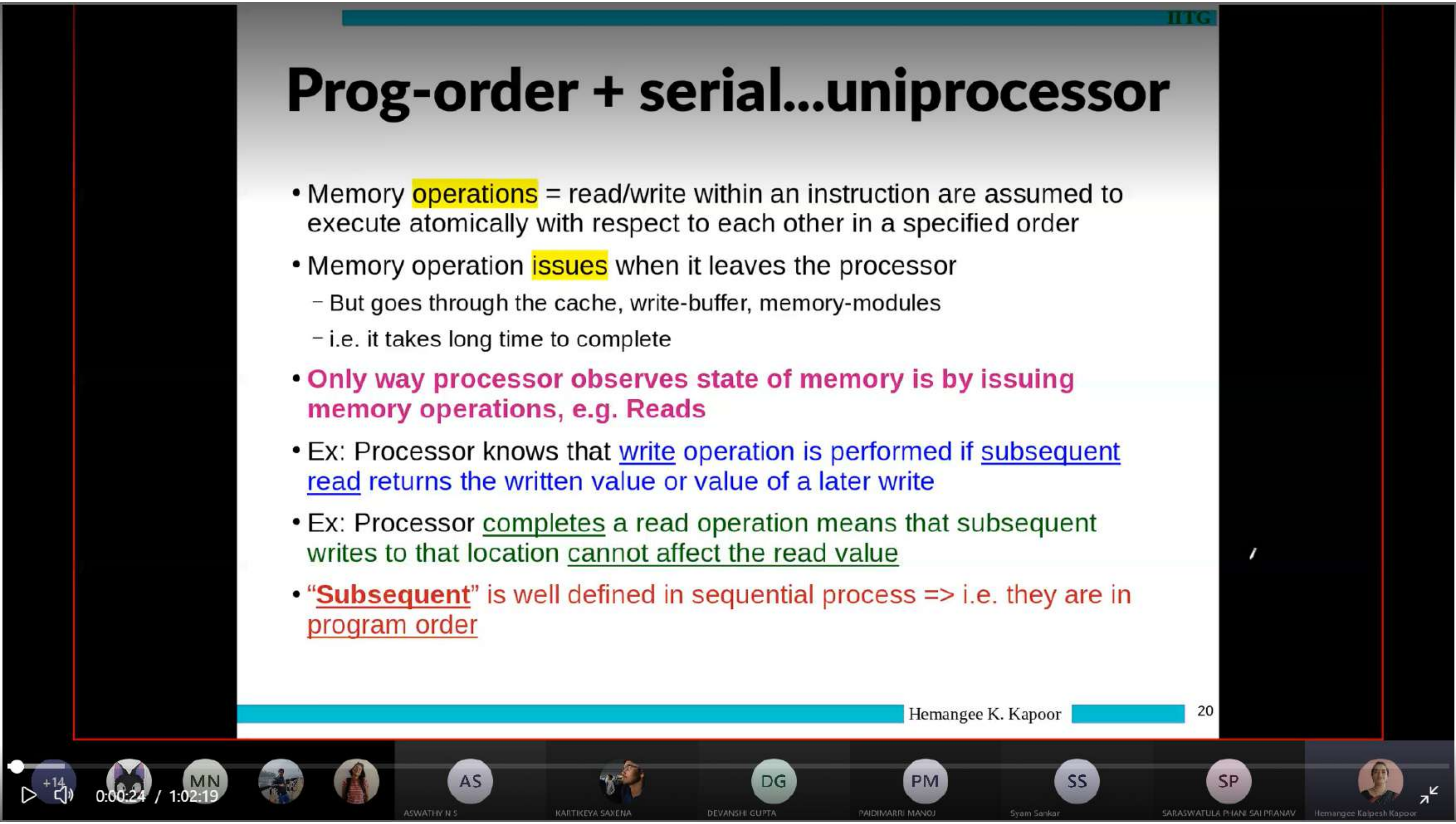
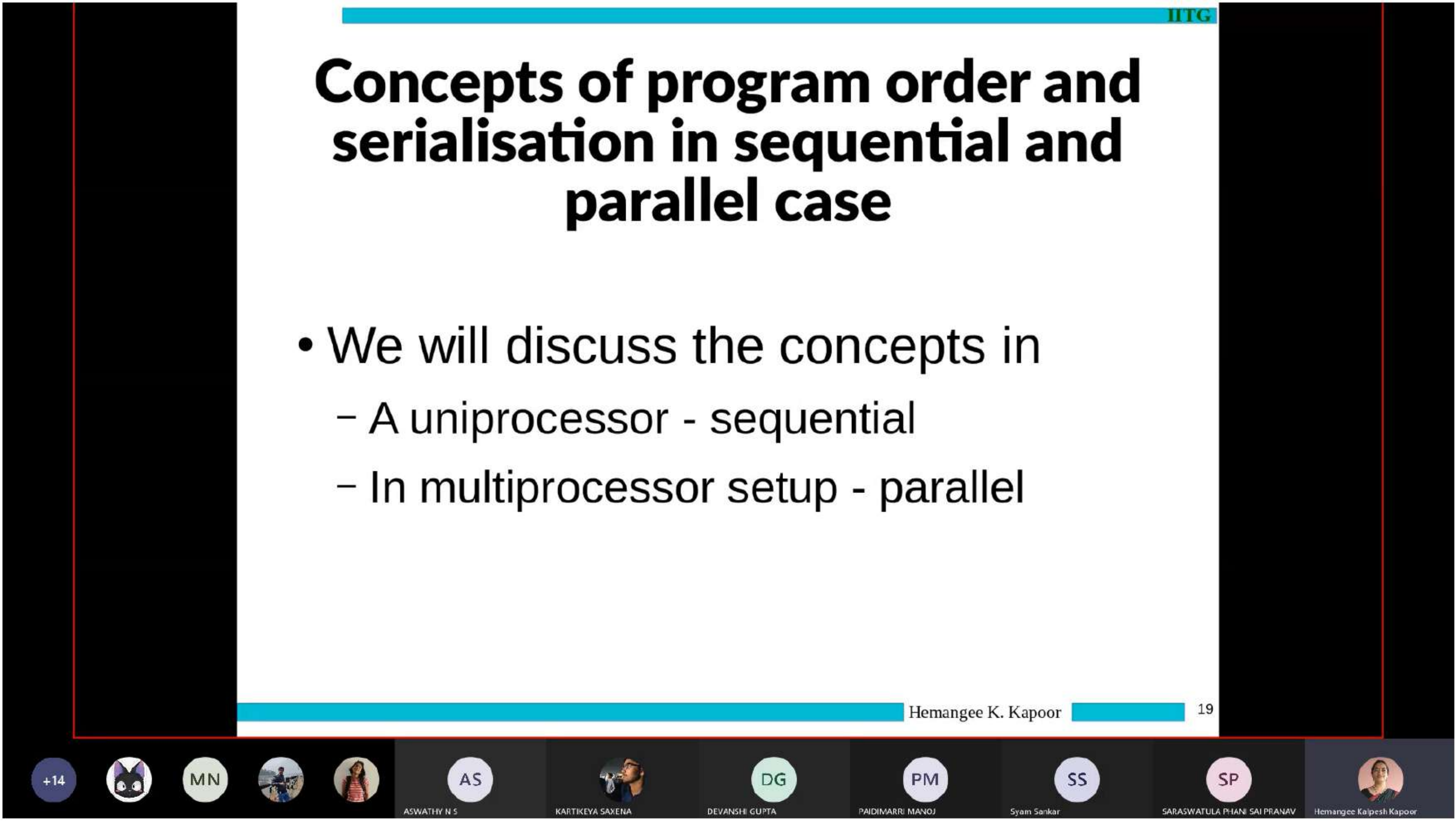


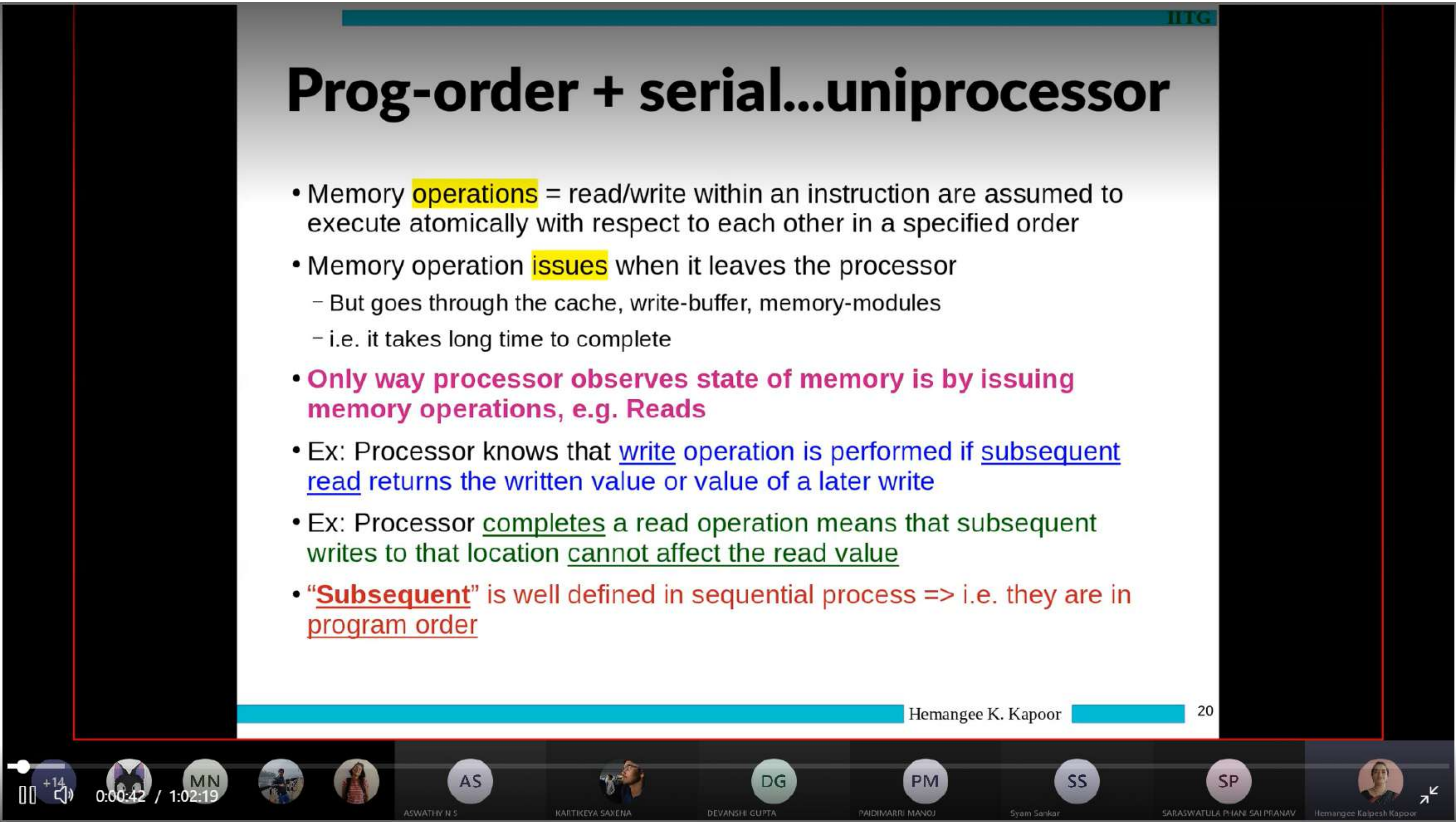
Prog-order + serial...uniprocessor

- Memory operations = read/write within an instruction are assumed to execute atomically with respect to each other in a specified order
- Memory operation issues when it leaves the processor
  - But goes through the cache, write-buffer, memory-modules
  - i.e. it takes long time to complete
- **Only way processor observes state of memory is by issuing memory operations, e.g. Reads**
- Ex: Processor knows that write operation is performed if subsequent read returns the written value or value of a later write
- Ex: Processor completes a read operation means that subsequent writes to that location cannot affect the read value
- "**Subsequent**" is well defined in sequential process => i.e. they are in program order



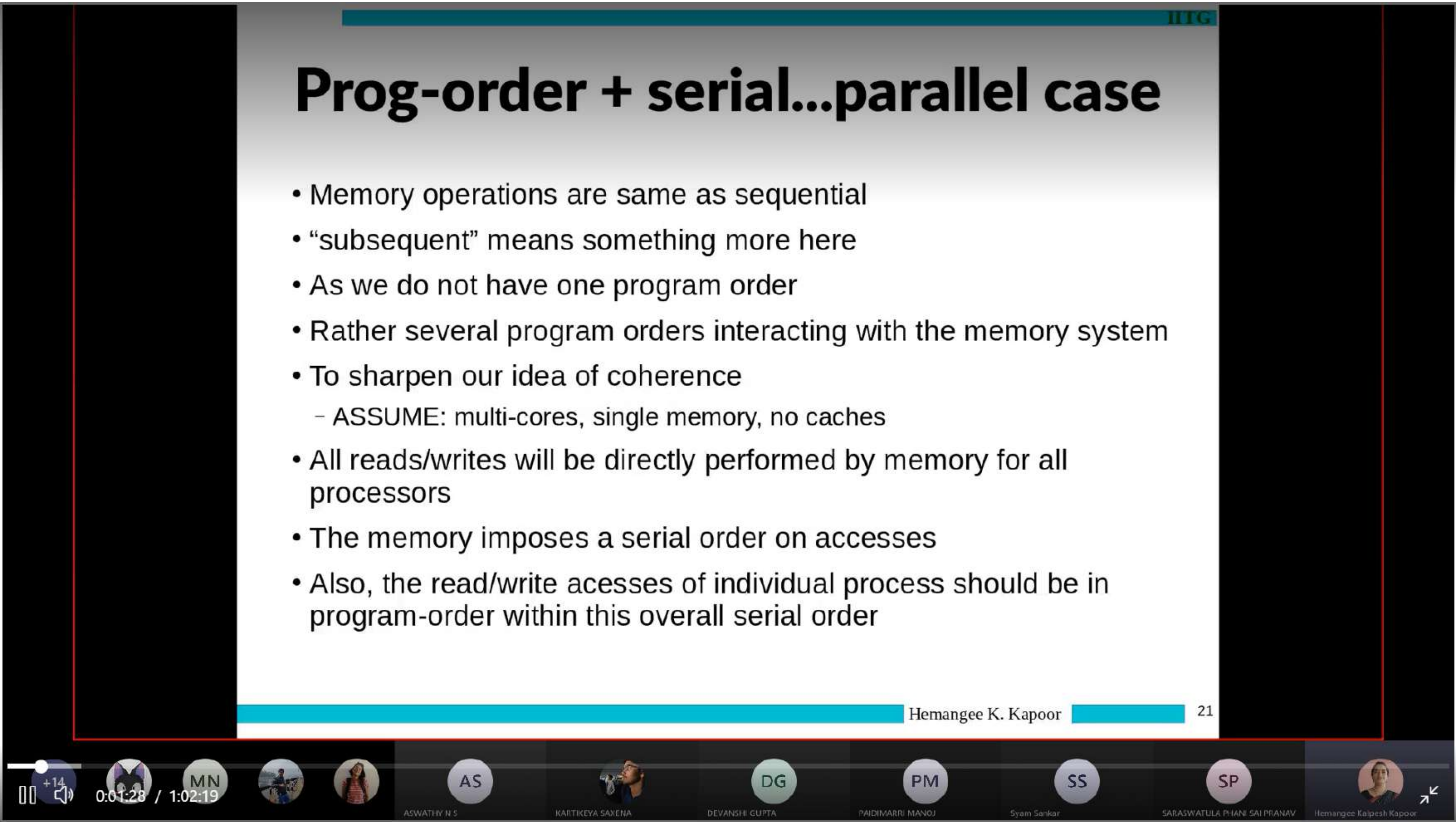
Concepts of program order and serialisation in sequential and parallel case

- We will discuss the concepts in
  - A uniprocessor - sequential
  - In multiprocessor setup - parallel



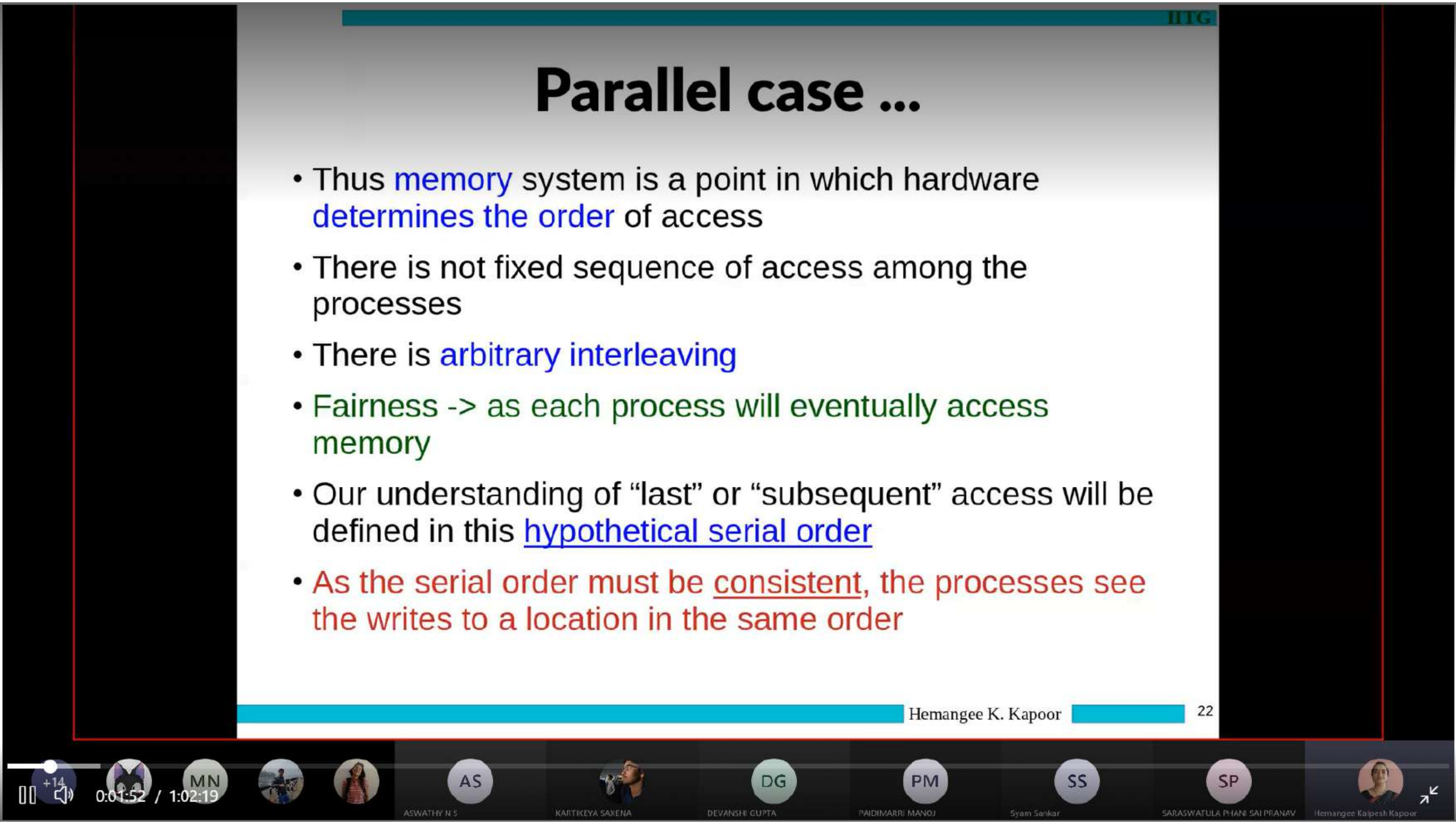
Prog-order + serial...uniprocessor

- Memory operations = read/write within an instruction are assumed to execute atomically with respect to each other in a specified order
- Memory operation issues when it leaves the processor
  - But goes through the cache, write-buffer, memory-modules
  - i.e. it takes long time to complete
- **Only way processor observes state of memory is by issuing memory operations, e.g. Reads**
- Ex: Processor knows that write operation is performed if subsequent read returns the written value or value of a later write
- Ex: Processor completes a read operation means that subsequent writes to that location cannot affect the read value
- "**Subsequent**" is well defined in sequential process => i.e. they are in program order



Prog-order + serial...parallel case

- Memory operations are same as sequential
- “subsequent” means something more here
- As we do not have one program order
- Rather several program orders interacting with the memory system
- To sharpen our idea of coherence
  - ASSUME: multi-cores, single memory, no caches
- All reads/writes will be directly performed by memory for all processors
- The memory imposes a serial order on accesses
- Also, the read/write acesses of individual process should be in program-order within this overall serial order



IITG

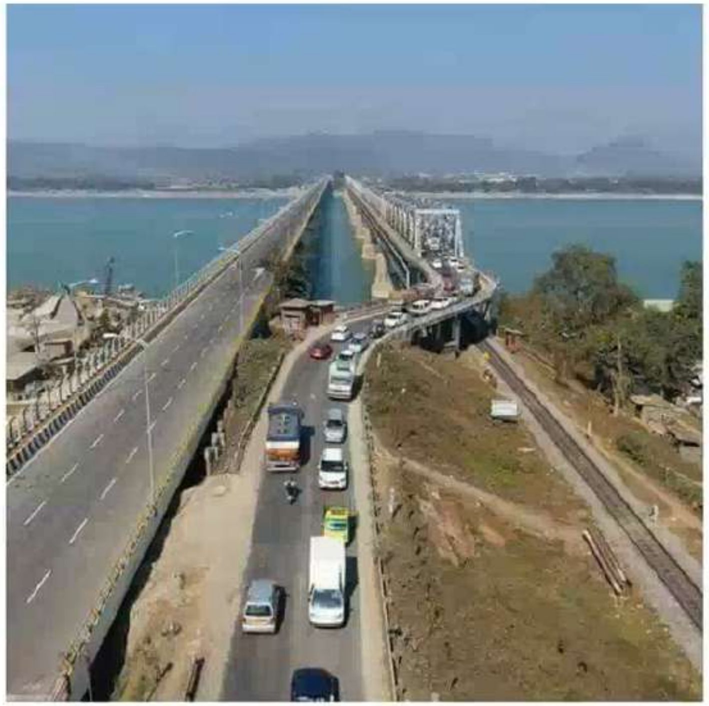
Parallel case ...

