SySCD
A System-Aware Parallel Coordinate Descent Algorithm

Nikolas Ioannou*
IBM Research

Celestine Mendler-Dünner*
UC Berkeley

Thomas Parnell
IBM Research

*equal contribution

Parallel Coordinate Descent

$$\min_{\boldsymbol{\alpha}} f(A\boldsymbol{\alpha}) + \sum_i g_i(\alpha_i)$$

Parallel Coordinate Descent

Input: Training data matrix $A \in \mathbb{R}^{d \times n}$
Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$

for $t = 1, 2, \ldots$ **do**
  parfor $j \in$ RandomPermutation(n) **do**
    Find δ minimizing $f(\mathbf{v} + A_{:,j}\delta) + g_j(\alpha_j + \delta)$
    $\alpha_j \leftarrow \alpha_j + \delta$
    $\mathbf{v} \leftarrow \mathbf{v} + \delta A_{:,j}$
  end parfor
end for

Parallel Coordinate Descent

$$\min_{\boldsymbol{\alpha}} f(A\boldsymbol{\alpha}) + \sum_i g_i(\alpha_i)$$

Parallel Coordinate Descent

Input: Training data matrix $A \in \mathbb{R}^{d \times n}$
Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$

for $t = 1, 2, \ldots$ **do**
  parfor $j \in$ RandomPermutation(n) **do**
    Find δ minimizing $f(\mathbf{v} + A_{:,j}\delta) + g_j(\alpha_j + \delta)$
    $\alpha_j \leftarrow \alpha_j + \delta$
    $\mathbf{v} \leftarrow \mathbf{v} + \delta A_{:,j}$
  end parfor
end for

Parallel Coordinate Descent

$$\min_{\boldsymbol{\alpha}} f(A\boldsymbol{\alpha}) + \sum_i g_i(\alpha_i)$$

System-level bottlenecks:

1. Inefficient cache accesses
2. Write-contention on **v**
3. Scalability across NUMA nodes

Parallel Coordinate Descent

Input: Training data matrix $A \in \mathbb{R}^{d \times n}$
Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$
for $t = 1, 2, \ldots$ **do**
  parfor $j \in$ RandomPermutation(n) **do**
    Find δ minimizing $f(\mathbf{v} + A_{:,j}\delta) + g_j(\alpha_j + \delta)$
    $\alpha_j \leftarrow \alpha_j + \delta$
    $\mathbf{v} \leftarrow \mathbf{v} + \delta A_{:,j}$
  end parfor
end for

Parallel Coordinate Descent

$$\min_{\boldsymbol{\alpha}} f(A\boldsymbol{\alpha}) + \sum_i g_i(\alpha_i)$$

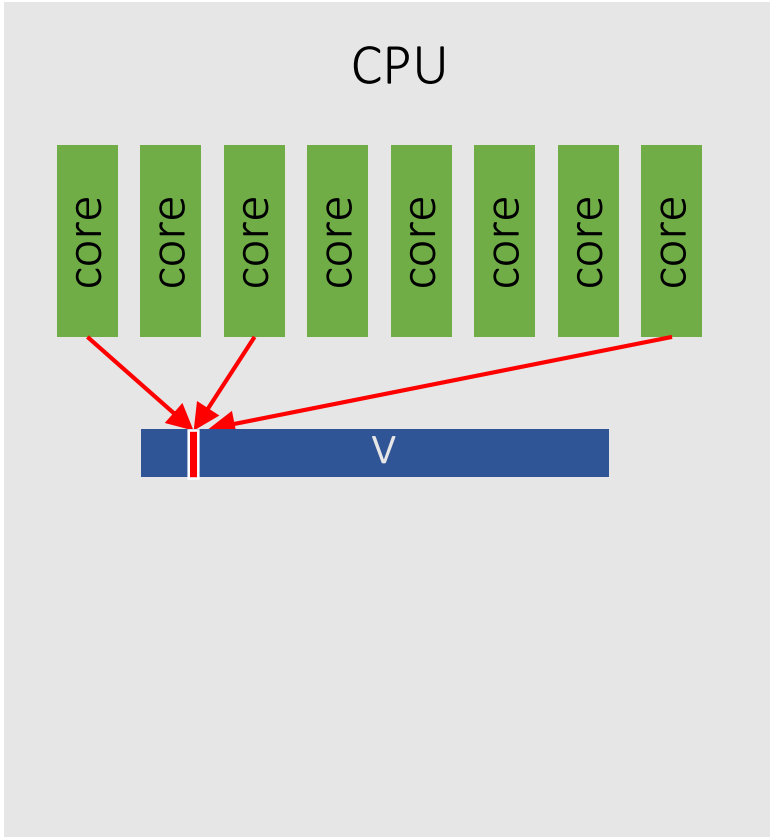
System-level bottlenecks:

1. Inefficient cache accesses
2. Write-contention on **v**
3. Scalability across NUMA nodes

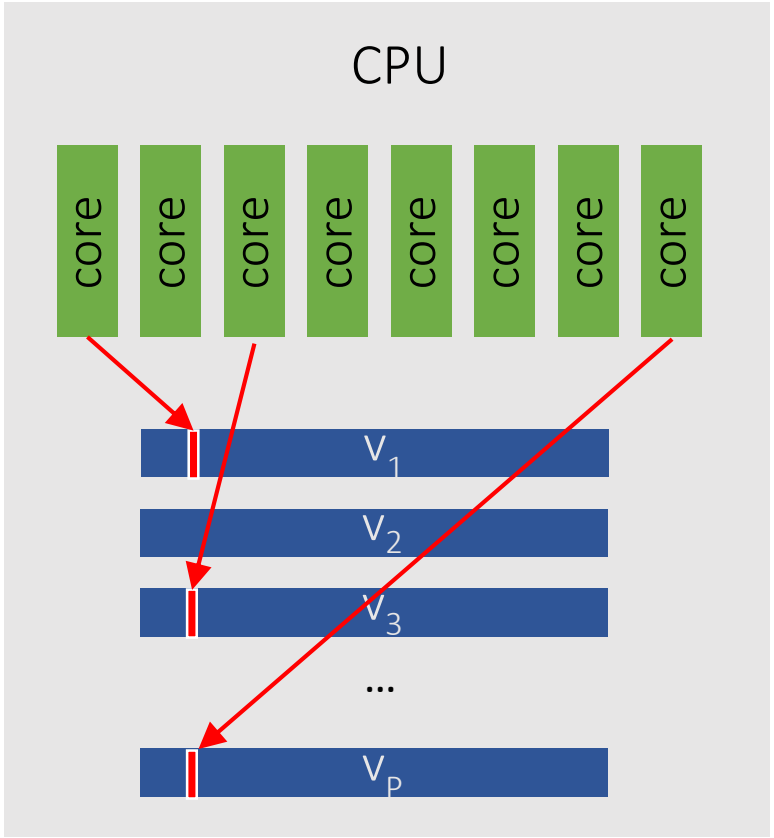
Parallel Coordinate Descent

Input: Training data matrix $A \in \mathbb{R}^{d \times n}$
Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$
for $t = 1, 2, \ldots$ **do**
  parfor $j \in$ RandomPermutation(n) **do**
    Find δ minimizing $f(\mathbf{v} + A_{:,j}\delta) + g_j(\alpha_j + \delta)$
    $\alpha_j \leftarrow \alpha_j + \delta$
    $\mathbf{v} \leftarrow \mathbf{v} + \delta A_{:,j}$
  end parfor
end for

Resolving write-contention on **v**



Resolving write-contention on **v**



Resolving write-contention on **v**

→ replicate **v** across threads

Parallel Coordinate Descent

1: **Input:** Training data matrix $A \in \mathbb{R}^{d \times n}$
Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$
2: **for** $t = 1, 2, \ldots$ **do**

4: **parfor** $j \in$ RandomPermutation(n) **do**
5: Find δ minimizing $f(\mathbf{v} + A_{:,j}\alpha_j) + g_j(\alpha_j + \delta)$
6: $\alpha_j \leftarrow \alpha_j + \delta$
7: $\mathbf{v} \leftarrow \mathbf{v} + \delta A_{:,j}$
8: **end parfor**

10: **end for**

Resolving write-contention on **v**

→ replicate **v** across threads

Parallel Coordinate Descent

Input: Training data matrix $A \in \mathbb{R}^{d \times n}$
Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$

for $t = 1, 2, \ldots$ **do**

threads

  $\mathbf{v}_p \leftarrow \mathbf{v} \quad \forall p \in [P]$

  parfor $j \in \text{RANDOMPERMUTATION}(n)$ **do**

    Find δ minimizing $\hat{f}(\mathbf{v}_p, A_{:,j}, \alpha_j) + g_j(\alpha_j + \delta)$

