Precise and Scalable Detection of Use-after-Compacting-Garbage-Collection Bugs

HyungSeok Han, Andrew Wesie, Brian Pak

Theori Inc.

Garbage Collection (GC)

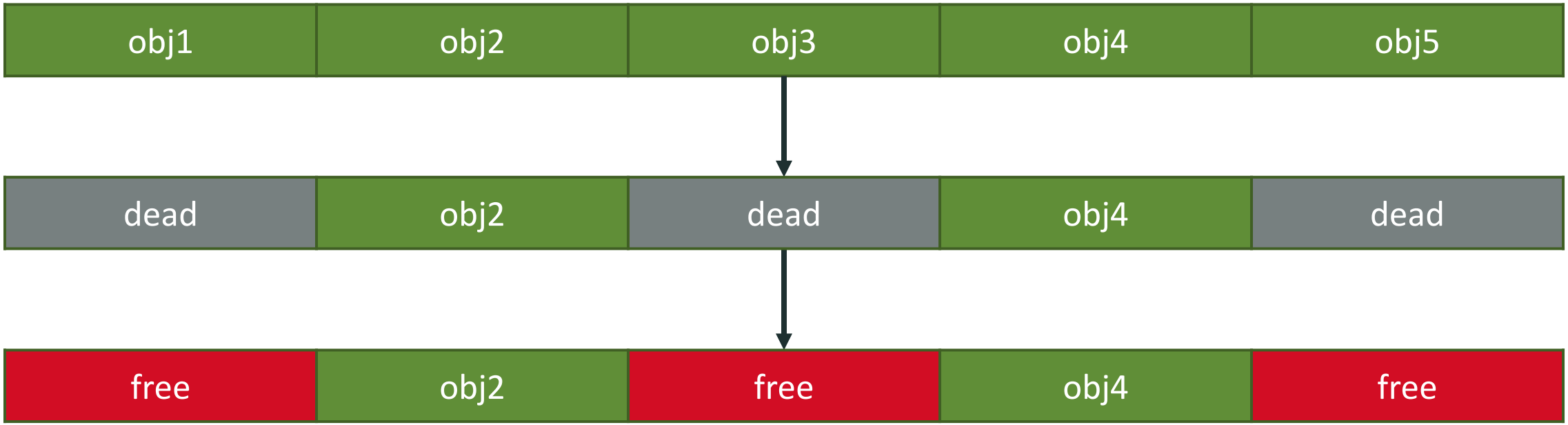
- Automatic memory management system.

.NET

...

Garbage Collection (GC)

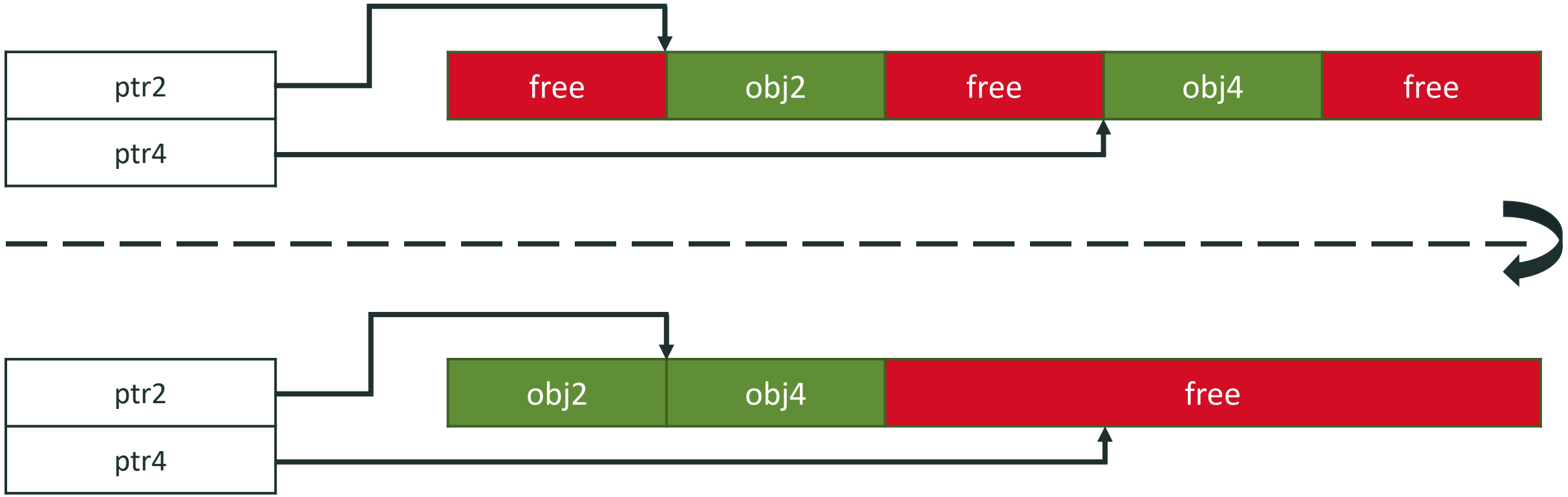
- Automatic memory management system.



Memory fragmentation

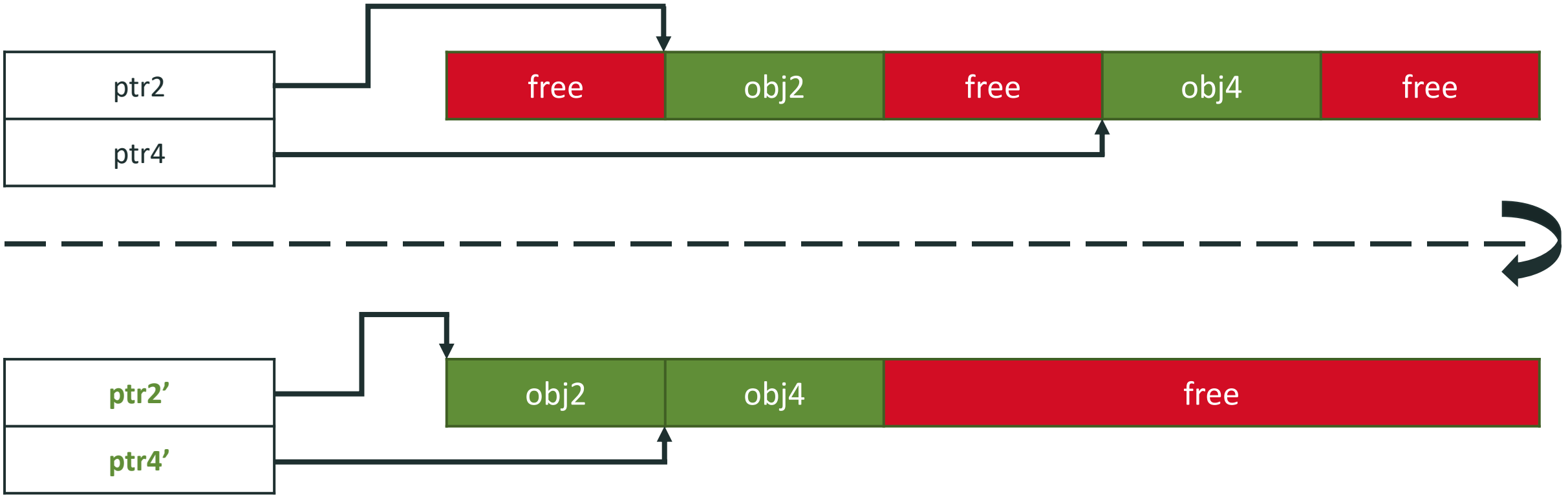
Compacting GC (CGC)

- Mitigate memory fragmentation by rearranging live memory objects.



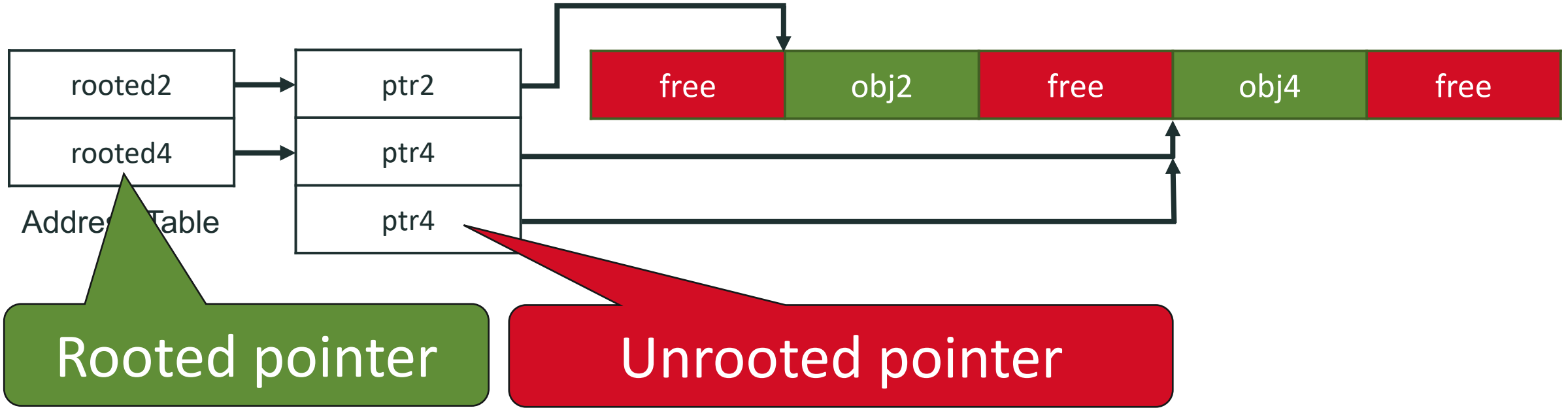
Compacting GC (CGC)

- Mitigate memory fragmentation by rearranging live memory objects.



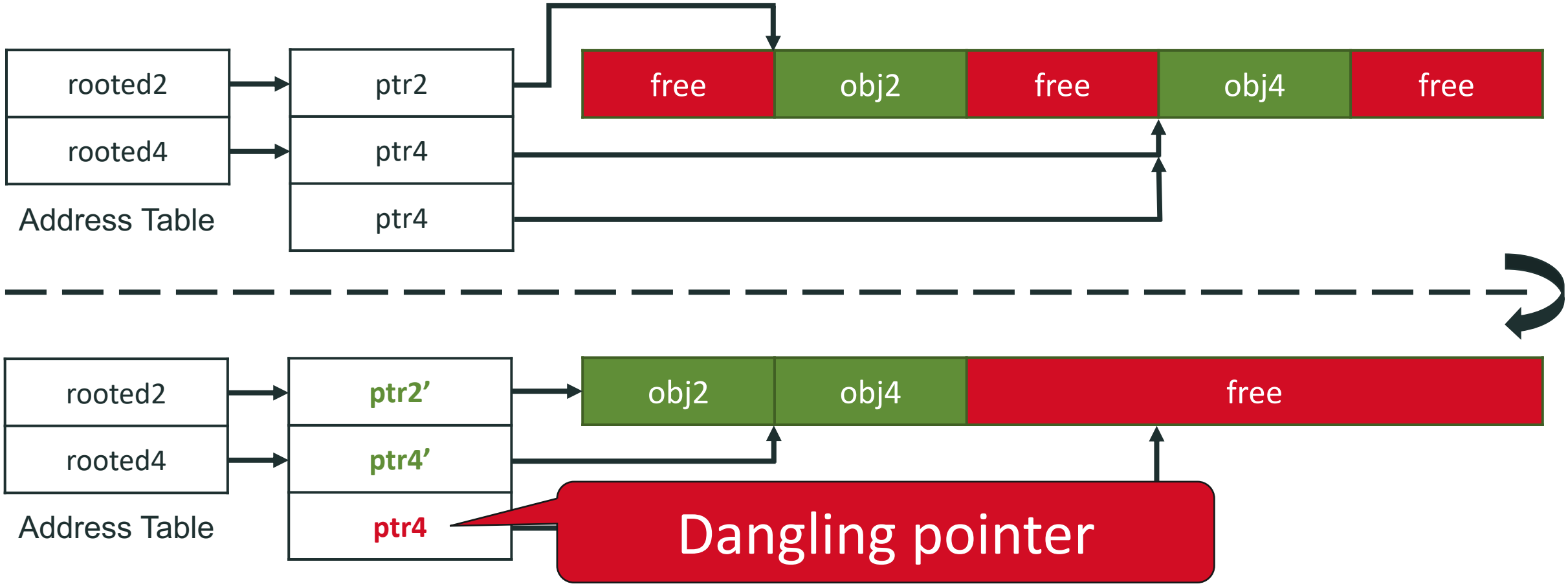
Compacting GC (CGC)

- To update references, it manages an address table containing memory addresses where the references are stored.



