HetPipe: Enabling Large DNN Training on (Whimpy) Heterogeneous GPU Clusters through Integration of Pipelined Model Parallelism and Data Parallelism

USENIX ATC 2020

Jay H. Park, Gyeongchan Yun, Chang M. Yi, Nguyen T. Nguyen, Seungmin Lee, Jaesik Choi[†], Sam H. Noh, and Young-ri Choi

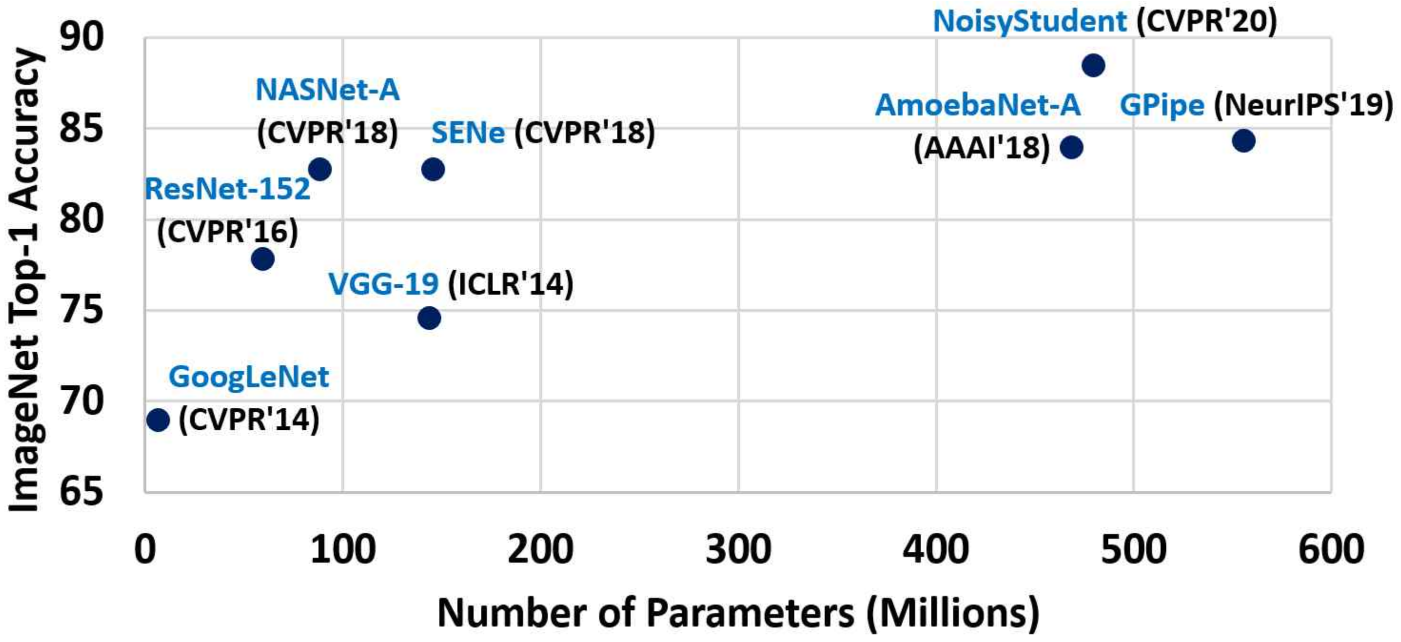
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

[†] KAIST

Contents

Motivation

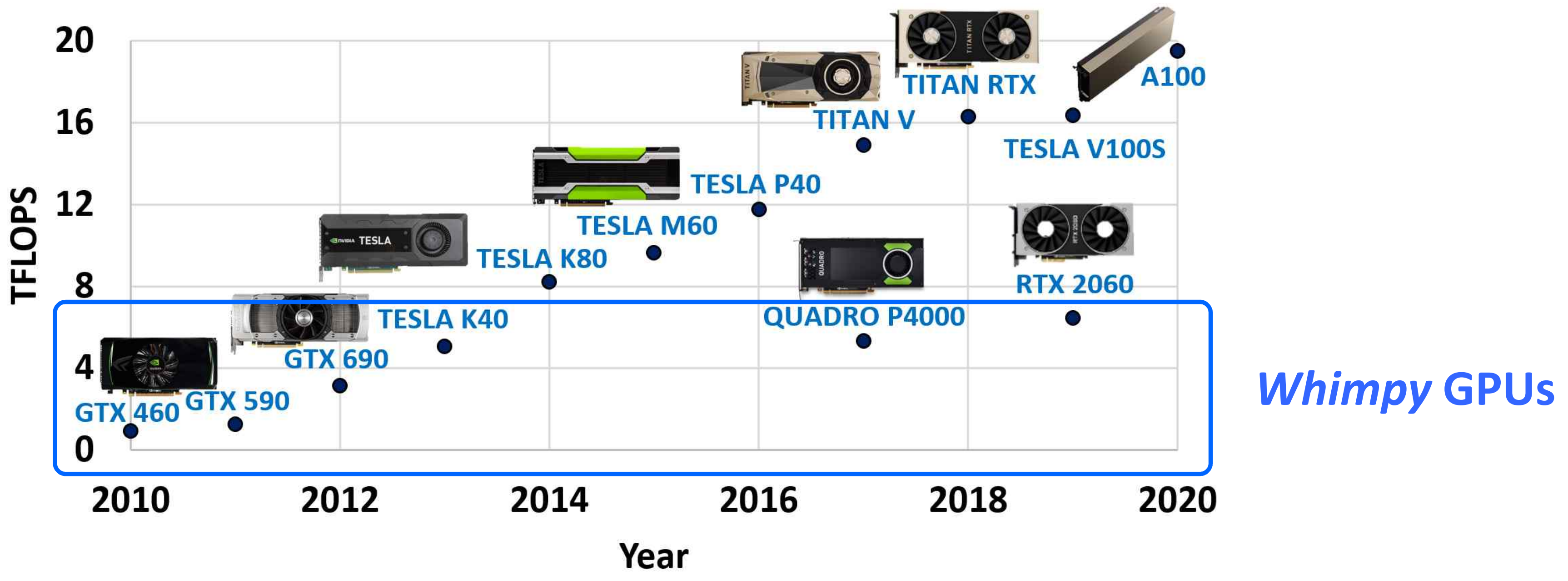
- DNN (Deep Neural Network) models continue to grow



- Need more powerful GPUs for training!

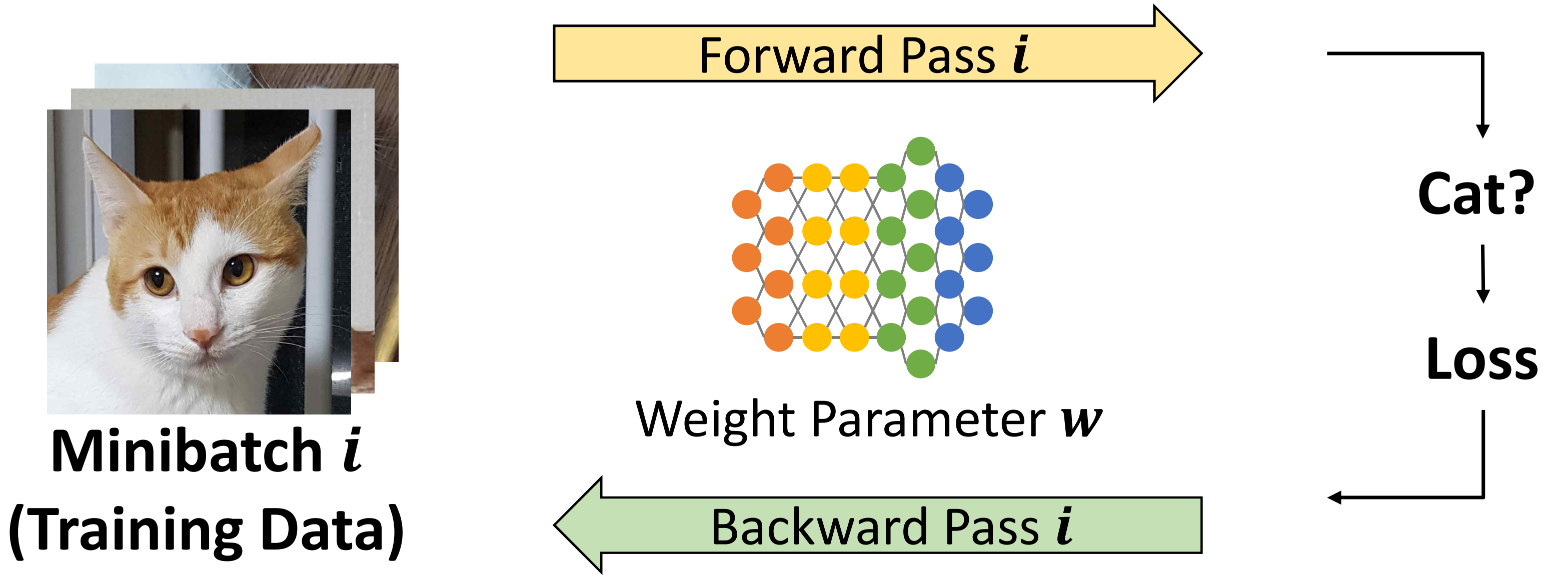
Motivation

- Short release cycle of new GPU architectures



- Use of heterogeneous GPUs is inevitable!
- What to do with *whimpy* GPUs?

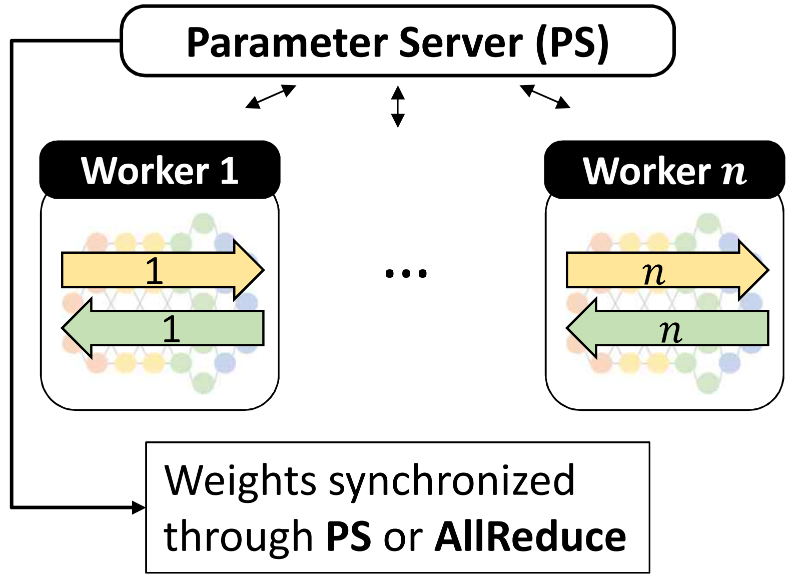
DNN Training



$$w_{i+1} = w_i - \eta \cdot u_i$$

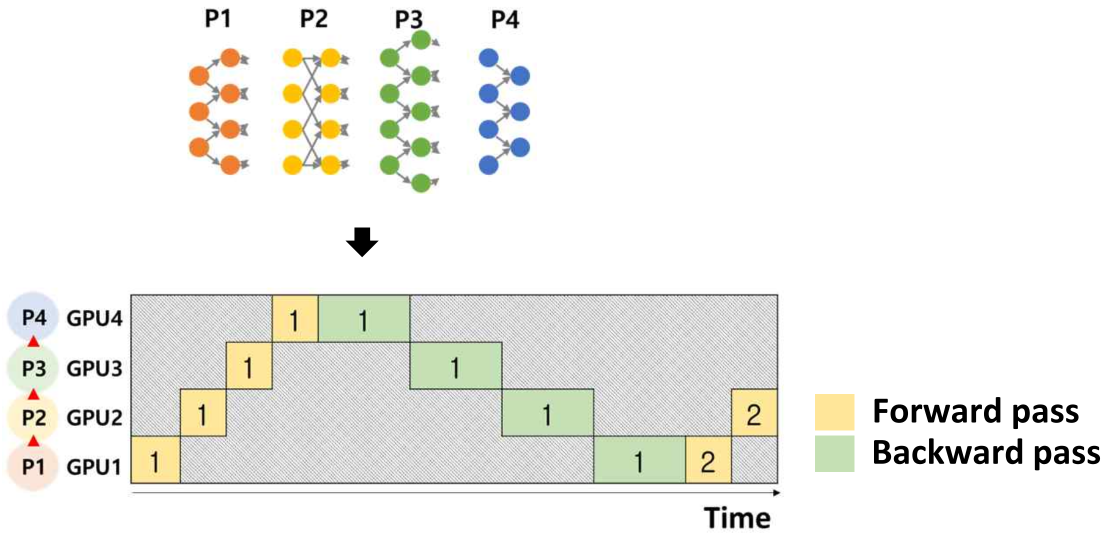
Parallelizing DNN Training

- Data parallelism (DP)



