Icarus: a Caching Simulator for Information Centric Networking

Lorenzo Saino, Ioannis Psaras and George Pavlou

Department of Electronic and Electrical Engineering
University College London

`http://icarus-sim.github.io`

Introducing Icarus

What is Icarus?

- Simulator for evaluating caching performance in ICN
- Not bound to any specific ICN architecture
- Design is generic enough to make it suitable to simulate any generic networked caching systems (KV stores, CDNs, content routers)

What Icarus is <u>not</u>?

- Not a suitable tool to evaluate other aspects of ICN architectures such as security, naming, congestion control, routing scalability

Requirements for caching simulators

General requirements:

- Reliability and accuracy
- Easy to use, fast iteration cycles
- Rich library of models, algorithms, protocols

Specific requirements:

- Large realistic topologies
- Large content catalogues and many content requests to allow caches to reach steady-state
- Support trace-driven simulations

Icarus objectives

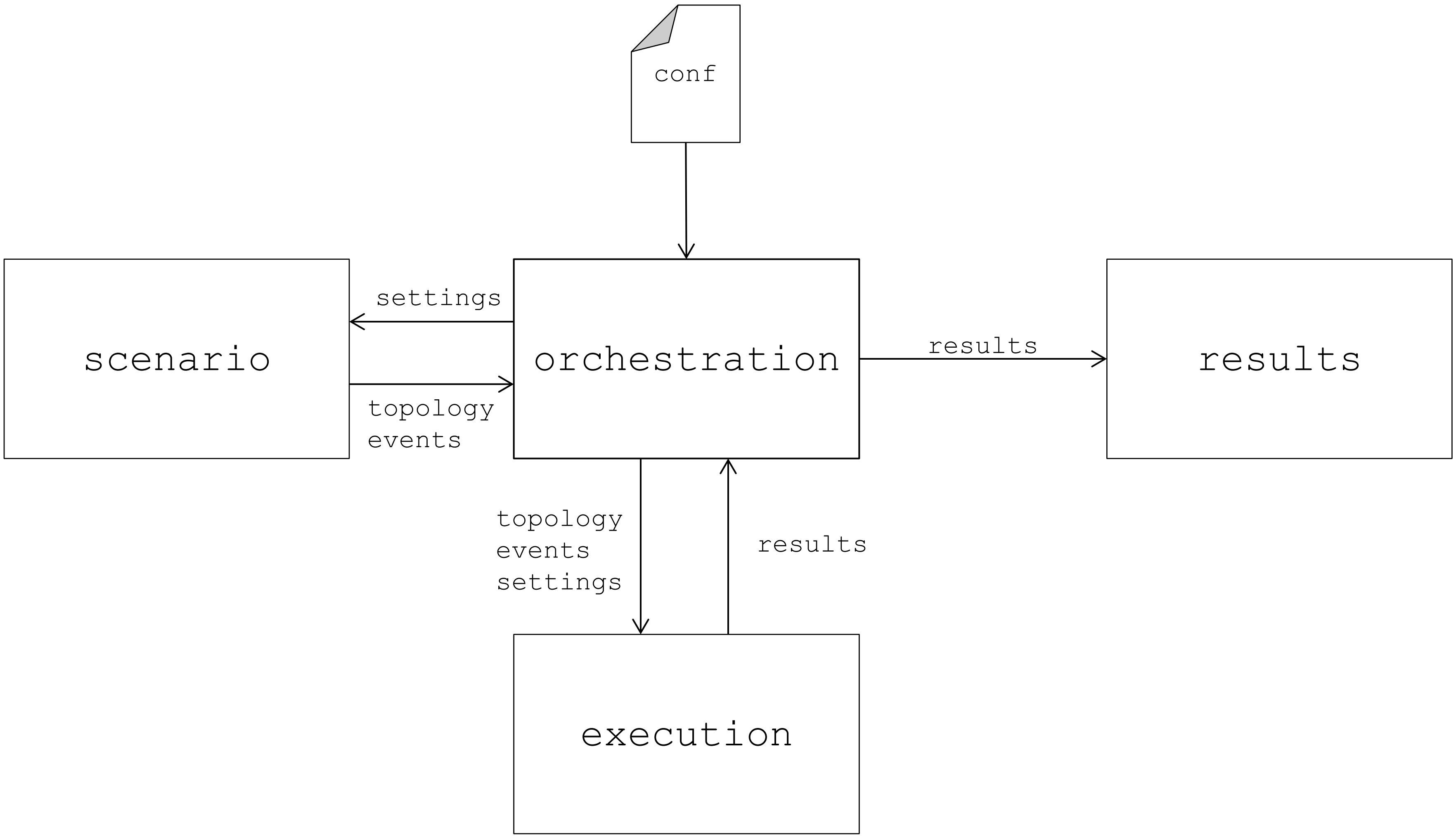
Use cases:

- Caching and routing strategies
- Cache replacement policies
- Cache placement algorithms
- Analytical models

Non-functional requirements:

- Extensibility
- Scalability

High-level architecture



Extensibility

- Python-based, built based on `fnss` and `networkx`
- Plug-in registration system and extensive use of bridge pattern to provide loose-coupling

```
@register_cache_policy('FOO')
class FooCache(Cache)

    def get(self, k):
        ...

    def put(self, k):
        ...
```

```
# config
    .
    .
POLICIES = [‘LRU‘, 'FOO‘]
    .
    .
    .
```

Pluggable components

- Caching and routing strategies
- Cache replacement policies
- Topologies
- Workloads (synthetic and trace-driven)
- Cache placement strategies
- Content placement strategies
- Performance metrics
- Results readers/writers

Caching and routing strategies

Currently implemented strategies:

- Leave Copy Everywhere (LCE)
- Leave Copy Down (LCD)
- ProbCache
- Cache Less for More (centrality-based caching)
- Hash-routing
- Random (choice and Bernoulli)
- Nearest Replica Routing (NRR)
- No Cache

Cache replacement policies

Replacement policies:

- Least Recently Used (LRU)
- Segmented LRU (SLRU)
- Least Frequently Used (LFU)
- First In First Out (FIFO)
- Random

