

# Enabling open and reproducible research at computer systems' conferences

*the good, the bad and the ugly*

cTuning.org / ae

fursin.net / research

**Grigori Fursin**

Chief Scientist, cTuning foundation, France  
CTO, dividiti, UK

**CNRS webinar**  
Grenoble  
March 2017

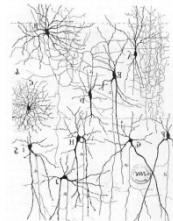
# Seminar outline

- What is computer systems research?
- Major problems in computer systems' research in the past 15 years
- Artifact Evaluation Initiative
  - **Good:** active community participation with ACM/industrial support
  - **Bad:** software and hardware chaos; highly stochastic behaviour
  - **Ugly:** ad-hoc, non-portable scripts difficult to customize and reuse
- Improving Artifact Evaluation
  - Preparing common replication/reproducibility methodology (new ACM taskforce on reproducibility)
  - Introducing community-driven artifact and paper reviewing
  - Introducing common workflow framework (Collective Knowledge)
  - Introducing simple JSON API and meta for artifacts
  - Introducing portable and customizable package manager
- Demonstrating open and reproducible computer systems' research
  - Collaboratively optimizing deep learning across diverse datasets/SW/HW
- Conclusions and call for action!

# Back to basics: what is computer systems' research?



Users of computer systems (researchers, engineers, entrepreneurs) want to quickly prototype their algorithms or develop efficient, reliable and cheap products



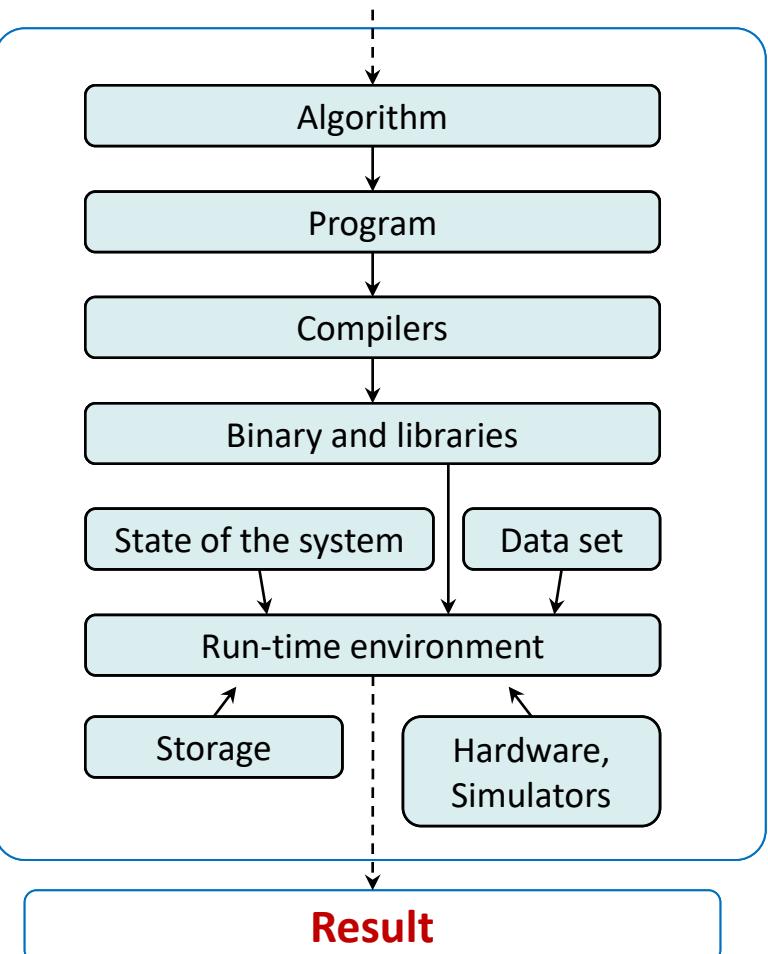
Computer System

Result

# Back to basics: what is computer systems' research?



Users of computer systems (researchers, engineers, entrepreneurs) want to quickly prototype their algorithms or develop efficient, reliable and cheap products



**Computer systems' researchers, software developers and hardware designers help end-users**

**improve all characteristics of their tasks**  
(execution time, power consumption, numerical instability, size, faults, price ...)

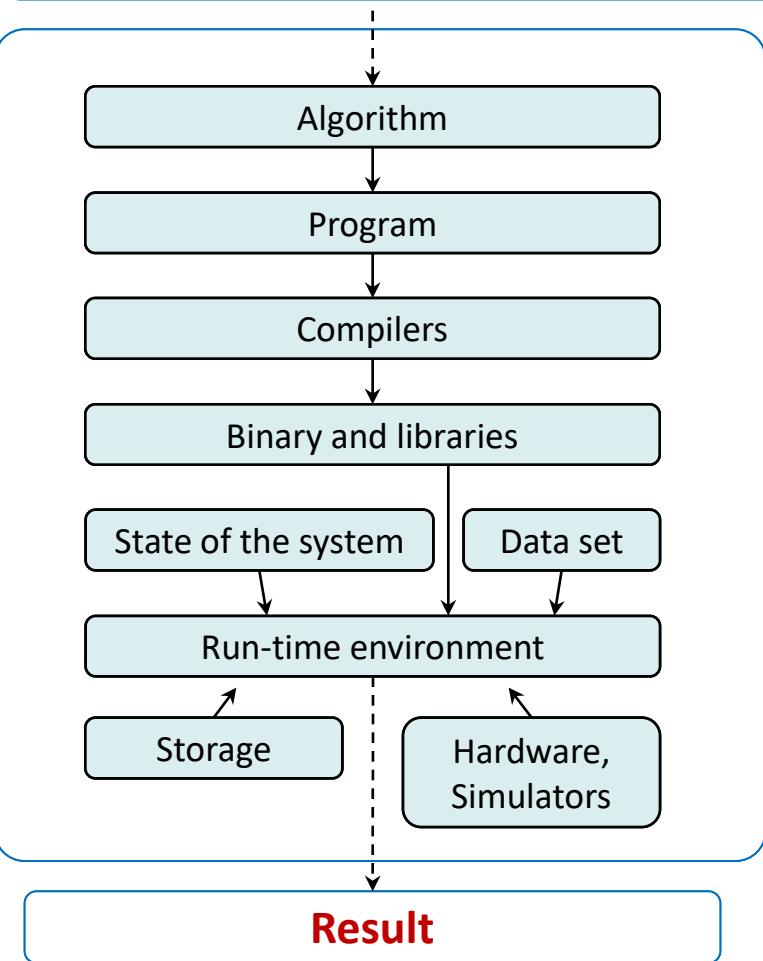
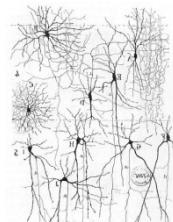
**guarantee real-time constraints**  
(bandwidth, QoS, etc)



# Two major problems: raising complexity and physical limitations



Finding efficient, reliable and cheap solution  
for end-user tasks is very non-trivial!



Thousands of benchmarks real applications  
MPI, OpenMP, TBB, CUDA, OpenCL, StarPU, OmpSs  
C,C++,Fortran,Java,Python,assembler  
LLVM,GCC,ICC,PGI (hundreds of optimizations)  
BLAS,MAGMA,ViennaCL,cuBLAS,clBLAST,cuDNN,  
openBLAS, cIBLAS  
TensorFlow, Caffe, Torch, TensorRT  
Infinite number of possible data sets  
Linux, Windows, Android, MacOS  
heterogeneous, many-core, out-of-order, cache  
x86, ARM, PTX, NN, extensions  
Numerous architecture and platform simulators

Too many design and optimization choices!

# What are you paying for?

Users expect new platforms to be faster, more energy efficient  
more accurate and more reliable – **is it true?**

*Supercomputers and data centers cost millions of \$ to build, install and use*



Weather prediction;  
physics; medicine; finances



Smart and powerful mobile devices are everywhere (mobile phones, IoT)



Many attempts to run DNN,  
HOG, Slam algorithms



Unchanged algorithms may run only a fraction of peak performance thus wasting expensive resources and energy!

Some years later you may get more efficient algorithms just before new systems arrives!

Various SW/HW optimizations may result in

7x speedups, 5x energy savings, but poor accuracy

2x speedups without sacrificing accuracy – enough to enable RT processing

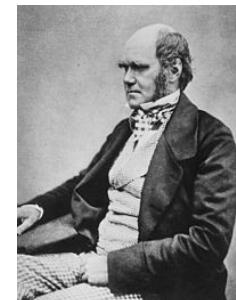
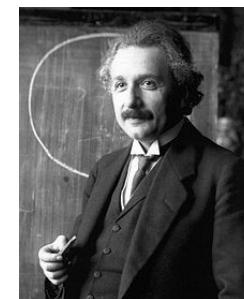
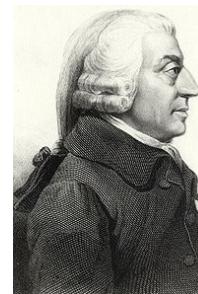
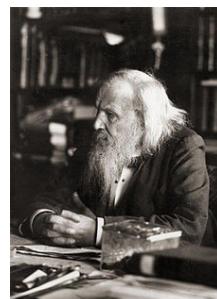
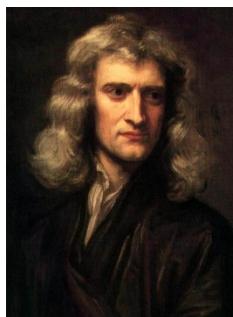
**New systems require further tedious, ad-hoc and error-prone optimization.  
This slows down innovation and development of new products.**

# How can we solve these problems?

My original background is NOT in computer engineering  
but in physics, electronics and machine learning

(first research project in 1994 to develop artificial neural networks chip)

