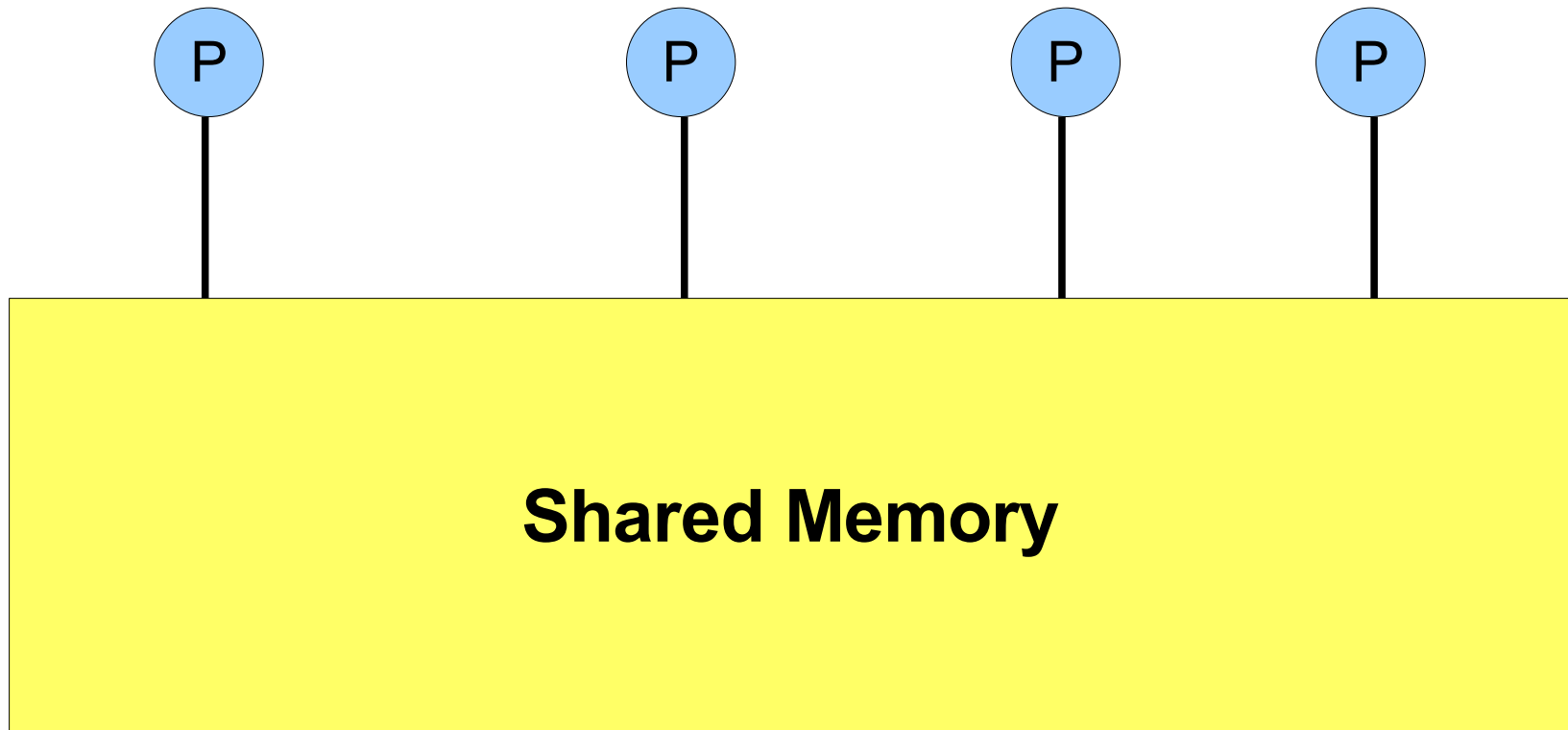


# Predictive Log-Synchronization

- Ori Shalev
  - Tel Aviv University, Sun Microsystems
- Nir Shavit
  - Sun Microsystems

