Program Profiling

Xiangyu Zhang

Outline

- What is profiling.
- Why profiling.
- Gprof.
- Efficient path profiling.
- Object equality profiling

What is Profiling

- Tracing is lossless, recording every detail of a program execution
  - Thus, it is expensive.
  - Potentially infinite.
- Profiling is lossy, meaning that it aggregates execution information to finite entries.
  - Control flow profiling
    - Instruction/Edge/Function: Frequency;
  - Value profiling
    - Value: Frequency

Why Profiling

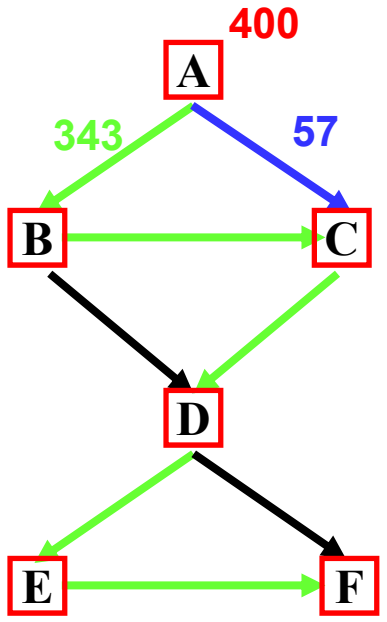
- Debugging
  - Enable time travel to understand what has happened.
- Code optimizations
  - Identify hot program paths;
  - Data compression;
  - Value speculation;
  - Data locality that help cache design;
  - Performance tuning
- Security
  - Malware analysis
- Testing
  - Coverage.

GNU gprof Profiler

- Gprof is a profiler for C programs.
- It profiles execution times for each individual functions and produces a call graph with call edges annotated with frequencies.
- Working
  - Gcc with the –p –pg options. –p tells the program to save profiling information, and –pg saves debug information in the compiled executable.
  - Gcc instruments the entry and exit of each function to record the calling frequency of each function.
  - Sampling is used to measure execution time.
    - inaccuracy
  - A gmon.out file will be created at the end.
  - Run *gprof ./a.out* to view the profiler's information.