- Thus memory system is a point in which hardware determines the order of access
- There is not fixed sequence of access among the processes
- There is arbitrary interleaving
- Fairness -> as each process will eventually access memory
- Our understanding of “last” or “subsequent” access will be defined in this <u>hypothetical serial order</u>
- As the serial order must be <u>consistent</u>, the processes see the writes to a location in the same order

Hemangee K. Kapoor 22

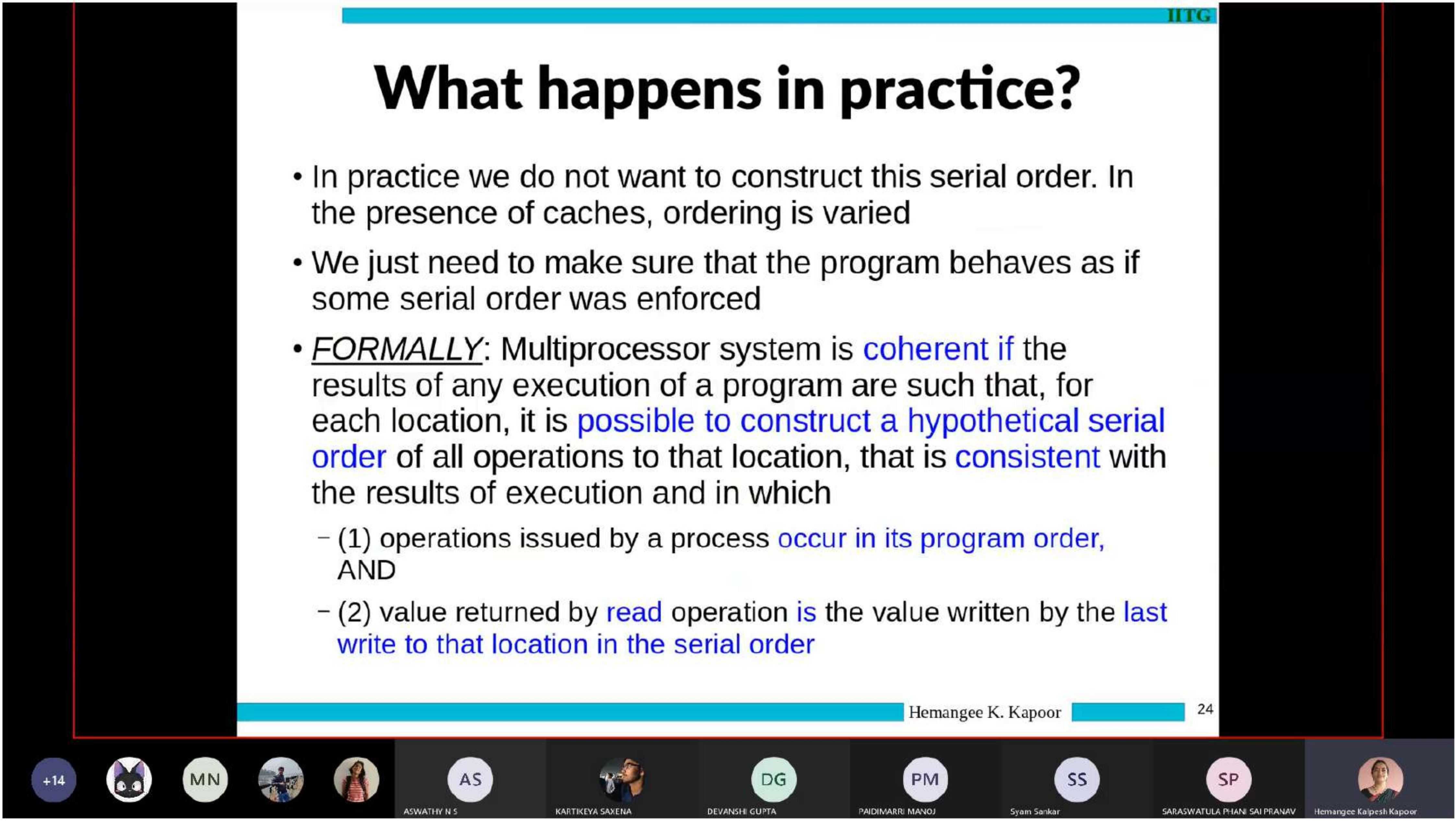
IITG

Merging of cars
=
Merging of mem accesses



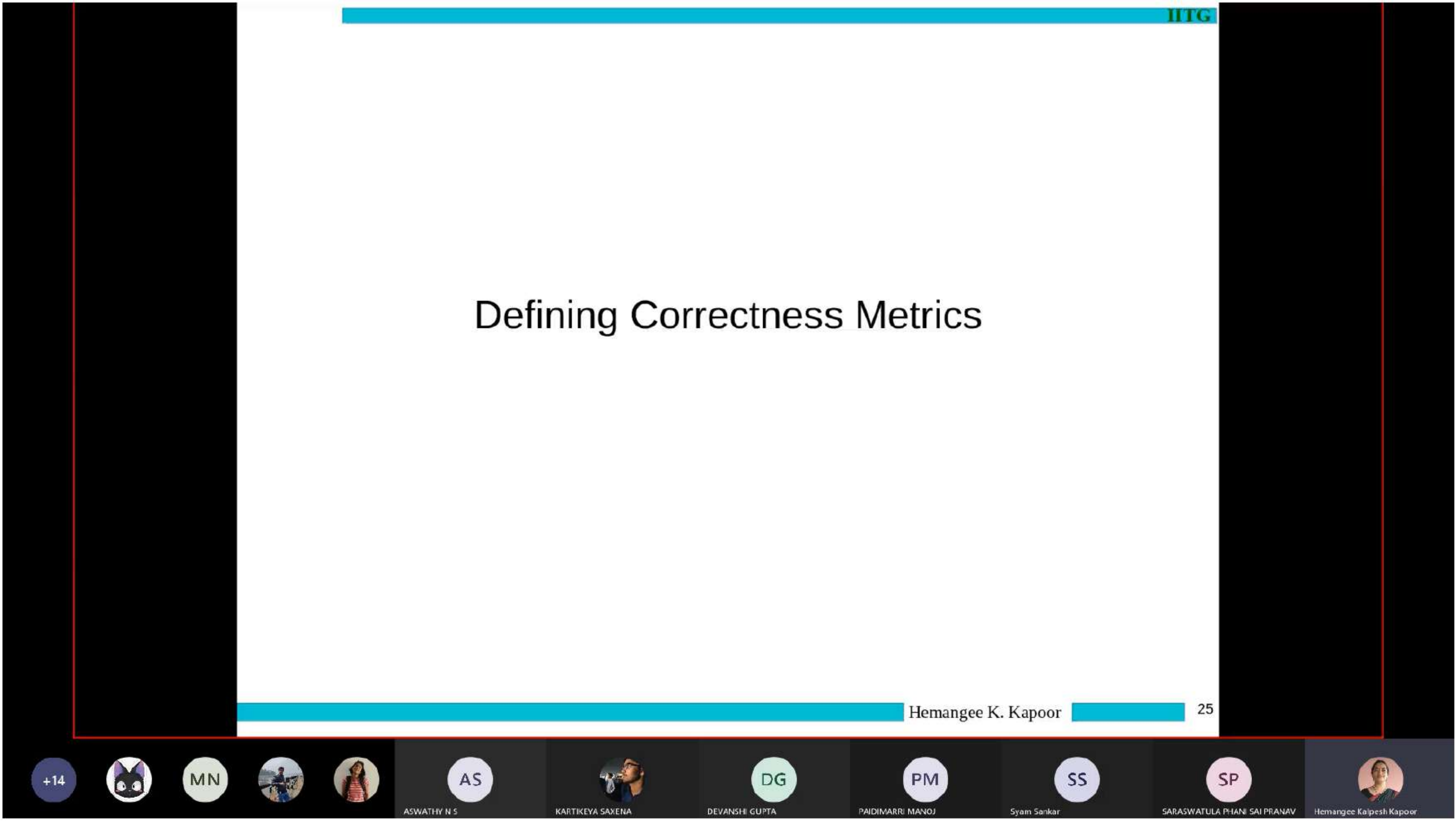
+14 MN AS DG PM SS SP

ASWATHY N S
KARTIKEYA SAXENA
DEVANSHI GUPTA
PAIDIMARRI MANOJ
Syam Sankar
SARASWATULA PHANI SAI PRANAV
Hemangee Kalpesh Kapoor

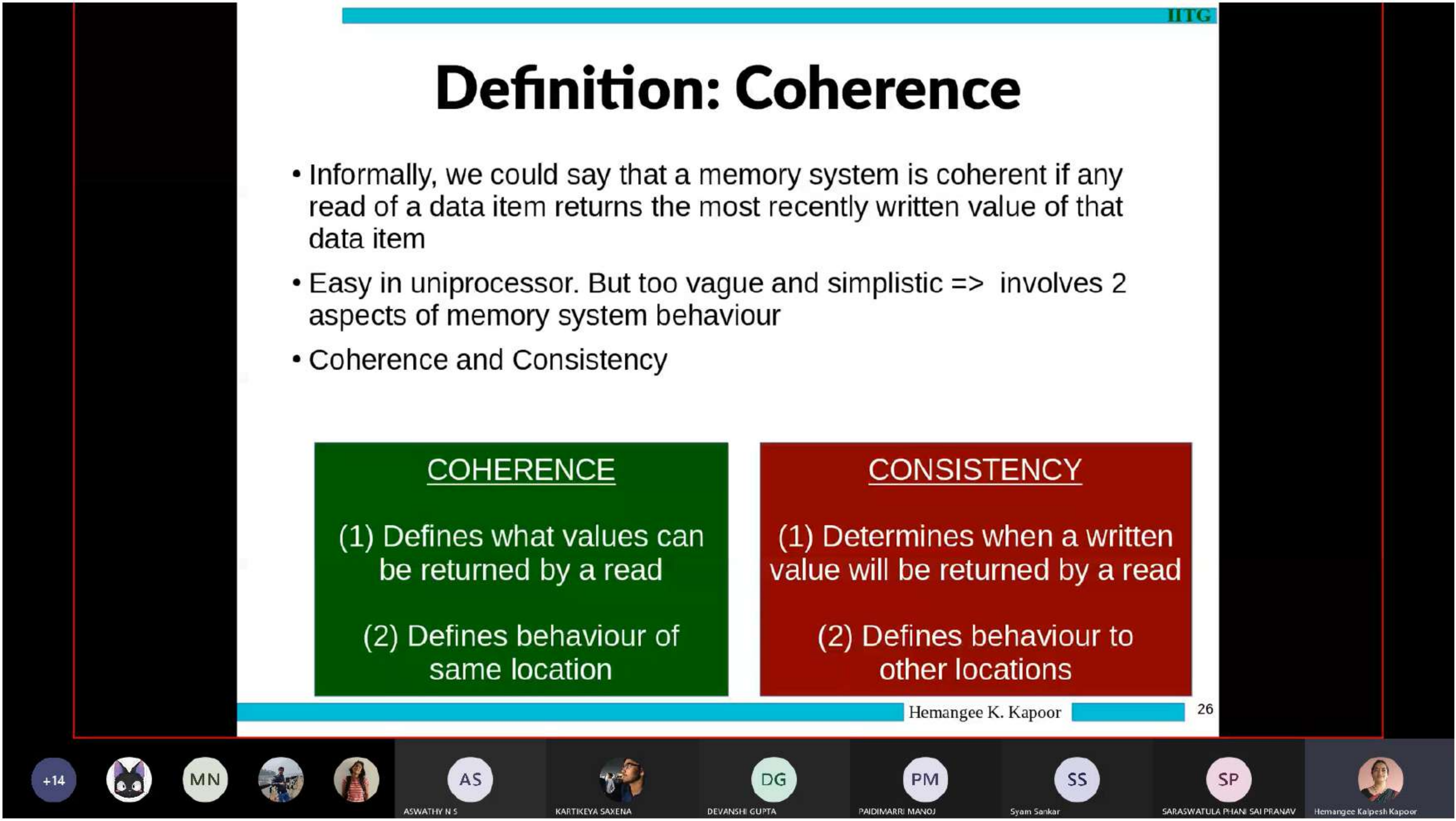


What happens in practice?

- In practice we do not want to construct this serial order. In the presence of caches, ordering is varied
- We just need to make sure that the program behaves as if some serial order was enforced
- *FORMALLY*: Multiprocessor system is coherent if the results of any execution of a program are such that, for each location, it is possible to construct a hypothetical serial order of all operations to that location, that is consistent with the results of execution and in which
  - (1) operations issued by a process occur in its program order, AND
  - (2) value returned by read operation is the value written by the last write to that location in the serial order



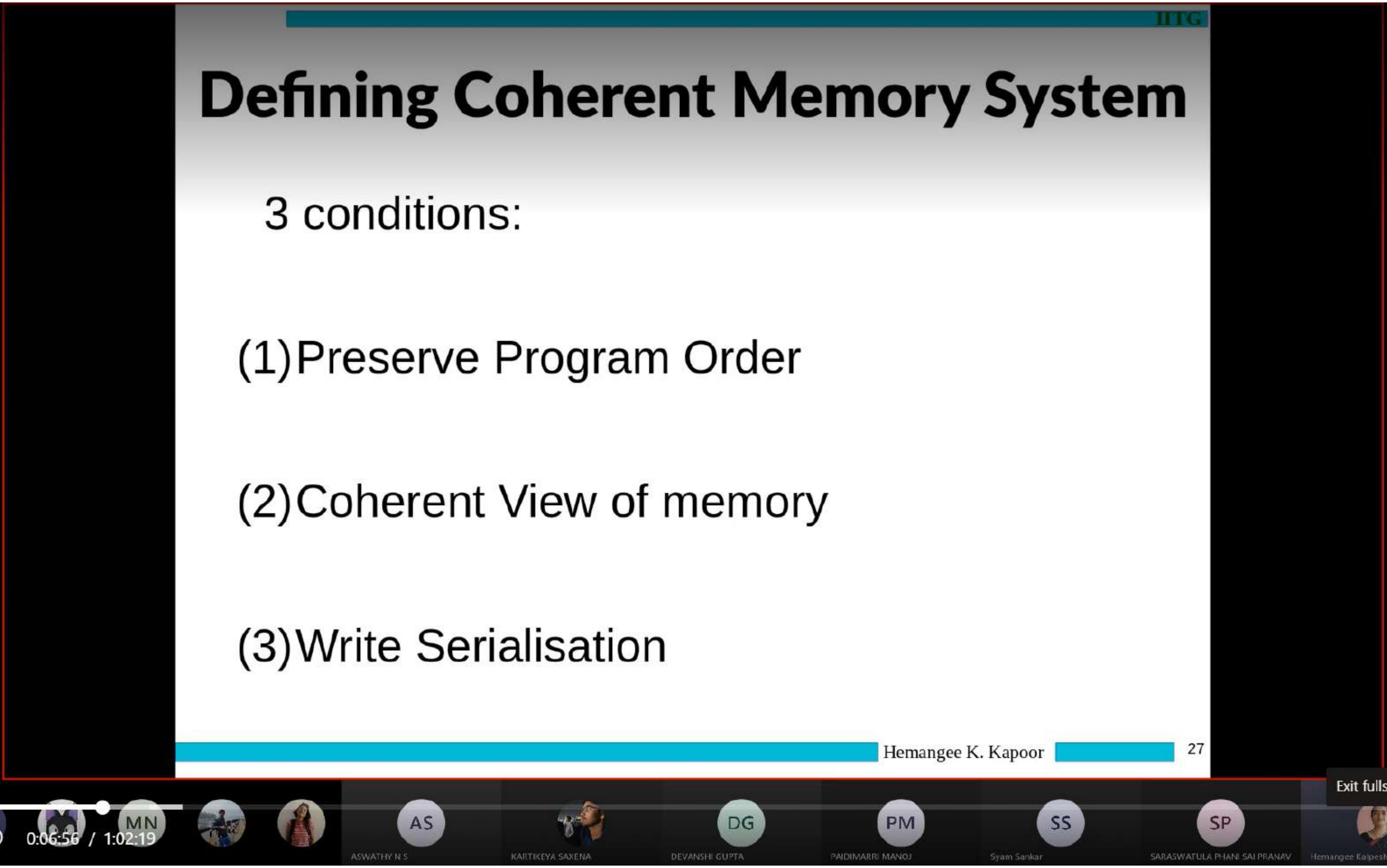
Defining Correctness Metrics



Definition: Coherence

- Informally, we could say that a memory system is coherent if any read of a data item returns the most recently written value of that data item
- Easy in uniprocessor. But too vague and simplistic => involves 2 aspects of memory system behaviour
- Coherence and Consistency

COHERENCE	CONSISTENCY
(1) Defines what values can be returned by a read	(1) Determines when a written value will be returned by a read
(2) Defines behaviour of same location	(2) Defines behaviour to other locations



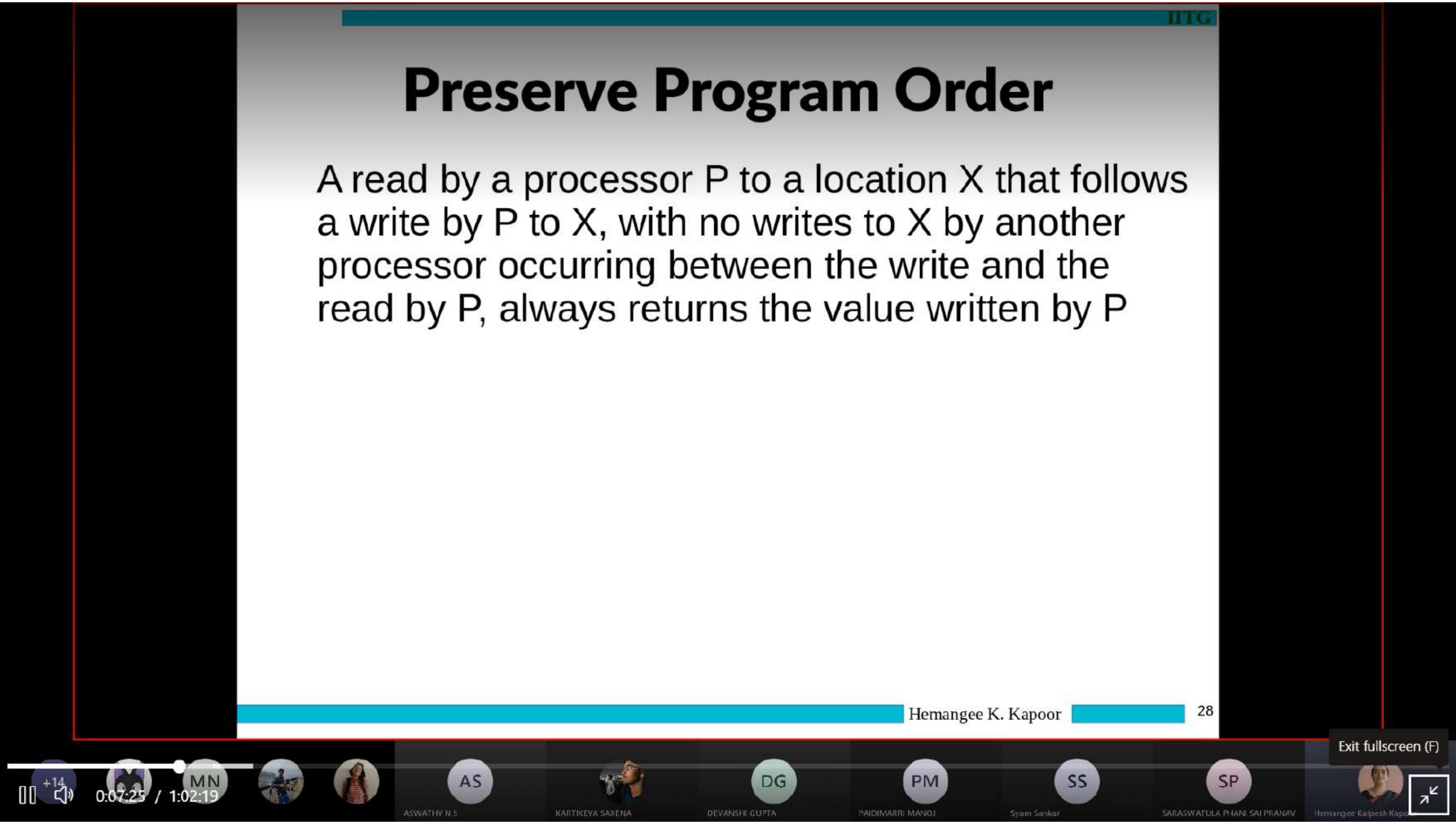
Defining Coherent Memory System

3 conditions:

