Presented at 15th Workshop on Foundations of Aspect-Oriented Languages (FOAL'16)
co-located with Modularity'16, March 15, 2016, Málaga, Spain

An Advice Mechanism for Non-local Flow Control

Hidehiko Masuhara
Kenta Fujita
Tomoyuki Aotani

Tokyo Tech

Example Scenarios of Non-local Flow Control

InitPlugin in your application:

- reads a config. file
- downloads a plugin def.
- constructs a plugin obj.

Scenarios:

1. no config file $\rightarrow$ go without plugins
2. network error $\rightarrow$ start over InitPlugin
3. incompatible plugin $\rightarrow$ download diff. version

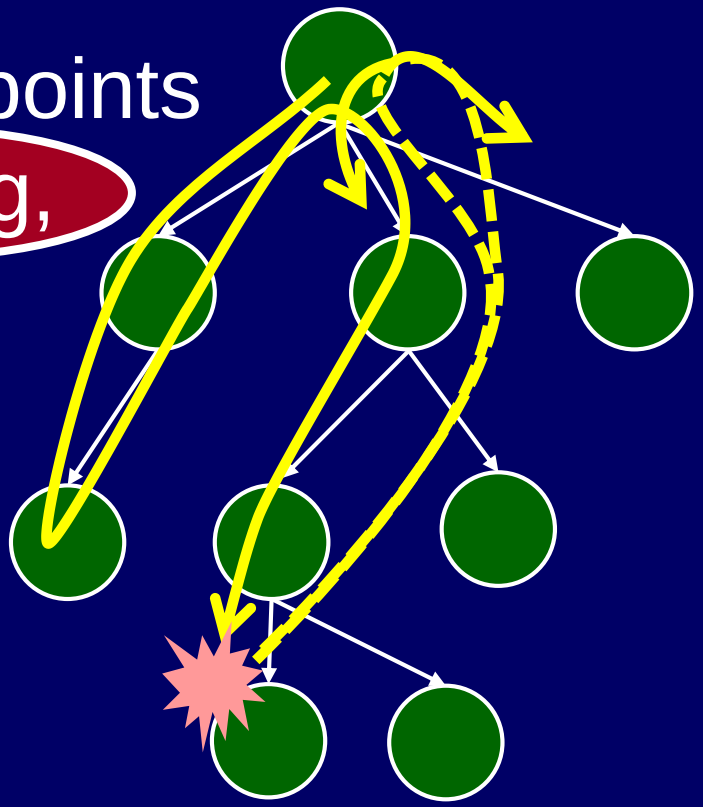
termination

retry

backtrack

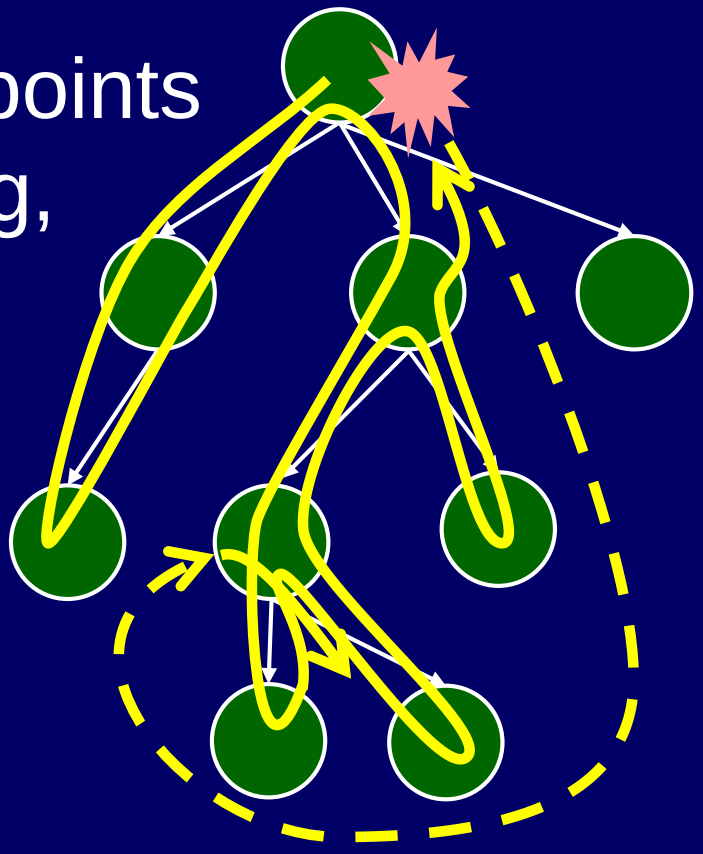
Non-local Flow Control

- is a transition of control
  between 2 join points
  e.g., termination, retrying, backtracking



