Solving the straggler problem with bounded staleness

Jim Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Gregory R. Ganger, Garth Gibson,

Kimberly Keeton*, Eric Xing

PARALLEL DATA LABORATORY

Carnegie Mellon University

* HP Labs

Carnegie Mellon
Parallel Data Laboratory

Overview

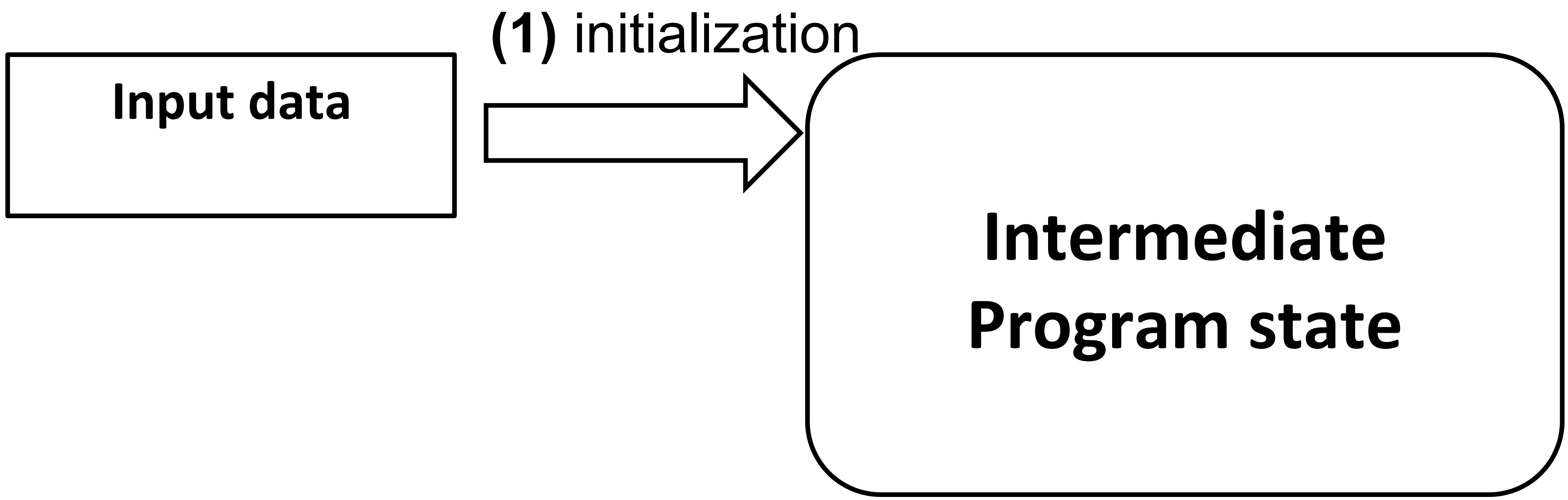
It's time for all applications (and systems) to worry about data freshness

- Current focus: parallel machine learning
- Often limited by synchronization overhead
- What if we explicitly allow stale data?

