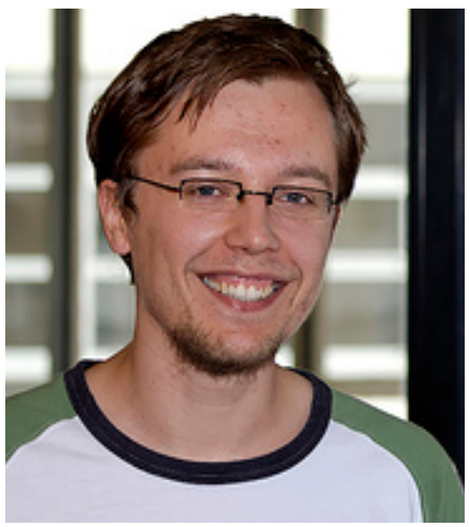
On the relationship between weavers and the build system

Bram Adams
GH-SEL, Ghent University

Kris De Schutter
PROG, VUB



Everything has a Makefile!

How do you bring AOP into your build system?

How to bring AOP in your build system?

- Why a potential problem/challenge?
  - source code modularity vs. build system dependencies
  - existing build system
  - weaver acceleration using build level-tricks

- But: Is there really a problem?

1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion

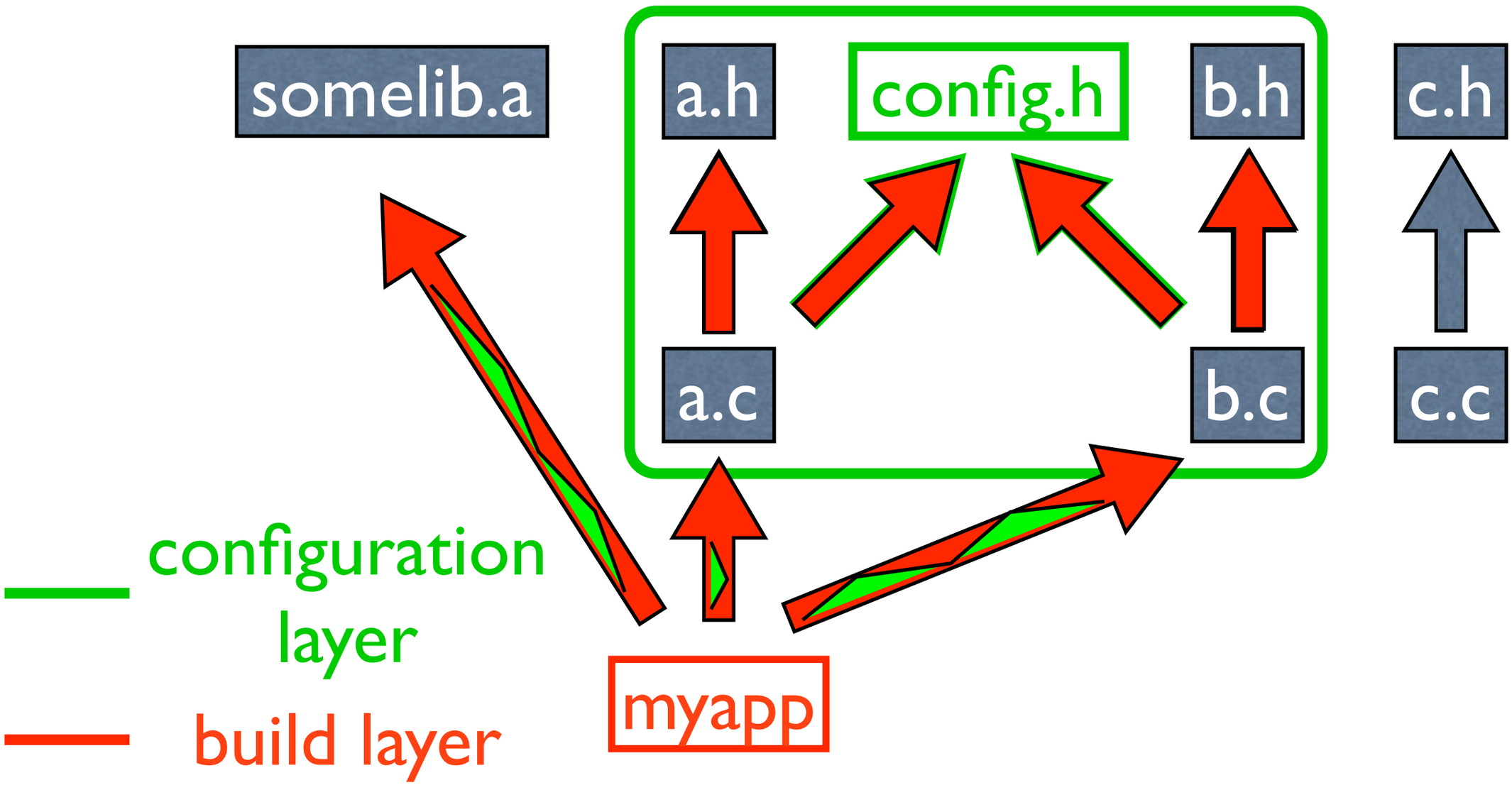
33 surveyed languages

prepro-cessing	Cobble, AspectC, AspectC++, XWeaver, Aspicere, C4, WeaveC, ACC, CaesarJ, Apostle, AHEAD
compile-time	AspectJ, abc, Hyper/J, AspectWerkz2, LogicAJ, Compose*, CARMA
link-time	Aspicere2
load-time	AspectJ, AspectWerkz2, Weave.NET
run-time	AspectC++, μDiner, TinyC, Arachne, TOSKANA, KLASY, TOSKANA-VM, Steamloom, AspectWerkz2, PROSE, Wool, JAC, Handi-Wrap, AspectS, AOP/ST, CARMA

