Exploiting Uncertainty and Error to Accelerate Simulations

David M. Nicol

University of Illinois at Urbana-Champaign

Modeling is the Art of Abstraction

Science is based on models of phenomena

- If there's an equation, there's a model
  - Chemical reactions
  - Biological systems
  - Physics
  - Engineering
  - Etc.

Every model abstracts away some detail, e.g.

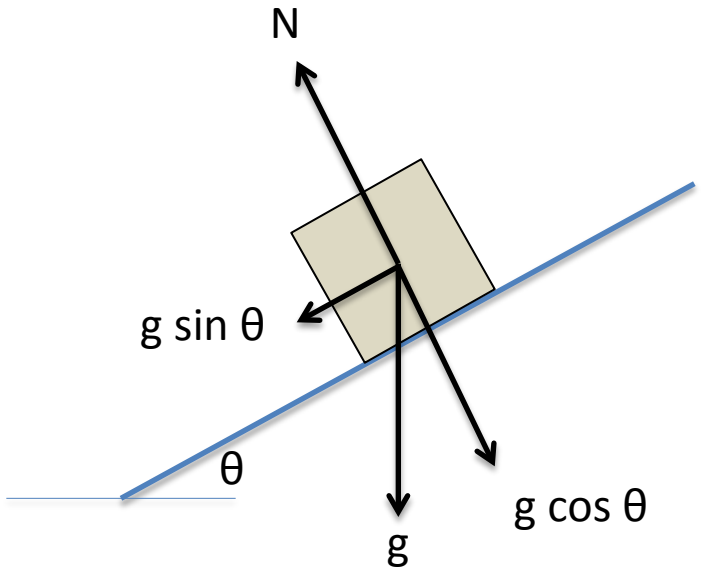
- Physical characteristic
- Time scale

Example from high school

Block on an inclined plane : no friction

Model :

- Center of mass
- Gravity g (decomposed in coordinates relative to plane)
- “Normal” force orthogonal to plane

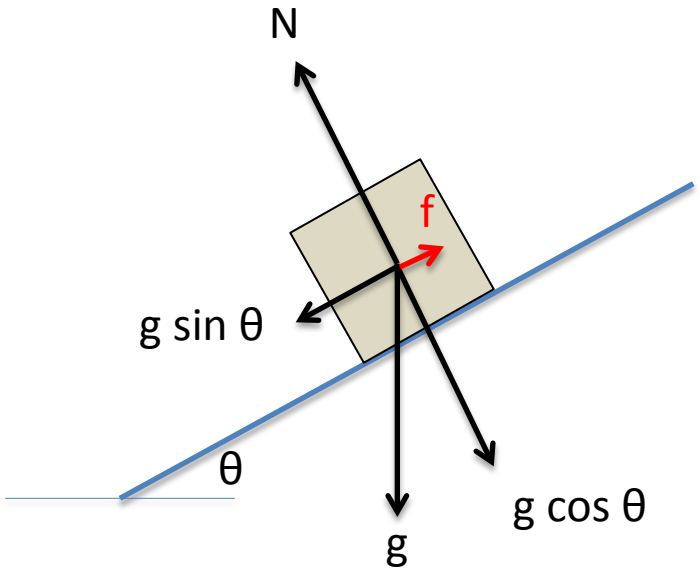


Example from high school

Block on an inclined plane : add friction

Model :

- Center of mass
- Gravity g (decomposed in coordinates relative to plane)
- “Normal” force orthogonal to plane
- Friction modeled as force opposed to movement along plane

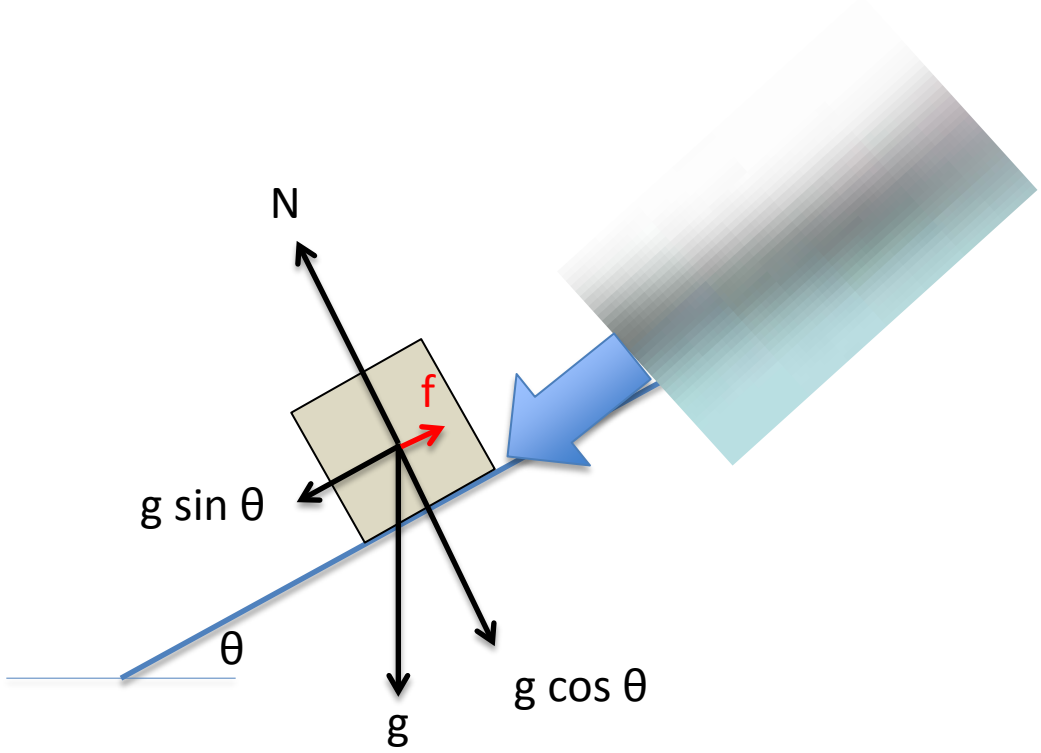


But what is friction, really?

At atomic scales friction is due to geometry and electrostatic forces

One could in principle model this ab initio

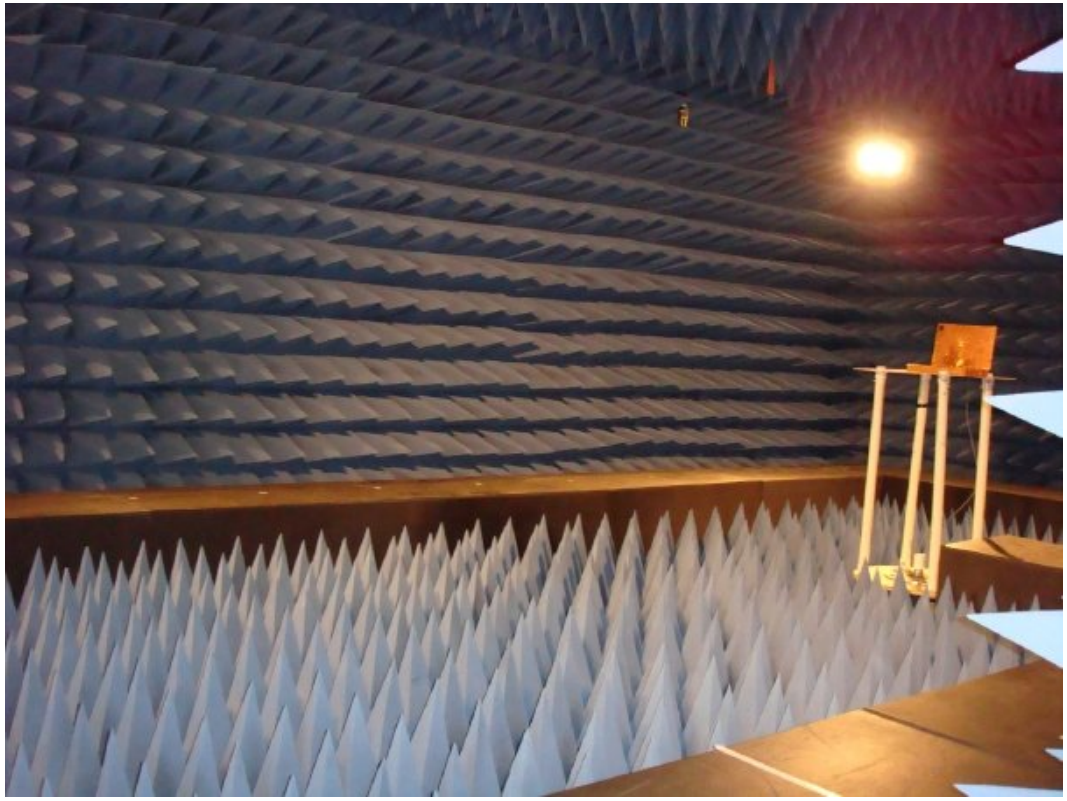
- But this is computationally intractable at scale
  - point being : abstraction helps to accelerate solution



Uncertainty in Parameters

Abstraction parameters may be uncertain

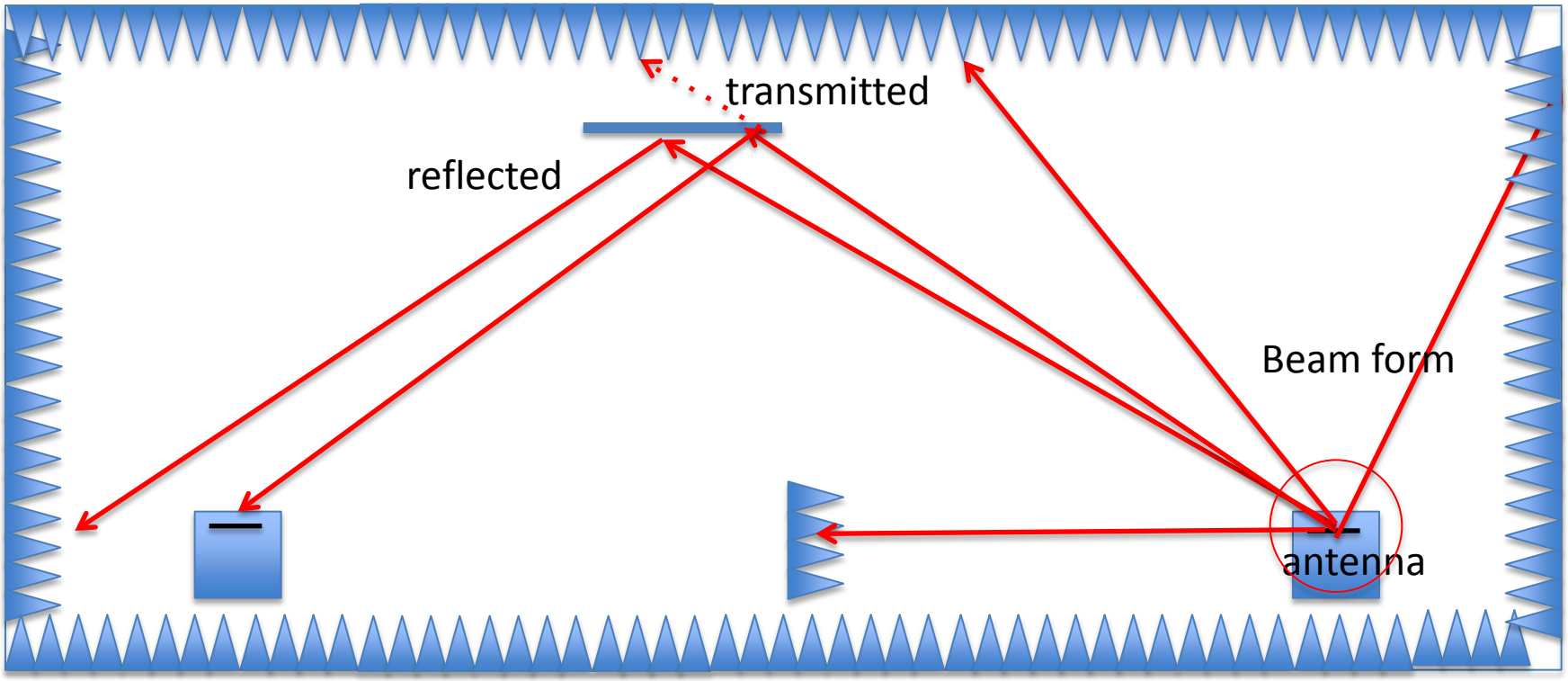
Example from recent work --- validation of radio channel models in anechoic chamber



