Collective Mind
a collaborative curation tool for program optimization

Towards collaborative, systematic and reproducible design and optimization of computer systems

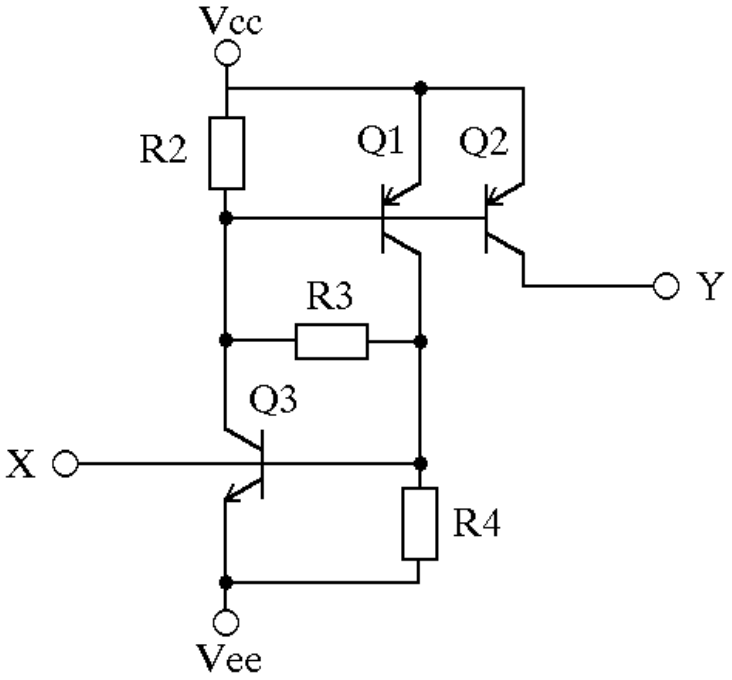
Grigori Fursin
INRIA, France

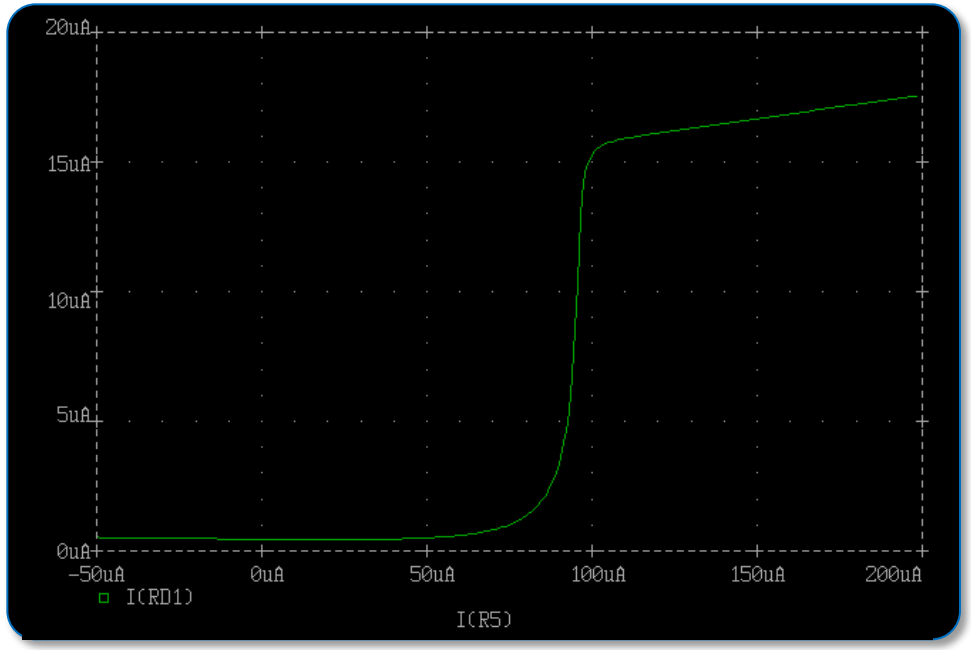
SEA 2014, Boulder, USA
April 2014

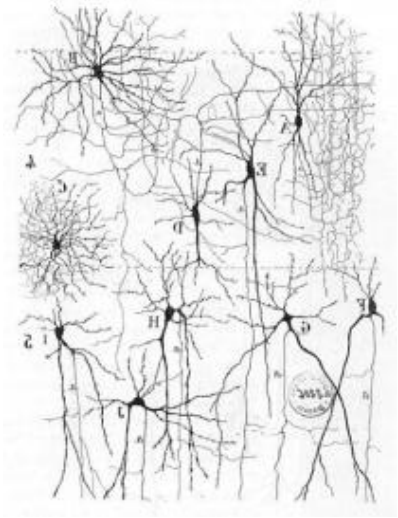
Outline

- Interdisciplinary background
- Back to basics: major problems in computer engineering
- Collaborative and Reproducible Research and Development:
  - *cTuning.org framework and repository*
  - *Exposed problems during practical usage*
  - *Collective Mind framework and repository*
  - *Collaborative research usage scenarios*
  - *Collaborative curation for program optimization*
  - *Changing R&D methodology and publication model*
- Conclusions and future work

Back to 1993: physics, electronics, machine learning







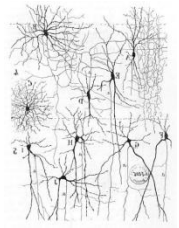
Semiconductor neuron -
base of neural accelerators
and possible future neuro computers

Modeling and understanding brain functions

My problem with modeling:

- **Slow**
- **Unreliable**
- **Costly**

Back to basics: task pipeline (workflow)



End User task

I needed help from computer scientists to find the best solutions for my tasks!

Available solutions

- Algorithm
- Application
- Compilers
- Binary and libraries
- State of the system
- Data set
- Run-time environment
- Storage
- Architecture

Result

Service/application providers
(cloud computing, supercomputers, mobile systems)

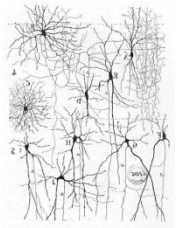
User requirements:

minimize all costs (characteristics)
(execution time, power consumption, price, size, faults, etc)

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

Hardware and software designers
(processors, memory, interconnects, languages, compilers, run-time systems)

Back to basics: ever rising complexity of computer systems



End User task

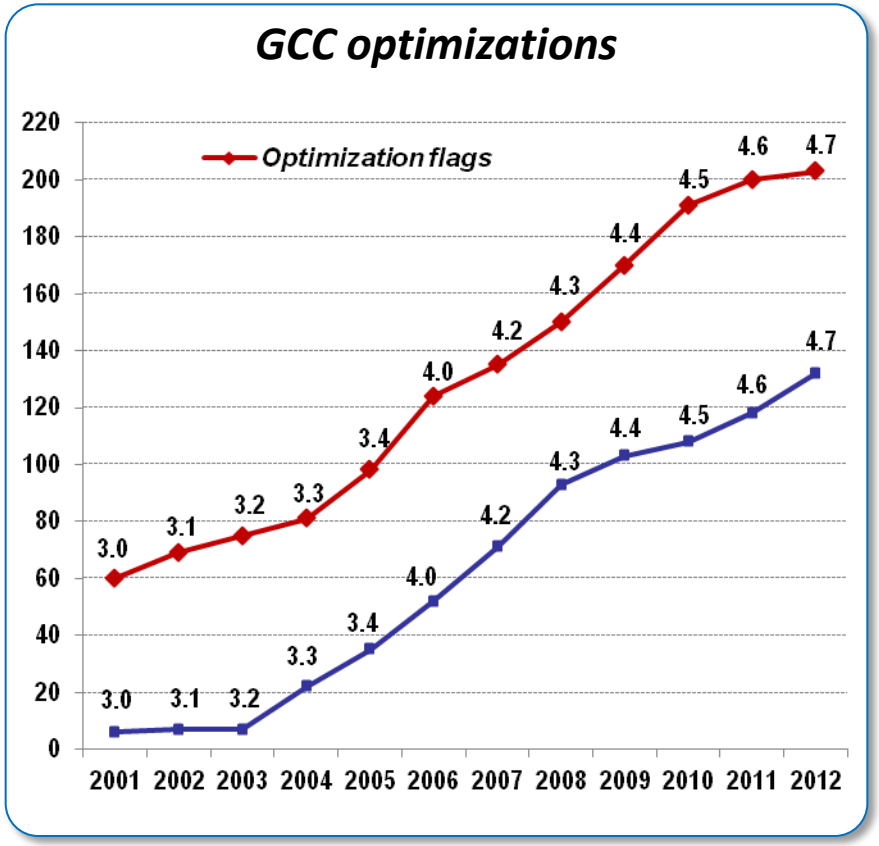
Deliver universal and optimal optimization heuristic is often impossible!

1) Too many design and optimization choices at all levels

2) Always multi-objective optimization: performance vs compilation time vs code size vs system size vs power consumption vs reliability vs return on investment

3) Complex relationship and interactions between ALL software and hardware components

4) Users and developers often have to resort to empirical auto-tuning



Result

Empirical auto-tuning is too time consuming, ad-hoc and tedious to be a mainstream!

cTuning.org: combine auto-tuning with machine learning and crowdsourcing

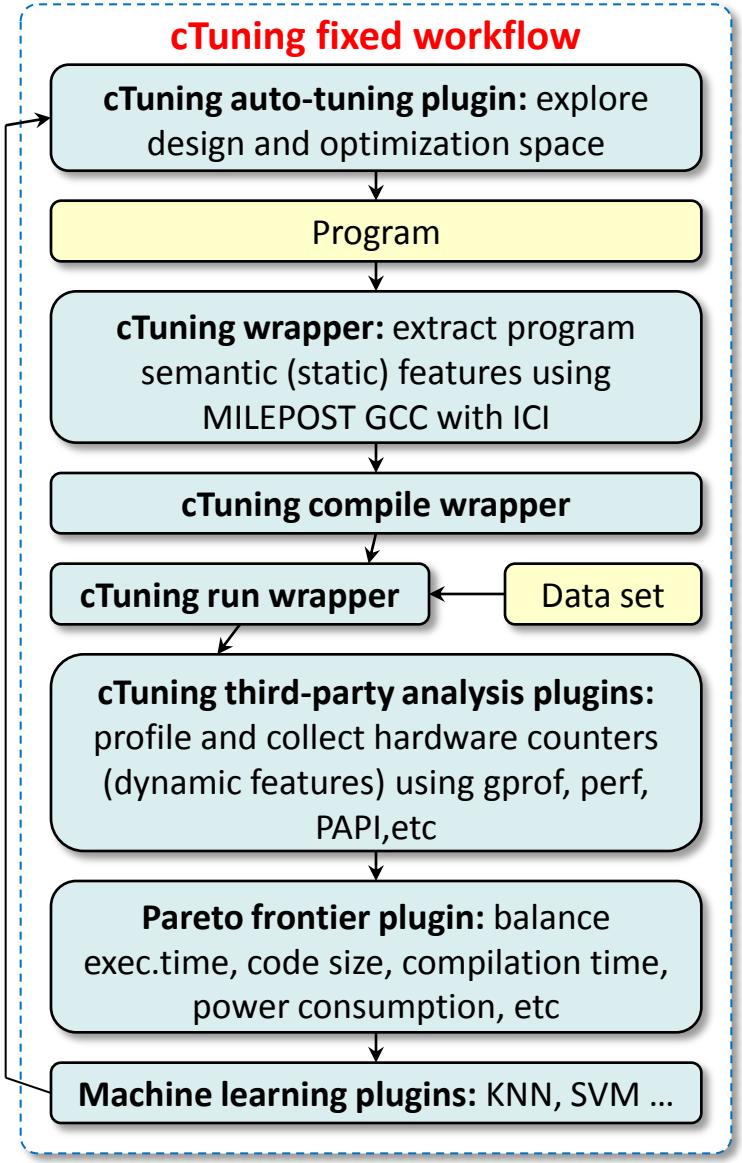
Many end users and service providers run numerous applications
on different architectures with different data sets, run-time systems, compilers and optimizations!