    $\alpha_j \leftarrow \alpha_j + \delta$

    $\mathbf{v}_p \leftarrow \mathbf{v}_p + \delta A_{:,j}$

  end parfor

  $\mathbf{v} \leftarrow \sum_p \mathbf{v}_p$

end for

Resolving write-contention on **v**

→ replicate **v** across threads

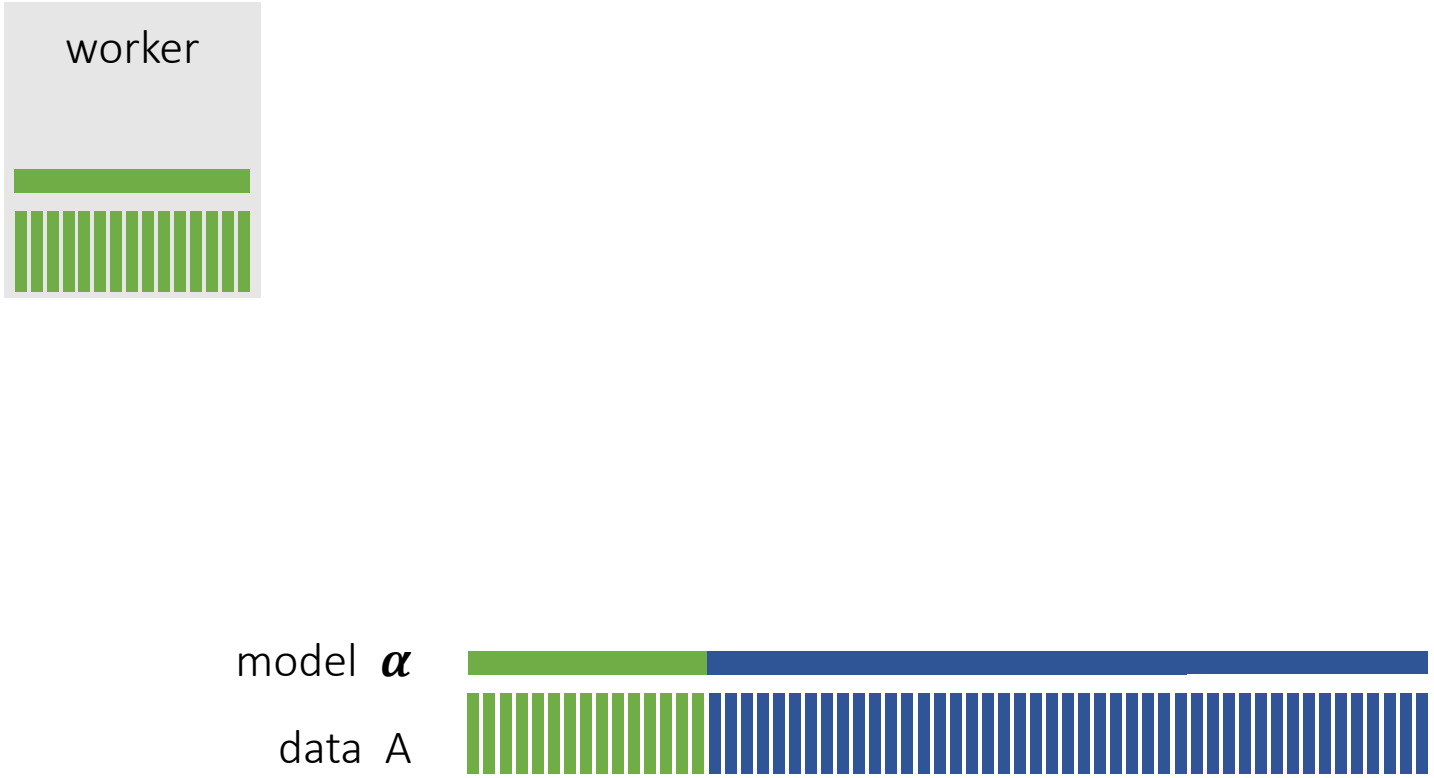
Parallel Coordinate Descent

Input: Training data matrix $A \in \mathbb{R}^{d \times n}$
Initial model $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$

