

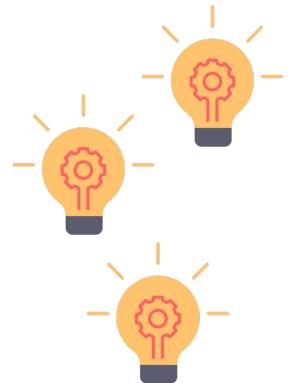
Artifact Evaluation: Enabling Reproducible Research

Bogdan “Bo” Stoica (UIUC)

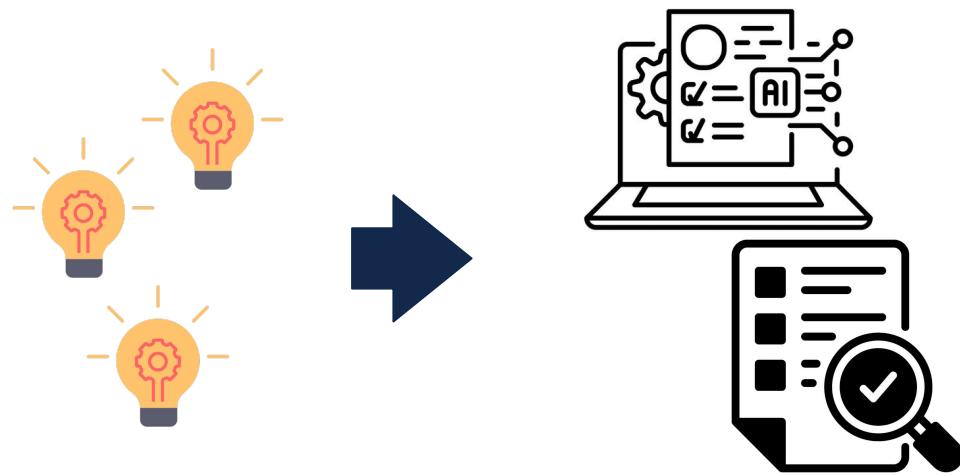


Microsoft
Research

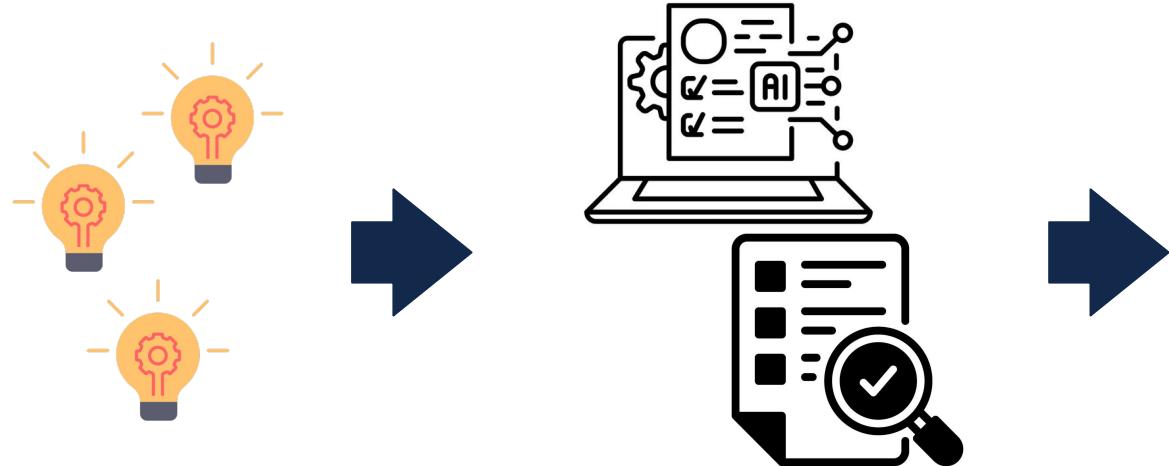
A typical “workflow” (Systems research)



A typical “workflow” (Systems research)



A typical “workflow” (Systems research)



LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation

Chris Lattner
University of Illinois at Urbana-Champaign
lattner.cs@illinois.edu
<http://llvm.cs.illinois.edu/>

ABSTRACT

This paper describes LLVM (Low Level Virtual Machine),

a compiler framework designed to support *transparent lifelong program analysis and transformation* for arbitrary programs,

by providing high-level information to compile transformations at compile-time, link-time, and run-time, and in idle state between runs. LLVM is a common, low-level code representation in Static Single Assignment (SSA) form,

with several novel features: a simple, *language-independent*

type-system that encodes the primitives commonly used to implement high-level languages; an integrated, type-safe, typed address arithmetic; and a simple abstraction that can be used to implement the exception handling features of high-level languages (and *setup/longjmp* in C) uniformly and efficiently. The LLVM representation is powerful and code reported together provide a combination of key capabilities that are important for practical, lifelong analysis and transformation of programs. To our knowledge, no existing compiler approach provides all these capabilities. We describe the design of the LLVM as a compiler and compiler framework, and evaluate the design in three ways: (a) the size and effectiveness of the representation, including the type system in particular; (b) compiler examples for several interoperability problems; and (c) illustrative examples of the benefits LLVM provides for several challenging compiler problems.

optimizations performed at link-time (to preserve the benefits of separate compilation), machine-dependent optimizations at install time on each system, dynamic optimization at runtime, and profile-guided optimization at run-time (“idle time”); (d) flexible interfaces to allow the end-user.

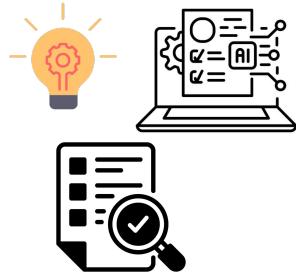
Program optimization is not the only use for static analysis and transformation. Other applications of static analysis are fundamentally interprocedural, and are therefore most convenient to perform at link-time (examples include static analysis for peak memory usage and memory management transformations [30]). Sophisticated analyses and transformations are being developed to enforce program safety, but must be done at software installation time or later [19]. The LLVM design is promising in that the program gives architects the power to evolve programs and exposed interfaces in more flexible ways [11, 20], while allowing legacy applications to run well on new systems.

This paper presents LLVM, a Low Level Virtual Machine, a compiler framework that aims to make lifelong program analysis and transformation available for arbitrary software, and in a manner that is transparent to programmers.