Comparison of AOP approaches

- discover prominent build system issues:
  - effort
  - risks
  - workarounds
- derive lessons learnt for stakeholders
- distill requirements for AOP-aware build system?

Build System ⇒ 5 Issues



a. Resolve Platform-dependencies
b. Module Configuration

c. Build Integration
d. Build Order
e. Incremental Compilation

1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion

a. Resolving platform-dependencies

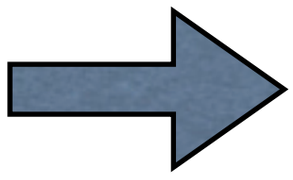
- Configuration layer parameterises:
  - source code:
    - base
    - aspects
  - build layer

abstraction of selected features, compiler, platform, ...

- no real problem ↔ do better than now?

Platform Dependency Issues

<table>
<tr><th></th><th>prepro-cessing</th><th>compile-time</th><th>link-time</th><th>run-time</th><th>load-time</th></tr>
<tr><td>language</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="2">tool</td><td colspan="3">conditional compilation
AOP in build
build tool support</td><td></td><td></td></tr>
<tr><td colspan="4">library management</td></tr>
<tr><td>user</td><td colspan="3">versioning problems
release management</td><td></td></tr>
</table>



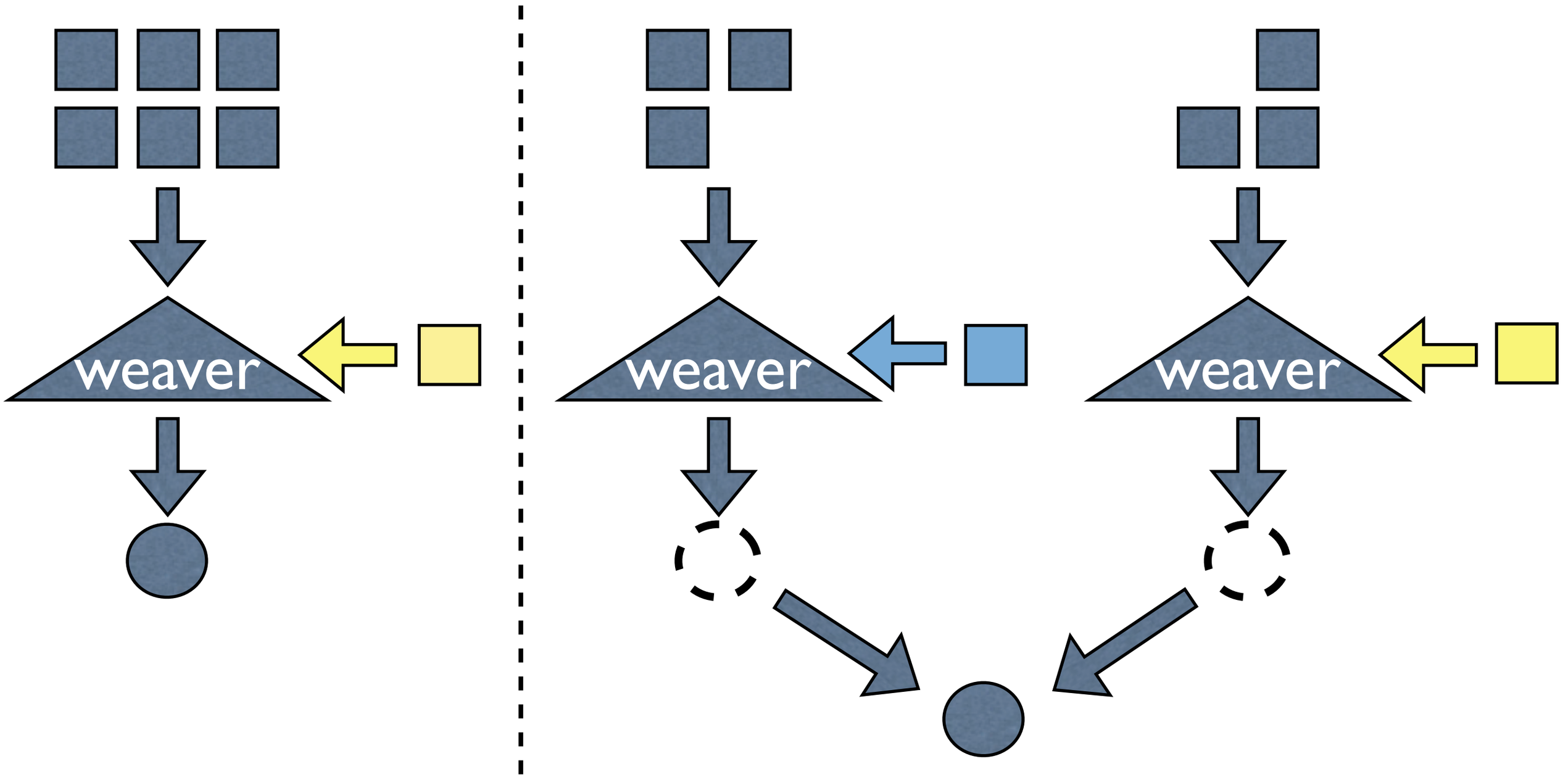
versioned modules/libraries?
other language support?

