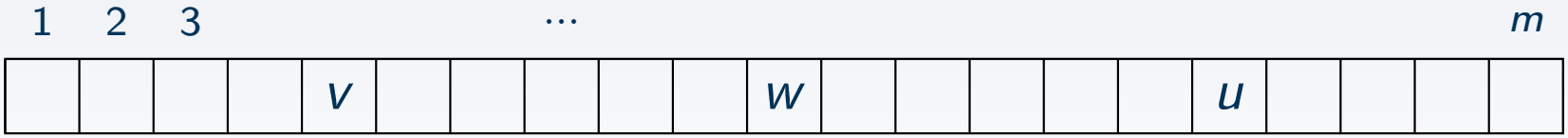
Cuckoo Hashing with Pages

Martin Dietzfelbinger Technische Universität Ilmenau
Michael Mitzenmacher Harvard University
Michael Rink Technische Universität Ilmenau

07.09.11

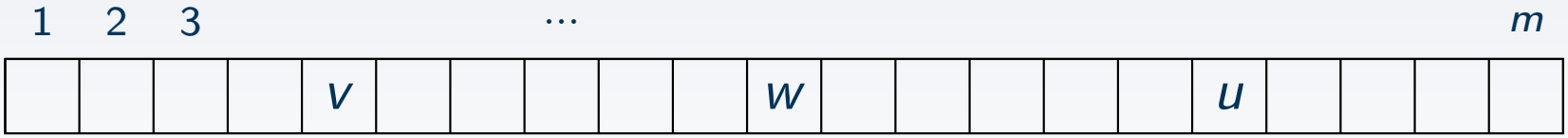
k-ary Cuckoo Hashing



[Fotakis et al., 2005]:

- n keys, table of size m

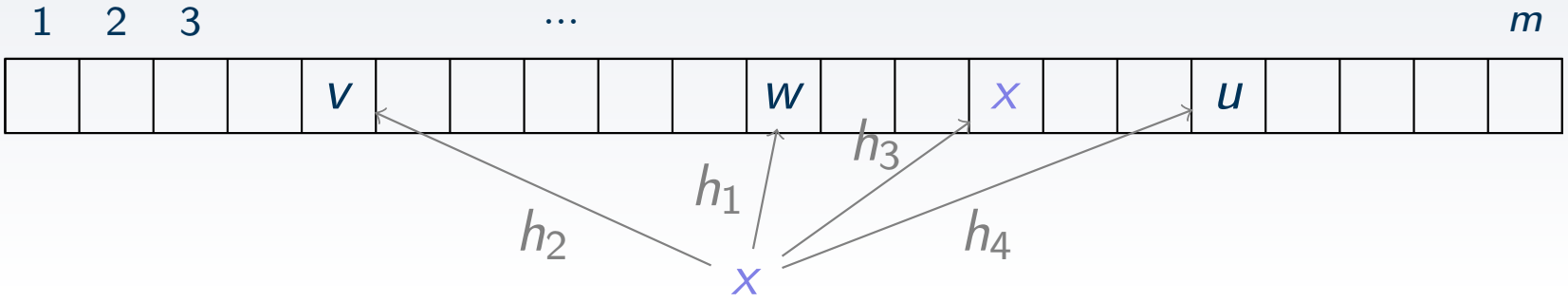
k-ary Cuckoo Hashing



[Fotakis et al., 2005]:

- n keys, table of size m
- k hash functions $h_i : U \to \{1, \ldots, m\}$

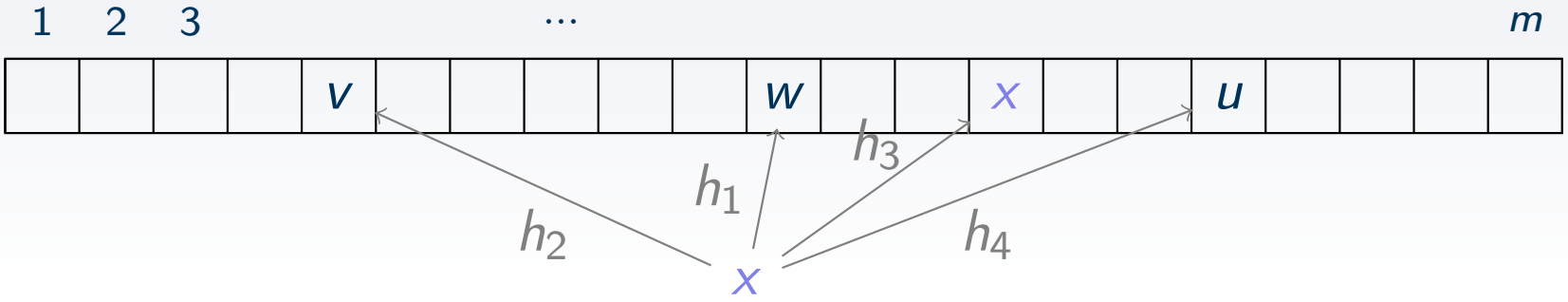
k-ary Cuckoo Hashing



[Fotakis et al., 2005]:

- n keys, table of size m
- k hash functions $h_i : U \to \{1, \ldots, m\}$
- store key x in one of $T[h_i(x)]$, $i = 1, \ldots, k$

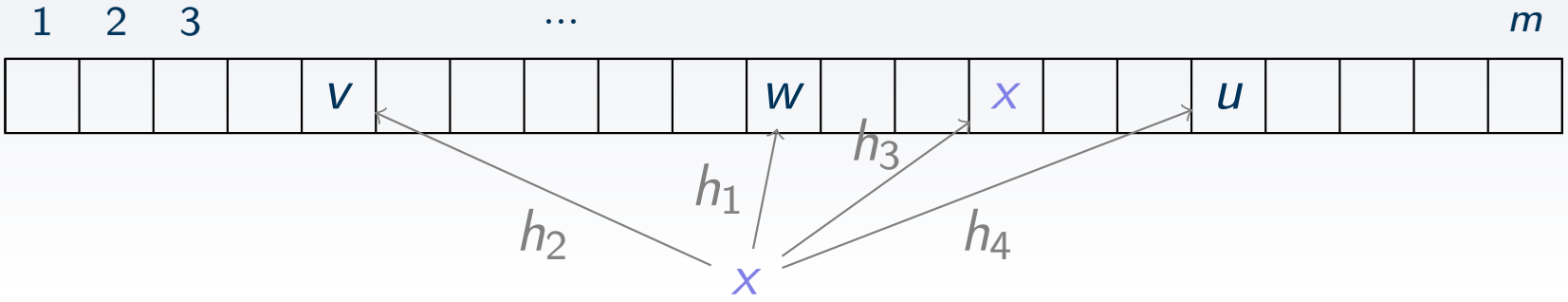
k-ary Cuckoo Hashing



[Fotakis et al., 2005]:

- n keys, table of size m
- k hash functions $h_i : U \to \{1, \ldots, m\}$
- store key x in one of $T[h_i(x)]$, $i = 1, \ldots, k$
- If this is possible for all $x \in S$: constant lookup time

k-ary Cuckoo Hashing

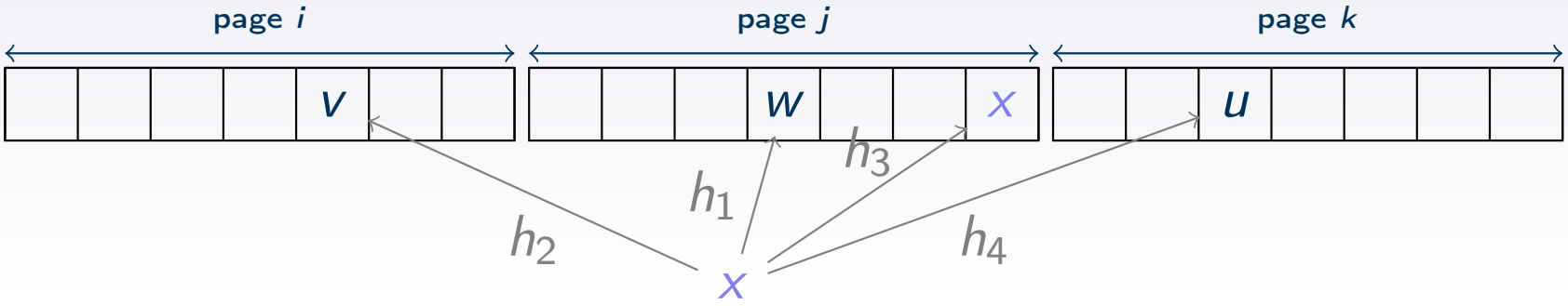


[Fotakis et al., 2005]:

- n keys, table of size m
- k hash functions $h_i : U \to \{1, \ldots, m\}$
- store key x in one of $T[h_i(x)]$, $i = 1, \ldots, k$
- If this is possible for all $x \in S$: constant lookup time

Downside:

k-ary Cuckoo Hashing

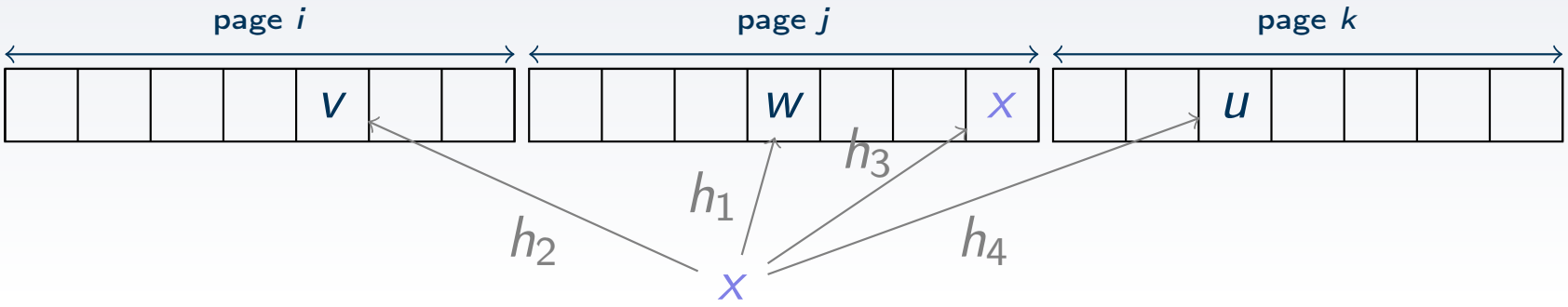


[Fotakis et al., 2005]:

- n keys, table of size m
- k hash functions $h_i : U \to \{1, \ldots, m\}$
- store key x in one of $T[h_i(x)]$, $i = 1, \ldots, k$
- If this is possible for all $x \in S$: constant lookup time

Downside: if memory arranged in pages + costly page requests

k-ary Cuckoo Hashing



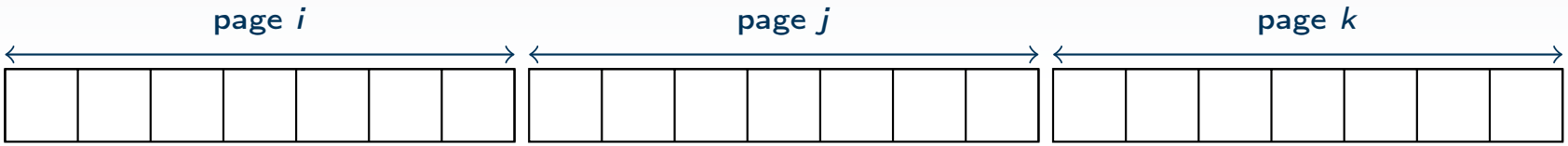
[Fotakis et al., 2005]:

- n keys, table of size m
- k hash functions $h_i : U \to \{1, \ldots, m\}$
- store key x in one of $T[h_i(x)]$, $i = 1, \ldots, k$
- If this is possible for all $x \in S$: constant lookup time

Downside: if memory arranged in pages + costly page requests $\longrightarrow$ **expensive lookups**

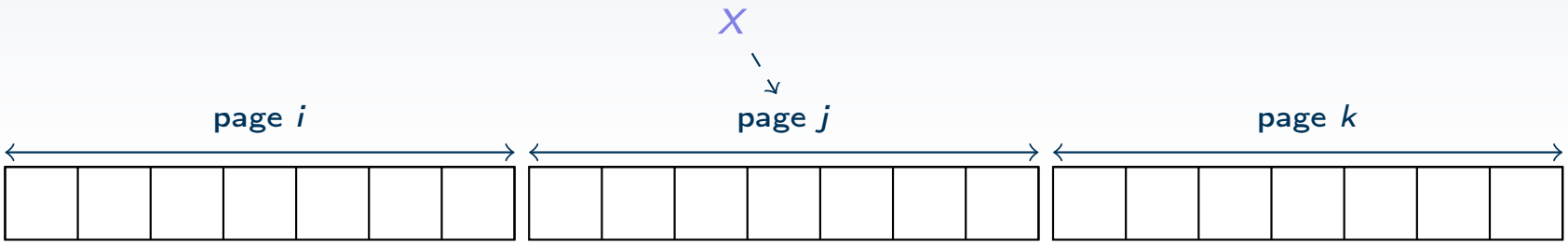
Dealing with Pages (1)

Natural approach:



Dealing with Pages (1)

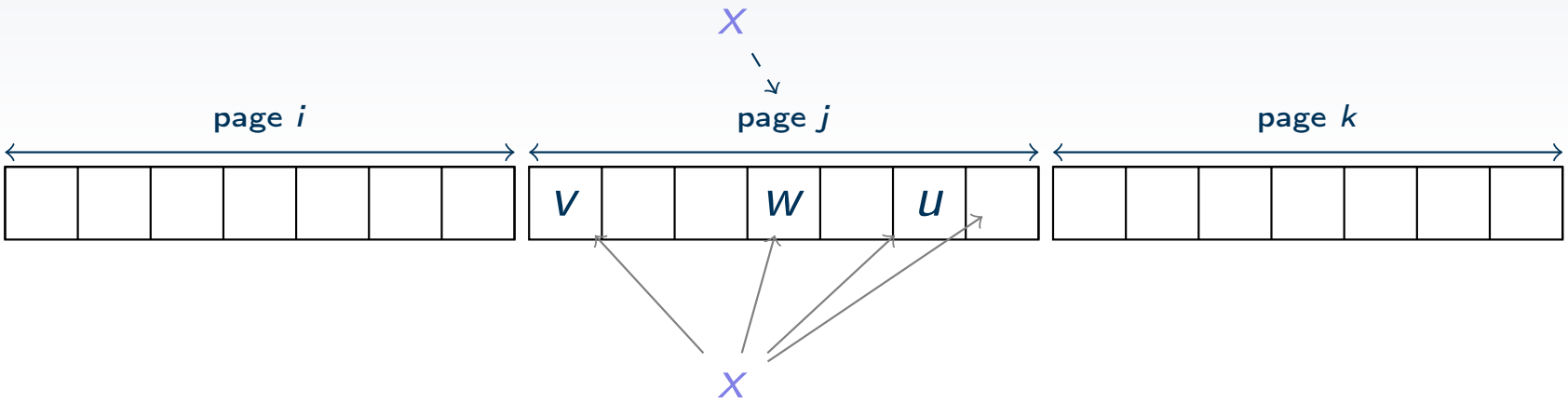
Natural approach:



- map key to a page

Dealing with Pages (1)

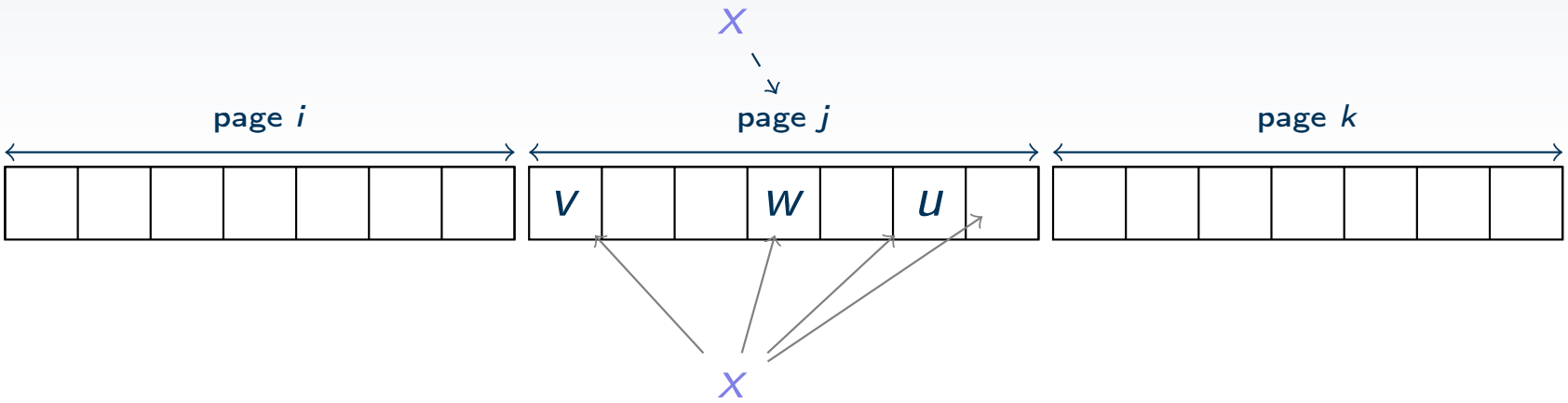
Natural approach:



- map key to a page
- separate cuckoo hash (sub-)table for each page

Dealing with Pages (1)

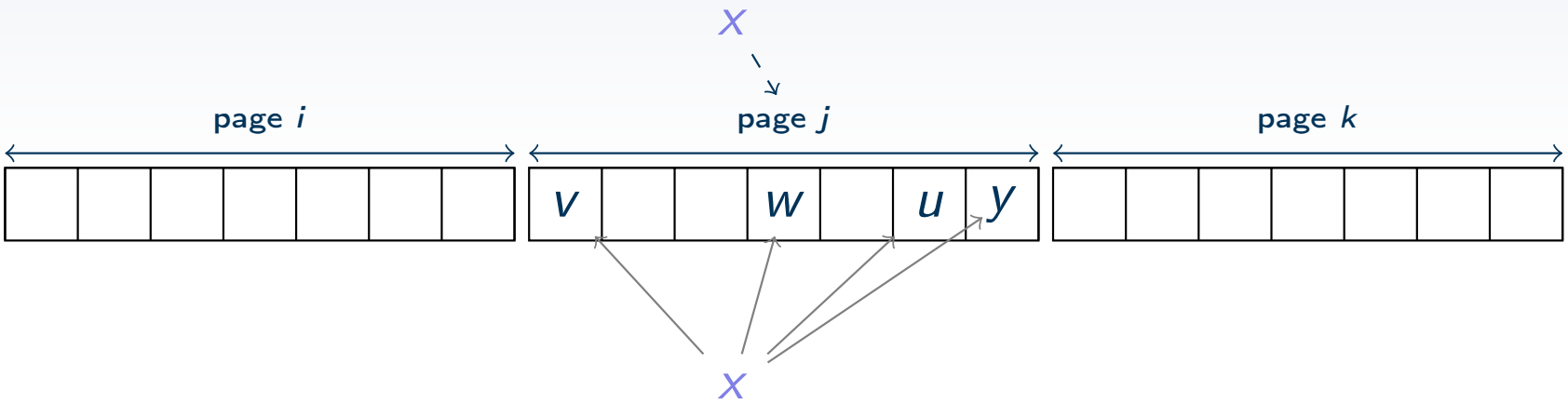
Natural approach:



- map key to a page
- separate cuckoo hash (sub-)table for each page
- **Pro:** only 1 page request

Dealing with Pages (1)

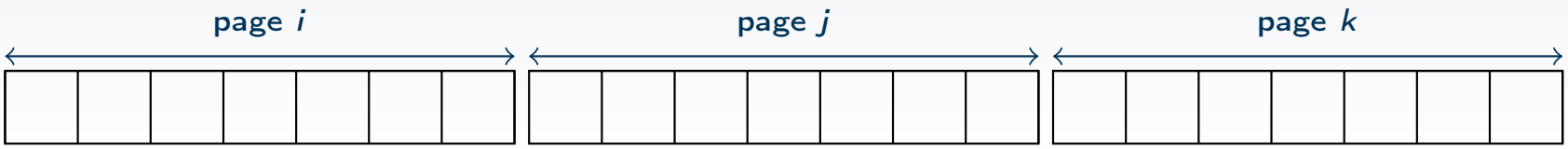
Natural approach:



- map key to a page
- separate cuckoo hash (sub-)table for each page
- **Pro:** only 1 page request
- **Con:** most overloaded sub-table limits load of overall table

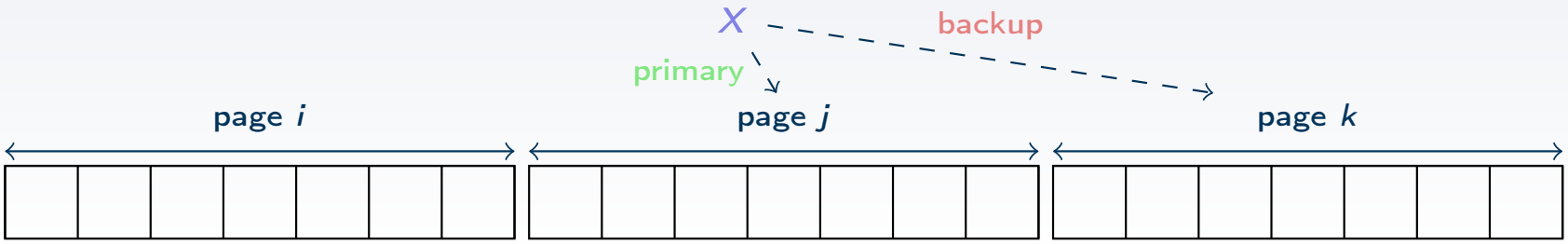
Dealing with Pages (2)

Our approach:



Dealing with Pages (2)

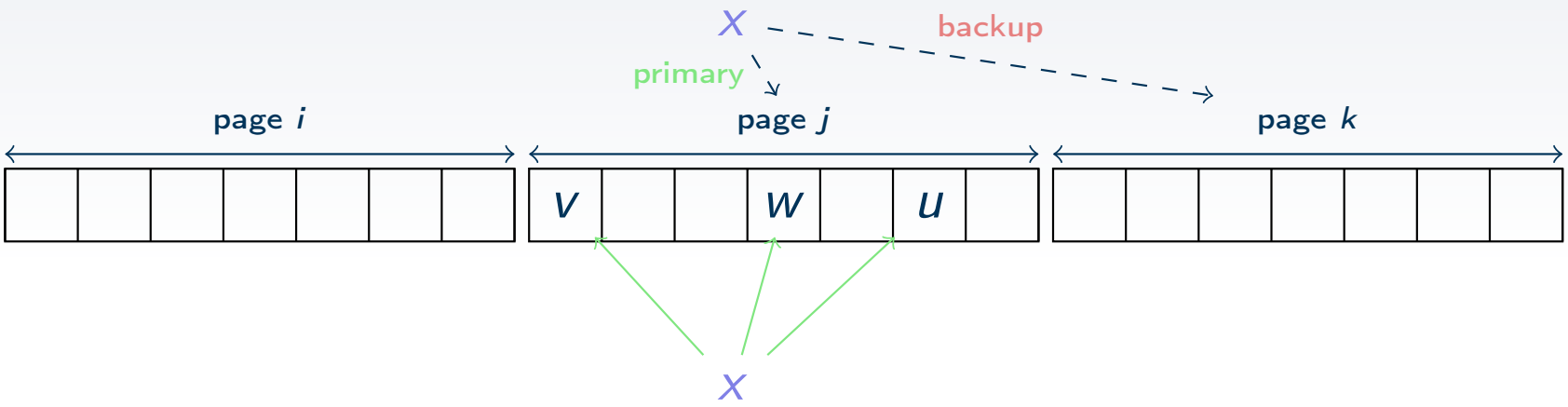
Our approach:



- each key has a primary and a backup page

Dealing with Pages (2)

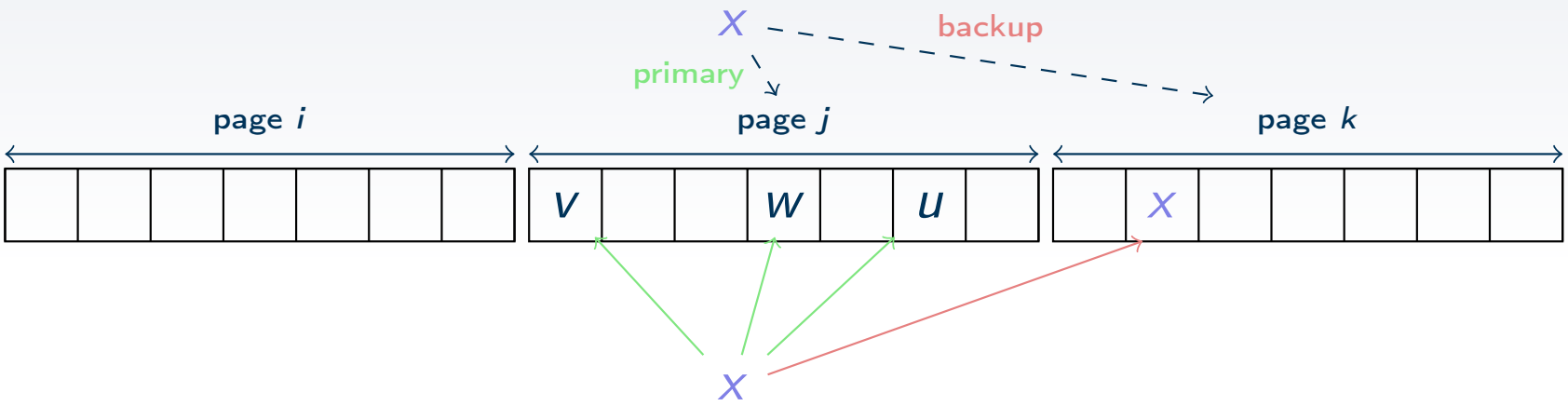
Our approach:



- each key has a primary and a backup page
- force most keys to their primary pages

Dealing with Pages (2)

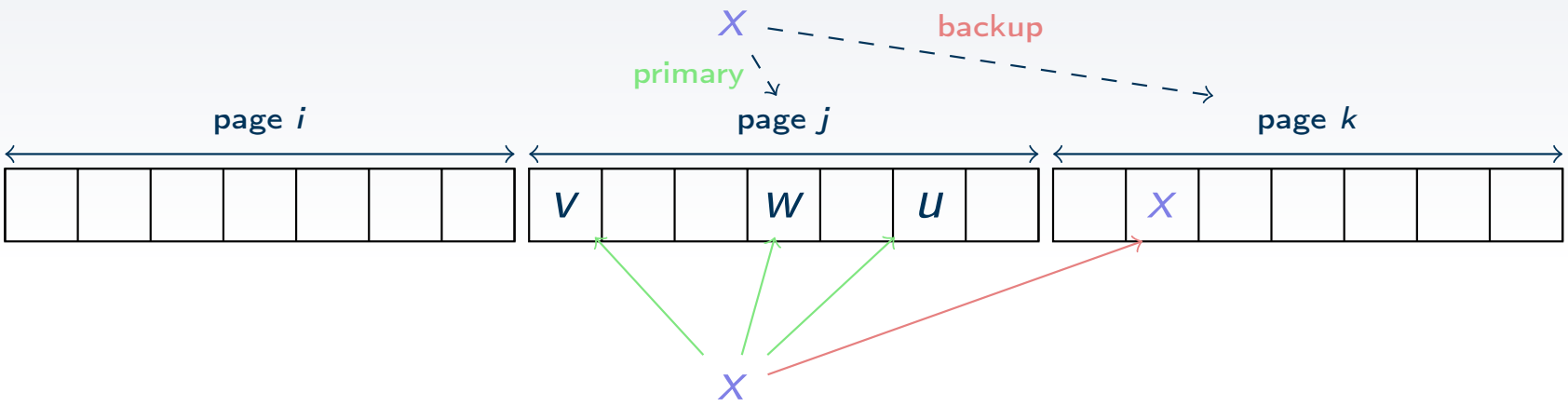
Our approach:



- each key has a primary and a backup page
- force most keys to their primary pages
- backup page allows to adjust load

Dealing with Pages (2)

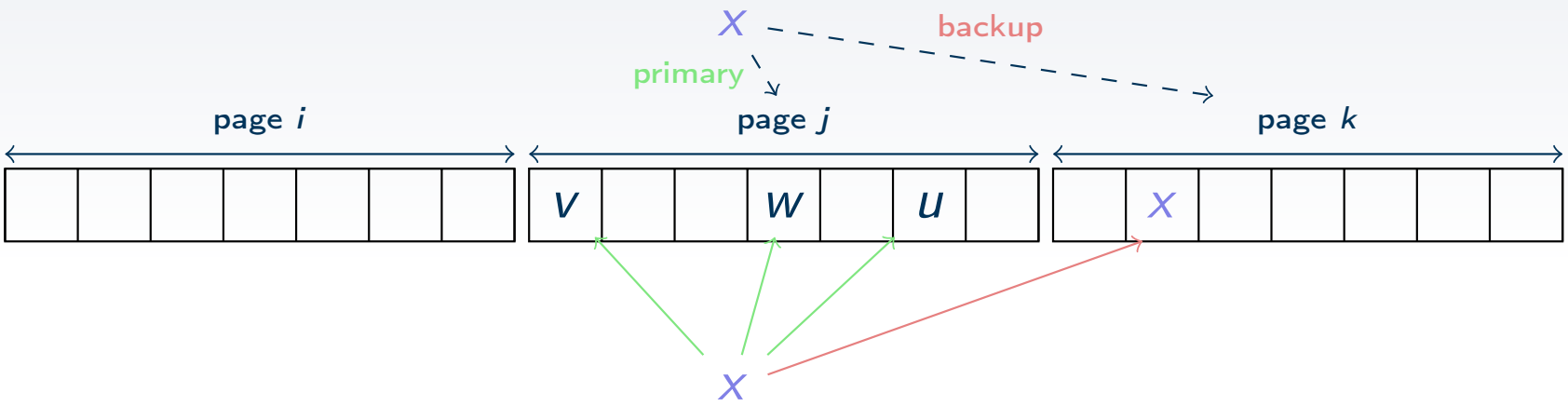
Our approach:



- each key has a primary and a backup page
- force most keys to their primary pages
- backup page allows to adjust load
- **Con:** at most 2 page requests

Dealing with Pages (2)

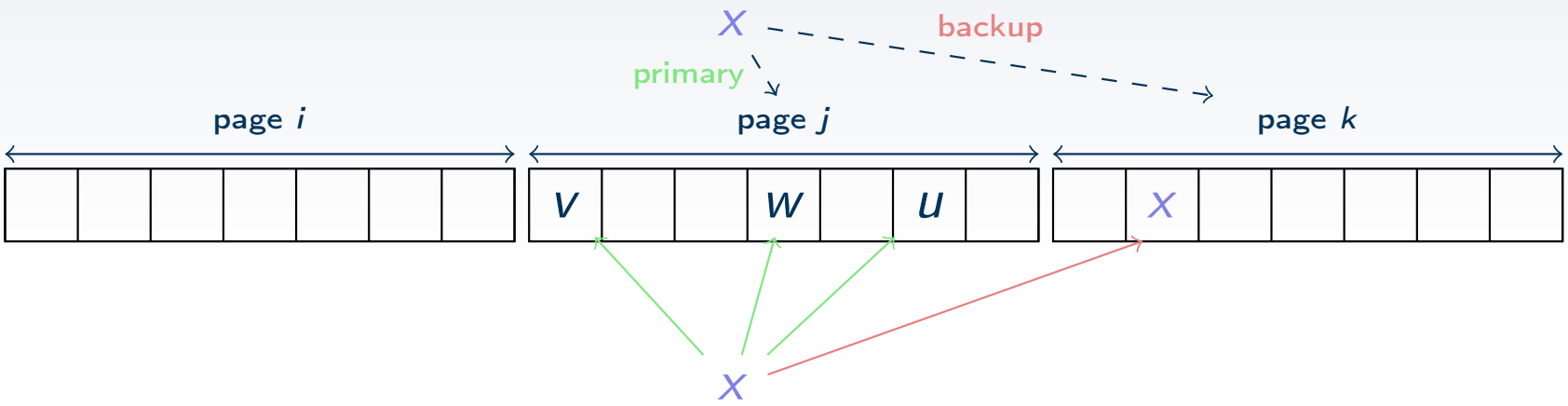
Our approach:



- each key has a primary and a backup page
- force most keys to their primary pages
- backup page allows to adjust load
- ~~**Con:** at most 2 page requests~~
- **Pro:** we achieve close to 1 page request (offline and online) + maximum load near k-ary cuckoo hashing

Dealing with Pages (2)

Our approach:



- each key has a primary and a backup page
- force most keys to their primary pages
- backup page allows to adjust load
- ~~**Con:** at most 2 page requests~~
- **Pro:** we achieve close to 1 page request (offline and online) + maximum load near k-ary cuckoo hashing

Work is entirely experimental!

