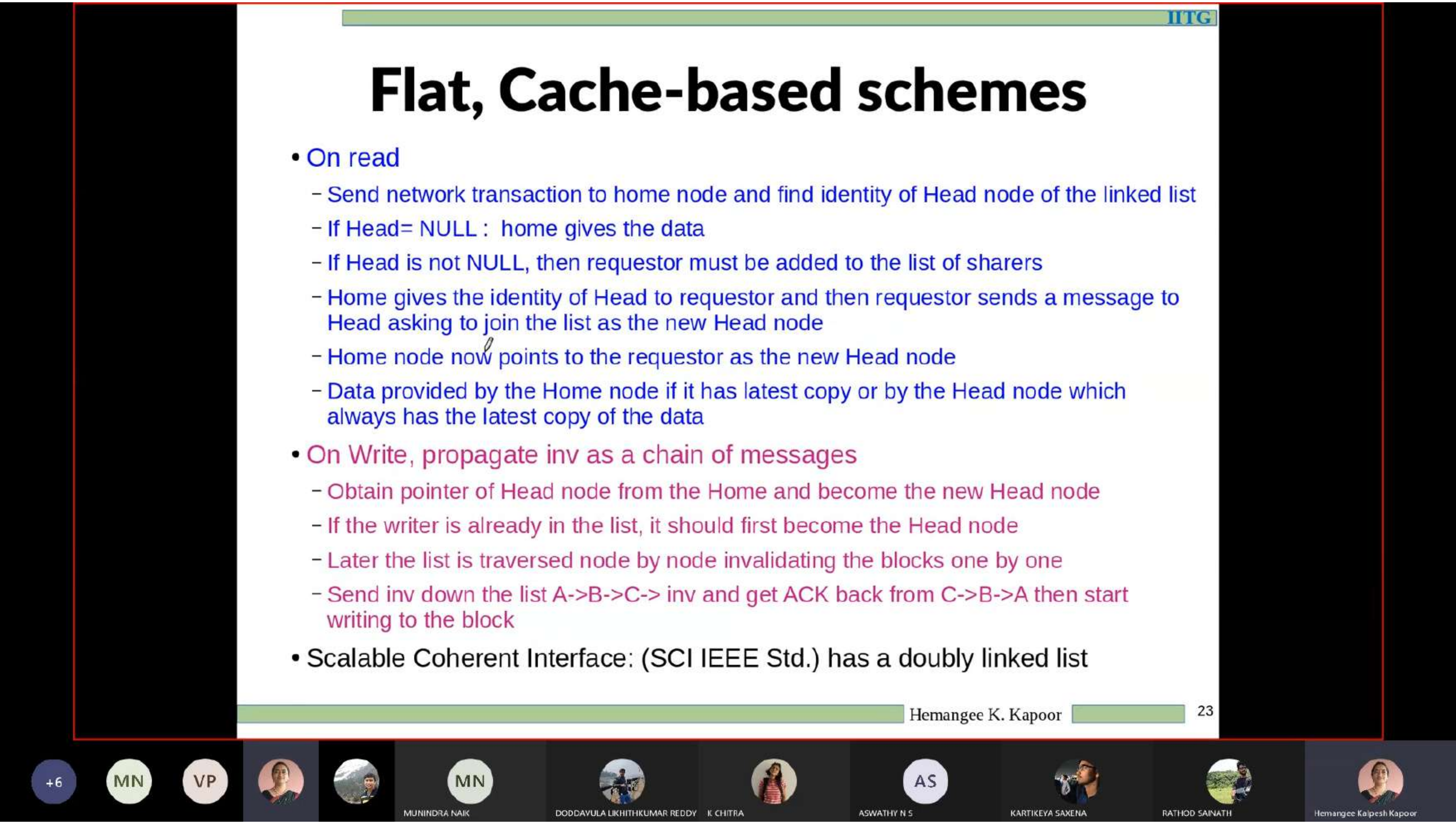
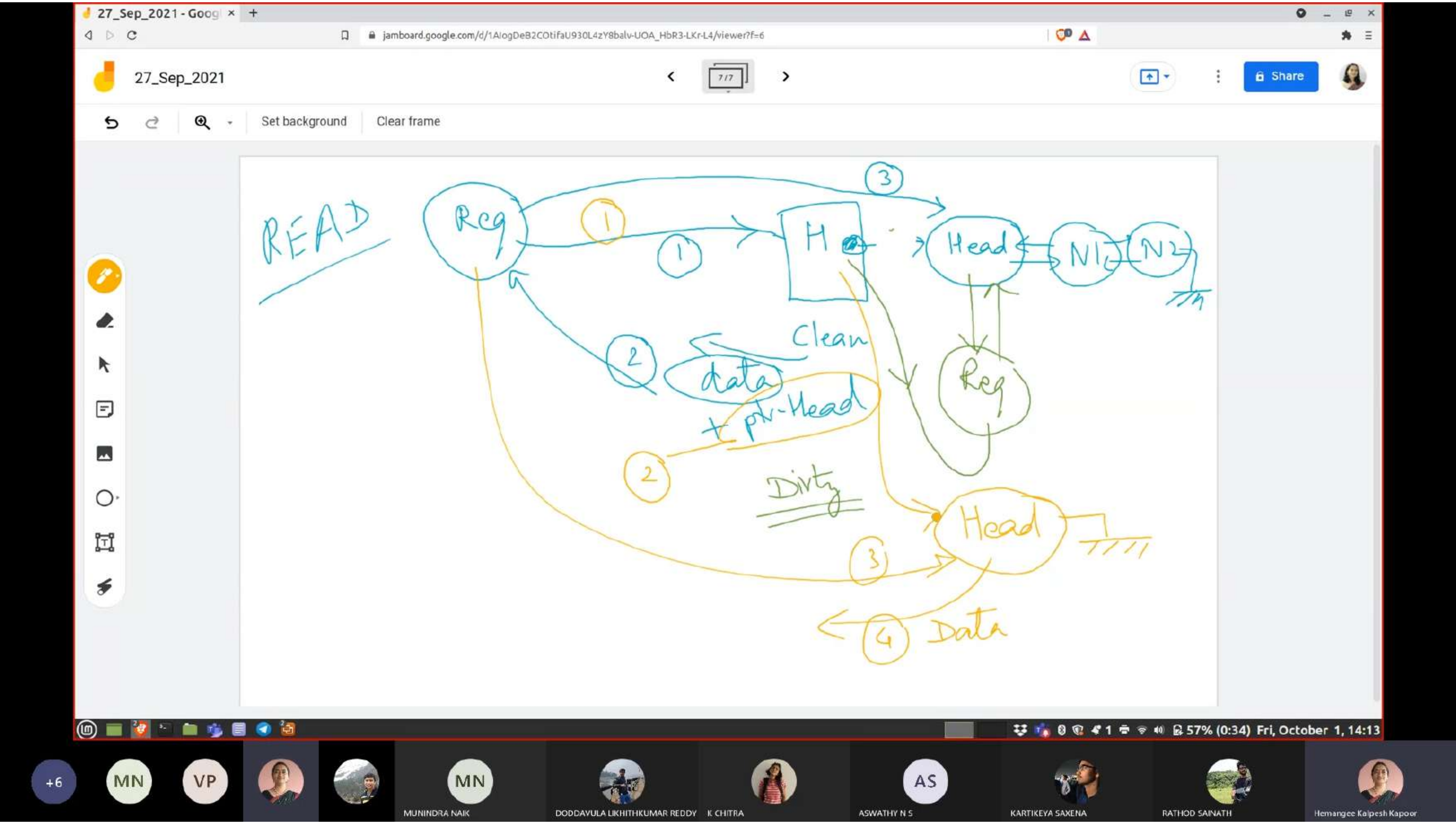
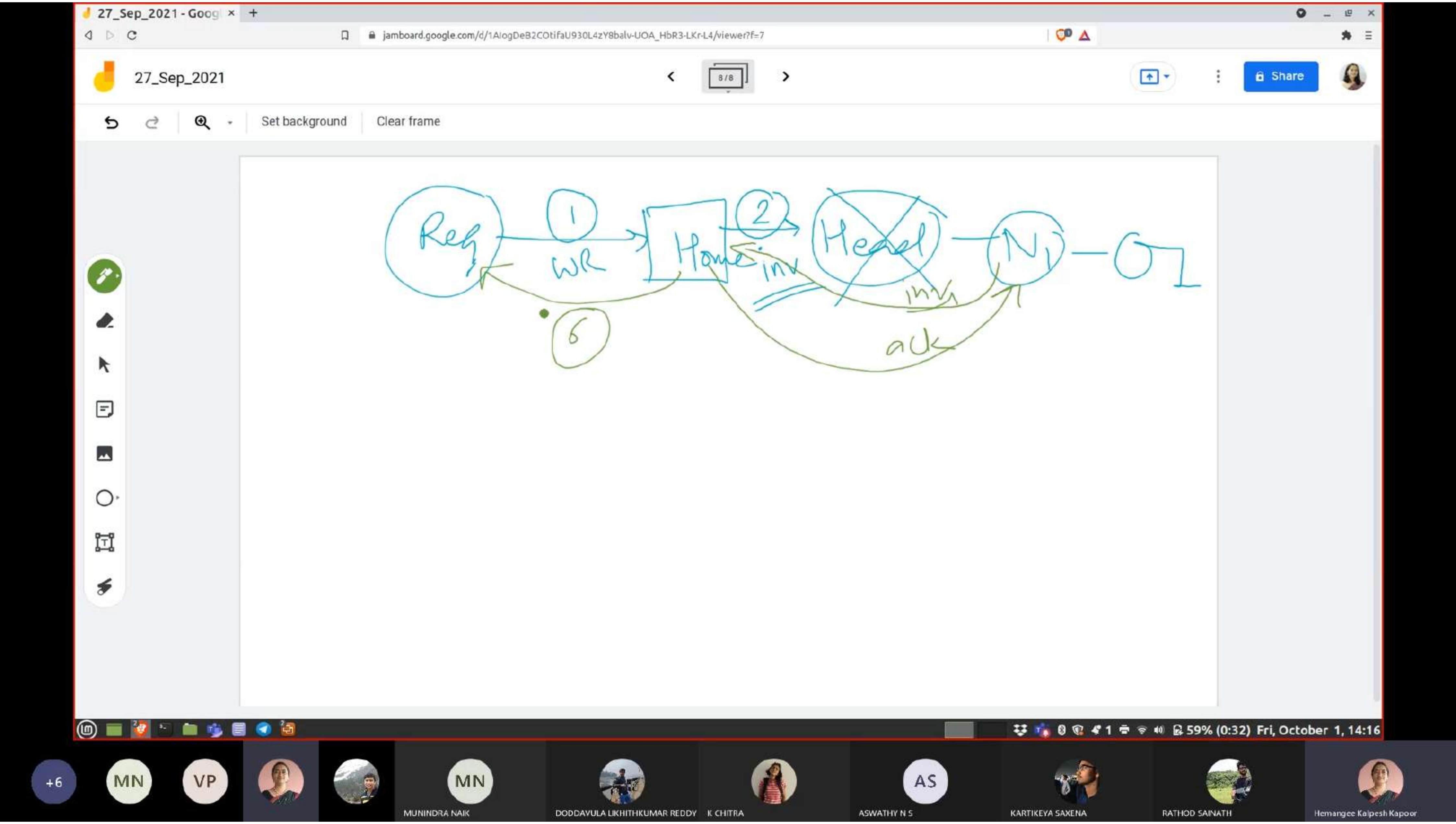
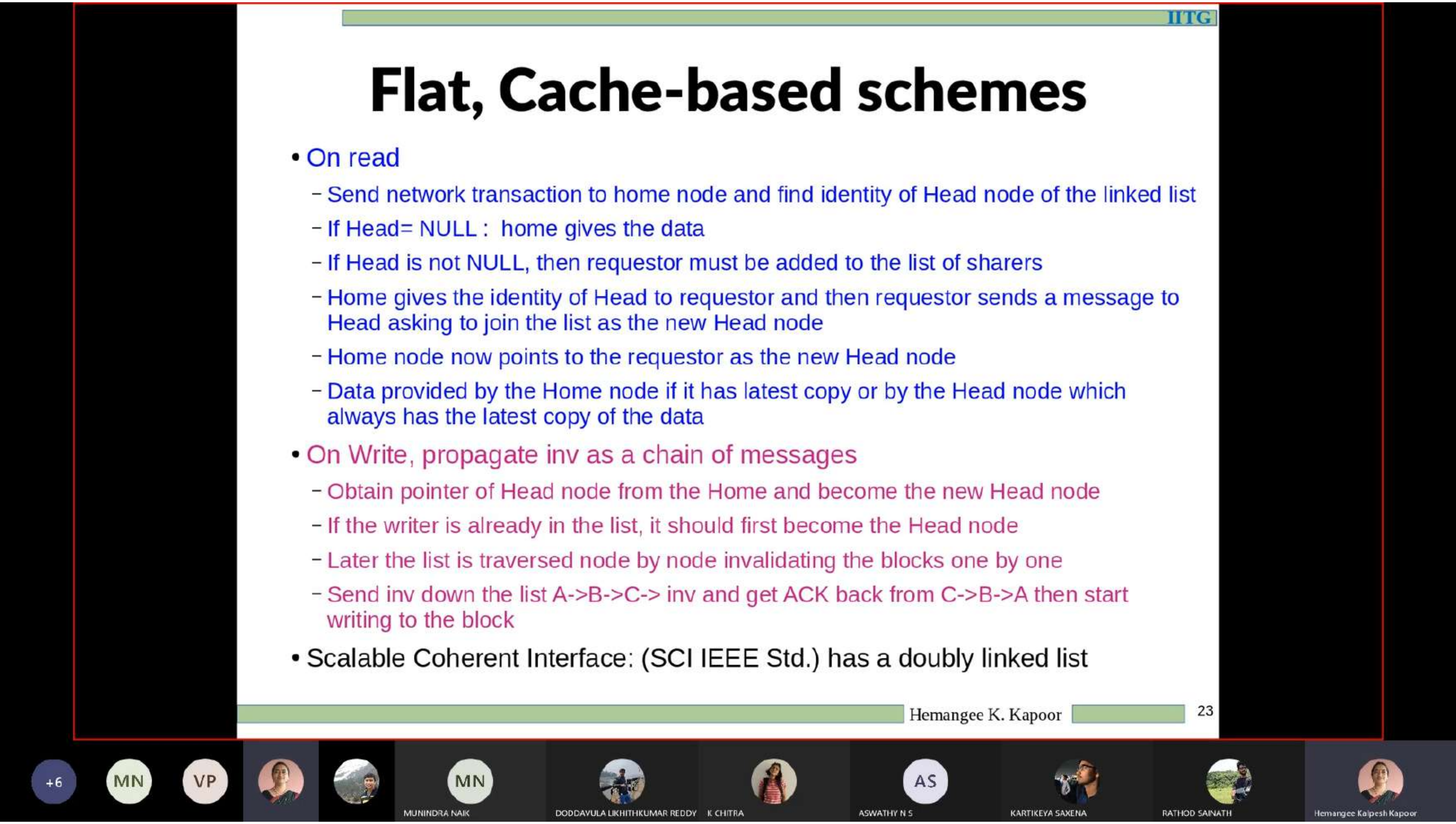


