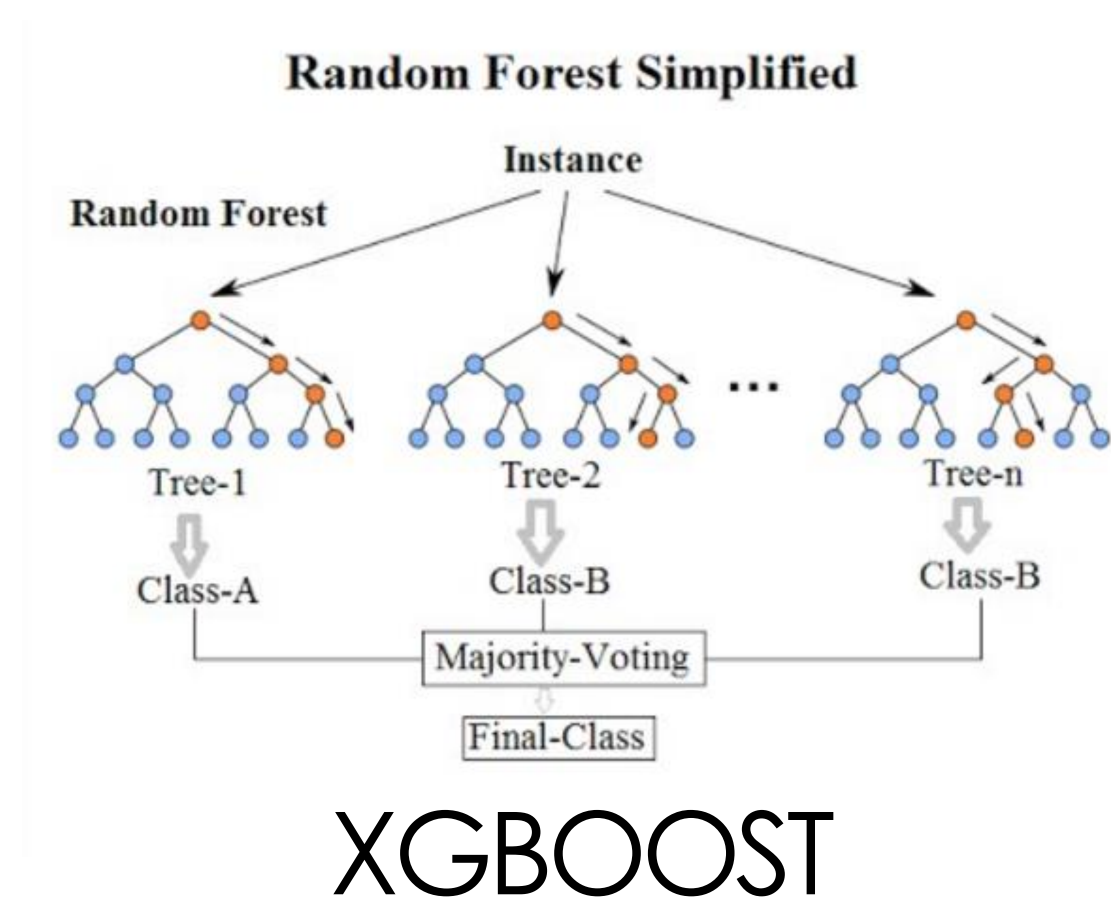
Where We Are

Machine Learning Systems	2012 - Now
Big Data	2010 - Now
Cloud	2000 - 2016
Foundations of Data Systems	1980 - 2000

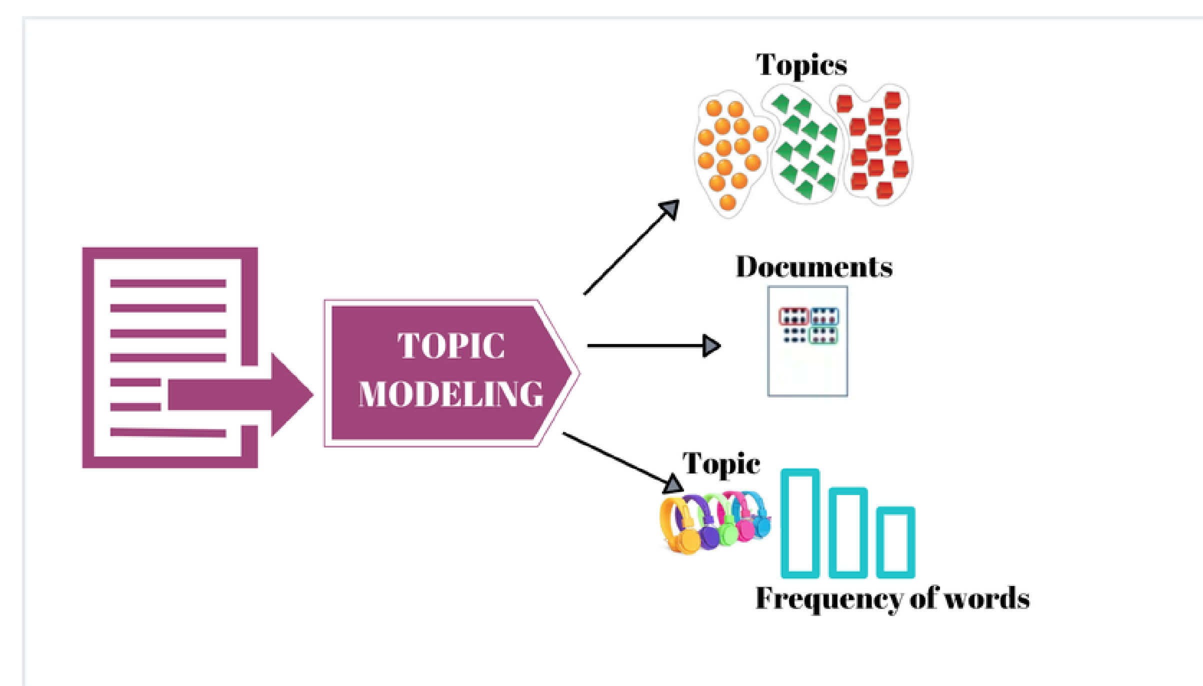
ML Era (roughly starts from 2008, even before Spark has taken off)

- ML was still very diverse (a.k.a. in a mess) in 2012

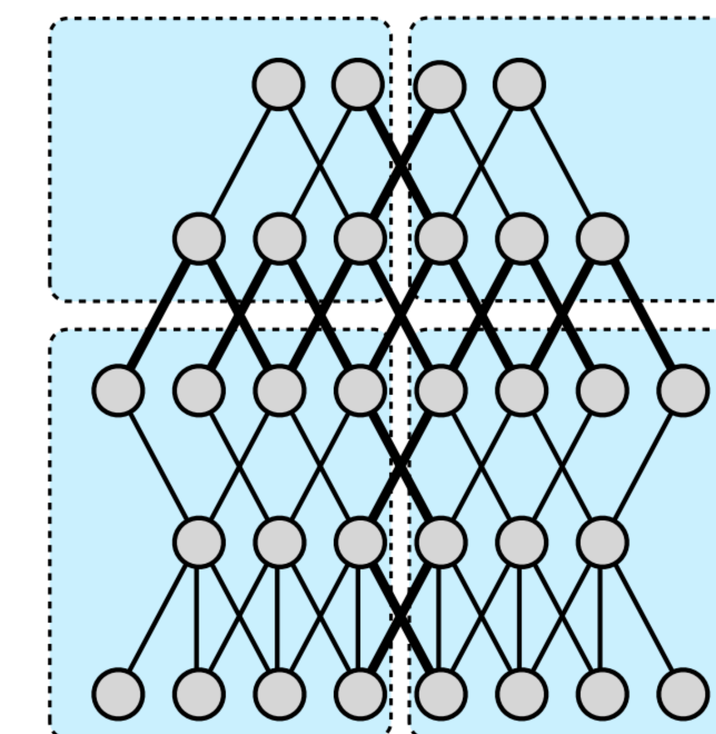


XGBOOST

Spark mllib



LDA



Torch (lua) / Theano / distbelief

Diversity -> Good News or Bad New?

