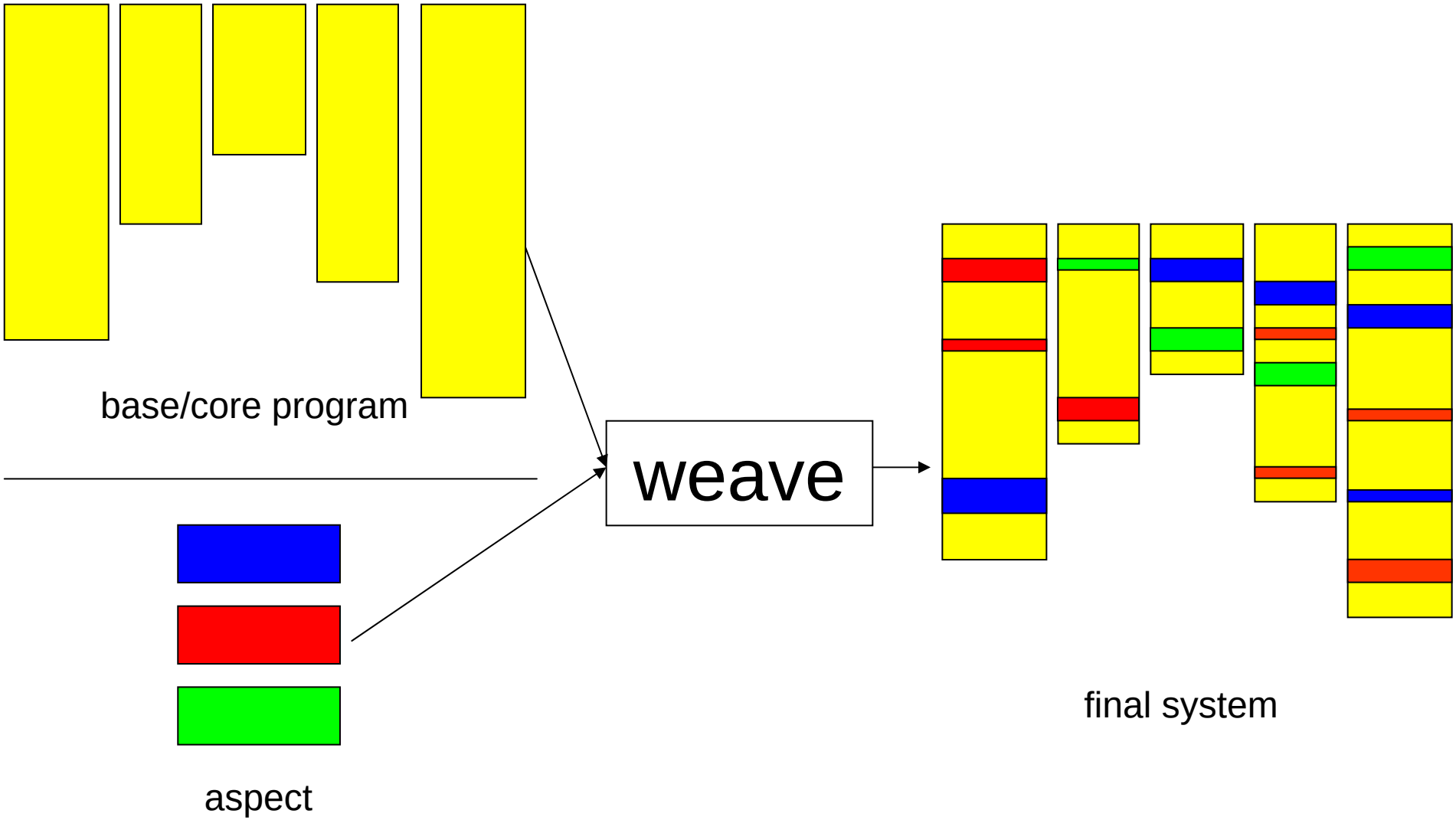
AspectC.net

Coding with AspectC

Hans-Arno Jacobsen - ECE344

AOP Key Idea



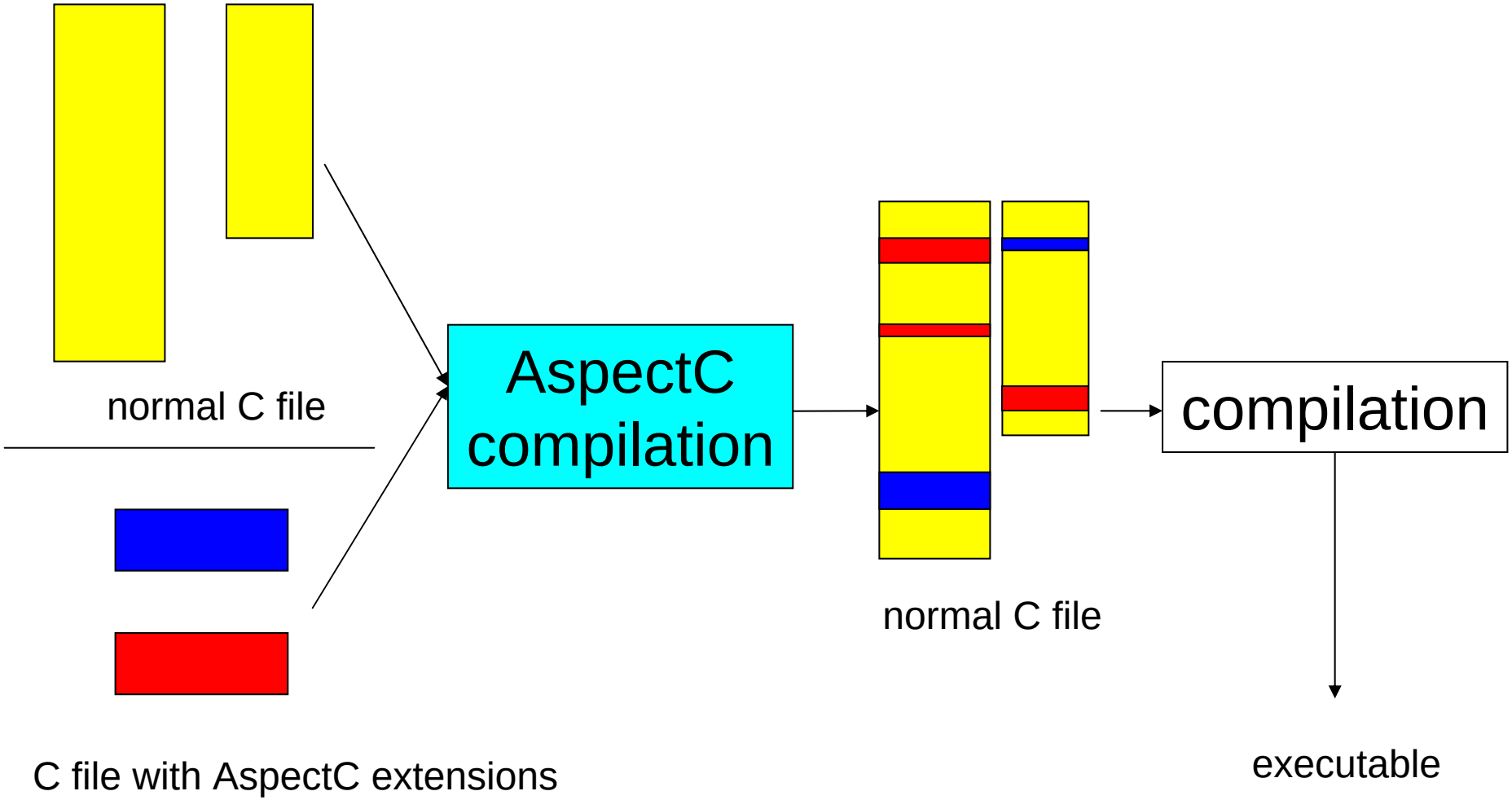
© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