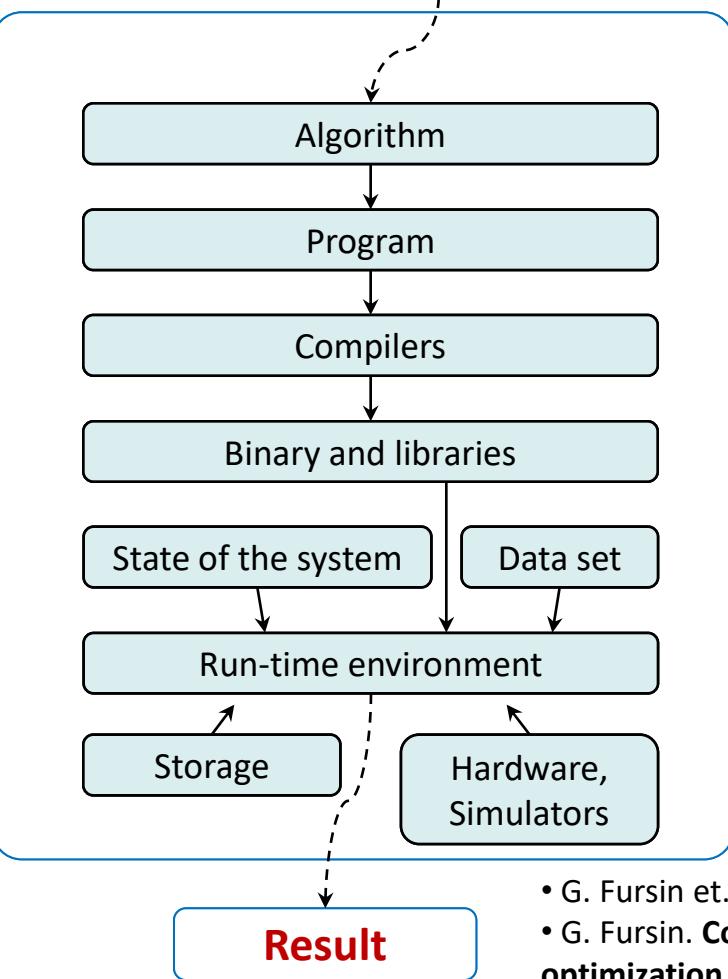
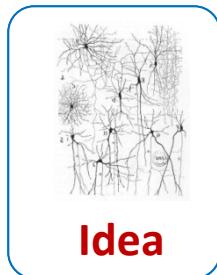
What did I learn from other sciences that deal with complex systems:  
*physics, mathematics, chemistry, biology, AI ...?*



Major breakthroughs came from collaborative and reproducible R&D based on

**statistical analysis,  
data mining,  
machine learning**

**sharing, validation, systematization  
and reuse of artifacts and knowledge!**



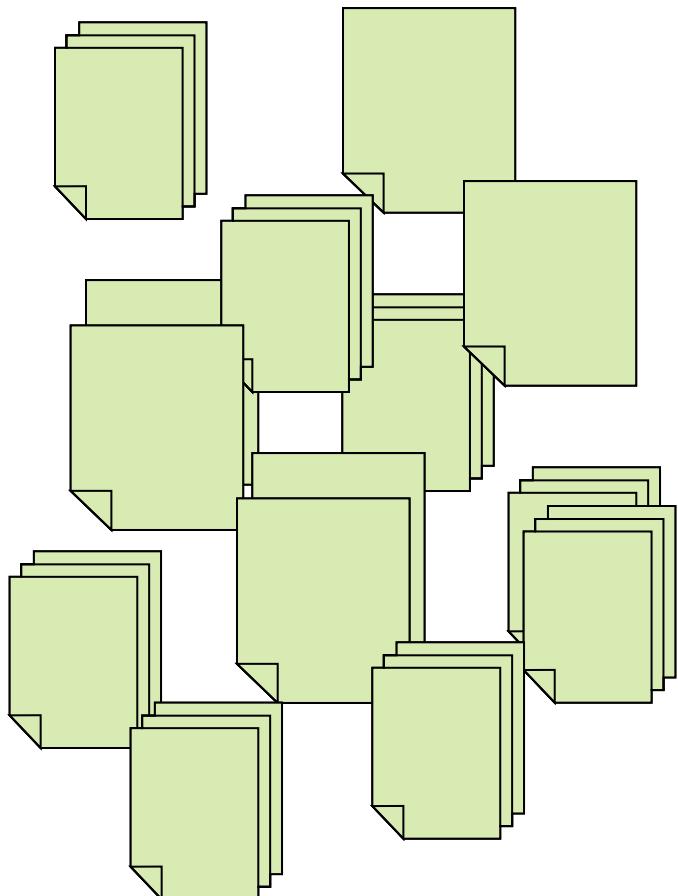
## cTuning<sub>1</sub> framework and public portal to

- 1) share realistic benchmarks and data sets
- 2) share whole experimental setups  
(benchmarking and autotuning)
- 3) crowdsource empirical optimization
- 4) collect results in a centralized repository
- 5) apply machine learning to predict optimizations
- 6) involve the community to improve models



- G. Fursin et.al. MILEPOST GCC: Machine learning based self-tuning compiler. 2008, 2011
- G. Fursin. Collective Tuning Initiative: automating and accelerating development and optimization of computing systems, 2009

## Numerous publications



## My personal AE goals



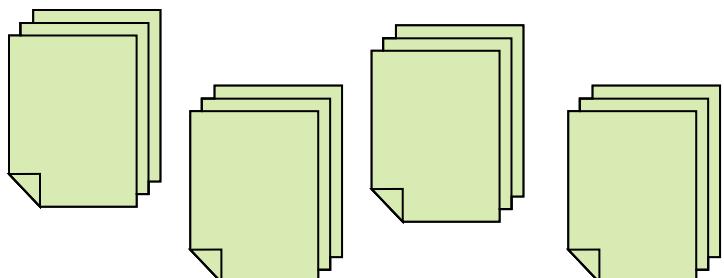
- validate experimental results from published articles and restore trust (see ACM TRUST'14 @ PLDI workshop: [cTuning.org/event/acm-trust2014](http://cTuning.org/event/acm-trust2014))
- promote artifact sharing (benchmarks, data sets, tools, models)
- enable fair comparison of results and techniques
- develop common methodology for reproducible computer systems' research
- build upon others' research

“Artifact Evaluation for Publications” (Dagstuhl Perspectives Workshop 15452), 2016,  
Bruce R. Childers, Grigori Fursin, Shriram Krishnamurthi, Andreas Zeller,  
<http://drops.dagstuhl.de/opus/volltexte/2016/5762>

# How Artifact Evaluation (AE) works?



PC members nominate one or two senior PhD students/engineers for AE committee



Authors of accepted articles has an option to submit related material for an AE committee to be evaluated

<http://ctuning.org/ae/reviewing.html>

Formalized reviewing process

Multiple criteria for artifact evaluation

Artifact ranking:

1. Significantly exceeded expectations
2. Exceeded expectations
3. Met expectations
4. Fell below expectations
5. Significantly fell below expectations

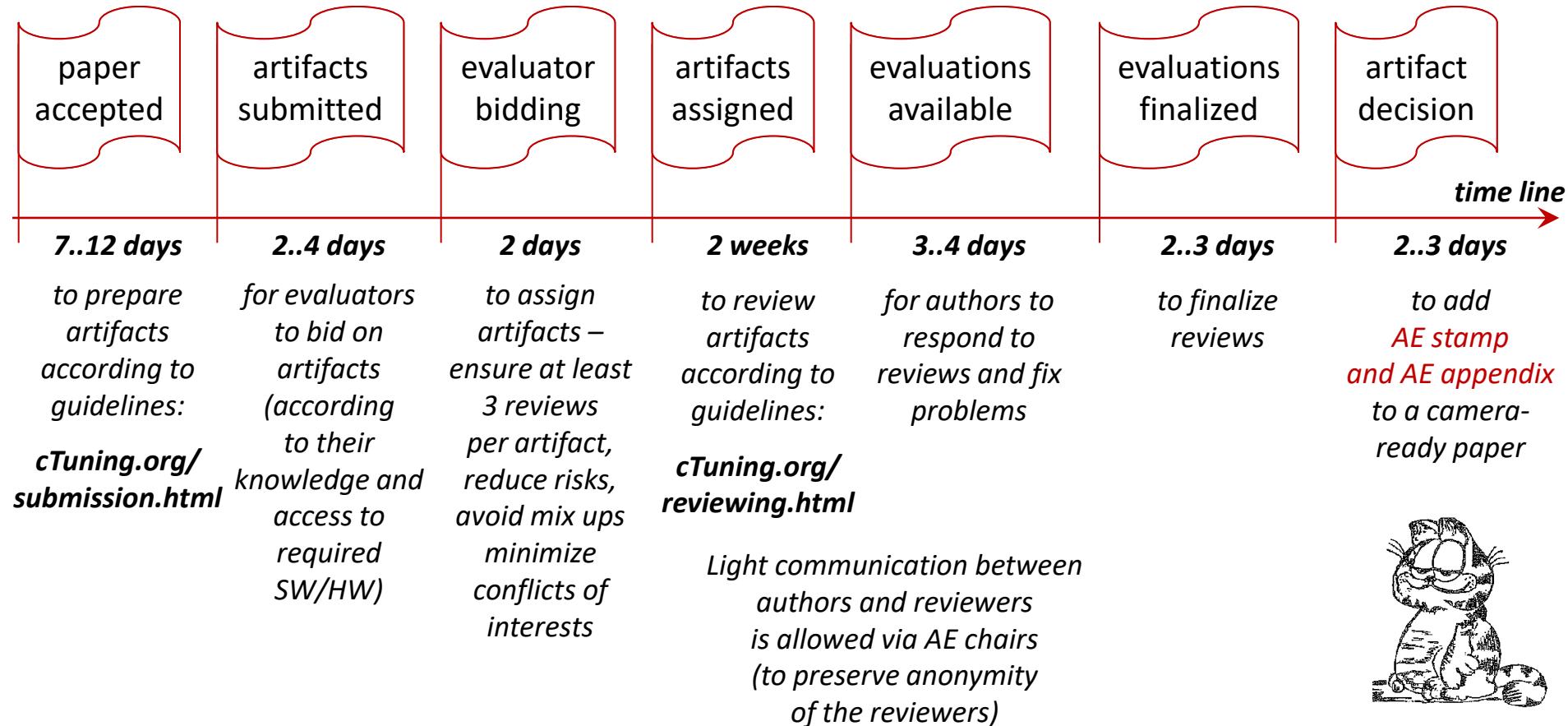
Paper with artifacts which passed evaluation receives AE badge



<http://cTuning.org/ae/submission.html>

- Abstract
- Packed artifact (or remote access)
- Artifact Appendix (with a version to keep track of a methodology)

# Artifact evaluation timeline



# Artifact Evaluation: good

- Strong support from academia, industry and ACM
- Active participation in AE discussion sessions
- Lots of feedback
- Many interesting artifacts!

Year	PPoPP	CGO	PACT	Total	Problems	Rejected
2015	10	8		18	7	2 ☹
2016	12	11		23	4	0
2016			5	5	2	0
2017	14	13		27	7	0

**NOTE:** we consider AE a cooperative process and try to help authors fix artifacts and pass evaluation (particularly if artifacts will be open-sourced)

We use this practical experience (> 70 papers in the past 3 years!) to continuously improve common experimental methodology for reproducible computer systems' research

# ACM taskforce on reproducibility

In 2016 ACM organized a special taskforce (former AE chairs) to develop common methodology for artifact sharing and evaluation across all SIGS!