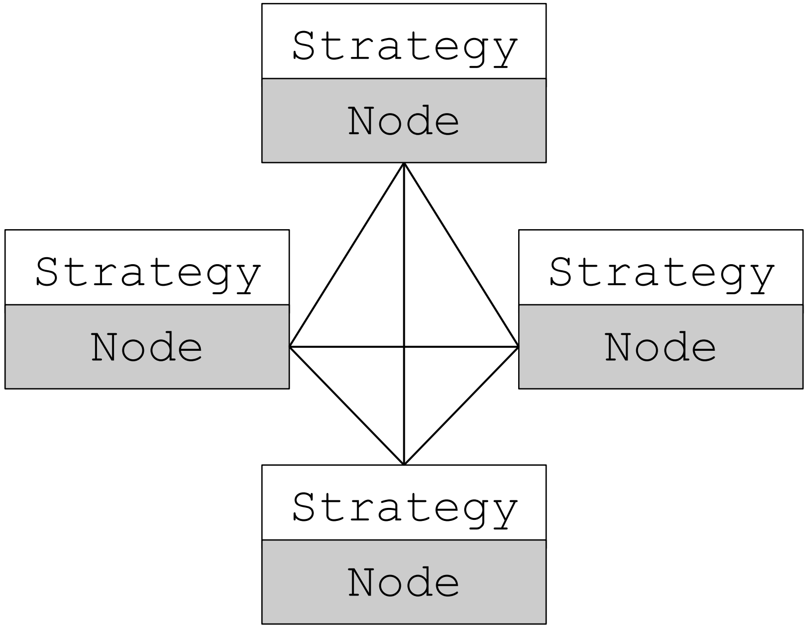
Add-ons:

- Probabilistic insertion
- TTL expiration

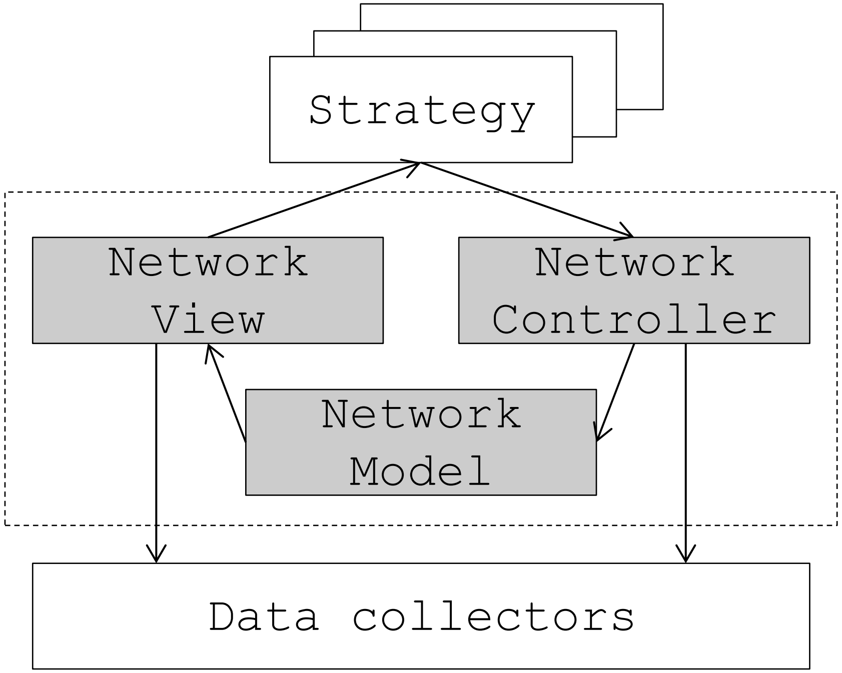
Logically centralized strategy implementation

- Strategies implemented as logically centralized entities
- Network implemented using Model-View-Controller (MVC)

Common agent-based designs



Icarus design



Scalability

- Flow-level abstraction
- No buffering
- Parallel execution of experiments
- Minimized I/O operations

Modelling tools

Cache performance

- Che's approximation
- Laoutaris' approximation
- Numerical hit ratio

Workloads

- Zipf fit
- Trace parsers
  - **Wikibench**
  - **YouTube**
  - **Squid**
  - **URL list**
  - **GlobeTraff**

Performance evaluation

Scalability

Extensibility

Accuracy

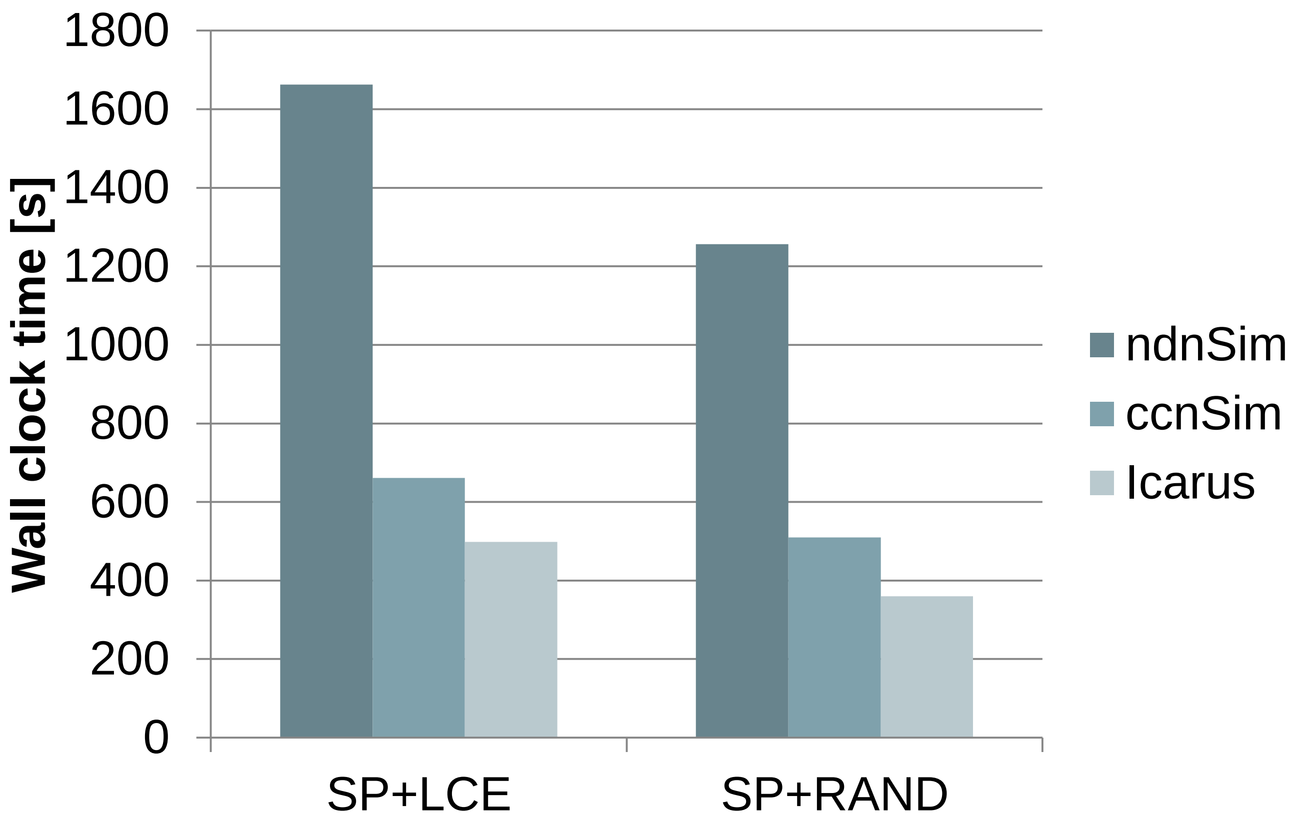
Simulators cross-comparison

Source:

M. Tortelli, D. Rossi, G. Boggia and L. Grieco,
“CCN Simulators: Analysis and Cross-Comparison”
ACM ICN’14, demo session

Simulator	Flow/packet	Buffers
Icarus	**Flow**	**No**
ccnSim	Packet	No
ndnSim	Packet	Yes

CPU utilization



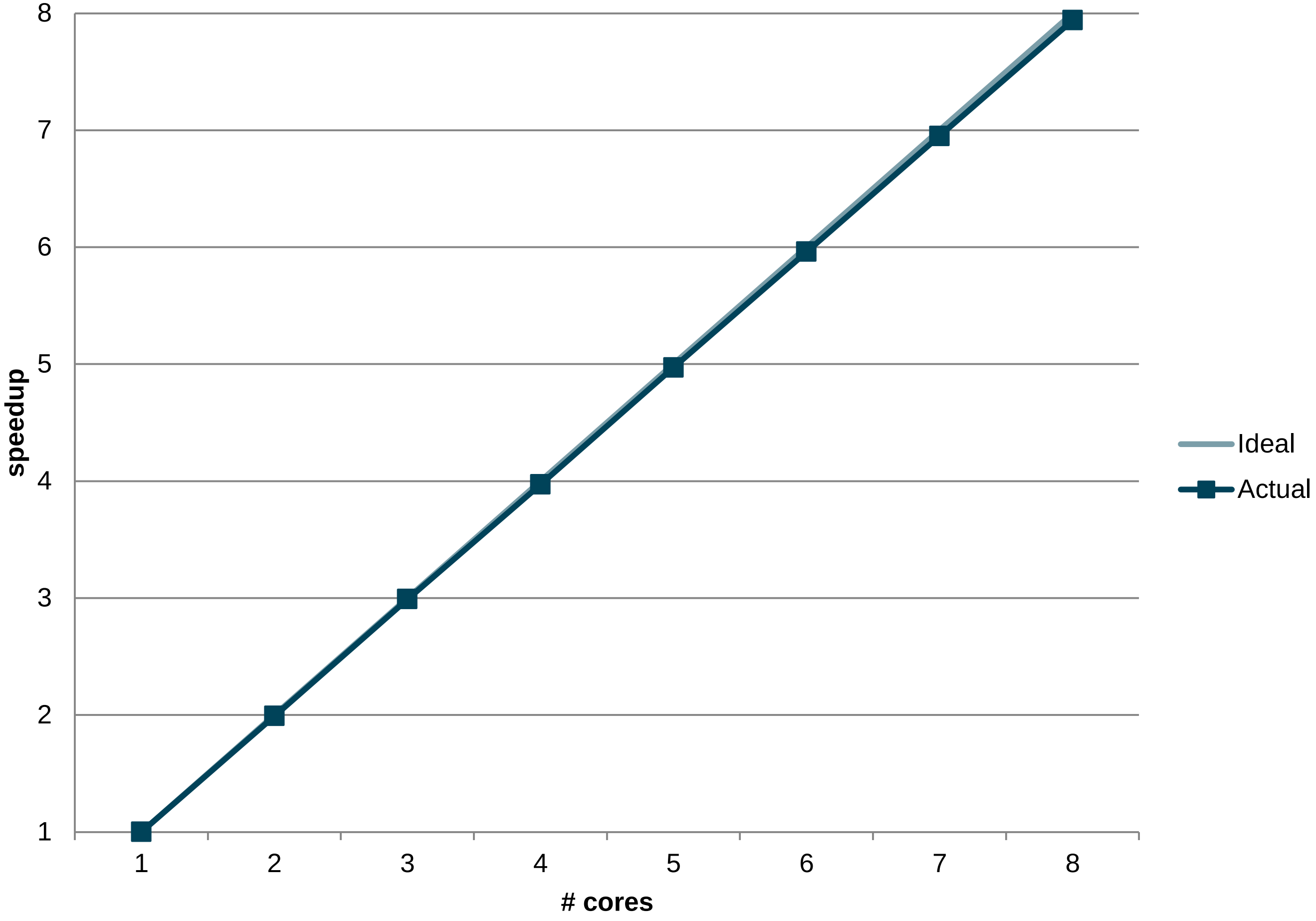
Source: Tortelli *et al.*, ICN’14

Memory utilization

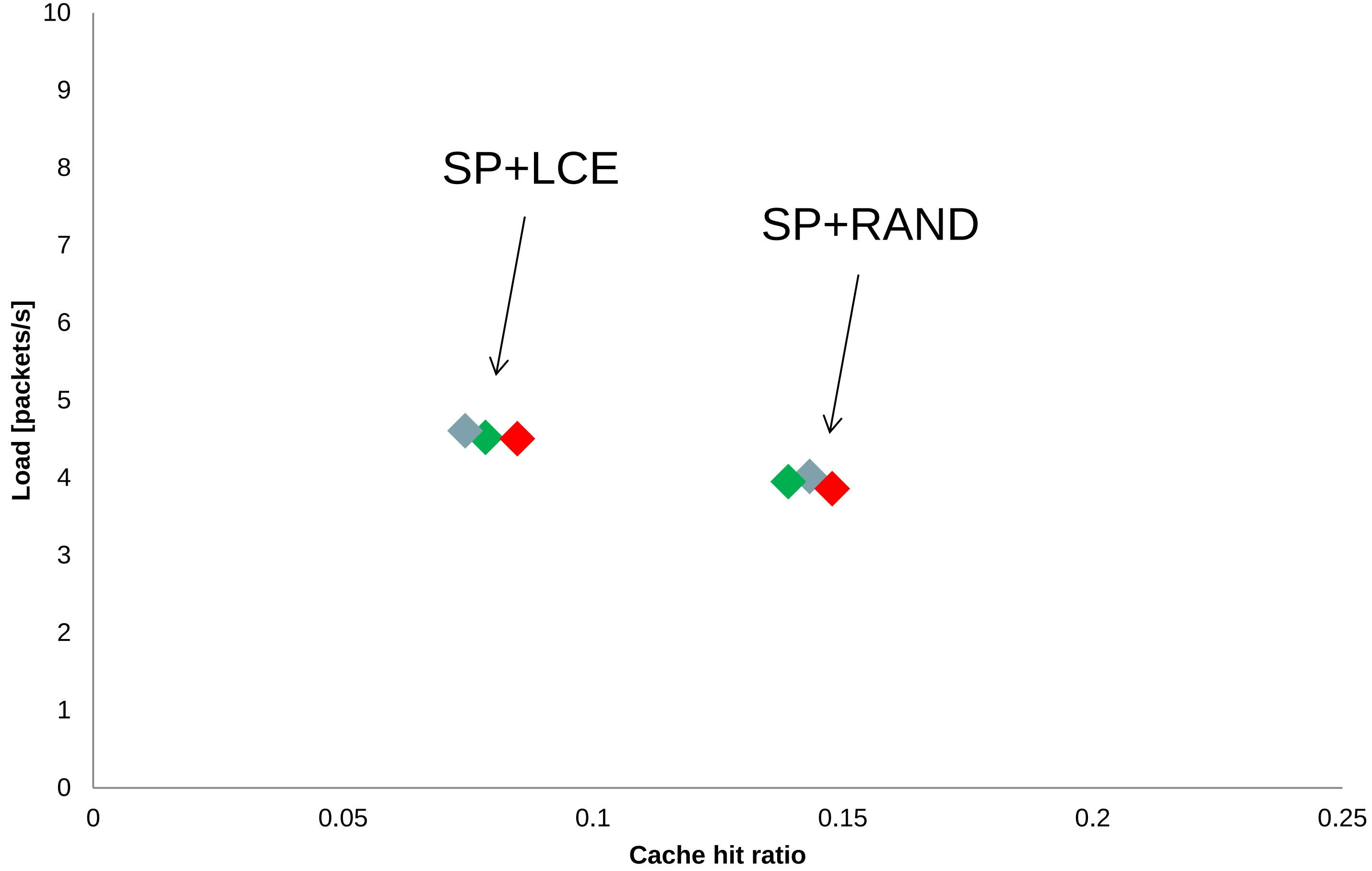
Simulator	SP+LCE	SP+RAND
ndnSim	9.82 GB	7.82 GB
ccnSim	53.68 MB	53.7 MB
Icarus	**111.05 MB**	**110.98 MB**

Source: Tortelli *et al.*, ICN’14

Parallel execution speedup



Accuracy



Source: Tortelli *et al.*, ICN’14

Extensibility

Implementing planned features

Strategy	LOC
Edge	23
LCE	17
LCD	20
ProbCache	32
Centrality-based	30
NRR	24

Extensibility

Implementing unplanned features

Feature	LOC
User-specified seed	3