Compacting GC (CGC)

- To update references, it manages an address table containing memory addresses where the references are stored.

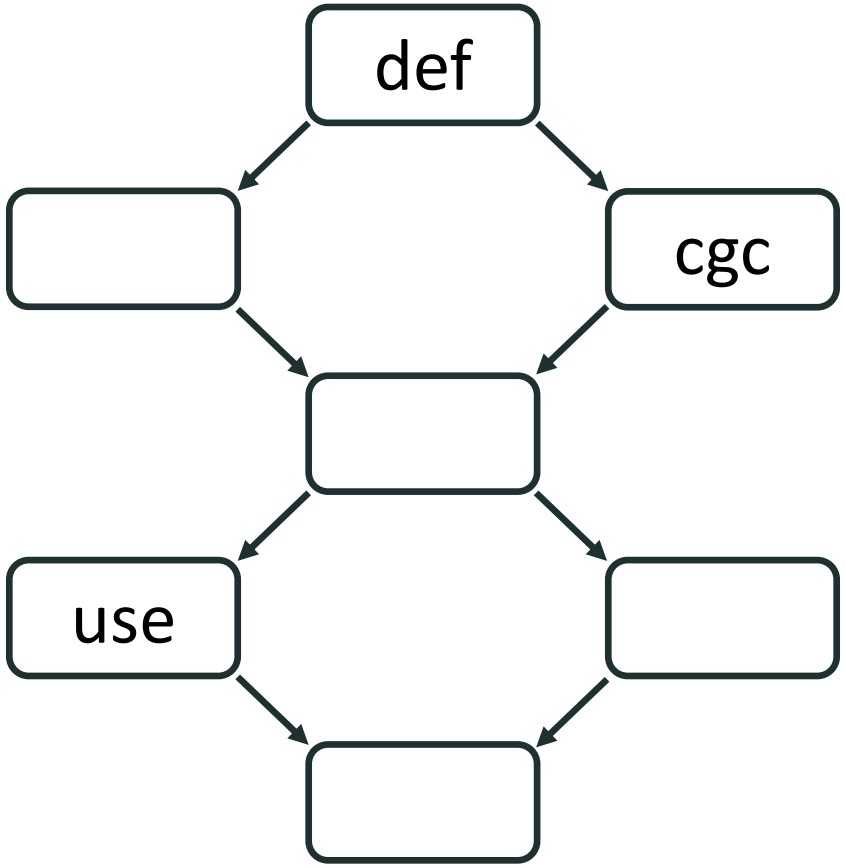


Use-after-Compacting-GC

- A kind of use-after-free bug caused by the **use of an unrooted pointer** that becomes a dangling pointer **after compacting GC.**

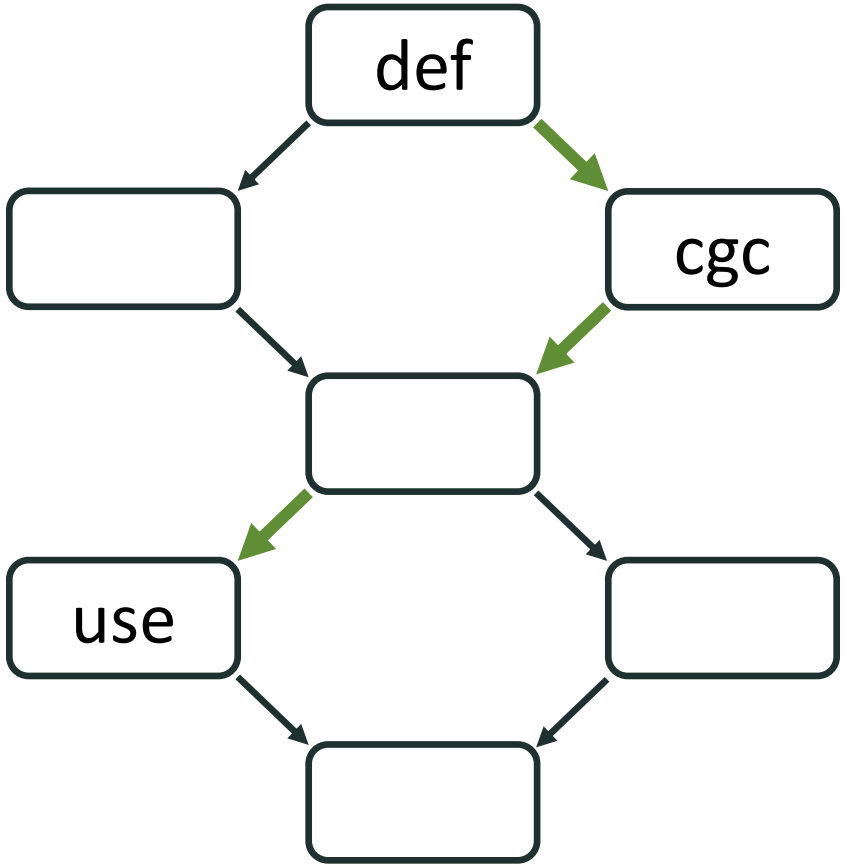
Use-after-Compacting-GC

- A kind of use-after-free bug caused by the **use of an unrooted pointer** that becomes a dangling pointer **after compacting GC.**



Use-after-Compacting-GC

- A kind of use-after-free bug caused by the **use of an unrooted pointer** that becomes a dangling pointer **after compacting GC.**



Example (CVE-2019-13696)

```
void InterpretedFrame::Summarize(...) const {
  ...
  // define an unrooted pointer, `code`.
  AbstractCode code = AbstractCode::cast(GetBytecodeArray());

  // `GetParameters` triggers a GC. (`code` becomes a dangling pointer)
  Handle<FixedArray> params = GetParameters();

  // `code` is used as a function argument. (use-after-compacting-gc bug)
  FrameSummary::JavaScriptFrameSummary summary(
      isolate(), receiver(), function(), code,
      GetBytecodeOffset(), IsConstructor(), *params);
  ...
}
```

Example (CVE-2019-13696)

```
void InterpretedFrame::Summarize(...) const {
  ...
  // define an unrooted pointer, `code`.
  AbstractCode code = AbstractCode::cast(GetBytecodeArray());

  // `GetParameters` triggers a GC. (`code` becomes a dangling pointer)
  Handle<FixedArray> params = GetParameters();

  // `code` is used as a function argument. (use-after-compacting-gc bug)
  FrameSummary::JavaScriptFrameSummary summary(
      isolate(), receiver(), function(), code,
      GetBytecodeOffset(), IsConstructor(), *params);
  ...
}
```

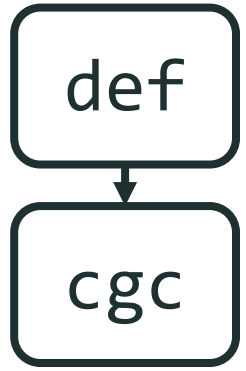
def

Example (CVE-2019-13696)

```
void InterpretedFrame::Summarize(...) const {
  ...
  // define an unrooted pointer, `code`.
  AbstractCode code = AbstractCode::cast(GetBytecodeArray());

  // `GetParameters` triggers a GC. (`code` becomes a dangling pointer)
  Handle<FixedArray> params = GetParameters();

  // `code` is used as a function argument. (use-after-compacting-gc bug)
  FrameSummary::JavaScriptFrameSummary summary(
      isolate(), receiver(), function(), code,
      GetBytecodeOffset(), IsConstructor(), *params);
  ...
}
```

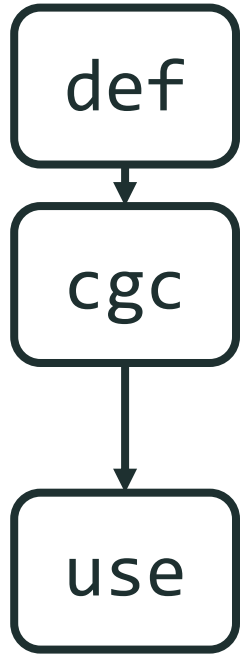


Example (CVE-2019-13696)

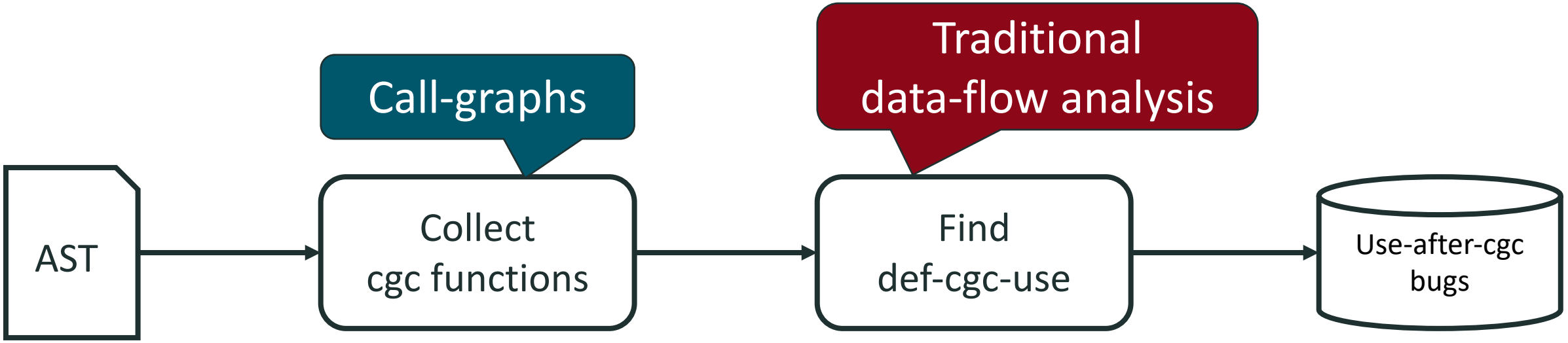
```
void InterpretedFrame::Summarize(...) const {
  ...
  // define an unrooted pointer, `code`.
  AbstractCode code = AbstractCode::cast(GetBytecodeArray());

  // `GetParameters` triggers a GC. (`code` becomes a dangling pointer)
  Handle<FixedArray> params = GetParameters();

  // `code` is used as a function argument. (use-after-compacting-gc bug)
  FrameSummary::JavaScriptFrameSummary summary(
      isolate(), receiver(), function(), code,
      GetBytecodeOffset(), IsConstructor(), *params);
  ...
}
```

