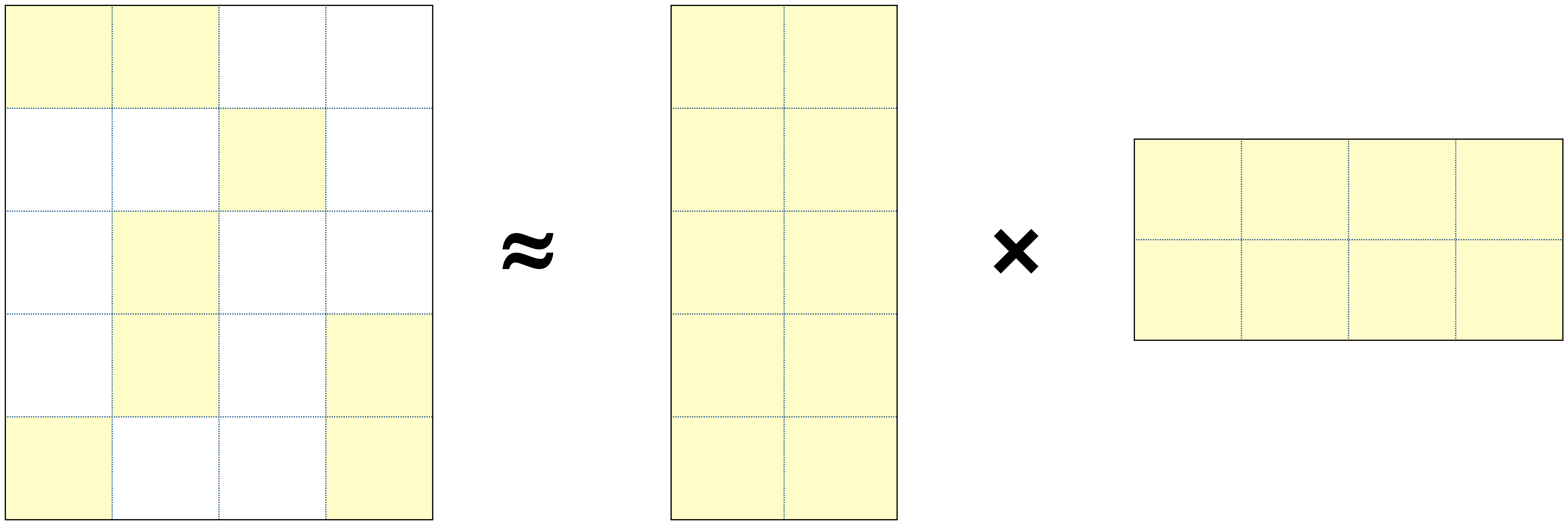
Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent

Rainer Gemulla, et al. (2011)

