Resource Scheduling for Composite Multimedia Objects

Minos N. Garofalakis *(Univ. of Wisconsin – Madison)*

Yannis E. Ioannidis *(Univ. of Athens & Univ. of Wisconsin – Madison)*

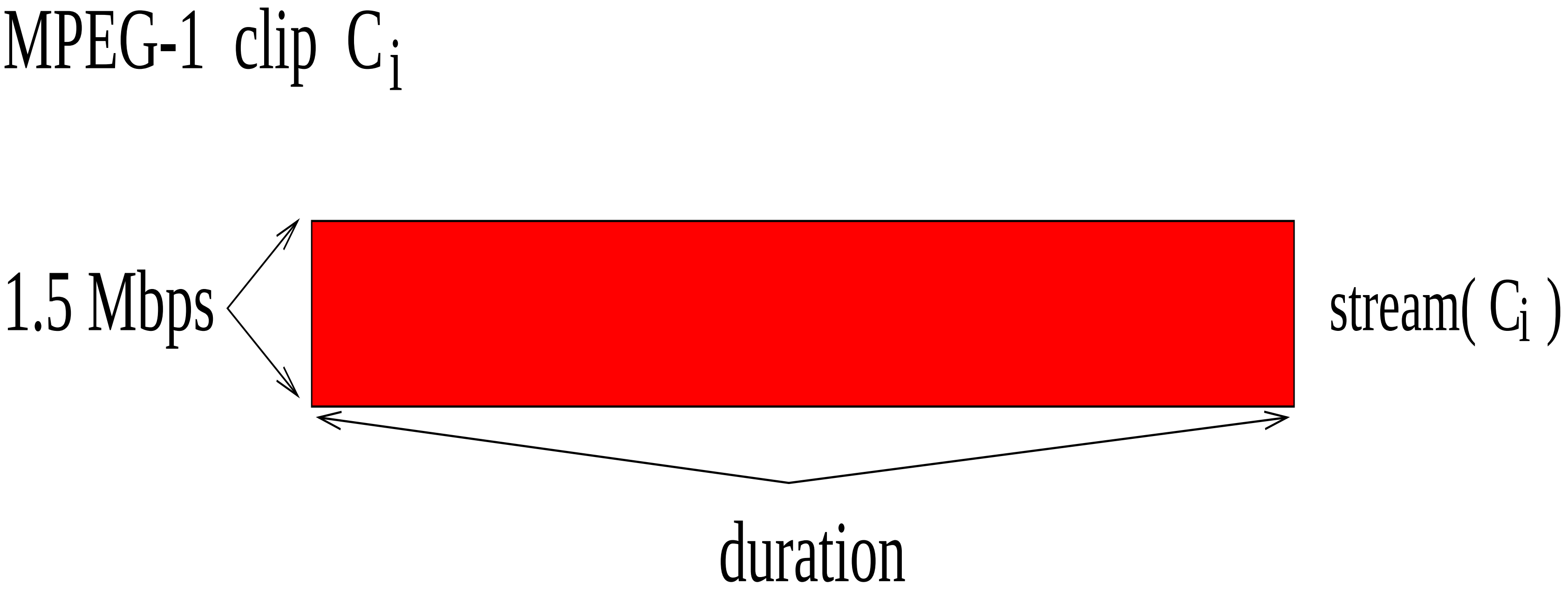
Banu Özden *(Bell Laboratories)*

Outline

- Introduction and Motivation
- Problem Formulation
- Scheduling Algorithms
- Experimental Study
- Conclusions

Introduction and Motivation

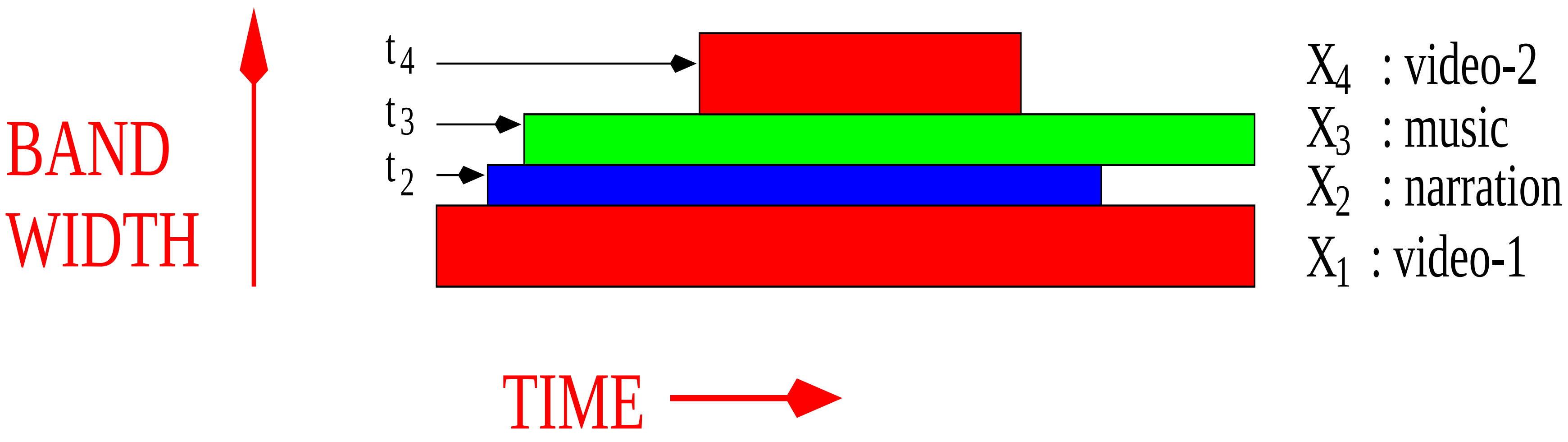
- *Continuous Media*: Real-time resource requirements for delivery



- Objective: Maximize utilization of given bounded resources

Introduction and Motivation (cont.)