1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion

b. Module Configuration

- Configuration layer selects:
  - which aspects?
  - onto which base modules are aspects applied?
    - explicit mapping
    - implicit mapping (weaving time, ...)
- aspect pluggability ↔ implicit dependencies
- fine-grained control ↔ keep it manageable

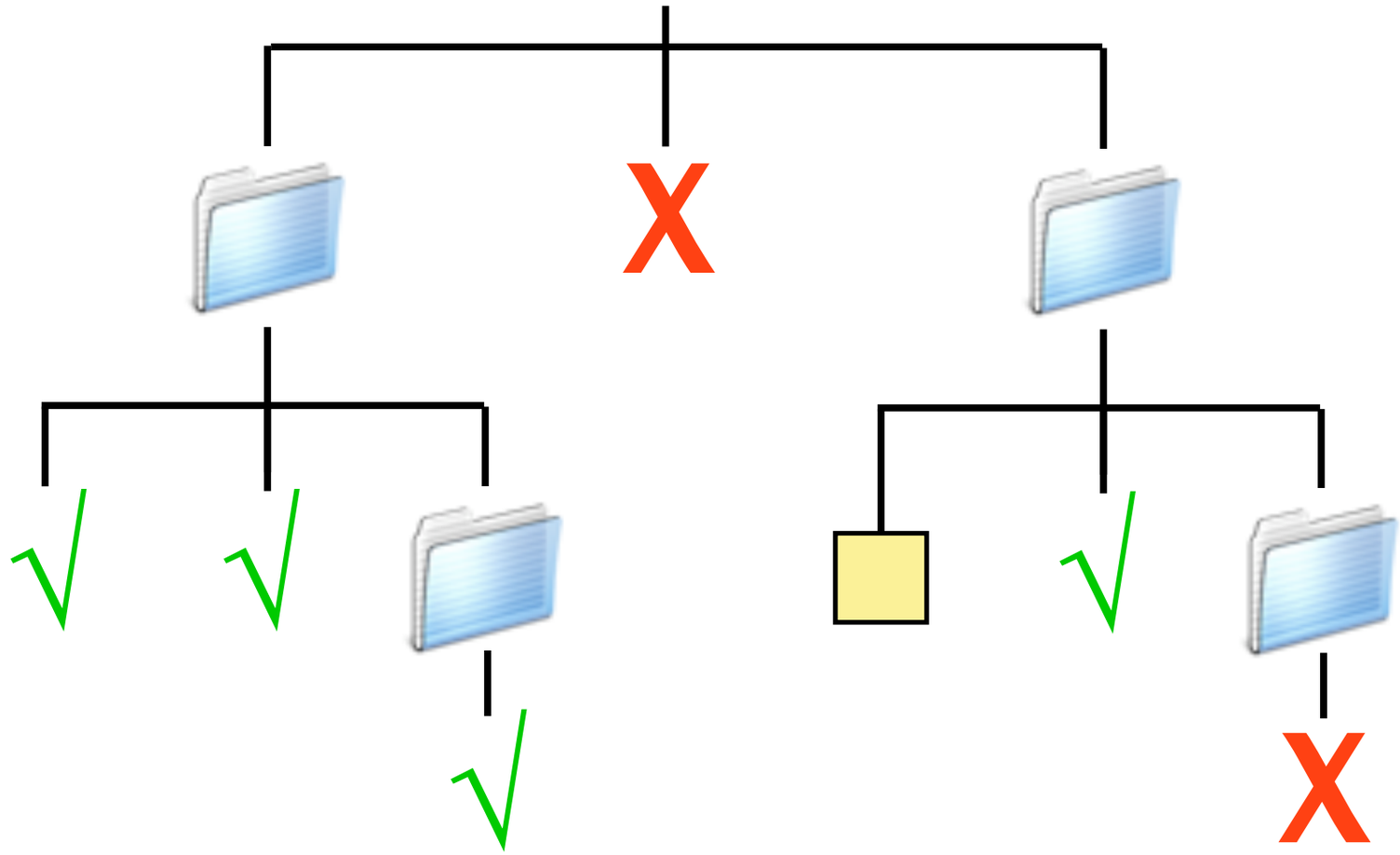
System-wide Configuration vs. Build Decomposition



Conventions

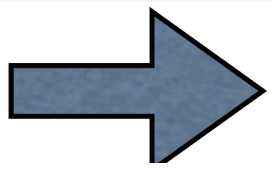
- ACC:
  - which aspects? local ↔ global
  - scope?

current-and-sibling rule



Module Configuration Issues

	prepro-cessing	compile-time	link-time	load-time	run-time
language	no control programmatic				programmatic
	application-wide textual list				
tool	easily enforceable command line switch		weaver complexity		deployment tools
	decompose build conditional compilation				cond. comp.
user		conventions			
	product line				



how to manage dependencies?

