MAKAO: Dealing with Legacy Build Systems

Bram ADAMS
Ghislain Hoffman Software Engineering Lab, INTEC, Ghent University
http://users.ugent.be/~badams

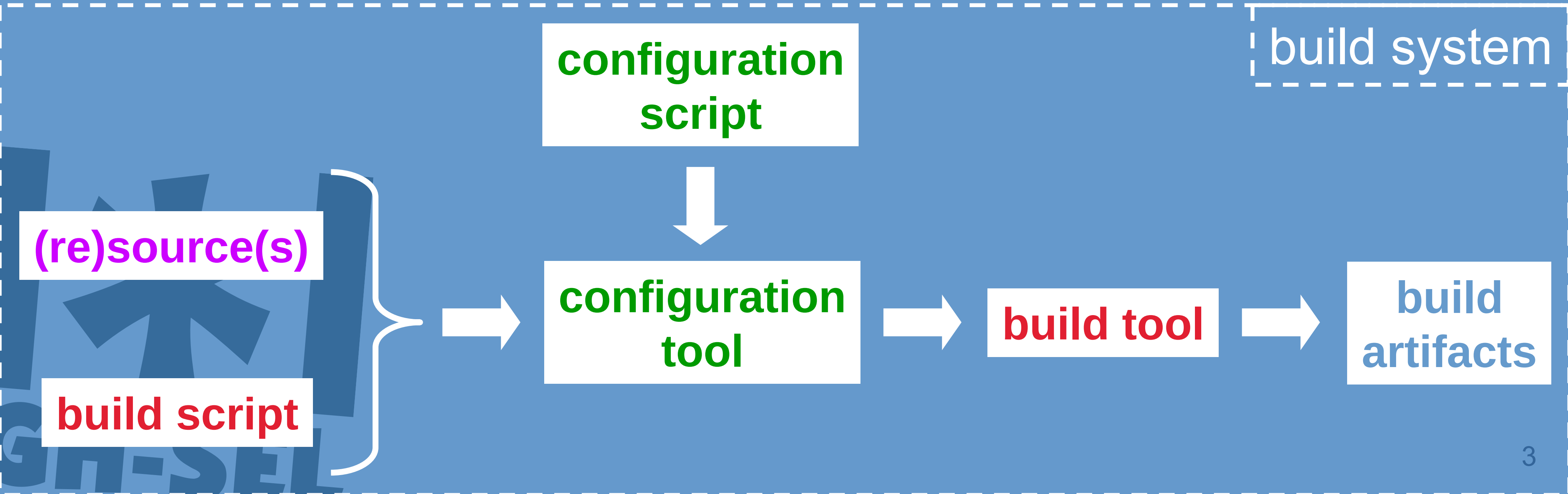
Outline

1. Build systems
2. Issues with legacy build systems
3. Conceptual solution
4. Make
5. GUESS
6. MAKAO
7. Issues revisited
8. Conclusion

1. Build systems

Some history:

- ...-1977: ad hoc build and install scripts
- 1977: make (Stuart Feldman), most influential build tool
- later:
  - various clones (GNU Make, ...) and alternatives
  - build configuration systems like imake and GBS



build system

configuration script

(re)source(s)

build script

configuration tool

build tool

build artifacts

2. Issues with legacy build systems (a)

Developers:

- “Why was this file not compiled?”
- “Where did the error originate?”
- “Where do I need to modify what makefile?”

Maintainers:

- “Why does this build take so long?”

KDE4:

Running "./configure; make; make install" seems easy enough, right? But the magic behind making those commands tick the right way is extremely difficult to master. ***Any developer, even of a relatively simple project, who has to setup the build system on their own will likely confirm this.***

