Nutch

이채현
xlos21@gmail.com

nutch란?

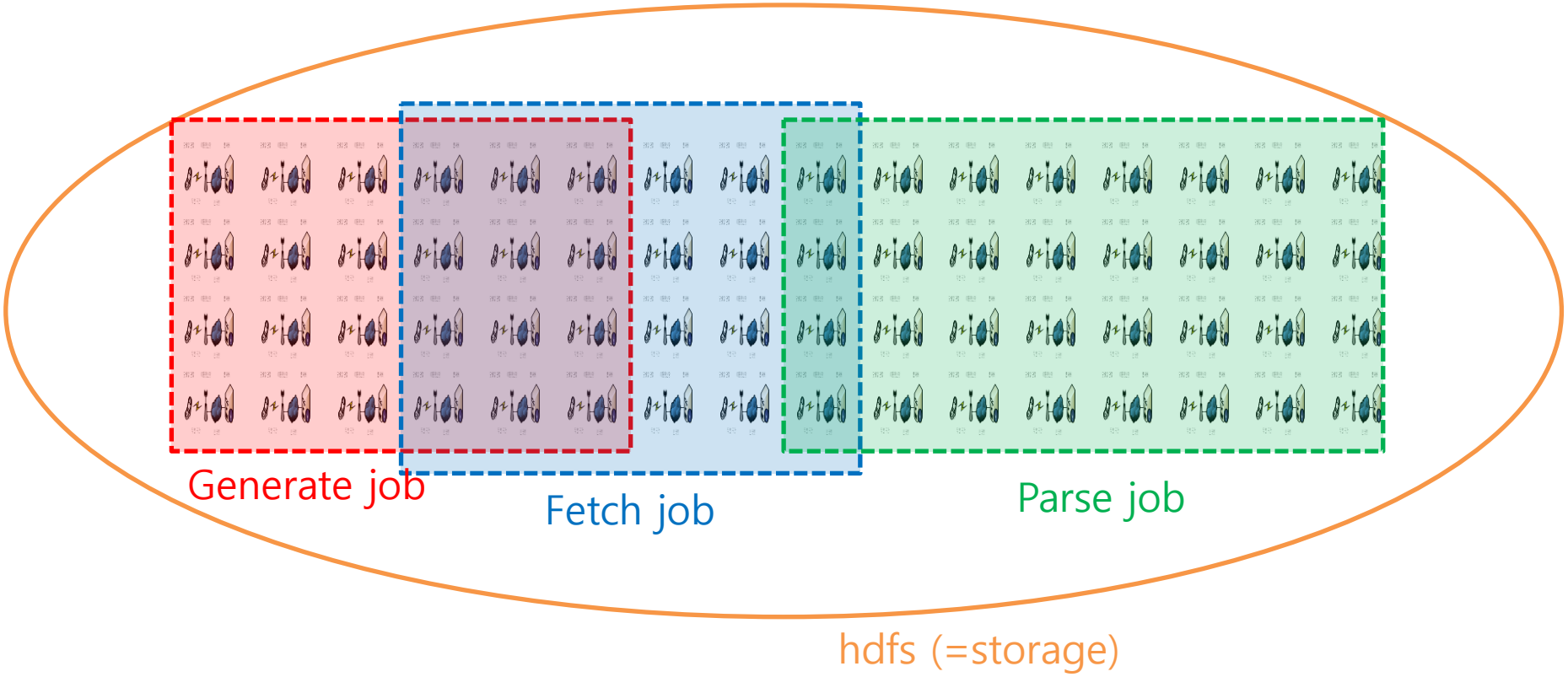
- 2000년 Lucene 프로젝트 시작
  - open source information retrieval software library
  - by Doug Cutting
- 2003년 Nutch 프로젝트 시작
  - by Doug Cutting and Mike Cafarella.
  - Lucene을 이용한 search engine을 만들어보자!
  - scheduler, fetcher, parser, indexer ...
- 분리
  - NDFS (Nutch Distributed File System) → Hadoop
  - Tika (content analyzer)
- 특징
  - no location server
  - highly scalable and robust
  - politeness & quality

Contents

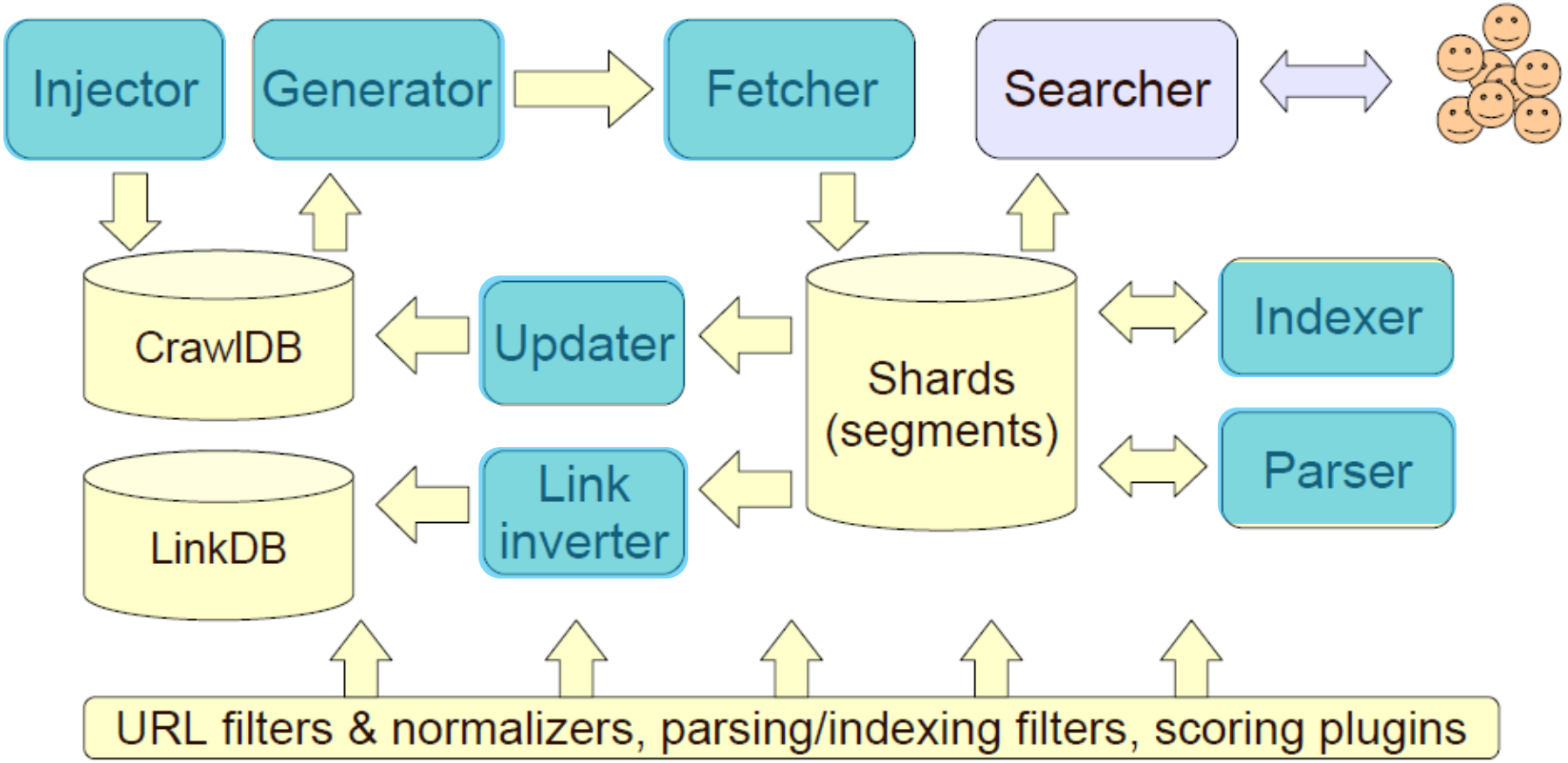
- Nutch의 대략적인 구조
- Nutch의 주요 알고리즘
  - generate, fetch, parse, index
- Data Structure
  - CrawlDB, LinkDB, CrawlDatum, ParseData
- Scoring
- Nutch의 장단점