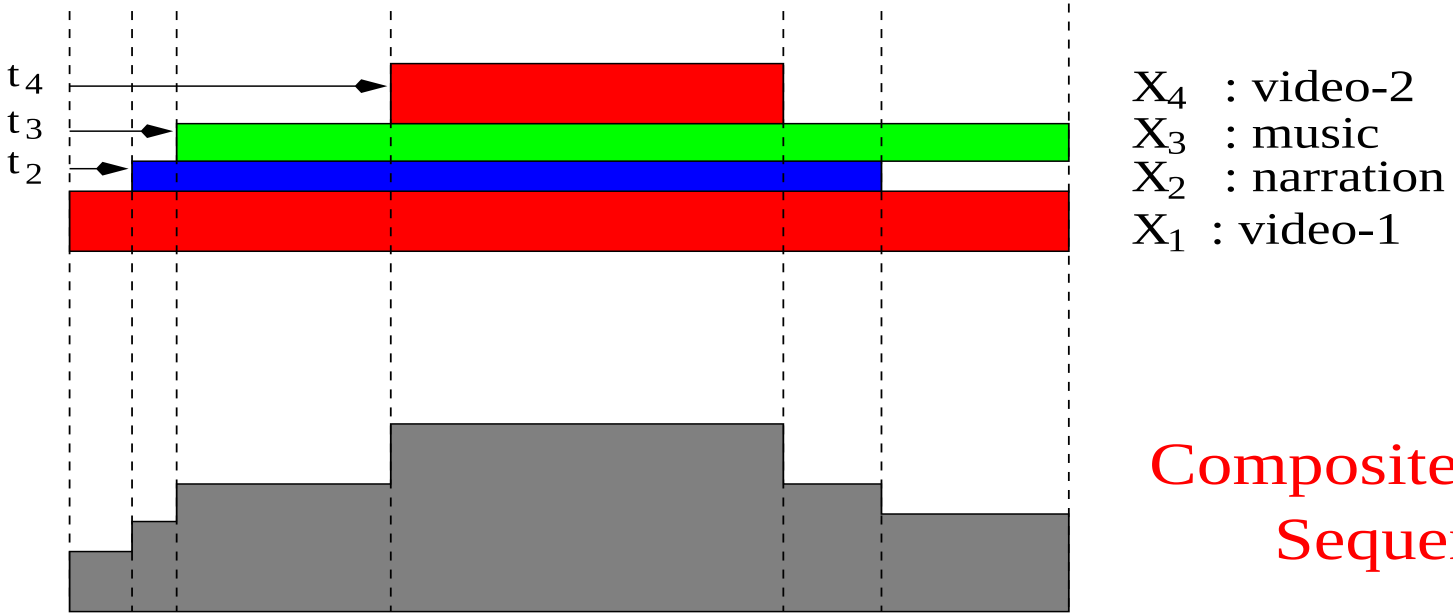
- *Composite Objects*: tailored multimedia presentations, using streams and temporal synchronization primitives



- Scheduling algorithm must preserve *inter-stream* synchronization

Introduction and Motivation (cont.)

Why is it difficult?

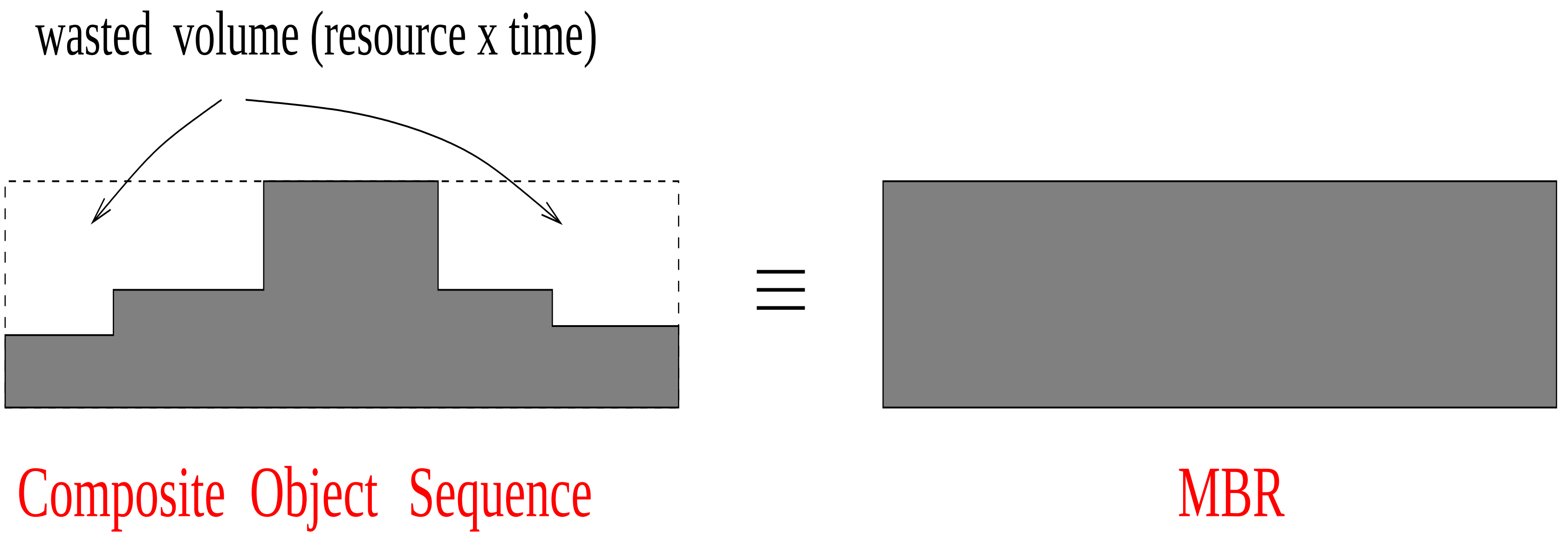


Composite Object Sequence

- Resource-constrained tasks with *time-varying* resource demands
- Problem essentially identical to *Variable Bit Rate (VBR)* stream scheduling

Introduction and Motivation (cont.)

Current approach: reserve resources based on *worst-case demand*, i.e., *MBR* simplification