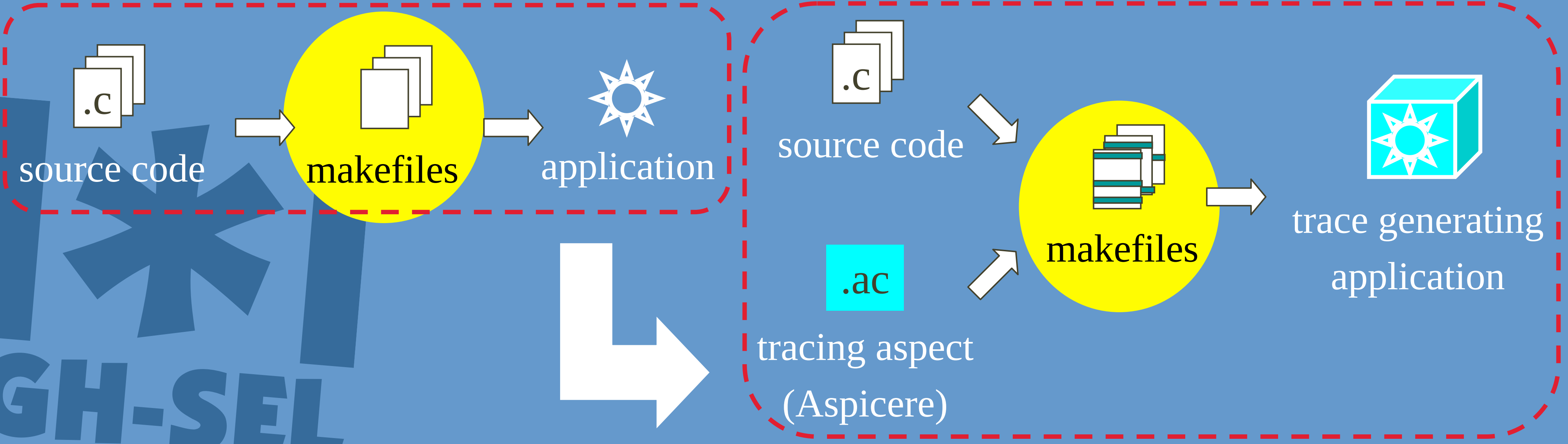
(http://lwn.net/Articles/188693/)

2. Issues with legacy build systems (b)

Case study using Aspicere:

- weaving tracing advice in industrial C code base
- weaver:
  - preprocesses base and advice code ...
  - ... and needs to link a generated file in each executable and library

⇒ how to integrate Aspicere into the build system?

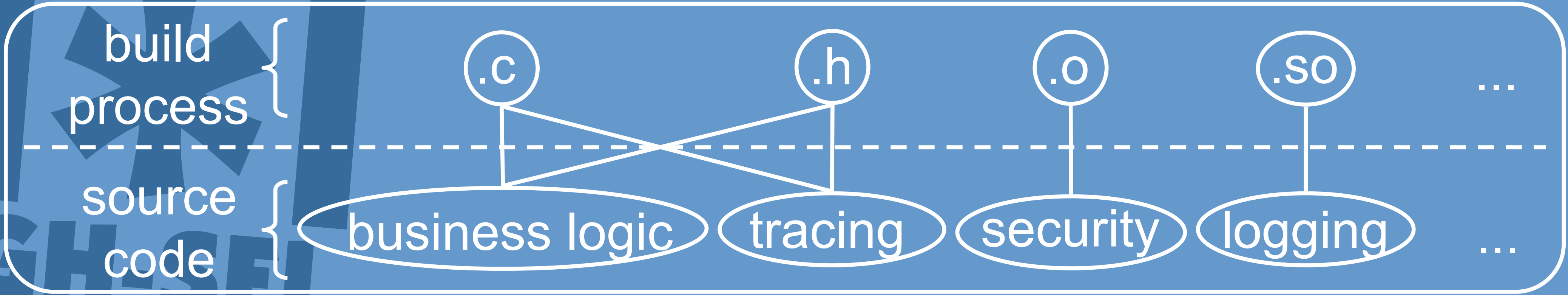


3. Conceptual solution

How can we solve these problems?

1. find suitable model for build process
2. build upon this model:
   - visualisation of flow and concerns
   - querying
   - modification
   - validation