- **Can we leverage their experience and computational resources?**
- **Can we build common research and experimentation infrastructure connected to a public repository of knowledge?**
- **Can we extrapolate collected knowledge using machine learning to predict optimal program optimizations, hardware designs and run-time adaptation scenarios?**

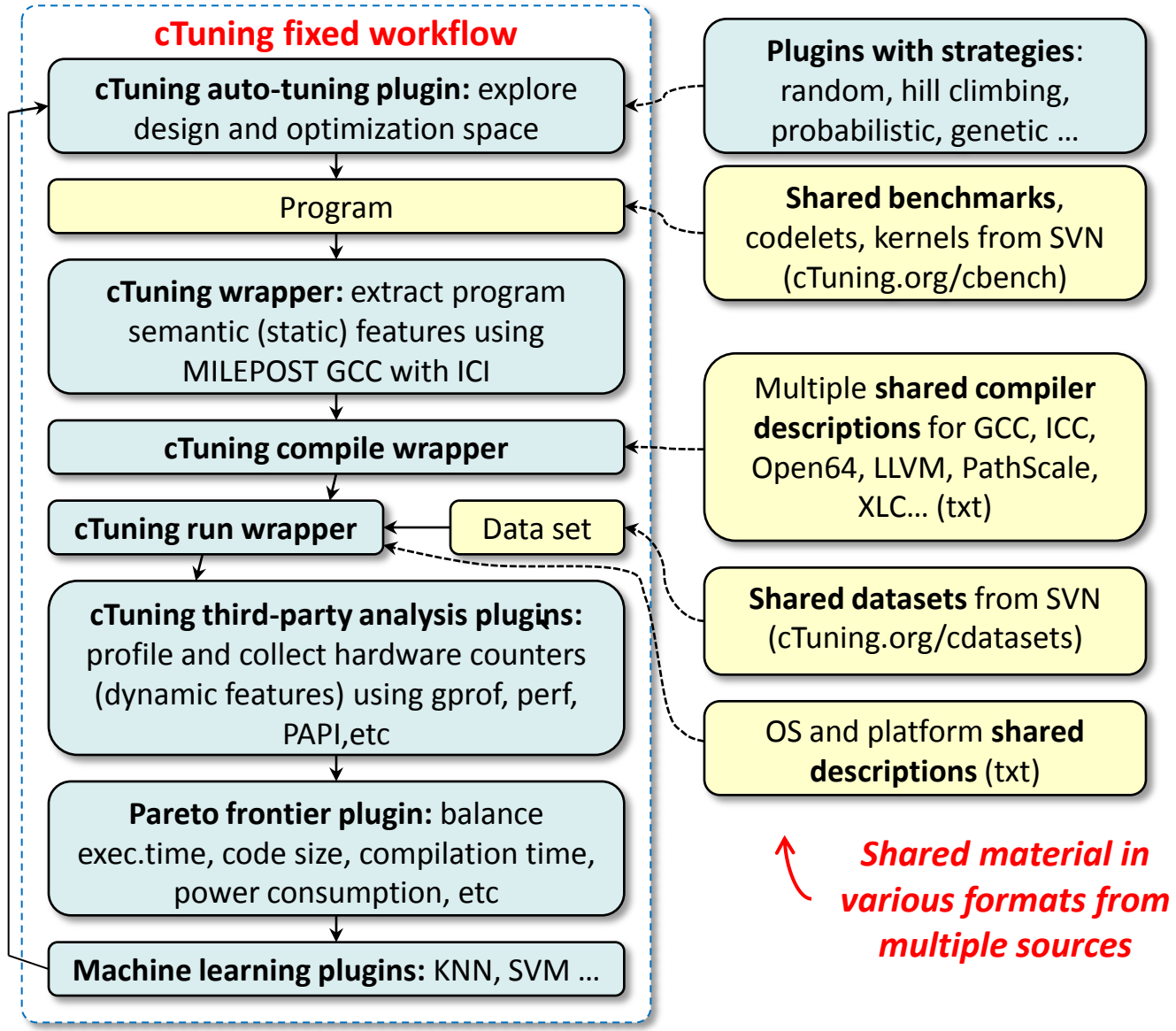
Proposed and developed in the MILEPOST/cTuning project (2006-2009) to automate and crowdsource training of a machine learning based self-tuning compiler for reconfigurable processors (SW/HW co-design).

cTuning.org: plugin-based auto-tuning and learning framework



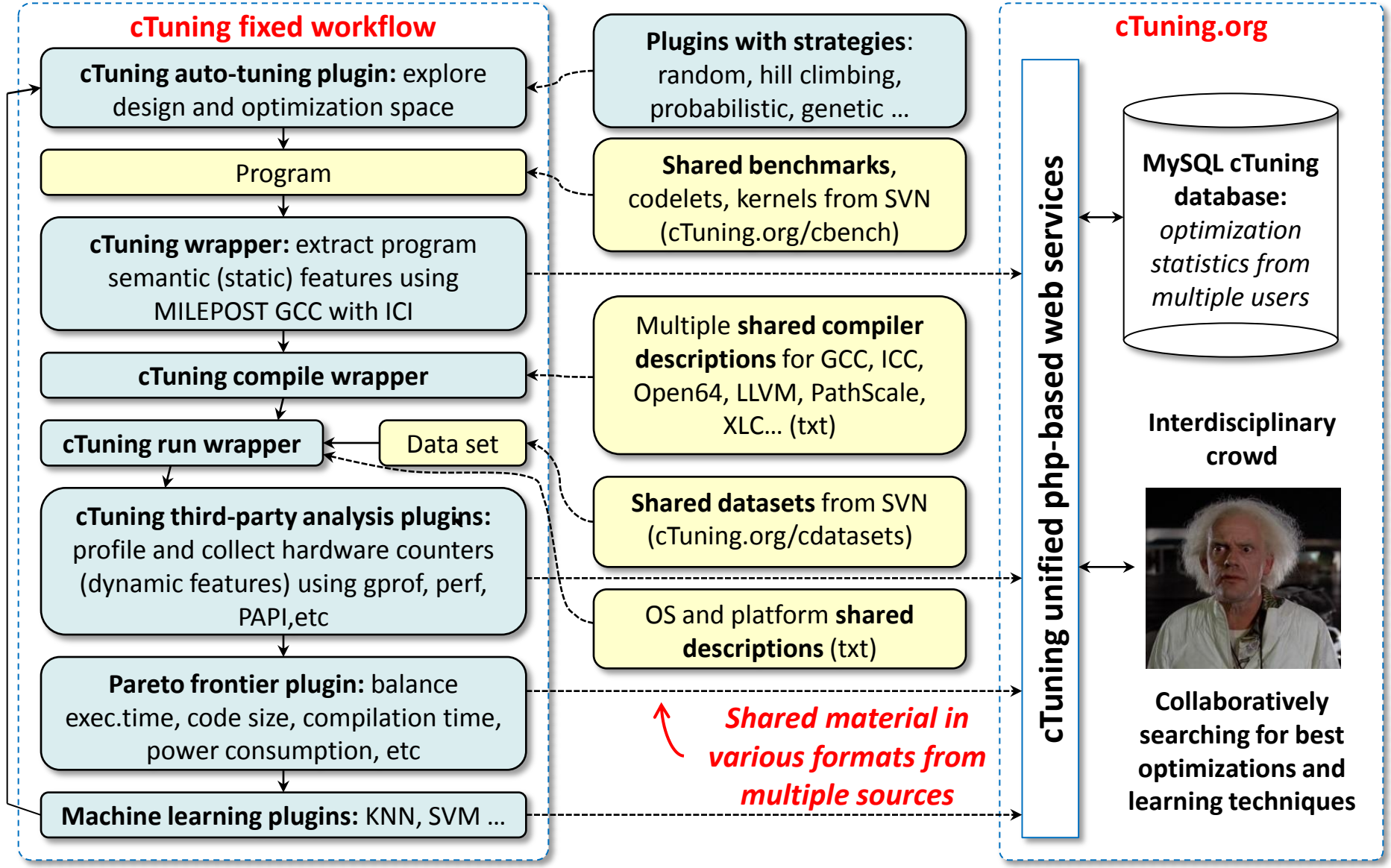
Common open-source cTuning CC framework (http://cTuning.org/ctuning-cc)

cTuning.org: plugin-based auto-tuning and learning framework



Common open-source cTuning CC framework (http://cTuning.org/ctuning-cc)

cTuning.org: plugin-based auto-tuning and learning framework



Common open-source cTuning CC framework (http://cTuning.org/ctuning-cc)

cTuning.org: plugin-based auto-tuning and learning framework

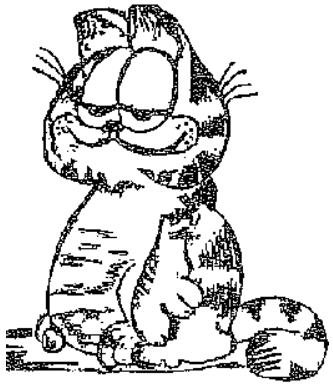
cTuning fixed workflow

cTuning auto-tuning plugin: explore design and optimization space

Plugins with strategies: random, hill climbing, probabilistic, genetic …

cTuning.org

In 2009, we managed to automatically tune customer benchmarks and compiler heuristics for a range of real platforms from IBM and ARC (Synopsis)



Everything is solved?

Machine learning plugins: KNN, SVM …

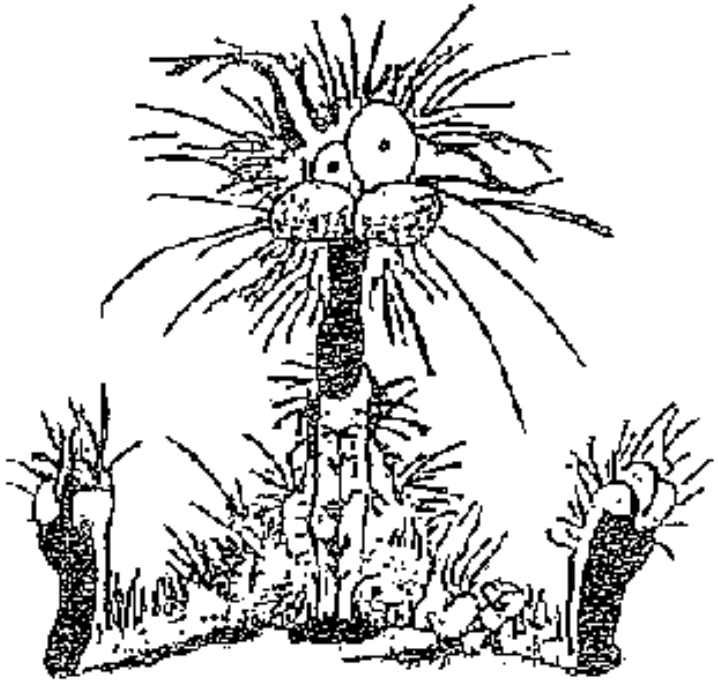
multiple sources

Common open-source cTuning CC framework (http://cTuning.org/ctuning-cc)

cTuning practical usage exposed a few problems

Technological chaos

LLVM 3.4
AVX
GCC 4.8.x
GCC 4.1.x
ATLAS
LLVM 2.8
hardware
perf
frequency
genetic
CUDA 5.x
function
counters
algorithms
ICC 11.1
level
pass
reordering
ARM v8
MPI
OpenCL
OpenMP
gprof
prof
CUDA 4.x
Intel SandyBridge
reliability
GCC 4.2.x
ARM v6
per phase
LLVM 3.0
reconfiguration
MVS 2013
HMPP
memory size
GCC 4.5.x
polyhedral
XLC
execution time
transformations
KNN
GCC 4.3.x
ICC 12.0
predictive
Scalasca
SSE4
scheduling
ICC 11.0
MKL
bandwidth
GCC 4.4.x
GCC 4.6.x
LLVM 2.6
LLVM 2.9
SimpleScalar
Codelet
PAPI
ICC 12.1
Jikes
process
Testarossa
Amplifier
Open64
HDD size
oprofile
TLB
loop-level
ISA
GCC 4.7.x
LLVM 2.7
SVM
VTune
threads
LTO
ICC 10.1
threads
Phoenix
program-
algorithm-
TAU
level
level
IPA
cache size
TBB
algorithm precision

