CYBER SECURITY COOPERATIVE RESEARCH CENTRE

CREST
Centre for Research on Engineering Software Technologies

THE UNIVERSITY of ADELAIDE

Patching and updating impact estimation

Victor Prokhorenko, Ali Babar

SOUTH AUSTRALIA
THE DEFENCE STATE

Government of South Australia
SA Health

cybersecuritycrc.org.au

Overview

- **Patching and updating**
  - Introduction
  - Goals
  - Taxonomies

- **Existing Runtime Patching Approaches**
  - Practical challenges
  - Features
  - Strategies

- **Patch Impact estimation**
  - Current work and preliminary results
  - Next steps

Terminology

- Dynamic Software Updating (DSU)
- Hotpatching
- Hotfixes
- Live updating
- Live patching
- Run-time patching
- Run-time updates
- Monkey patching

- In our context – patch and update → *change*

Software evolution

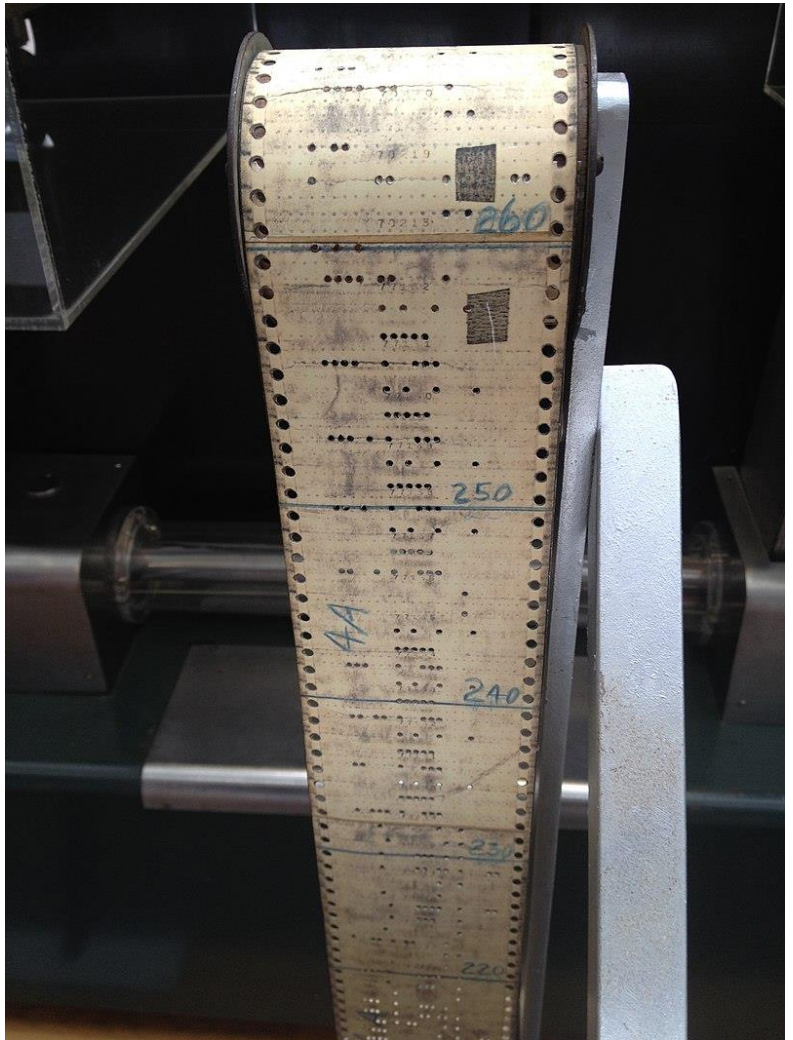
- **Changes due to:** user demands, environment, etc.
- **Expectations (wishes):** adjusted behavior, consistent data, lack of disruptions/downtime
- **Actions:** update, patch
- Common understanding of difference between patching and updating – *small fixes vs. functional changes* (sometimes reflected in versioning systems)

Patching and updating

- Software Development Life Cycle-related issues
  - Coding, Testing, Delivery, Deployment, Running

- **Traditional software updates**
  - Write code / Recompile
  - Deliver / Deploy
  - Stop old code / Run new code

- **Offline patch**
  - Aims to minimise changes



Runtime patching

- **Difference from offline patching**
  - Software is running – activities are carried out, resources are in use
- **Goals**
  - Minimize change-to-action delay, minimize disruptions
- **Challenges to solve**
  - Isolate changed code
  - Find the location of old code to change
  - Find a suitable time to apply the patch
  - Take care of currently running operations
  - Adjust data to match the expectations of the new code
  - Verify new behavior
  - Allow roll-back for failed patches

Existing taxonomies

System				Goals			
Target	Section	Name	Refs	Flexible	Correct	Efficient	Effective
C programs	2.3	OPUS	[ABBS05]	○	◔	○	●
		Ginseng	[NHSO06, SHB+07, NHFP08, NH09]	◔	●	◓	◔
		POLUS	[CYC+07]				
		UpStare	[MB09]				
		Ekiden	[HSHF11]				
		Kitsune	[HSD+12]				
		DynSec	[PBG13]				
OS Kernel		K42	[BHA+05]				
		LUCOS	[CCZ+06]				
		DynaMOS	[MR07]				
		KSplice	[AK09]				
		PROTEOS	[GKT13]				
Java Programs	2.4	JDrums	[RA00]				
		DVM	[MPG+00]				
		HotSwap	[Dmi01, Orab]				
		DUSC	[ORH02]				
		JVolve	[SHM09]				
		DCE VM	[WWS10]				
		JavAdaptor	[PGS+11]				
		JRebel	[KV12]				
OODBMS	2.5	O_2	[FMZ+95, Zic91]				
		Versant	[Ver15]				
		Objectivity/DB	[Obj13]				
		GemStone	[Gem14]				
		PJama	[DA99]				
Language	2.6	LISP	[Ste90]				
		Smalltalk	[GR83]				
		Erlang	[AVWW96]				
		UpgradeJ	[BPN08]				
		OSGi	[OSG14]				

Table 1 Summary of system and function call hooking tools and techniques based on the proposed class