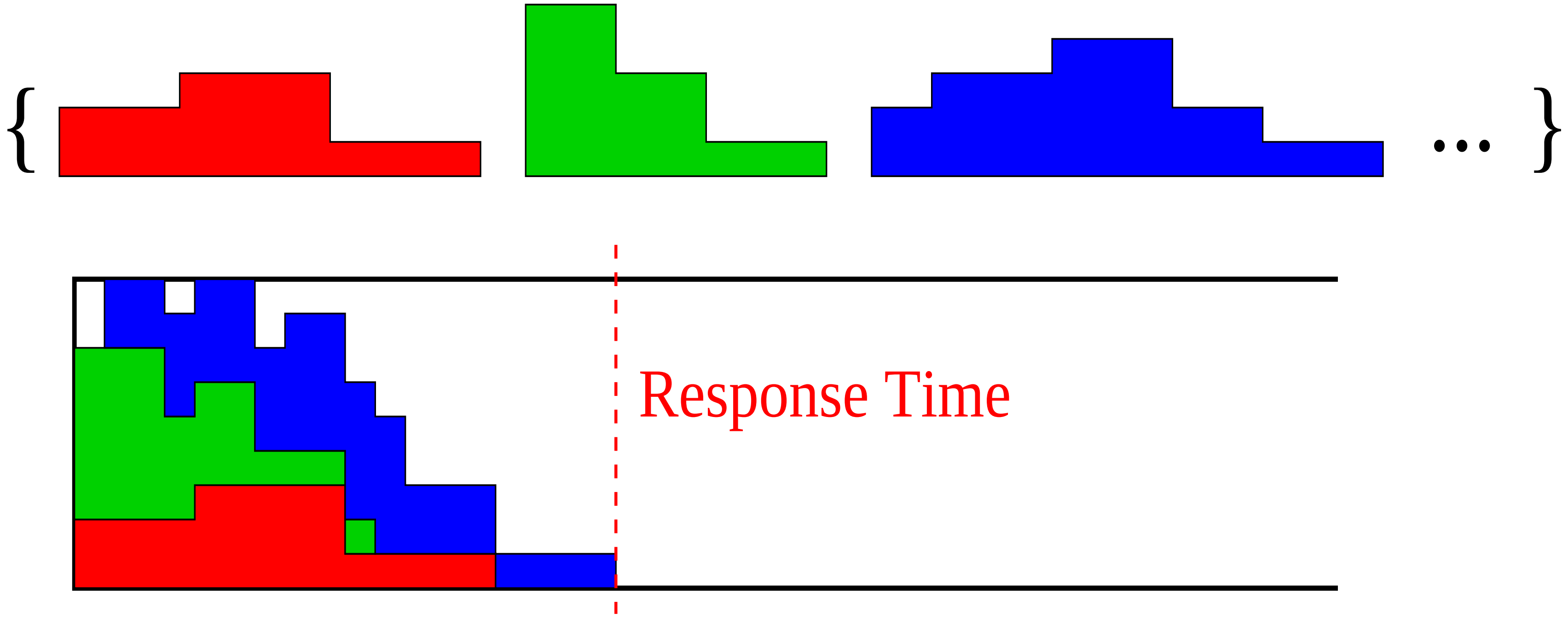
Problems:

- Wasted *volume* (resource x time) can be significant ("sparse" objects)
- "Hides" timing structure and opportunities for clever optimizations

Problem Formulation

Sequence Packing **Problem**

- Given: a set of (normalized) object sequences
- Find: a packing into a unit capacity bin s.t. the length is minimized

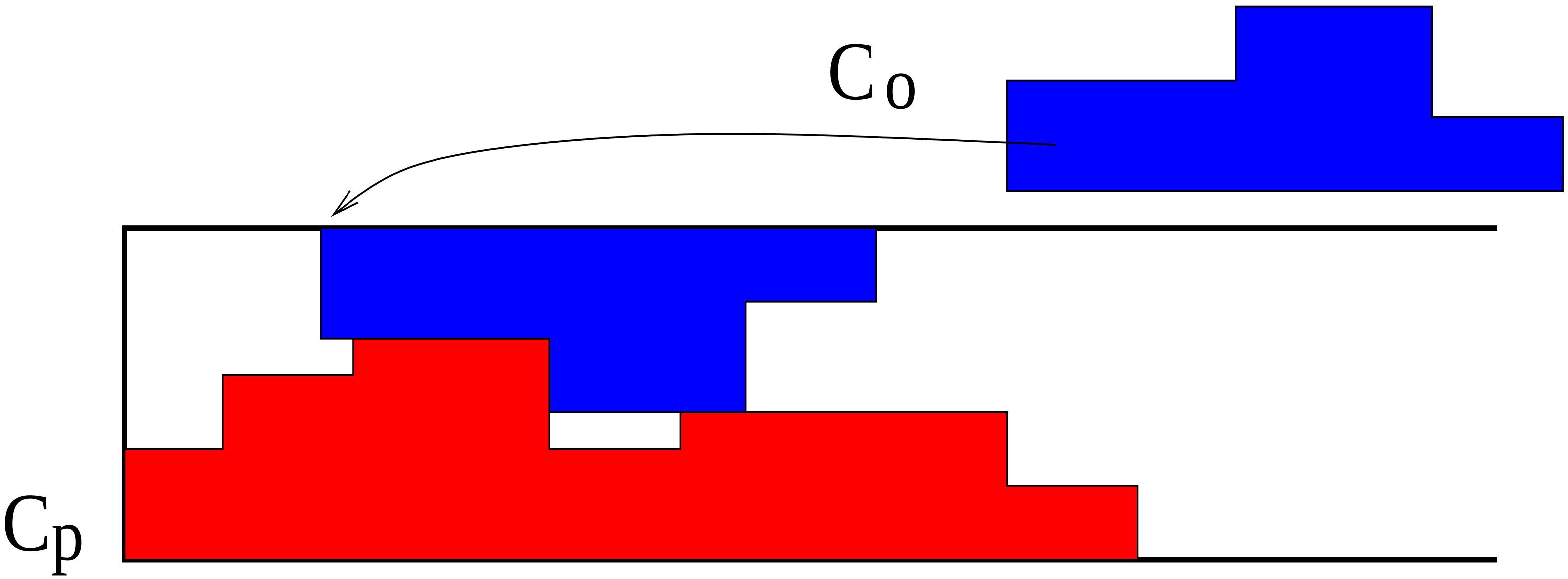


- $\mathcal{NP}$-hard! (generalization of known packing problems)

The Basic Step: Packing Two Sequences

Observation: The packing of any subset of sequences is itself a sequence!

- **Given:** Two sequences C_p (partial schedule) and C_o (new object)
- **Find:** Earliest point to start C_o without violating bin capacity



- Want to do this efficiently!

