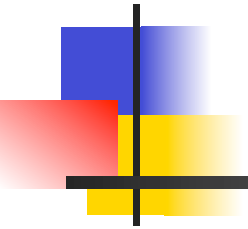


Model-Checker Case-Study

Alexandre David
1.2.05
adavid@cs.aau.dk

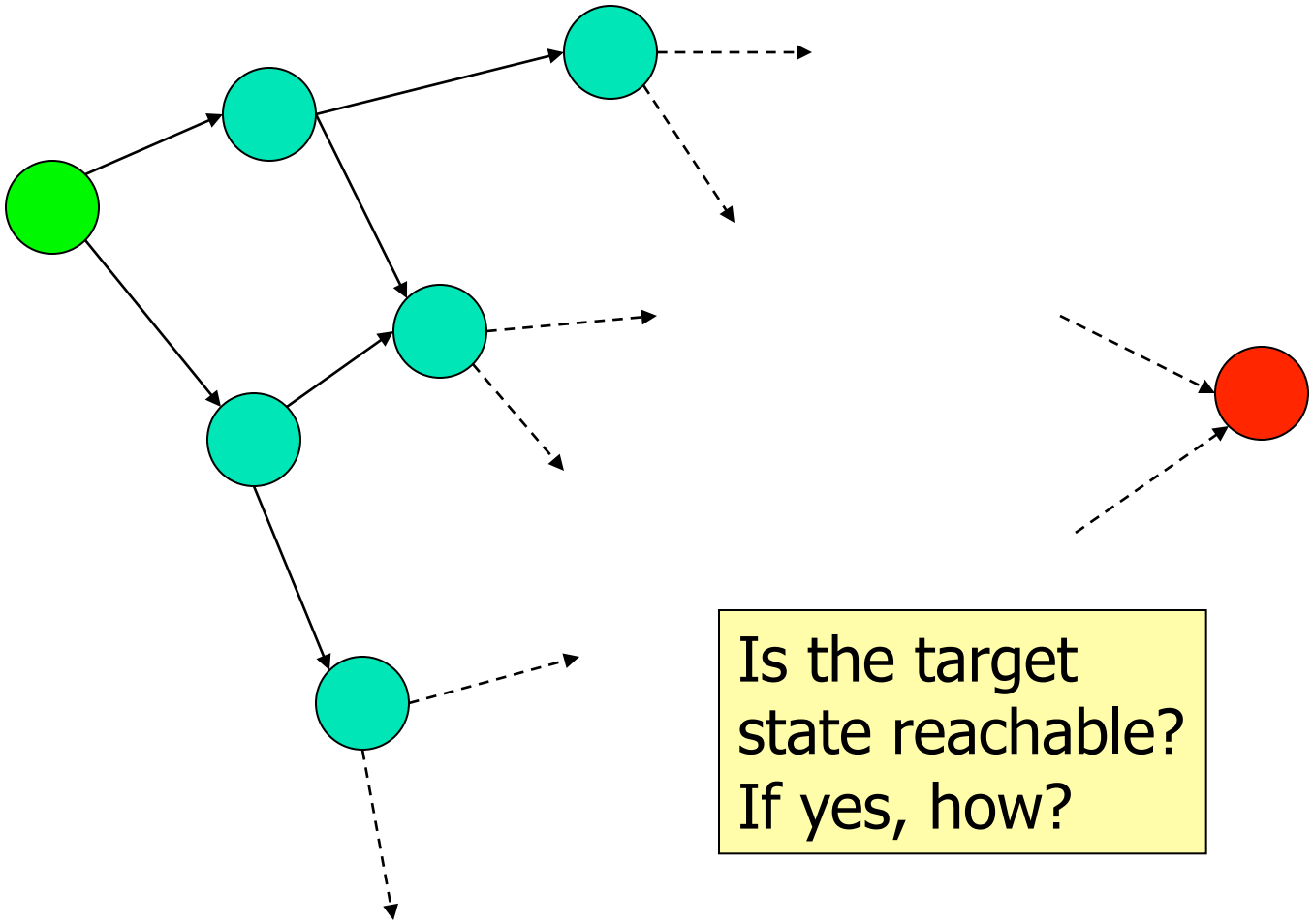
The Problem

- Application domain: Searching, planning, AI, scheduling, formal verification…
- Idea:
  - You make a **model** of a system. Description language = automaton/state-machine.
  - Your system changes its **state** according to a **transition relation** = set of rules that tell how the system may evolve.
  - Reachability problem: Given an initial state, how to reach a goal state?
  - Technique: Explore the **state-space**.