- GPU memory limitation

- Model parallelism (MP)

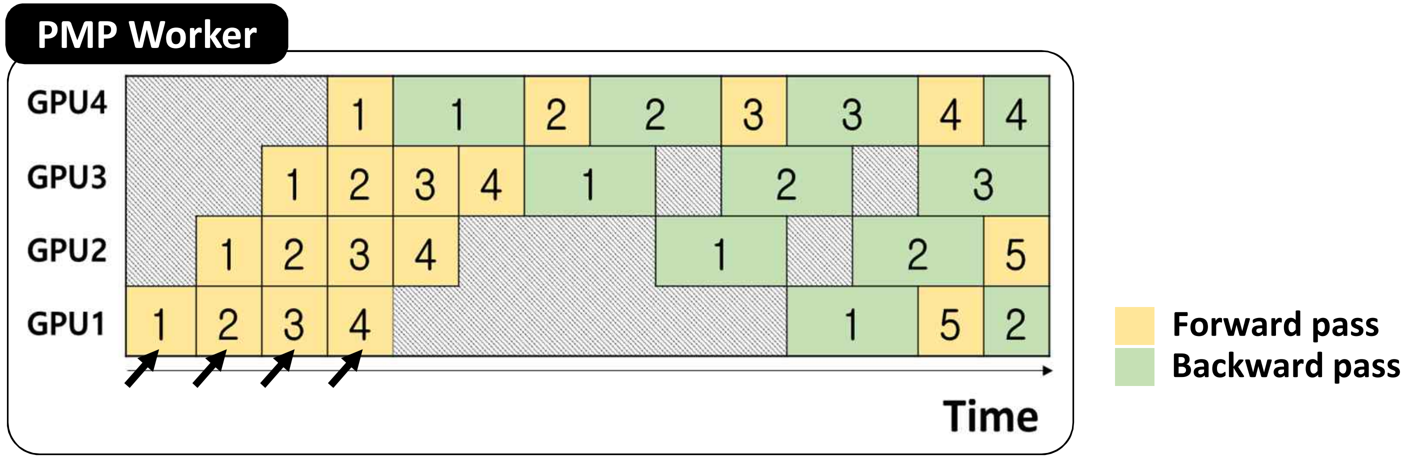


- Low GPU utilization

Parallelizing DNN Training

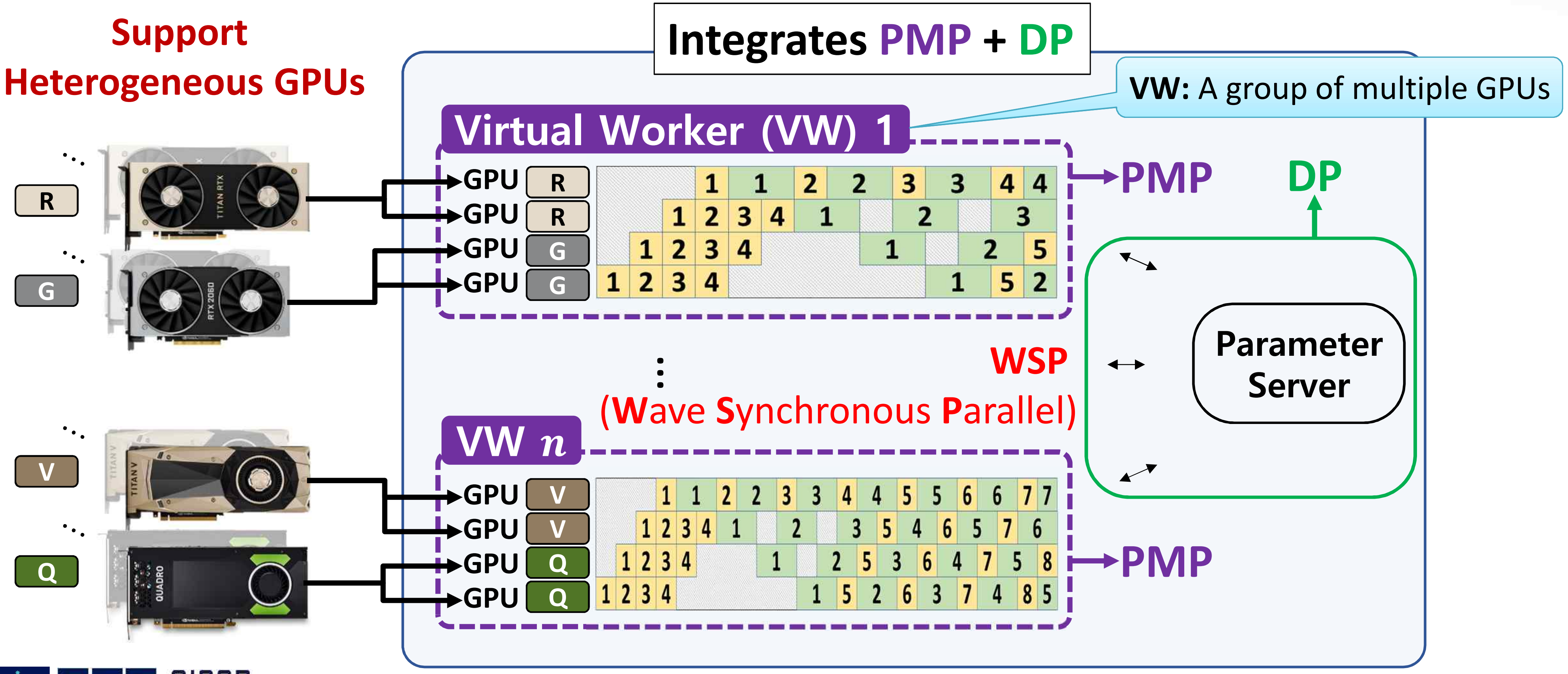
- **Attempts to improve MP utilization**
  - Pipelined model parallelism (PMP)

- **PipeDream** [SOSP'19]
- **GPipe** [NIPS'19]

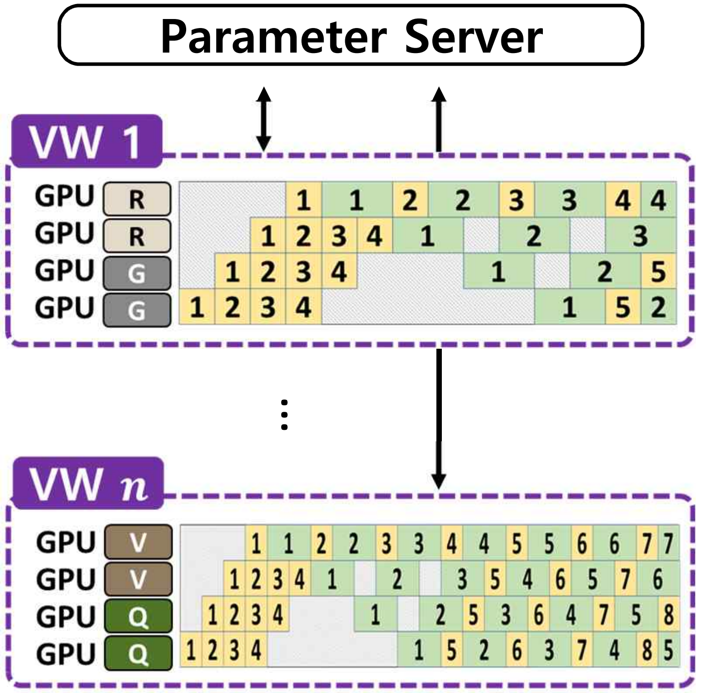


- Designed for homogeneous GPUs
- Designed for a single PMP worker

HetPipe in a Nutshell



Challenges in integration PMP+DP in Heterogeneous GPUs



- What weight version should be used by each VW to synchronize with other VWs?

- How do we reduce virtual worker stragglers when we consider DP?