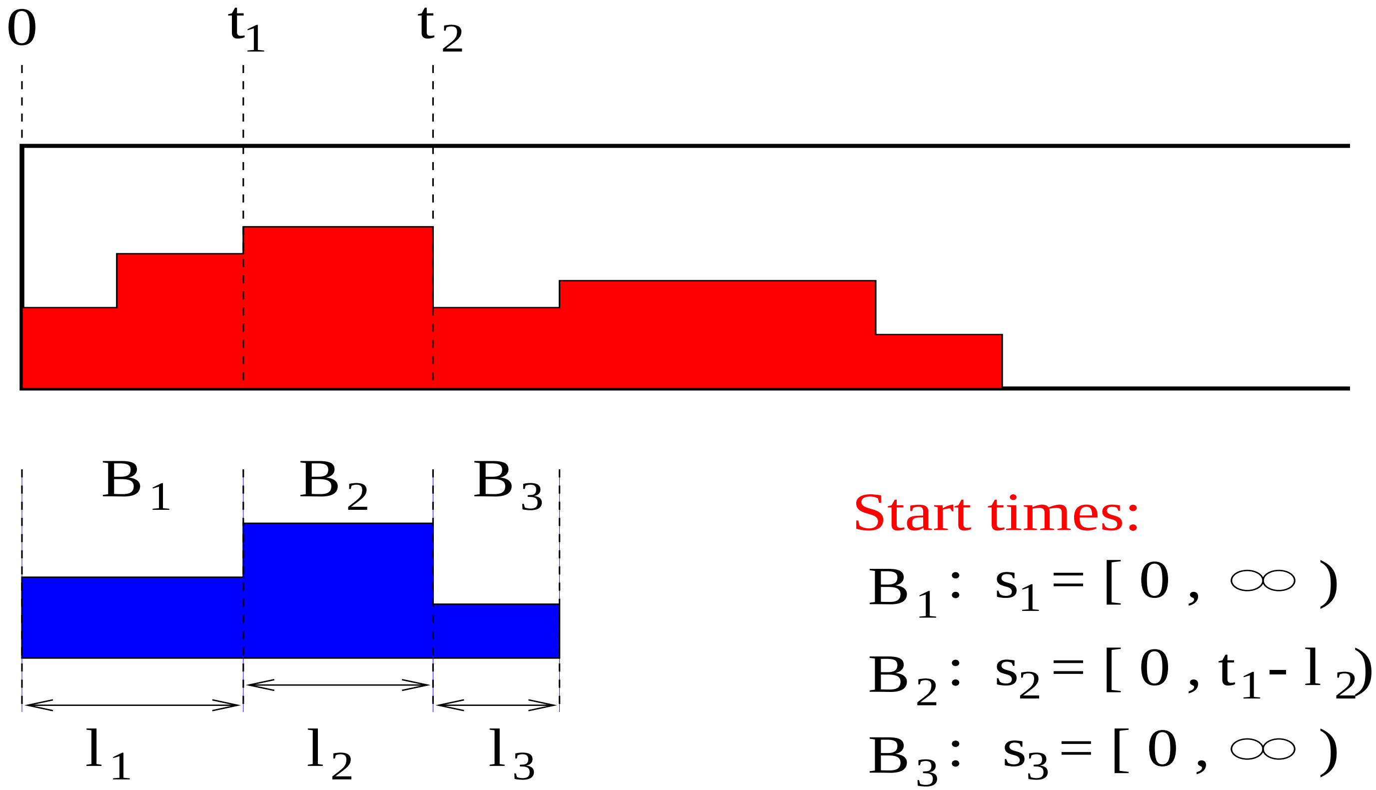
The Basic Step: Packing Two Sequences (cont.)

Algorithm FindMin

1. Divide sequence into *constant demand* blocks
2. Find start time intervals for each block
3. Take intersection of appropriately "shifted" start time intervals

- Complexity: O($n_o \cdot n_p$), n_i = number of component streams in C_i

The FindMin Algorithm



Start times:

B_1: $s_1 = [\,0\,,\ \infty\,)$

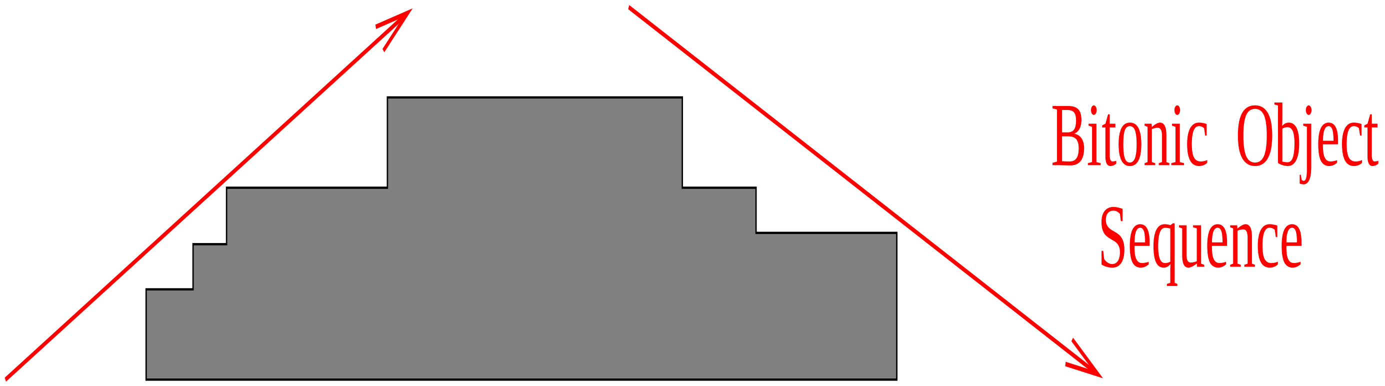
B_2: $s_2 = [\,0\,,\ t_1 - l_2) \ \bigcup\ [\,t_2\,,\ \infty\,)$

B_3: $s_3 = [\,0\,,\ \infty\,)$

SOLUTION : $s_1 \bigcap s_2 - l_1 \bigcap s_3 - (\,l_1 + l_2\,) = \underline{[\,t_2 - l_1\,,\ \infty\,)}$

Can we do better?

- **Results from Pattern Matching:** significant improvements over FindMin (e.g., a $O(n_o + n_p\ )$ algorithm) are *highly unlikely*
- Improved version of FindMin ($O(\ n_o + n_p \log n_o)$) based on Knuth-Morris-Pratt string matching, when C_o is *bitonic*



A List-Scheduling Algorithm for Sequence Packing

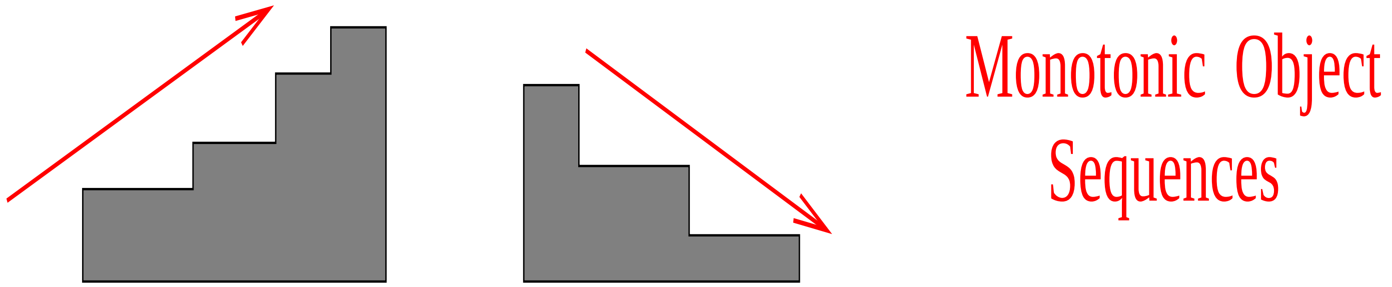
Algorithm $\mathcal{LS}$: Let L be the list of objects to schedule.