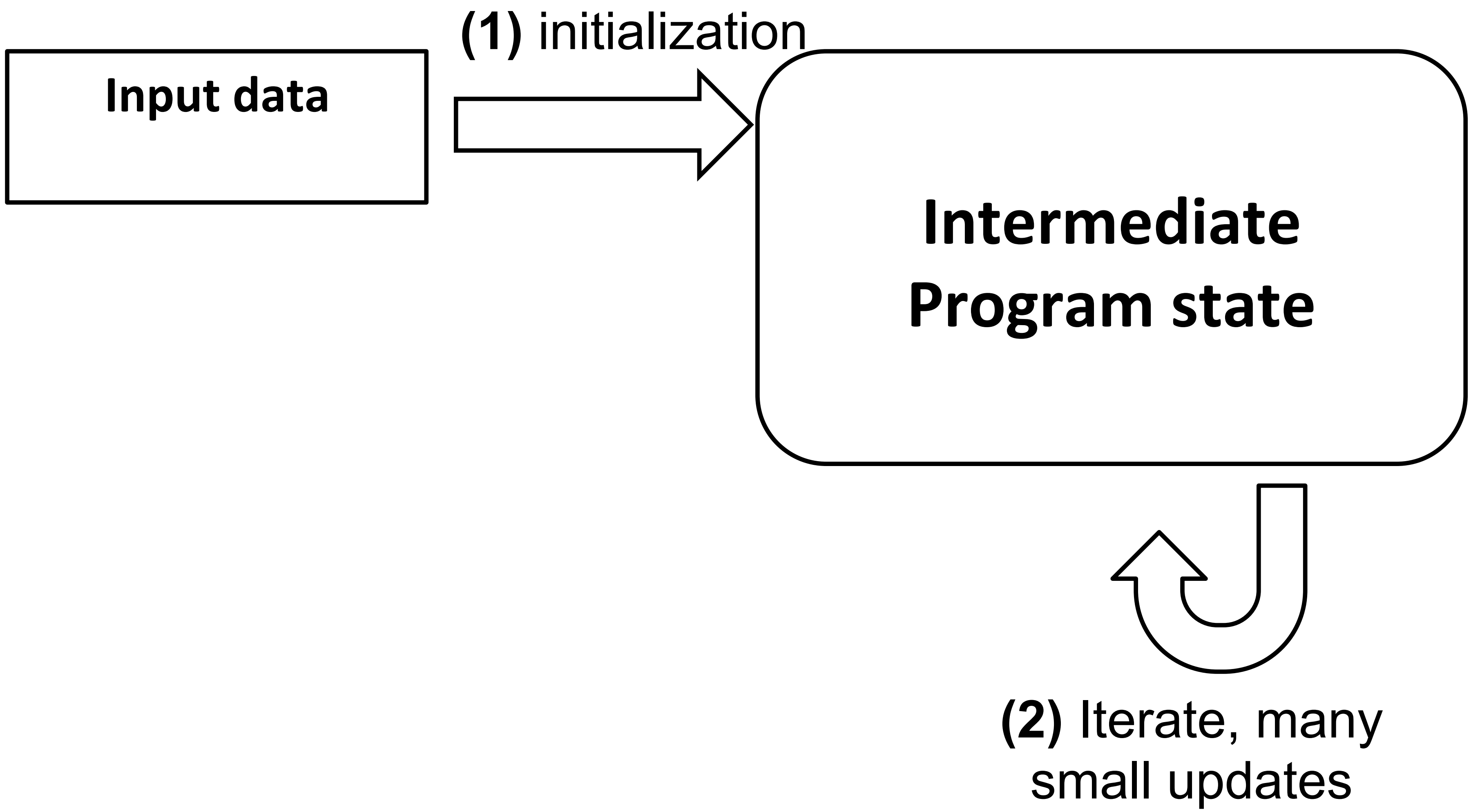
A typical ML algorithm

Input data

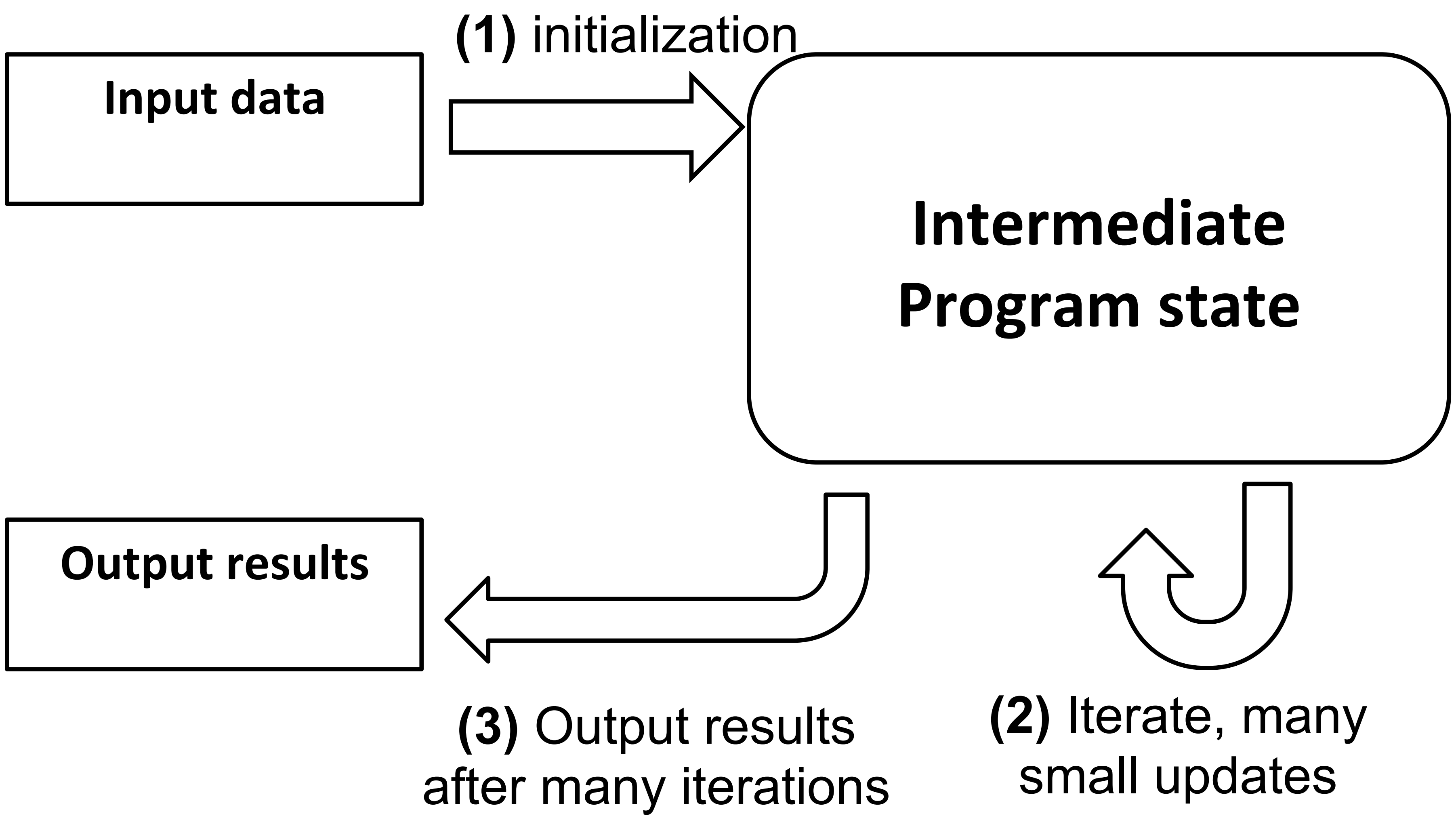
A typical ML algorithm



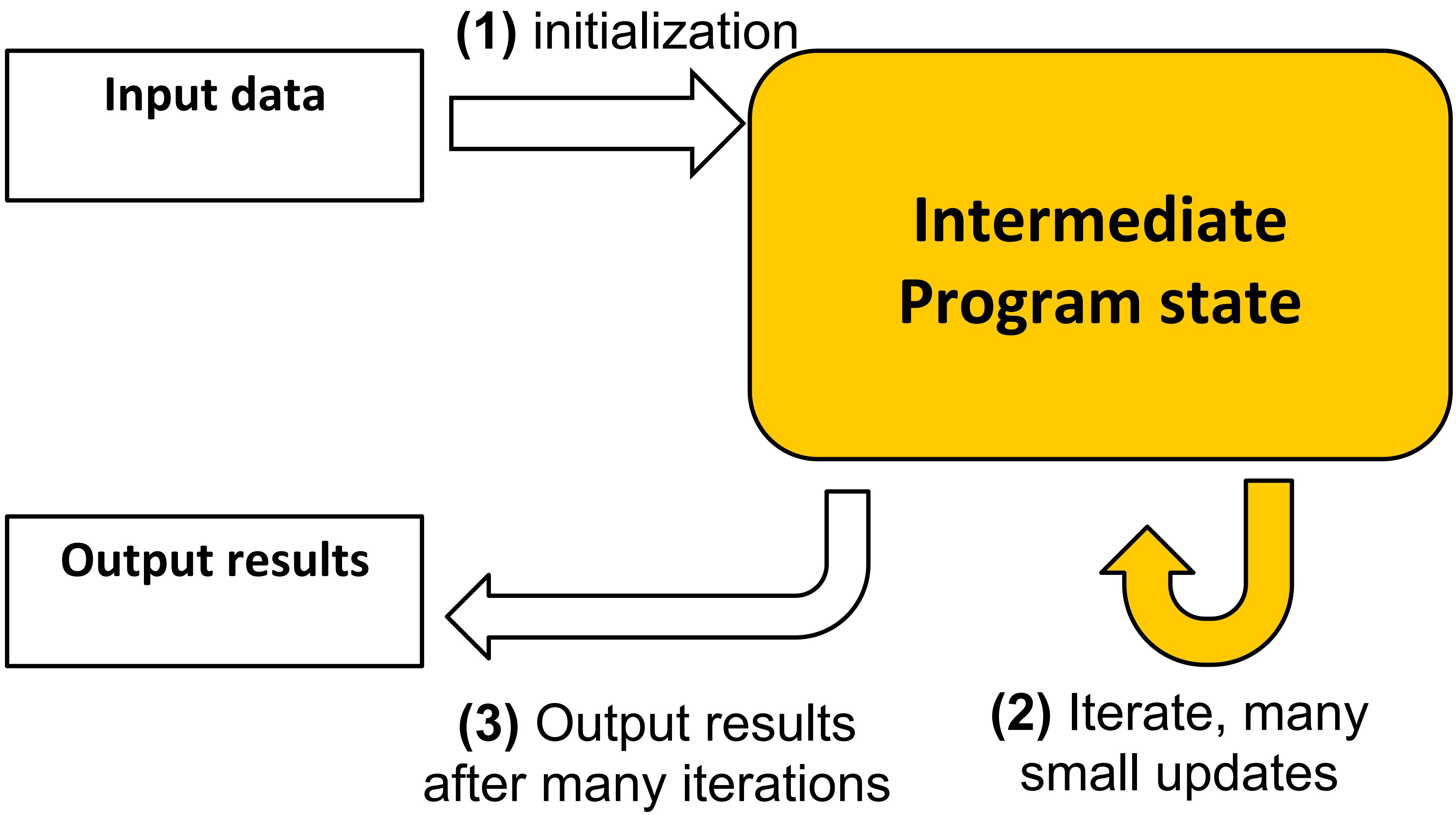
A typical ML algorithm



A typical ML algorithm



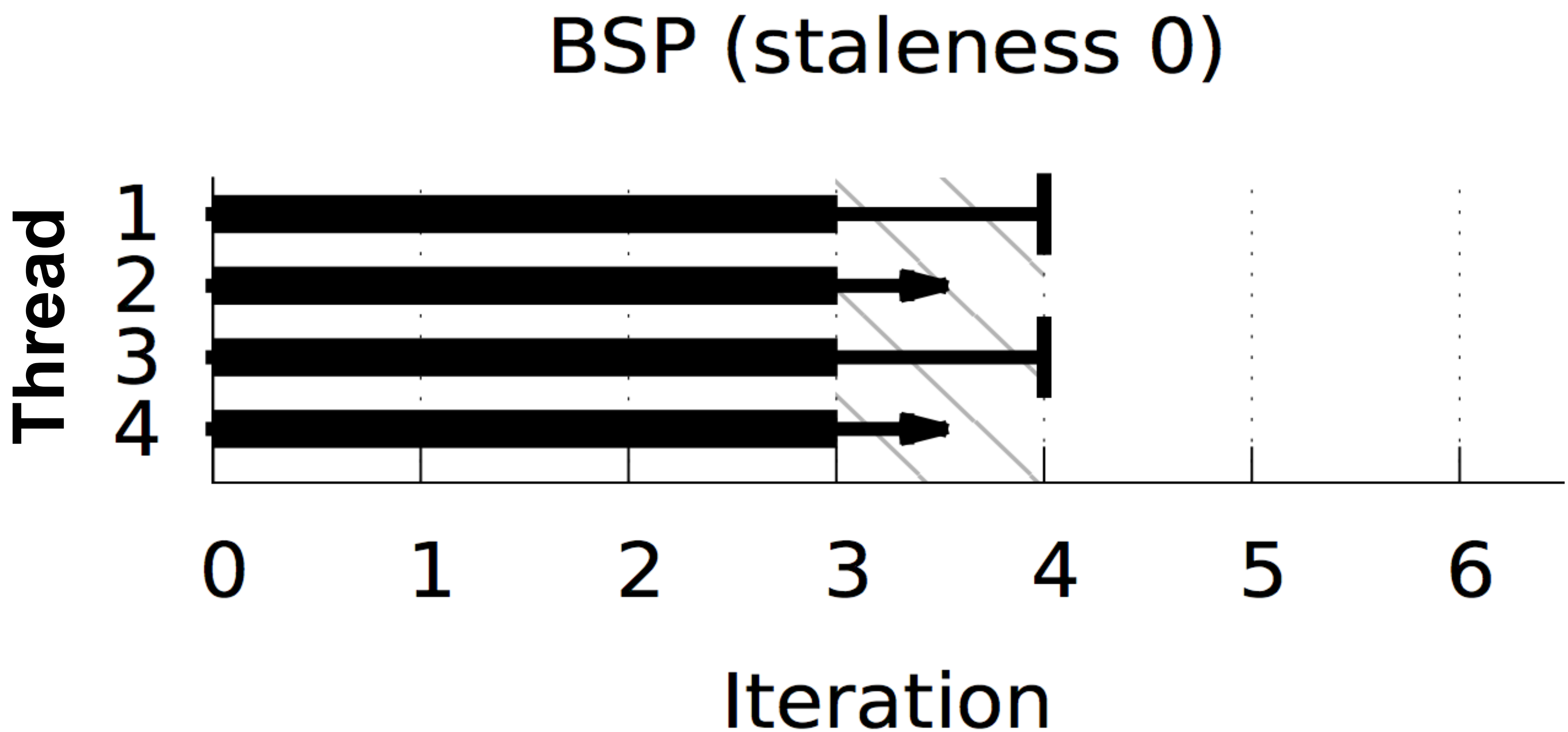
A typical ML algorithm



Parallel ML

- Generally follows **bulk synchronous parallel** model

- Many iterations of
  1. Computation: compute new values
  2. Synchronization: wait for all other threads
  3. Communication: send new values to other threads
  4. Synchronization: wait for all other threads… again