Flat, Cache-based schemes

- On read
  - Send network transaction to home node and find identity of Head node of the linked list
  - If Head= NULL : home gives the data
  - If Head is not NULL, then requestor must be added to the list of sharers
  - Home gives the identity of Head to requestor and then requestor sends a message to Head asking to join the list as the new Head node
  - Home node now points to the requestor as the new Head node
  - Data provided by the Home node if it has latest copy or by the Head node which always has the latest copy of the data
- On Write, propagate inv as a chain of messages
  - Obtain pointer of Head node from the Home and become the new Head node
  - If the writer is already in the list, it should first become the Head node
  - Later the list is traversed node by node invalidating the blocks one by one
  - Send inv down the list A->B->C-> inv and get ACK back from C->B->A then start writing to the block
- Scalable Coherent Interface: (SCI IEEE Std.) has a doubly linked list

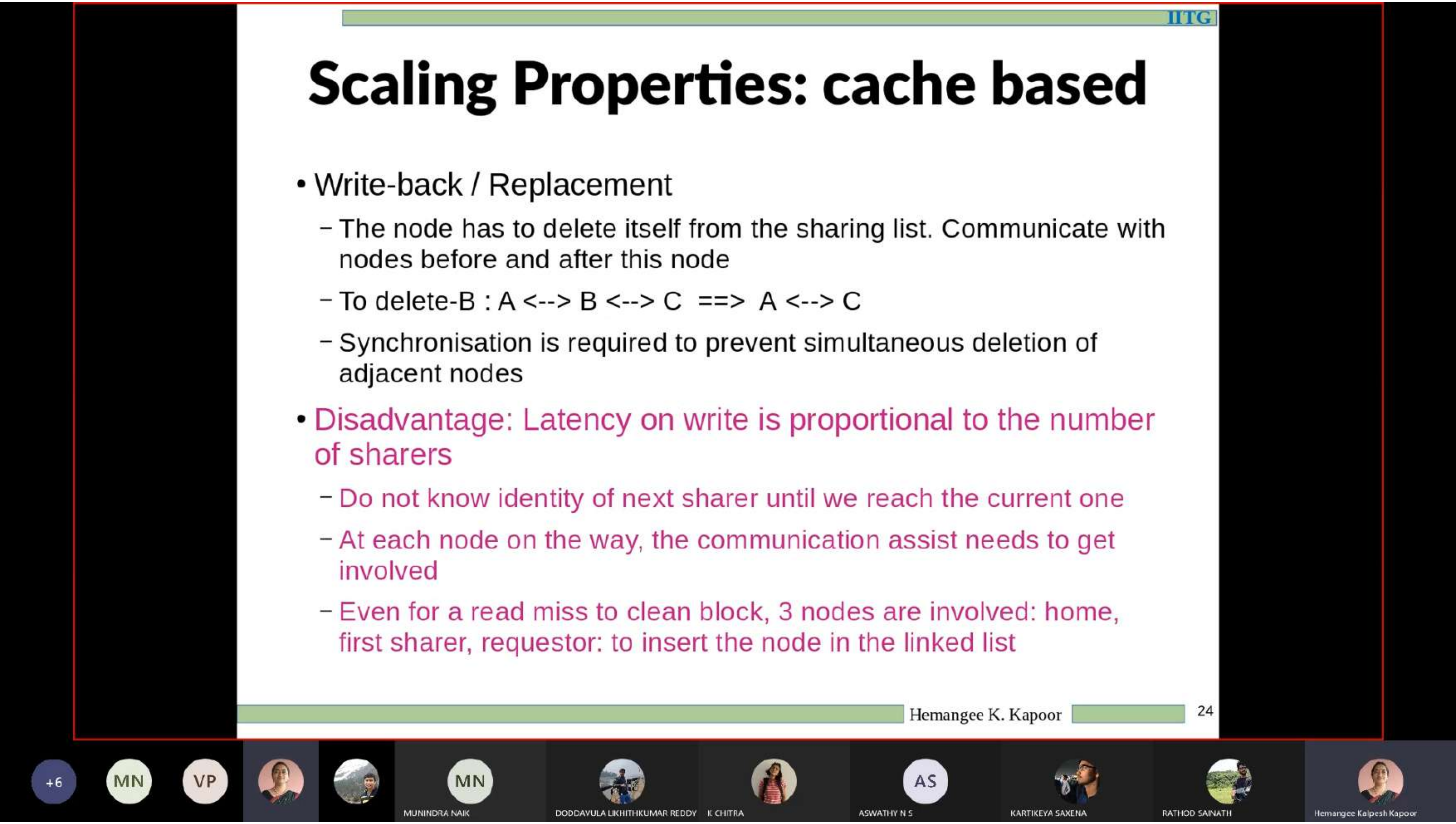

27_Sep_2021 - Goog
jamboard.google.com/d/1AlogDeB2COtifaU930L4zY8balv-UOA_HbR3-LKr-L4/viewer?f=6
27_Sep_2021
7/7
Share
Set background
Clear frame
READ
Req
1
1
H
Head
N1
N2
3
Clean
2
data
+ ptr-Head
2
Dirty
Req
Head
3
4
Data
57% (0:34) Fri, October 1, 14:13
+6
MN
VP
MN
MUNINDRA NAIK
DODDAVULA LIKHITHKUMAR REDDY
K CHITRA
AS
ASWATHY N S
KARTIKEYA SAXENA
RATHOD SAINATH
Hemangee Kalpesh Kapoor


27_Sep_2021 - Goog
jamboard.google.com/d/1AlogDeB2COtifaU930L4zY8balv-UOA_HbR3-LKr-L4/viewer?f=7
27_Sep_2021
8/8
Share
Set background
Clear frame
Req
1
WR
Home
2
Head
inv
N1
inv1
ack
6
59% (0:32) Fri, October 1, 14:16
+6
MN
VP
MUNINDRA NAIK
MN
DODDAVULA LIKHITHKUMAR REDDY
K CHITRA
ASWATHY N S
AS
KARTIKEYA SAXENA
RATHOD SAINATH
Hemangee Kalpesh Kapoor



