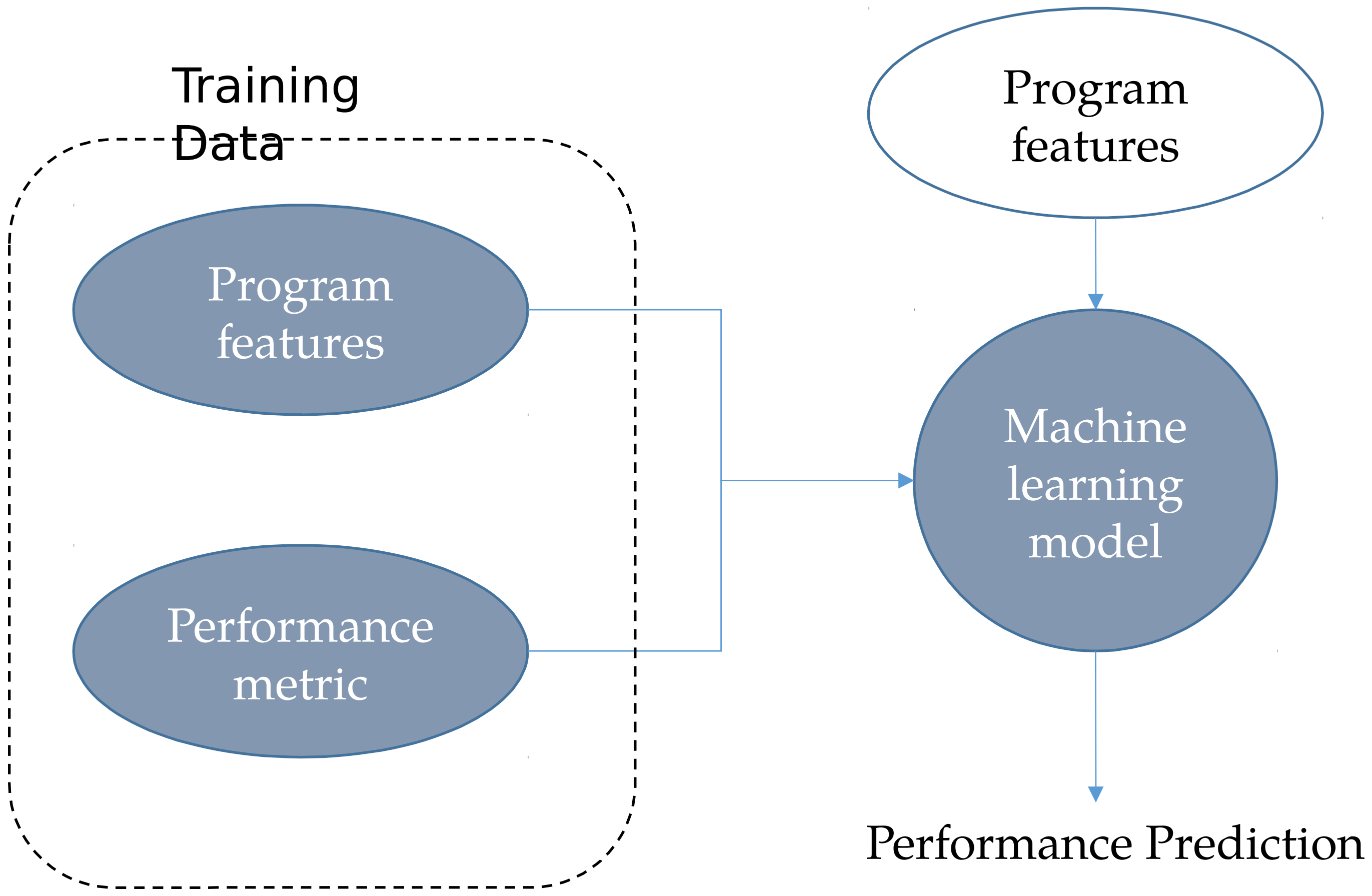
Performance Bug Diagnosis using Performance Predictions