(1) Preserve Program Order

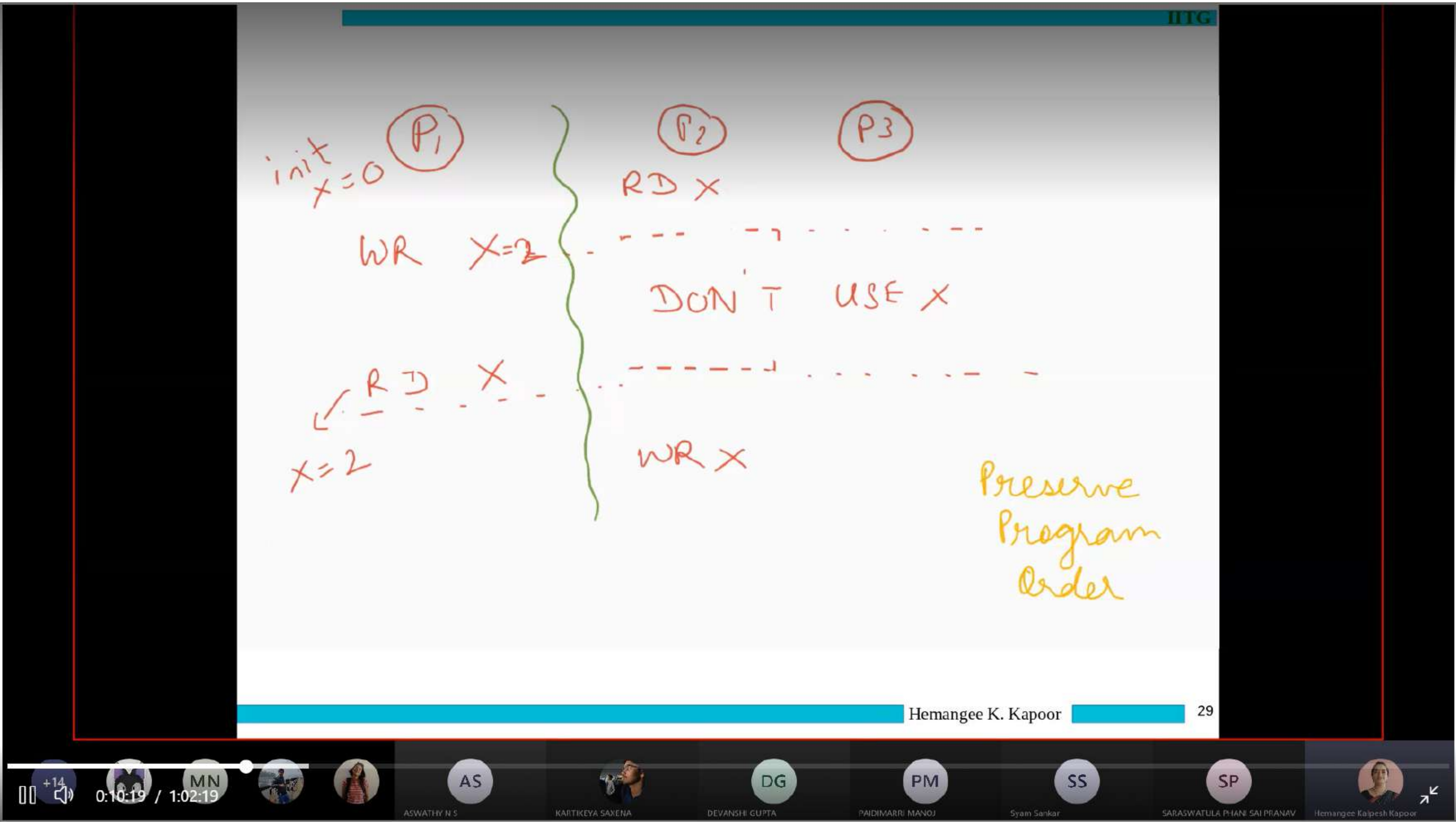
(2) Coherent View of memory

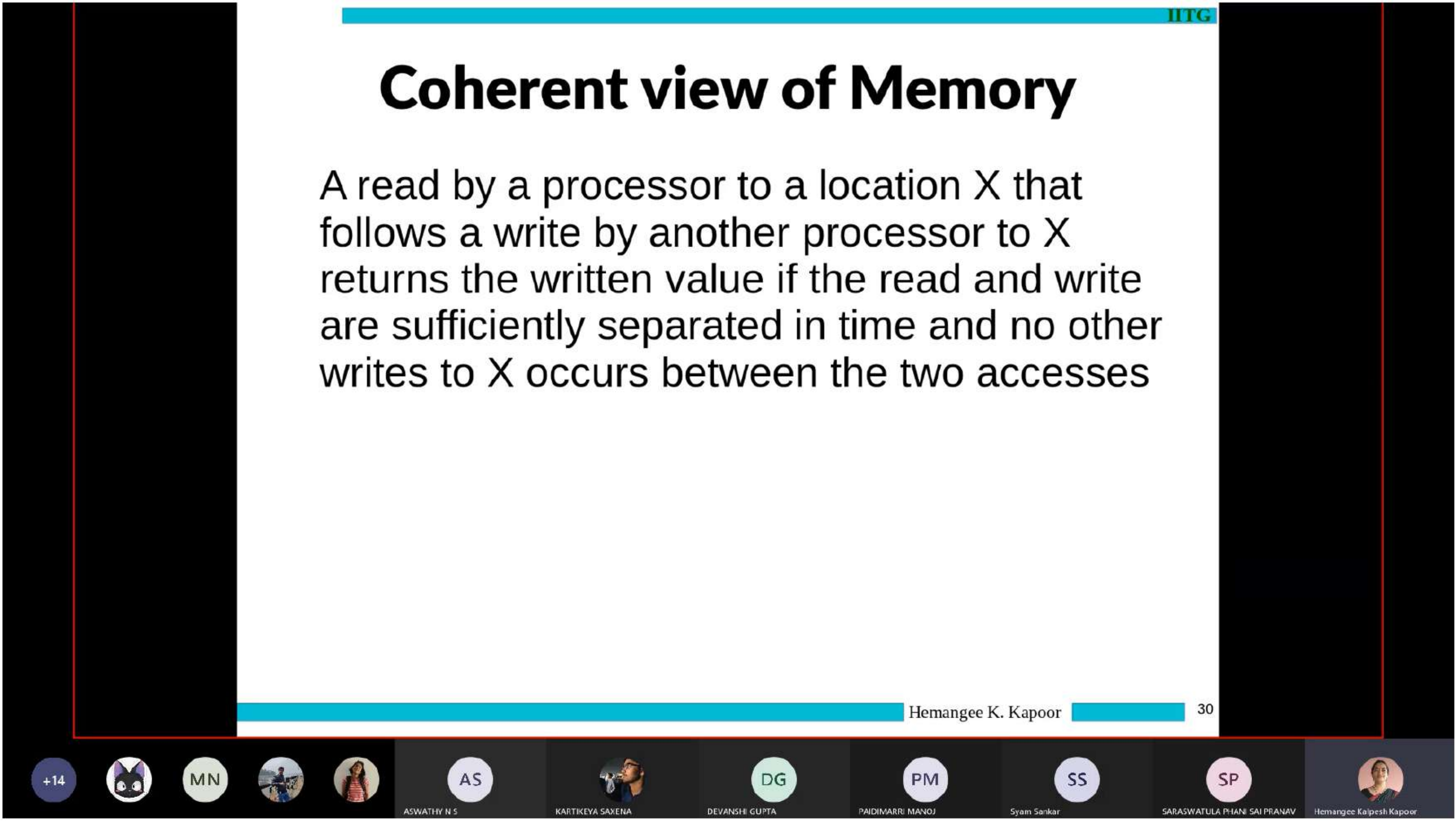
(3) Write Serialisation



Preserve Program Order

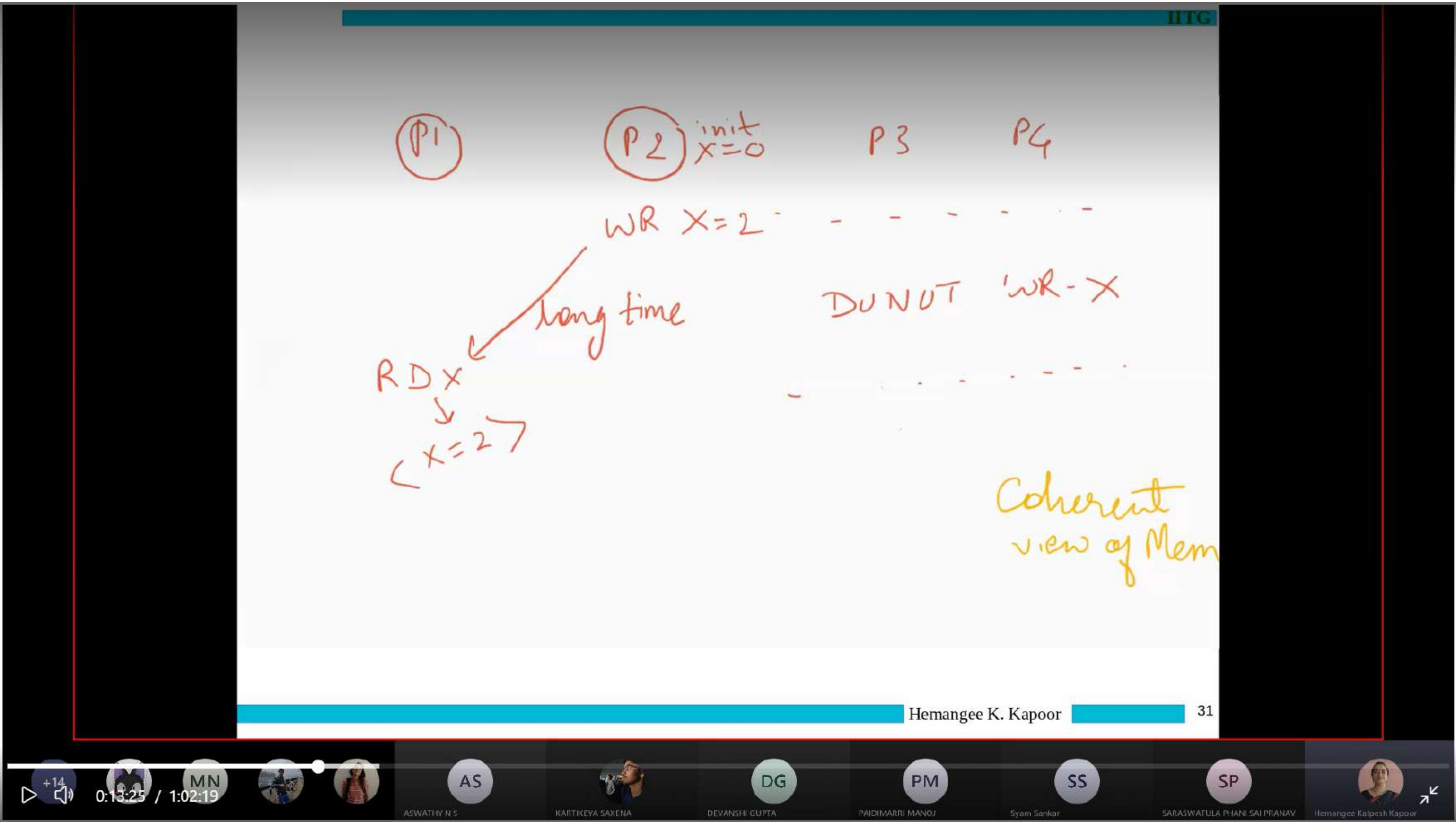
A read by a processor P to a location X that follows a write by P to X, with no writes to X by another processor occurring between the write and the read by P, always returns the value written by P


IITG
init
x=0
P1
P2
P3
RD X
WR X=2
DON'T USE X
RD X
x=2
WR X
Preserve Program Order
Hemangee K. Kapoor
29
0:10:19 / 1:02:19
+14
MN
AS
DG
PM
SS
SP
ASWATHY N S
KARTIKEYA SAXENA
DEVANSHI GUPTA
PAIDIMARRI MANOJ
Syam Sankar
SARASWATULA PHANI SAI PRANAV
Hemangee Kalpesh Kapoor



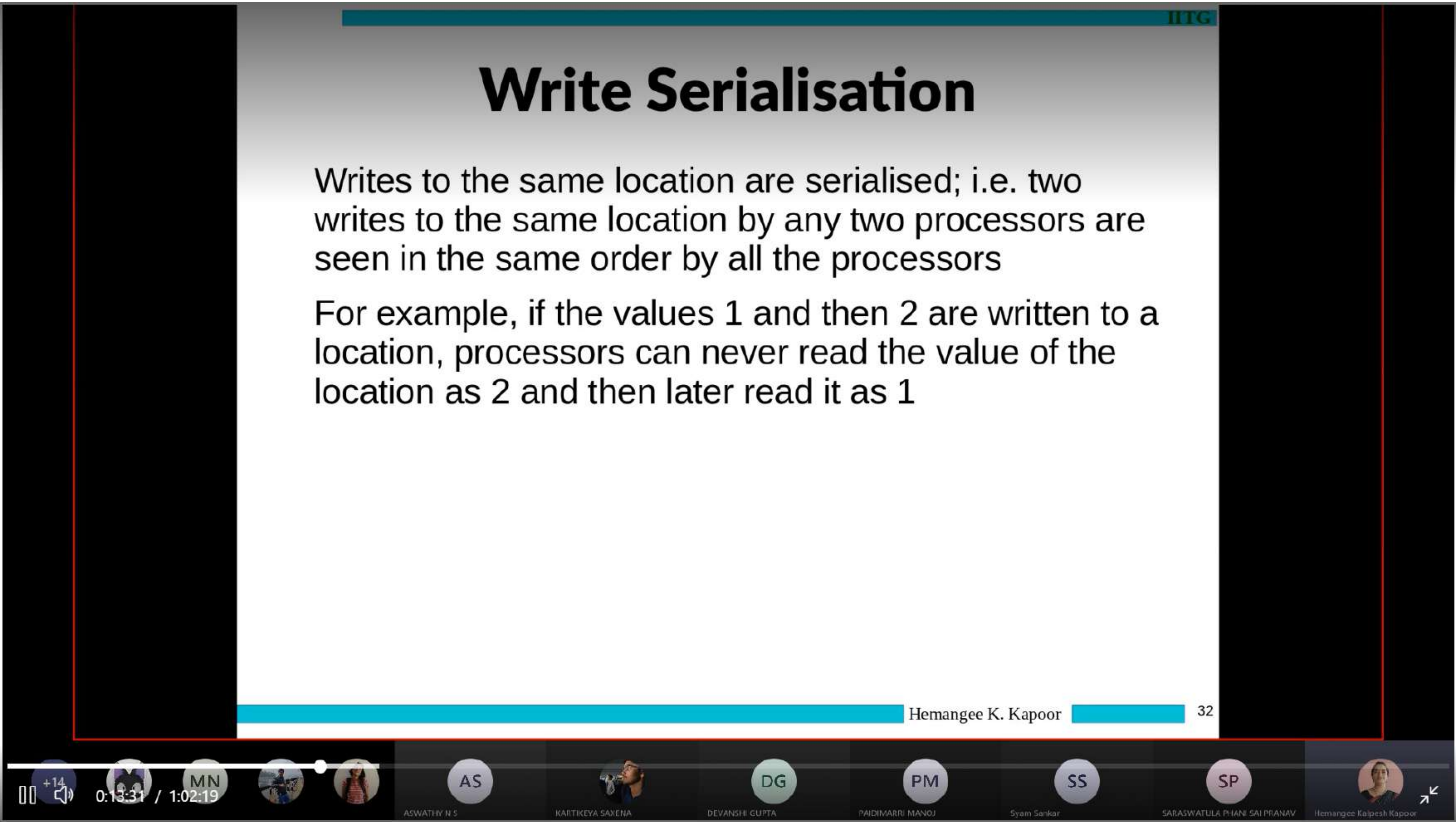
Coherent view of Memory

A read by a processor to a location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occurs between the two accesses



IITG

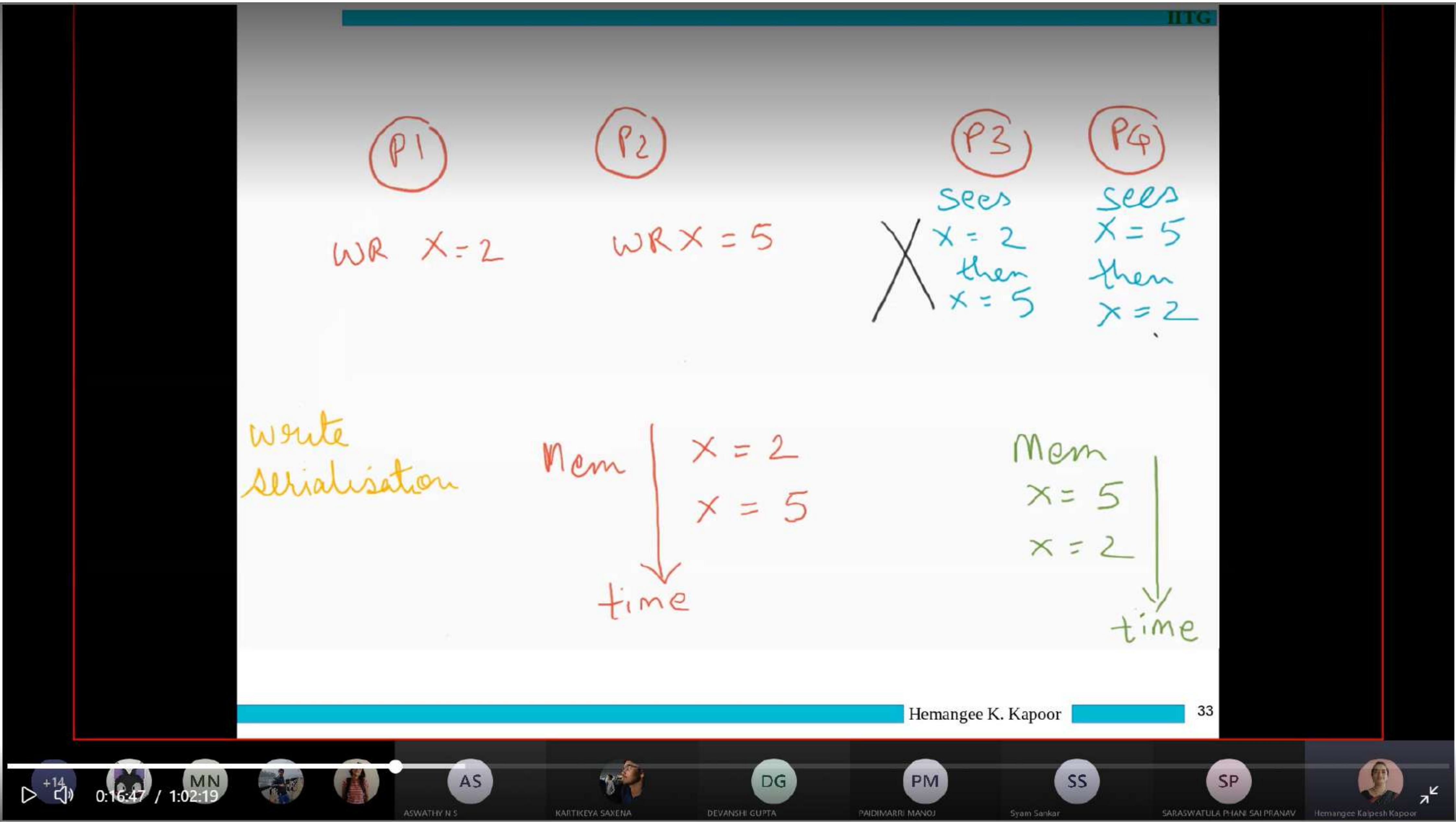
P1
P2
init
x=0
P3
P4
WR X=2
long time
RD X
<x=2>
DUNOT 'WR-X
Coherent
view of Mem
0:13:25 / 1:02:19
+14
MN
AS
DG
PM
SS
SP
ASWATHY N S
KARTIKEYA SAXENA
DEVANSHI GUPTA
PAIDIMARRI MANOJ
Syam Sankar
SARASWATULA PHANI SAI PRANAV
Hemangee Kalpesh Kapoor


IITG

Write Serialisation

Writes to the same location are serialised; i.e. two writes to the same location by any two processors are seen in the same order by all the processors

For example, if the values 1 and then 2 are written to a location, processors can never read the value of the location as 2 and then later read it as 1



