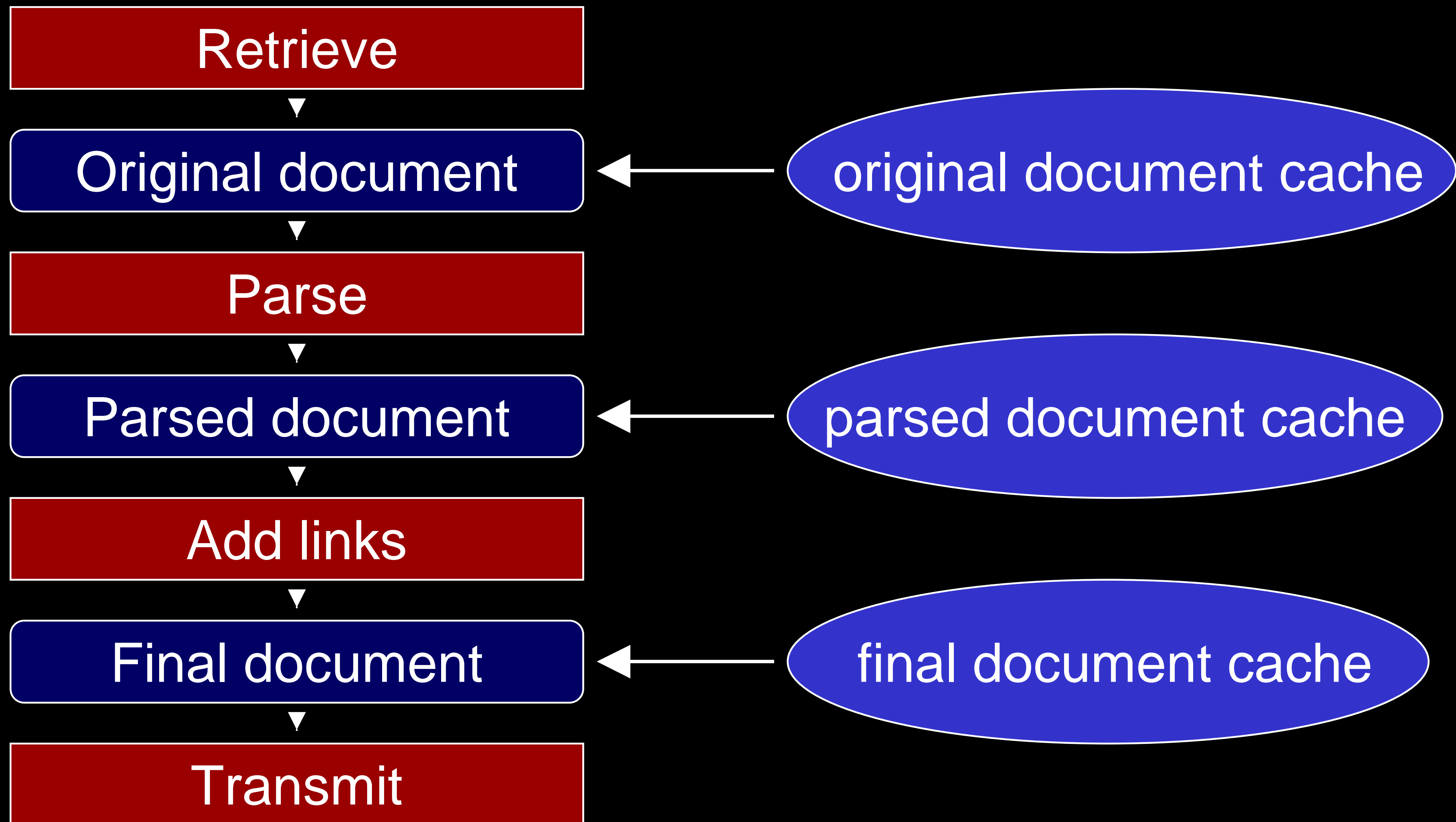
This by-passes the Retrieve stage of our process chart.