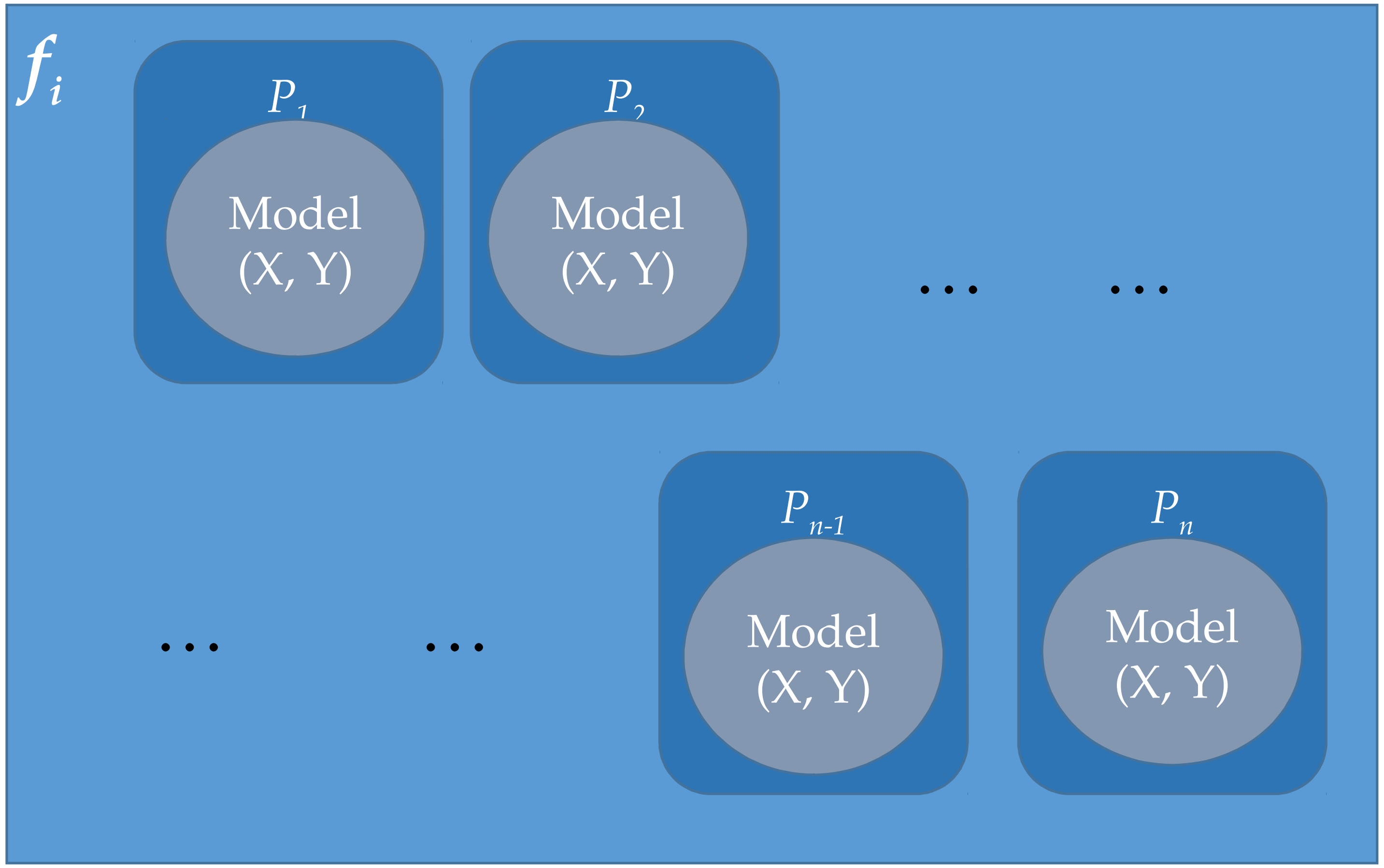
Program Performance Prediction



Program Performance Prediction

- Training data for different performance bugs
  - Simulating a buggy execution
  - Collect program features and corresponding performance metric
- Construct performance prediction model for specific performance bug scenario
  - Machine learning / Statistical models