Aspect-oriented Programming

- AOP is
  - a programming paradigm that aims to support the modularization of crosscutting concerns in software
  - **complementary** to existing paradigms
- Emerged about 10 years ago from different research efforts studying the separation of concerns in software
- Supported in industry today by IBM, BEA,…
- AOP support is available for Java, C++, C, PHP, …
- AspectJ, AspectC++, AspectC, AOPHP, …

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Key Idea



© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

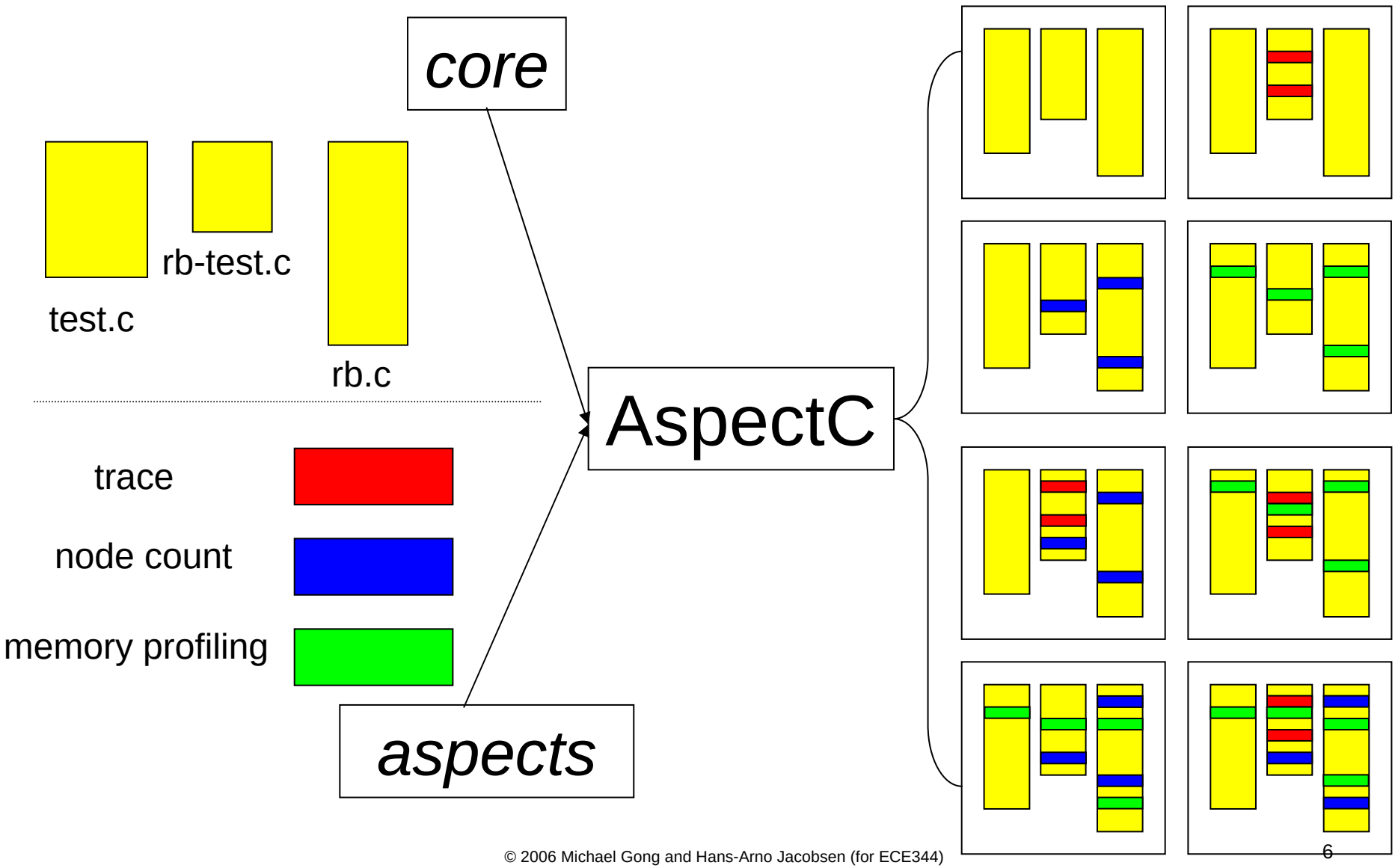
Use Cases

AspectC.net

- Software produce lines (previous slide)
- Isolation of crosscutting concerns for increasing software maintenance
- Software customization
- Debugging support
  - Tracing, logging, profiling, …
- Coding policy enforcement (not supported in AspectC)

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AOP-based Software Product Lines



© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

Relevant Features for OS/161 A #2

AspectC.net

- Join points
- Pointcuts (i.e., sets of join points)
  - A #0: Matched one place in the code
  - A #2: Match multiple places in the code
- Advice: before and after

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

OS/161 and AspectC Precautions

- Also documented in the assignment handout
- Profiler implementation via aspects
  - OS/161 is multi-tasking
  - Shared data is touched concurrently and needs to be protected.
  - This is unlike, the thread-unsafe memory profiler we show in the AspectC Tutorial (web site).
  - However, it may serve as starting point.

