Lookup
A Plumber's Swiss Army Knife

Rob van der Heij
IBM Global Services
rvdheij@nl.ibm.com

VM and VSE Technical Conference
June 2000, Orlando

Agenda

- Basic Principle of Lookup
- Counting the References
- Dynamic Updates to the Reference Table
- Stopping and Restarting

Examples shown inside appear smaller than they are.
Complete examples at http://pucc.princeton.edu/~pipeline/

Basic Principle of Lookup

process an input stream (detail records) against a reference (master records), comparing a key field

input streams

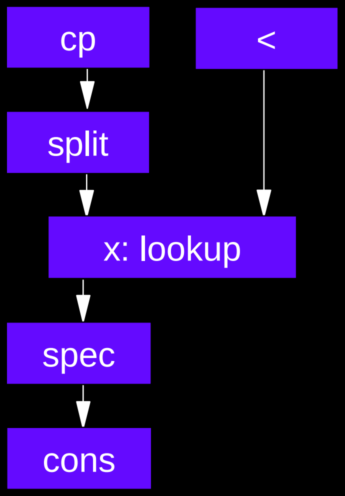
- detail records
- master records

output streams

- matched records
- unmatched records
- unused masters

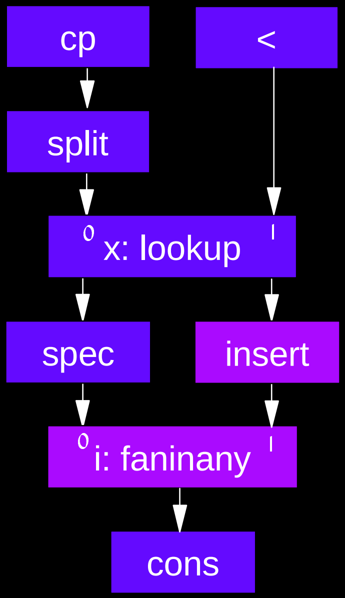
A Simple Example

```
PIPE (end \)
  \ cp q link 19e
  | split ,
  | x: lookup w1
       detail master
  | spec w2.2 1
       read 10-* nw
  | cons
  \ < users list
  | x:
```



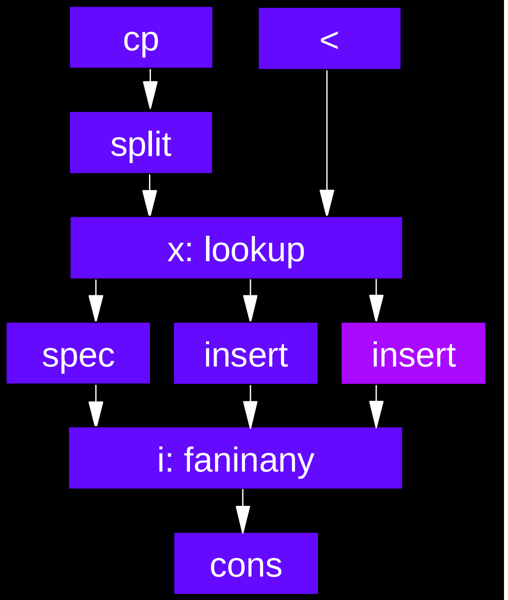
A Fairly Simple Example

```
PIPE (end \)
   \ cp q link 19e
   | split ,
   | x: lookup w1
        detail master
   | spec w2.2 1
        read 10-* nw
   | i: faninany
   | cons
   \ < users list
   | x:
   | insert /Userid /
   | i:
```



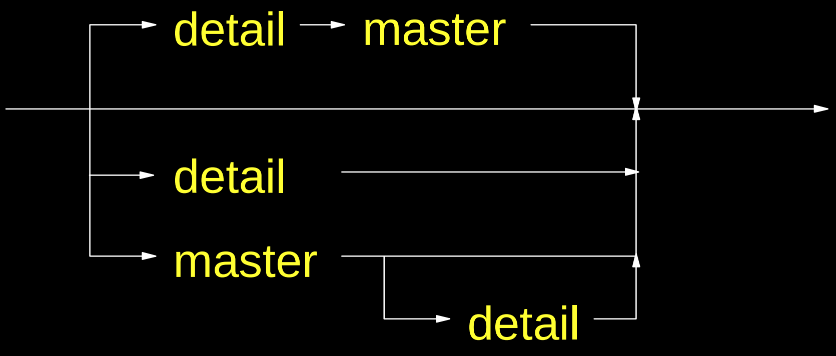
A Less Simple Example

```
PIPE (end \)
   \ cp q link 19e
   | split ,
   | x: lookup w1
        detail master
   | spec w2.2 1
        read 10-* nw
   | i: faninany
   | cons
   \ < users list
   | x:
   | insert /Userid /
   | i:
   \ x:
   | insert /No link /
   | i:
```



Matched Records Written

any combination of **detail** and **master**



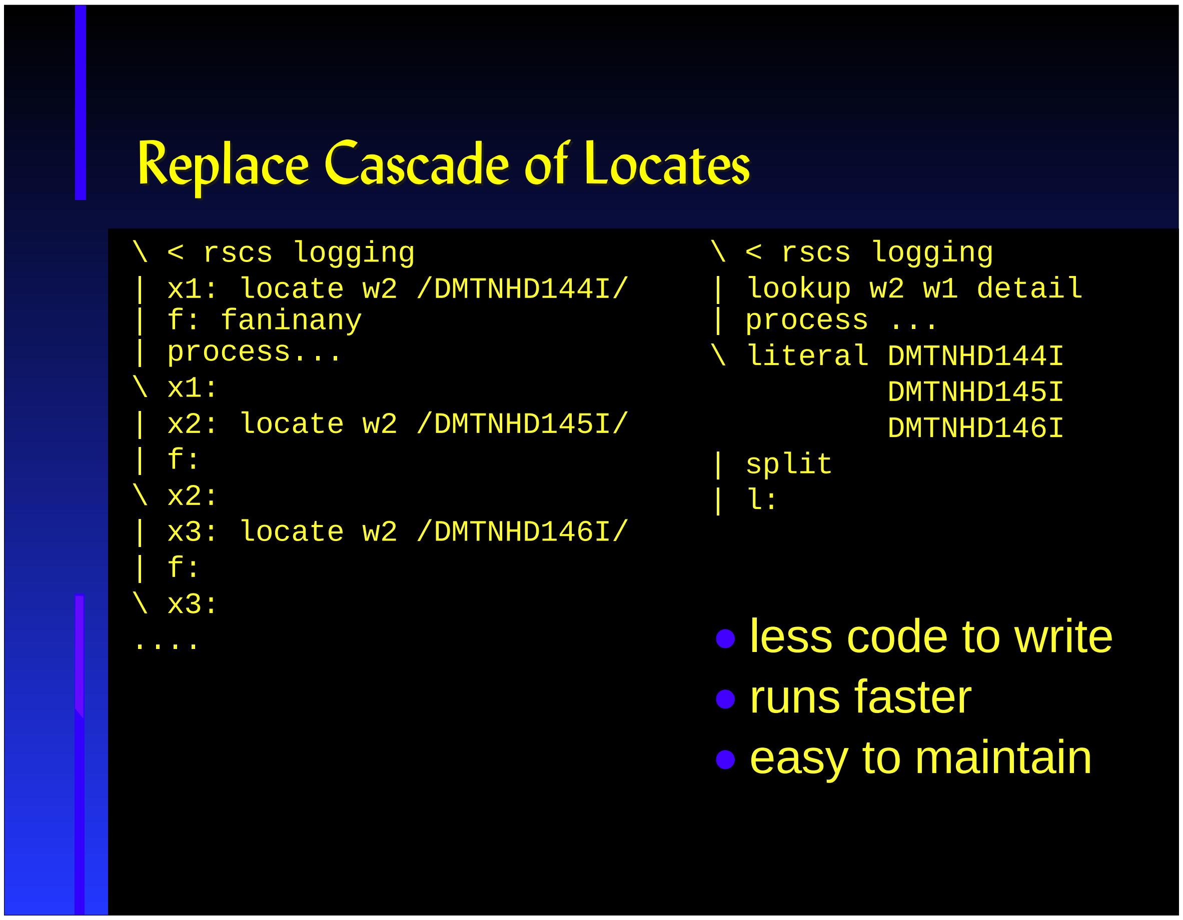
allmasters to retain duplicate masters

Input Ranges

two input ranges define the key field

pad	pad short records
anycase	case-insensitive match

input ranges: 1.8
w3
substr w2 of 11-*
substr fs : f2 of w3

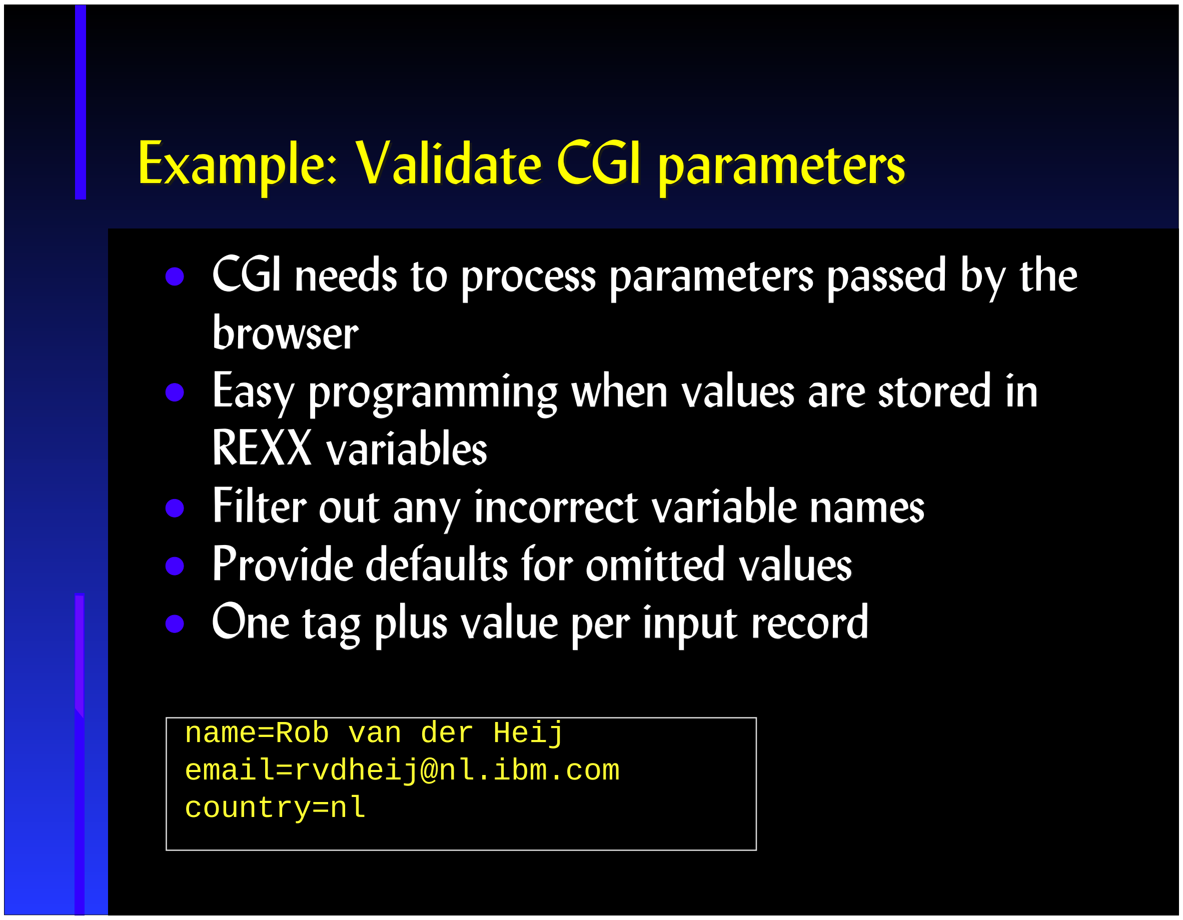


Replace Cascade of Locates

```
\ < rscs logging
| x1: locate w2 /DMTNHD144I/
| f: faninany
| process...
\ x1:
| x2: locate w2 /DMTNHD145I/
| f:
\ x2:
| x3: locate w2 /DMTNHD146I/
| f:
\ x3:
....
```

```
\ < rscs logging
| lookup w2 w1 detail
| process ...
\ literal DMTNHD144I
          DMTNHD145I
          DMTNHD146I
| split
| l:
```