LLVM is implemented through two parts: (a) a code representation with several novel features that act as a common

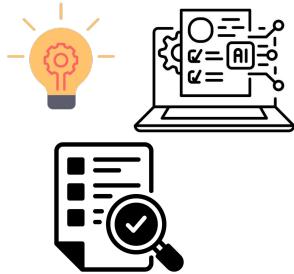
representation for analysis, transformation, and code distribution; and (b) a compiler design that exploits this representation to provide a combination of capabilities that is not available in any previous compilation approach we know of.

The peer-review process ...



Paper
preparation

The peer-review process ...



Paper preparation



LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation

Chris Lattner Vikram Adve
University of Illinois at Urbana-Champaign
(lattner,adve)@cs.uiuc.edu

ABSTRACT

minations performed at iteration k to possess the benefits (separate compilation, machine-dependent optimizations) of static *at system design*, dynamic optimizations of *at run time*, and static *at run time* (as in the *just-in-time*)! Once profitably informed by the end-of-chapter notes, you are free to interpret and use these notes to learn how to perform at least some of the static design, optimization, and run-time compilation transformations (Sect. 10.4). Separately, analysis and transformation of programs is a large and active field, but need not be at software architecture (line 10–19). Allowing fine-grained optimizations of the program at run time, as in the *just-in-time*, can expose improved performance in many ways [11, 26], while exposing increased memory usage [11, 26].

Finally, the **LIVIA** system (see Fig. 1) is a framework – a “coarse grainer” that aims to make program analysis and transformation as transparent as possible to the user, so as to be used in a manner that is transparent to programmers. LIVIA achieves this through two *principles*: (a) *code reuse* (i.e., reuse of code representations for analysis, transformation, and code generation); and (b) *cooperative design* that requires the user to provide *code skeletons* for analysis and transformation. It is not unusual in any previous complete approach to require the user to provide skeletons for analysis and transformation, but the **LIVIA** system is the first to do this in a transparent manner.

Paper deadline

The peer-review process ...



Paper preparation



LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation

Chris Lattner Vikram Adve
University of Illinois at Urbana-Champaign University of Illinois at Urbana-Champaign
(mailto:latner@cs.illinois.edu) (mailto:vadve@cs.illinois.edu)
<http://llvm.cs.illinois.edu/>

ABSTRACT

This paper describes LLVM (Low-Level Virtual Machine), a compiler framework designed to support transparent, lifelong program analysis and transformation. LLVM provides a high-quality, low-code-time interface for program analysis and transformation, by providing high-level information to complete transformations. LLVM is designed to support a wide range of transformations, from simple transformations that can be applied in a single pass, to complex transformations that require multiple passes. LLVM defines a common, low-level code representation for programs, which is used by various tools and libraries. LLVM is designed to be extensible, with several novel features: a single, language-independent representation of programs; a modular, extensible architecture; and a high-level interface for implementing high-level language features; an interface for translating programs between different languages; and a high-level interface for performing program analysis and transformation. LLVM provides a foundation for key capabilities in program analysis and transformation, including: (i) efficient, high-quality, and transparent program analysis and transformation; (ii) a high-quality, low-code-time interface for program analysis and transformation; (iii) a high-quality, low-code-time interface for performing program analysis and transformation; and (iv) a high-quality, low-code-time interface for performing program analysis and transformation.

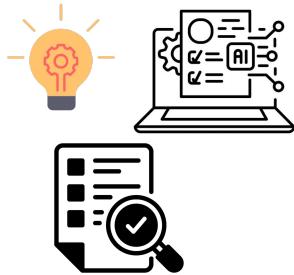
LLVM is designed to support a wide range of transformations, from simple transformations that can be applied in a single pass, to complex transformations that require multiple passes. LLVM is designed to be extensible, with several novel features: a single, language-independent representation of programs; a modular, extensible architecture; and a high-level interface for implementing high-level language features; an interface for translating programs between different languages; and a high-level interface for performing program analysis and transformation. LLVM provides a foundation for key capabilities in program analysis and transformation, including: (i) efficient, high-quality, and transparent program analysis and transformation; (ii) a high-quality, low-code-time interface for program analysis and transformation; (iii) a high-quality, low-code-time interface for performing program analysis and transformation; and (iv) a high-quality, low-code-time interface for performing program analysis and transformation.

Peer Reviewing ...



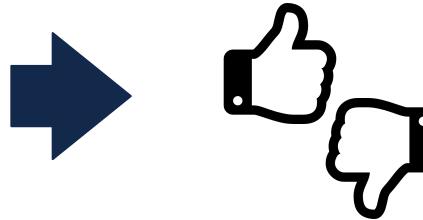
Paper deadline

The peer-review process ...



Chris Lattner Vikram Adve
University of Illinois at Urbana-Champaign
(lattner, vadv)@cs.illinois.edu

ABSTRACT



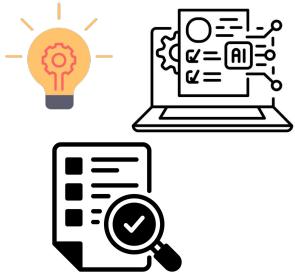
Paper preparation

Paper deadline

Paper notification

The peer-review process ...

Artifact Evaluation



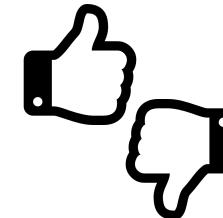
Paper preparation



LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation

Chris Lattner Vikram Adve
University of Illinois at Urbana-Champaign
(lattner, vadv)@cs.illinois.edu

ABSTRACT



Paper notification

Artifact evaluation: a definition ...

Artifact evaluation (AE) is the process of verifying that the artifacts released alongside a research paper (source code, datasets, scripts, configuration, etc.) faithfully correspond to the paper's description, and that they can be used to reproduce (or at least substantiate) the core paper claims, experimental setup, and reported results.

A bit of history ...

Software
Engineering

FSE'11**

*** SIGMOD ran a similar initiative 2008-2011, but a formal AE process started with at FSE'11*

A bit of history ...

Software
Engineering

FSE'11**

Programming
Language

OOPSLA'13

Security &
Privacy

WiSec'17

Systems

SOSP'19

*** SIGMOD ran a similar initiative 2008-2011, but a formal AE process started with at FSE'11*

A bit of history ...

Software Engineering

FSE'11**

ASE

ICSE

ISSTA

MSR

...

Programming Language

OOPSLA'13

ASPLOS

PLDI

POPL

PPoPP

...