For each object C in L:

Place C at earliest possible start point using FindMin

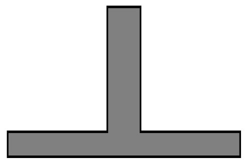
- Complexity: $O(N^2)$, N = total number of streams (for general objects)

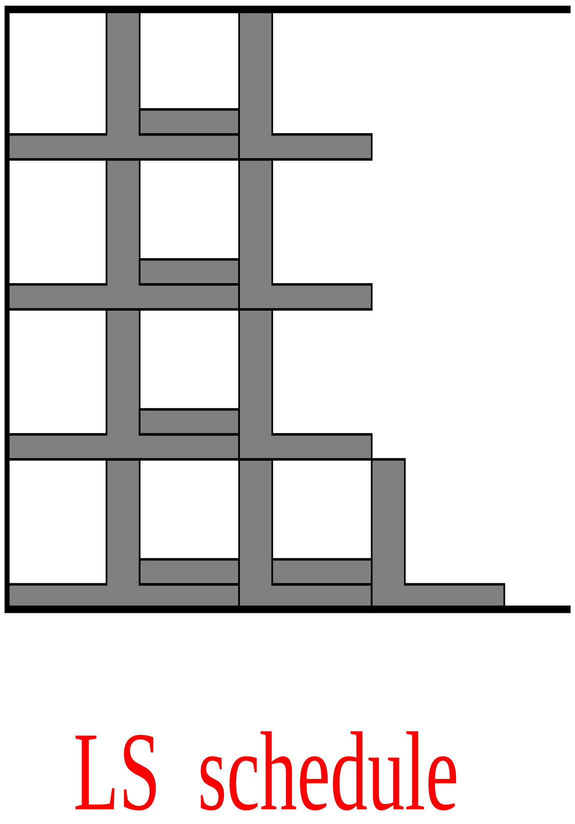
Theorem: $\mathcal{LS}$ is *near-optimal* for scheduling a collection of *monotonic* object sequences

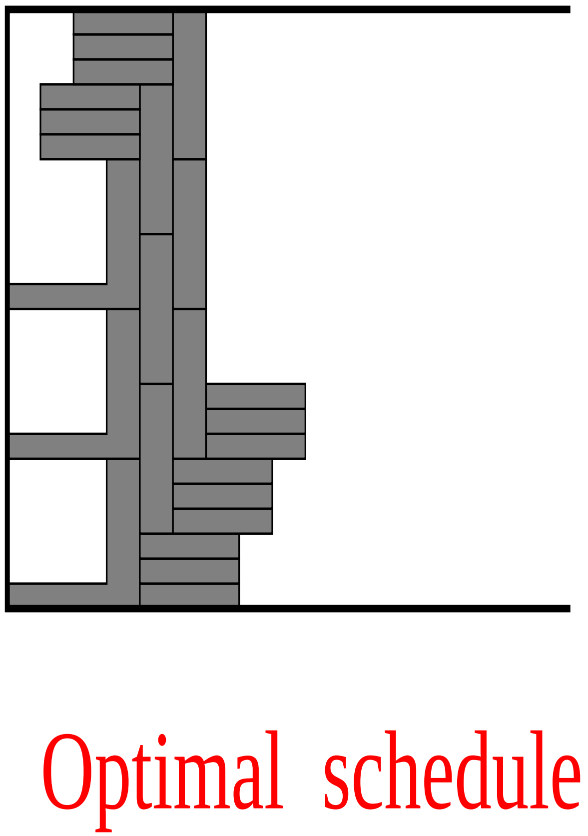


A Bad Example for $\mathcal{LS}$

all objects:



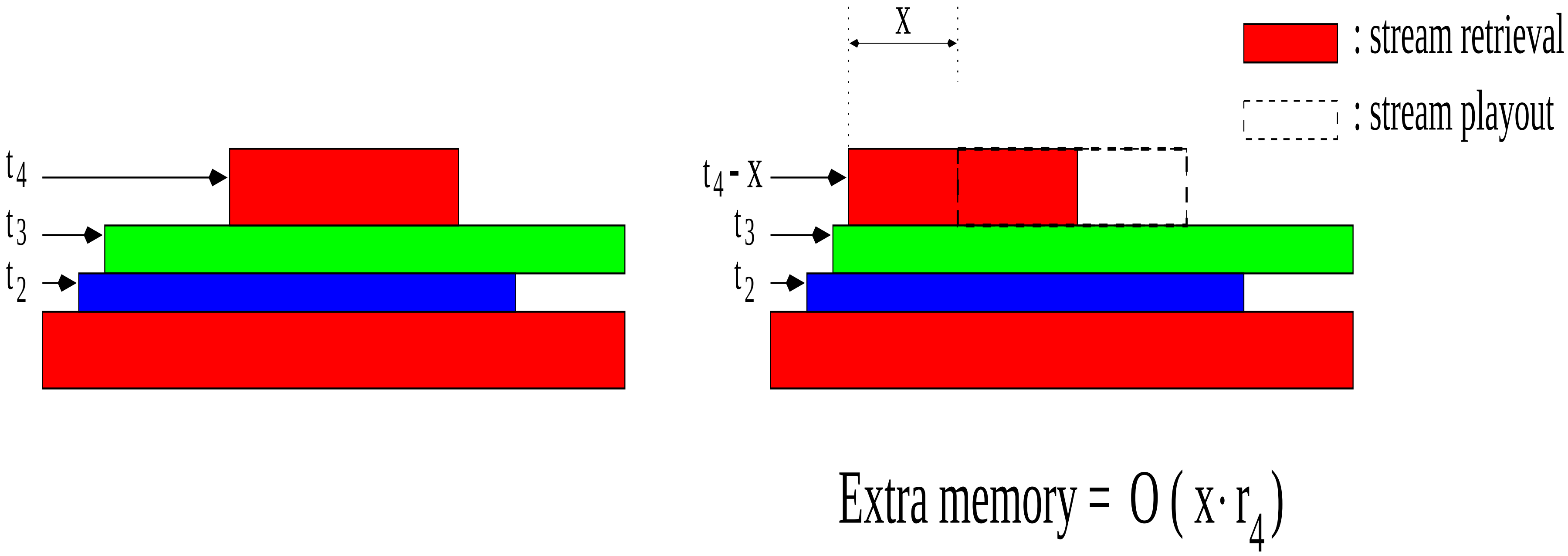




- Worst-case performance of $\mathcal{LS}$ can be arbitrarily bad, even with identical, bitonic objects