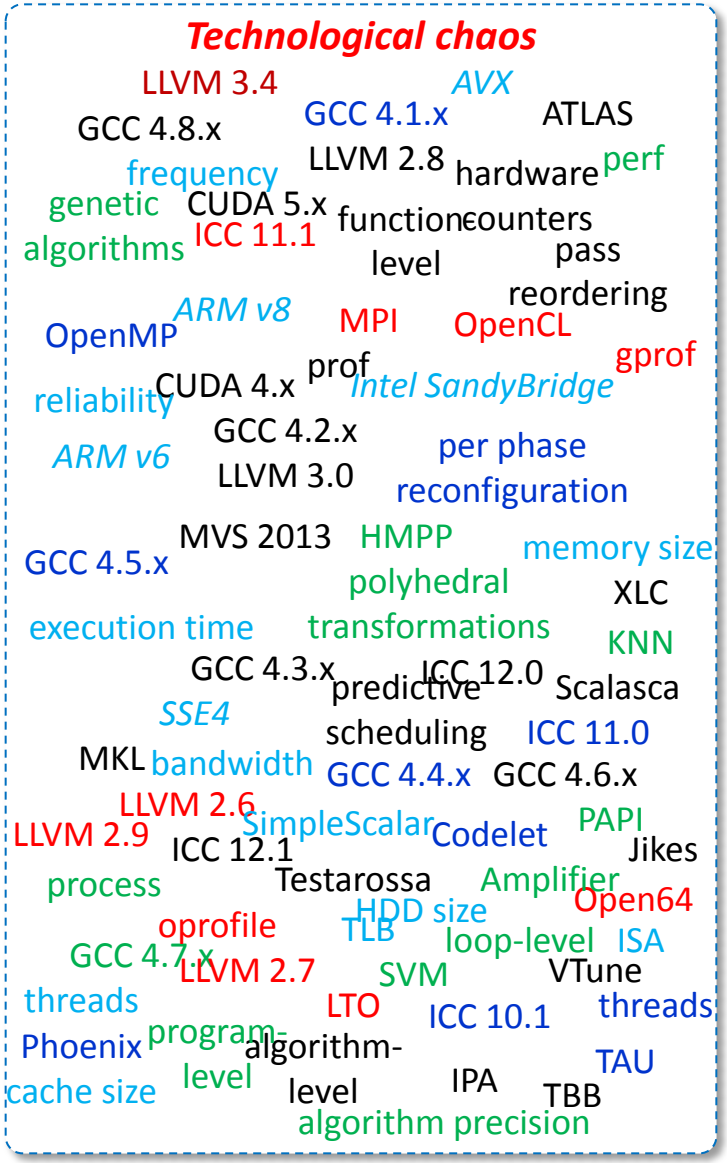


Many thanks to international academic and industrial users for their feedback

http://cTuning.org/lab/people

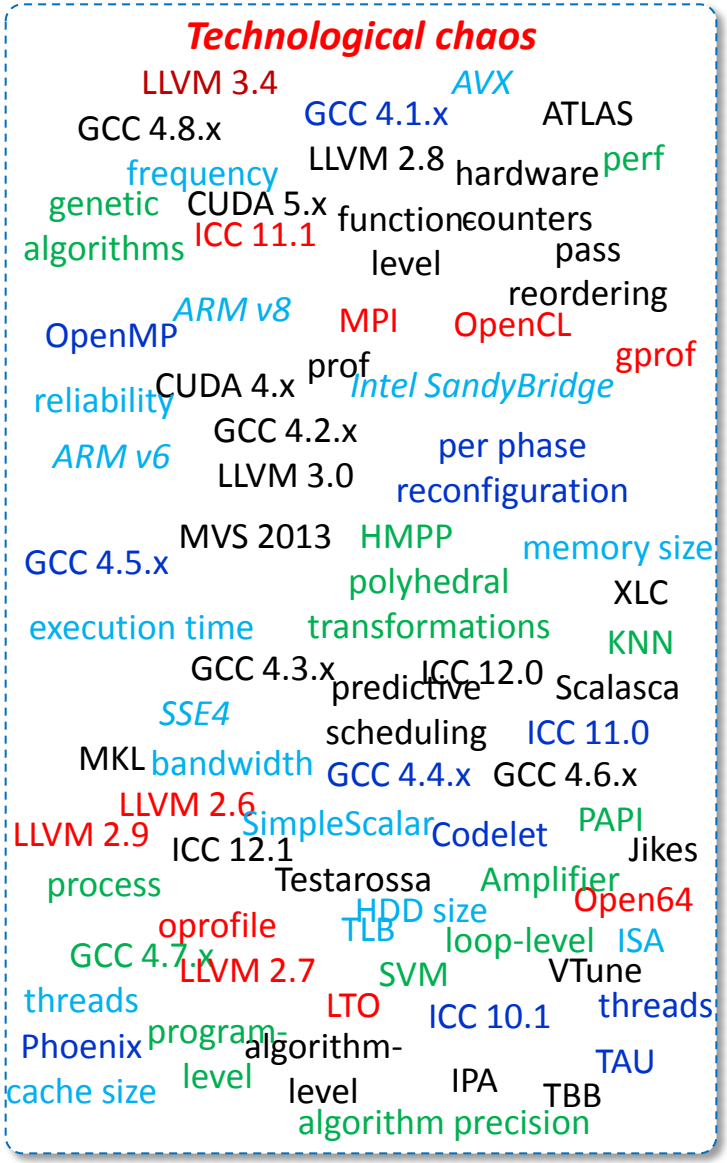
Thanks to Davide Del Vento and his interns from UCAR

cTuning practical usage exposed a few problems



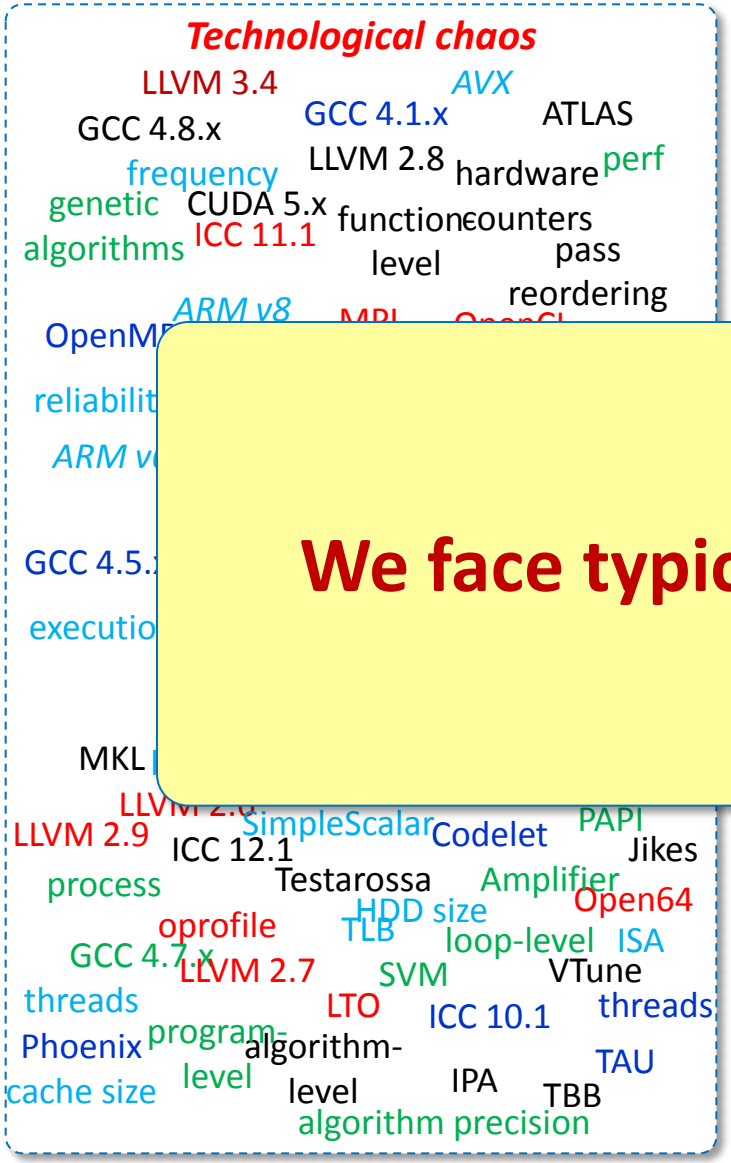
- **Ever changing environment (SW and HW choices, features, tools and libraries versions, interfaces, data formats …)**
  - *Difficult to reproduce experimental results from different users*
  - *Difficult to keep cTuning up-to-date - by the end of development cycle, new software and hardware is often available*
- **Complex deployment and maintenance**
  - *Complex, manual configuration of the framework, repository, plugins, hardware and all dependencies*
  - *Many used languages (C, C++, PHP, Java) and data exchange formats (txt, csv, xls, …)*
  - *Different versions of benchmarks, data sets, plugins, models, compilers, third-party tools, etc are scattered around many directories and disks*

cTuning practical usage exposed a few problems



- **Difficult or impossible to reproduce and reuse techniques from existing publications**
  - *No culture of sharing code and data in computer engineering unlike other sciences*
  - *Impossible to publish negative results or just validation of existing works*
  - *Academia focuses more and more on quantity (number of publications and citations) rather than quality and reproducibility of results*
- **No simple and scalable mechanism to preserve, share and reuse all heterogeneous experimental material**
  - *Centralized, MySQL-based repository got quickly saturated when collecting and processing data from many users*
  - *MySQL repository is difficult to adapt to continuous changes in the system*
  - *Difficult to preserve different versions of benchmarks, data sets, plugins and tools, and make them co-exist using SVN repositories*

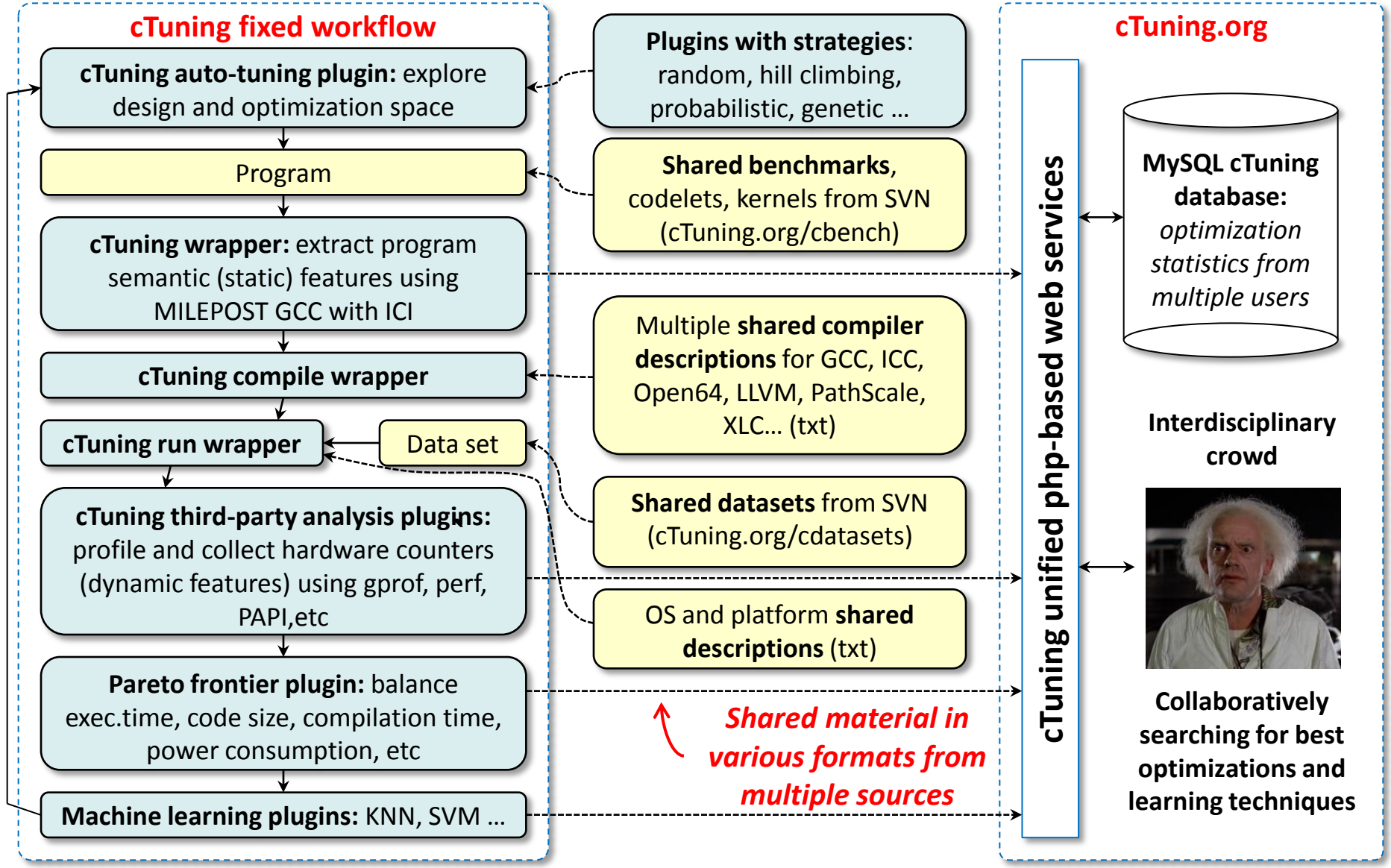
cTuning practical usage exposed a few problems



We face typical "big data" problem!

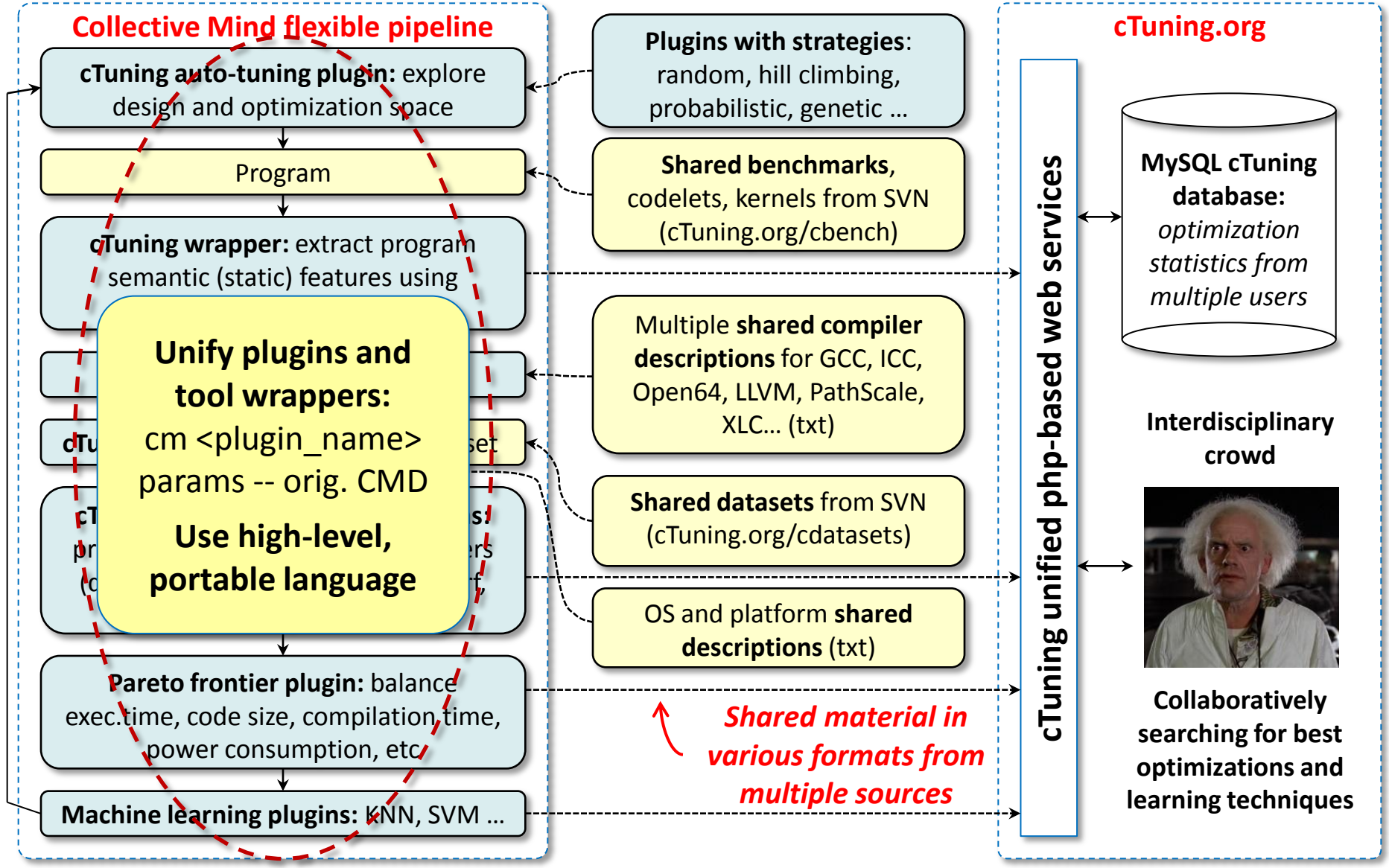
- **Difficult or impossible to reproduce and reuse techniques from existing publications**
  - *No culture of sharing code and data in computer engineering unlike other sciences*
  - *Impossible to publish negative results or just validation of existing works*
  - *quantity*
  - *) rather*
  - *ults*
  - **serve,**
  - **mental**
  - *got quickly saturated when collecting and processing data from many users*
  - *MySQL repository is difficult to adapt to continuous changes in the system*
  - *Difficult to preserve different versions of benchmarks, data sets, plugins and tools, and make them co-exist using SVN repositories*