User-defined experiment queue	7
Centrality-based cache placement	4
Results collector for debugging	20
Save results in CSV format	35

Summary and conclusions

- We presented Icarus, a caching simulator for ICN
- Designed for extensibility and scalability
- Independent cross-comparison validates soundness of design decisions
- Comprises a set of modelling tools for cache performance and workloads analysis

http://icarus-sim.github.io

Icarus tutorial

Lorenzo Saino

Department of Electronic and Electrical Engineering

University College London

`http://icarus-sim.github.io`

Agenda

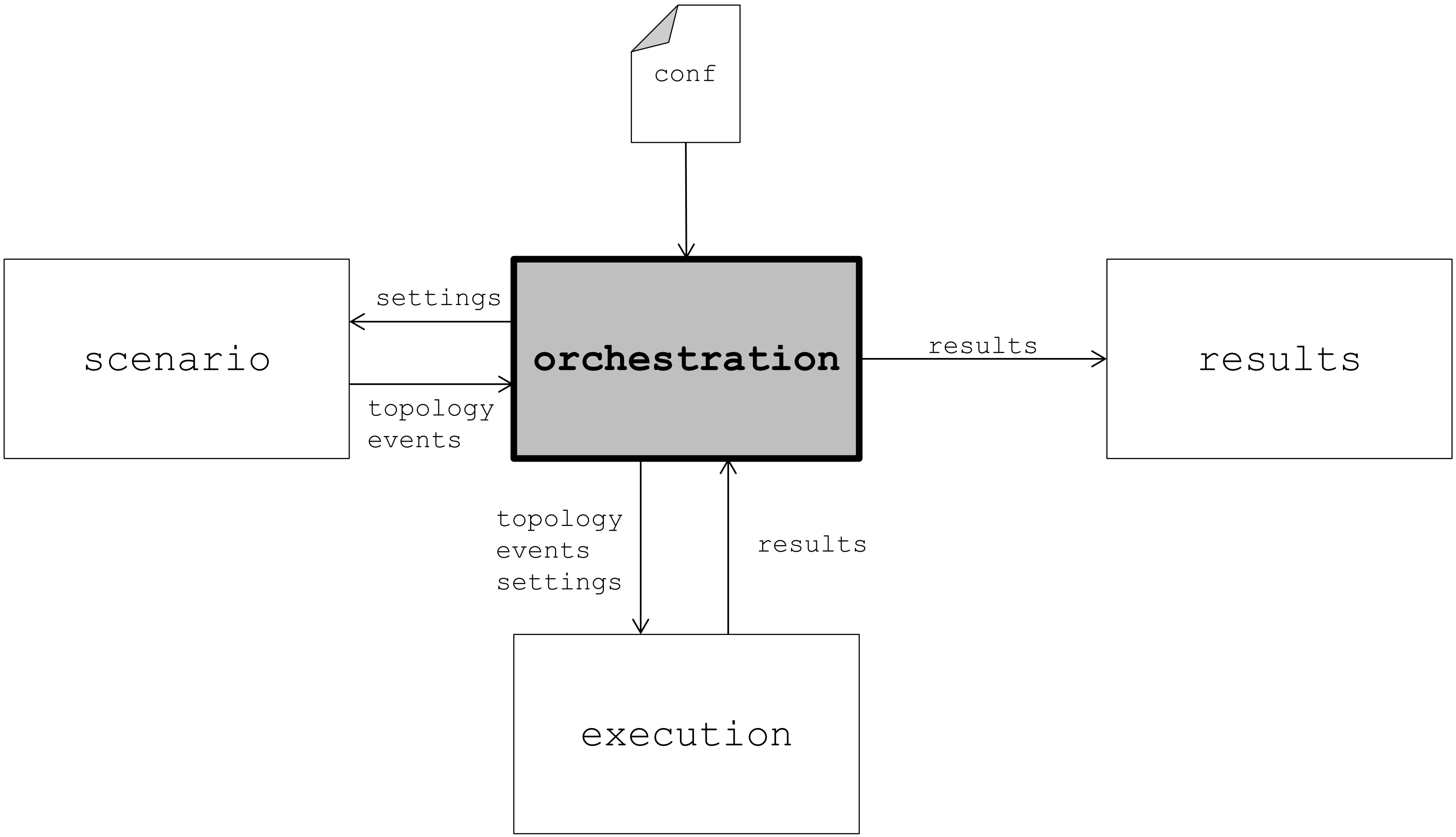
- Architecture overview
- How to download, install, use
- Code walk-through
- Implement new components
- Configuring a simulation campaign
- Using modelling tools
- Q/A