박민주

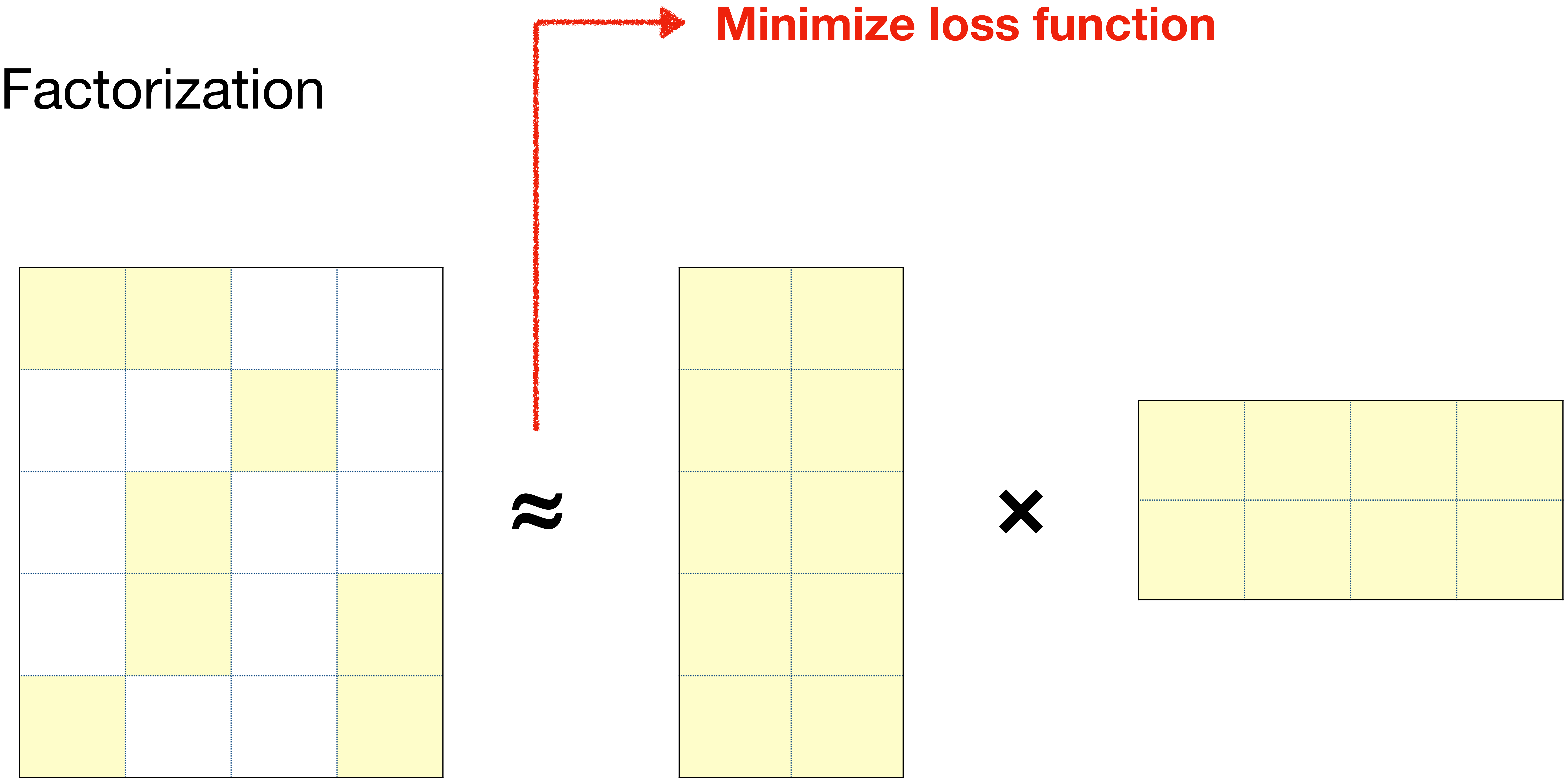
Motivation

- Matrix Factorization



Motivation

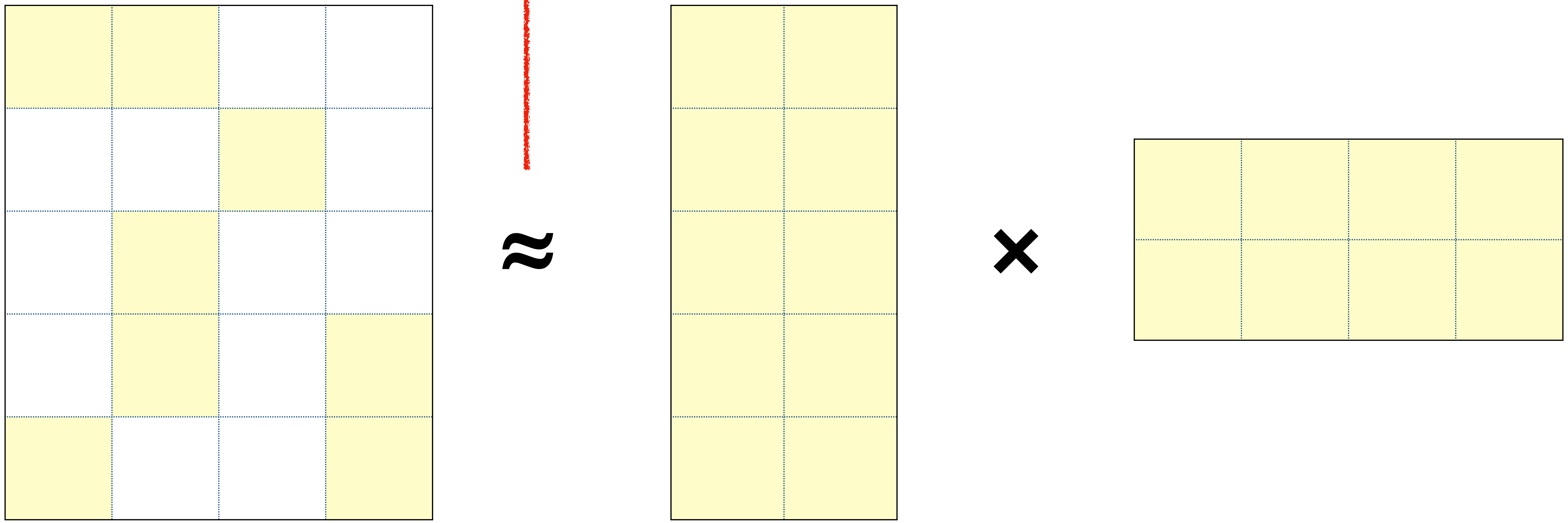
- Matrix Factorization



Motivation

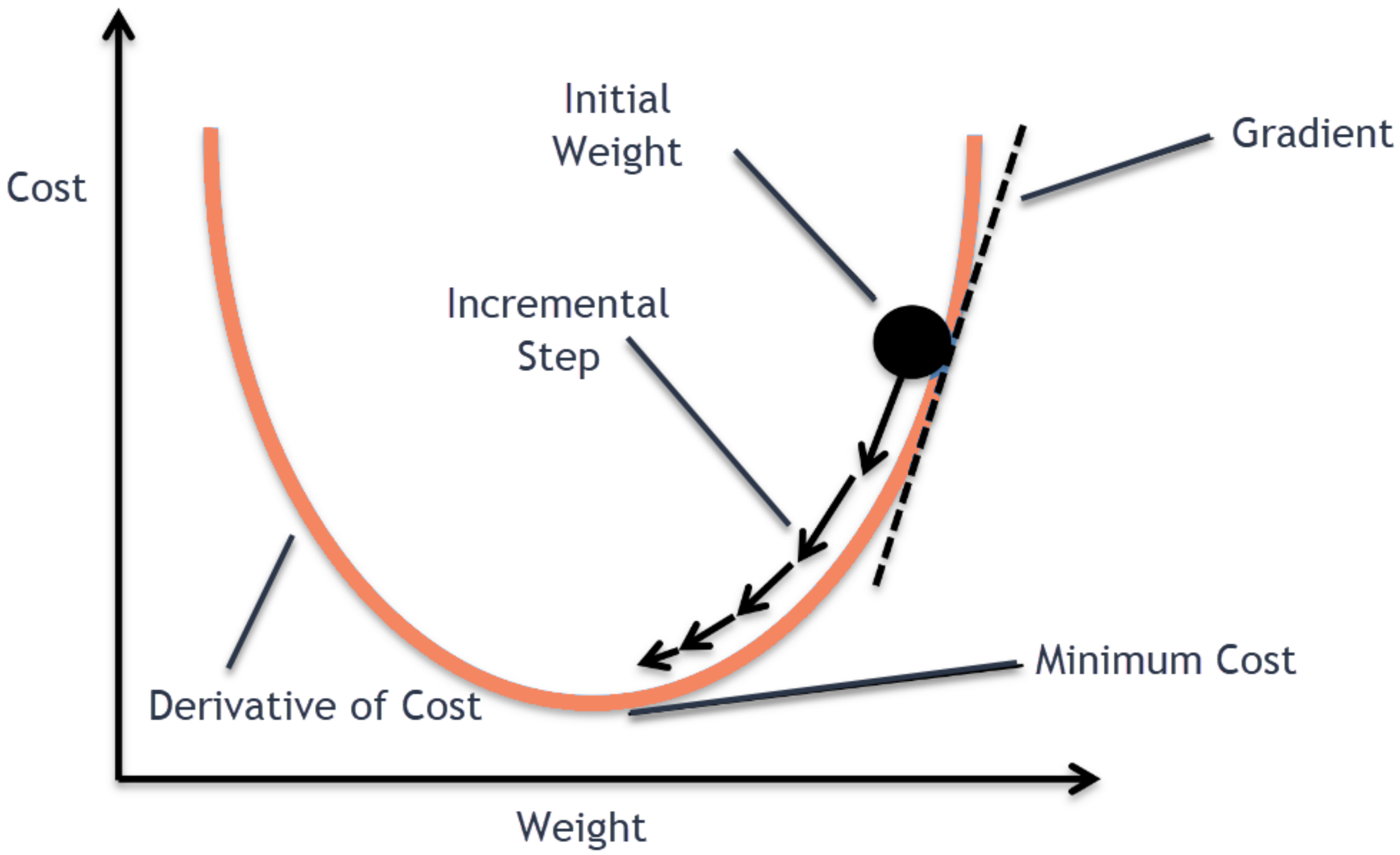
- Matrix Factorization

Minimize loss function with Gradient Descent



Background

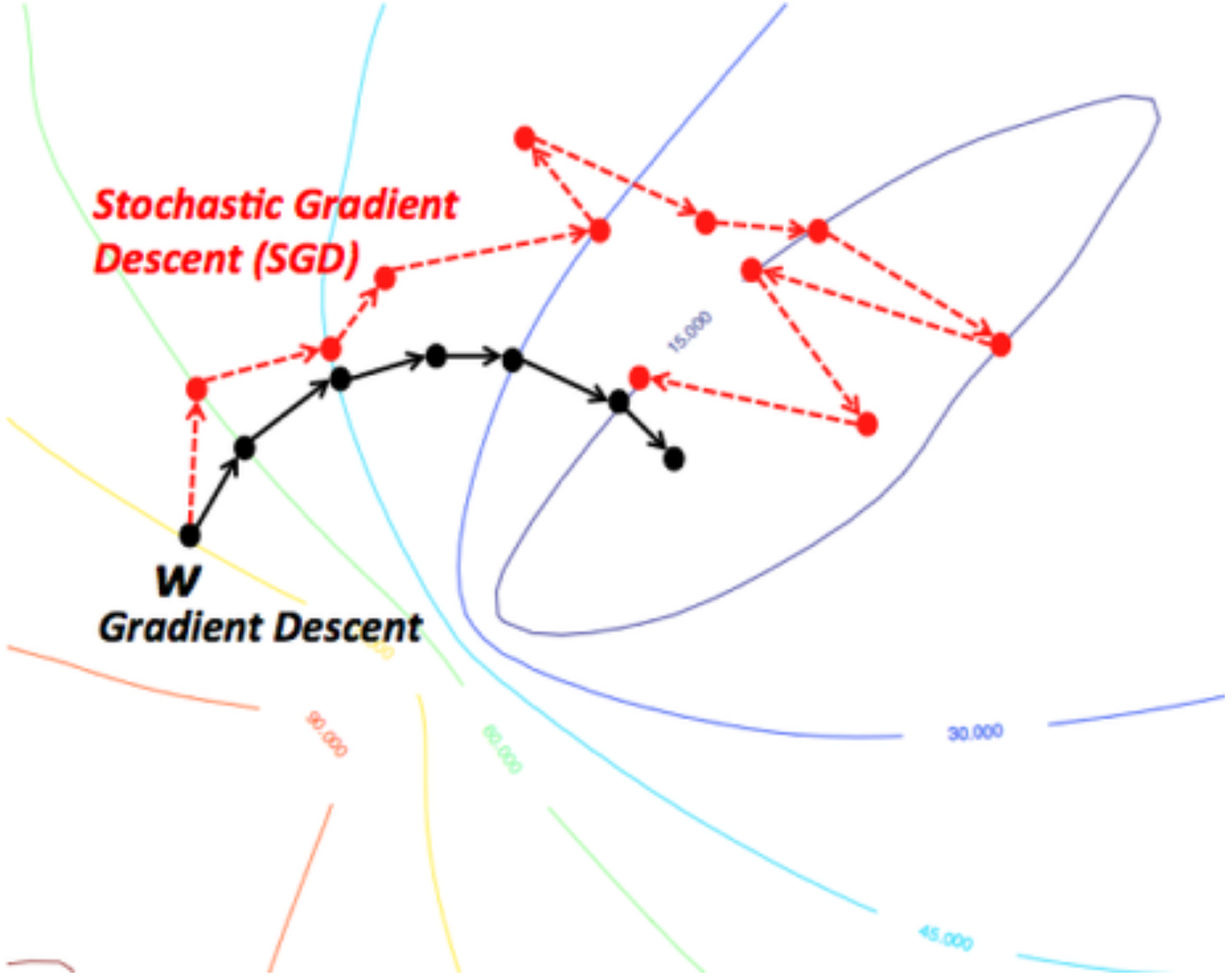
- Gradient Descent



$$\theta_{n+1} = \theta_n - \epsilon \hat{L}'(\theta_n)$$

Background

- Stochastic Gradient Descent



$$\theta_{n+1} = \theta_n - \epsilon \hat{L}'(\theta_n)$$

In practice, with additional projection

$$\theta_{n+1} = \Pi_H[\theta_n - \epsilon \hat{L}'(\theta_n)]$$

* H: Constraint set

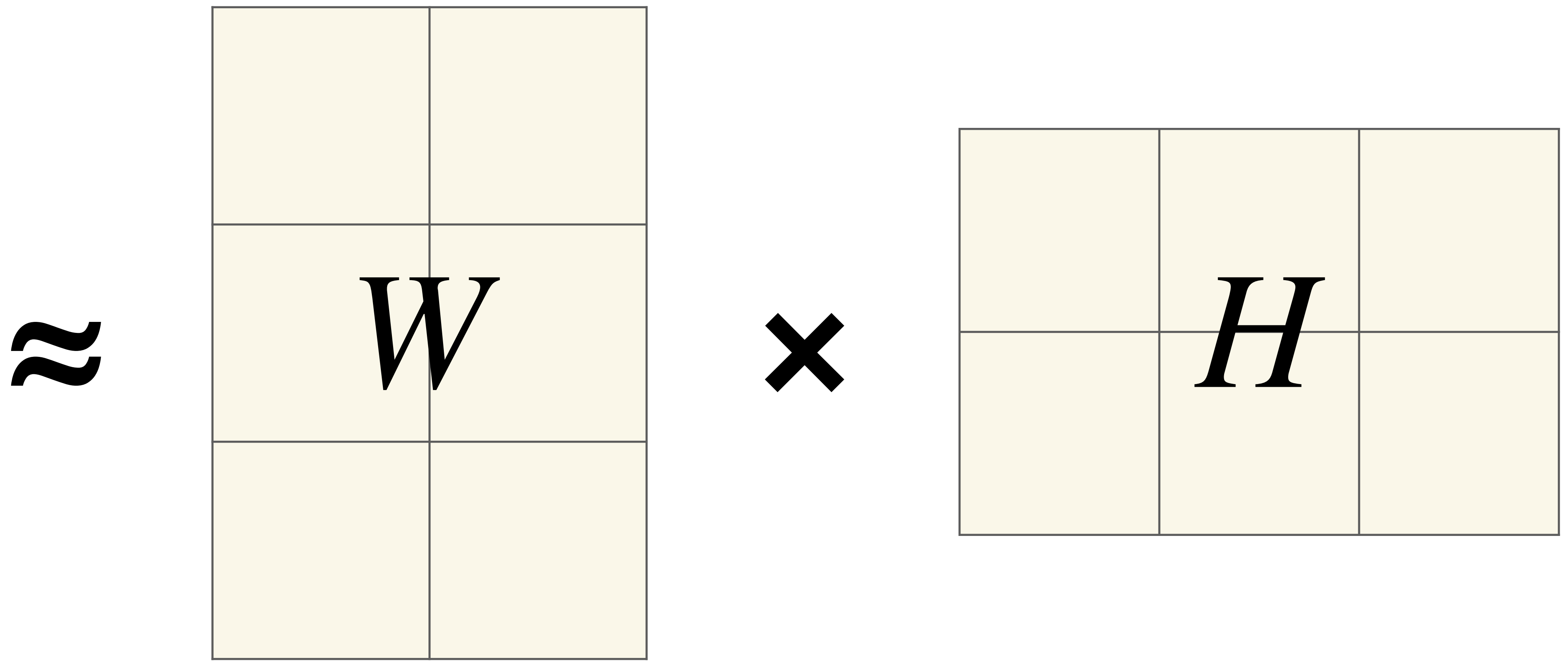
SGD for Matrix Factorization

- Stochastic Gradient Descent

Finding the best model $\underset{W,H}{argmin}\ L(V, W, H) \rightarrow \theta_{n+1} = \theta_n - \epsilon \hat{L}'(\theta_n)$

