Cache-aware suffix trees [Clark and Munro]

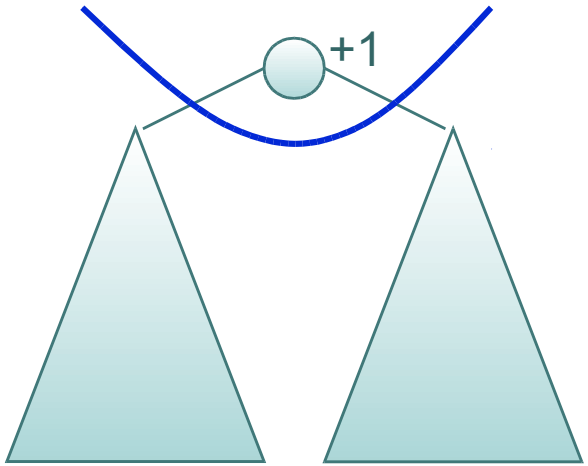
Greedy paging of nodes

Bottom up greedy paging
(pd = page depth, binary trees)

- pd(leaf) = 1

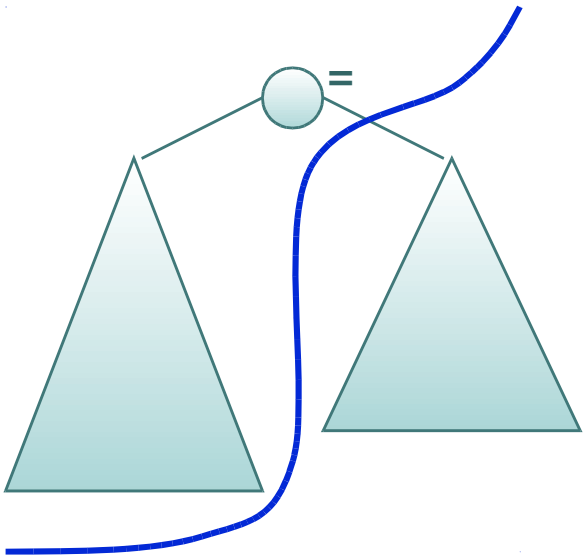
Greedy paging of nodes

- Same pd in the children (balanced case)
  - Children+parent fit in B $\Rightarrow$ keep pd
  - Else, pd(parent) = pd(children)+1



Greedy paging of nodes

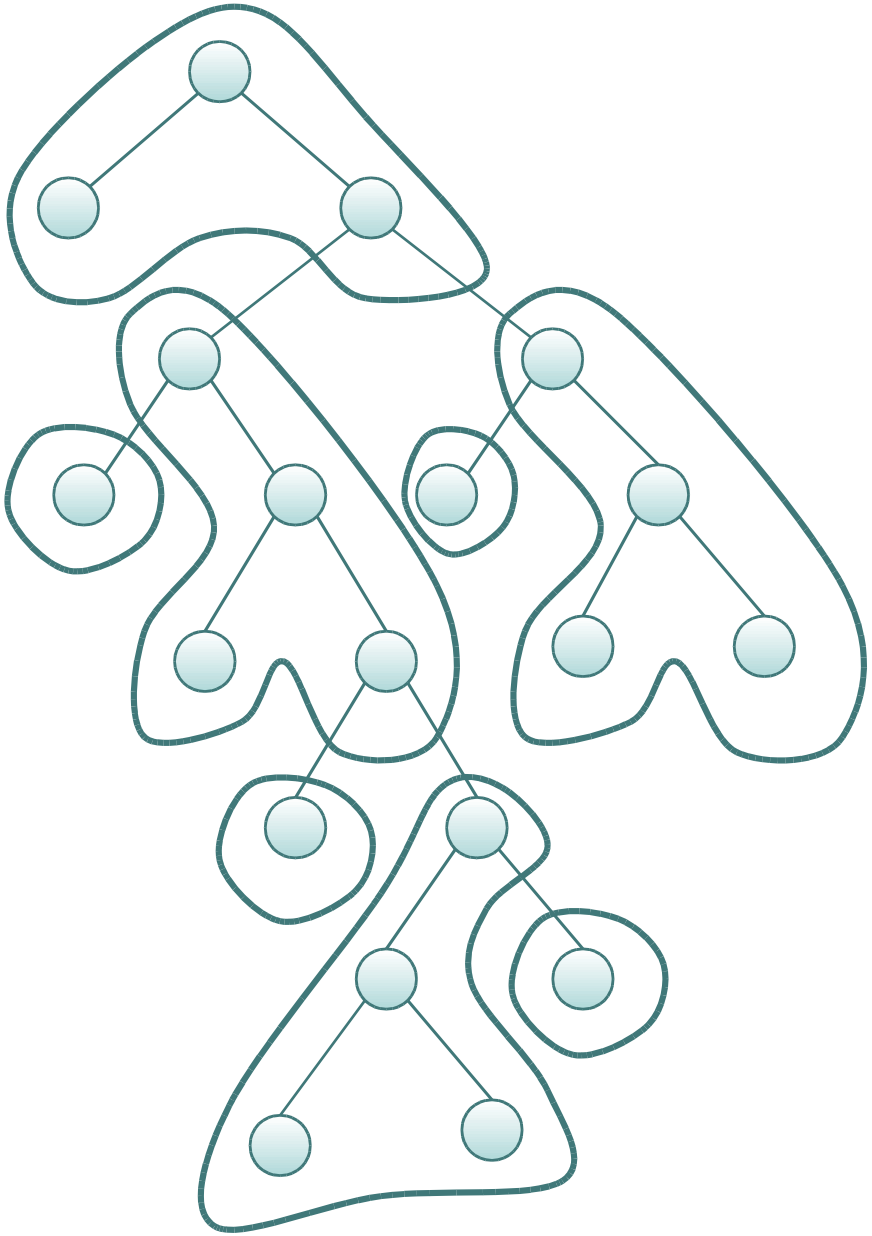
- Different pd in the children (unbalanced)
  - Child+parent fit in B $\Rightarrow$ retain largest pd



