the experimentation process (design of experiments).

system / solution
- algo
- prototype
- optimization
- new mechanism
- model ...

input / config (influences system / solⁿ) → system / solution → **outputs measured** (behaviour of system)

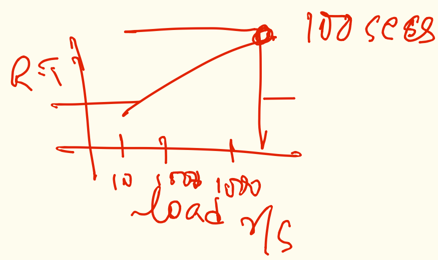
config ↑ (into system / solution)

Input / config:
- load
  - request rate
  - request mix
- CPU alloc. (resource)
- cache size
- b.map. balloon size

Outputs measured:
- utilization
- cache hits / misses
- tput
- latency
- # deadlines missed
- efficiency
- ...

0: meta-point — no question, no experiment.

Q ~ question (why?)



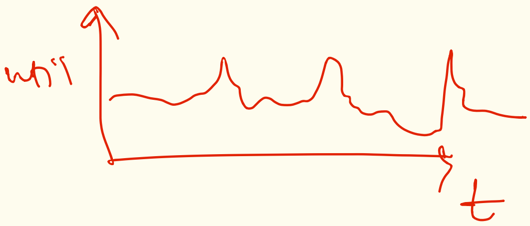
A ~ accurate (correct inference)

R ~ reproducibility (complete specification of setup, input, config)

C ~ completeness (does it cover enough ground to answer the question).

(1) types of questions

- correctness → is a cache-hit always yielding an object in the cache?
- comparative analysis
  - soln. 1 vs soln. 2 vs. soln. 3 . . .
  - beware of comparing w/ a very bad soln.
- cost vs benefit analyis
  ⇒ perf. vs resources / cost
- scalability / saturation behaviour
- failure analysis / performance analysis
- sensitivity analysis
- characterization



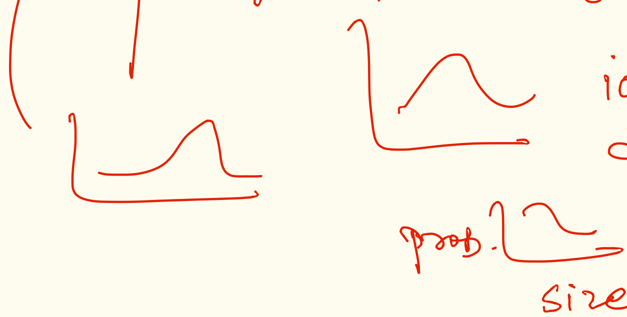
② **setup.** — hardware
software
workload.

(i) **workloads**

- **benchmarks** — custom program that generates sequence/extent of work.
  - requests at rps
  - req: mix — obj size

  synthetic workload

- **trace-driven replay**



iozone
dBench

- **real** load

  production systems facing real requests.

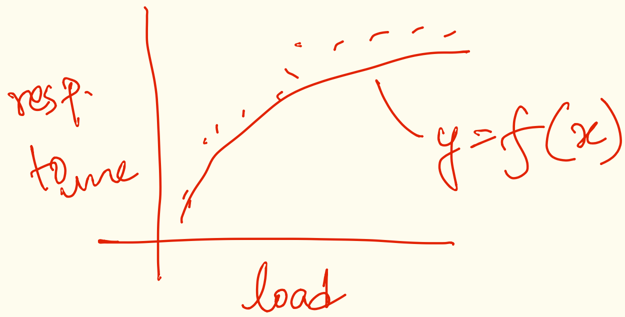
(ii) **modality** (how to execute experiments?)

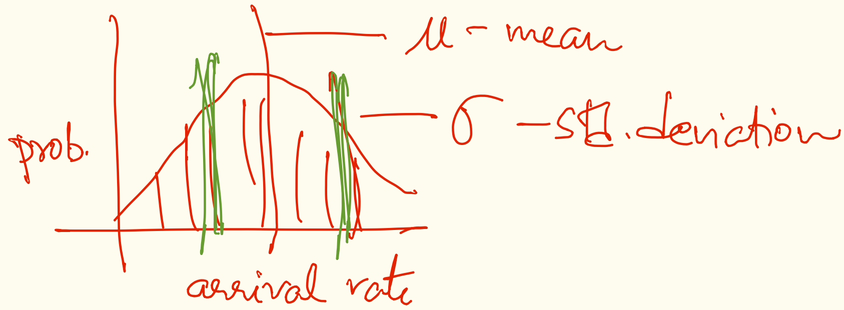
1. analytical models
2. Simulation
3. emulation
4. prototype
5. production system

(*) analytical modeling

1. - build a (mathematical) representation of the system/sol^n. and use it for expected-case behavior, what-if analysis.

