TStreams

A Model of Parallel Computation

HP Cambridge Research Lab

Kath Knobe

Dave Panariti

Alex Nelson

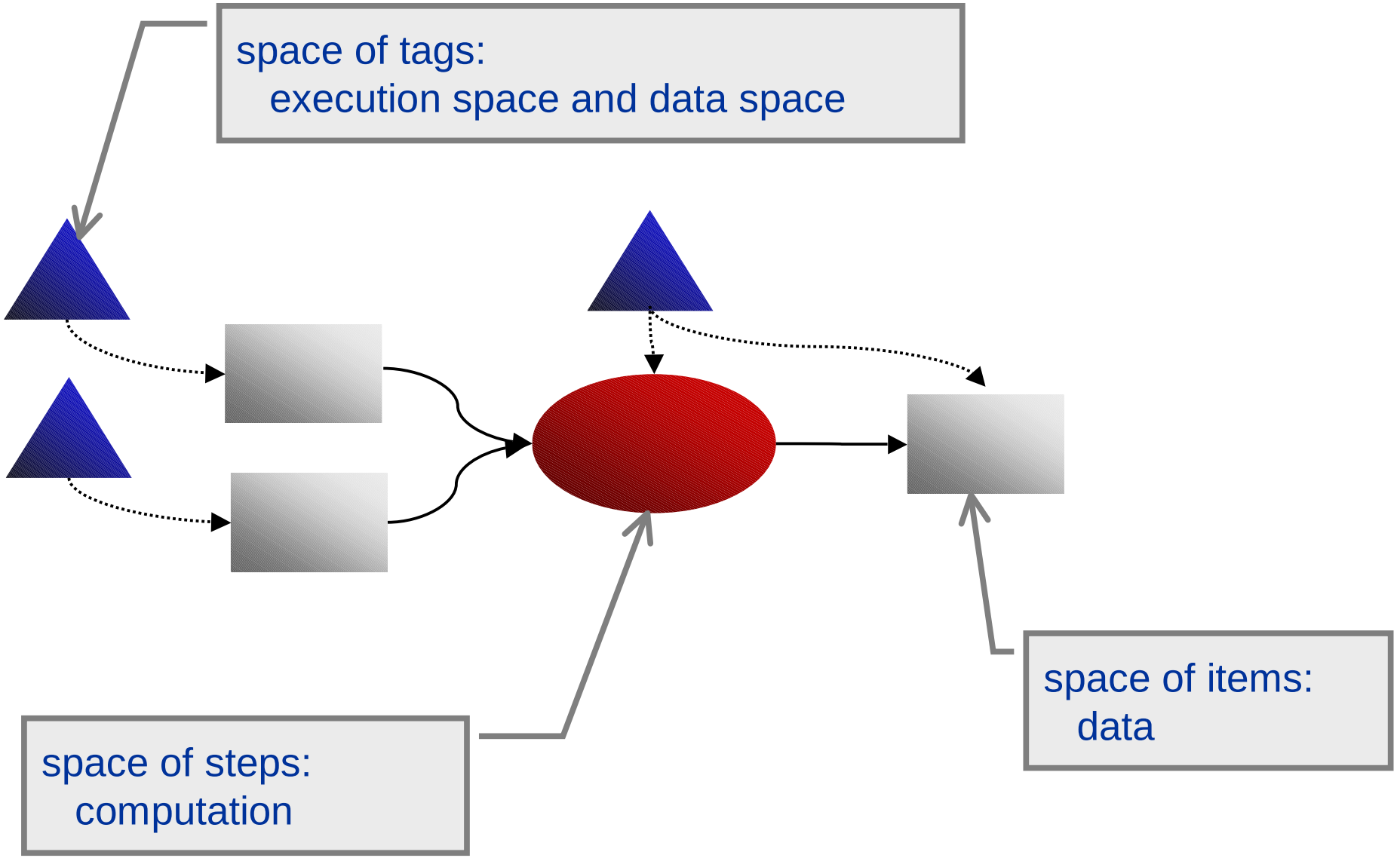
Carl Offner

Dick Flower

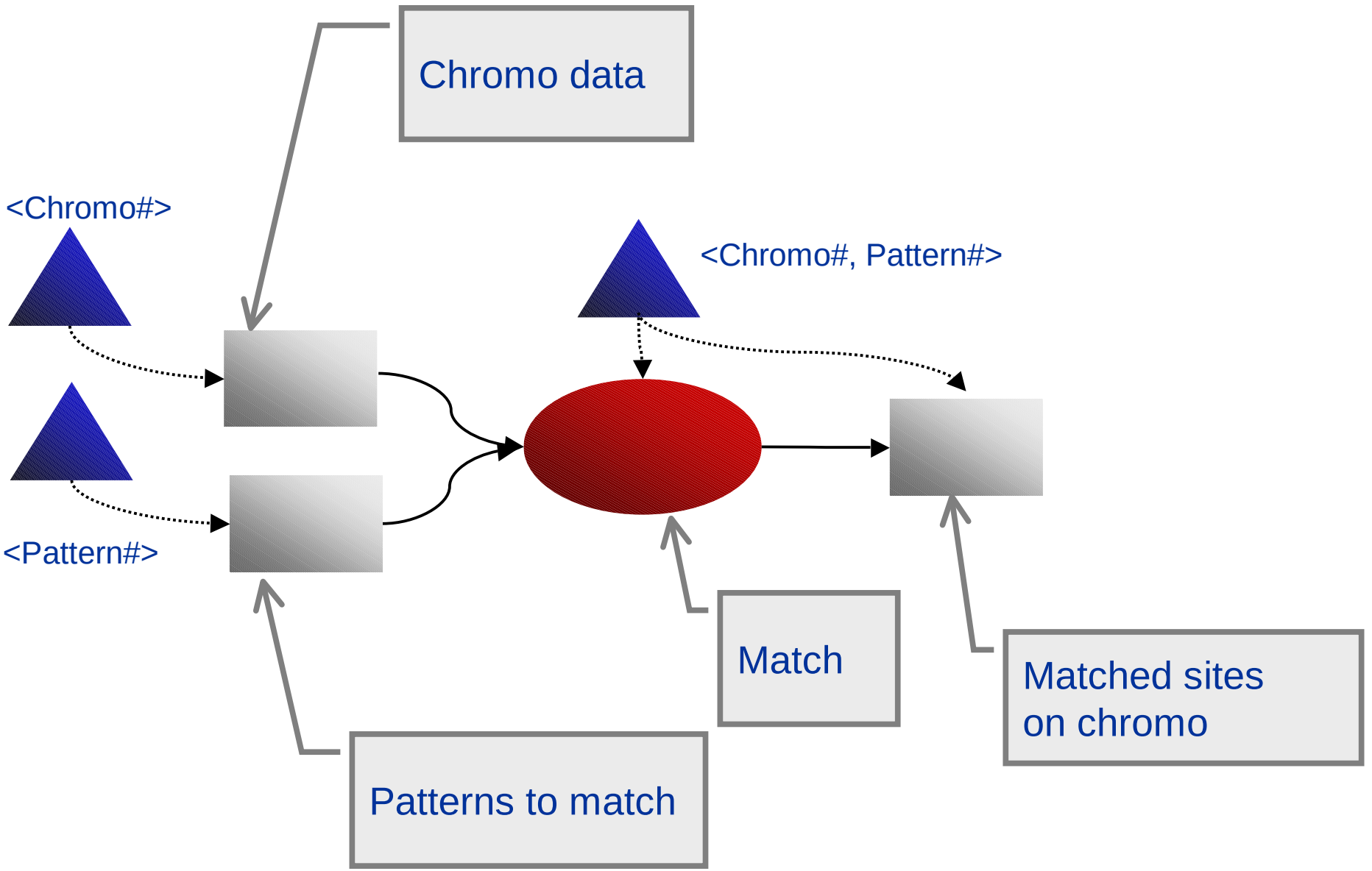
Background

- **Streams**
  - + very efficient
  - - limited to FIFO
- **Functional**
  - + scheduling flexible
  - - regular accesses not efficient
- **Tuple spaces**
  - + decoupled producer/consumer
  - - not functional: scheduling constrained by overwriting