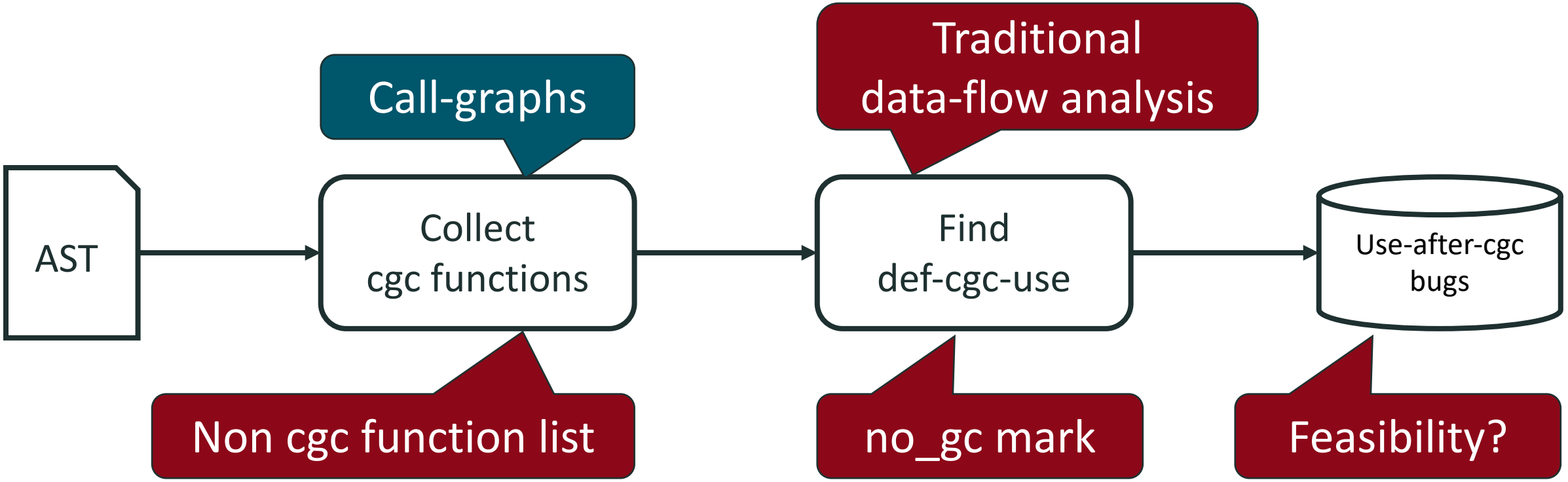


Prior Tools (gcmole, rootAnalysis)



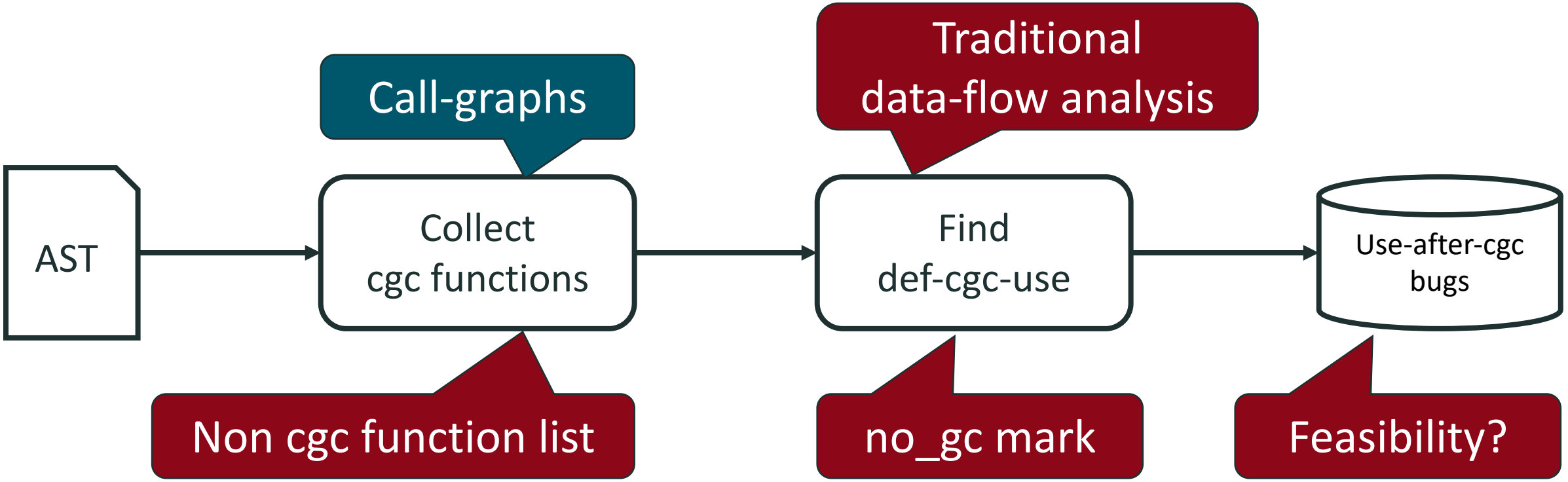
Miss data-flow through memory

Prior Tools (gcmole, rootAnalysis)



Scalable but *Path*-insensitive

Prior Tools (gcmole, rootAnalysis)



