Type-Safe Dynamic Placement with First-Class Placed Values

George Zakhour

Pascal Weisenburger

Guido Salvaneschi

University of St.Gallen

Programming Group

Dynamic Placement

Dynamic Placement

- Caching
- Geo-Replication
- Hybrid Cloud
- High-Performance Computing

Safe Placement

Safe Placement and Static Checking

Type-Safe Distributed Programming with ML5*

Tom Murphy VII, Karl Crary, and Robert Harper

Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA
{tom7,crary,rwh}@cs.cmu.edu

language for spa-
of ML, allows an
soned about as a
bility of any kind
gic, statically ex-
he ML5 compiler
ved in the com-
ifferent resources
iler and runtime
ing: a distributed
veb server.

A Formal Theory of Choreographic Programming

Luís Cruz-Filipe[1] · Fabrizio Montesi[1] · Marco Peressotti[1]

Received: 7 September 2022 / Accepted: 9 April 2023 / Published online: 27 May 2023

Abstract
Choreographic programming is a paradigm for writing coordination plans for distributed systems from a global point of view, from which correct-by-construction decentralised implementations can be generated automatically. Theory of choreographies typically includes a number of complex results that are proved by structural induction. The high number of cases and the subtle details in some of these proofs has led to important errors being found in published works. In this work, we formalise the theory of a choreographic programming language in Coq. Our development include
its Turing completeness, a compilation pro
characterisation of the correctness of this p
the benefits of using a theorem prover: we
the mechanised proof, and a significant si
offer a foundation for the future formal de

Links: Web Programming Without Tiers*

Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop

University of Edinburgh

ArchJava: Connecting Software Architecture to Implementation

Jonathan Aldrich Craig Chambers David Notkin

Department of Computer Science and Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350 USA
+1 206 616-1846
{jonal, chambers, notkin}@cs.washington.edu

Abstract
Software architecture describes the structure of a system, enabling more effective design, program understanding, and formal
ple implementation
istencies, causing
s, and inhibiting
sion to Java that
h implementation,
s to architectural
to a circuit-design
press architectural
and that it can aid
on.

language. Thus, it may be difficult to trace architectural features to the implementation, and the implementation may become inconsistent with the architecture as the program evolves. In summary, while architectural analysis in existing ADLs may reveal important architectural properties, these properties are not guaranteed to hold in the implementation.

In order to enable architectural reasoning about an

Fabric: A Platform for Secure Distributed Computation and Storage

Jed Liu Michael D. George K. Vikram
Xin Qi Lucas Waye Andrew C. Myers
{liujed,mdgeorge,kvikram,qixin,lrw,andru}@cs.cornell.edu
Department of Computer Science
Cornell University
4130 Upson Hall, Ithaca NY

Abstract
Fabric is a new system and language for building secure distributed information systems. It is a decentralized system that allows heterogeneous network nodes to securely share both information and computation resources despite mutual distrust. Its high-level programming language makes distribution and persistence largely transparent to programmers. Fabric supports data-shipping and function-shipping styles of computation: both computation and information can move between nodes to meet security requirements or to improve performance. Fabric provides a rich, Java-like object model, but data resources are labeled with confidentiality and integrity poli-
-time and
nsure con-
emination
sults from
lean, con-
enforces

does not offer general, principled techniques for implementing the functionality of these systems while also satisfying their security and privacy requirements. This lack motivates the creation of Fabric, a platform for building secure distributed information systems.

It is particularly difficult to build secure *federated* systems, which integrate information and computation from independent administrative domains—each domain has policies for security and privacy, but does not fully trust other domains to enforce them. Integrating information from different domains is important because it enables new services and capabilities.

To illustrate the challenges, consider the scenario of two medical institutions that want to securely and quickly share patient information. This goal is important: according to a 1999 Institute of Medicine study, at least 44,000 deaths annually result from medical errors, with incomplete patient information identified as a leading cause [25]. However, automated sharing of patient data poses difficulties. First, the security and privacy policies of the two institutions must be satisfied (as mandated by HIPAA [22] in the U.S.), restricting which information can be shared or modified by the two institu-

Distributed System Development with ScalaLoci

PASCAL WEISENBURGER, Technische Universität Darmstadt, Germany
MIRKO KÖHLER, Technische Universität Darmstadt, Germany
GUIDO SALVANESCHI, Technische Universität Darmstadt, Germany

Distributed applications are traditionally developed as separate modules, often in different languages, which react to events, like user input, and in turn produce new events for the other modules. Separation into components requires time-consuming integration. Manual implementation of communication forces programmers to deal with low-level details. The combination of the two results in obscure distributed data flows scattered among multiple modules, hindering reasoning about the system as a whole.

The ScalaLoci distributed programming language addresses these issues with a coherent model based on placement types that enables reasoning about distributed data flows, supporting multiple software architectures via dedicated language features and abstracting over low-level communication details and data conversions. As we show, ScalaLoci simplifies developing distributed systems, reduces error-prone communication code and favors early detection of bugs.