P1 WR X=2

P2 WR X=5

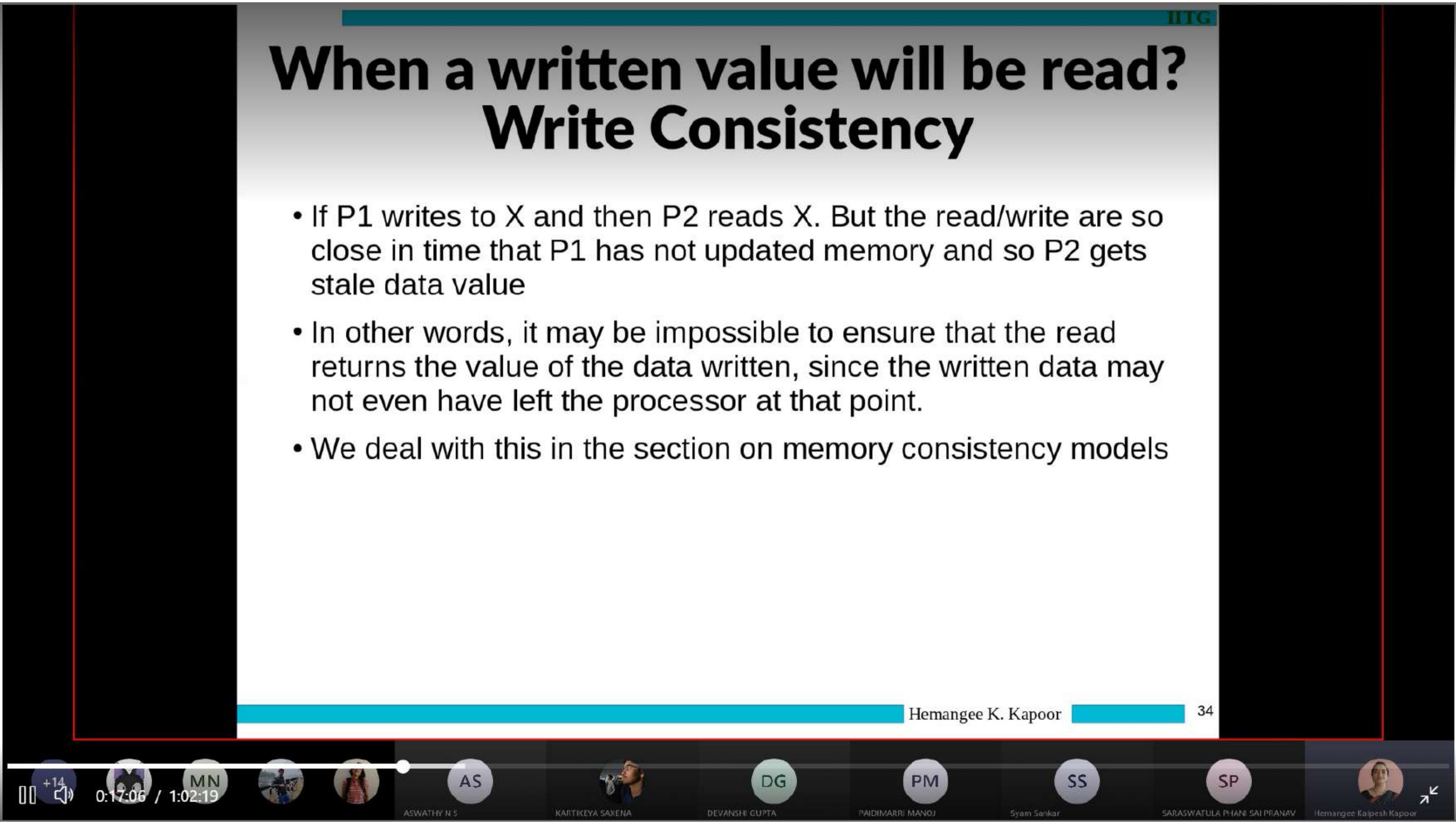
P3 sees X=2 then X=5

P4 sees X=5 then X=2

write serialisation

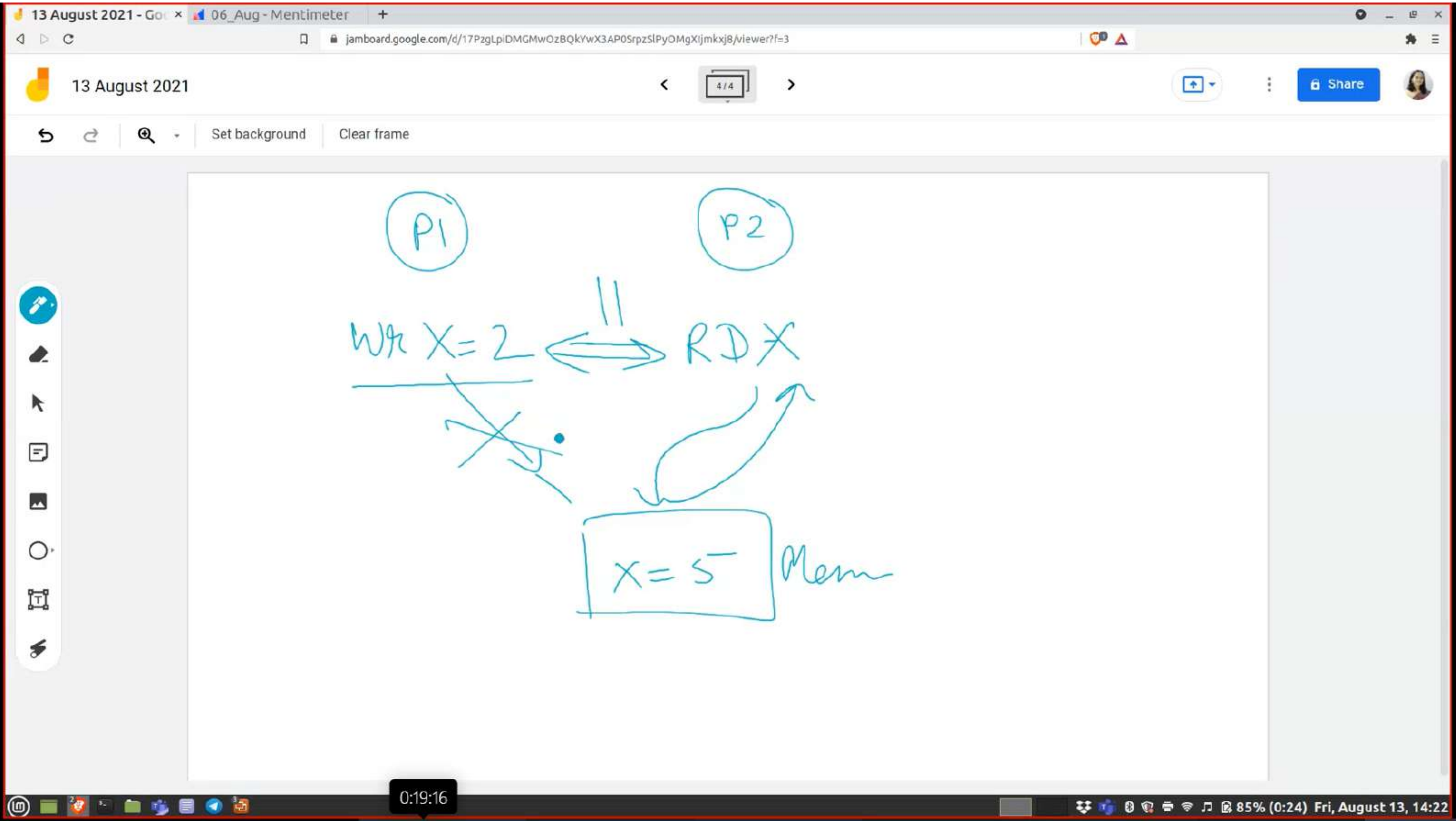
$$\text{Mem} \begin{array}{l} x=2 \\ x=5 \end{array} \downarrow \text{time}$$

$$\text{Mem} \begin{array}{l} x=5 \\ x=2 \end{array} \downarrow \text{time}$$



When a written value will be read? Write Consistency

- If P1 writes to X and then P2 reads X. But the read/write are so close in time that P1 has not updated memory and so P2 gets stale data value
- In other words, it may be impossible to ensure that the read returns the value of the data written, since the written data may not even have left the processor at that point.
- We deal with this in the section on memory consistency models


13 August 2021 - Go × 06_Aug - Mentimeter +

jamboard.google.com/d/17PzgLpiDMGMwOzBQkYwX3AP0SrpzSlPyOMgXIjmkxj8/viewer?f=3

13 August 2021

4/4

Share

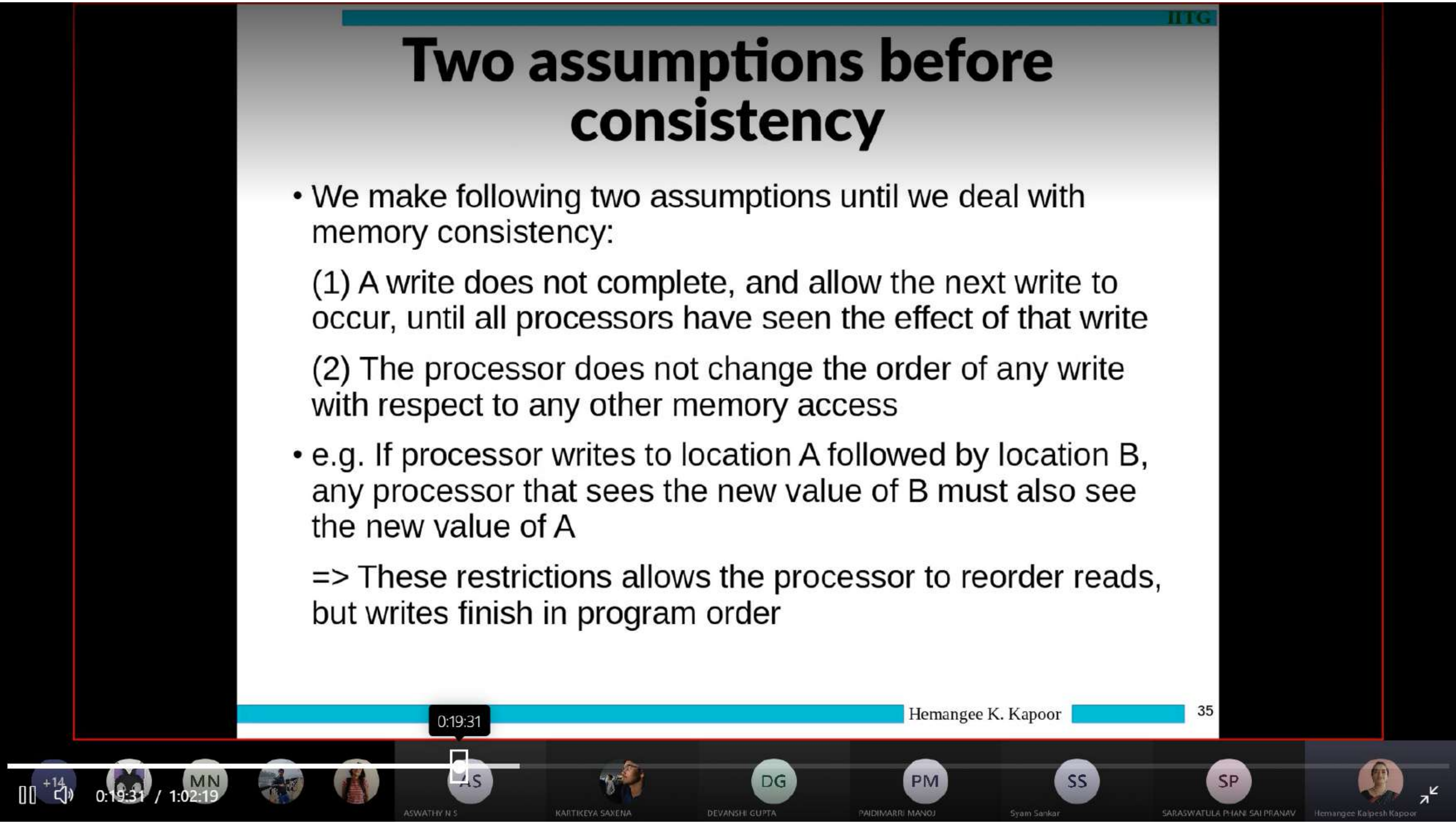
Set background Clear frame

$$\begin{tikzcd} P1 & & P2 \\ \text{Wr } X=2 \arrow[rr, leftrightarrow, "||"] \arrow[dr, phantom, "\times"] & & \text{RD } X \\ & \boxed{X=5} \text{ Mem} \arrow[ur, bend right] & \end{tikzcd}$$

85% (0:24) Fri, August 13, 14:22

0:19:16

0:19:16 / 1:02:19



IITG

Two assumptions before consistency

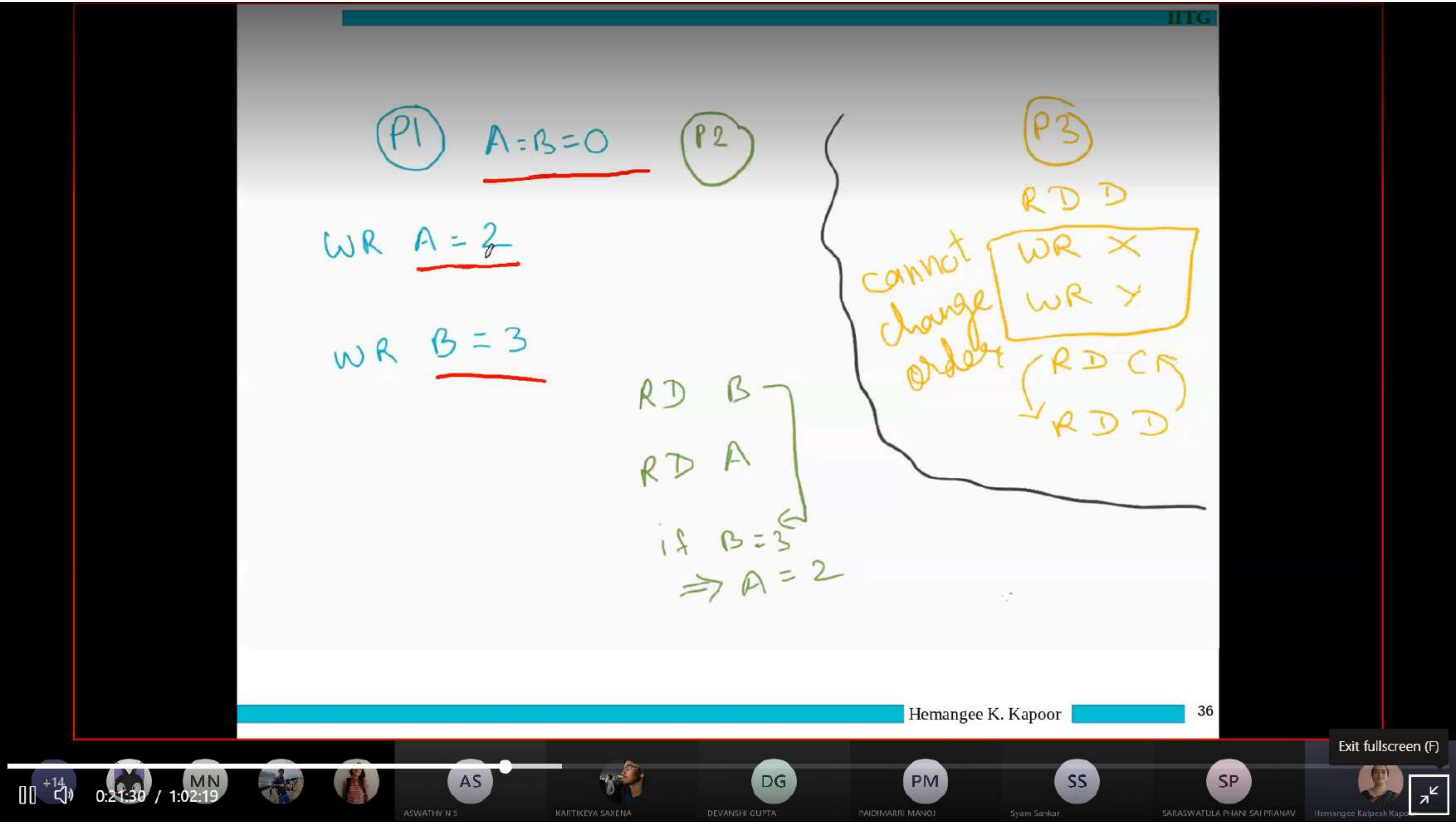
- We make following two assumptions until we deal with memory consistency:

  (1) A write does not complete, and allow the next write to occur, until all processors have seen the effect of that write

  (2) The processor does not change the order of any write with respect to any other memory access

- e.g. If processor writes to location A followed by location B, any processor that sees the new value of B must also see the new value of A

  => These restrictions allows the processor to reorder reads, but writes finish in program order



P1 A=B=0 P2 P3

WR A = 2

WR B = 3

RD B

RD A

if B=3 ⇒ A=2

RD D

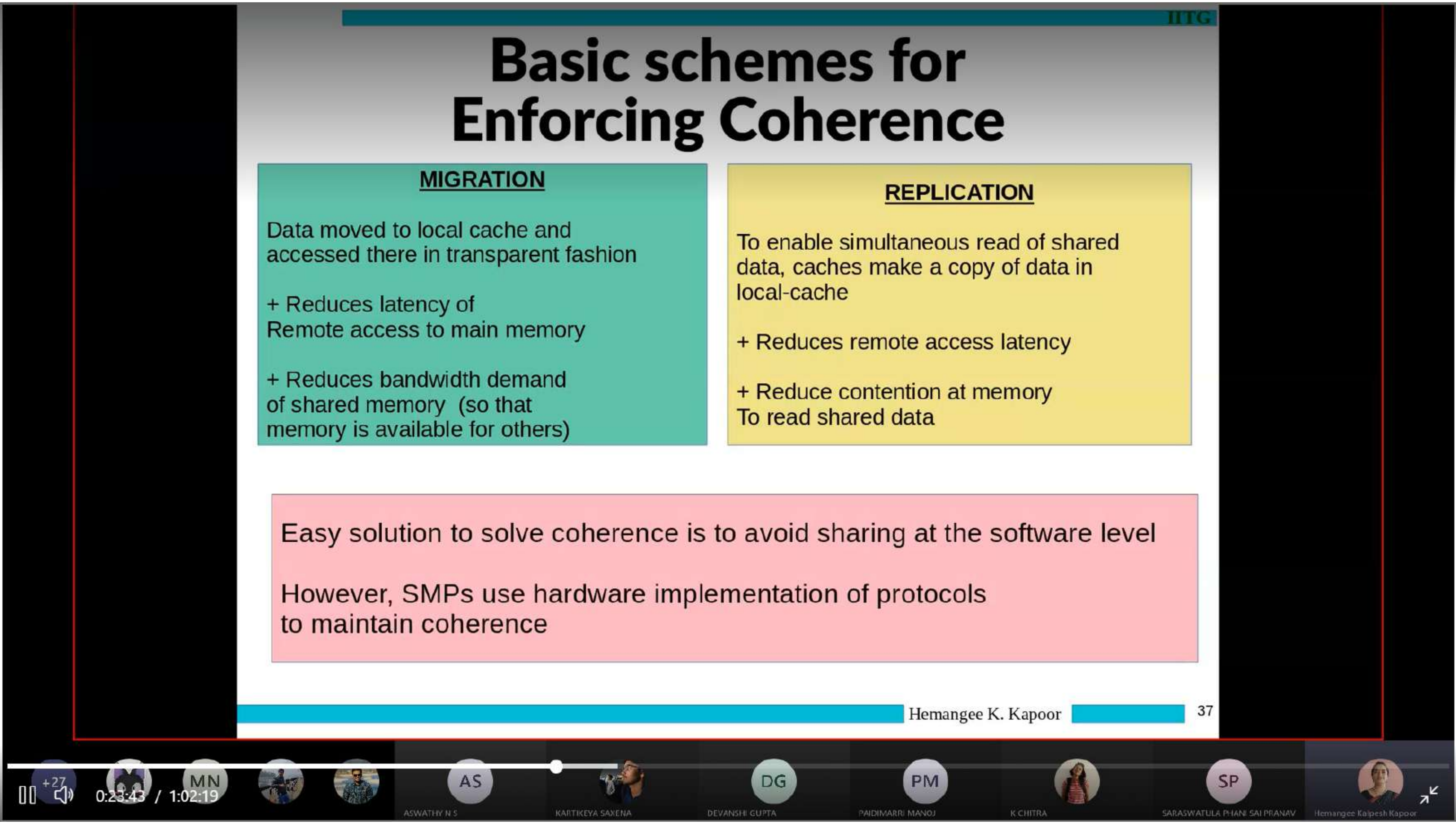
cannot change order

WR X

WR Y

RD C

RD D



Basic schemes for Enforcing Coherence

MIGRATION

Data moved to local cache and accessed there in transparent fashion

+ Reduces latency of Remote access to main memory

+ Reduces bandwidth demand of shared memory (so that memory is available for others)

REPLICATION

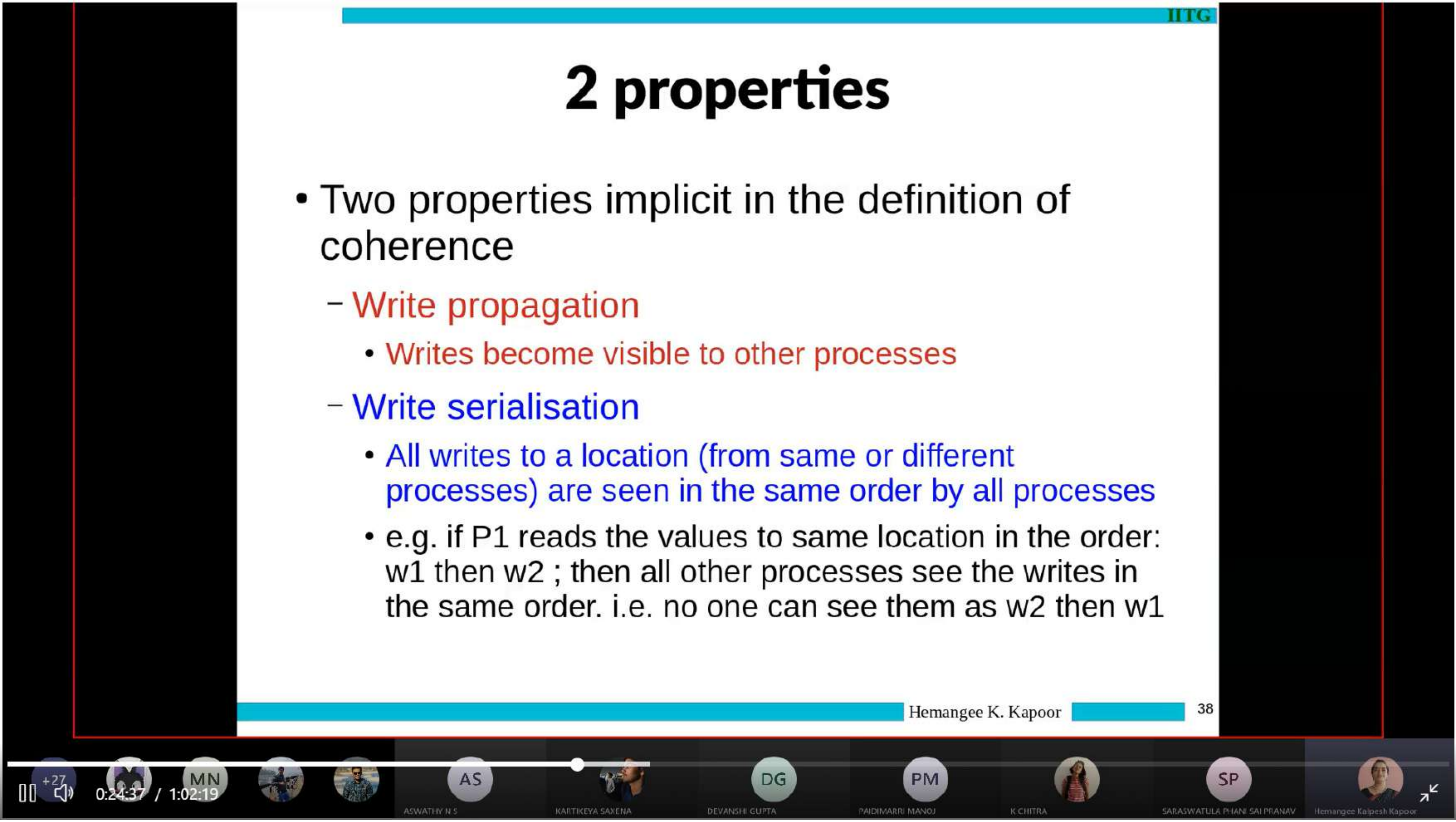
To enable simultaneous read of shared data, caches make a copy of data in local-cache