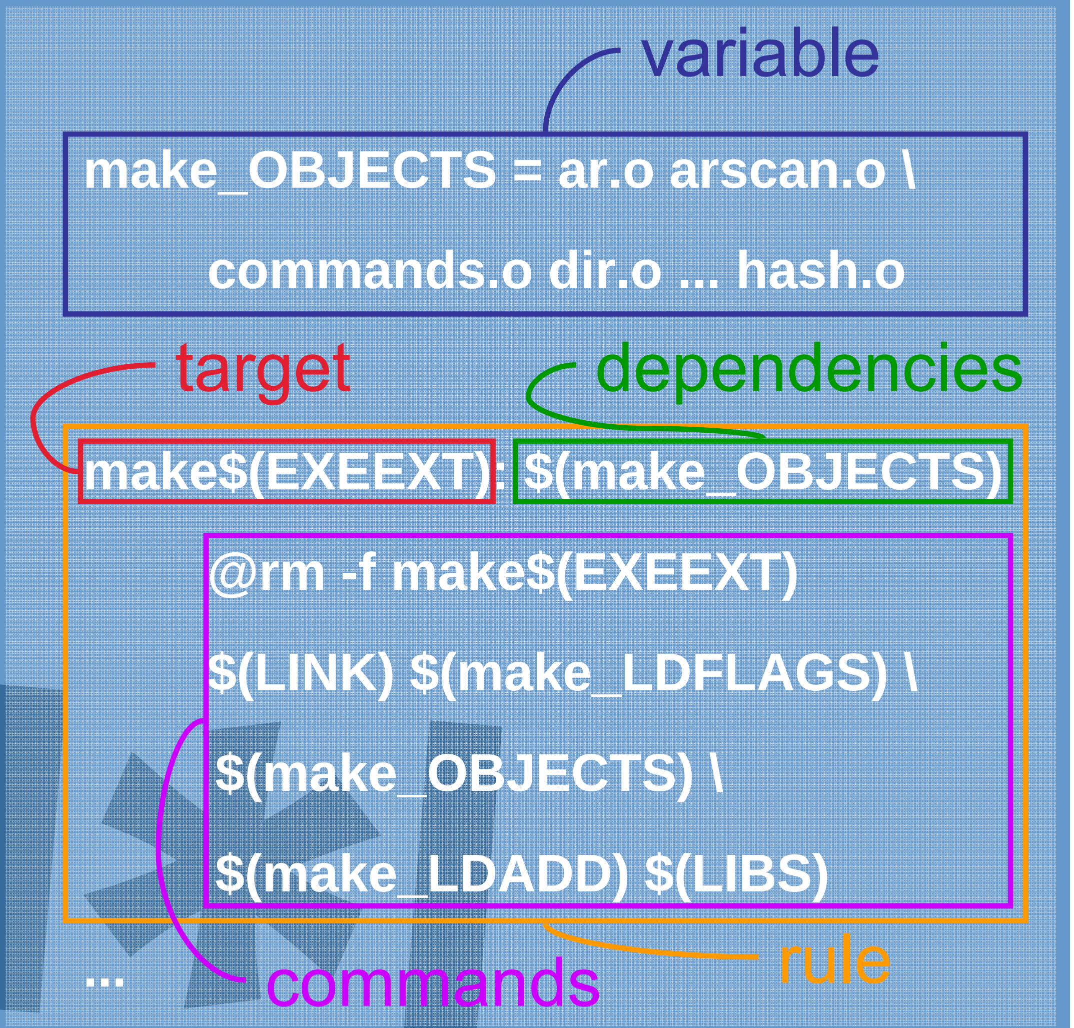
Can **AOP at makefile-level** help?



Feldman. “Make-a program for maintaining computer programs”. Software - Practice and Experience

4. Make

Makefile



variable

```
make_OBJECTS = ar.o arscan.o \
        commands.o dir.o ... hash.o
```