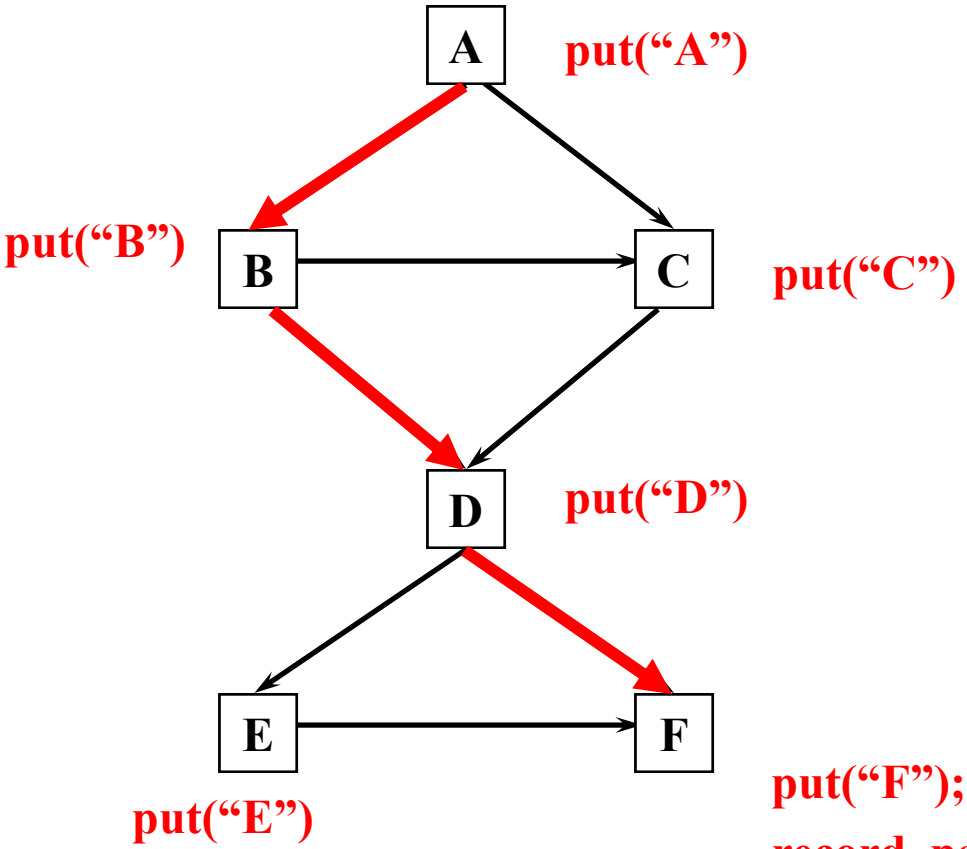
More Advanced: Path Profiling

- How often does a control-flow path execute?
- Levels of profiling:
  - blocks
  - edges
  - paths

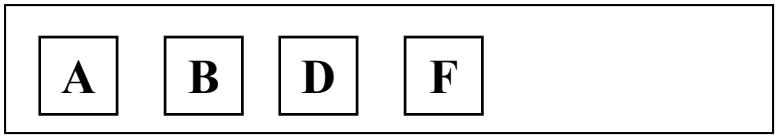


Edge profile equivalent to block profile?

Naive Path Profiling

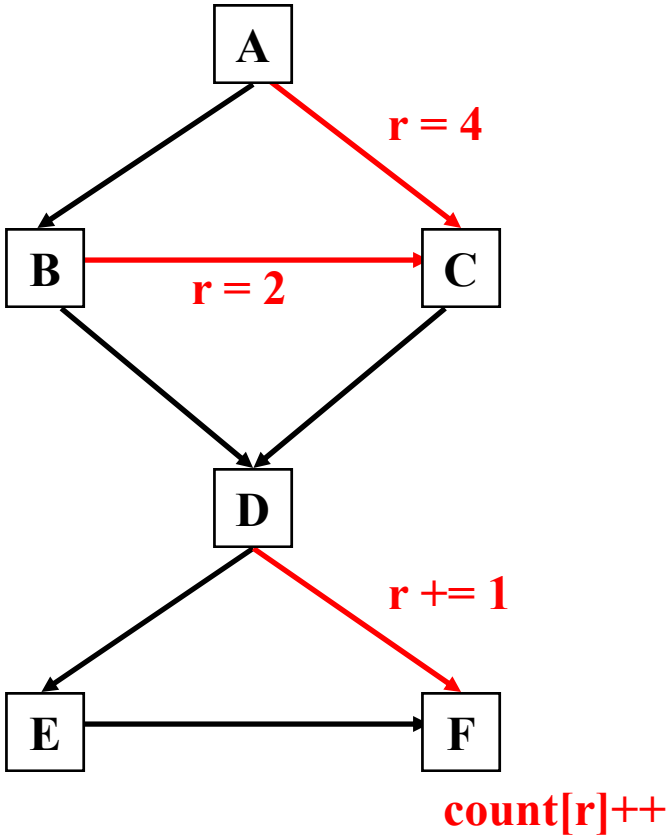


buffer



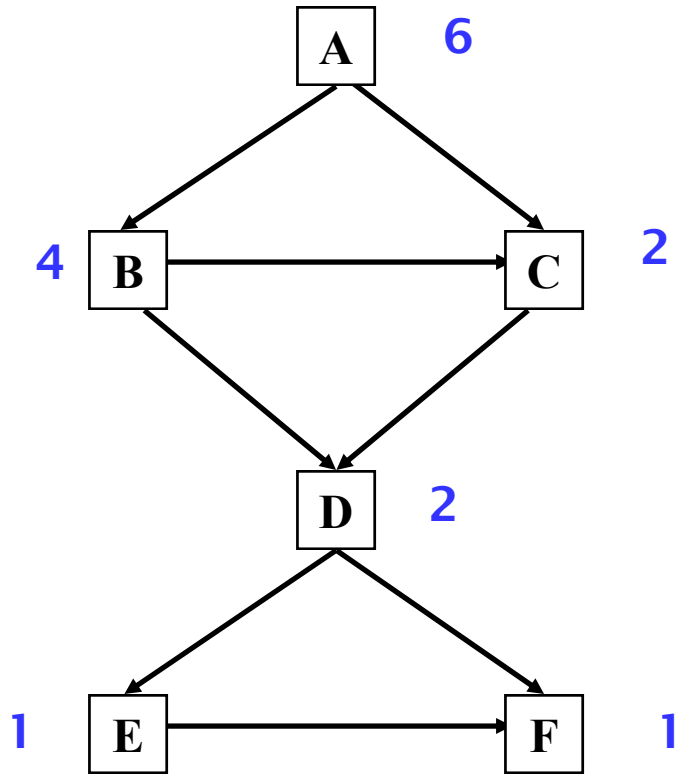
Find the smallest set of places to instrument