+ Reduces remote access latency

+ Reduce contention at memory To read shared data

Easy solution to solve coherence is to avoid sharing at the software level

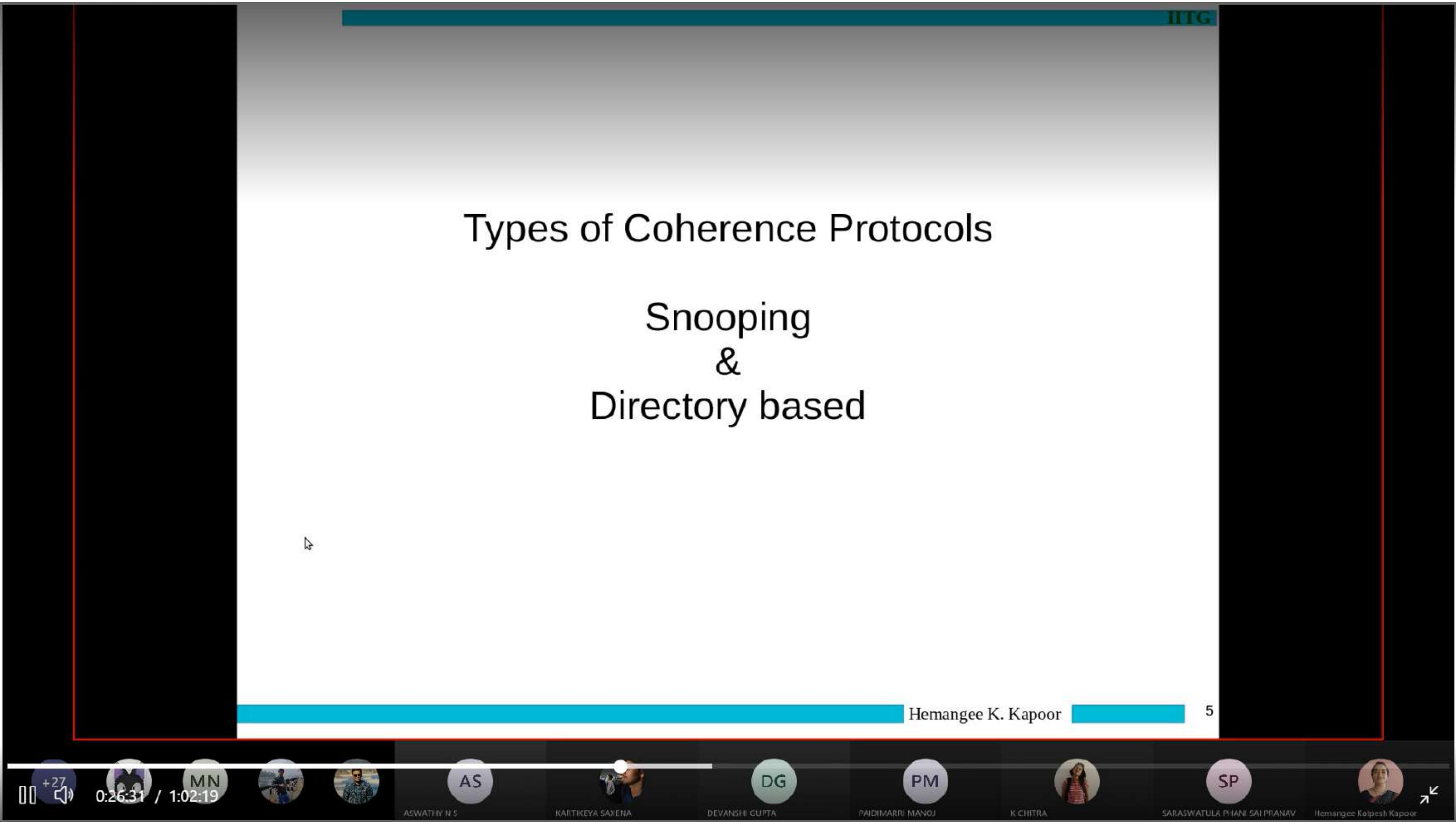
However, SMPs use hardware implementation of protocols to maintain coherence



IITG

2 properties

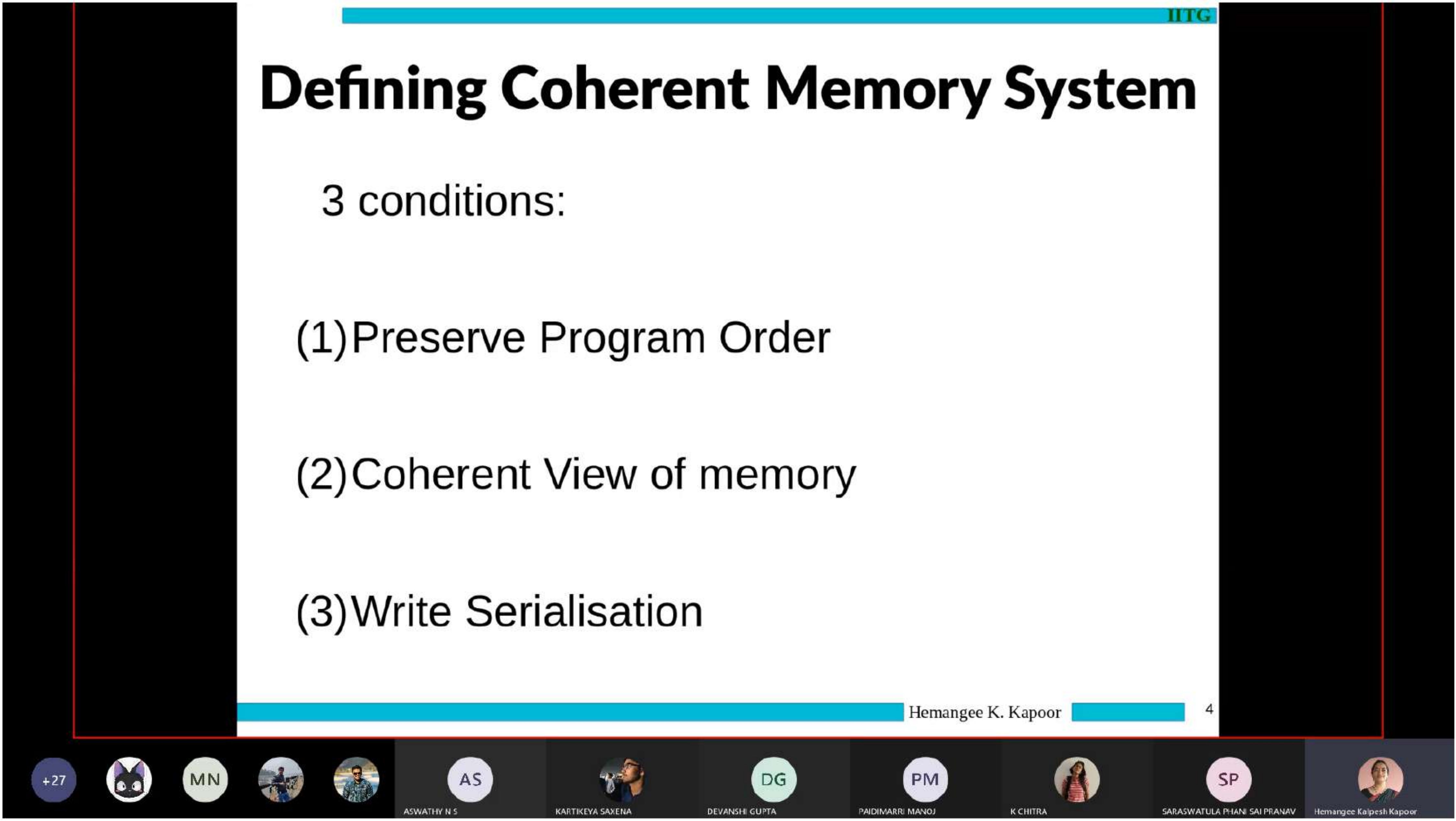
- Two properties implicit in the definition of coherence
  - Write propagation
    - Writes become visible to other processes
  - Write serialisation
    - All writes to a location (from same or different processes) are seen in the same order by all processes
    - e.g. if P1 reads the values to same location in the order: w1 then w2 ; then all other processes see the writes in the same order. i.e. no one can see them as w2 then w1



IITG

Types of Coherence Protocols

Snooping
&
Directory based



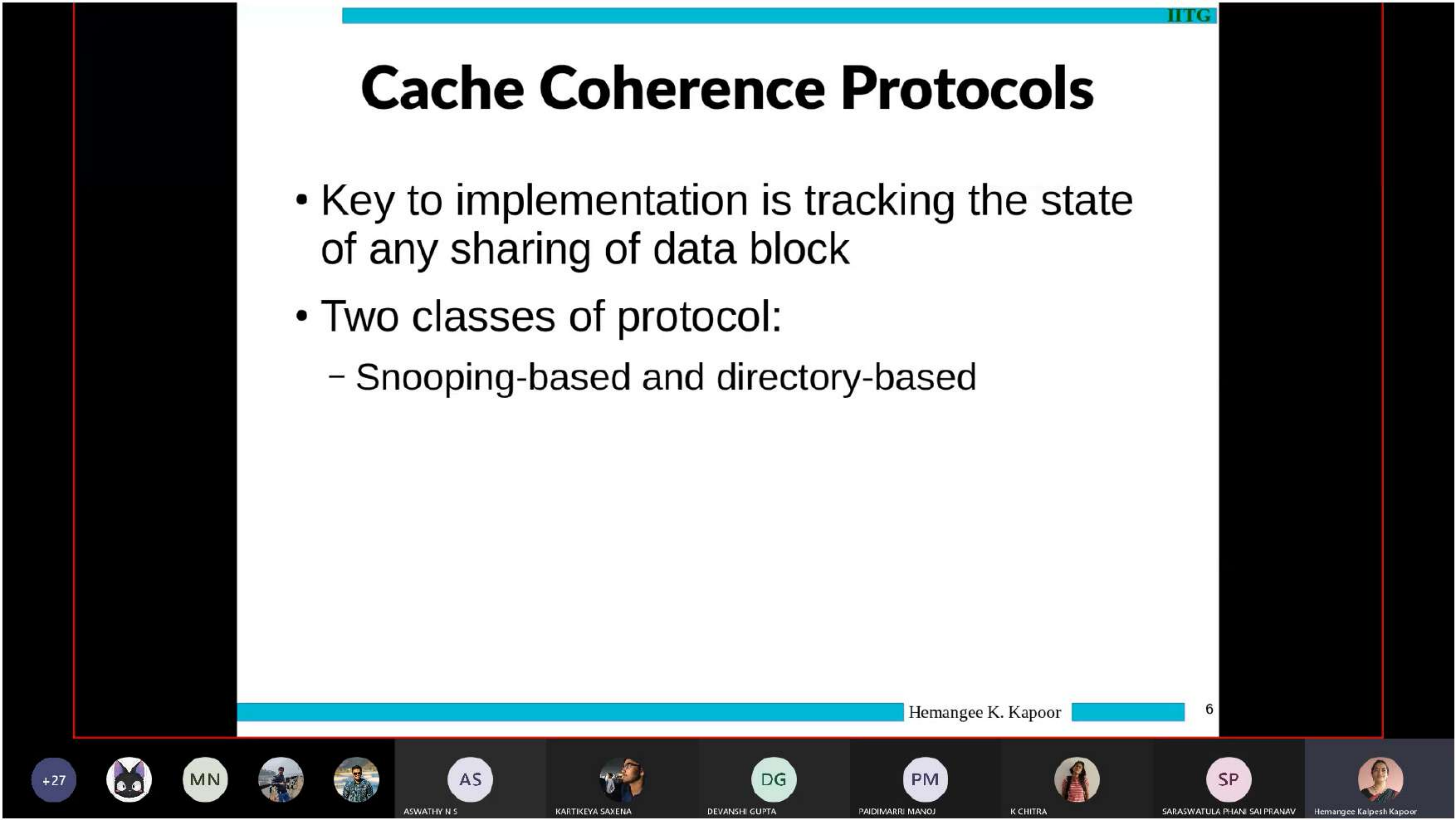
Defining Coherent Memory System

3 conditions:

(1)Preserve Program Order

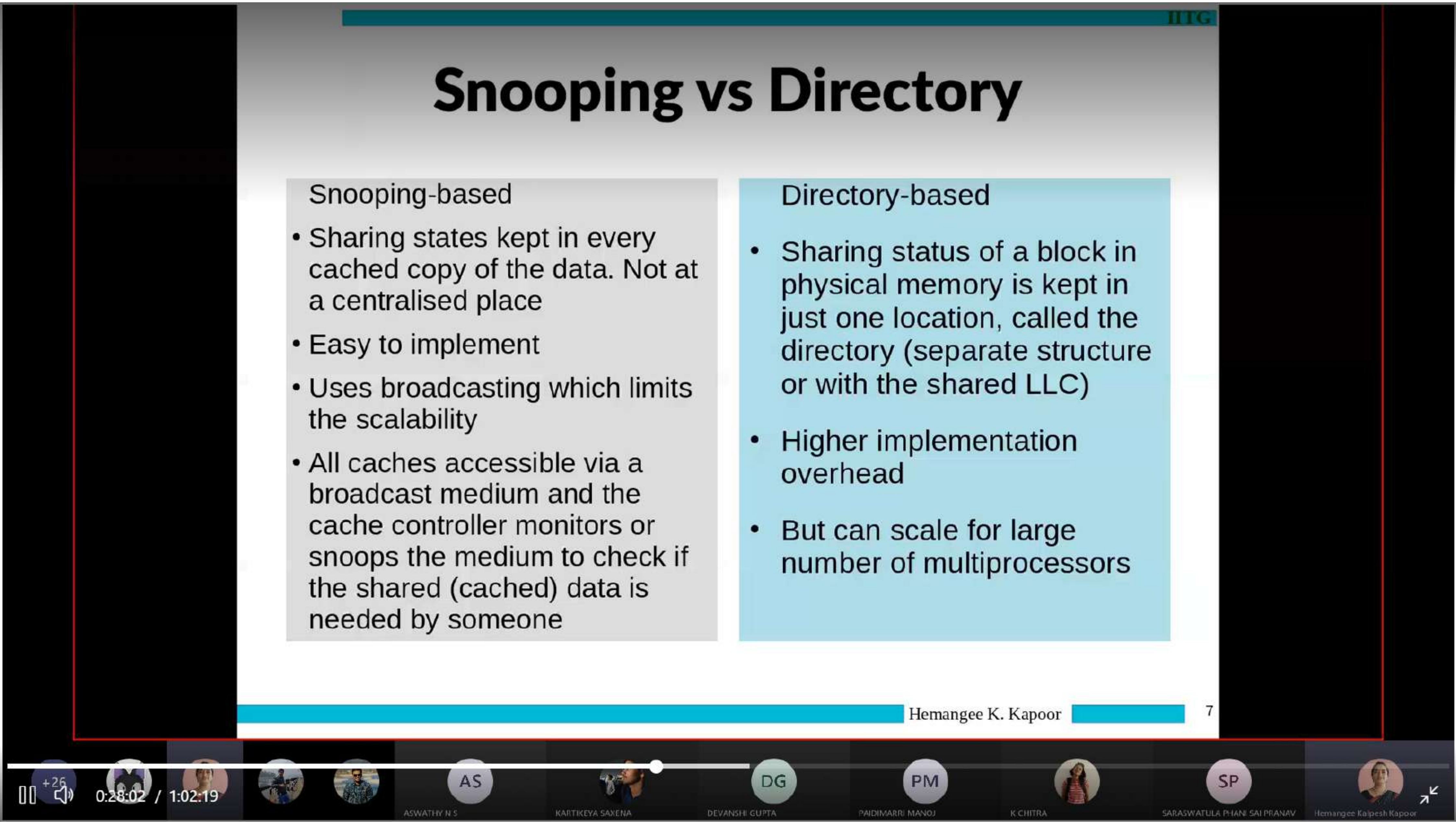
(2)Coherent View of memory

(3)Write Serialisation



Cache Coherence Protocols

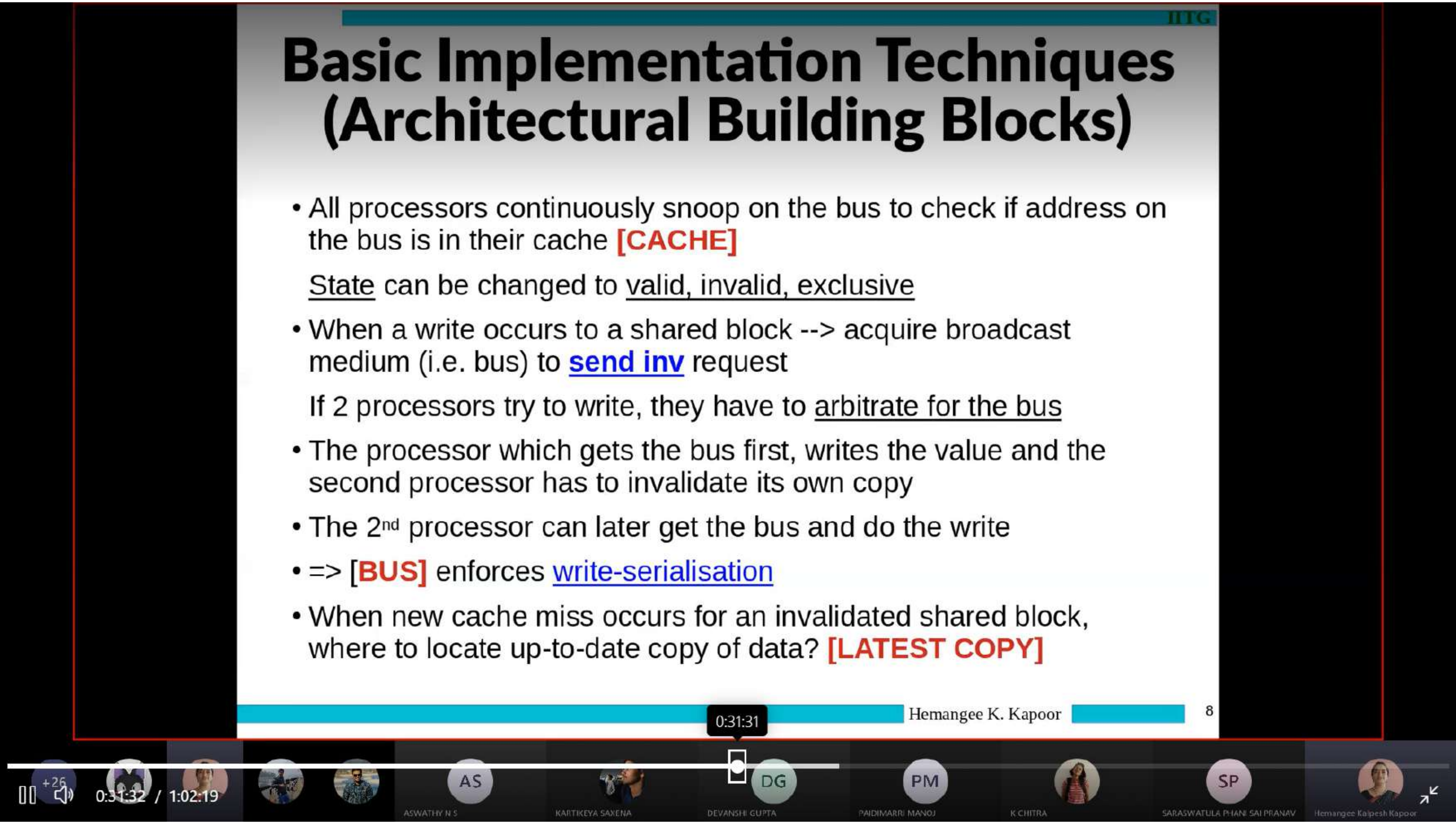
- Key to implementation is tracking the state of any sharing of data block
- Two classes of protocol:
  - Snooping-based and directory-based



IITG

Snooping vs Directory

Snooping-based	Directory-based
• Sharing states kept in every cached copy of the data. Not at a centralised place	• Sharing status of a block in physical memory is kept in just one location, called the directory (separate structure or with the shared LLC)
• Easy to implement	• Higher implementation overhead
• Uses broadcasting which limits the scalability	• But can scale for large number of multiprocessors
• All caches accessible via a broadcast medium and the cache controller monitors or snoops the medium to check if the shared (cached) data is needed by someone	



