Load Balancing in Indexing for Content-based Video Search on Peer-to-Peer Networks

นายชัยยุทธ ประดิษฐ์ทองงาม

Faculty of Engineering, Chiang Mai University, Thailand

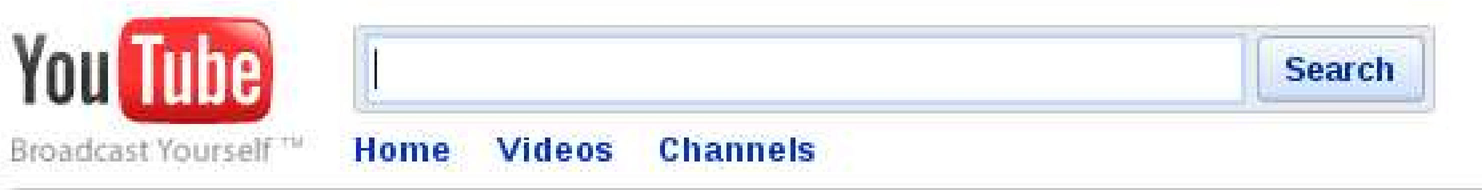
chaiyut@grad.cmu.ac.th

Outline

- Introduction
- Problem
- Solution
- Experimental
- Conclusion

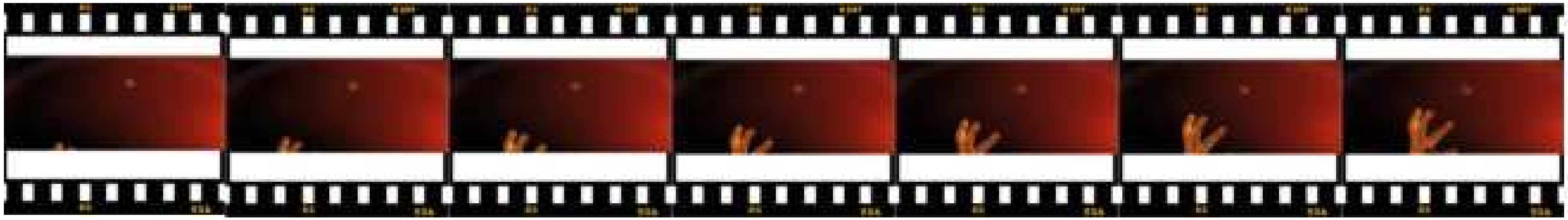
Introduction

- Video Search

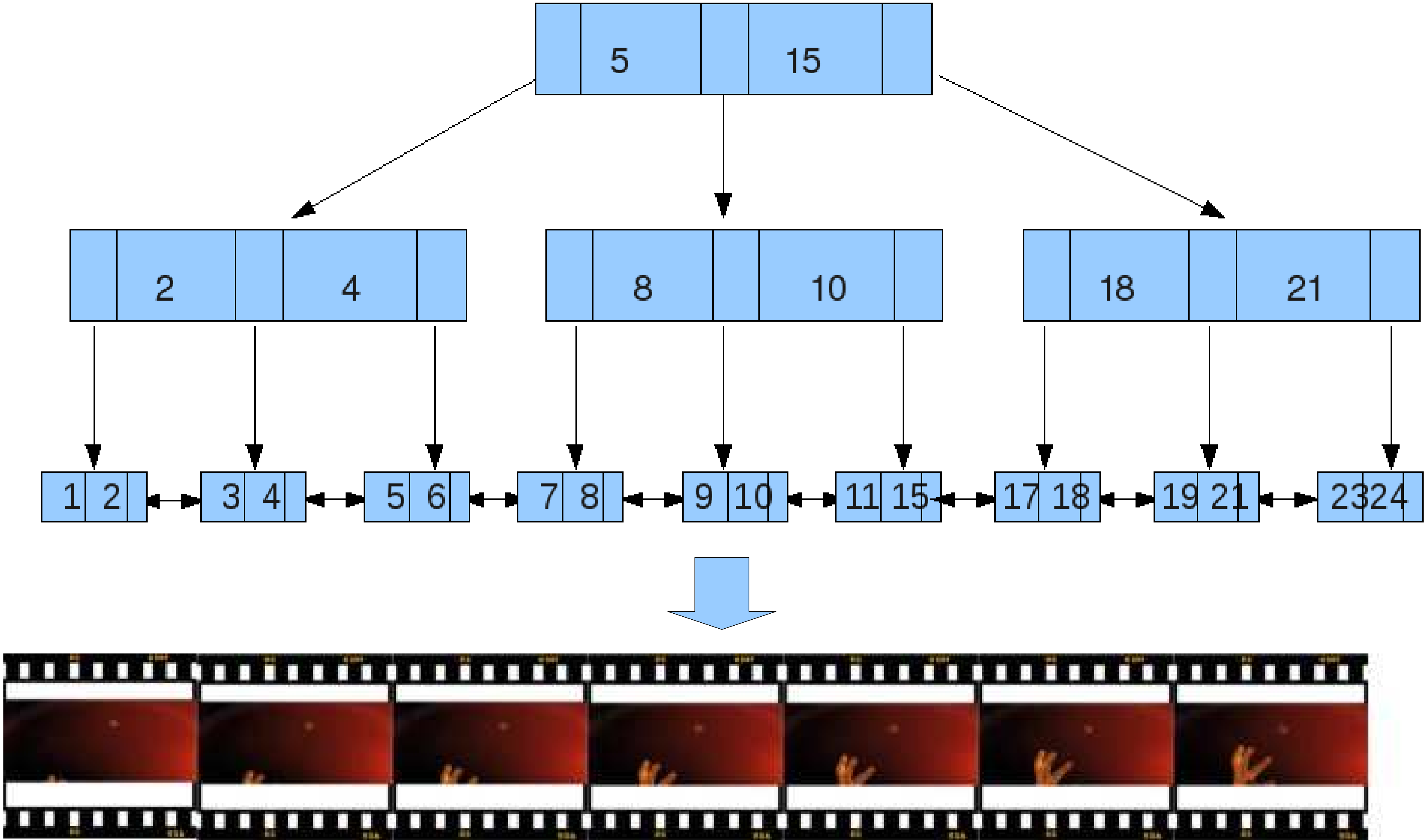


Introduction