Non-local Flow Control

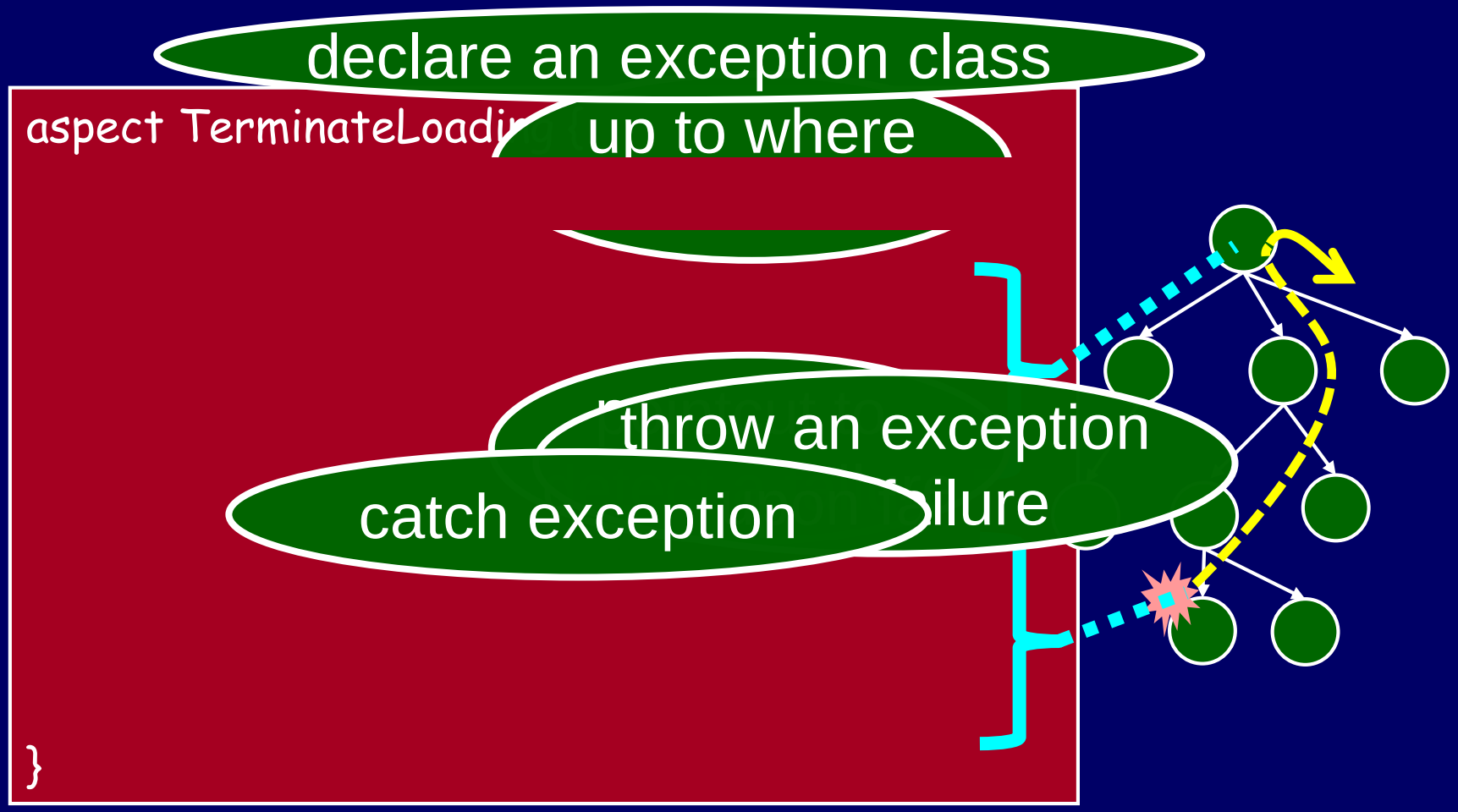
- is a transition of control
  between 2 join points
  e.g., termination, retrying,
  backtracking



"I can write aspects in AspectJ for non-local control flow.
Why a new mechanism?"

