Chapter 7

Qualitative Partitioning

7.1 Properties and Assumptions

- Given: Set of parallel programs *A*;
  Processor connection graph (P, E_P)

- Goal: Mapping from *A* to subsets of processors P

  $$\varphi : A \to \wp(P)$$

  with

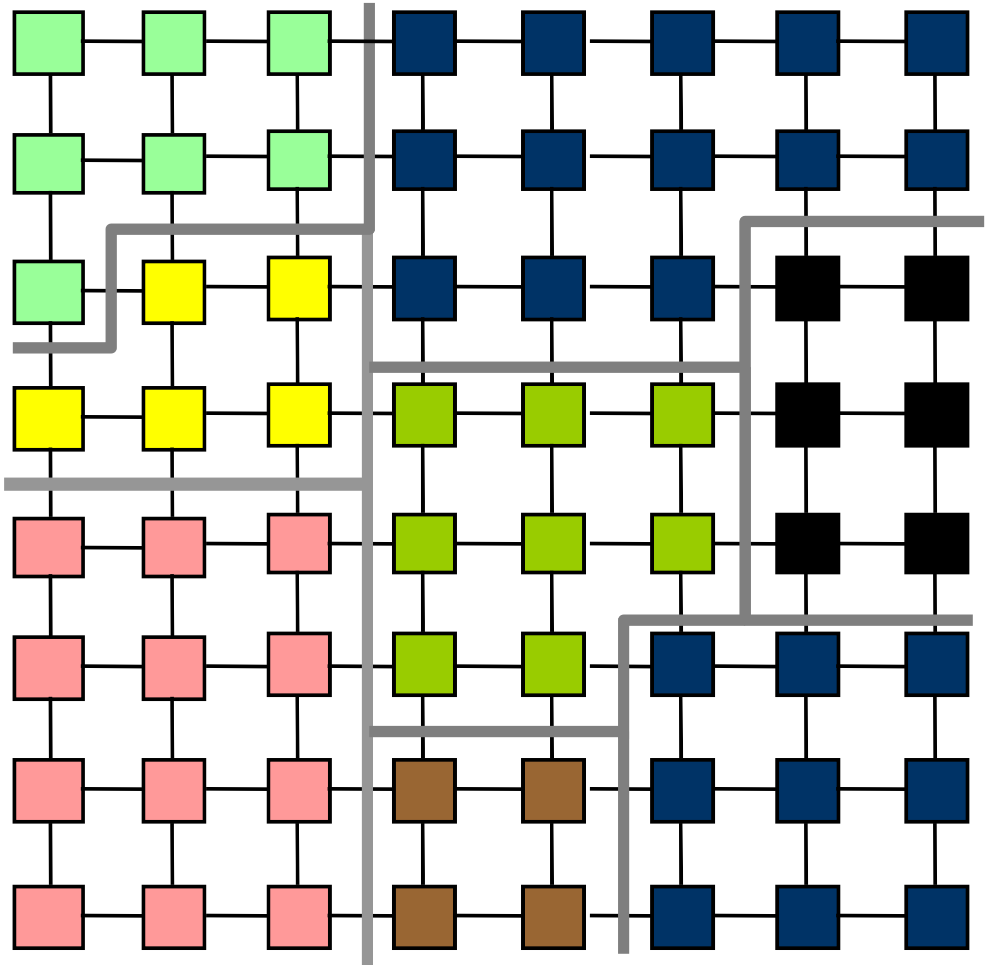
  $$\forall A_i, A_k \in A : \quad i \neq k \;\Rightarrow\; \varphi(A_i) \cap \varphi(A_k) = \emptyset$$

  i.e. the territories $\varphi(A_i)$ are pairwise disjoint
  and
  high utilization at low communication cost

Other Notions

- A territory is called **contiguous,** if the corresponding subgraph is connected.
- Otherwise, the territory is called **non-contiguous.**
- If all territories are contiguous, than the allocation φ is called contiguous.
- Contiguous, pairwise disjoint territories are called **partitions** (pieces).

Example of a Partitioning



Fragmentation

- Allocating parts of a processor network as contiguous pieces leads to fragmentation.

- **Internal fragmentation**:

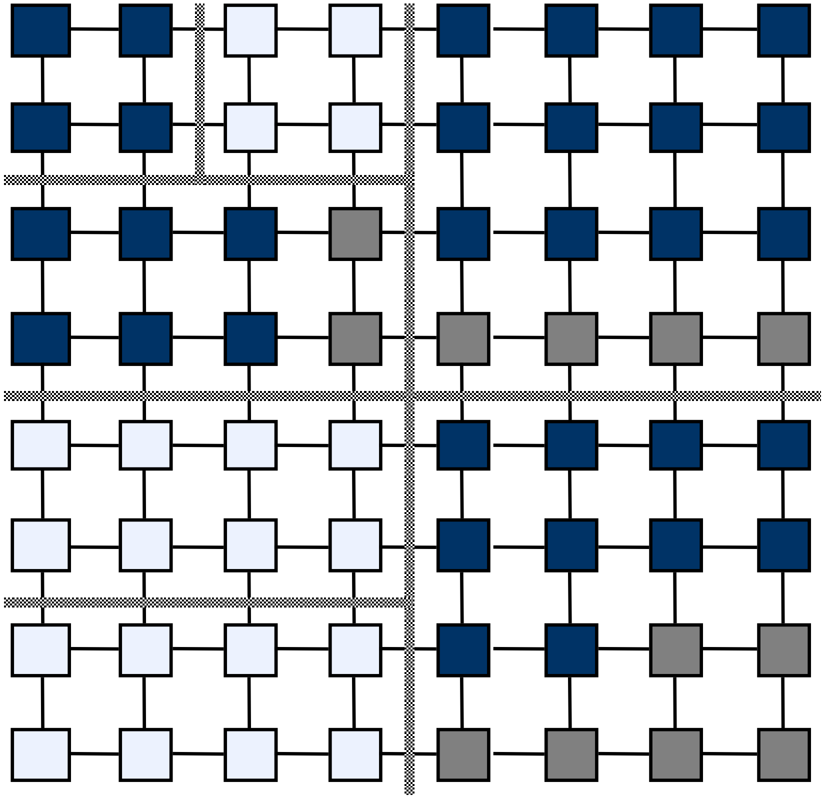
  The piece allocated is larger than requested. A fraction of the allocated processors will not be used.
- **External fragmentation**:

  In the course of allocations and releases small free pieces are generated that cannot be allocated due to their small size.

Fixed Partitioning: Example

A 64-node computer may have a fixed partitioning as indicated below.

One after another some request (programs) of sizes 12, 10, 4, 6, and 9 are arriving. The last request cannot be satisfied.



- ☐ external fragmentation
- ▨ internal fragmentation
- ■ in use

Further Assumptions

- Restriction to grid- or mesh architectures
  - Management of a spatially sharable resource.
  - Generalization of management mechanisms known from memory management (one-dimensional) to 2D or 3D.
  - **Recap: Memory management**
- Dynamic case
  - We assume the resource (processor mesh) is already partially occupied and we have to process requests for free contiguous partitions.
- Distinction
  - **Scalar request**: The request consists of a number (of processors).
  - **Formed request:** The request indicates a rectangle with some breadth and height (b,h).

7.2 Tailored Allocations

- The allocated partition meets the request exactly. No internal fragmentation.

7.2.1 Indicator based allocation

From memory management we know **indicator based management**:

1	1	1	1	0	1	0	0	0	1	1	1	1	0	0	1	0	0	0	0

Each available unit is represented by a bit indicating free (=0) or occupied (=1).

Selection procedure

- **First-fit**

  Sequential scan of the bit vector. First sufficiently large piece will be selected.

