Comparing Software Architecture Recovery Techniques Using Accurate Dependencies

Thibaud Lutellier*, Devin Chollak*, Joshua Garcia‡,
Lin Tan*, Derek Rayside*, Nenad Medvidovic†, Robert Kroeger§

***Univ. of Waterloo,** ‡George Mason Univ.,†Univ. of Southern California, §Google







Software Architecture Is Important

- **Software architecture** is:
  - a high-level representation of the structure of a system.
  - crucial for:
    - Program understanding
    - Software maintenance
    - Programmers communication
- Documented architectures often are outdated.

Recovering Architecture From Source Code Is Important

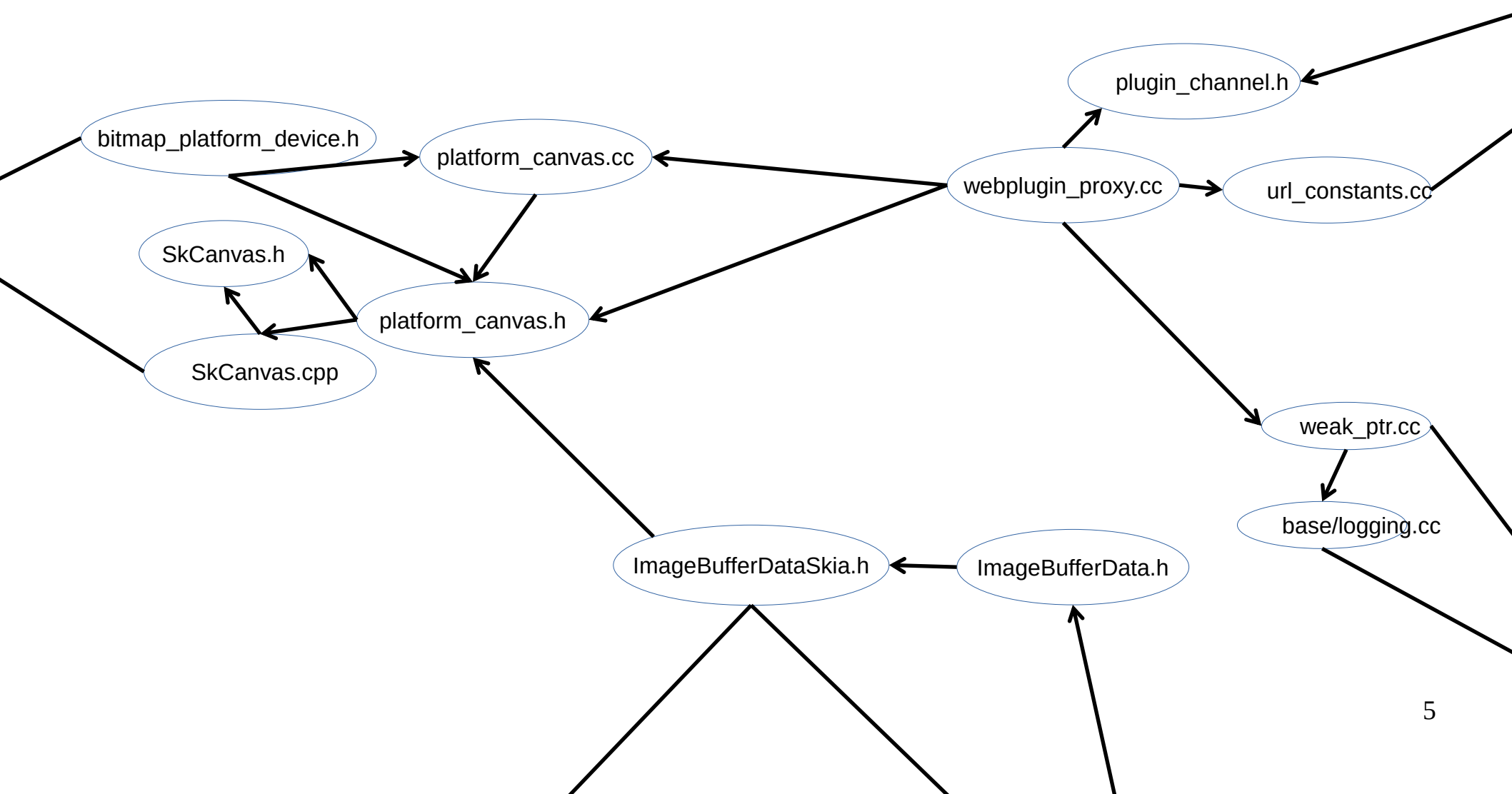
We need to recover the architecture of a program from the source code!

Automatic Architecture Recovery Is Needed

- Manual recovery is feasible but costly:
  - Recovering a medium-size project architecture takes **~100hours.** [Garcia et al., ICSE SEIP'13]
  - Deep knowledge of the project is necessary.
  - It can take **years** to manually recover the architecture of a large project!
- Many automatic recovery techniques exist.
  - They generally take a dependency graph as input.