Ur/Web: A Simple Model for Programming the Web

Adam Chlipala
MIT CSAIL
adamc@csail.mit.edu

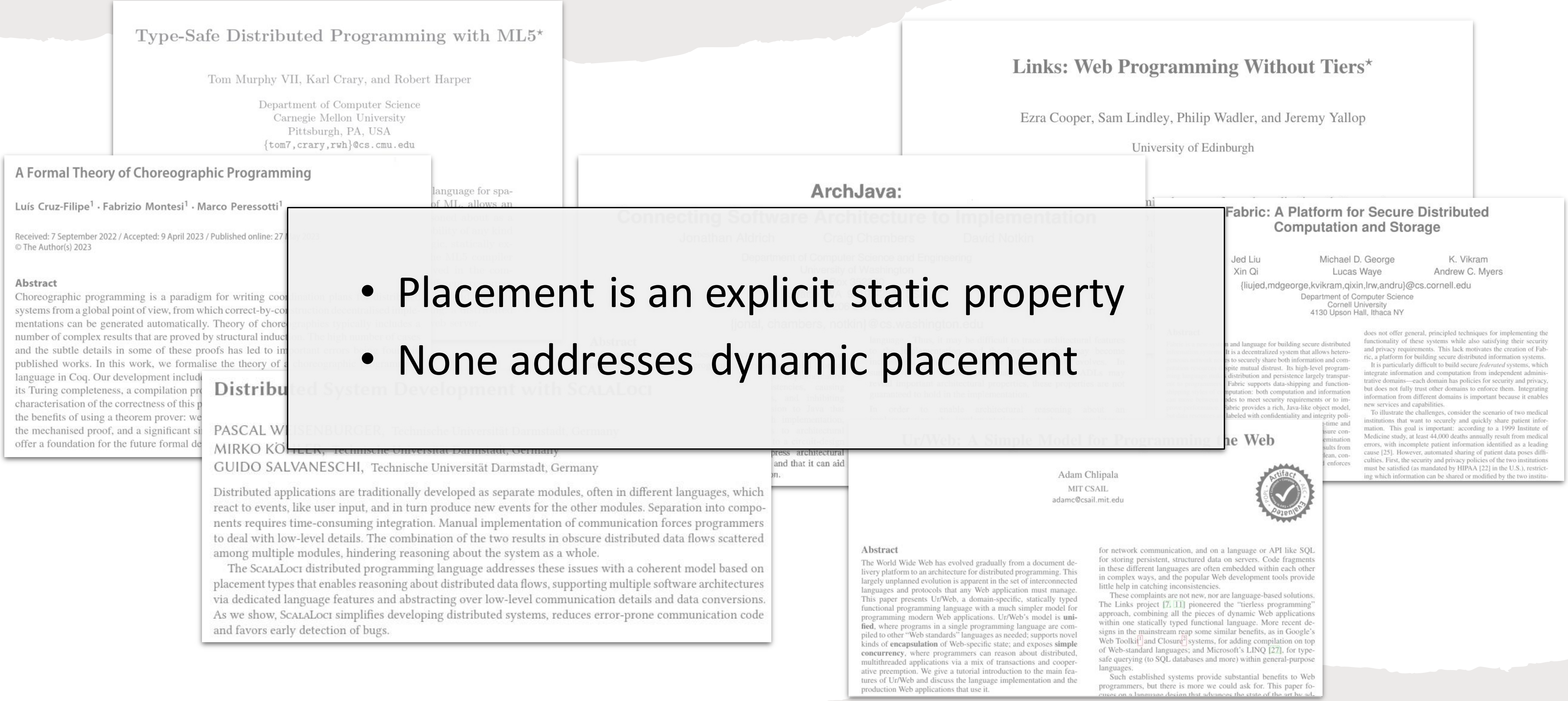
Abstract
The World Wide Web has evolved gradually from a document delivery platform to an architecture for distributed programming. This largely unplanned evolution is apparent in the set of interconnected languages and protocols that any Web application must manage. This paper presents Ur/Web, a domain-specific, statically typed functional programming language with a much simpler model for programming modern Web applications. Ur/Web's model is **unified**, where programs in a single programming language are compiled to other "Web standards" languages as needed; supports novel kinds of **encapsulation** of Web-specific state; and exposes **simple concurrency**, where programmers can reason about distributed, multithreaded applications via a mix of transactions and cooperative preemption. We give a tutorial introduction to the main features of Ur/Web and discuss the language implementation and the production Web applications that use it.

for network communication, and on a language or API like SQL for storing persistent, structured data on servers. Code fragments in these different languages are often embedded within each other in complex ways, and the popular Web development tools provide little help in catching inconsistencies.

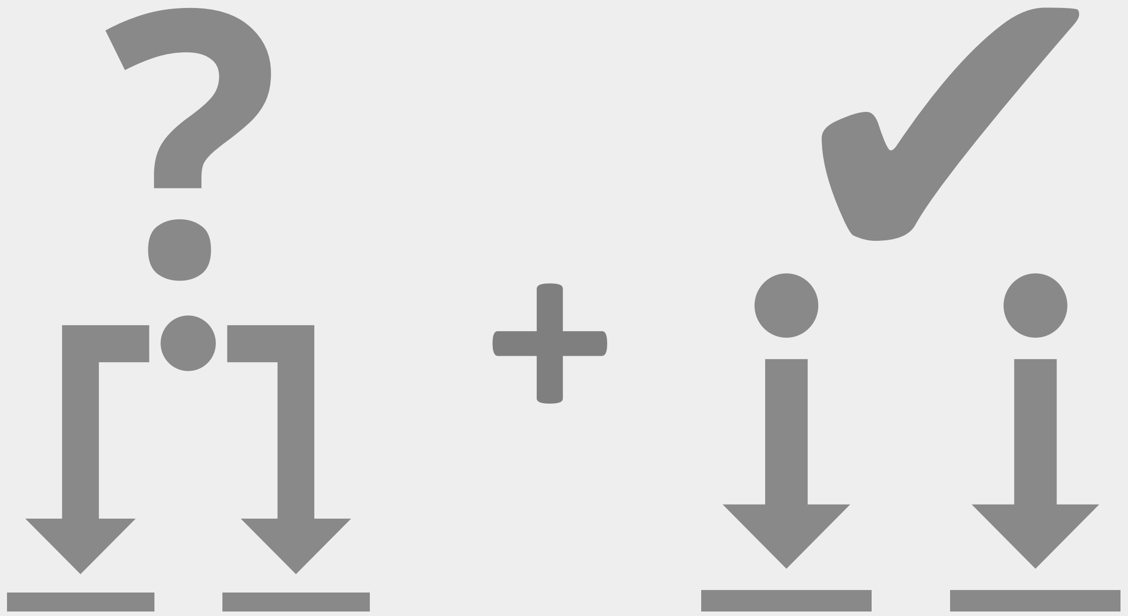
These complaints are not new, nor are language-based solutions. The Links project [7, 11] pioneered the "tierless programming" approach, combining all the pieces of dynamic Web applications within one statically typed functional language. More recent designs in the mainstream reap some similar benefits, as in Google's Web Toolkit[1] and Closure[2] systems, for adding compilation on top of Web-standard languages; and Microsoft's LINQ [27], for type-safe querying (to SQL databases and more) within general-purpose languages.

Such established systems provide substantial benefits to Web programmers, but there is more we could ask for. This paper fo-