  - Else, pd(parent) = largest pd(children)+1

Example

B = 4



Bounds of CA suffix trees

- Nearly optimal pattern search cost = **$O(P/\sqrt{B} + \lg_B n)$** block transfers

- Space **O(N/B)** blocks:
  pack multiple logical blocks into the same physical block

- Some technicalities to insert/delete nodes

Cache-oblivious suffix trees

General scheme for static trees

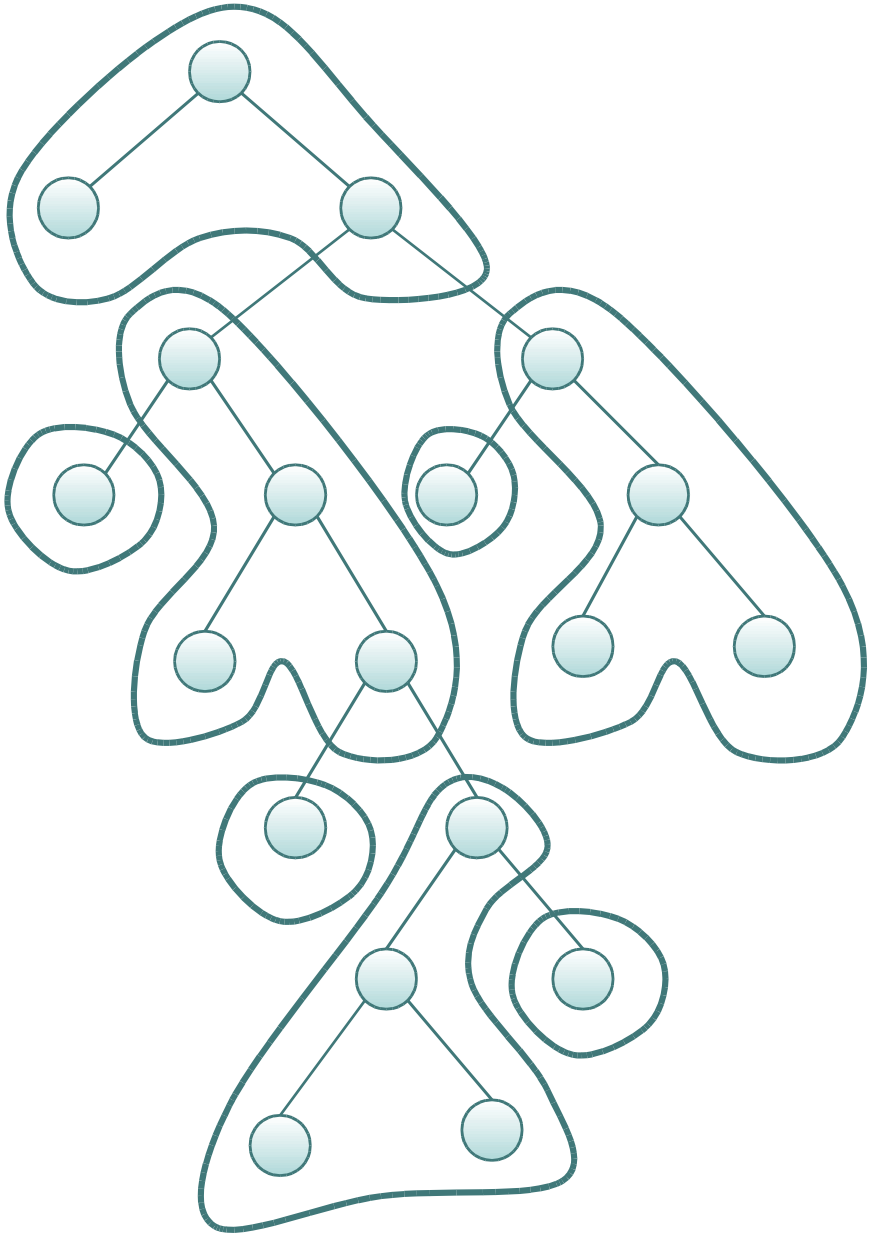
[Altstrup,Bender,Demaine,Farach-Colton,Munro, Rauhe,Thorup]

