Are We Serious About Using TLA+ For Statistical Properties?

A. Jesse Jiryu Davis
MongoDB Distributed Systems Research

Should I put a Mastodon handle here, or BlueSky or what?

Just send me a damn email: jesse@mongodb.com

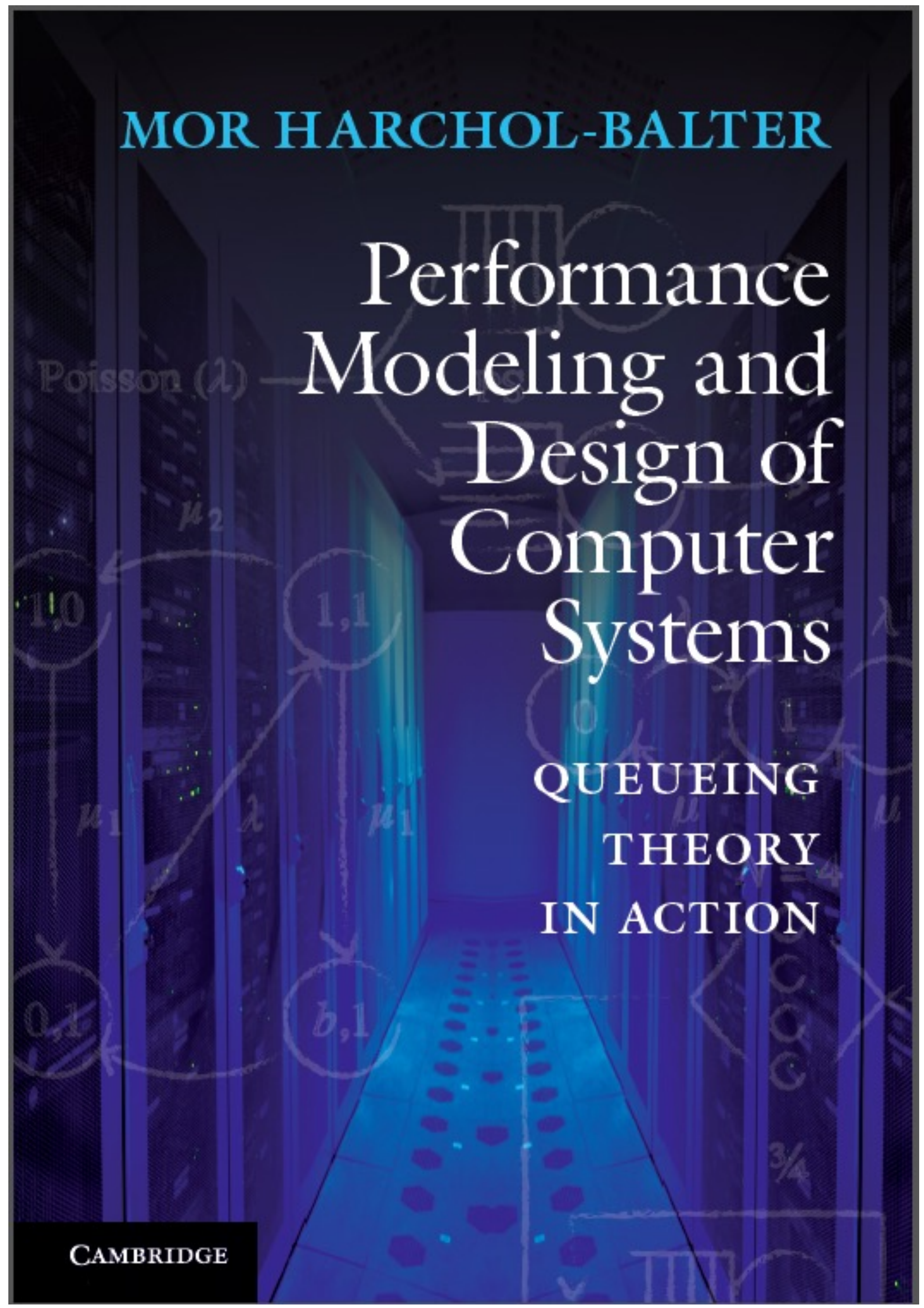
"Formal Methods Only Solve Half My Problems"

Marc Brooker, 2022:

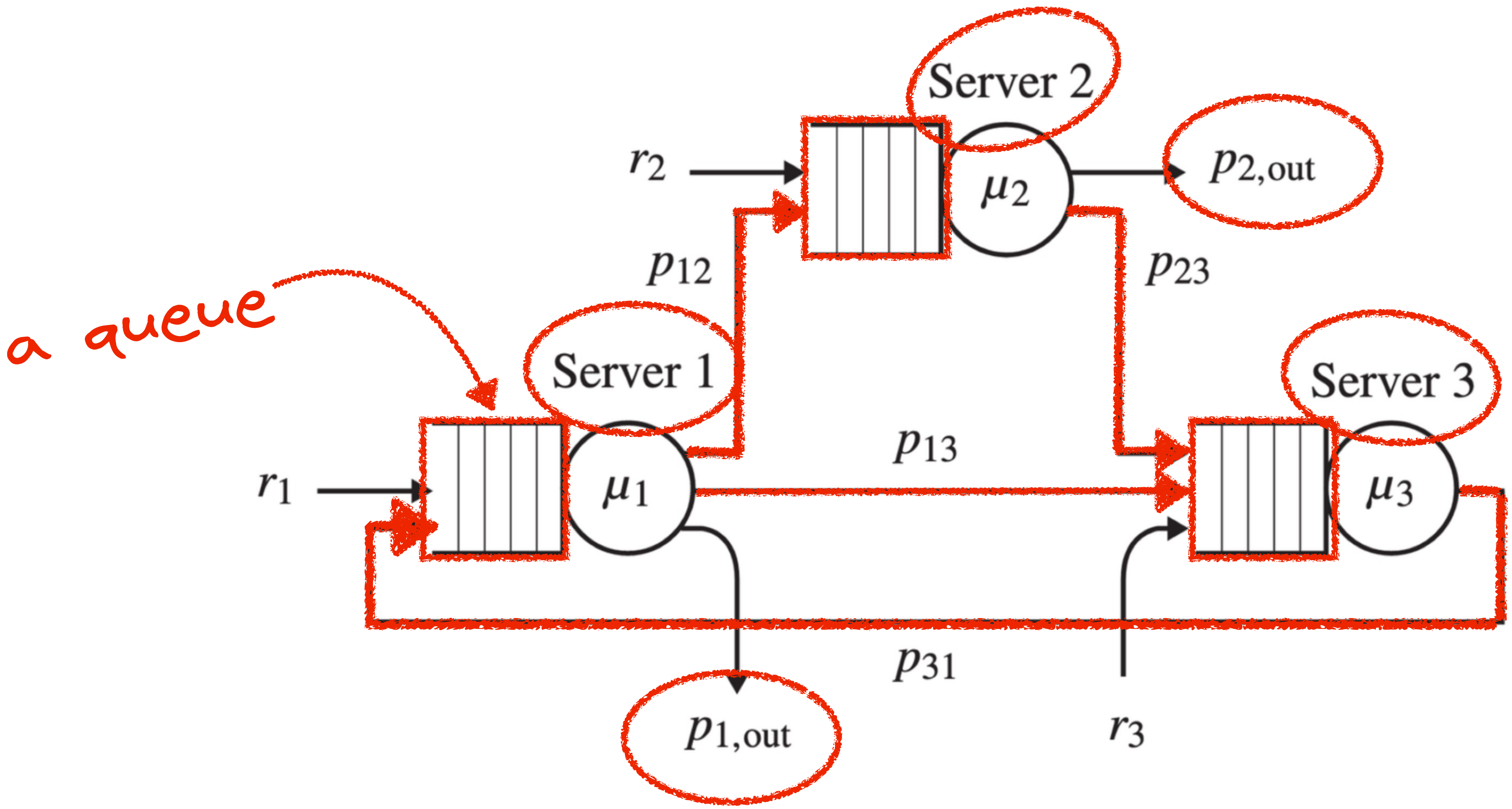
TLA+ can check correctness (safety and liveness), but not performance characteristics.