Architecture and design

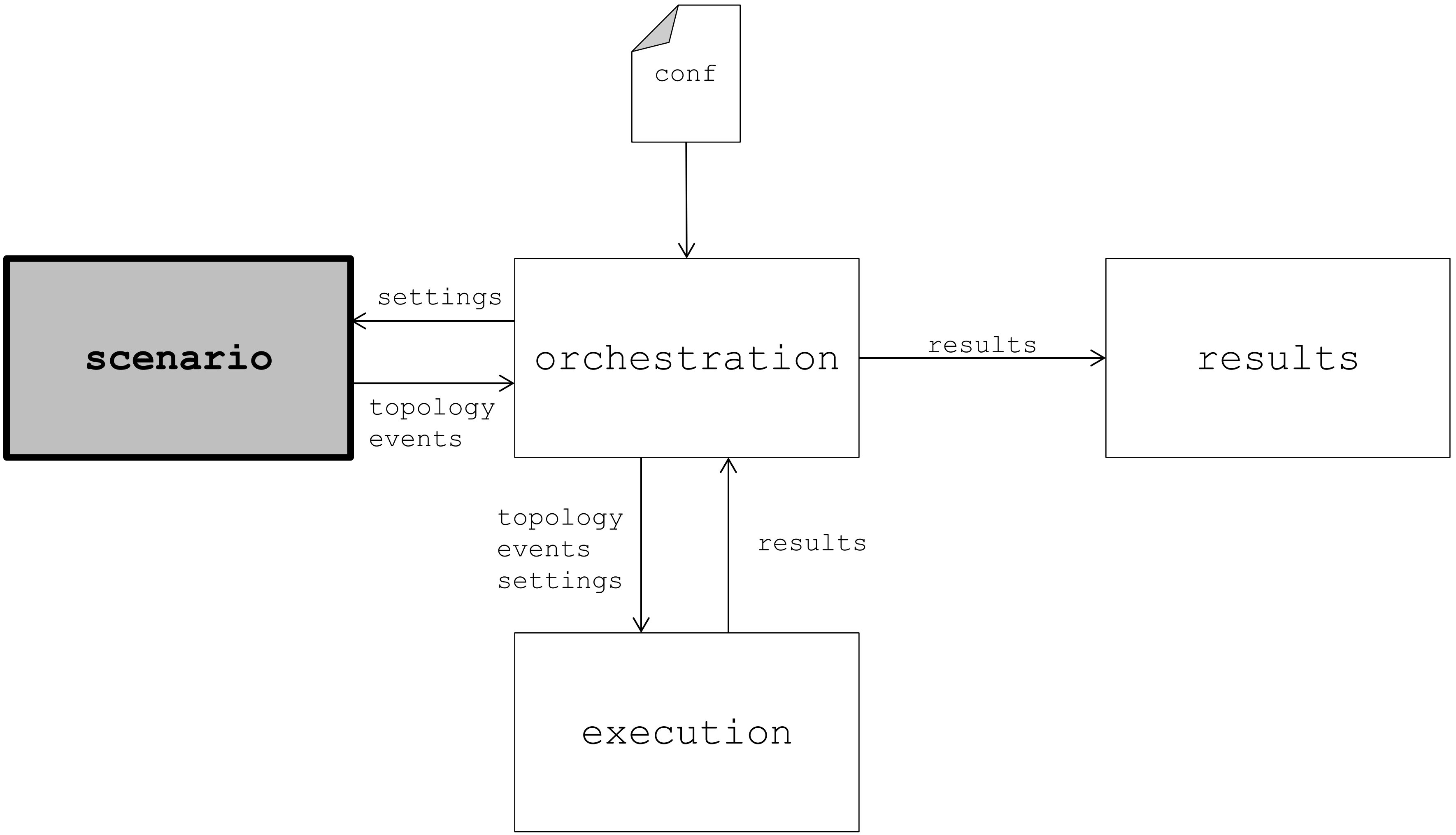
Code organized in four loosely-coupled subsystems:

- Orchestration
- Scenario generation
- Execution
- Results collection and analysis

