VERIFICAÇÃO DE UNICIDADE DE URLS EM COLETORES DE PÁGINAS WEB

Wallace Favoreto
Orientador: Prof. Nivio Ziviani

Universidade Federal de Minas Gerais

Agenda

- Introdução
  - Definição do Problema
  - Objetivos
  - Trabalhos Relacionados
- Arquitetura de um Coletor
- Algoritmo Proposto (VEUNI)
- Experimentos
  - ClueWeb09
  - Baseline
  - Requisitos de Tempo e de Espaço
  - Taxa de Coleta
- Conclusões e Trabalhos Futuros
- Contribuições

Definição do Problema

- Algoritmo ingênuo para verificação de unicidade de URLs:

Entrada: U: Conjunto de URLs encontradas no ciclo de coleta.

Saída: Repositório R de URLs existentes no coletor atualizado com as URLs de U.

1: **para todo** URL em U **faça**
2: **se** URL está presente em R **então**
3: URL é descartada.
4: **se não**
5: URL é adicionada a R.

Definição do Problema

- Considere que:
  - Aproximadamente 1 milhão de URLs deve ser verificada em um ciclo de coleta;
  - O disco tenha tempo de *seek* igual a 10 milissegundos.
- O algoritmo ingênuo leva pelo menos 3h para verificar a unicidade de 1 milhão de URLs.
- Um algoritmo mais eficaz e eficiente é necessário.

Definição do Problema

- Eficácia:
  - Dizer com certeza se uma URL já foi coletada ou não.
- Eficiência:
  - Executar em tempo não proibitivo ao funcionamento do restante do coletor.
- Quais os problemas de um verificador de unicidade de URLs ineficaz e ineficiente?

Definição do Problema

- Desperdício de espaço de armazenamento.
- Redução do número de páginas distintas coletadas.
- Desperdício de banda de internet no *fetcher*.
- Redução da expansão da coleta.
- Desrespeito ao *politeness*.

Objetivos

- Propor um novo algoritmo chamado VEUNI (VErificador de UNIcidade de URLs).
- Discutir a arquitetura do coletor que o algoritmo VEUNI está inserido.
- Comparar o VEUNI com o melhor algoritmo da literatura.

Trabalhos Relacionados

- Árvore-B [Pinkerton, 1994]
- Cache de memória [Heydon & Najork, 1999]
- *batch disk check* [Heydon & Najork 2002]

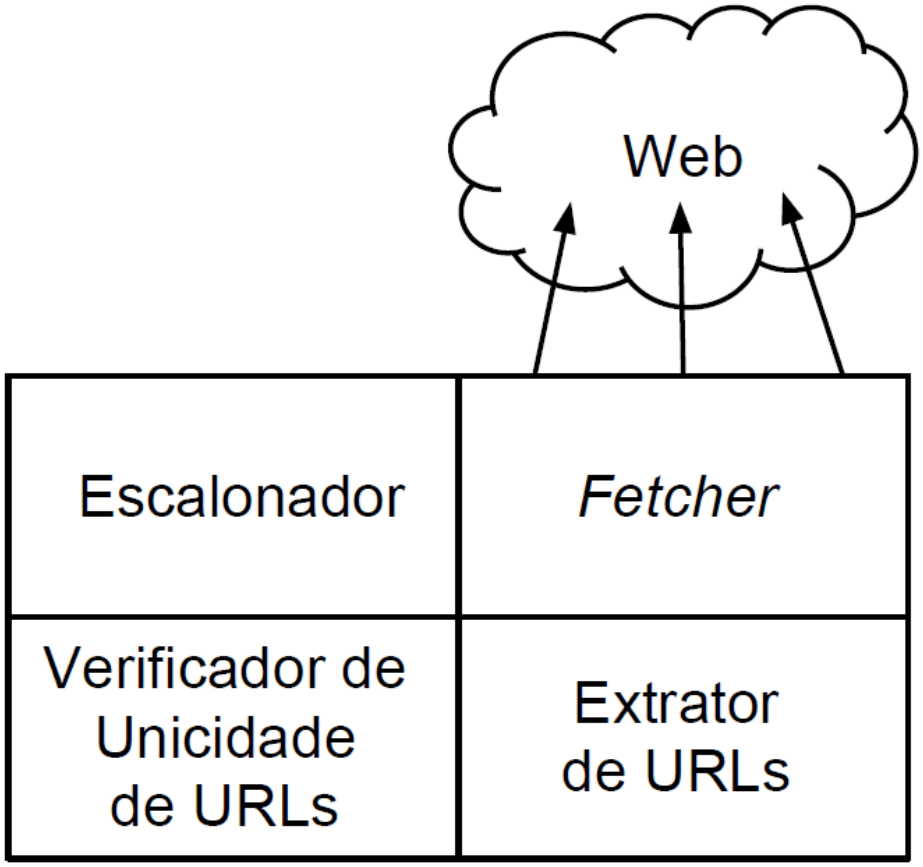
Máquina de Busca - Componentes

- Coletor
  - Encontra, coleta e armazena páginas da Web.
- Indexador
  - Mantém um índice de páginas coletadas.
- Processador de Consultas
  - Realiza buscas eficientes sobre o índice.

