ICPC Library

Wednesday, September 3, 2025

ICPC Library

- **ICPC Standardizations** (Fredrik Niemelä)
- **The Future of ICPC Problems** (Etienne Vouga)
- **An introduction to problemsetting** (Joshua Andersson)
- **Uses of the new Problem Format** (Rasmus John Hulthe)
- **ICPC Archive** (Miguel Revilla Rodríguez)

ICPC Standardizations

Fredrik Niemelä, ICPC Standards Committee Chair

Standards

- CCS System Requirements
- Contest API
- Contest Package Format
- Problem Package Format

CCS / Contest API / CPF WG

- Fredrik Niemelä
- John Buck
- John Clevenger
- Tim deBoer
- Jaap Eldering
- Nicky Gerritsen
- Pavel Kunyavskiy

Problem package format WG

- Fredrik Niemelä
- Per Austrin
- Arnar Bjarni Arnarson
- Johan Sannemo
- Ragnar Groot Koerkamp
- Jaap Eldering
- Greg Hamerly
- Thore Husfeldt
- Simon Lindholm
- Michael Zündorf
- Etienne Vouga
- Joel Niemelä
- Pehr Söderman
- Joshua Andersson

Want to know more, want to get involved?

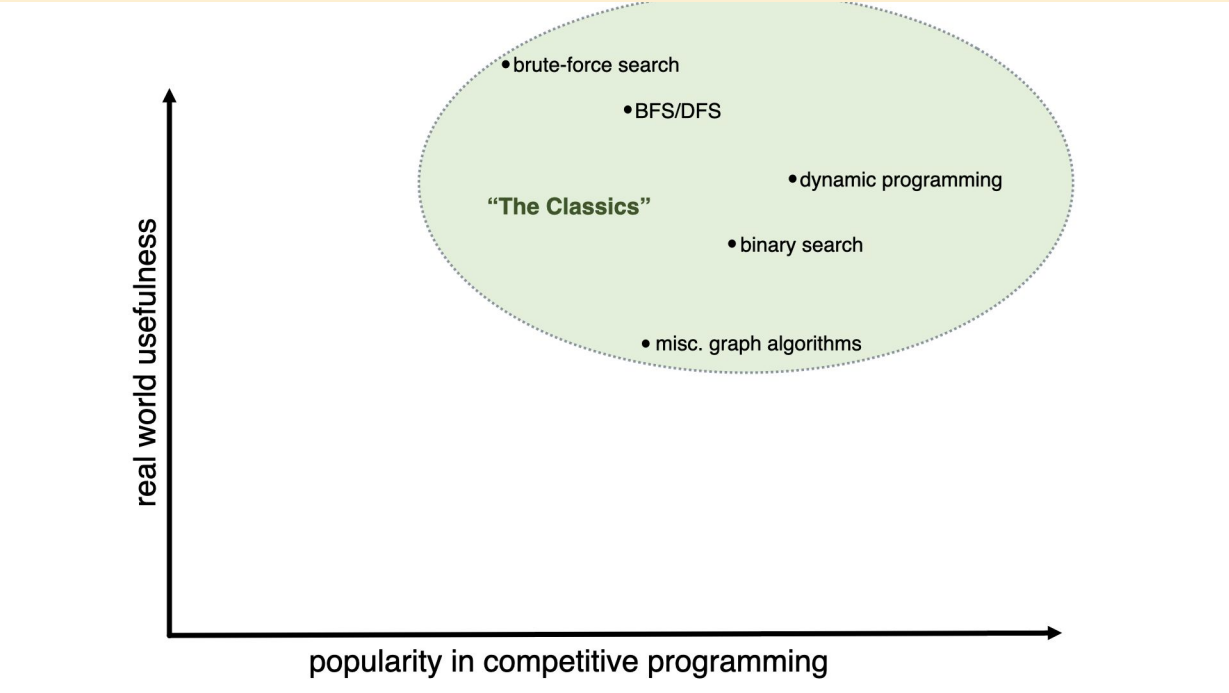
Come and talk to me!