- less code to write
- runs faster
- easy to maintain



Example: Validate CGI parameters

- CGI needs to process parameters passed by the browser
- Easy programming when values are stored in REXX variables
- Filter out any incorrect variable names
- Provide defaults for omitted values
- One tag plus value per input record

```
name=Rob van der Heij
email=rvdheij@nl.ibm.com
country=nl
```

Example: Validate CGI parameters

```
callpipe (end \ name validprm)
\ *:                tags to be validated
| l: lookup anycase fs = f1 detail
| i: fanin
| xlate fs = f1     uppercase name
| insert ,=,        make it VARLOADable
| *:
\ *.input.1:     list of tags plus default
| l:
\ l:                missing tags with default
| i:
```

```
tags plus default
name=
email=
street=
city=
country=us
```

a single lookup

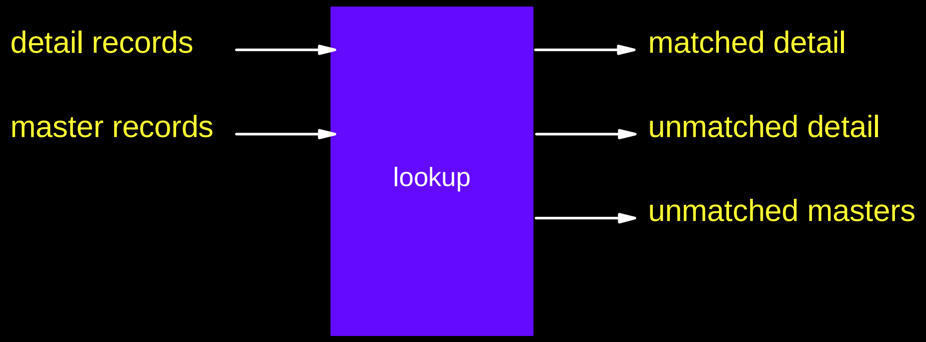
- validation of the tags
- supply a default value

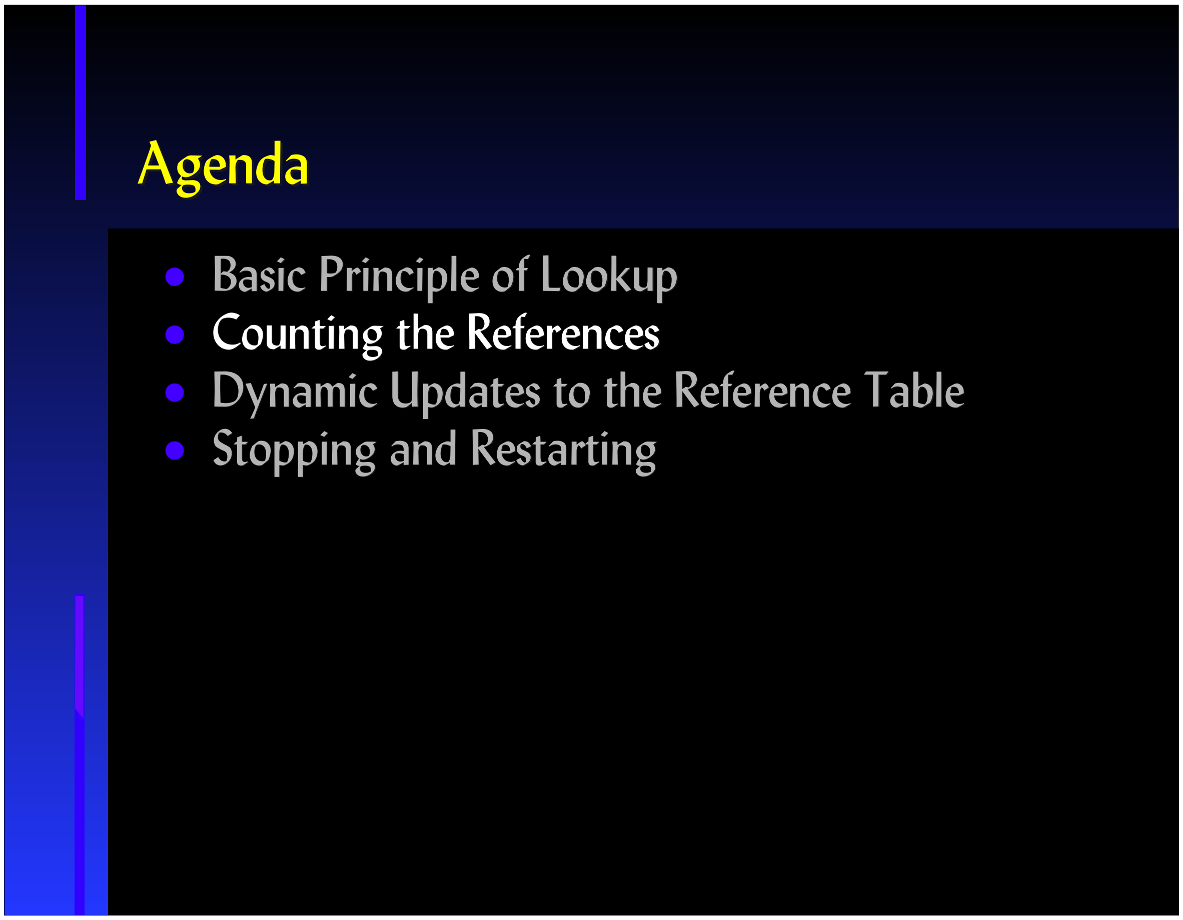
further enhancement

- deal with optional versus required fields

Streams used by lookup (so far)

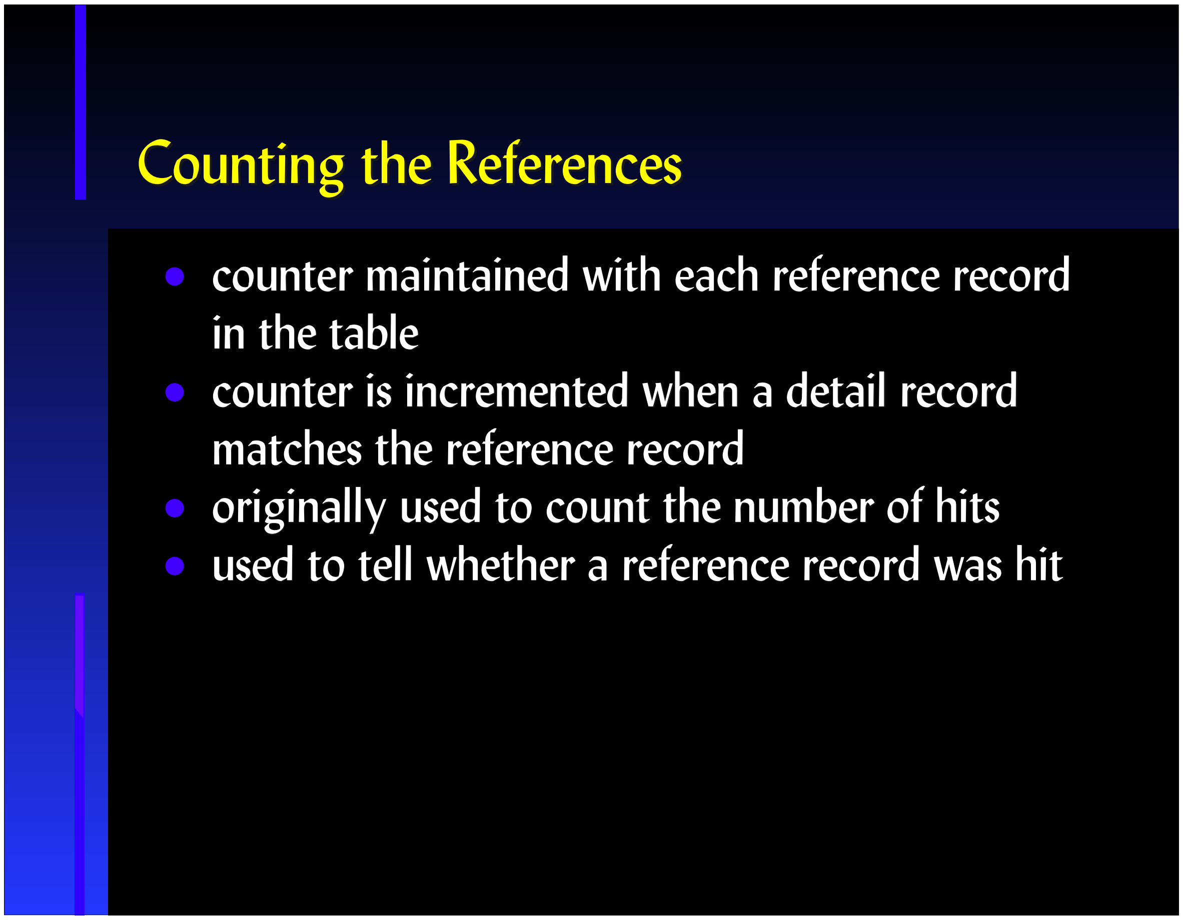
- master records on secondary stream read first
- unmatched masters on tertiary output written after termination





Agenda

- Basic Principle of Lookup
- Counting the References
- Dynamic Updates to the Reference Table
- Stopping and Restarting



Counting the References

- counter maintained with each reference record in the table
- counter is incremented when a detail record matches the reference record
- originally used to count the number of hits
- used to tell whether a reference record was hit

The COUNT option

- prefixes masters on tertiary output with the count value
- all reference records written to the tertiary output (both unmatched and matched)

sometimes sort count can do it as well

Example: COUNT option

```
\ cms listfile * *
| l: lookup count 10.8 1.8
           detail master
| spec 1-17 1 read 10-8 nw
| > file0 output a
\ < filetype describe
| l:
| > file1 output a
\ l:
| insert , , after 10
| > file2 output a
```

from detail

from master

```
RMHLUP   LIST3820 AFP Output
PIPELINE OFSLOGfl OV/VM Notelog
JPH      OFSLOGfl OV/VM Notelog
VMESA230 LIST3820 AFP Output
```

```
LIST3820 AFP Output
OFSLOGfl OV/VM Notelog
NOTEBOOK CMS Notebook
EXEC     REXX program
```

```
PROFILE  XEDIT
```

```
0 EXEC     REXX program
2 LIST3820 AFP Output
0 NOTEBOOK CMS Notebook
2 OFSLOGfl OV/VM Notelog
```