True, but those aspects are not ideal

Termination in AspectJ



Termina

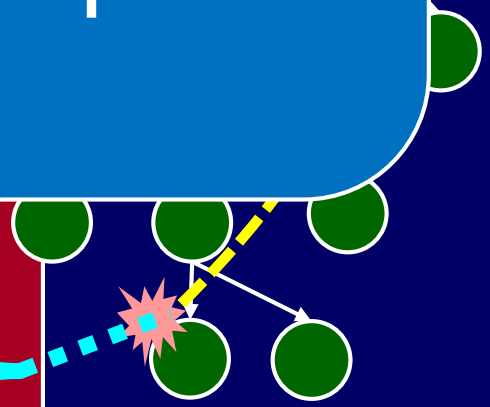
```
aspect TerminateLoadin
  static class NoConfig

  Plugin around(): call(* *
    try { return proceed
    catch (NoConfigE
  }

  before(File f) : call(* *.readFile(File)) &&
                   args(f) && if(!f.exists()) {
      throw new NoConfigError();
}}
```

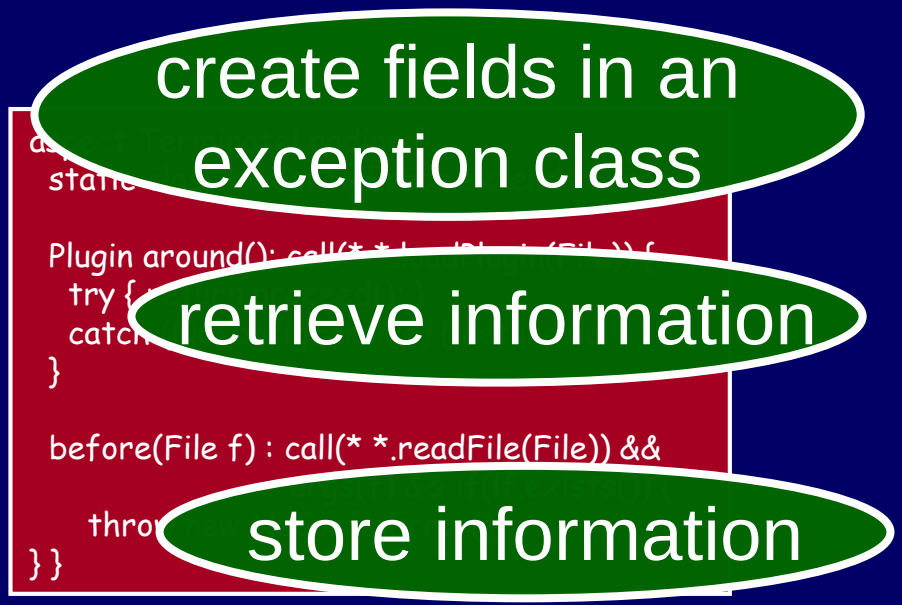
Problems

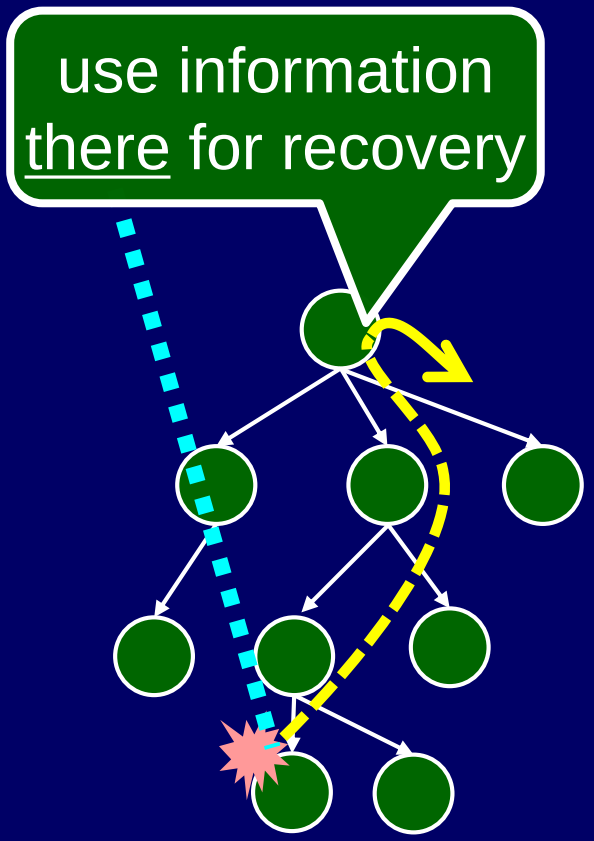
- 2 pieces of advice for 1 concern
- Use of exceptions for pairing