⋮

Many more in the paper

HetPipe Contributions

Enable Large DNN Training on Heterogeneous GPUs

Aggregate heterogeneous resources

Reduce the straggler problem

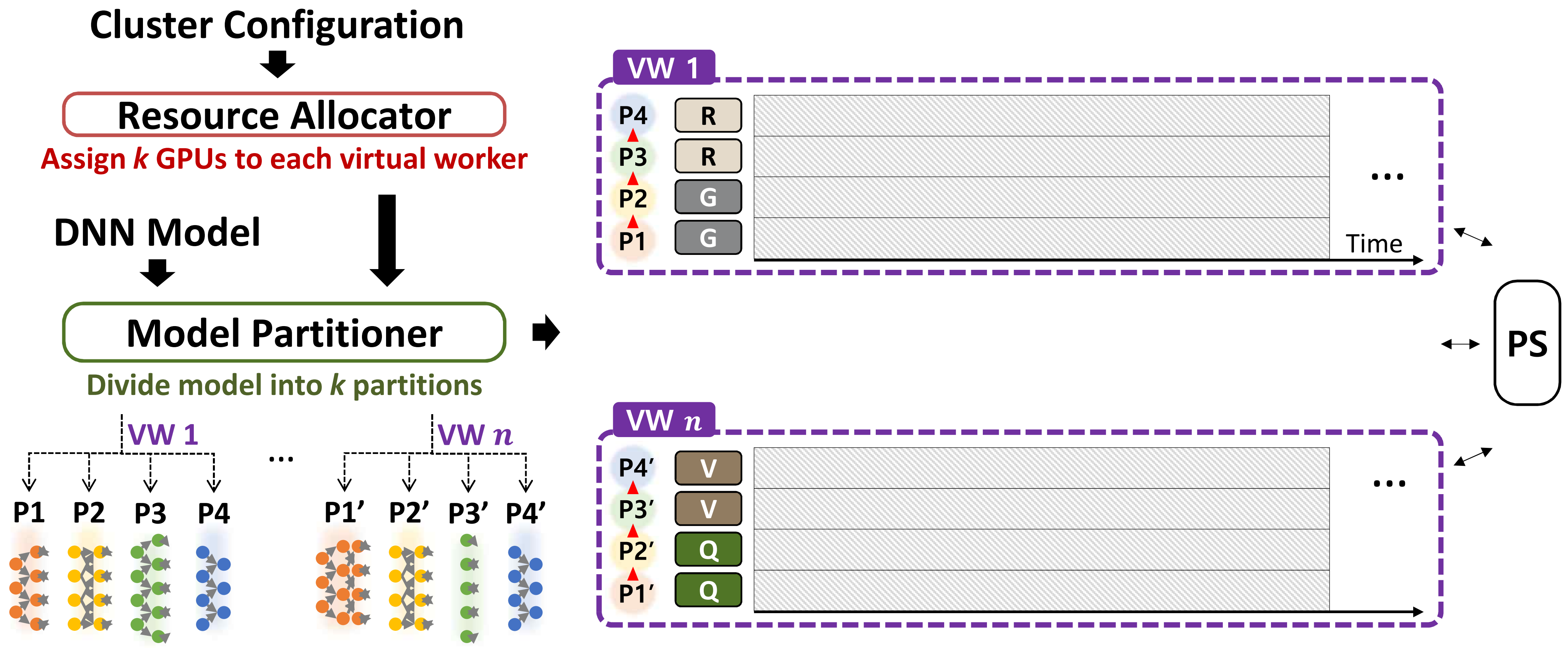
Integrates PMP + DP

Novel parameter synchronization model

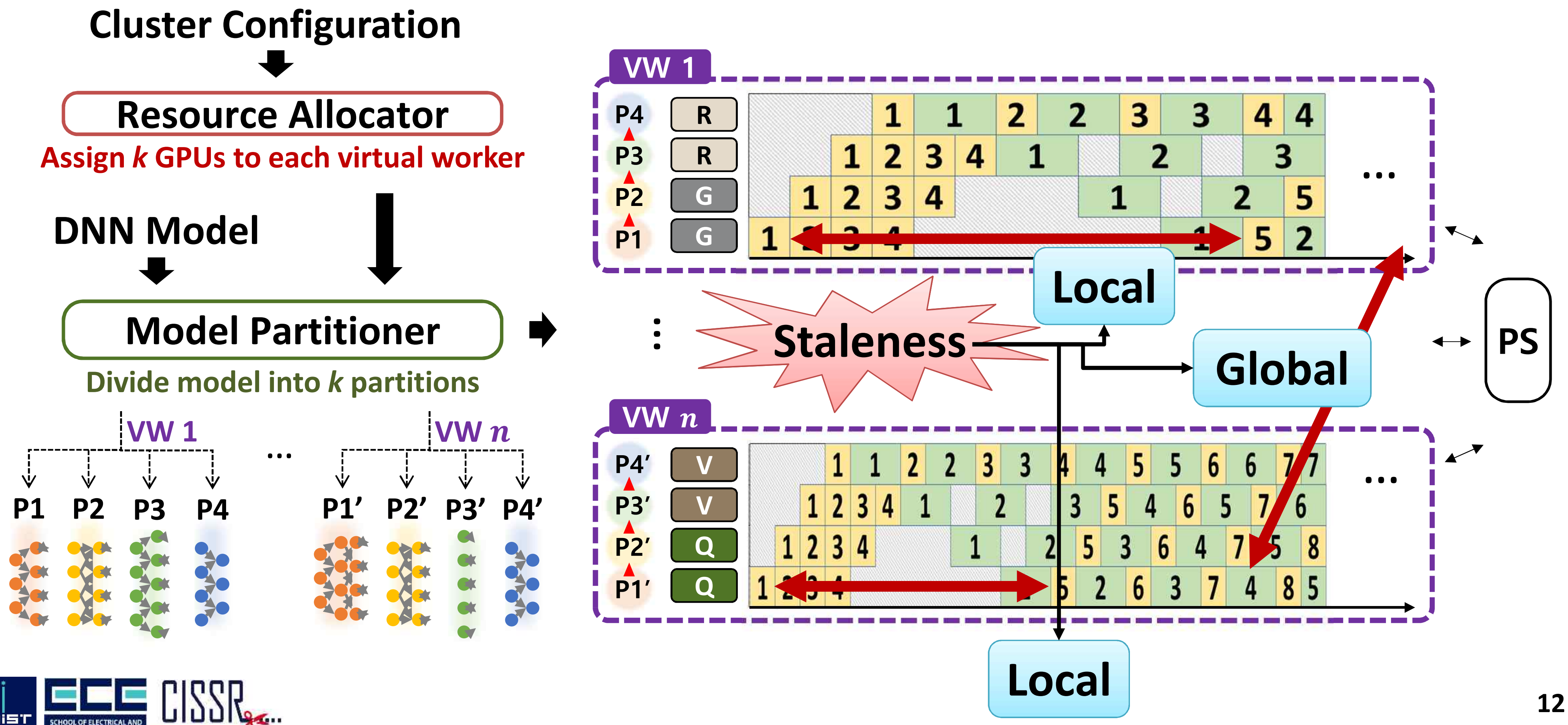
WSP (Wave Synchronous Parallel)

Proof of WSP Convergence

HetPipe Workflow



HetPipe Workflow

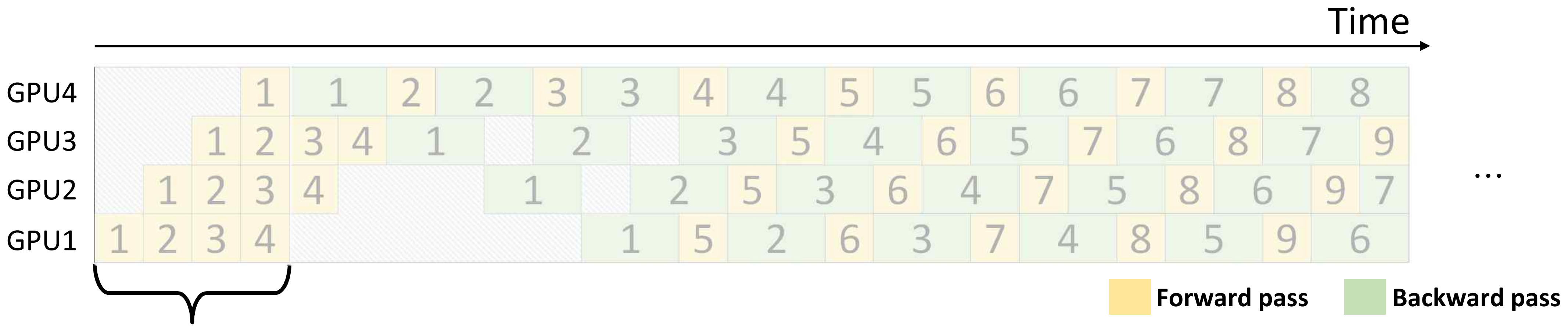


Outline

- Motivation & Background
- HetPipe in a Nutshell
- **Our System: HetPipe**
  - **Pipelined Model Parallelism Within a VW**
  - Data Parallelism with Multiple VWs
- Evaluation
- Conclusion

Pipelined Model Parallelism Within a VW

- **Execution of a virtual worker**



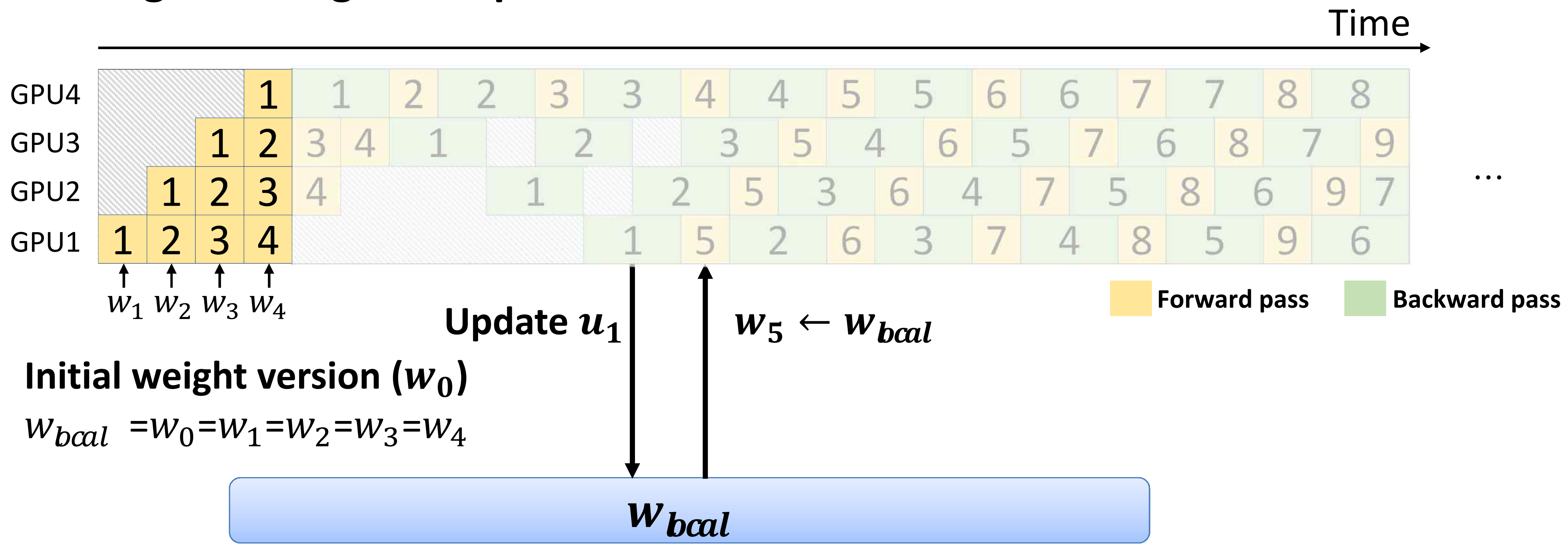
N_m minibatches processed concurrently in pipeline manner

w_{bcal}

W_{bcal} is a consistent version of weights within a VW

Pipelined Model Parallelism Within a VW

- **Weight management procedure**



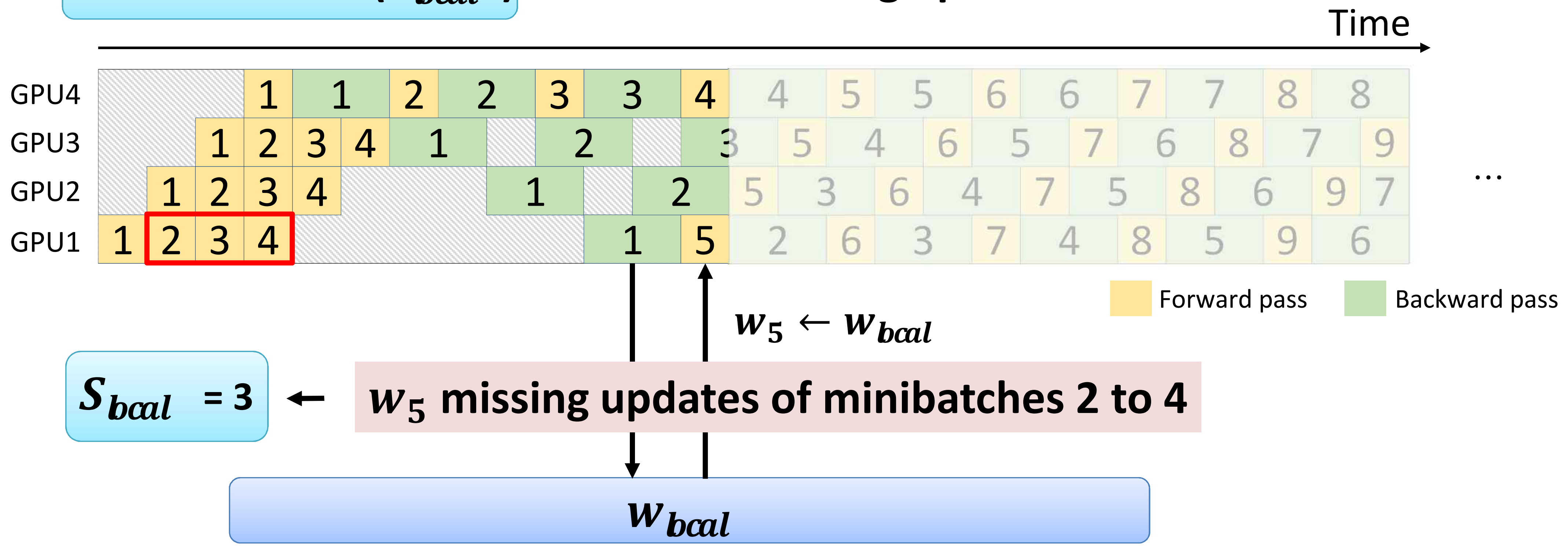
Initial weight version (w_0)