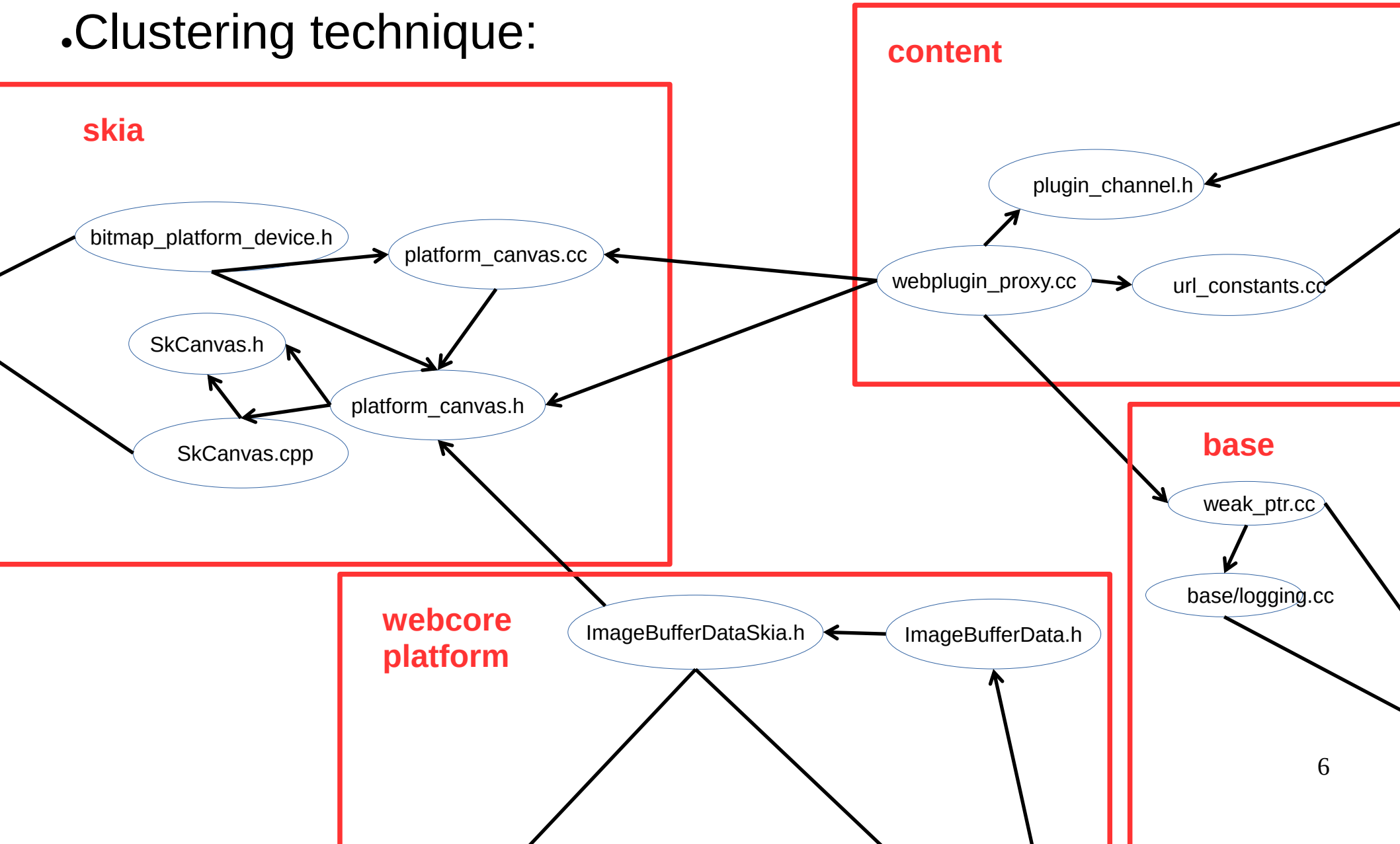
Example: Architecture Recovery

- Input: File dependency graph (Partial graph of Chromium)



Example: Architecture Recovery

- Clustering technique:



Motivation

- Previous work focuses on improving algorithms and heuristics. [Maqbool et al., CSMR'04; Andritsos et al.,EDBT'04,Garcia et al., ASE'11]
- They use **primitive and inaccurate** dependencies.
- More **accurate** dependencies based on **symbols** exist.
- It is unclear whether those techniques scale to 10MLOC.

Types of Dependencies

foo.c:

```
#include “foo.h”
#include “bar.h”
#include “unused.h”

void foo(){
        bar();
}
```

bar.h:

```
void bar();
```

bar.c:

```
#include “bar.h”
void bar(){
        do_something()};
```

unused.h:

```
void executeCommand(*Command);
```

Include Dependencies

foo.c:

```
#include “foo.h”
#include “bar.h”
#include “unused.h”

void foo(){
        bar();
}
```