1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion

c. Build integration

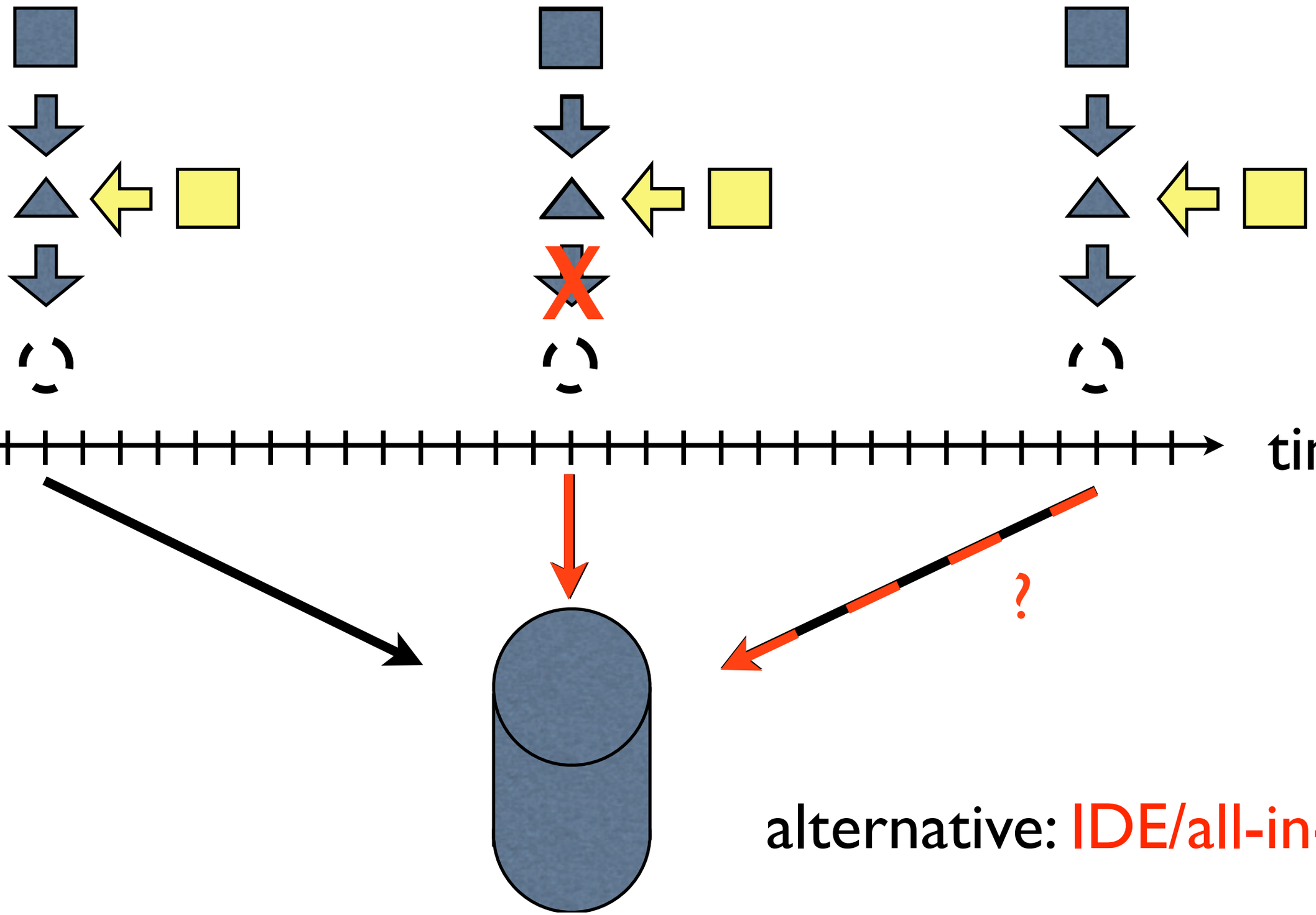
- pre-AOP:
  - file-level composition

current build systems

- AOP:
  - subfile-level, dynamic composition
  - whole-program view needed

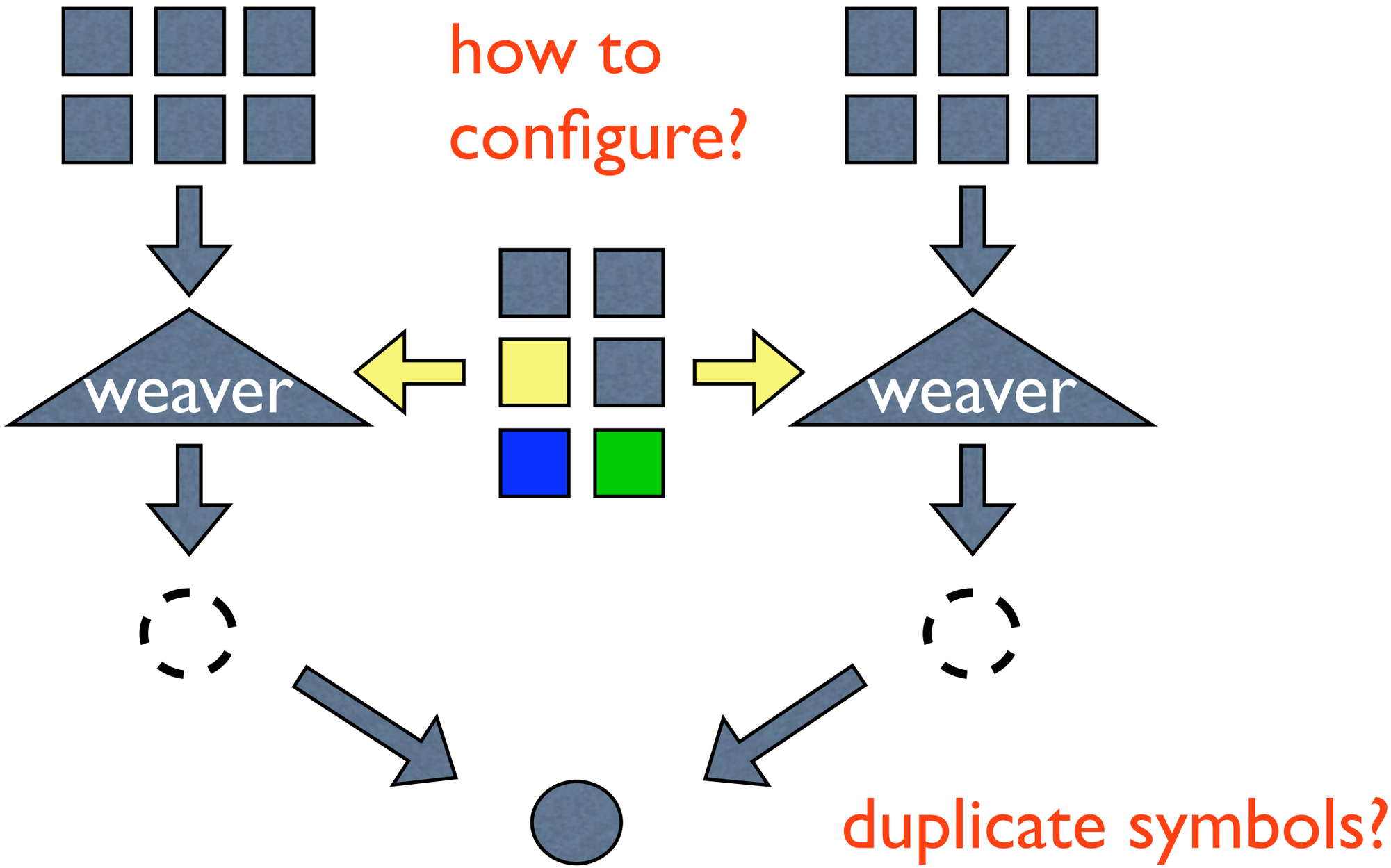
AOP-aware build systems

Repository