Problem: 2 pieces of advice

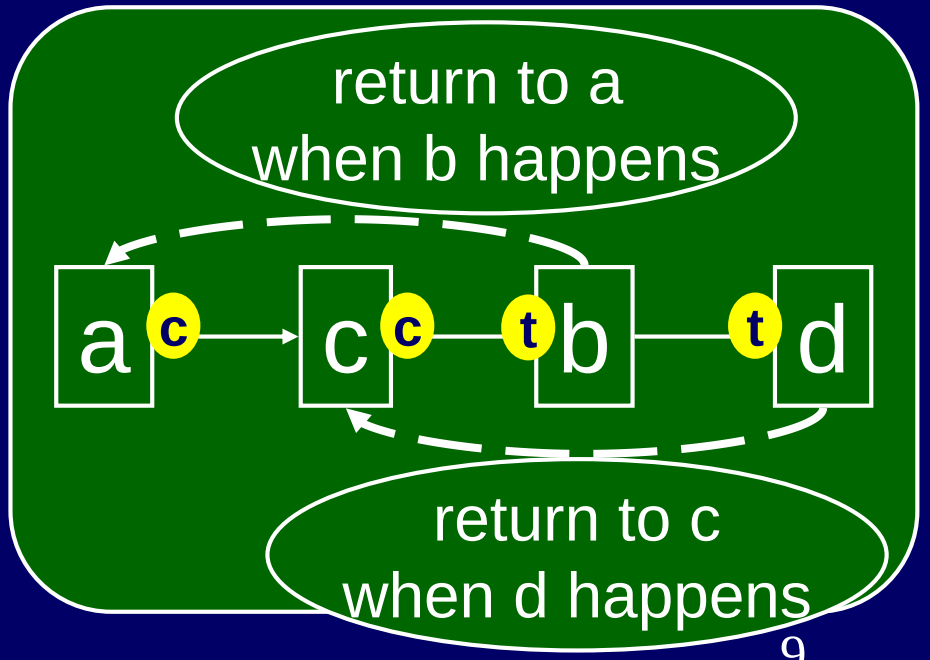
- clumsy
- difficult to pass information





Problem: use of exceptions

- indirect
- requires a unique exception class for each pair of flow control
  - otherwise, it can cause accidental catching