- **Next-fit**

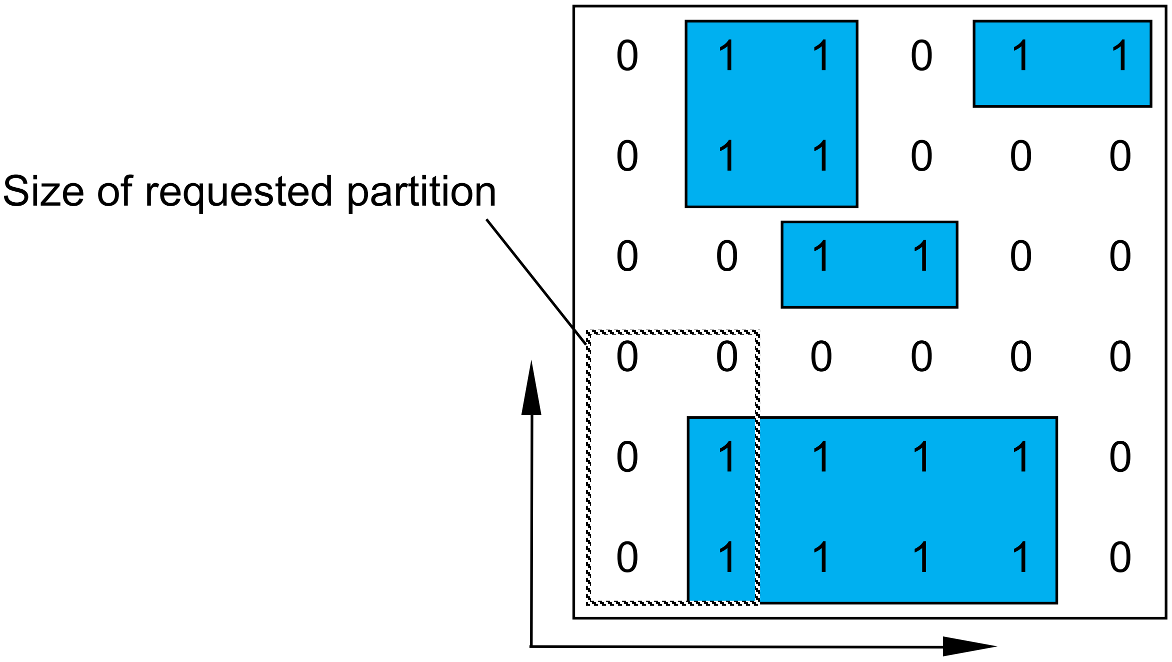
  Like First-fit, but a new scan starts at the position where the last scan was successful. Cyclic scan of vector.

- **Best-fit**

  Instead of taking the first piece that fits, we scan the complete vector to find the smallest piece large enough to fit the request.

2D-Resource (Processor mesh)

- Indicators as matrix
- Request as rectangle b x h
- Scan of matrix row by row from left to right

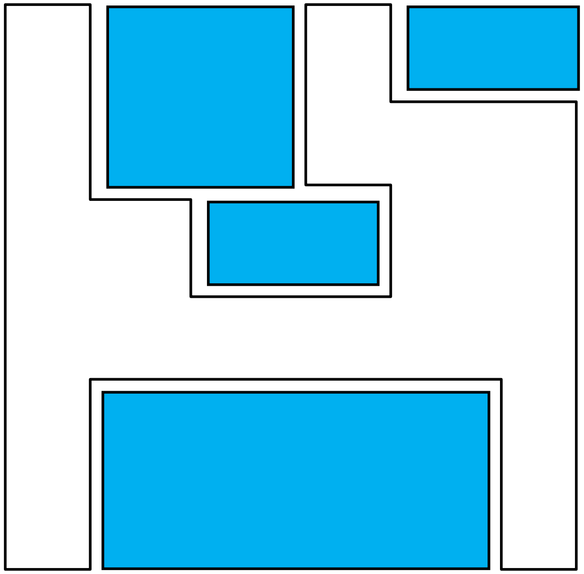


Size of requested partition

0	1	1	0	1	1
0	1	1	0	0	0
0	0	1	1	0	0
0	0	0	0	0	0
0	1	1	1	1	0
0	1	1	1	1	0

7.2.2 List-based Management

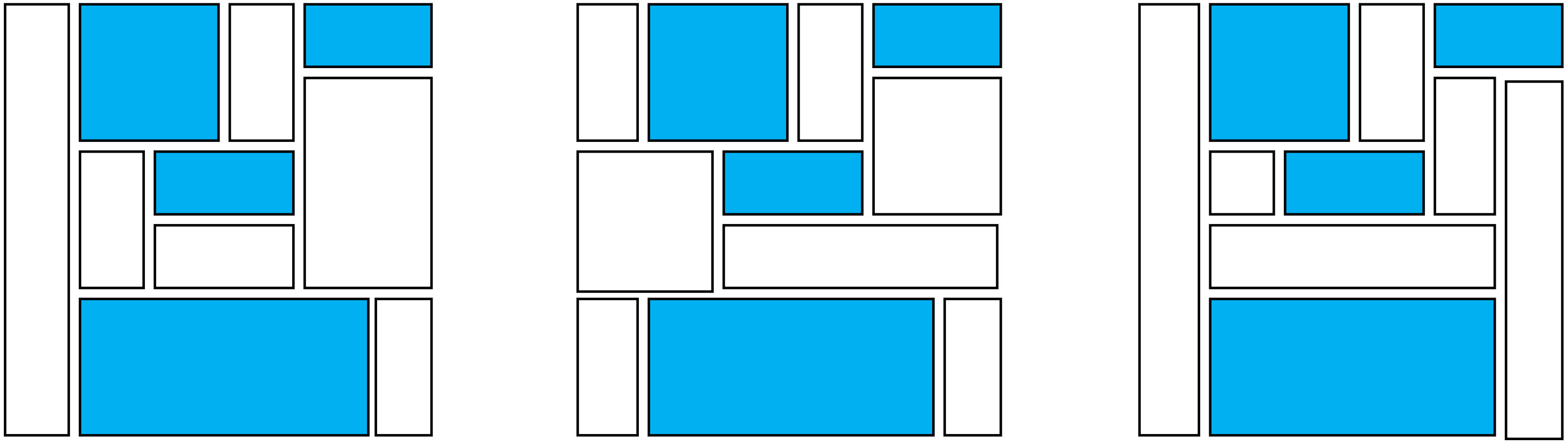
- Free pieces are kept in a list
- Sorting according to several criteria
  - Position
  - Size:
    - Area (no. of processors)
    - Breadth
    - Height



- **Problem**:
  Occupied partitions are well defined, free partitions not.

Decomposition into disjoint rectangles

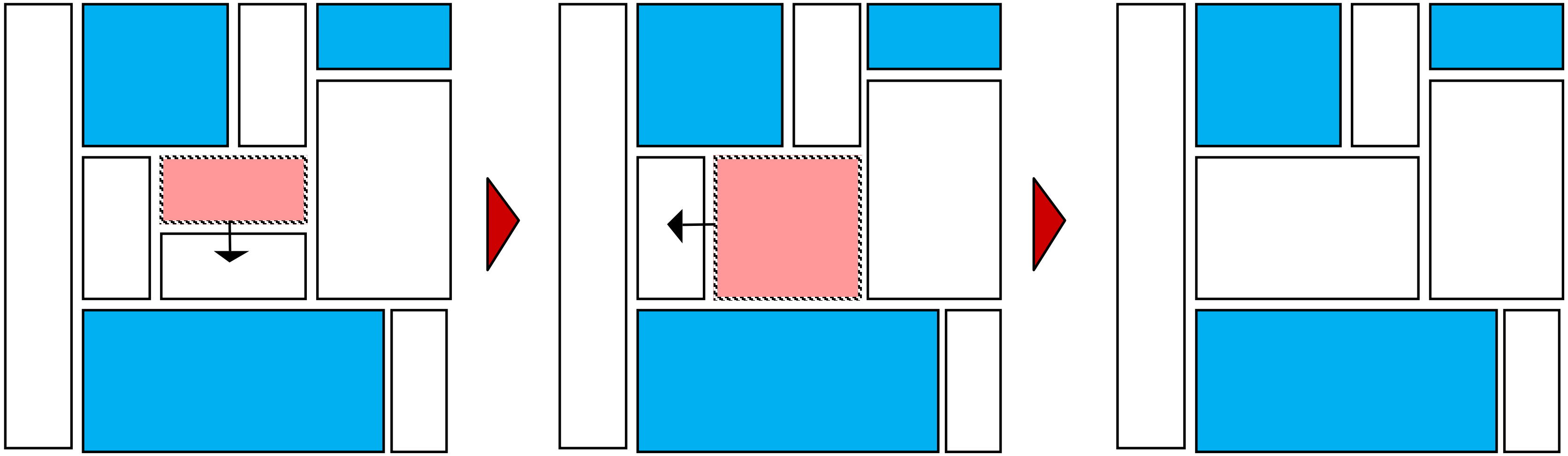
- Which decomposition is better?



