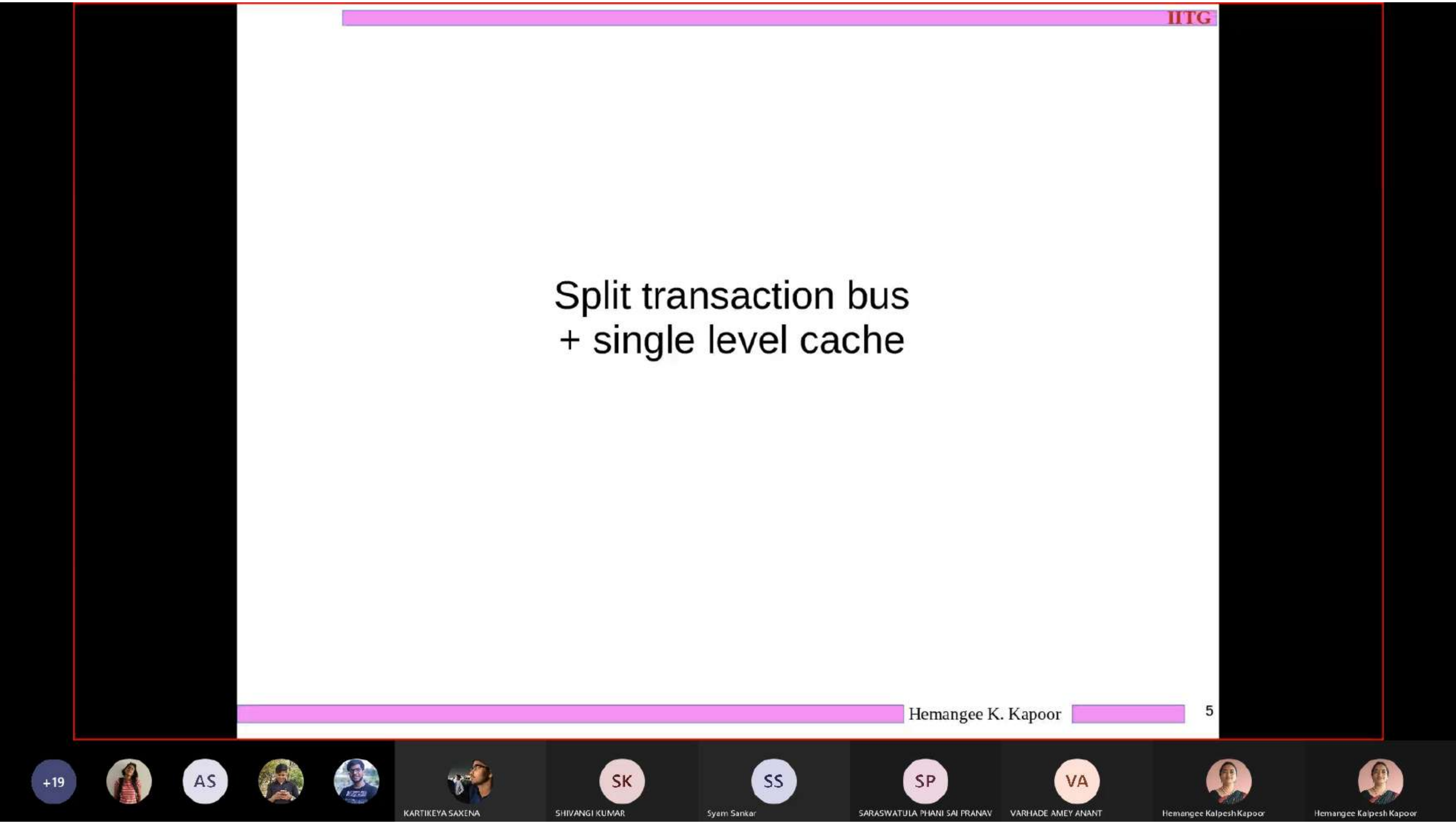
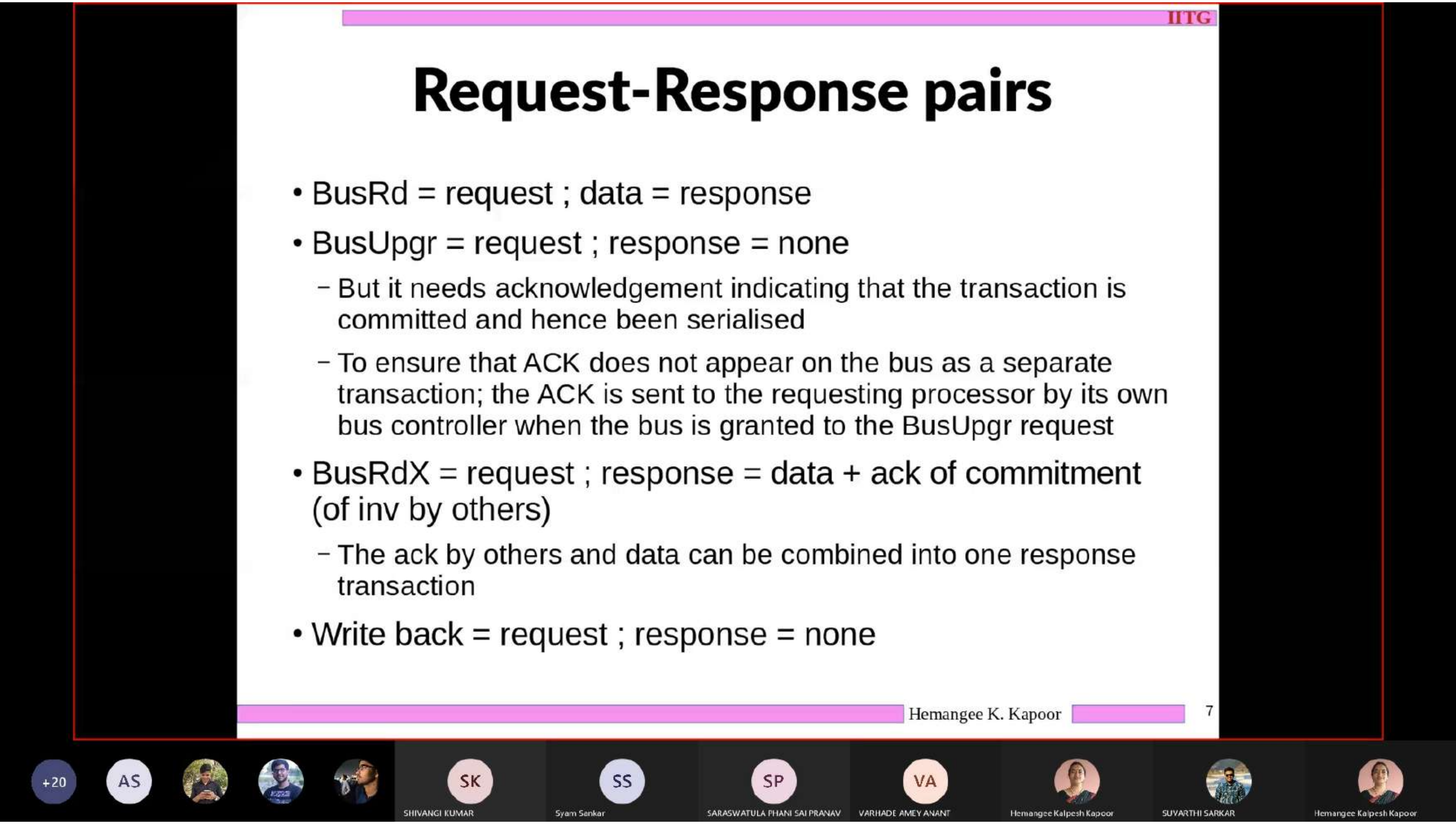
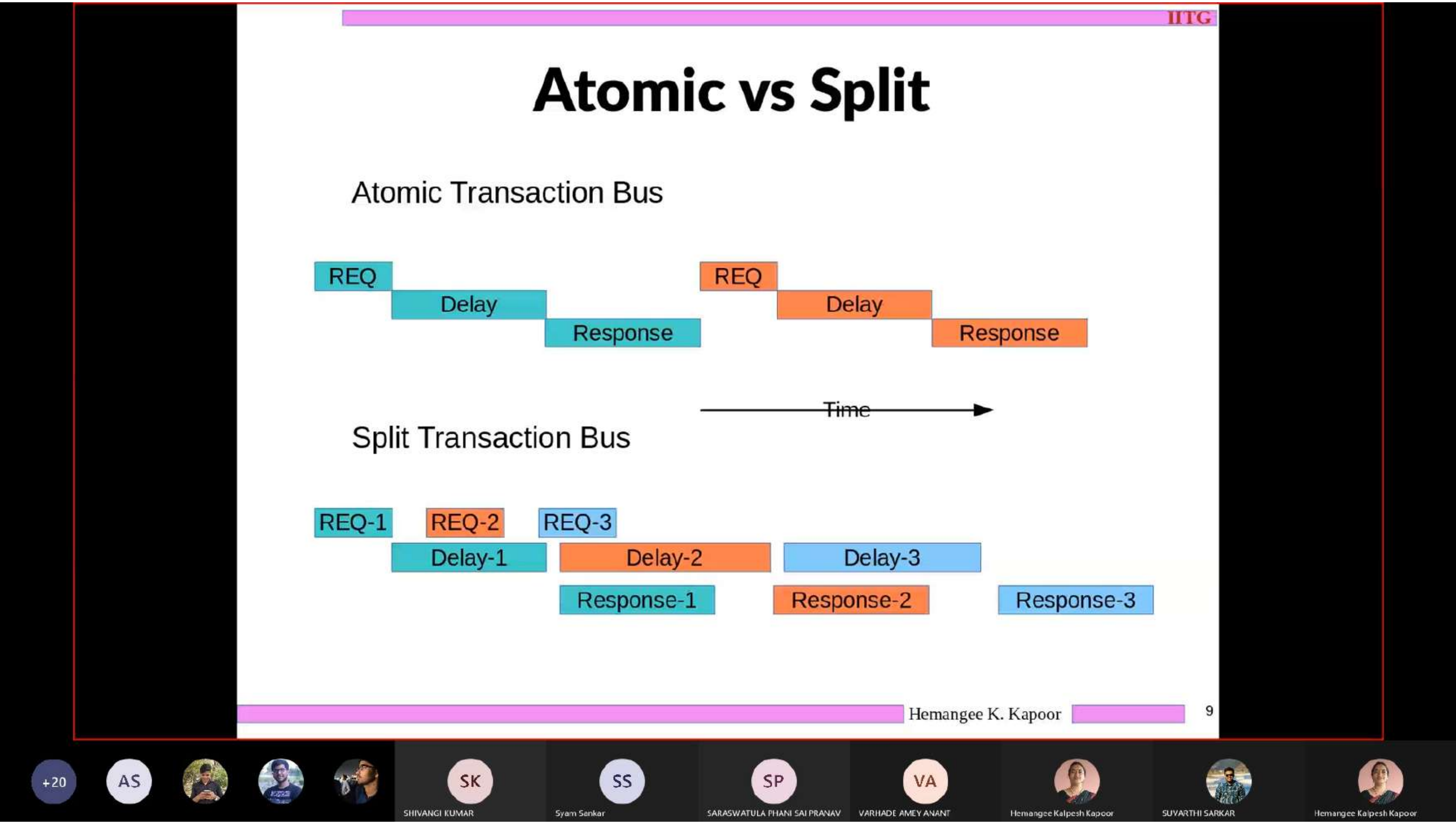


Split transaction bus + single level cache

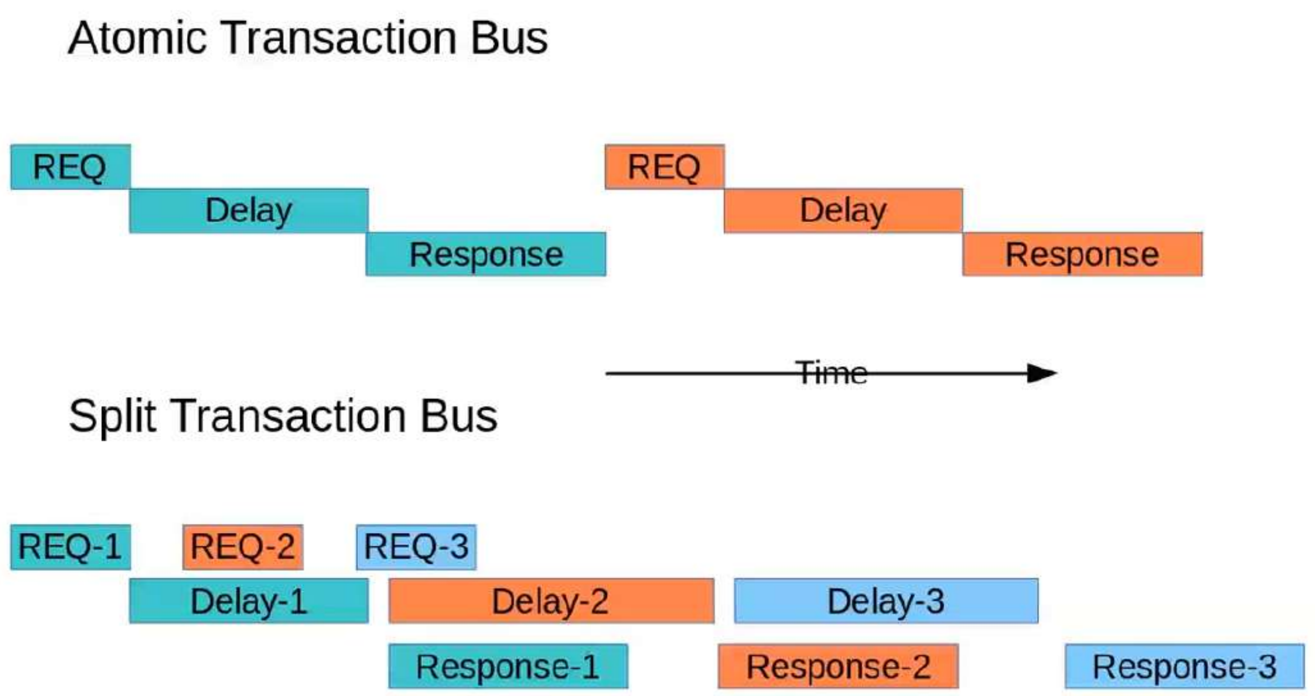


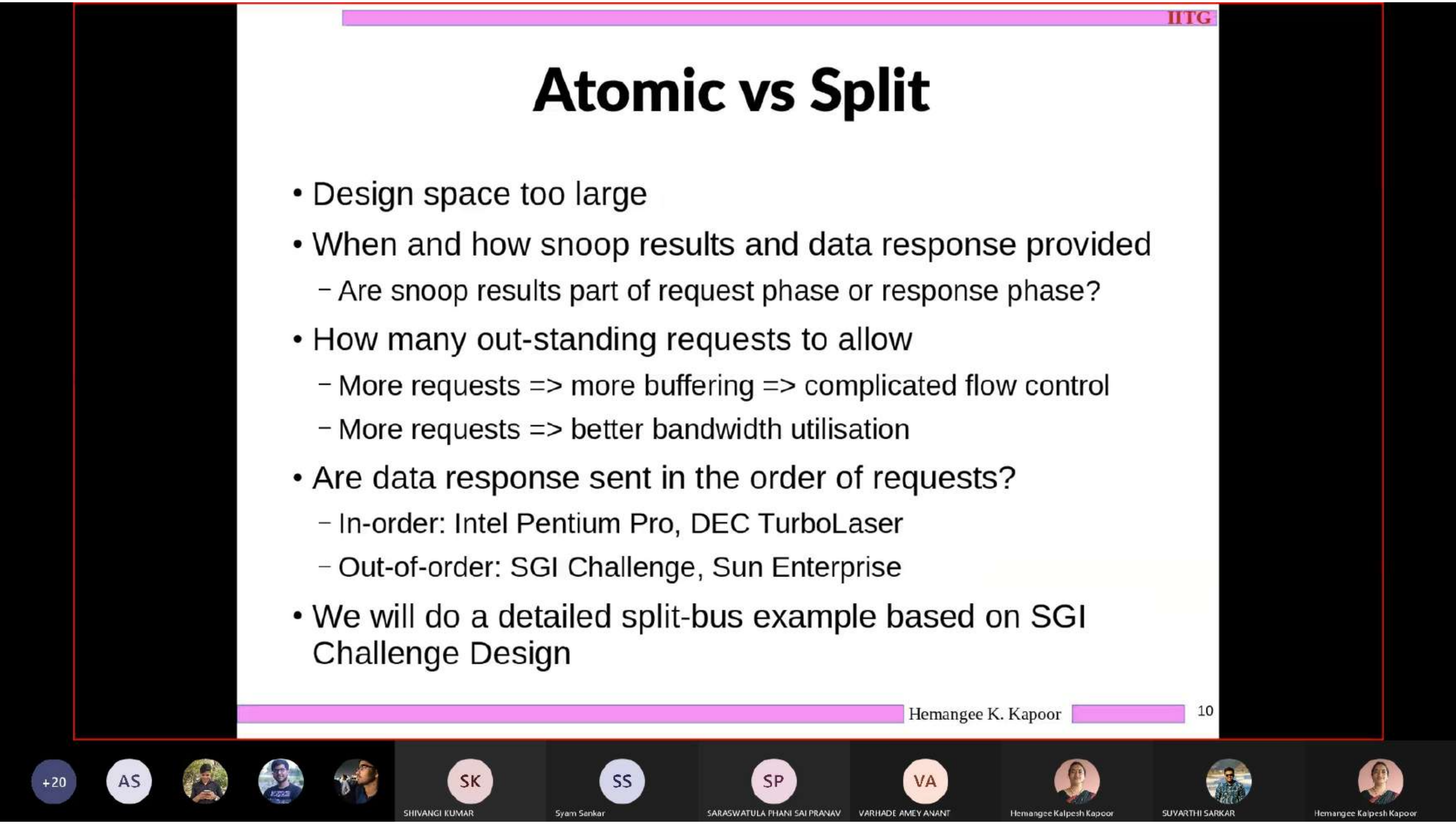
Request-Response pairs

- BusRd = request ; data = response
- BusUpgr = request ; response = none
  - But it needs acknowledgement indicating that the transaction is committed and hence been serialised
  - To ensure that ACK does not appear on the bus as a separate transaction; the ACK is sent to the requesting processor by its own bus controller when the bus is granted to the BusUpgr request
- BusRdX = request ; response = data + ack of commitment (of inv by others)
  - The ack by others and data can be combined into one response transaction
- Write back = request ; response = none



