LISP-Like DSL for Benchmarking

Renat Idrisov

LISP-like DSL for Benchmarking

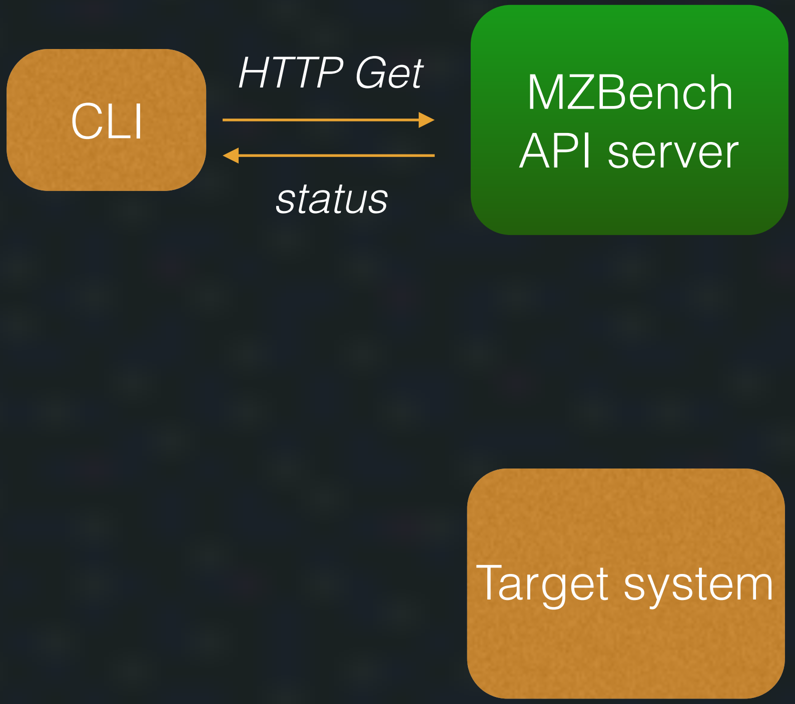
Presentation plan:

Our workflow

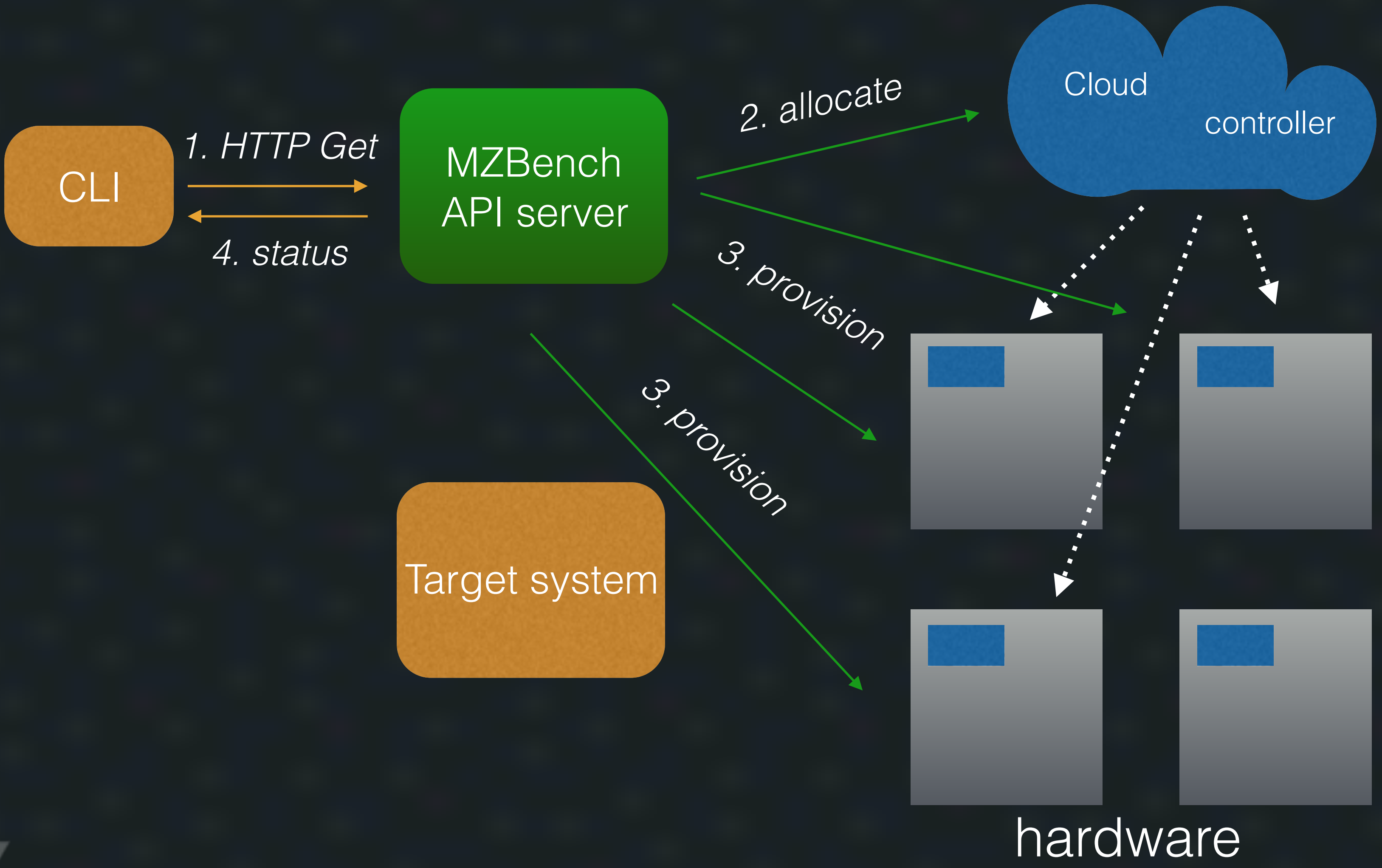
Why do we need a DSL?