	Microsoft Detours	EasyHook	Nektra SpyStudio	WinAPI Override32	Rohitab API Monitor	Frida	Sysinternals Process Monitor	LD_PRELOAD	DYLD_INSERT_LIBRARIES	Cydia-Substrate	Introspy	Xposed Framework
Subroutine Type / Hooking Scope												
Inner Functions	○	○	○	●	○	●	○	○	○	●	○	●
Exported Functions	●	●	●	●	●	●	●	●	●	●	●	●
System Calls	○	○	○	○	○	○	○	○	○	○	○	○
Hook Insertion												
Dynamic	●	●	●	●	●	●	○	○	○	○	○	○
Static	○	○	○	○	○	○	●	●	●	●	●	●
Instrumentation Type												
Passive	●	●	●	●	●	●	●	●	●	●	●	●
Active	●	●	○	●	●	●	○	●	●	●	○	●
Hooking Location												
On-device	●	●	●	●	●	●	●	●	●	●	●	●
Off-device	○	○	○	○	○	●	○	○	○	○	○	○

Table I. Evaluation of DSU research works based on capabilities and cons

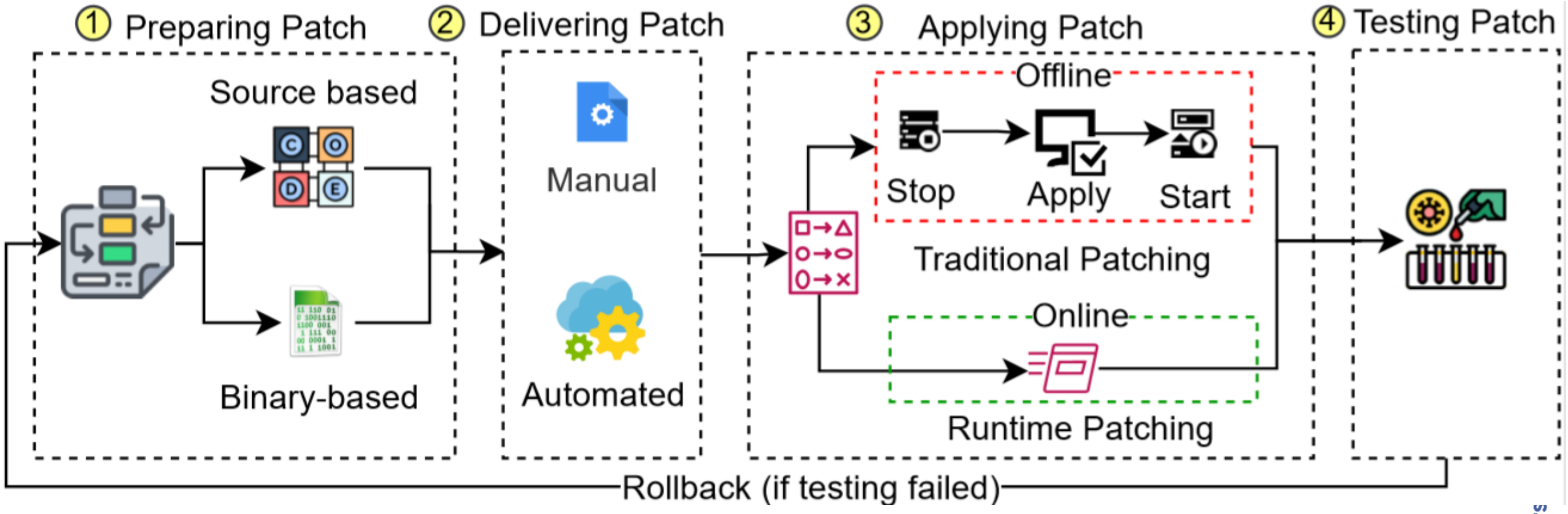
Scope*	Implemented	High level of Abs.	Upd. pre. started code	Simplicity	Consistency
×	×	×	×	✓	✓✓
×	×	×	×	×	×
×	×	×	✓	×	✓
×	×	×	×	✓✓	✓✓
×	×	×	×	✓	✓
×	×	×	×	×	✓
×	×	×	×	?	✓✓
✓	×	×	×	✓✓	×
×	×	×	✓	✓✓	✓✓
×	×	×	×	?	✓
×	×	×	×	✓	×
×	×	×	×	✓✓	✓
×	×	×	×	✓	✓
×	×	×	×	✓	✓✓
×	×	×	✓	✓	✓✓
×	×	×	×	✓	×
×	×	×	×	✓✓	✓

cybersec

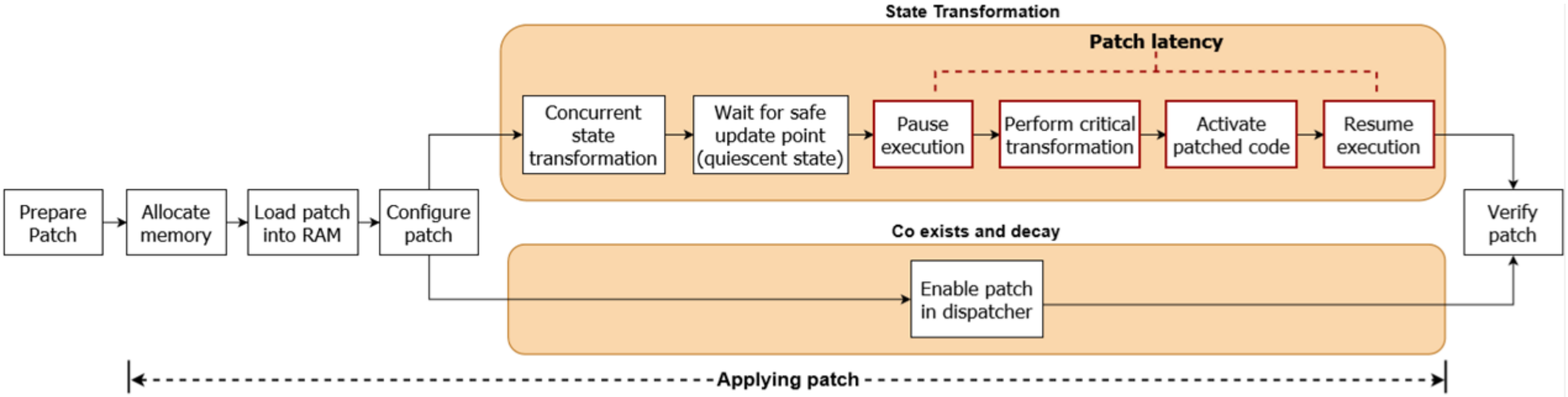
Proposed taxonomy

- **Granularity (What) –** Instruction, Function, Library, Process, Container, VM, Hypervisor, Kernel
- **Strategy – (How and when)**
  - Co-exist & Decay
  - Resource Transformation
- **Responsible entity (Who)**
  - Vendor
  - User
  - Third party