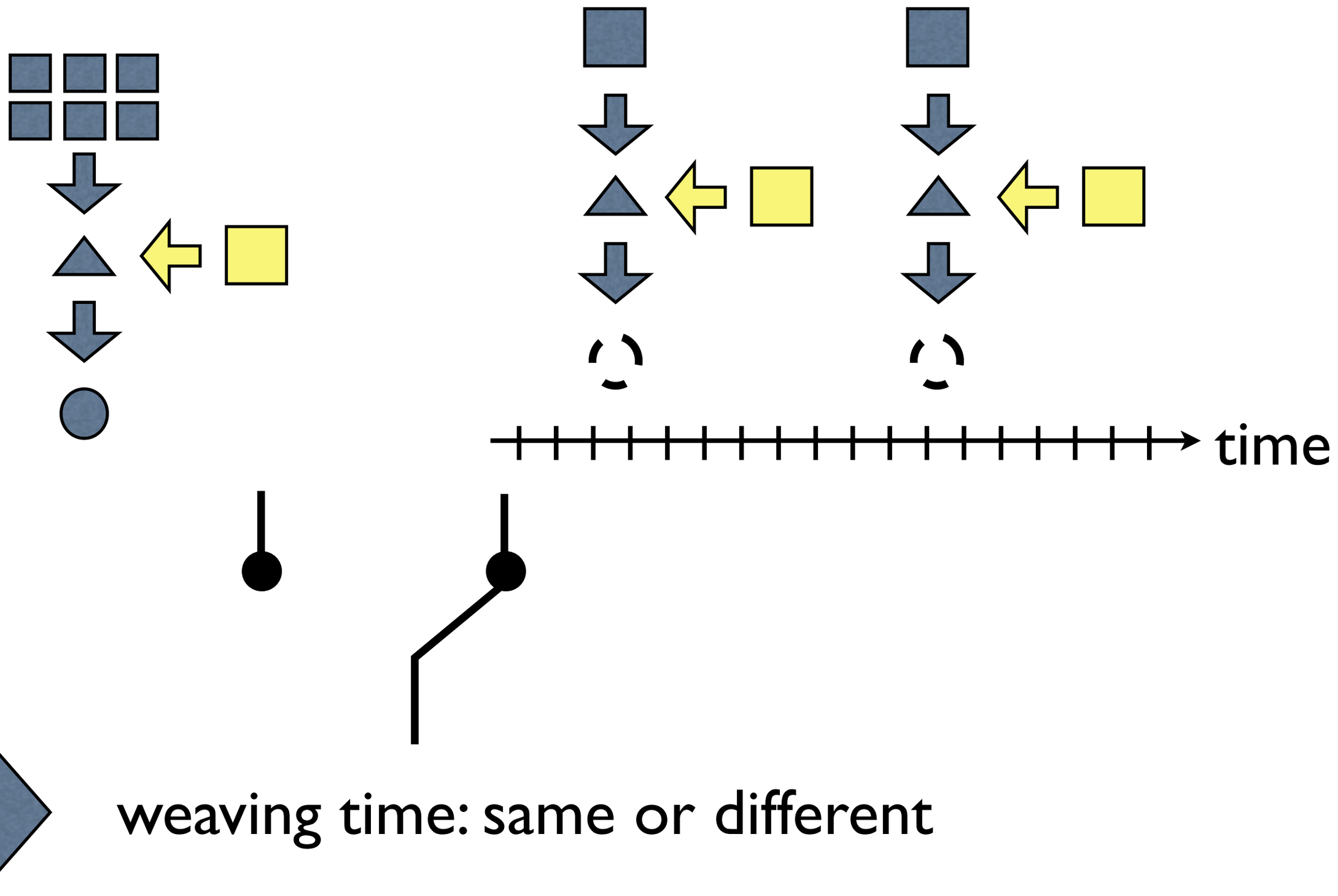


alternative: IDE/all-in-one

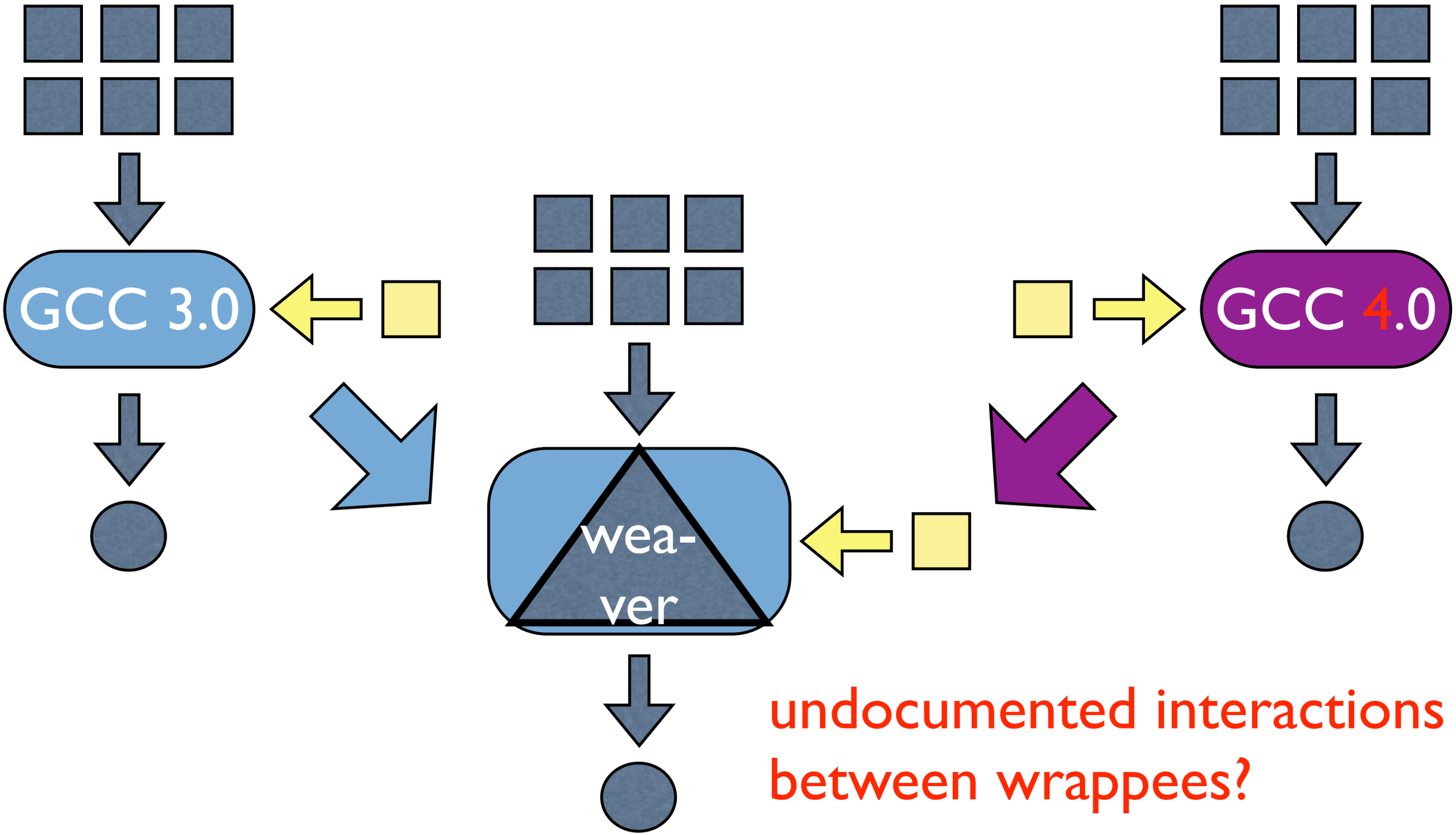
Support code



Switching between 2 or more weavers



Wrappers



undocumented interactions between wrappees?

Modes

- AspectC++:
  - whole-program ↔ single-translation unit
- ACC:
  - batch ↔ sequential weaving

	simple	complex