"What I want is tools that do both: tools that allow development of formal models ... and then allow us to ask those models questions about design performance."

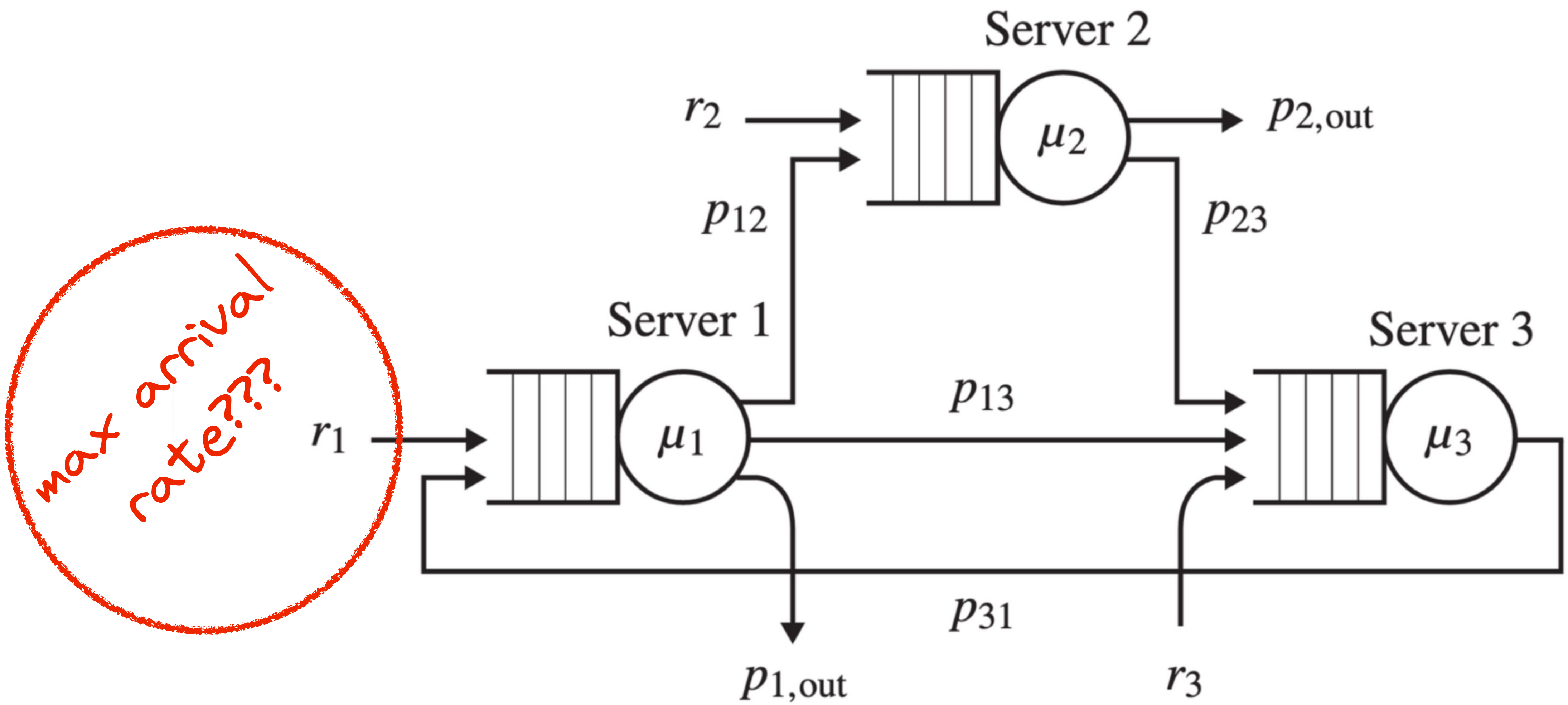
Learn Queueing Theory?



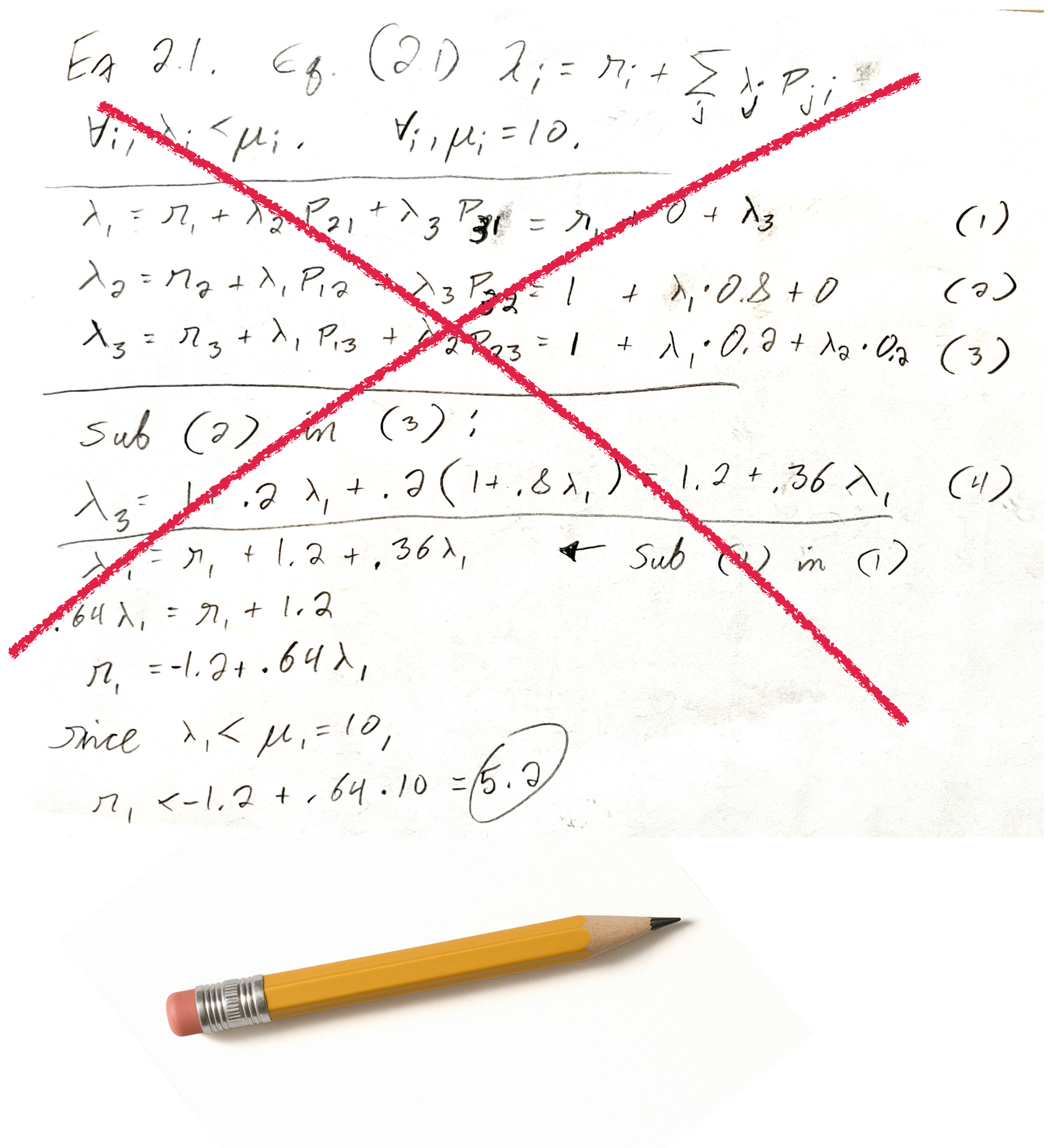
Learn Queueing Theory?