Improvements to Base $\mathcal{LS}$: Stream Sliding

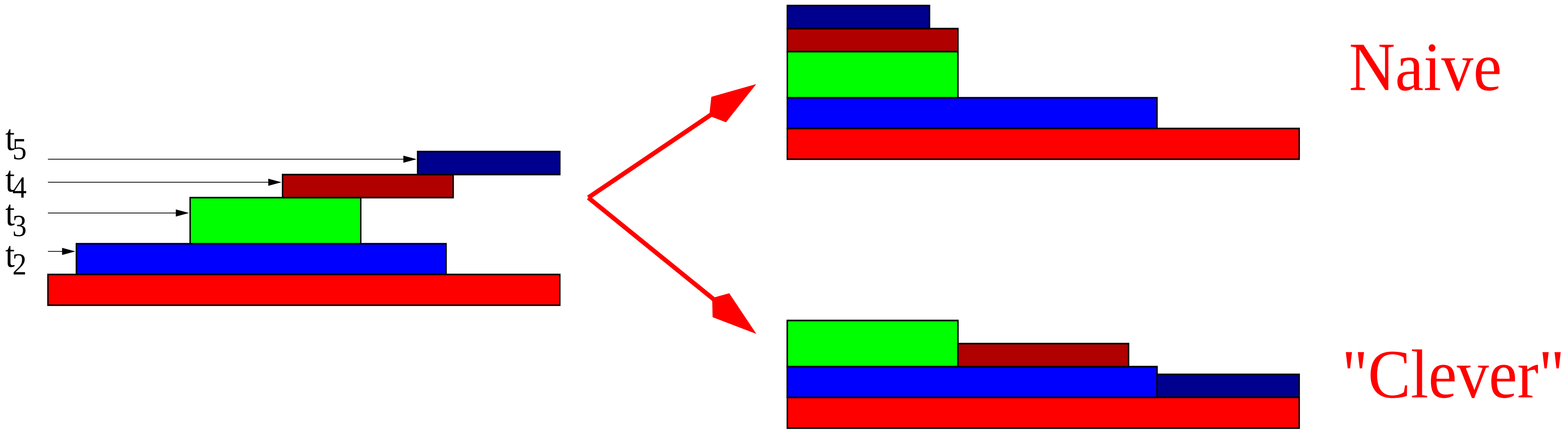
- *Stream sliding* [Chaudhuri et al., VLDB'95]: use extra memory to alter an object sequence



- Complicates scheduling problem significantly!

Improvements to Base $\mathcal{LS}$: Stream Sliding (cont.)

Idea: "clever" stream sliding to obtain monotonic shapes



- General problem is $\mathcal{NP}$-hard
- Heuristic based on *Weighted Graph Matching*

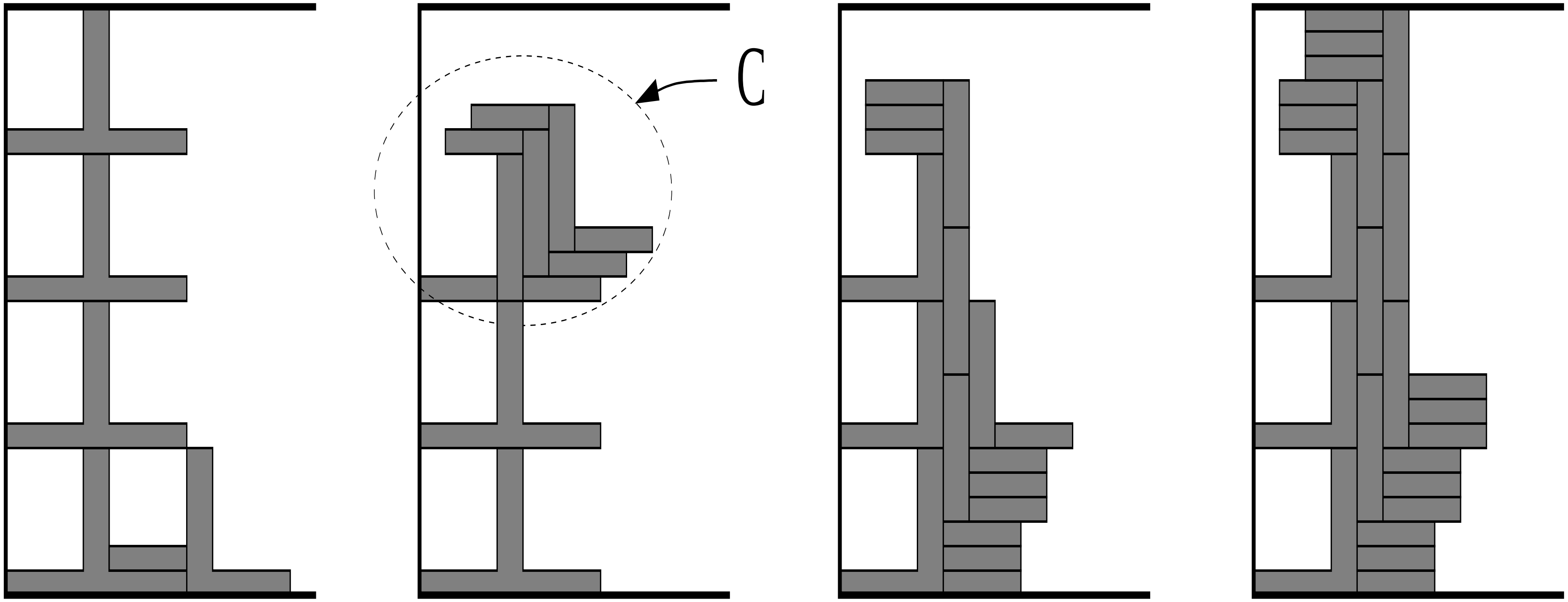
Improvements to Base $\mathcal{LS}$: Local Backtracking

Idea: try to improve schedule "density" using small local perturbations

$\mathcal{LSB}(k)$**:** List-scheduling with k-object backtracking

- Complexity: $\mathrm{O}(|L| \cdot N^2)$

Improvements to Base $\mathcal{LS}$: Local Backtracking (cont.)



LSB(3) gives optimal schedule for our bad example

Experimental Study

Goal: average performance of $\mathcal{LS}$ vs.