Security & Privacy

WiSec'17

Security

CSS

S&P

NDSS

...

Systems

SOSP'19

OSDI

EuroSys

FAST

SIGCOMM

NSDI

...

** SIGMOD ran a similar initiative 2008-2011, but a formal AE process started with at FSE'11

A bit of history ...

Software Engineering

FSE'11**

ASE
ICSE

ISSTA
MSR
...

Programming Language

OOPSLA'13

ASPLOS

PLDI
POPL

PPoPP

...

Security & Privacy

WiSec'17

Security
CSS

S&P
NDSS

...

Systems

SOSP'19

OSDI
EuroSys

FAST
SIGCOMM

NSDI

...

Computer Architecture

AI/ML

Database Systems

Cryptography

...

** SIGMOD ran a similar initiative 2008-2011, but a formal AE process started with at FSE'11

Artifact evaluation: goals

Increase thrust of published research (artifact “badges”)

Artifact evaluation: goals

Increase thrust of published research (artifact “badges”)

Ensure artifacts are available & easily accessible

Artifact evaluation: goals

Increase thrust of published research (artifact “badges”)

Ensure artifacts are available & easily accessible

Facilitate reproducibility of key findings

Artifact evaluation: goals

Increase thrust of published research (artifact “badges”)

Ensure artifacts are available & easily accessible

Facilitate reproducibility of key findings

Enable reusability & extensibility

Artifact evaluation: badges



Artifact Available: publicly & permanently available

Artifact evaluation: badges



Artifact Available: publicly & permanently available



Artifact Functional: documented, exercisable, and includes validation



Artifact Reusable: repurposable, modular, and extensible

Artifact evaluation: badges



Artifact Available: publicly & permanently available



Artifact Functional: documented, exercisable, and includes validation



Artifact Reusable: repurposable, modular, and extensible



Result Reproduced: re-obtained by using, in part, author-provided artifacts



Result Replicated: re-obtained without author-provided artifacts

Artifact evaluation: badges (Systems)