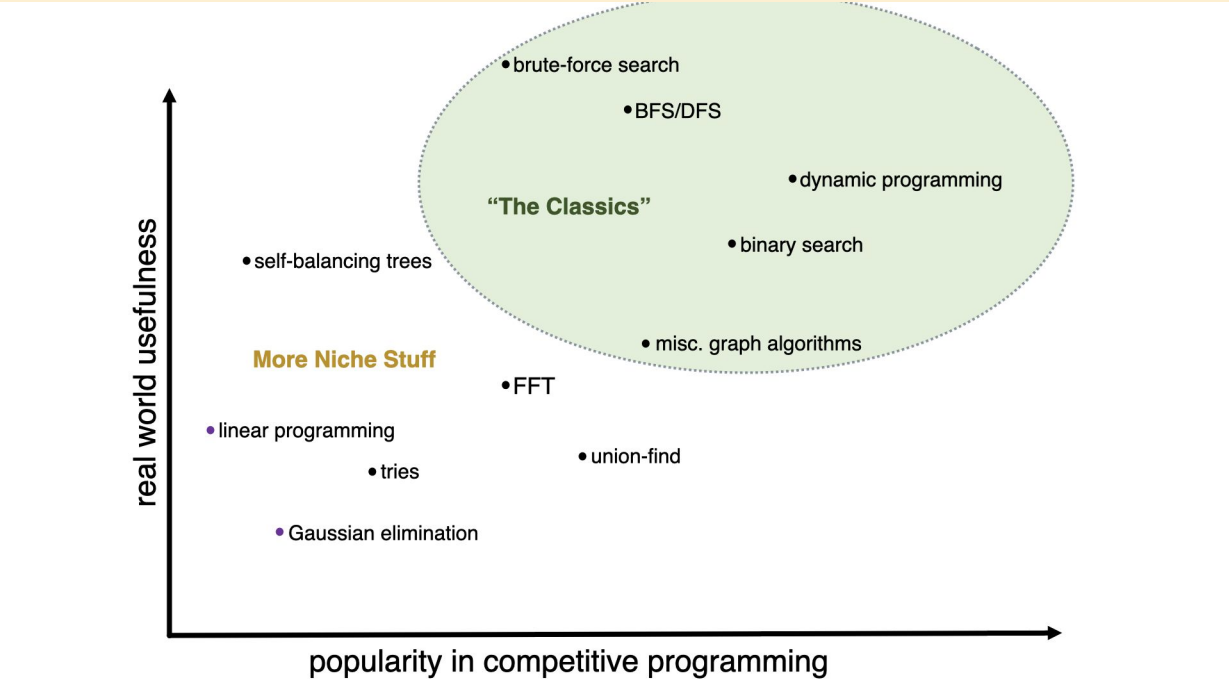
The Future of ICPC Problems

Etienne Vouga
CLI ICPC Compete Track
2025-08-31

Constraints on Current ICPC Problems

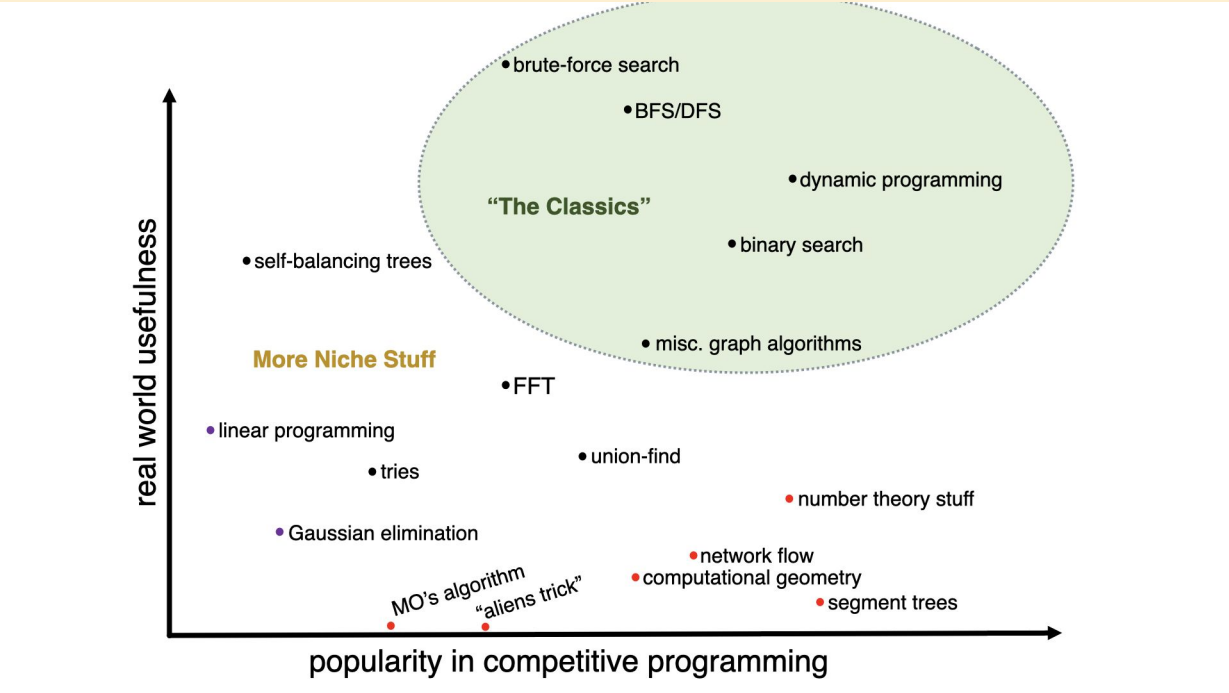
- Solution strategies are limited to material taught in undergrad core CS
- Solutions can use only the standard library of supported languages
- All input comes from `stdin`
- Problems should be “interesting” (involve nontrivial problem-solving)
- All solutions must receive binary (accept/reject) verdicts
- The hardest problems must separate the strongest teams
- The problem set must entertain the weakest teams for 5 hours


brute-force search
BFS/DFS
dynamic programming
"The Classics"
binary search
misc. graph algorithms
real world usefulness
popularity in competitive programming