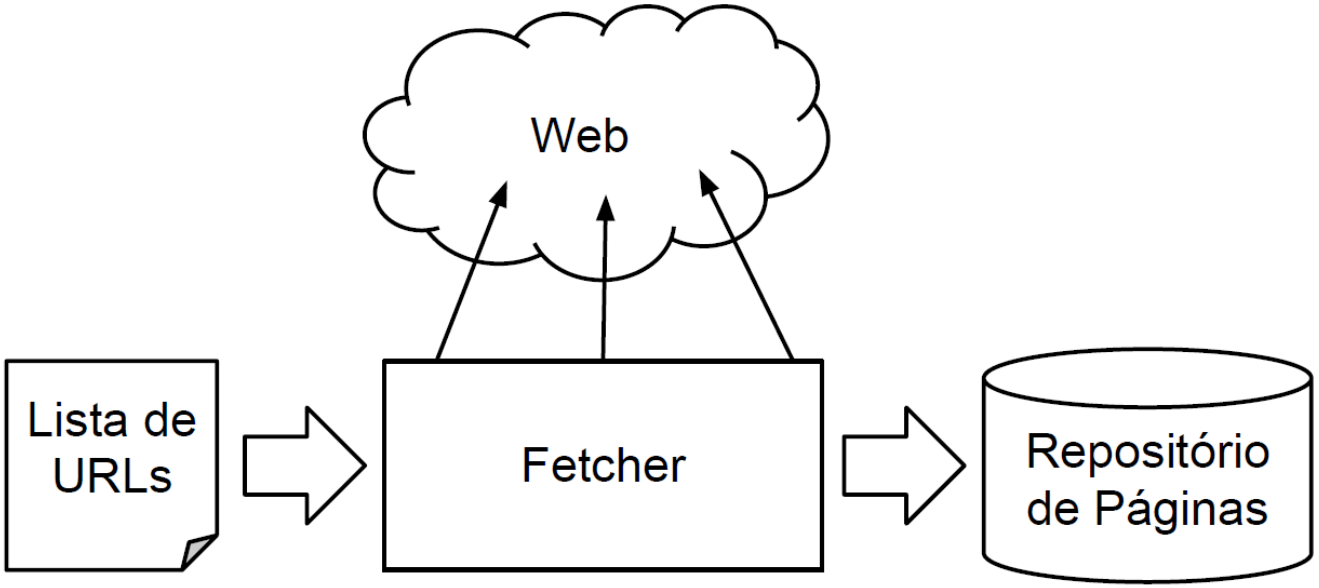
Coletor – Componentes

- *Fetcher*
- Extrator de URLs
- Verificador de Unicidade de URLs
- Escalonador

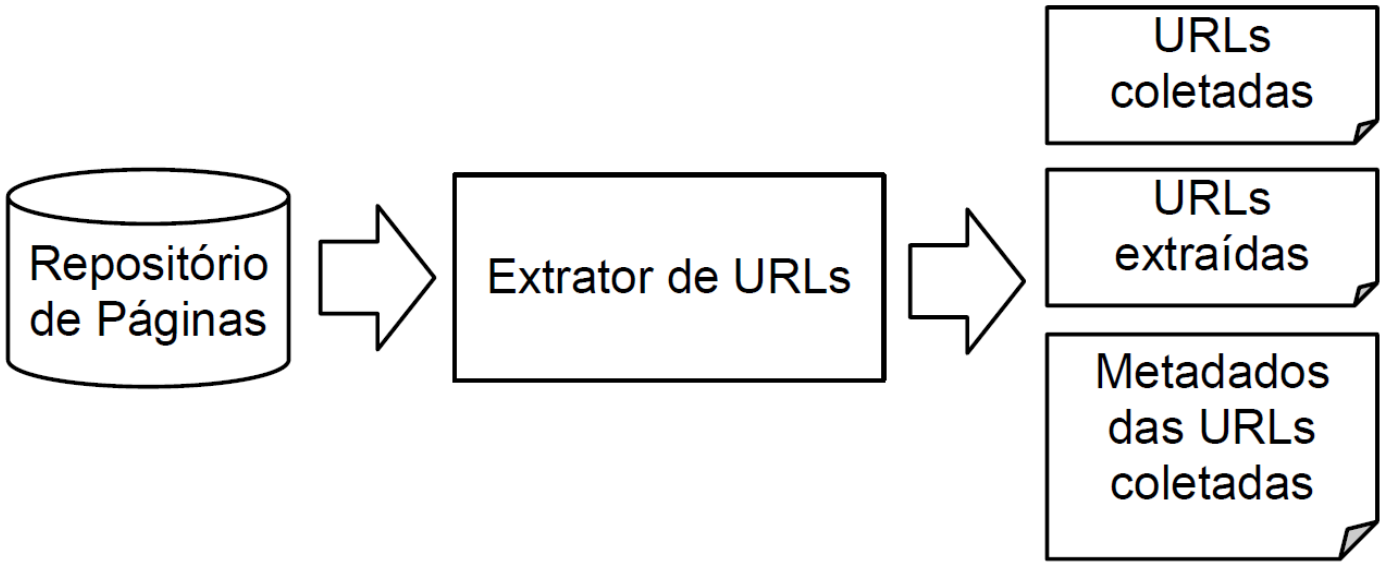
Arquitetura de um Coletor



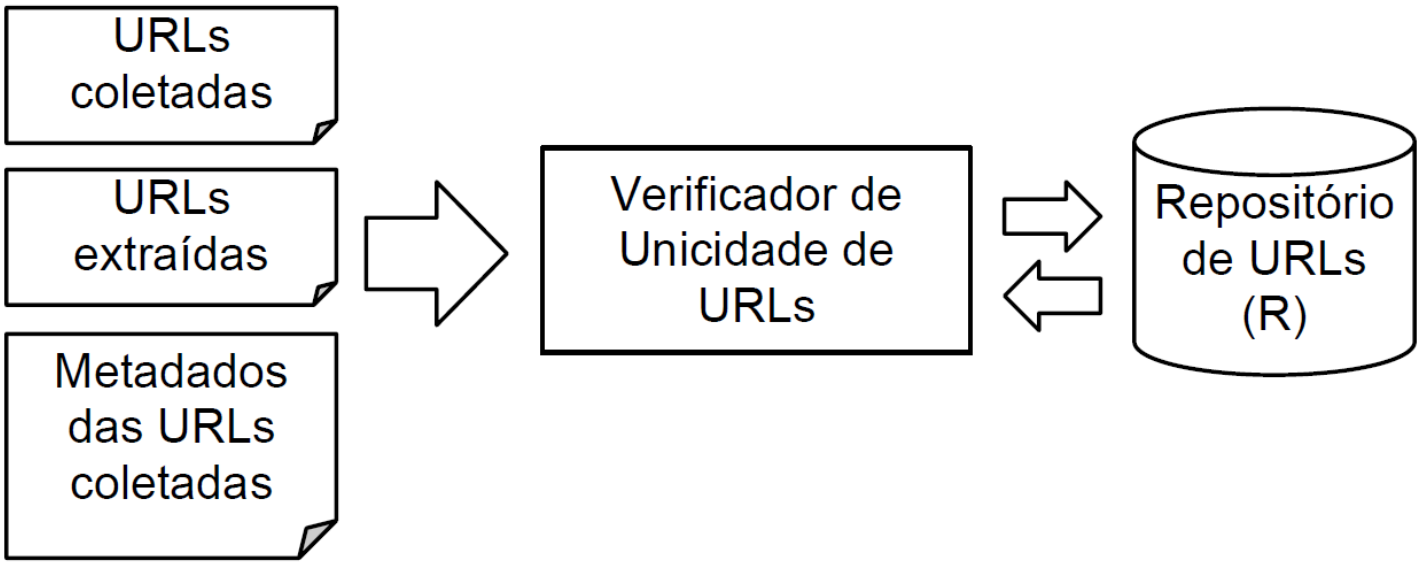
Fetcher



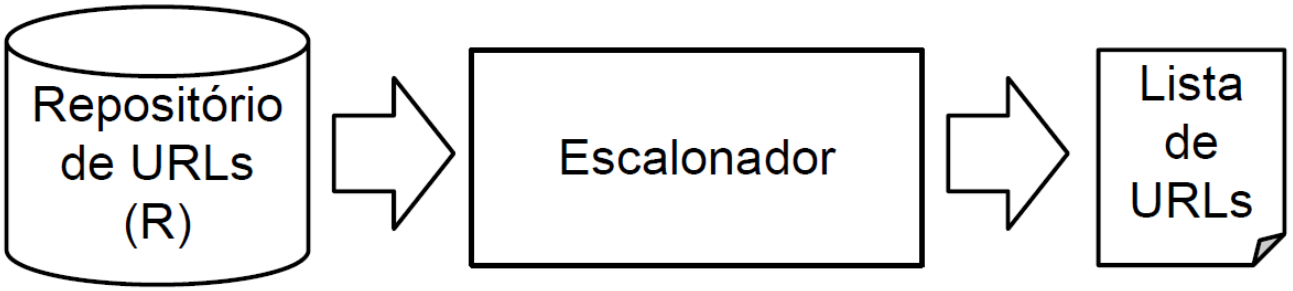
Extrator de URLs



Verificador de Unicidade de URLs

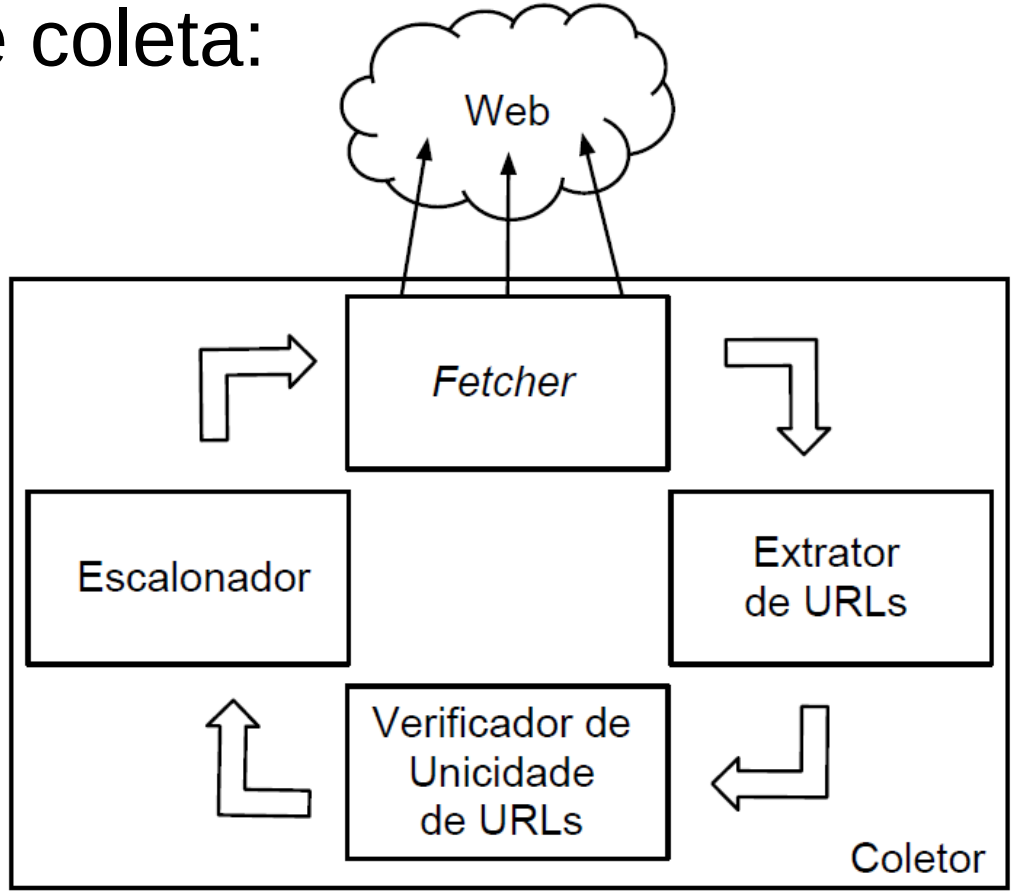


Escalonador

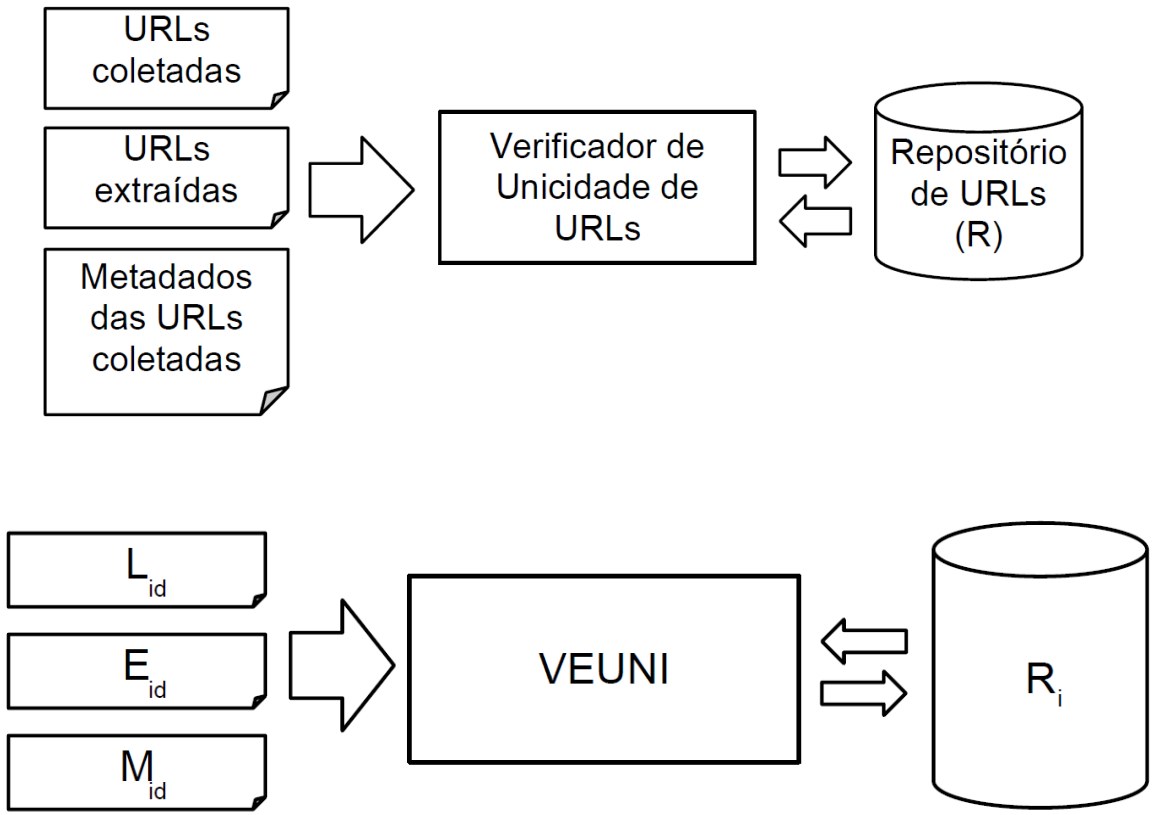


Algoritmo Proposto - VEUNI

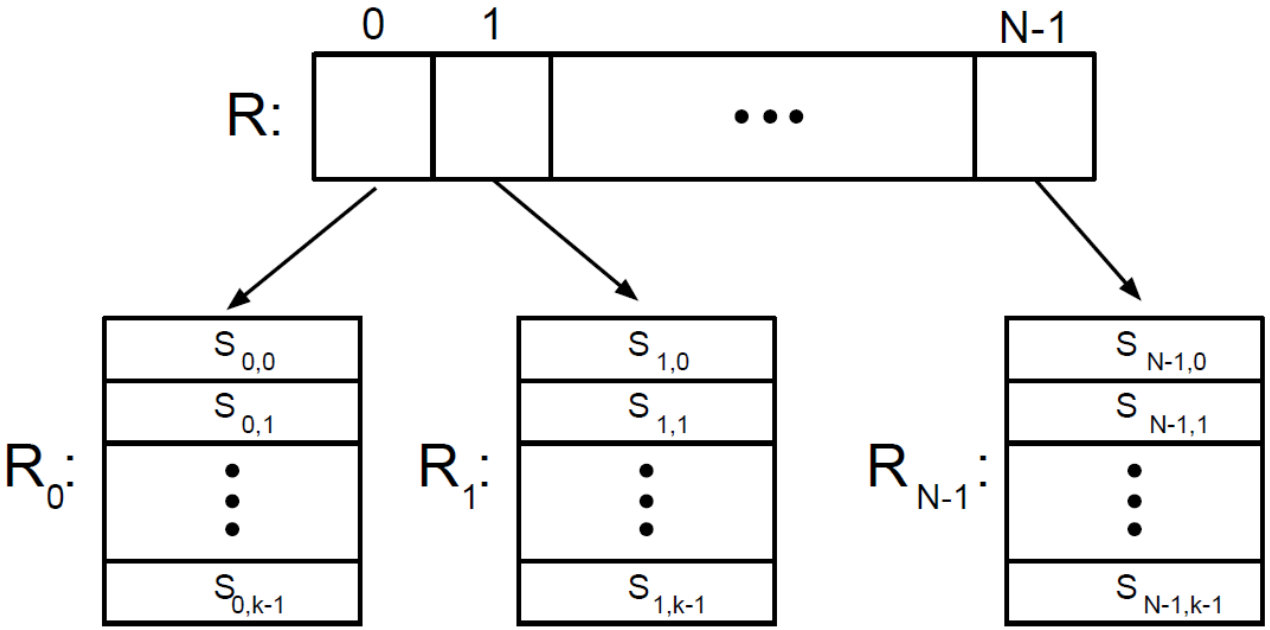
- O algoritmo VEUNI é parte do coletor.
- Ciclo de coleta:



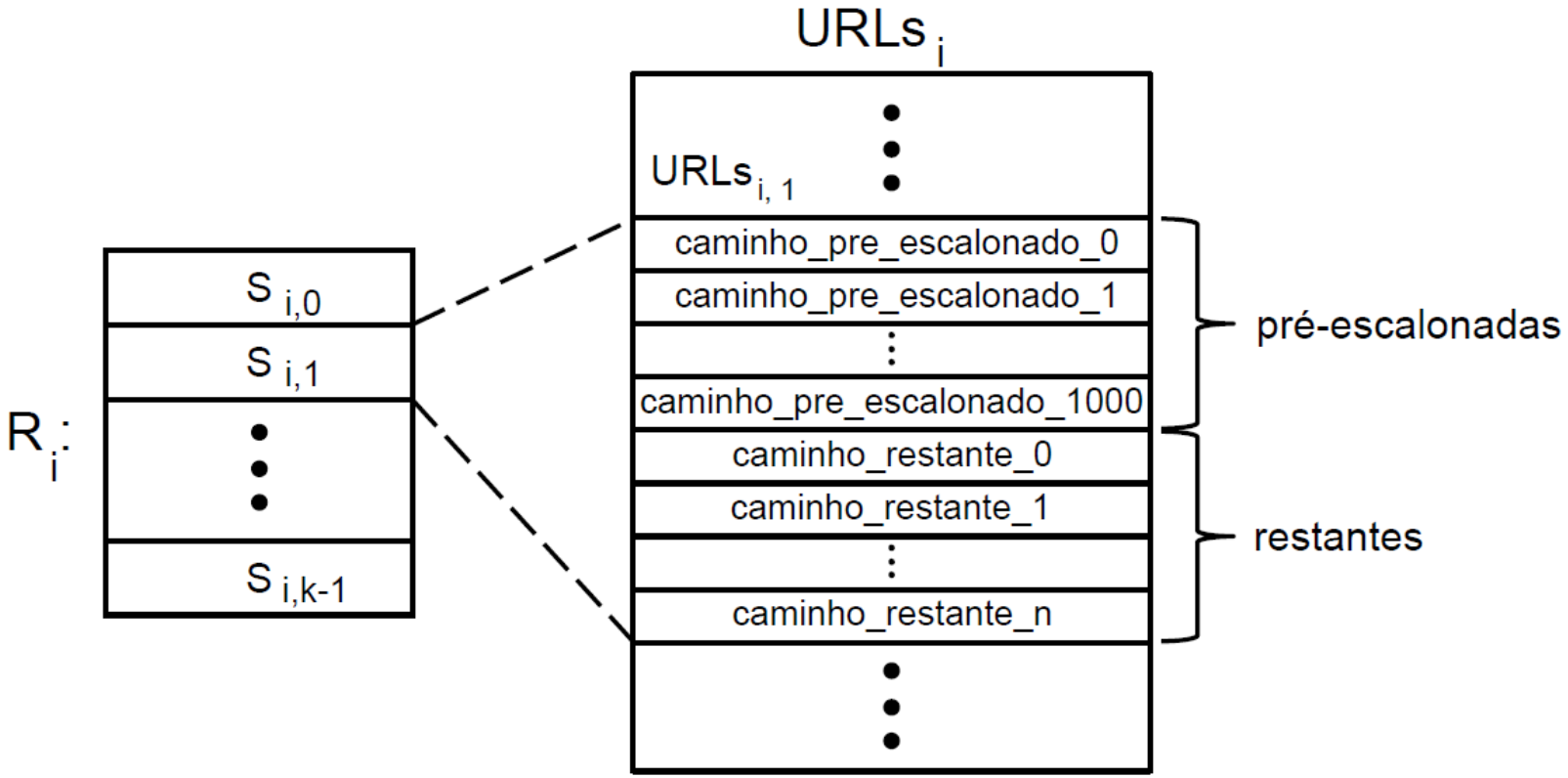
VEUNI – Entrada e Saída



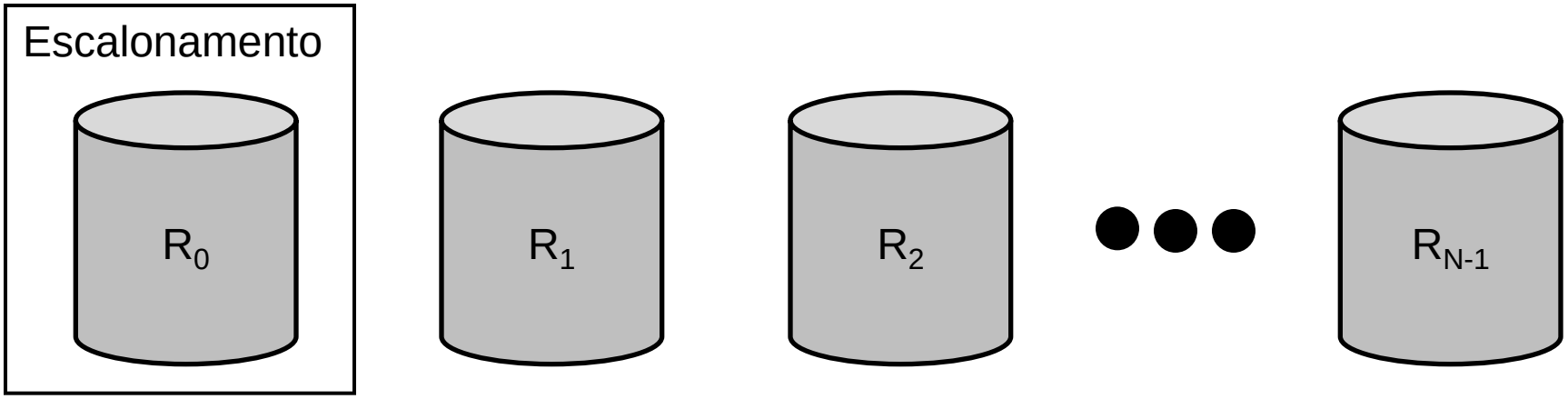
Repositório de URLs (R)