We produced “***Result and Artifact Review and Badging***” policy:

<http://www.acm.org/publications/policies/artifact-review-badging>

## 1) Define terminology

*Repeatability* (Same team, same experimental setup)

*Replicability* (Different team, same experimental setup)

*Reproducibility* (Different team, different experimental setup)

## 2) Prepare new sets of badges (covering various SIGs)

*Artifacts Evaluated – Functional*

*Artifacts Evaluated – Reusable*

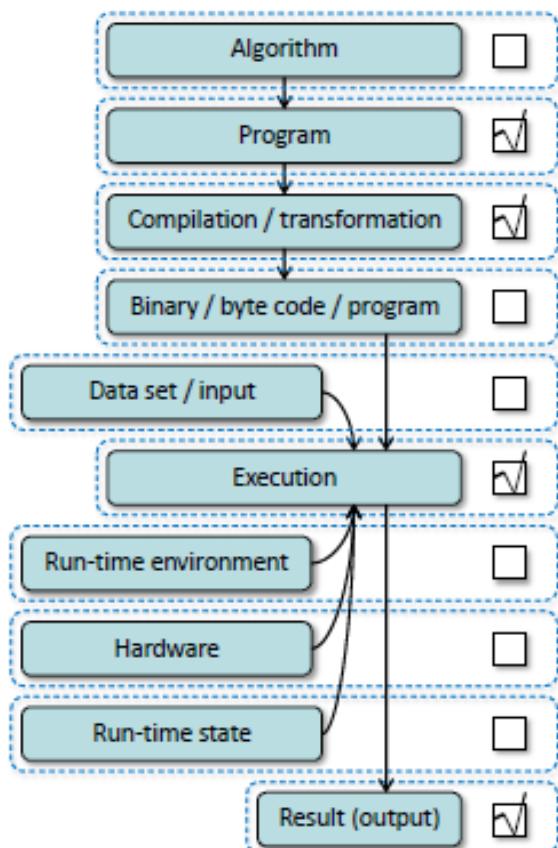
*Artifacts Available*

*Results Replicated*

*Results Reproduced*



# We introduced Artifact Appendices to describe experiments



Two years ago we introduced Artifact Appendix templates to unify Artifact submissions and let authors add up to two pages of such appendices to their camera ready paper:

<http://cTuning.org/ae/submission.html>

[http://cTuning.org/ae/submission\\_extra.html](http://cTuning.org/ae/submission_extra.html)

The idea is to help readers better understand what was evaluated and let them reproduce published research and build upon it.

We did not receive complaints about our appendices and many researchers decided to add them to their camera ready papers (see <http://cTuning.org/ae/artifacts.html>).

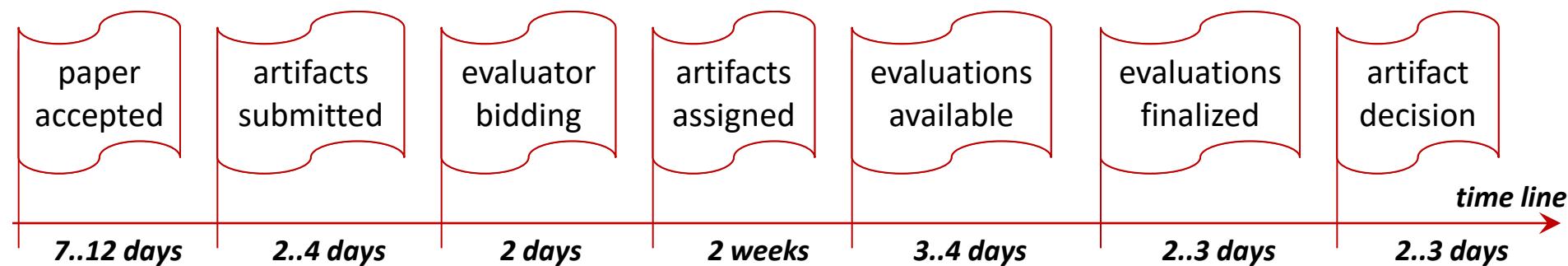
Similar AE appendices are now used by other conferences (SC, RTSS):

<http://sc17.supercomputing.org/submitters/technical-papers/reproducibility-initiatives-for-technical-papers/artifact-description-paper-title>

**We are now trying to unify Artifact Appendices across all SIGs**

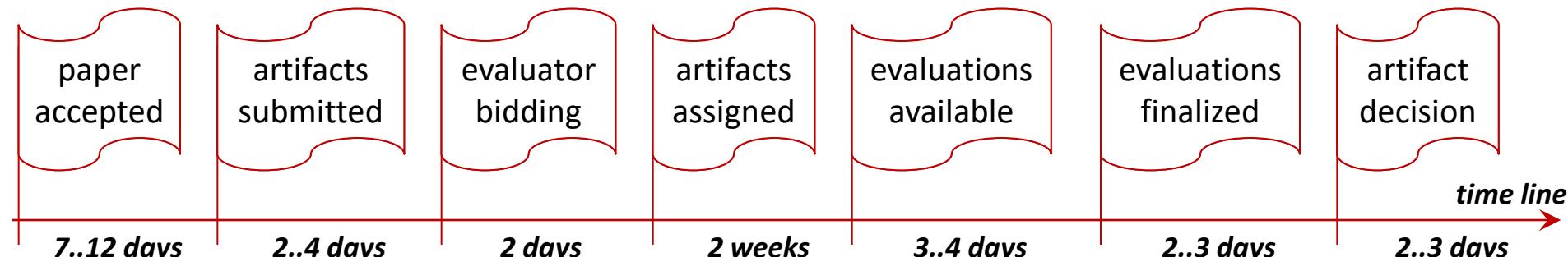
# Artifact Evaluation: bad

- too many artifacts to evaluate – need to somehow scale AE while keeping the quality (41 evaluators, ~120 reviews to handle during 2.5 weeks)
- difficult to find evaluators with appropriate skills and access to proprietary SW and rare HW
- very intense schedule and not enough time for rebuttals
- communication between authors and reviewers via AE chairs is a bottleneck

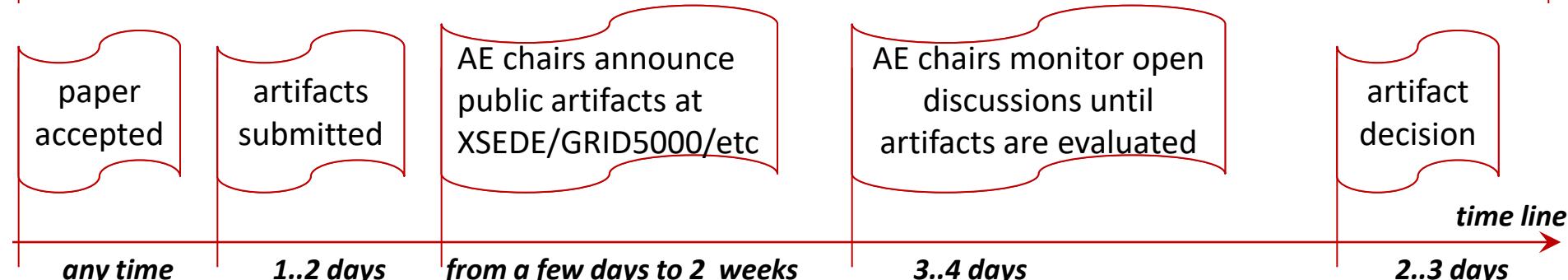


# New option of open artifact evaluation!

Introduce two evaluation options: private and public



a) traditional evaluation for private artifacts (for example, from industry, though less and less common)



b) open evaluation of public and open-source artifacts (if already available at GitHub, BitBucket, GitLab with “discussion mechanisms” during submission...)

# Trying open artifact and paper evaluation

At CGO/PPoPP'17, we have sent out requests to validate several open-source artifacts to the public mailing lists from the conferences, network of excellence, supercomputer centers, etc.

We found evaluators willing to help and having an access to rare hardware or supercomputers as well as required software and proprietary benchmarks

Authors quickly fixed issues and answered research questions while AE chairs steered the discussion!

**GRID5000 users participated in open evaluation of a PPoPP'17 artifact:**

**<https://github.com/thu-pacman/self-checkpoint/issues/1>**

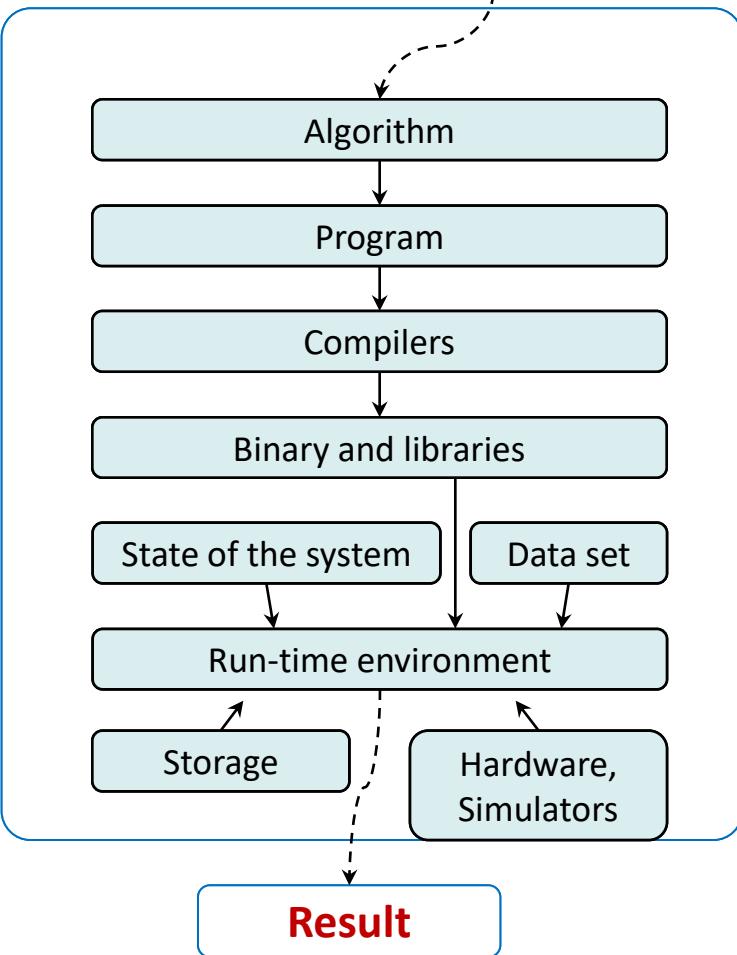
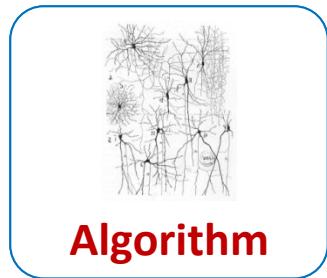
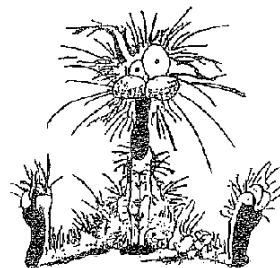
**See other public evaluation examples:**