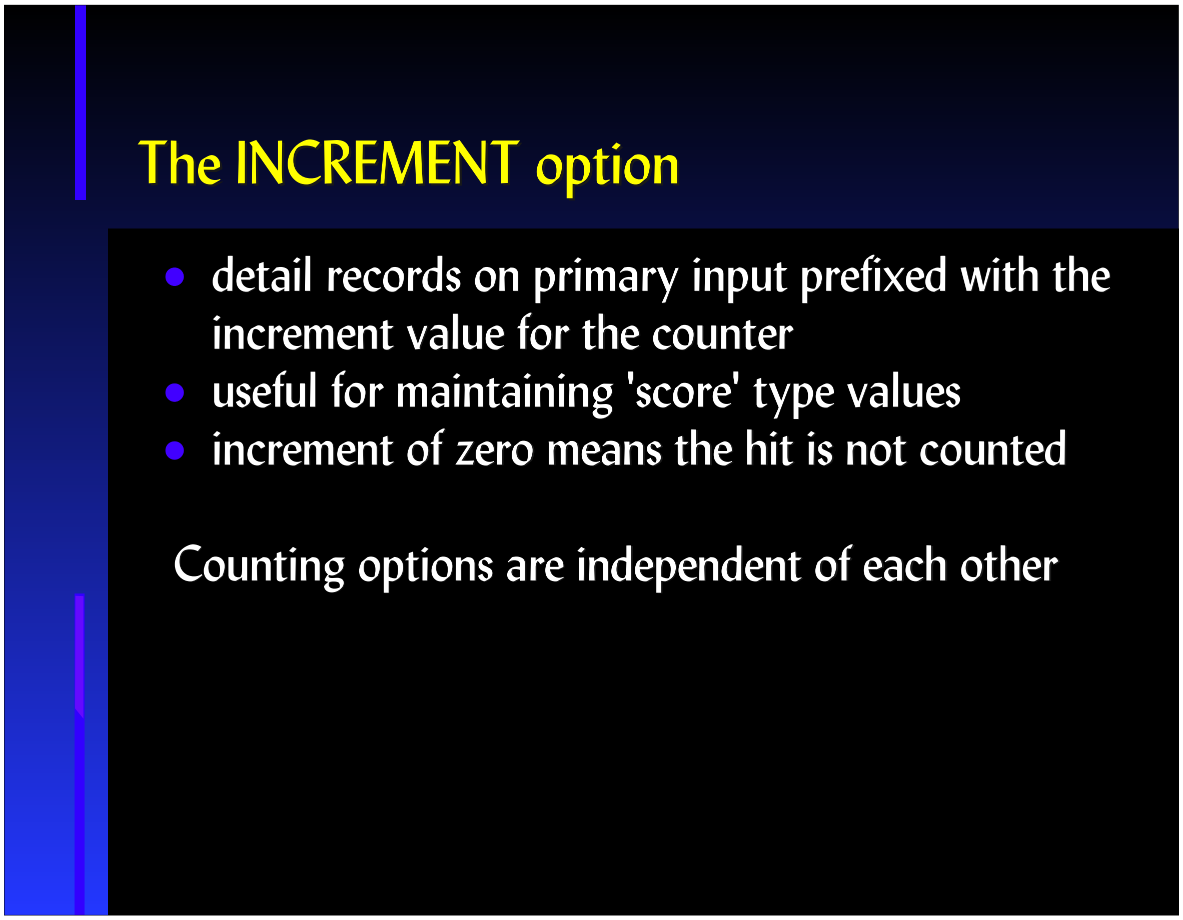
The TRACKCOUNT option

- prefixes master records on primary output with the current counter value
- counter is incremented before writing the output record (so output starts with 1)

useful to assign sequence numbers to records in a run without sorting them first

The SETCOUNT option

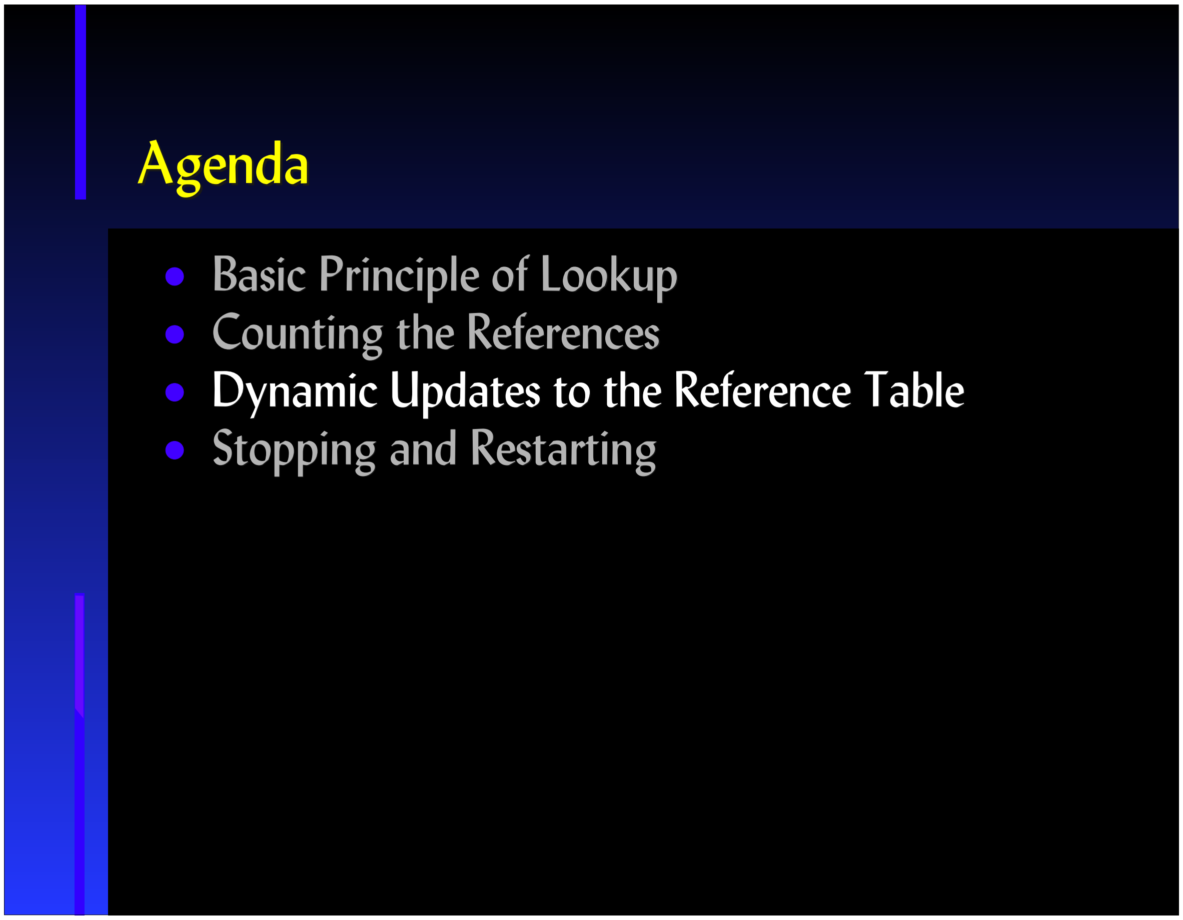
- master records prefixed with a 10-digit initial value for the counter
- useful if you need to reload the reference table from an earlier invocation
- also many other interesting applications
- negative value also possible



The INCREMENT option

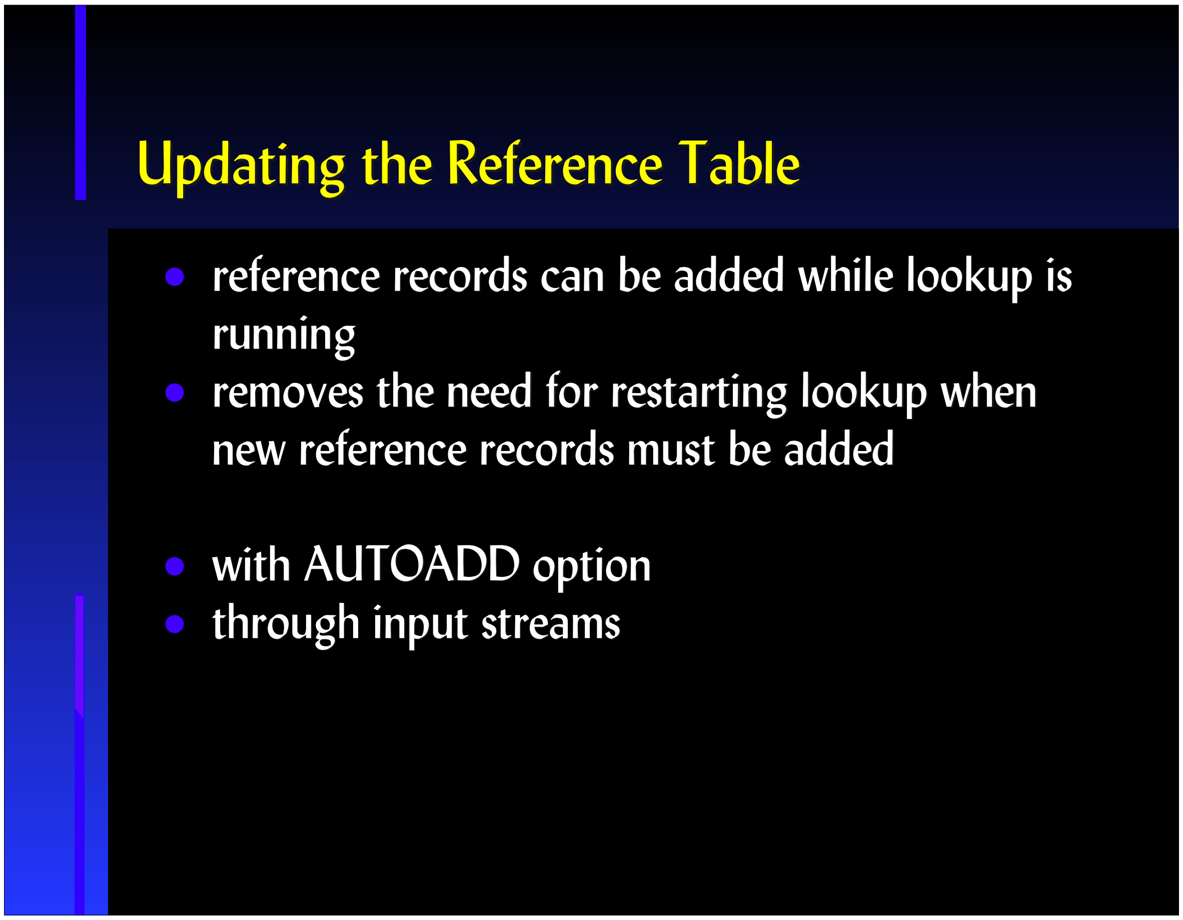
- detail records on primary input prefixed with the increment value for the counter
- useful for maintaining 'score' type values
- increment of zero means the hit is not counted