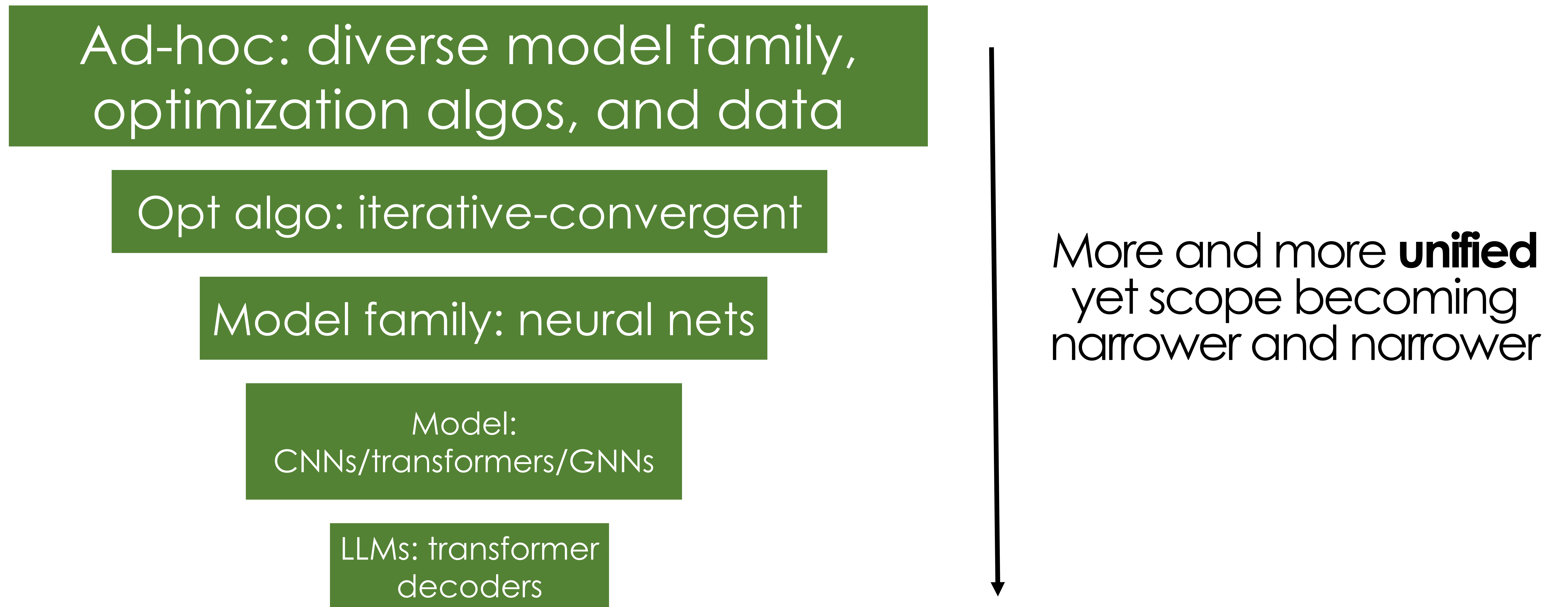
- ML is so diverse
  - Cons:
    - There is no unified model / computation
    - Hard to build a programming model / interface that cover a diverse range of applications
    - No idea where the system bottlenect is
  - Pros:
    - A lot of opportunities: Gold mining era

ML Systems Plan in DSC 204A

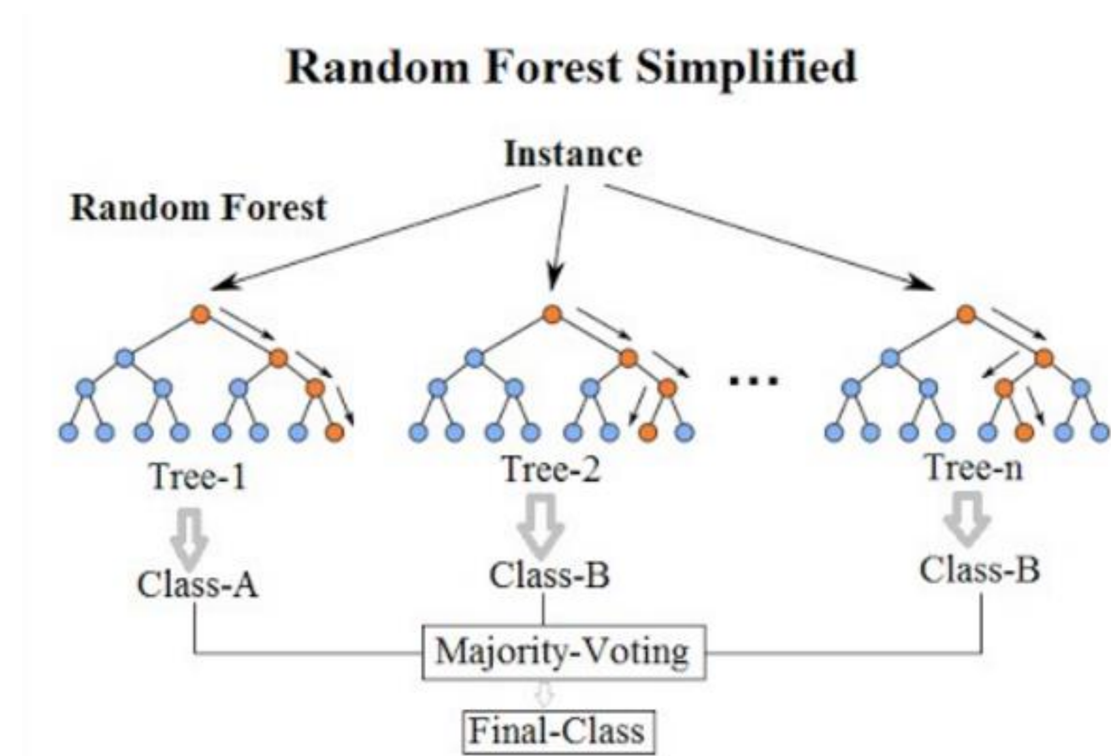
- ML System history: history of unification
- Parameter server
- Autodiff libraries: tensorflow, pytorch, etc.
- LLMs: Flash attention, paged attention, and how to scale up
- Want to dive into each topic? Enroll DSC 299 offered next quarter

ML System history

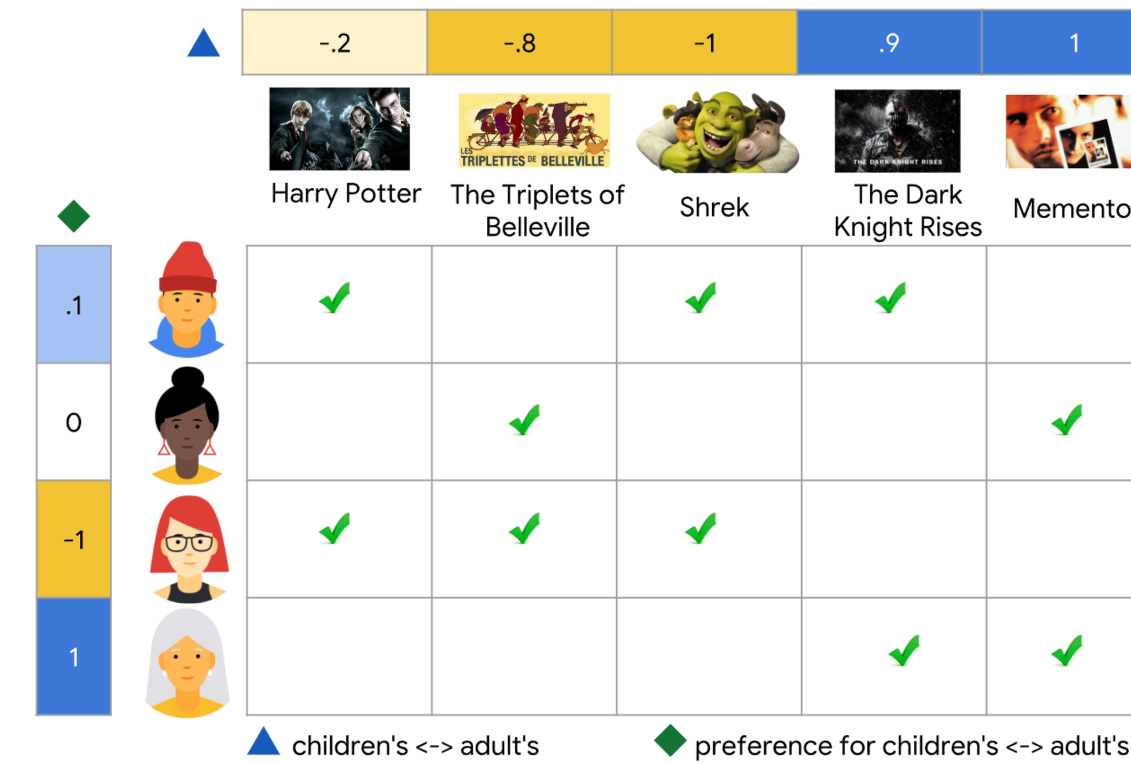
- ML Systems evolve as more and more ML components (models/optimization algorithms) are unified