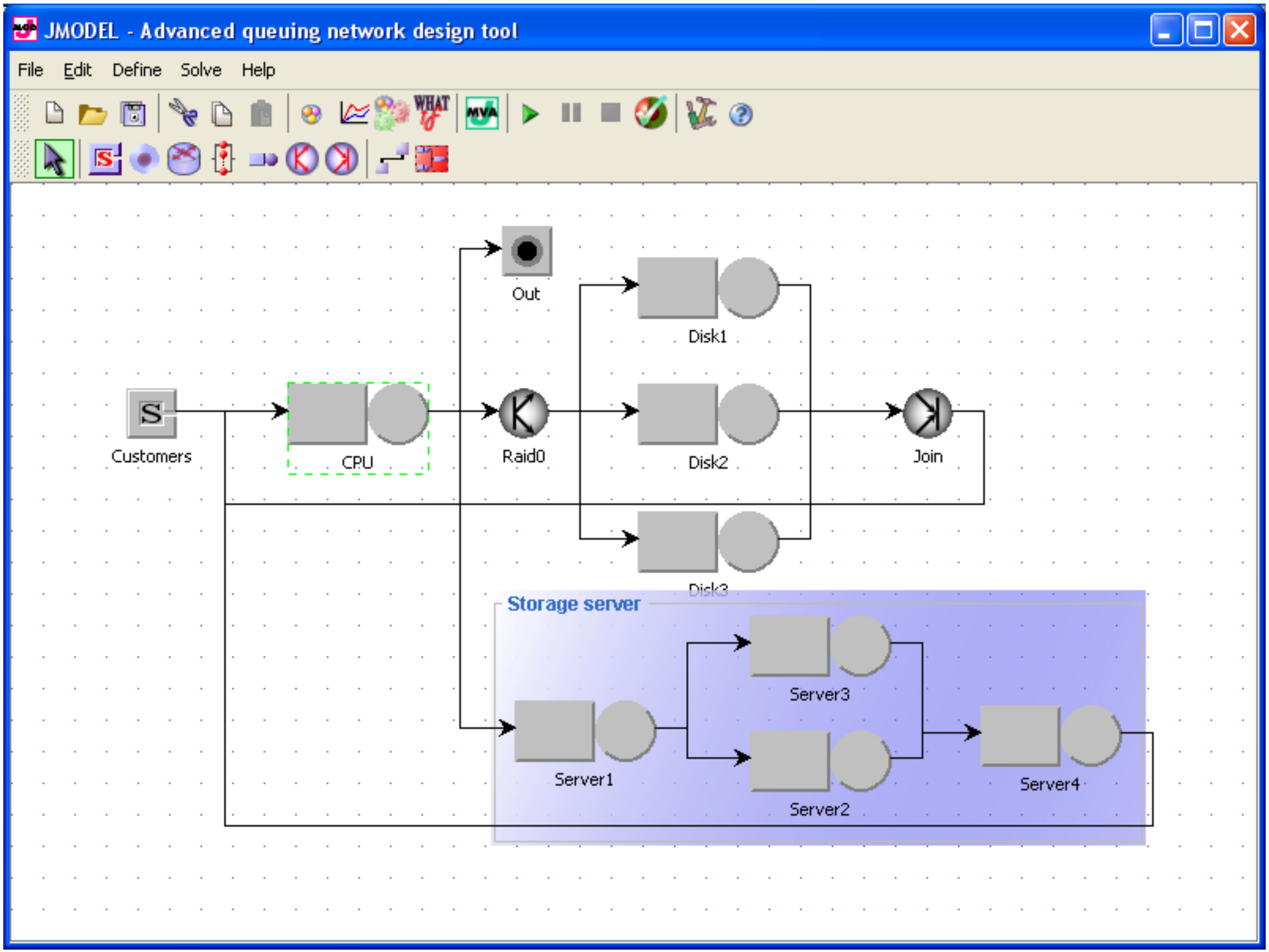
Learn Queueing Theory?



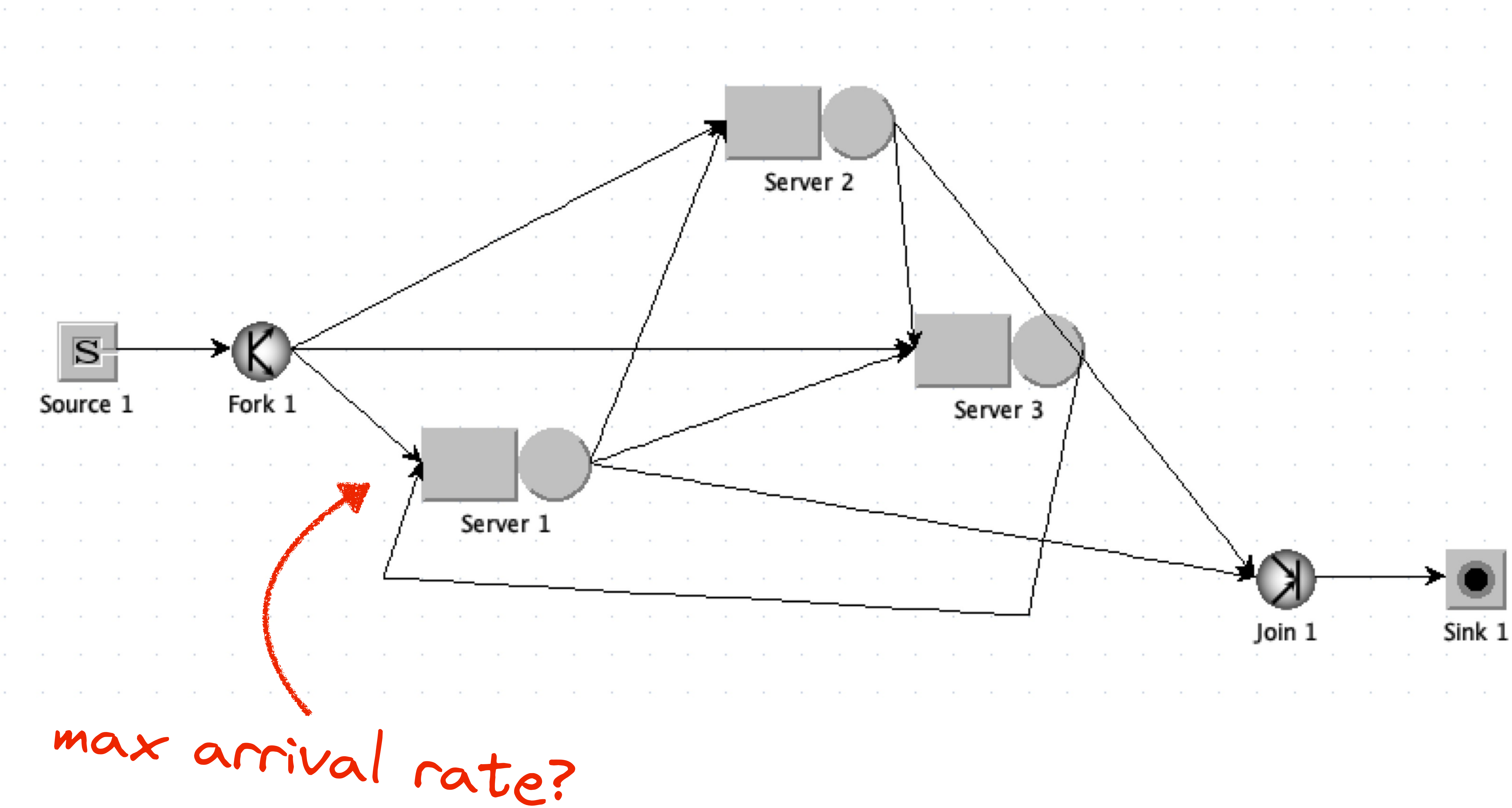
Learn Queueing Theory?