bar.h:

```
void bar();
```

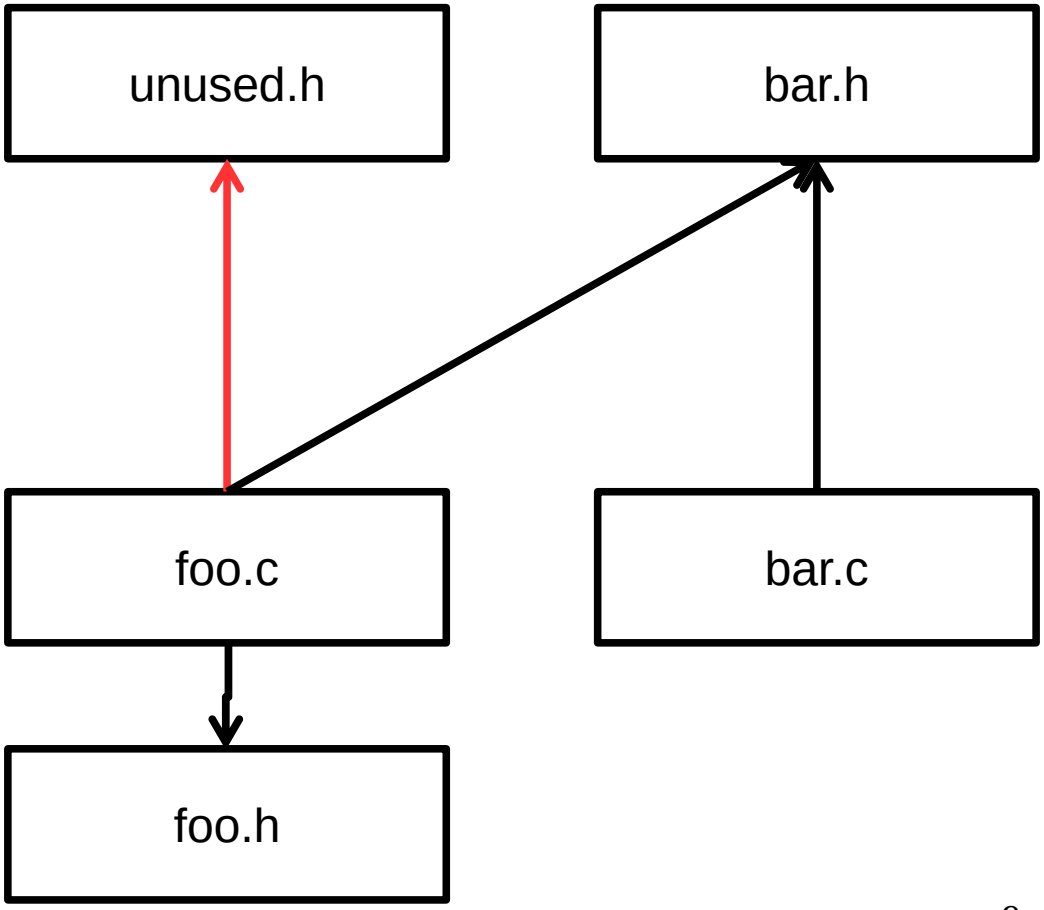
bar.c:

```
#include “bar.h”
void bar(){
        do_something()};
```

unused.h: ← Not used in foo.c!

```
void executeCommand(*Command);
```

Dependency graph based on include dep



Symbol Dependencies

foo.c:

```
#include “foo.h”
#include “bar.h”
#include “unused.h”

void foo(){
        bar();
}
```

bar.h:

```
void bar();
```

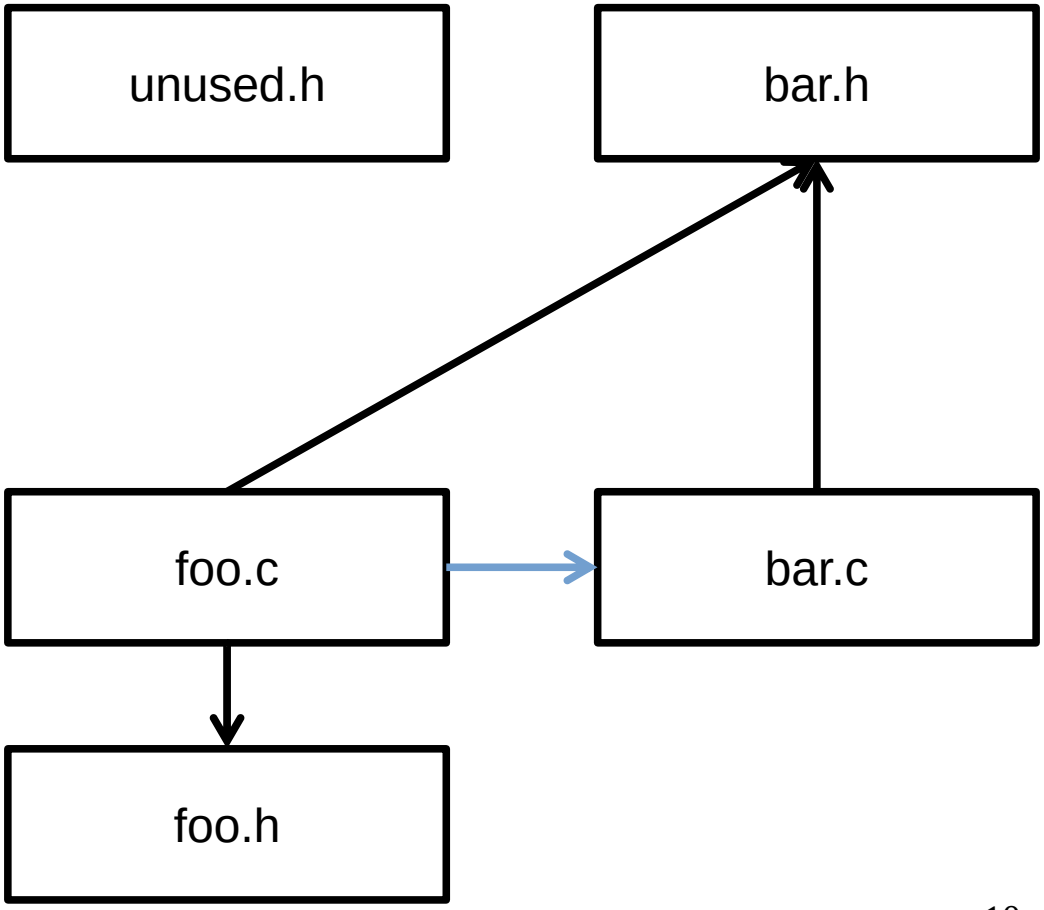
bar.c:

```
#include “bar.h”
void bar(){
        do_something()};
```