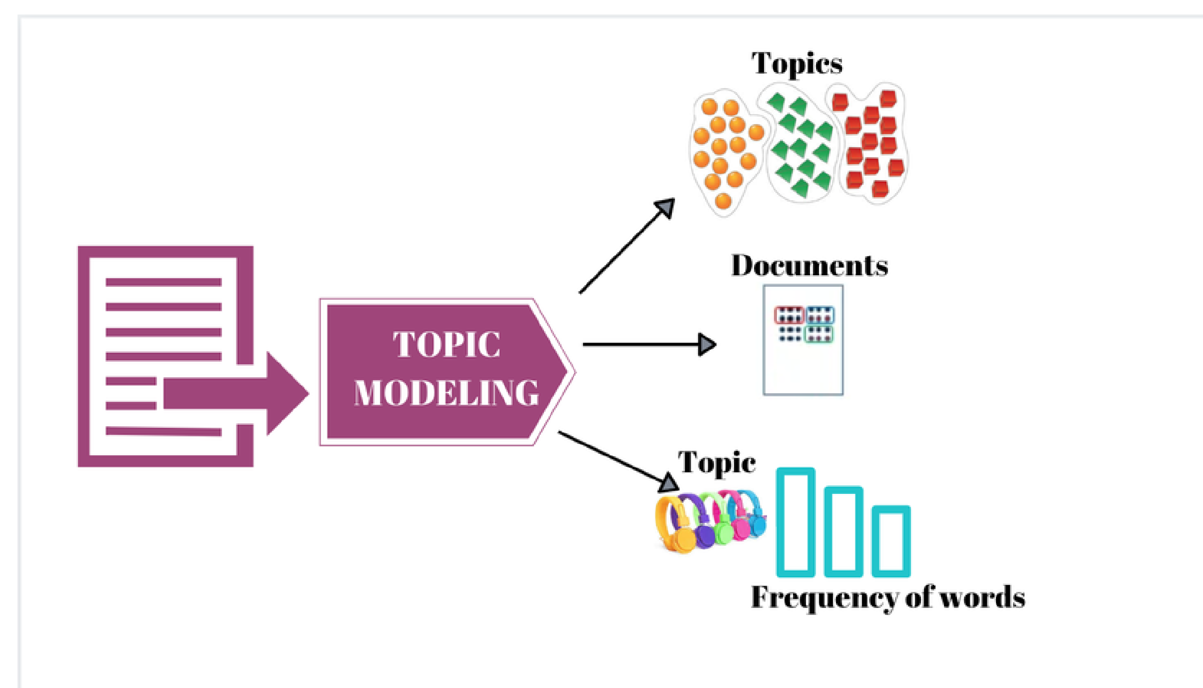
The first Unified component: Iterative-convergence Algo



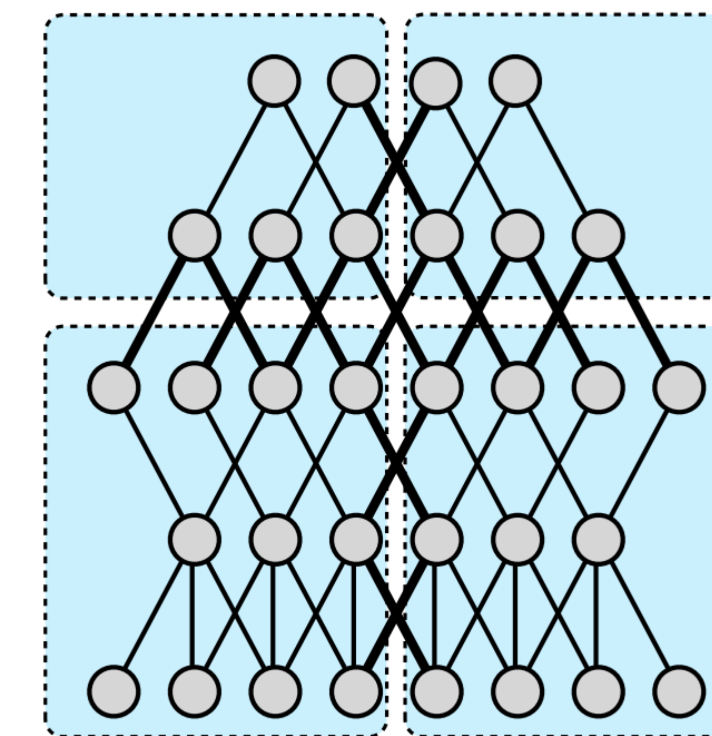
Gradient boosting tree



Coordinate descent



EM Algorithm



Gradient descent

Example: Gradient Descent

Gradient / backward computation

Recall collective communication

$$\theta^{(t)} = \theta^{(t-1)} + \boxed{\varepsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})}$$

objective   data

- The first unification:
  - Most ML algorithms are **iterative-convergent**
  - **iterative-convergent** is the master equation behind

How to Distribute this Equation?

Gradient / backward computation

$$\theta^{(t)} = \theta^{(t-1)} + \boxed{\varepsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})}$$

objective

data

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

How to perform this sum?

Problems if expressing this in Spark

- ML is too diverse; hard to express their computation in coarse-grained data transformations.

$$
\begin{aligned}
map(f : \mathrm{T} \Rightarrow \mathrm{U}) &: \quad \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]} \\
filter(f : \mathrm{T} \Rightarrow \mathrm{Bool}) &: \quad \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]} \\
flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]}) &: \quad \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]} \\
sample(fraction : \mathrm{Float}) &: \quad \mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]} \text{ (Deterministic sampling)} \\
groupByKey() &: \quad \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]} \\
reduceByKey(f : (\mathrm{V}, \mathrm{V}) \Rightarrow \mathrm{V}) &: \quad \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]} \\
union() &: \quad (\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]} \\
join() &: \quad (\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]} \\
cogroup() &: \quad (\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]} \\
crossProduct() &: \quad (\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]} \\
mapValues(f : \mathrm{V} \Rightarrow \mathrm{W}) &: \quad \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]} \text{ (Preserves partitioning)} \\
sort(c : \mathrm{Comparator[K]}) &: \quad \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]} \\
partitionBy(p : \mathrm{Partitioner[K]}) &: \quad \mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}
\end{aligned}
$$

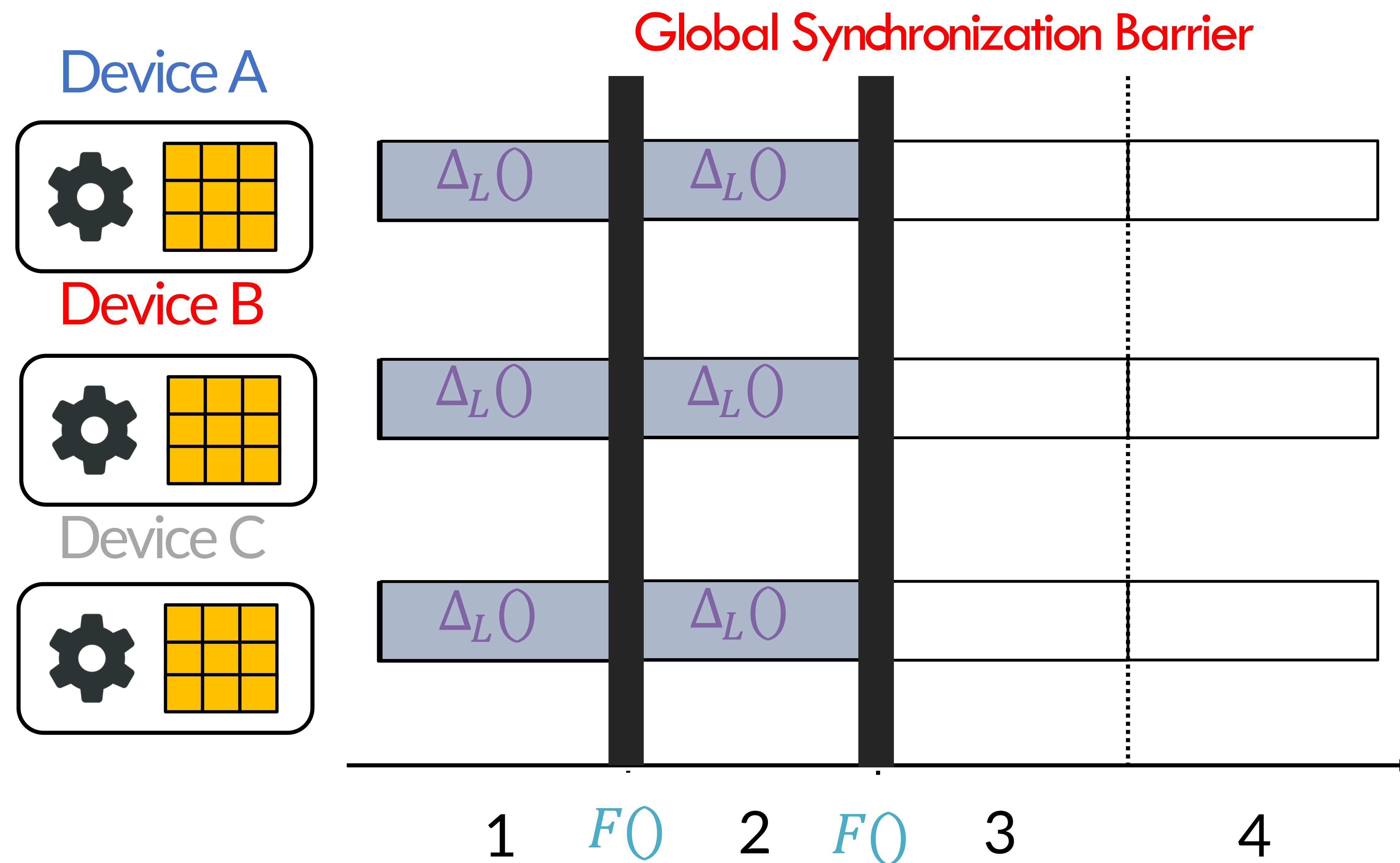
Problems if expressing this in Spark

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

- Very heavy communication per iteration
- Compute : communication = 1:10 in the era of 2012

