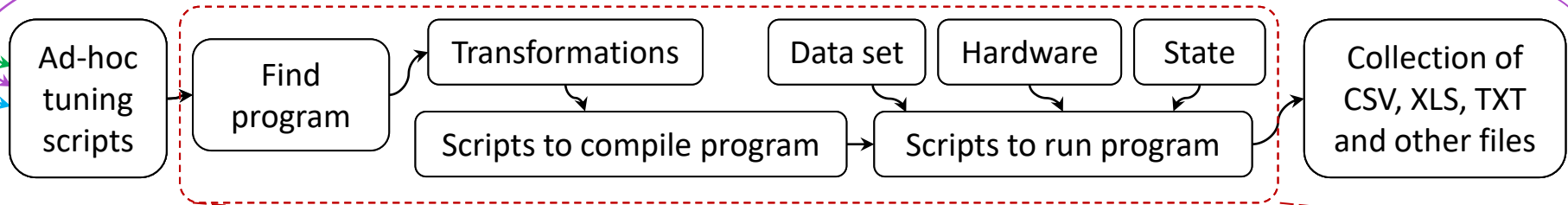


## Typical experimental workflow



# Example of CK module **"program"**  
# (unified dict/JSON API)

```
import ck.kernel as ck
```

```
def compile(i):
```

```
    # Process CMD
```

```
    prog=i.get('data_uoa',")
```

```
    flags=i.get('flags',"")
```

```
    speed=i.get('speed',"")
```

```
    # Get program path, UID and meta(dict)
```

```
    r=ck.access({'action':'load',  
                'module_uoa':'program',  
                'data_uoa':prog})
```

```
    if r['return']>0: return r
```

```
    path=r['path']
```

```
    uid=r['data_uid']
```

```
    meta=r['dict']
```

```
    ...
```

```
    return {'return':0}
```

## CK Python modules (tool wrappers) with a unified JSON API

CK input (JSON/dict)

Read unified program JSON meta data  
to know how to compile and run it

Detect required SW/HW dependencies

Set environment for selected SW& HW  
(including compilers and libs)

Compile

Run

AutoTune

CK output (JSON/dict)

Tools (compilers, profilers, etc)

Generated files

# **ck** – simple command line front-end to manage CK repositories,

# call CK modules, abstract tools and unify input/output

```
ck compile program:cbench-automotive-susan --speed --flags=-O3 ...
```

```
ck run program:cbench-automotive-susan
```

```
ck autotune program:cbench-automotive-susan
```