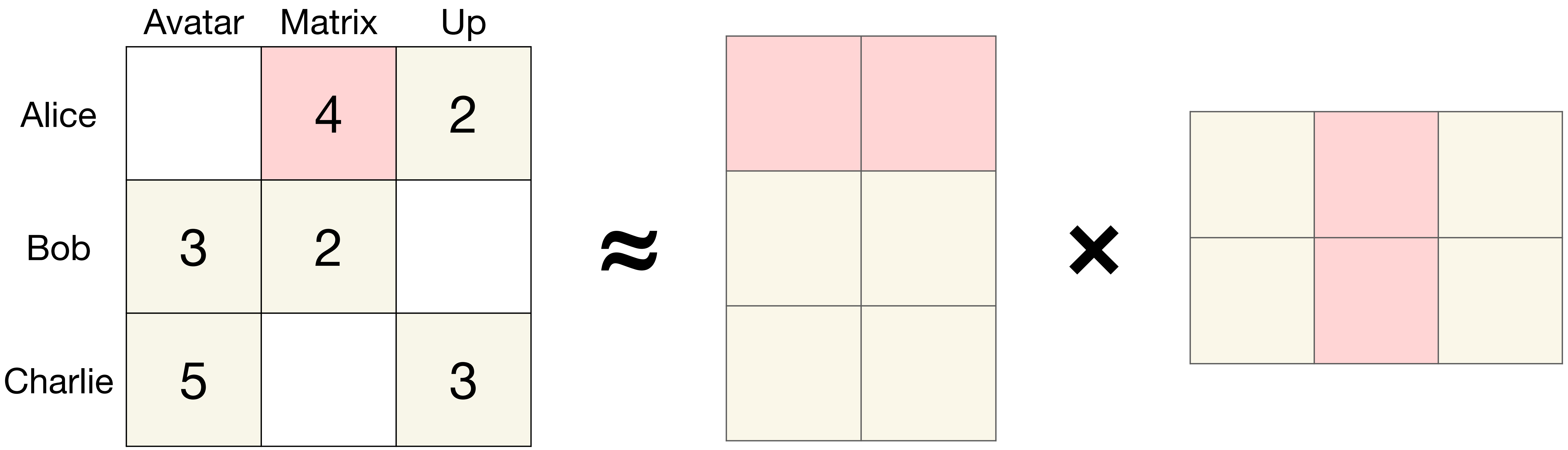
$$\theta = (W, H)$$

	Avatar	Matrix	Up
Alice		4	2
Bob	3	2	
Charlie	5		3



SGD for Matrix Factorization

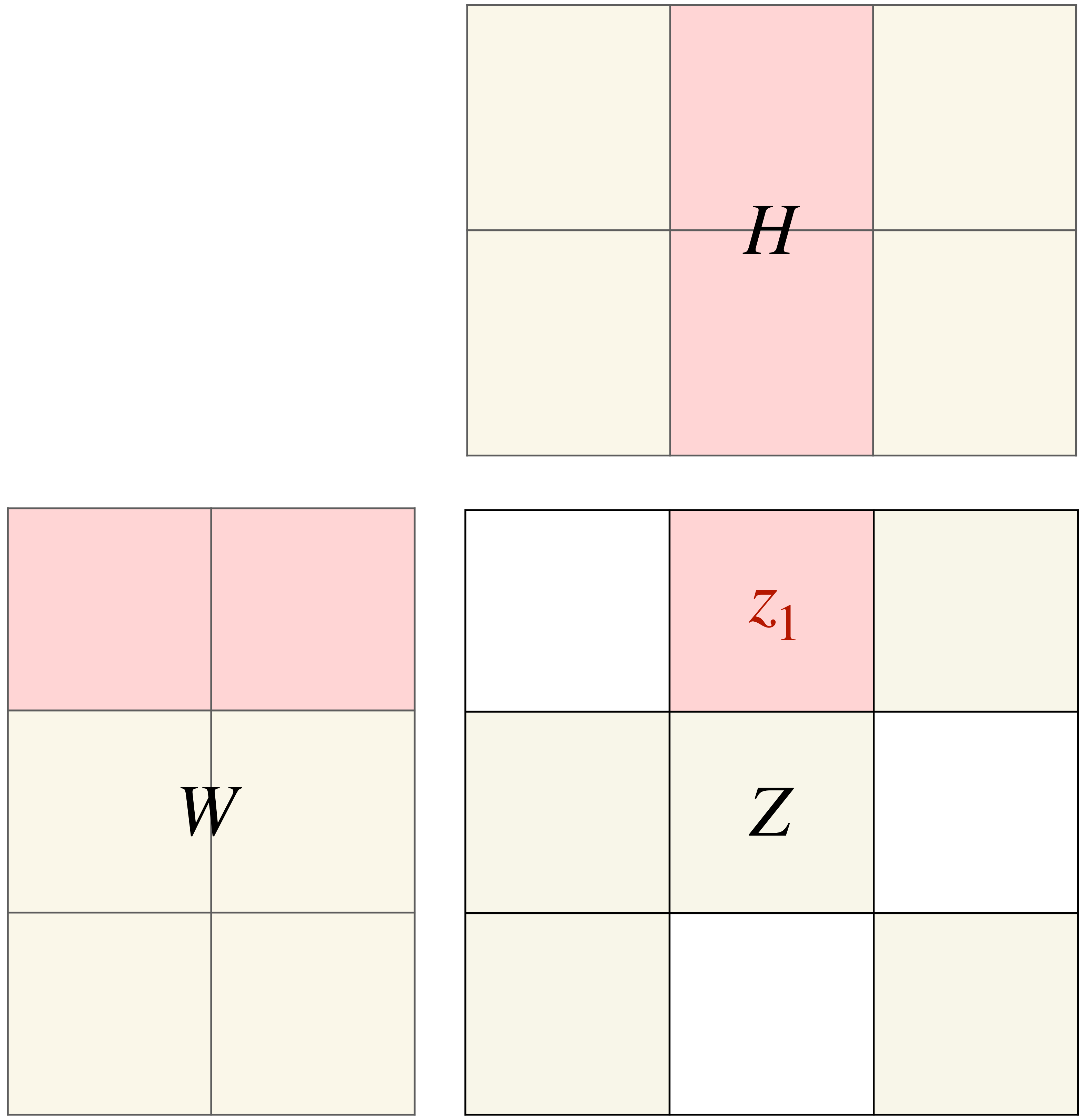
- Stochastic Gradient Descent



- Decompose Loss Function

$$L = \sum_{(i,j)\in Z} l(V_{ij}, W_{i*}, H_{*j}) \qquad (Z = \{(i,j) : V_{ij} \neq 0\})$$

SGD for Matrix Factorization



- $L_z(\theta) = L_{ij}(\theta) = l(V_{ij}, W_{i*}, H_{*j})$

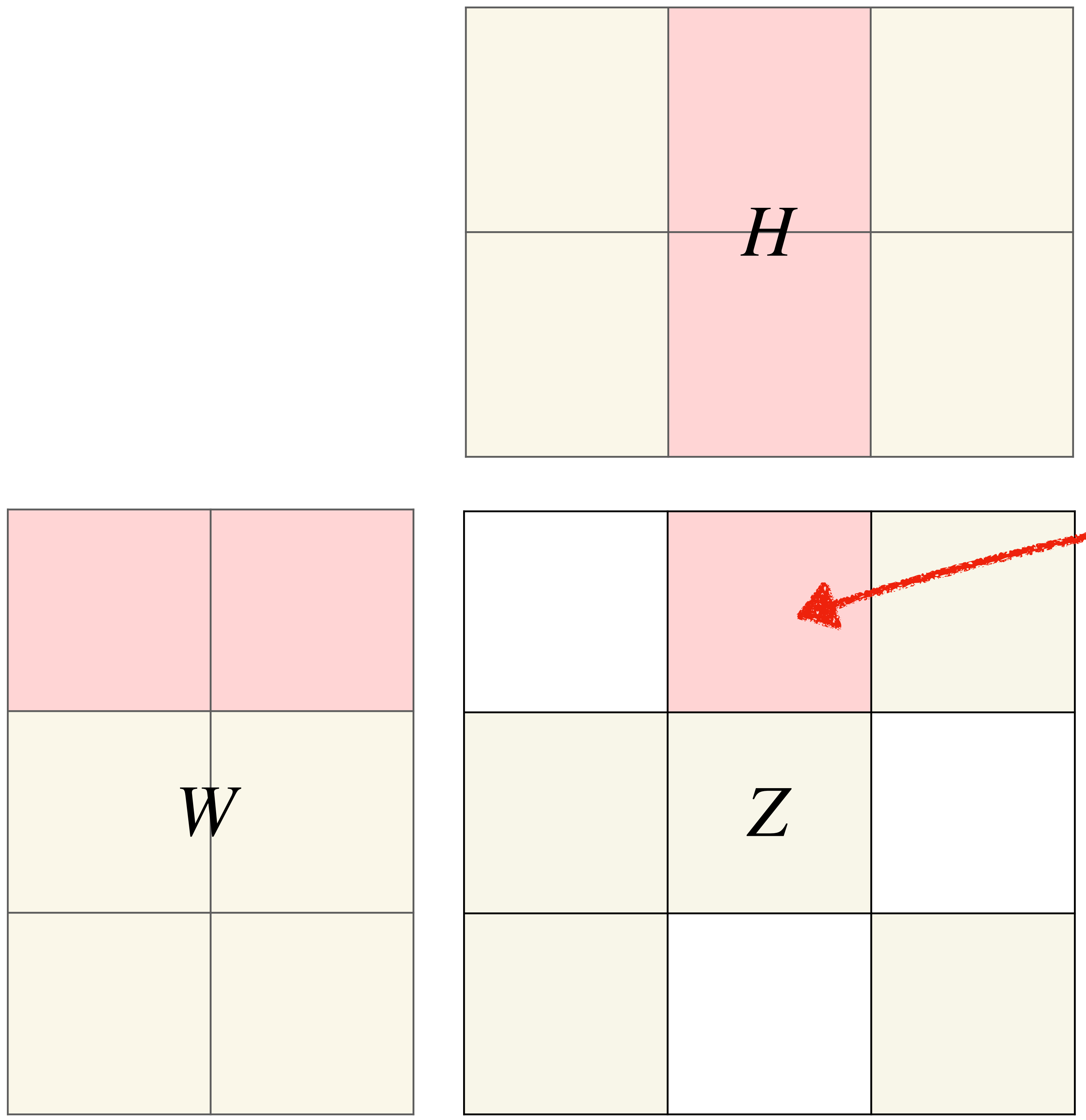
  : local loss at position $z = (i, j)$

- $L'(\theta) = \sum_z L'_z(\theta)$

- $\hat{L}'(\theta) = NL'_z(\theta) \qquad N = |Z|$

ex) $\hat{L}'(\theta) = 6L'_{z_1}(\theta)$

SGD for Matrix Factorization



Algorithm 1 SGD for Matrix Factorization

Require: A training set Z, initial values $\boldsymbol{W}_0$ and $\boldsymbol{H}_0$

  while not converged **do**   /* *step* */

    Select a training point $(i, j) \in Z$ uniformly at random.