**[cTuning.org/ae/artifacts.html](http://ctuning.org/ae/artifacts.html)**

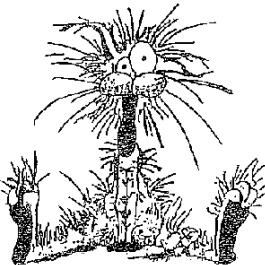
**We validated open reviewing of publications via Reddit at ADAPT'16:**

**<http://adapt-workshop.org>**

# Artifact Evaluation: what can possibly be ugly ;) ?



# Ugly: no common experimental methodology and SW/HW chaos



## Algorithm

CUDA 7.x LLVM 3.7 function-pass  
LLVM 3.4 ARM v8 MPI level reordering  
OpenMP ICC 11.1 OpenCL gprof  
ATLAS CUDA 4.x prof Intel SandyBridge  
LTO GCC 4.2.x per phase TBB  
GCC 5.2 LLVM 3.0 reconfiguration  
perf AVX MVS 2013 HMPP memory size  
GCC 4.5.x frequency polyhedral XLC  
execution time transformations IPA  
hardware GCC 4.3.x predictive Scalasca  
counters SSE4 program level scheduling ICC 11.0  
MKL bandwidth GCC 4.4.x GCC 4.6.x  
LLVM 2.6 SimpleScalar Codelet PAPI  
LLVM 2.9 ICC 12.1 Testarossa Jikes  
algorithm precision Amplifier Open64  
oprofile HDD size loop-level VTune  
GCC 4.7.x LLVM 2.7 SVM  
GCC 4.1.x algorithm- ICC 10.1 TAU  
Phoenix level

## Result

- difficult (sometimes impossible) to reproduce empirical results across ever changing software and hardware stack (highly stochastic behavior)
- everyone uses their own ad-hoc scripts to prepare and run experiments with many hardwired paths
- practically impossible to customize and reuse artifacts (for example, try another compiler, library, data set)
- practically impossible to run on another OS or platform
- no common API and meta information for shared artifacts and results (benchmarks, data sets, tools)

# Awarding distinguished artifacts –not enough!



The cTuning foundation and  
NVIDIA are pleased to present



## Joint CGO/PPoPP Distinguished Artifact Award

for

### “Demystifying GPU Microarchitecture to Tune SGEMM Performance”

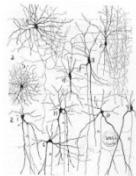
**Xiuxia Zhang<sup>1</sup>, Guangming Tan<sup>1</sup>, Shuangbai Xue<sup>1</sup>, Jiajia Li<sup>2</sup>, Mingyu Chen<sup>1</sup>**

<sup>1</sup> Chinese Academy of Sciences

<sup>2</sup> Georgia Institute of Technology

*February 2017*

# Using Virtual Machines and Docker



VM

VM or Docker images  
good at hiding complexity

but they do not solve  
major problems  
in computer systems' research

Result

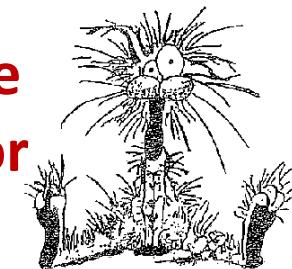
Docker is useful to archive artifacts  
while hiding all underlying complexity!

However Docker does not address many other  
issues vital for open and collaborative  
computer systems' research, i.e. how to

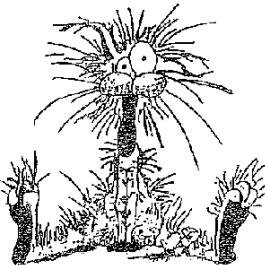
- 1) work with a native user SW/HW environment
- 2) customize and reuse artifacts and workflows
- 3) capture run-time state
- 4) deal with hardware dependencies
- 5) deal with proprietary benchmarks and tools
- 6) automate validation of experiments

Images are also often too large in size!

Need portable and customizable  
workflow framework suitable for  
computer systems' research!



# Some attempts to clean up this mess in computer systems' R&D



## Algorithm

CUDA 7.x LLVM 3.7 function-pass  
LLVM 3.4 ARM v8 MPI level reordering  
OpenMP ICC 11.1 OpenCL  
ATLAS CUDA 4.x prof gprof  
LTO GCC 4.2.x per phase TBB  
GCC 5.2 LLVM 3.0 reconfiguration  
perf AVX MVS 2013 HMPP memory size  
GCC 4.5.x frequency polyhedral XLC  
execution time transformations IPA  
hardware GCC 4.3.x predictive Scalasca  
counters SSE4 program level scheduling ICC 11.0  
MKL bandwidth GCC 4.4.x GCC 4.6.x  
LLVM 2.6 SimpleScalar Codelet PAPI Jikes  
LLVM 2.9 ICC 12.1 Testarossa Amplifier Open64  
algorithm precision oprofile HDD size loop-level VTune  
GCC 4.7.x LLVM 2.7 SVM ICC 10.1 TAU  
GCC 4.1.x algorithm-  
Phoenix level

## Result

## Third-party resources

cKnowledge.org/reproducibility

### Numerous online projects

*But we want to systematize our local artifacts without being locked up on third-party web services or learn complex GUI*

### Better package managers

*apt; yum; spack;  
pip ; homebrew ...*

### Smart build tools

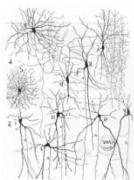
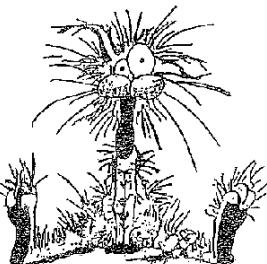
*cmake; easybuild ...*

### Multi-versioning

*spack; easybuild; virtualenv ...*

**Missing: portable and customizable workflow framework suitable for computer systems' research!**

# Open-source Collective Knowledge Framework (2015-cur.)



Algorithm

CUDA 7.x LLVM 3.7 function-pass  
LLVM 3.4 ARM v8 MPI level reordering  
OpenMP ICC 11.1 OpenCL  
ATLAS CUDA 4.x prof gprof  
LTO GCC 4.2.x per phase TBB  
GCC 5.2 LLVM 3.0 reconfiguration  
perf AVX MVS 2013 HMPP memory size  
GCC 4.5.x frequency polyhedral XLC  
execution time transformations IPA  
hardware GCC 4.3.x predictive ICC 12.0 Scalasca  
counters SSE4 program scheduling ICC 11.0  
MKL bandwidth GCC 4.4.x GCC 4.6.x  
LLVM 2.6 SimpleScalar Codelet PAPI  
LLVM 2.9 ICC 12.1 Testarossa Amplifier Jikes  
algorithm precision oprofile HDD size Open64  
GCC 4.7.x LLVM 2.7 SVM VTune  
GCC 4.1.x algorithm- ICC 10.1 TAU  
Phoenix level

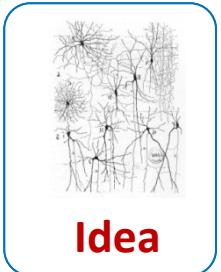
Result

Eventually we didn't have a choices but to use all our experience and develop portable and customizable workflow framework for computer systems' research:

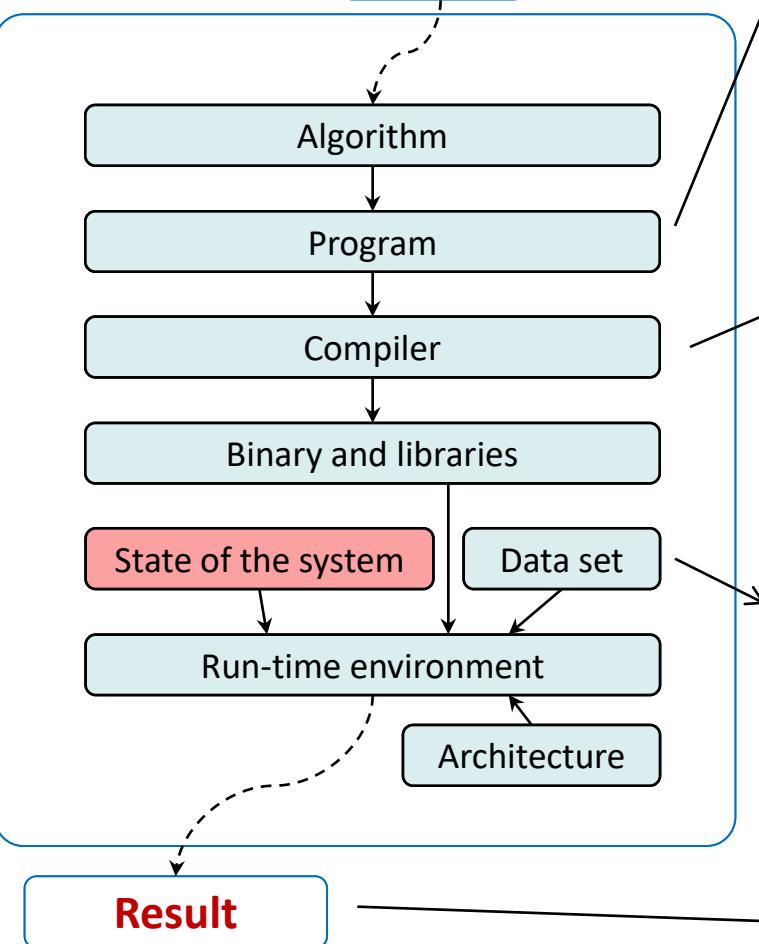
- 1) organize your artifacts (programs, data sets, tools, scripts) as customizable and reusable components with JSON API and meta data
- 2) assemble portable and customizable workflows with JSON API from shared artifacts as LEGO™
- 3) integrate portable package and environment manager which can detect multiple versions of required software or installed one across Linux, MacOS, Windows and Android
- 4) integrate web server to show interactive reports (locally!) or exchange data when crowdsourcing experiments

**cKnowledge.org ; github.com/ctuning/ck**

# Noticed during AE: all projects have similar structure



## Create project directory



Ad-hoc scripts to compile and run a program...

image corner detection

matmul OpenCL

compression

neural network CUDA

Have some common meta: which datasets can use, how to compile, CMD, ...

Ad-hoc scripts to set up environment for a given and possibly proprietary compiler

GCC V5.2

LLVM 3.6

Intel Compilers 2015

Have some common meta: compilation, linking and optimization flags

Ad-hoc dirs for data sets with some ad-hoc scripts to find them, extract features, etc

image-jpeg-0001

bzip2-0006

txt-0012

video-raw-1280x1024

Have some (common) meta: filename, size, width, height, colors, ...