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

OS/161 and AspectC Precautions

- Various parts of the kernel are not available until the bootup sequence is completed
  - See the boot function in `kern/main/main.c`
  - E.g., the thread subsystem is not initialized until bootup completes
  - Consequently profiling thread related information will fail and lead to unpredictable behavior
- Profiler should not begin until bootup is finished
  - Add a global variable that tracks completion of bootup and test it in the advice
  - Or, exclude certain functions / files in pointcut definitions via **`!infile(...)`** or **`!infunc(…)`**
  - Do not worry about profiling `thread_bootstrap`

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

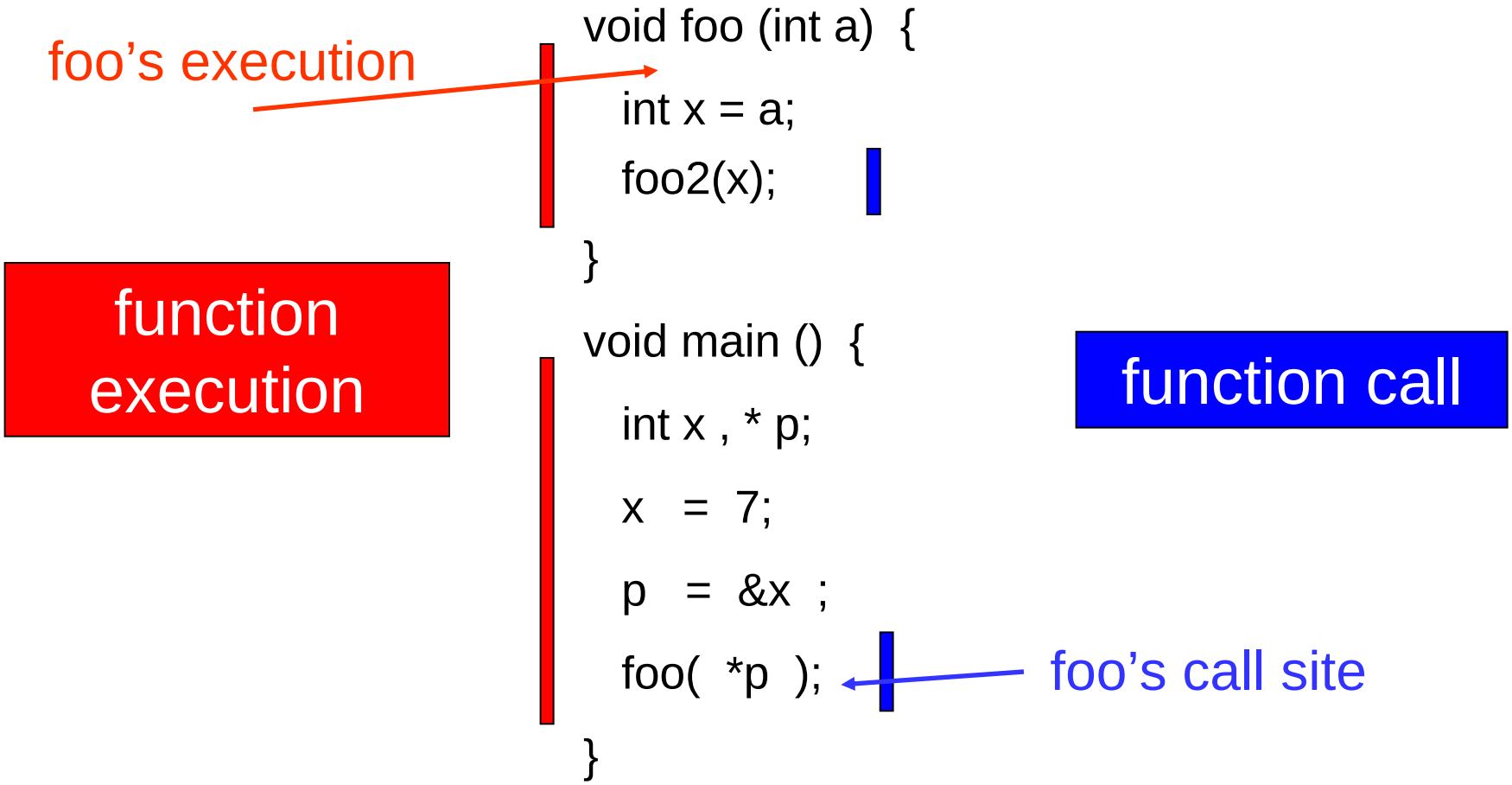
AspectC Features

- **join point**: call and execution join points
- **advice**: before, after and around
- **pointcut**:call(), execution(), args(), infile(), infunc(), result()
- pointcut composition: &&, || , !
- named pointcut
- proceed() call
- wildcard character matching through “$” and “…”
- recognize gcc extended keywords
  - __extension__,__attribute__, __builtin_va_list,__inline__ , __asm, …
- **cflow**() support (also under multi-threading)
- reflective information about join points
- static crosscutting support
  - add new struct/union member: intype() pointcut and introduce() advice
- generated code is thread-safe