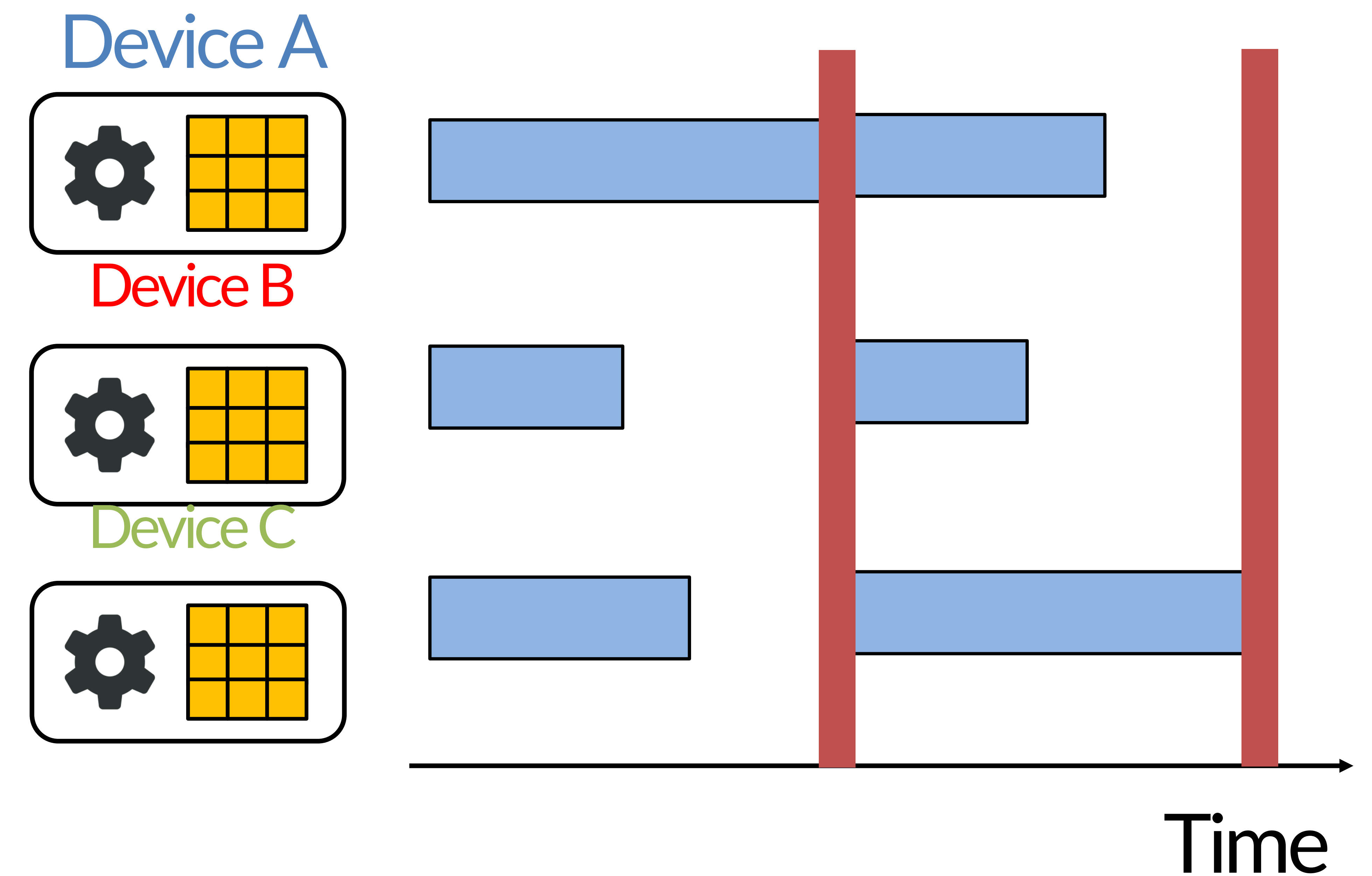
Consistency

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$



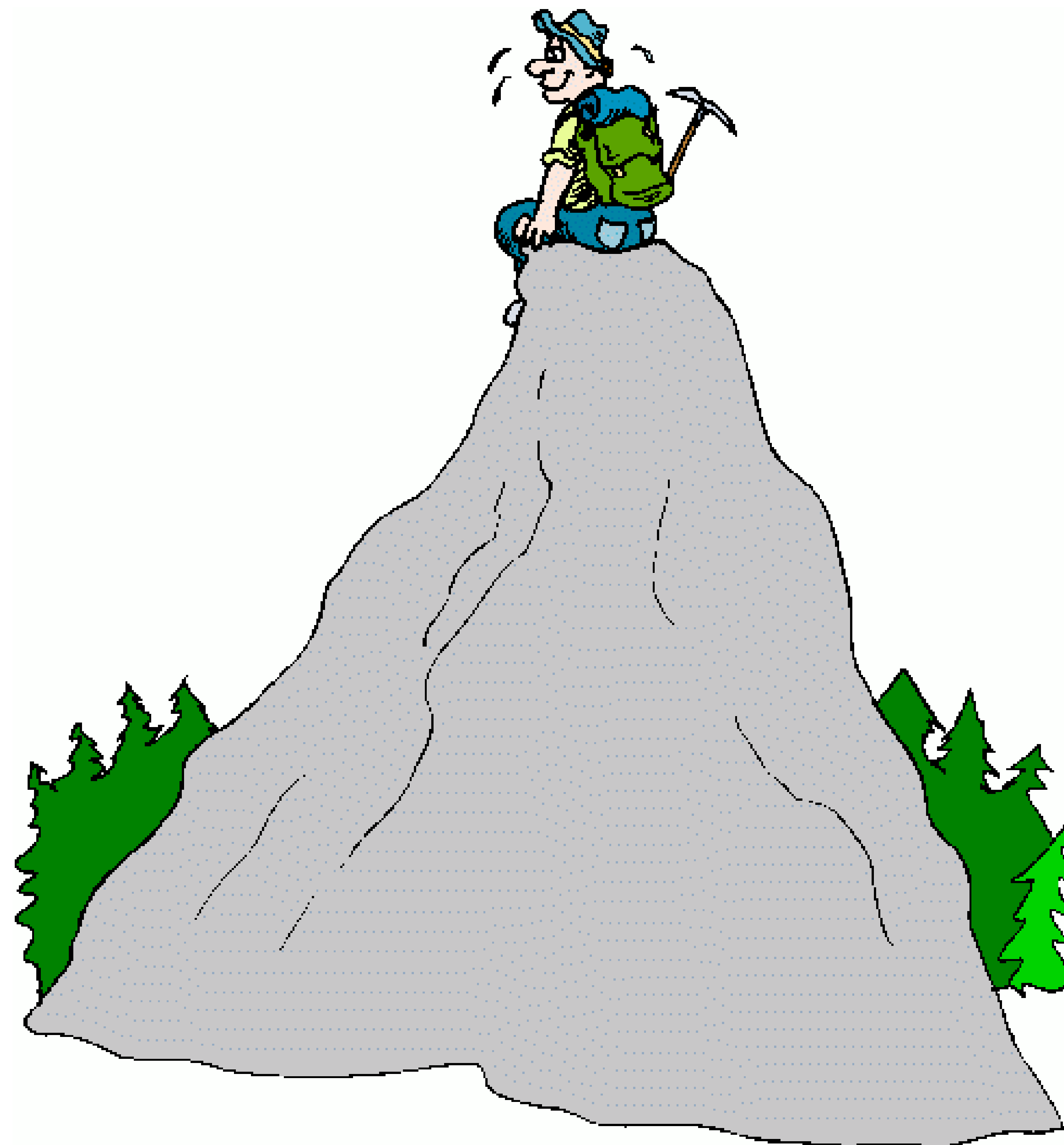
BSP's Weakness: Stragglers

- **BSP suffers from stragglers**
  - Slow devices (stragglers) force all devices to wait
  - More devices → higher chance of having a straggler
- **Stragglers are usually transient, e.g.**
  - Temporary compute/network load in multi-user environment
  - Fluctuating environmental conditions (temperature, vibrations)
- BSP's throughput is greatly decreased in large clusters/clouds, where stragglers are unavoidable