$w_{bcal} = w_0 = w_1 = w_2 = w_3 = w_4$

$$w_{bcal} \leftarrow w_{bcal} + u_1$$

Pipelined Model Parallelism Within a VW

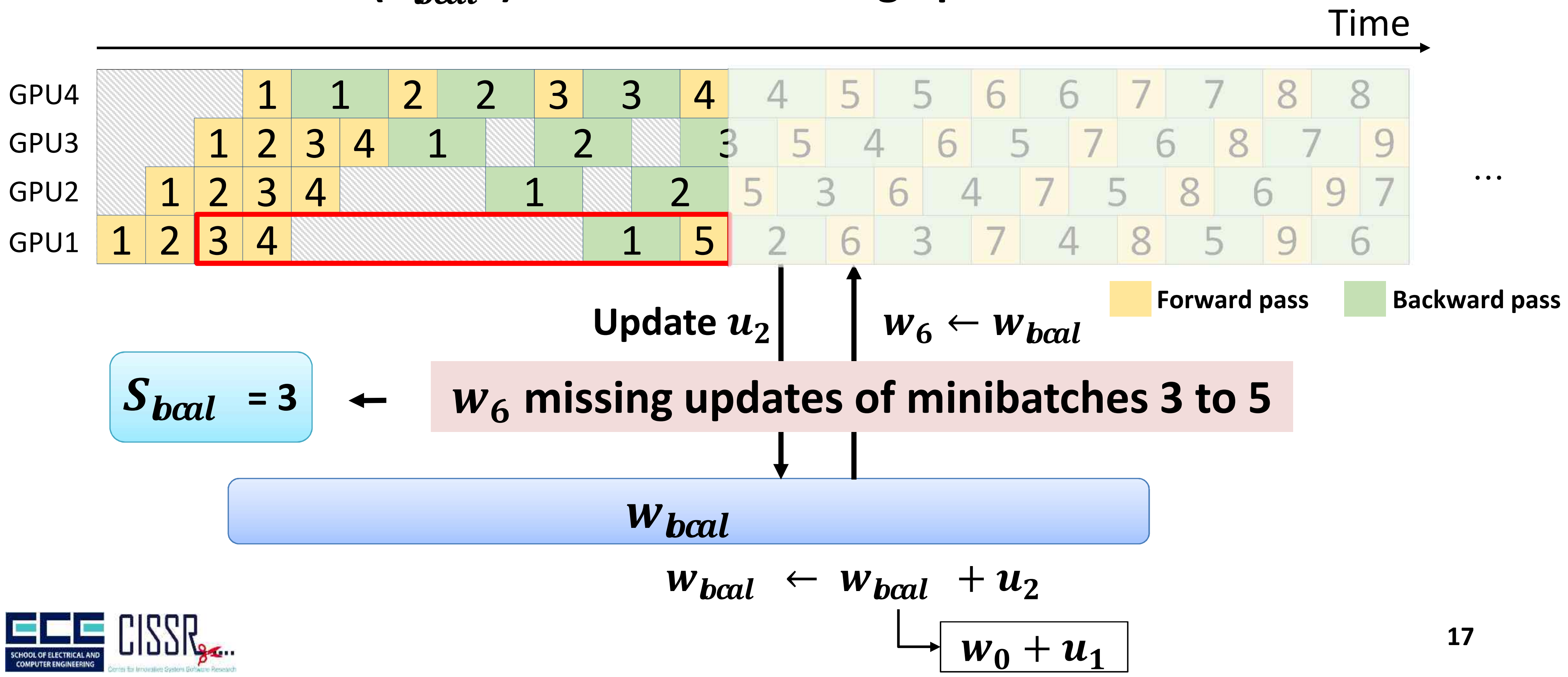
- Local staleness (s_{local}): maximum missing updates



$$w_{local} \leftarrow w_{local} + u_1$$

Pipelined Model Parallelism Within a VW

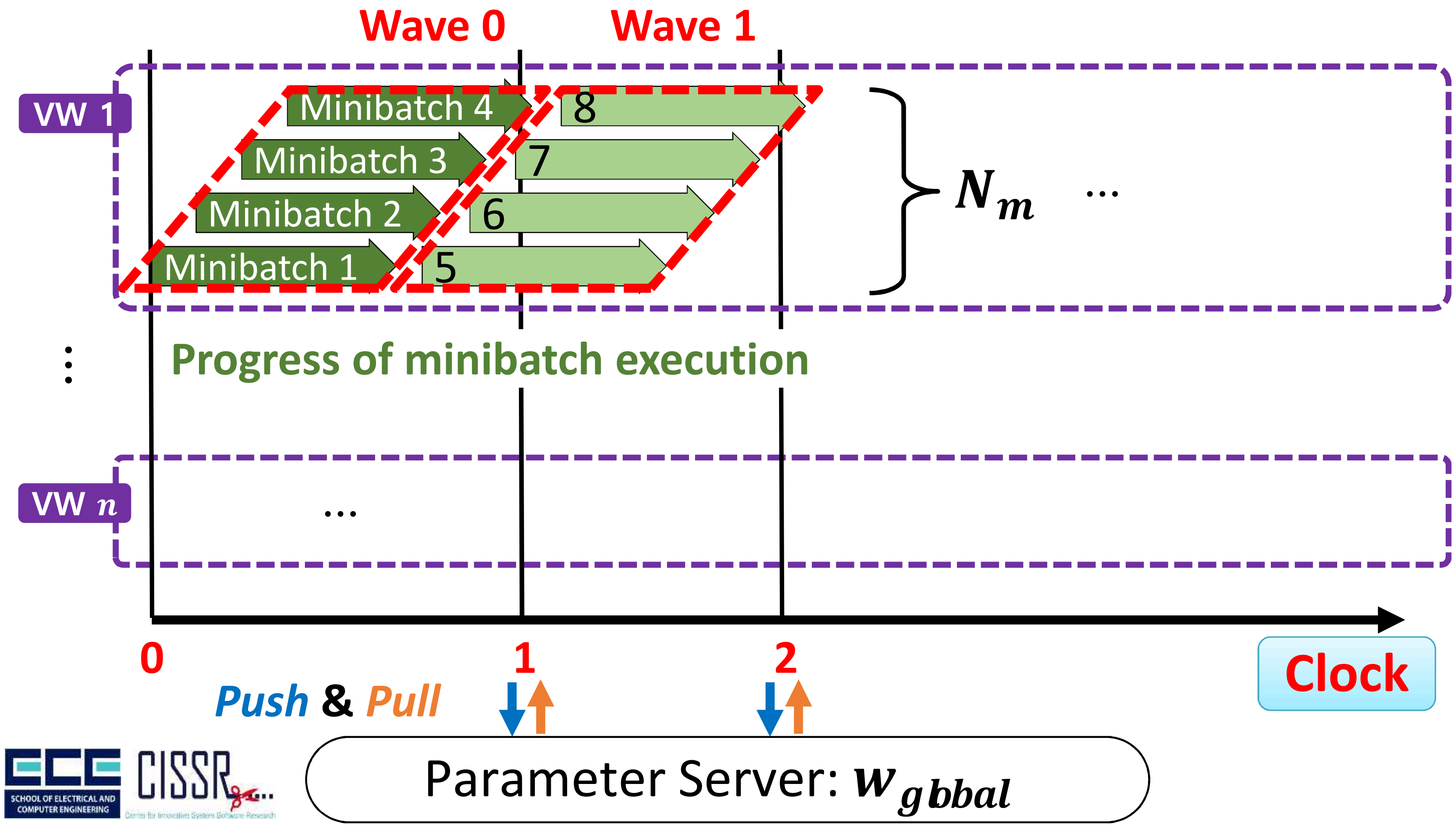
- **Local staleness (S_{local}): maximum missing updates**


Time
GPU4
GPU3
GPU2
GPU1
Forward pass
Backward pass
Update u_2
$w_6 \leftarrow w_{local}$
S_{local} = 3
w_6 missing updates of minibatches 3 to 5
w_{local}
$w_{local} \leftarrow w_{local} + u_2$
$w_0 + u_1$
…

Outline

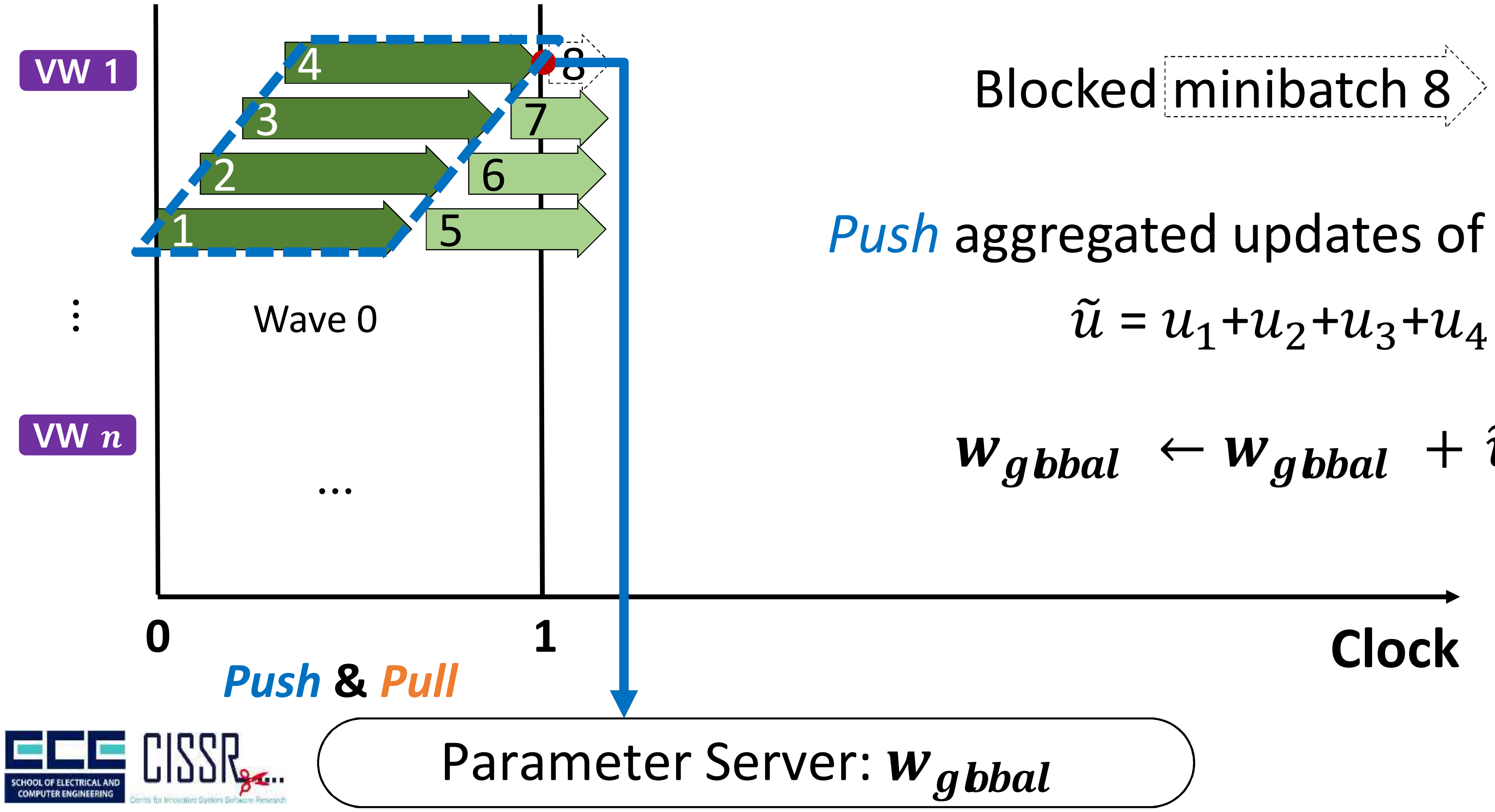
Data Parallelism with Multiple VWs

Wave: Sequence of concurrently executing N_m minibatches



Data Parallelism with Multiple VWs

- *Push* occurs every clock



Blocked minibatch 8

Push aggregated updates of wave0 ($\tilde{u}$)