Safe Placement and Static Checking

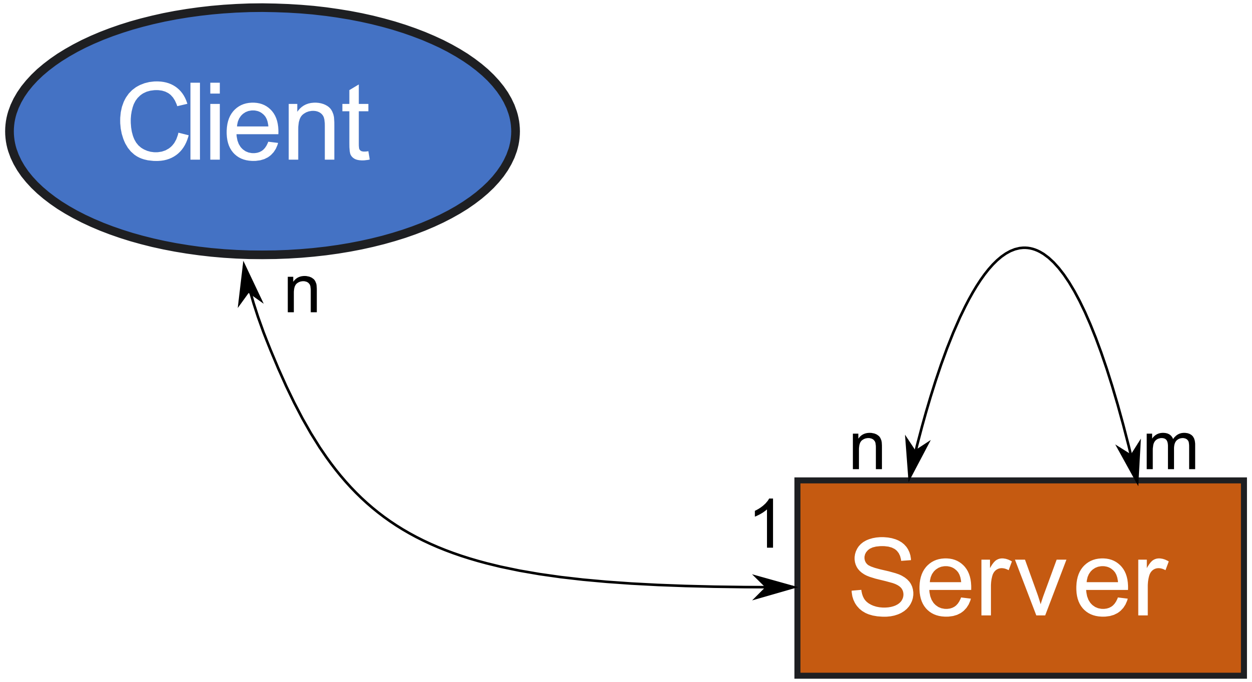


- Placement is an explicit static property
- None addresses dynamic placement



Achieving Safe Dynamic Placement with Placement Types

What are Placement Types?



What are Placement Types?

```
// The architecture is:
@peer type Client <: { type Tie <: Single[Server] }
@peer type Server <: { type Tie <: Multiple[Client] with Multiple[Server] }

// All clients have this String variable
var message: String on Client = "Greetings Earthlings"

// All servers can fetch the messages of their connected clients
def hashMessages(seed: Int): List[String] on Server =
  on[Client].run.capture(seed) {
    hash(message, seed)
  }.asLocalFromAll
```

A Principled Approach to Placement Safety

Re-Interpreting Type System Features as Placements

Interpreting Types as Placements

Type System		Placement System
Types *Int, Boolean, String*	⟷	**Placement Types** *Client, Server, Database*
Param. Polymorphism *Map[K,V], Set[T], Either[S,T]*	⟷	**Place-Polymorphic Modules** *LeaderElection[P], DistHashMap[P]*
?	⟷	**Dynamic Placement** *DB|Cache, CPU|GPU, LocalFS|RemoteFS*

[OOPSLA 2018]

[ECOOP 2019]

Interpreting Types as Placements

Type System		Placement System
Types *Int, Boolean, String*	⟷	Placement Types *Client, Server, Database*
Param. Polymorphism *Map[K,V], Set[T], Either[S,T]*	⟷	Place-Polymorphic Modules *LeaderElection[P], DistHashMap[P]*
Union Types *Peano Numbers, Linked Lists, Options*	⟷	Dynamic Placement *DB|Cache, CPU|GPU, LocalFS|RemoteFS*

[OOPSLA 2018]


Distributed System Development with ScalaLoci

[ECOOP 2019]


Multitier Modules

[OOPSLA 2023]


Type-Safe Dynamic Placement with First-Class Placed Values

Union Placement Types Statically Capture Placement Uncertainty

Solution: Union Placement Types

- First-class placed type: Data at (P | Q)
- Static uncertainty and dynamic certainty
- Eliminate runtime checks when access is safe

Solution: Union Placement Types

- Remote references inhabit placement unions

- Introduced with remote ref

```
def readFromCache(k: Key): (Data at Cache) on Cache =
  remote ref query(k)
```

- Eliminated with deref – typechecks if the architecture allows it

```
readFromCache("splash23").deref : Future[Value]
```

SAFE

Solution: Union Placement Types

- Introduced via subsumption `T at P <: T at (P | Q)`

```
def readFromCache(k: Key): Data at (DB | Cache) on Cache =
  remote ref query(k)
```

- Eliminated with `toEither[P, Q]`

```
data.toEither[Cache, DB] : Either[Data at Cache, Data at DB]
```

```
oString()
tsWith("file://")) {
 never happen
oid.os.FileUriExpose
```

Evaluation: URLs as References

Evaluation: URLs as References

```java
    public static void getBackgroundImage(ImageView imageView) {
        String backgroundUrl = PreferenceData.BACKGROUND_IMAGE.getValue(imageView.getContext());

        if (backgroundUrl != null && backgroundUrl.length() > 0) {
            if (backgroundUrl.startsWith("http"))
                Glide.with(imageView.getContext()).load(backgroundUrl).into(imageView);
            else if (backgroundUrl.contains("://")) {
                if (backgroundUrl.startsWith("content://")) {
                    String path = Uri.parse(backgroundUrl).getLastPathSegment();
                    if (path != null && path.contains(":"))
                        path = "/storage/" + path.replaceFirst(":", "/");
                    else path = Uri.parse(backgroundUrl).getPath();

                    //      "a haiku"
                    //I don't like storage
                    //I'm sorry, poor developer
                    //this is all my fault
                    //          - james fenn, 2018

                    Glide.with(imageView.getContext()).load(new File(path)).into(imageView);
                } else Glide.with(imageView.getContext()).load(Uri.parse(backgroundUrl)).into(imageView);
            } else Glide.with(imageView.getContext()).load(new File(backgroundUrl)).into(imageView);
        }
    }
```

https://github.com/fennifith/Alarmio/blob/main/app/src/main/java/me/jfenn/alarmio/utils/ImageUtils.java

Evaluation: URLs as References

```
    public static void getBackgroundImage(ImageView imageView) {
        String backgroundUrl = PreferenceData.BACKGROUND_IMAGE.getValue(imageView.getContext());

        if (bac
            if (

            els
```

```
//       "a haiku"
//I don't like storage
//I'm sorry, poor developer
//this is all my fault
//              - james fenn, 2018
```

```
                //
                //          - james fenn, 2018

                Glide.with(imageView.getContext()).load(new File(path)).into(imageView);
            } else Glide.with(imageView.getContext()).load(Uri.parse(backgroundUrl)).into(imageView);
        } else Glide.with(imageView.getContext()).load(new File(backgroundUrl)).into(imageView);
    }
}
```

https://github.com/fennifith/Alarmio/blob/main/app/src/main/java/me/jfenn/alarmio/utils/ImageUtils.java