Efficient Path Profiling



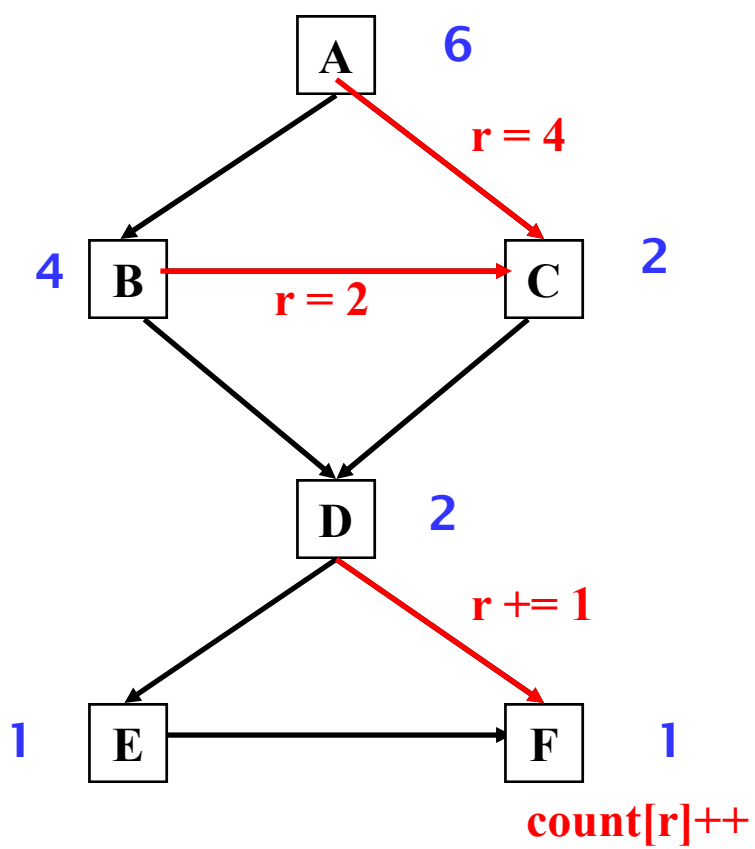
Path	Encoding
ABDEF	0
ABDF	1
ABCDEF	2
ABCDF	3
ACDEF	4
ACDF	5