Implementation milestones

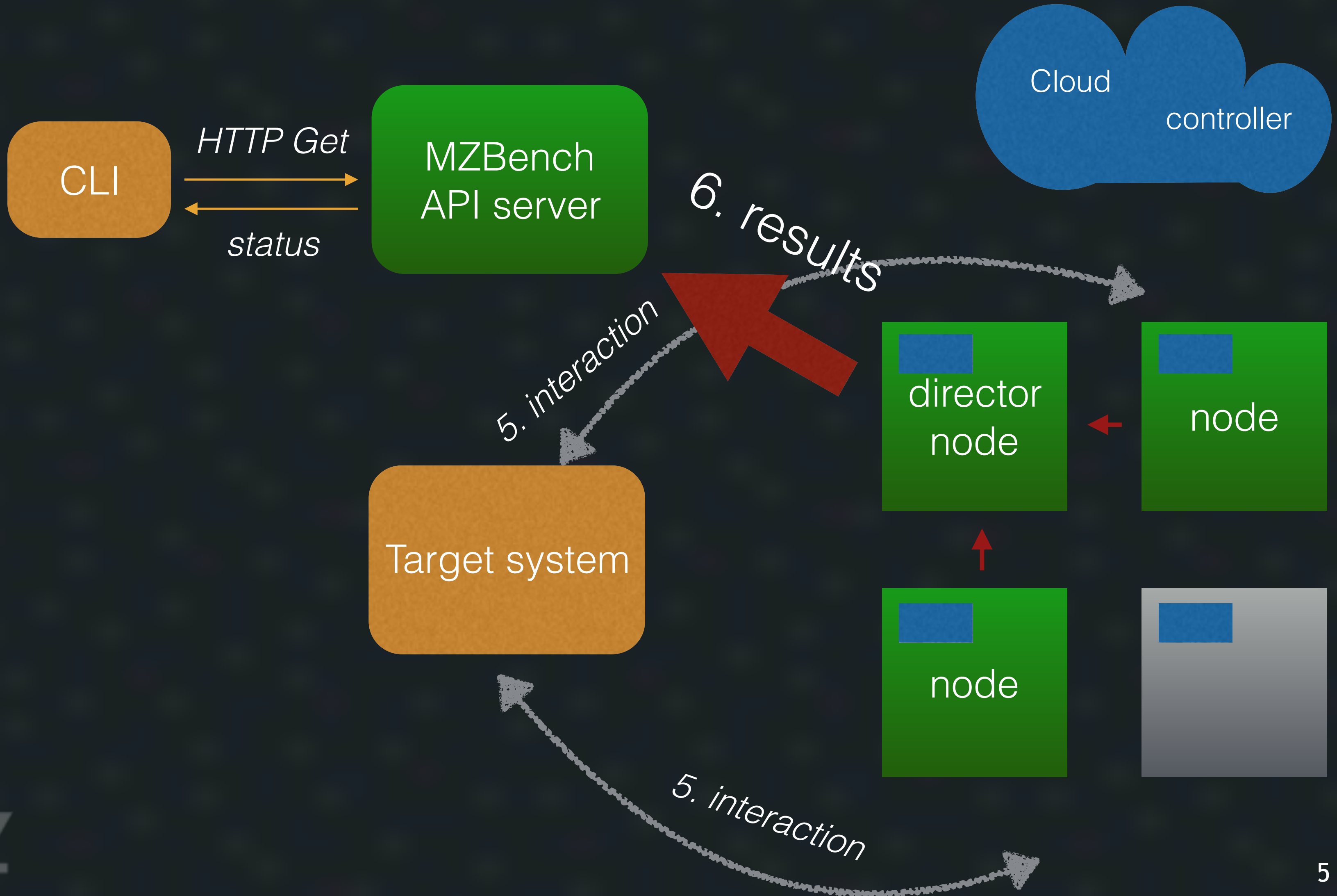
MZBench workflow