Evaluation: URLs as References

```
public static void getBackgroundImage(ImageView imageView) {
    String backgroundUrl = PreferenceData.BACKGROUND_IMAGE.getValue(imageView.getContext());

    if (backgroundUrl != null && backgroundUrl.length() > 0) {
        if (backgroundUrl.startsWith("http"))
            Glide.with(imageView.getContext()).load(backgroundUrl).into(imageView);
        else if (backgroundUrl.contains("://")) {
            if (backgroundUrl.startsWith("content://")) {
                String path = Uri.parse(backgroundUrl).getLastPathSegment();
                if (path != null && path.contains(":"))
                    path = "/storage/" + path.replaceFirst(":", "/");
                else path = Uri.parse(backgroundUrl).getPath();

                //      "a haiku"
                //I don't like storage
                //I'm sorry, poor developer
                //this is all my fault
                //          - james fenn, 2018

                Glide.with(imageView.getContext()).load(new File(path)).into(imageView);
            } else Glide.with(imageView.getContext()).load(Uri.parse(backgroundUrl)).into(imageView);
        } else Glide.with(imageView.getContext()).load(new File(backgroundUrl)).into(imageView);
    }
}
```

https://github.com/fennifith/Alarmio/blob/main/app/src/main/java/me/jfenn/alarmio/utils/ImageUtils.java

Evaluation: URLs as References

133 out of ~3K Android apps encode dynamic placement as URLs.

(F-Droid)

```
if (repo.getAddress().startsWith("content://") || repo.getAddress().startsWith("file://")) {
  // no need to show a QR Code, it is not shareable
  return; }
```

(VLC)

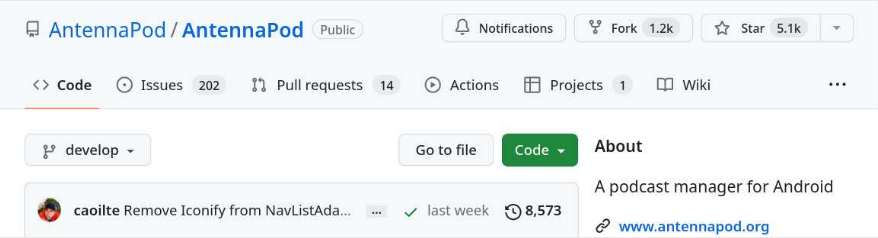
```
public MediaWrapper getMedia(Uri uri) {
  if ("content".equals(uri.getScheme())) return null;
```

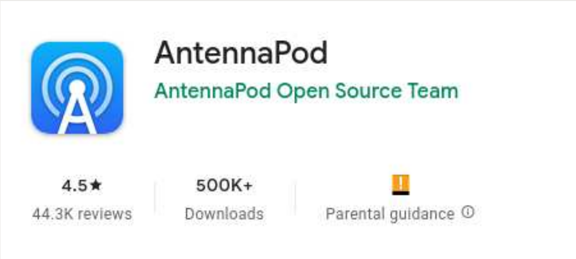
(Element.io)

```
value = uri.toString()
if (value.startsWith("file://")) {
  // it should never happen
  // else android.os.FileUriExposedException will be triggered.
  // see https://github.com/vector-im/riot-android/issues/1725
  return }
```