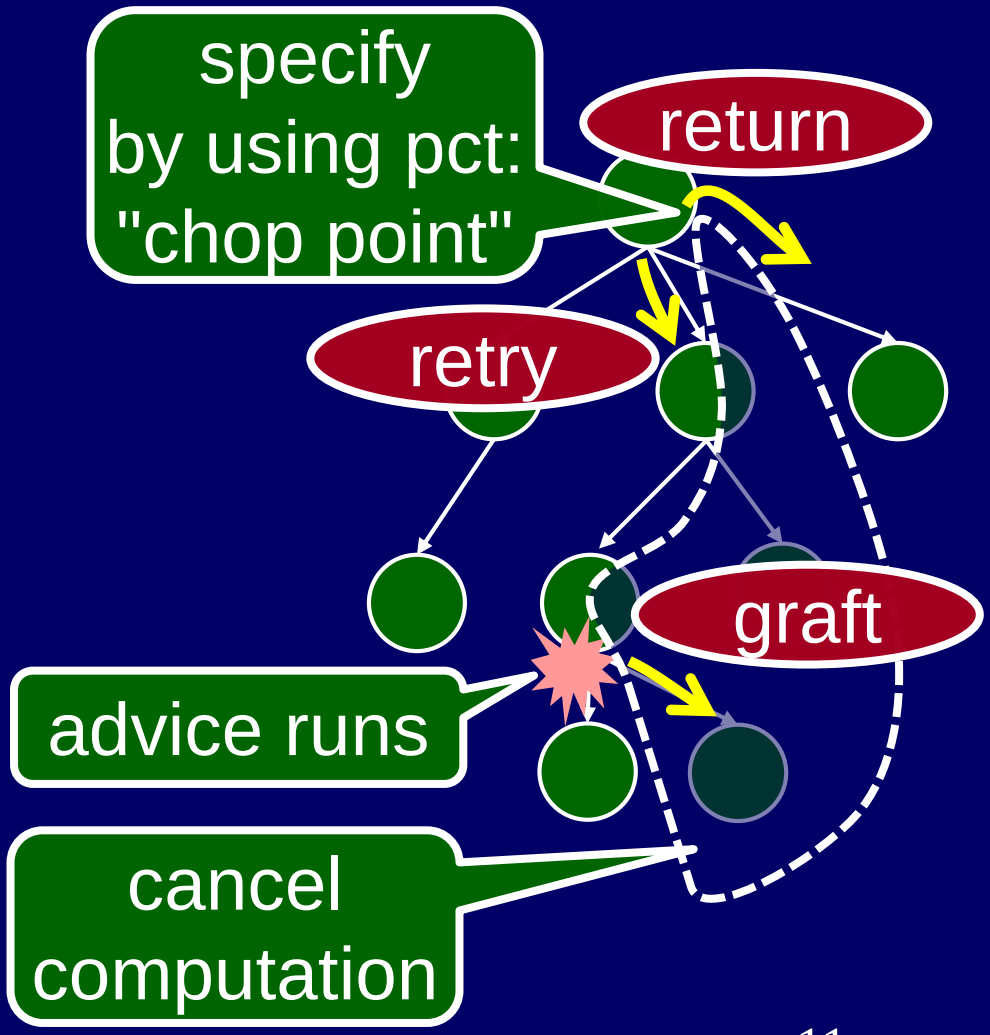
Generic Termination Aspect in AspectJ

```
abstract aspect Termination {
   class MyException extends Error {
     ExceptionHandling throwingAspect;
     Object recorded;
     MyException(ExceptionHandling throwingAspect,
                 Object recorded) {
       this.throwingAspect = throwingAspect;
       this.recorded = recorded; }
     boolean matches(ExceptionHandling catchingAspect) {
       return this.throwingAspect == catchingAspect; } }
   abstract pointcut entry();
   abstract pointcut bad();
   abstract Object recordInformation(JoinPoint jp);
   abstract Object recovery(Object recorded);
   Object around() : entry() {
     try { return proceed(); }
     catch (MyException e) {
       if (e.matches(this)) { return recovery(this.recorded); }
       else { throw(e); } } }

   before(): bad() && cflow(entry()) {
     throw new MyException(this, recordInformation(thisJoinPoint)); } }
```

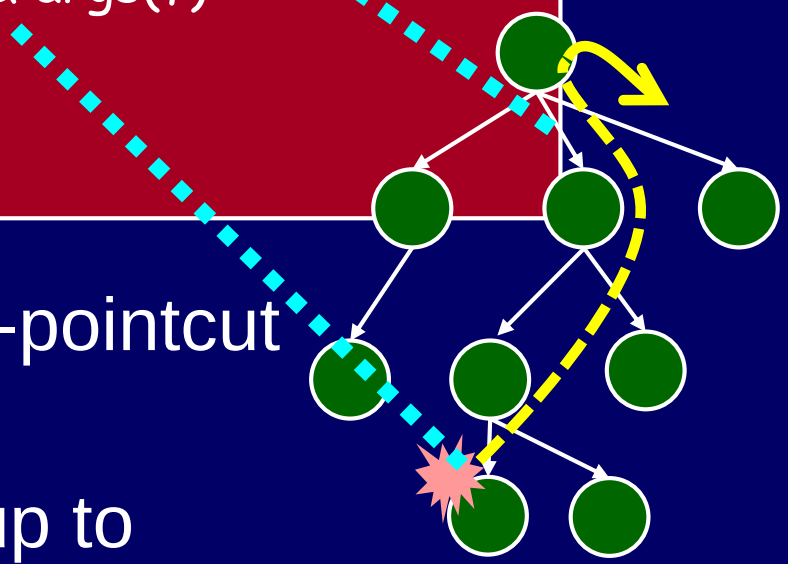
Proposal: Chop&Graft

- Use pointcut to specify the extent of cancellation
- Provide pseudo functions for resuming/retrying computations



Chop&Graft by Example: ***chop***

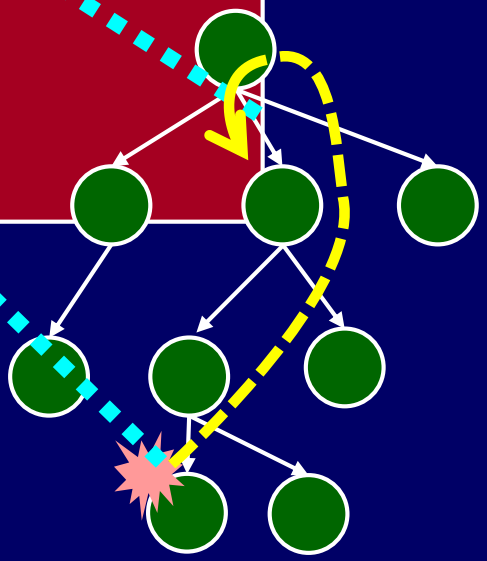
```
Plugin around(File f): chop(call(* *.loadPlugin(File)))
    && call(* *.readFile(File)) && args(f)
    && if(!f.exists()) {
  return null;
}
```