Counting options are independent of each other



Agenda

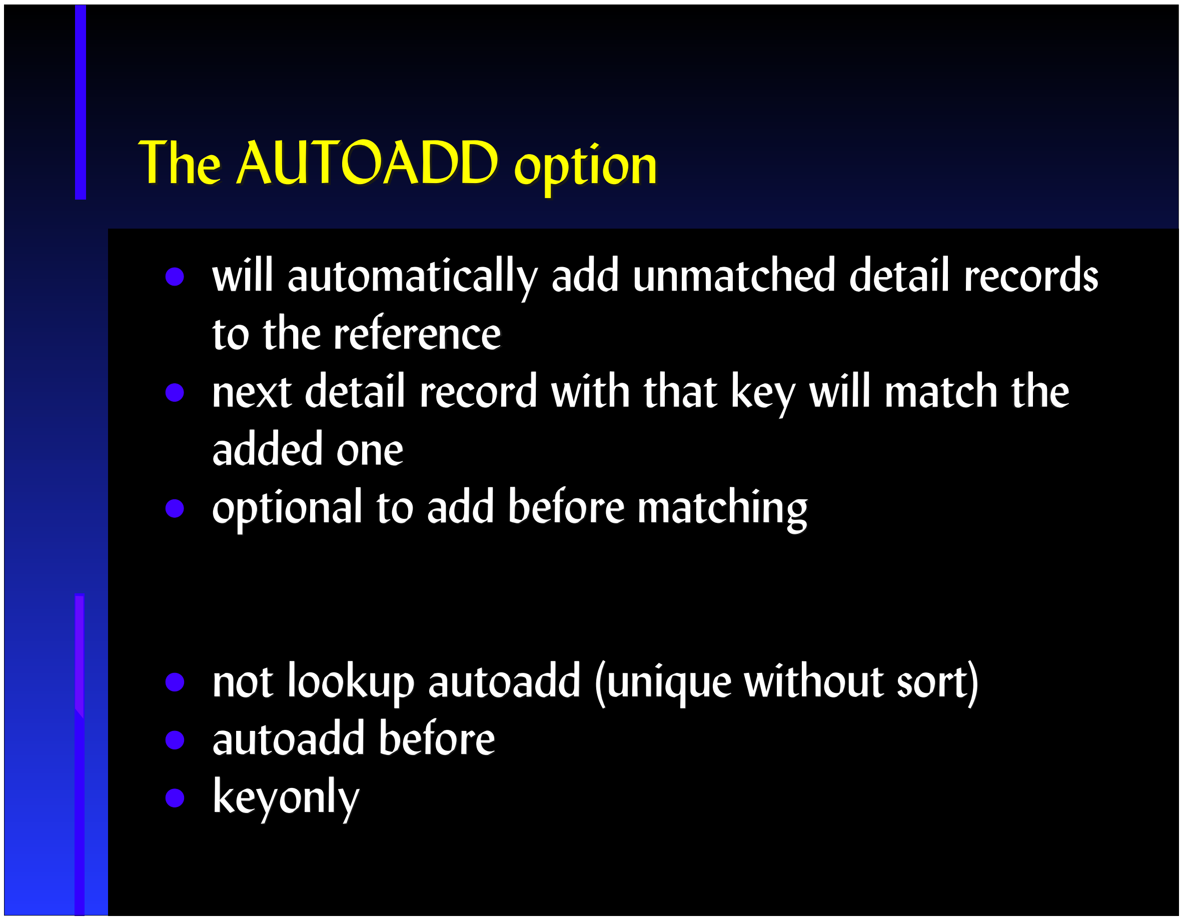
- Basic Principle of Lookup
- Counting the References
- Dynamic Updates to the Reference Table
- Stopping and Restarting



Updating the Reference Table

- reference records can be added while lookup is running
- removes the need for restarting lookup when new reference records must be added

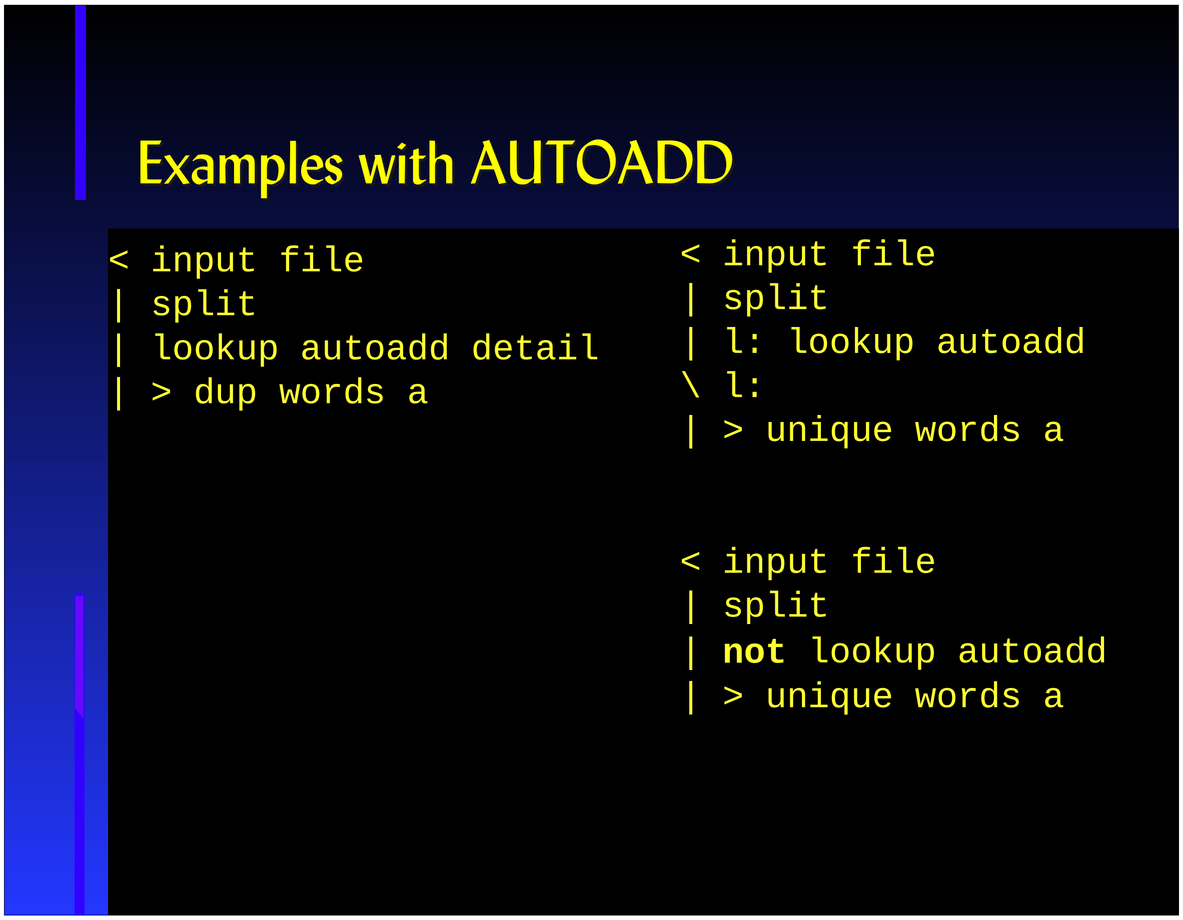
- with AUTOADD option
- through input streams



The AUTOADD option

- will automatically add unmatched detail records to the reference
- next detail record with that key will match the added one
- optional to add before matching

- not lookup autoadd (unique without sort)
- autoadd before
- keyonly

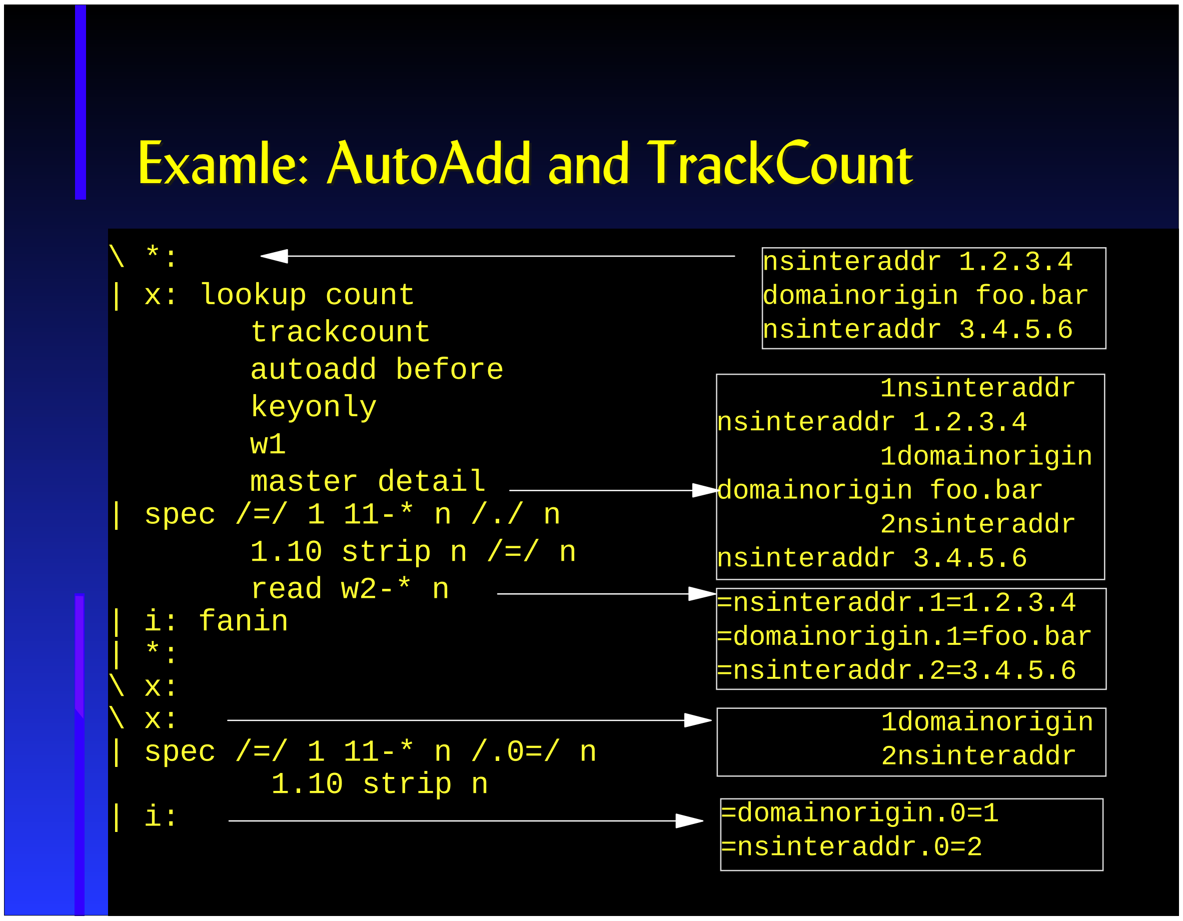


Examples with AUTOADD

```
< input file
| split
| lookup autoadd detail
| > dup words a
```

```
< input file
| split
| l: lookup autoadd
\ l:
| > unique words a
```

```
< input file
| split
| not lookup autoadd
| > unique words a
```



Examle: AutoAdd and TrackCount

```
\ *:
| x: lookup count
        trackcount
        autoadd before
        keyonly
        w1
        master detail
| spec /=/ 1 11-* n /./ n
        1.10 strip n /=/ n
        read w2-* n
| i: fanin
| *:
\ x:
\ x:
| spec /=/ 1 11-* n /.0=/ n
         1.10 strip n
| i:
```

```
nsinteraddr 1.2.3.4
domainorigin foo.bar
nsinteraddr 3.4.5.6
```

```
          1nsinteraddr
nsinteraddr 1.2.3.4
          1domainorigin
domainorigin foo.bar
          2nsinteraddr
nsinteraddr 3.4.5.6
```

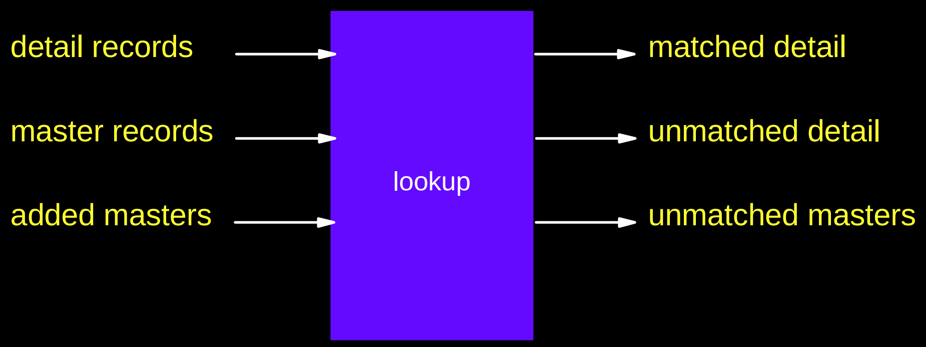
```
=nsinteraddr.1=1.2.3.4
=domainorigin.1=foo.bar
=nsinteraddr.2=3.4.5.6
```

```
          1domainorigin
          2nsinteraddr
```

```
=domainorigin.0=1
=nsinteraddr.0=2
```