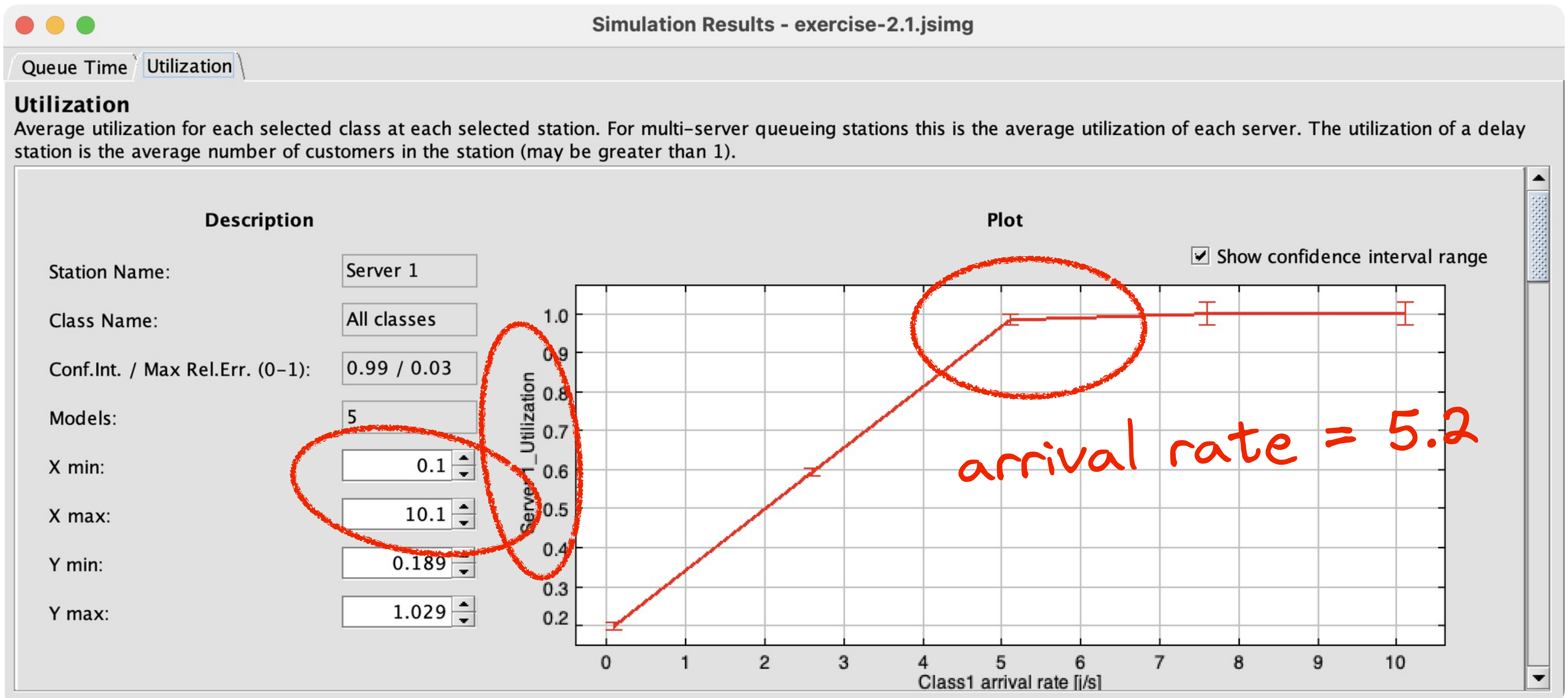
Java Modelling Tools



Java Modelling Tools



Java Modelling Tools



Learn Queueing Theory?

- Queueing theory has super-useful concepts: arrival rate, service rate, utilization, ergodicity, Little's Law, service discipline, open vs. closed loop, and many more.
- Queueing theory math is heinous.
- Don't try to learn the math.
- You can't estimate system performance by solving equations.
- Just run simulations.

Have We Solved All Marc's Problems?

“Formal Methods Only Solve Half My Problems”

“What I want is tools that do both: tools that allow development of formal models … and then allow us to ask those models questions about design performance.”

“Obtaining Statistical Properties via TLC Simulation”

Jack Vanlightly and Markus Kuppe

TLA$^+$ Conference 2022

Updating a statistic

Jack Vanlightly's TLA+ spec of a gossip protocol

```
\* Increment the updates counter by the number of incoming peer states.
TLCSet(updates_ctr_id, TLCGet(updates_ctr_id)
                       + Cardinality(DOMAIN incoming_peer_states))
```

"cost function"

Writing a CSV line

Jack Vanlightly's TLA+ spec of a gossip protocol

```
CSVWrite(
    "%1$s,%2$s,%3$s,%4$s,%5$s,%6$s,%7$s,%8$s,%9$s,%10$s,%11$s,%12$s,%13$s,"
    \o "%14$s,%15$s,%16$s,%17$s,%18$s,%19$s,%20$s,%21$s,%22$s,%23$s,%24$s,%25$s",
    <<behaviour_id,
        r, RoundMessageLoad(r), DirectProbeDeadMessageLoad(r), IndirectProbeDeadMessageLoad(r),
        TLCGet(updates_pr_ctr(r)), TLCGet(eff_updates_pr_ctr(r)), alive_count, suspect_count,
        dead_count, alive_states_count, suspect_states_count, dead_states_count,
        infective_states_count, infectivity, cfg_num_members, cfg_dead_members, cfg_new_members,
        SuspectTimeout, DisseminationLimit, cfg_max_updates, cfg_lose_nth, cfg_peer_group_size,
        cfg_initial_contacts, MaxRound>>,
    RoundStatsCSV)
```