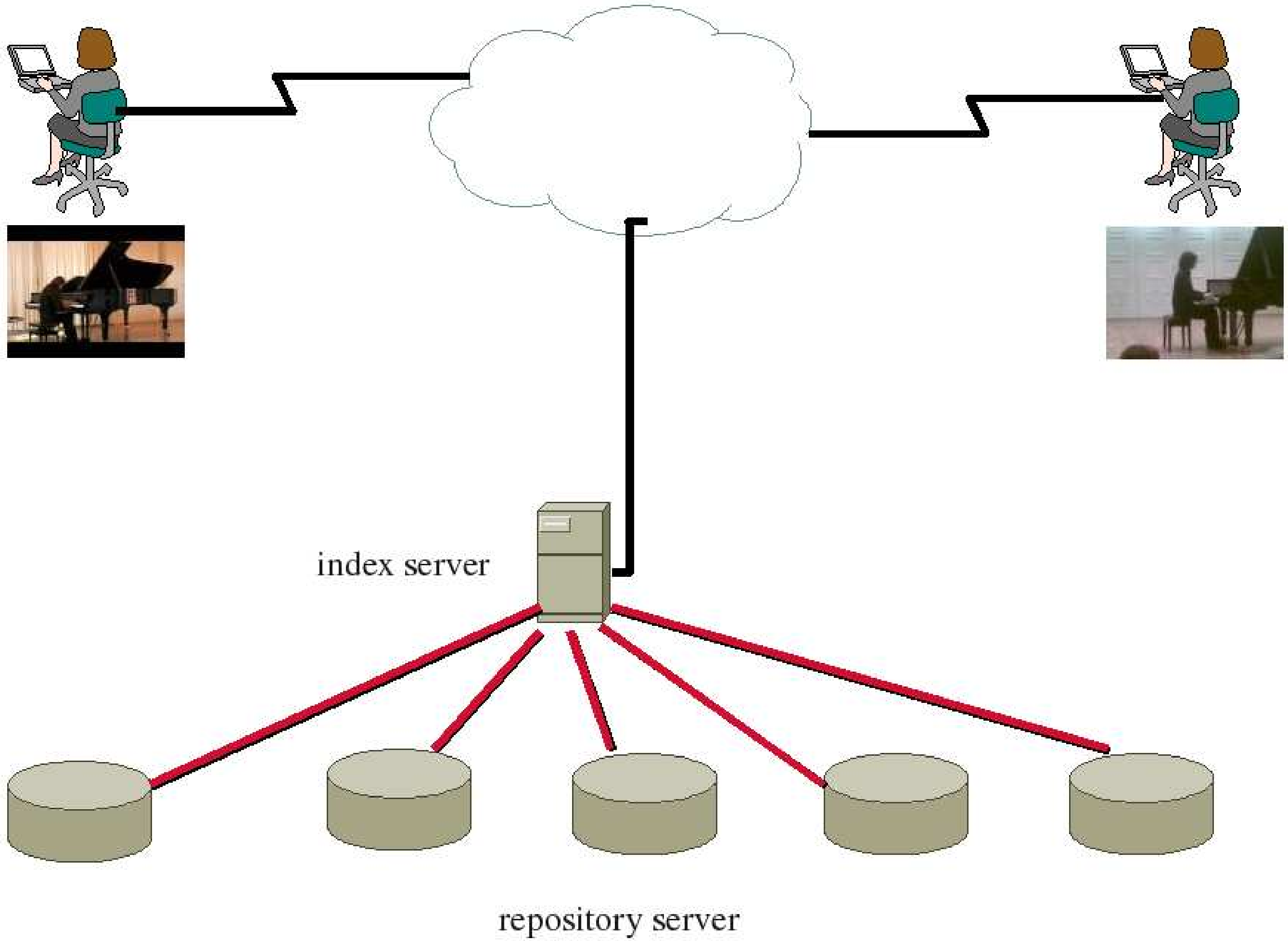
- Video Search



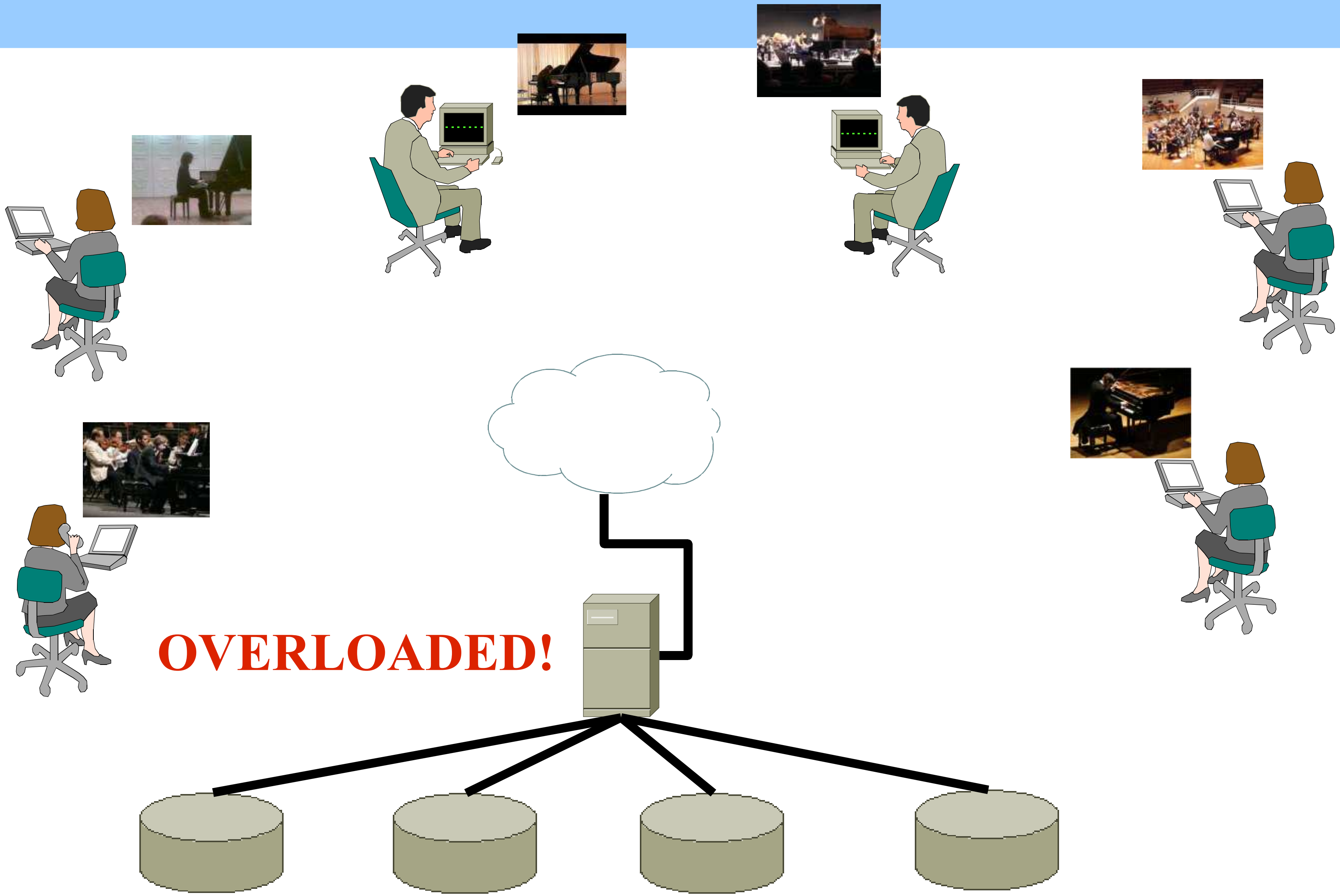
Introduction



Introduction

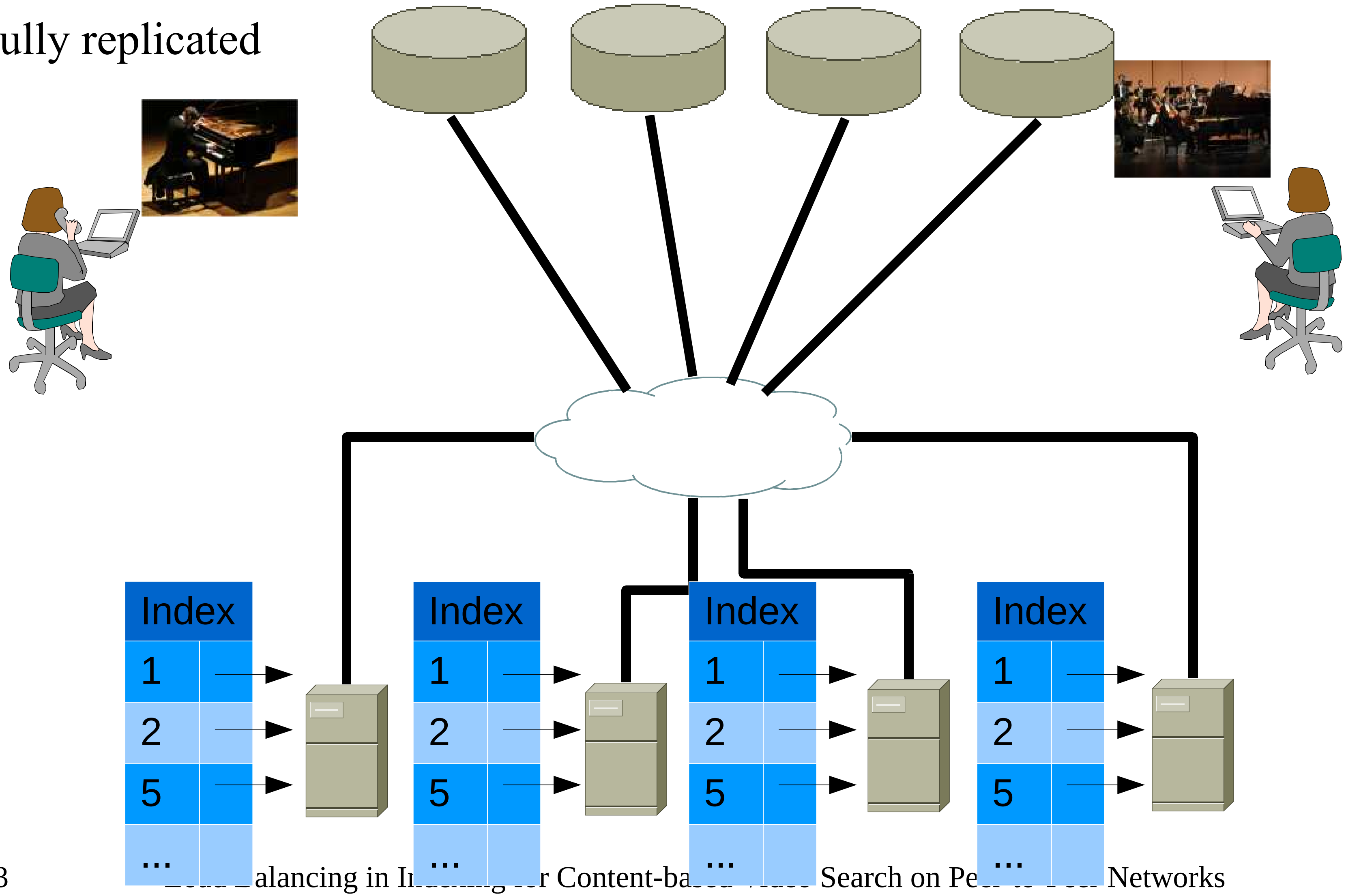


Problem

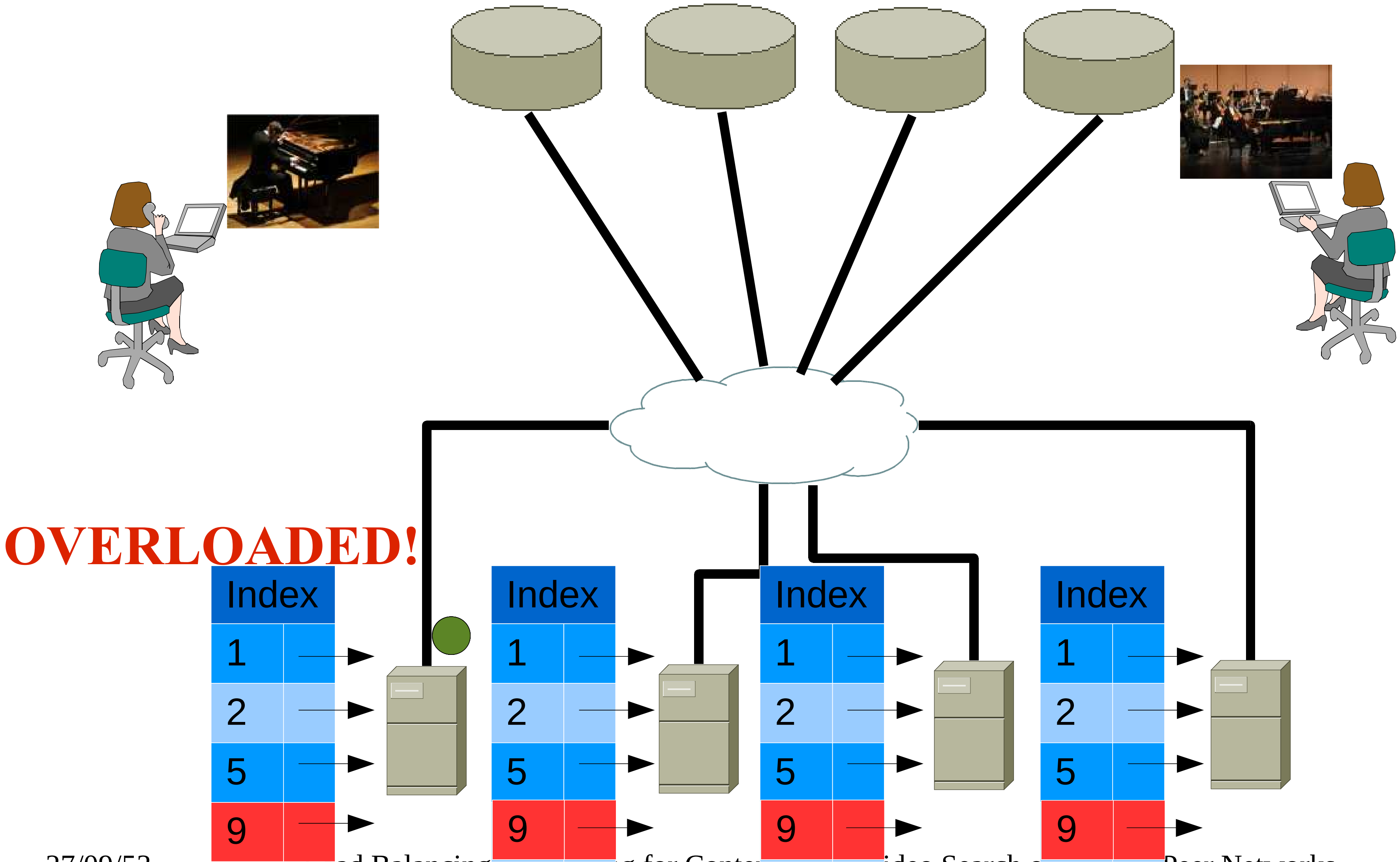


Problem

- Fully replicated

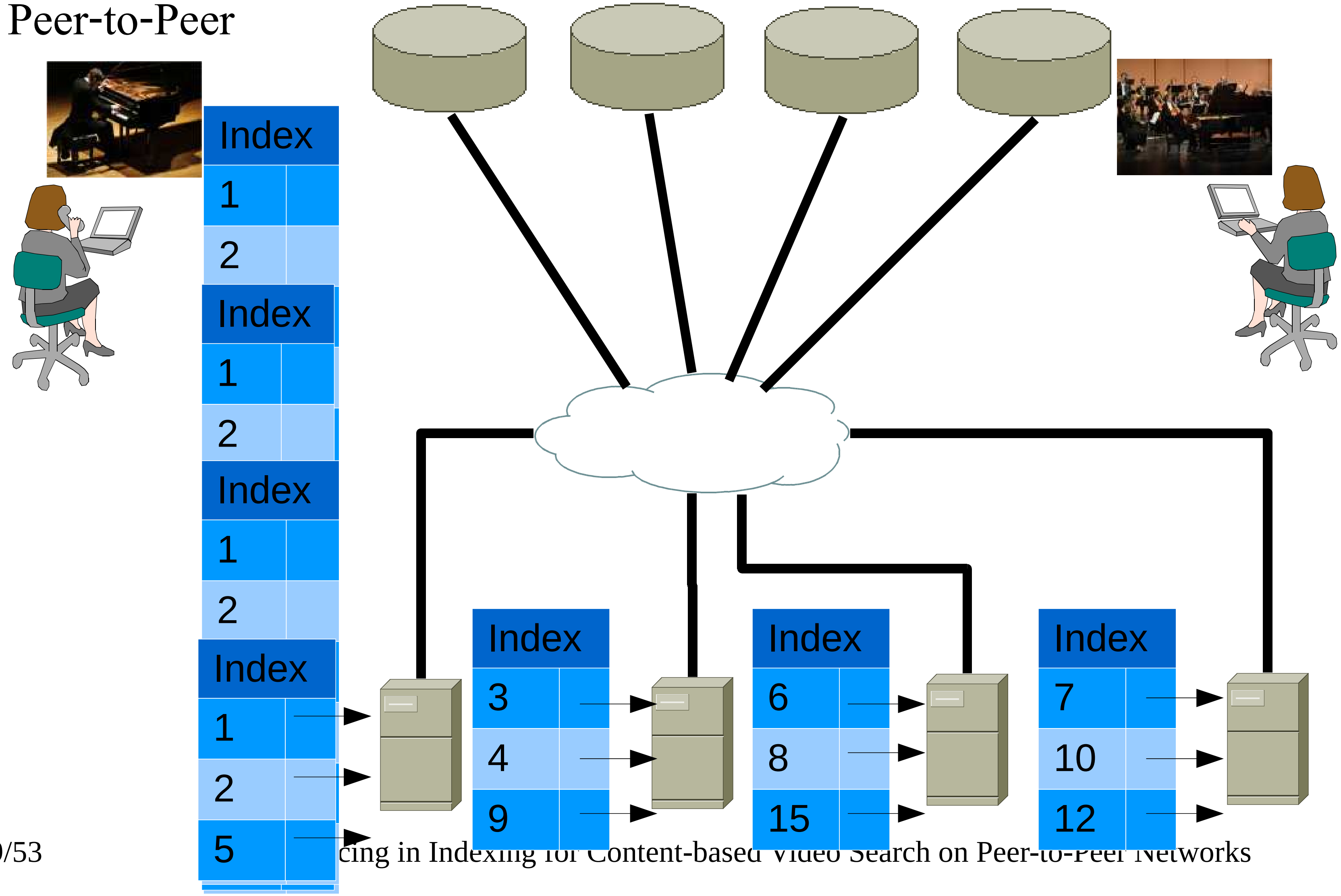


Problem



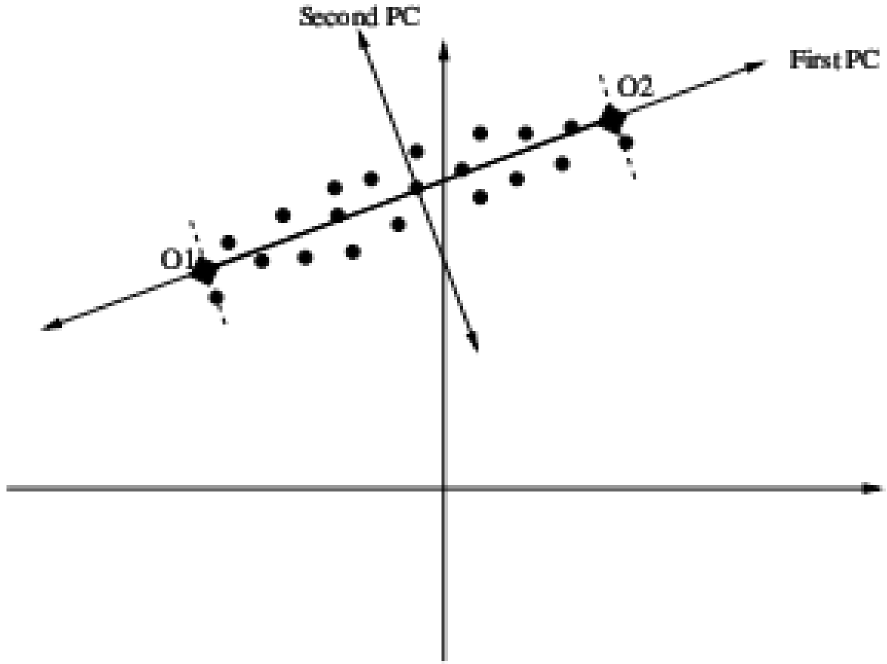
Solution

- Peer-to-Peer



Solution

- Index construction
  - Extract video frame