target

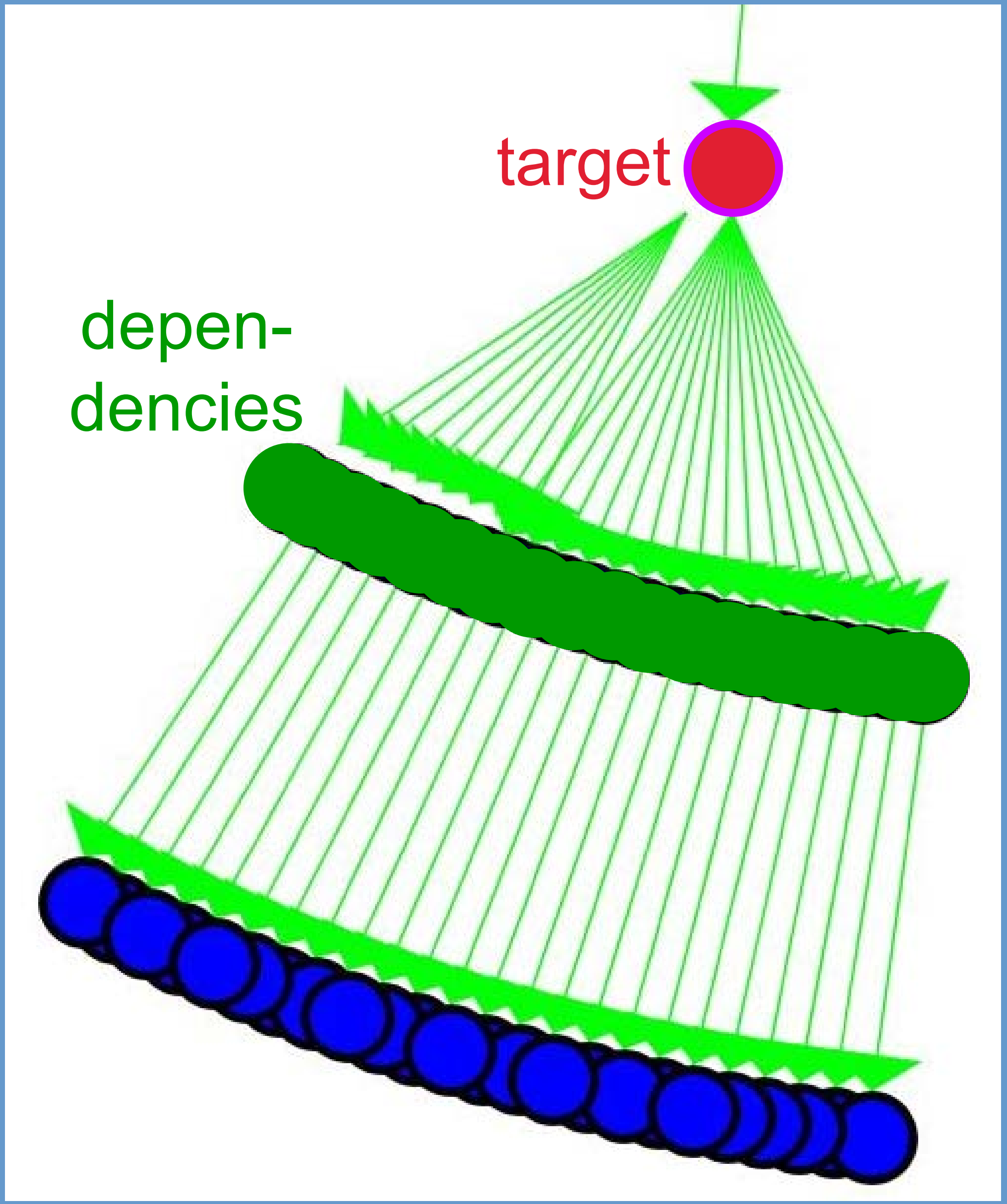
dependencies

```
make$(EXEEXT): $(make_OBJECTS)
        @rm -f make$(EXEEXT)
        $(LINK) $(make_LDFLAGS) \
        $(make_OBJECTS) \
        $(make_LDADD) $(LIBS)
...
```

commands

rule

Directed Acyclic Graph (DAG)