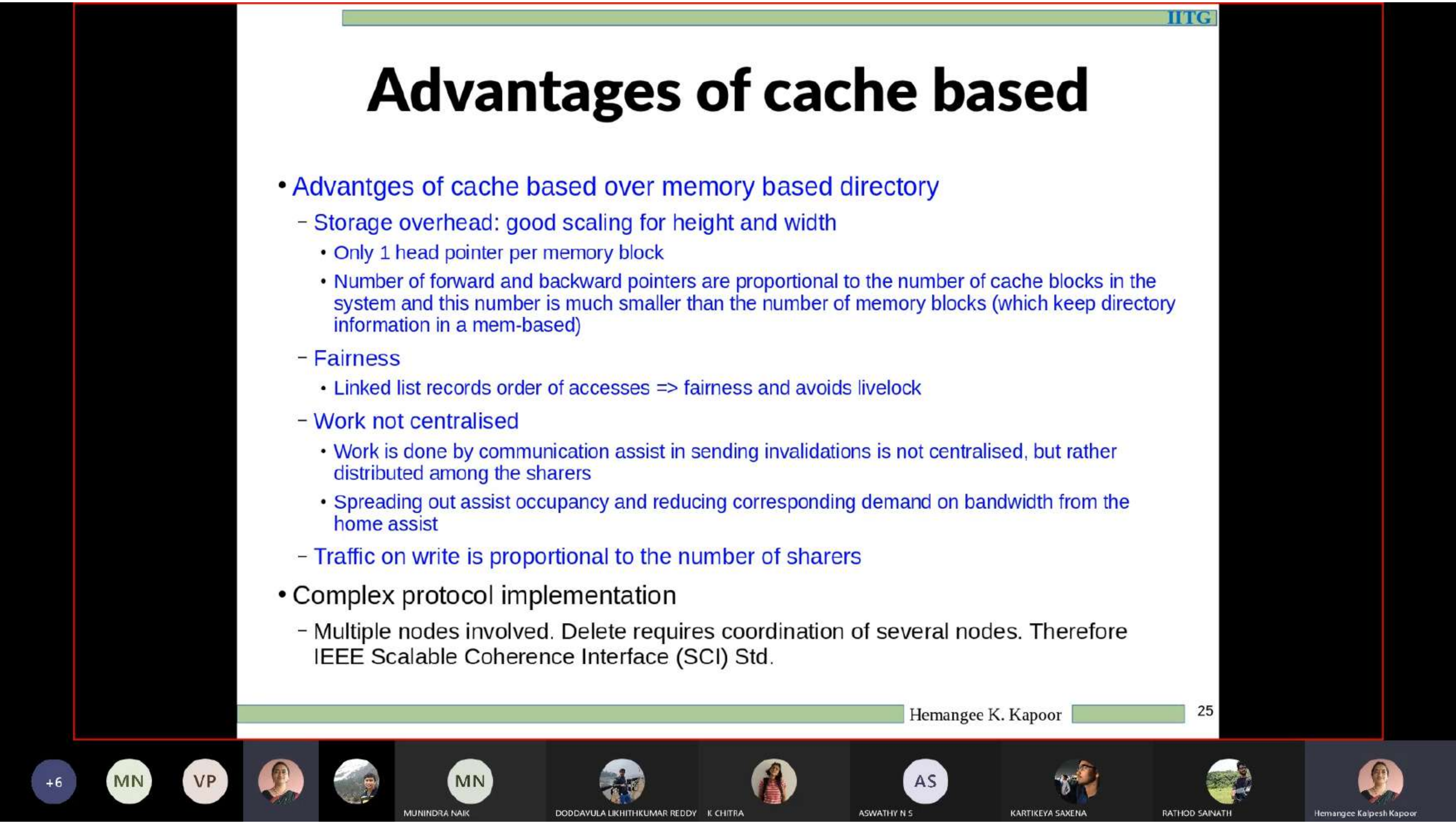
Flat, Cache-based schemes

- On read
  - Send network transaction to home node and find identity of Head node of the linked list
  - If Head= NULL : home gives the data
  - If Head is not NULL, then requestor must be added to the list of sharers
  - Home gives the identity of Head to requestor and then requestor sends a message to Head asking to join the list as the new Head node
  - Home node now points to the requestor as the new Head node
  - Data provided by the Home node if it has latest copy or by the Head node which always has the latest copy of the data
- On Write, propagate inv as a chain of messages
  - Obtain pointer of Head node from the Home and become the new Head node
  - If the writer is already in the list, it should first become the Head node
  - Later the list is traversed node by node invalidating the blocks one by one
  - Send inv down the list A->B->C-> inv and get ACK back from C->B->A then start writing to the block
- Scalable Coherent Interface: (SCI IEEE Std.) has a doubly linked list



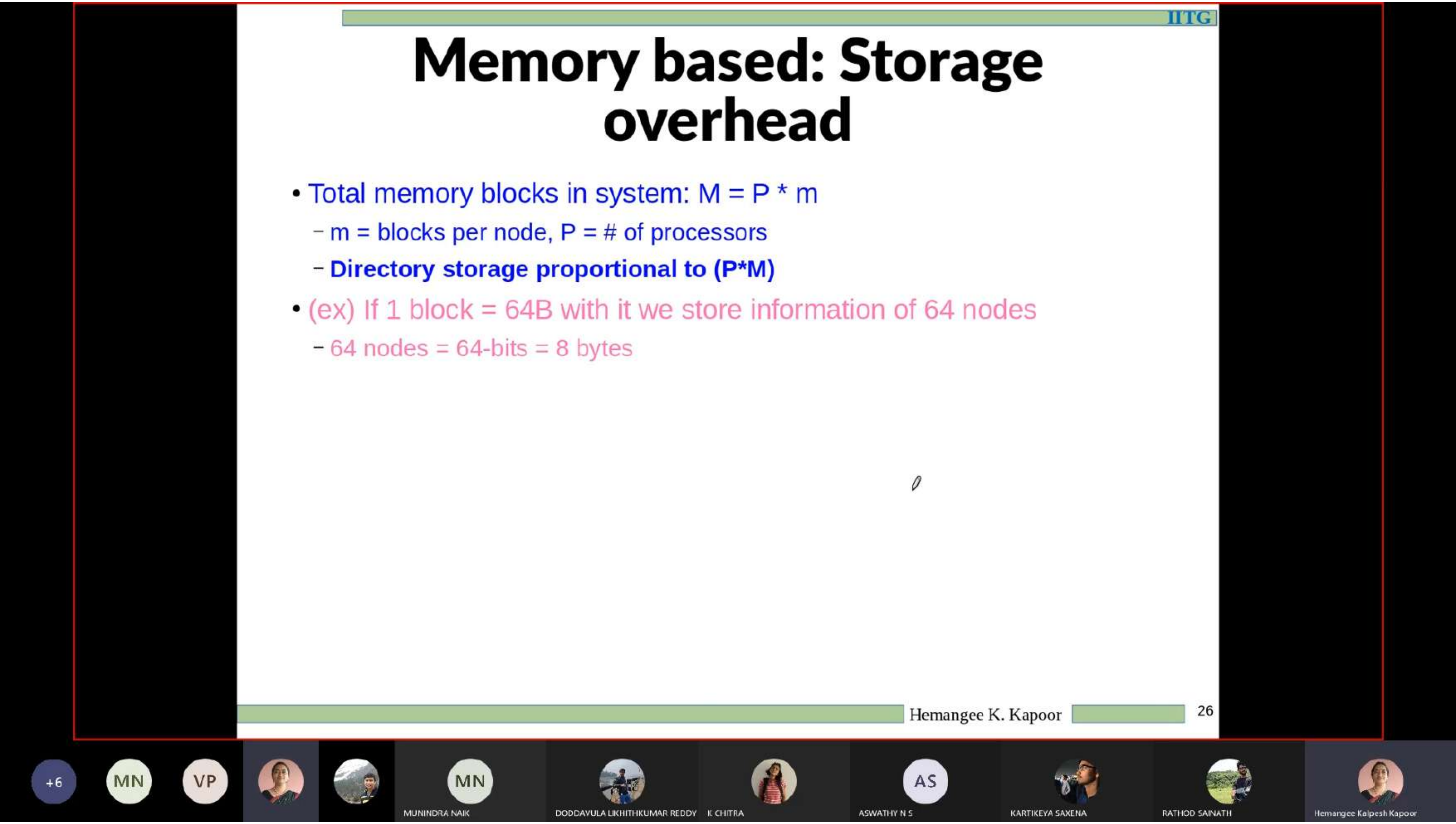
Scaling Properties: cache based

- Write-back / Replacement
  - The node has to delete itself from the sharing list. Communicate with nodes before and after this node
  - To delete-B : A <--> B <--> C ==> A <--> C
  - Synchronisation is required to prevent simultaneous deletion of adjacent nodes
- Disadvantage: Latency on write is proportional to the number of sharers
  - Do not know identity of next sharer until we reach the current one
  - At each node on the way, the communication assist needs to get involved
  - Even for a read miss to clean block, 3 nodes are involved: home, first sharer, requestor: to insert the node in the linked list



