Mémoire et volume d'entrées sorties pour les méthodes hors mémoire de factorisation de matrices creuses : même combat?

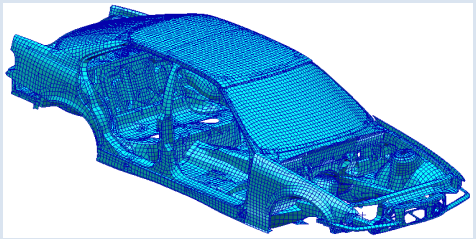
A. Guermouche

Labri / INRIA Futurs

Outline

Out-of-core

Solving sparse linear systems

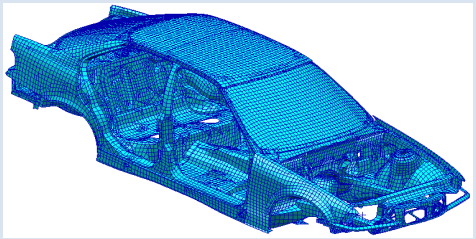


$Ax = b$: 1 M variables

$\Rightarrow A = LU$ (Direct methods)

Out-of-core

Solving sparse linear systems



$Ax = b$: 1 M variables

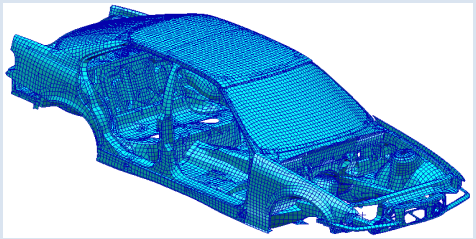
$\Rightarrow A = LU$ (Direct methods)

Current limits: BRGM matrix

- 3.7×10^6 variables
- 156×10^6 non zeros in A
- 4.5×10^9 non zeros in LU
- 26.5×10^{12} flops

Out-of-core

Solving sparse linear systems



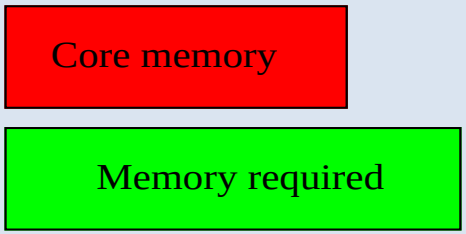
$Ax = b$: 1 M variables

$\Rightarrow A = LU$ (Direct methods)

Current limits: BRGM matrix

- 3.7×10^6 variables
- 156×10^6 non zeros in A
- 4.5×10^9 non zeros in LU
- 26.5×10^{12} flops

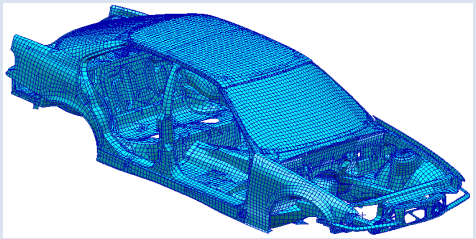
Out-of-core



Memory crash

Out-of-core

Solving sparse linear systems



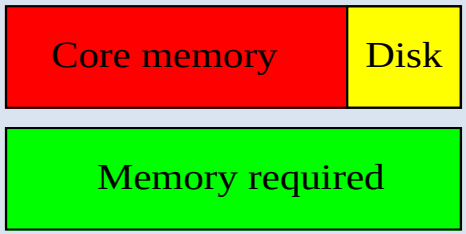
$Ax = b$: 1 M variables

$\Rightarrow A = LU$ (Direct methods)

Current limits: BRGM matrix

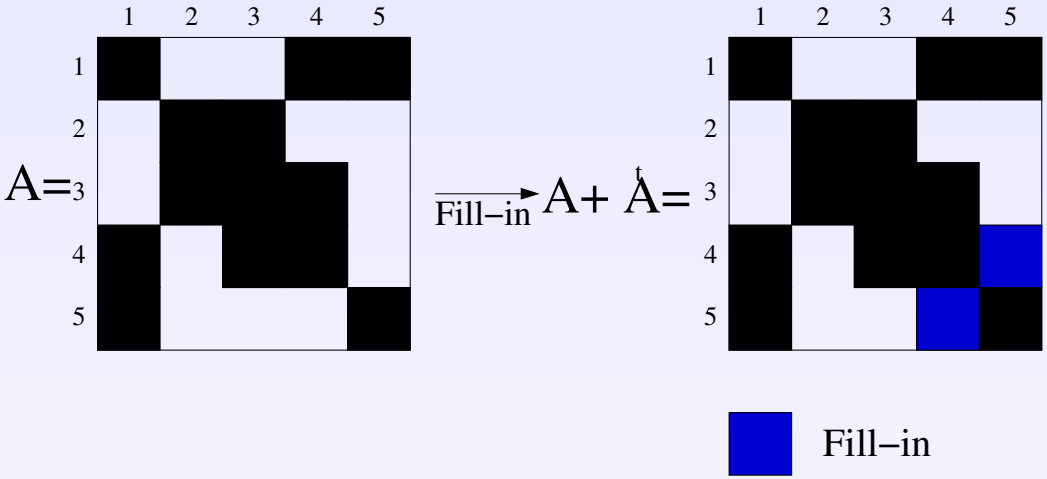
- 3.7×10^6 variables
- 156×10^6 non zeros in A
- 4.5×10^9 non zeros in LU
- 26.5×10^{12} flops

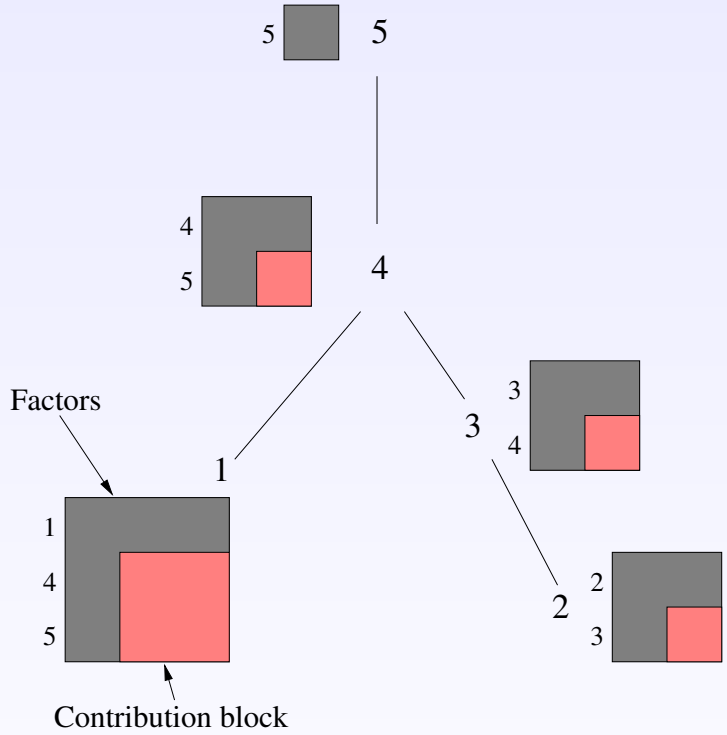
Physical constraint



Use of disks

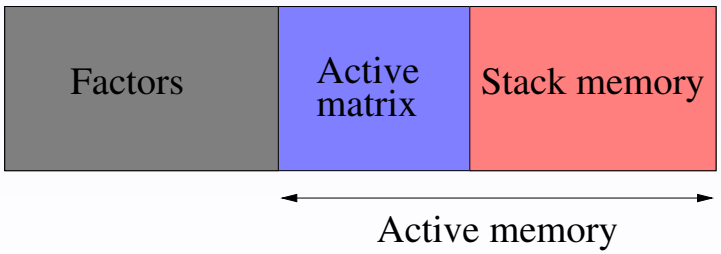
The multifrontal method (Duff, Reid'83)





Memory divided into two parts:

- Active memory
- Factors



Dependency tree

Outline

Sequential case: Memory behavior (1/2)

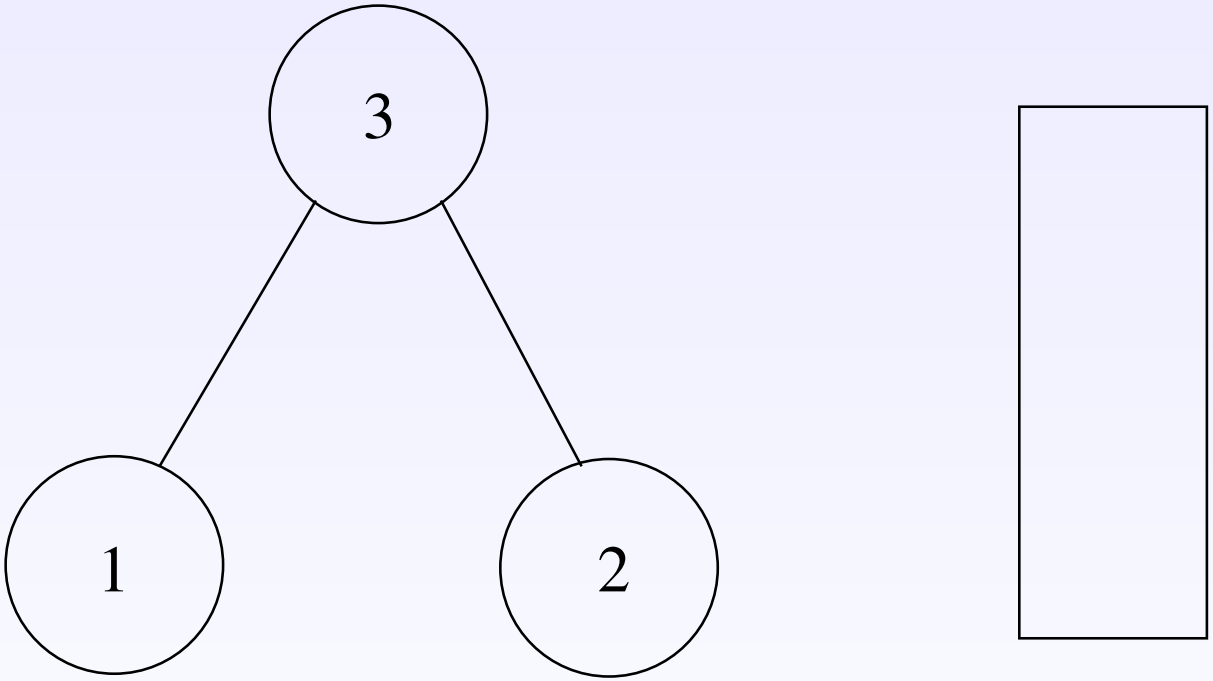


Figure: Example illustrating the memory behavior.

Sequential case: Memory behavior (1/2)

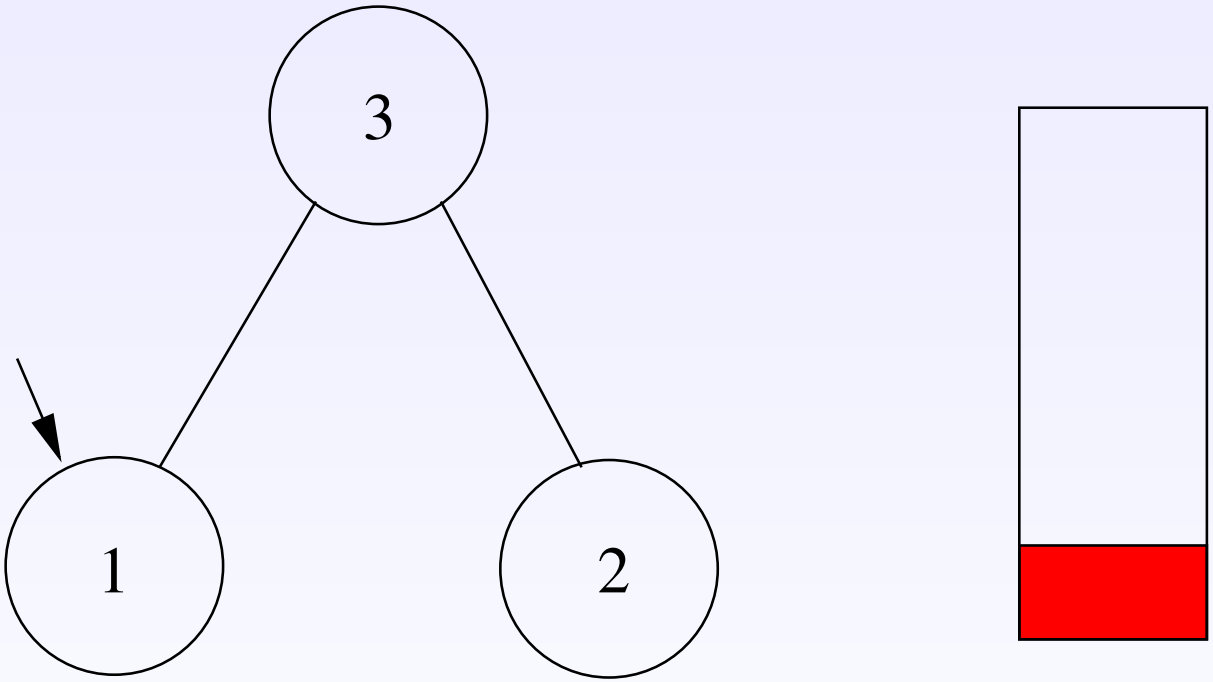


Figure: Example illustrating the memory behavior.

Sequential case: Memory behavior (1/2)

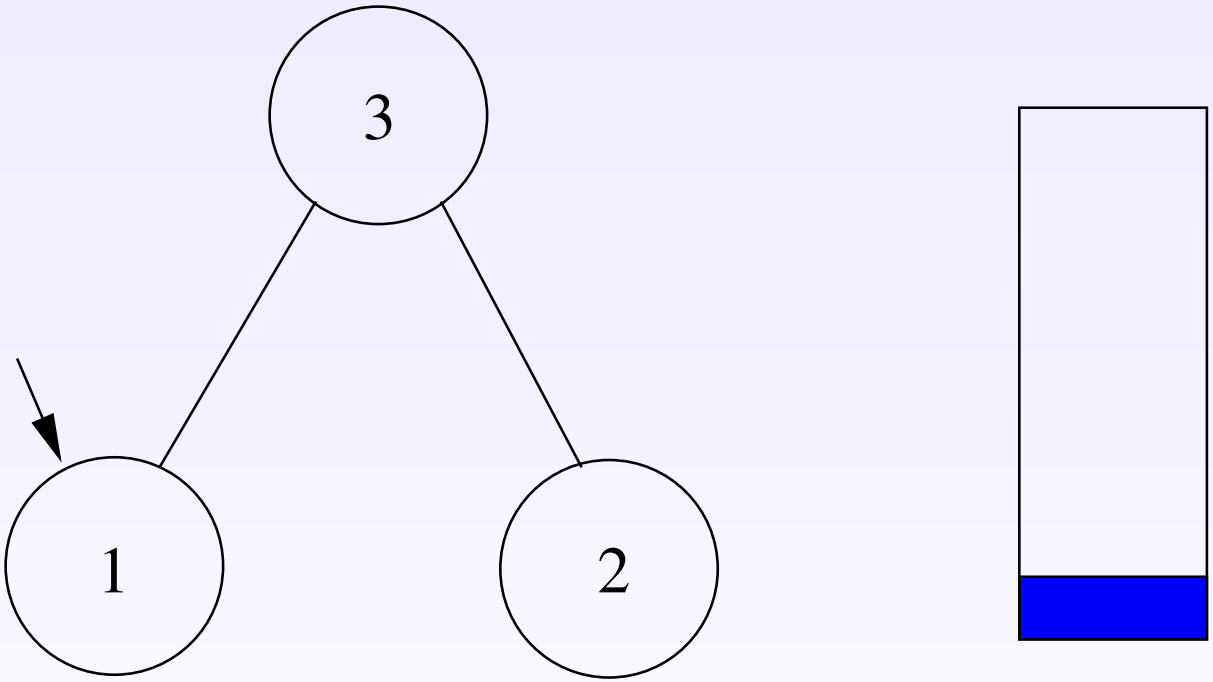


Figure: Example illustrating the memory behavior.

