# Asynchronous Distributed Machine Learning

Hagit Attiya and Noa Schiller (Technion)

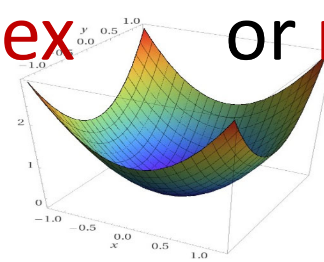
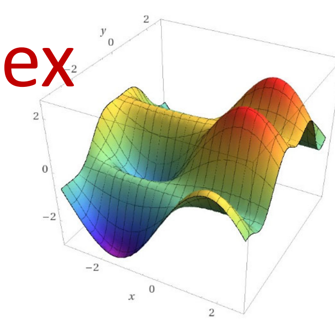
# Distributed Optimization

Processes access the same data distribution $D$ and loss function $\ell: \mathbb{R}^d \times D \to \mathbb{R}$

Cost function at $\mathrm{x} \in \mathbb{R}^{\mathrm{d}}$ is $Q(x) \triangleq \mathbb{E}_{z \sim D}[\ell(x, z)]$

☞Minimize $Q$ to find $x^* \in \underset{x \in \mathbb{R}^d}{\operatorname{argmin}}\, Q(x)$

$Q$ is differentiable and smooth

Can be either (strongly) convex  or non-convex 

# Stochastic Gradient Descent (SGD)

Stochastic gradient $G(x, z)$ estimates the true gradient $\nabla Q(x)$ using a random data point $z \in D$

**For** $t = 1, 2, \ldots$

**1. Draw a random data point** $z \overset{i.i.d}{\sim} D$

**2. Update** $x_{t+1} = x_t - \eta_t G(x_t, z)$

Learning rate ($\eta_t$)

Estimates are unbiased: $\mathbb{E}_{z \sim D}[G(x, z)] = \nabla Q(x)$

with bounded variance: $\mathbb{E}_{z \sim D}[\|G(x, z) - \nabla Q(x)\|_2] \leq \sigma$

# Mini-Batch SGD

Faster convergence by sampling several data points

**For** $t = 1, 2, \ldots$

1. **Draw** $M$ **random data points** $z_1, \ldots, z_M \overset{i.i.d}{\sim} D$
2. **Update** $x_{t+1} = x_t - \frac{\eta_t}{M} \sum_{i=1}^{M} G(x_t, z_i)$

# Simple Distributed Mini-Batch SGD

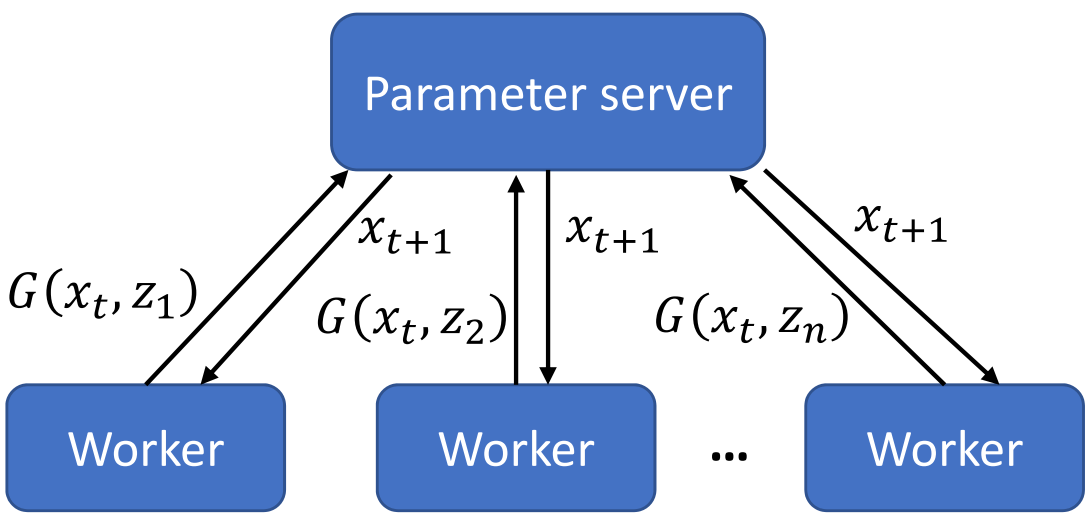
Centralized scheme requires synchronization