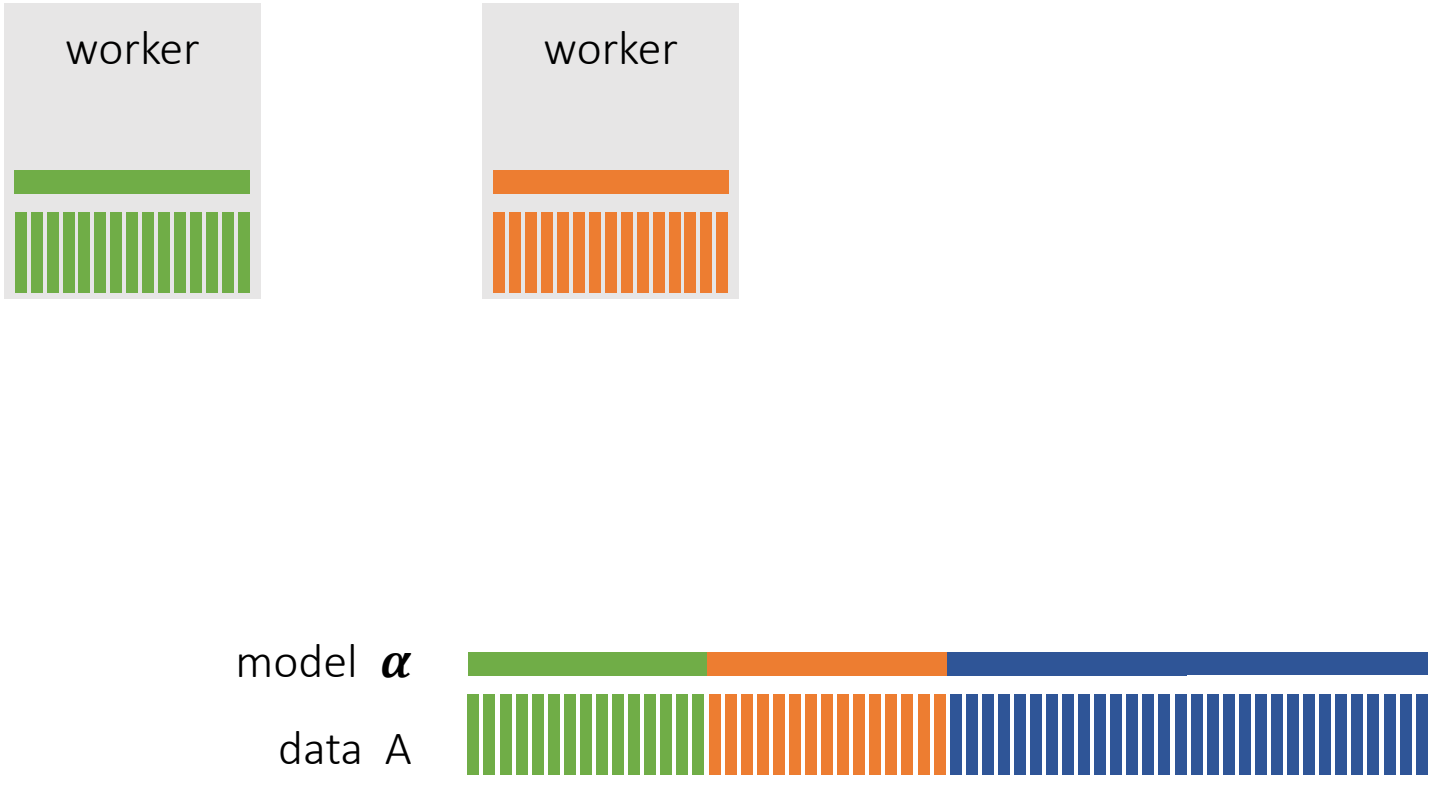
for $t = 1, 2, \ldots$ **do**
  $\mathbf{v}_p \leftarrow \mathbf{v} \quad \forall p \in [P]$ ← # threads
  parfor $j \in \text{RandomPermutation}(n)$ **do**
    Find δ minimizing $\hat{f}(\mathbf{v}_p, A_{:,j}, \alpha_j) + g_j(\alpha_j + \delta)$
    $\alpha_j \leftarrow \alpha_j + \delta$
    $\mathbf{v}_p \leftarrow \mathbf{v}_p + \delta A_{:,j}$
  end parfor
  $\mathbf{v} \leftarrow \sum_p \mathbf{v}_p$
end for

auxiliary model inspired by CoCoA [Smith'18]