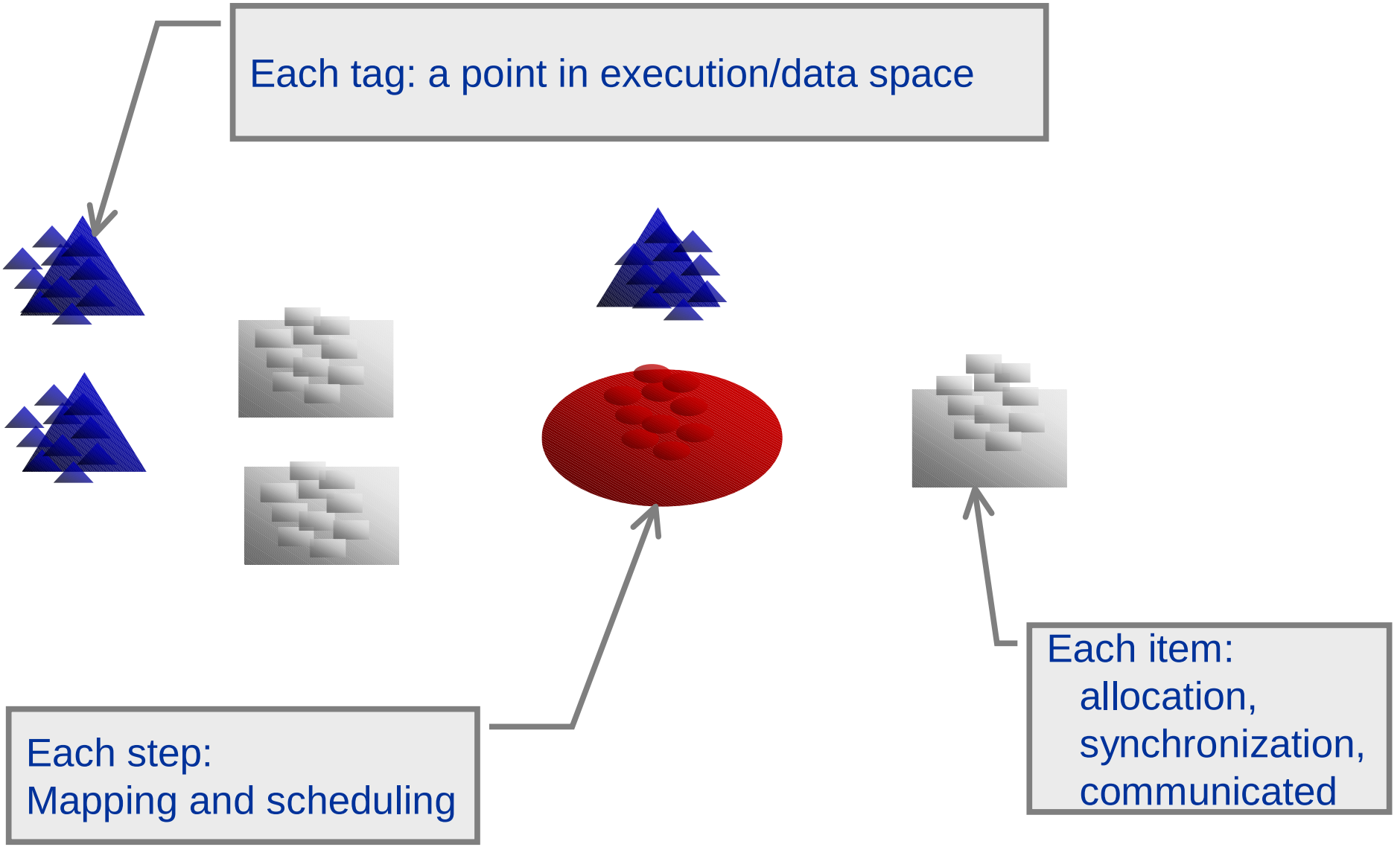
Static Components



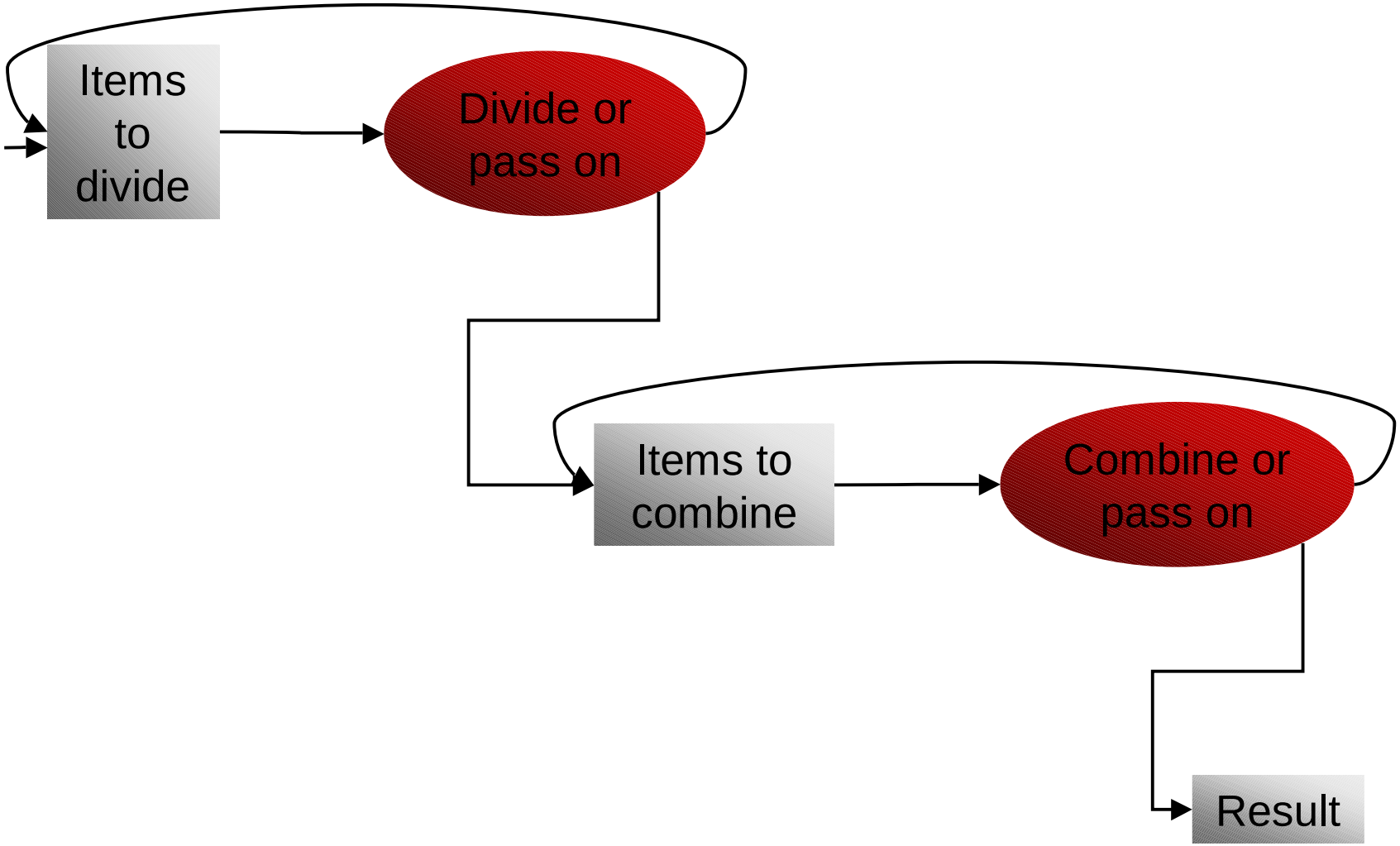
Example 1: Gene Matching



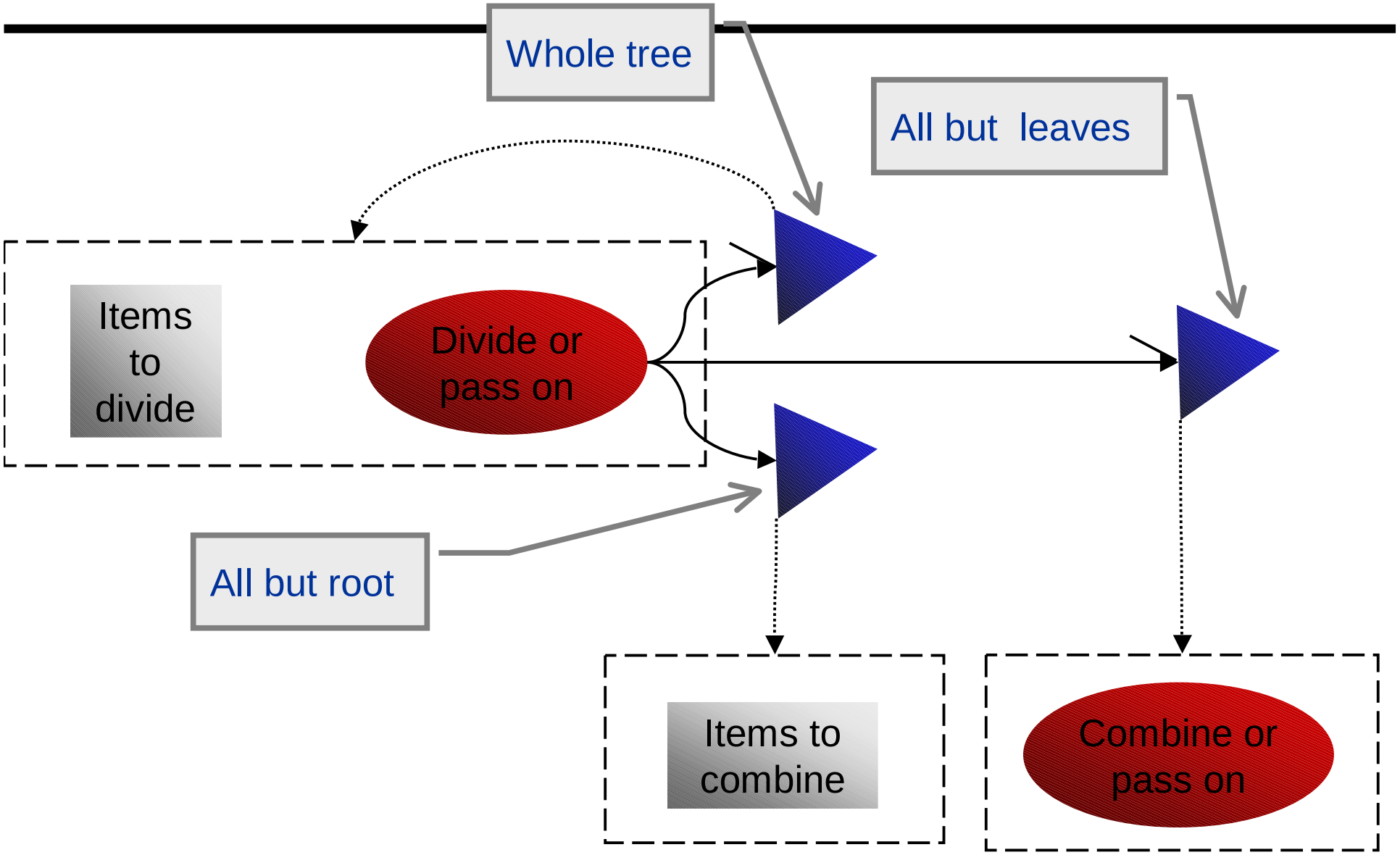
Dynamic Instances



Example 2: Divide & Conquer