Uncertainty in Parameters

Ray-tracing model needs

- Shape of antenna
- Shape of transmission (beam form)
- Transmission strengths
- Material coefficients for reflection / transmission

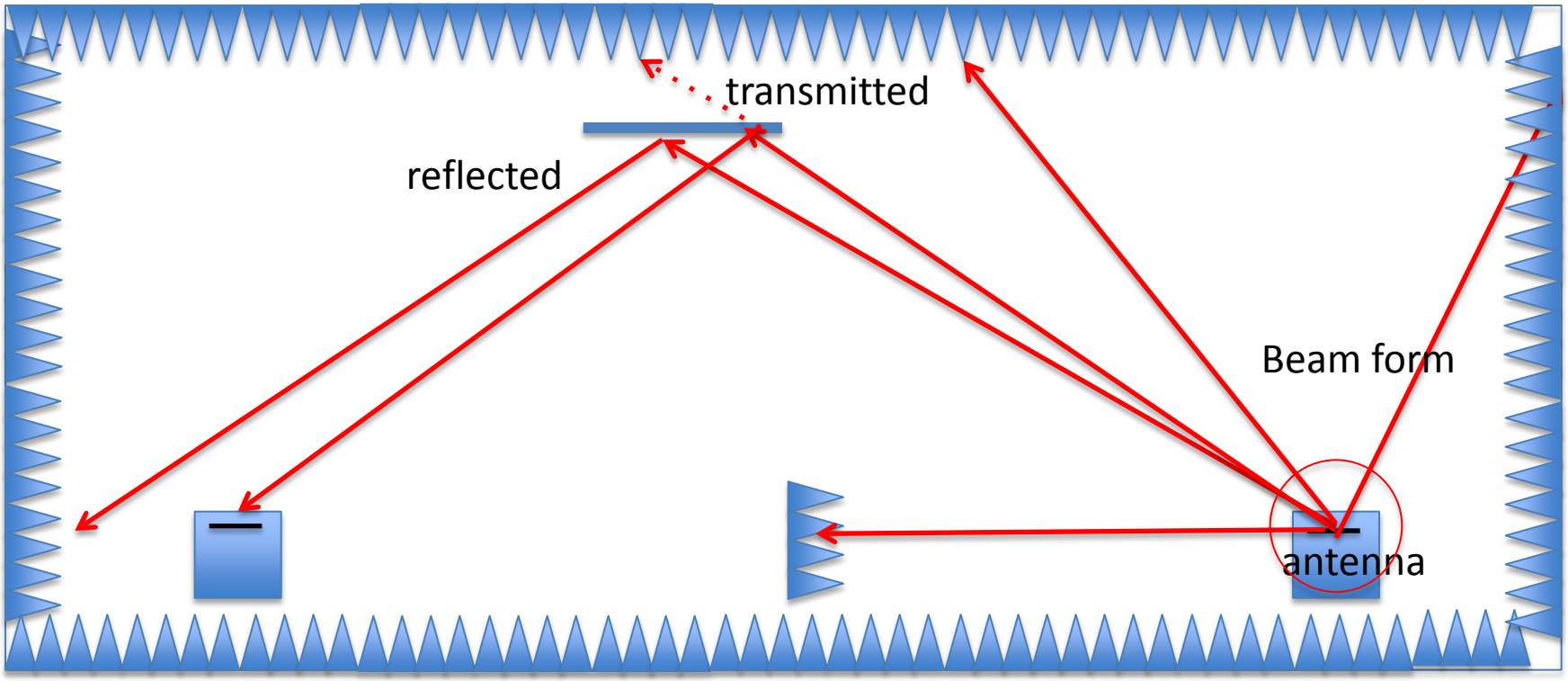


Uncertainty in Parameters

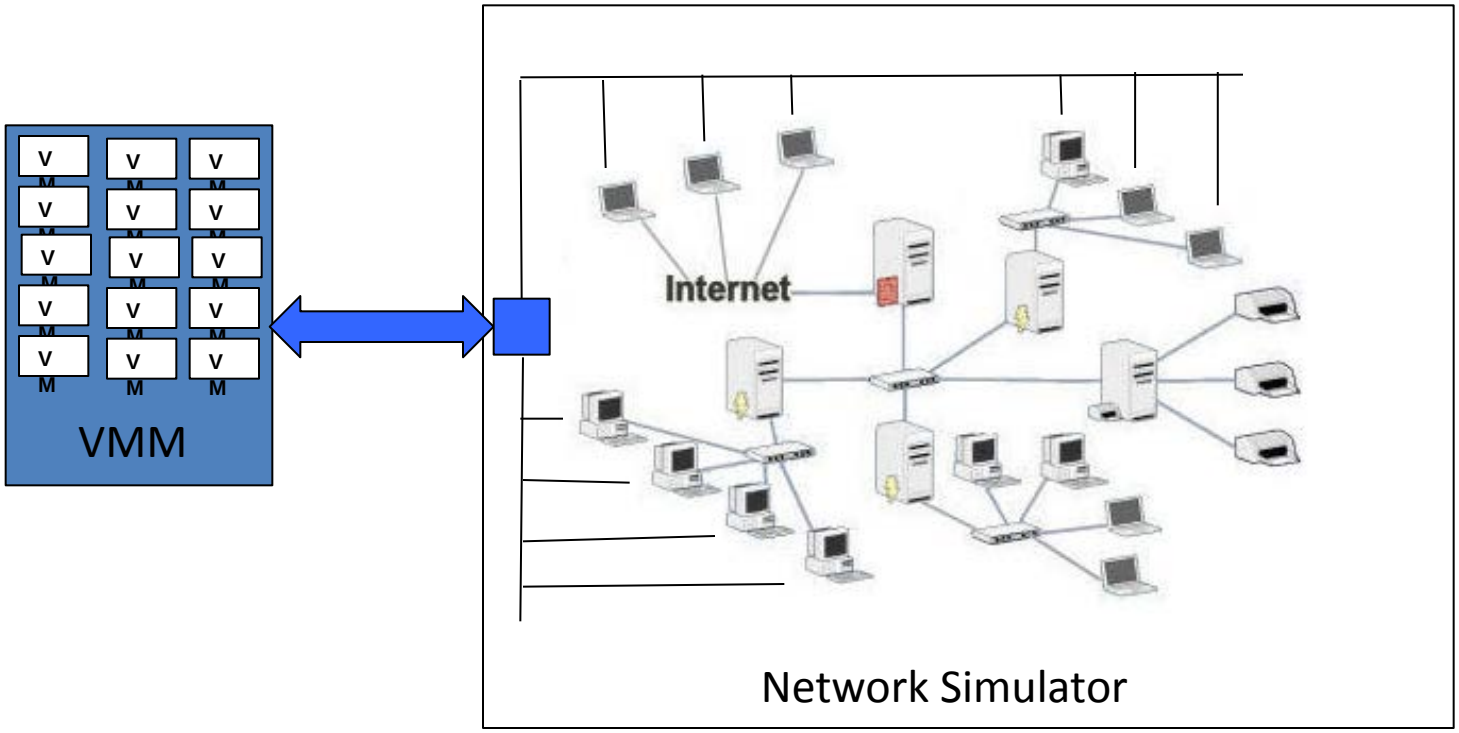
Ray-tracing model needs

- Shape of antenna
- Shape of transmission (beam form)
- Transmission strengths
- Material coefficients for reflection / transmission