    $\boldsymbol{W}'_{i*} \leftarrow \boldsymbol{W}_{i*} - \epsilon_n N \frac{\partial}{\partial \boldsymbol{W}_{i*}} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$

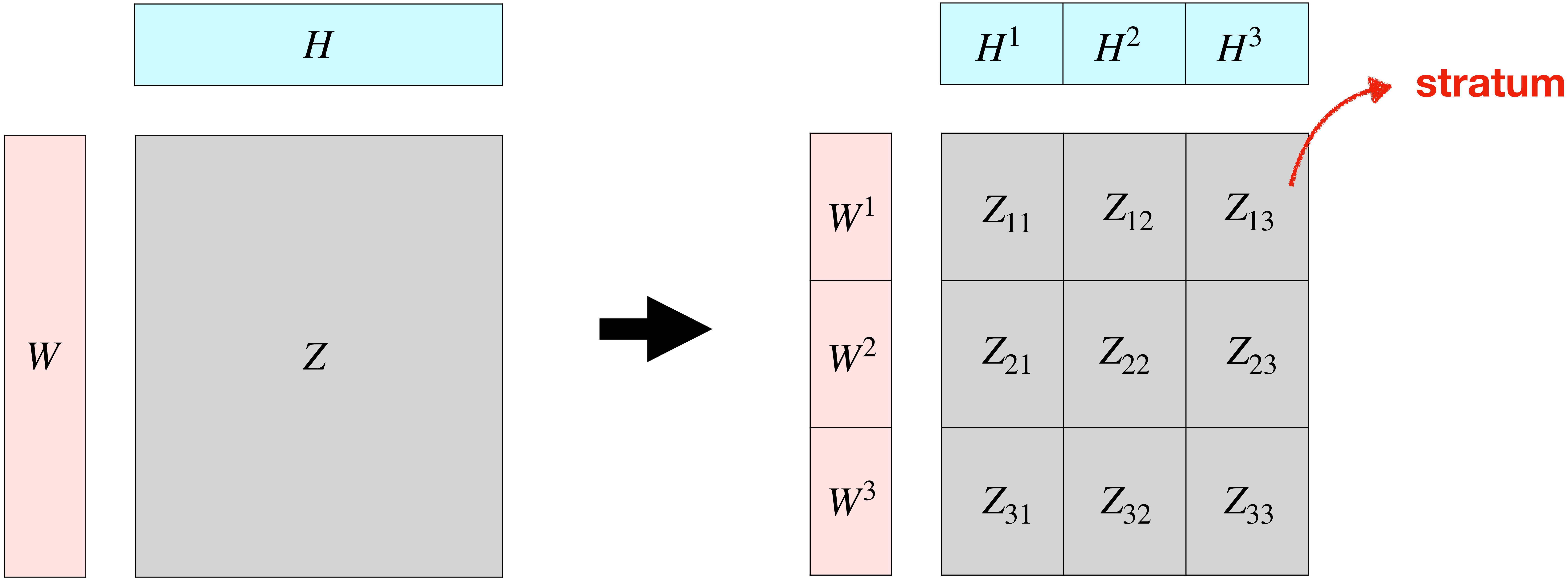
    $\boldsymbol{H}_{*j} \leftarrow \boldsymbol{H}_{*j} - \epsilon_n N \frac{\partial}{\partial \boldsymbol{H}_{*j}} l(\boldsymbol{V}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j})$

    $\boldsymbol{W}_{i*} \leftarrow \boldsymbol{W}'_{i*}$

  end while

SSGD

- Stratified SGD

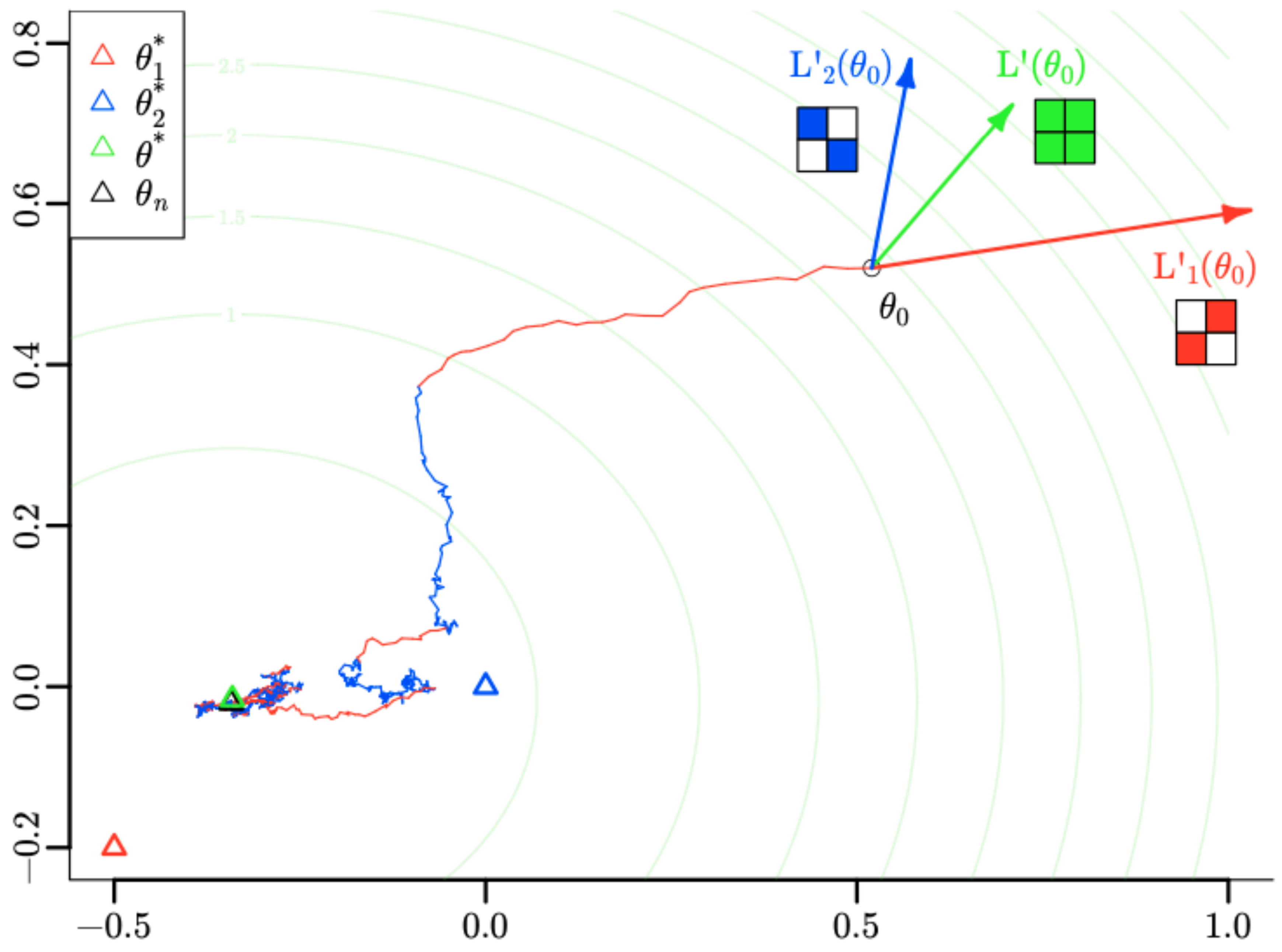


SSGD

- SSGD Algorithm
  - Loss function is decomposed into weighted sum of local loss functions

$$L(\theta) = w_1 L_1(\theta) + w_2 L_2(\theta) + \cdots + w_q L_q(\theta)$$

  - Stratum sequence $\{\gamma_n\} = \{1, \ldots, q\}$
  - Update $\theta_{n+1} = \Pi_H[\theta_n - \epsilon_n \hat{L}'_{\gamma_n}(\theta_n)]$



SSGD

Q) Does SSGD converge?

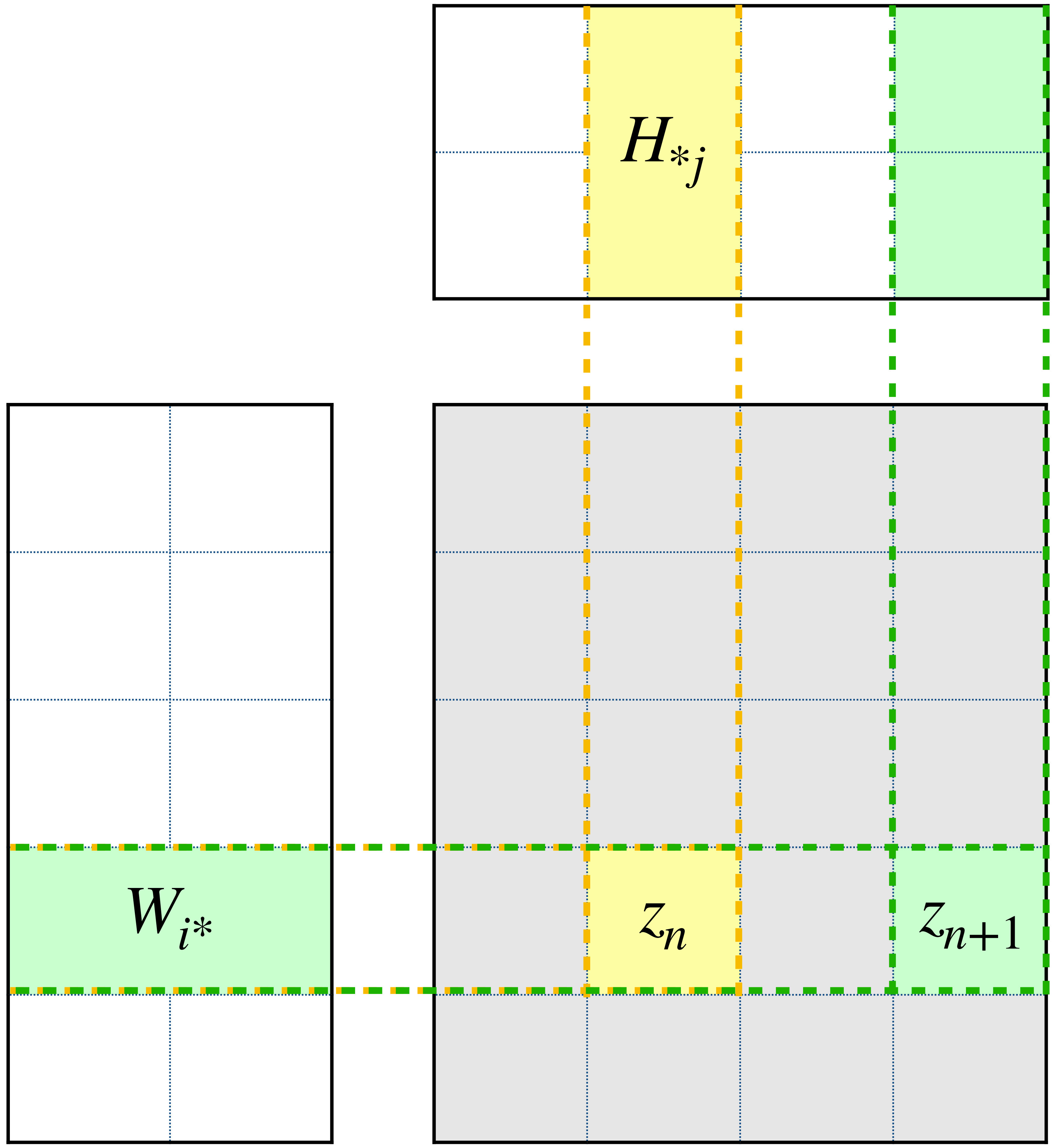
A) SSGD converges under several conditions