Divide & Conquer

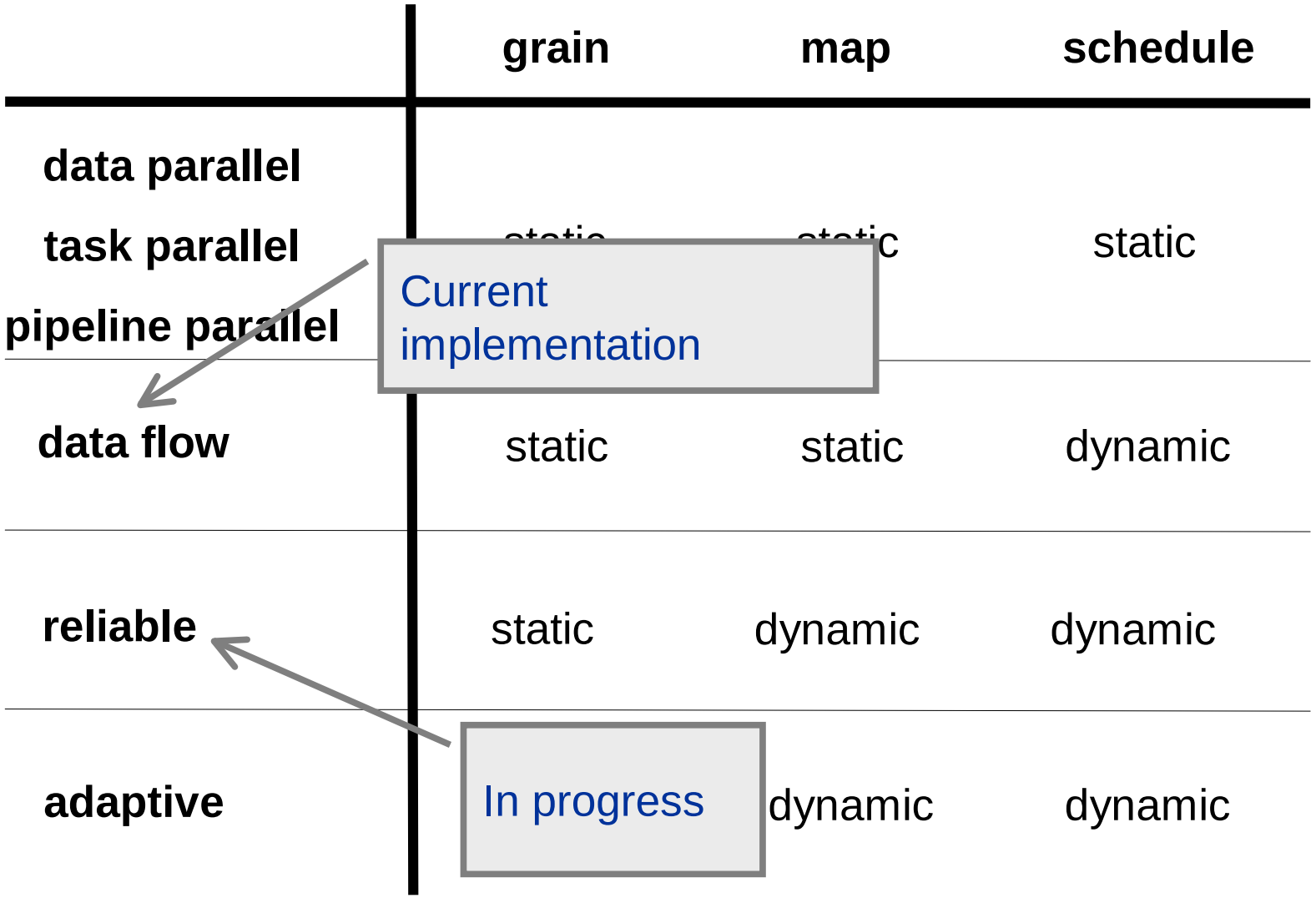


Properties

- State
  - tags in tag spaces, steps executed, produced items
- Atomicity
  - Tag: in a tag space or not
  - Step: executed or not
  - Item: produced or not
- Implications
  - redundant tag procuction, step execution, item production
  - start, drop and restart execution of steps

    start, drop and restart production of a tag or an item
- Dynamic, continuous, transparent checkpointing