Atomic vs Split



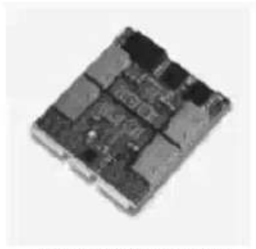


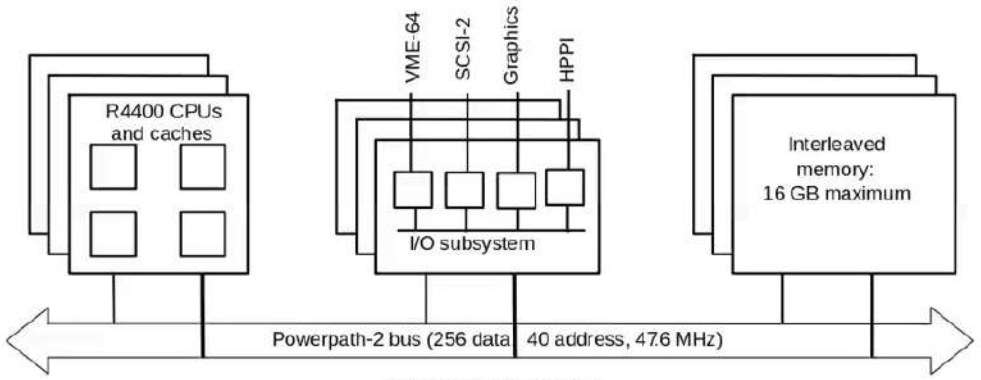
Atomic vs Split

- Design space too large
- When and how snoop results and data response provided
  - Are snoop results part of request phase or response phase?
- How many out-standing requests to allow
  - More requests => more buffering => complicated flow control
  - More requests => better bandwidth utilisation
- Are data response sent in the order of requests?
  - In-order: Intel Pentium Pro, DEC TurboLaser
  - Out-of-order: SGI Challenge, Sun Enterprise
- We will do a detailed split-bus example based on SGI Challenge Design

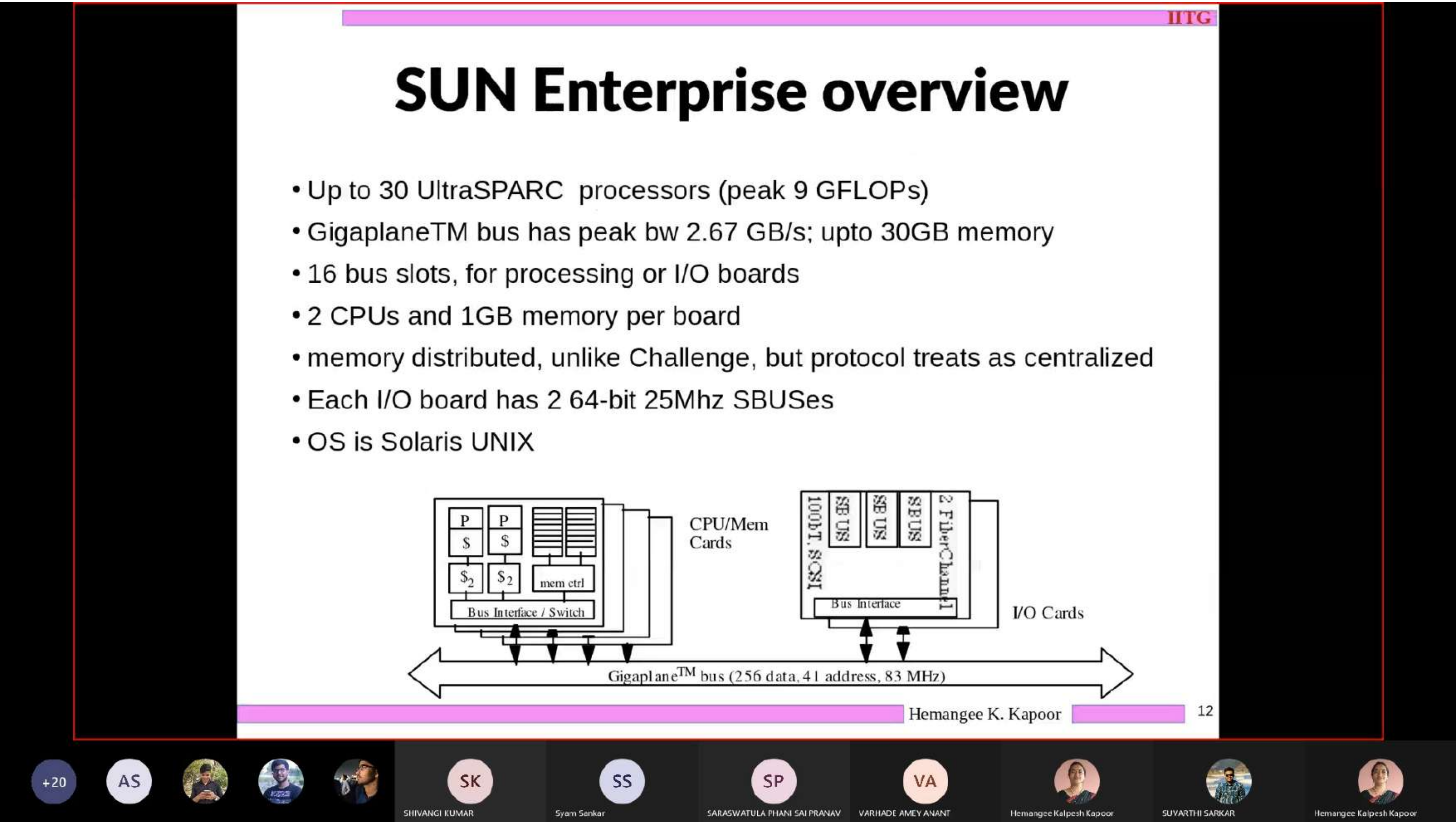
SGI Challenge overview

- 36 MIPS R4400 (peak 2.7 GFLOPS, 4 per board) or 18 MIPS R8000 (peak 5.4 GFLOPS, 2 per board)
- 8-way interleaved memory (up to 16 GB)
- 4 I/O busses of 320 MB/s each
- 1.2 GB/s Powerpath-2 bus @ 47.6 MHz, 16 slots, 329 signals
- 128 Bytes lines (1 + 4 cycles)
- Split-transaction with up to 8 outstanding reads
- all transactions take five cycles
- OS is variant of SVR4 UNIX called IRIX


(a) A four-processor board

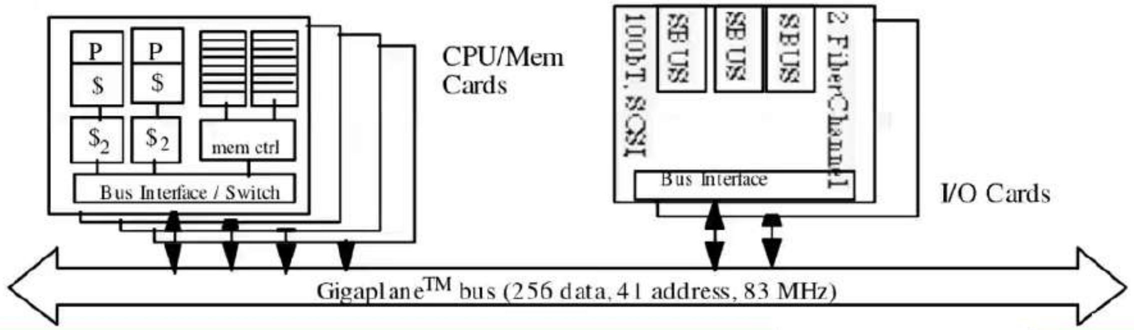

VME-64
SCSI-2
Graphics
HPPI
R4400 CPUs and caches
I/O subsystem
Interleaved memory: 16 GB maximum
Powerpath-2 bus (256 data, 40 address, 47.6 MHz)
(b) Machine organization

+20 AS SK SS SP VA
SHIVANGI KUMAR
Syam Sankar
SARASWATULA PHANI SAI PRANAV
VARHADE AMEY ANANT
Hemangee Kalpesh Kapoor
SUVARTHI SARKAR
Hemangee Kalpesh Kapoor



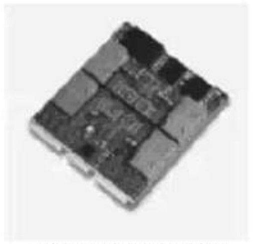
SUN Enterprise overview

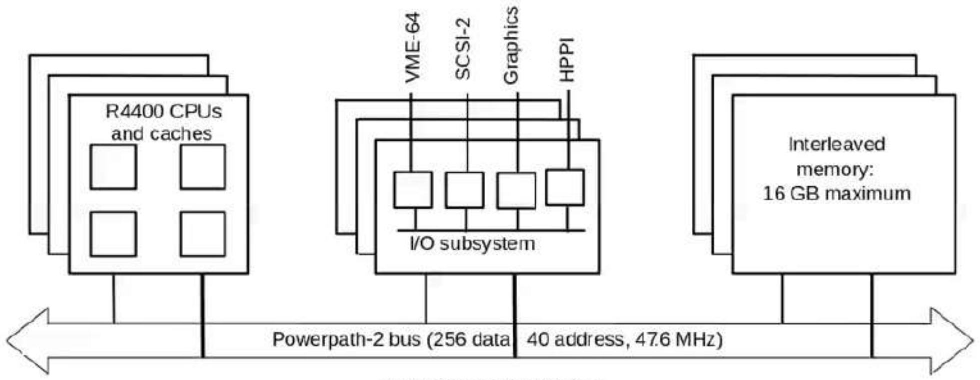
- Up to 30 UltraSPARC processors (peak 9 GFLOPs)
- GigaplaneTM bus has peak bw 2.67 GB/s; upto 30GB memory
- 16 bus slots, for processing or I/O boards
- 2 CPUs and 1GB memory per board
- memory distributed, unlike Challenge, but protocol treats as centralized
- Each I/O board has 2 64-bit 25Mhz SBUSes
- OS is Solaris UNIX



SGI Challenge overview

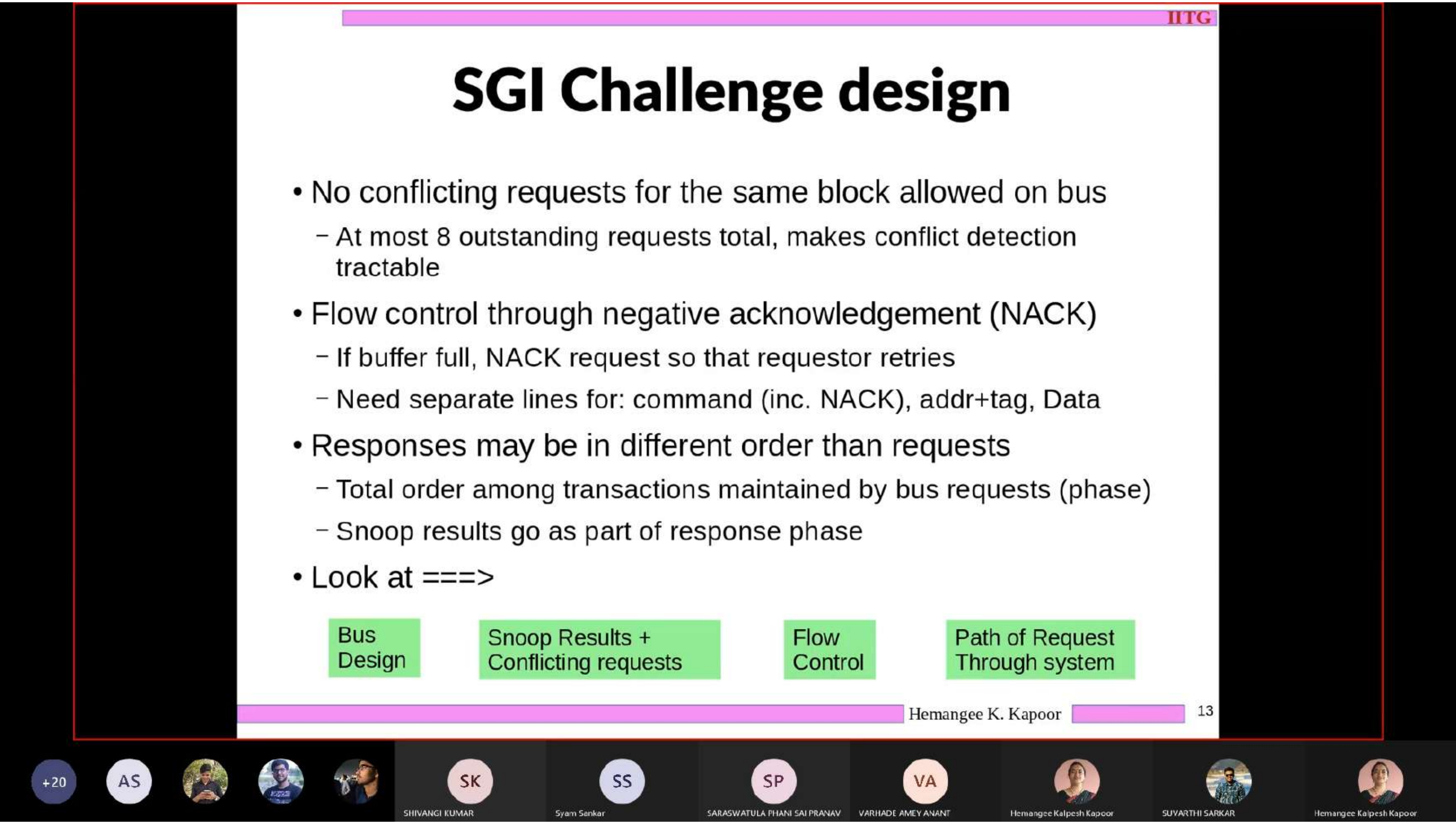
- 36 MIPS R4400 (peak 2.7 GFLOPS, 4 per board) or 18 MIPS R8000 (peak 5.4 GFLOPS, 2 per board)
- 8-way interleaved memory (up to 16 GB)
- 4 I/O busses of 320 MB/s each
- 1.2 GB/s Powerpath-2 bus @ 47.6 MHz, 16 slots, 329 signals
- 128 Bytes lines (1 + 4 cycles)
- Split-transaction with up to 8 outstanding reads
- all transactions take five cycles
- OS is variant of SVR4 UNIX called IRIX


(a) A four-processor board


VME-64
SCSI-2
Graphics
HPPI
R4400 CPUs and caches
I/O subsystem
Interleaved memory: 16 GB maximum
Powerpath-2 bus (256 data, 40 address, 47.6 MHz)
(b) Machine organization