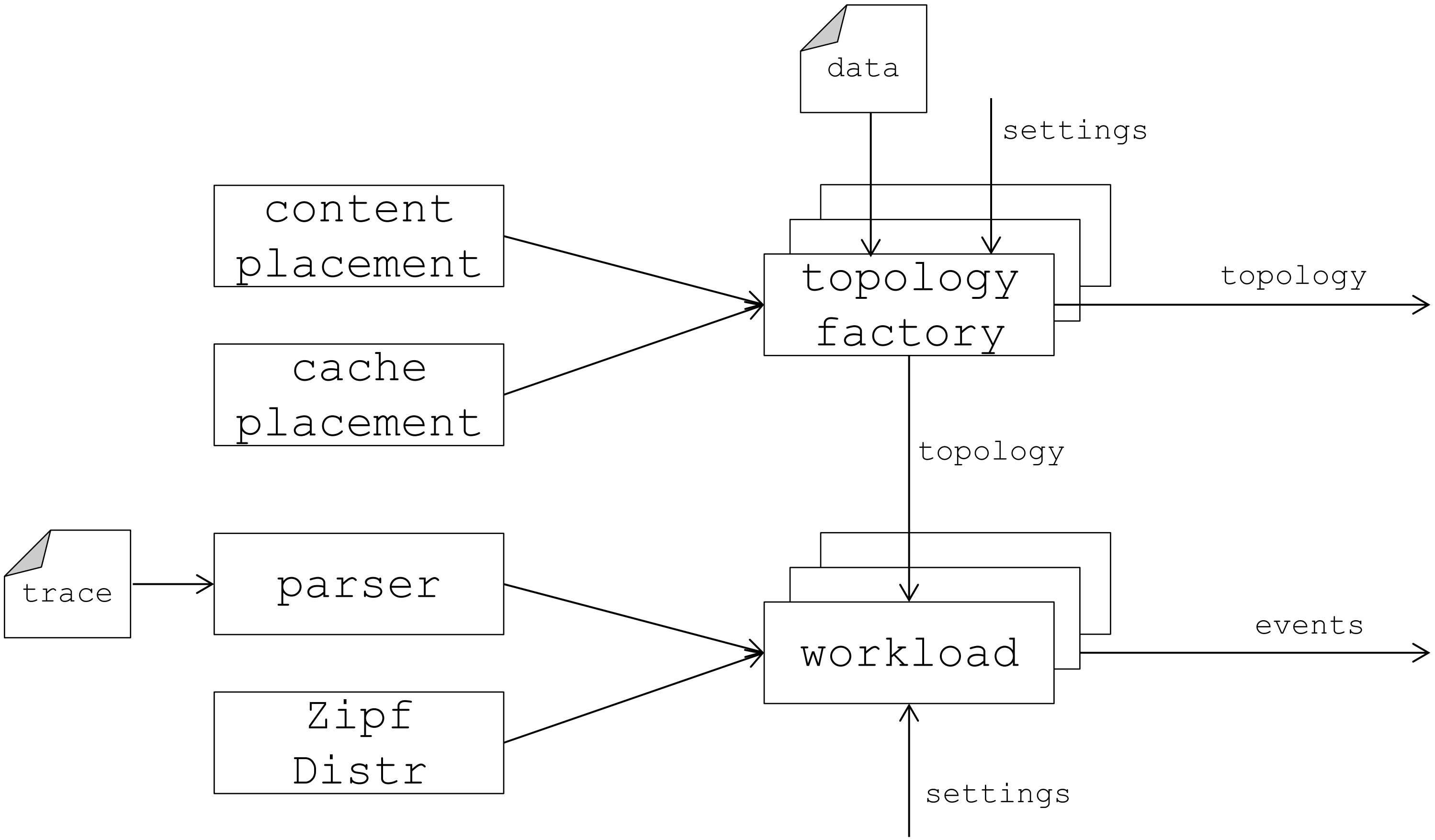
Orchestration



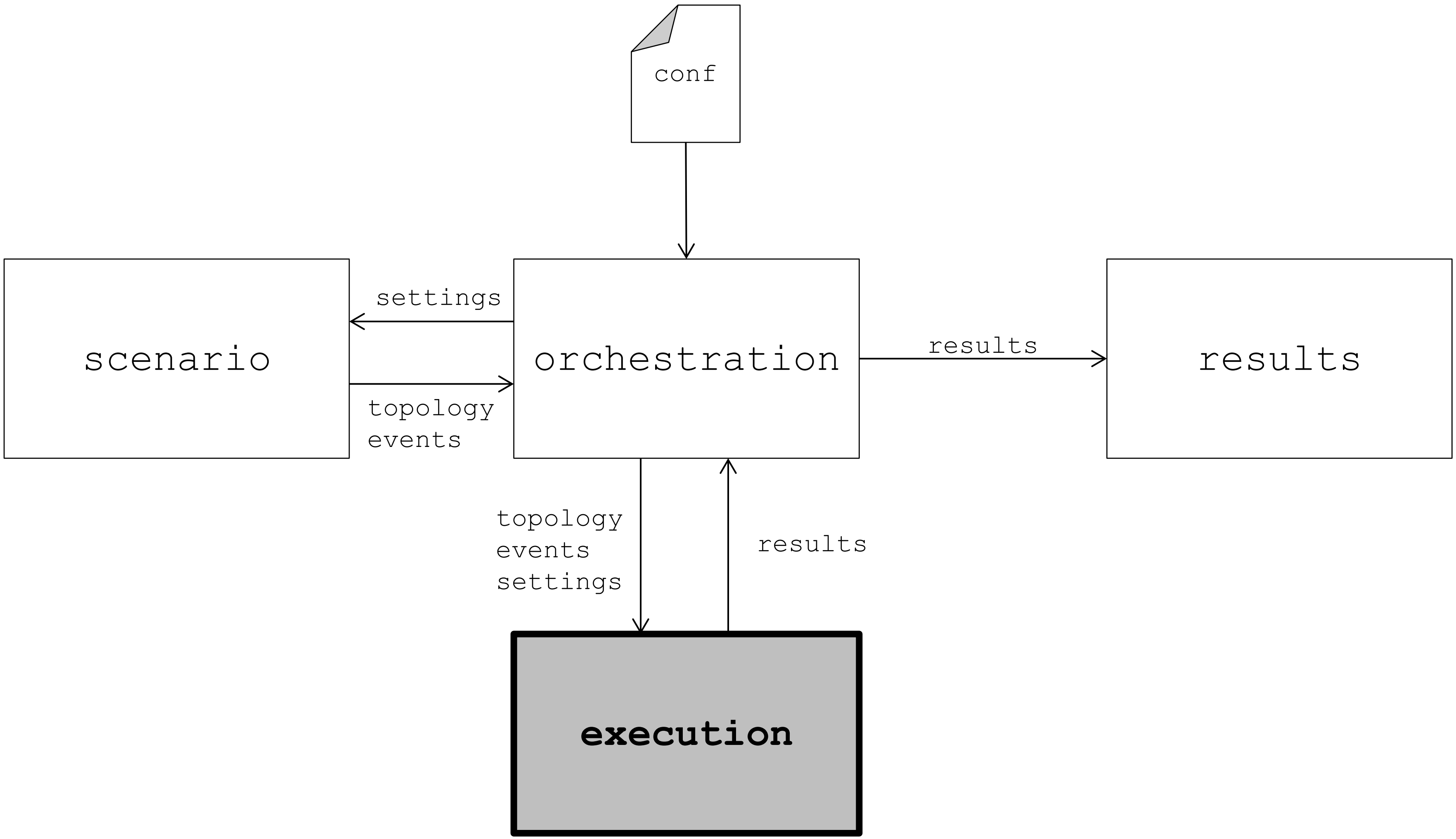
Scenario generation



Scenario generation

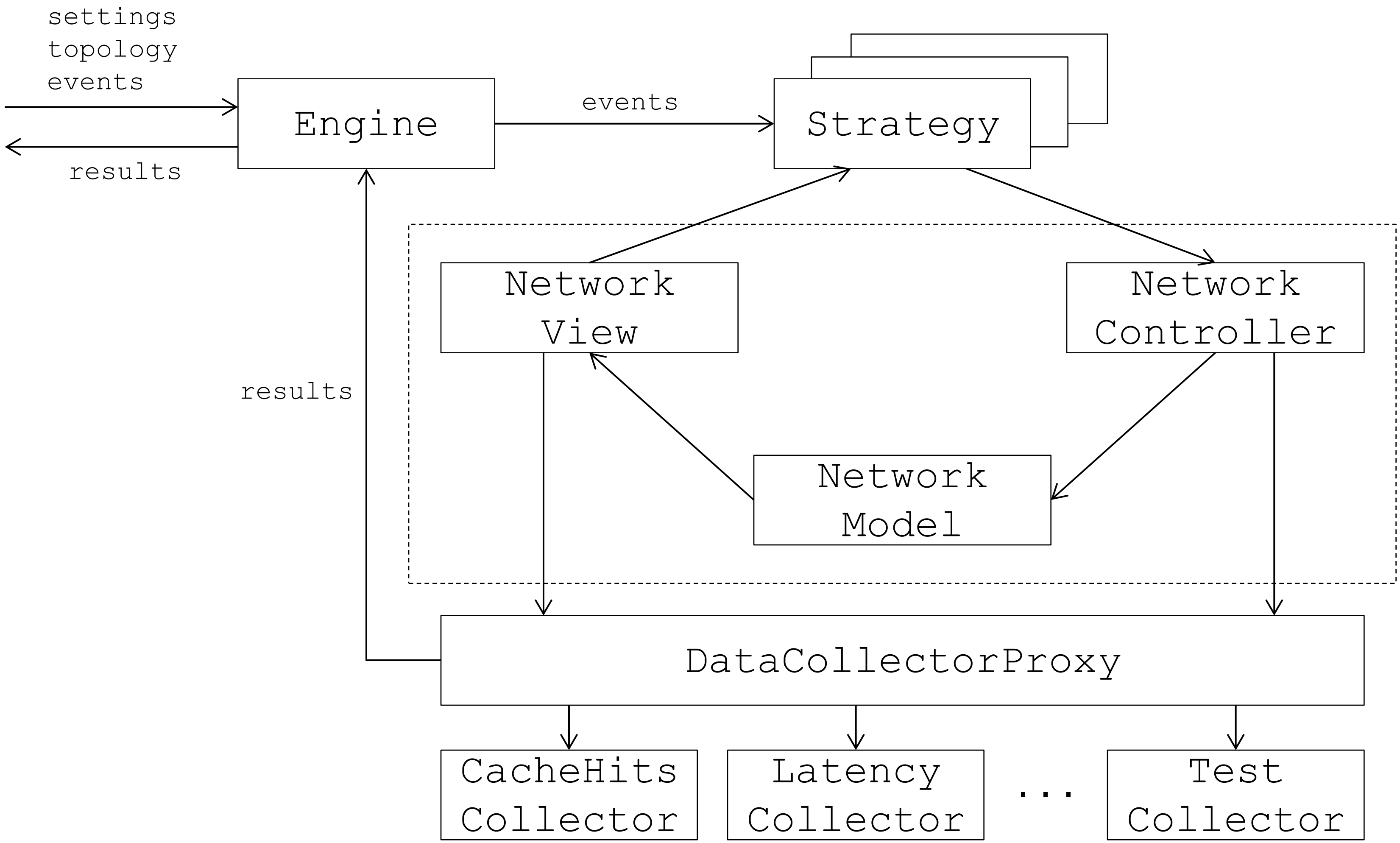