Artifact Available: publicly & permanently available



Artifact Functional: documented, exercisable, and includes validation



Result Reproduced: re-obtained by using, in part, author-provided artifacts

Artifact evaluation timeline

Preparation

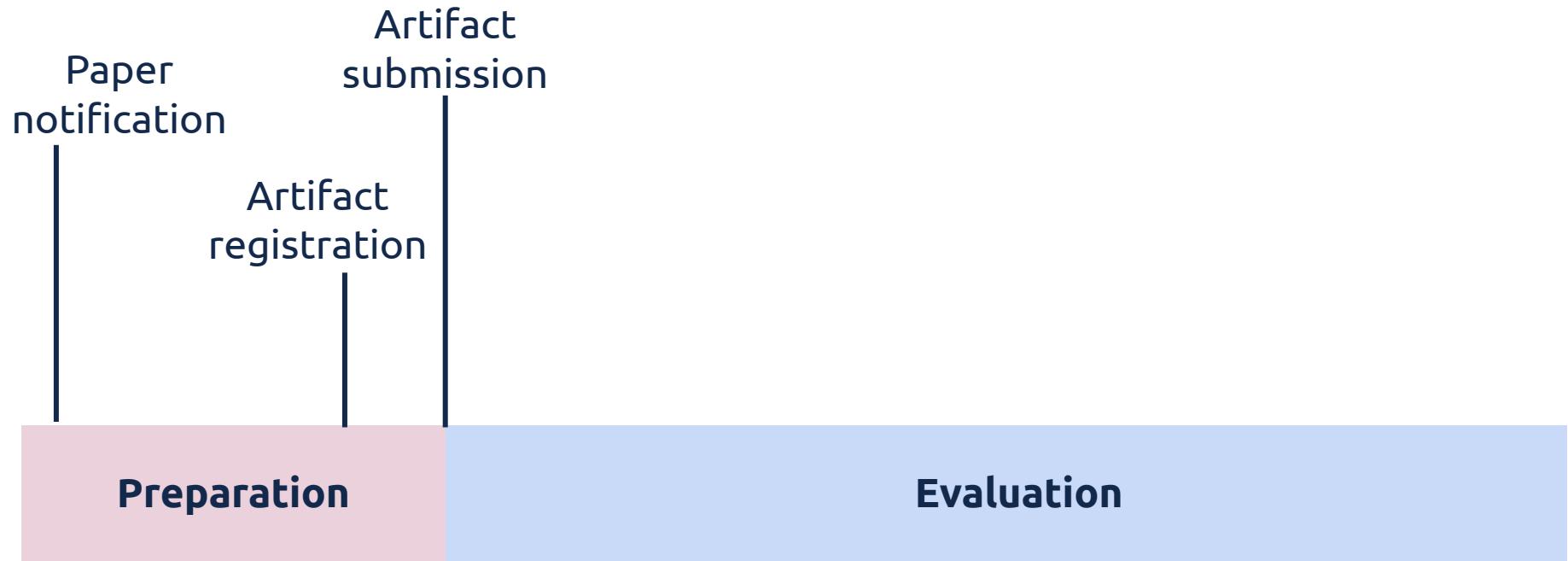
Evaluation

Artifact evaluation timeline

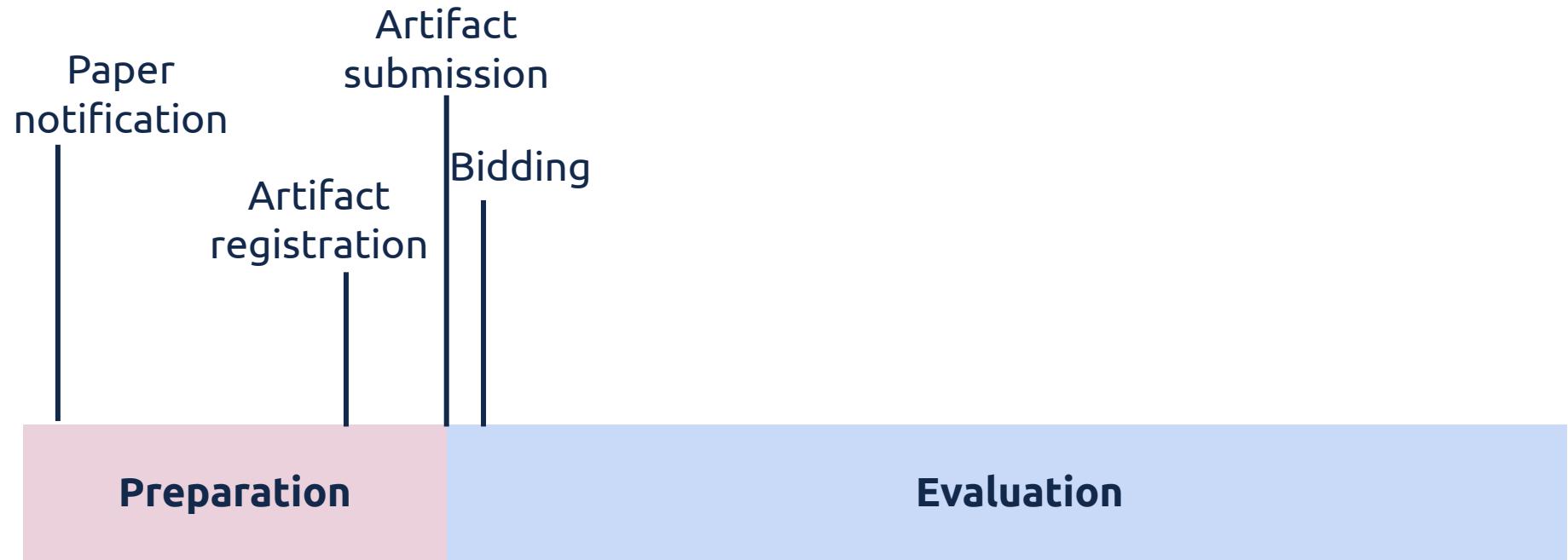
Paper
notification



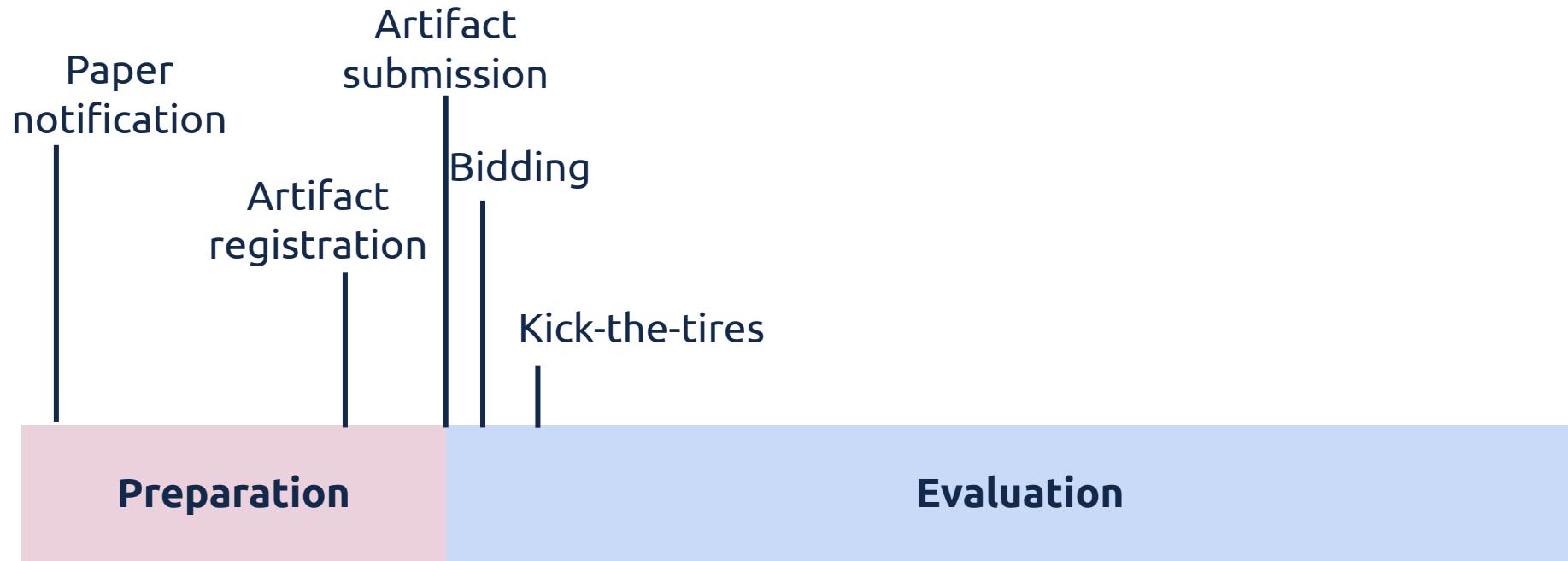
Artifact evaluation timeline



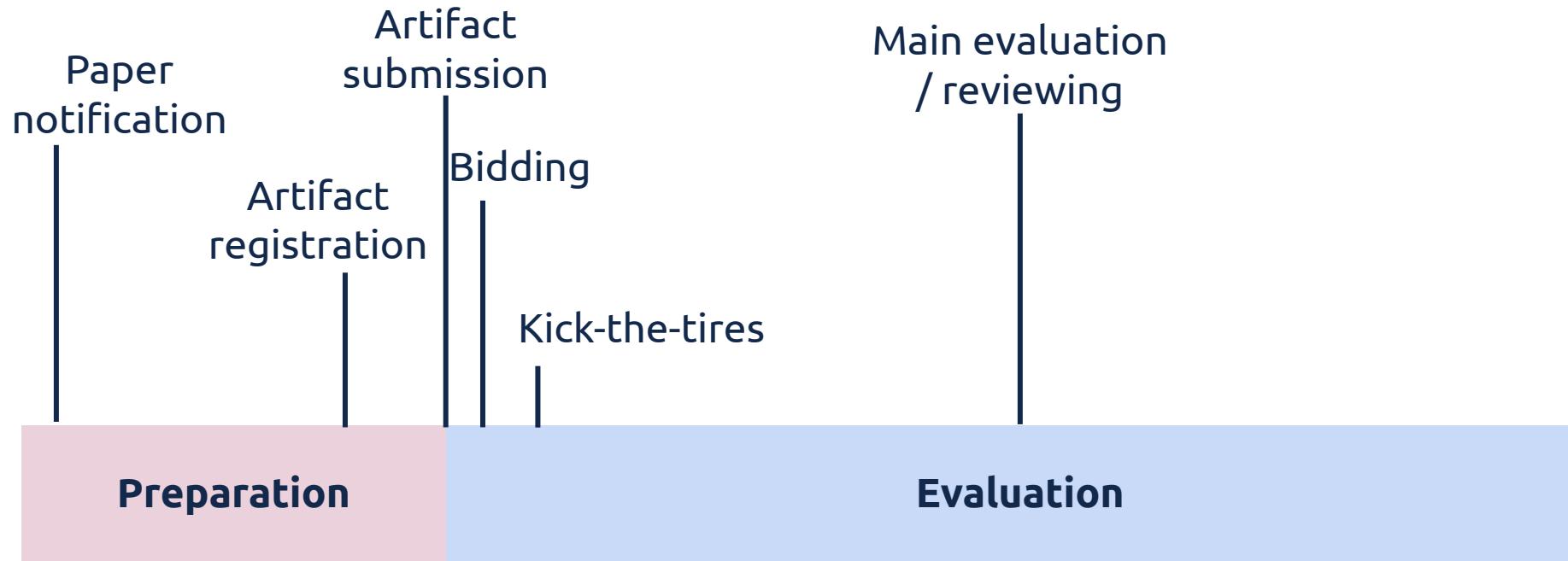
Artifact evaluation timeline



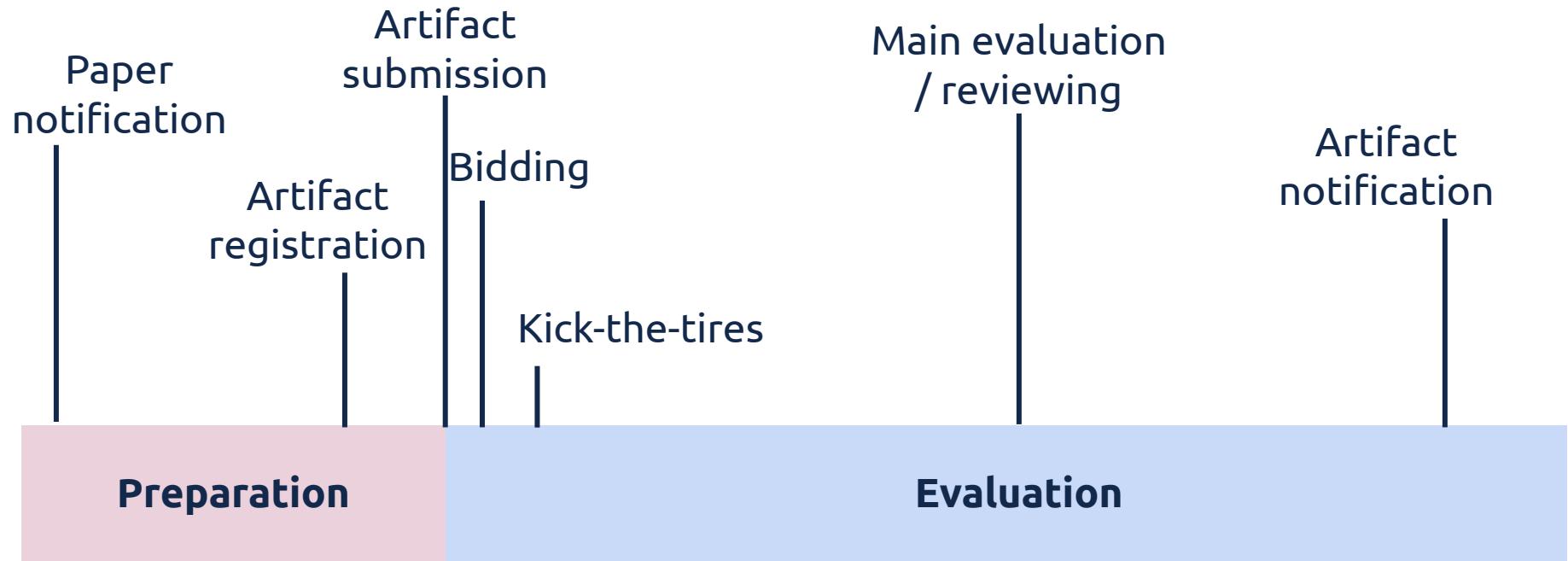
Artifact evaluation timeline



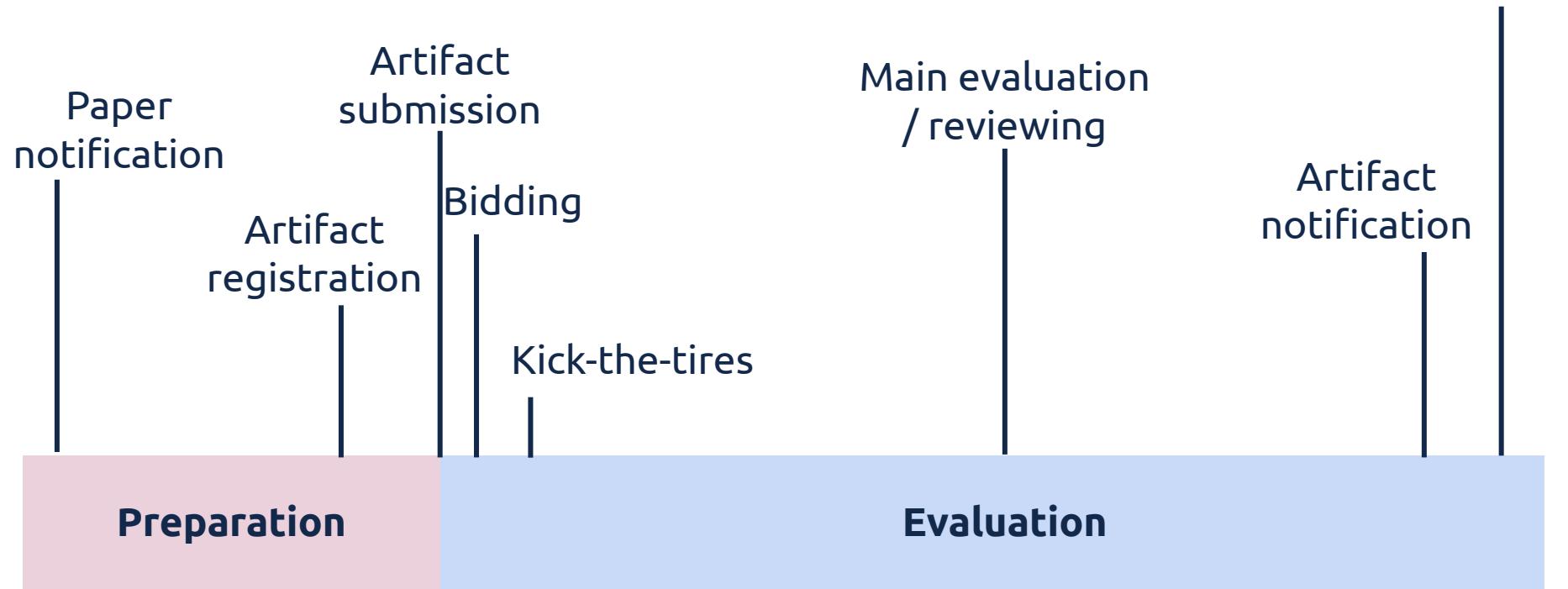
Artifact evaluation timeline



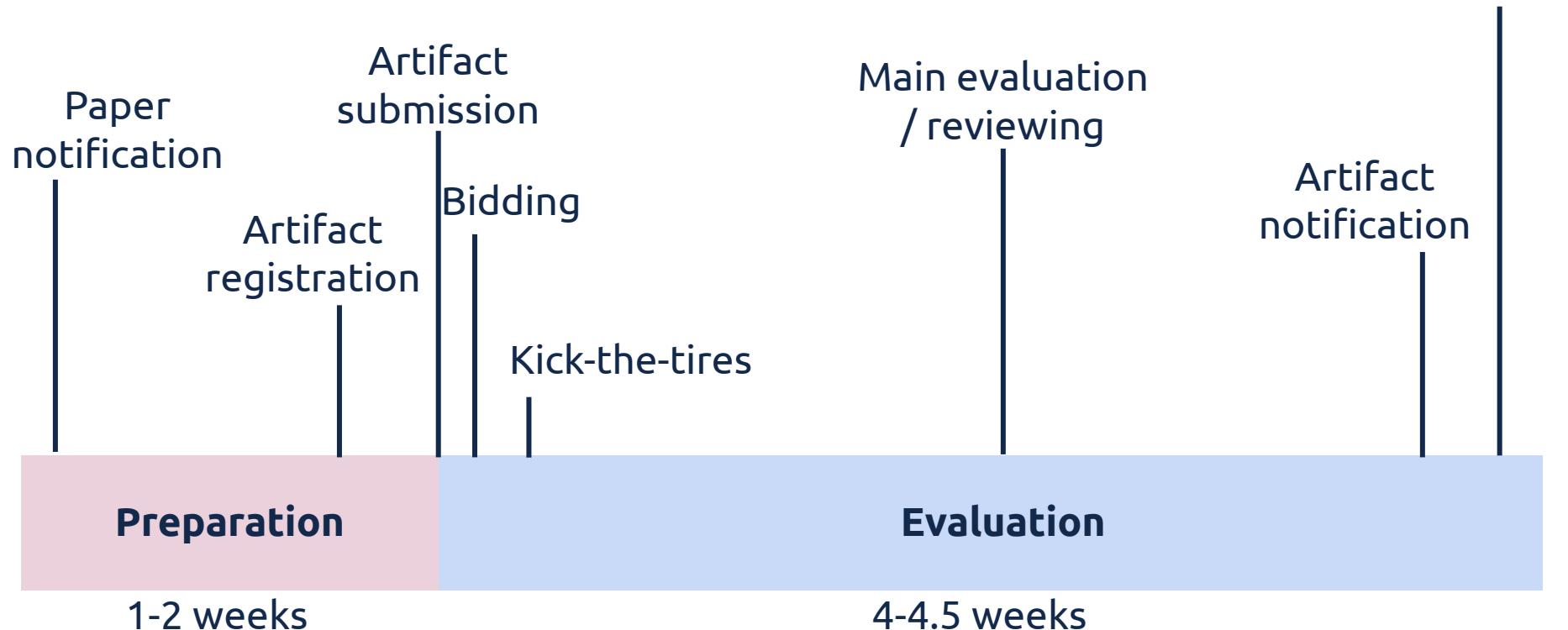
Artifact evaluation timeline



Artifact evaluation timeline



Artifact evaluation timeline



The role of authors

Prepare a self-contained artifact w/ persistent hosting

The role of authors

Prepare a self-contained artifact w/ persistent hosting

Write clear guidelines (appendix and/or README)

The role of authors