+20 AS SK SS SP VA

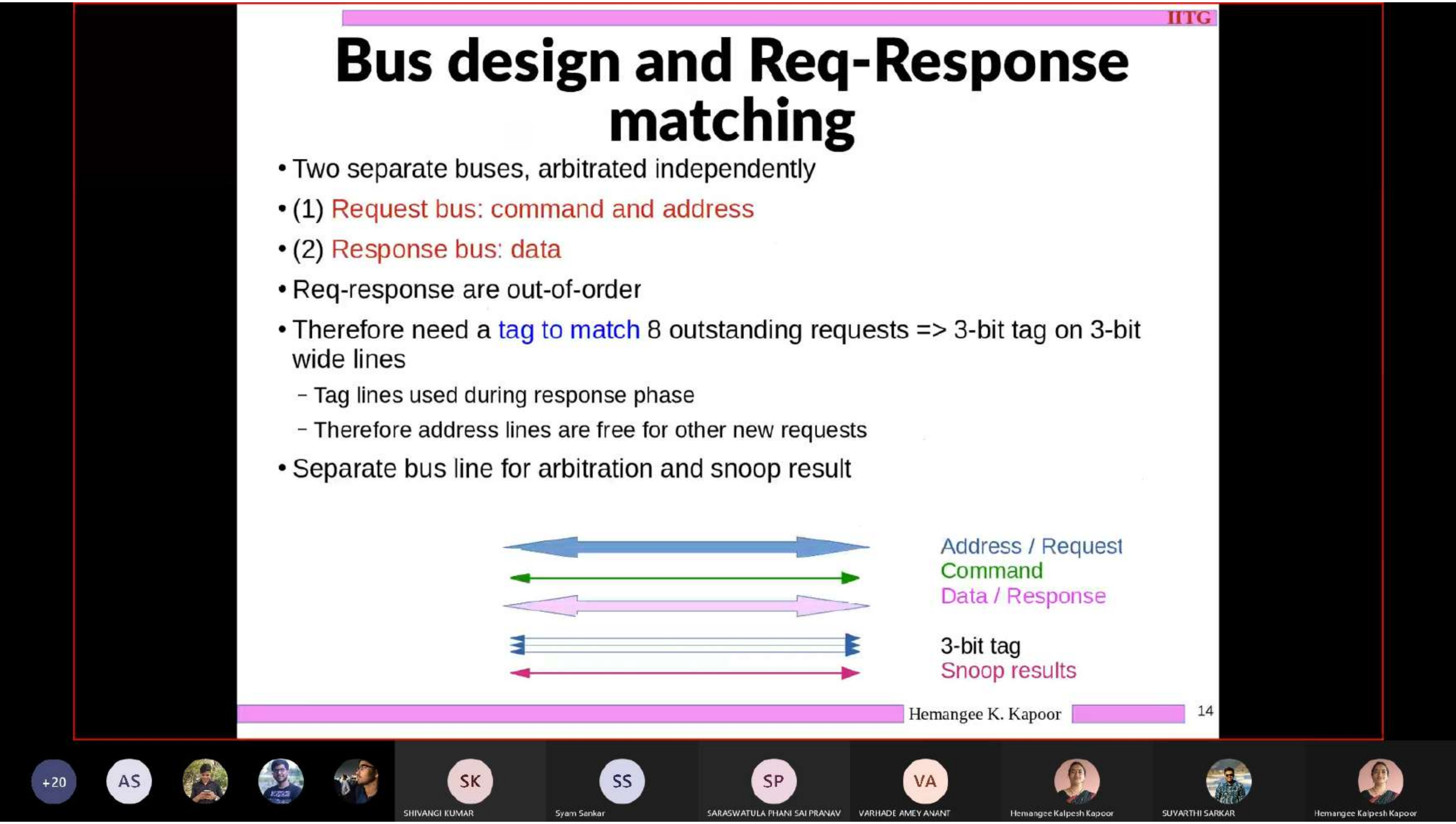
SHIVANGI KUMAR Syam Sankar SARASWATULA PHANI SAI PRANAV VARHADE AMEY ANANT Hemangee Kalpesh Kapoor SUVARTHI SARKAR Hemangee Kalpesh Kapoor



SGI Challenge design

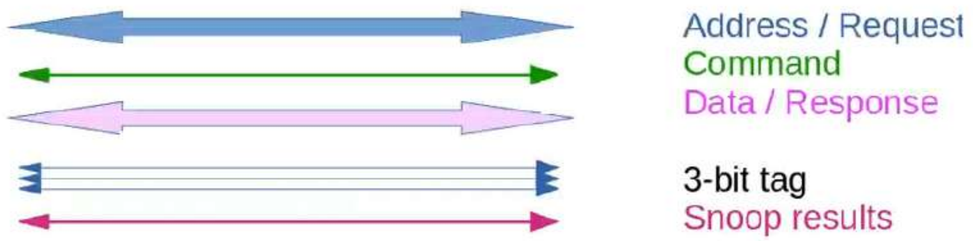
- No conflicting requests for the same block allowed on bus
  - At most 8 outstanding requests total, makes conflict detection tractable
- Flow control through negative acknowledgement (NACK)
  - If buffer full, NACK request so that requestor retries
  - Need separate lines for: command (inc. NACK), addr+tag, Data
- Responses may be in different order than requests
  - Total order among transactions maintained by bus requests (phase)
  - Snoop results go as part of response phase
- Look at ===>

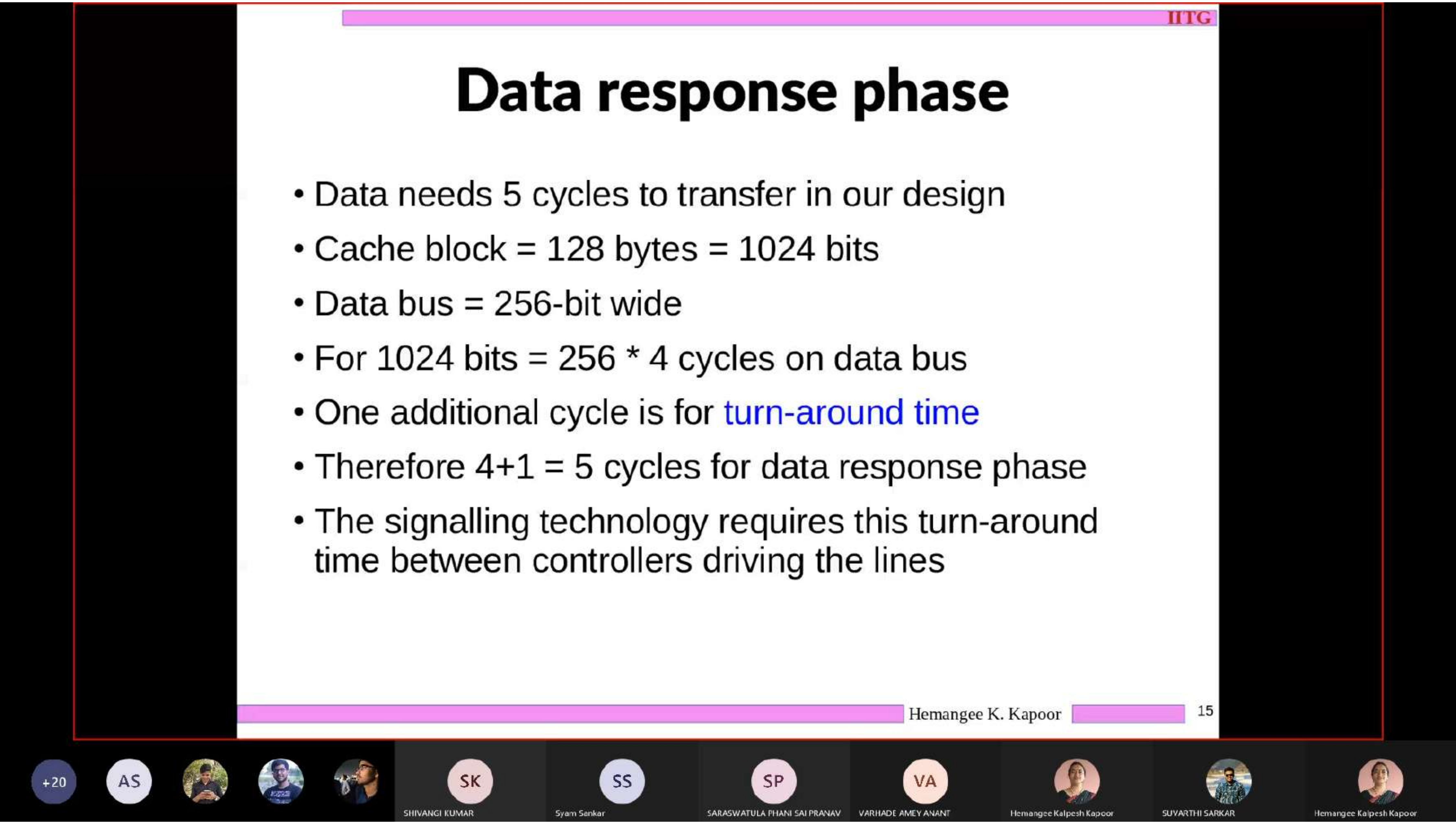
Bus Design | Snoop Results + Conflicting requests | Flow Control | Path of Request Through system



Bus design and Req-Response matching

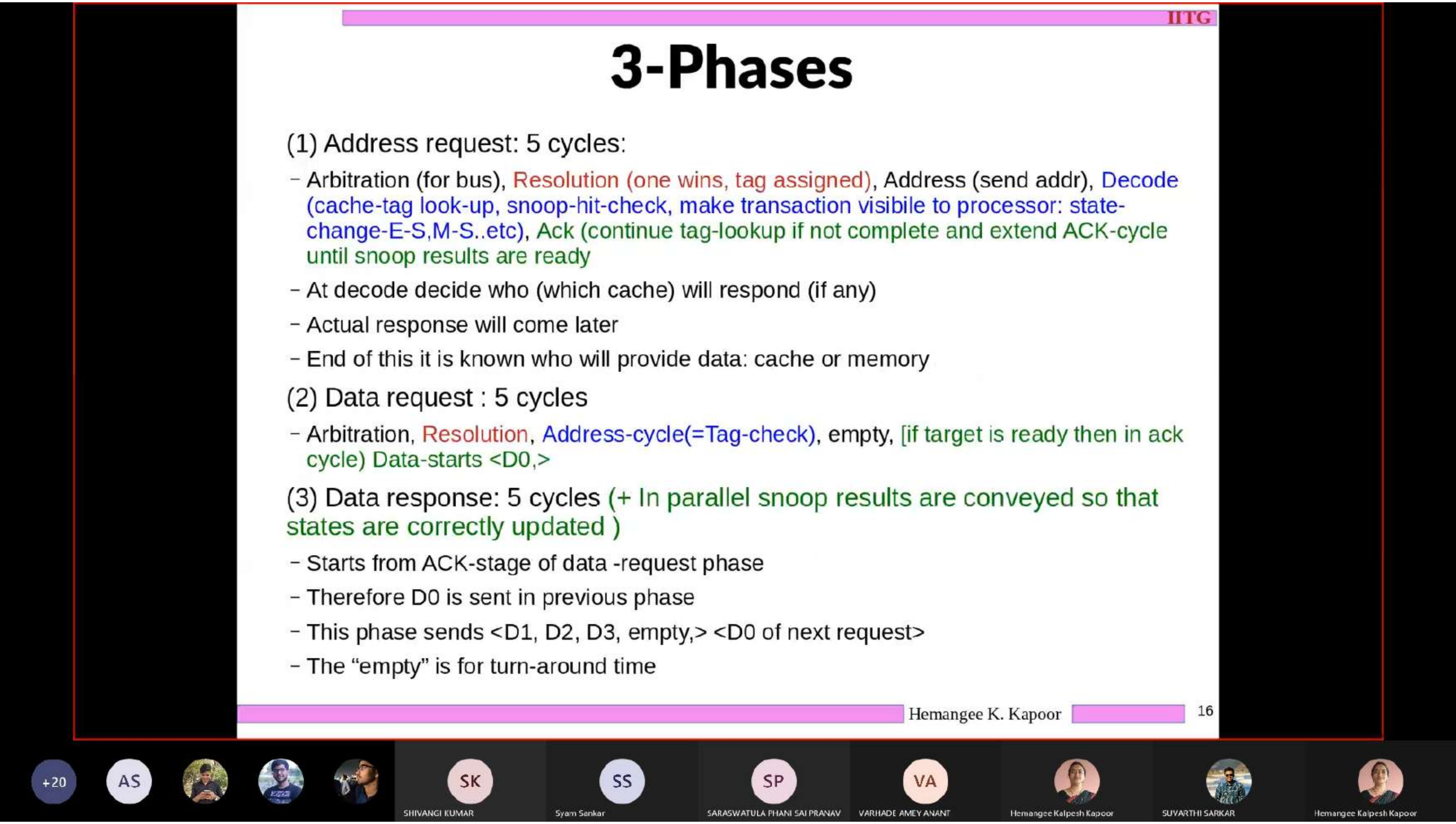
- Two separate buses, arbitrated independently
- (1) Request bus: command and address
- (2) Response bus: data
- Req-response are out-of-order
- Therefore need a tag to match 8 outstanding requests => 3-bit tag on 3-bit wide lines
  - Tag lines used during response phase
  - Therefore address lines are free for other new requests
- Separate bus line for arbitration and snoop result





Data response phase

- Data needs 5 cycles to transfer in our design
- Cache block = 128 bytes = 1024 bits
- Data bus = 256-bit wide
- For 1024 bits = 256 * 4 cycles on data bus
- One additional cycle is for turn-around time
- Therefore 4+1 = 5 cycles for data response phase
- The signalling technology requires this turn-around time between controllers driving the lines



3-Phases

(1) Address request: 5 cycles:

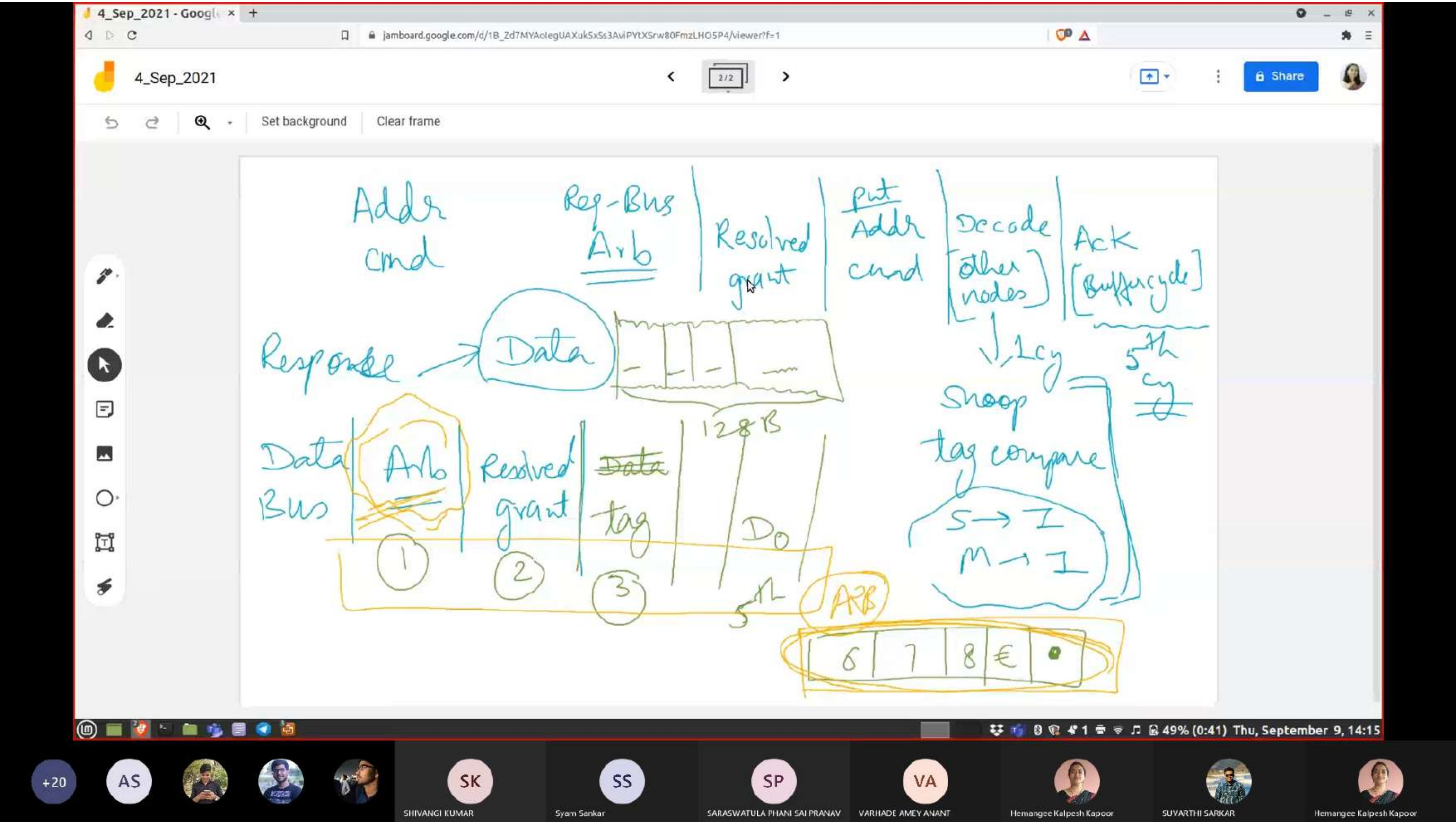
- Arbitration (for bus), Resolution (one wins, tag assigned), Address (send addr), Decode (cache-tag look-up, snoop-hit-check, make transaction visibile to processor: state-change-E-S,M-S..etc), Ack (continue tag-lookup if not complete and extend ACK-cycle until snoop results are ready
- At decode decide who (which cache) will respond (if any)
- Actual response will come later
- End of this it is known who will provide data: cache or memory