Definitions

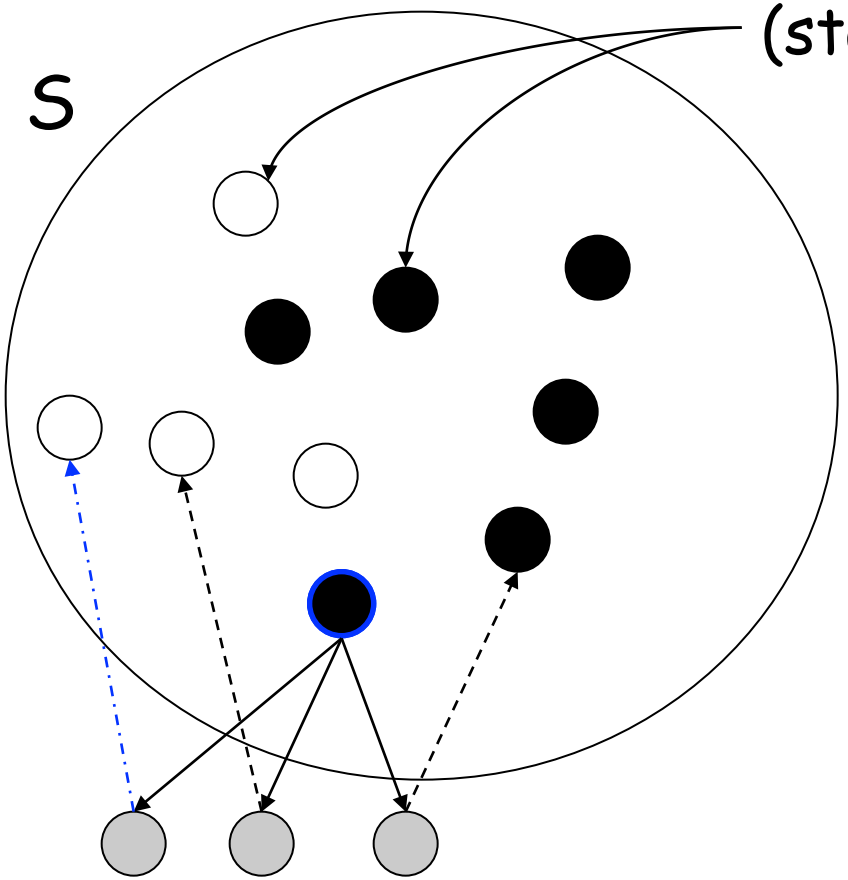
- A state is the snapshot configuration of a system.
- The system changes state by taking **transitions**. The rules are given by a transition relation.
- The set of all states is called the state-space.
- A state S is reachable if there exists a sequence of transitions from the initial state to S.
  - This sequence of transition is called trace, path, or witness.

State-Space Exploration



Is the target state reachable? If yes, how?

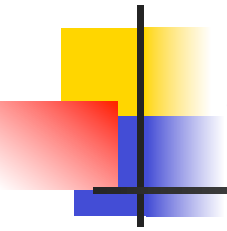
Exploration Algorithm



not explored (waiting)

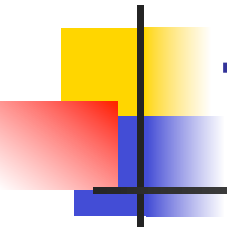
explored (visited)

1: Pick white.
2: Mark it black.
3: Generate its successor states.
4: Add them to S.
5: Mark them white.
6: Repeat until find the goal or no more white state to pick.



Correctness

- The algorithm explores all possible reachable states.
  - It will terminate if the state-space is finite. This is our case.
  - When it terminates, it proves that a state is reachable or not.
- Problem: State-space explosion.



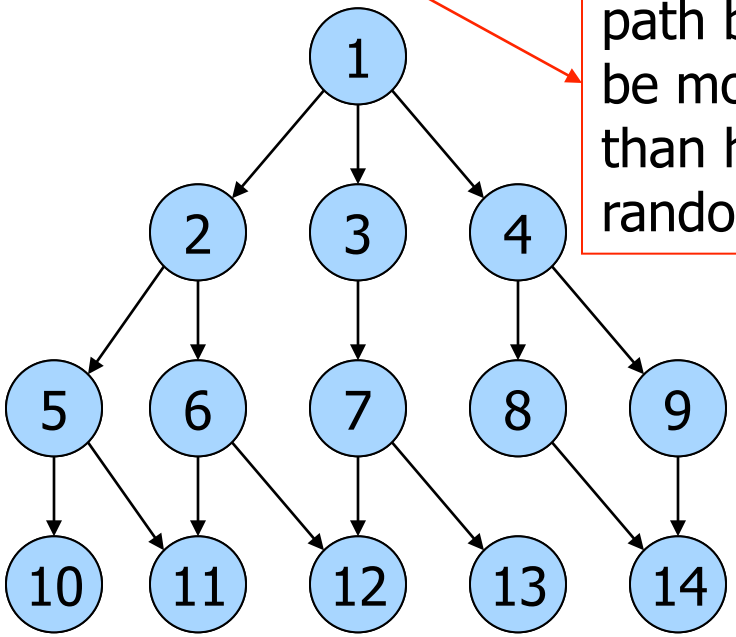
Technicalities

- How to represent S for efficient look-up?
  - Hash table.
- How to pick-up the next state to be explored?
  - FIFO: Breadth-first-search.
  - LIFO: Depth-first search.
  - Priority queue: Guided search with heuristics.