Sequential case: Memory behavior (1/2)

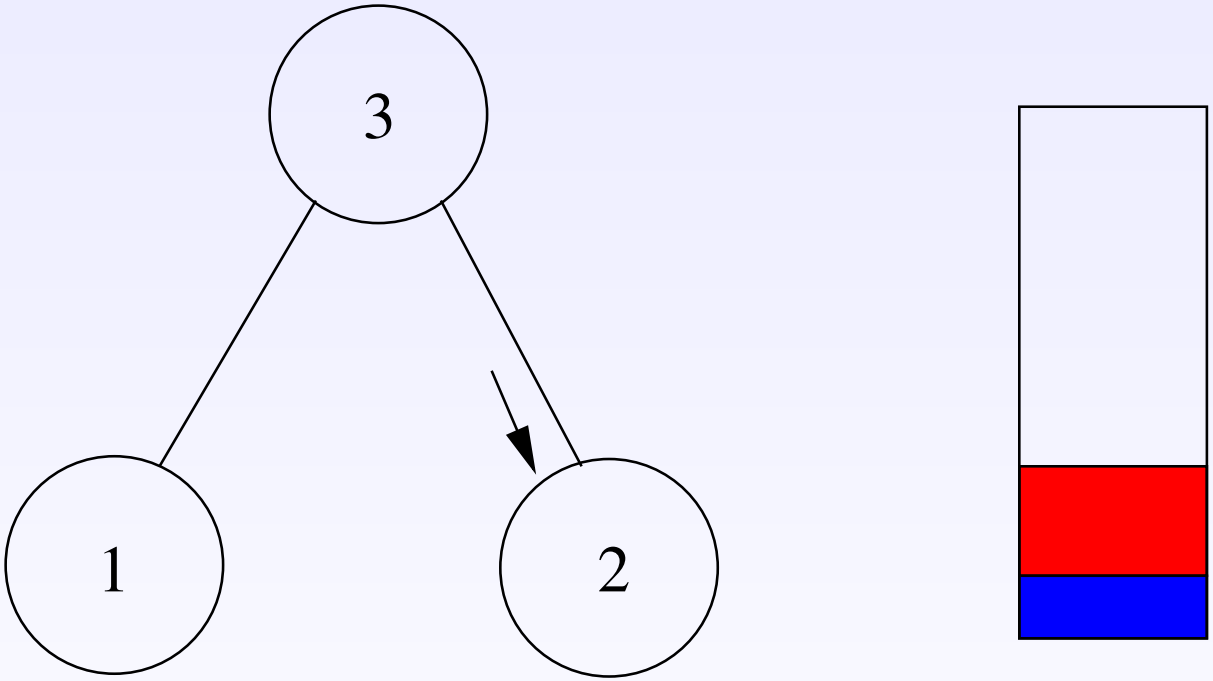


Figure: Example illustrating the memory behavior.

Sequential case: Memory behavior (1/2)

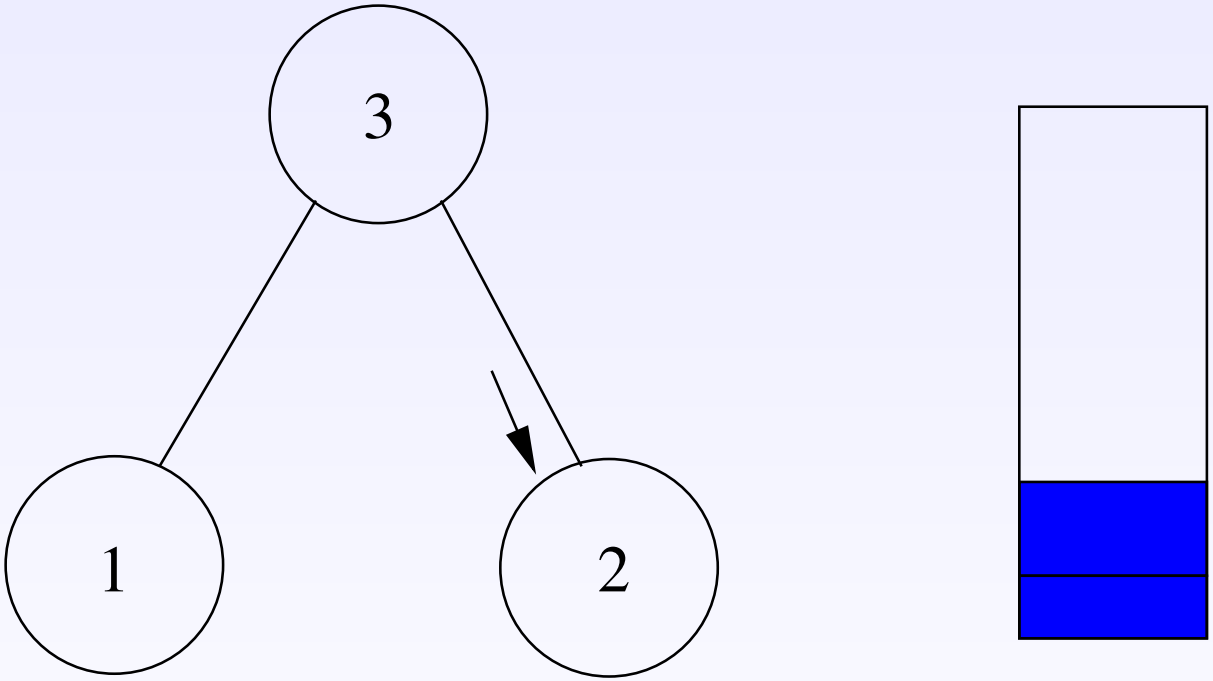


Figure: Example illustrating the memory behavior.

Sequential case: Memory behavior (1/2)

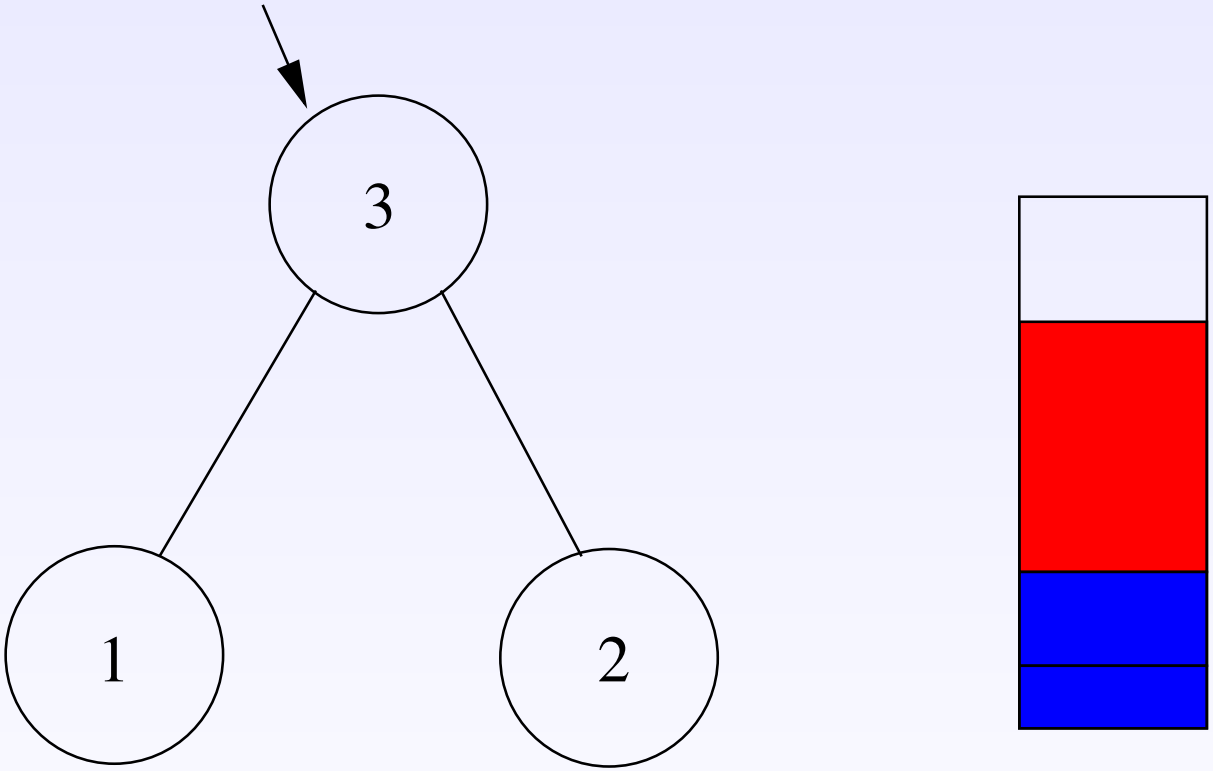


Figure: Example illustrating the memory behavior.

Sequential case: Memory behavior (1/2)

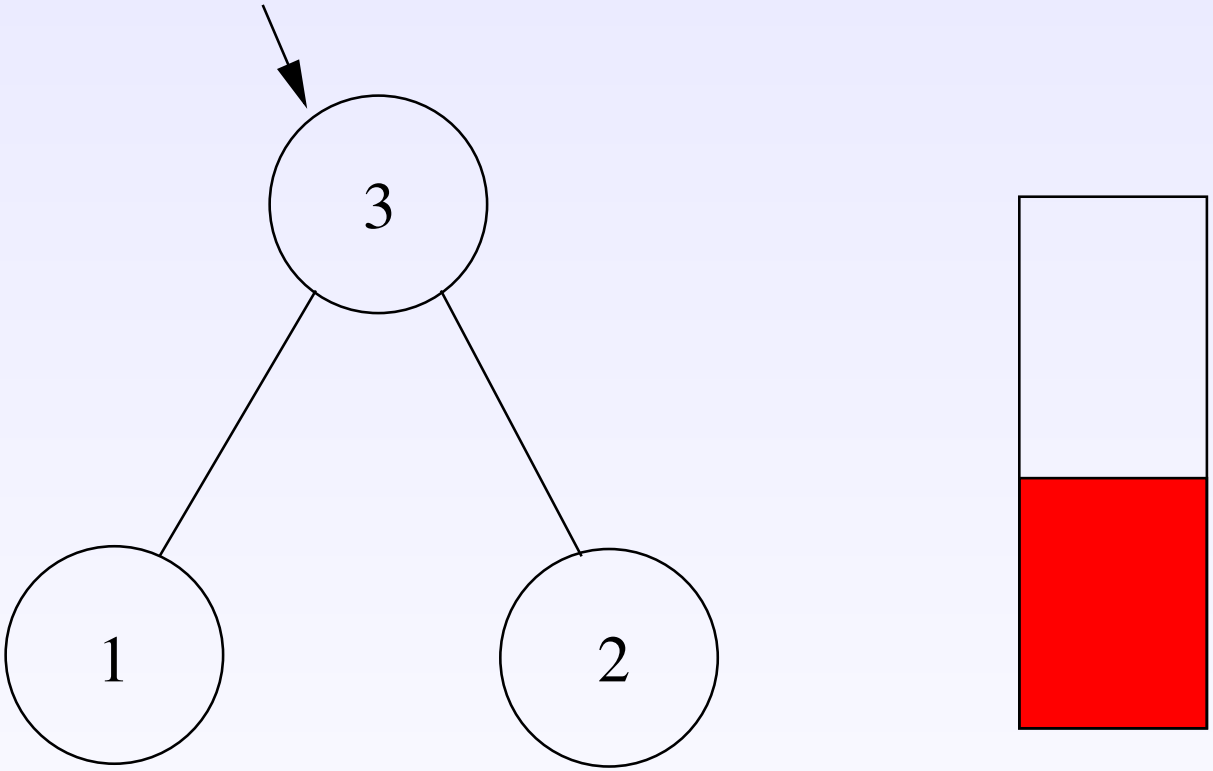


Figure: Example illustrating the memory behavior.

Sequential case: Memory behavior (1/2)

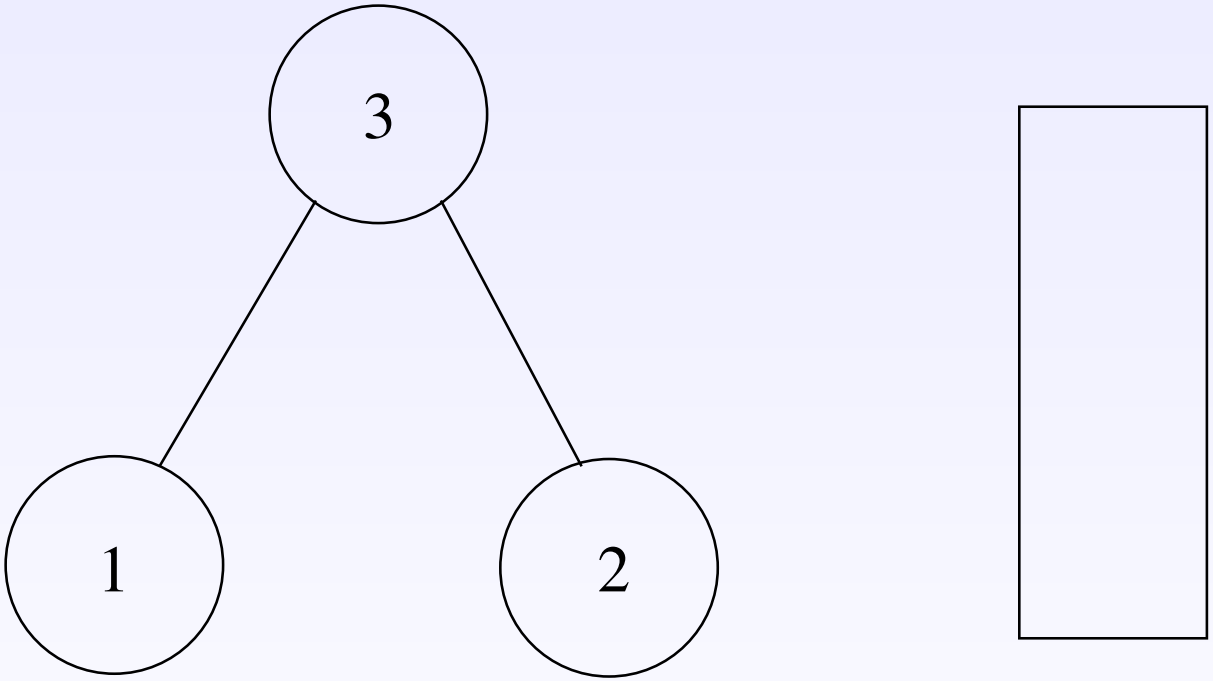
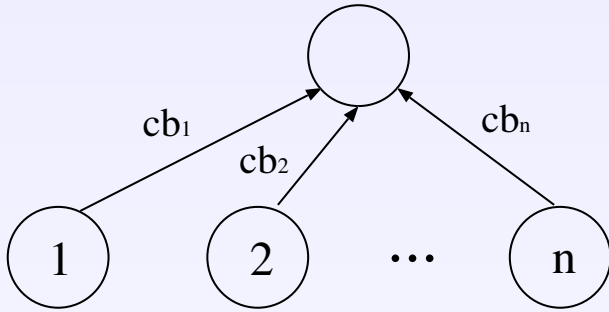


Figure: Example illustrating the memory behavior.