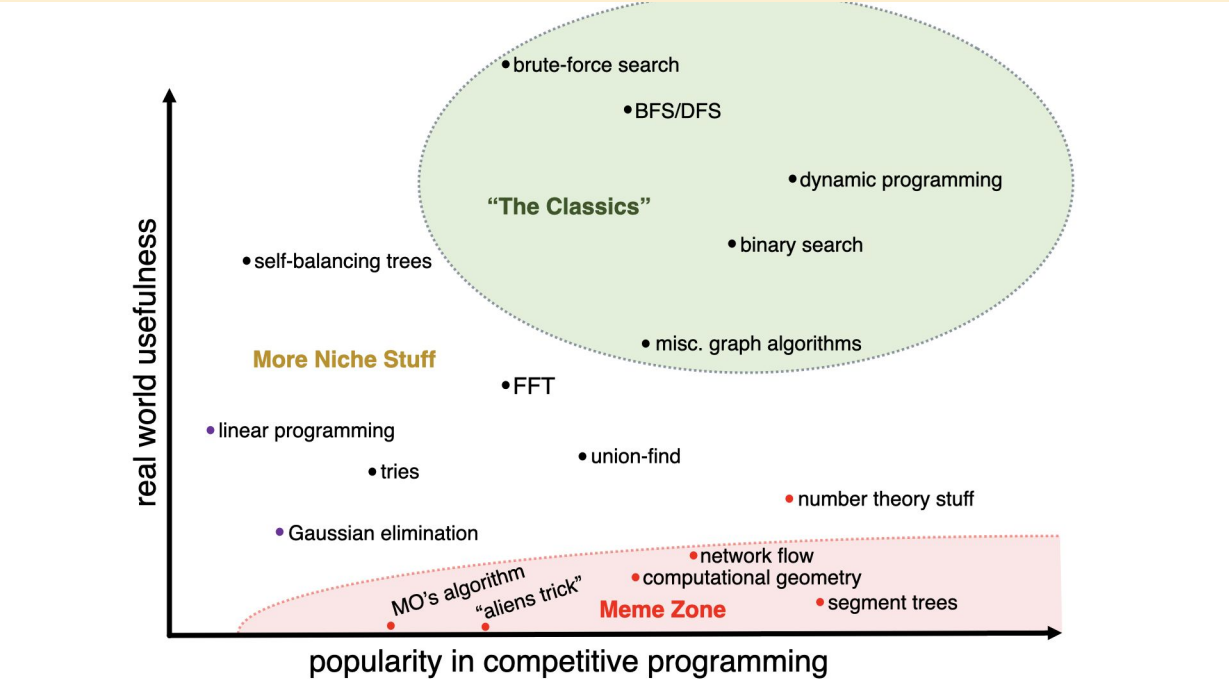
The 2025 ICPC World Finals Baku

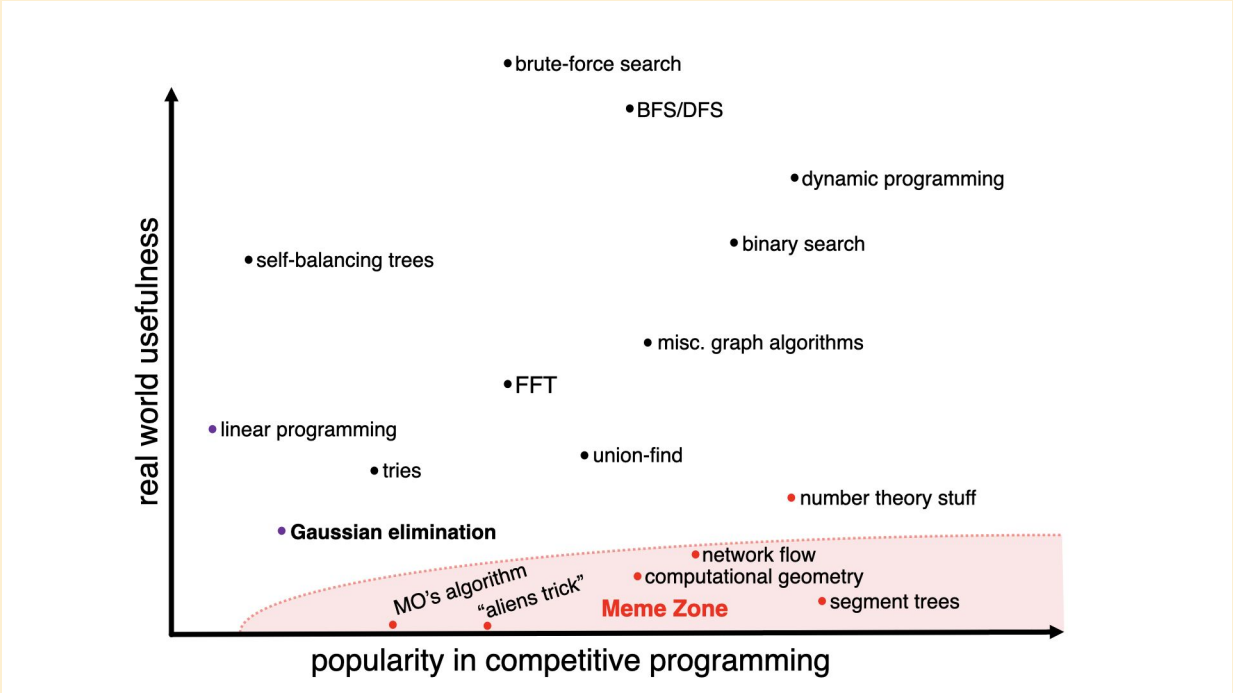
31 August – 5 September 2025 // hosted by **ADA University**



The 2025 ICPC World Finals Baku

31 August – 5 September 2025 // hosted by **ADA University**

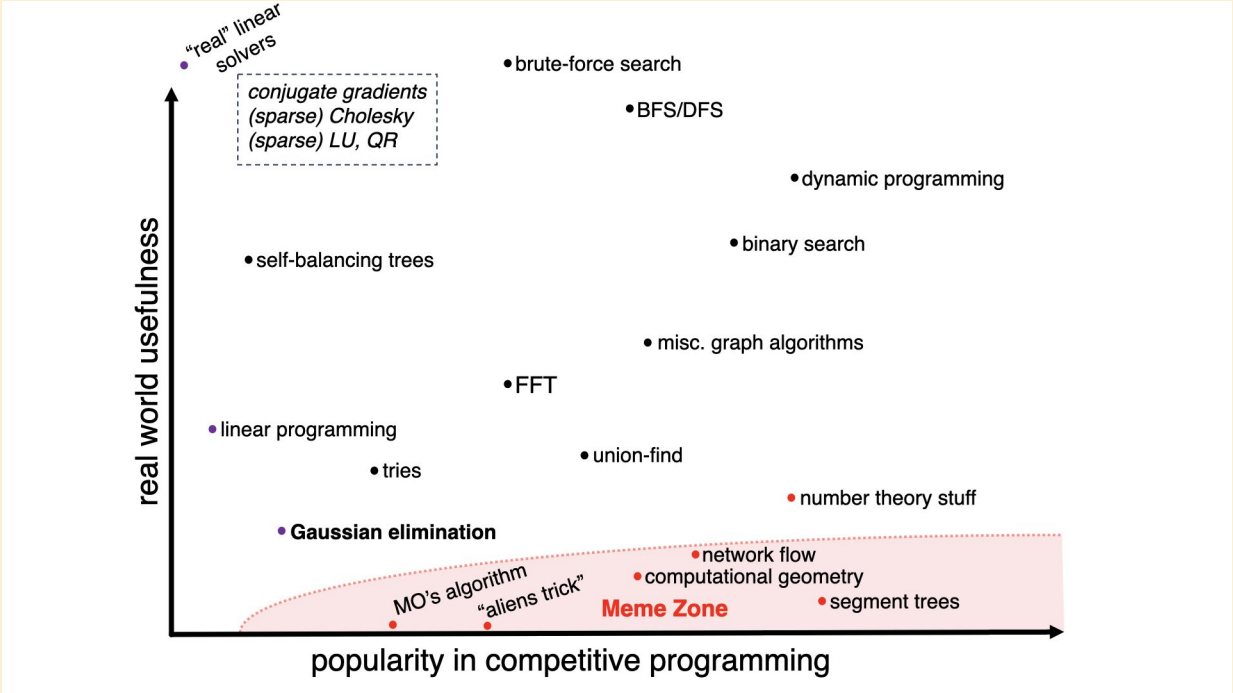




ICPC International Collegiate Programming Contest

The 2025 ICPC World Finals Baku

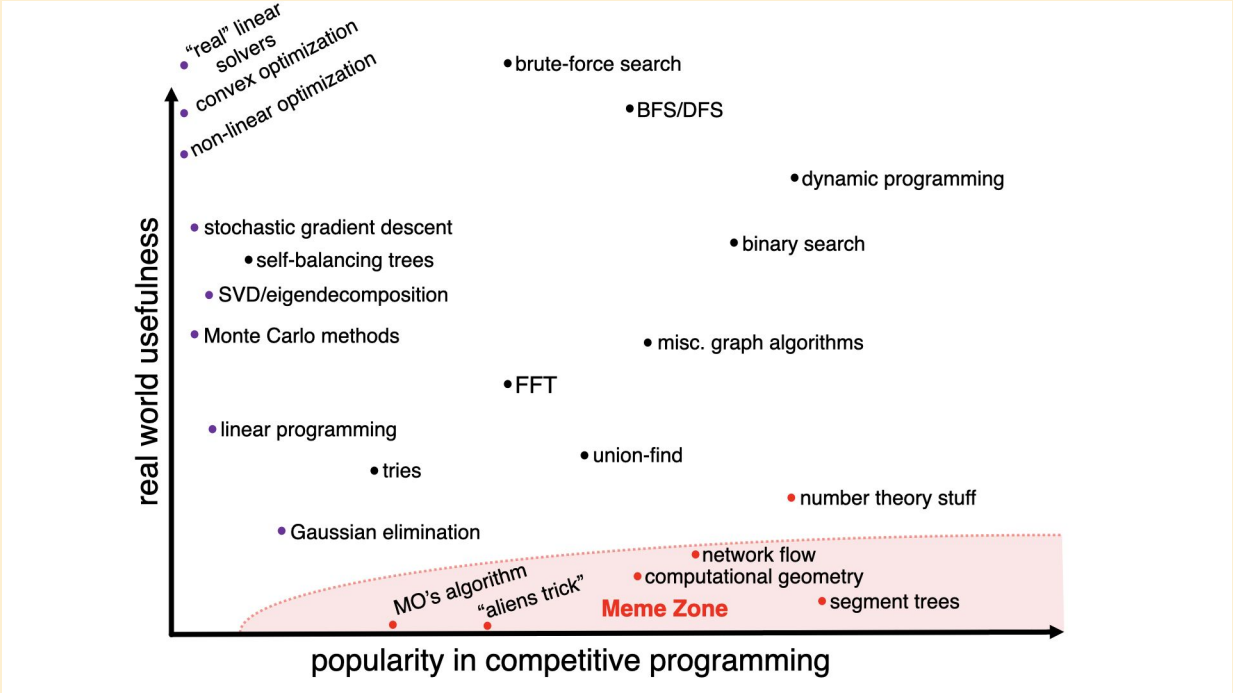
31 August – 5 September 2025 // hosted by **ADA University**