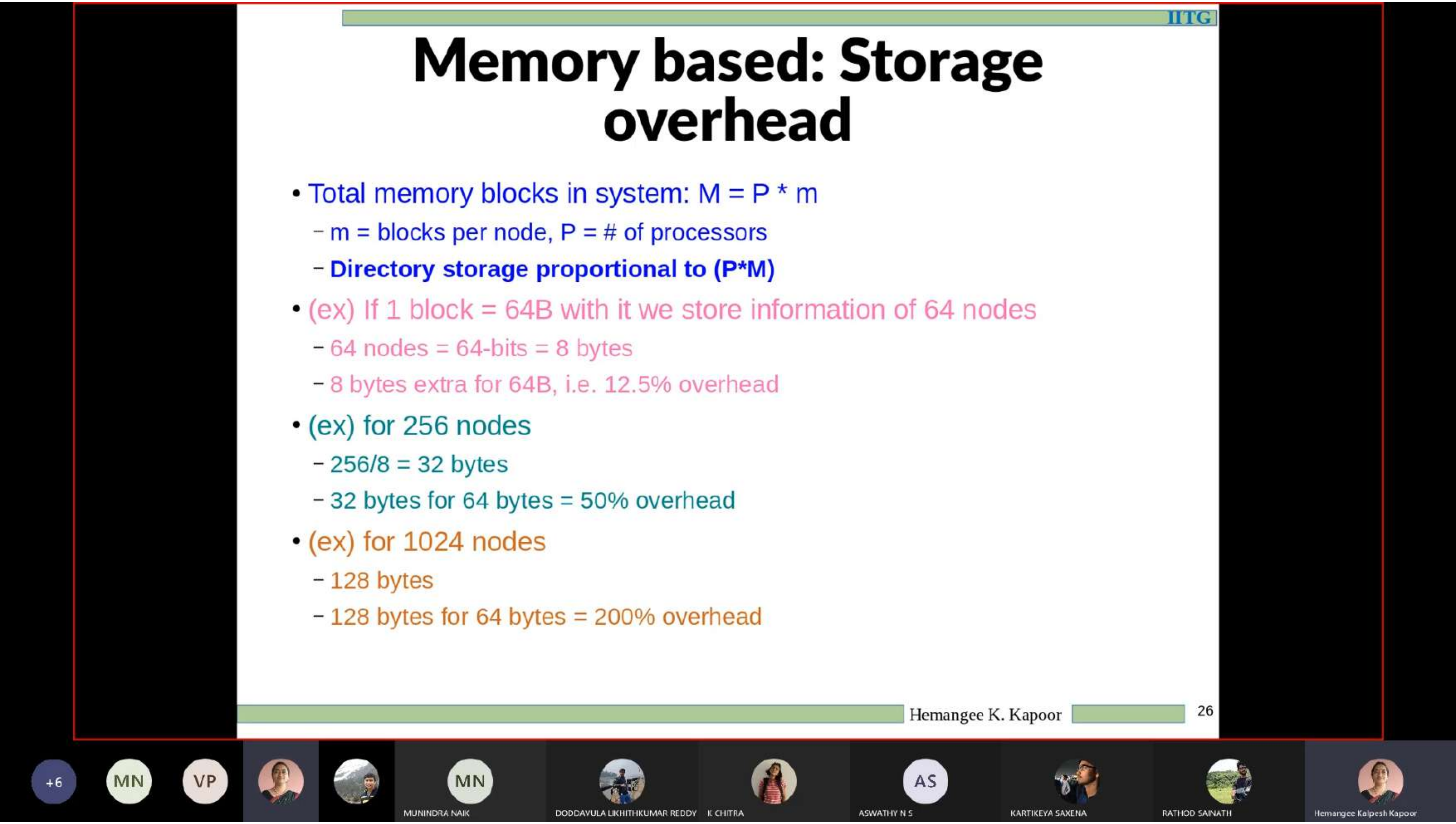
Advantages of cache based

- Advantges of cache based over memory based directory
  - Storage overhead: good scaling for height and width
    - Only 1 head pointer per memory block
    - Number of forward and backward pointers are proportional to the number of cache blocks in the system and this number is much smaller than the number of memory blocks (which keep directory information in a mem-based)
  - Fairness
    - Linked list records order of accesses => fairness and avoids livelock
  - Work not centralised
    - Work is done by communication assist in sending invalidations is not centralised, but rather distributed among the sharers
    - Spreading out assist occupancy and reducing corresponding demand on bandwidth from the home assist
  - Traffic on write is proportional to the number of sharers
- Complex protocol implementation
  - Multiple nodes involved. Delete requires coordination of several nodes. Therefore IEEE Scalable Coherence Interface (SCI) Std.



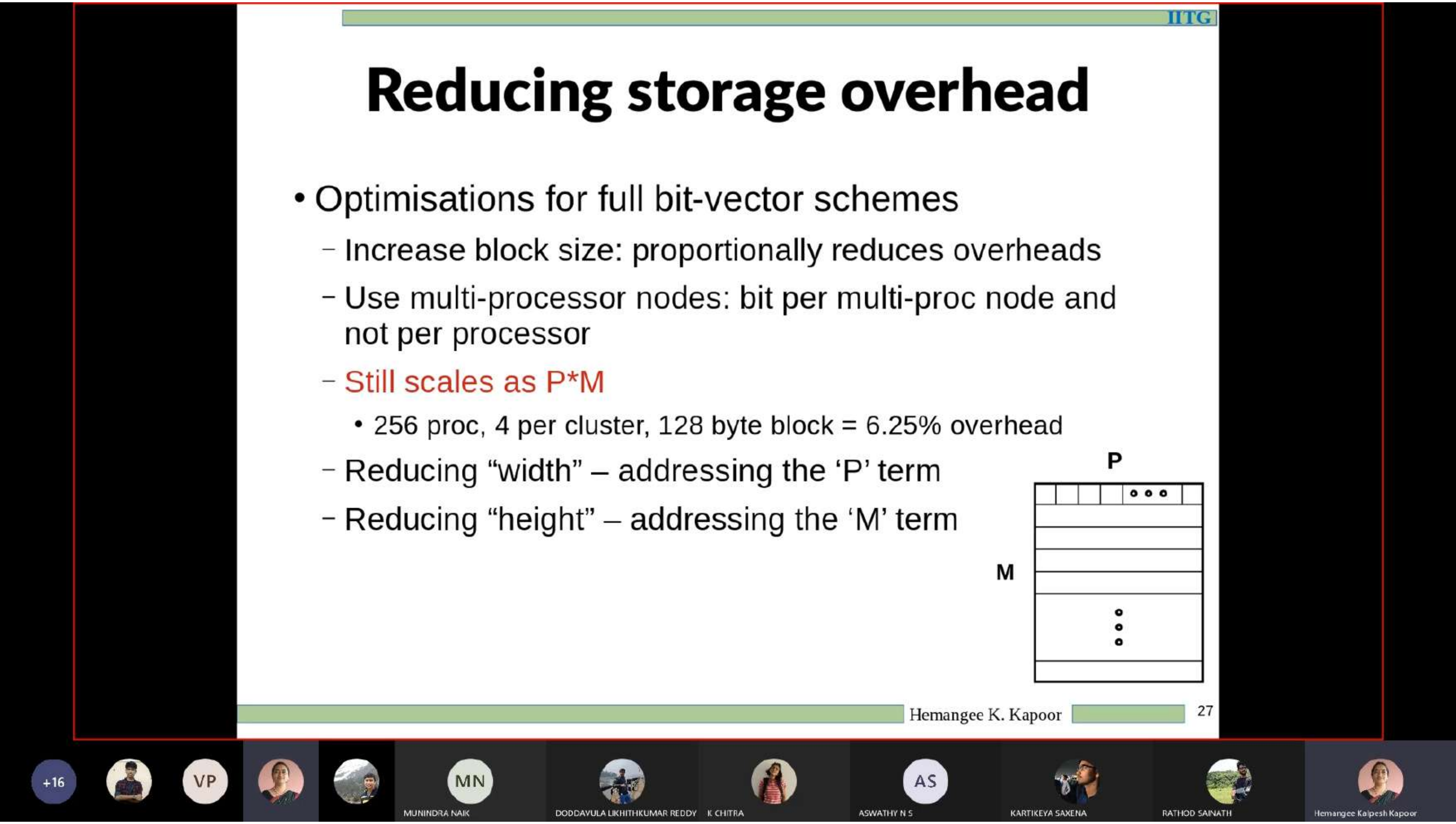
Memory based: Storage overhead

- Total memory blocks in system: M = P * m
  - m = blocks per node, P = # of processors
  - **Directory storage proportional to (P*M)**
- (ex) If 1 block = 64B with it we store information of 64 nodes
  - 64 nodes = 64-bits = 8 bytes



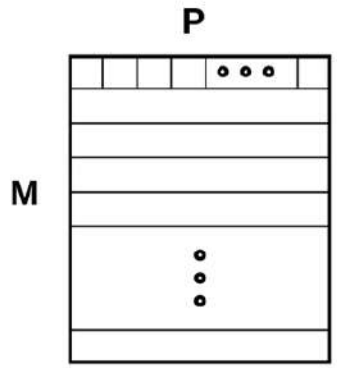
Memory based: Storage overhead

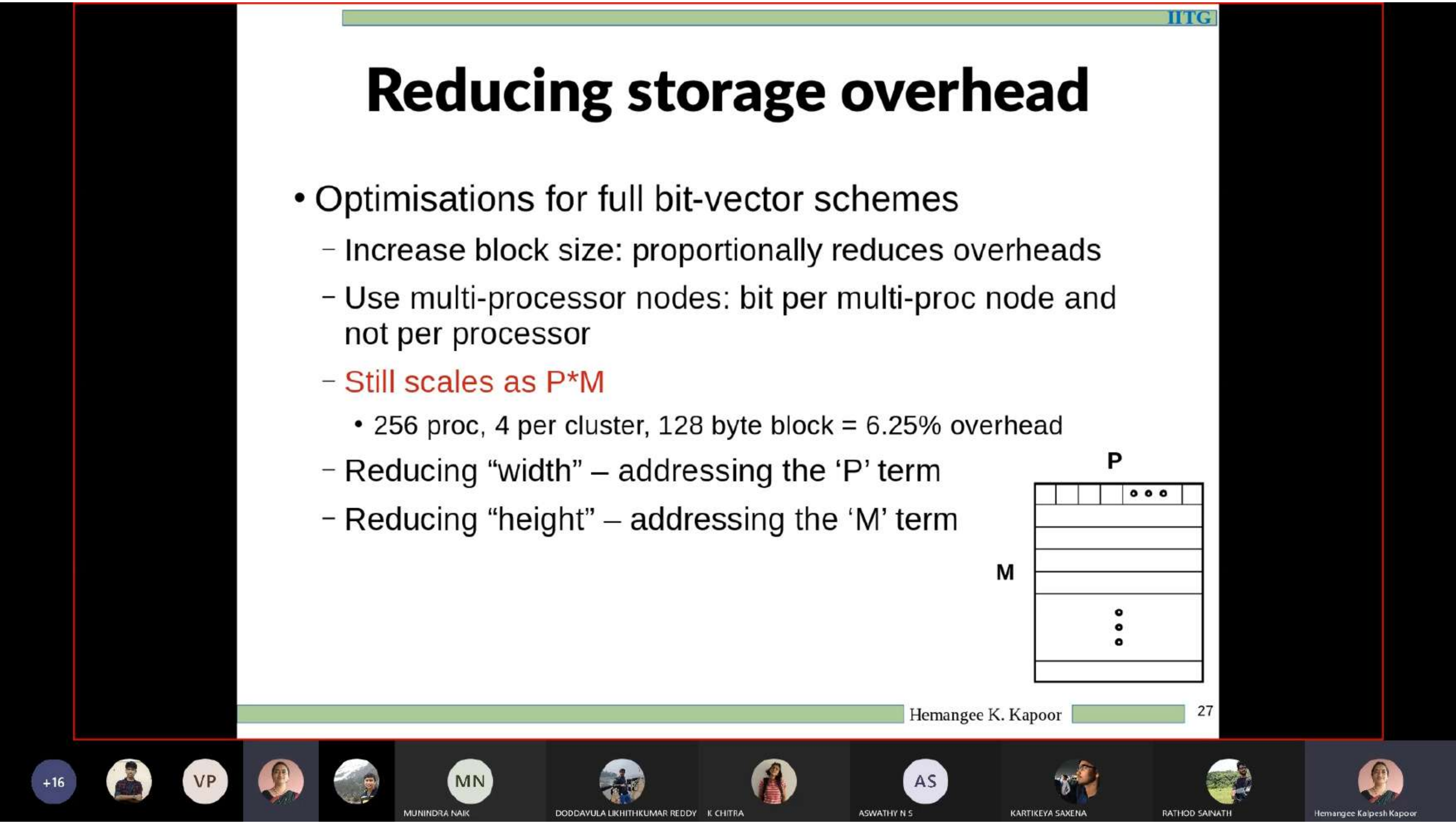
- Total memory blocks in system: M = P * m
  - m = blocks per node, P = # of processors
  - **Directory storage proportional to (P*M)**
- (ex) If 1 block = 64B with it we store information of 64 nodes
  - 64 nodes = 64-bits = 8 bytes
  - 8 bytes extra for 64B, i.e. 12.5% overhead
- (ex) for 256 nodes
  - 256/8 = 32 bytes
  - 32 bytes for 64 bytes = 50% overhead
- (ex) for 1024 nodes
  - 128 bytes
  - 128 bytes for 64 bytes = 200% overhead