unused.h:

```
void executeCommand(*Command);
```

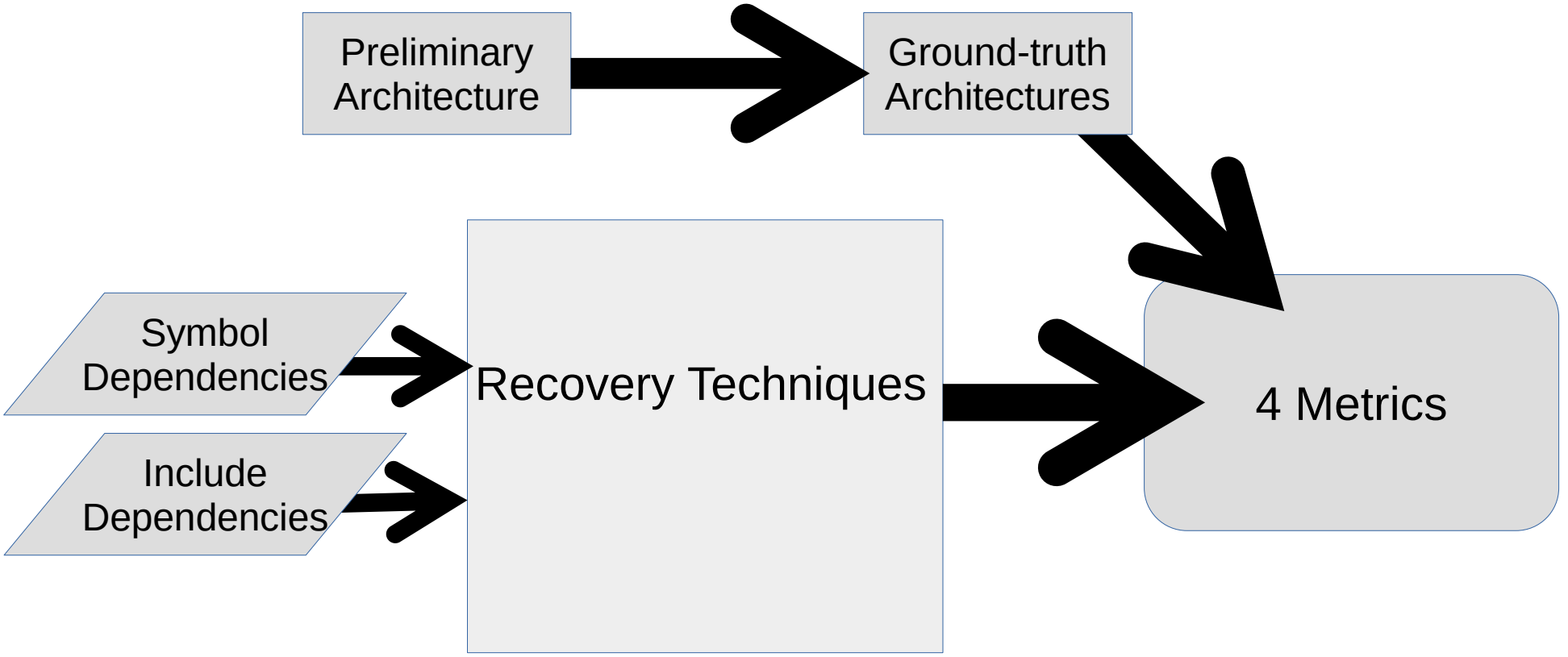
Dependency graph based on symbol dep



Contributions

- We compared 9 architecture recovery techniques:
  - **The accuracy of dependencies is a factor to consider** for high recovery accuracy.
  - Some techniques scale to large projects and some do not (ZBR, LIMBO, Bunch-SAHC).
- We recovered the ground truth of **Chromium**:
  - This is **the largest ground truth** recovered to date.
- We proposed a new groud-truth recovery technique.

Our Approach



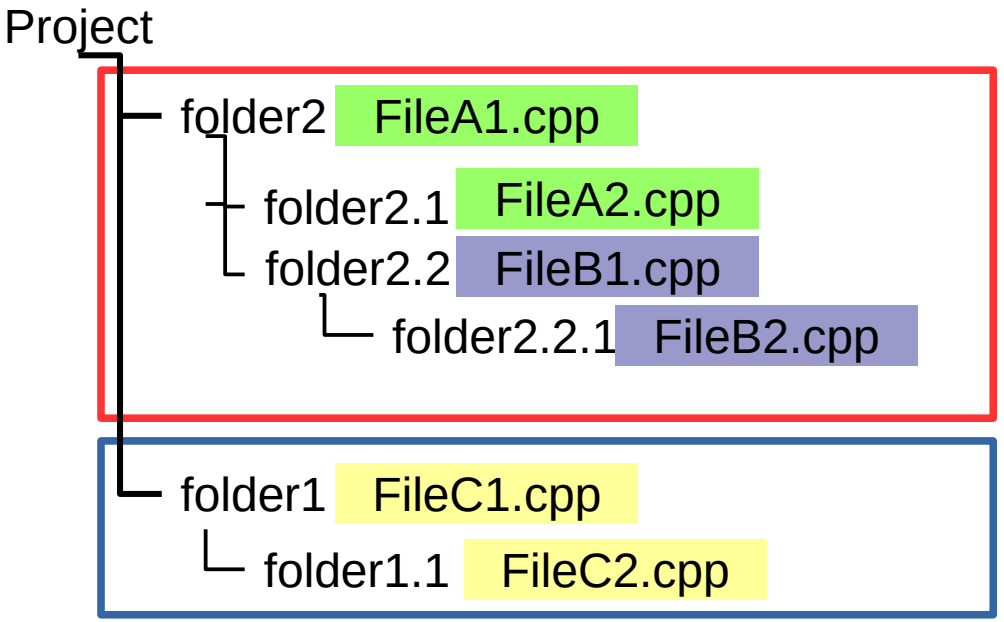
Recovering a Ground Truth

- Manually extracting a ground truth is a **tedious** task.
- People typically start with an automatic approach.
- We presented automatically recovered architectures to Chromium developers.
  - They found them difficult to interpret.
  - They prefered to use modules instead of files.