Target-specific

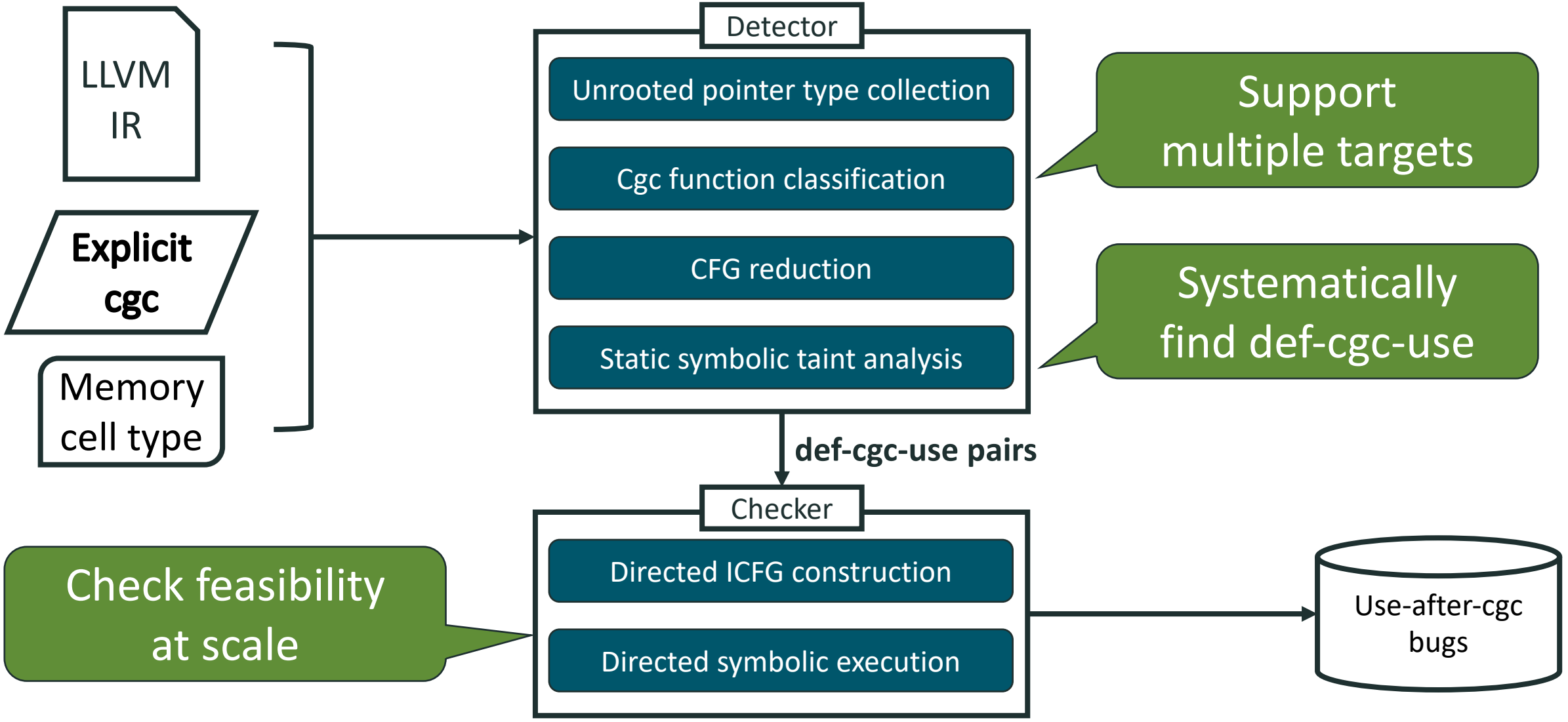
Our Goal

Support multiple targets

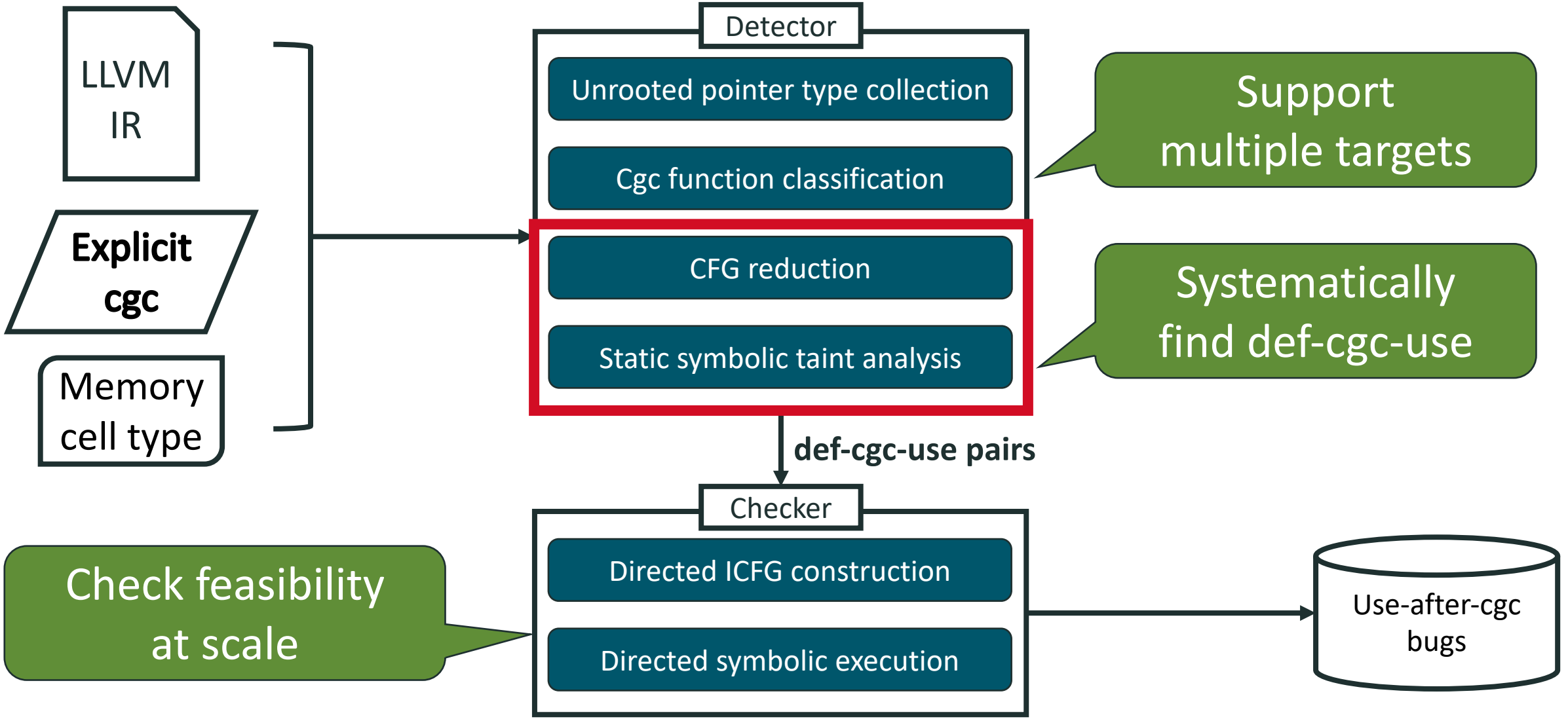
Systematically find def-cgc-use

Check feasibility at scale

CGSan

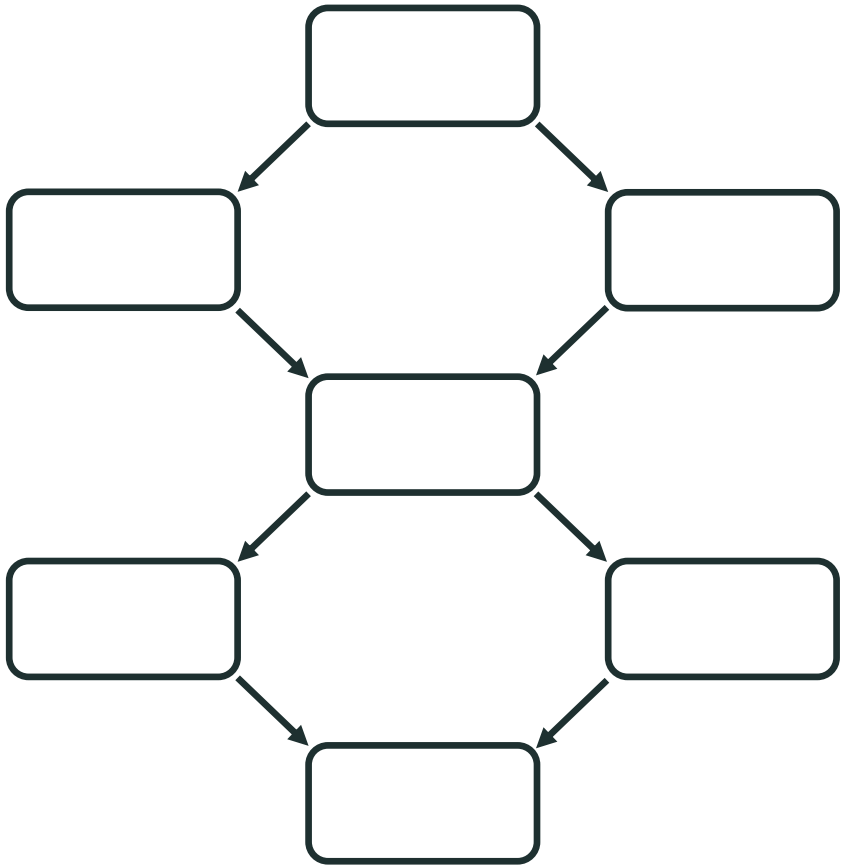

LLVM IR
Explicit cgc
Memory cell type
Detector
Unrooted pointer type collection
Cgc function classification
CFG reduction
Static symbolic taint analysis
Support multiple targets
Systematically find def-cgc-use
def-cgc-use pairs
Checker
Directed ICFG construction
Directed symbolic execution
Check feasibility at scale
Use-after-cgc bugs

CGSan


LLVM IR
Explicit cgc
Memory cell type
Detector
Unrooted pointer type collection
Cgc function classification
CFG reduction
Static symbolic taint analysis
Support multiple targets
Systematically find def-cgc-use
def-cgc-use pairs
Checker
Directed ICFG construction
Directed symbolic execution
Check feasibility at scale
Use-after-cgc bugs

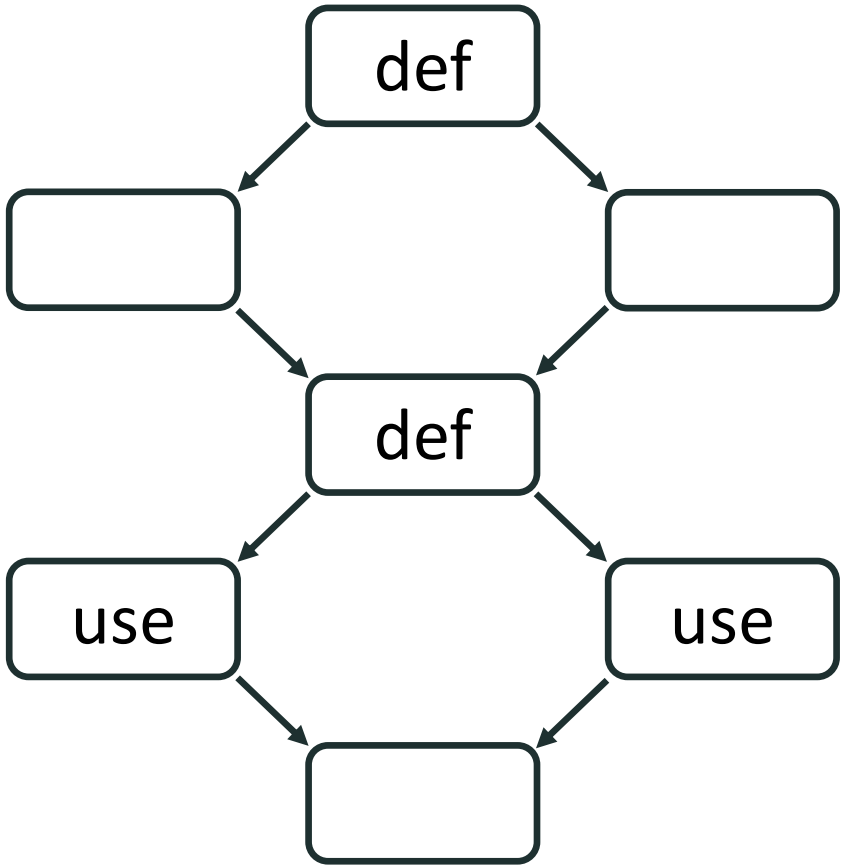
CFG Reduction

- Remove nodes and edges irrelevant to def-cgc-use.



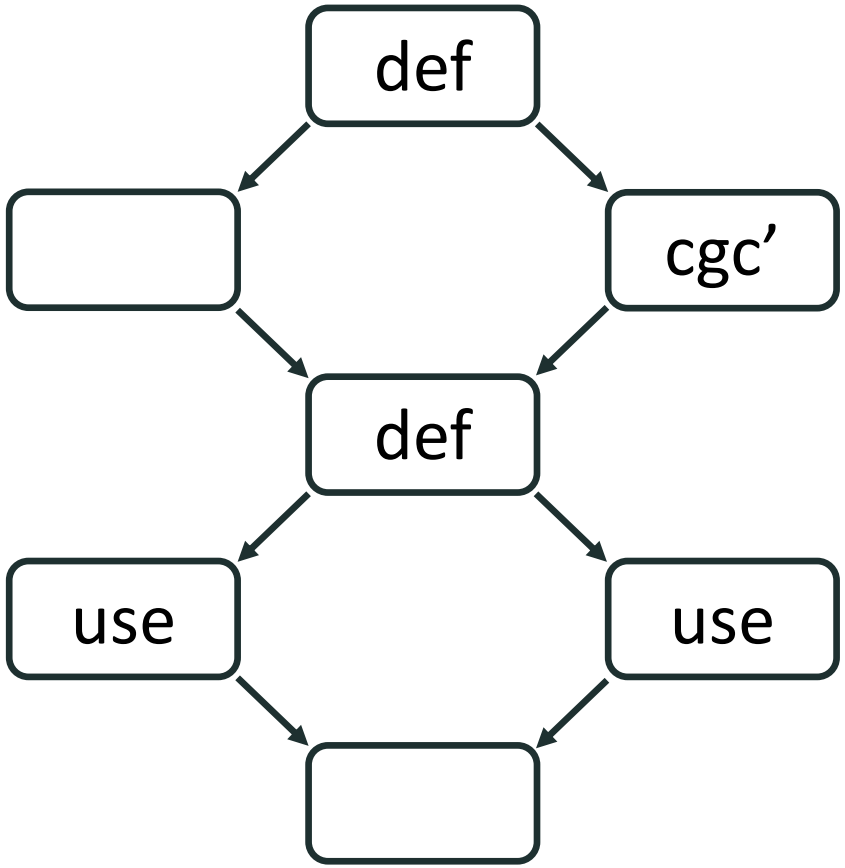
CFG Reduction

- Based on types, find nodes that define or use unrooted pointers.



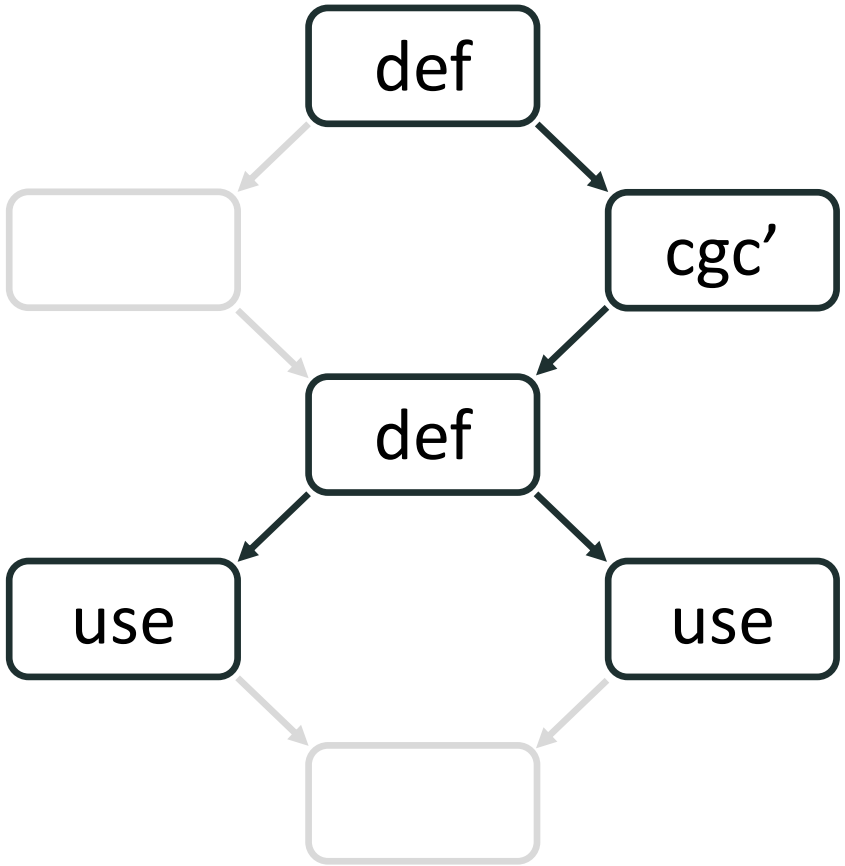
CFG Reduction

- Based on a list of cgc functions, find nodes that may trigger cgc. (Assume that cgc' internally triggers cgc.)