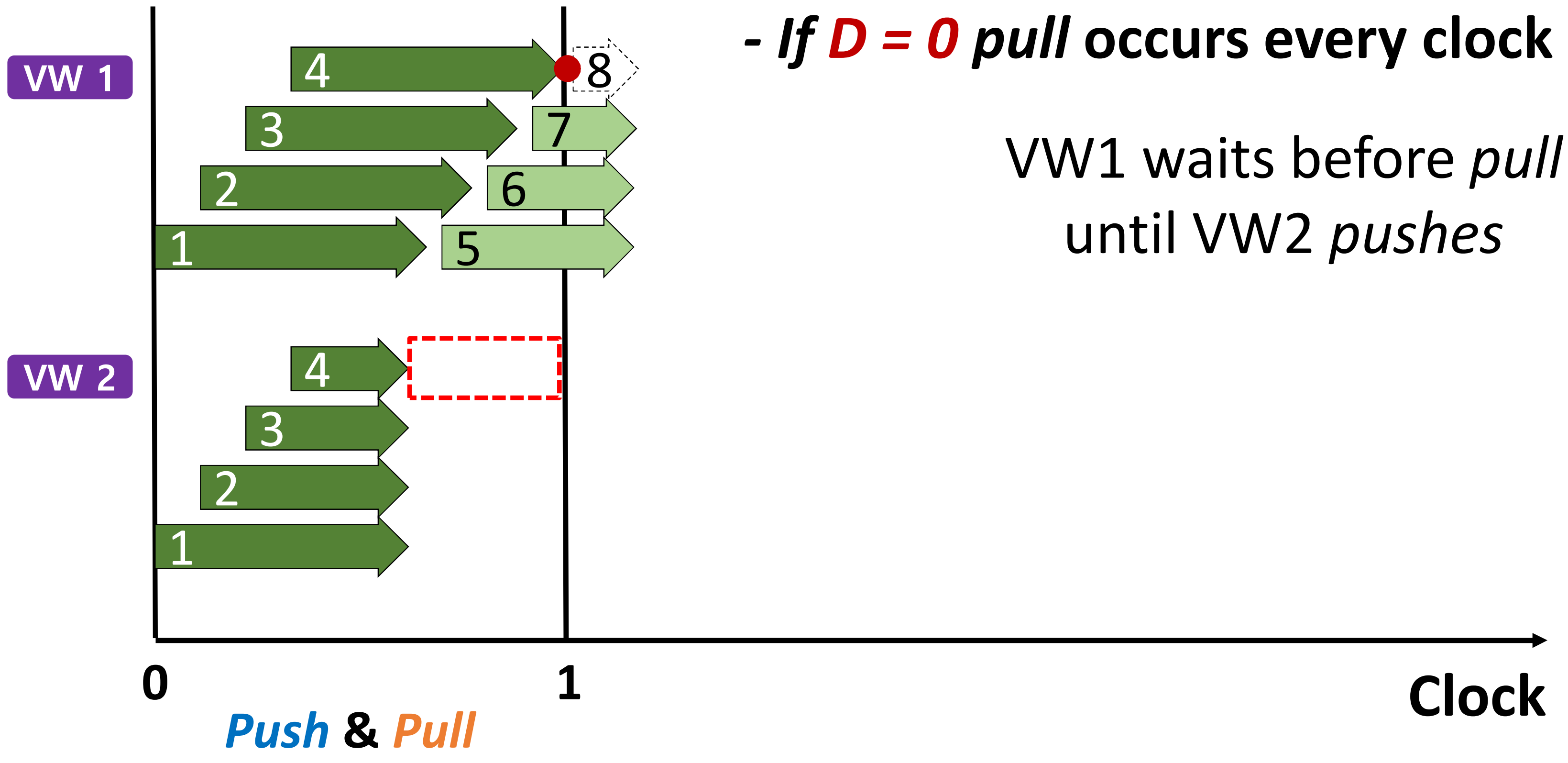
$$\tilde{u} = u_1 + u_2 + u_3 + u_4$$

$$\boldsymbol{w}_{gbbal} \leftarrow \boldsymbol{w}_{gbbal} + \tilde{\boldsymbol{u}}$$

Data Parallelism with Multiple VWs

- ***Pull*** **occurs intermittently - Depending on user defined *clock distance D***

  - *If D = 0 pull* occurs every clock

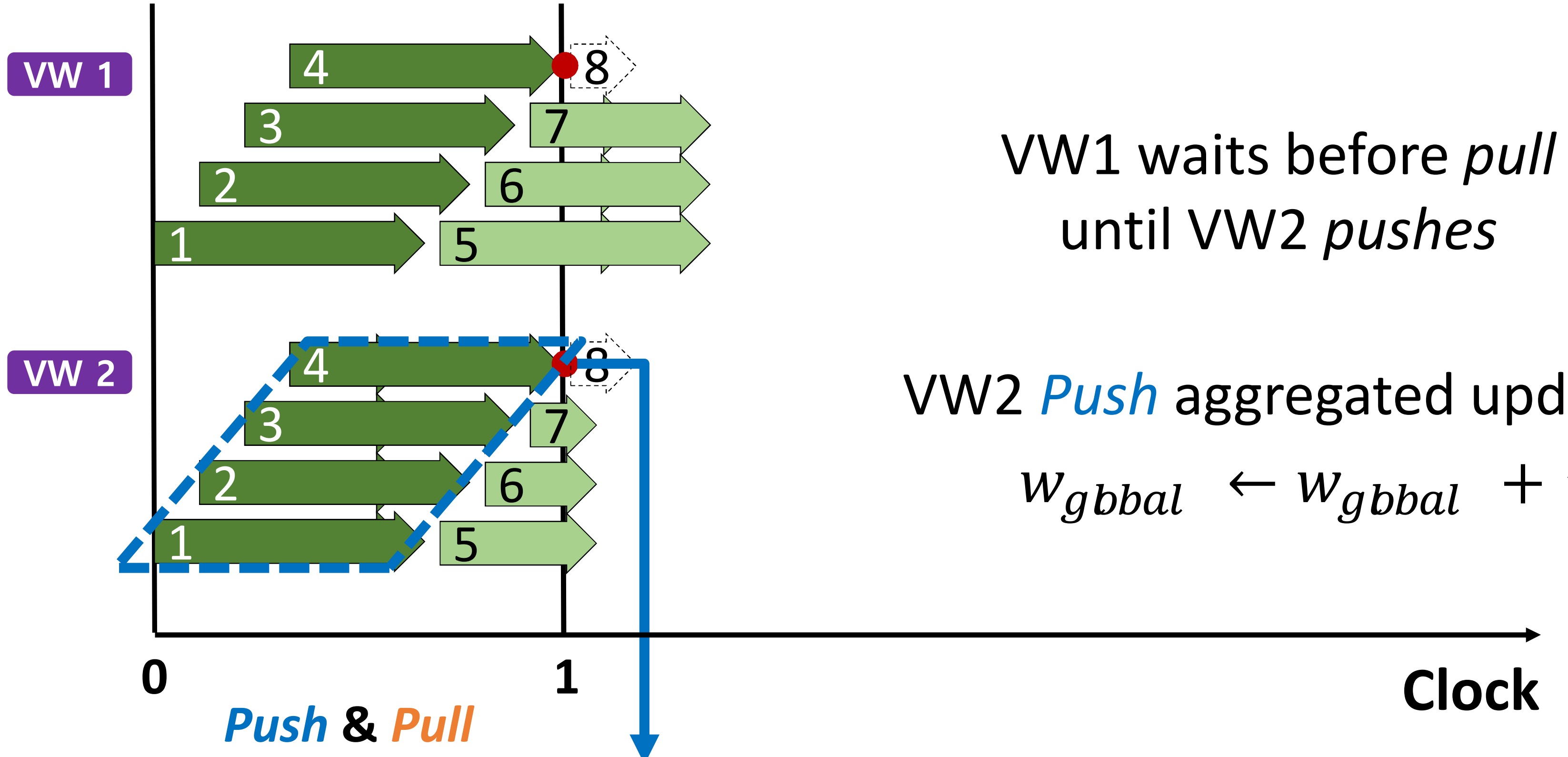


VW1 waits before *pull* until VW2 *pushes*

Parameter Server: $\boldsymbol{w}_{global}$

Data Parallelism with Multiple VWs

- *Pull* occurs intermittently - Depending on user defined *clock distance D*



If D = 0

VW1 waits before *pull* until VW2 *pushes*

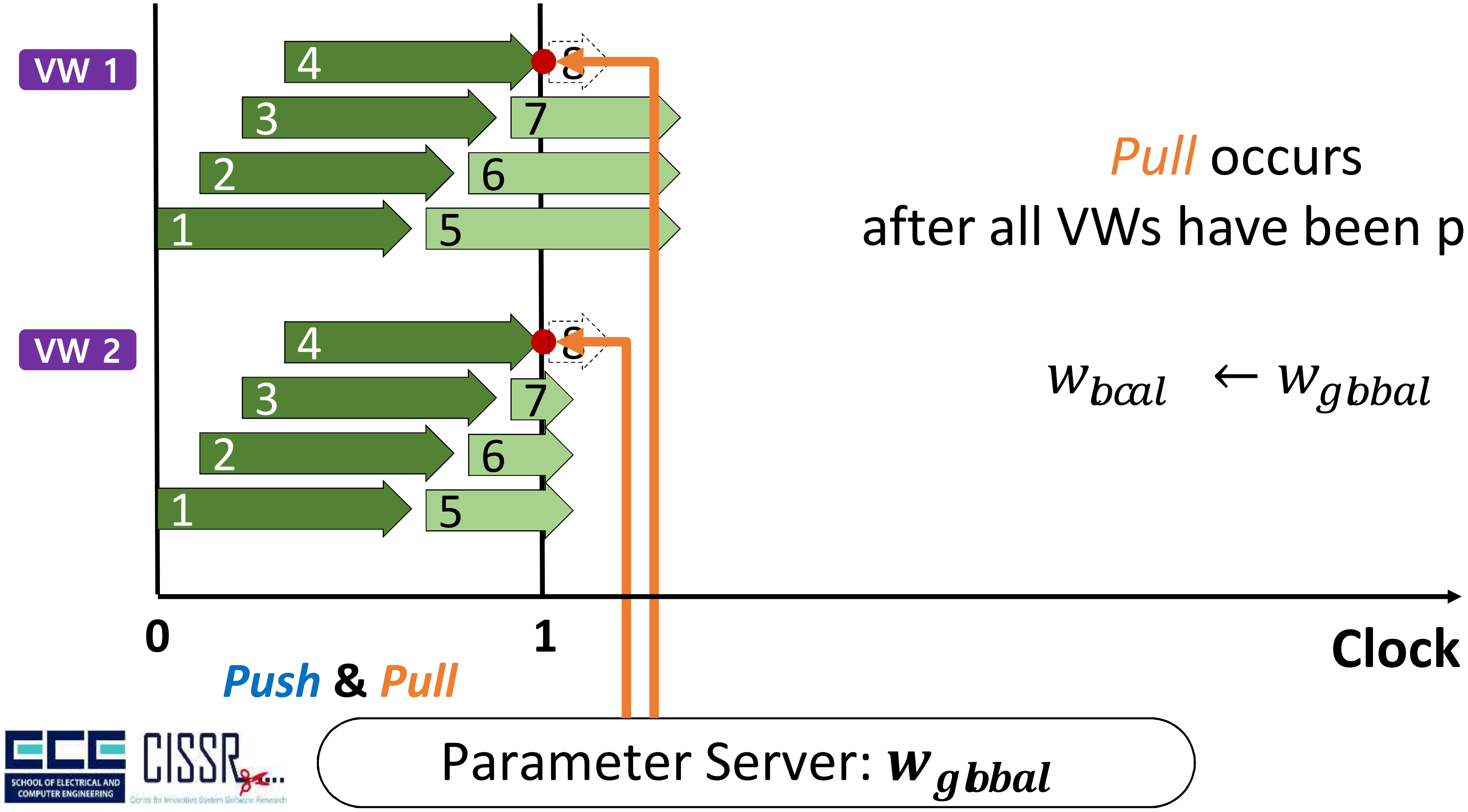
VW2 *Push* aggregated updates ($\tilde{u}$)