Execution



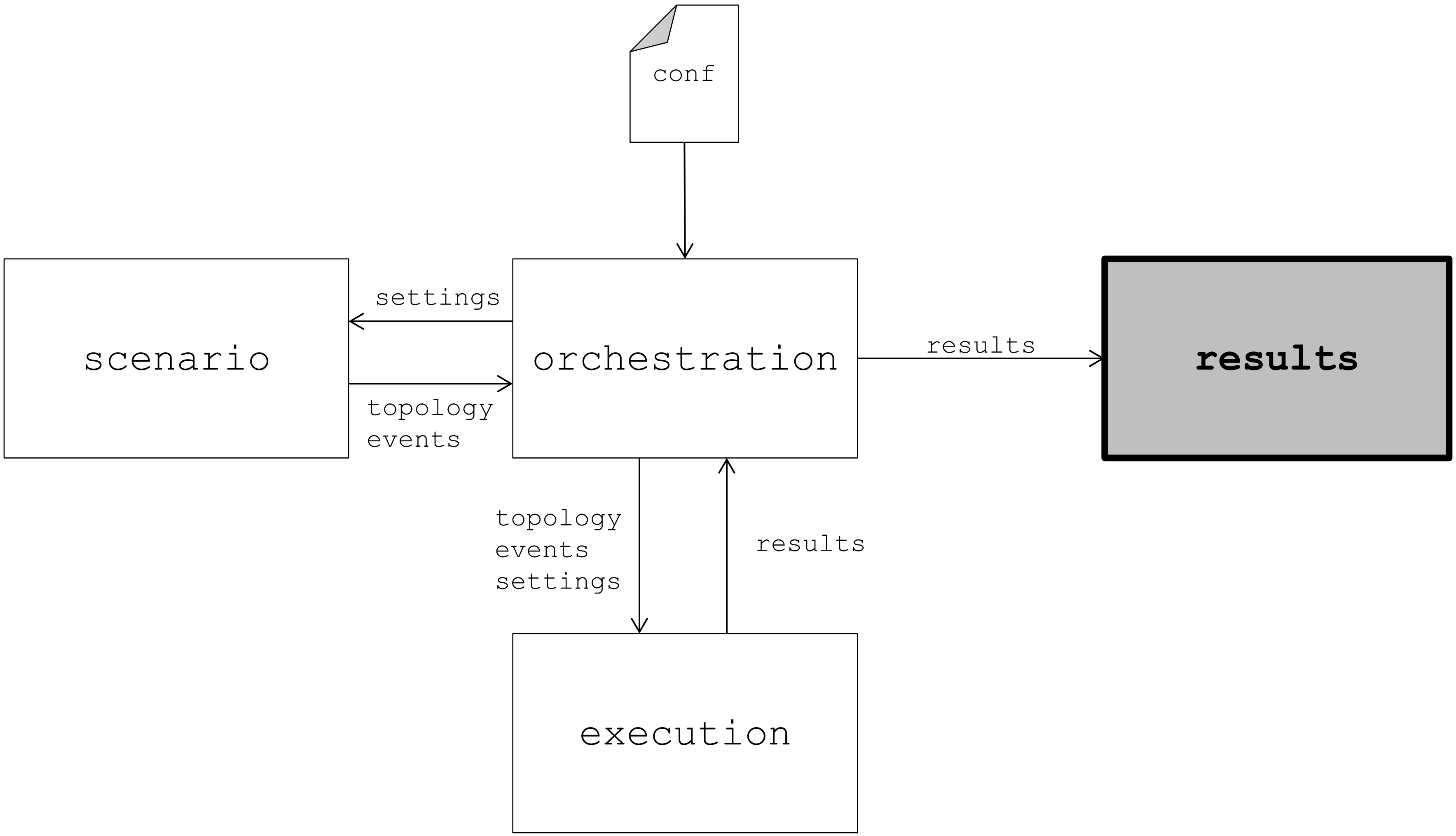
conf
scenario
settings
topology
events
orchestration
results
results
topology
events
settings
results
execution