Collective Mind: attempt to fix cTuning problems



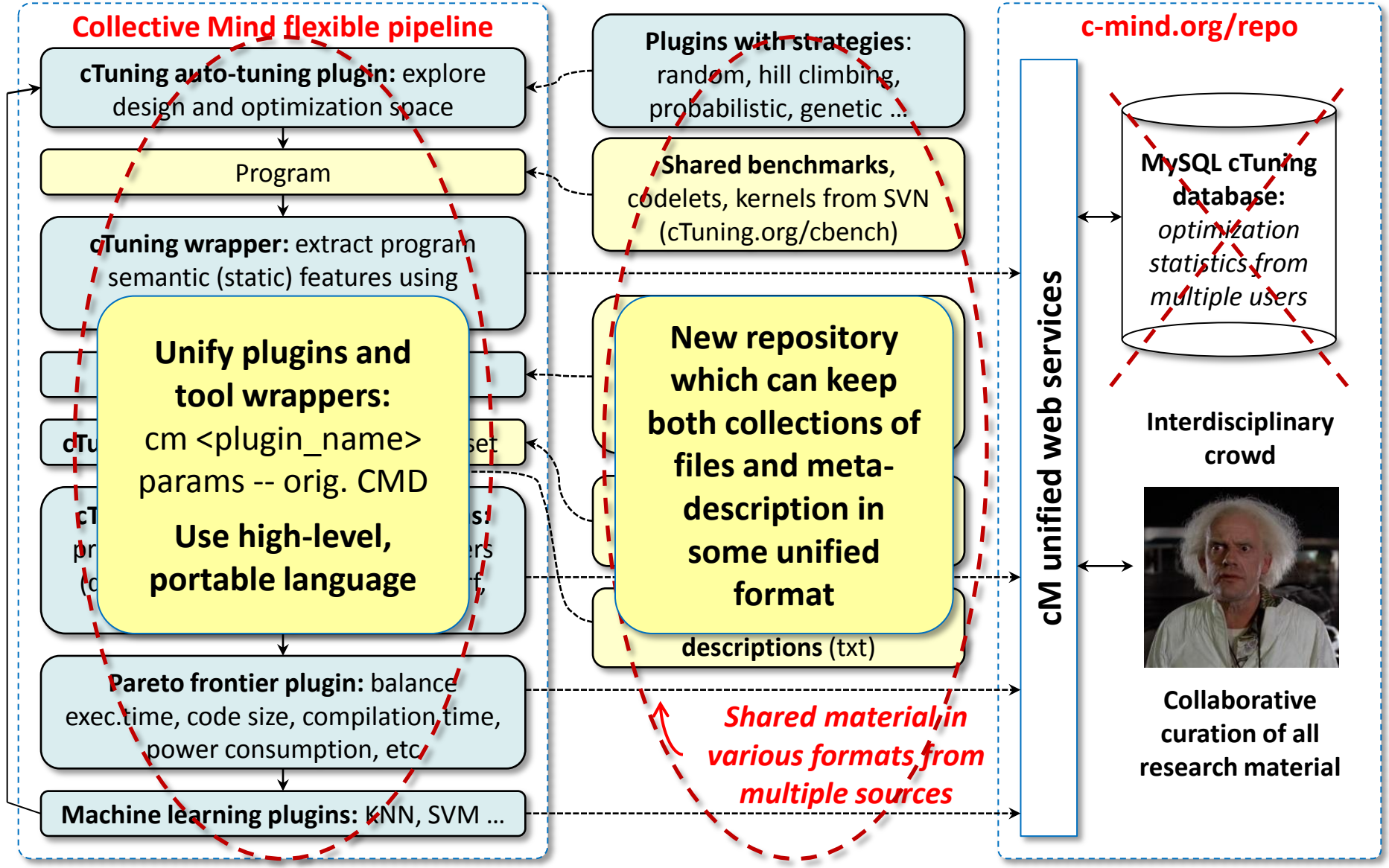
Common open-source Collective Mind framework (http://c-mind.org)

Collective Mind: attempt to fix cTuning problems



Common open-source Collective Mind framework (http://c-mind.org)

Collective Mind: attempt to fix cTuning problems



Common open-source Collective Mind framework (http://c-mind.org)

Collective Mind: which technology to use?

JSON: extensible and human-readable format

```
file data.json = {
  "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", ...}}
```

Can be directly indexed using web-based, distributed ElasticSearch (Apache2 license)

Python: high-level, portable, popular language with JIT, JSON support and many statistical and machine learning libraries

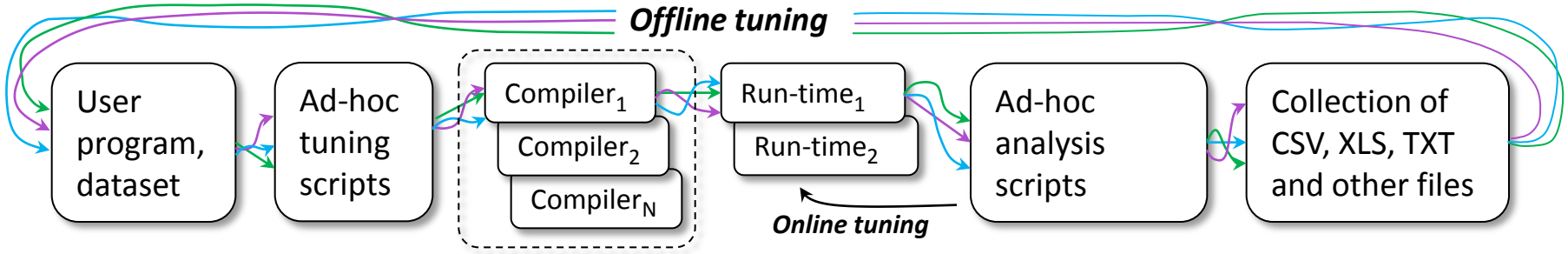
```
import json

f=file('data.json')
array=json.loads(f.read())
f.close()

et=array['characteristics']['execution_times']

for tm in et:
    print 'Execution time:', tm
```