Question can only answered if we know what typical requests look like.

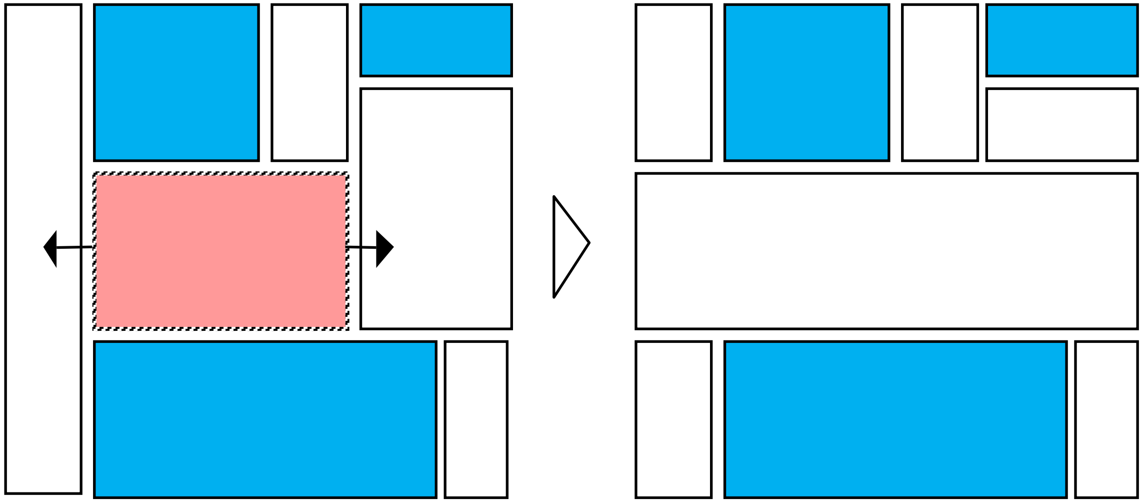
Disjoint free pieces: allocate and release

- Allocation simply means the search for a sufficiently large rectangle.
- After a release, a merger with free adjacent partitions should be performed.
- This is possible, if the two adjacent partitions have one dimension in common:



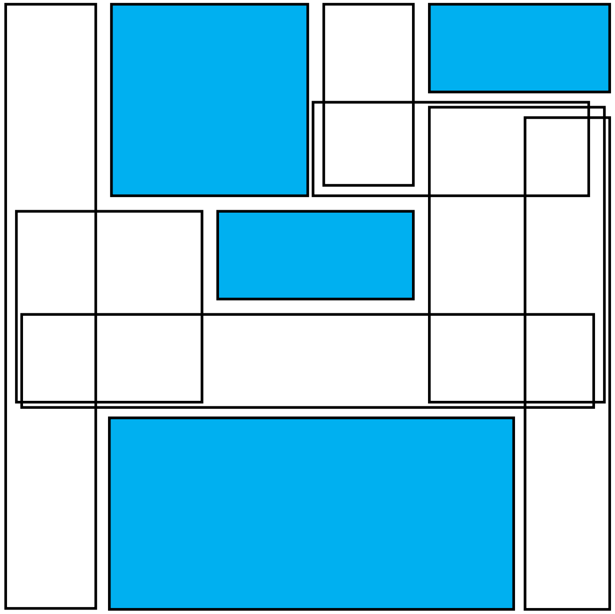
Merger of free partitions

- Further expansion of free partitions would only be possible at the expense of other partitions..
- Whether this makes sense depends on the statistical properties of the requests.



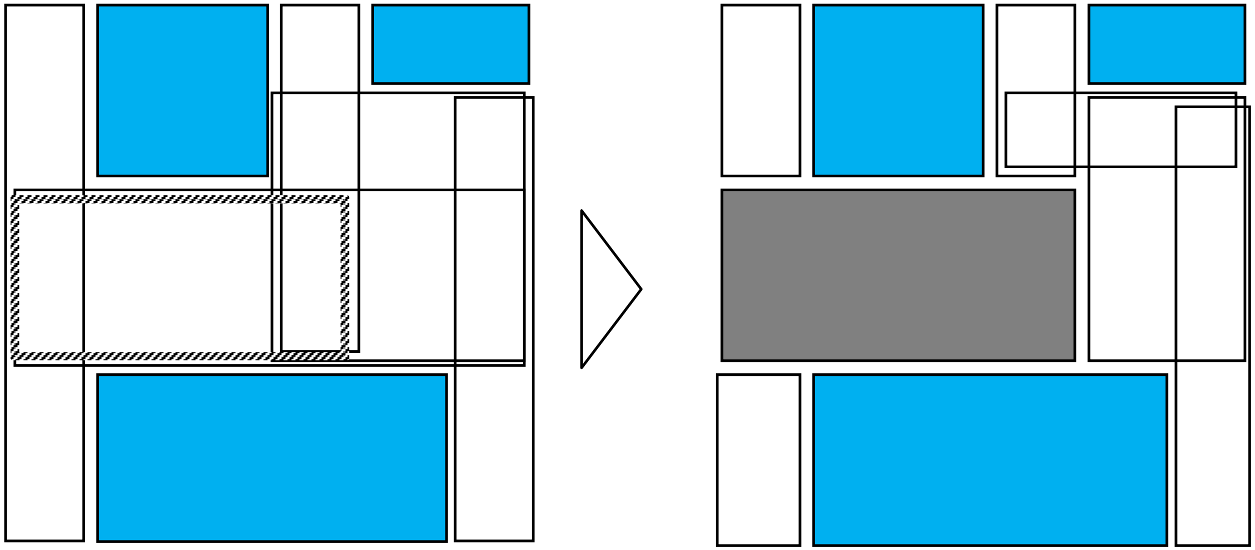
Overlapping free partitions

- When managing overlapping free partitions, we have to accept a higher management overhead, but we get all possible partitions and do not need to restrict to a particular decomposition.



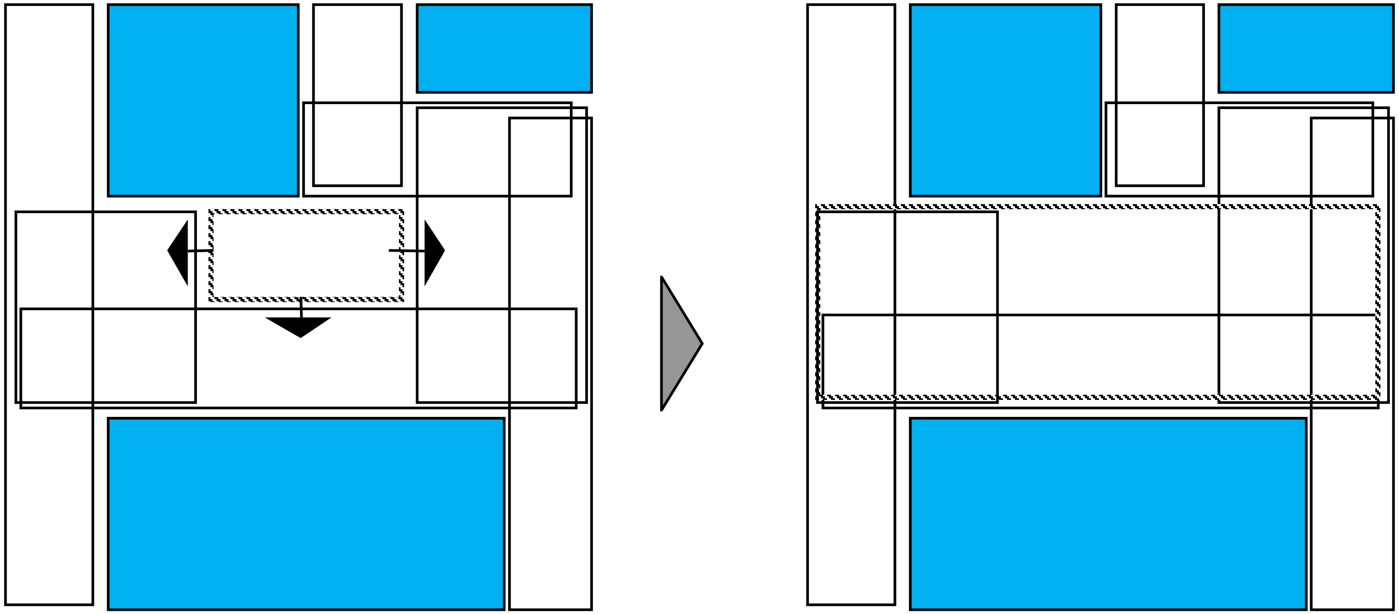
Allocation for overlapping free partitions