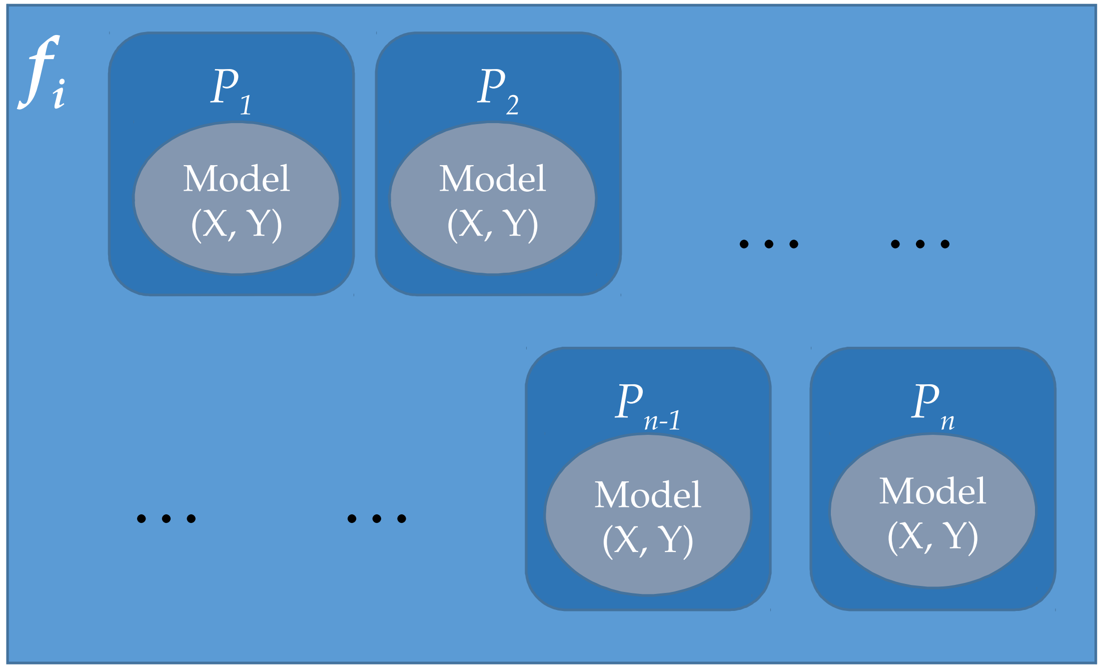
Modeling Performance Bug



- F ={ f_1, f_2 … }
  - Set of functions
- X = < x_1 , x_2 , …>
  - Set of features
- Y
  - Performance metric
- P = { p_1 , p_2 , …}
  - Performance bugs

Performance Bug Diagnois

- Predict performance for current features,
  - X = < x_1 , x_2 , …>
- Performance predictions for all bug models