  - Extract feature each frame
  - Find cluster centroid by K-Means
  - Find optimal reference point by PCA
  - Map high dimension to one dimension for B^+-Tree by distance from cluster centroid to optimal reference point



Solution

- Query
  - Extract video frame (query)
  - Extract feature
  - Find key x by distance from feature to optimal referance point
  - Search between $| x - \varepsilon |$ and $| x + \varepsilon |$

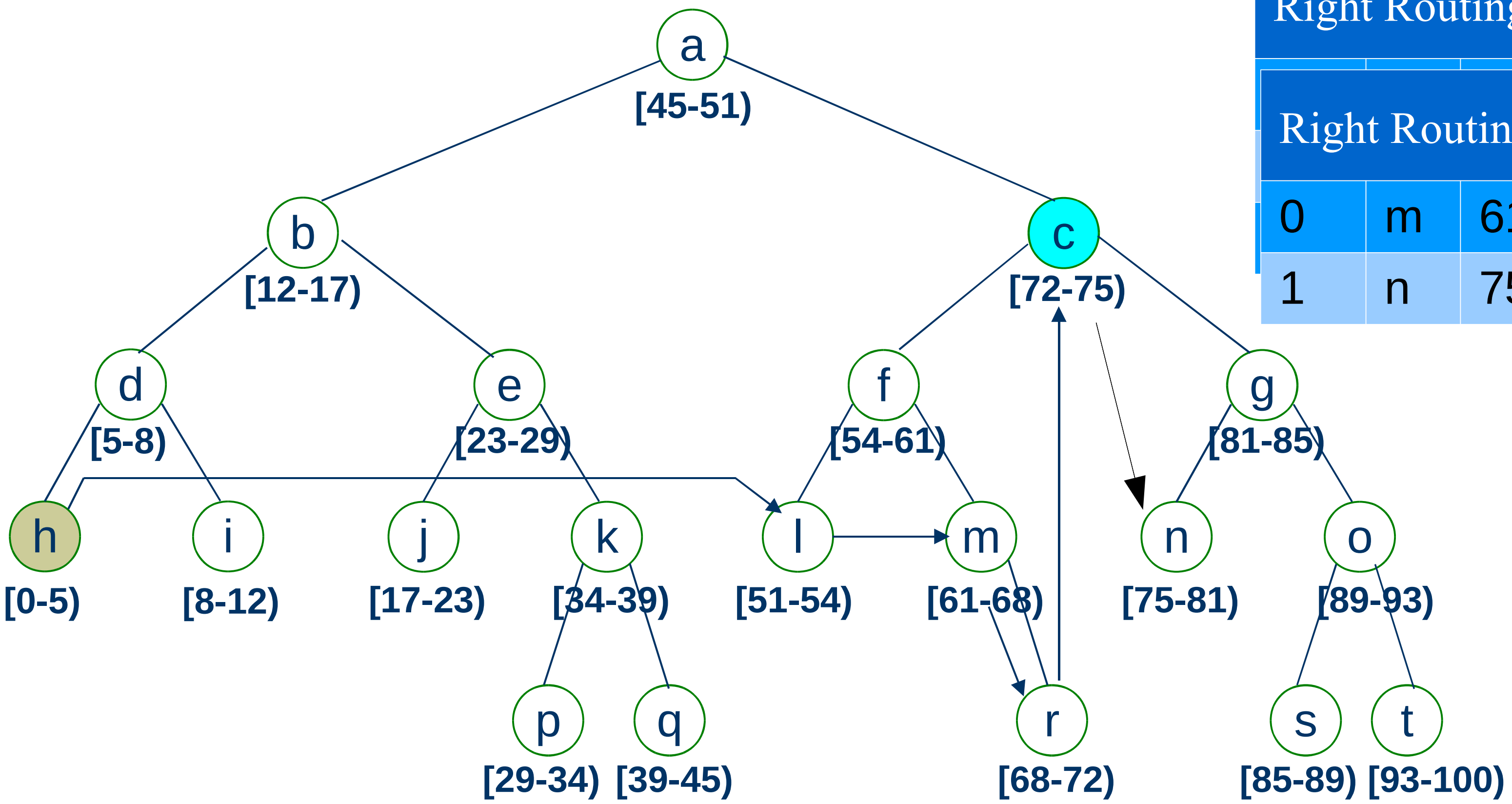
⇩

Range Query

Solution

- BATON support range query, Ex. query 74 - 80

Right Routingtable		
0	m	61 - 68
1	n	75 - 81

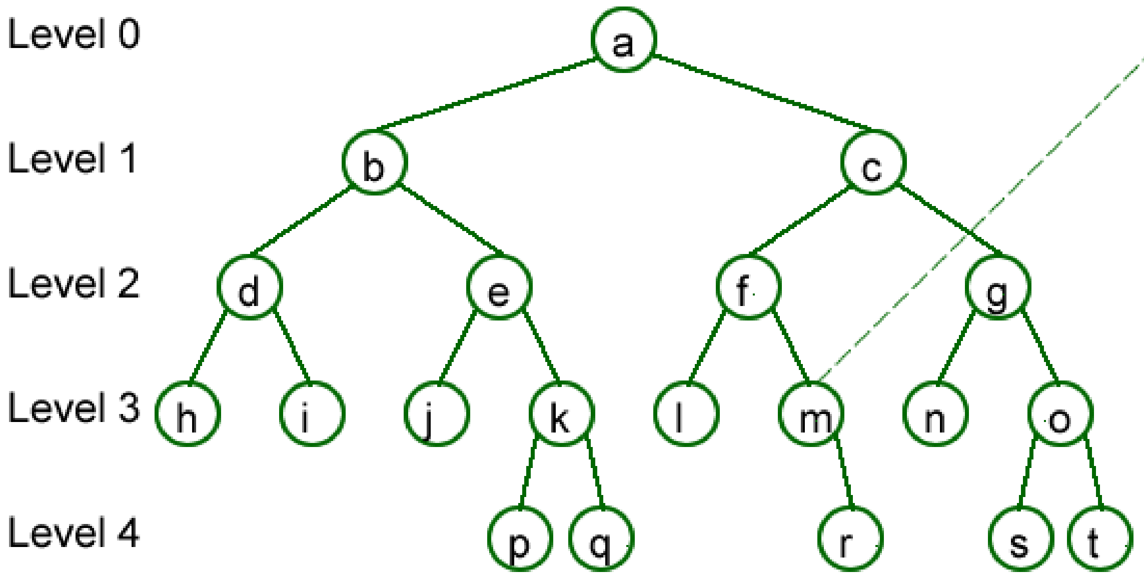


h d i b j e p k q a l f m r c n g s o t

Solution

- Binary balanced tree structure

Definition : tree is balanced if and only if at any node in the tree, the height of its two subtrees differ by at most one.