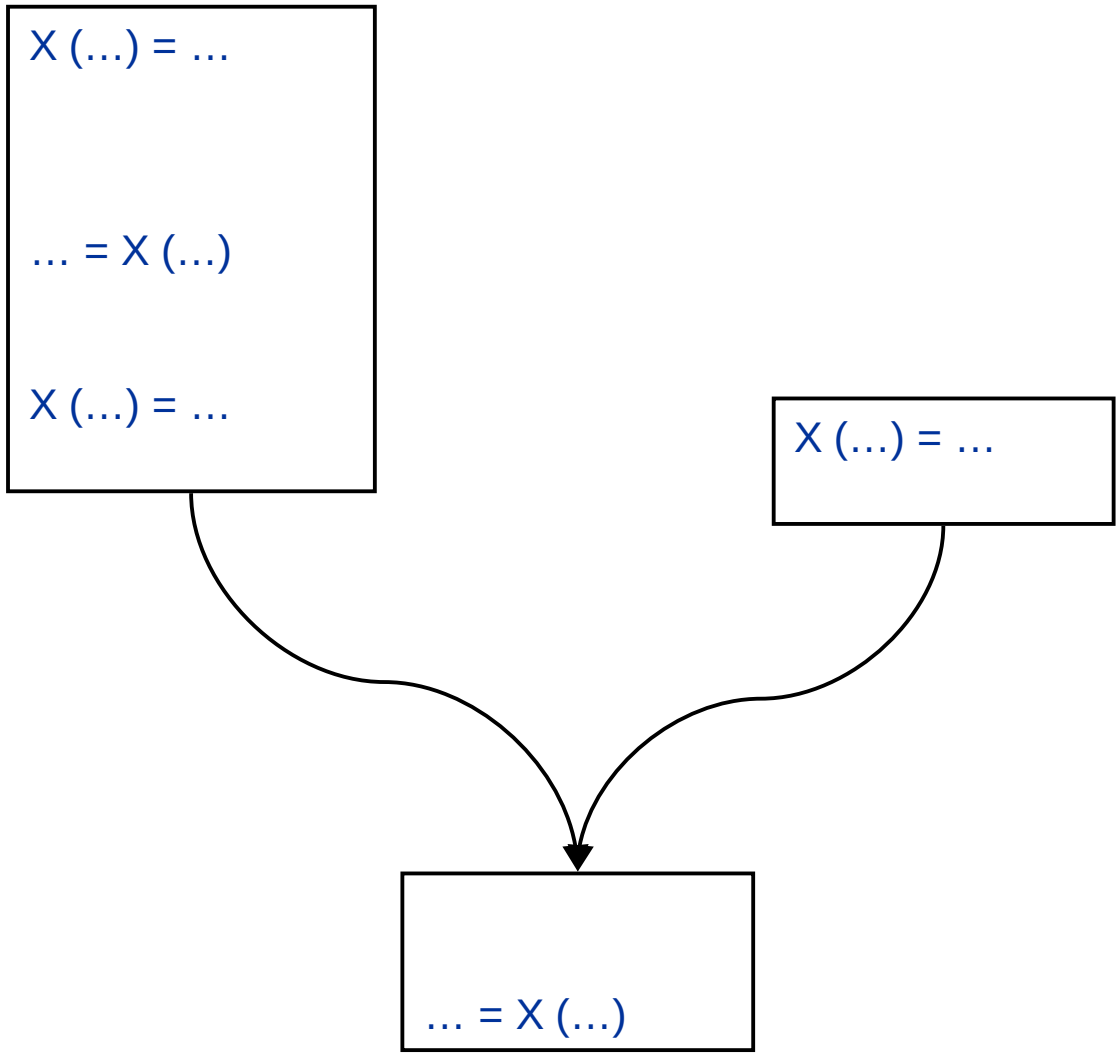
Execution Styles



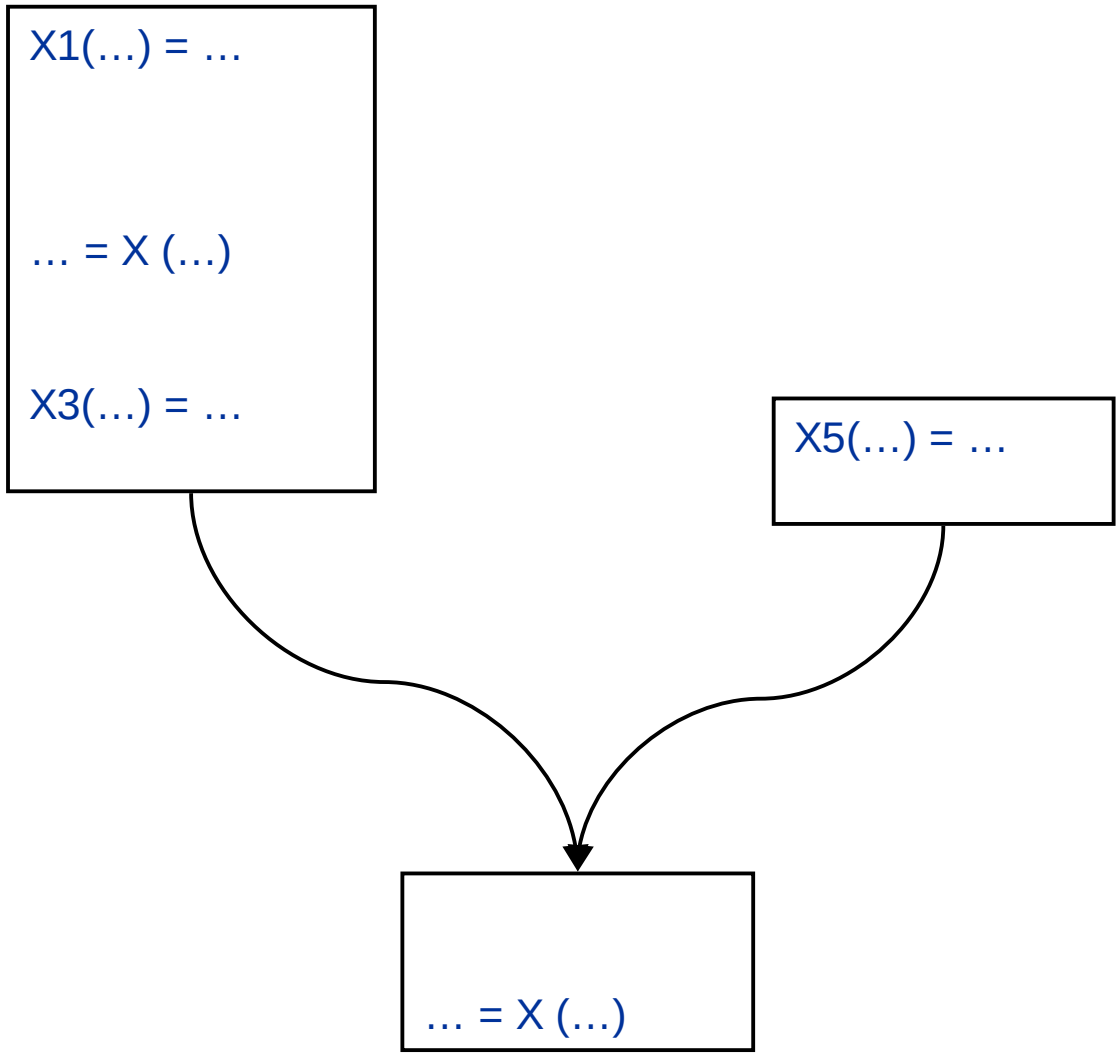
Compiler Algorithms: Transform Normal Programs to TStreams

- Subspace Analysis (Bill Dally)
- Array SSA (Vivek Sarkar)
- DSA (Carl Offner)