Sequential case: Memory behavior (2/2)

Consider a parent node in the tree:

- n is the number of children.
- j denotes the j^{th} child of the node.
- cb_j is the size of the contribution block of child j.
- m is the memory size of the frontal matrix of the parent.
- A (resp. A_j) is the amount of active memory needed to process the parent (resp. child j).



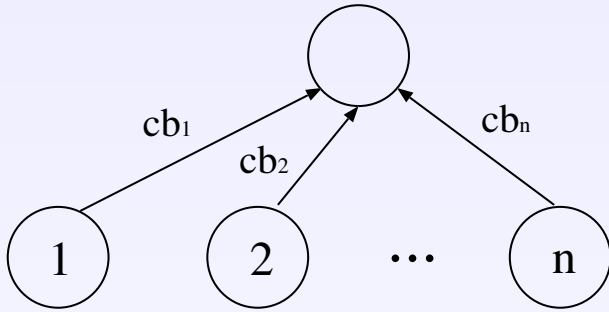
The assembly step requires a storage:

$$m + \sum_{j=1}^{n} cb_j$$

Sequential case: Memory behavior (2/2)

Consider a parent node in the tree:

- n is the number of children.
- j denotes the j^{th} child of the node.
- cb_j is the size of the contribution block of child j.
- m is the memory size of the frontal matrix of the parent.
- A (resp. A_j) is the amount of active memory needed to process the parent (resp. child j).



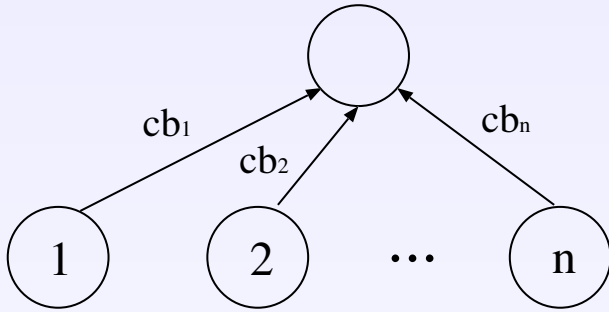
The storage required to process child j is:

$$A_j + \sum_{k=1}^{j-1} cb_k$$

Sequential case: Memory behavior (2/2)

Consider a parent node in the tree:

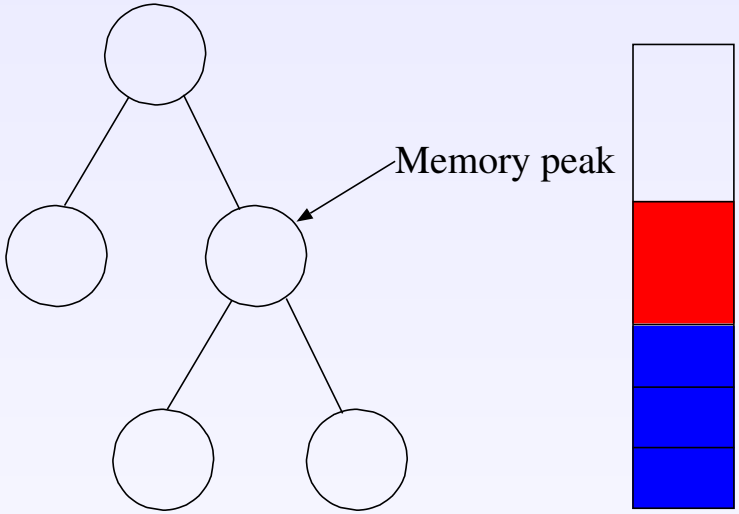
- n is the number of children.
- j denotes the j^{th} child of the node.
- cb_j is the size of the contribution block of child j.
- m is the memory size of the frontal matrix of the parent.
- A (resp. A_j) is the amount of active memory needed to process the parent (resp. child j).



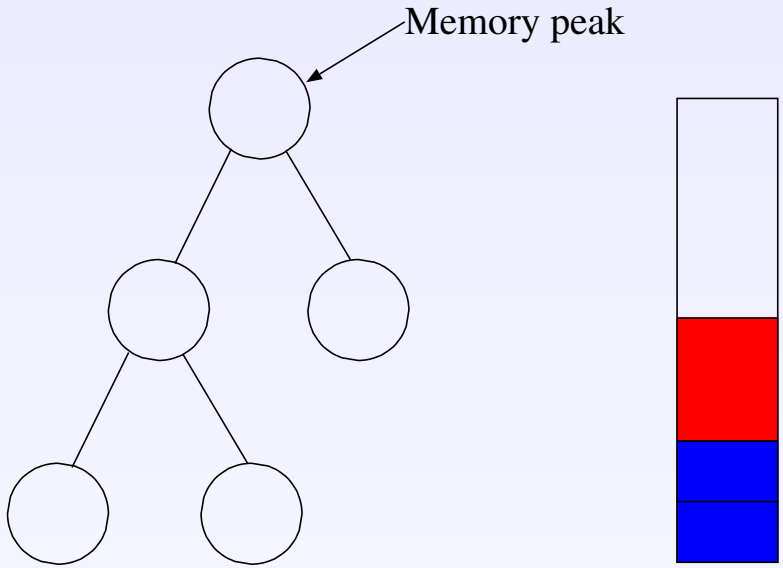
A is thus defined by:

$$A = \max(\max_{j=1,n}(A_j + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{n} cb_j)$$

Impact of the tree traversal



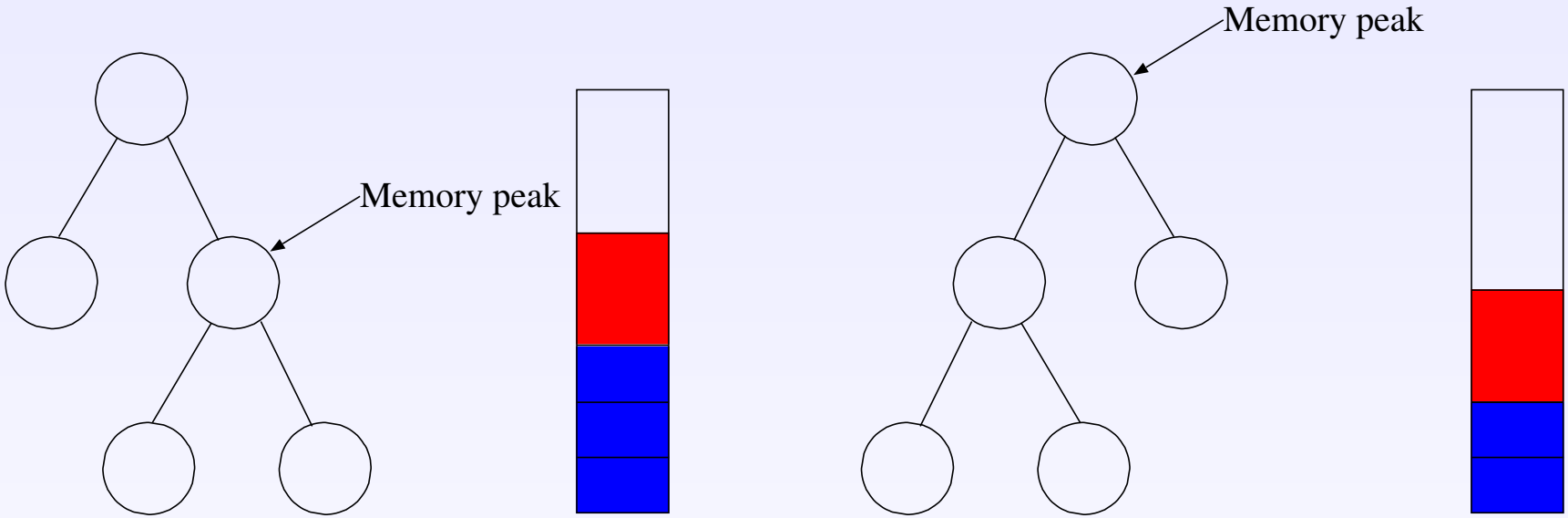
Worst case.



Best case.

Figure: Impact of the tree traversal on the memory behavior.

Impact of the tree traversal



Worst case. Best case.

Figure: Impact of the tree traversal on the memory behavior.

→ GOAL: Find the best tree traversal in terms of memory occupation

Outline

Remarks and properties

Liu's Theorem (Tree pebbling theorem)

The minimum of $\max_j(x_j + \sum_{i=1}^{j-1} y_j)$ *is obtained when the sequence* (x_i, y_i) *is sorted in decreasing order of* $x_i - y_i$,

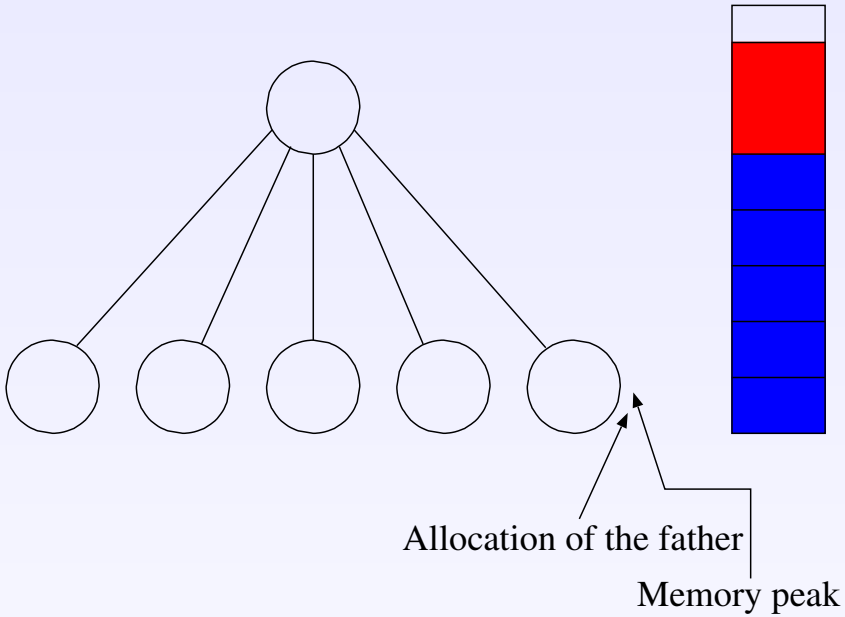
Consequence:

An optimal child sequence is obtained by rearranging the children nodes in decreasing order of $A_i - cb_i$.

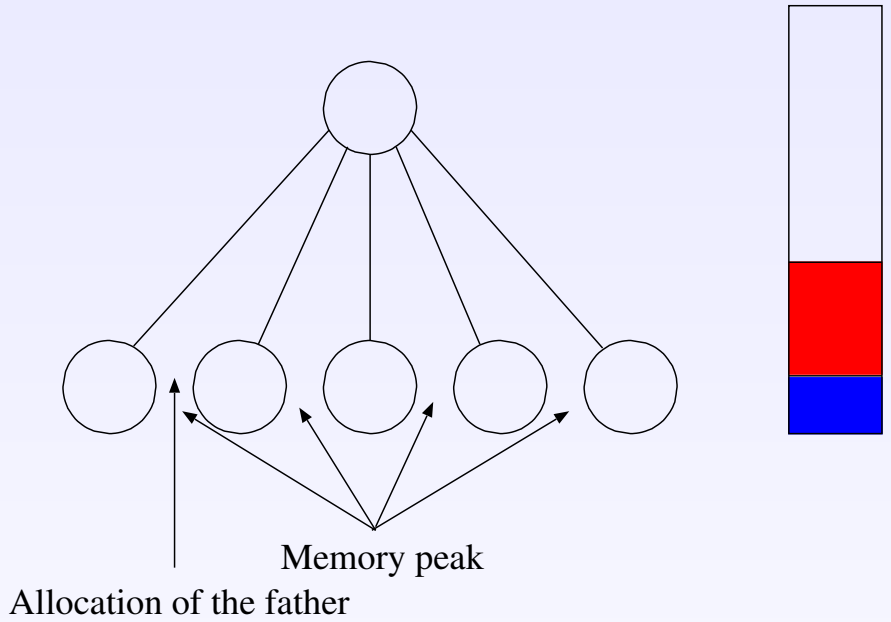
Algorithm:

- ⋆ Bottom-up greedy process.
- ⋆ Apply Liu's theorem at each level of the tree.

Limitation of the Classical scheme

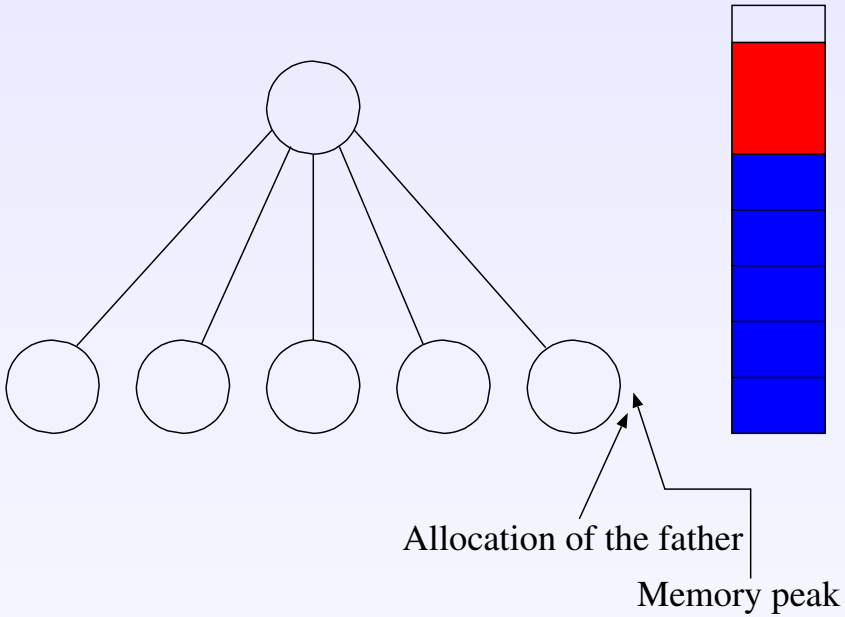


Classical approach.

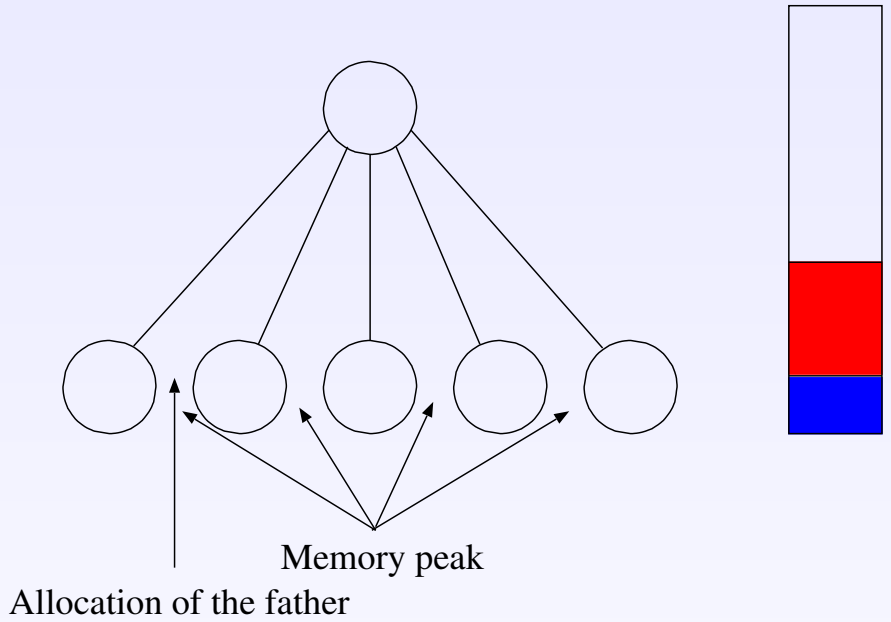


Flexible scheme.