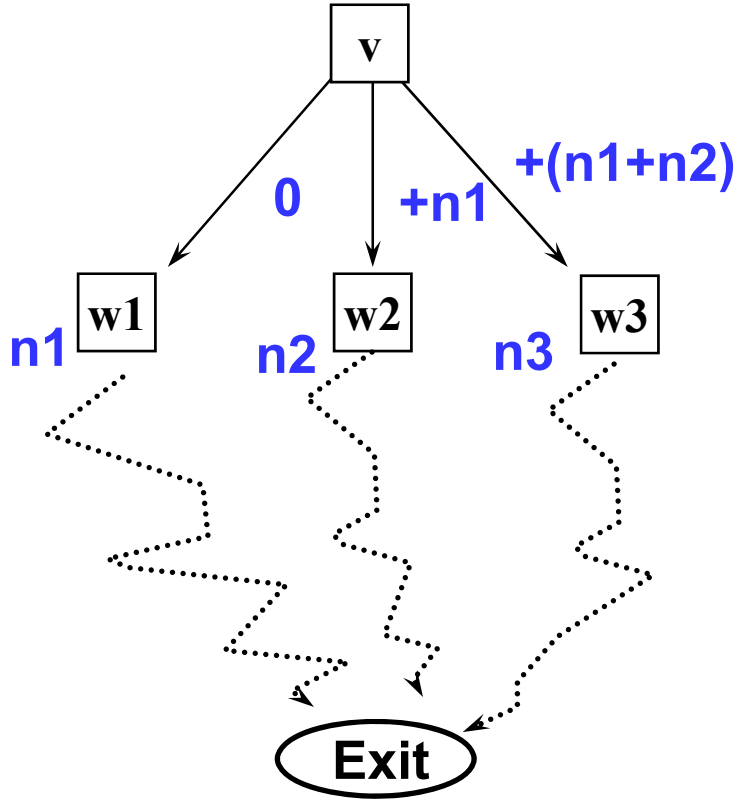
Efficient Path Profiling



- Each node is annotated with the number of paths from that node to the end.
- Num(n)= $\Sigma_{\text{(child i)}}$ Num(i)