All threads must be on the same iteration to continue

Stragglers in BSP

Slow thread(s) will hold up entire application

- Predictable stragglers
  - Slow/old machine
  - Bad network card
  - More data assigned to some threds

Stragglers in BSP

Slow thread(s) will hold up entire application

- Predictable stragglers ➔ Easy case
  - Slow/old machine
  - Bad network card
  - More data assigned to some threads

Stragglers in BSP

Slow thread(s) will hold up entire application

- Predictable stragglers ➔ Easy case
- Unpredictable stragglers ➔ ???

Stragglers in BSP

Slow thread(s) will hold up entire application

- Predictable stragglers → Easy case
- Unpredictable stragglers → ???
  - **Hardware**: disk seeks, network, CPU interrupts
  - **Software**: garbage collection, virtualization
  - **Algorithmic**: Calculating objectives and stopping conditions

Stragglers in BSP

Slow thread(s) will hold up entire application

- Predictable stragglers ➔ Easy case
- Unpred
  - **Hardw** CPU interr
  - **Softw** ion
  - **Algori** pping conditions

Don’t synchronize

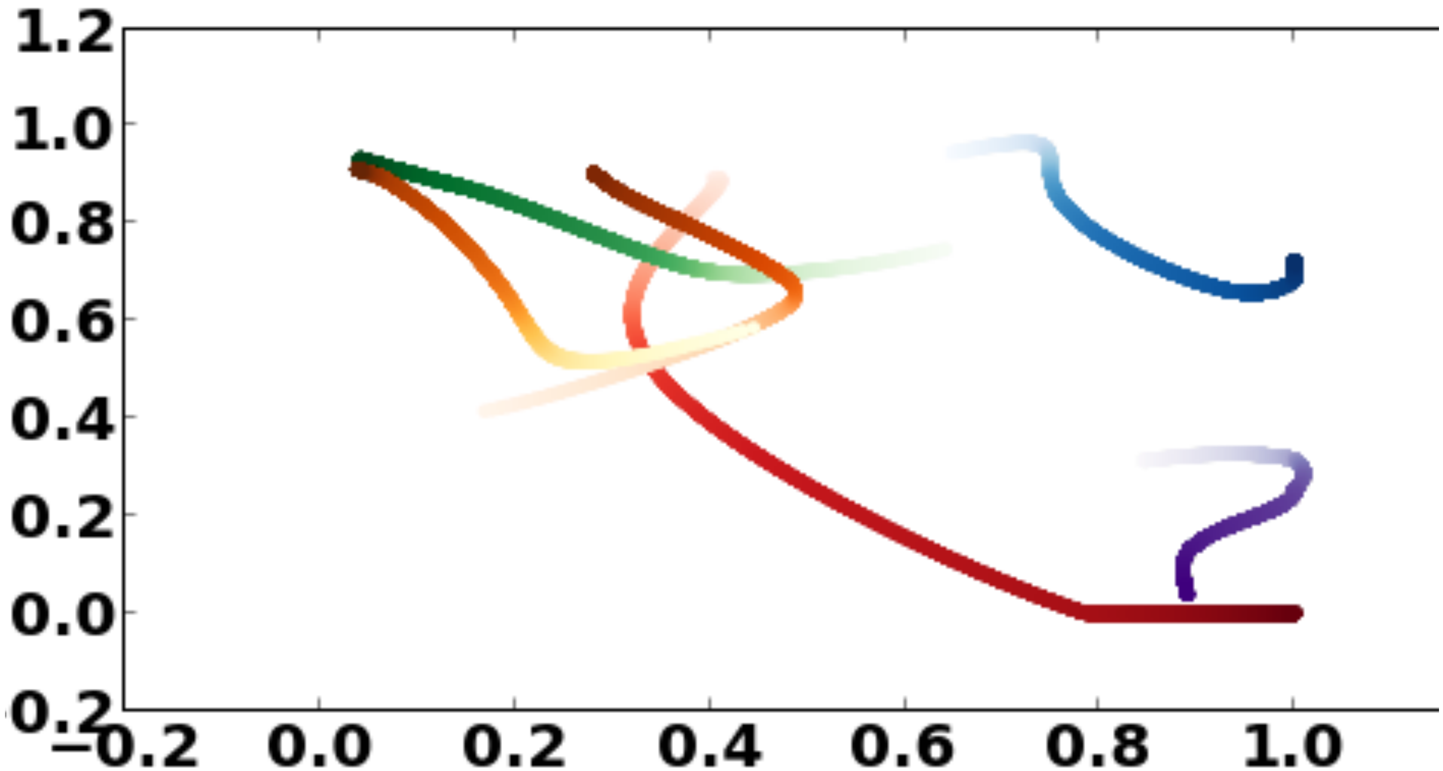
Don’t synchronize?