Limitation of the Classical scheme



Classical approach.

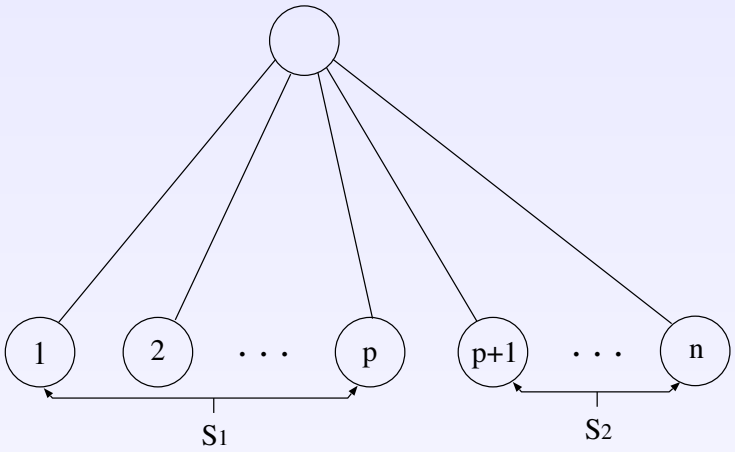


Flexible scheme.

$\rightarrow$ Decoupling the allocation and the computations can improve the memory behavior

Outline

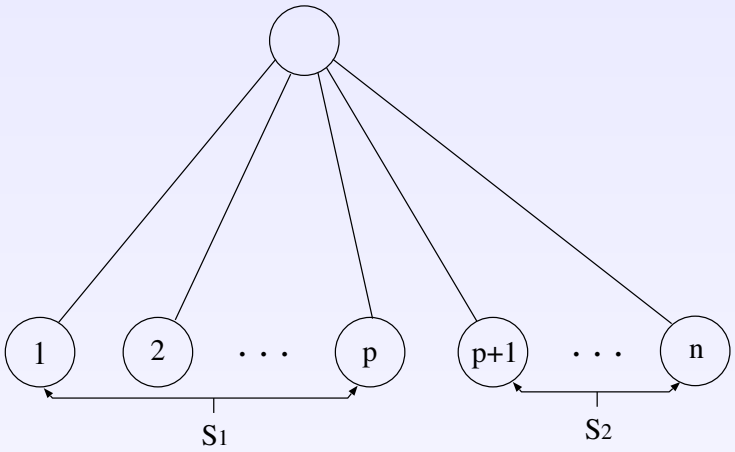
Flexible multifrontal scheme



- p is the position of the allocation of the parent.
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent.
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent.

- The memory behavior inside $\mathcal{S}_1$ is similar to the case of the classical multifrontal scheme.
- Inside $\mathcal{S}_2$, the order of the children has no impact on the memory behavior.

Flexible multifrontal scheme

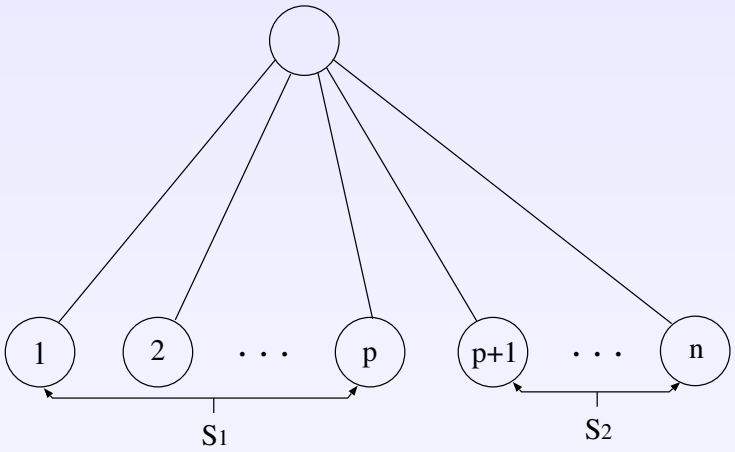


- p is the position of the allocation of the parent.
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent.
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent.

- The memory behavior inside $\mathcal{S}_1$ is similar to the case of the classical multifrontal scheme.
- Inside $\mathcal{S}_2$, the order of the children has no impact on the memory behavior.

$$A^{flex} = \max\left(\max_{j=1,p}(A_j^{flex} + \sum_{k=1}^{j-1} cb_k), m + \sum_{k=1}^{p} cb_k, m + \max_{j=p+1,n} A_j^{flex}\right)$$

Flexible multifrontal scheme



- p is the position of the allocation of the parent.
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent.
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent.

- The memory behavior inside $\mathcal{S}_1$ is similar to the case of the classical multifrontal scheme.
- Inside $\mathcal{S}_2$, the order of the children has no impact on the memory behavior.

$$A^{flex} = \max\left(\max_{j=1,p}(A_j^{flex} + \sum_{k=1}^{j-1} cb_k), m + \sum_{k=1}^{p} cb_k, m + \max_{j=p+1,n} A_j^{flex}\right)$$

A new memory minimization algorithm

Theorem

An optimal sequence can be obtained by :

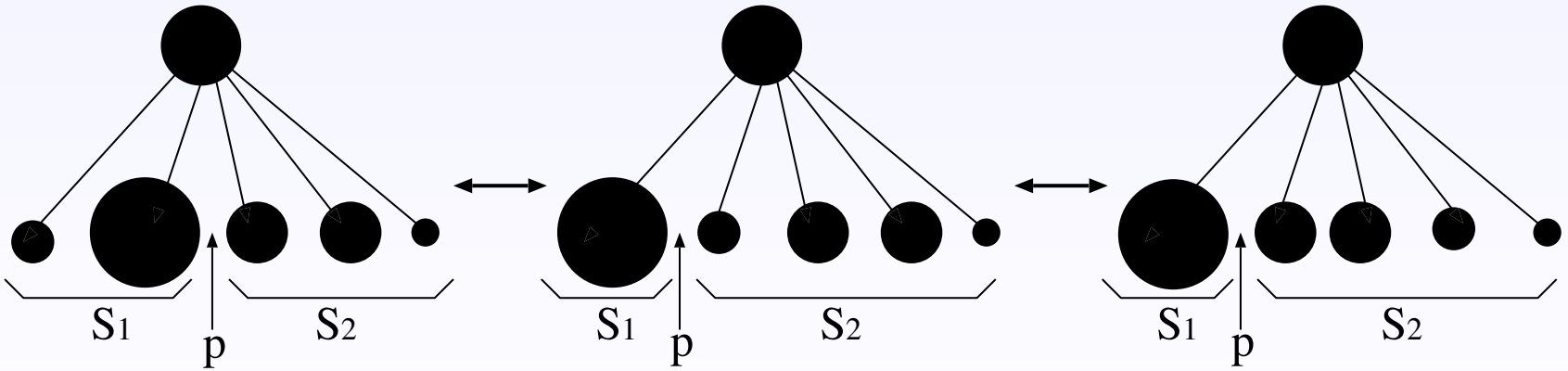
- *Sorting the children in decreasing order of* A_j^{flex}.
- *Trying all the possible positions for the allocation of the parent and sorting the children belonging to* $\mathcal{S}_1$ *according to Liu's Theorem.*
- *Selecting the configuration that gives the smallest peak.*

Algorithm:
Bottom-up greedy process where the theorem is applied at each level of the tree.

Proof

$$A^{flex} = \max\left(\max_{j=1,p}(A_j^{flex} + \sum_{k=1}^{j-1} cb_k), m + \sum_{k=1}^{p} cb_k, m + \max_{j=p+1,n} A_j^{flex}\right)$$

- Inside $\mathcal{S}_2$, the order of the children has no impact on the memory behavior.
- If $\exists j \in \mathcal{S}_1$ / $A_j^{flex} \leq \max_{i \in \mathcal{S}_2}(A_i^{flex}) \rightarrow j$ can be moved from $\mathcal{S}_1$ to $\mathcal{S}_2$ without increasing the peak.



Optimal configuration

Active memory minimization Algorithm

Algorithm:

Set $\mathcal{S}_1 = \{1, \ldots, n\}$, $\mathcal{S}_2 = \emptyset$ and $p = n$;
Find the schedule providing an optimal A^{flex} value for partition $(\mathcal{S}_1, \mathcal{S}_2)$;
repeat
  Find j such that $A_j^{flex} = \min_{k \in \mathcal{S}_1} A_k^{flex}$;
  Set $\mathcal{S}_1 = \mathcal{S}_1 \setminus \{j\}$, $\mathcal{S}_2 = \mathcal{S}_2 \cup \{j\}$, and $p = p - 1$;
  Find the schedule providing an optimal A'^{flex} value for partition $(\mathcal{S}_1, \mathcal{S}_2)$;
  if $A'^{flex} \leq A^{flex}$ **then**
    Keep the value of p, and the schedule of children in $\mathcal{S}_1$ and $\mathcal{S}_2$ corresponding to A'^{flex};
    Set $A^{flex} = A'^{flex}$;
  end if
until $p == 1$ or $A'^{flex} > A^{flex}$

Outline

Experimental environment

MUMPS: Multifrontal Parallel Solver for both LU and LDL^T.
Reordering techniques: *AMD*, *AMF*, *METIS*, *PORD*.
Test platform: *IBM* platform at *IDRIS*.
Test problems: Large range of matrices extracted from various collections (Rutherford-Boeing, University of Florida or PARASOL,...).
Schedules tested :

- Classical multifrontal scheme (parent allocated after all its children).
- Anticipated parent allocation scheme (parent allocated after its first child).
- Flexible parent allocation scheme (parent allocated at an arbitrary position).

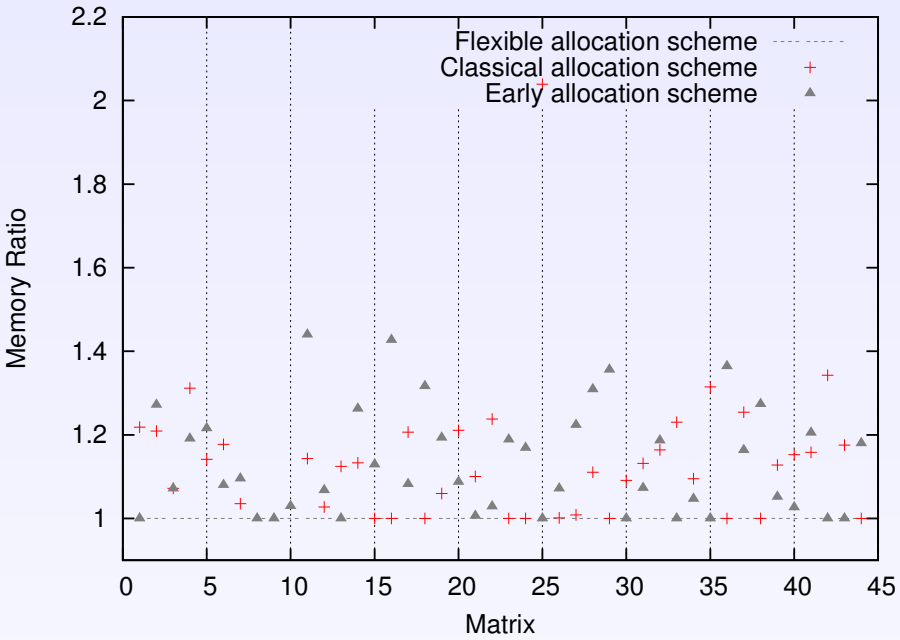
Experimental environment

MUMPS: Multifrontal Parallel Solver for both LU and LDL^T.
Reordering techniques: *AMD*, *AMF*, *METIS*, *PORD*.
Test platform: *IBM* platform at *IDRIS*.
Test problems: Large range of matrices extracted from various collections (Rutherford-Boeing, University of Florida or PARASOL,...).
Schedules tested :

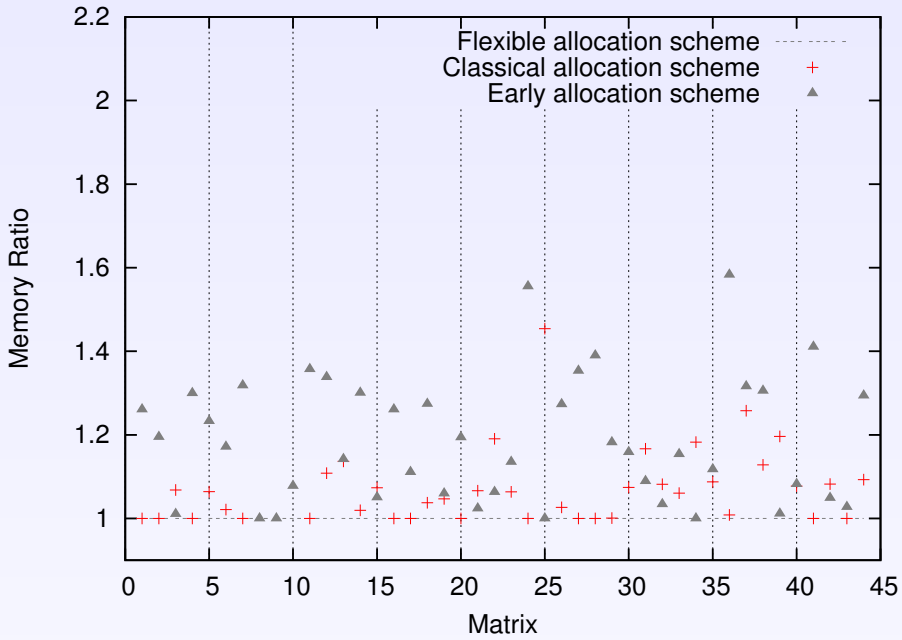
- Classical multifrontal scheme (parent allocated after all its children).
- Anticipated parent allocation scheme (parent allocated after its first child).
- Flexible parent allocation scheme (parent allocated at an arbitrary position).

Simulation of memory variations for all the schedules during the analysis step.

Experimental results



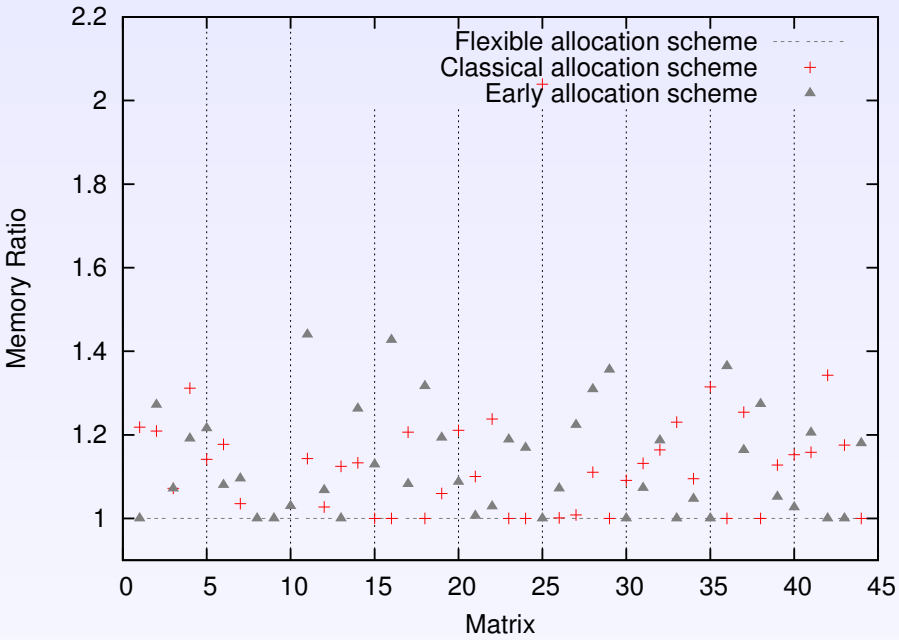
AMD.