Convert ad-hoc experiments to cM python modules



Hardwired experimental setups, very difficult to change, scale or share

Behavior

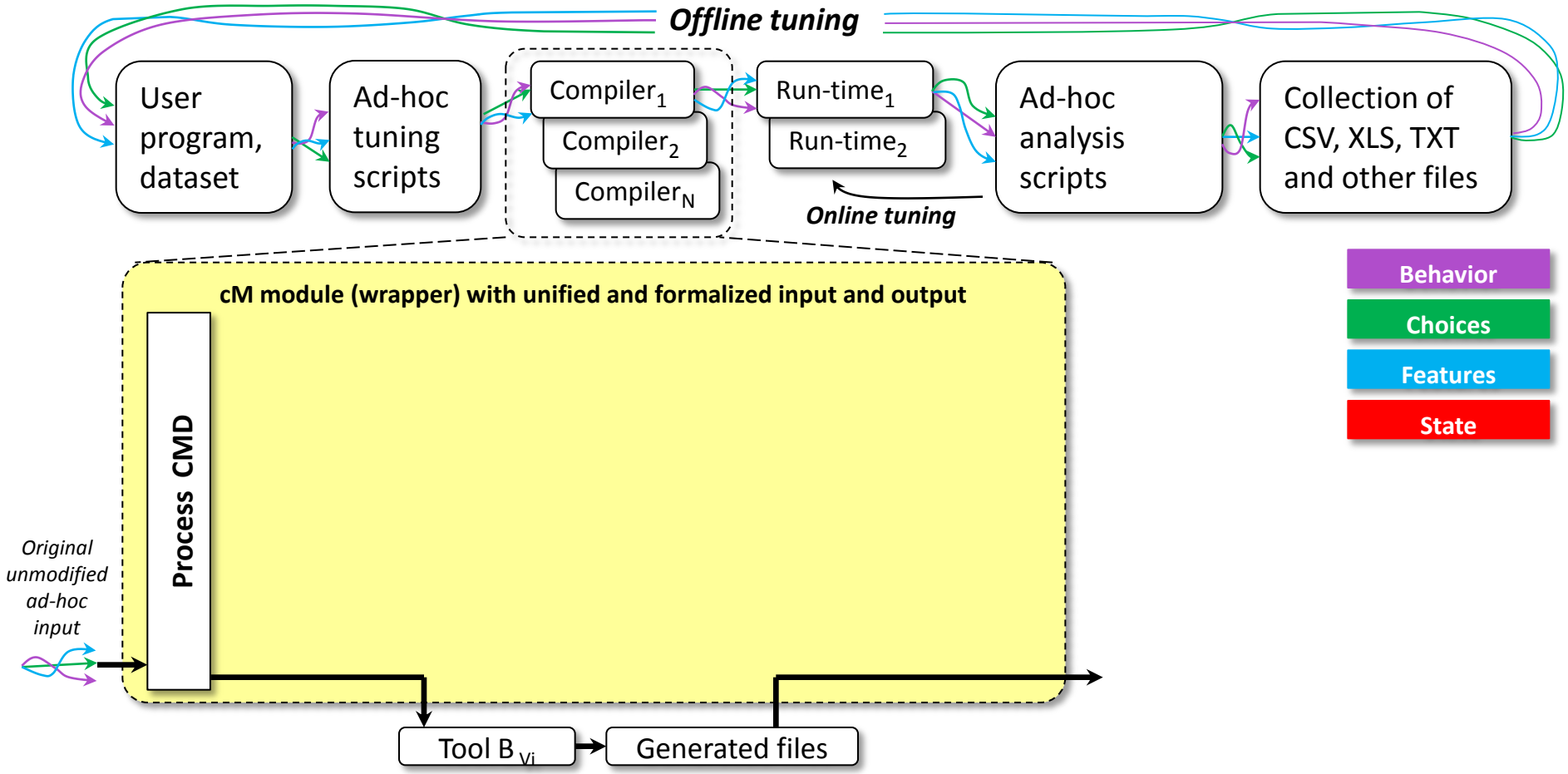
Choices

Features

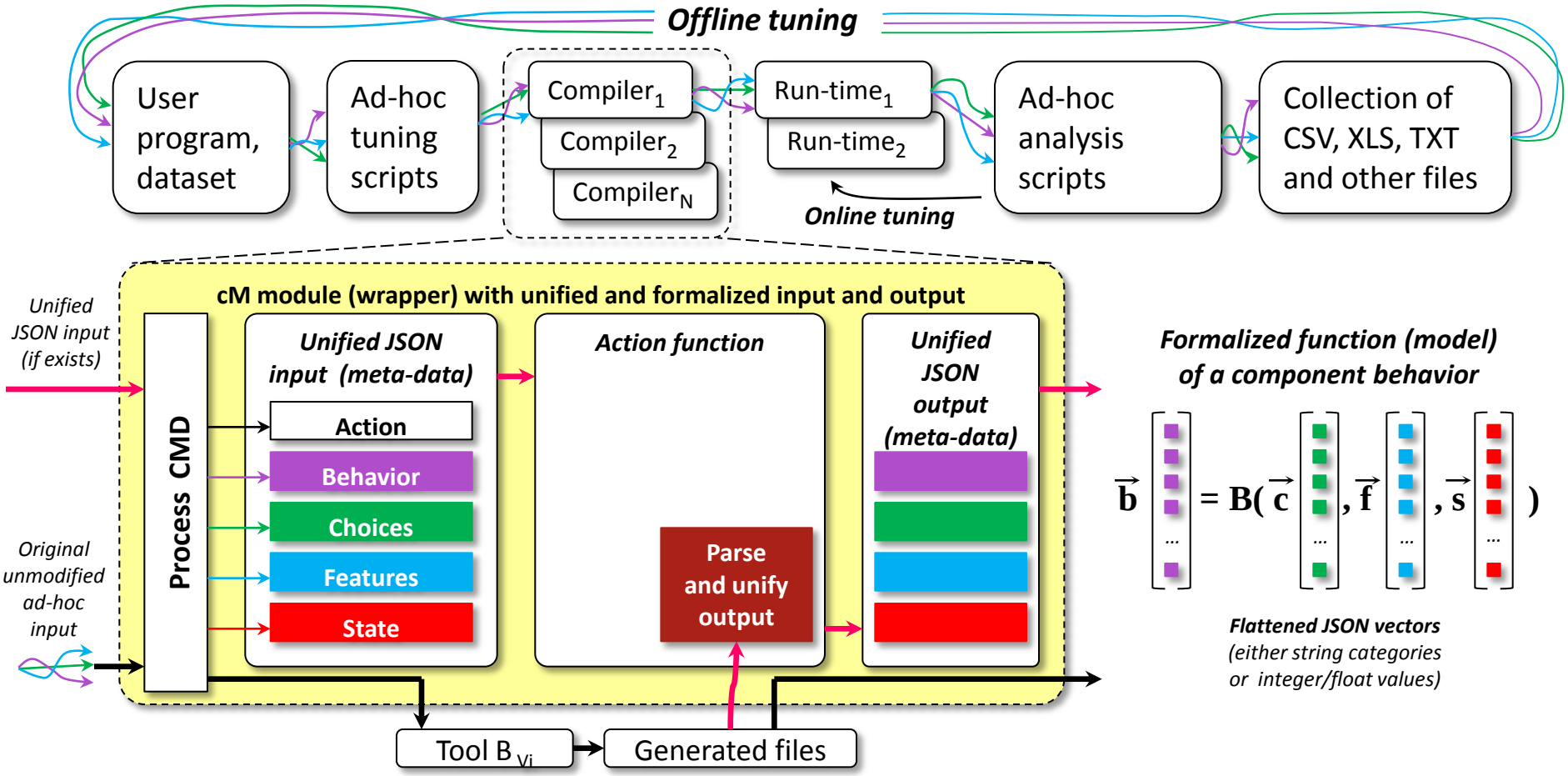
State

Meta description that should be exposed in the information flow for auto-tuning and machine learning

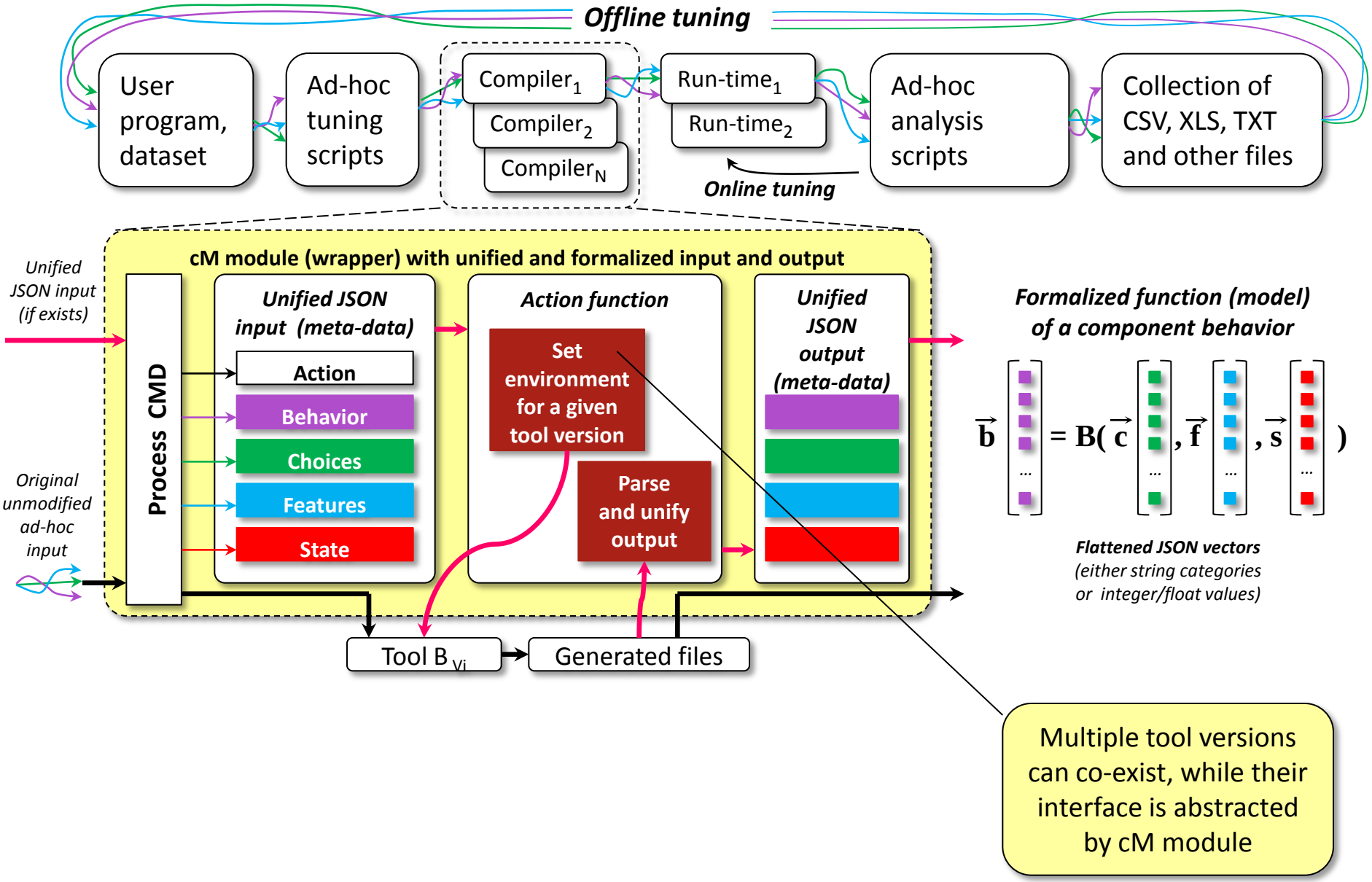
Convert ad-hoc experiments to cM python modules



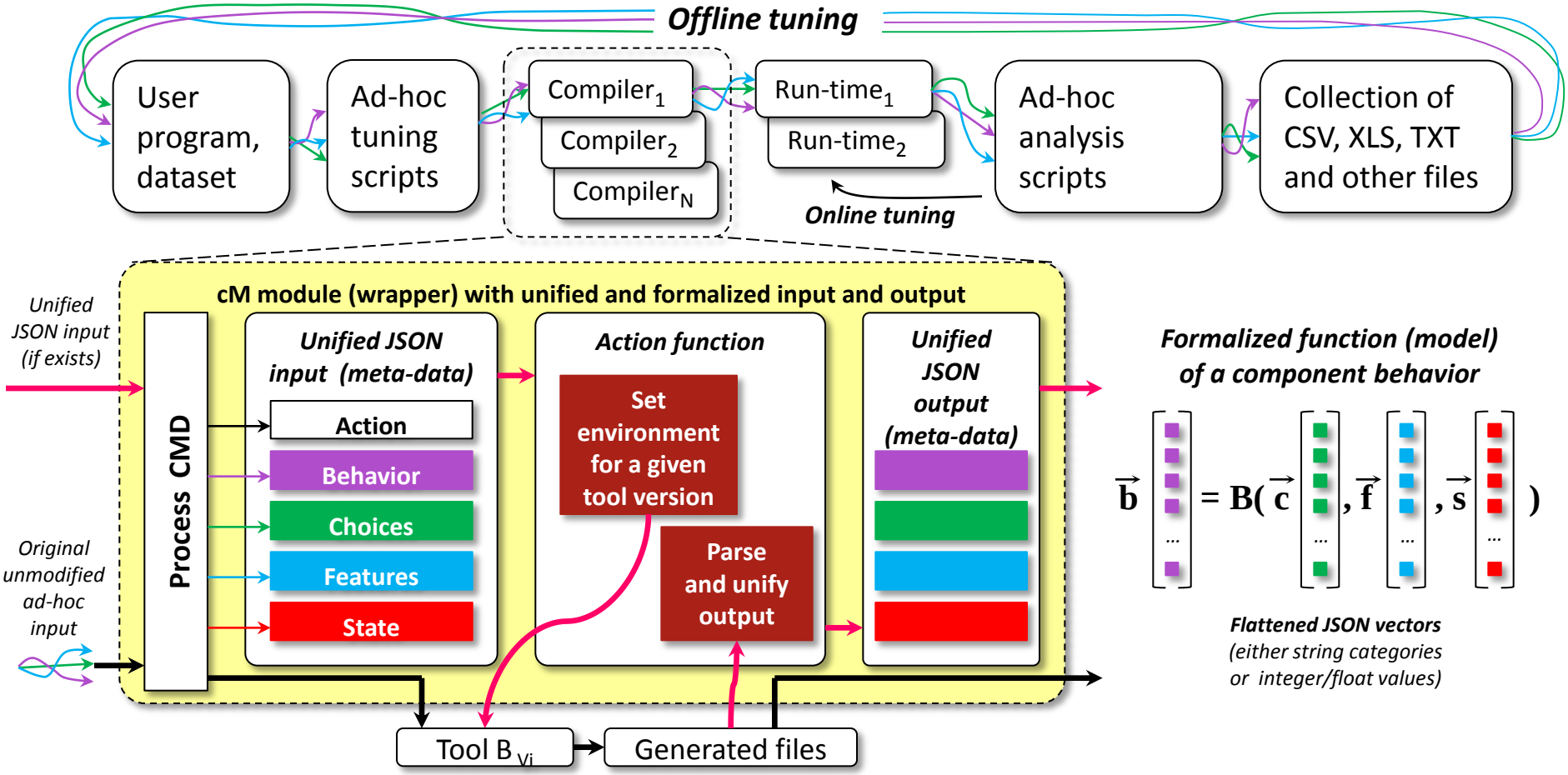
Convert ad-hoc experiments to cM python modules



Convert ad-hoc experiments to cM python modules



Convert ad-hoc experiments to cM python modules



cm ***[module name]*** [action] (param$_1$=value$_1$ param$_2$=value$_2$ … -- *unparsed command line*)
cm ***compiler*** build -- icc -fast *.c
cm ***code.source*** build ct_compiler=icc13 ct_optimizations=-fast
cm **code** run os=android binary=./a.out dataset=image-crazy-scientist.pgm

Should be able to run on any OS (Windows, Linux, Android, MacOS, etc)!