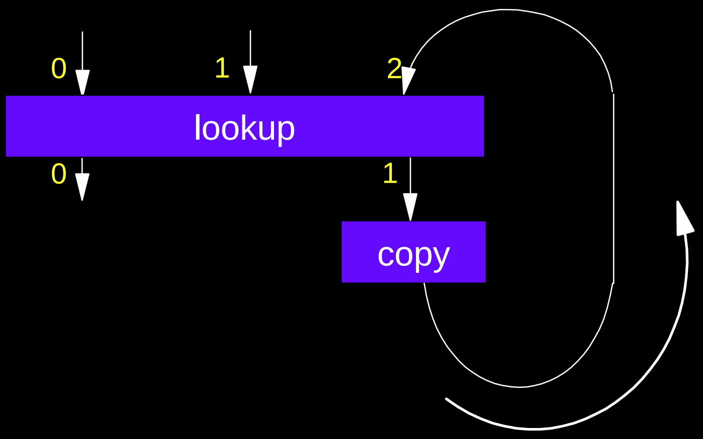
Tertiary Input Stream

master records on third input stream are added to the reference when they become available



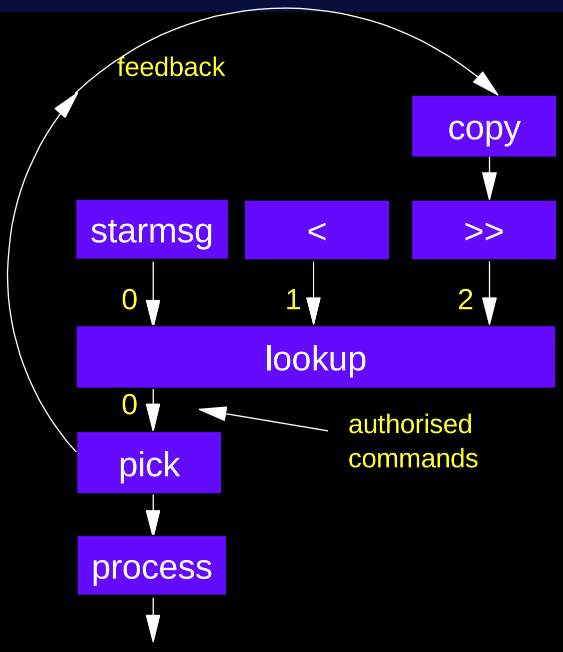
Tertiary Input Stream

write your own autoadd
the copy stage needed in feedback



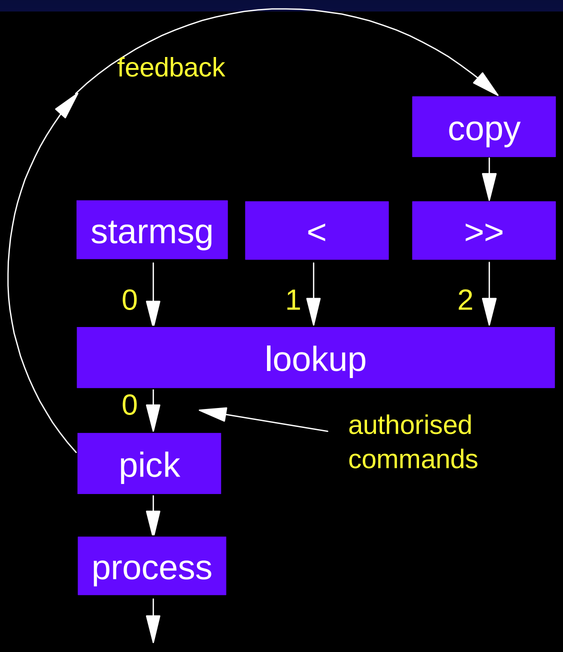
Example: Tertiary Input

service machine that processes commands from authorised users, including command to authorise users



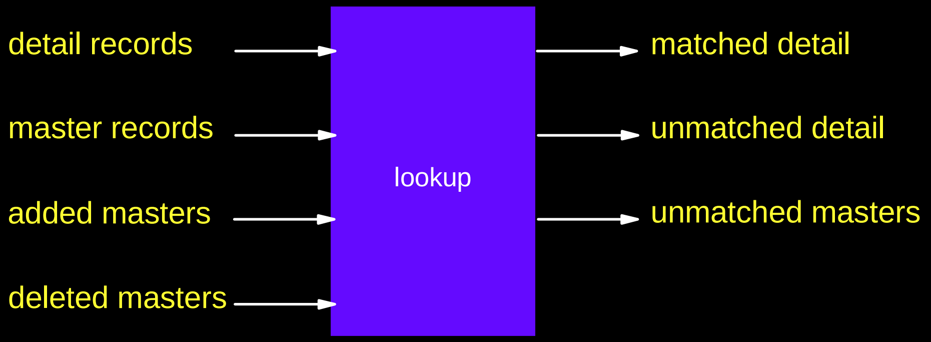
Example: Tertiary Input

```
\ starmsg
| not chop 8
| l: lookup 1.8 detail
| add: pick substr w1
          of 9-* /== ,ADD,
| process
\ < auth users | l:
\ add:
| spec substr w2
          of 9-* 1.8
| xlate | copy
| >> auth users
| l:
```

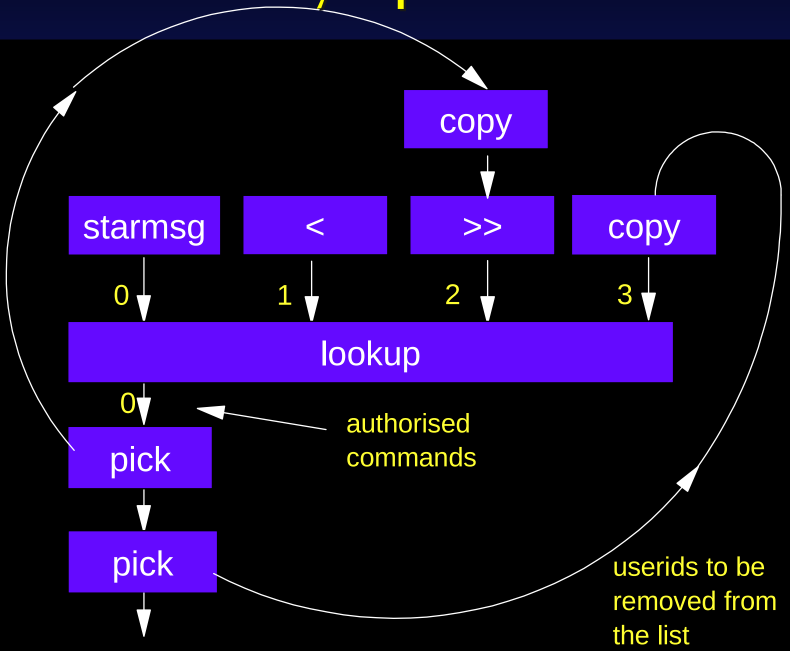


Quarternary Input Stream

- master records on fourth input stream are deleted from the reference
- further records will not match the reference

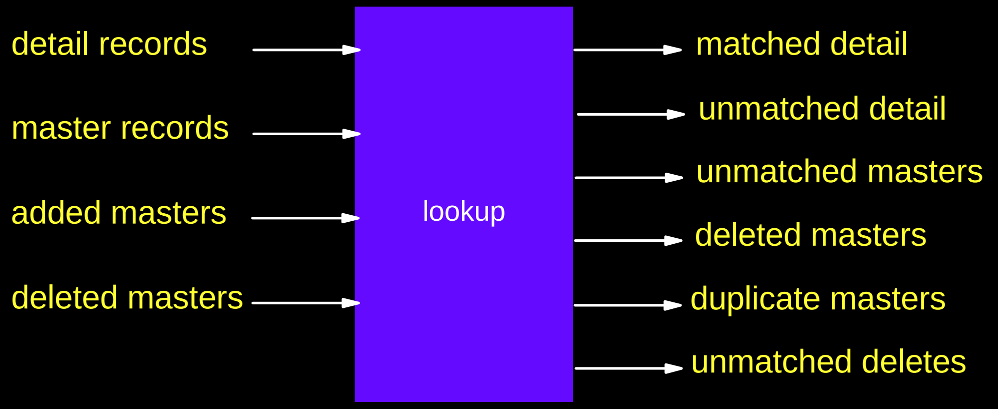


Example: Quarternary Input



Extra Output Streams

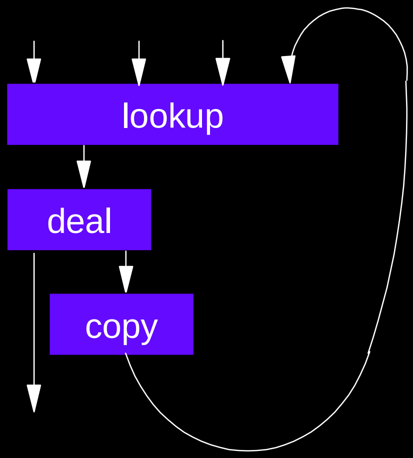
Quarternary	deleted master records
Quinary	duplicate (rejected) masters
Senary	unmatched (rejected) deletes



Feedback Loops (once)

e.g. delete master after first reference

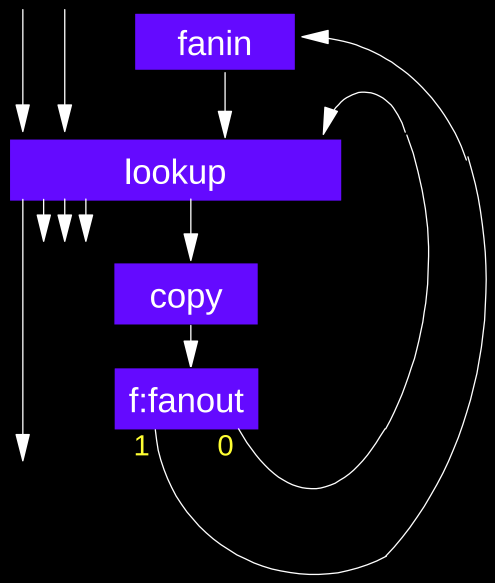
```
\ *:
| l: lookup 1.8 detail master
| d: deal
| *:
\ l:
\ *.input.1:
| l:
\ d:
| copy
| l:
```



Feedback Loops (replace)

e.g.replace reference
by duplicate master

```
\ *:
| l: lookup 1.8
     detail master
| *:
\ l:
\ *.input.1:
\ a: fanin
| l:
\ d: fanin | l:
\ l:
| copy
| f: fanout
| d:
\ f: | a:
```



Example: Allocate Mini Disks

MDISK cuu type start size volser mode ...

```
\ *:
| m: zone w1 abbrev MDISK anycase
| spec w5 1.10 r 1-* nw
| l: lookup autoadd before
    increment trackcount
    w6 master detail
| spec 1.10 1 read 1-* nw
| spec a: w1 - b: w6 -
    w2.3 1
    print a-b+1 nw.6 ri
    w6-* nw
| f: faninany
| *:
\ m: | f:
```

autoadd: insert each new volser in reference table

increment: add size to the running total for each volser

trackcount: output running total with each master (to compute start cylinder)