ICPC International Collegiate Programming Contest

The 2025 ICPC World Finals Baku

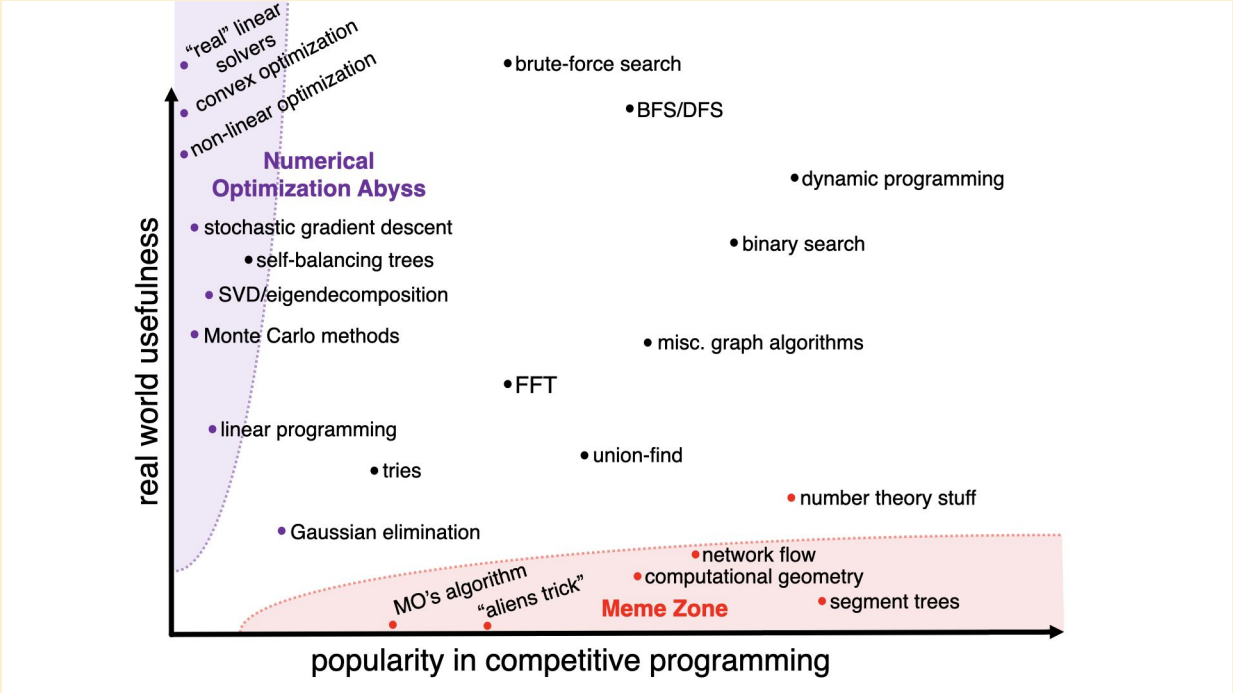
31 August – 5 September 2025 // hosted by **ADA University**



ICPC International Collegiate Programming Contest

The 2025 ICPC World Finals Baku

31 August – 5 September 2025 // hosted by **ADA University**



Allowed ICPC Topic Areas

- **Theory**: solution strategies are limited to material taught in core CS
  - welcoming for newcomers
  - provides concrete guidance on what to train for
  - delineates **collegiate** competitive programming from **professional** competitive programming events (e.g. Codeforces)

Allowed ICPC Topic Areas

- **Theory**: solution strategies are limited to material taught in core CS
- **Practice**: competitive programming has its own curriculum
  - heavily algorithms-focused
  - calculus vs. computational geometry
  - Ford-Fulkerson vs. Dinic's Algorithm
  - method of Lagrange multipliers vs. the "Aliens trick"
  - templates, `#pragma`s, and incantantations (e.g. `sync_with_stdio`)

Proposal #1: Diversify ICPC Problems

- Explicitly allow more topics covered in CS-related applications/electives
  - calculus, numerical linear algebra, numerical optimization
  - operating systems, databases, networking
  - AI, computer vision, robotics, statistics

Proposal #1: Diversify ICPC Problems

- Explicitly allow more topics covered in CS-related applications/electives

- Expand the available packages/libraries
  - to unlock more problem types: `Eigen`, `numpy`
  - to reduce reliance on copy/pasted book code
    - how many students **actually** understand their book FFT implementation?

Proposal #1: Diversify ICPC Problems

- Explicitly allow more topics covered in CS-related applications/electives
- Expand the available packages/libraries
- Take advantage of Interactive Problem support to simulate databases / containers not parsed from `stdin`
  - allows for more interesting performance characteristics and constraints

IO Using Interaction: Example

You are running an election and have gathered a list of ballots (candidate name strings). Given the **sorted** list of ballots, determine who (if anyone) won a majority (>50%) of the votes.

- **Constraints**: O(1) space, O(log N) time
- **Harder version**: unsorted list, O(1) space, O(N) time

- Interaction allows for random access into the list of ballots
  - constraint on number of queries