IITG

Basic Implementation Techniques (Architectural Building Blocks)

- All processors continuously snoop on the bus to check if address on the bus is in their cache **[CACHE]**

  State can be changed to valid, invalid, exclusive

- When a write occurs to a shared block --> acquire broadcast medium (i.e. bus) to **send inv** request

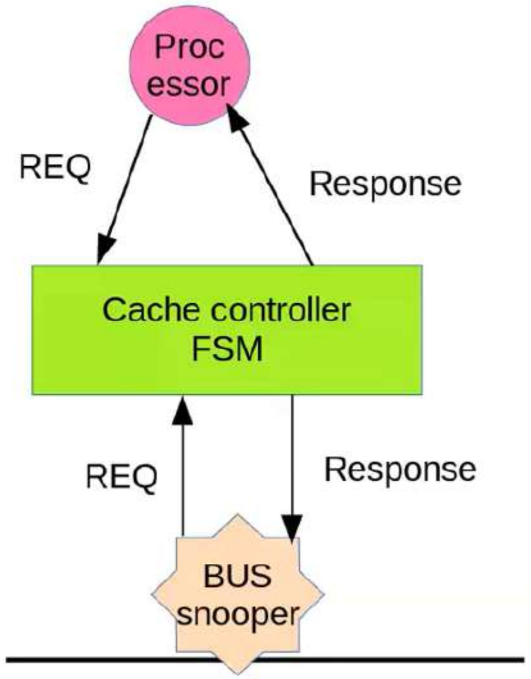
  If 2 processors try to write, they have to arbitrate for the bus

- The processor which gets the bus first, writes the value and the second processor has to invalidate its own copy
- The 2nd processor can later get the bus and do the write
- => **[BUS]** enforces write-serialisation
- When new cache miss occurs for an invalidated shared block, where to locate up-to-date copy of data? **[LATEST COPY]**

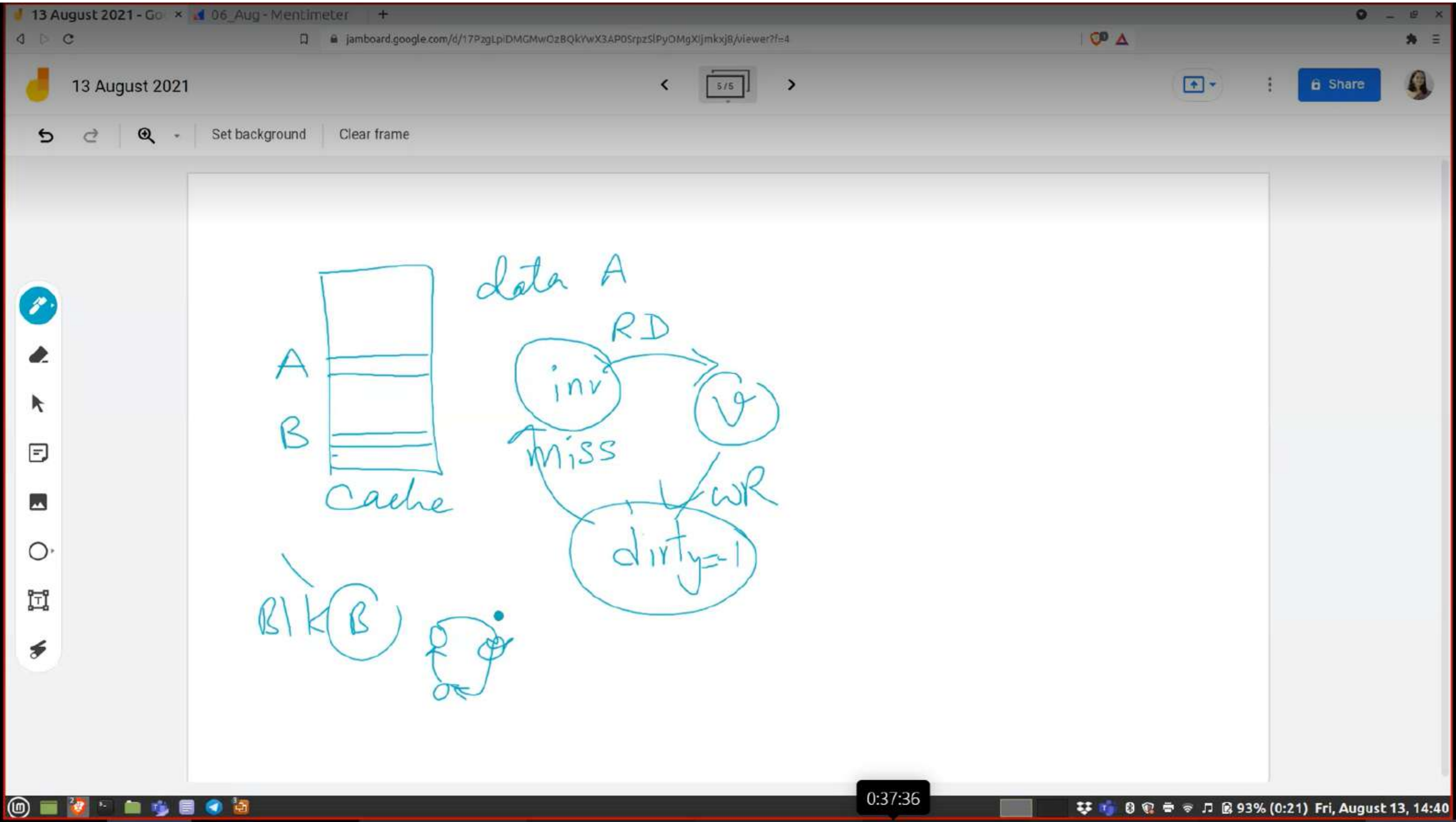
IITG

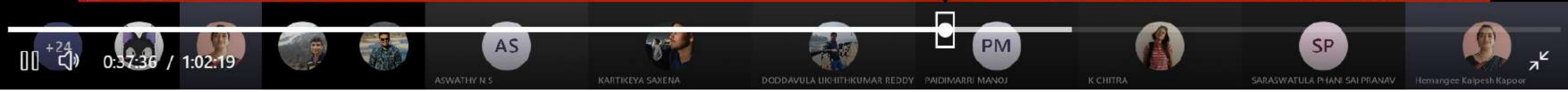
Protocol Implementation/Modelling

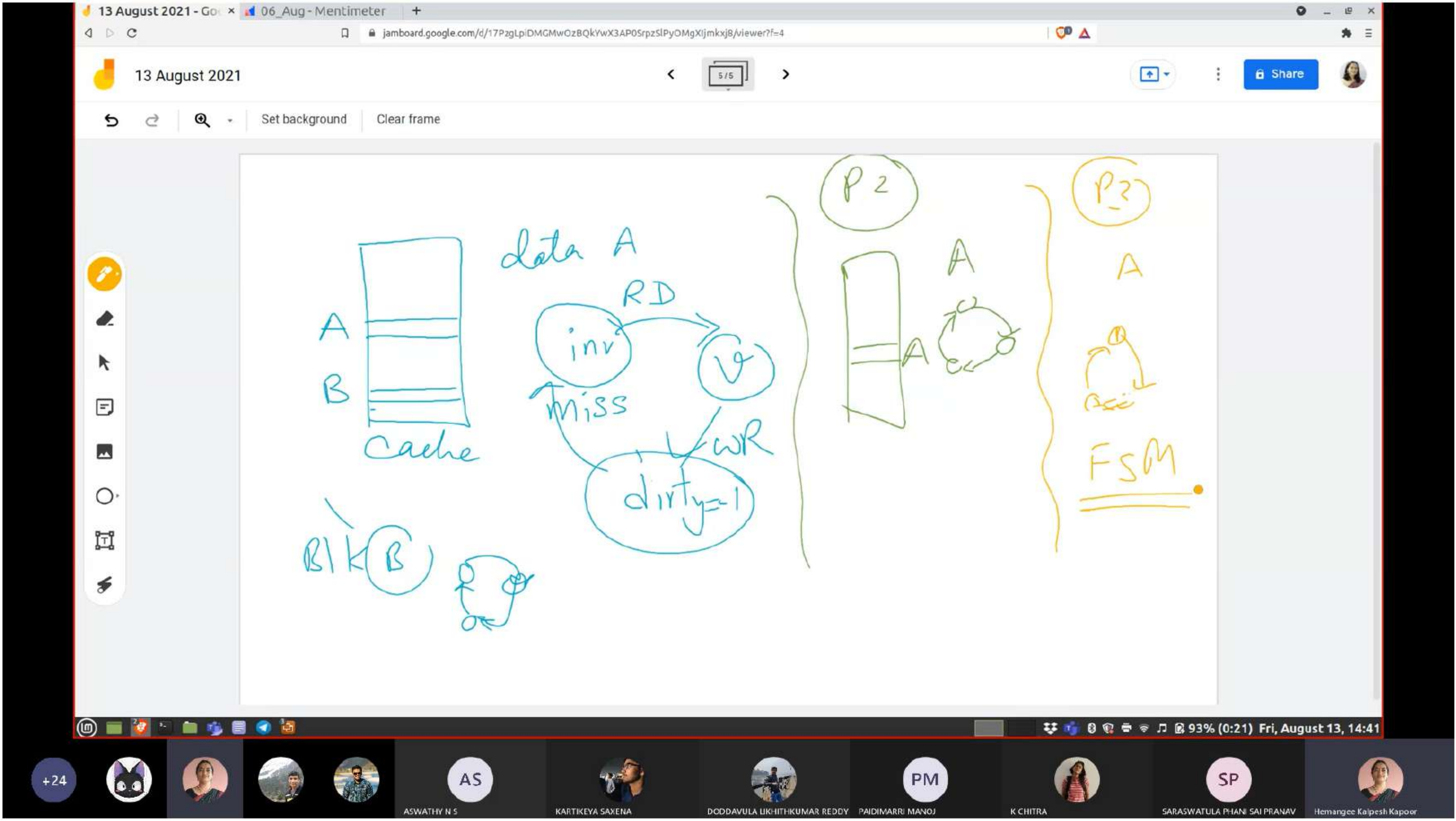
- In multiprocessor, each block has a state in each cache and the states change according to the state transition diagram
- If there are ‘p’ caches, then the block state is a vector of size=p
- Cache state is manipulated by a set of ‘p’ distributed finite state machines, implemented by the cache controllers
- Controller FSMs will receive inputs from processor or bus snooper, and they act accordingly

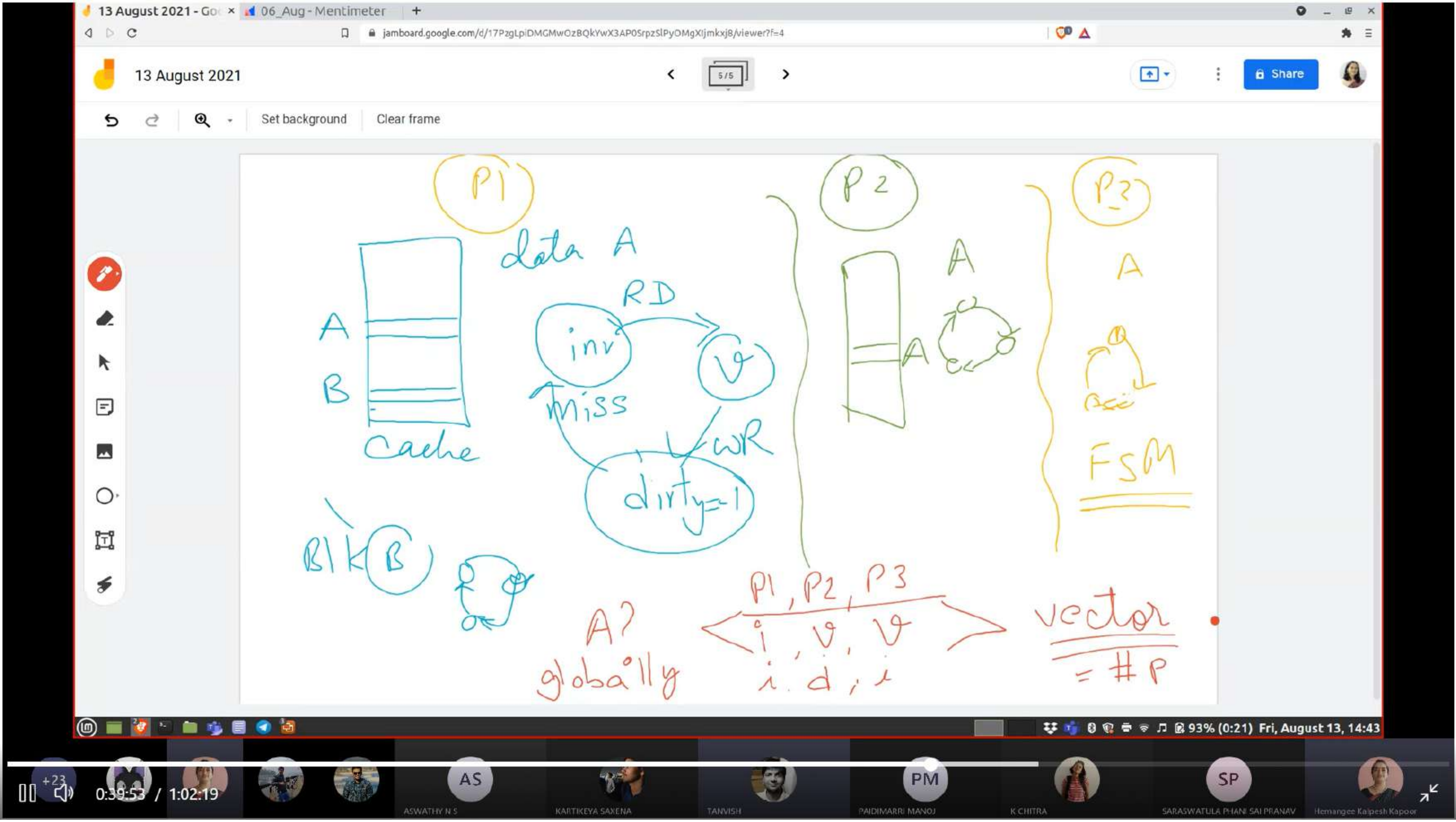


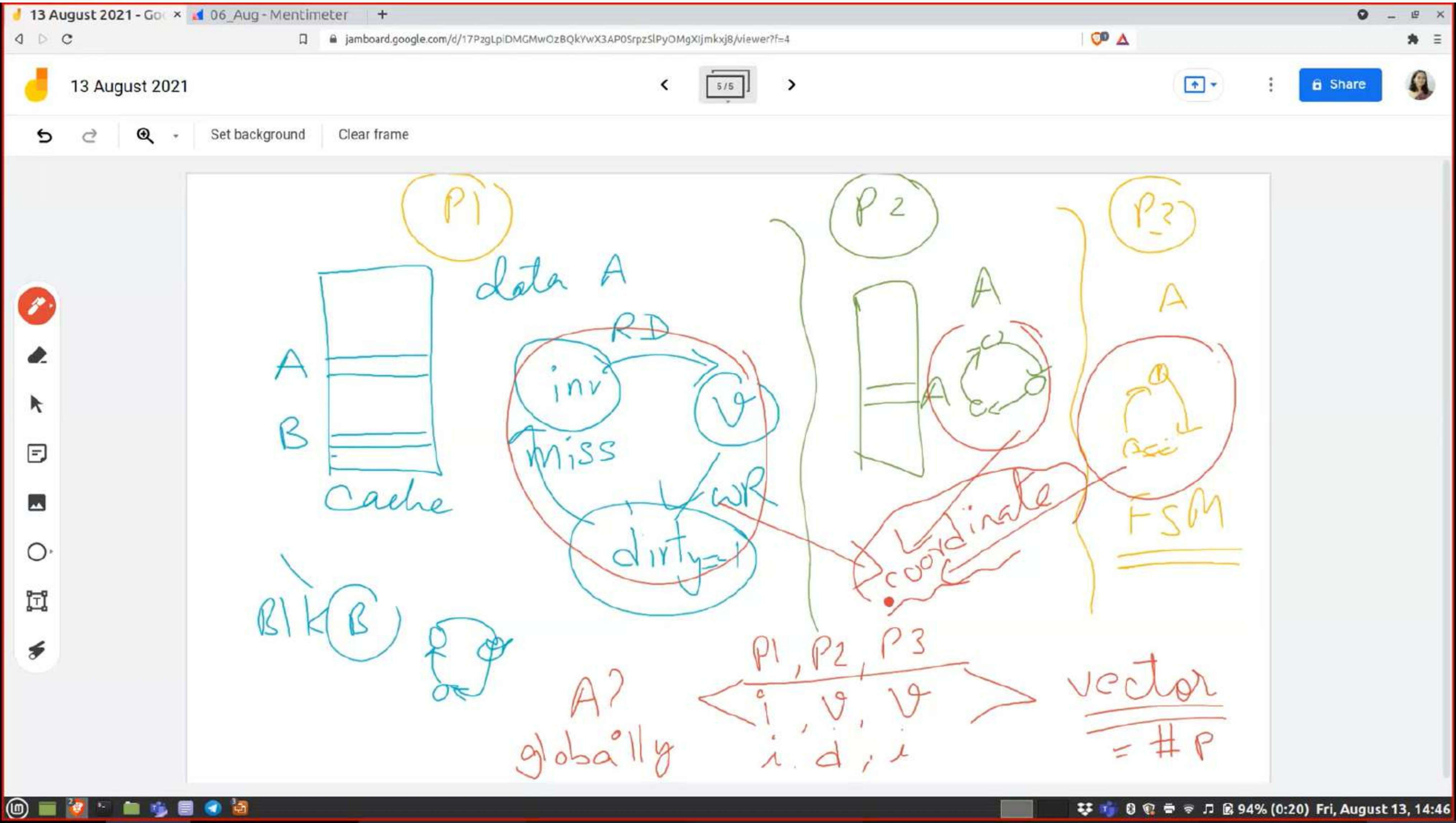
+26 AS ASWATHY N S KARTIKEYA SAXENA DG DEVANSHI GUPTA PM PAIDIMARRI MANOJ K CHITRA SP SARASWATULA PHANI SAI PRANAV Hemangee Kalpesh Kapoor


13 August 2021
data A
A
B
Cache
RD
inv
V
miss
WR
dirty=1
Blk B
Set background
Clear frame
Share
0:37:36

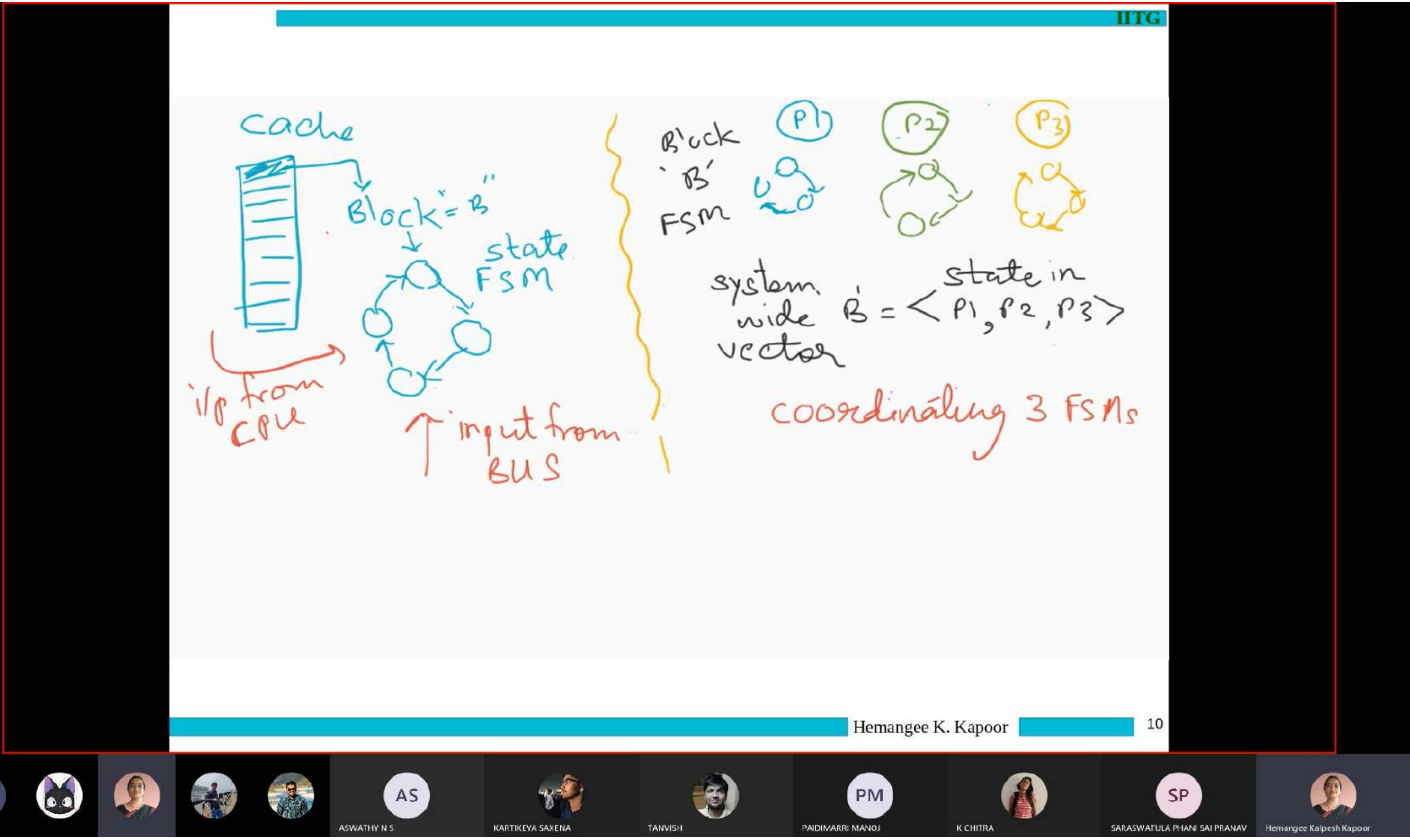

0:37:36 / 1:02:19
AS
PM
SP
ASWATHY N S
KARTIKEYA SAXENA
DODDAVULA LIKHITHKUMAR REDDY
PAIDIMARRI MANOJ
K CHITRA
SARASWATULA PHANI SAI PRANAV
Hemangee Kalpesh Kapoor