In most cases we cannot get these right

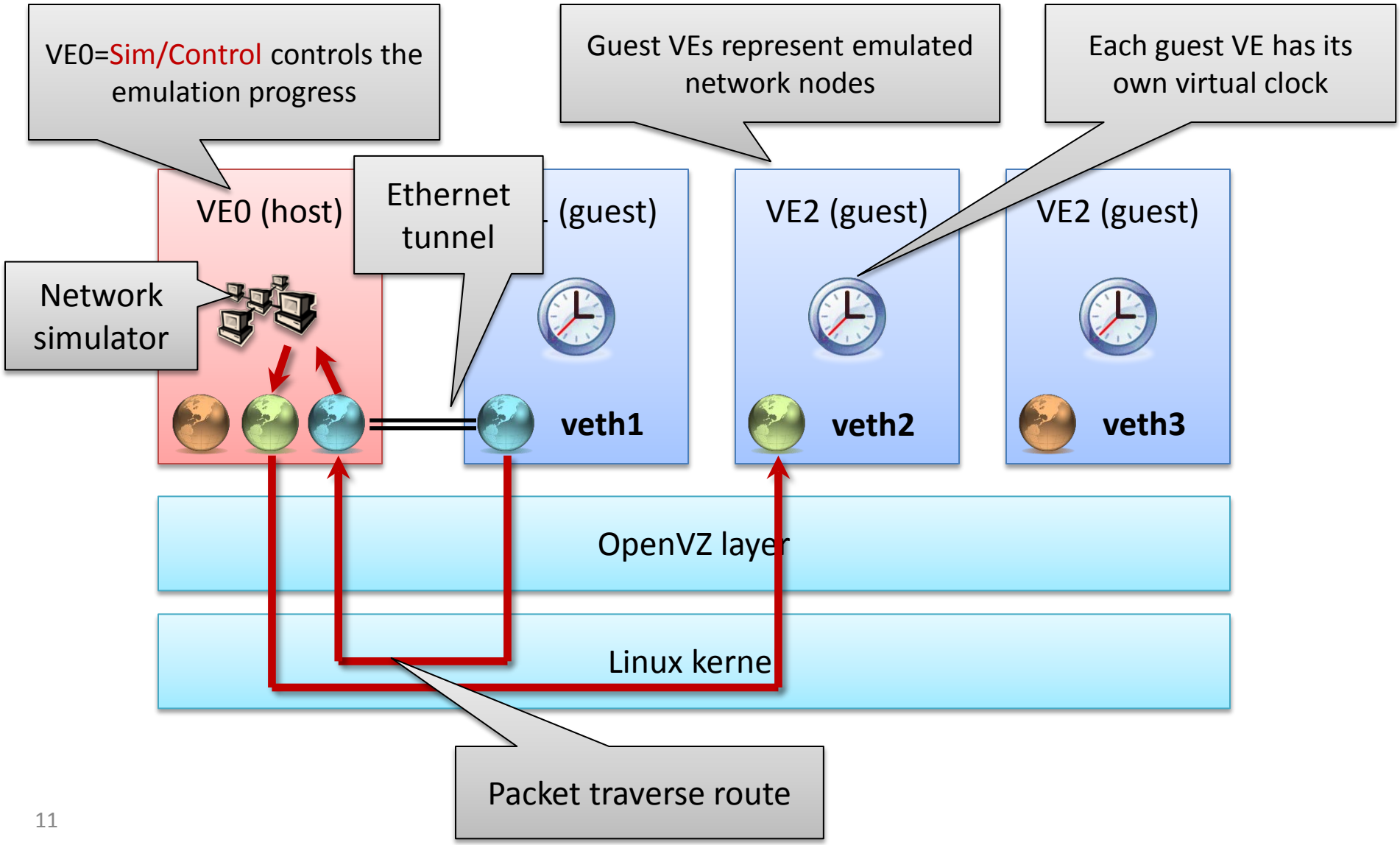


Emulation, Simulation, and Temporal Error



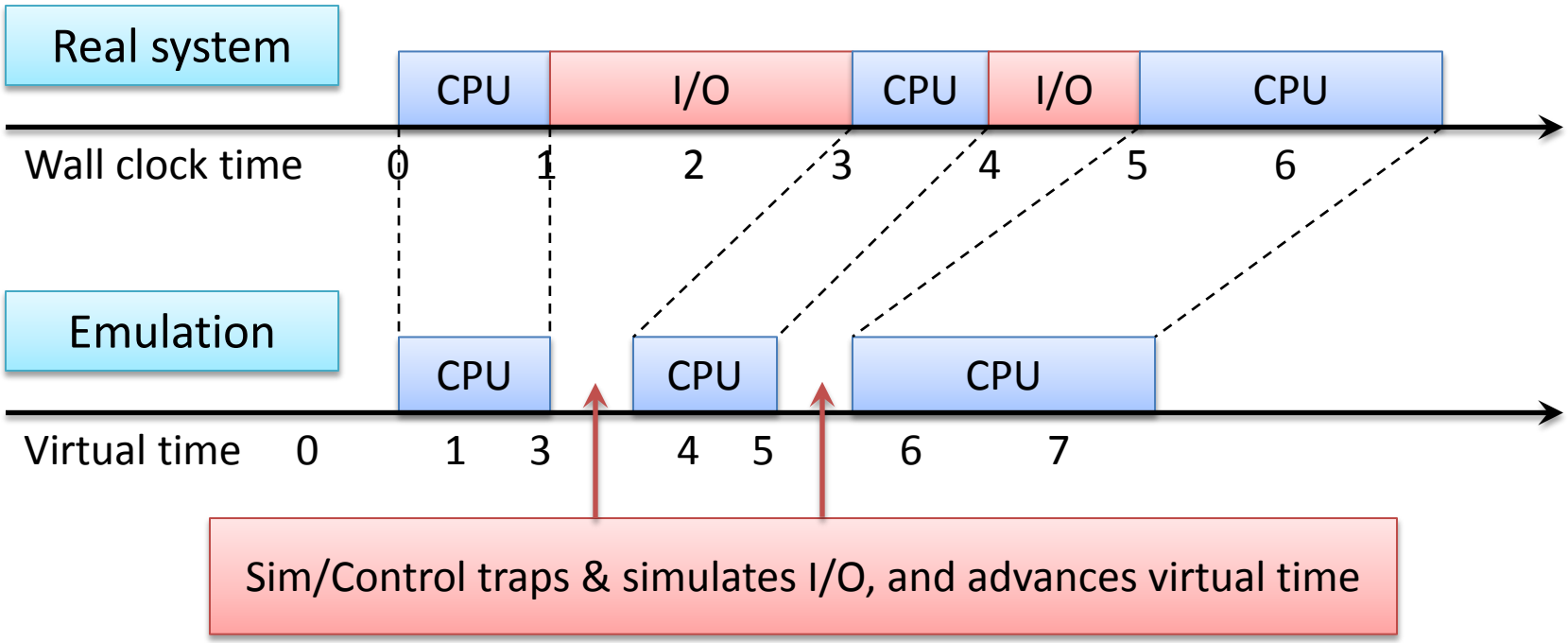
- Virtualization supports high fidelity application behavior
- Time-stamps on traffic affect congestion, hence latency and packet loss
- Emulators need to get access to virtual clocks, not physical ones

System Architecture



Virtual Time

- VEs perceive time *as if* running concurrently
- Clocks advance: *either* CPU *or* I/O



Sources of Temporal Errors

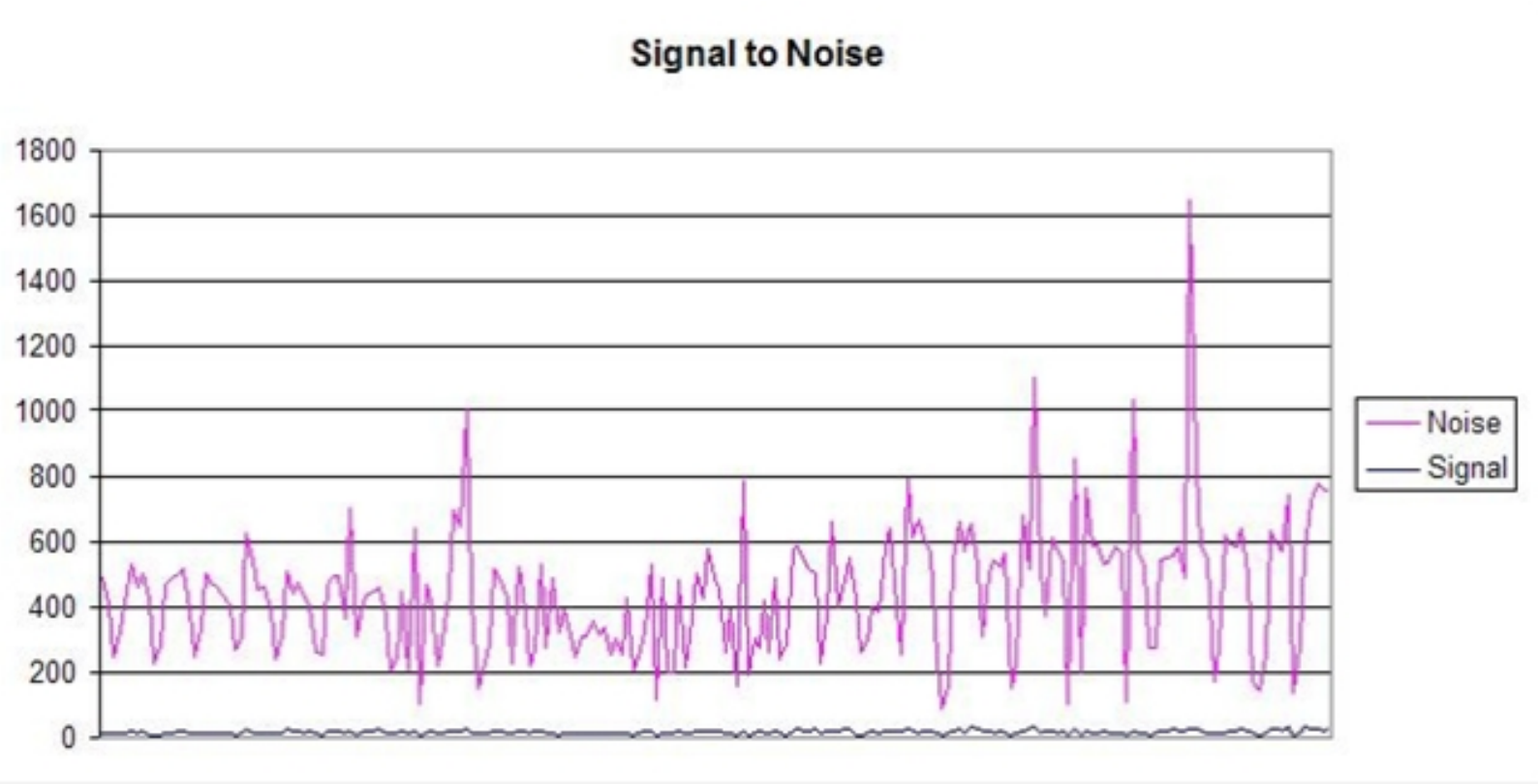
- Measuring execution time
  - Caching effects, pipeline effects, etc.
- Variation in timeslice lengths
  - 10% variation in 1 ms slice not uncommon
- Impact of background system activity
  - e.g., page faults, periodic timers, background daemons

To use emulation means one has to accept some uncertainties / errors

Abstraction and Uncertainty

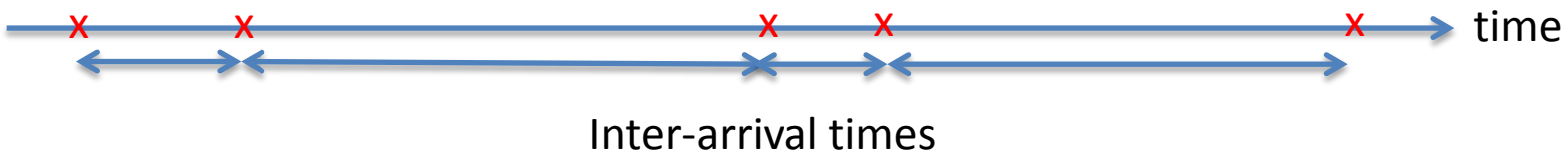
Sometimes abstraction covers uncertainty

- Signal processing : models of signal + noise
  - "noise" modeled with some probability distribution or time-series

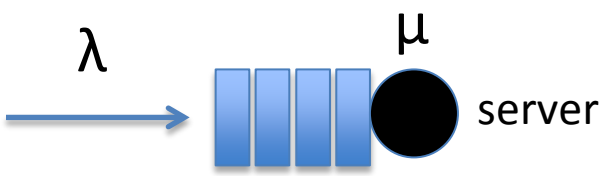


Abstraction and Uncertainty

- Sometimes abstraction covers uncertainty
- Arrivals to queuing systems
  - We don't know what complex psychological factors lead a population to make calls
    - But we model inter-arrivals stochastically to reflect variation, and observed intensities



Inter-arrival times



Queuing Model

Lots of uncertainty and error in models

Models are rife with uncertainty and error

How can we take take advantage of this?

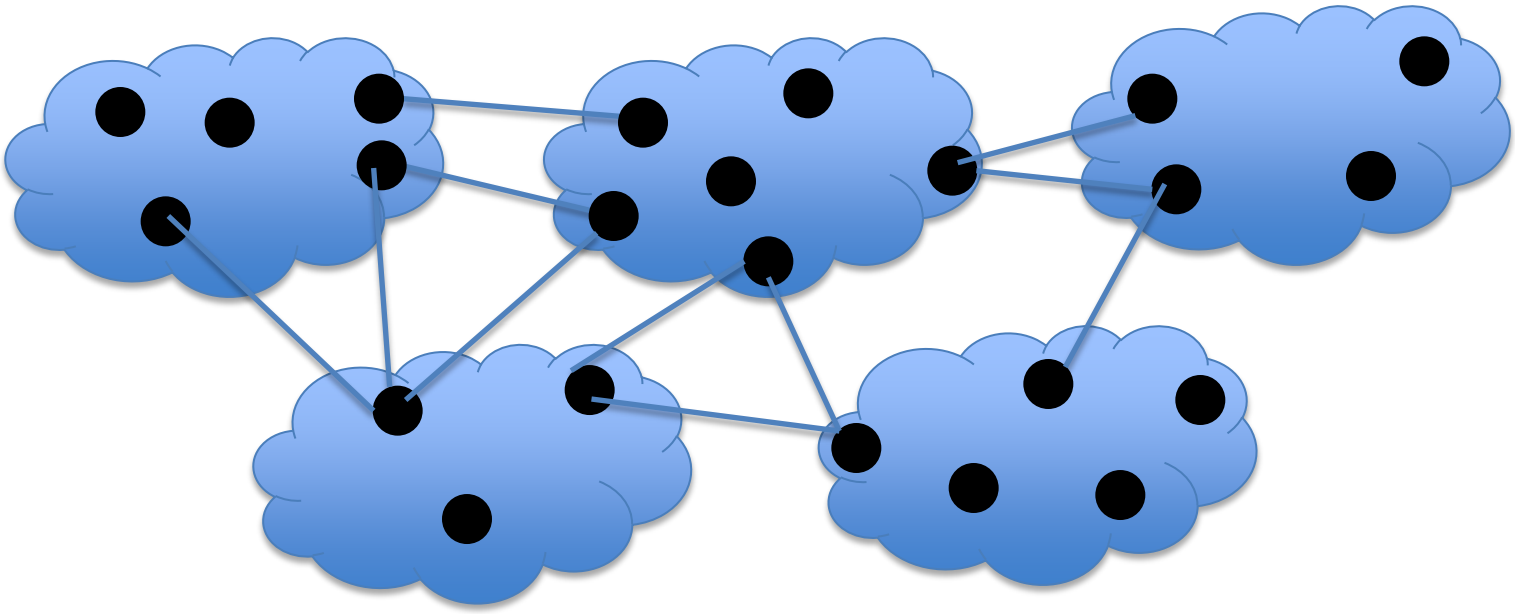
Use to accelerate simulation solution

- Abstract away details that don't help
  - Less work, faster answers!
- Other less obvious techniques?
  - Exploit for synchronization in parallel simulation

Brief Primer on Parallel Simulation

Most usual approach

- Partition model state by physical domain
- Each submodel advances through single “event-list”
- Some coordination needed because activity in one submodel may affect the other



"Conservative" means "Look-ahead"

So-called "conservative" approaches to synchronization involve "look-ahead"

- Prediction of future behavior w.r.t. interactions with other submodels
- Allows a sub-model to advance asynchronously through this safe period
- The challenge always is to exploit the model to get the look-ahead

How can we constructively use model uncertainty and error to create look-ahead?