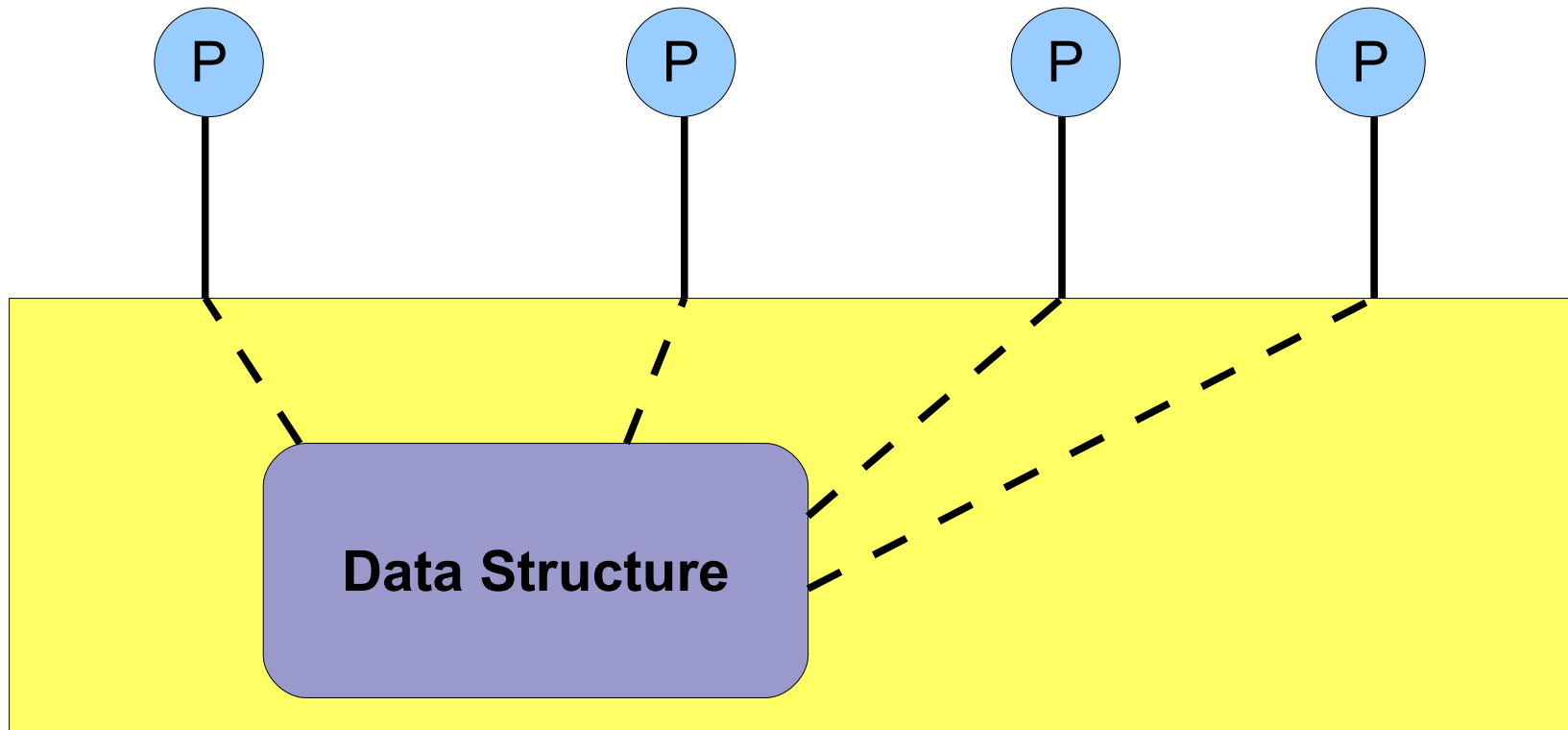
# Concurrent Data Structures

Shared-memory multiprocessor



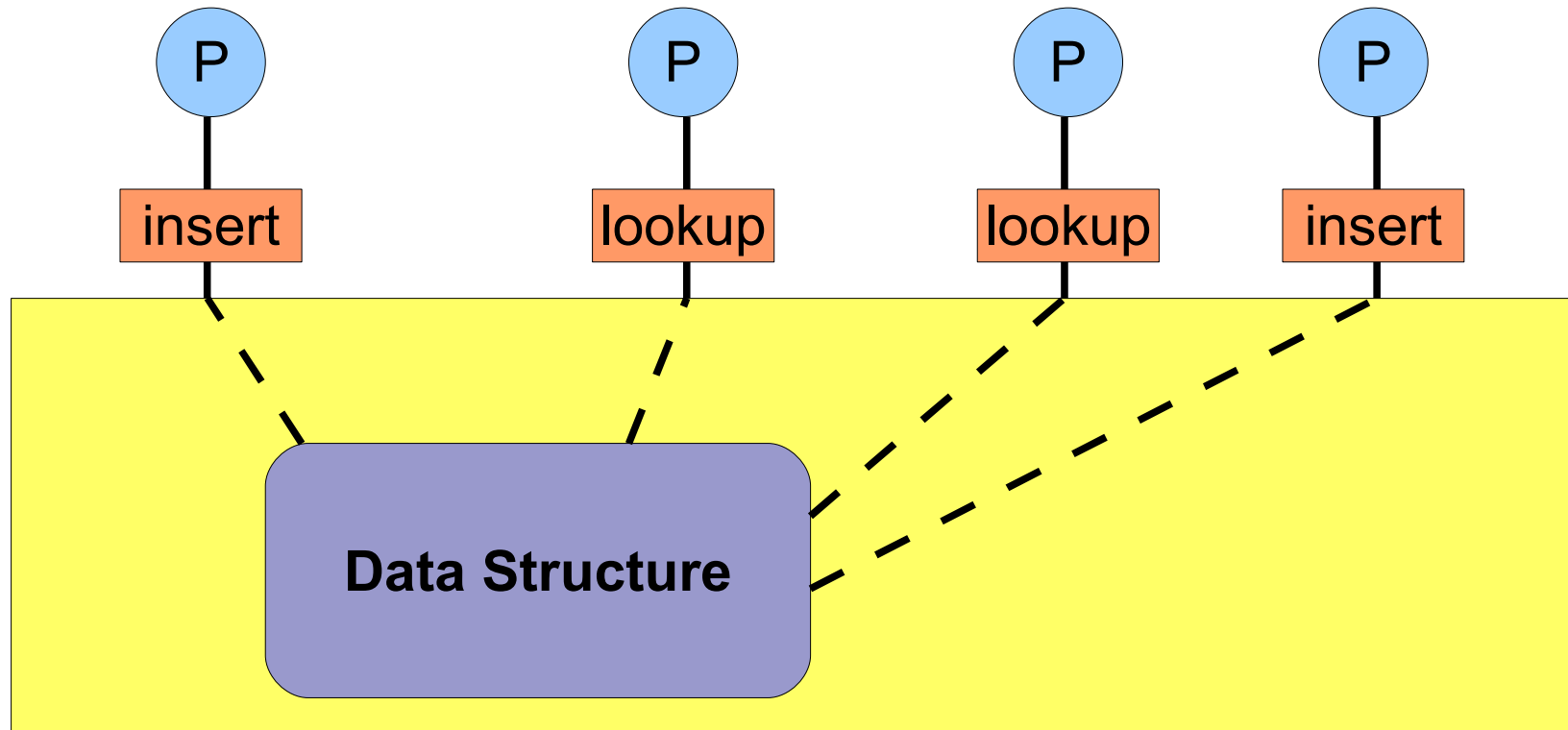
# Concurrent Data Structures

Shared-memory multiprocessor



# Concurrent Data Structures

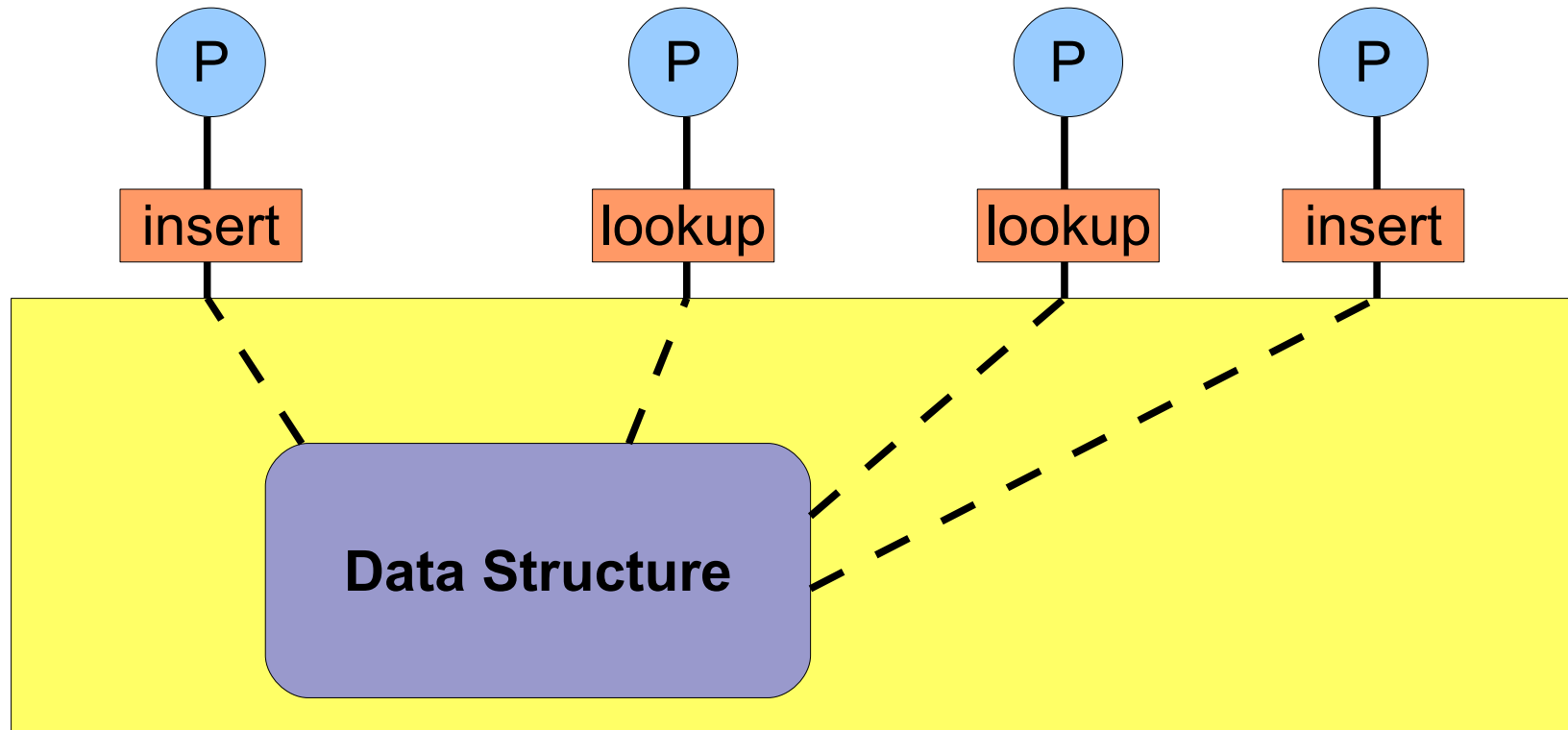
Shared-memory multiprocessor



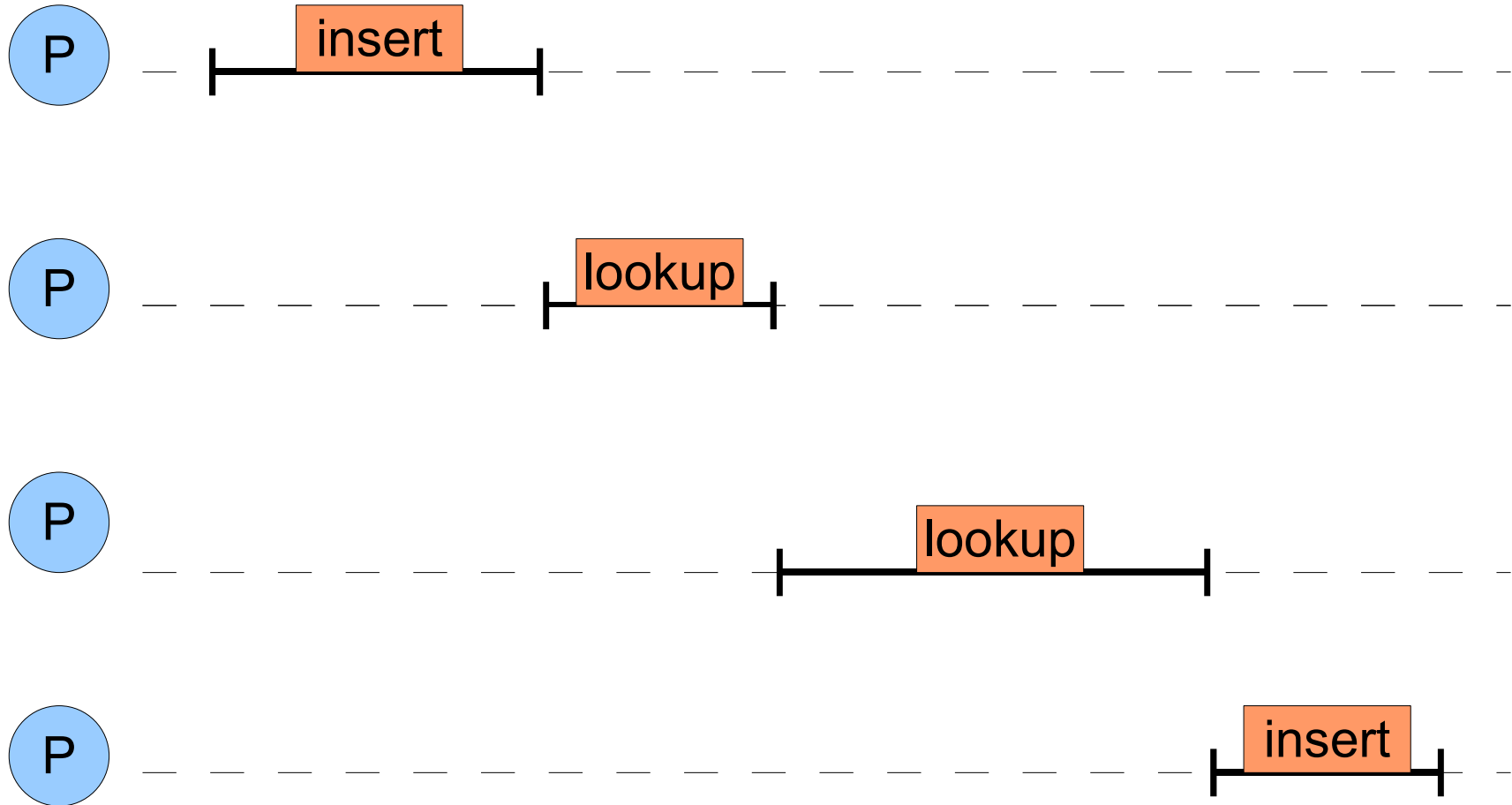
# Concurrent Data Structures

Shared-memory multiprocessor

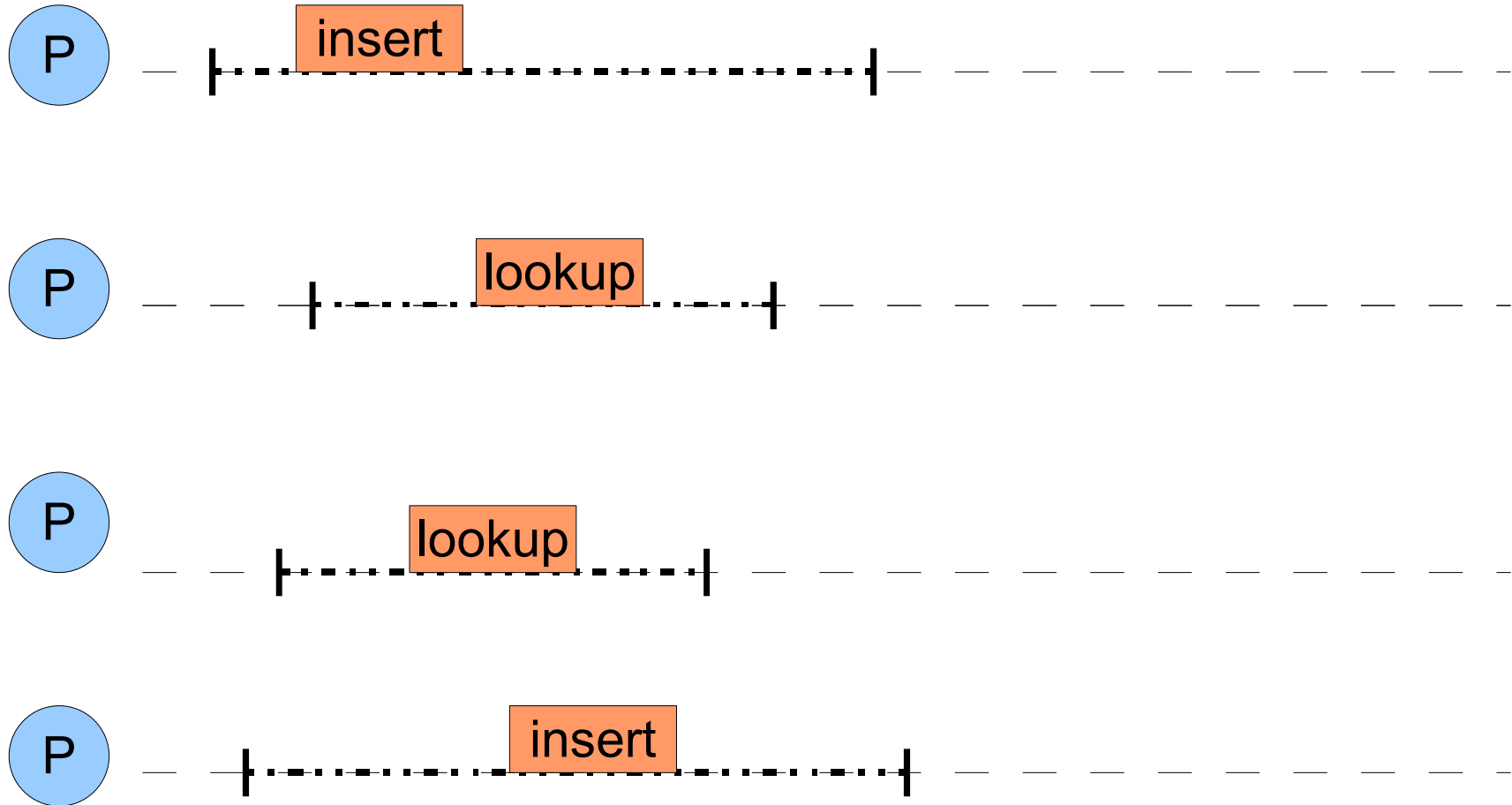
High % of **Read-only** operations



# Coarse-Grained Locking



# Fine-Grained Locking



# Motivation

## Concurrent programming

**Coarse** grained  
locks

**Fine** grained  
locks

programmability

performance

# Motivation

## Concurrent programming

**Coarse** grained  
locks

**Fine** grained  
locks

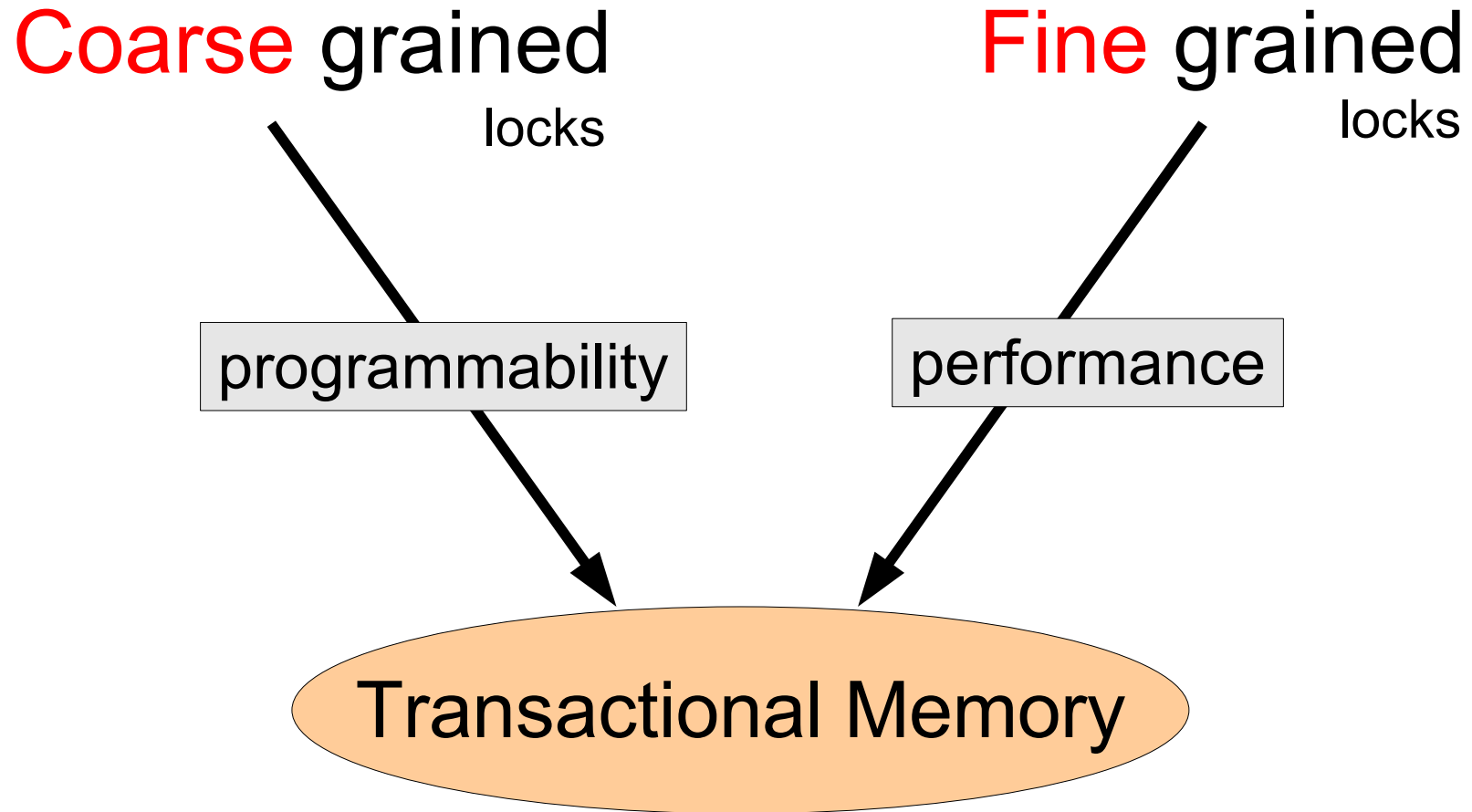
programmability

performance

?