Parameter server is a single point-of-failure

**For** $t = 1, 2, \ldots$

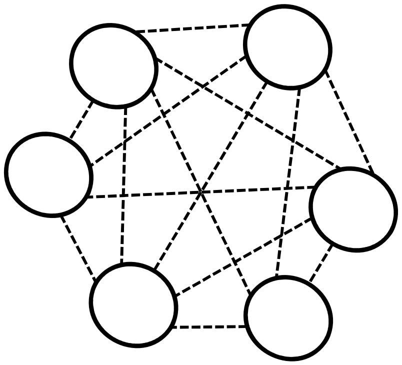
**1. Draw** $M$ **random data points** $z_1, \ldots, z_M \overset{i.i.d}{\sim} D$

**2. Update** $x_{t+1} = x_t - \frac{\eta_t}{M} \sum_{i=1}^{M} G(x_t, z_i)$



# Decentralized SGD

Fully-connected set of $n$ nodes



**For** $t = 1, 2, \ldots$

**1. A node draws a random data point** $\overset{i.i.d}{\sim} D$
**& computes new gradient**

**2. Get gradients from M nodes & update**

How good is this?

# Strongly Convex Q ⇨ "External" Convergence

Single minimum $x^*$ obtained at a unique point

For any $M$, any round $T$, and any node $i$

$$\mathbb{E}\left[\left\|x_T^i - x^*\right\|_2^2\right] \leq O\left(\frac{\left\|x^1 - x^*\right\|_2^2}{MT} + \frac{\sigma^2}{MT}\right)$$

Same convergence rate as sequential mini-batch

SGD, with batch size M

Implies external convergence $\mathbb{E}\left[\left\|x^i - x^*\right\|_2^2\right] \leq \epsilon$

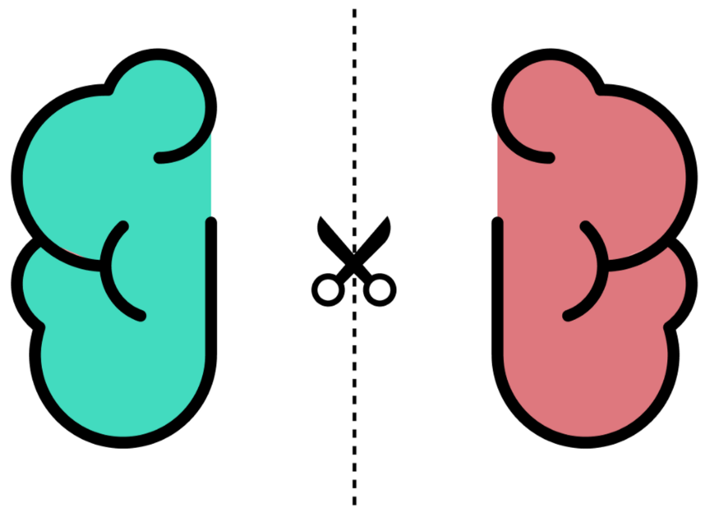
# Strongly Convex Q ⇨ "External" Convergence

Single minimum $x^*$ obtained at a unique point

For any $M$ ⇨ the algorithm withstands partitioning

I.e., converges even when communicating with only a minority

No "split-brain"



# Non Strongly Convex Functions Require a Majority

For some non-convex cost function Q, algorithm does not converge if a majority of processes fail

$$\max_{i,j} \mathbb{E}\left[\left\|\text{output}^i - \text{output}^j\right\|_2\right] > \delta$$

(No internal convergence)

Proof more complicated than expected and relies on probabilistic indistinguishability

[Goren, Moses & Spiegelman, DISC 2021]