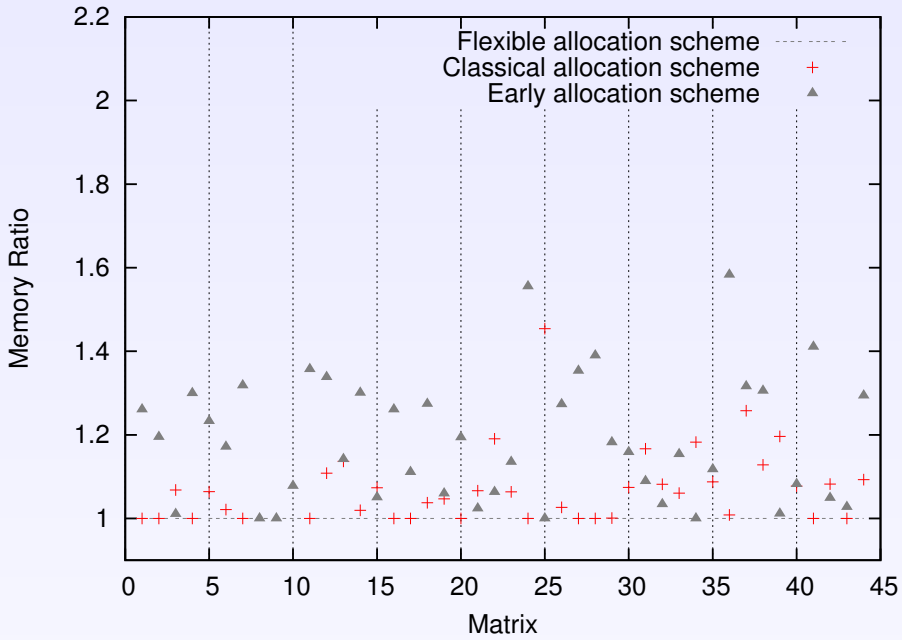
METIS.

Figure: Active memory ratios.

Experimental results



AMD.



METIS.

Figure: Active memory ratios.

Large gains against the *classical allocation scheme* for matrices 8, 9 and 10.

Outline

Total memory minimization (1/3)

Memory space T^{flex} needed for the processing of a node in the tree is given by:

Total memory minimization (1/3)

Memory space T^{flex} needed for the processing of a node in the tree is given by:

$$\mathcal{P}_1 = \max\Big(\max_{j=1,p}(T_j^{flex} + \sum_{k=1}^{j-1} (cb_k + F_k)), \; m + \sum_{k=1}^{p} (cb_k + F_k) \Big)$$

Total memory minimization (1/3)

Memory space T^{flex} needed for the processing of a node in the tree is given by:

$$\mathcal{P}_1 = \max\Big(\max_{j=1,p}(T_j^{flex} + \sum_{k=1}^{j-1} (cb_k + F_k)), \; m + \sum_{k=1}^{p} (cb_k + F_k) \Big)$$

$$\mathcal{P}_2 = \max\Big(m + \sum_{k=1}^{p} F_k + \max_{j=p+1,n} (T_j^{flex} + \sum_{k=p+1}^{j-1} F_k) \Big)$$

Total memory minimization (1/3)

Memory space T^{flex} needed for the processing of a node in the tree is given by:

$$\mathcal{P}_1 = \max\Big(\max_{j=1,p}(T_j^{flex} + \sum_{k=1}^{j-1} (cb_k + F_k)), \; m + \sum_{k=1}^{p} (cb_k + F_k) \Big)$$

$$\mathcal{P}_2 = \max\Big(m + \sum_{k=1}^{p} F_k + \max_{j=p+1,n} (T_j^{flex} + \sum_{k=p+1}^{j-1} F_k) \Big)$$

$$T^{flex} = \max(\mathcal{P}_1, \mathcal{P}_2).$$

Total memory minimization (1/3)

Memory space T^{flex} needed for the processing of a node in the tree is given by:

$$\mathcal{P}_1 = \max\Big(\max_{j=1,p}(T_j^{flex} + \sum_{k=1}^{j-1}(cb_k + F_k)),$$
$$m + \sum_{k=1}^{p}(cb_k + F_k)\Big)$$

$$\mathcal{P}_2 = \max\Big(m + \sum_{k=1}^{p} F_k + \max_{j=p+1,n}(T_j^{flex} + \sum_{k=p+1}^{j-1} F_k)\Big)$$

$$T^{flex} = \max(\mathcal{P}_1, \mathcal{P}_2).$$

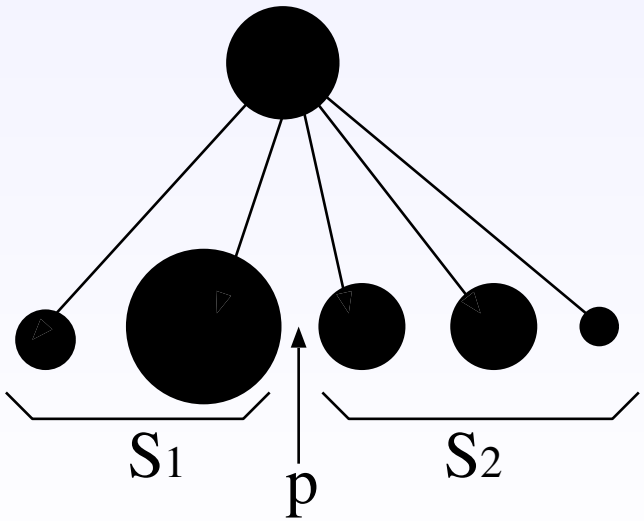
The order in $\mathcal{S}_2$ has an impact on the memory occupation.

Total memory minimization (2/3)

	$\mathcal{S}_1$	$\mathcal{S}_2$
Children sequence	$T_i^{flex} - (cb_i + F_i)$	$T_i^{flex} - F_i$

Total memory minimizing sequences inside $\mathcal{S}_1$ and $\mathcal{S}_2$.

Property:

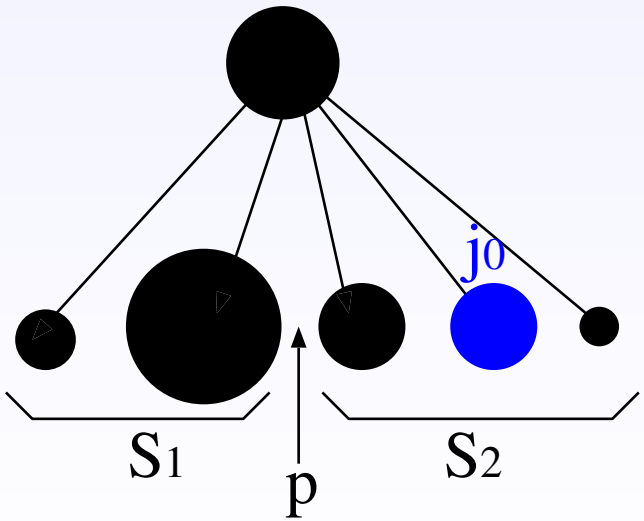


Total memory minimization (2/3)

	$\mathcal{S}_1$	$\mathcal{S}_2$
Children sequence	$T_i^{flex} - (cb_i + F_i)$	$T_i^{flex} - F_i$

Total memory minimizing sequences inside $\mathcal{S}_1$ and $\mathcal{S}_2$.

Property:



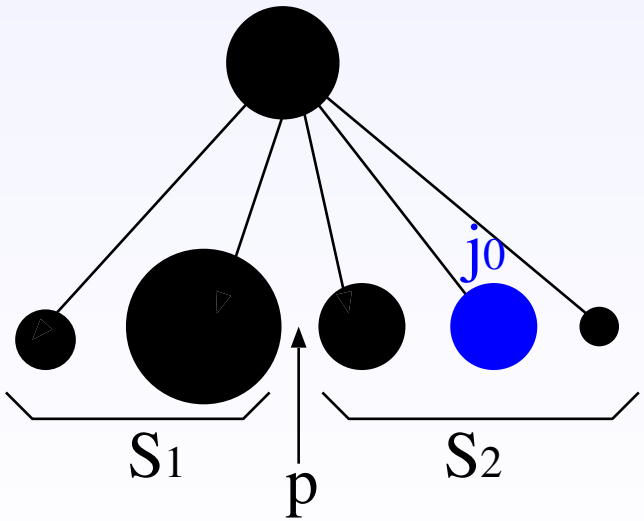
let $j_0 \in \mathcal{S}_2$ be the child for which the peak is reached inside $\mathcal{S}_2$.

Total memory minimization (2/3)

	$\mathcal{S}_1$	$\mathcal{S}_2$
Children sequence	$T_i^{flex} - (cb_i + F_i)$	$T_i^{flex} - F_i$

Total memory minimizing sequences inside $\mathcal{S}_1$ and $\mathcal{S}_2$.

Property:



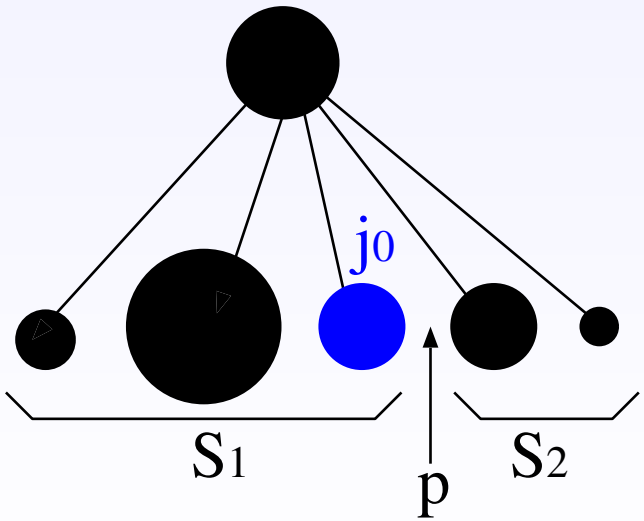
let $j_0 \in \mathcal{S}_2$ be the child for which the peak is reached inside $\mathcal{S}_2$. $\rightarrow$ The total memory peak cannot decrease if j_0 remains in $\mathcal{S}_2$ for all configurations where $\mathcal{S}_1 \subset \mathcal{S}'_1$.

Total memory minimization (2/3)

	$\mathcal{S}_1$	$\mathcal{S}_2$
Children sequence	$T_i^{flex} - (cb_i + F_i)$	$T_i^{flex} - F_i$

Total memory minimizing sequences inside $\mathcal{S}_1$ and $\mathcal{S}_2$.

Property:



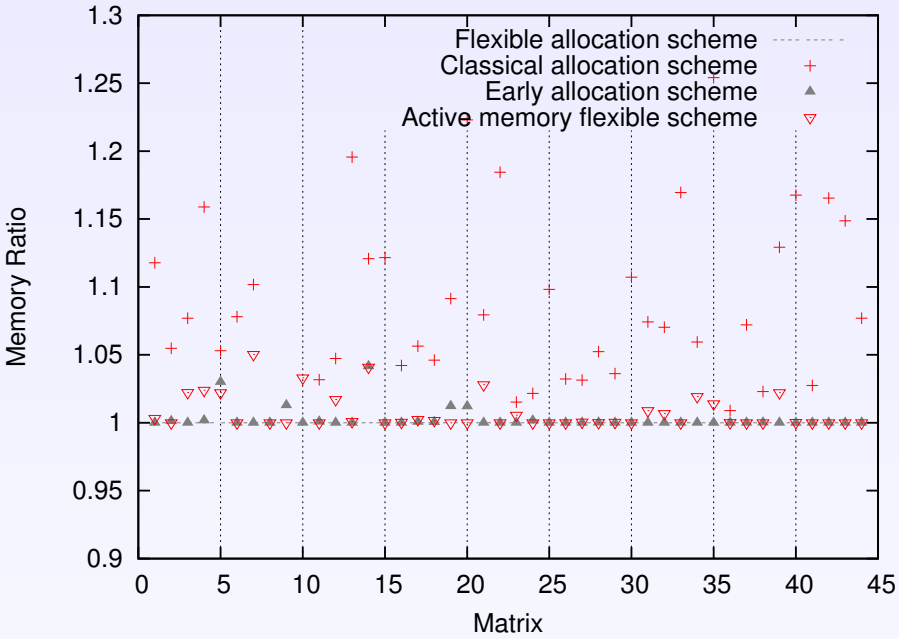
let $j_0 \in \mathcal{S}_2$ be the child for which the peak is reached inside $\mathcal{S}_2$. $\rightarrow$ The total memory peak cannot decrease if j_0 remains in $\mathcal{S}_2$ for all configurations where $\mathcal{S}_1 \subset \mathcal{S'}_1$.

Total memory minimization (3/3)

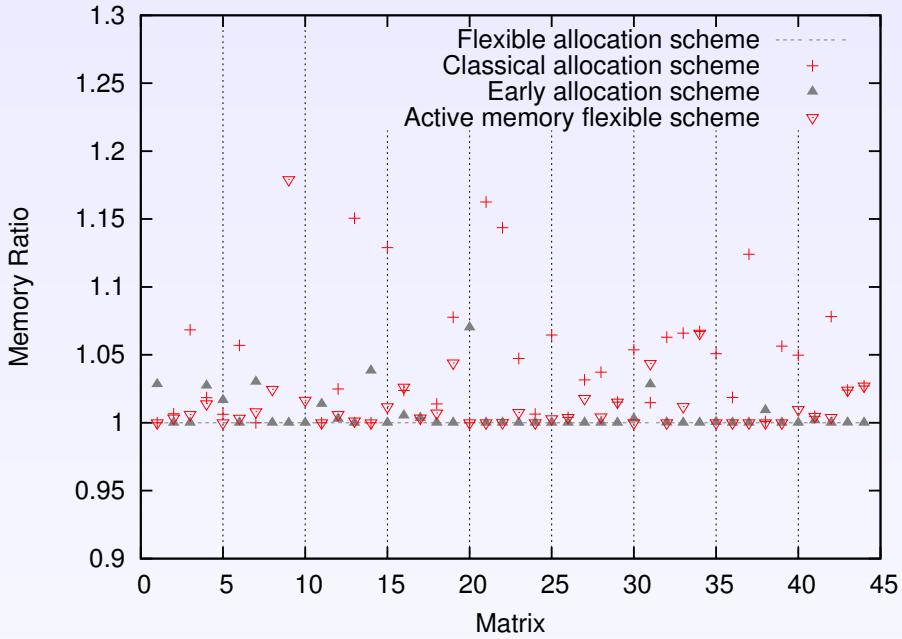
Algorithm:

Set $\mathcal{S}_1 = \emptyset$, $\mathcal{S}_2 = \{1, \ldots, n\}$ and $p = 0$;
Sort $\mathcal{S}_2$ according in decreasing order of $T_j^{flex} - F_j$;
Compute $T^{flex} = \mathcal{P}_2$;
repeat
  Find j_0 such that the peak in $\mathcal{P}_2$ is obtained for j_0;
  Set $\mathcal{S}_1 = \mathcal{S}_1 \cup \{j_0\}$, $\mathcal{S}_2 = \mathcal{S}_2 \setminus \{j_0\}$, and $p = p + 1$;
  (Remark: j_0 is inserted at the position in $\mathcal{S}_1$ so that the order inside this set is decreasing in terms of $T_j^{flex} - (cb_j + F_j)$.)
  Compute $\mathcal{P}_1$, $\mathcal{P}_2$, and $T'^{flex} = \max(\mathcal{P}_1, \mathcal{P}_2)$;
  if $T'^{flex} \leq T^{flex}$ **then**
    Keep the values of p, $\mathcal{S}_1$ and $\mathcal{S}_2$ and set $T^{flex} = T'^{flex}$;
  end if
until $p = n$ or $\mathcal{P}_1 \geq \mathcal{P}_2$

Experimental results



AMD.