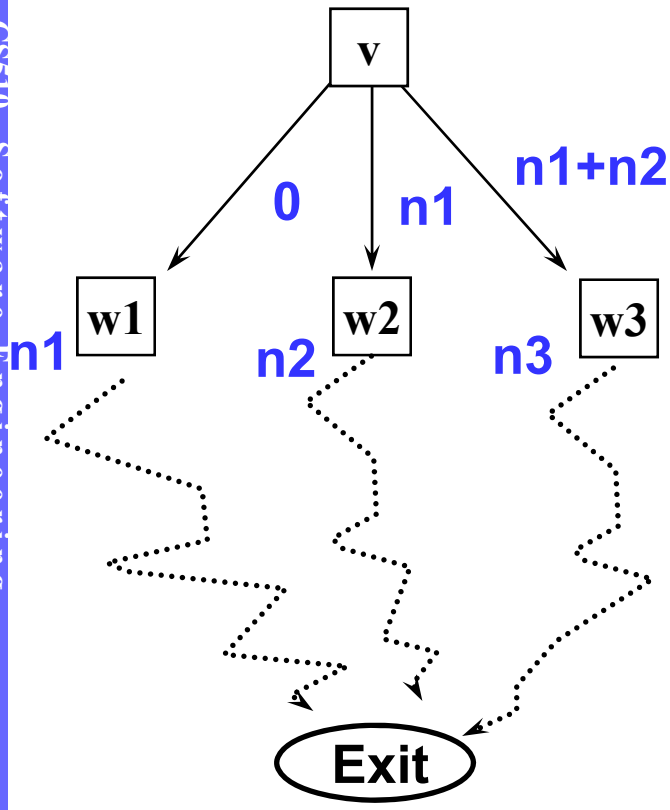
Efficient Path Profiling

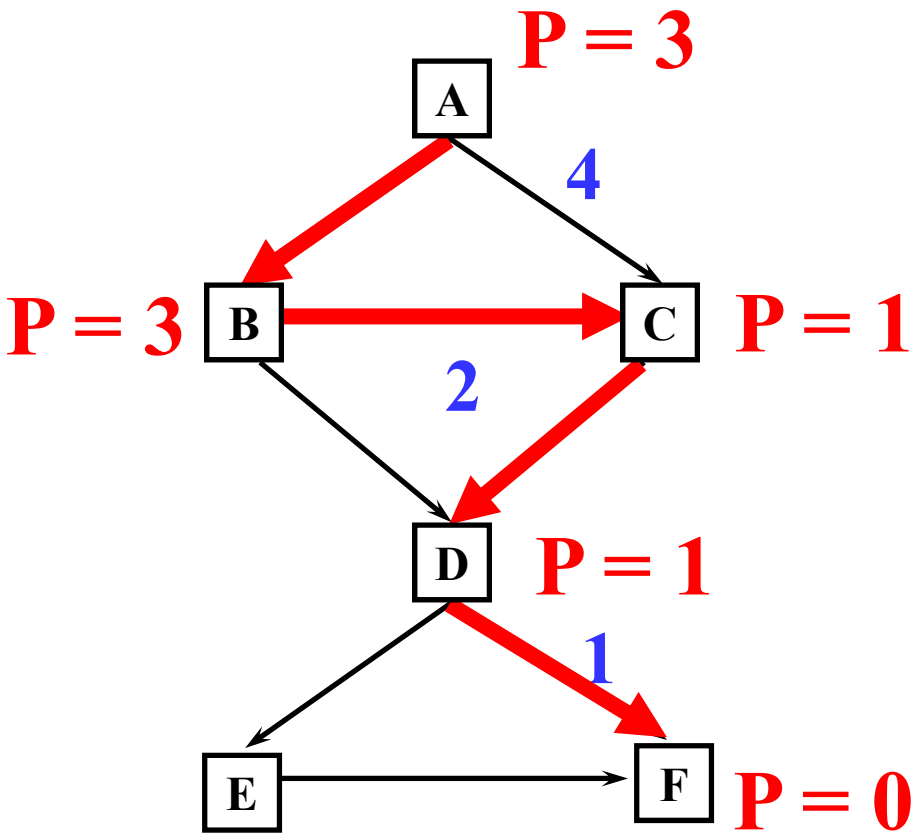




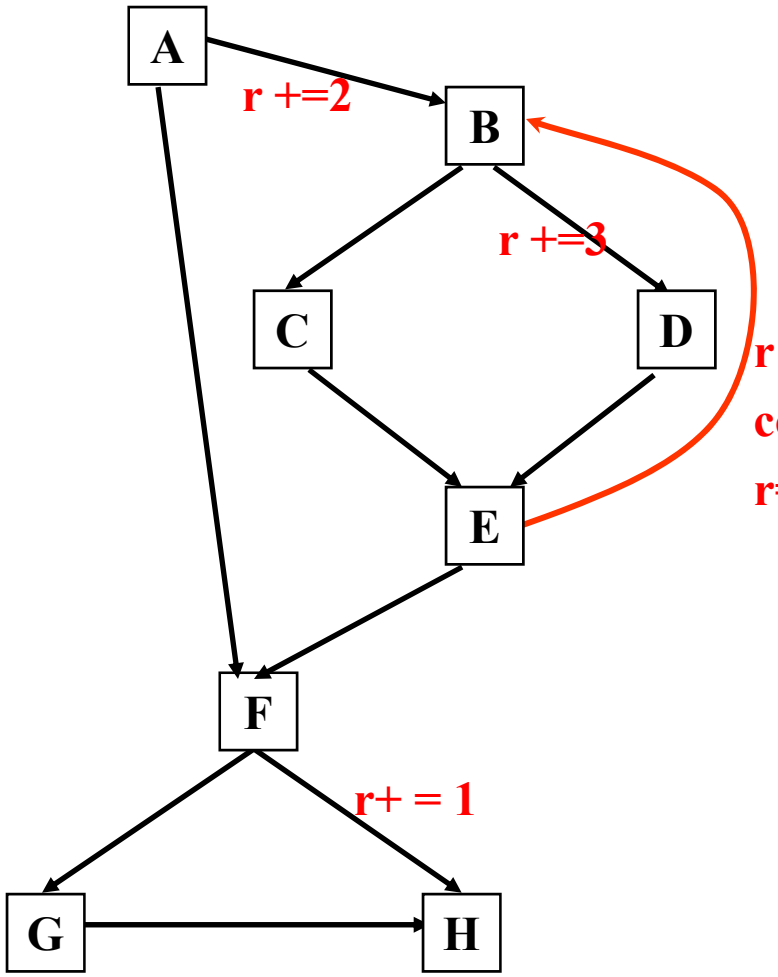
Path Regeneration

Given path sum P, which path produced it?