Ad-hoc dirs and scripts to record and analyze experiments

cvs speedups

txt hardware counters

xls table with graph

Have some common meta: features, characteristics, optimizations

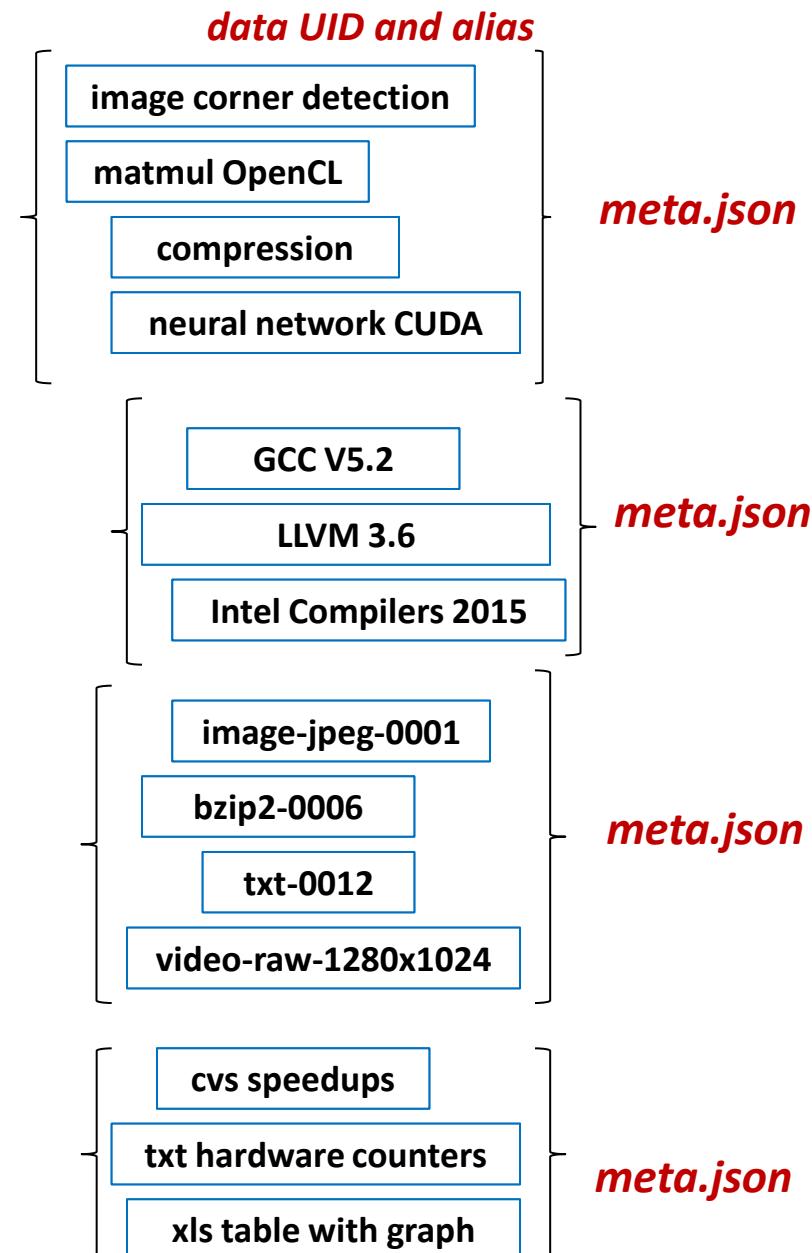
# Collective Knowledge: organize and share artifacts as reusable components

Python module  
“program”  
with functions:  
**compile and run**

Python module  
“soft”  
with function:  
**setup**

Python module  
“dataset”  
with function:  
**extract\_features**

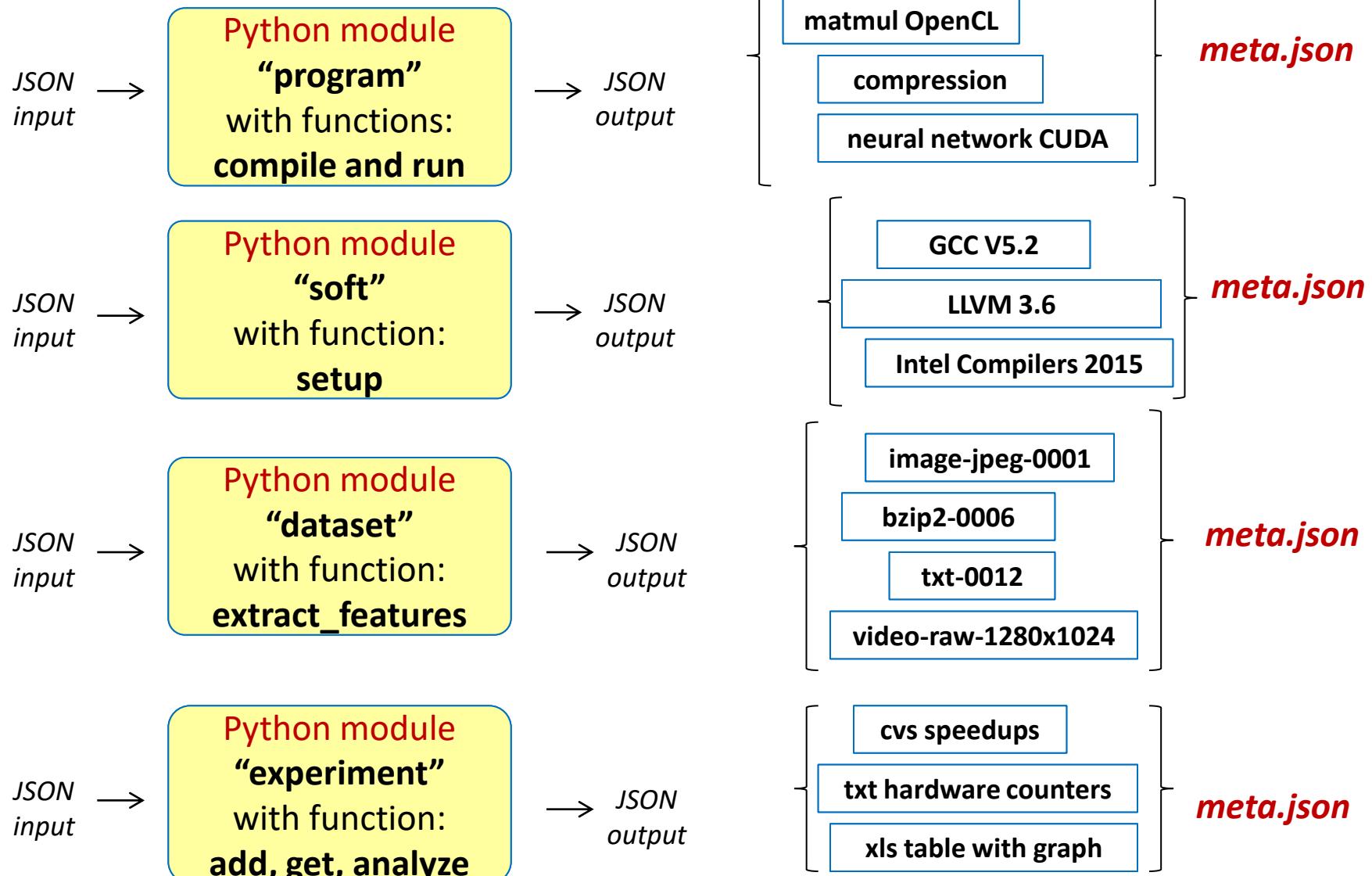
Python module  
“experiment”  
with function:  
**add, get, analyze**



# Collective Knowledge: organize and share your artifacts as reusable components

CK: small python module (~200Kb); no extra dependencies; Linux; Win; MacOS    **data UID and alias**

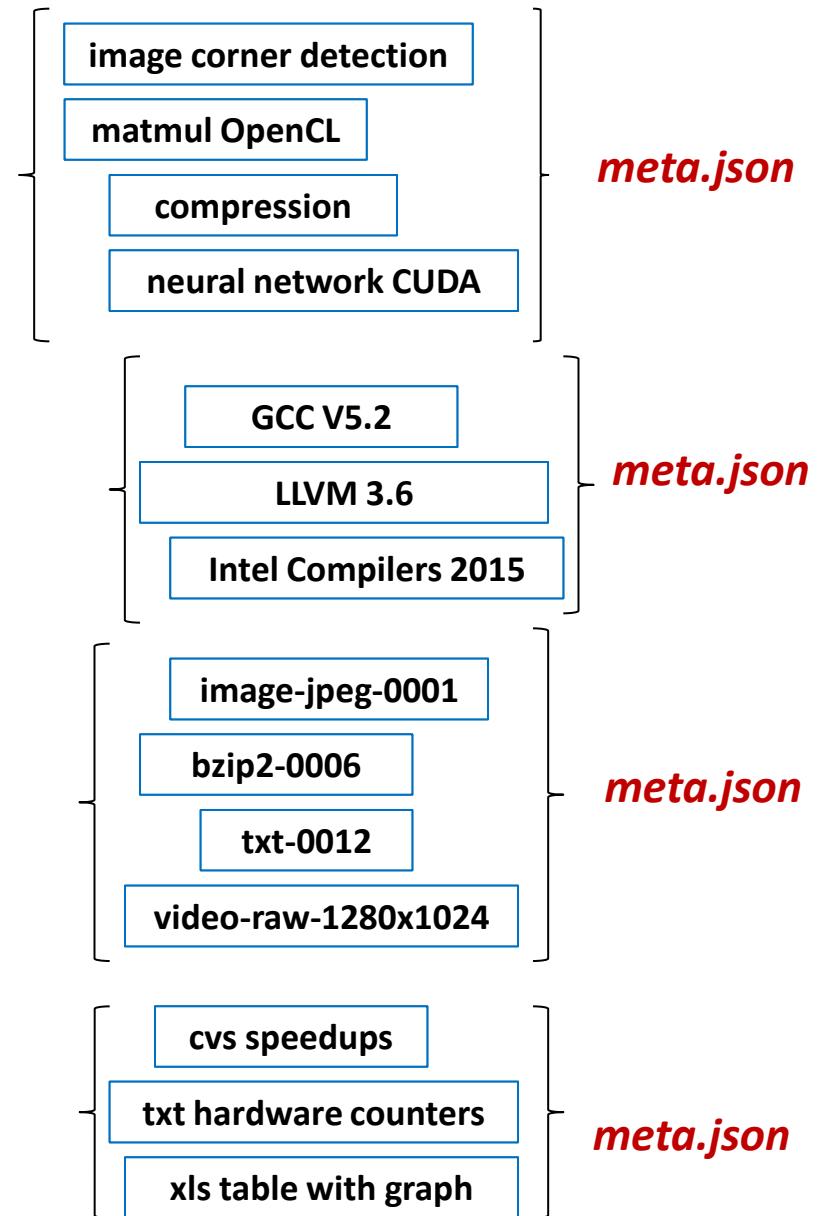
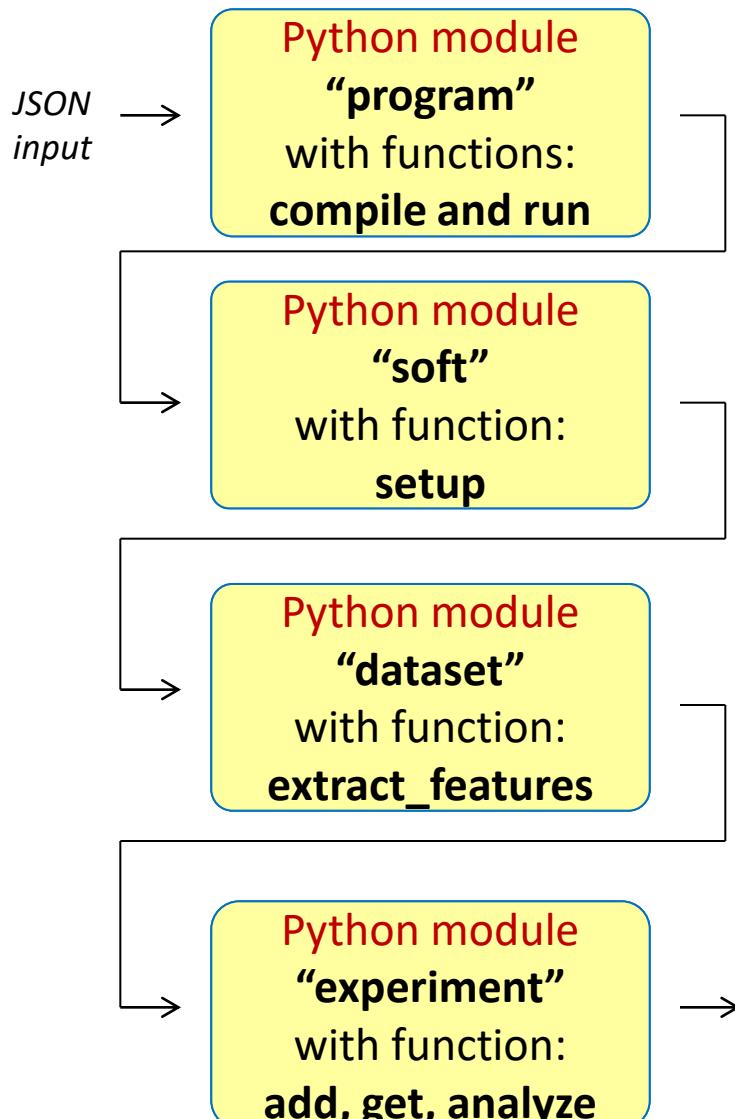
**ck <function> <module>:<data UID> @input.json**