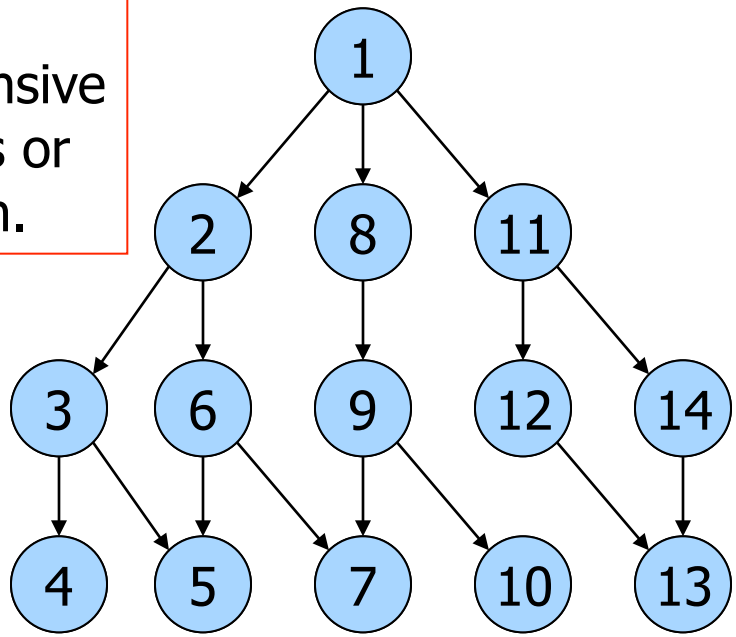
Search Orderings

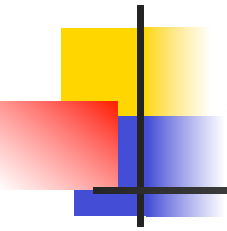
Breadth-first-search (BFS)

Gives shortest path but may be more expensive than heuristics or random search.



Depth-first-search (DFS)





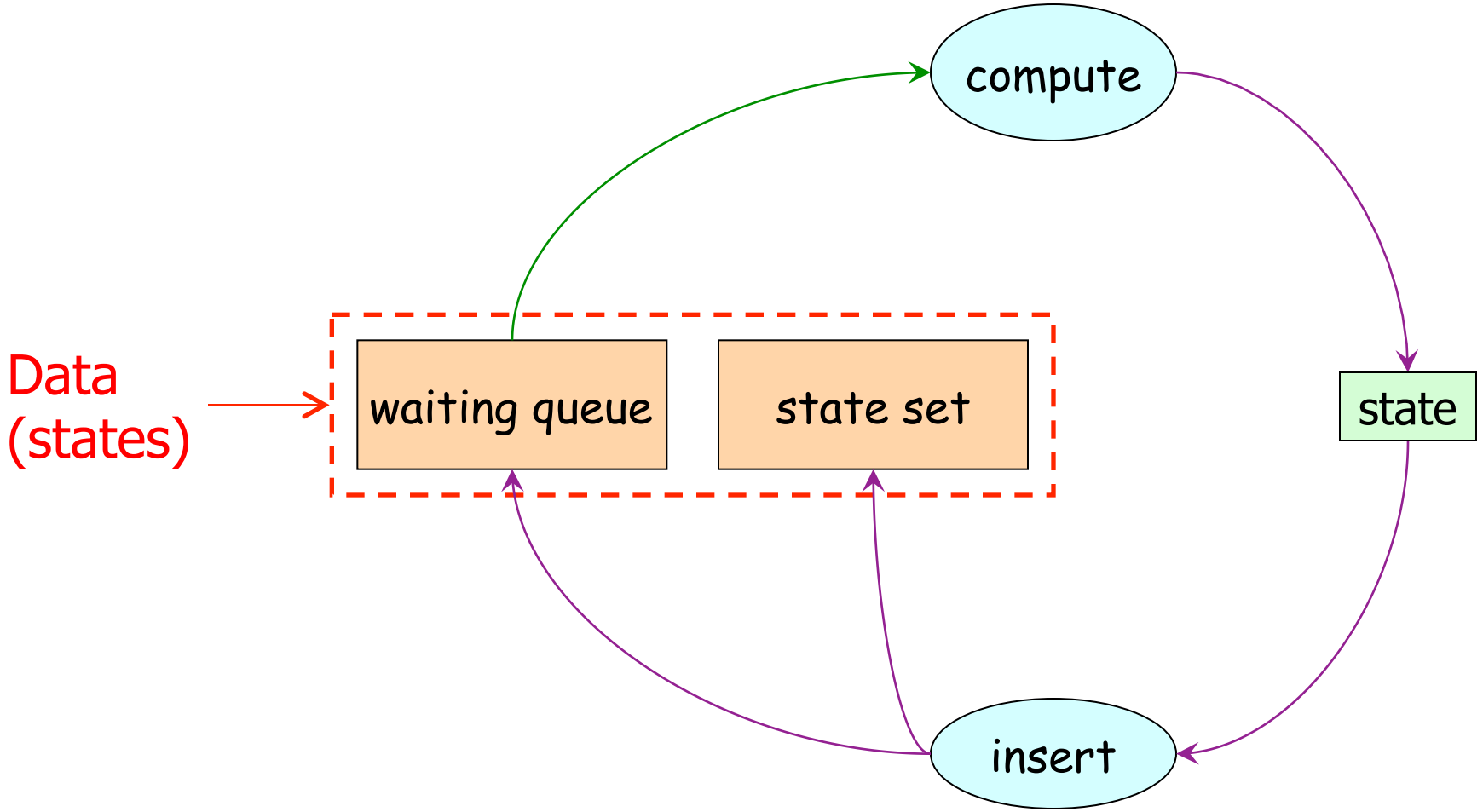
Classification

- Dynamic partitioning.
- Dynamic load balancing.
- Performance anomalies expected.
- Correctness issues w.r.t. search orderings.