Submodule-Based Approach

- Large software generally have **modules** defined.
- We cluster the files based on modules described in configuration files.
- We merge modules based on the directory tree.
- Developers found it more relevant to use as a preliminary architecture.

Submodule Approach: Example



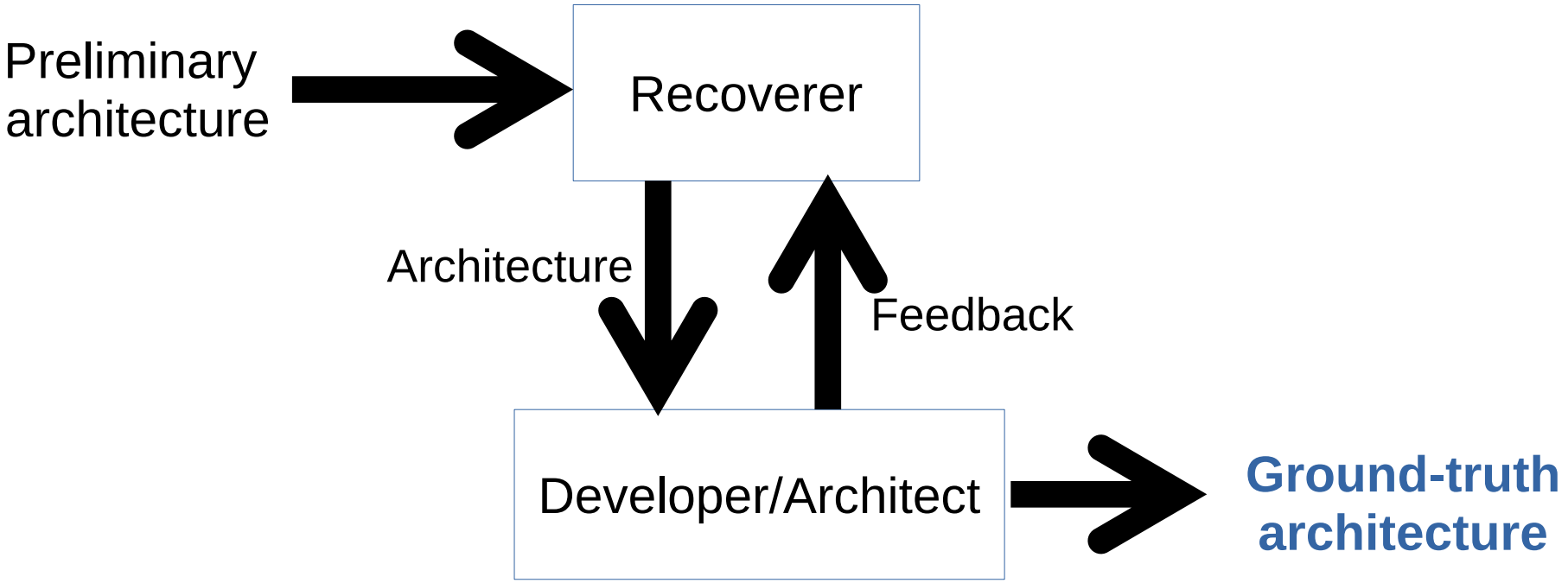
Module A

Module B

Module C

Cluster 1:	All files of Module B in a subdirectory of Module A files
Cluster 2:	All files of Module C in a different directory

Obtaining the Ground Truth



It took **2 years of discussion** with Chromium developers to obtain a ground truth.

Research Questions

Can more accurate dependencies improve the accuracy of existing architecture recovery techniques?

Can existing architecture recovery techniques scale to large projects comprising 10MSLOC or more?

Research Questions

Can more accurate dependencies improve the accuracy of existing architecture recovery techniques?

Can existing architecture recovery techniques scale to large projects comprising 10MSLOC or more?

Projects Evaluated

Project	Version	Description	SLOC	File
Chromium	svn-171054	Web Browser	9.7M	18,698
ITK	4.5.2	Image Segmentation Toolkit	1M	7,310
Bash	4.2	Unix Shell	115K	373
Hadoop	0.19.0	Data Processing	87K	591
ArchStudio	4	Architecture Development	55K	604

C/C++ Java

- We recovered the ground truth of Chromium.
- We updated the ground truths of Bash and ArchStudio.
- All ground truths have been validated by developers.

Techniques Evaluated

- ACDC
- Bunch-NACH
- Bunch-SAHC
- WCA-UENM
- WCA-UEM
- LIMBO

Dependency-based techniques

- ARC
- ZBR-tok
- ZBR-uni

Information retrieval techniques

Metrics Selection

- It is hard to compare graphs using a single number.
- We use **4** different metrics:
  - **MoJoFM** was commonly used in previous work.
  - **a2a** measures the evolution of the architecture. It address some of MoJoFM issues.
  - **c2c** assesses component-level accuracy.
  - **TurboMQ** measures the quality of an architecture independently of a ground truth.
- All metrics show that **using symbol dependencies improve architecture recovery.**