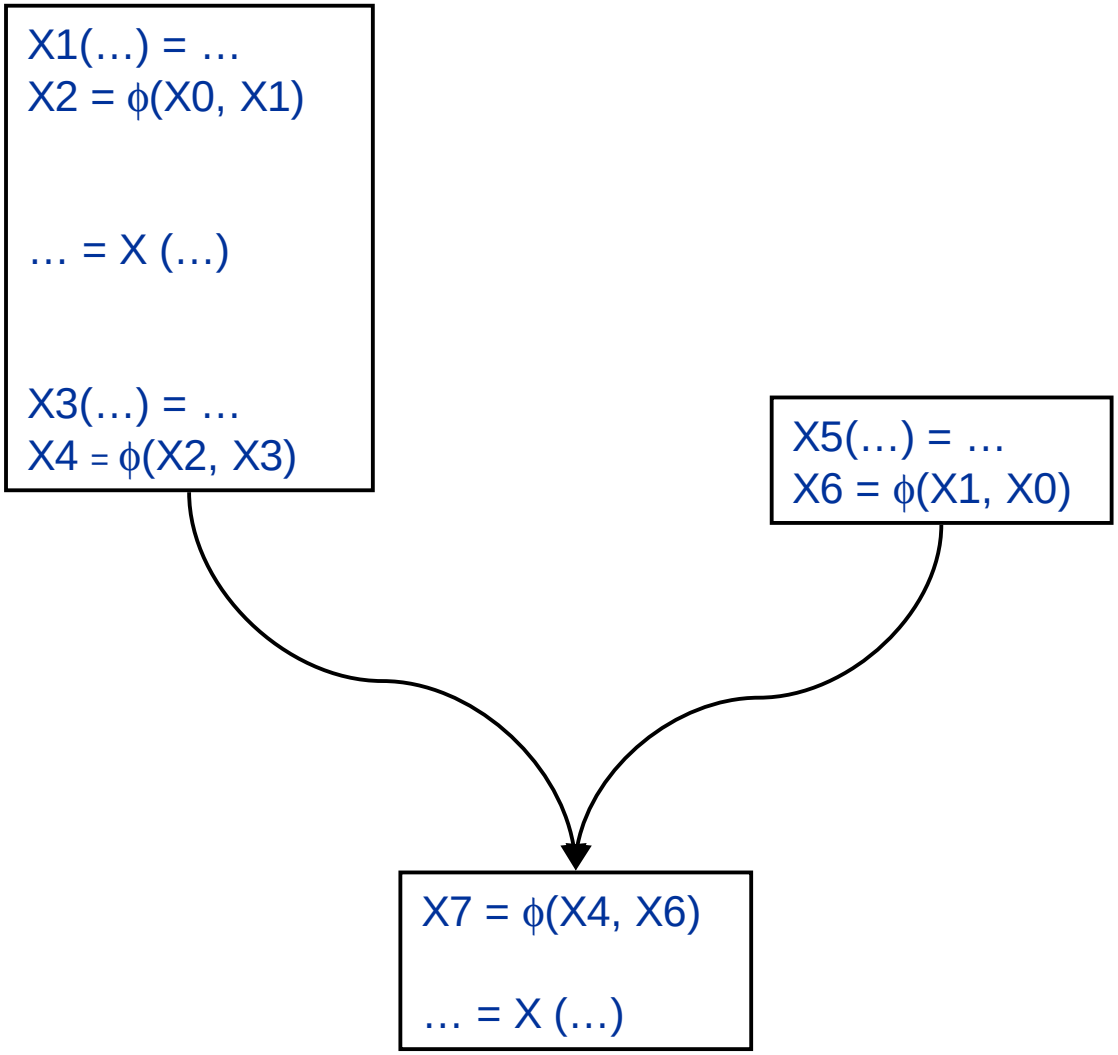
Original Code



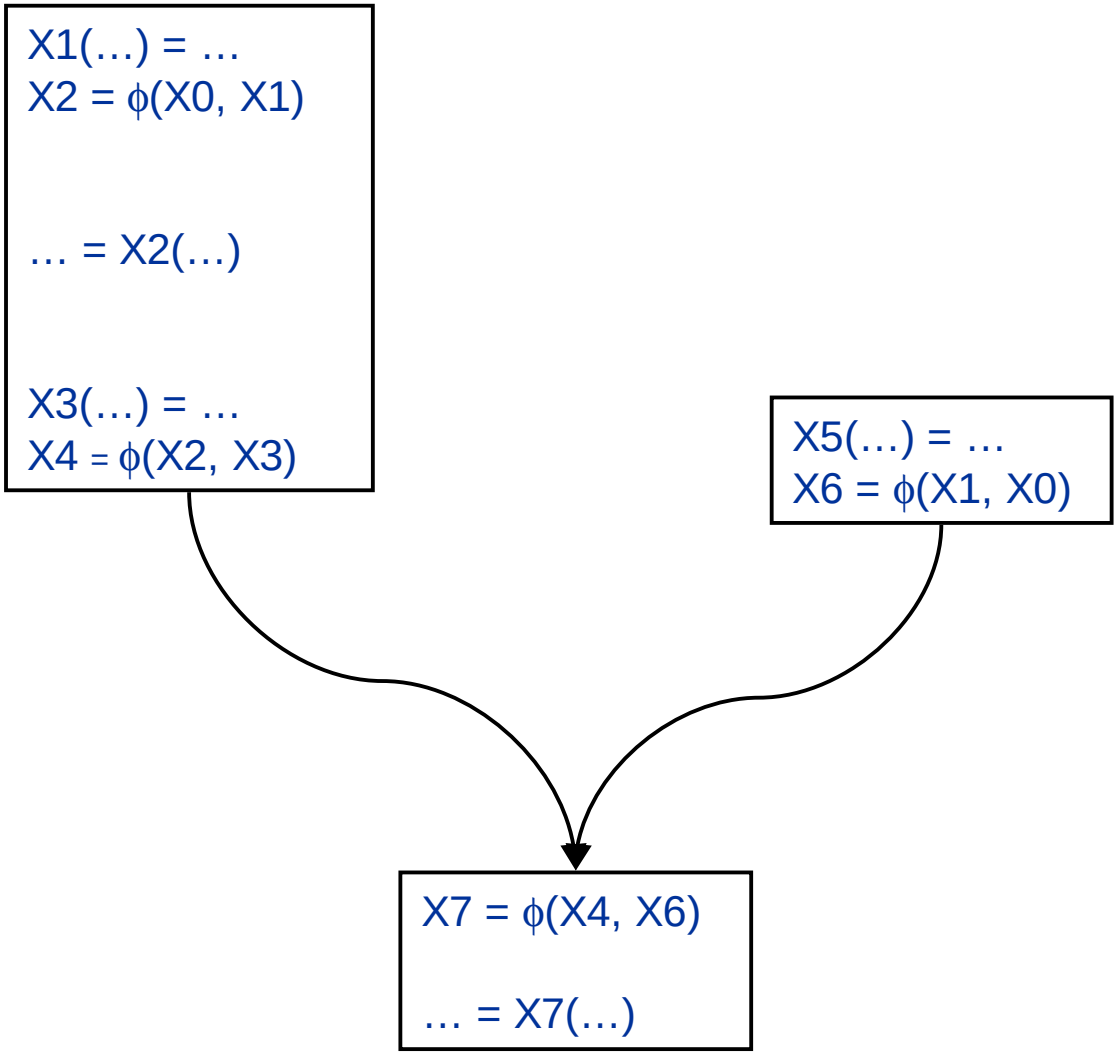
Static Single Assignment



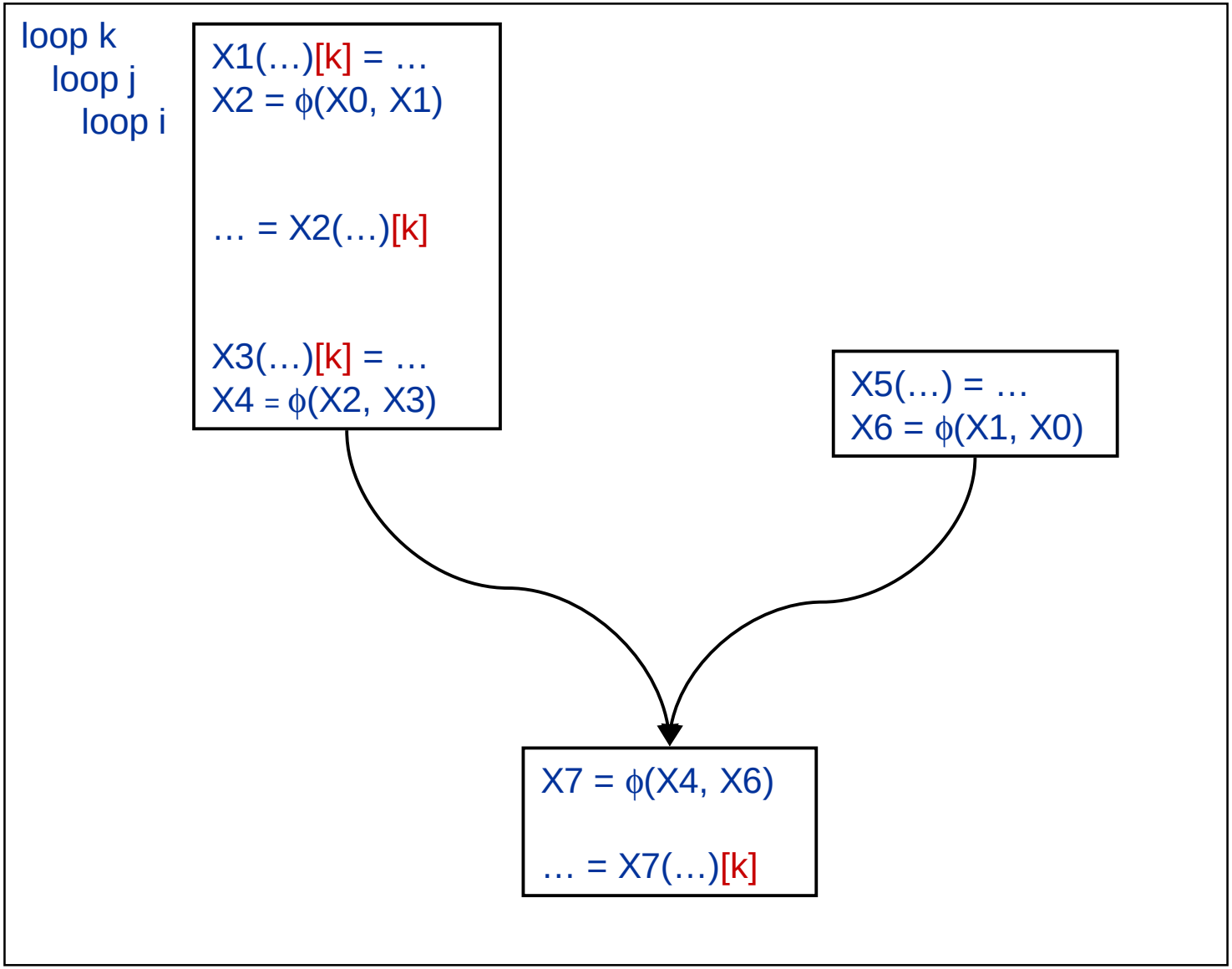
Static Single Assignment



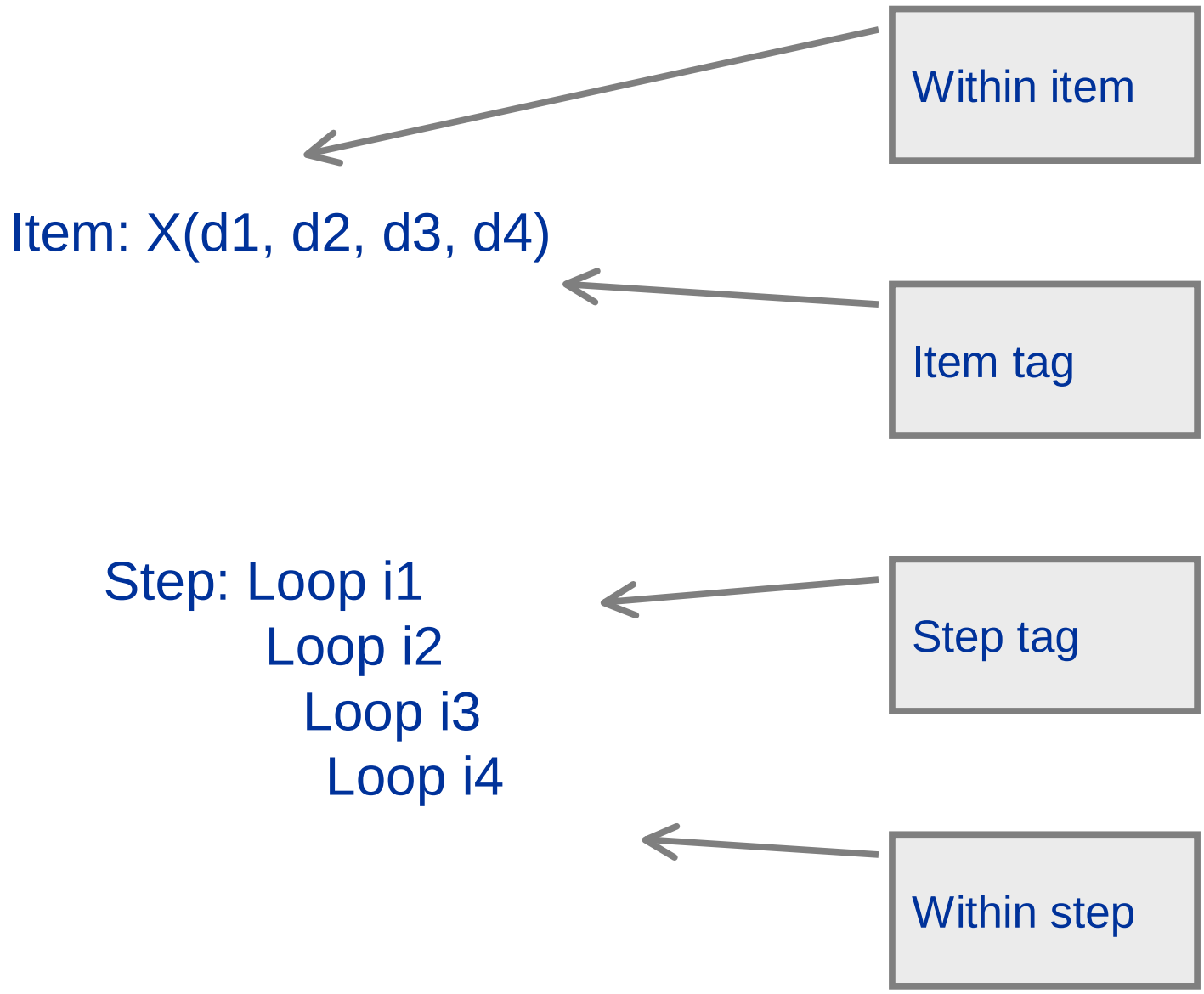
Static Single Assignment