- For an allocation the involved free partitions have to be changed:



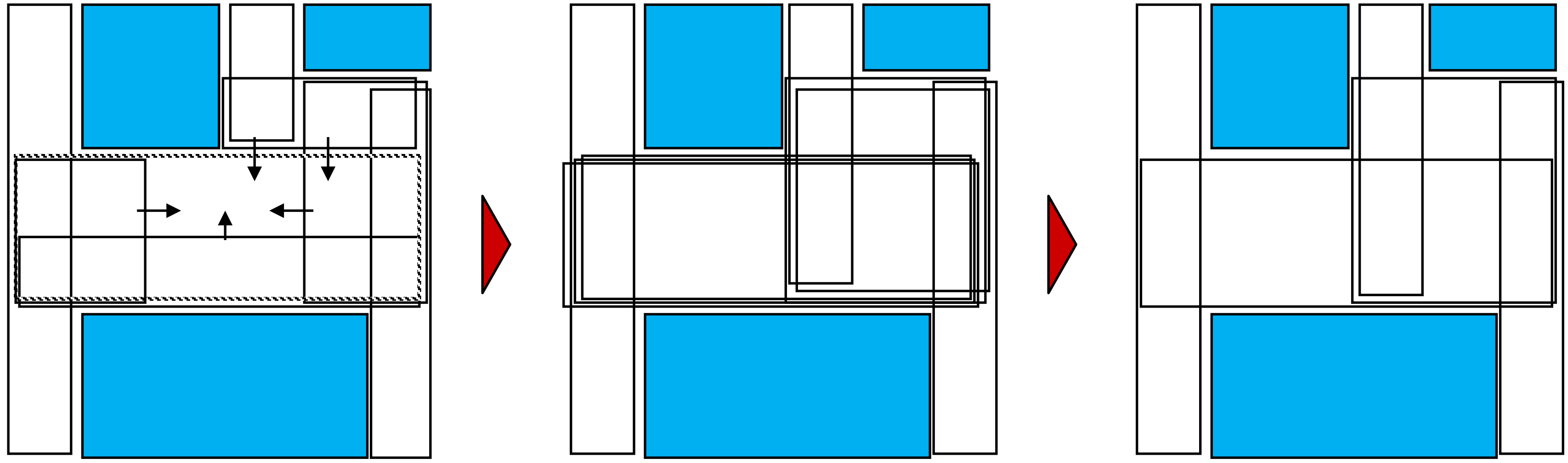
Release for overlapping free partitions

- Expansion of released partition to all possible directions:



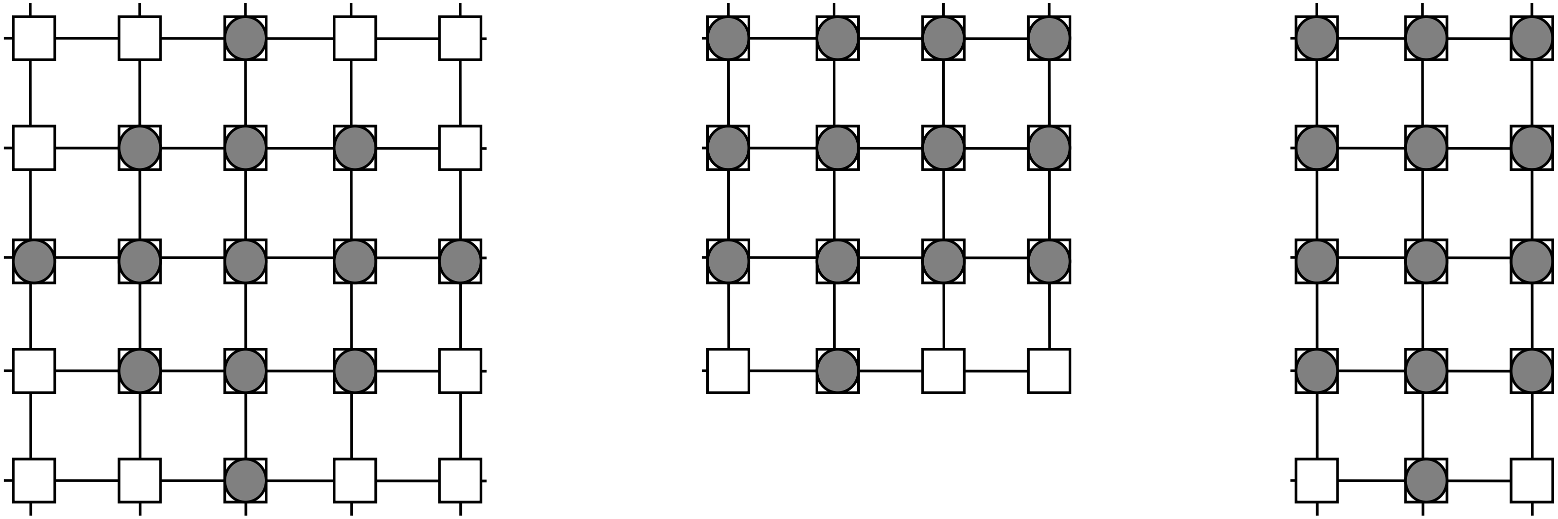
Release for overlapping free partitions 2

- The adjacent free partitions can be expanded as well.
- Identical free partitions that may be generated have to be deleted.



7.2.3 Shaping of scalar requests

- If a request does not specify a rectangle, but only a number (area), we have a new degree of freedom: We have to shape a rectangle of appropriate size.
- Without further knowledge of the program‘s properties (communication behavior) a „compact“ partition is usually better than a narrow one.
- Internal fragmentation may be generated.



Three possible allocations for a request of size 13.