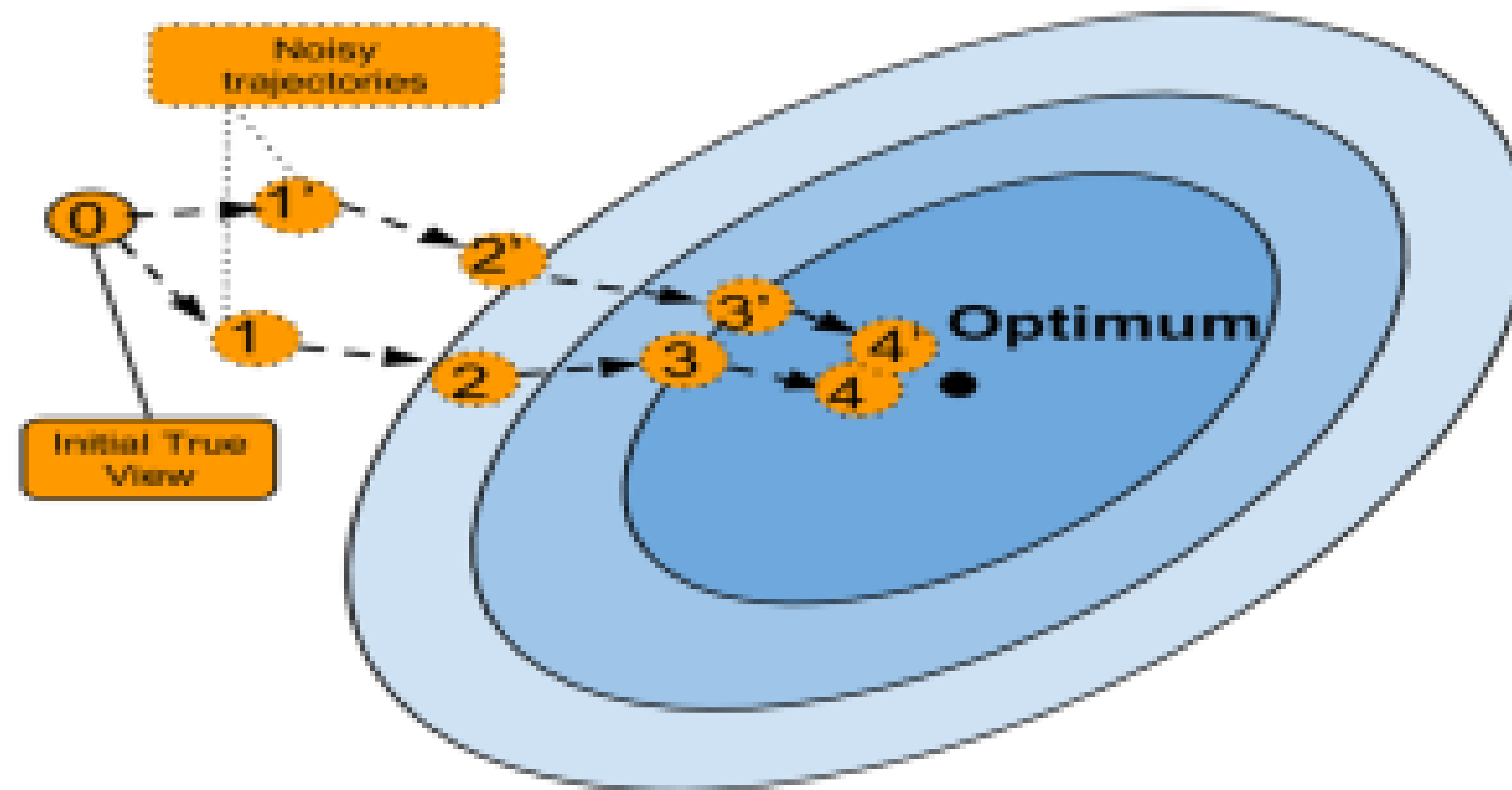


An interesting property of Gradient Descent (ascent)

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

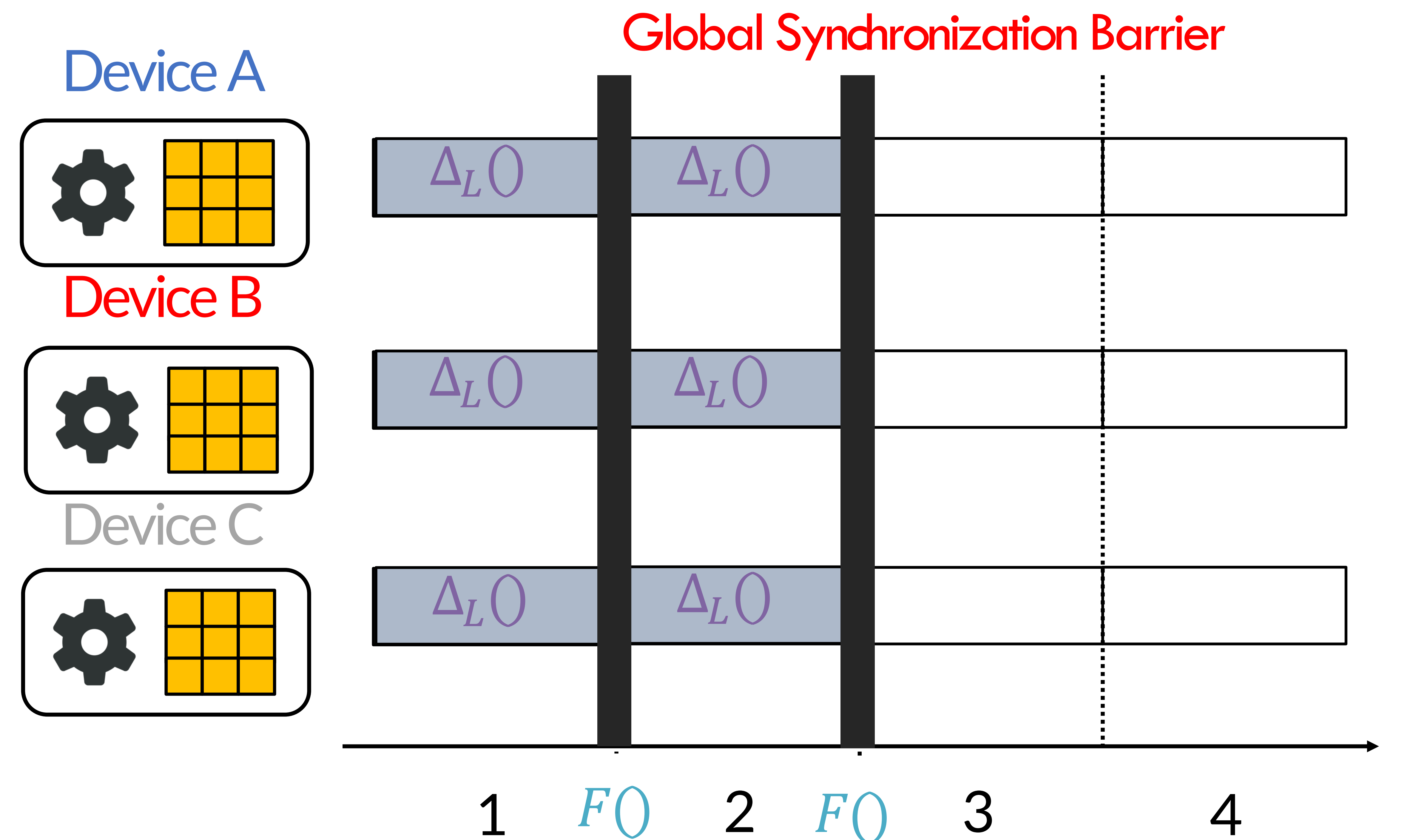


Machine Learning is Error-tolerant (under certain conditions)

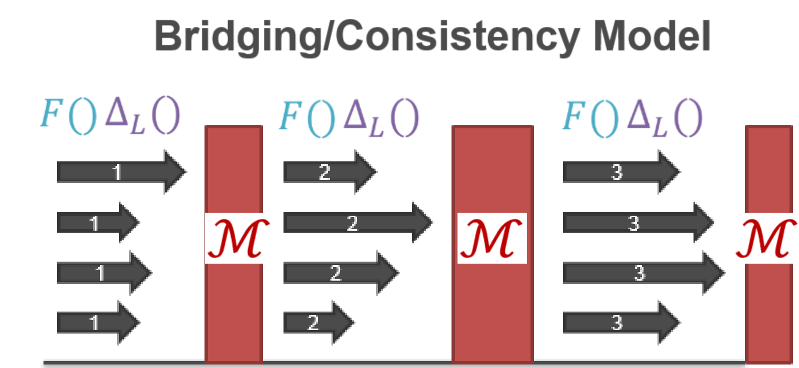


Background: Strict Consistency

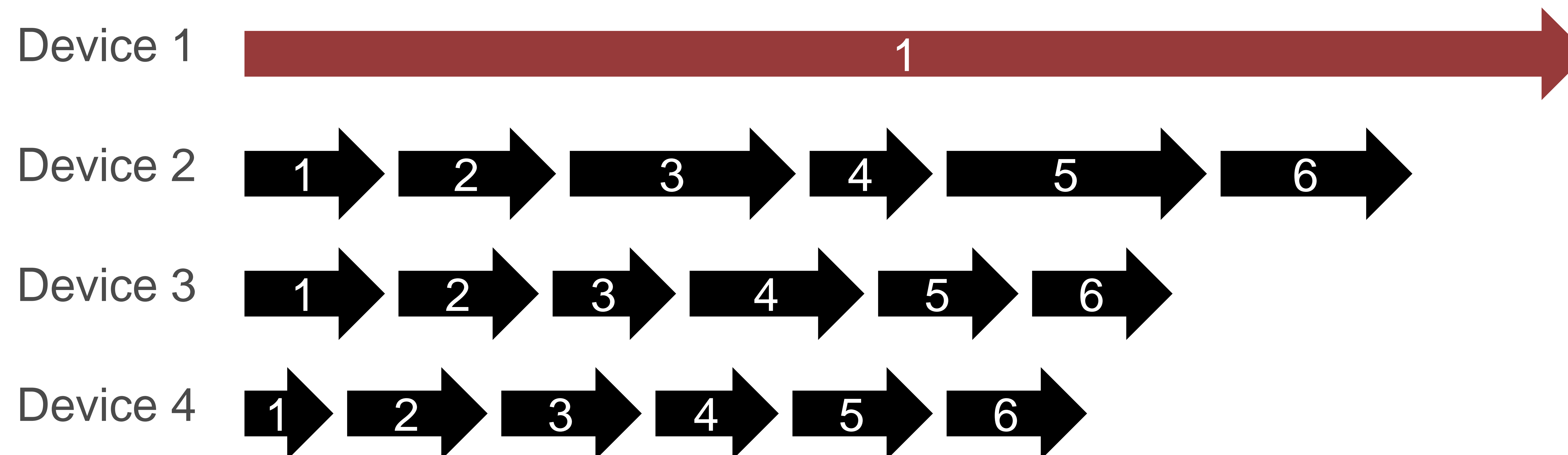
- **Baseline:** Bulk Synchronous Parallel (BSP)
  - MapReduce, Spark, many DistML Systems
- Devices compute updates $\Delta_L()$ between global barriers (iteration boundaries)
  - Messages $\mathcal{M}$ exchanged only during barriers
- **Advantage: Execution is serializable**
  - Same guarantees as sequential algo!
  - Provided that aggregation $F()$ is agnostic to order of messages $\mathcal{M}$ (e.g. in SGD)