Word of Caution

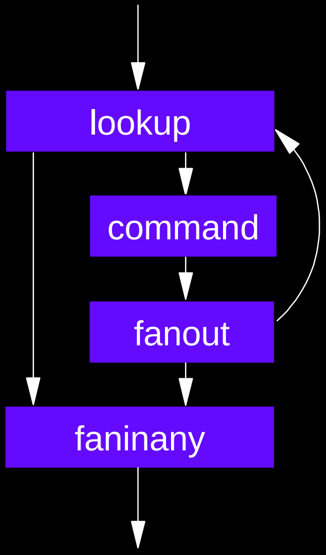
- storage is not freed when the reference is deleted
- after a number of deletions and additions the lookup stage should be recycled

Writing a Cache Stage

- provide a transparent way to store and retrieve results of earlier processing
- lookup can
  - search the reference table for a key
  - store new records in the reference table

NameFind Cache

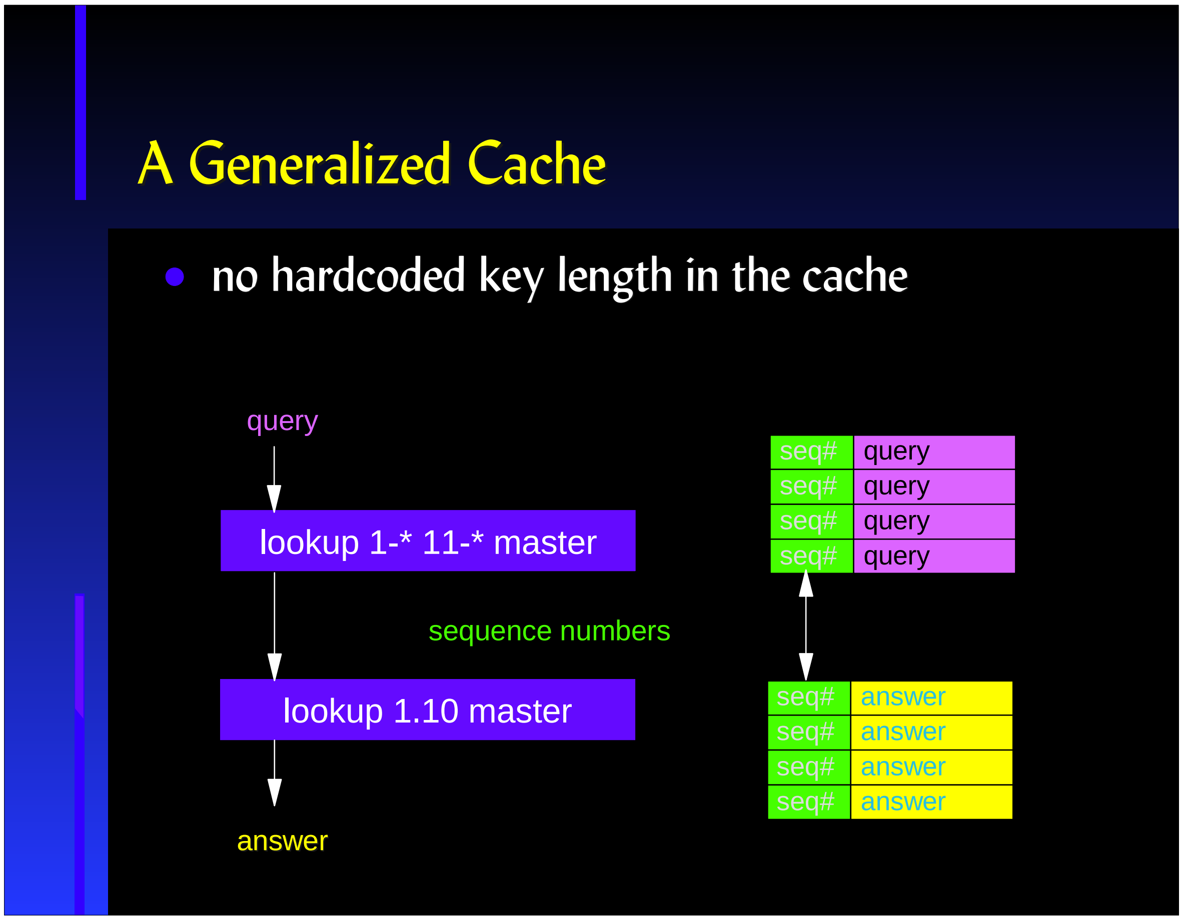
```
\ *:
| pad 8
| x: lookup 1.8 master
| f: faninany
| *:
\ x: | copy
| u: fanout | copy
| s: spec 1.8 1 select 1 1-* n
| b: fanout | x:
\ u:
| spec ,NAMEFIND :USERID, 1 1-* nw
          ,:NAME, nw
| command
| s:
\ b: | f:
```



A Generalized Cache

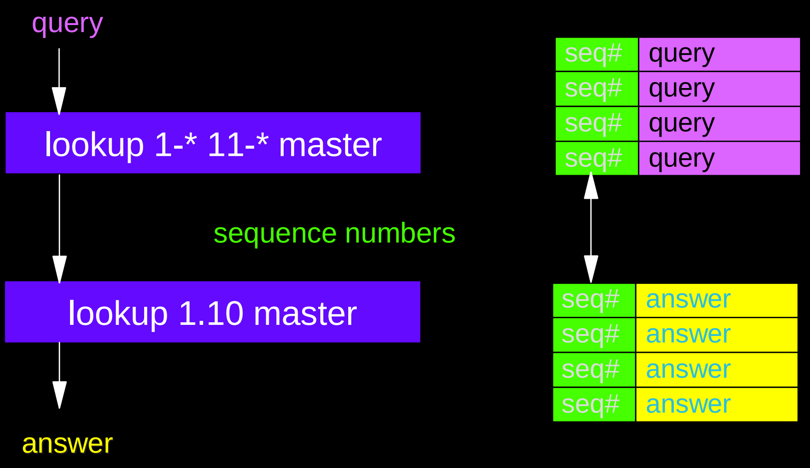
- no hardcoded key length in the cache
- resolver pipeline outside the cache

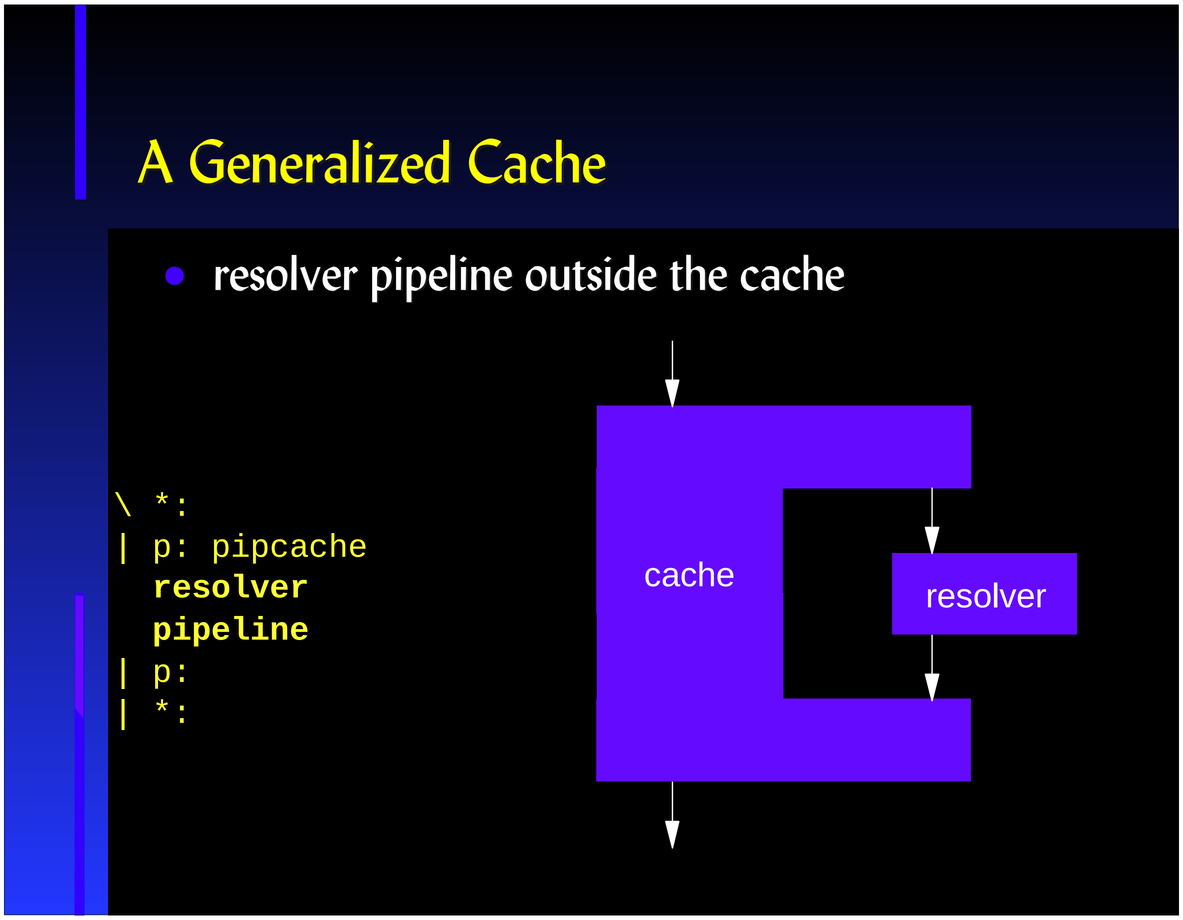
- code the cache once and concentrate on the real process



A Generalized Cache

- no hardcoded key length in the cache

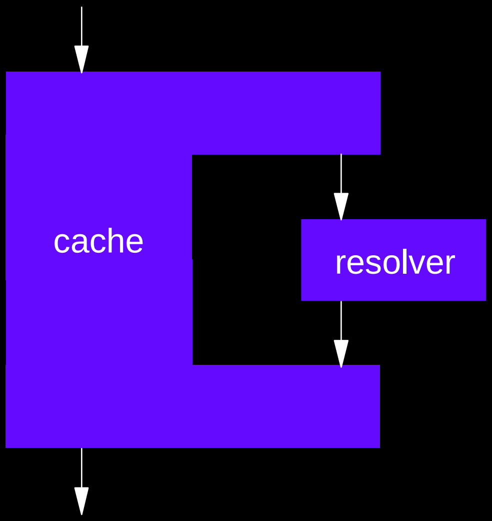




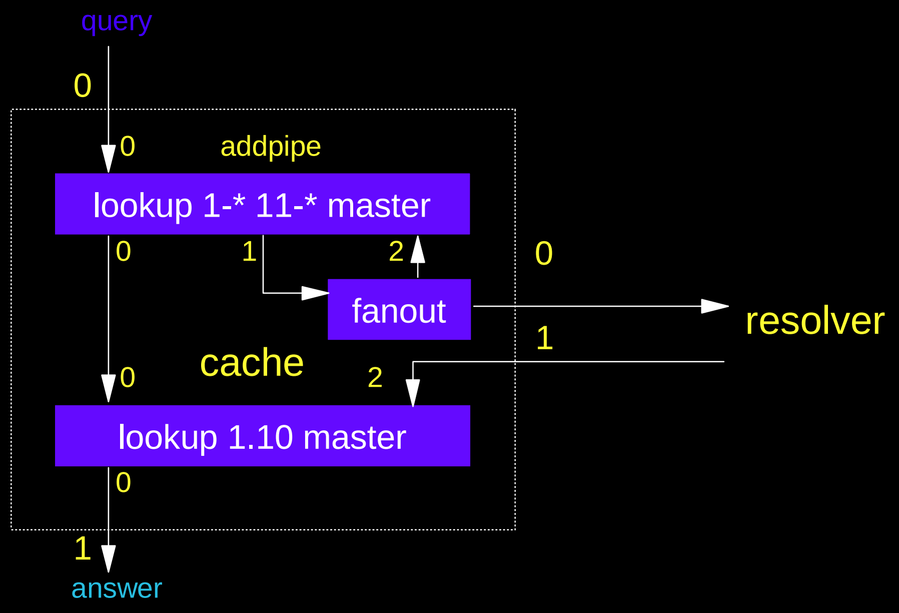
A Generalized Cache

- resolver pipeline outside the cache

```
\ *:
| p: pipcache
  resolver
  pipeline
| p:
| *:
```