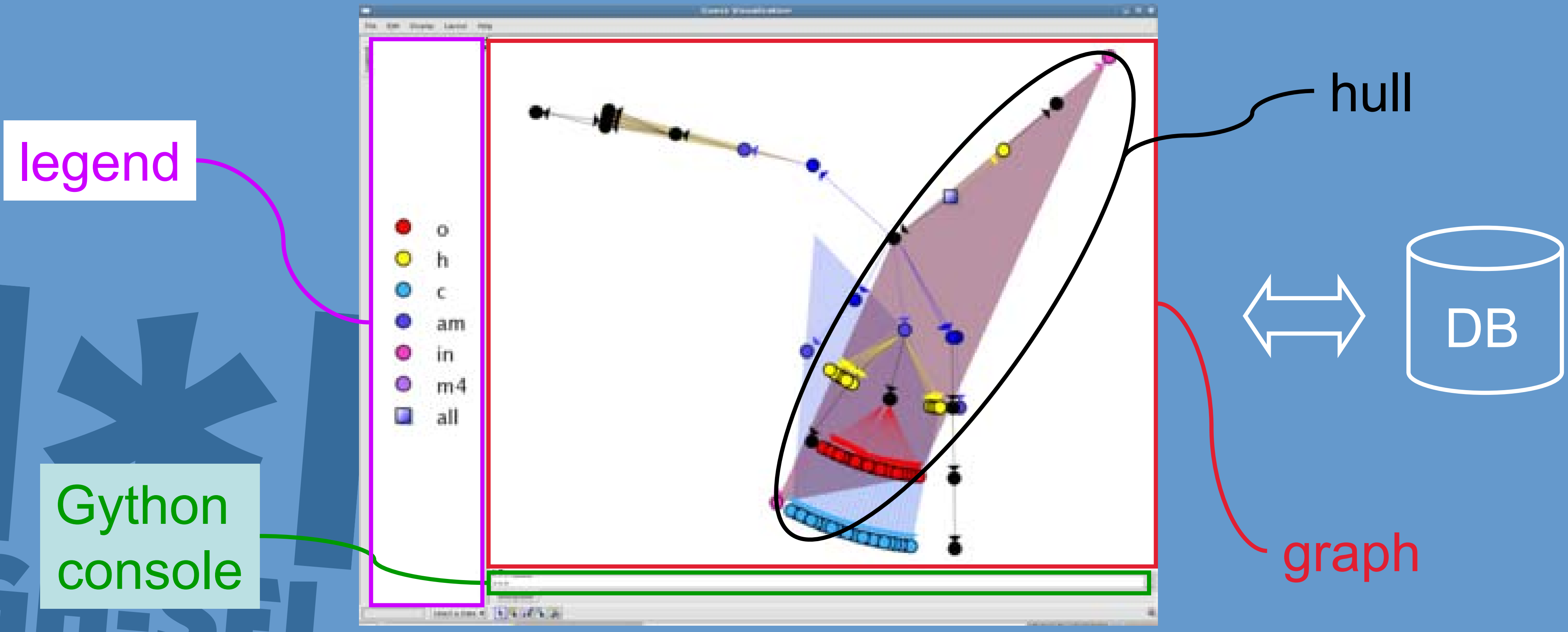


⇒ de facto build tool/process model!

5. GUESS

Graph Exploration System:

- graph analysis and visualisation
- embedded scripting language (Gython)
- database back-end



legend

hull

DB

Gython console

graph

6. MAKAO

Makefile Architecture Kernel for Aspect Orientation:

- re(verse)-engineering of build systems
- based on graph model of build [trace]
- built on top of GUESS:
  - get trace via “make -w --debug=v --debug=m -p”
  - dependency graph extracted to .gdf-file

Why?

AOP	***MAKAO***	***component***
join point	target or command	
pointcut	query	
advice	command modification	
weaving	propagation of changes to build and configuration scripts	

6. MAKAO: Explorer

Explorer:

- visualization of dependency graph:
  - **coloring** of targets based on “**build concern**”, i.e. extension (.o, .c, ...)
  - one **hull** around all targets of the same **makefile**
  - separate color per hull
- filtering of build concerns
- concern metadata like commands, line number and makefile, ...

→ Demo:

- exploring build process of GNU Make 3.81

6. MAKAO: Finder

Finder:

- query for targets (and commands) based on properties like:
  - specific concern
  - error message
  - commands
  - ancestor target’s properties

Problem	***Query***
all .o targets	(concern==“o”)
all targets depending on .c file	(node2.concern==“c”).node1
all source-processing commands for target T	[command for command in commands[T] for tool in ["CC","gcc","esql"] if command.find(tool)!=-1]

Gython

list comprehension

6. MAKAO: Adviser

Adviser:

- dynamically compose advice in Gython using:
  - queried targets and commands
  - existing variable definitions
  - dependency data

Example: Aspicere

1. Find all targets T depending on a .c-file (previous slide)
2. (comm,tool)=(only) source-processing command of target T (altered previous slide)
3. before-advice="\n".join(

```
        [comm.replace(tool,tool+” -E -o ${<}”),
         "aspicere -i ${<} -o ${<} -aspects aspects.lst"])
```

6. MAKAO: Weaver

Weaver:

- logically:
  - update graph with new edges
  - update advised targets' commands

  > impact analysis
- physically:
  - propagate modifications made in Adviser back to:
    - build scripts
    - configuration scripts

  ↔harder:
    - starting from one build trace
    - tracability from build script to configuration script?

7. Issues revisited

	Explorer	Finder	Adviser	Weaver
Developers:				
• "Why was this file not compiled?"	√	√		
• "Where did the error originate?"	√	√		
• "Where do I modify what makefile?"	√	√	[√	√]
Maintainers:				
• "Why does this build take so long?"	√	√		
Aspicere:				
• preprocesses base and advice code ...	√	√	√	√
• ... and needs to link a generated file in each executable and library	√	√	√	√

8. Conclusion

MAKAO:

- re(verse)-engineering of build process
- based on graph model
- built around flexible graph tool (GUESS)
- components:
  - Explorer
  - Finder
  - Adviser
  - Weaver

Explorer, Finder, Adviser } http://users.ugent.be/~badams/makao

Weaver } (currently) vaporware

Future work:

- Weaver, Validator, Simulator, ...
- apply MAKAO on case study (Aspicere, ...)

QuickTime™ en een
TIFF (ongecomprimeerd)-decompressor
zijn vereist om deze afbeelding weer te geven.

Thank you!