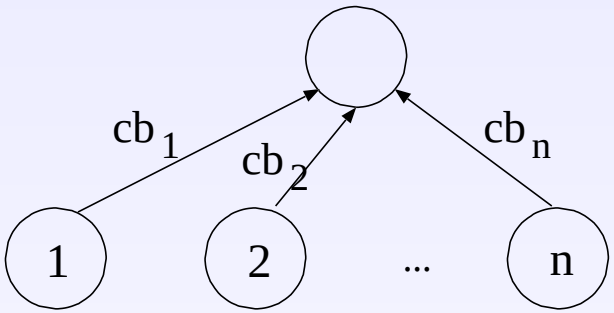


METIS.

Figure: Total memory ratios.

Outline

Notations et assumptions

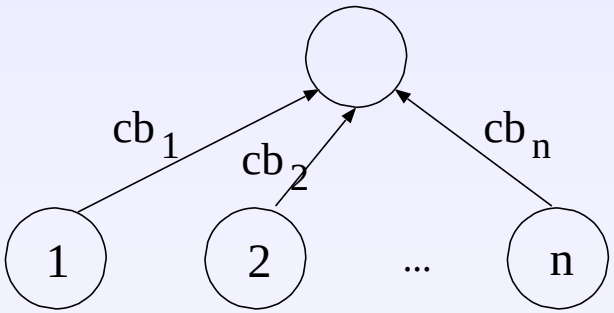


- M_0: Core memory available
- m: Size of the frontal matrix of the parent ($m < M_0$)
- n: Number of children
- i: i^{th} child
- cb_i: Size of the contribution block of child i
- M_i: Memory required to process the subtree rooted at child i

Assumptions:

- Factors are written to disk as soon as computed
- Each frontal matrix fits in core memory (*i.e.* $m < M_0$)
- Oldest CBs written first

Notations et assumptions

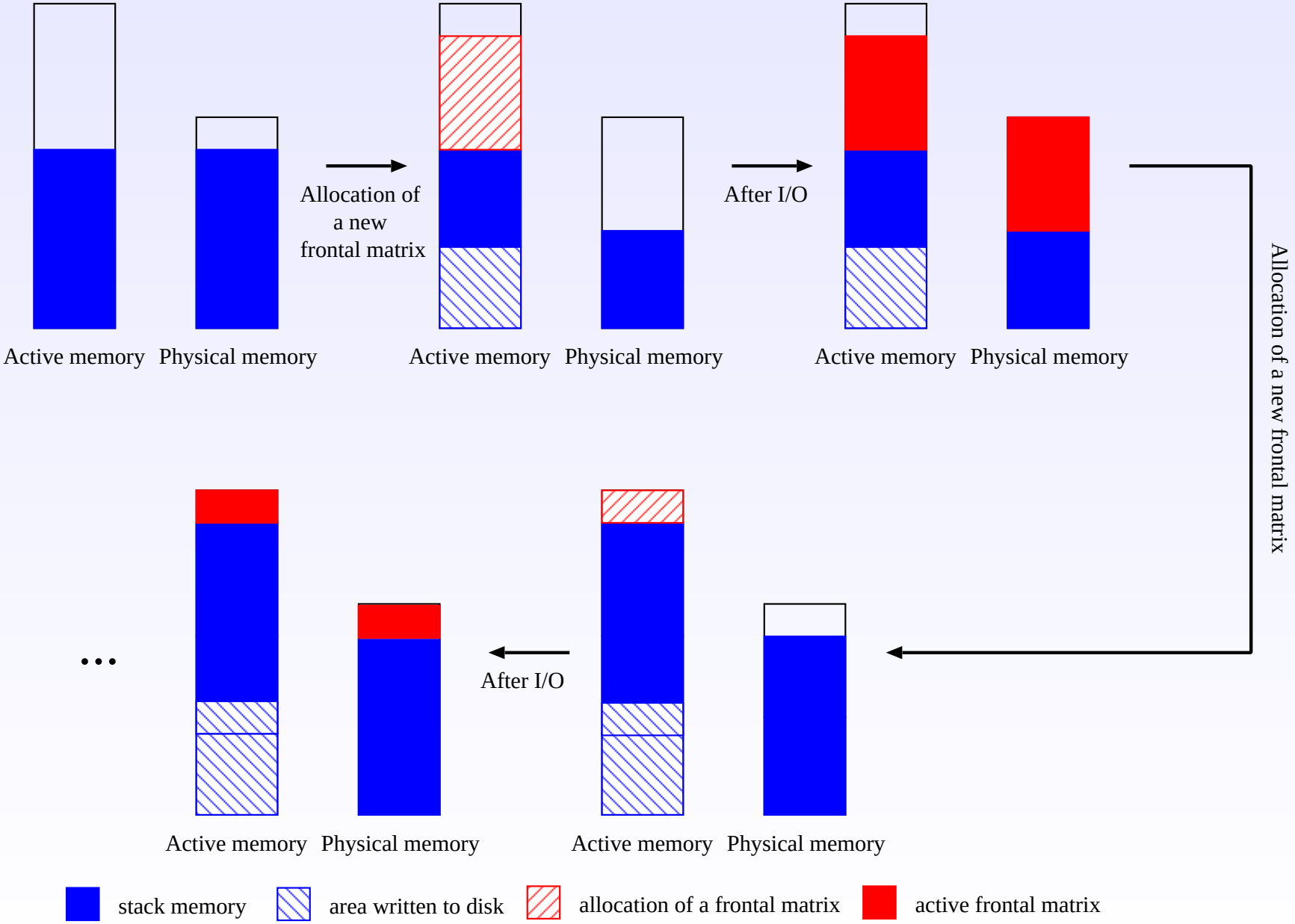


- M_0: Core memory available
- m: Size of the frontal matrix of the parent ($m < M_0$)
- n: Number of children
- i: i^{th} child
- cb_i: Size of the contribution block of child i
- M_i: Memory required to process the subtree rooted at child i

Assumptions:

- Factors are written to disk as soon as computed
- Each frontal matrix fits in core memory (*i.e.* $m < M_0$)
- Oldest CBs written first

I/O volume (1/3)


Active memory
Physical memory
Allocation of a new frontal matrix
Active memory
Physical memory
After I/O
Active memory
Physical memory
Allocation of a new frontal matrix
Active memory
Physical memory
After I/O
Active memory
Physical memory
...
stack memory
area written to disk
allocation of a frontal matrix
active frontal matrix

I/O volume (2/3)

Let be a parent and its children

Case 1: $\forall i \in 1;...;n : M_i < M_0$

I/O produced at assembly step:

$$\max(0, m + \sum_{j=1}^{n} cb_j - M_0)$$

I/O produced when processing the subtree rooted at child j:

$$\max(0, M_j + \sum_{k=1}^{j-1} cb_k - M_0)$$

I/O volume (2/3)

Let be a parent and its children

Case 1: $\forall i \in 1;...;n : M_i < M_0$

I/O produced at assembly step:

$$\max(0, m + \sum_{j=1}^{n} cb_j - M_0)$$

I/O produced when processing the subtree rooted at child j:

$$\max(0, M_j + \sum_{k=1}^{j-1} cb_k - M_0)$$

$V^{I/O}$ is thus:

$$V^{I/O} = \max\left(\max_{j=1,n}(M_j + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{n} cb_j\right)$$

I/O volume (2/3)

Let be a parent and its children

Case 1: $\forall i \in 1;...;n : M_i < M_0$

I/O produced at assembly step:

$$\max(0, m + \sum_{j=1}^{n} cb_j - M_0)$$

I/O produced when processing the subtree rooted at child j:

$$\max(0, M_j + \sum_{k=1}^{j-1} cb_k - M_0)$$

$V^{I/O}$ is thus:

$$V^{I/O} = \max\left(\max\left(\max_{j=1,n}(M_j + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{n} cb_j \right) - M_0, 0 \right)$$

Memory Peak & *I/O* volume (3/3)

Case 2: general case ($\exists i \in 1;...;n : M_i > M_0$)

- Same expression for the active memory
- New expression for the *I/O* volume:
  - If $M_i > M_0$ then:
    1. $V_i^{I/O}$ independent of position of child i
    2. When processing child i, CBs of previously processed children are written to disk

$$V^{I/O} = \max\left(\max_{j=1,n}(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{n} cb_j\right) - M_0 + \sum_{i=1}^{n} V_i^{I/O}$$

Memory Peak & *I/O* volume (3/3)

Case 2: general case ($\exists i \in 1;...;n : M_i > M_0$)

- Same expression for the active memory
- New expression for the *I/O* volume:
  - If $M_i > M_0$ then:
    1. $V_i^{I/O}$ independent of position of child i
    2. When processing child i, CBs of previously processed children are written to disk

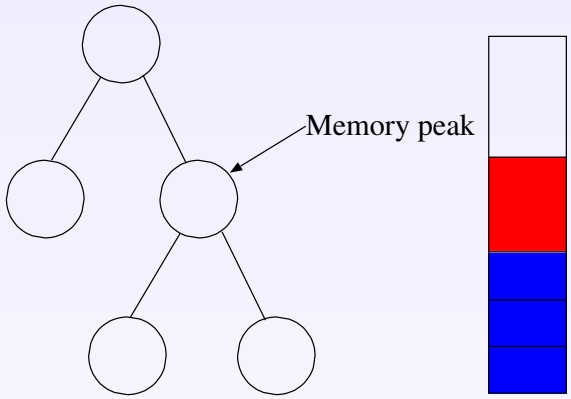
$$V^{I/O} = \max\left(\max_{j=1,n}(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{n} cb_j\right) - M_0 + \sum_{i=1}^{n} V_i^{I/O}$$

$$= \max\left(\max\left(\max_{j=1,n}(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{n} cb_j\right) - M_0, 0\right) + \sum_{i=1}^{n} V_i^{I/O}$$

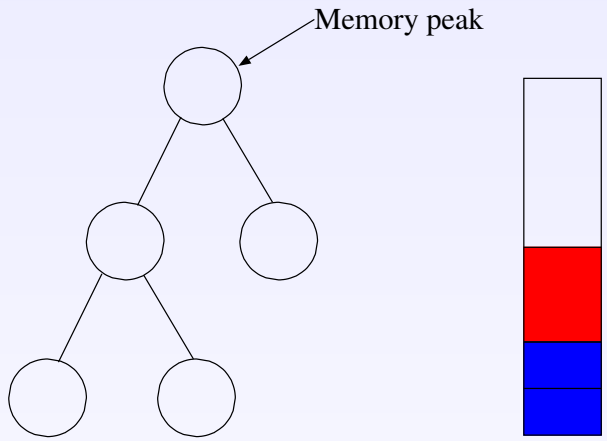
Outline

Problem

- Assumption: factors written to disk as soon as computed
- Active memory peak: tree traversal-dependent



Worst case

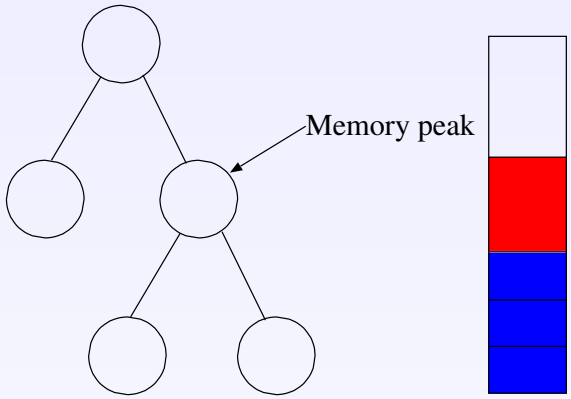


Best case

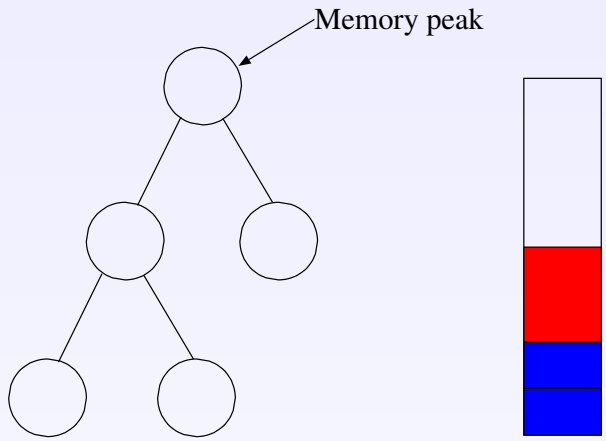
- LIU'86: Optimum algorithm for minimizing the peak of active memory

Problem

- Assumption: factors written to disk as soon as computed
- Active memory peak: tree traversal-dependent



Worst case



Best case

- LIU'86: Optimum algorithm for minimizing the peak of active memory
- Problem: How to minimize the *I/O* volume when the active memory does not hold in a given amount of physical memory M_0 ?

Minimizing *I/O* volume

How to process the children to minimize $V^{I/O}$?

Minimizing *I/O* volume

How to process the children to minimize $V^{I/O}$?

- Minimizing $V^{I/O}$ corresponds to minimize:

$$\max_{j=1,n}\left(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k\right)$$

Theorem (Tree pebbling theorem)

The minimum for $\max_j(x_j + \sum_{i=1}^{j-1} y_j)$ *is obtained when the sequence* (x_j, y_j) *is sorted in the increasing order of* $x_j - y_j$.

Consequence: An optimal sequence is got by ordering the children nodes in the increasing order of $\min(M_j, M_0) - cb_j$.

- Algorithm:
  - Greedy depth-first traversal process
  - Applying the theorem at each level of the tree

Minimizing *I/O* volume

How to process the children to minimize $V^{I/O}$?

- Minimizing $V^{I/O}$ corresponds to minimize:

$$\max_{j=1,n}\left(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k\right)$$

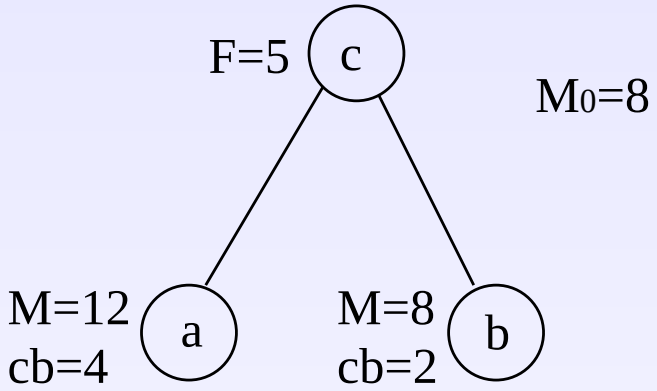
Theorem (Tree pebbling theorem)

The minimum for $\max_j(x_j + \sum_{i=1}^{j-1} y_j)$ *is obtained when the sequence* (x_j, y_j) *is sorted in the increasing order of* $x_j - y_j$.

Consequence: An optimal sequence is got by ordering the children nodes in the increasing order of $\min(M_j, M_0) - cb_j$.

- Algorithm:
  - Greedy depth-first traversal process
  - Applying the theorem at each level of the tree
- Experimental results:
  I/O volume decreased up to 5% on some real problems compared to Liu's algorithm.