Prediction from bug models of P_i == Actual observed performance ⇒ Bug P_i

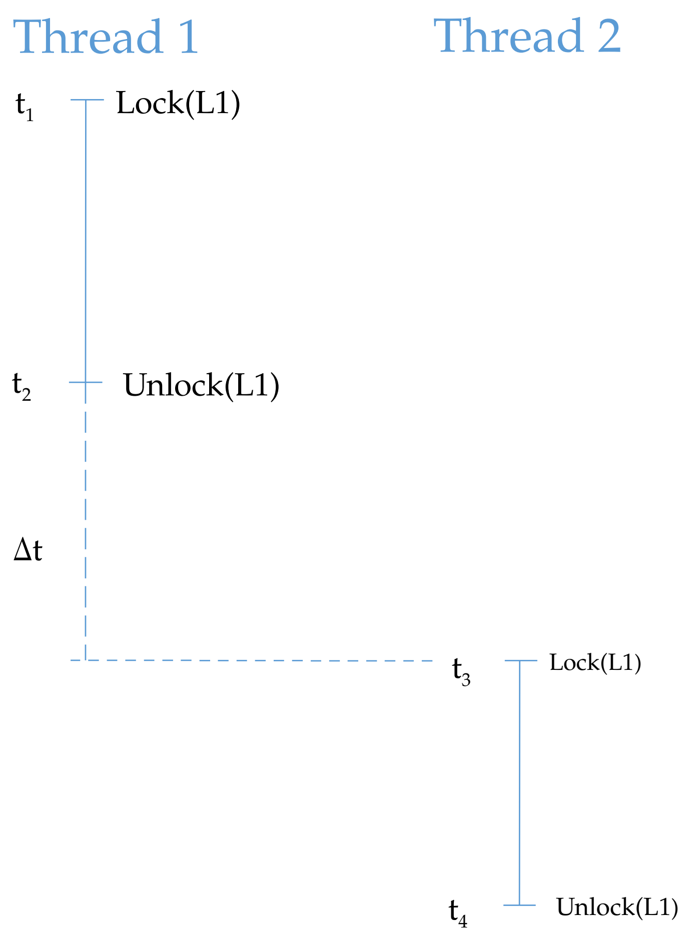
Modeling Performance Bug

- Set of functions, F ={ $f_{1,} f_2 \ldots$ }
  - Hot functions
- Set of features, X = < x_1 , x_2 , …>
  - Code features : CPI, load/store and branch count etc.
  - Data features : Loop count, branch miss rate, cache miss rate etc.
- Performance metric, y
  - Execution time
- Performance bugs P = { p_1 , p_2 , …}
  - High cache miss rate or branch misprediction rate, high cache or lock contentions etc.

Challenges

- What are the fixed set of features?
- How to generate training data for specific performance bug?
- How to create a performance prediction model accurate enough for performance anomaly detection?
- What is an effective granularity of program to predict performance?

Modeling Lock Contention Problem

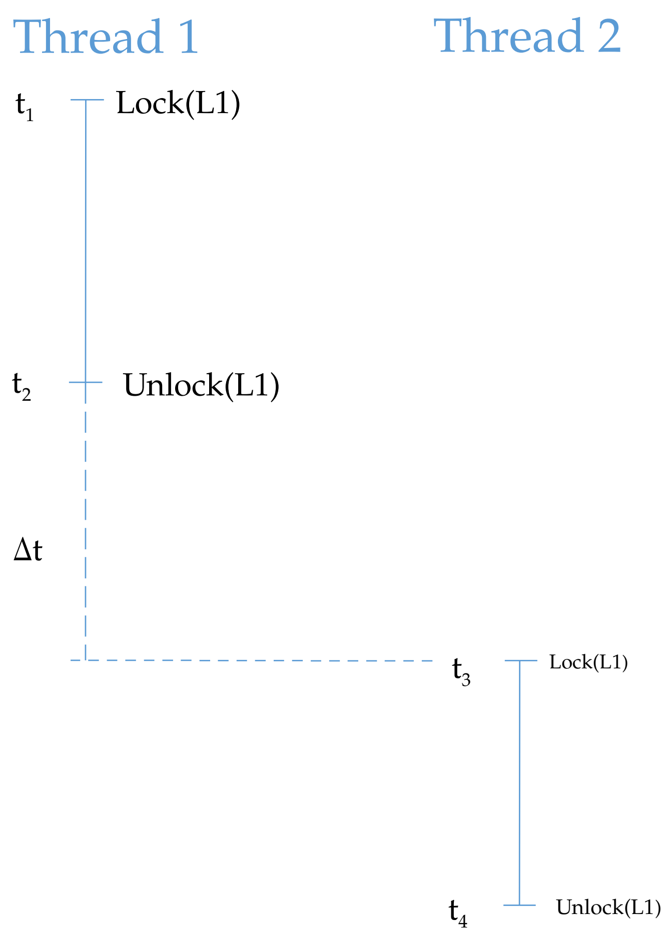


- Execution time of critical section (CS):