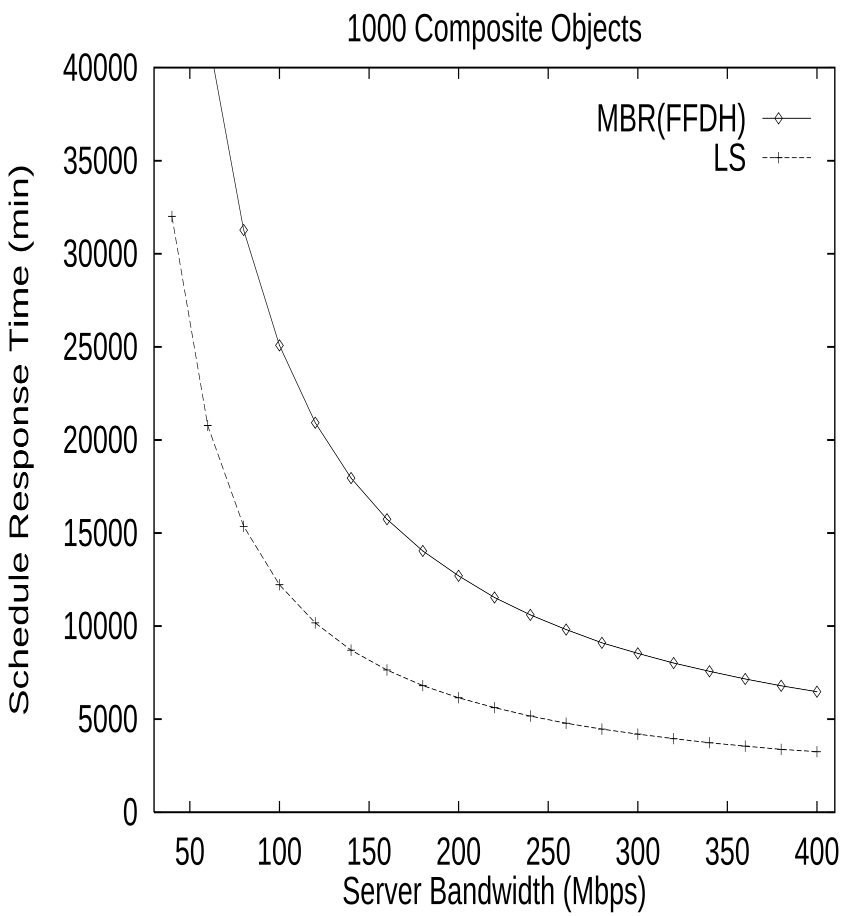
1. MBR-based solution (FFDH rectangle packing)
2. Lower bound on optimal solution

Testbed:

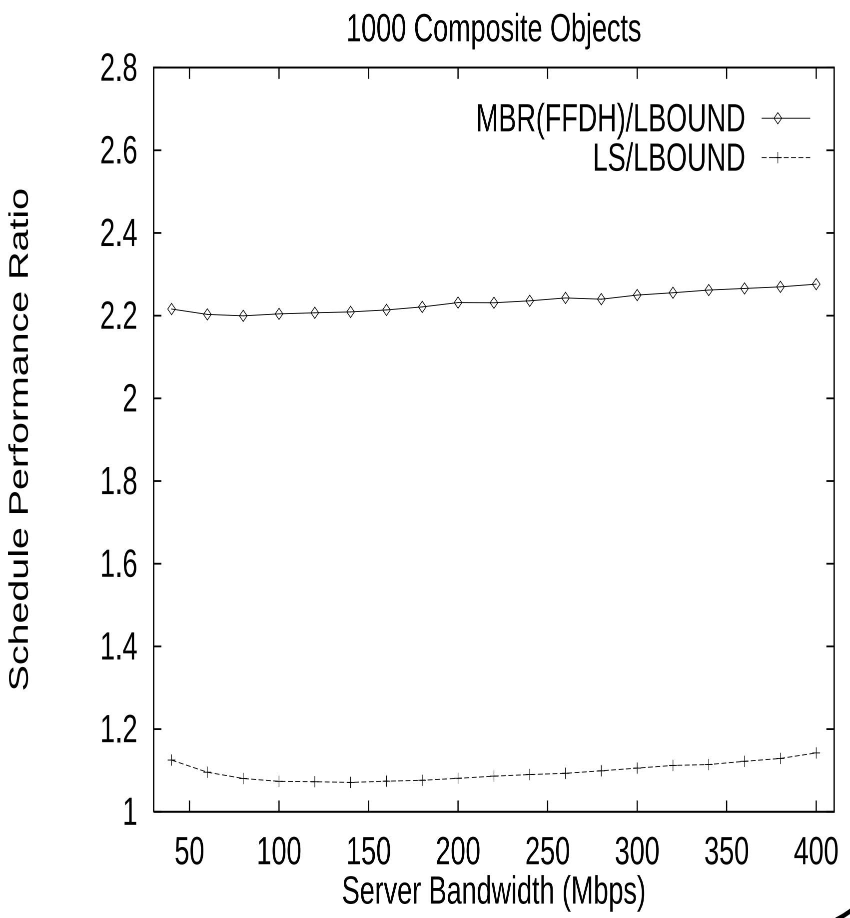
- Set of randomly generated composite objects
- Average response times & performance ratio over 10 runs

Experimental Results

- Avg. Response Times (1000 objs)