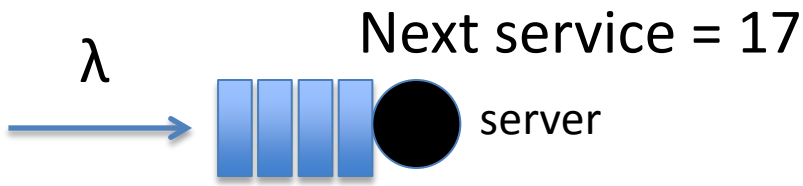
First approach – pre-sampling

Randomness captures model uncertainty and variation

- Inter-arrival times, service times, collection sizes

Typically these are computed on demand, using pseudo-random number generators

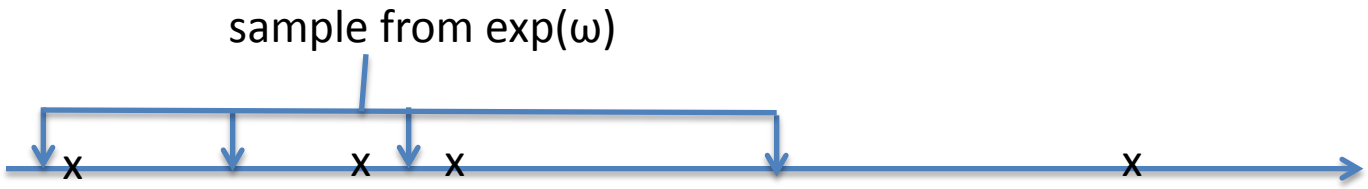
- Computing ahead of time sometimes allows look-ahead
  - Example, FCFS queuing, state-independent service times



- If queue is empty at time s, no departure possible before s plus pre-sampled next service time
- When arrival does occur, at t,
  - Next service sampled
  - Departure immediately reported, along with departure time plus new next service time

Fancier pre-sampling in CTMC

- Inter-arrivals in Poisson process are i.i.d. exp(λ)
- State holding times in continuous time Markov chain are independent exponentials, not identical
- "uniformization" is basis for fancier pre-sampling when rates are unknown, but bounded
- The clever trick: sample an exponential with rate λ, using potentially more samples of exponentials with rate ω > λ



Accept each event with probability λ/ω, stop on success

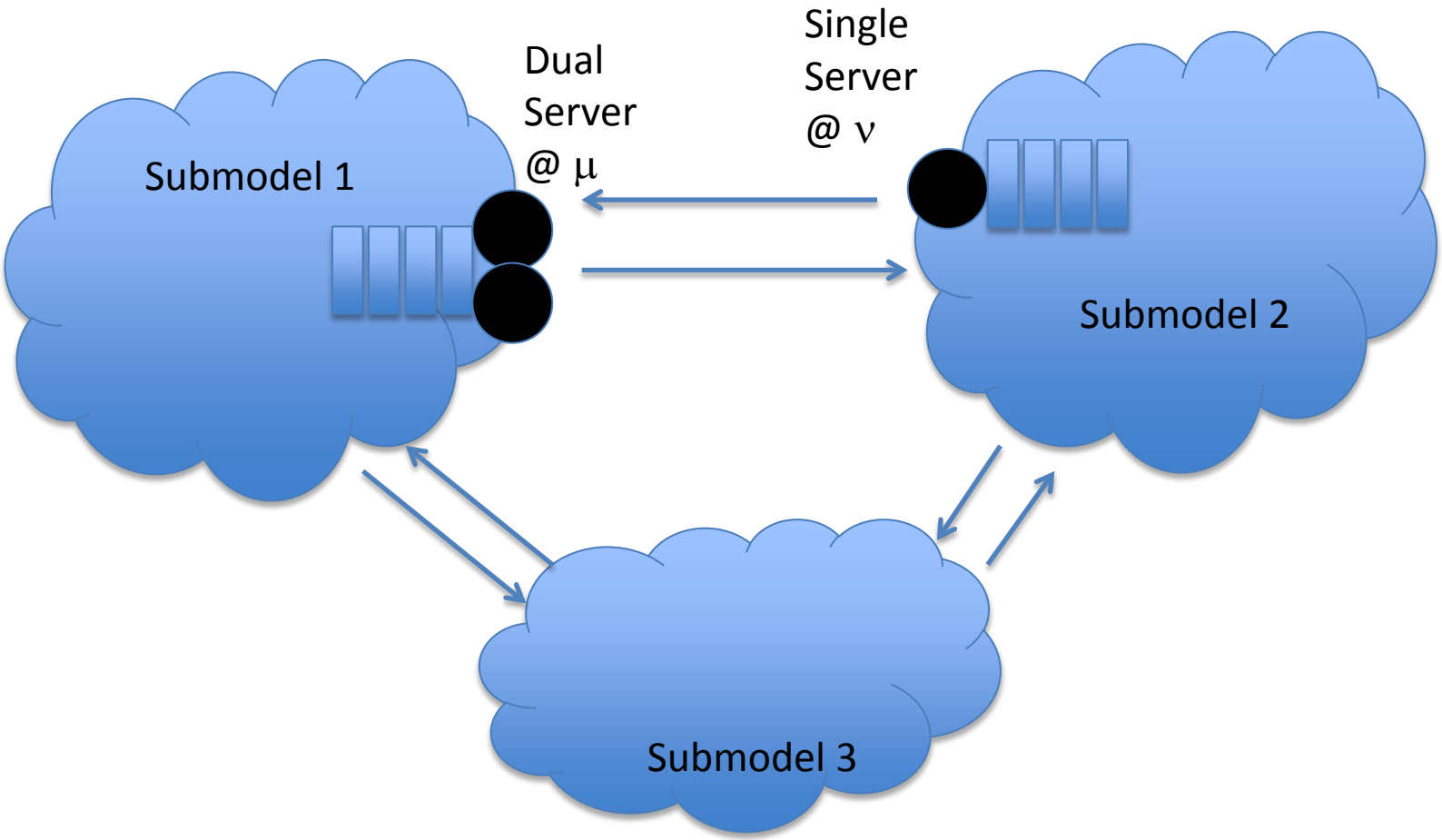
Technically, exp(λ) is a geometric sum of i.i.d. exp(ω)

Lookahead --- pre-sample the uniformized holding times

Concrete Example

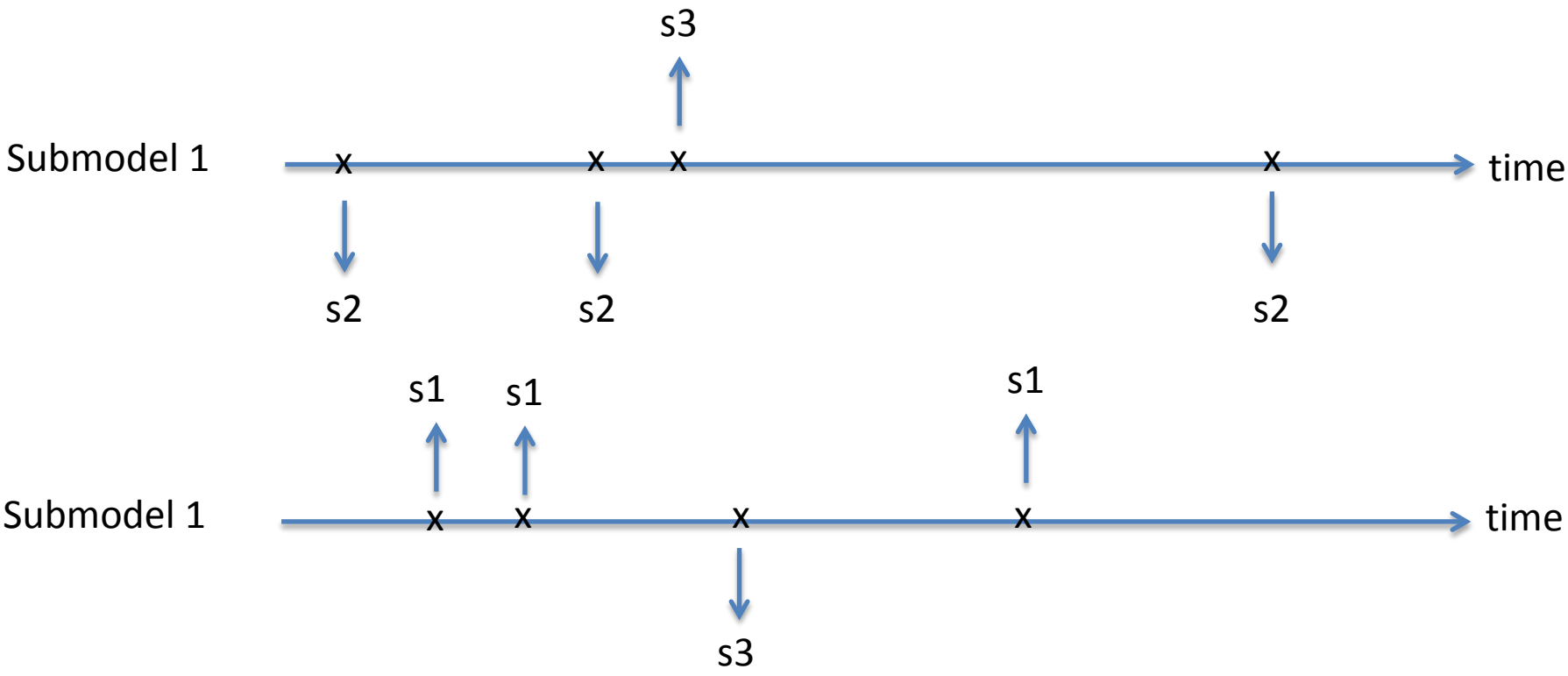
Uniformized rate submodel 1 to submodel 2 : 2μ; actual rate : (#busy servers)μ