Execution

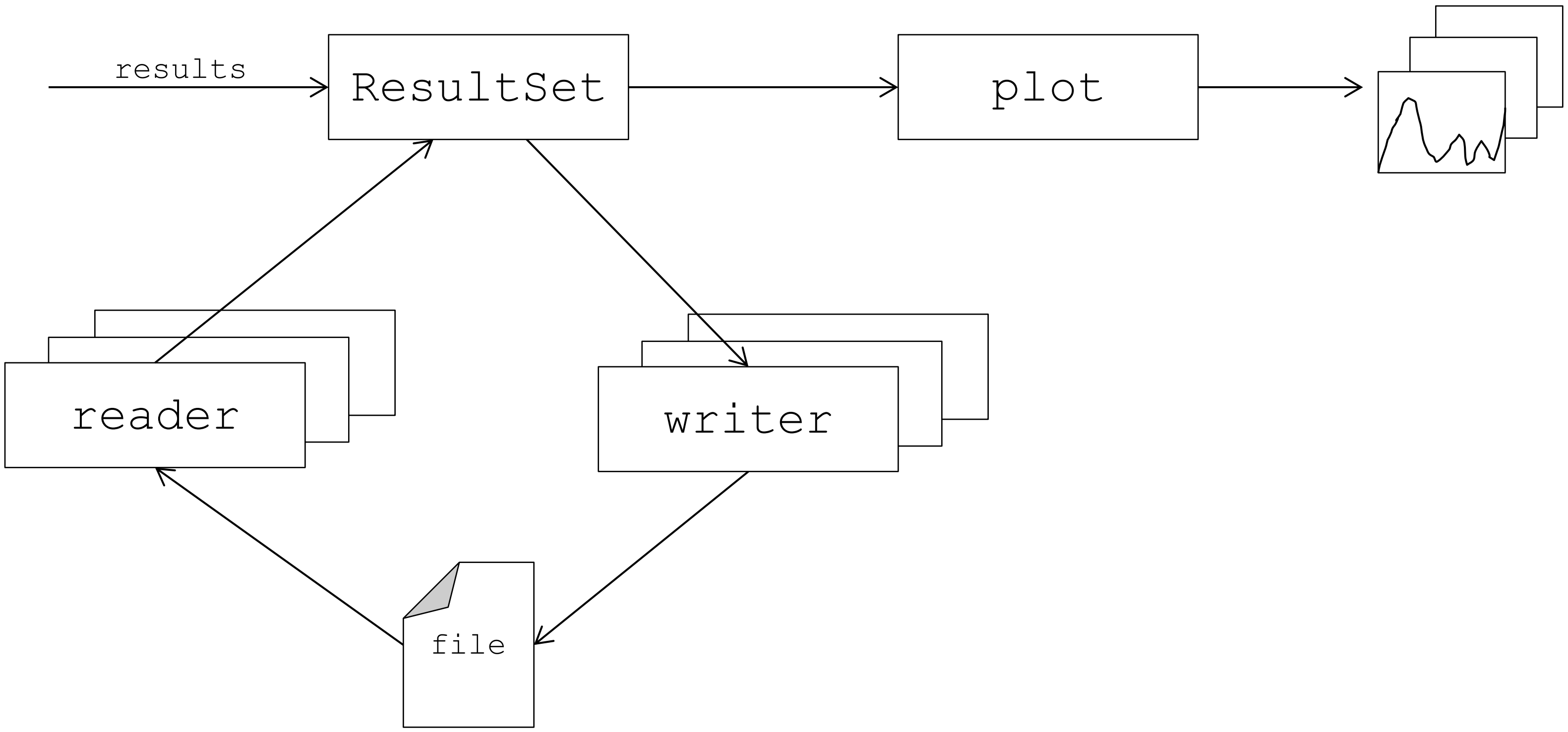


settings
topology
events
results
Engine
events
Strategy
results
Network View
Network Controller
Network Model
DataCollectorProxy
CacheHits Collector
Latency Collector
...
Test Collector

Results collection and analysis



Results collection and analysis



Let's look at the website

`http://icarus-sim.github.io`

Let's look at the code

- Code overview
- Pluggable components
- Network API

Let’s run a sample simulation

LCE vs ProbCache

- Topology: RocketFuel (1221)
- Cache placement: uniform
- Content placement: uniform
- Workload: synthetic, $\alpha = 0.8$
- Replacement policy: LRU
- Metrics: cache hit ratio, latency

Modelling tools

Cache performance

Workloads

Modelling tools

Cache performance

- Che's approximation

Workloads

```
>>> import icarus as ics
>>> ics.che_cache_hit_ratio(
          ics.TruncatedZipfDist(alpha=0.8, n=1000).pdf,
          100)
0.36482948293429832
```

Modelling tools

Cache performance

- Che's approximation
- Laoutaris' approximation

Workloads

```
>>> import icarus as ics
>>> ics.laoutaris_cache_hit_ratio(
          ics.TruncatedZipfDist(alpha=0.8, n=1000).pdf,
          100)
0.35934820935925
```

Modelling tools

Cache performance

- Che’s approximation
- Laoutaris’ approximation
- Optimal hit ratio

Workloads

```
>>> import icarus as ics
>>> ics.optimal_cache_hit_ratio(
          ics.TruncatedZipfDist(alpha=0.8, n=1000).pdf,
          100)
0.52582651157679017
```

Modelling tools

Cache performance

- Che's approximation
- Laoutaris' approximation
- Optimal hit ratio
- Numeric hit ratio

Workloads

```
>>> import icarus as ics
>>> ics.numeric_cache_hit_ratio(
          ics.TruncatedZipfDist(alpha=0.8, n=1000).pdf,
          ics.LruCache(100))
0.37861264056574684
```

Modelling tools

Cache performance

- Che’s approximation
- Laoutaris’ approximation
- Optimal hit ratio
- Numerical hit ratio

Workloads

- Zipf fit

```
>>> import icarus as ics
>>> ics.zipf_fit(ics.TruncatedZipfDist(alpha=0.8, n=1000).pdf)
(0.799999999571759, 1.0)
```

Modelling tools

Cache performance

- Che’s approximation
- Laoutaris’ approximation
- Optimal hit ratio
- Numerical hit ratio

Workloads

- Zipf fit
- Trace parsers

```
>>> import icarus as ics
>>> ics.parse_wikibench('wikibench.txt')
```

http://icarus-sim.github.io