What We Do Statically

- Run-time Constant Identification
  - Folding and propagation of RTCs
- Extracting Set-up Code and Templates
- Run Standard Optimizations
- Code generation

Related Work

- Leone and Lee (Fabius)
  - + safe even with bad annotations
  - - functional language, fewer optimizations

- Consel and Noel
  - + interprocedural
  - - fewer optimizations, structured code only

- Engler, Hsieh and Kaashoek(‘C)
  - + more flexible
  - - more programmer work

Future Work on the Stitcher

Currently Supported:

- Removes static branches and associated dead code
- Convert multiplies to shifts and adds
- Some scheduling of RTC loads
- Places small RTCs in immediate fields

Extensions:

- Better scheduling of instructions (especially multiple dispatch)
- Better register allocation
- Account for cache effects
- Use static information more aggresively to speed stitcher

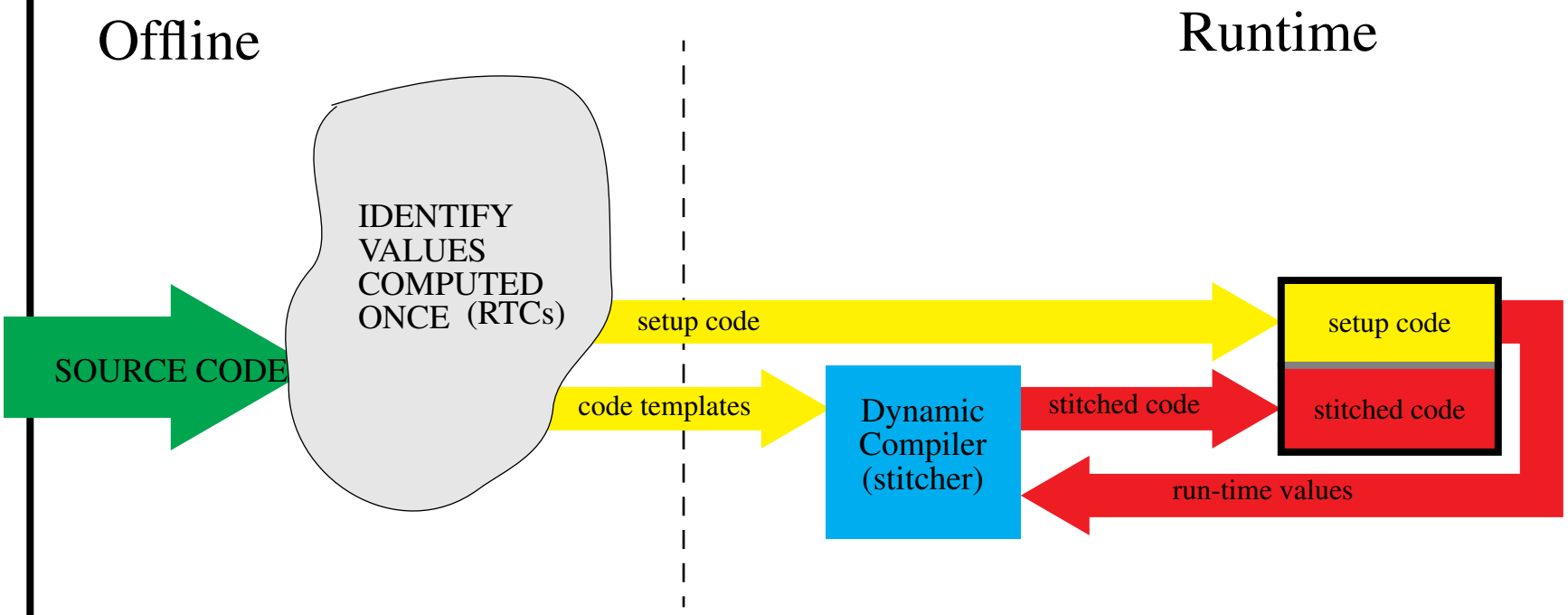
Future Work

Currently supported	Extensions
RTCs that never change	General framework for RTCs that change infrequently
RTCs that have a few values	
Multiple exit/single backedge loops	Multiple backedge loops
Multiple disjoint DRs	
Intraprocedural analysis	Interprocedural analysis
Programmer annotations	Automatic RTC detection

Results (present and future)

Application	Speed up	DC overhead
Reverse-polish stack-based desk calculator	1.9	2-15%
Scalar-matrix multiply	1.6	4-9%
Record comparison		
POV (raytracer)		
Xlisp-BC		
robot design		
sparse matrix		
spin dispatcher		