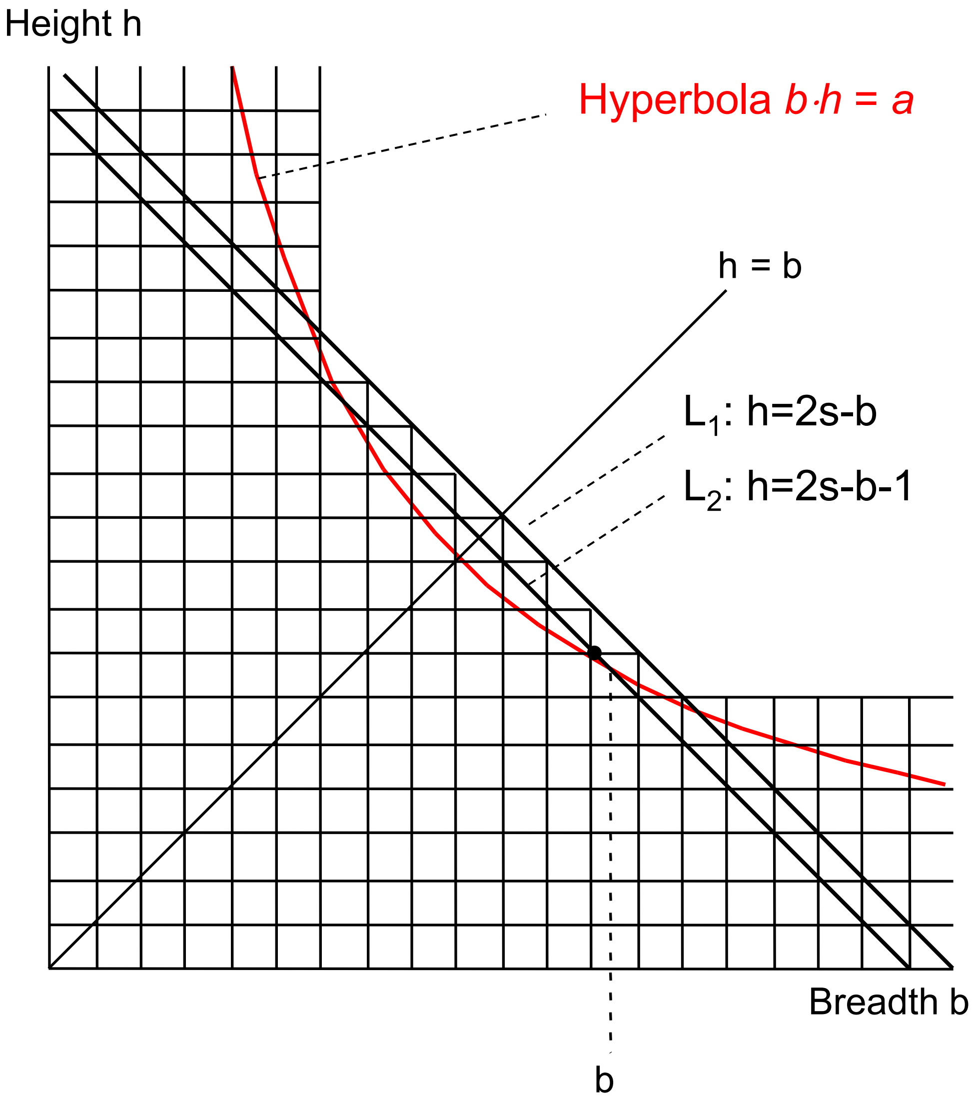
Shaping Problem

- Given: Request of size *a*
- Goal: find a rectangle with $b \cdot h \geq a$ *which*
  - is compact
  - has low internal fragmentation
- Compactness: Small diameter of rectangle to ensure low communication latencies.
  - Minimal diameter
  - Aspect ratio (breadth-height-ratio) of 1 (square)

Finding an optimal rectangle

- All sufficiently large rectangles lie to the right and above the hyperbola.
- On the lines L we find all rectangles with the same diameter.
- s is side length of nearest larger square.
- As intersection points of L with the hyperbola we find

$$b = s - \tfrac{1}{2} + \sqrt{s^2 - s + \tfrac{1}{4} - a}$$



Shaping algorithm

1	`shape(in:a, out:b,h)`	
2	$s \leftarrow \lceil \sqrt{a} \rceil$	Side length of sufficiently large square.
3	**if** a = s · s	If a is a square number,
4	**then**	
5	h ← s	optimal shape has been found
6	b ← s	and procedure stops.
7	**else**	Otherwise
8	$b_1 \leftarrow \lfloor s + \sqrt{s^2 - a} \rfloor$	Breadth b1 and corresponding
9	$h_1 \leftarrow \lceil a / b_1 \rceil$	height are calculated
10	**if** $s^2 - s + 1/4 - a \geq 0$	
11	**then**	If also line L2 intersects with hyperbola
12	$b_2 \leftarrow \lfloor s - 1/2 + \sqrt{s^2 - s + 1/4 - a} \rfloor$	Breadth b2 and corresponding height
13	$h_2 \leftarrow \lceil a / b_2 \rceil$	are calculated
14	**if** $b_2 \cdot h_2 \leq b_1 \cdot h_1$	The rectangle with the smaller area is
15	**then** $b \leftarrow b_2;\ h \leftarrow h_2$	selected. In case of equality we pick
16	**else** $b \leftarrow b_1;\ h \leftarrow h_1$	(b2,h2) due to its smaller diameter
17	**else** $b \leftarrow b_1;\ h \leftarrow h_1$	If line L2 does not intersect, (b1,h1) is a unique solution.
18	**end**	

Example

- Scalar request of size a = 82
- Rounding to the nearest larger square number yields side length s=10
- Algorithm delivers:
  - b1 = 14, h1 = 6
  - b2 = 12, h2 = 7
- Rectangle 12 x 7 is chosen (due to smaller diameter)

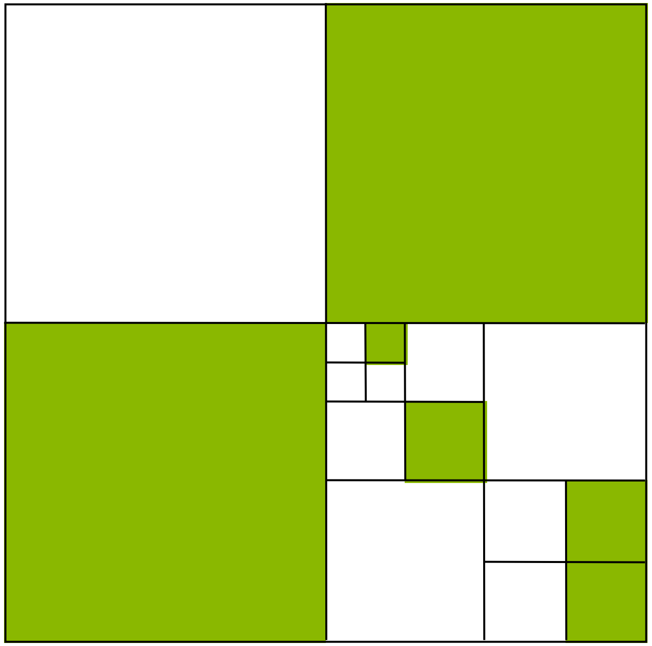
7.3 Buddy systems

- Tailored allocation avoids internal fragmentation but means overhead for finding the best partition.
- From memory management we know algorithms with constant complexity (O(1)).
- The so-called **Buddy system** adapts its offer of free partitions to the request profile.

- Recap: Binary Buddy in Memory Management

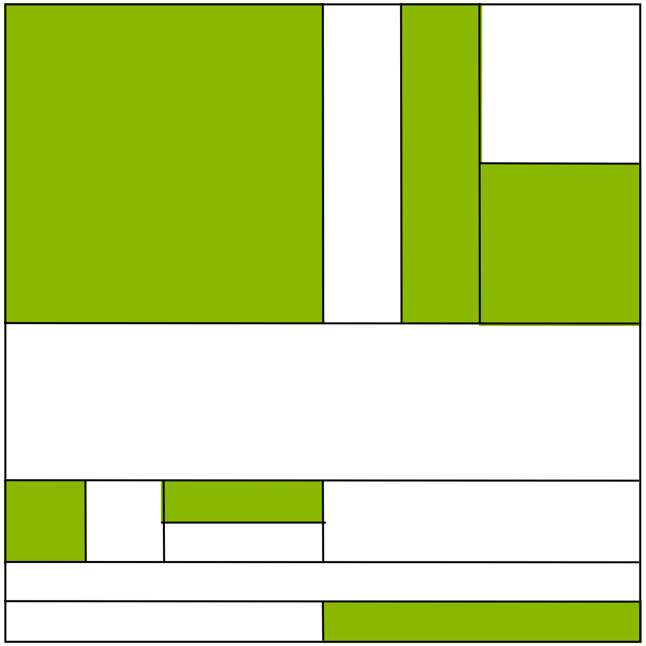
2D-Buddy-System

- Assumption: Prozessor mesh 2^n x 2^n
- Variant: Dividing simultaniously in both dimensions



2D-Buddy-System

- Variant: Dividing independently in both dimensions



2D-Buddy-System

- Variant: Dividing the area

Calculation of fragmentation

- Assumption: request uniformly distributed from $[1, 2^n]$.