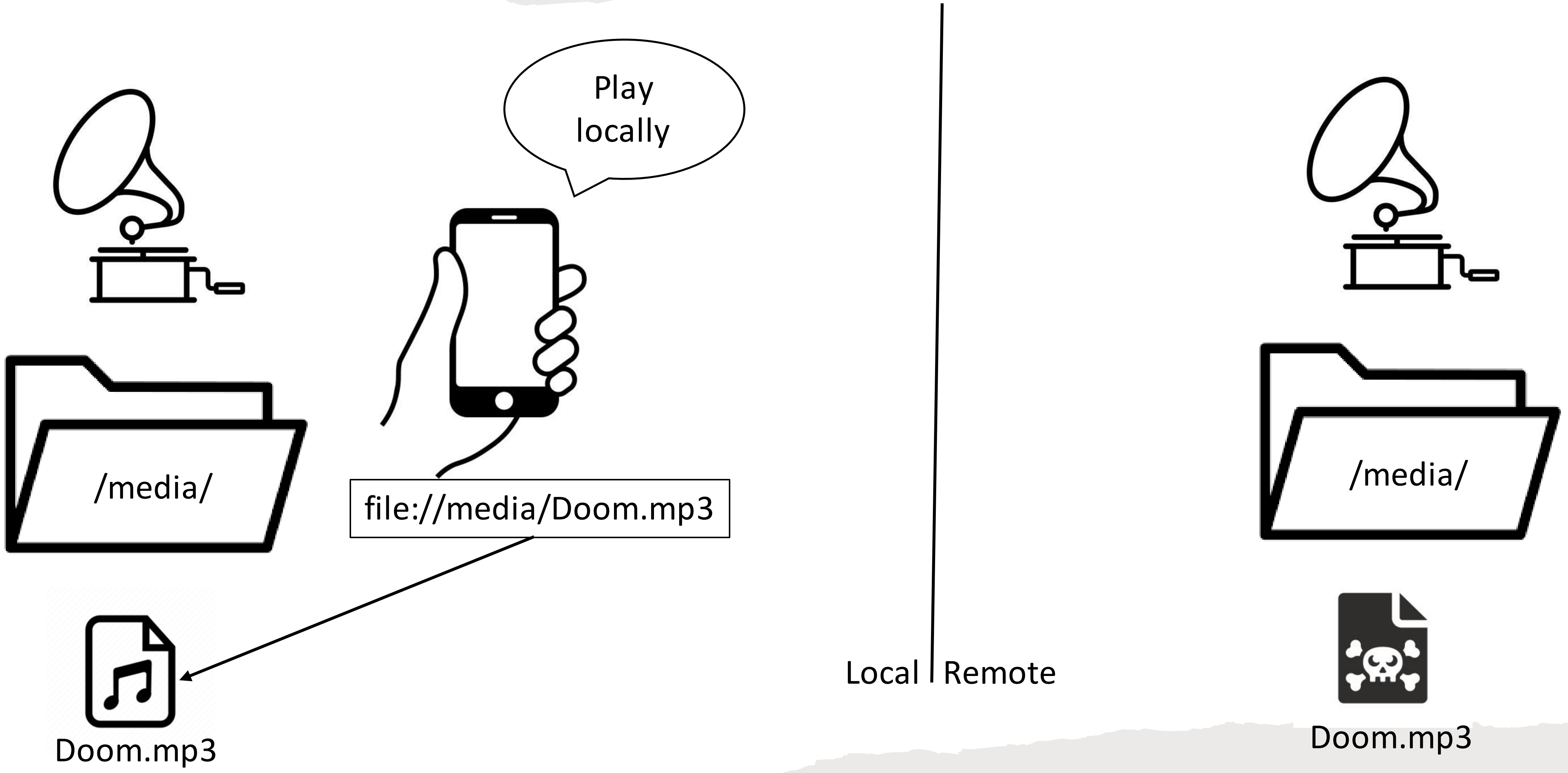
Evaluation: AntennaPod

- 5.1k ★
- 1.2k forks
- 500K+ downloads on the Play Store
- ~55.5 KLoC

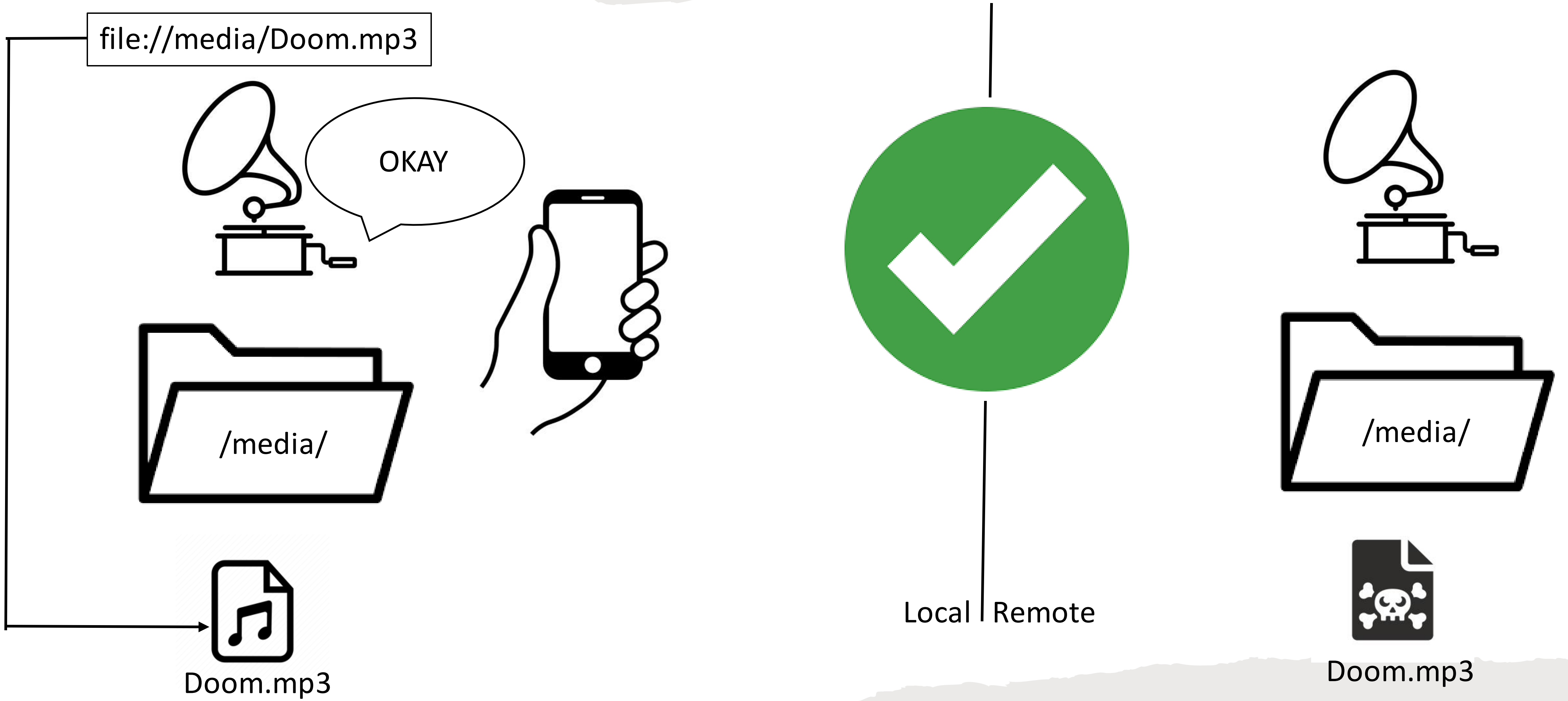




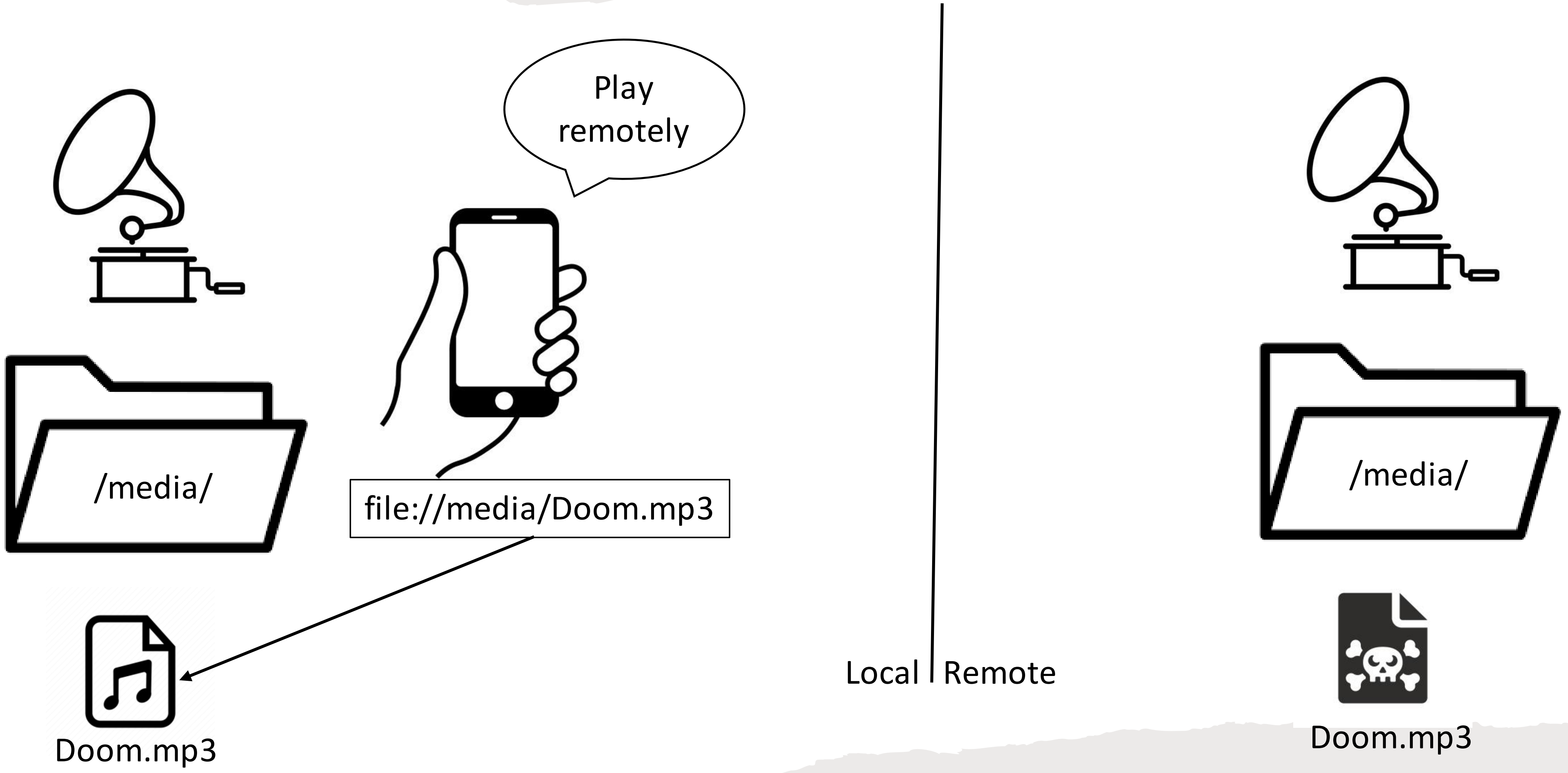
Evaluation: AntennaPod



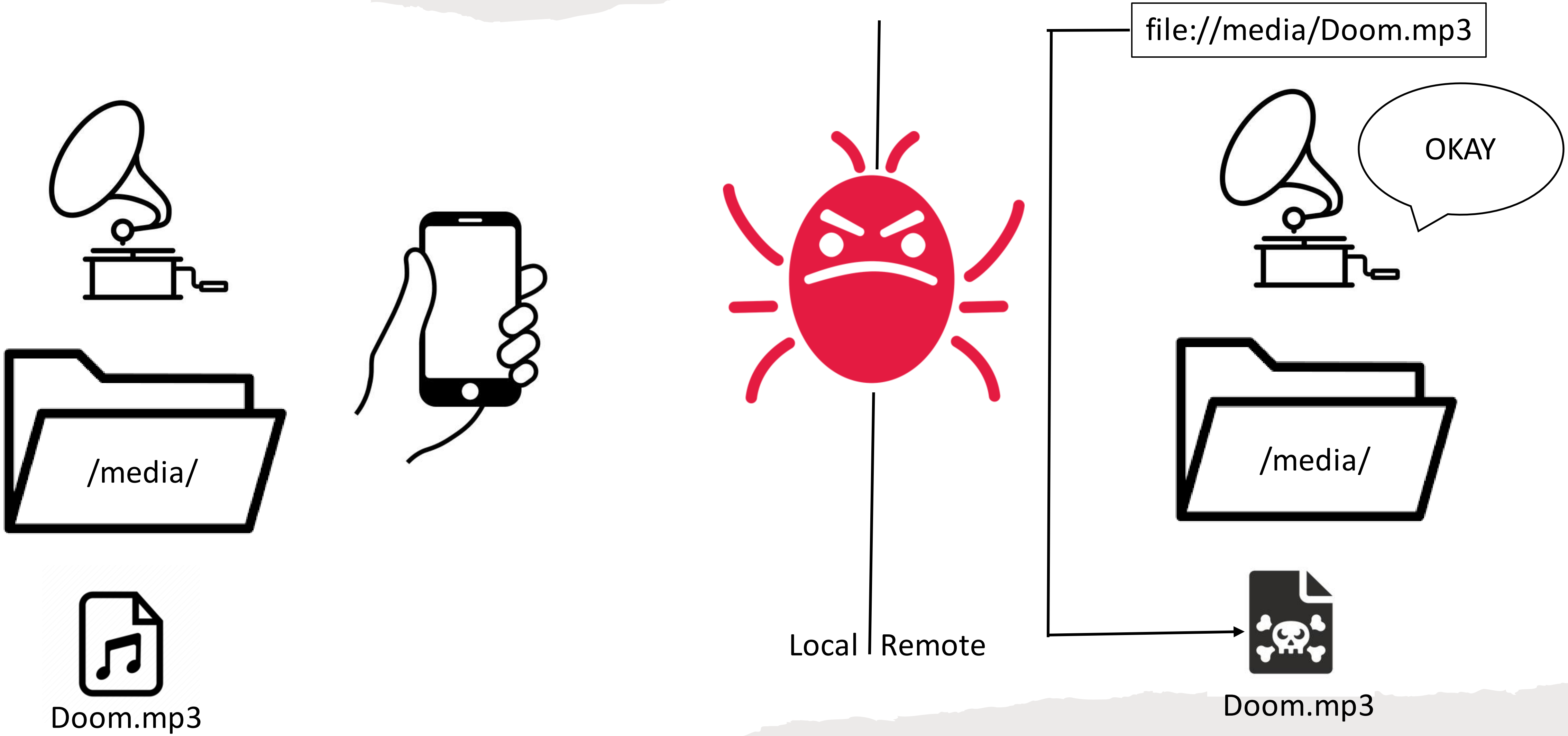
Evaluation: AntennaPod



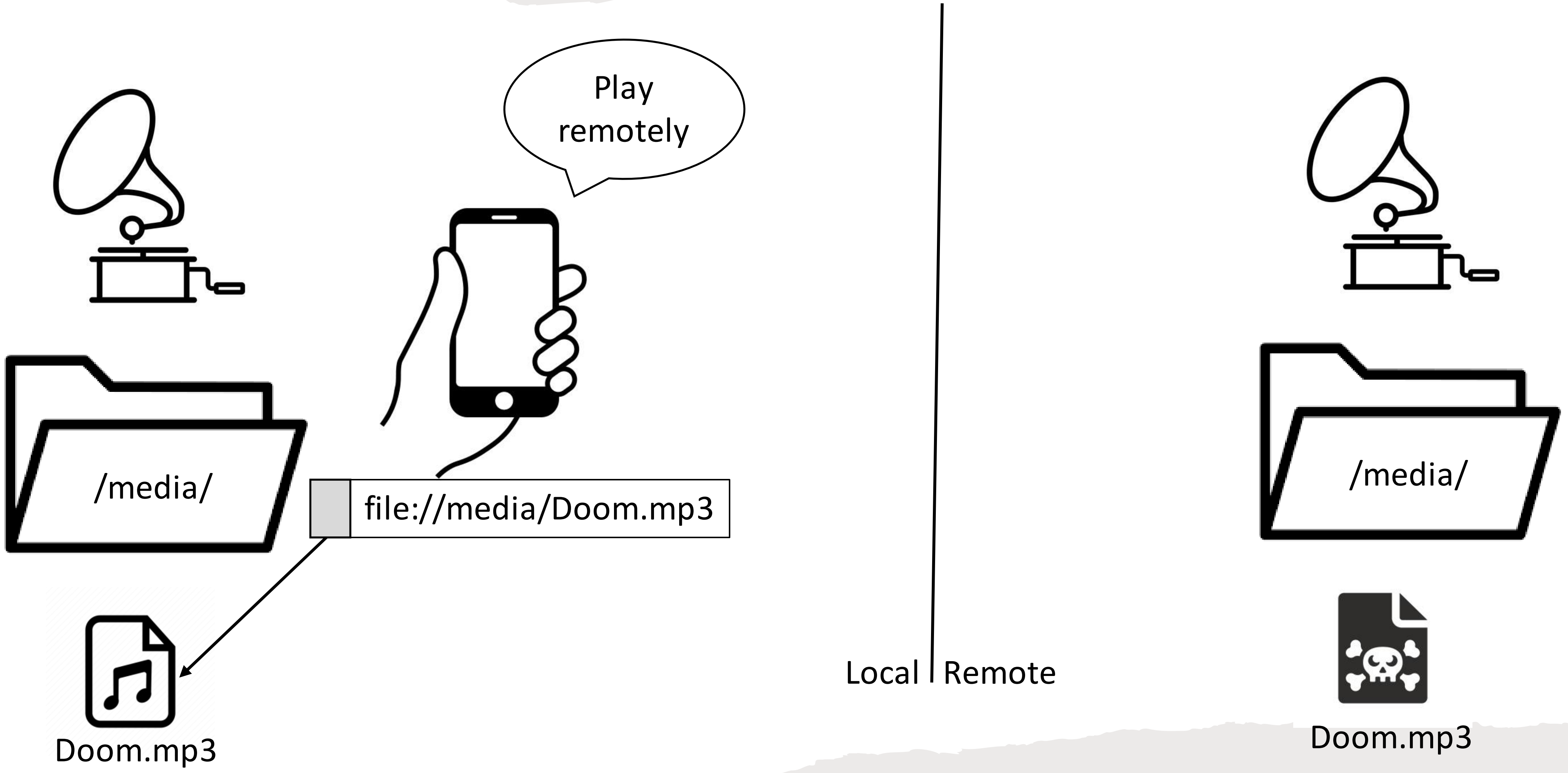
Evaluation: AntennaPod



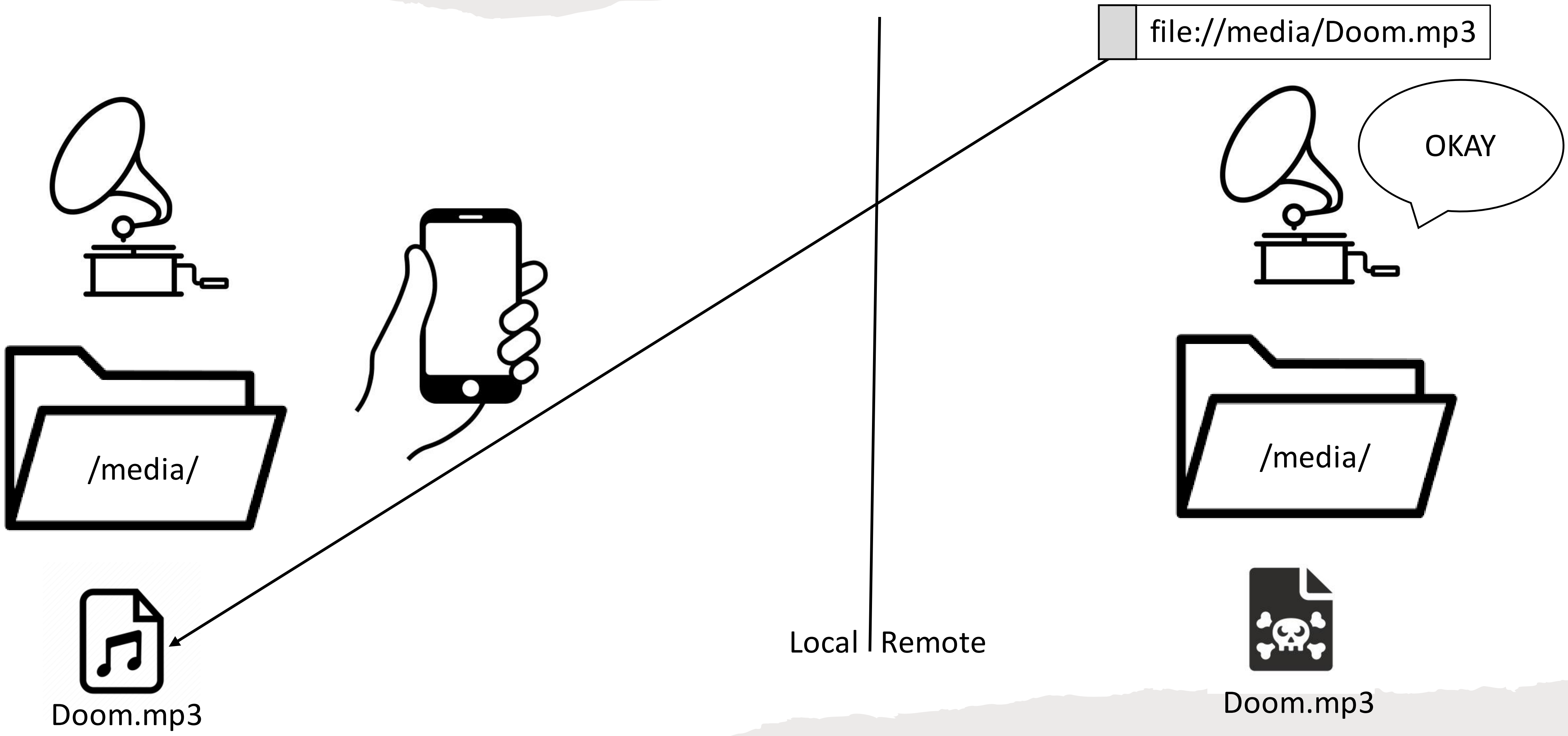
Evaluation: AntennaPod



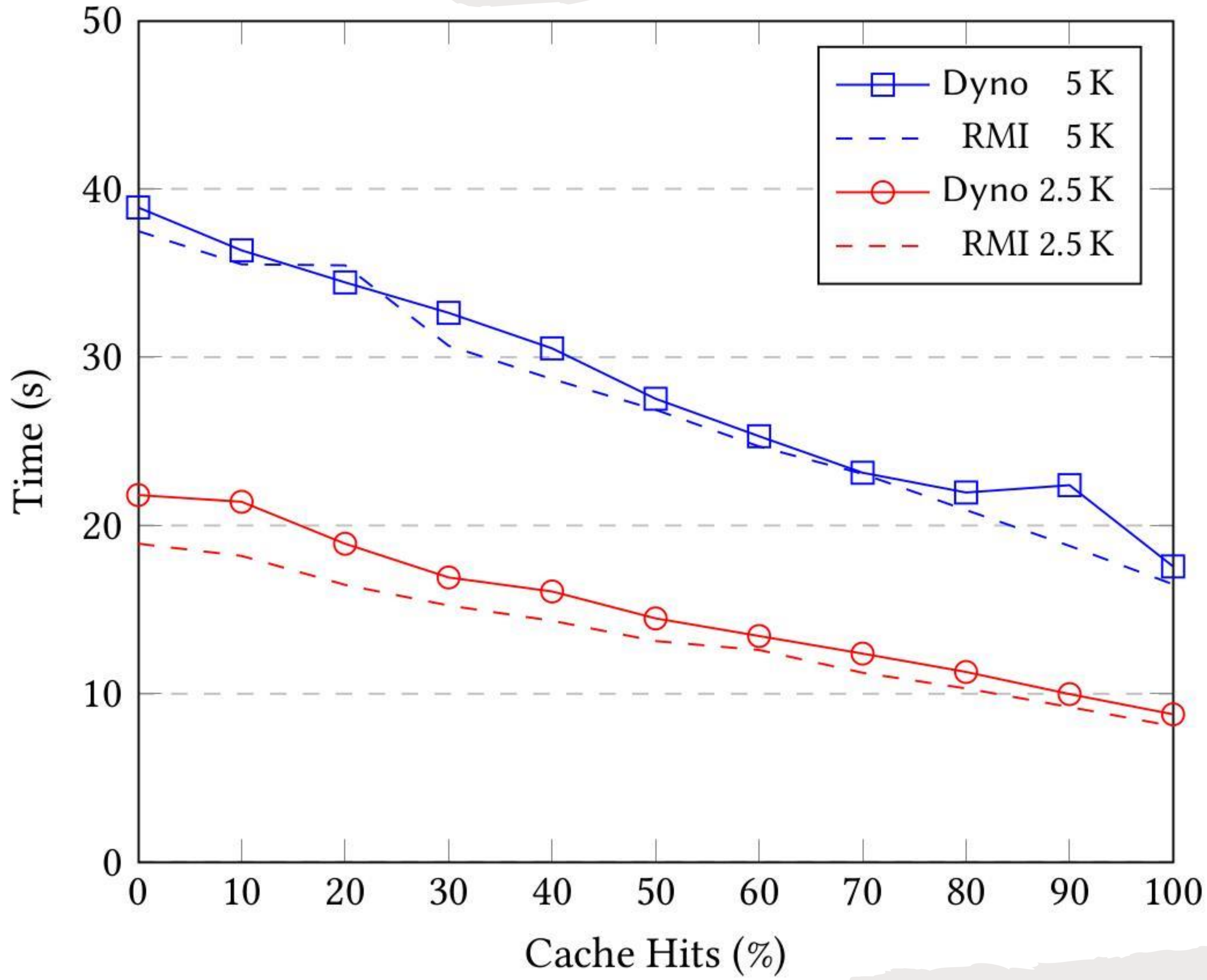
Evaluation: AntennaPod in Dyno



Evaluation: AntennaPod in Dyno



Evaluation: Performance



More goodies in the paper

- Case studies
- Comparative evaluation
- Implementation overview
- Formalization
- Theorems (and proofs in Appendix B)

Type-Safe Dynamic Placement with First-Class Placed Values

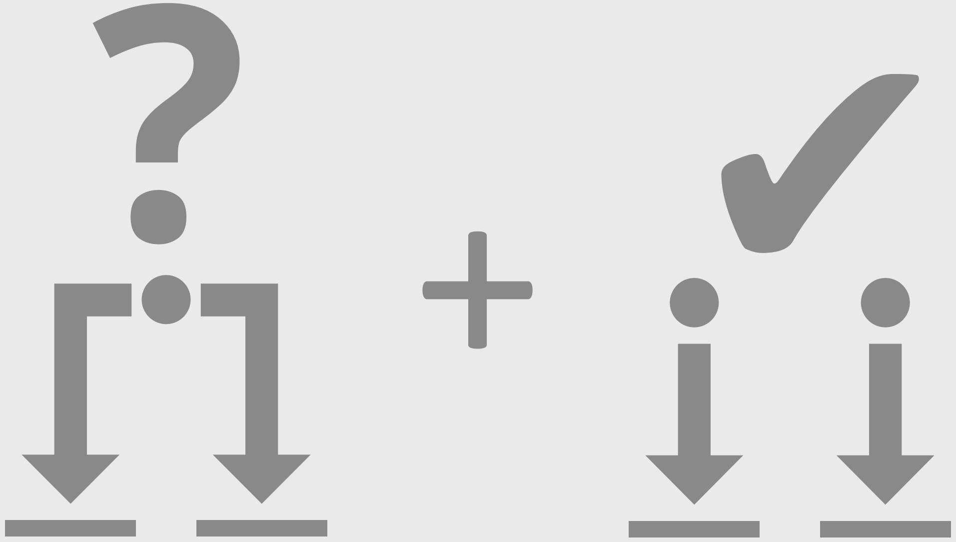
GEORGE ZAKHOUR, PASCAL WEISENBURGER, and GUIDO SALVANESCHI,