Background: Asynchronous Communication (No Consistency)



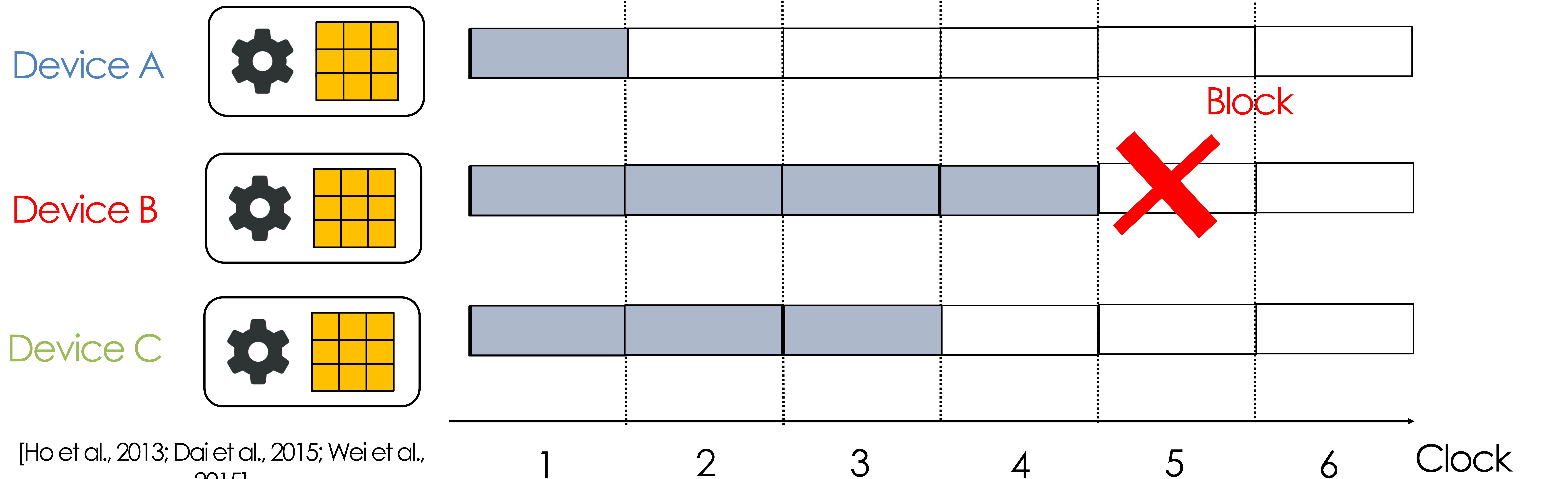
- **Asynchronous (Async):** removes all communication barriers
  - Maximizes computing time
  - Transient stragglers will cause messages to be extremely stale
    - Ex: Device 2 is at $t = 6$, but Device 1 has only sent message for $t = 1$
- **Some Async software:** messages can be applied while computing $F()$, $\Delta_L()$
  - Unpredictable behavior, can hurt statistical efficiency!



Background: Bounded Consistency

Bounded consistency models: Middle ground between BSP and fully-asynchronous (no-barrier)

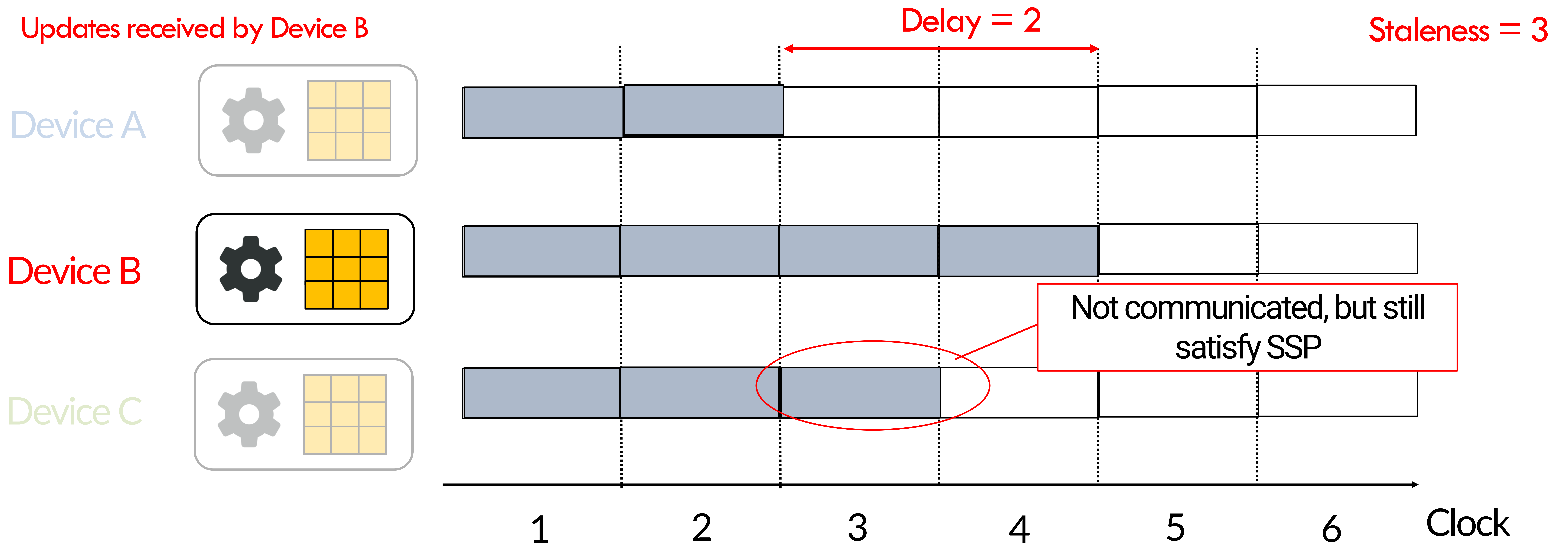
e.g. Stale Synchronous Parallel (SSP): Devices allowed to iterate at different speeds
- Fastest & slowest device must not drift > s iterations apart (in this example, $s = 3$)
  - s is the maximum staleness



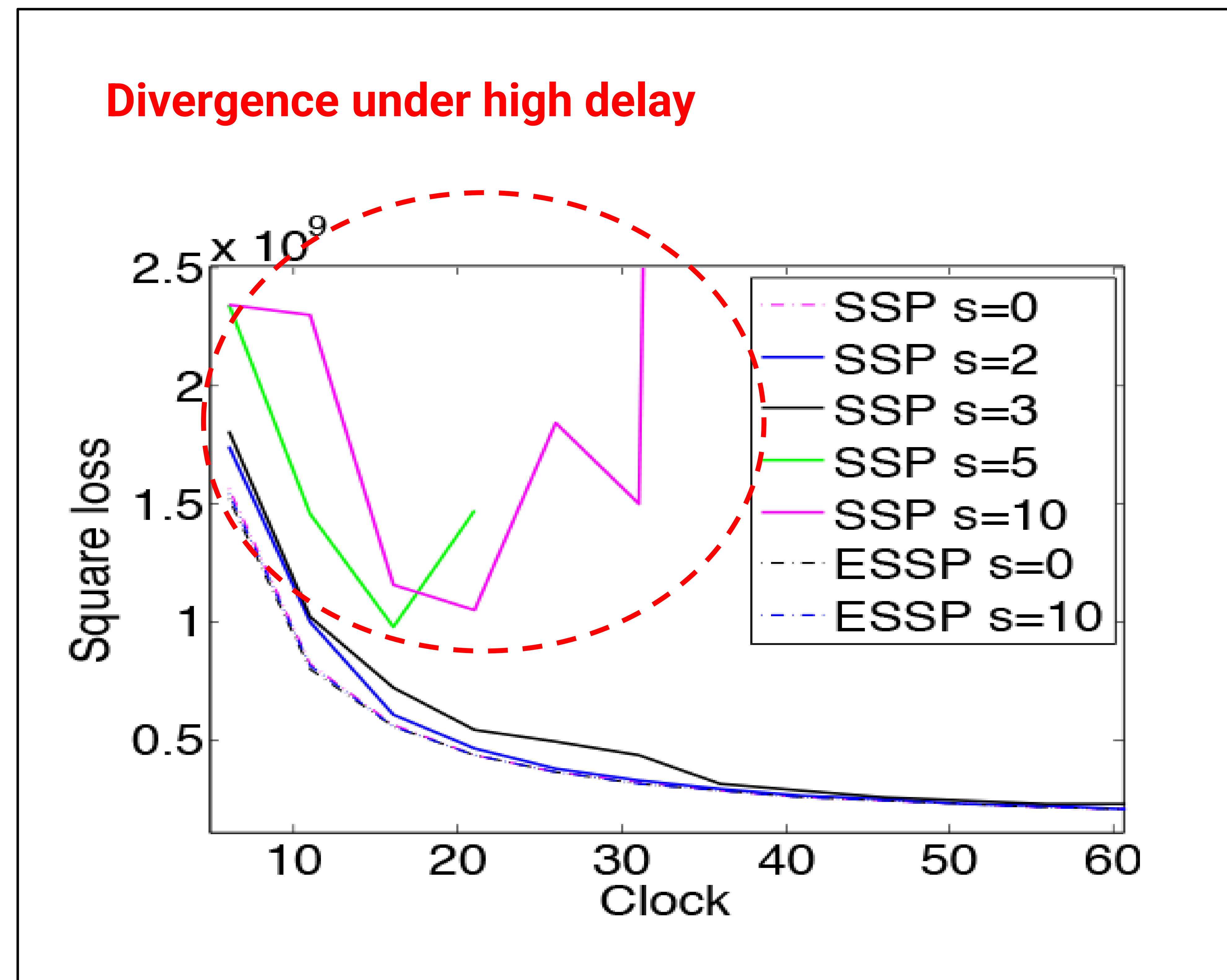
[Ho et al., 2013; Dai et al., 2015; Wei et al., 2015]