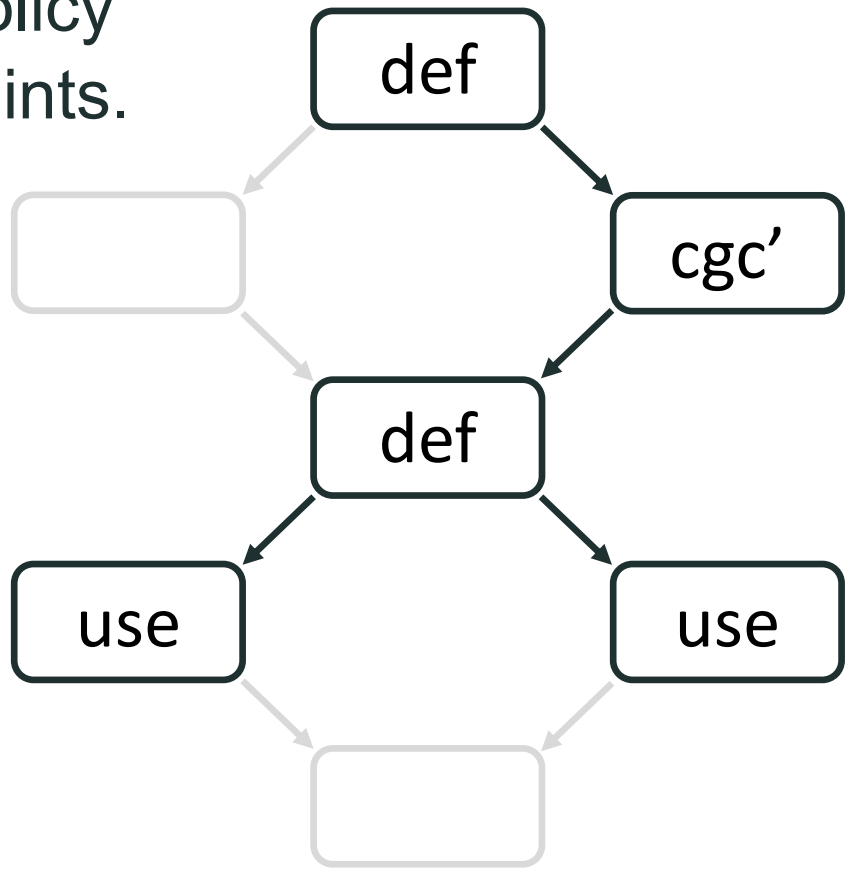
CFG Reduction

- Remove nodes and edges not in def-cgc-use paths.



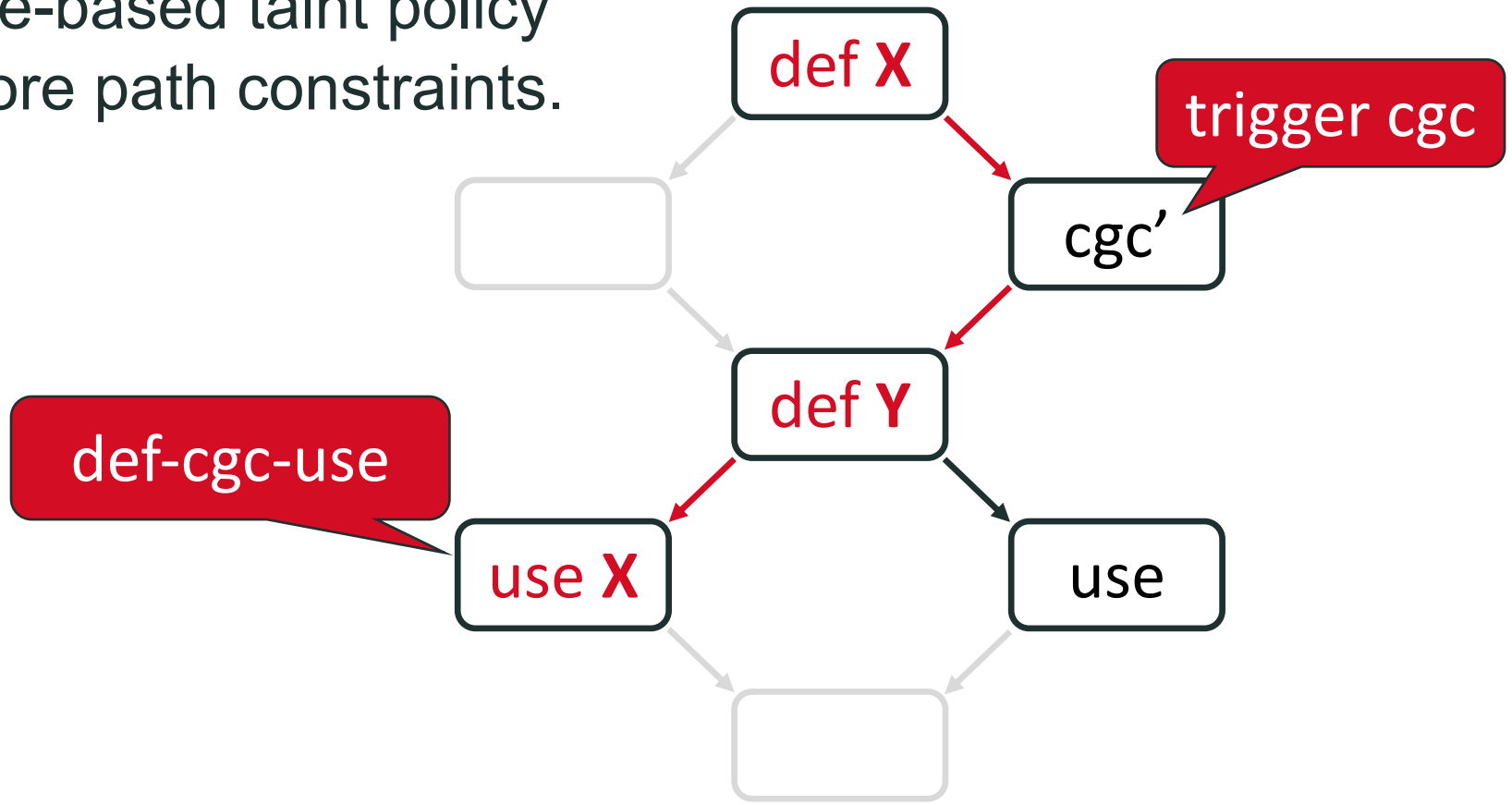
Static symbolic taint analysis

- Intra-procedurally track data-flows of unrooted pointers by symbolic evaluation while following the reduced CFG.
  - Type-based taint policy
  - Ignore path constraints.

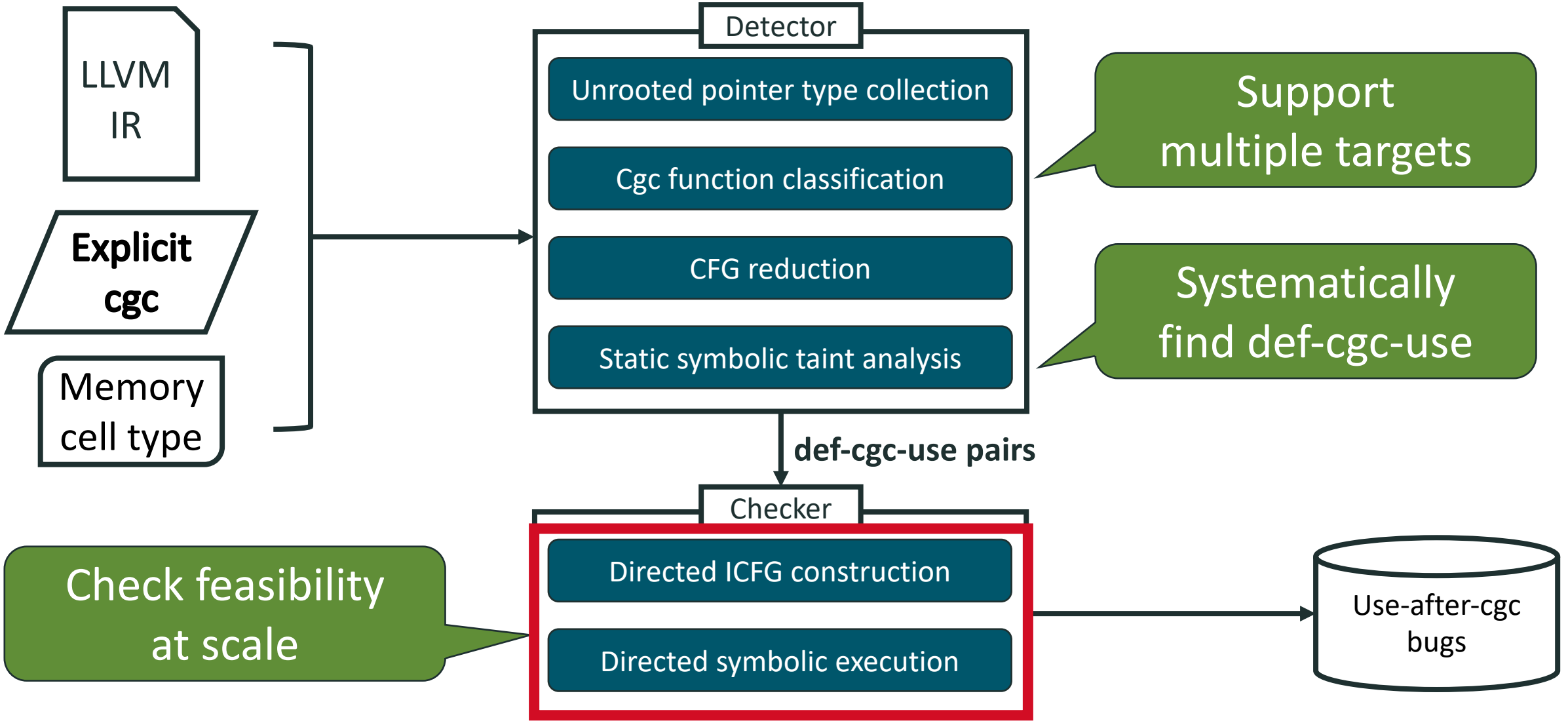


Static symbolic taint analysis

- Intra-procedurally track data-flows of unrooted pointers by symbolic evaluation while following the reduced CFG.
  - Type-based taint policy
  - Ignore path constraints.