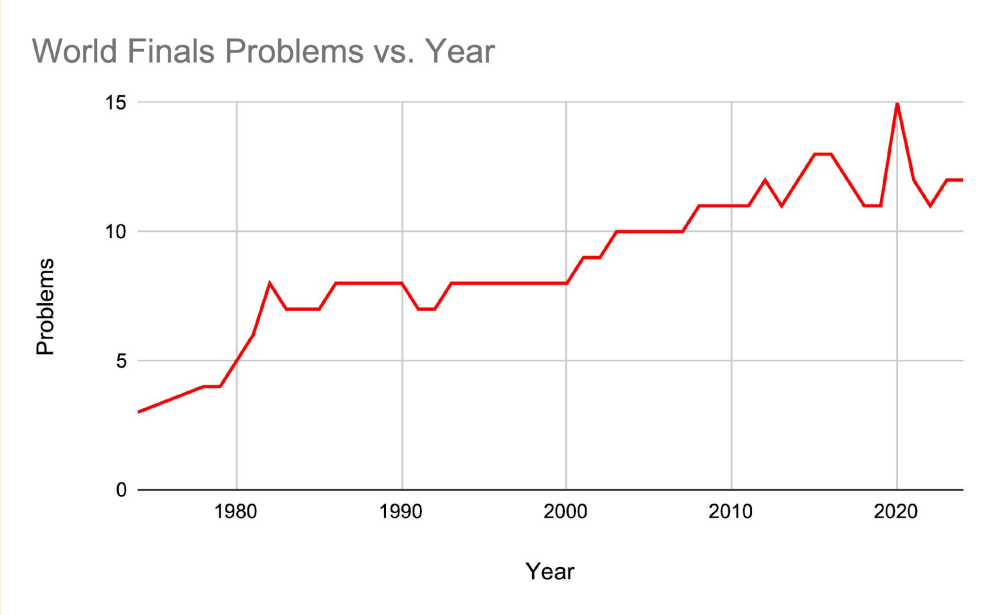
IO Using Interaction: Example

You are running an election and have gathered a list of ballots (candidate name strings). Given the **sorted** list of ballots, determine who (if anyone) won a majority (>50%) of the votes.

- **Constraints**: O(1) space, O(log N) time
- **Harder version**: unsorted list, O(1) space, O(N) time

- Other possibilities: emulating…
  - huge files or filesystems
  - databases that must be accessed with queries
  - networks with latency / path loss

Some Data

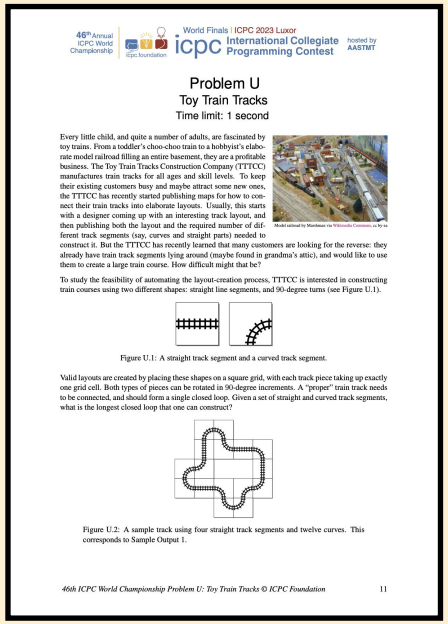


Proposal #2: Scoring Problems

- **Goal**: improve the contest experience for lower-ranking teams **without** significantly altering competition at the highest level or adding problems

- **Proposal**: make a few of the **medium-difficulty** problems **scored**
  - **AC** still earns 1 points
  - **TLE/RTE** earns 0 points
  - **WA** earns points in range [0,1)

Scoring Problem: Example



46th Annual ICPC World Championship | World Finals | ICPC 2023 Luxor | icpc International Collegiate Programming Contest | hosted by AASTMT

Problem U
Toy Train Tracks
Time limit: 1 second

Every little child, and quite a number of adults, are fascinated by toy trains. From a toddler's choo-choo train to a hobbyist's elaborate model railroad filling an entire basement, they are a profitable business. The Toy Train Tracks Construction Company (TTTCC) manufactures train tracks for all ages and skill levels. To keep their existing customers busy and maybe attract some new ones, the TTTCC has recently started publishing maps for how to connect their train tracks into elaborate layouts. Usually, this starts with a designer coming up with an interesting track layout, and then publishing both the layout and the required number of different track segments (say, curves and straight parts) needed to construct it. But the TTTCC has recently learned that many customers are looking for the reverse: they already have train track segments lying around (maybe found in grandma's attic), and would like to use them to create a large train course. How difficult might that be?

Model railroad by Mordnam via Wikimedia Commons, cc by-sa

To study the feasibility of automating the layout-creation process, TTTCC is interested in constructing train courses using two different shapes: straight line segments, and 90-degree turns (see Figure U.1).

Figure U.1: A straight track segment and a curved track segment.

Valid layouts are created by placing these shapes on a square grid, with each track piece taking up exactly one grid cell. Both types of pieces can be rotated in 90-degree increments. A "proper" train track needs to be connected, and should form a single closed loop. Given a set of straight and curved track segments, what is the longest closed loop that one can construct?

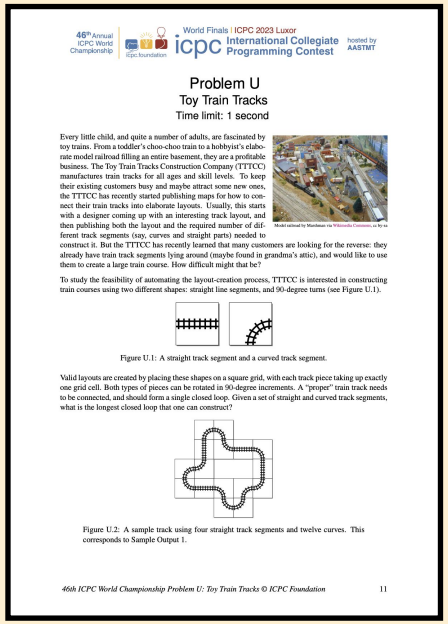
Figure U.2: A sample track using four straight track segments and twelve curves. This corresponds to Sample Output 1.

46th ICPC World Championship Problem U: Toy Train Tracks © ICPC Foundation 11

Toy Train Tracks, 2022 World Finals

Output a train loop using at most **s** straight segments and **c** curved segments, that has the longest length (in number of track segments used) under this restriction. The loop must be closed and cannot intersect itself. If there are multiple loops of maximal length, any one of them will be accepted.

Scoring Problem: Example



46th Annual ICPC World Championship | World Finals | ICPC 2023 Luxor | icpc International Collegiate Programming Contest | hosted by AASTMT

Problem U
Toy Train Tracks
Time limit: 1 second

Every little child, and quite a number of adults, are fascinated by toy trains. From a toddler's choo-choo train to a hobbyist's elaborate model railroad filling an entire basement, they are a profitable business. The Toy Train Tracks Construction Company (TTTCC) manufactures train tracks for all ages and skill levels. To keep their existing customers busy and maybe attract some new ones, the TTTCC has recently started publishing maps for how to connect their train tracks into elaborate layouts. Usually, this starts with a designer coming up with an interesting track layout, and then publishing both the layout and the required number of different track segments (say, curves and straight parts) needed to construct it. But the TTTCC has recently learned that many customers are looking for the reverse: they already have train track segments lying around (maybe found in grandma's attic), and would like to use them to create a large train course. How difficult might that be?

Model railroad by Mordnam via Wikimedia Commons, cc by-sa

To study the feasibility of automating the layout-creation process, TTTCC is interested in constructing train courses using two different shapes: straight line segments, and 90-degree turns (see Figure U.1).

Figure U.1: A straight track segment and a curved track segment.

Valid layouts are created by placing these shapes on a square grid, with each track piece taking up exactly one grid cell. Both types of pieces can be rotated in 90-degree increments. A "proper" train track needs to be connected, and should form a single closed loop. Given a set of straight and curved track segments, what is the longest closed loop that one can construct?