New opportunities



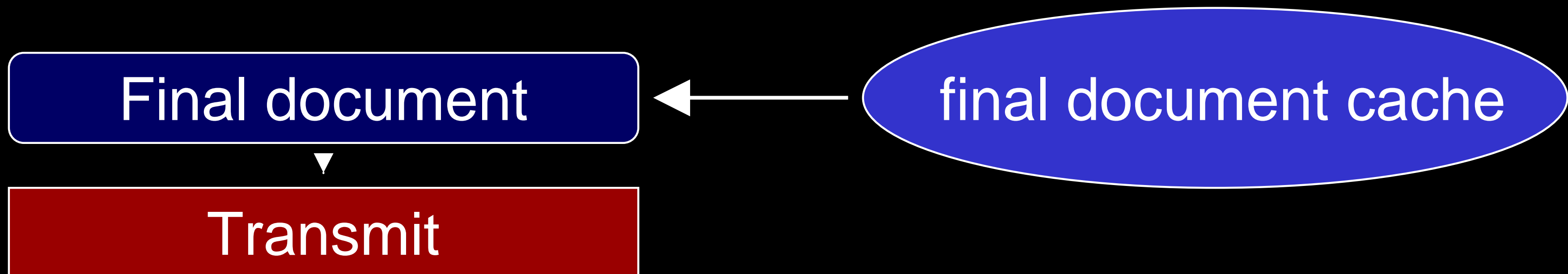
By bypassing the other processing stages we should be able to greatly reduce processing time and therefore improve performance.

New opportunities



New opportunities

Since we are caching the state of the document, not the process, retrieving from a cache skips all previous processing stages.



Evaluating

When (qualitatively) evaluating each approach we need to consider two things:

1. The processing cost of the approach
2. The storage cost of the approach

Processing cost

The processing cost measurement is a composite of real time, CPU time and other system load factors.

The total load for an approach is named Z.

For a system with no caching:

$Z = DOC\textit{transfer} + DOC\textit{parse} + LINK\textit{insert}$

Storage cost

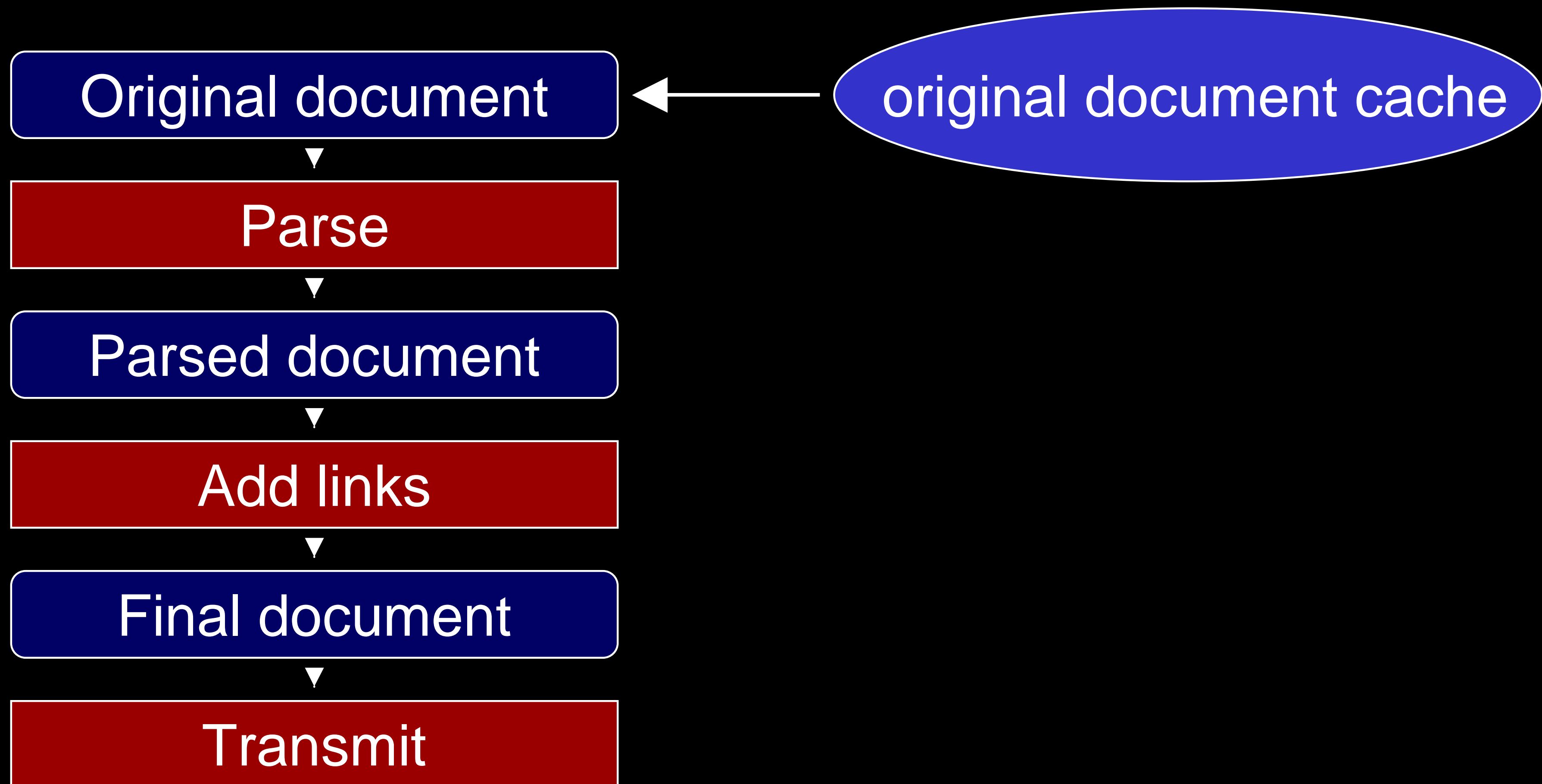
The storage cost of an approach describes how much space is required to implement.