Node m: level=3, number=6
parent=f, leftchild=null, rightchild=r
leftadjacent=f, rightadjacent=r
Left routing table:

	Node	Left child	Right child	Lower bound	Upper bound
0	l	null	null	l_{lower}	l_{upper}
1	k	p	q	k_{lower}	k_{upper}
2	i	null	null	i_{lower}	i_{upper}

Right routing table:

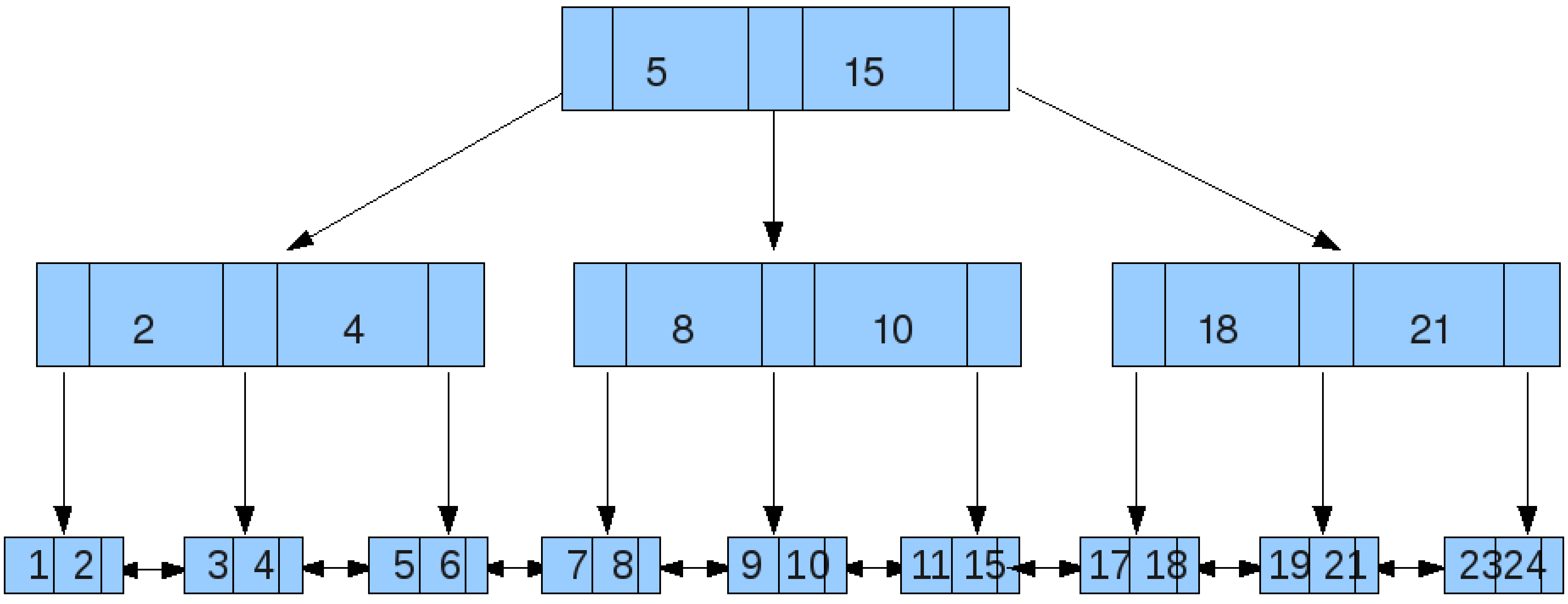
	Node	Left child	Right child	Lower bound	Upper bound
0	n	null	null	n_{lower}	n_{upper}
1	o	s	t	o_{lower}	o_{upper}

Solution

- Index Construction
  - Extract video frame
  - Extract feature each frame
  - Find cluster centroid by K-Means
  - Find optimal reference point by PCA
  - **Map high dimension to one dimension for B^+-Tree by distance from cluster centroid to optimal reference point**