Conceptual view of Nutch

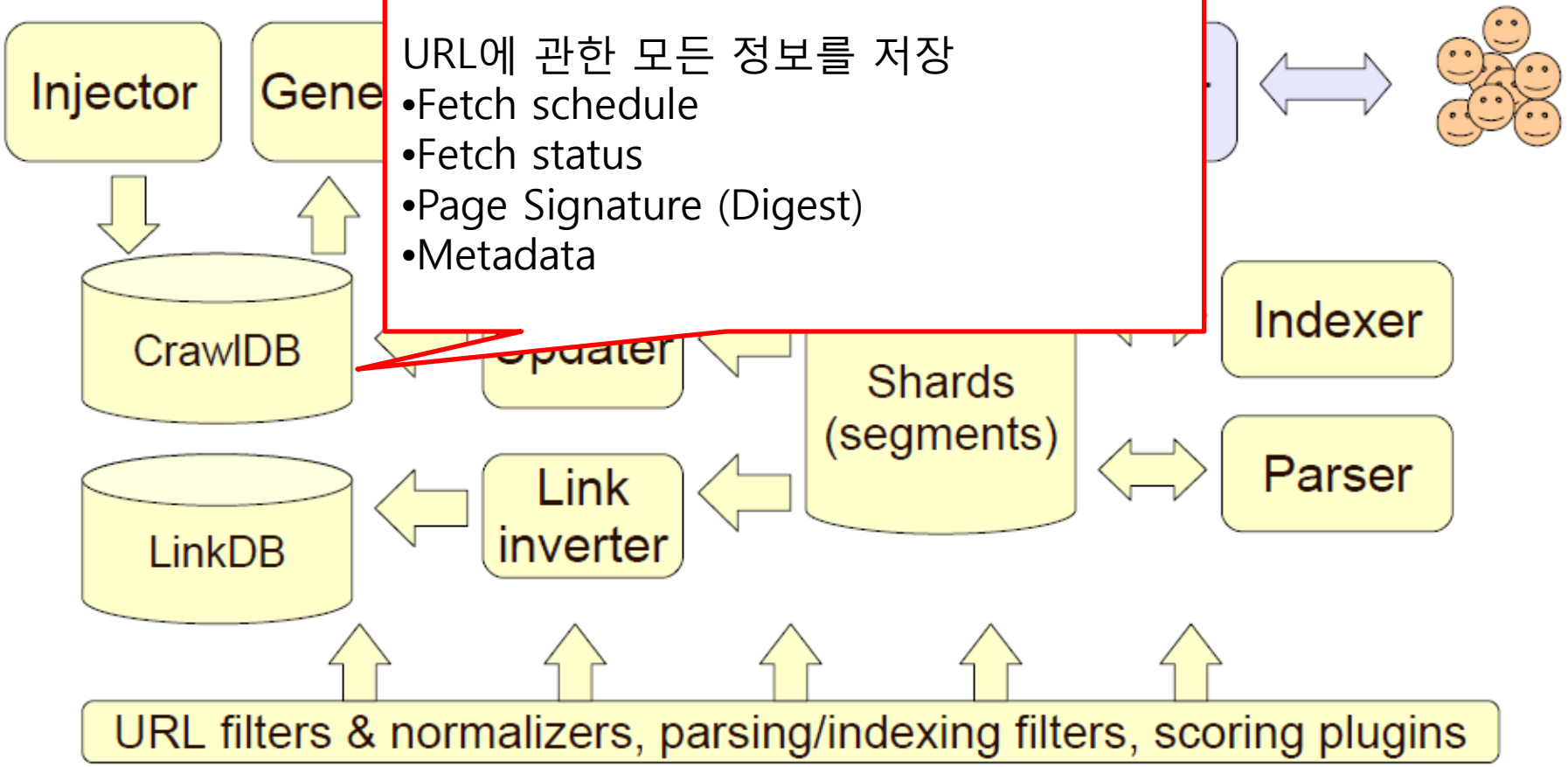
- nutch



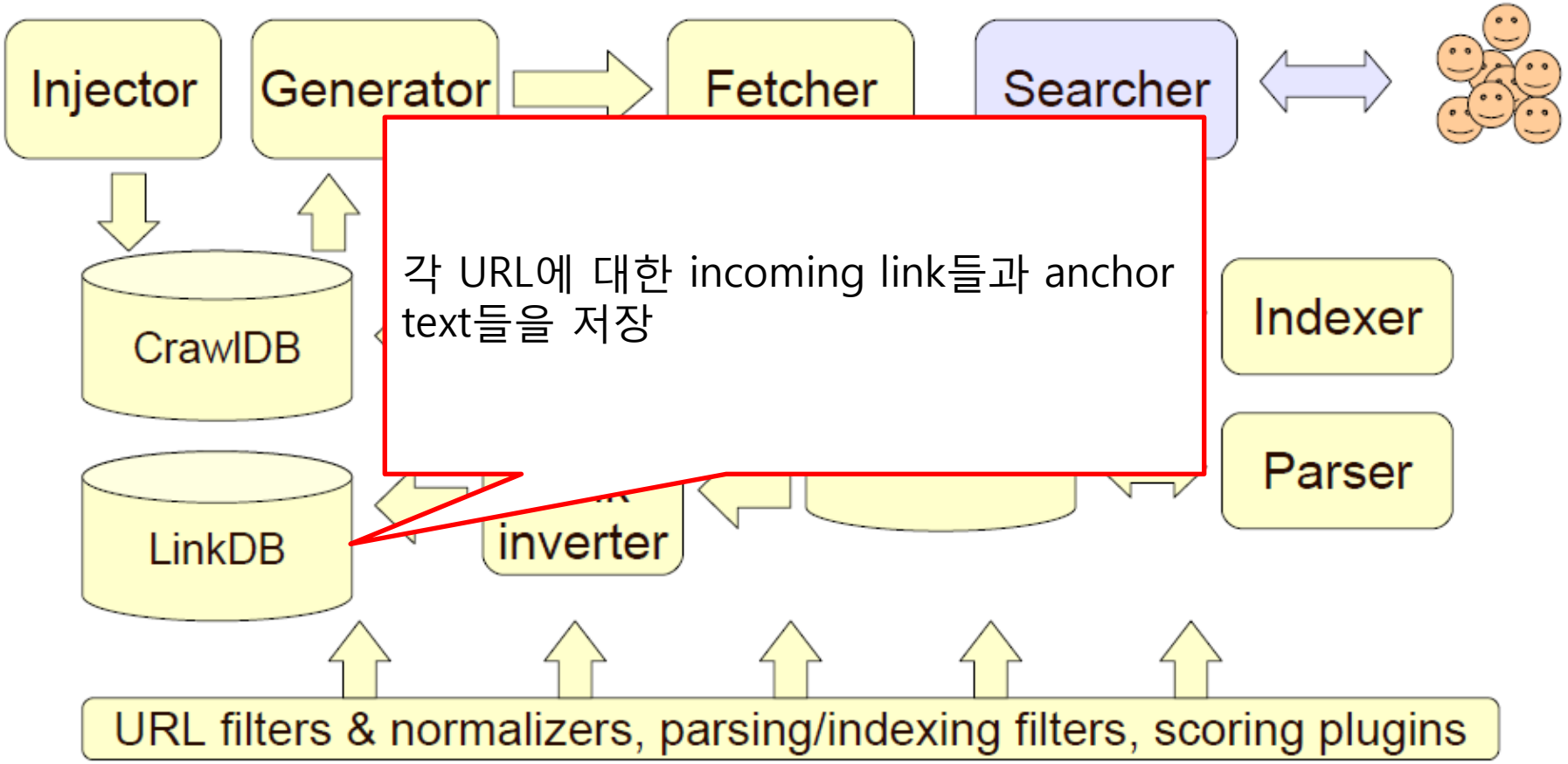
Nutch의 구조 (1/4)


Injector
Generator
Fetcher
Searcher
CrawlDB
Updater
Shards
(segments)
Indexer
LinkDB
Link
inverter
Parser
URL filters & normalizers, parsing/indexing filters, scoring plugins

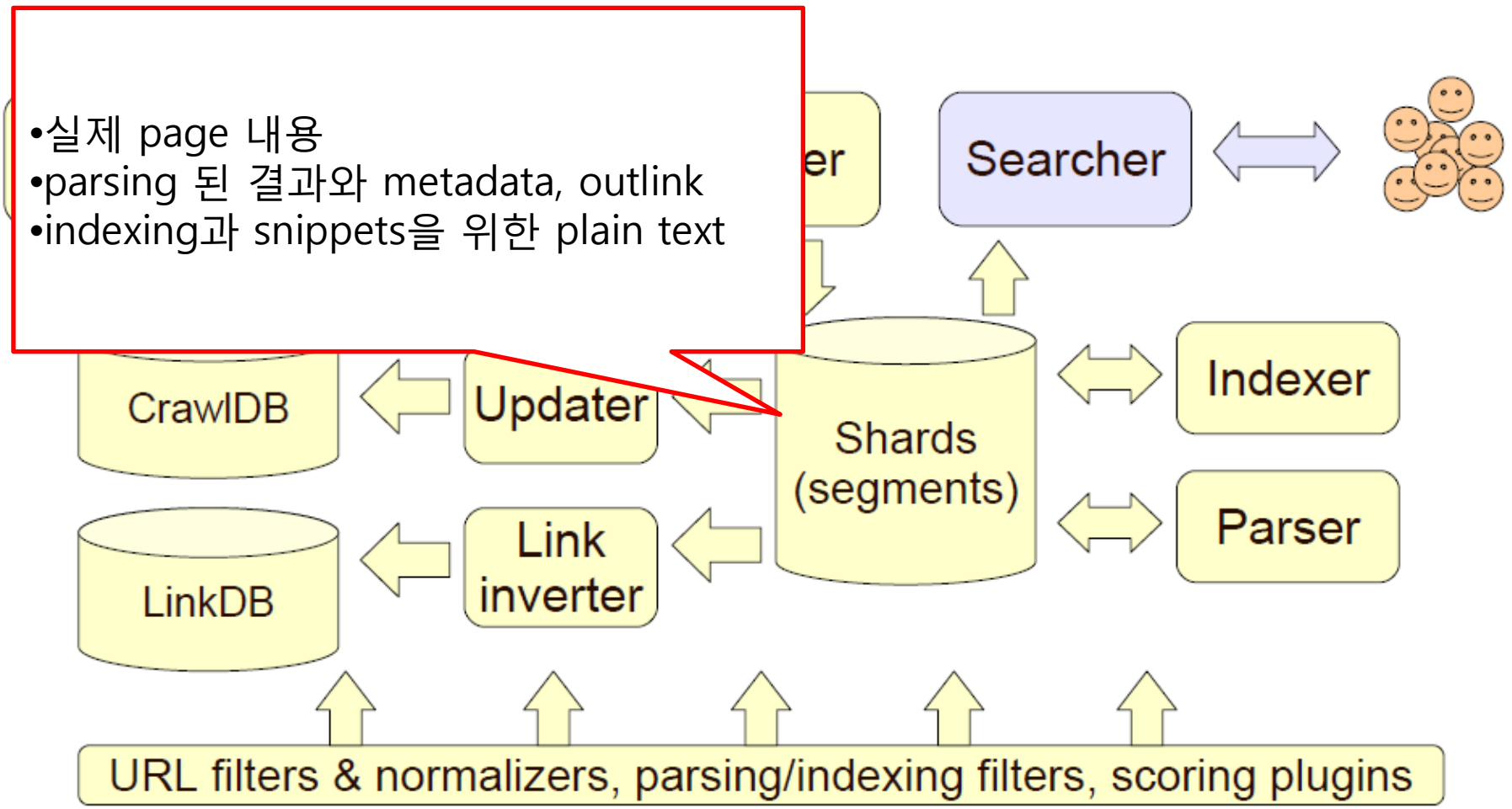
Nutch의 구조 (2/4)


URL에 관한 모든 정보를 저장
•Fetch schedule
•Fetch status
•Page Signature (Digest)
•Metadata
Injector
Gene
CrawlDB
Shards
(segments)
Indexer
Parser
Link
inverter
LinkDB
URL filters & normalizers, parsing/indexing filters, scoring plugins