(Some) Related Work

1 page, k choices, page size $s = m^{\delta}$

k-ary cuckoo hashing
[Frieze and Melsted, 2009, arXiv]
[Fountoulakis and Panagiotou, 2010]
[Dietzfelbinger et al., 2010]

+ cells of capacity ℓ
[Gao and Wormald, 2010]
[Fountoulakis et al., 2011]

(Some) Related Work

1 page, k choices, page size $s = m^{\delta}$

k-ary cuckoo hashing	+ cells of capacity ℓ
[Frieze and Melsted, 2009, arXiv]	[Gao and Wormald, 2010]
[Fountoulakis and Panagiotou, 2010]	[Fountoulakis et al., 2011]
[Dietzfelbinger et al., 2010]	

- asymptotically sharp load thresholds, well understood
- but: for small page sizes no results known

(Some) Related Work

1 page, k choices, page size $s = m^{\delta}$

k-ary cuckoo hashing
[Frieze and Melsted, 2009, arXiv]
[Fountoulakis and Panagiotou, 2010]
[Dietzfelbinger et al., 2010]

+ cells of capacity ℓ
[Gao and Wormald, 2010]
[Fountoulakis et al., 2011]

- asymptotically sharp load thresholds, well understood
- but: for small page sizes no results known

d pages, each with k choices, constant page size

cuckoo-choose-k [Porat and Shalem, 2011, arXiv]

(Some) Related Work

1 page, k choices, page size $s = m^{\delta}$

k-ary cuckoo hashing
[Frieze and Melsted, 2009, arXiv]
[Fountoulakis and Panagiotou, 2010]
[Dietzfelbinger et al., 2010]

+ cells of capacity ℓ
[Gao and Wormald, 2010]
[Fountoulakis et al., 2011]

- asymptotically sharp load thresholds, well understood
- but: for small page sizes no results known

d pages, each with k choices, constant page size

cuckoo-choose-k [Porat and Shalem, 2011, arXiv]

- lower bound on maximum achievable load
- # page requests for lookup not an optimization criterion

(Some) Related Work

1 page, k choices, page size $s = m^{\delta}$

k-ary cuckoo hashing
[Frieze and Melsted, 2009, arXiv]
[Fountoulakis and Panagiotou, 2010]
[Dietzfelbinger et al., 2010]

+ cells of capacity ℓ
[Gao and Wormald, 2010]
[Fountoulakis et al., 2011]

- asymptotically sharp load thresholds, well understood
- but: for small page sizes no results known

d pages, each with k choices, constant page size

cuckoo-choose-k [Porat and Shalem, 2011, arXiv]

- lower bound on maximum achievable load
- # page requests for lookup not an optimization criterion

2 pages, 1 choice on primary, 2 on backup page, $s_p \in \Omega(\varepsilon^{-2} \cdot \log(1/\varepsilon))$

backyard cuckoo hashing [Arbitman et al., 2010]

(Some) Related Work

1 page, k choices, page size $s = m^{\delta}$

k-ary cuckoo hashing
[Frieze and Melsted, 2009, arXiv]
[Fountoulakis and Panagiotou, 2010]
[Dietzfelbinger et al., 2010]

+ cells of capacity ℓ
[Gao and Wormald, 2010]
[Fountoulakis et al., 2011]

- asymptotically sharp load thresholds, well understood
- but: for small page sizes no results known

d pages, each with k choices, constant page size

cuckoo-choose-k [Porat and Shalem, 2011, arXiv]

- lower bound on maximum achievable load
- # page requests for lookup not an optimization criterion

2 pages, 1 choice on primary, 2 on backup page, $s_p \in \Omega(\varepsilon^{-2} \cdot \log(1/\varepsilon))$

backyard cuckoo hashing [Arbitman et al., 2010]

- 1 page request for $1 - \varepsilon$ fraction of keys
- additional perfect hashing scheme for each primary page needed

Experiments

Setup:

- $n = 10^6$ keys, m memory cells (each of capacity 1 key)

Experiments

Setup:

- $n = 10^6$ keys, m memory cells (each of capacity 1 key)
- memory is subdivided into pages of size s, e.g. $s = 100$

Experiments

Setup:

- $n = 10^6$ keys, m memory cells (each of capacity 1 key)
- memory is subdivided into pages of size s, e.g. $s = 100$
- each key fully randomly chooses 2 pages and $k = 4$ cells, 3 cells on the primary page, 1 cell on the backup page

Experiments

Setup:

- $n = 10^6$ keys, m memory cells (each of capacity 1 key)
- memory is subdivided into pages of size s, e.g. $s = 100$
- each key fully randomly chooses 2 pages and $k = 4$ cells, 3 cells on the primary page, 1 cell on the backup page

Aim:

- determine maximum achievable load $c = n/m$ (depending on s)

Experiments

Setup:

- $n = 10^6$ keys, m memory cells (each of capacity 1 key)
- memory is subdivided into pages of size s, e.g. $s = 100$
- each key fully randomly chooses 2 pages and $k = 4$ cells,
  3 cells on the primary page, 1 cell on the backup page

Aim:

- determine maximum achievable load $c = n/m$ (depending on s)
- minimize # page requests for lookup via
  maximize # keys stored on their primary page (primary keys)

Experiments

Setup:

- $n = 10^6$ keys, m memory cells (each of capacity 1 key)
- memory is subdivided into pages of size s, e.g. $s = 100$
- each key fully randomly chooses 2 pages and $k = 4$ cells,
  3 cells on the primary page, 1 cell on the backup page

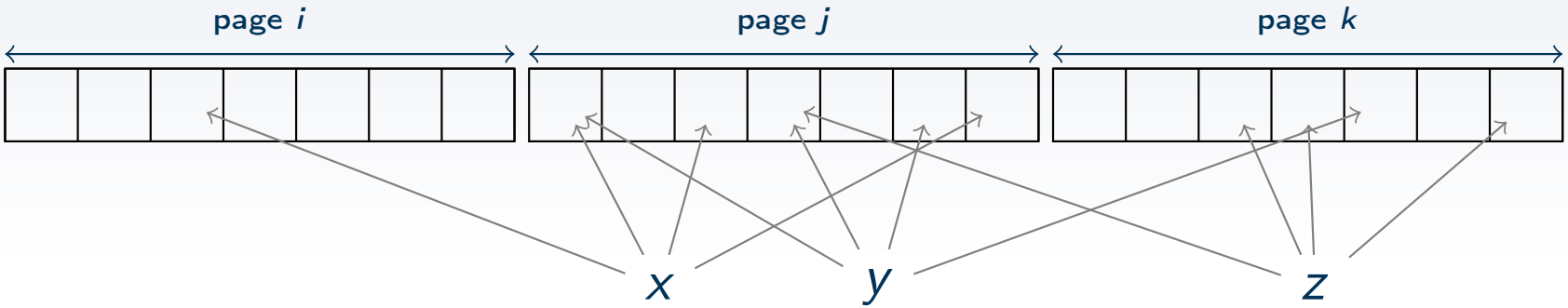
Aim:

- determine maximum achievable load $c = n/m$ (depending on s)
- minimize # page requests for lookup via
  maximize # keys stored on their primary page (primary keys)

Results:

- averages among 100 attempts

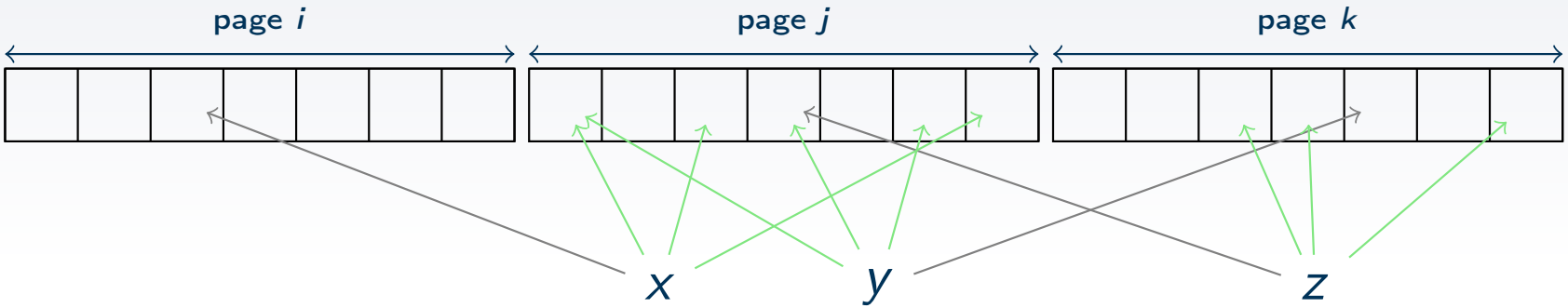
Static Case



Natural Approach:

- assign costs to the edges of the bipartite cuckoo graph

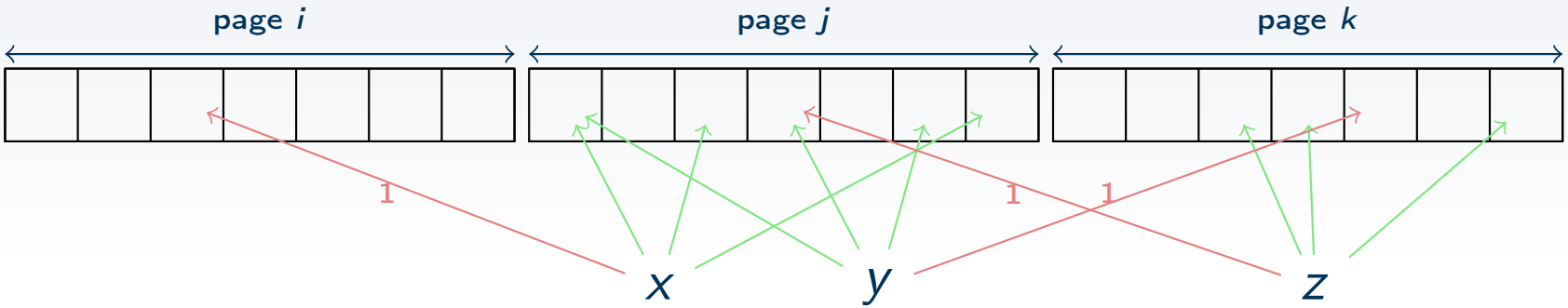
Static Case



Natural Approach:

- assign costs to the edges of the bipartite cuckoo graph
- primary edges have cost 0

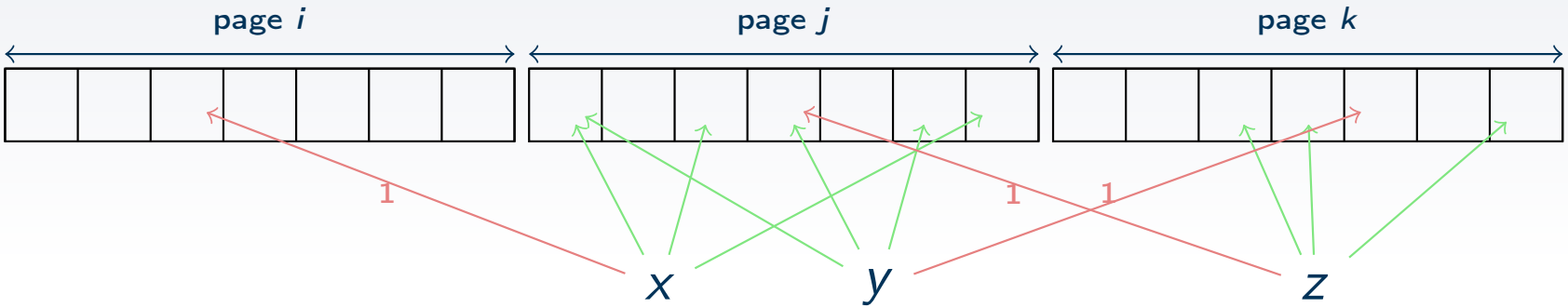
Static Case



Natural Approach:

- assign costs to the edges of the bipartite cuckoo graph
- primary edges have cost 0, backup edges have cost 1

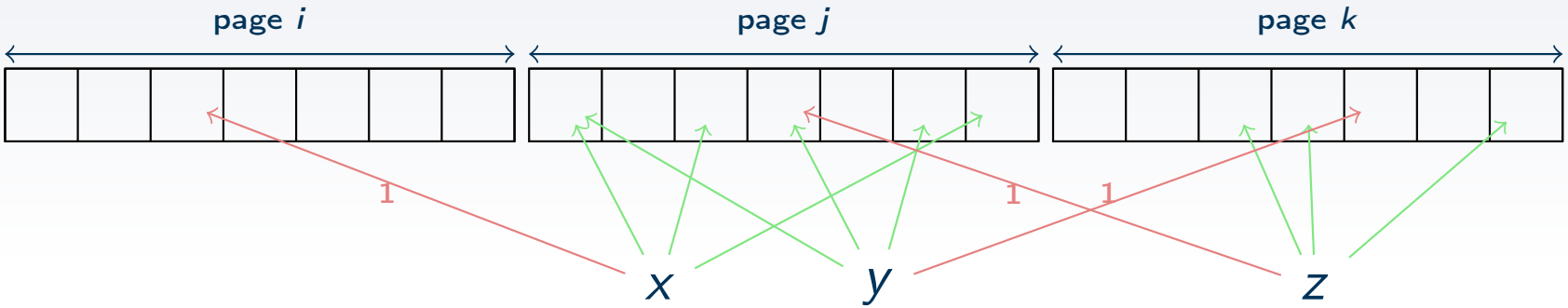
Static Case



Natural Approach:

- assign costs to the edges of the bipartite cuckoo graph
- primary edges have cost 0, backup edges have cost 1
- solve minimum cost matching problem

Static Case

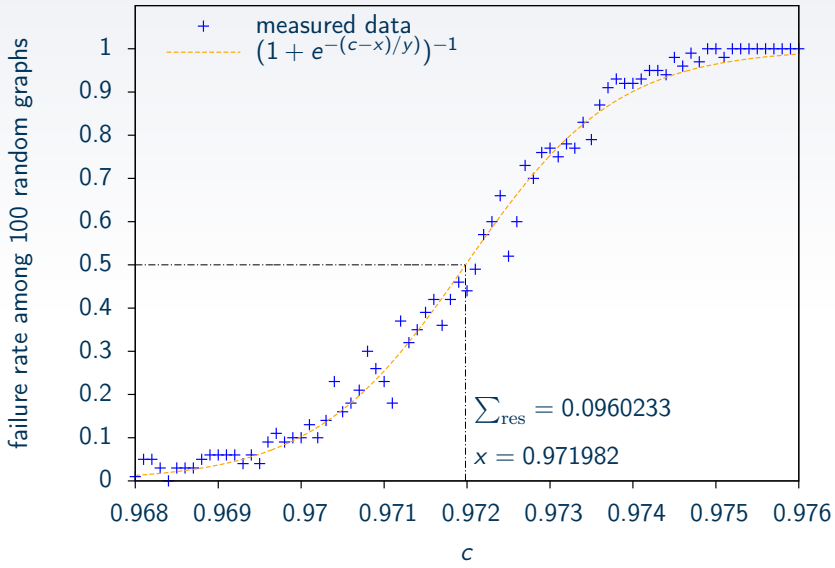


Natural Approach:

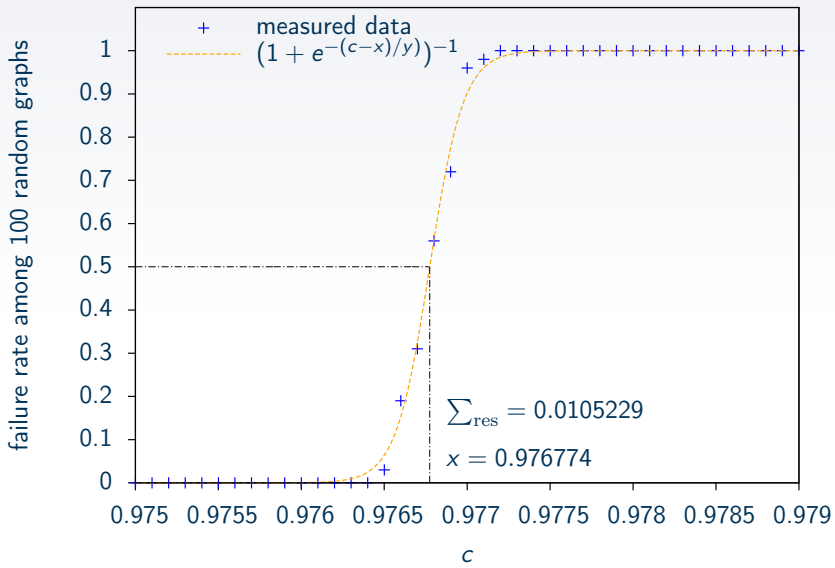
- assign costs to the edges of the bipartite cuckoo graph
- primary edges have cost 0, backup edges have cost 1
- solve minimum cost matching problem
- many algorithms available — used a successive shortest path algorithm (see full version on arXiv)

Comparison of Maximum Load

4 choices, 1 page (no backup)



3 primary, 1 backup choice

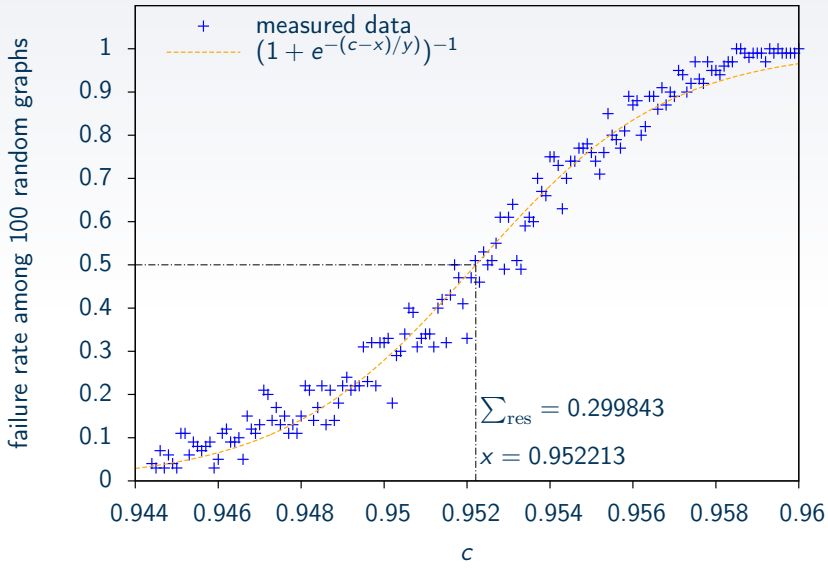


Threshold:

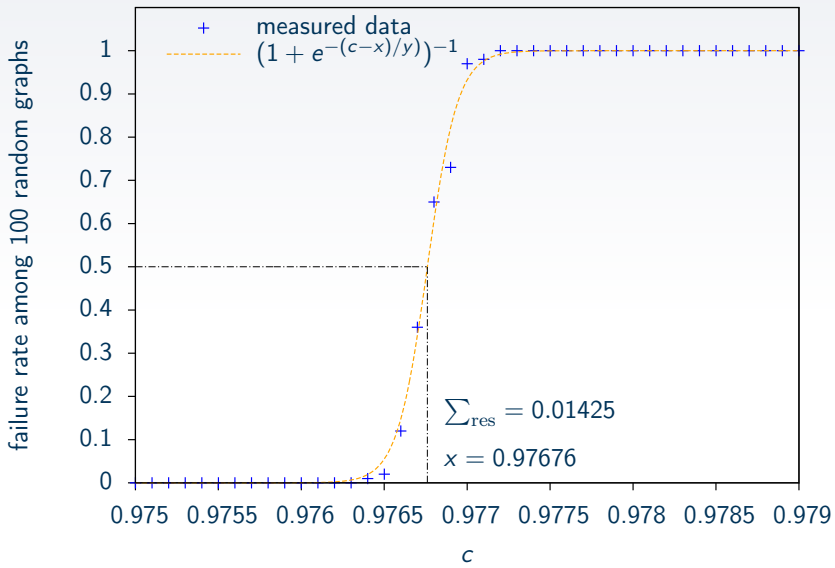
page size	4 choices, 1 page (no backup)	3 primary, 1 backup choice
10^5	0.972	0.977

Comparison of Maximum Load

4 choices, 1 page (no backup)



3 primary, 1 backup choice



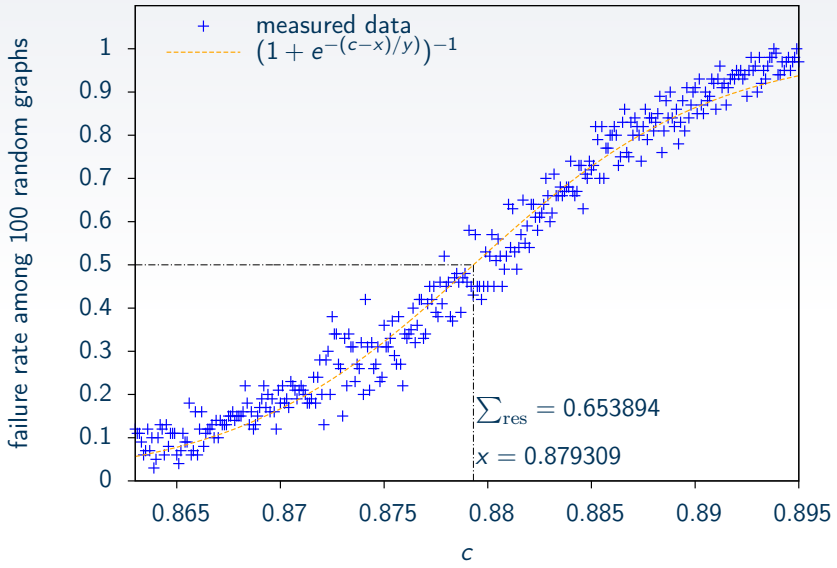
←– interval grows –→

Threshold:

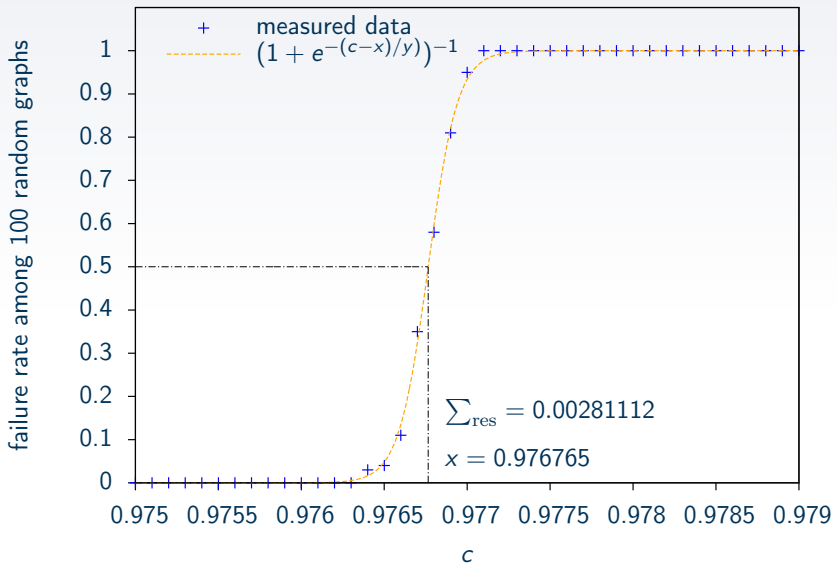
page size	4 choices, 1 page (no backup)	3 primary, 1 backup choice
10^5	0.972	0.977
10^4	0.952	0.977

Comparison of Maximum Load

4 choices, 1 page (no backup)



3 primary, 1 backup choice



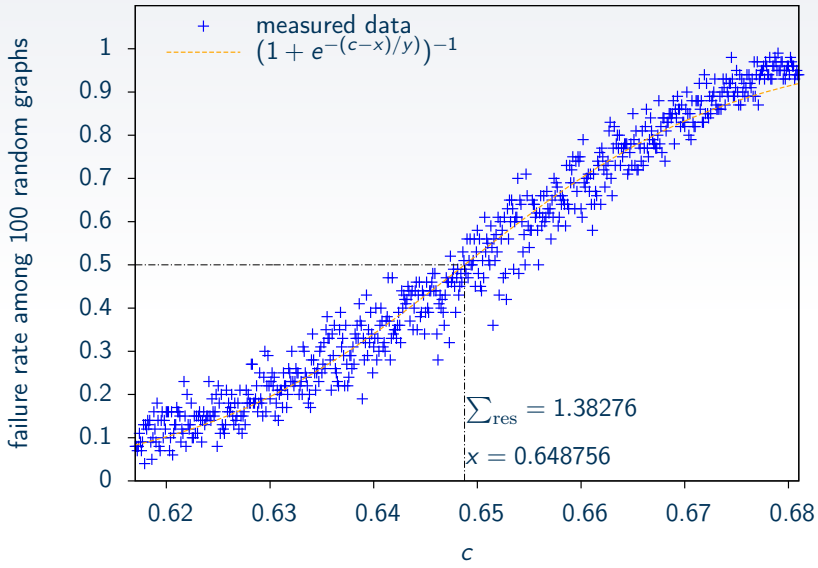
←– interval grows –→

Threshold:

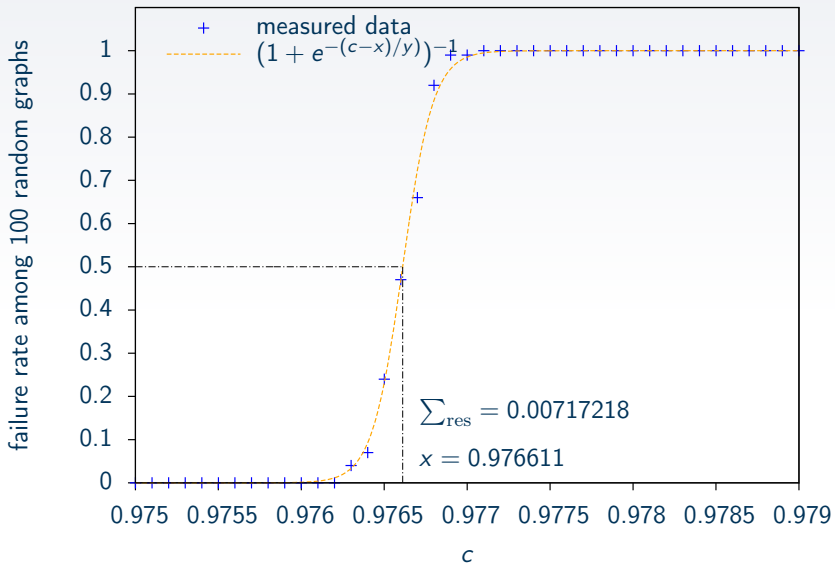
page size	4 choices, 1 page (no backup)	3 primary, 1 backup choice
10^5	0.972	0.977
10^4	0.952	0.977
10^3	0.879	0.977

Comparison of Maximum Load

4 choices, 1 page (no backup)



3 primary, 1 backup choice



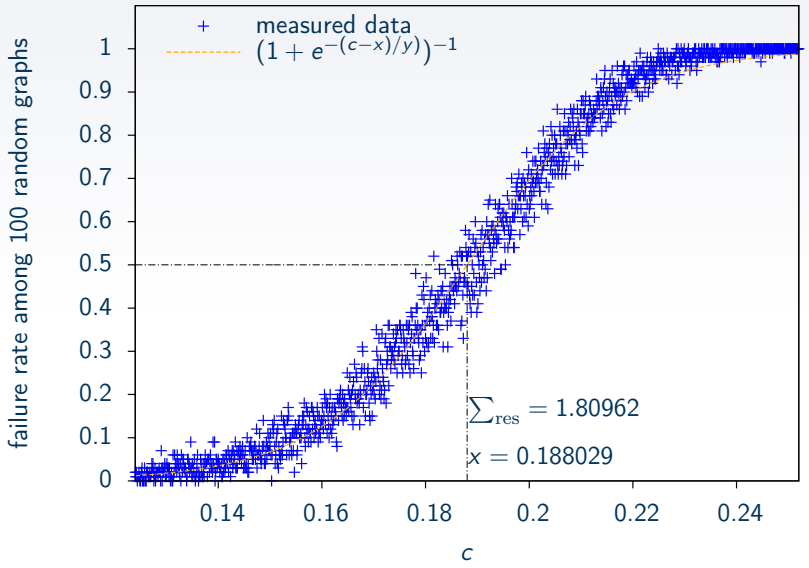
←– interval grows –→

Threshold:

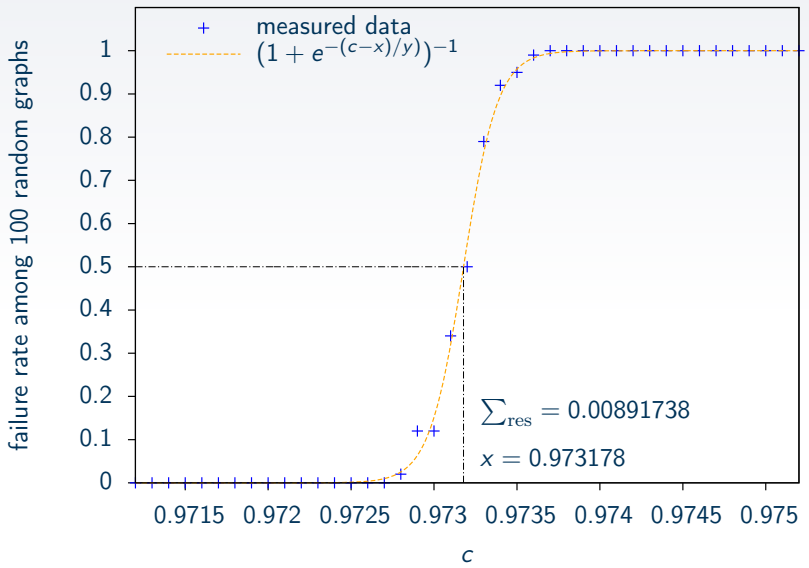
page size	4 choices, 1 page (no backup)	3 primary, 1 backup choice
10^5	0.972	0.977
10^4	0.952	0.977
10^3	0.879	0.977
10^2	0.649	0.977

Comparison of Maximum Load

4 choices, 1 page (no backup)



3 primary, 1 backup choice



←-- interval grows --→

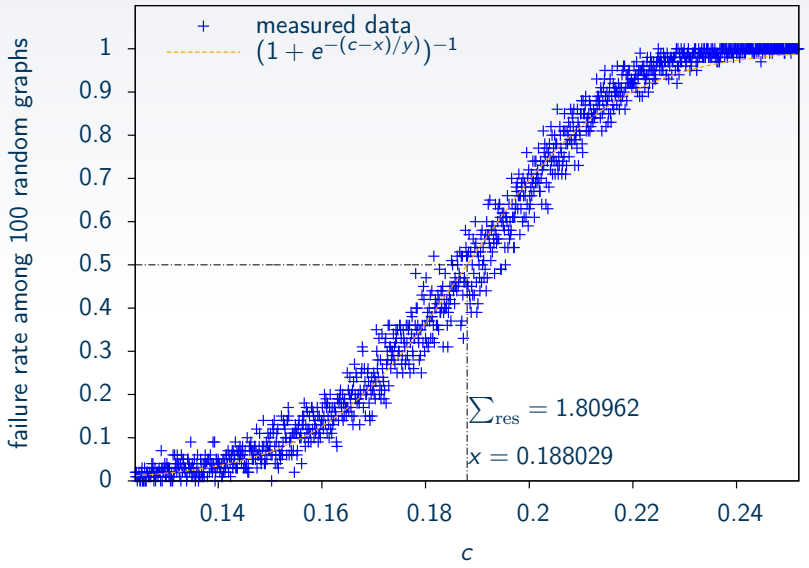
⟵ interval shift

Threshold:

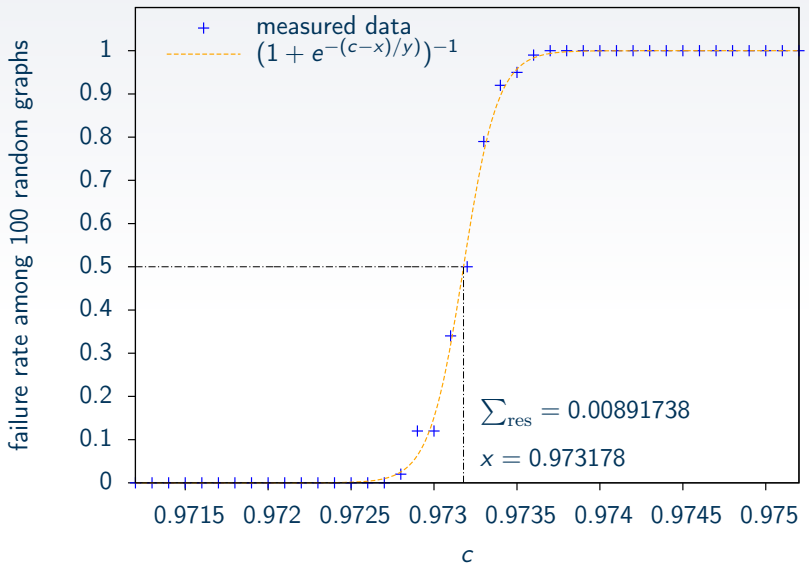
page size	4 choices, 1 page (no backup)	3 primary, 1 backup choice
10^5	0.972	0.977
10^4	0.952	0.977
10^3	0.879	0.977
10^2	0.649	0.977
10^1	0.188	0.973

Comparison of Maximum Load

4 choices, 1 page (no backup)



3 primary, 1 backup choice



←– interval grows –→

⟵ interval shift

Threshold: **decreasing** **stable (≈ 4-ary c.h.)**

page size	4 choices, 1 page (no backup)	3 primary, 1 backup choice
10^5	0.972	0.977
10^4	0.952	0.977
10^3	0.879	0.977
10^2	0.649	0.977
10^1	0.188	0.973

Primary Keys

From now on **only paging with backup option**.

Primary Keys

From now on **only paging with backup option**.

Maximum fraction of primary keys:

load \ page size	10^1	10^3	10^5

Primary Keys

From now on **only paging with backup option**.

Maximum fraction of primary keys:

load \ page size	10^1	10^3	10^5
0.90	0.903	0.996	1

- increases with growing page size

Primary Keys

From now on **only paging with backup option**.

Maximum fraction of primary keys:

load \ page size	10^1	10^3	10^5
0.90	0.903	0.996	1
0.95	0.872	0.974	0.976

- increases with growing page size
- decreases with increasing load

Primary Keys

From now on **only paging with backup option**.

Maximum fraction of primary keys:

load \ page size	10^1	10^3	10^5
0.90	0.903	0.996	1
0.95	0.872	0.974	0.976
0.97	0.850	0.957	0.961

- increases with growing page size
- decreases with increasing load

Primary Keys

From now on **only paging with backup option**.