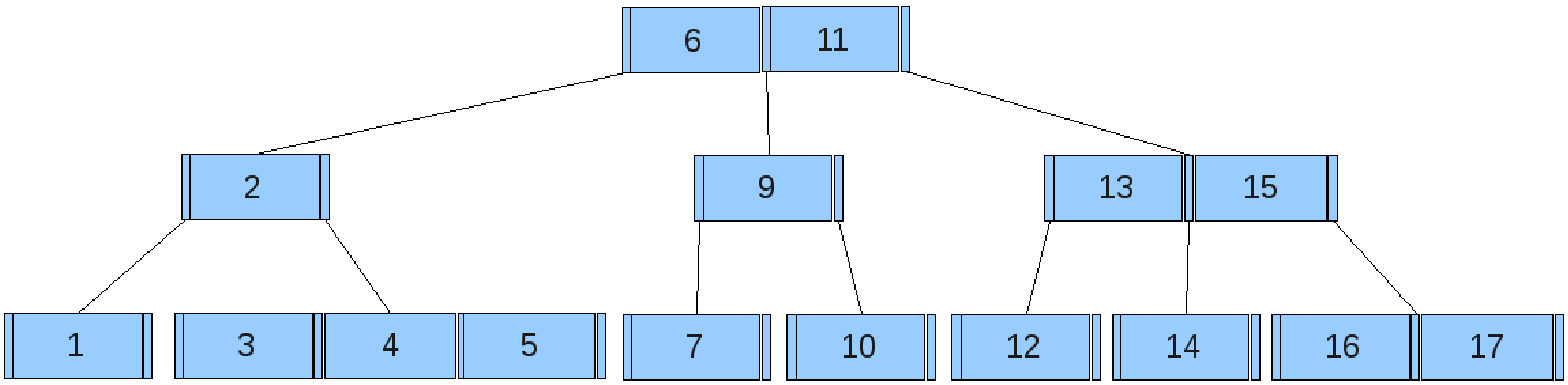
Solution

- B^{+}-Tree



Solution

- B-Tree



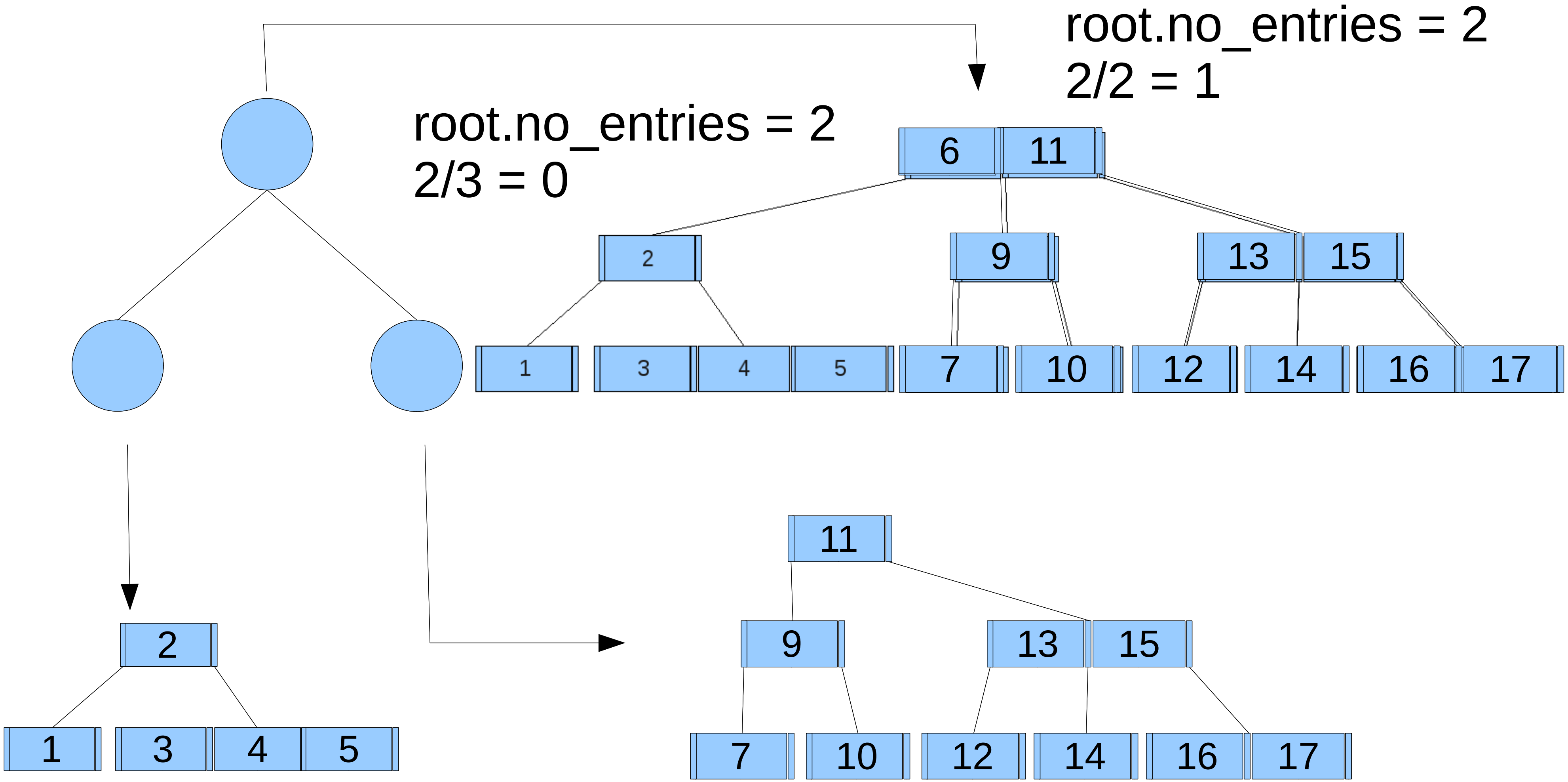
BATON Insert Algorithm

1. Let new_node be the joining node.
2. Let host_node be the node which new_node is to be joined to.
3. **If** isFull(left_routing_table(host_node)) and isFull(right_routing_table(host_node)) and ((host_node.left_child = NULL) or (host_node.right_child = NULL)) **Then,**
4. accept new_node as a child of host_node and split the B-tree structure.
5. **Else**
6. **If** (NOT isFull(left_routing_table(host_node))) or (NOT isFull(right_routing_table(host_node))) **Then,**
7. forward the join request of new_node to the parent node of host_node.
8. **Else**
9. let m be a node in left_routing_table(host_node) or right_routing_table(host_node) not having enough children.
10. **If** there exist m **Then,**
11. forward the join request of m.

Split B-Tree

1. Let t be the B-tree to be split.
2. Let new_node be the joining node.
3. **If** the new_node is to be joined as a left child **Then**,
4.   **If** t.root.left_child = NULL **Then**,
5.   rebuild index t.
6.   **End If**
7.   **If** Integer_part(t.root.no_entries/3) = 0 **Then**,
8.   assign the new B-tree rooted at t.root.left_child to new_node.
9.   **Else**
10.   assign the new B-tree rooted at t.root.entries[Integer_part(t.root.no_entries/3)] to new_node.
11.   **End If**
12. **Else**
13.   **If** t.root.right_child = NULL **Then**,
14.   rebuild index t.
15.   **End If**

Insert Operation



BATON Departure Algorithm

1. Let d_node be the departing node.
2. **If** (d_node.left_child = NULL and d_node.right_child = NULL and
   NOT exists neighbor_node of d_node in left_routing_table(d_node)
   or right_routing_table(d_node) having child) **Then**,
3. **If** NOT exists neighbor_node of d_node in left_routing_table(d_node)
   or right_routing_table(d_node) having child) **Then**,
4. transfer the B-tree structure of d_node to be merge with its parent node
   and depart the network.
5. **Else**
6. find replacement node from its child nodes.
7. **End If**
8. **Else**
9. find replacement node fromm its child nodes.
10. **End If**

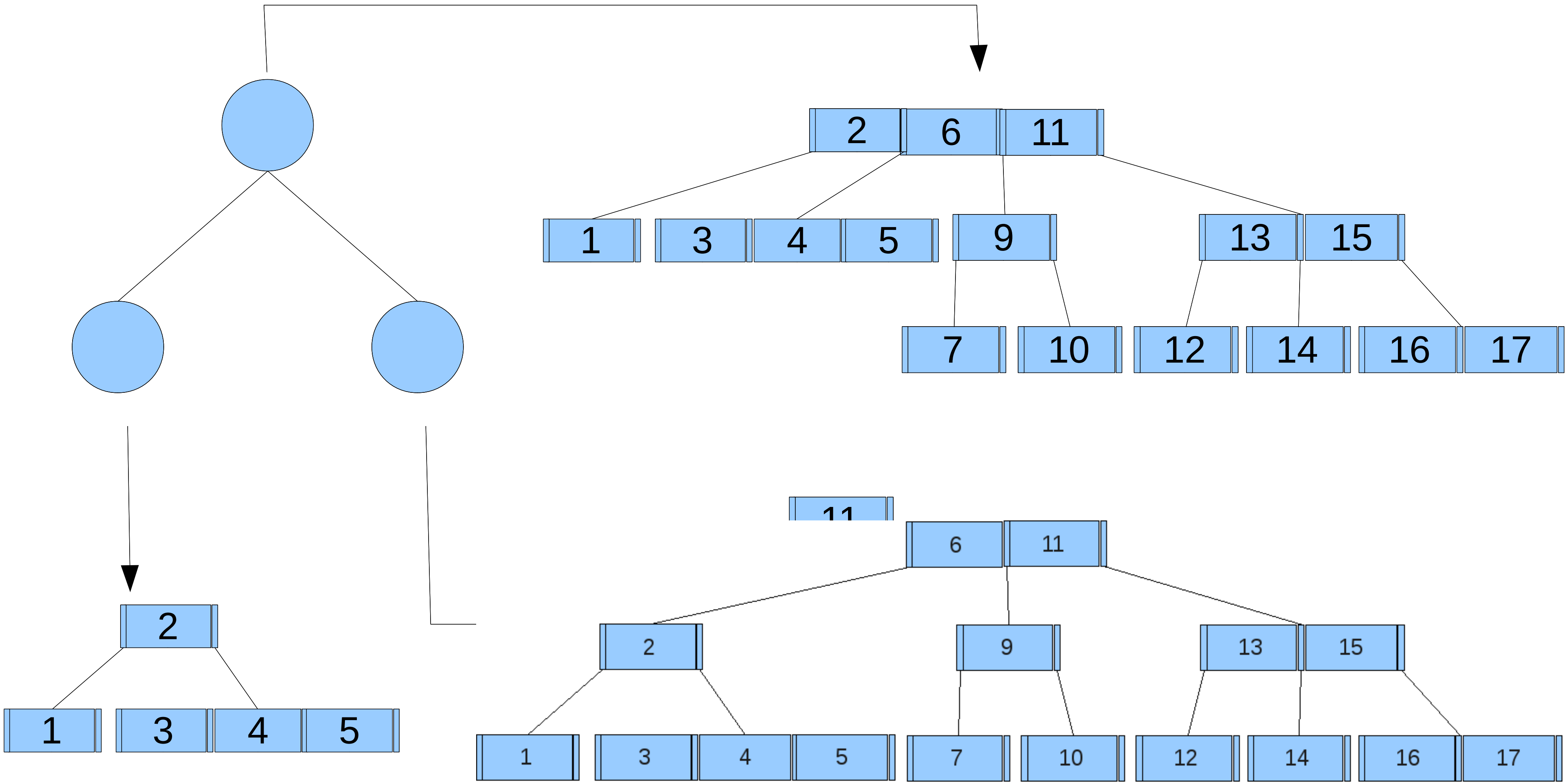
BATON Find and Replacement

1. Let d_node be the departing node.
2. **If** (d_node.left_child != NULL) **Then,**
3. forward the replacement request to d_node.left_child.
4. **Else If** (d_node.right_child != NULL)
5. forward the replacement request to d_node.right_child.
6. **Else**
7. Let m be a node in left_routing_table(d_node) or right_routing_table(d_node) not having enough children.
8. **If** there exist m **Then,**
9. forward the replacement request to a child of m.
10. **Else**
11. replace d_node with the current leaf node.
12. **End If**
13. **End If**

Merge B-tree Algorithm

```
Let p_tree be the B-tree of the parent node.
Let d_tree be the B-tree of the departing node.
Let y be d_tree.root.
Let z be p_tree.root.
Let u = y.get_first_index_entry().
While (y.has_next_index_entry() != NULL)
    Let temp = u.
    Let u = y.get_next()_index_entry.
    z.add(temp).
    If (z.subtree_size() > MAX_index_entry) Then,
        split node.
    End If
End While
return p_tree.
```