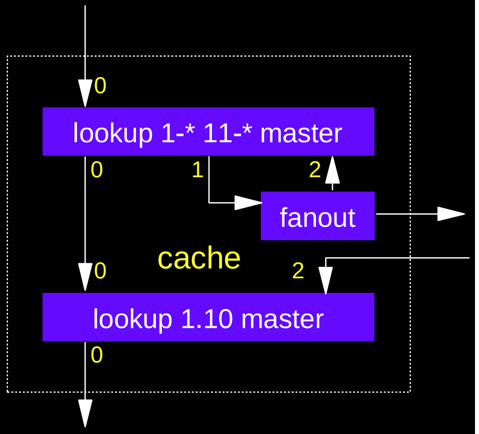


A Generalized Cache



A Generalized Cache

```
addpipe (end \)
\ *:
| x1: lookup 1-* 11-* master
| x2: lookup 1.10 master
| i: faninany
| not chop 10 | *.output.1:
\ x1:
| spec number 1.10 r 1-* n
| o1: fanout | copy | x1:
\ x2:
\ o1: | ch: chop 10 | copy
| s: spec 1-* 1 select 1 1-* n
| o2: fanout
| x2:
\ ch: | *.output.0:
\ *.input.1: | s:
\ o2: | i:
```



A Generalized Cache

- no cleanup mechanism included
- lookup does not release storage when references are deleted

- need to stop/restart the pipeline in a transparent way

Agenda

- Basic Principle of Lookup
- Counting the References
- Dynamic Updates to the Reference Table
- Stopping and Restarting

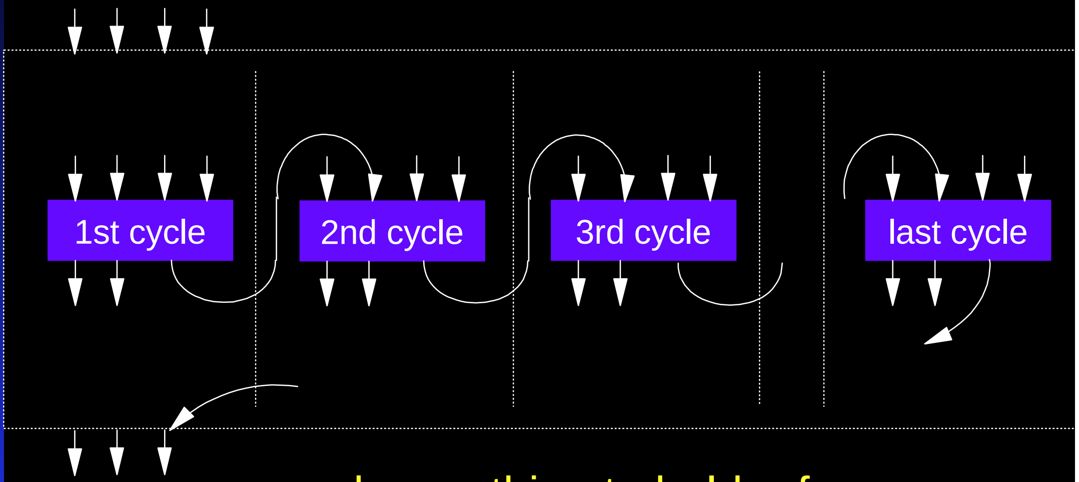
Restarting Lookup

```
  do forever
    peekto inp
    if rc <> 0 then leave
    callpipe ... | lookup | ..
  end
  return rc * ( rc <> 12 )
```

multiple invocations of lookup - cycles

- pass reference records to next cycle
- a mechanism to stop the pipeline

Restarting Lookup



need something to hold reference records between two callpipes

Restarting Lookup

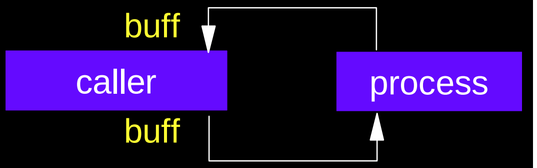
hold reference records between callpipes

- REXX stemmed variable (expensive)
- use disk files
- solution from the Piper:
  use a buffer in an affixed pipeline

Affixed Pipeline

- created by addpipe
- takes input from an output of the caller
- feeds output into an input of the caller
- runs independent from the caller

```
/* caller */
addstream both buff
addpipe
    *.output.buff:
  | process
  | *.input.buff:
```



Buffer in affixed Pipeline

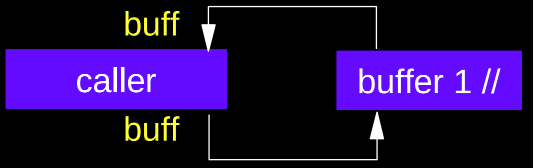
write to buffer:

```
| append strliteral //
| *.output.buff:
```

```
/* caller */
addstream both buff
addpipe
    *.output.buff:
  | buffer 1 //
  | *.input.buff:
```

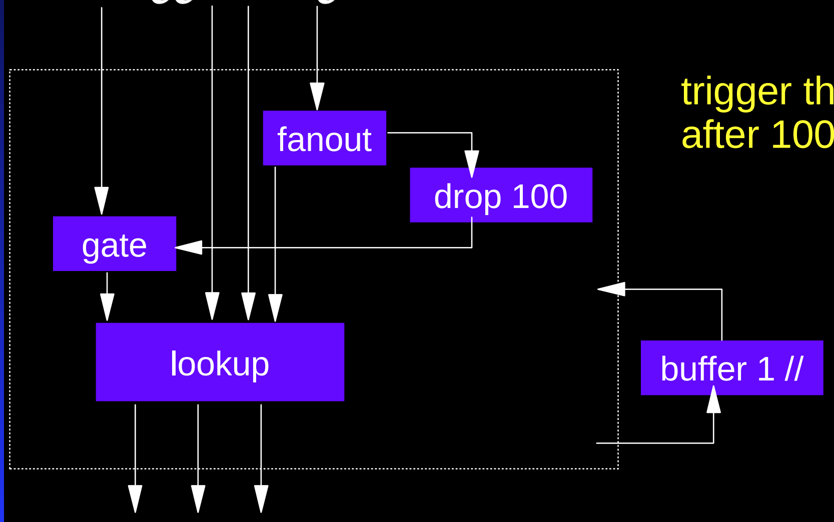
read from buffer:

```
\ *.input.buff:
| t: totarget nlocate 1
...
\ t: | drop
```



Stopping the lookup stage

- use a gate in the primary input stream
- trigger the gate with the deleted masters



trigger the gate
after 100 deletes

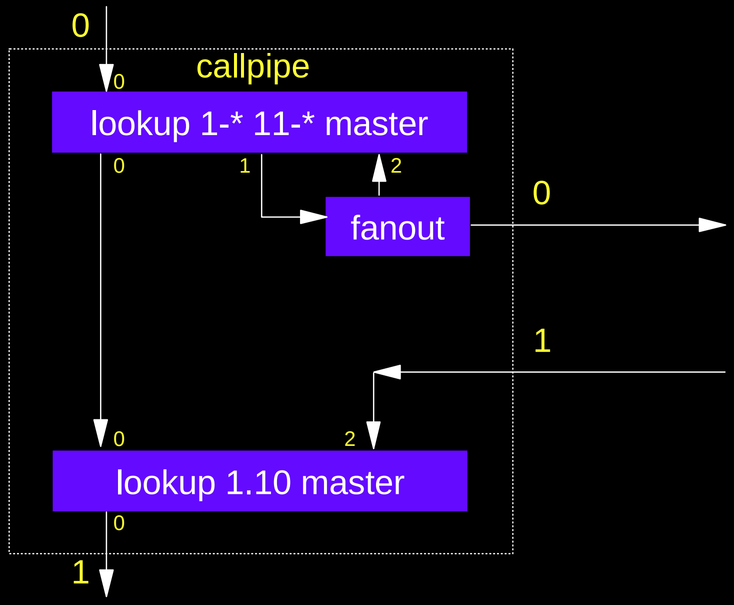
A Self-Cleaning Cache

- strategy for deleting entries
- deleting entries does not release storage

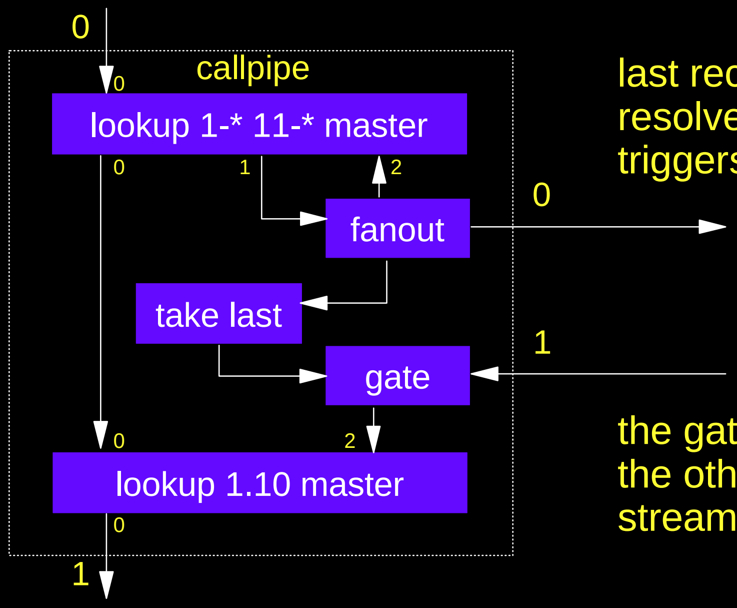
- alternative approach:
  - restart lookup stage now and then
  - reload a subset of the reference table

- change the addpipe into a callpipe
  - propagate End-of-File
  - use affixed pipeline to hold the table