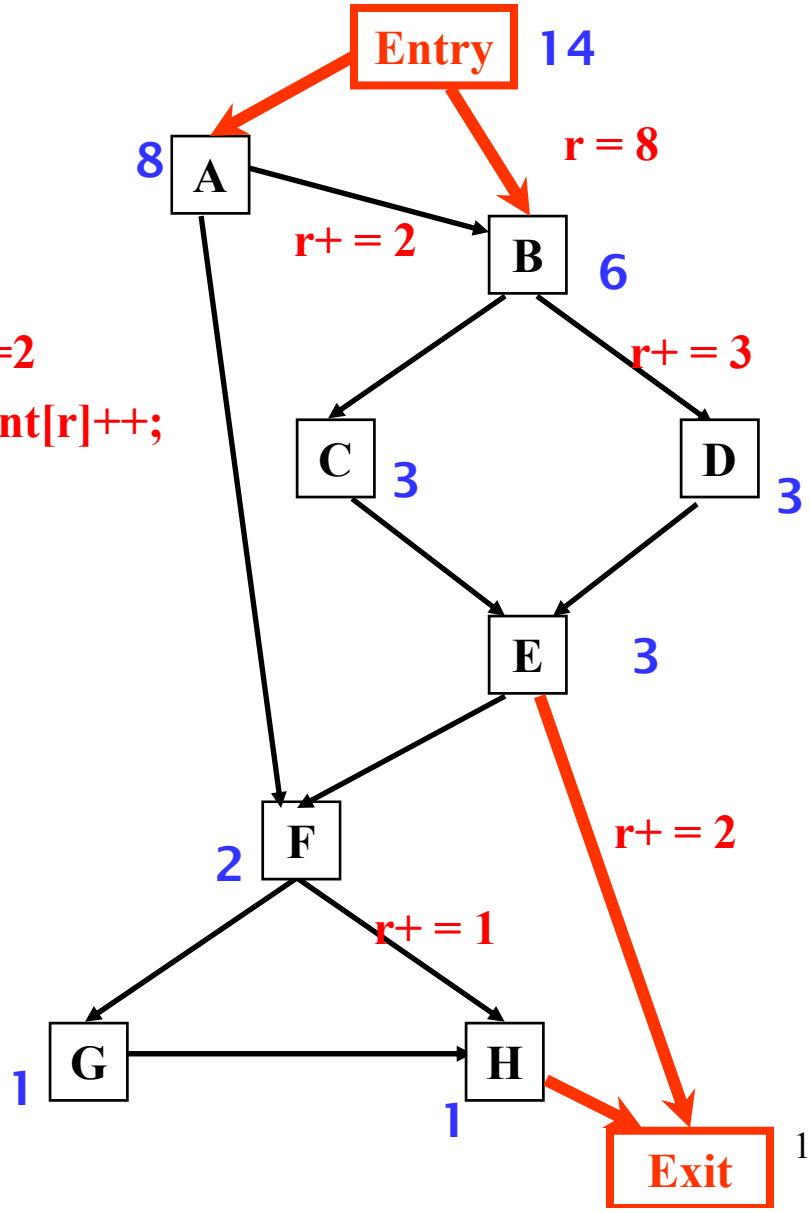




Handling Loops



r +=2
count[r]++;
r=8



Overhead and Others

- EPP causes 40% overhead on average.
- The path explosion problem.
  - If the number of paths is too large to enumerate, which is not very uncommon, hash maps have to be used.
- Can be used to achieve efficient tracing.
- Reading assignment
  - *Efficient Path Profiling*, by T. Ball and J. Larus, Micro 1996
    - The optimization (chord algorithm) is not required.

Object Equality Profiling (OEP)

- OEP discovers opportunities for replacing a set of equivalent object instances with a single representative object.
  - Replacing an object x with an object y means replacing all references to x by references to y.
  - Requires x and y have same field values.
- Object oriented programs typically create and destroy large numbers of objects. Creating, initializing and destroying an object consumes execution time and also requires space for the object while it is alive.
  - Many objects are identical

- In the white paper "*WebSphere Application Server Development Best Practices for Performance and Scalability*". Four of the eighteen "best practices" are instructions to avoid repeated creation of identical objects (in particular, "Use JDBC connection pooling", "Reuse data sources for JDBC connections").
- Merging objects reduces memory usage, improves memory locality, reduces GC overhead, and reduces the runtime costs of allocating and initializing objects.