# Motivation

## Concurrent programming



# Motivation

## Concurrent programming

Coarse grained

locks

programmability

still concurrent prog.

Fine grained

locks

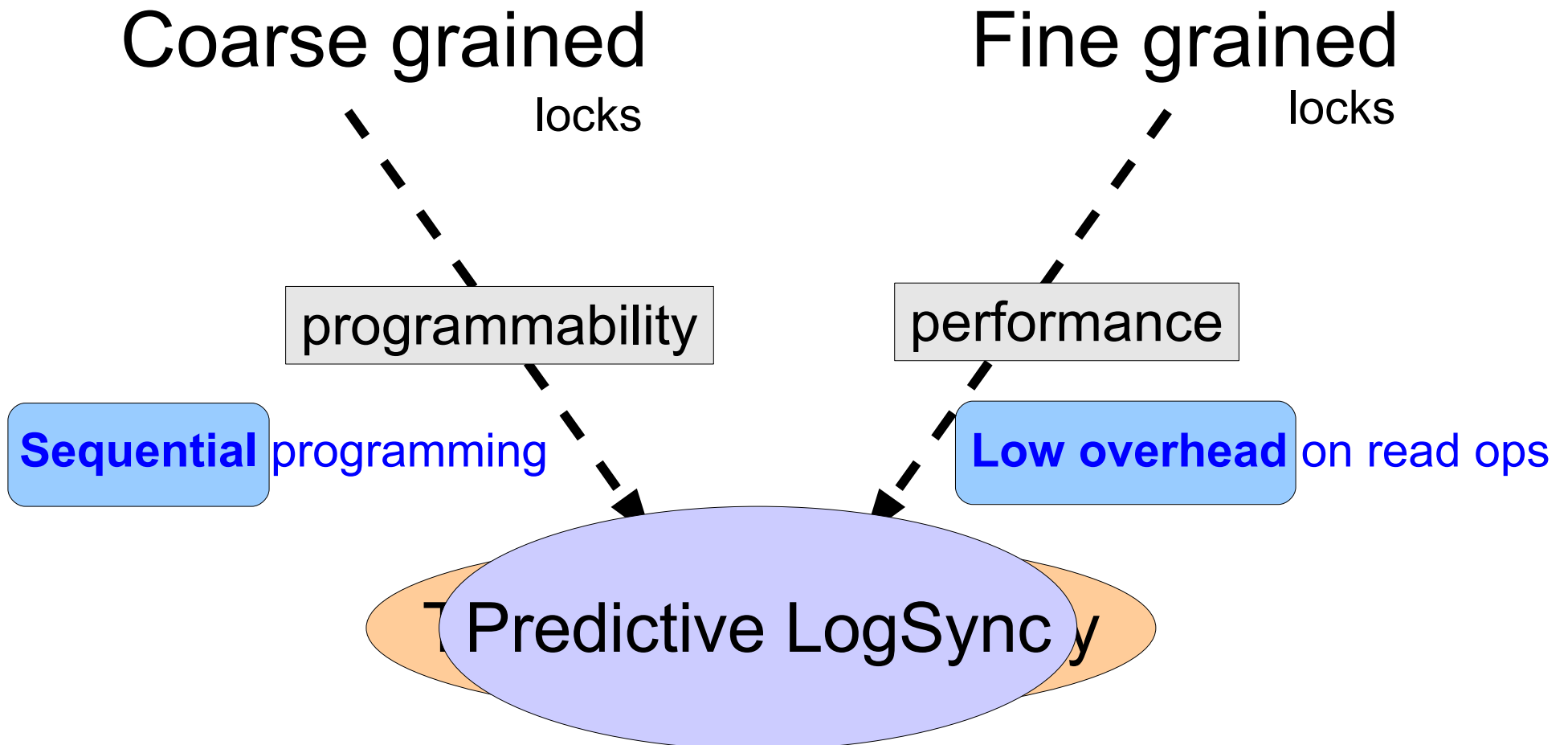
performance

overhead OR bounded

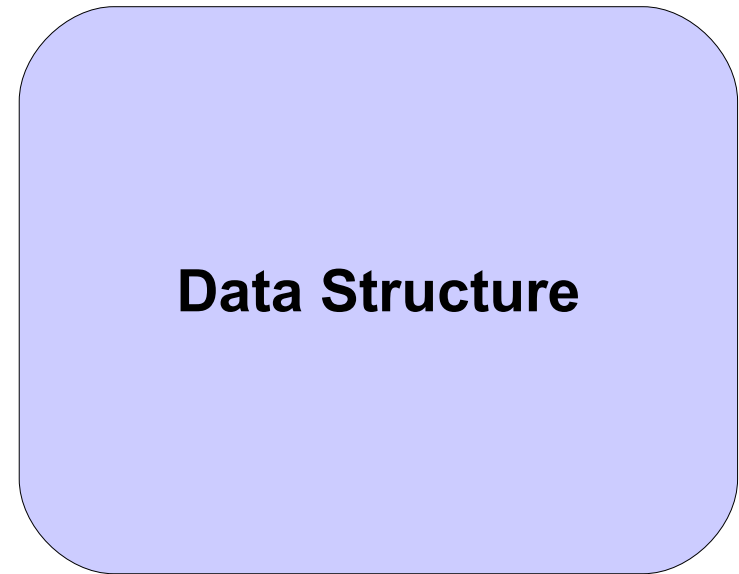
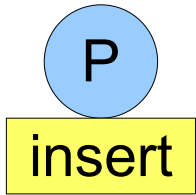
Transactional Memory

# Motivation

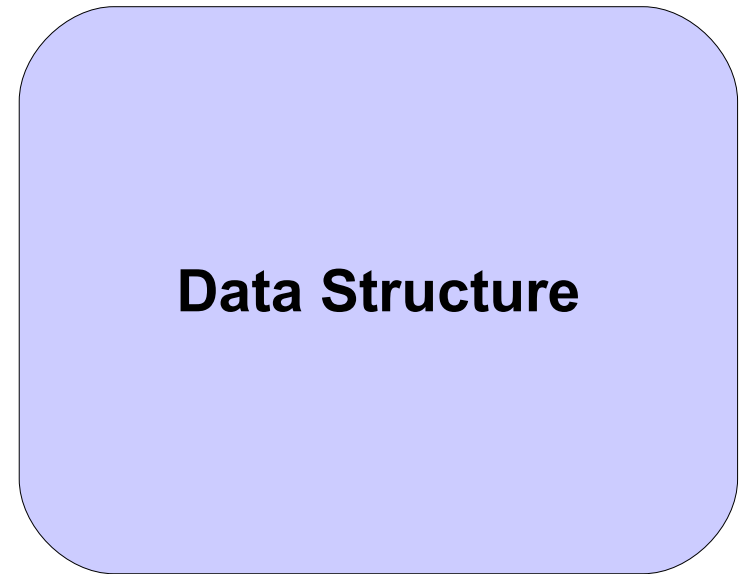
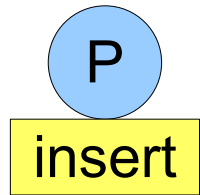
## Concurrent programming