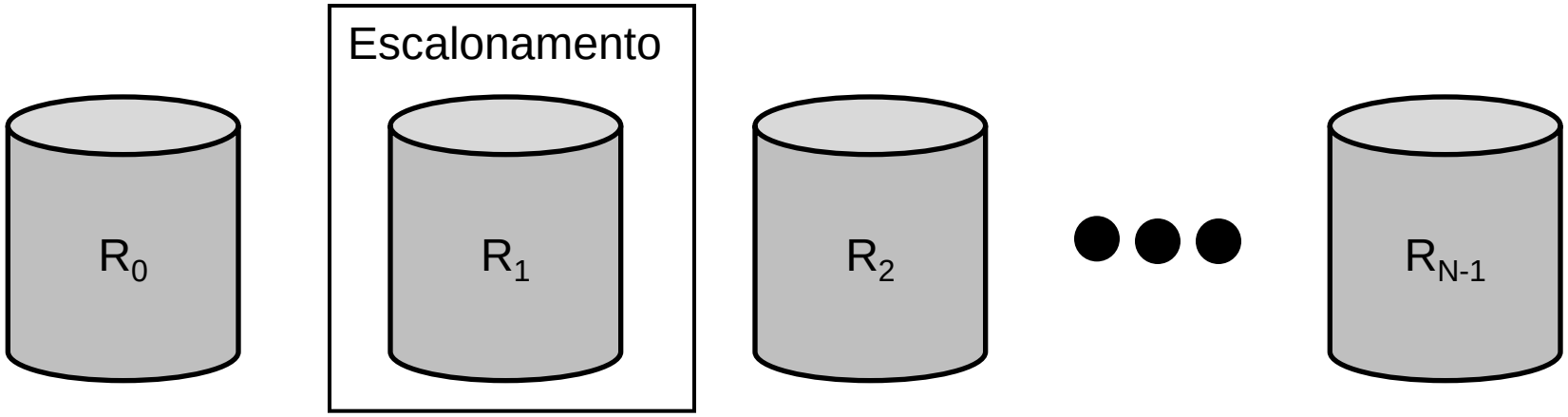
Algoritmo Proposto - VEUNI



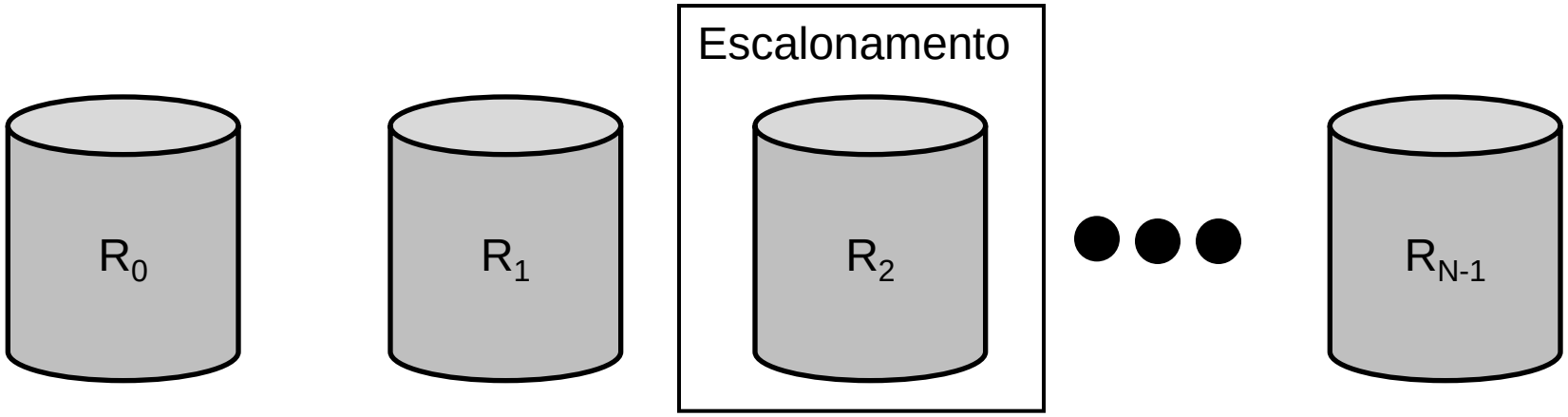
VEUNI - Exemplo de Funcionamento



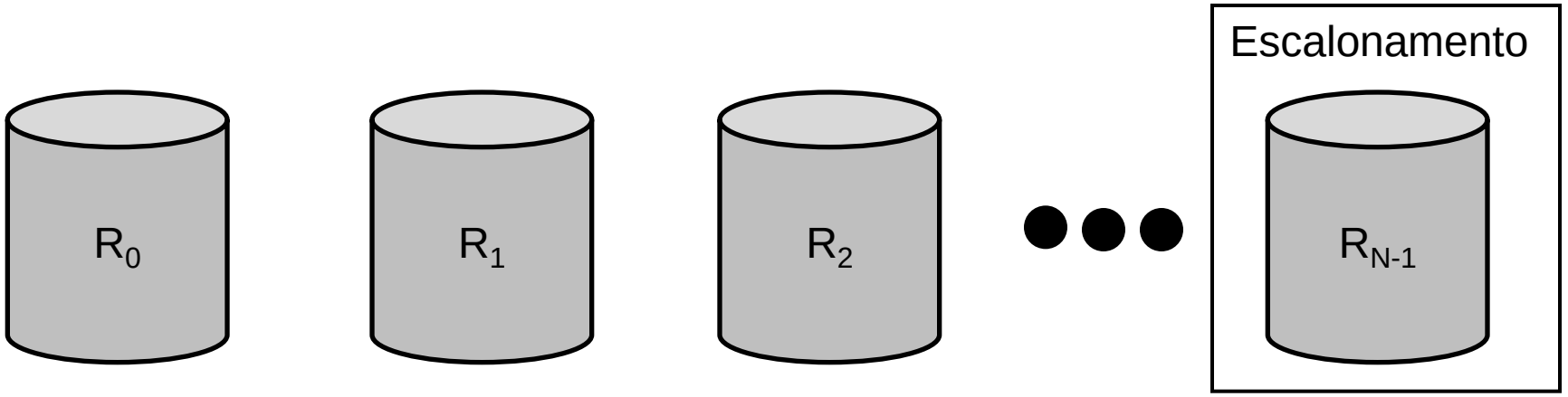
VEUNI - Exemplo de Funcionamento



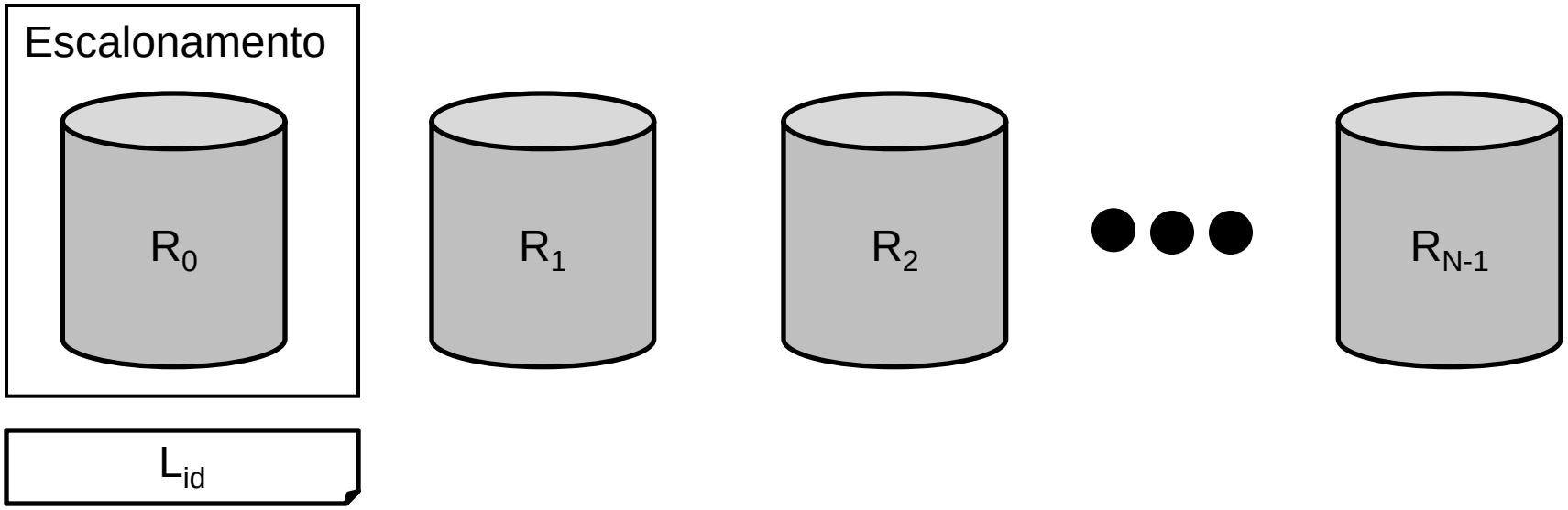
VEUNI - Exemplo de Funcionamento



VEUNI - Exemplo de Funcionamento

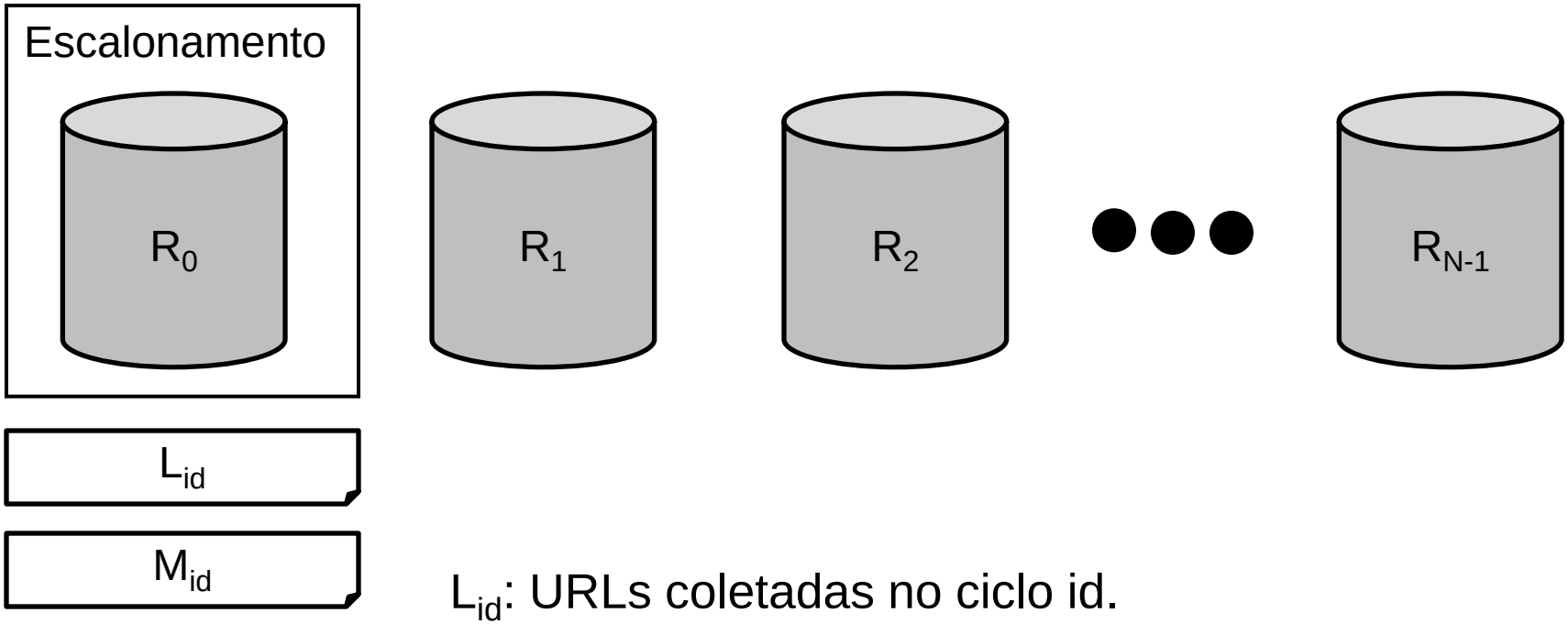


VEUNI - Entrada L_{id}



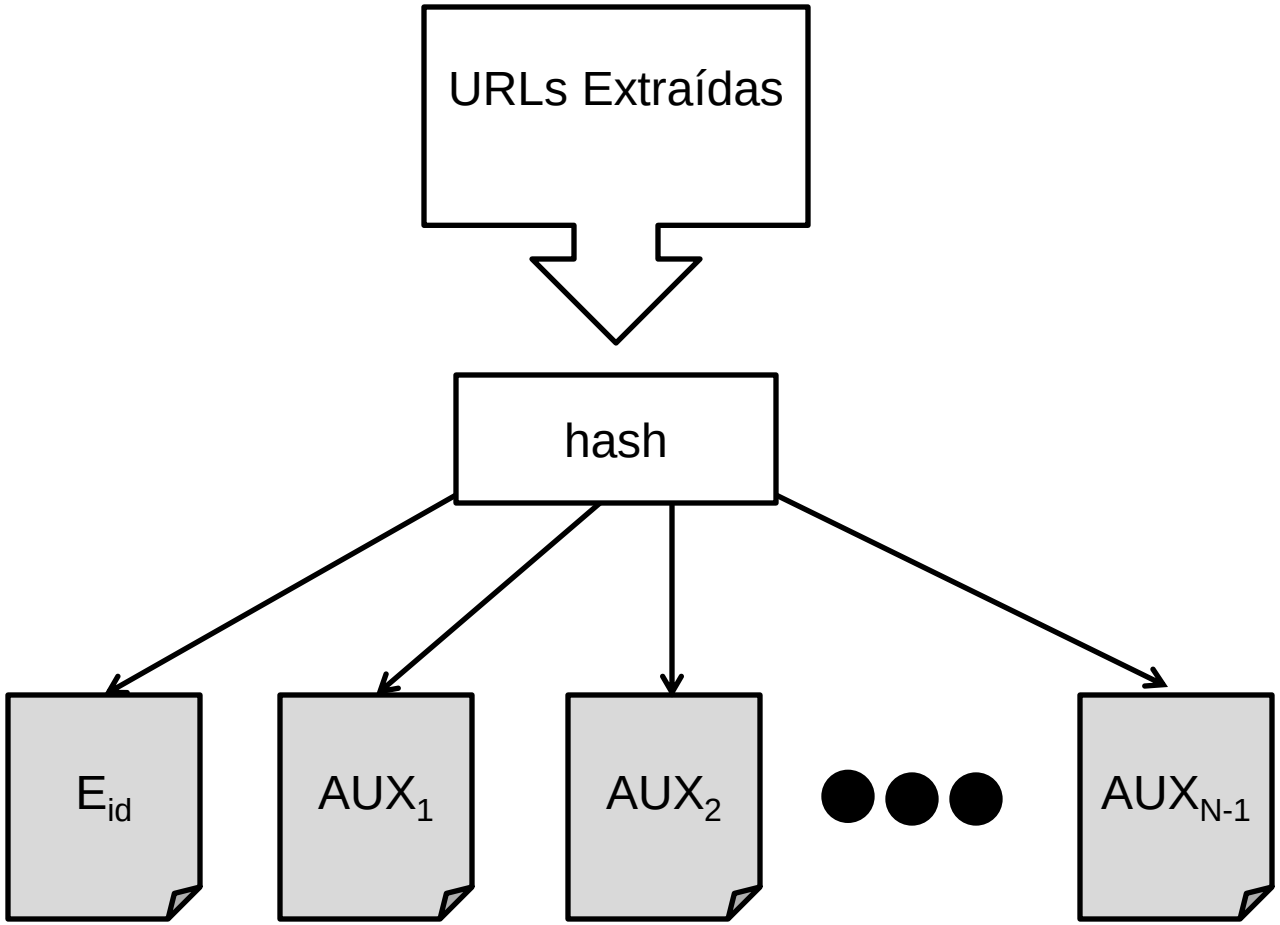
L_{id}: URLs coletadas no ciclo id.