Patch life cycle



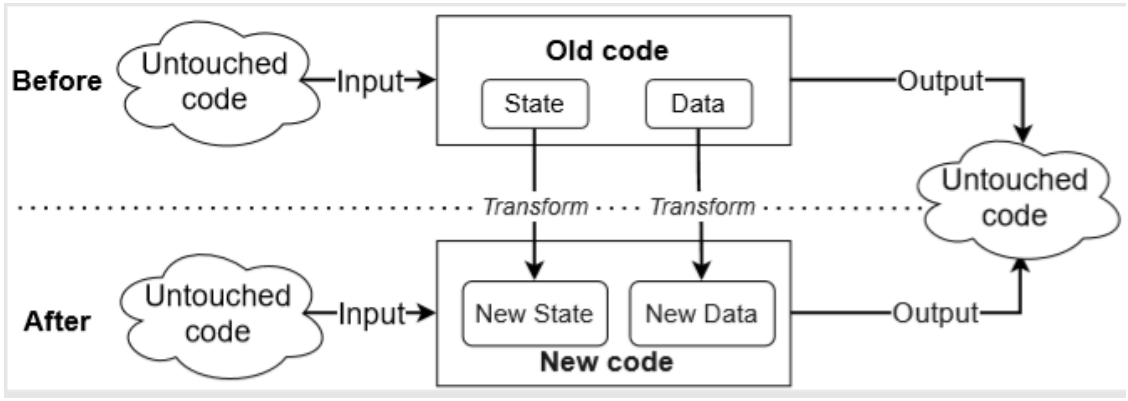
Runtime patch deployment workflow

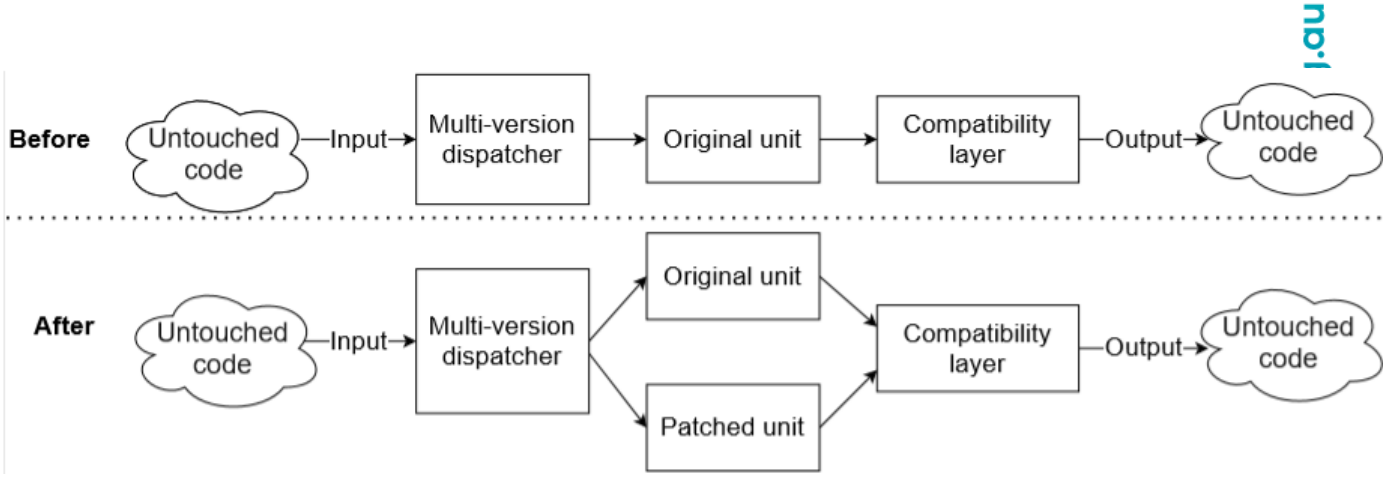


Patch strategies and applicability

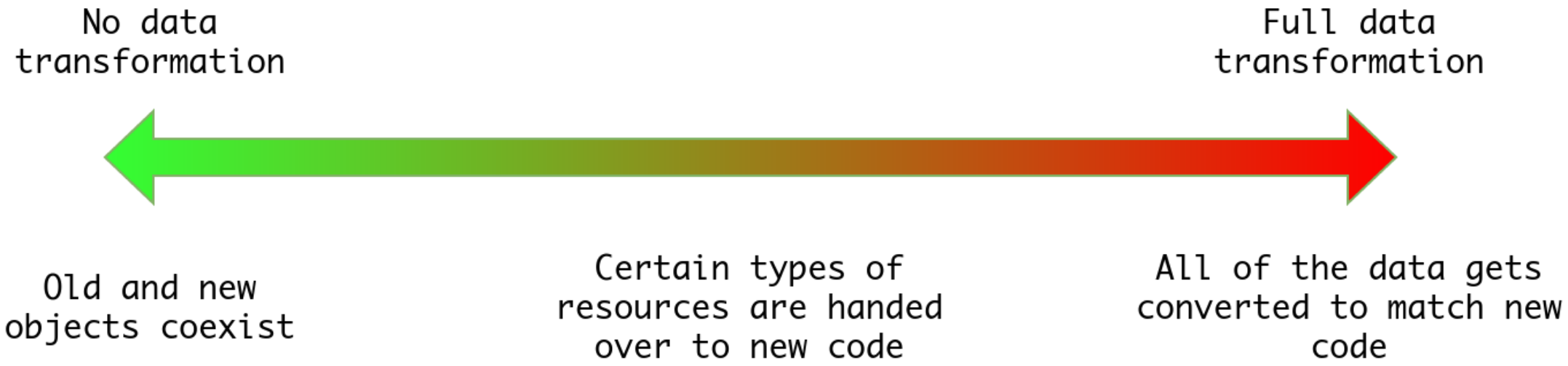
- **State transformation**
  - Pros: no individual user workflow disruptions
  - Cons: slow, not always possible (valid states), manual efforts required, potential service interruptions