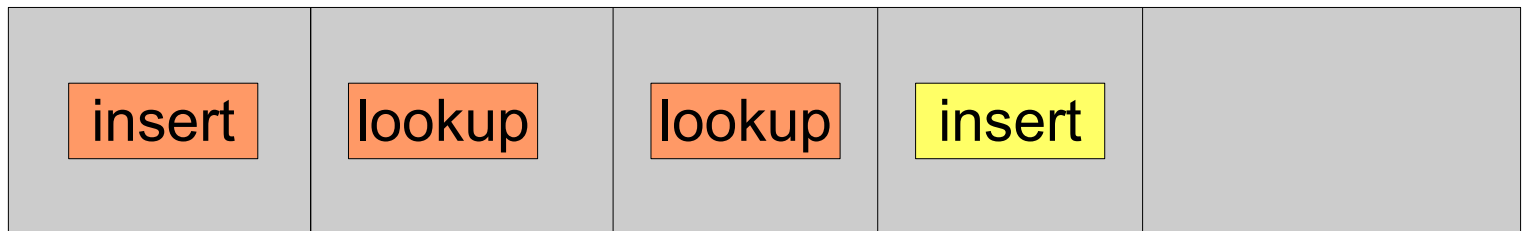
# Predictive LogSync Idea



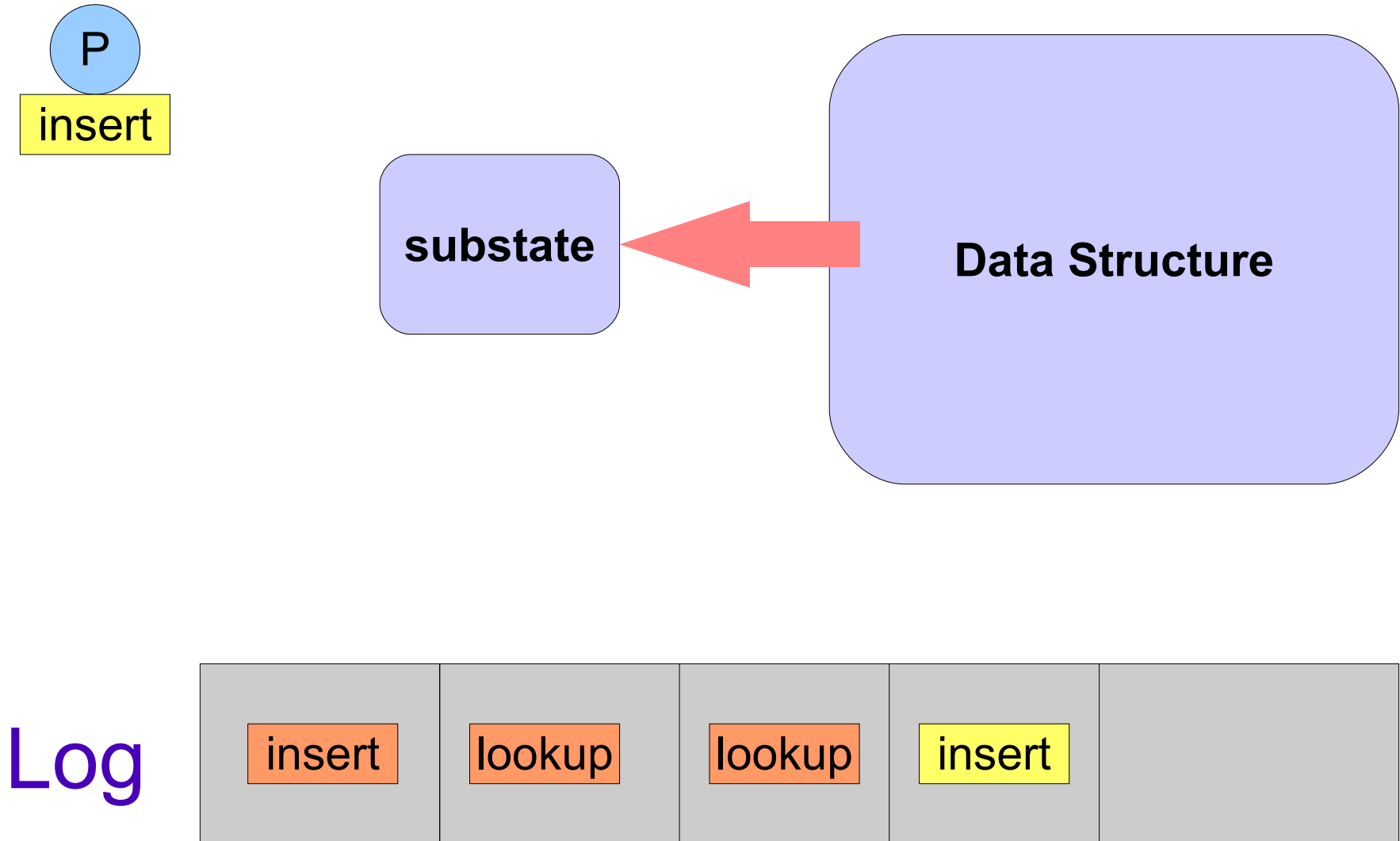
# Predictive LogSync Idea



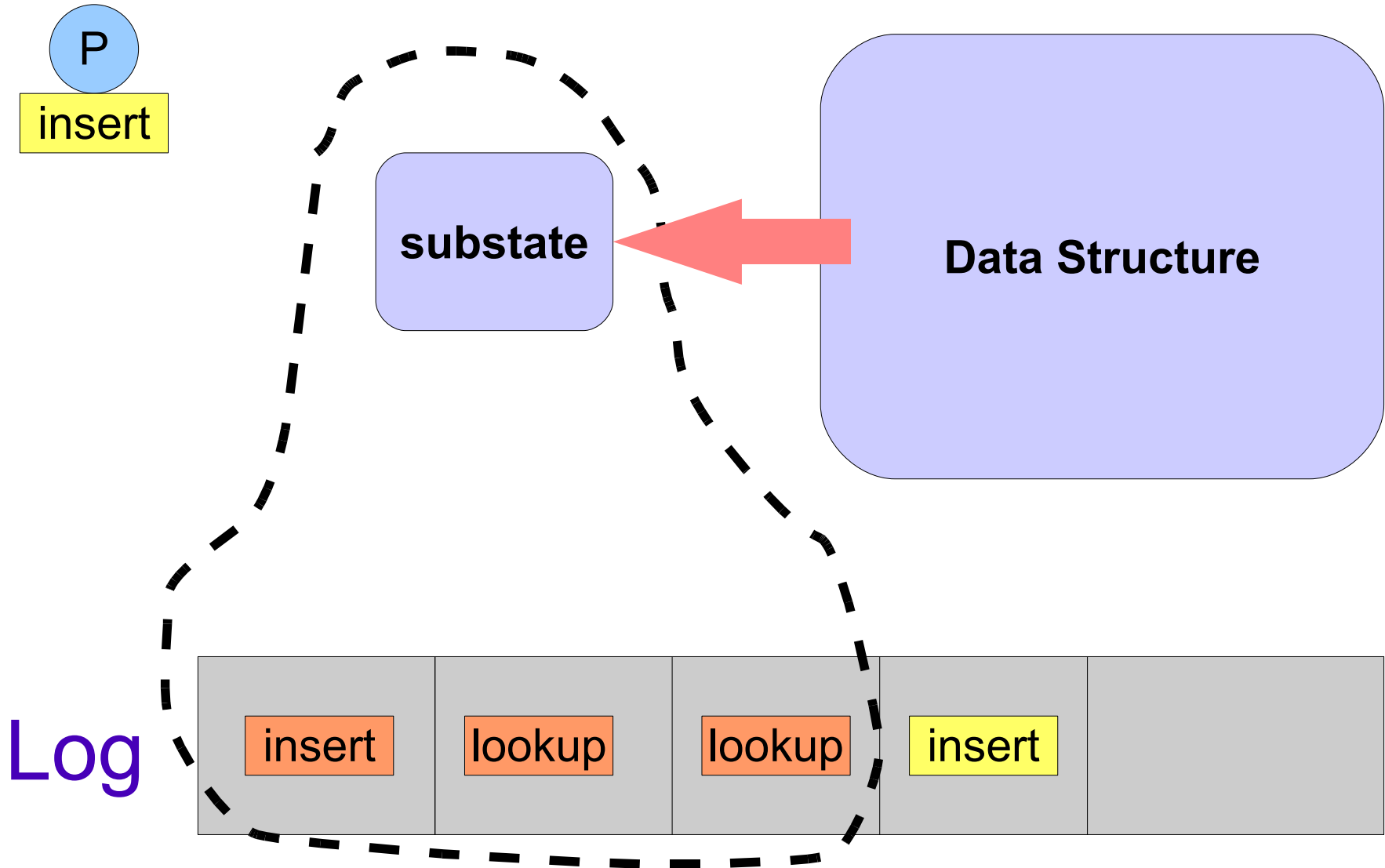
Log



# Predictive LogSync Idea



# Predictive LogSync Idea



# Predictive Log-Synchronization (PLS)

- Threads queue operations in a **log**

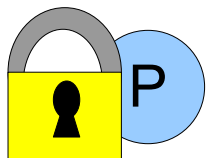
# Predictive Log-Synchronization (PLS)

- Threads queue operations in a **log**
- If **lock** acquired, then:  
**execute** operations in log

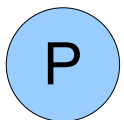
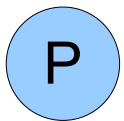
# Predictive Log-Synchronization (PLS)

- Threads queue operations in a **log**
- If **lock** acquired, then:  
    **execute** operations in log
- Otherwise:  
    threads can **predict** and proceed

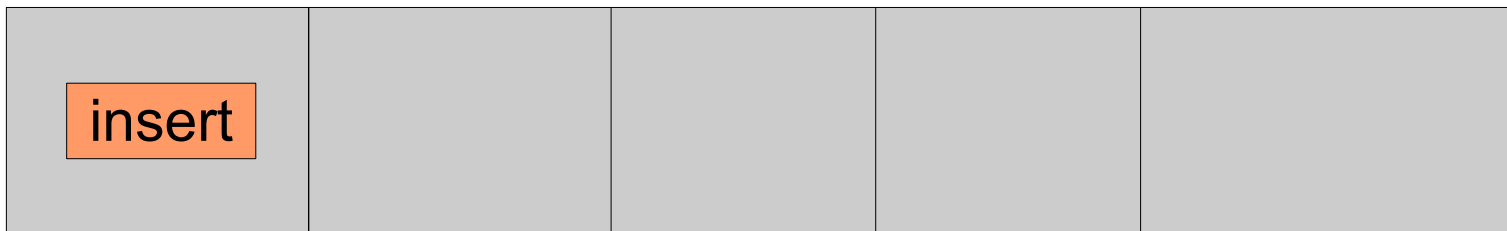
# PLS Core Idea



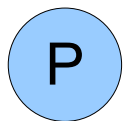
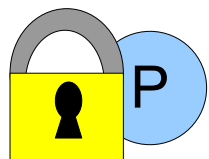
log



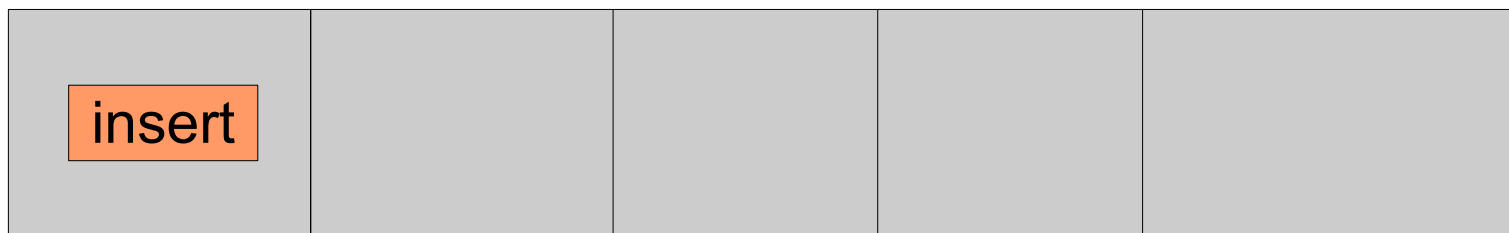
Log



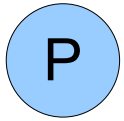
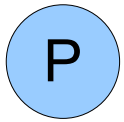
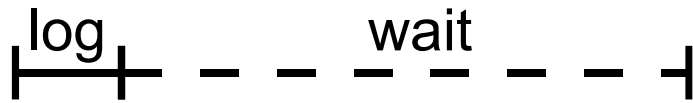
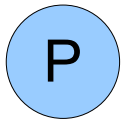
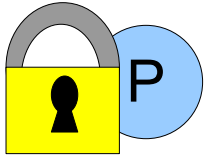
# PLS Core Idea



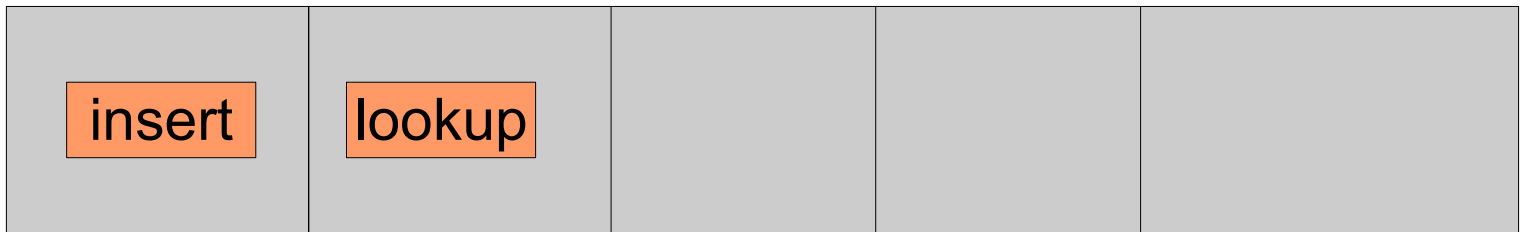
Log



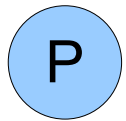
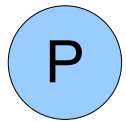
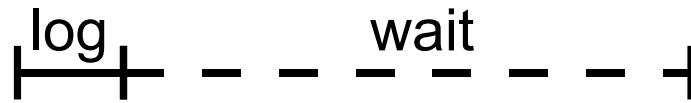
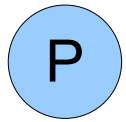
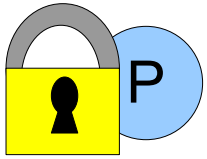
# PLS Core Idea



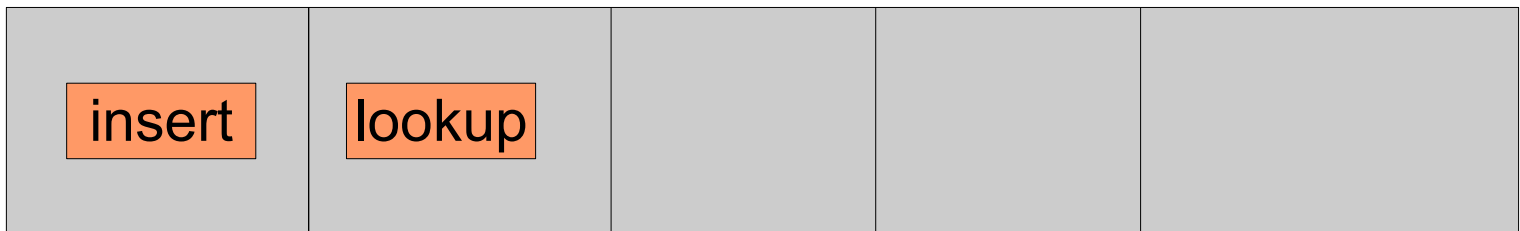
Log



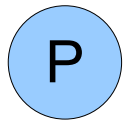
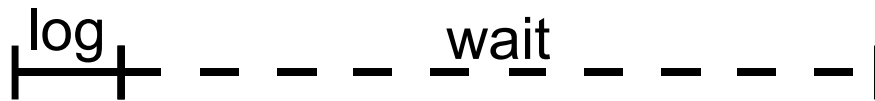
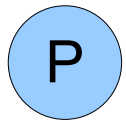
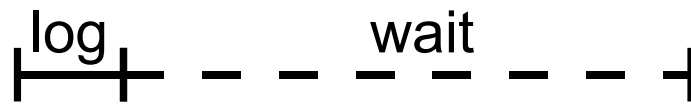
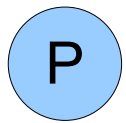
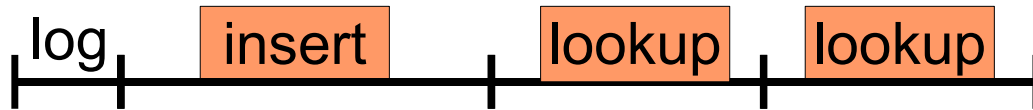
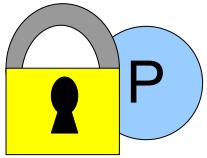
# PLS Core Idea