	Variant 1	Variant 2	Variant 3
Internal fragmentation	1 - 7/12 (42%)	1 - 9/16 (44%)	1 - 3/4 (25%)

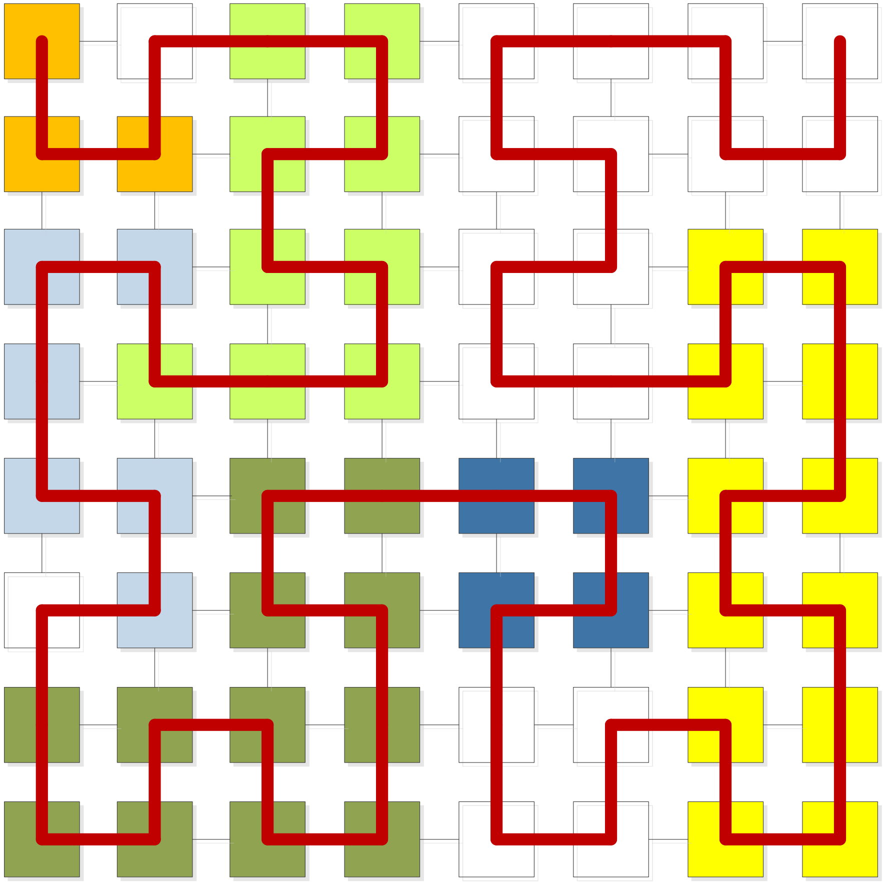
7.4 Hilbert curve - SLURM

- SLURM workload manager is a widely used resource management system for HPC systems

- SLURM implements tailored partitioning
  - First fit approach over vector of nodes (indicator bases management)

- Mapping extensions to consider architecture and topology of the machine:
  - Order nodes following Hilbert curve for mesh/grid topologies
  - Default configuration for 3- or multi-dimensional topologies

SLURM – Example



7.5 Non-contiguous Allocation

- The allocation of contiguous partitions generates internal fragmentation and also substantial external fragmentation.
- The utilization could be improved if we could satisfy a larger request with a set of some smaller free partitions.
- Non-contiguous allocation can be used complementary to contiguous allocation, when, e.g., a contiguous allocation fails.
- The decomposition of the parallel program into non-contiguous small partitions is influenced by two aspects:
  - Communication oriented decomposition:

    The communication graph TIG – if available – is decomposed according to its edge weights such that we obtain components with only little communication in between.
  - Fragmentation oriented decomposition:

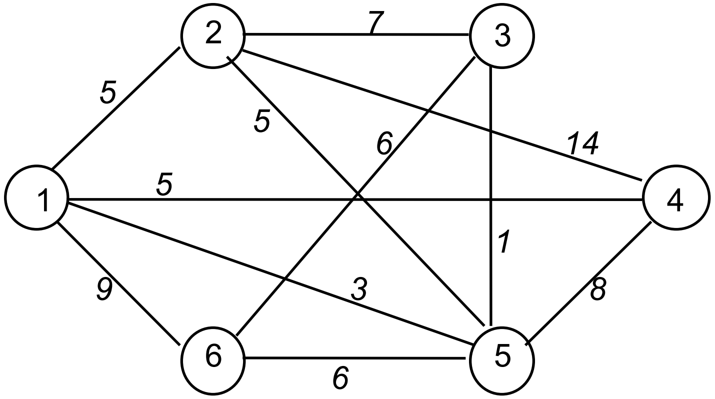
    The decomposition is governed by the offer of free partitions.

7.5.1 Decomposition of the communication graph

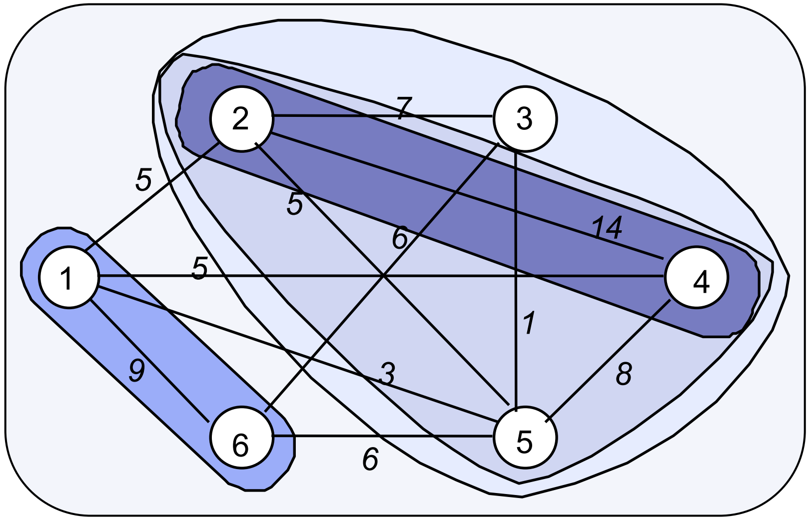
To decompose a parallel program according to the free partitions currently available, we use a recursive hierarchic clustering scheme:

1. Sort the edges of the graph according to decreasing edge weights.
2. For initialization, all *m* nodes make up single-element clusters.
3. Step by step clusters are being merged that are connected by the heaviest edge not yet considered.
4. The algorithm stops when all nodes have been merged to one single cluster.

Example



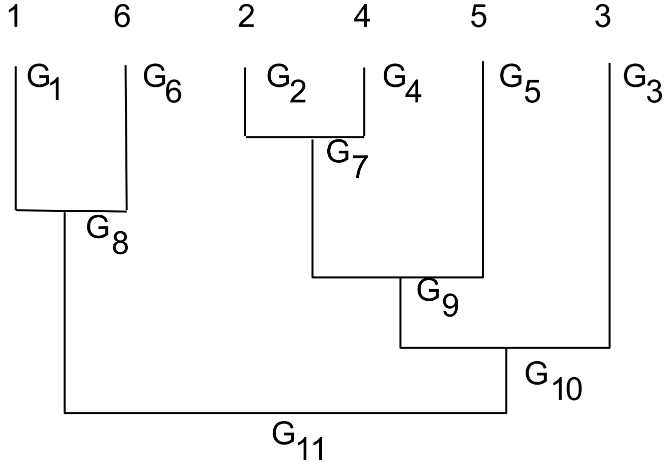
Communication graph (TIG)



Clustering

Dendrogram:

Level	Cluster
1	$G_1 G_6 G_2\ G_4\ G_5\ G_3$
2	$G_1 G_6\ G_7\ G_5\ G_3$
3	$G_8\ G_7\ G_5\ G_3$
4	$G_8\ G_9 G_3$
5	$G_8\ G_{10}$
6	G_{11}

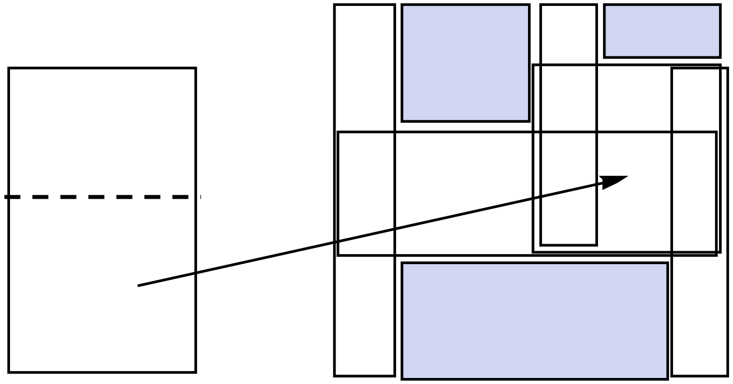


7.5.2 Fragmentation oriented decomposition

- Assumption:
  - Rectangular requests
  - List-based management of free partitions

- Based on a list-based approach that sorts free partitions according to breadth and height as well, we can formulate an algorithm:
  - Find a free partition that satisfies the request in one dimension.
  - Cut the fitting piece and search for a free partition for the remainder of the request.
  - Continue recursively until the request is satisfied.