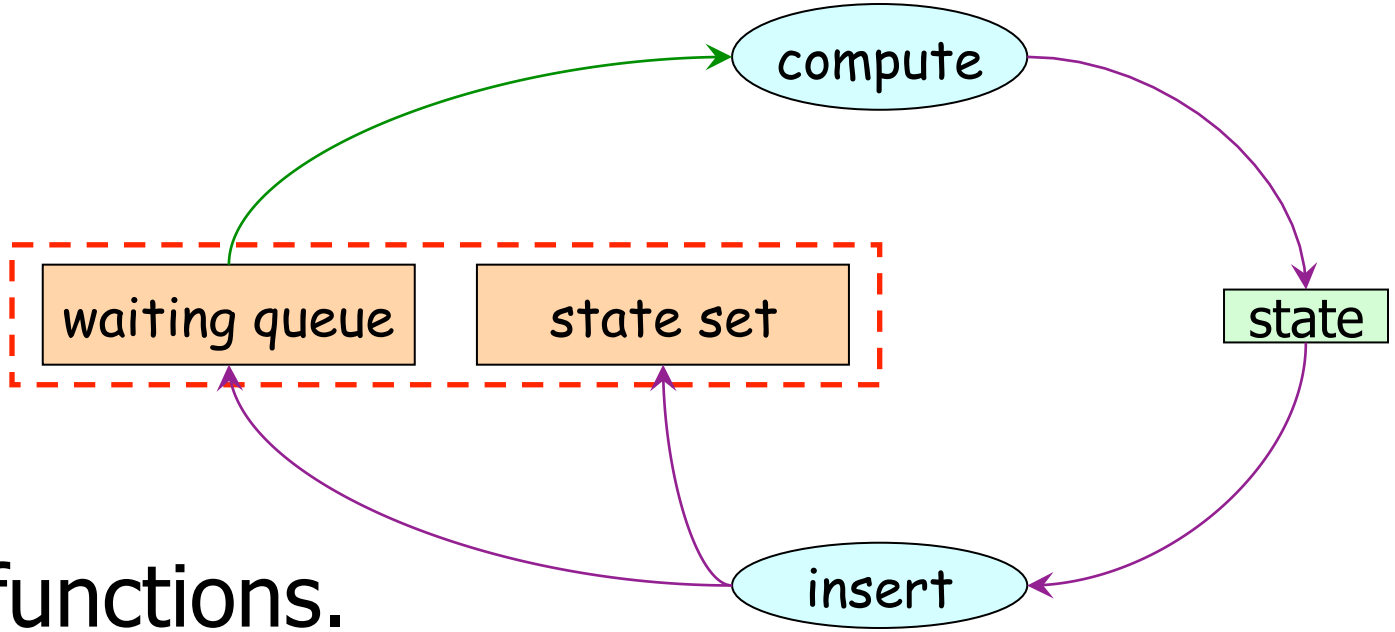
Basic Problems

- Where are the data?
  - Find the dataflow – data & functions.
- Which computations can be done in parallel?
- Identify critical sections.
- What data can be shared?
- How to solve load balancing?
- How to detect termination?

Simplified Dataflow

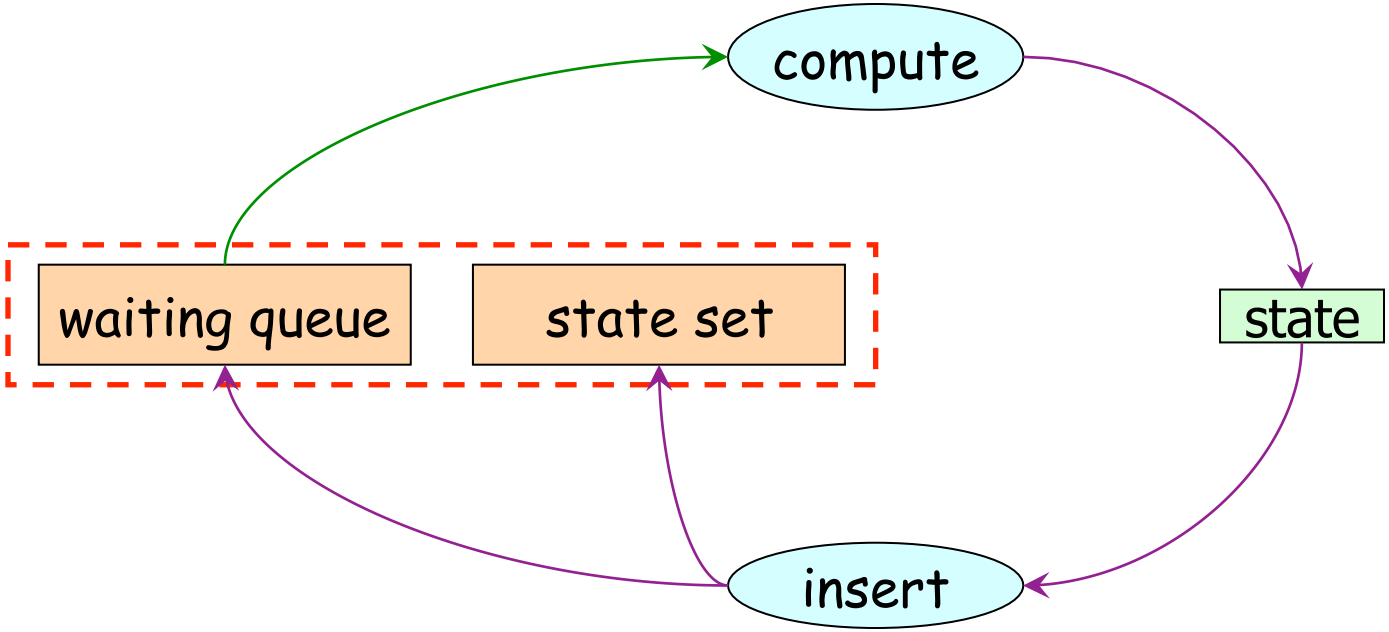


Which Computations Can Be Done in Parallel?