Uniformized rate submodel 2 to submodel 1 : ν; actual rate : (#busy servers) ν



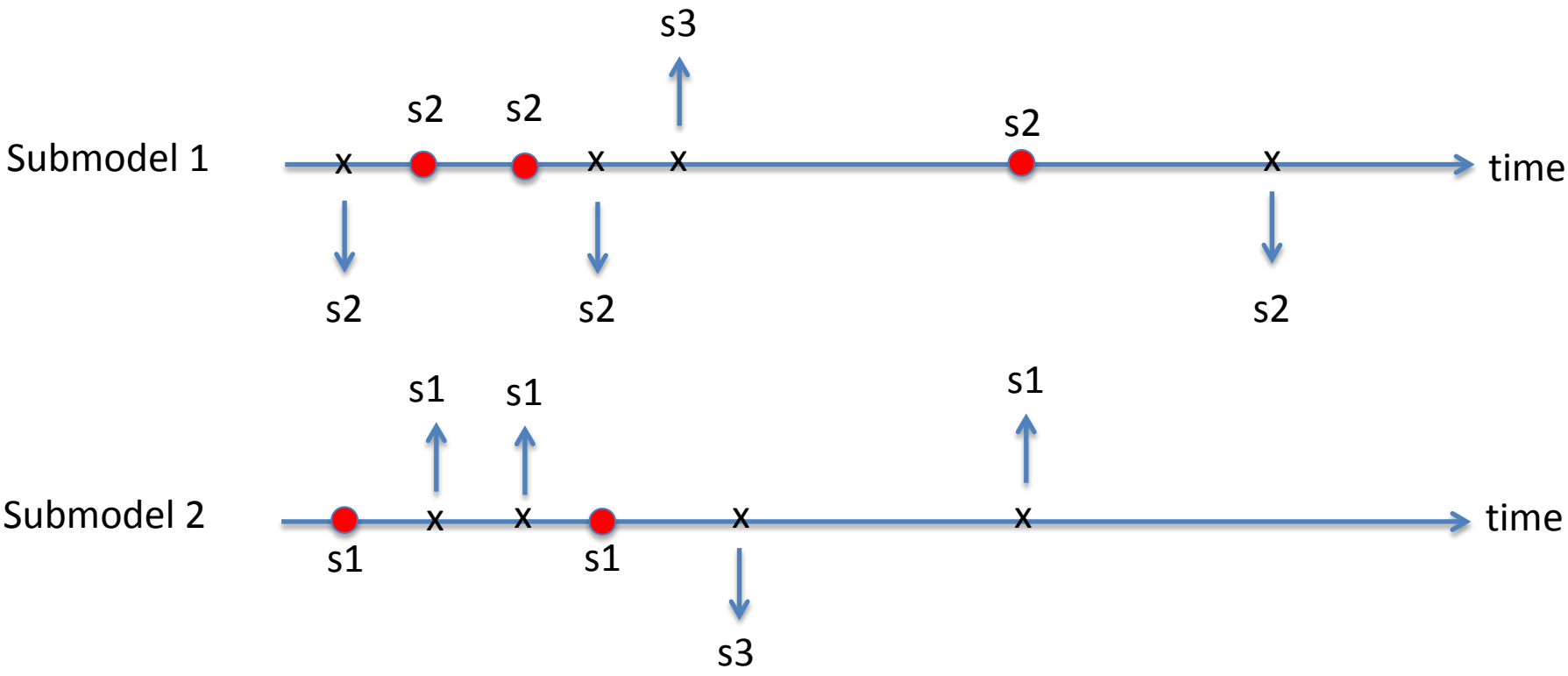
Fancier pre-sampling in CTMC

- Submodels generate pre-sampled times between uniformized events



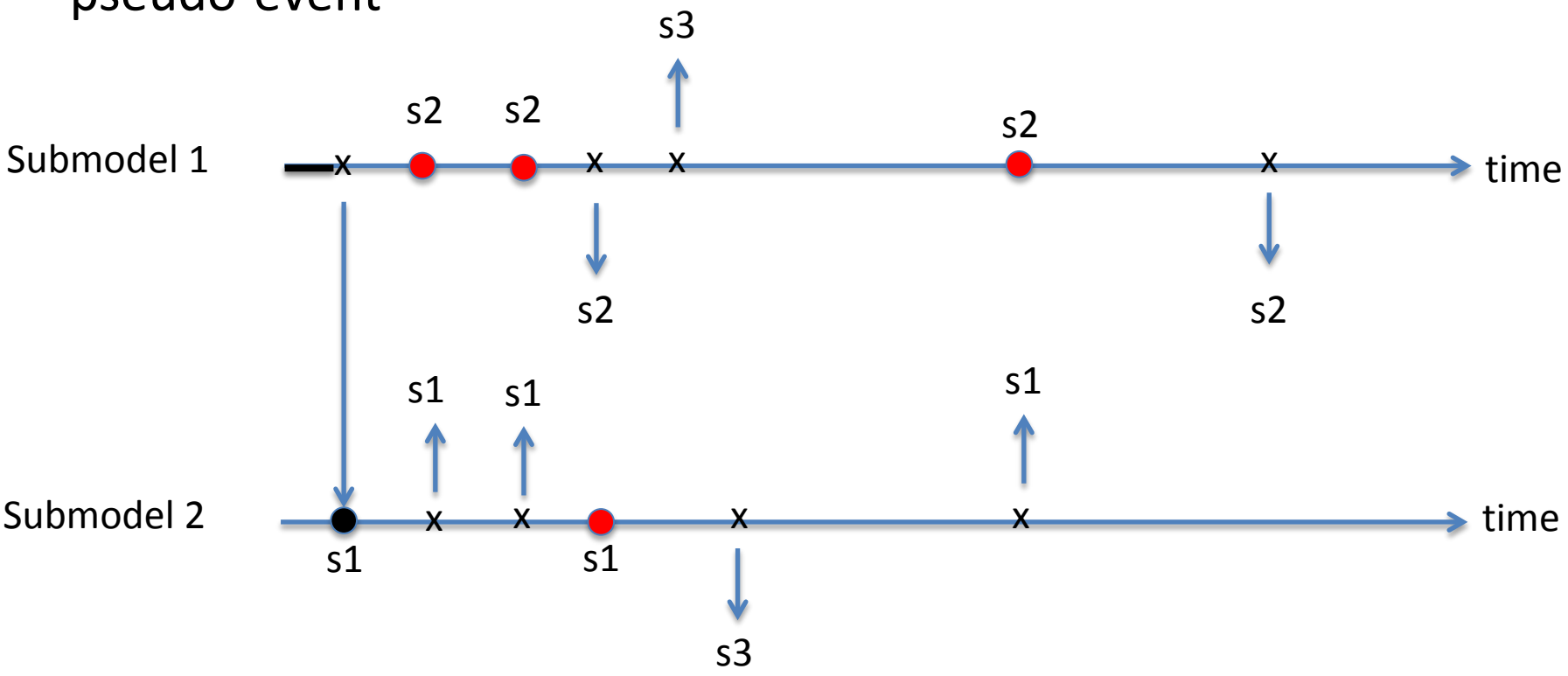
Fancier pre-sampling in CTMC

- Submodels exchange lists, create synchronization schedule



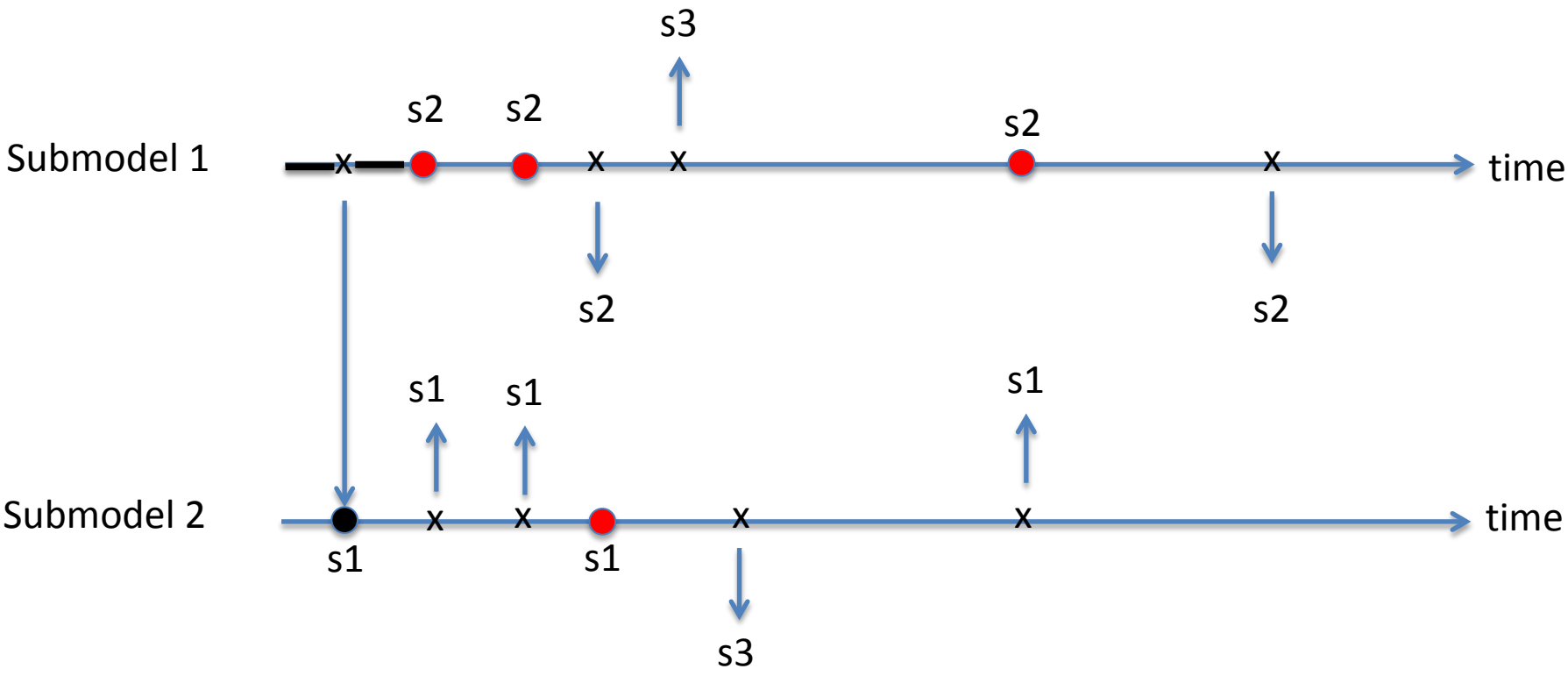
Fancier pre-sampling in CTMC

- When submodel reaches outbound event, flips coin and transitions with probability λ/ω, sends message reporting transition or pseudo-event



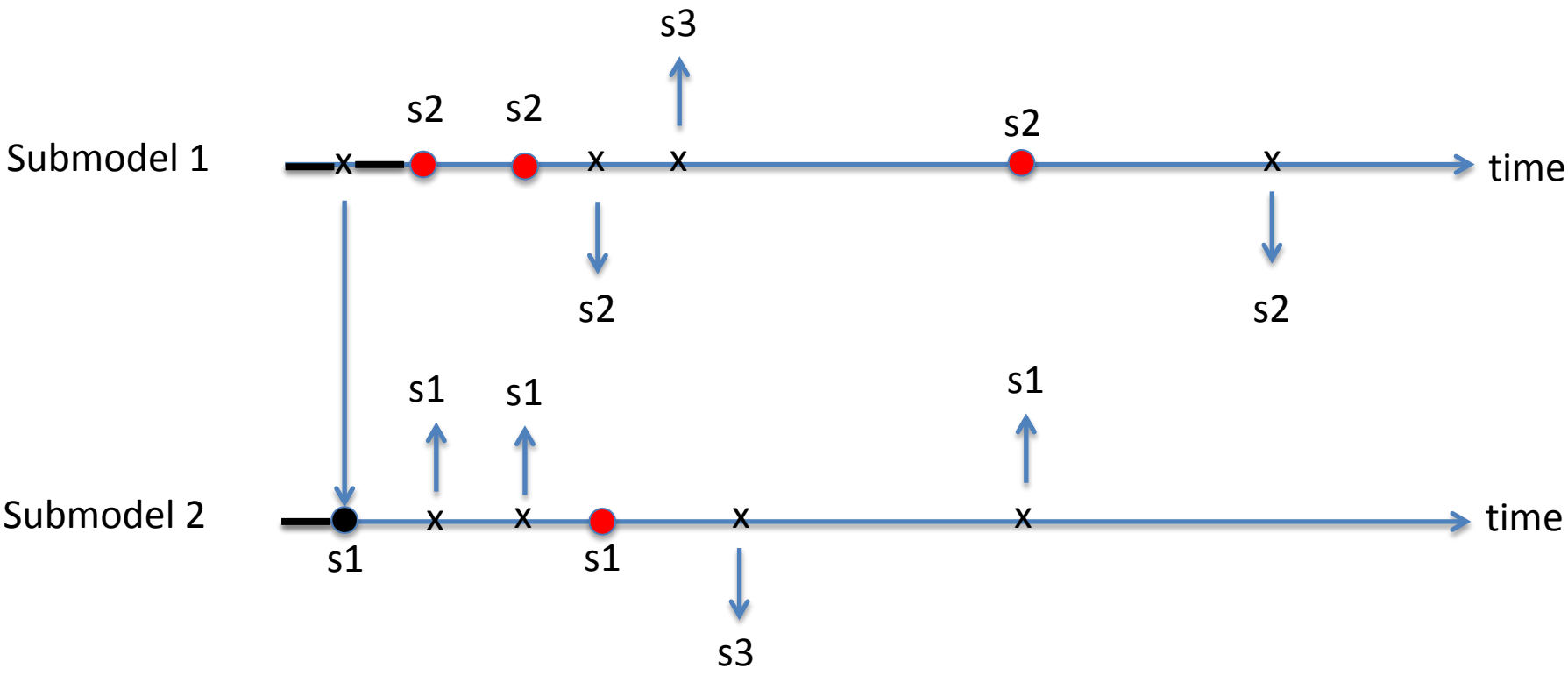
Fancier pre-sampling in CTMC

- When submodel reaches inbound event, waits for receipt of message before advancing



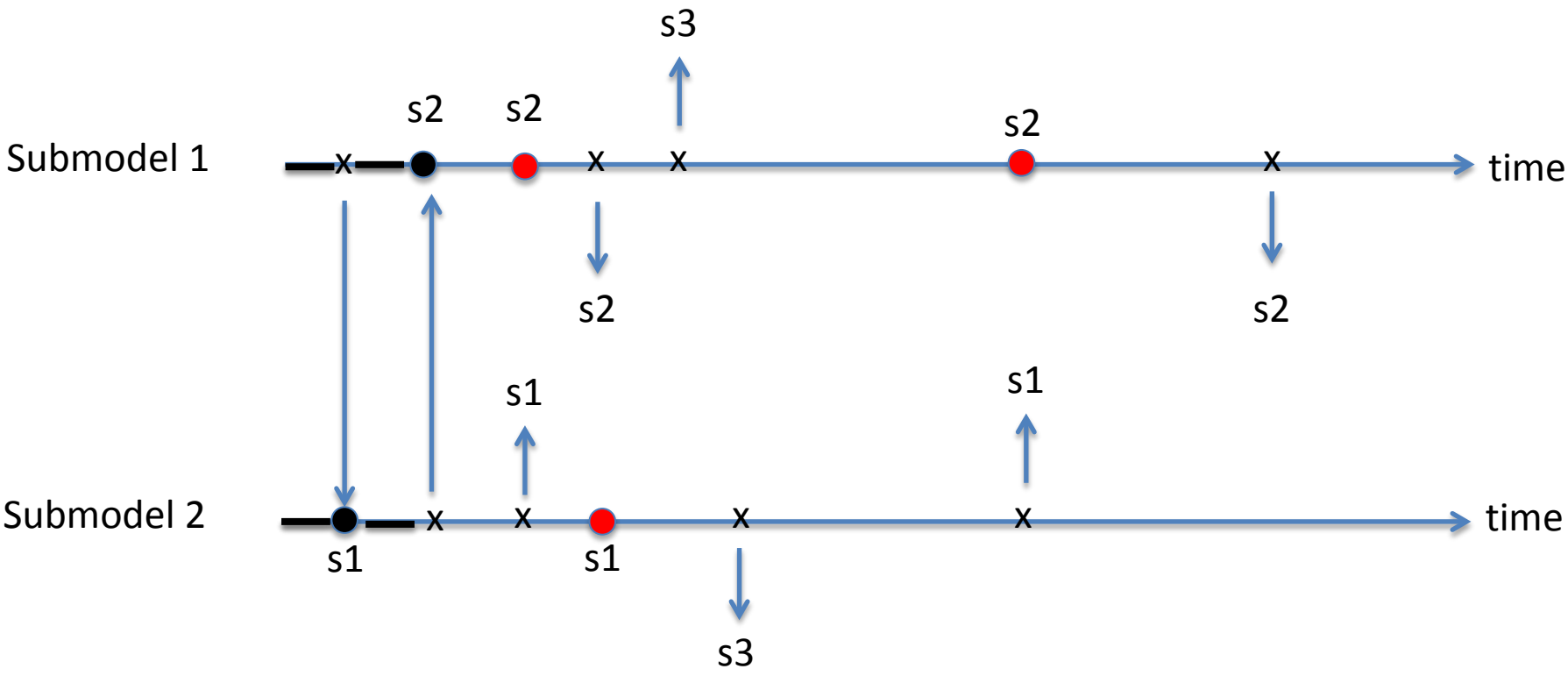
Fancier pre-sampling in CTMC

- When submodel reaches inbound event, waits for receipt of message before advancing