Complaint 1: syntax

Implementing a probability distribution

Jack Vanlightly's TLA+ spec of a gossip protocol

```
\*  'probabilistic' is a random chance of losing the message
\*  'exhaustive' is for model checking where both options are explored
GetDeliveredCount() ==
  CASE MessageLossMode = "probabilistic" ->
      IF RandomElement(1..cfg_lose_nth) = cfg_lose_nth THEN {0} ELSE {1}
    [] MessageLossMode = "exhaustive" -> {0,1}

SendMessage(msg) ==
  \E delivered_count \in GetDeliveredCount() :
    \* ... send the message if delivered_count is 1 ...
```

Complaint 2: randomization is incompatible with model-checking*

*correction: Markus says this is fixed

Complaint 3: randomization is very limited

Complaint 3: randomization is very limited

```
\* In your dreams
TLCSet(cost, TLCGet(cost) + 1)
TLCSet(cost, TLCGet(cost) + 2.5)
TLCSet(cost, TLCGet(cost) + Exponential(3))
```

Complaint 4:

no floats

no probability distributions besides "uniform"

Are We Serious About Statistical Properties?

State of the Art

1. Java Modelling Tools
2. PRISM
3. Runway
4. FizzBee

State of the Art #1 of 4:
Java Modelling Tools

- Comes with an extra L, straight from London, tariff-free.
- Made for statistical modeling and answering performance questions.
- Point-and-click interface — is this a pro or a con? 🤔
- Lots of probability distributions.
- Cost functions.
- Use real-world data sets as inputs!

State of the Art #2 of 4: PRISM

Probabilistic Model Checker

```
[my_action] x=0 -> 0.8:(x'=1) + 0.2:(x'=2);
```

probabilities or rates, for discrete-time or continuous-time models

PRISM
Cost Functions

A “cost” is any measurement of performance. PRISM calls them “rewards”.

```
rewards
    x=0 : 100;
    x>0 & x<10 : 2*x;
endrewards
```

Express good rewards like revenue, or bad costs like latency.

PRISM
Property Expressions

```
P<0.1 [ F<=100 num_errors > 5 ]
```

"the probability that more than 5 errors occur within the first 100 time units is less than 0.1"

```
P=? [ !proc2_terminate U proc1_terminate ]
```

"the probability that process 1 terminates before process 2 does"

PRISM

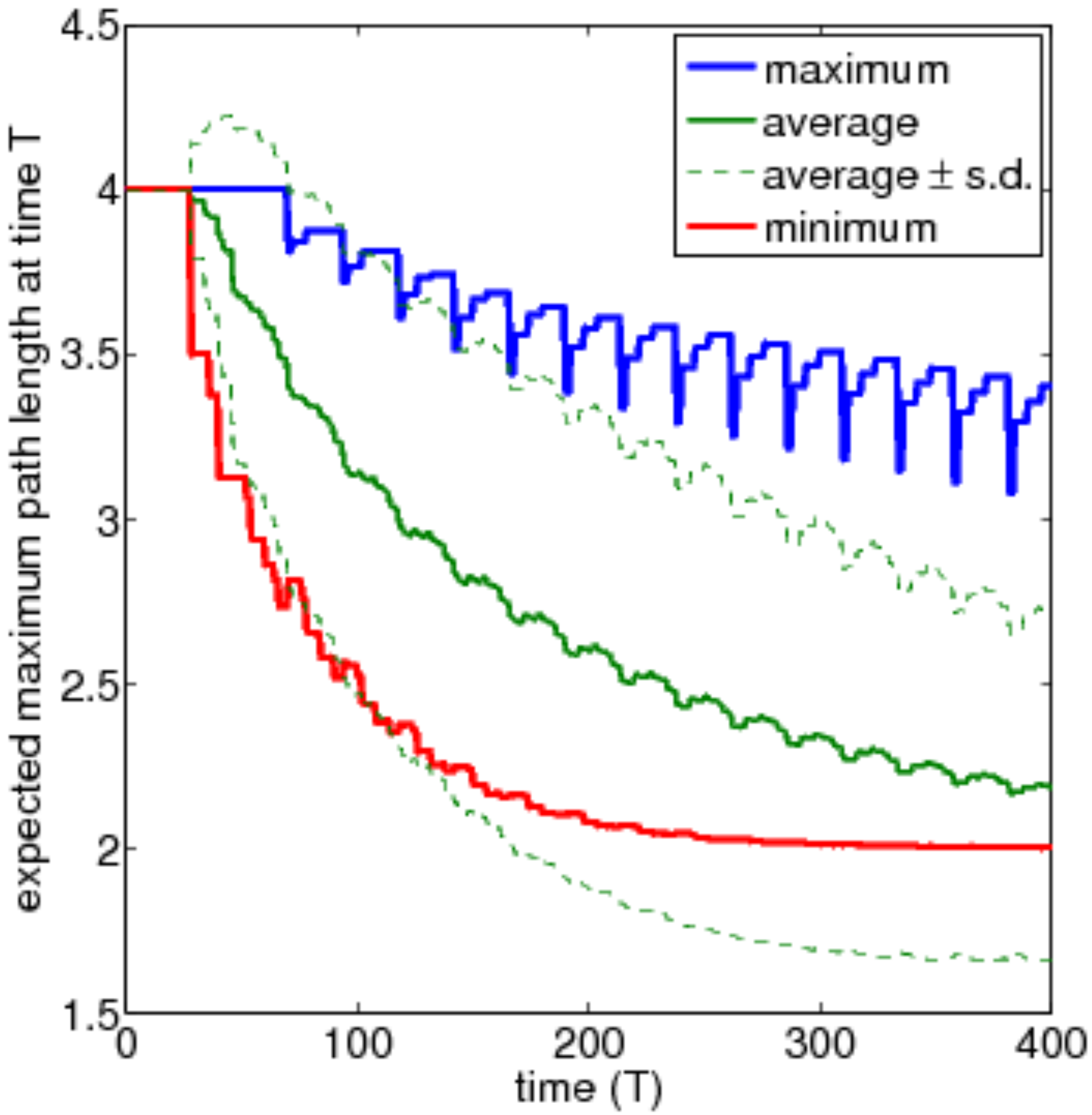
Property Expressions

Safety: long-run probability something bad happens is 0.

Liveness: long-run probability something good happens is 1.

Performance: p95 latency is less than x.