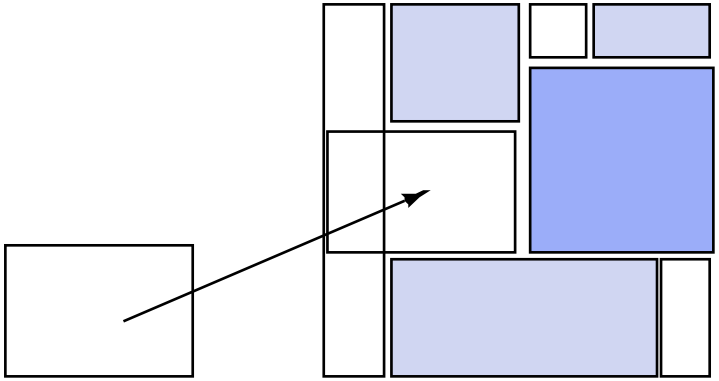
Example



request

allocation situation

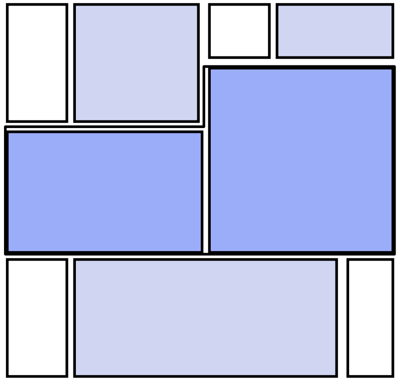
(a)



rest of request

new allocation situation

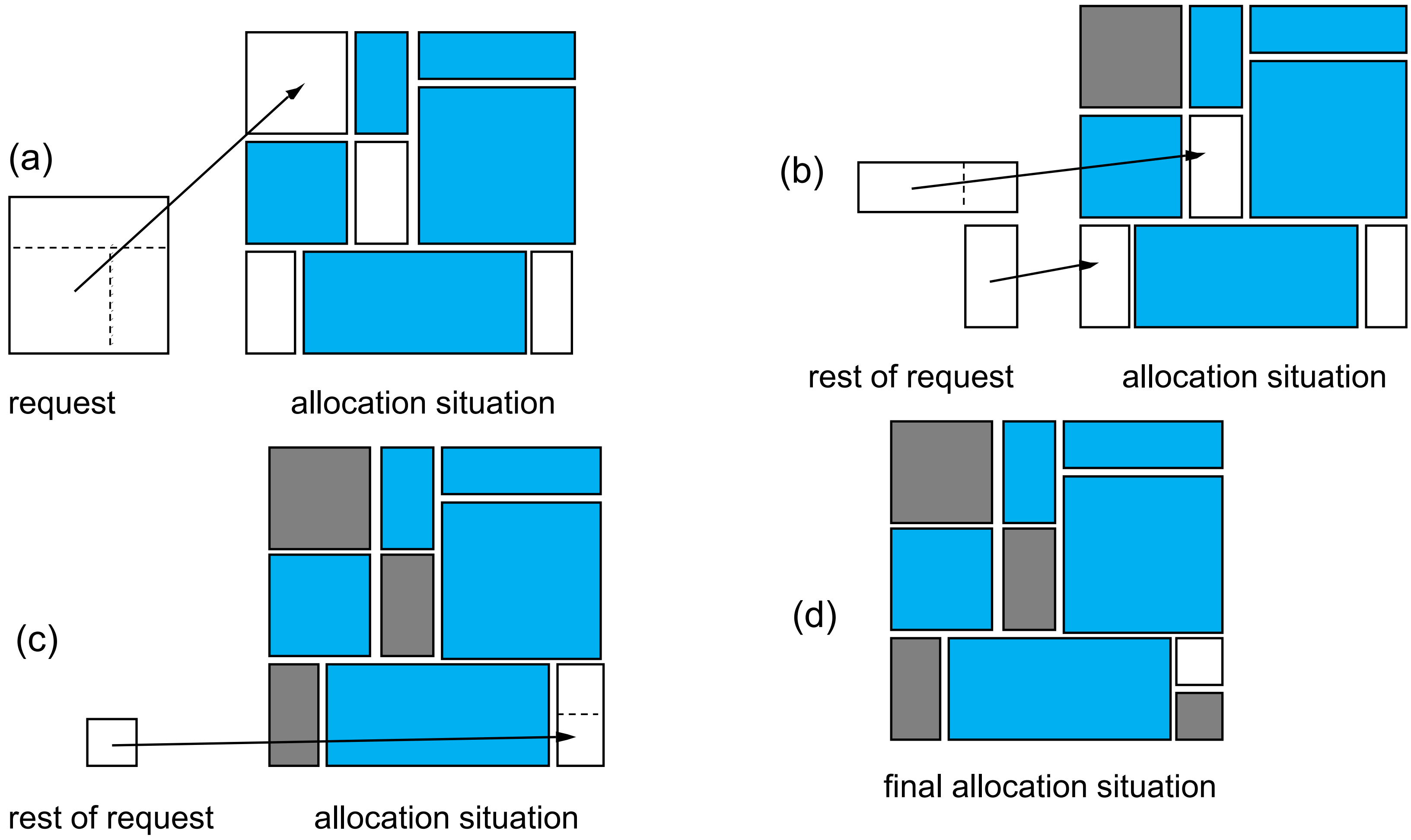
(b)



final allocation situation

(c)

Example

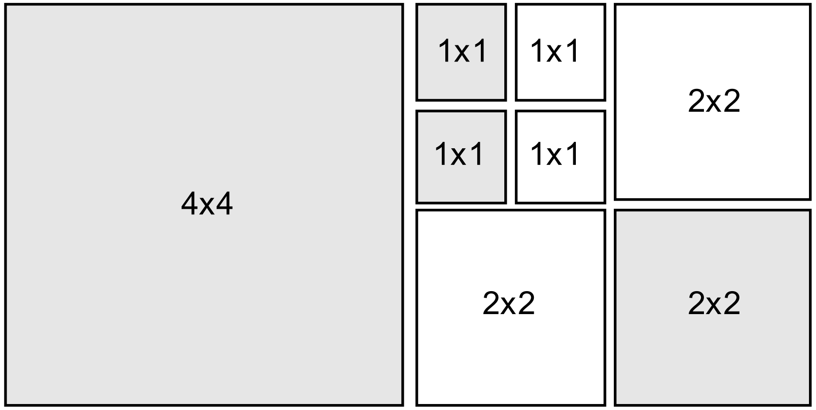

(a)
request
allocation situation
(b)
rest of request
allocation situation
(c)
rest of request
allocation situation
(d)
final allocation situation

7.5.3 Tree oriented Buddy-system

- Partitions are generated by cutting into halves.
- Allocation situation is represented by a binary tree.
- Each node (in tree) represents a dynamically generated partition and indicates the number of its free processors.
- Internal nodes represent partially occupied partitions.
- Leaf nodes represent partitions that are either completely free or completely occupied.