(2) Data request : 5 cycles

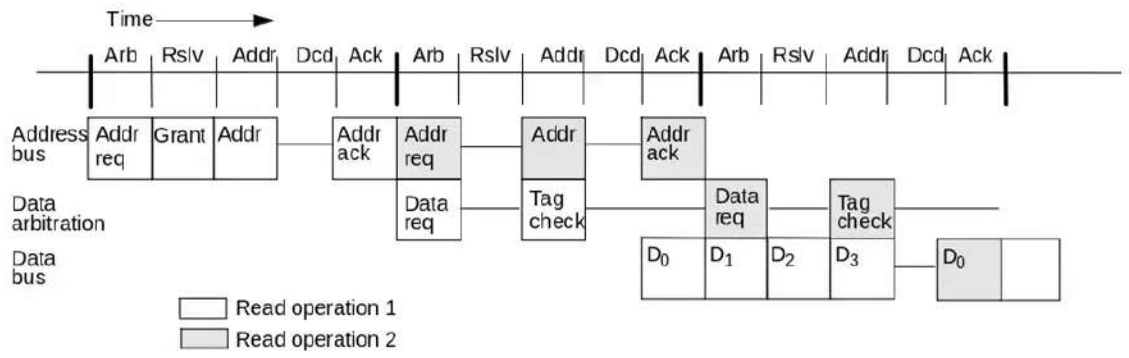
- Arbitration, Resolution, Address-cycle(=Tag-check), empty, [if target is ready then in ack cycle) Data-starts <D0,>

(3) Data response: 5 cycles (+ In parallel snoop results are conveyed so that states are correctly updated)

- Starts from ACK-stage of data -request phase
- Therefore D0 is sent in previous phase
- This phase sends <D1, D2, D3, empty,> <D0 of next request>
- The "empty" is for turn-around time


4_Sep_2021 - Google
jamboard.google.com/d/1B_Zd7MYAolegUAXukSxSs3AviPYtXSrw80FmzLHO5P4/viewer?f=1
4_Sep_2021
2/2
Share
Set background
Clear frame
Addr Cmd
Req-Bus Arb
Resolved grant
Put Addr cmd
Decode [other nodes]
Ack [Buffer cycle]
Response
Data
128B
1 cy
5th cy
Snoop tag compare
S → I
M → I
Data Bus
Arb
Resolved grant
~~Data~~
tag
D0
1
2
3
5th
ARB
6
7
8
€
49% (0:41) Thu, September 9, 14:15
+20
AS
SK
SHIVANGI KUMAR
SS
Syam Sankar
SP
SARASWATULA PHANI SAI PRANAV
VA
VARHADE AMEY ANANT
Hemangee Kalpesh Kapoor
SUVARTHI SARKAR
Hemangee Kalpesh Kapoor

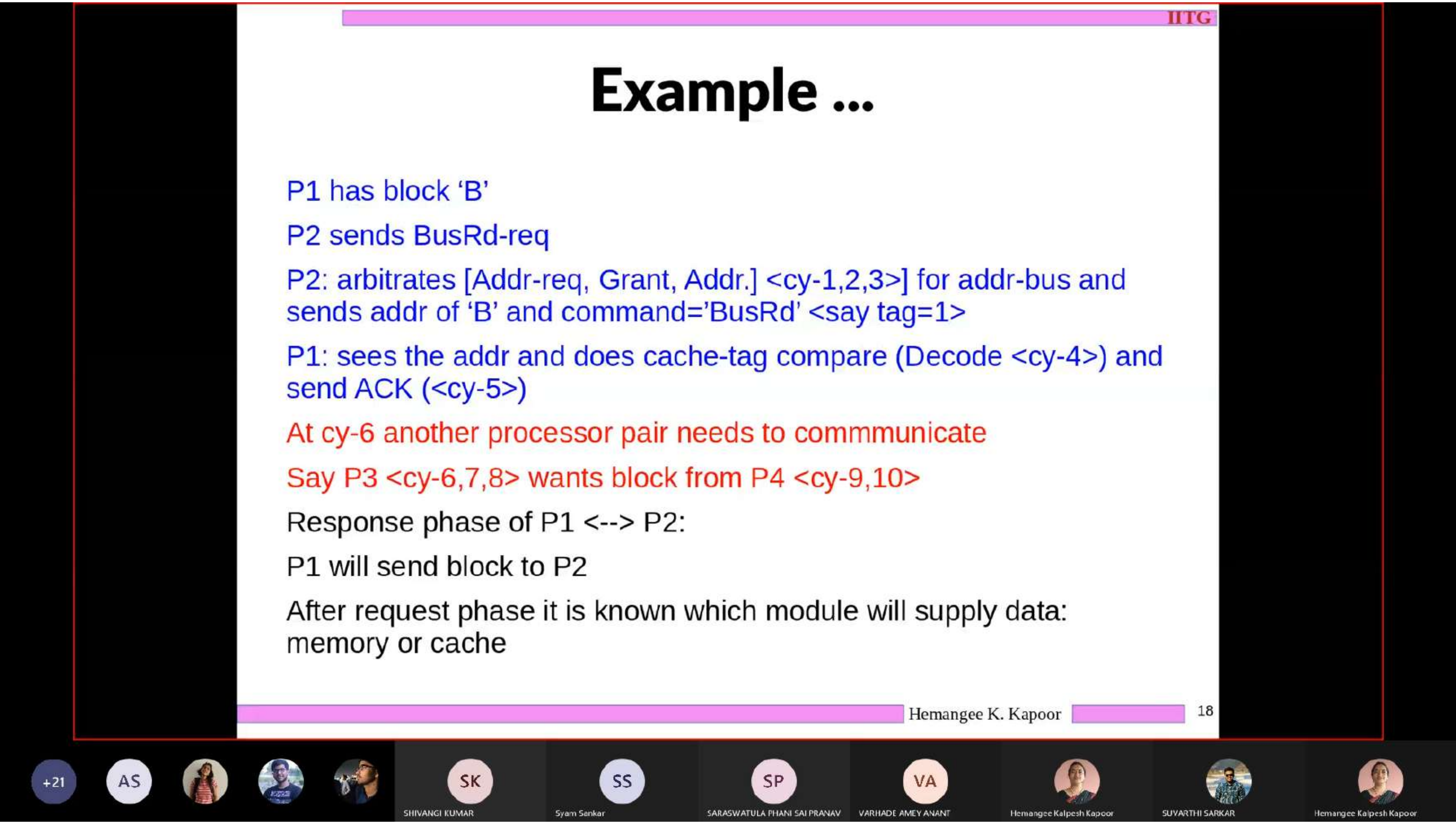
Ex: Read request that results in data transfer



- Scenario: cache has block in modified state
  - BusRd: send the block and state change E--> S
  - BusRdx or BusUpgr: send block and E --> I (inv.)

+21 AS SK SS SP VA

SHIVANGI KUMAR · Syam Sankar · SARASWATULA PHANI SAI PRANAV · VARHADE AMEY ANANT · Hemangee Kalpesh Kapoor · SUVARTHI SARKAR · Hemangee Kalpesh Kapoor



Example ...

P1 has block ‘B’

P2 sends BusRd-req

P2: arbitrates [Addr-req, Grant, Addr.] <cy-1,2,3>] for addr-bus and sends addr of ‘B’ and command=’BusRd’ <say tag=1>

P1: sees the addr and does cache-tag compare (Decode <cy-4>) and send ACK (<cy-5>)

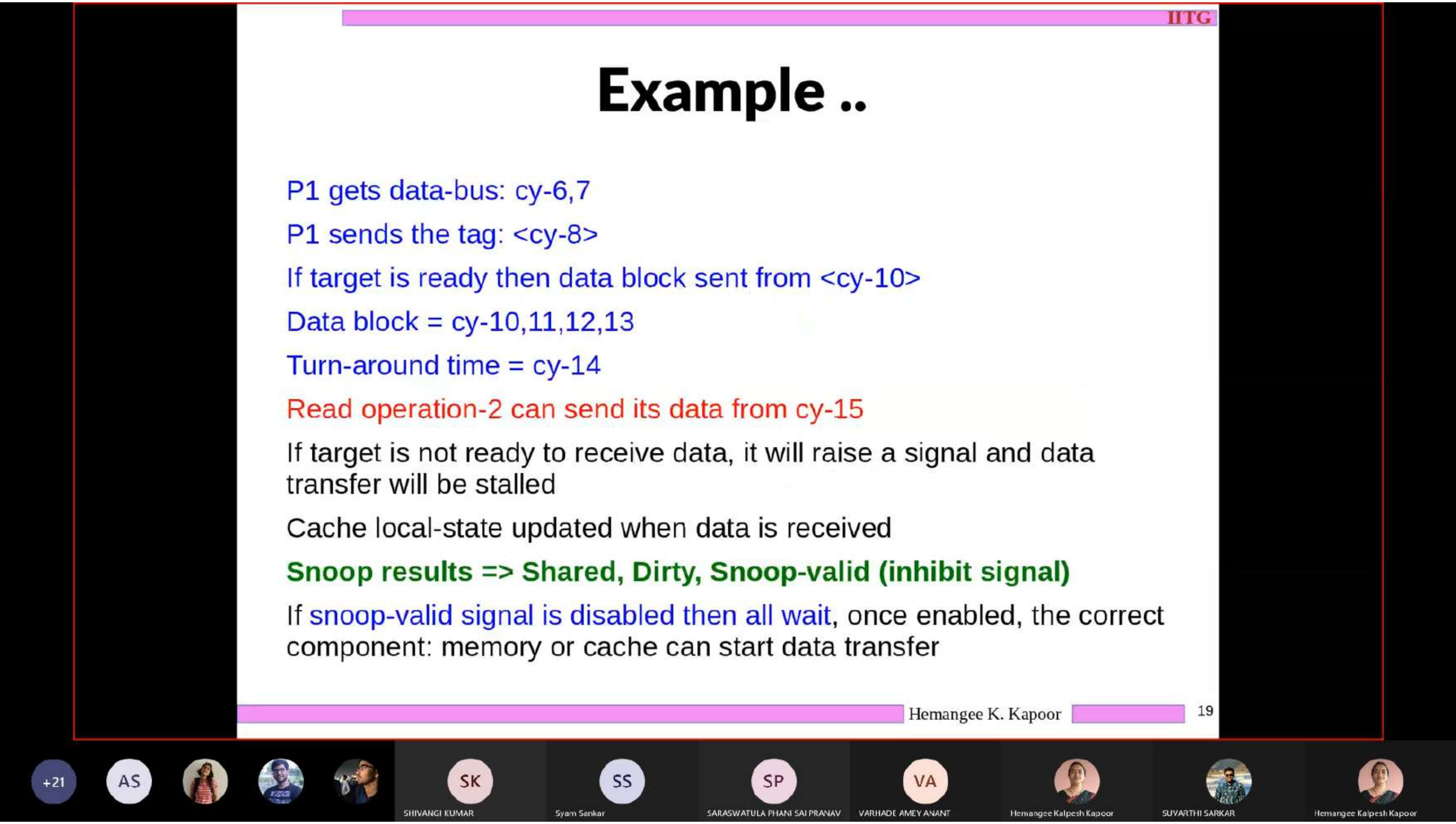
At cy-6 another processor pair needs to commmunicate

Say P3 <cy-6,7,8> wants block from P4 <cy-9,10>

Response phase of P1 <--> P2:

P1 will send block to P2

After request phase it is known which module will supply data: memory or cache



Example ..

P1 gets data-bus: cy-6,7

P1 sends the tag: <cy-8>

If target is ready then data block sent from <cy-10>

Data block = cy-10,11,12,13

Turn-around time = cy-14

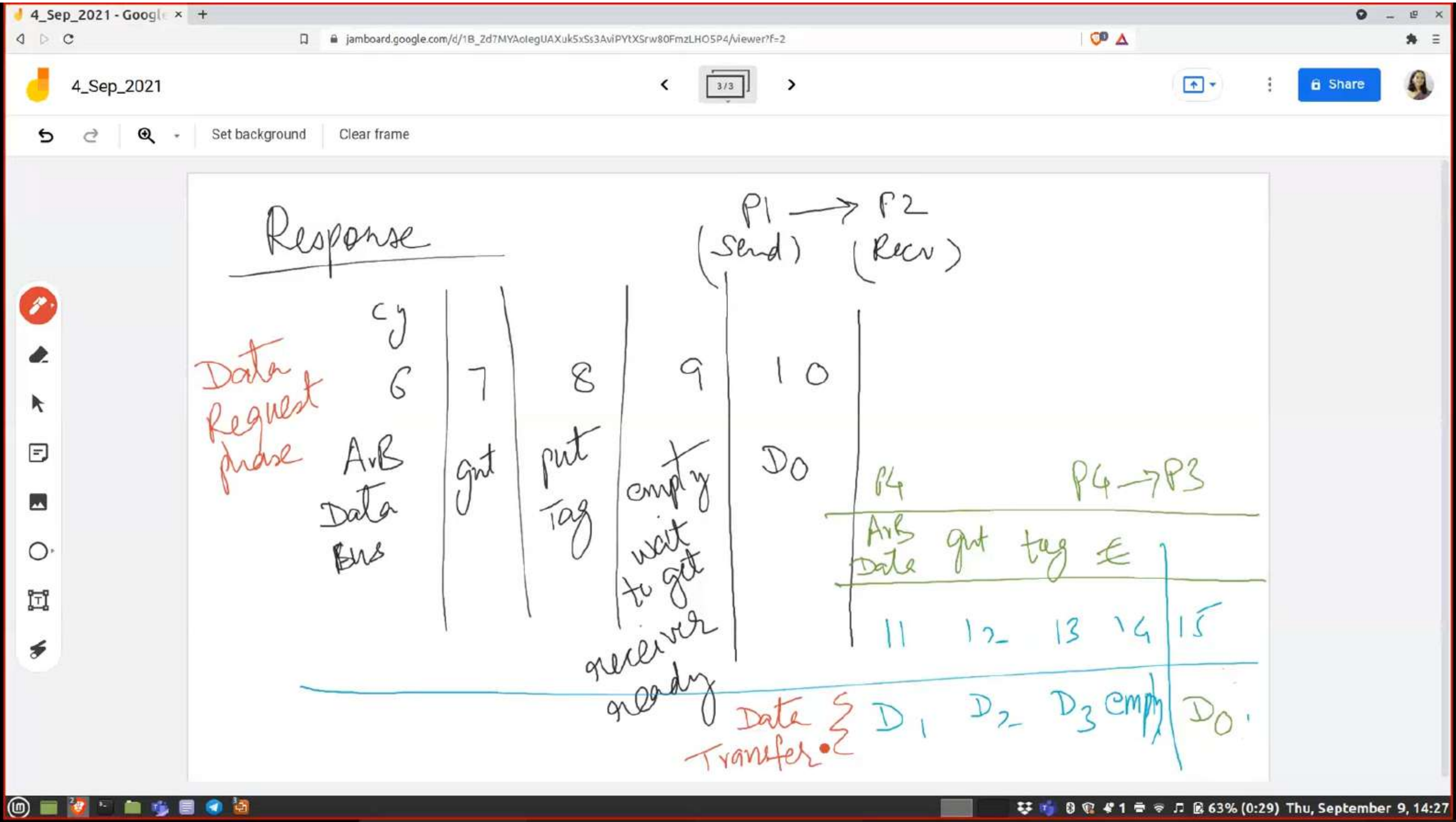
Read operation-2 can send its data from cy-15

If target is not ready to receive data, it will raise a signal and data transfer will be stalled

Cache local-state updated when data is received

Snoop results => Shared, Dirty, Snoop-valid (inhibit signal)