SSP: “Lazy” Communication

SSP: devices avoid communicating unless necessary
- i.e. when staleness condition is about to be violated
- Favors throughput at the expense of statistical efficiency



Impacts of Consistency/Staleness: Unbounded Staleness



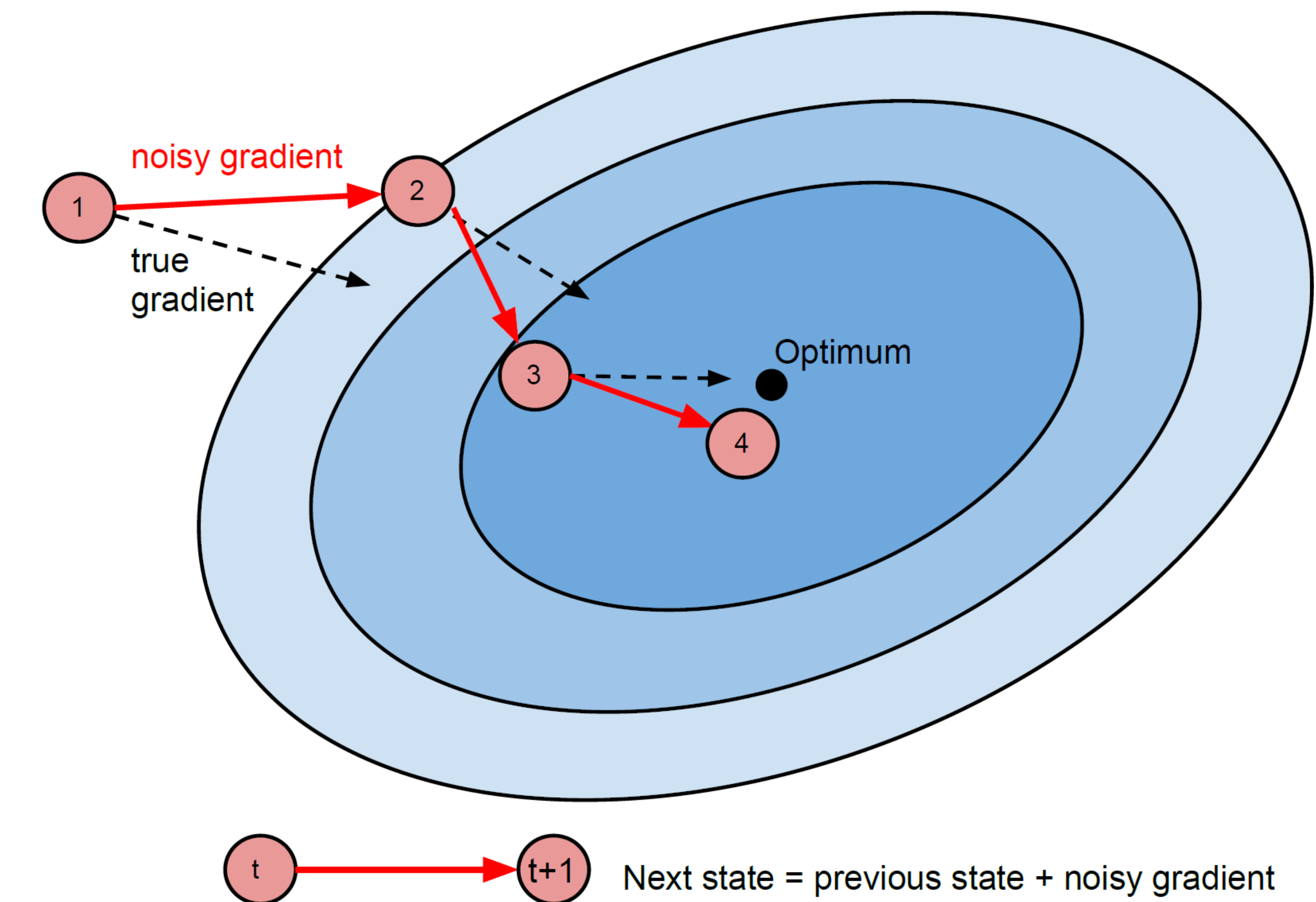
Theory: SSP Expectation Bound

- **Goal:** minimize convex $f(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} f_t(\mathbf{x})$

  (Example: Stochastic Gradient)

  - L-Lipschitz, problem diameter bounded by F^2
  - Staleness s, using P parallel devices
  - Use step size $\eta_t = \frac{\sigma}{\sqrt{t}}$ with $\sigma = \frac{F}{L\sqrt{2(s+1)P}}$

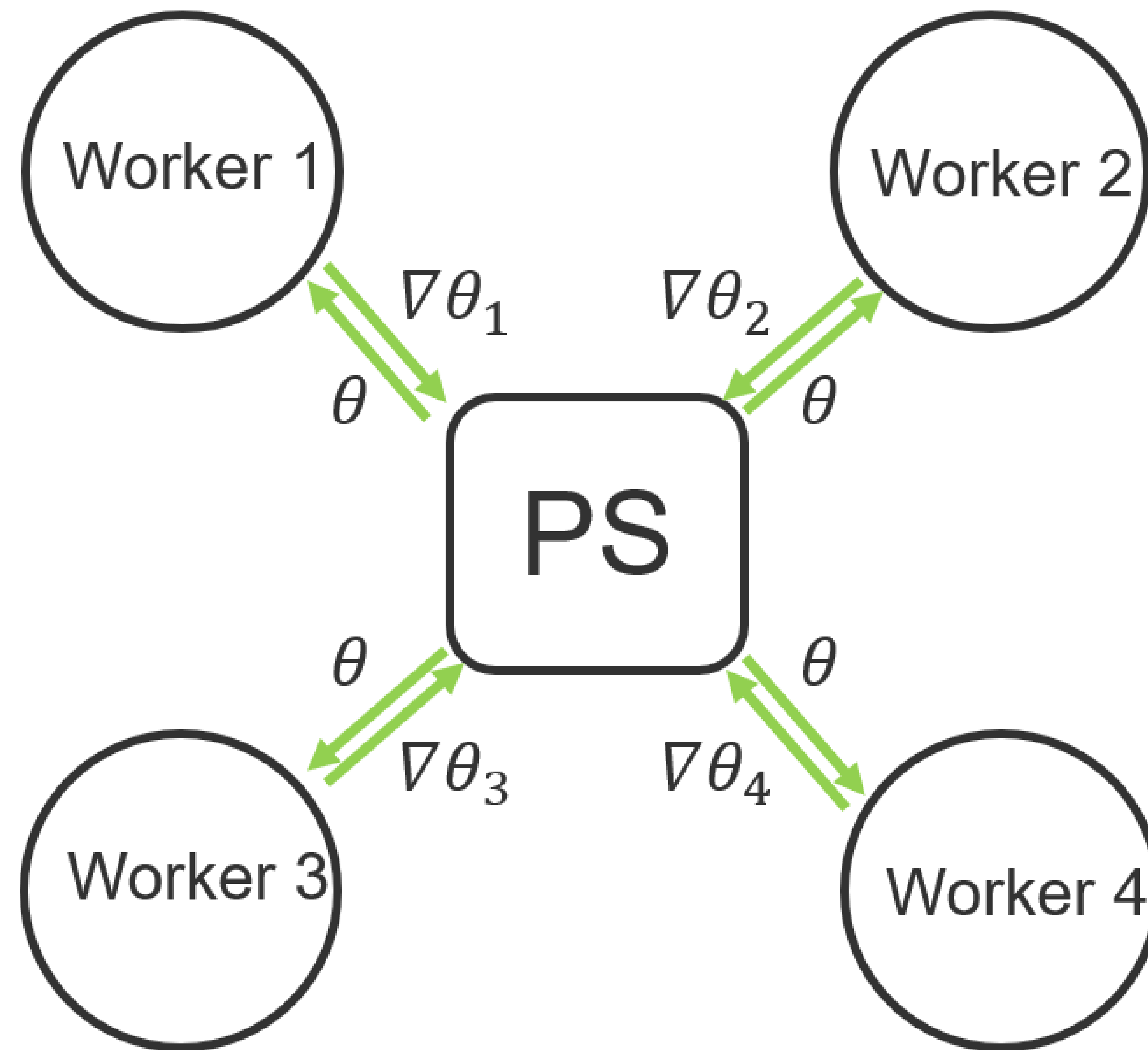
- **(E)SSP converges according to**
  - Where T is the number of iterations