# General Algorithm

Convergence rate similar to sequential algorithm

[Ghadimi,Lan, 2013]

Analysis is a simplified version of

[ElMhamdi,Farhadkhani,Guerraoui,Guirguis,Hoang,Rouault,NeuroIPS 2021]

**For iteration** $t = 1, \ldots, T$**:**

1. **Compute the stochastic gradient** $g_t^i$ **at** $x_t^i$
2. $x_t^i \leftarrow x_t^i - \eta_t \cdot g_t^i$
3. **Send** $\langle t, x_t^i \rangle$
4. **Wait to receive** $\geq M$ **iteration-**$t$ **messages**
5. $x_{t+1}^i \leftarrow \text{Avg}($**recieved models**$)$

# General Algorithm

But with an additive factor of $\Delta$, which is reduced using multi-dimensional approximate agreement

This algorithm needs communication with a majority

**For iteration** $t = 1, \ldots, T$**:**

1. **Compute the stochastic gradient** $g_t^i$ **at** $x_t^i$
2. $x_t^i \leftarrow x_t^i - \eta_t \cdot g_t^i$
3. **Send** $\langle t, x_t^i \rangle$
4. **Wait to receive** $\geq M$ **iteration-**$t$ **messages**
5. $x_{t+1}^i \leftarrow \text{Avg}(\text{recieved models})$

# Multi-Dimensional Approximate Agreement

A process starts with input $x^i \in \mathbb{R}^{\mathrm{d}}$ and returns $y^i \in \mathbb{R}^{\mathrm{d}}$, such that the outputs are

- In the convex hull of the inputs

- Contracted by a factor of $q$ relative to the inputs

$$\max_{i,j} \left\| y^i - y^j \right\|_2^2 \leq q \max_{i,j} \left\| x^i - x^j \right\|_2^2$$

[Mendes, Herlihy, Vaidya, Garg, DC 2015]
[Fugger, Nowak, DISC 2018]

# Convergence of General Algorithm

With an appropriate q, we get internal convergence

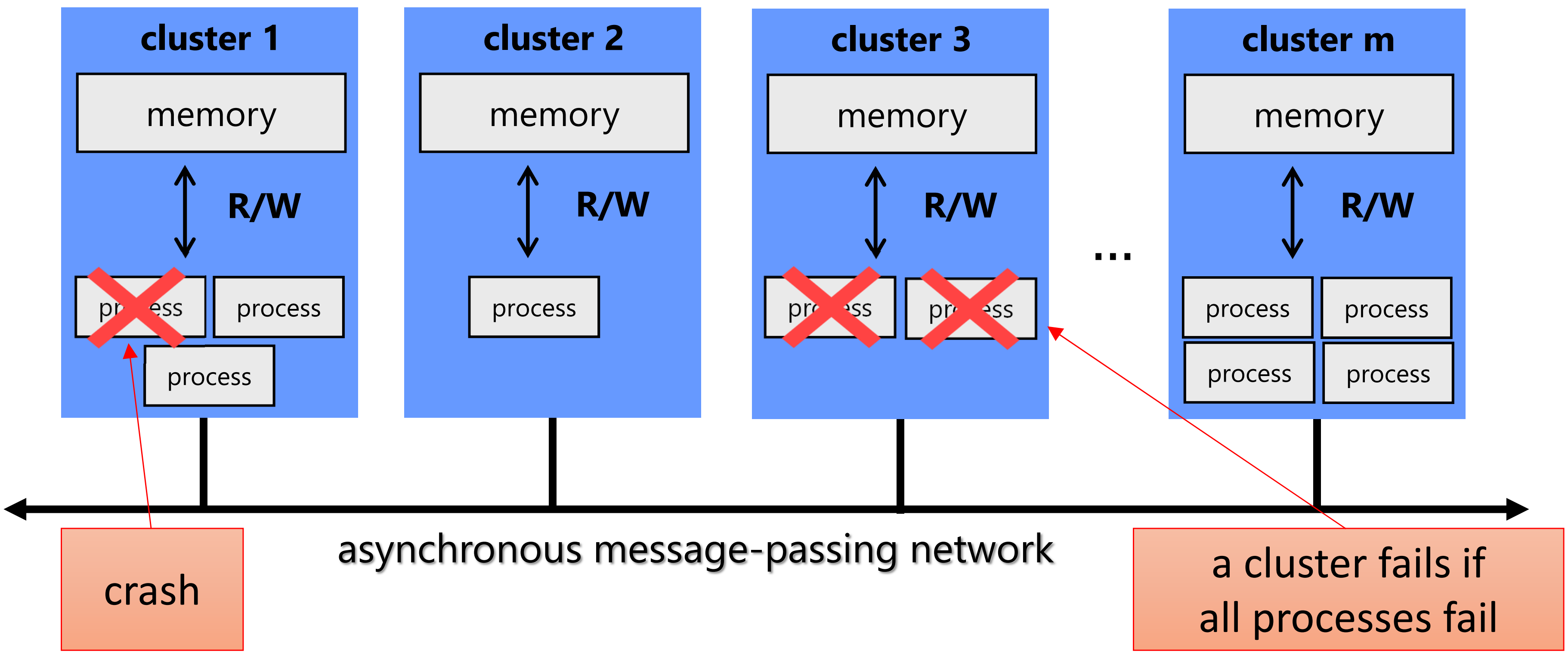
$$\max_{i,j} \mathbb{E}\left\|x^i - x^j\right\|_2 < \delta$$