If snoop-valid signal is disabled then all wait, once enabled, the correct component: memory or cache can start data transfer


4_Sep_2021 - Googl
jamboard.google.com/d/1B_Zd7MYAolegUAXuk5xSs3AviPYtXSrw80FmzLHO5P4/viewer?f=2
4_Sep_2021
3/3
Share
Set background
Clear frame
63% (0:29) Thu, September 9, 14:27

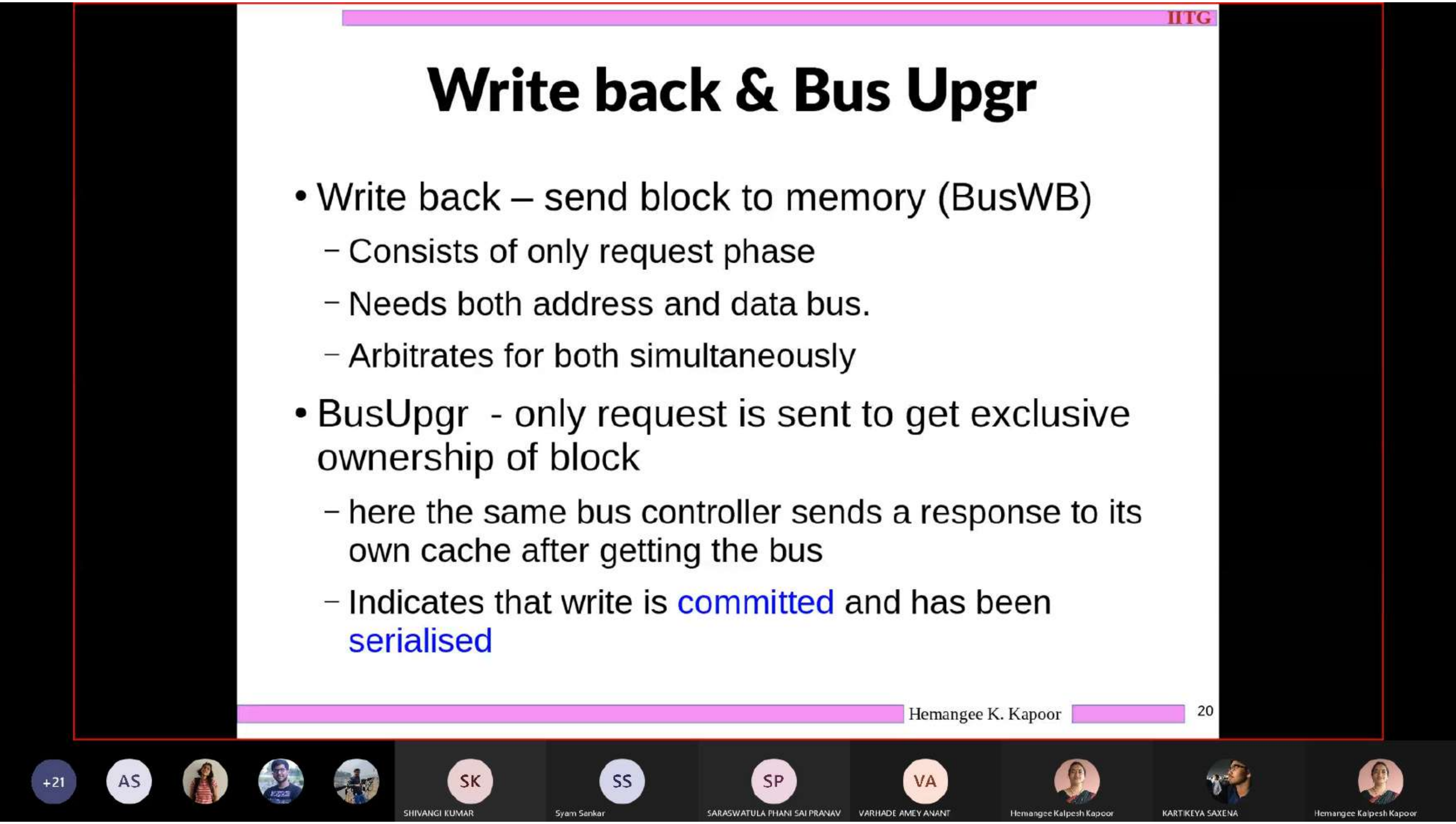
Response

P1 ⟶ P2
(Send) (Recv)

	cy				
	6	7	8	9	10
Data Request phase	A v B Data Bus	get	put Tag	empty wait to get receiver ready	D0

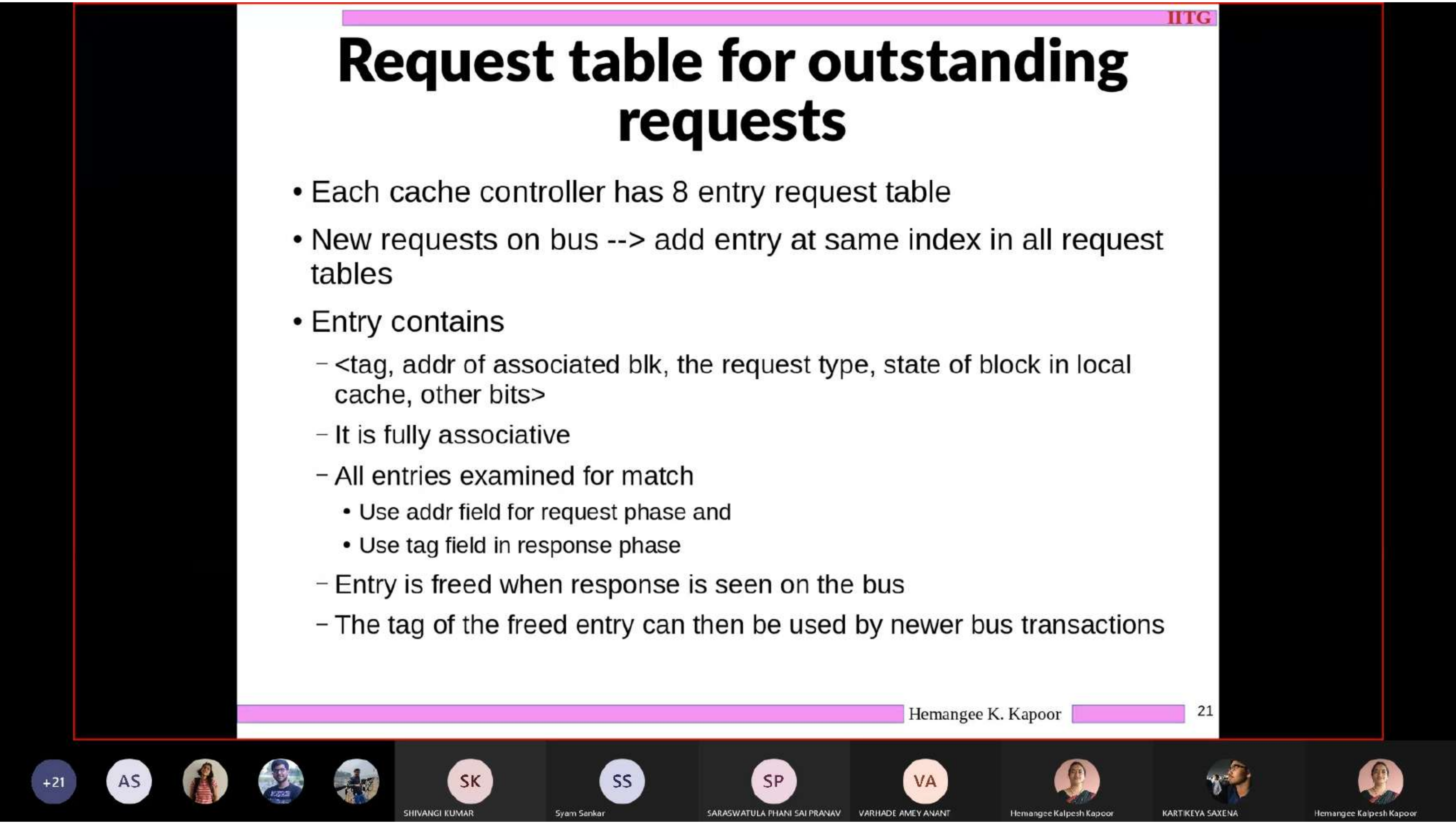
P4 ⟶ P3

P4				
A v B Data	get	tag	E	
11	12	13	14	15
Data Transfer: D1	D2	D3	empty	D0



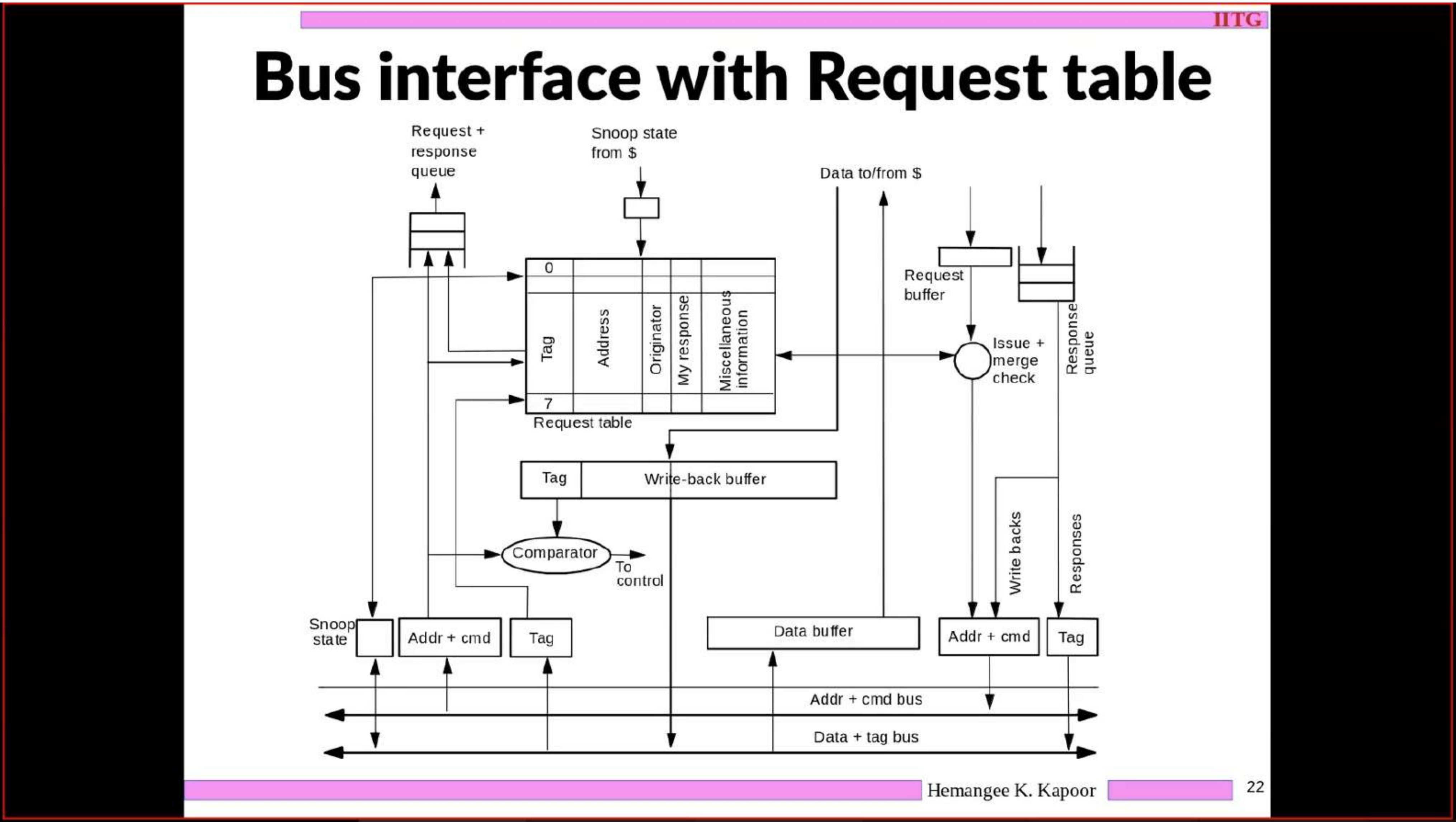
Write back & Bus Upgr

- Write back – send block to memory (BusWB)
  - Consists of only request phase
  - Needs both address and data bus.
  - Arbitrates for both simultaneously
- BusUpgr - only request is sent to get exclusive ownership of block
  - here the same bus controller sends a response to its own cache after getting the bus
  - Indicates that write is committed and has been serialised