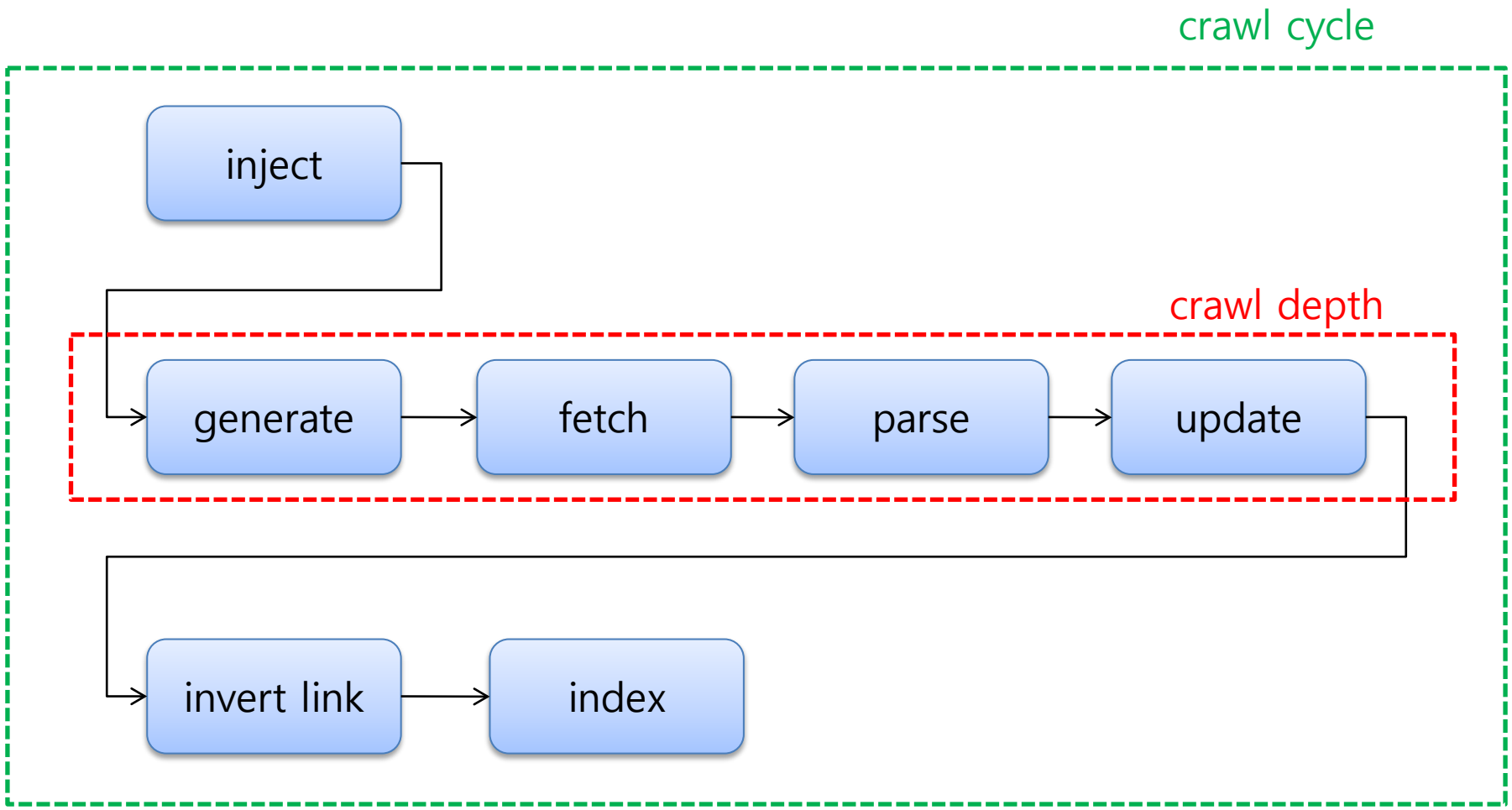
Nutch의 구조 (3/4)


Injector
Generator
Fetcher
Searcher
CrawlDB
LinkDB
inverter
Indexer
Parser
각 URL에 대한 incoming link들과 anchor text들을 저장
URL filters & normalizers, parsing/indexing filters, scoring plugins

Nutch의 구조 (4/4)


•실제 page 내용
•parsing 된 결과와 metadata, outlink
•indexing과 snippets을 위한 plain text
er
Searcher
CrawlDB
Updater
Shards
(segments)
Indexer
Parser
LinkDB
Link
inverter
URL filters & normalizers, parsing/indexing filters, scoring plugins

Nutch의 수집 단계



MapReduce in Nutch – Doug Cutting '05

Data Structure : Crawl Datum

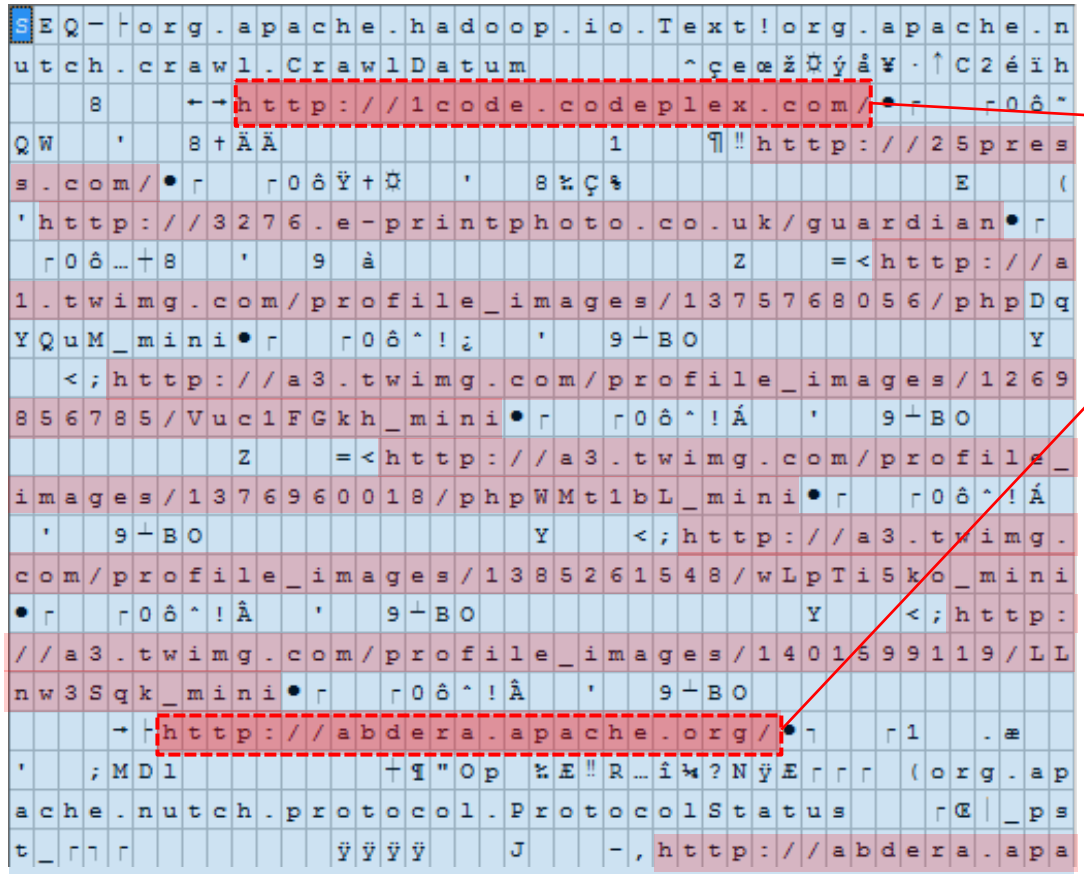
- http://nutch.apache.org/about.html Version: 7

field	value
Status	1 (db_unfetched)
Fetch time	Tue Jul 05 14:45:28 KST 2011
Modified time	Thu Jan 01 09:00:00 KST 1970
Retries since fetch	0
Retry interval interval	2592000 seconds (30 days)
Score	1.2089841
Signature	77adfc117e5ba9c0c29274c89c86753a
Metadata	_pst_: success(1), lastModified=0

- Status
  - db_unfetched, db_fetched, db_gone, linked,
  - fetch_success, fetch_fail, fetch_gone