Maximum fraction of primary keys:

load \ page size	10^1	10^3	10^5
0.90	0.903	0.996	1
0.95	0.872	0.974	0.976
0.97	0.850	0.957	0.961

- increases with growing page size
- decreases with increasing load
- load of primary keys (=fraction · load) can **exceed** threshold for 3-ary c.h. (≈ 0.918)

Page Requests for Lookups

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.974

Page requests X_s — successful searches:

Page Requests for Lookups

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.974

Page requests X_s — successful searches:

trivial: $\mathrm{E}(X_s) = 0.974 \cdot 1 + (1 - 0.974) \cdot 2 < 1.03$

Page Requests for Lookups

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.974

Page requests X_s — successful searches:

trivial: $\mathrm{E}(X_s) = 0.974 \cdot 1 + (1 - 0.974) \cdot 2 < 1.03$

Page requests X_u — unsuccessful searches:

Page Requests for Lookups

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.974

Page requests X_s — successful searches:

trivial: $\mathrm{E}(X_s) = 0.974 \cdot 1 + (1 - 0.974) \cdot 2 < 1.03$

Page requests X_u — unsuccessful searches:

circumvent $X_u = 2$ page requests:

Page Requests for Lookups

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.974

Page requests X_s — successful searches:

trivial: $\mathrm{E}(X_s) = 0.974 \cdot 1 + (1 - 0.974) \cdot 2 < 1.03$

Page requests X_u — unsuccessful searches:

circumvent $X_u = 2$ page requests:

- for each page p determine set B
  = keys with primary page p but
  stored in backup page

Page Requests for Lookups

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.974

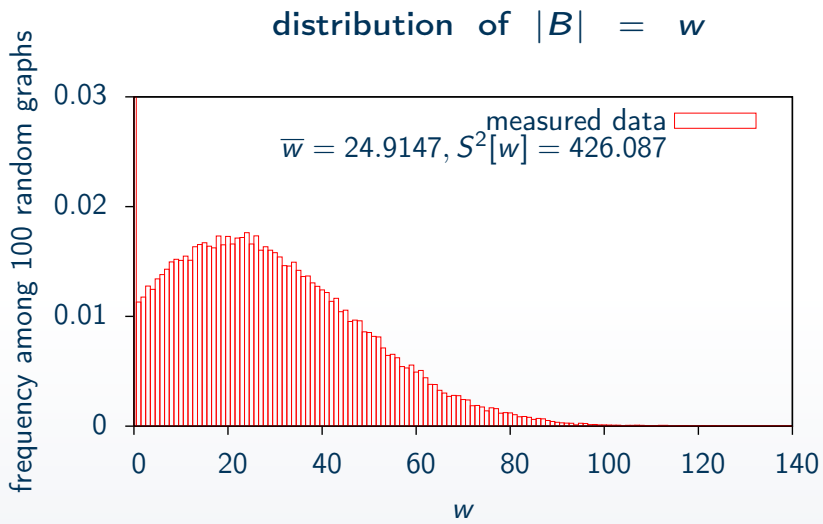
Page requests X_s — successful searches:

trivial: $\mathrm{E}(X_s) = 0.974 \cdot 1 + (1 - 0.974) \cdot 2 < 1.03$

Page requests X_u — unsuccessful searches:

circumvent $X_u = 2$ page requests:

- for each page p determine set B = keys with primary page p but stored in backup page
- store additional set membership tester for each page representing B



Page Requests for Lookups

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.974

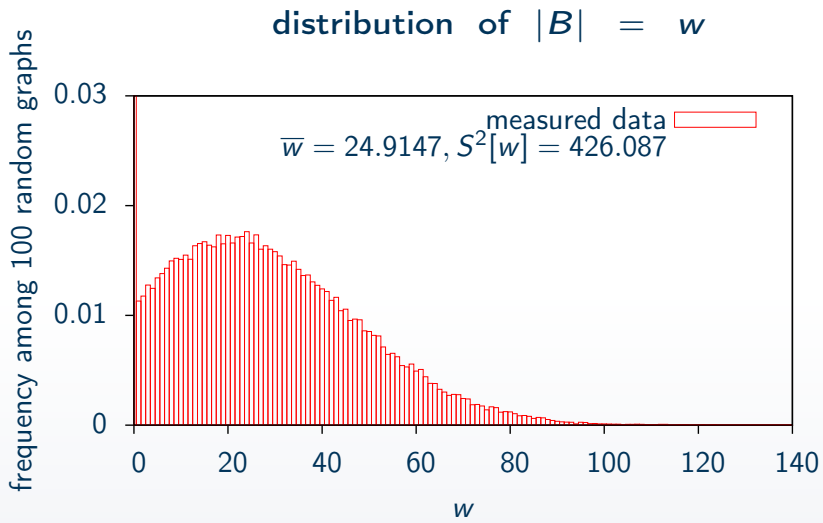
Page requests X_s — successful searches:

trivial: $\mathrm{E}(X_s) = 0.974 \cdot 1 + (1 - 0.974) \cdot 2 < 1.03$

Page requests X_u — unsuccessful searches:

circumvent $X_u = 2$ page requests:

- for each page p determine set B = keys with primary page p but stored in backup page
- store additional set membership tester for each page representing B
- e.g. Bloom filter with 3 hash functions + 1 bit per page cell $\Rightarrow$ 0.15 % false positives



Page Requests for Lookups

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.974

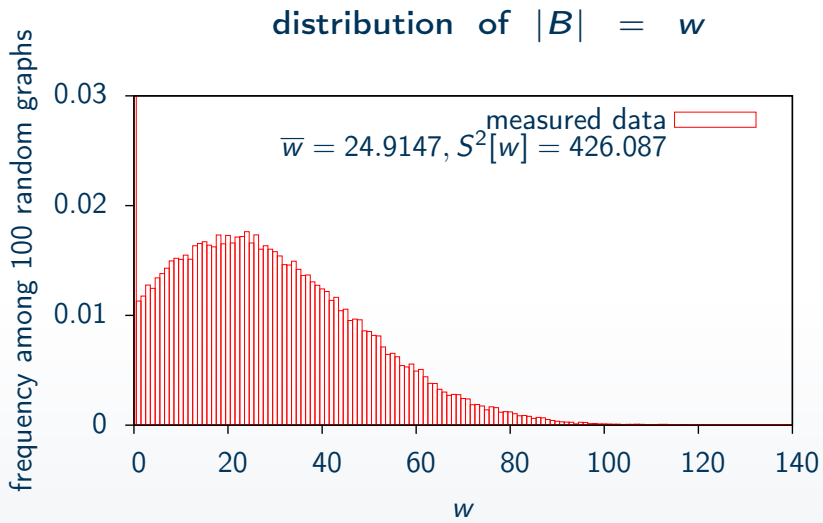
Page requests X_s — successful searches:

trivial: $\mathrm{E}(X_s) = 0.974 \cdot 1 + (1 - 0.974) \cdot 2 < 1.03$

Page requests X_u — unsuccessful searches:

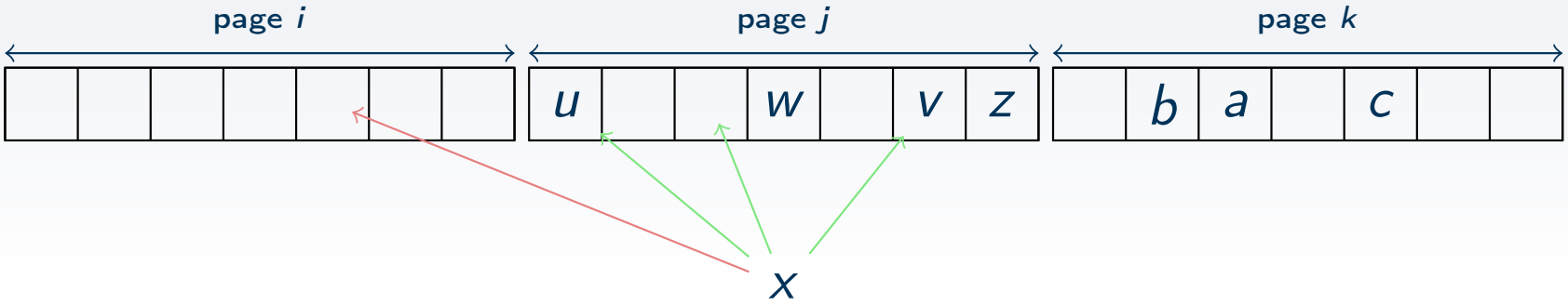
circumvent $X_u = 2$ page requests:

- for each page p determine set B = keys with primary page p but stored in backup page
- store additional set membership tester for each page representing B
- e.g. Bloom filter with 3 hash functions + 1 bit per page cell $\Rightarrow$ 0.15 % false positives



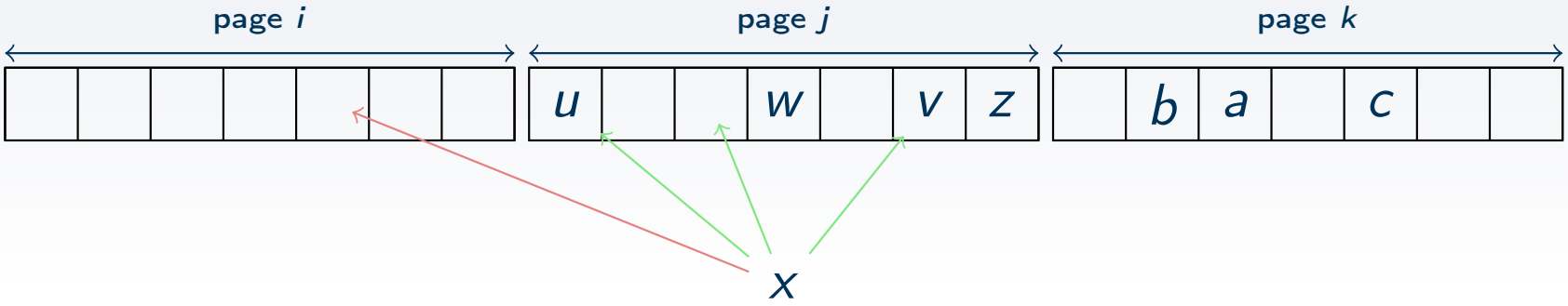
$\Rightarrow \mathrm{E}(X_u) < 1.0015$

Dynamic Case



Simple Random-Walk:

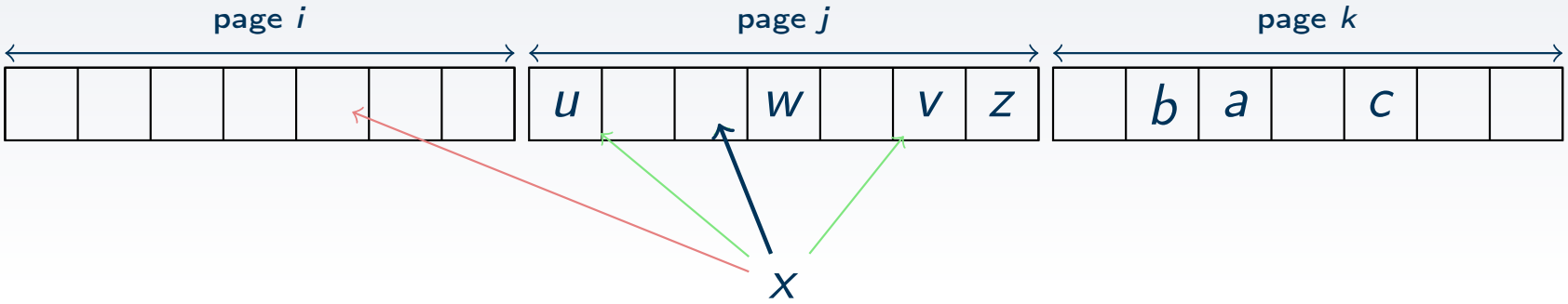
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position:

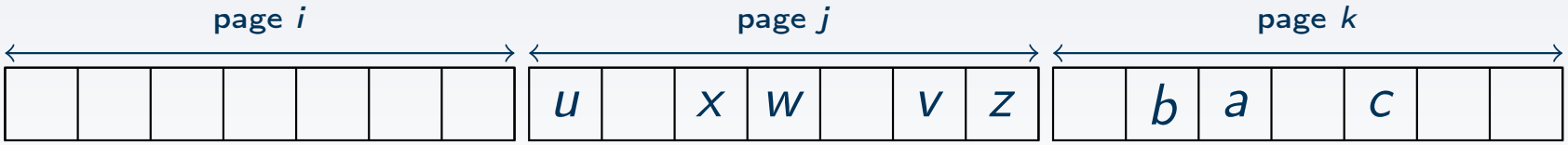
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position:

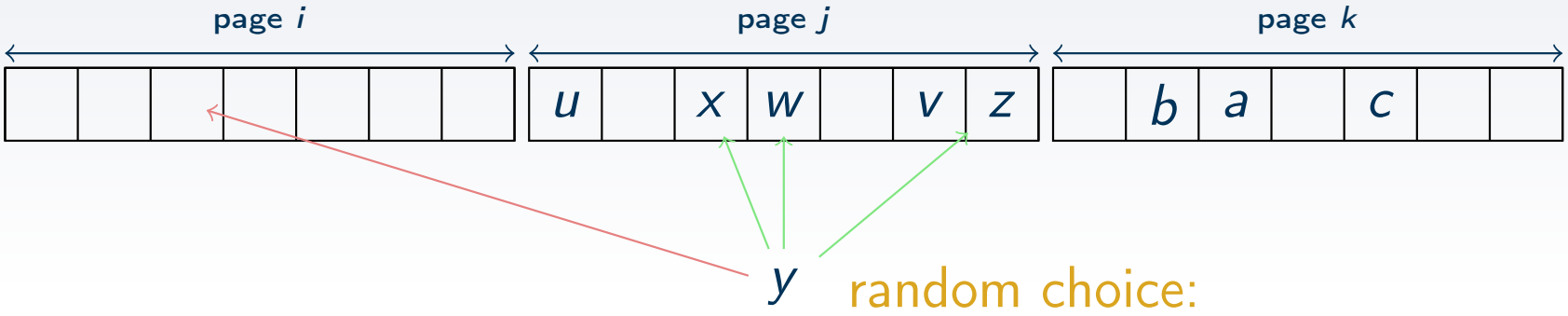
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position: place key (finished)

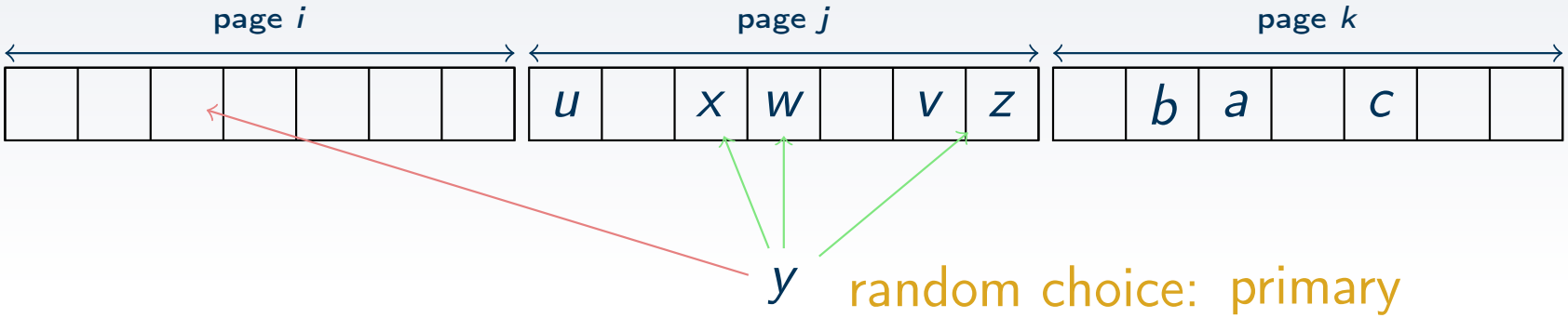
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)

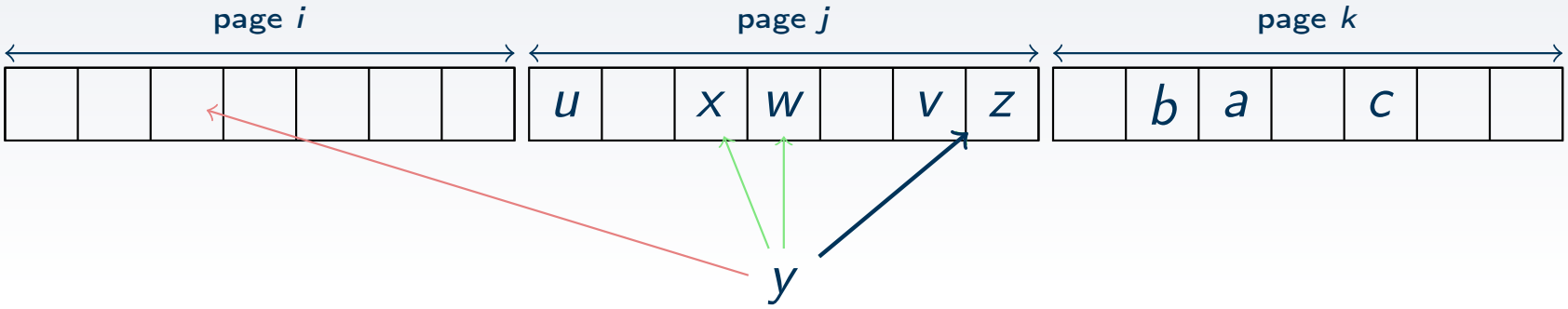
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)
   2.1 if primary page:

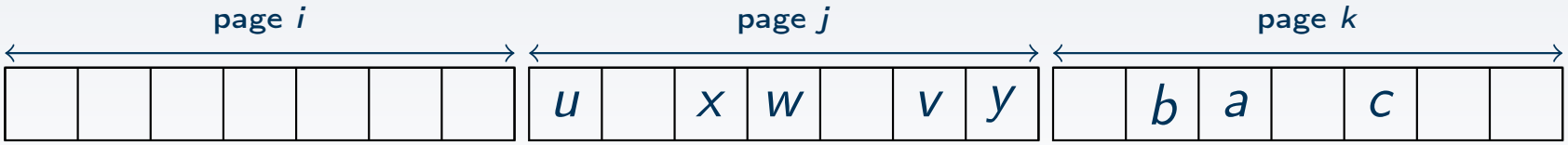
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)
   2.1 if primary page: choose random position

Dynamic Case

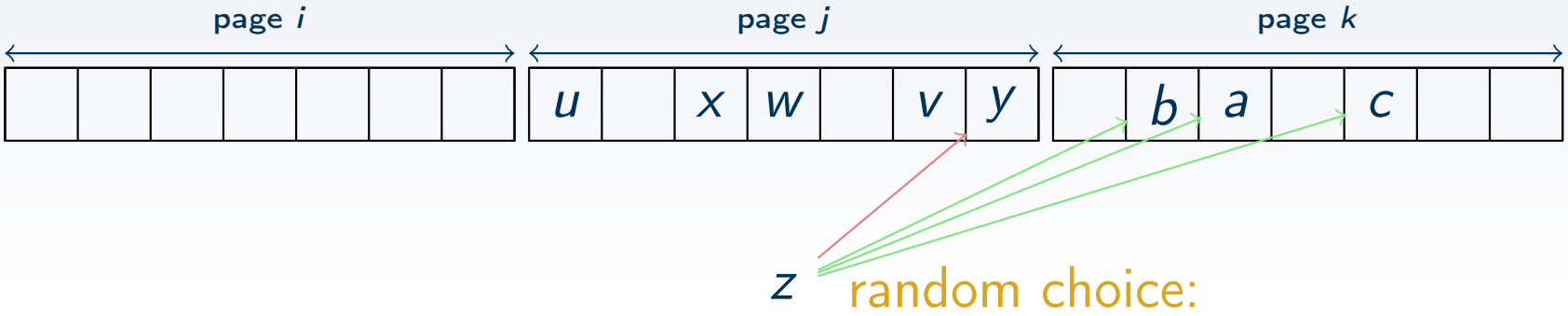


z

Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)
   2.1 if primary page: choose random position and swap keys (goto 1.)

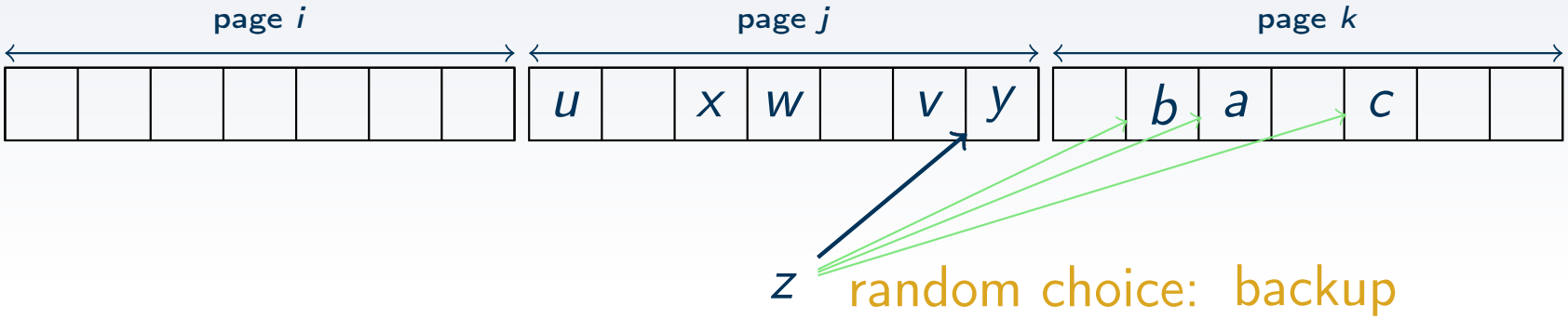
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)
   2.1 if primary page: choose random position and swap keys (goto 1.)

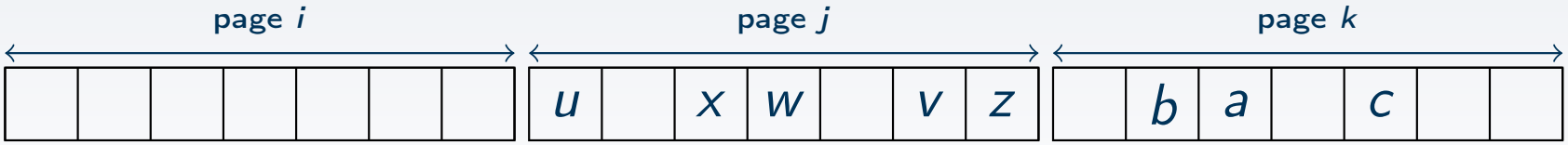
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)
   - 2.1 if primary page: choose random position and swap keys (goto 1.)
   - 2.2 if backup page:

Dynamic Case

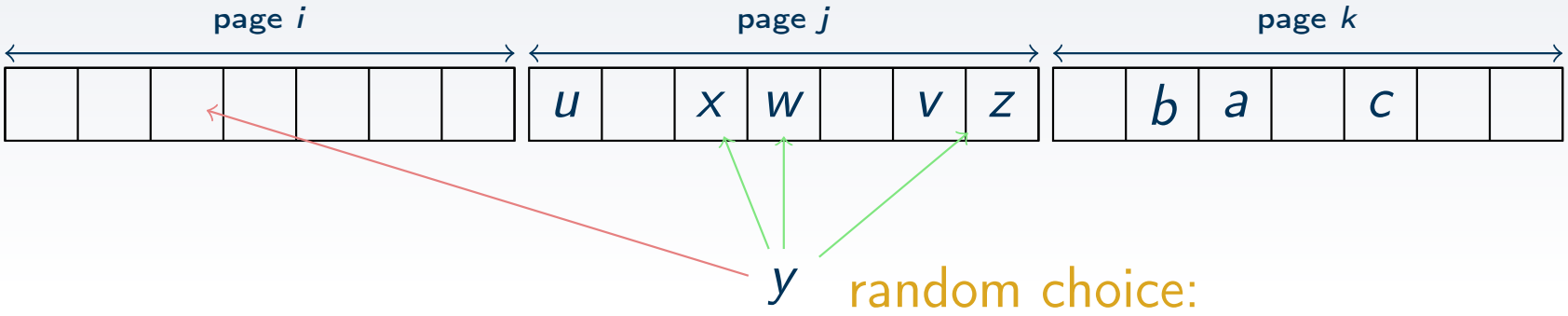


y

Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)
   - 2.1 if primary page: choose random position and swap keys (goto 1.)
   - 2.2 if backup page:
     - 2.2.2 swap keys (goto 1.)

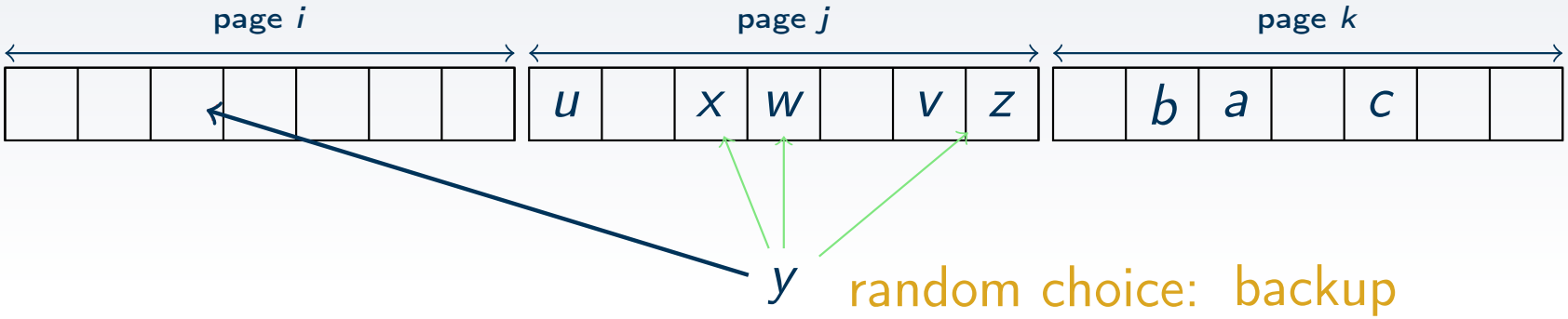
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)
   - 2.1 if primary page: choose random position and swap keys (goto 1.)
   - 2.2 if backup page:
     - 2.2.2 swap keys (goto 1.)

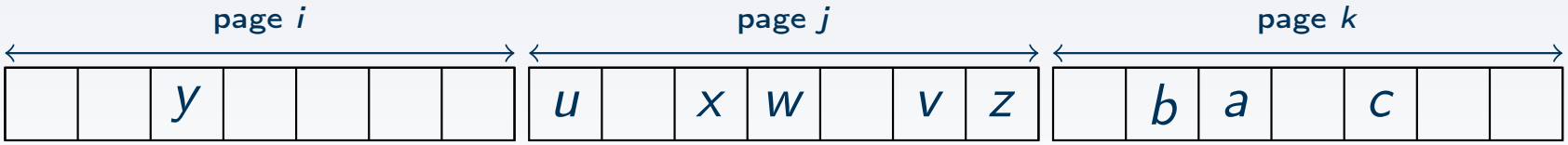
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)
   - 2.1 if primary page: choose random position and swap keys (goto 1.)
   - 2.2 if backup page:
     - 2.2.1 if position is free:
     - 2.2.2 otherwise: swap keys (goto 1.)

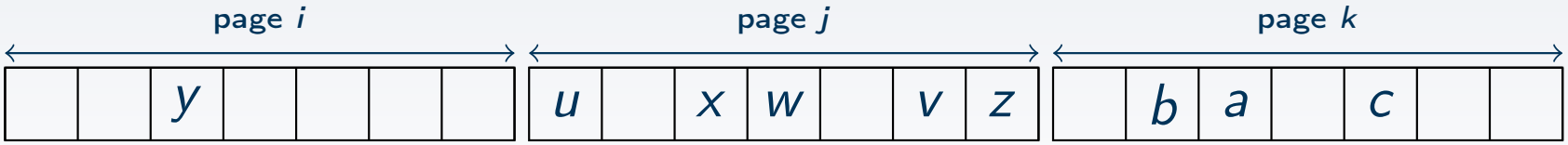
Dynamic Case



Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)
   - 2.1 if primary page: choose random position and swap keys (goto 1.)
   - 2.2 if backup page:
     - 2.2.1 if position is free: place key (finished)
     - 2.2.2 otherwise: swap keys (goto 1.)

Dynamic Case



Simple Random-Walk:

1. if there is a free primary position: place key (finished)
2. otherwise: choose page randomly (e.g. 95% primary, 5% backup)
   - 2.1 if primary page: choose random position and swap keys (goto 1.)
   - 2.2 if backup page:
     - 2.2.1 if position is free: place key (finished)
     - 2.2.2 otherwise: swap keys (goto 1.)

- Parameter: bias, e.g. choose primary page with probability 0.95

Dynamic vs Static (1)

Fraction of primary keys:

page size:		10^2	10^3
bias	load		
0.97	0.95		
0.90	0.97		

Dynamic vs Static (1)

Fraction of primary keys:

page size:		10^2	10^3
bias	load		
0.97	0.95	0.938 [0.955]	0.956 [0.974]
0.90	0.97		

- missed maximum achievable fraction [static case] up to 0.02 **for load** $c = 0.95$

Dynamic vs Static (1)

Fraction of primary keys:

page size:		10^2	10^3
bias	load		
0.97	0.95	0.938 [0.955]	0.956 [0.974]
0.90	0.97	0.887 [0.957]	0.898 [0.936]

- missed maximum achievable fraction [static case]
  up to 0.02 **for load** $c = 0.95$
  and 0.07 for laod $c = 0.97$

Dynamic vs Static (2)

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.956 [0.974]

Page requests X_s — successful searches:

Dynamic vs Static (2)

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.956 [0.974]

Page requests X_s — successful searches:

- dynamic $\mathrm{E}(X_s) = 1.044$, static $\mathrm{E}(X_s) = 1.026$

Dynamic vs Static (2)

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.956 [0.974]

Page requests X_s — successful searches:

- dynamic $\mathrm{E}(X_s) = 1.044$, static $\mathrm{E}(X_s) = 1.026$

Page requests X_u — unsuccessful searches:

Dynamic vs Static (2)

example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.956 [0.974]

Page requests X_s — successful searches:

- dynamic $\mathrm{E}(X_s) = 1.044$, static $\mathrm{E}(X_s) = 1.026$

Page requests X_u — unsuccessful searches:

- same Bloom filter approach for keys that are stored in backup page

Dynamic vs Static (2)

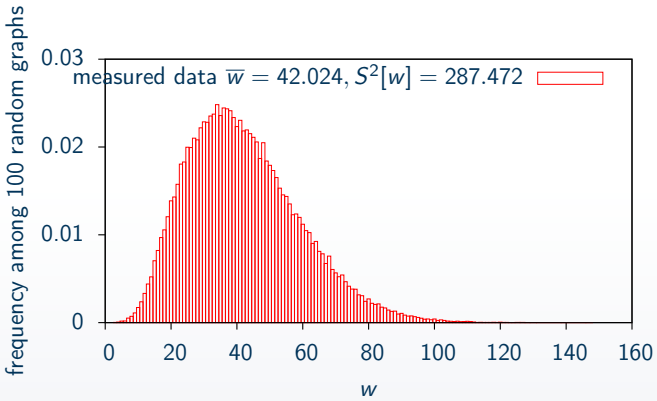
example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.956 [0.974]

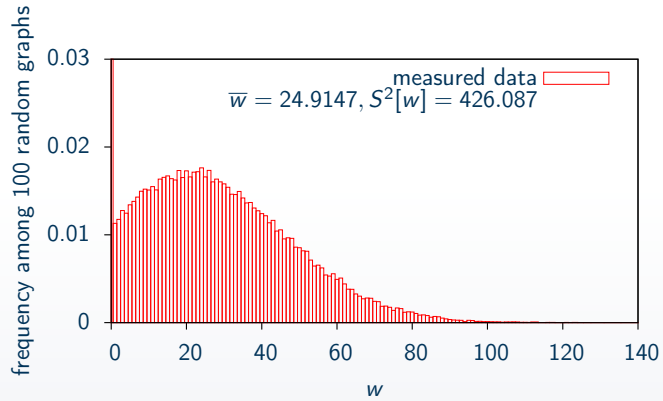
Page requests X_s — successful searches:

- dynamic $\mathrm{E}(X_s) = 1.044$, static $\mathrm{E}(X_s) = 1.026$

Page requests X_u — unsuccessful searches:

- same Bloom filter approach for keys that are stored in backup page





Dynamic vs Static (2)

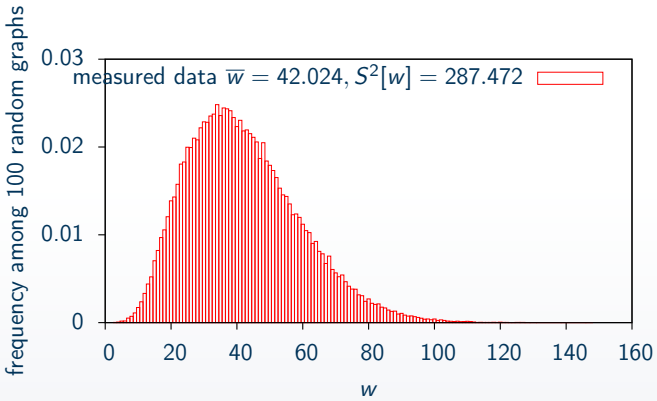
example: $c = 0.95$, $s = 10^3$, fraction of primary keys 0.956 [0.974]

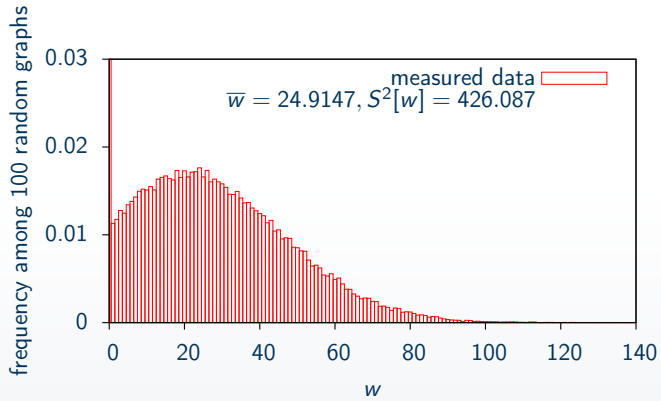
Page requests X_s — successful searches:

- dynamic $\mathrm{E}(X_s) = 1.044$, static $\mathrm{E}(X_s) = 1.026$

Page requests X_u — unsuccessful searches:

- same Bloom filter approach for keys that are stored in backup page





- dynamic $\mathrm{E}(X_u) = 1.0043$, static $\mathrm{E}(X_u) = 1.0015$

Summary

Cuckoo hashing with pages - 3 primary, 1 backup option:

- Offline:

Summary

Cuckoo hashing with pages - 3 primary, 1 backup option:

- Offline:
  - load very close to threshold for 4-ary c.h.