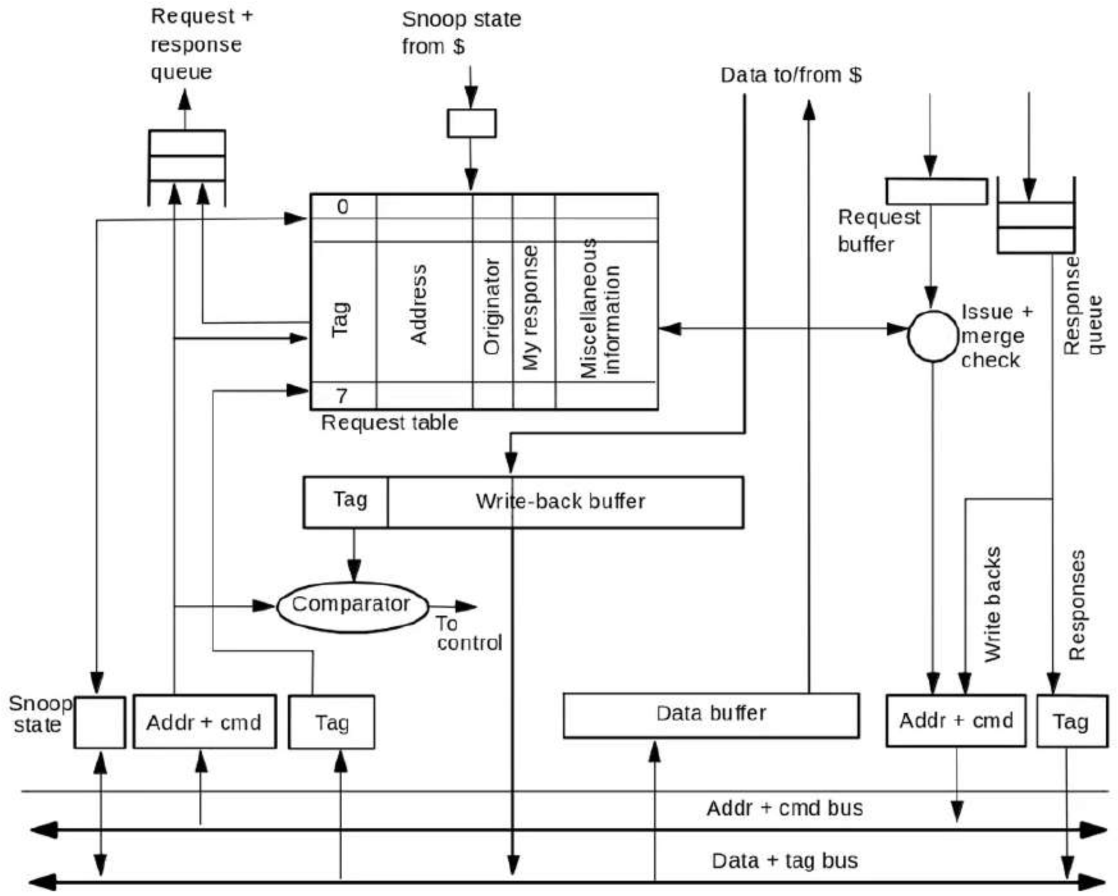


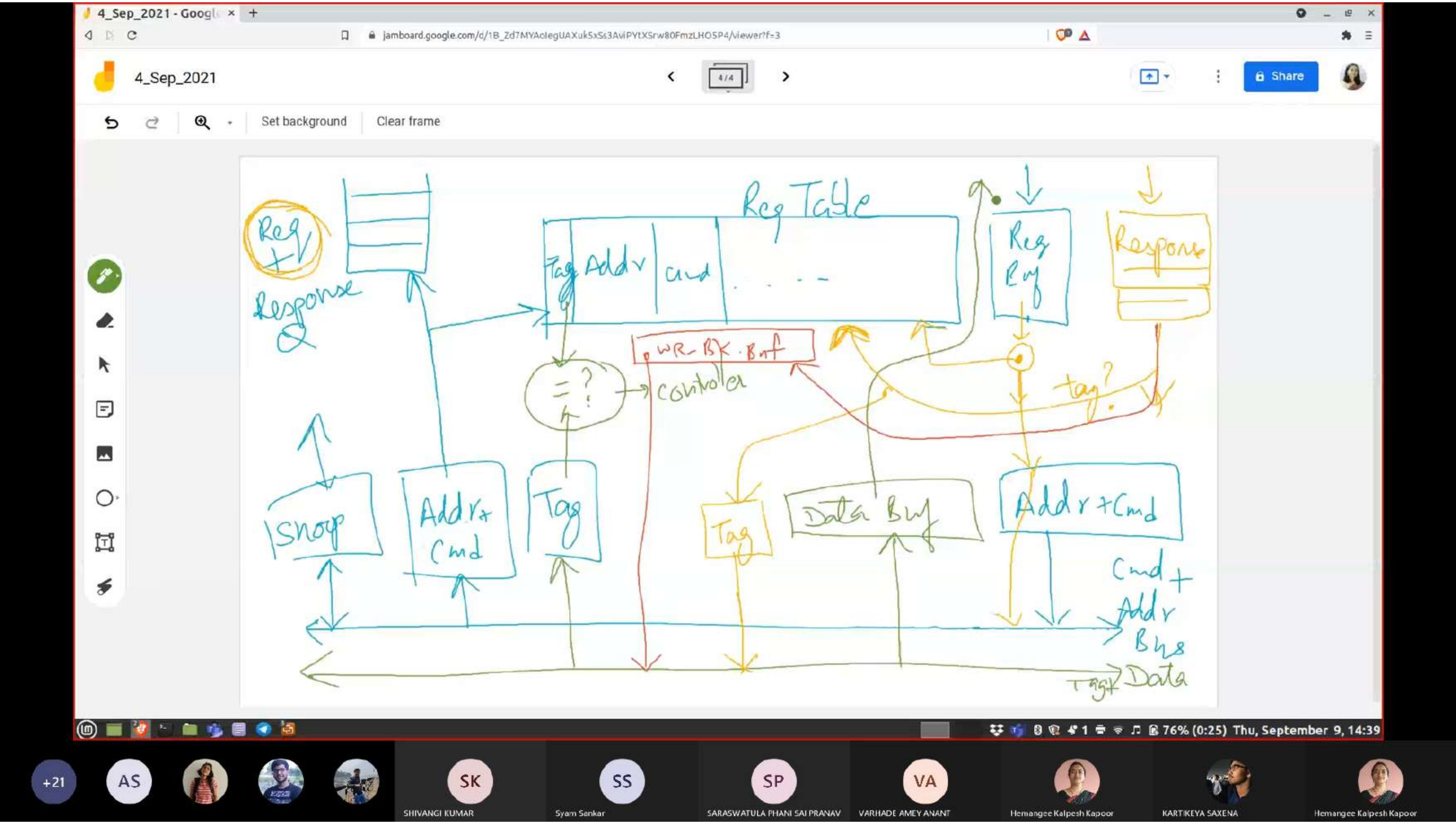
Request table for outstanding requests

- Each cache controller has 8 entry request table
- New requests on bus --> add entry at same index in all request tables
- Entry contains
  - <tag, addr of associated blk, the request type, state of block in local cache, other bits>
  - It is fully associative
  - All entries examined for match
    - Use addr field for request phase and
    - Use tag field in response phase
  - Entry is freed when response is seen on the bus
  - The tag of the freed entry can then be used by newer bus transactions

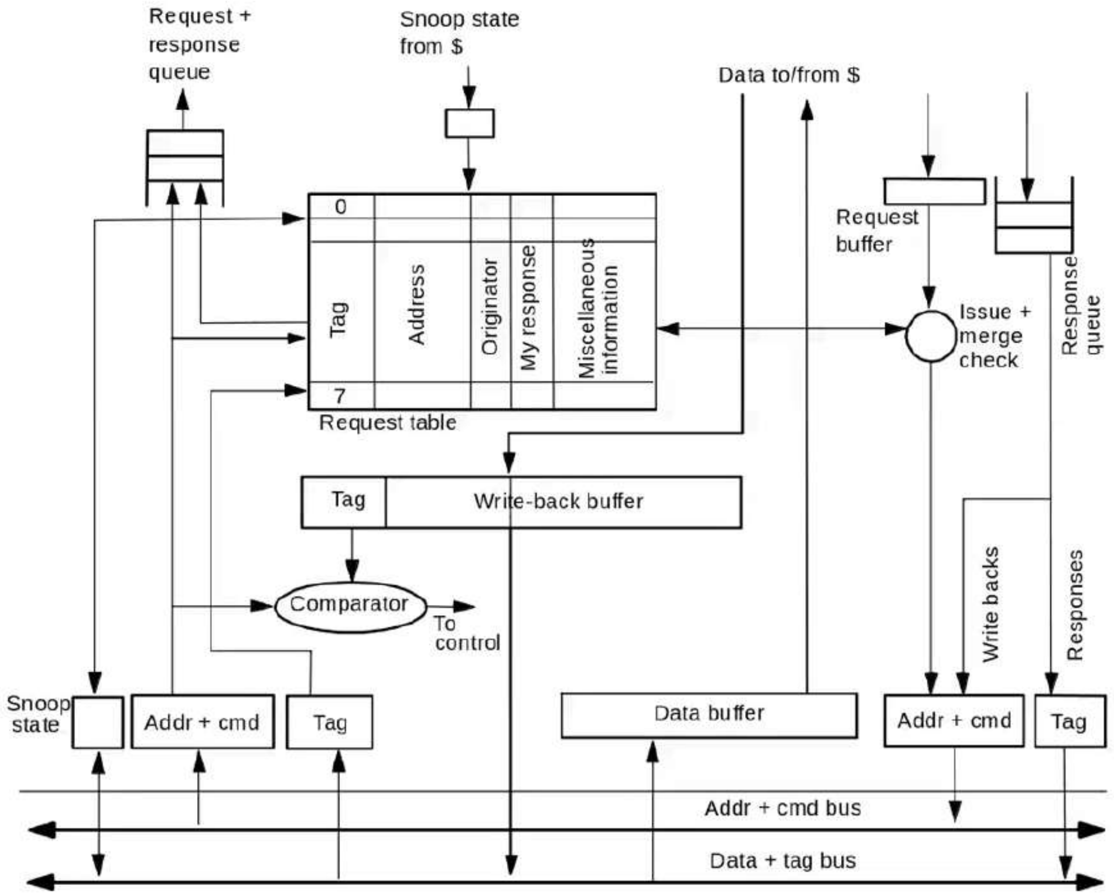


Bus interface with Request table

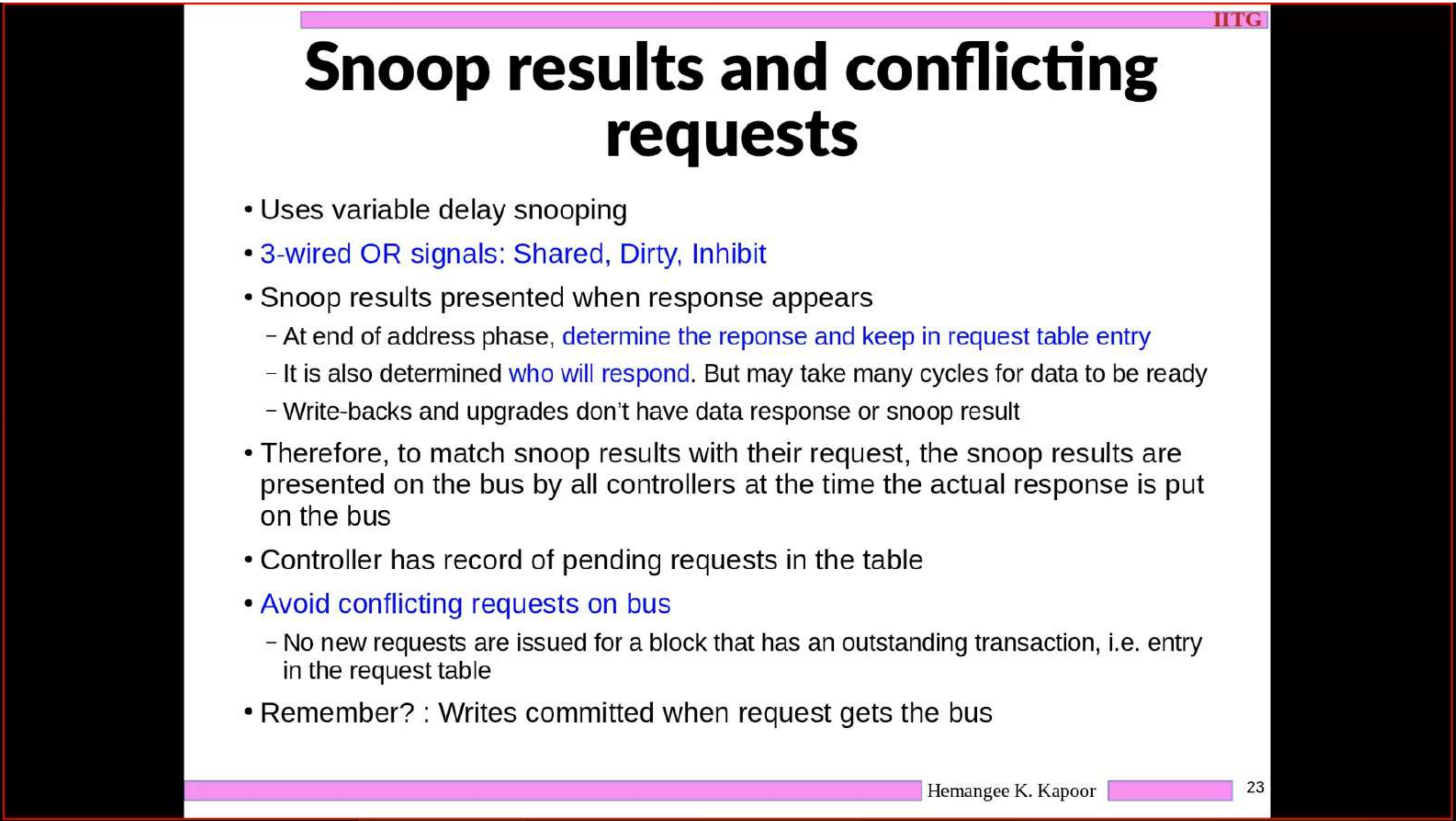



4_Sep_2021 - Google
jamboard.google.com/d/1B_Zd7MYAolegUAXuk5xSs3AviPYtXSrw80FmzLHO5P4/viewer?f=3
4_Sep_2021
4/4
Share
Set background
Clear frame
Req + Response
Response Q
Req Table
Tag
Addr
Cmd
. . . . .
Req Buf
Response
WR-BK.Buf
=?
controller
tag?
Snoop
Addr + Cmd
Tag
Tag
Data Buf
Addr + Cmd
Cmd + Addr Bus
Tag + Data
76% (0:25) Thu, September 9, 14:39
+21
AS
SK
SHIVANGI KUMAR
SS
Syam Sankar
SP
SARASWATULA PHANI SAI PRANAV
VA
VARHADE AMEY ANANT
Hemangee Kalpesh Kapoor
KARTIKEYA SAXENA
Hemangee Kalpesh Kapoor

Bus interface with Request table

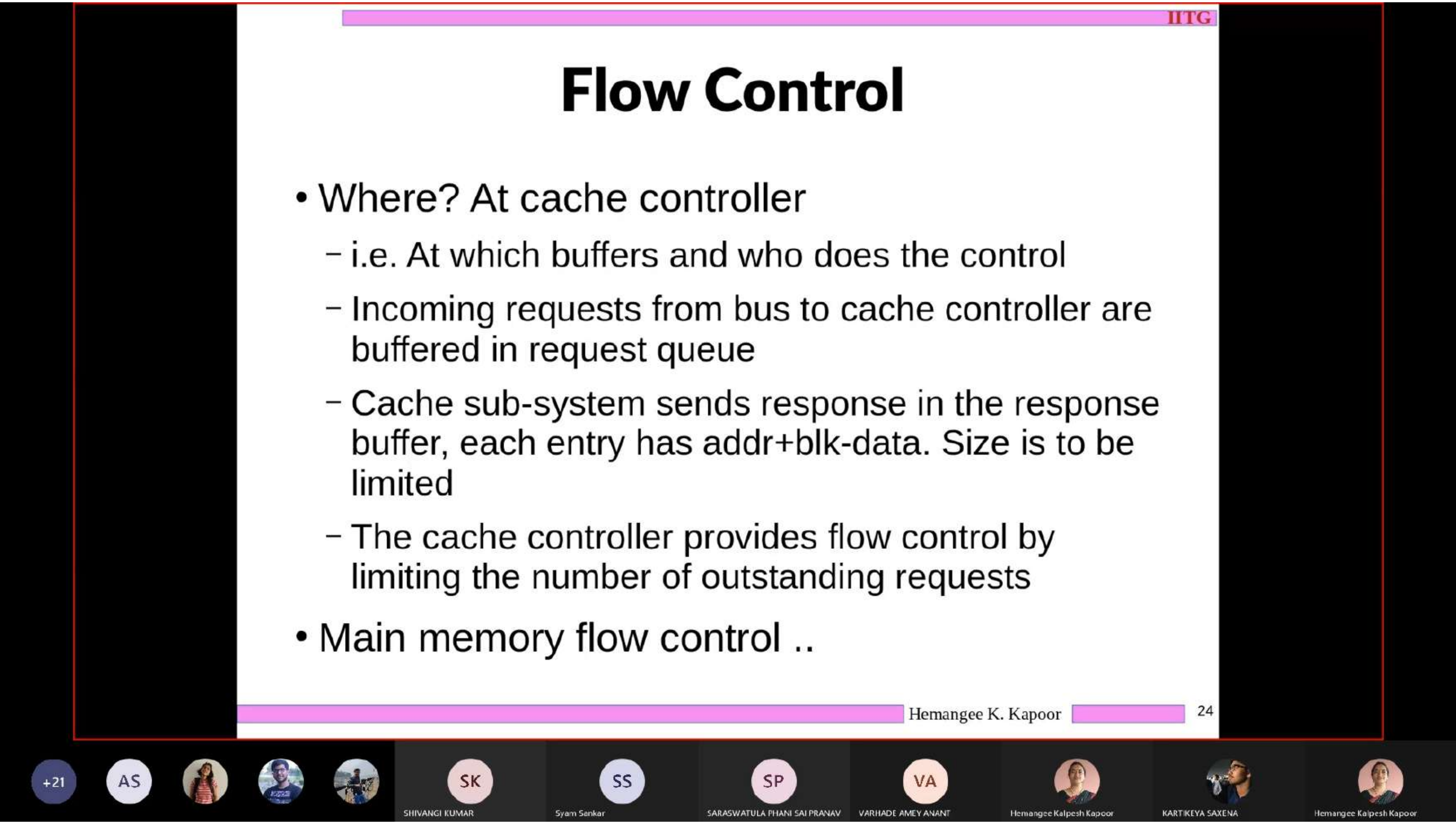


+21 AS SHIVANGI KUMAR Syam Sankar SARASWATULA PHANI SAI PRANAV VARHADE AMEY ANANT Hemangee Kalpesh Kapoor KARTIKEYA SAXENA Hemangee Kalpesh Kapoor