- is a pointcut taking a sub-pointcut (cf. cflow)
- terminates computation up to the matching jp
  - Advice body = a recovery process

Chop&Graft by Example: ***retry***

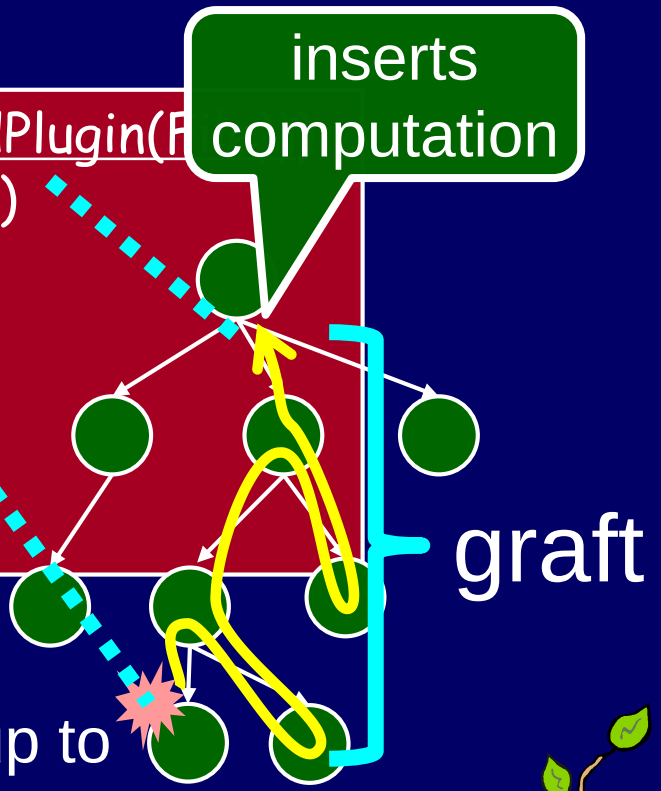
```
Plugin around(File f): chop(call(* *.loadPlugin(File)))
    && call(* *.readFile(File)) && args(f)
    && if(!f.exists()) {
  return retry();
}
```



- is a pseudo-function (cf. proceed)
- restarts the computation
  at the chop point
- can take alternative parameters

Chop&Graft by Example: ***graft***

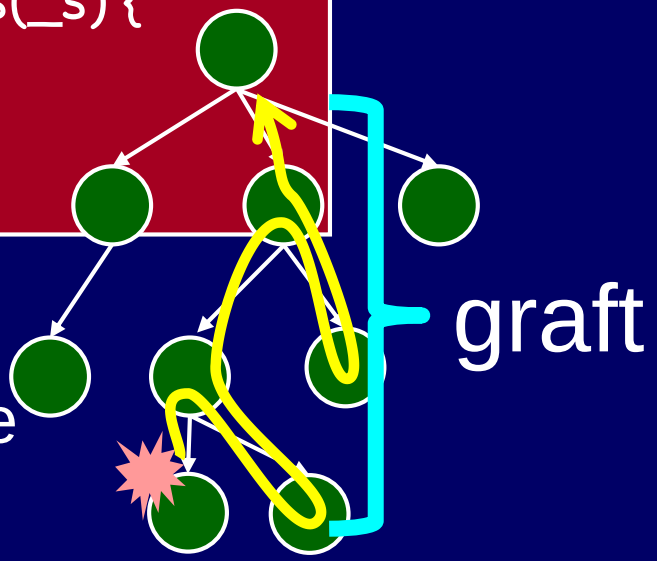
```
Plugin around(File f): chop(call(* *.loadPlugin(F
    && call(* *.readFile(File)) && args(f)
    && if(!f.exists()) {
  Plugin p = graft(DefaultCnf);
  p.setVerbose(false);
  return p;
}
```



- is a pseudo-function (cf. proceed)
- restarts the cancelled computation up to chop point
  - Advice can perform computation after graft