Dynamic Single Assignment



Tags

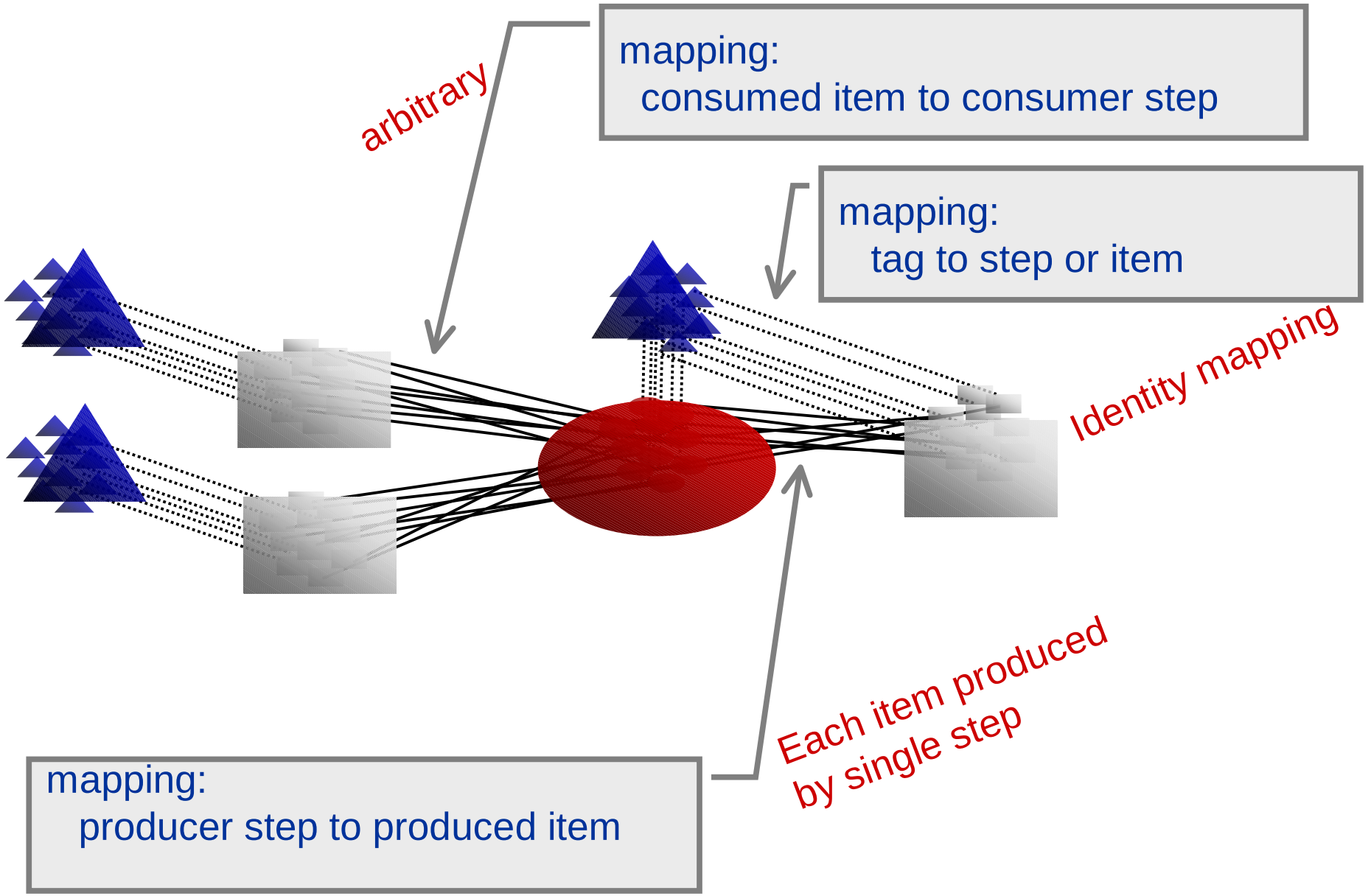

Within item
Item: X(d1, d2, d3, d4)
Item tag
Step: Loop i1
Loop i2
Loop i3
Loop i4
Step tag
Within step

Summary

- Best of streaming languages, functional languages and tuple-spaces
- Versatile execution styles
- Automatic generation
- (Hierarchical)
- (real-time)

END

Mappings


mapping:
consumed item to consumer step
arbitrary
mapping:
tag to step or item
Identity mapping
Each item produced
by single step
mapping:
producer step to produced item