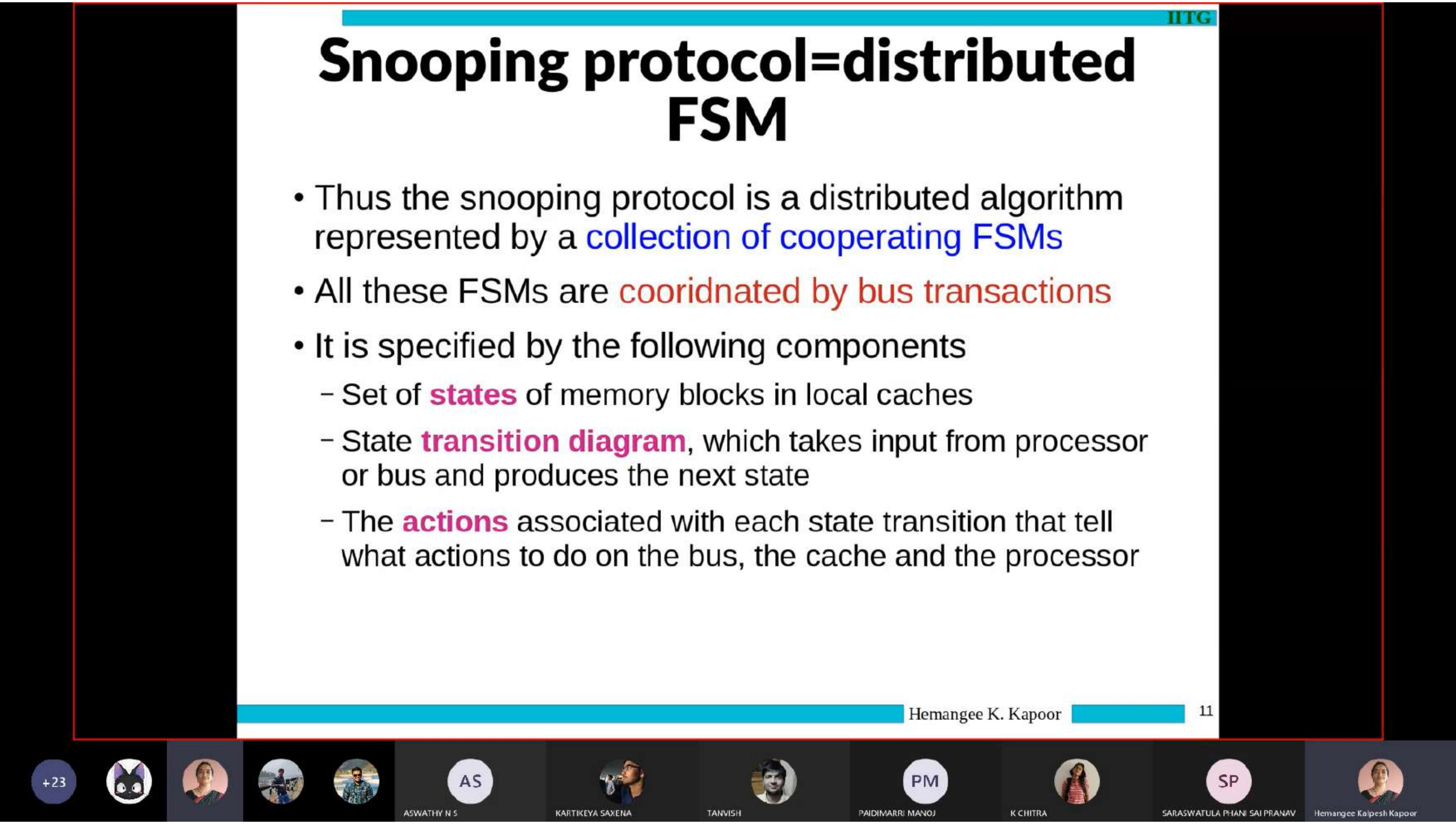

13 August 2021
jamboard.google.com/d/17PzgLpiDMGMwOzBQkYwX3AP0SrpzSlPyOMgXIjmkxj8/viewer?f=4
5 / 5
Share
Set background
Clear frame
data A
A
B
Cache
RD
inv
V
miss
WR
dirty=1
Blk B
P2
A
A
P3
A
FSM
93% (0:21) Fri, August 13, 14:41
+24
AS
ASWATHY N S
KARTIKEYA SAXENA
DODDAVULA LIKHITHKUMAR REDDY
PM
PAIDIMARRI MANOJ
K CHITRA
SP
SARASWATULA PHANI SAI PRANAV
Hemangee Kalpesh Kapoor


13 August 2021 - Go
06_Aug - Mentimeter
jamboard.google.com/d/17PzgLpiDMGMwOzBQkYwX3AP0SrpzSlPyOMgXIjmkxj8/viewer?f=4
13 August 2021
5 / 5
Share
Set background
Clear frame
P1
data A
A
B
Cache
RD
inv
V
miss
WR
dirty=1
Blk B
P2
A
A
P3
A
FSM
A?
globally
P1, P2, P3
< i, V, V >
i, d, i
vector
= #P
93% (0:21) Fri, August 13, 14:43
0:39:53 / 1:02:19
+23
AS
ASWATHY N S
KARTIKEYA SAXENA
TANVISH
PM
PAIDIMARRI MANOJ
K CHITRA
SP
SARASWATULA PHANI SAI PRANAV
Hemangee Kalpesh Kapoor


13 August 2021 - Go
06_Aug - Mentimeter
jamboard.google.com/d/17PzgLpiDMGMwOzBQkYwX3AP0SrpzSlPyOMgXIjmkxj8/viewer?f=4
13 August 2021
5 / 5
Share
Set background
Clear frame
P1
P2
P3
data A
A
B
Cache
RD
inv
V
miss
WR
dirty=1
Blk B
A
A
A
coordinate
FSM
A?
globally
P1, P2, P3
< i, V, V >
i, d, i
vector
= # P
94% (0:20) Fri, August 13, 14:46

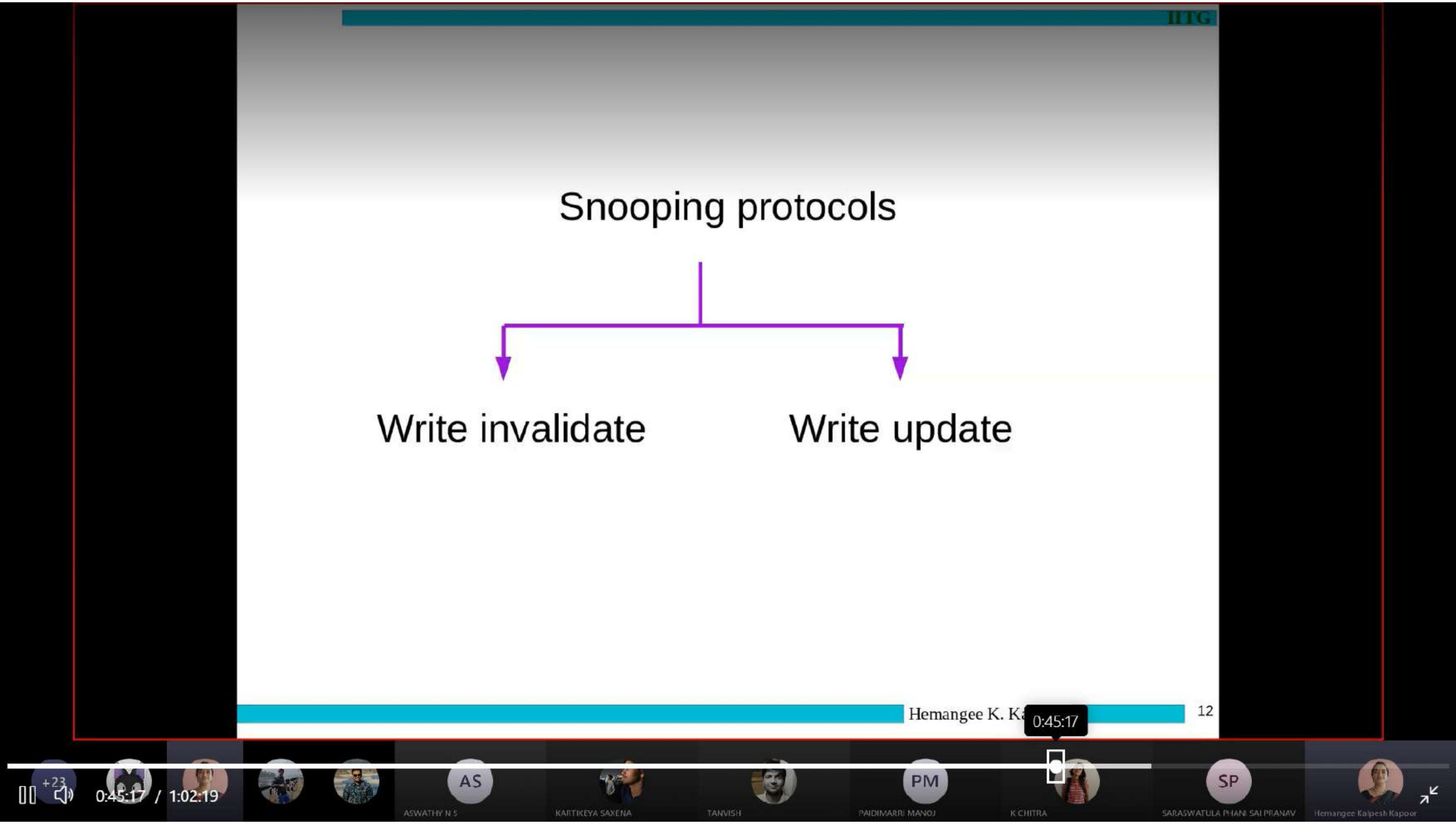
+23
AS
ASWATHY N S
KARTIKEYA SAXENA
TANVISH
PM
PAIDIMARRI MANOJ
K CHITRA
SP
SARASWATULA PHANI SAI PRANAV
Hemangee Kalpesh Kapoor


IITG
cache
Block "= B"
state FSM
i/p from CPU
input from BUS
Block 'B' FSM
P1
P2
P3
system wide state vector
state in B' = < P1, P2, P3 >
coordinating 3 FSMs
Hemangee K. Kapoor
10
+23
ASWATHY N S
KARTIKEYA SAXENA
TANVISH
PAIDIMARRI MANOJ
K CHITRA
SARASWATULA PHANI SAI PRANAV
Hemangee Kalpesh Kapoor



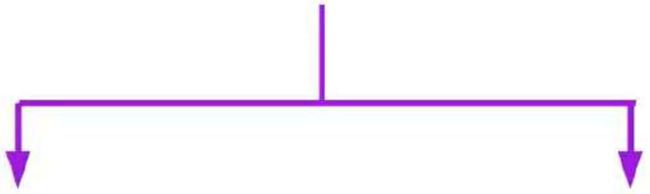
Snooping protocol=distributed FSM

- Thus the snooping protocol is a distributed algorithm represented by a collection of cooperating FSMs
- All these FSMs are cooridnated by bus transactions
- It is specified by the following components
  - Set of **states** of memory blocks in local caches
  - State **transition diagram**, which takes input from processor or bus and produces the next state
  - The **actions** associated with each state transition that tell what actions to do on the bus, the cache and the processor



IITG

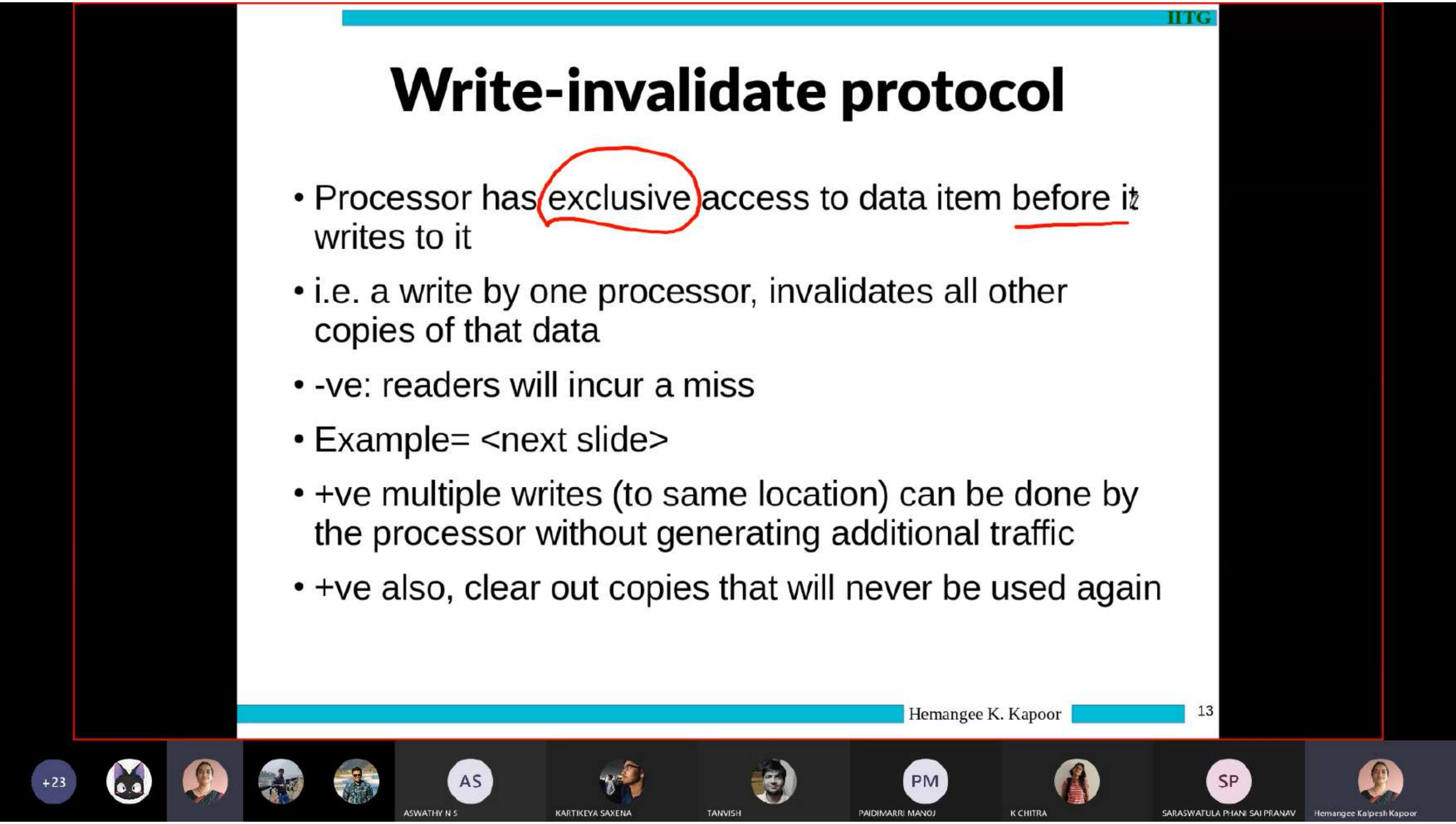
Snooping protocols



Write invalidate

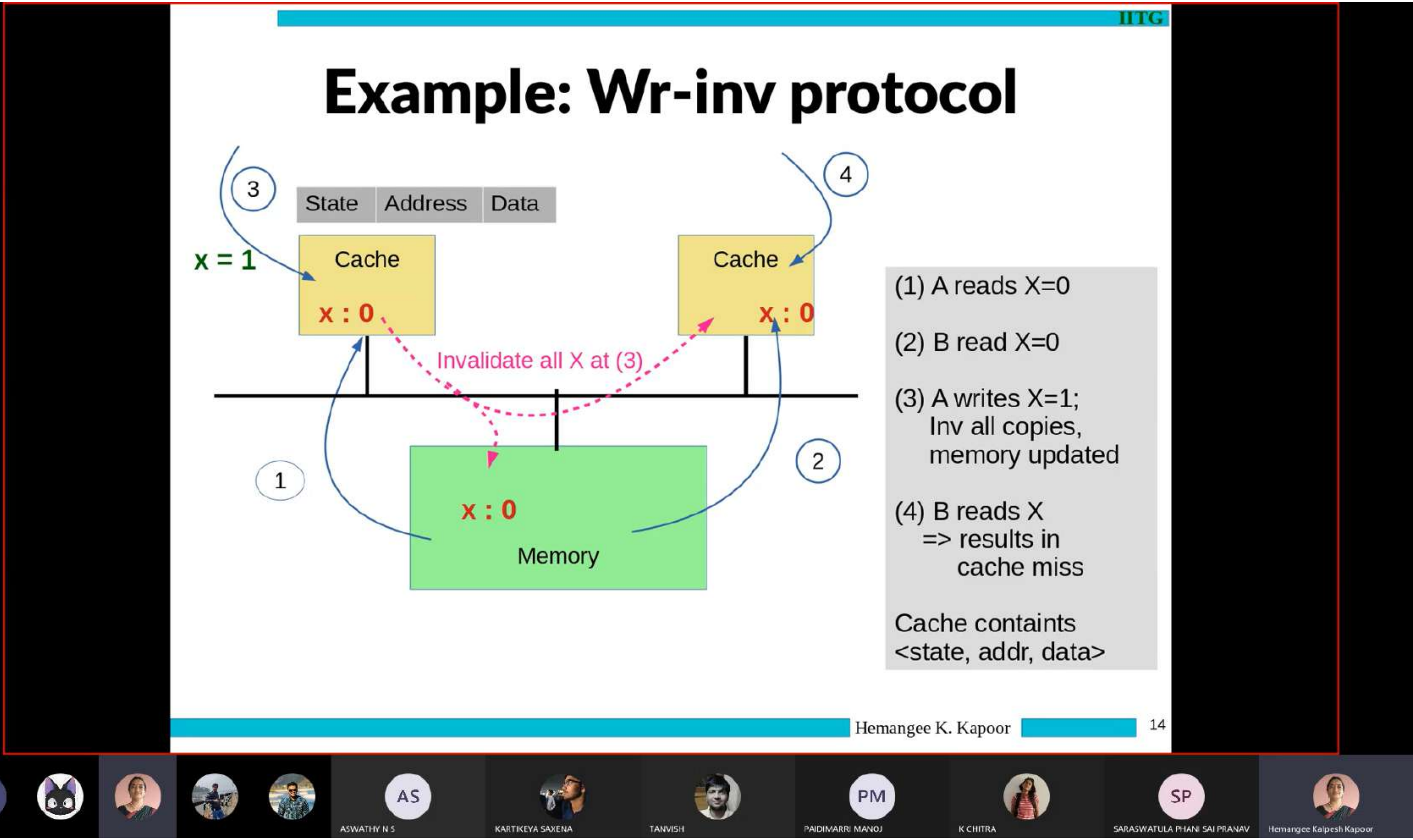
Write update

Hemangee K. Ka

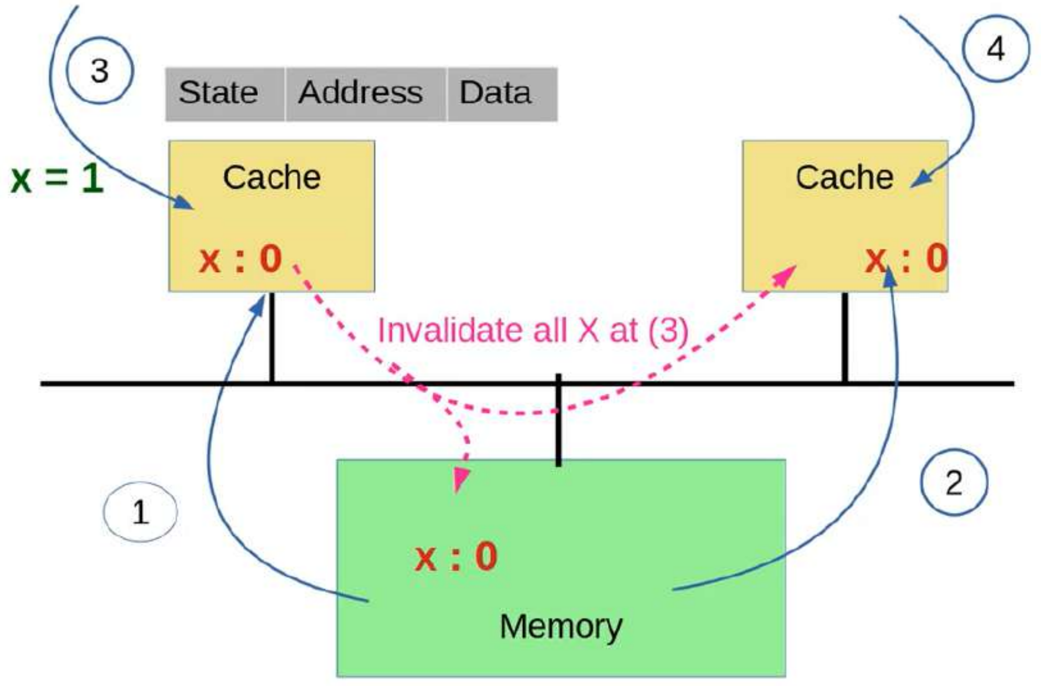


Write-invalidate protocol

- Processor has exclusive access to data item before it writes to it
- i.e. a write by one processor, invalidates all other copies of that data
- -ve: readers will incur a miss
- Example= <next slide>
- +ve multiple writes (to same location) can be done by the processor without generating additional traffic
- +ve also, clear out copies that will never be used again