Summary

Cuckoo hashing with pages - 3 primary, 1 backup option:

- Offline:
  - load very close to threshold for 4-ary c.h.
  - load of primary keys near or above threshold for 3-ary c.h.

Summary

Cuckoo hashing with pages - 3 primary, 1 backup option:

- Offline:
  - load very close to threshold for 4-ary c.h.
  - load of primary keys near or above threshold for 3-ary c.h.
- Online: Simple random walk algorithm can be close to optimal offline algorithm.

Summary

Cuckoo hashing with pages - 3 primary, 1 backup option:

- Offline:
  - load very close to threshold for 4-ary c.h.
  - load of primary keys near or above threshold for 3-ary c.h.
- Online: Simple random walk algorithm can be close to optimal offline algorithm.
- Both: Expected number of page requests can be made close to 1.

Summary

Cuckoo hashing with pages - 3 primary, 1 backup option:

- Offline:
  - load very close to threshold for 4-ary c.h.
  - load of primary keys near or above threshold for 3-ary c.h.
- Online: Simple random walk algorithm can be close to optimal offline algorithm.
- Both: Expected number of page requests can be made close to 1.

More results and details, e.g. concerning **performance of the random walk algorithm**, **deletions** and very **small pages**, see ESA paper and full version on arXiv.

Open Problem

Provable performance bounds
for cuckoo hashing with pages

e.g. threshold behavior in offline szenario

References

Arbitman, Y., Naor, M., and Segev, G. (2010).
Backyard Cuckoo Hashing: Constant Worst-Case Operations with a Succinct Representation.
In Proc. 51st FOCS, pages 787–796. IEEE.

Dietzfelbinger, M., Goerdt, A., Mitzenmacher, M., Montanari, A., Pagh, R., and Rink, M. (2010).
Tight Thresholds for Cuckoo Hashing via XORSAT.
In Proc. 37th ICALP (1), pages 213–225.

Fotakis, D., Pagh, R., Sanders, P., and Spirakis, P. G. (2005).
Space Efficient Hash Tables with Worst Case Constant Access Time.
Theory Comput. Syst., 38(2):229–248.

Fountoulakis, N., Khosla, M., and Panagioutou, K. (2011).
The Multiple-orientability Thresholds for Random Hypergraphs.
In Proc. 22nd SODA, pages 1222–1236. SIAM.

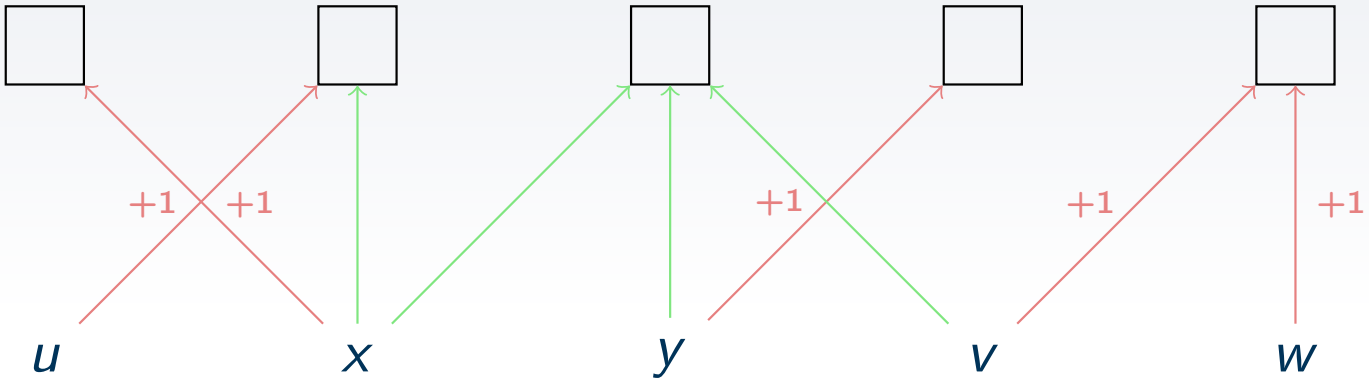
Fountoulakis, N. and Panagiotou, K. (2010).
Orientability of Random Hypergraphs and the Power of Multiple Choices.
In Proc. 37th ICALP (1), pages 348–359. Springer.

Frieze, A. M. and Melsted, P. (2009).
Maximum Matchings in Random Bipartite Graphs and the Space Utilization of Cuckoo Hashtables.
CoRR, abs/0910.5535.

Gao, P. and Wormald, N. C. (2010).
Load balancing and orientability thresholds for random hypergraphs.
In Proc. 42nd STOC, pages 97–104. ACM.

Porat, E. and Shalem, B. (2011).
Another one flew over the cuckoo's nest.
CoRR, abs/1104.5400.

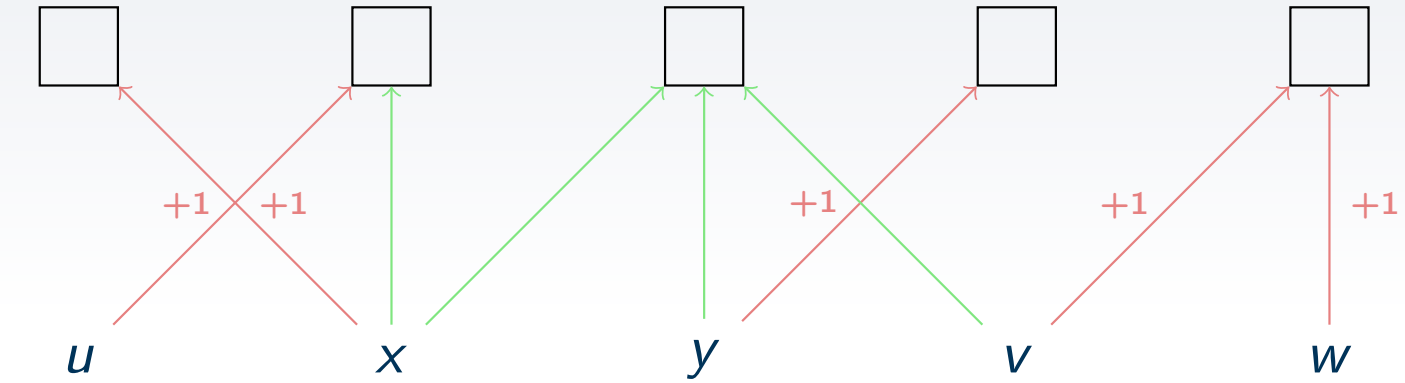
Successive Shortest Path Algorithm



Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1

Successive Shortest Path Algorithm

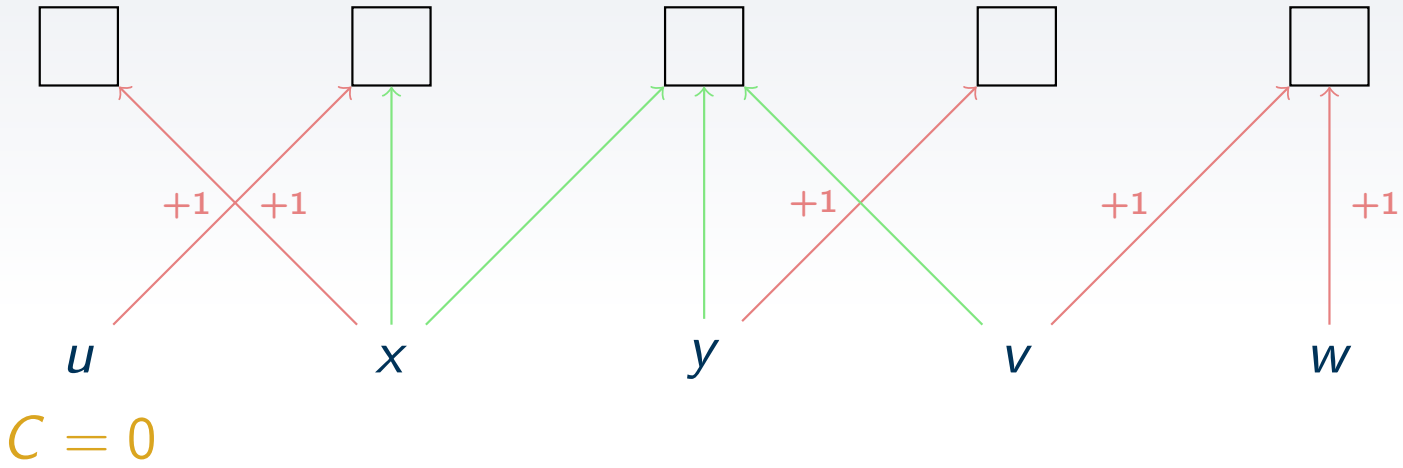


$C = 0$

Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$

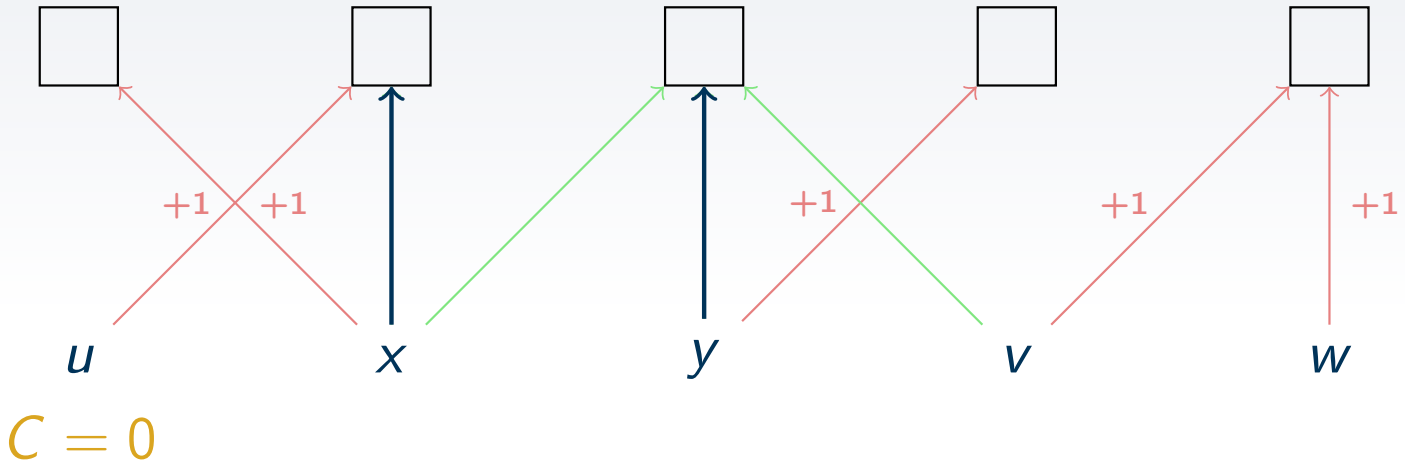
Successive Shortest Path Algorithm



Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path

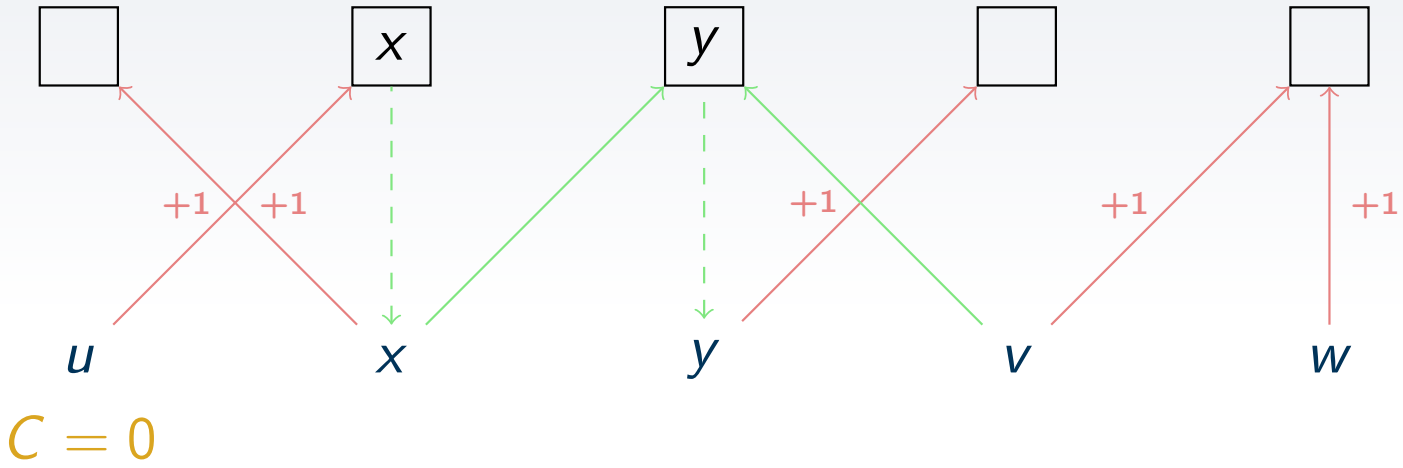
Successive Shortest Path Algorithm



Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path
  - augment matching using all paths with cost C

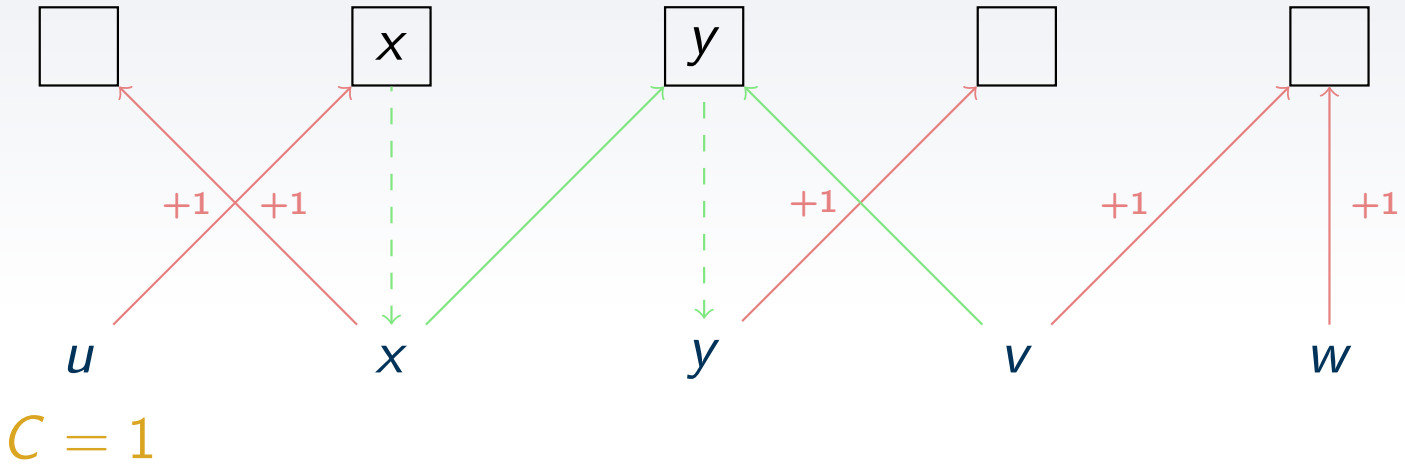
Successive Shortest Path Algorithm



Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path
  - augment matching using all paths with cost C