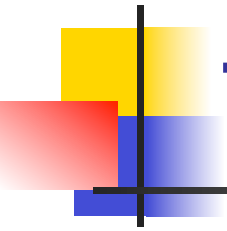


- All functions.
- Critical sections:
  - read & write to shared data.

Shared Data

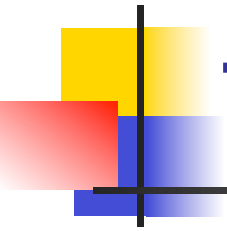


- Queue & state-set.
  - Evenly distributed among processes.
  - Load balancing through (universal) hash. Owner computes rule.



Termination Issues

- How to detect it is finished?
  - Load dynamic.
  - Work dynamic.
  - Quiescence now does not mean finished.
- How would you do it?

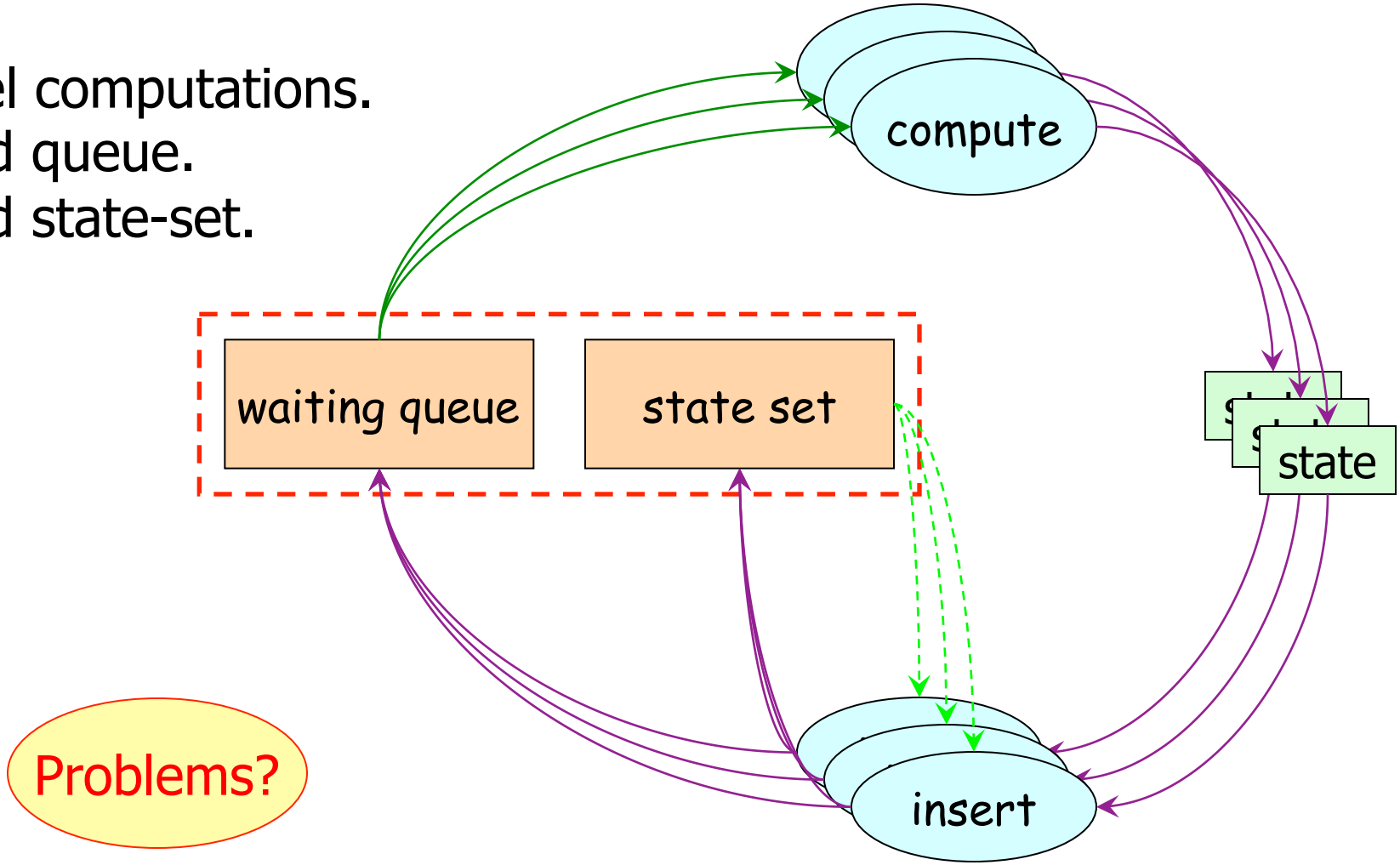


Termination Issues

- Detect that all processes are idle.
  - If process A is idle but B is working: no.
    - If B sends something to A and then becomes idle: no.
  - All processes idle and no data in transit: yes.
- Barrier protocol – principle:
  - Processes block on empty queues,
  - the last process detects termination.
  - Race condition issues
    - pthreads: condition synchronization.
    - MPI: distributed token based protocol.