VEUNI - Entrada M_{id}

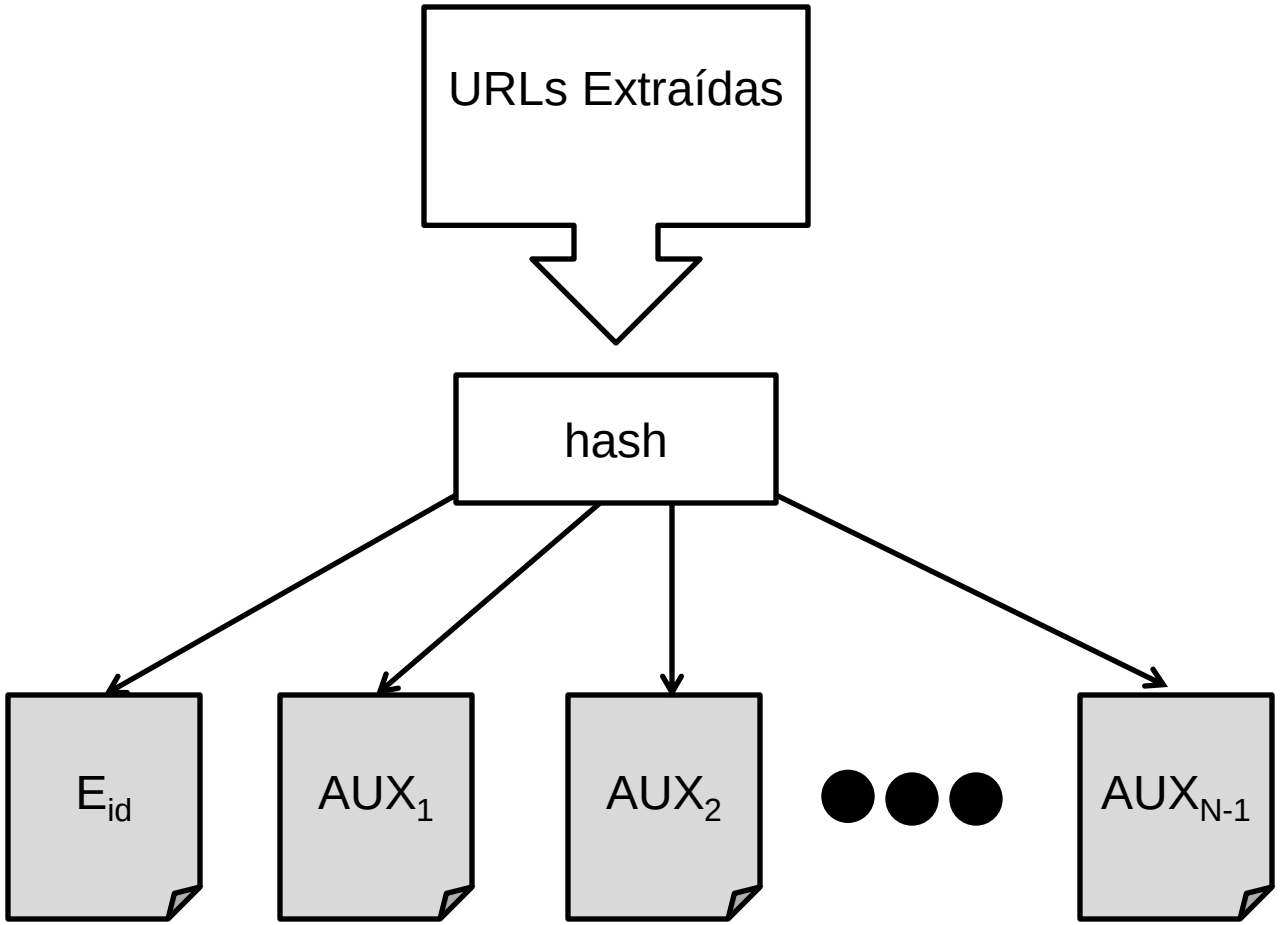


L_{id}: URLs coletadas no ciclo id.
M_{id}: Metadados das páginas coletadas no ciclo id.