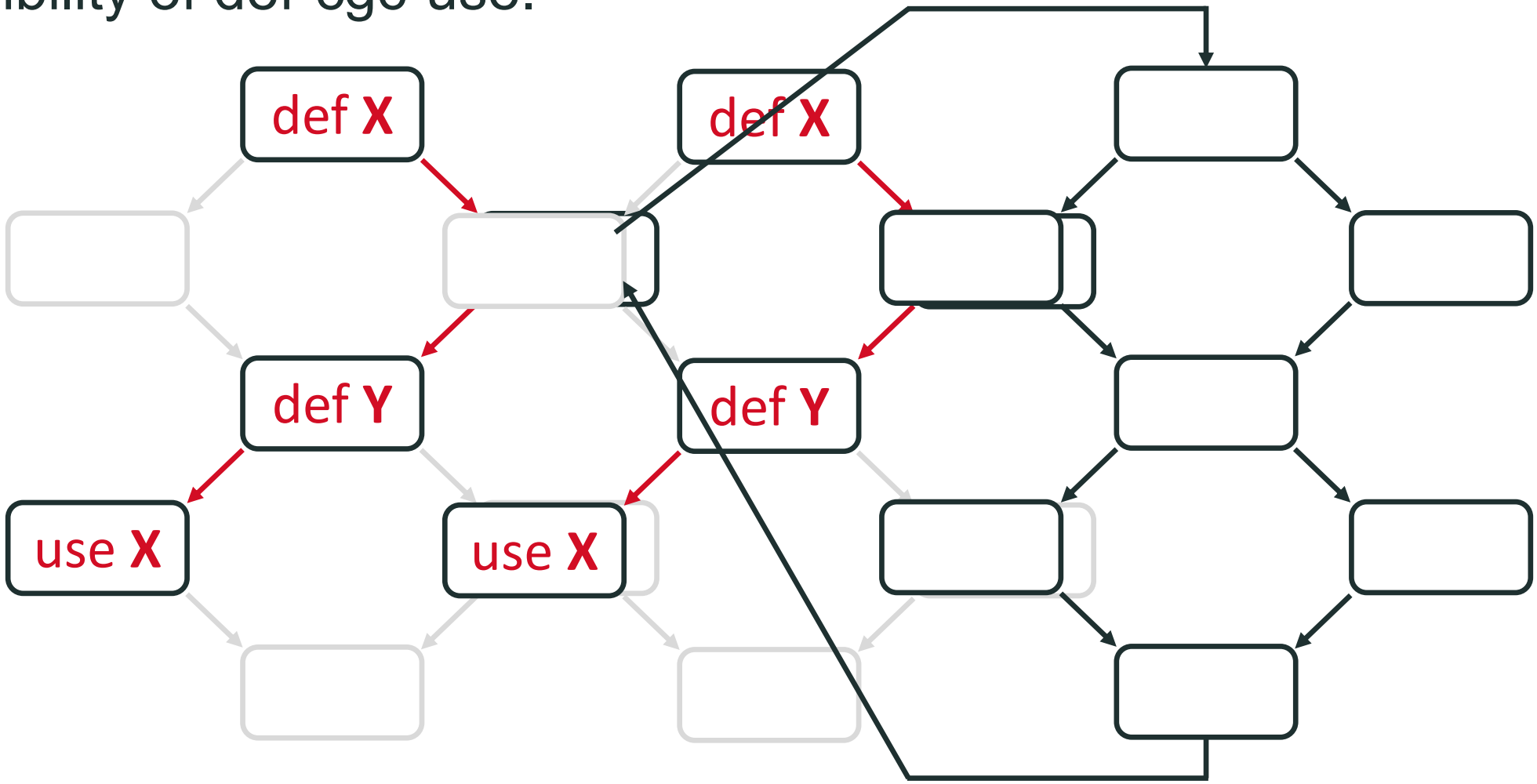


CGSan


LLVM IR
Explicit cgc
Memory cell type
Detector
Unrooted pointer type collection
Cgc function classification
CFG reduction
Static symbolic taint analysis
Support multiple targets
Systematically find def-cgc-use
def-cgc-use pairs
Checker
Directed ICFG construction
Directed symbolic execution
Check feasibility at scale
Use-after-cgc bugs

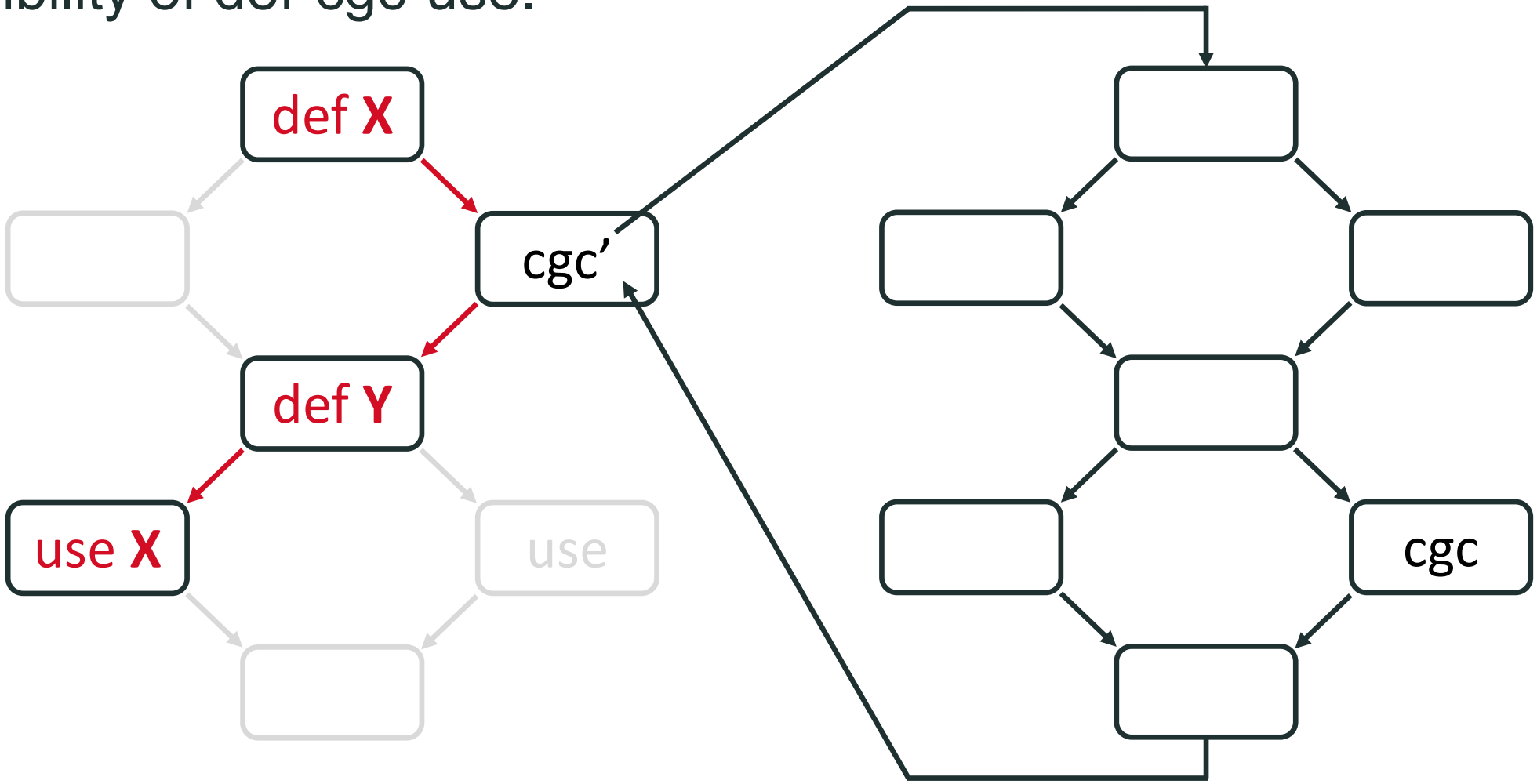
Directed ICFG Construction

- Optimize ICFG by removing nodes and edges irrelevant to checking feasibility of def-cgc-use.



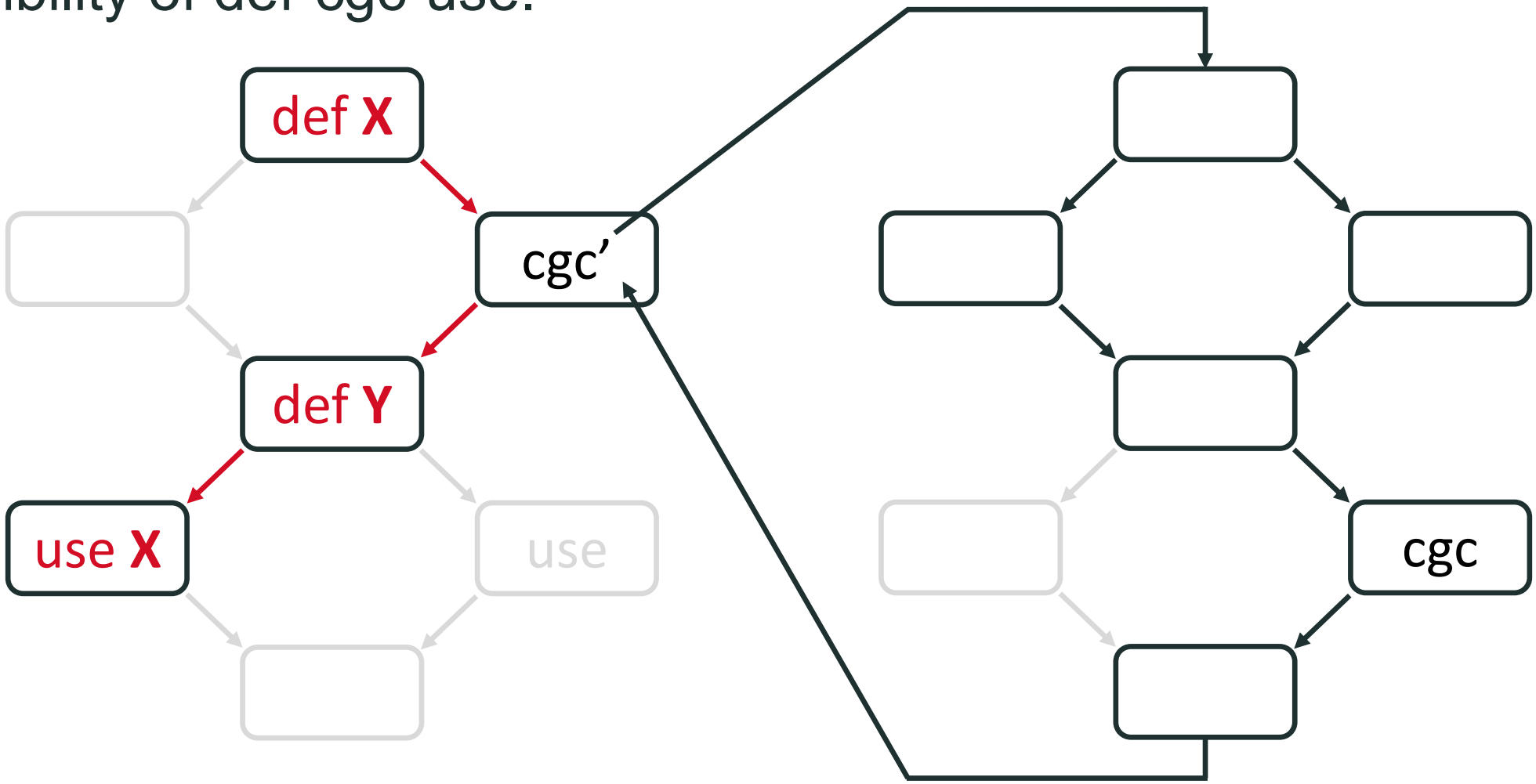
Directed ICFG Construction

- Optimize ICFG by removing nodes and edges irrelevant to checking feasibility of def-cgc-use.