Can also show external convergence

☝Better (1-dimension) AA when shared-memory is used

[A,Kumari,Schiller,OPODIS 2020]

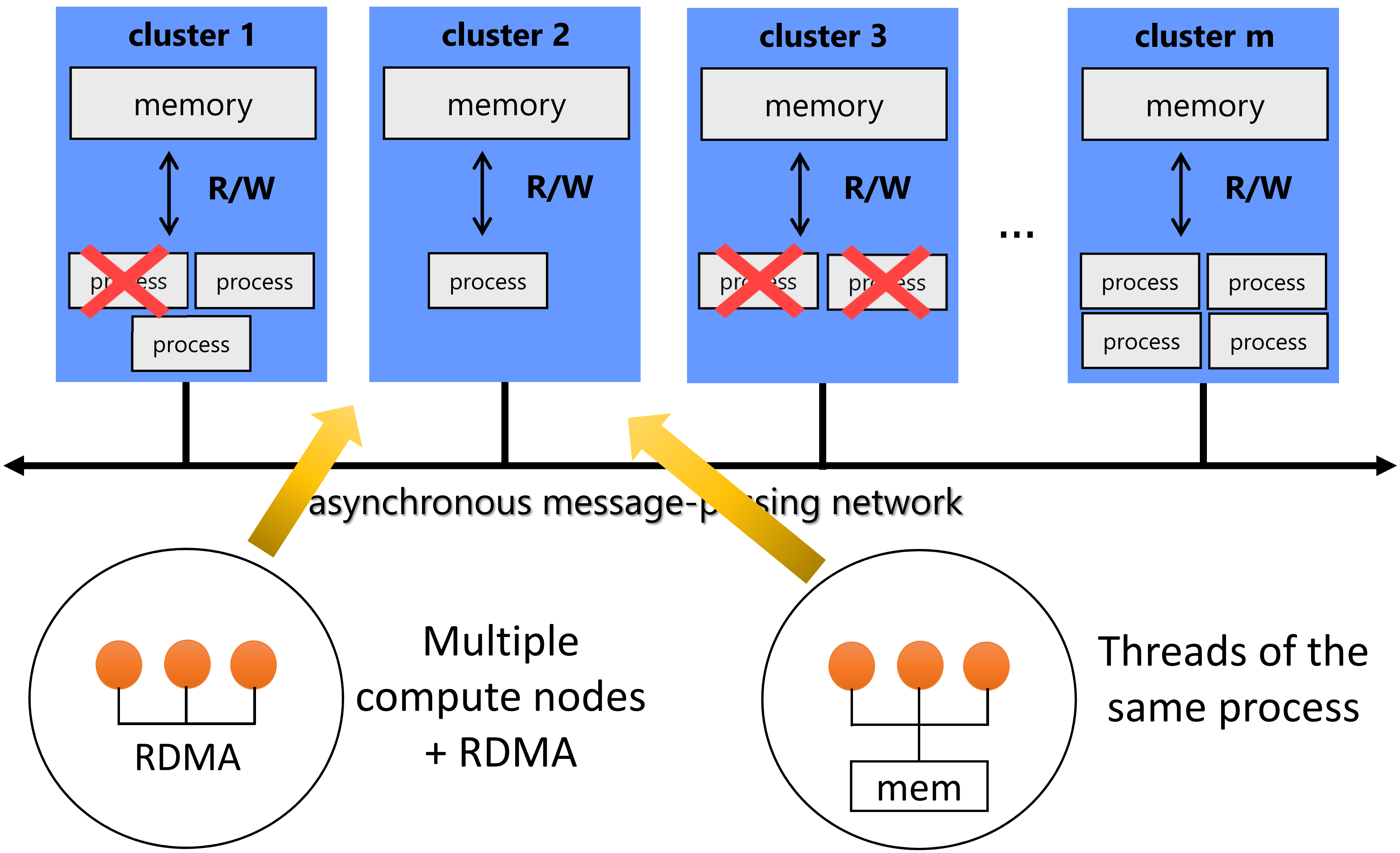
# Cluster-Based Model




- Disjoint clusters, each with shared read/write registers
- All processes can send asynchronous messages to each other

[Raynal, Cao,ICDCS 2019]

# Cluster-Based Model for HPC



# Multi-Dimensional AA in the Cluster-Based Model

```
For round r = 1 … R:
1. z_r = ClusterApproximateAgreement(y_r)
2. Send ⟨r, z_r⟩ to all processes