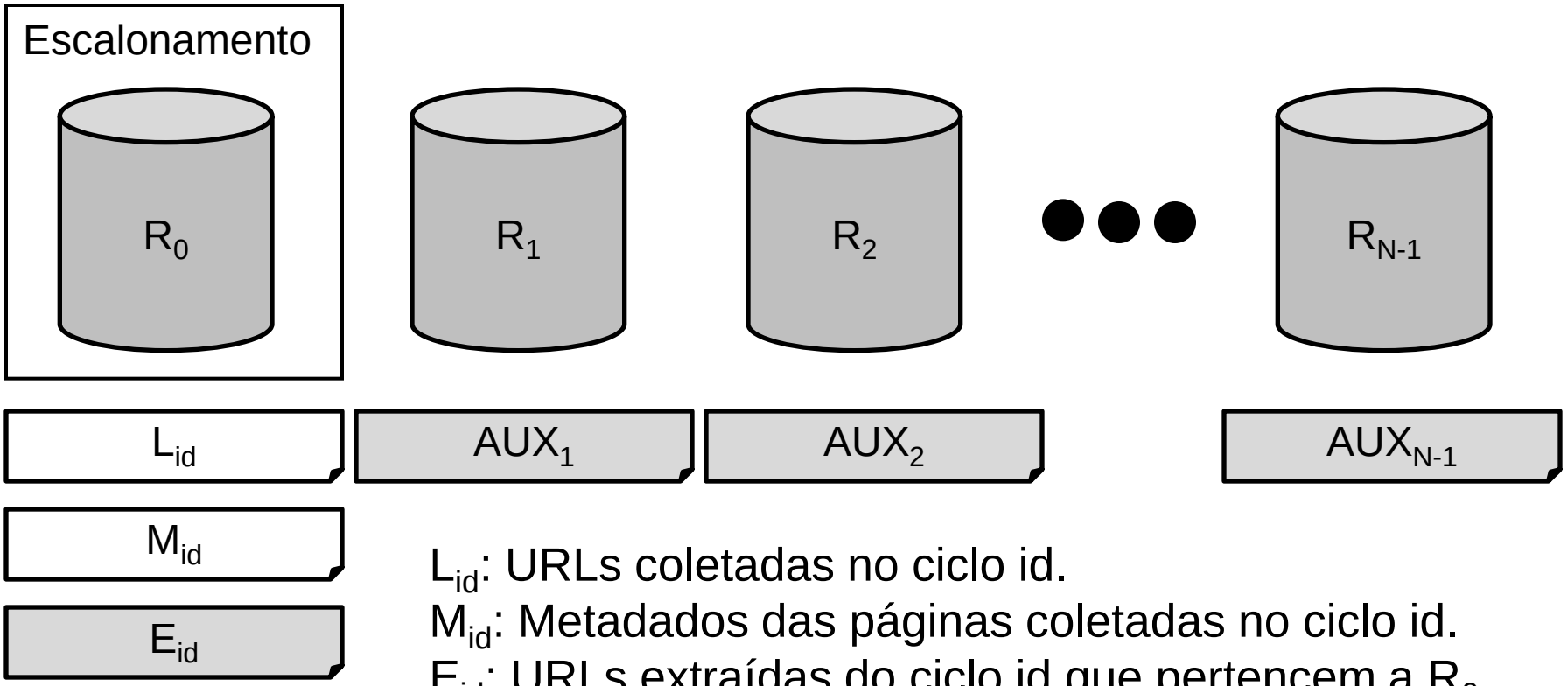
VEUNI - Entrada E_{id}



VEUNI - Entrada E_{id}

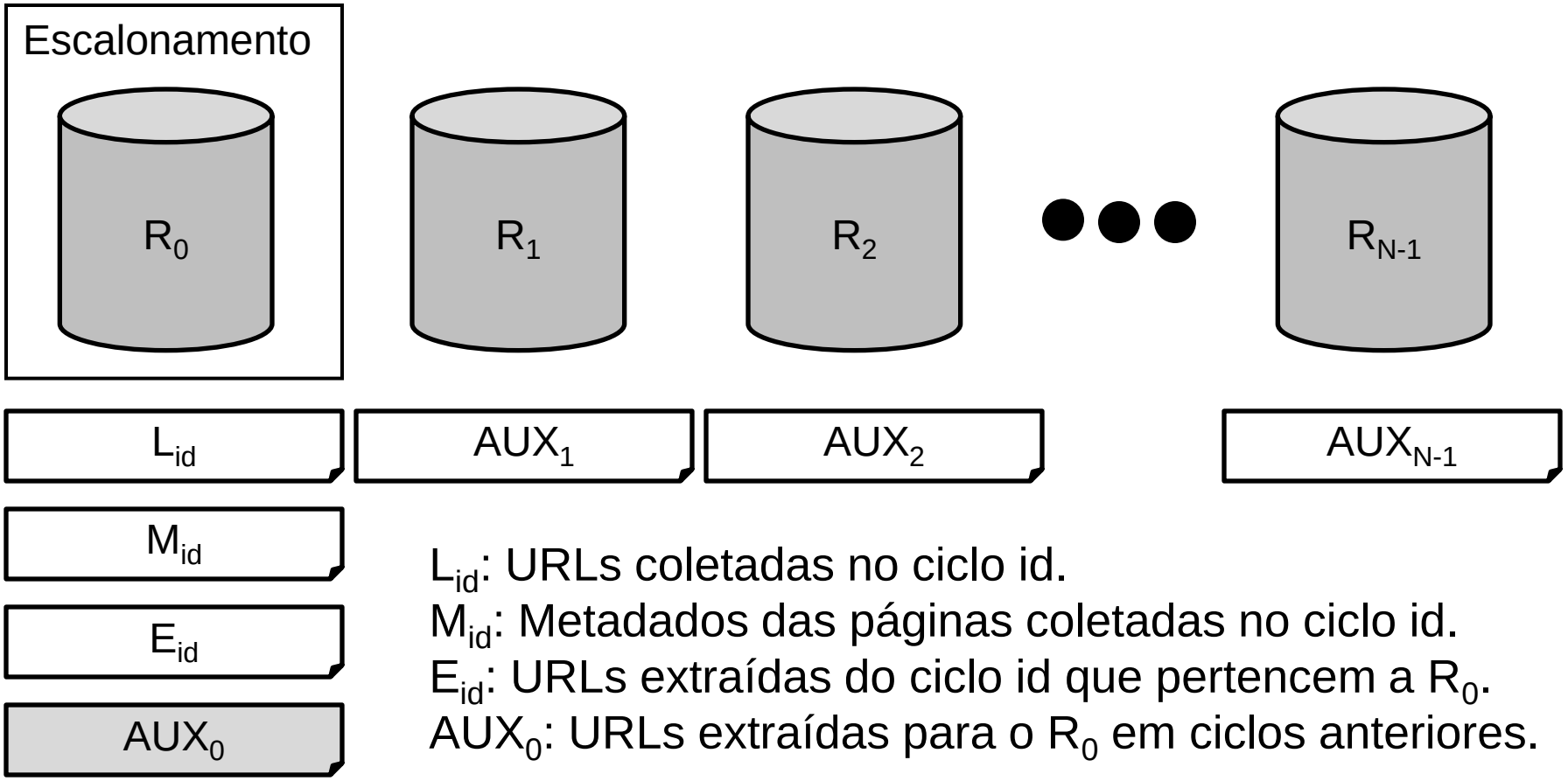


VEUNI - E_{id} e AUX_i



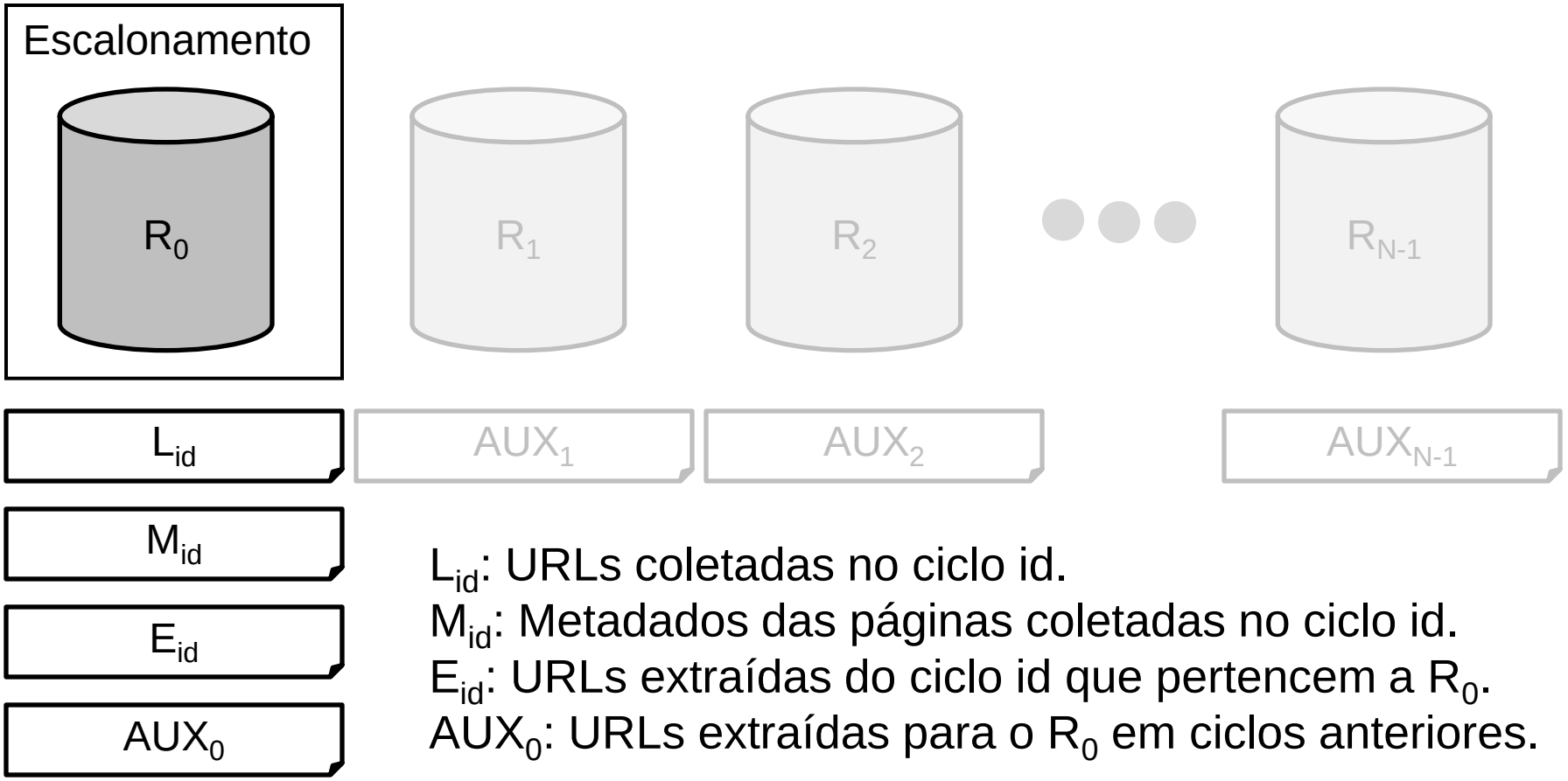
L_{id}: URLs coletadas no ciclo id.
M_{id}: Metadados das páginas coletadas no ciclo id.
E_{id}: URLs extraídas do ciclo id que pertencem a R_0.

VEUNI - AUX_0



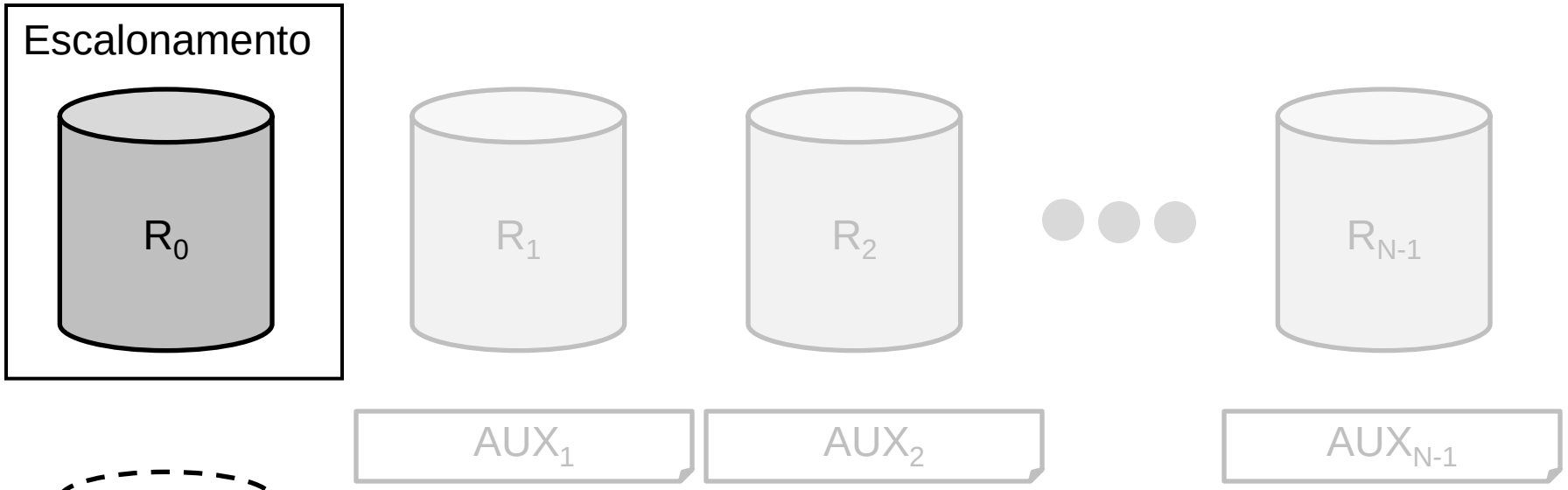
L_{id}: URLs coletadas no ciclo id.
M_{id}: Metadados das páginas coletadas no ciclo id.
E_{id}: URLs extraídas do ciclo id que pertencem a R_0.
AUX_0: URLs extraídas para o R_0 em ciclos anteriores.

VEUNI - Entrada



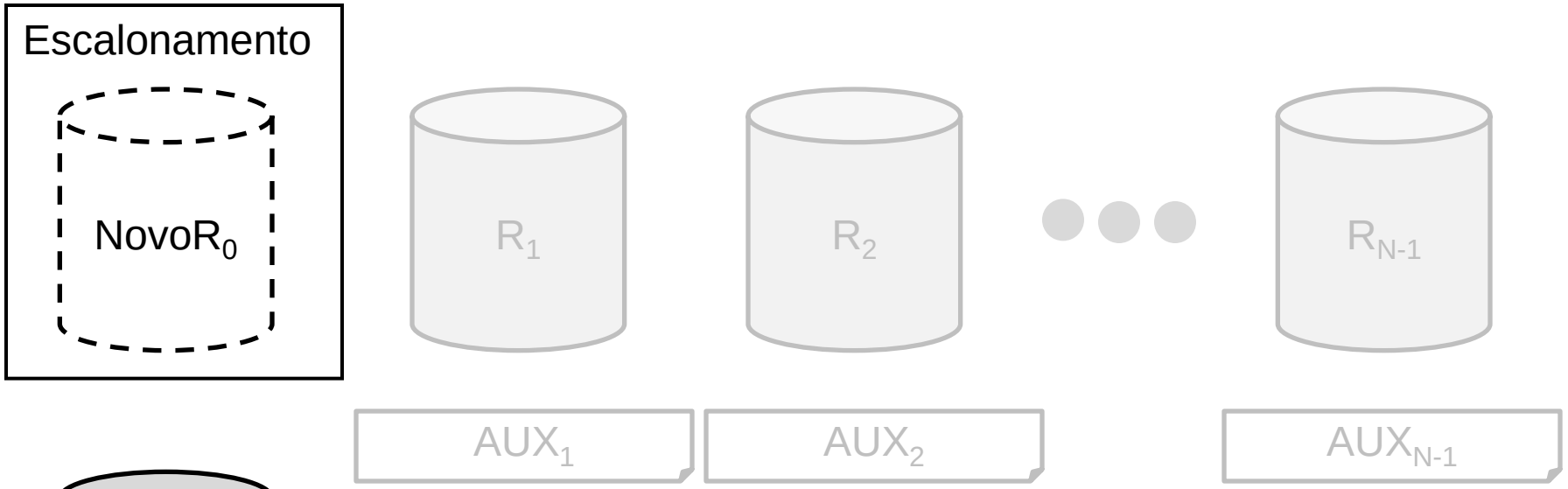
L_{id}: URLs coletadas no ciclo id.
M_{id}: Metadados das páginas coletadas no ciclo id.
E_{id}: URLs extraídas do ciclo id que pertencem a R_0.
AUX_0: URLs extraídas para o R_0 em ciclos anteriores.