The Framework So Far



- Need to replace blob with compiler pass

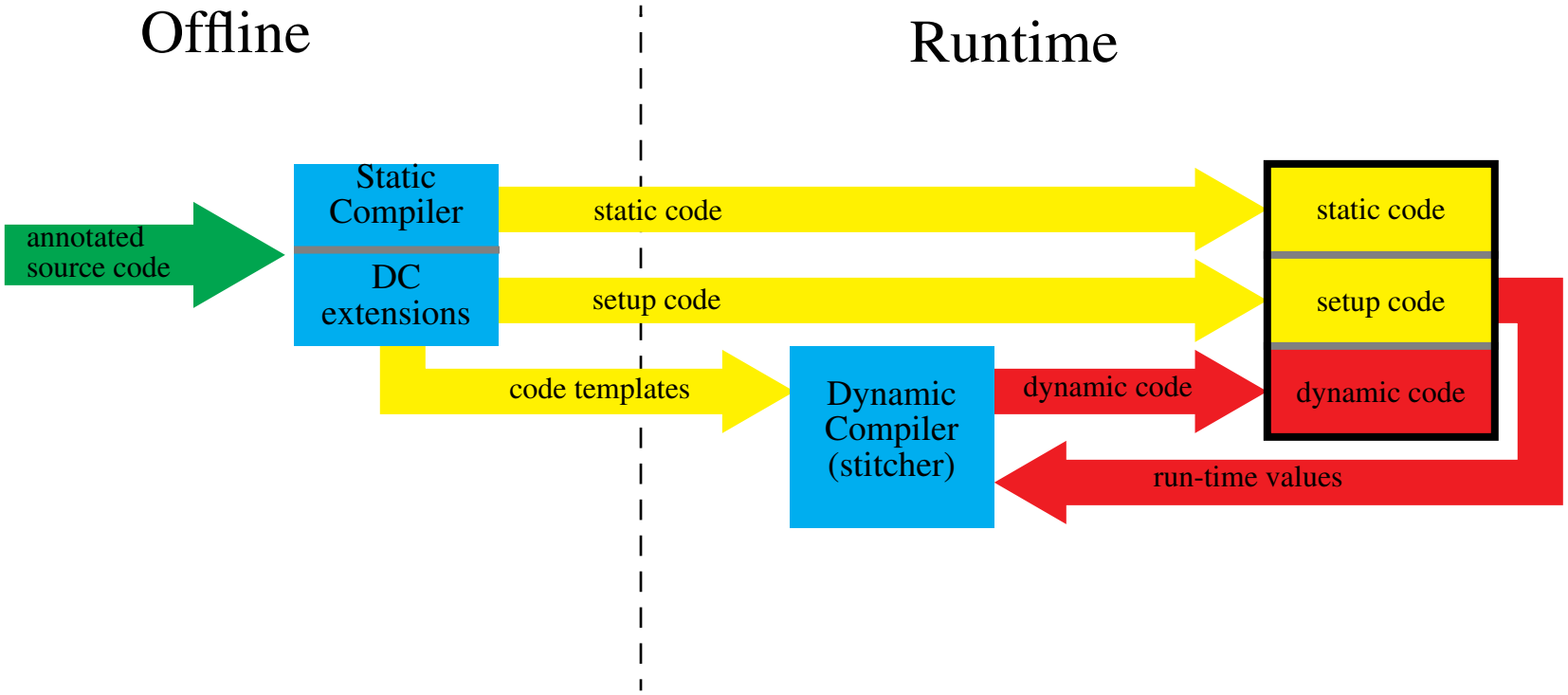
Annotated Cache Simulator

```
cacheResult cacheLookup(void *addr, Cache *cache) {
dynamicRegion(cache) {
  unsigned blockSize = cache->blockSize;
  unsigned numLines = cache->numLines;
  unsigned tag = (unsigned) addr / (blockSize * numLines);
  unsigned line = ((unsigned) addr / blockSize) % numLines;
  setStructure **setArray = cache->lines[line]->sets;
  unsigned associativity = cache->associativity;
  int set;
  unrolled
   for (set = 0; set < associativity; set++) {
    if (setArray[set]->tag == tag) return CacheHit;
   }
  return CacheMiss;
 }
}
```

Specialized Cache Simulator

```
unsigned tag = (unsigned)addr >> 14;
unsigned line = ((unsigned)addr >> 5) & 511;
setStructure **setArray = cacheLines[line]->sets;
if (setArray[0]->tag == tag) return CacheHit;
if (setArray[1]->tag == tag) return CacheHit;
if (setArray[2]->tag == tag) return CacheHit;
if (setArray[3]->tag == tag) return CacheHit;
return CacheMiss;
```

Dynamic Compilation Pipeline



Setup Code

```
HOLE_0 = cache->blockSize;
HOLE_1 = cache->numLines;
HOLE_2 = HOLE_0 * HOLE_1;
HOLE_3 = cache->lines;
associativity = cache->associativity;
i=0;
for (set = 0; set < associativity; set++) {
 HOLE_4:i = set;
 i++;
}
```

Template Code

```
cacheResult cacheLookup(void *addr, Cache *cache) {



  unsigned tag = (unsigned) addr / HOLE_2;
  unsigned line = ((unsigned) addr / HOLE_0) % HOLE_1;
  setStructure **setArray = HOLE_3[line]->sets;

  int set;
  unrolled
   for (i) {
    if (setArray[HOLE_4:i]->tag == tag) return CacheHit;
   }
  return CacheMiss;

}
```

Cache Simulator Example

```
cacheResult cacheLookup(void *addr, Cache *cache) {

  unsigned blockSize = cache->blockSize;
  unsigned numLines = cache->numLines;
  unsigned tag = (unsigned) addr / (blockSize * numLines);
  unsigned line = ((unsigned) addr / blockSize) % numLines;
  setStructure **setArray = cache->lines[line]->sets;
  unsigned associativity = cache->associativity;
  int set;

   for (set = 0; set < associativity; set++) {
    if (setArray[set]->tag == tag) return CacheHit;
   }
  return CacheMiss;

}
```

Our Approach

- Do as much work as possible at compile time
  - Compile all the way to machine code

- Leave only fast optimizations for runtime
  - e.g. Inserting constants into immediate fields

What is Dynamic Compilation?

- Deferring compilation until runtime

Why do it?

- Code optimized for runtime values runs faster

Why is it difficult?

- Time saved by optimization should exceed time spent optimizing

Talk Contents

What is Dynamic Compilation?

Example: Cache Simulator

Summary of our system

Results

Further Work

Joel Auslander, Matthai Philipose, Craig Chambers, Susan J. Eggers, and Brian N. Bershad

Fast, Effective Dynamic Compilation

Matthai Philipose, Joel Auslander

7 October 1995