3. Wait to receive round r messages representing
             a majority of processes
4. y_{r+1} = Aggregate(received values)
```

# Multi-Dimensional AA in the Cluster-Based Model

A process represents all processes in its cluster

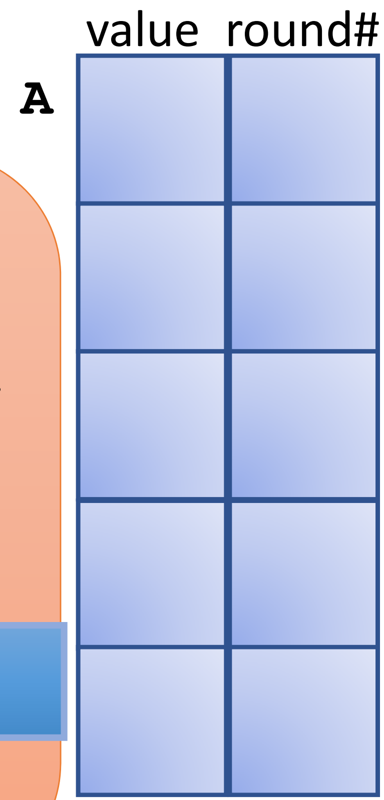
```
For round r = 1 … R:
1. z_r = ClusterApproximateAgreement(y_r)
2. Send ⟨r, z_r⟩ to all processes
3. Wait to receive round r messages representing
            a majority of processes
4. y_{r+1} = Aggregate(received values)
```