- In our case: apply Clark-Munro with block size B = n, n/2, n/4, …, O(1)
- Caution: paging blocks must be nested!

- Assign same integer id to nodes belonging to the same page, for any chosen B

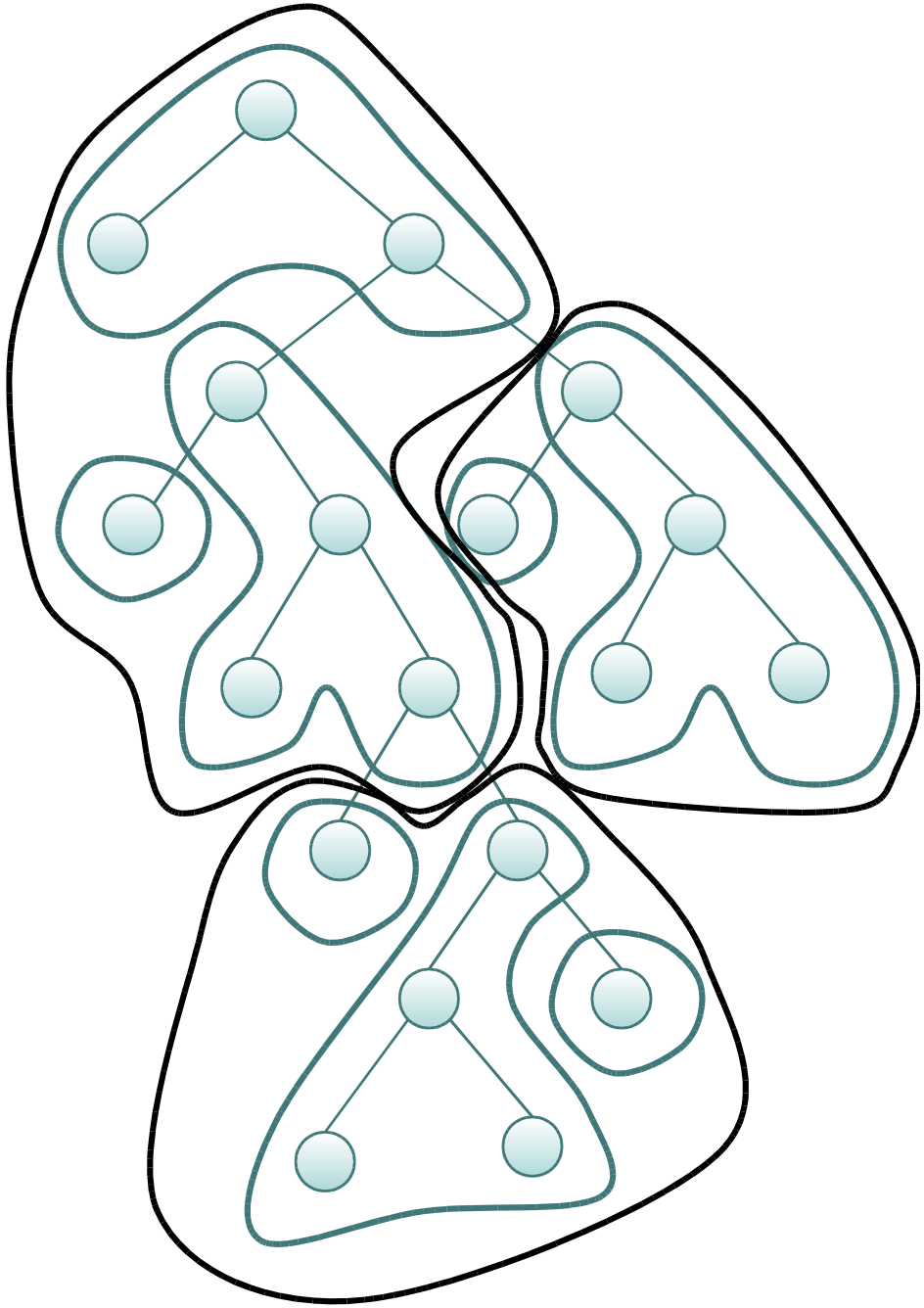
Example

B = 4



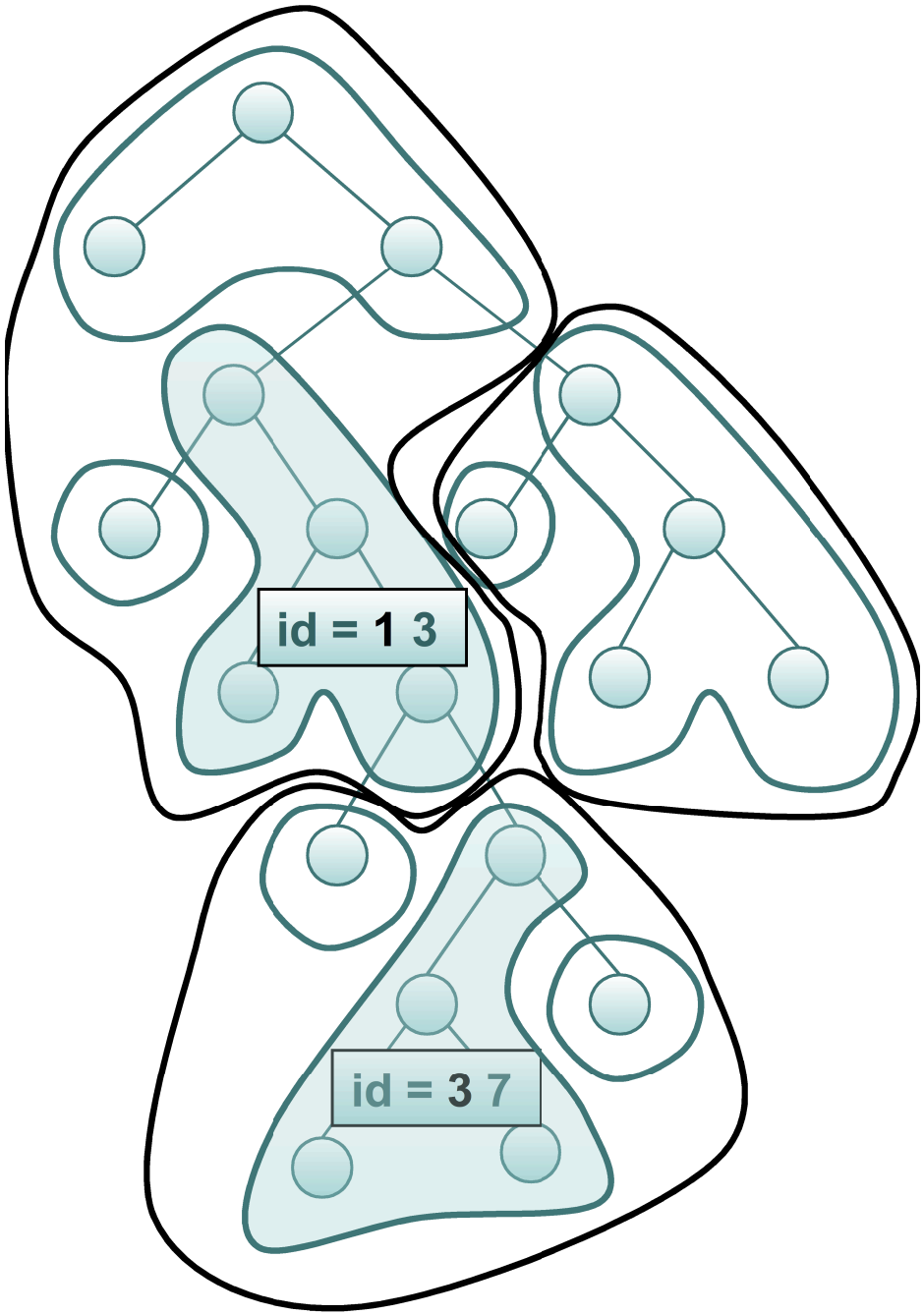
Example

B = 8,4



Example

B = 8,4



Cache-oblivious suffix trees

- Assign each node a **signature** of the resulting **lg n** integer ids
- Sort lexicographically the nodes by their signatures
- Store them into an array in that order $\Rightarrow$
  any (unknown) B achieves

  cache-oblivious cost < 2 x cache-aware cost