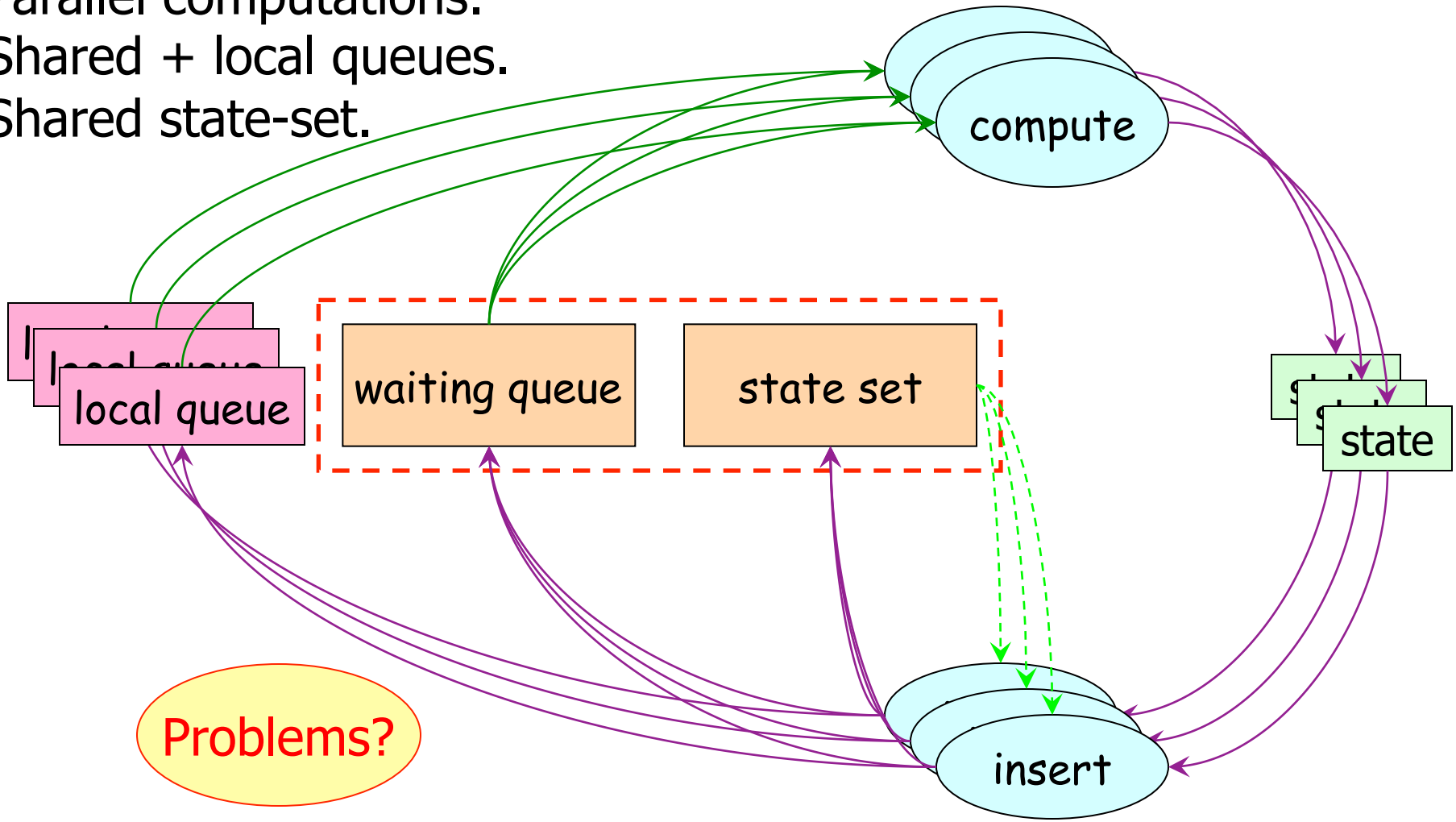
PThreads – 1

Parallel computations.
Shared queue.
Shared state-set.