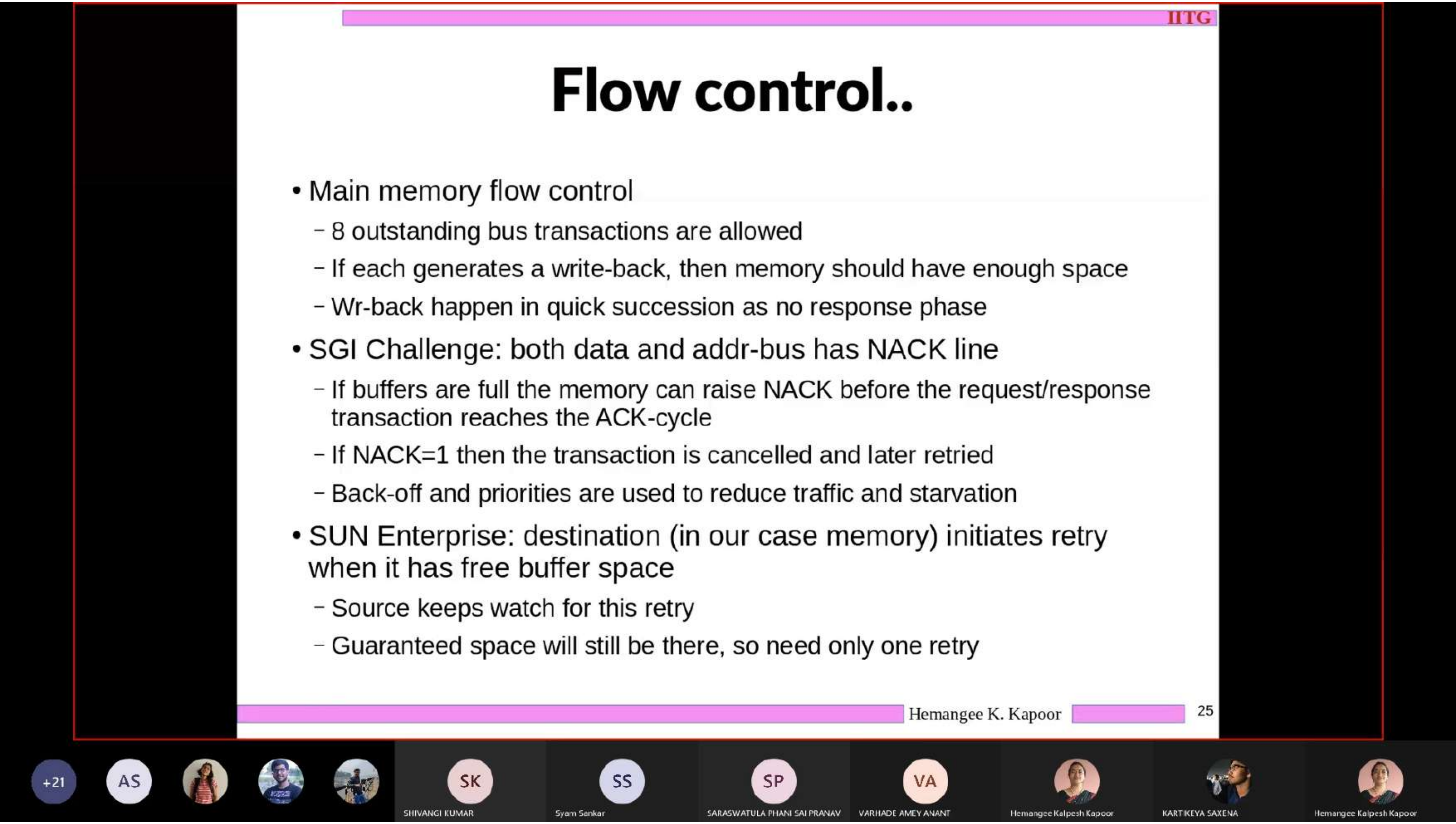
Snoop results and conflicting requests

- Uses variable delay snooping
- 3-wired OR signals: Shared, Dirty, Inhibit
- Snoop results presented when response appears
  - At end of address phase, determine the reponse and keep in request table entry
  - It is also determined who will respond. But may take many cycles for data to be ready
  - Write-backs and upgrades don't have data response or snoop result
- Therefore, to match snoop results with their request, the snoop results are presented on the bus by all controllers at the time the actual response is put on the bus
- Controller has record of pending requests in the table
- Avoid conflicting requests on bus
  - No new requests are issued for a block that has an outstanding transaction, i.e. entry in the request table
- Remember? : Writes committed when request gets the bus



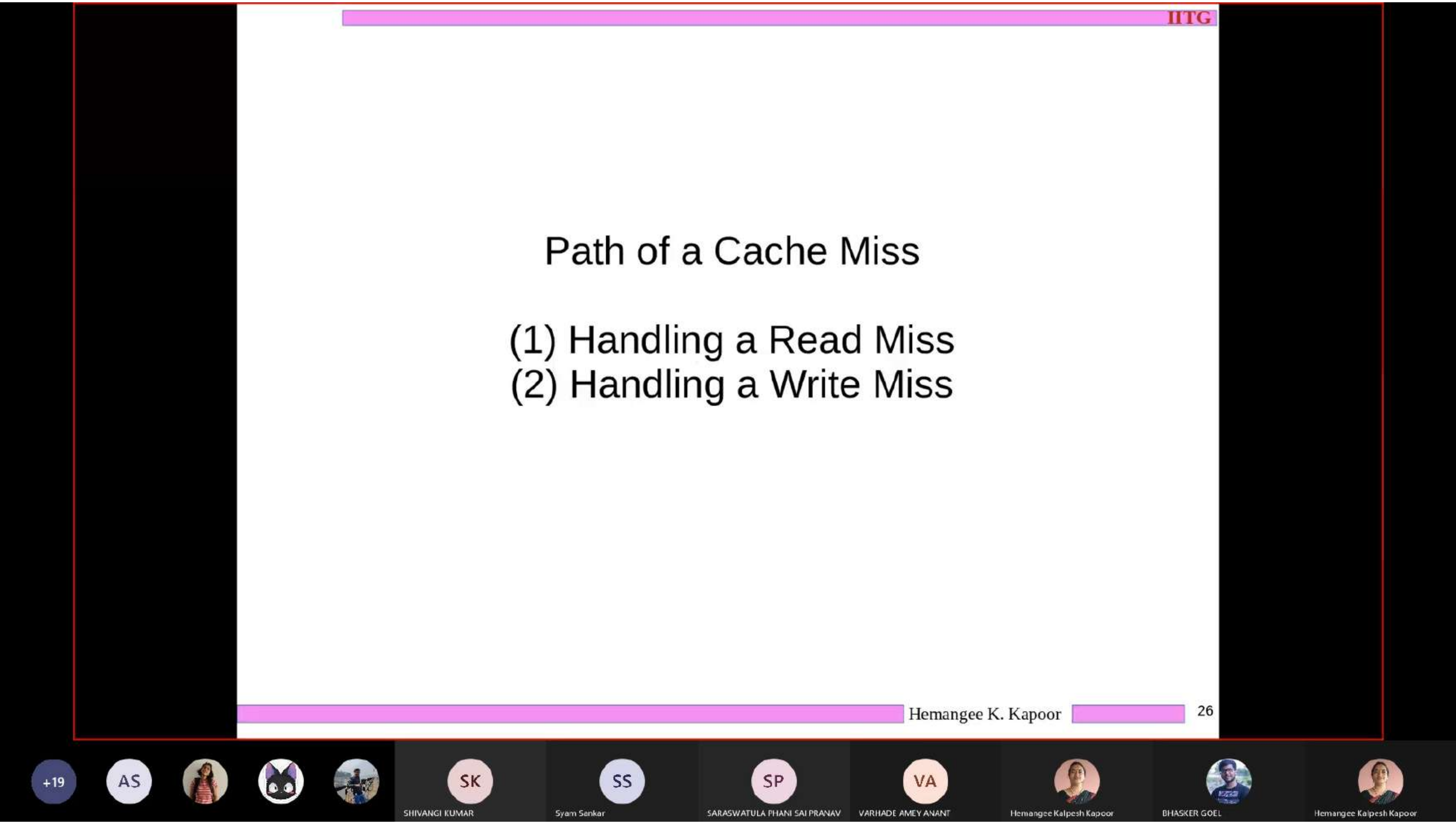
Flow Control

- Where? At cache controller
  - i.e. At which buffers and who does the control
  - Incoming requests from bus to cache controller are buffered in request queue
  - Cache sub-system sends response in the response buffer, each entry has addr+blk-data. Size is to be limited
  - The cache controller provides flow control by limiting the number of outstanding requests
- Main memory flow control ..



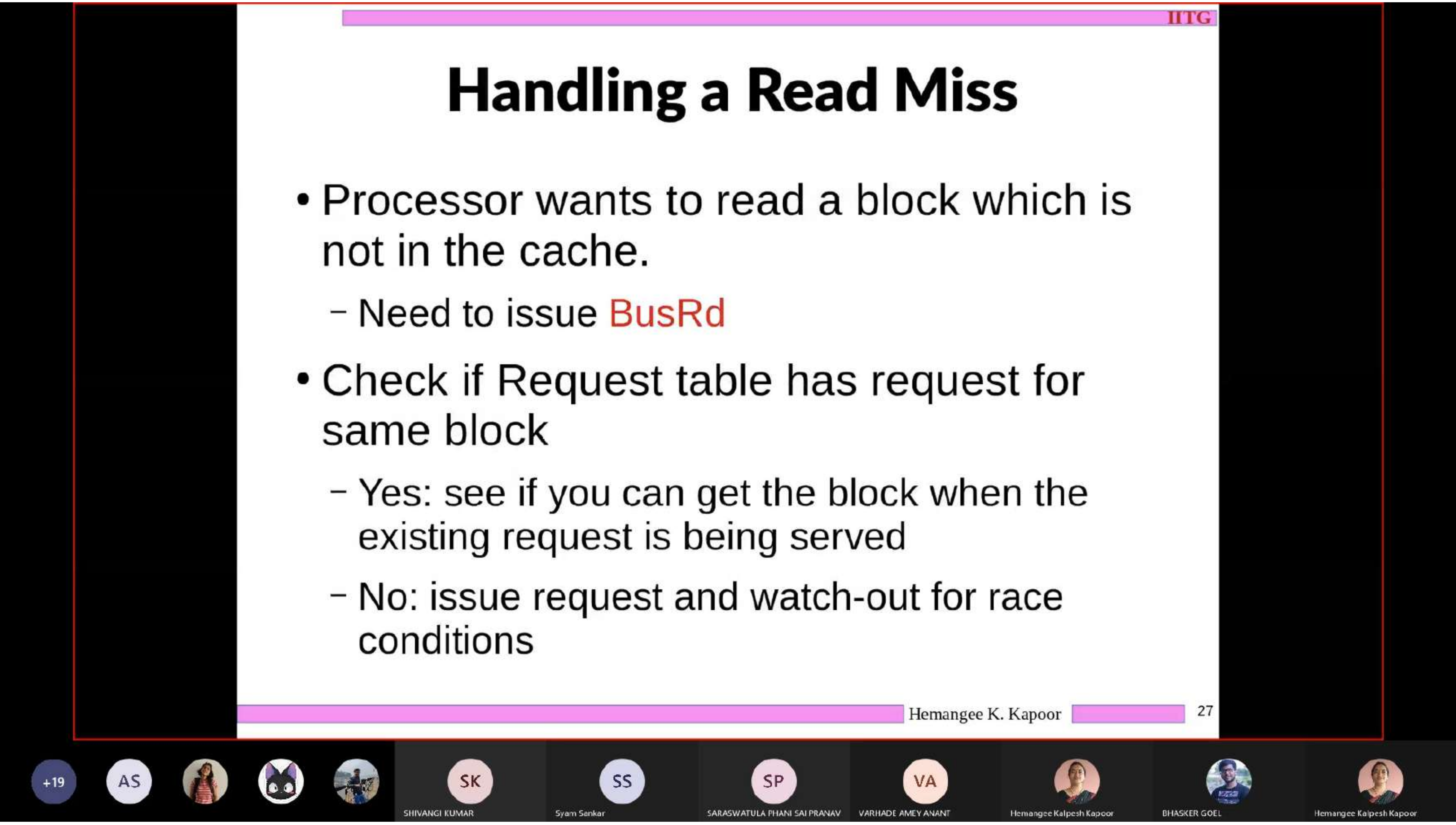
Flow control..

- Main memory flow control
  - 8 outstanding bus transactions are allowed
  - If each generates a write-back, then memory should have enough space
  - Wr-back happen in quick succession as no response phase
- SGI Challenge: both data and addr-bus has NACK line
  - If buffers are full the memory can raise NACK before the request/response transaction reaches the ACK-cycle
  - If NACK=1 then the transaction is cancelled and later retried
  - Back-off and priorities are used to reduce traffic and starvation
- SUN Enterprise: destination (in our case memory) initiates retry when it has free buffer space
  - Source keeps watch for this retry
  - Guaranteed space will still be there, so need only one retry



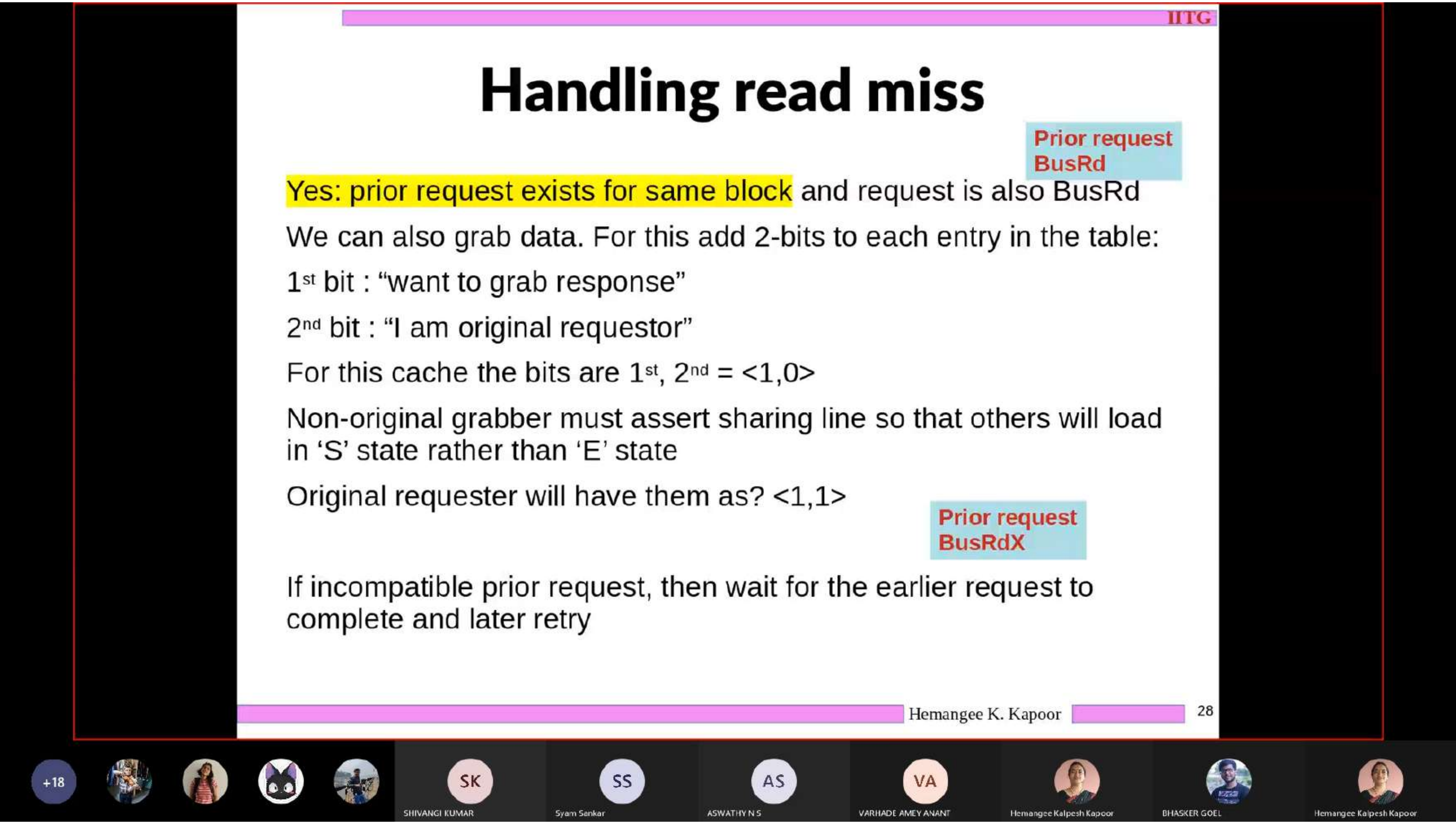
Path of a Cache Miss

(1) Handling a Read Miss
(2) Handling a Write Miss



Handling a Read Miss

- Processor wants to read a block which is not in the cache.
  - Need to issue BusRd
- Check if Request table has request for same block
  - Yes: see if you can get the block when the existing request is being served
  - No: issue request and watch-out for race conditions



Handling read miss

Prior request BusRd

Yes: prior request exists for same block and request is also BusRd

We can also grab data. For this add 2-bits to each entry in the table:

1st bit : "want to grab response"

2nd bit : "I am original requestor"