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Join Point Model

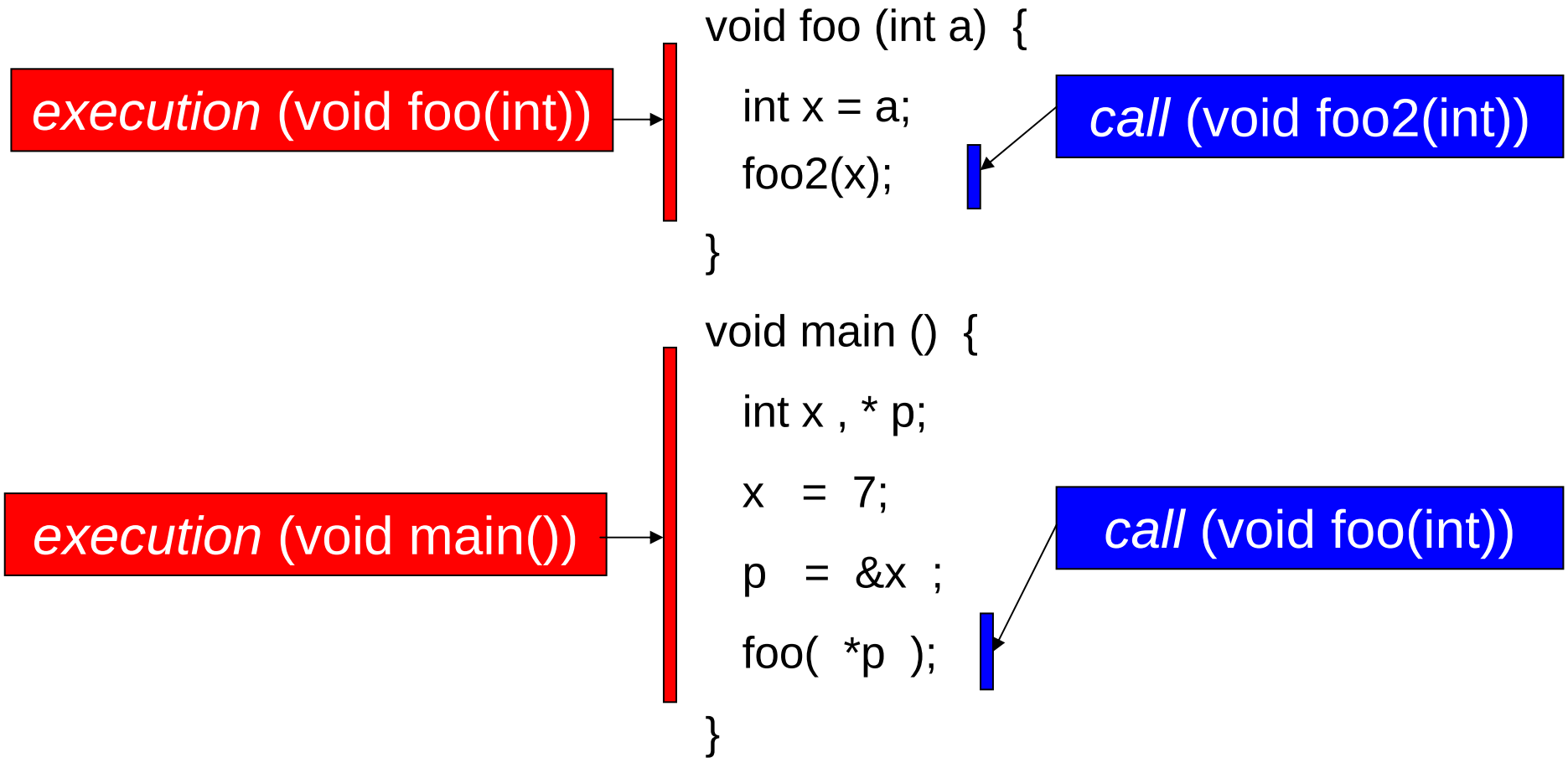
- join point
  - the **location** in the **base** program where **aspects** take effect



© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Pointcut

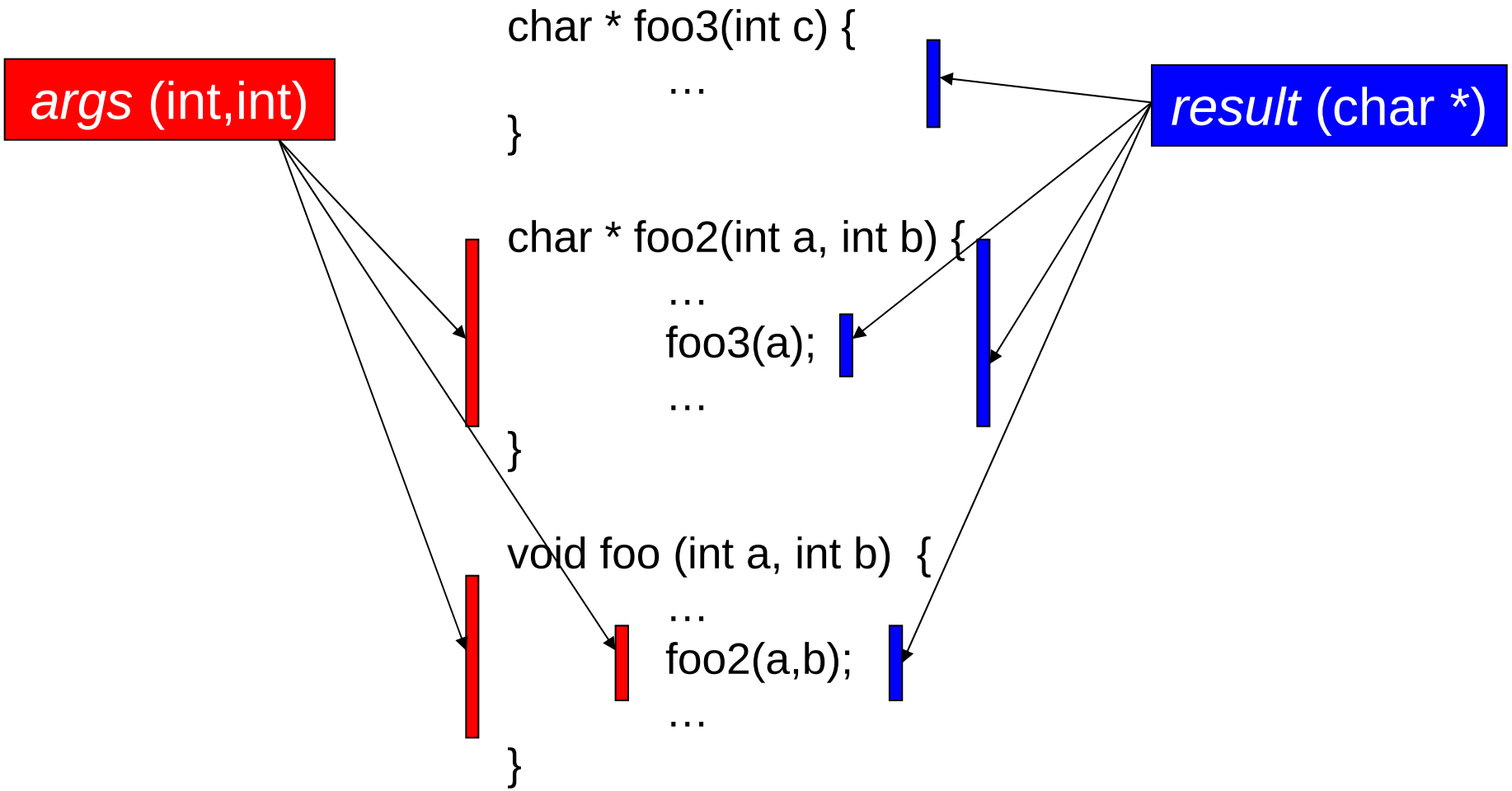
- pointcut
  - a language construct to denote sets of join points



© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Pointcut

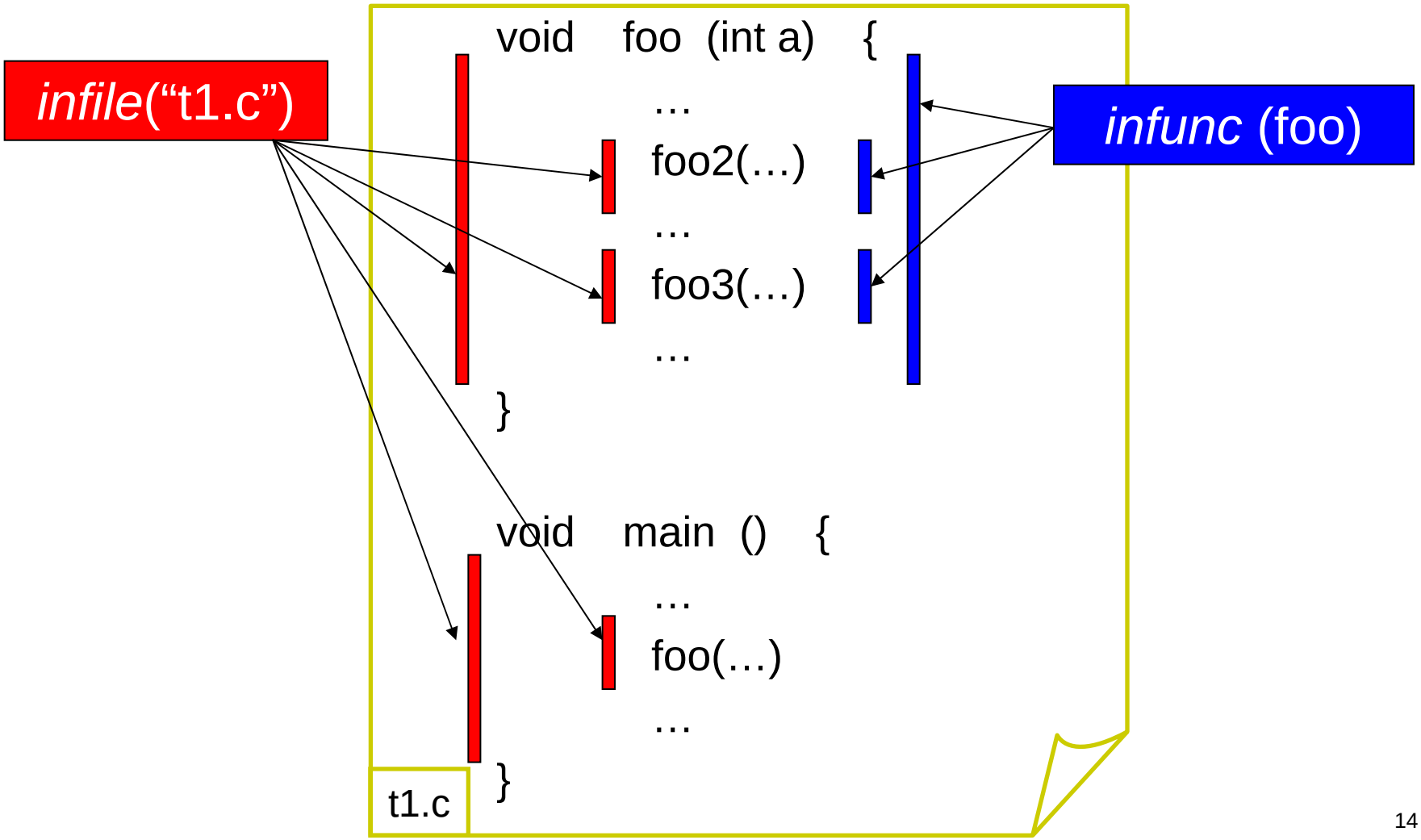
- *args* (parameter-type-list)
- *result* (return-type)



© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

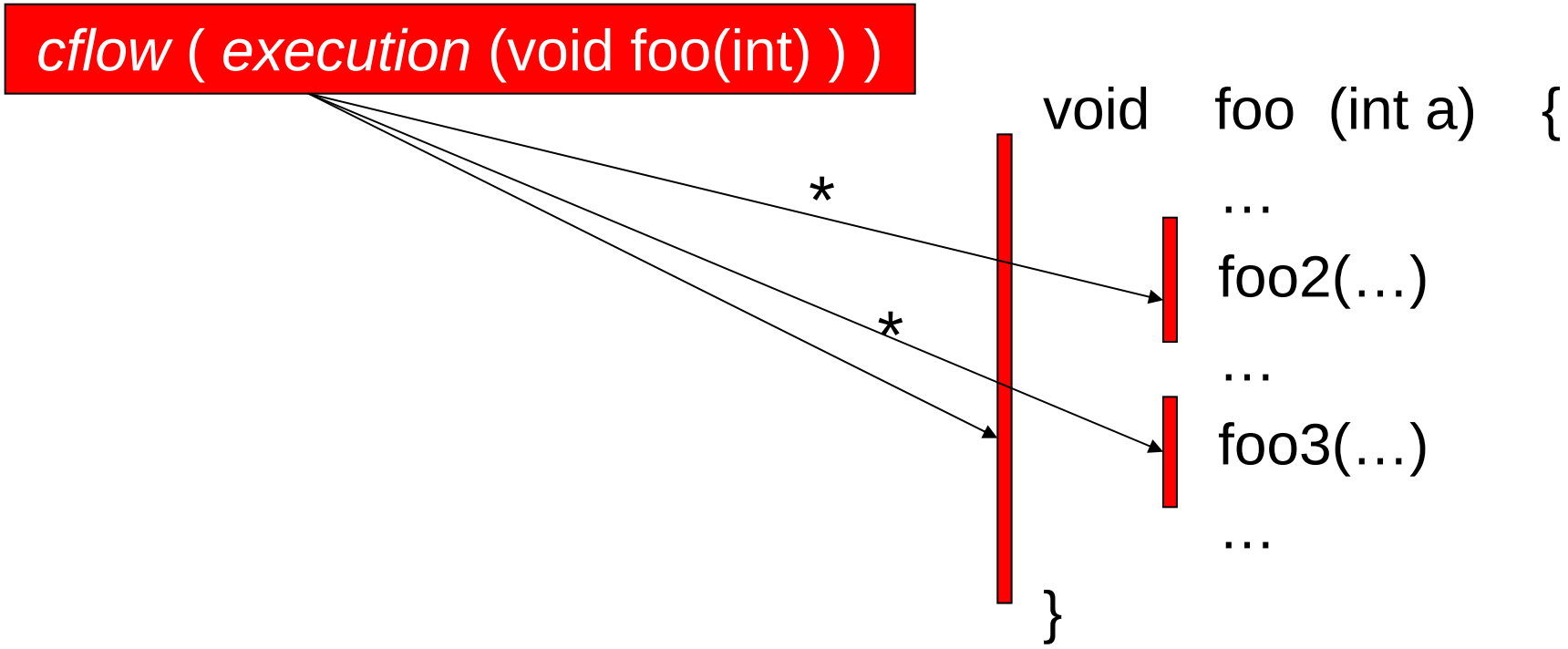
AspectC Pointcut

- *infile* (“file-name”)
- *infunc* (func-name)



AspectC Pointcut

- *cflow* (pointcut-declaration)
  - all join points happening under the control flow of other join points



* all join points happening inside foo2() or foo3() function calls

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Pointcut Composition

- compose pointcuts with &&, ||, ! (i.e., *and*, *or*, and *not*)

infile (“t1.c”) && *infunc* (foo)

execution (void foo(int)) || *args* (int,int)

(*result* (char *) || *infunc* (foo)) && ! *call* (void foo2(int))

cflow (*execution* (void foo(int))) && *call* (void foo2(int))

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Named Pointcut

- *pointcut* name (parameter-list) : pointcut-declaration

pointcut CallFoo2(): *call* (void foo2(int))

CallFoo2() || *args* (int,int)

cflow (*execution* (void foo(int))) && CallFoo2()

(*result* (char *) || *infunc* (foo)) && ! CallFoo2()

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Pointcut Using Wildcard Character

$: match any single item

... : match any list of items

call (lon$ foo$())

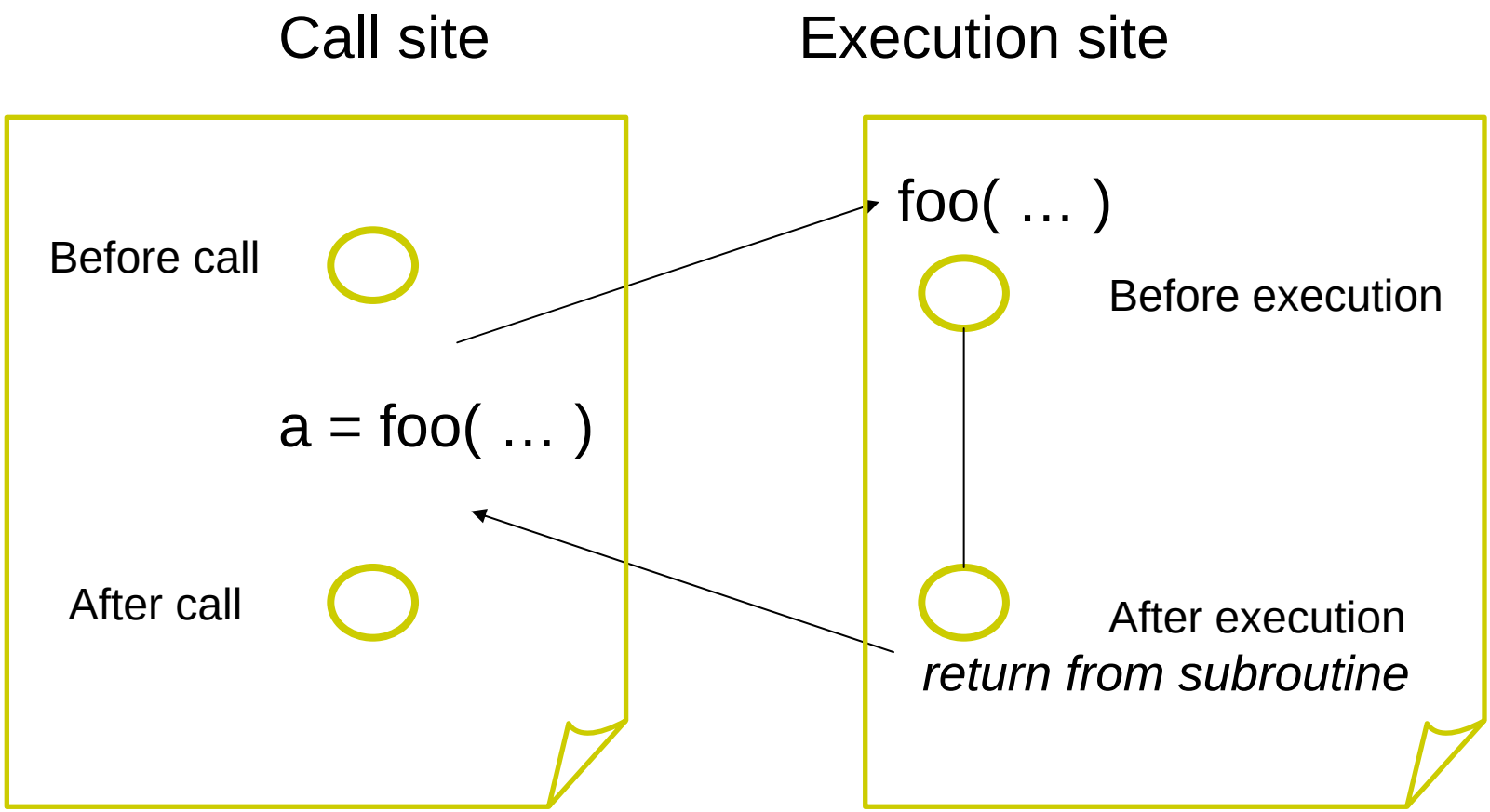
call (long long foo())
call (long foo2())
call (long int foo3())
...