PRISM model of a gossip protocol



PRISM model of a gossip protocol

```
// initial view of node 1 (can see 2 one hop away)
const int iv1_1_a = 2;
const int iv1_2_a = 0;
const int iv1_1_h = 1;
const int iv1_2_h = 4;

// initial view of node 2 (empty)
const int iv2_1_a = 0;
const int iv2_2_a = 0;
const int iv2_1_h = 4;
const int iv2_2_h = 4;

// initial view of node 3 (can see 2 one hop away)
const int iv3_1_a = 2;
const int iv3_2_a = 0;
const int iv3_1_h = 1;
const int iv3_2_h = 4;

// initial view of node 4 (can see 2 one hop away)
const int iv4_1_a = 2;
const int iv4_2_a = 0;
const int iv4_1_h = 1;
const int iv4_2_h = 4;
```

PRISM model of a gossip protocol

Some of Node 1's code:

```
// send to node 2
[push1_2_0] s1=3 & send1=id2 & i1=0 -> (i1'=i1+1);
[push1_2_1] s1=3 & send1=id2 & i1=1 & v1_1_h<4 -> (s1'=0) & (i1'=0) & (send1'=0);
[push1_2_end] s1=3 & send1=id2 & ((i1=1&v1_1_h=4) | (i1=2&v1_2_h=4)) -> (s1'=0) & (i1'=0) & (send1'=0);
// send to node 3
[push1_3_0] s1=3 & send1=id3 & i1=0 -> (i1'=i1+1);
[push1_3_1] s1=3 & send1=id3 & i1=1 & v1_1_h<4 -> (s1'=0) & (i1'=0) & (send1'=0);
[push1_3_end] s1=3 & send1=id3 & ((i1=1&v1_1_h=4) | (i1=2&v1_2_h=4)) -> (s1'=0) & (i1'=0) & (send1'=0);
// send to node 4
[push1_4_0] s1=3 & send1=id4 & i1=0 -> (i1'=i1+1);
[push1_4_1] s1=3 & send1=id4 & i1=1 & v1_1_h<4 -> (s1'=0) & (i1'=0) & (send1'=0);
[push1_4_end] s1=3 & send1=id4 & ((i1=1&v1_1_h=4) | (i1=2&v1_2_h=4)) -> (s1'=0) & (i1'=0) & (send1'=0);
```

PRISM model of a gossip protocol

Some of Node 2's code:

```
push1_2_0=push2_1_0, push1_2_1=push2_1_1, push1_2_2=push2_1_2, push1_2_3=push2_1_3, push1_2_end=push2_1_end,
push1_3_0=push2_3_0, push1_3_1=push2_3_1, push1_3_2=push2_3_2, push1_3_3=push2_3_3, push1_3_end=push2_3_end,
push1_4_0=push2_4_0, push1_4_1=push2_4_1, push1_4_2=push2_4_2, push1_4_3=push2_4_3, push1_4_end=push2_4_end,
push2_1_0=push1_2_0, push2_1_1=push1_2_1, push2_1_2=push1_2_2, push2_1_3=push1_2_3, push2_1_end=push1_2_end,
push3_1_0=push3_2_0, push3_1_1=push3_2_1, push3_1_2=push3_2_2, push3_1_3=push3_2_3, push3_1_end=push3_2_end,
push4_1_0=push4_2_0, push4_1_1=push4_2_1, push4_1_2=push4_2_2, push4_1_3=push4_2_3, push4_1_end=push4_2_end
```

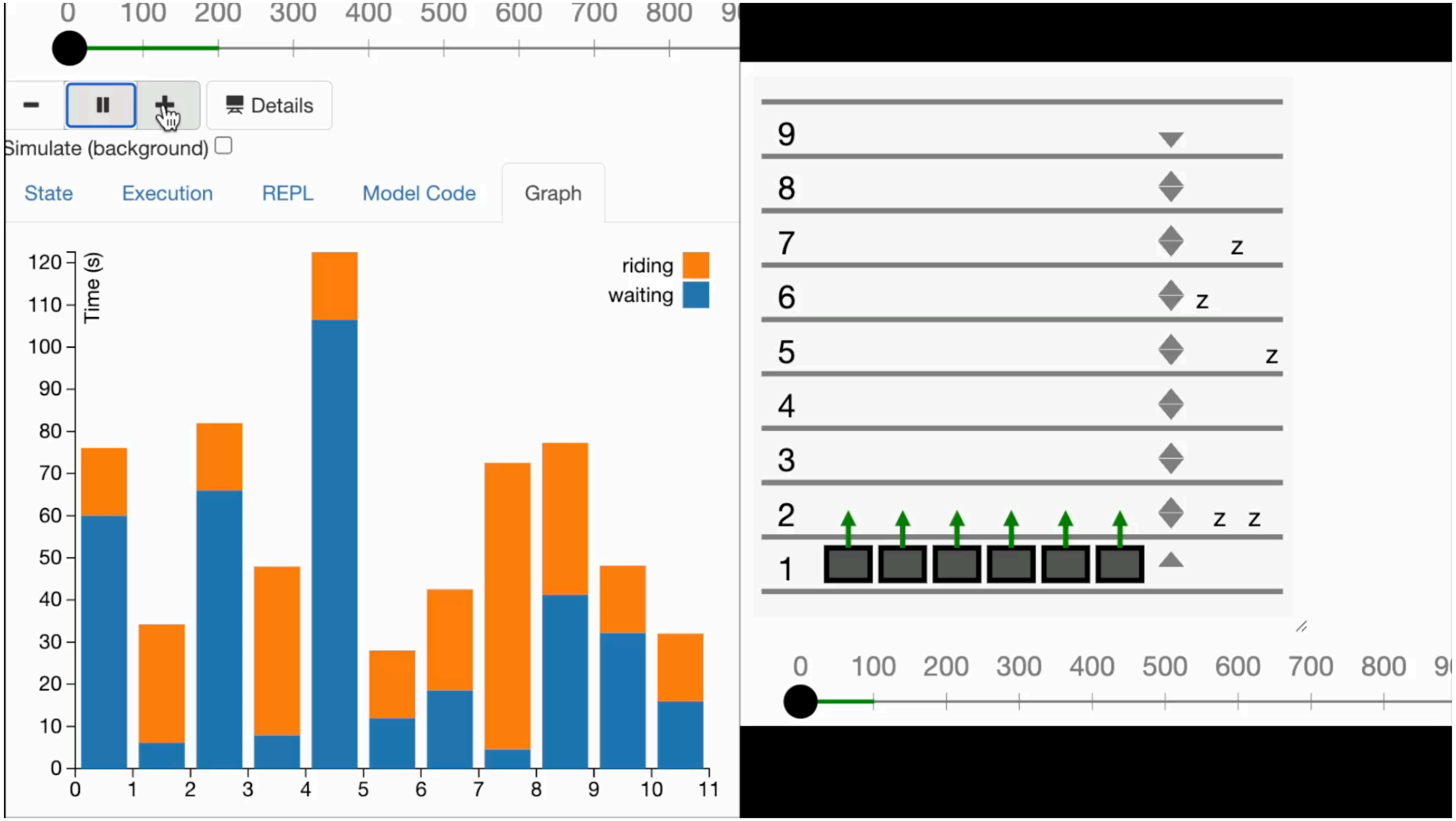
Start of the Art #3 of 4: Runway

Diego Ongaro

```
function quorum(serverSet: Set<ServerId>[ServerId]) -> Boolean {
  return size(serverSet) * 2 > size(servers);
}

function sendMessage(message: Message) {
  message.sentAt = later(0);
  message.deliverAt = later(urandomRange(10000, 20000));
  push(network, message);
}
```