- Well, don’t synchronize much
  - **Read old (stale) results from other threads**
  - Application controls how stale the data can be

- Machine learning can get away with that

- Algorithms are **convergent**
  - Given (almost) any state, will find correct solution
  - Errors introduced by staleness are usually ok

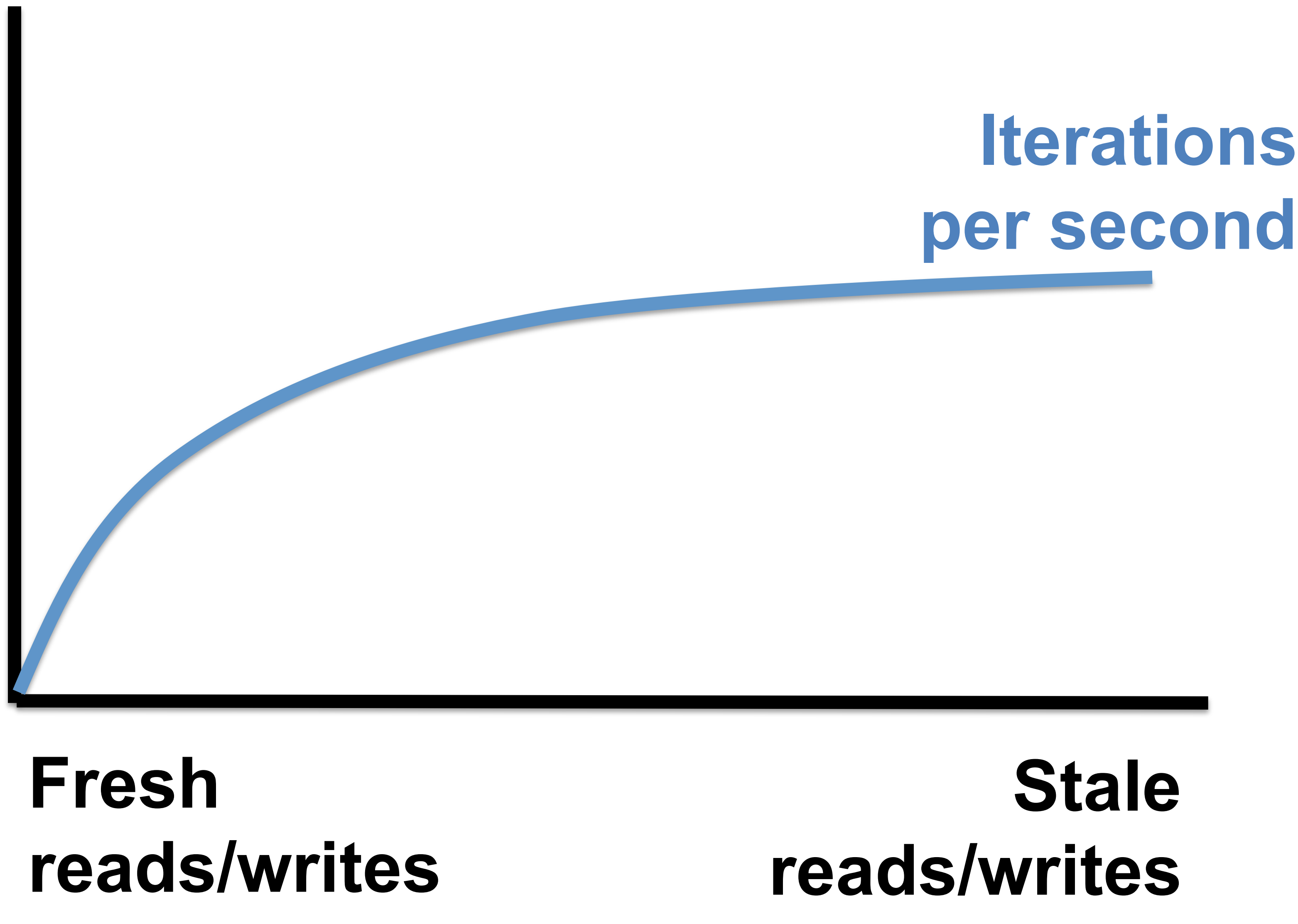
Trajectories of points in 2d



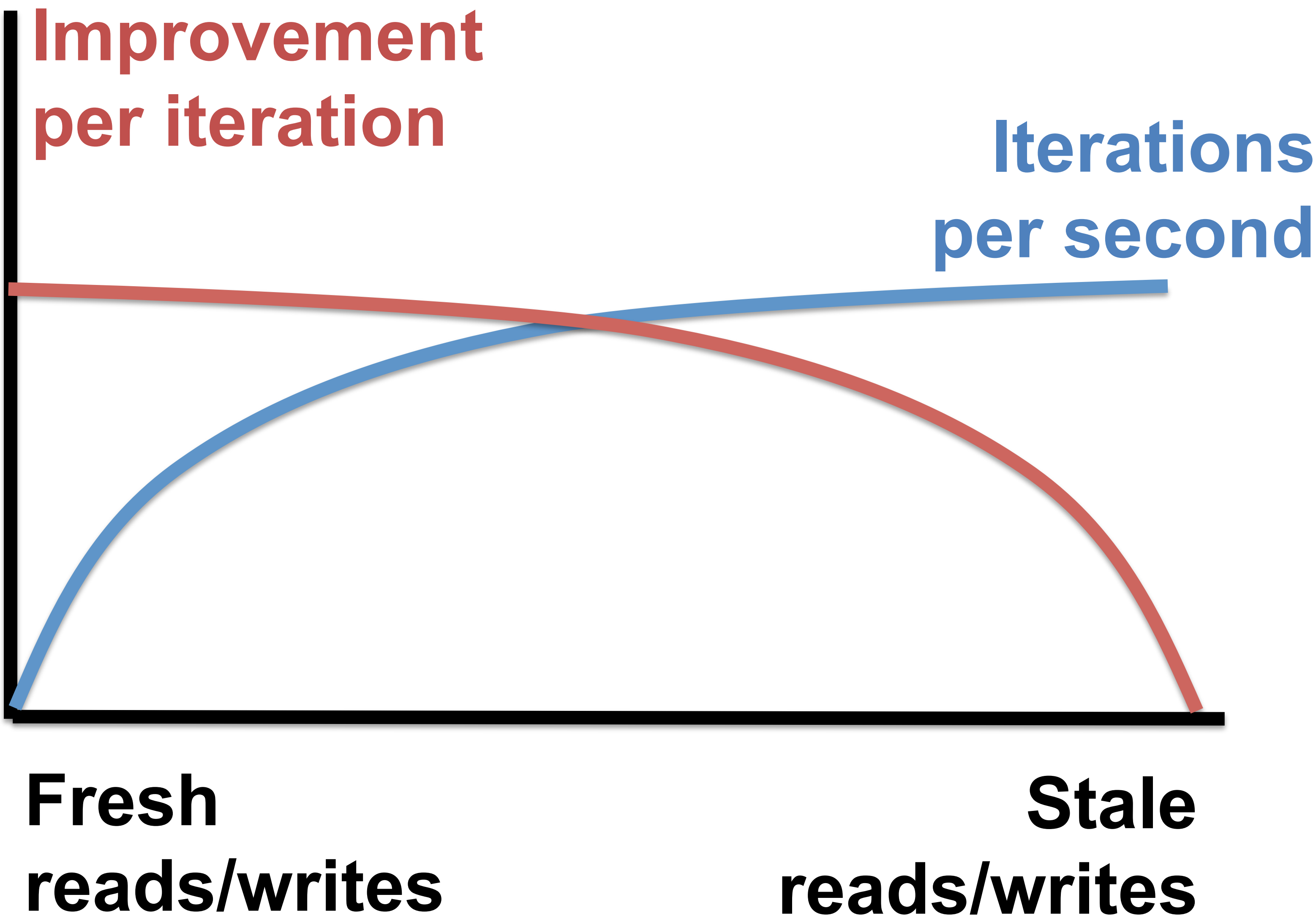
Points are initialized randomly,
Always settle to correct locations