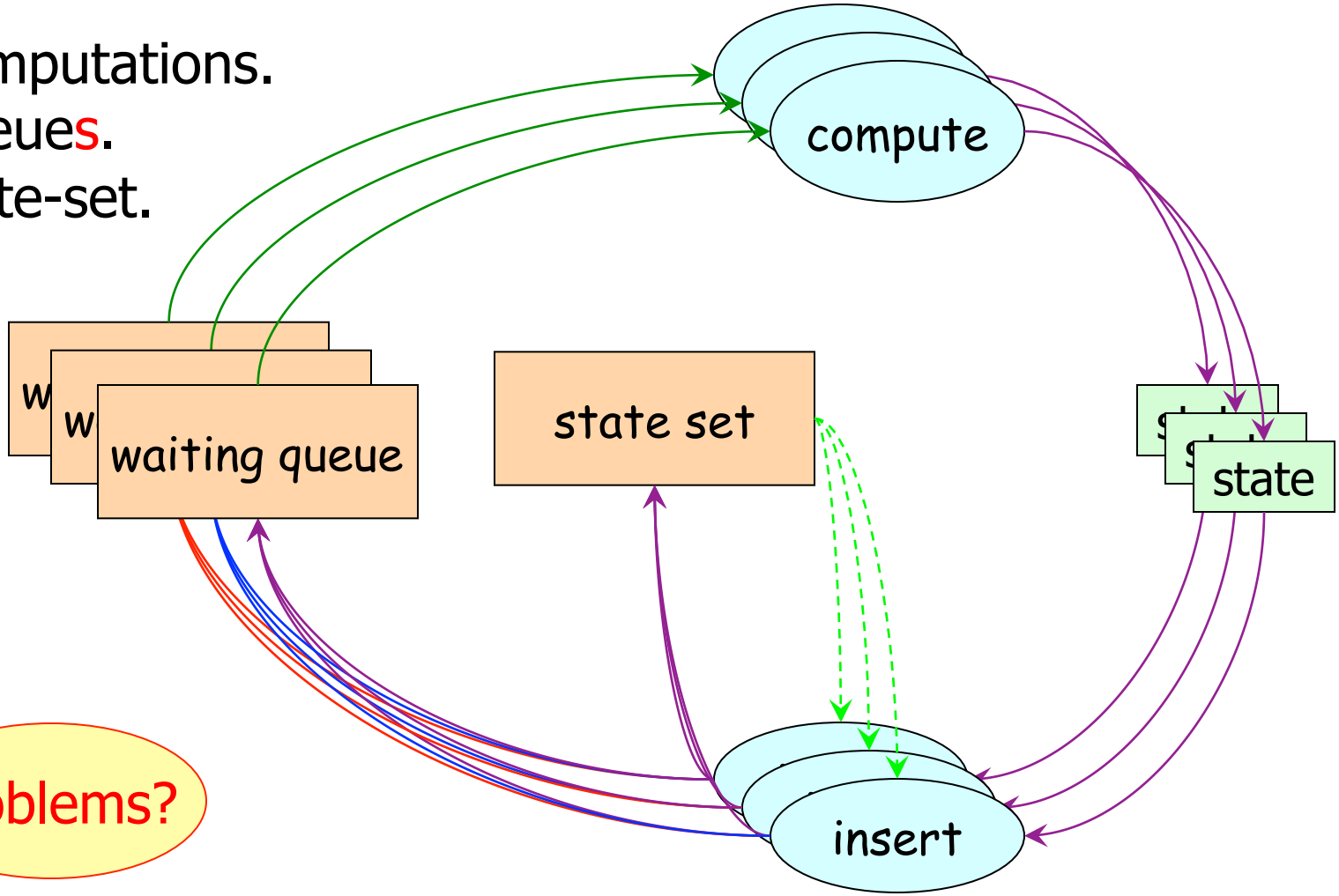
PThreads – 2

Parallel computations.
Shared + local queues.
Shared state-set.