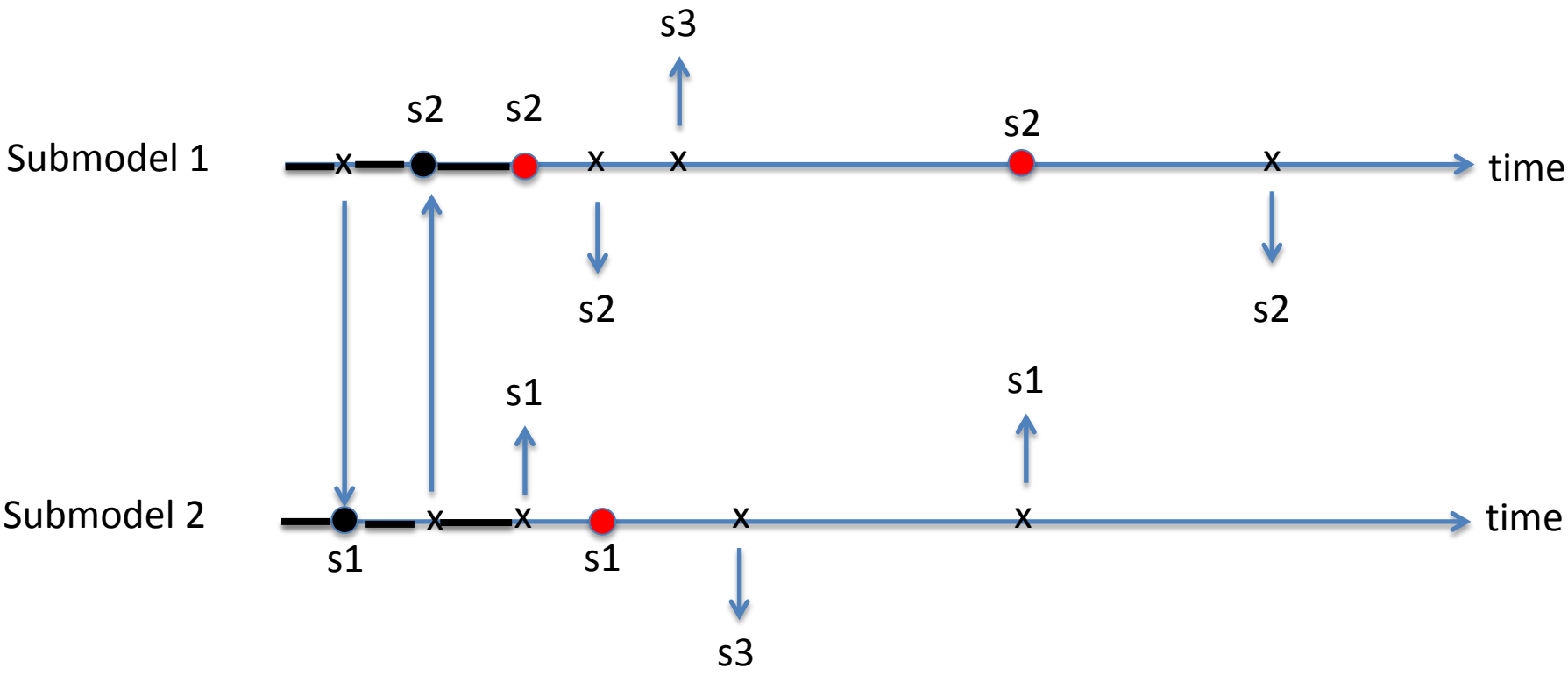
Fancier pre-sampling in CTMC

- When submodel reaches inbound event, waits for receipt of message before advancing



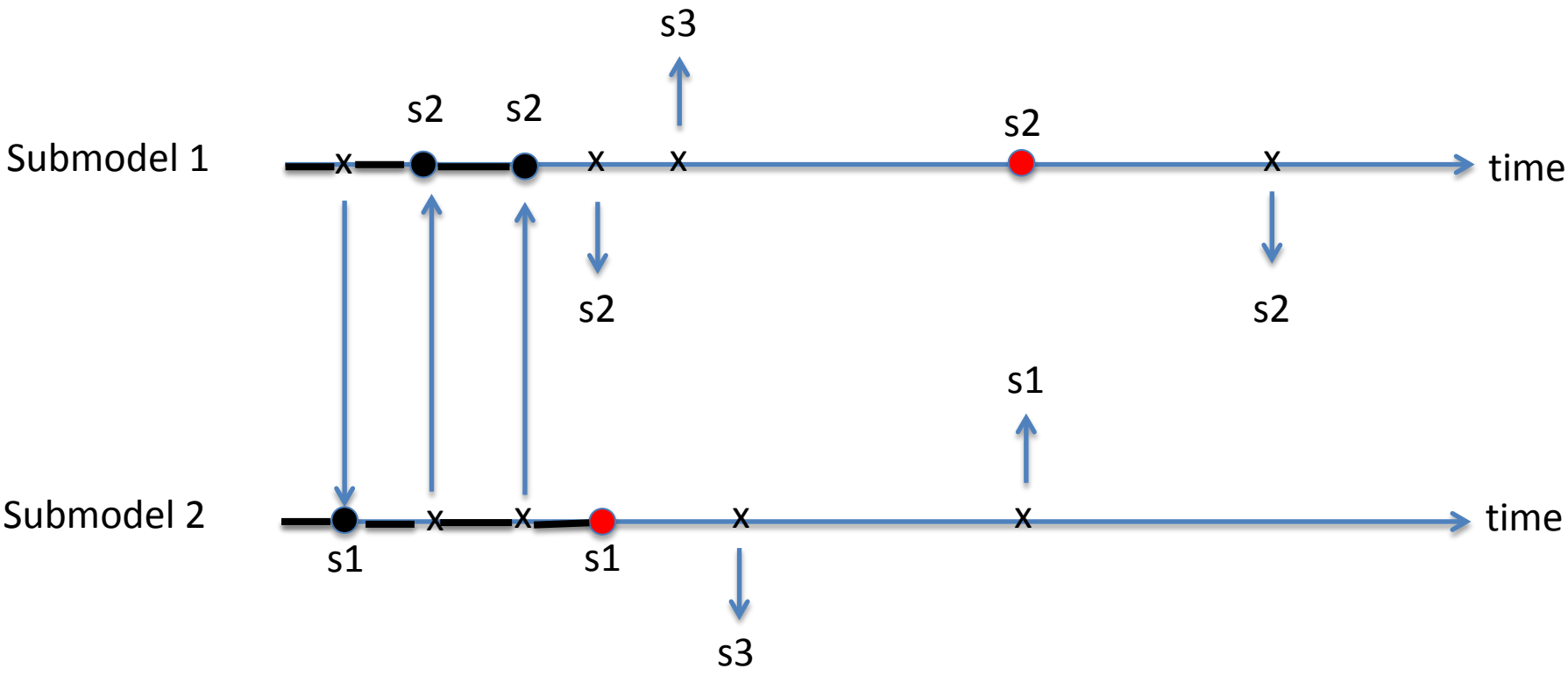
Fancier pre-sampling in CTMC

- When submodel reaches inbound event, waits for receipt of message before advancing



Fancier pre-sampling in CTMC

- When submodel reaches inbound event, waits for receipt of message before advancing



It really works

- FCFS pre-sampling gave nearly perfect speedups on problems formerly thought to defeat conservative parallel simulation

- A uniformization simulation once held the world's largest speedup record (I think)
  - O(500) speedup, massive queuing network

Notable --- these were techniques for changing the way a given model is evaluated

Changing the model to accelerate simulation

Argument : It can be OK to change the model since some of its parameters are wrong anyway

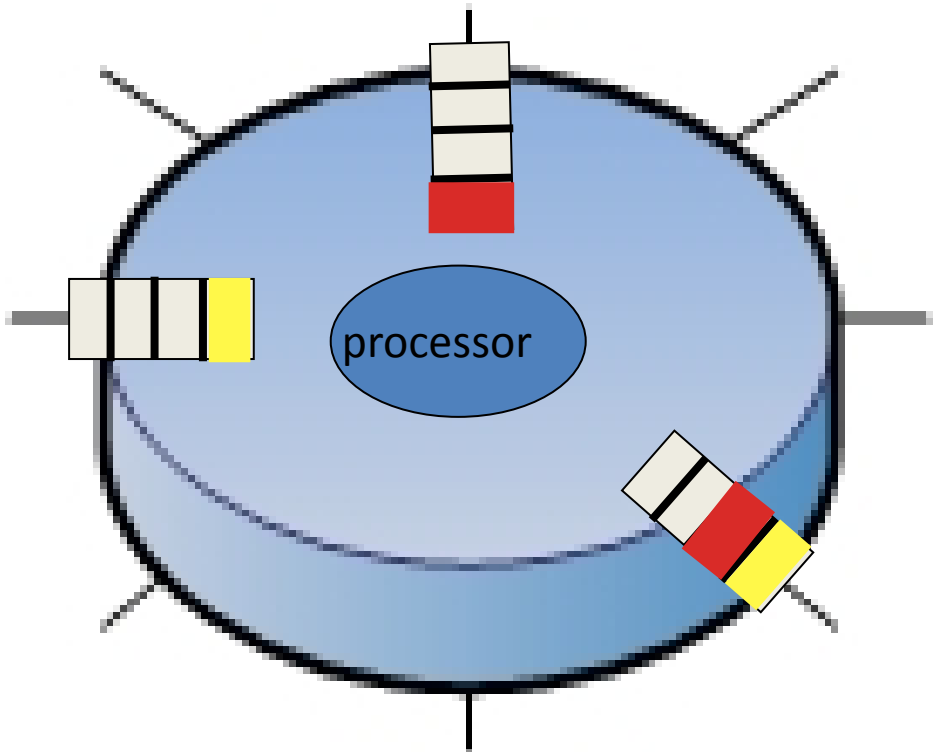
- More formally, little errors don't matter so much in the presence of big errors

Example --- modeling delay through a router or switch

- In large scale networks a packet may transit 10+ switches on average → a lot of network simulation is doing switching

Accelerating packet traffic

- Routing / switch involves multiple queues and various scheduling policies
  - In general case expect at least 2 events / packet
    - On entry, on exit



What really matters?

A model ought to

- Get the functionality right
- Estimate latency through the switch
- Determine whether a packet is dropped

Observe---**applications** operate at a time scale over a hundred times slower than a switch

- Small errors in latency will not be noticed

Observe---applications that use TCP react to **every packet loss**

Observe---TCP throughput depends on average RTT

Conclusion --- we should try to be accurate w.r.t. packet loss, but can accept individual errors in latency in exchange for speed, preserve average RTT

Model Switching Behavior

How can we understand what is going on within a device well enough to model it?