1. Step-size conditions
2. Loss conditions
3. Stratification conditions
4. Stratum-sequence conditions

Details in paper

[10] R. Gemulla, P. J. Haas, E. Nijkamp, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. Technical Report RJ10481, IBM Almaden Research Center, San Jose, CA, 2011. Available at www.almaden.ibm.com/cs/people/peterh/dsgdTechRep.pdf.

SGD cannot be used directly for rank-r factorization



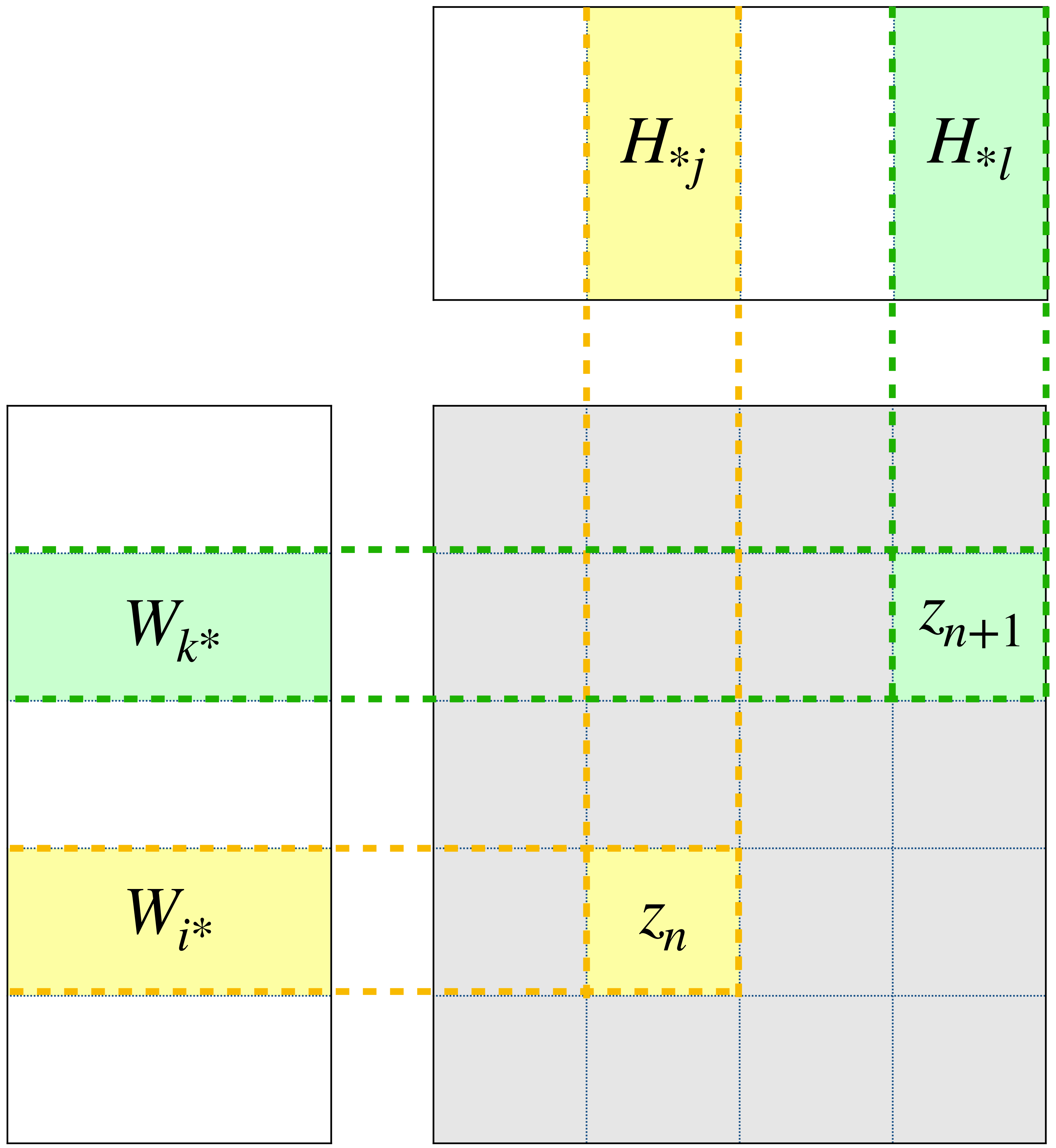
Individual steps depend on each other

$$\theta_{n+1} = \theta_n - \epsilon \hat{L}'(\theta_n)$$

⬇

Stratify the training set Z into strata so that <u>each individual stratum are independent</u>

DSGD



- Distributed SGD
  - "Interchangeability"
    : share neither row nor column
  - Two machines can be updated at the same time

Read W_{i*} **and** H_{*j}
Compute gradient of $L'(W_{i*}, H_{*j})$
Update W_{i*} **and** H_{*j}

No conflicts!

Read W_{k*} **and** H_{*l}
Compute gradient of $L'(W_{k*}, H_{*l})$
Update W_{k*} **and** H_{*l}

DSGD

- The idea behind DSGD is to find such blocks that have no data in common and update them simultaneously

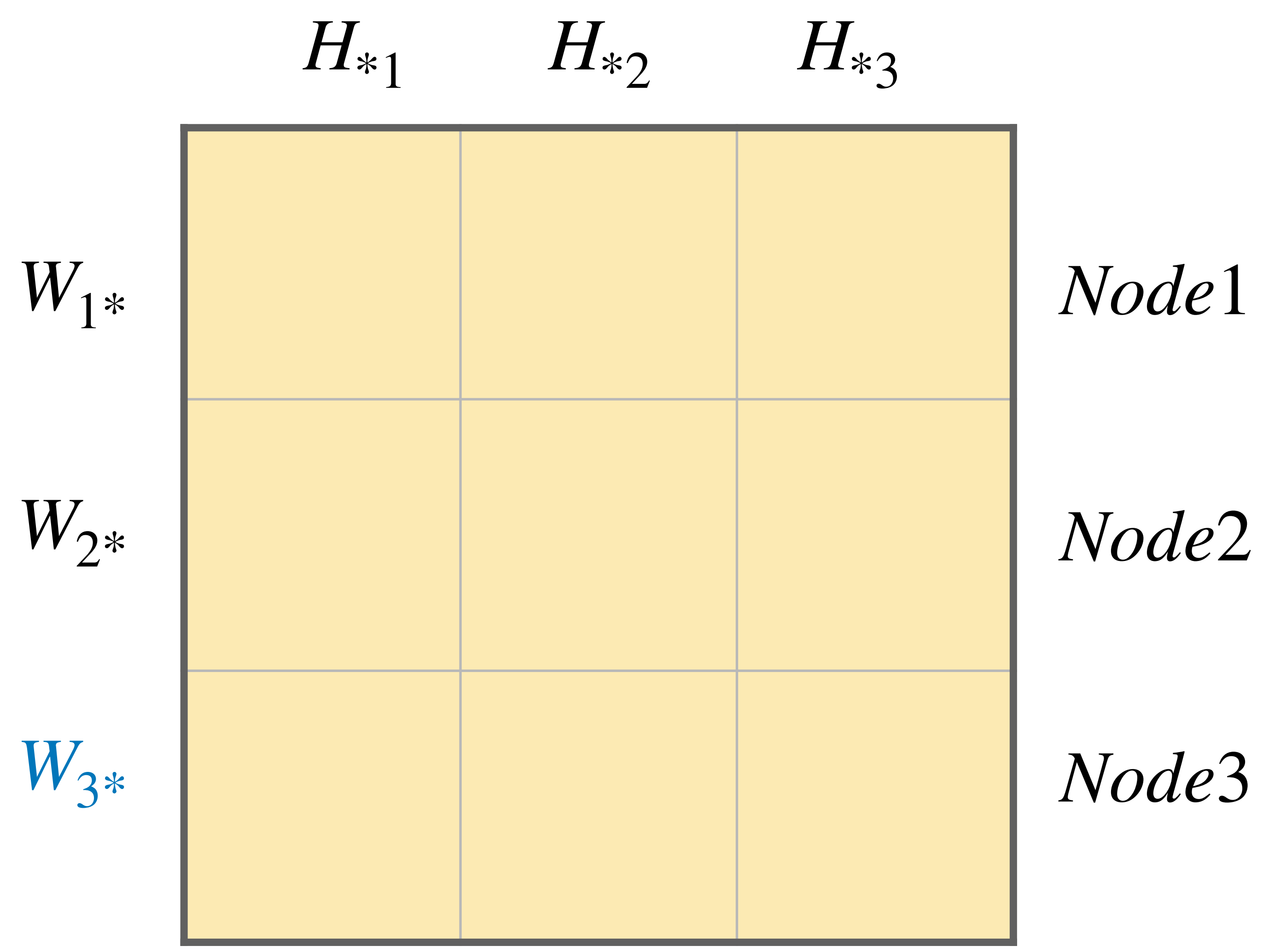
- **Goal** : minimize the loss function and update factor vectors (W, H)