- **Co-exist & decay**
  - Pros: no service interruptions
  - Cons: higher overhead (dispatching + cleanup), session or transaction latency



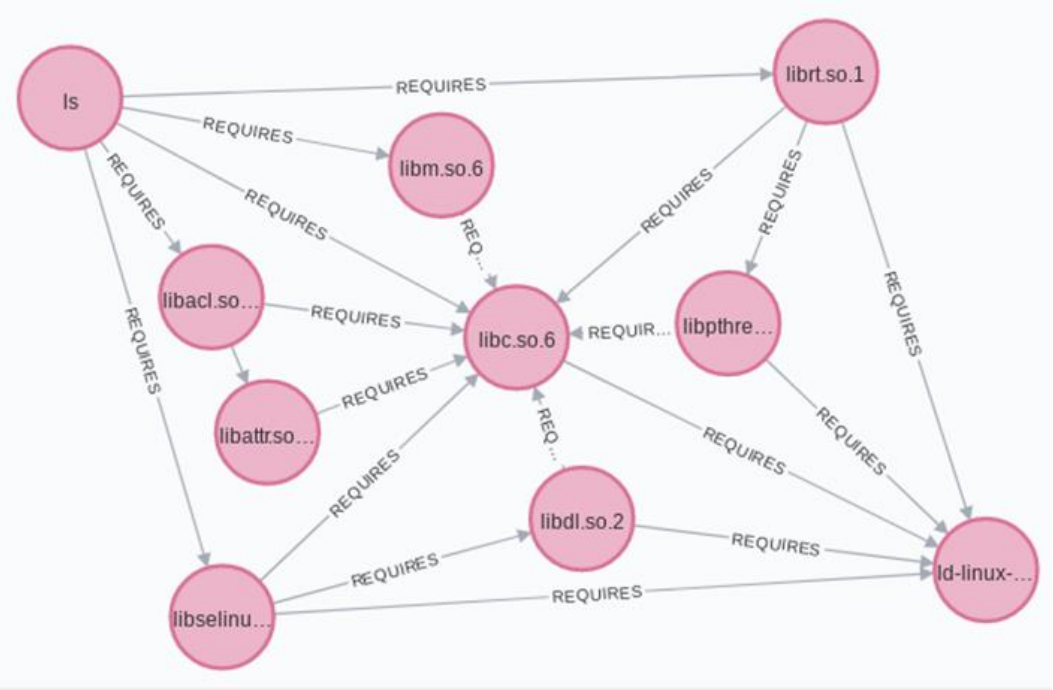


Patch strategies spectrum

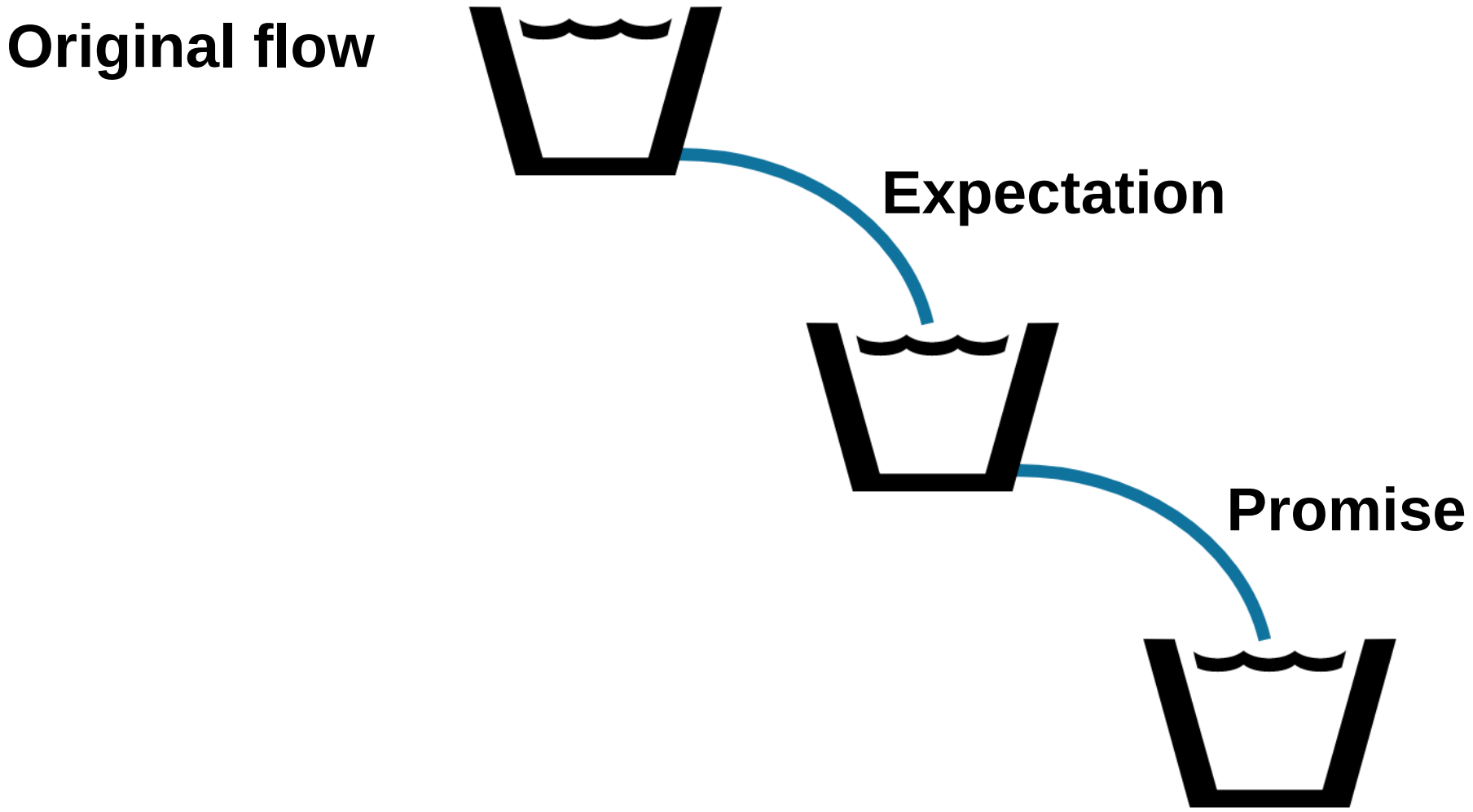


Patch impact estimation

- **Purpose:**
  - Detect potential issues prior to applying a patch
- **Selected granularity:**
  - Executables binaries and libraries
- **“Patchset” considerations**
- **Impact direction**
  - Complexity
  - Popularity