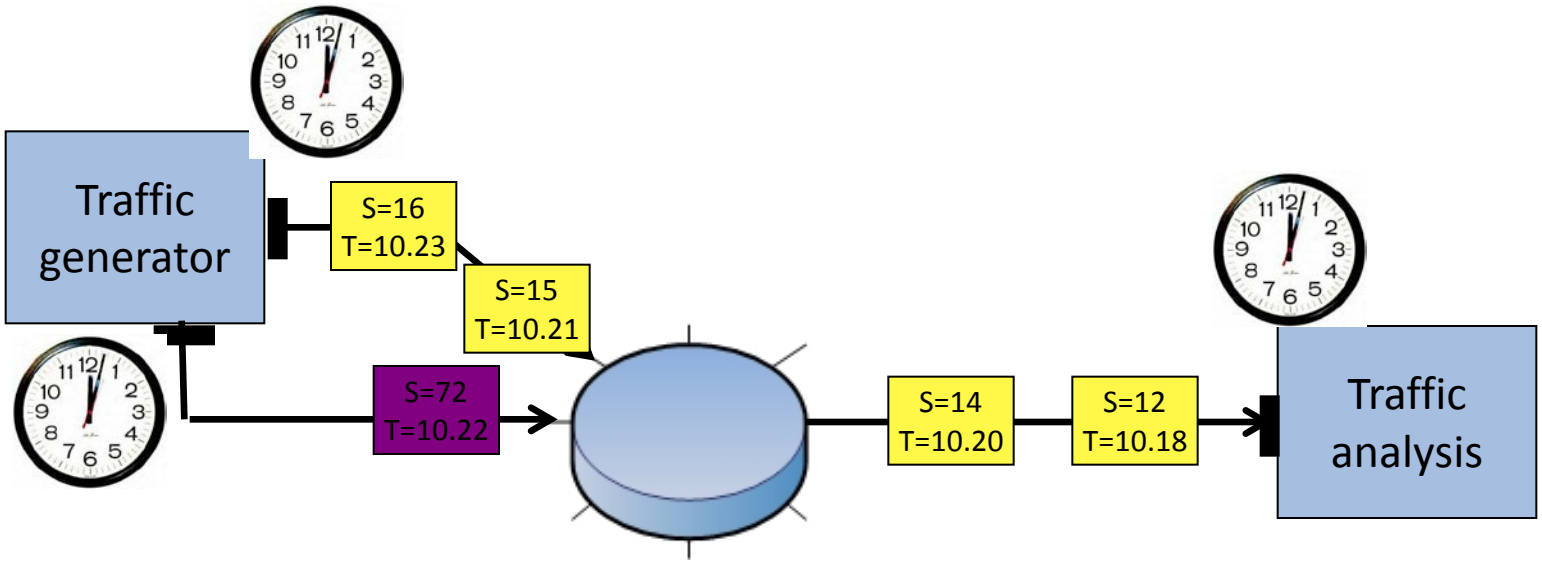
When we do understand it we want to model it with a single event

- ***On arrival*** determine
  - Exgress port
  - Whether packet is dropped
  - What latency through device will be
  - Schedule packet arrival at next device

Measuring latency---fail

Idea

1. Get some COTS switches
2. Shoot tagged traffic at a switch
   - Sequence numbers to detect loss
   - Hack NICs to insert time-stamps on send and receive
   - Use Univ. Wash. gulp software to monitor



Problem: overhead/delay on receipt added x10 factor of noise, unable to run at line rate

Measuring latency---hardware help

NetFPGA card

- programmable to store and transmit 10,000 packets, @1500 bytes
- packets read up from configuration file
  - contain sequence numbers and "release times"

Endace DAG card

- capable of capturing and storing millions of frames at line rate
- time-stamped on arrival with 10ns resolution

Observe : Two clocks require synchronization.

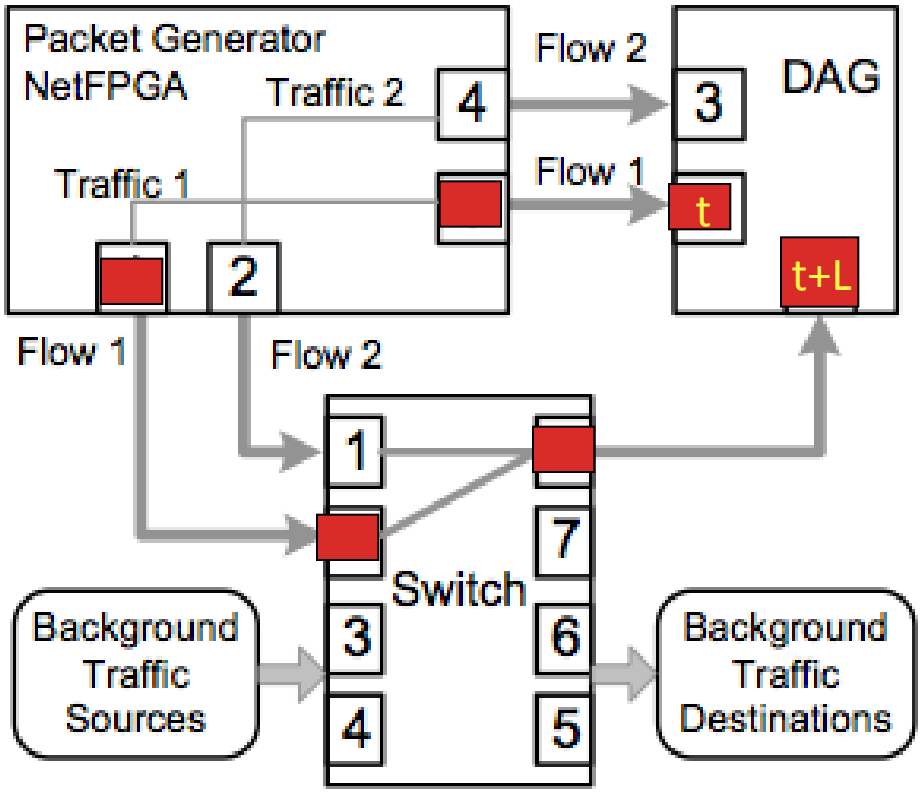
Observe : We have only one clock, at receiver.

??

Testbed Architecture

Key points

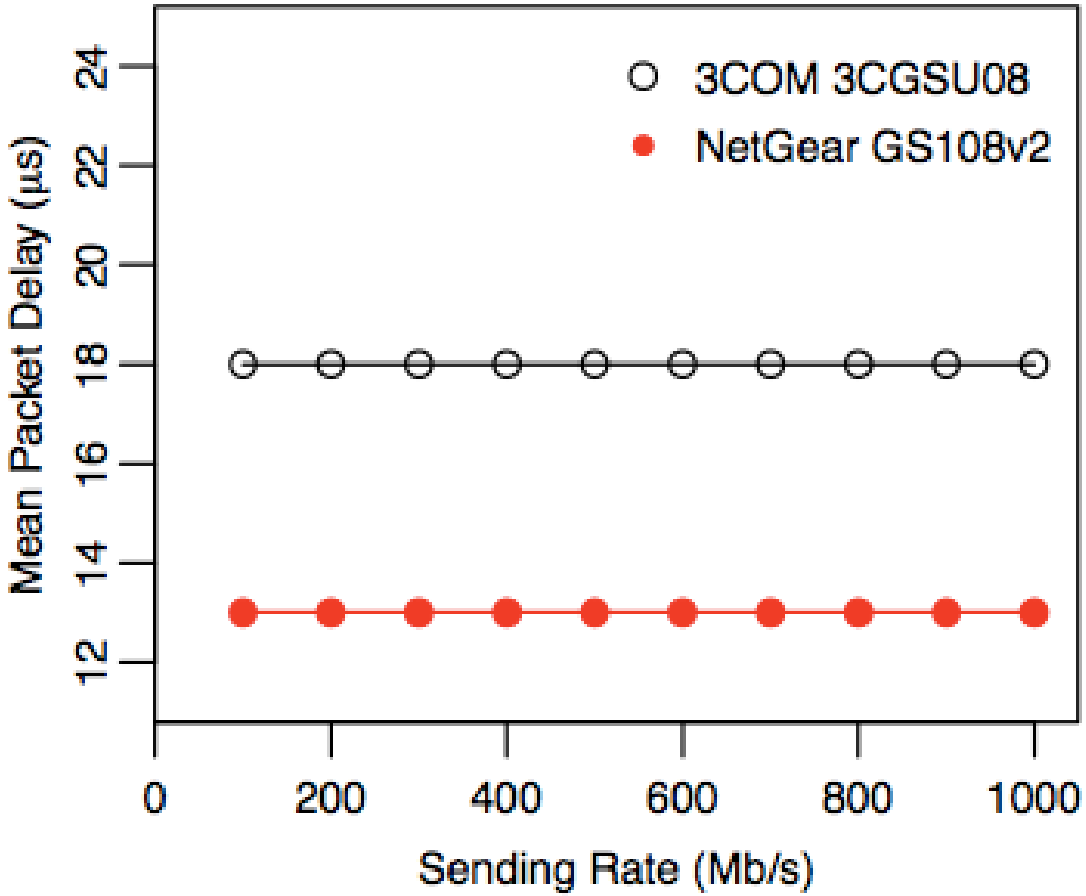
- NetFPGA can send duplicate packets out different ports simultaneously
  - Direct one flow through switch, entangled flow directly to DAG
  - Both are time-stamped, difference is latency!
- Background traffic introduced to create contention



Baseline Performance

Baseline performance == single flow, no overload

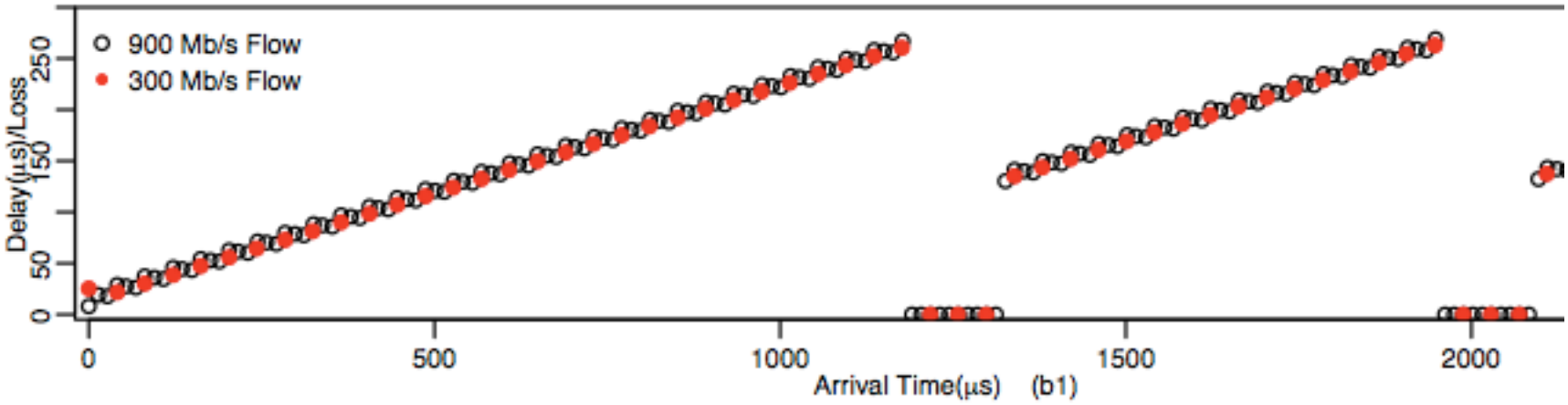
- Two switches evaluated, both carry very close to line speed
- Both flows give constant performance
  - 3COM's is somewhat slower