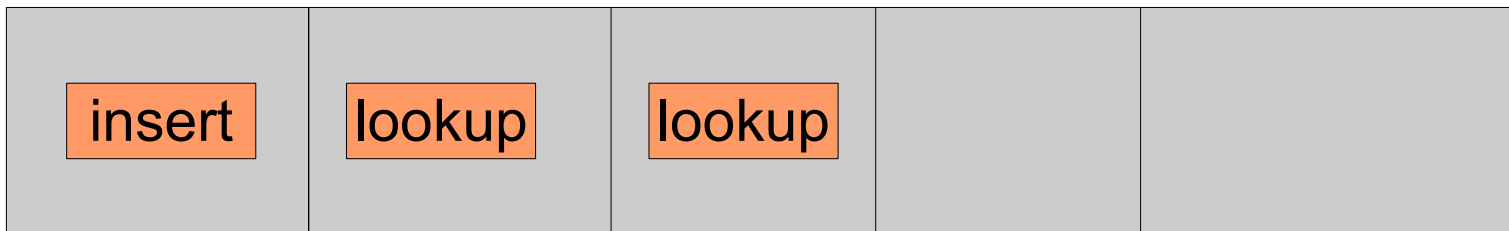
Log



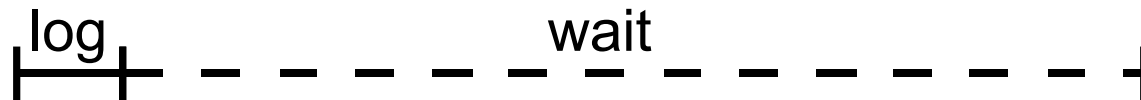
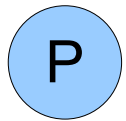
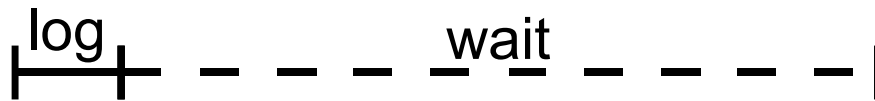
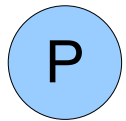
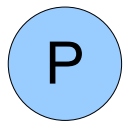
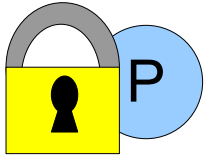
# PLS Core Idea



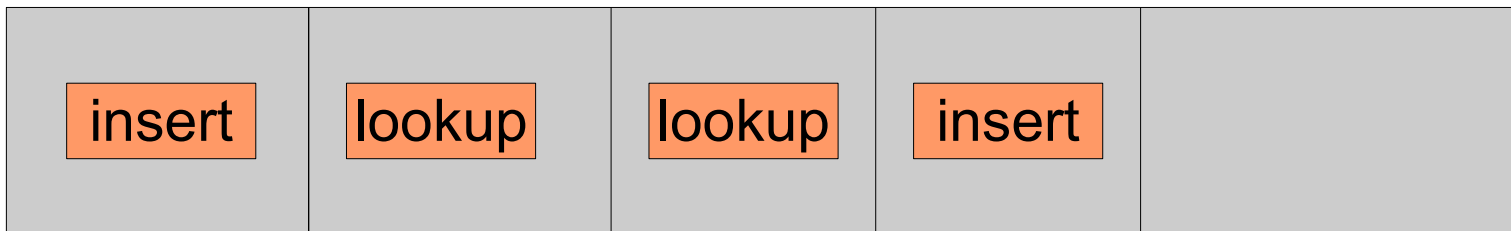
Log



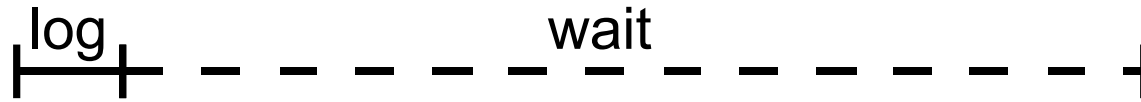
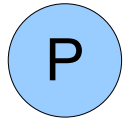
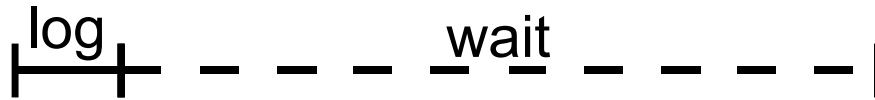
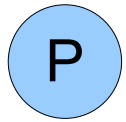
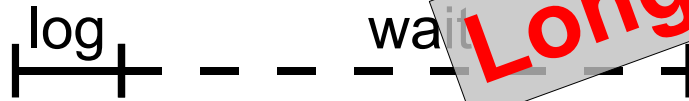
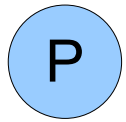
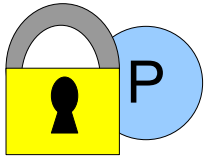
# PLS Core Idea



Log

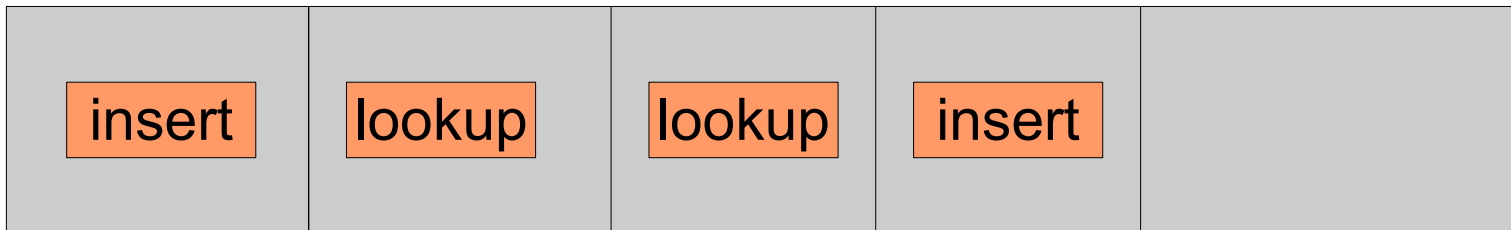


# PLS Core Idea

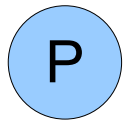
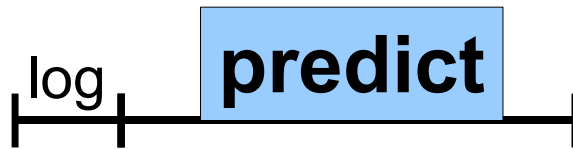
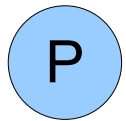
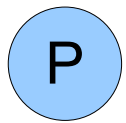
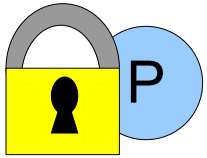


**Long critical path**

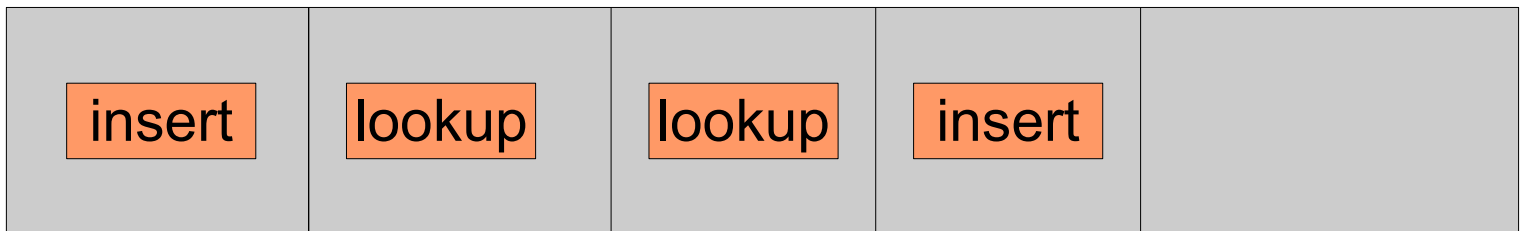
Log



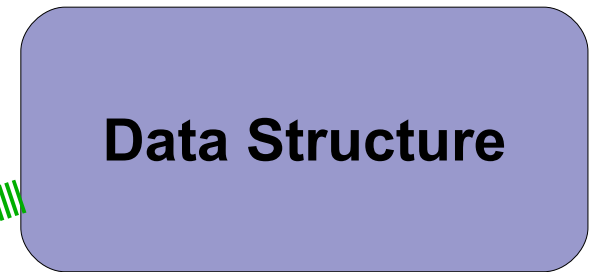
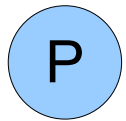
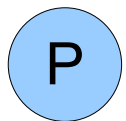
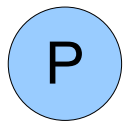
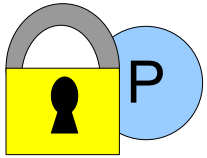
# PLS Idea: Prediction



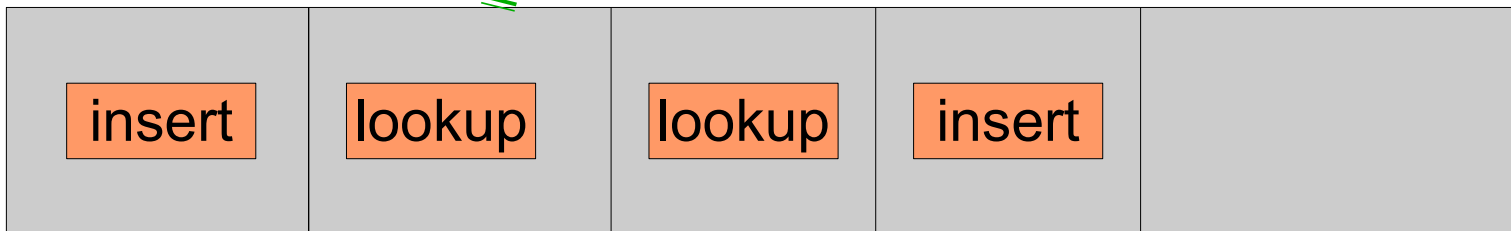
Log



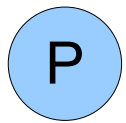
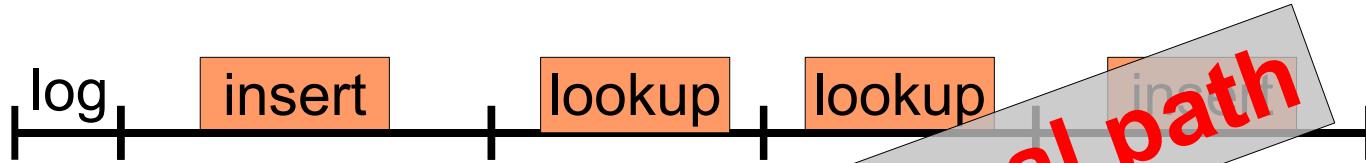
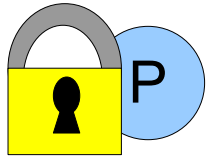
# PLS Idea: Prediction



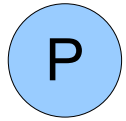
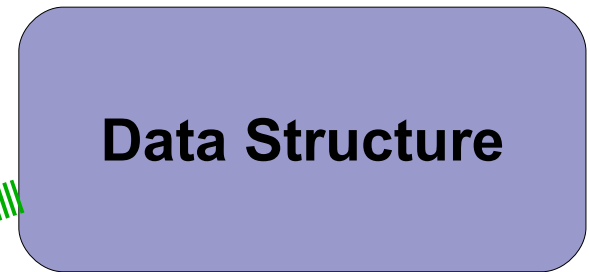
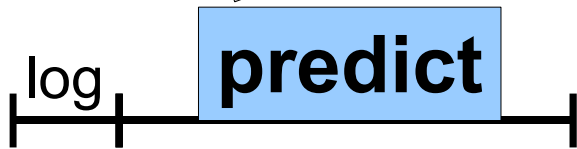
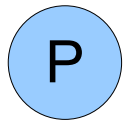
Log