Departure Operation



Video Search Algorithm

```
1.   Let min be the minimum key of the query key.
2.   Let max be the maximum key of the query key.
3.   Let node be the current query node.
4.   If node.minkey <= min and min <= node.maxkey Then,
5.        Return the physical addresses paired with the keys in the current node.
6.        If max >= node.maxkey Then,
7.             video_query_range( max, node )
8.        End If
9.   Else
10.       If node.minkey <= min and ( min < node.right_adjacent.minkey or
           Min < node.right_child.minkey ) Then,
11.            Not found the key in the tree use min in the current minkey node.
12.            If max >= node.maxkey Then,
13.                 video_query_range( max, node )
14.            End If
15.       Else If node.maxkey < min
16.            m = The_farthest_node_satisfying_condition( m.minkey <= min )
17.            If there exist m Then,
18.                 Propagate the query to the node m.
19.            Else
20.                 If there exist node.right_child Then,
```

video query range Algorithm

1. Let k be max key for query.
2. Let node be the current query node.
3. Let L(node) be load of node.
4. Let threshold = $\lfloor c\delta^{i} \rfloor$
5. **While** node.maxkey > k
6. Return the physical addresses paired with the keys in the current node.
7. **If** L(node)i > threshold **Then,**
8. ladbalancing(node)
9. **End If**
10. node = node.right_adjacent
11. **End While**

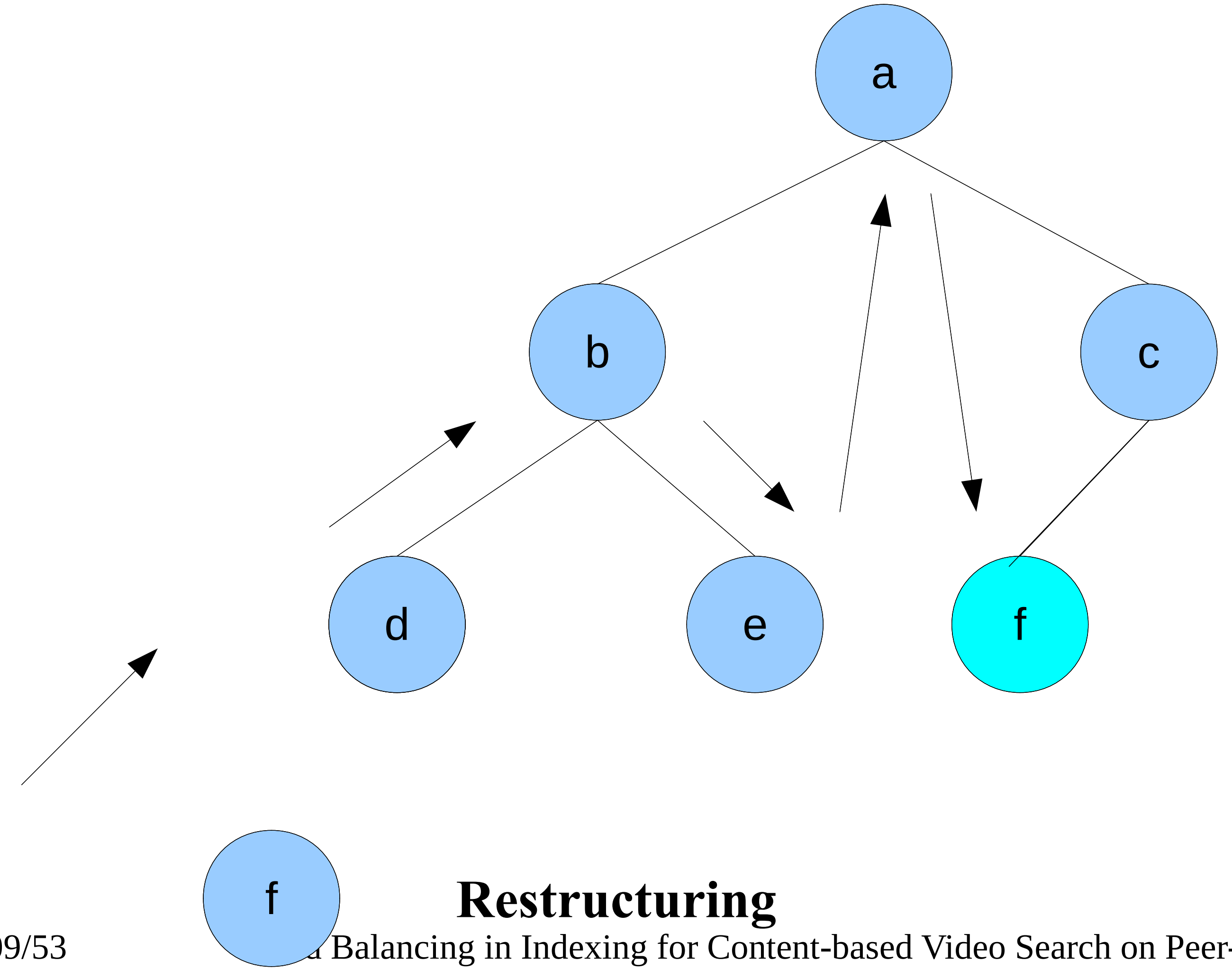
Load Balancing Algorithm

1. Let m be a lightly loaded node.
2. departure(m)
3. forcejoin(m)
4. **If** BATON tree imbalance **Then,**
5. restructuring(m)
6. **End If**

Restructure Algorithm

1. Let m be
2. Let n be m.right_adjacent.
3. **If** isFull(left_routing_table(n)) and isFull(right_routing_table(n)) and n.left_child = NULL **Then**,
4. Add m to child of n.
5. **Else If** isFull(left_routing_table(n)) and isFull(right_routing_table(n)) and n.right_child = NULL **Then**,
6. replace(n, m) and Add to child of m.
7. **Else**
8. replace(n,m) and call restructuring() again.
9. **End If**

Load balancing



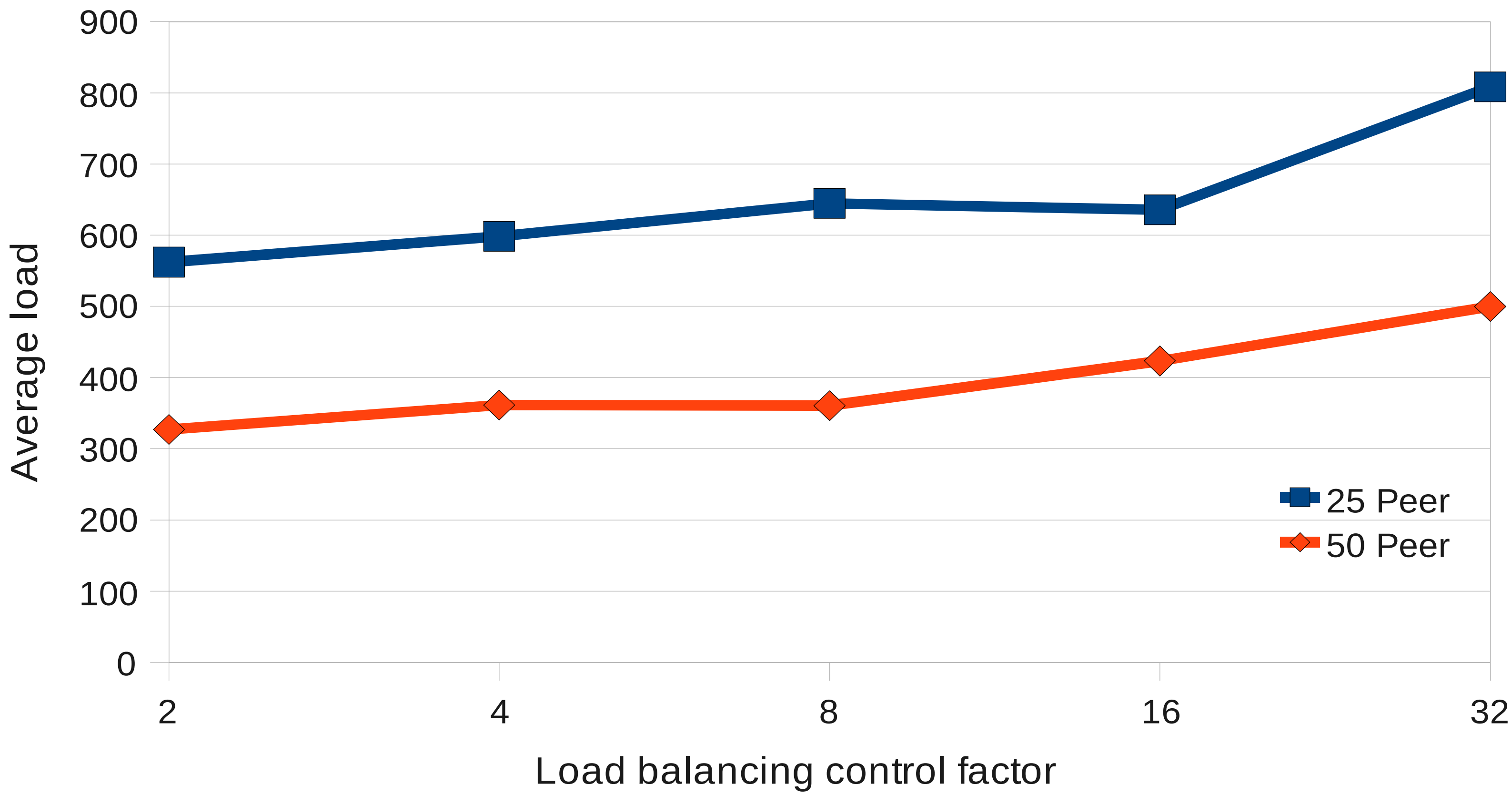
Restructuring

Experiment

- ทดลองกับวิดีโอจำนวน 1000 วิดีโอ
  - Frame rate 25 fps.
  - Video length 10s, 15s and 30s