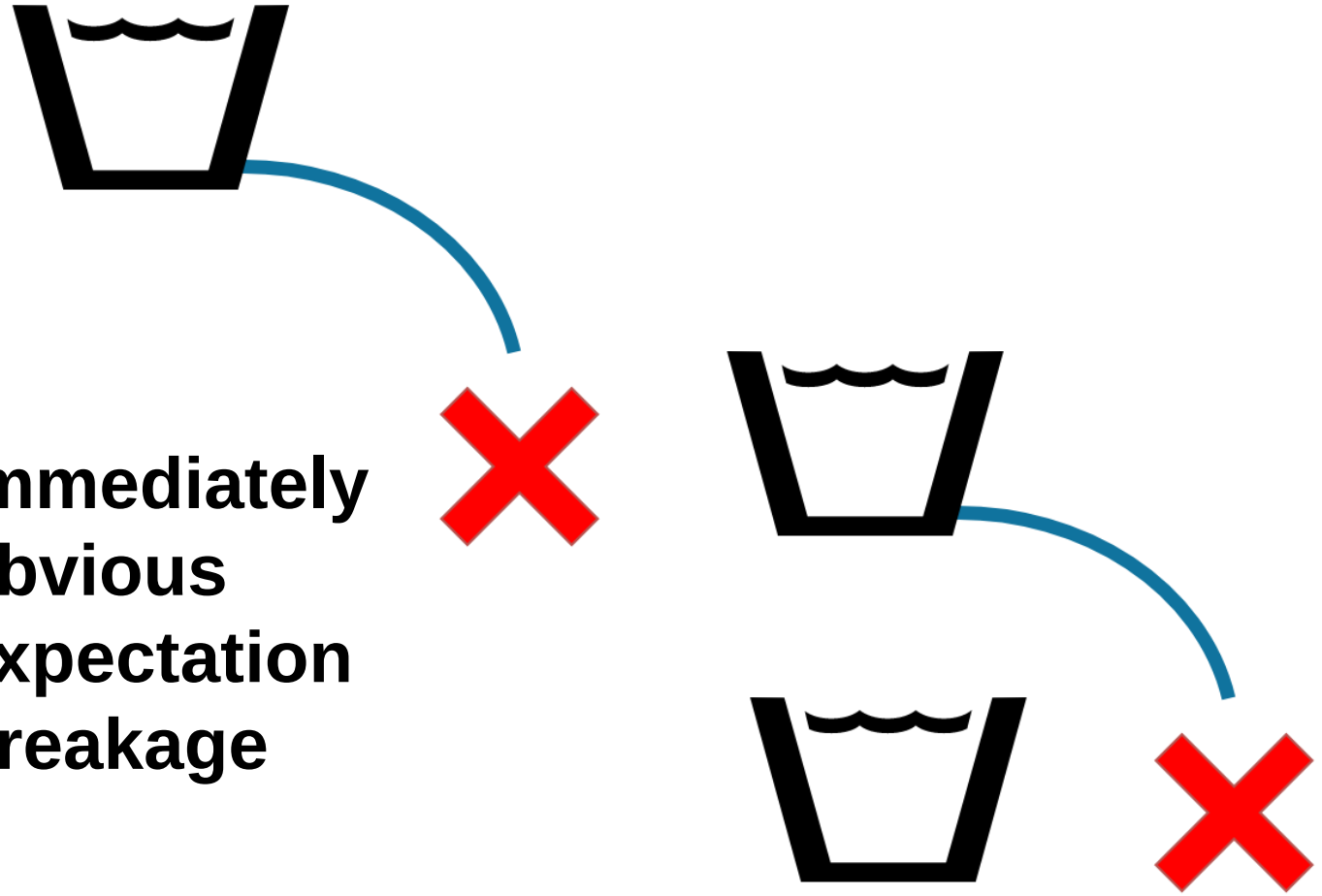


Conceptual patching model



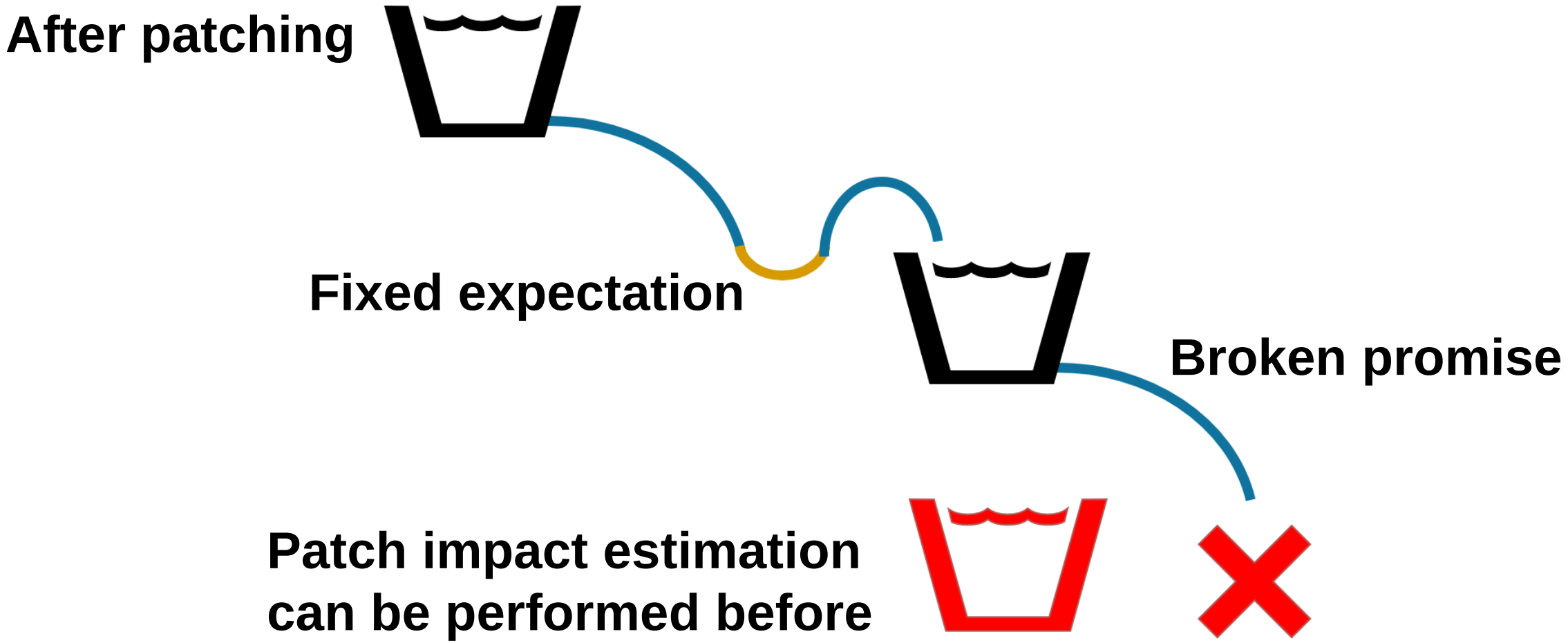
Conceptual patching model

Blind change

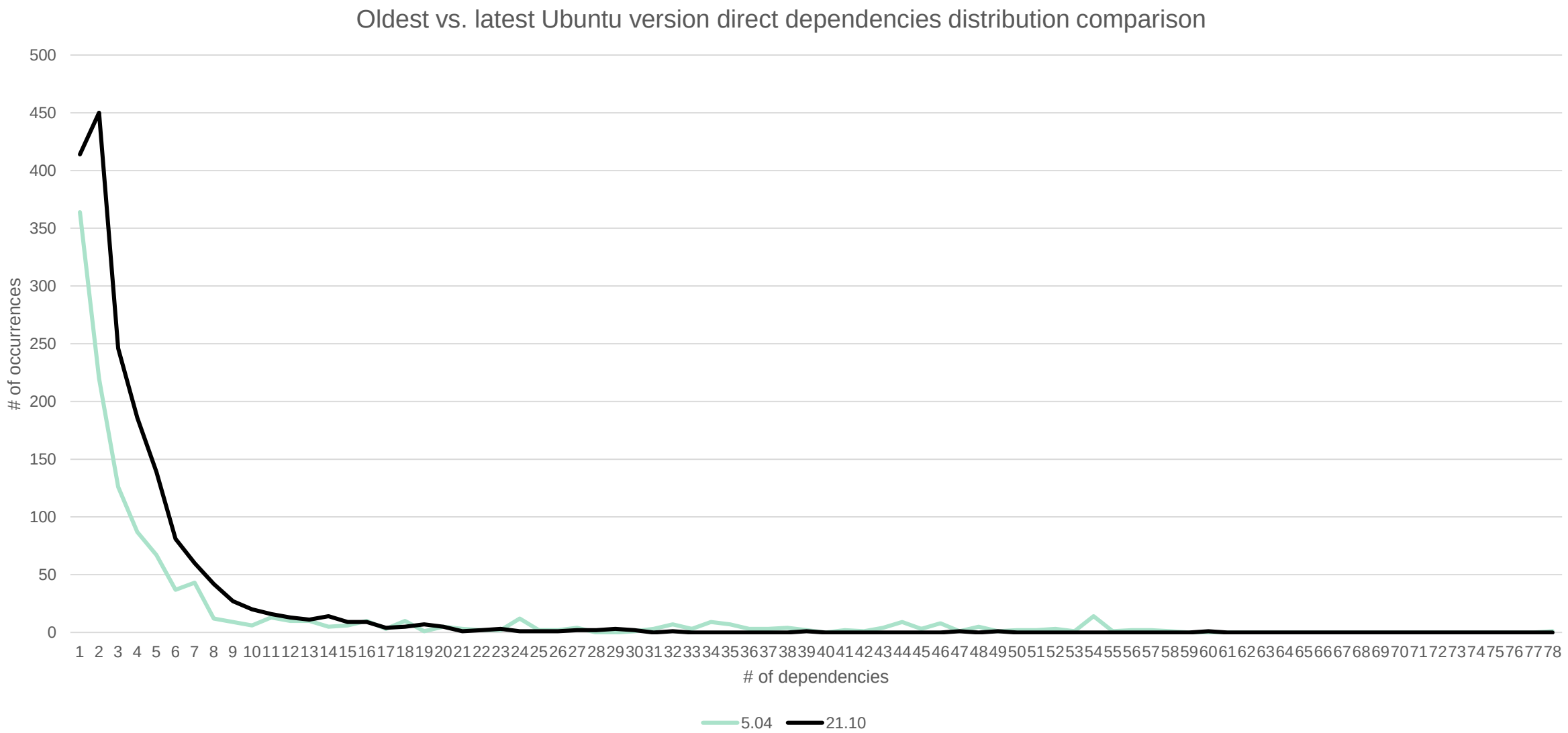


Immediately obvious expectation breakage

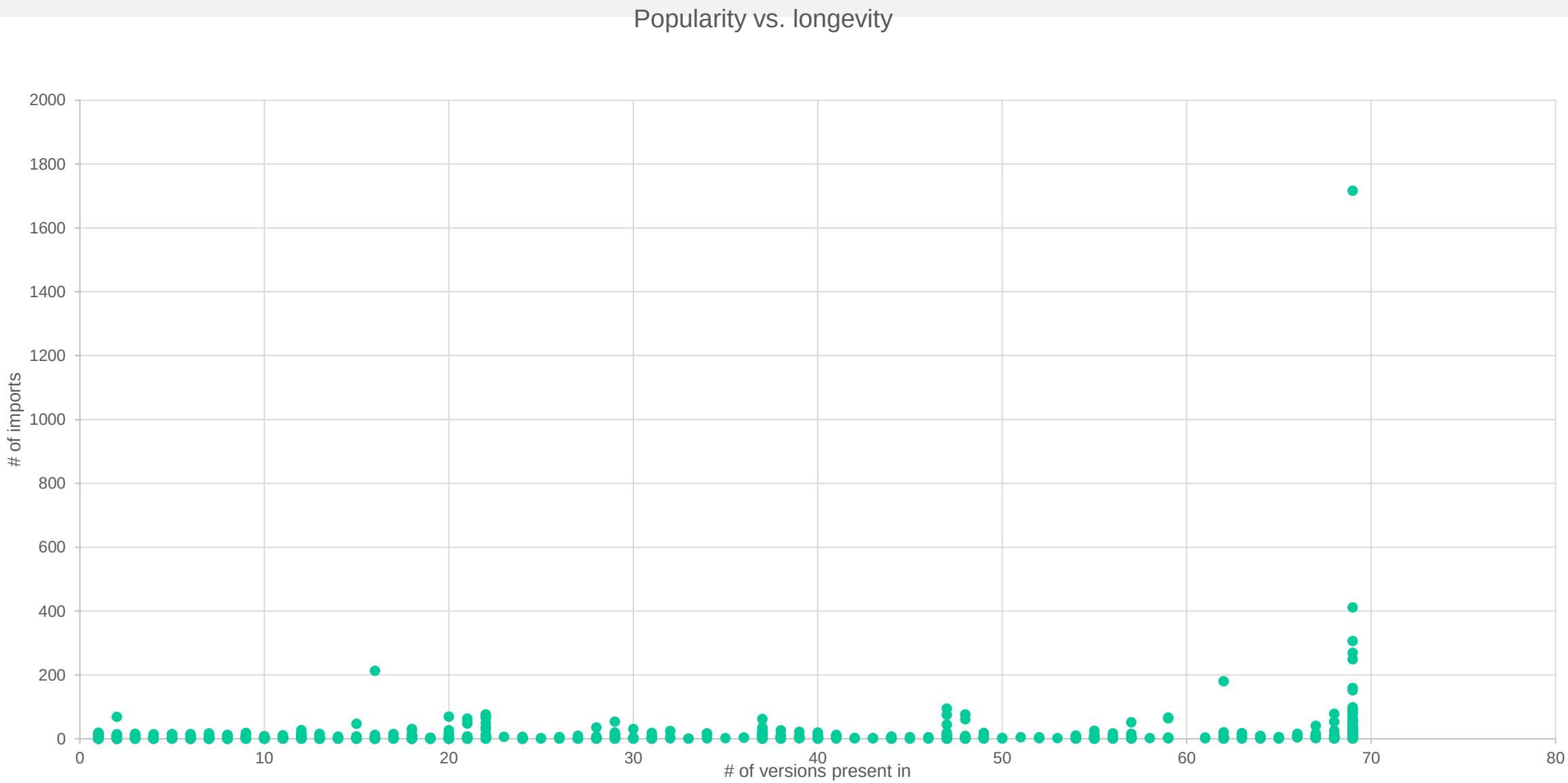
Conceptual patching model



Ubuntu case study: complexity evolution



Ubuntu case study: libraries popularity



cybersecuritycrc.org.au

Ubuntu case study: direct dependencies

ELF binary	Dependencies		First occurrence	Last occurrence		Versions present in	First version	Last version
	Max	Min						
ximian-connector-setup	100	55	55	→	100	4	5.04	6.06.1
evolution	94	52	78		52	22	5.04	11.04
exchange-connector-setup	77	11	77		11	18	6.10	11.04