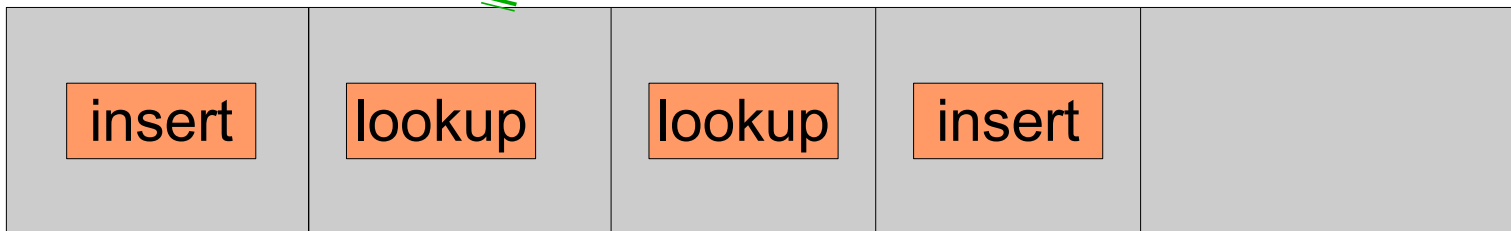
# PLS Idea: Prediction



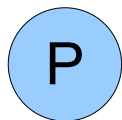
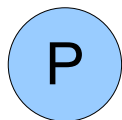
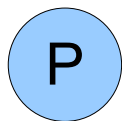
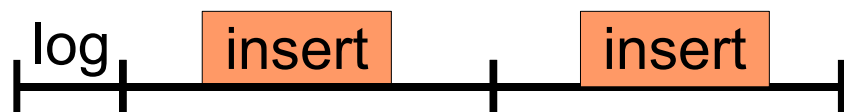
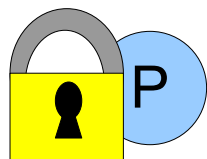
**Still long critical path**



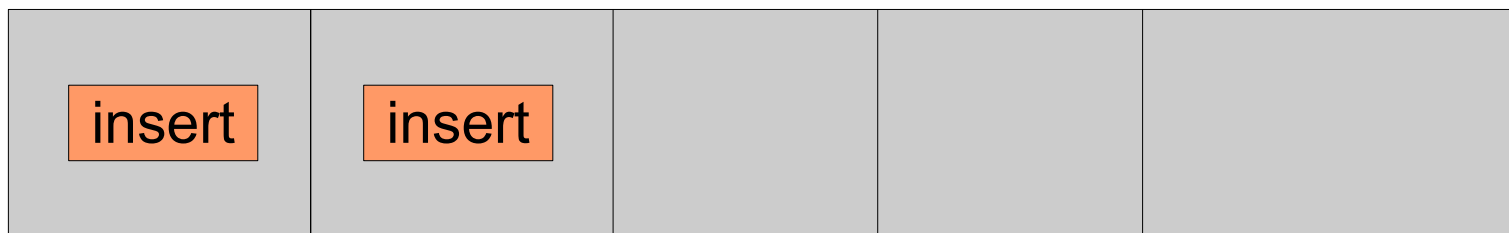
Log



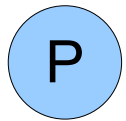
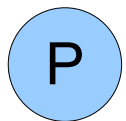
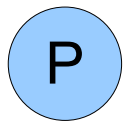
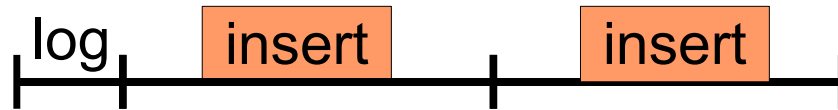
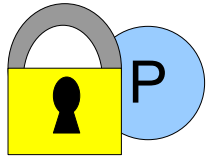
# PLS Idea: **Not logging readonly ops**



Log

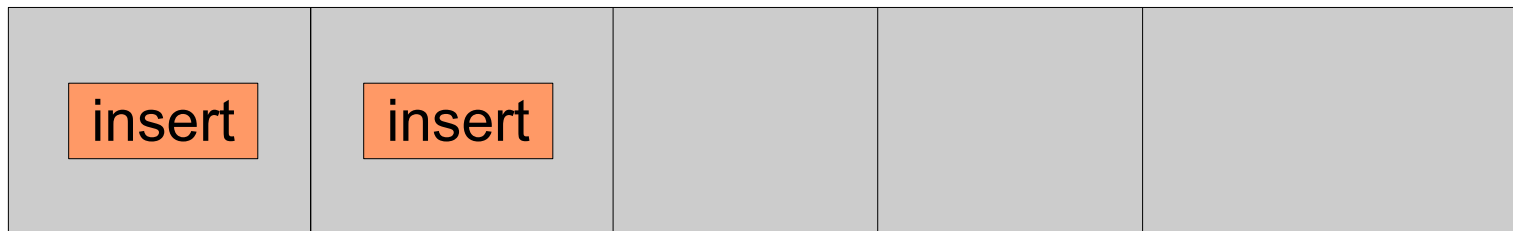


# PLS Idea: **Not logging readonly ops**



High % of  
Read-only ops

Log



# When does it work well?

- **Non-trivial** operations

# When does it work well?

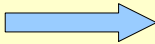
- **Non-trivial** operations
- **Many reads**

# When does it work well?

- **Non-trivial** operations
- Many **reads**
- Operation **semantics** support prediction
  - Sequential implementation *extended*

# The Programmer's Job

- Understands **sequential spec** only

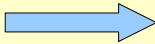
Substate + Log  Result

*Example* 

Red-black tree set

# The Programmer's Job

- Understands **sequential spec** only
  - Retrieve basic state (*substate*)

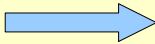
Substate + Log  Result

*Example* 

Red-black tree set

# The Programmer's Job

- Understands **sequential spec** only
  - Retrieve basic state (*substate*)
  - How each op. affects others

Substate + Log  Result

*Example* 

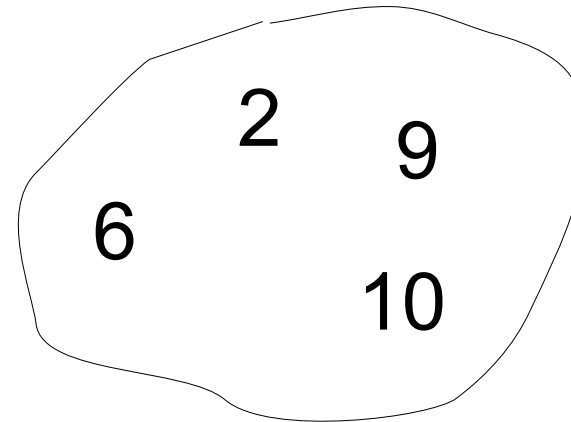
Red-black tree set

# Red-black tree set

- Supports **insert**, **delete**, and **lookup**

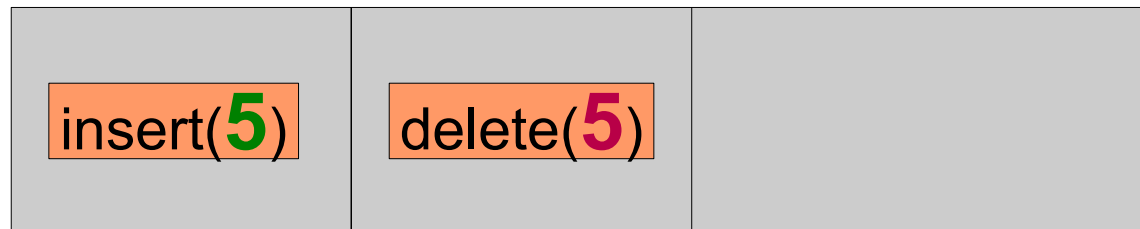
# Red-black tree set

- Supports **insert**, **delete**, and **lookup**



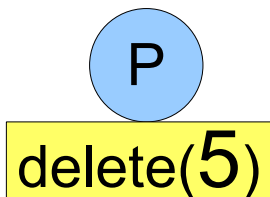
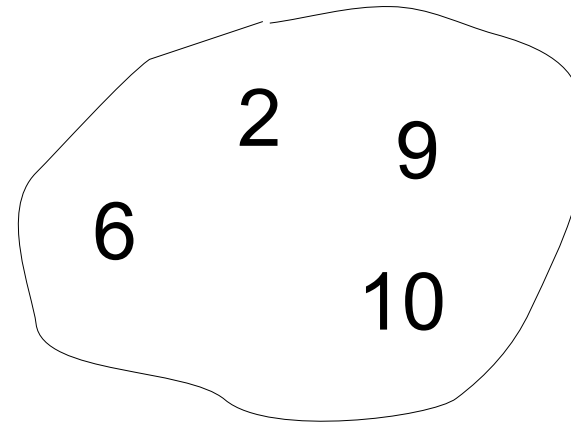
P  
delete(5)