Data Structure : Crawl DB internal

URL | CrawlDatum

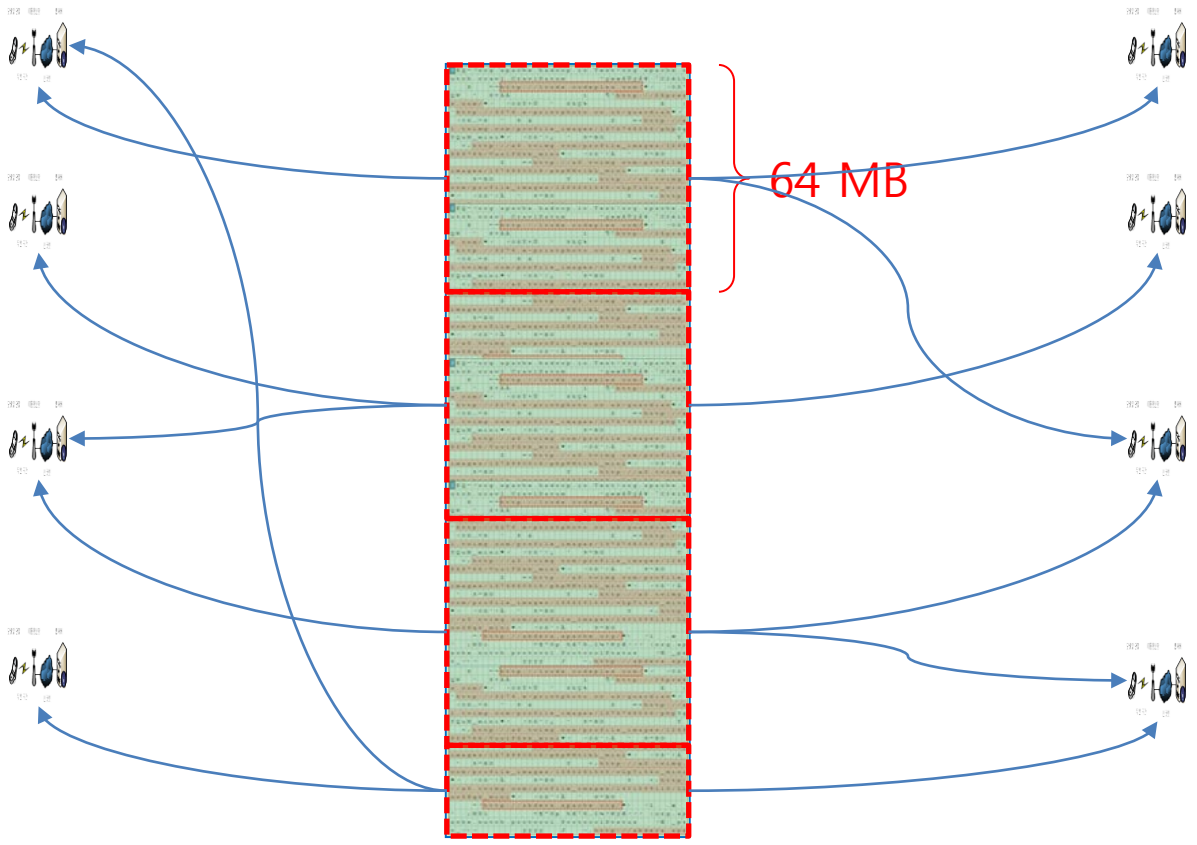


data file

url	offset
url1	23
url2	79
url3	123
url4	168
url5	198

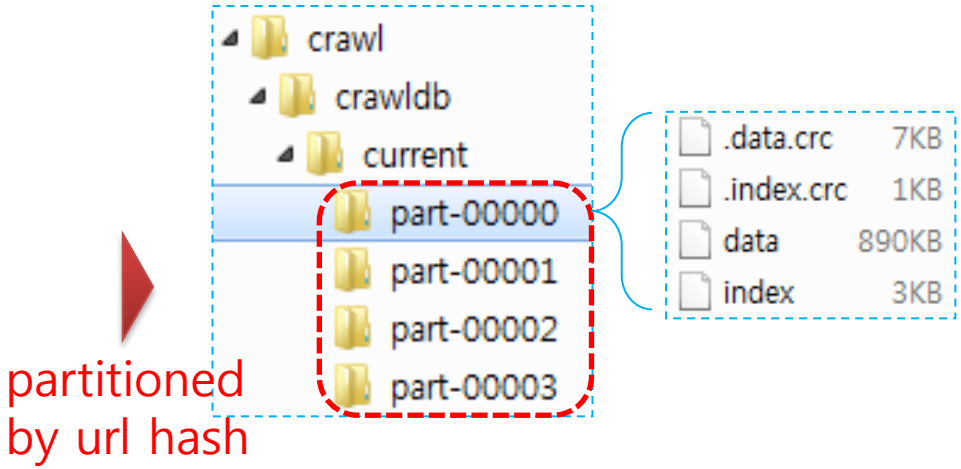
index file

Data Structure : Crawl DB file distribution



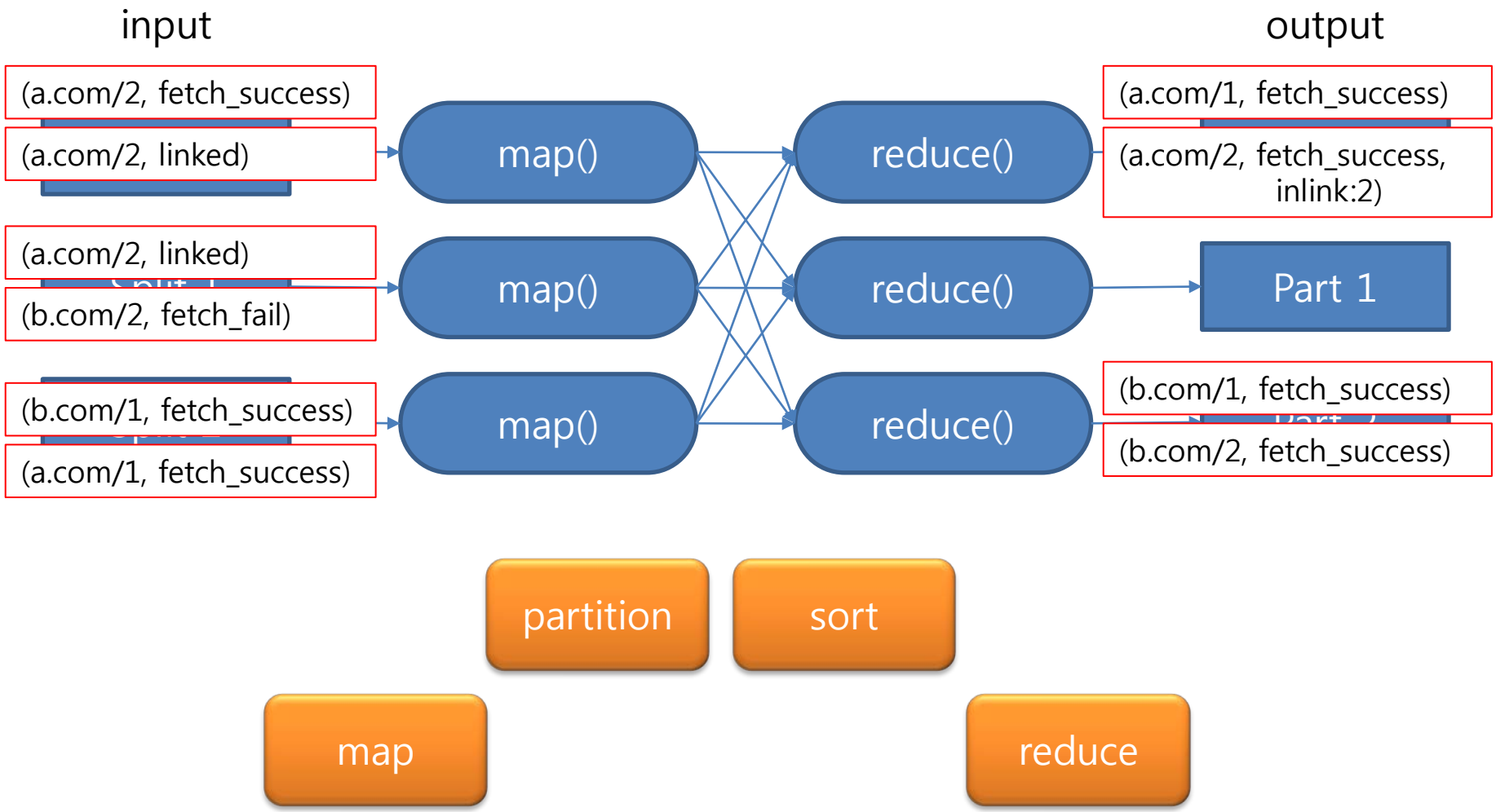
data file
dfs.block.size=64MB
dfs.replication=3

Data Structure : Crawl DB



- DBMS 수준은 아니고, 단순한 파일들
  - 굉장히 제한적인 random access
  - update 불가

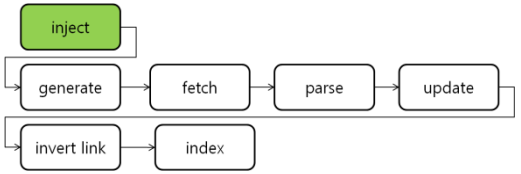
Map Reduce Diagram



Map Reduce in Nutch

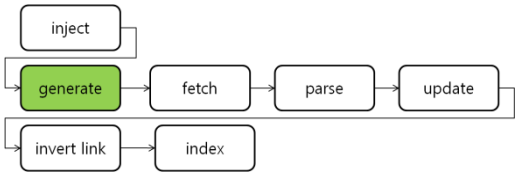
- Score가 높은 순으로 방문??
- Politeness를 지키자??
- Score 계산은 어떻게??

Algorithm: Inject



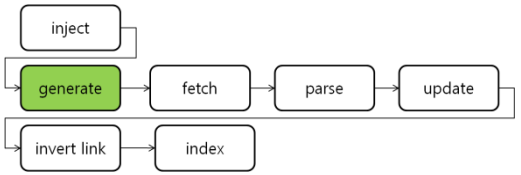
- Crawl DB에 url을 삽입
- MapReduce1 : Input Url을 DB format으로 변환
  - input : url list
  - Map (line) → <url, CrawlDatum>
  - Reduce() : identity
  - output : 임시 파일들의 디렉토리
- MapReduce2: 기존 Crawl DB와 merge
  - input: 첫 번째 job의 임시 결과와 기존 DB 파일들
  - Map() : identity
  - Reduce() : CrawlDatum을 하나의 entry로 합침
  - output : 새로운 버전의 Crawl DB

Algorithm: Generate (1/2)



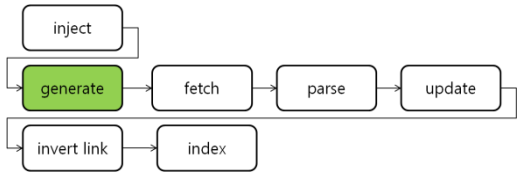
- CrawlDB에서 방문할 URL을 선정
  - 방문 날짜를 넘긴 URL 중 Score가 높은 N 개를 선택
- MapReduce1: fetch할 대상 url을 선정
  - input : Crawl DB files
  - Map() : if 방문날짜≥now, invert to <CrawlDatum, url>
    - **partitioned by url hash(!) to randomize**
    - **sorted by score**
  - Reduce()
    - score 순으로 정렬하고, N / reduceTask 만큼을 출력
    - 엄밀하게 score 순으로 N 개를 추출하지는 않음
  - output : 방문할 url N개 list <CrawlDatum, url>

Algorithm: Generate (2/2)

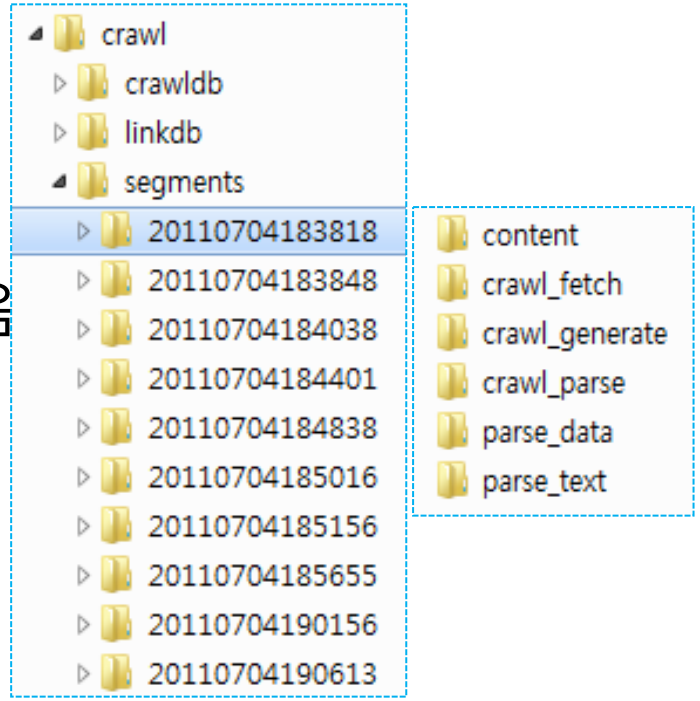


- MapReduce2: fetch를 준비
  - input : score 순으로 정렬된 <CrawlDatum, url> N개
  - Map() : invert to <url, CrawlDataum>
    - **partition by host for politeness**
    - **sorted by url hash**
  - Reduce() : identity
    - reduce task의 개수는 fetcher의 개수만큼
  - output: fetch할 url list 파일들<url,CrawlDatum>
    - 동일 host의 url들은 동일한 file로
    - url은 랜덤하게 정렬
    - fetcher의 개수만큼 파일로 나뉨
- 결과는 해당 시각의 Segment 디렉토리에 저장

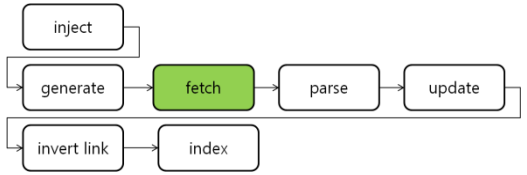
What is Segment?