→ **Proceeded in stratums**

DSGD

Algorithm 2 DSGD for Matrix Factorization

Require: $\boldsymbol{Z}$, $\boldsymbol{W}_0$, $\boldsymbol{H}_0$, cluster size d
  $\boldsymbol{W} \leftarrow \boldsymbol{W}_0$ and $\boldsymbol{H} \leftarrow \boldsymbol{H}_0$
  Block $\boldsymbol{Z}$ / $\boldsymbol{W}$ / $\boldsymbol{H}$ into $d \times d$ / $d \times 1$ / $1 \times d$ blocks
  while not converged **do** /* *epoch* */
    Pick step size ϵ
    for $s = 1, \ldots, d$ **do** /* *subepoch* */
      Pick d blocks $\{\boldsymbol{Z}^{1j_1}, \ldots, \boldsymbol{Z}^{dj_d}\}$ to form a stratum
      for $b = 1, \ldots, d$ **do** /* *in parallel* */
        Run SGD on the training points in $\boldsymbol{Z}^{bj_b}$ (step size $= \epsilon$)
      end for
    end for
  end while



DSGD

Algorithm 2 DSGD for Matrix Factorization

Require: $\boldsymbol{Z}, \boldsymbol{W}_0, \boldsymbol{H}_0$, cluster size d

  $\boldsymbol{W} \leftarrow \boldsymbol{W}_0$ and $\boldsymbol{H} \leftarrow \boldsymbol{H}_0$

  Block $\boldsymbol{Z}$ / $\boldsymbol{W}$ / $\boldsymbol{H}$ into $d \times d$ / $d \times 1$ / $1 \times d$ blocks

  while not converged **do** /* *epoch* */

    Pick step size ϵ

    for $s = 1, \ldots, d$ **do** /* *subepoch* */

      Pick d blocks $\{\boldsymbol{Z}^{1j_1}, \ldots, \boldsymbol{Z}^{dj_d}\}$ to form a stratum

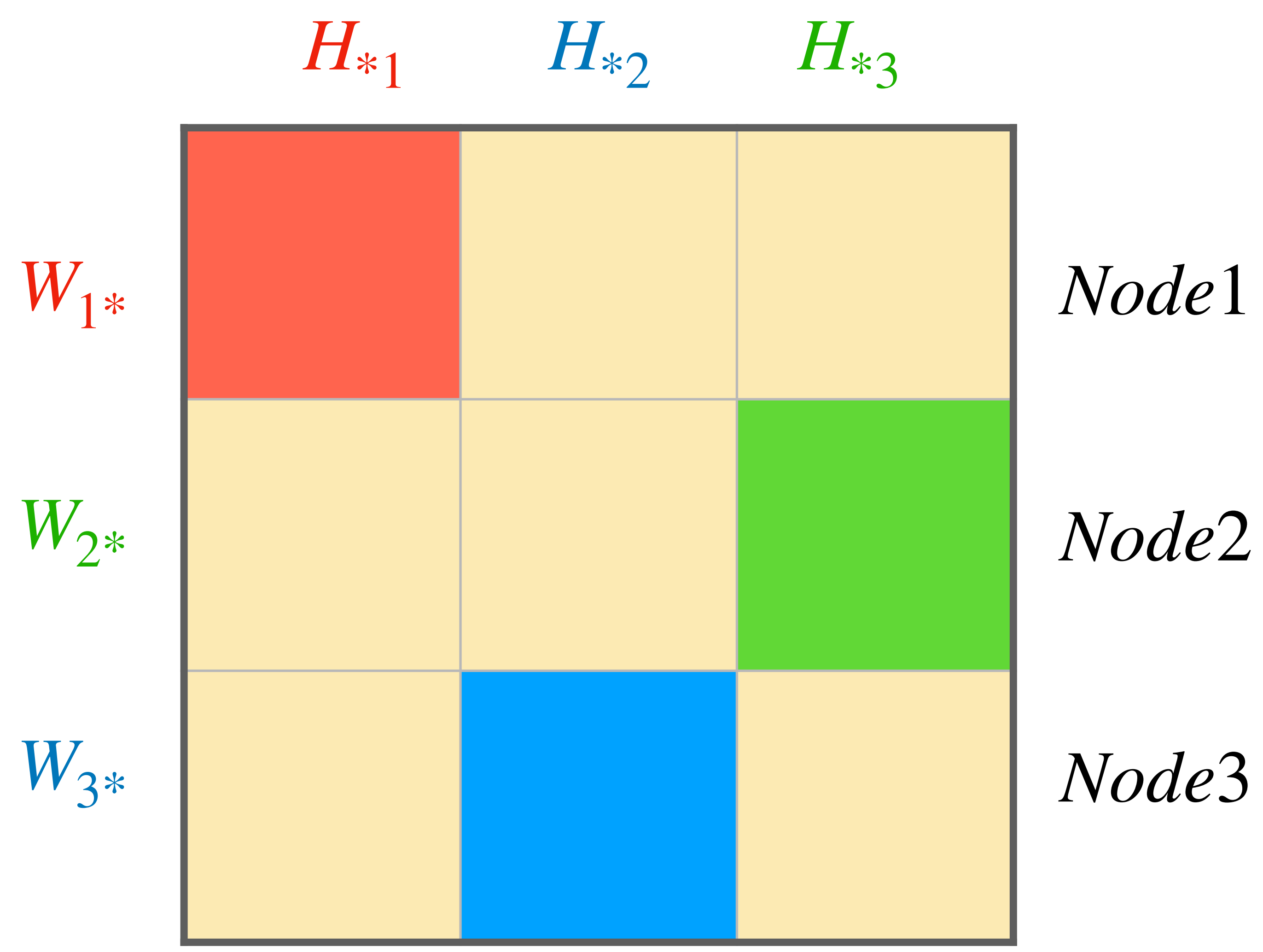
      for $b = 1, \ldots, d$ **do** /* *in parallel* */

        Run SGD on the training points in $\boldsymbol{Z}^{bj_b}$ (step size $= \epsilon$)

      end for

    end for

  end while



Sequence is chosen under SSGD conditions → **DSGD will converge!**

DSGD

Algorithm 2 DSGD for Matrix Factorization

Require: $\boldsymbol{Z}, \boldsymbol{W}_0, \boldsymbol{H}_0$, cluster size d

  $\boldsymbol{W} \leftarrow \boldsymbol{W}_0$ and $\boldsymbol{H} \leftarrow \boldsymbol{H}_0$

  Block $\boldsymbol{Z}$ / $\boldsymbol{W}$ / $\boldsymbol{H}$ into $d \times d$ / $d \times 1$ / $1 \times d$ blocks

  while not converged **do** /* *epoch* */

    Pick step size ϵ

    for $s = 1, \ldots, d$ **do** /* *subepoch* */

      Pick d blocks $\{\boldsymbol{Z}^{1j_1}, \ldots, \boldsymbol{Z}^{dj_d}\}$ to form a stratum

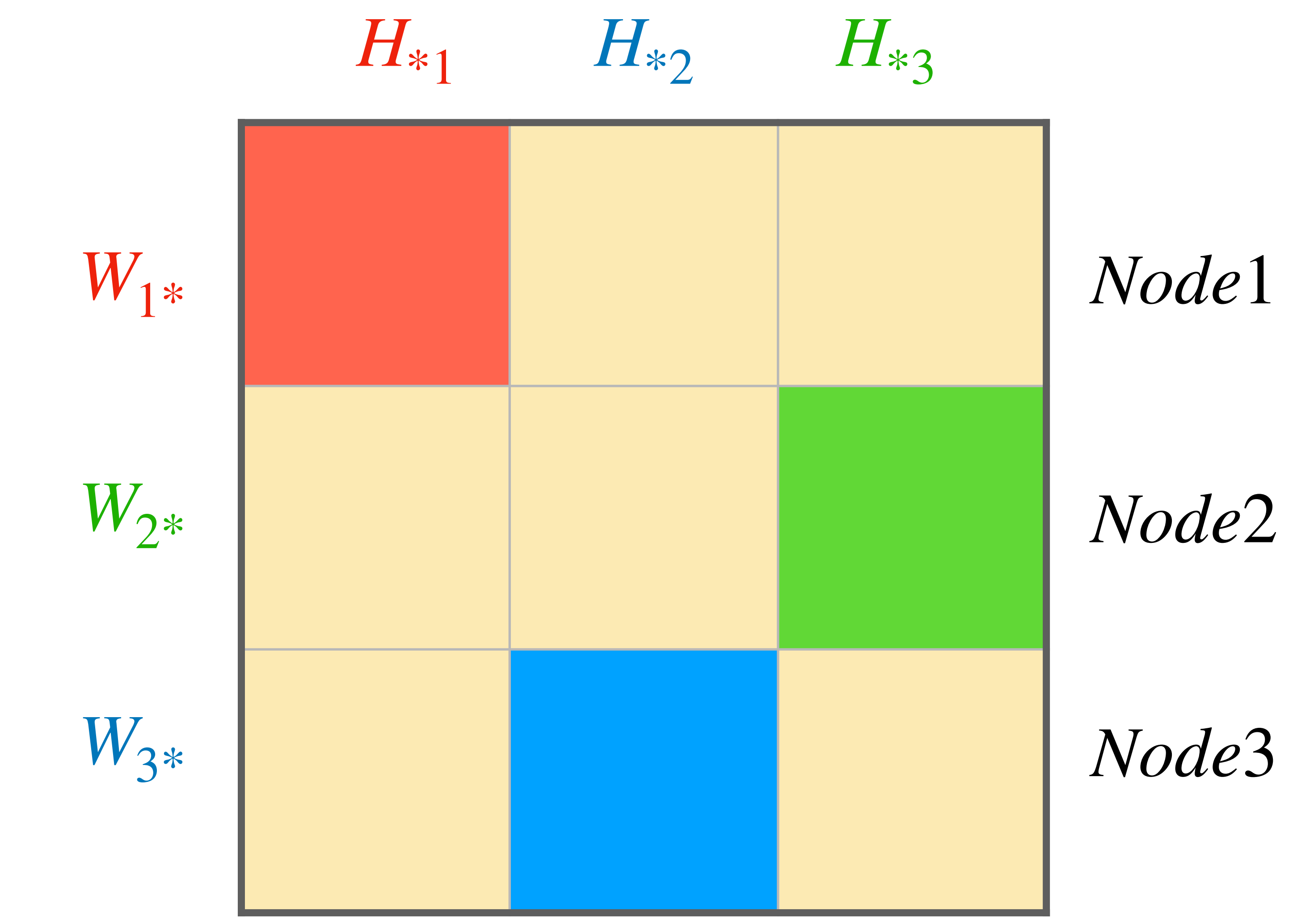
      for $b = 1, \ldots, d$ **do** /* *in parallel* */

        Run SGD on the training points in $\boldsymbol{Z}^{bj_b}$ (step size $= \epsilon$)