$$w_{global} \leftarrow w_{global} + \tilde{u}$$

Parameter Server: $\boldsymbol{w}_{\boldsymbol{global}}$

Data Parallelism with Multiple VWs

- *Pull* occurs intermittently - Depending on user defined *clock distance D*



Parameter Server: w_{global}

If D = 0

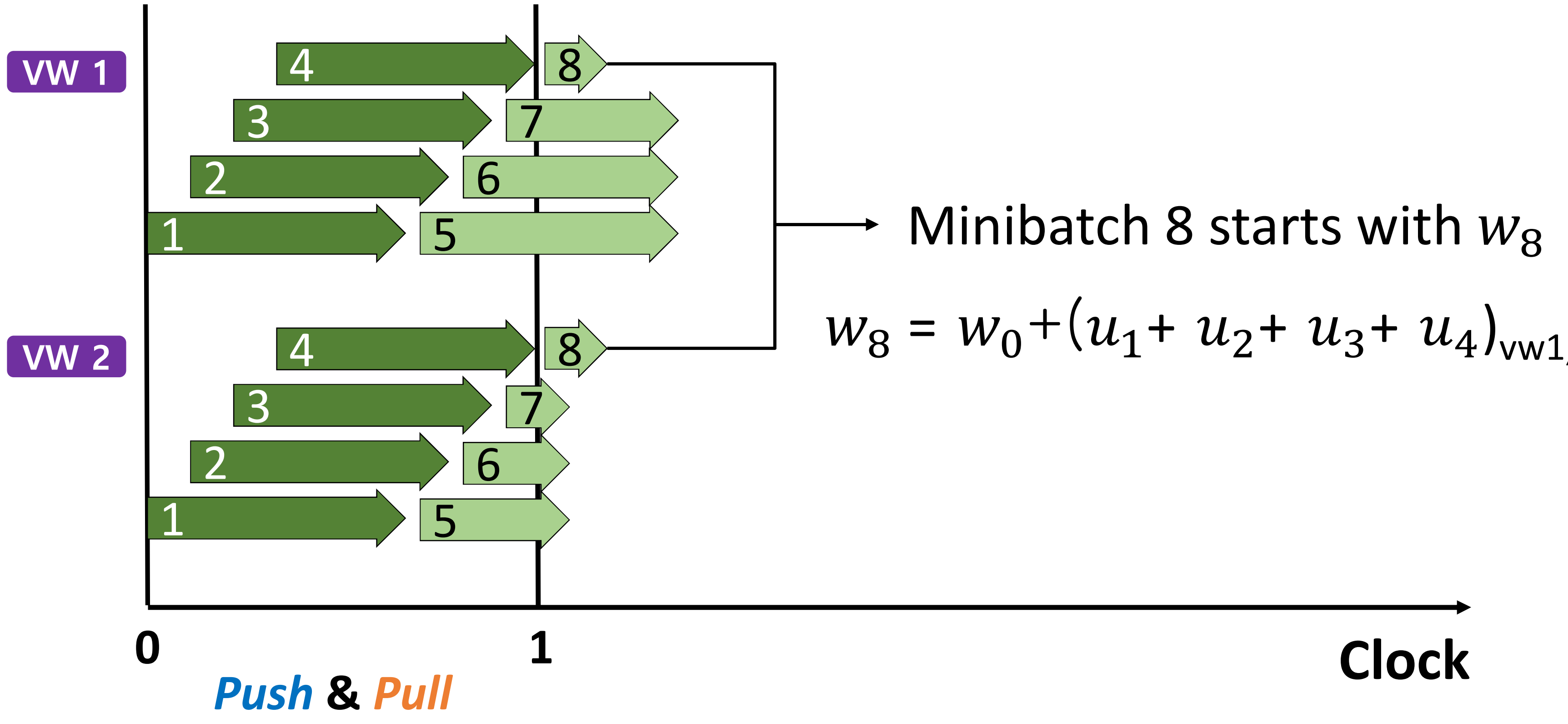
Pull occurs
after all VWs have been pushed

$w_{local} \leftarrow w_{global}$

Data Parallelism with Multiple VWs

- *Pull* occurs intermittently - Depending on user defined *clock distance D*

If D = 0



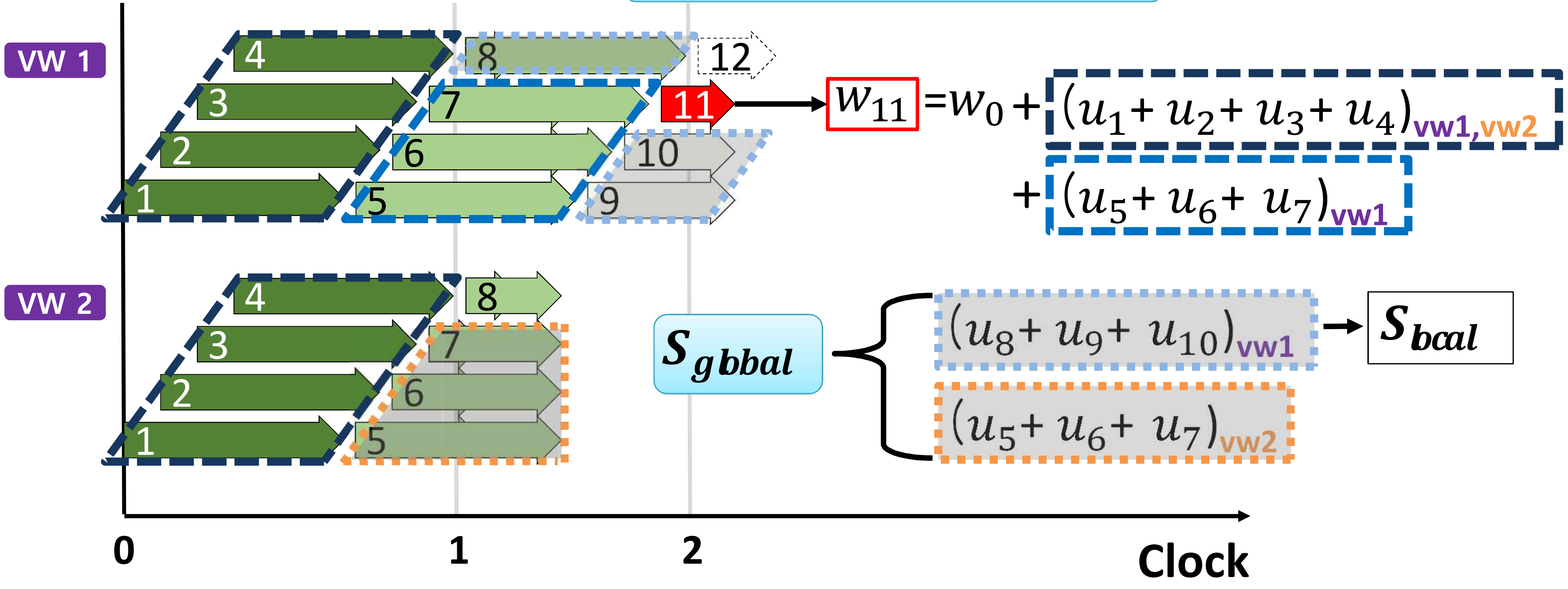
Minibatch 8 starts with w_8

$$w_8 = w_0 + (u_1 + u_2 + u_3 + u_4)_{vw1,vw2}$$

Parameter Server: $\boldsymbol{w}_{global}$

Data Parallelism with Multiple VWs

- Local staleness (S_{local}) and global staleness (S_{global}) with WSP



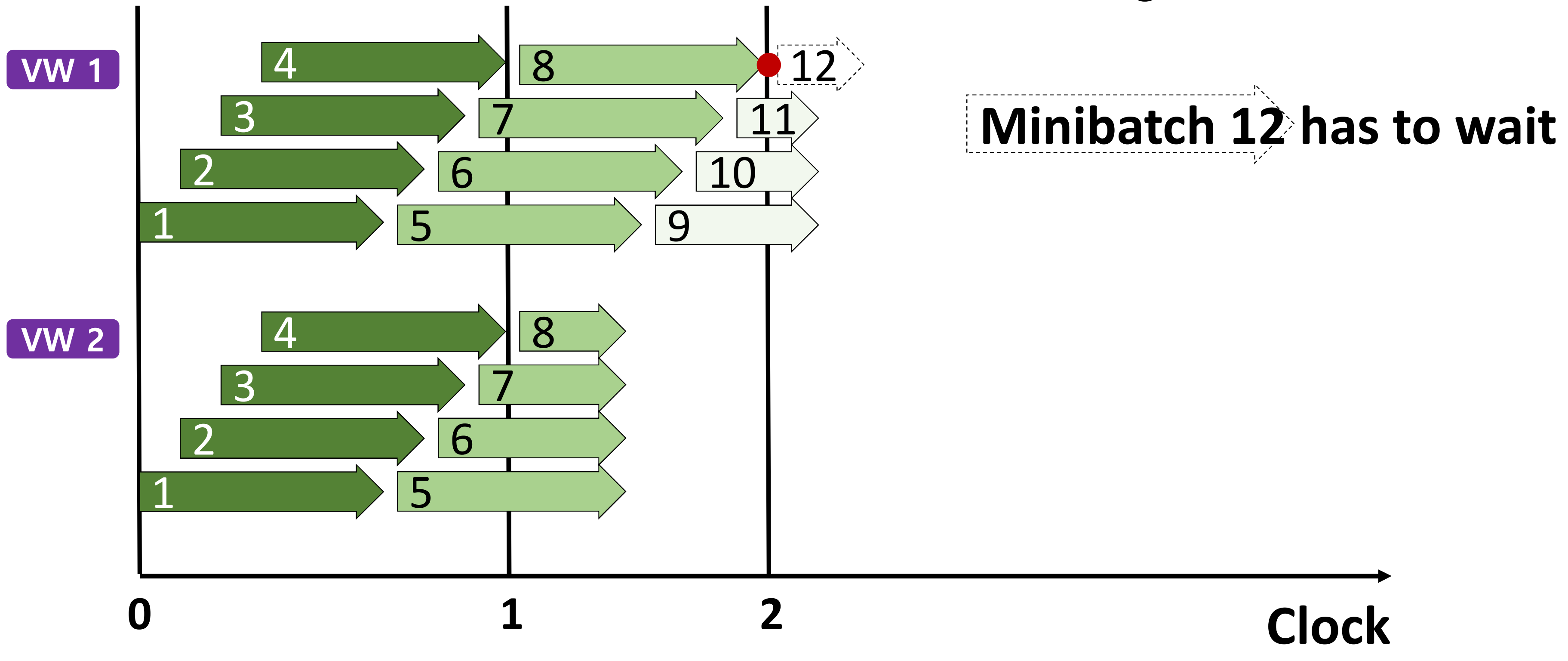
$$w_{11} = w_0 + (u_1 + u_2 + u_3 + u_4)_{vw1,vw2} + (u_5 + u_6 + u_7)_{vw1}$$

$$S_{global} \begin{cases} (u_8 + u_9 + u_{10})_{vw1} \rightarrow S_{local} \\ (u_5 + u_6 + u_7)_{vw2} \end{cases}$$

$$w_{global} = w_0 + (u_1 + u_2 + u_3 + u_4)_{vw1,vw2}$$

Data Parallelism with Multiple VWs

- **Local staleness (S_{local}) and global staleness (S_{global}) with WSP**



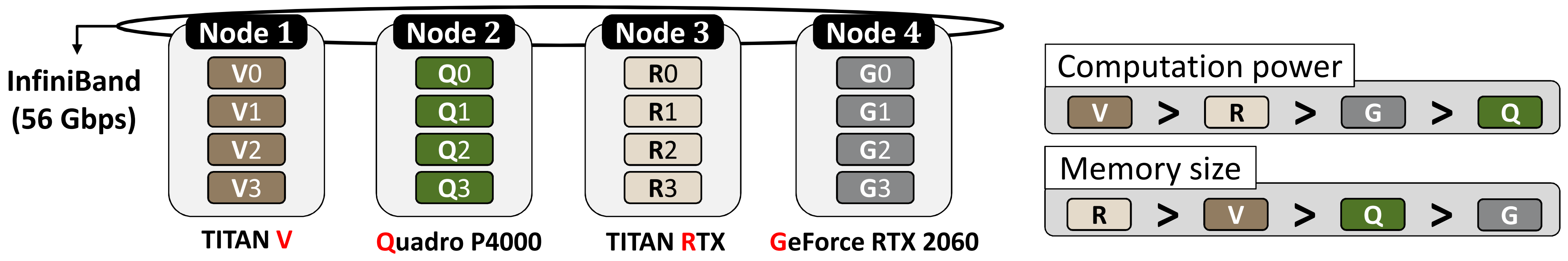
Parameter Server: w_{global}

Outline

- Motivation & Background
- HetPipe in a Nutshell
- Our System: HetPipe
- **Evaluation**
  - **Setup**
  - **Resource Allocation for Virtual Workers**
  - **Results**
- Conclusion