  $T_{cs} = t_2 - t_1$

- $\Delta t = 0$ ✉ lock contention
- $T_{cs} > t_3 - t_1$ ✉ lock contention

Modeling Lock Contention Problem



- Profile T_{cs} for a contention free execution
  - Positive training example
- Estimate T_{cs} for contention scenerio
  - Negative training example
  - $T_{cs} = t_3 - t_1$

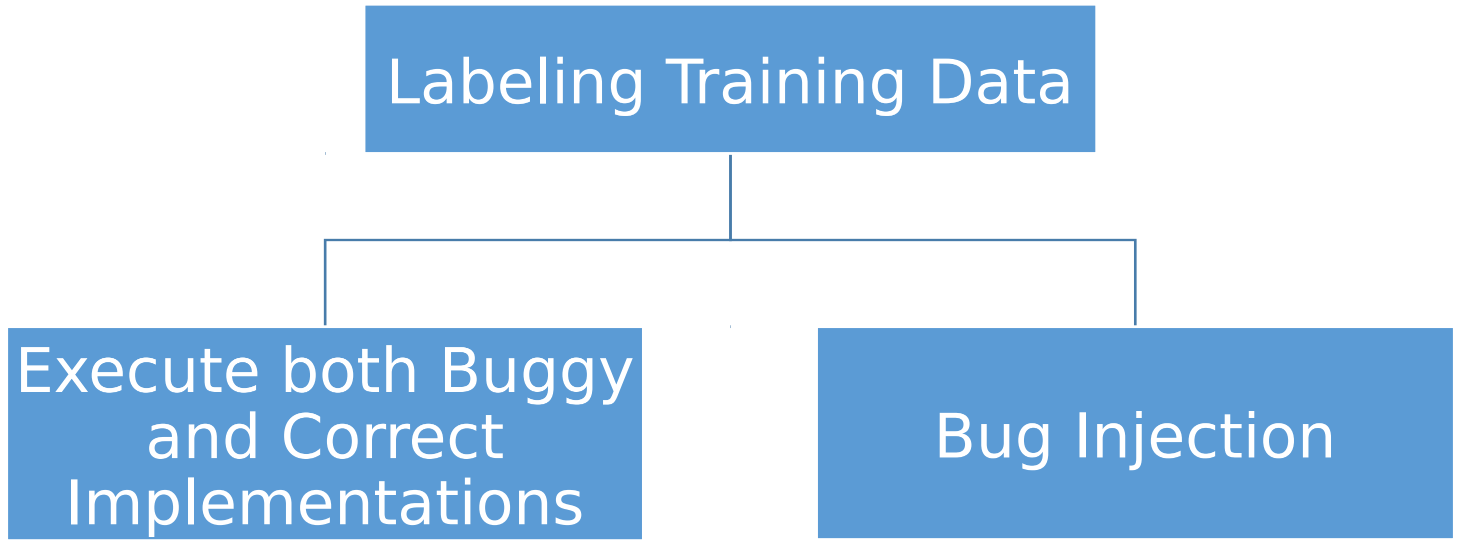
Performance Anomaly Detection

- Collect features for a specific function(or block of code)
- Features for Cache Contention Problem:
  - Cache invalidation rate
  - Number of threads
  - IPC

Machine Learning Model

- Training Data :
    - Features collected from dynamic instances of the function(or block of code)
    - Execute the program with
        - Different input sizes
        - Different thread numbers
        - Different machine configurations

Machine Learning Model



- When both implementations are available

- Injector thread
- Access resource to introduce contention in original program

Use a threshold difference in IPC to label data as an instance of anomalous execution

Experiments

- PARSEC & Phoenix benchmarks
    - Different input sizes (small, large)
    - Different thread numbers (8, 16 & 32 threads)
    - Different machine configurations (8, 16 & 32 core)
- Machine learning models: SVM, MLP & Decision tree

Experiments

- Labeling is done by running both correct & buggy implementations
- Anomaly due to cache contention detected(known issues):
  - Streamcluster – false sharing in function *`double pgain(...)`*
  - Dateset size: *26850*
  - Cross validation scores
    - SVM : ~99%
    - MLP : ~99 %
    - Decision Tree: ~99%