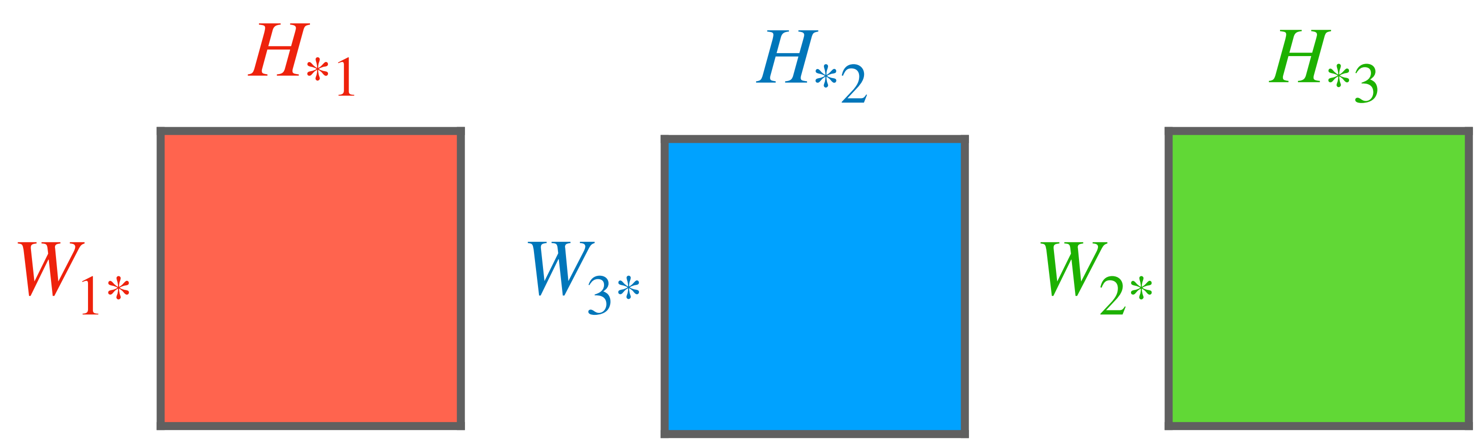
      end for

    end for

  end while



Run SGD on



DSGD

- Sum up losses in each stratum

$$L(W,H) = \sum_{s=1}^{q} w_s \underline{L_s(W,H)}$$

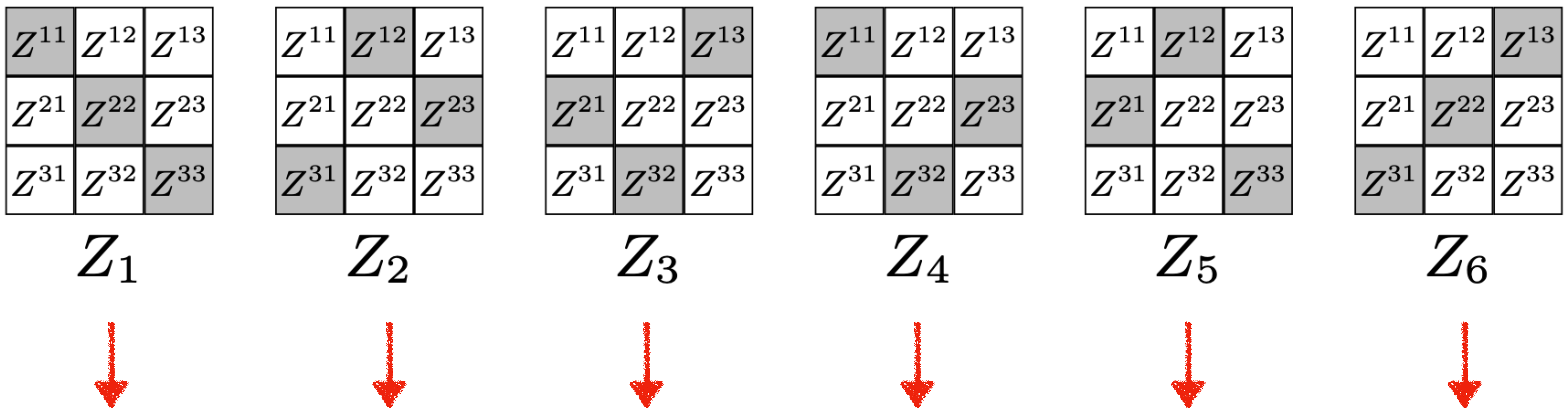
$$L_s(W,H) = c_s \sum_{(i,j)\in Z_s} L_{ij}(W,H)$$ *c_s: stratum-specific constant

- Gradient estimate

$$\hat{L}'_s(W,H) = N_s c_s L'_{ij}(W,H)$$ *$N_s = |Z_s|$:size of stratum

DSGD

- Distributed SGD



$$\hat{L}'_1(W,H) = 3c_1 L'_{ij}(W,H) \qquad \cdots \qquad \hat{L}'_6(W,H) = 3c_6 L'_{ij}(W,H)$$

Sum with weights

$$\hat{L}'(W,H)$$

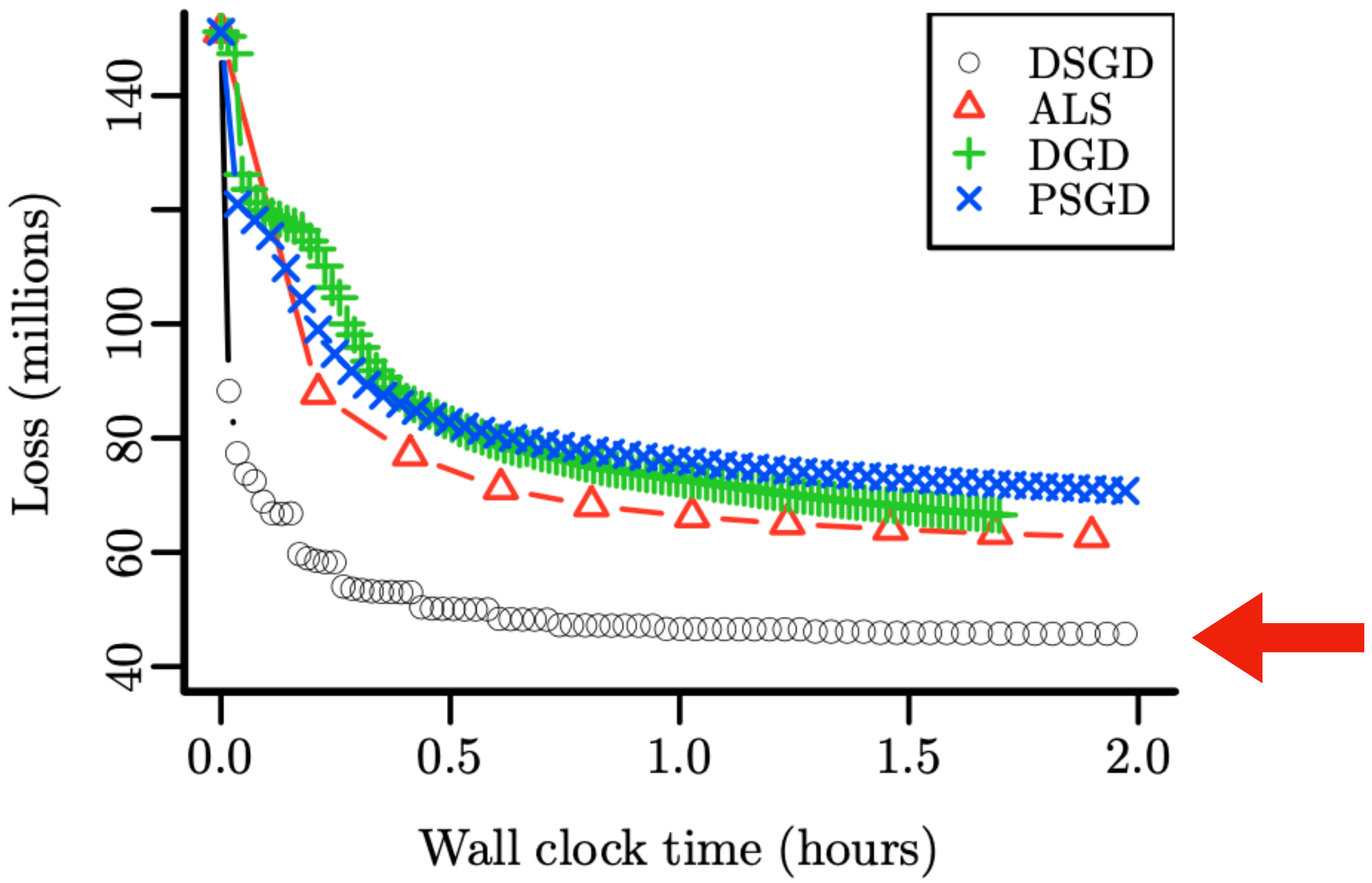
Experiment

- Compared various factorization algorithms w.r.t
  - Convergence
  - Runtime efficiency
  - Scalability