Data abstraction in Collective Mind

cM module

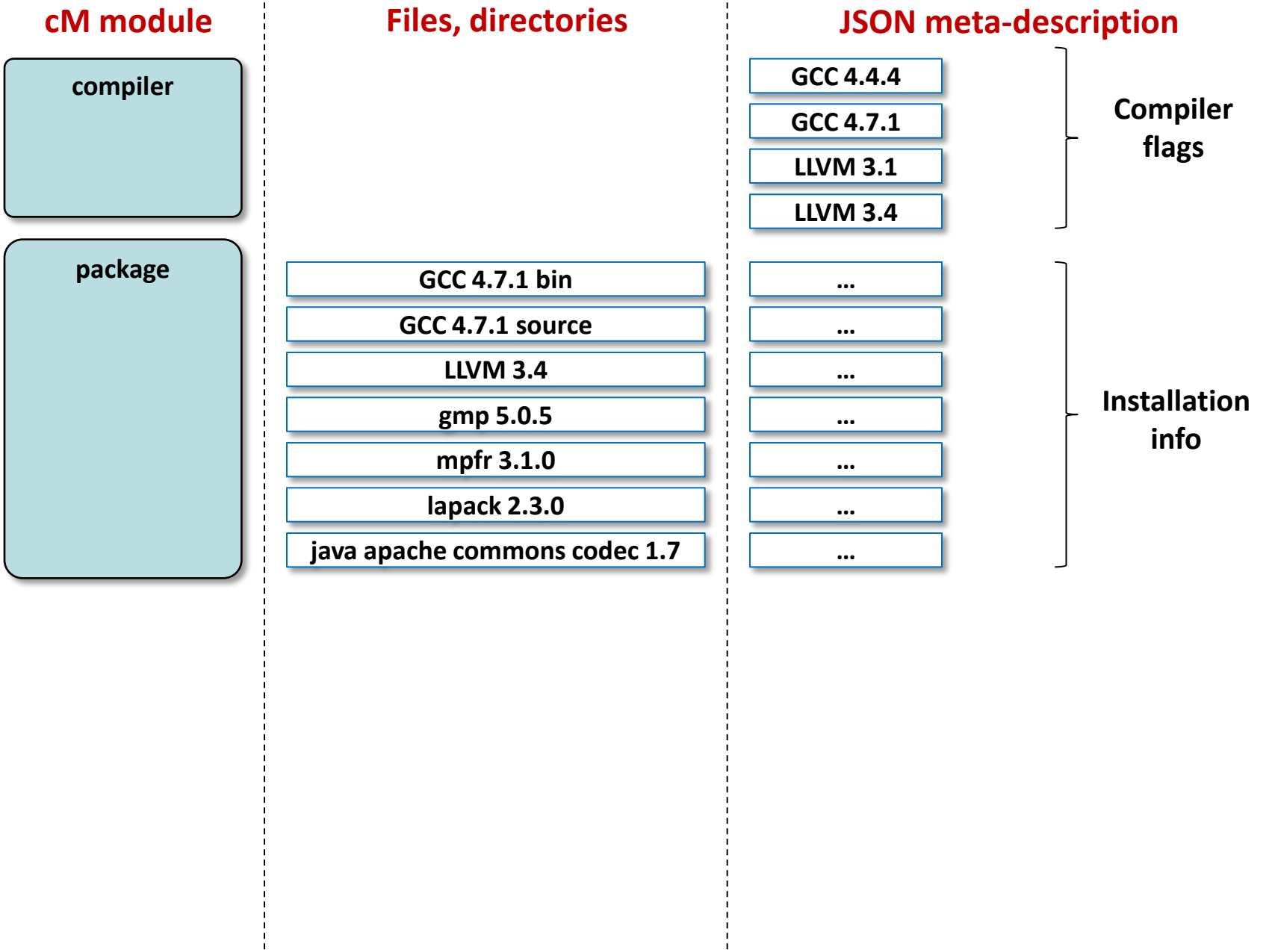
compiler

JSON meta-description

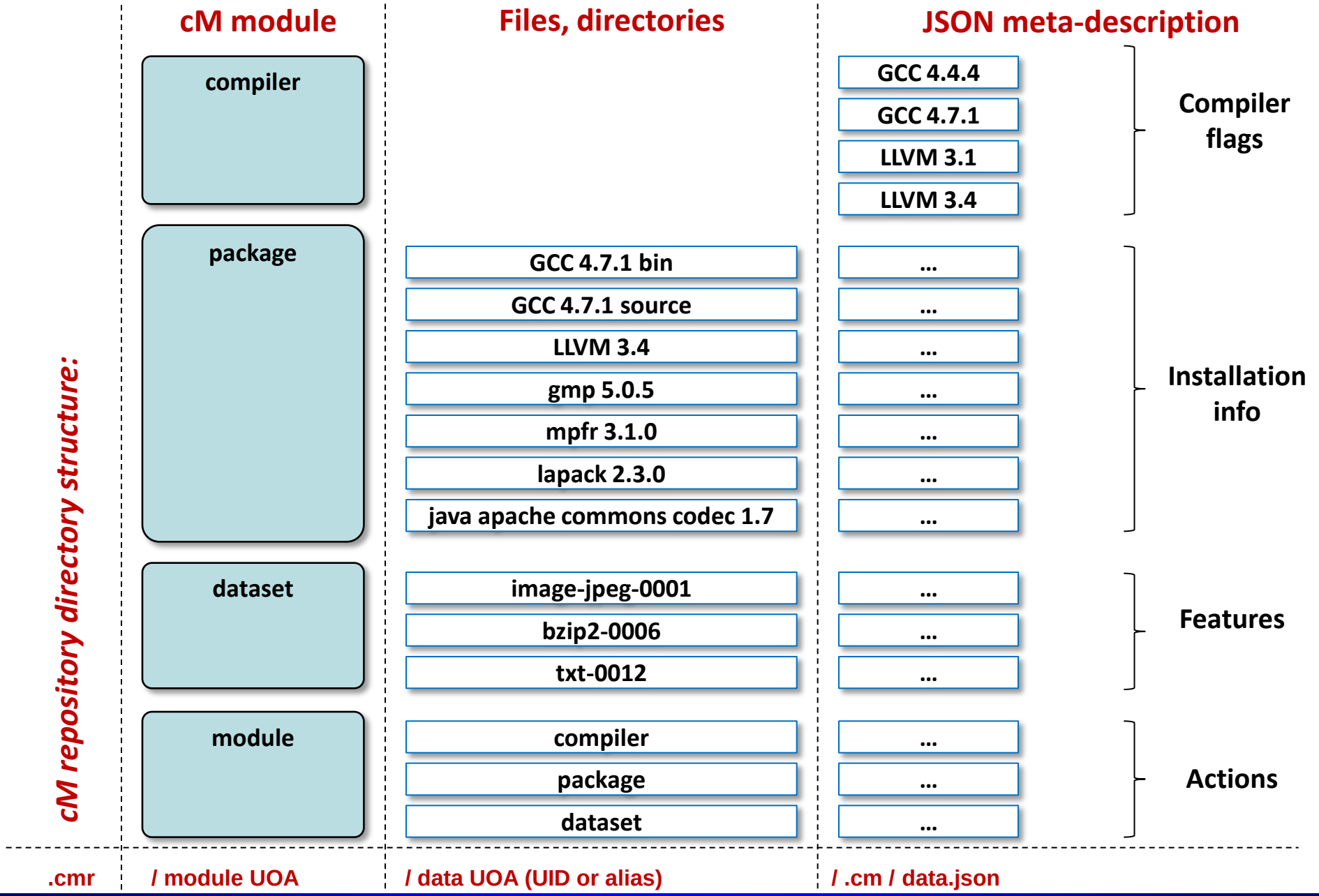
- GCC 4.4.4
- GCC 4.7.1
- LLVM 3.1
- LLVM 3.4

Compiler flags

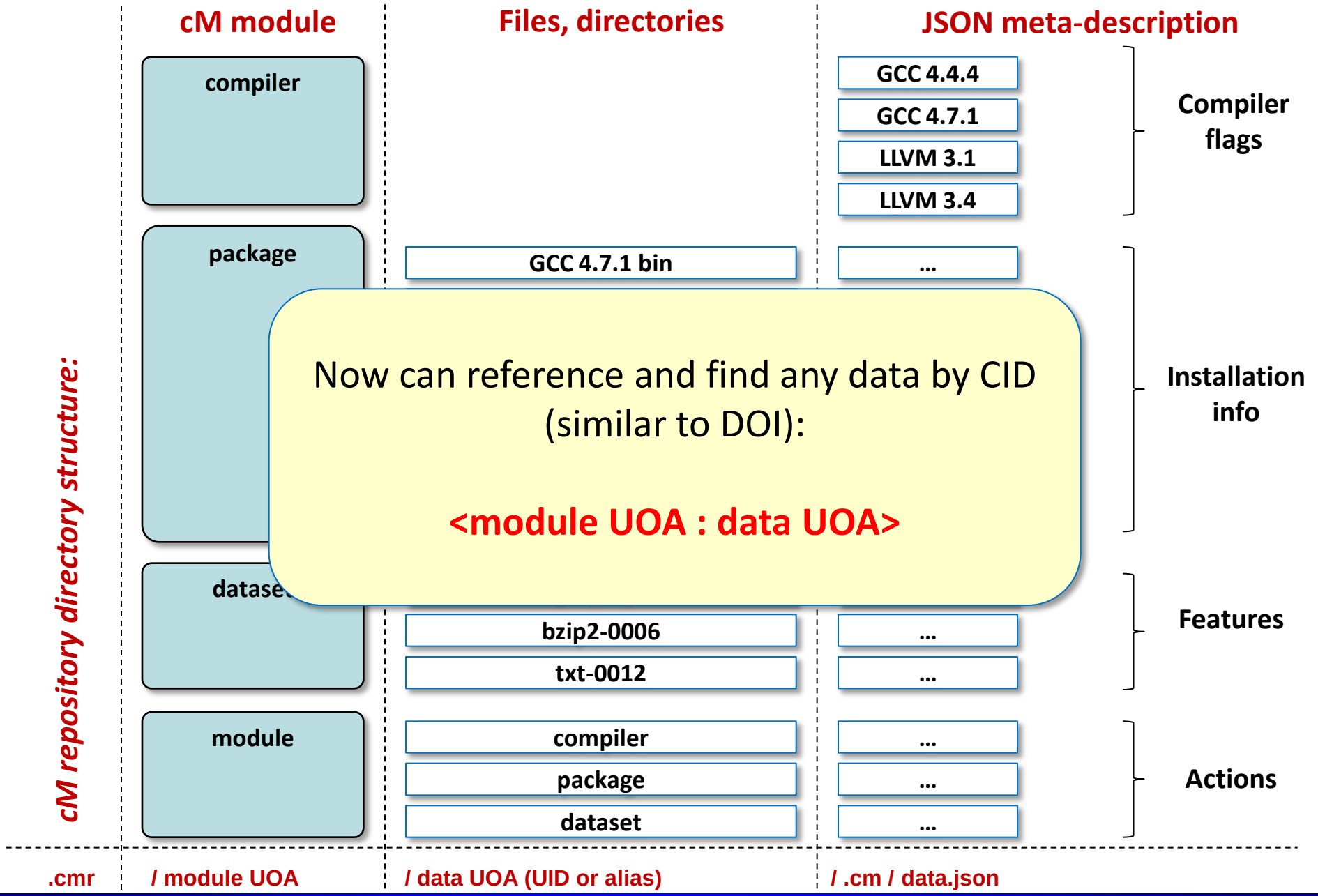
Data abstraction in Collective Mind



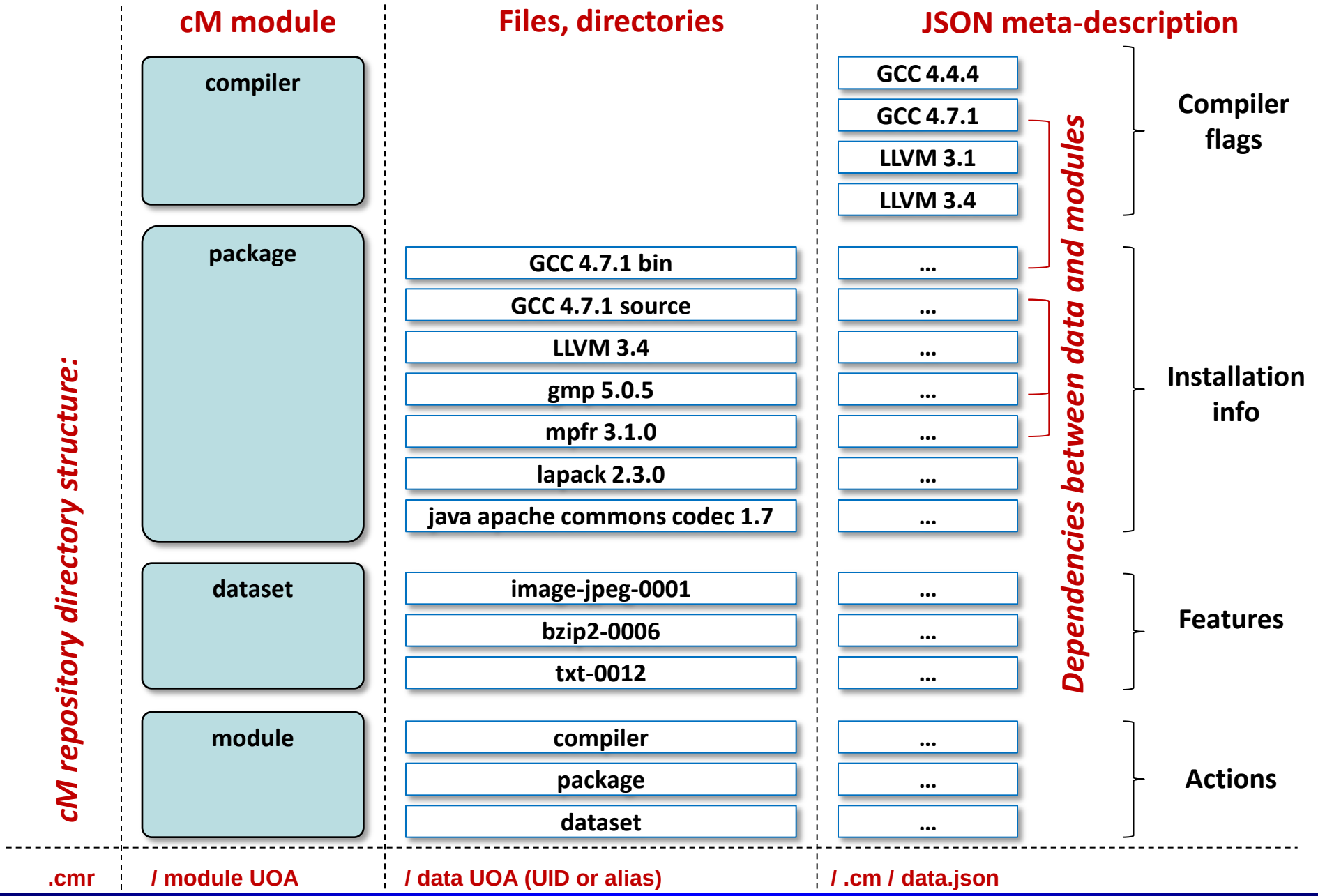
Data abstraction in Collective Mind



Data abstraction in Collective Mind

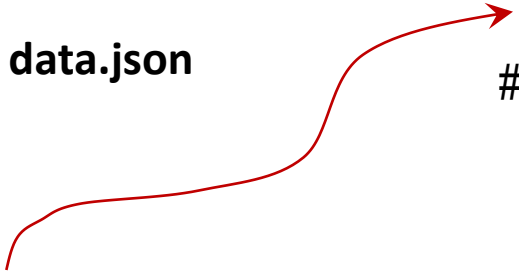


Data abstraction in Collective Mind



Gradually adding specification (agile development)

cTuning experiment module data.json



cM flattened JSON key

##characteristics#execution_times@1

```
{
  "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", ...}
}
```