An Example

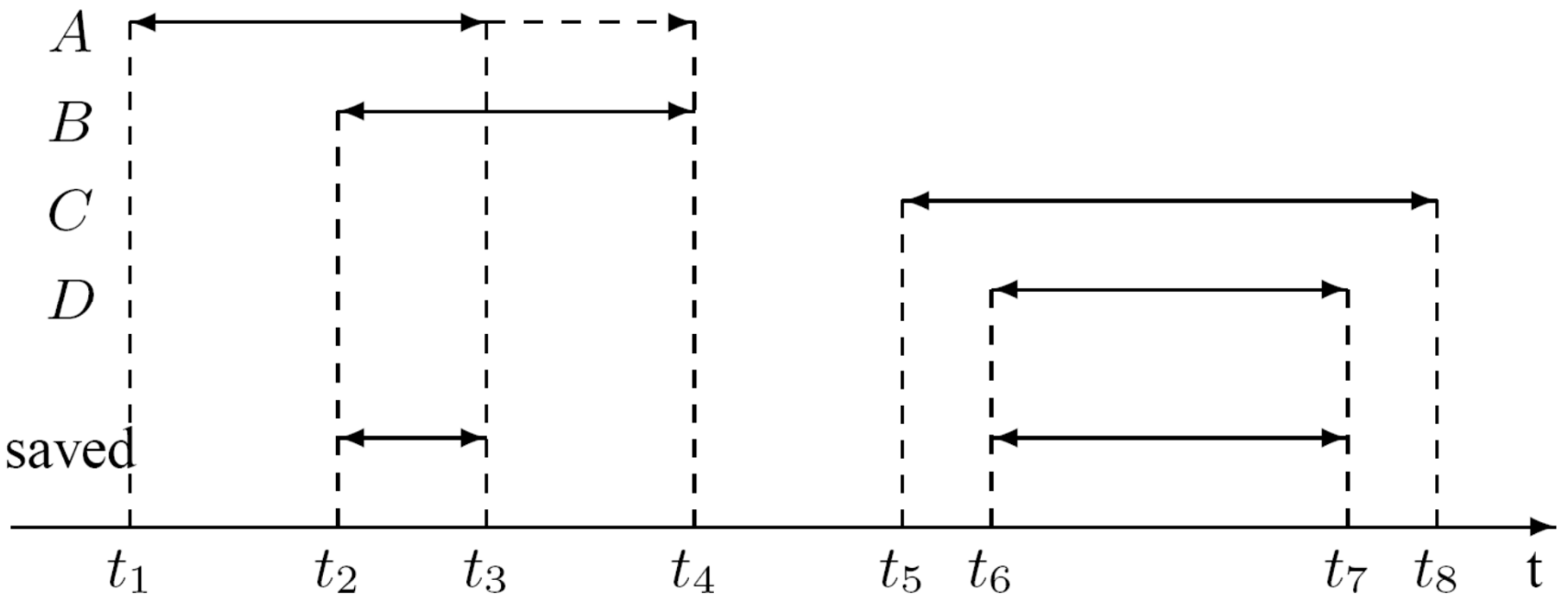
```
class Eval {
  public Value evaluate(Integer i) {
    return new Value(i);
  }
}

class CachedEval extends Eval {
  private HashTable cache = new HashTable();
  public Value evaluate(Integer i) {
    Value v = (Value)cache.get(i);
    if (v == null) {
      v = new Value(i);
      cache.put(i, v);
    }
    return v;
  }
}
```

Mergability

- The objects are of the same class.
- Each pair of corresponding field values in the objects is either a pair of identical values or a pair of references to objects which are themselves mergeable.
- Neither object is mutated in the future.
- The objects have overlapping lifetimes.

Mergability Profiling



- The profiler produces triples <class, allocation site, estimated saving>
- Information to collect
  - Allocation times
  - The last references
  - Field values

Results

- Reduce the memory footprints of two SpecJVM programs by 37% and 47%.
- In program DB, which is a database application

  entry.items.addElement (new String(buffer, 0, s, e-s));