Example: Wr-inv protocol



(1) A reads X=0

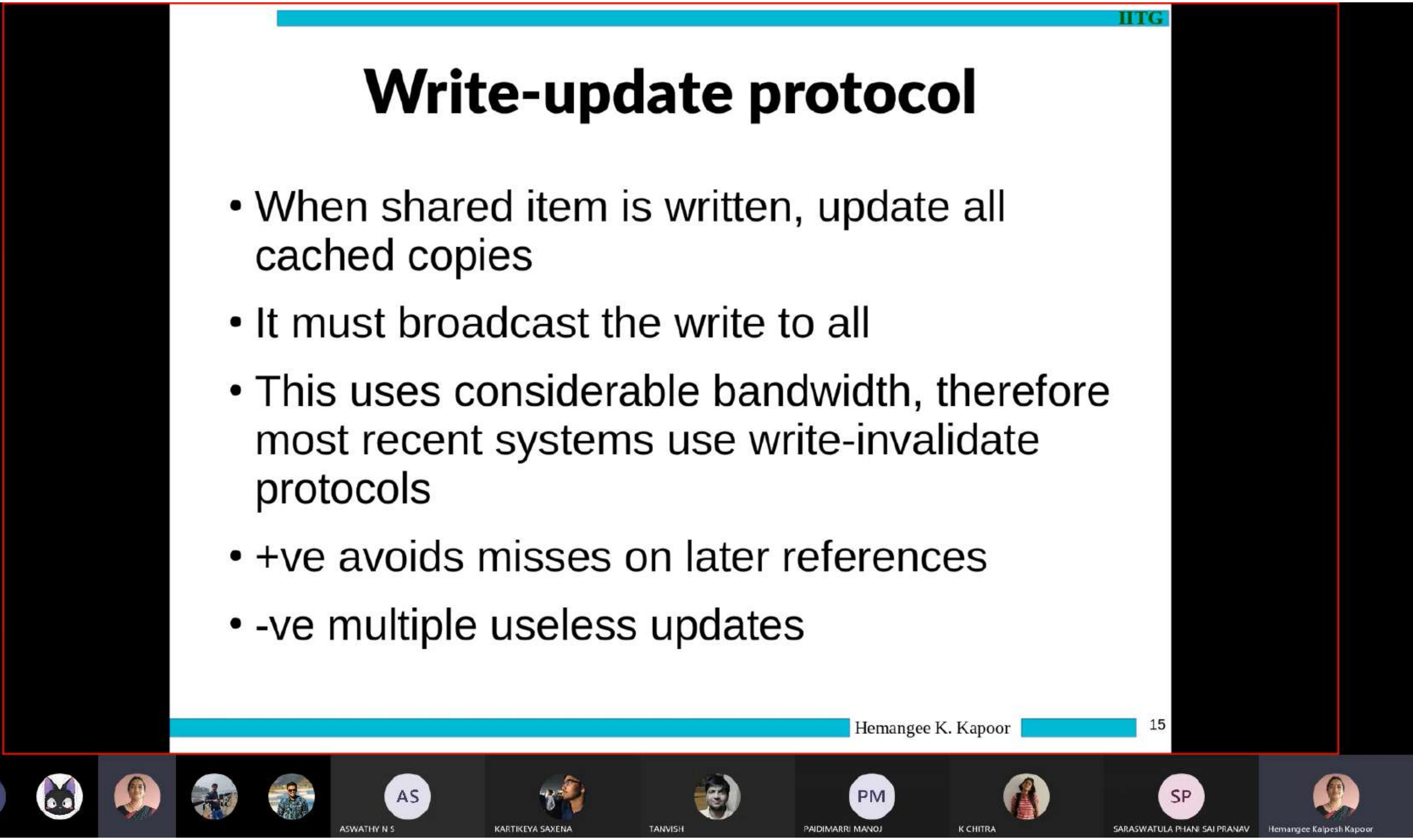
(2) B read X=0

(3) A writes X=1;
Inv all copies,
memory updated

(4) B reads X
=> results in
cache miss

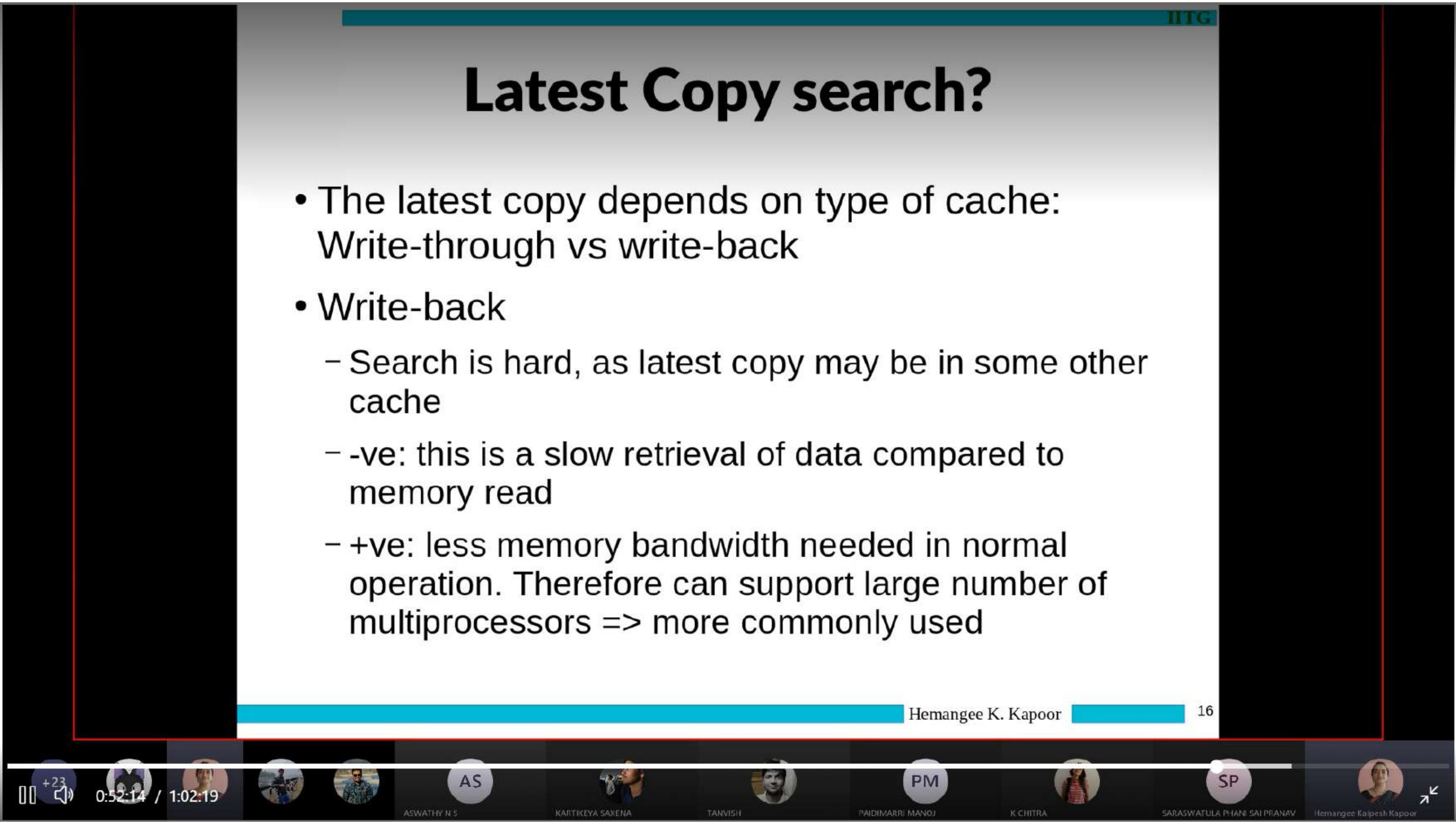
Cache containts
<state, addr, data>

+23
ASWATHY N S
KARTIKEYA SAXENA
TANVISH
PAIDIMARRI MANOJ
K CHITRA
SARASWATULA PHANI SAI PRANAV
Hemangee Kalpesh Kapoor



Write-update protocol

- When shared item is written, update all cached copies
- It must broadcast the write to all
- This uses considerable bandwidth, therefore most recent systems use write-invalidate protocols
- +ve avoids misses on later references
- -ve multiple useless updates

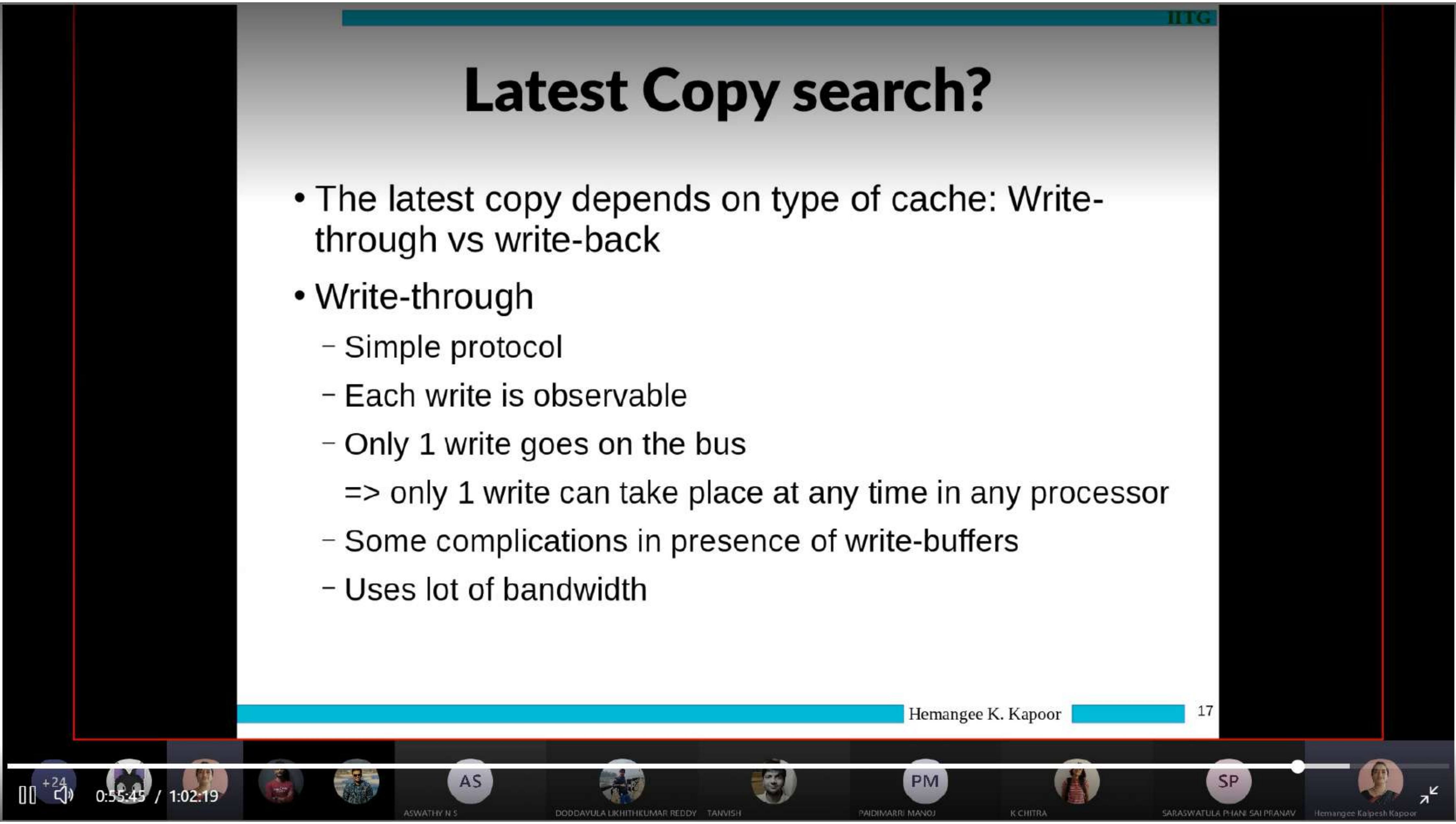


IITG

Latest Copy search?

- The latest copy depends on type of cache: Write-through vs write-back
- Write-back
  - Search is hard, as latest copy may be in some other cache
  - -ve: this is a slow retrieval of data compared to memory read
  - +ve: less memory bandwidth needed in normal operation. Therefore can support large number of multiprocessors => more commonly used

Hemangee K. Kapoor 16



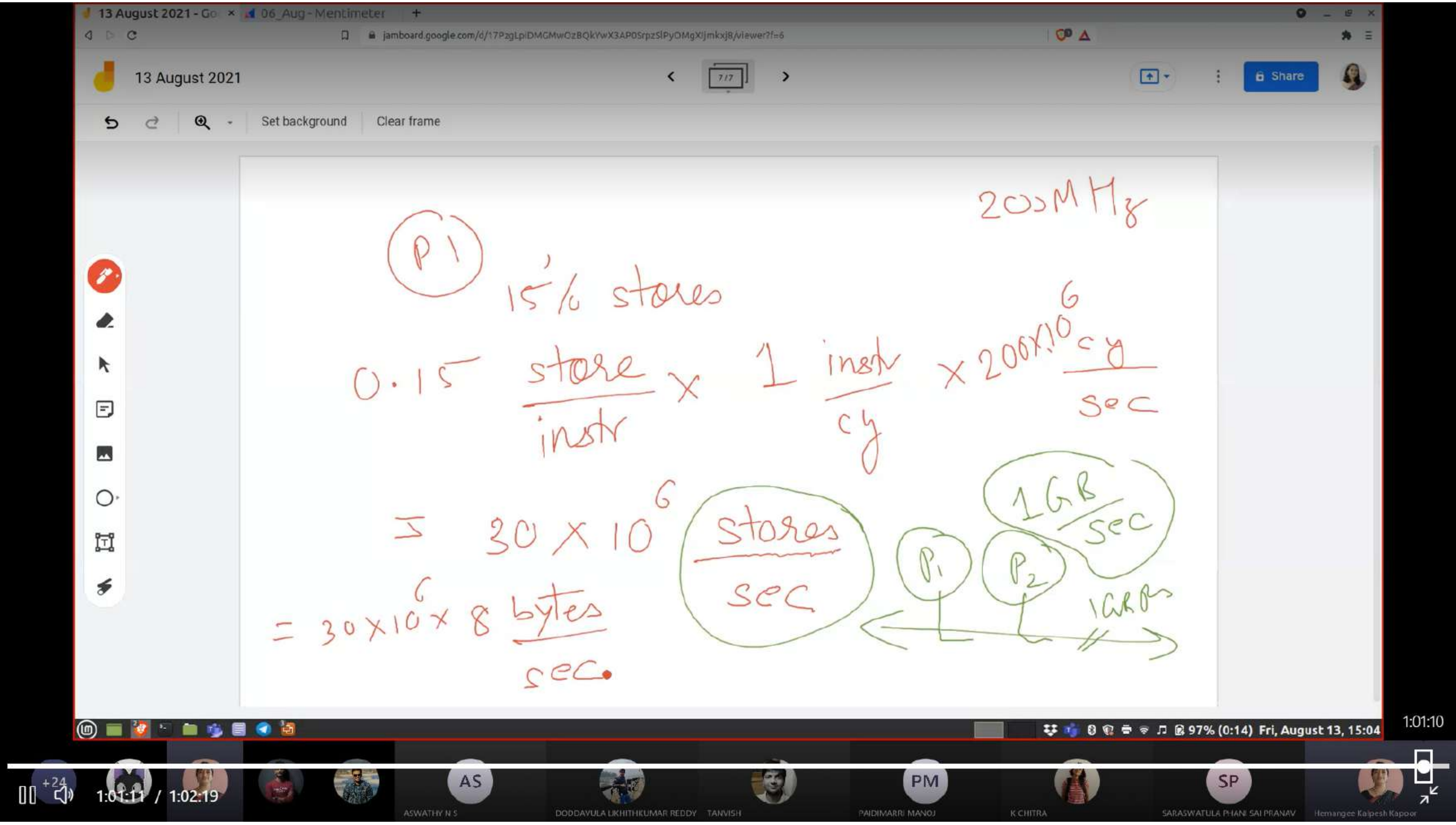
IITG

Latest Copy search?

- The latest copy depends on type of cache: Write-through vs write-back
- Write-through
  - Simple protocol
  - Each write is observable
  - Only 1 write goes on the bus

    => only 1 write can take place at any time in any processor
  - Some complications in presence of write-buffers
  - Uses lot of bandwidth

Hemangee K. Kapoor 17



200MHz

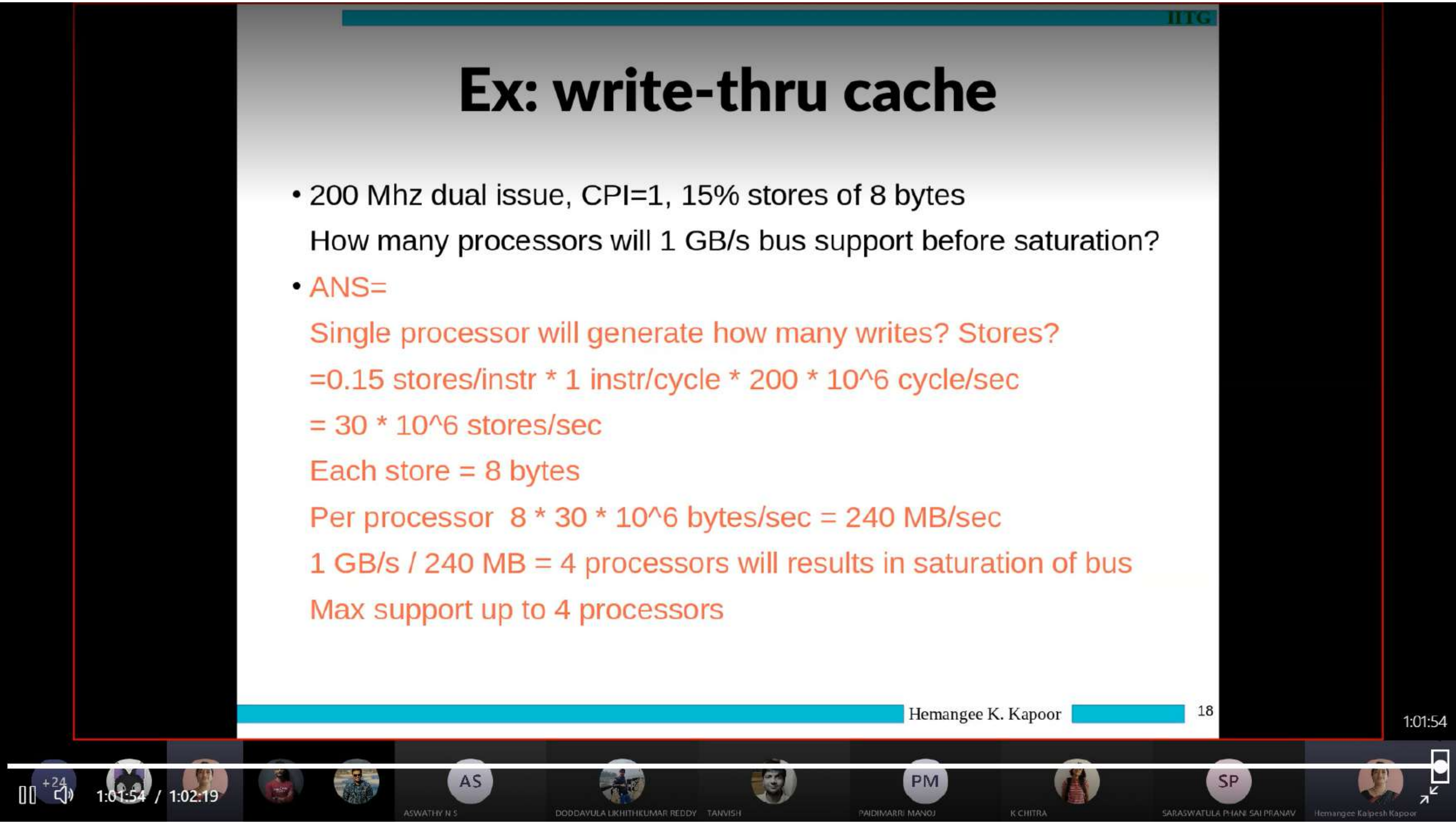
P1

15% stores

$$0.15 \frac{\text{store}}{\text{instr}} \times 1 \frac{\text{instr}}{\text{cy}} \times 200\times10^{6} \frac{\text{cy}}{\text{sec}}$$

$$= 30 \times 10^{6} \boxed{\frac{\text{stores}}{\text{sec}}}$$

$$= 30\times10^{6} \times 8 \frac{\text{bytes}}{\text{sec}}$$



Ex: write-thru cache

- 200 Mhz dual issue, CPI=1, 15% stores of 8 bytes
  How many processors will 1 GB/s bus support before saturation?
- ANS=
  Single processor will generate how many writes? Stores?
  =0.15 stores/instr * 1 instr/cycle * 200 * 10^6 cycle/sec
  = 30 * 10^6 stores/sec
  Each store = 8 bytes
  Per processor 8 * 30 * 10^6 bytes/sec = 240 MB/sec
  1 GB/s / 240 MB = 4 processors will results in saturation of bus
  Max support up to 4 processors