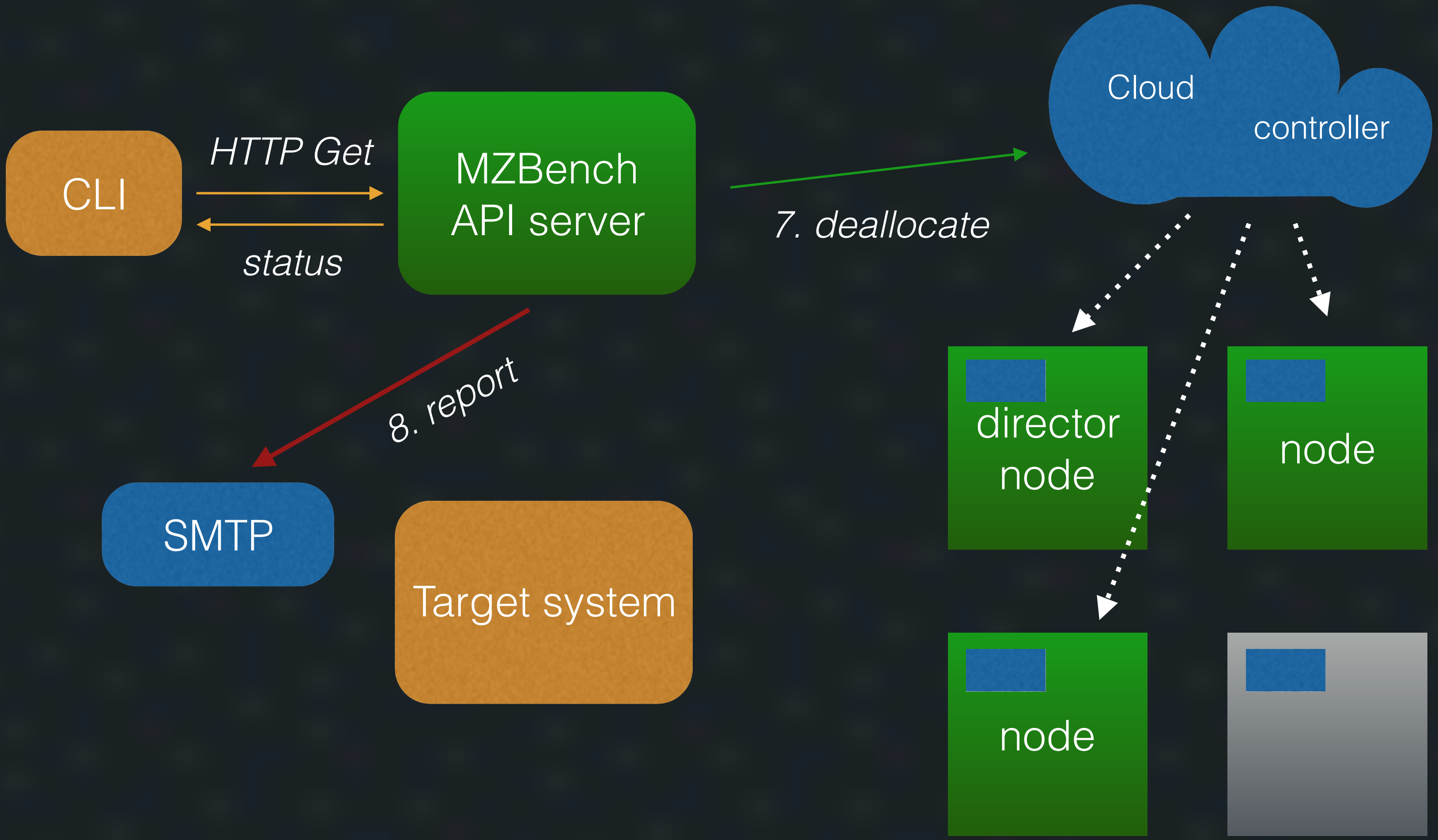
MZBench workflow



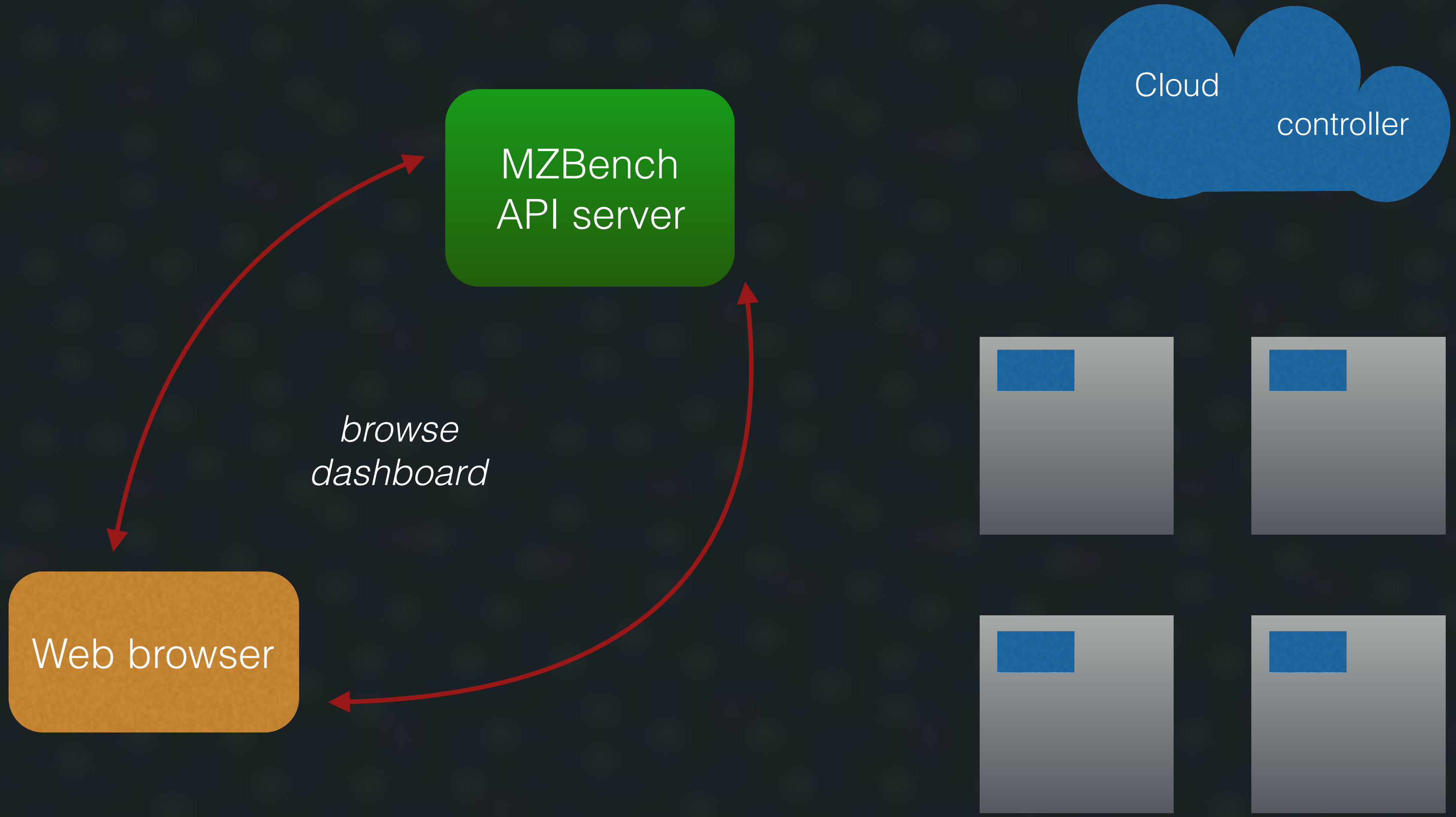
MZBench workflow



MZBench workflow



MZBench workflow



Why do we need a DSL?