- nutch의 기본적인 workflow 단위
  - 생성 시각을 디렉토리 이름으로 사용
- 실제 fetch, parsing 결과 저장
  - 한 번 작업이 완료되면 수정되지 않음
  - 일정 시점이 지나면 삭제
- cached view, snippet 서비스 제공
  - segment 이름을 index에 포함
  - 해당 서비스를 제공하지 않을 경우, indexing 후 삭제 가능

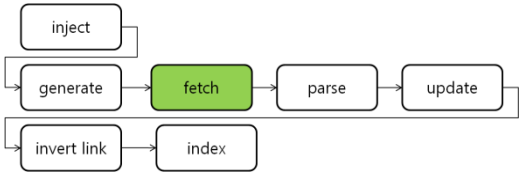


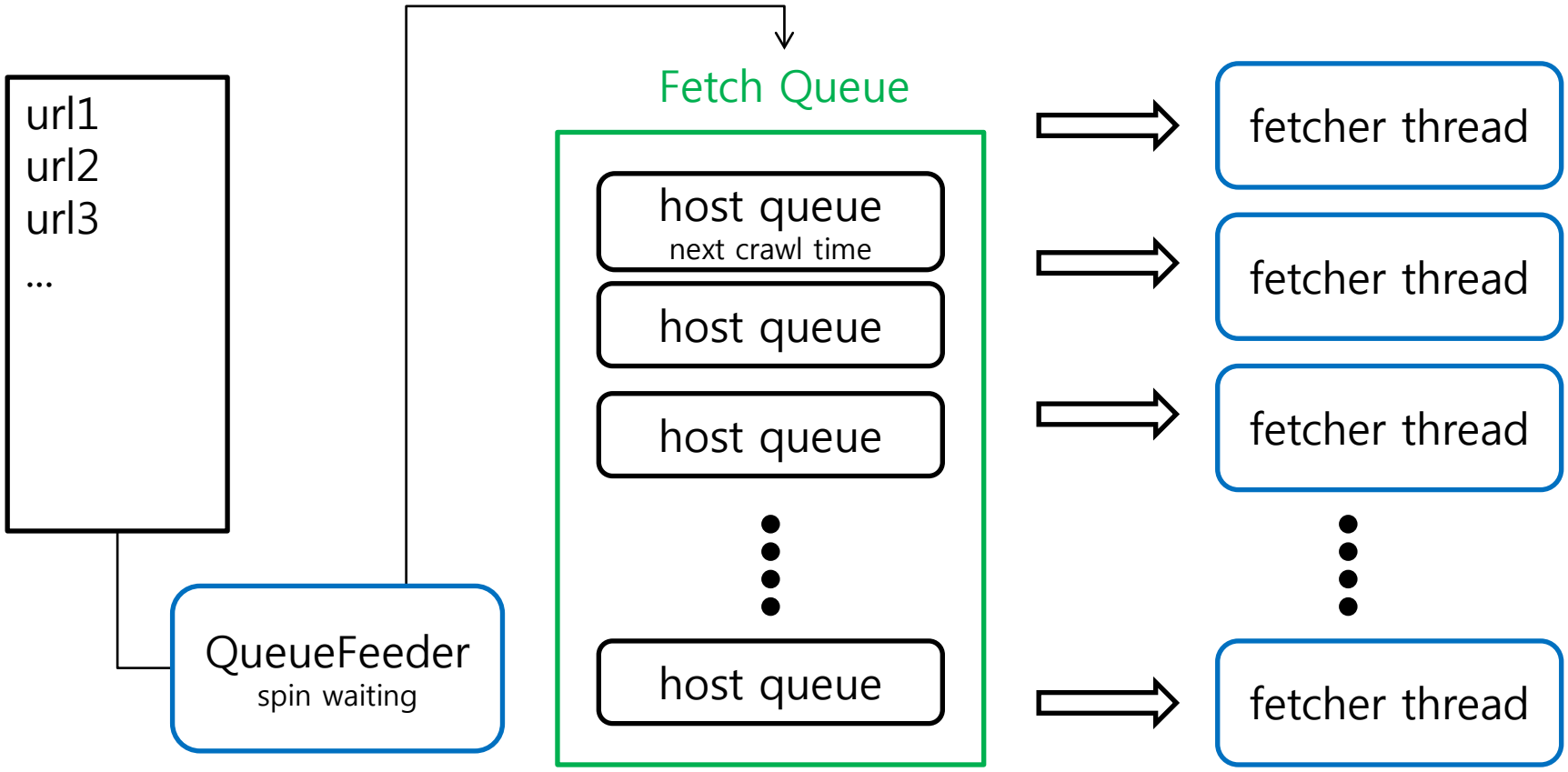
Algorithm: Fetch



- MapReduce: 페이지를 방문하여 content 저장
  - input: <url, CrawlDatum>
    - partitioned by host, sorted by url hash
    - score 순서대로 방문하지 않음
  - Map(url, CrawlDatum) → <url, FetcherOutput>
    - multi-threaded, async map implementation
    - FetcherOutput : <CrawlDatum, Content>
  - Reduce() : identity
  - output : 두 개의 파일
    - <url, CrawlDatum> : 문서 상태를 update
    - <url, Content> : raw html을 저장

Fetcher for politeness

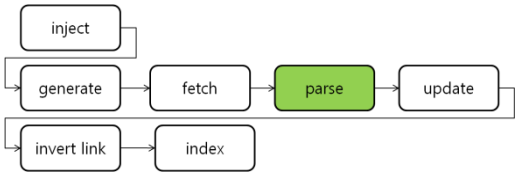




one producer – many consumer 구조

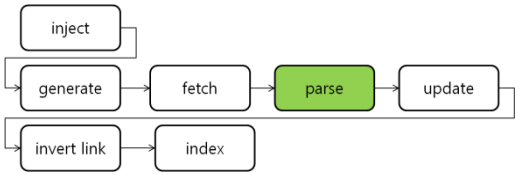
Q1) fetcher thread가 놀지는 않을까요?
Q2) 한 host에서 나온 url이 굉장히 많으면?

Algorithm: Parse



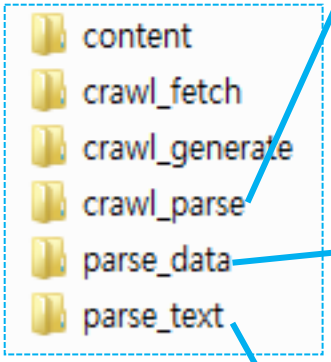
- MapReduce: parse content
  - input : <url, Content> files from Fetch
  - Map(url, Content) → <url, Parse>
    - Nutch parser plugins 사용
    - Parse: <ParseText, ParseData>
  - Reduce() : identity
  - output: 세 개의 파일로 저장
    - <url, CrawlDatum> for outlinks.
    - <url, ParseData>
    - <url, ParseText>

Result of Parser



<url, CrawlDatum>

```
http://xlos2.cafe24.com/test/pagea_a.html    Versi
Status: 67 (linked)
Fetch time: Tue Jul 05 15:35:37 KST 2011
Modified time: Thu Jan 01 09:00:00 KST 1970
Retries since fetch: 0
Retry interval: 2592000 seconds (30 days)
Score: 0.16666667
Signature: null
Metadata:
```



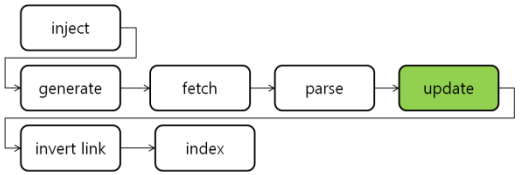
<url, ParseData>

```
http://xlos2.cafe24.com/test/pagea.html Version: 5
Status: success(1,0)
Title: page a
Outlinks: 2
  outlink: toUrl: http://xlos2.cafe24.com/test/index.html anchor: home
  outlink: toUrl: http://xlos2.cafe24.com/test/pagea_a.html anchor: pagea_a
Content Metadata: ETag="3e832f-b2-4a74bf09c3f00" Content-Length=178 Last-Mo
Parse Metadata: CharEncodingForConversion=utf-8 OriginalCharEncoding=utf-8
```

<url, String>

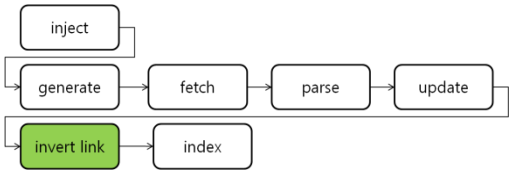
```
http://blog.foofactory.fi/2007/03/twice-speed-
http://clien.career.co.kr/  클리앙에 오신 것을 환영합니다. 로그인
http://nutch.apache.org/mailing_lists.html  Nutch Mailing Lists A
http://wiki.apache.org/nutch/  FrontPage - Nutch Wiki Search: Nu
http://www.apache.org/  Welcome to The Apache Software Foundation
http://www.apache.org/dyn/closer.cgi/nutch/ Apache Download Mirro
```

Algorithm: Update



- MapReduce: fetch 결과를 Crawl DB로 합침
  - input : <url, CrawlDatum>
    - 현재 Crawl DB, Segment내의 crawl 결과, parse 결과
  - Map() : identity
  - Reduce() : merges all entries into a single new entry
    - 기존 status 결과를 fetch 결과로 덮어씀
    - **score를 재계산**
  - output : 새로운 Crawl db <url, CrawlDatum>

Algorithm: Invert Links (1/2)



- ParseData를 뒤집어서 inlink를 계산, LinkDB에 update

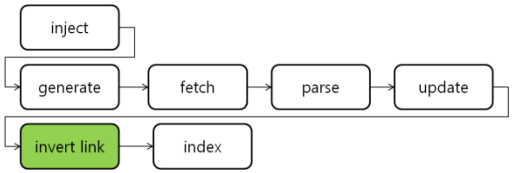
```
http://xlos2.cafe24.com/test/pagea.html Version: 5
Status: success(1,0)
Title: page a
Outlinks: 2
  outlink: toUrl: http://xlos2.cafe24.com/test/index.html anchor: home
  outlink: toUrl: http://xlos2.cafe24.com/test/pagea_a.html anchor: page
Content Metadata: ETag="3e832f-b2-4a74bf09c3f00" Content-Length=178 Last-
Parse Metadata: CharEncodingForConversion=utf-8 OriginalCharEncoding=utf-
```