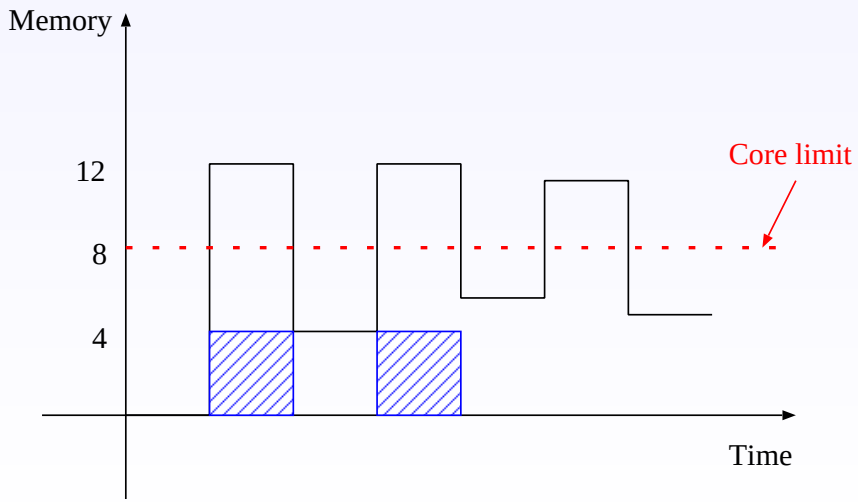
Toy example



Liu's Algorithm

sequence $\longrightarrow$ a-b-c

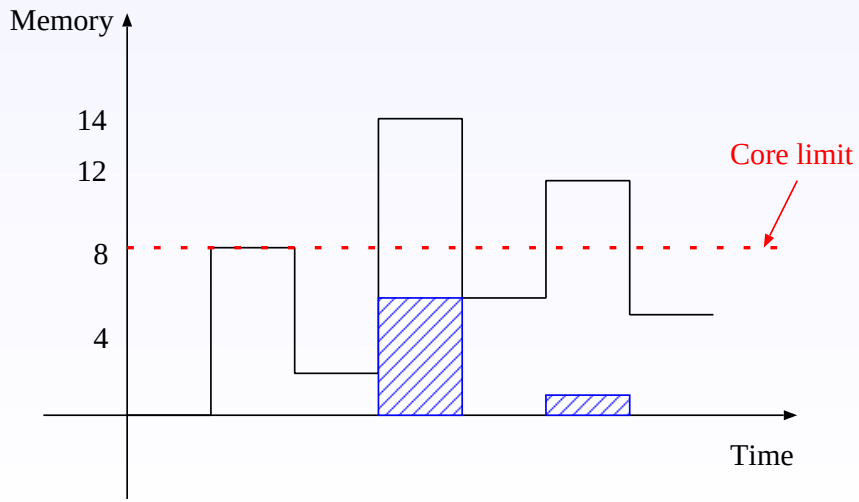
$A_{fath} = 12,\ V^{I/O} = 8$



I/O minimization Algorithm

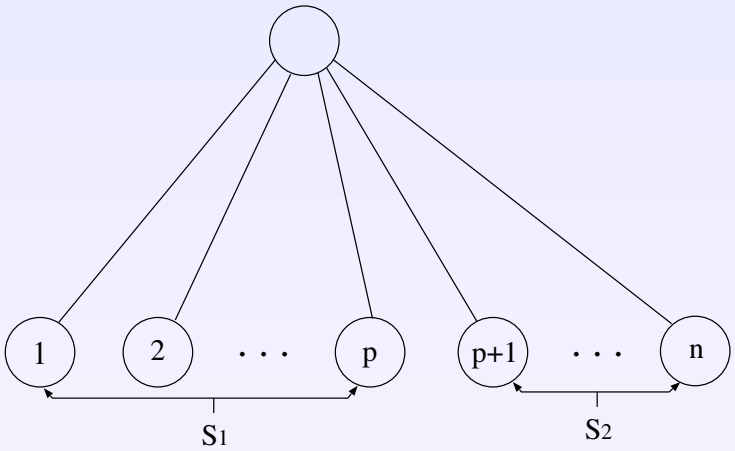
sequence $\longrightarrow$ b-a-c

$A_{fath} = 14,\ V^{I/O} = 7$



Outline

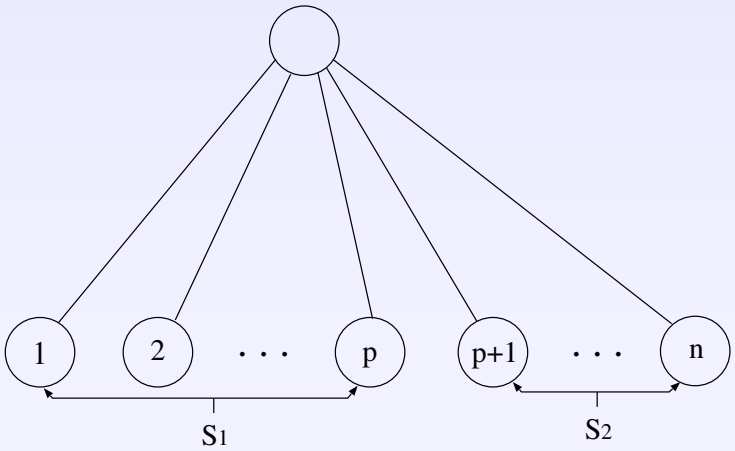
Flexible allocation scheme and *I/O*volume



- p is the position of the allocation of the parent
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent

- Inside $\mathcal{S}_1$: behavior of the classical multifrontal scheme
- Inside $\mathcal{S}_2$: no impact of the order on the *I/O* volume

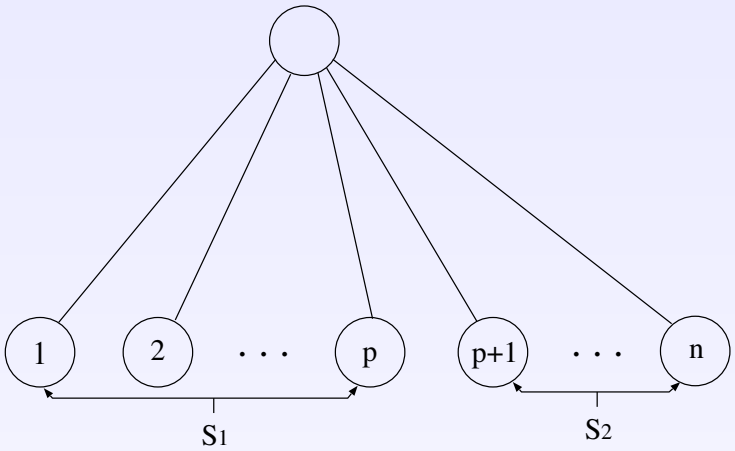
Flexible allocation scheme and *I/O*volume



- p is the position of the allocation of the parent
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent

- Inside $\mathcal{S}_1$: behavior of the classical multifrontal scheme
- Inside $\mathcal{S}_2$: no impact of the order on the *I/O* volume
- We may write (partially/entirely) the parent at any time after its allocation provided we read it again as soon as a CB is produced

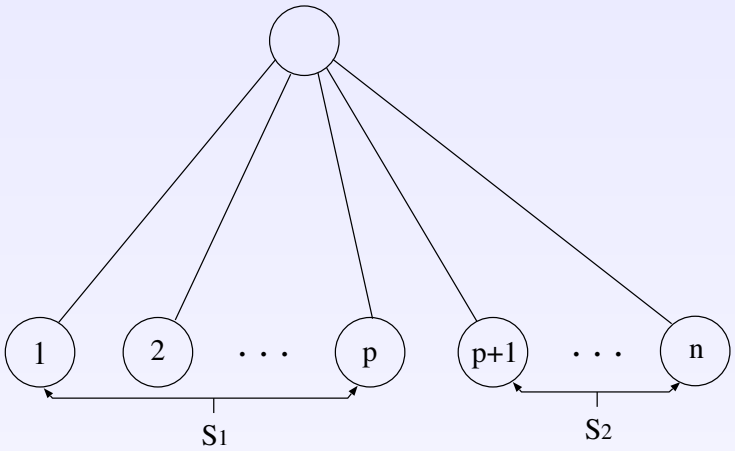
Flexible allocation scheme and *I/O*volume



- p is the position of the allocation of the parent
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent

$$V^{I/O} = \max(0, \max\left(\max_{j=1,p}(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{p} cb_j\right) - M_0)$$

Flexible allocation scheme and *I/O*volume

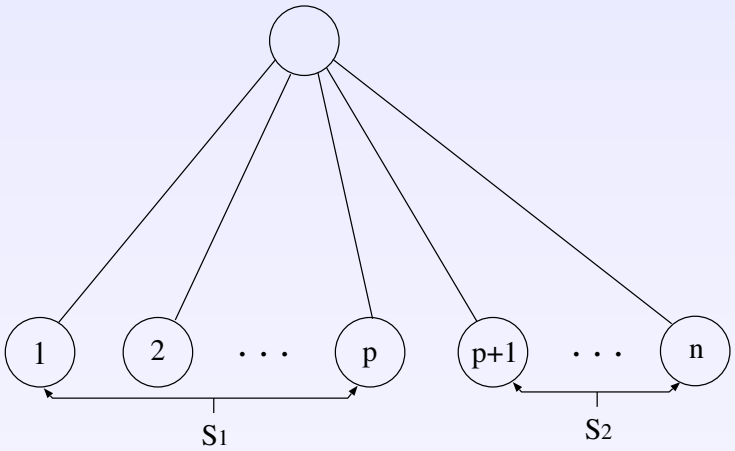


- p is the position of the allocation of the parent
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent

$$V^{I/O} = \max(0, \max\left(\max_{j=1,p}(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{p} cb_j\right) - M_0)$$

$$+ \sum_{j=p+1}^{n} \max(0, m + \min(M_j, M_0) - M_0)$$

Flexible allocation scheme and *I/O*volume

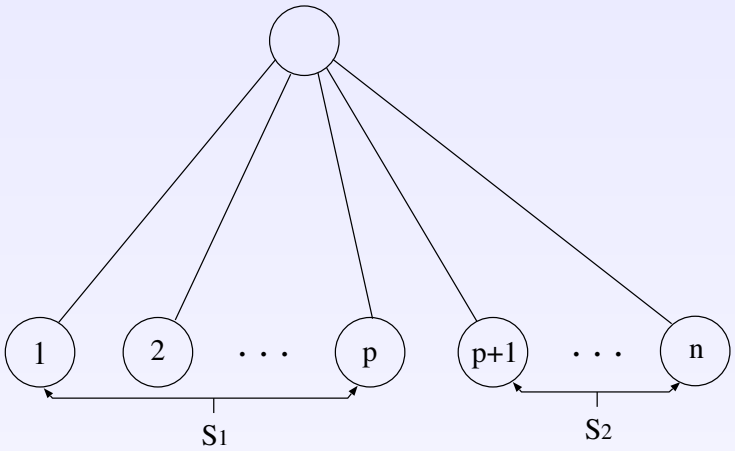


- p is the position of the allocation of the parent
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent

$$V^{I/O} = \max(0, \max\left(\max_{j=1,p}(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{p} cb_j\right) - M_0)$$

$$+ \sum_{j=p+1}^{n} \max(0, m + \min(M_j, M_0) - M_0) + \sum_{i=p+1}^{n} \max(0, cb_i + m - M_0)$$

Flexible allocation scheme and *I/O*volume



- p is the position of the allocation of the parent
- $\mathcal{S}_1$ is the set of children treated before the allocation of the parent
- $\mathcal{S}_2$ is the set of children treated after the allocation of the parent

$$V^{I/O} = \max(0, \max\left(\max_{j=1,p}(\min(M_j, M_0) + \sum_{k=1}^{j-1} cb_k), m + \sum_{j=1}^{p} cb_j\right) - M_0)$$

$$+ \sum_{j=p+1}^{n} \max(0, m + \min(M_j, M_0) - M_0) + \sum_{i=p+1}^{n} \max(0, cb_i + m - M_0)$$

$$+ \sum_{i=1}^{n} V_i^{I/O}$$

Some results

- ⋆ Optimal greedy algorithm when processing of children nodes inside $\mathcal{S}_1$ remains out-of-core after the use of the algorithm

Algorithm.

All the children are initially in $\mathcal{S}_1$
repeat
move to $\mathcal{S}_2$ the child that most reduces the volume of *I/O*
until we cannot save anything

- ⋆ Otherwise, the problem in NP-Complete (reduction to knapsack).
  $\longrightarrow$ The use of the greedy algorithm provides a guaranty

General problem: OOC Multiple Allocation

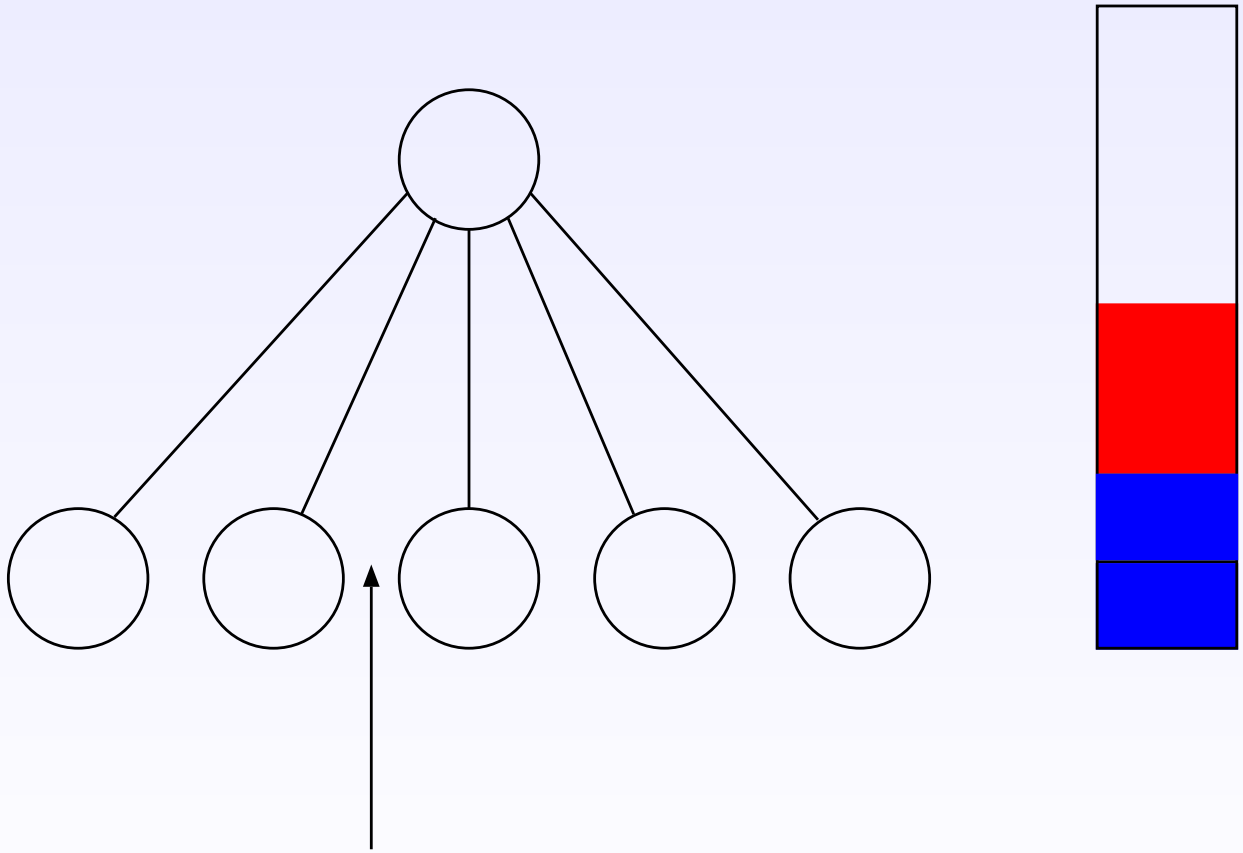
Motivation:

- We are accumulating a volume of *cb*
- We want to consume them from time to time

General problem: OOC Multiple Allocation

Motivation:

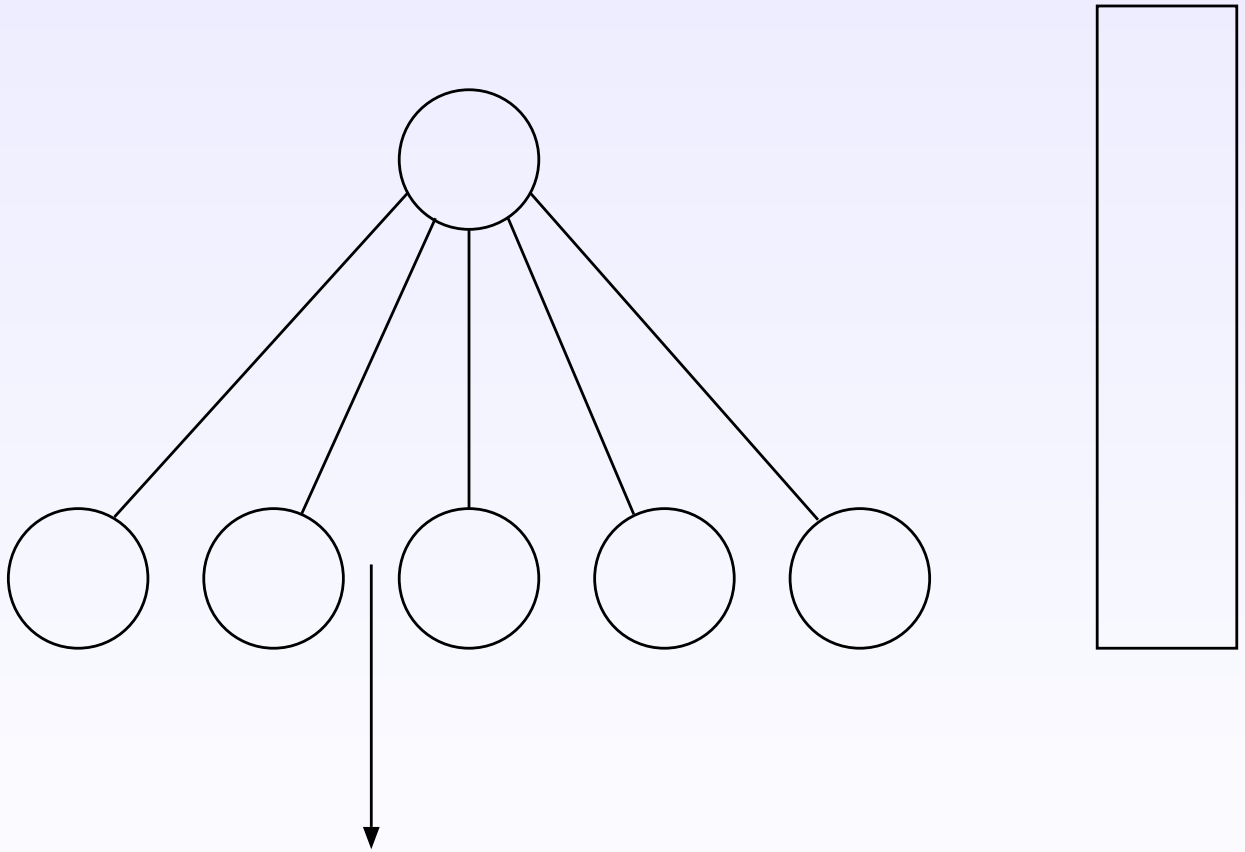
- We are accumulating a volume of *cb*
- We want to consume them from time to time



General problem: OOC Multiple Allocation

Motivation:

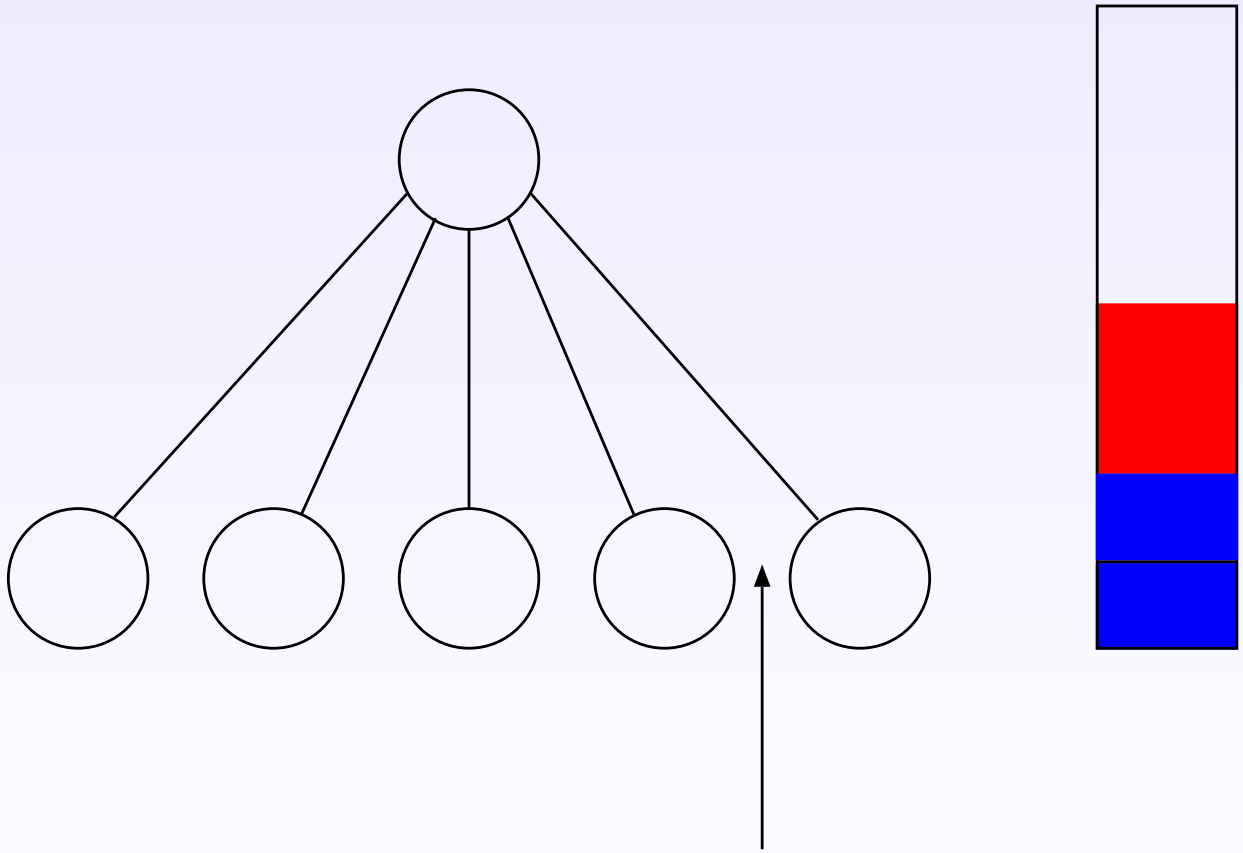
- We are accumulating a volume of *cb*
- We want to consume them from time to time



General problem: OOC Multiple Allocation

Motivation:

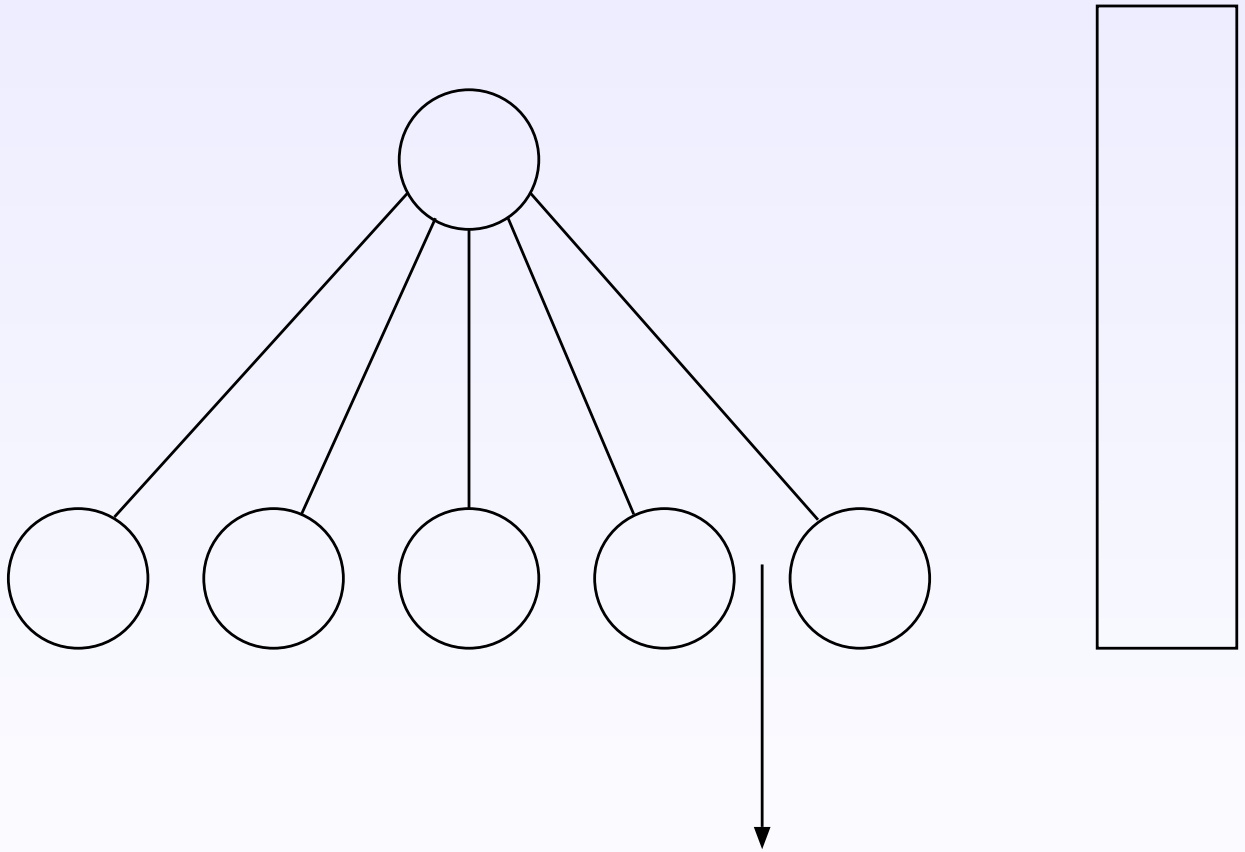
- We are accumulating a volume of *cb*
- We want to consume them from time to time



General problem: OOC Multiple Allocation

Motivation:

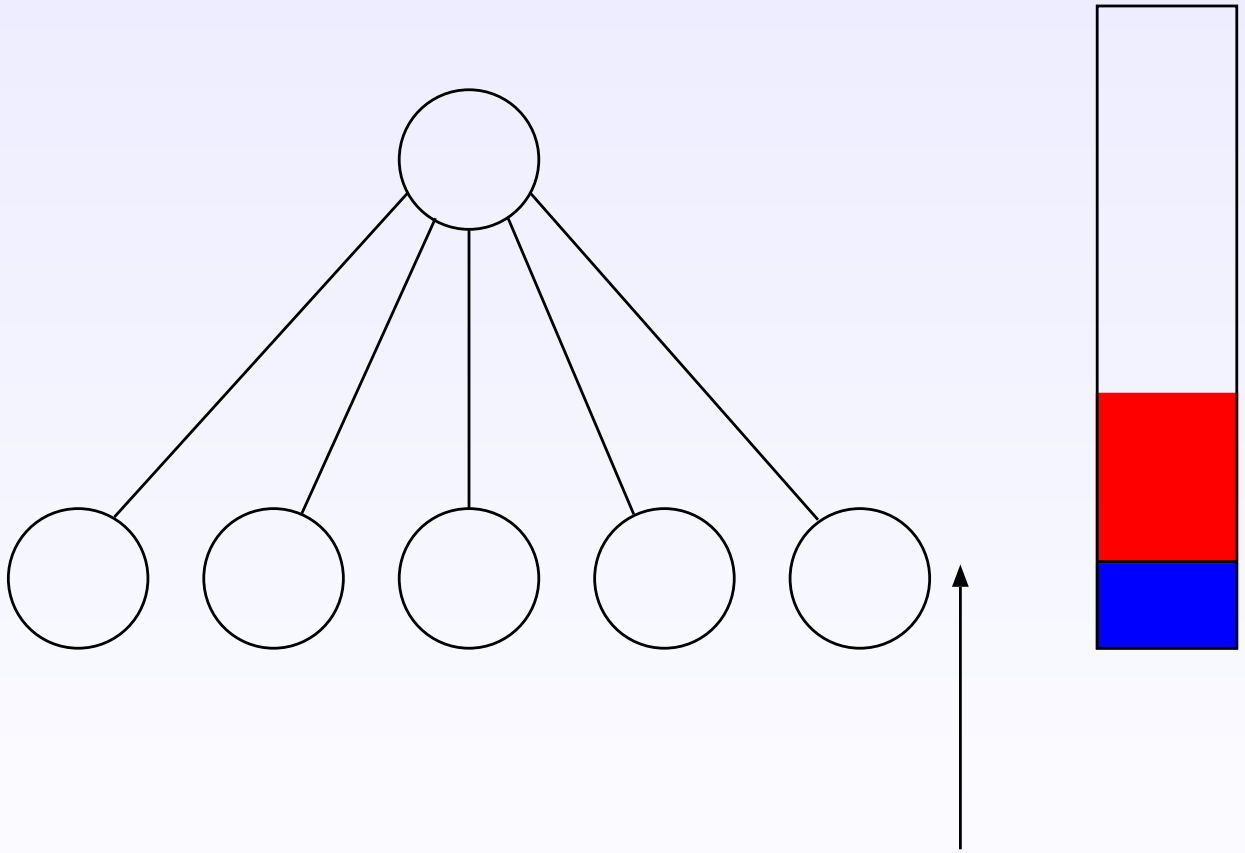
- We are accumulating a volume of *cb*
- We want to consume them from time to time



General problem: OOC Multiple Allocation

Motivation:

- We are accumulating a volume of *cb*
- We want to consume them from time to time



General problem: OOC Multiple Allocation

Motivation:

- We are accumulating a volume of *cb*
- We want to consume them from time to time

- Problem: We have to determine (at each parent node level)
  - a *partitioning* of the children so that for each partition the parent is allocated after its treatment
  - an *order* of treatment of these partitions

  in order to minimize the *I/O* volume

General problem: OOC Multiple Allocation

Motivation:

- We are accumulating a volume of *cb*
- We want to consume them from time to time

- Problem: We have to determine (at each parent node level)
  - a *partitioning* of the children so that for each partition the parent is allocated after its treatment
  - an *order* of treatment of these partitions

  in order to minimize the *I/O* volume
- The associated decision problem is NP-complete

General problem: OOC Multiple Allocation

Motivation:

- We are accumulating a volume of *cb*
- We want to consume them from time to time

- Problem: We have to determine (at each parent node level)
  - a *partitioning* of the children so that for each partition the parent is allocated after its treatment
  - an *order* of treatment of these partitions

  in order to minimize the *I/O* volume
- The associated decision problem is NP-complete
  - Idea of the proof: reduction to a 2-partition

Conclusion and Future work

Memory

- New memory management schemes and corresponding memory minimization algorithms proposed.
- Active memory and total memory cases considered. considered.

Volume of *I/O*

- Optimality for memory minimization $\not\Leftrightarrow$ optimality for volume of *I/O*minimization.
- Several contexts studied and various algorithms/heuristics proposed.

Future work:

- Real-life implementation (modification of the factorization).
- Extension to the parallel case:
  - Add fictive nodes to assemble the distributed contribution blocks?
  - Preallocate parent nodes?