Parallelizing Transducers

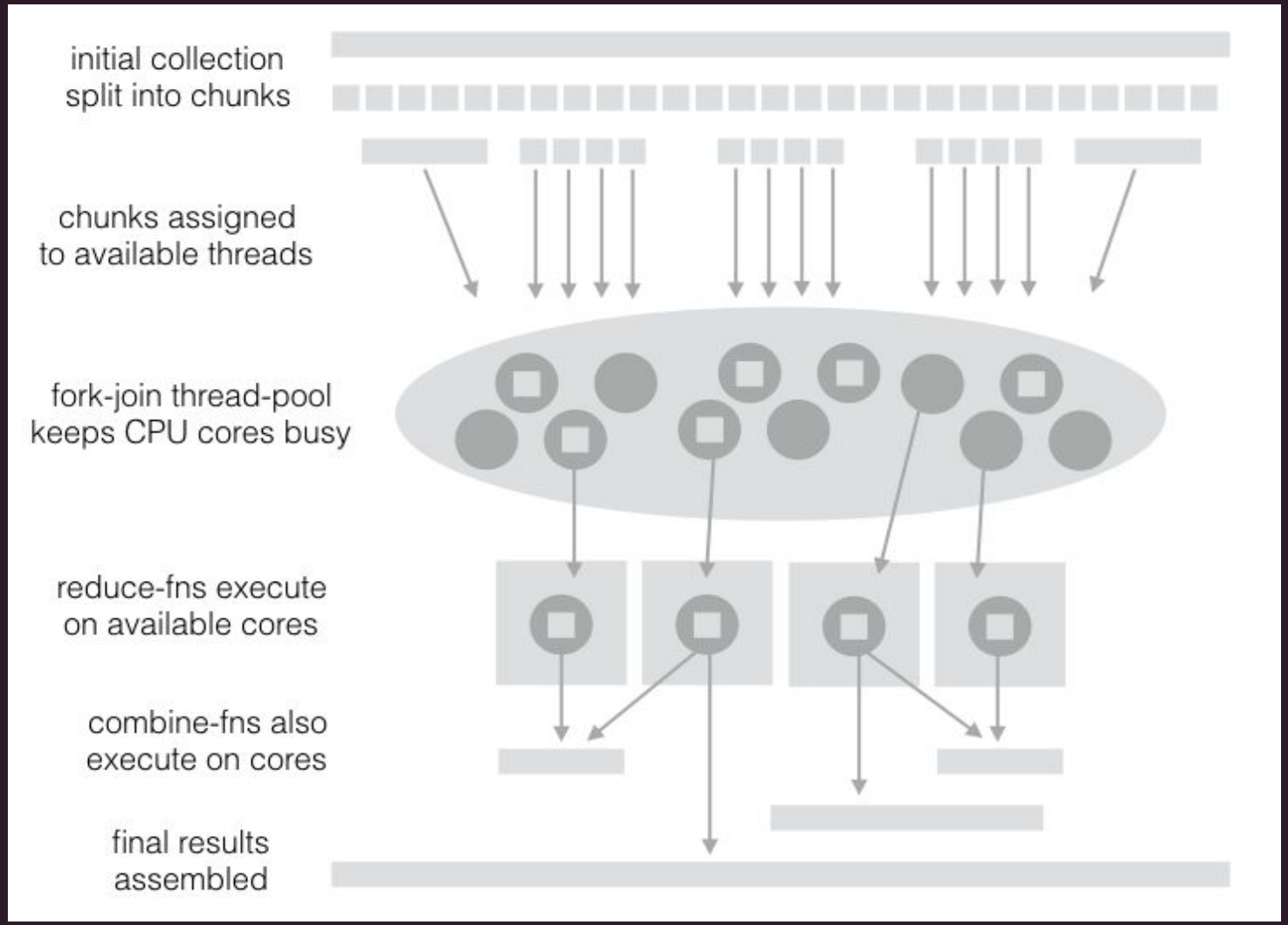
Transducers in Practice Workshop - CUFP 2017

Going parallel

- Transducers, especially big pipeline of transformations on top of large datasets, could run in parallel.
- There is no “ptransduce” but we could roll-out our own waiting for a standardized solution.
- **Approach 1**. Divide and conquer: split the input and work in parallel.
- **Approach 2**. Master-workers. Workers fight to take the next item from the dataset.

Parallelization with Reducers

- Reducers is a namespace part of Clojure: `clojure.core.reducers`
- They wrap the Java fork-join framework, a divide and conquer parallel strategy with work-stealing.
- They assume the input collection can be split into chunks (eagerly).
- They also assume combining the chunks back is commutative.


initial collection
split into chunks
chunks assigned
to available threads
fork-join thread-pool
keeps CPU cores busy
reduce-fns execute
on available cores
combine-fns also
execute on cores
final results
assembled

To use Reducers

- After requiring the namespace, r/fold is the main entry point
- They need a “reducef” reducing function (at the leaf).
- A “combinef” to join chunks back together.

Parallelization with core.async

- core.async provides a "pipeline" construct to associate multiple parallel threads to a channel
- Pipelines can be further "piped" together and apply a different transducer each
- Input data can be streamed or loaded from a collection in one go.

Resources

- “A Java fork-join framework” paper by Doug Lea
- Clojure Applied book contains chapters dedicated to Transducers with core.async pipes.

Lab 04
Parallelizing Transducers

Goal of Lab4

- Task1: parallelize the xform with reducers
- Task2: parallelize the xform with pipelines.
- Task3: different pipelines for different transducers

Open `transducers-workshop.lab04` to start.