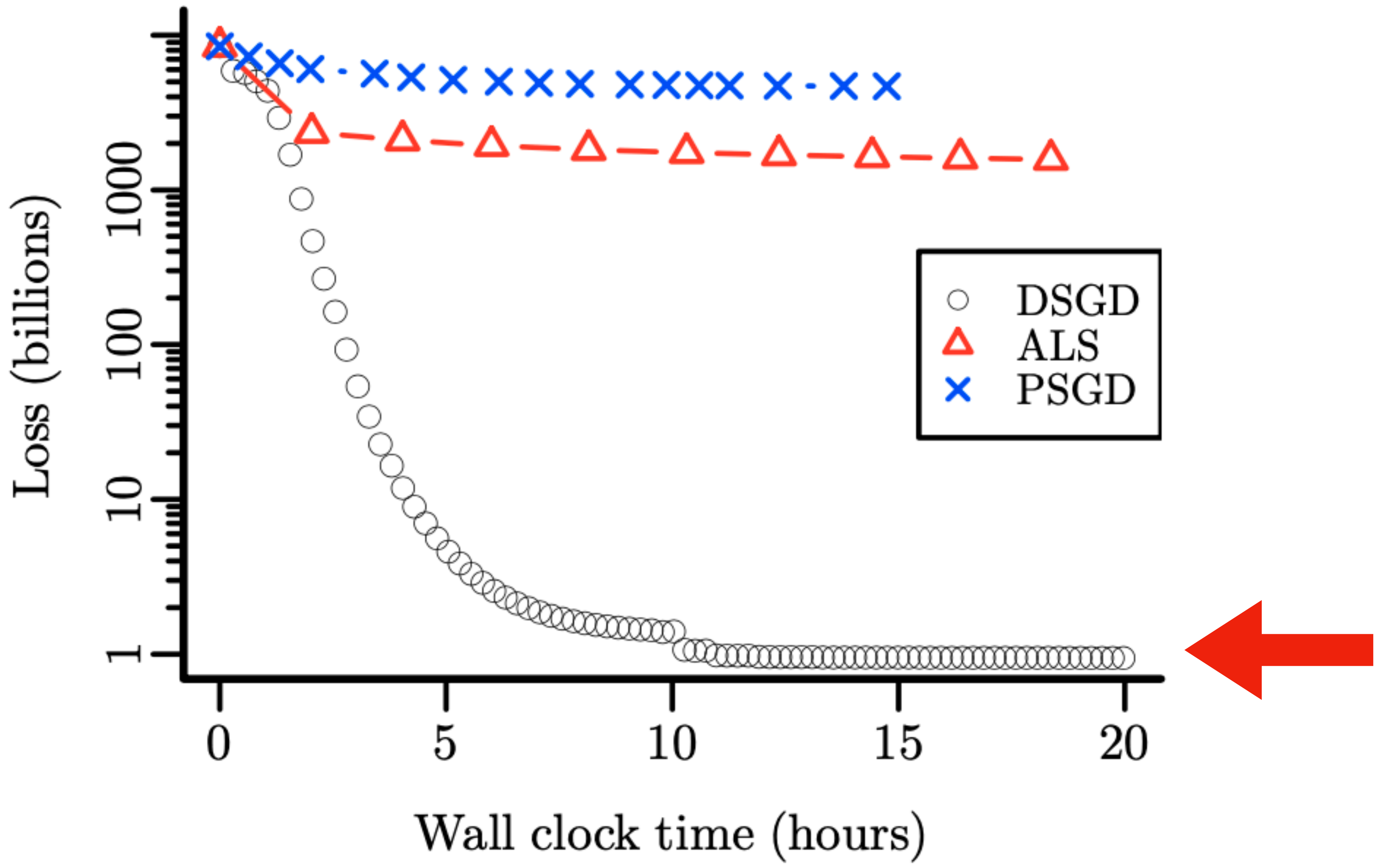
- Implement on MapReduce
  - In-memory experiment : R, C (Netflix competition dataset)
  - Large scaling experiment : Hadoop (larger synthetic dataset)

Experiment

1. Convergence & 2. Runtime Efficiency



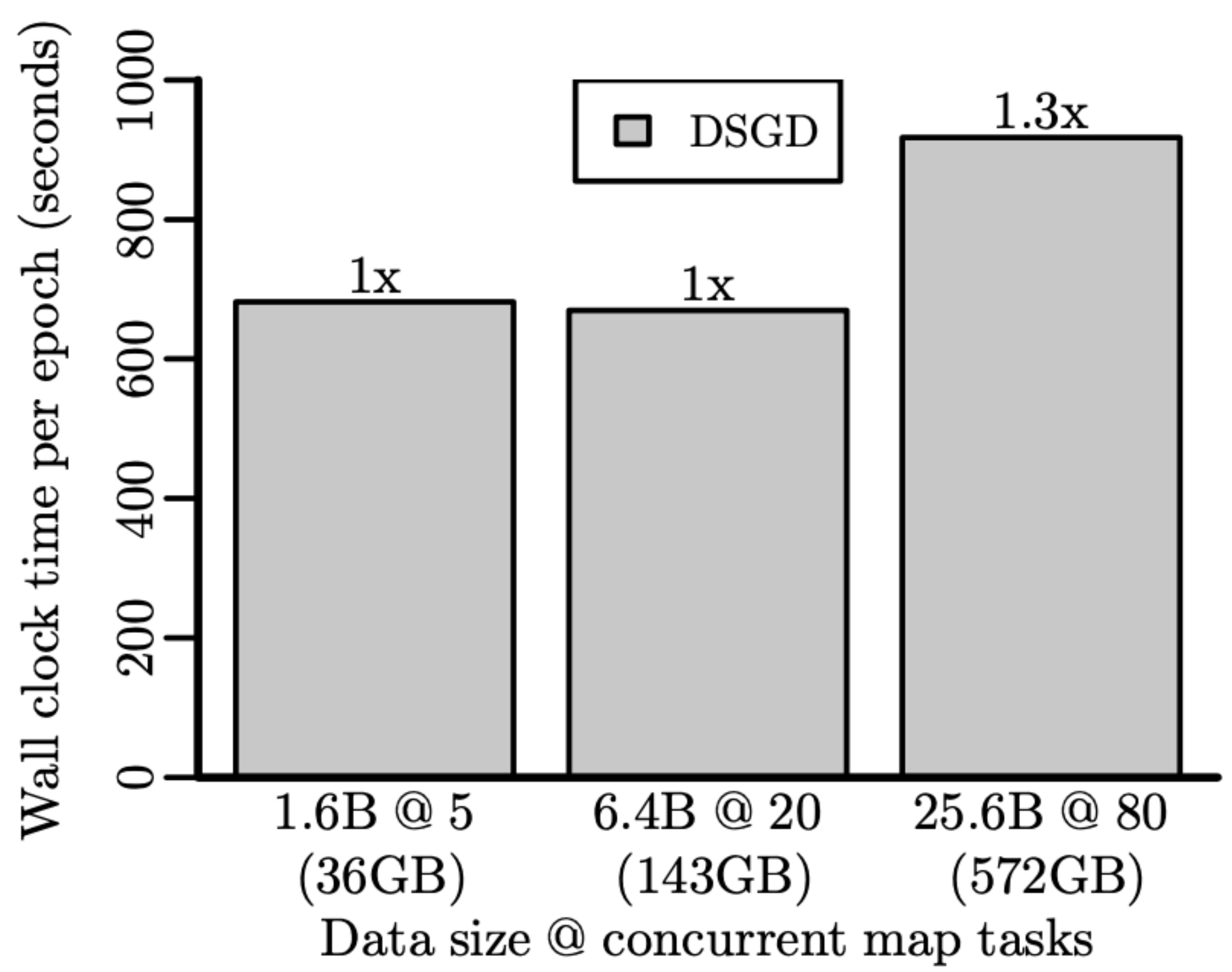
(a) Netflix data (NZSL, R cluster @ 64)



(b) Synth. data (L2, $\lambda = 0.1$, R cluster @ 64)

Experiment

3. Scalability



(c) Scalability (Hadoop cluster)

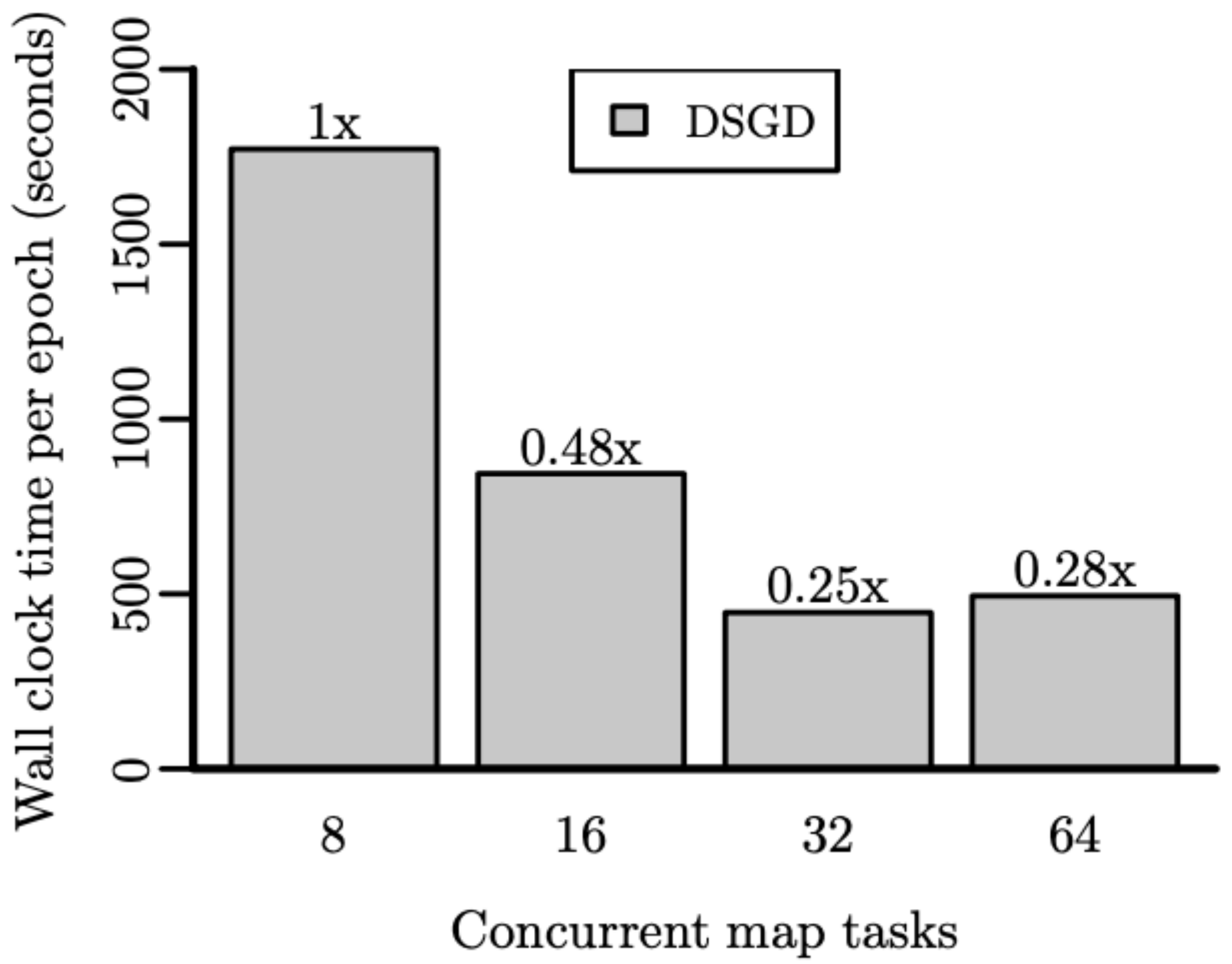


Figure 3: Speed-up experiment (Hadoop cluster, 143GB data)

Conclusion

- Develop stratified version of SGD **(SSGD)**
- Refine SSGD to obtain **DSGD**, a distributed matrix-factorization algorithm that can <u>efficiently handle web-scale matrices</u>
- Future Work
  - Investigate alternative loss functions, regularizations, stratification schemes, emerging distributed-processing platforms
  - Extend to other applications

Thank you