Chop&Graft by Example: backtracking

```
Plugin around(String s): chop(call(* *.loadPlugin(File)))
      && call(* URL.append(String)) && args(_s) {
    Plugin p = graft(proceed(s));
    if (p.isValid()) return p;
    return graft(proceed(".old"));  }
```



- Graft can be called more than once
  = going back to the middle of computation

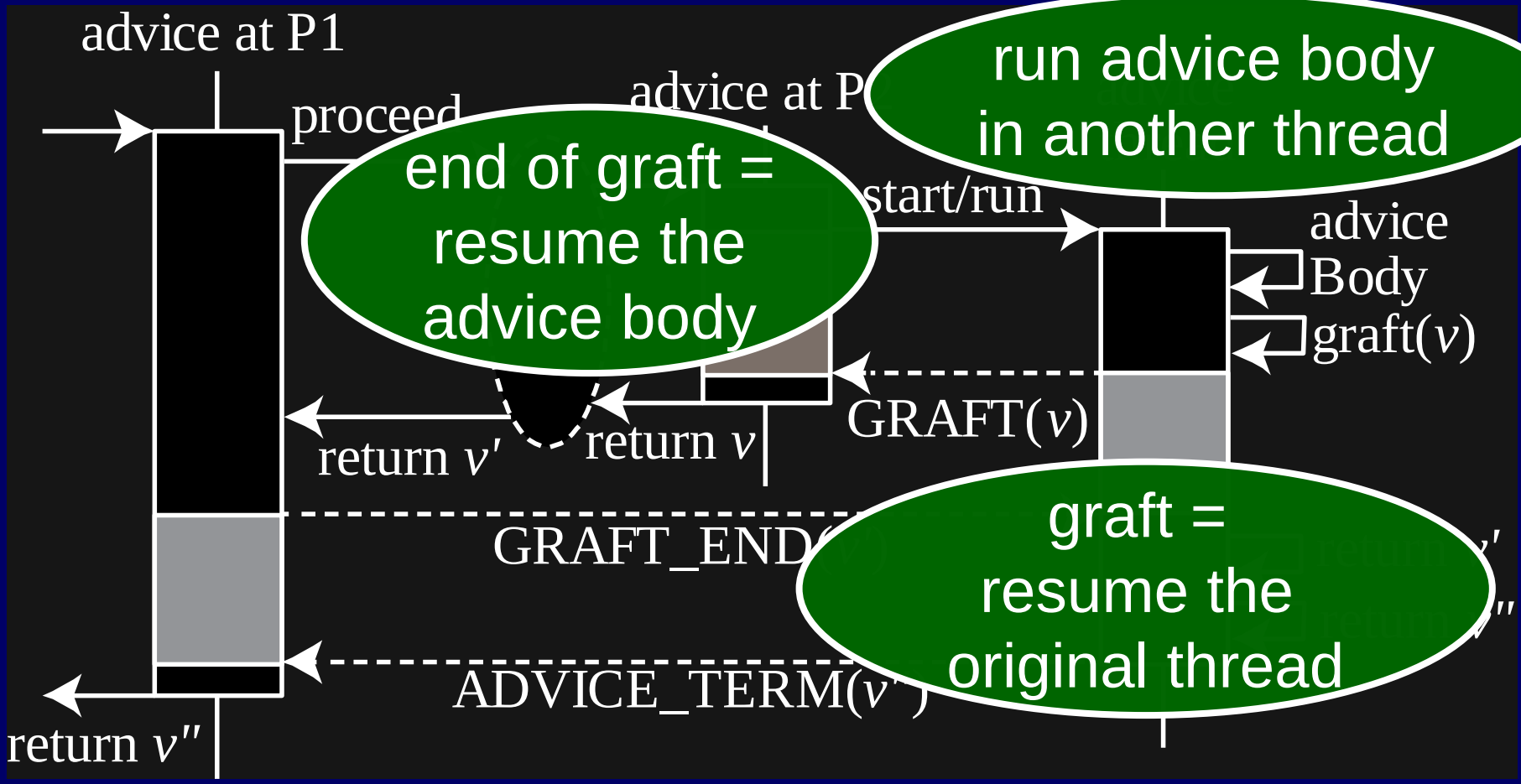
Implementations

1. By using delimited continuations
   [Felleisen'88, Danvy'90]
   - At chop point: (reset (proceed))
   - At advice point: (shift (lambda (graft) body)))