Freshness and convergence: the sweet spot

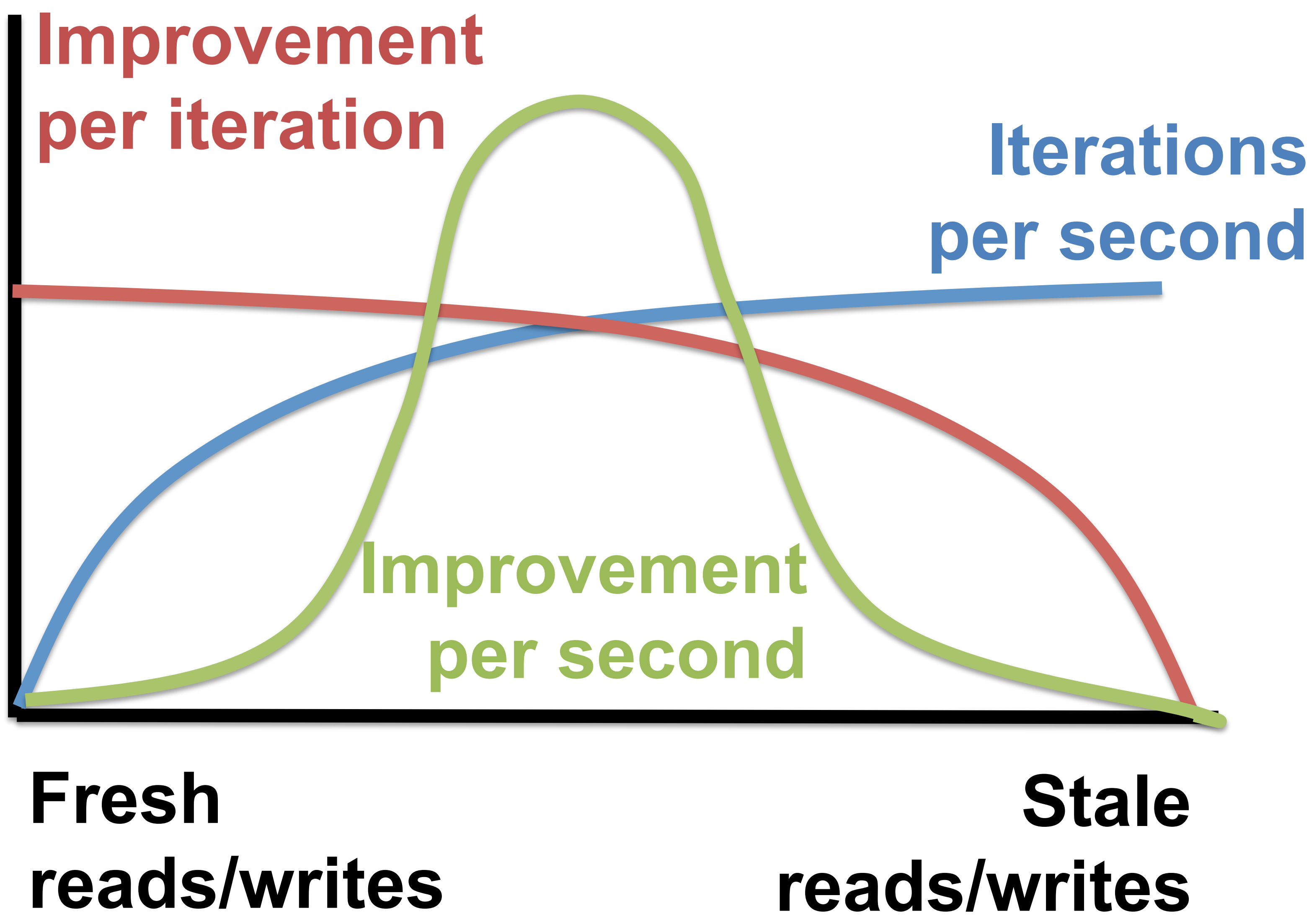
Freshness and convergence: the sweet spot