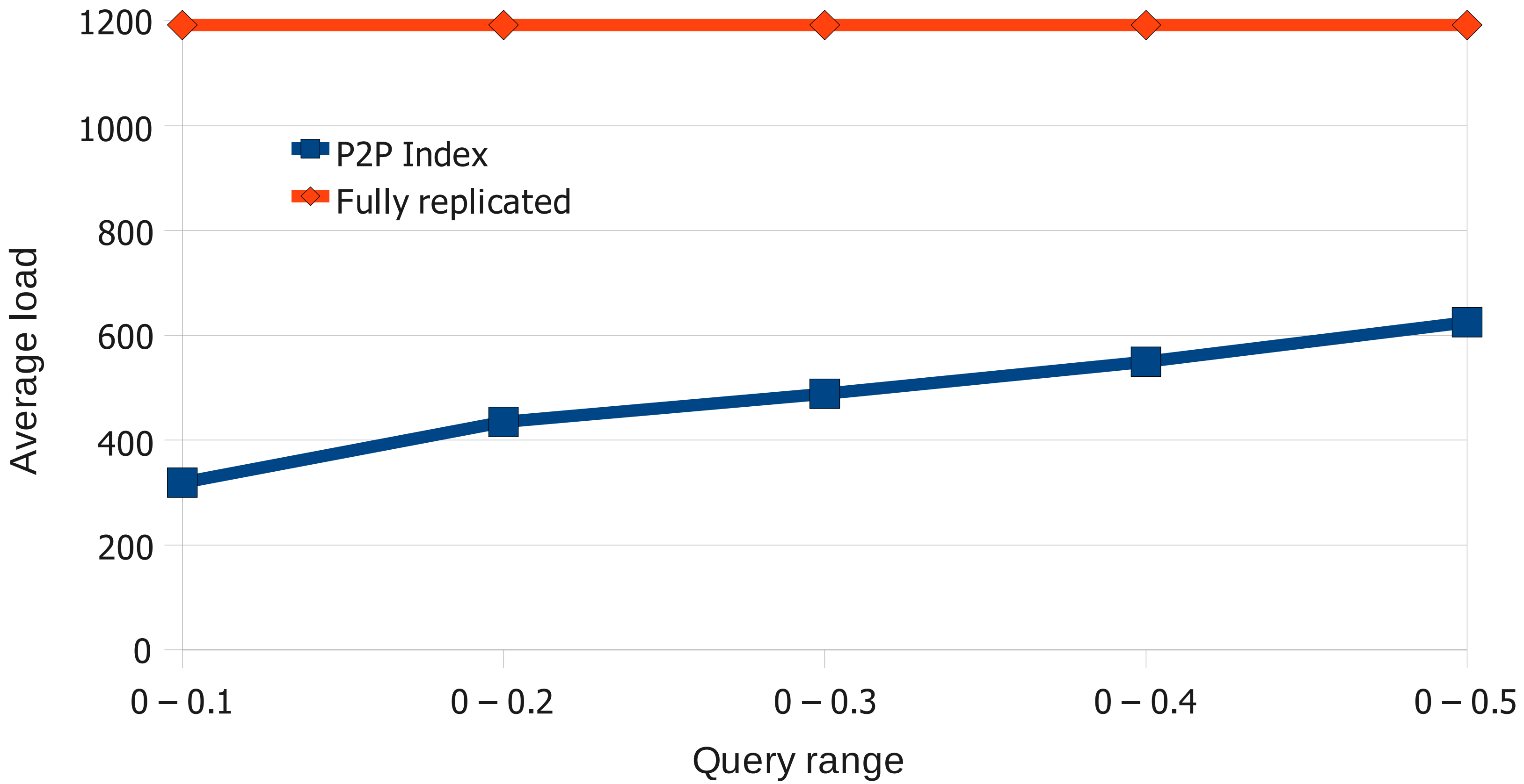
Load balancing control factor change

- 100 keys, 0 - 0.1



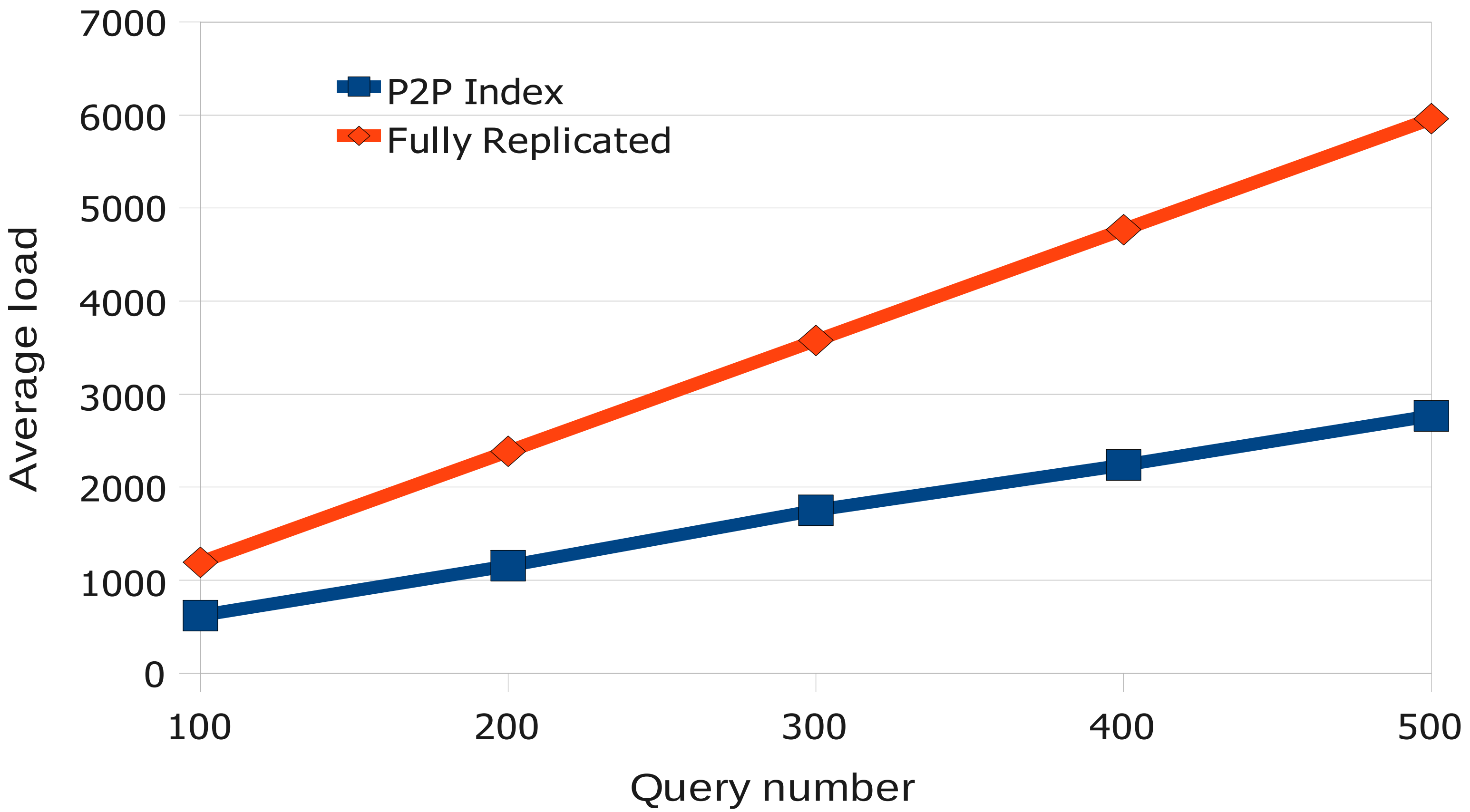
Query range change

- Delta = 2, 100 keys, 50 peer



Query number change

- Delta = 2, 0 – 0.5, 50 peer



Conclusion

- Using B-tree instead B^+-tree
- Framework index videos P2P model
- Our approach more efficient than comparing approach

Thank You

Q&A