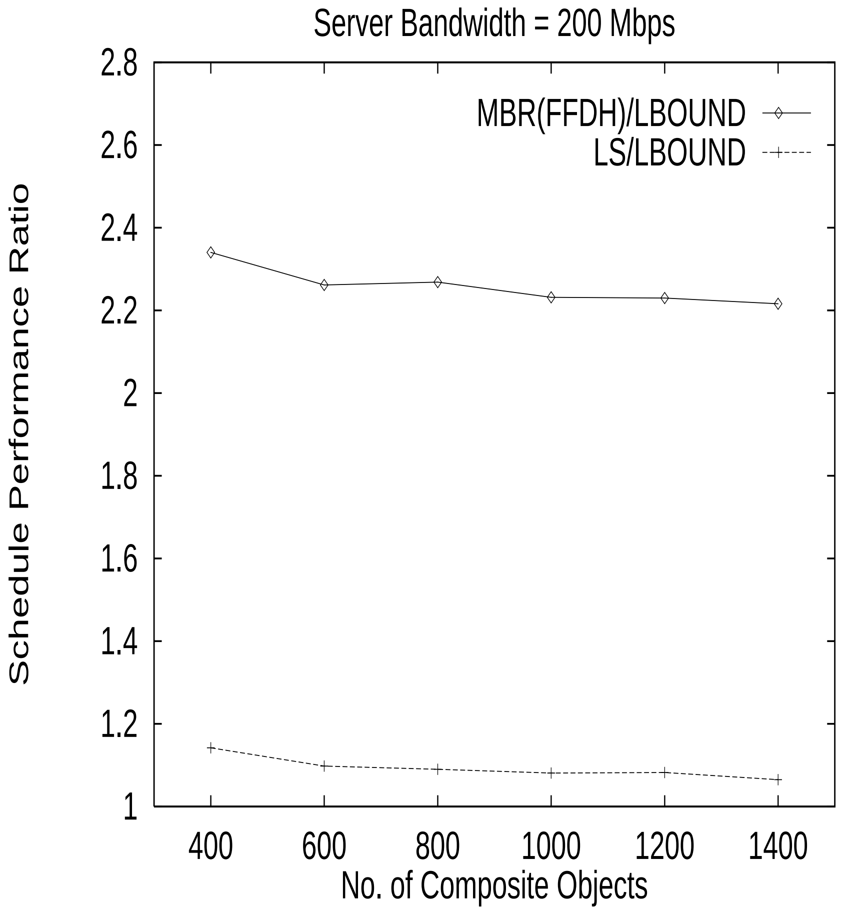


- Avg. Perf. Ratios (1000 objs)



Experimental Results (cont.)

- Avg. Performance Ratios (200 Mbps)



- Running Times

Time to schedule 1000 objects (in sec)

Bandwidth	MBR(FFDH)	LS
100	0.016	9.041
200	0.012	6.233
300	0.010	4.709
400	0.009	3.760

Conclusions and Future Work

- Composite multimedia object scheduling based on *sequence packing* formulation
- Efficient "basic step" algorithms for packing two sequences
- List-scheduling algorithm ($\mathcal{LS}$): *near-optimal* for monotonic objects
- Improvements on base $\mathcal{LS}$ algorithm (sliding, backtracking)
- Experimental study that verifies effectiveness of $\mathcal{LS}$

Future work:

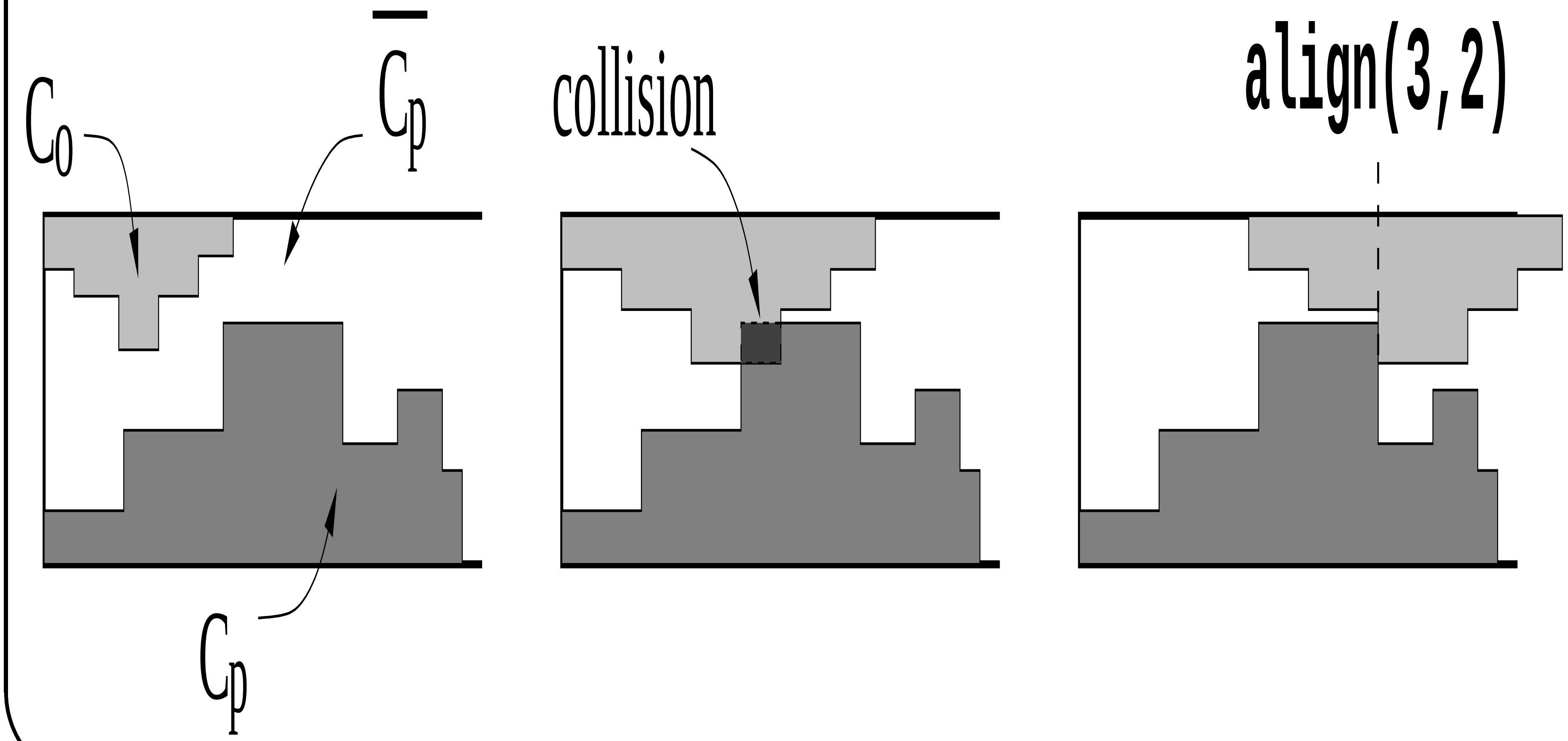
- *Stream sharing*: multimedia presentations sharing common streams
- Scheduling with memory and flexible QoS

Related Work

- A lot of work on *modeling* temporal synchronization (e.g., petri nets, interval models)
- Chaudhuri, Shahabi, and Ghandeharizadeh [VLDB'95, TKDE (to appear)] focus on:
  1. Using memory (sliding) to resolve "internal contention" (for a single object)
  2. Heuristic memory allocation policies for multiple *binary* objects

Algorithm BITONIC-FINDMIN

• For *bitonic* C_o, we never need to backtrack as we shift C_o over C_p (similar to KMP string matching)



Experimental Parameter Settings

Experimental Parameter	Value
Aggregate Server Disk Bandwidth (in Megabits per sec – Mbps)	40Mbps – 400Mbps
Number of Composite Objects	400 – 1400
Number of Streams per Object	1 – 8
Set of Possible Stream Lengths (in min)	{ 10 , 20 , 30, 60, 90, 120 , 180, 240, 300 }
Set of Possible Stream Bandwidth Requirements (in Mbps)	{ 0.0625 , 0.125 , 1 , 1.5 , 2 , 3 , 4 , 5 }