Prepare a self-contained artifact w/ persistent hosting

Write clear guidelines (appendix and/or README)

Provide a minimal, simple experiment as “running example”

The role of authors

Prepare a self-contained artifact w/ persistent hosting

Write clear guidelines (appendix and/or README)

Provide a minimal, simple experiment as “running example”

Engage with reviewers to improve the artifact

The role of reviewers

Evaluate the artifact, but also audit paper-code alignment

The role of reviewers

Evaluate the artifact, but also audit paper-code alignment

Start early, engage with the authors

The role of reviewers

Evaluate the artifact, but also audit paper-code alignment

Start early, engage with the authors

Follow the Chairs' guidelines and provided badge checklist

The role of reviewers

Evaluate the artifact, but also audit paper-code alignment

Start early, engage with the authors

Follow the Chairs' guidelines and provided badge checklist

Write a thorough, detailed, and respectful review

Available infrastructure

Individual machines (e.g., desktop, laptop)

Available infrastructure

Individual machines (e.g., desktop, laptop)

Academic cloud infrastructure



Available infrastructure

Individual machines (e.g., desktop, laptop)



Academic cloud infrastructure



Commercial cloud infrastructure



Available infrastructure

Individual machines (e.g., desktop, laptop)

Academic cloud infrastructure

Commercial cloud infrastructure



Most frequent challenges

C1: Short preparation & review windows

Most frequent challenges

C1: Short preparation & review windows

C2: Persistent artifact availability

Most frequent challenges

C1: Short preparation & review windows

C2: Persistent artifact availability

C3: Specialized hardware requirements

Most frequent challenges

C1: Short preparation & review windows

C2: Persistent artifact availability

C3: Specialized hardware requirements

C4: Environment setup, configuration, and installation friction

Most frequent challenges

Lessons Learned from Five Years of Artifact Evaluations at EuroSys

C1: Sho

C2: Per:

C3: Env

C4: Spe

ription

Daniele Cono D'Elia, Sapienza University of Rome, Italy
Thaleia Dimitra Doudali, IMDEA Software Institute, Spain
Cristiano Giuffrida, VU Amsterdam, Netherlands
Miguel Matos, IST Lisbon & INESC-ID, Portugal
Mathias Payer, EPFL, Switzerland
Solal Pirelli, Independent Researcher, Switzerland
Georgios Portokalidis, IMDEA Software Institute, Spain
Valerio Schiavoni, University of Neuchâtel, Switzerland
Salvatore Signorello, NOVA University Lisbon, Portugal
Anjo Vahldiek-Oberwagner, Intel Labs, Germany

Abstract

Artifact Evaluation (“AE”) is now an accepted practice in the systems community. However, AE processes are inconsistent across venues and even across different editions of the same venue. AE processes regularly encounter the same problems across venues and years. Based on our collective experience in chairing various and heterogeneous AE committees for five consecutive editions of EuroSys, a large systems conference, we present the challenges we believe most pressing. We propose concrete steps to address these challenges in future AEs, serving as guidelines for future chairs and AE committees.

overarching goal of these considerations is scaling up AE practices to increase their long-term impact. This mindset sparked the creation of various initiatives in CS research, such as the ACM Emerging Interest Group for Reproducibility and Replicability [11], the SIGSOFT Artifact Evaluation Working Group [24], the ACM SIGMOD ARI [13], and various other AE processes [23, 25].

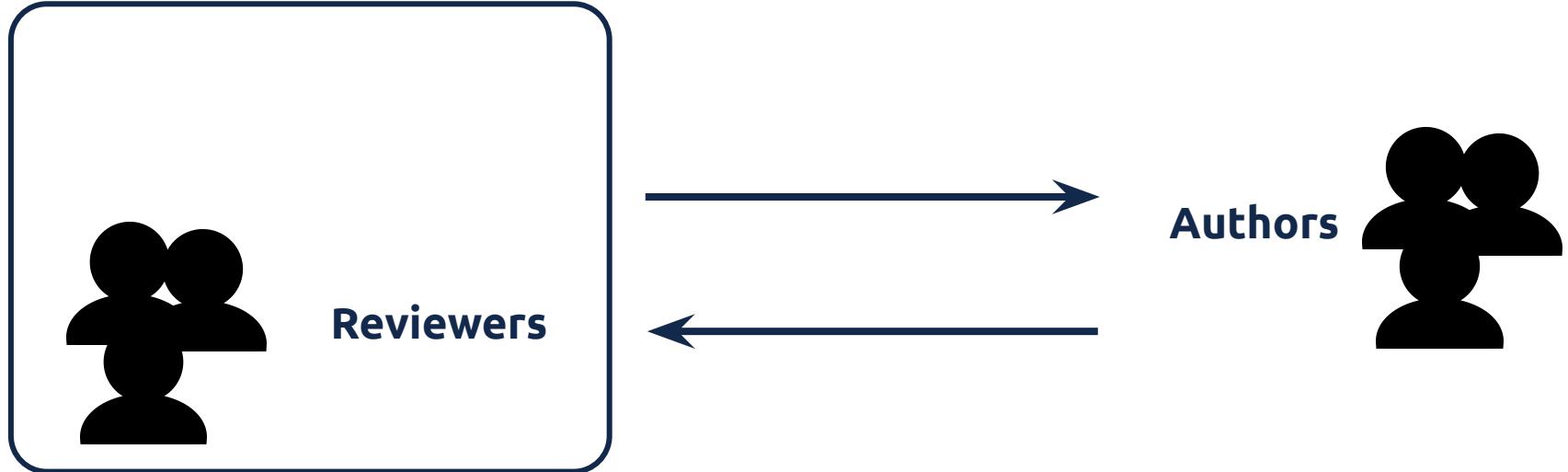
AE is the conceptually simple process of checking whether the artifacts published alongside a paper, such as code and data, correspond to what the paper describes. In practice, this leads to many questions and challenges. The very first AE process we are aware of, at ESEC/FSE 2011 [2], awarded a badge to papers that passed an

Future of Artifact Evaluation ...

C4: Environment setup, configuration, and installation friction

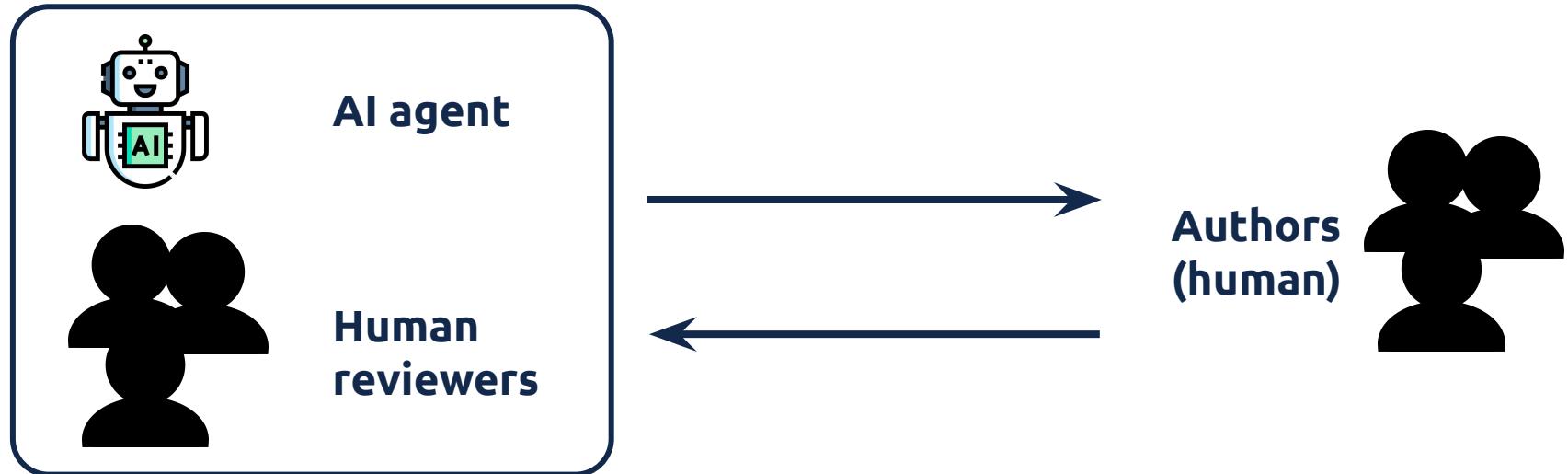
Future of Artifact Evaluation ...