Example of an allocation situation

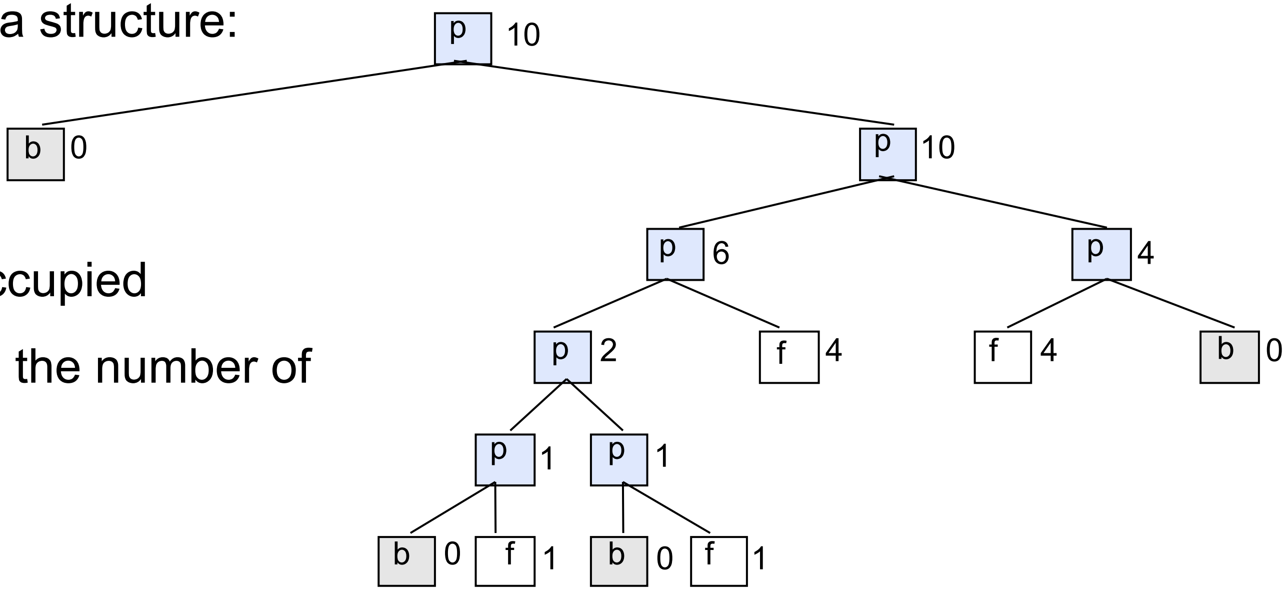
allocation situation in a 4x8-mesh



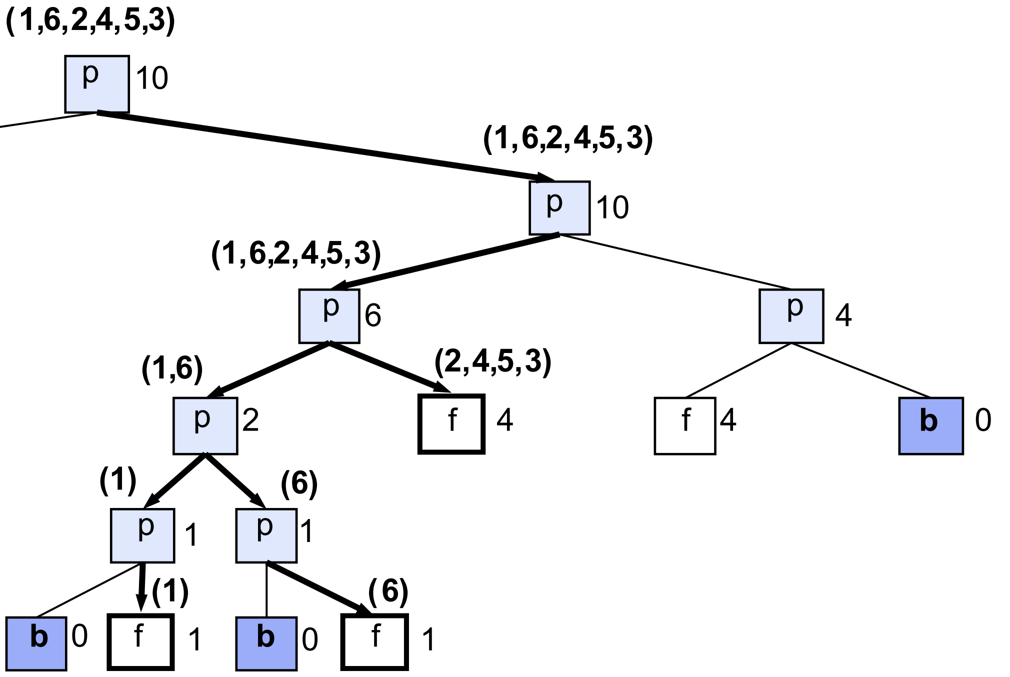
Correponding data structure:

- b: occupied
- f: free
- p: partially occupied

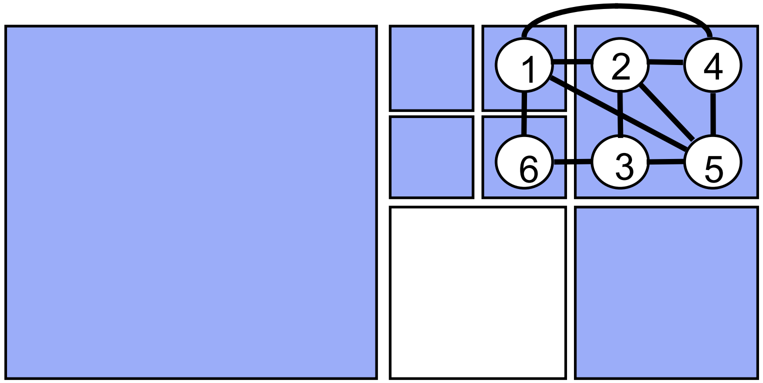
Numbers indicate the number of free processors



Example



Allocation for request from slide 31 (size 6)



Resulting allocation and placement of processes

Simulation results

Machine: 64x64 Processor mesh

Requests: Rectangles with side length equally distributed from [0,32]

Algorithm	Total runtime	Internal fragment.	External fragment.	Utilization
Contiguous allocation				
List-based disjoint	86	0	24%	76%
Buddy	118	37%	7%	56%
Fibonacci Buddy	147	22%	33%	45%
Non-contiguous allocation				
Tree-based Buddy v=1	71	0%	7%	93%
Tree-based Buddy v=1.5	72	5%	4%	91%

Summary

- Static vs. dynamic partitioning
  - Distribution of requests known?
- Tailoring vs. Standard sizes
  - Buddy-System shows smaller external fragmentation (<10%) than list-based management (20-40%).
  - Due to high internal fragmentation (>25%), the total utilization of buddy-system usually worse.
  - The low algorithmic complexity of buddy-system does not pay off with processor numbers < 10^6.
- Contiguous vs. non-contiguous allocation
  - Significantly better utilization with non-contiguous allocation (85-95%)
  - Application runtime depending on communication intensity
  - Appropriate for dynamic processor demands

Further Reading

- Heiss, H.-U.; Wiesenfarth,R.: *A Heuristic Algorithm for Dynamic Task Allocation in Highly Parallel Systems.* in: H.P. Zima (ed.): Parallel Computation, Lect. Notes in Comp. Science No. 591, Springer (1992) pp. 252-265.
- Heiss, H.-U.: *Processor Management in 2D-Grid Architectures: Buddy-Systems.* GI-PARS-Reports Nr. 12 (Proc. Workshop „Fine-grain and Massive Parallelism“, Dresden, 6.-8. April 1993), pp.14-23.
- Heiss, H.-U.: Dynamic Partitioning of Large Scale Multicomputer Systems, Proc. Conf. on Massively Parallel Computing Systems (MPCS'94), Ischia, 2.-6. Mai, 1994
- Bender, Michael A.; Bunde, David P. ; Demaine, Erik D.; Fekete, Sandor P.; Leung, Vitus J.; Meijer, Henk; Phillips, Cynthia A.: Communication-Aware Processor Allocation for Supercomputers, Proc. of the 9th Workshop on Algorithms and Data Structures (WADS), 2005
- De Rose, César A.F.; Heiss, Hans-Ulrich; Linnert, Barry: Distributed dynamic processor allocation for multicomputers, *Parallel Computing*, Volume 33, Issue 3, April 2007, pp. 145-158