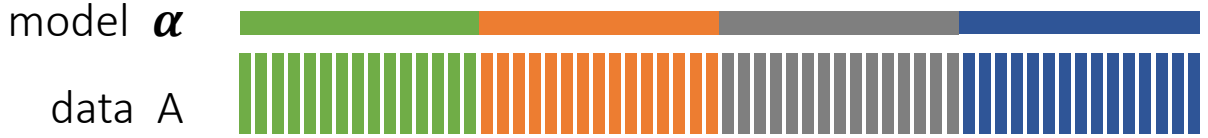
Connection to Distributed Methods



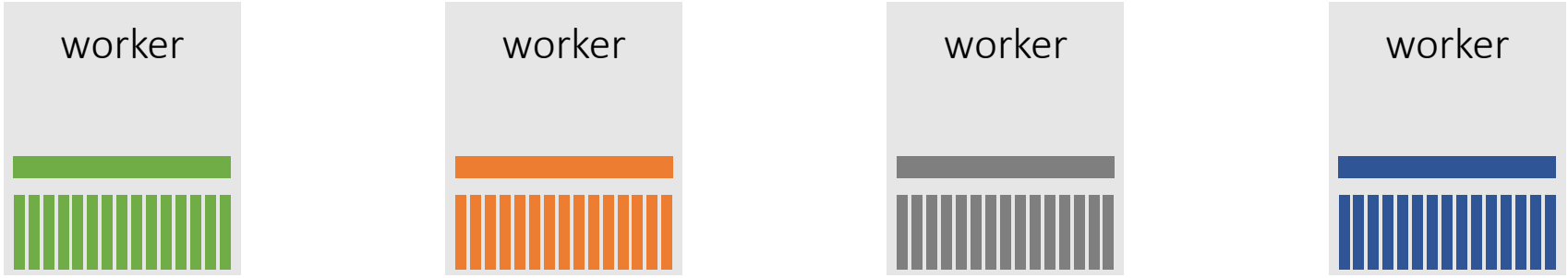
Connection to Distributed Methods

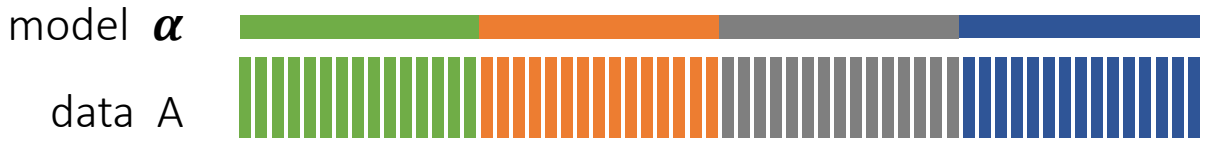


Connection to Distributed Methods

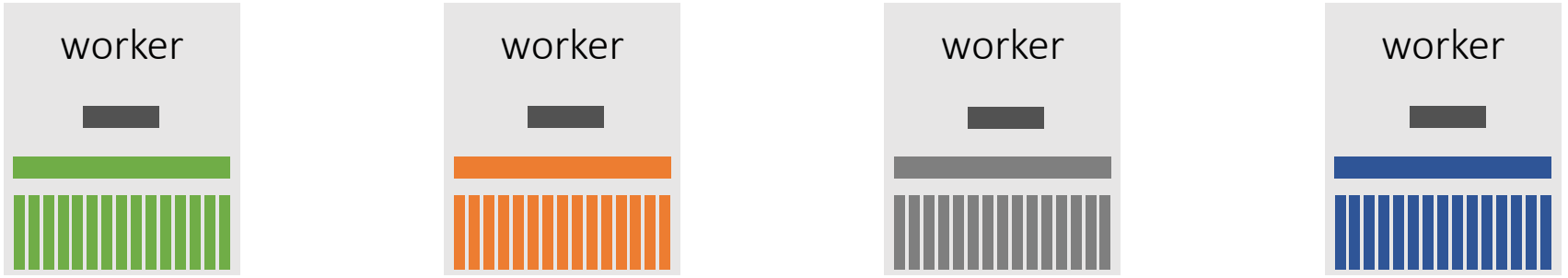


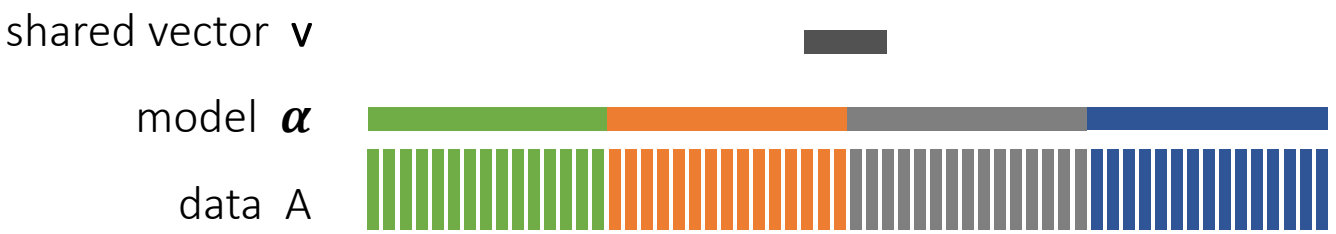
Connection to Distributed Methods



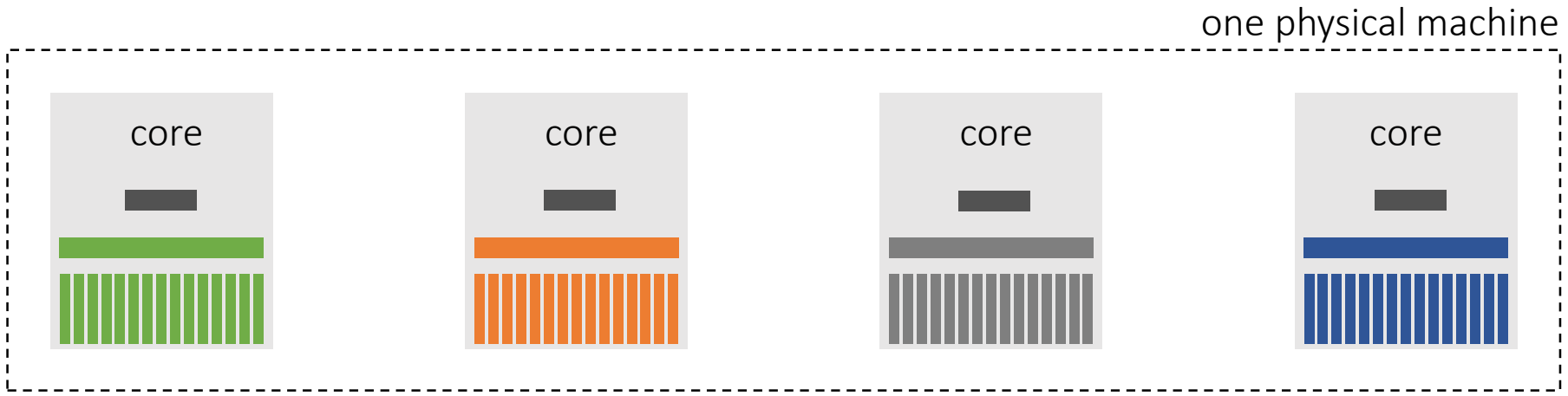


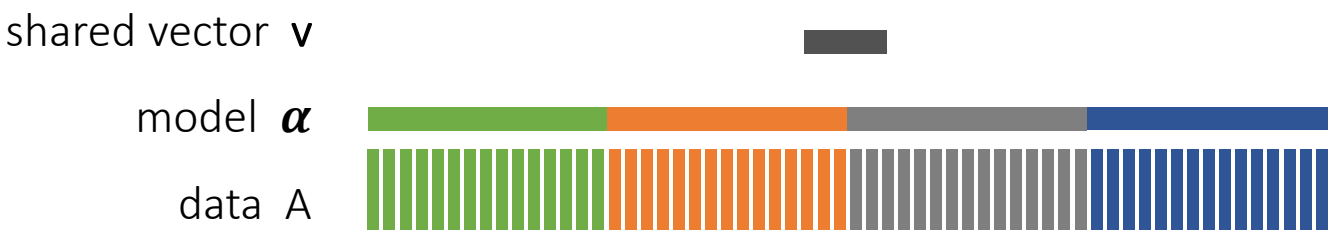
Connection to Distributed Methods



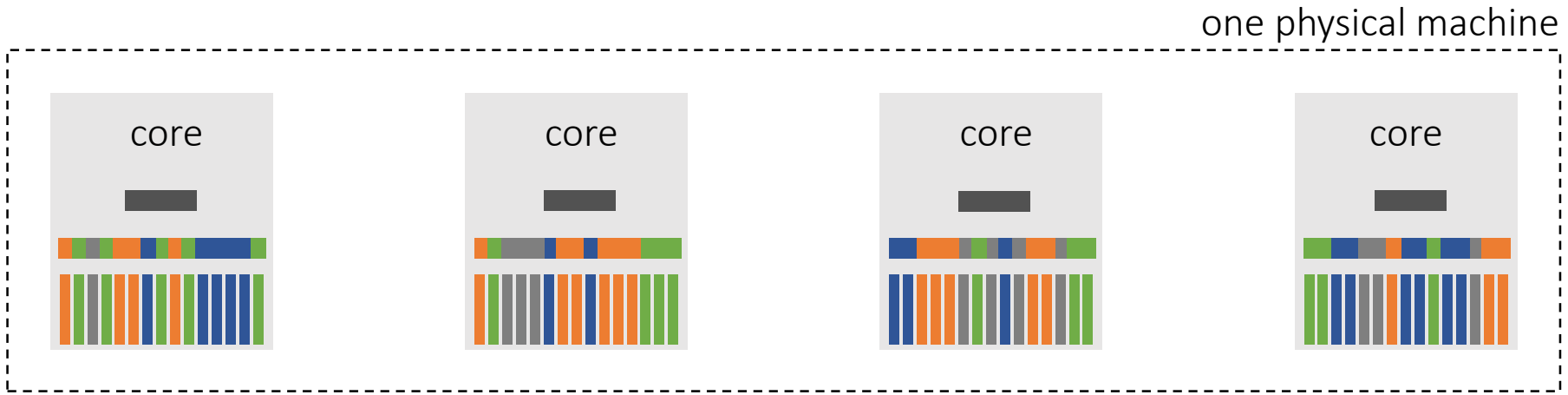


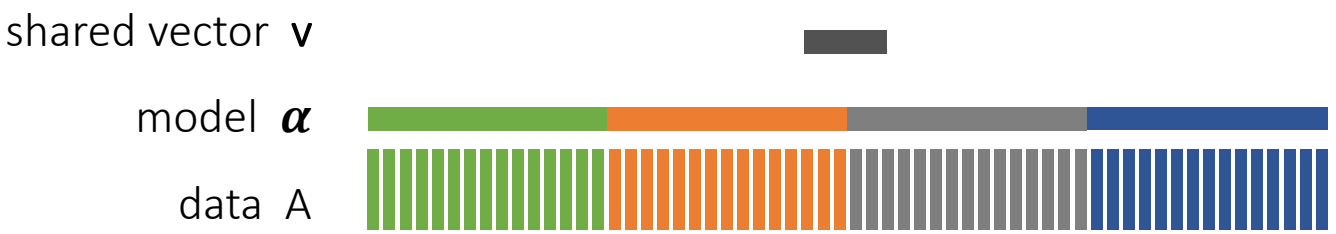