global	accmake	tacc (aspects in base directory)
		manually
local	manually	

	preprocessing	compiling	link-time	load-time	run-time
language					context?
tool	require all input repository weaver intelligence weaver modes		whole-program view (in theory)		
	all-in-one build/IDE MAKAO				
user	technical integration wrappers				
	preprocessing				idem
	intermediate files				
	library/support code				
	switch weavers test woven code				

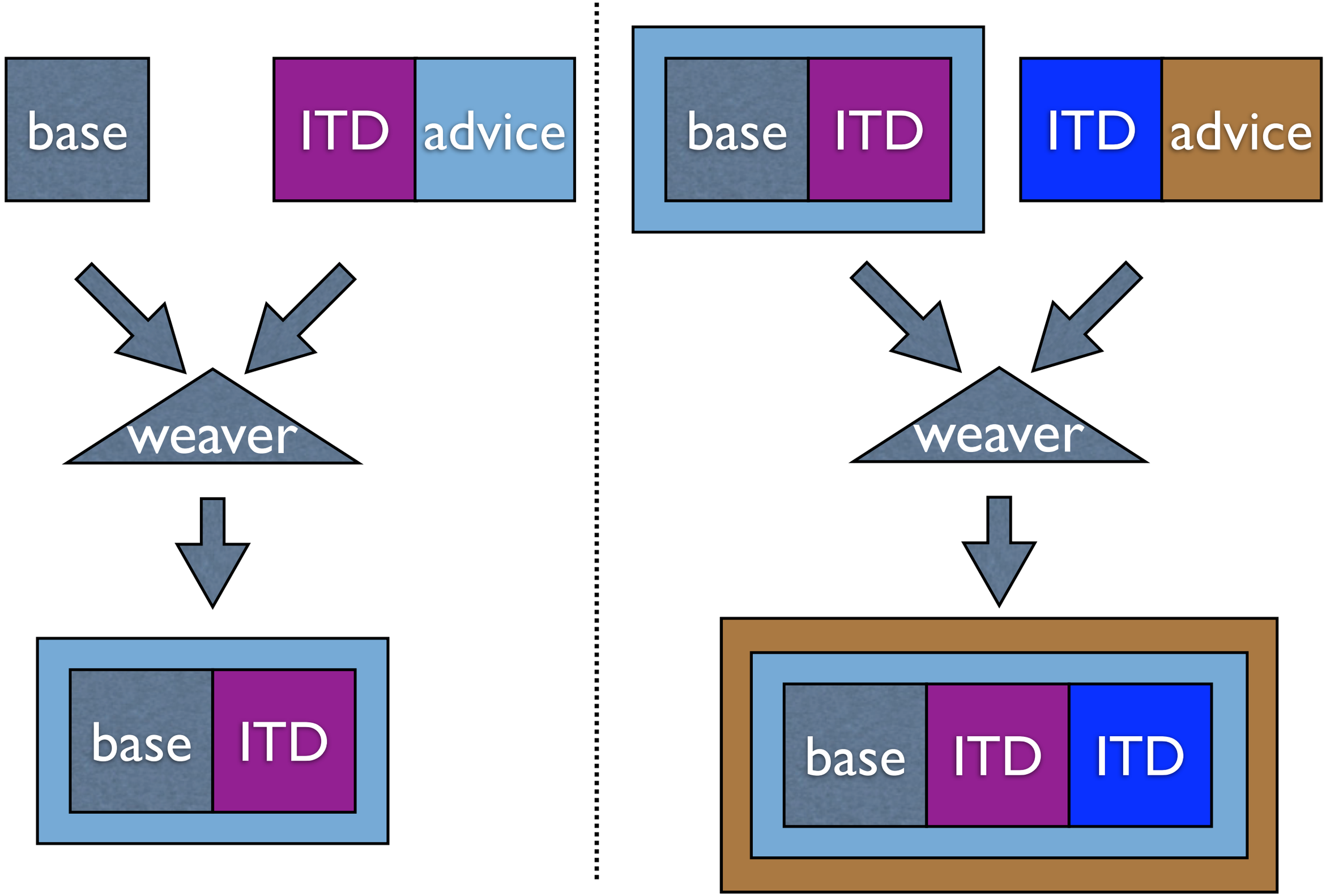
1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion

d. Build order

- pre-AOP:
  - separate compilation (Modula)
  - independent compilation (C/C++/Java)

- AOP: order of aspects
  - explicit control?
  - weaver implementation-dependent?