- example models: queuing models, $y = f(x)$
  ↑ determine 'f'.
  - inputs for building models are/can be based on real experiments/measurements.





(*) simulations.

2. - "simulate" main/solution components/action items, not all levels of functionality.
   - e.g: state handling — objects (IDs) in cache
   - procedures/sequences — scheduling/caching policy.
   - logic of sol^n./heuristic.

- when to use?
  - quick fdbk/analysis of solution.
  - no access to real setup/hardware
  - large & complicated system.
  - # config. parameters very large.

(*) simulations are approximate!

- do not account for all system interactions.
  eg: lock ordering, non-determinism in scheduling, h/w access ovhds, n/w latencies etc.

- use models
  eg: on cache miss need to simulate disk access latency

- no "actual" msgs or data passed on n/w pkts. passed.
  only "events" updated for these actions.

(*) basic simulator loop.

```
while (1) {
    e = get next event (Q);
    process event (e, Q);
}
```

Q: priority queue (based on time of event)

```
add event (e, Q) {
    t = time of event (e);
    add to queue (e, t, Q);
}.
```

(*): on cache miss event,

add a new event at time t+d

where 'd' is disk access latency.

& e.type is disk read completion.

(*) emulation

3: - closer to real solution.

- mimic behaviour of some components of solution.

① e.g: to emulate network latency on a WAN/Internet, add delay to delivery of each pkt. between endpoints.

~ need model to ~~emulate~~ add delay values!

② emulate behavior of resource/solution.

⇒ virtual disk.

(*) prototype

4: implement all components

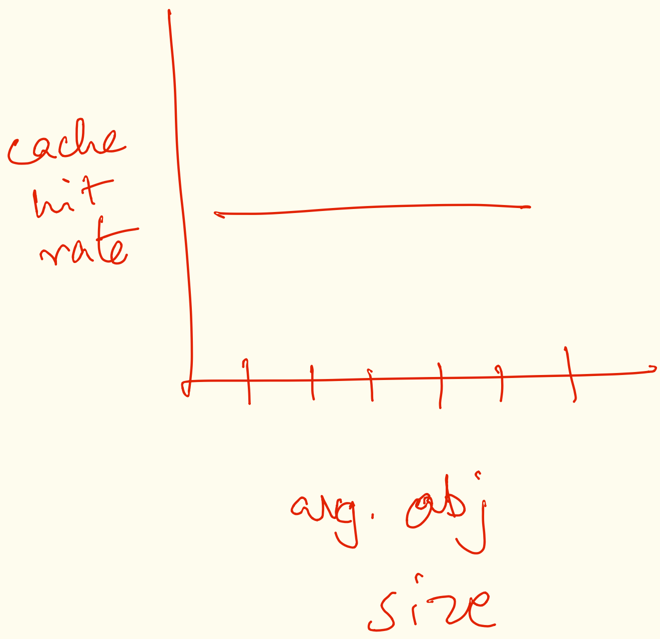
- scale of setup may be different (compared to production system).
- larger turn around times.

(iii) how to experiment?

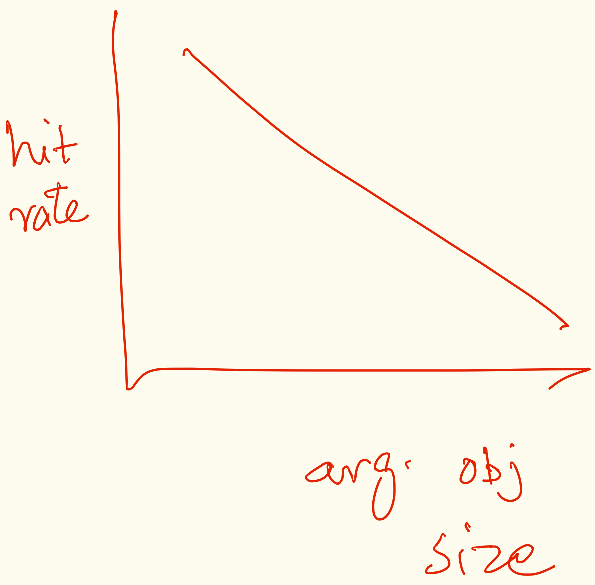
independent vs dependent variables.

varied — measured.

- across experiments all thing to remain same, except change in a single independent variable.



→ cache size proportionate to obj size



- cache size remains same

(iv) instrumentation ovhds.

- does WSS estimation slow the application down?

	w wss est.	w/o wss est.
tput:	100 rps	1000 rps

- quantify ovhds. explicitly.

(4) observation & inferences ⇒ QARC

(5) representation — graphs, tables

—