Connection to Distributed Methods



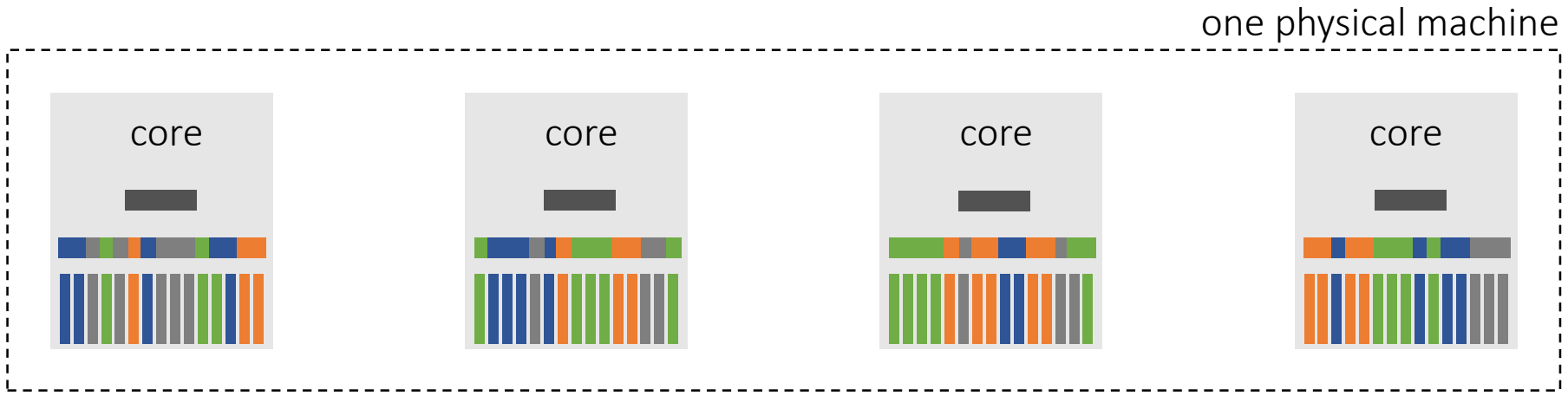


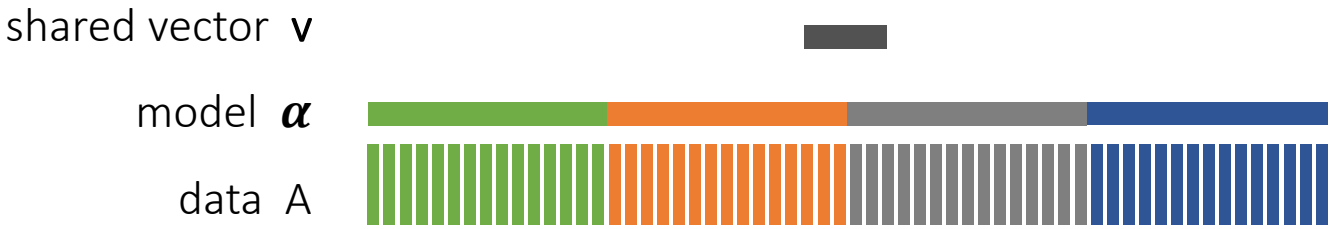
Repartitioning





Repartitioning





System-Aware Coordinate Descent (SySCD)

- Combination of distributed methods with repartitioning
  - ✓ high implementation efficiency
  - ✓ theoretically sound parallel method
  - ✓ scales to large degrees of parallelism

System-Aware Coordinate Descent (SySCD)

- Combination of distributed methods with repartitioning
  - ✓ high implementation efficiency