The total space required for an approach is named S.

For a system with no caching:

$S = 0$

Original page caching



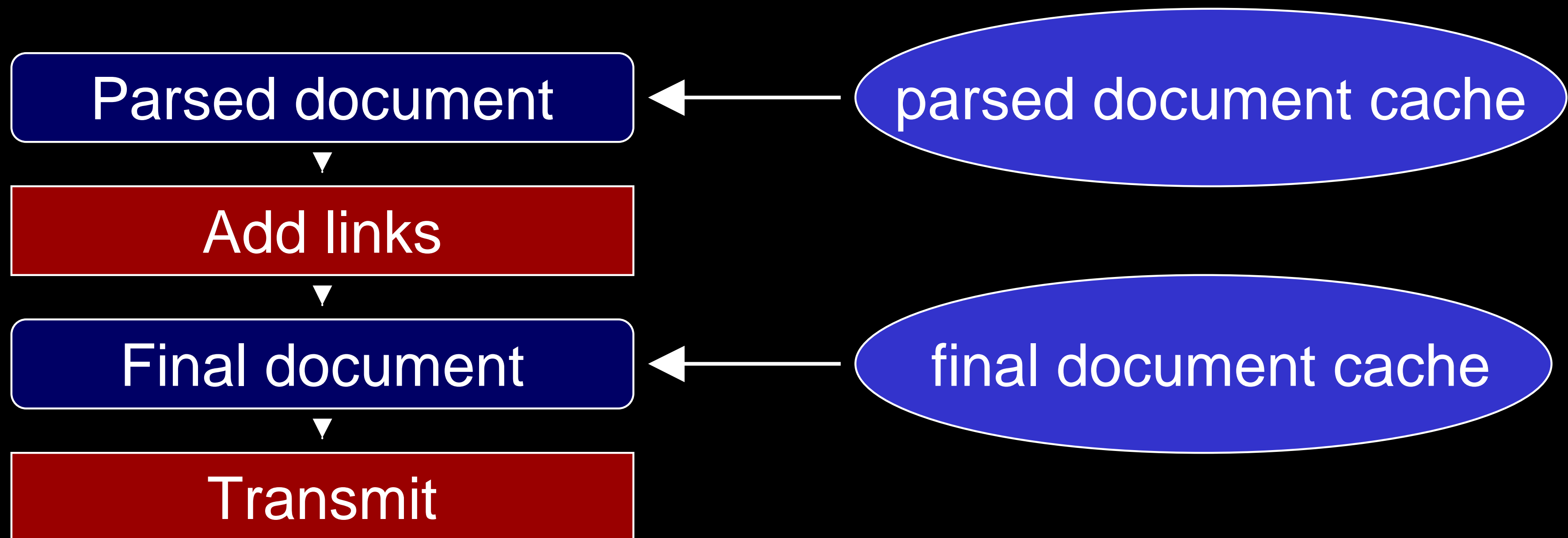
Original page caching

- Page is stored as it was sent from the server.
- Storage cost is low since the space required is the same as the document size.
- DOC*validate* is the cost of ensuring the cached version is up to date, DOC*cache* the cost of retrieving the document from the cache.

Z = DOC*validate* + DOC*cache* + DOC*parse* + LINK*insert*

S = DOC*size*

Parsed page caching



Parsed page caching

- Skips the computationally expensive parsing stage.
- The cost of retrieving from the cache (PARSED*cache*) should be much less than DOC*parse*.
- DOC*validate* still applies but there is no need for PARSED*validate* presuming the parser is consistent.