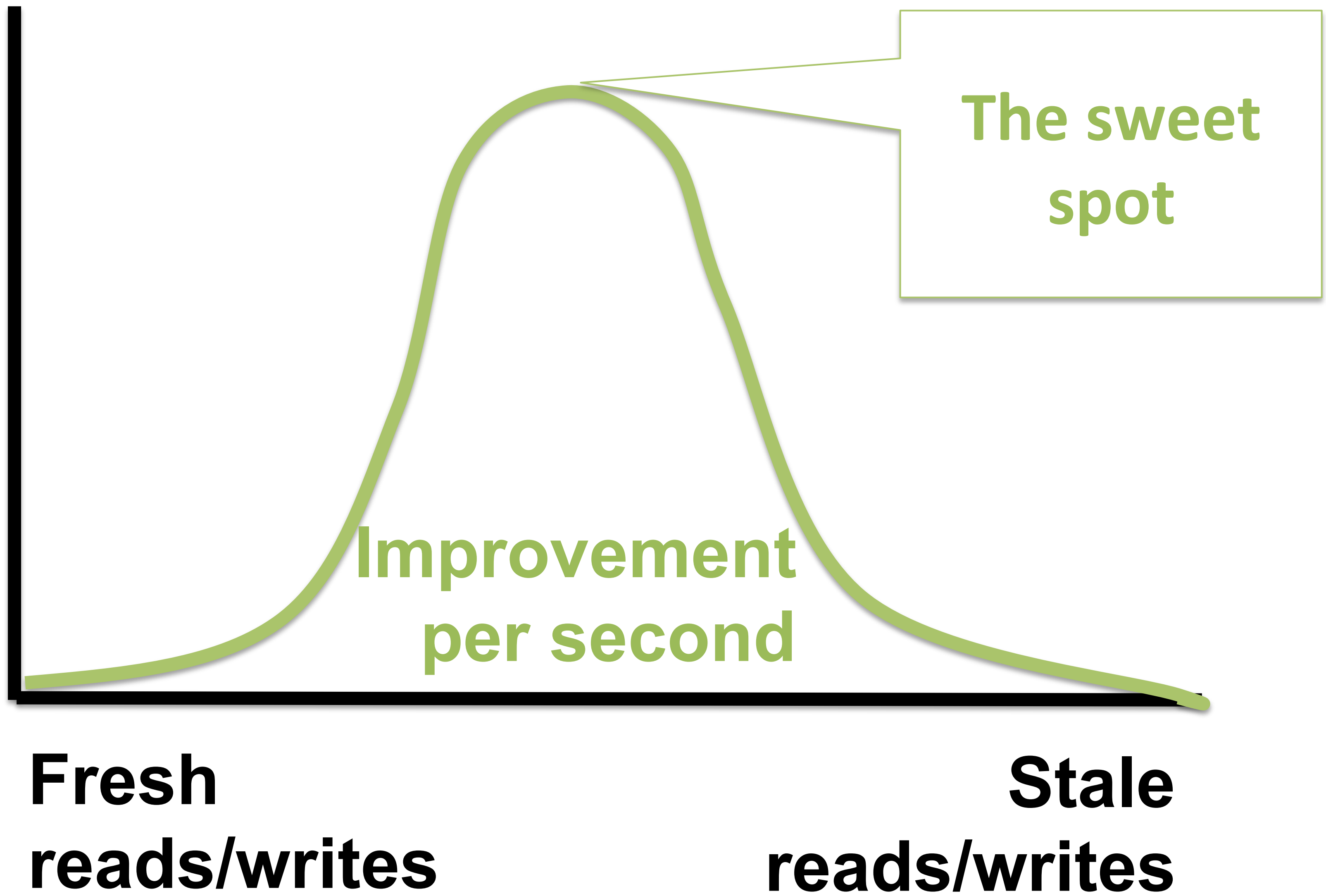
Freshness and convergence: the sweet spot