High Load – NetGear Switch

NetGear GS 108, TrendNet TEGS80G

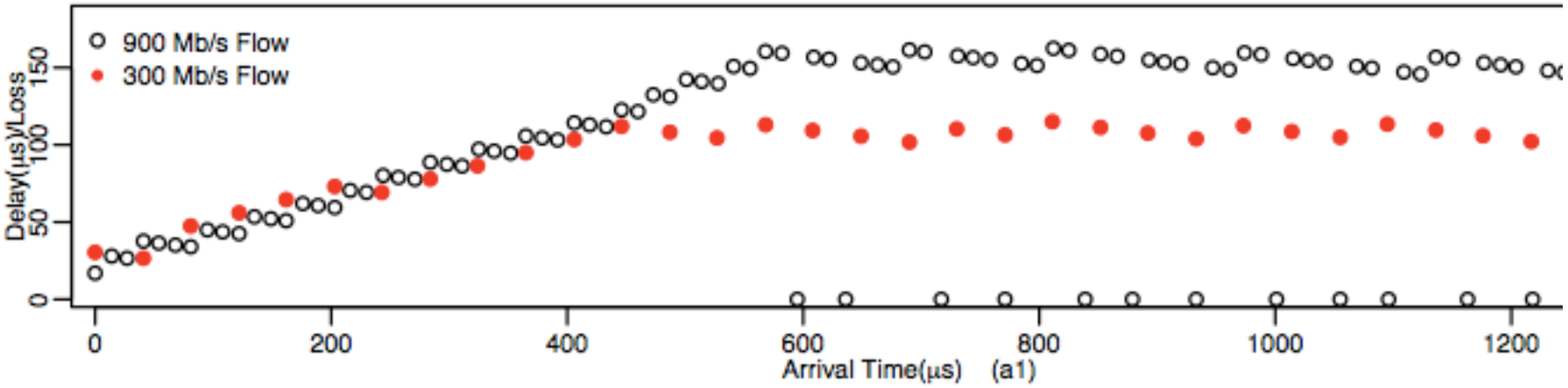
- Instrumented 2 flows, 900 Mbs, and 300 Mbs
- x-axis is packet index, y-axis is delay through switch
- "0" latency codes packet loss---both flows affected uniformly
- Appears queue "drains" to 1/2 length before admitting new traffic
- FCFS explains behavior
  - Implies "1 event passage" already possible



High Load – 3Com Switch

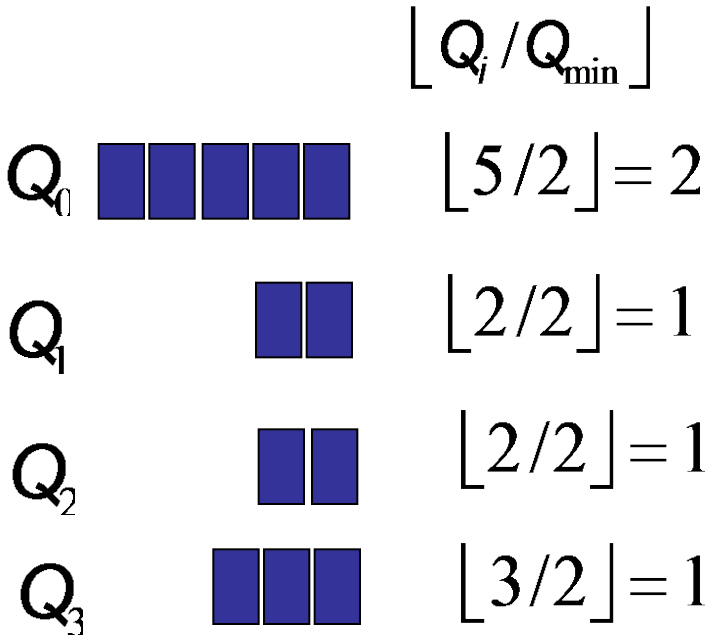
3COM 3CGSU08

- Same two flows
- Only "fast" flow sees loss
- Different flows have different mean latency
- Explainable by "Weighted Round Robin" scheduling



Weighted Round Robin Scheduling

- Schedules created for each "round"
  - Queue Q_i transmits $\lfloor Q_i / Q_{\min} \rfloor$ packets
  - Queues served in order of earliest packet arrival
  - Queue states at round calculation are not known
    - Implies that packet departure time cannot in general be predicted on arrival



Latency-Approximate WRR

Key ideas

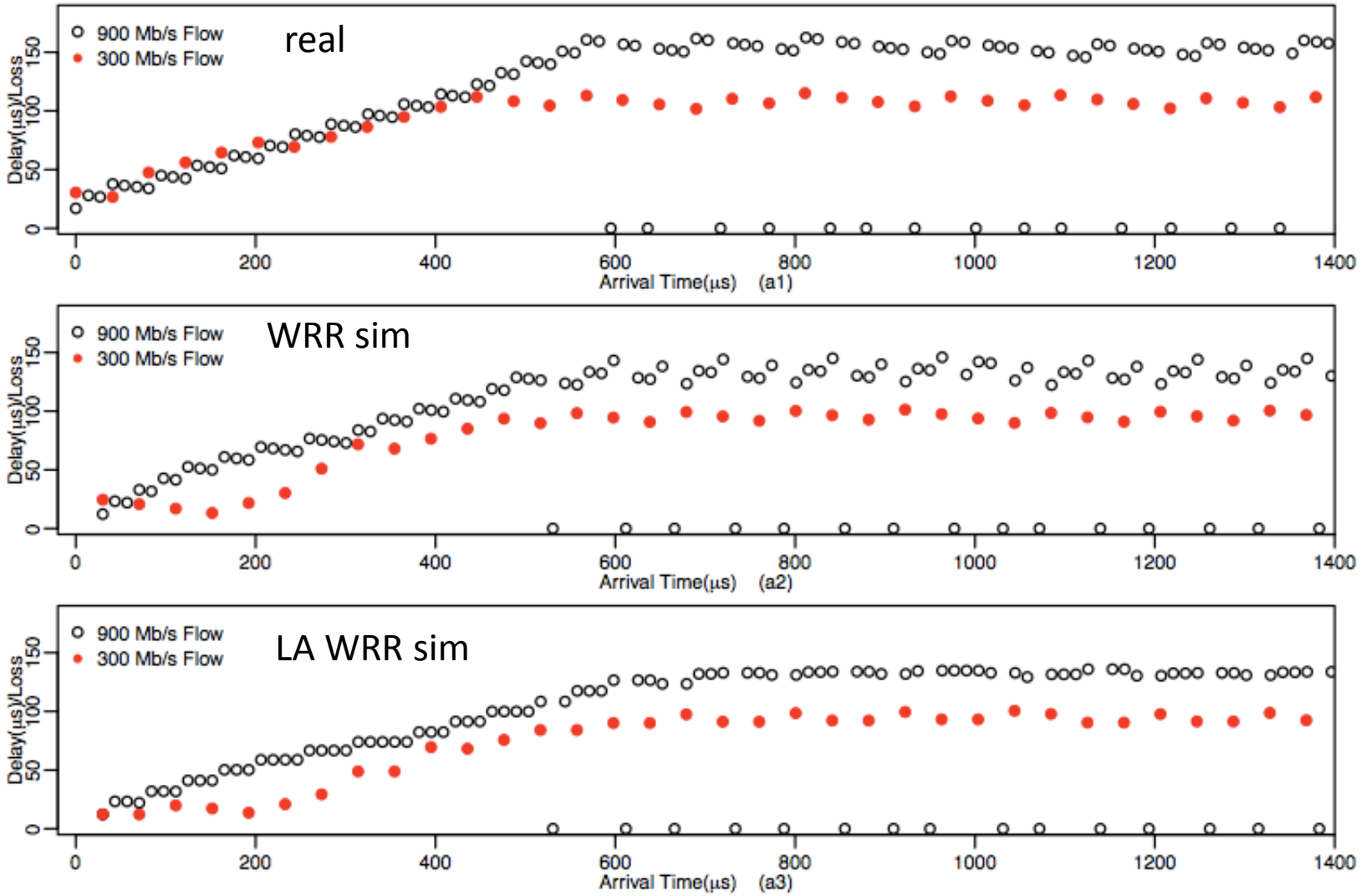
- Compute exactly the queue lengths and packet departures
- When a packet arrives
  - Determine whether lost or not
  - Choose a latency for it, accepting that it is incorrect
  - Forward the packet
- Chosen latency for the packet can be recent observed *real latencies*, or averages of these

Observe---the latencies used to forward packets are just lags on real ones

- one might actually correct them down stream

Validation

Real data, WRR simulation, Latency-Approximate WRR simulation



Performance Study

We can measure performance of simulation *overall*

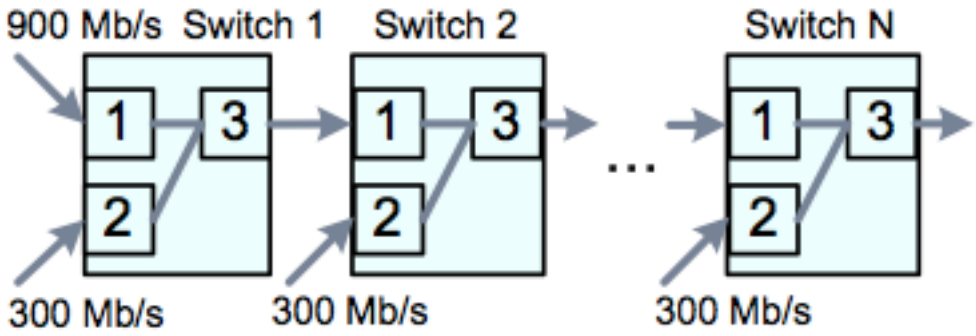
- But want to isolate performance of switch simulation

We want to consider full-scale 2 events / packet simulation (general)

- But recognize that under WRR departures of packets are known at the end of a scheduling period
  - Reduces event *count*, but increases event *complexity*

IDEA : Study performance of 3 models as the number of switches grows

- Not so much realism in network as making switch sim prominent

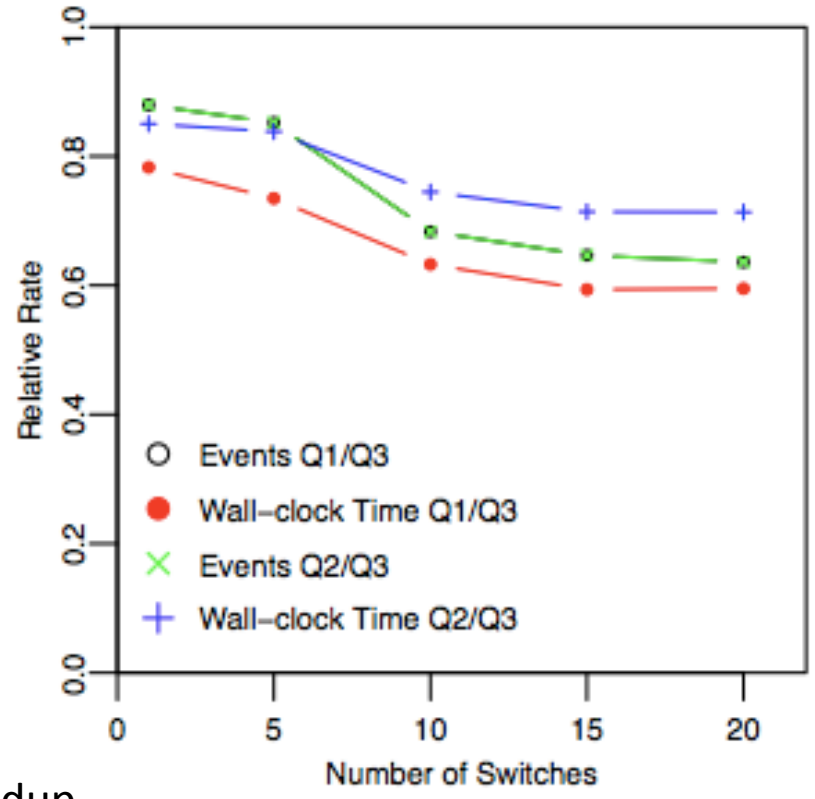


Performance Results

Models :

- Q1 --- latency approximate WRR
- Q2 --- detailed WRR simulator, 1 event per packet arrival, departing packets forwarded at end of scheduling round
- Q3 --- detailed WRR simulator, packet arrival and departure scheduled separately

Model	Events	Exc. Time	Time/Even
Q1	12.7M	32.15s	2.53μs
Q2	12.7M	38.36s	3.04μs
Q3	19.9M	54.04s	2.7 μs



Conclusion : not quite a factor of 2 speedup

But what about {accuracy,validation,science}?

- Various techniques shown look like a bunch of hacks
  - Clever, maybe, but where's the beef?
  - That's a different talk
- Questions we try to answer
  - What are quantifiable relationships between predictions of one model and another that uses these model changes / abstractions?
  - How does uncertainty grow with levels of abstraction? Can it be bounded?
  - Are there general principles for uncertainty containment or error masking?
  - These are all very hard problems! But they are the "modeling science" we need to supports "model engineering"

Done, at last

Take-home points

- Models inherently have abstractions
  - So inherently have deviations from reality
- We can take advantage of the abstractions to accelerate model solution
- Given that models have deviations from reality **anyway**, introducing smaller scale deviations deliberately to accelerate solution may not affect results
- A mathematics of Uncertainty Quantification for discrete systems is needed