Log

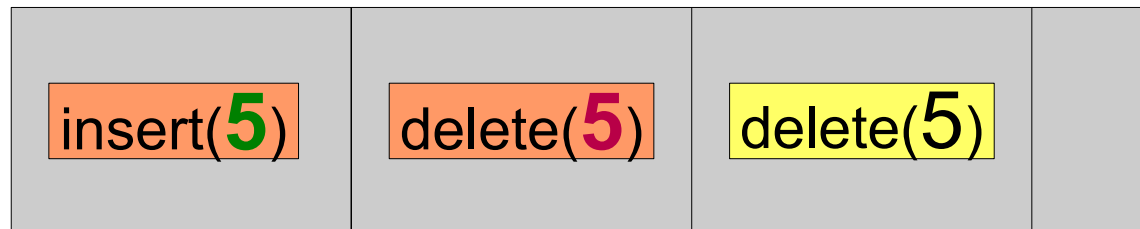


# Red-black tree set

- Supports **insert**, **delete**, and **lookup**

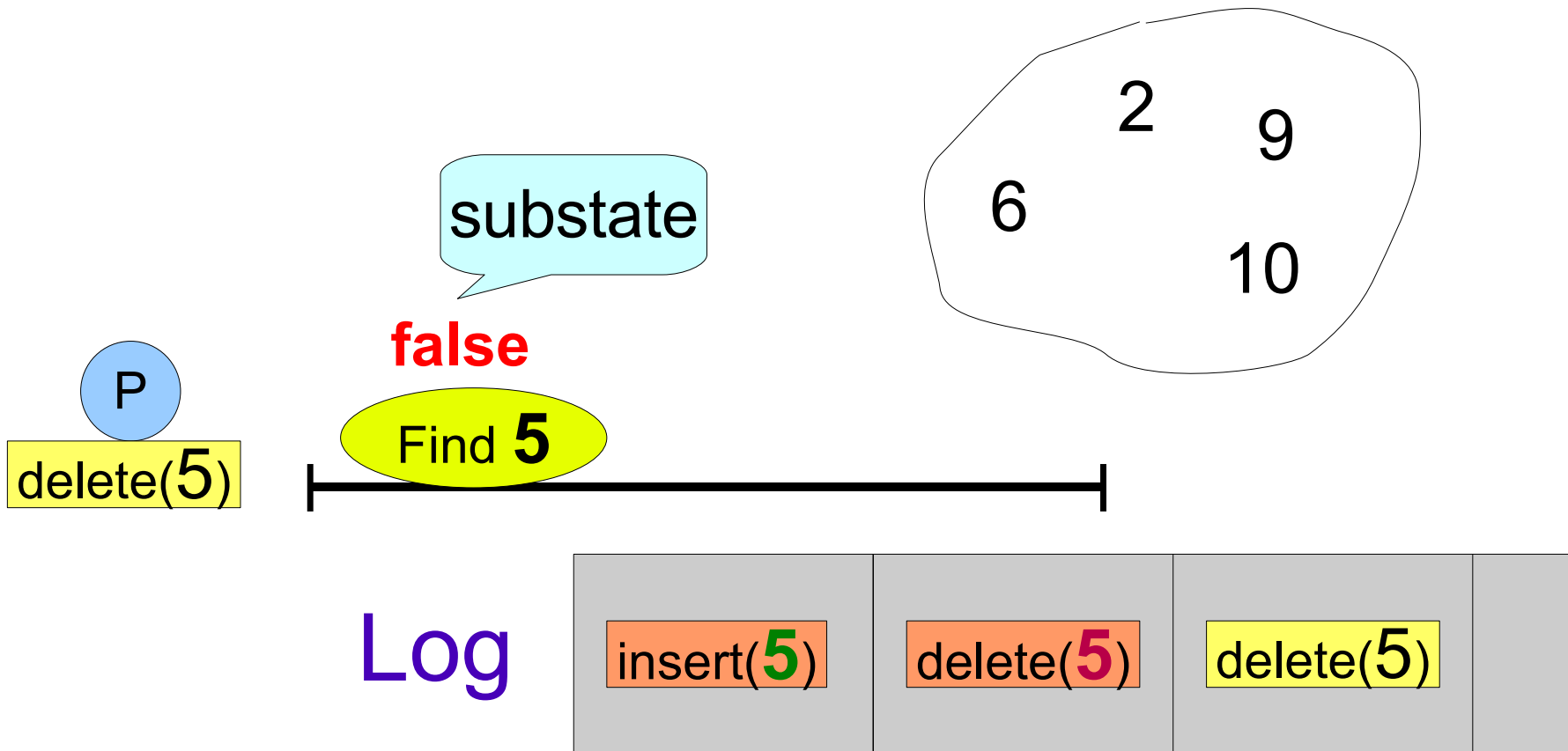


Log



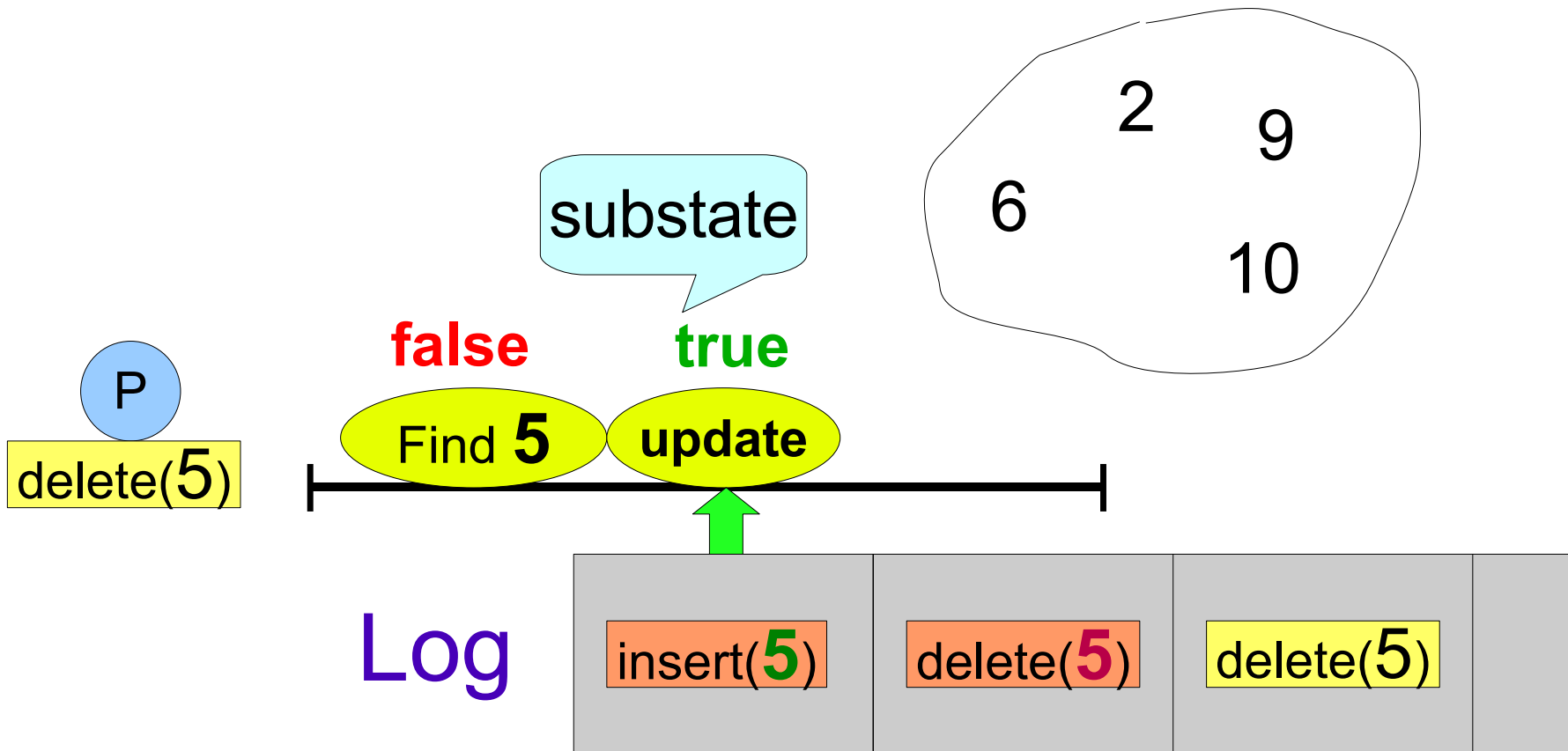
# Red-black tree set

- Supports **insert**, **delete**, and **lookup**



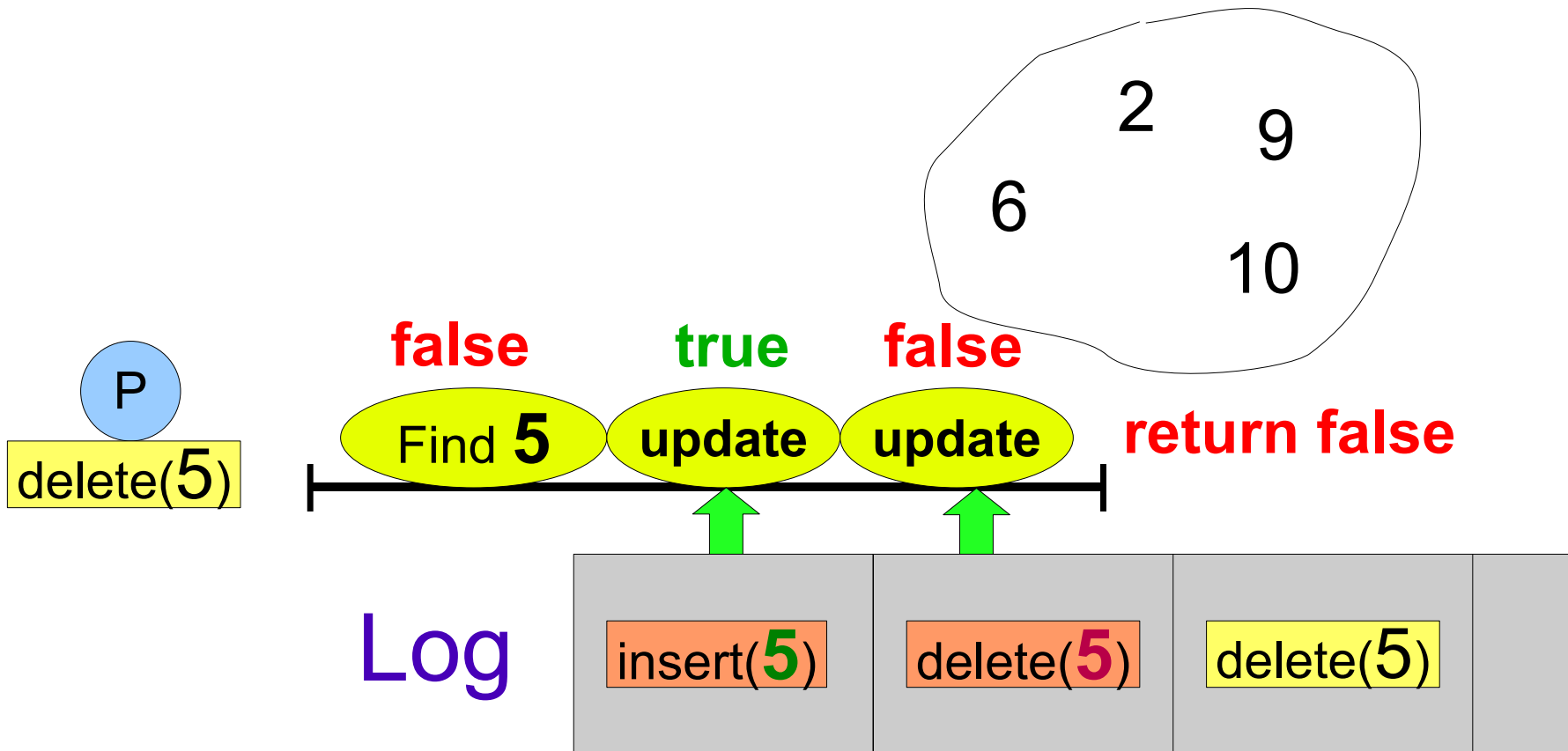
# Red-black tree set

- Supports **insert**, **delete**, and **lookup**



# Red-black tree set

- Supports **insert**, **delete**, and **lookup**



# Concurrent reads and writes

- Data structure **duplicated** and **double-buffered**
  - Writable copy: for the **lock-owner**
  - Read-only copy: for **prediction**

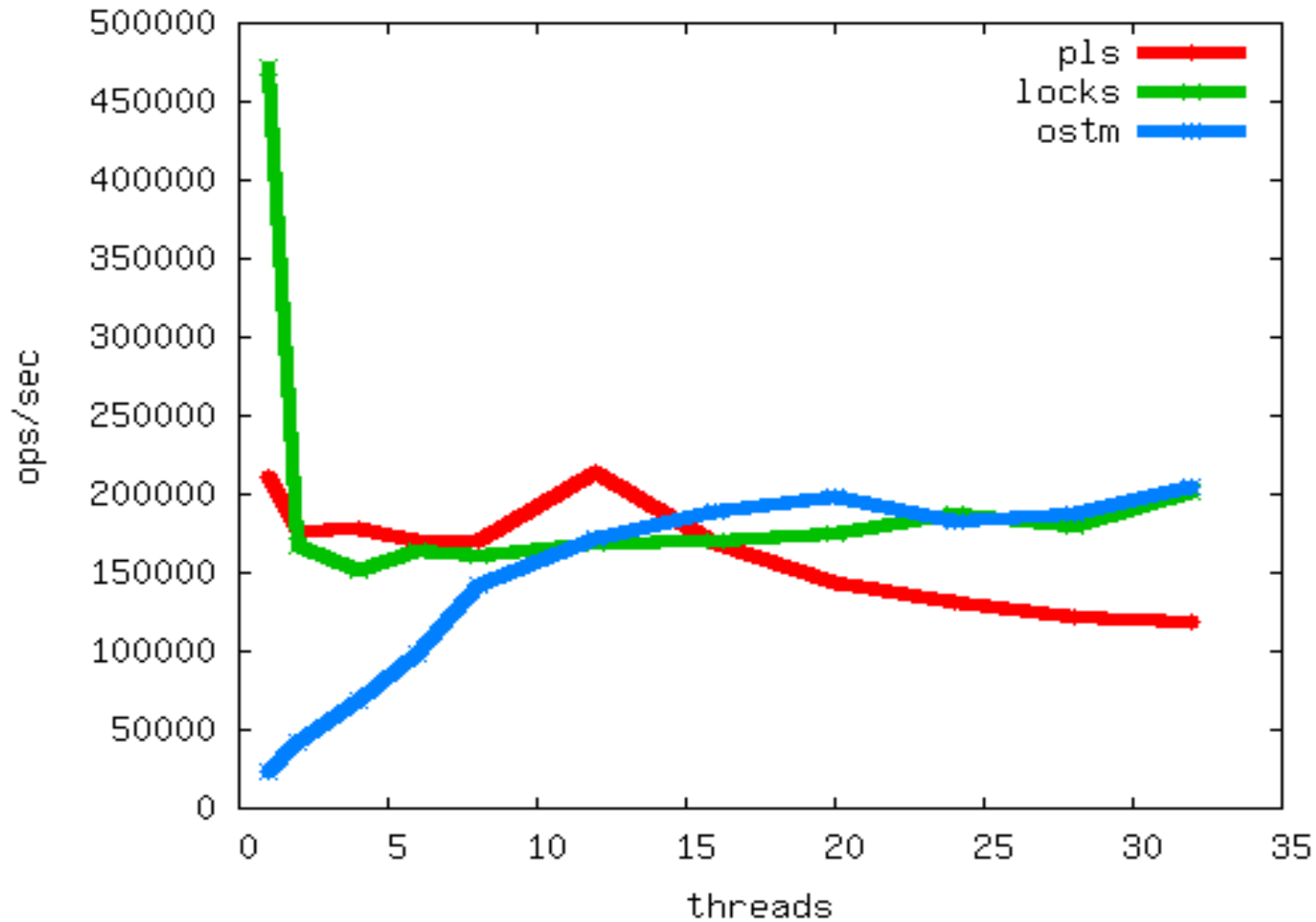
# Concurrent reads and writes

- Data structure **duplicated** and **double-buffered**
  - Writable copy: for the **lock-owner**
  - Read-only copy: for **prediction**
- Modifications are repeated
  - Programmer can optimize 2<sup>nd</sup> time modifications