Freshness and convergence: the sweet spot

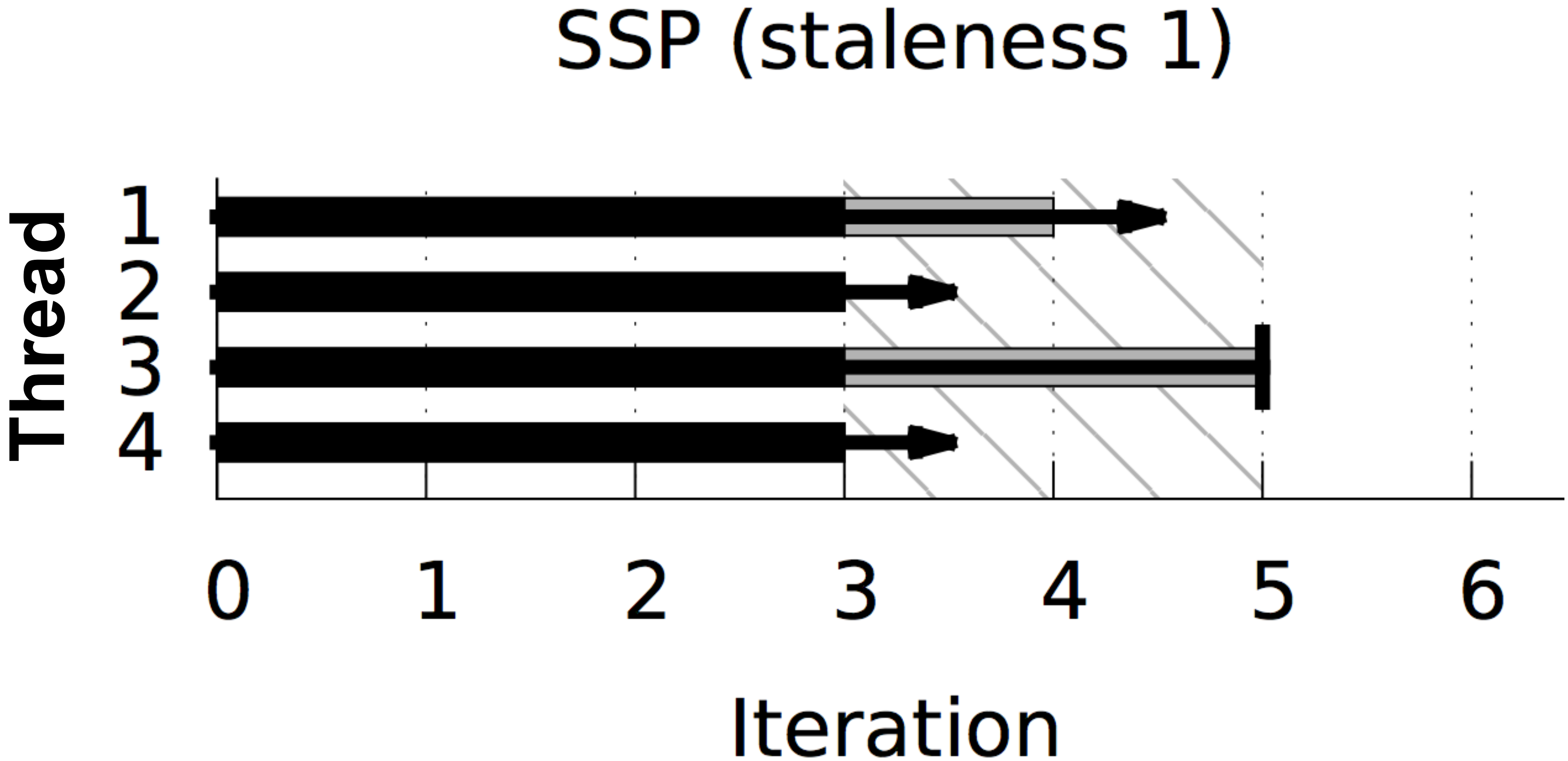


Freshness and convergence: the sweet spot



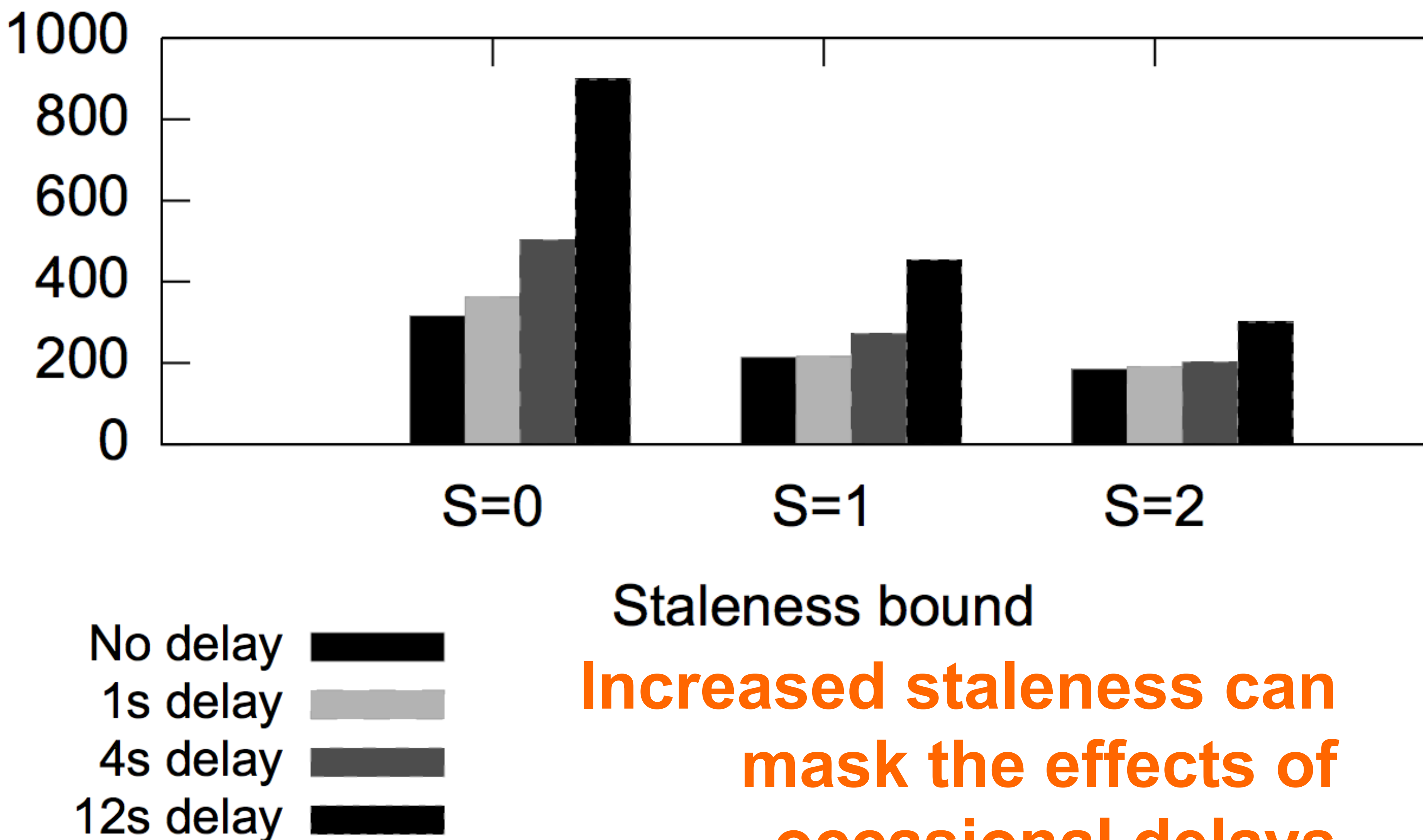
Stale synchronous parallel

- Allow threads to continue ahead of others
  - Avoids temporary straggler effects
- Application can limit allowed staleness
  - Ensure convergence
  - E.g. "threads may not be more than 3 iters ahead"



Threads proceed, possibly using stale data

Total convergence time



Increased staleness can mask the effects of occasional delays

Ongoing work

- Characterizing “staleness-tolerant” algorithms
  - Properties of algorithms, rules of thumb
  - Convergence proof
- Automatically tune freshness requirement
- Specify freshness by error bounds
  - “Read X with no more than 5% error”

Summary

Introducing staleness, but *not too much staleness*, can improve performance of machine learning algorithms.