Better contraction inside a cluster

Tune to get $q$-contraction in O(log $q$) rounds

# MDAA within a Cluster

An array **A** of ⟨value, round#⟩ for each cluster



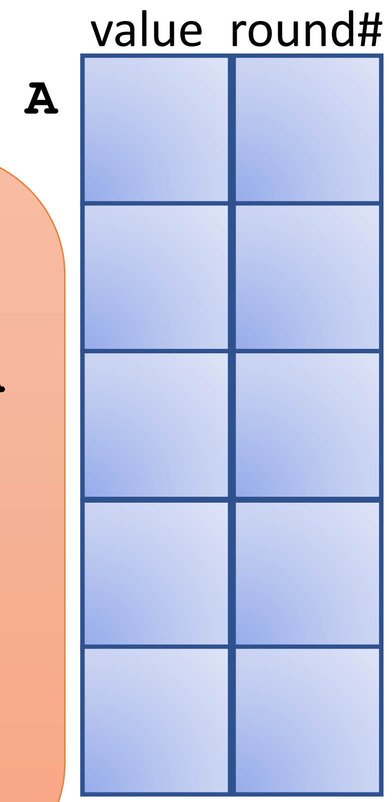

```
r ← 1 ; A[i] ← ⟨x,r⟩
while r < constant do
    let r_max be the largest round number in A
    if r = r_max then
        X ← values in A with round r_max
        A[i] ← ⟨MidExtremes(X),r+1⟩
        r ← r + 1
    else r ← r_max
return some x_j s.t. A[j] = ⟨x_j,r+1⟩
```

Aggregation rule

**MidExtremes** returns the average of the two values realizing the maximum Euclidean distance

# MDAA within a Cluster

An array **A** of ⟨value, round#⟩ for each cluster




```
r ← 1 ; A[i] ← ⟨x,r⟩
while r < constant do
   let r_max be the largest round number in A
   if r = r_max then
      X ← values in A with round r_max
      A[i] ← ⟨MidExtremes(X),r+1⟩
      r ← r + 1
   else r ← r_max
return some x_j s.t. A[j] = ⟨x_j,r+1⟩
```

Ensures constant contraction within O(1) rounds

# Skipping

$r \leftarrow 1$ ; $A[i] \leftarrow \langle x, r \rangle$<br>
**while** $r <$ constant **do**<br>
    let $r_{max}$ be the largest round number in A<br>
    **if** $r = r_{max}$ **then**<br>
        $X \leftarrow$ values in A with round $r_{max}$<br>
        $A[i] \leftarrow \langle \text{MidExtremes}(X), r+1 \rangle$<br>
        $r \leftarrow r + 1$<br>
    **else** $r \leftarrow r_{max}$<br>
**return** some $x_j$ s.t. $A[j] = \langle x_j, r+1 \rangle$

skipping

# Skipping

A process can skip to the most advanced iteration instead of going through intermediate iterations

**For round** $r = 1 \ldots R$**:**

1. $z_r =$ **ClusterApproximateAgreement(**$y_r$**)**
2. **Send** $\langle r, z_r \rangle$ **to all processes**
3. **Wait to receive round** $r$ **message from a majority of clusters**
4. $y_{r+1} =$ **AggregationRule(received values)**
5. **If received round** $r'$ **messages,** $r' > r$ **, then skip to round** $r'$

# Recovery through Skipping

Allows recovering process to rejoin the computation

Non-volatile memory can be used to checkpoint the current status

```
For round r = 1 ... R:
1. z_r = ClusterApproximateAgreement(y_r)
2. Send ⟨r, z_r⟩ to all processes
3. Wait to receive round r message from
            a majority of clusters
4. y_{r+1} = AggregationRule(received values)
5. If received round r' messages, r' > r ,
            then skip to round r'
```

# Some Related Work

Other work does not ignore (stale) parameters from previous iterations

[Li,Ben-Nun,Di Girolamo,Alistarh,Torsten Hoefler,PPoPP 2020]

[Li,Ben-Nun,Di Girolamo, Dryden,Alistarh,Torsten Hoefler,TPDS 2021]

Elastic consistency bounds the staleness

[Nadiradze,Markov,Chatterjee,Kungurtsev,Alistarh, AAAI 2021]

Our MDAA algorithm "beats" a $f(d+2)$-redundancy lower bound for Byzantine failures

[Mendes,Herlihy,Vaidya,Garg, DC 2015]

$2f$-redundancy is a necessary and sufficient condition for f-resilient Byzantine optimization

[Su,Vaidya,PODC 2016]