args (int , ... , char *)

args (int, char *)
args (int, char, char *)
args (int, int *, char *)
args (int, char, char , char *)
...

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

Call Site vs. Execution Site



© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Advice

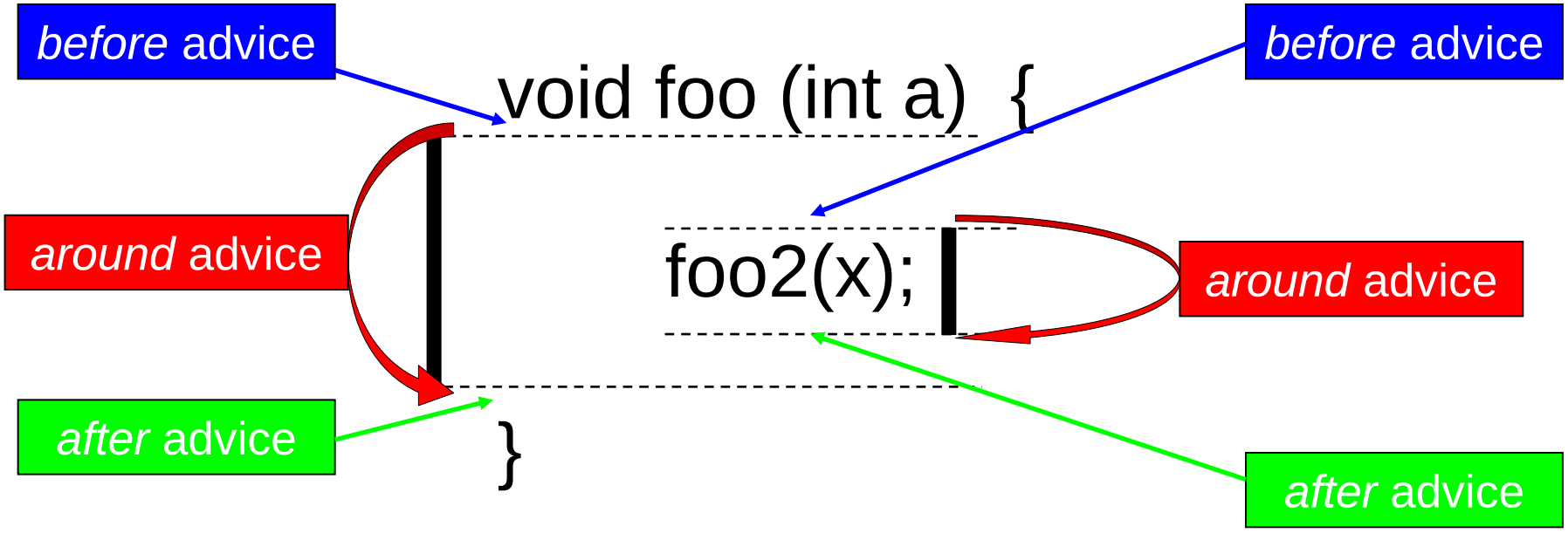
- the code to run for a pointcut

  before/after (parameter-list) : pointcut-declaration

  { //advice body }

  return-type *around* (parameter-list) : pointcut-declaration

  { //advice body }



© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Advice Example

AspectC.net

```
before (): execution (void foo(int)) {
        printf(" before exec\n");
}
```

```
void foo (int a) {
    foo2(x);
}
```

```
void around ():
    execution (void foo(int)) {
        printf("around exec\n");
}
```

```
after (): execution (void foo(int)) {
        printf(" after exec\n");
}
```

AspectC Advice Example

AspectC.net

```
before (): call (void foo2(int)) {
        printf(" before call\n");
}
```

```
void foo (int a)  {
        foo2(x);
}
```

```
void around ():
        call (void foo2(int)) {
        printf("around call\n");
}
```

```
after (): call (void foo2(int)) {
        printf(" after call\n");
}
```

AspectC Advice Example

- access argument value by using *args* ()
- access return value by using *result* ()

```
before (int i): call (void foo2(int))  && args (i) {
        printf(" before call foo2, argument = %d\n", i);
}
```

```
after (int res): call (int foo2(int)) && result (res) {
        printf(" after call foo2, return %d\n", res);
}
```

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Advice Example

- around advice
  - invoke original function via *proceed* ()

```
void around ( ): call (void foo2(int)) {
        printf("around begin\n");
        proceed();
        printf("around end\n");
}
```

```
void foo2(int a) {
   printf("in foo2\n");
}
int main() {
    foo2(3);
}
```

if no *proceed()* used:
  around begin
  around end

```
around begin
in foo2
around end
```

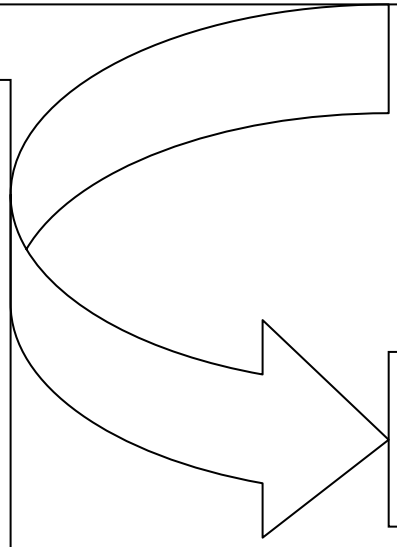
© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Advice Example