ParseData

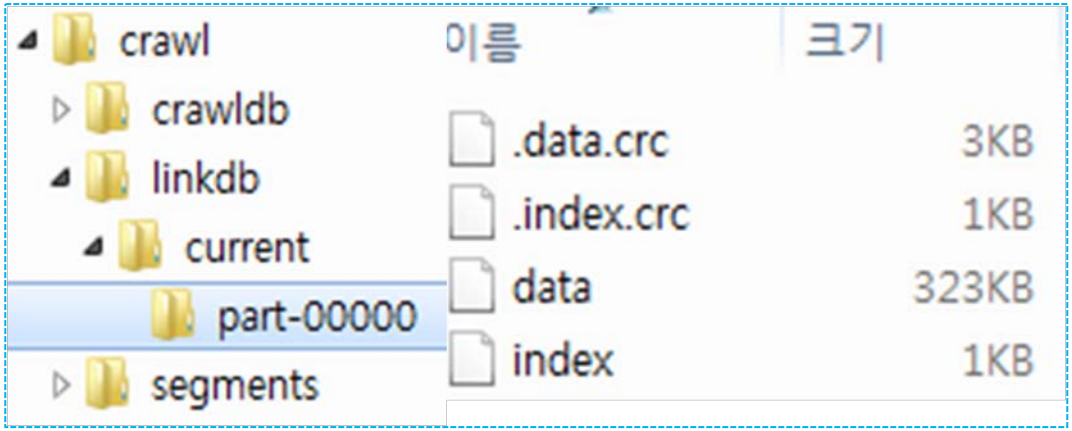
```
http://xlos2.cafe24.com/test/index.html Inlinks:
 fromUrl:http://xlos2.cafe24.com/test/pagea.html anchor: home
 fromUrl:http://xlos2.cafe24.com/test/pageb.html anchor: home
```

Inlinks

Data Structure : Link DB



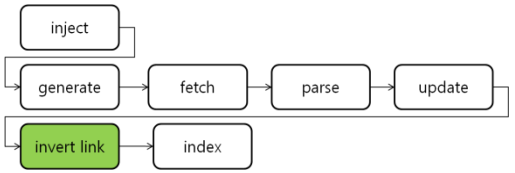
- Crawl DB와 마찬가지로 파일들의 집합



  - data : <URL, InLinks>을 저장, URL 순으로 정렬

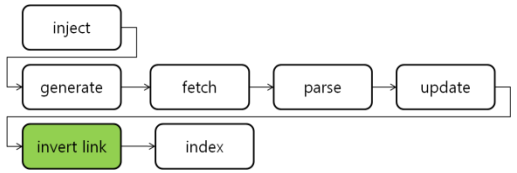
```
http://issues.apache.org/jira/browse/NUTCH  Inlinks:
 fromUrl: http://nutch.apache.org/ anchor: Jira
 fromUrl: http://nutch.apache.org/mailing_lists.html anchor: Jira

http://jackrabbit.apache.org/   Inlinks:
 fromUrl: http://www.apache.org/dyn/closer.cgi/nutch/ anchor: Jackrabbit
 fromUrl: http://www.apache.org/ anchor: Jackrabbit
```

Algorithm: Invert Links (2/2)



- MapReduce1: Segment의 ParseData에서 inlinks을 계산
  - input : <url,ParseData>
  - Map(srcUrl, ParseData> → <destUrl, Inlinks>
    - Inlinks: <srcUrl, anchorText>*
    - 각각의 outlink에 대해 하나의 Inlinks를 생성
    - 페이지 하나당 outlink를 제한
  - Reduce() appends inlinks
  - output : <url, Inlinks>
- MapReduce2: 기존 linkDB에 merge
  - input : 첫 번 째 job의 결과 + 기존 LinkDB
  - Map(url, Inlinks)
  - Reduce() appends inlinks
  - output : 새로운 linkDB

Algorithm: Index

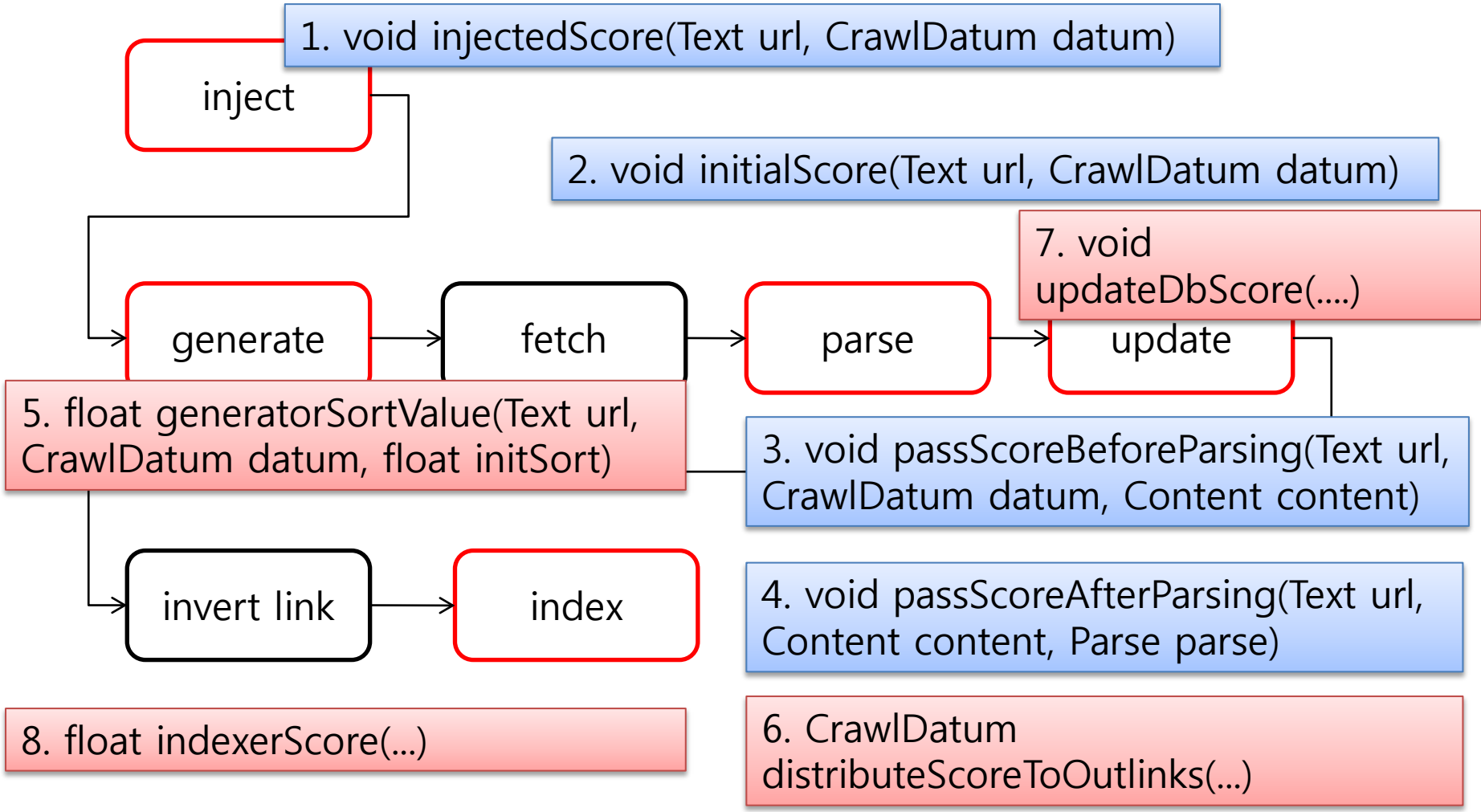


- MapReduce : Lucene indexes 생성
  - input : CrawlDB, LinkDB, 최근 수집한 segment들
    - <url, ParseData> from parse, for title, metadata, etc.
    - <url, ParseText> from parse, for text
    - <url, Inlinks> from invert, for anchors
    - <url, CrawlDatum> from fetch, for fetch date
  - Map() : identity
  - Reduce() : Lucene Document를 생성
    - 기존에 존재하는 Nutch indexing plugins 을 사용
  - output : build Lucene index; copy to fs at end

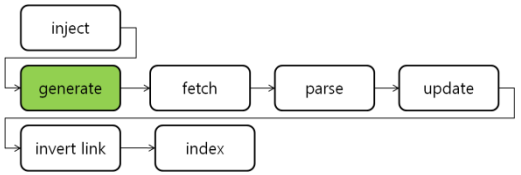
방문 Score 계산

- Nutch의 방문 우선 순위 기반은 score
- Score를 계산하기 위한 Scoring Filter Plug-in
  - 각 crawling 전략 마다 ScoreFilter interface 를 구현
  - 현재 nutch에는 **Adaptive OPIC 전략이 사용됨**
  - 두 개 이상의 ScoreFilter를 동시 적용 가능
  - ScoreFilter interface
    - 8개의 method
    - Breath first, OPIC 등 다양한 전략을 구현할 수 있는 수준

Scoring Filter (1/5)



Scoring Filter (2/5)

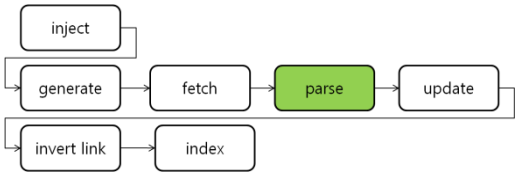


5. float generatorSortValue(
   Text url, CrawlDatum datum, float initSort)
   - generate 단계에서 점수를 가져오는 mapper에서 사용됨
     - 신규 방문 문서에 대해 우선 순위를 가중 ?
     - 방문 실패 횟수에 따라 우선 순위 조절 ?
     - 한 host에서 다량의 topN 문서가 나오는 것을 방지 ?

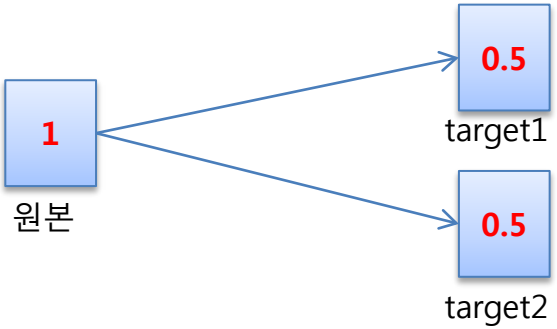
Url	Score
http://a.com/1.html	10
http://a.com/2.html	10
http://a.com/3.html	10
http://a.com/4.html	10
http://a.com/5.html	10
http://a.com/6.html	10
http://b.com/1.html	8
http://c.com/1.html	8

Url	Score
http://a.com/1.html	10
http://a.com/2.html	9
http://a.com/3.html	8
http://a.com/4.html	7
http://a.com/5.html	6
http://a.com/6.html	5
http://b.com/1.html	8
http://c.com/1.html	8

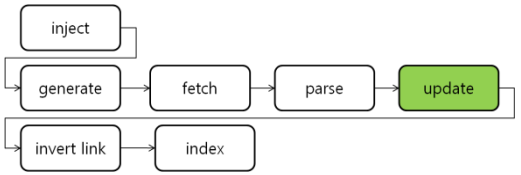
Scoring Filter (3/5)