  - ✓ theoretically sound parallel method
  - ✓ scales to large degrees of parallelism
- Additional optimizations (not covered in this talk)
  - ✓ NUMA - affinity
  - ✓ alignment with cache access pattern

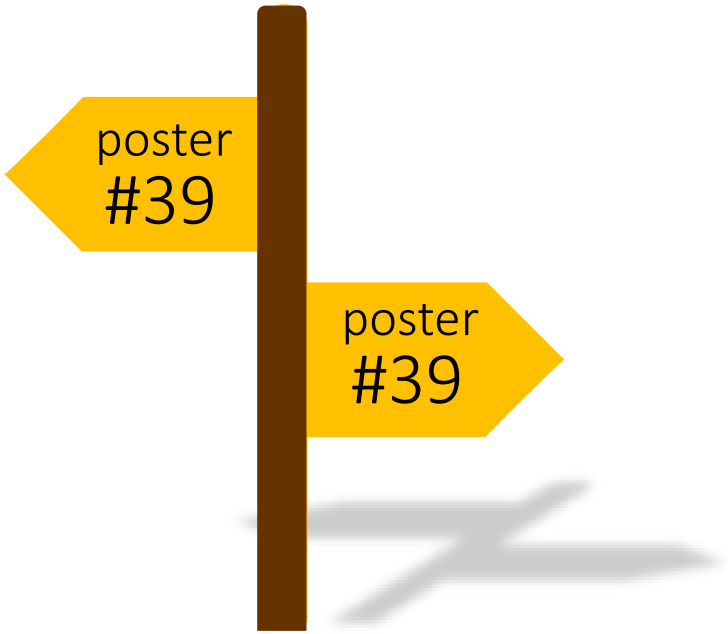
System-Aware Coordinate Descent (SySCD)

- Combination of distributed methods with repartitioning
  - ✓ high implementation efficiency
  - ✓ theoretically sound parallel method
  - ✓ scales to large degrees of parallelism
- Additional optimizations (not covered in this talk)
  - ✓ NUMA - affinity
  - ✓ alignment with cache access pattern

➡ > 10x faster than sate-of-the-art asynchronous CD methods

System-Aware Coordinate Descent (SySCD)

- Combination of distributed methods with repartitioning
  - ✓ high implementation efficiency
  - ✓ theoretically sound parallel method
  - ✓ scales to large degrees of parallelism
- Additional optimizations (not covered in this talk)
  - ✓ NUMA - affinity
  - ✓ alignment with cache access pattern



➡ > 10x faster than sate-of-the-art asynchronous CD methods

IBM Berkeley UNIVERSITY OF CALIFORNIA