- reflective information about join point
  - *this->funcName*, *this->kind*

```
before ( ): call (void foo2(int)) {
        printf("before %s %s \n", this->kind, this->funcName);
}
```

```
void foo2(int a) {
   printf("in foo2\n");
}
int main() {
    foo2(3);
}
```



```
before call foo2
in foo2
```

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

AspectC Advice Example

- wildcard matching increases the usability

```
before ( ): call (void $(…)) {
        printf("before %s %s \n", this->kind, this->funcName);
}
```

```
void fun1() {
   printf("in fun1\n\n");
}

void foo2(int a) {
   printf("in foo2\n\n");
}

void foo3(int a, char * s) {
   printf("in foo3\n\n");
}
```

```
int main() {
    fun1();
    foo2(3);
    foo3(3, "s");
}
```

```
before call fun1
in fun1

before call foo2
in foo2

before call foo3
in foo3
```

AspectC Static Crosscutting

- add new data members to struct/union types

introduce ( ) : *intype* (type-name) {

// new member declaration

}

```
introduce ( ): intype ( struct X ) {
        double x;
        char * parent;
}
```

```
struct X {
    int a;
    char b;
};
int main() {
    printf("size of X = %d\n",
        sizeof(struct X);
}
```

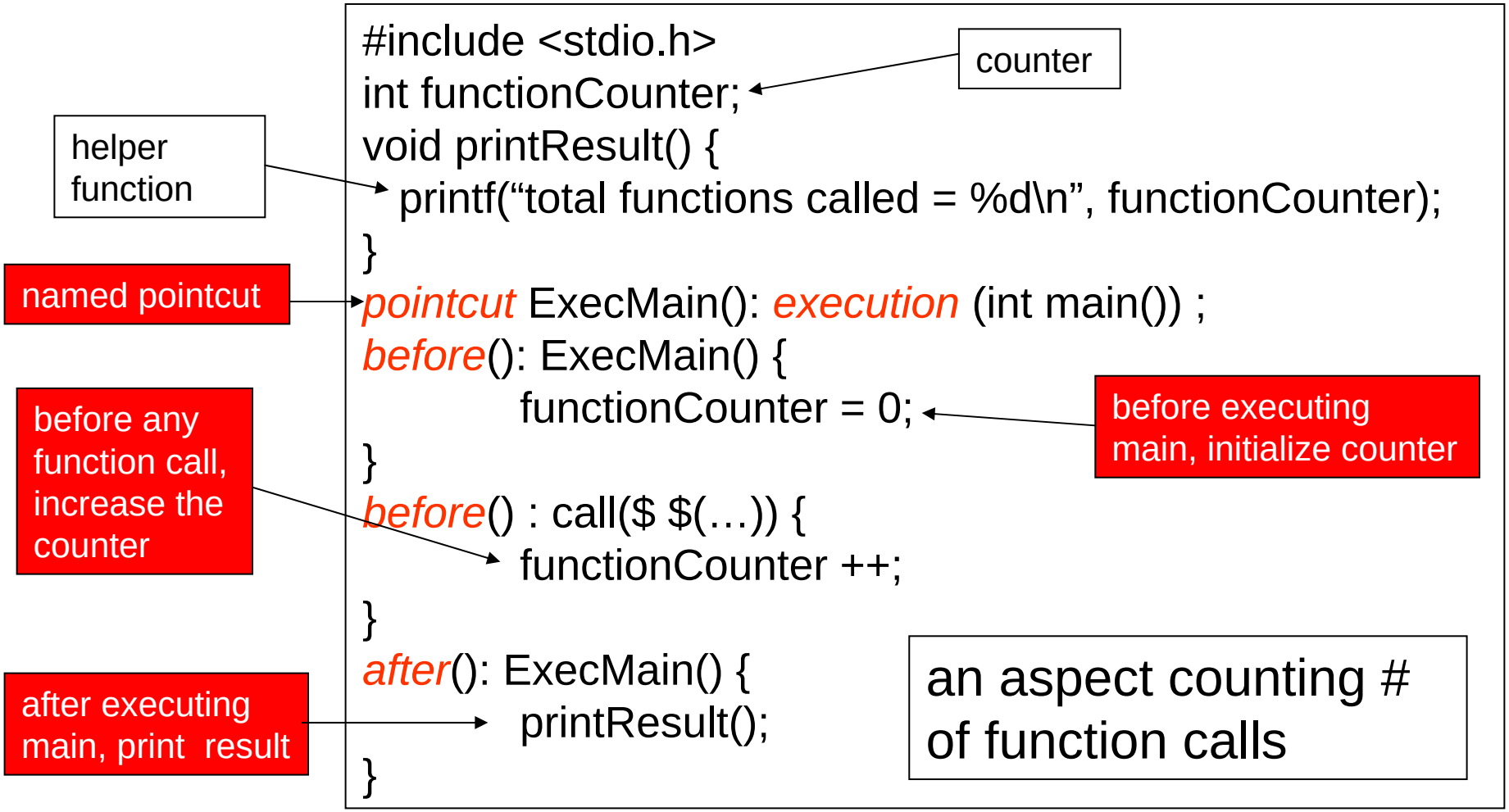
```
struct X {
    int a;
    char b;
    double x;
    char * parent;
}
```

```
size of X = 20
```

© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

Aspects in AspectC

- An *aspect* is a file including **AspectC extensions** & **C code**



© 2006 Michael Gong and Hans-Arno Jacobsen (for ECE344)

More Information

- AOP
  - http://www.aosd.net
- AOP on C
  - http//www.aosd.net/wiki/index.php?title=Aspects_in_C
- AspectC
  - http://www.AspectC.net
  - latest version is 0.5 (Coming in March, 2007)
    - Current ECE344 version is 0.45
  - source code, compiler manual, examples, tutorials, and much more
  - *we welcome any comments or feedback*

Thanks for reading! ☺