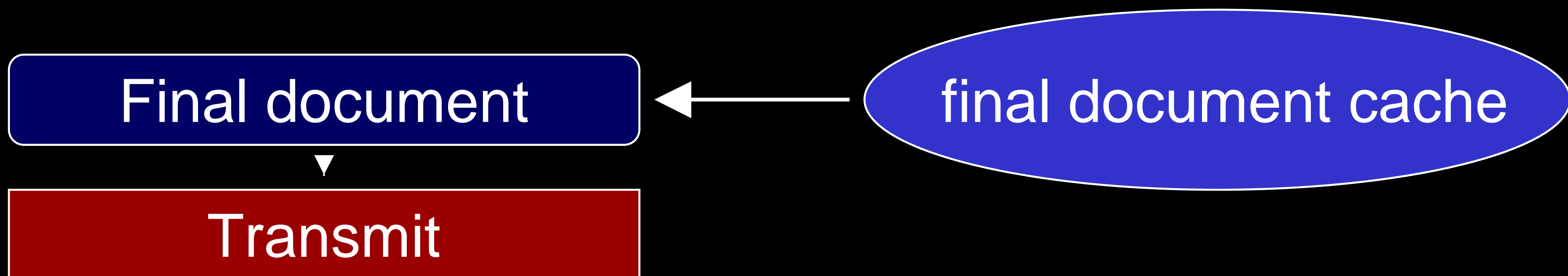
Z = DOC*validate* + PARSED*cache* + LINK*insert*

Parsed page caching

- Storage cost is high due to the structured nature of the result.
- ITEM*size* represents the overhead of storing an item versus, its textual size. ITEM*num* is the number of items in the document.

$$S = DOC\mathit{size} + (ITEM\mathit{size} \times ITEM\mathit{num})$$

Final page caching



Final page caching

- Final page caching stores the document as it was sent to the browser, skipping the link insertion stage.
- As well as checking the document is still valid, we also have to check the added links are valid.
- We presume $LINK_{validate} < LINK_{insert}$.

$$Z = DOC_{validate} + LINK_{validate} + FINAL_{cache}$$

Final page caching

- The cached version can either be stored as structured data or a flattened string.
- Storage cost for the structured approach is similar to parsed page cache, for the flattened approach it is similar to original page caching.
- In terms of Z both approaches perform equally for a cache hit.
- For a cache miss the structured approach has advantages.

Final page caching

• If $LINK_{validate}$ shows the links are invalid, but in some way we can tell that only some of the links are invalid (e.g. links from a certain source) we can reduce the cost of the miss:

$$LINK_{insert'} = LINK_{insert} \times \frac{SCOPES_{invalid}}{SCOPES_{total}}$$

Fall-back caching

- It is possible to use more than one approach at once.
- Useful where the links are invalid but the document isn't.
- Using a parsed page cache will still give better overall performance then no-caching even during a 'miss'.
- The cost of this one event is:

Z = DOC*validate* + LINK*validate* + PARSED*cache* + LINK*insert*

Cache retrieval costs

- Retrieving for memory is faster than from disc, but space is limited.
- Disc storage cannot natively store structures such as trees.
- Structured data must be serialised before storing on disc.

Cache retrieval costs

- Deserialisation less expensive than parsing since data can be stored in machine friendly way.
- Serialised disc storage only useful if the deserialisation costs are less than the cost of fresh processing.

Delta storage

- So far we've considered each cache as being separate.
- By exploiting similarities between caches we can cut down on the storage costs.
- This works by storing the delta (difference) between two states.
- Less storage but higher Z.

Delta storage

- Caches must be in the same format, i.e.:
    - Flattened vs flattened.
    - Structured vs structured.

- Link addition stage a good candidate as the difference will be the links.

Conclusions

- There is no point in original page caching – use (serialised) parsed page caching.
- If links are as (or more) stable than the document there is no point in parsed page caching either – use flattened final page caching.
- If links are less stable than the document there are a number of possibilities.

Conclusions

- Flattened storage more space efficient then structured.
- Structured with delta-storage is also efficient but has the extra Z cost of delta application.
- Structured allows the removal of invalid scopes:
    - Useful with stable, high insertion cost scopes
    - Less useful for instable, high insertion cost scopes
    - Less useful for stable, low insertion cost scopes
- Disc based caching with cache records in RAM should at worst have no negative effect.

Questions?