Metrics: MoJoFM

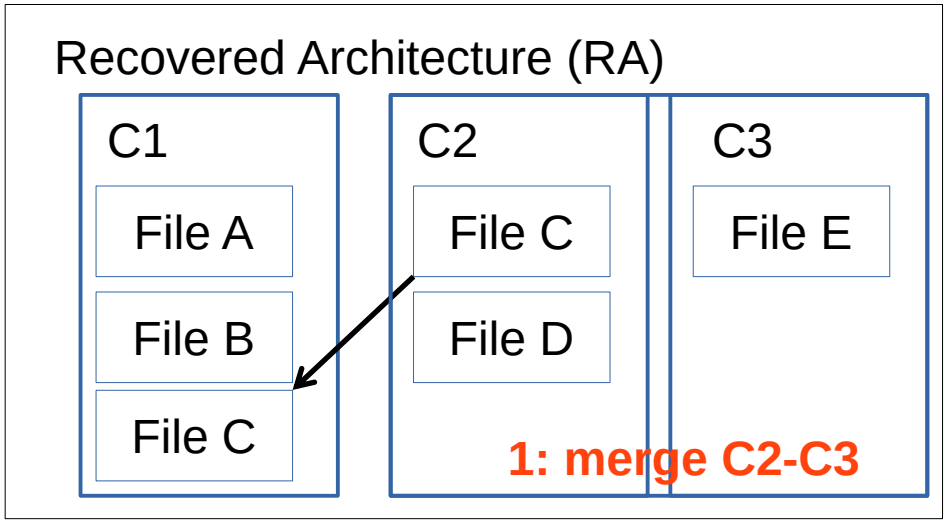
$$MoJoFM(M) = (1 - \frac{mno(A,B)}{max(mno(\forall A,B))}) \times 100\%$$

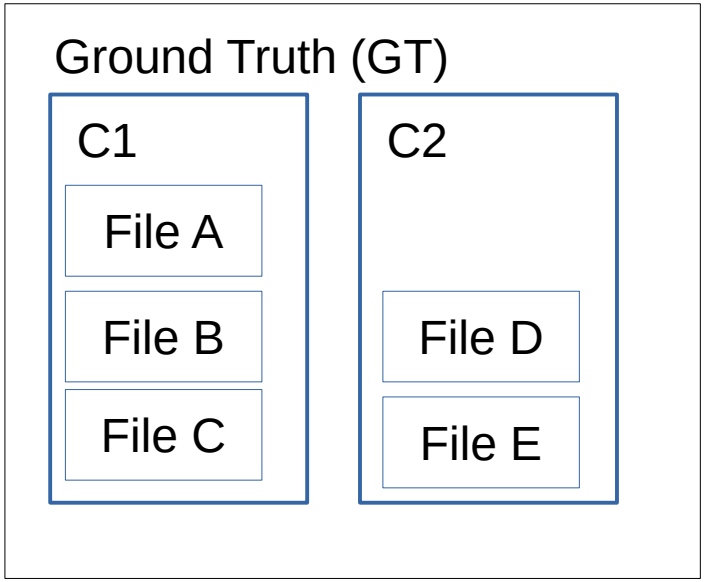
$mno(A,B)$ Minimum number of operations to transform A into B.

Possible operations are:

- To move an element from cluster 1 to cluster 2,
- To merge 2 clusters.