Figure U.2: A sample track using four straight track segments and twelve curves. This corresponds to Sample Output 1.

46th ICPC World Championship Problem U: Toy Train Tracks © ICPC Foundation 11

Toy Train Tracks, 2022 World Finals

Output a train loop using at most **s** straight segments and **c** curved segments, that has the longest length (in number of track segments used) under this restriction. The loop must be closed and cannot intersect itself. If there are multiple loops of maximal length, any one of them will be accepted.

You may output a shorter loop for partial credit. If the loop of maximum length has length **M**, a valid loop of length **L** earns **L / M** points.

Scoring Problems: How it Works

- Top-tier teams will immediately **AC** the problem and earn one point. To these teams, there is no difference to the current problem format

Scoring Problems: How it Works

- Top-tier teams will immediately **AC** the problem and earn one point. To these teams, there is no difference to the current problem format
- Lower-tier teams can earn partial credit throughout the contest
  - improving the score gives them something to grind on when stuck
  - they get incremental rewards each time they see their number go up on the scoreboard
  - the Resolver becomes more exciting at the lower rankings

Scoring Problems: How it Works

- Top-tier teams will immediately **AC** the problem and earn one point. To these teams, there is no difference to the current problem format
- Lower-tier teams can earn partial credit throughout the contest
  - improving the score gives them something to grind on when stuck
  - they get incremental rewards each time they see their number go up on the scoreboard
  - the Resolver becomes more exciting at the lower rankings
- Crucially, there must be **no benefit** (in terms of score or penalty) to partially solving a problem when a team could fully solve instead!

Scoring Problems: Some Details

- How to incorporate partial credit is problem-type-specific:
  - IOI-style test groups are possible (but I think it's best to keep ICPC a distinct style relative to IOI!)

Scoring Problems: Some Details

- How to incorporate partial credit is problem-type-specific:
  - IOI-style test groups are possible (but I think it's best to keep ICPC a distinct style relative to IOI!)
  - constructive problems: allow suboptimal solutions
  - interactive problems: start penalizing solutions using too many interactions

Scoring Problems: Some Details

- How to incorporate partial credit is problem-type-specific:
  - IOI-style test groups are possible (but I think it's best to keep ICPC a distinct style relative to IOI!)
  - constructive problems: allow suboptimal solutions
  - interactive problems: start penalizing solutions using too many interactions
  - allow partial credit for suboptimal approximate solutions
  - allow partial credit for solving small instances of the problem
  - allow partial credit based on number of corner cases handled

Scoring Problems: Time Penalty

- **Status quo**: every (non-**CE**) submission before the first **AC** earns a penalty
  - there is no penalty if you never solve the problem
  - there is no penalty for additional submissions after **AC**

Scoring Problems: Time Penalty

- **Status quo**: every (non-**CE**) submission before the first **AC** earns a penalty
  - there is no penalty if you never solve the problem
  - there is no penalty for additional submissions after **AC**
- **Proposal**: every (non-**CE**) submission before the highest-scoring positive-score submission earns a penalty
  - there is (still) no penalty if you never (even partially) solve the problem
  - there is no penalty for additional submissions after your highest-scoring attempt
  - there **is** a penalty for partial solutions that are later improved

An introduction to problemsetting

Joshua Andersson

Problemsetting

- Word not in the dictionary
- Approximately means: inventing and implementing problems. Assembling them into a contest
- Why?
  - Fun!
  - Engage with the community, get invited to contests
  - Educational- improve problem-solving

My credentials

- Head judge for Swedish IOI selection for 2 years
- Solved over 2000 problems
- Created over 80 problems
- “Touched” over 400 problems
- I have agreeable opinions :)

Contents of the talk

- How do you define problem quality?
- How to invent (high-quality) problems?
- How to implement the problem for a judge
- How to select a good problemset

Problem idea quality

Quality is highly subjective. Even depends on target audience.

For weaker contestants:

- The statement should be clear and natural
- Is the problem “fun”?
- Being standard is not bad- often happy for being able to use their knowledge
- Avoid floating-point, fast input parsing, …

Problem idea quality

For stronger contestants:

- Should bring a new idea
  - Problem should not just be a single technique or data structure
- “Beautiful”
- Should not: appeared in previous contest, in OEIS, Google, be cheesable by weird heuristic/bitset/SIMD (pragma)

Apart from there, it’s more subjective:

- Some value short implementations, some like heavy theory, other dislike it. Know your audience! (Look at previous editions of the contest)

Inventing problems

Method 1:
Think of some original problem (often from everyday situation) and try to solve it. Usually requires changing parts of the problem (easier/harder). More work, but **usually produces the best problems**.

Method 2:
Take solution/insight, turn it into a problem. Can turn out artificial/boring, easier solutions might exist.

Inventing problems

Method 3:
Take existing problem/exercise/paper, modify it. Can turn out good, but usually less original.

Inventing problems

Stranger methods:

- Misread problem statement
- Read problem name, try to guess what the problem is
- Start with NP/O(N^2) problem, see if it's solvable by adding constraint
- Look for inspiration: Google maps, image database, large list of common setups (grid, graph, permutation, …)
- "Problemsetting tricks", like "reversing" a problem: contestant has to generate input

Inventing problems

Practical tips:

- Write down all your ideas in central place, even “bad” ones
  - Looking through old, discarded ideas after a long time- fine wine
- Solve lots of problems!
  - Read lots of high-quality problems: ICPC WF, regionals, IOI, BOI, CEOI, JOI…..
- If you can’t solve problem, ask for help (don’t overuse)

Problem implementation

Multiple parts:

- Clear, unambiguous statement
- Correct judge solution
- Strong and correct test data
- Sensible limits

Problem implementation

Statement

A statement should:

- Prioritize clarity above all else (flavor)
- Define unknown terms (cactus graph, morphism,...)
- Use correct grammar, common and precise phrases (LLMs are good spellcheckers)
- Describe all constraints

Good to have:

- Examples, with explanations
- Illustrations when relevant (tikz, google slides, graphviz)
- Fun story (don't force)

Problem implementation

Judge solution

The judge solution *must* be correct

- Write multiple solutions, slow solutions, then stress-test
- Write carefully, anticipate errors- int overflow, floating point precision, edgecases
- Prove correctness
- Should be solved by at least 2 different judges

Problem implementation

Test data

Correctness:

- Write a program that validates all constraints from the statement (sync these!!)