Reducing storage overhead

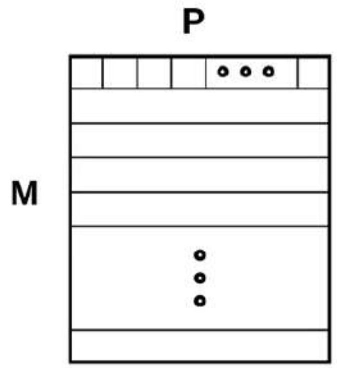
- Optimisations for full bit-vector schemes
  - Increase block size: proportionally reduces overheads
  - Use multi-processor nodes: bit per multi-proc node and not per processor
  - Still scales as P*M
    - 256 proc, 4 per cluster, 128 byte block = 6.25% overhead
  - Reducing "width" – addressing the 'P' term
  - Reducing "height" – addressing the 'M' term

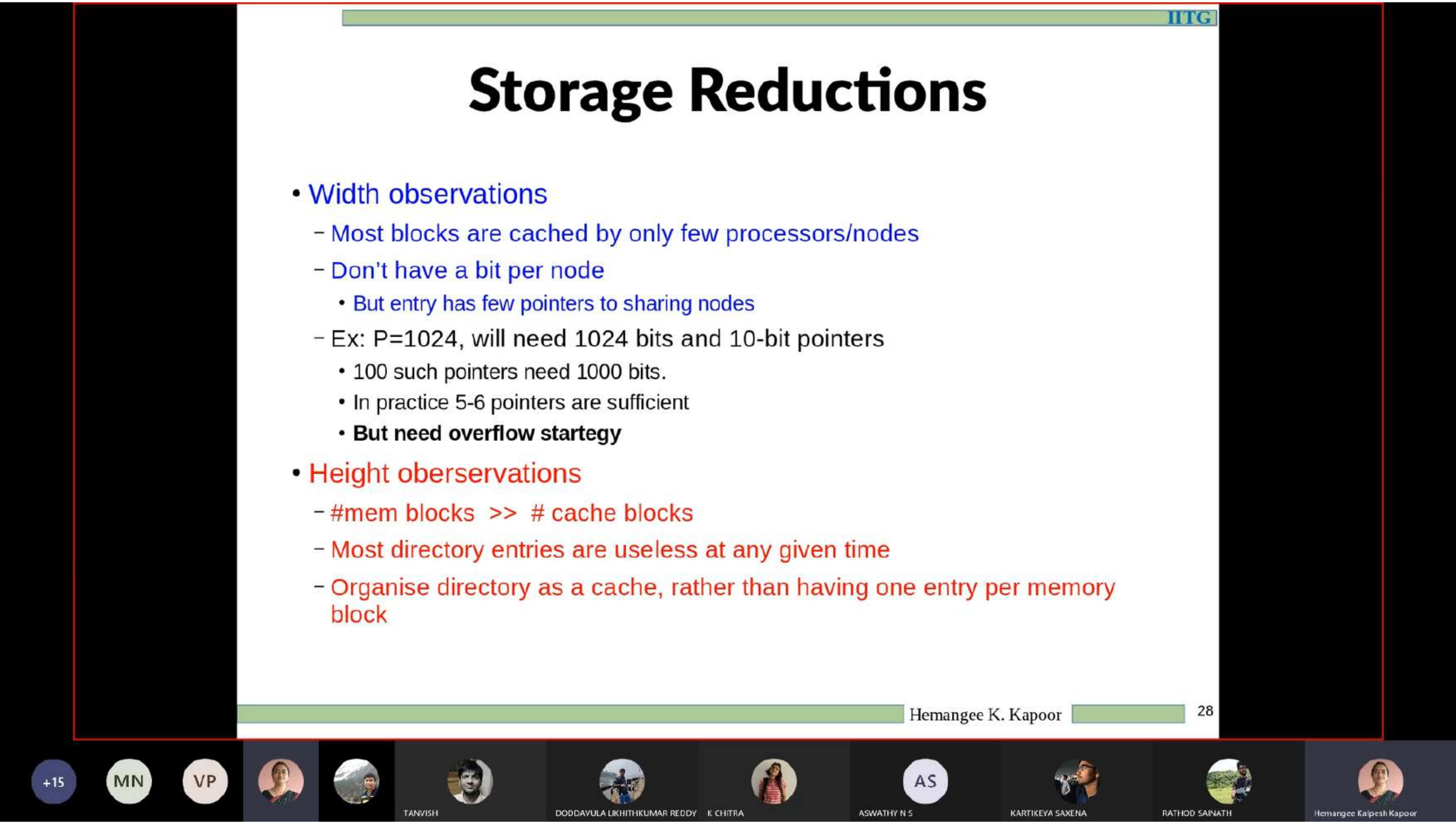




Reducing storage overhead

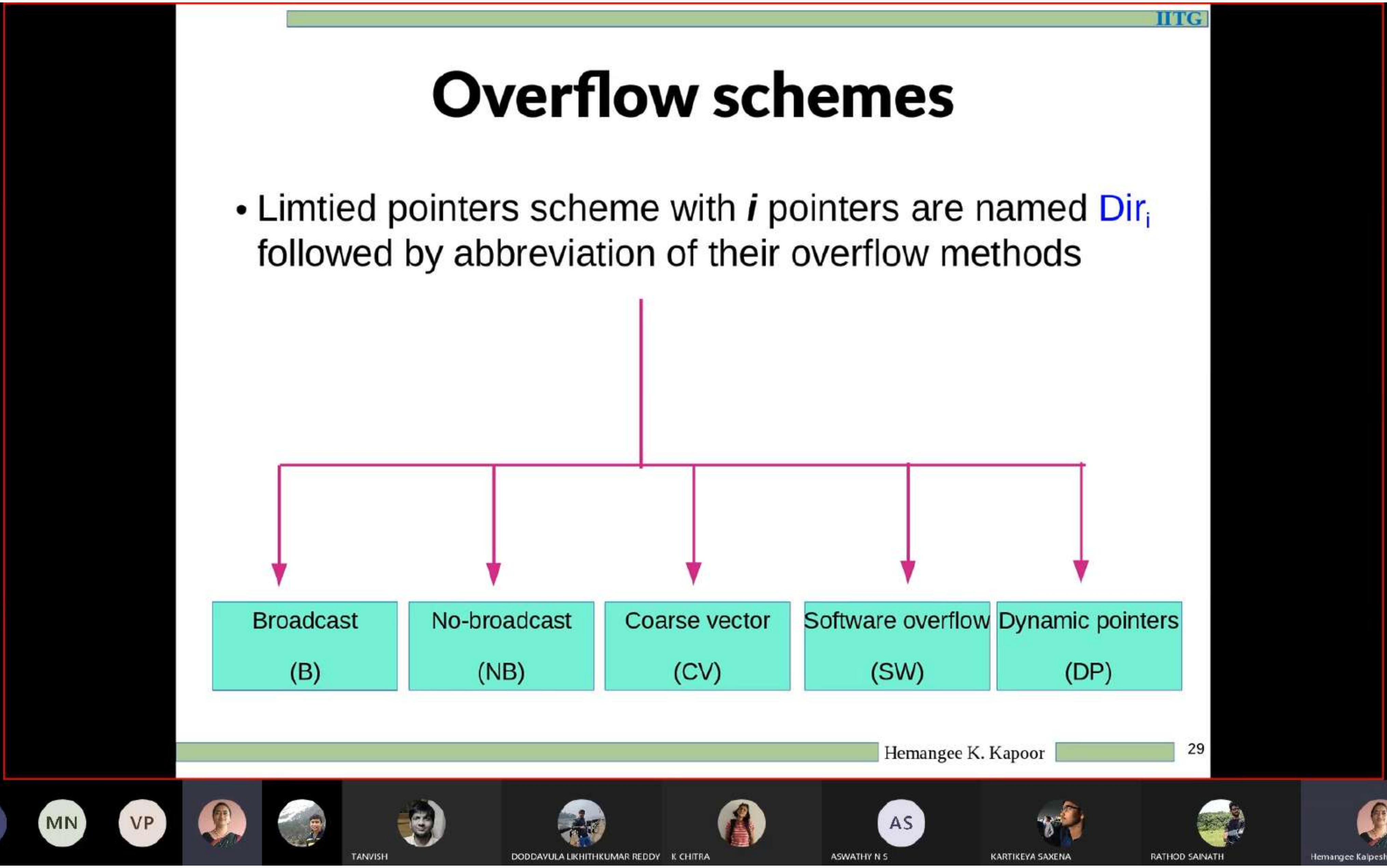
- Optimisations for full bit-vector schemes
  - Increase block size: proportionally reduces overheads
  - Use multi-processor nodes: bit per multi-proc node and not per processor
  - Still scales as P*M
    - 256 proc, 4 per cluster, 128 byte block = 6.25% overhead
  - Reducing “width” – addressing the ‘P’ term
  - Reducing “height” – addressing the ‘M’ term