Four levels of expertise

Ability to change test parameters

– rates, message sizes, cluster size, time…

Ability to change test scenario (in DSL)

– load profile, packet sequences…

Ability to add new protocol

– particular database, queue…

Ability to add new core functionality

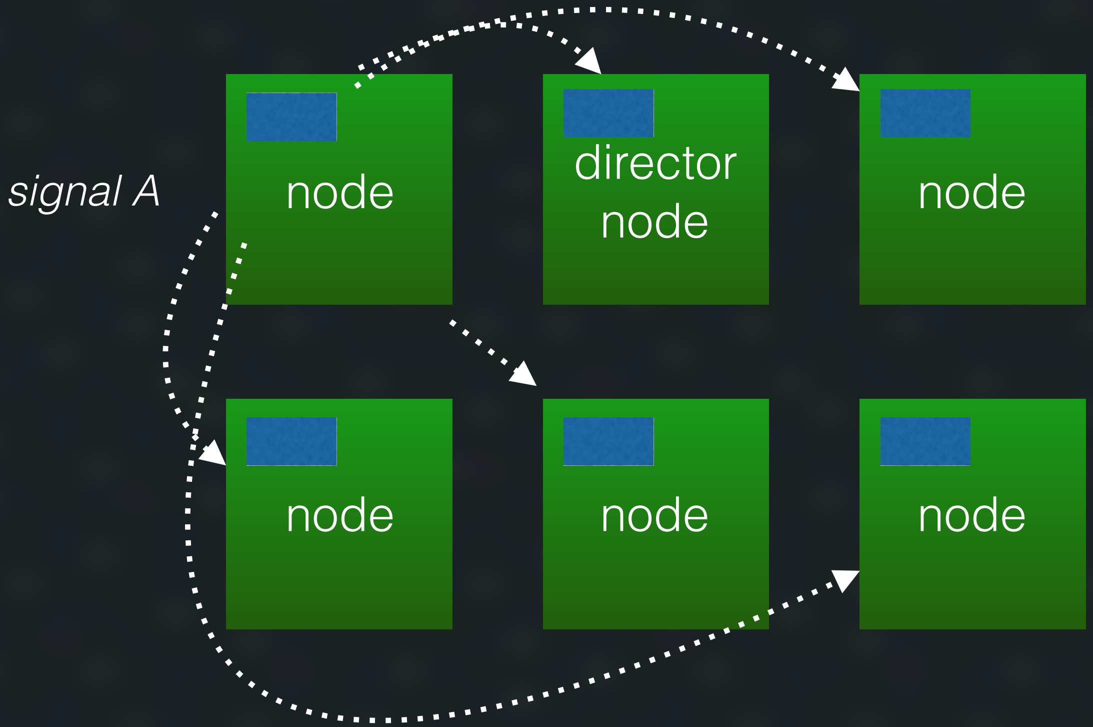
– improve signal analyzer…

Limitation

The language is simple:
- easy to start
- all information gathered implicitly

A person who works on test scenario could not:
- affect system scalability
- make a deadlock

Synchronization analysis



Static signal graph analysis

Implementation milestones

file:consult

```
[{assert, always, {gt, "http_ok.rps.value", 0.5}},
  {make_install, [{git, "https://github.com/machinezone/mzbench.git"},
    {dir, "workers/simple_http"}]}, % sub-folder in git repo
  {pool, [{size, {numvar, "worker_count", 20}}, % 20 parallel "threads"
      {worker_type, simple_http_worker}],
    [{loop, [{time, {120, sec}},
        {rate, {ramp, linear, {1, rps}, {{numvar, "max_rps", 200},
rps}}}],
      [{get, {var, "target_url", "http://172.21.3.3/index.html"}}]}]}]}].
```

More detailed error messages

erl_parse:parse_exprs

ast transform

erl_parse:normalise

```
{cons,1,
          {tuple,1,[{atom,1,size},
                   {integer,1,3}]},
          {cons,2,
           {tuple,2,[{atom,2,worker_type},
                    {atom,2,dummy_worker}]}
```

Lightweight CLI

Easy to be parsed from Python

```
    grammar = Grammar("""\
    entry = (term _ "." _)* _
    term = boolean / atom / list / tuple / map /
string / binary / number
    atom = ~"[a-z][0-9a-zA-Z_]*" / ("'" ~"[^']*" "'")
    _ = ~"\s*"
...
```

Additional frontend

```
#!benchDL
# total number of print operations should be greater than 200 at least
# for 30 seconds
assert(30 sec, "print" > 200)
assert(always, "workers.pool1.failed" == 0)
# number of failed workers should always be 0
assert(always, 9 < "print.rps") # 9 should be always less than print rate
pool(size = 1, worker_type = dummy_worker):
# one execution "thread"
    loop(time = 1 min, # total time is 1 minute
         rate = 10 rps): # constant rate is 10 operations per second
            print("FOO") # this operation prints "FOO" to console
```

DSL

Limited

Easy to analyse

Expressive

Infrastructure independent

MZBench

Cloud-aware
Extendable
Scalable
Open-source

Distributed metric subsystem
Python and Lisp-like DSLs for scenarios
Python/Erlang/Lua for extensions

Questions?