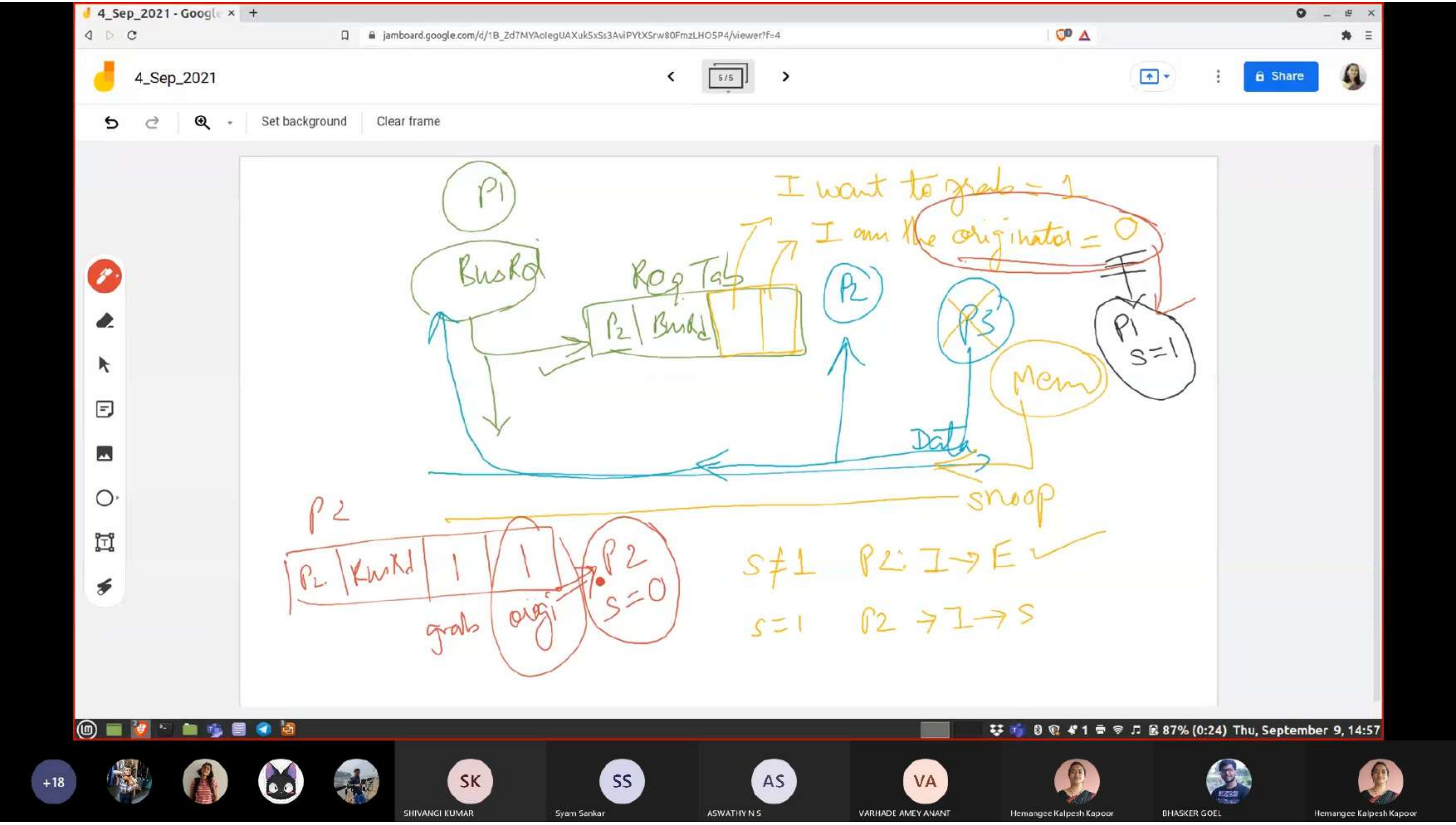
For this cache the bits are 1st, 2nd = <1,0>

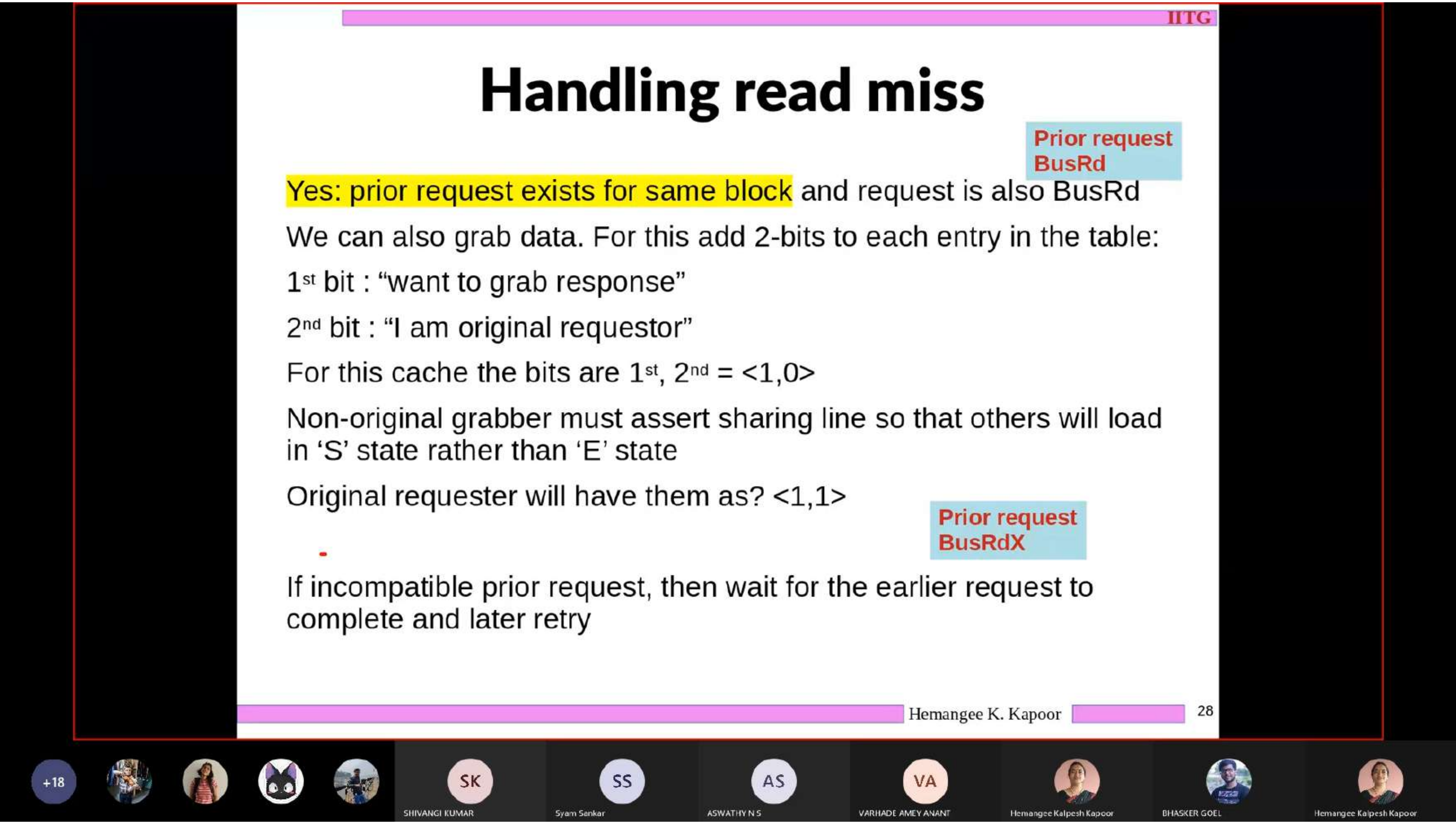
Non-original grabber must assert sharing line so that others will load in 'S' state rather than 'E' state

Original requester will have them as? <1,1>

Prior request BusRdX

If incompatible prior request, then wait for the earlier request to complete and later retry


4_Sep_2021 - Google
jamboard.google.com/d/1B_Zd7MYAoIegUAXuk5xSs3AviPYtXSrw80FmzLHOSP4/viewer?f=4
4_Sep_2021
5 / 5
Share
Set background
Clear frame
P1
BusRd
Req Tab
P2 | BusRd
P2
P3
Mem
P1
S=1
Data
snoop
I want to grab = 1
I am the originator = 0
P2
P2 | BusRd | 1 | 1
grab
origi
P2
S=0
S≠1 P2: I→E
S=1 P2→I→S
87% (0:24) Thu, September 9, 14:57
+18
SK
SHIVANGI KUMAR
SS
Syam Sankar
AS
ASWATHY N S
VA
VARHADE AMEY ANANT
Hemangee Kalpesh Kapoor
BHASKER GOEL
Hemangee Kalpesh Kapoor



Handling read miss

Prior request BusRd

Yes: prior request exists for same block and request is also BusRd

We can also grab data. For this add 2-bits to each entry in the table:

1st bit : "want to grab response"

2nd bit : "I am original requestor"

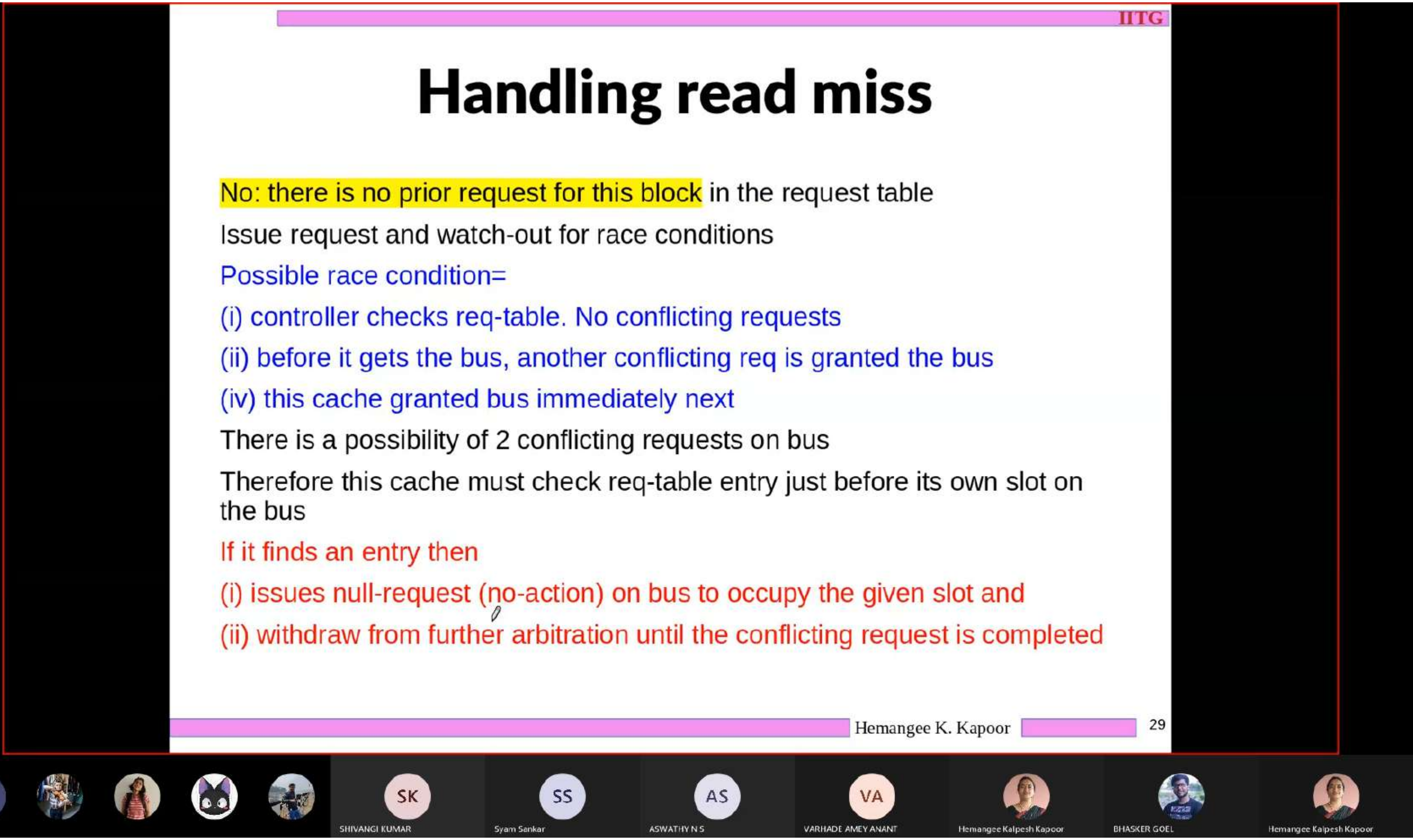
For this cache the bits are 1st, 2nd = <1,0>

Non-original grabber must assert sharing line so that others will load in 'S' state rather than 'E' state

Original requester will have them as? <1,1>

Prior request BusRdX

If incompatible prior request, then wait for the earlier request to complete and later retry



Handling read miss

No: there is no prior request for this block in the request table

Issue request and watch-out for race conditions

Possible race condition=

(i) controller checks req-table. No conflicting requests

(ii) before it gets the bus, another conflicting req is granted the bus

(iv) this cache granted bus immediately next

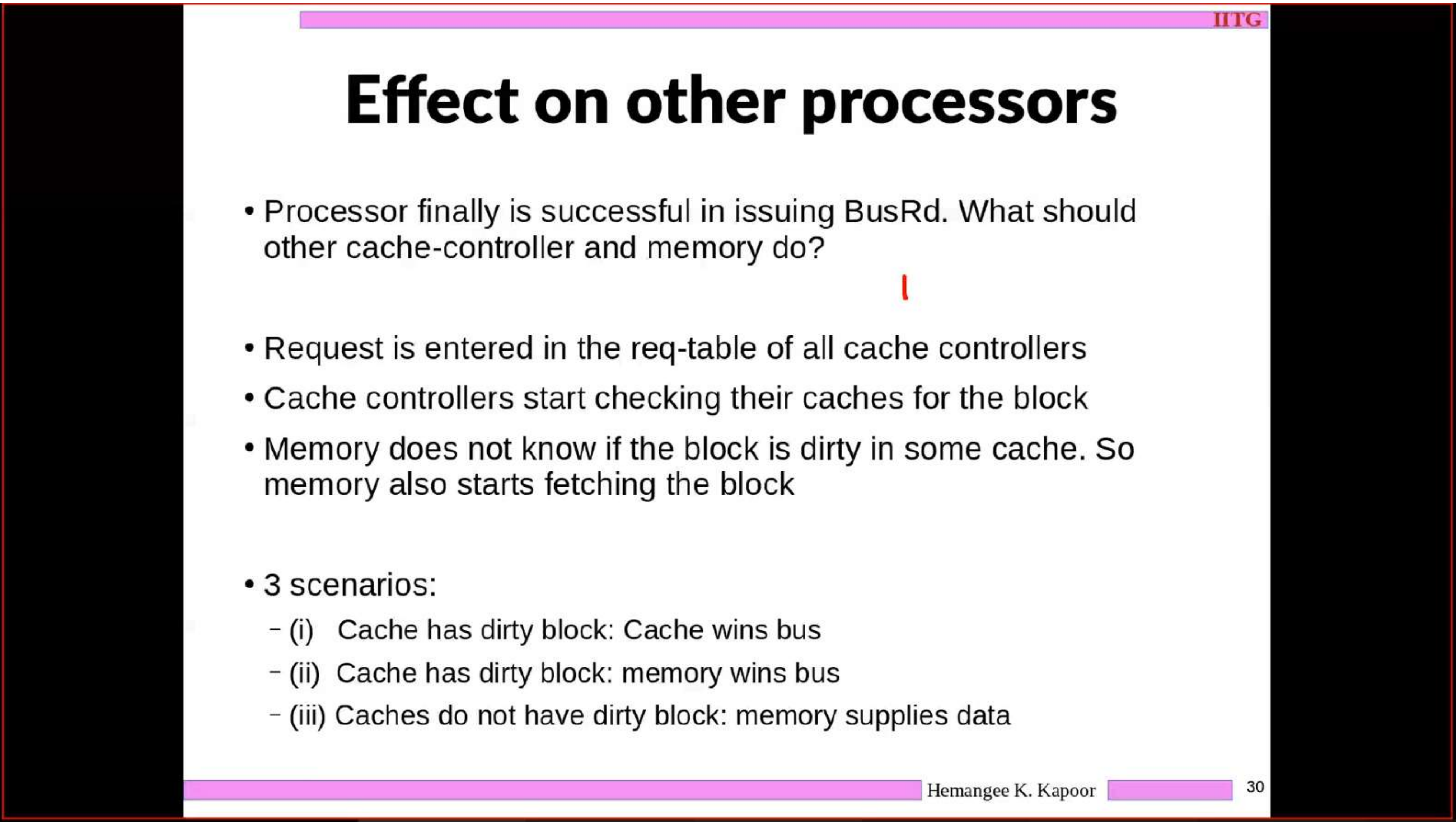
There is a possibility of 2 conflicting requests on bus

Therefore this cache must check req-table entry just before its own slot on the bus

If it finds an entry then

(i) issues null-request (no-action) on bus to occupy the given slot and

(ii) withdraw from further arbitration until the conflicting request is completed



Effect on other processors

- Processor finally is successful in issuing BusRd. What should other cache-controller and memory do?

- Request is entered in the req-table of all cache controllers
- Cache controllers start checking their caches for the block
- Memory does not know if the block is dirty in some cache. So memory also starts fetching the block

- 3 scenarios:
  - (i) Cache has dirty block: Cache wins bus
  - (ii) Cache has dirty block: memory wins bus
  - (iii) Caches do not have dirty block: memory supplies data