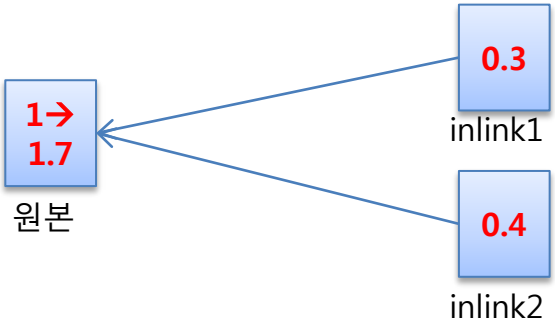
6. CrawlDatum distributeScoreToOutlinks(...)
   - 원본 문서의 점수를 어떻게 배분할 것인가
   - parameters
     - Text fromUrl
     - ParseData parseData
       - 현재 이 문서의 score를 저장
     - **Collection<Entry<Text, CrawlDatum>> targets**
       - 이 문서에서 나온 outlink들의 집합
     - CrawlDatum adjust
       - 이 문서의 원래 CrawlDatum 정보
     - int allCount
       - 원본 문서에서 나온 전체 outlink의 개수
       - targets의 개수는 제한될 수 있음



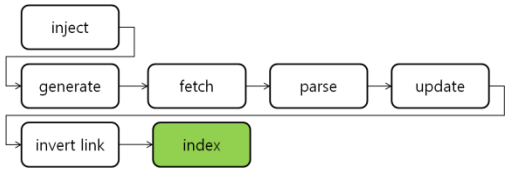
Scoring Filter (4/5)



7. void updateDbScore(...)
   - 기존 score와 inlink들의 score를 기반으로 score 재계산
   - parameters
     - Text url
     - CrawlDatum old
       - 문서의 이전 상태 정보
     - **CrawlDatum datum**
       - fetch 결과를 포함한 가장 최근 문서 정보
     - List<CrawlDatum> inlinked
       - 현재 update batch에서 찾은 inlink 정보들



Scoring Filter (5/5)



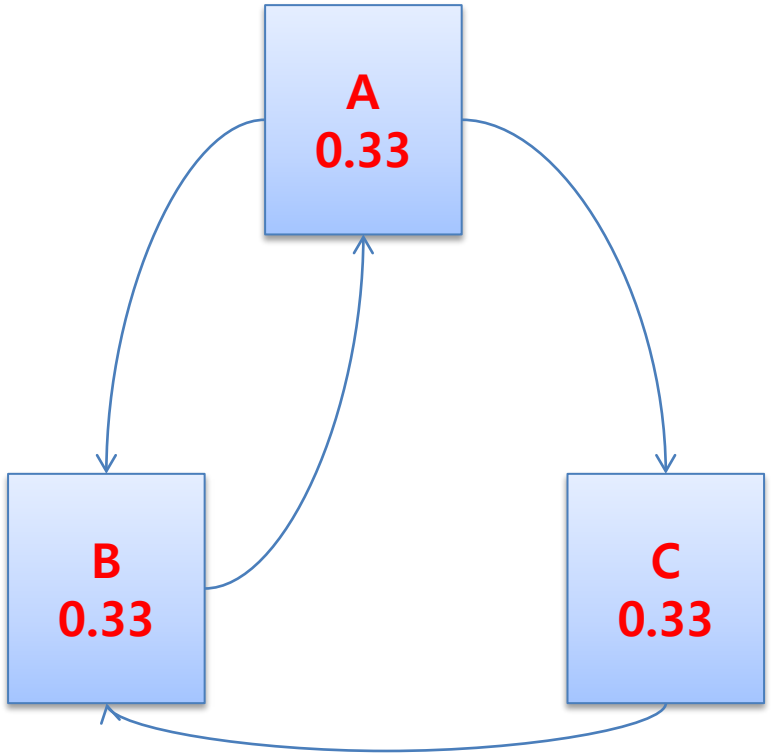
8. float indexerScore(...)
   - lucene에 전달할 boost 점수를 계산
   - parameters
     - Text url
     - **NutchDocument doc**
       - lucene document
     - CrawlDatum dbDatum
     - CrawlDatum fetchDatum
     - Parse parse
     - Inlinks inlinks
       - LinkDB로부터 가져온 모든 inlink 정보
     - float initScore

WebGraph (1/6)

- 기존 Adaptive OPIC
  - score가 계속적으로 증가하는 문제
- PageRank
  - 상호 링크 교환을 통한 abusing
- WebGraph
  - 위 두 가지 문제를 해결한 새로운 Scoring algorithm
  - 아직 nutch에 정식으로 적용되지는 않음

WebGraph (2/6)

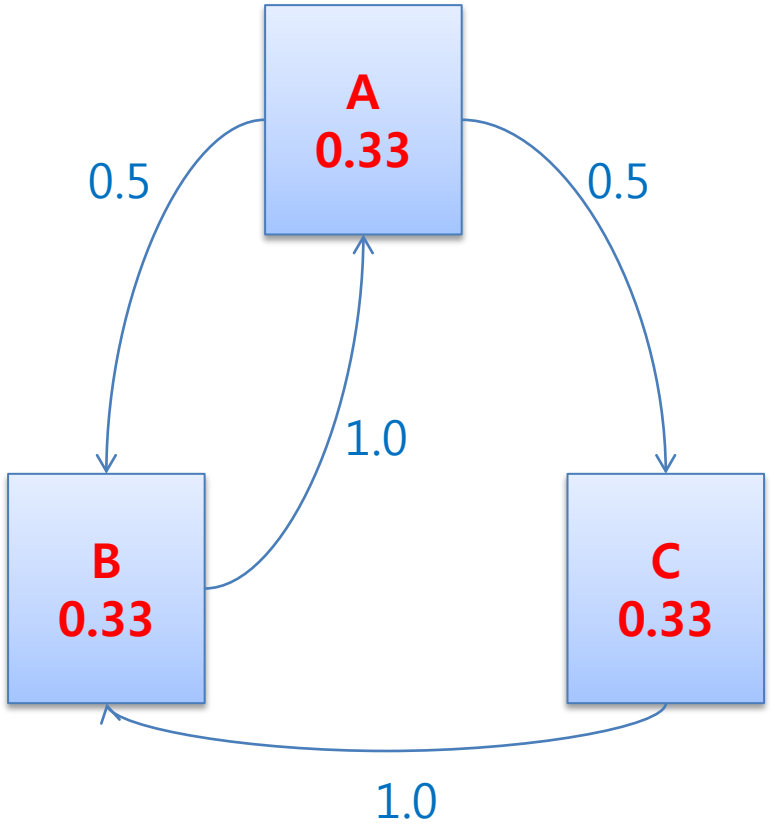
- 기본 개념
  - node: 각 문서
  - edge: link
  - RankOneScore (ROS)
    = 1/# of total node



WebGraph (3/6)

- Iteration 1
  - 초기 InlinkScore=1
  - OutlinkScore (OS) 계산

$$OS = \frac{InlinkScore}{\#OfOutLinks}$$

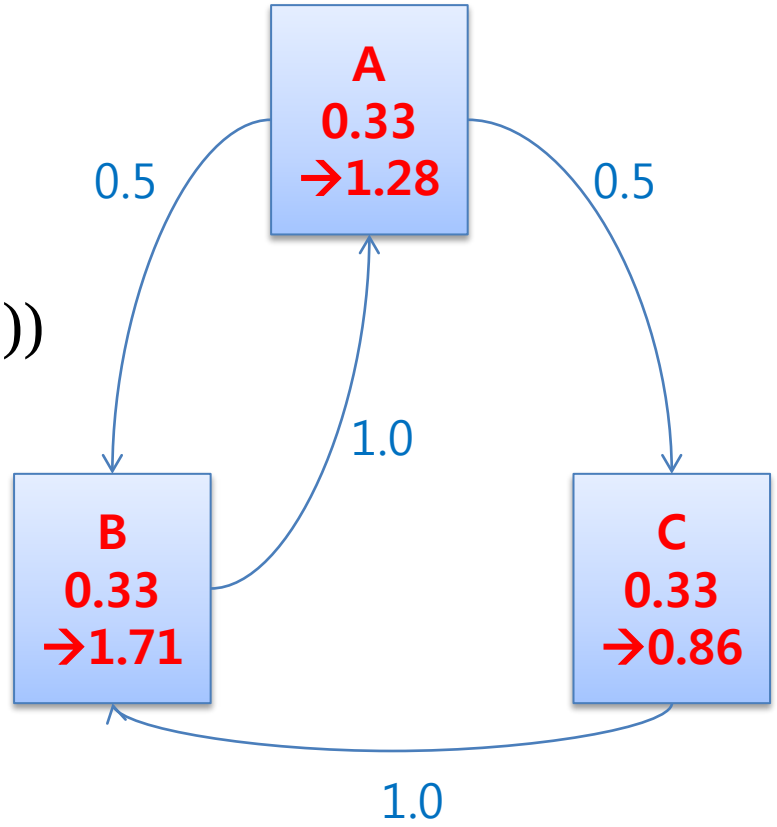


WebGraph (4/6)

- Iteration 1
  - InlinkScore (IS) 계산
  - **<u>이전 score는 고려하지 않음</u>**

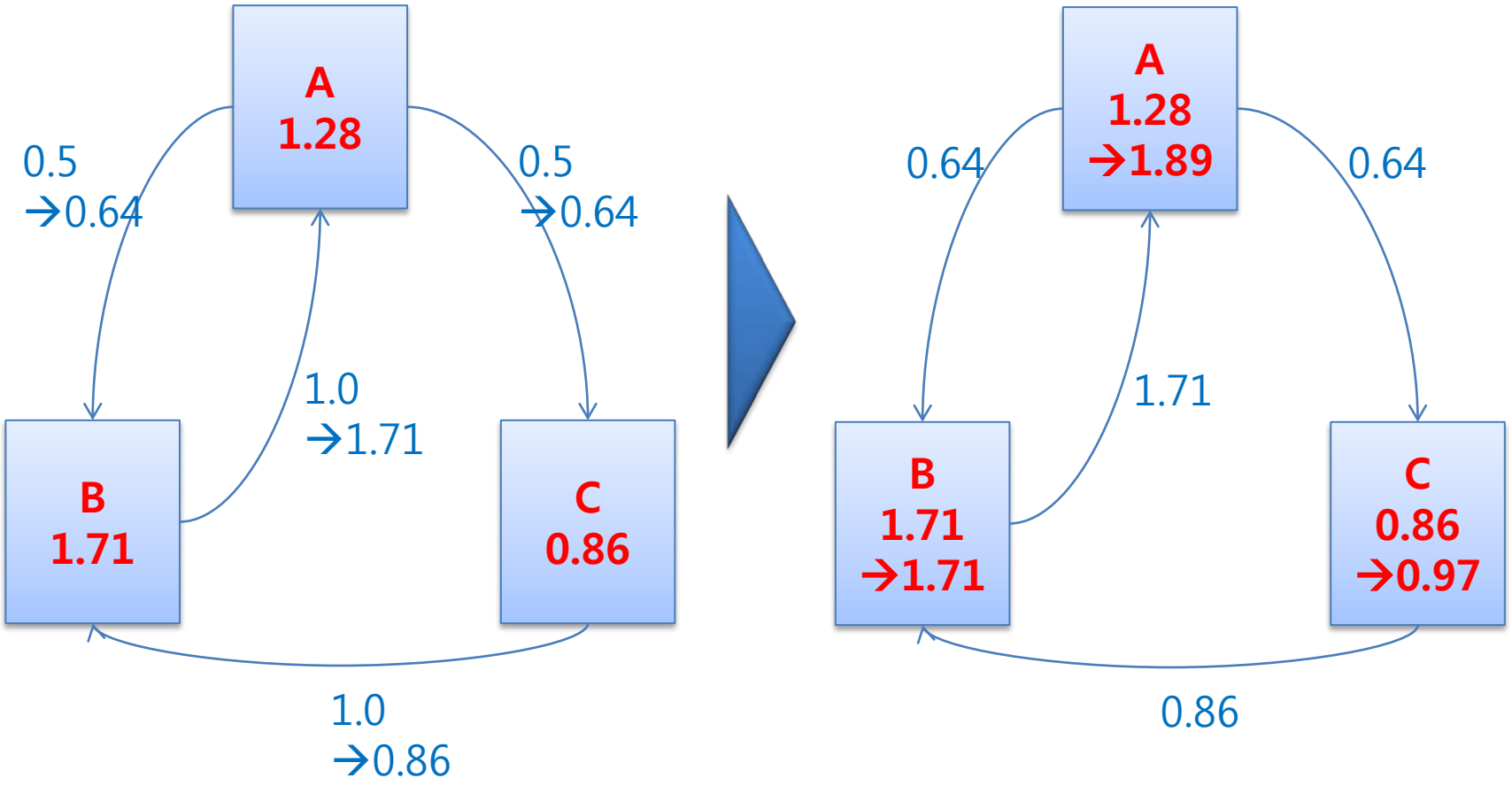
$$IS = (1 - df) + (df \times (\sum IS + ROS))$$