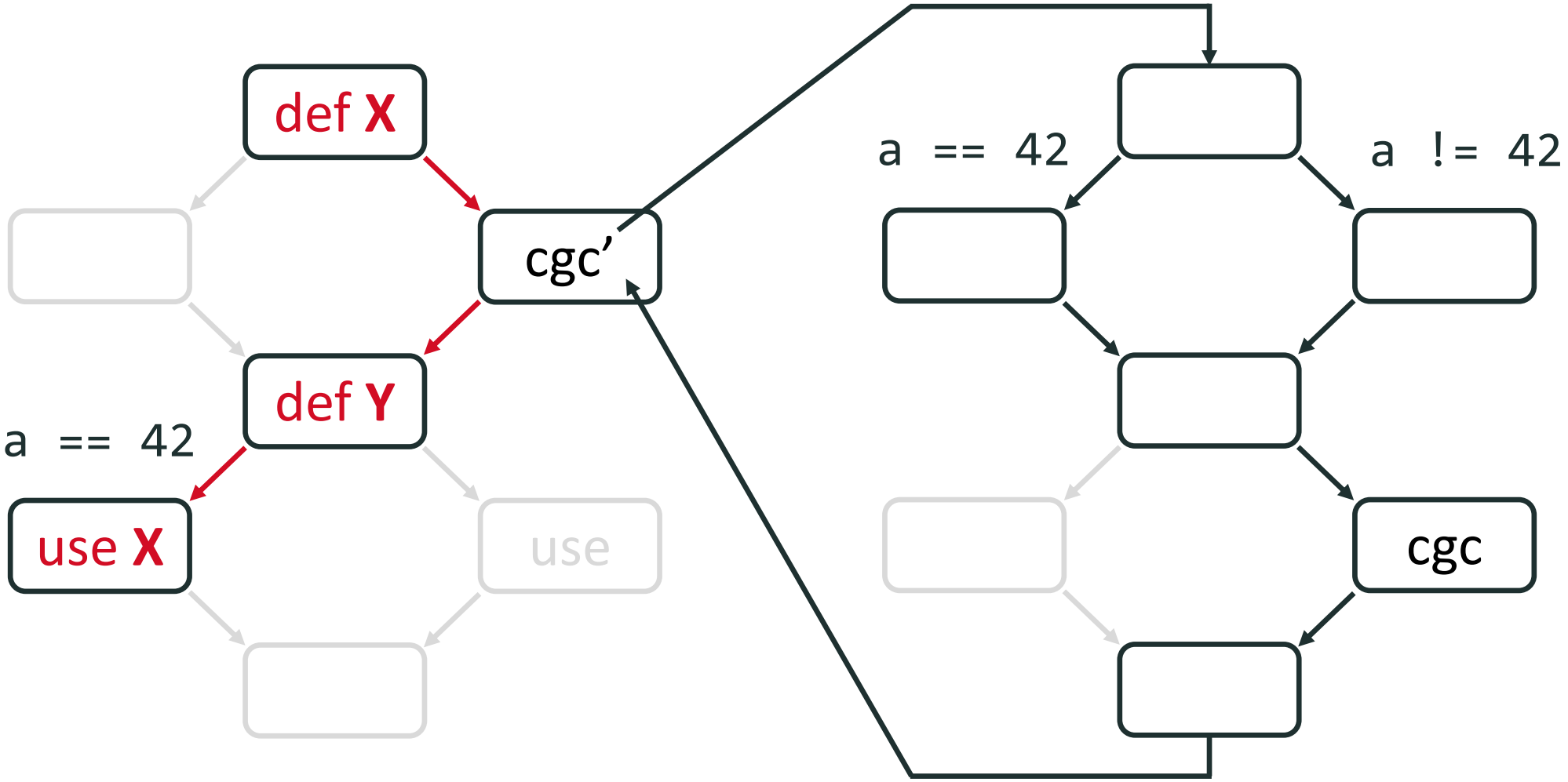
Directed ICFG Construction

- Optimize ICFG by removing nodes and edges irrelevant to checking feasibility of def-cgc-use.



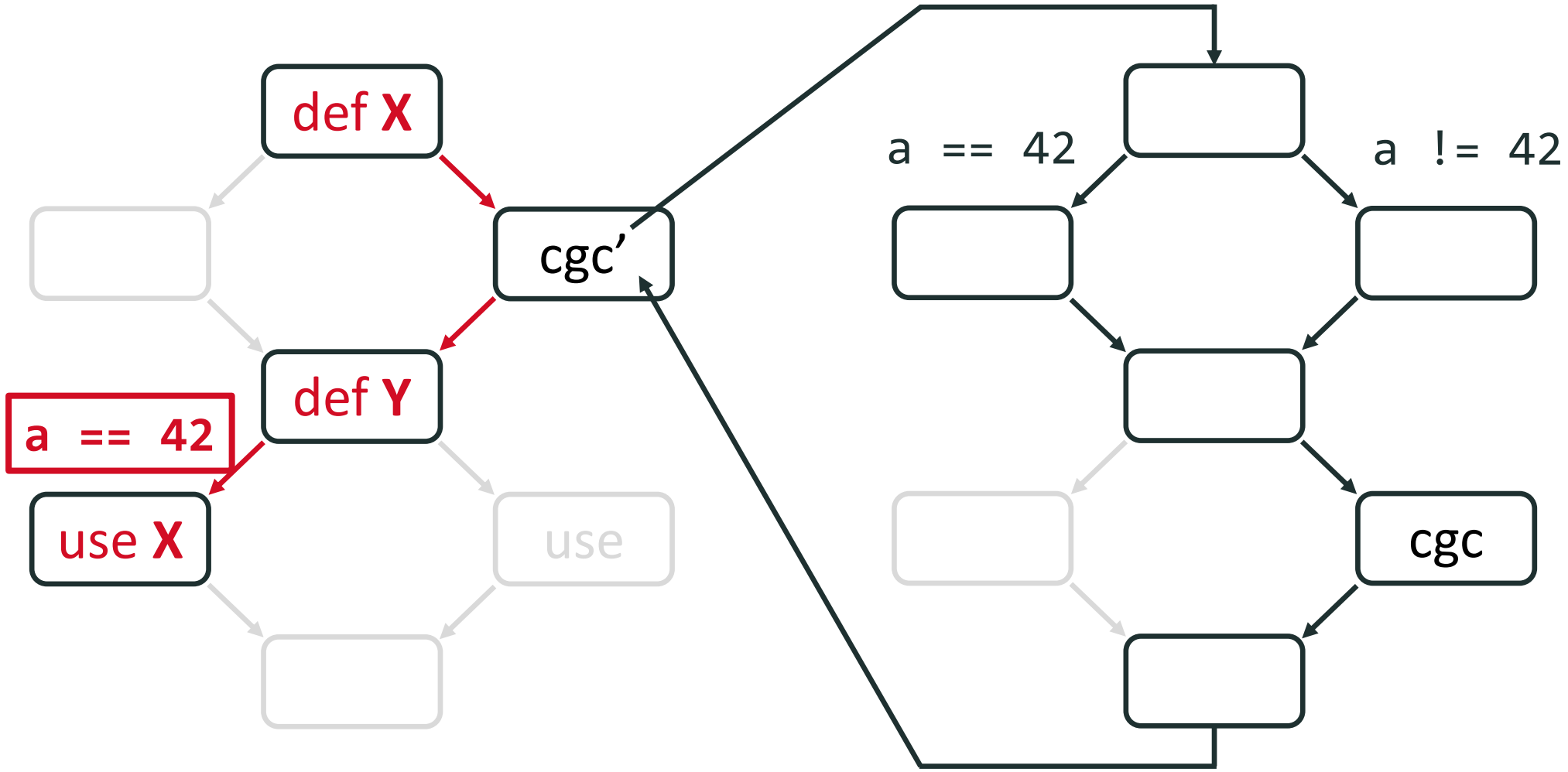
Directed ICFG Construction

- Propagate constant constraints and optimize the ICFG.



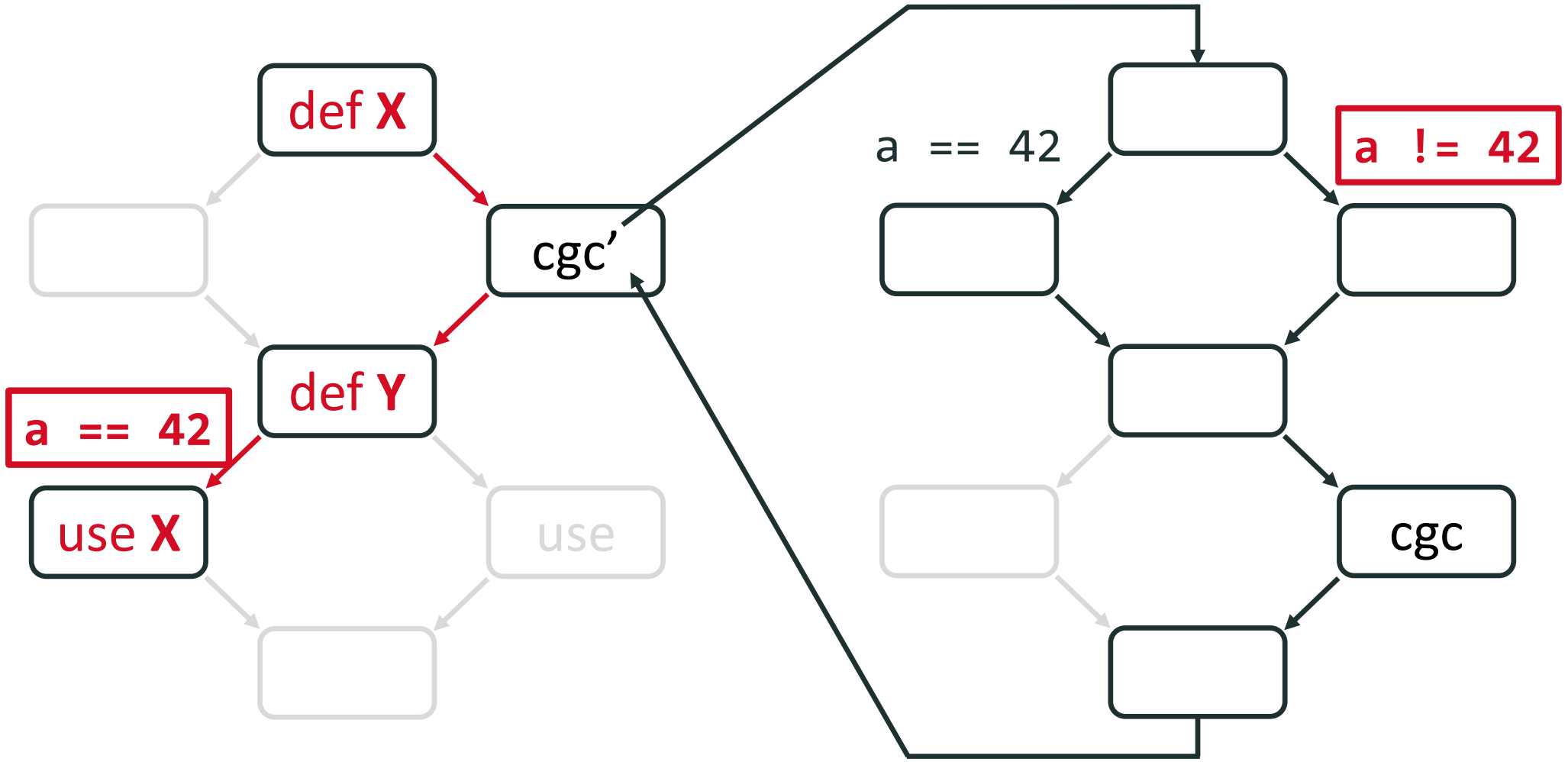
Directed ICFG Construction

- Propagate constant constraints and optimize the ICFG.