Gradually adding specification (agile development)

cTuning experiment module data.json

```
{
  "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", ...}
}
```

cM 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],
        "explore_start": "start number if numerical
range",
        "explore_stop": "stop number if numerical
range",
        "explore_step": "step if numerical range",
        "can_be_omitted" : "yes" | "no"
        ...
}
```

Gradually adding, extending and improving modules and data

Category	cM module	Module actions	All data	Meta-description
Third-party tools, libraries	package	*common*, install*	...	...
High-level algorithms	algorithm	*common*, transform*	...	...
Applications, benchmarks, codelets, kernels	code.source	*common*, build*	...	...
Auto-tuning compilers	ctuning.compiler	*common*, compile_program*	...	...
Compiler machine learning plugins	math.model	*common*, build, predict, fit, detect_representative_points*	...	...
Binaries and libraries	code	*common*, run*	...	...
Data sets	dataset	*common*, create*	...	...
Operating Systems	os	*common*, detect_host_family*	...	...
Processors	processor	*common*, detect_host_processor*	...	...
Statistical and data mining plugins	math.statistics.r	*common*, analyze*	...	...

Transparently indexed by Elastic Search

** add, list, view, copy, move, search*

cM repository directory structure:	.cmr	/ module UOA (UID or alias)	/ data UOA	/ .cm / data.json

cM interface: only one main function!

Simple and minimalistic high-level cM interface - **one function (!)**

(python dictionary) *output* = cm_kernel.access ((python dictionary) *input*)

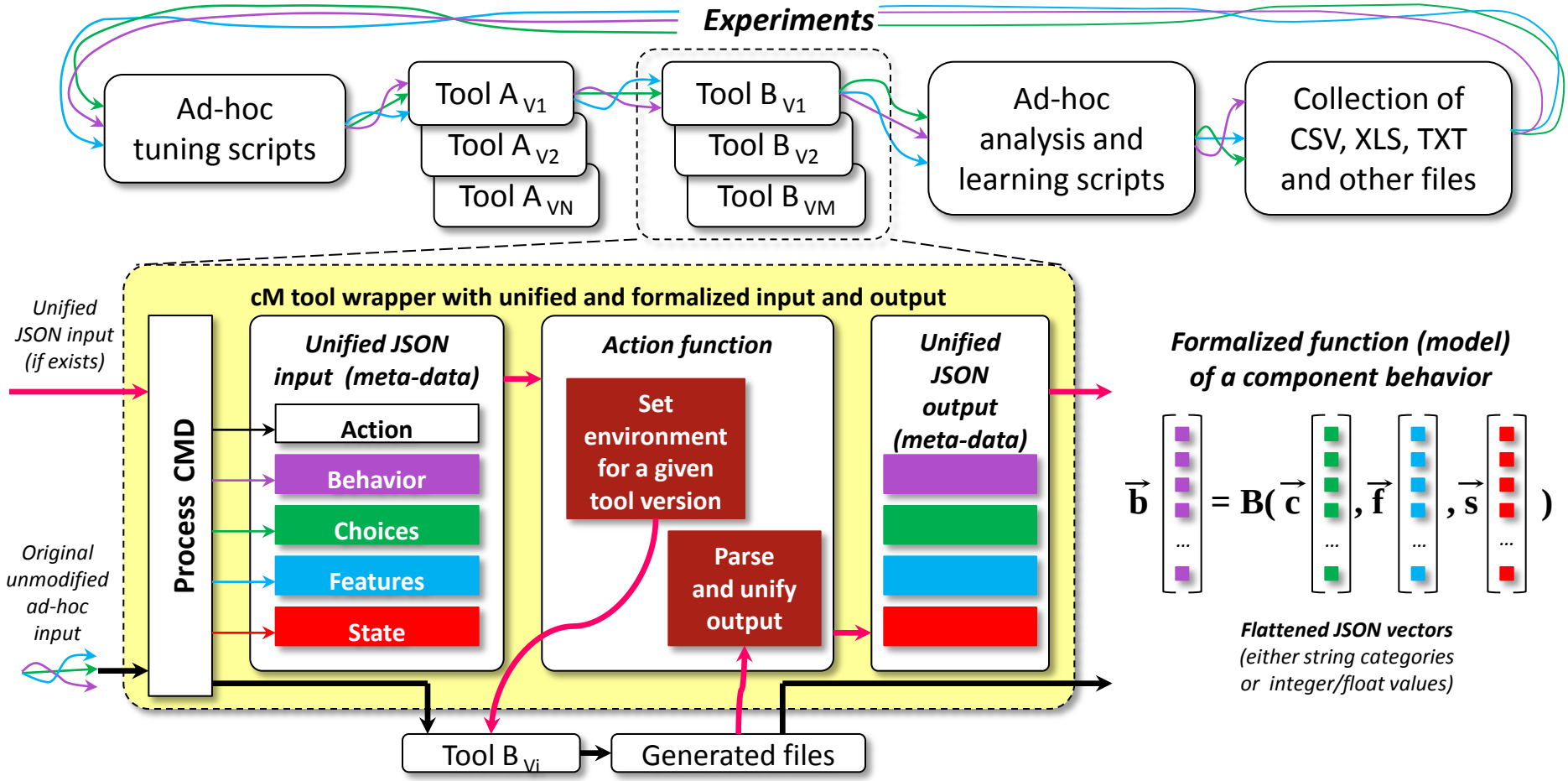
Load data:

```
r = cm_kernel.access({'cm_run_module_uoa':'dataset',
                      'cm_action':'load',
                      ' cm_data_uoa':'image-jpeg-0001'})
if r['cm_return']>0: return r
d=r['cm_data_obj']['cfg']
```

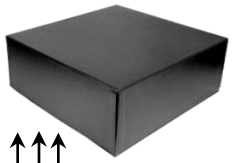
Call R machine learning module:

```
r = cm_kernel.access({'cm_run_module_uoa':' cm math.model.r',
                      'cm_action':' build',
                     'model_name':'earth', ...})
if r['cm_return']>0: return r
```

Assembling, preserving, sharing and extending the whole pipeline as "LEGO"



Top-down decomposition and learning of computer systems



		Gradually expose some characteristics	Gradually expose some choices and features
Compile Program		time …	compiler flags; pragmas …
Run code	Run-time environment	time; CPI, power consumption …	pinning/scheduling …
	System	cost;	architecture; frequency; cache size…
	Data set	size; values; description …	precision …
Analyze profile		time; size …	instrumentation; profiling …

I now gradually convert to Collective Mind and validate past research techniques with the great help of volunteers (sadly such work is usually not funded, can't be easily published, and is often considered as a waste of time by academic community)

Methodology similar to physics: start from coarse-grain and gradually move to fine-grain level!

Start coarse-grain decomposition of a system (detect coarse-grain effects first). Add universal learning modules.

Growing, plugin-based cM pipeline for auto-tuning and learning

http://c-mind.org/ctuning-pipeline

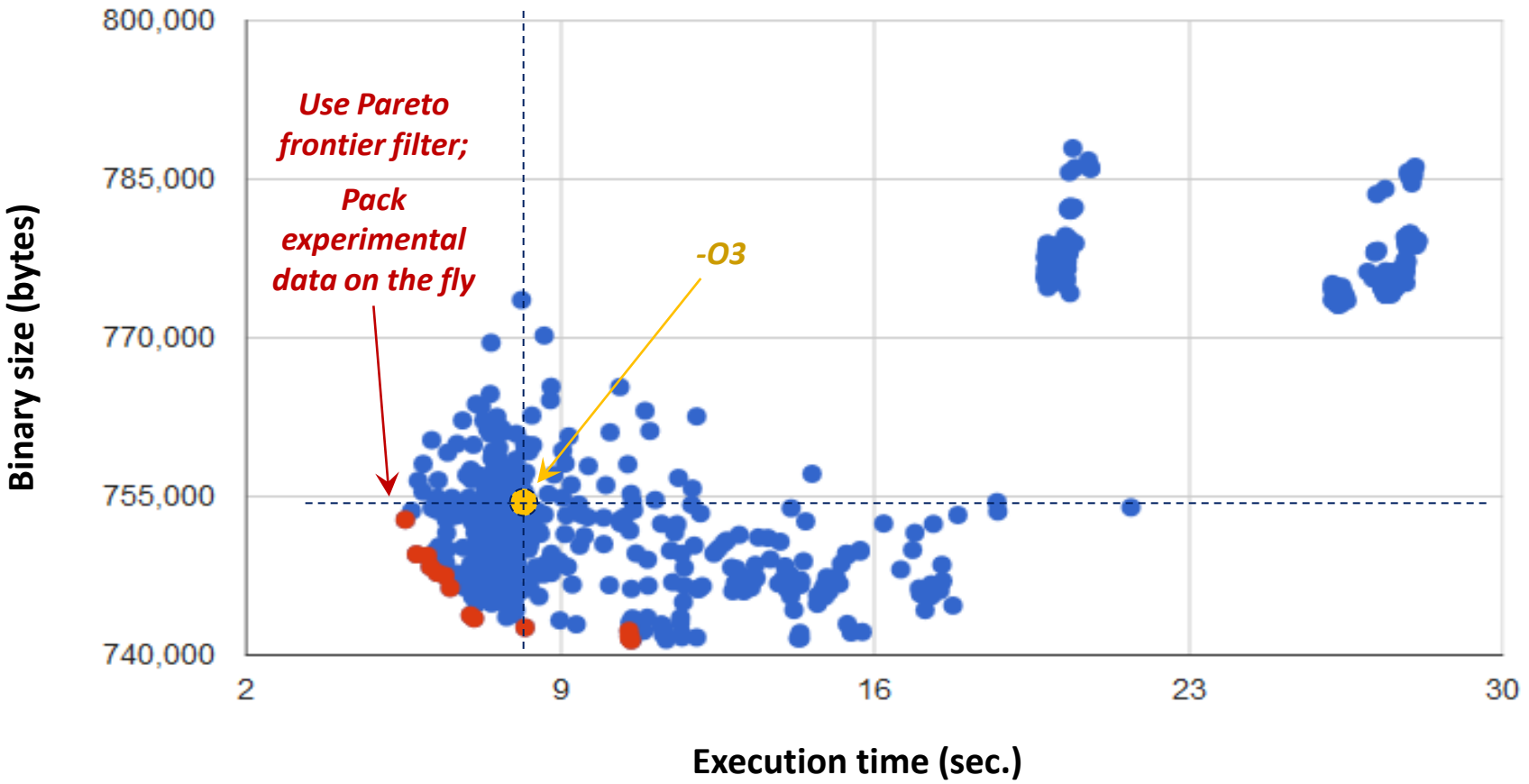
- **Init pipeline**
- Detected system information
- Initialize parameters
- Prepare dataset
- **Clean program**
- **Prepare compiler flags**
- Use compiler profiling
- Use cTuning CC/MILEPOST GCC for fine-grain program analysis and tuning
- Use universal Alchemist plugin (with any OpenME-compatible compiler or tool)
- Use Alchemist plugin (currently for GCC)
- **Build 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**

Publicly shared research material (c-mind.org/repo)

Our Collective Mind Buildbot and plugin-based auto-tuning pipeline supports the following shared benchmarks and codelets:

- Polybench - numerical kernels with exposed parameters of all matrices in cM
  - CPU: 28 prepared benchmarks
  - CUDA: 15 prepared benchmarks
  - OpenCL: 15 prepared benchmarks
- cBench - 23 benchmarks with 20 and 1000 datasets per benchmark
- Codelets - 44 codelets from embedded domain (provided by CAPS Entreprise)
- SPEC 2000/2006
- Description of 32-bit and 64-bit OS: Windows, Linux, Android
- Description of major compilers: GCC 4.x, LLVM 3.x, Open64/Pathscale 5.x, ICC 12.x
- Support for collection of hardware counters: perf, Intel vTune
- Support for frequency modification
- Validated on laptops, mobiles, tables, GRID/cloud - can work even from the USB key

Implemented scenario: multi-objective compiler auto-tuning using mobile phones



Program:	*image corner detection*	Processor:	*ARM v6, 830MHz*
Compiler:	*Sourcery GCC for ARM v4.7.3*	OS:	*Android OS v2.3.5*
System:	*Samsung Galaxy Y*	Data set:	*MiDataSet #1, image, 600x450x8b PGM, 263KB*

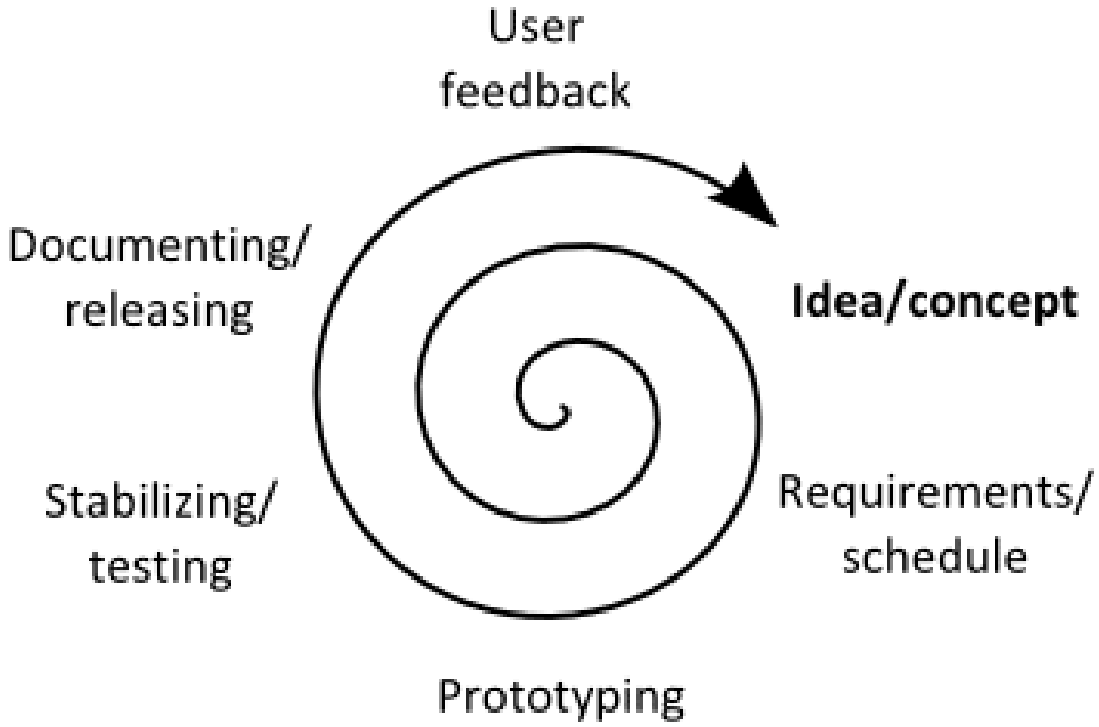
*500 combinations of random flags -O3 -f**(no-)**FLAG*

Powered by Collective Mind Node (Android Apps on Google Play)

Collective Mind agile research and development

From agile development to agile research!

Community shares, validates and improves benchmarks, data sets, tools, design and optimization choices, features, characteristics, models, …



Collective Mind experimental pipelines can now survive changes in the system!
Modules and data can evolve with the evolution of the technology!

Reproducibility of experimental results

Reproducibility came as a side effect!

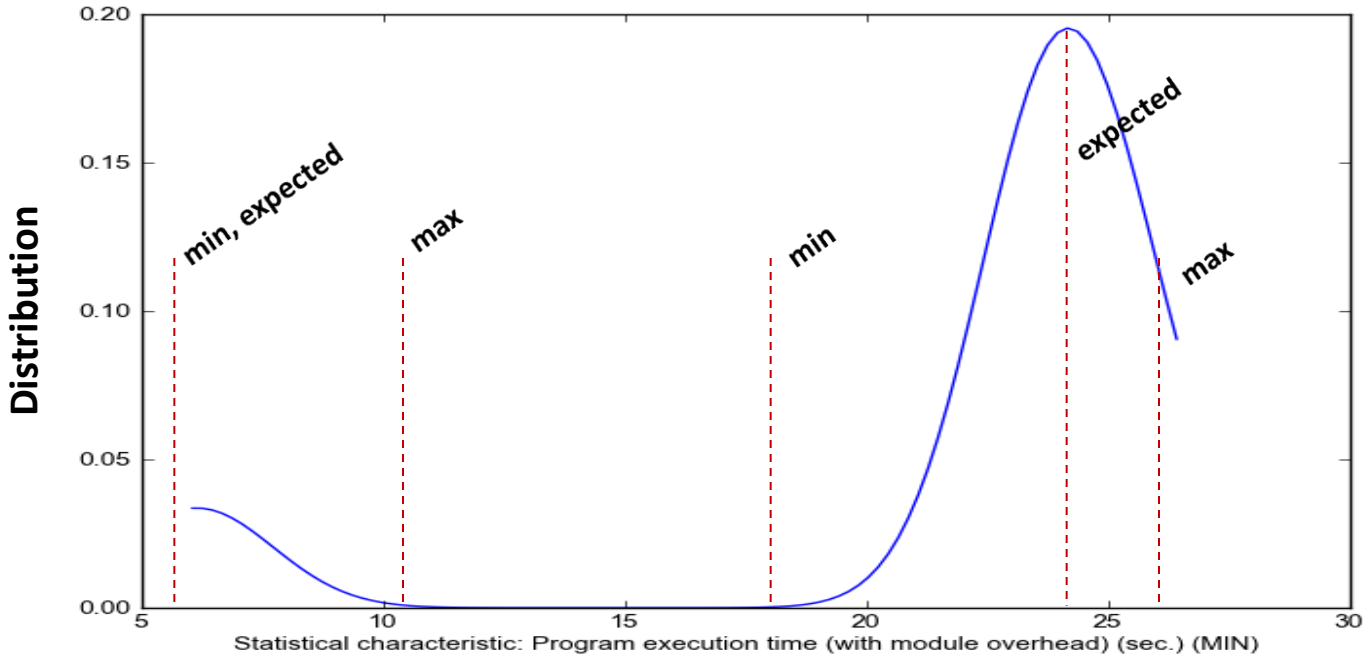
- Can preserve the whole experimental setup with all data and software dependencies
- Can perform statistical analysis (normality test) for characteristics
- Community can add missing features or improve machine learning models

Reproducibility of experimental results

Reproducibility came as a side effect!

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

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

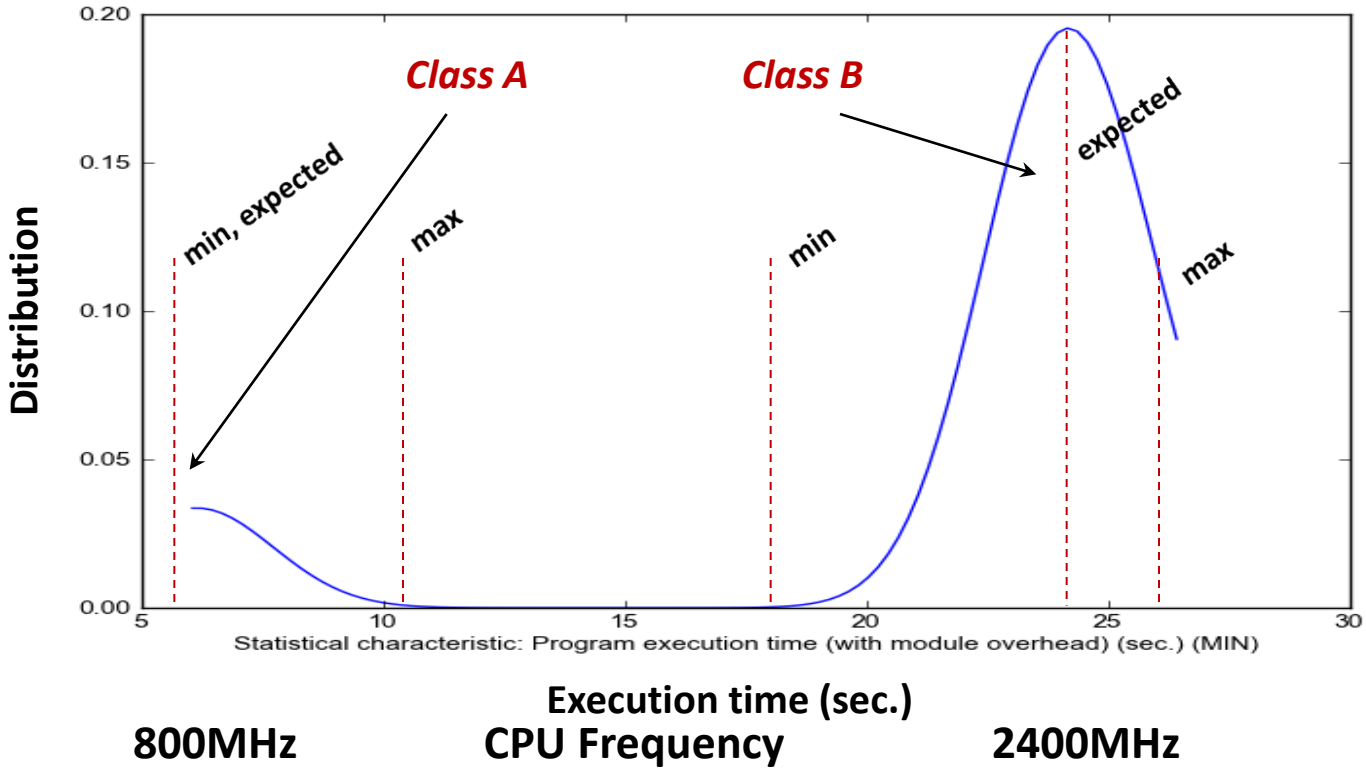


Reproducibility of experimental results

Reproducibility came as a side effect!

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

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

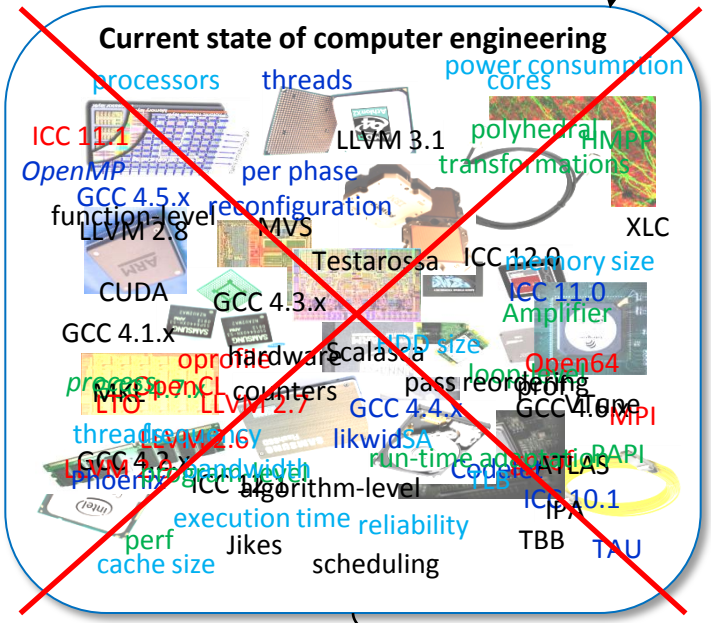


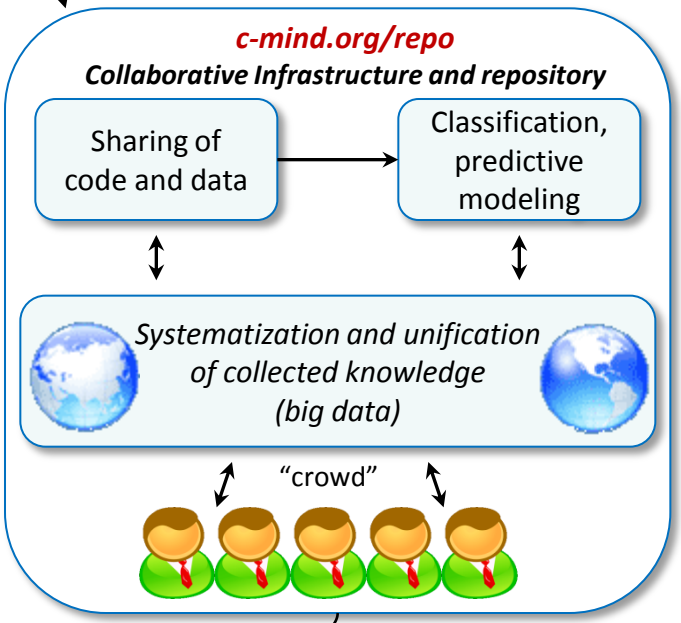
My dream: systematic, collaborative and reproducible computer engineering

- Prototype research idea
- Validate existing work
- Perform end-user task

- Quick, non-reproducible hack?
- Ad-hoc heuristic?
- Quick publication?
- No shared code and data?

- Share code, data with their meta-description and dependencies
- Systematize and classify collected optimization knowledge
- Develop and preserve the whole experimental pipeline
- Validate experimental results by the community
- Extrapolate collected knowledge to build faster, smaller, more power efficient and reliable computer systems





Result

Current status and future work

- Pilot live repository for public curation of research material: **http://c-mind.org/repo**
- Infrastructure is available at SourceForge under standard BSD license: **http://c-mind.org**
- Example of crowdsourcing compiler flag auto-tuning using mobile phones: "**Collective Mind Node**" in Google Play Store
- Several publications under submission
- Raising funding to make cM more user friendly and add more research scenarios

Education	Academic research
New publication model where research material and experimental results are shared, validated and reused by the community http://ctuning.org/reproducibility • **ACM SIGPLAN TRUST 2014 @ PLDI 2014** http://c-mind.org/events/trust2014 • **Panel at ADAPT 2014 @ HiPEAC 2014** http://adapt-workshop.org (this year we attempted to reproduce some submitted articles)	 • Systematizing, validating, sharing past research techniques on auto-tuning and machine learning through cM pipeline • Optimal feature and model selection • Finding representative benchmarks and data sets • Run-time adaptation and ML

Grigori Fursin, "Collective Mind: cleaning up the research and experimentation mess in computer engineering using crowdsourcing, big data and machine learning", INRIA Tech. report No 00850880, August 2013

http://hal.inria.fr/hal-00850880 **http://arxiv.org/abs/1308.2410**

Acknowledgements

- Colleagues from NCAR (USA): ***Davide Del Vento et.al.***
- Colleagues from STMicroelectronics (France):

  Christophe Guillone, Antoine Moynault, Christian Bertin
- Colleagues from ARM (UK): **Anton Lokhmotov**
- Colleagues from Intel (USA): ***David Kuck and David Wong***
- cTuning community:

  http://cTuning.org/lab/people

- EU FP6, FP7 program and HiPEAC network of excellence

  http://www.hipeac.net
- INRIA fellowship (2012-2016)

Thank you for your attention!

Questions and comments: grigori.fursin@inria.fr