Storage Reductions

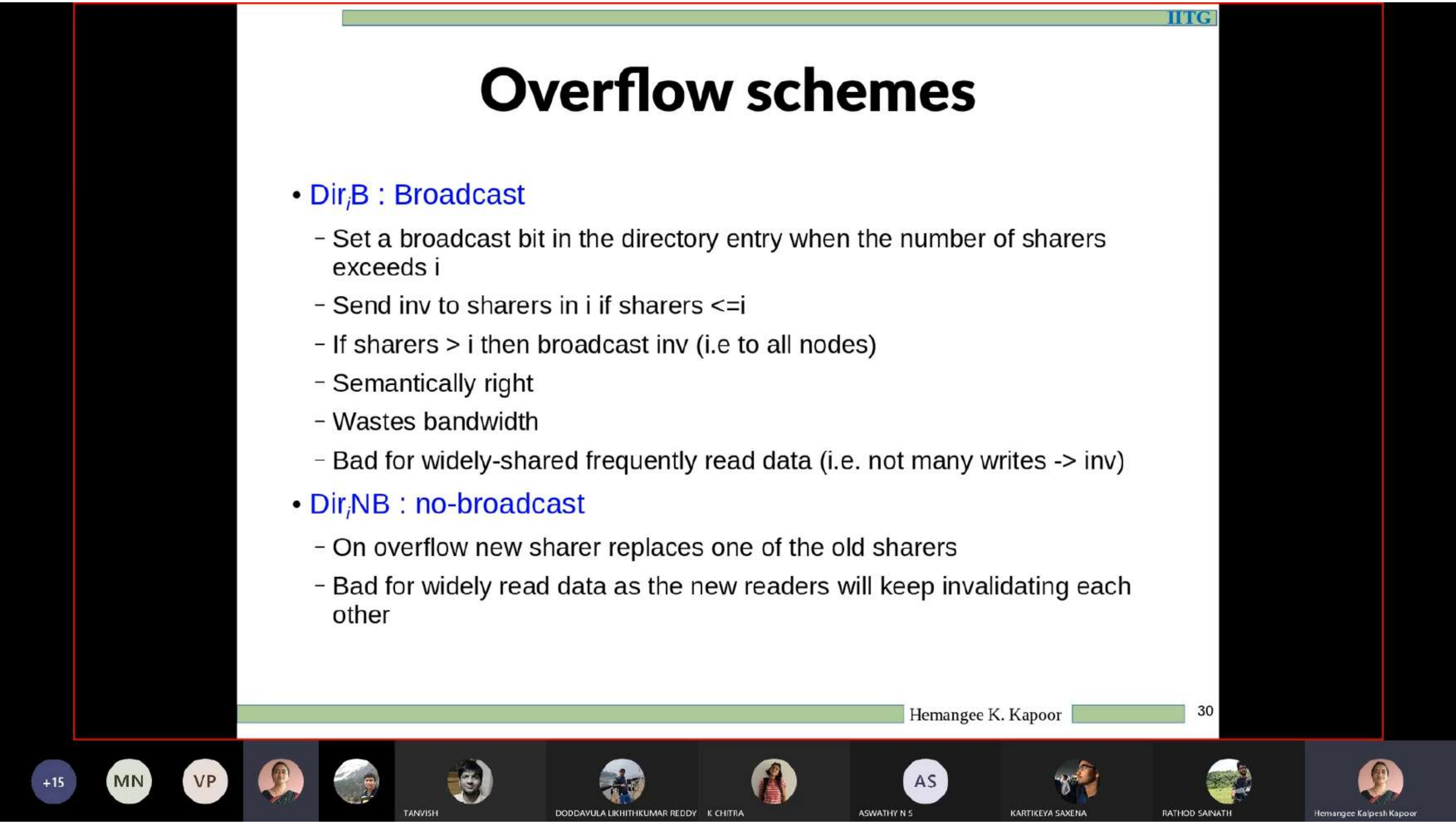
- Width observations
  - Most blocks are cached by only few processors/nodes
  - Don't have a bit per node
    - But entry has few pointers to sharing nodes
  - Ex: P=1024, will need 1024 bits and 10-bit pointers
    - 100 such pointers need 1000 bits.
    - In practice 5-6 pointers are sufficient
    - **But need overflow startegy**
- Height oberservations
  - #mem blocks >> # cache blocks
  - Most directory entries are useless at any given time
  - Organise directory as a cache, rather than having one entry per memory block



IITG

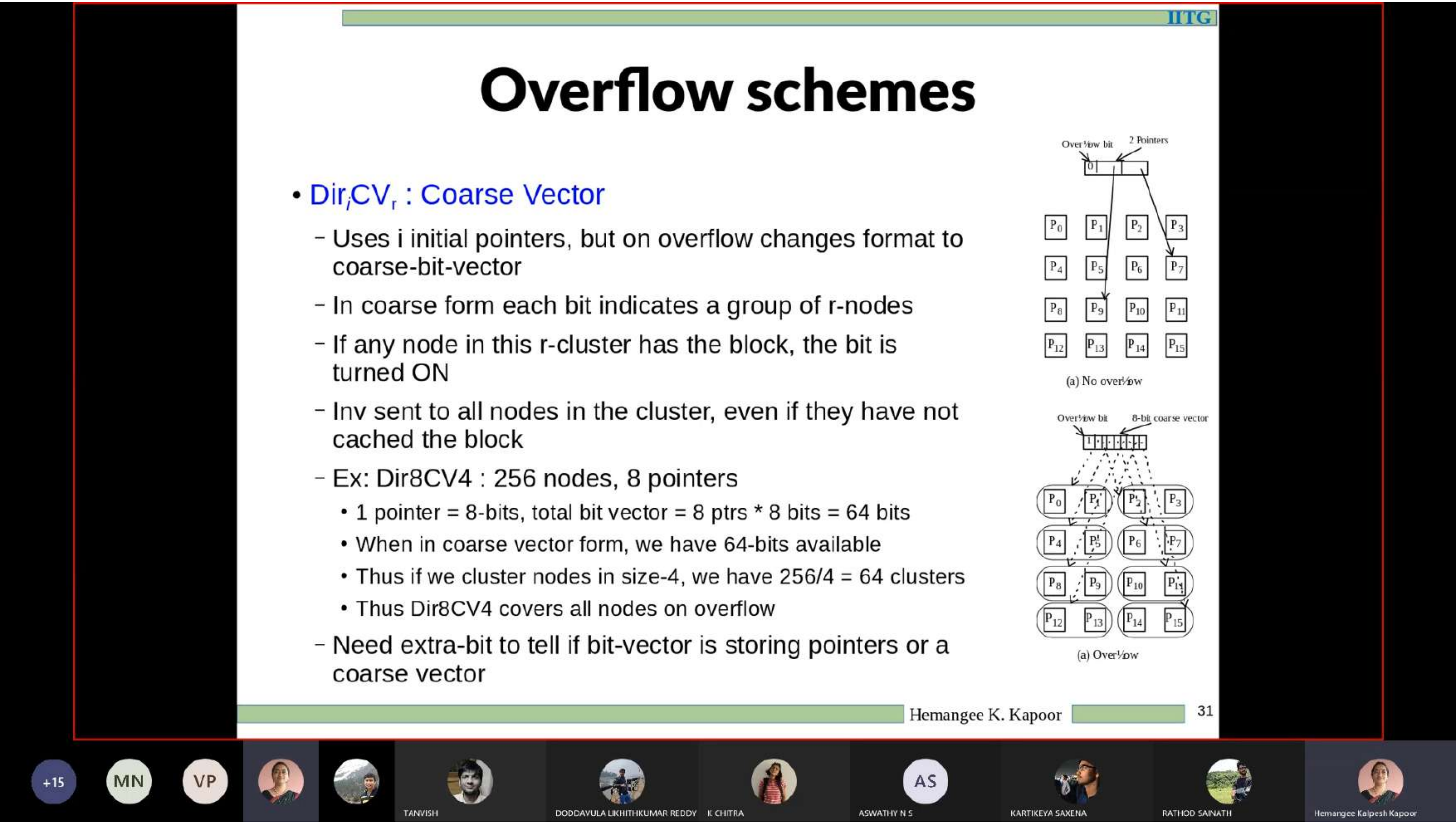
Overflow schemes

- Limtied pointers scheme with ***i*** pointers are named Dir_i followed by abbreviation of their overflow methods



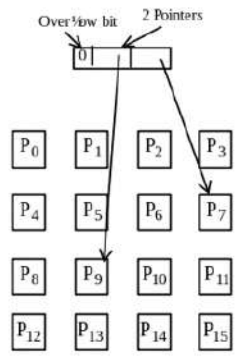
Overflow schemes

- Dir$_i$B : Broadcast
  - Set a broadcast bit in the directory entry when the number of sharers exceeds i
  - Send inv to sharers in i if sharers <=i
  - If sharers > i then broadcast inv (i.e to all nodes)
  - Semantically right
  - Wastes bandwidth
  - Bad for widely-shared frequently read data (i.e. not many writes -> inv)
- Dir$_i$NB : no-broadcast
  - On overflow new sharer replaces one of the old sharers
  - Bad for widely read data as the new readers will keep invalidating each other