Propagate End-of-File Forwards



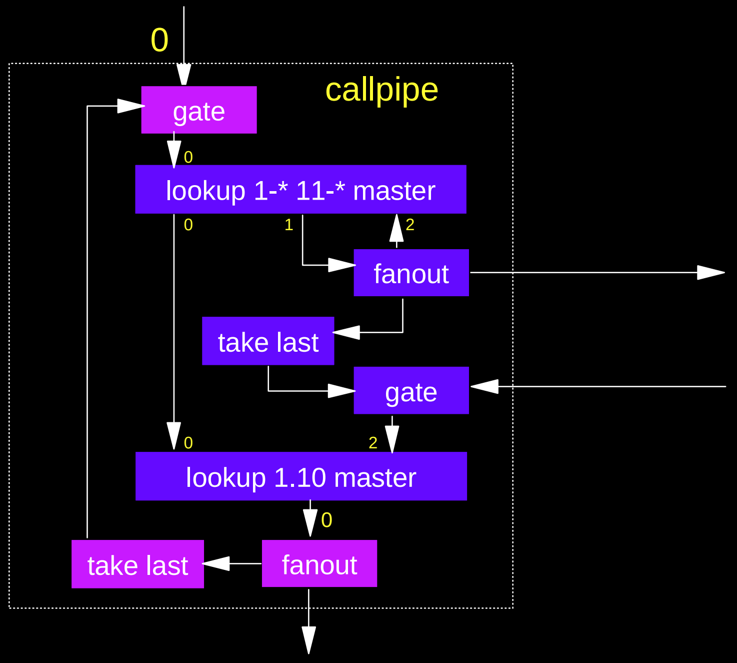
Propagate End-of-File Forwards

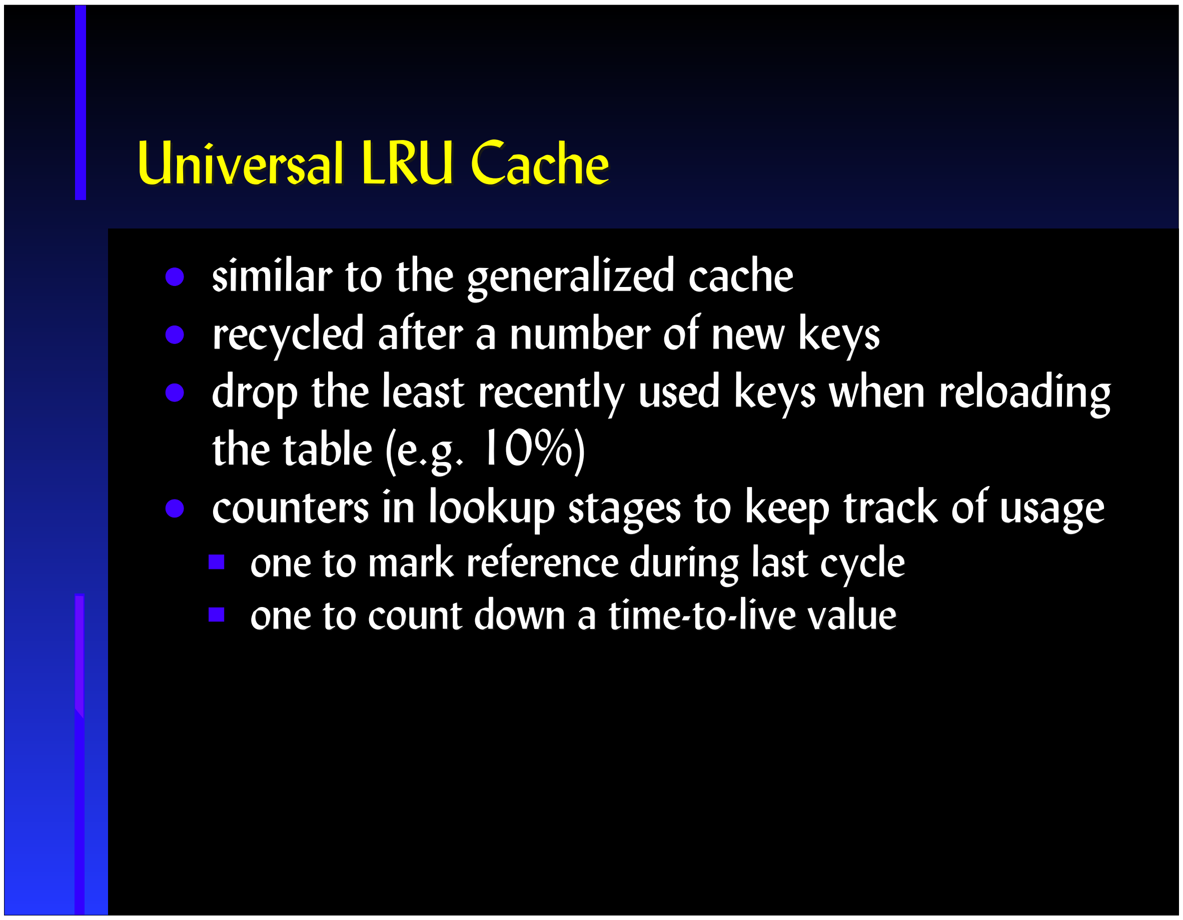


last record to the resolver pipeline triggers the gate

the gate severs the other input stream of lookup

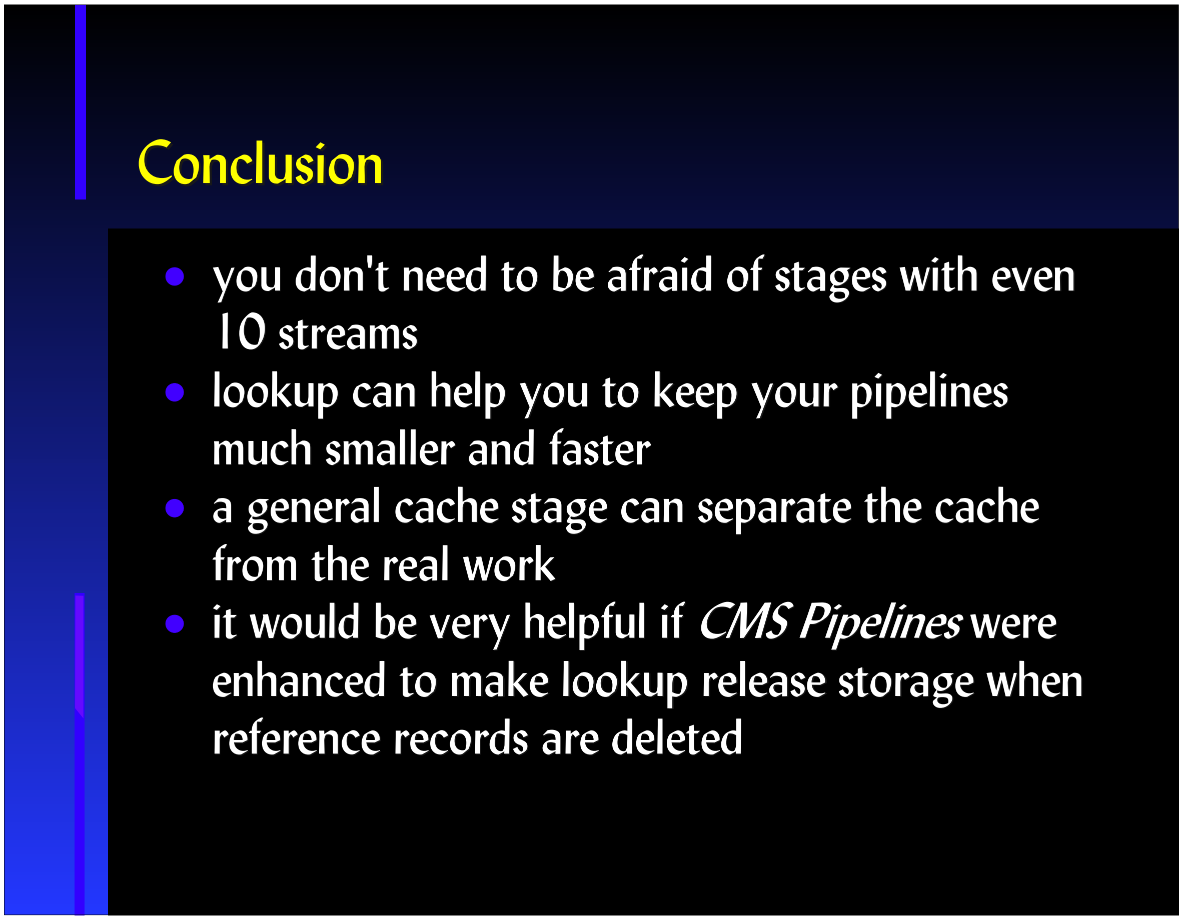
Propagate End-of-File Backwards





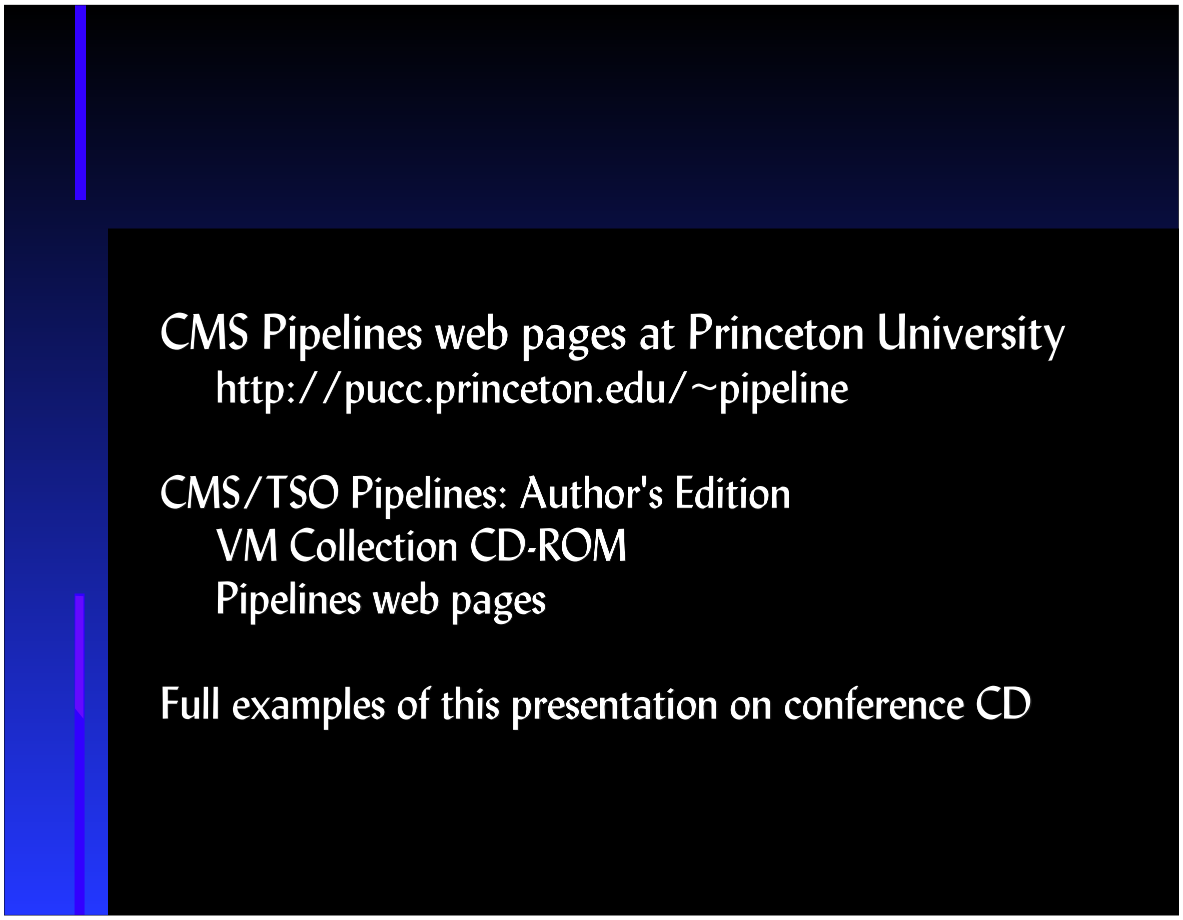
Universal LRU Cache

- similar to the generalized cache
- recycled after a number of new keys
- drop the least recently used keys when reloading the table (e.g. 10%)
- counters in lookup stages to keep track of usage
  - one to mark reference during last cycle
  - one to count down a time-to-live value



Conclusion

- you don't need to be afraid of stages with even 10 streams
- lookup can help you to keep your pipelines much smaller and faster
- a general cache stage can separate the cache from the real work
- it would be very helpful if *CMS Pipelines* were enhanced to make lookup release storage when reference records are deleted



CMS Pipelines web pages at Princeton University
- http://pucc.princeton.edu/~pipeline

CMS/TSO Pipelines: Author's Edition
- VM Collection CD-ROM
- Pipelines web pages

Full examples of this presentation on conference CD