Strength:

- Multiple solvers, multiple approaches, kill incorrect ones
- Sanity-check the data: visualize it, ensure outputs are diverse
- Ask LLM to solve it using greedy or weird approach
- Anticipate cheeses

Problem implementation

Sensible limits

- Allow a margin of at least ~2x compared to the judge solution
  - Don’t write it too optimized: avoid super fast input/library code, SIMD
- Set limits as low as possible while confident you fail incorrect solutions
- Write judge solutions for all allowed languages
- Avoid giving higher TL to Python.. it can run C++

Problemset

- Know your audience
- Make it enjoyable for both strong and weak
  - It's rarely bad to add more easy problems
- Try to cover broad set of techniques/ideas
  - … But better to have a 2nd DP problem that is good than a bad problem

Problemset

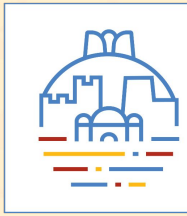
- Smooth difficulty curve: every contestant/team should be 1-2 hour away from solving another problem
- No team solves everything, every problem solved by someone

Where to begin?

- Get in contact with community! Usually, the lowest bar is in national IOI selection. Get in contact! Usually short-staffed
  - … or hold a contest for your school/university yourself!
- Try a contest preparation system- Polygon, Kattis Problemtools, CMS, …
- Submit problems to smaller-scale contests

Questions?

joshua.andersson@kodsport.se



ICPC 2025
WORLD FINALS BAKU
HOSTED BY ADA UNIVERSITY

Uses of the new Problem Format

Rasmus Hulthe

What is the Problem Format?

- Needed to systematically define problems
- Standardised integration with contest systems
- Widely supported:
  - Kattis
  - DOMjudge
  - PC^2
  - Community developed tools

Changes and new features

- Removed scoring objective; this now always is "maximize".
- Removed `scoring` keyword from `problem.yaml`.
- Python 3 is now used for `.py` files; for backward compatibility `.py2` can still be used for Python 2.
- Clarify various things: sample files for interactive problems, `testdata.yaml` inheritance.
- Updated the CC BY-SA license key to mean version 4.0.
- Allow either a build or run script to be present.
- Change specification of time limit multipliers and allow to explicitly specify a problem time limit.
- Add a multi-pass problem type.
- Support invalid testdata that validators must fail on.
- Only allow a single output validator, remove `validator_flags` from `problem.yaml`, update `{input,output}_validator_flags` in `testdata.yaml`.
- Make `name` required and allow a map from language code to name in that language.
- Add `uuid` to `problem.yaml`.
- Add `languages` to `problem.yaml`.
- Add support for Markdown problem statements.
- Change `languages:` and `keywords:` in `problem.yaml` to be lists of strings rather than a string of space separated words.
- Clarified when code limit is applied in the case of included code
- Make `uuid` required in `problem.yaml`.
- Add `/submissions/rejected` directory.
- Clarify working directories for submissions and validators.
- Add `allow_writing_files` to `problem.yaml`.
- Rename `testdata.yaml` to `test_group.yaml`.

- Multi-pass
- Static validators
- Submit answer

General overview of the changes

- Statements can now be written in Markdown
- Write-ups for each problem
- Allowed languages can be configured
- Time limits can now be explicitly specified

General overview of the changes

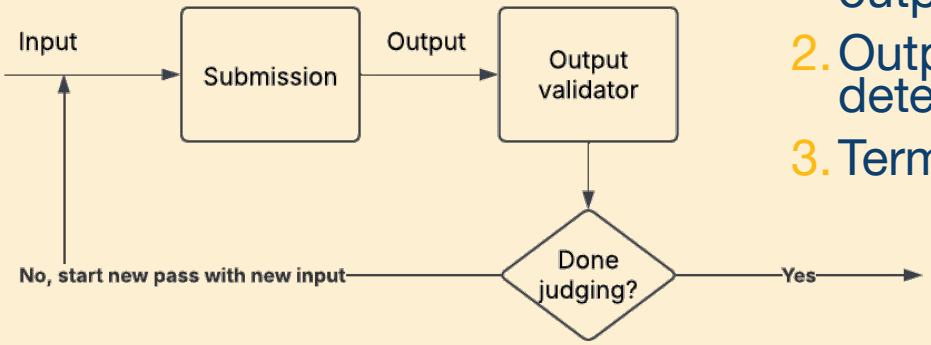
- Custom graders are completely removed
- Better structure to problem solutions
- More ways to credit problem developers
- Visualizers for input and output

Output validators

- Judges submissions
- Responsible for judgement and feedback
- *Must* now be written in C++ or Python3
- Default validator has been improved

ICPC International Collegiate Programming Contest
The 2025 ICPC World Finals Baku
31 August – 5 September 2025 // hosted by ADA University

Multi-pass



1. Submission run on initial input or output validator input
2. Output validator validates and determines whether to rerun
3. Terminates with verdict **OR** jumps to 1

- **Multi-pass**
- Static validators
- Submit answer

Communication problem

Encrypt+Decrypt, one program, two people

- Bob needs to send a message to Alice and gets to add some extra bits of information
- Some bits of the message are flipped in transit
- Alice has to point out which bits got flipped, and what the original message was

- **Multi-pass**
- Static validators
- Submit answer

Flight to the Ford

- BOI 2022, day 2, communication
- Multi-pass + interactive
- Develop a messaging system
  - encode(),decode()
  - Two bits will not flip in a row
- $3 \le N \le 10^9$

ICPC International Collegiate Programming Contest
The 2025 ICPC World Finals Baku
31 August – 5 September 2025 // hosted by ADA University

Static validator

- Validates based on the ***source code*** rather than ***output***
  - Code golf
- Source code parsing is very complex
- Useful within academia

ICPC International Collegiate Programming Contest
The 2025 ICPC World Finals Baku
31 August – 5 September 2025 // hosted by ADA University

Recursive fibonacci problem

- Teaching recursion using the fibonacci sequence
- Widely supported:
  - Print the fibonacci sequence up to 10^5
  - *for* and *while* loops are disallowed
- Use static validation to *WA* any submission containing either ‘for’ or ‘while’

- Multi-pass
- **Static validators**
- Submit answer

The pylint problem

- Check if integer is prime
- If valid solution, score is pylint score

33.3p

```python
def p(n):
    if n <= 1:return False
    if n == 2:return True
    if n % 2 == 0:return False
    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:return False
    return True
try:print(p(int(input())))
except Exception:print(False)
```

```
************* Module prime
prime.py:9:0: C0304: Final newline missing (missing-final-newline)
prime.py:1:0: C0114: Missing module docstring (missing-module-docstring)
prime.py:1:0: C0116: Missing function or method docstring (missing-function-docstring)
prime.py:2:14: C0321: More than one statement on a single line (multiple-statements)
prime.py:3:14: C0321: More than one statement on a single line (multiple-statements)
prime.py:4:18: C0321: More than one statement on a single line (multiple-statements)
prime.py:6:22: C0321: More than one statement on a single line (multiple-statements)
prime.py:9:7: W0718: Catching too general exception Exception (broad-exception-caught)
prime.py:8:4: C0321: More than one statement on a single line (multiple-statements)
prime.py:9:17: C0321: More than one stat                    multiple-statements)
-------------------------------------------------------------------
Your code has been rated at 3.33/10
```