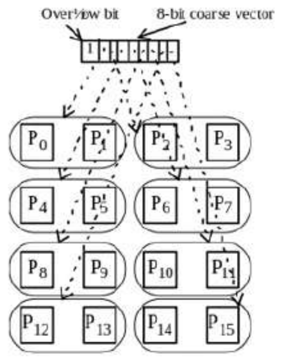


Overflow schemes

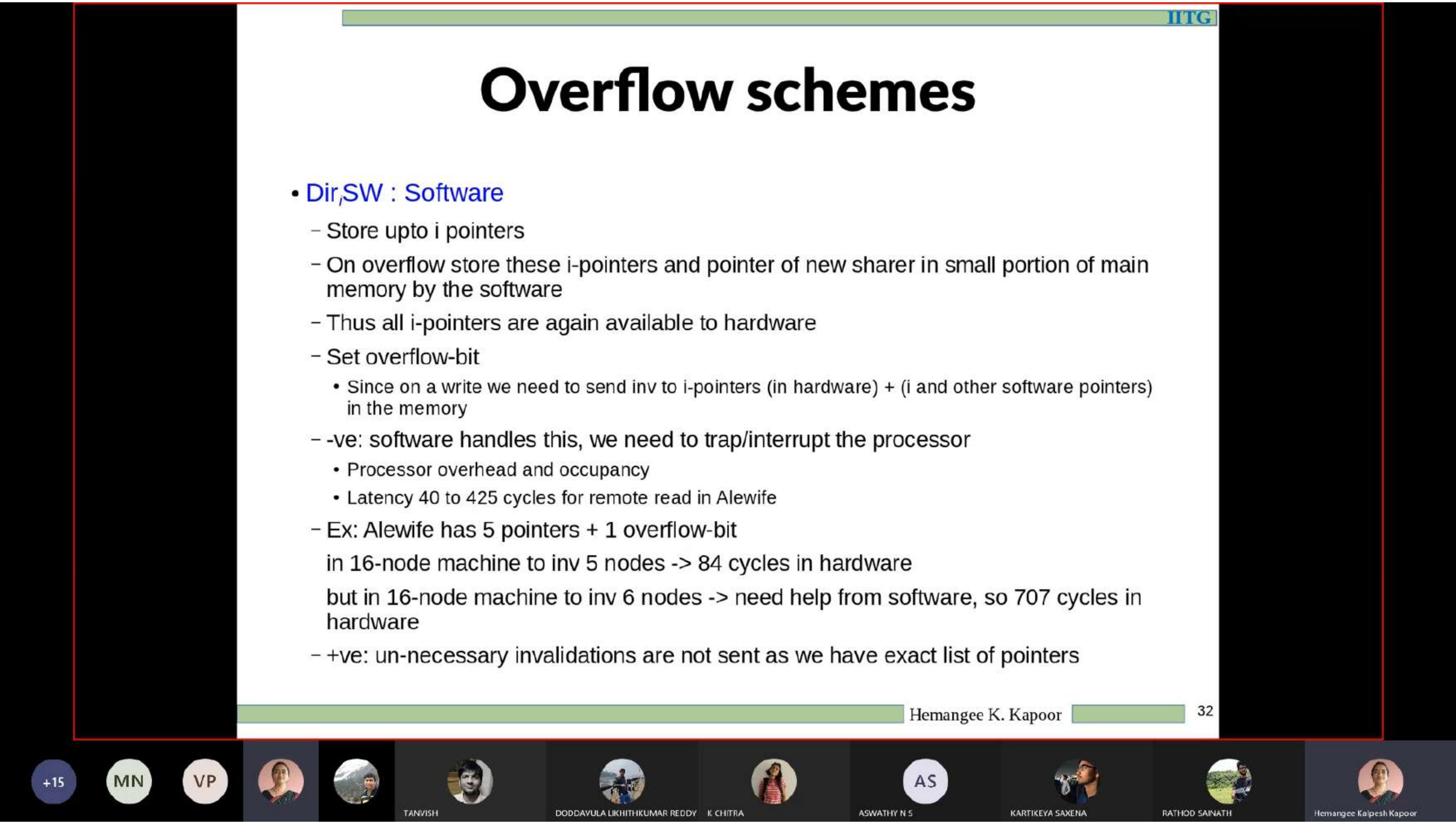
- Dir$_i$CV$_r$: Coarse Vector
  - Uses i initial pointers, but on overflow changes format to coarse-bit-vector
  - In coarse form each bit indicates a group of r-nodes
  - If any node in this r-cluster has the block, the bit is turned ON
  - Inv sent to all nodes in the cluster, even if they have not cached the block
  - Ex: Dir8CV4 : 256 nodes, 8 pointers
    - 1 pointer = 8-bits, total bit vector = 8 ptrs * 8 bits = 64 bits
    - When in coarse vector form, we have 64-bits available
    - Thus if we cluster nodes in size-4, we have 256/4 = 64 clusters
    - Thus Dir8CV4 covers all nodes on overflow
  - Need extra-bit to tell if bit-vector is storing pointers or a coarse vector



(a) No overflow



(a) Overflow

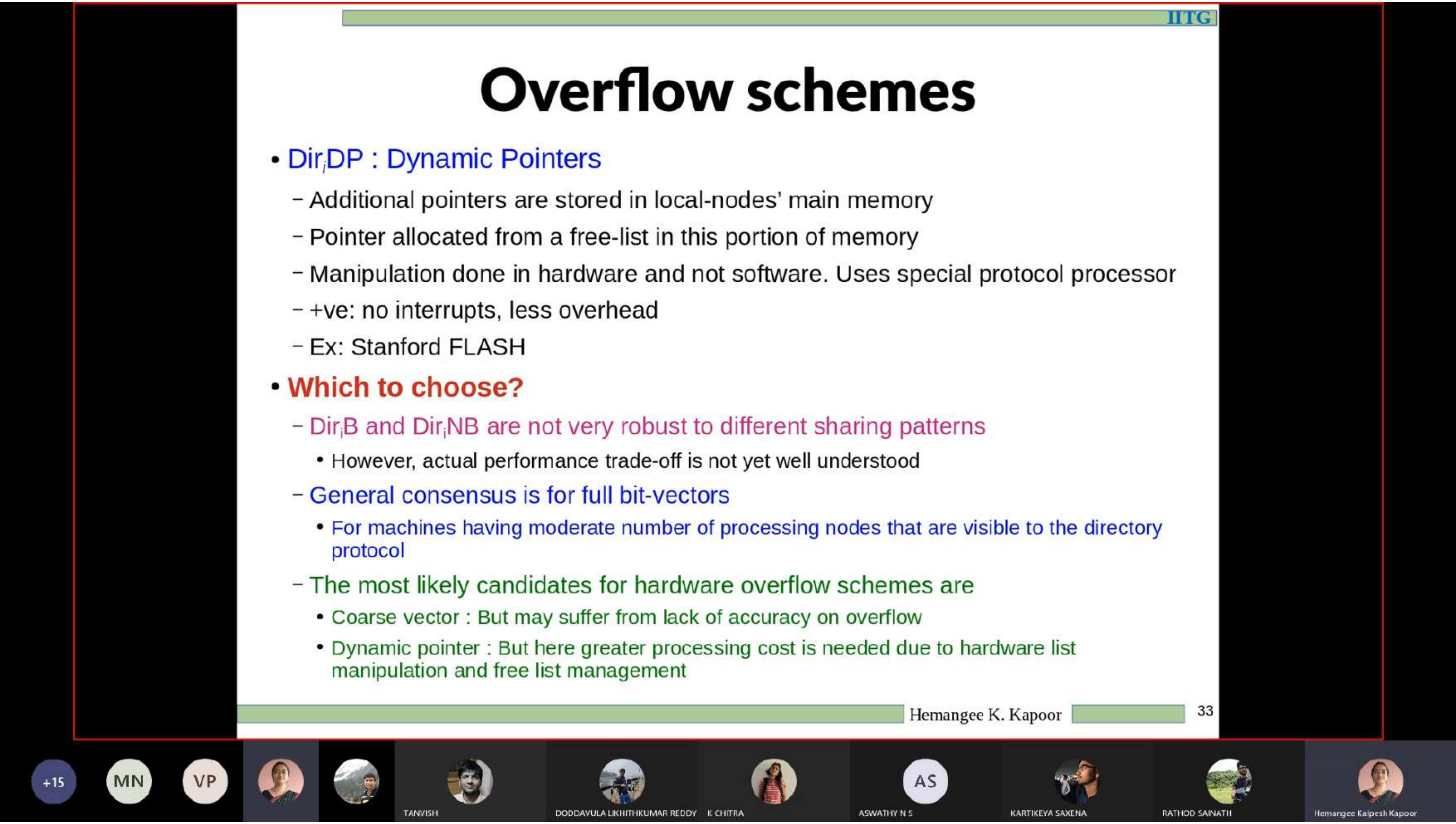


Overflow schemes

- Dir$_i$SW : Software
  - Store upto i pointers
  - On overflow store these i-pointers and pointer of new sharer in small portion of main memory by the software
  - Thus all i-pointers are again available to hardware
  - Set overflow-bit
    - Since on a write we need to send inv to i-pointers (in hardware) + (i and other software pointers) in the memory
  - -ve: software handles this, we need to trap/interrupt the processor
    - Processor overhead and occupancy
    - Latency 40 to 425 cycles for remote read in Alewife
  - Ex: Alewife has 5 pointers + 1 overflow-bit

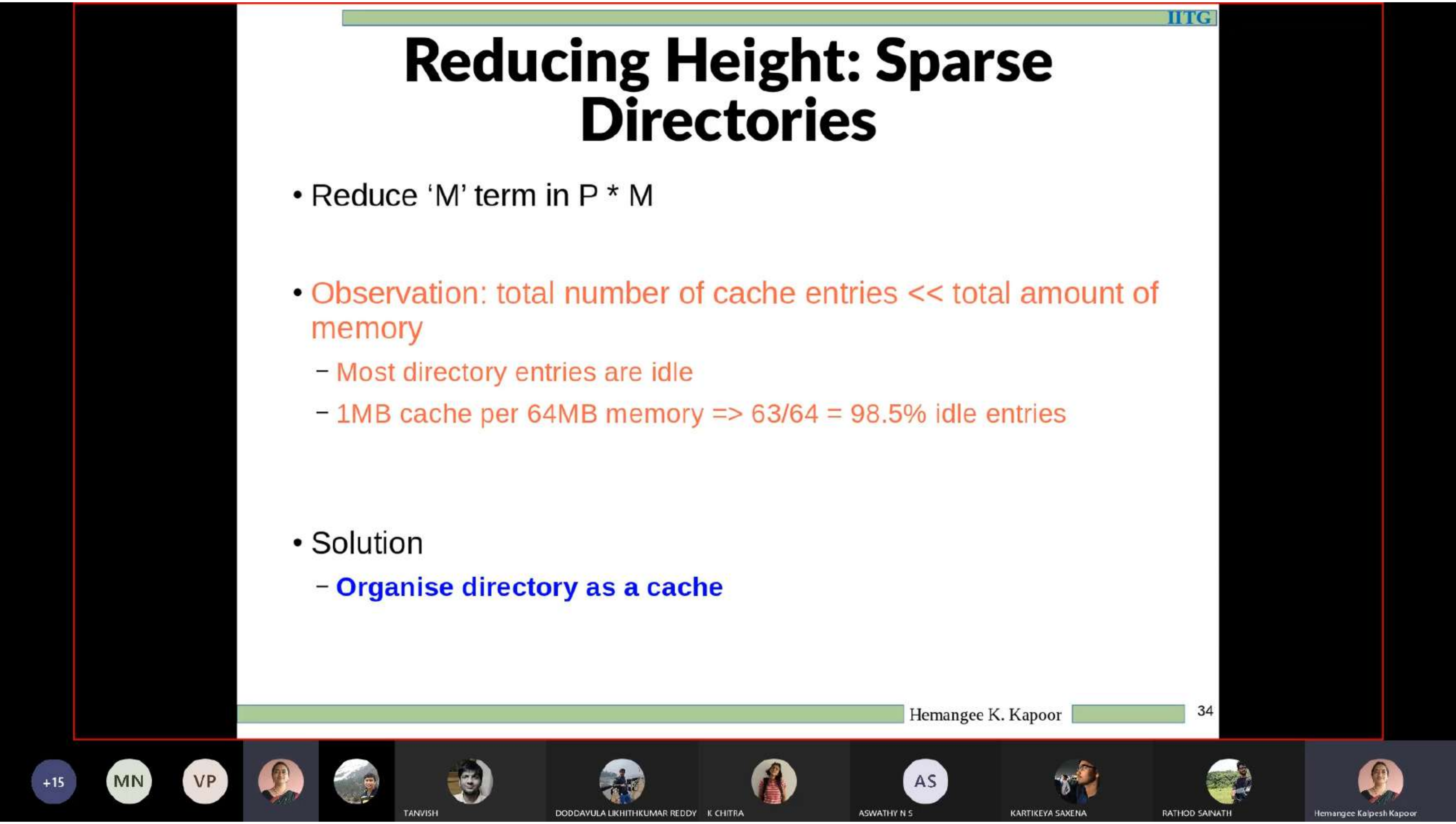
    in 16-node machine to inv 5 nodes -> 84 cycles in hardware

    but in 16-node machine to inv 6 nodes -> need help from software, so 707 cycles in hardware
  - +ve: un-necessary invalidations are not sent as we have exact list of pointers