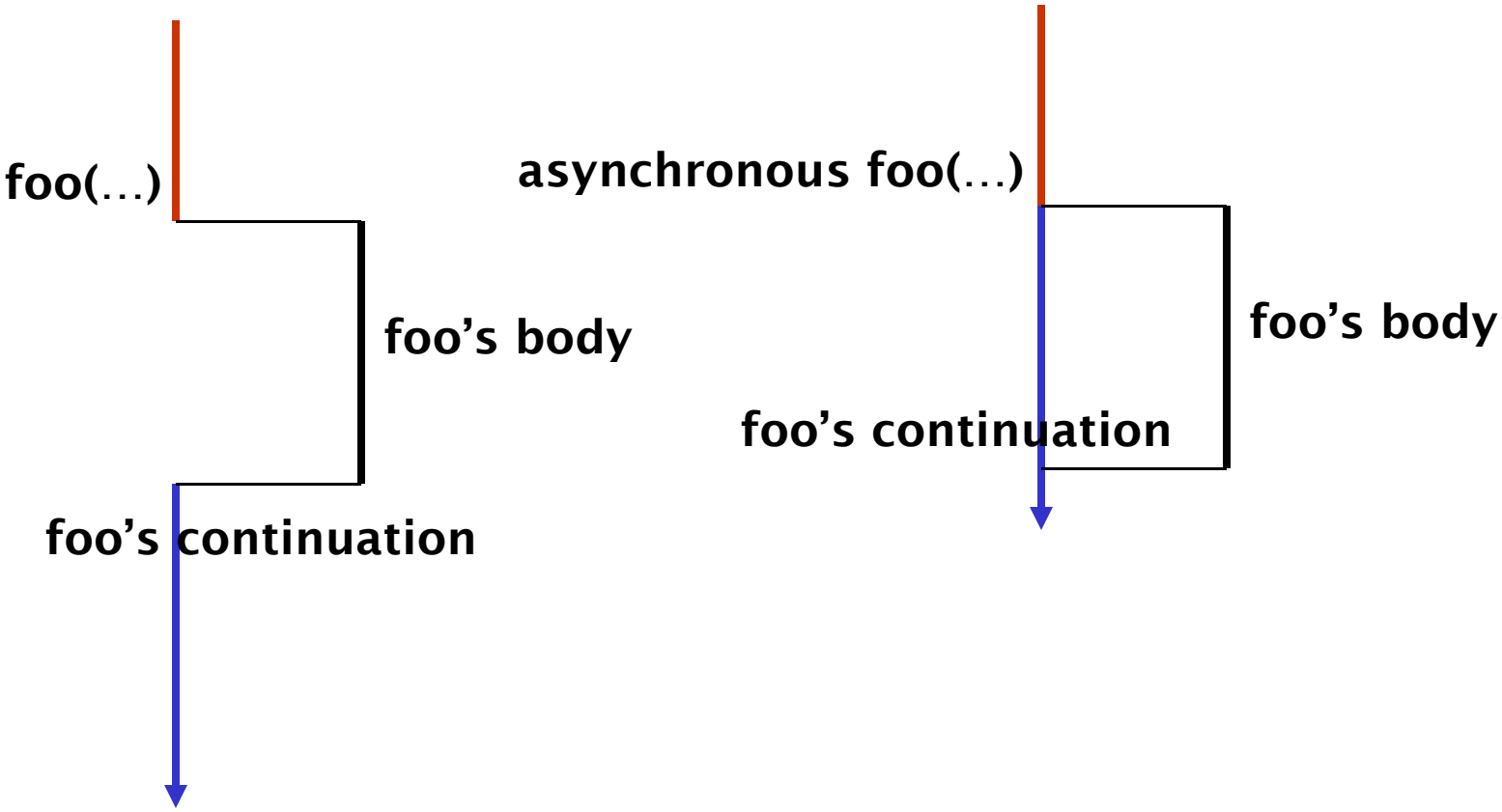
Extra Credit Challenge

- Encoding backtrace
  - A very useful feature of GDB is backtrace, which captures the sequence of call sites leading from the main function to the current execution point. For instance, in case of a segfault, the user can use the "bt" command to investigate the current call sequence.
  - Consider the sequence of call sites as a path in the call graph from the main function to the current execution point. Please design an efficient encoding scheme for call paths. The requirement is that it should be able to distinguish the multiple call paths to the same program point. The scheme ought to be minimal, meaning using the minimal number of ids. Apply your technique to the following program.

```
A ( ) {     B ( ) {     C ( ) {     D ( ) {       E ( ) {       F ( ) {
   B( );       D( );       D( );       E( );         segfault;
   C( );       E( );     }             F( );       }             }
}           }                         }
```

Challenge (1 extra credit)

- A recent trend to parallelize a sequential program is to spawn a method call as a separate thread.



- Devise a profiler that identifies method calls that are amenable to such parallelization.
  - Hint: you ought to consider if dependences between a method and its continuation are broken in the parallelized version.
  - You can assume a dependence detector.