VEUNI - Saída



L_{id}: URLs coletadas no ciclo id.
M_{id}: Metadados das páginas coletadas no ciclo id.
E_{id}: URLs extraídas do ciclo id que pertencem a R_0.
AUX_0: URLs extraídas para o R_0 em ciclos anteriores.

VEUNI - Saída



L_{id}: URLs coletadas no ciclo id.
M_{id}: Metadados das páginas coletadas no ciclo id.
E_{id}: URLs extraídas do ciclo id que pertencem a R_0.
AUX_0: URLs extraídas para o R_0 em ciclos anteriores.

VEUNI – Primeiro Refinamento

Entrada: id, L_{id}, E_{id} e M_{id}
Saída: Repositório R_i de URLs existentes no coletor atualizado com as URLs do ciclo de coleta id.

1: i recebe id mod N.
2: E_{id} recebe a união de E_{id} com AUX_i.
3: Particione L_{id} e E_{id} em subgrupos B_j, $0 \leq j < \text{Nserv}_{id}$, tal que B_j possua URLs de um mesmo servidor j.
4: Inicializa um novo repositório Rnovo_i em disco.
5: **para todo** S_{ij} em R_i **faça**
6: Intercala S_{ij} com B_j.
7: Grava em Rnovo_i o resultado da intercalação.
8: **para todo** B_j contendo URLs de novos servidores **faça**
9: Insere B_j em Rnovo_i.
10: Rnovo_i passa a ser o novo R_i.

VEUNI - Intercalação

Entrada: S_{ij}: URLs do Servidor$_{ij}$ existentes em R_i;
B_j: URLs coletadas, metadados e URLs extraídas do Servidor$_{ij}$.
Saída: Snovo$_{ij}$: resultado da intercalação de B_{ij} com S_{ij}.

1: Carrega URLs$_{ij}$ para a memória utilizando as informações de S_{ij}.
2: **se** B_j != vazio **então**
3: Intercala o conjunto URLs$_{ij}$ com o conjunto de URLs de B_j.
4: Armazena em Snovo$_{ij}$ o resultado da intercalação realizada.
5: **se não**
6: Armazena S_{ij} em Snovo$_{ij}$.

VEUNI - Intercalação

Entrada: URLs_{ij}: URLs do Servidor_{ij} que estão armazenadas em R_i;
B_j: URLs do Servidor_{ij} encontradas no ciclo de coleta.
Saída: URLsnovo_{ij}: resultado da intercalação de URLs_{ij} com B_j.

1: Separa as URLs do conjunto URLs_{ij} em $\text{URLs\_PreEscalonadas}_{ij}$ e $\text{URLs\_Restantes}_{ij}$.
2: Separa as URLs do conjunto B_j em $B\_\text{Coletadas}_j$ e $B\_\text{Extraidas}_j$.
3: Ordena lexicograficamente os conjuntos $B\_\text{Coletadas}_j$ e $B\_\text{Extraidas}_j$.
4: Intercala as URLs dos conjuntos $\text{URLs\_PreEscalonadas}_{ij}$ e $B\_\text{Coletadas}_j$ e armazena em um *buffer* chamado *pre_escalonadas*.
5: Intercala as URLs dos conjuntos $\text{URLs\_Restantes}_{ij}$ e $B\_\text{Extraidas}_j$ e armazena em um *buffer* chamado *restantes*.
6: Redistribui as URLs dos *buffers pre_escalonadas* e *restantes* se for necessário.
7: Armazena em URLsnovo_{ij} as URLs dos conjuntos *pre_escalonadas* e *restantes*.

VEUNI – Inserção de Novos servidores

Entrada: B_j: URLs coletadas, metadados
e URLs extraídas agrupados por servidor.
Saída: Rnovo_i: Repositório atualizado com as URLs de B_{ij}.
1: Divide as URLs de B_j nas categorias *pré-escalonadas* e *restantes* e armazena o resultado em $B\_\text{PreEscalonadas}_{ij}$ e $B\_\text{Restantes}_{ij}$.
2: Insere $B\_\text{PreEscalonadas}_{ij}$ e $B\_\text{Restantes}_{ij}$ em Rnovo_i.

Experimentos

- Requisitos de tempo e espaço
  - Tempo de atualização das estruturas de dados.
  - Espaço requerido para armazenamento das estruturas de dados.
- Taxa de coleta
  - Medir a taxa no decorrer da coleta.
  - Número de páginas/segundo.

ClueWeb09

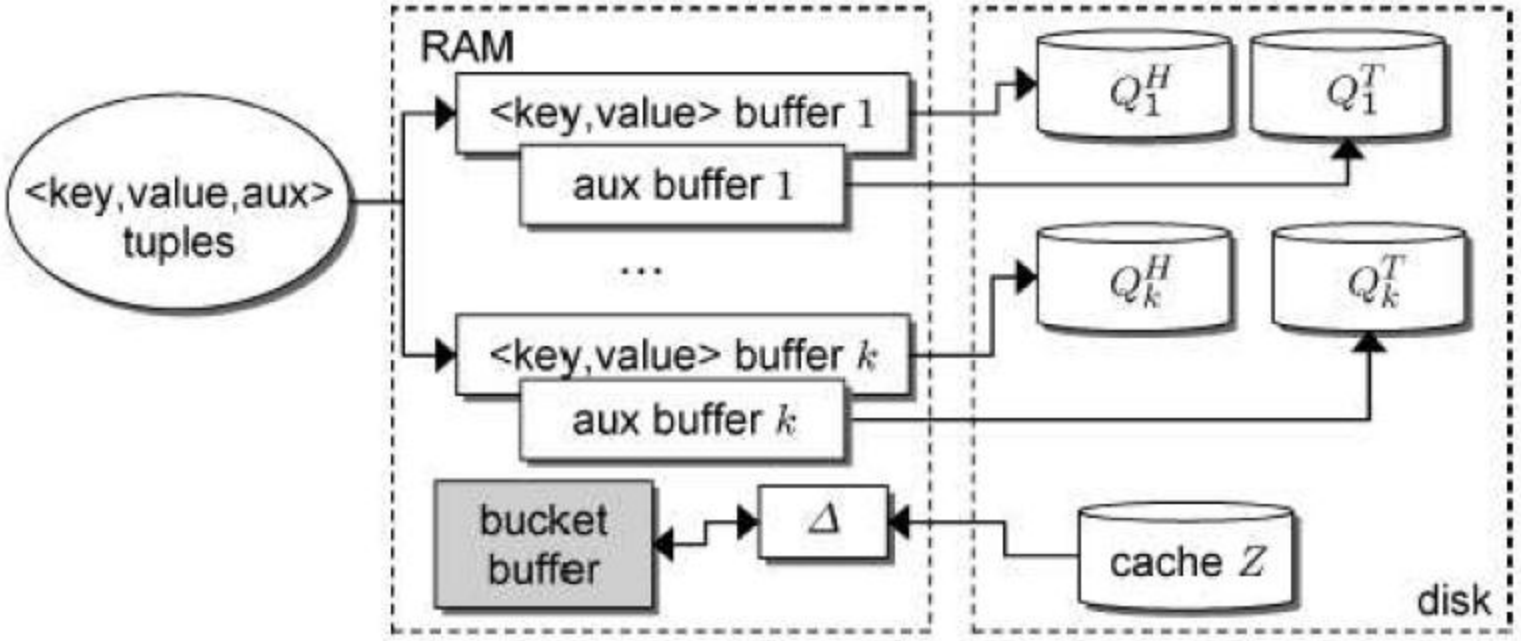
- Simulação de uma coleta utilizando as páginas da ClueWeb09.
- Evita a necessidade de utilização do *fetcher*, do extrator de URLs e do escalonador.
- Experimento mais controlado.
- Reprodução facilitada.
- 350 milhões de URLs foram extraídas da ClueWeb09.

Coleta Simulada

- Conjunto de URLs da ClueWeb09: 350 milhões
- # de URLs em cada ciclo de coleta: 1 milhão
  - # de URLs coletadas em cada ciclo: 100 mil
  - # de URLs extraídas em cada ciclo: 900 mil
- 350 ciclos de coleta realizados.

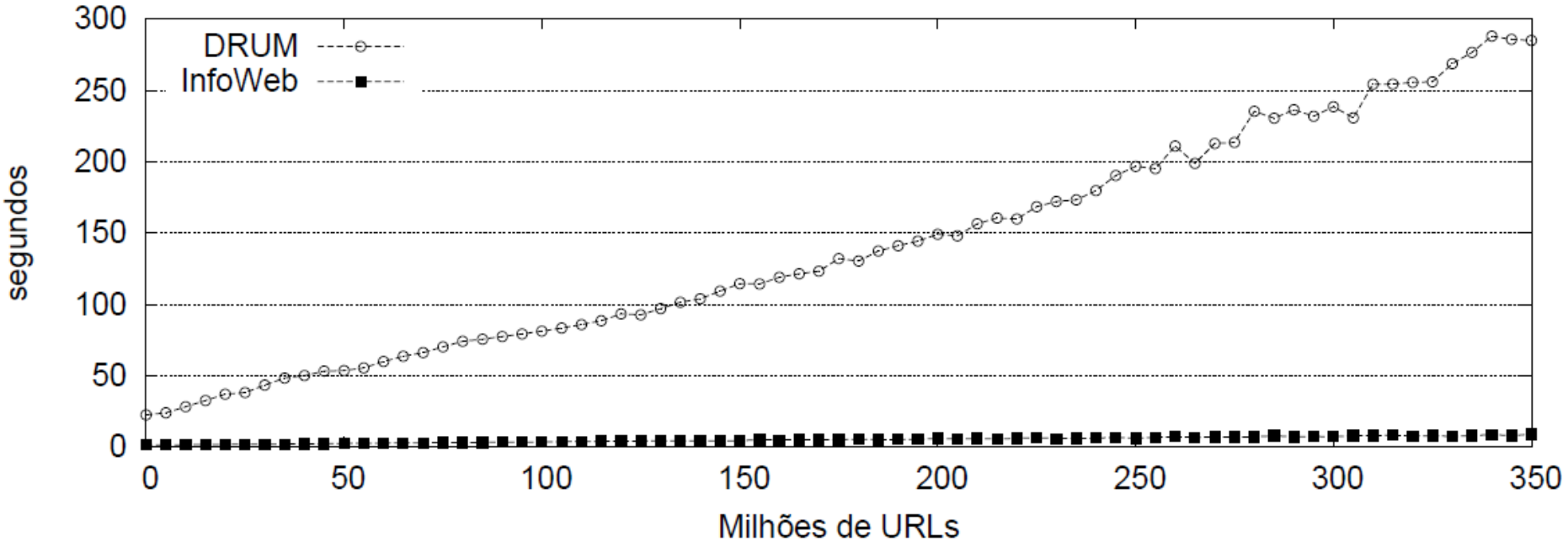
Baseline

- DRUM (Disk Repository with Update Management)