Overflow schemes

- $Dir_i DP$: Dynamic Pointers
  - Additional pointers are stored in local-nodes' main memory
  - Pointer allocated from a free-list in this portion of memory
  - Manipulation done in hardware and not software. Uses special protocol processor
  - +ve: no interrupts, less overhead
  - Ex: Stanford FLASH
- **Which to choose?**
  - $Dir_i B$ and $Dir_i NB$ are not very robust to different sharing patterns
    - However, actual performance trade-off is not yet well understood
  - General consensus is for full bit-vectors
    - For machines having moderate number of processing nodes that are visible to the directory protocol
  - The most likely candidates for hardware overflow schemes are
    - Coarse vector : But may suffer from lack of accuracy on overflow
    - Dynamic pointer : But here greater processing cost is needed due to hardware list manipulation and free list management

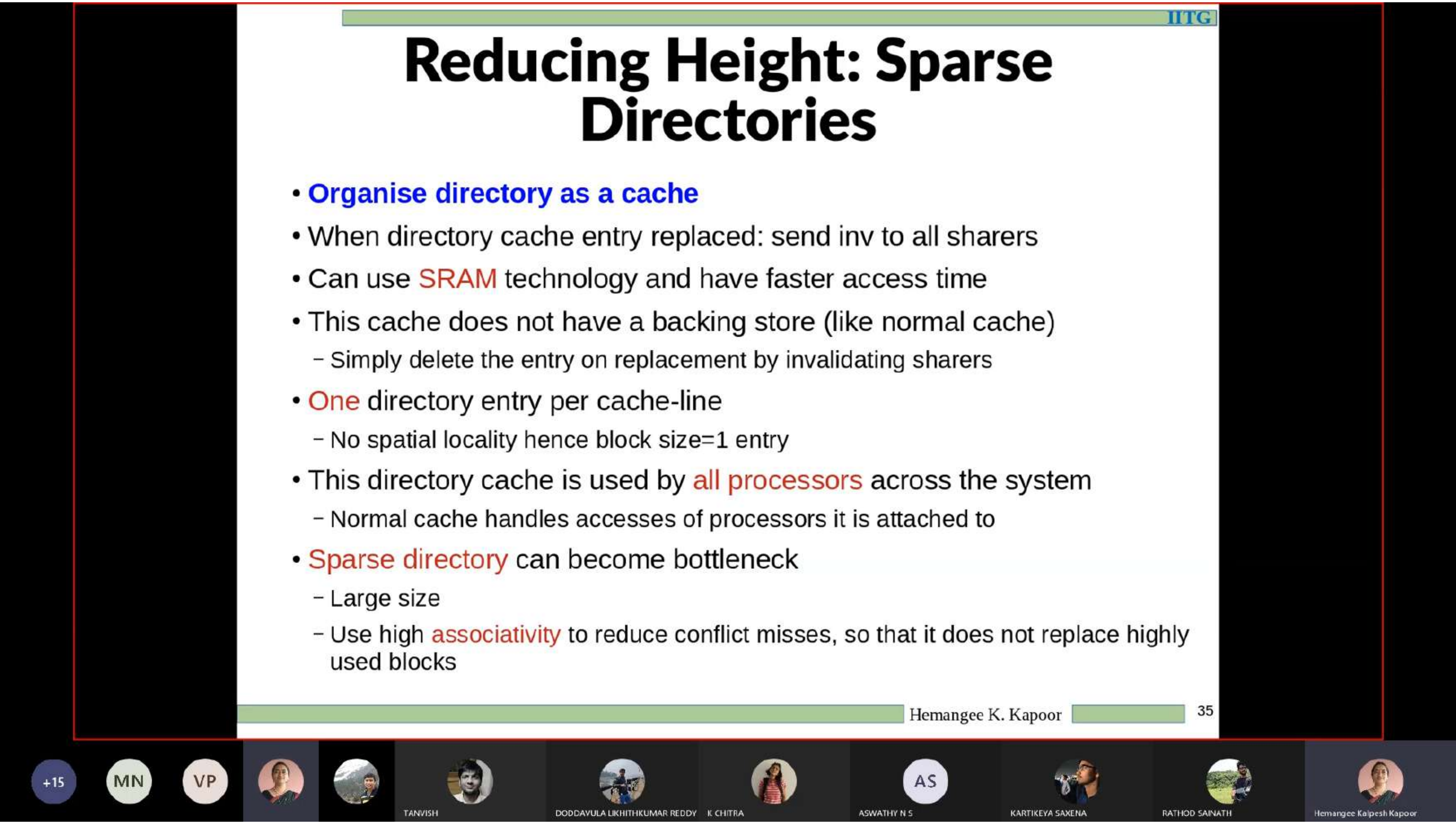


Reducing Height: Sparse Directories

- Reduce 'M' term in P * M

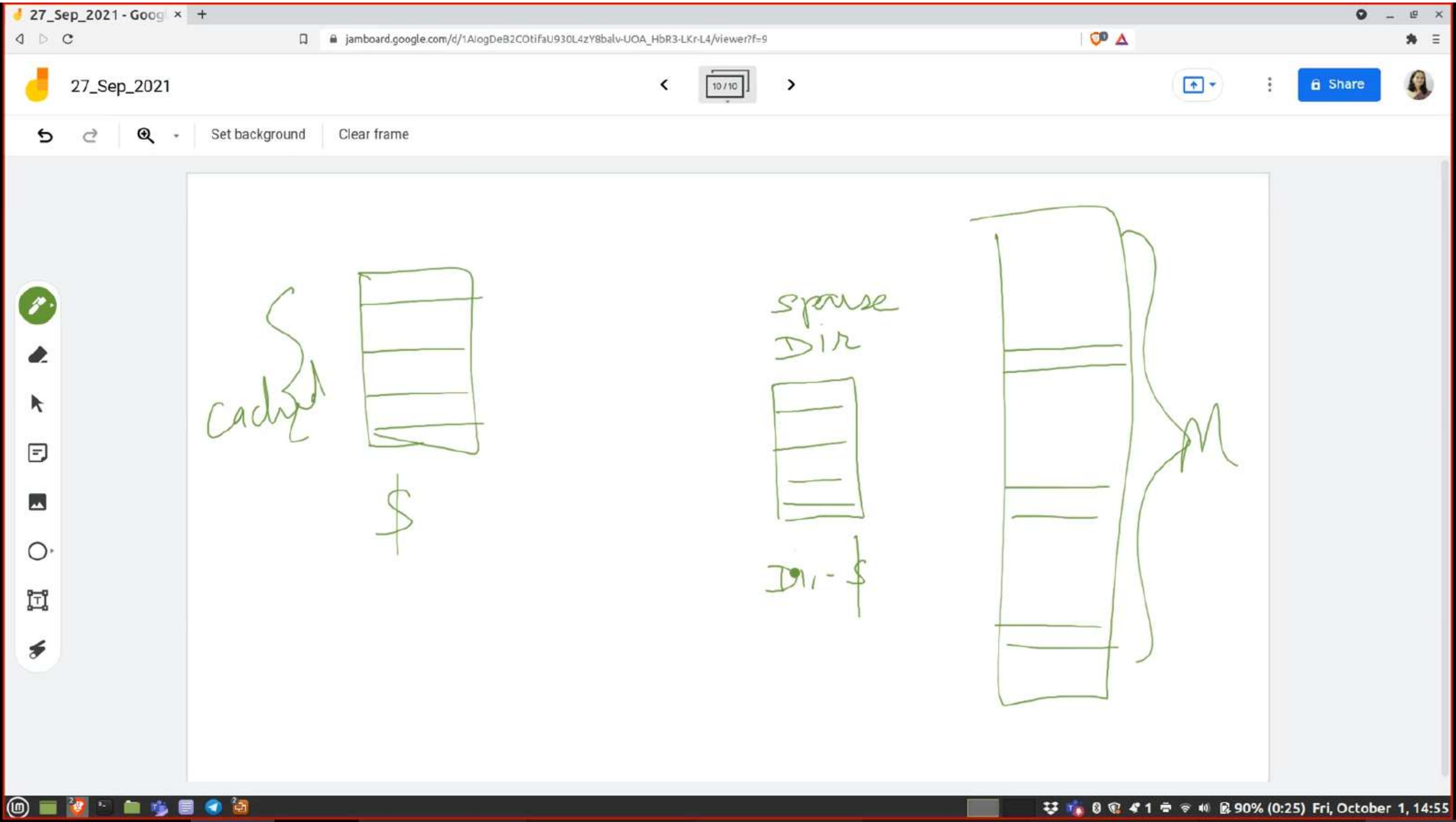
- Observation: total number of cache entries << total amount of memory
  - Most directory entries are idle
  - 1MB cache per 64MB memory => 63/64 = 98.5% idle entries

- Solution
  - **Organise directory as a cache**



Reducing Height: Sparse Directories

- **Organise directory as a cache**
- When directory cache entry replaced: send inv to all sharers
- Can use SRAM technology and have faster access time
- This cache does not have a backing store (like normal cache)
  - Simply delete the entry on replacement by invalidating sharers
- One directory entry per cache-line
  - No spatial locality hence block size=1 entry
- This directory cache is used by all processors across the system
  - Normal cache handles accesses of processors it is attached to
- Sparse directory can become bottleneck
  - Large size
  - Use high associativity to reduce conflict misses, so that it does not replace highly used blocks



27_Sep_2021 - Goog

jamboard.google.com/d/1AlogDeB2COtifaU930L4zY8balv-UOA_HbR3-LKr-L4/viewer?f=9

27_Sep_2021

10 / 10

Share

Set background

Clear frame

90% (0:25) Fri, October 1, 14:55