# Collective Knowledge: organize and share your artifacts as reusable components

CK: small python module (~200Kb); no extra dependencies; Linux; Win; MacOS    **data UID and alias**

**Connect into workflows as LEGO™**



# Local repository structure (with internal UID and meta)!

CK  
local  
project  
repo

	/ module UID and alias	/ data UID or alias	/ .cm / meta.json	
program		image corner detection	...	Local files – no locks up on third-party web services
		matmul OpenCL	...	
		compression	...	
		neural network CUDA	...	
soft		gcc 5.2	...	No complex GUI
		llvm 3.6	...	
		icc 2015	...	
dataset		image-jpeg-0001	...	Can be shared via GIT/SVN/etc
		bzip2-0006	...	
		video-raw-1280x1024	...	
module	program		...	Can be easily indexed via ElasticSearch to speed up search
	soft		...	
	dataset		...	
	experiment		...	

Both code (with API) and data (with meta) inside repository

Can be referenced and cross-linked via CID (similar to DOI):

**module UOA : data UOA**

Local files – no locks up on third-party web services

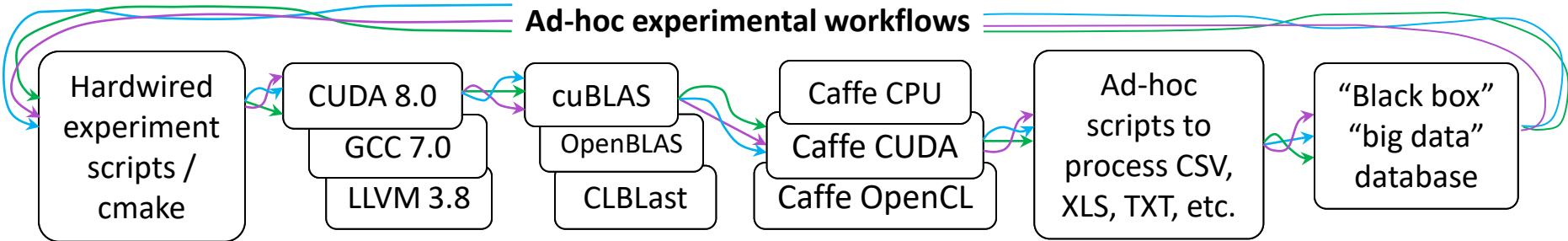
No complex GUI

Can be shared via GIT/SVN/etc

Can be easily indexed via ElasticSearch to speed up search

Can be easily connected to Python-based predictive analytics (sklearn-kit)

# Provide customizable and SW/HW independent access to tools



*Implement SW/HW independent, reusable and shareable CK modules with unified CMD and JSON API for various experimental scenarios (such as **program module**)*

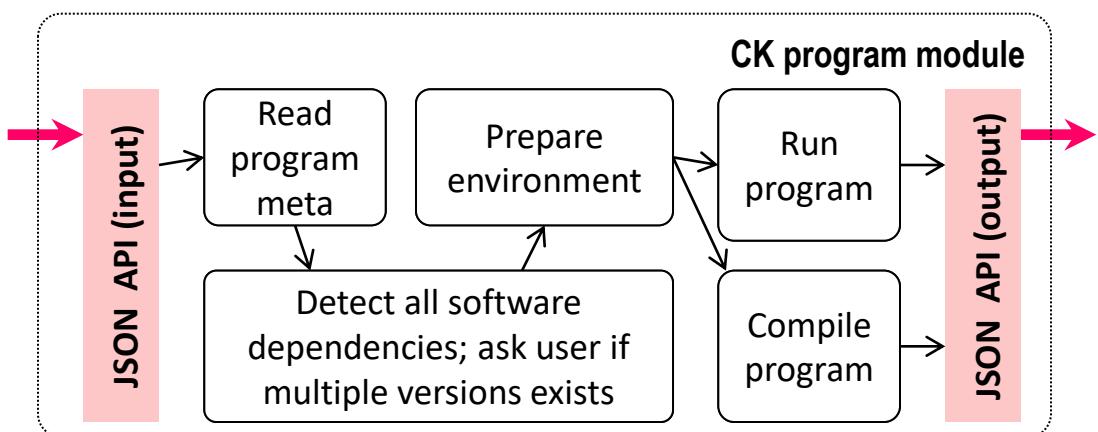
```
$ ck pull repo:ck-autotuning  
$ ck compile program:cbench-automotive-susan --speed  
$ ck run program:cbench-automotive-susan  
$ ck run program:caffe
```

*Use **program entries** to describe a given program in **meta.json** and store sources (if needed)*

## CK program entry (*ck list program*)

.cm/meta.json – describes soft dependencies , data sets, and how to compile and run this program

Source files and auxiliary scripts



# Portable and customizable package manager in the CK

Works on Linux, Windows, MacOS, Android

Detects multiple versions of various software

Builds missing software

Extended by the community

**Soft entries** in CK describe how to detect if a given software is already installed, how to set up all its environment including all paths (to binaries, libraries, include, aux tools, etc), and how to detect its version.

```
$ ck list soft:compiler*
```

```
$ ck detect soft:compiler.gcc
```

```
$ ck detect soft --tags=compiler,cuda
```

```
$ ck detect soft:compiler.llvm --target_os=android19-arm
```

```
$ ck search soft --tags=blas
```

```
$ ck detect soft:lib.cublas
```

**Env entries** are created in CK local repo for all found software instances together with their meta and an auto-generated environment script **env.sh** (on Linux) or **env.bat** (on Windows).

```
$ ck show env
```

local / env / 03ca0be16962f471 / env.sh  
Tags: compiler,cuda,v8.0

```
$ ck show env --tags=cublas
```

local / env / 0a5ba198d48e3af3 / env.bat  
Tags: lib,blas,cublas,v8.0

Local CK repo

**Package entries** describe how to install a given software if it is not installed (using **install.sh** script on Linux host or **install.bat** on Windows host).

```
$ ck list package:*caffemodel*
```

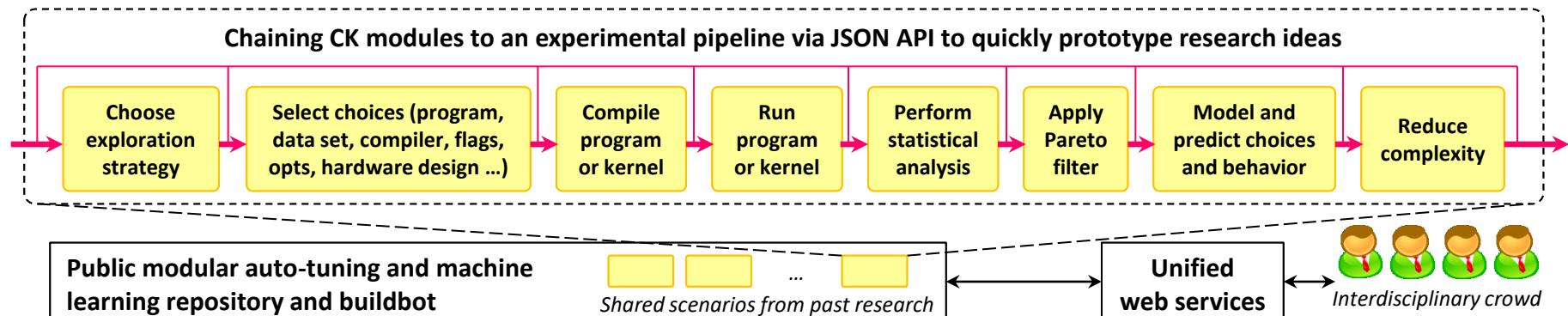
```
$ ck install package:caffemodel-bvlc-googlenet
```

```
$ ck search package --tags=caffe
```

```
$ ck install package:imagenet-2012-val
```

```
$ ck install package:lib-caffe-bvlc-master-cuda
```

# Customizable and reusable workflows with JSON API as LEGO™



Expose and unify information needed for performance analysis and optimization combined with data mining!

Optimizing/modeling behavior **b**

of any object in the CK (program, library function, kernel, ...) as a function of design and optimization choices **c**, features **f** and run-time state **s**

$$\vec{b} = B(\vec{c}, \vec{f}, \vec{s})$$

Flattened JSON vectors

The equation shows a vector  $\vec{b}$  being produced by a function  $B$  that takes three vectors as input:  $\vec{c}$ ,  $\vec{f}$ , and  $\vec{s}$ . Each vector is represented as a column of colored squares (purple, green, blue, red) with ellipses indicating more elements.