Evaluation Setup

- **Cluster setup - 4 heterogeneous GPU nodes**

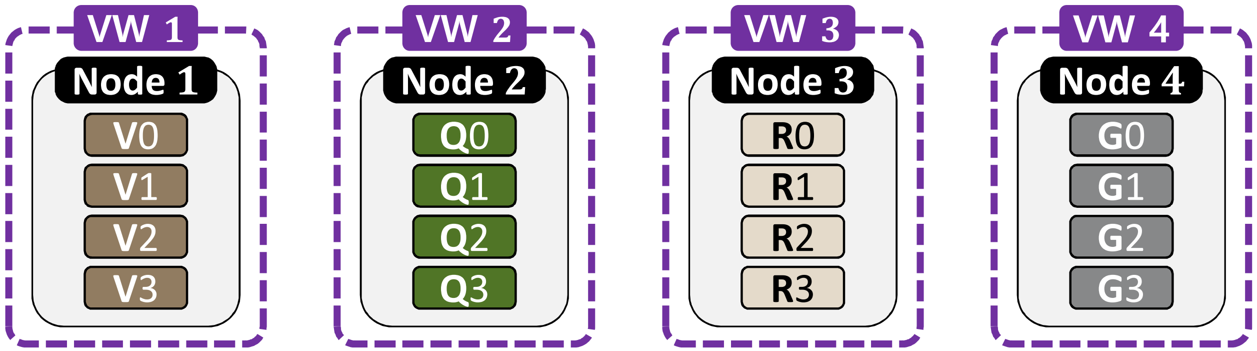


- **Two DNN models**

	ResNet-152	**VGG-19**
Dataset, minibatch size	ImageNet, 32	
Model parameter size	230 MB	548 MB
Characteristic	Large activation output	Large parameter size

Resource Allocation for Virtual Workers: NP, ED, HD

- NP (Node Partition)

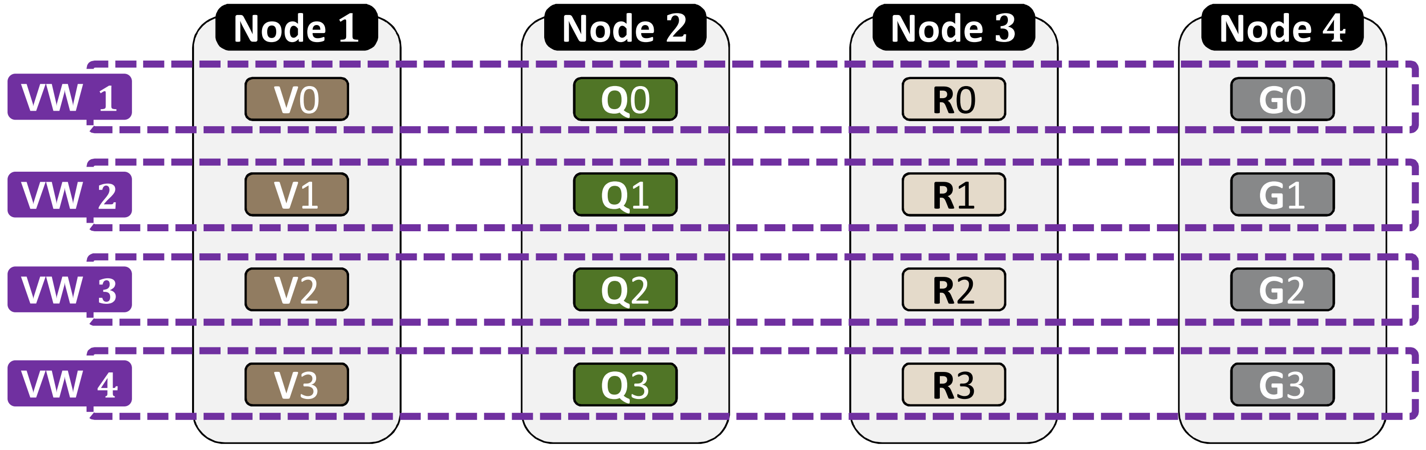


- Minimum communication overhead within VW

- Performance of each virtual worker varies
- Straggler may degrade performance with DP

Resource Allocation for Virtual Workers: NP, ED, HD

- ED (Equal Distribution)

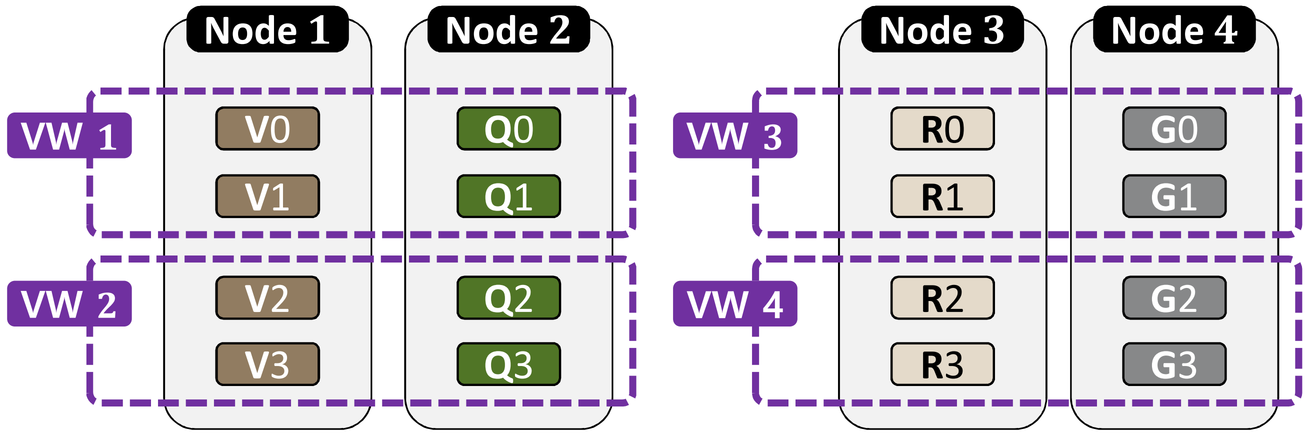


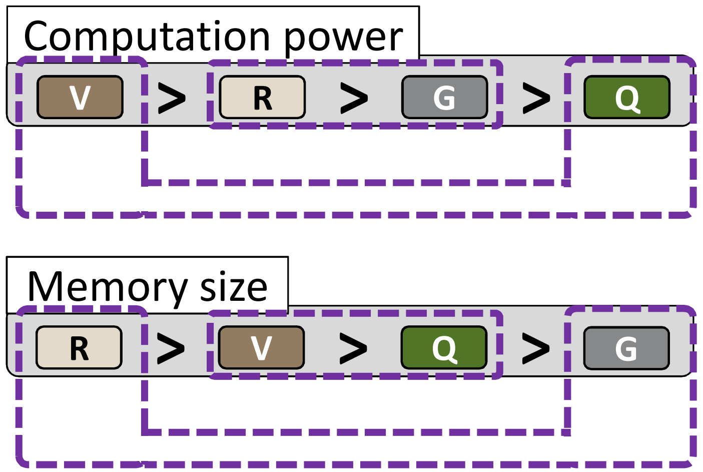
- Performance will be the same across the VWs
- Mitigates the straggler problem

- High communication overhead within each VW

Resource Allocation for Virtual Workers: NP, ED, HD

- HD (Hybrid Distribution)





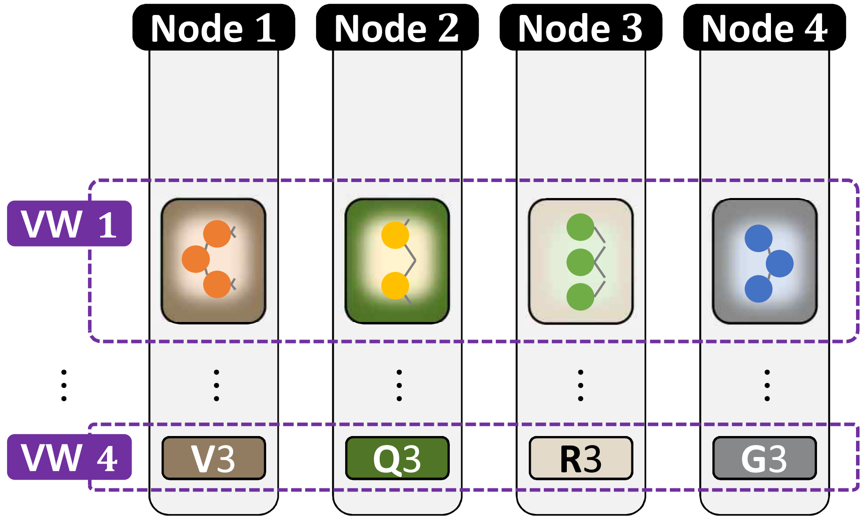
- Mitigates the straggler problem
- Reduces communication overhead within each VW

Parameter Placement

- **Round-robin policy (default)**
  - Can be used in all three policies: **NP**, **ED**, and **HD**

Parameters of each layer:

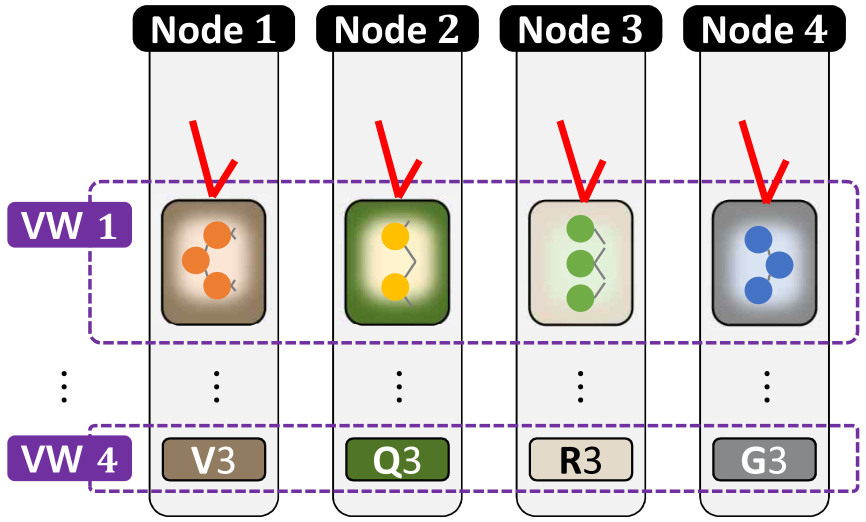
Example: ED



Parameter Placement

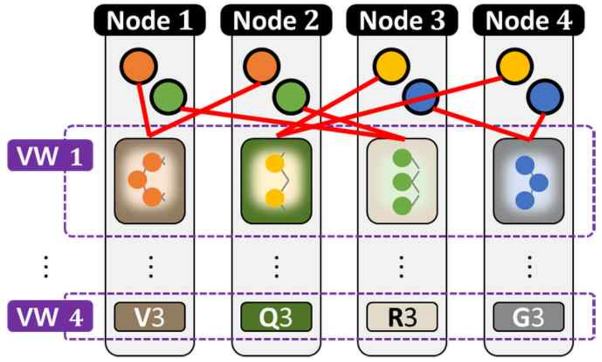
- Local placement policy
  - ED-local

Parameters of each layer:



- Significantly reduces communication overhead

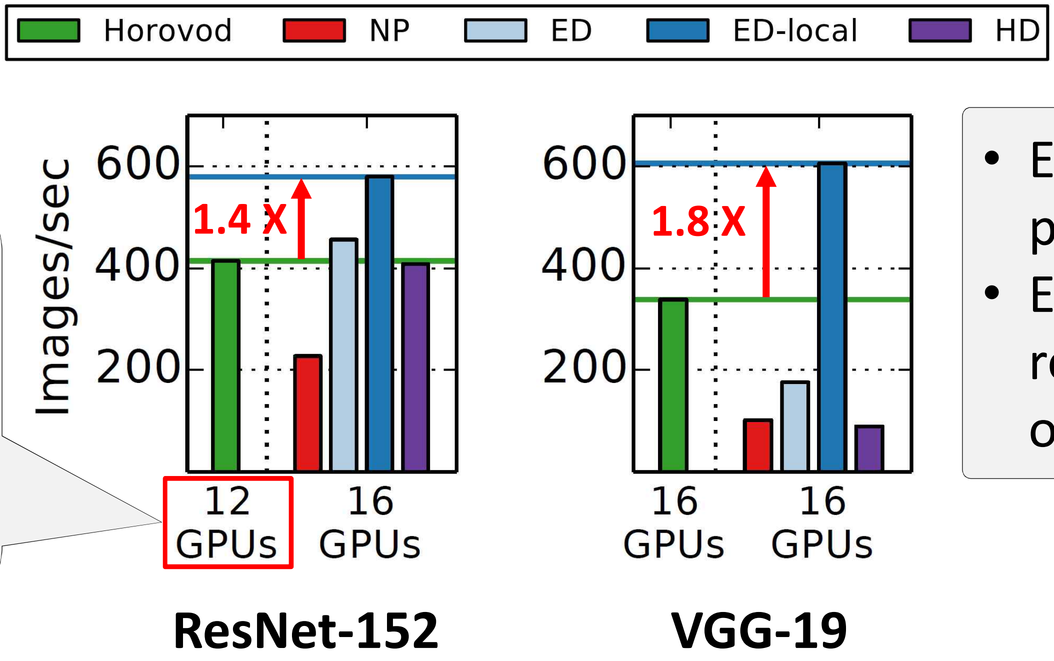
ED



- Parameter communication occurs

Compare Throughput with Horovod

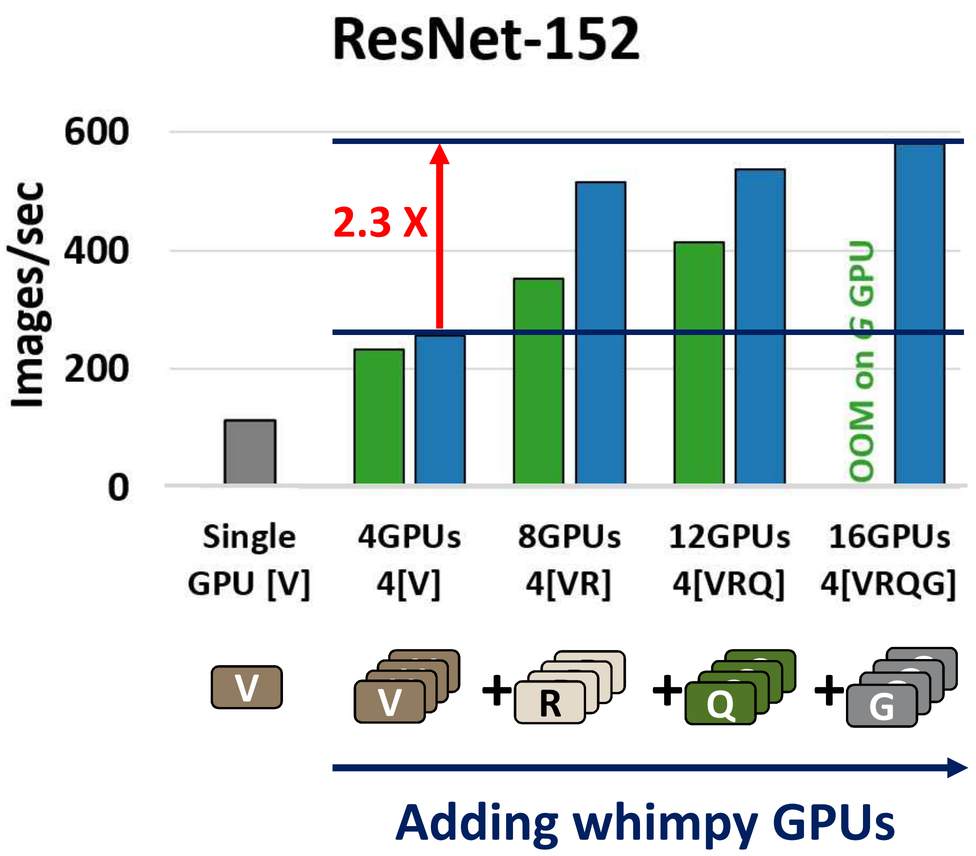
- **Baseline Horovod**
  - State-of-the-art DP using AllReduce



- For ResNet-152, the whole model is too large to be loaded into a single ***G*** type GPU (batch size = 32)

- ED: reduces the straggler problem
- ED-local: significantly reduces communication overhead

Performance Improvement of Adding Whimpy GPUs



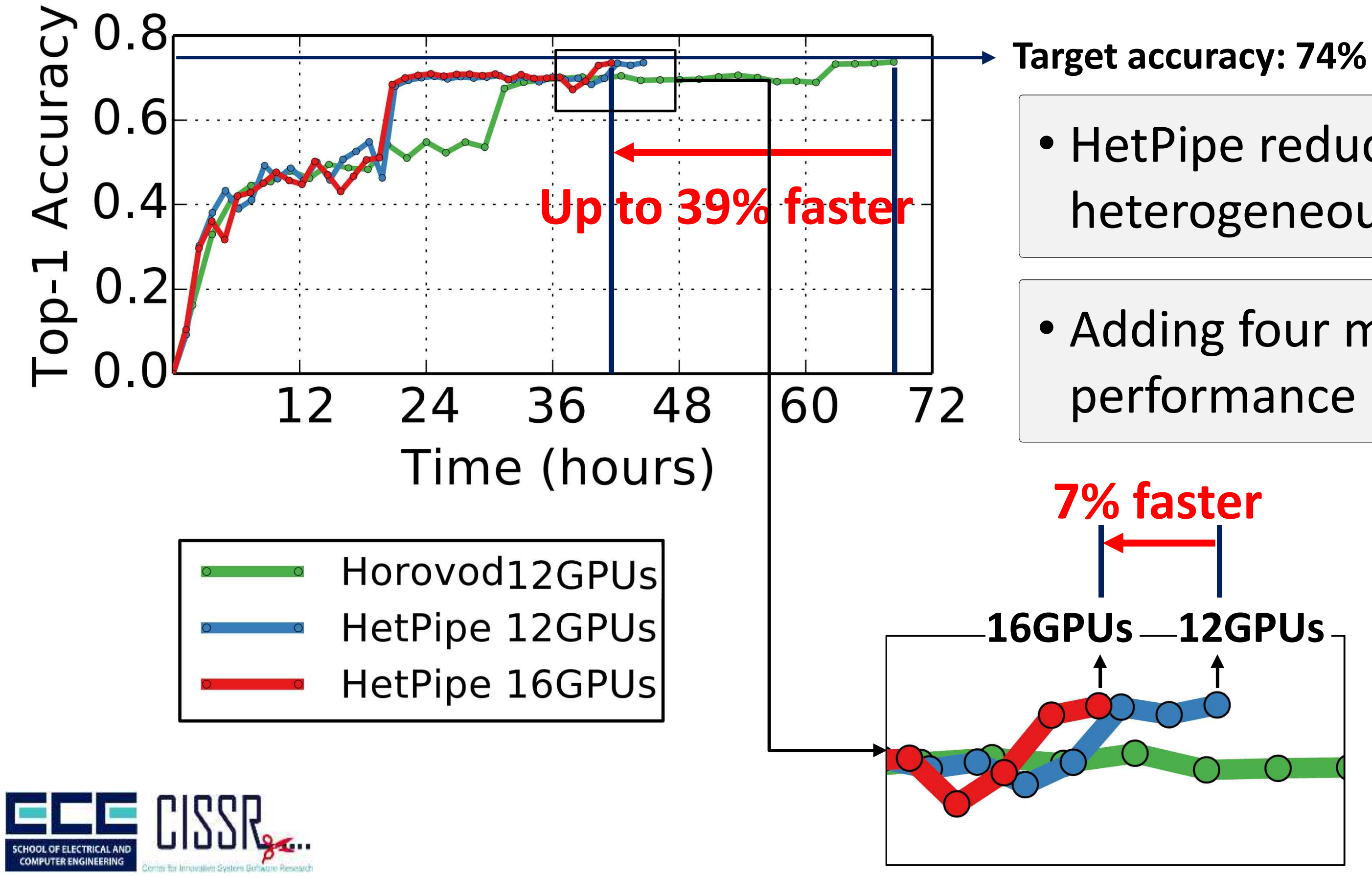
⬛ Single GPU 🟩 Horovod 🟦 HetPipe

- With additional GPUs, HetPipe achieves up to 2.3X speed up

- Additional whimpy systems allow for faster training

Convergence Results

- **ResNet-152**

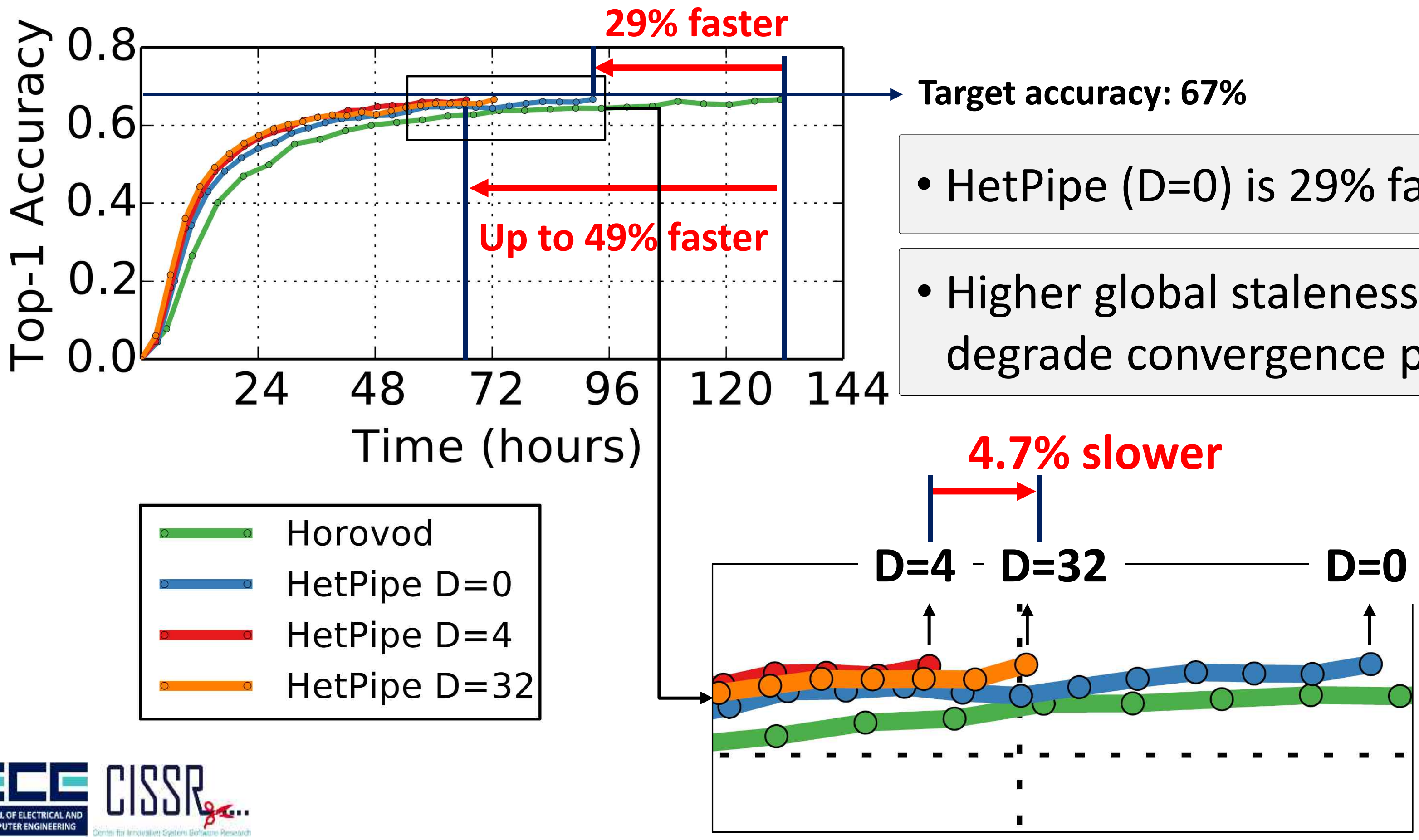


- HetPipe reduces straggler problem in heterogeneous environment

- Adding four more whimpy *G* GPUs, performance improves even more

Convergence Results

- **VGG-19**



- HetPipe (D=0) is 29% faster than Horovod

- Higher global staleness (i.e., 32) can degrade convergence performance

Conclusion

- HetPipe makes it possible to efficiently **train large DNN models with heterogeneous GPUs**
- **Integrate pipelined model parallelism with data parallelism**
- Propose a novel parameter synchronization model: **WSP**
- DNN models converge up to **49% faster with HetPipe**

Thank you!

UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY
2007