Runway

Elevator Simulation



State of the Art #4 of 4: FizzBee

Jayaprabhakar "JP" Kadarkarai

cache.fizz

```
atomic action Lookup:
  cached = LookupCache()
  if cached == "hit":
      return cached
  found = LookupDB()
  return found

func LookupCache():
  oneof:
    `hit` return "hit"
    `miss` return “miss"
```

perf_model.yaml

```
configs:
  LookupCache.call:
    counters:
      latency_ms:
        numeric: 10
  LookupCache.hit:
    probability: 0.2
  LookupCache.miss:
    probability: 0.8
```

cost function

probabilities

```
Metrics(mean={'latency_ms': 84.4})
   2: 0.20000000 state: {} / returns: {"Lookup":"\"hit\""}
   4: 0.72000000 state: {} / returns: {"Lookup":"\"found\""}
   5: 0.08000000 state: {} / returns: {"Lookup":"\"notfound\""}
```

perf_model.yaml

```yaml
configs:
  LookupCache.call:
    counters:
      latency_ms:
        distribution: lognorm(s=0.3, loc=2)
  LookupCache.hit:
    probability: 0.2
  LookupCache.miss:
    probability: 0.8
```

Any probability distro supported by SciPy

or bring your own histogram

"Formal Methods Only Solve Half My Problems"

Marc Brooker, 2022:

"What I want is tools that do both: tools that allow development of formal models ... and then allow us to ask those models questions about design performance. Ideally, those tools would allow real-world data on network performance, packet loss, and user workloads to be used, alongside parametric models."

cache.fizz

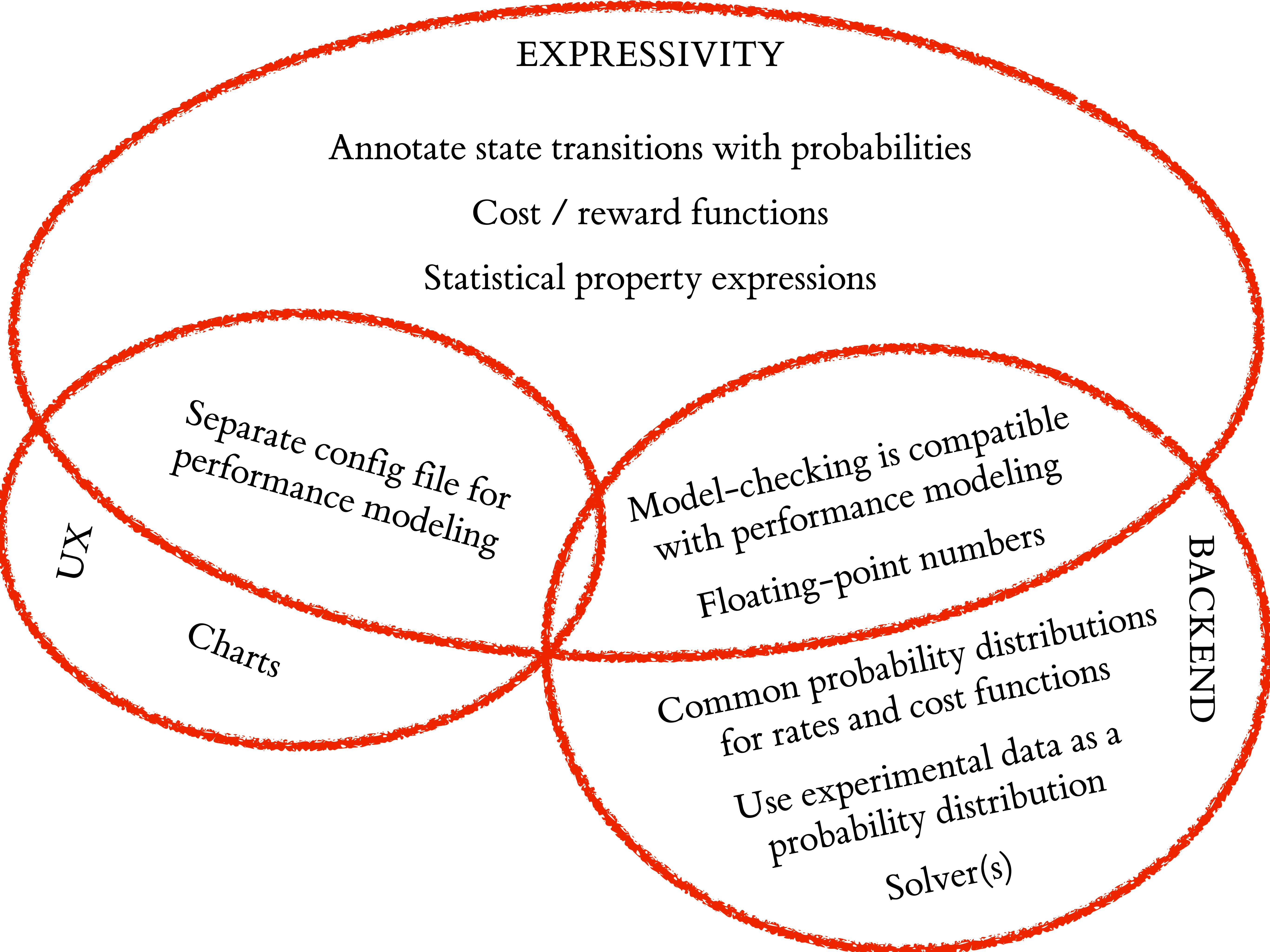
```
atomic action Lookup:
  cached = LookupCache()
  if cached == "hit":
      return cached
  found = LookupDB()
  return found

func LookupCache():
  oneof:
    `hit` return "hit"
    `miss` return “miss"
```

perf_model.yaml

```
configs:
  LookupCache.call:
    counters:
      latency_ms:
        numeric: 10
  LookupCache.hit:
    probability: 0.2
  LookupCache.miss:
    probability: 0.8
```

cost function

probabilities


EXPRESSIVITY
Annotate state transitions with probabilities
Cost / reward functions
Statistical property expressions
Separate config file for performance modeling
UX
Charts
Model-checking is compatible with performance modeling
Floating-point numbers
BACKEND
Common probability distributions for rates and cost functions
Use experimental data as a probability distribution
Solver(s)

Possible Syntax??

MySpec.tla

```
SendMessage(m) ==
  \E messageIsDropped \in {FALSE, TRUE}:
    ...
```

Possible Syntax??

MySpec.tla

```
SendMessage(m) ==
  \E messageIsDropped \in MessageLossProbability(FALSE, TRUE):
    ...
```

nondeterministically false / true
or a label for a
probability distribution

Possible Syntax??

MySpec.tla

```
SendMessage(m) ==
  \E messageIsDropped \in MessageLossProbability(FALSE, TRUE):
    ...
```

MySpec.cfg

```
DISTRIBUTION
  MessageLossProbability = BooleanChoice(0.23)
```

Possible Syntax??