Resultados

- Requisito de Tempo



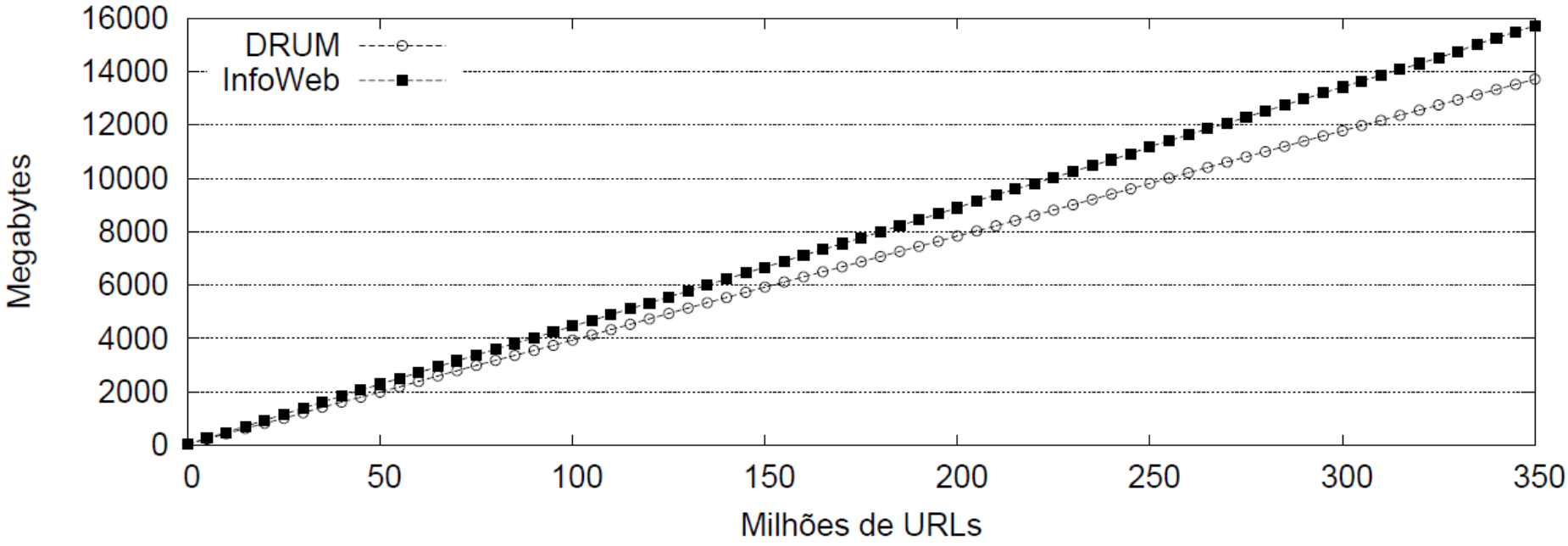
Resultados

- Requisito de Tempo

# de URLs (milhões)	Tempo (s)	
	Algoritmos	
	DRUM	VEUNI
1	22.17	1.35
50	53.53	2.60
100	81.24	3.53
150	114.55	4.47
200	149.11	6.11
250	196.60	6.17
300	238.53	7.26
350	284.92	9.07

Resultados

- Requisito de Espaço



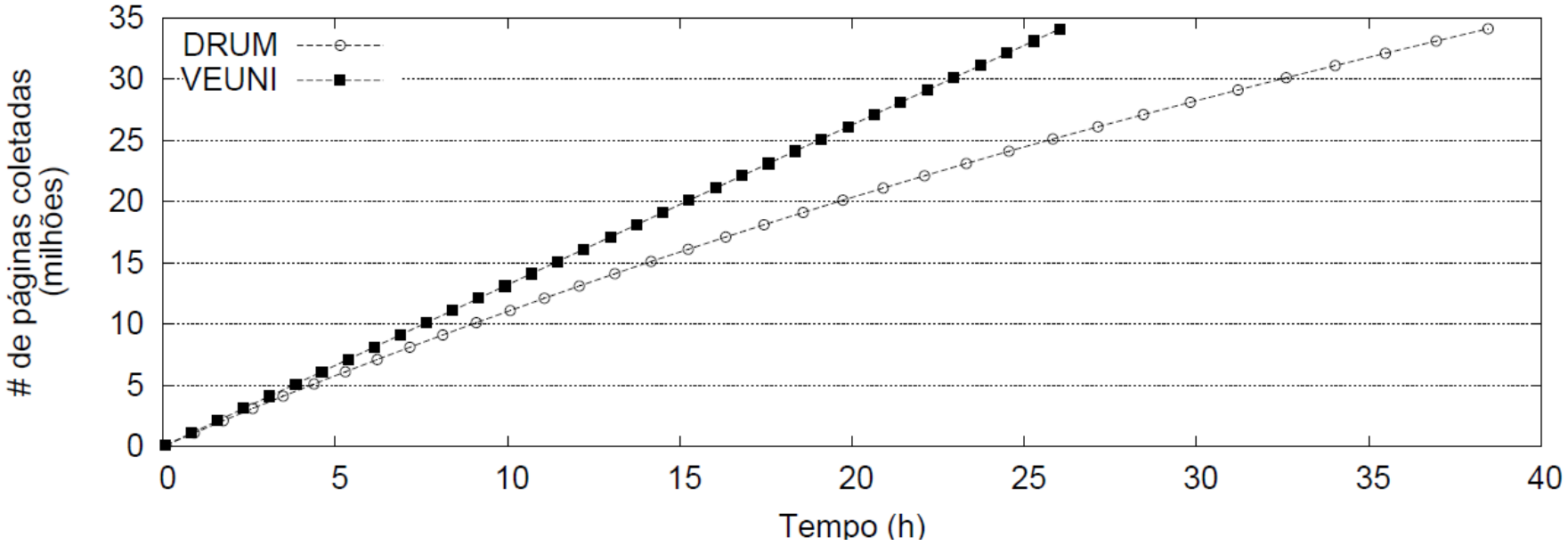
Resultados

- Requisito de Espaço

# de URLs (milhões)	Espaço (MB)	
	Algoritmos	
	DRUM	VEUNI
1	51.34	43.95
50	1991.42	2294.19
100	3940.20	4461.68
150	5925.89	6672.58
200	7836.91	8894.62
250	9809.36	11190.72
300	11782.43	13419.82
350	13713.81	15712.71

Resultados

- Tempo de Coleta



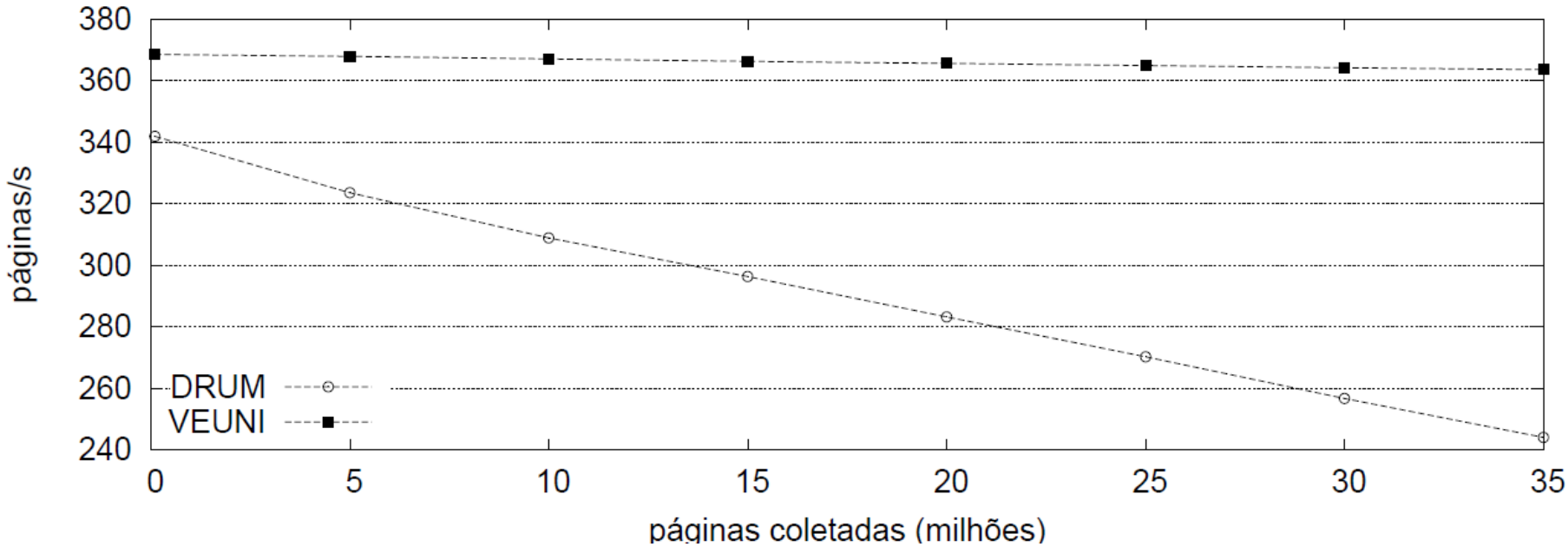
Resultados

- Tempo de Coleta

# de páginas coletadas (milhões)	Tempo(h) Algoritmos DRUM	VEUNI
0.1	0.08	0.07
5	4.29	3.77
10	8.99	7.56
15	14.06	11.37
20	19.61	15.19
25	25.70	19.03
30	32.46	22.88
35	39.84	26.74

Resultados

- Taxa de Coleta



Resultados

- Taxa de Coleta

# de páginas coletadas (milhões)	páginas/s Algoritmos DRUM	VEUNI
0.1	341.85	368.49
5	323.55	367.87
10	308.84	367.03
15	296.32	366.26
20	283.19	365.57
25	270.19	364.87
30	256.68	364.18
35	244.00	363.51

Conclusões

- O algoritmo VEUNI é mais eficiente que o algoritmo *baseline*.
- O algoritmo VEUNI requer mais espaço de armazenamento que o algoritmo *baseline*.
- A taxa de coleta do coletor que utiliza o algoritmo VEUNI praticamente se mantém no decorrer da coleta.

Trabalhos Futuros

- Comparar o algoritmo VEUNI com uma implementação do algoritmo DRUM que não utilize Berkeley DB.
- Estudar maneiras de simplificar a representação das URLs, sem perder a eficiência relatada.

Contribuições

- Apresentação da arquitetura de um coletor de páginas da Web. (Capítulo 2)
- Algoritmo VEUNI para verificar unicidade de URLs (Capítulo 3)
- Discussão de propostas que levaram ao algoritmo VEUNI (Seção 3.3)
- Ambiente de experimentação controlado que utiliza informação da ClueWeb09 (Seção 4.2)

Perguntas