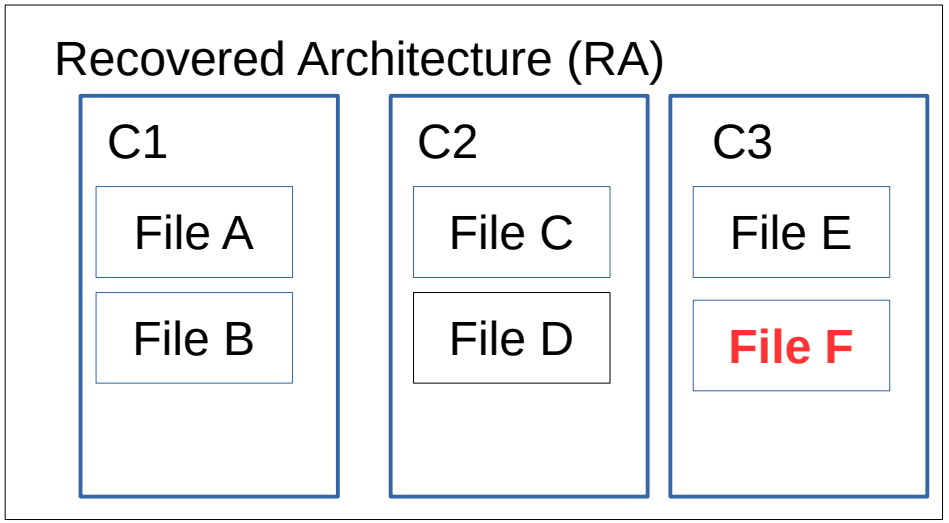
Metrics: MoJoFM

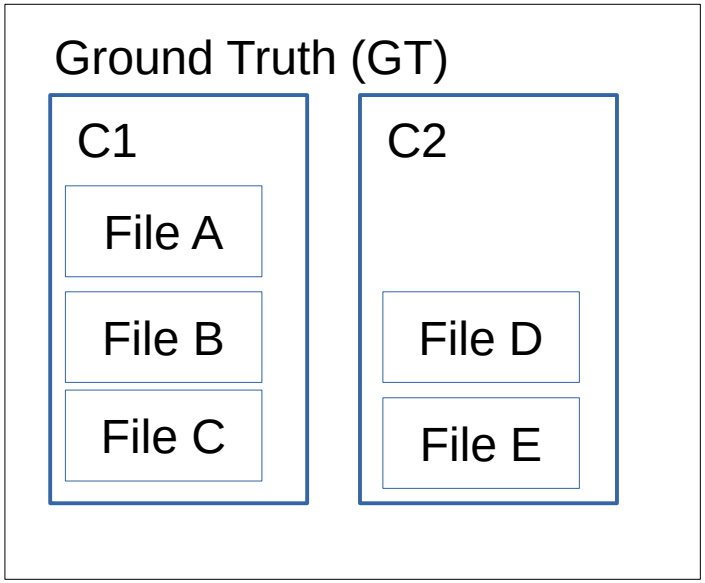




- **2** operations from RA to GT.
- **MoJoFM=33%**

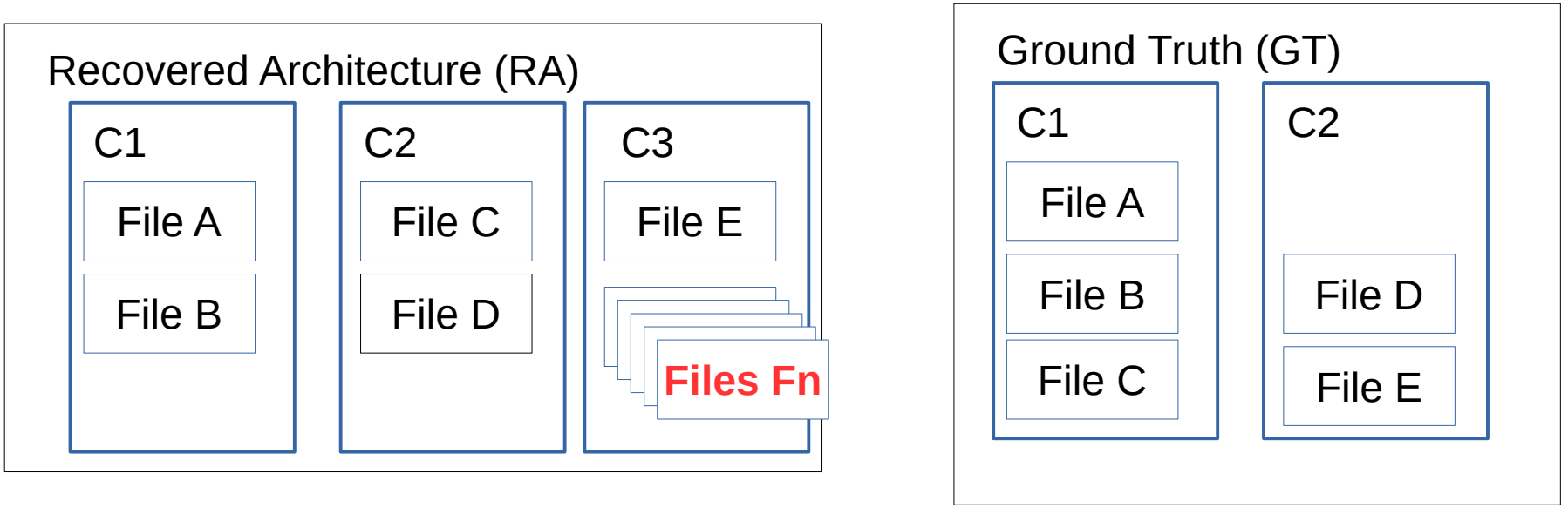
Metrics: MoJoFM





- What happens if some elements of RA are not in GT?
- *mno* cannot be correctly calculated.
- MoJoFM results will be **inaccurate**.
- With the existing implementation:
- *File F* will be ignored.
- **MoJoFM=33%**
- It's the same value than for the previous architecture!

Metrics: MoJoFM



- What happens if some elements of RA are not in GT?
- *mno* cannot be correctly calculated.
- MoJoFM results will be **inaccurate**.
- With the existing implementation:
- *Files Fn* will be ignored.
- **MoJoFM=33%**
- It's the same value than for the previous architecture!

Metrics: a2a

$$a2a(A_i, A_j) = (1 - \frac{mto(A_i, A_j)}{aco(A_i) + aco(A_j)}) \times 100\%$$

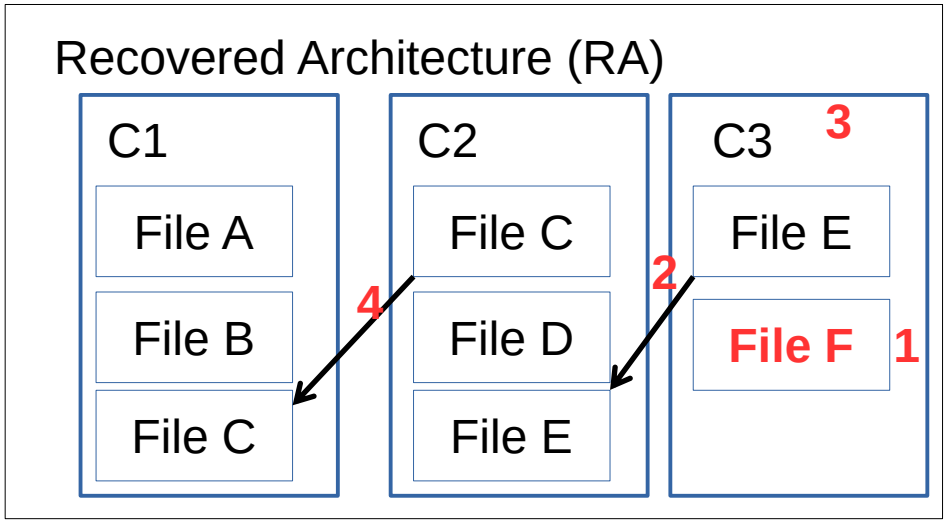
$mto(A_i, A_j)$ Minimum number of operations to transform A_i to A_j.