3.33/10

75p

```python
def p(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:
            return False
    return True
print(p(int(input())))
```

```
************* Module pylinted
pylinted.py:12:0: C0304: Final newline missing
pylinted.py:1:0: C0114: Missing module docstring
pylinted.py:1:0: C0116: Missing function or method docstring

-------------------------------------------------------------------
Your code has been rated at 7.50/10
```

100p

```python
"""Module to check if a number is prime."""

def p(n):
    """Return True if n is a prime number, else False."""
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:
            return False
    return True

print(p(int(input())))

-------------------------------------------------------------------
Your code has been rated at 10.00/10
```

ICPC International Collegiate Programming Contest
The 2025 ICPC World Finals Baku
31 August – 5 September 2025 // hosted by ADA University

Use static validation with caution

- Tempting to forbid unintended solutions
  - Difficult to parse every way to write a data structure
  - More difficult in contests supporting multiple languages

- Multi-pass
- Static validators
- **Submit answer**

Submit answer

- Submitting an answer, not code
- Beginner friendly
- Lower server load
  - Faster grading

Problem Format adaptation

- Final details still being finalised
- Implementation is on-going
- Why adopt the new format?
  - Opens up many new/interesting problems
  - Simplifies problem implementation

Any questions?

rasmus.hulthe@kodsport.se

ICPC ARCHIVE

Miguel Revilla Rodríguez
archive@icpc.global

Why an ICPC Archive?

- ICPC's legacy is scattered across sites, drives, and personal folders.
- Risk of loss: link rot, platform shutdowns, format obsolescence.
- Hard to discover, cite, and reuse materials at scale.

Vision and Scope

- A living, open, **authoritative** archive of ICPC's global history.
- Serve researchers, organizers, coaches, students, and media.
- Modern UX: maps, timelines, galleries, APIs.

What We Preserve (Beyond Problem Sets)

- Contest artifacts: schedules, posters, programs, badges, websites.
- Media: photos, videos, livestreams, press coverage.
- People and teams: rosters, bios, testimonies, interviews.
- Venues and hosts: locations, institutions, logistics documents.
- Community outputs: training materials, talks, postmortems, papers.

Archiving is not Storing

- Not every PDF is archival.
- Not every image is equal.
- Not every video “just works”.
- If it’s not described, it’s effectively lost (metadata or it didn’t happen).
- Also, `final_final2_reallyFINAL.pdf` is not a metadata strategy.

Archival Formats — Documents & Images

- Documents: PDF/A-2b or A-3u; fonts embedded; no encryption; XMP metadata.
- Images (masters): TIFF 6.0 (lossless LZW/ZIP); retain EXIF/IPTC/XMP and color profiles.
- Graphics/figures: PNG (lossless) or SVG (vector); PDF/A for print-ready.
- JPEGs are great for memes; masters deserve TIFF/PNG.

Archival Formats — Audio, Video, Data

- Audio: WAV (PCM) or FLAC for masters; MP3/AAC as access copies.
- Video: retain original mezzanine; access MP4 (H.264 + AAC); subtitles as WebVTT/SRT.
- Data: CSV (RFC 4180, UTF-8) or JSON/JSON Lines; include schema and README.

Preservation Workflow

- Ingestion checks: MIME validation, duplicate detection, required fields.
- Checksums: SHA-256 on upload; scheduled fixity verification.
- OCR for scanned PDFs; text extraction powers search and snippets.
- Versioning: resources & metadata with provenance and audit logs.
- Derivatives: access-friendly copies generated where appropriate.

CKAN & Architecture

- Why CKAN: open-source, robust, dataset/resource model; orgs, groups, roles; facets, APIs, previews, harvesting.
- Stack: CKAN core + PostgreSQL + Solr/Elastic; FileStore (local/S3); DataStore for tabular/JSON.
- Extensions: scheming, spatial, harvest, archiver, QA, DCAT; theming, analytics, monitoring, backups.

What's Already Done — Platform

- Pilot CKAN instance live with public API.
- ICPC theme: colors, typography, and logo integrated.
- Home UX with map, highlights, and rich cards.
- S3-backed storage configured for scalable media.

What's Already Done — Schemas

- Custom schemas via ckanext-scheming: Events, Media, Teams.
- Core fields: edition, year, host, city/country, geo, media type, rights.
- Controlled vocabularies: event types, roles, formats, languages.
- Persistent IDs (UUIDs) and provenance fields implemented.

Already Done — Ingestion & Harvesting

- Bulk imports (CSV/JSON) for historical editions.
- `ckanext-harvest` prototypes: GitHub releases, YouTube playlists.
- Automated enrichment: geocoding, tag normalization, people/event linking.
- Link checking via `ckanext-archiver` and quality via `ckanext-qa`.

Rights, Licenses, and Ethics

- License fields enforced; CC licenses encouraged by default.
- Rights holder, source URL, and citation captured systematically.
- Consent workflow for testimonies/interviews; takedown policy drafted.
- “Found on the internet” is not a license (we checked).

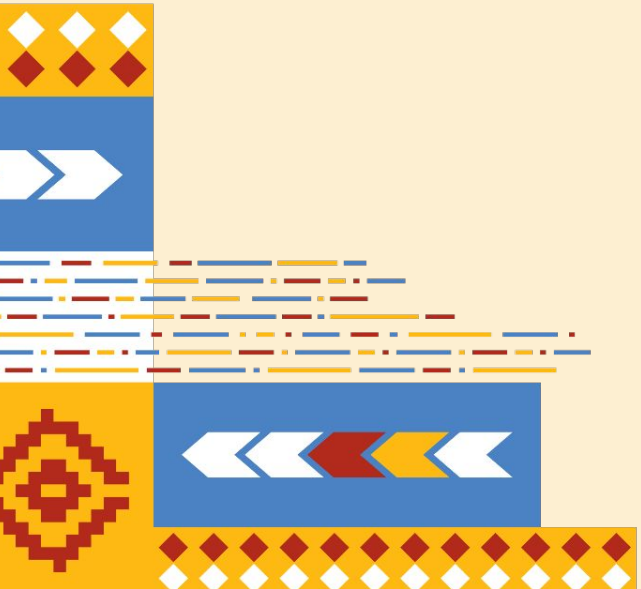
Contributions and Governance

- Roles: submitter, editor, reviewer, curator (RBAC in CKAN).
- Submission forms & bulk templates with validation.
- Review checklist for metadata quality and rights.
- Contributor recognition: attributions and featured showcases.

Roadmap — Next Milestones

- Expand ingestion across regions; fill gaps in early editions.
- Oral histories: interviews and annotated timelines.
- Advanced search: people/teams graph and related stories.
- Partner ingest: regional sites, universities, media teams.

archive@icpc.global



Thank you!

Thank you & Feedback Survey!

- Thank you for coming to the final session of **2025 ICPC CLI Symposium**!
- I hope to see you again in **2026 ICPC CLI Symposium**!
- Please send me a speaker suggestion at christian.lim@icpc.global.