# Gradually add JSON specification (depends on research scenario)

Autotuning and machine learning specification:

```
{  
  "characteristics": {  
    "execution times": ["10.3", "10.1", "13.3"],  
    "code size": "131938", ...},  
  "choices": {  
    "os": "linux", "os version": "2.6.32-5-amd64",  
    "compiler": "gcc", "compiler version": "4.6.3",  
    "compiler_flags": "-O3 -fno-if-conversion",  
    "platform": {"processor": "intel xeon e5520",  
      "l2": "8192", ...}, ...},  
  "features": {  
    "semantic features": {"number_of_bb": "24", ...},  
    "hardware counters": {"cpi": "1.4", ...}, ...}  
  "state": {  
    "frequency": "2.27", ...}  
}
```

CK flattened JSON key

##characteristics#execution\_times@1

"flattened\_json\_key":{

"type": "text"|"integer" | "float" | "dict" | "list"

| "uid",

"characteristic": "yes" | "no",

"feature": "yes" | "no",

"state": "yes" | "no",

"has\_choice": "yes" | "no",

"choices": [ list of strings if categorical

choice],

range",

range",

"explore\_start": "start number if numerical

"explore\_stop": "stop number if numerical

"explore\_step": "step if numerical range",

"can\_be\_omitted": "yes" | "no"

...

# The community now use CK to collaboratively tackle old problems

Crowdsourcing performance analysis and optimization, machine-learning, run-time adaptation across diverse workloads and hardware provided by volunteers

```
$ sudo pip install ck
```

```
$ ck pull repo:ck-crowdtuning
```

```
$ ck crowdtune program --gcc
```

- "Collective Mind: Towards practical and collaborative autotuning",  
Journal of Scientific Programming 22 (4), 2014  
<http://hal.inria.fr/hal-01054763>
- "Collective Mind, Part II: Towards Performance- and Cost-Aware Software Engineering as a Natural Science", CPC 2015, London, UK  
<http://arxiv.org/abs/1506.06256>
- Android application to crowdsourcing autotuning across mobile devices:  
<https://play.google.com/store/apps/details?id=openscience.crowdsource.experiments>
- "Collective Knowledge: towards R&D sustainability", DATE 2016, Dresden, Germany  
<http://bit.ly/ck-date16>
- "Optimizing convolutional neural networks on embedded platforms with OpenCL",  
IWOCN 2016, Amsterdam  
<http://dl.acm.org/citation.cfm?id=2909449>

# Reproducibility of experimental results as a side effect

## Reproducibility came as a side effect!

- Can preserve the whole experimental setup with all data and software dependencies
  - Can perform statistical analysis for characteristics
  - Community can add missing features or improve machine learning models

**Execution time:**

**10 sec.**

# Reproducibility of experimental results as a side effect

## Reproducibility came as a side effect!

- Can preserve the whole experimental setup with all data and software dependencies
  - Can perform statistical analysis for characteristics
- Community can add missing features or improve machine learning models

## Variation of experimental results:

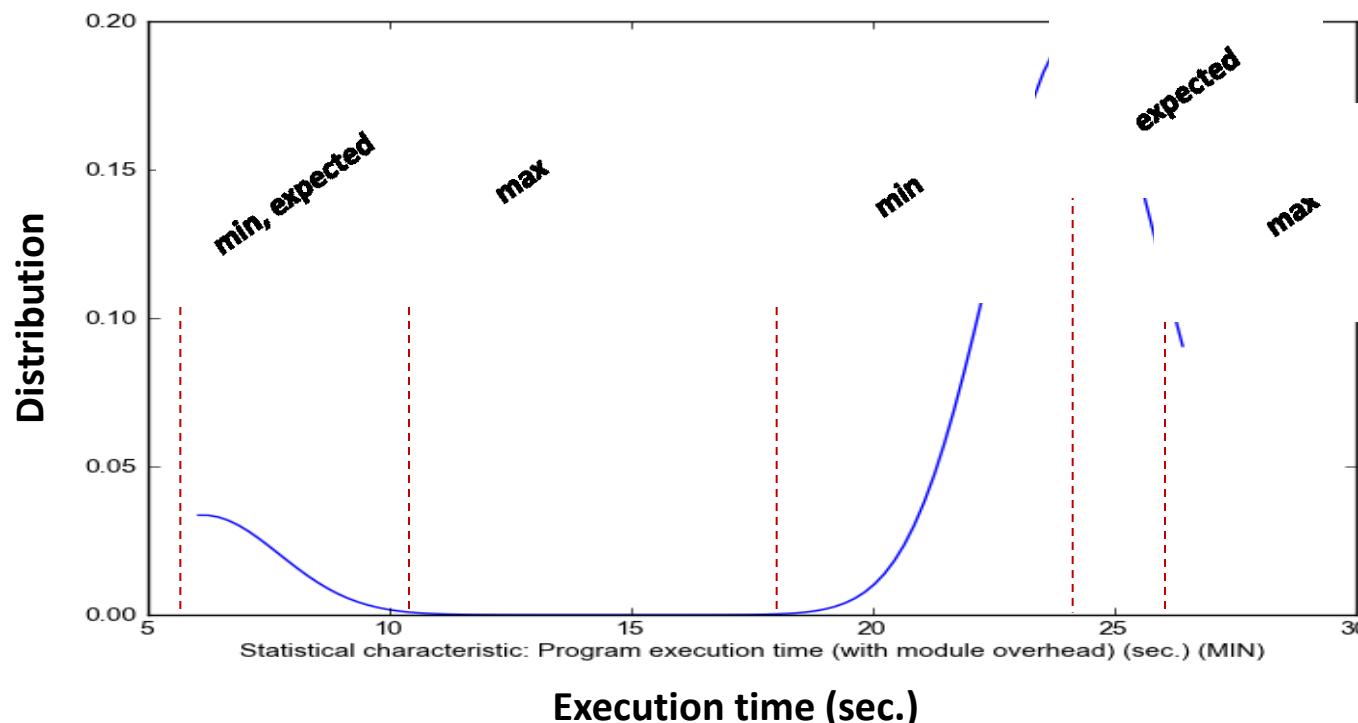
$10 \pm 5$  secs.

# Reproducibility of experimental results as a side effect

## Reproducibility came as a side effect!

- Can preserve the whole experimental setup with all data and software dependencies
  - Can perform statistical analysis for characteristics
- Community can add missing features or improve machine learning models

**Unexpected behavior - expose to the community including experts  
to explain, find missing feature and add to the system**

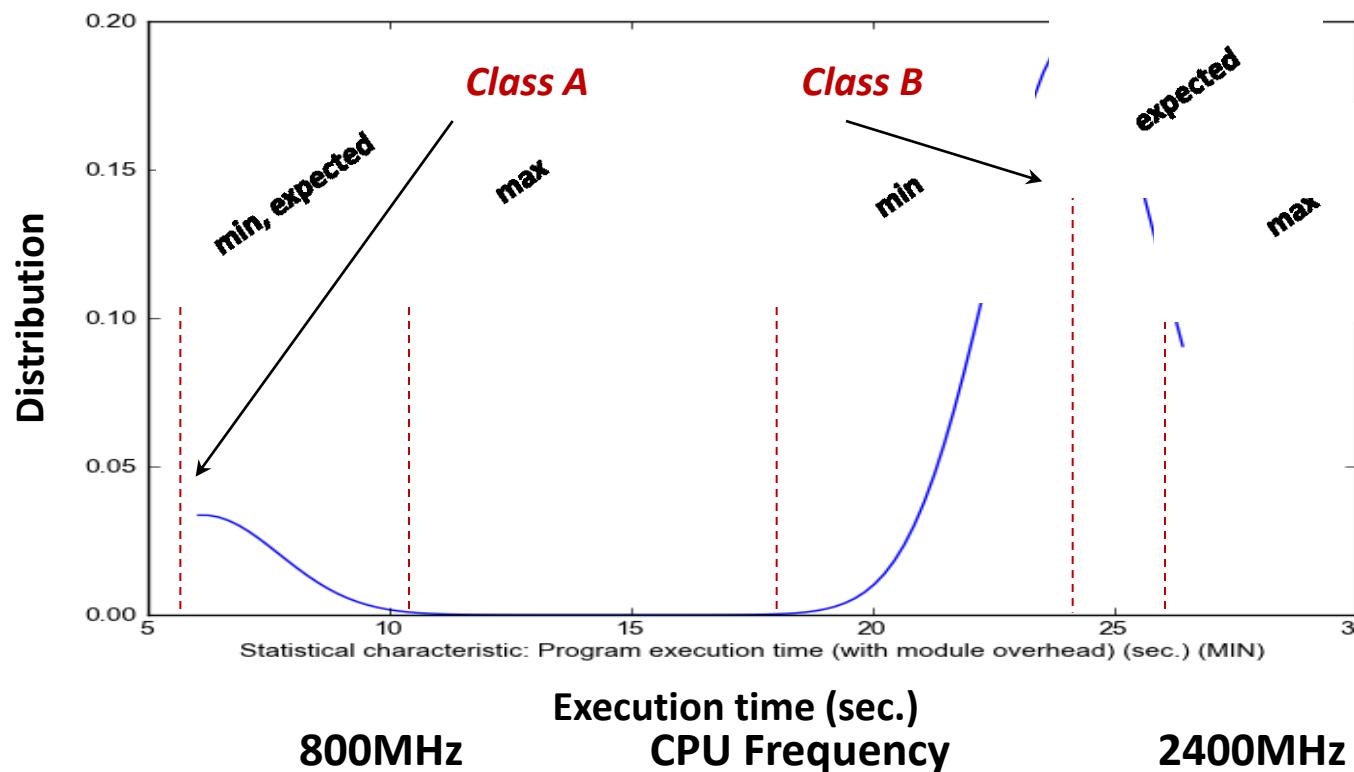


# Reproducibility of experimental results as a side effect

## Reproducibility came as a side effect!

- Can preserve the whole experimental setup with all data and software dependencies
  - Can perform statistical analysis for characteristics
- Community can add missing features or improve machine learning models

Unexpected behavior - expose to the community including experts  
to explain, find missing feature and add to the system



# Gradually extend/fix shared workflows by the community during AE

## • Init pipeline

- Detected system information
- Initialize parameters
- Prepare dataset

## • Clean program

## • Prepare compiler flags

- Use compiler profiling
- Use cTuning CC/MILEPOST GCC for fine-grain program analysis and tuning
- Use universal Alchemist plugin (with any OpenME-compatible compiler or tool)
- Use Alchemist plugin (currently for GCC)

## • Compile program

- Get objdump and md5sum (if supported)
- Use OpenME for fine-grain program analysis and online tuning (build & run)
- Use 'Intel VTune Amplifier' to collect hardware counters
- Use 'perf' to collect hardware counters
- Set frequency (in Unix, if supported)
- Get system state before execution