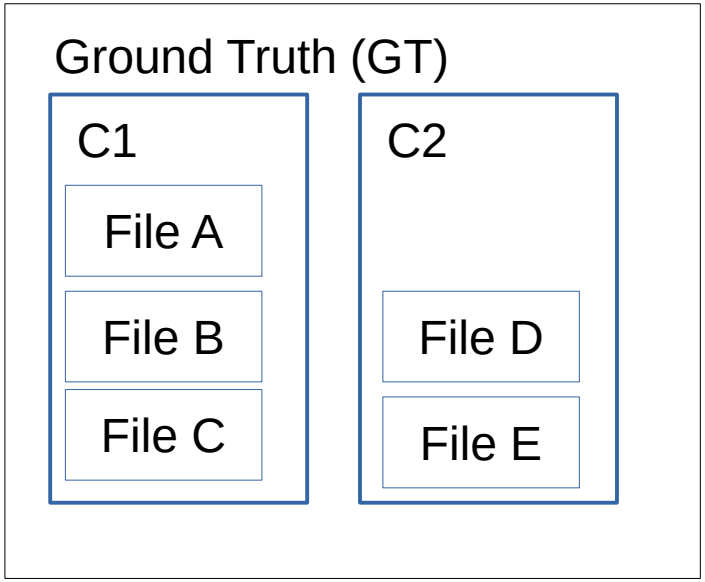
$aco(A_i)$ Minimum number of operations to obtain A_i from a “null” architecture.

Possible operations are:

- Remove a cluster, add a cluster,
- **Remove an element, add an element,**
- Move an element from cluster 1 to cluster 2.

Metrics: a2a





- Remove File F
- Move File E from C3 to C2
- Remove C3
- Move File C from C2 to C1

With *File F*:
4 moves from RA to GT
a2a=75%

Without *File F*: **3** moves
from RA to GT
a2a=80%

Experimental Results

- A2a (%) results:

Algo.	Bash	
	Inc	Sym
ACDC	64	81
B-NAHC	66	86
B-SAHC	68	87
WCA-UE	63	81
WCA-UENM	63	81
LIMBO	63	80
ARC	67	
ZBR-tok	31	
ZBR-uni	32	

Include Dependencies

Symbol Dependencies

Source-Code based techniques

Experimental Results

- a2a (%) results:

Algo.	Bash	
	Inc	Sym
ACDC	64	81
B-NAHC	66	86
B-SAHC	68	87
WCA-UE	63	81
WCA-UENM	63	81
LIMBO	63	80
ARC	67	
ZBR-tok	31	
ZBR-uni	32	

Include Dependencies

Symbol Dependencies

Source-Code based techniques

- Using **symbol dependencies improves the results.**
- We obtained similar results with other metrics.

Symbol vs. Include deps.

- a2a (%) results:

Algo.	Bash		ITK		Chromium		ArchStudio		Hadoop	
	Inc	Sym	Inc	Sym	Inc	Sym	Inc	Sym	Inc	Sym
ACDC	64	**81**	67	**74**	71	**73**	71	**88**	68	**84**
B-NAHC	66	**86**	71	**80**	69	**73**	71	**83**	68	**81**
B-SAHC	68	**87**	69	**80**	60*	**71***	72	**85**	69	**83**
WCA-UE	63	**81**	74	**82**	70	**75**	71	**84**	68	**81**
WCA-UENM	63	**81**	74	**82**	70	**75**	71	**84**	68	**81**
LIMBO	63	**80**	71	**80**	TO	**71**	67	**79**	68	**80**
ARC	67		60		56		87		84	
ZBR-tok	31		MEM		MEM		85		81	
ZBR-uni	32		MEM		MEM		86		83	

- Using **symbol dependencies improves the results.**
- We obtained similar results with other metrics.

Symbol vs. Include deps.

- TurboMQ results:

Algo.	Bash		ITK		Chromium		ArchStudio		Hadoop	
	Inc	Sym	Inc	Sym	Inc	Sym	Inc	Sym	Inc	Sym
ACDC	5	17	503	422	183	443	19	52	13	22
B-NAHC	3	5	6	27	3	7	8	22	12	10
B-SAHC	3	7	5	193	141*	291*	16	22	11	14
WCA-UE	0.1	2	0.5	2	0.1	1	0.5	18	1	7
WCA-UENM	0.1	2	0.5	1.5	0.1	1	0.5	18	1	7
LIMBO	1	5	1.8	2.5	TO	1	1	25	1	15
ARC	3	9	0.1	0.3	8	10	14	37	5	25
ZBR-tok	0.3	1	MEM		MEM		4	15	3	13
ZBR-uni	0.6	0.6	MEM		MEM		3	15	4	17

- Using **symbol dependencies improves the results.**
- We obtained similar results with other metrics.

Scalability

Algo.	Bash		ITK		Chromium		ArchStudio		Hadoop	
	Inc	Sym	Inc	Sym	Inc	Sym	Inc	Sym	Inc	Sym
ACDC	Few seconds		< 1 h		< 2 hrs		Few seconds		Few seconds	
B-NAHC			2 h	1 h	~24h	~20h				
B-SAHC			2-3h		24h*	24h*				
WCA-UE					~14h	~8 h				
WCA-UENM										
LIMBO					TO	~14 h				
ARC			2-3 h		~8 h.					
ZBR-tok	Few minutes		MEM		MEM		Few minutes		Few minutes	
ZBR-uni			MEM		MEM					

MEM: The algorithm ran out of memory (Java heap 48GB).

***:** The algorithm was stopped after 24 hours. Intermediate results are available.

TO (Time Out): The algorithm did not terminate after 3 days.
No intermediate results are provided.

Lessons Learned

- The **granularity of the dependencies** affects the scalability and accuracy of the recovery techniques.
- Extreme architectures expose limits of the metrics.
- Different techniques can be used for different purposes:
  - Large software: ACDC, ARC, WCA
  - Low-level architectures: ACDC
  - Different level of abstractions: WCA, LIMBO, ARC

Conclusion

- We evaluated 9 recovery techniques on 5 projects.
- Using **symbol dependencies** improves the accuracy of recovery techniques.
- Our study include Chromium, the largest project used in an evaluation of architecture recovery techniques.