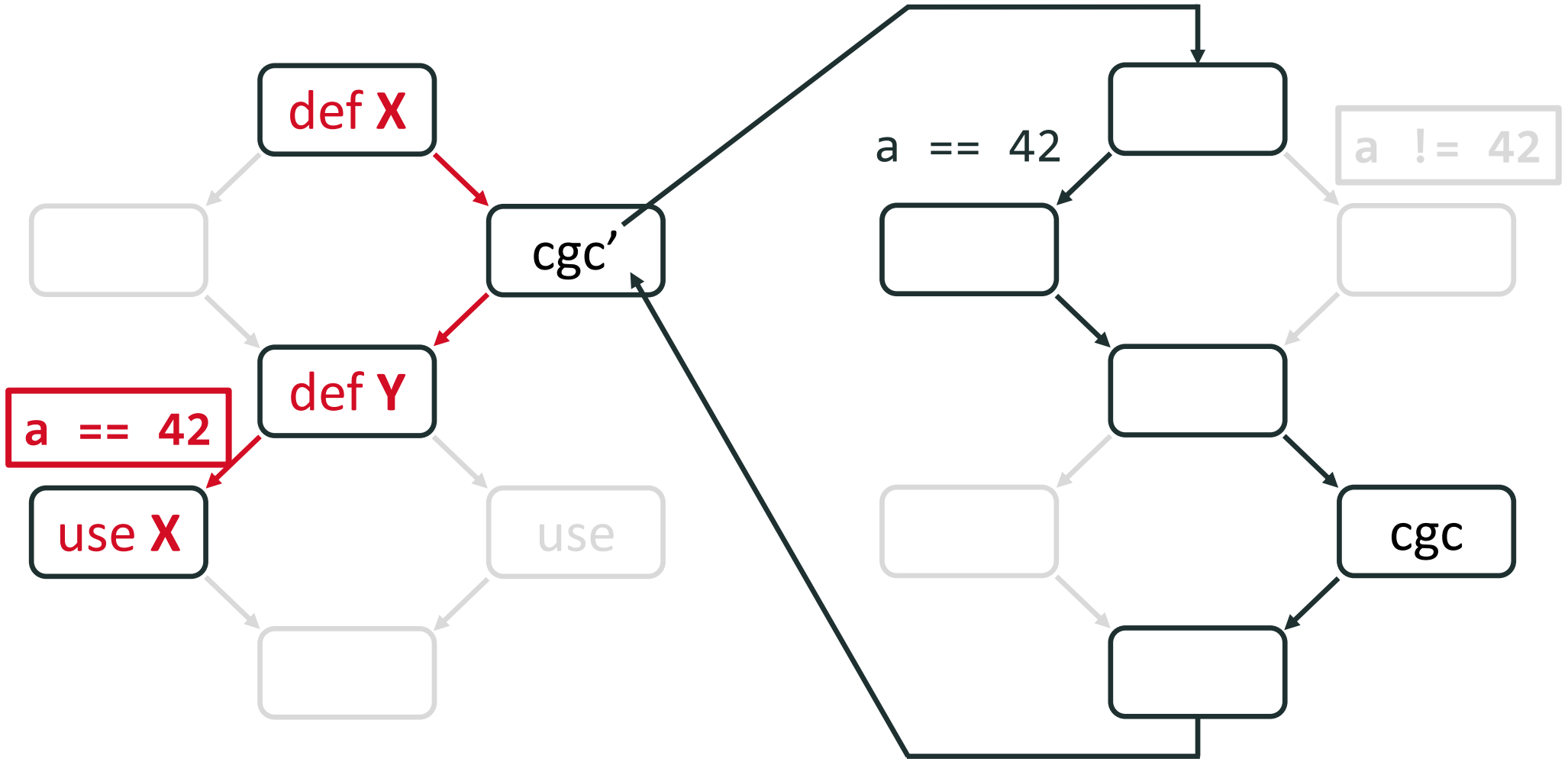
Directed ICFG Construction

- Propagate constant constraints and optimize the ICFG.



Directed ICFG Construction

- Propagate constant constraints and optimize the ICFG.

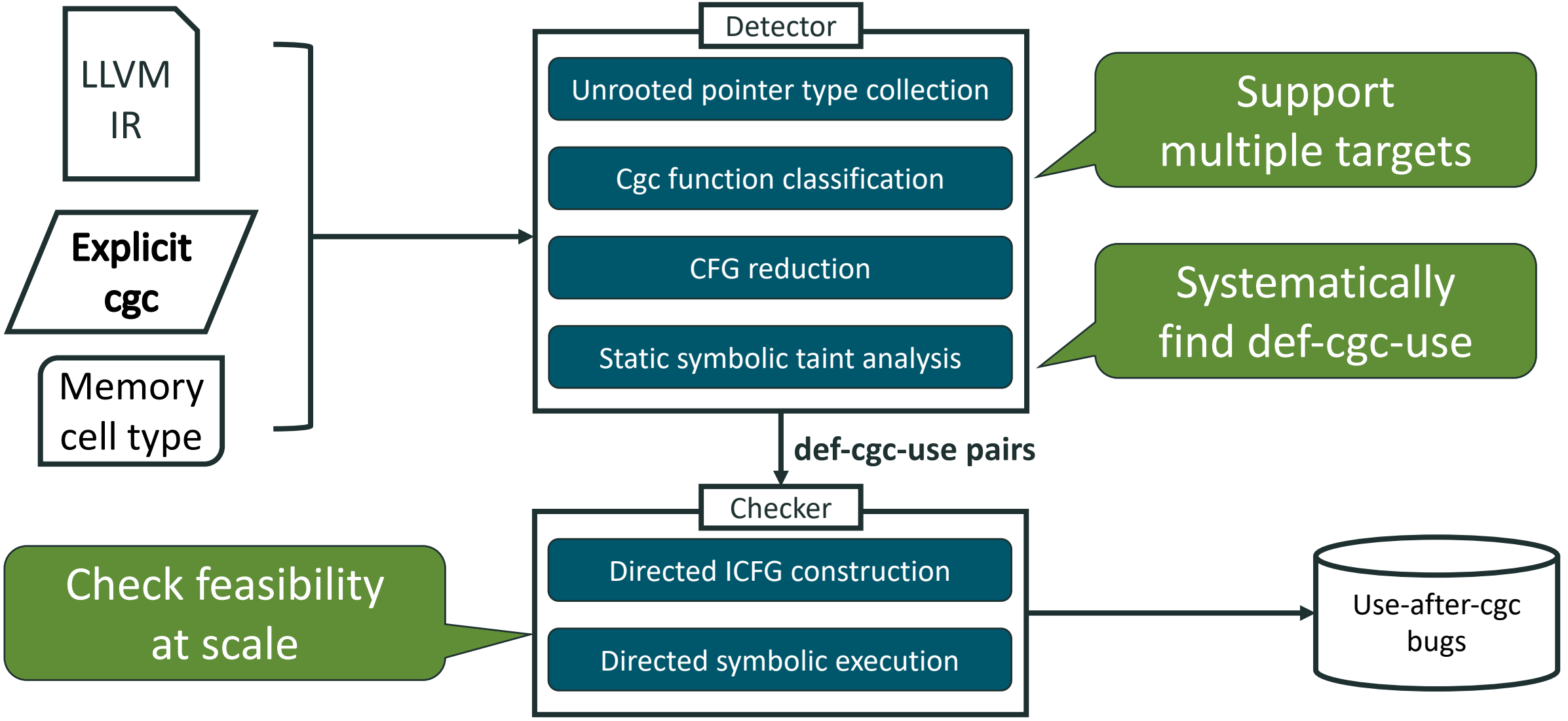


Directed Symbolic Execution

- Check feasibility of def-cgc-use while traversing the directed ICFG.
  - Inter-procedural symbolic execution
  - Under-constrained symbolic execution

- Prioritize path traversals using the directed scheduling
  - Loop scheduling
  - Stateful scheduling
  - Retrievable scheduling

Evaluation


LLVM IR
Explicit cgc
Memory cell type
Detector
Unrooted pointer type collection
Cgc function classification
CFG reduction
Static symbolic taint analysis
Support multiple targets
Systematically find def-cgc-use
def-cgc-use pairs
Checker
Directed ICFG construction
Directed symbolic execution
Check feasibility at scale
Use-after-cgc bugs

Experiment Setup

- The latest two JavaScript engines
  - Google V8 8.1
  - Mozilla SpiderMonkey 74

- Memory cell type
  - V8: Object
  - SpiderMonkey: Cell

- Compacting GC function
  - V8: `CollectGarbage`
  - SpiderMonkey: `gc`

- Timeouts
  - 10 seconds for analyzing a function in the detector module
  - 10 minutes for checking the feasibility of a detected def-cgc-use in the checker module

Detector Statistics

- Detector totally found 1,484 def-cgc-use pairs in 71 minutes while having 230 timeout cases.

- With CFG reduction, the detection time decreased by 40% and timeout cases decreased by 48%.

Target	DETECTOR w/ CFG reduction			DETECTOR w/o CFG reduction		
	def-cgc-use	Timeout	Time	*def-cgc-use*	Timeout	Time
V8	20	112	36m	20	263	67m
SpiderMonkey	1,464	118	35m	1,426	178	51m

Checker Statistics

- Checker found 18 feasible def-cgc-use pairs and verified 1,309 infeasible def-cgc-use pairs.

- After using the directed ICFG, checker detected 2.25x feasible def-cgc-use pairs and 163x infeasible def-cgc-use pairs.
  - The average time to check a def-cgc-use pair decreased by 83%.

Target	CHECKER w/ Directed ICFG				CHECKER w/o Directed ICFG			
	Feasible	Infeasible	Timeout	Avg. Time	Feasible	Infeasible	Timeout	Avg. Time
V8	18	0	2	98s	8	0	12	364s
SpiderMonkey	0	1,309	155	64s	0	8	1,456	597s

Bug Findings

- Triage def-cgc-use pairs by their function names.

Idx	Function	Def	Use	Status	Patch Strategy	Prev.
1	BaseNameDictionary<Derived, Shape>::CollectKeysTo	Call	Call	Fixed	Relocate *compact-gc*	✗
2	Deserializer::DeserializeDeferredObjects	Call	Call	Fixed	Remove *compact-gc*	✗
3	Deserializer::ReadObject	Call	Return	Fixed	Remove *compact-gc*	✗
4	Factory::AllocateRawWithImmortalMap	Arg	Call	Won't Fix	-	✓
5	Factory::NewFixedArrayWithFiller	Arg	Call	Fixed	Use *rooted* pointer	✓
6	Logger::ICEvent	Arg	Call	Fixed	Use *rooted* pointer	✓
7	Logger::MapEvent	Arg	Call	Fixed	Use *rooted* pointer	✗
8	Map::DeprecateTransitionTree	Arg	Call	Submitted	-	✗
9	MapUpdater::ConstructNewMap	Call	Call	Fixed	Use *rooted* pointer	✓
10	NativeRegExpMacroAssembler::CheckStackGuardState	Arg	Call	Won't Fix	-	✗
11	ObjectDeserializer::Deserialize	Call	Store	Fixed	Remove *compact-gc*	✗
12	PartialDeserializer::Deserialize	Call	Call	Fixed	Remove *compact-gc*	✗
13	SourceTextModule::AddAsyncParentModule	Arg	Call	Fixed	Use *rooted* pointer	✗
14	ToPropertyDescriptorFastPath	Call	Call	Fixed	Relocate *compact-gc*	✗
15	V8HeapExplorer::AddEntry	Arg	Store	Fixed	Remove *compact-gc*	✓

Limitation

- We assume that compacting GC must move all unrooted pointers.
- We cannot handle compacting GC triggered in other threads.
- Path explosion problem still existed.

More in the paper

- Directed scheduling
- Patch strategies for use-after-cgc bugs
- Other evaluation and details

Question?