Successive Shortest Path Algorithm

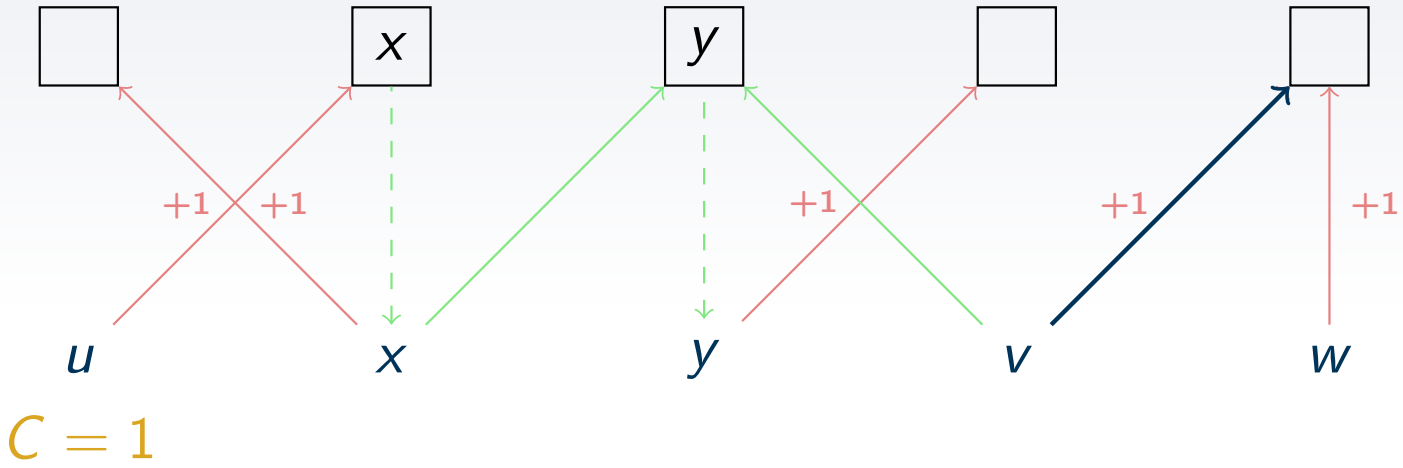


$C = 1$

Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path
  - augment matching using all paths with cost C
  - increment C by 1

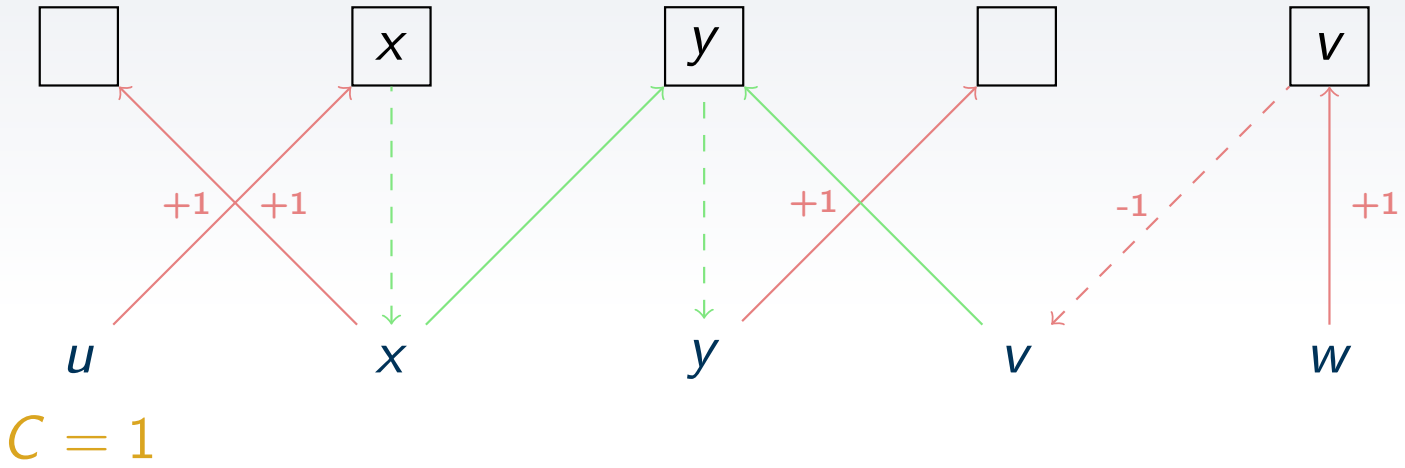
Successive Shortest Path Algorithm



Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path
  - augment matching using all paths with cost C
  - increment C by 1

Successive Shortest Path Algorithm

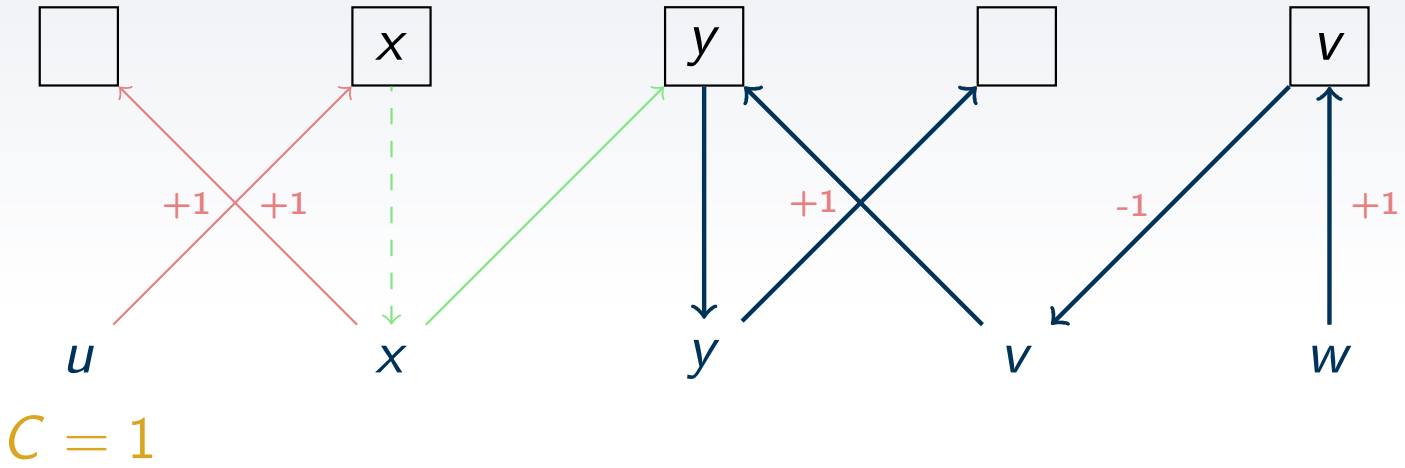


$C = 1$

Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path
  - augment matching using all paths with cost C
  - increment C by 1

Successive Shortest Path Algorithm

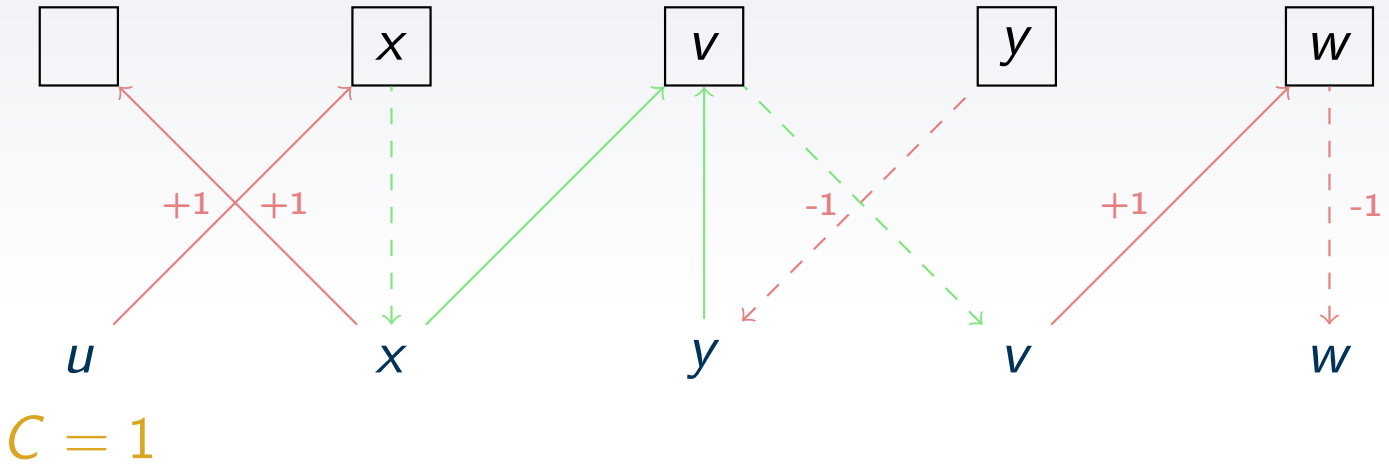


$C = 1$

Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path
  - augment matching using all paths with cost C
  - increment C by 1

Successive Shortest Path Algorithm

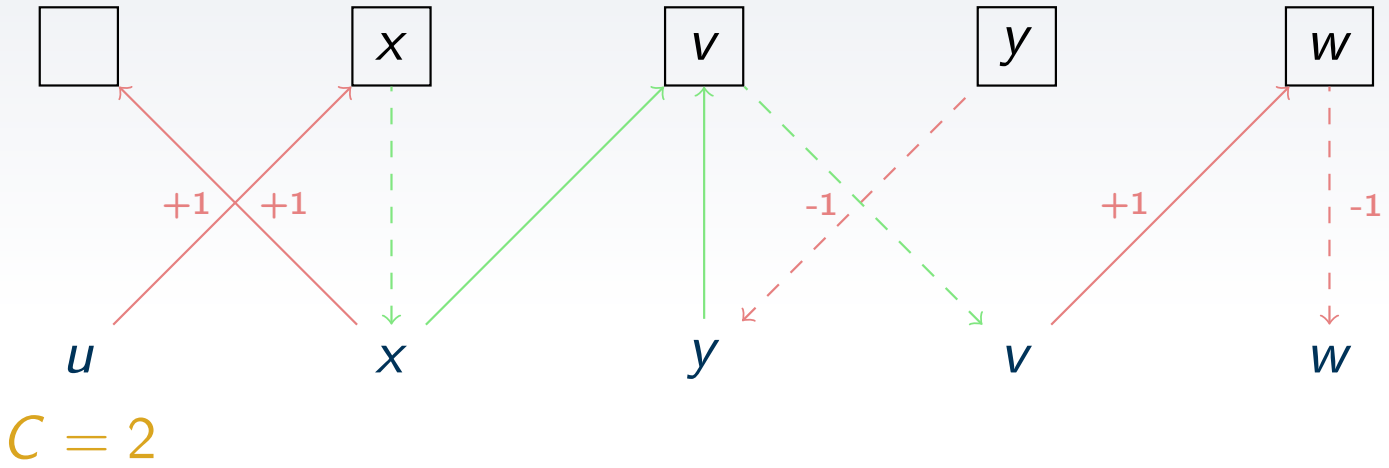


$C = 1$

Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path
  - augment matching using all paths with cost C
  - increment C by 1

Successive Shortest Path Algorithm

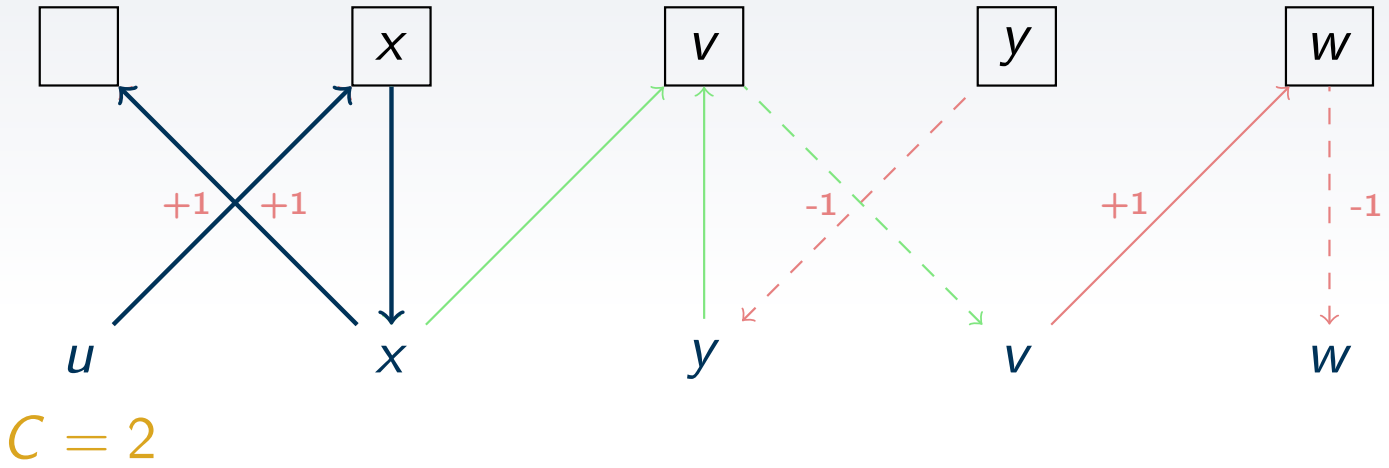


$C = 2$

Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path
  - augment matching using all paths with cost C
  - increment C by 1

Successive Shortest Path Algorithm

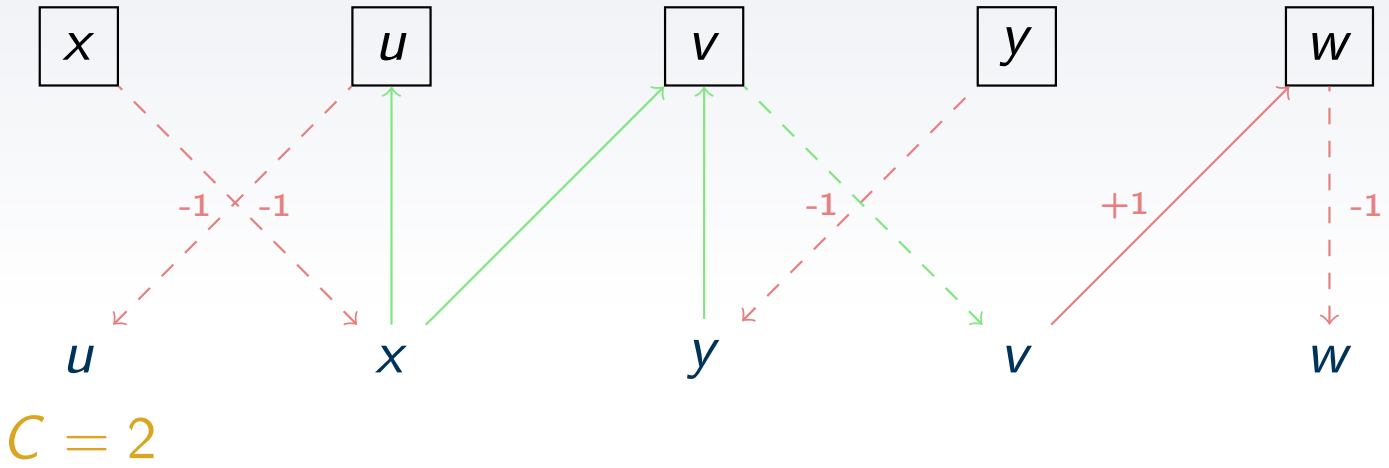


$C = 2$

Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path
  - augment matching using all paths with cost C
  - increment C by 1

Successive Shortest Path Algorithm



Variant of Hopcroft-Karp algorithm:

- primary edges have cost 0, backup edges have cost $+1$ or -1
- search for augmenting path of fixed costs C, start with $C = 0$
- as long as there is any augmenting path
  - augment matching using all paths with cost C
  - increment C by 1

Properties of Random Walk (1)

Failure rate among 10^6 attempts:

max avg. #steps:		30	25
page size:		10^2	10^3
bias	load		
0.97	0.95		
0.90	0.97		

Properties of Random Walk (1)

Failure rate among 10^6 attempts:

max avg. #steps:		30	25
page size:		10^2	10^3
bias	load		
0.97	0.95	0	0
0.90	0.97	0	0

Properties of Random Walk (1)

Failure rate among 10^6 attempts:

max avg. #steps:		30	25
page size:		10^2	10^3
bias	load		
0.97	0.95	0	0
0.90	0.97	0	0

$\Rightarrow$ **failure probability is** $< 10^{-5}$ with level of significance $> 1 - e^{-10}$

Properties of Random Walk (1)

Failure rate among 10^6 attempts:

max avg. #steps:		30	25
page size:		10^2	10^3
bias	load		
0.97	0.95	0	0
0.90	0.97	0	0

$\Rightarrow$ **failure probability is** $< 10^{-5}$ with level of significance $> 1 - e^{-10}$

$\Rightarrow$ ignore max avg. #steps for these configurations

Properties of Random Walk (2)

Average number of steps and page requests:

page size:		10^2		10^3	
bias	load				
0.97	0.95	(	)	(	)
0.90	0.97	(	)	(	)

Properties of Random Walk (2)

Average number of steps and page requests:

page size:		10^2	10^3
bias	load		
0.97	0.95	(22.81, 2.25)	( )
0.90	0.97	( )	( )

Properties of Random Walk (2)

Average number of steps and page requests:

page size:		10^2	10^3
bias	load		
0.97	0.95	(22.81, 2.25)	(16.60, 1.89)
0.90	0.97	( )	( )

- decreasing for growing page size

Properties of Random Walk (2)

Average number of steps and page requests:

page size:		10^2	10^3
bias	load		
0.97	0.95	(22.81, 2.25)	(16.60, 1.89)
0.90	0.97	(23.29,)	(19.49,)

- decreasing for growing page size
- **moderate number of steps**

Properties of Random Walk (2)

Average number of steps and page requests:

page size:		10^2	10^3
bias	load		
0.97	0.95	(22.81, 2.25)	(16.60, 1.89)
0.90	0.97	(23.29, 5.36)	(19.49, 4.61)

- decreasing for growing page size
- **moderate number of steps**
- number of page requests high near maximum load threshold