Bounded quantification



Build Order

	prepro-cessing	compile-time	link-time	load-time	run-time
language	lexical and configuration-based conventions language control (precedence)				programmatic
tool	CT's (base code-independent) GROOVE (whole-program view needed) feature map				
	bounded quantification				
				class loader	
user	build configuration flow				

1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion

e. Incremental Compilation

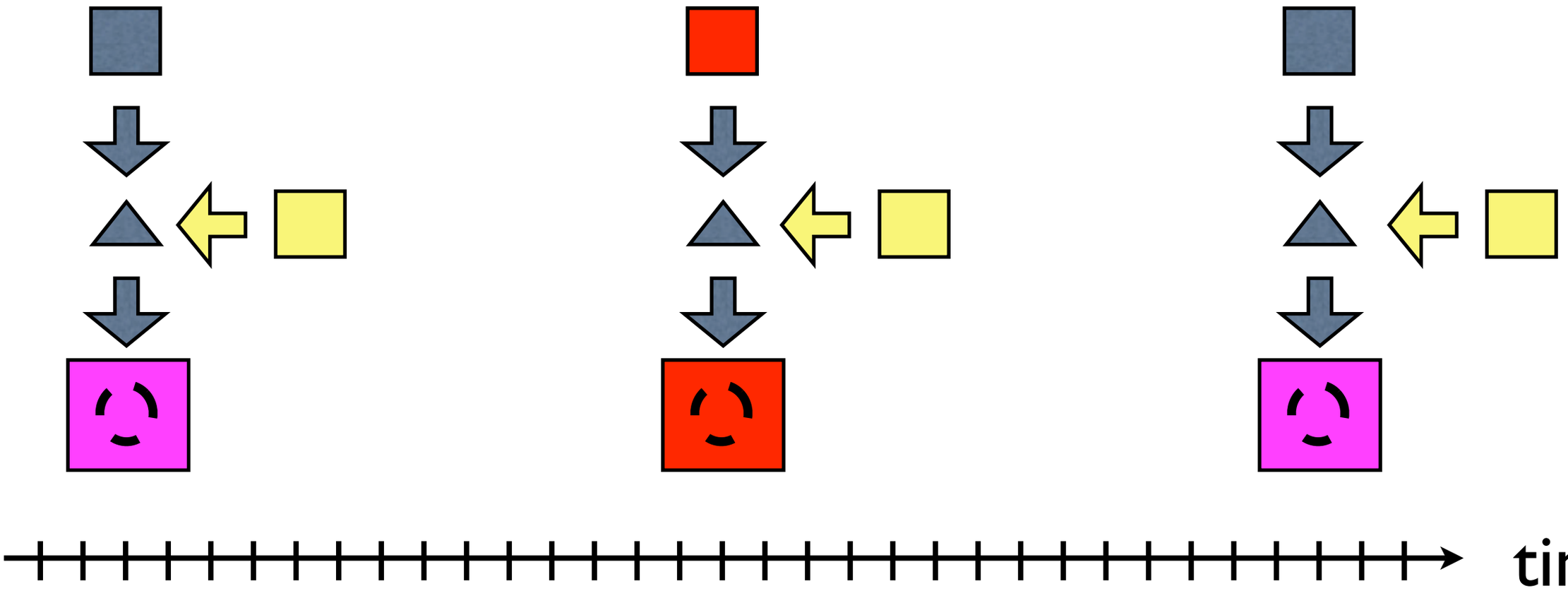
	X build time
WeaveC	2 (→1.3)
Aspicere1	70
C4	4 (→2)
AspectJ	“4”
abc	8
Compose*	→<2

- Problems:
  - no weaver from scratch
  - whole-program reasoning
  - traditional incremental compilation not applicable

Aspect configuration

- limit scope of aspects in build configuration:
  - extra/missing matches
  - implicit dependencies?
  - weaver-dependent
- partition base code (e.g. AspectJ)
  - according to non-interacting aspects
  - weave into binary form

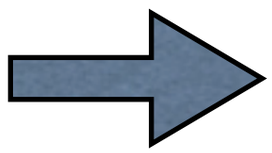
Caching



- source-to-source weavers (AspectC++)
- compile-time (AspectJ)

Incremental Compilation

	prepro-cessing	compile-time	link-time	load-time	run-time
language					
tool	caching			by design	
	explicit weaver support				
user	aspect configuration				
	partitioning system				



limit expressivity aspect language?

1. Problem Statement
2. Approach
3. Issues:
   a) Platform Dependencies
   b) Module Configuration
   c) Build Integration
   d) Build Order
   e) Incremental Compilation
4. Conclusion

Conclusion

- No structural solutions for build integration
- Many open questions... ⇒ opportunities for research :-)
- AOP-aware build system?

QUESTIONS?