    - df (damping factor) = 0.85
    - ROS=0.33



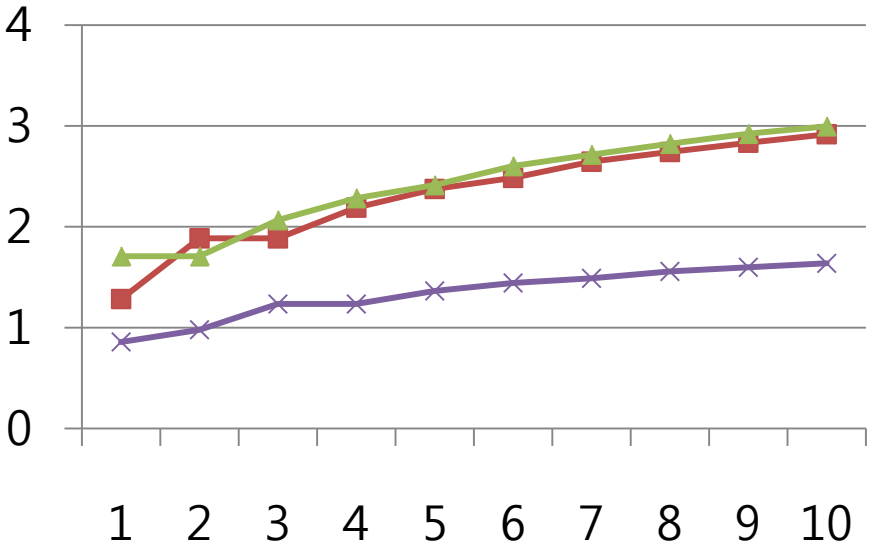
WebGraph (5/6)

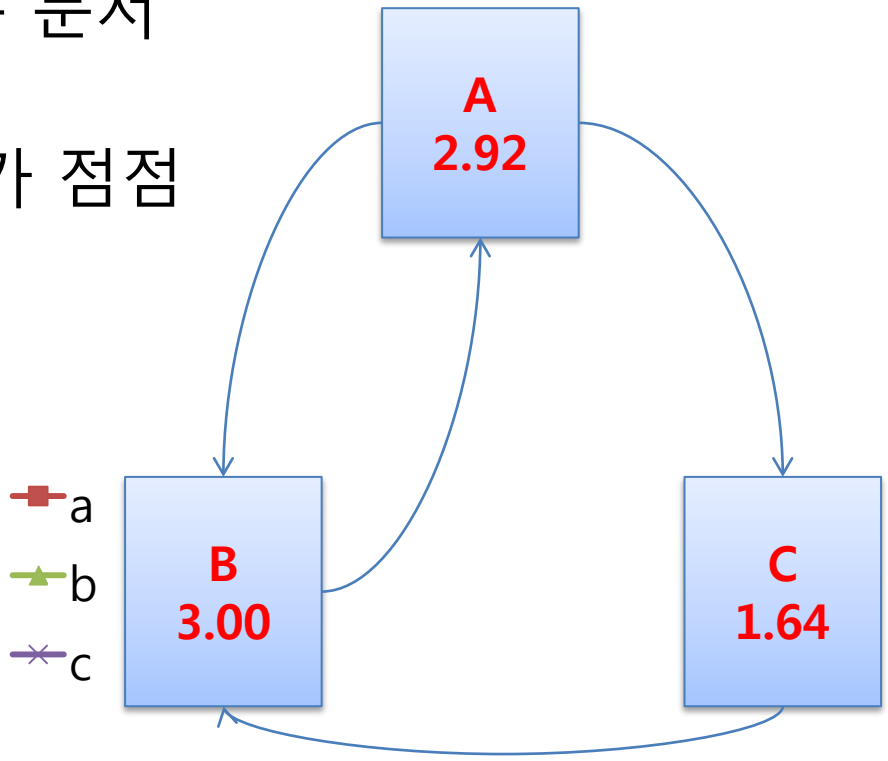
- Iteration 2



WebGraph (6/6)

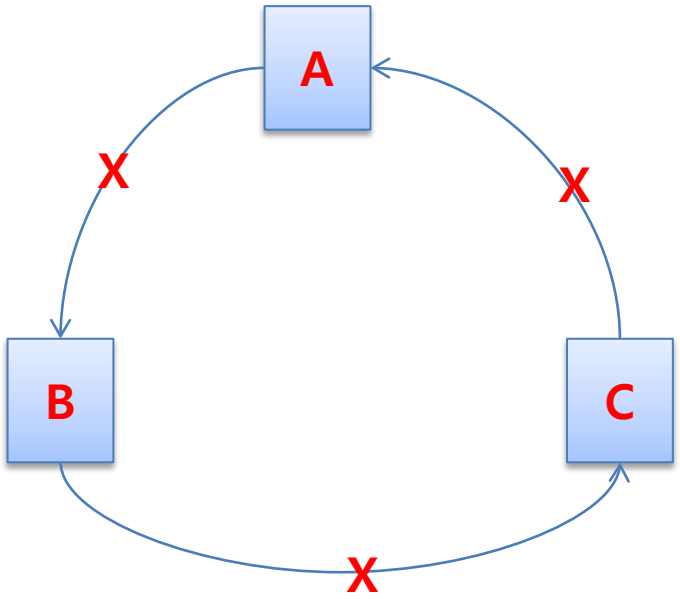
- Iteration 10회 후
  - 초기 값에 상관없이 좋은 문서(?)에 좋은 점수가 배분
  - iteration을 돌면서 점수가 점점 converge됨





Loops (optional)

- Spam site 제거
  - web graph에서 link cycle을 찾음
  - 비용이 매우 크기 때문에 3~4 depth 정도까지만 계산
    - 찾아진 loop는 web graph에서 제거
    - 가격대 성능비가 별로 이므로 비추



WebGraph 실전

- WebGraph
  - Inlink DB, Outlink DB, Node DB 생성
- Loops
  - A → B → C → A 와 같은 loop를 발견
  - **발견된 loop에 포함되는 link는 제외**
- LinkRank
  - iteration (default 10회)을 돌면서 점수를 계산
- ScoreUpdater
  - CrawlDB에 score를 업데이트
  - WebGraph에서 발견되지 않은 url의 score는 clear

기타 작업 – 중복 문서 처리

- 중복 문서 처리
  - 수집 단계에서는 중복 문서를 처리 안 함
  - index가 끝난 다음, 중복 문서들을 index에서 삭제
  - 원본 문서는 score(boost)가 가장 높은 문서

기타 작업 – 재방문 설정

- 기본적인 재방문 주기는 30일
- 페이지 변경 여부에 따라 방문 주기 변경
  - 페이지 변경 여부 : signature (RAW_DIGEST) 이용
  - 변경되었다면, 기존 주기 * 0.2 만큼 감소
    - 최소 60 초까지 감소
  - 변경되지 않았다면, 기존 주기 * 0.4만큼 증가
    - 최대 365일까지 증가
- db.fetch.interval.max 설정
  - 방문한지 90 일이 지나면 무조건 재방문

기타 작업 – Url Pattern 처리

- Url Normalizer
  - scope 분리 가능
    - partition / fetcher/ crawldb/ linkdb/ inject/ outlink ...
- Url Filter
  - file로 관리
  - domain, prefix, suffix, regex-urlfilter
    - -₩.(gif|GIF|jpg|JPG|png|PNG)$ // skip image file
    - 도메인 한정 수집
      # accept hosts in naver.com
      +http://([a-z0-9]*₩.)*naver.com/.*
      # skip everything else
      -.*

Nutch 2.0

- Web-graph & page repository -> ORM layer
  - CrawlDB, LinkDB, Segments를 합침
  - 귀찮은 Segments 관리를 피함
  - HBase
- Next step
  - Re-crawl algorithm
  - Crawler Trap 처리
  - 중복 제거 개선, mirror detection
  - Template detection
  - Spam & junk 제어

Nutch의 장단점

- Pros
  - Highly scalable, Highly modular
  - 높은 안정성
  - 빠른 batch 작업
  - 전략적인 scheduling
- Cons
  - 상대적으로 많은 기계가 필요
    - 안정성을 위해 최소 replication=3 유지
  - 실시간 업데이트 불가 (HDFS file 기반의 DB)
  - url – html match가 복잡함
  - Name node, job tracker 장애 시 수집 정지
    - 현재 hadoop의 한계

Conclusions

- 잘 만들어진 open source search engine
  - 분산 처리 환경에 최적화
  - 설정이 간편하고 modular 된 구조

- 장점 vs 단점

Reference

- nutch wiki
  - http://wiki.apache.org/nutch/
  - 대부분이 오래된 정보
- Nutch as a Web mining platform by Andrzej Białecki 2010
  - http://www.slideshare.net/abial/nutch-as-a-web-data-mining-platform
- MapReduce in Nutch by Doug Cutting 2005
  - http://frutch.free.fr/docs/mapred.pdf