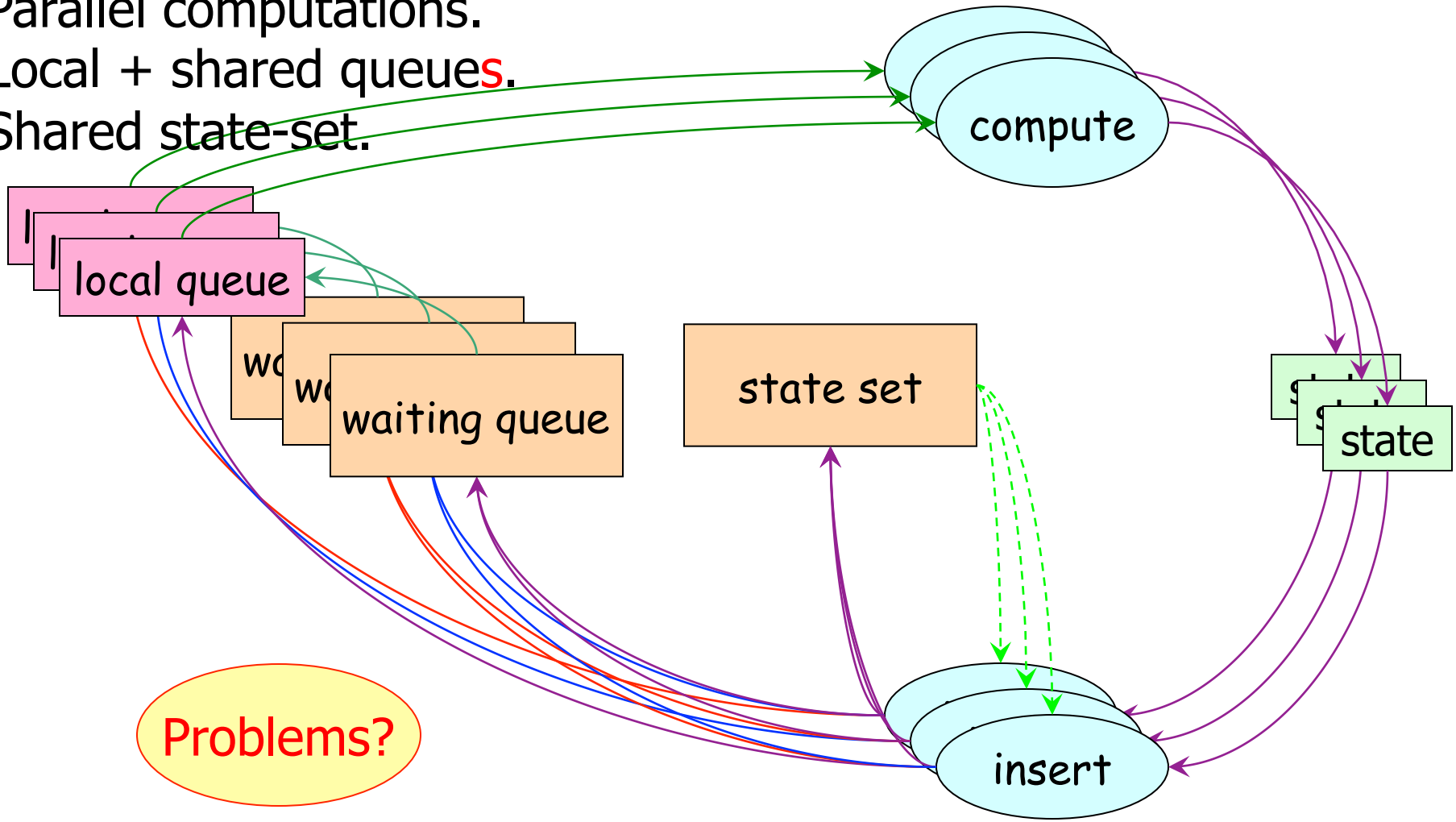
PThreads – 3

Parallel computations.
Shared queues.
Shared state-set.



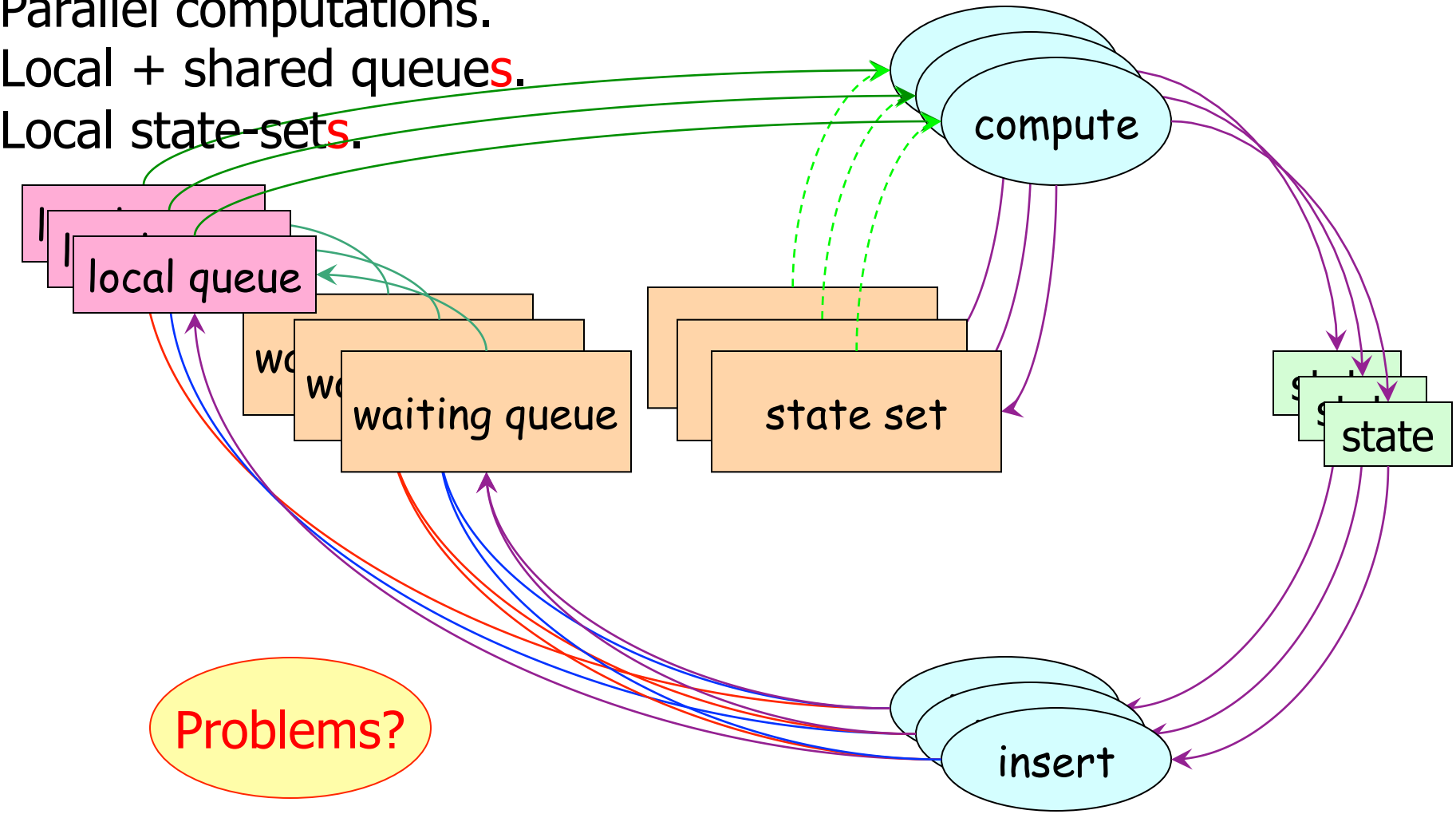
PThreads – 4

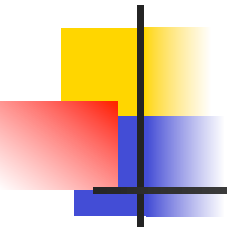
Parallel computations.
Local + shared queues.
Shared state-set.



PThreads – 5

Parallel computations.
Local + shared queues.
Local state-sets.





Issues

- Contention.
- False sharing.
  - Data.
  - Locks!!!
    All the threads will want to lock all the locks.
- Detect termination! (overhead)
- Solutions: tryLock, lock on hash entries.
- Poor speedup, not efficient.
- Alternative: non-blocking shared data-structures!