yelp	76	8	43		8	69	5.04	21.10
rhythmbox	75	6	48		6	67	5.04	21.10
totem	72	7	58		7	62	5.04	21.10
sound-juicer	71	38	38	→	52	12	5.04	8.04.4
evince	70	16	68		16	68	5.10	21.10
totem-video-thumbnailer	69	8	56		8	62	5.04	21.10
evince-thumbnailer	65	7	63		7	68	5.10	21.10

Ubuntu case study: all dependencies

ELF binary	Dependencies		First occurrence	Last occurrence	Versions present in	First version	Last version
	Max	Min					
gnome-control-center	288	47	59	280	52	5.04	21.10
gnome-calendar	169	98	98	157	28	16.04	21.10
gnome-todo	169	146	159	156	18	18.04	21.10
rhythmbox	160	59	59	112	67	5.04	21.10
empathy	159	108	116	159	27	9.10	15.10
empathy-accounts	154	76	117	154	26	10.04	15.10
empathy-debugger	154	61	117	154	26	10.04	15.10
totem	153	67	67	118	62	5.04	21.10
net	152	18	18	152	37	5.04	18.04.4
gnome-shell	148	135	148	140	20	17.10	21.10

Ubuntu case study: libraries popularity

ELF library	Dependencies		First occurrence	Last occurrence	Versions present in	First version	Last version
	Max	Min					
libc.so	2204	1177	1177	1777	69	5.04	21.10
libpthread.so	649	113	242	113	69	5.04	21.10
libdl.so	397	56	290	56	69	5.04	21.10
libm.so	385	174	360	189	69	5.04	21.10
libglib.so	349	235	235	311	69	5.04	21.10
libgobject.so	302	206	206	278	69	5.04	21.10
libX11.so	269	119	130	122	69	5.04	21.10
librt.so	267	5	78	5	69	5.04	21.10
libgio.so	229	32	32	214	62	8.04	21.10
libz.so	215	67	215	78	69	5.04	21.10