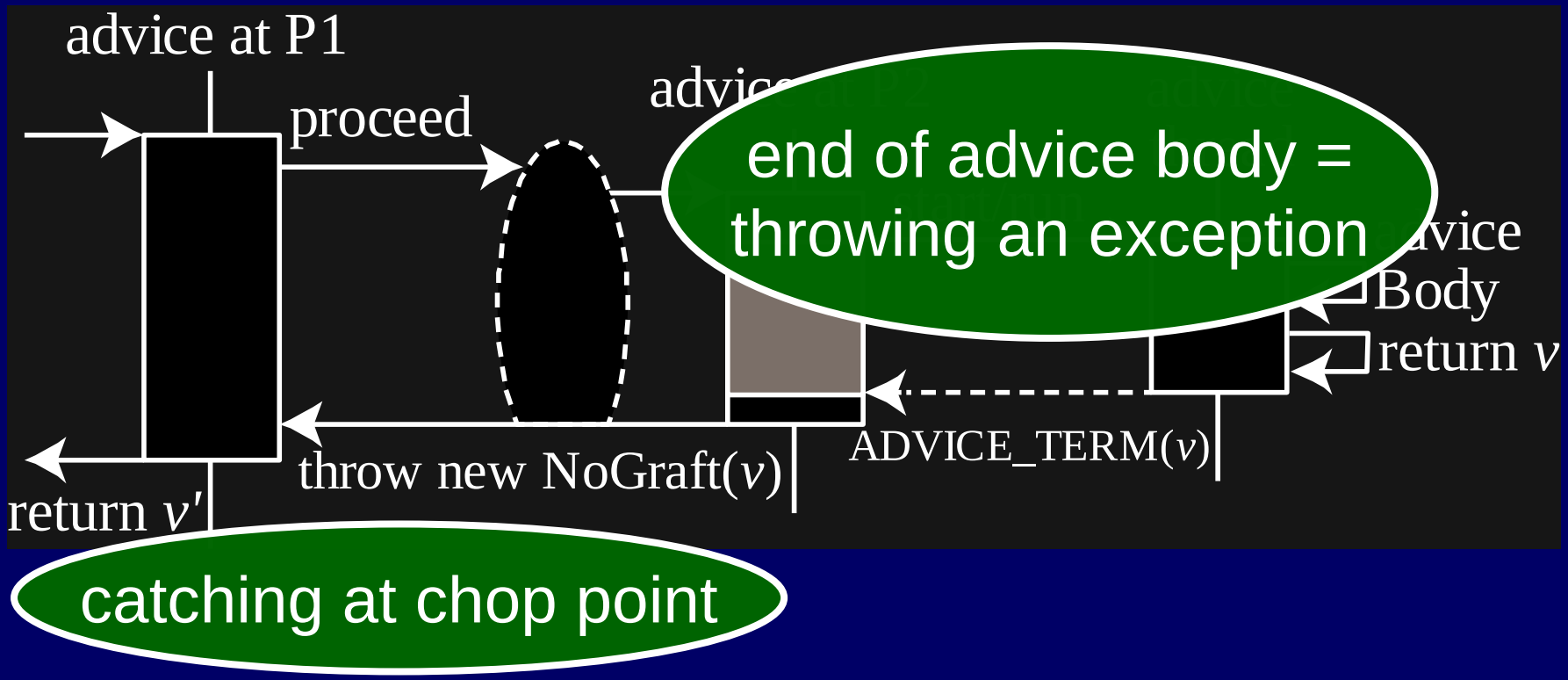
   Need multi-prompts for pairing

   implemented on AspectScheme + Racket
2. By using threads
   - Limitation: can call graft at most once
   - (for now) hand-compiling strategy

Implementation with threads (graft is called)



Implementation with threads (graft is ***not*** called)



Related work

- Aspects for exception handling
  [Lippert'00, Colyer'04, Filho'06, Taveira'09, Rebêlo'10]:
  --- only one side of catching/throwing
- EJFlow [Cacho'08]
  --- more precise exception catching
- Loop, closure, region jps/pointcuts
  [Harbulot'06, Bodden'11, Akai'09] --- for "local" flow control
- Delimited continuations [Felleisen'88, Danvy'90]
  --- for non-local flow control;
  low-level and based on names

Final Remarks

- Proposed Chop&Graft mechanism

  Pointcuts are powerful abstraction to express remote points of control

- Future work: precise semantics; full implementation; empirical studies