# Performance

- Java red-black tree
- 50% / 50% inserts and deletes
- 8% / 2% / 90% inserts, deletes, lookups
- Large tree: key range = 1M
- Small tree: key range = 200

# Large tree, 50% ins. 50% del.



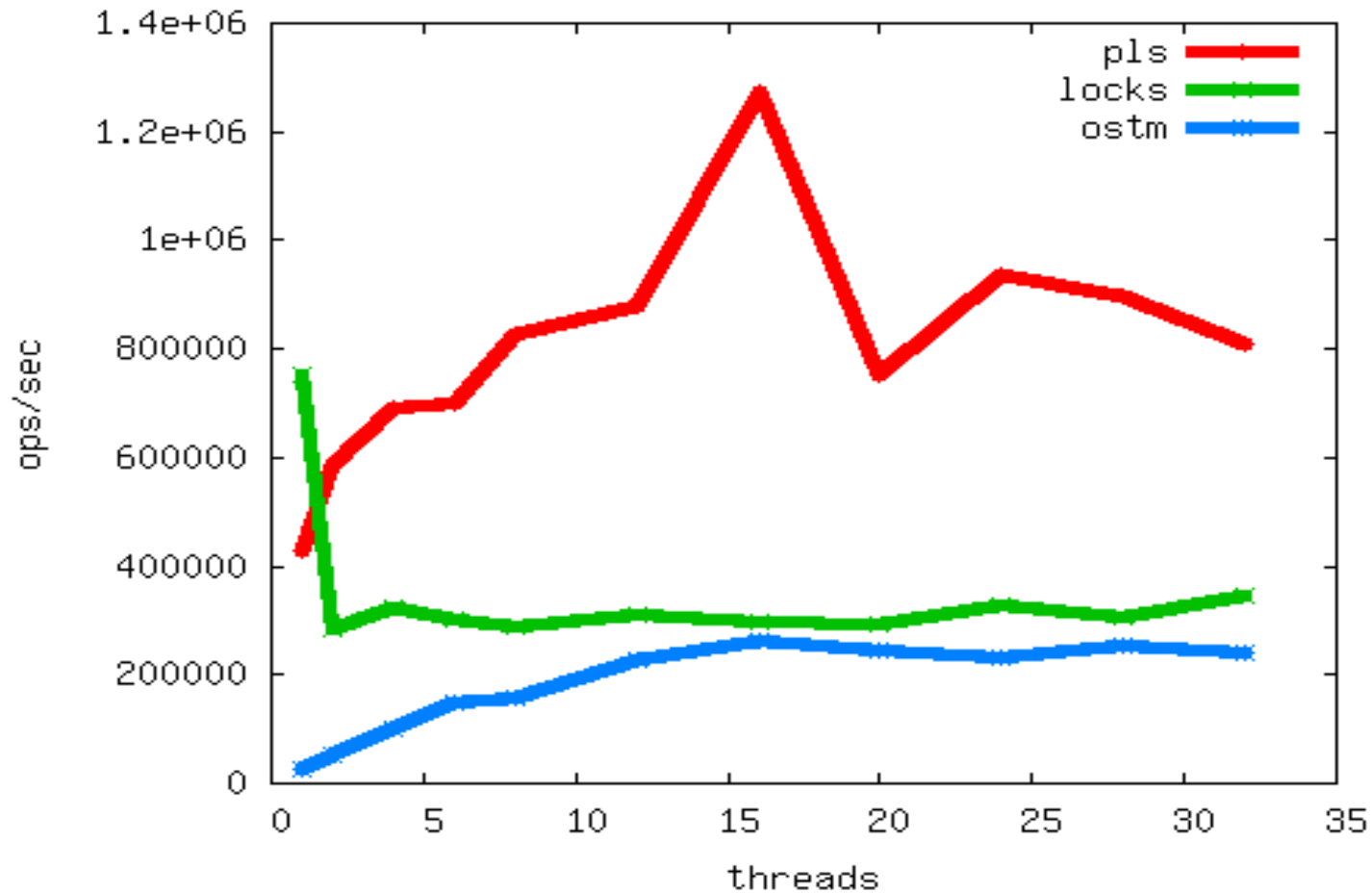
PLS

locks

OSTM\*

\* rewritten in  
Java

# Large tree, 8% ins. 2% del. 90% lookups



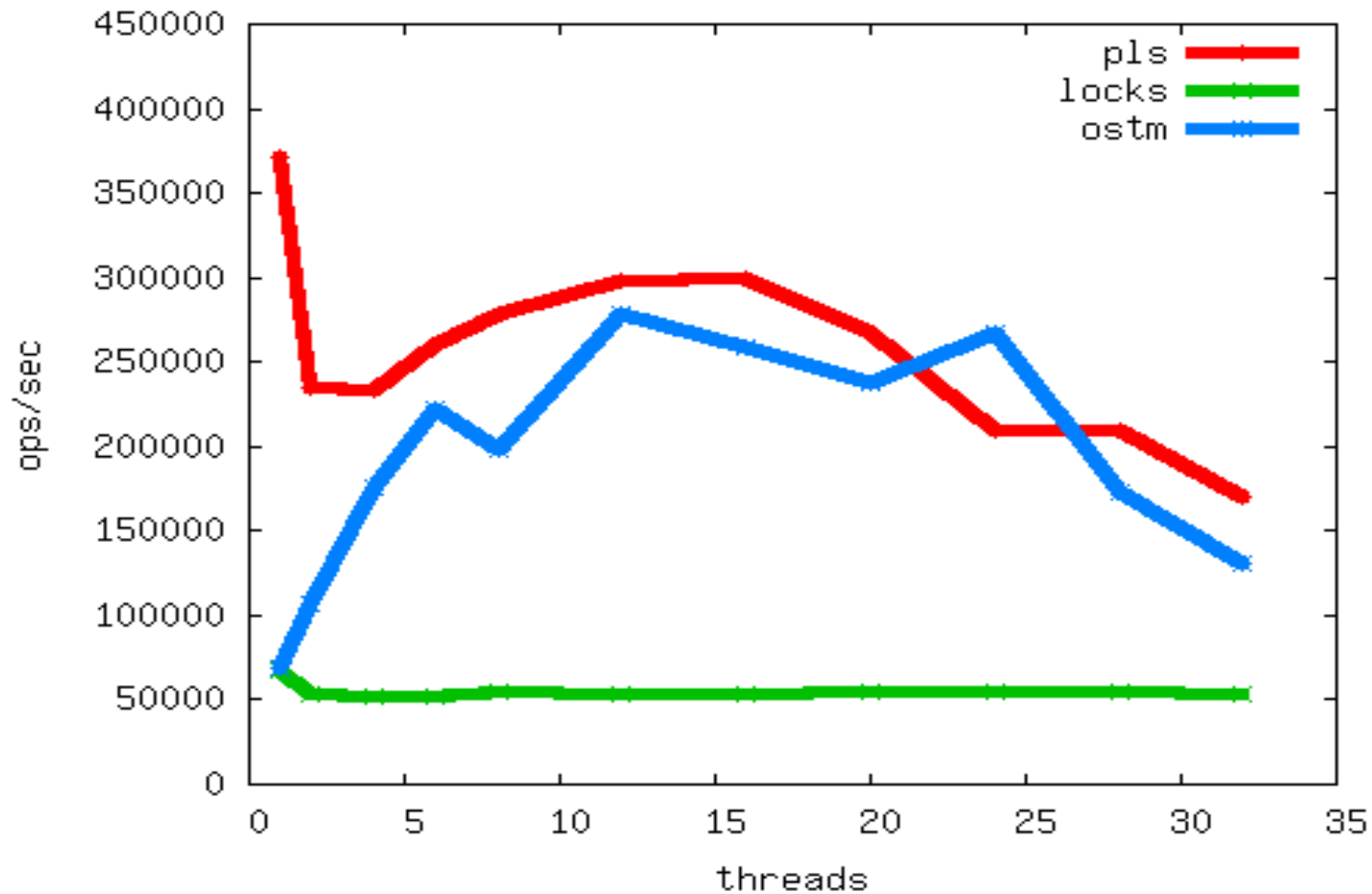
PLS

locks

OSTM\*

\* rewritten in Java

# Small tree, 50% ins. 50% del.



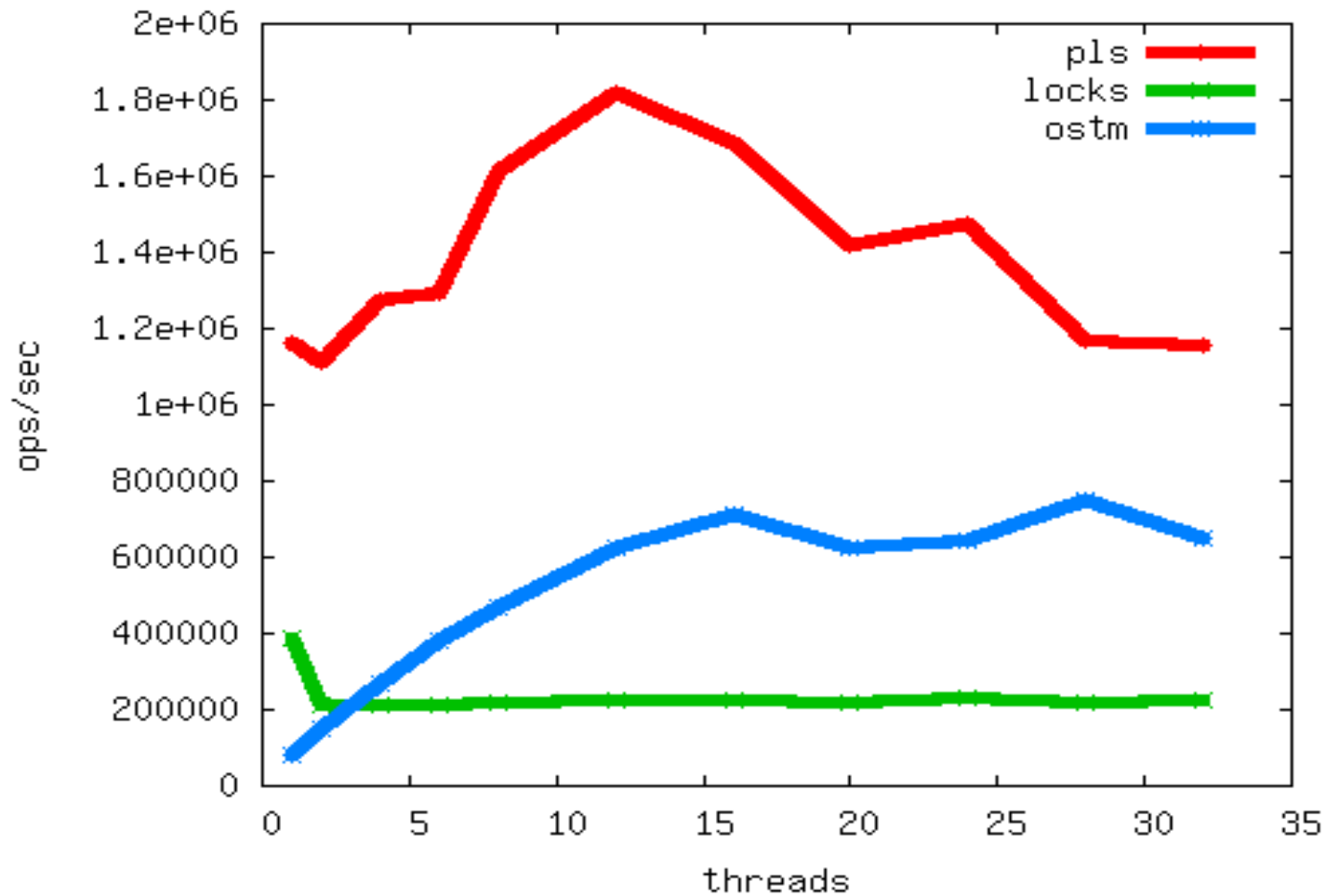
PLS

locks

OSTM\*

\* rewritten in  
Java

# Small tree, 8% ins. 2% del. 90% lookups



PLS

locks

OSTM\*

\* rewritten in Java

# Predictive Log Synchronization

- Bad:
  - Not always applicable
  - Limited scalability with high % of **modifying** ops
- Good:
  - High throughput if high % of **read-only** ops
  - Does not require any "parallel programming" skills