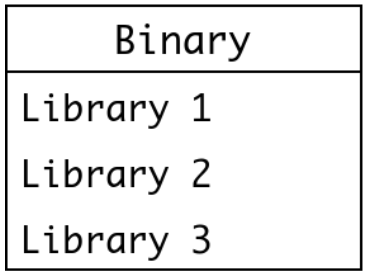
Current work: multi-language impact

- **Python**
  - import, from … import …, os.system(“python…”), execfile, __import__
- **Bash**
  - source, . , bash, …sh
- **PHP**
  - include, require, include_once, require_once, php.ini opcache.preload/auto_append_file/auto_prepend_file
- **ELF/PE binaries**
  - Imported functions (Windows and Linux), syscalls (Linux-only)

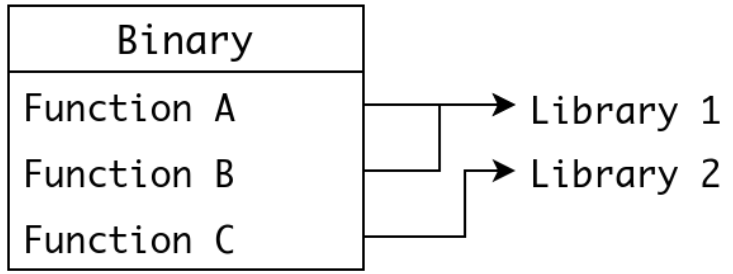
Future steps: patch impact metrics

- **Presence:** 33% each
- **Coverage:** Library1 is twice as “important”
- **Occurrence:** Library2 has 60% (3 out of 5) of calls
- **Usage:** highly depends on value of X and “condition” (only available at runtime).

Presence



Coverage



Occurrence

(static code)

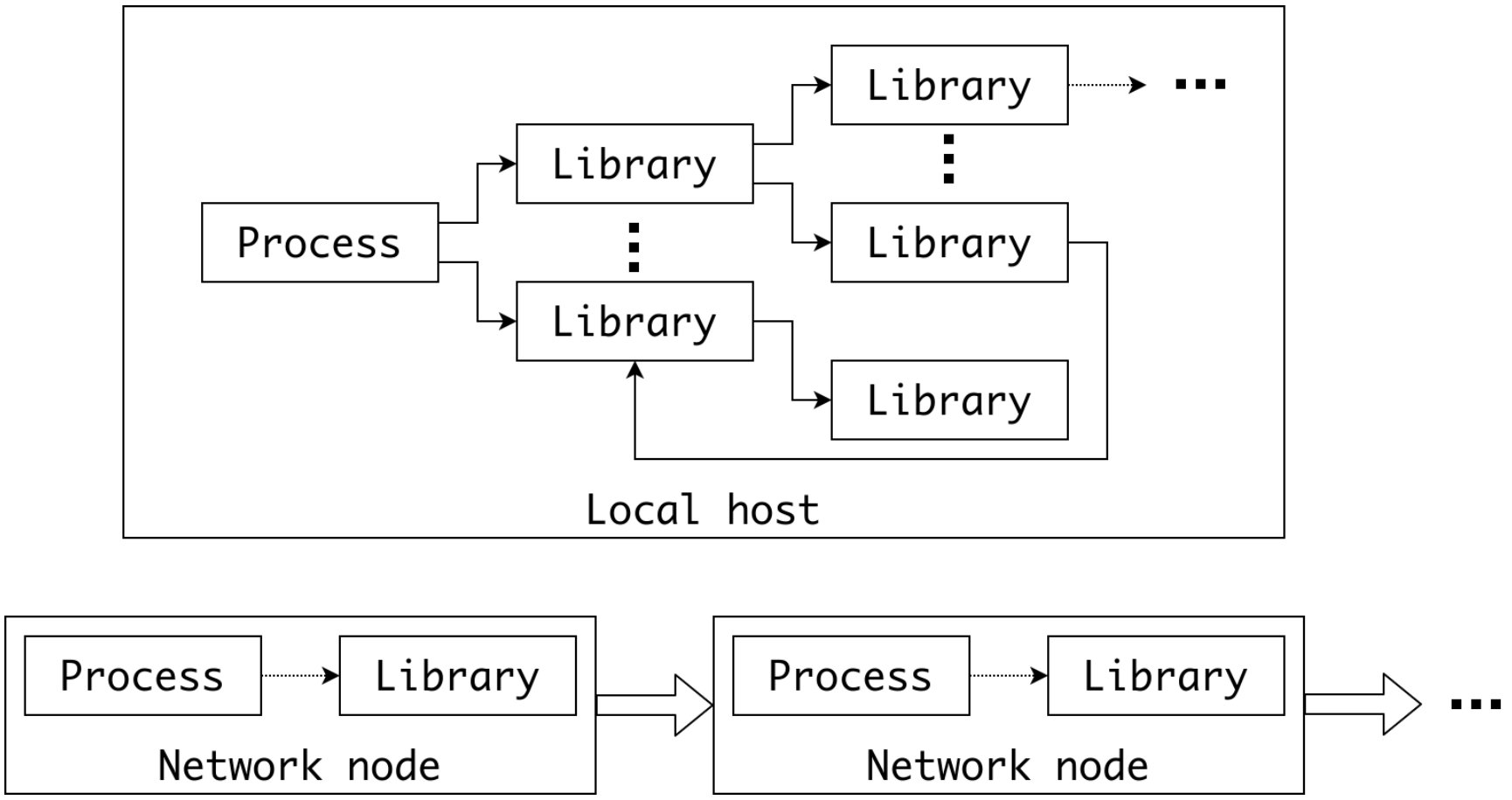
```
...
FunctionA()
...
FunctionC()
...
FunctionC()
FunctionC()
...
FunctionB()
...
```

Usage

(runtime calls)

```
...
Loop X times:
  FunctionA()
...
If condition:
  FunctionC()
...
FunctionB()
...
```

Future steps: network-level impact



CYBER SECURITY COOPERATIVE RESEARCH CENTRE

CREST
Centre for Research on Engineering Software Technologies

THE UNIVERSITY of ADELAIDE

Thanks for your attention

cybersecuritycrc.org.au