Difference between
SSP estimate and true optimum

$$R[\mathbf{X}] := \left[\frac{1}{T}\sum_{t=1}^{T} f_t(\tilde{\mathbf{x}}_t)\right] - f(\mathbf{x}^*) \leq 4FL\sqrt{\frac{2(s+1)P}{T}}$$

Parameter Server Naturally emerges

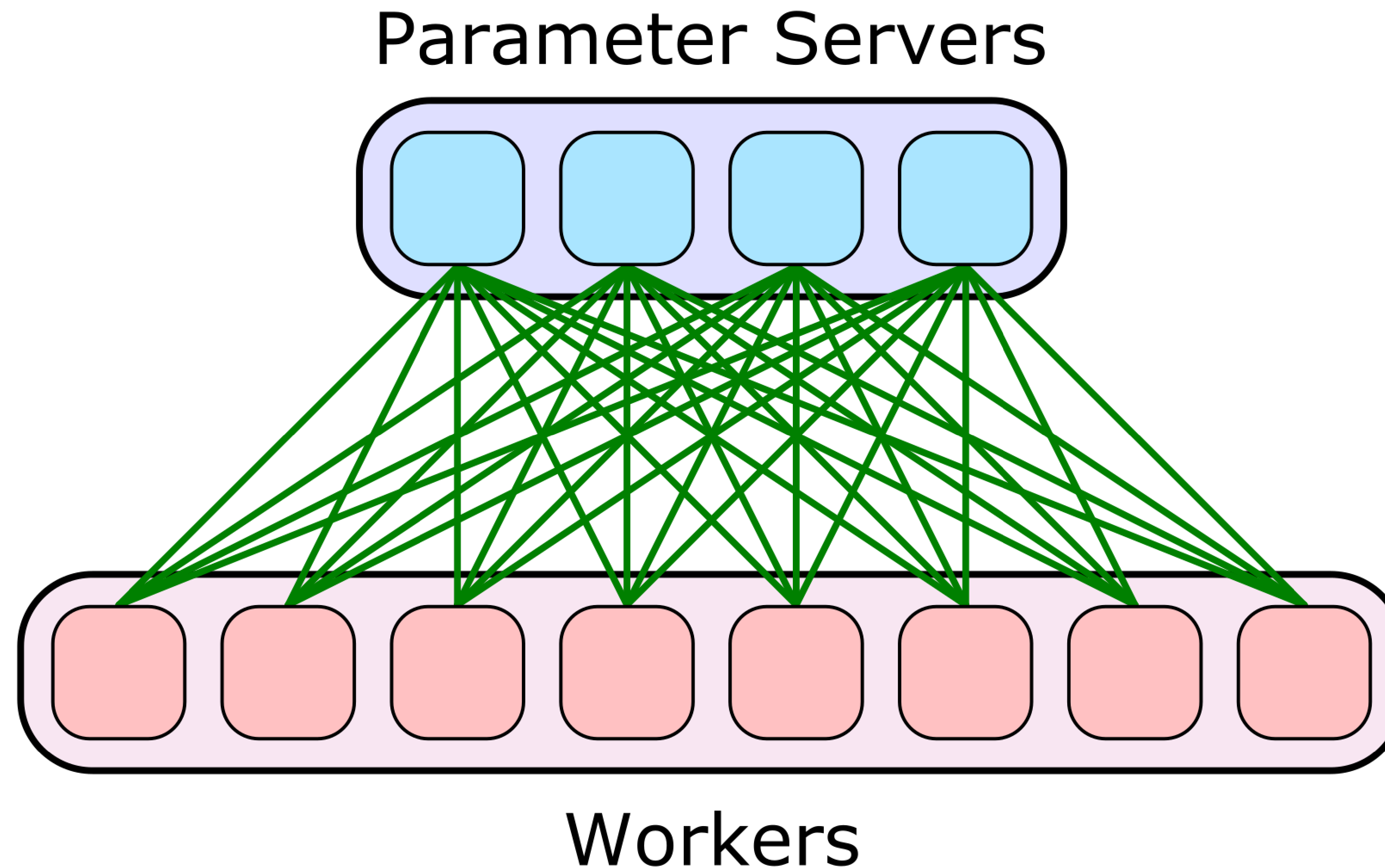


How to Implement Parameter Server?

- Key considerations:
  - Server: Communication bottleneck
  - Fault tolerance
  - Programming Model
  - Handling GPUs

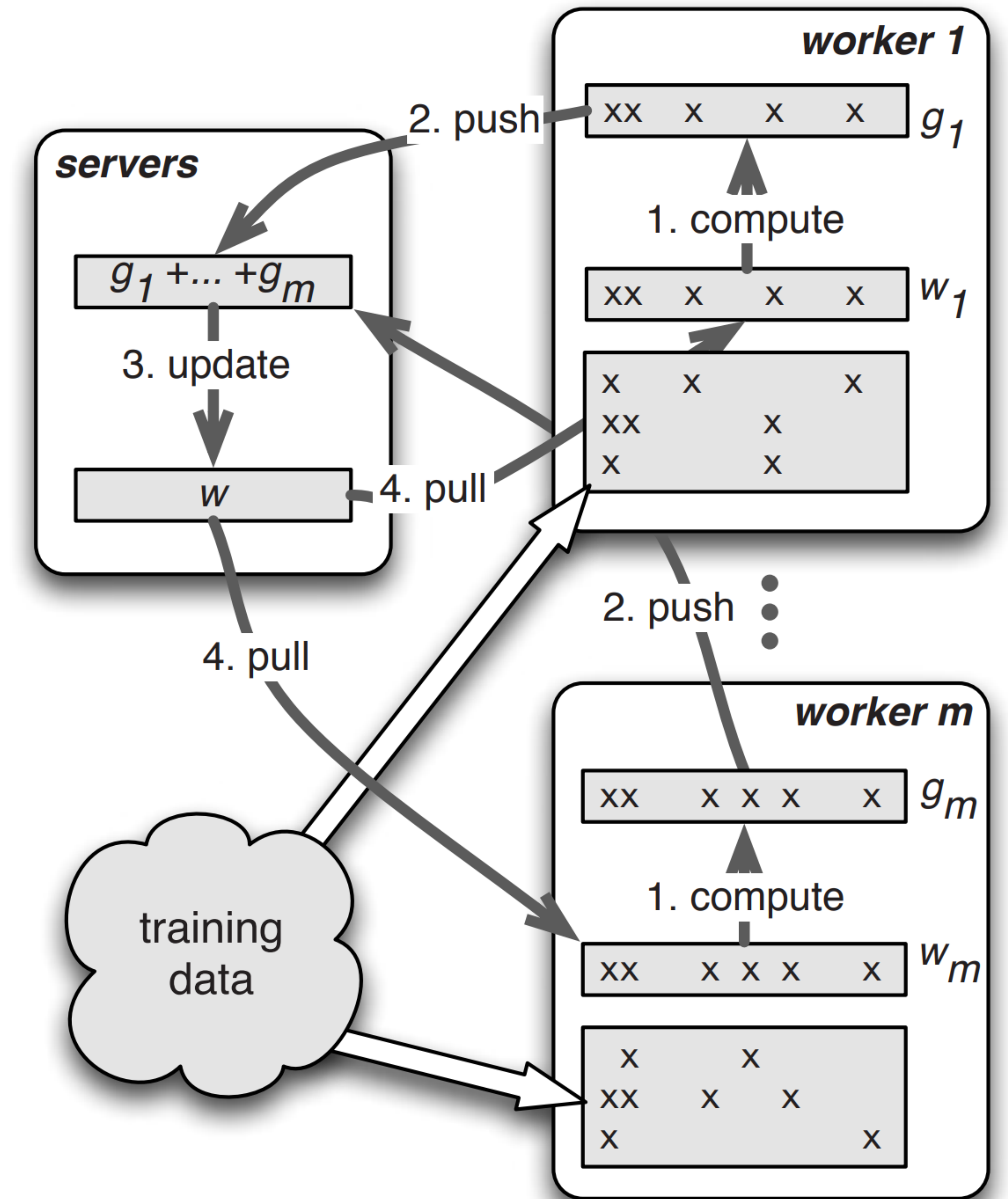
Parameter Server Implementation

- Sharded parameter server: sharded KV stores
  - Avoid communication bottleneck
  - Redundancy across different PS shards



Programming Model

- Client:
  - Push()
  - Pull()
  - Compute()
- Server:
  - Update()
- Very similar to the spirit of Map Reduce
- A lot of flexibility for users to customize
  - Recall Mapreduce vs. Spark



Summary: Parameter Server

- Why does it emerge?
  - Unification of iterative-convergence optimization algorithm
- What problems does it address and how?
  - Heavy communication, via flexible consistency
- Pros?
  - Cope well with iterative-convergent algo
- Cons?
  - Extension to GPUs?
  - Strong assumption on communication bottleneck