MySpec.tla

```
SendMessage(m) ==
  \E messageIsDropped \in MessageLossProbability(FALSE, TRUE):
    ...
```

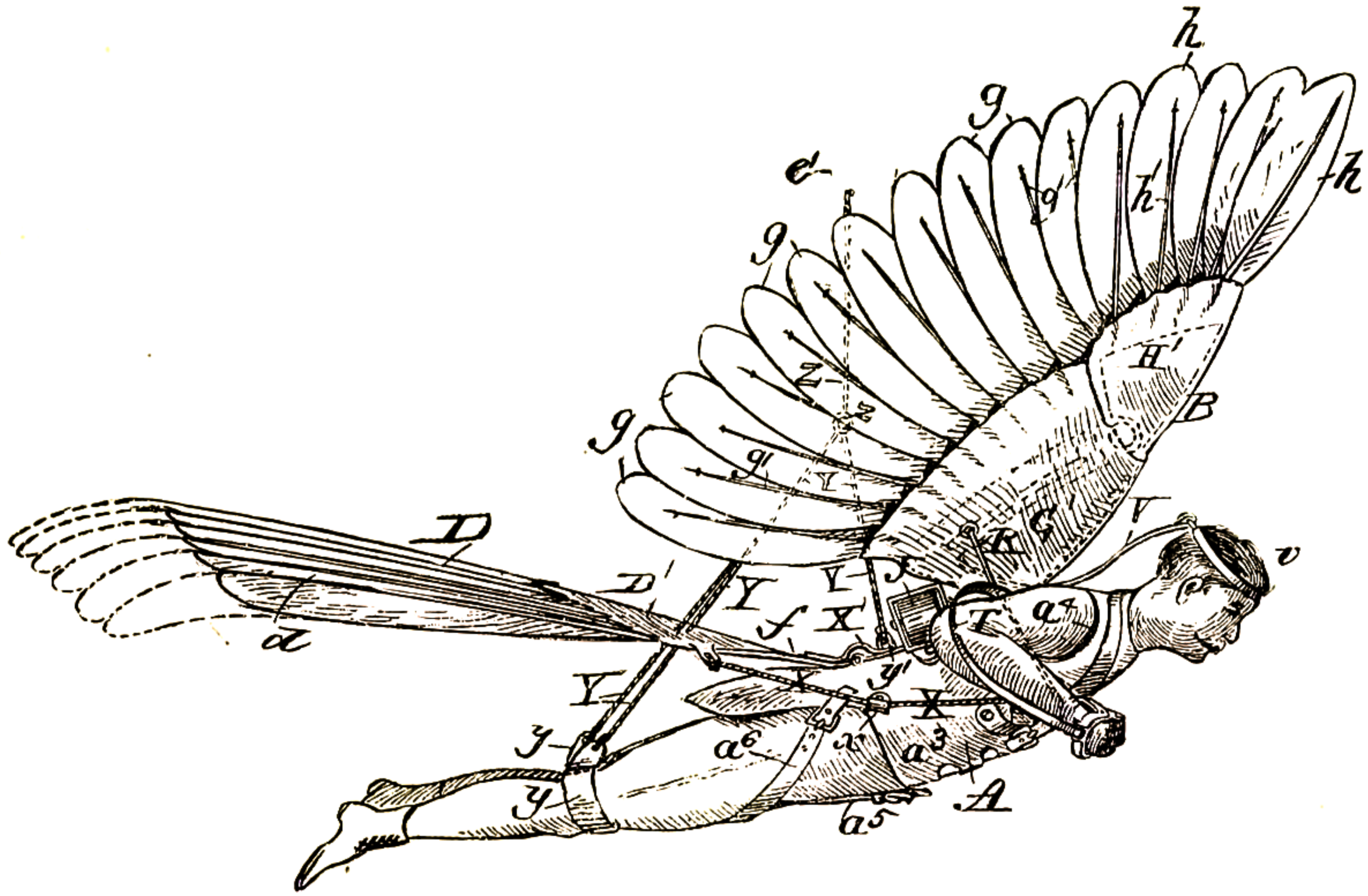
MySpec.cfg

```
DISTRIBUTION
  MessageLossProbability = BooleanChoice(0.23)

COST
  SendMessage = Exponential(3.17)
```

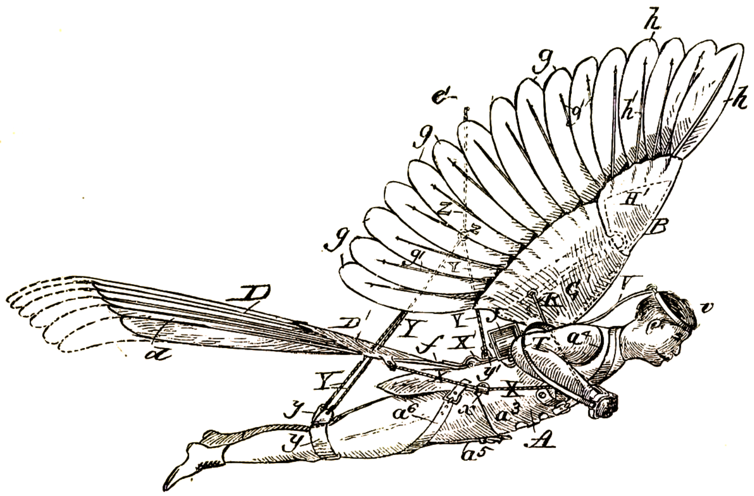
TLA+ with Probabilistic Solvers

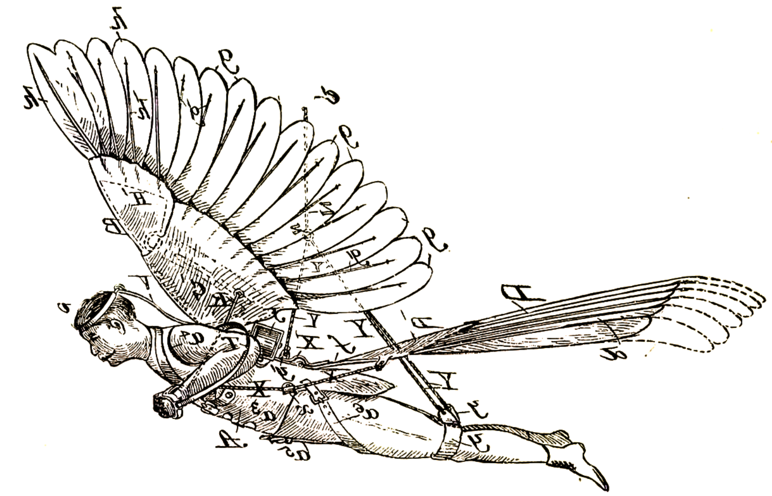
In order of ambitiousness....

- Just use `-generate`, generate thousands of behaviors, average the stats.
- Use `-generate`, run until stats stabilize within some precision, perhaps prune branches of the state graph as they stabilize.
- Use PRISM's solvers (by translating the state graph to PRISM?).
- Write a solver or solvers from scratch: translate the state graph to a Markov chain and find its steady-state probability distribution.



TLA⁺ with Performance Modeling





One model could:

- Express the algorithm.
- Check correctness.
- Evaluate performance.
- Simulate "what-if" experiments using real-world inputs.
- Confidently explore optimizations.

Acks

- Andrew Helwer
- Jayaprabhakar Kadarkarai
- Murat Demirbas
- Will Schultz

Questions

1. What syntax should TLA+ use for annotating state transitions with probabilities?

2. What syntax for cost functions?

3. How do we separate performance-modeling config from the spec and model-checking config?

4. Should TLC do the probabilistic checking, or another tool?

5. Could the TLA+ Foundation get new funding for this work?

6. Is any of this a good idea or should TLA+ stick to correctness?