University of St. Gallen, Switzerland

Several distributed programming language solutions have been proposed to reason about the placement of data, computations, and peers interaction. Such solutions include, among the others, multitier programming, choreographic programming and various approaches based on behavioral types. These methods statically ensure safety properties thanks to a complete knowledge about placement of data and computation at compile time. In distributed systems, however, dynamic placement of computation and data is crucial to enable performance optimizations, e.g., driven by data locality or in presence of a number of other constraints such as security and compliance regarding data storage location. Unfortunately, in existing programming languages, dynamic placement conflicts with static reasoning about distributed programs: the flexibility required by dynamic placement hinders statically tracking the location of data and computation.

In this paper we present Dyno, a programming language that enables static reasoning about dynamic placement. Dyno features a type system where values are explicitly placed, but in contrast to existing approaches, placed values are also first class, ensuring that they can be passed around and referred to from other locations. Building on top of this mechanism, we provide a novel interpretation of dynamic placement as unions of placement types. We formalize type soundness, placement correctness (as part of type soundness) and architecture conformance. In case studies and benchmarks, our evaluation shows that Dyno enables static reasoning about programs even in presence of dynamic placement, ensuring type safety and placement correctness of programs at negligible performance cost. We reimplement an Android app with ~ 7 K LOC in Dyno, find a bug in the existing implementation, and show that the app's approach is representative of a common way to implement dynamic placement found in over 100 apps in a large open-source app store.

CCS Concepts: • **Software and its engineering** → **Distributed programming languages**; Domain specific languages; • **Theory of computation** → *Distributed computing models.*

Conclusion



Dynamic Placement + **Safe Placement**

Type System	Placement System
Types *Int, Boolean, String*	Placement Types *Client, Server, Database*
Param. Polymorphism *Map[K,V], Set[T], Either[S,T]*	Place-Polymorphic Modules *LeaderElection[P], DistHashMap[P]*
Union Types *Peano Numbers, Linked Lists, Options*	Dynamic Placement *DB|Cache, CPU|GPU, LocalFS|RemoteFS*

Data at (P | Q)

University of St.Gallen

p.g Programming Group