## • Run program

- Check output for correctness (use dataset UID to save different outputs)
- Finish OpenME

## • Misc info

## • Observed characteristics

- Observed statistical characteristics

## • Finalize pipeline

We've shared and continuously extend our experimental workflow for autotuning:

<http://github.com/ctuning/ck-autotuning>

<http://cknowledge.org/repo>

<http://github.com/ctuning>

- Hundreds of benchmarks/kernels/codelets (CPU, OpenMP, OpenCL, CUDA)
- Thousands of data sets
- Wrappers around all main tools and libs
- Optimization description of major compilers

# Distinguished CGO'17 artifact in Collective Knowledge format

Highest ranked artifact at CGO'17 turned out to be implemented using Collective Knowledge Framework

*"Software Prefetching for Indirect Memory Accesses",  
Sam Ainsworth and Timothy M. Jones*

<https://github.com/SamAinsworth/reproduce-cgo2017-paper>

It takes advantage of a portable package manager to install required LLVM for either x86 or ARM platforms, automatically build LLVM plugins, run empirical experiments on a user machine via CK workflow, compare speedups with pre-recorded results by authors, and prepare interactive report

See PDF with Artifact Evaluation Appendix:

<http://ctuning.org/ae/resources/paper-with-distinguished-ck-artifact-and-ae-appendix-cgo2017.pdf>

See snapshot of an interactive CK dashboard:

<https://github.com/SamAinsworth/reproduce-cgo2017-paper/files/618737/ck-aarch64-dashboard.pdf>

# Collective Knowledge mini-tutorials

Create repository:	<code>ck add repo:my_new_project</code>
Add new module:	<code>ck add my_new_project:module:my_module</code>
Add new data for this module:	<code>ck add my_new_project:my_module:my_data @@dict {"tags":"cool","data"}</code>
Add dummy function to module:	<code>ck add_action my_module --func=my_func</code>
Test dummy function:	<code>ck my_func my_module --param1=var1</code>
List my_module data:	<code>ck list my_module</code>
Find data by tags:	<code>ck search my_module --tags=cool</code>
Archive repository:	<code>ck zip repo:my_new_project</code>
Pull existing repo from GitHub:	<code>ck pull repo:ck-autotuning</code>
List modules from this repo:	<code>ck list ck-autotuning:module:*</code>
Compile program (using GCC):	<code>ck compile program: cbench-automotive-susan --speed</code>
Run program:	<code>ck run program: cbench-automotive-susan</code>
Start server for crowdsourcing:	<code>ck start web</code>
View interactive articles:	<code>ck browser http://localhost:3344</code>

<https://michel-steuwer.github.io/About-CK/>

<https://github.com/ctuning/ck/wiki/Compiler-autotuning>

<http://github.com/ctuning/ck/wiki>

- “Deep” (multi-layered) neural networks that take advantage of structure in digital signals (e.g. images).
- State-of-the-art for applications in automotive, robotics, healthcare, entertainment (e.g. image classification).
- Commonly trained on workstations or clusters with GPGPUs.
- Increasingly deployed on mobile and embedded systems.

**Difficult and time consuming to build and optimize  
due to multiple programming models, multiple objectives  
(performance, energy, accuracy, memory footprint)  
and continuously evolving hardware  
particularly with constrained resources (sensors, IoT)**

**Perfect use case for Collective Knowledge Framework!**

# Deploying DNNs on resource-constrained platforms

<b>NVIDIA Drive PX2</b>	~ 250 Watts	~ \$15,000 ( <i>early development boards</i> ) ~ \$1,000 per unit?
?	< 10 Watts	< \$100

- Can we deploy a DNN to achieve desired performance (rate and accuracy of recognition) on a given platform?
- Can we identify or build such a platform under given constraints such as those on power, memory, price?
- If all else fails, can we design (autotune) another DNN (topology) by trading off performance, accuracy and costs?

# Customizable CK workflow to collaborative co-design DNN

We collaborate with General Motors and ARM to develop open-source CK-based tools to collaboratively evaluate and optimize various DNN engines and models across diverse platforms and datasets

**cKnowledge.org/ai**  
**cKnowledge.org/repo**

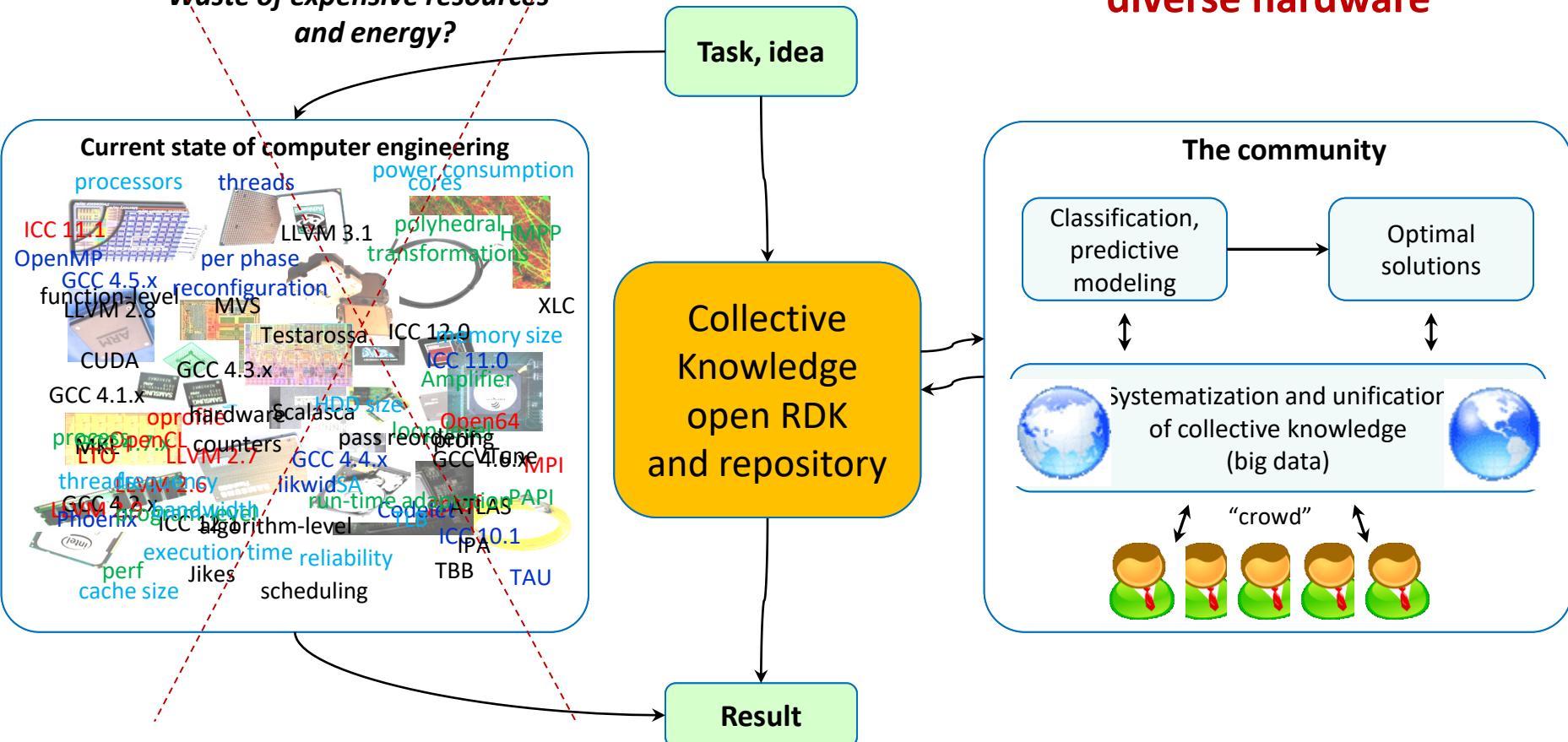
**Download Android app to participate in collaborative evaluation and optimization of DNN using your mobile device and CK web-service:**  
<https://play.google.com/store/apps/details?id=openscience.crowdsource.video.experiments>

*CK portable workflow systems and package manager allows users to run DNN experiments on Windows, Linux, MacOS, Android!*

```
$ ck pull repo --url=http://github.com/dividiti/ck-caffe  
$ ck install package:lib-caffe-bvlc-master-cpu-universal  
$ ck install package:lib-caffe-bvlc-opencl-viennacl-universal --target_os=android21-arm64  
$ ck run program:caffe-classification  
$ firefox http://cKnowledge.org/repo
```

- Collective Knowledge approach helps fight SW/HW chaos
- CK is also changing the mentality of computer systems' researchers:
  - sharing artifacts as customizable and reusable components with JSON API
  - sharing customizable and portable experimental workflows
  - building a repository of representative benchmarks and data sets
  - crowdsourcing experiments and sharing negative/unexpected results
  - collaboratively improving reproducibility of empirical experiments
  - using data mining (statistical analysis and machine learning) to predict efficient software optimizations and hardware designs
  - collaboratively improving prediction models and finding missing features
  - formulating and solving important real-world problems
- CK brings closer together industry and academia (common research methodology, reproducible experiments, validated techniques)

*Quick, non-reproducible hack?  
Ad-hoc heuristic?  
Quick publication?  
Waste of expensive resources  
and energy?*



**Extrapolate shared knowledge to build fast, energy efficient, reliable and cheap computer systems to boost innovation in science and technology!**

**We need your feedback - remember that new AE procedures may affect you at the future conferences**

- AE steering committee: <http://cTuning.org/ae/committee.html>
- Mailing list: <https://groups.google.com/forum/#!forum/collective-knowledge>

**Feel free to reuse and improve our Artifact Evaluation procedures and Artifact Appendices:**

<http://cTuning.org/ae>

<https://github.com/ctuning/ck-web-artifact-evaluation>

## Extra resources

- ACM Result and Artifact Review and Badging policy:  
<http://www.acm.org/publications/policies/artifact-review-badging>
- Evaluation of frameworks for reproducible R&D:  
<https://wilkie.github.io/reproducibility-site>
- Community driven artifact evaluation and paper reviewing:  
<http://dl.acm.org/citation.cfm?doid=2618137.2618142>

# Questions, comments, collaborations?

## Acknowledgments

ARM



UNIVERSITY OF  
CAMBRIDGE



Association for  
Computing Machinery



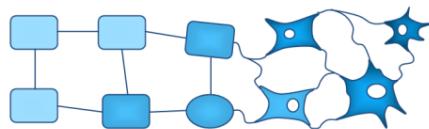
Science & Technology  
Facilities Council

Microsoft®  
**Research**



**Many new exciting opportunities for collaboration**

*Artifact sharing, customizable and portable workflows, experiment crowdsourcing, collaborative deep learning optimization, public repositories of knowledge, adaptive systems*



*cTuning*  
**Foundation**



*Powered by*  
**Collective Knowledge Framework**

$$\frac{d\vec{v}}{dt}$$