C4: Environment setup, configuration, and installation friction

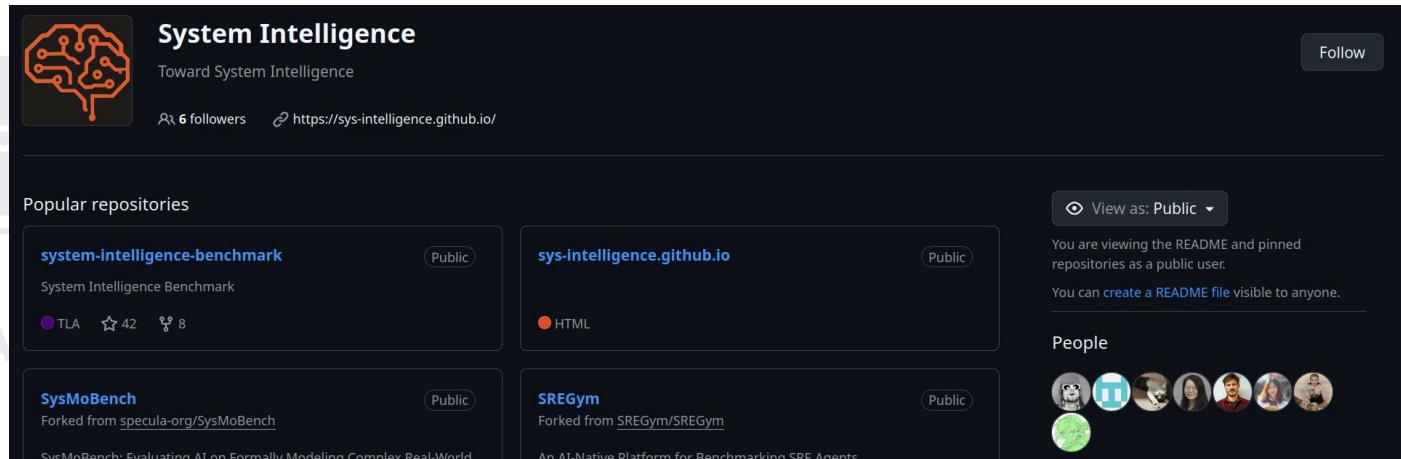


Future of Artifact Evaluation ...

C4: Environment setup, configuration, and installation friction



Future of Artifact Evaluation ...



System Intelligence
Toward System Intelligence

Follow

6 followers <https://sys-intelligence.github.io/>

Popular repositories

Repository	Type	Actions
system-intelligence-benchmark	Public	TLA ⭐ 42 8
sys-intelligence.github.io	Public	HTML
SysMoBench	Public	Forked from specula-org/SysMoBench
SREGym	Public	Forked from SREGym/SREGym

View as: Public

You are viewing the README and pinned repositories as a public user.

You can create a README file visible to anyone.

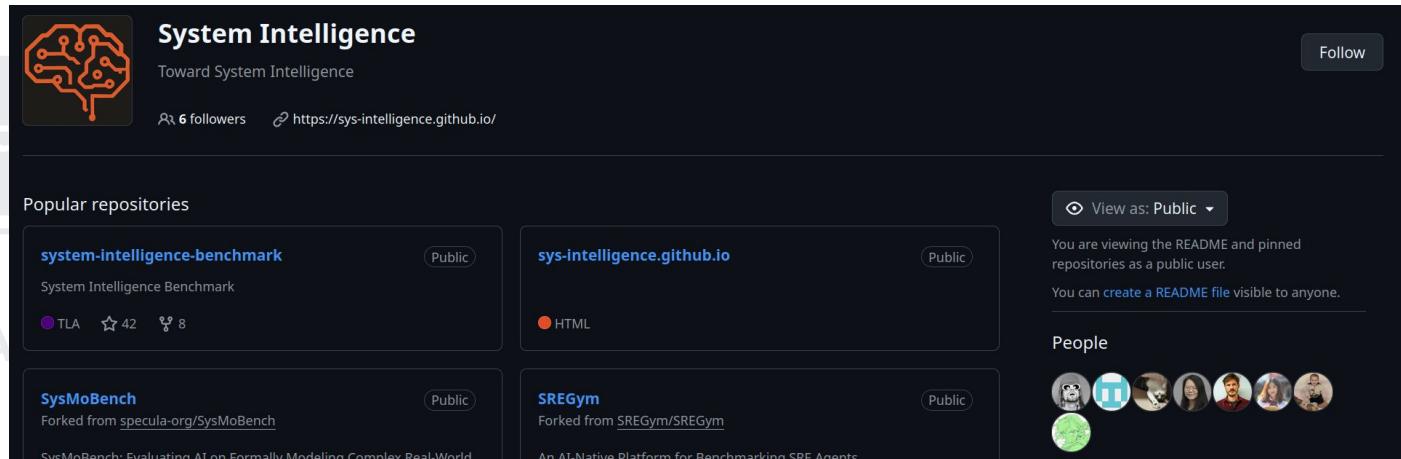
People



Authors (human)

Human
reviewers

Future of Artifact Evaluation ...

A screenshot of a GitHub repository page for "System Intelligence". The repository has 6 followers and a public URL: <https://sys-intelligence.github.io/>. The page shows a "Popular repositories" section with four items: "system-intelligence-benchmark" (Public, TLA, 42 stars, 8 forks), "sys-intelligence.github.io" (Public, HTML), "SysMoBench" (Public, Forked from specula-org/SysMoBench), and "SREGym" (Public, Forked from SREGym/SREGym). A "View as: Public" dropdown is open. The "People" section shows a list of contributors with their profile pictures. The background of the slide features a large, semi-transparent watermark of a robot head with "AI" on its chest and the word "AUTHORS" repeated in the background.

Check out the “System Intelligence” series on ACM SIGOPS Blog

Want to get involved?

Ongoing: EuroSys'26 (email us by Jan 30: aec-2026@eurosys.org)

Upcoming: OSDI'26, SOSP'26, EuroSys'27, ASPLOS'27, etc.

Want to get involved?

Ongoing: EuroSys'26 (email us by Jan 30: aec-2026@eurosys.org)

Upcoming: OSDI'26, SOSP'26, EuroSys'27, ASPLOS'27, etc.

SysIntelligence: -- contact Bo (bastoica@illinois.edu)
-- drop by @ <https://github.com/sys-intelligence/>

Thank you!